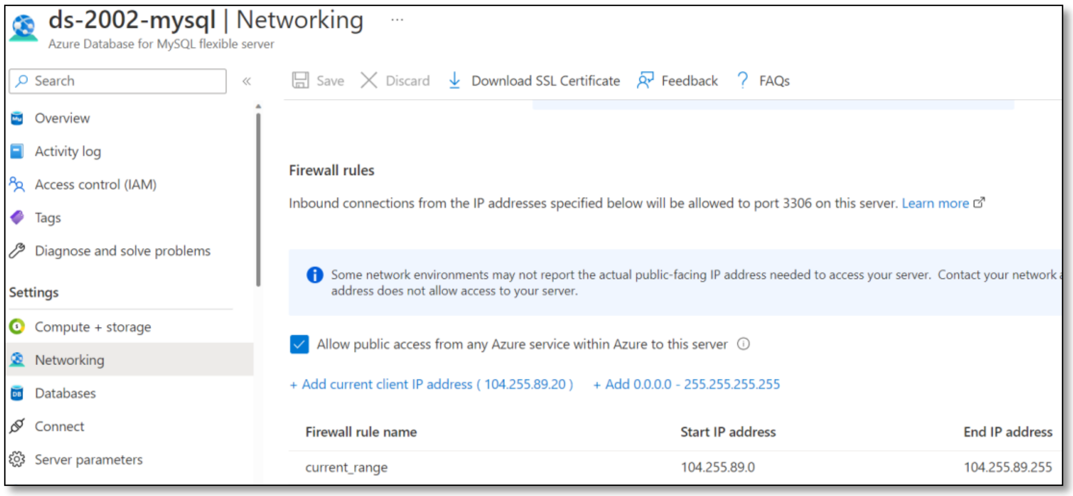
Connecting to Azure MySQL

There are several steps to configuring access to your MySQL server/database from your laptop, and from your Databricks cluster.

Accessing Azure MySQL from Workbench on your Laptop

First, once you've created the MySQL server, create a firewall rule to enable your laptop to connect to the MySQL server. This will allow you to connect using MySQL Workbench for the sake of creating your project's "source" database.



Firewall rules

Inbound connections from the IP addresses specified below will be allowed to port 3306 on this server. Learn more

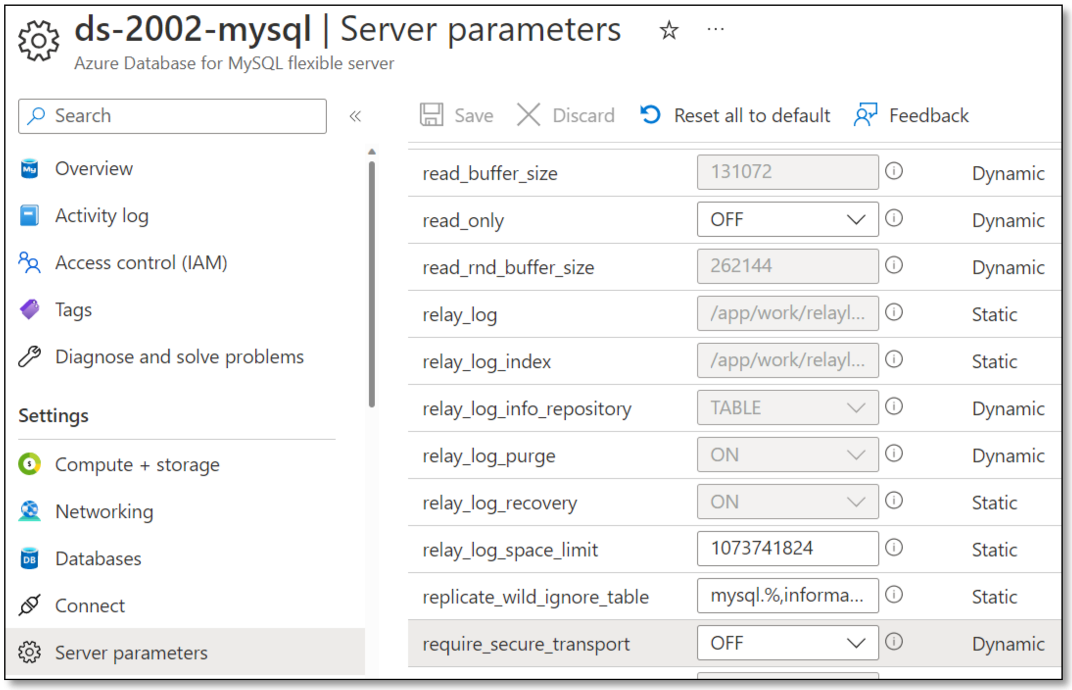
Some network environments may not report the actual public-facing IP address needed to access your server. Contact your network
address does not allow access to your server.

