Python Wrangling for SAP HANA Application Developers
DAT-368

Exercises / Solutions (UPDATED)

Andrew Lunde / SAP

SAP TECHED

TABLE OF CONTENTS

BEFORE YOU START

System Host: wdflbmt0794.wdf.sap.corp
System Instance Number: 00
System User ID: DAT368
All Passwords: WelcomeSAP2018
XSA Organization: TECHED
XSA Development Space: DEV

Getting Help

If you need addition help resources beyond this document, we would suggest the following content:

- The Online Help at https://help.sap.com/viewer/4505d0bdaf4948449b7f7379d24d0f0d/2.0.03/en-US/8d786ec8ab964145a7453c1f53f452db.html

Source Code Solutions

The source code repository for the first part of this exercise with the python module disabled can be found in the following webpage.

https://github.com/alundesap/TechEd2018.DAT368

In exercise #3 we will switch to another branch of the project where the python module is enabled.

Open the browser and enter the following URL to access the python enabled solution web page

https://github.com/alundesap/TechEd2018.DAT368/tree/solution

Updated versions of this exercise document and presentation are available here.

DAT368_Exercises (UPDATED)

DAT368_Presentation

Time Considerations

If you are familiar with the SAP HANA Web IDE and the xs command line interface, you may consider skipping some sections of the exercises in the interests of time.

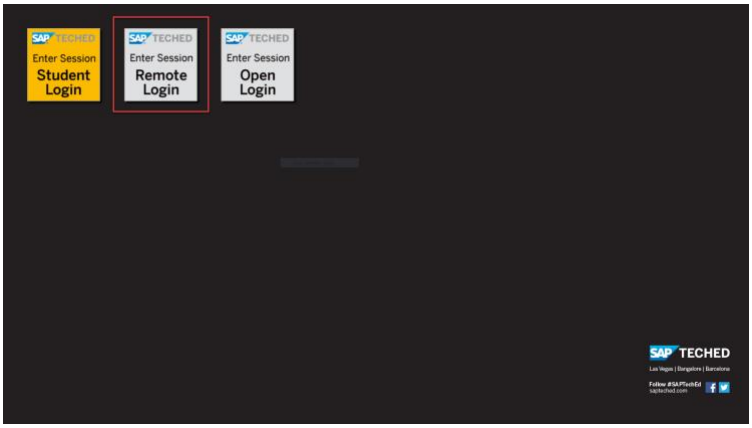
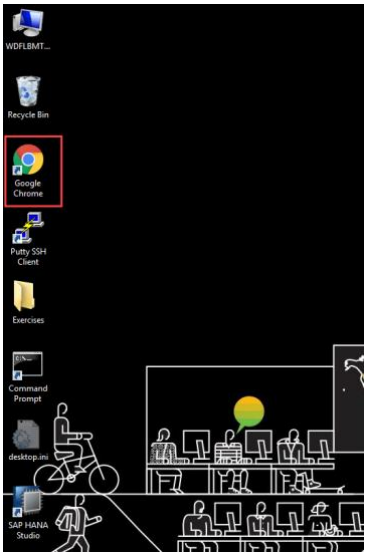
Be sure to start with the first exercise and complete all the steps until you come to a Time Check: notation. Skip ahead to the point indicated and continue with the steps until you come to another Time Check:. This insures that you've completed the required steps and have read the supporting discussion.

EXERCISE 1

During this exercise it's important to understand that the primary tool that SAP provides for micro-service application development is a convenient front end to the server's underlying activities. We can perform the same activities manually with the command line interface by ssh'ing into the server itself. We will be jumping back and forth from the Web IDE and the XS CLI in order to get you comfortable with the concepts and our action's effects. First we'll start with a project the doesn't contain a Python module to get the feel of things.

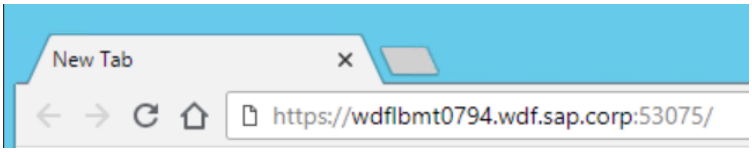
Time Check: If you are familiar with the functioning of the Web IDE you can save time by skipping to Exercise 1.2.

Exercise 1.1: MTA Project - Web IDE Usage and Limitations

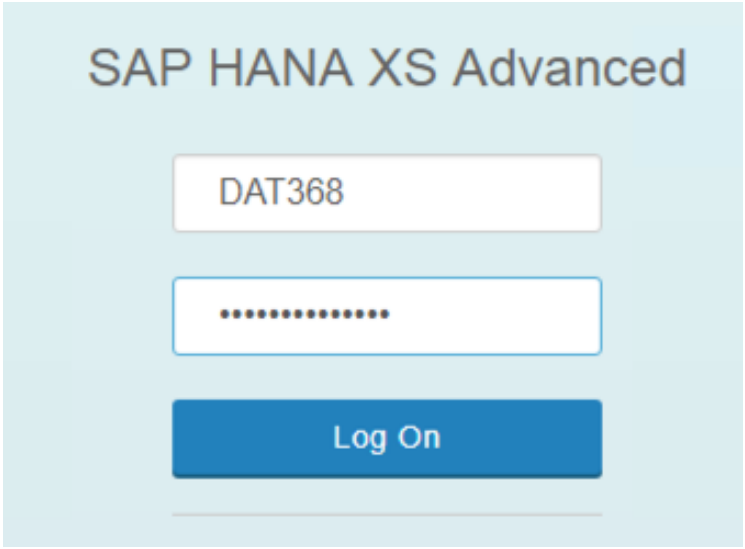
Explanation	Screenshot
1. Please click on the **Remote Login** tile. You will be asked for username and password - please enter: **.\student** with password: **Welcome18**	
2. From your desktop launch the Google Chrome web browser.	

Launch the SAP Web IDE for SAP HANA at the following URL in your web browser.

https://wdflbmt0794.wdf.sap.corp:53075/



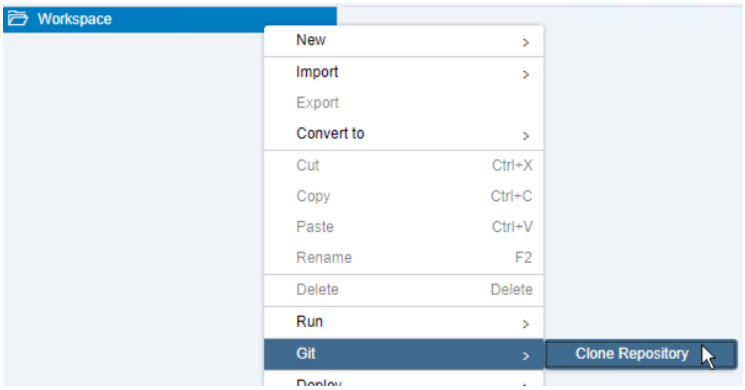
3. User: **DAT368**
   Password: **WelcomeSAP2018**



Note: If the Tips and Tricks dialog window appears, just close it by clicking the **Close** button.

Since the point of this exercise is to illustrate what the Web IDE does for us, we will begin by importing an existing MTA project into the workspace.

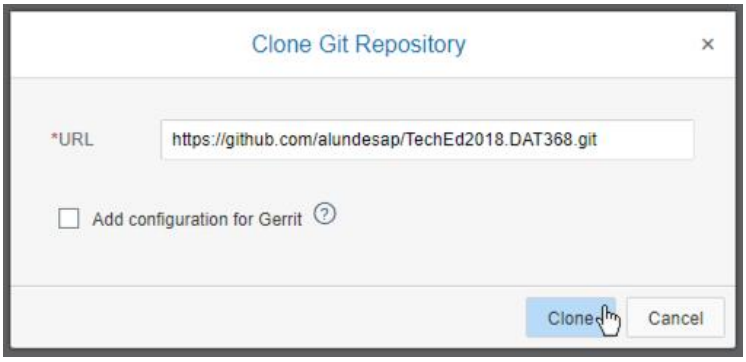
4. Right-click on Workspace and select **Git->Clone Repository**



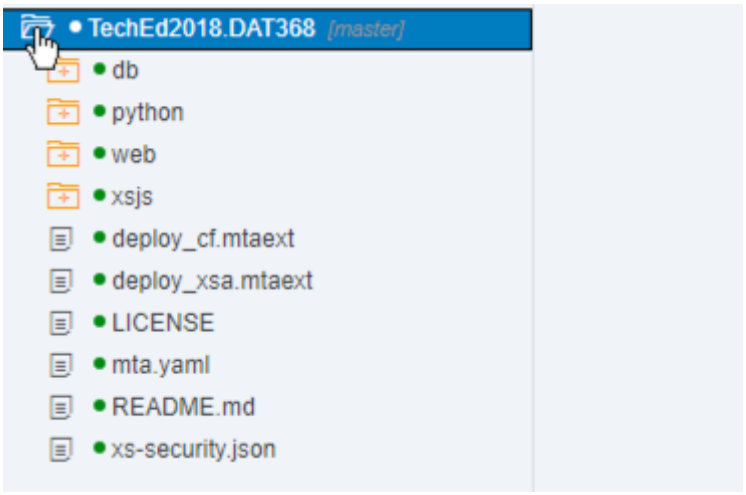
5. Use the following URL.

   https://github.com/alundesap/TechEd2018.DAT368.git

   and click **Clone**



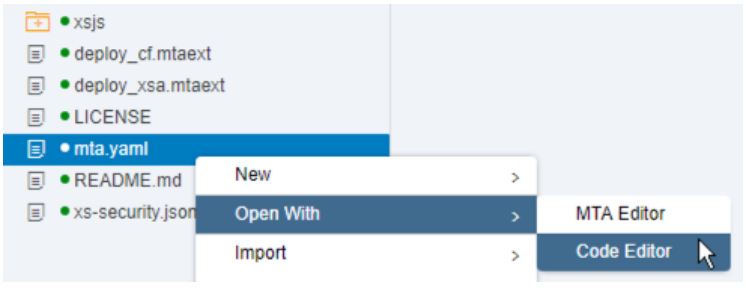
6. Expand the TechEd2018.DAT368 project to show it's modules.



