Back to Top

Building The DiDAP Platform

Table of Contents

Prerequisite 1: Configuring Your Cloudera VM, Hive, and Sqoop

Before learning about servlets, you must know about the Hadoop Ecosystem and its basic capabilities. The Hive Lab Documentation Guide will take you through the steps required to set up your Cloudera Virtual MachineVM using VirtualBox (assuming you are using a Windows Host Machine), configuring Sqoop to transfer data between the Hive data warehouse and an Oracle XE instance. The Guide also covers major Hive Topics that may not be necessary for this setup process. It is recommended that you go through Phases 1, 2, 3, 4, and 5 from the provided guide. In Phase 3, there is no need to understand Hive queries beyond knowing how to create a database, create a Hive Table, delete a Hive Table, and run a simple SELECT query to view all rows of a Hive Table.

Prerequisite 2: Using Terminal Editors (vi)

Below, you can find some important vi commands that will allow you to edit a file quickly using the vi editor as you go along. We still recommend editing code files within the Eclipse IDE on the VM, but it shouldn't be an issue in this setup process:

- To open a file, use `vi <filename>`.
- Press "i" to enter insert mode.
- Press "Esc" to leave a mode, like insert mode.
- Type ":w" to save a file.
- Type ":q" to leave a file without saving changes.
- Type ":wq" to save and leave a file.
- Append commands like ":wq!" with an exclamation mark to force the vi command.

Prerequisite 3: Important Commands & Environment Variables

Important Linux Commands

Here are some important commands to know:

1) Exchanging files between your VM and Windows Host Machine (Refer to Phase 3 of the Hive Lab Documentation to set up a shared folder system)
   `$ sudo mount -t vboxsf <Folder Name> <VM Folder Path>`
   Or enter as root user with `su` and then run `# mount -t vboxsf <Folder Name> <VM Folder Path>`
   My Local Example: `# mount -t vboxsf ADSB_Exchange /home/cloudera`

2) Running Hadoop Commands:
   `$ hadoop fs [command with options]` or `$ hdfs dfs [command with options]`
3) `$ hadoop version` → Hadoop version output
4) `$ java -version` → JDK/JRE Java installation output
5) `$ tomcat version` → Tomcat version output
6) `$ sqoop version` → Sqoop version output
7) `$ cat /etc/passwd` → Displays all users
8) `$ cat /etc/group` → Displays all groups and associated users

Important Environment Variables & Standard Info

Here are some important Environment Variables to be familiar with while setting up Tomcat to work with the Hadoop Ecosystem present on your VM:

1) `$ echo $JAVA_HOME` → Location of your Java binary files
2) `$ echo $HIVE_HOME` → Location of your Hive config/setup files
3) `$ echo $SQOOP_HOME` → Location of your Sqoop installation from Phase 3 of the Hive Lab Documentation
4) `$ echo $HADOOP_CLASSPATH` → Location of your Hadoop installation for Sqoop to use
5) `$ echo $CATALINA_HOME` → Location of your Tomcat installation (important location for logs, config files, and webapp location), for me (`/usr/share/tomcat`)
6) `catalina.base` → Tomcat running instance or current configured location
7) `catalina.home` → Tomcat base files location, for me (`/usr/share/tomcat`)
8) `ps aux | grep catalina` → To view value of `catalina.base` & `catalina.home`
9) When using `sudo`, Hue, and other major tools, the username and password are `cloudera` and `cloudera`, by default.

Phase 1: Installing & Configuring Tomcat

Yum is a standard package manager on RHEL systems, like your Cloudera VM, which runs on CentOS, a standard RHEL Operating System. Installing Tomcat is as easy as one command using yum are some important commands to know:

Install Tomcat following this tutorial:
`sudo yum install tomcat` → Tomcat user is automatically created in /usr/share/tomcat

You may encounter an error mentioning `"database disk image malformed"` or `"no more mirrors to try"`.In that case, run the following commands. Then, repeat the command above to install tomcat with yum.

`sudo yum clean dbcache`
`sudo yum --disablerepo=epel -y update  ca-certificates`
→ Fixes yum disk error

`sudo vi /usr/share/tomcat/conf/tomcat.conf` → Open this file using the vi editor
`JAVA_OPTS="-Djava.security.egd=file:/dev/./urandom`
`-Djava.awt.headless=true -Xmx512m -XX:MaxPermSize=256m`
`-XX:+UseConcMarkSweepGC"` → Add this JAVA_OPTS Line. You may see another JAVA_OPTS line in the file. Feel free to ignore it and add the line above to the file.

`sudo yum install tomcat-webapps tomcat-admin-webapps` → Install Tomcat admin packages

`sudo yum install tomcat-docs-webapp tomcat-javadoc` → OPTIONAL installation of Tomcat documentation

`sudo vi /usr/share/tomcat/conf/tomcat-users.xml` → Open this file

Find a spot between the `<tomcat-users>` and `</tomcat-users>` xml tags and add the following line anywhere between it. You may see tons of arbitrary comments in the file but you can safely ignore them and add the lines shown below. This creates an admin user to access Tomcat host settings, only when necessary:
`<user username="admin" password="password" roles="manager-gui,admin-gui"/>`

`sudo service tomcat start` → Tomcat service must be started before being accessed

`cd /etc/profile.d` → change directories to access `tomcat.sh`
`sudo vi tomcat.sh` → add the line `#!/usr/bin/env bash` to the top of the file to `tomcat.sh`

`export CATALINA_HOME=/usr/share/tomcat` → add the export line to `tomcat.sh` to use the `$CATALINA_HOME` environment variable

Lastly, run `tomcat version` to verify your Tomcat installation.

Phase 2: Setting Up The Servlet Codebase

Understanding the Tomcat Web Application Structure

Before deploying the application it is essential to understand the file structure that Tomcat uses for hosting webapps.

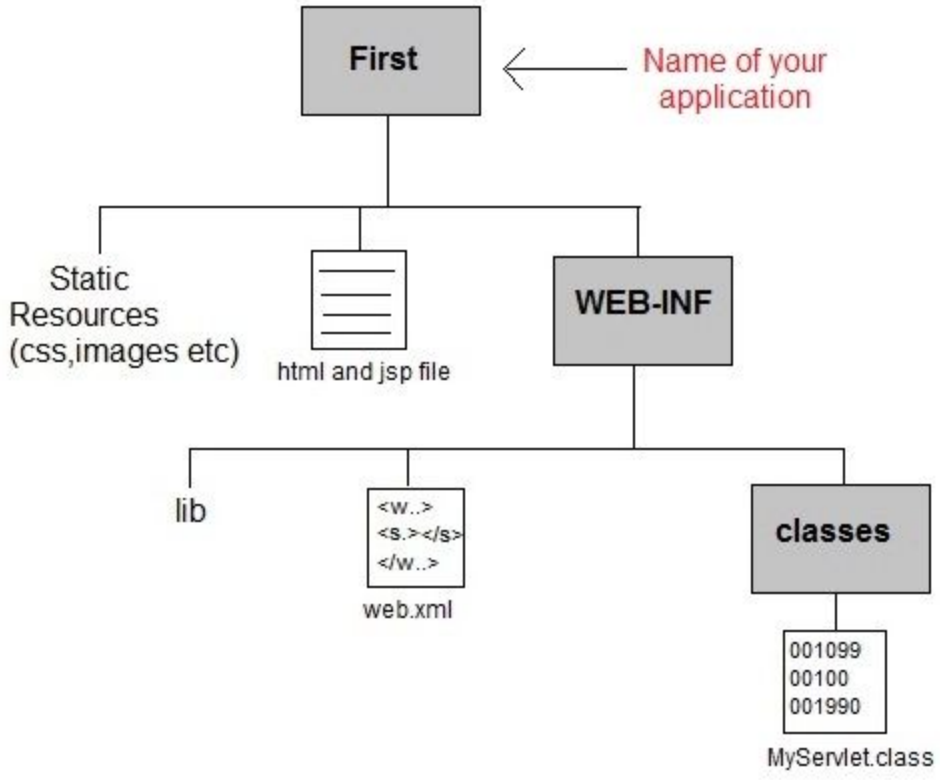


Image Source

The file structure above displays the directory structure of a Tomcat webapp by the name First located in `/usr/share/tomcat/webapps` or `$CATALINA_HOME/webapps`. All HTML and static files (Images, CSS, etc.) are kept directly under the web application's main folder. All servlet code is kept in the classes folder.

The `web.xml` Deployment Descriptor is kept under the WEB-INF (`$CATALINA_HOME/webapps/didap/WEB-INF/`) folder. This is an important file used by Java web applications to determine how URLs map to servlets and other information. To learn more about Deployment Descriptors, check this documentation.

Configuring Tomcat to Compile A Servlet & Verifying Tomcat's Installation

When installing tomcat, there should be several standard folders installed in `$CATALINA_HOME`.

1) `bin` → Tomcat Binary Files
2) `conf` → Tomcat config files
3) `lib` → Additional JAR files for Tomcat to work with programs like Hadoop, etc.

