# Contents

## 1 Tech Ecosystem
- 1.1 Data sources ................................................. 6
- 1.2 Data governance .............................................. 7
- 1.3 Tools & IDEs ................................................ 8
- 1.4 Packages & libraries ......................................... 9
- 1.5 Spark & Hadoop clusters .................................... 10
- 1.6 App frameworks ............................................. 11
- 1.7 Model publishing ............................................ 11

## 2 Get started
- 2.1 Step 0: Orient yourself to Domino ........................... 14
- 2.2 Step 1: Create a project .................................... 14
- 2.3 Step 2: Configure your project ............................. 15
- 2.4 Step 3: Start a workspace ................................... 18
- 2.5 Step 4: Get your files and data ............................ 20
- 2.6 Step 5: Develop your model .............................. 21
- 2.7 Step 6: Clean up [Workspaces] ............................ 27
- 2.8 Step 7: Deploy your model ............................... 29
- 2.9 Domino training videos .................................. 53

## 3 Get started
- 3.1 Step 0: Orient yourself to Domino (R Tutorial) ............. 56
- 3.2 Step 1: Create a project .................................... 56
- 3.3 Step 2: Configure your project ............................. 57
- 3.4 Step 3: Start a workspace ................................... 60
- 3.5 Step 4: Get your files and data ............................ 62
- 3.6 Step 5: Develop your model .............................. 64
- 3.7 Step 6: Clean up [Workspaces] ............................ 69
- 3.8 Step 7: Deploy your model ............................... 72
- 3.9 Domino training videos .................................. 90

## 4 Get Started with MATLAB
- 4.1 Step 0: Orient yourself to Domino ........................... 92
- 4.2 Step 1: Create a Domino Project ........................... 92
- 4.3 Step 2: Configure Your Domino Project ..................... 93
- 4.4 Step 3: Start a MATLAB Workspace ....................... 96
- 4.5 Step 4: Fetch and Save Your Data .......................... 99
New to Domino?
Watch the video for an introduction and visit try.dominodata.com to start a free trial.

Domino is a data science platform that enables fast, reproducible, and collaborative work on data products like models, dashboards, and data pipelines. Users can run regular jobs, launch interactive notebook sessions, view vital metrics, share work with collaborators, and communicate with their colleagues in the Domino web application.

0. Access resources you need by choosing a custom hardware tier or compute environment.

1. Get data into Domino with a web upload, using the command line interface, by connecting to external data sources and Git repositories, or with the Domino API.

2. Prepare data in an interactive workspace using your preferred tools.

3. Develop your model by running Jobs, and create repeatable experiment pipelines with Scheduled Jobs.

4. Deploy your model with a Model API, web application, or Launcher.

5. Share and collaborate by creating an organization or searching prior work.
Browse or search for tutorials, guides, and reference materials.
Domino is an open enterprise platform for data science, machine learning, and AI research.

Domino works with an expansive list of industry leading tools and technologies to enrich the data science research, development, and deployment process. Our product works with a wide range of data sources, languages, IDEs, tools, libraries, and publication targets, including:

- Certified partners who have worked with Domino to integrate and verify their tools
- Other 3rd party tools and technologies known to work with Domino
- Access to other tools and technologies through code-first APIs or connections

The following topics in our documentation describe these integrations in more detail. The integrations are listed alphabetically and grouped according to the categories shown in the diagram.

If you see something missing, please reach out to Domino Support as we are always adding new integrations and always interested in hearing what’s top of mind for data scientists.
The **Center** of the ML Ecosystem

- **Connect to your data sources**
  - snowflake
  - PurgeSQL
  - S3
  - OKERA
- **Develop in your programming language**
  - python
  - R
  - MATLAB
  - Julia
- **Use your tool or IDE**
  - jupyter
  - Studio
  - Visual Studio Code
  - Superset
- **Build and host apps with your app framework**
  - matplotlib
  - plotly
  - django
  - Shiny
  - R
  - DASK
  - Databricks
  - MAPR
- **Install your libraries or packages**
  - Anaconda
  - PyTorch
  - H2O.ai
  - TensorFlow
  - DataRobot
- **Send jobs to your distributed compute cluster**
- **Run on your infrastructure**
  - NVIDIA
  - AWS
  - Kubernetes
  - Microsoft Azure
## 1.1 Data sources

<table>
<thead>
<tr>
<th>Solution</th>
<th>Integration information</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="snowflake.png" alt="Snowflake" /></td>
<td>• Connecting to Snowflake from Domino</td>
</tr>
<tr>
<td><img src="s3.png" alt="S3" /></td>
<td>• Connecting to S3 from Domino</td>
</tr>
<tr>
<td><img src="redshift.png" alt="Amazon Redshift" /></td>
<td>• Connecting to Redshift from Domino</td>
</tr>
<tr>
<td><img src="postgresql.png" alt="PostgreSQL" /></td>
<td>• Connecting to PostgreSQL from Domino</td>
</tr>
<tr>
<td><img src="sqlserver.png" alt="Microsoft SQL Server" /></td>
<td>• Connecting to Microsoft SQL Server from Domino</td>
</tr>
<tr>
<td><img src="impala.png" alt="Impala" /></td>
<td>• Connecting to Impala from Domino</td>
</tr>
<tr>
<td><img src="oracle.png" alt="Oracle Database" /></td>
<td>• Connecting to Oracle DB from Domino</td>
</tr>
<tr>
<td><img src="mysql.png" alt="MySQL" /></td>
<td>• Connecting to MySQL from Domino</td>
</tr>
</tbody>
</table>
1.2 Data governance

<table>
<thead>
<tr>
<th>Solution</th>
<th>Partner</th>
<th>Integration information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OKERA</strong></td>
<td></td>
<td><strong>Connecting to Okera from Domino</strong></td>
</tr>
</tbody>
</table>
1.3 Tools & IDEs

<table>
<thead>
<tr>
<th>Solution</th>
<th>Part Integration information</th>
</tr>
</thead>
<tbody>
<tr>
<td>RStudio</td>
<td>• RStudio comes standard in the Domino Analytics Distribution</td>
</tr>
<tr>
<td>Jupyter</td>
<td>• Jupyter and JupyterLab come standard in the Domino Analytics Distribution</td>
</tr>
<tr>
<td>MATLAB®</td>
<td>• Domino distributes a base MATLAB environment image which you can add as a Workspace</td>
</tr>
<tr>
<td>SAS®</td>
<td>• Use your SAS Viya Data Science Studio in Domino. Learn how to create a SAS Workspace Environment</td>
</tr>
<tr>
<td>Visual Studio Code</td>
<td>• Using Visual Studio Code in Domino</td>
</tr>
<tr>
<td>Apache Zeppelin</td>
<td>• How to set up Zeppelin Workspaces in Domino</td>
</tr>
</tbody>
</table>
## 1.4 Packages & libraries

<table>
<thead>
<tr>
<th>Solution</th>
<th>Integration information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Miniconda</td>
<td>• Miniconda comes standard in the <em>Domino Analytics Distribution</em> with a variety of common packages. You can also point to a local mirror.</td>
</tr>
<tr>
<td>DataRobot</td>
<td>• Connecting to DataRobot in Domino</td>
</tr>
</tbody>
</table>


### 1.5 Spark & Hadoop clusters

<table>
<thead>
<tr>
<th>Solution</th>
<th>Integration information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloudera</td>
<td>Connecting to a Cloudera CDH5 cluster from Domino</td>
</tr>
<tr>
<td>Hortonworks</td>
<td>Connecting to a Hortonworks cluster from Domino</td>
</tr>
<tr>
<td>MapR</td>
<td>Connecting to a MapR cluster from Domino</td>
</tr>
<tr>
<td>Amazon EMR</td>
<td>Connecting to an Amazon EMR cluster from Domino</td>
</tr>
</tbody>
</table>
## 1.6 App frameworks

<table>
<thead>
<tr>
<th>Solution</th>
<th>Partner integration information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Shiny</strong></td>
<td>• Getting started with Shiny in Domino</td>
</tr>
<tr>
<td><strong>Dash</strong></td>
<td>• Getting started with Dash in Domino</td>
</tr>
<tr>
<td><strong>Flask</strong></td>
<td>• Getting started with Flask in Domino</td>
</tr>
<tr>
<td><strong>django</strong></td>
<td>• Getting started with Django in Domino</td>
</tr>
</tbody>
</table>

## 1.7 Model publishing

<table>
<thead>
<tr>
<th>Solution</th>
<th>Partner integration information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amazon SageMaker</strong></td>
<td>• With Domino you can export a model endpoint which is compatible with AWS Sagemaker. Find instructions for publishing <a href="#">here</a>.</td>
</tr>
</tbody>
</table>

If you are a partner interested in certifying your solution in Domino, please contact partners@dominodatalab.com
This tutorial will guide you through a common model lifecycle in Domino. You will start by working with data from the Balancing Mechanism Reporting Service in the UK. We will be exploring the Electricity Generation by Fuel Type and predicting the electricity generation in the future. You'll see examples of Jupyter, Dash, pandas, and Prophet used in Domino.

The following content is meant to be followed in sequence.
2.1 Step 0: Orient yourself to Domino

When you first log in, you will find yourself in the Lab section of Domino on the Projects Overview page. You can use the left side bar to navigate to other areas of Domino.

If you would like to search the Domino documentation for help, click on the Help icon on the bottom left of the page. To send a question to a member of the Domino support staff, use the Support button on the bottom right of the page.

2.2 Step 1: Create a project

Work in Domino happens in projects. Projects contain data, code, and environment settings, and the entire project is tracked and revisioned automatically. A new commit is written to a project each time its files are changed by user action, or by the execution of code in the project. Users in Domino can create their own new projects, invite other users to collaborate on them, and export data or results for consumption by other projects.

To create a project:

1. In the left side bar, click on

2. Click on on the right side of the page.
2.1. Give your project an informative name (like power-generation)
2.2. Set its Visibility to Private.
2.3. Click Create Project.

2.3 Step 2: Configure your project

- Introduction
- Step 2.1: Select your hardware tier
- Step 2.2: Configure your environment
- Step 2.3: Configure the project permissions
2.3.1 Introduction

Every project has its own settings. The following options are important to consider when configuring a new project:

- Hardware Tier
- Environment
- Collaborators

2.3.2 Step 2.1: Select your hardware tier

A Hardware Tier represents the compute resources that will be available for your run. You can specify memory, CPU cores, and GPUs with hardware tiers.

The hardware tier dropdown menu lists your available options. The selected hardware tier will be used by default for all subsequent executions of code in the project. It can also be changed at any point in the future.

1. In the Project menu, click Settings.

2. Click on the Hardware tier dropdown menu to choose the compute resource to execute your code on. Your options may look different from the image below.

2.1. Choose the smallest or default hardware tier for this tutorial.

This list of available hardware tiers is customizable by your Domino administrators. If you would like additional resources, contact your Domino administrator.
2.3.3 Step 2.2: Configure your environment

An Environment is a Domino abstraction on top of a Docker image that provides additional flexibility and versioning. You can configure the software, packages, libraries, and drivers that you need in your environment.

Domino comes with a default environment called the Domino Analytics Distribution, which includes Python, R, Jupyter, RStudio, and hundreds of data science related packages and libraries.

We’ll choose the Python 3 version.

1. Click on the Compute Environment dropdown menu to choose the Environment.

Your compute environment dropdown will likely have different options. If you’re interested in learning how to add more packages and customize or create your own Environment, see the help article on Domino Environments.

2.3.4 Step 2.3: Configure the project permissions

As the owner of the project, you can set different access levels for collaborators and colleagues. Feel free to invite a colleague to be a Contributor to your project.

2.3. Step 2: Configure your project
1. Click on the **Access & Sharing** tab at the top of the page

![Collaborators and permissions](image)

2. (Optional) Enter the email or the username of the user that you would like to invite
3. (Optional) Enter a welcome message to be sent to your collaborator

The **Contributor** role allows the invited user to read, write, and execute code in this project.

Visit the **Collaborators and permissions** support article for more information on the permissions for each collaborator role.

### 2.4 Step 3: Start a workspace

Workspace sessions are interactive sessions hosted by a Domino executor where you can interact with code notebooks like Jupyter and RStudio. The software tools and associated configurations available in your session are called Workspaces.

For this tutorial, we will start a Jupyter Workspace.
1. Click on **Workspaces** from the project menu.

2. Select **Jupyter**.

3. Click **Launch Now**.

When you launch a workspace, a new containerized session is created on a machine (also known as an **executor**) in the required hardware tier. The workspace tool you requested is launched in that container, and your browser is automatically redirected to the workspace’s UI when it’s ready.
Once your workspace is up and running, you will see a fresh Jupyter interface. If you’re brand new to Jupyter, you might find the Jupyter and Jupyterlab documentation helpful.

If you are interested in adding additional Workspaces for tools that are available by default, see the pluggable notebooks section of your Domino environment documentation.

### 2.5 Step 4: Get your files and data

There are basically two strategies to working with data in Domino:

- **You can copy your data into Domino**
  
  If you are working with data that is on your local machine or in a shared server, you might want to upload your data into Domino.

- **You can query your data from Domino**
  
  If you have a large dataset stored in a database or data service, you may just need to query the database or the API for the data service.

In this step, you’ll copy your data into the project using the Jupyter terminal. If you have not done so, please complete Step 3 to start a Jupyter workspace.

The starting file path in your Jupyter workspace is `/mnt`. By default, this is considered the root of your Domino project. If you add or modify files in `/mnt`, you can save them back to your project when you stop or sync the workspace.

1. Use the New menu to open a Jupyter terminal.

2. In the new terminal, run the following command to fetch some data from the BMRS:
3. Type an informative commit message in the textbox above the **Sync All Changes** button. Now, click the **Sync All Changes** button in the toolbar to perform a sync and save the new `data.csv` file to your Domino project.

Return to your project in Domino by navigating to a new browser tab. Then, navigate to the “Files” section of Domino. Notice that the raw data has been saved in the latest revision.

See the documentation on other methods of both *copying data into Domino* and *querying data from Domino*.

### 2.6 Step 5: Develop your model
2.6.1 Introduction

When you are developing your model, you want to be able to quickly execute code, see outputs, and make iterative improvements. Domino enables this with Workspaces. Step 3 covered starting a Workspace and explored Workspace options like VSCode, RStudio, and Jupyter.

In this section, we will use Jupyter to load, explore, and transform some data. After the data has been prepared, we will train a model.

2.6.2 Step 5.1: Load and explore the dataset

1. Click the Jupyter logo at the top to return to the files browser within the Jupyter Workspace. You should see data.csv in /mnt. If not, please return to Step 4 to download the dataset.

2. Use the New menu to create a Python notebook.
3. In the first cell, enter these lines to import some packages, then hit Shift+Enter to execute:

```python
%matplotlib inline
import pandas as pd
import datetime
```

4. Next, read the file you downloaded into a pandas dataframe:

```python
df = pd.read_csv('data.csv', skiprows=1, skipfooter=1, header=None, engine='python')
```

5. Rename the columns according to information on the column headers at https://www.bmreports.com/bmrs/?q=generation/fueltype/current and display the first five rows of the dataset using `df.head()`.

```python
df.columns = ['HDF', 'date', 'half_hour_increment', 'CCGT', 'OIL', 'COAL', 'NUCLEAR', 'WIND', 'PS', 'NPSHYD', 'OCGT', 'OTHER', 'INTFR', 'INTIRL', 'INTNED', 'INTEW', 'BIOMASS', 'INTEM', 'INTEL', 'INTIFA2', 'INTNSL']
df.head()
```

We can see that this is a time series dataset. Each row is a successive half hour increment during the day that details the amount of energy generated by fuel type. Time is specified by the `date` and `half_hour_increment` columns.

6. Create a new column named `datetime` that represents the starting datetime of the measured increment. For
example, a 20190930 date and 2 half hour increment means that the time period specified is September 19, 2019 from 12:30am to 12:59am.

def['datetime'] = pd.to_datetime(df['date'], format='%Y%m%d')
def['datetime'] = df.apply(lambda x:x['datetime'] + datetime.timedelta(minutes=30*(int(x['half_hour_increment'])-1)), axis = 1)

7. Visualize the data to see how each fuel type is used during the day by plotting the data.

def.drop(
    ['HDF', 'date', 'half_hour_increment'], axis = 1
).set_index('datetime').plot(figsize=(15,8))

The CCGT column representing “combined-cycle gas turbines” seems to be the most interesting. It generates a lot of energy and is very volatile.

We will concentrate on this column and try to predict the power generation from this fuel source.

2.6.3 Step 5.2: Train a model

Data scientists have access to many libraries and packages that help with model development. Some of the most common for Python are XGBoost, Keras, and scikit-learn. These packages are already installed in the Domino Analytics Distribution, the default environment. However, there may be times that you want to experiment with a package that is new and not installed in the environment.
We will build a model with the Facebook Prophet package, which is not installed into the default environment. You will see that you can quickly get started with new packages and algorithms just as fast as they are released into the open source community.

1. In the next Jupyter cell, install Facebook Prophet, which also requires PyStan, a slightly older version of Plotly (to be compatible with Prophet), and Cufflinks. Note that PyStan requires 4 GB of RAM to be installed. Please make sure your workspace is set to use a large enough hardware tier:

```bash
!sudo pip install cufflinks==0.16.0
!sudo pip install "pystan==2.17.1.0" "plotly<4.0.0"
!sudo pip install "fbprophet==0.6"
```

2. For Facebook Prophet, the time series data needs to be in a DataFrame with 2 columns named `ds` and `y`:

```python
df_for_prophet = df[['datetime', 'CCGT']].rename(columns = {'datetime':'ds', 'CCGT':'y'})
```

3. Split the dataset into train and test sets:

```python
X = df_for_prophet.copy()
y = df_for_prophet['y']
proportion_in_training = 0.8
split_index = int(proportion_in_training*len(y))
X_train, y_train = X.iloc[:split_index], y.iloc[:split_index]
X_test, y_test = X.iloc[split_index:], y.iloc[split_index:]
```

4. Import Facebook Prophet and fit a model:

```python
from fbprophet import Prophet
m = Prophet()
m.fit(X_train)
```

If you encounter an error when running this cell, you may need to downgrade your pandas version first. In that case you would run

```bash
!sudo pip install pandas==0.23.4
```

and then

```python
from fbprophet import Prophet
m = Prophet()
m.fit(X_train)
```

After running the code above, you may encounter a warning about code deprecation. The warning can be ignored for the purposes of this walkthrough.

5. Make a DataFrame to hold prediction and predict future values of CCGT power generation:

```python
future = m.make_future_dataframe(periods=int(len(y_test)/2), freq='H')
forecast = m.predict(future)
# forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail() #uncomment to inspect the DataFrame
```

6. Plot the fitted line with the training and test data:

```python
import matplotlib.pyplot as plt
plt.gcf()
fig = m.plot(forecast)
```

(continues on next page)
7. Rename the notebook to be `Forecast_Power_Generation`

8. Save the notebook.
2.6.4 Step 5.3: Export the model

Trained models are meant to be used. There is no reason to re-train the model each time you use the model. Export or serialize the model to a file to load and reuse the model later. In Python, the pickle module implements protocols for serializing and de-serializing objects. In R, you can commonly use the serialize command to create RDS files.

1. Export the trained model as a pickle file for later use:

   ```python
   import pickle
   # m.stan_backend.logger = None  # uncomment if using Python 3.6 and
   with open("model.pkl", "wb") as f:
       pickle.dump(m, f)
   ``

We will use the serialized model in Step 7 when we create an API from the model.

2.7 Step 6: Clean up [Workspaces]

- Introduction
  - Option 1: Stop your Workspace session from inside the workspace
  - Option 2: Stop your Workspace session from the Workspaces page

2.7.1 Introduction

To avoid spending unnecessary compute resources, make sure to stop any Workspace sessions that you started as a part of this tutorial. If your Domino is deployed in the cloud, this will prevent you from incurring unnecessary charges. If your Domino is deployed on premises, this will free up your compute resources for others to use.

There are two places in the Domino UI where you can stop your sessions.

Option 1: Stop your Workspace session from inside the workspace

1. Above your Jupyter notebook in the blue menu bar, click Stop.
2. Enter a descriptive commit message in the text box above Sync all Changes. Now click on Sync All Changes.

Option 2: Stop your Workspace session from the Workspaces page

1. Go to the Workspaces page in Domino. Since the Jupyter workspace opened in a new tab, you may have to select the previous Domino tab.

2. Click on the Stop button corresponding to your workspace session.

3. Click Stop My Workspace.
In both options, you were able to select Stop and Commit. The Stop and Commit button stops the workspace session and also saves your work back to your project. The new files created from Step 5 should now be visible on the Files page.

### 2.8 Step 7: Deploy your model

- **Introduction**
- **Package setup**
- **Scheduled reports**
- **Launchers**
- **Model APIs**
2.8.1 Introduction

Once you have developed your model and deemed it good enough to be useful, you will want to deploy it. There is no single deployment method that is best for all models. Therefore, Domino offers four different deployment options. One may fit your needs better than the others depending on your use case.

The available deployment methods are:

- Scheduled reports
- Launchers
- Web applications
- Model APIs

The remaining sections of this tutorial are not dependent on each other. For example, you will not need to complete the Scheduled report section to understand and complete the Web application section.

2.8.2 Package setup

A pre-requisite to the following sections will be to install a few packages. We will do this expediently by creating a requirements.txt file in the project, which will install the Python packages listed in the file prior to every job or workspace session.

1. Go to the Files page of your project.
2. Click the Add File button.
3. Name it `requirements.txt`, copy and paste the following contents, and Save:

```python
convertdate
pyqt5<5.12
jupyter-client>6.0.0
nbformat>5.0
papermill<2.0.0
pystan==2.17.1.0
plotly<4.0.0
dash
requests
nbconvert >= 5.4
```

If you want to install these libraries permanently into a custom environment, find out more in the `Model API` tutorial.

### 2.8.3 Scheduled reports

The *Scheduled Jobs* feature in Domino allows you to run a script on a regular basis. In Domino, you can also schedule a notebook to run from top to bottom and export the resulting notebook as an HTML file. Since notebooks can be formatted with plain text and embedded graphics, you can use the scheduling feature to create regularly scheduled, automated reports for your stakeholders.

In our case, we can imagine that each day we receive new data on power usage. To make sure our predictions are as accurate as possible, we can schedule our notebook to re-train our model with the latest data and update the visualization accordingly.

1. *Start* a new Jupyter session.
2. Create a copy and open the Jupyter notebook you created in *Step 5*.
3. Add some dynamically generated text to the upcoming report. We want to pull the last 30 days of data.

3.1. Insert a new cell above the first cell by selecting the first cell and selecting **Insert Cell Above**.

3.2. Copy and paste the following code into the new cell:

```python
import datetime
today = datetime.datetime.today().strftime('%Y-%m-%d')
one_month = (datetime.datetime.today() - datetime.timedelta(30)).strftime('%Y-%m-%d')
!curl -o data.csv "https://www.bmreports.com/bmrs/?q=ajax/filter_csv_download/FUELHH/csv/FromDate%3D{one_month}%26ToDate%3D{today}/&filename=GenerationbyFuelType_20191002_1657" 2>/dev/null
```

4. Since this is a report, you will want to add some commentary to guide the reader. For this exercise, we will just add a header to the report at the top. To add a Markdown cell:

4.1. Insert a new cell above the first cell again by selecting the first cell and selecting **Insert Cell Above**.

4.2. Change the cell type to Markdown.
4.3. Enter the following in the new Markdown cell:

```markdown
# New Predictions for Combined Cycle Gas Turbine Generations
```

5. Save the notebook.

6. Sync All Changes in the workspace session.

7. Test the notebook.

7.1. Go the Files page.

7.2. Click on the link for the new copy of the notebook.

7.3. Click the Run button in the top right of the page.

7.4. Click **Start** on the modal.

7.5. Wait for the run to complete. While running, the Status icon will appear blue and logs will stream on the right side of the page.

7.6. Once the job has completed successfully, you'll see the Status icon turn green and be able to browse the Results tab.

2.8. Step 7: Deploy your model
8. At this point, you can schedule the notebook to run every day. Go to the Scheduled Jobs page.

9. Start a new scheduled job and enter the name of the file that you want to schedule to run. This will be the name of your Jupyter notebook.

10. Select how often and when to run the file.

11. Enter emails of people to send the resulting file(s) to.
12. Click Schedule.

To discover more tips on how to customize the resulting email, see Custom Notifications for more information.

2.8.4 Launchers

Launchers are web forms that allow users to run templatized scripts. They are especially useful if your script has command line arguments that dynamically change the way the script executes. For heavily customized scripts, those command line arguments can quickly get complicated. Launcher allows you to expose all of that as a simple web form.

Typically, we parameterize script files (i.e. files that end in .py, .R, or .sh). Since we have been working with Jupyter notebooks until now, we will parameterize a copy of the Jupyter notebook that we created in Step 5.

To do so, we will insert a few new lines of code into a copy of the Jupyter notebook, create a wrapper file to execute, and configure a Launcher.

1. Parameterize the notebook with a Papermill tag and a few edits:

   1.1. Start a Jupyter session. Make sure you are using a Jupyter workspace, not a Jupyterlab workspace. We recently added the requirements.txt file, so the session will take longer to start.
1.2. Create a copy of the notebook that you created in Step 5. Rename it
Forecast_Power_Generation_for_Launcher.

1.3. In the Jupyter menu bar, select View/Cell Toolbar/Tags.

1.4. Create a new cell at the top of the notebook and enter the following into the cell

```python
!pip install fbprophet==0.6
```

1.5. Create another new cell.

1.6. Add a parameters tag to the top cell.

1.7. Enter the following into the cell to create default parameters:

```python
start_date_str = 'Tue Oct 06 2020 00:00:00 GMT-0700 (Pacific
˓→Daylight Time)'
fuel_type = 'CCGT'
```

1.8. Insert another cell below.

1.9. Launcher parameters get passed to the notebook as strings. The notebook will need the date parameters to be in a differently formatted string.

```python
import datetime
today = datetime.datetime.today().strftime('%Y-%m-%d')
start_date = datetime.datetime.strptime(start_date_str.split(' (')[0], '%a %b %d %Y 00:00:00 GMT%z').strftime('%Y-%m-%d')
```

1.10. Insert another new cell below with the following code:

```bash
!curl -o data.csv "https://www.bmreports.com/bmrsl/?q=ajax/˓→filter_csv_download/FUELHH/csv/FromDate%3D{start_date}˓→%26ToDate%3D{today}/&filename=GenerationbyFuelType_˓→20191002_1657" 2>/dev/null
```

The top of your notebook should look like this:

1.11. In the cell where df_for_prophet is defined, replace 'CCGT' with fuel_type:

```python
df_for_prophet = df[['datetime', fuel_type]].rename(columns = {
˓→'datetime':'ds', fuel_type:'y'})
```
1.12. Save the notebook.

1.13. Stop and Commit the workspace session.

2. Create a wrapper file to execute.

   2.1. Navigate back to the Files page.

   2.2. Create a new file called `forecast_launcher.sh`.

   2.3. Copy and paste the following code for the file and save it:

   ```bash
   papermill Forecast_Power_Generation_for_Launcher.ipynb
   → forecast.ipynb -p start_date "$1" -p fuel_type $2
   ```

   The command breaks down as follows:

   ```bash
   papermill <input ipynb file> <output ipynb file> -p
   <parameter name> <parameter value>
   ```

   We will pass in our values as command line arguments to the shell script `forecast_launcher.sh`, which is why we have $1 and $2 as our parameter values.

3. Configure the Launcher.

   3.1. Navigate to the Launcher page, found under the Publish menu on the left side of the screen.
3.2. Click New Launcher.

3.3. Name the launcher “Power Generation Forecast Trainer”

3.4. Copy and paste the following into the field “Command to run”:

```bash
forecast_launcher.sh ${start_date} ${fuel_type}
```

You should see the parameters show up below:

3.5. Select the `start_date` parameter and change the type to `Date`.

3.6. Select the `fuel_type` parameter and change the type to `Select (Drop-down menu)`

3.6.1. Copy and paste the following into the “Allowed Values” field:
3.7. Click Save Launcher

4. Try out the Launcher.
   4.1. Navigate back to the main Launcher page.
   4.2. Click Run for the “Power Generation Forecast Trainer” launcher.
   4.3. Select a start date for the training data.
   4.4. Select a fuel type from the dropdown.
   4.5. Click Run

This will execute the parameterized notebook with the parameters that you selected. In this particular launcher, a new dataset was downloaded and the model was re-trained. Graphs in the resulting notebook represent the new dataset. You can see them in the Results tab.

When the run has been completed, an email will be sent to you and others that you optionally specified in the launcher with the resulting files. If you would like to customize the email, see Custom Notifications for more information.

2.8.5 Model APIs

If you want your model to serve another application, you will want to serve it in the form of an API endpoint. Domino Model APIs are scalable REST APIs that can create an endpoint from any function in a Python or R script. The Domino Model APIs are commonly used when you need an API to query your model in near real-time.
For example, we created a model to forecast power generation of combined cycle gas turbines in the UK.

In this section, we will deploy an API that uses the model that we trained in Step 5 to predict the generated power given a date in the future. To do so, we will create a new *compute environment* to install necessary packages, create a new file with the function we want to expose as an API, and finally deploy the API.

1. Create a new compute environment.

   1.1. Navigate to the Environments page in Domino.

   1.2. Click Create Environment.

   1.3. Name the environment and enter a description for the new environment.
1.4. Click Create Environment.

1.5. Click Edit Definition in the top right corner of the page.

1.6. In the Dockerfile Instructions section, enter the following:

```bash
RUN pip install "pystan==2.17.1.0" "plotly<4.0.0" "papermill<2.0.0" "requests dash" && pip install fbprophet==0.6
```
1.7. Scroll to the bottom of the page and click Build.

This will start the creation of your new compute environment. These added packages will now be permanently installed into your environment and be ready whenever you start a job or workspace session with this environment selected. Note that PyStan needs 4 GB of RAM to install, please reach out to your admin if you see errors so they can ensure that builds have the appropriate memory allocation.

1.8. Navigate back to your project page and navigate to the Settings page.

1.9. Select your newly created environment from the Compute Environments dropdown menu.

2. Create a new file with the function we want to expose as an API

2.1. From the Files page of your project, click Add File.
2.2. Name your file `forecast_predictor.py`.

2.3. Enter the following contents:

```python
import pickle
import datetime
import pandas as pd

with open('model.pkl', 'rb') as f:
    m = pickle.load(f)

def predict(year, month, day):
    """
    Input:
    year - integer
    month - integer
    day - integer
    
    Output:
    predicted generation in MW
    """
    ds = pd.DataFrame({'ds': [datetime.datetime(year, month, day)]})
    return m.predict(ds)['yhat'].values[0]
```

2.4. Click Save.

3. Deploy the API.

3.1. Navigate to the Publish/Model APIs page in your project.

3.2. Click New Model.
3.3. Name your model, provide a description, and click Next.

**New Model**

- **Model Name**: Power Generation Predictor for CCGT
- **Description (optional)**: API for predicting the power generation by combined cycle gas turbines in UK.
  - Input: month, day, year
  - Output: float (MW)

3.4. Enter the name of the file that you created in the previous step.
3.5. Enter the name of the function that you want to expose as an API.
3.6. Click Create Model.
4. Test the API.

4.1 Wait for the Model API status to turn to Running. This may take a few minutes.

4.2. Click the Overview tab.

4.3. Enter the following into the Request box in the tester:

```json
{
    "data": {
       "year": 2019,
       "month": 10,
       "day": 15
    }
}
```

4.4. Click Send. If successful, you will see the response on the right panel.
As a REST API, any other common programming language will be able to call it. Code snippets from some popular languages are listed in the other tabs.

Model APIs are built as docker images and deployed on Domino. You can export the model images to your external container registry and deploy them in any other hosting environment outside of Domino using your custom CI/CD pipeline. Domino supports REST APIs that enable you to programmatically build new model images on Domino and export them to your external container registry.

### 2.8.6 Web applications

When experiments in Domino yield interesting results that you want to share with your colleagues, you can easily do so with a *Domino App*. Domino supports hosting Apps built with many popular frameworks, including Flask, Shiny, and Dash.

While Apps can be significantly more sophisticated and provide far more functionality than a Launcher, they also require significantly more code and knowledge in at least one framework. In this section, we will convert some of the code that we developed in *Step 5* and create a *Dash* app.
1. Add the `app.py` file, which will describe the app in Dash, to the project:

```python
# -*- coding: utf-8 -*-
import dash
import dash_core_components as dcc
import dash_html_components as html
from datetime import datetime as dt
from dash.dependencies import Input, Output
import requests
import datetime
import os

import pandas as pd
import matplotlib.pyplot as plt
from fbprophet import Prophet
import plotly.graph_objs as go

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

app.config.update({'requests_pathname_prefix': '/{}
→{}/'.format(os.environ.get("DOMINO_PROJECT_OWNER"),
os.environ.get("DOMINO_PROJECT_NAME"),
os.environ.get("DOMINO_RUN_ID")))

colors = {
    'background': '#111111',
    'text': '#7FDBFF'
}

# Plot configs
prediction_color = '#0072B2'
error_color = 'rgba(0, 114, 178, 0.2)' # '#0072B2' with 0.2 opacity
actual_color = 'black'
cap_color = 'black'
trend_color = '#B23B00'
line_width = 2
marker_size = 4
uncertainty=True
plot_cap=True
trend=False
changepoints=False
changepoints_threshold=0.01
xlabel='ds'
ylabel='y'

app.layout = html.Div(style={'paddingLeft': '40px', 'paddingRight': '40px
→'}, children=[
    html.H1(children='Predictor for Power Generation in UK'),
    html.Div(children='''
This is a web app developed in Dash and published in Domino.
You can add more description here to describe the app.
'''),
    html.Div({
        'Select a Fuel Type:', className='fuel_type', id='fuel_type_paragraph'),
```
(continues on next page)
dcc.Dropdown(
    options=[
        {'label': 'Combined Cycle Gas Turbine', 'value': 'CCGT'},
        {'label': 'Oil', 'value': 'OIL'},
        {'label': 'Coal', 'value': 'COAL'},
        {'label': 'Nuclear', 'value': 'NUCLEAR'},
        {'label': 'Wind', 'value': 'WIND'},
        {'label': 'Pumped Storage', 'value': 'PS'},
        {'label': 'Hydro (Non Pumped Storage', 'value': 'NPSHYD'},
        {'label': 'Open Cycle Gas Turbine', 'value': 'OCGT'},
        {'label': 'Other', 'value': 'OTHER'},
        {'label': 'France (IFA)', 'value': 'INTFR'},
        {'label': 'Northern Ireland (Moyle)', 'value': 'INTIRL'},
        {'label': 'Netherlands (BritNed)', 'value': 'INTNED'},
        {'label': 'Ireland (East-West)', 'value': 'INTEW'},
        {'label': 'Biomass', 'value': 'BIOMASS'},
        {'label': 'Belgium (Nemolink)', 'value': 'INTEM'},
        {'label': 'France (Eleclink)', 'value': 'INTEL'},
        {'label': 'France (IFA2)', 'value': 'INTIFA2'},
        {'label': 'Norway 2 (North Sea Link)', 'value': 'INTNSL'},
    ],
    value='CCGT',
    id='fuel_type',
    style={'width':'auto', 'min-width': '300px'}
),
html.Div([html.Div('Training data will end today.'), html.Div('Select the starting date for the training data:'), html.Div(''),
    dcc.DatePickerSingle(
        id='date-picker',
        date=dt(2020, 9, 10)
    )], style={'marginTop': 25}),
html.Div([dcc.Loading(
    id="loading",
    children=[dcc.Graph(id='prediction_graph',)],
    type="circle",
),
], style={'marginTop': 25})
)
today = datetime.datetime.today().strftime('%Y-%m-%d')
start_date_reformatted = start_date.split('T')[0]
url = 'https://www.bmreports.com/bmrs/?q=ajax/filter_csv_download/
    →FUELHH/csv/FromDate%3D{start_date}&ToDate%3D{today}/&
    →filename=GenerationbyFuelType_20191002_1657'.format(start_date = start_
    →date_reformatted, today = today)
open('data.csv', 'wb').write(r.content)
df = pd.read_csv('data.csv', skiprows=1, skipfooter=1,
    →header=None, engine='python')
df.columns = ['HDF', 'date', 'half_hour_increment',
    'CCGT', 'OIL', 'COAL', 'NUCLEAR',
    'WIND', 'PS', 'NPSHYD', 'OCGT',
    'OTHER', 'INTFR', 'INTIRL', 'INTNED', 'INTEW', 'BIOMASS',
    'INTEM',
    'INTEL', 'INTIFA2', 'INTNSL']
df['datetime'] = pd.to_datetime(df['date'], format='%Y%m%d')
df['datetime'] = df.apply(lambda x:
    x['datetime']+ datetime.timedelta(minutes=30*(int(x['half_hour_increment'])-1)),
    axis = 1)
df_for_prophet = df[['datetime', fuel_type]].rename(columns = {
    'datetime': 'ds', fuel_type: 'y'})
m = Prophet()
m.fit(df_for_prophet)
future = m.make_future_dataframe(periods=72, freq='H')
fcs = m.predict(future)
# from https://github.com/facebook/prophet/blob/master/python/
    →fbprophet/plot.py
data = []
# Add actual
    data.append(go.Scatter(
        name='Actual',
        x=m.history['ds'],
        y=m.history['y'],
        marker=dict(color=actual_color, size=marker_size),
        mode='markers')
    )
# Add lower bound
    if uncertainty and m.uncertainty_samples:
        data.append(go.Scatter(
            x=fcs['ds'],
            y=fcs['yhat_lower'],
            mode='lines',
            line=dict(width=0),
            hoverinfo='skip'
            ))
# Add prediction
    data.append(go.Scatter(
        name='Predicted',
        x=fcs['ds'],
        y=fcs['yhat'],
        mode='lines',
        line=dict(color=prediction_color, width=line_width),
        fillcolor=error_color,
        fill='tonexty' if uncertainty and m.uncertainty_samples_
    )
    else 'none'
# Add upper bound
    if uncertainty and m.uncertainty_samples:
        data.append(go.Scatter(
            x=fcst['ds'],
            y=fcst['yhat_upper'],
            mode='lines',
            line=dict(width=0),
            fillcolor=error_color,
            fill='tonexty',
            hoverinfo='skip'
        ))

# Add caps
    if 'cap' in fcst and plot_cap:
        data.append(go.Scatter(
            name='Cap',
            x=fcst['ds'],
            y=fcst['cap'],
            mode='lines',
            line=dict(color=cap_color, dash='dash', width=line_width),
            fillcolor=error_color,
            fill='tonexty',
            hoverinfo='skip'
        ))

    if m.logistic_floor and 'floor' in fcst and plot_cap:
        data.append(go.Scatter(
            name='Floor',
            x=fcst['ds'],
            y=fcst['floor'],
            mode='lines',
            line=dict(color=cap_color, dash='dash', width=line_width),
            fillcolor=error_color,
            fill='tonexty',
            hoverinfo='skip'
        ))

# Add trend
    if trend:
        data.append(go.Scatter(
            name='Trend',
            x=fcst['ds'],
            y=fcst['trend'],
            mode='lines',
            line=dict(color=trend_color, width=line_width),
            fillcolor=error_color,
            fill='tonexty',
            hoverinfo='skip'
        ))

# Add changepoints
    if changepoints:
        signif_changepoints = m.changepoints[
            np.abs(np.nanmean(m.params['delta'], axis=0)) >=
            changepoints_threshold
        ]
        data.append(go.Scatter(
            x=signif_changepoints,
            y=fcst.loc[fcst['ds'].isin(signif_changepoints), 'trend'],
            marker=dict(size=50, symbol='line-ns-open', color=trend_color,
                        line=dict(width=line_width)),
            mode='markers',
            hoverinfo='skip'
        ))

layout = dict(
    showlegend=False,
    )
2. Add an `app.sh` file to the project, which provides the commands to instantiate the app:

```bash
python app.py
```

3. Publish the App.

   3.1. Navigate to the App page under the Publish menu of your project.
   
   3.2. Enter a title and a description for your app.

2.8. Step 7: Deploy your model
3.3. Click Publish.

3.4. Once the app status appears as “Running” (which may take a few minutes), you can click View App to open it.

4. Share your app with your colleagues.

4.1 Navigate to the Publish/App page and select the Permissions tab.

4.2. Invite your colleagues by username or email.

4.3. Or, toggle the Access Permissions level to make it publicly available.

Learn more about Domino Apps.
2.9 Domino training videos

Log in to https://learn.dominodatalab.com/ for access to Domino training videos.

2.9.1 Domino 4.5.x training

- Domino 101 for Domino 4.5.x
- Domino 201 for Domino 4.5.x
- Publishing in Domino 4.5.x

2.9.2 Domino 4.4.x training

- Domino 101 for Domino 4.4.x
- Domino 201 for Domino 4.4.x
- Publishing in Domino 4.4.x
This tutorial will guide you through a common model lifecycle in Domino. You will start by working with data from the Balancing Mechanism Reporting Service in the UK. We will be exploring the Electricity Generation by Fuel Type and predicting the electricity generation in the future. You’ll see examples of Rstudio, Shiny, tidyverse, and Prophet used in Domino.

The following contents is mostly meant to be following in sequence.
3.1 Step 0: Orient yourself to Domino (R Tutorial)

When you first log in, you will find yourself in the Lab section of Domino on the Projects Overview page. You can use the left side bar to navigate to other areas of Domino.

If you would like to search the Domino documentation for help, click on the Help icon on the bottom left of the page. To send a question to a member of the Domino support staff, use the Support button on the bottom right of the page.

3.2 Step 1: Create a project

Work in Domino happens in projects. Projects contain data, code, and environment settings, and the entire project is tracked and revisioned automatically. A new commit is written to a project each time its files are changed by user action, or by the execution of code in the project. Users in Domino can create their own new projects, invite other users to collaborate on them, and export data or results for consumption by other projects.

To create a project:

1. In the left sidebar, click on

2. Click on on the right side of the page.
2.1. Give your project an informative name (like power-generation).

2.2. Set its Visibility to Private.

2.3. Click Create Project.

3.3 Step 2: Configure your project

- Introduction
- Step 2.1: Select your hardware tier
- Step 2.2: Configure your environment
- Step 2.3: Configure the project permissions
3.3.1 Introduction

Every project has its own settings. The following options are important to consider when configuring a new project:

- Hardware tier
- Environment
- Collaborators

3.3.2 Step 2.1: Select your hardware tier

A hardware tier is a representation of the compute resources that will be available. In addition to memory and CPU cores, you can also choose to use GPUs with hardware tiers.

The hardware tier dropdown menu lists your available options. The selected hardware tier will be used by default for all subsequent executions of code in the project. It can also be changed at any point in the future.

1. In the Project menu, click Settings.
2. Click on the Hardware tier dropdown menu to choose the compute resource to execute your code on. Your options may look different from the image below.
   2.1. Choose the smallest or default hardware tier for this tutorial.

This list of available hardware tiers is customizable by your Domino administrators. If you would like additional resources, contact your Domino administrator.
3.3.3 Step 2.2: Configure your environment

An Environment is a Domino abstraction on top of a Docker image, that provides additional flexibility and versioning, which allows you to configure the software, packages, libraries, and drivers that you need.

Domino comes with a default environment called the Domino Analytics Distribution, which includes Python, R, Jupyter, RStudio, and hundreds of data science related packages and libraries.

We’ll choose the default environment which includes a recent version of R.

1. Click on the Compute Environment dropdown menu to choose the Environment.


* Your compute environment dropdown will likely have different options.

If you’re interested in learning how to add more packages and customize or create your own Environment, see the help article on Domino Environments.
3.3.4 Step 2.3: Configure the project permissions

As the owner of the project, you can set different access levels for collaborators from the Collaborators and permissions panel.

Feel free to invite a colleague to be a Contributor to your project.

1. (Optional) Enter the email or the username of the user that you would like to invite
2. (Optional) Enter a welcome message to be sent to your collaborator

The Contributor role allows the invited user to read, write, and execute code in this project.

Visit the Collaborators and permissions support article for more information on the permissions for each collaborator role.

3.4 Step 3: Start a workspace

Workspace sessions are interactive sessions hosted by a Domino executor where you can interact with code notebooks like Jupyter and RStudio. The software tools and associated configurations available to you are called Workspaces.

For this tutorial, we will start an Rstudio Workspace.
1. Click on Workspaces from the project menu.

2. Select Rstudio.

3. Click Launch Rstudio Workspace.

When you launch a workspace, a new containerized session is created on a machine (also known as an executor) in the required hardware tier. The workspace tool you requested is launched in that container, and your browser is automatically redirected to the workspace’s UI when it’s ready.
Once your workspace is up and running, you will see a fresh Rstudio interface. If you’re brand new to Rstudio, you might find the Rstudio intro and cheatsheet helpful.

If you are interested in adding additional Workspaces for tools that are available by default, see the pluggable notebooks section of your Domino environment documentation.

### 3.5 Step 4: Get your files and data

There are basically two strategies to working with data in Domino:

- You can copy your data into Domino
  
  If you are working with data that is on your local machine or in a shared server, you might want to upload your data into Domino.

- You can query your data from Domino
  
  If you have a large dataset stored in a database or data service, you may just need to query the database or the API for the data service.

In this tutorial, we will use the terminal in Rstudio to copy data into the project.

1. If you have not done so, please complete Step 3 to start a Rstudio workspace.

Your starting file path is /mnt. By default, this is considered the root of your Domino project. If you add or modify files in /mnt, you can save them back to your project when you stop or sync the workspace.

2. Use the Tools > Terminal > New Terminal menu to open a Rstudio terminal.
3. In the new terminal, run the following command to fetch some data from the BMRS:

```
curl -o data.csv "https://www.bmreports.com/bmrs/?q=ajax/filter_csv_→download/FUELHH/csv/FromDate%3D2019-09-15%26ToDate%3D2019-10-02/&→filename=GenerationbyFuelType_20191002_1657"
```

4. Do a Sync All Changes to save the new file `data.csv` to Domino.

3.5. Step 4: Get your files and data
Sync all changes will save any changes that were made in your workspace session back to your project.

5. Return to your project in Domino by navigating to a new browser tab. Then, navigate to the “Files” section of Domino. Notice that the raw data has been saved in the latest revision.

See the documentation on other methods of both copying data into Domino and querying data from Domino.

3.6 Step 5: Develop your model

- Introduction
- Step 5.1: Load and explore the dataset
3.6.1 Introduction

When you are developing your model, you want to be able to quickly execute code, see outputs, and make iterative improvements. Domino enables this with Workspaces. Step 3 covered starting a Workspace and explored Workspace options like VSCode, RStudio, and Jupyter.

In this section, we will use Rstudio to load, explore, and transform some data. After the data has been prepared, we will train a model.

3.6.2 Step 5.1: Load and explore the dataset

1. If you have not done so, please complete Step 4 to download the dataset.

2. You should see data.csv in /mnt in files pane. If not, please return to Step 4 to download the dataset.

3. Use the New menu to create a R script and save as power.R.
4. Enter these lines to import some packages:

```r
library(tidyverse)
library(lubridate)
```

5. Create a list of the columns according to information on the column headers at https://www.bmreports.com/bmrs/?q=generation/fueltype/current.

```r
col_names <- c('HDF', 'date', 'half_hour_increment', 'CCGT', 'OIL', 'COAL', 'NUCLEAR', 'WIND', 'PS', 'NPSHYD', 'OCGT', 'OTHER', 'INTFR', 'INTIRL', 'INTNED', 'INTEW', 'BIOMASS', 'INTEM')
```

6. Next, read the file you downloaded into a dataframe and set the column names, then display the data:

```r
#Load the data into a data frame
df <- read.csv('data.csv',header = FALSE,col.names = col_names,
               stringsAsFactors = FALSE)

#remove the first and last row
df <- df[-1,]
df <- df[-nrow(df),]

#Preview the data
View(df)
```

Execute the script by selecting ‘Code’ menu -> ‘Source’ command. This will view the content of the df data frame in a new tab:

We can see that this is a time series dataset. Each row is a successive half hour increment.
during the day that details the amount of energy generated by fuel type. Time is specified by the date and half_hour_increment columns.

7. Tidy that data so that variables are in columns, observations are in rows, and values are in cells. Switch back to the power.R code tab and add the following:

```r
df_tidy <- df %>% gather('CCGT', 'OIL', 'COAL', 'NUCLEAR', 'WIND', 'PS', 'NPSHYD', 'OCGT', 'OTHER', 'INTFR', 'INTIRL', 'INTNED', 'INTEW', 'BIOMASS', 'INTEM', key="fuel", value="megawatt" )
```

8. Create a new column datetime that represents the starting datetime of the measured increment. For example, a 20190930 date and 2 half hour increment means that the time period specified is September 19, 2019 from 12:30am to 12:59am.

```r
df_tidy <- df_tidy %>% mutate(datetime=as.POSIXct(as.Date(date, "%Y%m%d"))+minutes(30*(half_hour_increment-1)))
```

9. Visualize the data to see how each fuel type is used during the day by plotting the data.

```r
#plot the graph
p <- ggplot(data=df_tidy, aes(x=datetime, y=megawatt, group=fuel)) + geom_line(aes(color=fuel))
print(p)
```

Execute the script again by selecting ‘Code’ menu -> ‘Source’ command. This will update the ‘Plot’ tab:

The CCGT column representing “combined-cycle gas turbines” seems to be the most interesting. It generates a lot of energy and is very volatile.

We will concentrate on this column and try to predict the power generation from this fuel source.

3.6. Step 5: Develop your model
3.6.3 Step 5.2: Train a model

Data scientists have access to many libraries and packages that help with model development. Some of the most common for R are randomForest, caret, and nnet. These packages are already installed in the Domino Analytics Distribution, the default environment. However, there may be times that you want to experiment with a package that is new and not installed in the environment.

We will build a model with the Facebook Prophet package, which is not installed into the default environment. You will see that you can quickly get started with new packages and algorithms just as fast as they are released into the open source community.

1. In your R console, install Facebook Prophet. Note that this may take >5 minutes to install. If it fails, make sure that you’ve selected a hardware tier with >2gb of ram:

```
# Install specific version of the prerequisite RcppParallel package as its latest version 5.0.2 fails to install
install.packages('https://cran.r-project.org/src/contrib/Archive/RcppParallel/RcppParallel_5.0.1.tar.gz', repos=NULL, type="source")
# Now install latest version of Facebook Prophet
install.packages('prophet')
```

2. For Facebook Prophet, the time series data needs to be in a DataFrame with 2 columns named ds and y. Let’s rename the columns and filter to just to fuel type “CCGT”:

```
df_CCGT <- df_tidy %>% filter(fuel=="CCGT") %>% select(datetime,megawatt)
names(df_CCGT) <- c("ds","y")
```

3. Split the dataset into train and test sets:

```
split_index <- round(nrow(df_CCGT)*.8)
df_CCGT_train <- df_CCGT[1:split_index,]
df_CCGT_test <- df_CCGT[(split_index+1):nrow(df_CCGT),]
```

4. Import Facebook Prophet and fit a model:

```
library(prophet)
m <- prophet(df_CCGT_train)
```

5. Make a DataFrame to hold prediction and predict future values of CCGT power generation:

```
predict_ln <- round(nrow(df_CCGT_test)/2)
future <- make_future_dataframe(m, periods = predict_ln,freq = 1800 )
forecast <- predict(m, future)
```

6. Plot the fitted line with the training and test data:

```
p <- dyplot.prophet(m, forecast)
print(p)
```
7. Save the code.

3.6.4 Step 5.3: Export the model

Trained models are meant to be used. There is no reason to re-train the model each time you use the model. Export or serialize the model to a file to load and reuse the model later. In R, you can commonly use the `saveRDS` command to create RDS files.

1. Export the trained model as an rds file for later use:

   ```r
   saveRDS(m, file = "model.rds")
   ```

   We will use the serialized model in Step 7 when we create an API from the model.

3.7 Step 6: Clean up [Workspaces]

- **Introduction**
  - Option 1: Stop your Workspace session from inside the workspace
  - Option 2: Stop your Workspace session from the Workspaces page

3.7.1 Introduction

To avoid spending unnecessary compute resources, make sure to stop any Workspace sessions that you started as a part of this tutorial. If your Domino is deployed in the cloud, this will prevent you from incurring unnecessary charges. If your Domino is deployed on premises, this will free up your compute resources for others to use.

There are two places in the Domino UI that you can stop your session.
Option 1: Stop your Workspace session from inside the workspace

1. Above your Rstudio workspace in the blue menu bar, save your files and click Stop.

2. Enter a descriptive commit message.

3. Click Stop and Commit.
Option 2: Stop your Workspace session from the Workspaces page

1. Go the to the Workspaces page in Domino. Since the Rstudio workspace opened in a new tab, you may have to select the previous Domino tab.

2. Click on the Stop button corresponding to your workspace session.
3. Click Stop and Commit.

In both options, you were able to select Stop and Commit. The Stop and Commit button stops the workspace session and also saves your work back to your project. The new files created from Step 5 should now be visible on the Files page.

3.8 Step 7: Deploy your model

- Introduction
- Compute Environments
- Scheduled reports
- Launchers
- Model APIs
- Web applications
3.8.1 Introduction

Once you have developed your model and deemed it good enough to be useful, you will want to deploy it. There is no single deployment method that is best for all models. Therefore, Domino offers four different deployment options. One may fit your needs better than the others depending on your use case.

The available deployment methods are:

- Scheduled reports
- Launchers
- Web applications
- Model APIs

The remaining sections of this tutorial are not dependent on each other. For example, you will not need to complete the Scheduled report section to understand and complete the Web application section.

3.8.2 Compute Environments

In our previous section, Step 5, we installed the Prophet package in Rstudio in order to train the model. In Domino any package installed in one worksession will not persist to another. In order to avoid having to re-install Prophet each time we need it, you can add it to a custom compute environment.

1. Create a new compute environment.
   1.1. Go to the Environments page in Domino.
1.2. Click Create Environment.

1.3. Name the environment and enter a description for the new environment.
1.4. Click Create Environment.

1.5. Click Edit Definition in the top right corner of the page.

1.6. In the Dockerfile Instructions section, enter the following:

```
RUN R --no-save -e "install.packages(c('prophet'))"
```

1.7. Scroll to the bottom of the page and click Build

This will start the creation of your new compute environment. These added packages will now be permanently installed into your environment and be ready whenever you start a job or workspace session with this environment selected.

3.8. Step 7: Deploy your model
1.8. Go back to your project page and navigate to the Settings page.

1.9. Select your newly created environment from the Compute Environments dropdown menu.

If you want to learn more about how to customize your environment, check out the Environments tutorials. You can also learn more about what’s included in our default environments, the Domino Analytics Distribution.

3.8.3 Scheduled reports

The Scheduled Jobs feature in Domino allows you to run a script on a regular basis. In Domino, using the R package knitr, you can blend text, code, and plots in an RMarkdown to create attractive HTML or pdf reports automatically.

In our case, we can imagine that each day we receive new data on power usage and want to email out a visualization of the latest data daily.

1. Start a new Rstudio session.

2. Create a new Rmarkdown file named `power_report.Rmd` and select HTML as our desired output.
3. RStudio automatically creates a sample Rmarkdown file for you, but you can replace it entirely with the following which reuses code from our `power.R` script from step 5.

```{r}
---
title: "Power_Report"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
library(tidyverse)
library(lubridate)
col_names <- c('HDF', 'date', 'half_hour_increment', 'CCGT', 'OIL', 'COAL', 'NUCLEAR', 'WIND', 'PS', 'NPSHYD', 'OCGT', 'OTHER', 'INTFR', 'INTIRL', 'INTNED', 'INTEW', 'BIOMASS', 'INTEM')
df <- read.csv('data.csv', header = FALSE, col.names = col_names, stringsAsFactors = FALSE)
# remove the first and last row
df <- df[-1,]
df <- df[-nrow(df),]
# Tidy the data
df_tidy <- df %>% gather('CCGT', 'OIL', 'COAL', 'NUCLEAR', 'WIND', 'PS', 'NPSHYD', 'OCGT', 'OTHER', 'INTFR', 'INTIRL', 'INTNED', 'INTEW', 'BIOMASS', 'INTEM')
```
R Markdown

Combining R Markdown, Knitr and Domino allows you to create attractive scheduled reports that mix text, code and plots.

```
R
```{r, echo=FALSE, warning=FALSE}
df_tidy <- df_tidy %>% mutate(datetime=as.POSIXct(as.Date(date, "%Y%m%d")) + minutes(30*(half_hour_increment-1)))
print(head(df_tidy))
```

Including Plots

You can also embed plots, for example:

```
R
```{r, echo=FALSE}
p <- ggplot(data=df_tidy, aes(x=datetime, y=megawatt, group=fuel)) + geom_line(aes(color=fuel))
print(p)
```

4. With your new Rmarkdown file, you can “knit” this into an html file and preview it directly in Domino by hitting the “Knit” button.

5. To create a repeatable report, you need to create a script that you can schedule that will automatically render your Rmarkdown file to html. Start by creating a new R script named render.R with the following code:

```
rmarkdown::render("power_report.Rmd")
```

6. Save your files and Stop and Commit your workspace.

7. Go to the Scheduled Jobs page.
8. Enter the file that you want to run. This will be the `render.R` script you created earlier.

9. Select how often and when to run the file.

10. Enter emails of people to send the resulting file(s) to.

11. Click Schedule.

To discover more tips on how to customize the resulting email, see Custom Notifications for more information.
3.8.4 Launchers

Launchers are simple web forms that allow users to run templatized scripts. They are especially useful if your script has command line arguments that dynamically change the way the script executes. For heavily customized script, those command line arguments can quickly get complicated. Launcher allow you to expose all of that as a simple web form.

Typically, we parameterize script files (i.e. files that end in .py, .R, or .sh). Since we have been working with an R script until now, we will parameterize and reuse our R script that we created in Step 5.

To do so, we will insert a few new lines of code into a copy of the R script, and configure a Launcher.

1. Parameterize your R script by setting it to take command line arguments:

   1. Start an Rstudio session.
   1.2. Create script named Power_for_Launcher.R with the following:

```r
library(tidyverse)
library(lubridate)

#Pass in command line arguments
args <- commandArgs(trailingOnly = TRUE)
fuel_type <- args[1]

col_names <- c('HDF', 'date', 'half_hour_increment',
               'CCGT', 'OIL', 'COAL', 'NUCLEAR',
               'WIND', 'PS', 'NPSHYD', 'OCGT',
               'OTHER', 'INTFR', 'INTIRL', 'INTNED',
               'INTEW', 'BIOMASS', 'INTEM')
df <- read.csv('data.csv', header = FALSE, col.names = col_names,
              stringsAsFactors = FALSE)
#remove the first and last row
df <- df[-1,]
df <- df[-nrow(df),]
#Tidy the data
df_tidy <- df %>% gather('CCGT', 'OIL', 'COAL', 'NUCLEAR',
                         'WIND', 'PS', 'NPSHYD', 'OCGT',
                         'OTHER', 'INTFR', 'INTIRL', 'INTNED',
                         'INTEW', 'BIOMASS', 'INTEM', key="fuel", value="megawatt"
                         )
#Create a new column datetime that represents the starting datetime of
#the measured increment.
df_tidy <- df_tidy %>% mutate(datetime=as.POSIXct(as.Date(date, "%Y%m%d
"))+minutes(30*(half_hour_increment-1)))
#Filter the data
df_fuel_type <- df_tidy %>% filter(fuel==fuel_type) %>% select(datetime,
                         megawatt)
#Save out data as csv
write.csv(df_fuel_type, paste(fuel_type,"_",Sys.Date(),".csv",sep=""))
```

1.3. Notice the lines in our script that define an object from a command line arguments.
1.4. Save the files and Stop and Commit the workspace session.

2. Configure the Launcher.

2.1. Go to the Launcher page. It is under the Publish menu on the project page.

2.2. Click New Launcher.

2.3. Name the launcher “Power Generation Forecast Data”

2.4. Copy and paste the following into the field “Command to run”:

```
Power_for_Launcher.R ${fuel}
```

You should see the parameters show up below:
2.5. Select the fuel_type parameter and change the type to Select (Drop-down menu).

2.5.1. Copy and paste the following into the “Allowed Values” field:

CCGT, OIL, COAL, NUCLEAR, WIND, PS, NPSHYD, OCGT, OTHER, ...
INTFR, INTIRL, INTNED, INTEW, BIOMASS, INTEM

2.6. Click Save Launcher

3. Try out the Launcher.

3.1. Go back to the main Launcher page.

3.2. Click Run for the “Power Generation Forecast Trainer” launcher.

3.3. Select a fuel type from the dropdown.

3.4. Click Run
This will execute the parameterized R script with the parameters that you selected. In this particular launcher, your dataset is filtered based on your input parameter with the results returned as a csv. When the run has been completed, an email will be sent to you and others that you optionally specified in the launcher with the resulting files. If you would like to customize the email, see Custom Notifications for more information.
3.8.5 Model APIs

If you want your model to serve another application, you will want to serve it in the form of an API endpoint. Domino Model APIs are scalable REST APIs that can create an endpoint from any function in a Python or R script. The Domino Model APIs are commonly used when you need an API to query your model in near real-time.

For example, we created a model to forecast power generation of combined cycle gas turbines in the UK.

In this section, we will deploy an API that uses the model that we trained in Step 5 to predict the generated power given a date in the future. To do so, we will create a new compute environment to install necessary packages, create a new file with the function we want to expose as an API, and finally deploy the API.

1. Create a new file with the function we want to expose as an API

   2.1. From the Files page of your project, click Add File.

   2.2. Name your file forecast_predictor.R.

   2.3. Enter the following contents:

   ```r
   library("prophet")
   m <- readRDS(file = "model.rds")

   model_api <- function(year, month, day, hour, minute) {
     date <- paste(year, "-", month, "-", day, " ", hour, ":", minute, sep="")
     date = as.POSIXct(date, "%Y-%m-%d %H:%M")
     df_api <- data.frame(ds=date)
     df2 <- predict(m, df_api)
     return(df2["yhat"])
   }
   ```

   2.4. Click Save.
3. Deploy the API.

   3.1. Go to the Publish/Model APIs page in your project.

   3.2. Click New Model.

   3.3. Name your model, provide a description, and click Next.
3.4. Enter the name of the file that you created in the previous step.
3.5. Enter the name of the function that you want to expose as an API.
3.6. Click Create Model.

4. Test the API.
   4.1 Wait for the Model API status to turn to Running. This may take a few minutes.
   4.2 Click the Overview tab.
   4.3 Enter the following into the tester:

   ```
   {
       "data": {
           "year": 2019,
       }
   }
   ```
4.4. Click Send. If successful, you will see the response on the right panel.

As a REST API, any other common programming language will be able to call it. Code snippets from some popular languages are listed in the other tabs.

Model APIs are built as docker images and deployed on Domino. You can export the model images to your external container registry and deploy them in any other hosting environment outside of Domino using your custom CI/CD pipeline. Domino supports REST APIs that enable you to programmatically build new model images on Domino and export them to your external container registry.

3.8.6 Web applications

When experiments in Domino yield interesting results that you want to share with your colleagues, you can easily do so with a Domino App. Domino supports hosting Apps built with many popular frameworks, including Flask, Shiny, and Dash.

While Apps can be significantly more sophisticated and provide far more functionality than a Launcher, they also require significantly more code and knowledge in at least one framework. In this section, we will convert some of the code that we developed in Step 5 and create a Shiny app.
1. Add the app.R file, which will describe the app in Shiny, to the project:

```r
library(tidyverse)
library(lubridate)
library(prophet)
library(dygraphs)

col_names <- c('HDF', 'date', 'half_hour_increment',
               'CCGT', 'OIL', 'COAL', 'NUCLEAR',
               'WIND', 'PS', 'NPSHYD', 'OCGT',
               'OTHER', 'INTFR', 'INTIRL', 'INTNED',
               'INTEW', 'BIOMASS', 'INTEM')

df <- read.csv('data.csv',header = FALSE,col.names = col_names,
               stringsAsFactors = FALSE)

#remove the first and last row
df <- df[-1,]
df <- df[-nrow(df),]

fuels <- c('CCGT', 'OIL', 'COAL', 'NUCLEAR',
          'WIND', 'PS', 'NPShyd', 'OCGT',
          'OTHER', 'INTFR', 'INTIRL', 'INTNED',
          'INTEW', 'BIOMASS', 'INTEM')

predict_ln <- round((nrow(df))*.2)

#Tidy the data and split by fuel
df_tidy <- df %>%
  mutate(ds=as.POSIXct(as.Date(date, "%Y%m%d")+minutes(30*(half_hour_increment-1)))) %>%
  select(-c('HDF', 'date', 'half_hour_increment')) %>%
  gather("fuel", "y", -ds) %>%
  split(.,$fuel)

#remove unused column
df_tidy <- lapply(df_tidy, function(x) { x["fuel"] <- NULL; x })

#Train the model
m_list <- map(df_tidy, prophet)

#Create dataframes of future dates
future_list <- map(m_list, make_future_dataframe, periods = predict_ln, freq = 1800)

#Pre-Calc yhat for future dates
forecast_list <- map2(m_list, future_list, predict) # map2 because we have two inputs

ui <- fluidPage(
  verticalLayout(
    h2(textOutput("text1")),
    selectInput(inputId = "fuel_type", label = "Fuel Type",
                 choices = fuels, selected = "CCGT"),
    dygraphOutput("plot1"))
)
```

(continues on next page)
server <- function(input, output) {
  output$plot1 <- renderDygraph({
    forecast <- predict(m_list[[input$fuel_type]], future_list[[input$fuel_type]])
    dyplot.prophet(m_list[[input$fuel_type]], forecast)
  })
  output$text1 <- renderText({ input$fuel_type })
}
shinyApp(ui = ui, server = server)

2. Add an app.sh file to the project, which provides the commands to instantiate the app:

```sh
R -e 'shiny::runApp("app.R", port=8888, host="0.0.0.0")'
```

3. Publish the App.

   3.1. Navigate to the App page under the Publish menu of your project.
   3.2. Enter a title and a description for your app.

![Publish App](image)

3.3. Click Publish.

3.4. Once your app starts successfully, which may take a few minutes, you can click View App to open it.

4. Share your app with your colleagues.

   4.1 Back on the Publish/App page, select the App Permissions tab.
   4.2. Invite your colleagues by username or email.
   4.3. Or, toggle the Access Permissions level to make it publicly available.

3.8. Step 7: Deploy your model 89
Learn more about *Domino Apps*.

### 3.9 Domino training videos

Log in to [https://learn.dominodatalab.com/](https://learn.dominodatalab.com/) for access to Domino training videos.

#### 3.9.1 Domino 4.5.x training

- Domino 101 for Domino 4.5.x
- Domino 201 for Domino 4.5.x
- Publishing in Domino 4.5.x

#### 3.9.2 Domino 4.4.x training

- Domino 101 for Domino 4.4.x
- Domino 201 for Domino 4.4.x
- Publishing in Domino 4.4.x
In this walkthrough, we’ll look at historical weather data for Berlin, Germany to predict future weather in Berlin. We’ll develop a model to help a fictional friend decide whether or not they should purchase an air conditioner (something uncommon in Germany).

Here are some questions that Domino will help us answer:

• How many “hot days” occur in a year?
• How many “hot days” will occur in the next 20 days?

We’ll also decide what temperature counts as a “hot day”.
4.1 Step 0: Orient yourself to Domino

When you first log in, you will find yourself in the Lab section of Domino on the Projects Overview page. You can use the left side bar to navigate to other areas of Domino.

If you would like to search the Domino documentation for help, click on the Help icon on the bottom left of the page.

To send a question to a member of the Domino support staff, use the Support button on the bottom right of the page.

4.2 Step 1: Create a Domino Project

Work in Domino happens in projects. Projects contain data, code, and environment settings, and the entire project is tracked and revisioned automatically. A new commit is written to a project each time its files are changed by user action, or by the execution of code in the project. Users in Domino can create their own new projects, invite other users to collaborate on them, and export data or results for consumption by other projects.

To create a project:

1. In the left sidebar, click on

2. Click on on the right side of the page.
2.1. Give your project an informative name (like weather-prediction)

2.2. Set its Visibility to Private.

2.3. Click Create Project.

4.3 Step 2: Configure Your Domino Project

Every project has its own settings. The following options are important to consider when configuring a new project:

- Hardware Tier
- Environment
- Collaborators
4.3.1 Step 2.1: Select Your Hardware Tier

A Hardware Tier represents the compute resources that will be available for your run. You can specify memory, CPU cores, and GPUs with hardware tiers.

The hardware tier dropdown menu lists your available options. The selected hardware tier will be used by default for all subsequent executions of code in the project. It can also be changed at any point in the future.

1. In the Project menu, click Settings.

2. Click on the Hardware tier dropdown menu to choose the compute resource to execute your code on. Your options may look different from the image below.

   2.1. Choose the smallest or default hardware tier for this tutorial.

This list of available hardware tiers is customizable by your Domino administrators. If you would like additional resources, contact your Domino administrator.

4.3.2 Step 2.2: Configure Your MATLAB Environment

You can configure the software, packages, libraries, and drivers that you need by defining your Environment.

Depending on your corporate Domino setup, Domino may include one or more options for MATLAB. Environments normally map to MATLAB Releases (e.g., R2019a) or to combinations of releases and toolboxes (e.g., MATLAB and Statistics and Machine Learning Toolbox, MATLAB and Simulink, etc.).

To access the MATLAB environment, navigate to the Compute Environment menu and select the MATLAB version you would like to use. Note: The menu options may be different based on your Domino installation.
4.3.3 Step 2.3: Configure Project Permissions

As the owner of the project, you can set different access levels for collaborators and colleagues. Feel free to invite a colleague to be a Contributor to your project.

1. Click on the Access & Sharing tab at the top of the page
2. (Optional) Enter the email or the username of the user that you would like to invite
3. (Optional) Enter a welcome message to be sent to your collaborator

The **contributor** role allows the invited user to read, write, and execute code in this project.
Visit the **Collaborators and permissions** support article for more information on the permissions for each collaborator role.

### 4.4 Step 3: Start a MATLAB Workspace

Workspace sessions are interactive sessions hosted by a Domino executor where you can interact with code in MATLAB stored in Domino.

#### 4.4.1 Step 3.1: Launch a Workspace

1. Create a new workspace by clicking the + **New Workspace** button in the **Workspaces** menu.
2. Enter a name for your workspace and ensure that your workspace environment is set to a MATLAB environment. All other options can be left as-is for the purposes of this walkthrough. Launch the workspace (you may need to monitor your browser to ensure a new tab appears).
3. Your browser will be automatically redirected to the workspace’s UI when it’s ready. Once your workspace
is up and running, you’ll see a MATLAB desktop. If you’re new to MATLAB, you might find the MATLAB documentation and on-ramp tutorials helpful.

4.5 Step 4: Fetch and Save Your Data

There are basically two strategies to working with data in Domino:

- You can copy your data into Domino
  
  If you are working with data that is on your local machine or in a shared server, you might want to upload your data into Domino.

- You can query your data from Domino
  
  If you have a large dataset stored in a database or data service, you may just need to query the database or the API for the data service.

In this step, you’ll copy your data into the project using the MATLAB terminal. If you have not done so, please complete Step 3 to start a MATLAB workspace.

The starting file path in your MATLAB workspace is /mnt. By default, this is considered the root of your Domino project. If you add or modify files in /mnt, you can save them back to your project when you stop or sync the workspace.
4.5.1 Step 4.1: Fetch Your Data

In the command window, enter the following command to fetch some data from the US NOAA Climatology Service:

```bash
!curl -o tegel.csv https://www.ncei.noaa.gov/data/global-historical-climatology-network-daily/access/GME00121150.csv
```

The downloaded file will appear in the Current Folder file navigator of your MATLAB desktop.
4.5.2 Step 4.2: Save Your Data to Your Domino Project

To save this data to your Domino project, click the **Save & Push All** button, enter an informative commit message, and then click **Continue with Sync**. After doing this, Domino will show you which files have changed during your session.

**Note:** Note that the temperature data is in tenths of Celsius degrees. Learn more here: README File for Daily Global Historical Climatology Network.
>> !curl -o tegel.csv https
total  | received | x
---|---|---
0 | 0 | 0
5 | 187k | 0
100 | 3136k | 0

>>
4.5. Step 4: Fetch and Save Your Data
4.6 Step 5: Develop Your Model

As you develop a model, you’ll want to be able to quickly execute code, see outputs, and make iterative improvements. If you have not done so, please complete Step 3 to start a MATLAB workspace.

In this section, we will use MATLAB to load, explore, and transform some data. After the data has been prepared, we will train a model.

**Attention:** Be sure to occasionally save your work by clicking the **Save & Push All** button in Domino.

4.6.1 Step 5.1: Load and Explore the Dataset

1. Be sure you have downloaded and saved the `tegel.csv` dataset from Step 4 in this walkthrough. You should see the file in your **Current Browser** file navigator.
2. Use the New menu in the MATLAB desktop to create a new MATLAB Live Script. Save it as `ber_hot_weather.mlx`.
3. Examine the file by typing the following command into your Live Script:

```matlab
opts = detectImportOptions('tegel.csv');
```

4. Next, load `tegel.csv` into MATLAB. The file has four columns we want to keep (the rest we can ignore):
- **Date** – date the temperature was read
- **PRCP** – total precipitation for the day
- **TMIN** – lowest temperature measured that day
- **TMAX** – highest temperature measured that day

Run the following command to specify these columns when loading the file and to assign data types to them. This command will also load the data using the `readtable()` function, giving it the import options object as a second argument:

```matlab
opts.SelectedVariableNames = {'DATE', 'PRCP', 'TMIN', 'TMAX'};
opts = setvartype(opts, {'DATE','PRCP','TMIN','TMAX'},{'datetime','double','double','double'});
berWeatherTbl = readtable("tegel.csv", opts);
head(berWeatherTbl)
```

The result of running this command should look similar to the following:

<table>
<thead>
<tr>
<th></th>
<th>DATE</th>
<th>PRCP</th>
<th>TMIN</th>
<th>TMAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12-Jun-193..</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2</td>
<td>15-Jun-193..</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>3</td>
<td>16-Jun-193..</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>4</td>
<td>18-Jun-193..</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>5</td>
<td>19-Jun-193..</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>6</td>
<td>22-Jun-193..</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>7</td>
<td>23-Jun-193..</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>8</td>
<td>24-Jun-193..</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>

5. Create a new section in your script by clicking the **Section Break** button.
In this section, we’ll format the dates into a Year/Month/Day format and store each field as a table variable. This will help with examining the data. Run the following command:

\[
\text{[berWeatherTbl.year, berWeatherTbl.month, berWeatherTbl.day]} = \text{ymd(berWeatherTbl.DATE)};
\]

To speed up processing of the data, our data set will be limited to temperatures between January 2000 and December 2019 (currently, the last full year of data), inclusive. Let’s remove the rows with data outside that range.

\[
\text{berWeatherTbl} = \text{berWeatherTbl(berWeatherTbl.year > 1999 & berWeatherTbl.year <= max(berWeatherTbl.year), :)};
\]

Notice that temperatures in the TMAX or TMIN columns look a bit odd (outside of data denoted as NaN, or “not a number”). NOAA uses a temperature format consisting of a tenth-of-a-degree in the Celsius scale. We therefore need to divide all temperature data by 10 to get temperatures in full Celsius degrees. Run the following command:

\[
\text{berWeatherTbl.TMAX} = \text{berWeatherTbl.TMAX/10};
\text{berWeatherTbl.TMIN} = \text{berWeatherTbl.TMIN/10};
\]

Like most data sources, some data may be missing, so we’ll fill in missing data with interpolated information by running the following command:

\[
\text{berWeatherTbl} = \text{fillmissing(berWeatherTbl, 'linear')};
\]

Preview the start of the table by running the head() function as follows:

\[
\text{head(berWeatherTbl)}
\]

The result should look like the following:
6. Create another section in your live script. In this section, we’ll calculate how many hot days have occurred in Berlin since the year 2000. To calculate this, we’ll first need to define “hot day”. We’ll use 29 degrees Celsius for our definition, and use this as our baseline threshold, as such:

   ```matlab
   hotDayThreshold = 29;
   ```

7. Now let’s figure out how many hot days have occurred since (and including) the year 2000. To do this, we’ll create a table column indexing the days with maximum temperatures (\text{TMAX}) that meet or exceed the hot day threshold.

   ```matlab
   berWeatherTbl.HotDayFlag = berWeatherTbl.TMAX >= hotDayThreshold;
   ```

Next, we’ll use \texttt{groupsummary()} to count how many hot days were flagged:

   ```matlab
   numHotDaysPerYear = groupsummary(berWeatherTbl, 'year', 'sum', 'HotDayFlag');
   ```

We will repeat the same approach to find the highest temperature of each year:

   ```matlab
   maxTempOfYear = groupsummary(berWeatherTbl, 'year', 'max', 'TMAX');
   ```

We will finally combine the two variables to create a table called \texttt{annualMaxTbl}:

   ```matlab
   annualMaxTbl = join(numHotDaysPerYear, maxTempOfYear);
   annualMaxTbl.Properties.VariableNames = {'Year', 'daysInYear', 'hotDayCount', 'maxTemp'};
   annualMaxTbl
   ```

The table should look like the following:
8. Create another section in your live script. In this section, we’ll visualize the weather data using a chart with that combines a bar graph and line graph. The chart will use two y-axes. The bar graph will represent the hot day count (for a given year), and the line graph will represent the highest annual temperature (in Celsius, for a given year). The y-axis on the left side of the chart will correspond to the hot day count, and the y-axis on the right side of the chart will correspond to the highest annual temperature.

Start with the hot day count bar graph:

```matlab
figure
hold on
yyaxis left
bar(annualMaxTbl.Year, annualMaxTbl.hotDayCount, 'FaceColor', 'b');
```

Add titles and labels to the x-axis and left side y-axis:

```matlab
titleText = sprintf("%s%d%s%d", "Number of hot days (over ", ...
              hotDayThreshold, "\circC) - ", min(annualMaxTbl.Year), "-"),
              max(annualMaxTbl.Year));
title(titleText)
ylabel("Hot days per year")
xlabel("Year")
```

Now draw the line plot for the highest temperature each year:

```matlab
yyaxis right
ylabel("Highest Annual Temperature in \circC")
plot(annualMaxTbl.Year, annualMaxTbl.maxTemp, 'Color', 'r', "Marker", '*')
hold off
```

Your chart should look something like this:
4.6.2 Step 5.2: Generating Predictions from the Data

In this step, we will use an interactive machine-learning MATLAB application called **Regression Learner** to develop a model that can predict the weather for the next 20 days.

First, let’s prepare the data that we will use with Regression Learner. We’ll partition the data into two sets: one set to train the model, and a second set to test the model.

1. Create a new section in the live script and remove the `HotDayFlag` column.

   ```matlab
   berWeatherTbl.HotDayFlag = [];   
   ```

Partition the data:
2. Next, find the Regression Learner app by clicking on the **Apps** tab in the MATLAB toolbar. If you do not see the Regression Learner app, you may need to expand the full app list by clicking on the arrow to the far right of the list.

3. Click on the Regression Learner icon. The application will open in a new, blank window.

**Tip:** You can use the Domino desktop window controls to navigate between windows (in case the Regression Learner window disappears behind the MATLAB window).
4. Click the **New Session** button and select “From Workspace” in the dropdown.
5. A “New Session” window will open. Here we can specify the input variables that should be used for prediction in our model, as well as the outputs (or “response”) you would like to predict – for us, that means the maximum temperature.

For the input variable, select `dataTrain`, under the “Workspace Variable” section of the window.

For the output, select `TMAX` (maximum temperature), under the “Response” section of the window. Click the Start Session button.
The Regression Learner window will refresh and display the original data set and values of TMAX.
6. Next, select the type of model that should be used for model training. For this walkthrough, we’ll use the “Fine Gaussian SVM” and “Coarse Tree” models and compare the results. Feel free to select additional models if you’d like. Alternatively, you can select all models and compare the results for the best fit for your data.
Attention: Regression Learner runs best on a container with multiple cores. Multiple cores allow it to run in parallel and produce models rapidly. If you are using a single-core container, please turn off parallel processing by clicking the “Use Parallel” button in Regression Learner.

7. Click the **Train** button in the Regression Learner toolbar to start the model training process. The Domino container will spin up a “parallel pool” – a method to optimize the model training. Once the models finish training, the model list will automatically pick the model that best fits the data. Several visualizations are offered to demonstrate this fit (the visualization shown below is the default visualization).
Clicking on the Predicted vs. Actual Plot button in the toolbar will display a chart that shows how many of the predictions that were made by the model fit correct values in the data. The closer the predictions are to the diagonal, the better the predictions will be.

8. In a later step, we’ll deploy a model using Domino. To do that, we first need to create a function that can be used
for model deployment. We can do that with Regression Learner. Click the **Generate Function** button.

MATLAB will generate the function in an M-file (as shown below). Save the file as `trainRegressionModel.m`.

9. Let’s export the model to your Domino workspace so that we can use it for predictions. Navigate back to the Regression Learner window and export the model as shown below.
Give the model a name, such as `weatherModel`, and click “OK”.

Close the Regression Learner app (confirm your decision in the pop-up) and you’ll see the trained model available to you in your workspace.
You will also notice that the Command Window shows information on how to use the model to make predictions, specifically with the following line of code:

\[
yFit = weatherModel.predictFcn(T);
\]

This line of code will output a prediction (in the form of a table) as a result of inputting a table of data. The input table must include data organized similarly to the data we had in `berWeatherTbl`—date, precipitation, minimum temperature, month, day and year (it should not include TMAX, as that value will be predicted). The model will predict the TMAX value and include it in `yFit`.

10. Let’s test the model with the data we partitioned earlier. Create a new section in your live script. Use the model with the test data, using the function call we saw in the Command Window:

\[
yFit = weatherModel.predictFcn(dataTest);
\]

Now compare the results column to the actual values in the test data set:

\[
err = yFit - dataTest.TMAX;
\]

Finally, draw a histogram to visualize the results:

```matlab
figure;
histogram(err);
xlim([-15 15])
ylabel('Number of predictions');
xlabel('Gap with actual test data')
```

The result should look like the following:
Now that we’ve seen that the model works, let’s save it for later use. In the Command Line window, enter:

```
save weathermodel weathermodel
```
11. We are finally ready to use our model to predict the weather for the next year. We will generate a table with next year’s dates and add randomly selected, historical precipitation and minimum temperature data to the table (for the same dates), which we’ll need for the model to properly make predictions.

Create a new section in your live script, then create a new table with date and temperature input data:

```matlab
todayDate = datetime('today');
daysIntoFuture = 365;
endDate = todayDate + days(daysIntoFuture);
predictedMaxTemps = table('Size', [daysIntoFuture+1 7], 'VariableTypes', ...
  {'datetime', 'double', 'double', 'double', 'double', 'double', 'double'}, ...
  'VariableNames', berWeatherTbl.Properties.VariableNames);
x=1;
```

Next, loop through the next 20 days and populate the table.

```matlab
for i=todayDate:endDate
    [y, m, d] = ymd(i);
    prcps = berWeatherTbl.PRCP(berWeatherTbl.month == m & berWeatherTbl.day == d);
end
```

(continues on next page)
The result will be a preview of a table containing historical weather data that we can use for our weather predictions. The predictions will appear in the TMAX column of the table after the table is run through the model.

12. Let’s run the model!

```matlab
yFit = weatherModel.predictFcn(predictedMaxTemps);
result = table(predictedMaxTemps.DATE, yFit, 'VariableNames', {'Date', 'Predicted TMAX'});
```
Your very own, AI-driven, weather prediction!

13. Let’s draw this out in another plot and count how many hot days will be forecasted:

```matlab
result = 366×2 table

<table>
<thead>
<tr>
<th>Date</th>
<th>Predicted TMAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-Sep-20...</td>
<td>24.2708</td>
</tr>
<tr>
<td>11-Sep-20...</td>
<td>22.8500</td>
</tr>
<tr>
<td>12-Sep-20...</td>
<td>22.4090</td>
</tr>
<tr>
<td>13-Sep-20...</td>
<td>24.4699</td>
</tr>
<tr>
<td>14-Sep-20...</td>
<td>20.7627</td>
</tr>
<tr>
<td>15-Sep-20...</td>
<td>20.6574</td>
</tr>
<tr>
<td>16-Sep-20...</td>
<td>19.8160</td>
</tr>
<tr>
<td>17-Sep-20...</td>
<td>19.6501</td>
</tr>
<tr>
<td>18-Sep-20...</td>
<td>20.0744</td>
</tr>
<tr>
<td>19-Sep-20...</td>
<td>20.0720</td>
</tr>
</tbody>
</table>
```

```matlab
g = figure
plot(result.Date, result.("Predicted TMAX"))
titleText = sprintf("%s%d%s", "Weather forecast for the next ",...
                    daysIntoFuture, " days in Berlin, Germany (\circ C));
title(titleText)
ylabel('Forecasted Daily High Temperature')
```
We can also predict how many hot days will happen during the next year:

```matlab
hotWeatherDaysIdx = result(result.('Predicted TMAX') > hotDayThreshold, :);
height(hotWeatherDaysIdx)
```

The result on September 10, 2020 was a prediction of 17 hot days between September 2020 and October 2021. The results may vary for you based on the dates, data, and model used.

To export your model, all you need to do is save it into a MAT file:

```matlab
save weatherModel weatherModel
```

Anyone in your Domino project will be able to load it later using:

```matlab
load weathermodel.mat
```

### 4.7 Step 6: Clean Up Your Workspace

To avoid spending unnecessary compute resources, make sure to stop any workspace sessions that you started as a part of this tutorial. This will prevent you from incurring unnecessary charges if your Domino is deployed in the cloud. If your Domino is deployed on premises, this will free up your compute resources for others to use.

There are two places in the Domino UI that you can stop your session.
Option 1: Stop your workspace session from inside the workspace.

1. Save your files and click “Stop” in the Domino menu bar.

2. Confirm your decision by clicking “Yes, Stop”. The MATLAB session will close and a message will appear informing you that the session has been stopped.
Option 2: Stop your workspace session from the Workspaces section.

1. Navigate to the Workspaces section in Domino (this may be in a different tab).
2. Click on the Stop button corresponding to your workspace session.
3. Click “Stop and commit results”.

The “Stop and commit results” button stops the workspace session and also saves your work to your Domino project. The new files created from Step 5 should now be visible in the Files section of Domino.

### 4.8 Step 7: Deploy Your Model

Once you have developed your model and deemed it good enough to be useful, you will want to deploy it. There is no single deployment method that is best for all models. Therefore, Domino offers four different deployment options. One may fit your needs better than the others depending on your use case.

The available deployment methods are:

- Scheduled Reports
- Launchers
- Web Applications
- Model APIs
For this tutorial, we’ll show you how to deploy your MATLAB-generated model using “Scheduled Reports” or “Launchers”. The sections in this step are not dependent on each other. For example, you will not need to complete the Scheduled Jobs section to understand and complete the Launchers section.

### 4.8.1 Scheduled Jobs

The Scheduled Jobs feature in Domino allows you to run a script on a regular basis. You can create reports in MATLAB using the `publish()` function and the robust MATLAB Report Generator. To simplify things, we will use the `publish()` function, which uses a MATLAB m. file as a document template.

In our case, let’s imagine that we receive new data everyday about Berlin’s weather. We decide we want to generate a scheduled email visualizing this data, as well as the forecasted number of hot days in the next 365 days.

To do that, we will create two files:

- An m. file, based on our live script, that will:
  - Load Berlin weather data from a URL
  - Prepare the data
  - Generate predictions using the model we create with the data
  - Call the `publish()` function

- An m. file that will be our report template, displaying:
  - Weather prediction plot
  - Number of predicted hot days

### Step 7.1: Publish the Weather Prediction Report

1. Start a new MATLAB session.
2. Create a new file named `predictWeatherReport.m`.
3. Add the following code to your file. This code will initialize a `struct` to hold the results, define a hot day temperature threshold (in degrees Celsius), and download the current data for Berlin weather and save it to the workspace. As before, we will read the downloaded file into a table format.

```matlab
%% Initial setup
result = struct;
hotDayThreshold = 30;

%% Download data file
urlString = 'https://www.ncei.noaa.gov/data/global-historical-climatology-network-daily/access/GME00121150.csv';
if ~isfolder('data')
    mkdir('data');
end
```
4. Let’s prepare the data for use. We will again start with data from the year 1999, adjust the temperature data to full degrees, and fill in any missing data. If there are less than 1000 rows of data, we will quit processing the data.

```matlab
% Read the downloaded file
savedFileName = sprintf("%s%s", "data", filesep, "berlin.csv");
websave(savedFileName, urlString);

%% Read the downloaded file
opts = detectImportOptions(savedFileName);
opts.SelectedVariableNames = {'DATE', 'PRCP', 'TMIN', 'TMAX'};
opts = setvartype(opts, {'DATE','PRCP','TMIN','TMAX'},{'datetime','double',
    'double', 'double'});
stationWeatherTbl = readtable(savedFileName, opts);
```

4. Next, let’s load the model we created for Berlin and save it into a .mat file. Then we’ll create the table to use as input with the updated data we read from the URL in step 3.

```matlab
%% Check if we have a model for this weather station
modelFileName = sprintf("%s%s%s%s", "models", filesep, ... weatherStationId,
    ".mat");

% make sure we have a folder for the models
if ~isfile(modelFileName)
    mkdir('models');
end

if ~isfile(modelFileName)
    disp('Training model for weather station...');
    cv = cvpartition(stationWeatherTbl.year, 'Holdout', 0.3);
    dataTrain = stationWeatherTbl(cv.training, :);
    [weatherModel, validationRMSE] = trainRegressionModel(dataTrain);
    disp(validationRMSE);
    doneMessage = sprintf('%s%d', "Done. Model RMSE:", validationRMSE);
    disp(doneMessage);
    save(modelFileName, 'weatherModel');
end
```

5. Next, let’s load the model we created for Berlin and save it into a .mat file. Then we’ll create the table to use as input with the updated data we read from the URL in step 3.
else
    load(modelFileName, 'weatherModel');
end

```matlab
%% Create table for future date prediction
todayDate = datetime('today');
daysIntoFuture = 365;
endDate = todayDate + days(daysIntoFuture);
predictedMaxTemps = table('Size', [daysIntoFuture+1 7], 'VariableTypes', ...
    {'datetime', 'double', 'double', 'double', 'double', 'double', 'double'}, ...
    'VariableNames', stationWeatherTbl.Properties.VariableNames);
x=1;
for i=todayDate:endDate
    % get the average perception and minimum temps on this date
    [y, m, d] = ymd(i);
    minTemps = stationWeatherTbl.TMIN(stationWeatherTbl.month == m & ...
        stationWeatherTbl.day == d);
    prcps = stationWeatherTbl.PRCP(stationWeatherTbl.month == m & ...
        stationWeatherTbl.day == d);
    curMinTemp = NaN;
    historicalRowCount = size(minTemps);
    randomRow = randi([1 historicalRowCount]);
    curMinTemp = minTemps(randomRow);
    predictedMaxTemps.TMIN(x) = curMinTemp;
    randomRow = randi([1 historicalRowCount]);
    predictedMaxTemps.PRCP(x) = prcps(randomRow);
    predictedMaxTemps.DATE(x) = i;
    predictedMaxTemps.month(x) = m;
    predictedMaxTemps.year(x) = y;
    predictedMaxTemps.day(x) = d;
    predictedMaxTemps.TMAX(x) = 0;
    x = x+1;
end
```

6. Run the model and load the result struct with the prediction.

```matlab
%%
yFit = weatherModel.predictFcn(predictedMaxTemps);
predResult = table(predictedMaxTemps.DATE, yFit, 'VariableNames', {'Date', ...
    'Predicted TMAX'});
result.predictedTemps = predResult;
hotWeatherDaysIdx = predResult(predResult.("Predicted TMAX") > ...
    hotDayThreshold, :);
result.hotDayCountPrediction = height(hotWeatherDaysIdx);
```

7. The publish() function runs in isolation from the workspace. As such, we need to share the prediction result with our template using a .mat file. To ensure the data file has a unique name for each run of this script, we will use the Domino environment variable for the run number.

```matlab
%% save data to file
dominoRunId = getenv('DOMINO_RUN_NUMBER');
outputFileName = sprintf('%s%s%s', 'results', filesep, 'predictData_','.string(dominoRunId));
```
8. Finally, call the publish() function. The report will be published to a subfolder of the results/ folder, along with the number of the current run in the filename.

```matlab
%% Publish the report
% options for the report
pub_options.format = 'pdf';
% hide the report code
pub_options.showCode = false;
pub_options.outputDir = sprintf('%s%s%s', 'results', filesep, dominoRunId);
doc = publish('predictWeatherReportTemplate.m', pub_options);
```

Step 7.2: Create the Report Template

1. Now let’s create the report template. Create a new file named predictWeatherReportTemplate.m and load the data. Use the following format for the filename when loading the data: results/predictData_<Domino Run Number>. We’ll store the data in a variable called result.

```matlab
dominoRunId = getenv('DOMINO_RUN_NUMBER');
inputFileName = sprintf('%s%s%s', 'results', filesep, 'predictData_', string(dominoRunId));
load(inputFileName, 'result');
```

2. Add a title to the template. Note that in a MATLAB template of this kind, comments will be rendered as markup. For more information, please read MATLAB’s documentation on markup comments for publishing.

```matlab
%% Predicted Weather
```

3. Finally, add the plot with the data we loaded and display the hot day predictions as output.

```matlab
predResult = result.predictedTemps;
plot(predResult.Date, predResult.('Predicted TMAX'));
titleText = 'Weather forecast for the next 365 days (°C)';
title(titleText);
ylabel ('Forecasted Daily High Temperature')

%%
countPredictionText = sprintf("%s%d%s", "There will be ", ...
   result.hotDayCountPrediction, " hot days in the next 365 days");
disp(countPredictionText);
```
4. Save and commit your changes using the **Save & Push All** button in the Domino toolbar.

5. Let’s test the `predictWeatherReport.m` file. Call the file in the “Command Window”. A new folder and new file will be created inside of the **results/** folder.

```
fs >> predictWeatherReport
```

6. Once again, save and commit your changes.
7. Stop your MATLAB session and navigate to the “Scheduled Jobs” page in Domino.

8. Schedule a new job by clicking the + New Scheduled Job button.
9. Give the new scheduled job a name and select your hardware tier. Enter `predictWeatherReport.m` as the file to run for this job. Click Next.
10. Omit the “Spark Cluster” section, as we will not be attaching a Spark cluster.

11. Schedule the job to run every weekday. Leave the “Run sequentially” option checked. Click Next.
12. Finally, enter the email addresses of individuals you like to notify when the job is complete. Click **Create**.
You have now scheduled a job that will use your MATLAB code to generate a report. You can similarly run the job on an ad-hoc basis using the Domino “Jobs” function.

To discover more tips on how to customize the resulting email, refer to Domino’s documentation on Custom Notifications.

### 4.8.2 Launchers

Launchers are simple web forms that allow users to run templatized scripts. They are especially useful if your script has command line arguments that dynamically change the way the script executes. For heavily customized scripts, those command line arguments can quickly get complicated. Launchers allow you to expose all of that as a simple web form.

To do this with MATLAB, we will refactor the .m file we used in our scheduled job as a function, and give Launcher users the ability to specify two parameters:

- The NOAA station ID (station IDs are listed here: [GHCND Stations](#))
- A “hot day” temperature threshold
Step 7.1: Refactor the .m file

1. Start a new MATLAB Workspace. We’ll update our scheduled job code.

2. Open the file we previously created, `predictWeatherReport.m`, and save it as `predictWeatherReport-Launcher.m`.

3. Start by wrapping the script with a function statement. The function statement requires an end statement as well. To avoid confusion with code placement, we’ll add the end statement as the last step of this walkthrough.

   ```matlab
   function result = predictWeatherReport(weatherStationId, hotDayThreshold)
   
   result = struct;
   % Download data file
   baseUrlString = "https://www.ncei.noaa.gov/data/global-historical-climatology-network-daily/access/";
   urlString = sprintf("%s%s", baseUrlString, weatherStationId, ".csv");
   % save the data into the /data subfolder
   savedFileName = sprintf("%s%s%s%s", "data", filesep, weatherStationId, "csv");
   websave(savedFileName, urlString);
   
   % Read the downloaded file
   opts = detectImportOptions(savedFileName);
   opts.SelectedVariableNames = {'DATE', 'PRCP', 'TMIN', 'TMAX'};
   opts = setvartype(opts, {'DATE','PRCP','TMIN','TMAX'},{'datetime','double',
                      'double', 'double'});
   stationWeatherTbl = readtable(savedFileName, opts);
   
   % MATLAB strength
   [stationWeatherTbl.year, stationWeatherTbl.month, stationWeatherTbl.day] = ymd(stationWeatherTbl.DATE);
   
   % check if there is enough data for prediction
   dataRows = size(stationWeatherTbl, 1);
   if dataRows < 1000
       disp('Not enough data for prediction')
       result.error = 'Not enough data for prediction';
       return;
   end
   
   result = struct;
   % Download data file
   baseUrlString = "https://www.ncei.noaa.gov/data/global-historical-climatology-network-daily/access/";
   urlString = sprintf("%s%s", baseUrlString, weatherStationId, ".csv");
   % save the data into the /data subfolder
   savedFileName = sprintf("%s%s%s%s", "data", filesep, weatherStationId, "csv");
   websave(savedFileName, urlString);
   
   % Read the downloaded file
   opts = detectImportOptions(savedFileName);
   opts.SelectedVariableNames = {'DATE', 'PRCP', 'TMIN', 'TMAX'};
   opts = setvartype(opts, {'DATE','PRCP','TMIN','TMAX'},{'datetime','double',
                      'double', 'double'});
   stationWeatherTbl = readtable(savedFileName, opts);
   
   % MATLAB strength
   [stationWeatherTbl.year, stationWeatherTbl.month, stationWeatherTbl.day] = ymd(stationWeatherTbl.DATE);
   
   % check if there is enough data for prediction
   dataRows = size(stationWeatherTbl, 1);
   if dataRows < 1000
       disp('Not enough data for prediction')
       result.error = 'Not enough data for prediction';
       return;
   end
   ```

4. Let’s refactor the first section of our code:

   ```matlab
   result = struct;
   % Download data file
   baseUrlString = "https://www.ncei.noaa.gov/data/global-historical-climatology-network-daily/access/";
   urlString = sprintf("%s%s", baseUrlString, weatherStationId, ".csv");
   % save the data into the /data subfolder
   savedFileName = sprintf("%s%s%s%s", "data", filesep, weatherStationId, "csv");
   websave(savedFileName, urlString);
   ```

5. The next section of code should remain unchanged. For reference, here it is again:

   ```matlab
   % Read the downloaded file
   opts = detectImportOptions(savedFileName);
   opts.SelectedVariableNames = {'DATE', 'PRCP', 'TMIN', 'TMAX'};
   opts = setvartype(opts, {'DATE','PRCP','TMIN','TMAX'},{'datetime','double',
                      'double', 'double'});
   stationWeatherTbl = readtable(savedFileName, opts);
   
   % MATLAB strength
   [stationWeatherTbl.year, stationWeatherTbl.month, stationWeatherTbl.day] = ymd(stationWeatherTbl.DATE);
   
   % check if there is enough data for prediction
   dataRows = size(stationWeatherTbl, 1);
   if dataRows < 1000
       disp('Not enough data for prediction')
       result.error = 'Not enough data for prediction';
       return;
   end
   ```
6. Since Launcher users will be able to request predictions for any weather station, we’ll first need to check whether a model for that station exists. If a model does not exist, we will train a model using the file we saved earlier, `trainRegressionModel` (in Step 5), along with the data we downloaded. Models will have a filename format of `<station ID>.mat`.

```matlab
%% Check if we have a model for this weather station
modelFileName = sprintf("%s%s%s%s", "models", filesep, ... weatherStationId, ".mat");

% make sure we have a folder for the models
if ~isfolder('models')
    mkdir('models')
end

if ~isfile(modelFileName)
    disp('Training model for weather station...')
    cv = cvpartition(stationWeatherTbl.year, 'Holdout', 0.3);
    dataTrain = stationWeatherTbl(cv.training, :);

    [weatherModel, validationRMSE] = trainRegressionModel(dataTrain);

    % display prediction precision
    doneMessage = sprintf('%s%d', "Done. Model RMSE:", validationRMSE);
    disp(doneMessage);
    save(modelFileName, 'weatherModel');
else
    load(modelFileName, 'weatherModel');
end
```

7. Let’s re-use the code we previously had for creating the data table to drive the prediction model.

```matlab
%% Create table for future date prediction
todayDate = datetime('today');
daysIntoFuture = 365;
endDate = todayDate + days(daysIntoFuture);
predictedMaxTemps = table('Size', [daysIntoFuture+1 7], 'VariableTypes', ...
    {'datetime', 'double', 'double', 'double', 'double', 'double', 'double'}, ...
    'VariableNames', stationWeatherTbl.Properties.VariableNames);
x=1;

for i=todayDate:endDate
    % get the average perception and minimum temps on this date
    [y, m, d] = ymd(i);

    minTemps = stationWeatherTbl.TMIN(stationWeatherTbl.month == m & ...
        stationWeatherTbl.day == d);
    prcps = stationWeatherTbl.PRCP(stationWeatherTbl.month == m & ...
        stationWeatherTbl.day == d);

    curMinTemp = NaN;
    [historicalRowCount z] = size(minTemps);
    randomRow = randi([1 historicalRowCount]);
    curMinTemp = minTemps(randomRow);
    predictedMaxTemps.TMIN(x) = curMinTemp;
    randomRow = randi([1 historicalRowCount]);
end
```

(continues on next page)
predictedMaxTemps.PRCP(x) = prcps(randomRow);
predictedMaxTemps.DATE(x) = i;
predictedMaxTemps.year(x) = y;
predictedMaxTemps.month(x) = m;
predictedMaxTemps.day(x) = d;
predictedMaxTemps.TMAX(x) = 0;
x = x+1;
end

8. Keep the rest of the file as-is. Here is the code again for reference:

```matlab
%% run model with future data
yFit = weatherModel.predictFcn(predictedMaxTemps);
predResult = table(predictedMaxTemps.DATE, yFit, 'VariableNames', {'Date', 'Predicted TMAX'});
result.predictedTemps = predResult;
hotWeatherDaysIdx = predResult(predResult.
  "Predicted TMAX" > hotDayThreshold, :);
result.hotDayCountPrediction = height(hotWeatherDaysIdx);

%% save data to file
dominoRunId = getenv('DOMINO_RUN_NUMBER');
outputFileName = sprintf('%s%s%s', 'results', filesep, 'predictData_','predictWeatherReportTemplate.m', pub_options);
save(outputFileName, 'result');

%% publish to report
pub_options.format = 'pdf';
pub_options.showCode = false;
pub_options.outputDir = sprintf('%s%s%s', 'results', filesep, dominoRunId);
doc = publish('predictWeatherReportTemplate.m', pub_options);
```

9. Don’t forget to add the end statement for our function.

```matlab
%% Function end
end
```

10. Click the Save & Push All button to save your work. Stop the workspace.

---

### Step 7.2: Set up the shell script for the Launcher

1. Close the workspace tab in your browser and expand the “Publish” menu in Domino.

2. Next, we’ll need to create a shell script that will specify to the Launcher which MATLAB script should run, along with which parameters should be passed to the script from the launcher. Navigate to the “Files” section under your project in Domino.
4.8. Step 7: Deploy Your Model
3. Click the new file icon to create a new file. Name the file `weather_launcher.sh`.

4. Add the following code starting on line 1 of the file:

```
matlab -nodisplay -nodesktop -nosplash -r "predictWeatherReportLauncher('$1', $2')"
```

This line of code instructs Domino to run MATLAB from the command line and execute the script called `predictWeatherReportLauncher` with two arguments. MATLAB will look for the function in an .m file with the same name as the function being called.

5. Click the Save button to save your work.

---

**Step 7.3: Create the Launcher**

1. Click “Launchers” from inside the “Publish” menu, then click the New Launcher button.
4.8. Step 7: Deploy Your Model
2. Give the launcher a descriptive title (e.g., “Weather Predictor”), a description, and select a hardware tier. In the “Command to run” section, enter `weather_launcher.sh`.

3. Click the Add Parameter button. This section of the modal will expand to show parameter options.
4. In the “Command to run” field, replace `parameter1` with `station_id`. You’ll notice that the parameter’s name in the form below will update. Ensure that the parameter name remains enclosed within `${}`, such that the parameter is formatted as such: `${name_of_the_parameter}`.
5. Let’s give this parameter a default value for the station ID. We’ll use \texttt{MXM00076680} (for Mexico City). In the description, enter “The ID of the station to predict weather for”. The default value will be used if the user does not enter any value in the Launcher form.
6. Click the Add Parameter button once again and enter `hot_temp` as the parameter's name. This parameter will represent the hot temperature threshold. Keep the parameter's type as “Text” and enter a default value (e.g., 30) and a description for the parameter.
7. Finally, click Save Launcher and click the Back to Launchers link on the upper left corner of the form. You should see your launcher in the launcher list.
8. Click the Run button. The launcher form will appear. Enter a title for your run (e.g., “First launcher run”) and click Run.
Weather Predictor

Predicts daily maximum temperatures from historical data.

station_id
MXM00076680
The ID of the station to predict weather for

hot_temp
30
Threshold determining what the minimum temperature is on a hot day.

Title your run. This will appear on the Runs dashboard.

Run title

Notify other users when this run is done by filling in their emails

e.g., 'chris@foo.com, jim@bar.com' or 'John
9. You should see the “Jobs” view in Domino while your launcher executes the selected job. Note the job number (in this example, job number 61). Expand the “User Output” by clicking on the toggle next to it. You should see the results as Domino run the job.
Chapter 4. Get Started with MATLAB
10. Running this job will produce a PDF report. To access the PDF report, navigate to “Files” within Domino, open the “Results” folder, open the folder created by your job (in this example, it is folder 61/), and click on the PDF file to open and display it.
Chapter 4. Get Started with MATLAB
4.8. Step 7: Deploy Your Model
Chapter 4. Get Started with MATLAB
4.9 Step 8: Working with Domino Datasets

If your Domino project uses a very large number of files (e.g., more than 10,000), or a single file larger than 8GB, consider using a Domino dataset.

To summarize the lifecycle of a dataset, consider this workflow:

- Datasets are defined in a .yaml file, along with input folders and output folders.
- A newly defined dataset will live in the input folder specified in the .yaml file. By default, the dataset in the input folder is read-only, while files in the output folder are writable.
- If you do not write anything to the output folder, the dataset will remain unchanged (as-is).
- Any files that you’d like to persist from the dataset in the input folder must be copied to the output folder.
- If you write to the output folder, the dataset files will be overwritten. Datasets, however, are saved as snapshots, allowing you to roll back to a previous snapshot of the dataset if needed.

This step will provide an example of how to use a dataset with our weather project.

4.9.1 Step 8.1: Add a dataset to your project

1. Navigate to the Data section of your project.
2. Click **Create New Dataset**.

3. Enter a name (i.e., “get-started-MATLAB-dataset”) and description for your dataset, then click **Upload Contents**.

4. Next, we’ll create a dataset by creating an initial snapshot (the initial version of your dataset). For our project, we’ll create the dataset using a MATLAB workspace. Click **+ New Workspace** in the “Create with Interactive Workspace” option. The “Launch New Workspace” modal should appear.

5. Select MATLAB as your workspace IDE. Click **Launch**. Your MATLAB workspace will launch with a new folder used to store the data that is part of your dataset.

6. Locate this new folder by first clicking the “/” in the file path of your MATLAB workspace. Next, navigate to the dataset folder that Domino created for you. The folder path should be: `/domino/datasets/domino/datasets/output/get-started-MATLAB-dataset`. Let’s put some data in that folder.
Trial License ... for use to evaluate programs for possible purchase as an end-user only.
4.9. Step 8: Working with Domino Datasets
7. To populate the dataset, we will download a series of weather station files from the same NOAA repository we used earlier in the project. In our usual work directory (/mnt), create file called `downloadToDatasetDir.m`. In it, we’ll create a function to download the NOAA data:

```
function downloadToDatasetDir()
    % NOAA data URL
    baseUrlString = "https://www.ncei.noaa.gov/data/global-historical-climatology-network-daily/access/";

    % Prefix shared by weather stations in Argentina
    baseWeatherStationId = 'AR0000000';

    % the location to save the files - the dataset output directory
    datasetFolder = "/domino/datasets/domino/datasets/output/get-started-MATLAB-dataset/";

    % There are 16 weather station files. We will iterate and download each one
    for counter=1:16
        if counter<10
            weatherStationId = sprintf('%ss%s%d', baseWeatherStationId, '0', counter);
        else
            weatherStationId = sprintf('%ss%s%d', baseWeatherStationId, counter)
        end

        urlString = sprintf("%ss%s", baseUrlString, weatherStationId, "csv")
        savedFileName = sprintf("%ss%s", datasetFolder, weatherStationId, "csv");
        websave(savedFileName, urlString);
    end
end
```

8. Save the file, then run it from the Command Window in your MATLAB workspace. The output should look something like this as each file is downloaded.

```
urlString = 'https://www.ncei.noaa.gov/data/global-historical-climatology-network-daily/access/AR00000001S.csv'
weatherStationId = 'AR00000001S'
urlString = 'https://www.ncei.noaa.gov/data/global-historical-climatology-network-daily/access/AR00000001G.csv'
```

9. Now let’s save the files we downloaded to Domino. Stop your workspace by clicking the **Stop** button in your workspace. The “Uncommitted Changes” modal will appear. Enter a commit message and click **Stop and Commit**. Note that the files we saved to the dataset location will be saved as well.
11. Domino will display confirmation messages. Proceed to shut down your workspace. Close the browser tab and return to the Domino project. Return to the Data section and notice that the dataset is now available to you.
12. Click the dataset. You should see the list of files we just downloaded. They are now part of the dataset.
4.9.2 Step 8.2: Updating and writing to a dataset

The dataset you’ve created is now available in a read-only state. If you want to modify the dataset, or if your MATLAB algorithm will produce a very large number of files, you’ll need to create a new snapshot.

All new snapshots start out as empty and read-only. To create a snapshot that persists the previous snapshot’s data, you’ll need to copy the previous snapshot’s data into the new snapshot. To do this, we’ll need to create a file called domino.yaml.
1. To create `domino.yaml`, navigate to the Files section of your project.
2. Click the new file icon.

3. In the file editor that appears, enter `domino.yaml` as the file name, and paste the following into the file’s body. Click Save.

```yaml
datasetConfigurations:
  - name: "AppendRaw"
    inputs:
      - path: "raw-input"
        dataset: "get-started-MATLAB-dataset"
    outputs:
      - path: "raw-output"
        dataset: "get-started-MATLAB-dataset"
```
4. The other change we need to make is to use `domino.yaml` when starting a new MATLAB workspace. Click the Quick Action button.
5. Click **Continue** under the **Launch a Workspace** option.
6. In the “Launch New Workspace” modal, select MATLAB as the workspace IDE and expand the “Data” section.
7. Click the “Advanced” tab under “Data Configuration”.

4.9. Step 8: Working with Domino Datasets
Launch New Workspace

Workspace Name

yuvalz's MATLAB session

Hardware Tier

Small
1 core · 4 GiB RAM · $0.002/min
< 1 MIN

Workspace Environment

MATLAB R2020a (RDP)

Workspace IDE

MATLAB

Data

Directory: /domino/datasets/yuvalz/get-started-MATLAB/scratch

Data Configuration

Default datasets Advanced

The following Datasets will be available for read-only use in your Workspace.
8. In the “Advanced” tab, click the “Configuration” drop-down list and select the **AppendRaw** configuration from the *domino.yaml* file.
9. Click **Launch** to start the workspace. Startup time may be longer due to Domino’s need to mount the additional dataset.

10. After the workspace launches you’ll be able to find your dataset under `/domino/datasets/raw-input`. Note that
the current dataset snapshot is located where in the folder we specified in the \texttt{domino.yaml} file. The files in this folder are read-only and represent the last snapshot that was saved.

The \texttt{/raw-output} folder will hold the next snapshot of your dataset. If you leave it as-is and never write to it, it will keep the current snapshot of the dataset in place. However, if you write new files and commit them to Domino, they will overwrite and replace the current dataset.
9. Step 8: Working with Domino Datasets

- [Image of a file explorer interface showing directories and files related to Domino datasets]
11. To demonstrate how to overwrite and update a dataset, let’s copy the existing dataset to the output folder, and then add a few new files. In the Command Window of your MATLAB workspace, enter the following line:

```
! cp -rf /domino/datasets/raw-input/* /domino/datasets/raw-output/
```

Note that this line starts with the `!` to denote that this is a system command. You will notice that the input dataset is now present in the output dataset folder.
4.9. Step 8: Working with Domino Datasets
12. Let’s update our code from the previous section and download a new set of weather data files into the `/raw-output` folder. To start, change the directory back to `/mnt` entering by typing the following into the Command Window:

```bash
cd /mnt
```

13. Open the file `downloadToDatasetDir.m` and save it as `downloadToDatasetDir2.m` in the same directory. Make the following changes to the file (different stations):

```matlab
function downloadToDatasetDir2()

baseUrlString = "https://www.ncei.noaa.gov/data/global-historical-climatology-network---daily/access/";
baseWeatherStationId = 'ASN000050cd'; % different stations
datasetFolder = "/domino/datasets/raw-output/";

for counter=1:16
    if counter<10
        weatherStationId = sprintf('%s%s%d', baseWeatherStationId, '0', counter);
    else
        weatherStationId = sprintf('%s%d', baseWeatherStationId, counter);
    end

    urlString = sprintf("%s%s%s", baseUrlString, weatherStationId, ".csv")
    savedFileName = sprintf("%s%s%s", datasetFolder, weatherStationId, ".csv")
    websave(savedFileName, urlString);
end
end
```

14. Run the file by clicking the **Run** button in MATLAB. The files will download to the `/raw-output` directory.

15. Once the files are done downloading, click the **Stop** button in your Domino workspace. Domino will present the notification below. As you can see, Domino will save the output dataset if we stop and commit changes. Click **Stop and Commit** to end the session and save the new dataset snapshot.
16. To verify the files were saved, navigate to the Data section in your Domino project.
17. Click the dataset we just updated. The newly downloaded files will appear in the dataset listing.
This is the data set to test outputting files to.

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR000000012.csv</td>
<td>641 KB</td>
</tr>
<tr>
<td>AR000000013.csv</td>
<td>651 KB</td>
</tr>
<tr>
<td>ASND0096016.csv</td>
<td>4.5 MB</td>
</tr>
<tr>
<td>ASND0096007.csv</td>
<td>3.3 MB</td>
</tr>
<tr>
<td>AR000000016.csv</td>
<td>555 KB</td>
</tr>
<tr>
<td>AR000000012.csv</td>
<td>3.8 MB</td>
</tr>
<tr>
<td>AR000000014.csv</td>
<td>0 B</td>
</tr>
<tr>
<td>ASND0096003.csv</td>
<td>3.2 MB</td>
</tr>
<tr>
<td>AR000000007.csv</td>
<td>506 KB</td>
</tr>
<tr>
<td>AR000000014.csv</td>
<td>569 KB</td>
</tr>
<tr>
<td>ASND0096010.csv</td>
<td>1.4 MB</td>
</tr>
<tr>
<td>ASND0096011.csv</td>
<td>2.8 MB</td>
</tr>
<tr>
<td>AR000000005.csv</td>
<td>620 KB</td>
</tr>
<tr>
<td>ASND0096009.csv</td>
<td>2.3 MB</td>
</tr>
<tr>
<td>ASND0096014.csv</td>
<td>4.2 MB</td>
</tr>
<tr>
<td>ASND0096005.csv</td>
<td>2.9 MB</td>
</tr>
<tr>
<td>AR000000009.csv</td>
<td>574 KB</td>
</tr>
<tr>
<td>AR000000011.csv</td>
<td>553 KB</td>
</tr>
</tbody>
</table>
5.1 Projects

5.1.1 Projects overview

- Overview
- Project files
  - Special files
- Project settings
  - Hardware tier
  - Compute environment
  - Environment variables
  - Access and sharing
- Project stage and status
- Project activity
- Export and Import
  - Setting up export
  - Setting up import
Overview

Work in Domino happens in projects. Projects contain data, code, and environment settings, and the entire project is tracked and revisioned automatically. A new commit is written to a project each time its files are changed by user action, or by the execution of code in the project. Users in Domino can create their own new projects, invite other users to collaborate on them, and export data or results for consumption by other projects.

Project files

Domino manages a collection of files for every project. Files can be added to a project in the following ways:

- uploaded from the Domino web application
- uploaded from the Domino Command Line Interface
- uploaded via the Domino API
- created and edited in the Domino web application
- generated by the execution of code in a workspace or run

Each of these ways of modifying a project’s files creates a new revision of the project. Whenever you start a Run from a Domino project, the files in that project are loaded onto the machine hosting the Run. This machine is known as the executor. The project files are mounted in the /mnt directory, which is in the filesystem root of the executor. Domino keeps track of changes to this directory. When a Run completes, Domino will record changes to /mnt as a new revision of the project. To learn more about how files are loaded into and changed by Runs, read about the Domino service filesystem.

It is also possible to add external Git repositories to projects. Doing so makes the contents of those repositories available in runs and workspaces in the /repos directory of the executor. To learn more, read about Git repositories in Domino.

Special files

There are several special files reserved by Domino. These files control the revisioning behavior, results display, and run comparison features of a project.
README Files

All Domino projects are initialized with a README file, which is accessible through the Overview section of your project. README files are typically used to give readers a general overview of a project. The contents of the README file in your project are written in Markdown, a lightweight and easy-to-read/write markup language.

Tip: If you plan to reference other project files within the README file, make sure you prepend raw/latest to the beginning of a file’s relative path.

For example, consider a file named overview.jpg stored in a folder called images/ in your Domino files, such that the file’s path is images/overview.jpg. To reference this file within the README file, the complete path would be raw/latest/images/overview.jpg.

dominoignore

By default, all projects include a .dominoignore file in the project root folder. This file functions similarly to a .gitignore file, and can be used to exclude certain file patterns from being written to future revisions of the project files. Domino will ignore files that match the specified patterns whenever a new revision is created. This includes revisions created by syncing from your local machine using the CLI, as well as new revisions created by a run or workspace session.

To ignore a file pattern, add it to .dominoignore. Patterns can be filenames, folder names, or UNIX shell regular expressions. Adding a folder will ignore it along with all of its contents. Note that the * symbol in UNIX shell regular expressions is a wildcard, and will match. All paths must be relative to the project root. Take a look at the contents of the default .dominoignore in one of your projects to see commented examples of excluded patterns.

Note

A .git/ directory is always ignored by Domino sync operations, even if that pattern is not listed in .dominoignore.

.dominoreresults

Domino projects include a special file named .dominoreresults. This file controls which files appear in the results dashboard for this project’s runs. It is constructed similarly to .dominoignore, but lists file patterns to include instead of exclude. If no patterns are listed in this file, all files changed by a run will be included in the results dashboard. If any patterns are listed in this file, only files which match those patterns will be included in the results dashboard for this project’s runs.

For example, a .dominoreresults file that contains the following lines will only display the two specified files in the results dashboard.

```
histogram.pdf
output.txt
```

A .dominoreresults file that contains the following lines will display all PDF files in the project, plus any PNG files that are in the results/ folder.

```
*.pdf
results/*.png
```

dominostats.json
Domino’s run comparison feature checks for a file named `dominostats.json` to compare key measurables from individual runs. This file is automatically deleted at the beginning of a run, and will only exist in the project revision produced by a run if a fresh version is written during execution. Read more about runs to learn the details of how this feature works.

**Project settings**

There are several important settings attached to every Domino project. To access project settings, open a project overview and click Settings from the left sidebar.

**Hardware tier**

Hardware tiers describe the compute hardware that will be used for project executors. Executors can either be virtual instances from a cloud services provider, or a machine running in your deployment’s on-premise data center. Local administrators will configure the hardware tiers available for your deployment. Use the Hardware & Environment tab of the project settings to set a specific hardware tier for your project.

You should choose a hardware tier that will provide the performance your workflow needs, bearing in mind the cost of the hardware in cloud deployments, and the impact of your tenancy on local hardware in on-premise deployments. Domino will use this hardware tier for all runs started in the project. When the hardware tier is changed, it will be the default for future runs in the project, although the default can be overridden when starting a run that requires alternate hardware.

**Compute environment**

Compute environments are specifications for containers in which Domino projects will run. Users can create new environments and access public environments shared in their deployment or organization. Whenever a new executor is launched or provisioned for use with a project, Domino loads the compute environment specified in the Hardware & Environment tab of the project settings.

Click to read more about how environments work.
**Environment variables**

Domino pulls environment variables from three sources whenever it loads a run or workspace:

1. User, project, and hardware information. These are stored in variables set by Domino automatically.
2. Environment variables defined in the user profile of the user starting a run.
3. Environment variables defined in the **Hardware & Environment** tab of the project settings.

These environment variables can be used to securely store keys and credentials needed by the project. The names of these variables must start with a letter, and contain only alphanumerics characters and underscores.

**Access and sharing**

Read **Sharing and collaboration** for details on how to grant various types of access to your projects.
Project stage and status

In Domino 3.5+, projects can be labeled with configurable stages that track their progress through a data science life cycle. If your Domino administrator has configured project stages, you will see the current stage of your project displayed in brackets below the project name in the project menu. If you click the project name, a panel will open with a dropdown menu that you can use to change the project stage if you are an owner or contributor on the project.

Tracking your project through the stages used by your team will help your colleagues understand what kind of work is happening in the project and how they can contribute. Changing the stage of a project is an event that will appear in the project’s activity feed.

Projects also have a status, which is indicated by the colored pip next to the project name.

A project’s status can be:

- **Green**
  
  Marks an active and progressing project.

- **Red**
  
  Marks an active project that is blocked.

- **Grey**
  
  Marks a completed project.
By default, new projects are set to a green, active status. To modify a project’s status, click the project name at the top of the project menu to open a panel with options to mark a project as blocked or complete. When you do so, you’ll be given the option to supply a message describing the blocker or end result of the project. Changing the status of a project is an event that will appear in the project’s activity feed with an attached comment thread, so project collaborators can discuss blockers or project conclusions.

Marking a project as blocked

Project owners and contributors can mark a project as blocked. You should mark a project as blocked when you need assistance from colleagues or administrators to make progress. Domino administrators and your project collaborators will receive an email notification when you mark a project as blocked. Some common cases where raising a blocker can help are:

- You need assistance setting up additional tools in a Domino environment
- You need access to new data sources
- You need hardware capabilities not covered by your current hardware tier

The same menu used to mark a project as blocked can be used to unblock the project, which returns it to a green, active status.

Marking a project as complete

Project owners and contributors can mark a project as complete. This allows for final conclusions and products to be recorded in the project’s activity feed, and filters the project out of active project views. On your projects overview, you will find a checkbox to Show completed projects you can use to find projects that have been marked as complete.

The same menu used to mark a project as complete can be used to reopen the project, which returns it to a green, active status. Note that a project marked as complete is still a fully functional Domino project. You can modify its files and start Runs in it, but before doing so you may want to consider reopening the project to indicate that work is continuing.

Project activity

Click Activity in the project menu to open the project’s Activity Feed. On this page you will see the history of activity in the project, including:

- Jobs started
- Workspaces started
• Comments left on Jobs or Workspaces
• Comments left on files
• Project stage changes
• Blockers raised or resolved
• Files created, edited, or deleted in the Domino UI
• Files modified in Workspace sessions
• Models or Apps published
• Scheduled Jobs published or edited

You can use the dropdown menu at the top of the feed to filter out comments, Jobs, or Workspaces. If you check two successfully completed Runs in the feed, you can use the comparison button next to the filter menu to open a Run comparison.

Export and Import

It is possible to import content from one Domino project into another. The importing project may have access to the exporting project’s files, environment variables, or both, depending on configuration. During runs with imported files, each project directory is located at /mnt/<username>/<project name>. When a run or workspace is started, these files are pulled in alongside the current project’s files. Imported directories are read-only.

Note
The path of your project will also change from `/mnt` to `/mnt/username/project name` when you have imported projects. If you have hardcoded any paths in your project code to `/mnt`, you should replace them with paths that use the `$DOMINO_WORKING_DIR` environment variable.

### Setting up export

From the project overview page, in the left sidebar click **Exports** under **Settings**. From this interface you can enable exports for the project’s files and environment variables separately, or export the project files as a Python or R package. If none of these are enabled, other projects will not be able to import anything from this project.

<table>
<thead>
<tr>
<th>Export</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>If checked, all projects which import this project will be able to access</td>
</tr>
<tr>
<td>variables</td>
<td>this project's environment variables.</td>
</tr>
<tr>
<td>Files</td>
<td>If checked, all projects which import this project will be able to access</td>
</tr>
<tr>
<td></td>
<td>this project's files.</td>
</tr>
<tr>
<td>Code Package</td>
<td>If selected, an R or Python package will be available to projects which</td>
</tr>
<tr>
<td></td>
<td>import this one.</td>
</tr>
</tbody>
</table>

By default, projects will make their latest revision available for export when configured. You can also make revisions produced by specific runs available for import by tagging those runs with `release`. From the runs page of a project, select the runs you want to export by clicking the checkbox next to them, then click the tag button at the top of the list. Enter the exact string `release` to mark the revision created by the selected runs as available for export.
Setting up import

From the Files page of the project you want to import into, click the Other Projects tab. Enter the path to the project you want to import, with the format `<username>/<project-name>`. The following conditions must be true for you to successfully import a project:

- You must have Project Importer, Contributor, or Owner permissions on the project.
- The project must be configured for export.

After adding a project that exports files, you can choose which revision of the project files you want to import with the Release menu.

5.1.2 Projects Portfolio

The Projects Portfolio allows you to track the status of projects you have access to, including:

- projects you own
- projects you have been added to as a collaborator
- or if you are a system administrator, all projects across Domino

To open the Projects Portfolio, click Control Center in the Switch To menu.
You will find the Projects Portfolio as an option in the Control Center main menu.

The Control Center interface above shows the Projects Portfolio with the following important elements:

1. This is the main menu option that opens the Projects Portfolio
2. This button filters the table of projects to show only projects that are marked by users as blocked
3. These tabs filter the table of projects by active or complete status
4. These buttons filter the table of projects by project stage
5. This menu selects which columns of data to display in the projects table

This interface allows you to quickly digest the state of work in your projects. To maximize the usefulness of this tool, be sure to understand how administrators can configure meaningful project stages for their teams, and read about how users set project stages, raise and resolve blockers, and change project status.

5.1.3 Project Goals in Domino 4+

Overview

In Domino 4.0+ you can add goals to projects. Goals represent outcomes or subtasks within the project. Contributors to the project can link files, Workspace sessions, Jobs, and Apps to goals, which show up on the goal card in the project overview.
This provides a way to track all work related to a specific goal in the project, and can make navigating large and busy projects easier.

Creating goals

From the project overview, click the Goals tab, then click + Add Goals. Provide a title and description, then click Save.
Managing goals

You can always see the current status of a goal by returning to the Goals tab on the project overview. Click the menu button in the top right of a goal card to see options for editing, deleting, or changing the completion status of a goal.

Linking work to goals

You can link Workspace sessions or Jobs to a goal by checking the desired entries in their respective dashboards and clicking the Link to goal button.
You can link an App to a goal by clicking **Link to goal** on the App settings tab.

You can link Model APIs to goals by clicking **Link to Goal** in the **Actions** column of the Model APIs overview.

You can also link files to goals by clicking **Link to Goal** in the upper right of the file view.
5.1.4 Jira Integration in Domino

**Attention:** To use this feature:

- Your Domino deployment must be configured to use Jira.
- A Jira administrator must configure this integration within Jira.
- Your Jira account must have the necessary privileges to modify tickets within Jira. Otherwise, you’ll encounter errors within Domino if you try to use this feature.

To learn more about how to configure your Domino deployment to use Jira, please read: [Jira Admin Configuration](#).

**Overview**

With Domino’s Jira integration, you can perform common Jira actions from your Domino project.

**Connecting Domino and Jira**

To use this feature, you’ll first need to connect your Domino and Jira accounts, and then authorize your Domino account to take action in Jira on your behalf.

1. Navigate to the **Overview** section in Domino.
2. Click on the **Manage** tab.
3. Click on the **Add Credentials** button. You’ll be redirected to Jira’s authentication page. Enter your Jira credentials and authorize read and write access for Domino.

The connection between your Domino and Jira accounts will persist until you clear your Domino-related browser cookies.
Linking a Jira Ticket to Your Domino Project

Once you’ve connected your Domino and Jira accounts, you’ll have the option to link a Jira ticket to your Domino project. Only a single Jira ticket can be linked to a Domino project at any given time.

To link a Jira ticket to the Domino project, you can search by its Jira key or title text, browse among issues assigned to you, or browse issues in a given Jira project.

Linking a Jira ticket to your Domino project will automatically pull in the ticket’s metadata (description, comments, etc.) into Domino. Any changes made to a Jira ticket from within Domino will also be reflected in Jira.

If a Jira ticket is modified from within Jira after it has been linked to a Domino project, you’ll need to sync the changes using the **Sync** button in the **Manage** tab in Domino.
Jira Subtasks or Child Issues Become Domino Goals

Recall that a Domino project can be linked to a single Jira ticket. If that ticket has child issues or Subtasks associated with it they will automatically become Domino goals when linked in your project. The Domino-Jira linkage is bidirectional – you can add, edit, or delete goals from within Domino and have those changes reflected in Jira (and vice versa).

If your Jira workflow commonly utilizes Epics with child issues like Tasks, Stories, Bugs, etc., we recommend you link the Epic issue to your Domino project. The child issues (Tasks, etc.) within the linked Epic will automatically become Goals in the Domino project.

If your Jira workflow does not typically utilize Epics, or if a single Epic usually maps to multiple data science projects, then it would be best to link a Jira ticket of type Task or Story to your Domino project. If the Task contains Subtasks then the Subtasks automatically become Goals in the Domino project.

The following operations on a Domino Goal will be reflected in the corresponding Jira ticket, and vice versa:

- Creating a new goal
- Editing the goal title or description
- Deleting an existing goal
- Changing status

To learn more about goals in Domino, please read: Project Goals in Domino 4.

Unlinking a Jira Ticket from Your Domino Project

1. Navigate to the Overview section in Domino.
2. Click on the Manage tab.
3. Click on the ... button and click Unlink Ticket to unlink the Jira ticket.

After a ticket has been unlinked from your Domino project, you can decide to keep or discard any associated goals in your project. You can also link the ticket to another Domino project, if desired.

**Attention:** Be sure to discard any goals that were created in your project as a result of unintentionally linking the wrong Jira ticket to your Domino project. If you do not discard these goals, they will be kept and added to the set of goals of a subsequent Jira ticket.

Limitations

- Currently, images, Markdown, and any other Jira-compliant formatting are rendered as plain-text in Domino.

5.1. Projects
5.1.5 Uploading files to Domino using your browser

When you want to move an existing project into Domino, you have a few options. For smaller projects - less than 550 mb - you can drag and drop files into Domino.

If your project is larger than 550 mb, check out this article on how to use our command line interface or a Domino run to import larger data sets and projects.

5.1.6 Forking and merging projects

Forking a project copies all of its files into a new, separate project, allowing for independent development and experimentation. Changes in the forked project can later be reviewed and merged back into the original project.

**Forking**

To fork a project, open the project overview and click Fork. Enter a name for the fork when prompted. You must be the project owner, or have access to the project as a contributor or results consumer in order to fork it. You can learn more about project roles and access control here.

Copied to the newly forked project:

1. All files
2. Revision history of all files
3. Environment variables

These things are **not** copied to the newly forked project:

1. Run history
2. Project settings, including collaborators and compute environment
3. Launchers
4. Discussion history
Projects that have been forked, or were created by forking another project, will link to related forks on the project overview page.

**Merging**

Once you’ve made some changes to the new fork, you can initiate a merge by clicking Request Review on the project overview page. You must be the project owner, or have access to the project as a contributor in order to request a merge review.
You will be prompted to submit a review request, in which you can review the changes and describe their effects with a message. Once submitted, contributors to the main-line project are notified. The merge will occur when a contributor accepts the review, and a new revision of the main-line project will be written with the forked changes merged in.

To view a history of Review Requests, including the status of current requests, select **Reviews** from the left-hand menu.

### 5.1.7 Search

#### Using Search

Domino’s search feature is a comprehensive tool for locating specific files or bits of text across your entire deployment. To use the search feature, click **Search** in the left navigation bar. Type in the term you’re looking for. The search panel will automatically update with results organized by source, including project names, files, runs, comments, and more.
When Domino searches files, it will return results found in both filenames and file contents. However, Domino only indexes the latest revisions of files, so the search results will not contain occurrences of your search term from previous versions.

**Security**

Domino search respects the collaboration and privacy settings for projects. If you do not have read access to a project, then that project will never appear in your search results.

**Advanced Search Options**

In addition to the four search tabs, you can use the following commands in your queries to target what you search:

- `project.tag=`
- `project.tag.approved=`
- `project.description=`
- `project.name=`
- `project.user=`

For example:
5.1.8 Sharing and collaboration

- Overview
- Visibility settings
- Managing collaborators
- Access levels
- Files permissions
- Runs permissions
- Publishing permissions
- Settings permissions
- Dataset permissions
- Import permissions

Overview

Domino makes it easy to collaborate on projects and share project outputs.

There are two things that affect who has access to your project:

- the project’s visibility settings
Visibility settings

You can change your project’s visibility by going to the **Access & Sharing** tab of the project’s settings page.

There are three different visibility options:

**Public**

- Anyone can view your files and runs, even if they don’t have a Domino account.
- If file exports are enabled, anyone can import your project files.
- Only explicitly added collaborators can modify files, start runs, or import environment variables, unless you check the *allow runs by anonymous users* box described below.
Searchable

- Anyone will be able to see that this project exists, and see its name and description in search results, but only explicitly added collaborators can see the project’s contents.

Private

- Only collaborators can view this project or discover its existence through search results.

If your project is publicly visible, there is an additional option to allow runs by anonymous users. This will allow users to start runs even if they don’t have a Domino account. Runs started by anonymous users will show up as being started by the project owner.

Allowing anyone to run your code can be dangerous. Be careful granting this level of access, and make sure to think through any information you may be revealing, such as environment variables you have set in your project that contain bearer tokens, API keys, or passwords.

Managing collaborators

To grant other users access to a project, you can add them as collaborators. To add collaborators, you must be a Contributor to the project, or the project Owner.

Click Settings from the project menu, then click the Access & Sharing tab and scroll down to the Collaborators and permissions panel. You can add new collaborators by their username or email address. If you supply an email address belonging to a Domino user, that user will be invited to join the project as a collaborator. If you supply an email address that is not associated with an existing Domino user, an email will be sent to that address inviting them to join Domino.

The collaborators tab is also where you specify how each collaborator should be notified when runs complete. This can be a powerful tool to keep your collaborators in sync on the work that each person is doing.
Access levels

The owner of a project can set different access levels for collaborators from the Collaborators and permissions panel. The basic capabilities of the various types of project collaborators are as follows:

• **Contributors**

  Can read and write project files, and start runs. On the Settings page, Contributors can read and write project environment variables, and they can invite new collaborators. Contributors can also change hardware tier and environments.

• **Results Consumers**

  Can only read files and access published apps.

• **Launcher Users**

  Can only view and run Launchers, and see the launcher runs results. They cannot view project files.

• **Project Importers**

  Can import the project, but otherwise cannot access it.

• **Owners**

  Are the only users who can archive a project, change the owner, change collaborator types, set automatic workspace shutdown times, or change the project default hardware tier or environment.

For complete, itemized project permissions set on each type of collaborator, consult the tables below:
### Files permissions

<table>
<thead>
<tr>
<th>Permission</th>
<th>Project Importer</th>
<th>Launcher User</th>
<th>Results Consumer</th>
<th>Contributor</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read files</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Write files</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Add external Git repository</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

### Runs permissions

<table>
<thead>
<tr>
<th>Permission</th>
<th>Project Importer</th>
<th>Launcher User</th>
<th>Results Consumer</th>
<th>Contributor</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Run</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>View Completed Runs</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Start Workspace</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>View Workspaces</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Schedule Run</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>View Scheduled Runs</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

### Publishing permissions

<table>
<thead>
<tr>
<th>Permission</th>
<th>Project Importer</th>
<th>Launcher User</th>
<th>Results Consumer</th>
<th>Contributor</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Launcher</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>View App</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Publish App</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Unpublish App</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Invite users to App</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Change App hardware tier</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Publish Model API</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Create Launcher</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
Settings permissions

<table>
<thead>
<tr>
<th>Permission</th>
<th>Project Importer</th>
<th>Launcher User</th>
<th>Results Consumer</th>
<th>Contributor</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set environment variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Invite collaborator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change project stage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Raise a blocker</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set project status as complete</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Manage collaborator permissions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Change visibility setting</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Change default environment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Change default hardware tier</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Change project name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Handle merge request</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Transfer project ownership</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Archive project</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Dataset permissions

<table>
<thead>
<tr>
<th>Permission</th>
<th>Project Importer</th>
<th>Launcher User</th>
<th>Results Consumer</th>
<th>Contributor</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mount Dataset from project for read-only use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write new Snapshot to Dataset in project</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Import permissions

<table>
<thead>
<tr>
<th>Permission</th>
<th>Project Importer</th>
<th>Launcher User</th>
<th>Results Consumer</th>
<th>Contributor</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import project</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.1.9 Comments

Domino makes it easy to collaborate and coordinate with others through commenting.

Comments can be posted on:

- Jobs
- Workspaces
- Results
- Files
- Reviews
- Run comparisons
Domino supports Mathjax and Markdown for richly formatted comments. You can also mention other users to tag and notify your collaborators.

Discussion

@sujaym2 the F1 scores look odd.
Are you sure you used the following equation?

$$F1 = \frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall}^2 + \text{precision}^2}$$

Other metrics to look at:

* ROC AUC
* Weighted F1 score

Here's the syntax for weighted F1 score in sklearn:

```python
sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None, zero_division='warn')
```

Preview

@sujaym2 the F1 scores look odd.
Are you sure you used the following equation?

$$F1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}}$$

Other metrics to look at:

* ROC AUC
* Weighted F1 score

Here's the syntax for weighted F1 score in sklearn:

```python
sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None, zero_division='warn')
```
**MathJax formatting**

Discussion

$$F1=\frac{2}{\frac{1}{\text{recall}}+\frac{1}{\text{precision}}}$$

Preview

$$F1 = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}}$$

To produce clear and professional-looking symbols and equations, Domino uses MathJax. MathJax automatically formats the mathematical symbols and equations that you enter in comments using LaTeX notation.

Many resources for learning and using MathJax are available online, including the official MathJax Documentation.

For quick reference and examples of commonly used MathJax notation, we recommend this tutorial available on the Mathematics meta stack exchange.
Markdown formatting

Discussion

```markdown
# my header

* ROC AUC

```

```
```python
sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None, zero_division='warn')
```

---

Preview

```
my header

- ROC AUC
- Weighted F1 score

sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None, zero_division='warn')
```

---

Comment Markdown and Mathjax are supported. You can also mention people.

Domino also supports Markdown in comments. Markdown is a lightweight markup language used to add formatting elements to plaintext text documents.

They are especially useful in Domino for formatting longer comments or creating README files.

Read about the basics of Markdown syntax here.
Mentioning collaborators

Discussion

@sujaym2 the F1 scores look odd.
Are you sure you used the following equation?

You can mention your collaborators in a comment with @username. Users mentioned in a comment will receive an email notification.

5.1.10 Domino service filesystem

Overview

This article describes the filesystem structure you will find in Domino Runs.