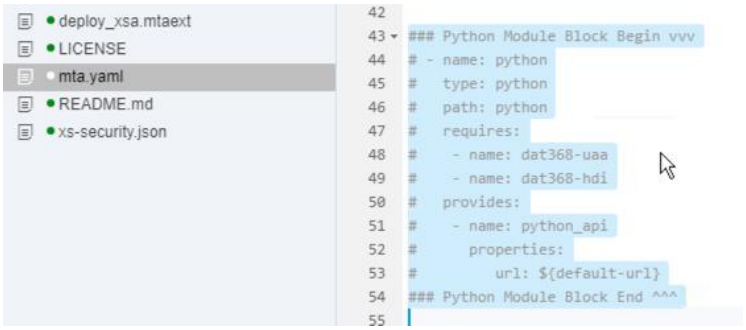
Notice that there are module folders named db, python, web, and xsjs. While the db, web, and xsjs modules were created with the Web IDE, the python folder is just a simple folder with python files in it. If you right click on the db, web, or xsjs folders, you'll see there there is build option available. If you do the same with the python folder, no build option is present. This is because the Web IDE currently doesn't recognize modules of the type python.

Now let's examine the mta.yaml file that defines the project's modules and relations to each other and to other services. The Web IDE provides a graphical editor to yaml files, but we will want to look at the file in the code editor.

7. Right-click on the mta.yaml file and select **Open With -> Code Editor**.



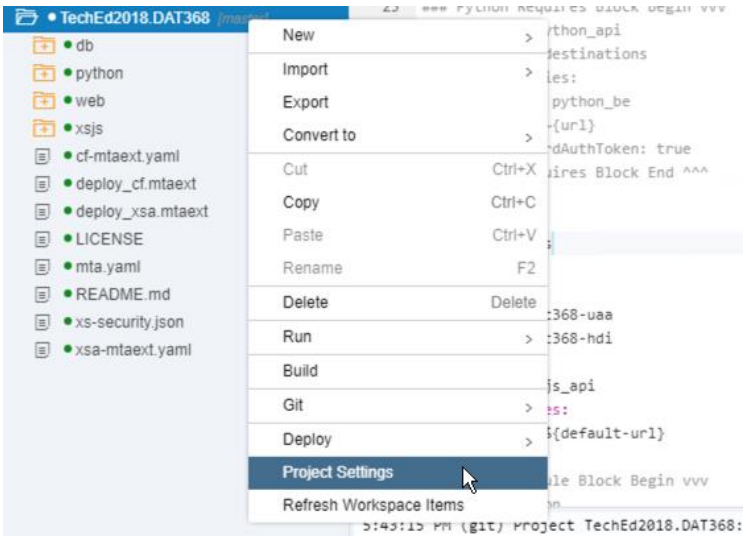
8. Scroll the mta.yaml file in the editor window and notice that the lines pertaining to the python module are commented out with # at the beginning of the line. Yaml allows for comments so these lines will not be processed.



9. Let's build the db module.

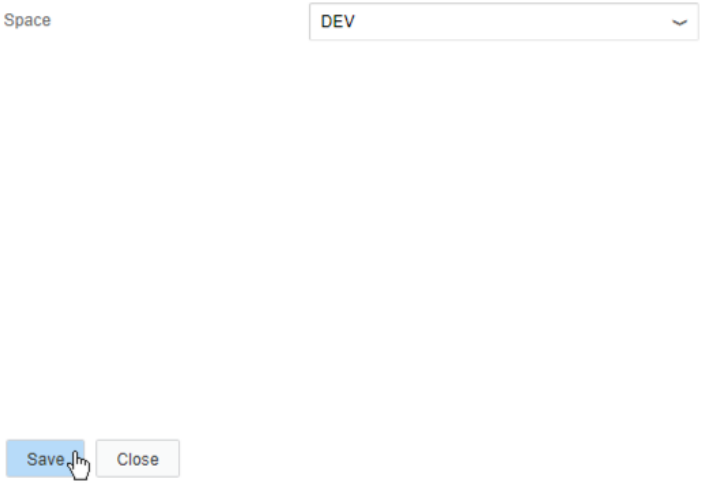
   However, since we've just imported this project, the system needs to know what space to associate the project with.

   Right-click on the project level and then click **Project Settings.**

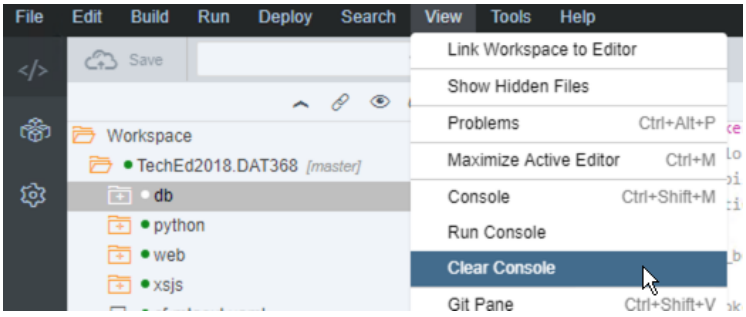


10. Select **Space**.

11. And select **DEV** from the **Space** dropdown. Then click the **Save** button, then **Close**.

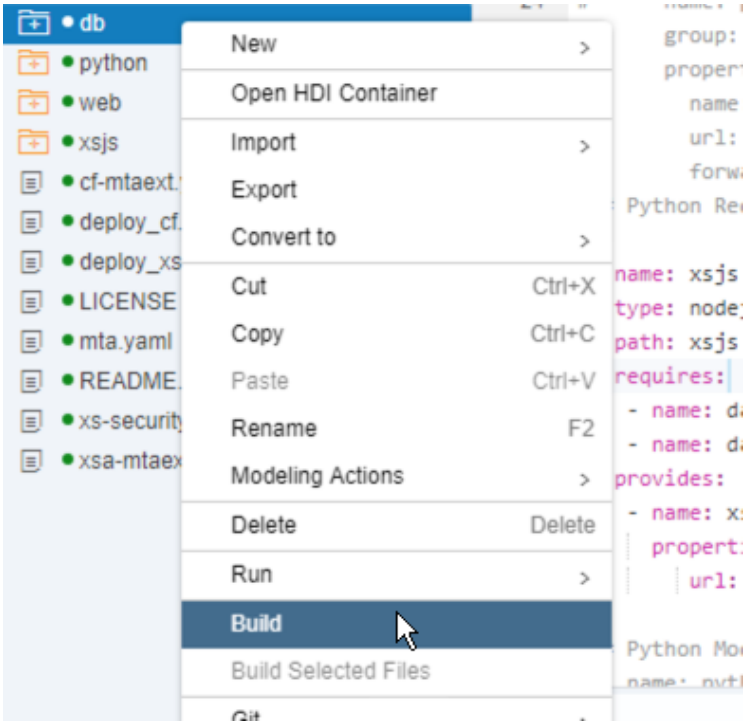


12. Before initiating a build operation, it's handy to clear out the contents of the console window first. From the menu, select **View->Clear Console**. This removes any prior build output lines so you are not confused by them.



13. Now we can build.

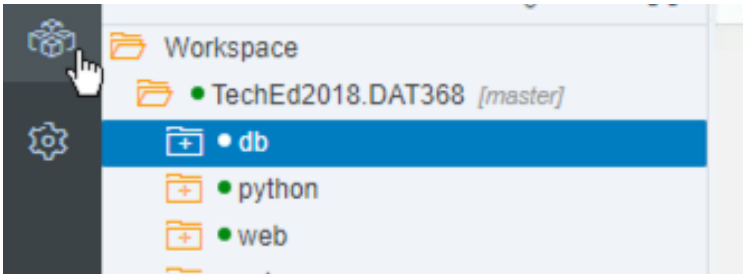
    Right-click on the db folder and select **Build**.



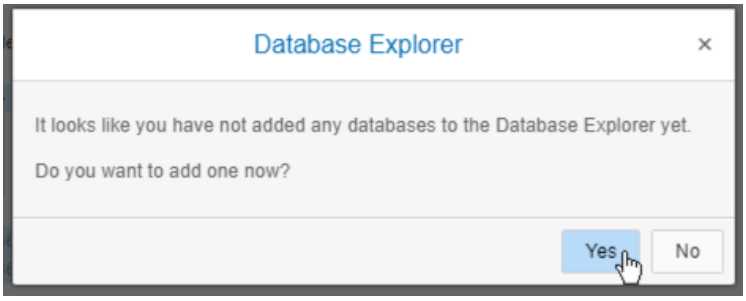
If the build went well, you should see the following lines at the bottom of the console output.

What really is going on is that a special nodejs module is being run that reads the files in the db folder and produces a private schema in the database with all the tables and views you've defined as well as a system generated user and password for access. This isolates the data from other schemas that may be deployed in the system. After the nodejs is done running, it stops but the resulting schema remains. We can connect to it and explore it's artifacts.

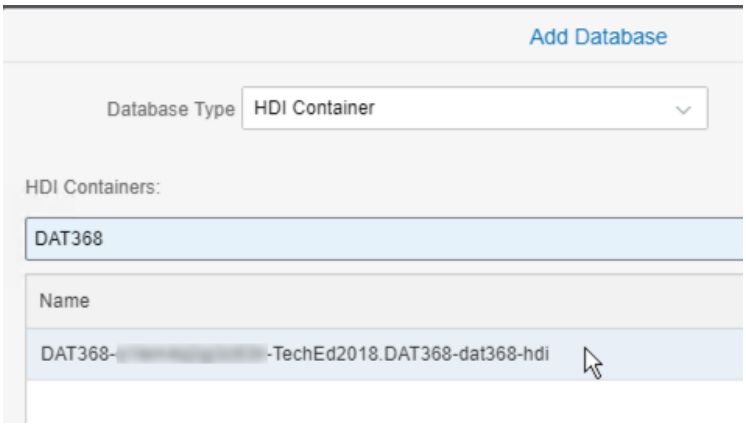
14. Click on the DB Explorer icon.