4) `logs` → Tomcat logs
5) `temp` → No need to worry about this folder
6) `webapps` → Tomcat webapps folder to host your applications
7) `work` → No need to worry about this folder

In order to compile a servlet, Tomcat requires a file called `tomcat-servlet-3.0-api.jar`. This file should be provided during your Tomcat installation, but you should verify that it is located in the `$CATALINA_HOME/lib` folder. The file name may vary depending upon the version of Tomcat that yum installs for you (it assumes the latest version), but you should be able to easily identify it. Without this file, you cannot run a servlet and should not move forward with this setup process.

Moving the DiDAP Codebase From Sakai To Your VM

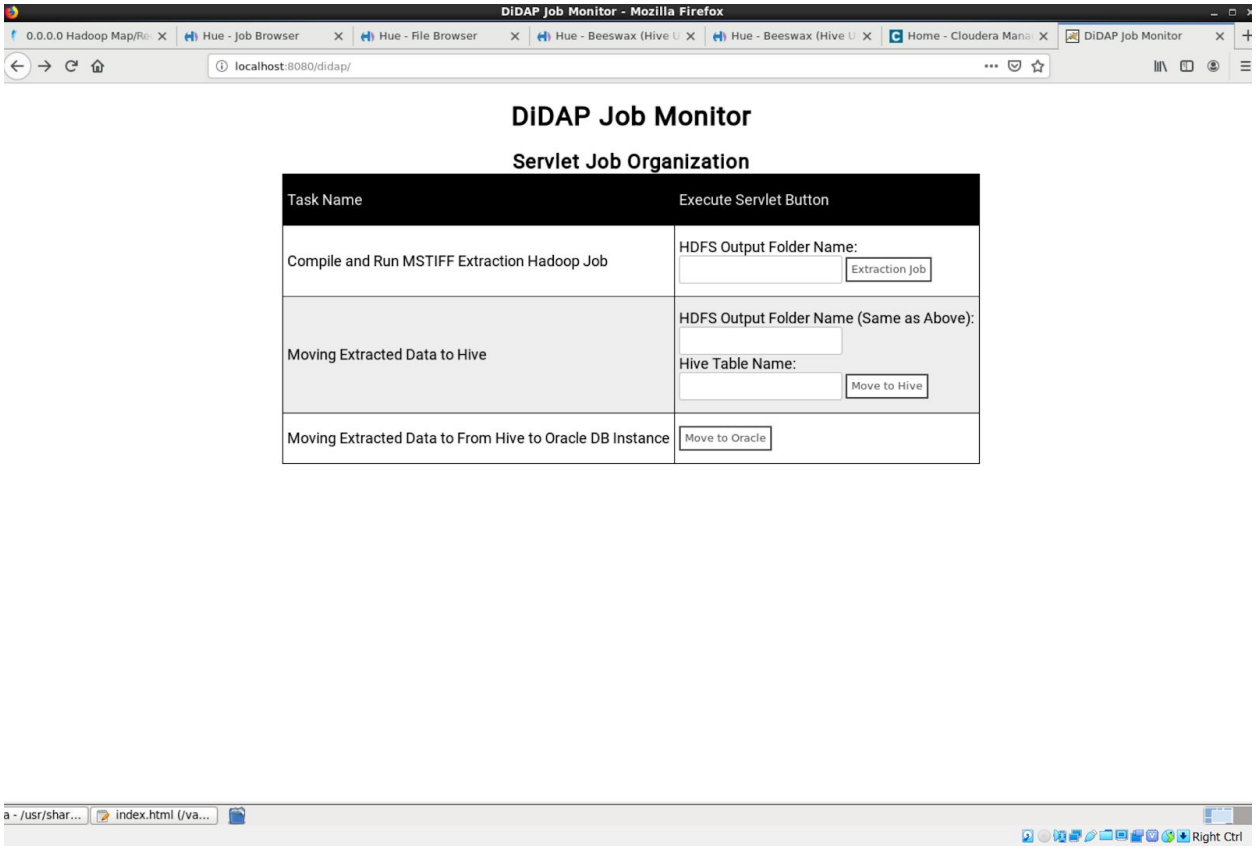
The Tomcat DiDAP Web Application Codebase is located in the ExMCM_C4I Sakai Site. Download and move the zip file located in this folder path (`2020_Tomcat_Webapp_Codebase/DiDAP_Code`) into your VM's `/usr/share/tomcat/webapps` or `$CATALINA_HOME/webapps` folder. You will need to download the zip file, extract it, and move it to your VM in the location specified above. The name of the folder will be simply "didap" once extracted.

To move files between your Host Machine and VM, refer to Phase 3 of the Hive Lab Documentation and the `mount` command at the top of this document. You can also log in to Sakai on your VM's firefox browser and download the codebase folder to your VM to get rid of this extra step. You can use this command to do so: `sudo mv <location of codebase on your VM>/didap $CATALINA_HOME/webapps`. As always, you can replace `$CATALINA_HOME/webapps` with `/usr/share/tomcat/webapps`.

The directory name of the codebase may be `webapps`, but make sure you change the folder name to didap because the name of the folder is essential to Tomcat to map your application to your URL. You can change the folder name with the following command: `sudo mv <current folder name> didap`.

To start up your local Tomcat server instance, run `sudo service tomcat start`.

Then, navigate to the following URL in Firefox on your VM: `localhost:8080/didap`. If you experience an error at this step, check back to make sure you changed your directory name from `webapp` to `didap`. Port 8080 is the default port used by Tomcat. This can be changed, but you don't need to worry about it in this process. You should see a page similar to the following:



Keep in mind that none of the buttons in the applications will do anything since we have not fully configured Tomcat to work with Hadoop as we would like.

A great first exercise is to work with a Hello World Servlet included in the codebase to get used to Java Servlet Programming. Let's go through that in the next section.

Introduction to Java Servlet Programming

First, find the file `index.html` in the web application directory for the didap project (`$CATALINA_HOME/webapps/didap`). This is the file that dictates the view of your application. Tomcat automatically knows that this file is what it needs to use to display your Java Web Application when you go to the URL: `localhost:8080/didap`.

When you open the file, you should see standard regular CSS and HTML Content. Scroll down to between the `<body>` html tags and you will find a commented out block of HTML Code for the Hello World Servlet.

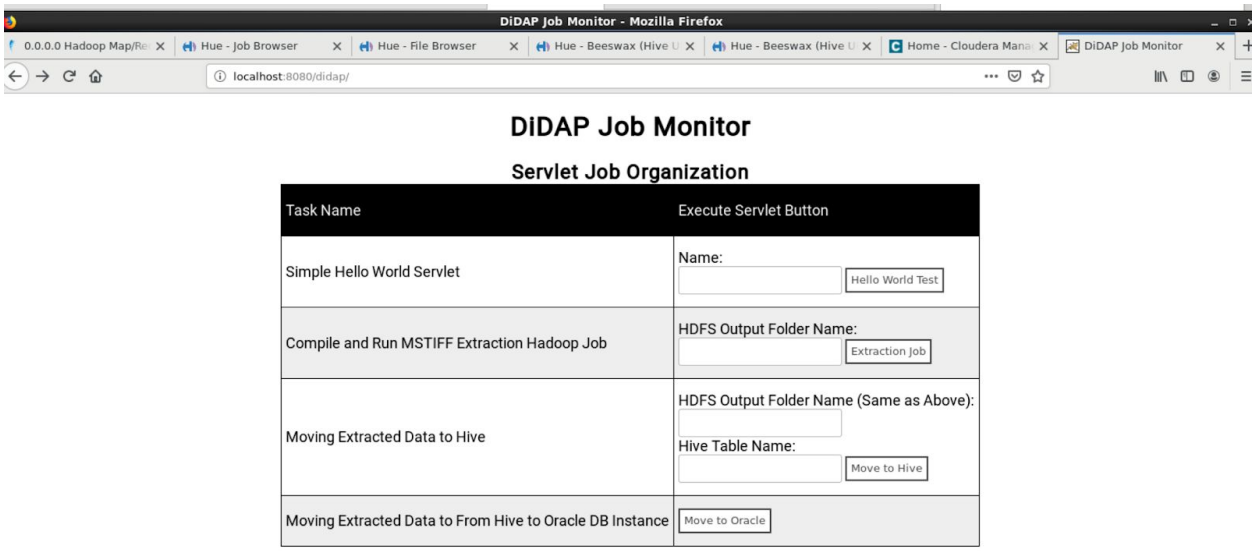
```
<body>
        <p>
        <h1 style="font-family:Roboto, sans-serif; text-align:center;">DiDAP Job Monitor</h1>

        <table class="tg" id="t01" style="margin-left:auto;margin-right:auto;">
        <caption style="font-family:Roboto, sans-serif;font-size:24px;"><b>Servlet Job Organization</b></caption>
        <thead>
          <tr>
            <th class="tg-0lax">Task Name</th>
            <th class="tg-0lax">Execute Servlet Button</th>
          </tr>
        </thead>
        <tbody>
          <!-- <tr>
            <td>Simple Hello World Servlet</td>
            <td>
              <form action="HelloWorld" method="GET">
              <label for="data">Name:</label><br>
              <input type="text" id="data" name="data">
              <input class="button" type="submit" value="Hello World Test">
              </form>
            </td>
          </tr> -->
          <tr>
            <td>Compile and Run MSTIFF Extraction Hadoop Job</td>
            <td>
              <form action="MSTIFF_Extraction" method="GET">
              <label for="data">HDFS Output Folder Name:</label><br>
              <input type="text" id="directory_name" name="directory_name">
              <input class="button" type="submit" value="Extraction Job">
              </form>
            </td>
          </tr>
```