☑ Allow public access from any Azure service within Azure to this server

+ Add current client IP address (104.255.89.20)   + Add 0.0.0.0 - 255.255.255.255

Firewall rule name	Start IP address	End IP address
current_range	104.255.89.0	104.255.89.255

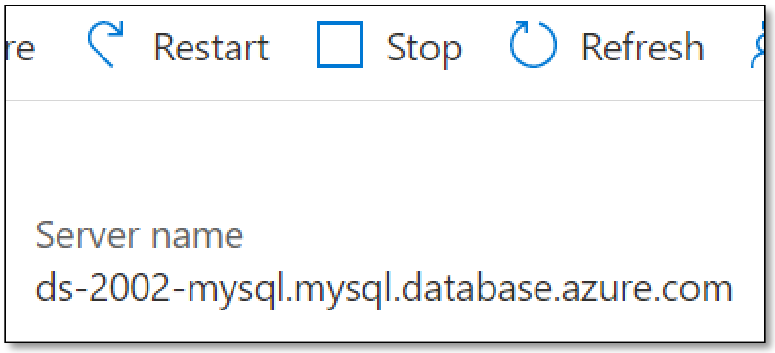
Second, because your Azure MySQL Server is configured to only accept connection over an SSL encrypted connection, you must reconfigure the **require_secure_transport** server parameter to **OFF.**



read_buffer_size	131072	Dynamic
read_only	OFF	Dynamic
read_rnd_buffer_size	262144	Dynamic
relay_log	/app/work/relayl...	Static
relay_log_index	/app/work/relayl...	Static
relay_log_info_repository	TABLE	Dynamic
relay_log_purge	ON	Dynamic
relay_log_recovery	ON	Static
relay_log_space_limit	1073741824	Static
replicate_wild_ignore_table	mysql.%,informa...	Static
require_secure_transport	OFF	Dynamic

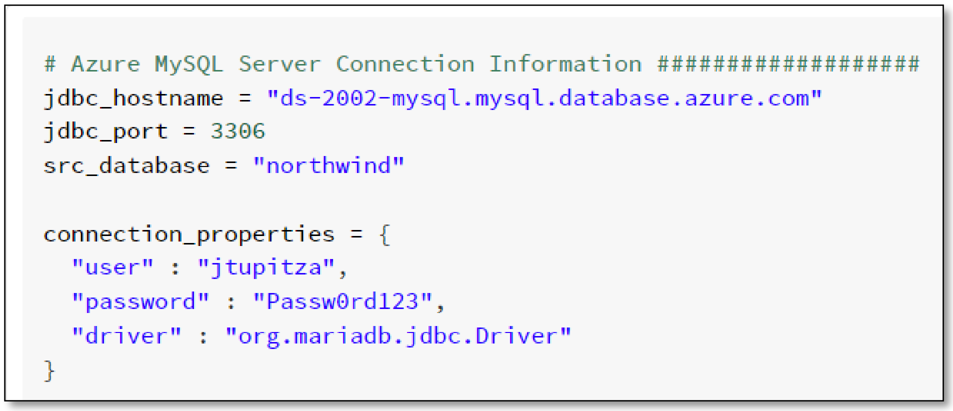
Accessing Azure MySQL from a Databricks Notebook Running on your Cluster

First, take note of the **Server name** of your MySQL server in Azure.