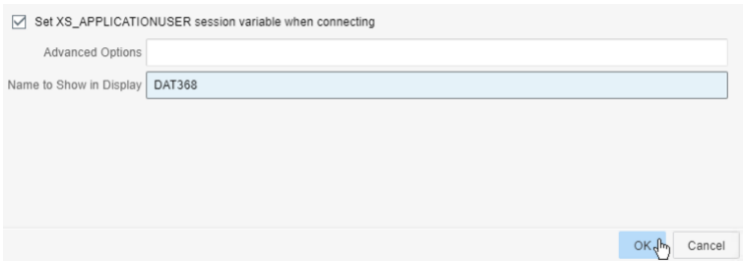
15. Since no DB connections have yet been made, click **Yes**.



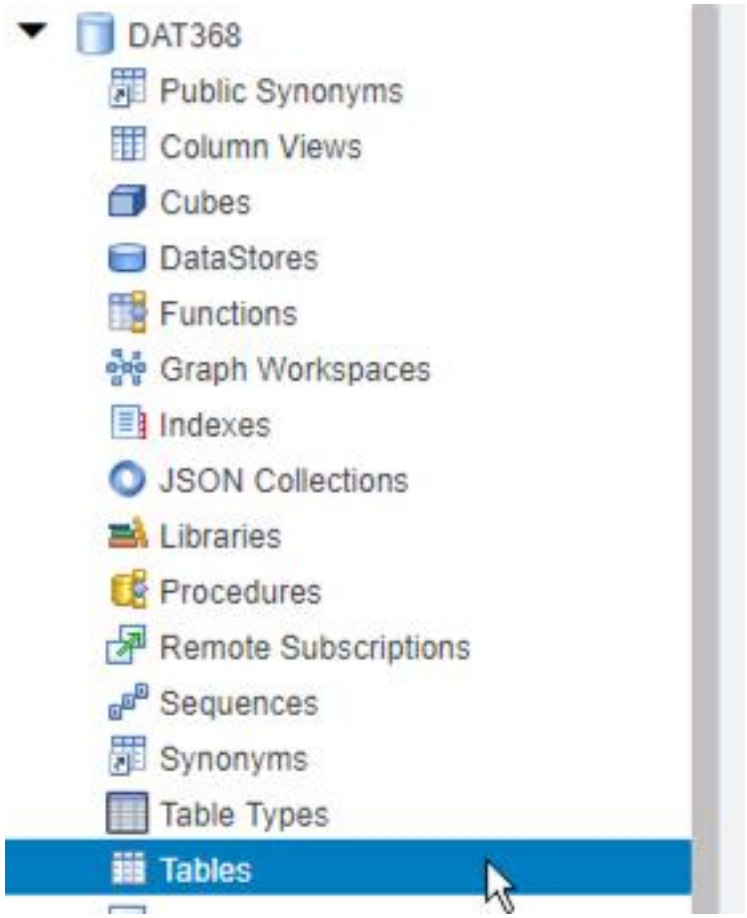
16. Enter **DAT368** in the HDI Containers field to narrow the list, and then select the line that starts with DAT368.



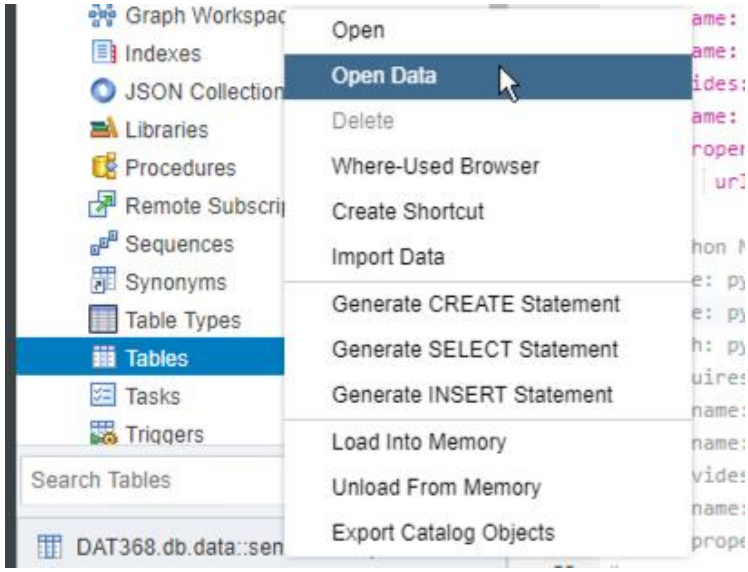
17. Remove the long suggested name and enter **DAT368** in the Name to Show in Display field and then click the **OK** button.



18. Left-click on Tables. Notice that a table named DAT368.db.data::sensors.temp appears in the list below.



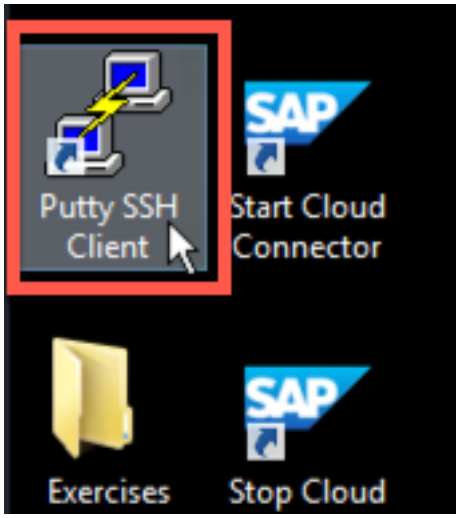
19. Right-click on the DAT368.db.data::sensors.temp table and select **Open Data**.



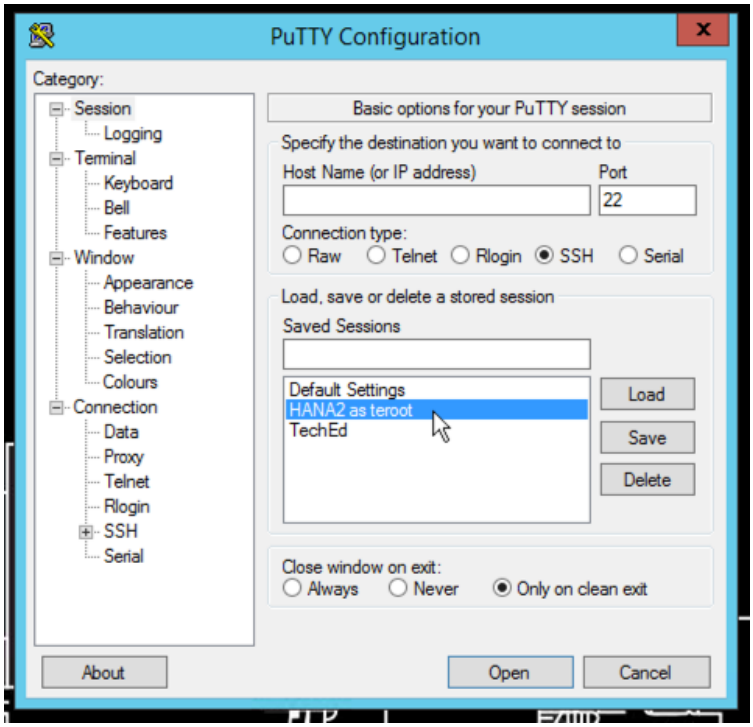
You will see the table's contents in the window to the right. There is currently one row of data with tempVal of 100 that exists in a single table. This was inserted as part of the processing of the db module.

Exercise 1.2: Using the XS CLI

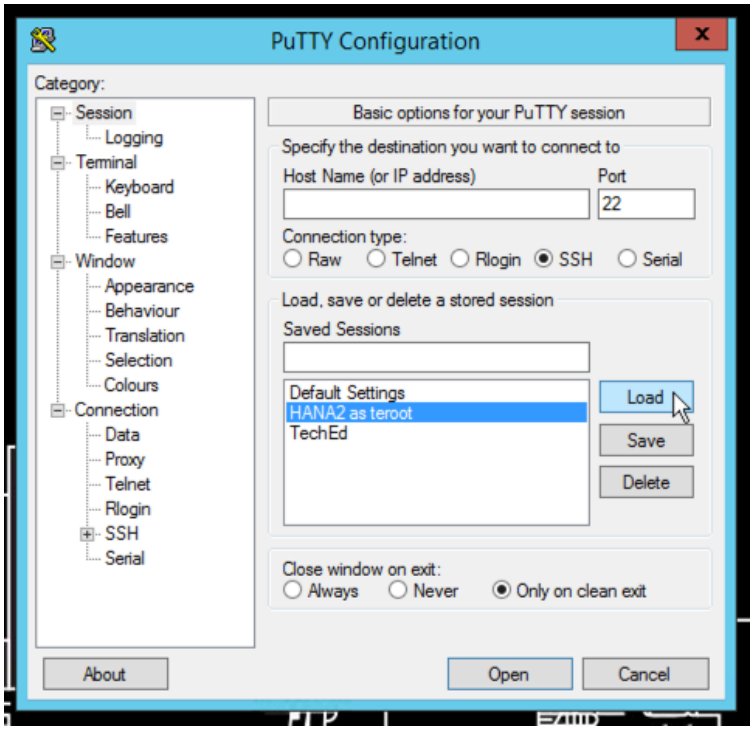
We will use a common windows Secure Shell tool called Putty SSH to inspect the underlying system with the XS Command Line Interface(CLI). While it's possible to install the XS CLI on the local workstation, it's often better to work on the server itself so that the environment that you're building with is similar to the environment you will deploy into.