The filesystem root (/) contains the following directories.

```
/  
  |__ bin
  |__ boot
  |__ dev
  |__ domino # contains datasets directory
  |__ etc
  |__ home
  |__ lib
  |__ lib32
  |__ lib64
  |__ media
  |__ mnt # contains working directory
  |__ opt
  |__ proc
  |__ root
  |__ run
  |__ sbin
  |__ scripts
  |__ srv
  |__ sys
  |__ tmp
  |__ usr
  |__ var
```
**Domino working directory**

When you start a Run from a Domino project, your project files and some additional special files and directories are loaded into the *Domino working directory*. There are two different paths where you may find this directory, depending on how your project is configured:

1. By default, your working directory will just be /mnt. The folders and files from your project will be in that directory, along with the special files and folders described below.

2. If your project is set up to *import* another project, your working directory will instead be /mnt/ <project-owner-username>/<project-name>.

Note that Domino sets the `DOMINO_WORKING_DIR` special environment variable for all Runs, and it will always contain the path to your working directory.

Inside your working directory you will find your project files. Additionally, the following folders and files have special significance in the working directory:

```
DOMINO_WORKING_DIR/
|-- ipynb_checkpoints # folder with your auto-saved Jupyter states
|-- results # folder with your generated results
   |-- stdout.txt # tail of the console output from your Run
|-- requirements.txt # add this file to specify python package dependencies
|-- .dominoresults # controls which files are rendered as results
|-- .dominoignore # add file patterns here for Domino to ignore
|-- .dominokeep # add this to an empty folder to make Domino keep it
|-- dominostats.json # values written here are shown in the Jobs dashboard
|-- email.html # used to format your own notification emails
|-- .noLock # create this file to remediate "too many open files"
|-- app.sh # put your app-launching code here for Domino Apps
|-- domino.log # in local CLI projects only; contains CLI logs
|-- .domino.vmoptions # in local CLI projects only; contains proxy settings
```

Learn more about:

- `.dominoresults`
- `requirements.txt`
- `.dominoignore`
- `.noLock`
- `email.html`
- `dominostats.json`
- `.domino.vmoptions`

### 5.1.11 Comparing file revisions

Domino can show you a rich comparison of the differences between two revisions of a file in your Domino project. To view this comparison, open the file from the Files page and click *Compare Revisions*.
Domino will open the file comparison tool and display two menus above its contents. The left menu, labeled Base, sets the starting version of the file for your comparison. The other menu, labeled Target, lets you select another version of the file to compare to the base version.

When you are viewing the Files page for your project, you can select between all revisions for the project. When you are viewing an individual file, the revisions dropdown is limited to only those revisions where a change was made to that file.

### 5.1.12 Reverting projects and files

#### Overview

Domino allows you to easily revert entire projects or individual files back to previous versions at any time. This is intended to encourage experimentation and creative approaches to problem-solving by eliminating the overhead cost of recovering earlier, functional versions if experiments don’t pan out.
Reverting a Project

To revert to an earlier version of a project, open the project in Domino, then click Files from the project menu. Above the table of files you’ll see a dropdown menu that controls which revision of the project files is shown. The current revision is at the top, and older revisions appear below it chronologically. Click on an older revision to view it.

While you’re viewing an older revision, the files page will have a yellow background, and a callout will appear below the dropdown menu, alerting you to the fact that this is an older state of the project, and providing a link to switch back to the latest.

You’ll also see a Revert Project button next to the dropdown menu. Clicking this button will revert your project to the state shown in the revision you’re currently viewing. This will result in a new revision being applied on top of the existing project history. The revision you reverted from is not lost.
Reverting a File

Domino allows you to revert individual files as well. This can be useful in cases where you wish to revert to an earlier version of a file while still preserving changes made to other files in the project.

To revert to an earlier version of a file, open the project containing the file in Domino, then click Files from the project menu. Click on the name of the file you want to revert to view the contents of the file. You’ll see a dropdown menu above the file contents that can be used to select which revision of the file you want to view.

While viewing an older revision of the file, you’ll see a Revert File button next to the dropdown. Clicking this button will write a new revision of the project that changes this file to match the state you want to revert to. Just like with reverting a whole project, the revision you’re reverting from is not lost.
**5.1.13 Git-based Projects with CodeSync**

**Note:** We recommend using Git-based projects in Domino to take advantage of our CodeSync technology if:

- you have at least a beginner’s proficiency with Git,
- you have experience using hosted version control (GitHub, Bitbucket, etc.), and
- you frequently collaborate with multiple data scientists on projects.

To learn more about common workflows using Git and branching, we recommend the following resources: [Git Feature Branch Workflow](#).

If you’re not familiar with Git or hosted version control, or if you’re primarily looking to reproduce data science research, then a project that uses the Domino File System (DFS) may be a better fit for you.

Git-based projects with CodeSync are a beta feature. If you’d like to provide feedback on Git-based projects, please file a ticket.

**Overview**

Git-based projects provide a full CodeSync experience for your code by using Git and a Git service provider of your choice. Integrated CodeSync technology ensures that all of the common Git workflows, like committing, pushing changes, and more, are available to you natively within workspaces launched in Git-based projects. This makes it easy for you to engage in version controlled, code-based collaboration with fellow project team members, all from within Domino. Git-based projects also organize your projects’ assets as either Code, Data, or Artifacts, an organizational structure intended to support common data science workflows.
Create a Git-based project

If you plan on using a private Git repository to store your code, then you’ll first need to add the corresponding Git credentials in your Domino account settings prior to creating your project. After adding credentials, you’ll be able to easily create a Git-based project in Domino, thereby enabling CodeSync experience.

If you plan on using a public Git repository to store your code, then you won’t need to add any Git credentials.

To create a Git-based project:

1. Click on Projects in the Domino sidebar menu and then click the New Project button. A project creation modal will appear.
2. Enter a name for your project.
3. Set your project’s visibility.
4. Under “Code Repository”, select “Git (beta)”.
5. Under “Git Repository”, enter the URL of the Git repository that you’ll be using for your project.

**Attention:** If the repository you’re using to store your code contains one or more files exceeding 2 GB in size, then Domino will create your Git-based project but will not use the repository for your project. Please note that your Git service provider may also impose size limits on individual files.

You can use the following tool to check the total size of a Git repository, as well as the size of individual files within the repository: git sizer.

6. Under “Git Credentials”, select the credentials associated with the Git repository your project will use. If the repository is public, then you won’t need to enter any credentials.
7. Under “Git Service Provider”, select your Git service provider.
8. Click Create Project.
Code, Data, & Artifacts

In Domino, the Domino File System (DFS) is the traditional way of storing a project’s assets. DFS-based projects organize all of your project’s assets as either Data or Files. Git-based projects, however, organize your project’s assets as either Code, Data, or Artifacts, and apply CodeSync experience to Code assets.

**Code** – This section of your Git-based project organizes and lists all of the Git-based repositories used to store your project’s code, as well as any additional imported repositories. For more information, see *Git repositories in Domino*. Files within any of these repositories can be accessed from within a Domino workspace via CodeSync technology.

The common Git workflows, like committing, pushing, pulling, and more, are available to you when interacting with your code from within a Domino workspace. For more information, see *Using Git in your workspace*.

**Data** – Similar to DFS projects, this section of your Git-based project organizes and lists all data sources used in your project, including Domino datasets, external data volumes, and dataset scratch spaces. For more information on how to use data with your project, please refer to the *Domino datasets documentation*.

**Artifacts** – Git-based projects introduce “Artifacts”. Artifacts are typically results or products from your research and analysis, like plots, charts, serialized models, and more. You can organize these outputs in this section, as well as import artifacts from other projects.
Navigating the workspace directory structure

Git-based projects with CodeSync use a different directory structure in workspaces than DFS-based projects. The directory structure is shown below.

The default working directory for your code is `/mnt/code`.

```
/mnt
  /code  # Git repository and default working directory
  /data
    # Project Datasets
    /{dataset-name}  # latest version of dataset
    /{dataset-name-out}
    # Dataset Scratch Space
    /scratch
  # Project Artifacts
  /artifacts
  # External mounted volumes
```

(continues on next page)
/external-volume-name

/imported

  # Imported Git Repos
  /code

  /{imported-repo-name}

  /data

  # Mounted Shared Datasets
  /{shared-dataset-name} # contains contents of latest snapshot unless otherwise specified by yaml

  # Imported Project Artifacts
  /artifacts

  /{imported-project-name}

Attention: The default working directory for your project’s code is /mnt/code. In addition, external data volumes are only displayed in your directory structure if you’ve mounted external volumes to your Git-based project, and the imported/ directory is only displayed if you’ve imported additional Git repositories into your Git-based project.

Running jobs

Warning: If you run a Job in a Git-based project, CodeSync ensures that only artifacts will be automatically synced and saved to the Domino File System (DFS). Code, on the other hand, will not be automatically synced / pushed to the Git repository being used for the Git-based project. This is intentional and intended to support the “Code”, “Data”, and “Artifacts” workflow. To learn more, see running jobs.

Working with artifacts in your workspace
Attention: All files in Artifacts are saved exclusively to the Domino File System (DFS). If you do not want to save a particular asset to the Domino File System, we recommend that you do not save it as an artifact. To learn more, see Syncing your work to Domino.

Artifacts are results from your research, like plots, charts, serialized models, and more. In Domino, you can save these results in the Artifacts section of your project.

Saving artifacts and pushing changes

1. Click on the File Changes option in the sidebar of your workspace.
2. Under “Artifacts”, view changes by expanding “File Changes”.
3. Enter a commit message.
4. Click Sync to Domino. Domino will save your artifacts to the Domino File System (DFS).

Pulling changes

To pull the latest artifacts (from the Domino File System) into your workspace:

1. Click the File Changes option in the sidebar menu of your workspace.
2. Under the “Artifacts” section, click Pull. Domino will pull the latest changes into your workspace.
5.1.14 Advanced project settings

Project dependencies

- Overview
  - How it works
  - Setup
- Troubleshooting

Overview

You can configure projects to import files and environment variables from other projects. This allows you to use your team’s existing work as reusable building blocks, and avoid unnecessary repetition.

For example:

- A canonical dataset used by multiple projects can be managed in a single place.
• Your code can reside in a separate project from your data. No need to duplicate large datasets within multiple projects.

• An external data source requiring login credentials (such as a database) may be securely represented via environment variables in a single project, and then used by many projects.

• Results from one model (such as trained model files, or R workspaces) can be imported and used by multiple different downstream projects.

• If a project’s files are organized as an R or Python package, then you can configure other projects to automatically install them at runtime.

---

**How it works**

The first step is to configure the exporting project. Projects can export files and environment variables.

Other projects can import content from projects that are configured for export. After you’ve set up import, the content from the exporting projects is accessible when you run code in the importing project.

During runs with imported files, each project directory is located at `/mnt/<username>/<project name>`, where `<username>` is the owner of that particular project. Imported directories are read-only.

The path of your main project will also change from `/mnt` to `/mnt/<username>/<project name>`. If you have hardcoded any paths in your projects to `/mnt`, we recommend replacing the hardcoded paths to using the `$DOMINO_WORKING_DIR` environment variable. This will ensure the correct path regardless of whether other projects are imported. See the support article on Domino Environment Variables for more information.

**Setup**

1. First, you need to set up the projects you want to export from. You need to have Owner, Collaborator, or Project Importer access to the projects to set them up for export.

2. To set up export, open the project in Domino and click Settings from the project menu, then open the Exports tab. In the panel on that tab, click the checkbox for files or environment variables to make those types of content available to other projects. To export as a package, select the appropriate language from the Code Package dropdown.
3. Next, open the project into which you want to import content. Click **Files** from the project menu, then open the **Other Projects** tab. Add projects by filling in the project name field, then clicking **Import**. You’ll see projects currently being imported listed in a table below.

Only the files from the directly imported project will be viewable when you import. For example if project **A** is imported into project **B**, and then your project imports **B**, only the contents of **B** will be accessible to your project.

---

**Troubleshooting**

**Running python scripts from imported project**

When running a Python script from an imported project you may encounter the error message:

```
FileNotFoundError: [Error 2] No such file or directory:
```
When a python script runs an import, it executes that code with the current working directory, so if you have a relative path in the imported file, it will try to find the file in the current folder and fail. In this case, you can update your imported script to use an absolute path based on the current path of the imported file using `os.path`, eg.

```python
import os
file_name = os.path.join(os.path.dirname(__file__), 'your_referenced_file.dat')
```

**Project tags**

Project tags are an easy way to add freeform metadata to a project. Tags help colleagues and consumers organize and find projects that interest them in Domino. Tags can be used to describe the subject matter explored by a project, the packages and libraries it uses, or the source of the data within.

**Tagging a Project**

Tags can be added, deleted and modified from a project’s overview page by clicking the ‘+’ button above the description.

While you can create a tag with whatever content you’d like, tags indicated in green have been marked as [approved](#) by your Domino admin or [librarian](#) to help reduce duplicate tags.

**Managing Tags**

Domino admins and [librarians](#) can manage the tags in a Domino deployment. From the left navigation menu you can click [Tags](#) to open the tags interface.

From this screen you can add, delete, and edit existing tags. You can also merge tags. You can mark a tag as approved which will make it appear green to all of your users, and signal that its use is encouraged.

**Renaming a project**

Changing the name of a project in Domino is a quick and simple process. However, there are repercussions involved in changing a project’s name. For example, when a project is renamed, its endpoints are automatically unpublished, and they will all have new URLs when they are re-published. Additionally, any local copies will have to be cloned again, since the project metadata will be pointed at the old project name. Since renaming a project cannot be undone, make sure you are aware of and have prepared for these events.

To change a project’s name, go to the project’s [Setting](#) page under the [Access & Sharing](#) tab.
Domino will check to ensure the new name is not currently in use by another project. When you have finished entering the new project name, Domino will show you a warning screen. Be sure to read this warning thoroughly. If any of these events are unacceptable to you, click **Cancel**.

If, however, you still wish to rename your project, click **Rename Project** to complete the process. Domino will rename your project, and you may continue working on it immediately.

**Setting up your project to ignore some files**

**Using .dominoignore**

Domino lets you “ignore” certain files from your project, excluding them from the `domino sync` operation. This includes syncing from your local machine (using the CLI or R package), as well as syncing at the end of a run or during a workspace session.

To ignore a file, add it to the file named `.dominoignore` that is automatically created at the root of every new project. Adding a folder will ignore it along with its contents. The `*` symbol may be used as a wildcard to match files of a similar pattern. All paths must be relative to the project root.

Note: a `.git/` directory is always ignored by the sync operation, even if not listed in `.dominoignore`.

**Using isolated results branches**

You can configure your project to save results to an isolated branch. You can access these results from the UI as well as CLI, but they will not be downloaded by domino sync, nor will they be used as input to future runs unless you explicitly specify so. More information is available in this help article.

**Using project importing**

In certain situations, neither `.dominoignore` nor results branches are sufficient approaches. For example, suppose you are developing a recurring data cleaning task that produces large files used by other projects. You want to sync these results back to the mainline project at the end of the run, but don’t want to sync them to your local machine during development.

This can be accomplished by putting your code in one project and importing it into a different project in which the large-output runs will occur. This way you can sync with the “code project” and avoid downloading the results stored in the “data output project”. You can even configure the code project to be imported as a package, and it will be automatically installed and accessible at runtime.
Uploading files larger than 550MB

Out of the box, Domino limits the size of a file that can be uploaded through the Web UI to 550MB.

*Note: If you are using an on premises or VPC deployment of Domino, this size limit is configurable by your administrator thus, your limit may be different.*

There are two easy ways to upload files larger than 550MB.

1) Using the CLI

If your files are on your computer, the easiest method is to use the command line (CLI) client. This will allow you to quickly sync any project that you have on your local machine with the project that you have on Domino.

If you have not installed the CLI client yet, you can do so by following these instructions.

Once the CLI client is installed and you have authenticated (per the installation instructions), navigate to the folder on your local machine that contains the file(s) you want to be uploaded to your Domino project.

We’ll associate your Domino project with this folder so that you can easily sync files to and from it.

Type `domino restore` to identify which Domino project to associate with this folder. It will download any project files that you have in Domino for that project.

Finally, type `domino sync`, which will sync the files in your local directory to the project in Domino, thereby uploading any files over 550MB.

2) Using `wget` or a workspace session within Domino

If your file is on the web or accessible via a URL, you can have Domino download the file to your project. You can do this through a shell script or an workspace session.

We’ll walk through using a shell script.

Create a new `.sh` file in your Domino file tab and run it in Domino. See example, below.

```bash
wget "https://transparencyreport.storage.googleapis.com/google-websearch-copyright-removals.zip"
```
This will import the files downloaded through wget. They will appear in the Files tab. You only need to do this once to bring the files into your project.

Note: if you are working with large data, e.g. 10s of GB, you may see longer “Preparing” times. During the “Preparing” stage of the run, your files are being copied to the executor.

Larger amounts of data take longer. For large data projects, Domino natively integrates with Hadoop, Spark and other big data platforms. Please contact support@dominodatalab.com for more information.

Exporting files as a Python or R package

If you organize the files in a project as an installable package, then you can choose to export it as such. When another project import this project, Domino will automatically install the package at runtime, making it available to your code.

To export as a package, configure your project to export files, and select the appropriate language under “code package”.

You must follow the language-specific pattern required for any package, described below.

R

For an in-depth guide to writing R extensions, please reference the official manual.

In summary, each R package requires:

- A directory called R/ containing code files.
- A directory called man/ containing documentation files
- A file named DESCRIPTION, with each line following the pattern <key>: <value>. Required keys include:

  Package
  Version (e.g. 0.1)
  Title
  Description
  Author
Maintainer (a name followed by an email address in angle brackets, e.g. Sample Maintainer <maintainer@example.com>)

License

- A file named NAMESPACE that describes the namespace of the package. If you aren’t sure what to put here, exportPattern( "." ) can work in many cases.

Python

For an in-depth guide, please reference this documentation.

In summary, each Python package requires:

- A setup.py file. This must contain a setup() function (imported from setuptools), with arguments as described here.
- A folder containing your Python modules and packages. Usually this is given the same name as the overall package.
- It’s also a good idea to include some sort of README file.

Transferring project ownership

You can transfer any project you own to another user or organization. For example, a project that started out as a private scratch workspace can be shared with colleagues by transferring ownership to your organization.

To do this, (1) go to the project’s settings page, (2) choose the “Archive Project or Transfer Ownership” tab, and (3) click the “Change Project Owner” on the right side. A pop up will show where you can enter the new owner’s user or organization name.

After ownership is transferred, the new owner will receive an email notifying them of the change. Organizations do not receive emails.

API Endpoints and Ownership Transfer
If you were using an API endpoint in the original project, it will stop serving requests after ownership transfer. The new owner can re-publish it for further use, but note that the endpoint URL will change and your API key will no longer work (unless you are given Contributor access). So be sure to re-point any external system that calls it, and to coordinate around the downtime.

To avoid downtime, you can instead follow these steps:

1. Have the new owner fork the project (you will need to give them access as a Contributor or Results Consumer).
2. Re-publish the endpoint in the new project.
3. Point your system to the new URL.
4. Shut down the original endpoint (and archive the original project, if you like).

5.2 Domino runs

A Run in Domino is a code execution task assigned to a Domino executor. Runs include jobs, workspace sessions, and web apps. Domino makes it easy to schedule runs and to compare results or track diagnostic statistics.

5.2.1 Jobs

- Overview
- Starting a Job in Domino
  - Jobs dashboard
  - Project files page
  - Domino CLI
  - Scheduled Job
  - Launchers
  - API
  - Quick Action menu
- Managing Jobs with the Jobs dashboard
  - Viewing Job details and results
  - Tagging Jobs
  - Archiving Jobs
  - Stopping Jobs
  - Comparing Jobs
  - Customizing your Jobs dashboard columns
Overview

Jobs are a type of Domino Run where an executor machine is assigned to execute a specified command in its OS shell. You can use Jobs to run Python, R, or Bash scripts from your Project.

Each Job keeps a snapshot of all your files, including any results the code creates or modifies. This includes charts, tables, data files, and serialized model data. All assets are versioned and can be compared across different Jobs. Read about the Domino service filesystem to learn more about the filesystem environment your Jobs will run in.

When you start a Job, Domino launches a new environment for your code on the target executor. You can start multiple concurrent Jobs. Each gets its own container environment, so you can try multiple parameters and techniques in parallel. You can schedule recurring Jobs and set up notifications, including customized reports.

Starting a Job in Domino

There are many ways to start Jobs in Domino.

Jobs dashboard

Click Jobs from the Project menu to open the Jobs dashboard. This dashboard shows all Jobs from the history of your project, sorted into tabs by Job status.

Click Run at the top to start a new Job.
After clicking **Run**, you will see a Start Run window with options to configure the Job. You need to enter the filename of the script you want Domino to execute, followed by any arguments the script takes.
You can start a Job directly from the project Files page. Find the file you want to run and click the gear icon to the right of its entry in the files list, then click Run.

You will then see a dialog you can use to configure the Run. This window has additional options to set up recurring scheduled Jobs and select different hardware tiers.

- Title

Project files page
In this field, enter a name for the Job. This name will be the label for the Job in the Jobs dashboard.

- **Parameters**

  If there are any arguments or parameters you’d like to pass to your script, enter them here.

- **Hardware tier**

  This dropdown list lets you set the hardware tier used by the Job.

- **Schedule to repeat**

  In this section, specify if you want the run to execute only once, or to repeat on a schedule.

- **Publish after complete**

  Check this if you want Domino to republish a Model API once the Job completes successfully.

---

**Domino CLI**

From your workstation, you can start Jobs through the Domino CLI. Follow these instructions to install the CLI on your system.

Once you have logged in and opened a project, you can start Jobs with:

```
domino run <filename>
```

**Examples**

```
domino run calculateStats.r
domino run runModel.m model1 .05
domino run --wait runModel.m model1 .05
domino run --direct "pip freeze | grep pandas"
```
Scheduled Job

Domino allows you to schedule Jobs in advance, and set them to execute on a regular schedule. These can be useful when you have a data source that is updated regularly.

Learn more about creating and managing scheduled Jobs.

Launchers

A Launcher is a web form that rests on top of a script that Domino can execute in a Job. You can use a Launcher to pass arguments to your script from UI fields in a customized web form.

Learn more about Launchers.

API

You can use the Domino API to start a Job with a POST request to:

https://<domino-url>/v1/projects/<username>/<project-name>/runs

For more information on how to send valid configuration data, read the API docs.

Quick Action menu

You can launch a Job from a Quick Action menu on the nav bar.
Managing Jobs with the Jobs dashboard

Click Jobs from the Project menu to open the Jobs dashboard. Here you can see a table of all Jobs from the history of this Project, sorted into tabs by queued, running, completed, and archived state. You can use the Jobs dashboard to find, manage, and view results from your Jobs in the following ways:

Viewing Job details and results

Click a row in the Jobs dashboard table to see a panel with details on the listed Job. You can filter the table by Job title with the Search box.
Attention: Domino retains up to 15 days worth of run usage details. This applies to all Runs and includes Jobs, Workspace sessions, and web apps.

Tagging Jobs

You can tag an individual job by clicking + under the Job title in the details panel.
You can also bulk tag Jobs by checking them in the table and then clicking the Bulk Tag button.

![Table of Jobs]

**Archiving Jobs**

You can archive Jobs by checking them in the table and clicking the Bulk Archive button.

![Table of Jobs]

**Stopping Jobs**

You can find currently running Jobs on the Running tab, and stop them by checking them and clicking the Bulk Stop button.
Comparing Jobs

You can create a Jobs comparison report by checking exactly two Jobs in the table, then clicking the Compare button.

Customizing your Jobs dashboard columns

Click the funnel icon above the table of Jobs to customize which columns are displayed in the Jobs dashboard. Note that if any Jobs in your project have produced Run diagnostic statistics, the keys from dominostats.json will become available as columns, allowing you to quickly compare these outputs.
5.2.2 Diagnostic statistics with dominostats.json

Understanding the performance of your experiments can involve analyzing many outputs and results. It’s often useful to see key metrics at a glance across all your runs, to allow you to quickly identify which experiments are worth investigating further.

Domino’s diagnostic statistics functionality allows you to do just that.

To use this feature, write a file named dominostats.json to the root of your project directory. Use keys in this JSON file to identify the outputs you’re interested in, and then add the corresponding values.

Here is some example R code that writes three key/value pairs to dominostats.json:

```r
diagnostics = list("R^2" = 0.99, "p-value" = 0.05, "sse" = 10.49)
library(jsonlite)
fileConn<-file("dominostats.json")
writeLines(toJSON(diagnostics), fileConn)
close(fileConn)
```

Here is the same data being written to dominostats.json by Python code:

```python
import json
with open('dominostats.json', 'w') as f:
    f.write(json.dumps({'"R^2": 0.99, "p-value": 0.05, "sse": 10.49'}))
```

The resulting dominostats.json file from these code examples looks like this:

```json
{
    "sse": 10.49,
    "R^2": 0.99,
    "p-value": 0.05
}
```

The dominostats.json file is deleted before each run automatically by Domino. Therefore, past dominostats.json files will not pollute new Jobs on your Jobs dashboard. If Domino detects that this file has
been written to the project root by a Job, it will parse the values out and show them as columns on the Jobs dashboard. You can see the keys represented as available columns in the dashboard, and each row contains the corresponding value from a Job.

You can also click Jobs Timeline at the top of the dashboard to expand a line chart of dominostats.json values over time. This chart shows all Jobs displayed in the current dashboard. To filter to a specific set of related Jobs, use tagging to create a separate dashboard view.

The x-axis ticks on the timeline represent individual Jobs, and the y-axis represents the values for the statistics in those jobs. Hover along the chart to see individual data points as tooltips, and click at a point to open the details for the Job that produced that value. You can also click and drag on the chart to zoom in, and you can click on each stat in the legend at upper right to toggle its line on and off.

### 5.2.3 Notifications

When a run finishes in Domino, by default you will be notified of failed project runs by email. You and your project collaborators can also choose to receive notifications of successful runs. On successful runs, you will receive an email with the last few lines of stdout, and up to 10 Results files from your run (Domino will detect which files have been added or changed during your run, and will capture those as the run’s Results).

You may edit your notification preferences in the project collaborators tab:
Custom Notifications

If you’d like to format your own success email to display your figures with more context, create a file named `email.html` in the root of your project folder as part of your run. The HTML will be used as the body of the email sent on success.

Tips and tricks:

- Include images by referencing the path to the image from the root of the folder (the image can be anywhere in your project). For instance, to include an image written to `plots/plot1.png` you would write `<img src="plots/plot1.png">`.
- Put all CSS styles in inline `style` attributes. Most email clients ignore `<style>` blocks in HTML emails.
- If you want to do complex layout, use tables. Most email clients ignore CSS positioning.
- You can customize the subject by including a `<head>` tag at the start of the file. `<head><title>Custom Title</title></head>` will result in an email with the subject “[Domino] Custom Title”. Be sure to include the `<title>` tags as well.
- If you would like to explicitly define the files that are sent with your success email, create a file named `.dominoresults` and write each filename per line in the `.dominoresults` file.

For example, here is an R script and the resulting email:

```r
# generate and save plot
ggplot <"pressure.png">
plot(pressure)
dev.off()

# generate HTML in a string
html_string <- paste0("<head>
<title>" , Sys.Date() , " - Pressure report" , "</title>
</head>
<body>
<h2>Exponential pressure growth!</h2>
<h3>" , Sys.time() , "</h3>
<img src="pressure.png" />
<p>Caption goes here</p>
</body>
")

# write string to file
write(html_string, "email.html")
```
Python example:

```python
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8])
y = np.array([495, 620, 761.88, 899.8, 1039.93, 1270.63, 1589.04, 1851.31, 2154.92])
plt.axis([0, 8, 0, 2200])
plt.xlabel('year')
plt.ylabel('balance')
plt.plot(x, y)
plt.savefig('results/plot.png')

email_content = ""
  <head>
    <title>Latest projections</title>
  </head>
  <body>
    <h1>Latest projections</h1>
    <div>
      <img src="results/plot.png">
    </div>
  </body>
""

f = open('email.html', 'w')
f.write(email_content)
f.close()
```
5.2.4 Results

The “Results” of a run are the set of files that your code generates or modifies when it runs. You can view the results of a given run through the Runs tab of your project. Or you can view the latest results on the Results tab of your project.

Results branching behavior

By default, when your code finishes running, Domino will save its results (i.e., any new files it produced) back into your project folder. This means those changes will be downloaded the next time you sync your project.

If your code generates large results that you don’t need to synchronize each time, you can set your project to save results to isolated “results branches.” Your results will be accessible through the web interface (and permanently saved), but they won’t automatically be accessible to subsequent runs, or automatically downloaded to your computer.

You can control this behavior on your project’s settings page, under the “result behavior” section.

Downloading results from an isolated branch

You can download isolated results from the UI as well as the CLI.

From the UI, select the run from the runs page, and click “view results”. From there you can choose to download each results file.

With the CLI, you can use the `domino download-results` command. For details on usage syntax, please run `domino help download-results`.

Controlling which files are shown in the results dashboard

If many files get changed during a run it might be difficult to navigate or get a high level summary. To control which files should be rendered as results, you can create a special file in your project folder called `.dominoreresults` (note the leading period). If this file exists and has entries, Domino will only render results that match patterns in the file.

Including single files in the results dashboard

To include only specific files of interest, your “.dominoreresults” file must list the relative path to those, each on a new line. Use the following example to display only `histogram.pdf` and `output.txt` files located in the top level directory of the project.

```text
histogram.pdf
output.txt
```
Using patterns with wildcard characters

Domino can use wildcards patterns in “.dominoresults”. This allows you to specify groups of files you want to be displayed. The following example will limit the files to PDF documents in the top-level directory, all PNG files in a directory images and text files arbitrarily deep in sub-directories of the project.

```
*.pdf
results/*.png
**/*.txt
```

5.2.5 Run comparison

Job comparison makes it really easy to figure out how two experiments differed in their inputs, and how those changes affected the results that were generated.

Comparing two Jobs is easy. From the Jobs dashboard, check the two Jobs you’re interested in then click the Compare button.

This will generate a report, summarizing the differences between those two Jobs.
Comparing Run #20 ...Run #19

Compare results or input files that may have changed across two runs.

<table>
<thead>
<tr>
<th></th>
<th>Base: #20</th>
<th>Target: #19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>experiment.sh 10000</td>
<td>experiment.sh 10000</td>
</tr>
<tr>
<td>Command</td>
<td>experiment.sh 10000</td>
<td>experiment.sh 10000</td>
</tr>
<tr>
<td>Status</td>
<td>✔️ Succeeded</td>
<td>✔️ Succeeded</td>
</tr>
<tr>
<td>Queued</td>
<td>Feb 14 10:19 AM</td>
<td>Feb 14 10:18 AM</td>
</tr>
<tr>
<td>Duration</td>
<td>0m</td>
<td>0m</td>
</tr>
<tr>
<td>Started By</td>
<td>njablonski</td>
<td>njablonski</td>
</tr>
<tr>
<td>Environment</td>
<td>Domino Analytics Distribution - Py3.6 R3.4 - quay base from shiny - Version #8</td>
<td>Domino Analytics Distribution - Py3.6 R3.4 - quay base from shiny - Version #8</td>
</tr>
<tr>
<td>Hardware Tier</td>
<td>default (8 CPUs - 32GB RAM)</td>
<td>default (8 CPUs - 32GB RAM)</td>
</tr>
<tr>
<td>Imported Domino Projects</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Imported Repos</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

0 result files changed

0 input files changed

Diagnostic statistics changed

<table>
<thead>
<tr>
<th></th>
<th>Base</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>spicy-factor</td>
<td>50000</td>
<td>spicy-factor</td>
</tr>
<tr>
<td>viscosity</td>
<td>5000</td>
<td>viscosity</td>
</tr>
</tbody>
</table>

If you are tracking Run Diagnostic Statistics, the comparison view will show you the difference between your stats.

Remember that Domino snapshots the state of all the files in the project before the run starts (the inputs) and snapshots the project after the run completes (the outputs). Any files that were added or modified between the input and outputs are considered results.

Domino will do its best to show you differences as sensibly as it can. For text, we will highlight the lines in the file that are different:
For files that we know how to render, we will render those files side-by-side so you can easily visually compare:

For files Domino doesn’t know how to render, we’ll give you some simple metadata and links to download the exact version so you can look at them on your own computer:

5.2. Domino runs
5.2.6 Advanced options for Domino runs

Run states

When you start a Run in Domino, the Run will move through several lifecycle states. This page describes what these different stages mean.

1. Queued

The Run is waiting for a machine of your specified hardware tier to become available. If one is available, it will quickly leave this state. However if no slots are available, it can take several minutes to start up a new machine.

2. Scheduled

This is when the dispatcher has requested an executor to process the Run and the executor acknowledges this request, but hasn’t begun the processing yet. A Run only remains in this state for a few seconds.

3. Preparing

Your project files are being copied to the executor where your code will run. Depending on the size of your data and the number of files in your project, this might finish quickly, or may take a while. Files are cached when possible, so if you start a Run on a hardware tier you used recently with the same project, this may be quick, even for projects with large files.

4. Building

If you are using a custom environment,
you may need to wait for the Docker image to build. This is cached whenever possible, so subsequent Runs on the same hardware tier may skip this step.

5. **Pulling**

When your Docker image has been saved to a network-attached repository, this state indicates we are fetching the image.

6. **Running**

Your code is executing. You can view the console output and resource usage on the Jobs or Workspaces dashboard.

7. **Finishing**

Your Run has completed, and any file changes are being copied back to the Domino file store.

8. **Succeeded**

Your Run has finished without error.

9. **StopRequested**

The request to stop your Run has been received.

10. **Stopping**

If you manually stop your Run, it will enter this state while any new or updated files are synced back to the project.

11. **Stopped**

The Run has been manually stopped.

12. **Failed**

5.2. **Domino runs**
Your Run did not complete due to a problem with your code.

13. **Error**

Some problem outside of your code caused the Run to terminate.

**Domino environment variables**

Domino automatically injects several environment variables whenever it runs your code, as part of the context of your run.

If you’re looking to define your own environment variables, please see Environment variables for secure credential storage.

**Motivation**

First, you can use these to programmatically determine if your code is running in Domino or not. This is useful in cases where your code might do something different when running locally vs running on Domino.

Second, these variables can be useful when generating artifacts or outputs – e.g., a report you produce might refer to the run number.

**Variables**

Domino automatically injects the following variables:

- **DOMINO_USER_API_KEY** — Useful if you want to use the Domino API to access another project
- **DOMINO_API_HOST** — same as above
- **DOMINO_PROJECT_OWNER** — username of the owner of the running project
- **DOMINO_PROJECT_NAME** — name of the running project
- **DOMINO_RUN_ID** — run ID of the current run
- **DOMINO_RUN_NUMBER** — run number of the current run
- **DOMINO_HARDWARE_TIER_ID** - hardware tier the current run is executing on (new in v1.42)
- **DOMINO_STARTING_USERNAME** - username of the user who began the run (new in v1.43)
- **DOMINO_WORKING_DIR** - working directory for the running project
- **AWS_SHARED_CREDENTIALS_FILE** - path to your AWS credential file for connecting to addition AWS resources (e.g. S3, Redshift). Learn more.
- **DOMINO_TOKEN_FILE** - path to a jwt token signed by Domino useful for authenticating with the Domino API or other third party services. Learn more.

Note: These variables are not available in the Model Manager

**Usage**

Here are some examples on retrieving an environment variable within your code:

**R**

```r
Sys.getenv("DOMINO_RUN_ID")
```
Environment variables for secure credential storage

Domino’s Environment Variables give you a safe and easy way to inject sensitive configuration into the execution of your analysis or models.

Environment Variables are stored securely, modifiable only by owners of a project or in the case of models also by editors, and are not tied to the version history of your project or model, so they are easily revokable.

Why use environment variables for configuration?

Your code may need to connect to external resources, like a database or S3. Often these connections are authenticated via a secure password, key, or token. It is a bad idea to put this type of secure configuration directly in your source because:

- You often want to share source files but don’t want to leak those credentials
- It’s difficult to scrub references to those credentials from a version control system like Git or Domino
- You may want only a more privileged user (like the project owner) to be able to change certain configuration parameters. If configuration is all done through code, all users that can modify the scripts can potentially change the config.

It is a better idea to have your configuration stored and permission separately, and have it injected when your code executes.

Setting up project environment variables

You can configure your secure configuration to be injected at execution time via environment variables.

To set this up, go to the Settings tab on the project you wish to configure. Under the “Environment variables” section you can add key/value pairs which will be injected as env vars at execution time:
The values are passed verbatim, so no escaping is required. Note that there is a 64K length limit for the value.

**Setting up user environment variables**

Environment variables can also be configured on a per-user basis. These variables will be injected at execution time for any run the user starts.

User Environment Variables are automatically imported into Runs across all projects, and can be accessed like any other Environment Variables. Note that user specific environment variables are not used or available in Models.

To configure your user environment variables, visit your account settings page by clicking your username and then “Account Settings” from the top right of the Domino navigation bar. From your account settings page, scroll down to the section titled “User environment variables”. Here you can configure variables for your user account in the same way as project environment variables (described above).

**Setting up model environment variables**

You can configure your secure configuration to be injected at execution time via environment variables.

To set this up, go to the **Settings** tab on the model you wish to configure. Under the “Environment” section you can add key/value pairs which will be injected as env vars at execution time, in the same manner as project environment variables.

The values are passed verbatim, so no escaping is required. Note that there is a 64K length limit for the value.

When you add a variable the values are pushed to all running model versions. Only owners or editors can create environment variables.

Note that project level and user level environment variables are not used in Models and must be set separately on the model.
A note about connecting to Git repos

If you’re using a user level environment variable to connect to Github and download repos, you will need to add that into your environment definition this way:

```
RUN pip install git+https://personalaccesstoken:personalaccesstoken@github.com/<repo>
```

Using Environment Variables in Pre- or Post-setup Scripts of your Environment

If you want to reference custom-defined environment variables in the pre- or post-setup script of your custom compute environment, you’ll need to make sure the variable name has the prefix “DRT_”.

Hierarchy of environment variables

It is possible to set the same variable in different places, each of the following will override the previous one in the following order:

- Compute environment
- Project
- User Account

So to clarify for a given variable the table below outlines which values are set and the expected result

<table>
<thead>
<tr>
<th>Place set</th>
<th>Run#1</th>
<th>Run#2</th>
<th>Run#3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute Environment</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Project</td>
<td></td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>User Account</td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>Run Result</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

Reading the environment variables

Every language has its own way of reading env vars. In Python, it might look like this:

```
import os
s3 = S3Client(os.environ['S3_KEY'], os.environ['S3_SECRET'])
```

For more details, please read this Python help documentation.

In R, it might look like this:
makeS3Client(Sys.getenv("S3_KEY"), Sys.getenv("S3_SECRET"))

For more details, please read this R help documentation.

Using Apache Airflow with Domino

- **Overview**
- **Domino and Airflow webinar**
- **Getting started with Airflow**
- **Installing python-domino on your Airflow workers**
- **How Airflow tasks map to Domino Jobs**
- **Example pipeline**

**Overview**

Data science projects often require multiple steps to go from raw data to useful data products. These steps tend to be sequential, and involve things like:

- sourcing data
- cleaning data
- processing data
- training models

Once you understand the steps necessary to deliver results from your work, it’s useful to automate them as a repeatable pipeline. Domino has the ability to schedule Jobs, but for more complex pipelines you can pair Domino with an external scheduling system like Apache Airflow.

This article will describe how to integrate Airflow with Domino by using the python-domino package.

**Domino and Airflow webinar**

This video is a recording of a webinar held on February 21st, 2019. This webinar walks through a detailed example of integrating Airflow with Domino.

Click here to see the slides from this presentation.
Getting started with Airflow

Airflow is an open source platform to author, schedule, and monitor pipelines of programmatic tasks. As a user, you can define pipelines with code and configure the Airflow scheduler to execute the underlying tasks. The Airflow UI can be used visualize, monitor, and troubleshoot pipelines.

If you are new to Airflow, read the Airflow QuickStart to set up your own Airflow server.

There are many options for configuring your Airflow server, and for pipelines that can run parallel tasks, you will need to use Airflow’s LocalExecutor mode. In this mode you can run tasks in parallel and execute multiple dependencies at the same time. Airflow uses a database to keep records of all the tasks it executes and schedules, so you will need to install and configure a SQL database for LocalExecutor mode.

Read the following guide to learn more about setting up LocalExecutor mode:

- A Guide On How To Build An Airflow Server/Cluster

For more information about scheduling and triggers, notifications, and pipeline monitoring, read the Airflow documentation.

Installing python-domino on your Airflow workers

To create Airflow tasks that work with Domino, you need to install python-domino on your Airflow workers. This library will enable you to add tasks in your pipeline code that interact with the Domino API to start Jobs.

Connect to your Airflow workers, and follow these steps to install and configure python-domino:

1. Install from pip
   
   ```bash
   pip install git+https://github.com/dominodatalab/python-domino.git
   ```

2. Set up an Airflow variable to point to the Domino host. This is the URL where you load the Domino application in your browser.

   **Key:** DOMINO_API_HOST  
   **Value:** <your-domino-url>

3. Set up an Airflow variable to store the user API key you want to use with Airflow. This is the user Airflow with authenticate to Domino as for the purpose of starting Jobs.

   **Key:** DOMINO_API_KEY  
   **Value:** <your-api-key>
How Airflow tasks map to Domino Jobs

Airflow pipelines are defined with Python code. This fits in well with Domino’s code-first philosophy. You can use python-domino in your pipeline definitions to create tasks that start Jobs in Domino.

Architecturally, Airflow has its own server and worker nodes, and Airflow will operate as an independent service that sits outside of your Domino deployment. Airflow will need network connectivity to Domino so its workers can access the Domino API to start Jobs in your Domino project. All the code that performs the actual work in each step of the pipeline – code that fetches data, cleans data, and trains data science models – is maintained and versioned in your Domino project. This way you have Domino’s Reproducibility engine working together with Airflow’s scheduler.

Example pipeline

The following example assumes you have an Airflow server where you want to set up a pipeline of tasks that fetches data, cleans and processes data, performs an analysis, then generates a report. It also assumes you have all the code required to complete those tasks stored as scripts in a Domino project.

The example graph shown above is written using Airflow and python-domino, and executes all the dependencies in Domino using the Airflow scheduler. It trains a model using multiple datasets, and generates a final report.

See the commented script below for an example of how to configure an Airflow DAG to execute such a pipeline with Domino Jobs.

```python
from datetime import datetime, timedelta
from airflow import DAG
from airflow.operators.dummy_operator import DummyOperator
from airflow.operators.python_operator import PythonOperator
```

(continues on next page)
from domino import Domino
from airflow.models import Variable

# Initialize Domino API object with the api_key and host
api_key=Variable.get("DOMINO_API_KEY")
host=Variable.get("DOMINO_API_HOST")
domino = Domino("sujaym/airflow-pipeline",api_key,host)

# Parameters to DAG object
default_args = {
    'owner': 'domino',
    'depends_on_past': False,
    'start_date': datetime(2019, 2, 7),
    'retries': 1,
    'retry_delay': timedelta(minutes=.5),
    'end_date': datetime(2019, 2, 10),
}

# Instantiate a DAG
dag = DAG('domino_pipeline', description='Execute Airflow DAG in Domino',default_args=default_args,schedule_interval=timedelta(days=1))

# Define Task instances in Airflow to kick off Jobs in Domino
t1 = PythonOperator(task_id='get_dataset_1', python_callable=domino.runs_start_blocking, dag=dag, op_kwargs="command":["src/data/get_dataset_1.py"])
t2 = PythonOperator(task_id='get_dataset_2', python_callable=domino.runs_start_blocking, op_kwargs="command":["src/data/get_dataset_2.py"], dag=dag)
t3 = PythonOperator(task_id='get_dataset_3', python_callable=domino.runs_start_blocking, op_kwargs="command":["src/models/get_dataset_3.sh"], dag=dag)
t4 = PythonOperator(task_id='clean_data', python_callable=domino.runs_start_blocking, op_kwargs="command":["src/data/cleaning_data.py"], dag=dag)
t5 = PythonOperator(task_id='generate_features_1', python_callable=domino.runs_start_blocking, op_kwargs="command":["src/features/word2vec_features.py"], dag=dag)
t6 = PythonOperator(task_id='run_model_1', python_callable=domino.runs_start_blocking, op_kwargs="command":["src/models/run_model_1.py"], dag=dag)
t7 = PythonOperator(task_id='do_feature_engg', python_callable=domino.runs_start_blocking, op_kwargs="command":["src/features/feature_eng.py"], dag=dag)
t8 = PythonOperator(task_id='run_model_2', python_callable=domino.runs_start_blocking, op_kwargs="command":["src/models/run_model_2.py"], dag=dag)
t9 = PythonOperator(task_id='run_model_3', python_callable=domino.runs_start_blocking, op_kwargs="command":["src/models/run_model_3.py"], dag=dag)
t10 = PythonOperator(task_id='run_final_report', python_callable=domino.runs_start_blocking, op_kwargs="command":["src/report/report.sh"], dag=dag)

# Define your dependencies
t2.set_upstream(t1)
t3.set_upstream(t1)
t4.set_upstream(t2)

(continues on next page)
t5.set_upstream(t3)
t6.set_upstream([t4, t5])
t7.set_upstream(t4)
t8.set_upstream(t7)
t9.set_upstream(t7)
t10.set_upstream([t6, t8, t9])

5.3 Scheduled Jobs

Domino allows you to schedule Jobs in advance, and set them to execute on a regular cadence. These can be useful when you have a data source that is updated regularly.

To schedule a Job, or manage existing scheduled Jobs, click Scheduled Jobs from the project menu.

1. Job Definition
   • Scheduled Job Name
     Enter the name of the job. Each job will have this name on the Jobs Dashboard.
   • File Name
     Enter the name of the file you’d like to execute. Include any optional arguments you wish to pass to your file.
• Hardware tier
  This drop-down list enables you to set the hardware tier used by the Job.

• Environment
  This drop-down list enables you to set the compute environment used by the Job.

• Datasets
  This expanding panel lists the Datasets configuration used by the Job.

2. Spark Cluster

• Attach Domino Managed Spark Cluster
  This option will allow you to provision and attach an on-demand Spark cluster to the Job.

• The remainder of the configurations are explained in Spark Cluster Settings.
3. Schedule

- **Repeat every**
  
  Here you can set the frequency at which you want the Job to repeat.

- **Use custom expression**
  
  Here you can enter a custom Quartz CronTrigger expression, if the desire scheduling option is not available in the “Repeat every” selector.

- **Run sequentially**
  
  Setting a Job to Run sequentially will cause the scheduler to always wait for the last Job it started to complete before starting the next one. For example, if you set up a scheduled Job to run once per hour, and one of the Jobs launched by the scheduler takes 90 minutes to complete, the next hourly Job will not start until the previous one has finished. Otherwise, multiple Jobs from this scheduler will be allowed to run simultaneously. The scheduler will not wait for the previous Job to finish if it's still running. This mode should be used when your Job doesn’t depend on output from the previous Job.
4. Actions

- Emails to notify
  In this field, add the email addresses of everyone who should be notified when the Job completes.
- Update Model API
  If a Model API has been publishing from the Project, the selected Model API will be republished after the Job has completed. This is useful for re-training and updating a Model API on a regular basis.

5.4 Domino workspaces

Domino workspaces function as a server and allow you to use your preferred tools in a reproducible and customizable environment.

5.4.1 Workspaces

Overview

A Domino workspace is an interactive session where you can conduct research, analyze data, train models, and more. Workspaces enable you to work in a development environment of your choice, like Jupyter notebooks, RStudio, VS Code, and many other customizable environments.
Launch a workspace

1. Click **Workspaces** in the Domino sidebar menu.

2. Click the + **Create New Workspace** button. A workspace launch modal will appear.
3. Enter a name for your workspace.

4. Select an environment for your workspace. An environment is the software configuration (language, packages, etc.) that will be used in your workspace. You can select one of Domino’s pre-defined environments, or create a custom environment of your own. To learn more about managing environments, refer to the Domino environments documentation.

5. Select the integrated development environment, or IDE, that you’ll use in your workspace (i.e., Jupyter, etc.). The IDEs available in this step are determined by the environment you select. The list of IDEs will update dynamically if a different environment is selected. To learn more about managing IDEs, refer to the Domino pluggable notebooks documentation.

6. Select a hardware tier. A hardware tier represents the compute hardware used for your run. It can be a virtual instance in a cloud services provider, or a physical machine running in your deployment’s on-premise data center. To learn more about hardware tiers, refer to the hardware tier documentation.
7. Optionally, configure datasets or external data volumes in the “Data” section of the modal.

8. If necessary, attach a Spark compute cluster to your workspace. To learn more about Spark clusters, please refer to the Domino Spark documentation.
9. Click the **Launch** button. A loading screen will open in a new tab. You’ll be directed to your workspace when it’s ready. If you’re unable to access your workspace, check your browser’s configuration settings. Your browser should be configured to allow pop-up windows from Domino.
Attention: If you’re unable to access a newly launched workspace, check your browser’s configuration settings. Your browser should be configured to allow pop-up windows from Domino.

Attention: Domino imposes a limit of 2 workspaces per user per project, by default. To change this setting, please contact your Domino administrator.

Save your work

Use the various “Save” options in your IDE to save work you complete in your workspace.

Note that saving your work in a workspace is not the same thing as syncing your work to Domino. When you save changes in your workspace (using any of the various “Save” option(s) in your IDE) you’re simply saving your work within your workspace. If you’d like to push those changes to the Domino File System (DFS), you’ll have to sync your work to Domino instead.
Sync your work to Domino

**Attention:** If you’re using a Git-based project with CodeSync, then the sidebar menu in your workspace look a little different. To learn more about saving and syncing your work using Git in a Git-based project with CodeSync, please see Working with Git.

Sync your work to the Domino File System (DFS) using the sidebar menu in your Domino workspace. You have the option of syncing all changes at once, or syncing only recent changes made to your files. We recommend that you sync your work at least daily.

**Sync Changes to Files**

1. Click **File Changes** in the sidebar of your workspace.
2. Under “Files”, view changes by expanding “File Changes”.
3. Enter a commit message.
4. Click **Sync to Domino**. Changes to files in the `/mnt` directory of your workspace will be synced to the Domino File System (DFS). Changes to files outside of the `/mnt` directory will not be synced.
5. After syncing to Domino is complete, you’ll be able to view your files in the Files section of Domino.

Sync All Changes

1. Click File Changes in the sidebar of your workspace.
2. Under “All Changes”, enter a commit message.
3. Click Sync All Changes. Domino will save all of your work / changes to the Domino File System (DFS).
Warning: When saving your work, remember that changes to files outside of the /mnt directory will not persist if you stop your workspace and resume the workspace at a later time.

Stop a workspace

If necessary, you can stop a workspace and resume it at a later time. Frequently stopping and resuming a workspace session is a good way of managing compute costs (e.g., EC2). Storage costs (e.g., EBS), however, will continue to incur.

Attention: Stopping a workspace will not automatically sync your work to Domino. If you’d like to sync your work and files to Domino, you’ll need to sync from within a workspace.

To stop a workspace:

1. Click Stop at the top of your workspace session. A modal with information about pausing your workspace will appear.
2. Click **Stop My Workspace**.
Persisted and reloaded settings

If you stop your workspace, the following settings will persist and will be available to you when you resume your workspace session at a later time:

- Files saved in the /mnt directory

The following settings, however, will not persist and instead will reload when you resume your stopped workspace:

- Files outside of the /mnt directory, including installed packages
- Objects in memory
- Datasets

If you’re using a Git-based project with CodeSync and the credentials to the git repository backing your code are updated or deleted while your workspace session is stopped, then the credentials will be updated correspondingly when you resume your workspace.

Resume a workspace

To resume a stopped workspace:

1. Click Workspaces in the Domino sidebar menu.
2. Navigate to the stopped workspace in your workspace dashboard. Click Start. Your workspace will resume in a new tab.
View workspaces

View Your Workspaces

To view workspaces that you’ve launched:

1. Click Workspaces in the Domino sidebar menu.
2. Click My Workspaces in the toolbar above your workspaces.
View All Workspaces in your Project

1. Click Workspaces in the Domino sidebar menu.
2. Click All Workspaces in the toolbar above your workspaces. Workspaces that you started, or that collaborators on your project started, are listed here.

View Deleted Workspaces

1. Click Workspaces in the Domino sidebar menu.
2. Click Deleted in the toolbar above your workspaces.
View Utility Workspaces

All non-durable workspaces (“legacy” workspaces) are listed in the **Utility** section. To view these workspaces:

1. Click **Workspaces** in the Domino sidebar menu.
2. Click **Utility** in the toolbar above your workspaces.
Delete a workspace

**Warning:** Ensure that all of your work is synced to Domino before deleting a workspace. Failure to do so will result in unrecoverable work.

You can delete a workspace if it’s no longer needed. Workspaces must be stopped before they can be deleted.

1. If necessary, first stop your workspace. You can stop a workspace by clicking **Stop** in the workspaces dashboard, or clicking the **Stop** button in the toolbar from within your workspace dashboard.
2. Navigate to the workspaces dashboard. Click the red trash bin icon.
3. In the modal that appears, confirm that you’d like to delete the workspace, then click **Delete**. The workspace will be deleted. You can view deleted workspaces by clicking **Deleted** at the top of your workspaces dashboard.
Workspace settings and data

You can view your workspace’s settings, usage, logs, and history from the workspaces dashboard.

**Workspace Settings**

1. Click **Workspaces** in the Domino sidebar menu.

2. Click **Settings** in the corresponding workspace’s panel. You can view settings information about your code, environment, hardware tier, and datasets. You can also edit workspace settings by clicking the **Edit Settings** button. If you’d like to edit the workspace’s settings, stop the workspace.

**Workspace Usage**

1. Click **Workspaces** in the Domino sidebar menu.

2. Click **Usage** in the corresponding workspace’s panel. The panel will expand to show CPU usage for your workspace.
Workspace Logs

1. Click **Workspaces** in the Domino sidebar menu.
2. Click **Logs** in the corresponding workspace’s panel.

Here, you can view “User” logs and “Setup” logs about your workspace. User logs include information about actions you take within your workspace. Setup logs contain information about Domino and Kubernetes. If your workspace ever fails, the logs are a good place to begin investigating.
Workspace History

1. Click Workspaces in the Domino sidebar menu.

2. Click History in the corresponding workspace’s panel. A modal will appear where you can view historical information about your workspace, like when a workspace was launched, stopped, and more.
Workspace volume size

Volumes represent the storage space dedicated to your workspace or Job. The default volume size for all workspaces and jobs in Domino is 10GiB. You can change the size of your volume if you find that your workspace (or Job) will require more storage space.

To change the size of your volume:

1. Navigate to your project’s settings.
2. Under “Workspace and Jobs Volume Size”, enter your desired volume size. The default minimum volume size is 4GiB, while the default maximum volume size is 200GiB. Both of these values are configurable. If you’d like to change the default minimum and maximum volume size limits, please contact your Domino administrator.

**Attention:** Changes you make to the volume size will not impact existing workspaces. Instead, the changes will be applied to subsequent, new workspaces.

5.4.2 Using Git in Your Workspace

Overview

Git is fully supported in Domino workspaces in Git-based projects with CodeSync, and imported Git repos. You can use common Git workflows like creating branches, committing (saving) your code, pushing or pulling changes, and more, all from within a workspace.
Methods for working with Git

There are a few ways of using Git in a Git-based project with CodeSync:

- **Git service provider and workspace sidebar** – You can use your Git service provider’s (i.e. GitHub, etc.) user interface in collaboration with the sidebar actions (Sync to Git, Sync to Domino, Pull) in a Domino workspace to work with Git in your project. We recommend using Git in your project this way if you don’t have a lot of experience using Git with a terminal and instead prefer more guidance through a user interface. The rest of the screen captures on this page will demonstrate this method.

- **A terminal within a Domino workspace** – Start a terminal in your workspace and use git just like you would on a local environment. For a refresher on common workflows with Git, see the following resource: [Git cheatsheet](#).
• **Git plugins** – Many IDEs integrate natively with Git or have plugins or extensions for Git, as shown in the screenshot below. These features are fully supported in Git-based projects via CodeSync technology, and you can use them to interact with Git in Domino. For information on how to use git version control with your preferred IDE, please see the following resources:

  - Jupyter – Version control for Jupyter notebooks
  - RStudio – Managing GitHub and RStudio
  - VS Code – Version control in VS Code
Commit code and push changes

Note: Note that in a Git-based project with CodeSync, the default working directory for your project’s code is 
/mnt/code.

To commit changes to your code and push them to your repository:

1. Click on the File Changes option in the sidebar of your workspace.
2. Under “Code”, view changes by expanding “Uncommitted Changes”.
3. Enter a commit message.
4. Click Sync to Git. Domino will sync and push your changes to the Git repository you’re using for your code.

The up (↑) arrow in the “Code” section indicates the number of commits that your branch is ahead of the upstream branch, while the down (↓) arrow indicates the number of commits that your branch is behind the upstream branch.

Attention: You may encounter merge conflicts when pulling or pushing code to your repository. To resolve merge conflicts, please refer to the section on resolving merge conflicts.
Commit all work

1. Click on the **File Changes** option in the sidebar of your workspace.
2. Under “All Changes”, enter a commit message.
3. Click **Sync All Changes**. Domino will commit changes to your code and sync them to your Git repository, as well as commit changes to artifacts and sync them to the Domino File System (DFS).

**Note:** Note that in a Git-based project with CodeSync, the default working directory for your project’s code is /mnt/code.

Pull the latest changes

If you’re collaborating with others in a Git-based project with CodeSync, then you’ll need to pull the latest changes to your code from time to time.
To pull the latest changes from your Git repo:

1. Click the **File Changes** option in the sidebar menu of your workspace.
2. Under the “Code” section, click **Pull**. Domino will pull the latest changes into your workspace.

**Note:** Note that in a Git-based project with CodeSync, the default working directory for your project’s code is `/mnt/code`.

**Attention:** You may encounter merge conflicts when pulling or pushing code to your repository. To resolve merge conflicts, please refer to the section on resolving merge conflicts.

### Resolve merge conflicts

On occasion, you may encounter merge conflicts in your code as a result of collaborating with colleagues. You can resolve merge conflicts in one of the following ways.

- **Git service provider and workspace sidebar**

  When merge conflicts arise, Domino will create a new branch with the conflicts in your repository and push them up to your Git service provider (i.e., Github, Bitbucket, etc.). To resolve the conflicts:
1. Navigate to your Git service provider.

2. Using your Git service provider’s user interface, create a pull request between the newly created branch and your original branch.

3. Using your Git service provider’s user interface, merge the pull request.

4. Navigate back to Domino. Do not pull any changes.

5. Start a workspace terminal and reset the branch using `git reset --hard origin/name-of-branch`, where `name-of-branch` is the name of your original branch.

6. Click Pull in the sidebar menu in your workspace. Domino will pull all of the latest changes (i.e., the resolved merge conflicts). Your branch should now be free of merge conflicts.

   • A terminal in your workspace

If you prefer to work in a terminal, you can resolve merge conflicts from within a workspace terminal. If you’re not familiar with the typical Git workflow to resolve merge conflicts, we recommend the following resource: Git merge conflicts.

### 5.4.3 Using Visual Studio Code in Domino Workspaces

#### Contents

- Using Visual Studio Code in Domino Workspaces
  - Overview
  - Launching a VSCode Workspace
  - Installing VSCode extensions
  - Installing VSCode to older environments

#### Overview

Some Domino Standard Environments support launching Visual Studio Code (VSCode) in interactive Workspaces. VSCode is an open-source multi-language editor maintained by Microsoft. Domino can serve the VSCode application to your browser with the power of code-server from Coder.com.

#### Prerequisites

VSCode support is available in the latest versions of the following Domino Standard Environments:

- Domino Analytics Distribution for Python 2.7
  - quay.io/domino/base:Ubuntu18_DAD_Py2.7_R3.5-20190501
- Domino Analytics Distribution for Python 3.6
Launching a VSCode Workspace

When using a VSCode-equipped Domino environment, there are two ways to launch the VSCode application.

Option 1: Launch VSCode directly

You can launch VSCode directly from the Workspaces dashboard or Quick Action menu, the same way you would launch RStudio or Jupyter.

Select Quick Action

Press \( \text{a} \) to open this Quick Action prompt
If launched this way, your Workspace will open with the Domino controls around a VSCode editor. You can work with your project files in VSCode, and commit and sync with the Domino Workspace UI as normal.

Option 2: Launch VSCode from JupyterLab

In VSCode-equipped environments, you will also find VS Code IDE as a notebook option in JupyterLab.

If launched this way, JupyterLab will open a new tab that will serve the VSCode application. This editor is running in the same *Domino Run container* as your JupyterLab application. However, the VSCode tab will *not* show the Domino Workspace controls. If you want to sync, commit, or stop your Workspace after working in VSCode, you must do so from the JupyterLab tab.
Installing VSCode extensions

You can use the extensions manager in VSCode to install extensions from the marketplace as you would usually. However, note that these extensions are installed only in the current Workspace session, and will not persist once the session is shut down.

To install persistent extensions that will be available in every new VSCode Workspace, you must build them into your environment. Use the following steps to set up such an environment.

1. Find the extension you want to install in the Visual Studio Marketplace. In this example, we’ll install the scala-lang extension.

2. Microsoft obscures the download URL for the extension by default, so you will need to first open your browser’s development tools, then click the Download extension link.
3. You can retrieve the download URL for the extension by looking at the request details in your browser’s development tools. It should end with `/vspackage`. Copy this URL for use in your custom environment.

4. In Domino, create a new environment. As the base image, you must use one of the VSCode-equipped Domino Standard Environments, listed in the prerequisites at the beginning of this article.
5. Add the following instructions to your new environment’s Dockerfile, replacing the folder names and example
/vspackage URL with the extension URL you retrieved earlier. These commands download the extension, extract the required files, and adds them to the appropriate folder.

```bash
RUN apt-get update
RUN apt-get install -y bsdtar
RUN mkdir -p /home/ubuntu/.local/share/code-server/extensions/ms-python.python-2019.3.6558
RUN cd /home/ubuntu/.local/share/code-server/extensions/ms-python.python-2019.3.6558
RUN cd /home/ubuntu/.local/share/code-server/extensions/ms-python.python-2019.3.6558/extension/ && mv * ../
RUN chown ubuntu:ubuntu /home/ubuntu/.local/share/code-server/
```

6. When finished, click **Build**. Following a successful build you can use this new environment to launch VSCode Workspace sessions with your desired extensions already installed.

---

**Installing VSCode to older environments**

VSCode can be added to some older environments by adding the following to your compute environment. The base environment must be 2018-05-23 or newer.
You can add the following to your compute environment docker file instructions:

```bash
#note: Make sure you are using the latest release if you'd like the latest version of the workspaces
#https://github.com/dominodatalab/workspace-configs/releases

RUN cd /tmp && wget https://github.com/dominodatalab/workspace-configs/archive/2019q2-v1.3.zip &&
unzip 2019q2-v1.3.zip cp -Rf workspace-configs-2019q2-v1.3/vscode /var/opt/\nworkspaces/vscode &&
rm -rf /var/opt/workspaces/workspace-logos &&
rm -rf /tmp/workspace-configs-2019q2-v1.3

RUN \
chmod +x /var/opt/workspaces/vscode/install && sleep 2 &&
/var/opt/workspaces/vscode/install
```

Next, add the following to your compute environment’s **Pluggable Workspace Tools**:

```json
vscode:  
title: "vscode"  
start: [ "/var/opt/workspaces/vscode/start" ]  
httpProxy:  
  port: 8888
```

### 5.4.4 Persisting RStudio preferences

In the context of your runs, the RStudio user preferences (such as theme) are stored in a file located at /home/ubuntu/.rstudio/monitored/user-settings/user-settings. You can launch RStudio with custom preferences by modifying this file via the pre-setup script of a custom compute environment.

**Method 1: Write lines to settings file**

If you know the line you need to add to the settings file, you can write it directly in the pre-setup script. For example:

```bash
mkdir -p /home/ubuntu/.rstudio/monitored/user-settings/  
echo 'uiPrefs="{"theme": "Mono Industrial"}" >> /home/ubuntu/.rstudio/monitored/user-settings/user-settings
chown -R ubuntu:ubuntu /home/ubuntu/.rstudio
if [ -f .domino/launch-rstudio-server ] ; then  
  sed -i.bak 's# > ~/.rstudio/monitored/user-settings/user-settings# >> ~/.rstudio/monitored/user-settings/user-settings#' .domino/launch-rstudio-server
  chown ubuntu:ubuntu .domino/launch-rstudio-server
fi
```

Here’s what each line is doing:

- The `mkdir` statement creates the encompassing directory.
- The `echo` statement writes the theme to the file. This can be replaced with a copy operation if you’d prefer to store a file in your project (see next section).
- The `chown` statements are needed to avoid a permissions error.
- The `sed` statement modifies a Domino script that would otherwise overwrite this settings file.

**Method 2: Copy a saved settings file**
If you aren’t sure which lines to write, or if you want to persist this settings file in your project, you can save a copy in your project and use the following pre-setup script code to apply it to your session.

First, run a session and modify the RStudio preferences to your liking. Before you stop the session, copy the user-settings file to the root of your project directory. You can do so with this line of R code:

```r
file.copy("/home/ubuntu/.rstudio/monitored/user-settings/user-settings", ".")
```

Then, add the following lines to the pre-setup script of your environment definition, in order to load the preferences file (if it exists) on subsequent runs:

```bash
if [ -f user-settings ]; then
    mkdir -p /home/ubuntu/.rstudio/monitored/user-settings/
    cp user-settings /home/ubuntu/.rstudio/monitored/user-settings
    sed -i.bak '/initialWorkingDirectory=/d' /home/ubuntu/.rstudio/monitored/user-settings/user-settings
    chown -R ubuntu:ubuntu /home/ubuntu/.rstudio
    if [ -f .domino/launch-rstudio-server ]; then
        sed -i.bak 's# > ~/.rstudio/monitored/user-settings/user-settings# >> ~/rstudio/monitored/user-settings/user-settings#' .domino/launch-rstudio-server
        chown ubuntu:ubuntu .domino/launch-rstudio-server
    fi
fi
```

Note: the sed statement that deletes the “initialWorkingDirectory” variable is necessary to make sure your session starts with the correct working directory.

### 5.4.5 Accessing multiple hosted applications in one Workspace session

#### Overview

For security reasons, Domino Workspace sessions are only accessible on one port. For example, Jupyter typically uses port 8888. When you launch a Jupyter Workspace session, a Domino executor starts the Jupyter server in a Run, and opens port 8888 to serve the Jupyter application to your browser. If you were to attempt to use the Jupyter terminal to start another application on a different port, it would not be accessible.

However, in some cases you may want to run multiple interactive applications in the same Workspace session. These cases include:

- Editing and debugging Dash or Flask apps live
- Using Tensorboard to view progress of a live training job

Domino 3.5+ supports this with Jupyter Server Proxy and JupyterLab.

#### Prerequisites

- Python 3+
- Jupyter Server Proxy

Jupyter Server Proxy is installed by default in the latest Domino Standard Environments To install it in one of your existing environments, see the instructions below.
Installing Jupyter Server Proxy in your environment

If you are not on the recent version of the Domino Standard Environments, you can install Jupyter Server Proxy in your Domino environment, by following these steps.

1. Add the following lines to your environment’s Dockerfile Instructions.

   ```sh
   # Install NodeJS
   # You can omit this step if your environment already has NodeJS 6+ installed
   RUN curl -sL https://deb.nodesource.com/setup_8.x | bash - &&
       apt-get install nodejs -y &&
       rm -rf /var/lib/apt/lists/*

   # Switch to the latest JupyterLab start script
   RUN rm -rf /var/opt/workspaces/Jupyterlab/start.sh &&
       cd /var/opt/workspaces/Jupyterlab/ &&
       wget https://raw.githubusercontent.com/dominodatalab/workspace-configs/2019q4-v1/Jupyterlab/start.sh &&
       chmod 777 /var/opt/workspaces/Jupyterlab/start.sh

   # Install and enable jupyter-server-proxy
   RUN pip install --upgrade jupyterlab=0.35.4 &&
       pip install nbserverproxy jupyter-server-proxy &&
       jupyter serverextension enable --py --sys-prefix nbserverproxy &&
       jupyter labextension install jupyterlab-server-proxy
   ```

2. Update the JupyterLab definition in the Pluggable Workspace Tools section of your environment.

   ```json
   jupyterlab:
       title: "JupyterLab (Beta)"
       iconUrl: "/assets/images/workspace-logos/jupyterlab.svg"
       start: [ /var/opt/workspaces/Jupyterlab/start.sh ]
       httpProxy: false
       internalPath: "{{ownerUsername}}/{{projectName}}/{{sessionPathComponent}}/\{{runId\}}/{{#if pathToOpen}}tree/{{pathToOpen}}{{/if\}}"
       port: 8888
       rewrite: false
       requireSubdomain: false
   ```

Using Jupyter Server Proxy

If you launch a JupyterLab Workspace session in an environment with Jupyter Server Proxy installed, you can start and serve additional applications as long as they are served on a different port than JupyterLab itself.

Once an additional application is started, you can access it at the following URI:
Suppose your JupyterLab session is served at:

https://app.dominodatalab.com/workspace?owner=chuckhead&projectName=demo&runId=5cef31de46e0fb00083f9708

If you then use the JupyterLab terminal to start a Dash app on port 8887 for debugging, you could open the Dash app at:

https://app.dominodatalab.com/chuckhead/demo(notebookSession/5cef31de46e0fb00083f9708/proxy/8887/

If you instead use the JupyterLab terminal to start a Bokeh app on port 5006, you could open the Bokeh app at:

https://app.dominodatalab.com/chuckhead/demo(notebookSession/5cef31de46e0fb00083f9708/proxy/5006/

With this model you can host multiple applications on different ports and expose each through your JupyterLab workspace.

5.5 Spark on Domino

Apache Spark is a fast and general-purpose cluster computing system that offers a unified analytics engine for large-scale data processing and machine learning.

Domino provides flexibility on how to use Spark. Users can dynamically provision an on-demand Spark cluster orchestrated by Domino or alternatively can connect to an existing Spark cluster that lives outside of Domino.

5.5.1 On-Demand Spark
On-Demand Spark Overview

Overview

Domino offers the ability to dynamically provision and orchestrate a Spark cluster directly on the infrastructure backing the Domino instance. This allows Domino users to get quick access to Spark without having to rely on their IT team to create and manage one for them.