Next, uncomment the code block by removing the `<!--` and `-->` tags. Your code should look like this:

```
            <tr>
              <th class="tg-0lax">Task Name</th>
              <th class="tg-0lax">Execute Servlet Button</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td>Simple Hello World Servlet</td>
              <td>
                <form action="HelloWorld" method="GET">
                <label for="data">Name:</label><br>
                <input type="text" id="data" name="data">
                <input class="button" type="submit" value="Hello World Test">
                </form>
              </td>
            </tr>
            <tr>
              <td>Compile and Run MSTIFF Extraction Hadoop Job</td>
              <td>
                <form action="MSTIFF_Extraction" method="GET">
                <label for="data">HDFS Output Folder Name:</label><br>
                <input type="text" id="directory_name" name="directory_name">
                <input class="button" type="submit" value="Extraction Job">
                </form>
              </td>
            </tr>
```

When you refresh your webpage, you should see a new UI like below:



Now, let's dive deeper into the `web.xml` Deployment Descriptor File.

```
<web-app>
  <servlet>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>HelloWorld</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>MSTIFF_Extraction</servlet-name>
    <servlet-class>MstiffExtraction</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>MetadataTableCreation</servlet-name>
    <servlet-class>MetadataTableCreation</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>OracleDataTransfer</servlet-name>
    <servlet-class>OracleDataTransfer</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/HelloWorld</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>MSTIFF_Extraction</servlet-name>
    <url-pattern>/MSTIFF_Extraction</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>MetadataTableCreation</servlet-name>
    <url-pattern>/MetadataTableCreation</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>OracleDataTransfer</servlet-name>
    <url-pattern>/OracleDataTransfer</url-pattern>
  </servlet-mapping>
</web-app>
```

Deployment Descriptors help describe the classes, resources and configuration of the application and how the web server uses them to serve web requests. When the web server receives a request for the application, it uses the deployment descriptor to map the URL of the request to the code that handles the request.

The `<servlet>` tag enables you to map a certain Java class file (between the `<servlet-class>` tags) from the `$CATALINA_HOME/webapps/didap/WEB-INF/classes` folder to a `<servlet-name>` of your choice. The `<servlet-mapping>` tag allows you to map a `<servlet-name>` to a specific `<url-pattern>`. The `<url-pattern>` value is important because it can be used to access each page in our application.

When you go back to the `index.html` file, you may notice that the `form` tag for the HelloWorld Servlet Code Block has an `action` pointing to `HelloWorld`. This means that every time we click the Hello World Servlet Button, it will run the servlet with the name of HelloWorld (which points to the Java classname of 'HelloWorld'). You can test this out by entering a name of your choice into the text input box and clicking the 'Hello World' button. You should see an output saying: `Hello, World!, I'm [the name entered in the text box]`. You can easily navigate back to the homepage of the application by simply clicking the back arrow in the VM's browser.

To view the raw Java code for the Servlet, navigate to the classes folder of the web application (`$CATALINA_HOME/webapps/didap/WEB-INF/classes`). You should see a file by the name of `HelloWorld.java`. This file contains embedded HTML in Java that produces the output you saw earlier. The code is well-commented so you can take a look through the `HelloWorld.java` file to understand its function. It is recommended to open `.java` files in the Eclipse IDE, which comes pre-installed on the VM.

One important part is understanding how the servlet takes user input from the input boxes. When you look at the HelloWorld Servlet Code Block in `index.html`, you will notice a line: `<input type="text" id="data" name="data">`. The HTML Attributes of `id` and `name` specify a unique identifier to the input box. When the servlet is executed, a simple expression in `HelloWorld.java` retrieves user input by using the `id` and `name` attribute values used to refer to the input box: `request.getParameter("data")`.

You may also notice that every `.java` file in the classes folder has a corresponding `.class` file. The `.class` files are important since they are compiled versions of the `.java` files that Tomcat uses to run your web application. **This means that everytime you**

change raw Java servlet code or anything past inside the `WEB-INF` folder, you must restart Tomcat with `sudo service tomcat restart` to display your changes.

Other Resources to Understand Servlets

1) https://www.geeksforgeeks.org/introduction-java-servlets/
2) https://www.geeksforgeeks.org/starting-first-servlet-application/
3) https://www.infoworld.com/article/3510460/what-is-apache-tomcat-the-original-java-servlet-container.html#:~:text=Apache%20Tomcat%20is%20a%20long,four%20years%20after%20Java%20itself.

Phase 3: Connecting Tomcat with Hadoop to Run Our Code

Managing the MSTIFF Extraction Source Code

The MSTIFF Extraction Servlet requires some source code which is located in this folder path (`2020_Tomcat_Webapp_Codebase/MSTIFF_Code.zip`) in the ExMCM_C4I Sakai Site. Download and move the folder into your VM's `/home/cloudera` directory. You will need to download the zip file, extract it, and move it to your VM in the location specified above. The folder will be named simply "MSTIFF_Code" once extracted.

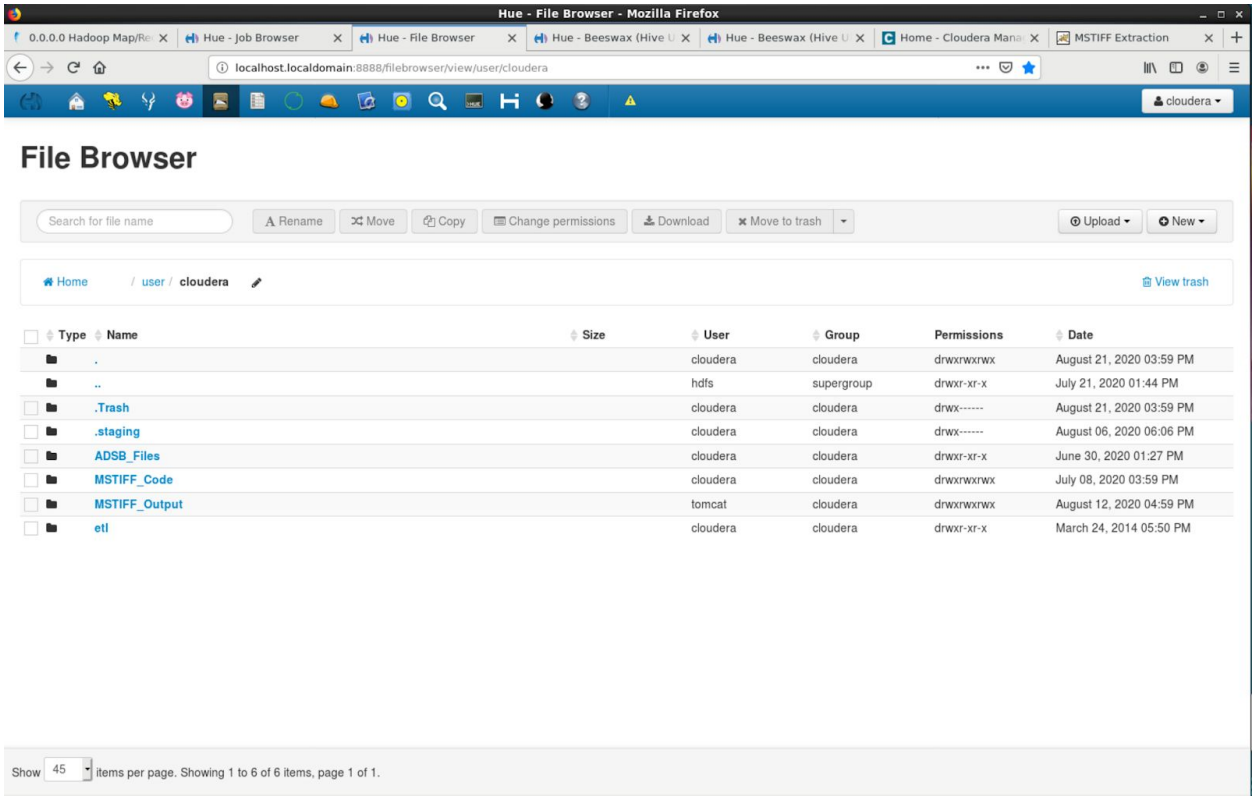
The folder must be in this location because the servlet code and scripts look at that exact path to compile and run the Hadoop MSTIFF Data Extraction Job.

Adding the MSTIFF Input Files to HDFS

The MSTIFF Extraction Servlet also requires some MSTIFF input files which are located in this folder (`2020_Tomcat_Webapp_Codebase/MSTIFF_Code.zip`) that you just moved to your VM's `/home/cloudera` directory. You will need to now copy the same "MSTIFF_Code" folder on your VM to HDFS at the following location (`/user/cloudera`).