Explanation	Screenshot
1. Return to the desktop and double-click on the Putty SSH Client icon.	

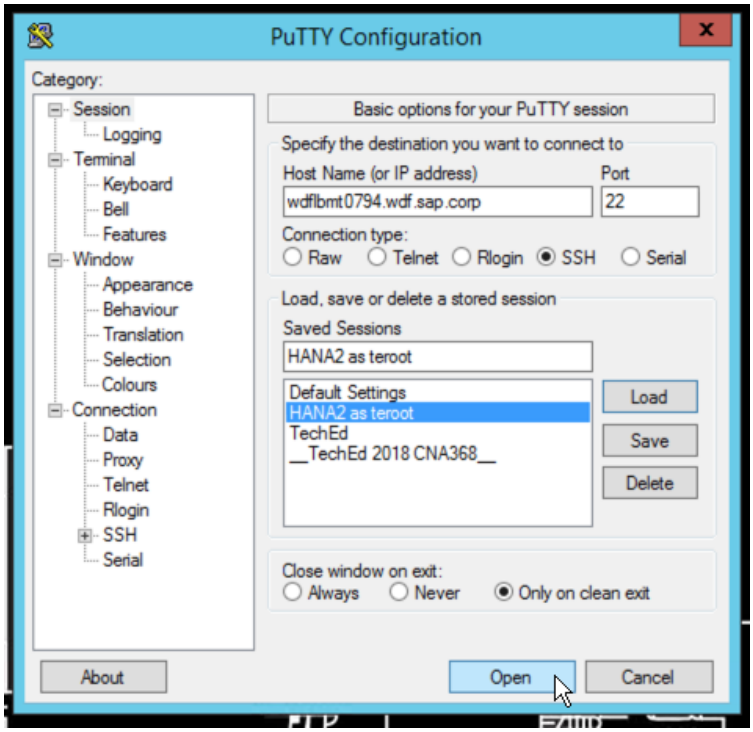
2. Select the **HANA2 as teroot** line from the Saved Sessions.



3. Then click **Load**.



4. And finally **Open.**



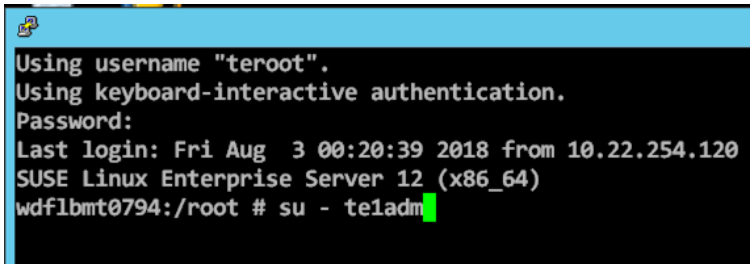
5. A putty window will open and you'll be prompted to enter a password.

   Password: **Welcome18**

6. Become the user te1adm with the command.

   su - te1adm

   This is the DB administration user for the TE1 tenant DB and we'll use this user for subsequent commands.



You may want to open multiple Putty session windows, just repeat the prior login.

Note: Putty will copy text into the copy buffer by simply selecting it. You can then past the text in Windows with Ctrl-V. To perform the opposite, use Ctrl-C in Windows and then right-click in the Putty console to paste the copy buffer into the command line.

Be very careful not to pick up any whitespace when selecting text! Double check after pasting!

Note: If this document is a pdf, make sure that your pdf reader is in select mode and not action mode (hand icon) when trying to select the bolded text commands in the document.

7. Verify the the xs cli is set correctly.

xs api

```
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00> xs api

API endpoint: https://wdflbmt0794.wdf.sap.corp:30030
SSL trust: The authenticity of host 'wdflbmt0794.wdf.sap.corp'
router/default.root.crt.pem' certificate.

te1adm@wdflbmt0794:/usr/sap/TE1/HDB00>
```

If you find the api not set properly, you can reset it with this command.

xs api https://wdflbmt0794.wdf.sap.corp:30030 --cacert /hana/shared/TE1/xs/controller_data/controller/ssl-pub/router/default.root.crt.pem

8. Log in as the XSA_ADMIN user.

xs login -u XSA_ADMIN -p WelcomeSAP2018

```
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00> xs login -u XSA_ADMIN -p WelcomeSAP2018

API_URL: https://wdflbmt0794.wdf.sap.corp:30030
USERNAME: XSA_ADMIN
Authenticating...
ORG: TechEd
SPACE: DEV
API endpoint:   https://wdflbmt0794.wdf.sap.corp:30030 (API version: 1)
User:           XSA_ADMIN
Org:            TechEd
Space:          DEV
```

9. If the space is not DEV then select the DEV space with the xs target command.

xs t -s DEV

```
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00> xs t -s DEV

API endpoint:   https://wdflbmt0794.wdf.sap.corp:30030 (API version: 1)
User:           XSA_ADMIN
Org:            TechEd
Space:          DEV

te1adm@wdflbmt0794:/usr/sap/TE1/HDB00>
```

Time Check: If you are familiar with how to use the xs cli you can skip to Exercise 1.3.

10. Use the xs apps command to list all the applications(modules) in the space.

xs apps

```
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00> xs apps

Getting apps in org "TechEd" / space "DEV" as XSA_ADMIN...
Found apps:

name                          requested state   instances   memory   disk         urls
-------------------------------------------------------------------------------------------------------------------
hrtt-service                  STARTED           1/1         512 MB   <unlimited>  https://wdflbmt0794.wdf.sap.corp:51009
sqlanlz-svc                   STARTED           1/1         256 MB   <unlimited>  https://wdflbmt0794.wdf.sap.corp:51010
sqlanlz-ui                    STARTED           1/1         128 MB   <unlimited>  https://wdflbmt0794.wdf.sap.corp:51011
hrtt-core                     STARTED           1/1         512 MB   <unlimited>  https://wdflbmt0794.wdf.sap.corp:51012
di-local-npm-registry         STARTED           1/1         1.00 GB  <unlimited>  https://wdflbmt0794.wdf.sap.corp:51015
di-core                       STARTED           1/1         2.00 GB  <unlimited>  https://wdflbmt0794.wdf.sap.corp:51016
devx-ui5                      STARTED           1/1         1.00 GB  <unlimited>  https://wdflbmt0794.wdf.sap.corp:51017
di-runner                     STARTED           1/1         2.00 GB  <unlimited>  https://wdflbmt0794.wdf.sap.corp:51020
di-cert-admin-ui              STOPPED           0/1         1.00 GB  <unlimited>  https://wdflbmt0794.wdf.sap.corp:51021
di-space-enablement-ui        STOPPED           0/1         1.00 GB  <unlimited>  https://wdflbmt0794.wdf.sap.corp:51022
tools                         STARTED           1/1         128 MB   <unlimited>  https://wdflbmt0794.wdf.sap.corp:51023
webide                        STARTED           1/1         512 MB   <unlimited>  https://wdflbmt0794.wdf.sap.corp:53075
xsa-admin-backend             STARTED           1/1         128 MB   <unlimited>  https://wdflbmt0794.wdf.sap.corp:51026
xsa-admin                     STARTED           1/1         128 MB   <unlimited>  https://wdflbmt0794.wdf.sap.corp:51025
xsa-cockpit                   STARTED           1/1         512 MB   <unlimited>  https://wdflbmt0794.wdf.sap.corp:51027
di-builder                    STARTED           1/1         1.00 GB  <unlimited>  https://wdflbmt0794.wdf.sap.corp:51006
sapui5_fesv4                  STARTED           1/1         256 MB   <unlimited>  https://wdflbmt0794.wdf.sap.corp:51024
sapui5_sb                     STARTED           1/1         128 MB   <unlimited>  https://wdflbmt0794.wdf.sap.corp:51028
web-console                   STARTED           1/1         1.00 GB  <unlimited>  https://wdflbmt0794.wdf.sap.corp:51029
TechEd2018CentralDB.core_db   STOPPED           0/1         256 MB   <unlimited>  <none>

te1adm@wdflbmt0794:/usr/sap/TE1/HDB00>
```

Note that there are quite a few applications. Some of them are system tools and would normally be installed in a different space. In order to focus on our application we will often filter the output with grep. As an example, let's look at the apps with xsa in their name. I'll use the shorthand for the apps command (a).

11. Enter this command.

xs a | grep xsa

```
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00> xs a | grep xsa
xsa-admin-backend              STARTED          1/1          128 MB
xsa-admin                      STARTED          1/1          128 MB
xsa-cockpit                    STARTED          1/1          512 MB
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00>
```

12. Let's list the services that are in the space. Again, I'll use the shorthand(s) and filter out all but our container.

xs s | grep DAT368

Note: If you had skipped ahead, you won't see this service.

```
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00> xs s | grep DAT368
DAT368-[redacted]-TechEd2018.DAT368-dat368-hdi    hana
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00>
```

13. And finally the (r)outes that apps in this space have registered.

xs r

```
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00> xs r

Getting routes in org "TechEd" / space "DEV" as XSA_ADMIN...

host   domain                    port    path   type   apps
-------------------------------------------------------------------------------
       wdflbmt0794.wdf.sap.corp   51009   /      HTTP   hrtt-service
       wdflbmt0794.wdf.sap.corp   51010   /      HTTP   sqlanlz-svc
       wdflbmt0794.wdf.sap.corp   51011   /      HTTP   sqlanlz-ui
       wdflbmt0794.wdf.sap.corp   51012   /      HTTP   hrtt-core
       wdflbmt0794.wdf.sap.corp   51015   /      HTTP   di-local-npm-registry
       wdflbmt0794.wdf.sap.corp   51016   /      HTTP   di-core
       wdflbmt0794.wdf.sap.corp   51017   /      HTTP   devx-ui5
       wdflbmt0794.wdf.sap.corp   51020   /      HTTP   di-runner
       wdflbmt0794.wdf.sap.corp   51021   /      HTTP   di-cert-admin-ui
       wdflbmt0794.wdf.sap.corp   51022   /      HTTP   di-space-enablement-ui
       wdflbmt0794.wdf.sap.corp   51023   /      HTTP   tools
       wdflbmt0794.wdf.sap.corp   53075   /      HTTP   webide
```