Orchestrating Spark on Domino

Domino supports fully containerized execution of Spark workloads on the Domino Kubernetes cluster. Users can interact with Spark interactively through a Domino workspace or in batch mode through a Domino job as well as directly with spark-submit.

When you start a workspace or a job that uses an on-demand cluster, Domino orchestrates a cluster in Standalone mode. The master and workers are newly deployed containers and the driver is your Domino workspace or job.

Suitable use cases

The Domino on-demand Spark cluster is suitable for the following workloads:

- **Distributed machine learning**
  Easily parallelize compute heavy workloads such as distributed training or hyper-parameter tuning. Spark comes with powerful machine learning algorithms bundled in MLlib for this purpose.

- **Interactive exploratory analysis**
  Efficiently load a large data set in a distributed manner in order to explore and understand the data using familiar query techniques with Spark SQL.

- **Featurization and data transformation (for experienced Spark users)**
  Sample, aggregate, relabel, or otherwise manipulate a large data sets to make it more suitable for analysis or training.

  **Note:** Optimal performance requires a cluster with sufficient resources and a data science practitioner who is adept at tuning their Spark application and writing performant Spark transforms.

Unsuitable use cases

The following are usage patterns that are presently not suitable for on-demand Spark on Domino:

- **Stream processing pipeline**
  While Spark itself offers a robust stream processing engine, the ephemeral nature of the on-demand clusters on Domino, makes it not a great fit for long-lived stream processing applications.

  For such cases, you should consider using an externally managed Spark cluster.

- **Colocated Spark and HDFS**
  The Domino on-demand clusters do not come with an HDFS installation and are generally not suitable for collocating data and compute.
Data in Domino clusters is intended to reside outside the cluster (e.g. object store or Domino data set). For cases where it is desirable to use the cluster as long term HDFS storage, you should consider using an externally managed Spark cluster.

- **Data pipelines with strict performance SLA**
  
  While Domino orchestrates Spark on Kubernetes in a reliable way, no extensive performance tuning or optimization has been performed. The cluster configuration and default context configuration parameters may not be optimized for such workloads.

  **Note:** If you intend to explore on-demand Domino spark clusters for such workloads you should perform extensive validation and tuning of your jobs.

### Configuring prerequisites

- **Enabling Spark on your deployment**

- **Creating a base Spark cluster environment**
  
  - **Base Spark cluster environment** - default Hadoop client libraries
  
  - **Base Spark cluster environment (Advanced)** - custom Hadoop client libraries

- **Preparing your PySpark compute environment**

  - **PySpark compute environment** - default Hadoop client libraries

  - **PySpark compute environment (Advanced)** - custom Hadoop client libraries

Before you can start using on-demand Spark clusters on Domino you need to ensure that this functionality is enabled and properly configured on your deployment.

**Note:** Domino on-demand Spark functionality is available starting with Domino 4.2.

### Enabling Spark on your deployment

Domino Administrators need to:

- **Enable on-demand Spark functionality**

  Set `ShortLived.SparkClustersEnabled` feature flag to `true`.

- **Enable new workspace experience**

  Set `ShortLived.UseNewWorkspaceChrome` to `true`. This enables the new *Domino workspace* experience which allows for easy access to the Spark Web UI.

### Creating a base Spark cluster environment

By default, Domino does not come with a Spark compatible *Compute Environment* that can be used for the components of the cluster. Without at least one such environment available, you will not be able to create a cluster.
Note that when using on-demand Spark in Domino you will have two separate environments - one for the Spark cluster and one for the workspace/job.

To create a new base Spark cluster environment, you will follow the general Environment Management with the following Environment attributes.

- **Base image**

  select the Custom Image option and specify an image URI that points to a deployable Spark image.

  It is recommended that you use the latest specific release tag for your desired version of Spark. For example, a suitable recent version would be represented by `bitnami/spark:2.4.6-debian-10-r14`.

  **Note:** Minimum image tag

  When using Spark 2.4.5, you need to use image `bitnami/spark:2.4.5-debian-10-r136` or a more recent revision. This will ensure proper compatibility for S3 access.

5.5. Spark on Domino
Domino’s on-demand Spark functionality has been developed and tested using open-source Spark images from Bitnami. While not been explicitly verified, it may be possible to use a different base image, as long as that image is compatible the Bitnami Spark Helm Chart.

For more information on the benefits, see the Why use Bitnami images section of the Bitnami image distribution page.

- **Supported clusters**
  
  Select the Domino managed Spark option (REQUIRED). This will ensure that the environment will be available for use when creating Spark clusters from workspaces and jobs.

- **Visibility**
  
  You can set this attribute the same way you would for any other Compute Environment based on your desired visibility.

- **Dockerfile Instructions**
  
  Leave blank to use the default Hadoop client libraries or follow the instructions for custom Hadoop client libraries.

  You can further add to this section to include any additional dependencies (JARs and packages) that should be available on the cluster nodes of any cluster.

  To learn more, refer to Managing dependencies

- **Pluggable Notebooks / Workspace Sessions**
  
  This section should remain blank as the Spark base environments are not intended to also include notebook configuration.

**Base Spark cluster environment - default Hadoop client libraries**

Leave the Docker Instructions section blank, if you want a thin base image that only contains core Spark with the default Hadoop client libraries. Currently both Spark 2.4.x and Spark 3.0.0 come with Hadoop 2.7.

**Base Spark cluster environment (Advanced) - custom Hadoop client libraries**

In some cases the Hadoop client libraries pre-bundled with your desired Spark version may not be appropriate for your needs. This would typically be the case if you want to utilize cloud object store connector improvements introduced post Hadoop 2.7.

Add the following to the Docker Instructions section adjusting for the desired Spark and Hadoop version.

```bash
### need if using the recommended Bitnami base image
USER root

### Make sure wget is available
RUN apt-get update && apt-get install -y wget && rm -r /var/lib/apt/lists /var/cache/
    →apt/archives

### Modify the Hadoop and Spark versions below as needed.
### NOTE: The HADOOP_HOME and SPARK_HOME locations should not be modified
ENV HADOOP_VERSION=2.9.2
ENV HADOOP_HOME=/opt/bitnami/hadoop
ENV HADOOP_CONF_DIR=/opt/bitnami/hadoop/etc/hadoop
```

(continues on next page)
Preparing your PySpark compute environment

In addition to the base Spark cluster environment, you also need to configure the PySpark compute environments for workspaces and/or jobs that will connect to your cluster.

You can either enhance the Docker Instructions section of an existing environment or create a new environment that uses an existing environment as its base.

5.5. Spark on Domino 309
PySpark compute environment - default Hadoop client libraries

The Bitnami base images referenced above come pre-built with Hadoop. For Spark 2.4.X and Spark 3.0.0, the pre-built Hadoop version currently is 2.7. If this is appropriate for your needs, you can use the simplified configuration instructions below to install and configure PySpark.

**Note:** PySpark 2 does not support Python 3.8 or higher. Build PySpark 2 compute environments from images with Python before 3.8.

**Note:** The Spark version below should match the Spark version used when creating the *Spark-enabled compute environment* above. In the example here, this version is 2.4.6 but should be adjusted as Spark evolves.

```bash
### Clear any existing PySpark install that may exist
### Omit if you know the environment does not have PySpark
RUN pip uninstall pyspark &>/dev/null

### Install PySpark matching the Spark version of your base image
### Modify the version below as needed
RUN pip install pyspark==2.4.6

### Set SPARK_HOME on the driver to point to the version installed by pyspark
RUN \
    SPARK_HOME=$(pip show pyspark | grep "Location" | awk '{print $2}')/pyspark & & \
    chown -R ubuntu:ubuntu $({SPARK_HOME}) & & \
    echo "export SPARK_HOME=${SPARK_HOME}" >> /home/ubuntu/.domino-defaults & & \
    echo "export PATH=$PATH:${SPARK_HOME}/bin" >> /home/ubuntu/.domino-defaults

### Optionally copy spark-submit to spark-submit.sh to be able to run from Domino jobs
RUN spark_submit_path=$(which spark-submit) & & \
    cp $({spark_submit_path}) ${spark_submit_path}.sh

### Hadoop 2.7 does not come with the required binaries for AWS access so we add them
### hadoop-aws.jar must match the hadoop-common.jar version of your install
RUN \
    SPARK_HOME=$(pip show pyspark | grep "Location" | awk '{print $2}')/pyspark & & \
    rm -rf $({SPARK_HOME})/hadoop-aws* & & \
    rm -rf $({SPARK_HOME})/aws-java-sdk* & & \
    curl https://repo1.maven.org/maven2/org/apache/hadoop/hadoop-aws/2.7.3/hadoop-aws-2.7.3.jar --output $({SPARK_HOME})/jars/hadoop-aws-2.7.3.jar & & \
    curl https://repo1.maven.org/maven2/com/amazonaws/aws-java-sdk/1.7.4/aws-java-sdk-1.7.4.jar --output $({SPARK_HOME})/jars/aws-java-sdk-1.7.4.jar
```

PySpark compute environment (Advanced) - custom Hadoop client libraries

In some cases the Hadoop libraries pre-bundled with your desired Spark version may not be appropriate for your needs. This would typically be the case if you want to utilize cloud object store connector improvements introduced post Hadoop 2.7.

You can follow the instructions below to configure your environment with PySpark and a custom Hadoop client libraries version.
RUN mkdir -p /opt/domino

### Modify the Hadoop and Spark versions below as needed.

ENV HADOOP_VERSION=2.9.2
ENV HADOOP_HOME=/opt/domino/hadoop
ENV HADOOP_CONF_DIR=/opt/domino/hadoop/etc/hadoop
ENV SPARK_VERSION=2.4.6
ENV SPARK_HOME=/opt/domino/spark
ENV PATH="$PATH:$SPARK_HOME/bin:$HADOOP_HOME/bin"

### Enable this for access to ADLS Gen2 when using Hadoop 3.2+
### ENV HADOOP_OPTIONAL_TOOLS=hadoop-azure

### Install the desired Hadoop-free Spark distribution

RUN rm -rf $SPARK_HOME &&
    tar -xf spark-$SPARK_VERSION/bin-without-hadoop.tgz &&
    mv spark-$SPARK_VERSION/bin-without-hadoop $SPARK_HOME &&
    chmod -R 777 $SPARK_HOME/conf

### Install the desired Hadoop libraries

RUN rm -rf $HADOOP_HOME &&
    hadoop-$HADOOP_VERSION.tar.gz &&
    tar -xf hadoop-$HADOOP_VERSION.tar.gz &&
    mv hadoop-$HADOOP_VERSION $HADOOP_HOME

### Setup the Hadoop libraries classpath and Spark related envvars for proper init in Domino

RUN echo "export SPARK_HOME=$SPARK_HOME" >> /home/ubuntu/.domino-defaults
RUN echo "export HADOOP_HOME=$HADOOP_HOME" >> /home/ubuntu/.domino-defaults
RUN echo "export HADOOP_CONF_DIR=$HADOOP_CONF_DIR" >> /home/ubuntu/.domino-defaults
RUN echo "export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HADOOP_HOME/lib/native" >> /home/ubuntu/.domino-defaults
RUN echo "export PATH=$PATH:$SPARK_HOME/bin:$HADOOP_HOME/bin" >> /home/ubuntu/.domino-defaults

### Complete the PySpark setup from the Spark distribution files

WORKDIR $SPARK_HOME/python

RUN python setup.py install

### Optionally copy spark-submit to spark-submit.sh to be able to run from Domino jobs

RUN spark_submit_path=$(which spark-submit) &&
    cp $spark_submit_path $spark_submit_path.sh

### Hadoop 2.7 does not come with the required binaries for AWS access so we add them
### hadoop-aws.jar must match the hadoop-common.jar version of your install

RUN \
    SPARK_HOME=$(pip show pyspark | grep "Location" | awk '{print $2}')/pyspark &&
    rm -rf $SPARK_HOME/hadoop-aws* &&
    rm -rf $SPARK_HOME/aws-java-sdk* &&
    curl https://repo1.maven.org/maven2/org/apache/hadoop/hadoop-aws/2.7.3/hadoop-aws-2.7.3.jar --output $SPARK_HOME/jars/hadoop-aws-2.7.3.jar

(continues on next page)
### Optionally install boto3 which can help working with AWS credential file profiles
### Can omit if not needed

```
RUN pip install boto3
```

**Working with your cluster**

**Creating a cluster with workspaces**

To create an on-demand Spark cluster attached to a Domino Workspace, click New Workspace from the Workspaces menu. On the Launch New Workspace dialog select the option to Attach Cluster. Specify the desired cluster settings and launch your workspace. Once the workspace is up, it will have access to the Spark cluster you configured.

The Hardware Tier for your workspace will determine the compute resources available to your Spark driver process.

---

**Creating a cluster with jobs**

Similarly to workspaces, to create an on-demand Spark cluster attached to a Domino job, click on Run from the Jobs menu. One the Start a Job dialog select the option to Attach Cluster. Specify the desired cluster settings and launch your job. The job will have access to the Spark cluster you configured.
As your command, you can use any Python script that contains a PySpark job.

You can also submit jobs using `spark-submit` but since it is not recognized automatically as one of the Domino supported job types you will need to wrap it with a shell script unless you included a copy as `spark-submit.sh` as part of preparing your compute environment.

Below is an example of a simple wrapper `my-spark-submit.sh`

```
#!/usr/bin/env bash

spark-submit $@
```

**Understanding your cluster settings**

Domino makes it simple to specify key settings when creating a Spark cluster.
**Number of Executors**

Number of Executors that will be available to your Spark application.

When you instantiate Spark context with the default settings, the `spark.executor.instances` Spark setting will be set to the number specified in the above dialog.

**Quota Max**

The maximum number of executors that you can make available to your cluster is limited by the number of per-user executions that your Domino administrator has configured for your deployment.

In addition to the number of Spark application executors, you will need 1 slot for your cluster master and 1 slot for your workspace or job.

**Executor Hardware Tier**

The amount of compute resources (CPU and memory) that will be made available to Spark executors.

When you instantiate Spark context with the default settings, the `spark.executor.cores` and `spark.executor.memory` Spark settings will be set to the values corresponding to the Hardware Tier.

For cores, the number of cores will be rounded up to the nearest integer. For example, a HW Tier with 1.5 cores will result in `spark.executor.cores` set to 2 when creating the default context.

**Master Hardware Tier**

Same mechanics as the Executor Hardware Tier, but applied to the resources that will be available for your Spark cluster master.

The master in the Domino configuration has a pretty limited role in Spark application scheduling, and it does not need a significant amount of resources.

**Cluster Compute Environment**

Designates your environment. Projects already using this environment will be allowed to continue to use it until a new project environment is set.
• **Dedicated local storage per executor**

  The amount of dedicated storage in Gigabytes ($2^{30}$ bytes) that will be available to each executor.

  The storage will be automatically mounted to `/tmp` which is the default location for `spark.local.dir` that serves as the location for RDDs, when they need to be on disk, and for shuffle data.

  The storage will be automatically provisioned when the cluster is created and de-provisioned when it is shut down.

  **Warning:** The local storage per executor should not be used for storing any data which needs to be available after the cluster is shut down.

---

Advanced cluster configuration topics

Spark executors vs workers

Domino gives you the ability to have flexible resource allocation through Hardware Tiers which makes it convenient to plan your cluster resources needed for your application in terms of executors as opposed to workers. When creating on-demand Spark clusters, Domino will automatically manage the relationship between Spark executors and workers (see Spark definitions). By default, there will be one worker for each executor.

Resource overhead

When setting up worker and executor memory limits, Domino ensures that the worker containers have additional memory on top of the Hardware Tier memory definition to account for the recommended overhead of 10% of executor memory with a minimum of 384MB. The usable executor memory will always be based on the value from the actual Hardware Tier.

You may need to take this into account when designing Hardware Tiers to achieve optimal packing of Spark worker containers on the physical instances of the Domino Kubernetes cluster.

Because of the worker memory overhead and the approach of using 1 worker per executor, it is possible that using lots of executors with smaller Hardware Tiers may result in excessive memory overhead.

If this is undesirable, an experienced data scientist can chose a large Hardware Tier when creating the cluster and then overwrite the executor core and memory settings to achieve the desired effect. See *Adding and overwriting parameters* for details on how to make such changes.

Connecting to your cluster

When provisioning your on-demand Spark cluster, Domino sets up key default cluster configuration parameters (e.g. `spark.master`, `spark.driver.host`) to the appropriate cluster URLs so that creating `SparkSession` or `Spark-Context` with default context will properly connect the cluster to your workspace or job.

Creating SparkSession

You can create `SparkSession` using the following:
from pyspark.sql import SparkSession

spark = SparkSession(
    .builder(
        .appName("MyAppName")
    ).getOrCreate()

# You can examine the full config
spark.sparkContext.getConf().getAll()

Creating SparkContext

Alternatively, you can create SparkContext using the following:

from pyspark import SparkContext

conf = SparkConf().setAppName(appName).setMaster(master)
sc = SparkContext(conf=conf)

Adding and overwriting parameters

It is possible to add additional config parameters or overwrite any of the defaults by manipulating your context configuration.

from pyspark.sql import SparkSession

spark = SparkSession(
    .builder(
        .appName("MyAppName")
    ).config("spark.some.config.option", "some-value")
    .getOrCreate()

You can also supply additional configuration settings that will apply for any clusters associated with a Domino project from Settings->Integrations->Domino managed on-demand cluster

![Project settings](image-url)

- Domino managed on-demand cluster

  Spark Configuration Options

  spark.driver.memory 2g
  spark.driver.maxResultSize 2g

  Each configuration option consists of a key-value pair separated by whitespace. Configuration options are separated by a new line. Learn more about dynamically loading Spark properties.
Warning: You should not overwrite the default configuration values for `spark.master`, `spark.driver.host` since you may lose proper connectivity between your cluster and workspace or job.

Accessing the Spark UI

Spark provides a suite of web user interfaces that you can use to monitor the status and resource consumption of your Spark cluster.

Domino makes the Spark Web UI available for active on-demand clusters attached to both workspaces and jobs.

Spark UI from Workspaces

The Spark UI is available from a dedicated tab in your workspace.

Spark UI from Jobs

The Spark UI is also available for running jobs from Details tab.
Cluster lifecycle

On workspace or job startup, a Domino on-demand Spark cluster with the desired cluster settings is automatically provisioned and attached to the workspace or job as soon as the cluster becomes available.

On workspace or job termination, the on-demand Spark cluster and all associated resources are automatically terminated and de-provisioned. This includes any compute resources and storage allocated for the cluster.

Cluster network security

The on-demand Spark clusters created by Domino are not meant for sharing between multiple users. Each cluster is associated with a given workspace or a job instance. Access to the cluster and the Spark Web UI is restricted only to users who can access the workspace or the job attached to it. This restriction is enforced at the networking level and the cluster is only reachable from the execution that provisioned it.

Managing dependencies

In a shared Spark cluster, it can be challenging for teams to manage their dependencies (e.g. Python packages or JARs). Installing every dependency that a Spark application may need before it runs and dealing with version conflicts can be complex and time-consuming.

Domino allows you to easily package and manage dependencies as part of your Spark-enabled compute environments. This approach creates the flexibility to manage dependencies for individual projects or workloads without having to deal with the complexity of a shared cluster.

To add a new dependency, add the appropriate statements in the Docker Instructions section of the relevant Spark and execution compute environments.

For example to add numpy you would want to include the following.

```bash
### Optionally specify version if desired
RUN pip install numpy
```

Working with data

- Overview
- Using Domino datasets
- Using S3
  - S3 Usage Examples
- Using Azure Data Lake Storage Gen2
- Access AWS Resources from a Spark Cluster

Overview

When using a Domino on-demand Spark cluster any data that will be used, created, or modified as part of the interaction needs to go into an external data store.
Note: On-demand Spark clusters are not intended as a permanent store of any data or collocating a big data layer such as HDFS. Any data that is not stored externally from the cluster will be lost upon termination.

Using Domino datasets

When you create a Spark cluster attached to a Domino workspace or job, any Domino dataset accessible from the workspace or job will also be accessible from all components of the cluster under the same dataset mount path. Data can be accessed using the file:// path prefix.

For example, to read a file you would use the following.

```python
rdd = sc.textFile("file:///path/to/file")
```

No additional configuration of the Spark cluster environment or the execution environment is required.

Using S3

In order to enable working with data in Amazon S3 (or S3 compatible object store) you need to ensure that your base Spark cluster environment and compatible PySpark compute environment are configured with the Hadoop-AWS module.

The environments created when configuring prerequisites will at a minimum include Hadoop 2.7.3 client libraries which are sufficient for basic access. A number of additional commonly used features (e.g. temporary credentials, SSE-KMS encryption, more efficient committers, etc) are only available in more recent Hadoop-AWS module versions.

Consult the documentation for the relevant version to determine what may be the best fit for you.

- Hadoop-AWS Module 2.7.3
- Hadoop-AWS Module 2.8.5
- Hadoop-AWS Module 2.9.2
- Hadoop-AWS Module 3.1.3
- Hadoop-AWS Module 3.2.1

For Spark 2.4.x, a good advanced option would be Hadoop 2.9.2.

S3 Usage Examples

Now that you have your environments properly setup, you can interact with S3. Below are several common access patterns.

Access bucket with AWS credentials in environment variables

```python
import os
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()

# the default configuration will pick up your credentials from environment variables
# No additional configuration is necessary
```

(continues on next page)
# test reading
def = spark.read.json("s3a://bucket/prefix1/prefix2/people.json")
df.show()

Access bucket with SSE-KMS encryption

**Note:** Requires Hadoop-AWS 2.9.2+

```python
import os
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()

# for write operations you will need the ARN of the key to use
# Note that the credentials used need to have proper access to use the key
kms_key_arn = "<your key ARN here>"

# configure the connector
# This example assumes credentials from environment variables so no need to configure
# Note: The encryption config is not needed for read only operations
hadoop_conf = spark.sparkContext._jsc.hadoopConfiguration()
hadoop_conf.set("fs.s3a.server-side-encryption-algorithm", "SSE-KMS")
hadoop_conf.set("fs.s3a.server-side-encryption-key", kms_key_arn)

# test reading
def = spark.read.json("s3a://bucket/prefix1/prefix2/people.json")
df.show()

# test writing
df.write.mode("overwrite").parquet("s3a://bucket/prefix1/prefix2/write-test/output")
```

Access a bucket with Domino assumed temporary credentials

**Note:** Requires Hadoop-AWS 2.9.2+

It is important that no AWS credential variables are set in your user profile or project

```python
import os
from pyspark.sql import SparkSession

try:
    spark.stop()
except:
    pass
spark = SparkSession.builder.getOrCreate()

# The name of one of the roles you are entitled to
profile_name="my-role-name-read-write"

# use boto3 for convenience to get credentials form credentials file populated by Domino
# can use any method desirable to extract the credentials
```
import boto3
role_creds = boto3.Session(profile_name=profile_name).get_credentials().get_frozen_credentials()

# configure the connector
# Use the TemporaryAWSCredentialsProvider
hadoop_conf = spark.sparkContext._jsc.hadoopConfiguration()
hadoop_conf.set("fs.s3a.aws.credentials.provider", "org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider")
hadoop_conf.set("fs.s3a.access.key", role_creds.access_key)
hadoop_conf.set("fs.s3a.secret.key", role_creds.secret_key)
hadoop_conf.set("fs.s3a.session.token", role_creds.token)

# test reading
df = spark.read.json("s3a://bucket/prefix1/prefix2/people.json")
df.show()

# test writing
df.write.mode("overwrite").parquet("s3a://bucket/prefix1/prefix2/write-test/output")

For full set of configuration options refer to the documentation for the Hadoop-AWS module.

**Using Azure Data Lake Storage Gen2**

In order to enable working with data in Azure Data Lake Storage (ADLS) Gen2 you need to configure your base Spark environment and your compute environment with the Hadoop-Azure ABFS connector.

The ABFS connector requires Hadoop 3.2+

To accomplish this set SPARK_VERSION=3.0.0 and HADOOP_VERSION=3.2.1 when following the advanced instructions for base Spark cluster environment and compatible PySpark compute environment.

**Note:** It is also required that you enable the ENV HADOOP_OPTIONAL_TOOLS=hadoop-azure directive in your environments.

**Access AWS Resources from a Spark Cluster**

**Note:** This feature requires Hadoop 2.9.2 or higher. If you’re using an earlier version of Hadoop, you’ll need to configure your base Spark environment or your PySpark environment to use Hadoop 2.9.2+

You can configure on-demand Spark clusters in your Domino workspace to access AWS resources using temporary credentials issued by AWS. To do this, your Domino deployment must use single sign-on (SSO) with a trusted identity provider (IdP). The credentials can also be continuously refreshed, allowing your Spark cluster to have continuous access to AWS resources.

The specific credentials (and associated privileges) issued by AWS to your Spark cluster are based on role profiles defined in your IdP by a Domino administrator. These profiles include identity attributes used by AWS to issue appropriate temporary credentials corresponding to a role profile. The temporary credentials are then automatically distributed to your on-demand Spark cluster.

5.5. Spark on Domino

321
For more details on this credential propagation architecture, please refer to the AWS Credential Propagation section in the Domino Administrator’s guide.

To take advantage of this feature, you can either (1) configure your Spark context dynamically to work with profile role credentials in your code, or (2) configure the desired profile in your project settings. Both methods are described below and you should select the option that best matches your use case.

**Configure your Spark context dynamically in your code**

This method provides you with more flexibility and is recommended if you need to frequently change role profiles. Recall that your Spark clusters must use Hadoop 2.9.2 or higher and may need to be configured accordingly prior to implementing the following code snippet.

```python
import os
from pyspark.sql import SparkSession

try:
    spark.stop()
except:
    pass

# First, set the AWS_PROFILE environment variable to the name of the profile found in the credentials file $AWS_SHARED_CREDENTIALS_FILE.
# If you’re doing this in a notebook, first stop your Spark session or context for the change to take effect.

os.environ['AWS_PROFILE'] = 'name-of-profile-to-use'

# Next, configure the Spark connector by setting up the provider type and the name of the profile to use. Be sure to replace the .appName() argument with the name of your app.

spark = SparkSession.builder \
    .appName("Credential Spark Test") \
    .config("spark.hadoop.fs.s3a.aws.credentials.provider", "com.amazonaws.auth.profile.ProfileCredentialsProvider") \
    .config("spark.executorEnv.AWS_PROFILE", os.environ['AWS_PROFILE']) \
    .getOrCreate()

# Read some data from AWS (replace with your S3 URI)

df = spark.read.json("s3a://foobar/bazbux.json")

df.show()
```

**Configure your Spark context from your Domino project settings**

You can also enable this feature by adding Spark configuration options in your Domino project settings. This method provides less flexibility and is recommended for projects that will utilize one consistent role profile. To enable this feature:

1. Navigate to “Settings” in your Domino project.
2. Click on the “Integrations” tab.
3. In the “Apache Spark mode” section, select “Domino managed on-demand cluster”.
4. In the “Spark Configuration Options” text area, add the keys and values specified below. Ensure one whitespace between the key and the value.

```
spark.hadoop.fs.s3a.aws.credentials.provider com.amazonaws.auth.profile.
    ProfileCredentialsProvider
spark.executorEnv.AWS_PROFILE name-of-profile-to-use
```
5. Navigate to your Domino account settings and click (or scroll to) “User Environment Variables”. Under “Set user environment variable”, set “Name” to `AWS_PROFILE` and set “Value” to the name of the profile you’d like to use (name-of-profile-to-use in the previous step). Click Set Variable.

5.5.2 External Hadoop and Spark

Hadoop and Spark overview
Overview

Apache Hadoop is a collection of open source cluster computing tools that supports popular applications for data science at scale, such as Spark.

You can interact with Hadoop from your Domino executors by configuring your Domino environment with the necessary software dependencies and credentials. Domino supports most providers of Hadoop solutions, including MapR, Cloudera, and Amazon EMR. Once a Domino environment is set up to connect to your cluster, Domino projects can use the environment to work with Hadoop applications.

For a thorough video introduction to using Spark in Domino, watch the webinar recording:

- Using Apache Spark with Domino

Using a Hadoop-enabled environment in your Domino project

If your Domino administrators have already created an environment for connecting to a Hadoop cluster, you can follow these subsections of the setup instructions to use that environment in your Domino project.

For users setting up projects to work with an existing environment, read these subsections:

- Configuring a Domino project for use with a Cloudera CDH5 cluster
- Configuring a Domino project for use with an Amazon EMR cluster
- Configuring a Domino project for use with a MapR cluster
- Configuring a Domino project for use with a Hortonworks cluster
Once your project is set up to use the environment, you can execute code in your Domino Runs that connects to the cluster for Spark, HDFS, or Hive functionality.

Setting up Domino to connect to a new Hadoop cluster

To connect to your existing Hadoop cluster from Domino, you must create a Domino environment with the necessary dependencies installed. Some of these dependencies, including binaries and configuration files, will come directly from the cluster itself. Others will be external software dependencies like Java and Spark, and you will need to match the version you install in the environment to the version running on the cluster.

The basic steps for setting up an environment to connect to your cluster are:

1. Gather binaries and configuration files from your cluster
2. Gather dependencies from external sources, like Java JDKs and Spark binaries
3. Upload all dependencies to a Domino project, to make them accessible to the Domino environment builder
4. Author a new Domino environment that pulls from the Domino project, then installs and configures all required dependencies
For Domino admins setting up a Domino environment to connect to a new cluster, read the full provider-specific setup guides:

- Connecting to a Cloudera CDH5 cluster from Domino
- Connecting to an Amazon EMR cluster from Domino
- Connecting to a MapR cluster from Domino
- Connecting to a Hortonworks cluster from Domino

Additional capabilities

Domino also supports running Spark on a Domino executor in local mode, querying Hive tables with JDBC, and authenticating to clusters with Kerberos. See the following guides for more information.

- Kerberos authentication
- Running local Spark on a Domino executor
- Interactive PySpark notebooks

Using Apache Spark with Domino

The below video is a recording of a webinar titled *Using Apache Spark in Domino*, held in May 2019. The video covers how Domino interacts with Spark clusters, essential Domino features for supporting Spark, and how to handle some common Spark use cases and workflows in Domino.

Click here to view or download the slides used in the presentation.
For additional information and guides on setting up Domino for use with Spark, read the Hadoop and Spark Overview.

Connecting to a Cloudera CDH5 cluster from Domino

- **Overview**
- Gathering the required binaries and configuration files
- Uploading the binaries and configuration files to Domino
- Creating a Domino environment for connecting to CDH5
- Configure a Domino project for use with a CDH5 cluster

**Overview**

Domino supports connecting to a Cloudera CDH5 cluster through the addition of cluster-specific binaries and configuration files to your Domino environment.

At a high level, the process is as follows:

1. Connect to your CDH5 edge or gateway node and gather the required binaries and configuration files, then download them to your local machine.
2. Upload the gathered files into a Domino project to allow access by the Domino environment builder.
3. Create a new Domino environment that uses the uploaded files to enable connections to your cluster.
4. Enable YARN integration for the Domino projects that you want to use with the CDH5 cluster.

Domino supports the following types of connections to a CDH5 cluster:

- FS shell
- spark2-shell
- spark2-submit
- pyspark
- YARN shell
Gathering the required binaries and configuration files

You will find most of the necessary files for setting up your Domino environment on your CDH5 edge or gateway node. To get started, connect to the edge node via SSH, then follow the steps below.

1. Create a directory named `hadoop-binaries-configs` at `/tmp`.
   ```bash
   mkdir /tmp/hadoop-binaries-configs
   ```

2. Create the following subdirectories inside `/tmp/hadoop-binaries-configs/`.
   ```bash
   mkdir /tmp/hadoop-binaries-configs/configs
   mkdir /tmp/hadoop-binaries-configs/parcels
   ```

3. (Optional) If your cluster uses Kerberos authentication, create the following subdirectory in `/tmp/hadoop-binaries/configs/`.
   ```bash
   mkdir /tmp/hadoop-binaries-configs/kerberos
   ```
   Then, copy the `krb5.conf` Kerberos configuration file from `/etc/` to `/tmp/hadoop-binaries-configs/kerberos`.
   ```bash
   cp /etc/krb5.conf /tmp/hadoop-binaries-configs/kerberos/
   ```

4. Copy the CDH and SPARK2 directories from `/opt/cloudera/parcels/` to `/tmp/hadoop-binaries-configs/parcels/`. These directories will have a version number appended to their names, so complete the appropriate directory name in the commands shown below.
   ```bash
   cp -R /opt/cloudera/parcels/CDH-<version>/ /tmp/hadoop-binaries-configs/parcels/
   cp -R /opt/cloudera/parcels/SPARK2-<version>/ /tmp/hadoop-binaries-configs/parcels/
   ```

5. Copy the hadoop, hive, spark, and spark2 directories from `/etc/` to `/tmp/hadoop-binaries-configs/configs/`.
   ```bash
   cp -R /etc/hadoop /tmp/hadoop-binaries-configs/configs/
   cp -R /etc/hive /tmp/hadoop-binaries-configs/configs/
   cp -R /etc/spark2 /tmp/hadoop-binaries-configs/configs/
   cp -R /etc/spark /tmp/hadoop-binaries-configs/configs/
   ```

6. On the edge node, run the following command to identify the version of Java running on the cluster.
   ```bash
   java -version
   ```
   You should then download a JDK .tar file from the Oracle downloads page that matches that version. The filename will have a pattern like the following.
   ```bash
   jdk-8u211-linux-x64.tar.gz
   ```
Keep this JDK handy on your local machine for use in a future step.

7. Compress the /tmp/hadoop-binaries-configs/ directory to a gzip archive.

```
cd /tmp
tar -zcf hadoop-binaries-configs.tar.gz hadoop-binaries-configs
```

When finished, use SCP to download the archive to your local machine.

8. Next, you’ll need to extract the archive on your local machine, add a java subdirectory, then add the JDK .tar file you downloaded earlier to the java subdirectory.

```
tar xzf hadoop-binaries-configs.tar.gz
mkdir hadoop-binaries-configs/java
cp jdk-8u211-linux-x64.tar.gz hadoop-binaries-configs/java/
```

9. When finished, your hadoop-binaries-configs directory should have the following structure.

```
hadoop-binaries-configs/
  ├── configs/
  │   ├── hadoop/
  │   ├── hive/
  │   ├── spark/
  │   └── spark2/
  ├── java/
  │   └── jdk-8u211-linux-x64.tar.gz
  └── parcels
      ├── CDH-version/
      └── SPARK-version/
```

10. If your directory contains all the required files, you can now compress it to a gzip archive again in preparation for uploading to Domino in the next step.

```
tar -zcf hadoop-binaries-configs.tar.gz hadoop-binaries-configs
```

---

**Uploading the binaries and configuration files to Domino**

Use the following procedure to upload the archive you created in the previous step to a public Domino project. This will make the file available to the Domino environment builder.

1. Log in to Domino, then create a new public project.
2. Open the **Files** page for the new project, then click to **browse for files** and select the archive you created in the previous section. Then click **Upload**.

3. Once the archive has been uploaded, click the gear menu next to it on the Files page, then right click **Download** and click **Copy Link Address**. Save the copied URL in your notes, as you will need it in the next step.

   Once you have recorded the download URL of the archive, you’re ready to build a Domino environment for connecting to your CDH5 cluster.
Creating a Domino environment for connecting to CDH5

1. Click **Environments** from the Domino main menu, then click **Create Environment**.

2. Give the environment an informative name, then choose a base environment that includes the version of Python that is installed on the nodes of your CDH5 cluster. Most Linux distributions ship with Python 2.7 by default, so you will see the Domino Analytics Distribution for Python 2.7 used as the base image in the following examples. Click **Create** when finished.

3. After creating the environment, click **Edit Definition**. Copy the below example into your Dockerfile Instructions, then be sure to edit it wherever necessary with values specific to your deployment and cluster.

   In this Dockerfile, wherever you see a hyphenated instruction enclosed in carats like `<paste-your-domino-download-url-here>`, be sure to replace it with the corresponding value you recorded in previous steps.

   You may also need to edit commands that follow to match downloaded filenames.

   ```
   USER root
   ```

(continues on next page)
# Give user ubuntu ability to sudo as any user including root
RUN echo "ubuntu ALL=(ALL:ALL) NOPASSWD: ALL" >> /etc/sudoers

# Set up directories
RUN mkdir -p /opt/cloudera/parcels &&
    mkdir /tmp/domino-hadoop-downloads &&
    mkdir /usr/java

# Download the binaries and configs gzip you uploaded to Domino.
# This downloaded gzip file should have the following
# - CDH and Spark2 parcel directories in a 'parcels' sub-directory.
# - java installation tar file in 'java' sub-directory
# - krb5.conf in 'kerberos' sub-directory
# - hadoop, hive, spark configuration directories a 'configs' sub-directory
RUN wget --no-check-certificate <paste-your-domino-download-url-here> -O /tmp/
    domino-hadoop-downloads/hadoop-binaries-configs.tar.gz &&
    tar xzf /tmp/domino-hadoop-downloads/hadoop-binaries-configs.tar.gz -C /tmp/
    domino-hadoop-downloads/

# Install kerberos client and update the kerberos configuration file
RUN apt-get -y install krb5-user telnet &&
    cp /tmp/domino-hadoop-downloads/hadoop-binaries-configs/kerberos/krb5.conf /etc/krb5.conf

# Install version of Java that matches hadoop cluster and update environment
# variables
# Note that your JDK may have a different filename depending on your cluster's
# version of Java
RUN tar xvf /tmp/domino-hadoop-downloads/hadoop-binaries-configs/java/jdk-8u162-
    linux-x64.tar -C /usr/java
ENV JAVA_HOME=/usr/java/jdk1.8.0_162
RUN echo "export JAVA_HOME="/usr/java/jdk1.8.0_162" >> /home/ubuntu/.domino-
    defaults &&
    echo "export PATH=\$JAVA_HOME/bin:\$PATH" >> /home/ubuntu/.domino-defaults

# Install CDH hadoop-client binaries from cloudera ubuntu trusty repository.
# This example shows client binaries for CDH version 5.15 here.
# Update these commands with the CDH version that matches your cluster.
    cdh trusty-cdh5.15.0 contrib" >> /etc/apt/sources.list.d/cloudera.list &&
    echo "deb-src http://archive.cloudera.com/cdh5/ubuntu/trusty/amd64/cdh trusty-
    cdh5.15.0 contrib" >> /etc/apt/sources.list.d/cloudera.list &&
    wget http://archive.cloudera.com/cdh5/ubuntu/trusty/amd64/cdh/archive.key -O /
    tmp/domino-hadoop-downloads/archive.key &&
    apt-key add /tmp/domino-hadoop-downloads/archive.key &&
    apt-get update &&
    apt-get -y -t trusty-cdh5.15.0 install zookeeper &&
    apt-get -y -t trusty-cdh5.15.0 install hadoop-client

# Copy CDH and Spark2 parcels to correct directories and update symlinks
# Note that the version strings attached to your directory names may be different
# than the below examples.
RUN mv /tmp/domino-hadoop-downloads/hadoop-binaries-configs/parcels/CDH-5.15.0-1.
    cdh5.15.0.p0.21 /opt/cloudera/parcels/ &&
    mv /tmp/domino-hadoop-downloads/hadoop-binaries-configs/parcels/SPARK2-2.3.0.
    cloudera3-1.cdh5.13.3.p1.458809 /opt/cloudera/parcels/ &&
    ln -s /opt/cloudera/parcels/CDH-5.15.0-1.cdh5.15.0.p0.21 /opt/cloudera/
    parcels/CDH &&
    (continues on next page)
# Copy hadoop, hive and spark2 configurations

```bash
RUN ln -s /opt/cloudera/parcels/SPARK2-2.3.0.cloudera3-1.cdh5.13.3.p0.458809 /opt/
    → cloudera/parcels/SPARK2

# Copy hadoop, hive and spark2 configurations
RUN mv /etc/hadoop /tmp/domino-hadoop-downloads/hadoop-binaries-configs/configs/
    → hadoop-etc-local.backup &&
    mv /tmp/domino-hadoop-downloads/hadoop-binaries-configs/configs/hadoop /etc/
    → hadoop &&
    mv /tmp/domino-hadoop-downloads/hadoop-binaries-configs/configs/hive /etc/
    → hive &&
    mv /tmp/domino-hadoop-downloads/hadoop-binaries-configs/configs/spark2 /etc/
    → spark2 &&
    mv /tmp/domino-hadoop-downloads/hadoop-binaries-configs/configs/spark /etc/
    → spark

# Create alternatives for hadoop configurations. Update the extensions with the _same_ strings as found in your edge node
# Example: In the command 'update-alternatives --install /etc/hadoop/conf hadoop-
# → conf /etc/hadoop/conf.cloudera.yarn 55'
# make sure that /etc/hadoop/conf.cloudera.yarn is named the same as the corresponding file on your edge node.
# Sometimes in the CDH5 edgenode, that is named something like /etc/hadoop/conf.
    → cloudera.yarn
RUN update-alternatives --install /etc/hadoop/conf hadoop-conf /etc/hadoop/conf.
    → cloudera.yarn 55 &&
    update-alternatives --install /etc/hive/conf hive-conf /etc/hive/conf.
    → cloudera.hive 55 &&
    update-alternatives --install /etc/spark2/conf spark2-conf /etc/spark2/conf.
    → cloudera.spark2_on_yarn 55 &&
    update-alternatives --install /etc/spark/conf spark-conf /etc/spark/conf.
    → cloudera.spark_on_yarn 55

# These instructions are for Spark2
# Creating alternatives for Spark2 binaries, also create symlink for pyspark
    → pointing to pyspark2
RUN update-alternatives --install /usr/bin/spark2-shell spark2-shell /opt/
    → cloudera/parcels/SPARK2/bin/spark2-shell 55 &&
    update-alternatives --install /usr/bin/spark2-submit spark2-submit /opt/
    → cloudera/parcels/SPARK2/bin/spark2-submit 55 &&
    update-alternatives --install /usr/bin/pyspark2 pyspark2 /opt/cloudera/
    → parcels/SPARK2/bin/pyspark2 55 &&
    ln -s /usr/bin/pyspark2 /usr/bin/pyspark

# Update SPARK and HADOOP environment variables. Make sure py4j file name is _→ correct per your edgenode
ENV SPARK_HOME=/opt/cloudera/parcels/SPARK2/lib/spark2
RUN echo "export HADOOP_HOME=/opt/cloudera/parcels/CDH/lib/hadoop" >> /home/
    → ubuntu/.domino-defaults &&
    echo "export HADOOP_CONF_DIR=/etc/hadoop/conf" >> /home/ubuntu/.domino-
    → defaults &&
    echo "export YARN_CONF_DIR=/etc/hadoop/conf" >> /home/ubuntu/.domino-defaults_
    → &&
    echo "export SPARK_HOME=/opt/cloudera/parcels/SPARK2/lib/spark2" >> /home/
    → ubuntu/.domino-defaults &&
    echo "export SPARK_CONF_DIR=/etc/spark2/conf" >> /home/ubuntu/.domino-
    → defaults &&
    echo "export PYTHONPATH=$SPARK_HOME/python:$SPARK_HOME/python/lib/py4j-0.10.7-
    → src.zip" >> /home/ubuntu/.domino-defaults
```

(continues on next page)
Configure a Domino project for use with a CDH5 cluster

This procedure assumes that an environment with the necessary client software has been created according to the instructions above. Ask your Domino admin for access to such an environment.

1. Open the Domino project you want to use with your CDH5 cluster, then click Settings from the project menu.

2. On the Integrations tab, click to select YARN integration from the Apache Spark panel, then click Save. You do not need to edit any of the fields in this section.

3. If your cluster uses Kerberos authentication, you can configure credentials at the user level or project level. Do so before attempting to use the environment. Note that if you followed the instructions above on creating your environment, your Kerberos configuration file has already been added to it.

4. On the Hardware & Environment tab, change the project default environment to the one with the cluster’s binaries and configurations files installed.

You are now ready to start Runs from this project that interact with your CDH5 cluster.

Connecting to a Hortonworks cluster from Domino

• Overview
Overview

Domino supports connecting to a Hortonworks cluster through the addition of cluster-specific binaries and configuration files to your Domino environment.

At a high level, the process is as follows:

1. Connect to your Hortonworks cluster edge node and gather the required binaries and configuration files, then download them to your local machine.
2. Upload the gathered files into a Domino project to allow access by the Domino environment builder.
3. Create a new Domino environment that uses the uploaded files to enable connections to your cluster.
4. Enable YARN integration for the Domino projects that you want to use with the Hortonworks cluster.

Domino supports the following types of connections to a Hortonworks cluster:

- FS shell
- spark-shell
- spark-submit
- pyspark
- YARN shell
Gathering the required binaries and configuration files

You will find most of the necessary files for setting up your Domino environment on your edge node. To get started, connect to the edge node via SSH, then follow the steps below.

1. Create a directory named `hadoop-binaries-configs` at `/tmp`.

   ```
   mkdir /tmp/hadoop-binaries-configs
   ```

2. Create the following subdirectory inside `/tmp/hadoop-binaries-configs/`.

   ```
   mkdir /tmp/hadoop-binaries-configs/configs
   ```

3. (Optional) If your cluster uses Kerberos authentication, create the following subdirectory in `/tmp/hadoop-binaries/configs/`.

   ```
   mkdir /tmp/hadoop-binaries-configs/kerberos
   ```

   Then, copy the `krb5.conf` Kerberos configuration file from `/etc/` to `/tmp/hadoop-binaries-configs/kerberos`.

   ```
   cp /etc/krb5.conf /tmp/hadoop-binaries-configs/kerberos/
   ```

4. Copy the `hadoop`, `hive`, `spark`, and `spark2` directories from `/etc/` to `/tmp/hadoop-binaries-configs/configs/`.

   ```
   cp -R /etc/hadoop /tmp/hadoop-binaries-configs/configs/
   cp -R /etc/hive /tmp/hadoop-binaries-configs/configs/
   cp -R /etc/spark2 /tmp/hadoop-binaries-configs/configs/
   cp -R /etc/spark /tmp/hadoop-binaries-configs/configs/
   ```

5. On the edge node, run the following command to identify the version of Java running on the cluster.

   ```
   java -version
   ```

   You should then download a JDK .tar file from the Oracle downloads page that matches that version. The filename will have a pattern like the following.

   ```
   jdk-8u211-linux-x64.tar.gz
   ```

   Keep this JDK handy on your local machine for use in a future step.

6. Compress the `/tmp/hadoop-binaries-configs/` directory to a gzip archive.

   ```
   cd /tmp
   tar -zcf hadoop-binaries-configs.tar.gz hadoop-binaries-configs
   ```

   When finished, use SCP to download the archive to your local machine.

7. Next, you’ll need to extract the archive on your local machine, add a java subdirectory, then add the JDK .tar file you downloaded earlier to the java subdirectory.

   ```
   tar xzf hadoop-binaries-configs.tar.gz
   mkdir hadoop-binaries-configs/java
   cp jdk-8u211-linux-x64.tar.gz hadoop-binaries-configs/java/
   ```
8. When finished, your hadoop-binaries-configs directory should have the following structure.

```
hadoop-binaries-configs/
  configs/
    hadoop/
    hive/
    spark/
    spark2/
  java/
    jdk-8u211-linux-x64.tar.gz
  kerberos/  # optional
    krb5.conf
```

9. If your directory contains all the required files, you can now compress it to a gzip archive again in preparation for uploading to Domino in the next step.

```
tar -zcf hadoop-binaries-configs.tar.gz hadoop-binaries-configs
```

---

**Uploading the binaries and configuration files to Domino**

Use the following procedure to upload the archive you created in the previous step to a public Domino project. This will make the file available to the Domino environment builder.

1. Log in to Domino, then create a new public project.
2. Open the Files page for the new project, then click to browse for files and select the archive you created in the previous section. Then click Upload.

3. Once the archive has been uploaded, click the gear menu next to it on the Files page, then right click Download and click Copy Link Address. Save the copied URL in your notes, as you will need it in the next step.

Once you have recorded the download URL of the archive, you’re ready to build a Domino environment for connecting to your Hortonworks cluster.
Creating a Domino environment for connecting to Hortonworks

1. Click **Environments** from the Domino main menu, then click **Create Environment**.

![Environments Overview](image1)

2. Give the environment an informative name, then choose a base environment that includes the version of Python that is installed on the nodes of your Hortonworks cluster. Most Linux distributions ship with Python 2.7 by default, so you will see the Domino Analytics Distribution for Python 2.7 used as the base image in the following examples. Click **Create** when finished.

![New Environment](image2)

3. After creating the environment, click **Edit Definition**. Copy the below example into your Dockerfile Instructions, then be sure to edit it wherever necessary with values specific to your deployment and cluster.

   In this Dockerfile, wherever you see a hyphenated instruction enclosed in carats like `<paste-your-domino-download-url-here>`, be sure to replace it with the corresponding value you recorded in previous steps.

   Additionally, follow the instructions in the comments carefully, as you may need to modify the commands based on versions and filenames from your system. You may also need to edit commands that follow to match downloaded filenames.

---

**5.5. Spark on Domino** 339
# Give user ubuntu ability to sudo as any user including root in the compute

RUN echo "ubuntu ALL=(ALL:ALL) NOPASSWD: ALL" >> /etc/sudoers

# Setup directories

RUN mkdir /tmp/domino-hadoop-downloads &&
   mkdir /usr/jdk64

# This downloaded gzip file should have the following
# - java installation tar file in 'java' sub-directory
# - krb5.conf in 'kerberos' sub-directory
# - hadoop, hive, spark2 and spark config directories from hadoop edgenode in a
#   'configs' sub-directory
# Make sure your URL is updated to reflect where you uploaded your configs.
# You should have this saved from your preparation steps
RUN wget --no-check-certificate <paste-your-domino-download-url-here> -O /tmp/
   domino-hadoop-downloads/hadoop-binaries-configs.tar.gz &&
   tar xzf /tmp/domino-hadoop-downloads/hadoop-binaries-configs.tar.gz -C /tmp/
   domino-hadoop-downloads/

# Install kerberos client and update the kerberos configuration file

RUN apt-get update &&
   apt-get -y install krb5-user telnet &&
   cp /tmp/domino-hadoop-downloads/hadoop-binaries-configs/kerberos/krb5.conf /
   etc/krb5.conf

# Install version of java that matches hadoop cluster and update environment

RUN tar xvf /tmp/domino-hadoop-downloads/hadoop-binaries-configs/java/jdk-8u112-
   linux-x64.tar -C /usr/jdk64 &&
   ln -s /usr/jdk64/jdk1.8.0_112 /usr/jdk64/default

ENV JAVA_HOME=/usr/jdk64/default

RUN echo "export JAVA_HOME=/usr/jdk64/default" >> /home/ubuntu/.domino-defaults &&
   echo "export PATH=$JAVA_HOME/bin:$PATH" >> /home/ubuntu/.domino-defaults

# Install HDP hadoop-client and spark binaries from Hortonworks Ubuntu repository.
# Update the repo URL based for the version that matches what's running on your
# cluster.
# This example shows version 2.6.5.
RUN wget http://public-repo-1.hortonworks.com/HDP/ubuntu14/2.x/updates/2.6.5.0/
   hdp.list -O /etc/apt/sources.list.d/hdp.list &&
   apt-key adv --keyserver keyserver.ubuntu.com --recv 07513CAD &&
   apt-get update &&
   apt-get -y install hadoop-client spark2-python spark-python

# Copy hadoop, hive, spark and spark2 configurations

RUN mv /etc/hadoop /tmp/domino-hadoop-downloads/hadoop-binaries-configs/configs/ &&
   mv /etc/hadoop-local.backup /etc/hadoop &&
   mv /etc/spark /tmp/domino-hadoop-downloads/hadoop-binaries-configs/configs/ &&
   mv /etc/spark-local.backup /etc/spark &&
   mv /etc/spark2 /tmp/domino-hadoop-downloads/hadoop-binaries-configs/configs/ &&
   mv /etc/spark2-local.backup /etc/spark2 &&
   cp -r /tmp/domino-hadoop-downloads/hadoop-binaries-configs/configs/hadoop /
   etc/hadoop &&
(continues on next page)
(continued from previous page)

```bash
cp -r /tmp/domino-hadoop-downloads/hadoop-binaries-configs/configs/hive /etc/
  
  cp -r /tmp/domino-hadoop-downloads/hadoop-binaries-configs/configs/spark2 /etc/spark2
  
  cp -r /tmp/domino-hadoop-downloads/hadoop-binaries-configs/configs/spark /etc/spark

# Update symlinks to point to correct configurations
# When you are creating these symlinks make sure that right versions are specified.
# Example: In the command 'ln -s /etc/spark2/2.6.5.0-292/0 /etc/spark2/conf'
# make sure that 2.6.5.0-292/0 is right version according to the edgenode and set the correct version similar to the hortonwork edgenode.

RUN rm /etc/spark2/conf &&
   rm /etc/spark/conf &&
   rm /etc/hadoop/conf &&
   rm /etc/hive/conf &&
   ln -s /etc/spark2/2.6.5.0-292/0 /etc/spark2/conf &&
   ln -s /etc/spark/2.6.5.0-292/0 /etc/spark/conf &&
   ln -s /etc/hadoop/2.6.5.0-292/0 /etc/hadoop/conf &&
   ln -s /etc/hive/2.6.5.0-292/0 /etc/hive/conf

# Update SPARK and HADOOP environment variables. Make sure py4j file name is correct as per your edgenode
ENV SPARK_HOME=/usr/hdp/2.6.5.0-292/spark2
RUN echo "export HADOOP_HOME=/usr/hdp/2.6.5.0-292/hadoop" >> /home/ubuntu/.domino-
    defaults &&
   echo "export HADOOP_CONF_DIR=/etc/hadoop/conf" >> /home/ubuntu/.domino-
    defaults &&
   echo "export YARN_CONF_DIR=/etc/hadoop/conf" >> /home/ubuntu/.domino-defaults &&
   echo "export SPARK_HOME=/usr/hdp/2.6.5.0-292/spark2" >> /home/ubuntu/.domino-
    defaults &&
   echo "export SPARK_CONF_DIR=/etc/spark2/conf" >> /home/ubuntu/.domino-
    defaults &&
   echo "export SPARK_MAJOR_VERSION=2" >> /home/ubuntu/.domino-defaults &&
   echo "export PYTHONPATH=$SPARK_HOME/python:$SPARK_HOME/python/lib/py4j-0.10.6-$
    src.zip" >> /home/ubuntu/.domino-defaults

# Backup existing spark-defaults.conf file. # Change spark-defaults.conf directory permission as a new spark-defaults.conf file gets created by Domino's spark integration
RUN mv /etc/spark2/conf/spark-defaults.conf /etc/spark2/ &&
   chmod 777 /etc/spark2/2.6.5.0-292/0
```

4. Scroll down to the Pre Run Script field and add the following lines.

```bash
cat /etc/spark2/spark-defaults.conf >> /etc/spark2/conf/spark-defaults.conf
sed -i.bak '/spark.ui.port\=0/d' /etc/spark2/conf/spark-defaults.conf
```

5. Scroll down and click Advanced to expand additional fields. Add the following line to the Post Setup Script field.

```bash
echo "export YARN_CONF_DIR=/etc/hadoop/conf" >> /home/ubuntu/.bashrc
```

6. Click Build when finished editing the Dockerfile instructions. If the build completes successfully, you are ready to try using the environment.

5.5. Spark on Domino
Configure a Domino project for use with a Hortonworks cluster

This procedure assumes that an environment with the necessary client software has been created according to the instructions above. Ask your Domino admin for access to such an environment.

1. Open the Domino project you want to use with your Hortonworks cluster, then click Settings from the project menu.
2. On the Integrations tab, click to select YARN integration from the Apache Spark panel, then click Save. You do not need to edit any of the fields in this section.
3. If your cluster uses Kerberos authentication, you can configure credentials at the user level or project level. Do so before attempting to use the environment. Note that if you followed the instructions above on creating your environment, your Kerberos configuration file has already been added to it.
4. On the Hardware & Environment tab, change the project default environment to the one with the cluster’s binaries and configurations files installed.

You are now ready to start Runs from this project that interact with your Hortonworks cluster.

Connecting to a MapR cluster from Domino

- Overview
- Gathering the required binaries and configuration files
- Uploading the binaries and configuration files to Domino
- Creating a Domino environment for connecting to MapR
- Configure a Domino project for use with a MapR cluster

Overview

Domino supports connecting to a MapR cluster through the addition of cluster-specific binaries and configuration files to your Domino environment.

At a high level, the process is as follows:
1. Connect to a MapR Edge node and gather the required binaries and configuration files, then download them to your local machine.

2. Upload the gathered files into a Domino project to allow access by the Domino environment builder.

3. Create a new Domino environment that uses the uploaded files to enable connections to your cluster.

4. Enable YARN integration for the Domino projects that you want to use with the MapR cluster.

Domino supports the following types of connections to a MapR cluster:

- FS shell for MapRFS
- spark-shell
- spark-submit
- pyspark
- YARN shell
- Hive with Beeline

Gathering the required binaries and configuration files

You will find most of the necessary files for setting up your Domino environment on your MapR Edge node. To get started, connect to the Edge node via SSH, then follow the steps below.

1. Create a directory named `hadoop-binaries-configs` at `/tmp`.

```bash
mkdir /tmp/hadoop-binaries-configs
```

2. Copy `hive-site.xml` from `/opt/mapr/spark/spark-<version>/conf` to `/tmp/hadoop-binaries-configs/`. Be sure to replace the `<version>` string in the command below with the number that matches the folder name on your edge node.

```bash
cp /opt/mapr/spark/spark-<version>/conf /tmp/hadoop-binaries-configs/
```

3. Copy the `ssl_truststore` from `/opt/mapr/conf` to `/tmp/hadoop-binaries-configs/`

```bash
cp /opt/mapr/conf/ssl_truststore /tmp/hadoop-binaries-configs/
```

4. Once you’ve copied the above files into `/tmp/hadoop-binaries-configs.tar.gz` zip up the directory for transfer to your local machine.

```bash
cd /tmp
tar -zcf hadoop-binaries-configs.tar.gz hadoop-binaries-configs
```

Then use SCP from your local machine to download the zipped archive. After transfer, extract the files to your local filesystem and keep them handy for a future step where they will be uploaded to Domino.
5. On the MapR edge node, run the following command to identify the version of Java running on the cluster.

```
java -version
```

You must then download a JDK .tar file from the Oracle downloads page that matches that version. The filename will have a pattern like the following.

```
jdk-8u211-linux-x64.tar.gz
```

Keep this JDK handy for use in a future step.

---

**Uploading the binaries and configuration files to Domino**

Use the following procedure to upload the files you retrieved in the previous step to a public Domino project. This will make the files available to the Domino environment builder.
1. Log in to Domino, then create a new public project.

2. Open the Files page for the new project, then click to browse for files and select the files you downloaded from the MapR edge node, and the JDK you downloaded from Oracle. Then click Upload.

3. From the Files page of your project, click New File. Name the file run-client.sh, and in its contents you must construct an invocation of the MapR configure.sh script that is valid for setting up a client to connect to your cluster. A full explanation of how to invoke this script is beyond the scope of this document. Read the full documentation on the script from MapR, and consider the following example.

   **run-client.sh**

   ```bash
   #!/bin/bash
   /opt/mapr/server/configure.sh -N <clustername> -c -secure -C <host1>:7222,<host2>:7222,<host3>:7222 -HS <historyServer>
   ```

4. Once your project contains the files from the MapR edge node, the correct JDK, and a run-client.sh script that
wraps the MapR configuration script, click the gear menu next to each of those files, then right click Download and click Copy Link Address. Save these URLs in your notes, as you will need them in the next step.

Once you have recorded the download URL of the binaries and configuration files, you’re ready to build a Domino environment for connecting to MapR.

Creating a Domino environment for connecting to MapR

1. Click Environments from the Domino main menu, then click Create Environment.

2. Give the environment an informative name, then choose a base environment that includes the version of Python that is installed on the nodes of your MapR cluster. Most Linux distributions ship with Python 2.7 by default, so you will see the Domino Analytics Distribution for Python 2.7 used as the base image in the following examples. Click Create when finished.
3. After creating the environment, click **Edit Definition**. Copy the below example into your Dockerfile Instructions, then be sure to edit it wherever necessary with values specific to your deployment and cluster.

In this Dockerfile, wherever you see a hyphenated instruction enclosed in carats like `<paste-your-domino-download-url-here>`, be sure to replace it with the corresponding value you recorded in previous steps. You may also need to edit commands that follow to match downloaded filenames.

```bash
# Base Image: quay.io/domino/base:Ubuntu16_DAD_Py2.7_R3.4-20180727
USER root

# Give the ubuntu user ability to sudo as any user including root in the compute environment
RUN echo "ubuntu ALL=(ALL:ALL) NOPASSWD: ALL" >> /etc/sudoers

# Set up directories
RUN mkdir /tmp/mapr-cluster-downloads && 
    mkdir /usr/jdk64

# Create a mapr user and group
RUN groupadd -g 5000 mapr
RUN useradd -u 5000 -g mapr mapr
RUN usermod -s /bin/bash mapr

# Use the following wget commands to download the four files you added to Domino in the previous section.
# You should have copied down the URLs to download a JDK .tar, the two files from the edge node, and the run-client.sh script you created.
# The example below will use a JDK file named jdk-8u112-linux-x64.tar.gz. If you're using a different version or have a different filename, replace it wherever it occurs.
RUN cd /tmp/mapr-cluster-downloads && 
    wget <paste-your-hive-site-dot-xml-download-url-here> -O /tmp/mapr-cluster-downloads/hive-site.xml.gz
    wget <paste-your-jdk-tar-download-url-here> -O /tmp/mapr-cluster-downloads/jdk-8u112-linux-x64.tar.gz
    wget <paste-your-ssl-truststore-download-url-here> -O /tmp/mapr-cluster-downloads/ssl_truststore.gz
```

(continues on next page)
gunzip run-client.sh.gz ¶
gunzip hive-site.xml.gz ¶
gunzip jdk-8u112-linux-x64.tar.gz ¶
gunzip ssl_truststore.gz ¶
cd ~

# Install Java from the JDK
RUN tar xvf /tmp/mapr-cluster-downloads/jdk-8u112-linux-x64.tar -C /usr/jdk64 ¶
  ln -s /usr/jdk64/jdk1.8.0_112 /usr/jdk64/default
ENV JAVA_HOME=/usr/jdk64/default
RUN echo "export JAVA_HOME=/usr/jdk64/default" >> /home/ubuntu/.domino-defaults ¶
  echo "export PATH=$JAVA_HOME/bin:$PATH" >> /home/ubuntu/.domino-defaults

# Install mapr-client and Spark binaries from the MapR ubuntu repository.
# These examples are for MapR 6.1.0.
# If you are using a different version of MapR, replace these URLs with the
RUN echo "deb https://package.mapr.com/releases/v6.1.0/ubuntu binary trusty" >> /etc/apt/sources.list
RUN echo "deb https://package.mapr.com/releases/MEP/MEP-6.0.0/ubuntu binary trusty" >> /etc/apt/sources.list
RUN wget -O - https://package.mapr.com/releases/pub/maprgpg.key | sudo apt-key add -
RUN apt-get update
RUN apt-get -y install mapr-client mapr-spark mapr-hive

# Copy the ssl_truststore file from the /opt/mapr/conf directory on the cluster
# to the /opt/mapr/conf directory on the client
RUN cp /tmp/mapr-cluster-downloads/ssl_truststore /opt/mapr/conf/

# Make your customized script from the previous section executable
RUN chmod +x /tmp/mapr-cluster-downloads/run-client.sh

# Update SPARK and HADOOP environment variables.
# Make sure the Spark and Hadoop version numbers match what is installed on your
# cluster.
# The examples below show Spark 2.3.1 and Hadoop 2.7.0.
# If you are using different versions, be sure to edit the file and directory
# names to match.
# Make sure the py4j file name is correct per your edgenode.
ENV SPARK_HOME=/opt/mapr/spark/spark-2.3.1
RUN echo "export SPARK_HOME=/opt/mapr/spark/spark-2.3.1" >> /home/ubuntu/.domino-defaults ¶
  echo "export HADOOP_HOME=/opt/mapr/hadoop/hadoop-2.7.0" >> /home/ubuntu/.domino-defaults ¶
  echo "export HADOOP_CONF_DIR=/opt/mapr/hadoop/hadoop-2.7.0/etc/hadoop" >> /home/ubuntu/.domino-defaults ¶
  echo "export YARN_CONF_DIR=/opt/mapr/hadoop/hadoop-2.7.0/etc/hadoop" >> /home/ubuntu/.domino-defaults ¶
  echo "export SPARK_CONF_DIR=/opt/mapr/spark/spark-2.3.1/conf" >> /home/ubuntu/.domino-defaults ¶
  echo "export PYTHONPATH=$SPARK_HOME/python:$SPARK_HOME/python/lib/py4j-0.10.7-src.zip" >> /home/ubuntu/.domino-defaults

# Change spark configuration directory permission as a new spark-defaults.conf
# file gets created by Domino's spark integration
4. Scroll down to the **Pre Run Script** field and add the following lines, being sure to match the Spark version in the directory name to the one being set up by the Dockerfile instructions.

```bash
# Configure mapr-client with your customized script.
sudo bash /tmp/mapr-cluster-downloads/run-client.sh

# Copy hive-site.xml to the spark configuration directory
# Be sure to match the Spark version in this folder name to match what you set up above.
cp /tmp/mapr-cluster-downloads/hive-site.xml /opt/mapr/spark/spark-2.3.1/conf
```

5. (Optional) If you want to store and access MapR user tickets as Domino environment variables, follow these additional steps.

1. Request a long-running MapR ticket from your cluster administrator, and copy its contents to your local machine. The ticket will be formatted as:
   
   `<cluster-name> <token>`

2. Add that token as a **Domino environment variable** to your Domino user account with the name **USERTICKET**.
   
   ![User environment variables](image)

   3. **Add the following lines to the bottom of the Pre Run Script field for the environment you edited previously.**

   ```bash
   ## Write maprticket in environment variable to a file during runtime
   echo $USERTICKET > /tmp/maprticket_12574
   ```

---

5.5. **Spark on Domino**
chown ubuntu:ubuntu /tmp/maprticket_12574
chmod 600 /tmp/maprticket_12574

Note that if you do this, every user that wants to use this environment must set up a USERTICKET environment variable as described in the previous step.

6. Click **Build** when finished editing the Dockerfile instructions. If the build completes successfully, you are ready to try using the environment.

Configure a Domino project for use with a MapR cluster

This procedure assumes that an environment with the necessary client software has been created according to the instructions above. Ask your Domino admin for access to such an environment.

1. Open the Domino project you want to use with your MapR cluster, then click **Settings** from the project menu.

2. On the Integrations tab, click to select **YARN** integration from the Apache Spark panel, then click **Save**. You should not need to edit any of the fields in this section.

3. On the **Hardware & Environment** tab, change the project default environment to the one you built earlier with the binaries and configuration files.

You are now ready to start Runs from this project that interact with your MapR cluster.

Connecting to an Amazon EMR cluster from Domino

- **Overview**
- **Requirements**
- **Gathering and serving the required binaries and configuration files**
- **Creating a Domino environment for connecting to EMR**
- **Configure a Domino project for use with an EMR cluster**
Overview

Domino supports connecting to an Amazon EMR cluster through the addition of cluster-specific binaries and configuration files to your Domino environment.

At a high level, the process is as follows:

1. Connect to the EMR Master Node and gather the required binaries and configuration files needed by Domino.
2. Create a new Domino environment that uses the uploaded files to enable connections to your cluster.
3. Enable YARN integration for the Domino projects that you want to use with the EMR cluster.

Domino supports the following types of connections to an EMR cluster:

- FS shell
- spark-shell
- spark-submit
- pyspark

Requirements

These instructions are written with the following requirements:

- Domino needs to be routable from the EMR cluster by private EC2 IP. This can be achieved by launching EMR directly into Domino’s VPC or via VPC Peering.
- Your security groups are configured to allow traffic between EMR and Domino. The Domino node security group, the EMR Master Node, and the EMR Worker Node security groups all need to allow TCP traffic between them.

Gathering and serving the required binaries and configuration files

You will find the necessary files for setting up your Domino environment on the EMR Master Node. To get started, connect to your Master Node via SSH.
Once connected to the Master Node, use \texttt{vi} or the editor of your choice to create a script called \texttt{domino-emr-config-maker.sh}. Copy in the following code and save the script.

\textbf{Note:} This script also captures Hive binaries. If you do not have Hive installed on your EMR cluster, you may need to remove the Hive-related commands from this script.

```bash
#!/bin/bash
rm -rf www
rm -rf /tmp/hadoop-binaries-configs
mkdir -p www
mkdir -p /tmp/hadoop-binaries-configs/configs

cp -rL /etc/hadoop /tmp/hadoop-binaries-configs/configs

cp -rL /etc/hive /tmp/hadoop-binaries-configs/configs

cp -rL /etc/spark /tmp/hadoop-binaries-configs/configs

cp -r /usr/lib/hadoop /tmp/hadoop-binaries-configs

cp -r /usr/lib/hadoop-lzo /tmp/hadoop-binaries-configs

cp -r /usr/lib/spark /tmp/hadoop-binaries-configs

cp -r /usr/share/aws /tmp/hadoop-binaries-configs

cp -r /usr/share/java /tmp/hadoop-binaries-configs

cd /tmp/hadoop-binaries-configs/configs/hadoop/conf/

sed -i '$ d' hdfs-site.xml

echo "<property>" >> hdfs-site.xml

echo "<name>dfs.client.use.datanode.hostname</name>" >> hdfs-site.xml

echo "<value>true</value>" >> hdfs-site.xml

echo "</property>" >> hdfs-site.xml

echo "</configuration>" >> hdfs-site.xml

cd /tmp

tar -zcf hadoop-binaries-configs.tar.gz hadoop-binaries-configs

cd ~

mv /tmp/hadoop-binaries-configs.tar.gz www/

cd www

/usr/bin/python3 -m http.server
```

This script bundles together all of the required binaries and configurations and serves it via a webserver on port 8000 of the Master Node. You will need to open port 8000 in your cluster’s security group if you have not already.