You can do so with the following command: `hadoop fs -put /home/cloudera/MSTIFF_Code /user/cloudera`

Then, verify in your HDFS File structure, which is viewable in Hue (described in Phase 4 of the Hive Lab Documentation), that you can see the "MSTIFF_ Code Folder" as shown below:



Now, let's move forward!

Setting Tomcat User Configurations

In order for Tomcat to access local files, we must add the tomcat user to the cloudera user group. The cloudera user group is one that exists on your current VM configuration. The tomcat user is a semi-user created when Tomcat is installed.

To view users on your current system, run: `cat /etc/passwd`.
To view groups and associated users, run: `cat /etc/group`.
To view the cloudera group and its associated users run: `cat /etc/group | grep cloudera`.

We want to add the tomcat user to the cloudera group. You can do so by running:
`sudo usermod -a -G cloudera tomcat`
`sudo usermod -a -G cloudera cloudera` → This adds the cloudera user to the cloudera group just for security.

To verify that this worked, run: `cat /etc/group | grep cloudera` to view the cloudera group and its associated users.

Setting Up Tomcat to Work with Hadoop & Hive.

In order for Tomcat to run the MSTIFF Extraction Source Code in Hadoop, it needs to be designated as an HDFS supergroup user (1). We must also revise our `.bashrc` file (2) and install the proper drivers to access Hive (3) for some of our servlets. Then, the entire `$CATALINA_HOME` (`/usr/share/tomcat`) directory must be set with the proper user & group permissions (4). Follow the instructions below to do so:

(1) Superuser Setup:

1) Navigate to the `/etc/hadoop/conf` directory with: `cd /etc/hadoop/conf`. This directory contains the standard Hadoop configuration files.
2) Open `hdfs-site.xml` with: `sudo vi hdfs-site.xml`.
3) Add the following lines to the file between the `<configuration>` and `</configuration>` tags:

```
<property>
    <name>dfs.permissions.superusergroup</name>
    <value>tomcat</value>
</property>
```

4) Create a new user group called `supergroup` with: `sudo groupadd supergroup`.
5) Then, add the tomcat user to the `supergroup` with: `sudo usermod -a -G supergroup tomcat`.

(2) Bashrc File Setup:

1) Tomcat requires you to modify your CLASSPATH environment variable so it can actually execute your servlets. You want to get all lines of the “sample_bashrc” located in this folder path (`2020_Tomcat_Webapp_Codebase/sample_bashrc`) in the ExMCM_C4I Sakai Site to be identical to your VM’s .bashrc file. You will need to click on the “sample_bashrc” file in Sakai to view its contents.
2) Then, copy it into your `~/.bashrc` file with the following commands and actions:
    - `sudo vi ~/.bashrc`
    - Type “50dd” to delete existing file contents. This makes the file empty.
    - Press “i” on your keyboard to enter insert mode.
    - Paste the contents you just copied from the Sakai file.
    - Press “Esc”.
    - Type “:wq” to save and exit the file
    - `source ~/.bashrc` to sync changes.
3) You can view the file from now on with: `vi ~/.bashrc`. It is a standard Linux file which runs automatically when you start a shell session.
4) Now, let’s go over what that file contains:
    - The part that we are concerned with begins with `export HIVE_HOME ...`

i) We are adding the location of our HIVE and HADOOP installations to our PATH, so our VM can recognize them whenever we are working with Hive or Hadoop in our servlets.

- The next section focuses on all the `export CLASSPATH` statements.
  i) These add the necessary `.jar` files we need for Hive, Hadoop, our servlets, and the Hive Drivers that we will install soon to work together.
- The next section encompasses the remainder of the file.
  i) These are standard Sqoop export lines that you added when installing Sqoop in Phase 5 of the Hive Lab Documentation. No need to delve deep into this for now.

(3) Adding the Necessary Hive Drivers:

The MSTIFF Extraction Servlet requires some source code which is located in this folder path (`2020_Tomcat_Webapp_Codebase/JDBC_Drivers.zip`) in the ExMCM_C4I Sakai Site. Download and move the folder into your VM's `/home/cloudera` directory. You will need to download the zip file, extract it, and move it to your VM in the location specified above. The folder name will simply be "jdbc" once extracted.

Then, move the contents of the folder into the `$CATALINA_HOME/lib` directory with the following command: `sudo mv /home/cloudera/jdbc/* $CATALINA_HOME/lib`.

The folder must be in this location or else one of our servlets (MetadataTableCreation Servlet) will not work.

(4) Tomcat User/Cloudera Group Setup:

1) Navigate to the `/usr/share` directory with: `cd /usr/share/`.
2) Run the following commands (One-By-One) in succession to reset all permissions for tomcat to also be used by the cloudera group (you may receive warnings or errors with these commands, but you can safely ignore all of them):

```
sudo chown tomcat:cloudera tomcat
sudo chown tomcat:cloudera tomcat/*

cd tomcat

sudo chown tomcat:cloudera bin/*

sudo chown tomcat:cloudera /etc/tomcat/
sudo chown tomcat:cloudera /etc/tomcat/*
```

```
sudo chown tomcat:cloudera /usr/share/java/tomcat/
sudo chown tomcat:cloudera /usr/share/java/tomcat/*
sudo chown tomcat:cloudera /usr/share/java/ecj.jar

cd lib/
sudo chown tomcat:cloudera /usr/share/java/tomcat-el-2.2-api.jar
sudo chown tomcat:cloudera /usr/share/java/tomcat-jsp-2.2-api.jar
sudo chown tomcat:cloudera /usr/share/tomcat/bin/tomcat-juli.jar
sudo chown tomcat:cloudera /usr/share/java/tomcat-servlet-3.0-api.jar

cd ..
sudo chown tomcat:cloudera /var/log/tomcat/
sudo chown tomcat:cloudera /var/log/tomcat/*

sudo chown tomcat:cloudera /var/cache/tomcat/temp/

sudo chown tomcat:cloudera /var/lib/tomcat/webapps/
sudo chown tomcat:cloudera /var/lib/tomcat/webapps/*

sudo chown tomcat:cloudera /var/cache/tomcat/work
sudo chown tomcat:cloudera /var/cache/tomcat/work/*

cd webapps/
sudo chown tomcat:cloudera didap/*

cd didap/
sudo chown tomcat:cloudera WEB-INF/*

cd WEB-INF/
sudo chown tomcat:cloudera classes/*
sudo chown tomcat:cloudera scripts/*
```

Now, all permissions are set for Tomcat to run the MSTIFF Extraction Servlet. But let's look into how the MSTIFF Extraction Servlet runs.

Phase 4: Running Our Servlets

Running the MSTIFF Extraction Servlet

From our work with the HelloWorld servlet, you should be familiar with the Tomcat servlet file structure. The MSTIFF Extraction Servlet has several moving parts: its HTML Block in `index.html`, its XML block in `web.xml`, the Java servlet code in `MstiffExtraction.java`, and the script that is called by the servlet (`mstiff.sh`) located in `$CATALINA_HOME/webapps/didap/WEB-INF/scripts`.

Let's take a look at the file `mstiff.sh`. The first two commands in the file (`java -cp … -d …` & `jar -cvf`) compile the code at the provided path, which we set above for the source code: `/home/cloudera/MSTIFF_Code`.

Additionally, the `java -cp … -d …` command provides a path to hadoop on the VM (`/usr/lib/hadoop/*:/usr/lib/hadoop/client-0.20/*`), a directory to compile all class files to (`/home/cloudera/MSTIFF_Code`), and the main code file(`/home/cloudera/MSTIFF_Code/GET_MST_mod.java`).

The `jar -cvf` command allows you to package the compiled class files into one executable `.jar` file. The first parameter (`GET_MST_mod.jar`) specifies the name and location of the `.jar` file, which is the current directory that the shell script runs in (`$CATALINA_HOME/webapps/didap/WEB-INF/scripts`). The second parameter (`-C /home/cloudera/MSTIFF_Code/ .`) signifies the directory in which the compiled class files that are to be packaged are located. Keep in mind that the period at the end of the command is important or else the script will fail.

The `hadoop jar` command allows you to run the newly created executable `.jar` file. The first parameter (`GET_MST_mod.jar`) specifies the location of the `.jar` file, which is the current directory that the shell script runs in (`$CATALINA_HOME/webapps/didap/WEB-INF/scripts`). The second parameter (`GET_MST_mod`) helps identify the main class to be run. The third parameter (`/user/cloudera/MSTIFF_Code /user/cloudera/$1`) specifies the HDFS output path, which is explained below.