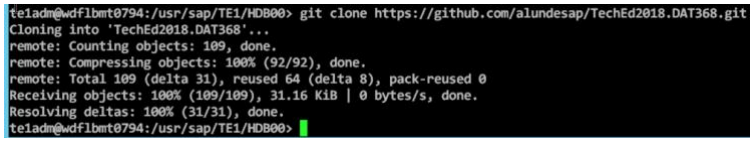
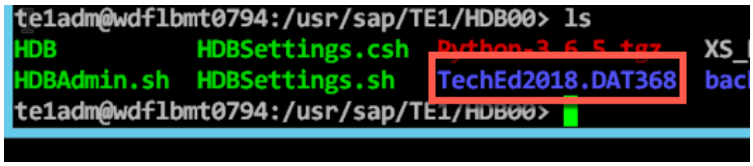
We can see by listing the services that there is one named DAT368-....-dat368-hdi that matches the name of the container that we viewed using the DB Explorer function in the Web IDE. Keep in mind that while the Web IDE provides a nice GUI, the underlying actions can also be performed with the xs command line tool.

Exercise 1.3: Cloning the project into the server's file space

We will need a copy of the same example project in the te1adm's home directory as well so that we can perform the tasks for building the project with the python module. Enter the following git clone command.

git clone https://github.com/alundesap/TechEd2018.DAT368.git

This can take about a 1 1/2 minutes.

1. Use git to clone the repo. **git clone https://github.com/alundesap/TechEd2018.DAT368.git**	
2. List the files and folders in the current directory. **ls** Make sure you see the **TechEd2018.DAT368** folder.	
3. Change into the git cloned directory. **cd TechEd2018.DAT368** and list the files. **ls -1**	

The reason we need to clone the project in the filesystem as well is that we need to reference the xs-security.json file when setting up our application's authorization service.

The Web IDE can't create an authentication service instance for us so we need to create one for ourselves. In addition, we will pass a configuration file that defines the role our application will need. Perform this xs command to manually create the xsuaa service instance with the xs-security.json configuration file.

Note: We set up the authentication service instance ahead of time so that we can created the needed role collections and assign them to our user before deploying our full application.

4. Create the service instance.

xs cs xsuaa default dat368-uaa -c ./xs-security.json

```
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00/TechEd2018.DAT368> xs cs xsuaa

Creating service "dat368-uaa"...
OK

te1adm@wdflbmt0794:/usr/sap/TE1/HDB00/TechEd2018.DAT368>
```

Exercise 1.4: Role Collection Setup

Once the xsuaa service instance is created, the system understands the roles that you've defined in the xs-security.json file. We need to create role collections to contain these roles that we will then assign to users. To do so, we will need to access the XSA Cockpit utility. Find it's url by using the xs cli.

Note: Normally the xsa-cockpit tool will not be installed the the same space you would be targeting for development. However, for this workshop the tool is installed here so we don't have to change to a new space to get details about it.

1. Query the xsa-cockpit app for the urls that it's registered.

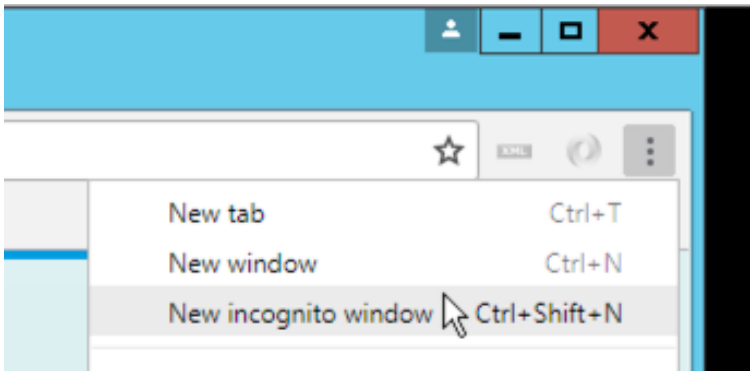
xs app xsa-cockpit --urls

```
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00/TechEd2018.DAT368> xs app xsa-cockpit --urls
https://wdflbmt0794.wdf.sap.corp:51027

te1adm@wdflbmt0794:/usr/sap/TE1/HDB00/TechEd2018.DAT368>
```

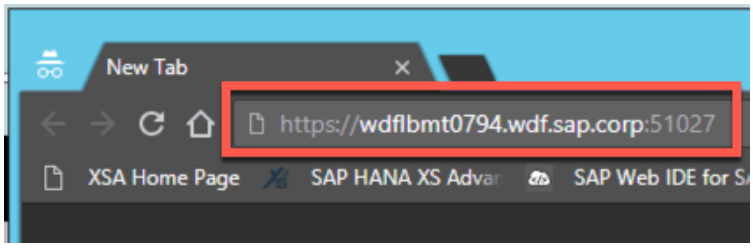
Double check that the URL that's returned matches this one and adjust the port number if yours differs.

https://wdflbmt0794.wdf.sap.corp:51027

2. In the Chrome browser, open a **new incognito window** by using the key combination Ctrl+Shift+N or clicking on the stacked dots icon in the upper-right corner of the browser window.



3. Paste the URL into the browser making sure the port number is what was returned above.

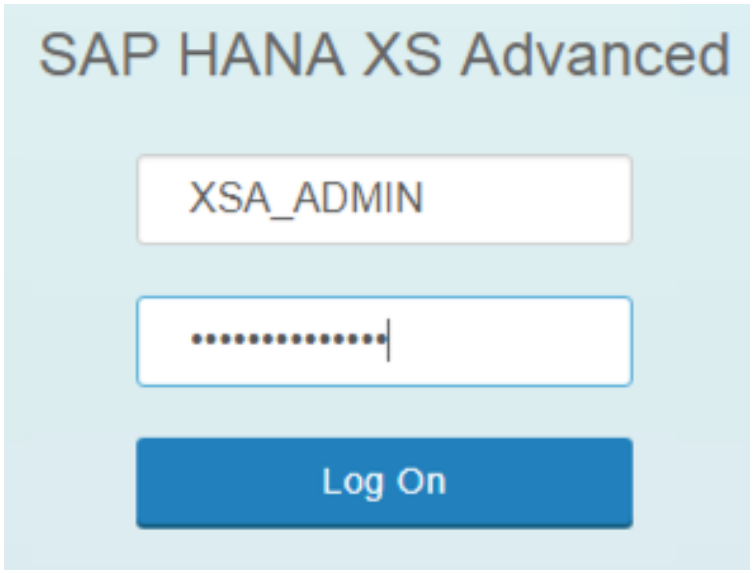


4. Log in but this time use the XSA_ADMIN user.

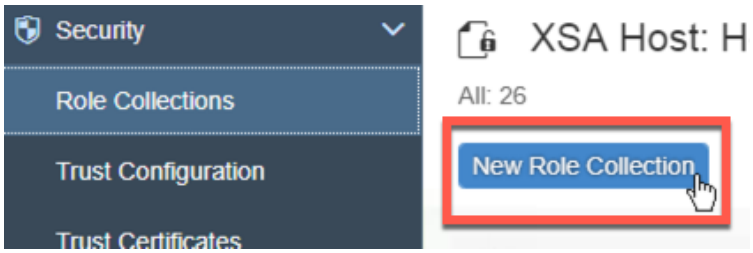
User: **XSA_ADMIN**
Password: **WelcomeSAP2018**

Note: You may want to type in the user and password by hand in order to avoid accidentally getting any whitespace characters when cutting and pasting.

Note: If you don't get presented with a login prompt, it likely means the you didn't perform these steps in an incognito window. On some system, even an incognito window won't allow you to log in as a different user. You can try to use a different browser like IE or try clearing your browser's cookies. Don't proceed unless you are able to login with the XSA_ADMIN user.