Before moving on, note the private IP address of your EMR Master Node. This will be available in your connection’s prompt.

Execute this script via the command `bash domino-emr-config-maker.sh` to begin the bundling and launch the webserver. You will want to leave your \texttt{ssh} connection to the Master Node open while finishing the rest of this setup.
Creating a Domino environment for connecting to EMR

1. Create a new Domino environment with the latest version of the Domino Analytics Distribution as its base image.

2. Edit this environment, and add the following code to the environment’s Dockerfile. Be sure to replace `<MASTER_NODE_PRIVATE_IP>` with the private IP address you noted earlier.

```bash
ENV EMR_MASTER_PRIVATE_IP <MASTER_NODE_PRIVATE_IP>

USER root
RUN echo "ubuntu ALL=(ALL:ALL) NOPASSWD: ALL" >> /etc/sudoers

RUN mkdir /tmp/domino-hadoop-downloads

# Download the binaries and configs gzip from EMR master.
# This downloaded gzip archive should contain a configs directory with
# hadoop, hive, and spark subdirectories directories.
# You may need to edit this depending on where you are running the web server on your EMR master.
   tar xzf /tmp/domino-hadoop-downloads/hadoop-binaries-configs.tar.gz -C /tmp/domino-hadoop-downloads/

RUN cp -r /tmp/domino-hadoop-downloads/hadoop-binaries-configs/configs/hadoop /etc/hadoop &&
   cp -r /tmp/domino-hadoop-downloads/hadoop-binaries-configs/configs/hive /etc/hive &&
   cp -r /tmp/domino-hadoop-downloads/hadoop-binaries-configs/configs/spark /etc/spark

RUN mv /tmp/domino-hadoop-downloads/hadoop-binaries-configs/aws /usr/share/aws
RUN mv /tmp/domino-hadoop-downloads/hadoop-binaries-configs/hadoop /usr/lib/hadoop
RUN mv /tmp/domino-hadoop-downloads/hadoop-binaries-configs/hadoop-lzo /usr/lib/hadoop-lzo
RUN mv /tmp/domino-hadoop-downloads/hadoop-binaries-configs/spark /usr/lib/spark
RUN cp -r /tmp/domino-hadoop-downloads/hadoop-binaries-configs/java/* /usr/share/java/

RUN \
   echo 'export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64' >> /home/ubuntu/.domino-defaults \n   echo 'export HADOOP_HOME=/usr/lib/hadoop' >> /home/ubuntu/.domino-defaults \n   echo 'export SPARK_HOME=/usr/lib/spark' >> /home/ubuntu/.domino-defaults \n   echo 'export PYTHONPATH=$(PYTHONPATH:-):$(SPARK_HOME:-)/python/' >> /home/ubuntu/.domino-defaults \n   echo 'export PYTHONPATH=$(PYTHONPATH:-):$(SPARK_HOME:-)/python/lib/py4j-0.10.7-src.zip' >> /home/ubuntu/.domino-defaults \n   echo 'export PATH=$(PATH:-):$(SPARK_HOME:-)/bin' >> /home/ubuntu/.domino-defaults \n   echo 'export PATH=$(PATH:-):$(HADOOP_HOME:-)/bin' >> /home/ubuntu/.domino-defaults \n   echo 'export HADOOP_CONF_DIR=/etc/hadoop/conf' >> /home/ubuntu/.domino-defaults \n   echo 'export YARN_CONF_DIR=/etc/hadoop/conf' >> /home/ubuntu/.domino-defaults

(continues on next page)
Configure a Domino project for use with an EMR cluster

This procedure assumes that an environment with the necessary client software has been created according to the instructions above. Ask your Domino admin for access to such an environment. Note that you may need to provide Domino with additional options when setting up your project. Your Domino or AWS administrators should be able to provide you with the correct values for these options.

1. Open the Domino project you want to use with your EMR cluster, then click Settings from the project menu.
2. On the Integrations tab, click to select YARN integration from the Apache Spark panel.
3. Use root as the Hadoop username.
4. If your work with the cluster generates many warnings about missing Java packages, you can suppress these by adding the following to Spark Configuration Options.

   Key: spark.hadoop.yarn.timeline-service.enabled
   Value: false

5. After inputting your YARN configuration, click Save.
6. On the Hardware & Environment tab, change the project default environment to the one you built earlier with the binaries and configuration files.

You are now ready to start Runs from this project that interact with your EMR cluster.

Running local Spark on a Domino executor
Using a local Spark cluster

Typically, users interested in Hadoop and Spark have data volumes and workloads that demand the power of cluster computing. However, some people use Spark for its expressive API, even if their data volumes are small or medium. Because Domino lets you run code on powerful VM infrastructure, with up to the 32 cores in AWS, you can use Domino to create a local Spark cluster and easily parallelize your tasks across all 32 cores.

Configuring Spark in Local mode

To configure Spark integration in Local mode, open your project and go to “Project settings.” Under “Integrations”, choose the “Local mode” option for Apache Spark. Click “Save” to save your changes.

Using PySpark in Jupyter Workspaces

Overview

You can configure a Domino Workspace to launch a Jupyter notebook with a connection to your Spark cluster. This allows you to operate the cluster interactively from Jupyter with PySpark.

The instructions for configuring a PySpark Workspace are below. To use them, you must have a Domino environment that meets the following prerequisites:

- The environment must use one of the Domino Standard Environments as its base image
- The necessary binaries and configurations for connecting to your Spark cluster must be installed in the environment. Refer to the provider-specific guides for setting up the environment.

Note: PySpark 2 does not support Python 3.8 or higher. Build PySpark 2 compute environments from images with Python before 3.8.

Adding a PySpark Workspace option to your environment

1. From the Domino main menu, click Environments.
2. Click the name of an environment that meets the prerequisites listed above. It must use a Domino standard base image and already have the necessary binaries and configuration files installed for connecting to your spark cluster.
3. On the environment overview page, click Edit Definition.
4. In the Pluggable Workspace Tools field, paste the following YAML configuration.

```yaml
pyspark:
  title: "PySpark"
  start: [ "/var/opt/workspaces/pyspark/start" ]
  iconUrl: "https://raw.githubusercontent.com/dominodatalab/workspace-configs/develop/workspace-logos/PySpark.png"
  httpProxy:
    port: 8888
    rewrite: false
    internalPath: "/{{#if pathToOpen}}tree/{{pathToOpen}}{{/if}}"
  supportedFileExtensions: [ ".ipynb" ]
```

5.5. Spark on Domino 355
When finished, the field should look like this.

### Pluggable Workspace Tools

<table>
<thead>
<tr>
<th>Properties for Workspaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
</tbody>
</table>

5. Click **Build** to apply the changes and build a new version of the environment. Upon a successful build, the environment is ready for use.

### Launching PySpark Workspaces

1. Open the project you want to use a PySpark Workspace in.

2. Open the project settings, then follow the provider-specific instructions from the Hadoop and Spark overview on setting up a project to work with an existing Spark connection environment. This will involve enabling YARN integration in the project settings.

3. On the **Hardware & Environment** tab of the project settings, choose the environment you added a PySpark configuration to in the previous section.

4. Once the above settings are applied, you can launch a PySpark Workspace from the Workspaces dashboard.

#### Kerberos authentication

### Overview

Domino supports Kerberos authentication, allowing users to authenticate as themselves when connecting to Kerberos-secured systems.
Users can enable Kerberos authentication at the project-level or user-level by uploading a Kerberos keytab and principal into Domino. Once set up, Runs started by Kerberos-enabled users or in Kerberos-enabled projects in Domino will automatically run kinit and retrieve the ticket to be able to authenticate.

Adding your Kerberos configuration file to Domino

There are two ways to add your krb5.conf file to Domino.

1. **Add it to your project in a folder named `kerberos`:**

   ![Image of Domino workspace with a file named krb5.conf]

   ```bash
   RUN echo "[libdefaults]" >> /etc/krb5.conf &&
   echo " default_realm = [YOUR-DEFAULT-REALM]" >> /etc/krb5.conf &&
   echo " default_tkt_enctypes = des3-hmac-sha1 des-cbc-crc" >> /etc/krb5.conf &&
   echo " default_tgs_enctypes = des3-hmac-sha1 des-cbc-crc" >> /etc/krb5.conf &&
   echo " dns_lookup_kdc = true" >> /etc/krb5.conf &&
   echo " dns_lookup_realm = false" >> /etc/krb5.conf &&
   echo " EXAMPLE.COM = { kdc = kerberos.example.com" >> /etc/krb5.conf &&
   echo " admin_server = kerberos.example.com" }" >> /etc/krb5.conf
   ```

2. **Add it to your environment at `/etc/krb5.conf`:**

   ```bash
   RUN echo "[libdefaults]" >> /etc/krb5.conf &&
   echo " default_realm = [YOUR-DEFAULT-REALM]" >> /etc/krb5.conf &&
   echo " default_tkt_enctypes = des3-hmac-sha1 des-cbc-crc" >> /etc/krb5.conf &&
   echo " default_tgs_enctypes = des3-hmac-sha1 des-cbc-crc" >> /etc/krb5.conf &&
   echo " dns_lookup_kdc = true" >> /etc/krb5.conf &&
   echo " dns_lookup_realm = false" >> /etc/krb5.conf &&
   echo " EXAMPLE.COM = { kdc = kerberos.example.com" >> /etc/krb5.conf &&
   echo " admin_server = kerberos.example.com" }" >> /etc/krb5.conf
   ```

Adding Kerberos credentials to your user

To add a keytab and principal that will be used for Runs started by your user, open your **Account Settings** and click **Kerberos Integration** from the settings menu. Click **Keytab file based authentication**, supply your keytab and principal, then click **Save**.
To add a keytab and principal that will be used for Runs started by a specific project, open the project Settings and click to open the Integrations tab.

In the Kerberos panel, click Keytab file based authentication, supply your keytab and principal, then click Save.
5.6 Customize the Domino software environment

Modify your environment to include the packages and dependencies you need or add new workspaces.

5.6.1 Video introduction to Domino Environments

To learn more about environments, read:

- Environment management
- Adding new workspaces with pluggable notebooks
5.6.2 Environment management

- Introduction
- Overview
- Managing environments
  - Overview tab
  - Revisions tab
  - Projects, Data Sets, and Models tabs
- Environment attributes
- Raw Dockerfiles
- Dockerfile best practices

Introduction

For a quick video introduction to Domino Environments, check out:
- Introduction to Environments

Overview

Environment management is the practice of creating new Domino environments and editing existing environments to meet your specific language and package needs. This work is typically done by an administrator or advanced Domino user.

Here are some examples of when to create or modify an environment:
- You need to install a package for Python, R, Octave, or some other software dependency.
• You use a library that takes a long time to install, and you’d prefer to cache that package into the environment so that it’s always immediately available.

• You are managing an organization, and want to create a default environment for your team across all projects.

Domino uses Docker for environments. An environment is a Domino abstraction on top of a Docker image, that provides additional flexibility and versioning. When Domino starts your run, it creates a Docker container based on the environment associated with your project. Each run takes place in an isolated Docker container.

Managing environments

The environment your project will use is set from the project’s Settings page.

The dropdown allows you to select the environment that runs inside this project will use. You’ll see global environments plus any environments you own or that have been shared with an organization you belong to.
Click Manage Environments to open the environments overview. You’ll see the environments you have access to, including your deployment’s global environments, environments in use by projects you are a collaborator on, and environments shared with organizations you are a member of.

You can create a new environment by clicking Create Environment at top right. You’ll be asked to name your new environment and define its visibility. Administrators will see a third option to have the new environment be available globally (to all users of the deployment).

After creating your environment, you will be taken to the environment detail page, where you can define the Dockerfile and supporting scripts and settings for the environment.

Environment actions

- **Edit Definition**

  Takes you to a page where you can edit all of your environment’s attributes.

- **Duplicate Environment**
Clones your environment.

- **Archive Environment**

Hides your environment. Projects already using this environment will be allowed to continue to use it until a new project environment is set.

**Overview tab**

The overview tab shows all metadata about your environment including the following attributes. Click **Edit Environment** in the top right to go into edit mode to make changes to your environment. After each save, your environment’s revision number will be incremented by one and your Domino deployment will rebuild the environment and push it to the local docker registry.

**Revisions tab**

The revisions tab shows a list of all revisions of your compute environment along with each revision’s build status, timestamp, and docker image URI. You can click the gear icon to reveal additional options including the ability to view build logs, cancel builds, or set a revision as Active.

**Projects, Data Sets, and Models tabs**

Once the environment has been assigned to a project, data set or model, you will be able to see a list of those entities on their tab. This is useful for seeing who you need to contact if you update or want to archive an environment, for example.
Environment attributes

Base Environment

Gives you a choice between basing your compute environment on your deployment default or on a custom Dockerfile URI (e.g. registry.hub.docker.com/library/python:3.8-slim). This defines the FROM line in the Dockerfile Domino constructs for you.

Dockerfile Instructions

Enter your Dockerfile layers here. Docker’s official site has a handy guide here. You can also read our primer on Dockerfiles below.

Pluggable Notebooks / Workspace Sessions

Define which interactive tools should be available in a project using this environment. See this for more details.

Scripts

Here you can input lines of bash code which will be executed at the specified step in your experiment’s lifecycle. These commands are run as root and are executed at runtime.

- **Pre-setup scripts**
  
  Run before the Python packages in your project’s requirements.txt are installed.

- **Post-setup scripts**
  
  Run after the requirements.txt installation process.

- **Pre-run scripts**
  
  Run right after post-setup scripts

- **Post-run scripts**
  
  Run at the beginning of the Stopping run state. Due to the way Domino handles shutting down runs and workspaces, these scripts are subject to a runtime limit, which defaults to 1 hour. If the script hasn’t completed by the end of the limit, it will timeout and the process will terminate. The timeout duration is configurable. Contact your local administrator or mail to support@dominodatalab.com for assistance in modifying timeouts.

Docker Arguments

Here, admins can specify arguments that will be passed to the underlying docker run command. Arguments must be separated by newlines. In almost all cases, you shouldn’t need to modify this.

Username

Admins can specify a non-default username for your environment here.

Environment variables

You can set environment variables at the environment level.
Raw Dockerfiles

You may wish to install packages directly to your environment. This can come in handy if your package installation takes a long time. Installations in a Domino environment are cached, so you won’t have to wait for it every time. The Domino platform uses Docker containers to manage isolated environments. If you already have a Docker image you’d like to use, you can specify it in the preceding Base Environment field. If you don’t set this, we will use the Domino default environment as your base image. Consult the official Docker documentation to learn more about Dockerfiles:

- Reference
- Best practices

Note that Domino takes care of the FROM line for you, pointing to the base image specified when setting up the environment. Do not start your Dockerfile instructions in Domino with a FROM line.

The most common Dockerfile instructions you’ll use are RUN, ENV, and ARG:

RUN commands execute lines of bash, for example:

```bash
RUN wget http://d3kbcqa49mib13.cloudfront.net/spark-1.5.1-bin-hadoop2.6.tgz
RUN tar xvzf spark-1.5.1-bin-hadoop2.6.tgz
RUN mv spark-1.5.1-bin-hadoop2.6 /opt
RUN rm spark-1.5.1-bin-hadoop2.6.tgz
```

ARG commands set build-time variables, and ENV commands set container bash environment variables. They will be accessible from runs that use this environment. For example:

```bash
ENV SPARK_HOME /opt/spark-1.5.1-bin-hadoop2.6
```

If you set environment variables as part of the Environment variables section of your environment definition, you need to specify the variable name only with an ARG statement:

```bash
ARG SPARK_HOME
```

This will be available for the build step. If you want the variable to be available in the final compute environment you also need to add an ENV statement referencing the argument name:

```bash
ENV SPARK_HOME=$SPARK_HOME
```

Examples: Package Installation

You can click the R Package or Python Package buttons when editing your environment, and these will insert a line with the correct syntax to install packages (just fill in the names of the packages you want). Or you can add the commands yourself, following these examples:

- R Package Installation: Example with the devtools package.

```bash
RUN R --no-save -e "install.packages('devtools')"
```

- Python Package Installation with Pip: Example with the numpy package.

```bash
RUN pip install numpy
```
Dockerfile best practices

- Docker optimizes its build process by keeping track of commands it has run and aggressively caching the results. This means that if it sees the same set of commands as a previous build, it will assume it can use the cached version. A single new command will invalidate the caching of all subsequent commands.

- There is a limit to the number of layers (that is, commands) a docker image can have. Currently, this limit is 127. Keep in mind that the image upon which you are building may have already used many layers. One way to work around this limit is to combine several commands into one via `&&`, like this:

```bash
RUN
  wget http://d3kbcqa49mib13.cloudfront.net/spark-1.5.1-bin-hadoop2.6.tgz &&
  tar xvzf spark-1.5.1-bin-hadoop2.6.tgz &&
  mv spark-1.5.1-bin-hadoop2.6 /opt &&
  rm spark-1.5.1-bin-hadoop2.6.tgz
```

- If you are installing multiple python packages via pip, it’s almost always best to use a single pip install command. This ensures that dependencies and package versions are properly resolved. If you install via separate commands, you may end up inadvertently overriding a package with the wrong version, due a dependency specified by a later installation. For example:

```bash
RUN pip install luigi nolearn lasagne
```

5.6.3 Domino Analytics Distribution

- Overview
- Domino Analytics Distribution (DAD)
- Domino Minimal Distribution (DMD)
- Example of Implementing a New Environment
- FAQ
Overview

Each run and workspace in Domino operates in its own Docker container. These Docker containers are defined by Domino compute environments. Environments can be shared and customized, and they are automatically versioned by Domino.

New installations of Domino come with a standard set of environments known as the Domino Analytics Distribution. Periodically, Domino publishes a new set of standard environments with updated libraries and packages. These environments include many common data science packages and libraries pre-configured for use in Domino.

We also make available a set of minimal environments (Domino Minimal Distribution) which includes only the necessary packages required to work with in Domino. These would be an appropriate option for a user who wants to build a Domino-compatible environment from scratch.

Domino Analytics Distribution (DAD)

The Domino Analytics Distributions are designed to handle most of what a typical data science workflow needs out of the box. They include the most common Python and R packages. Various versions are available if you need CUDA support.

You can review the available dockerfile and descriptions here: Domino Base Images.

The built images are hosted on quay.io/domino/base unless otherwise stated in the READMEs for the corresponding image.

Domino Minimal Distribution (DMD)

While the DAD includes most of what a data scientist needs to do their work, the DMD includes only the bare necessities required to work in Domino.

Specifically, the objective for the DMD is to provide an image which will allow one to: - Open Jupyter, Jupyterlab, VSCode and R Studio workspaces - Batch run Python and R jobs - Host a Shiny web app - Publish a Python and R Model API - Use Domino’s Git integration - Install Python and R packages

You can shrink the DMD to be smaller by removing any of the workspaces you won’t be using or removing either Python or R.

You can review the available dockerfile and descriptions here: Domino Base Images.

The built images are hosted on quay.io/domino/base unless otherwise stated in the READMEs for the corresponding image.

5.6. Customize the Domino software environment
Example of Implementing a New Environment

1. Select an environment from the available by choosing the python and R version. Typically, you’ll always want to chose the latest environment.

   • Note: Environments tagged “_legacy” are designed to work with Domino versions <4. The only difference between a regular and legacy environment is the way they handle CUDA given the switch to using nvidia-docker2 in Domino version 4.0.

2. Find the Appropriate Name, Description, Image URI and “Pluggable Properties” for your environment.

   • For example, for this environment:

     Title: DAD Py3.7 R3.6
     URI: dominodatalab/base:DAD_py3.7_r3.6_2019q4
     Description:

     Ubuntu 18.04
     Mini-conda 4.7.12.1
     Python 3.7.4
     R 3.6.2
     Jupyter, Jupyterlab, VSCode, Rstudio
     Cuda 10.0
     https://github.com/dominodatalab/Domino_Base_Images/tree/master/
     →Domino_Analytics_Distribution/2019_q4_py3.7_r3.6

**Pluggable Workspace Tools**

```json
jupyter:
  title: "Jupyter (Python, R, Julia)"
  iconUrl: "/assets/images/workspace-logos/Jupyter.svg"
  start: [ "/var/opt/workspaces/jupyter/start" ]
  httpProxy:
    port: 8888
    rewrite: false
    internalPath: "/{{ownerUsername}}/{{projectName}}/
    →{{sessionPathComponent}}/{{runId}}/{{#if pathToOpen}}tree/
    →{{pathToOpen}}{{/if}}"
  supportedFileExtensions: [ ".ipynb" ]

jupyterlab:
  title: "JupyterLab"
  iconUrl: "/assets/images/workspace-logos/jupyterlab.svg"
  start: [ "/var/opt/workspaces/Jupyterlab/start.sh" ]
  httpProxy:
    internalPath: "/{{ownerUsername}}/{{projectName}}/
    →{{sessionPathComponent}}/{{runId}}/{{#if pathToOpen}}tree/
    →{{pathToOpen}}{{/if}}"
    port: 8888
    rewrite: false
    requireSubdomain: false

vscode:
```

(continues on next page)
3. Create a new Domino Compute environment

- See Compute Environment Management for an overview of how to create and manage environments.

4. Update your Domino AMI (not required for non-cloud)

- Once you’ve created a compute environment with a new base image, you’ll want to work with your admin to update your Domino’s AMI (or if not on AWS, the GCP or Azure equivalent) by caching the new image. As Domino spins up and down new executors, if your new image is not in the AMI, it will need to pull that image onto the executor the first time it starts up. This can cause a ~10 minute delay for starting workspaces on new executors. See here for the procedure to snap and update your AMI

FAQ

1. How can I tell which image I’m currently using?

   The URI for the image will be listed on your compute environments overview page. If you environment is built on top of the another environment, you may need to click through to the parent environment before seeing the underlying docker image.

2. I have a third party docker image, can I use that in Domino?

   Maybe, but not likely without some customization. The DAD and DMD are tested and configured to meet the Domino platform requirements and conventions. For example, by convention Domino uses /mnt as the default working directory. By and large, these requirements are best understood by reviewing the DMD dockerfiles. If you have a dockerfile you’d like to use within Domino, it’s recommended that you add those instructions to either the DMD or DAD rather than starting from scratch.

3. How can I learn about new versions of the DAD and make feature requests?

   Check out the Domino community forum for news and updates.

5.6.4 Installing packages and dependencies

Domino is pre-installed with many common libraries, so we recommend that you first try running your code before customizing your configuration. If you need to, there are specific ways to add your own dependencies for...

- Python

- R

If you have more specialized dependencies, please let us know.
**Persistent installations**

By default, Domino runs each of your scripts (or interactive sessions) from a fresh environment. I.e., any packages you install will install each time. We have more advanced functionality for managing your own compute environments with software changes that stay permanently installed. Learn more about custom environments.

---

**Python**

**Checking what’s already installed**

Many common modules are installed by default. To get a list of pre-installed modules, you can include `help('modules')` at the start of your script or in a new IPython Notebook session. Or using the Domino CLI tool, you can run `domino run --direct "pip freeze"`

**Adding your own packages**

To specify your own additional dependencies, you can use pip with Domino. To specify module dependencies for your project, add a pip requirements file named `requirements.txt` to the root of your project folder.

The requirements file specifies which libraries and any version requirements for them. An example:

```plaintext
pandas
lxml==3.2.3
numpy>=1.7.1
```

For a full reference on the syntax of the requirements file, read this.

If you’re using pip on your local machine, the easiest way to generate the `requirements.txt` file is to run the following command in the root of your project folder:

```bash
~/domino/myProject $ pip freeze > requirements.txt
```

For performance reasons, you should prune that file so that it includes only the libraries you need for your actual analysis.

Alternatively, if you’re working in a Jupyter Notebook, you can also use pip to install dependencies interactively. In a notebook cell, you can run

```bash
! pip install --user <package>
```

(The ‘!’ tells the notebook to execute the cell as a shell command)

**Installing Packages Hosted from a Git Repository**

**Warning:** this is an advanced topic!

Pip can install Python packages from source by cloning a public Git repository over https. For full reference, see here. To specify this, you will need to add something like the following line to your `requirements.txt` file:

```plaintext
-e git+https://git.yourproject.org/you/Project.git#egg=YourProject
```
The most common host of Git projects is GitHub. If the package you wish to install is publicly accessible, then the instructions above will work. However, if you need to install any private repositories, Domino can securely integrate with GitHub to access those private repositories. Read our instructions on securely storing your GitHub credentials.

Do not embed your GitHub credentials directly in the requirements.txt file. Instead, integrate Domino with GitHub securely by following our instructions here.

---

**R**

If you’re using Domino for R scripts, you may also want to check out our R Package for controlling Domino in your R IDE.

Most common packages are installed by default (you can include `installed.packages()` at the start of your script to print out a list of installed packages). If you need additional packages, you can use R’s built-in package manager to install and load them: simply add `install.packages()` calls for any packages your code needs to the top of your R scripts, e.g.,

```r
install.packages("rpart", dependencies=TRUE, repos='http://cran.us.r-project.org')
# install other packages...

library('rpart')
# rest of your script...
```

These package installation calls must be at the top of every R script that Domino runs. If your packages take a long time to install, we recommend that you create a custom compute environment for efficiency.

### 5.6.5 Adding new workspaces with pluggable notebooks

**Overview**

In the same way that Domino allows you to create compute environments to meet your specific language and package needs, this functionality allows you to define web-based tools inside of your compute environment. This work is typically done by an administrator or advanced Domino user. We suggest you reach out to your admin for help with defining the tool you wish to use.

**Benefits**

- Upgrade to a newer version of currently supported Domino tools such as Jupyter or RStudio
- Add new web-based tools like JupyterLab
- Manage the standard default tool for your team or organization across all projects
Setting up tools in Environments

Configuring a tool in an environment involves two parts: giving the environment’s docker image installation instructions, and defining how Domino will serve up that tool.

Dockerfile instructions

In your environment, enter the instructions to install and configure a tool in the Dockerfile instructions:

Python version >2.7.9

```bash
###Remove any old workspaces
RUN \
    apt-get remove rstudio-server -y && \
    rm -rf /usr/local/lib/rstudio-server/rstudio-server && \
    rm -rf /var/opt/workspaces

###Setup workspaces directory and retrieve workspace configs
RUN mkdir /var/opt/workspaces
    rm -rf /var/opt/workspaces/workspace-logos && rm -rf /tmp/workspace-configs-2018q2-v1.9

#add update .Rprofile with Domino customizations
RUN \
    mv /var/opt/workspaces/rstudio/.Rprofile /home/ubuntu/.Rprofile && \
    chown ubuntu:ubuntu /home/ubuntu/.Rprofile

# # # #Install Rstudio from workspaces
RUN chmod +x /var/opt/workspaces/rstudio/install
RUN /var/opt/workspaces/rstudio/install

# # # # # #Install Jupyterlab from workspaces
RUN chmod +x /var/opt/workspaces/Jupyterlab/install
RUN /var/opt/workspaces/Jupyterlab/install

# # #Install Jupyter from workspaces
RUN chmod +x /var/opt/workspaces/jupyter/install
RUN /var/opt/workspaces/jupyter/install

# Clean up temporary files
RUN \
    rm -Rf /var/lib/apt/lists/* && \
    rm -Rf /tmp/*
```
###Remove any old workspaces

```
RUN \n  apt-get remove rstudio-server -y && \n  rm -rf /usr/local/lib/rstudio-server/rstudio-server && \n  rm -rf /var/opt/workspaces
```  

###Setup workspaces directory and retrieve workspace configs

```
RUN mkdir /var/opt/workspaces

```

#add update .Rprofile with Domino customizations

```
RUN \n  mv /var/opt/workspaces/rstudio/.Rprofile /home/ubuntu/.Rprofile && \n  chown ubuntu:ubuntu /home/ubuntu/.Rprofile
```

# # # #Install Rstudio from workspaces

```
RUN chmod +x /var/opt/workspaces/rstudio/install

RUN /var/opt/workspaces/rstudio/install
```

# # # # # #Install Jupyterlab from workspaces (pinned to avoid working directory bug in Jupyterlab)

```
RUN pip install jupyterlab==0.31.12
```

# # # Install Jupyter from workspaces

```
RUN chmod +x /var/opt/workspaces/jupyter/install

RUN /var/opt/workspaces/jupyter/install
```

# Clean up temporary files

```
RUN \n  rm -Rf /var/lib/apt/lists/* && \n  rm -Rf /tmp/*
```

###Properties

Notebook properties are stored as YAML data mapping notebook names to their definitions. Enter this in the Properties for Notebooks field in the Environment definition.

**Example:**

```yaml
jupyter:
  title: "Jupyter (Python, R, Julia)"
  iconUrl: "/assets/images/workspace-logos/Jupyter.svg"
  start: [ "/var/opt/workspaces/jupyter/start" ]
  httpProxy:
    port: 8888
  rewrite: false
  internalPath: "/{{ownerUsername}}/{{projectName}}/{{sessionPathComponent}}/{{runId}}/{{if pathToOpen}}tree/{{pathToOpen}}{{/if}}"
  requireSubdomain: false
  supportedFileExtensions: [ ".ipynb" ]
jupyterlab:
```

(continues on next page)
5.6.6 Advanced options for Domino software environment

Installing custom packages in Domino with Git integration

Domino allows users to install internally developed custom R and Python packages stored in GitHub or GitHub Enterprise. For best performance, this should often be done by creating a custom environment. However, if the user does not want to create a custom environment or is running a quick test or prototype, it is possible to install these packages inside of a Domino project.

Configuring Git Integration

Users should begin by configuring git integration in their desired Domino projects.

Installing a Custom Python Package

If it doesn’t already exist, create a file named requirements.txt in the root directory of your project. Add the following line to requirements.txt:

```
-e /repos/<repo_name>
```

Once this requirements.txt is in place, Domino will automatically install the Python package every time a workspace or batch run starts within the project.

Installing a Custom R Package

If it doesn’t already exist, create a file named install.R; in the root directory of your project. Add the following line to install.R:
For example, if a user wanted to install ggplot2 from a git repository, his or her install.R should look like:

```r
my_domino_project / install.R
```

Once this install.R is in place, users should put a line:

```r
source('install.R')
```

at the beginning of their work within the project to install the project for their sessions.

**Adding custom DNS servers to your Domino Environment**

**Overview**

Custom DNS name servers are used to handle internal or private domain names and URLs. This guide shows how to add a new DNS server to your Domino environment to allow access to additional domains and hostnames.

**Instructions**

On a Linux host like your Domino executor, a configuration file at `/etc/resolv.conf` controls which DNS servers will be used to resolve hostnames. You can use a pre-setup script in your Domino environment to modify this file and add custom DNS servers by following these instructions.

1. Open the environment you want to modify, then click **Edit Definition**.
2. Scroll down to the bottom of the definition, then click to expand the **Advanced** section.

3. In the **Pre Setup Script** field, enter the following lines of Bash. Fill in the address of the custom server you want to use where indicated.

```bash
echo 'nameserver your-custom-server-address' > /tmp/resolv.conf.tmp
cat /etc/resolv.conf >> /tmp/resolv.conf.tmp
cat /tmp/resolv.conf.tmp > /etc/resolv.conf
```

When finished, your interface should look like this:
4. Click **Build** at the bottom of the definition page to create a new version of the environment.

5. Following a successful build, you can use this environment to reach hosts through the added DNS name server

**Configure a Compute Environment to use private CRAN/Conda/PyPi mirrors**

Many of our customers maintain internal mirrors of public package repos. This is typically done in concert with removing access to external, public repos via the use of a whitelist. This may be for security or for standardization.

By adding the following to your Dockerfile instructions of your global compute environments, you can set Conda, pip or Rstudio to reference your internal mirrors by default:

```
RUN \
    # Add $CUSTOMER mirrors to condarc config files. Also disabled auto_update since mirrors are not always up to date.\n    printf "\nchannels:\n - $CUSTOMER_CHANNEL_1\n - $CUSTOMER_CHANNEL_1
\n" >> /\n    /opt/conda/.condarc && printf "\nallow_other_channels: False\n" >> /opt/conda/.condarc && printf "\nnssl_verify: False \n" >> /opt/conda/.condarc && printf "\nshow_channel_urls: True\n" >> /opt/conda/.condarc && printf "\nauto_update_conda: False\n" >> /opt/conda/.condarc && \n\n    # Pointing R to $CUSTOMER CRAN mirror \n    printf "\nlocal({ \n r <- getOption('repos') \n r['CRAN'] <- '$CRAN_MIRROR_URL' \n options(repos = r) \n})\n" >> /home/ubuntu/.Rprofile && \n\n    # Pointing pip to $CUSTOMER PyPi mirror \n    printf "\n[global]\nindex-url=$CUSTOMER_PYPI_MIRROR\ntrusted-host=$IP_OR_FQDN\n\n" >> /etc/pip.conf && \n\n    chown -R ubuntu:ubuntu /home/ubuntu /opt/conda/
```

**Scala notebooks**

In our cloud-hosted environment, we have the scala-jupyter kernel installed for Jupyter, so you can create Scala notebooks.

5.6. Customize the Domino software environment
If you have Domino deployed on your own hardware, you can create a custom environment and install the Scala kernel to give yourself the same functionality. To install the kernel, follow the instructions on the project’s Github page linked above. At the time of writing this help article, installation was just running:

```
curl -L -o jupyter-scala https://git.io/vzhRi && chmod +x jupyter-scala && ./jupyter-scala && rm -f jupyter-scala
```

To start a Scala notebook, choose “Scala 2.11” from the “new” menu on the Jupyter main page.
The scala-jupyter kernel provides a number of nice helper functions. One of the most important is

```scala
classpath.add("organization" % "name" % "version")
```

which lets you add dependencies into the Scala environment. E.g., to load the Spark jars you would use:

```scala
classpath.add("org.apache.spark" % "spark-core_2.11" % "1.6.0")
```

---

**Using TensorBoard in Jupyter workspaces**

**Overview**

TensorBoard is a tool for visualizing TensorFlow data. TensorBoard operates by reading events files, which contain summary data that generated by TensorFlow. You can visualize your TensorFlow graph, plot quantitative metrics about graph, and show additional data that passes through the graph.

You can use TensorBoard inside your Domino Jupyter workspace by installing and enabling the Jupyter-TensorBoard server extension in your compute environment.
Environment Setup

You will need to create or modify an environment to enable this extension in your Domino workspaces. Read about Compute Environment Management if you aren’t already familiar.

1. In Domino, click Environments in the top navigation bar.
2. Click Create Environment.
3. Give the environment an informative name, and then choose a base image that has Python 3.6+ installed. You can use an environment as a base image if it uses this Domino standard:

   quay.io/domino/base:standard-py3.6-v1.3+

4. After selecting the appropriate base image, choosing a visibility setting, and optionally providing a description, click Create Environment to finalize.
5. The environment will be created and you will be automatically redirected to its Overview page. Scroll down to the Docker Settings and click Edit Dockerfile.
6. Add the following two lines to the Dockerfile Instructions:

   ```
   RUN pip install ipython jupyter jupyter-tensorboard --upgrade
   RUN jupyter tensorboard enable --system
   ```

7. Click Build. You will be redirected to the Revisions page for the environment. If the new revision builds successfully, you are ready to use this environment.

Using Jupyter-TensorBoard

1. Open the project you want to use with Jupyter-Tensorboard, then click Settings in the left navigation bar.
2. In the Compute environment panel, use the dropdown menu to choose the environment you created in the previous section. A notification will appear verifying that the new environment is now set.
3. Click Workspaces in the left navigation bar, then select Jupyter and launch a new workspace.
4. From the Files tab in the workspace, click New -> Tensorboard.

5. You can now access the new TensorBoard from the Running tab.
6. Refer to the TensorBoard README to learn how to start consuming TensorFlow events.

**Troubleshooting**

If you encounter issues loading Tensorboard initially, ensure that Tensorflow is operating properly first. Loading the UI has a dependency on Tensorflow. Furthermore, by default, Domino’s standard compute environments have tensorflow-gpu installed (e.g. pip install tensorflow-gpu). Thus, Tensorboard and Tensorflow will not work on a CPU hardware tier. If you’d like to use Tensorboard on a CPU make sure that CPU optimized Tensorflow is installed (e.g. pip install tensorflow).

**Using MATLAB® as a Workspace**

- **Overview**
- **Prerequisites**
  - Network Licensing structure for MATLAB
  - (Optional) Hardware Tiers with the correct GPU drivers
- **Environment Setup**
  - Choose a base image
  - Edit the Dockerfile
  - Add Pluggable Workspace Tools definition
  - (Optional) Persist MATLAB Preferences and Add-ons between sessions
  - (Optional) Generate a startup script for proxy settings
  - Build and Test the Environment
- **FAQ/Troubleshooting**
  - GPU functionality does not work as expected
  - MATLAB workspace fails to launch
    - License configuration problems
    - Advanced troubleshooting

5.6. Customize the Domino software environment 381
Overview

Interactive MATLAB® sessions are supported in Domino by means of configuring a Compute Environment and Interactive Workspace. If your administrators have already configured such an environment, you may be able to use it by simply switching your project to that environment.

This page provides the steps necessary to create a suitable Compute Environment if it does not already exist. Some steps may require assistance from a Domino or MATLAB admin.

Prerequisites

Network Licensing structure for MATLAB

To get MATLAB working using this guide, you will need a network license. A network concurrent license is recommended, but a network named license may also work IF the usernames specified in the license match the usernames within Domino. If you don’t currently have a network concurrent license, reach out to your MATLAB license administrator to determine if one is available or needs to be purchased. This information will be needed when configuring the Compute Environment (see below). You may have a single license server or a set of 3 license servers. We will show examples for two styles of providing the license server information:

- The first method requires only the port and host for your license server(s), e.g. 27000@lic1.customer.com (and optionally 27000@lic2.customer.com, 27000@lic3.customer.com). This is recommended for most cases.
- The second method applies if you have been given a network.lic file, e.g.
Use server

This is recommended only if you have multiple sets of license servers (and thus multiple license files), i.e. in the case of separate license servers for MATLAB Parallel Server.

See Mathworks help for more details on the various options or talk to your MATLAB license administrator.

(Optional) Hardware Tiers with the correct GPU drivers

To make full use of GPU capabilities, the hardware tier you are using within Domino needs to have GPU drivers installed that can support the version of MATLAB you want to use. Compatibility is determined by this table describing the minimum CUDA Toolkit version requirement for each version of MATLAB, and by this table describing the minimum NVIDIA driver version requirement for each version of the CUDA Toolkit.

The MATLAB base images provided by Domino (see below) will already have a suitable version of the CUDA Toolkit installed. The NVIDIA drivers must be configured at the Hardware Tier level. If you already have a GPU-enabled hardware tier configured, it is likely that it already has suitable NVIDIA drivers. See the Troubleshooting section below for instructions on how to check the driver version during a MATLAB session to verify it is configured correctly.

If you do not have the correct NVIDIA drivers configured on your GPU Hardware Tier, talk to your Domino administrator and refer them to 4.4 for configuring NVIDIA drivers in EKS.

Environment Setup

If a MATLAB-specific environment does not already exist in your Domino deployment, you will need to create one. Read about Compute Environment Management if you aren’t already familiar.

Choose a base image

1. In Domino, click Environments in the top navigation bar.
2. Click Create Environment.
3. Give the environment an informative name, and then choose a base image that has MATLAB installed. The following are the most current options available from Domino.

```
quay.io/domino/matlab:R2019b-complete-20201014
quay.io/domino/matlab:R2020a-complete-20201014
```

More details about each image are documented in our github repo: https://github.com/dominodatalab/Domino_Base_Images/tree/master/MATLAB_Base_Images. This includes lists of Toolboxes, release notes and known issues, and information for all past images.

4. After selecting the appropriate base image, choosing a visibility setting, and optionally providing a description, click Create Environment to finalize.
5. The environment will be created and you will be automatically redirected to its Overview page. Scroll down to the Docker Settings and click Edit Dockerfile to continue setting up the environment.
**Edit the Dockerfile**

Add the following lines to your *Dockerfile Instructions*, editing the license server information to match your actual license servers. Also be sure to edit the `MATLAB_VERSION` variable to match the MATLAB version of the base image you chose in the previous step.

If you have port and host information for your license servers (recommended method):

```bash
ENV MATLAB_VERSION='R2019b'
ENV MLM_LICENSE_FILE=27000@lic1.customer.com,27000@lic2.customer.com,27000@lic3.customer.com
```

Or, if you have a `network.lic` file (more useful if you have multiple license files) you can adapt this example:

```bash
ENV MATLAB_VERSION='R2019b'
RUN mkdir -p /usr/local/MATLAB/$MATLAB_VERSION/licenses &&
    printf "SERVER lic1.customer.com 000000000000 27000
SERVER lic2.customer.com 000000000010 27000
SERVER lic3.customer.com 000000000020 27000
USE_SERVER" > /usr/local/MATLAB/$MATLAB_VERSION/licenses/network.lic
```

**Add Pluggable Workspace Tools definition**

Add the appropriate “Pluggable Properties” to the *Properties for Workspaces*.

These are documented alongside the base image descriptions here: https://github.com/dominodatalab/Domino_Base_Images/tree/master/MATLAB_Base_Images.

Note that the MATLAB base images are built on top of the *Domino standard environments*, so you can add the workspace configs for e.g. Jupyter or RStudio for those standard environments also.

For example, if you are using the `quay.io/domino/matlab:r2019b-20200521` base image, add the following lines for the MATLAB workspace:

```json
matlab:
    title: "MATLAB"
    start: [ "/var/opt/workspaces/matlab/start.sh" ]
    httpProxy:
        port: 8888
    requireSubdomain: false
```

Because it is built on top of `quay.io/domino/base:Ubuntu18_DAD_Py3.6_R3.6_20190916`, you can optionally add any of the following sections for workspaces from that environment:

```json
jupyter:
    title: "Jupyter (Python, R, Julia)"
    iconUrl: "/assets/images/workspace-logos/Jupyter.svg"
    start: [ "/var/opt/workspaces/jupyter/start" ]
    httpProxy:
        port: 8888
    rewrite: false
```

(continues on next page)
(Optional) Persist MATLAB Preferences and Add-ons between sessions

In order to persist your MATLAB Preferences and Add-ons between Domino sessions, you can define a Post Run Script to automatically copy them to your Project Files when you Stop and Sync a session, and define a Pre Run Script to copy them from your Project Files into the locations MATLAB will recognize at the start of subsequent sessions.

Put the following lines into your Pre Run Script.

```bash
if [ -d /mnt/matlab-addons ]; then
    mkdir -p /home/ubuntu/Documents/MATLAB/SupportPackages/$MATLAB_VERSION/
fi
if [ -d /mnt/matlab-preferences/$DOMINO_STARTING_USERNAME ]; then
    mkdir -p /home/ubuntu/.matlab/$MATLAB_VERSION/
    rsync -r /mnt/matlab-preferences/$DOMINO_STARTING_USERNAME/ /home/ubuntu/.matlab/.
fi
```

Put the following lines into your Post Run Script.

```bash
if [ -d /home/ubuntu/Documents/MATLAB/SupportPackages/$MATLAB_VERSION ]; then
    mkdir -p /mnt/matlab-addons
fi
```

5.6. Customize the Domino software environment 385
Note that these scripts will save add-ons per project, so that any collaborators launching a workspace in the same project will have the same add-ons. For preferences, however, they will be saved per user in the project so that each user can have a different set of preferences.

(Optional) Generate a startup script for proxy settings

If your company uses a proxy to connect to external networks, you may already have lines like the following in many of your Compute Environment Dockerfiles:

```
ENV http_proxy='http://someusername:password123@proxy.company.com:80'
ENV https_proxy='http://someusername:password123@proxy.company.com:80'
ENV HTTP_PROXY='http://someusername:password123@proxy.company.com:80'
ENV HTTPS_PROXY='http://someusername:password123@proxy.company.com:80'
```

Without setting the proxy information in MATLAB as well, you may not be able to connect to external networks from MATLAB (e.g. for downloading community Add-ons). Proxy information can be set manually in MATLAB under Preferences -> Web, and these will be persisted across sessions if the Pre and Post Run Scripts from the previous section are included in your environment.

Alternately, you can define a startup.m file in your project to ensure the proxy settings are correct at every launch of MATLAB. The file would look something like the following.

```
% This startup file was created from the environment pre-run script, to set proxy settings for MATLAB
disp('Setting proxy configuration via startup.m file')
com.mathworks.mlwidgets.html.HTMLPrefs.setUseProxy(true)
com.mathworks.mlwidgets.html.HTMLPrefs.setProxyHost('proxy.company.com')
com.mathworks.mlwidgets.html.HTMLPrefs.setProxyPort('80')
com.mathworks.mlwidgets.html.HTMLPrefs.setUseProxyAuthentication(true)
com.mathworks.mlwidgets.html.HTMLPrefs.setProxyUsername('someusername')
com.mathworks.mlwidgets.html.HTMLPrefs.setProxyPassword('password123')
```

Put the following lines into your Pre Run Script to automatically generate such a file in any project where it does not exist, whenever this Compute Environment is used to launch a session. (Omit the Authentication, Username, and Password lines if your proxy does not use them.)

```
if [ ! -f /mnt/startup.m ]; then
    echo "% This startup file was created from the environment pre-run script, to set proxy settings for MATLAB" > /mnt/startup.m
    echo "disp('Setting proxy configuration via startup.m file')" >> /mnt/startup.m
    echo "com.mathworks.mlwidgets.html.HTMLPrefs.setUseProxy(true)" >> /mnt/startup.m
    echo "com.mathworks.mlwidgets.html.HTMLPrefs.setProxyHost('proxy.company.com')" >> /mnt/startup.m
    echo "com.mathworks.mlwidgets.html.HTMLPrefs.setProxyPort('80')" >> /mnt/startup.m
    echo "com.mathworks.mlwidgets.html.HTMLPrefs.setUseProxyAuthentication(true)" >> /mnt/startup.m
    echo "com.mathworks.mlwidgets.html.HTMLPrefs.setProxyUsername('someusername')" >> /mnt/startup.m
    echo "com.mathworks.mlwidgets.html.HTMLPrefs.setProxyPassword('password123')" >> /mnt/startup.m
```

(continues on next page)
Build and Test the Environment

Click **Build** when you have finished editing the above sections. You will be redirected to the *Revisions* page for the environment.

If the new revision builds successfully, you are ready to test! Go to the Project Settings for a project where you wish to use MATLAB and change the Compute Environment to use this new environment; once you do, you should see a MATLAB icon appear as an option for Workspaces in that project.

See the Troubleshooting section below for information on known issues and frequently encountered problems if anything does not work as expected.
## FAQ/Troubleshooting

### GPU functionality does not work as expected

The most common problem with GPU functionality is not having the correct NVIDIA driver version for your environment configured in your Hardware Tier. The following shell command should print some information about the GPU driver versions so you can verify it with the compatibility tables linked in the *Prerequisites* section.

```
>> !nvidia-smi
```

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Tesla V100-SXM2...</td>
<td>Off</td>
<td>00000000:00:1B.0</td>
<td>Off</td>
</tr>
<tr>
<td>N/A</td>
<td>39C</td>
<td>50W / 300W</td>
<td>0MiB / 16160MiB</td>
<td>0% Default</td>
</tr>
</tbody>
</table>

| 1 | Tesla V100-SXM2... | Off | 00000000:00:1C.0 | Off | 0 |
| N/A | 41C | 50W / 300W | 0MiB / 16160MiB | 0% Default |

| 2 | Tesla V100-SXM2... | Off | 00000000:00:1D.0 | Off | 0 |
| N/A | 40C | 52W / 300W | 0MiB / 16160MiB | 0% Default |

| 3 | Tesla V100-SXM2... | Off | 00000000:00:1E.0 | Off | 0 |
| N/A | 40C | 54W / 300W | 0MiB / 16160MiB | 4% Default |

<table>
<thead>
<tr>
<th>Processes: GPU Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU PID Type Process name Usage</td>
</tr>
<tr>
<td>=============================================================================</td>
</tr>
<tr>
<td>No running processes found</td>
</tr>
</tbody>
</table>

### MATLAB workspace fails to launch

#### License configuration problems

License issues are the most common cause of problems launching a MATLAB workspace. To debug, start a Jupyter or RStudio workspace (still using the MATLAB environment) and open a Terminal. Check if your license server information has been correctly entered into the `network.lic` file, in the correct part of the MATLAB directory structure (i.e. matching your MATLAB version):

```
$ which matlab
/usr/local/MATLAB/R2019b/bin/matlab
$ cat /usr/local/MATLAB/R2019b/licenses/network.lic
```

If that contains the license server information you expect, but there are still problems, you can use the `lmutil` utility to check that the license server is reachable and contains the correct information.

```
$ lmutil lmstat -a -c 27000@lic1.customer.com
```

It should output information on the licenses being managed by that license server, including the MATLAB licenses.
Advanced troubleshooting

The workspace launch process for MATLAB respects several environment variables related to logging, which may help you debug more obscure or advanced problems. You can set these as project-level environment variables, similar to how you would set environment variables for secure credential storage, to enable toggling them on or off without having to edit the Compute Environment.

- **WS_LOGGING=[MATLAB] | ALL | DEBUG.** By default, only the MATLAB logs are shown.
  - ALL shows logs for the underlying screen management `xterm` and `xpra`.
  - DEBUG shows everything from ALL, and activates a flag to show all commands.
- **TTY_MODE=[disabled] | PRE | POST.** Runs `ttyd` before or after the workspace. It’s a terminal that runs in a browser so you can easily inspect things.
  - PRE will run `ttyd` before the workspace launch.
  - POST will run `ttyd` after the workspace launch fails (if it fails).
- **START_LOGGING=[false] | true.** Increases logging verbosity in certain stages.

MATLAB workspace is laggy

In poor network conditions you may notice some lag in typing and general interactions with the MATLAB UI. This is to be expected in certain circumstances. Other common Domino workspaces like Jupyter or RStudio use browser-based IDEs, while this MATLAB environment is running a (Linux) desktop version of MATLAB in a remote desktop style interface. This results in more sensitivity to slow network conditions than a browser-based IDE.

For these situations, if you encounter them frequently, you may find it useful to install the Domino CLI on your local machine to facilitate editing files locally, then syncing them to Domino.

Keyboard shortcuts for copy/paste do not work

MATLAB in Domino runs the Linux version of MATLAB, which uses the “Emacs” set of keyboard shortcuts by default. To enable keyboard shortcuts Ctrl+C and Ctrl+V for copy and paste, go to Preferences -> Keyboard -> Shortcuts and change this from “Emacs” to “Windows”. See the MATLAB help for more details.

Connection error when downloading Add-ons or using `urlread`

If you get network errors when downloading Add-ons or using `urlread` to download files, it may be due to proxy settings. See the (Optional) Generate a startup script for proxy settings section above for options for setting your proxy in MATLAB either within the current session or more globally.

Toolbox is not available

A selection of Toolboxes is pre-installed in each MATLAB base image, but this may not include every Toolbox you have on your local installation. You can see the list of Toolboxes by running the `ver` command within MATLAB, and the list is also documented for each base image. We may release more base images in the future, with different selections, and you can contact your Domino account team with any specific requests for Toolboxes to include.

In addition to having the Toolbox installed in the base image, you must have an available license for using it on your license server.
Known Issues and Release Notes

If your problem is not listed in this FAQ, it may be worth checking the release notes for the base image you are using. Especially if your environment is built on an older image tag, checking the release notes of newer images may give you a quick answer about whether rebuilding on the most current image will help.

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See http://mathworks.com/trademarks for a list of additional trademarks.

Creating a SAS Data Science Workspace Environment

In this guide, we will walk through building a SAS Data Science Docker container image that will be integrated with Domino Data Lab. Before we dive in, we will answer a few common questions and provide additional resources.

Preparation

Before getting started, you will need a few things. Configuring and getting ready for these are outside of the scope of this guide.

Internet Access

Although you are not required to run the completed SAS Data Science container image on a Domino Data Lab environment that has Internet access, you will need Internet access to download the appropriate tools that are used to build the SAS Data Science container image.

Docker Client

We will be using a Docker CLI client to build the SAS Data Science container image. Although all of the commands shown can be copy/pasted, it is good to have some familiarity with the Docker CLI tools.

Docker Registry

A Docker Registry will be used to store the final SAS Data Science image before it can be consumed in Domino. There many options for Docker Registry providers and software. If you do not feel comfortable with setting up a Docker Registry to store the Docker images for your Domino Data Lab environment, please contact your Domino Customer Success Manager (CSM) or Technical Account Manager (TAM).

Git Client

We will be checking out a SAS Container Recipes Git repository. Although there are other ways to download this repository from the Internet, Git CLI will be used in this guide.

SAS Data Science License

This installation does require that you have a valid SAS Data Science license, which is provided to you by SAS Institute Inc. As part of the license, you should have a file called SAS_Viya_deployment_data.zip that will contain all of your license information and will be used to download the appropriate software.
Comfort with Linux Command-Line Utilities

All of the instructions in this guide are written for Red Hat Enterprise Linux variants. The instructions are primarily for CentOS 7, but can easily be adapted to support Red Hat Enterprise Linux, SuSE Enterprise Linux, or Oracle Linux, which are all supported by the SAS Data Science platform.

Please see the following page for Linux 64-bit operating systems that SAS Data Science (Viya family) supports: SAS Supported Operating Systems.

Creating a SAS Data Science Docker Image

The instructions for building the base SAS Data Science image that we follow are based on the SAS Container Recipes, which is available on the GitHub webpage below. Please consult the directions in the following GitHub repository for exact instructions for your situation: SAS Container Recipes.

In this guide, we will be building a SAS Data Science image with a CentOS 7 base. This will be a single Viya container instead of the full-blown Viya platform across multiple containers.

The general build instructions are as follows:

1. Clone the GitHub repository for SAS Container Recipes

   Shell Command

   1. `git clone https://github.com/sassoftware/sas-container-recipes.git`
   2. `cd sas-container-recipes`
   3. `cp PATHTO/SAS_Viya_deployment_data.zip .`

   Replace PATHTO above with the directory that contains your SAS_Viya_deployment_data.zip file.

2. Build the SAS Viya image using the build.sh utility provided

   Shell Command (cont)

   4. `./build.sh --base-image centos --base-tag 7 --type single --zip ./SAS_Viya_deployment_data.zip`

At the end of this process, you should have a SAS Data Science Docker image locally.

If you run into any issues, please contact your SAS Institute Inc. representative for support in resolving the issues.

Adding Additional Licensed SAS Software

Although it is outside the scope of this document, if you require installing any additional components like SAS/ACCESS modules or database drivers, please consult with your SAS representatives. These additional components can be layered on top of your base SAS Data Science Docker image.

Integrating the SAS Data Science Docker Image with Domino

We will now switch over to the Domino GitHub repository for the SAS Data Science image build. The Domino repository contains all of the files necessary to finalize the build of the SAS Data Science container image to make it integrated with Domino.

Please follow the README instructions on the Domino repository for more information about the individual files.
These are the steps you will need to follow to complete the build process:

1. Clone the Domino GitHub repository

<table>
<thead>
<tr>
<th>Shell Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

2. Modify the Dockerfile’s FROM instruction to use the SAS Data Science image you built in the prior steps

<table>
<thead>
<tr>
<th>Shell Command (cont)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

Please change NAME:TAG above to the Docker image tag that was created in the Creating a SAS Data Science Docker Image step.

3) Build the Docker image

<table>
<thead>
<tr>
<th>Shell Command (cont)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

Please change NAME:TAG above to your final Docker Registry image name and tag. This is the Docker image that will be later used inside of a Domino Compute Environment.

Testing the Docker Image Locally

Before pushing the Docker image to your Docker Registry, it is a good idea to test it locally first. There are two modes to test:

Interactive (SAS Studio)

<table>
<thead>
<tr>
<th>Shell Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

After a couple of minutes when you launch the interactive SAS Studio, you should see a message “SAS Studio is now running”. This is when you can visit http://localhost/SASSudio/start.html in your web browser to test SAS Studio.
Batch

<table>
<thead>
<tr>
<th>Shell Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 SAS_BATCH_PROGRAM=PROGRAM.SAS.</td>
</tr>
<tr>
<td>2 docker run -u domino:domino -w /mnt -v $PWD/tests:/mnt -it $DOMINO_SASDS_DOCKER_TAG run_sas.sh $SAS_BATCH_PROGRAM</td>
</tr>
</tbody>
</table>

Please change PROGRAM.SAS above with your test SAS program.

**Push the Domino-Integrated SAS Data Science Docker Image to a Docker Registry**

The final step is to push the Domino-integrated SAS Data Science Docker image to a Docker Registry. This Docker Registry will be later used to pull the Docker image into your Domino Data Lab environment.

<table>
<thead>
<tr>
<th>Shell Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 docker push $DOMINO_SASDS_DOCKER_TAG</td>
</tr>
</tbody>
</table>

Replace NAME:TAG with the Docker Registry tag you used in the Integrating the SAS Data Science Docker Image with Domino step.

Please work with your Domino Data Lab technical account team on the best method to pull the Docker image into your Domino Data Lab environment.

**Configuring the SAS Data Science Compute Environment in Domino**

Congratulations, you are near the end of the installation process. The last step is to configure your Compute Environment in your Domino Data Lab environment.

1. In your Domino Data Lab environment, navigate to the Domino Compute Environments page and create a new Compute Environment
2. Set the “Custom Image” location to your Docker Registry image. For the Custom Image URL, use the Docker Registry image URL that you created in the Push the Domino-Integrated SAS Data Science Docker Image to a Docker Registry step.
3. Create a Pluggable Workspace for SAS Studio in your Compute Environment
4. When you are done defining the Pluggable Workspace, click the Build button at the bottom of the Compute Environment page to finalize your SAS Data Science configuration for Domino Data Lab.

Maintenance and License Updates

The easiest way to keep your SAS Data Science updated is to repeat the steps in this guide whenever a new release of SAS Data Science is available. The same process should be repeated when you need to update a license file during renewals.

Repeating this process will ensure that you are staying current with the latest version of the SAS Data Science software.

Troubleshooting

**SAS Studio Timeout**

By default SAS Studio will log the user out after 30 minutes - so no further development can be done in that session, and changes not written to the filesystem cannot be saved.

The recommendation is to set timeout to a high value e.g. 24 hours.

In the SAS Data Science Compute Environment in Domino, set the following in the Dockerfile:

File: `/opt/sas/viya/config/etc/sysconfig/sasstudio.conf`

Setting:

```
export java_global_option_server servlet_session_timeout="-Dserver.servlet.
session.timeout=1440m"
```

This is a Spring Boot 2.0 property rather than a Studio property; use the ‘m’ for specifying minutes or an interval alone for seconds.
NB: this will be baked into the initial image build in future releases.

**SAS Studio Tabs Lost after Session Timeout**

To prevent tabs being lost after losing connection, configure the following option in Preferences.

```
  Preferences

  Start Up
  General
  √ SAS Programs
    Code and Log
    Results
  Editors
  Tables

  Start Up
  TABS TO OPEN
  ○ None
  ○ Restore tabs listed in Open Files pane
  ○ Open a new file
```

**Configuring ODBC connections**

Ensure that the `LD_LIBRARY_PATH` is set first, before individual ODBC libraries, as per the example below:

```bash
export SASINSIDE=/sasinside/odbc
export ODBCINI=/sasinside/odbc.ini
export ODBCINST=/sasinside/odbcinst.ini
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${SASINSIDE}/lib:${SASINSIDE}/lib/snowflake_odbc/lib
export SIMBAINI=/sasinside/odbc/lib/snowflake_odbc/lib/simba.snowflake.ini
```

**ERROR: Failed to load the Apache Parquet support extension**

Errors can be generated when trying to read Parquet files if the `LD_LIBRARY_PATH` has not been set correctly: please see *Configuring ODBC connections* above.

5.7 Publish your work

Domino makes it easy to publish your work through model APIs, web applications or Launchers.

5.7.1 Publish a Model API

Publish your model through the model API feature and control deployment settings, access, and collaboration.
Model publishing overview

- Introduction
- Overview
- Key features
- Environments for models
- Project files in models
- Publishing a model
- Calling a model
- Updating a model
- Troubleshooting

Introduction

For a quick video introduction to models in Domino, watch:

- Introduction to Models

Overview

Domino models are REST API endpoints that run your Domino code. These endpoints are automatically served and scaled by Domino to provide programmatic access to your R and Python data science code. You can use Domino models to quickly and easily put data science code into production.

A Domino model is a REST API endpoint wrapped around a function in your code. The arguments to your function are supplied as parameters in the request payload, and the response from the API includes the return value from your function.

When a model is published, Domino first runs the script containing the function. The process then waits for input, so any objects or functions from the script remain available in memory. Each call to the endpoint runs the function within this same process. Because the script is only sourced once — at publish time — any more expensive initialization can happen up front, rather than happening on each call.
Key features

• Production-ready infrastructure

Models are reliable, available, and scalable for most businesses’ production use cases. Read about model deployment configuration for more details.

• Versioning and reproducibility

Models are versioned and each version can be redeployed giving you the ability to revert to previous good states.

• Discoverability and access control

Domino models are first class objects in Domino, separate from projects.

– Models have their own permissions. Read about model access and collaboration for more details.
– Models have audit logging so you can track usage, management, and maintenance activity.
– Model endpoints can be set up to require access token authentication. Read about model invocation settings for more details.

• Promote-to-production workflow

Domino supports an advanced routing mode which allows for a promote-to-production workflow where you can test with one version of a model and take production traffic on another version. Read about model deployment configuration for more details.

Environments for models

Models run in Domino Environments, similarly to Runs and Workspaces. However, there are a few important details to note.

• Model hosts do not read requirements.txt files or execute commands defined in the pre-setup, post-setup, pre-run, or post-run scripts of your environment. If your project currently uses requirements.txt or any of these setup scripts to install certain packages or repositories, you must add them to the Dockerfile instructions of your environment.
• Your model does not inherit environment variables set at the project level. However, you can set model-specific environment variables on the model settings page. This is intended to decouple the management of projects and models. See this page for more details.

---

**Project files in models**

Your model has access to the project files for the project from which it was published. The project files are loaded onto the model host like they would be for an executor hosting a Run or Workspace, with few important differences:

• The project files are added to the model image when the model version is built. Stopping and starting an existing model version will not cause the files available to that model version to change. If your project files have changed since your current model version was built, you need to build a new version of the model if you want it to see those changes.

• Model hosts mount your project files at `/mnt/<username>/<project_name>`. This is different from the default behavior of a Run or Workspace, which hosts your project files at `/mnt`. There is a default Domino environment variable called `DOMINO_WORKING_DIR` that always points to the directory where your project is mounted, and allows you to easily write code that can work in both the standard run and model host environments.

• Git repositories attached to projects are only pulled when a model version is built, not every time a model is started. If your external Git repository changes, and you want to pick up those changes in your model, you should build a new version.

• Any project files mentioned in the `.modelignore` file present in the project’s root directory are excluded from the generated model image. Hence these excluded files are not mounted on the model host.

---

**Publishing a model**

There are three ways to publish a model.

1. **from the Domino web application**
   1. Click Publish from the project menu.
   2. Click to open the Models tab.
   3. Click New Model.
4. Fill in the first page of model setup by choosing a name for the model, supplying an optional description, setting the environment you want the model to run in, and choosing a logging mode. By default, Domino only logs basic origin and response code information about requests to the model. If you check the **Log HTTP requests and responses to model instance logs** box, Domino will also log the contents of the requests and responses, allowing you to see model inputs and outputs in the instance logs.

5. Click **Next** to advance to the second page of model setup. Enter the filename that contains your model code, and the function that you want called when the model handles a request. Optionally if you want to exclude any files from the model image, you can list those file patterns in a file named `.modelignore` and save the `.modelignore` file in the project’s root folder. Click **Publish** when finished. Your model will build, and upon a successful build it will automatically deploy.

### (2) with a scheduled run

When setting up a scheduled run, you will see an option to Publish Model after Complete. This setting will use the state of the project’s files after the run to build and deploy a new model version. You can use this option with a script that pulls fresh data from sources your model depends on to keep the model up-to-date automatically.

### (3) with the Domino API

Read the [API docs](#) for more information on programmatic model publishing.
Calling a model

On the overview page of a model, you will find a model tester. This can be used to make calls to the model from the Domino web application. You will find additional tabs on the overview with example code for calling the model with other tools and in various programming languages.

These examples all show a sample JSON input scheme. To construct your input JSON, you can use either a dictionary or an array. Each element of the list will be passed to the function as positional or named arguments. The elements themselves may be lists or arbitrary objects, as long as they are valid JSON.

If you’re using named parameters in your function definition, for example:

```python
my_function(x, y, z)
```

You can use either a data dictionary or a parameter array:

```json
{"data": {"x": 1, "y": 2, "z": 3}}
```

```json
{"parameters": [1, 2, 3]}
```

If you’re using a dictionary in your function definition, for example:

```python
my_function(dict)
```

and your function then uses `dict["x"]`, `dict["y"]` etc, you can use only a parameter array:

```json
{"parameters": [{"x": 1, "y": 2, "z": 3}]}
```

In Python, you can also use `kwargs` to pass in a variable number of arguments. If you do this:

```python
my_function(x, **kwargs)
```

and your function then uses `kwargs["y"]` and `kwargs["z"]`, you can use a data dictionary to call your model:

```json
{"data": {"x": 1, "y": 2, "z": 3}}
```

Domino will take care of converting the inputs to the proper types in the language of your endpoint function.

<table>
<thead>
<tr>
<th>JSON Type</th>
<th>Python Type</th>
<th>R Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>dictionary</td>
<td>dictionary</td>
<td>named list</td>
</tr>
<tr>
<td>array</td>
<td>list</td>
<td>list</td>
</tr>
<tr>
<td>string</td>
<td>str</td>
<td>character</td>
</tr>
<tr>
<td>number (int)</td>
<td>int</td>
<td>integer</td>
</tr>
<tr>
<td>number (real)</td>
<td>float</td>
<td>numeric</td>
</tr>
<tr>
<td>true</td>
<td>True</td>
<td>TRUE</td>
</tr>
<tr>
<td>false</td>
<td>False</td>
<td>FALSE</td>
</tr>
<tr>
<td>null</td>
<td>None</td>
<td>N/A</td>
</tr>
</tbody>
</table>
The model’s output is contained in the result object which can be a literal, array or dictionary.

### Updating a model

You can publish a new version of your model at any time. For example, you may want to re-train the model with new data, or switch to a different machine learning algorithm. Click New Version from the model overview page. The process is similar to when publishing a model for the first time.

You can also unpublish the model and Domino will stop serving it.

### Troubleshooting

**dataTypeError: don’t know how to serialize class**

You may see this error with Python endpoints if you return values that are NumPy objects, rather than native Python primitives. To fix this, just convert your NumPy values to Python primitives. An easy way to do this is to call `numpy.asscalar`.

**TypeError: <result> is not JSON serializable**

The result of your endpoint function gets serialized to JSON before being sent in the response. However some object types, such as Decimal or certain NumPy data types, are not JSON serializable by default. For Decimal data types, cast the Decimal type to a Float type. For NumPy data types, convert the values to Python primitives. An easy way to do this is to call `numpy.asscalar` as described above.

**Exception in thread “Thread-9” java.io.IOException: Server returned HTTP response code: 500 for URL <model_url>**
You may encounter this and other storage related errors in the build log when your Model is too large in size. Currently there is a project size limit of 500MB for Models. If your project includes a large trained model data set then we do recommend that you exclude this upon publishing a new version of your model.

**Video introduction to model publishing**

If you want to follow along with the example shown in the video, you can copy the snippets below:

1. Command to fetch zipcode lookup table:

```bash
wget https://s3-us-west-2.amazonaws.com/dominodatalab-fzd/free-zipcode-database.csv
```

2. Contents of `zip_pop.py` script used in the model:

```python
import pandas as pd
zips = pd.read_csv("free-zipcode-database.csv")

def zip_pop (my_zip):
    my_zip_pop = zips.loc[zips['Zipcode'] == my_zip]['EstimatedPopulation'].values[0]
    return my_zip_pop
```

**Model invocation settings**

**Overview**

There are two authorization modes for Domino models. You can select which mode you want your model to use from the **Invocation** tab of the model settings page.

- **Unrestricted mode**

  If your model is unrestricted, anyone with network access to Domino can call your model. No access token is required when sending a request.
**Restricted mode**

If your model is restricted, callers must send a valid access token along with their requests to use the endpoint. Check out the code examples in the model overview tab to see how to send access tokens with your requests.

**Access tokens**

On the **Invocation** tab, you can generate access tokens for your model. You can generate as many access tokens as you want. For convenience, you can use the name field to help you keep track of which tokens have been issued to whom, and for what purpose.

You can also revoke or regenerate a token by clicking the gear button.

**Model access and collaboration**

**Overview**

It’s easy to collaborate with your team on models, and to share your models broadly with interested consumers. There are two controls on the **Access & Sharing** tab of the model settings page that affect who has access to your model:

1. The model’s visibility setting
2. The model’s collaborators

**Model Visibility**

- Public: Viewable by anyone
- Private: Only viewable or discoverable by your collaborators

**Collaborators**

You can invite team members and organizations to this model. As an owner you have control over who can view or edit this model.

Learn more about collaborators

<table>
<thead>
<tr>
<th>Username</th>
<th>Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>domino-nickj</td>
<td>Nicholas Jablonski</td>
<td>Owner</td>
</tr>
</tbody>
</table>

Add Collaborators
Visibility settings

There are two visibility options:

• **Public**
  – Anyone with access to your Domino deployment can search, discover, and view your model.
  – Only collaborators can modify or deploy model versions or settings.

• **Private**
  – Only collaborators can search, discover, and view your model.
  – Only collaborators can modify or deploy model versions or settings.

Manage collaborators

To grant other users access to your model, go to the Access & Sharing tab of the model settings page and scroll down to the Collaborators section. You can add new collaborators by their username or email address, and you can also add organizations as collaborators, granting permissions to all members.

The owner of a project can set different access levels for collaborators.

• **Viewers**

  Can only view the model versions and logs. They will not be able to view or edit model settings, or publish new versions. A viewer cannot see any access tokens.

• **Editors**

  Can deploy new versions if they have collaborator access to the underlying project. They can view logs and audit history, and they can change most settings. They cannot invite new collaborators or change model visibility. An editor can see all access tokens and
create new ones.

- **Owners**

  Have all permissions as above and they can invite new collaborators, change visibility, and transfer ownership. An owner can see and revoke all access tokens and create new ones.

**Model deployment configuration**

**Overview**

Domino model APIs are scalable, high-availability REST services. The Deployment tab of the model settings page allows you to configure three important things for your model:

1. The compute resources available to your model hosts
2. The number of model hosts serving your model
3. The number of routes – or versions – you want to expose

### Deployment Configuration

Configure the details of your model deployment. Learn more about Deployment Configuration

<table>
<thead>
<tr>
<th>Overview</th>
<th>Versions</th>
<th>Audit Log</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access &amp; Sharing</td>
<td>Deployment</td>
<td>Environment</td>
<td>Invocation</td>
</tr>
</tbody>
</table>

#### Scale Configuration

- **Number of Instances**
  - 2 Instances

#### Resource Quota

- **Compute Resources per Instance**
  - default: 0.25 cores / 0.146484375 GB RAM

#### Routing Mode

- **Basic mode**
  - In this mode, the traffic simply gets redirected to the latest deployed version. You can reactivate an older version which would take over the traffic from the latest version.

- **Advanced mode**
  - This mode allows you to have two deployed versions concurrently. You can promote one version to production, which would be accessible from a labeled route. You have to manually deactivate the model versions that are no longer in use.

**Scaling your model**

There are two dimensions on which to scale your model.

1. **Horizontal scale**

   You can select the number of model hosts that you want running at any given time. Domino will automatically load-balance requests to the model endpoint between these hosts. A minimum of 2 instances
allows you to have a high-availability model and is the default selection. Domino supports up to 32 instances per model.

2. **Vertical scale**

You can choose a hardware tier that will determine the amount of RAM and CPU resources available to each model host.

When you change either of these selections, your model will be restarted with the new settings.

**Routing your model**

Domino supports two routing modes.

1. **Basic mode**

   In this mode, you only have one route exposed that always points to the latest successfully deployed model version. When you deploy a new one, the old one is shut down and replaced with the new one while maintaining availability. The route has the following signature:

   *Latest*: `/models/<modelId>/latest/model`

2. **Advanced mode**

   In this mode, you can have two running versions - a promoted version and a latest version. This allows you to have a workflow where your clients always point to the promoted version and you can test with the latest. When the latest version is ready for production, you can seamlessly switch it to be the promoted version with no downtime. The routes have the following signature:

   *Latest*: `/models/<modelId>/latest/model`

   *Promoted*: `/models/<modelId>/labels/prod/model`

**Promoting projects to production**

When a project is ready to be promoted to production, there are often reasons to continue experimenting with the model even after the project is deemed production-grade. When that happens, it’s often useful to make sure others with access to the project do not haphazardly or accidentally make changes to the production version.

In Domino, you can use the Organizations feature to accomplish this. If one does not already exist, create a new organization that will contain only projects that are production-ready, or already in production. Transfer ownership of your production-grade project to this organization before you begin experimenting.

**Note**: All Domino users with access to this organization should clearly understand its purpose.

To experiment on this project, fork it into a new project, following the process described in the official Domino documentation, and conduct your experiments in the new project.

When you are ready to merge your changes back into the original, production-grade project, click on Request Review to submit a merge request to the project owner.
Note: The process of forking and merging is covered in detail in the official Domino documentation. If you are not familiar with the process, we recommend you consult that link before trying it yourself.

Some Domino customers do this even for models that are not actually in production, as a way of submitting their work to their managers.

Other ways to promote to production

While changing organizational ownership of a project is usually the simplest way to promote a project, Domino offers three other methods.

Change Results Behavior

By saving runs to isolated branches instead of in the main line, you can ensure that the results of each run will be independent from each other, and not synchronized. One user’s run will then have no impact at all on the next user’s run.

To change results behavior, first click Settings in the left-hand sidebar. Then, in the Project Settings window, click the Results tab.

![Project settings](image)

Under Results behavior, select To isolated branches.

Exporting the Project

You can also use Imports/Exports to export files from your production-designated project into a separate working project. The code in the original designated project would then be read-only, while any actual code execution would occur in the working project.

To use this feature for promoting a project to production, click on Imports/Exports in the left-hand sidebar. Then, under Exports, check the checkbox labeled Files. This will ensure that any project that imports this one will use the current set of files for your production-designated project.
Create and name a new project, and then import the files from your original, production-designated project. This new project will be your working project.

Additionally, you can use Domino’s run tagging feature to let users in the working project select which version of the production-ready project they’d prefer to use. Simply assign the “release” tag to any runs you want to make available to these users.

### Project Tagging

A third way to promote a project to production is to use Domino’s Project Tagging feature to designate a project as being production-ready. However, this option is only available for on-premises or private deployments.

Click on **Overview** in the left-hand sidebar. In the box labeled **Select Tags**, begin typing the appropriate tag ("prod" or “promote,” for example).

If the tag has already been used, Domino will prompt you with an auto-fill option for the **Select Tags** text box. Otherwise, you will have the option to add a new tag.

Once the project is tagged, make sure all users with access to the project understand the tag’s meaning.
Exporting Model Image

Some times you may want to build models in Domino but host them in an environment outside Domino. This is for reasons such as you may have already made huge investments in the production environment outside Domino that supports very high scale or low latency, the production data can not be exported outside of an environments due to legal/compliance issues, you want to control deployment using a custom CI/CD pipeline, etc.

Domino allows you to export model images built in Domino to an external container registry. These images include all the information needed to run the model including the model code, artifacts, environment, project files, etc.

Domino has exposed REST APIs that can be called by your CI/CD pipeline or workflow to programmatically build and export the model Image. By default the images are built in Model API format. These images can be easily deployed in an environment that can run docker containers. However, you can also export images which are in AWS Sagemaker-compatible format so that they can be directly deployed in AWS Sagemaker. Following APIs are now available:

**Build Model Image API**

This API builds a docker image for a model and stores it in Domino’s internal Registry. This can later be fetched from the registry (using other export APIs) by your CI/CD pipeline. The exported Image can be deployed in the your runtime env (outside Domino). Your CI/CD pipeline can add more layers to this image to do further customizations (such as adding Auth, etc).

**Build Model Image Status API**

Given a modelId and modelVersionId, this API will return the status of the build operation.

**Build Model Image Logs API**

Given a modelId and modelVersionId, this API will return the logs for the build operation.

**Export Model Image API**

This API can be used to Push a Model Image to a 3rd party container registry outside Domino. It assumes the Image was already built and available within Domino. As part of the API request, users need to provide credentials for their registry to push Image to it. These credentials are not saved inside Domino and can have a TTL attached.

**Export Model Image Status API**

Given a exportID, this API will return the status of the export operation.

**Export Model Image Logs API**

Given a exportID, this API will return the logs for the export operation.

**Export Model Image For Sagemaker API**

This API builds a docker image for a given version of an Model API in a AWS Sagemaker-compliant format and then exports it to AWS ECR or any 3rd party container registry outside Domino. As part of the API request, users need to provide credentials for their registry to push Image to it. These credentials are not saved inside Domino and can have a TTL attached.

Sagemaker has the capability to train and then deploy a model to serve requests. Model export functionality in Domino only supports the serve use case since train operation would have already happened in Domino. The Image is ready to be deployed in the Sagemaker environment. All the necessary files required to make predictions are packaged inside the container.

**Note:** This API only builds and exports an Sagemaker-compliant image to AWS ECR or any 3rd party container registry outside Domino. It does not automatically deploy the image inside Sagemaker. Users need to take care of ‘deploying’ on Sagemaker as part of their CI/CD or deployment scripts.
5.7.2 Publish a web application

Use Domino to publish web applications in Dash, Flask, or Shiny and easily control permissions and track usage.

App publishing overview

- Overview
- How do I publish an App?
- How do I view my App?
- Who can see my App?
- Where do I find Apps in Domino?

Overview

When experiments in Domino yield interesting results that you want to share with your colleagues, you can easily do so with a Domino App. Domino Apps host web applications and dashboards with the same elastic infrastructure that powers Jobs and Workspace sessions.

Domino supports hosting Apps built with many popular frameworks, including Flask, Shiny, and Dash. Apps are first-class objects in Domino, and Domino includes features that allow for easy sharing, monitoring, collaborating, and iterating on Apps.

How do I publish an App?

Apps are published from Domino Projects. To publish an App, you need to:

1. Have all of your application code in the project files for the Project you want to publish from, or loaded into your Project from an external git repository.

2. Configure your application to serve from a host address of 0.0.0.0 on port 8888. This is the host and port Domino will use when directing users to your application server.
3. Have an `app.sh` file in the project. Domino will look for and execute a shell script named `app.sh` after creating the hardware that will host your App. Put all commands required to launch your application in `app.sh`.

When your application code is ready for publishing, and you’ve set up `app.sh`, click **Publish** from the project menu, then click **App**. Give your App an informative name and description, choose a permissions setting, and toggle the **Show in Launchpad** checkbox to control whether your App appears in the Domino Launchpad.

Click **Publish** when you’re finished.

Once your App is published, the **App** link from the project menu will navigate you to the App settings page, where you can click **View App** to open your App or copy a link to it for sharing with your colleagues.
Publish App

Publish a web application from your project files. You must have a script named app.sh in your project’s root directory that launches your application. Learn more about Apps.

App Status: Running

• Copy App Link

View App  Stop  View Execution Details

Hardware Tier

Turn off app to change hardware tier.

For complete end-to-end examples of App publishing, check out these tutorials:

• Getting started with Dash (Python App framework)
• Getting started with Shiny (R App framework)
• Publishing a Flask web app in Domino

Remember these key facts about App publishing

• Your App will run on the same Domino execution hardware your project uses normally. Make sure your environment has all the dependencies your application requires.
• Your application must be configured to serve from a host address of 0.0.0.0 on port 8888.
• The performance of your App will depend on the design of the underlying application. Read more about designing applications for performance.

How do I view my App?

The App settings page has a View App button that can be used to open your App while it is running. You can also copy a persistent URL that can be used to access and share your App.

You can also see all of your own Apps from the Launchpad.
A dashboard showing the history of Runs hosting your App can be seen on the **App Versions** tab of the App settings page.

**Who can see my App?**

Project owners, contributors, and results consumers automatically have access to an App. To control more general access to an App, use the Permissions tab on the App settings page.
Under **Access Permissions** are the following four options:

- **Anyone, including anonymous users**

  Any request to the App URL will be served the App. This setting means that anyone with network access to Domino can view your App. This is useful for sharing Apps with people on your network who do not have Domino accounts.

- **Anyone with an account**

  All users who have Domino accounts and are logged in to Domino can view your App.

- **Invited users (other users may request access)**

  All users who have Domino accounts and are logged in to Domino can request access to your App if it appears in the Launchpad, but cannot view the App until the owner grants the request.

- **Invited users only**

  Only Domino users who are added by the App owner via the **Invite People** field can view the App. Users cannot request access.

Use the **Invite People** field to send email invites. Domino users who receive an invite will be able to access the App.
Users who request access will appear provisionally in the **Who has access** table, and their requests can be granted or denied from controls in the **Status** column.

---

**Where do I find Apps in Domino?**

All Apps that are configured for **Show in Launchpad** will appear in the Domino Launchpad.

![Launchpad with Apps](image)

This is the primary interface for Domino users who want to consume Apps. You can click on an App in this list to see its description, settings, recent usage, and a link to either view the app or request access.

---

5.7. **Publish your work**

417
When viewing an App in Domino, App consumers have access to a toolbar with controls to view the App description, and to contact the App owner.
Video introduction to App publishing

The video below is a recording of a webinar held on March 28, 2019.
The webinar provides an overview of App publishing in Domino, and walks through some example demonstrations.

NOTE: Direct SSH access to running Apps (referenced at 31:50 in the video below) has been deprecated in Domino 4.0

Click here to view and download the slides from the webinar presentation.

For more information about Apps, read:

- App publishing overview
- Getting started with Dash
- Getting started with Shiny
- App scaling and performance

Getting started with Dash

- Overview
- Set up environment
- Set up project
- Publish
- Share and consume

Overview

This article will show you how to publish a Python App with Dash in Domino.

In this tutorial you will:
- configure a Domino environment with the necessary dependencies to publish a Dash App
- create a project and set it up for App publishing
• publish an App to the Domino launchpad
• observe how other users in Domino can use the App

You’ll be working with the second example from Basic Dash Callbacks, part of the Dash User Guide. In this example, the application serves an interactive scatter plot of countries by GDP per capita and life expectancy.

It will take approximately 15 minutes to get this example running in Domino.

Set up environment

The first step is to create a Domino compute environment capable of running your App.

1. From the Lab, click Environments.
2. From the Environments Overview, click Create Environment.
3. Give your environment a descriptive name and description, and select Domino Analytics Distribution Py3.6 R3.4 from the Environment dropdown under Base Image. This selection means that the setup instructions we provide for this environment will be applied on top of a base image with Python 3.6 and some analytics modules already installed. Read Domino standard environments to learn more about the contents of this base image.
4. Click Create Environment when finished.
5. You will be directed to the Overview tab for your new environment. Click Edit Dockerfile.
6. In the Dockerfile Instructions field, paste in the following instructions:
# Install the libraries we want to use in our app

```
RUN pip install dash==0.22.0 && 
    pip install dash-renderer==0.13.0 && 
    pip install dash-html-components==0.11.0 && 
    pip install dash-core-components==0.26.0 && 
    pip install plotly --upgrade
```

7. Click **Build** when finished.

You will be directed to the Revisions tab for your environment. Here you’ll be able to monitor the build process for your new version of the environment. If the build succeeds, you’re ready to use this environment for App publishing.

---

**Set up project**

The next step is creating a project with the settings and content you need to publish your App.

1. From the Lab, click **Projects**.
2. Click **New Project**.
3. Give your project an informative name, then click **Create Project**.
4. Click **Settings** in the project sidebar, then set the Compute environment to the one you created in the previous step.
5. Click **Files** in the project sidebar, then click **Add File**.

   ![Compute environment]

6. Name the file `app.py` in the title field above the editor.
7. In the body of the file, paste the following example App code.

   ```python
   import dash
   import dash_core_components as dcc
   import dash_html_components as html
   import pandas as pd
   ```

(continues on next page)
```python
import plotly.graph_objs as go
import os

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminderDataFiveYear.csv')

app = dash.Dash()

# Configure Dash to recognize the URL of the container
user = os.environ.get("DOMINO_PROJECT_OWNER")
project = os.environ.get("DOMINO_PROJECT_NAME")
runid = os.environ.get("DOMINO_RUN_ID")
runurl = '/' + user + '/' + project + '/r/notebookSession/'+ runid + '/

app.config.update({
'routes_pathname_prefix': runurl,
'requests_pathname_prefix': runurl
})

# Define layout
app.layout = html.Div(style={'paddingLeft': '40px', 'paddingRight': '40px'}, children=[
dcc.Graph(id='graph-with-slider'),
dcc.Slider(id='year-slider',
        min=df['year'].min(),
        max=df['year'].max(),
        value=df['year'].min(),
        step=None,
        marks={str(year): str(year) for year in df['year'].unique()}
    )
])

@app.callback(
dash.dependencies.Output('graph-with-slider', 'figure'),
dash.dependencies.Input('year-slider', 'value'))
def update_figure(selected_year):
    filtered_df = df[df.year == selected_year]
    traces = []
    for i in filtered_df.continent.unique():
        df_by_continent = filtered_df[filtered_df['continent'] == i]
        traces.append(go.Scatter(
            x=df_by_continent['gdpPercap'],
            y=df_by_continent['lifeExp'],
            text=df_by_continent['country'],
            mode='markers',
            opacity=0.7,
            marker={
                'size': 15,
                'line': {'width': 0.5, 'color': 'white'}
            },
            name=i
        ))
    return {
```

'data': traces,
'layout': go.Layout(
    xaxis={
        'type': 'log', 'title': 'GDP Per Capita'},
    yaxis={
        'title': 'Life Expectancy', 'range': [20, 90]},
    margin={'l': 40, 'b': 40, 't': 10, 'r': 10},
    legend={'x': 0, 'y': 1},
    hovermode='closest'
)

if __name__ == '__main__':
    app.run_server(port=8888, host='0.0.0.0', debug=True)

Make note of the code block after `app = dash.Dash()`. When serving a Dash application from Domino, you must configure Dash to serve from a relative path, instead of the default root path. This configuration is different in different Dash versions, but this code snippet has been tested for Dash versions 0.22 and higher. If you need to support an older version of Dash, please contact Domino support.

8. Make note of two important variables in the final line of the file. Domino-hosted applications must run on a host of `0.0.0.0` and listen on port `8888`. These are the settings Domino will expect when directing users to your App.

9. Click **Save** when finished.

10. The last thing to do before publishing your App is to create an `app.sh` file. This is a Bash script that Domino runs after initializing the host that will serve your App. It should contain all commands required to launch your App. In this example, the only command you need is `python app.py`. Create this file the same way you did for `app.py`, then save it.

---

**Publish**

Now you’re ready to publish your App.

---

5.7. Publish your work

1. Click **Publish** from the project sidebar.

2. Give your App an informative title and description, and set Permissions to *Anyone can access*. This will allow anyone with a network connection to your Domino deployment to access the App if they have the URL.

<table>
<thead>
<tr>
<th>App</th>
<th>Models</th>
<th>Launchers</th>
<th>API Endpoint</th>
</tr>
</thead>
</table>

Publish a Flask or Shiny App that can be shared across your organization. You must have a script titled `app.sh` in your Domino file directory. One app per project is currently supported. [Learn more about Apps](#)

   - View app.sh file

   - **Show in App Store**: ✓
   - **Title**: My first Dash App
   - **Description**: Interactive scatterplot charting GDP per capita and life expectancy over time.


   You can format the text using Markdown. **Preview**

   - **Permissions**: Anyone can access

3. Click **Publish**.

4. Once the App status says *Running*, click **View App** to load your App. You should see the interactive scatterplot with a Domino toolbar above it showing the project it’s published from, plus buttons to email the App owner and open the description panel.
Share and consume

Now that your App is published, if you set the permissions to *Anyone can access*, you can now easily share it with colleagues who have access to your instance of Domino. You can try this out yourself by opening a private or incognito browser, or logging out of Domino, and navigating to the App URL.

Getting started with Shiny

- Overview
- Set up project
- Publish
- Share and consume
Overview

This article will show you how to publish an R App with Shiny in Domino.

In this tutorial you will:

- create a project and set it up for App publishing
- publish a simple two file Shiny App to the Domino Launchpad
- observe how other users in Domino can use the App

You’ll be working with the Telephones by region example from the Shiny gallery. In this example, the application serves an interactive bar chart of consumer telephones by region from 1951 to 1961.

It will take approximately 10 minutes to get this example running in Domino.

Set up project

The first step is creating a project with the settings and content you need to publish your App.

1. From the Lab, click Projects.

2. Click New Project.

3. Give your project an informative name, choose the Private visibility setting, then click Create Project.

4. Click Settings in the project sidebar, then set the Compute environment to Domino Analytics Distribution Py3.6 R3.4. Read Domino standard environments to learn more about the contents of this base image.

5. Click Files in the project sidebar, then click Add File.

6. Name the file server.R in the title field above the editor. It’s important that the file be named exactly this, as launching a two file Shiny application requires a directory to contain files named server.R and ui.R.

7. In the body of the file, paste the following example Shiny server code.

```r
# Rely on the 'WorldPhones' dataset in the datasets
# package (which generally comes preloaded).
library(datasets)
```

(continues on next page)
# Define a server for the Shiny app
function(input, output) {

  # Fill in the spot we created for a plot
  output$phonePlot <- renderPlot({

    # Render a barplot
    barplot(WorldPhones[,input$region]*1000,  
             main=input$region,  
             ylab="Number of Telephones",  
             xlab="Year")
  })
}

8. Click Save when finished.

9. Click Add File again, and name the new file ui.R.

10. Paste in the following content, then click Save when finished.

```r
# Rely on the 'WorldPhones' dataset in the datasets  
# package (which generally comes preloaded).
library(datasets)

# Use a fluid Bootstrap layout
fluidPage(

  # Give the page a title
  titlePanel("Telephones by region"),

  # Generate a row with a sidebar
  sidebarLayout(

    # Define the sidebar with one input
    sidebarPanel(
      selectInput("region", "Region:",  
                   choices=colnames(WorldPhones)),
      hr(),
      helpText("Data from AT&T (1961) The World's Telephones.")
    ),

    # Create a spot for the barplot
    mainPanel(
      plotOutput("phonePlot")
    )
  )
)
```

11. The last thing to do before publishing your App is to create an app.sh file. This is a Bash script that Domino runs after initializing the host that will serve your App. It should contain all commands required to launch your App. In this example, the only command you need is:

```bash
R -e 'shiny::runApp("./", port=8888, host="0.0.0.0")'
```

Make note of two important parameters in this command. Apps in Domino must run with a host of 0.0.0.0 on port 8888. This is where Domino will direct users to your application. Create the app.sh file the same
way you did for `server.R` and `ui.R`, then save it.

**Publish**

Now you’re ready to publish your App.

1. Click **Publish** from the project sidebar.

2. Give your App an informative title and description, and set Permissions to **Anyone can access**. This will allow anyone with a network connection to your Domino deployment to access the App if they have the URL.

```
<table>
<thead>
<tr>
<th>Details</th>
<th>WorldPhones</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show in App Store</td>
<td>✔</td>
</tr>
<tr>
<td>Title</td>
<td>WorldPhones</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Telephones by region, 1951 to 1961.</td>
<td></td>
</tr>
<tr>
<td><a href="https://shiny.rstudio.com/gallery/telephones-by-region.html">From the Shiny gallery.</a></td>
<td></td>
</tr>
</tbody>
</table>

You can format the text using Markdown [Preview](#) |

```

3. Click **Publish**.
4. Once the App status says *Running*, click **View App** to load your App. You should see the interactive bar chart with a Domino toolbar above it showing the project it’s published from, plus buttons to email the App owner and open the description panel.

![Image of a bar chart showing Telephones by region]

**Share and consume**

Now that your App is published, if you set the permissions to *Anyone can access*, you can now easily share it with colleagues who have access to your instance of Domino. You can try this out yourself by opening a private or incognito browser, or logging out of Domino, and navigating to the App URL.

**Getting started with Flask**

The following guide will set up your Domino project more as a proper web site powered by Flask. This isn’t difficult; you just need to make sure all the folders and files are in place to ensure the app runs correctly.

**Creating the App Structure**

First, you’ll need to create the app structure. The following is what we’ll end up with and will discuss each area in turn.

Note that /mnt is your Files directory on Domino. Also, the following names matter: `app.sh`, `__init__.py`, `/static`, `/templates`. Besides those, everything can be customized to your liking in this article.
Create the folders

In the Files section of Domino make a new folder called flask_app. This will be a Python package that will contain the web app.

In the flask_app folder put two other folders, one called static the other templates. In the static folder you put your static files and in the templates folder you put your template files. The static and templates folders will be recognized by Flask to know from where to fetch the appropriate files. It’s possible to change which folder is noticed as the static folder, but common practice is to just leave it as static.

Create the files

Create an __init__.py file in the flask_app folder. This file will make flask_app a Python package. In that file put the following:

```python
from flask import Flask
class ReverseProxied(object):
    def __init__(self, app):
        self.app = app
    def __call__(self, environ, start_response):
        script_name = environ.get('HTTP_X_SCRIPT_NAME', '')
        if script_name:
            environ['SCRIPT_NAME'] = script_name
            path_info = environ['PATH_INFO']
            if path_info.startswith(script_name):
                environ['PATH_INFO'] = path_info[len(script_name):]
        return self.app(environ, start_response)

app = Flask(__name__)
app.wsgi_app = ReverseProxied(app.wsgi_app)
```

Back in your Files section of Domino create a new file called run.py.

In run.py add the following:
from flask_app import app
from flask_app import views

if __name__ == '__main__':
    app.run()

Change your app.sh file in the Files section of Domino to:

```bash
#!/usr/bin/env bash
export FLASK_APP=run.py
export FLASK_DEBUG=1
python -m flask run --host=0.0.0.0 --port=8888
```

Customization

Now we’re going to add a views.py file. This file is where you place all your app logic and should be created in the flask_app directory.

We’ll make a very simple app that just returns “Hello World!”

Add the following to views.py:

```python
from flask_app import app

@app.route('/

def index():
    return "Hello World!"
```

If you have a template in the templates folder, you would add:

```python
from flask import render_template
from flask_app import app

@app.route('/

def index():
    return render_template('hello.html')
```

Testing

Now once you publish the app, when you go to your app url, you should see “Hello World!”

Cross-Origin Security in Domino web apps

Published Domino web apps are served in an HTML Inline Frame, or iframe. By default, Domino enables the “sandbox” attribute for the iframe. The “sandbox” attribute applies extra security restrictions to your web app, like blocking
cross-origin requests, form submissions, script executions, and much more. To learn more about these extra security restrictions, please read the “sandbox” attribute notes in the following resource: HTML Inline Frame Attributes.

**Warning:** Your web app may not function properly if it uses AJAX requests or stores information in cookies or localstorage. If you’d like to change this behavior, please contact your Domino administrator.

**Advanced web application settings in Domino**

**App scaling and performance**

**Overview**

Launching an App in Domino works the same as any other Domino run. Domino assigns hosting of your App to an executor machine in the **hardware tier** your App is configured to use. That executor then retrieves the default Domino **environment** configured for your project, and creates a container based on that image. Domino then loads your project files onto the machine and executes the **app.sh** file that you have authored and placed in the project root, at which point your application will be running in its container.

Depending on which hardware tier you select, the container running your application may share a host machine with other containers or run on a dedicated host. Your selection of hardware tier allows you to specify available memory and CPU. However, Domino Apps do not automatically scale horizontally to multiple hosts. Your App can scale vertically to use all available resources on the executor host machine by correctly configuring the underlying application.

The following sections describe configuring web applications to optimize scalability and performance for several popular frameworks.

**Flask and Dash (Python)**

By default, Flask and Dash will run single-threaded on a single process. The authors of Flask do not recommend this configuration if you are going to serve more than 10 users concurrently, or for any externally consumed applications. The **Flask documentation** provides many ways to serve the application in a more scalable way.

For example, you can serve a Flask application through **gunicorn**. To do this in Domino, change the project’s **app.sh** file from:

```bash
python app.py
```

to

```bash
gunicorn -w 4 -b 0.0.0.0:8888 myproject:app
```

This will start serving the Flask application on 4 processes.

The performance and scalability of your App will depend on the compute demands of your application, and the compute resources available on the host machine. If there is a command in your application that will use 100MB
RAM and 20% of a standard VM CPU, then an executor host machine with 1 core and 1 GB RAM could handle 5 concurrent users running that command without suffering reduced performance. A 6th user attempting to run the command would cause the App’s performance to suffer. There would be RAM available, but not enough CPU cycles.

Shiny (R)

Like Python Apps, your Shiny Apps’ performance will depend on design of the underlying application. While multiple users can view Shiny applications in independent sessions, R is a single process language. This means that multiple users can view and interact with the App in their own isolated session, but only one can do any processing at a time regardless of the memory or CPU of the machine.

Shiny Apps typically cannot scale to more than a handful of concurrent users.

- Make Shiny fast by doing as little work as possible

Hosting HTML pages from Domino

Summary

This is a simple example on how to host a HTML page on Domino. There are a number of ways to host web applications of course but this example shows how you can integrate a simple HTTP server using python with Domino. You could also add java script and other pages to your project. The example in this note just shows how you would start the server to support your page(s).

Files

You’ll need to create two files in your project (in addition to your files required for your page such as index.html etc)

app.sh

```bash
#!/usr/bin/env bash
python ./app.py
```

app.py

```python
import http.server
import socketserver
PORT = 8888
Handler = http.server.SimpleHTTPRequestHandler
httpd = socketserver.TCPServer(('*', PORT), Handler)
print "serving at port", PORT
httpd.serve_forever()
```

5.7. Publish your work
Publishing

To publish your app, go to the left-hand sidebar in your browser and click Publish.

The Publish page should now be visible in your browser. Click the App tab. Your screen should now look something like this.

Click **Publish** to publish your app.

Getting started with App Publishing.

How to get the Domino username of an App viewer

Overview

Domino passes the username of a user accessing your Domino App in an HTTP header named `domino-username`. If your App framework gives you access to the HTTP headers of the active request, you can retrieve the username for use by your App code. If you allow users who are not logged in to Domino to view your Apps, for their requests to the App the value of the `domino-username` header will be “Anonymous.”

Prerequisite
An App framework that can access proxied HTTP headers. This is supported by Flask and Dash by default, but is only supported by Shiny if using Server Pro.

**Example**

Consider this simple Flask example where you have the following files in your project.

An app.sh script that starts your app code listening on the correct port.

```bash
#!/usr/bin/env bash
export LC_ALL=C.UTF-8
export LANG=C.UTF-8
export FLASK_APP=app.py
export FLASK_DEBUG=1
python -m flask run --host=0.0.0.0 --port=8888
```

A simple app.py file that renders a template named `index.html`. Note that this app imports `request` from flask, which gives you access to the headers of the active HTTP request.

```python
import flask
from flask import request, redirect, url_for

class ReverseProxied(object):
    def __init__(self, app):
        self.app = app
    def __call__(self, environ, start_response):
        script_name = environ.get('HTTP_X_SCRIPT_NAME', '')
        if script_name:
            environ['SCRIPT_NAME'] = script_name
            path_info = environ['PATH_INFO']
            if path_info.startswith(script_name):
                environ['PATH_INFO'] = path_info[len(script_name):]
        return self.app(environ, start_response)

app = flask.Flask(__name__)
app.wsgi_app = ReverseProxied(app.wsgi_app)

@app.route('/
def index_page():
    return flask.render_template("index.html")

A template file at templates/index.html that fetches the domino-username header from the requests object and renders it.

```html
<!DOCTYPE html>
<html>
  <body>
    <h1>Your username is {{ request.headers.get("domino-username") }}</h1>
  </body>
</html>
```

If you host this App in Domino and open it, you’ll see something like the following, where the username shown will match the username of the user viewing the App.

5.7. Publish your work
5.7.3 Launchers

Create a self-service web form for your results consumers with Domino’s Launchers feature.

Launchers overview

- About Launchers
- Basics
  - The command
  - Outputs
  - Parameters
    - Writing your code to process parameter values
      - Python argument handling
      - R argument handling
- Full examples in R and Python
  - R example
  - Python
    - Writing the script
- Building the Launcher
  - Using the Launcher
About Launchers

Launchers let you turn your analyses into self-service web forms that less technical colleagues can interact with. They are great for creating templatized reports and analyses, so stakeholders can answer questions without bothering data scientists.

You can build a Launcher to let consumers of your project upload data and specify parameters that are neatly consumed by your code to produce consumer-visible results. For example, you could give your users something like this:
That would produce for them something like this:
A Launcher is essentially a web form on top of any script you could run in Domino. Any arguments your script or executable expects can be exposed as UI elements in the web form.

The command

When you create a Launcher, you specify a command that runs under the hood. This command serves as a template: when an end-user runs the Launcher through the web form, parameters in your command template will be replaced with the user’s input values, and the resulting command will run.
Outputs

When your code runs, it runs just like anything else in Domino. Namely, Domino will detect new files your code produces and treat those as the results. Whoever runs your Launcher will get a link to those results to view on the web, and they’ll get an email when the results are ready. Your code can produce rich images, even interactive HTML dashboards, dynamically based on a user’s input.

Parameters

Like any other script you run through Domino, your command can take parameters/arguments. Anything in your command of the form `${param_name}` will be treated as a parameter in the Launcher.

Your parameters can be of the following types:

- **Text**: normal text field
- **Select**: drop down where you can select one value from a list
- **File**: button to select and upload files
- **Multiselect**: list where you can select multiple values

An end user would see those parameters rendered like this in the final web form:
Note:
When parameter values are executed in a Job started by the launcher, those values are encased in single quotes. This preserves spaces and other special shell characters in the values, however it means that such values cannot be used to expand to environment variables.

Writing your code to process parameter values

When an end-user runs your Launcher through the web form, the user’s input values will be passed into your command in place of the corresponding placeholders you specified, and that final command will be run as though it were a command-line executable. That means your underlying code can access parameters using any standard method for reading command-line inputs. The most common techniques would be:

Python argument handling

Use `sys.argv`.

```python
import sys
p1 = sys.argv[1]
```

(continues on next page)
R argument handling

Use the commandArgs function.

```r
args <- commandArgs(trailingOnly=TRUE)
p1 <- args[1]
p2 <- args[2]

# a file upload parameter
print(readLines(args[3], n = 1))

# a multi-select parameter
for (each in strsplit(args[4],",")) {
  cat(each, sep="\n")
}
```

Note that file parameters will be passed into your file as the path to the file. Multi-select parameters will be passed in as the comma-separated list of all selected choices.

Full examples in R and Python

R example

Our simple R example lets users input two numbers, and behind the scenes, we run some R code that adds them and prints the sum. Our script is in a file called launcher.R, and we could tell Domino to run `launcher.R 10 20`, so our Launcher’s command will be `launcher.R ${A} ${B}`.

```r
args <- commandArgs(trailingOnly = TRUE)
a <- as.integer(args[1])
b <- as.integer(args[2])
```
if (is.na(a)) {
    print("A is not a number")
} else if (is.na(b)) {
    print("B is not a number")
} else {
    paste("The sum of", a, " and", b, " is:", a+b)
}

The example Launcher is set up to take A and B as parameters.

When operational, the user will see:
R launcher

Reusing values from your most recent use of this launcher.

Returns the sum of 2 numbers (A+B)

A

5

B

3

Title your run. This will appear on the Runs dashboard.

Run title

Notify other users when this run is done by filling in their emails

e.g., 'chris@foo.com, jim@bar.com' or 'John Smith <john@foobar.com>; Ali

Close  Run
Python

This Python example uses a script that creates an interactive scatter plot, using Bokeh, from a CSV file that anyone can upload using a web form. The user provides (a) a file (b) what to put on the X- and Y- axes and (c) some information how to color the data points. The Python script generates an interactive Bokeh scatterplot.

Writing the script

This is the complete code of the Python script itself.

```python
from bokeh.plotting import show, output_file
from bokeh.charts import Scatter
import pandas as pd
import sys

output_file("scatter.html")

data = pd.read_csv(sys.argv[1])
scatter = Scatter(data, x = sys.argv[2], y = sys.argv[3], color = sys.argv[4], legend = "top_left")
show(scatter)
```

Building the Launcher

The Launcher itself needs 4 parameters.

Their exact names aren’t important for the script. The parameter names are human-readable guidance for the users of the Launcher. The order of the parameters is directly linked to the script. Rename them to File, X, Y, Color. After renaming the parameters, you should change their types. In the overview of the parameters, click File and use the dropdown next to Type to select Upload File. X, Y and Color can remain type Text.

Make sure to put in a clear name for the Launcher. Your result should look something like this:
Using the Launcher

Click **Launchers** from the project menu then click **Run**.
As a sample dataset, use `scanvote.csv` containing the percentage of a population voting “Yes” per district in Finland, Sweden and Norway.

With this data I would like to create a scatter plot with population (Pop) as X, “Yes” vote percentage as Y (Yes), and the points colored based on the country (Country).

Putting this in the launcher form will give the picture below. Click Run and wait for the result,
**Advanced launcher editor**

**Note:** This feature is currently only available in private deployments of Domino Enterprise.

You can access the JSON representation of a Launcher by clicking its settings button, then clicking **Edit (Advanced)**.
You can copy Launcher definitions between projects using this JSON editor.

5.7.4 Assets portfolio overview

Prerequisites Domino 4.0+

Overview

Users in Domino can author and publish several types of data products and assets, including:

- Model APIs
- Apps
- Launchers
- Scheduled Jobs
These are valuable, lasting resources that are meant to deliver results for consumers and colleagues. To make it easy to discover assets, users in Domino 4.0+ can find all of the above types of assets they have access to in the Assets Portfolio in the Control Center.

**Opening the Assets Portfolio**

To open the Assets Portfolio, use the Switch to menu to open the Control Center, then click Assets in the Domino main menu.
Navigating the Assets Portfolio

The Assets Portfolio will show assets in projects that you own or have been added to as a collaborator. Additionally, Domino System Administrators will see assets in all projects across the Domino instance, and Domino Project Managers will see assets in all projects owned by users in their organizations.

When first opened, the Assets Portfolio will show Model APIs by default. Click one of the four asset type buttons at the top of the Assets Portfolio to view that type of asset.

For each asset type, the table of information will include unique columns that provide useful metrics and information specific to that asset. You can select which columns you want to view with the column-picker above the table, and you can filter the table by typing a query in the text box next to it.
5.8 Connect to your data

Connect to a database, git repository, or cluster in Domino to easily access your data. The datasets feature also provides a high-performance, versioned filesystem for storing large files in Domino.
5.8.1 Domino Datasets

Datasets overview

- **Overview**
- **Writing to a local Dataset**
- **Reading from a shared Dataset**
- **Managing Datasets**
Overview

Domino Datasets provide high-performance, versioned and structured filesystem storage in Domino. With Domino Datasets, you can build multiple curated pipelines of data in one project, and share them with your fellow contributors across their projects.

A Domino Dataset is a series of Snapshots. Each Snapshot is a completely independent state of the Dataset, and represents the contents of a filesystem directory from the time when the snapshot was written. There are two key ways to interact with a Domino Dataset:

1. you can write a new snapshot to one of your project’s local Datasets
2. you can read from an available snapshot of a shared Dataset you have mounted

Writing to a local Dataset

Domino Datasets belong to Domino projects. Permission to read and write from a dataset is granted to project contributors, just like the behavior of project files. A Dataset that belongs to a project is considered to be local to that project. To create a new Dataset in your project, click Datasets from the project menu, then click Create New Dataset.
Supply a name and optional description, then click **Upload Contents**. The upload page provides four ways to write to your local dataset.

1. **Browser Upload**

   In Domino 3.5+ you can use the Upload Files section to queue up to 50GB or 50,000 individual files for upload through your browser. You can pause this upload and resume within 24 hours. You can upload directories and subdirectories to preserve your filesystem structure.

   ![Create a New Snapshot](image)

2. **CLI Upload**

   After installing and configuring the Domino CLI, you can copy and paste the displayed command to upload a directory of files from your local machine to the Dataset. Note that all contents of the directory you specify are
written to the Dataset.

For the example shown above, if the files you want to write to the Dataset are in /Users/myUser/data, you would run the following command:

domino upload-dataset njablonski/datasets-demo/test /Users/myUser/data

When finished, click Complete. You will then be taken to the Dataset overview where you should see a new Snapshot has been written. The new Snapshot will contain exactly those files that were in the folder you uploaded from your local machine.

3. Upload by Running Script

Before using this method, you need a script in your project files that is configured to write to the target Dataset. Supply the name of a Bash, Python, or R script and click Start to launch a Job. During the Job, an empty folder will be available at the path shown in Output Directory. At the conclusion of the Job, any files that your script has written to the output directory will be written to your Dataset as a new Snapshot.
In the example above, the **Output Directory** is `/domino/datasets/main/output`. For the simplest possible example, you could run the below script for a situation where there is a file named `data.csv` in your project.

```bash
write-dataset.sh
```

```bash
cp $DOMINO_WORKING_DIR/data.csv /domino/datasets/output/<Dataset_Name>
```

When the script runs, it will copy the data file to the Dataset output directory. Then, when the Job is finished, Domino will write a new snapshot to the Dataset. The new Snapshot will contain the exact contents of the output directory, which in this case is just the `data.csv` file.

### 4. Upload by Launching a Workspace

This method works similarly to uploading by running a script. You will have all of the usual options available from your Domino environment for launching a Workspace. When the Workspace is launched, an empty folder will be available at the path shown in **Output Directory**. When you stop and sync the Workspace, any files that you have written to the output directory will be written to your Dataset as a new Snapshot.

There is a configurable limit to the number of snapshots a Dataset may contain. This limit defaults to 20 Snapshots. If your Dataset is at this limit, attempting to start an upload with any of the above methods will result in an error message. Before you can write additional Snapshots, you will need to delete old Snapshots or increase the limit. Talk to your local administrator for more information.

You can create a maximum of 5 local Datasets per project. If you are setting up a pipeline that requires more than 5 Datasets, use separate projects for each logical task that import Datasets from the project that precedes them in the pipeline.
Reading from a shared Dataset

To access the contents of an existing Dataset snapshot, you must mount the target Dataset in your project. To mount a Dataset, click **Datasets** from the project menu, then click **Mount Shared Dataset**.

Click the **Dataset to Mount** field to see an autocomplete dropdown of Datasets you have access to. To access a Dataset, you must be an Owner, Contributor, Project Importer, or Results Consumer on the project that contains the Dataset.

There are three different settings under Update Behavior that control which Snapshot of the target Dataset your project will mount. You can mount the latest Snapshot, a tagged Snapshot, or a fixed Snapshot number. When finished, click **Mount**.

Now, on the Datasets page for your project you will see the Dataset you mounted listed under **Shared Datasets**.
The Path shown for the Dataset points to a directory where you will find the file contents of the mounted Snapshot in your project’s Runs and Workspaces. When mounted this way, the Dataset is read-only.

Managing Datasets

From the Datasets page of your project, click the name of a local or shared Dataset to open its overview page. At the top of the overview page you will see the Dataset name and description, plus buttons to upload to, rename, or archive the Dataset.

Below the description is a panel with Dataset details. Use the dropdown menu at the top of the panel to select a Snapshot, then the panel will display a list of the files it contains, plus some metadata about the Snapshot.

There are two important actions you can take on a Dataset snapshot:

1. Add Tag

   Above the Snapshot selection dropdown you will find a list of tags applied to the Snapshot, followed by a + Add Tag button. Tags can be used to identify a Snapshot when mounting a shared Dataset for input. This allows the
Dataset owner to tag a Snapshot for production use, and move the tag to whichever Snapshot is in the desired state as the Dataset changes over time.

2. Mark for Deletion
   
   Clicking the Mark for Deletion button in the lower right of the panel will mark the currently selected Snapshot for deletion, changing its status. Such Snapshots can no longer be mounted in Runs by consuming projects. The Snapshot will be flagged to a Domino administrator as ready for deletion, but will not be fully deleted until the administrator takes an additional action to delete it, at which point the data is erased and no longer recoverable.

Video introduction to Domino Datasets

To learn more about Datasets, read:

- Datasets overview
- Datasets advanced mode tutorial
- Datasets best practices

About domino.yaml

Overview

domino.yaml is a file that defines Dataset configurations. It isn’t there by default, and should be created at the root of your project.

A Dataset configuration controls:

1. Existing Dataset Snapshots and how those Snapshots are mounted for input.
2. New directories that can become Snapshots and how those directories are mounted for output.

Schema

The domino.yaml file respects the following schema. Spaces matter.

```yaml
datasetConfigurations:  # contains array of configurations
  - name: string  # identifier for this configuration
    inputs:  # contains array of datasets to mount for input
      - path: string  # path appended to /domino/datasets
dataset: string  # name of the dataset to mount as input
    outputs:  # contains array of datasets to mount for input
```

(continues on next page)
Valid fields in the YAML object are:

- **datasetConfigurations**
  
  This is a required field. It must be the first very first field on the first line. Only one of these fields can exist in the YAML file. This will contain an array of individual Dataset configurations.

- **name**
  
  Identifier for a specific configuration.

- **path**
  
  Desired mount path for Dataset Snapshot or new Snapshot directory.

- **dataset**
  
  Name of dataset. If configured as input, the latest Snapshot of the Dataset will be mounted by default. A different tagged Snapshot can be specified using a colon, like `dataset-name:{tag}`. For example: `iris:test`

- **inputs**
  
  Contains array of one or more [path, dataset] specifications to be mounted for input.

- **outputs**
  
  Contains array of one or more [path, dataset] specifications.

---

**Error Handling**

If you attempt to use an invalid `domino.yaml`, you may see one of these categories of error.

- **Invalid field that Domino does not recognize in a particular position**

  The error indicates the field found and shows valid field options for that position.

  *Example*

  ```yaml
  Found invalid field in domino.yaml: "output".
  Valid field options: "inputs", "outputs", "name"
  ```

- **Valid field that Domino recognizes in a particular position, but there is an error.**

  An example of this is two outputs fields in one name block.

  *Example*
There is an error in domino.yaml encountered while processing field “outputs”. Please check all your “outputs” fields.

- **Valid field that Domino recognizes in an incorrect position.**
  
  An example of this is a valid field with the wrong indentation.

  **Example**
  
  There is a formatting error in domino.yaml encountered while processing field “dataset”. Please check all your “dataset” fields.

- **Syntax error**
  
  An example of this is a missing quote. In some cases, we can identify the region the error occurs.

  **Example**
  
  There is a formatting error in domino.yaml in the block near line 12.

- **Catch all**

  We are having trouble parsing domino.yaml. Please see the support article linked above. If you still cannot identify the problem, please email support@dominodatalab.com about your problem and include your domino.yaml.

Datasets advanced mode tutorial

- **Overview**
- **Creating Datasets**
- **Setting up an initial Dataset configuration**
- **Writing to an output Dataset**
- **Reading from an input Dataset**
- **Appending to a Dataset**
- **Reverting a Dataset to a previous snapshot**
- **Sharing a Dataset in advanced mode**
- **domino.yaml schema**
Overview

This article will guide you through a hands-on tutorial on using Domino Datasets in advanced mode. Advanced mode allows you to set up multiple customized dataset configurations, and swap between them from one Run to the next.

If you want to understand the basic features of Domino Datasets, read the Datasets overview.

In this tutorial, you’ll be working with data from the Climate Analysis Indicators Tool (CAIT) via World Resources Institute. If you follow along you’ll learn how to write, read, append, revert, and manage Domino Datasets.

This tutorial will take you approximately 30 minutes to complete.

Creating Datasets

Domino Datasets belong to Domino projects. Permission to read and write from a Dataset is granted to project contributors, just like the behavior of project files. For your first step in this tutorial, you should create a new project.

This project will be used to ingest, process, and store data.

Navigate to your Domino application, then click + New Project from the landing page. Give the project an informative name, set its visibility to Private, then click + Create Project.
You’ll be automatically redirected to the Files page for the new project. From the project menu, click **Datasets**. This page will show information about the Datasets in this project, plus any shared Datasets that are mounted. Since this is a new project, there are no Datasets to display here. Click **Create New Dataset** to get started.
For the purpose of this tutorial, you’ll want to create two new Datasets. One to store raw imported data, and one to store some derivative processed data. Name the first Dataset `cait-raw`, give it an informative description like the one shown above, then click **Upload Contents**.

The Dataset will be created, and you will then be directed to the upload page. You can ignore this page for now, since this tutorial focuses on writing to Datasets with advanced mode. Click Datasets from the project menu again, and you’ll see the Dataset you just created listed with zero active Snapshots.
Now that your Datasets are created, the next step is to configure your project to use them in advanced mode.

Setting up an initial Dataset configuration

When you want to interact with a Dataset from your Domino project in advanced mode, you must mount the Dataset. There are two ways to do so:

1. Mounting a Dataset as an *input Dataset* makes the contents of a specific snapshot (the most recent, by default) available in a directory at the specified mount point in your Workspace or Run. A Dataset mounted only for *input* cannot be modified.

2. Mounting a Dataset as an *output Dataset* creates an empty directory at the specified mount point, and when your Run or Workspace is stopped a new snapshot is written to the Dataset with the contents of that directory. Note that the new snapshot will only contain exactly those files that are in the mounted output directory. Snapshots do not append by default.
It's important to note that the same Dataset can be mounted for input and output simultaneously at different mount points, which will be important when we perform append and revert operations later. For now, we'll set up our project to mount the `cait-raw` Dataset for output, so we can populate it with some data.

Dataset configurations for Domino projects are controlled by a file named `domino.yaml`. If Domino sees this file in the root of your project, it will attempt to read it and make available the Dataset configurations specified within.

The `domino.yaml` file doesn’t exist by default, so from the Files page of your project, click the Add File button.

Name the file exactly `domino.yaml`. You can read the full YAML scheme for this configuration file [here], but for now you can just copy and paste the following markup to create a configuration that mounts `cait-raw` for output.

```yaml
datasetConfigurations:
  - name: "PopulateRaw"
    outputs:
      - path: "raw-output"
        dataset: "cait-raw"
```

The three important values in this configuration are:

1. "PopulateRaw" is the name you give this configuration so you can identify and select it when starting a Workspace or Run.
2. "raw-output" is the directory name that will be mounted to receive new output at `/domino/datasets`.
3. "cait-raw" is the name of the Dataset you want to mount.

Once you’ve filled in the filename and contents, click Save.

You’re now set up to write data to a Dataset.
Writing to an output Dataset

If you’ve created a valid `domino.yaml` file in the root of a project, you’ll see an option to select the configurations defined within when launching a Run or Workspace. To populate some initial data into `cait-raw`, you should start up a Jupyter Workspace with the `PopulateRaw` configuration selected.

From the project menu, click Workspaces. Click Jupyter, give the workspace session an informative name, then click the Advanced tab in the Datasets panel. Select the PopulateRaw configuration in the dropdown menu, then click Launch Jupyter Workspace.

If you get an error saying that a valid Dataset configuration file was not found, double-check that your file is correct YAML, uses spaces instead of tabs for indentation, and is named exactly correctly with no spaces before or after the filename.

In your new Jupyter Workspace, click New > Terminal to access the executor shell.
Domino has made some of the CAIT data available in a public bucket on Amazon S3. Run the following commands to fetch two files containing data on CO2 emissions by country for the years 2010 and 2011.

```
wget https://s3.amazonaws.com/dominodatalab-cait/country-emissions-2010.csv
```

When finished, you should have the two downloaded files in your /mnt working directory.

To write these files to your output Dataset, you need to copy them to the mount path set in your Dataset configuration. In the PopulateRaw configuration, the output Dataset was mounted at `raw-output`. That directory path gets appended to a base path of `/domino/datasets`.

```
mv country-emissions-2010.csv /domino/datasets/raw-output/
mv country-emissions-2011.csv /domino/datasets/raw-output/
```

Now, the output mount directory contains the two files you want to write to the next snapshot for the output Dataset.
To write the snapshot, all you need to do is stop your Workspace session. Click **Stop** from the top menu, then **Stop and Commit** in the prompt. Domino will detect that there is data in the output mount, and will write a new snapshot.

Back in Domino, open the **Datasets** page. Click the name of the output Dataset (**cait-raw**) to view details on it, and you’ll see the snapshot you just wrote to the Dataset.

You now have a populated Dataset that you and other contributors to your project can use to access data for analysis and transformation.
Reading from an input Dataset

In this step, you’ll read in the data from the raw Dataset, transform it, and write it to a new Dataset. The first step is to create a new Dataset configuration.

From the Files page of your project, click the filename of `domino.yaml` to open the file, then click **Edit**. Add the following new Dataset configuration at the end of the file, then click **Save**.

```
- name: "WriteTrend"
  inputs:
    - path: "raw-input"
      dataset: "cait-raw"
  outputs:
    - path: "trend-output"
      dataset: "emissions-trend"
```

When finished, your `domino.yaml` file will describe two configurations:

1. the **PopulateRaw** configuration you used in the previous step
2. the new **WriteTrend** configuration that mounts the `cait-raw` Dataset for input, and the `emissions-trend` Dataset for output

Using the new **WriteTrend** configuration, you can write code that reads from the input Dataset, performs some processing or analysis operations on the data within, then writes to the different output Dataset.

For this step you will write and execute a Python script as a Domino Run, rather than using a Domino Workspace. The same operations could be done in a Workspace if desired, but using a batch Run will create a repeatable step in a data pipeline, which you could run every time the input Dataset changes to get an updated output Dataset.

From the Files page of your project, click **Add File**. Name the file `calculate-trend.py`, paste in the Python script below, then click **Save**.

```
from __future__ import division
import pandas as pd
import os
import glob

# load all files from input datasets as pandas dataframes
(continues on next page)
df_list = []
data_dir = "/domino/datasets/raw-input/"
data_files_list = glob.glob(data_dir+'*')

# function takes two raw MtCO2 columns and returns percentage change
def percentage_change(year1_df, year2_df):
    year1_data = year1_df['CO2 emissions (Mt)']
    year2_data = year2_df['CO2 emissions (Mt)']
    change = (year2_data-year1_data)/year1_data * 100
    change_formatted = str(round(change,2))+'%'
    return change_formatted

# create output dataset
emissions = pd.DataFrame()
emissions['Country'] = pd.read_csv(data_files_list[0])['Country']

# write new columns for each year on year pair
for i in range(len(data_files_list)):
    if i == 0:
        continue
    df_year_1 = pd.read_csv(data_files_list[i-1])
    df_year_2 = pd.read_csv(data_files_list[i])
    year1 = str(df_year_1.loc[0,'Year'])
    year2 = str(df_year_2.loc[0,'Year'])
    column_name = "Emissions change " + year1 + " to " + year2
    emissions[column_name] = percentage_change

# send data to output dataset
output_file = "/domino/datasets/trend-output/emissions-trend.csv"
emissions.to_csv(output_file)

To run this script with the WriteTrend Dataset configuration, click Jobs from the project menu, then click Run at the top of the Runs list. Enter calculate-trend.py as the file you want to run, then below that click the Advanced Datasets configuration tab and choose WriteTrend from the dropdown menu.
Click **Start Run** to execute your code. When finished, you’ll see a new snapshot written to the `emissions-trend` Dataset, containing the transformed data from `cait-raw`. Every time you run `calculate-trend.py` with this Dataset configuration, it will read and process the latest snapshot of `cait-raw` and write a new snapshot of `emissions-trend`.

In the next step, you’ll learn how to append to a Dataset by adding a new file to `cait-raw`.

---

### Appending to a Dataset

Suppose you receive a fresh batch of raw data, in this case a new file with data from 2012 that you want to store alongside the data from 2010 and 2011. The logical operation you want to do is append that file to the existing content of the last snapshot. The procedure for appending to a Domino Dataset involves mounting it for both input and output simultaneously.

Remember that by default, mounting a Dataset for input makes available the files in the most recent snapshot, and mounting a Dataset for output provides an empty directory, the contents of which will become the next snapshot at the end of the Run or Workspace session.

The high-level steps to an append are:

1. Start a Run or Workspace session with the Dataset mounted for both input and output
2. Copy the contents of the input mount to the output mount

---

5.8. Connect to your data
3. Add the data you want to append to the Dataset to the output mount

To continue this tutorial example, you first need to write a new Dataset configuration. From the Files page of your project, click the filename of domino.yaml to open the file, then click Edit. Paste the following new Dataset configuration at the end of the file, then click Save.

```yaml
- name: "AppendRaw"
  inputs:
    - path: "raw-input"
      dataset: "cait-raw"
  outputs:
    - path: "raw-output"
      dataset: "cait-raw"
```

This new AppendRaw configuration mounts the cait-raw Dataset for both input and output. The input mount will be at /domino/datasets/raw-input and the output mount will be at /domino/datasets/raw-output.

Now you can perform your append operation by starting up a Domino Workspace with the AppendRaw configuration.

1. From the project menu click Workspaces.
2. Select Jupyter.
3. Give the Workspace an informative name.
4. Click to open the Advanced tab in the Datasets panel.
5. Choose AppendRaw from the dropdown menu.
6. Click Launch Jupyter Workspace.

In your Jupyter Workspace, click New > Terminal to access the executor shell. Follow these steps to complete your append operation:

1. Fetch the new file with 2012 data.
wget https://s3.amazonaws.com/dominodatalab-cait/country-emissions-2012.csv

2. Copy the previous snapshot of the Dataset from the input mount to the output mount.

cp /domino/datasets/raw-input/* /domino/datasets/raw-output/

3. Move the new file to the output mount.

mv country-emissions-2012.csv /domino/datasets/raw-output/

4. Click **Stop** then **Stop and Commit** to end this session and write the new snapshot of the Dataset.

If you examine the cait-raw Dataset from the **Datasets** page of your project, you’ll see a new snapshot with the 2012 file appended to the contents of the previous snapshot. Now, if you want to, you can start a fresh Run of **calculate-trend.py** with the WriteTrend configuration, to update the emissions-trend Dataset with the 2012 data.

---

**Reverting a Dataset to a previous snapshot**

Reverting a Dataset is similar to appending, in that you’ll mount the Dataset for input and output simultaneously. However, instead of mounting the latest snapshot for input, you’ll mount a specified previous snapshot that you want to revert to. Suppose there was an issue with the 2012 file you added in the previous section, and you want to go back to the initial snapshot of cait-raw.

To identify a specific snapshot in your Dataset configuration, you need to tag the snapshot. From the Files page of your project, click the **Datasets** tab. Next, click the name of the cait-raw Dataset.

From the dropdown menu, choose the snapshot you want to tag, in this case **Snapshot 0**.

Click the **Add Tag** button below the Dataset name in the upper left, then fill in the string you want to tag the snapshot with. In the below example the snapshot is tagged with “good.”
When finished, you'll see a blue tag with the string you entered appear next to the + Add Tag button. You'll now need to write a Dataset configuration that mounts the tagged snapshot of the Dataset. From the Files page of your project, click the filename of `domino.yaml` to open the file, then click Edit. Paste the following new Dataset configuration at the end of the file, then click Save.

```yaml
- name: "RevertRaw"
  inputs:
    - path: "raw-input"
      dataset: "cait-raw:good"
  outputs:
    - path: "raw-output"
      dataset: "cait-raw"
```

This is similar to the append configuration, but note that the input Dataset name is entered as `cait-raw:good`. This is how to set up a dataset configuration to mount a tagged input snapshot. A colon is appended to the Dataset name followed by the tag string.

Now you can perform your revert operation by starting up a Domino Workspace with the `RevertRaw` configuration.

1. From the project menu click Workspaces.
2. Select Jupyter.
3. Give the Workspace an informative name.
4. Click to open the Advanced tab in the Datasets panel.
5. Choose RevertRaw from the dropdown menu.
6. Click Launch Jupyter Workspace.

In your Jupyter Workspace, click New > Terminal to access the executor shell. Follow these steps to complete your revert operation:

1. Copy the tagged snapshot contents from the input mount to the output mount.

   ```bash
   cp /domino/datasets/raw-input/* /domino/datasets/raw-output/
   ```

2. Click Stop then Stop and Commit to end this session and write the new snapshot of the Dataset.

If you examine the `cait-raw` Dataset from the Datasets tab on the Files page of your project, you'll see a new Snapshot 2 with only the original two files from Snapshot 0 in it.
Sharing a Dataset in advanced mode

To give another user access to the Datasets in your project, you need to add them to the project as a Contributor, Results Consumer, or Project Importer. Once your colleague has been granted one of those permissions on the project, he or she can refer to your Datasets in a domino.yaml file with the scheme:

`<your-username>/<your-project-name>/<dataset-name>`

For example, to mount the `emissions-trend` Dataset from the above examples as an input Dataset, your colleague would use a configuration like this, noting that `documentation` in this path is the username:

```yaml
datasetConfigurations:
  - name: "import"
    inputs:
      - path: "cait-input"
        dataset: "documentation/cait-data/emissions-trend"
```

domino.yaml schema

The domino.yaml file respects the following schema.

```yaml
datasetConfigurations: # contains array of configurations
  - name: string # identifier for this configuration
    inputs: # contains array of datasets to mount for input
      - path: string # path appended to /domino/datasets
        dataset: string # name of the dataset to mount as input
    outputs: # contains array of datasets to mount for input
      - path: string # path appended to /domino/datasets
        dataset: string # name of the dataset to mount for output
```

Datasets best practices

This article describes how to use Domino Datasets to solve problems, improve collaboration, and open new workflow possibilities in Domino.

5.8. Connect to your data
You can use Datasets to...

- Store more files, bigger files, and access them faster
- Build multiple curated collections of shareable data in your project
- Track production and testing states of your data
- Simplify working with Domino locally
- Automatically pipe data from external sources into Domino

Store more files, bigger files, and access them faster

When you start a Run or launch a Workspace, Domino copies your project files to an executor. When working with large volumes of data, this presents three potential problems:

1. The number of files that can be stored in Domino project files may exceed the configurable limit. By default, the limit is 10,000 files.
2. There is a limit to the size of any individual file that can be transferred to and from your Domino project files. By default, this limit is 8GB.
3. The time required to transfer data to and from the executor is proportional to the size of the data. It can take a long time if the size of the data is very large, leading to long startup and shutdown times for Runs and Workspaces.

You can solve these problems with Domino Datasets:

1. There is no limit to the number of files that can be stored in a Domino Dataset.
2. There is no limit to the size of any individual file stored in a Domino Dataset.
3. Domino Datasets are attached to executors as networked filesystems, removing the need to transfer their contents to the executor when starting a Run or Workspace.
Build multiple curated collections of shareable data in your project

If you use project imports and exports to share data with other members of your team, the consumers of your project will receive the entire contents of your project files in their Runs and Workspaces. That works well if your project is small, simple, and narrowly scoped.

However, for large projects that produce many data products, you may want to expose them to your consumers in smaller, curated subsets. You can do this with Domino Datasets.

Consider the project shown below.

This project has a small folder full of code, plus nine folders full of various kinds of various output data. Each data folder is larger than 10GB, and the whole project is 100GB. It would be impractical to ask your data consumers to import this project, but you also don’t want to separate the data from the code that produced it by moving the data to a different project.

The solution is to organize the data into Datasets, with one Dataset for each type of data your consumers are interested in. In this example, suppose you have two colleagues who want to consume your data. One of them is only interested in the data from the experiment1 folder, and the other is only interested in the data from experiment9.

You can follow the instructions on the Dataset overview to create and write to two Datasets with scripts like the following, where it’s assumed you have named the Datasets experiment1-data and experiment9-data.

experiment1-populate-dataset.sh

```bash
cp -R $DOMINO_WORKING_DIR/experiment1/. /domino/datasets/experiment1-data/output
```

experiment9-populate-dataset.sh

```bash
cp -R $DOMINO_WORKING_DIR/experiment9/. /domino/datasets/experiment9-data/output
```
Your consumers can then mount only the Datasets they are interested in.

If you are working with data at this scale, you should write it to Datasets whenever you produce it, instead of storing it in your project files.

You can execute your experiment code from the Datasets upload interface, and make the Dataset output directory the destination for your data products. If you want to write to multiple Datasets in the same Run, check out the Datasets advanced mode.

Track production and testing states of your data

If you have a Dataset that is being used by downstream consumers for critical work, tagging allows you to continue to improve, process, and experiment with new Snapshots without impacting those consumers. When you have improved data ready for use, you can switch which Snapshot is tagged, and your tag consumers will automatically start getting your new data.

Consider the Dataset shown below.
The Dataset has three active Snapshots. If you decide that you want consumers of this Datasets to work from Snapshot 1, since Snapshot 2 represents an experimental modification of the data that you are not yet confident in, you can apply a tag like `prod` to Snapshot 1.

When your consumers mount the Dataset in their projects, they have the option to mount whichever Snapshot is marked with a given tag. When they choose the **Pin your snapshot at a certain tag** update behavior, they will see a dropdown menu of available tags.
When you are confident that your experimentation has produced a new Snapshot that is ready for production use, you can remove the **prod** tag from Snapshot 1, and apply it to the new Snapshot. Your consumers will then automatically see the newly tagged Snapshot mounted in their Runs and Workspaces. Note that trying to apply the tag again without first removing it from the previously tagged Snapshot will result in an error.

**Simplify working with Domino locally**

If you use the **Domino CLI** to work with projects to your local machine, you may find that storing large data files slows down your **download** and **sync** operations, and fills up a lot of your local disk storage. You can prevent this by storing data in a Domino Dataset, and reserving your project files for the scripts and documents you want to work with locally.
Follow these steps to simplify your local workflow:

1. Create a Dataset in your project, and write your large data files to it.
2. Once the files have been written to the Dataset, you can remove them from your project files.
3. Fetch a fresh, slimmed-down copy of your project.
4. Update your code to reference your data files in their new location, at:
   /domino/datasets/local/<dataset-name>/
5. When everything is working smoothly, you can delete any copies of the project from your local machine that have the large data files in them.

Automatically pipe data from external sources into Domino

If you have data in an external source that you want to periodically fetch and load into Domino, you can do so with scheduled Jobs set up to write to Datasets with advanced mode.

Suppose you have data stored in an external data source that is periodically updated. If you wanted to fetch the latest state of that file once per week and load it into a Domino Dataset, you could use the following process to set up a scheduled Run.

1. Create a Dataset to store the data from the external source.
2. Write a script that fetches the data and writes it to the Dataset.
3. Set up an advanced mode configuration to bridge between your script and your Dataset.
4. Create a scheduled job to run your script with the new Dataset configuration.

Below is a detailed example showing how to fetch a large, dynamic data file from a private S3 bucket with a scheduled Run once per week.

First, create a Dataset to hold the file. This example shows the Dataset being named fetched-from-S3.
After clicking **Upload Contents**, the Dataset will be created. However, instead of using one of the UI options to perform an upload, you should instead click Files from the project menu, then click Add File to start creating the script for your scheduled Run.

For this example, assume the S3 bucket is named *my_bucket* and the file you want is named *some_data.csv*. In that case, you can set up your script like this:

```
import boto3
import io

# create new S3 client
client = boto3.client('s3')

# download some_data.csv from my_bucket and write to latest-S3 output mount
file = client.download_file('my_bucket',
                           'some_data.csv',
                           '/domino/datasets/latest-S3/some_data.csv')
```

It’s important to note that the latest-S3 part of the path in the last line of the script is a folder you need to set up as part of your Datasets advanced mode configuration. To set that up, create another new file in your project, and name it *domino.yaml*.

To match the script shown above, its contents should be the following:

```
domino.yaml

datasetConfigurations:
- name: "pipe-in"
  outputs:
    - path: "latest-S3"
      dataset: "fetched-from-S3"
```

That configuration sets up the fetched-from-S3 Dataset created earlier for new input at the latest-S3 path used by the fetch-data.py script. The last step is to set up a scheduled Job that executes this script once per week with the correct Dataset configuration.
Datasets scratch spaces

Overview

A Datasets Scratch Space is a scalable mutable (i.e. read-writeable) filesystem directory for temporary data storage and exploration. They are a complement to the core Datasets functionality. They provide a space to keep intermediate data results or candidates for Dataset Snapshots as you explore your data. These spaces are designed for when you don’t know what you want just yet (i.e. you don’t know what you don’t know). These spaces are automatically mounted for Workspace sessions for each User for every Project (i.e. they are Per User Per Project). Here are some key properties:

1. They do not preserve reproducibility. Files placed in a Datasets Scratch Space are not versioned or tracked. A Datasets Scratch Space is simply a long-lived directory with the scalable properties of Datasets (i.e. large file sizes and many individual files).

2. They are only available for Workspaces.

3. You get a unique Datasets Scratch Space per User per Project.

4. If you shutdown and launch Workspaces, the Datasets Scratch Space is exactly as you left it. All the contents remain; unless you promote the contents to a Dataset Snapshot.

5. If you spin up multiple, concurrent Workspaces in a Project, all those Workspaces will see the same Datasets Scratch Space. Remember, your Scratch Space is private to you, so any file locks present are due to your actions or code.
6. When no Workspaces are running, the contents of the Datasets Scratch Space can be promoted to a Snapshot of a Dataset within the Project.

Finding my Scratch Space

For a given project (with name \textit{project-name}), your Datasets Scratch Space for that Project will be at: 
\texttt{/domino/datasets/\{username\}/\{project-name\}/scratch}, where \textit{username} is your Domino username. Remember, you only get Scratch Spaces with Workspaces.

Examples

Let’s say for these examples, my username is \textit{dara-data}.

1. \textbf{Project}: \textit{big-data}, \textbf{Owner}: \textit{alex-algo}

   My Scratch Space is at:

   \texttt{/domino/datasets/dara-data/big-data/scratch}

   Notice that it doesn’t matter than the owner is \textit{alex-algo}. The scratch path is still with my username. Also, it is assumed I have the appropriate permissions to the Project.

2. \textbf{Project}: \textit{big-data}, \textbf{Owner}: \textit{dara-data}

   My Scratch Space is at:

   \texttt{/domino/datasets/dara-data/big-data/scratch}

   Notice that it doesn’t matter that there are other Projects with the name \textit{big-data}. They won’t conflict because the scope of the path is only for Workspaces in that Project.

Seeing the contents of my Datasets Scratch Space

You can always view the contents of your Datasets Scratch Space by launching a Workspace and navigating to your Scratch Space. It is simply another directory (with all the high performance properties of Datasets).
1. **File Browser.**

If you’d like to get an idea of the contents of the Scratch Space without using a Workspace, you can navigate to the Datasets Project Level Page.

There you will find a file browser that displays the contents of your Scratch Space. If you have lots of files, you can paginate through. You can also drill down into any folders that are in your Scratch Space. As you modify the contents of your Scratch Space (e.g. add/delete/edit files, add/delete/edit folders), refreshing the file browser will reflect those changes.
2. Calculated Size.

The used Scratch Space size is calculated anytime a Workspace is stopped. The timestamp of the calculation is also provided.

In this example, I have two Workspaces open, and my current calculation is 10 GB. This actually doesn’t reflect the 12 GB in the transactions folder that I created since the last time a Workspace was closed.
When I close one of the Workspaces, the used Scratch Space size is updated to 22 GB.
Promoting Scratch Space Contents to a Dataset Snapshot

The contents of your Scratch Space can be made into a Dataset Snapshot.

1. No Workspaces can be running to create a Dataset Snapshot from the contents of your Scratch Space. The Create a Dataset Snapshot from Scratch Space is only enabled when no Workspaces are running.

2. You will only be able to promote to a Dataset within your Project.

3. Once the contents of the Space Scratch is made into a Dataset Snapshot, contents of the Scratch Space will be deleted (i.e. the Scratch Space is cleared)

Workflow

1. Click Create a Dataset Snapshot from Scratch Space

2. Using the input box in the modal, select a Dataset in your Project.

   1. **WARNING**: When you create a Dataset Snapshot from the contents of the Datasets Scratch Space, those contents will no longer be in the Datasets Scratch Space (i.e. The contents of the Datasets Scratch Space will be deleted upon promotion to a Dataset Snapshot).
2. Click Create Snapshot

The contents of the Datasets Scratch Space will be made into a Dataset Snapshot, which can be shared. The contents of the Datasets Scratch Space will be deleted.

Select Dataset

churn

3. To confirm, you can navigate to the details page of your newly created Snapshot.

5.8. Connect to your data
Risk Notifications

Using a Datasets Scratch Space for indefinite storage is discouraged. Not only does this potentially lead to wasteful storage consumption and costs, it may also be unnecessarily compromising reproducibility. Finally, while a Datasets Scratch Space is reliable persistent storage, an accidental loss could occur from a User accidentally deleting contents from the Scratch Space. Creating a Snapshot of valuable work from the Scratch Space prevents accidental loss.

To mitigate the risk of using Scratch Spaces for indefinite storage, a user interface indicator provides a Scratch Space risk notification that alerts the user of the risk associated with the contents in the Scratch Space. Specifically, we want to notify the user of the length of time since potentially new work has not been made into a Snapshot. Here, the length of time is a proxy for risk profile.

By default, the three ranges are:

1. **Low Risk**: Less than or equal to five days.
2. **Medium Risk**: Greater than five days and less than or equal to ten days.
3. **High Risk**: Greater than ten days.

There are three risk ranges, separated by two thresholds; hence, the thresholds define the ranges. The risk ranges are in terms of days and have a lower bound of zero days and no upper bound. The thresholds are configurable via two central configuration values.

**Namespace**: common  
**Key**: com.cerebro.domino.dataset.scratch.riskThresholdOneInDays  
**Value**: number  
**Default**: 5

*This option controls the first Datasets Scratch Space risk threshold in days.*

**Namespace**: common  
**Key**: com.cerebro.domino.dataset.scratch.riskThresholdTwoInDays  
**Value**: number  
**Default**: 10

*This option controls the second Datasets Scratch Space risk threshold in days.*
Administration

The administration page for Dataset Scratch Spaces is the same as the Datasets administration page. Once in the Administration area, you can navigate by selecting **Datasets** under the **Advanced** menu option.

A Dataset Scratch Spaces administration section is below the Datasets administration section. A table of all Dataset Scratch Spaces is shown.

1. **Project**
   
   Unique identifier which is a concatenation of: 

   `{owner_username}/{project_name}`
2. **User**
   Full name of user.

3. **Size**
   Used Scratch Space Size.

4. **Last Time Size Calculated**
   Timestamp of when the value in the Size column was calculated.

5. **Last Time Snapshot**
   The last time the contents of that Scratch Space was promoted into a Dataset Snapshot. No value means the contents of the Scratch Space has never been made into a Dataset Snapshot.

**Filtering**

A filtering input box is provided at the top of the administration table. The filter will do a case insensitive string match on all the columns in the administration table and return a table with rows with matching elements (i.e. rows that contain the filtering string).

**Deleting Scratch Space Contents**

Administrators can delete the contents of a Datasets Scratch Space. Only Scratch Spaces with files are eligible to be deleted; this is the case even if the files are zero bytes. The **Delete Contents** button for Scratch Spaces that contain no files will be disabled.
Workflow

1. Click **Delete Contents** for the Datasets Scratch Space you want to delete the contents of.

2. This will bring up a confirmation modal. If you are sure you want to delete the contents, press **Delete Contents**. Deleting the contents of a Scratch Space cannot be undone.

   **Are you sure?**

   This will delete the contents of the Dataset Scratch space for User Nathan Summers in Project nathan/ltv permanently.

   This Scratch Space has never been used.

   ![Deleting Scratch Space Confirmation]

   Deleting this Scratch Space cannot be undone.

3. The administration table will be refreshed and you can confirm that the contents of your selected Datasets Scratch Space are deleted.

5.8. Connect to your data
Non-Empty Length of Time

The length of time used to notify the user is specifically the time the Scratch Space has been non-empty. Recall the following:

1. The storage size of the Datasets Scratch Space contents is computed after the stopping of any Workspace.
2. There are three cases the Scratch Space becomes empty.
   1. Initial state
   2. User clears it (e.g. `rm -rf *`)
   3. Promote to Snapshot

Consider the following figure, which illustrates the state of a Datasets Scratch Space over time. The horizontal axis is time. Portions where the Datasets Scratch Space is non-empty is shown in orange. A non-empty Scratch Space is one where there exists at least one file (of any size). At times $t_2$ and $t_4$, the Scratch Space is cleared (emptied) by the User and through a promotion to a Snapshot, respectively.
Assume we are at the moment in time marked “Now” and we’ve closed a Workspace, and hence, the Scratch Space storage size is computed. The non-empty time we report will be that moment in time back to $t_5$, the most recent point Scratch Space storage was calculated to be non-empty coming from the empty state. This non-empty time is specified as $T_{\text{NON-EMPTY}}$.

FAQ

1. I tried to write to a Datasets Scratch Space in a Job and it failed?
   A Datasets Scratch Space is only available in Workspace sessions.

2. Why is the stated “Used Scratch Space Size” in the Datasets Scratch Space file browser different from what I expected based on the files that are currently in my Datasets Scratch Space?
   The “Used Scratch Space Size” (size calculation) is not updated in a real-time fashion. Instead, it is calculated every time a Workspace is closed. Notice that with every stated “Used Scratch Space Size”, a note is presented on when the last time the value was calculated. See “Why is the size calculation in the file browser only updated when a Workspace is closed?”.

3. Why is the size calculation in the file browser only updated when a Workspace is closed?
   Constantly having a process periodically calculating a file system size for open Workspaces can be taxing on the system. Remember, the Scratch Space is designed to have all the performance properties of Datasets: it can handle large data (i.e. TBs) and many number of individual files (i.e. millions of files). Also, if Workspaces are already open, Users can inspect the filesystem and sizes directly. Finally, the convenience of the file browser really comes into play when no Workspaces are running, which at that time, everything is static and up-to-date.

4. Why the “Create a Dataset Snapshot from Scratch Space” button disabled sometimes?
   You can only snapshot a Datasets Scratch Space when no Workspaces are running.

5. Why are the contents of the Datasets Scratch Space deleted upon promotion to a Dataset Snapshot?
   This is for performance reasons. Like Datasets, Scratch Spaces can potentially contain large sized files and a large number of individual files. In order to create a Dataset Snapshot and preserve the contents of the Scratch Spaces, we would need to perform an expensive and robust copy operation (i.e. long time and computation resources). By allowing the Scratch Space to be cleared upon promoting its contents to a Datasets Snapshot, we are able to cleverly perform the operation instantly. Both the Scratch Space and newly created Dataset Snapshot are available for use after promotion.

6. I want to promote my Scratch Space to a Dataset that doesn’t exist yet. How do I create a Dataset?
   You can create an empty Dataset as described in the Creating Datasets Support article.

7. What if I want to take the contents of my Dataset Scratch Space and make it into a Snapshot of a Dataset not in my current Project?
   You would have to do this manually and just treat the Scratch Space like a directory you’d like to copy or move into new Snapshot directory.
   First, you must have access to create a new Snapshot to a Project (see Sharing and Collaboration).
Assuming you have permission, you would need to create a configuration in your `domino.yaml` file where you mount an output directory for your desired Dataset. Because the Dataset is not in your Project, you will have to refer to the Dataset in a fully qualified way: `{project_owner_username}/{project_name}/ {dataset_set}`.

So, for example, if I wanted to mount an output directory so that I could create a new Snapshot to a Dataset called `iris` in a Project called `datascience` owned by `john_smith`, I should have a YAML entry that looks something like:

```yaml
datasetConfigurations:
  - name: "new iris snapshot"
    outputs:
      - path: "new_iris"
    dataset: "john_smith/datascience/iris"
```

Here, I called the configuration `new iris snapshot` and I chose a mount point of `new_iris`; both this are up to you.

Once you’ve properly mounted the output directory for your new Snapshot, you can simply launch a Run and copy or move the contents of your Scratch Space to the output directory.

8. **What Dataset Scratch Spaces are shown in the administration table?**

   All Dataset Scratch Spaces are shown in administration table for Dataset Scratch Spaces. A Dataset Scratch Space becomes “active” any time a User starts a Workspace in a Project. Hence, there is a Dataset Scratch Space for any Project that a User starts a Workspace.

9. **Why am I allowed to delete the contents of a Datasets Scratch Space that uses zero bytes?**

   Scratch Spaces with any files are eligible to be deleted; this is the case even if the files are zero bytes.

---

### Converting legacy Data Sets to Domino Datasets

#### Overview

This article describes how to convert legacy Data Sets workflows to use Domino Datasets. This is a two-step process that involves moving your data into a new Domino Dataset, and then updating all projects and artifacts that consume the data to retrieve it from the new location.

#### Migrating data from a legacy Data Set into a Domino Dataset

Legacy Data Sets are semantically similar to Domino Projects. If your deployment is running a version of Domino with the new Domino Datasets feature, you can create Domino Datasets inside legacy Data Sets. This will allow for a very simple migration path for a legacy Data Set, where all of the existing data is added to a single Domino Dataset owned by the legacy Data Set, and the entire file structure is preserved.
The long term deprecation plan for legacy Data Sets is to transform them into ordinary Domino Projects, which will continue to contain and share any Domino Datasets you created in them.

To get started, you need to add a script to the contents of your legacy Data Set that can transfer all of your data into a Domino Dataset output mount. From the **Files** page of your legacy Data Set, click **Add File**:

Name the file `migrate.sh`, and paste in the example command provided below.