Navigate to the `MstiffExtraction.java` file. In the `doGet()` method, you will see a code block like the one below:

```
String[] command = {"/bin/bash",
"/usr/share/tomcat/webapps/didap/WEB-INF/scripts/mstiff.sh",
request.getParameter("directory_name")};
```

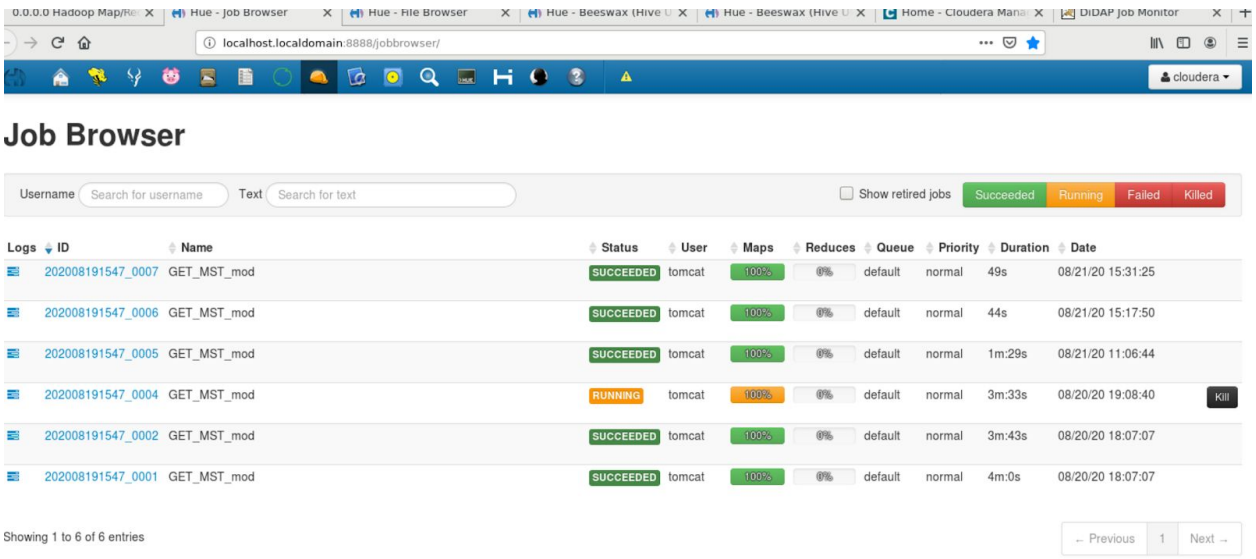
```
ProcessBuilder p = new ProcessBuilder(command);
Process p2 = p.start();
try {
  p2.waitFor();
} catch (InterruptedException e) {
  out.println("Did not execute in hadoop!");
  e.printStackTrace();
}
```

This code block executes the script using a simple bash command separated into individual array elements. This is essential to understanding how the Mstiff Extraction Servlet works. The third element of the command array (`request.getParameter("directory_name")`) specifies the HDFS output path name which is passed a bash script variable (`$1`) in `mstiff.sh`.

Now, let's run and test the servlet out. Go to the Didap Homepage in your VM's Firefox browser.

The text input box asks for an output folder that will be created in HDFS with the `.csv` output of the fields we would like to extract. This input system to the servlet is similar to the one described in the HelloWorld Servlet. Keep in mind that you will need to do some garbage keeping at some point when you keep testing the servlet and cluttering your home folder in HDFS (HDFS home folder filepath: `/user/cloudera`). You can delete old output folders and manage HDFS directory structure with Hue (described in Phase 4 of the Hive Lab Documentation). The URL to access HDFS on Hue on your VM is: `localhost.localdomain:8888/filebrowser/`.

When you run the servlet it will take some time to fully execute the Hadoop job in the background (between 2-5 minutes). If the job finishes too quickly, you should look back in this document to find your setup error. You can monitor the Hadoop job status by looking at the Job Status window in Hue by clicking the Orange Construction Cap Logo in the navigation bar at the top or going to this URL: `localhost.localdomain:8888/jobbrowser/`.



You may see a running job or successful job like the picture above depending on whether or not the job is finished. If it says “FAIL”, you should look back at your steps beginning at the top of Phase 3.

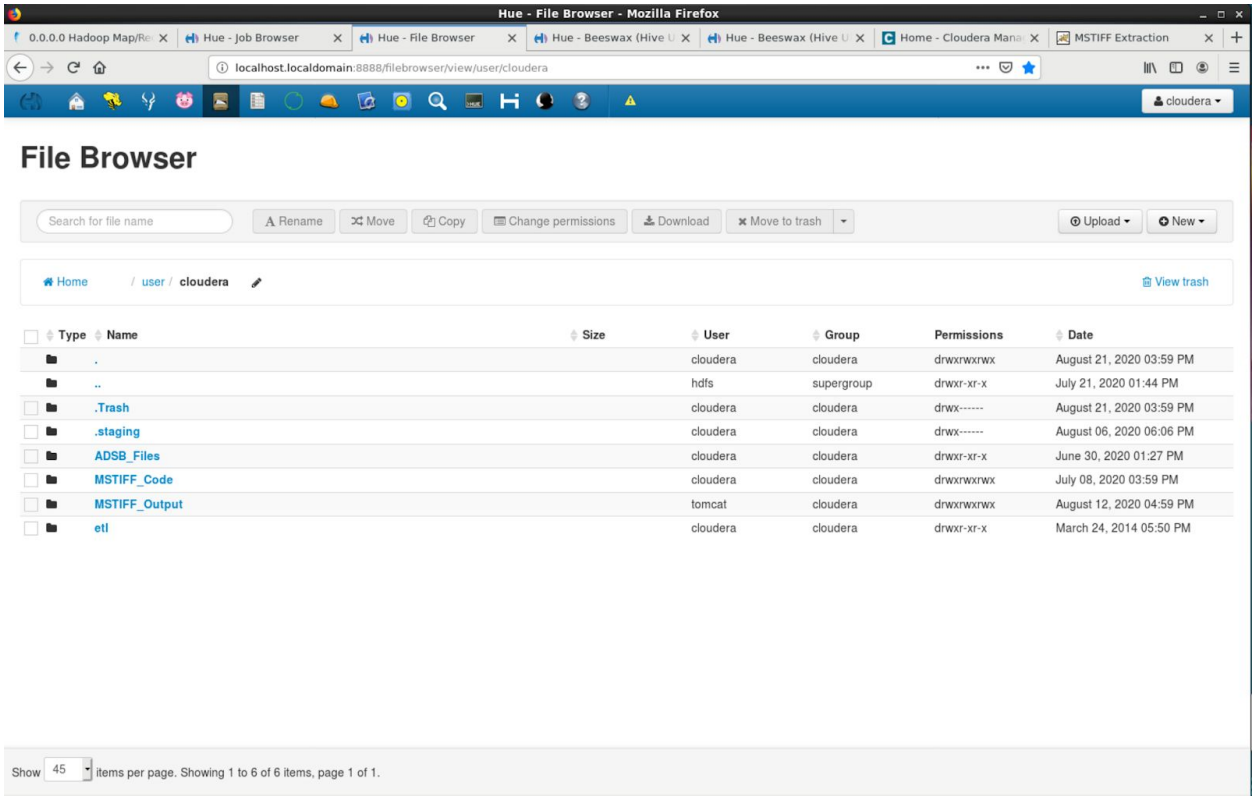
Once the job finishes, you should see a lengthy output in your Firefox browser from the bash script displayed due the following code block in `MstiffExtraction.java`:

```
BufferedReader br = new BufferedReader(new
InputStreamReader(p2.getInputStream()));
String line;

out.println("Output of running " + Arrays.toString(command) + " is: ");

while ((line = br.readLine()) != null) {
  out.println(line + "\n");
}
```

The displayed output also includes a “Job Status”, with a 0 for success and 1 for failure.



You should then verify that the output folder was created in HDFS by looking in Hue My output folder name is “MSTIFF_Output”. When you click on the folder, you should see a file named “SUCCESS” and another file named “part-m-00000”, which is the proper CSV output file used in the following servlet. If the output folder doesn’t contain these file, review Phase 4 again.

Running the Metadata Table Creation Servlet

Before running the Metadata Table Creation Servlet, let’s do some more setup operations. First off, before loading our data into a Hive table, we need to create the Hive table ourselves. This may be done by a DB Administrator in the future.

Let’s begin by entering Hive from our terminal with the command: `hive`. You should then wait until the hive terminal loads up.

In Hive, there are databases and tables associated with each database. In order to see what databases are in your Hive platform, simply issue the following command:

```
hive> show databases;
```

You should already see the databases titled “default” and “tutorial”. In our case, we will need to create our own Hive database and create our tables within that database. To create our “metadata” database, issue the following command:

```
hive> CREATE DATABASE metadata;
```

After creating the database, issue the `show databases;` command again in Hive. You should see an output similar to this:

```
OK
default
metadata
tutorial
Time taken: 4.7 seconds
```

Now that you have created our database within Hive, you need to specify which database you want to use before modifying the database by adding tables, etc. Issue the following command to use a database:

```
hive> USE  metadata;
```

Now, you want to create a table in your “metadata” database.

You can look in our Sakai folder (`2020_Tomcat_Webapp_Codebase/CreateMetadataHiveTable.q`) and copy the HiveQL query you see below to create a table named “florida” in the “metadata” database in Hive:

```
CREATE TABLE IF NOT EXISTS Metadata.florida
(filePath STRING, dateStamp INT,navTime STRING,FathTime STRING,latitude
DOUBLE,longitude DOUBLE,waterDepth FLOAT,towfishDepth FLOAT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/hive/warehouse/metadata.db';
```

You can also run HiveQL Queries in Hue’s Beeswax Hive UI Tool (Yellow Elephant Bee Figure in the top of your Hue) at this URL in your VM’s Firefox Browser: `http://localhost.localdomain:8888/beeswax/`. On the left part of your screen, select the database to be “metadata”. Then, run the query you just copied and you should see all rows of inserted data after approximately 10-20 seconds.