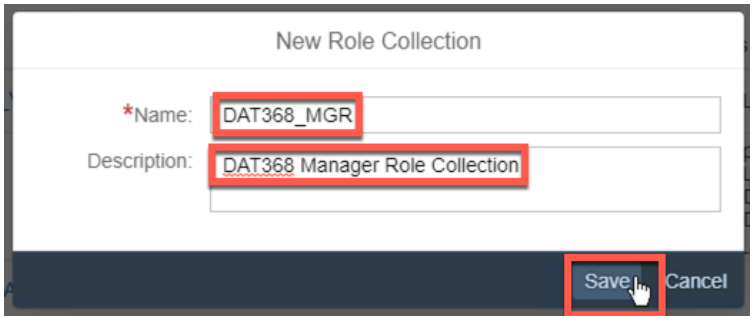


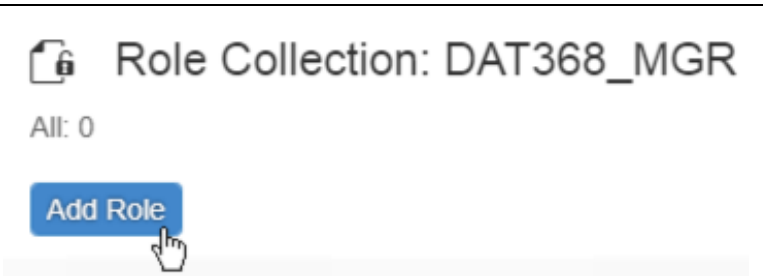
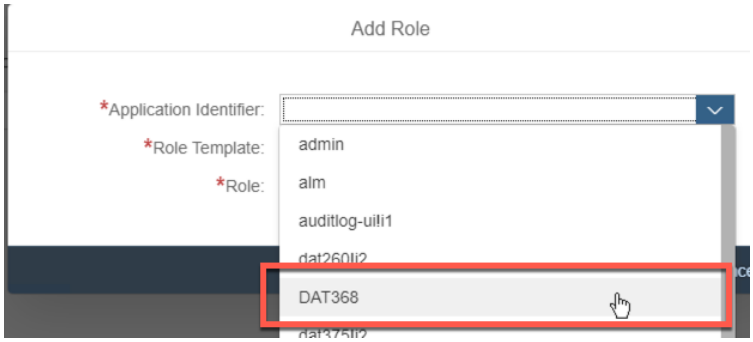
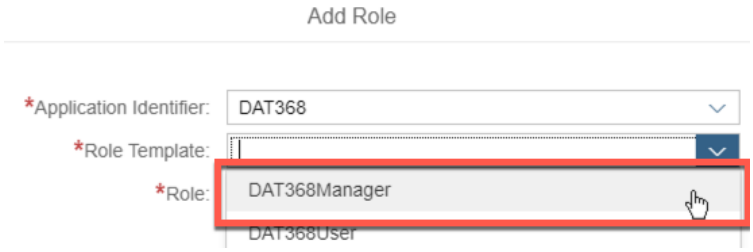
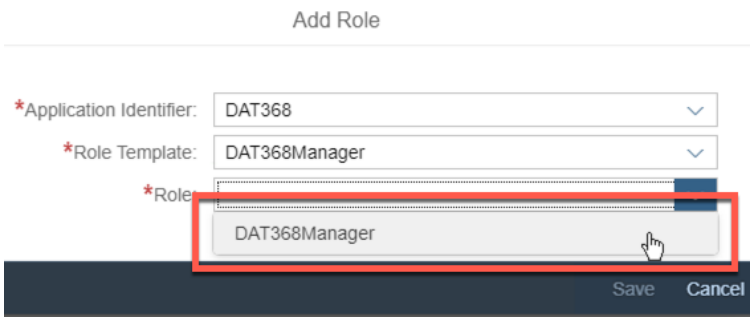
5. Under **Security**, select **Role Collections** and **New Role Collection**.



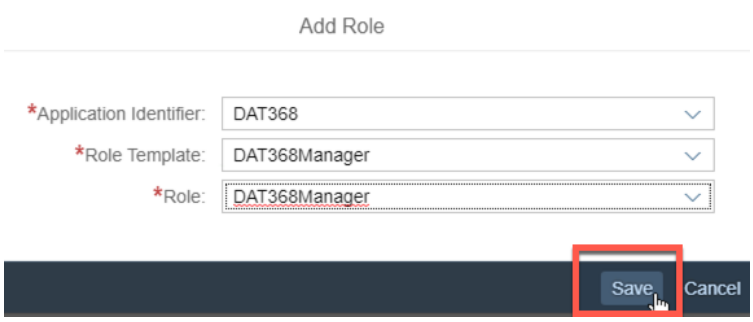
6. Give it a name and description and click **Save**.

Name: **DAT368_MGR**
Description: **DAT368 Manager Role Collection**



7. Click on the DAT368_MGR link.	
8. Click on **Add Role**.	
9. You will get a dialog box with 3 dropdowns. Starting with the top one, select DAT368.	
10. Then for Role Template, select **DAT368Manager**.	
11. The for Role, select **DAT368Manager**.	

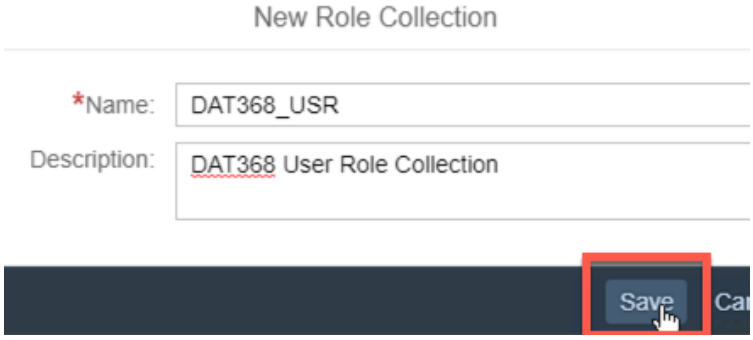
12. And finally, click **Save**.



Note: If the Application Identifier dropdown doesn't list DAT368, there was an issue with the creation of the xsuaa service instance. Go back and delete the dat368-uaa with "**xs ds dat368-uaa**" and re-create it making sure to pass the ./xs-security.json configuration file.

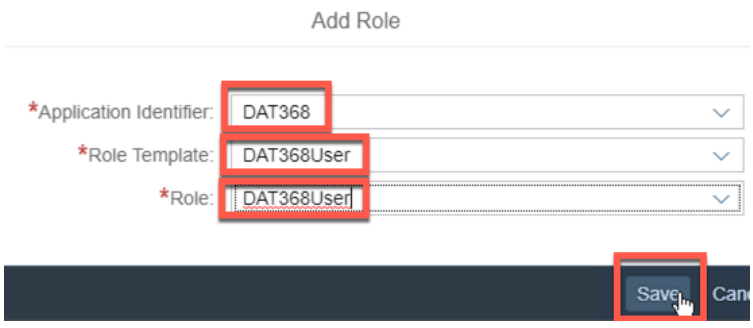
Go back to Home and create a **New Role Collection** as before this time calling it **DAT368_USR** with Description **DAT368 User Role Collection**.

13. Click **Save** when finished.



14. Click on the DAT368_USR link.

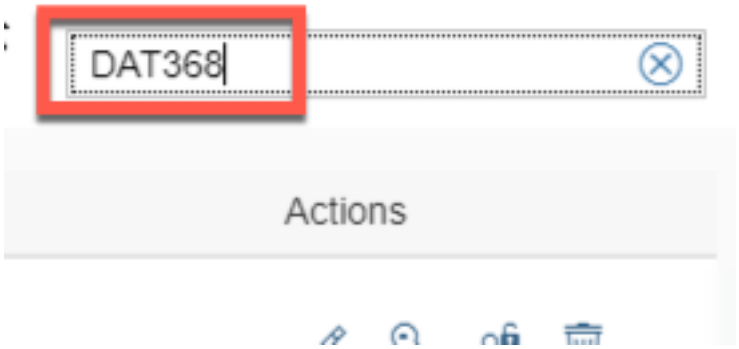
15. Click **Add Role** and select these values from the dropdowns and click **Save**.



Now that we've added roles to our role collections, we need to assign the role collection to a user. We'll use the **DAT368** user for this.

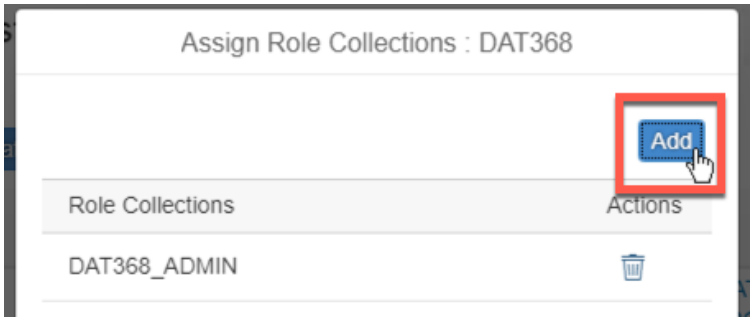
Click the Home link near the top of the page and **User Management**. In the search box type **DAT368**. This will limit the users being displayed.

16. In the search box type **DAT368**.

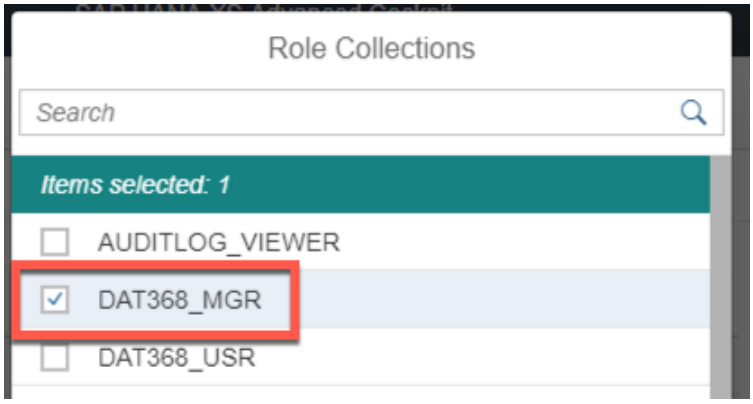


17. On the line with User Name DAT368, select the Assign Role Collections icon.

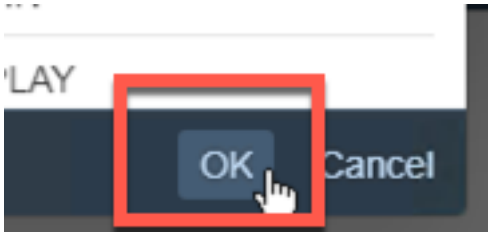
18. Click the **Add** button.