```
cp -R $DOMINO_WORKING_DIR/. /domino/datasets/output/main
```

This example migration script copies the contents of `$DOMINO_WORKING_DIR`, which is a default Domino environment variable that always points to the root of your project, to a Domino Dataset output mount path. The directory named `main` in the path below is derived from the name of the Domino Dataset that will be created to store the files from this legacy Data Set.

Click **Save** when finished. Your script should look like this:

```
cp -R $DOMINO_WORKING_DIR/. /domino/datasets/main/output
```

Next, click **Datasets** from the project menu, then click **Create New Dataset**.
Be sure to name this Dataset to match the path to the output mount in the migration script. If you copied the command above and added it to your script without modification, you should name this Dataset `main`. You can supply an optional description, then click **Upload Contents**. On the upload page, click to expand the **Create by Running Script** section.
Create a New Snapshot

- **Create from CLI** - Upload up to 250 MB from your machine with the Domino CLI
- **Create by running script** - Create unlimited-sized, reproducible Snapshots

  **Script to run**

  [Input field]

  **Output directory**

  `/domino/datasets/output/movies`

  [Start button]

- **Create by launching a workspace** - Create Snapshots from an interactive session
- **Append** - Create a new Snapshot based on an existing Snapshot

Double-check to make sure the listed **Output Directory** matches the path from your migration script, then enter the name of your script and click **Start**. A Job will be launched that mounts the new Dataset for output and executes your script. If the Job finishes successfully, you can return to the Datasets page from the project menu and click the name of your new Dataset to see its contents.
You now have all of the data from your legacy Data Set loaded into a Domino Dataset. This method preserves the file structure of the legacy Data Set, which is useful for the next step: updating consumers to use the new Dataset.

### Updating data consumers to use the new Domino Dataset

Potential consumers of your legacy Data Set are those users to whom you granted Project Importer, Results Consumer, or Contributor permissions. As the project Owner, you also may have other projects consuming the contents of your legacy Data Set. This same set of permissions will grant access to your new Domino Dataset.
A project consuming data from your legacy Data Set will import it as a project dependency, and it will be visible on the Other Projects tab of the Files page.

In the example above, the `global-power` project imports the `data-quick-start` legacy Data Set. The contents of `data-quick-start` are then available in `global-power` Runs and Workspaces at the path shown in the Location column. Anywhere your code for batch Runs, scheduled Runs, or Apps refers to that path will need to be updated to point to the new Domino Dataset.

To determine the new path and set up access to the Domino Dataset, you need to mount the Dataset. With the consuming project open, click Datasets from the project menu, then click Mount Shared Dataset. The Dataset to Mount field is a dropdown menu that will show shared Datasets you have access to. In the above example, the main Dataset from the `data-quick-start` project will be mounted at the latest snapshot. Select the Dataset that you migrated your data into earlier, then click Mount.

When finished, you will see the Dataset you added listed under Shared Datasets. The Path column shows the path where the contents of the Dataset will be mounted in this project’s Runs and Workspaces.
Remember that if you used the migration script shown earlier, the file structure at that path will be identical to the file structure of the imported legacy Data Set location. All you need to do to access the same data is change the path to this new Domino Dataset mount.

Be sure to contact other users who are consuming your legacy Data Set and provide them with information about the new Domino Dataset.

### 5.8.2 Data sources overview

Domino is an open platform, and capable of connecting to many data sources. In addition to manually uploading data to Domino’s native file store via the UI or CLI, many users choose to connect directly to data sources from their Python and R code.

In principle, Domino should be able to connect to any data source that has a Python or R package, or Ubuntu driver. Additionally, Domino can access data via APIs, or anything available through a service such as wget.

When configuring a connection to a data source there are three main things to consider:

- **Network connectivity**
  
  To access a data source from Domino, there must be network connectivity from Domino to the source. This can be a LAN connection, or connection over the Internet.

- **Package or driver**
  
  You need to have the appropriate package or driver installed in your environment. There is a large collection of publicly available resources specific to almost any data source, and Domino has authored some guides to common examples.

- **Credentials**
  
  To authenticate to your data source, you will need to store your credentials for the Data source in Domino. Instead of adding them as plain text in your code, we recommend users use environment variables to securely store any usernames or passwords. For those connecting to a data source using Kerberos, users can store their keytab securely by adding it to User Settings > Kerberos Integration.
5.8.3 Connecting to Data Sources

Connecting Domino to DataRobot

This article will walk you through how to connect DataRobot with Domino.

Use the below commands to install the DataRobot Python client in the Dockerfile instructions section of your environment:

```
RUN pip install "datarobot==<version>"
```

To confirm that it installed, run the following commands in your notebook:

```
import datarobot as dr
dr.Client(token='your_token', endpoint='https://app.datarobot.com/api/v2', ssl_verify=False)
dr.Project.list()
```

If everything worked correctly, the last command will return the list of projects you have in DataRobot.

Connecting to BigQuery from Domino

Domino can connect to and query any common database, including Google BigQuery. In this article, we outline the steps to create a Google service account, authenticate to Google, and use the BigQuery API to query a public table.

Obtaining credentials

1. Go to Google’s Service Accounts page. Select a previous project or create a new project.
2. Create a Service account for your project.

3. Define the access that the Service account should have to BigQuery. See Google’s Access Control documentation for more information.
4. Confirm that your Service account has been created.

![Service account created]

New service account **service-big-query** has been created.

5. On the Service Accounts page, create a new key.

<table>
<thead>
<tr>
<th>Service account name</th>
<th>Service account ID</th>
<th>Key ID</th>
<th>Key creation date</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>service-big-query</td>
<td>service-big-query@example-big-query-170823</td>
<td>No keys</td>
<td></td>
<td>![Edit, Delete, Create key]</td>
</tr>
</tbody>
</table>

6. Download the JSON key and keep it in a safe place. You will use this key later to programmatically authenticate to Google.
Enable the BigQuery API

1. Click on the Google APIs logo in the top left of the screen.

2. In the Library page, select the Big Query API.

3. If it is not enabled, click Enable.
Activating your credentials from Domino

Google Cloud activates your credentials using the Google Cloud SDK, which is already installed in the Domino Default environment. You will need to execute the following bash command:

```
/home/ubuntu/google-cloud-sdk/bin/gcloud auth activate-service-account <service account name> --key-file <key file path>
```

Example:

```
/home/ubuntu/google-cloud-sdk/bin/gcloud auth activate-service-account big-query-example@example-big-query-170823.iam.gserviceaccount.com --key-file key.json
```

You may want to use a custom Domino compute environment and enter this command in Domino pre-setup script to activate the credentials before each run. Otherwise, you can execute them in workspace sessions. Make sure to read more on how to store your credentials securely.

Authenticate and query using Python

You will require two Python packages: gcloud and oauth2client==1.4.12. You can install them using

```
pip install --user gcloud oauth2client==1.4.12
```

in either your custom Domino compute environment or in your workspace session.

Use the following code snippet to authenticate your Google credentials and query a public Big Query table:

```python
from oauth2client.client import GoogleCredentials
from googleapiclient.discovery import build

# Grab the application's default credentials from the environment.
credentials = GoogleCredentials.get_application_default()

# Construct the service object for interacting with the BigQuery API.
bigquery_service = build('bigquery', 'v2', credentials=credentials)

query_request = bigquery_service.jobs()
query_data = {
    'query': (
        'SELECT TOP(corpus, 10) as title, ' \\
        'COUNT(*) as unique_words ' \\
        'FROM [publicdata:samples.shakespeare];')
}

query_response = query_request.query(
    projectId='example-big-query-170823',  # Substitute your ProjectId
    body=query_data).execute()
```

(continues on next page)
Connecting to IBM Db2 from Domino

Overview

This article describes how to connect to IBM Db2 from Domino.

Python and ibm_db2

Domino recommends the `ibm_db2` package for interacting with Db2 databases from Python.

Environment setup

Use the Dockerfile instruction below to add `ibm_db` to your environment.

This instruction assumes you already have pip installed.
RUN pip install ibm_db ibm_db_sa

For a basic introduction to modifying Domino environments, watch this tutorial video.

Credential setup
There are several environment variables you should set up to store secure information about your Db2 connection. Set the following as Domino environment variables on your user account:

- `db_username`
- `db_password`

Read Environment variables for secure credential storage to learn more about Domino environment variables.

Usage
Read the `ibm_db` for detailed information on how to use the package. Below is a simple example for connecting to Db2 with `ibm_db` where:

- you have set up environment variables noted above with the `'db_username'` and `db_password`
- you’ve replaced ‘my.host.name’ with the host name for your machine

```python
import ibm_db
import ibm_db_dbi
import pandas as pd

hostname = 'my.host.name'
port = 50001
username = os.environ['db_username']
password = os.environ['db_password']

def query_db(sql):
    ibm_db_conn = ibm_db.connect("DATABASE=IBMPROD;HOSTNAME={};PORT={};PROTOCOL=TCPIP;UID={};PWD={}").format(hostname, port, username, password), ", ", ")
    conn = ibm_db_dbi.Connection(ibm_db_conn)
    df = pd.read_sql_query(sql, conn)
    ibm_db.close(ibm_db_conn)
    return df

sql_cmd = ""
SELECT
    *
FROM
    table
""
```

(continues on next page)
df_cmd = query_db(sql_cmd)
df_cmd

Further reading: IBM database server in Python

R and ibmR

Domino recommends the `ibmR` library for interacting with db2 databases from R.

Environment setup

Use the Dockerfile instruction below to add `ibmR` to your environment.

```
RUN R -e 'install.packages("ibmR")'
```

For a basic introduction to modifying Domino environments, watch this tutorial video.

Usage

Read the `ibmR` documentation for detailed information on how to use the package.

Connecting to Impala from Domino

- **Overview**
- **Using Impala ODBC Connector for Cloudera Enterprise with pyodbc**
Overview

This article describes how to connect to Apache Impala from Domino.

Apache Impala is an open source massively parallel processing SQL query engine for data stored in a computer cluster running Apache Hadoop.

Using Impala ODBC Connector for Cloudera Enterprise with pyodbc

Domino recommends using the Impala ODBC Connector for Cloudera Enterprise in concert with the pyodbc library for interacting with Impala from Python.

Environment setup

1. Visit the Cloudera downloads page to download the Impala ODBC Connector for Cloudera Enterprise to your local machine. For default Domino images of Ubuntu 16.04, you should download the 64-bit Debian package. Keep track of where you save this file, as you will need it in a later step.

2. Create a new public project in your Domino instance to host the driver files for use in Domino environments.
3. In the new project, click **browse for files** and select the driver file you downloaded earlier to queue it for upload. Click **Upload** to add it to the project.
4. After the driver file has been added to your project files, click the gear next to it in the files list, then right click Download and click Copy link address. Save this address somewhere and keep it handy, as you will need when setting up your environment.

5. Add the below Dockerfile instructions below to install the driver and pyodbc in your environment, pasting in the URL you copied earlier where indicated on line 5.

```bash
# download the driver from your project
RUN mkdir /ref_files
RUN \
    cd /ref_files && \
    wget --no-check-certificate [paste-download-url-from-previous-step-here] && \
    gzip -d clouderaimpalaodbc_2.6.0.1000-2_amd64.deb.gz

# install the driver
```

(continues on next page)
RUN gdebi /ref_files/clouderaimpalaodbc_2.6.0.1000-2_amd64.deb --n

# update odbc.ini file for impala driver
RUN \\ echo "\n\[Cloudera ODBC Driver for Impala\] \n Driver=/opt/cloudera/impalaodbc/lib/64/libclouderaimpalaodbc64.so \n KrbFQDN=_HOST \n KrbServiceName=impala \n" >> /etc/odbcinst.ini

# set up impala libraries
RUN export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/cloudera/impalaodbc/lib/64
RUN ldd /opt/cloudera/impalaodbc/lib/64/libclouderaimpalaodbc64.so

# install pyodbc
RUN pip install pyodbc

For a basic introduction to modifying Domino environments, watch this tutorial video.

Credential setup
There are several environment variables you should set up to store secure information about your Impala connection. Set the following as Domino environment variables on your user account:

• IMPALA_HOST
  Hostname where your Impala service is running. Make sure your Impala service and network firewall are configured to accept connections from Domino.

• IMPALA_PORT
  The port your Impala service is configured to accept connections on.

• IMPALA_KERB_HOST
  Hostname of your Kerberos authentication service.

• IMPALA_KERB_REALM
  The name of the Kerberos realm used by the Impala service.

Read Environment variables for secure credential storage to learn more about Domino environment variables.

Usage
Read the pyodbc documentation for detailed information on how to use the package to interact with a database. Below are some example for how to set up a connection.

import pyodbc
import os

(continues on next page)
# fetch values from environment variables
hostname = os.environ['IMPALA_HOST']
service_port = os.environ['IMPALA_PORT']
derby_host = os.environ['IMPALA_KERB_HOST']
derby_realm = os.environ['IMPALA_KERB_REALM']

# create connection object
conn = pyodbc.connect('Host=hostname;'
                     + 'DRIVER={Cloudera ODBC Driver for Impala};'
                     + 'PORT=service_port;'
                     + 'KrbRealm=derby_realm;'
                     + 'KrbFQDN=derby_host;'
                     + 'KrbServiceName=impala;'
                     + 'AUTHMECH=1', autocommit=True)

# if you see:
# 'Error! Filename not specified'
# while querying Impala using the connection object,
# add the following configuration line:
# conn.setencoding(encoding='utf-8', ctype=pyodbc.SQL_CHAR)

# if your Impala uses SSL, add SSL=1 to the connection string
# conn = pyodbc.connect('Host=hostname;'
#   + 'DRIVER={Cloudera ODBC Driver for Impala};'
#   + 'PORT=service_port;'
#   + 'KrbRealm=derby_realm;'
#   + 'KrbFQDN=derby_host;'
#   + 'KrbServiceName=impala;'
#   + 'AUTHMECH=1;'
#   + 'SSL=1;'
#   + 'AllowSelfSignedServerCert=1', autocommit=True)

Connecting to MSSQL from Domino

- **Overview**
- **Python and MSSQL**
- **R and RODBC to MSSQL**
Overview

This article describes how to connect to a Microsoft SQL Server (MSSQL) from Domino.

Python and MSSQL

Domino recommends the pymssql package for interacting with MSSQL databases from Python.

Environment setup

Use the Dockerfile instruction below to install pymssql in your environment.

This instruction assumes you already have pip installed.

```bash
RUN pip install pymssql
```

For a basic introduction to modifying Domino environments, watch this tutorial video.

Credential setup

There are several environment variables you should set up to store secure information about your MSSQL connection. Set the following as Domino environment variables on your user account:

- `DB_SERVER`
- `DB_USERNAME`
- `DB_PASSWORD`

Read Environment variables for secure credential storage to learn more about Domino environment variables.

Usage

Read the pymssql documentation for detailed information on how to use the package. Below is a simple example for connecting to MSSQL with Python where:

- you have set up environment variables noted above
• the server hosts a database named “myData” with a table named “addresses”

```python
from os import getenv
import pymssql

server = getenv("DB_SERVER")
user = getenv("DB_USERNAME")
password = getenv("DB_PASSWORD")

conn = pymssql.connect(server, user, password, "myData")
cursor = conn.cursor()

cursor.execute('SELECT * FROM addresses')
row = cursor.fetchone()

while row:
    print("ID=%d, Name=%s" % (row[0], row[1]))
    row = cursor.fetchone()

conn.close()
```

---

### R and RODBC to MSSQL

Domino recommends the RODBC library for interacting with MSSQL databases from R; however, you may use an alternative package if you’d like.

### Environment setup

Use the Dockerfile instruction below to add the MSSQL drivers to your Ubuntu 16.04 environment.

```bash
RUN curl https://packages.microsoft.com/keys/microsoft.asc | apt-key add -
RUN curl https://packages.microsoft.com/config/ubuntu/16.04/prod.list > /etc/apt/\n    --sources.list.d/mssql-release.list
RUN apt-get update
RUN ACCEPT_EULA=Y apt-get install msodbcsql17
```

For a basic introduction to modifying Domino environments, watch [this tutorial video](#).

### Usage

Read the [RStudio RODBC documentation](#) for detailed information on how to use the package.
Connecting to MySQL from Domino

- Overview
- Python and mysql-connector-python
- R and RMySQL

Overview

This article describes how to connect to MySQL from Domino. MySQL is an open source relational database management system.

Python and mysql-connector-python

Domino recommends the mysql-connector-python library for interacting with MySQL databases from Python.

Environment setup

Use the Dockerfile instruction below to install psycopg2 in your environment. This instruction assumes you already have pip installed.

```bash
RUN pip install mysql-connector-python
```

For a basic introduction to modifying Domino environments, watch this tutorial video.
Credential setup

There are several environment variables you should set up to store secure information about your MySQL connection. Set the following as Domino environment variables on your user account:

- **MYSQL_HOST**
  
  Hostname where your MySQL service is running. Make sure your MySQL service and network firewall are configured to accept connections from Domino.

- **MYSQL_USER**
  
  The MySQL user you want to authenticate as.

- **MYSQL_PASSWORD**
  
  The password for the user chosen above.

Read [Environment variables for secure credential storage](#) to learn more about Domino environment variables.

Usage

Read the [mysql-connector-python documentation](#) for detailed information on how to use the package. Below is a simple example for connecting to MySQL with mysql-connector-python where:

- you have set up environment variables noted above with the host, user, and password
- your user has access to a database named `db1` in the target MySQL instance
- the `db1` database contains a table called `employees`

```python
from mysql.connector import connection
import os

# fetch values from environment variables and set the target database
hostname = os.environ['MYSQL_HOST']
username = os.environ['MYSQL_USER']
password = os.environ['MYSQL_PASSWORD']
dbname = 'db1'

# establish connection to db1 database in your mysql service
cnx = connection.MySQLConnection(user=username,
                                   password=password,
                                   host=hostname,
                                   database=dbname)

# create cursor for passing queries to database
cursor = cnx.cursor()

# define query
query = ('SELECT * FROM employees')

# execute query
cursor.execute(query)

# print results
for row in cursor:
    print(row)
```

(continues on next page)
# close connection
cnx.close()

## R and RMySQL

Domino recommends the RMySQL library for interacting with MySQL services from R.

### Environment setup

Use the Dockerfile instructions below to add RMySQL to your environment.

```sh
RUN sudo apt-get install -y libmariadb-client-lgpl-dev
RUN R -e 'install.packages("RMySQL")'
```

For a basic introduction to modifying Domino environments, watch this tutorial video.

### Credential setup

There are several environment variables you should set up to store secure information about your MySQL connection. Set the following as Domino environment variables on your user account:

- **MYSQL_HOST**
  Hostname where your MySQL service is running. Make sure your MySQL service and network firewall are configured to accept connections from Domino.

- **MYSQL_USER**
  The MySQL user you want to authenticate as.

- **MYSQL_PASSWORD**
  The password for the user chosen above.

Read Environment variables for secure credential storage to learn more about Domino environment variables.
Usage

Read the RMySQL documentation for detailed information on how to use the package. Below is a simple example for connecting to MySQL with RMySQL where:

• you have set up environment variables noted above with the host, user, and password
• your user has access to a database named db1 in the target MySQL instance
• the database contains a table named employees

```r
# load the library
library(RMySQL)

# fetch values from environment variables and set the target database
hostname <- Sys.getenv['MYSQL_HOST']
username <- Sys.getenv['MYSQL_USER']
password <- Sys.getenv['MYSQL_PASSWORD']
database <- 'db1'

# set up a driver and use it to create a connection to your database
con <- dbConnect(RMySQL::MySQL(), host = hostname,
                 user = username, password = password, dbname = database)

# run a query and load the response into a dataframe
df_mysql <- dbGetQuery(con, "SELECT * FROM employees")

# close your connection when finished
dbDisconnect(con)
```

Connecting to Oracle from Domino

• Overview

• Making the Oracle drivers available to Domino

• Python and cx_Oracle

• R and ROracle

Overview

This article describes how to connect to Oracle from Domino.

Oracle Database is a proprietary relational database available as a cloud service or enterprise on-premises solution.
Making the Oracle drivers available to Domino

There are two pieces of client software that must be installed in your environment before you can connect to Oracle:

1. The basic package: instantclient-basic-linux.x64-<oracle-version>dbru.zip
2. The SDK package: instantclient-sdk-linux.x64-<oracle-version>dbru.zip

This software is not hosted by Oracle in a way that permits programmatic installation. You will need to download these files from the Instant Client Downloads page using your Oracle customer login, then host an internal mirror of the files somewhere accessible to your Domino hosts.

In the example environments shown in this article, you will see that these files are retrieved from a private S3 bucket with `wget`. You will need to make them available in a similar manner for your Domino deployment.

Python and cx_Oracle

Domino recommends the `cx_Oracle` library for interacting with Oracle databases from Python.

Environment setup

Use the Dockerfile instruction below to install the Oracle client drivers and `cx_Oracle` in your environment. Note that you cannot copy and paste this Dockerfile directly, as you need to set up your own internal host of the Oracle clients and modify the `wget` step shown here to retrieve them.

```
RUN \
  wget https://s3-us-west-2.amazonaws.com/<s3-bucket-name>/instantclient-basic-linux.x64-12.1.0.2.0.zip \  
    -O /home/ubuntu/instantclient-basic-linux.x64-12.1.0.2.0.zip && \  
  wget https://s3-us-west-2.amazonaws.com/<s3-bucket-name>/instantclient-sdk-linux.x64-12.1.0.2.0.zip \  
    -O /home/ubuntu/instantclient-sdk-linux.x64-12.1.0.2.0.zip && \  
  cd /home/ubuntu && \  
  unzip instantclient-basic-linux.x64-12.1.0.2.0.zip && \  
```

(continues on next page)
For a basic introduction to modifying Domino environments, watch this tutorial video.

## Credential setup

There are several environment variables you should set up to store secure information about your Oracle connection. Set the following as Domino environment variables on your user account:

- **ORACLE_HOST**
  Hostname where your database is running. Make sure your Oracle host and network firewall are configured to accept connections from Domino.

- **ORACLE_SERVICE**
  The service name of the Oracle service running on the target host.

- **ORACLE_USER**
  The Oracle user you want to authenticate as.

- **ORACLE_PASSWORD**
  Password for the user specified above.

Read Environment variables for secure credential storage to learn more about Domino environment variables.
Usage

Read the cx_Oracle documentation for detailed information on how to use the package. Below is a simple example for connecting to Oracle with cx_Oracle where:

- you have set up environment variables noted above with the hostname, service name, username, and password
- your user has access to a database named **houses** in the target Oracle instance

```python
from __future__ import print_function
import cx_Oracle
import os

# fetch values from environment variables and set the target database
hostname = os.environ['ORACLE_HOST']
service = os.environ['ORACLE_SERVICE']
username = os.environ['ORACLE_USER']
password = os.environ['ORACLE_PASSWORD']
connection_string = hostname + '/' + service

# Connect as user "hr" with password "welcome" to the "oraclepdb" service running on this computer.
connection = cx_Oracle.connect(username, password, connection_string)
cursor = connection.cursor()
cursor.execute(""
    SELECT address
    FROM houses
    WHERE zip = 90210"")
for address in cursor:
    print("Address:", address)
```

R and ROracle

Domino recommends the **ROracle** library for interacting with Oracle databases from R.

Environment setup

Use the Dockerfile instruction below to install the Oracle client drivers and RODBC in your environment. Note that you cannot copy and paste this Dockerfile directly, as you need to set up your own internal host of the Oracle clients and modify the wget step shown here to retrieve them.
For a basic introduction to modifying Domino environments, watch this tutorial video.

Credential setup

There are several environment variables you should set up to store secure information about your Oracle connection. Set the following as Domino environment variables on your user account:

- **ORACLE_HOST**
  
  Hostname where your database is running. Make sure your Oracle host and network firewall are configured to accept connections from Domino.

- **ORACLE_SERVICE**
  
  The service name of the Oracle service running on the target host.
• ORACLE_USER
  The Oracle user you want to authenticate as.
• ORACLE_PASSWORD
  Password for the user specified above.

Read Environment variables for secure credential storage to learn more about Domino environment variables.

Usage
Read the ROracle documentation for usage details.

Connecting to PostgreSQL from Domino

• Overview
• Python and psycopg2
• R and RPostgreSQL

Overview

This article describes how to connect to PostgreSQL from Domino.
PostgreSQL is an open-source relational database that can run on a wide variety of local systems and clouds.

Python and psycopg2

Domino recommends the psycopg2 library for interacting with PostgreSQL databases from Python.

Environment setup

Use the Dockerfile instruction below to install psycopg2 in your environment.
This instruction assumes you already have pip installed.
RUN pip install psycopg2

For a basic introduction to modifying Domino environments, watch this tutorial video.

Credential setup

There are several environment variables you should set up to store secure information about your PostgreSQL connection. Set the following as Domino environment variables on your user account:

- **POSTGRES_HOST**
  
  Hostname where your DB is running. Make sure your PostgreSQL DB and network firewall are configured to accept connections from Domino.

- **POSTGRES_USER**
  
  The PostgreSQL user you want to authenticate as.

- **POSTGRES_PASSWORD**
  
  The password for the user chosen above.

Read Environment variables for secure credential storage to learn more about Domino environment variables.

Usage

Read the psycopg2 documentation for detailed information on how to use the package. Below is a simple example for connecting to PostgreSQL with psycopg2 where:

- you have set up environment variables noted above with the hostname, username, and password
- your user has access to a database named `db1` in the target PostgreSQL instance
- the database contains a table named `metrics`

```python
import psycopg2
import os

# fetch values from environment variables and set the target database
hostname = os.environ['POSTGRES_HOST']
username = os.environ['POSTGRES_USER']
password = os.environ['POSTGRES_PASSWORD']
dbname = 'db1'

# set up a connection object with parameters for your database
conn = psycopg2.connect(
    host=hostname,
    port=5432,
    user=username,
    password=password,
    database=dbname,
)
```

(continues on next page)
# create a cursor in your connection
cur = conn.cursor()

# execute a query on the metrics table and store the response
cur.execute("SELECT * FROM metrics;")
results = cur.fetchall()

# display the contents of the response
print(results)

Note that the results object created in the example above is a Python array of entries from the queried table.

---

**R and RPostgreSQL**

Domino recommends the RPostgreSQL library for interacting with PostgreSQL databases from R.

**Environment setup**

Use the Dockerfile instruction below to add RPostgreSQL to your environment.

```
RUN R -e 'install.packages("RPostgreSQL")'
```

For a basic introduction to modifying Domino environments, watch this tutorial video.

**Credential setup**

There are several environment variables you should set up to store secure information about your PostgreSQL connection. Set the following as Domino environment variables on your user account:

- **POSTGRES_HOST**
  Hostname where your DB is running. Make sure your PostgreSQL DB and network firewall are configured to accept connections from Domino.

- **POSTGRES_USER**
  The PostgreSQL user you want to authenticate as.
• POSTGRES_PASSWORD

The password for the user chosen above.

Read Environment variables for secure credential storage to learn more about Domino environment variables.

Usage

Read the RPostgreSQL documentation for detailed information on how to use the package. Below is a simple example for connecting to PostgreSQL with RPostgreSQL where:

• you have set up environment variables noted above with the hostname, username, and password
• your user has access to a database named db1 in the target PostgreSQL instance
• the database contains a table named metrics

```r
# load the library
library(RPostgreSQL)

# fetch values from environment variables and set the target database
hostname <- Sys.getenv('POSTGRES_HOST')
username <- Sys.getenv('POSTGRES_USER')
password <- Sys.getenv('POSTGRES_PASSWORD')
database <- 'db1'

# set up a driver and use it to create a connection to your database
drv <- dbDriver("PostgreSQL")
conn <- dbConnect(
  drv,
  host=hostname,
  port=5432,
  user=username,
  password=password,
  dbname=database )

# run a query and load the response into a dataframe
df_postgres <- dbGetQuery(conn, "SELECT * from metrics;")

# close your connection when finished
dbDisconnect(conn)
```

Connecting to Redshift from Domino

You can configure Domino to query an external data warehouse such as Redshift during a run. This guide shows how to create a connection and retrieve the results of a query using Python as well as R.

Note: if your database is behind a firewall, you may need to speak with your ops/IT/networking team in order to allow the connection from Domino. Please contact us for more information, or to set up a meeting with your team to work out the details.

Credentials

Your database is likely secured with a username and password. We recommend storing these credentials as environment variables in your project so you can access them at runtime without needing to include them in your code.
Python

To establish a connection to Redshift with the psycopg2 library:

```python
import psycopg2
import os

HOST = os.environ['REDSHIFT_HOST']
PORT = 5439  # redshift default
USER = os.environ['REDSHIFT_USER']
PASSWORD = os.environ['REDSHIFT_PASSWD']
DATABASE = 'mydatabase'

def db_connection():
    conn = psycopg2.connect(  
        host=HOST,  
        port=PORT,  
        user=USER,  
        password=PASSWORD,  
        database=DATABASE,
    )
    return conn

example_query = "SELECT * FROM my_table LIMIT 5"

conn = db_connection()
try:
    cursor = conn.cursor()
    cursor.execute(example_query)
    results = cursor.fetchall()  # careful, the results could be huge
    conn.commit()
    print(results)
finally:
    conn.close()

# using pandas
import pandas as pd
conn = db_connection()
try:
    df = pd.read_sql(example_query, conn)
    df.to_csv('results/outfile.csv', index=False)
finally:
    conn.close()
```

R

To establish a connection to Redshift with the RPostgreSQL library:

```r
install.packages("RPostgreSQL")
library(RPostgreSQL)

redshift_host <- Sys.getenv("REDSHIFT_HOST")
redshift_port <- "5439"
redshift_user <- Sys.getenv("REDSHIFT_USER")
redshift_password <- Sys.getenv("REDSHIFT_PASSWORD")
redshift_db <- "mydatabase"

drv <- dbDriver("PostgreSQL")
conn <- dbConnect(  
    host=redshift_host,  
    port=redshift_port,  
    user=redshift_user,  
    password=redshift_password,  
    dbname=redshift_db,  
    driver=drv,
)
```

(continues on next page)
### Connecting to S3 from Domino

- **Overview**
- **Credential Setup**
- **Getting a file from an S3-hosted public path**
- **AWS CLI**
- **Python and boto3**
- **R and aws.s3**

### Overview

This article describes how to connect to Amazon Simple Storage Service (S3) from Domino. S3 is a cloud object store available as a service from AWS.
Credential Setup

There are two main ways to authenticate with S3 from Domino. Both methods follow the common naming convention of environment variables for AWS packages so that you do not need to explicitly reference credentials in your code.

1) Using a short-lived credential file obtained via Domino's AWS Credential Propagation feature.

Once this feature has been configured by your administrator, Domino will automatically populate any run or job with your AWS credentials file. These credentials will be periodically refreshed throughout the duration of the workspace to make sure they don’t expire.

Following common AWS conventions, you will see an environment variable AWS_SHARED_CREDENTIALS_FILE which contains the location to your credential files which will be placed at /var/lib/domino/home/.aws/credentials.

Learn more about using a credential file with the AWS SDK.

2) Storing AWS your access keys securely as environment variables.

In order to connect to the S3 buckets your AWS account has access to, you’ll need to provide your AWS Access Key and AWS Secret Key to the AWS CLI. By default AWS utilities will look for these in your environment variables.

You should set the following as Domino environment variables on your user account:

- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY

Read Environment Variables for Secure Credential Storage to learn more about Domino environment variables.

Getting a file from an S3-hosted public path

If you have files in S3 that are set to allow public read access, you can fetch those files with Wget from the OS shell of a Domino executor, the same way you would for any other resource on the public Internet. The request for those files will look similar to this:

```
wget https://s3-<region>.amazonaws.com/<bucket-name>/<filename>
```

This method is very simple, but doesn’t allow for any authentication or authorization, and should not be used with sensitive data.
AWS CLI

A more secure method of reading S3 from the OS shell of a Domino executor is the AWS CLI. Making the AWS CLI work from your executor is a two-step process. You need to install it in your environment, and provide it with your credentials.

Environment setup

AWS CLI is available as a Python package from pip. The Dockerfile instruction below is what you’ll need to install the CLI and automatically add it to your system PATH. This instruction assumes you already have pip installed.

```
RUN pip install awscli --upgrade
```

For a basic introduction to modifying Domino environments, watch this tutorial video.

Usage

Once your Domino environment and credentials are set up correctly, you can fetch the contents of an S3 bucket to your current directory by running:

```
aws s3 sync s3://<bucket-name> .
```

If you are using an AWS credential file with multiple profiles, you might need to specify the profile. (the “default” profile is assumed if none is specified)

```
aws s3 sync s3://<bucket-name> . --profile <profile name>
```

Read the official AWS CLI documentation on S3 for more commands and options.

Python and boto3

The best available library for interacting with AWS services from Python is boto3, which has been officially supported by Amazon since 2012.
Environment setup

If you’re using one of the Domino standard environments, boto3 will already be installed. If you want to add boto3 to an environment, use the following Dockerfile instruction.

This instruction assumes you already have pip installed.

```
RUN pip install boto3
```

For a basic introduction to modifying Domino environments, watch this tutorial video.

Usage

There are many methods for interacting with S3 from boto3 detailed in the official documentation. Below is a simple example for downloading a file where:

- you have set up your credentials as instructed above
- your account has access to an S3 bucket named `my_bucket`
- the bucket contains an object named `some_data.csv`

```python
import boto3
import io
import pandas as pd

# create new S3 client
client = boto3.client('s3')

# download some_data.csv from my_bucket and write to ./some_data.csv locally
file = client.download_file('my_bucket', 'some_data.csv', './some_data.csv')
```

Alternatively, for users using a credential file.

```python
import boto3

#Specify your profile if you are credential file contains multiple profiles
session = boto3.Session(profile_name='<profile name>')

#Specify your bucket name
users_bucket = session.resource('s3').Bucket('my_bucket')

# 'list' bucket should succeed
for obj in users_bucket.objects.all():
    print(obj.key)

#download a file
users_bucket.download_file('some_data.csv', './some_data.csv')
```

Note that this code does not provide credentials as arguments to the client constructor, since it assumes either:

- credentials will be automatically populated at `/var/lib/domino/home/.aws/credentials` as specified in the environment variable `AWS_SHARED_CREDENTIALS_FILE`
- you have already set up credentials in the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.
After running the above code, you would expect a local copy of `some_data.csv` to now exist in the same directory as your Python script or notebook. You could follow this up by loading the data into a pandas dataframe.

```python
df = pd.read_csv('some_data.csv')
```

Check out part 1 of the Get Started (Python) tutorial for a more detailed example of working with CSV data in Python.

---

**R and aws.s3**

The cloudyr project offers a package called `aws.s3` for interacting with S3 from R.

**Environment setup**

If you’re using one of the Domino standard environments, aws.s3 will already be installed. If you want to add aws.s3 to an environment, use the following Dockerfile instructions.

```
RUN R -e 'install.packages(c("httr","xml2"), repos="https://cran.r-project.org")'
RUN R -e 'install.packages("aws.s3", repos = c("cloudyr" = "http://cloudyr.github.io/drat"))'
```

For a basic introduction to modifying Domino environments, watch this [tutorial video](#).

**Usage**

You can find basic instructions on using aws.s3 from the package README. Below is a simple example for downloading a file where:

- you have set up the correct environment variables with credentials for your AWS account
- your account has access to an S3 bucket named `my_bucket`
- the bucket contains an object named `some_data.csv`

```r
# load the package
library("aws.s3")

# If you are using a credential file and that file has multiple profiles. Otherwise, this can be excluded.
Sys.setenv("AWS_PROFILE" = "<AWS profile>")
```

(continues on next page)
### Connect to Snowflake from Domino

- **Overview**
- **Connecting to Snowflake with Python**

### Overview

Snowflake is a cloud-based data-warehouse. This article describes how to connect to Snowflake from Domino. In order to connect successfully, you must have network connectivity between Snowflake and your Domino deployment.

### Connecting to Snowflake with Python

Domino recommends the Snowflake Python connector (`snowflake-connector-python`).

**Environment setup**

Use the Dockerfile instruction below to install `snowflake-connector-python` and its dependencies in your environment.
RUN apt-get install -y libssl-dev libffi-dev \ 
    pip install -U pip && pip install --upgrade snowflake-connector-python

If you encounter an error due to your Ubuntu version, use the following Dockerfile instruction:

RUN pip install -U pip && pip install --upgrade snowflake-connector-python

For a basic introduction to modifying Domino environments, watch this tutorial video.

Credential setup

There are several environment variables you should set up to store secure information about your Snowflake connection. Set the following as Domino environment variables on your user account:

- SNOWFLAKE_USER
- SNOWFLAKE_PASSWORD
- SNOWFLAKE_ACCOUNT

Read Environment variables for secure credential storage to learn more about Domino environment variables.

Usage

Read the Snowflake python connector documentation for detailed information on how to use the package. Below is a simple example.

```python
import snowflake.connector
import os

# Gets the version
ctx = snowflake.connector.connect(  
    user=os.environ['SNOWFLAKE_USER'],
    password=os.environ['SNOWFLAKE_PASSWORD'],
    account=os.environ['SNOWFLAKE_ACCOUNT']
)

cs = ctx.cursor()

try:
    cs.execute("SELECT current_version()")
    one_row = cs.fetchone()
    print(one_row[0])
finally:
    cs.close()
ctx.close()
```

Alternatively, you can use generic Python JDBC or ODBC tools to connect to Snowflake. However, they are not specialized for use with Snowflake. They may have inferior performance and will require more time to set up.

For more information on JDBC / ODBC connections, read:

https://docs.snowflake.net/manuals/user-guide/jdbc.html
Connecting to Okera from Domino

Overview

This article describes how to connect to the Okera Active Data Access Platform from Domino. The Okera Active Data Access Platform unifies and manages access for data consumers by providing an Automated Schema Registry, Policy Engine and Audit Engine. Domino and Okera make it easy to access securely and simply data governed by Okera for analysis in Domino.

Credential Setup

Option A - JWT Token

The simplest method for authentication with Okera from Domino is to use the JWT token provided by Domino. Domino automatically populates all Workspaces and Batch Jobs with a user specific, securely signed token that can be used for authentication. When used in conjunction with Single Sign On (SSO), you can achieve a chain of identity from you identity provider through Domino to Okera. No additional set up in Domino is required.

Warning: We do not recommend using this authentication method if are not using a Single Sign On (SSO) and allow end users to create their own accounts.

Since all Domino issued tokens are securely signed by Domino, Okera needs to be configured to validate the JWT signature using Domino’s public key.

For me detailed instructions on Okera configuration see the Okera docs here.

Required metadata from Domino:

- **JWT_PUBLIC_KEY** for your Domino deployment can be found at `<your domino url>/auth/realms/DominoRealm`
- **JWT_ALGORITHM** will be RS256 by default
- **JWT_USER_CLAIM_KEY** allows you to bind users identity in Domino to their identity in Okera so a user can only authenticate at themselves. Typically, you will want to use the `preferred_username` from the Domino token if you’ve used the same username pattern in Domino and Okera. Alternatively, you can create a custom claims in your Domino token by mapping in additional SAML attributes or deriving a value from the user’s email address, for example.
Option B - Username & Password

Alternatively, Okera also allows for authentication via a username and password. This requires an extra set up step on the part of each end user. Similar to storing credential for other data sources, users can add their credentials as Domino user environment variables. These variables are then populated to that user’s workspace and batch jobs.

Read [Environment Variables for Secure Credential Storage](#) to learn more about Domino environment variables.

Environment Libraries Setup

Installation

Okera’s [Pyokera](#) package is available from [pip](#). The Dockerfile instruction below is what you’ll need to install Pyokera.

```bash
RUN pip install pyokera
```

For a basic introduction to modifying Domino environments, watch [this tutorial video](#).

Alternative Installation

A more secure method of reading S3 from the OS shell of a Domino executor is the [AWS CLI](#). Making the AWS CLI work from your executor is a two-step process. You need to install it in your environment, and provide it with your credentials.

Usage (Accessing Data in Python)

Once Pyokera is installed in your compute environment, you can access your Okera data using the following code snippet by adding in the URI for your Okera instance:

```python
#Load the Pyokera package
import os
from okera import context

#Load the Domino Auth Token which should always be available and refreshed at $DOMINO_TOKEN_FILE
def get_token():
    return open(os.environ.get('DOMINO_TOKEN_FILE'), 'r').read().strip()

#Connect to Okera using your token and using the environment variable $DOMINO_STARTING_USERNAME which be populated with your Domino username
ctx = context()
ctx.enable_token_auth(token_func=get_token, user=os.environ['DOMINO_STARTING_USERNAME'])

#Query Okera
with ctx.connect(host='<URI for your Okera Instance>', port=12050) as conn:
    display(conn.scan_as_pandas('select * from okera_sample.whoami', strings_as=utf8=True))
```

Learn more about using [Pyokera](#)

---

5.8. Connect to your data 541
5.8.4 External Data Volumes

Attention: To use external data volumes in your Domino project, a Domino administrator must first register and configure the volumes in your Domino deployment. Please reach out to your Domino administrator to learn more about registered external volumes and your corresponding access privileges.

If you’re a Domino administrator and would like to register or configure external volumes for a Domino deployment, please visit the corresponding external volume documentation in the Domino Administrator’s Guide.

Overview

Domino projects can access data stored in external storage volumes, like Network File Systems (NFS). When configured for use with your Domino deployment, external data volumes are automatically mounted to supported Domino executions. At the moment, external volumes are supported in the following Domino executions:

- Jobs (including Scheduled Jobs)
- Workspaces
- Apps
- Launchers
- On-demand Spark clusters

NFS is currently the only supported volume type in Domino, but there are plans to support more volume types in future versions of Domino. Please contact your local Domino Administrator or e-mail support@dominodatalab.com for more information.

Mount an external volume

1. Click on Data in the Domino sidebar menu.
2. Scroll to the “External Data Volumes” section.

3. Click the Add External Volume button.
4. In the modal that appears, select an available volume. Click into the text area to expand a dropdown menu of available volumes, or type into the text area to quickly search for a volume.

Attention: If you're unable to view, search for, or select external volumes, they are either not registered in your deployment or you may not have the proper access privileges to the volumes. Please contact your Domino administrator for assistance.

5. Click the Add button. If your volume is successfully mounted, it’ll be listed in a table in the “External Data Volumes” section of Data.
View mounted volumes

1. Click on **Data** in the Domino sidebar menu.
2. Scroll to the “External Data Volumes” section. If no volumes have been mounted, you’ll be prompted to mount a volume. If the project already has mounted volumes, they’ll appear here listed in a table along with the volume’s properties.

**Attention:** Unclickable or “greyed out” volumes listed in the table indicate that you do not have the privileges to use these volumes. If you need access to these volumes, please contact your Domino administrator. To learn more, please refer to the Censored volumes section.
Partial volume censorship

You may encounter mounted volumes that are “greyed out” in your volume table. This means that the volume(s) have been mounted to your project, but that you do not have access to them. To gain access to the volume(s), please contact your Domino administrator.

Full volume censorship

Your project may contain volumes that have been mounted to the project but are not listed (i.e., fully censored from view) in the volume table. A notification banner will appear above the table informing you about this. To gain access to the volumes, please contact your Domino administrator.
Properties of mounted volumes

Mounted volumes will appear in a table in the Data section of your project. The table will also display the following properties of the volume:

- **Name** – An alias for the volume. To change this setting, contact your Domino administrator.
- **Type** – The type of volume. At the moment, Domino only supports NFS volumes.
- **Description** – A description of the volume, set by your Domino administrator. To change the description, contact your Domino administrator.
- **Mount Path** – The mount path of the volume: `/domino/edv/name-of-volume`. Use this mount path when using the volume in a Job, Workspace, or other supported Domino execution.
Use a mounted volume

By default, external volumes that are mounted to your project (and that you have access to) are also automatically mounted in supported executions. At the moment, supported Domino executions include Jobs (including Scheduled Jobs), Workspaces, Apps, Launchers, and on-demand Spark clusters. You can access a volume within an execution by referencing the mount path of the volume(s).

Mounted volumes in a Job

Mounted volumes in a Workspace
Mounted volumes in an App

Mounted volumes with a Launcher

**Attention:** Launcher users must have sufficient access privileges to any external volumes used by the Launcher. Otherwise, the volume(s) will not mount and the Launcher may not function properly. Please contact your Domino administrator for assistance with granting access privileges to users.

Mounted volumes with an on-demand Spark cluster

External data volumes are mounted at the full mount path on the driver and all Spark executors.
Unmount a volume

1. Click on **Data** in the Domino sidebar menu.
2. Click on the three vertical dots to the right of the corresponding entry in the table and then click on Remove in the menu that appears.

3. Confirm (or cancel) removal in the subsequent modal that appears.
Current Limitations

- Model APIs do not support external volumes.
- External data volume actions are not exposed by the Domino REST API or Domino CLI.

5.8.5 Git and Domino

Git repositories in Domino

- Overview
- Step 1: Create credentials
  - Option 1: SSH key creation
  - Option 2: Personal Access Token creation
  - Option 3: Username and password
- Step 2: Add your credential to Domino
  - Option 1: SSH private key
  - Option 2: Personal Access Token
- Step 3: Add a repository to a project
Overview

Domino supports importing Git repositories to projects. Repositories that have been added to a project are available to Runs started in that project, allowing you to access the contents of those repositories just as you would your Domino files. This article explains how you can import a Git repository to a project, access the added repository from within a Workspace, and commit any changes back to the repository.

Domino supports connecting to Git servers via HTTPS and SSH for both public and private repositories.

Note: If you want full Git experience for your code by using Git and a Git service provider of your choice, we recommend you set up Git-based projects with CodeSync. Check out this article to learn more.

Step 1: Create credentials

If you are adding a private repository, want to write commits to remote, or are using SSH, you will need to add Git credentials to your Domino account. Domino will use these credentials to authenticate with the service hosting your repository when you start a Run.

Domino supports storing three types of credentials:

- Personal Access Tokens
- SSH private keys
- Username and password
Option 1: SSH key creation

To connect with SSH, you’ll need a private SSH key that corresponds to a public key that you’ve added to your Git service. Check out the GitHub documentation for thorough instructions on creating and adding keys.

Option 2: Personal Access Token creation

You will need a Personal Access Token to access a private repository via HTTPS. You will need a Personal Access Token if the URI you want to use to interact with a repository is formatted as:

https://<domain>/<user>/<repository>.git

Personal Access Tokens are supported by the following Git services.

- GitHub Personal Access Tokens
- GitLab Personal Access Tokens

To connect to Bitbucket repositories via HTTPS from Domino, you must add a Bitbucket App Password credential to your Domino account.

If your GitHub organization requires SSO then you will need to authorize the PAT or SSH key in order to access private repositories via Domino.

Read the GitHub documentation for instructions on authorizing keys for SSO on Github.

Option 3: Username and password

For other Git service providers, including Amazon AWS CodeCommit, Azure DevOps, we also support authentication using App username and passwords.

Step 2: Add your credential to Domino

Option 1: SSH private key

You will need an SSH Private Key to access a repository via SSH. You will need an SSH private key if the URI you want to use to interact with a repository is formatted as:

$user>@<domain>:<username>/<repository>.git

SSH access is supported by the following Git services.
After setting up SSH access with your Git service, you should have both a public key that you provided to the Git service, and a private key. Use these steps to add the private key to Domino:

1. Copy your SSH key
   
   From a terminal, `pbcopy < ~/.ssh/id_rsa` is the easiest way to copy the key to your clipboard assuming you set up your key with the standard name and in the standard location.
   
   If you copy the key manually, remember to include the `-----BEGIN OPENSSH PRIVATE KEY-----` and `-----END OPENSSH PRIVATE KEY-----` header and footer.

2. In Domino, click your username at the bottom of the main menu, then click Account Settings.

3. Scroll down to the panel labeled Git Credentials, then click Add a New Credential.

4. In the Domain field, enter the exact domain of the service hosting your repository, such as `github.com`, `bitbucket.com`, or `your-internal-gitlab-url.com`.

5. For Authentication Credential Type, click to select Private SSH Key.

6. Paste in your private key. This will be the contents of the private key file that matches the public key you provided to your Git service.

7. If you set up your SSH keys to require a passphrase when used, enter it in the Passphrase field, then click Add Credential.

8. You should now see your credential listed in the Git Credentials panel. You can also delete it from this panel if desired.
Option 2: Personal Access Token

The Personal Access Token you generate needs to have read and write access to your private repositories. After generating a Personal Access Token in your Git service, use these steps to add it to Domino:

1. In Domino, click your username in the upper right, then click Account Settings.
2. Scroll down to the panel labeled Git Credentials, then click Add a New Credential.
3. In the Domain field, enter the exact domain of the service hosting your repository, such as github.com, bitbucket.com, or your-internal-gitlab-url.com.
4. For Authentication Credential Type, click to select Personal Access Token.
5. Enter your Personal Access Token, then click Add Credential.
6. You should now see your credential listed in the Git Credentials panel. You can also delete it from this panel if desired.

Step 3: Add a repository to a project

1. Open the project you want to add a repository to, then click Files from the left navigation bar.
2. Click to open the **Git Repositories** tab, then click **Add a New Repository**.

3. Enter an optional directory name and the HTTPS or SSH URI of the repository you want to add. The directory name will be the directory in `/repos` that this repository clones into. It defaults to the name of the repository.

4. Use the dropdown menu to choose which branch of the repository you want Domino to check out when it clones this repository into a run or workspace. If you leave this setting at **Use default branch**, Domino will check out the branch specified as default by your Git service, typically **master**. You can also specify a different branch name, tag name, commit ID, or supply a custom Git ref.
Add a New Repository

**URI**
(https or ssh only)

https://github.com/MayHuDomino/dominotest.git

**Git Credentials**

pat (https)

**Git Service Provider**

Github

**Git Reference**

Branches: master

**Directory Name**
(optional, defaults to repository name)

---

5. Click **Add Repository**.
Working with a Git repository in Domino

When you start a run or workspace in a project, any repositories added to the project are cloned into /repos and will have the branch or commit you specified checked out.

Remember that your Domino working directory is in /mnt, which is a sibling of /repos. Both directories are in the filesystem root (/). Scripts you have added as Domino files can interact with the contents of these repositories by specifying an absolute path to /repos/<repo-name>/<file>.

Committing back to Git Repositories

When you start a Workspace session in a project that has added Git repositories, you will see those repositories listed in the Session Overview under Git repos. If you make changes to the contents of those repositories while running the workspace, those changes will be itemized file-by-file under each repository.

If you want to commit those changes back to the repository, click the checkbox next to the repository name and then click Full Sync.
You will be prompted to supply a commit message. This commit message will be attached to commits to the selected Git repositories, and to a new revision of the Domino project if there are changes to Domino files. Git commits will
be pushed to the default branch you specified when adding the repository.

**Add Commit Message**

**Repositories that will be committed:**
- Domino Files
- deployer-docs

**Commit message:**

```
my commit message
```

If you attempt to stop your workspace while there are uncommitted changes to your Git repositories, you will be prompted to commit those changes. This works the same as the Session Overview interface. Click the checkbox next to the repositories you want to commit to, supply a commit message, and click **Stop and Commit.**
If you try to commit when there are conflicts between your local changes and the state of the default branch in remote, Domino will create a new branch from its local state. Domino will then push that new branch to remote.

After this happens, you will need to resolve those conflicts outside of Domino, or use the command line in your Workspace session to resolve them. The next time you launch a Workspace session, Domino will check out the default branch from remote, not the new branch it pushed.

**Git interaction from the workspace command line**

Both Jupyter and RStudio workspaces have command line tools. You can use these to interact with your repositories with conventional Git commands. Navigate to `/repos` in your command line to find your project’s repositories. Visit
the official Git documentation to learn more about using Git on the command line.

To open the RStudio command line, click Tools -> Shell...

To open the Jupyter command line, from the Files tab click New -> Terminal
Tracking changes to repositories made in Domino Runs

When viewing the Details tab of a Domino Run, at the bottom you will find a Repositories panel. You can expand this panel to see details of how the repository changed during the Run. Domino records the checked out commit at the start of the Run and the end of the Run.

Troubleshooting

Run Error:

Errors occurred while processing dependencies. Please contact support@dominodatalab.com:
Credentials are required for your repository: project-name (ssh://git@github.com/your-org/projectname.git)
Solution:
Your Git Credential added to Domino may have the incorrect Domain. Double-check the domain field in your Git credential to ensure it matches your exact Git repository URL, like:

- github.com
- bitbucket.com
- your-internal-gitlab-url.com

Run Error:

```
Errors occurred while processing dependencies. Please contact support@dominodatalab.com:

Authentication is required for your repository:
The repository provided requires credentials but none were found.
Please add SSH or PAT authentication to your Domino account.
```

Solution:
There are a couple steps to check when encountering this error. First, ensure your private SSH key or PAT has been added to the Git Credentials section of your Domino Account Settings page. Second, if your organization’s Git repository requires SSO access, you may need to authorize the key you have added. Take a look at the following instructions on authorizing keys for SSO on Github.

Run Error:

```
Errors occurred while processing dependencies. Please contact support@dominodatalab.com:

remote: Invalid username or password.
fatal: Authentication failed for 'https://github.com/<your account>/<your repo>'/
```

Solution:
If your organization’s Git repository requires SSO access, you may need to authorize the key you have added. Take a look at the following instructions on authorizing keys for SSO on Github.

Working from a commit ID in Git

When you add an external Git repository to your project, you have the option to specify which state of the repository you want checked out by default in your runs and workspaces. You can specify a branch, tag, commit ID, or custom ref.

Commit IDs are unique SHA-1 hashes that are created whenever a new commit is recorded. If you specify a commit ID when adding a repository, Domino will always pull the state of the repository specified by that commit in a detached HEAD state. This represents a fixed point in the repository’s history, and will not evolve over time like a branch.
Domino cannot automatically push to repositories it has pulled in this way. If you want to push changes to such a repository, you can use the workspace command line to manually commit and push to a new branch. Read Git repositories in Domino to learn more about interacting with Git in workspaces.

**Example**

In this example, a repository called `domino-manual` has been added to our project with a specified commit ID. When we start our workspace, it gets pulled to `/repos/domino-manual` with the target commit checked out in a detached HEAD state. We can verify this in our workspace command line by running `git status`.