You should see an output “OK” once the query executes with no errors. The query is pretty straightforward and has columns with proper datatypes based on the output from our MstiffExtraction Servlet.

You can then run `hive> show tables;` in your terminal window to view the table that you just created in the “metadata” database.

Lastly, run `hive> exit;` to return to your normal terminal.

Now, we need to start our hive server, so that our servlet can connect to it. You can do so with the following command: `hive --service hiveserver &`. It should output something similar to the following: `[1] (random process ID here)` and `Starting Hive Thrift Server`.

Then, hit enter to see the normal terminal prompt. Run `ps` to verify that the hive service is running (on its default port of 10000). You should see a process listed under the name of “java” and associated with the random process ID number you saw just a few moments ago.

Keep this window open since it will throw specific output while the MetadataTableCreation Servlet is running. Open a new terminal window for your commands from now on.

Next, let’s go over the MetadataTableCreation Servlet code before testing it out.

The file in `$CATALINA_HOME/webapps/didap/WEB-INF/classes/DBMain.java` contains the Hive Driver connection code. There are four main methods that you can look through (they are well-commented). The first method `DBinit()` creates and maintains the connection to the Hive Server in the following code block:

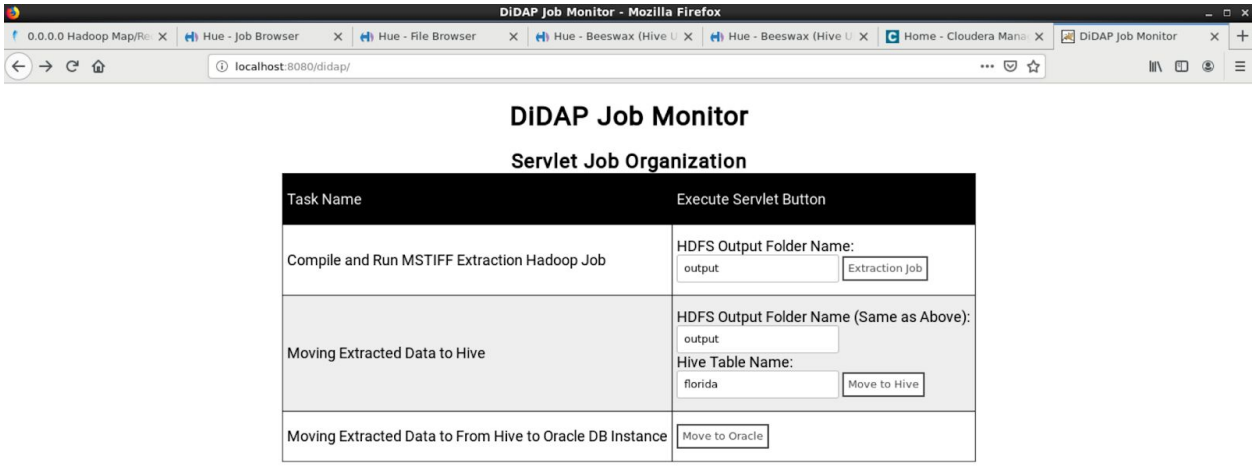
```
18
19      /**
20         * This method is used to establish the DB connection in Hive
21         * and handles the appropriate exception when an error occurs.
22         */
23      public static void DBinit(PrintWriter p) {
24              out = p;
25              if (!connEstablished) {
26                      try {
27                              // Register driver and create driver instance
28                              String driverName = "com.cloudera.hive.jdbc4.HS1Driver";
29                              Class.forName(driverName);
30                      } catch (ClassNotFoundException e) {
31                              e.printStackTrace(out);
32                              System.exit(1);
33                      }
34
35                      try {
36                              // Establish connection
37                              String url = "jdbc:hive://localhost:10000/metadata"; // metadata is the database name within local Hive instance
38                              conn = DriverManager.getConnection(url, "", ""); // no authentication needed for Hive Server 1 access
39                              out.println("Connected to Database!");
40
41                              // Create statement
42                              stmt = conn.createStatement();
43                              connEstablished = true;
44                      } catch (SQLException e) {
45                              e.printStackTrace(out);
46                              System.exit(1);
47                      }
48              }
49      }
```

This connects to the “metadata” database we created using the DriverManager url provided as a String. The method DBCreateTable() is commented out since that is generally something a DB Admin will do just as we did with our Hive script. This operation should generally not be executed by the servlet.

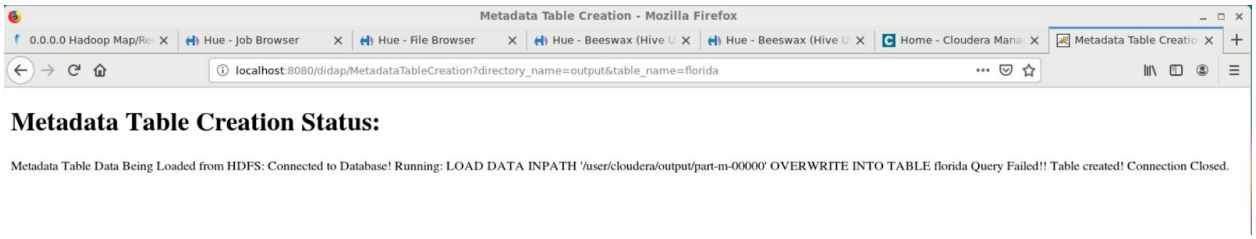
The actual servlet code is located in `$CATALINA_HOME/webapps/didap/WEB-INF/classes/MetadataTableCreation.java`. Again, the code is well-documented so feel free to take a look through it. The real job occurs when the `DBMain.DBExecuteQuery()` method is called. It loads the data into the Hive table we created with two user inputs: **the HDFS output folder of the MstiffExtraction Servlet and the Hive table name.**

Now, you can try the servlet out on your own. Go to the Didap Homepage in your VM’s Firefox browser. Enter the two inputs as instructed in the previous paragraph. You should see an output like the one below. See the 2 images:

Input:



Output:

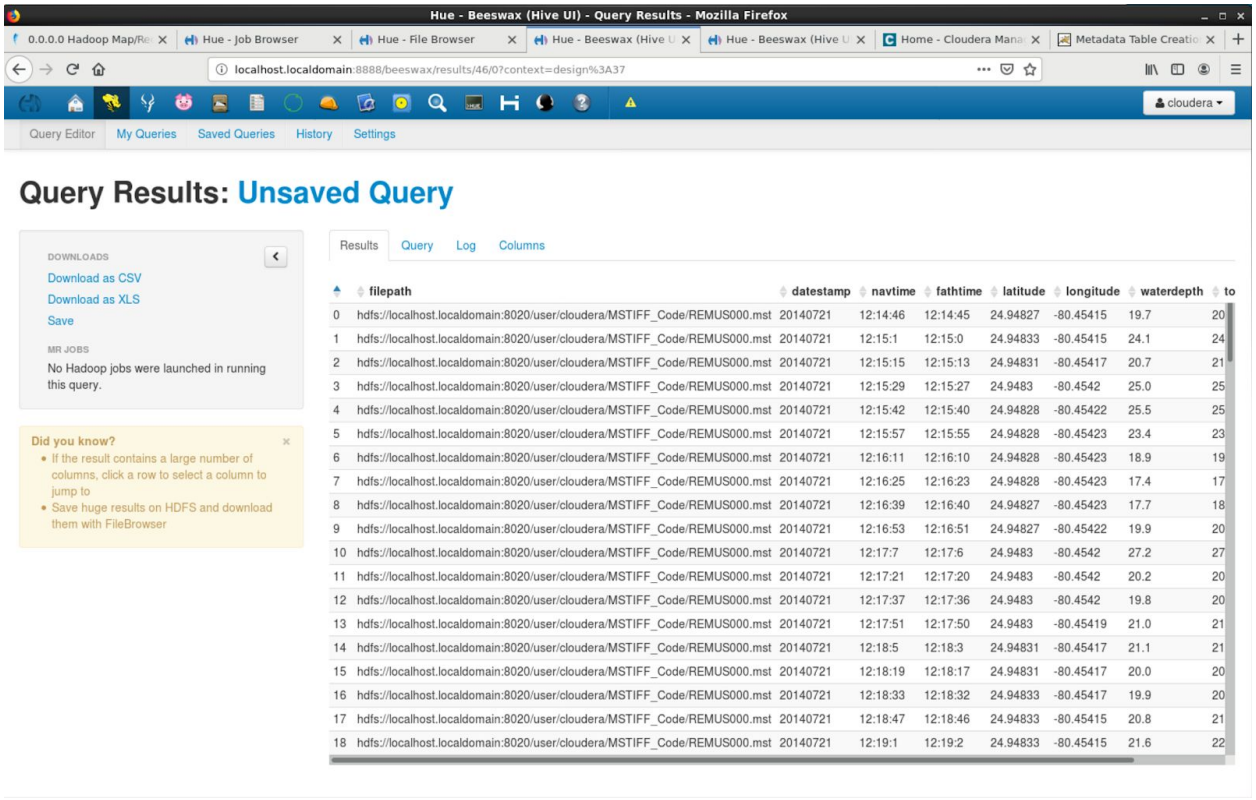


You should see a warning (`Permission denied`)in your hive server terminal window that you saved like the one below. Keep in mind this is a warning, but not an error. The servlet should have carried out its execution in about 10-15 seconds.

```
[cloudera@localhost ~]$
[cloudera@localhost ~]$ Hive history file=/tmp/cloudera/hive_job_log_dcc63efa-17
c5-4646-af98-96e3c5f727c2_330098781.txt
OK
Hive history file=/tmp/cloudera/hive_job_log_3aee128b-1fff-4f0d-b632-4bfc05ed915
d_1614057738.txt
OK
Loading data to table metadata.florida
chgrp: changing ownership of '/user/hive/warehouse/metadata.db/part-m-00000': Pe
rmission denied
chmod: changing permissions of '/user/hive/warehouse/metadata.db/part-m-00000':
Permission denied
Table metadata.florida stats: [num_partitions: 0, num_files: 0, num_rows: 0, tot
al_size: 0, raw_data_size: 0]
OK
```