Server name
ds-2002-mysql.mysql.database.azure.com

Second, set the **jdbc_hostname** variable to your Azure MySQL **Server name** property, and be certain to configure the **connection_properties** dictionary to use the correct **user** name, **password**, and **driver** for MySQL Server.



```
# Azure MySQL Server Connection Information ###################
jdbc_hostname = "ds-2002-mysql.mysql.database.azure.com"
jdbc_port = 3306
src_database = "northwind"

connection_properties = {
  "user" : "jtupitza",
  "password" : "Passw0rd123",
  "driver" : "org.mariadb.jdbc.Driver"
}
```

The sample notebook contains the following function definition that I created. Notice that the connection string (**jdbcUrl**) is formatted for SQL Server, but not for MySQL in Azure.

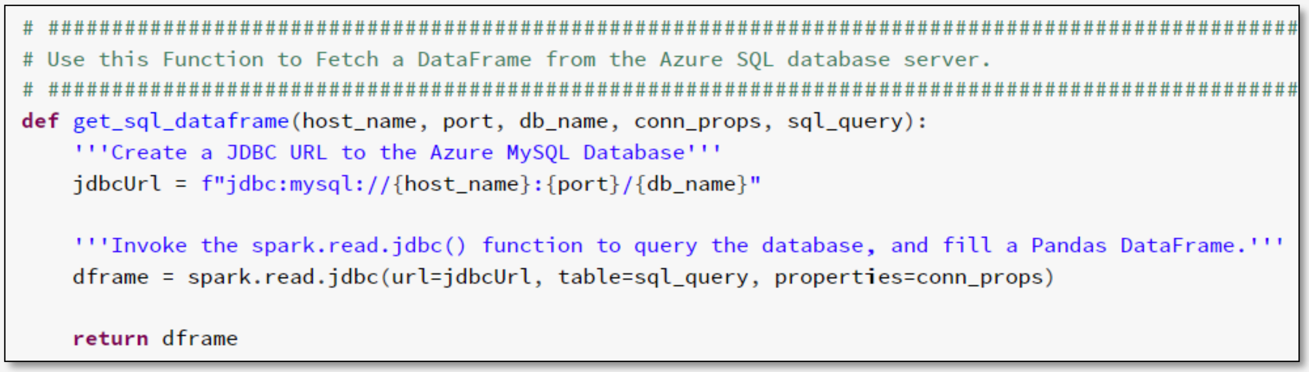
```
#######################################################################################
# Use this Function to Fetch a DataFrame from the Azure SQL database server.
#######################################################################################
def get_sql_dataframe(host_name, port, db_name, conn_props, sql_query):
    '''Create a JDBC URL to the Azure SQL Database'''
    jdbcUrl = f"jdbc:sqlserver://{host_name}:{port};database={db_name}"

    '''Invoke the spark.read.jdbc() function to query the database, and fill a Pandas DataFrame.'''
    dframe = spark.read.jdbc(url=jdbcUrl, table=sql_query, properties=conn_props)

    return dframe
```

You need to alter the following line of code to make it work with MySQL:

Change: `jdbcUrl = f"jdbc:sqlserver://{host_name}:{port};database={db_name}"`

To: `jdbcUrl = f"jdbc:mysql://{host_name}:{port}/{db_name}"`

Your completed function should look like this:

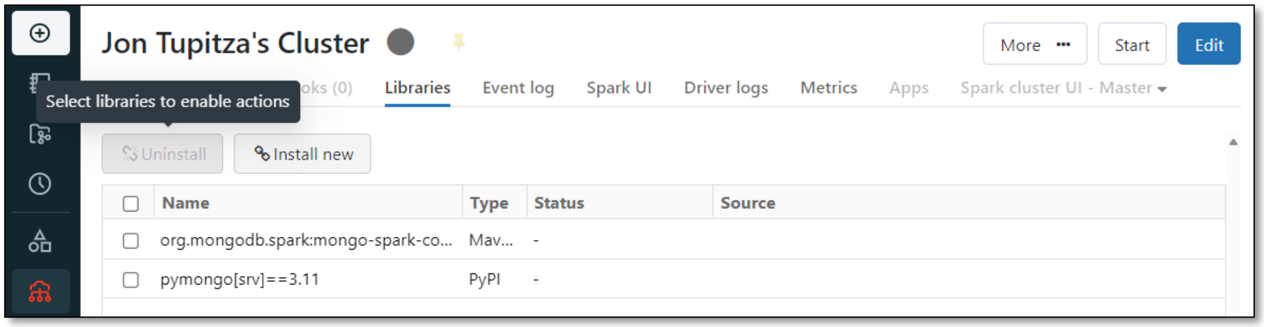


```
# ##########################################################################################
# Use this Function to Fetch a DataFrame from the Azure SQL database server.
# ##########################################################################################
def get_sql_dataframe(host_name, port, db_name, conn_props, sql_query):
    '''Create a JDBC URL to the Azure MySQL Database'''
    jdbcUrl = f"jdbc:mysql://{host_name}:{port}/{db_name}"

    '''Invoke the spark.read.jdbc() function to query the database, and fill a Pandas DataFrame.'''
    dframe = spark.read.jdbc(url=jdbcUrl, table=sql_query, properties=conn_props)

    return dframe
```

Connecting to MongoDB Atlas in Azure

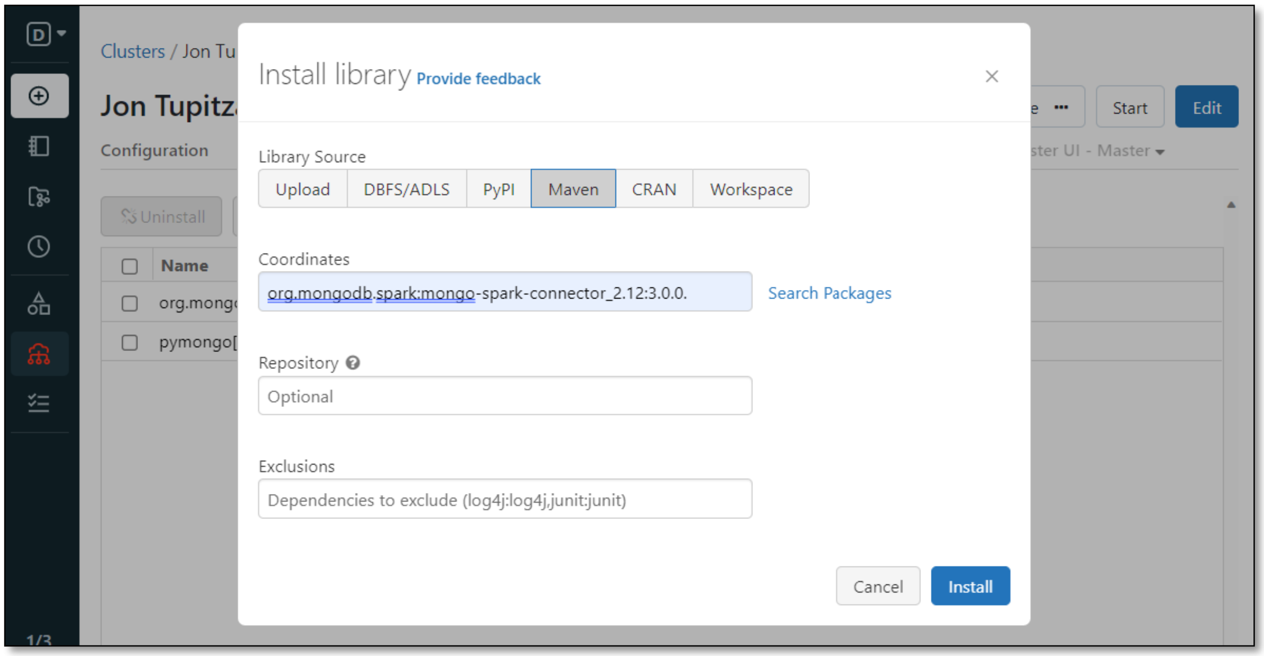
There are several steps to configuring access to a MongoDB Atlas cluster from a Databricks cluster