```
ubuntu@domino-run-5b0497d0be4b0dfb3685c9140:~/repos/domino-manual$ git status
HEAD detached at deaef05
nothing to commit, working directory clean
ubuntu@domino-run-5b0497d0be4b0dfb3685c9140:~/repos/domino-manual$
```

Suppose during the course of our workspace session we make a change to the repository. It will remain in the detached HEAD state, but Git will continue to track changes.
We can add and commit those changes as normal.

However, if we try to push from the detached HEAD state, we will encounter a fatal error. Git must have a branch to push to.

The solution is to create a local branch from our detached HEAD, check it out, and push to remote with `git push -u origin <branch-name>`. 
5.8.6 Working with big data in Domino

Introduction

When you start a Run, Domino copies your project files to the executor that is hosting the run. After every run in Domino, by default, the Domino will try to write all files in the working directory back to the project as a new revision. When working with large volumes of data, this presents two potential problems:

1. The number of files that are written back to the project may exceed the configurable limit. By default, the file limit for Domino project files is 10,000 files.

2. The time required for the write process is proportional to the size of the data. It can take a long time if the size of the data is very large.

There are three solutions to consider for these problems, discussed in detail below. The following table shows the recommended solution for various cases.

<table>
<thead>
<tr>
<th>Case</th>
<th>Data size</th>
<th># of files</th>
<th>Static / Dynamic</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large volume of static data</td>
<td>Unlimited</td>
<td>Unlimited</td>
<td>Static</td>
<td>Domino Datasets</td>
</tr>
<tr>
<td>Large volume of dynamic data</td>
<td>Up to 300GB</td>
<td>Unlimited</td>
<td>Dynamic</td>
<td>Project Data Compression</td>
</tr>
</tbody>
</table>

Domino Datasets

When working on image recognition or image classification deep learning projects, it is common to require a training dataset of thousands of images. The total dataset size can easily become tens of GB. For these types of projects, the initial training also uses a static dataset. The data is not constantly being changed or appended to. Furthermore, the actual data that is used is normally processed into a single large tensor.

You should store your processed data in a Domino Dataset. Datasets can be mounted by other Domino projects, where they are attached as a read-only network filesystem to that project’s Runs and Workspaces.
For more information on Datasets, see the Datasets overview.

---

**Project Data Compression**

There may be times when you want to work with logs as raw text files. When that is the case, typically new log files are constantly being added to the dataset, so your dataset is dynamic. Here, we can encounter both potential problems detailed in the introduction simultaneously:

1. number of files being over the 10k limit
2. long preparing and syncing times.

We currently recommend storing your large number of files in a compressed format. If you need the files to be in an uncompressed format during your Run, you can use Domino Compute Environments to prepare the files. In the pre-run script, you can uncompress your files:

```
tar -xvzf many_files_compressed.tar.gz
```

Then in the post-run script, you can re-compress the directory:

```
tar -cvzf many_files_compressed.tar.gz /path/to/directory-or-file
```

If your compressed file is still very large, the preparing and syncing times may still be long, depending on how large your compressed file is. Consider storing these files in a Domino Dataset to minimize the copying times.

### 5.8.7 Working with lots of files

How to work around problems that arise when your project has thousands (or more) of files.

**Too many open files**

**The problem:** Domino gives you an error like, “too many open files”

**The solution:**
Create a file in your project folder called `.noLock` (note that the leading ‘.’ is critical, and that capitalization matters)

**Details (if you’re interested):**
When Domino synchronizes your files with the server, it takes out a lock on all your project files, to ensure they don’t change during the synchronization process. If you have a very large number of files, Domino will run into the operating system’s limit for the number of files that can be “open” at once. The `.noLock` file tells Domino to skip its normal step of locking your files during synchronization — so you should be careful not to change any of your project files while Domino is uploading or downloading them.
5.8.8 Basics of moving data over a network

Overview

When you start run or workspace in Domino, the software and filesystem context for your code is defined by two things:

- a Domino environment defines the container your run executes in
- your project files are mounted at /mnt in the container

Both of these are stored within Domino itself. Domino maintains a versioned repository of your project files, and caches the latest image built from your environment.

There are several circumstances where you may want to retrieve data from a source outside of Domino:

- when executing code stored in your project files, you may want to retrieve fresh data from an external source for analysis
- when building a new revision of your environment, you may want to retrieve and install new dependencies or different versions of existing dependencies
- when running a Domino workspace, you may want to retrieve either dependencies or fresh data to advance your experimentation

In this article we’ll introduce some standard tools for moving data from one filesystem to another. Note that all of these require that you have network access to the computer you’re trying to get data from. This can mean accessing a machine over your corporate LAN, or the Internet.

Domino executors run on Linux. All of the tools and examples in this article are presented for use on a Domino-supported Linux operating system like Ubuntu or RHEL. However, these tools will work in any GNU Bash shell, including the macOS terminal.

These methods are suited to retrieving specific files that are hosted at a URL or stored on a filesystem. If you have a relational database or other data source that doesn’t serve simple files, you should check our how-to guides on data source connections.

Wget

Wget is a built-in utility for GNU operating systems that can download files from network locations over HTTP, HTTPS, and FTP. Files that you want to retrieve with Wget must be served over one of those protocols at a URL your machine has access to.

Wget is extremely simple to use. Commands take the form:

```
wget PROTOCOL://URL
```

If you need to supply the target web server with a basic username and password for authentication, you can use the `–user` and `–password` flags. Here’s a complete example:
wget --user myUsername --password myPassword HTTPS://web.server.url/path/to/file.csv

Many cloud object stores like Amazon S3 and Azure Blob Storage can be configured to serve files at a URL over the Internet. See the first part of the Get Started (Python) tutorial for an example of retrieving data from S3 with Wget. You can also host files on computers in your local network with web servers like Apache or SimpleHTTPServer. However, Wget is more limited than curl in terms of supported protocols and authentication schemes.

curl

curl is a tool for making web requests over a wide variety of protocols and with support for many authentication and encryption schemes. curl can be used to query a web server for a standard HTTP response like you would get from Wget, but it can also be used to construct more complex queries for REST APIs.

curl requests can become quite complex when passing in many headers or setting many options, but the basic format is similar to Wget:

curl "PROTOCOL://URL"

For example, you can use curl to query the Domino API itself for data about your Domino deployment. Here’s a complete example:

curl --include \
-H "X-Domino-Api-Key: <your-api-key>" \
'https://<your-domino-url>/v4/gateway/runs/getByBatchId'

You can also use curl to download a file from s3 by using the below code. The assumption here is that your s3 bucket resides in us-west-2 region, but you can change that in the url to make sure it reflects the right region in which your s3 bucket is located.

#!/bin/sh
file="<your-file-name>" 
bucket="<your-bucket-name>" 
resource="/"$bucket"/"$file" 
contentType="<content-type>" 
dateValue=""date +"%a, %d %b %Y %H:%M:%S %z''" 
stringToSign="GET\n\n$contentType\n$dateValue\n$resource" 
s3KeyId="$AWS_ACCESS_KEY_ID" 
s3Secret="$AWS_SECRET_ACCESS_KEY" 
signature="echo -en ₹(stringToSign) | openssl sha1 -hmac ₹3Secret) -binary | base64" 
curl -H "Host: ₹bucket).s3-us-west-2.amazonaws.com" 
-H "Date: ₹$dateValue)" 
-H "Content-Type: ₹contentType)" 
-H "Authorization: AWS ₹s3KeyId:₹signature)" 
https://₹bucket).s3-us-west-2.amazonaws.com/₹file"
5.9 Advanced user configuration settings

5.9.1 User API keys

To interact with secured endpoints of the Domino API, you must send an API authentication key along with your request. These keys identify you as a specific Domino user, and allow Domino to check for authorization.

To find your API key, first open your Account Settings.

In the setting menu, click API Key. You will see a panel that displays your key. Refer to the API documentation for information about how to use this key.

Be advised that any person bearing this key can authenticate to the Domino application as the user it represents and take any actions that user is authorized to perform. Treat it like a sensitive password.

5.9.2 Domino Token

Domino uses the OpenID Connect (OIDC) protocol to generate user-specific JWT tokens, or “Domino Tokens”. This Domino token is signed by Domino and made available in your Workspace, Job, or App. As this token is securely signed by Domino, it can be used to authenticate to third party resources or data sources. Additionally, this token can be used to authenticate with the Domino API.

As the token is short lived and automatically refreshed by Domino, it can be used in scenarios where you would to limit a user’s access to, for example, a data source exclusively from Domino which has security advantages over provisioning them long lived keys that can be used anywhere or by anyone.
Domino Documentation, Release 4.4

To disable this feature, your admin must add the central configuration value `com.cerebro.domino.auth.refreshTokenInRun.enabled` and set it to `false`. Services must be restarted for central configuration changes to take effect. This should not require downtime.

**Usage**

All workspaces, runs and apps will have a Domino Token for the user that can be found at path `/var/lib/domino/home/.api/token` which is indicated by environment variable `DOMINO_TOKEN_FILE`. By default the Domino Token has an expiration of 5 minutes, but will be automatically refreshed throughout your workspace or job.

For example, from a workspace, you can authenticate to one of the Domino APIs with the following:

```
TOKEN=`cat $DOMINO_TOKEN_FILE`
```

**Configuring a service to trust Domino’s JWT**

Each system will have different steps for configuration but will likely require you to enter Domino’s JWKS endpoint which is `<Domino URL>/auth/realms/DominoRealm/protocol/openid-connect/certs>`. For offline services, retrieve and record the JWKS data to use offline with your library. Alternatively, you can use the public key or certificate which can be found at `<Domino URL>/auth/realms/DominoRealm`.

Learn more about JWKS

**Token Claims**

By default, the Domino Token contains standard claims like name, username and email, but additional custom claims can be included by adding additional mappers in the domino-play client in Keycloak. This may be useful if you intend to set up bounded claims in the system you’re integrating to. For example, you might want to only authorize users with a specific organization claim or any other SAML assertion that flows into Domino from your Identity Provider.

**Additional Resources**

JWT.io - A useful resource for decoding and inspecting the content of a JWT token

**5.9.3 Organizations overview**

Domino lets you create “organizations” with groups of users, so you can permission your project to many users at once and more easily add/remove collaborators from multiple projects.

To create an Organization, navigate to your Account Settings page (using the button with your username in the lower left), and click the “New Organization” button:
An Organization is associated with one specific user account who owns and manages the organization (the “Admin”). If you are logged in under this account, you can manage organization membership on your Account page <Domino URL>/account>. From there, you can add/remove users from your Organization.

Permissions of Organization Members

All members of your Organization have “owner-level” access to any projects under the Organization’s account. To make that concrete…

Let’s say you have an Organization account with username your_org; it has members nick, chris, and matthew. That means nick, chris, and matthew will all have owner-level access to any your_org’s projects, e.g., your_org/quick-start, your_org/project1.

They cannot, however, change the Organization’s membership – only you can do that.

Transferring projects to an Organization

Organization members can transfer ownership of a project to the Organization by clicking Settings from the project menu, then on the Archive Project or Transfer Ownership tab, clicking Change Project Owner. Enter the current owner and project name, then for the New owner username use the name of the Organization.
API endpoints will be unpublished, need to be manually re-published and clients must be updated with the updated URL and the new owner's API key. The environment of the project will be set to the default environment. All currently executing runs will be stopped. You will lose access to the project until you are added as a collaborator.

You will not be able undo this.

To confirm this operation, please enter the following details

- Current owner: organization-name
- Project name: project-to-transfer

[Transfer Ownership]

Organization Collaborators

If you add an organization as a collaborator to one of your projects, all members of the organization will be granted collaborator-level access.

Typical Uses for Organizations

Simple project access management

Project access management is the most common use for organizations in Domino.

Domino sees organizations and users as near-equivalents; the only real difference is that an organization contains multiple users. Because of this, organizations are a handy tool for simplifying project access for groups of users.
For example, inviting a team to join a project you are working on is as simple as creating an organization consisting of all the members of that team, and then adding the organization to the project. This is also useful in the event that you elect to remove the team from the project. You can also break up an existing organization into smaller groups, and then un-invite the original team while retaining the smaller, more relevant organization.

You can also use organizations to provide restricted access to your projects, instead of full access. For example, to give a team read-only access to a project, simply add the organization to your project and set its role to Results Consumer. If there’s a group that should only be able to use Launchers, add the organization and give them Launcher User access.

Organization admins have the power to add or subtract people to their organization. Adding users to (or removing them from) organizations instead of each individual project that organization owns can be a real time-saver. Simply add or remove them once, at the organization level. Domino will take care of updating their access to each project for you.

Projects in production

Organizations are also a useful way to manage projects that have reached a “production grade”-level of quality. Users can set up organizations that are specifically intended to be homes for projects that are ready to be promoted to production, or that have already been promoted. Users working with projects owned by this organization will understand that these projects should be carefully documented and kept free of clutter. Any changes that are needed will require a fork and merge request process. Runs executed in these projects end up being very deliberate, with the purpose of updating data or models.

The process of using organizations to manage projects promoted to production can unfold in one of two ways. The first occurs when a project starts out as an individual user’s project. That user invites others to this project, either individually or by using the Organizations feature, and over time other users come to find this project particularly useful. After a certain point, a decision is made that access to the project should be easier, and that it should be owned by the organization instead of the individual user.

The other path involves initial ownership of the project by an organization. This usually happens when the project is deemed to be important from the very beginning, and the project can best be built up by a group. As soon as this project is created, all users have access to all its resources, and project development proceeds mostly by forking and merging. Execution of code is still done in the organization-owned project as a way to check in your analysis or results.

Some Domino customers do this even for models that are not actually in production, as a way of submitting their work to be counted.

Sharing compute environments

One of the advantages of organizations is that they streamline and simplify the practice of sharing compute environments. Project owners who are members of an organization can use all of that organization’s compute environments for their own projects.

For example, if a user is a member of a corporate organization, whatever environments that organization’s admins have created will be available to the user for their own projects. Domino does not constrain users to membership in just one organization. If a user is a member of two organizations - a ‘corporate organization’ with production environments and an ‘R&D organization’ with more cutting edge environments - that user has access to the compute environments of both organizations, as well as any global environments and environments the user already owns directly.
What would happen if this user were to be removed from an organization he was a member of, even though several of his projects rely on that organization’s environments? In that case, the now-unavailable environments would be reset to the user’s default environment. Domino sends a notification to any affected users whenever this happens.

5.10 Use the Domino command line interface (CLI)

5.10.1 Installing the Domino Command Line (CLI)

1. Click your username at the bottom of the main menu to download the CLI installer.
2. Run the installer application.

3. Open a new command prompt and run:

   \[ \text{domino login <url>} \]

   Replace \texttt{<url>} with the URL where you access Domino.

4. When asked for your login information, use the username and password for your Domino account.

5. You should see something like this:

   \[
   \begin{align*}
   \text{\$ domino login https://app.dominodatalab.com # or the url of your domino deployment} \\
   \text{Enter your Domino username or email address} \\
   > \text{username} \\
   \text{Enter your password (typing will be hidden)} \\
   > \\
   \end{align*}
   \]

   Congrats, you’re all set! Check out the \texttt{CLI reference} for documentation about using the CLI.

   Having trouble? Ask in our Community Hub, or email us.

5.10.2 CLI reference
• Installing and getting started with the CLI
• To run your code
• To create a new project
• To create a project in a directory that already has files
• To get a project from the server onto your computer for the first time
• To connect a folder to an existing project
• To synchronize the files on your computer with the server
• To download the latest version of your files
• To upload your files without starting a run
• To get help
• To get the version of your CLI

Installing and getting started with the CLI

How to install and get started with the Domino CLI

To run your code

run [--direct][--wait][--no-sync][--tier][--local] <file> [args...]

“run” will copy all files in your project folder to the cloud, where it will execute the specified file with any arguments you have specified, unless you have specified the –local flag.

The optional --wait flag will run your script synchronously, i.e., the command won’t return until your job finishes.

The optional --direct flag lets you run a shell command directly on the Domino machines.

The optional --no-sync flag will run the latest version of code on the server, without uploading any local changes on your computer.

The optional --tier flag lets you select a specific hardware tier to execute the specified file on.

The optional --local flag lets you run your script on your local machine.

The optional --title flag lets you supply a title for the run.

Local runs are useful if your desired compute environment is not immediately available, but you would like to record your results in Domino. When a local run is invoked, Domino commits a snapshot of project files to the server and then runs the given command (e.g., “python main.py”) on the local machine. When the command has completed, Domino detects results that have been produced and commits those back to the server, tracking them as “results” of the run as though the command ran on the server.
Viewing the “Results” for Run #16 will bring up a screen showing the output of the main.py where colleagues can view and discuss the output.

**Examples**

```bash
domino run calculateStats.R
domino run runModel.m model1 .05
domino run --wait runModel.m model1 .05
domino run --direct "pip freeze | grep pandas"
```

5.10. Use the Domino command line interface (CLI)
“create” will create a new project by making a new folder on your computer and telling the Domino server about your new project. Optionally, you can specify the name of the project. If you don’t specify a name, Domino will ask you for one. To create a project on behalf of an organization you belong to, use the --owner flag.

**Examples**

```
domino create
domino create myProject
domino create --owner myOrg myProject
```

---

**To create a project in a directory that already has files**

```
init [projectName]
```

“init” will create a new Domino project inside your current folder. This is useful if you already have a working folder and you’d like to convert it to a Domino project. While the “create” command creates a new folder inside your current directory, “init” will initialize a project from your current folder. The --owner flag can be used to set an organization you belong to as the project owner.

Note: “init” will **not** upload any of your files, you’ll need to use “upload” or “run” after you run “init”.

**Examples**

```
domino init
domino init myProject
domino init --owner myOrg myProject
```

---

**To get a project from the server onto your computer for the first time**

```
get [username]/projectName
```

“get” will find an existing project on the server and copy it down to your computer. This is useful to get a project you created on a different computer, or to get a project that someone else created, and you are collaborating on.

If you are getting someone else’s project, specify their username and the project name. If you are getting one of your own projects, you can omit the username.

**Examples**


To connect a folder to an existing project

restore

“restore” will “connect” your current directory to an existing Domino project on the server. Restore will look for evidence of a project name inside “.domino/config.json”. If it doesn’t find such a file, it will prompt you for the name of an existing project to use.

This is particularly useful if you using Domino and git to track the same folder. If you clone a project with git, git will likely have ignored the hidden files that identify it has a Domino project. So you can “git clone” and then “domino restore” to re-connect the folder to its Domino project.

Examples

domino restore

To synchronize the files on your computer with the server

sync

“sync” will synchronize the project folder on your computer with the project stored on the server. This is equivalent to running a download followed by an upload.

Examples

domino sync
To download the latest version of your files

download

“download” will download the latest copy of your files from the cloud into your current project folder. If you have made changes that conflict with changes in the cloud, you will see both versions of the conflicting file side-by-side.

Note that there are two reasons files in the cloud might change: first, your collaborators on a project might make changes; second, you might have executed a run that produced new output files.

Examples

domino download

To upload your files without starting a run

upload [-m "message"]

“upload” will upload your current project folder contents to the cloud, but will not begin a new run. The optional message flag lets you record a message, which will be displayed when browsing past commits on the “Files” tab of your project.

Examples

domino upload -m "this is a great message"

To get help

help

“help” will print out a list of commands you can run, with information about each one, similar to this page.

Examples

domino help
domino help run
To get the version of your CLI

```
version
```

“version” will output your CLI version

**Examples**

```
domino --version
```

### 5.10.3 Downloading individual files or folders with the CLI

If you need to download individual files from a project, the simplest way to do this is from the Files page in the UI. However, if your files are too numerous to manage from the UI, there is an alternate method using a CLI command:

```
domino download-results.
```

This command accepts a filter that allows you to download a subset of the project. Documentation of this command is available by running:

```
domino help download-results
```

As an example, you could download the contents of a folder with a command like this:

```
domino download-results --filter 'myfolder/*' 548fda01e4b0a8f06b7e8d99
```

- The `--filter` option is what limits the downloaded files to the pattern that follows it
- `548fda01e4b0a8f06b7e8d99` is the run ID

You can also use a run number instead of an ID:

```
domino download-results --filter 'myfolder/*' 1
```

Note that this downloads the output files from a run, so it requires you to have submitted at least one run in the project. Files appear in the local directory under `results/run-<runId or runNumber>`.

### 5.10.4 Force-restoring a local project

If you encounter errors with the Domino CLI, it may be due to a problem with your local project metadata. This can happen for example if a file is modified by another program during the sync and file locking is turned off. You may be able to resolve this issue by performing a force-restore. To do so, follow these steps:

1) Navigate to your local copy, and remove the Domino project information.

```
cd <project folder>
rm -rf .domino
```

5.10. Use the Domino command line interface (CLI)
2) Restore the Domino project information. Enter the project name when prompted.

```
domino restore
```

Note that this will download any files that are on the server that are not present in your local project. Simply delete them if you do not need them. Additionally, if any of your local files are different from the server version, the local version will be pushed the next time you run ‘domino sync’.

### 5.10.5 How to move a project from one Domino deployment to another

These are the steps to take to move a project from one Domino deployment to another. For example, perhaps you had a project in our Trial deployment and you want to move it to our Production deployment or to your private Domino deployment. This can be done via the CLI. It’s important to note that this process only moves the latest files, and not your past history and revisions.

For more information about the CLI, please visit Installing the Domino Client (CLI).

This assumes you have the latest local copy of your project on your workstation. Please review “How to” / CLI reference on how to “get” a local copy, if you do not have a local copy.

Here are the commands to use in the CLI:

1. Navigate to your project on your local system that you’d like to migrate
   ```
   cd <your project>
   ```
2. .domino is what identifies it as a domino project, so we remove it
   ```
   rm -rf .domino
   ```
3. Log into the target deployment you’re moving to, initialize and upload your project
   ```
   domino login <target URL>
   domino init
   domino upload
   ```

Once you have completed the steps above, log into the new Domino system where your project is hosted and verify the access and collaboration settings are correct. In some cases, access and collaboration settings may need to be updated as they do not transfer with the project.

Your project migration is complete.

### 5.10.6 Using the Domino CLI Behind a Proxy

You can configure the Domino CLI to use your proxy with the following changes. You’ll need to modify files in the client installation directory:

- On Mac, the directory is: `/Applications/domino`
- On Windows, the directory is: `C:\Users\<your_username>\AppData\Local\Programs\domino`

In the root of the installation directory, there is a file named `domino.vmoptions` with some configuration options available. Add the line `-include-options ~/.domino/domino.vmoptions` at the end of the file for any *nix based system (Windows users will need to specify the appropriate path where `domino.vmoptions` will be created):

```
-Ddomino.defaultHost=
-Dhttp.proxyHost=
-Dhttp.proxyPort=
-Dhttp.nonProxyHosts=
```

(continues on next page)
You will need to create the user level configuration file `domino.vmoptions` in the appropriate location/folder. The folder `~/.domino/` may not exist, in which case you should run `mkdir ~/.domino` to create it. A user level configuration file is necessary to ensure your settings are never modified in the scenario that the CLI is reinstalled or updated. In the newly created `~/.domino/domino.vmoptions`, assign your proxy’s port and host configuration:

```
-Dhttp.proxyHost=YOURPROXYHOST
-Dhttps.proxyHost=YOURPROXYHOST
-Dhttp.proxyPort=1234
-Dhttps.proxyPort=1234
```

You’ll need to replace `YOURPROXYHOST` and `1234` with your actual proxy connection information.

### Proxy Authentication with the Domino CLI

CLI supports **Basic** & **NTLM** proxy authentication. Add the following parameters to your `~/.domino/domino.vmoptions` file for proxy authentication:

**Basic**

```
-Djava.net.useSystemProxies=false
-Djdk.http.auth.tunneling.disabledSchemes=
-Dhttps.proxyUser=YOURPROXYUSER
-Dhttps.proxyPassword=YOURPROXYPASSWORD
-Dhttp.proxyUser=YOURPROXYUSER
-Dhttp.proxyPassword=YOURPROXYPASSWORD
```

**NTLM**

```
-Dhttp.auth.ntlm.domain=YOURNTLMDOMAIN
-Dhttps.auth.ntlm.domain=YOURNTLMDOMAIN
```

**Note:** The proxy username is case sensitive when using **NTLM** proxy authentication.
5.11 Browser Support

As part of our effort to make data science teams more productive by providing a secure, central system of record, Domino strives to deliver a consistent, equal user experience across whichever platforms our users prefer.

Domino supports the following browsers on Windows and MacOS desktop:

- Google Chrome (latest 3 major versions)
- Firefox (latest 3 major versions)
- Safari (latest 3 major versions)
- Microsoft Edge (latest 3 major versions)

The following browsers are officially not supported:

- Internet Explorer
- Mobile browsers
- Beta, “preview”, or other pre-release versions of desktop browsers
6.1 Domino API

Much of Domino’s functionality is available programmatically through:

- the Domino REST API
- the Domino API Python binding
- the Domino CLI

Note that this is different from our Model API functionality:

- Model API lets you publish your R or Python function as a low-latency web service.
- The Domino REST API lets you control Domino’s core functionality programmatically.

6.2 Python wrapper for Domino API

The Python binding for the Domino API is installable via pip:

```
pip install git+https://github.com/dominodatalab/python-domino.git
```

If you would like to use the Python binding within a Domino workbook session simply include the following line to your project’s requirements.txt file. This will make the Python binding available for each new workbook session (or batch run) started within the project:

```
-e git+https://github.com/dominodatalab/python-domino.git#egg=domino
```

You can find Domino’s public Python bindings project [here](#).

Once installed, you can instantiate library either with your API Key or Auth Token File.

**Token File:** To use Token File to instantiate Python domino, you need to pass the path to this token file, either via class constructor (domino_token_file=<Path to Token file>) or via environment variable.
export DOMINO_TOKEN_FILE=PATH_TO_DOMINO_TOKEN_FILE

If you are using Python package in code running in that is already running in Domino, The DOMINO_TOKEN_FILE will be set automatically to be the token file for the user who started the run.

API Key: you’ll need to get your API key from your account page. To get your API Key, log into Domino and click on your name on the right hand side of the top menu. Select Account Settings and select the API Key option from the left hand menu. Copy the API key to your clipboard.

The Python library will read this key from environment variables, so set it as follows in your shell:

export DOMINO_USER_API_KEY=YOUR_API_KEY

If you are using the Python package in code that is already running in Domino, the DOMINO_API_USER_KEY variable will be set automatically to be the key for the user who started the run.

Note:
1. In case both API Key and Token file are present, default preference will be given to token file. To use API Key instead, unset the (DOMINO_TOKEN_FILE) environment variable
2. Documentation for the Domino REST API can be accessed here.

Here is an example of usage:

```python
from domino import Domino

# By and large your commands will run against a single project,
# so you must specify the full project name
domino = Domino("chris/canon")

# List all runs in the project, most-recently queued first
all_runs = domino.runs_list()['data']

latest_100_runs = all_runs[0:100]

print(latest_100_runs)

# all runs have a commitId (the snapshot of the project when the
# run starts) and, if the run completed, an "outputCommitId"
# (the snapshot of the project after the run completed)
most_recent_run = all_runs[0]

commitId = most_recent_run['outputCommitId']

# list all the files in the output commit ID -- only showing the
# entries under the results directory. If not provided, will
# list all files in the project. Or you can say path="/" to
# list all files
files = domino.files_list(commitId, path='results/')['data']

for file in files:
    print file['path'], '->', file['url']

print(files)
```

(continues on next page)
# to handle it
print(domino.blobs_get(files[0]["key"]).read())

# Start a run of file main.py using the latest copy of that file
domino.runs_start(["main.py", "arg1", "arg2"])

# Start a "direct" command
domino.runs_start(["echo 'Hello, World!'"], isDirect=True)

# Start a run of a specific commit
domino.runs_start(["main.py"], commitId="aabbccddee")

## 6.3 Domino R package

Access the full power of Domino’s cloud compute and version control without leaving your R environment.

- **Install**
- **Logging In**
- **Getting a Project**
- **Running Code**
- **Downloading Results**

### 6.3.1 Install

The package is hosted on CRAN, like any standard R package.

Make sure you have the Domino command line client installed (you won’t use it directly, but it does need to be installed on your computer).

Then, from inside R run:

```r
install.packages("domino", repos='http://cran.r-project.org', type="source")
```

Once it’s installed you can import the package as you would any other R package:

```r
library(domino)
```

### 6.3.2 Logging In

You’ll need to login again from R to verify your identity to Domino.

To do that type:

```r
domino.login("yourUsername", host="https://app.dominodatalab.com")
```

You should get a success message. Note that if you’re running in a private deployment, your “host” will be the address of your internal Domino server.
6.3.3 Getting a Project

Note: If you already have a copy of your project on your computer, then you can skip this step. Set your working directory to the root of that project folder and continue to the next section.

To download an existing project, run this command:

```
domino.get("quick-start")
```

You should see something like this:

```
> domino.get("quick-start")
Getting project chris/quick-start...
Project get complete. Make sure to navigate to your new project folder by running:
  cd quick-start
To run a file on Domino, type:
  domino run [command-to-run]
[1] "Changed working directory to new project's director"
```

This command also changes your working directory to the root of the project folder. In general, these Domino commands will only work if your working directory is set to be inside your project folder.

6.3.4 Running Code

To start the execution of a script on Domino, type:

```
domino.run("myScript.R")
```

You should see something like this:

```
> domino.run("myScript.R")
Determining which files are out of date...
No changes to upload to server, because you're up to date.
Run for project chris/quick-start started. You can view progress here:
  https://app.dominodatalab.com/chris/quick-start/run/53051188e4b0b5aeedd6a73b
```

You can also pass in arguments into the scripts:

```
domino.run("myScript.R", 1, "someString")
```

You can read in those parameters in your script:

```
> firstParameter 1
> secondParameter "someString"
```

You can thus launch many jobs:

```
for (alpha <- 1:3) {
  for (beta <- 1:3) {
    domino.run("myScript.R", alpha, beta)
  }
}
```

This will queue 9 jobs, each with a unique setting of the values alpha and beta.
6.3.5 Downloading Results

Once your runs have completed, you get download the results back to your computer by running:

```r
domino.download()
```

This will download all the new and modified files in your project to your local computer. Once this command completes, these files will be available immediately to be loaded into your R workspace:

```r
> domino.download()
Downloading latest changes to project...
Determining which files are out of date...
Changes from the server:
------------------------
  x Modifying file results/stdout.txt
  x Modifying file someData.RData
Starting download of new files:
-------------------------------
  Downloading "someData.RData" ...
    complete!
  Downloading "results/stdout.txt" ...
    complete!
Download complete.
> load("someData.RData") # --> load the R objects
```

6.3.6 Two-Way Sync

You can perform a two-way sync with the command:

```r
domino.sync()
```

This runs a download followed by an upload. If there is a conflict between the local and server versions of a file, you will get both versions so that you can fix the conflict and then re-sync.

6.3.7 Initializing a Project

To initialize a new project inside an existing directory, navigate to the directory and then run:

```r
domino.init("projectName")
```

This will create a Domino project named “projectName”, with its root at the current working directory. You can then sync this project to the server with

```r
domino.sync()
```

6.3.8 Other Commands

You can also run other commands from the CLI, with methods named according to their CLI counterparts. For additional details, you may reference the documentation hosted on CRAN.
7.1 Domino 4.4 (February 2021)

New Features

• **Durable Workspaces** – Stop and resume your workspaces sessions as needed. Your work persists from session to session, ensuring that you never lose your work and enabling you to commit your work to version control whenever you want. To learn more, refer to the [workspace sessions documentation](#).

• **Git-based Projects with CodeSync** – Native integration with widely used Git repositories is provided by CodeSync technology in new Git-based project type to help data scientists save, find and reproduce work, and engage in version-controlled, code-based collaboration with other team members. Create branches, commit your code, and push or pull changes to your code, all from within a workspace, in a Git-based project with CodeSync. To learn more, refer to the [Git-based Projects with CodeSync documentation](#).

Upgrades & Enhancements

• The Domino Analytics Distribution (DAD) has been upgraded to use Python 3.8. Notable upgrades are listed below. Please see the corresponding changes outlined in the “Breaking Changes” section.
  
  – Upgrade to R 4.0 from 3.6
  – Upgrade to Python 3.8 from 3.6
  – Upgrade Scala kernel to 2.12.9 and almond to 0.10.9
  – Upgrade Julia to 1.5.3 from 1.1.0
  – Upgrade JupyterLab 2.2.9 from 1.2.12
  – Upgrade Jupyter Notebook 6.1.6 from 6.0.3
  – Upgrade (VS Code) Code server 3.7.3 from 3.4.1
  – New Python language server for VS Code
  – Python extension fix for VS Code
  – Jupyterlab-git extension for Jupyterlab
Breaking Changes

- The default Domino Analytics Distribution (DAD) uses Python 3.8, however, PySpark 2 does not support Python 3.8 or higher. If you need a PySpark 2 environment, make sure to build it from images that use Python versions < 3.8.

- CUDA has been removed from the upgraded DAD in Domino 4.4.0 to keep the size of the DAD image to a minimal size. If you need to use CUDA 10 or CUDA 11, use the following image: quay.io/domino/base:Ubuntu18_DAD_Py3.8_R4.0-20210127_CUDA11.0_full. Please note that drivers must be present on the underlying hosts.

- You can now set your own volume size for your workspace sessions and Jobs. Your Domino administrator can configure the minimum, maximum, and default sizes for the volume. In Domino 4.4.0, the default volume size is 10 GB. If your Domino deployment previously utilized a custom volume setting in a prior version of Domino, make sure to edit the corresponding central configuration values after upgrading to Domino 4.4.0 to ensure your runs execute as intended.

- If you are upgrading to 4.4.0 from Domino version 4.1.9 or earlier, there is a Keycloak client mapper that Domino depends on which will need to be manually created prior to upgrading.

  To create the mapper:
  - Go to the Clients tab in the DominoRealm in Keycloak and select the domino-play client.
  - Select the Mappers tab for the domino-play client
  - Create a new mapper with type User Attribute and the following settings:
    * Name: <Your desired friendly name>
    * Mapper Type: User Attribute
    * User Attribute: domino-system-roles
    * Token Claim Name: roles
    * Claim JSON Type: string
    * Add to ID token: Yes
    * Add to access token: Yes
    * Add to userinfo: Yes
    * Multivalued: Yes
    * Aggregate attribute values: No

- URL authorization will fail when attempting to use an older version of the Domino CLI with Domino 4.4.0. To avoid this issue, we recommend reinstalling the Domino CLI.

Workarounds

- After modifying email settings, manually restart services for the email settings to take effect.

- Emails are not sending to user(s) mentioned using the “@” tagging feature in a file comment or job comment. Please communicate with the intended user(s) in a different way.

- If you’re launching a Job in a Git-based project with CodeSync, use absolute paths if you’re specifying the name of a file outside of the /mnt/code working directory. The .. command is currently not supported.

- If you’re using a Launcher in a Git-based project with CodeSync, note that files or scripts specified in the “Command to run” text area will execute in /mnt/code. Be sure to account for this if your script results in outputs you’d like to persist.

Bug Fixes
• Fixed an issue that prevented workspaces from launching in Domino if Domino was deployed with Istio.
• Fixed an issue that affected file creation and file editing in projects.
• Fixed an issue that prevented Spark clusters from launching if they used a GPU-based hardware tier.
• Fixed an issue that prevented users with the “Practitioner” role from creating a project.
• Fixed an issue that prevented Domino runs from executing properly if a file in a linked repository contained non-ASCII characters.
• Restored the “Support” button for all Domino 4.x deployments.
• Fixed an issue that prevented Domino workspaces from being stopped.
• Fixed an issue that prevented data frames from being read properly when using a Spark cluster.
• Fixed an issue that prevented model APIs from being properly exported.
• Fixed an issue that caused failed attempts to push to a protected branch in an external Git repository to appear to have succeeded.
• Fixed an issue that prevented a Git credential from being created if the domain was entered as an IP address.
• Fixed an issue that prevented new models from being created.
• Fixed an issue that prevented project settings from being properly propagated to Spark clusters.
• Fixed an issue that allowed users to edit their own roles.
• Fixed issues that prevented usage report emails from being sent.
• Fixed an issue that prevented datasets from mounting properly when executing a Domino run using the “Save & Run” option present in the “Files” page of a Domino project.
• Fixed an issue that prevented SMPT configuration values from being properly set.
• Fixed an issue that prevented users from viewing datasets in their project if the dataset originated from an imported project.
• Fixed an issue that prevented scratch spaces from being assigned to workspaces that were launched from the “Files” page of a Domino project.
• Fixed an issue that prevented workspaces from launching if a snapshot that was marked for deletion was mounted to another project through a shared dataset.
• Fixed an issue that caused a 502 error when attempting to generate a user activity report.
• Fixed an issue that prevented some Python-based extensions from functioning properly in workspaces that utilized VS Code.
• Fixed an issue that caused files specified in .modelignore to continue to appear with a published model’s files.
• Fixed an issue that allowed multiple hardware tiers to be designated as the “default” hardware tier for a Domino deployment.
• Fixed an issue that affected the “Name” field in cloned hardware tiers.
• Fixed an issue affecting “Command to Run” parameters when using a Launcher.

Browser Support for Domino
Domino is supported on the latest stable version of most browsers. Version 4.4.0 of Domino was tested on the following browsers:

• Chrome 88 on Windows 7
• Firefox 85 on Windows 7
• Safari 14 on MacOS 10.15 Catalina
• Edge Chromium 88 on Windows 10

7.2 Domino 4.3.3 (December 2020)

New Features
• Access AWS Resources from On-Demand Spark Clusters – You can now configure on-demand Spark clusters in your Domino workspace to access AWS resources using temporary credentials issued by AWS. To enable this feature, your Domino deployment must use single sign-on (SSO) with a trusted identity provider (IdP). To learn more about this feature, please refer to the corresponding documentation.

• Access Data in External Data Volumes – Your Domino projects can now access data stored in external storage volumes, like Network File Systems. External data volumes are supported in Jobs (including Scheduled Jobs), Workspaces, Apps, Launchers, and on-demand Spark clusters. To learn more about this feature, please refer to the external data volumes documentation.

• Domino 4.3.3 is now compatible with Istio (version 1.5+), which provides TLS encryption for service-to-service communication inside the Domino cluster. Domino installation tools can be configured to install Istio 1.7 controllers and agents, or Domino can be installed in a cluster that is already running Istio 1.5+.

• Domino now supports adding custom tolerations to the Kubernetes workloads it deploys, allowing compute nodes to have Domino-specific taints that prevent non-Domino pods in a multi-tenant cluster from being scheduled on them.

• The next generation containerized environment image builder is now in limited availability for Domino 4.3.3.

Breaking Changes
• Domino user roles and authorization have been overhauled to introduce new user types, streamline access control, and assist with license management. Read more about user roles in the updated administrator’s guide. Note that if you use SSO role synchronization, you must add the new Practitioner role to your users in SSO, or they may lose access to some features.

• The default volume size in Domino 4.3.3 is 5 GB.

Upgrades & Enhancements
• Organize multiple sets of credentials using labels in the “Git Credentials” section of your Domino account settings.

• Receive immediate notifications if there are authentication or access problems when using git repositories in Jobs or Workspaces.

• On-demand Spark cluster usage is now recorded and included in run export data.

• Project collaborators with the “Collaborator” role can now modify hardware tiers and environments.

• Fixed HTTP to HTTPS redirect behavior in the Domino login flow that will trigger an unsecure redirect warning beginning in Chrome v88. Users of Chrome with automatic updating enabled will need to upgrade to Domino 4.3.3+ prior to January 19, 2021, when v88 is scheduled to become the new stable version of Chrome, to avoid seeing this warning on login.

Workarounds for earlier versions of Domino are:

– Do not upgrade to Chrome v88 stable
– Unset the #mixed-forms-interstitial feature flag in chrome://flags
– Ignore the warning at login and click Continue
Upgrading to Domino 4.3.3 is the recommended path.

- Restrict contributors from adding additional contributors to a project. To learn more, refer to the “Authorization” feature flags in the Domino administrator’s guide.

Bug Fixes

- Fixed a UI issue that visually obscured Job results.
- Fixed an issue that impaired the readability of a Run’s status.
- Fixed an issue that prevented Goals from being linked to a Domino project.
- Fixed an issue that prevented searching of Goals when linking a goal.
- Fixed an issue that caused the base image in an environment to revert back to the default global environment when editing an environment.
- Fixed a UI issue that displayed an incorrect value for the number of PVCs provisioned when utilizing a Spark cluster. The correct number of PVCs, however, were correctly provisioned.
- Fixed an issue that incorrectly allowed an environment to use itself as the base image.
- Fixed an issue that caused resource consumption to be displayed with incorrect units when viewing resource usage for a Job.
- Fixed an issue that prevented archived Workspaces and Jobs from displaying in Domino.
- Fixed an issue that caused Domino to display an incorrect number of Spark executors when launching a workspace with a Spark cluster. The specified number of Spark executors, however, were correctly allocated to the workspace.
- Fixed an issue that prevented workspaces from properly detecting file changes when performing volume recovery.
- Fixed an issue that prevented workspace recovery with a salvaged volume if a re-login to Domino was required.
- Fixed an issue that prevented image source projects from being downloaded.
- Fixed a pagination issue that caused search results from being properly displayed in the Environments dashboard.
- Fixed an issue that prevented environment search in the Environments dashboard if the name of the environment contained special characters.
- Fixed an issue that impaired workspace search in the Workspaces dashboard if the name of the workspace contained special characters.
- Fixed an issue that prevented workspaces from launching if a user was removed from an Okta group with credential propagation enabled.
- Fixed an issue that prevented the Usage chart for a project from displaying recent usage data.
- Fixed a UI issue in the workspace launch modal that impaired workspace launches.
- Fixed an issue that caused the asset count to display an incorrect value in the Project Portfolio view of the Domino Control Center.
- Fixed an issue that prevented Jobs from being scheduled if the job utilized an on-demand Spark cluster.
- Fixed an issue that prevented project ownership from being transferred to an organization.
- Fixed an issue that caused a 500 error in the Domino REST API when publishing an app using the Domino Python wrapper.
- Fixed an issue that prevented models from being exported as an endpoint to the SageMaker API.
• Fixed an issue that prevented the Jira credential panel from appearing in your Domino account settings.
• Fixed an issue that prevented git repo service providers from being added or selected when adding git repo credentials.
• Fixed an issue that caused Domino to become unresponsive when adding Jira credentials.
• Fixed an issue that prevented Bitbucket repositories from being imported and prevented the creation of projects when using a Bitbucket repository that required credentials.
• Fixed an issue that prevented anonymous users from using Launchers associated with public Domino projects.
• Fixed an issue that impacted file permissions in imported / exported projects.

7.3 Domino 4.3.2

7.3.1 4.3.2 (November 2020)

Bug Fixes
• Fixed an issue that prevented adding users to an organization in bulk if a duplicate user was detected.
• Fixed an issue that caused the assets portfolio to become unresponsive if the portfolio contained a large number of assets.
• Fixed an issue that caused a stopped workspace to re-launch when re-logging into Domino after credential expiration.

7.4 Domino 4.3.1

7.4.1 4.3.1 (October 2020)

New Features
• Domino can now be installed and run on the latest versions of OpenShift (OCP 4.4+). To learn more, refer to the Domino OpenShift documentation in the Domino Administrator Guide.

Upgrades & Enhancements
• Domino now uses version 11.0.2 of Keycloak.
• You can now override the shared memory (/dev/shm) limit of 64MB in an execution container to utilize more memory. To learn more, refer to the Domino Compute Grid documentation in the Domino Administrator Guide.

Bug Fixes
• Fixed an issue that caused the Domino UI to become unresponsive if an invalid URI was entered when attempting to import a GitHub repository.
• Fixed an issue that prevented image size limit errors from being recorded in the Domino build logs.
• Fixed an issue that prevented scheduled jobs from adding a Spark cluster.
• Fixed an issue that prevented searching and filtering shared datasets.
• Fixed an issue that prevented creating datasets through file uploading.

7.5 Domino 4.3

7.5.1 4.3.0 (August 2020)

New Features

• Domino now generates short-lived JWT tokens that can be used to authenticate to third party resources, data sources, and the Domino API. Tokens are made available in Workspaces, Jobs, or Apps. Learn more about Domino’s JWT tokens here.

• All components of Domino On-demand Spark clusters are now displayed with key infrastructure and configuration information in the administrator’s execution dashboard.

Changes

• fleetcommand-agent v25 released to support 4.3.0 installation.
• Domino now uses version 10.0.1 of Keycloak.
• Domino now uses version 7.7.1 of Elasticsearch.
• Fixed an issue that prevented hardware tiers from functioning as intended if the Cores Limit field was left blank. Execution specifications would still be produced with a limit equal to the CPU request. Now, leaving the Cores Limit field blank will correctly produce execution specifications with no CPU limit.
• Fixed an issue that prevented GPU-based Spark clusters from functioning properly.
• Fixed an issue where restarting a node would cause apps in Domino to fail.
• Fixed an issue that prevented workspaces from properly detecting file changes when performing volume recovery.
• Fixed an issue that prevented workspace recovery with a salvaged volume if a Domino session was expired.
• Fixed an issue where the base image in an Environment changed when other non-image changes were made to the Environment.
• Fixed an issue that prevented archived workspaces and jobs from appearing in their corresponding dashboard.

7.6 Domino 4.2

7.6.1 4.2.4 (July 2020)

Changes
• Fixed an issue that prevented workspaces from being stopped.

7.6.2 4.2.3 (July 2020)

Changes
• Improved logging in the Image Cache Agent.
• Fixed an issue that prevented file uploads through the Domino CLI.
• Fixed an issue where the Scheduled Jobs page failed to load if a job could not find a hardware tier.
• Fixed an issue that caused Workspace & App links in the Compute & Spend report to return errors, preventing access to the associated job.
• Fixed an issue that caused the /health endpoint to return a 404 incorrectly.
• Fixed an issue where Spark worker storage would not be provisioned unless the value for the storage size was modified. The default storage of 30GiB is now created without requiring custom input.

7.6.3 4.2.2 (May 2020)

Changes
• Fixed an issue where collaborators attempting to start workspaces in projects they did not own would get redirected to the wrong URL, resulting in 404.
7.6.4 4.2.1 (May 2020)

Changes

- Improved performance of the files browser in the UI
- Fixed an issue where user-uploaded custom certificates were not being used when connecting to Git servers for cloning and working with externally hosted repositories.
- Domino 4.2.1 is supported by fleetcommand-agent v23 or greater. See the fleetcommand-agent release notes for breaking changes and new options.

7.6.5 4.2.0 (May 2020)

New features

- Domino 4.2 introduces the ability to launch and work with on-demand Spark clusters backed by the Domino compute grid. When setting up an environment, you can now use a base image with Spark worker and client components installed, such as bitnami/spark:2.4.5-debian-10-r84 and then mark the environment as ready for use with Domino managed Spark.

Click to read more about environment setup for on-demand Spark.

Once your environment is configured, you can attach on-demand spark clusters to jobs and workspaces that use that environment. The cluster and clients will be automatically configured, and you can use PySpark or
spark-submit in your job or workspace code to interact with the cluster.

- In Domino 4.2, if a workspace stops or shuts down unexpectedly without syncing files, and the attached volume enters a salvaged state, the user will automatically be prompted to start a recovery workspace that will reconnect to the salvaged volumes for syncing.

- Domino in AWS now supports custom KMS key encryption of S3 stores.

- The Keycloak authentication service deployed alongside Domino 4.2 comes pre-configured with a SAML identity provider that has valid default user attribute mappers, plus pre-configured client mappers for AWS credential propagation and organization or role mappings.

- New API endpoints have been introduced for working with Model APIs. You can now use the API to export a Model API image in a Sagemaker-compatible format directly to your own container registry, and there are also new endpoints for starting, stopping, and getting the deployment status of Model APIs.

- Domino projects can now integrate with Atlassian Jira to link Domino work to Jira issues and projects.

To enable this feature, a Domino administrator must configure a Jira integration. From the admin portal, click Advanced > Jira Configuration, then supply your Jira URL to start the integration.
Once a Jira integration has been added, users can go to the **Manage** tab of a project overview to authenticate to the Jira service and connect the Domino project to a Jira issue. Users can then link Domino project goals to Jira subtasks, and comments and project status changes will automatically be recorded in Jira.

Changes

- The user interface for workspaces has been redesigned to support new features like separate interface tabs for the Spark UI when using on-demand Spark clusters.
The workspaces dashboard has also been redesigned for simplicity and readability.

The process for launching a new workspace is now started from the **New Workspace** button at top right. This button opens a redesigned dialog for starting a workspace.
• Performance improvements to the workspace startup flow have reduced the minimum time to start a workspace, and additional progress information is now shown during startup.

• A number of issues related to reverse proxy and subdomain access to workspaces have been fixed.

• The default resource requests for Domino platform services have been tuned to provide baseline capacity for orchestrating 300 concurrent executions. Read the infrastructure sizing overview to learn more.

### 7.7 Domino 4.1

#### 7.7.1 4.1.9 (March 2020)

Changes

• Added `ShortLived.iFrameSecurityEnabled` feature flag that removes the `sandbox` attribute from app iframes when set to false.

• Domino Jobs that are missing the executable file to run will now move to an errored state instead of hanging until killed.

• Fixed issue where Workspace syncs could fail when syncing very large numbers of files.

• Fixed issue where a user with `Launcher User` permissions on a project would incorrectly see a 403 error when trying to open the project.
• Fixed some issues with uploading files to projects through the UI that would cause unintended Request entity too large errors.

• Fixed issue where the Windows CLI installer was using an incorrect link for installing the Java runtime.

• Fixed issue where no_proxy settings in cluster container runtimes were not being respected by Domino containers.

• Fixed issue where the hardware tier dropdown in launcher dialogs would sometimes be empty.

Known Issues

• Model APIs, advanced routing mode does not work with the model tester UI in the model API overview page. It will always return the result for the latest version of the model. Note that this is just a UI bug and the actual model URLs for older versions will work as expected. This issue affects 4.1.0 - 4.1.9

7.7.2 4.1.8 (February 2020)

Changes

• Fixed performance issues with workspace log polling.

• Fixed issue where resetting passwords for local user accounts would fail following upgrade.

• Fixed issue where deploying an app would fail when running Domino in Kubernetes v1.16+ due to Kubernetes API changes.

• Fixed various issues where pressing ENTER would not trigger submission on some forms, dialogs, and modals.

• Fixed issue where scaling the number of instances of a deployed model, or deploying a new model version, would result in a short period of unavailability. The model will now correctly maintain availability of the existing deployment until the new deployment is fully ready.

• Fixed issue where some app would fail to publish due to issues with formatting on iframe sandbox attributes.

• Improved error message when attempting to open files as workspaces where the file extension is mapped to multiple pluggable tools in the default environment.

• Improved error message when trying to access a private project for which your user is not authorized.

7.7.3 4.1.7 (February 2020)

Changes

• Domino executions will no longer immediately enter an errored state with Error retrieving run messages if no frontends are available.

• Fixed an issue where Domino running on EKS with Calico installed would consistently fail to start the first execution on a newly launched node created by cluster scale-up.

• Fixed an issue where executions would fail to start if the $DEFAULT_COMPUTE_ENV_IMAGE was not present on the node.
7.7.4 4.1.6 (January 2020)

Changes

• Leading and trailing whitespace on central configuration keys and values entered in the administration console will now be automatically trimmed.

7.7.5 4.1.5 (January 2020)

Changes

• Fixed an issue where the link to the executable file from the details panel of a Job would error due to quotes around the filename in the URL not being removed.

7.7.6 4.1.4 (January 2020)

Breaking changes

• All arguments supplied to a job or launcher in the Domino UI will be quoted with single quotes. Previously, it was possible to supply shell environment variables that would be expanded.

Changes

• Fixed an issue where transient issues with heartbeats from execution pods in a deploying state would lead to the execution moving to an error state, aborting deployment.

• Fixed an issue where workspaces would hang during startup if there was an error executing requirements.txt.

7.7.7 4.1.3 (December 2019)

Changes

• Changed the default setting of com.cerebro.domino.dispatcher.internalUrl from the dispatcher external URL to http://dispatcher-nucleus.domino-platform, which resolves via service discovery to the actual internal URL.

• Improved replicator garbage collection.

• Fixed issue where attempting to log in from the logout page would just reload the logout page.
7.7.8 4.1.2 (December 2019)

Changes

- Fixed an issue where Domino could not push to remote Git repositories via PAT credentials during a workspace sync.
- Fixed an issue where using Bitbucket PAT credentials containing `/` characters would cause Domino to fail to clone repos from Bitbucket. The PAT is now encoded and decoded like a URI.
- Fixed an issue where the dataset mounting page would sometimes show the owner and project name on a dataset as unknown.
- Fixed an issue where datasets scratch spaces would sometimes fail to delete.
- Removed the ability to set pod memory limits in hardware tier definitions. Memory limits are now automatically set to the same value as the pod memory request.
- Users with the Support Staff role can now access User Activity Reports in the administration console.

7.7.9 4.1.1 (December 2019)

Changes

- Fixed issue where if the run container in a Domino execution pod stopped unexpectedly, the executor container could be orphaned and left running, causing the pod to not be deleted.
- Added a ShortLived.LaunchersAccessible feature flag. When set to false, the launchers page in the UI is disabled.
- Fixed issue where project names that contained only numbers, such as a project named 123, were not quoted correctly in API payloads and therefore various operations on them would fail, including starting runs.
- Fixed issue where a newline at the end of a Full Delete Message would cause the full delete operation to fail.
- Fixed issue where Domino runs with Spark integration enabled were passing the run pod name as the `spark.driver.host` which was unresolvable by external clusters.
- Added two central config options for enabling setting Domino application roles and organization membership based on SAML attributes:
  - `com.cerebro.domino.authentication.oidc.externalRolesEnabled` when true will apply a Domino admin role for users with SAML attribute values containing role names mapped to a `domino-system-roles` user attribute.
  - `com.cerebro.domino.authentication.oidc.externalOrgsEnabled` when true will set organization membership for users with SAML attribute values containing organization names mapped to a `domino-groups` user attribute.
- Fixed an issue where setting organization membership from SAML attributes was incorrectly enforcing case sensitivity when matching organization names to the user attribute values.
- Fixed an issue where EBS volumes used as Domino execution persistent volumes were sometimes not deleted correctly during garbage collection, and could become orphaned by Domino.
- Fixed an issue where users would sometimes see Request entity too large errors when trying to upload files larger than 1MB to a project.
7.7.10 4.1.0 (November 2019)

Domino 4.1 delivers powerful enhancements to enterprise authentication by enabling credential propagation through Domino to other systems that user code interacts with. Domino 4.1 also introduces new options for administrators to manage per-user compute resource consumption, and adds the ability to fully purge data from the Domino file store.

New features

- Domino 4.1 introduces a full delete operation that allows Domino system administrators to completely remove all instances of a file from the Domino file system. Performing a full delete on a file finds all instances of the file’s contents across all revisions of all projects, erases those contents wherever they appear, and replaces them with a message indicating that the file was subject to a full delete. This will affect all files that have identical contents to the target file, even if they have different filenames. It will not affect files with the same filename if they have different contents.

To perform a full delete, open the file in question and click the Full Delete button. Full delete can only be performed on one file at a time.

You will be prompted to supply a commit message before confirming the full delete. The full delete will remove all instances of the file contents from the Domino file system. Note that this operation does not alter the revision history of any of the affected projects. All commits that contained the file contents will continue to exist, but when viewing the file contents within you will only see the full delete message.

One limitation of full delete is that if any Model APIs have been published from project revisions that contained a file that has been fully deleted, the Model API images will continue to contain that file. There is currently no way to permanently purge that image from Domino. Contact support@dominodatalab.com if you have questions about such files.
• Domino now supports mapping assertions from your SAML identity provider or SSO service to Domino admin roles.

To automatically assign admin roles to a user, you will need to assert one of the following role values in a SAML attribute statement for that user:

- SysAdmin
- Librarian
- ReadOnlySupportStaff
- SupportStaff
- ProjectManager

Consider this example assertion that could be used to assign a user system admin roles:

```xml
<saml2:AttributeStatement xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
  <saml2:Attribute Name="DominoSystemRoles">
    <saml2:AttributeValue xsi:type="xs:string">SysAdmin</saml2:AttributeValue>
  </saml2:Attribute>
</saml2:AttributeStatement>
```

Additional instructions on how to finalize these configurations with mappers and enable this feature in the Domino will be available in the Keycloak authentication section of the Domino Administrator’s Guide.

• Domino now supports mapping assertions from your SAML identity provider or SSO service to Domino organization membership.

To automatically grant a user membership in a Domino organization, you will need to pass the names of the organizations as values in a SAML attribute statement for that user, such as in the following example.

```xml
<saml2:AttributeStatement xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
  <saml2:Attribute Name="DominoOrganizations">
    <saml2:AttributeValue>nyc-data-scientists</saml2:AttributeValue>
    <saml2:AttributeValue>all-data-scientists</saml2:AttributeValue>
    <saml2:AttributeValue>sensitive-claims-users</saml2:AttributeValue>
  </saml2:Attribute>
</saml2:AttributeStatement>
```

Additional instructions on how to finalize these configurations with mappers and enable this feature in the Domino will be available in the Keycloak authentication section of the Domino Administrator’s Guide.

• Domino 4.1 introduces the ability to propagate AWS temporary credentials for accessing external systems into Domino run environments. This requires federation from AWS IAM to your SAML identity provider.

To enable credential propagation for a user, you must pass SAML attributes for that user that define the IAM roles they can assume and configurations for temporary credentials. Domino will then enable automatic requests...
for temporary credentials when the user signs in, and will load them into an AWS credentials file in the user’s run environments.

Additional instructions on how to finalize these configurations with mappers and enable this feature in the Domino will be available in the Keycloak authentication section of the Domino Administrator’s Guide.

- In Domino 4.1 admins can now optionally configure a maximum number of simultaneous executions per user. This can be used to prevent individual users from monopolizing available compute resources. This feature is enabled by default, but can be turned off with the `ShortLived.UserExecutionsQuotaEnabled`.

The maximum number of per-user executions is configurable with a new `com.cerebro.domino.computegrid.userExecutionsQuota.maximumExecutionsPerUser` option in the central config, and it defaults to 25. If a user attempts to start a new execution when already at the limit, the new execution will be queued in a `WaitingForResources` state until one of their existing executions finishes.

Executions that remain in a `WaitingForResources` state for more than 24 hours will be terminated. This timeout is configurable with a new `com.cerebro.domino.computegrid.timeouts.sagaStateTimeouts.waitingForResourcesStateTimeoutSeconds` option in the central config.

Note that Model APIs and Web Apps are exempt from these limits, as they are considered long-lived services hosted by Domino and are not treated as variable user demand. Jobs, Scheduled Jobs, and Workspace sessions are subject to the limits.

- A new option is available in the Executions tab of the admin interface to download a Support Bundle. This option downloads a `.zip` archive of all logs related to the selected execution, including Domino events logs, Kubernetes deployment logs, and Domino application logs.
• Hardware tiers now have an Overprovisioning Pods option. This number represents a number of empty pods that will be deployed to the cluster using the hardware tier’s resourcing requests and node pool settings. If overprovisioning pods are present when a new user-launched execution pod is sent to the cluster for assignment, the user’s pod replaces one of the overprovisioning pods.

Admins can use this feature to build in some automatically reserved capacity for the hardware tier, so that user executions on the hardware tier always have a node with reserved resources ready to go, and users will never have to wait for autoscaling to spin up a new node.
Additionally, admins can check the **Enable Overprovisioning Pods On a Schedule** option to limit deployment of overprovisioning pods to a specified range of hours on specified days of the week. When outside the scheduled period, overprovisioning pods will be removed to allow the cluster to scale down. When the scheduled period begins again, overprovisioning pods will be deployed.

A new `com.cerebro.computegrid.periodicProcessing.overprovisioningInterval` option has been added to the central config that sets how often Domino checks to see if new overprovisioning pods are needed. This defaults to 600 seconds.

- Deployment logs for Domino executions are now available in `.csv` format. Click the `.csv` link to download the new format. Clicking the `Deployment logs` link will still download the logs in the original JSON format.
• New API endpoint have been added to enable exporting Docker images of Model APIs built in Domino to external registries. Additional details will be available in the Domino API docs.

• Added a new `com.cerebro.domino.modelmanager.requestBufferSize` option in the central config that controls the maximum size in bytes of a request block to a Domino Model API. This defaults to 8192 and can be set to a value from 1-65535.

Changes

• Improved messages on the Workspace startup page that provide more granular information about the startup process.
• The **Cores limit** and **Memory limit** options have been removed from Domino hardware tiers. Domino now automatically sets the limit values equal to the request values to ensure stable and predictable behavior from Domino execution pods and keep assignment simple and reliable.

• Fixed an issue where having a large number of published models failing to deploy could cause future attempts to publish new models to fail in deployment scheduling. Published models that fail to deploy and end up in a crash loop will now be automatically unscheduled to improve reliability of scheduling newly published models.

• Domino 4.1 improves performance, UX responsiveness, and user feedback design in many components across the application.

### 7.8 Domino 4.0

#### 7.8.1 4.0.4 (November 2019)

**Changes**

• Fixed an issue where some execution events tracked by Domino could be logged or presented out of order.

• Fixed an issue where Domino executions with Spark integration could create Kubernetes resources in the wrong namespace.

• The console output panel for a Domino run will now surface and display more types of errors.

#### 7.8.2 4.0.3 (October 2019)

**New Features**

With the release of Domino v4.0.3, **Datasets** functionality has been added to the new platform infrastructure. A feature first introduced in V3.3 is now accessible with the updated architecture.

**Changes**

• Fixed issue where the “modified” column in the Environments and Models table of the UI wouldn’t sort chronologically.
• Restores support for connecting to VPNs from Run containers
• Various minor bug fixes and stability improvements

7.8.3 4.0.2 (October 2019)

Changes
• Fixed issue where Model API’s timeout override was not taking effect
• Fixed issue where Control Center could become inaccessible when a job’s queue end time and run end time are the same time stamp.
• Various additional minor bug fixes and stability improvements

7.8.4 4.0.1 (October 2019)

Changes
• Multiple minor bug fixes and adjustments to the default configuration settings for new deployments

7.8.5 4.0.0 (September 2019)

Welcome to Domino 4!

In addition to helpful new features for data scientists and project leaders, Domino 4 introduces a new architecture with all components running on Kubernetes. This change makes Domino easier to install, configure, monitor, and administer, and allows Domino to run in more environments than ever before. Visit admin.dominodatalab.com to learn about the technical design of Domino 4 and read guides for configuration and administration.

Breaking changes
• Domino 4.0 fully sunsets support for V1 environments. Previously, V1 environments had been demarcated with an asterisk when listed in your project settings environments list. Typically, these should not be present for Domino deployments which originated after the release v3.0.
• Domino 4.0 fully sunsets support for legacy API endpoints. Only Model APIs are supported. Typically, legacy API Endpoints should not be present for Domino deployments which originated after the release v3.0.
• Many previous interfaces and options for managing Domino executors (e.g. the legacy “Dispatcher” interface) have been replaced with the introduction of the new Kubernetes compute grid. There are new dashboards for viewing Kubernetes infrastructure and active execution pods, and new options for configuring Hardware Tiers.
Domino Documentation, Release 4.4

Click to read more about Managing the compute grid in Domino 4.

• Domino 4.0 removes support for SSH access to a Run container.

• Domino 4.0 removes support for arbitrary Docker arguments for things like custom volume mounts.

• Domino 4.0 temporarily removes support for connecting to VPNs from Run containers. Support returns in 4.0.3.

• In Domino 4.0, user logins must use the new Keycloak authentication service. Any existing legacy LDAP integrations will need to have their configurations migrated to Keycloak.

• Domino 4.0 ships with a new collection of Domino 4.0 standard environments. Users who want to use NVIDIA GPUs in Domino 4.0 will need to switch their compute environments to the latest version as Domino now utilizes NVIDIA Docker. Note that these new standard environments do not support working with GPUs in Python 2.

New features

• Domino now runs fully kubernetes native. Both front ends, central services and executors now run on the Domino kubernetes platform. Read more about the new infrastructure.

• Domino 4.0 adds a new Assets Portfolio that allows users to quickly discover and see key information about the data products they have access to in Domino, including Model APIs, Apps, Launchers, and Scheduled Jobs.

• A new Project Manager admin role is available. This role grants a user contributor access to projects owned by other users who are members of the same organization as the project manager. This allows the project manager to view those projects in the Projects Portfolio, discover their published assets in the Assets Portfolio, and view the projects’ contents as a contributor.
• Domino 4.0 introduces Project Goals. Goals represent outcomes or subtasks within projects. Project contributors can link files, Workspace sessions, Jobs, Apps, and Model APIs to goals, which show up on the goal card in the project overview. This provides a way to track all work related to a specific goal in the project, and can make navigating large and busy projects easier.

• New options are available in the Notifications and Workspace Settings sections of user Account Settings that allow for opt-in to email notifications or auto-termination for long-running Workspace sessions with a configurable duration.

 Admins also now have additional options for defining which Workspace sessions to treat as long-running, en-
forcing notification requirements for users, and sending additional global notifications about long-running sessions to admins.

**File size units**

To harmonize file size formats across Domino, and to align with common practices in user interfaces, starting with Domino 4.0, file sizes are displayed using base 10 metric prefixes (e.g. $1\text{GB} = 10^9 \text{bytes}$) as opposed to base 2 binary prefixes (e.g. $1\text{GiB} = 2^{30} \text{bytes}$). For additional information on the differences between the units, please see [https://en.m.wikipedia.org/wiki/Byte#Multiple-byte_units](https://en.m.wikipedia.org/wiki/Byte#Multiple-byte_units).

The change affects the file summary screen and other locations where file sizes are displayed in Domino. As a result, a user may observe a visual difference between the reported size in GB, MB, or KB of a file between Domino 3.6 (or earlier) and Domino 4.0, even though the absolute size of a file in bytes has not changed.

**Additional changes**

- Visual styling and design for tables, buttons, links, accordion headers, breadcrumbs, and tab navigation have all been improved and made consistent across the Domino application.
- Run usage functionality is impaired and will be addressed in an upcoming Domino version.

### 7.9 Domino 3.6

#### 7.9.1 3.6.29 (November 2020)

**Changes**

- Fixed an issue that prevented logins via the Domino CLI from Windows hosts.

#### 7.9.2 3.6.28 (August 2020)

**Changes**

- Fixed an issue that occasionally prevented workspaces from starting, resulting in the error “The image name was not set.”
- Fixed an issue that, under certain circumstances, caused a run to launch on more than one executor.

#### 7.9.3 3.6.27 (June 2020)

**Changes**

- Fixed an issue that caused an error to occur when installing the Domino CLI with MacOS Catalina.

7.9. Domino 3.6 621
7.9.4 3.6.26 (June 2020)

Changes

• Fixed an issue that resulted in excessive CPU usage when using a Launcher for projects with many runs.
• Fixed an issue that caused Domino to become unresponsive when a large files were uploaded.

7.9.5 3.6.25 (May 2020)

Changes

• Fixed an issue that caused the contents of an imported git repo to visually disappear within the repository’s file directory in Domino.
• Fixed an issue that prevented run logs from appearing in the Domino UI.

7.9.6 3.6.24 (April 2020)

Changes

• Maintenance release to update internal components.

7.9.7 3.6.23 (March 2020)

Changes

• Fixed an issue that prevented project size from being calculated when launching a run.
• Fixed an issue that prevented Domino deployments from connecting to MongoDB instances.

7.9.8 3.6.22 (March 2020)

Changes

• Fixed an issue that caused the Jobs dashboard to become unresponsive when there were many active runs.
• Fixed a CPU usage issue that caused Domino to become unresponsive.
• Domino now uses MongoDB 3.6. For more information on this version of MongoDB, please refer to the release notes for MongoDB 3.6.
7.9.9 3.6.21 (February 2020)

Changes

- Fixed an issue that prevented users from uploading files into Launchers when using Microsoft Edge 42.
- Fixed an issue that prevented notifications about long-running workspaces from displaying properly.
- Fixed an issue that caused downtime when scaling a model.

7.9.10 3.6.20 (January 2020)

Changes

- Fixed an issue where links to files in subdirectories in the Job details panel would result in a 404 due to quotes around the file path.

7.9.11 3.6.19 (January 2020)

Changes

- Fixed issues related to use of the File parameter type in advanced Launcher editing causing the Launcher interface to fail to render. Domino will now better handle cases of empty or invalid default values for such parameters.
- Fixed issue where following links on the Datasets overview would sometimes lead to a stalled loading spinner requiring a refresh.
- All parameters passed to Launchers are now single quoted to preserve spaces and other special shell characters.
- Added retry for some S3 operations that could experience intermittent service level failure.

7.9.12 3.6.18 (December 2019)

Changes

- Improved garbage collection processes in platform services to reduce disk usage.
- Fixed an issue where attempting to view an invite-only app while not logged in did not correctly redirect the user to the login flow.
- Consolidated platform messaging queues to improve platform performance and durability.
• Fixed an issue where the Login button on the You have been logged out of Domino page did not correctly redirect the user to the login flow.

7.9.13 3.6.17 (December 2019)

Changes
• Fixed an issue where first-time login for a new user could fail to trigger new-user setup, instead showing an error until the user navigated to a new interface. First-time logins should now reliably trigger new-user setup and redirect the new user to their projects page with their default projects created.
• Fixed an issue where models could fail to start if there were many models queued for publishing.
• Added new styling to the logout page.
• Fixed an issue where the com.cerebro.domino.publicProjects.enabled setting did not affect creation via API. Setting that option to false now disables the ability to create public projects both in the UI and in the API.

7.9.14 3.6.16 (November 2019)

Changes
• Added URI-encoding to Bitbucket PATs used by Domino to protect against errors caused by unsupported characters in the password.
• Fixed an issue where Domino Datasets containing a very large number of individual files could fail to mount in runs.

7.9.15 3.6.15 (November 2019)

Changes
• Fixed an issue where the control center could fail to load if many runs are queued.
• Fixed an issue where passing the username of a Domino app viewer would intermittently fail.
• Updated Domino management tooling to require that AMIs used for Domino executors have EBS auto-termination enabled.
• Improved performance of loading detailed run information across the UI.
7.9.16 3.6.14 (October 2019)

Changes

• Fixed an issue where admins were unable to publish a new version of a model when the admin was not an owner or contributor on the project containing the model.

• Fixed an issue where some text input fields in the Kubernetes volumes section of the advanced model settings page were too small to see a reasonable amount of text input. These fields have been resized.

7.9.17 3.6.13 (October 2019)

Changes

• Fixed an issue where the environment revisions page would sometimes incorrectly label an inactive version of an environment as the Active version. This interface will now only mark the actually active version.

• Fixed an issue where attempting to reopen a recently closed workspace in Edge 17+ would result in a 404 error.

7.9.18 3.6.12 (October 2019)

Changes

• Improved performance of loading model instance logs in the UI.

• Removed checkboxes from the rows in the app versions dashboard, as there were no bulk actions available.

• Fixed an issue where attempting to change a project collaborator’s role could result in an error.

• Setting a custom override for model timeouts will now work as expected. Previously, models would time out after a maximum of 60 seconds even if a longer override was set.

• The search box on the workspaces dashboard now supports filtering by tag.

7.9.19 3.6.11 (September 2019)

Changes

• Removed the checkboxes on table rows when there are no bulk actions available on objects in the table.

• Clicking the button to duplicate an environment now redirects the user to the duplicate environment description page. Previously, taking this action would reload the original environment’s description page.
7.9.20 3.6.10 (September 2019)

Changes

• Fixed an issue where stopping a Workspace from the Workspaces dashboard would erroneously indicate to the user that they were discarding changes even when there were no uncommitted changes in the session.

7.9.21 3.6.9 (August 2019)

Changes

• Improved performance of loading model instance logs in the UI.
• Improved garbage collection for temporary folders and volumes on executors.
• Fixed an issue where stopping and committing from the Workspace UI would not execute the post-run scripts from the user’s environment.
• Added a `modelManager.modelContainer.restartCountLimit` option which defines how many times a Model API can fail to launch before being descheduled and not restarting further.

7.9.22 3.6.8 (August 2019)

Changes

• Restyled the support button so that it no longer obscures other interactive UI components.

7.9.23 3.6.7 (August 2019)

Changes

• Fixed an issue with event telemetry reporting that could cause some actions in the UI to hang indefinitely.
• Fixed an issue where logs would sometimes erroneously indicate a lack of executor capacity.

7.9.24 3.6.6 (August 2019)

Changes

• Added a `modelmanager.requestBufferSize` to override the size of the uWSGI request buffer for published models. This will allow creating models that take larger requests without causing invalid HTTP request size errors.
• Fixed an issue that could cause opening a Workspace to 404 for users with uppercase characters in their LDAP federated usernames.

7.9.25 3.6.5 (August 2019)

Changes
  • Fixed an issue with CLI login.

7.9.26 3.6.4 (August 2019)

Changes
  • Fixed an issue where the support widget could cover up interactions in the application.

7.9.27 3.6.3 (July 2019)

Changes
  • Added a button to the dispatcher admin UI to download the files from the working directory of an active Run.

7.9.28 3.6.2 (July 2019)

Changes
  • Fixed an issue where if a user had made changes in a workspace, then reopened that workspace in a new tab or window, attempting to stop and commit changes would not correctly commit the changes.

7.9.29 3.6.1 (July 2019)

Changes
  • Fixed some issues with rendering custom support buttons.
  • Admin users who stop their own workspaces will no longer receive a notification that an admin stopped their workspace.
• Fixed a rendering issue with the help link in the workspace logs panel.

7.9.30 3.6.0 (July 2019)

Breaking changes

• The legacy Runs dashboard is no longer available in Domino 3.6. After upgrading to 3.6, the new Jobs dashboard interface will be enabled.

Changes

• Domino 3.6 introduces Datasets scratch spaces. These are mutable filesystem directories for temporary data storage and exploration. They are a complement to the core Datasets functionality. Read more about scratch spaces here.

• New API endpoints for working with Datasets are available.

• The following types of events have been added to project activity feeds:
  – Publishing a Model API
  – Publishing an App
  – Publishing or modifying a scheduled Job
  – Creating, editing, or deleting files from the Domino UI
  – Creating, editing, or deleting files as the result of a Workspace sync
- A new timeline component has been added to the Jobs dashboard. This component shows a time series of dominostats.json values being tracked across experiments.

- If you are a collaborator on a project, and an administrator or project owner stops one of your Workspace sessions, you will now get an email notification with details.

- For new deployments, Domino 3.6 introduces a new authentication service that supports additional protocols and SSO providers.

- Tables throughout the UI have been switched over to a new component type for unified and improved styling.

- Previously, when an executor in Maintenance Mode was stopped, starting the executor from the Dispatcher UI would take the executor out of Maintenance Mode automatically. Now, the executor will start but remain in Maintenance Mode. Executors will only exit Maintenance Mode when an administrator manually toggles Maintenance Mode.

- Fixed some issues with model tester connectivity to published models.

- Fixed formatting issues with UI text on the environment definition page.

- Fixed an issue with port assignment for Spark that could impact connectivity.
CHAPTER 8

Get help with Domino

8.1 Find documentation for your version of Domino

Any user can look up the version of Domino that they are currently running by going to `<your domino url>/version`. For example, users on try.domino.tech can see the version at try.domino.tech/version.

To view documentation for a specific version of Domino, click Read the Docs at the bottom of the left navigation menu, then click the version you’re looking for. Note that Domino only has versioned documentation for versions 3.6+

8.2 Contact Domino technical support

8.2.1 Submitting a ticket

There are three ways to submit a ticket to the Domino technical support team:

1) The easiest way to submit a ticket is through our ticketing portal at tickets.dominodatalab.com.
2) Another option is via the Help widget in the bottom right of your Domino UI.
3) Finally, you can also email support@dominodatalab.com.

For critical issues, always submit a ticket via the web form or widget.

8.2.2 Guidelines

- Try to provide as much information as you can
- If there are steps we can use to reproduce your problem, tell us what they are
• For issues with a specific Run, please include a link to the run and a screenshot of the error
• For issues with the CLI, please include the contents of the domino.log file from the folder in which you ran the command
• For issues with projects, please include a link to the project
• For issues with environments, please include a link to the environment

8.3 Support bundles

Occasionally, when working with the Domino support team, you may be asked to retrieve a support bundle. A support bundle is ZIP archive containing logs and reports from various Domino components, with information pertaining to a specific Domino execution. Support bundles are available for Jobs, Workspaces, and Apps.

To retrieve a support bundle for an execution, you need its execution ID. This can be found as part of execution assignment messages at the top of the Setup Output in the logs panel for the execution. The messages will look like this:

```
Successfully assigned <node-pool>/run-<execution-id>-<pod-id> to <node-name>
```

For example, in the screenshot below, the execution ID is 5e3451b1f7c7e006d99d80.

Once you have the execution ID, you can retrieve that execution’s support bundle by visiting the following URL:

```
<domino-url>/v4/admin/supportbundle/<execution-id>
```