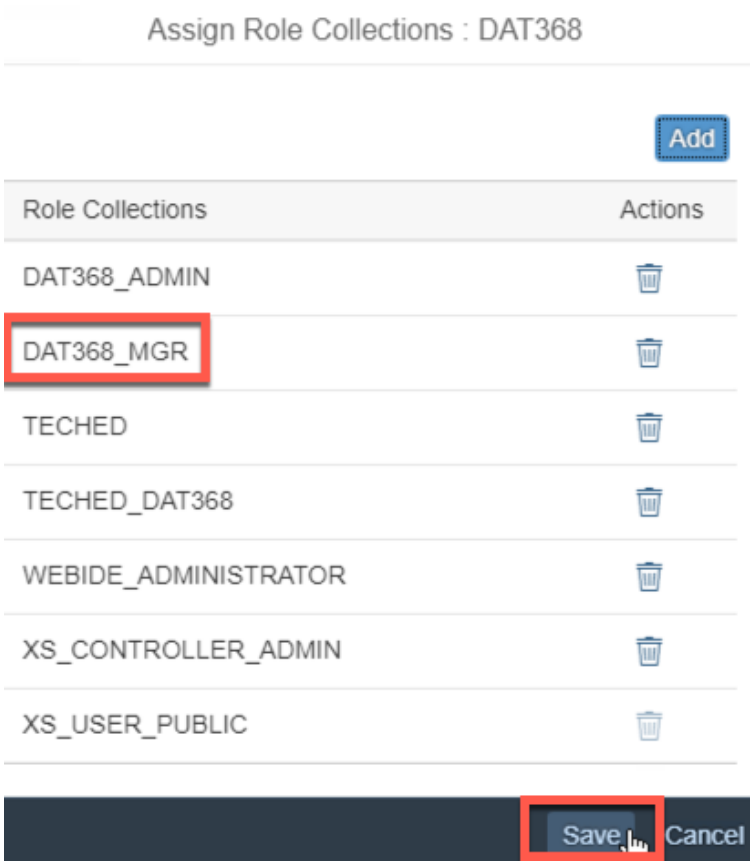
19. Select the **DAT368_MGR** Role Collection by checking the checkbox.



20. And click **OK**.



21. Verify that DAT368_MGR is in the list and click the **Save** button.



22. Close this incognito window.

This is a good time to verify that the user indeed has a proper role collection.

Time Check: If you are running short on time, feel free to skip ahead to the next exercise.

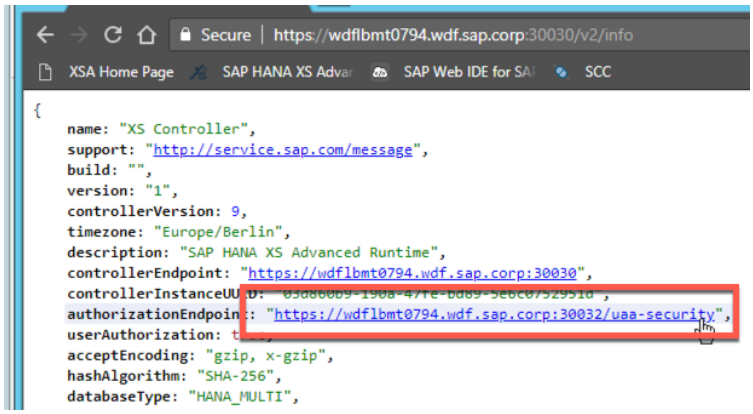
The best way to do this is to open a new incognito window and log into the authorization endpoint for our server. We need to dig out some information to do this.

First. find the api endpoint with the "xs api" command and then append /v2/info and browse to that URL. I've done this for you and here is the full URL.

https://wdflbmt0794.wdf.sap.corp:30030/v2/info

Paste this url into your fresh incognito browser window. The server will respond with a JSON file. If the browser you're using doesn't format json automatically, it may be difficult to read. Look for the authorizationEndpoint and select the URL that follows it.

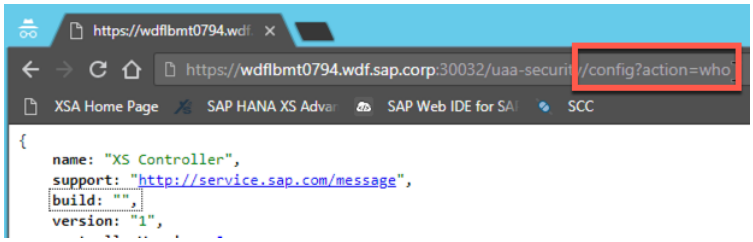
23. Open a new incognito window and browse to the URL above.

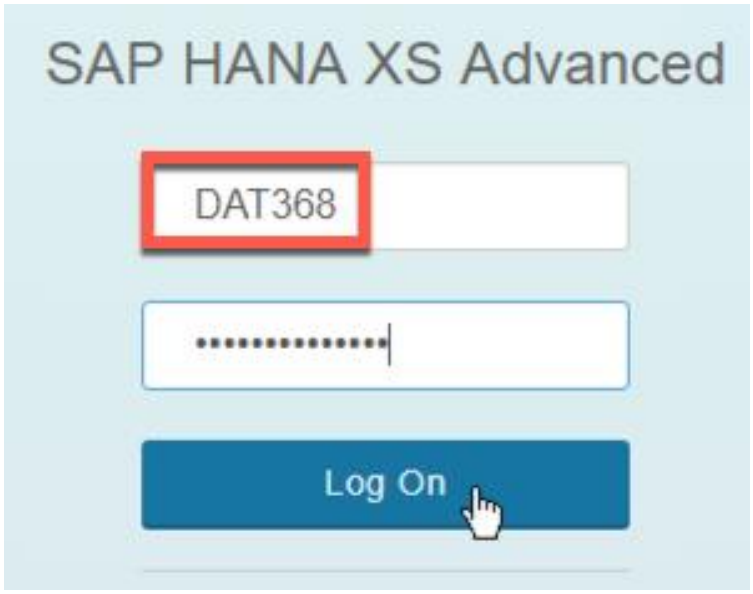
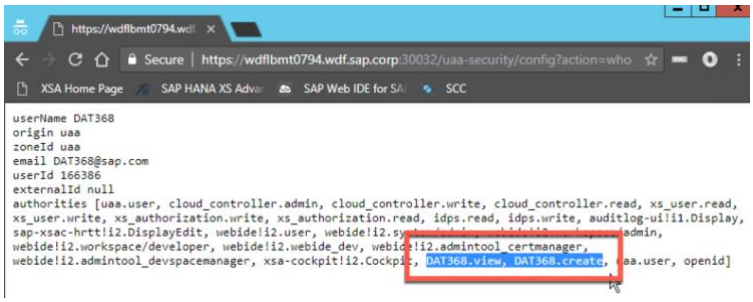


Open a new tab and paste the URL in the incognito window but don't hit Enter yet. Append "/config?action=who" to it first.

https://wdflbmt0794.wdf.sap.corp:30032/uaa-security/config?action=who

24. Replace the URL with the one above.



25. This time log in with the following. User: **DAT368** Password: **WelcomeSAP2018**	
26. Verify that the **DAT368.view** and **DAT368.create** authorities are in the authorities array. This confirms that the user has the proper role collection.	
27. Close this incognito window.	

EXERCISE 2

Exercise 2.1: Building an MTAR

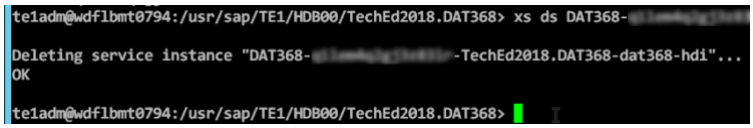
At this point we've been using the Web IDE to build the Database container and the XS CLI to create the authorization service instance. If the Web IDE could support the Python module type then we would be able to use it to build the python module from within the Web IDE. As of the time this workshop was created, this was not possible.

In order to get around this limitation we will use a command line tool to build the project's mtar file and deploy it with the xs cli. This gives us more control over the deploy process and will also work with the python module type.

But first, if you followed Exercise 1, let's remove the HDI container we created earlier. Otherwise skip the next two steps.

Time Check: Exercise 2 builds the project without the python module being enabled. If you want to save time and are familiar with building an MTAR file you can skip to Exercise 3.

Return to the Putty console window.

Explanation	Screenshot
1. Find the HDI container we created earlier in the Web IDE by searching for the a service instance with the name **DAT368** in it. **xs s | grep DAT368**	
2. Delete it with the ds command. **xs ds DAT368-xxxxx-TechEd2018.DAT368-dat368-hdi -f** Note that your name will differ. Replace the xxxxx with your system generated value. The easiest way to do this is to manually type xs ds and then cut/paste the DAT368... name and add a -f.	

DO NOT delete the **dat368-uaa** service instance as we've already used it to set up our application's role collections.

In the following steps we'll be using a command line tool called the MTA builder. This tool can be found at the following URL.

https://tools.hana.ondemand.com/#cloud

Find it under the section:

Multi-target Application Archive Builder

Note that the MTA Build Tool is provided as a java jar file. I've created a simple bash script to wrap it called **mta** and placed it in the path so that it can be used within any project folder. I've done this on the server for you, but you can substitute **java -jar mta_archive.jar** for **mta** and get the same result.

When you run the MTA build tool, it will invoke various build tools for the different types of modules that reside in your project. Since our project contains some nodejs modules, the nodejs package manager(npm) will be called to pull in the needed dependencies. In order to make sure that npm

finds the versions of the packages we want it to, it's important to set up the environment so that it finds sap packages first from the sap repository. Run this command to make sure this is the case.

npm config set @sap:registry "https://npm.sap.com/" ; npm config set registry "https://registry.npmjs.org/" ; npm config set strict-ssl true

3. Run the npm commands. **npm config set @sap:registry "https://npm.sap.com/"** **npm config set registry "https://registry.npmjs.org/"** **npm config set strict-ssl true**	

Now let's use the MTA build tool to create an MTA Archive(mtar) file for our project.

4. Create a target folder to store the mtar file that will get generated. **mkdir -p target** **ls -1**	
5. Invoke the build tool with the following. **mta --build-target XSA --mtar target/dat368_xsa.mtar build** Note: The mta build should take just over a minute.	

Exercise 2.2: Deploying the MTAR.

Notice that the mta builder pulls all the dependencies for NodeJS based modules as part of the mtar packaging process.

Now we are going to deploy our mtar file to the system using an mta extension file.