You can verify that your data has been loaded by:

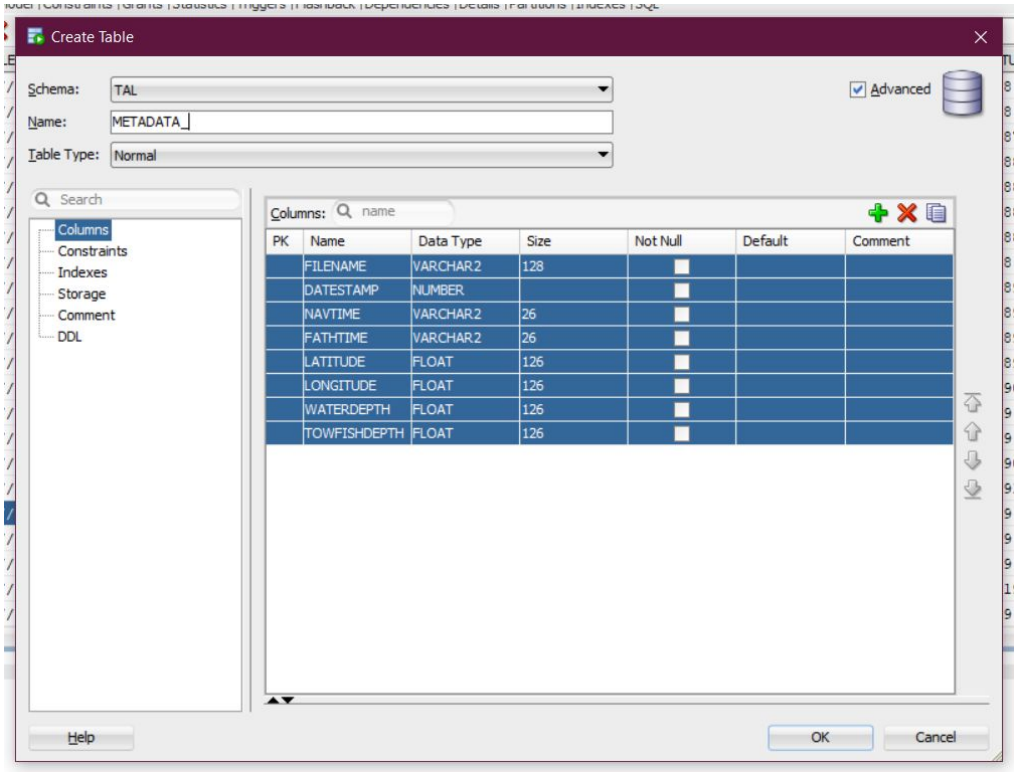
1) Entering the hive CLI with the `hive`, running `hive> use metadata;`, and running `hive> select * from florida;`.
2) You can also run a HiveQL Query in Hue's Beeswax Hive UI Tool (Yellow Elephant Bee Figure in the top of your Hue) at this URL in your VM's Firefox Browser: `http://localhost.localdomain:8888/beeswax/`. On the left part of your screen, select the database to be "metadata". Then, run the same `select * from florida;` query and you should see all rows of inserted data after approximately 10-20 seconds.



Great! Now, you have two servlets up and running. Let's take a look at the last servlet.

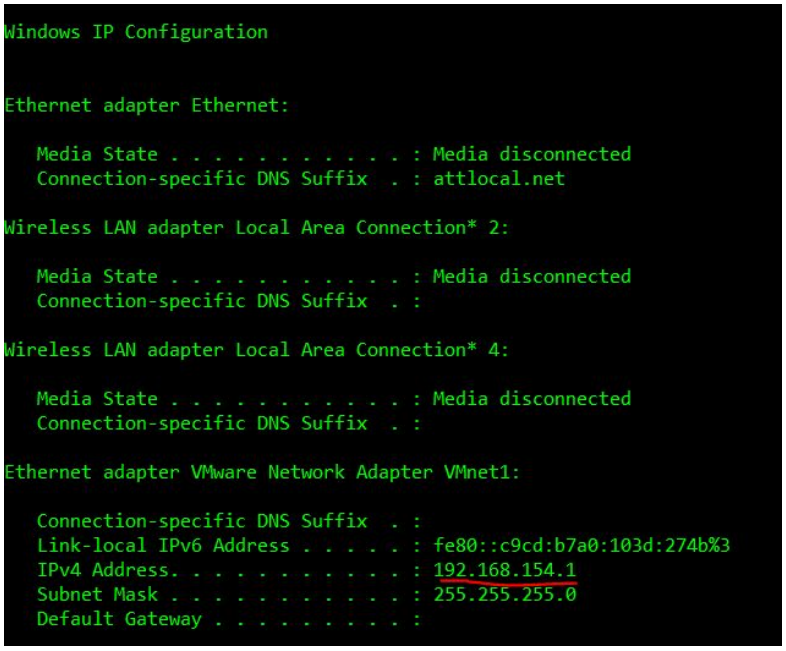
Setting Up Tomcat and Running our Sqoop Data Transfer Servlet

Before setting up the Sqoop Data Transfer Servlet, we will need to do a little bit of set up on the local host machine. Open SQL developer and navigate to the database connection where you want the data to be transferred. Right click on the Tables folder and create a new table, naming it Metadata or a name of your choosing. Add columns to this table so that it matches your Hive database:



If you are unfamiliar with using SQL Developer, you can simply run the script located at this folder path (`2020_Tomcat_Webapp_Codebase/CreateMetadataOracleTable.sql`) in the ExMCM_C4I Sakai Site. The script can be run by copy and pasting in your SQL Developer application window, then pressing the run button (green play button). It is important that the table you create is named “Metadata” because that is how the bash script on the VM recognizes the Oracle table and moves the data into Hive. Also, make sure to add your existing user’s username in front of the table name in the SQL Script in the format: CREATE TABLE username.Metadata. We have left a space for you to so do so in the file in Sakai.

Run `'ipconfig'` on your local Windows Host Machine command line to verify and update IPv4 address before this job runs or it will fail. Now, you should see an IP address listed for the connection with the Virtual Machine. The output should look similar to the following:



Copy the underlined IP address and navigate to the `OracleSqoopTransfer.sh` file in the Tomcat folder of the VM. Under the params function in this file, you should see various flags. In the `--connect` line, replace the given IP address with the IP address you just copied. Similarly, replace all instances of the given username and password with your own username and password for the Oracle XE database, as well as table name, which you can access from SQL developer on your local machine. If the table name in the file does not match the table name you have created in Hive, update that as well and save the file.

You should now be able to successfully run the `Move to Oracle` servlet! Check that the job has been completed by viewing the table in SQL developer.

Phase 5: Setting up the Jupyter Notebook Visualization Platform

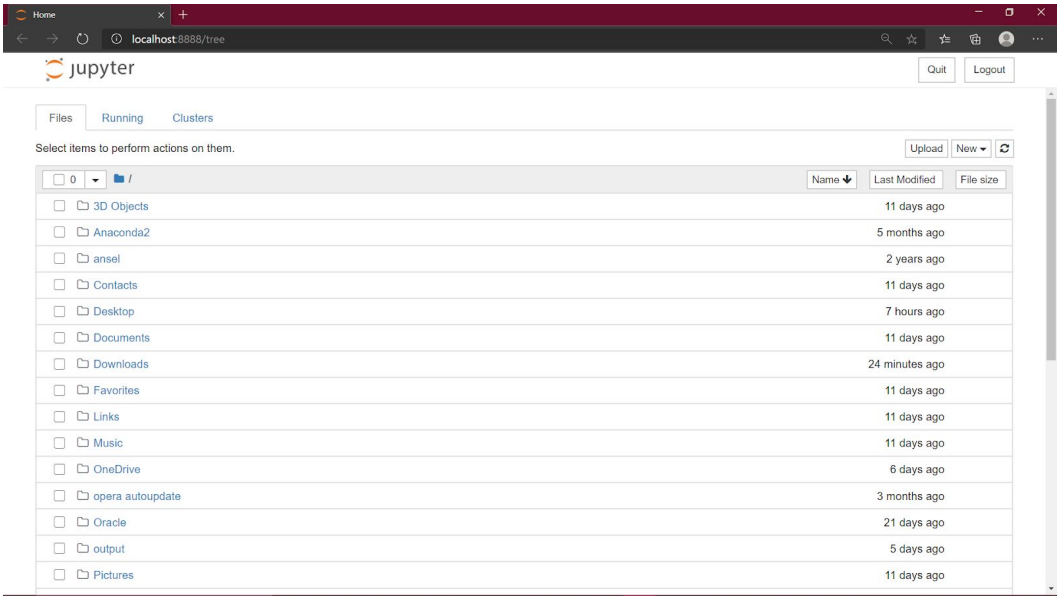
Installing Jupyter

To more easily work with this data that we have extracted, we will be using Jupyter notebook. Jupyter allows you to use Python in an interactive format, allowing you to run and modify small bits of code at a time.