First, you have to install the correct libraries on your Databricks cluster; one for the device drivers and another to allow Python to work with Mongo.



Name	Type	Status	Source
org.mongodb.spark:mongo-spark-co...	Mav...	-	
pymongo[srv]==3.11	PyPI	-	

To accomplish this, first click the **Install new** button, and select the **Maven** library source. Next, type in the value **org.mongodb.spark:mongo-spark-connector_2.12:3.0.1**, and then click the **Install** button.



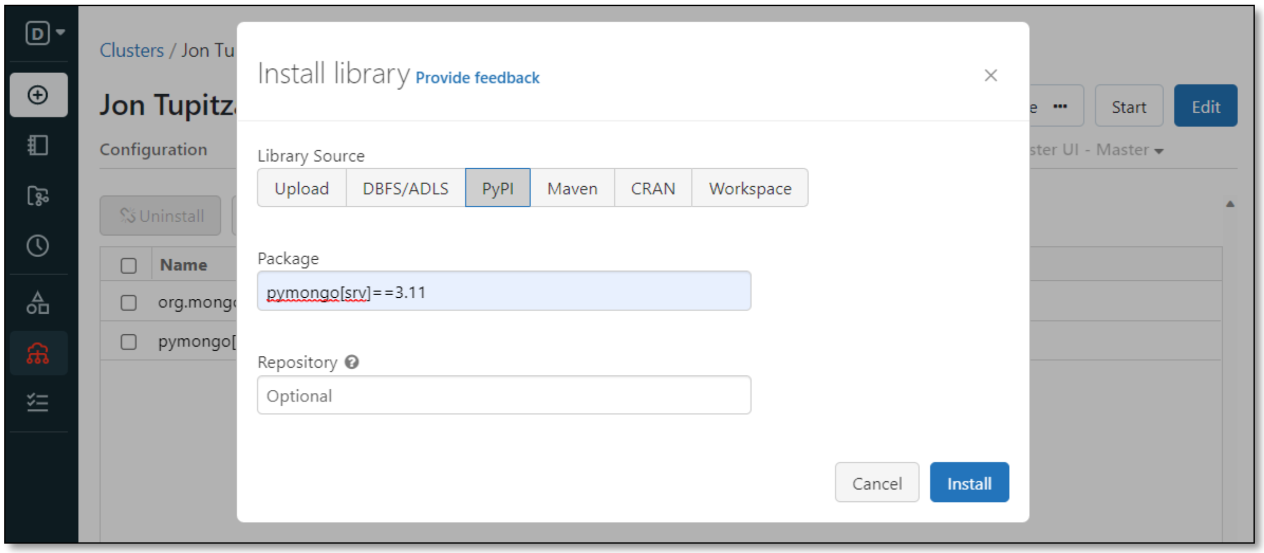
Library Source: Upload | DBFS/ADLS | PyPI | [✓ Maven] | CRAN | Workspace

Coordinates: org.mongodb.spark:mongo-spark-connector_2.12:3.0.0.

Repository: Optional

Exclusions: Dependencies to exclude (log4j:log4j,junit:junit)

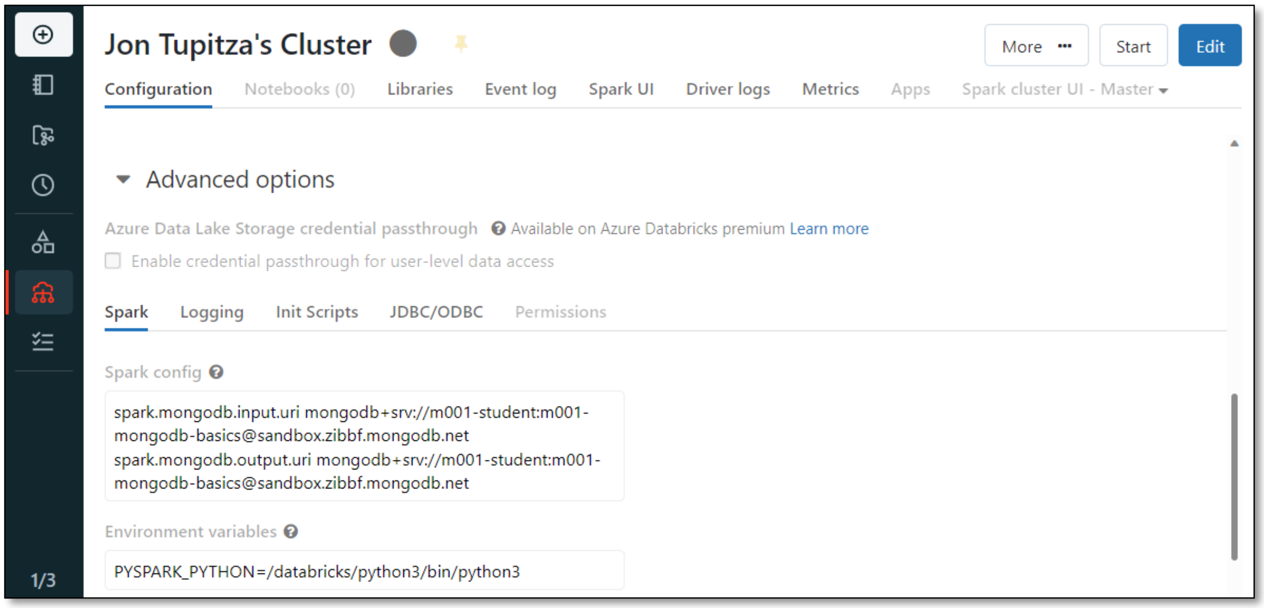
Then, repeat the process to install the pymongo library. However, this time select the **PyPI** library source, and type in the value **pymongo[srv]**.



Package: pymongo[srv]==3.11

Repository: Optional

Next, in your Databricks cluster, under **Advanced Options**, pass your mongodb connection string to both the **spark.mongodb.output.uri** and **spark.mongod.input.uri** variables. Your connection string should follow the format: **mongodb+srv://<username>:<password>@<databasename>.xxxxx.mongodb.net/**


Jon Tupitza's Cluster
More
Start
Edit
Configuration
Notebooks (0)
Libraries
Event log
Spark UI
Driver logs
Metrics
Apps
Spark cluster UI - Master
Advanced options
Azure Data Lake Storage credential passthrough
Available on Azure Databricks premium Learn more
Enable credential passthrough for user-level data access
Spark
Logging
Init Scripts
JDBC/ODBC
Permissions
1/3