First, let's set up Jupyter Notebook. If your host machine does not have Python installed, check out this website to get the latest release. Next, install Jupyter by running `pip install notebook` and `pip install jupyter` in your command line (or `conda`

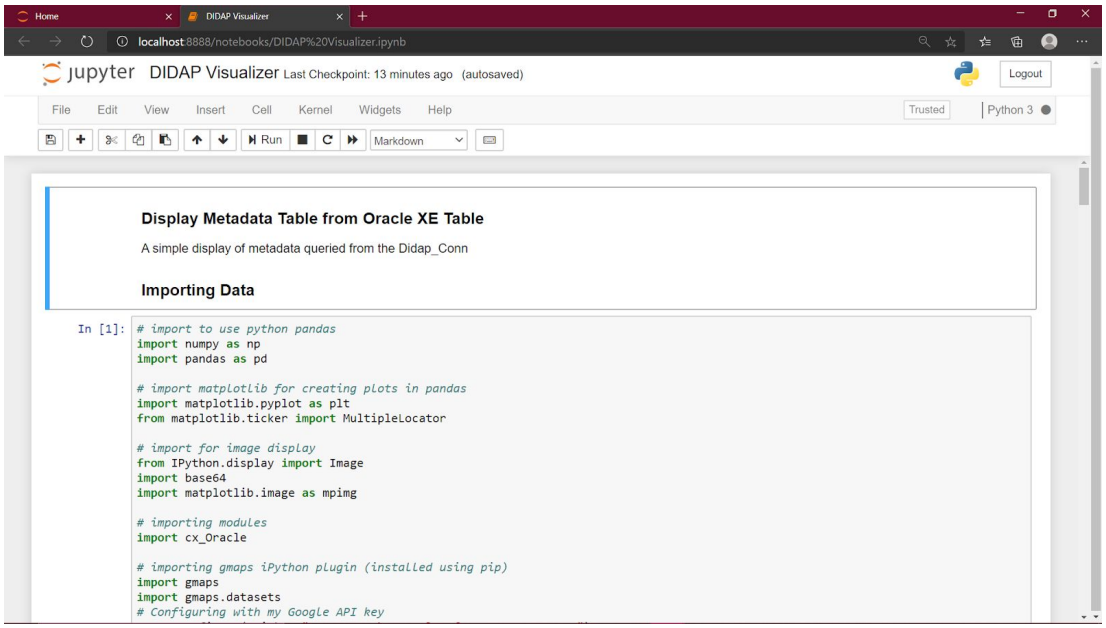
`install -c conda-forge notebook` if you have Python through a conda distribution). For more help, see this site.

Now, you should be able to run the notebook with the command `jupyter notebook` or, if that does not work, `python -m notebook`. If the installation is done correctly, a window should open similar to the following:



Syncing the Jupyter Notebook Python Codebase

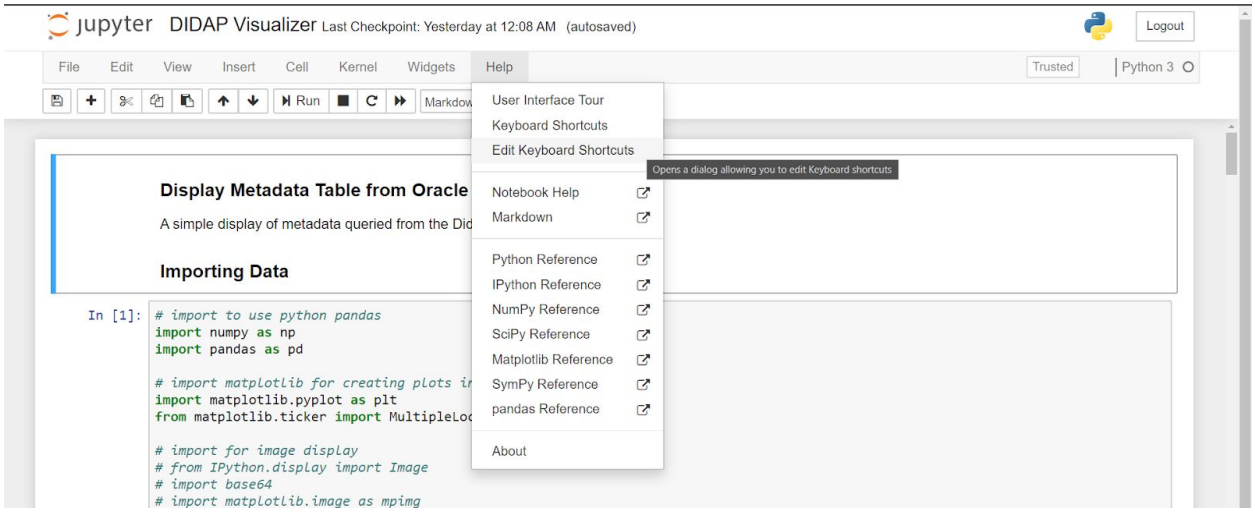
Download the `DIDAP Visualizer.ipynb` notebook from the Data Visualization folder on Sakai. In your Jupyter tab, find the downloaded notebook and select it. The notebook should open in another tab.



Setting up the Visualizer

Each individual box is called a cell, and can be run by pressing the “Run” button or a custom keyboard shortcut. You can also set shortcuts to run the entire file at once.

To set custom shortcuts, simply navigate to Help > Edit Keyboard Shortcuts:



Before running the code, you will need to install and enable various modules. If you installed Jupyter with pip, run the following commands in the command line:

```
> pip install numpy
> pip install pandas
```

```
> pip install matplotlib
> pip install ipython-sql
> jupyter nbextension enable --py --sys-prefix widgetsnbextension
> pip install gmaps
> jupyter nbextension enable --py --sys-prefix gmaps
```

If using conda, run these commands instead:

```
> conda install -c anaconda numpy
> conda install -c anaconda pandas
> conda install -c conda-forge matplotlib
> conda install -c conda-forge gmaps
> conda install -c conda-forge ipython-sql
```

The first cell contains a Google API key to access the Google Maps API. If you have your own key, replace the long string in the `gmaps.configure(api_key="key")` line. Now, you should be able to run the first cell without any errors. If you get an Import Error, `No module named x`, then double check that you have installed the required module using the commands above. If the problematic module is not listed, then it may be an unused module which you can remove from the code.

In the second cell, there is only one line. For the developer's version of this application, images are retrieved from the local machine. By moving the files to the shared folder, you should download the images which were saved by the Hadoop job on your VM to a folder in the same directory as the Jupyter notebook we are working with. Set the variable `im_path` to this local directory at which the input images are located, and run this cell.

Now, in the next cell, we will set up the connection with the Oracle database. After the module is reloaded, the next line should be in the format `%sql oracle+cx_oracle://USER:PASS@localhost/`. Replace USER and PASS with the username and password you use to login as your pre-created user into the Oracle XE SQL Developer (If you don't have this set up yet, you are very behind. Please refer to Phases 1-3 of the Oracle XE/SQL Developer Documentation). In the following line, replace the last word of the query with your table name, for the Oracle DB table containing the metadata. Similarly, in the next cell, replace the last word of the query with the table containing the mine image information.

Now, use your keyboard shortcut or navigate to Cell >> Run All to run the entire file and create the graphs and maps showing the data from the original MSTIFF files.

Congratulations! You have successfully run the Visualizer!

Code Explanation

As the code comments show, the first half of the notebook primarily imports and cleans the data. After importing data into the notebook, we save the table as a dataframe so that we can use pandas functions to easily remove all rows with missing values, sort the rows by time, and split the data into different dataframes if there are large time gaps. There are also a few conversions to ensure that the dates and times are in a workable format. Finally, a list is created of all image files containing mines.

The second section plots data. The first cell creates plots for water depth and towfish depth over time. The next few cells create a Google maps widget, plot the paths of each mission, add a pin with a key to show which dates are associated with which colored paths, and add pins at the approximate location at which a mine may be located along with its corresponding image. The last cell also plots latitude and longitude in a plain grid format.

At this point, you should be ready to work with the DiDAP Platform to your liking. You've done a great job working through this tutorial!