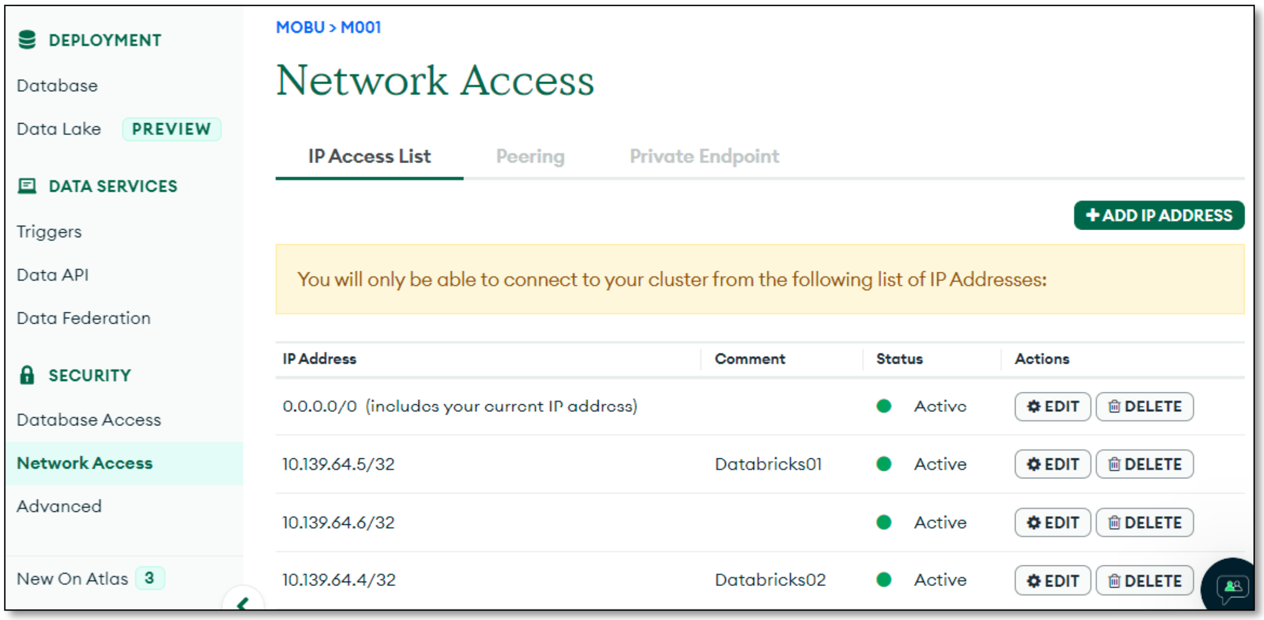
Spark config

```
spark.mongodb.input.uri mongodb+srv://m001-student:m001-
mongodb-basics@sandbox.zibbf.mongodb.net
spark.mongodb.output.uri mongodb+srv://m001-student:m001-
mongodb-basics@sandbox.zibbf.mongodb.net
```

Environment variables

```
PYSPARK_PYTHON=/databricks/python3/bin/python3
```

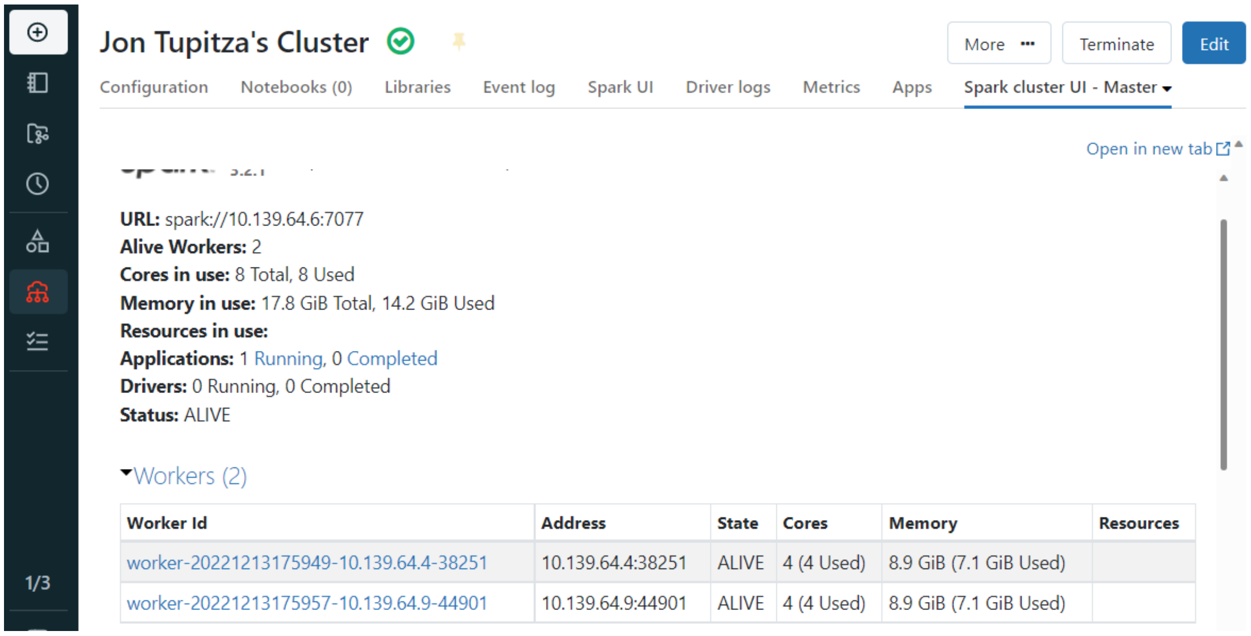
Next, add a **firewall rule** to your MongoDB Atlas cluster to allow your Databricks cluster to communicate with it. To accomplish this, log into your Atlas account, and then select the **Network Access** tab on the left-hand tool bar. Here you can click the **+ADD IP ADDRESS** button to input IP Address for your Databricks cluster.



You will only be able to connect to your cluster from the following list of IP Addresses:

IP Address	Comment	Status	Actions
0.0.0.0/0 (includes your current IP address)		● Active	EDIT DELETE
10.139.64.5/32	Databricks01	● Active	EDIT DELETE
10.139.64.6/32		● Active	EDIT DELETE
10.139.64.4/32	Databricks02	● Active	EDIT DELETE

You can find the IP Address(es) for your Databricks cluster on the **Spark cluster UI – Master** tab of your Databricks cluster. If you can't find any addresses, then you could always just create a Firewall Rule using **0.0.0.0/0.** This is not a best practice because it will allow any computer to connect to your cluster.



URL: spark://10.139.64.6:7077
Alive Workers: 2
Cores in use: 8 Total, 8 Used
Memory in use: 17.8 GiB Total, 14.2 GiB Used
Resources in use:
Applications: 1 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (2)

Worker Id	Address	State	Cores	Memory	Resources
worker-20221213175949-10.139.64.4-38251	10.139.64.4:38251	ALIVE	4 (4 Used)	8.9 GiB (7.1 GiB Used)	
worker-20221213175957-10.139.64.9-44901	10.139.64.9:44901	ALIVE	4 (4 Used)	8.9 GiB (7.1 GiB Used)	

The sample notebook contains the two function definitions that I created to write to and read from MongoDB Atlas. They both contain the same connection string, which you probably need to alter in order to successfully communicate with your MongoDB Atlas cluster. Recall the issue we had in Lab 4 where the cluster you created may have had a different **url** than the one I created.

```
######################################################################################
# Use this Function to Fetch a DataFrame from the MongoDB Atlas database server Using
PyMongo.
######################################################################################
def get_mongo_dataframe(user_id, pwd, cluster_name, db_name, collection, conditions,
projection, sort):
    '''Create a client connection to MongoDB'''
    mongo_uri =
f"mongodb+srv://{user_id}:{pwd}@{cluster_name}.zibbf.mongodb.net/{db_name}?retryWrite
s=true&w=majority"

    client = pymongo.MongoClient(mongo_uri)
```

You may need to alter the connection string as follows to make it work with your cluster:

```
mongo_uri =
f"mongodb+srv://{user_id}:{pwd}@{cluster_name}.zibbf.mongodb.net/{db_name}”
```

Remember that your cluster probably has different characters **here**. Also, it is safe to remove the trailing characters: `?retryWrites=true&w=majority`