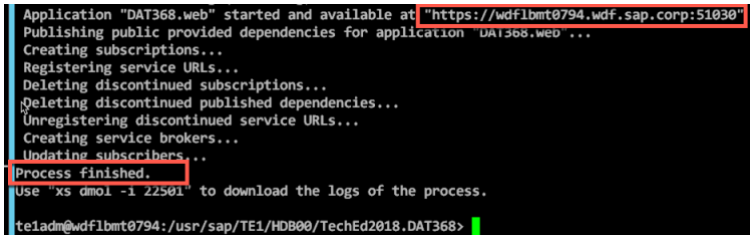
The Multi-Target Application approach is designed to allow a developer to specify much of the applications requirements and organization. However, when deploying it in different environments, the specifics of module naming, buildpack names, service names/ports, or the module's environment, etc. can vary. To allow flexibility after a developer has handed the application over to the dev-ops person, the MTA allows for an MTA extension file. This enables adjusting things without needing to modify the mta.yaml file and re-building the mtar package.

We use the --use-namespaces param to give our application modules more unique names and we use the --no-namespaces-for-services to make our service instances names simpler. The mtaext file gets merged with the mtad.yaml file that is derived from the mta.yaml file and specifies additional information for this particular deployment. We would then typically use a different mtaext file when deploying to Cloud Foundry. Exercise 4 illustrates this.

Time Check: If you are running short on time, skip to Exercise 3.

1. Deploy the mtar file.

xs deploy target/dat368_xsa.mtar --use-namespaces --no-namespaces-for-services -e deploy_xsa.mtaext



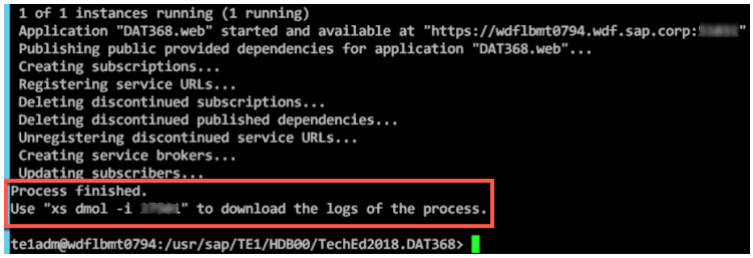
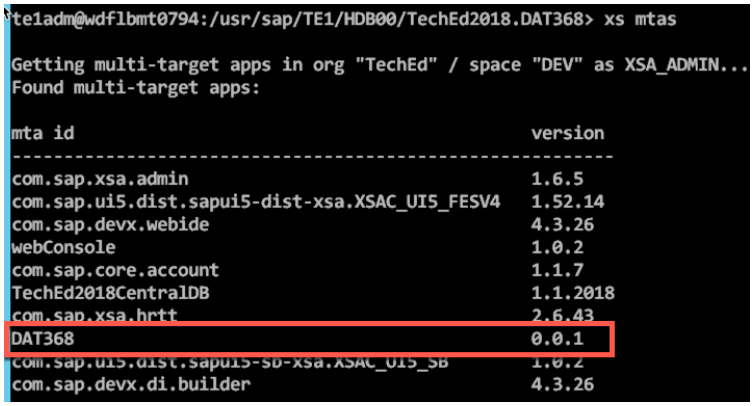
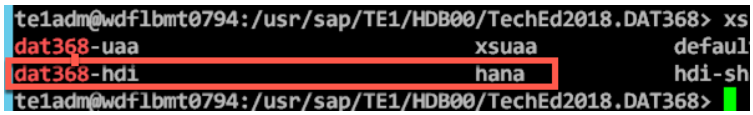
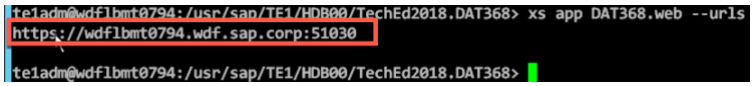
Watch the deploy process output carefully. It tells you what's going on behind the scenes.

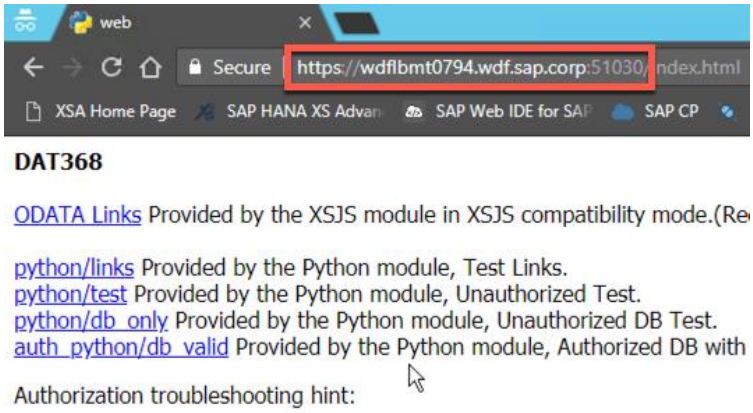
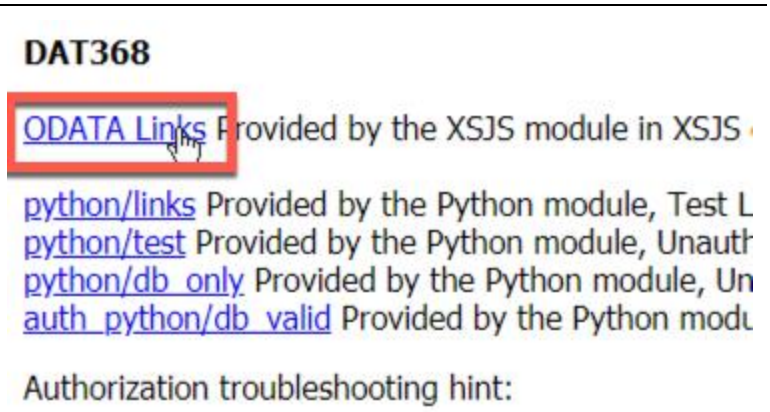
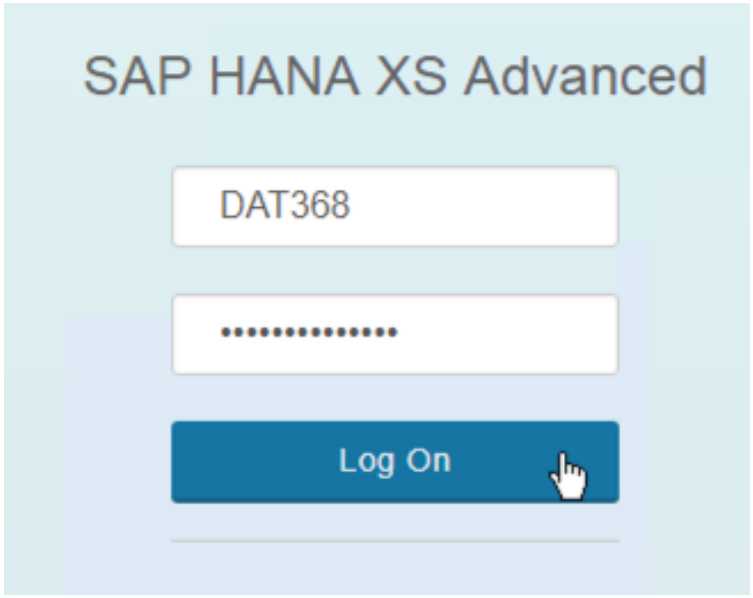
Some things to note. When you see this line.

Omitting npm install: node_modules directory is already present and rebuild is not enabled, using dependencies as provided

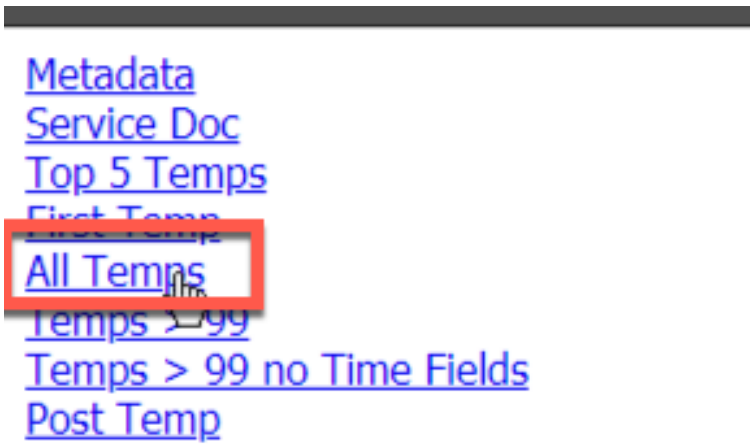
This lets you know that the dependencies were packages with the mtar and that the deployer will not run npm install again (and possibly fail if the Internet was not available).

The deploy will take approximately 10 minutes. You may want to take a break and stretch your legs at this time.

2. Verify the deploy completed successfully by seeing output similar to the following.	
3. Check that's it's listed in the mta list. **xs mtas**	
4. List it's app modules	
5. And it's services. **xs s | grep dat368** Notice that the HDI container has been re-created and that the uaa service still exists.	
6. Get the web module's URL. Note the port number may differ for you. **xs app DAT368.web --urls**	

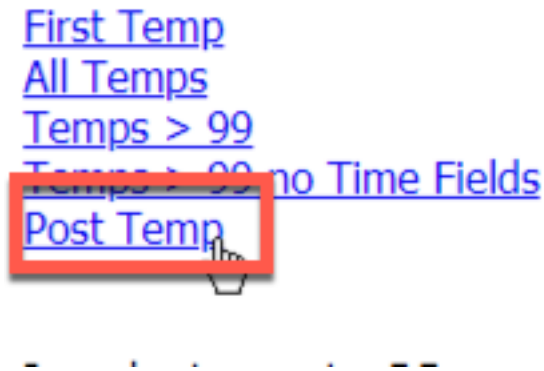
7. Cut the URL and paste it into a new Chrome incognito browser window.	
8. Click on the **ODATA Links** link. This will prompt you for login since the nodejs module requires authentication.	
9. Use **DAT368** with password **WelcomeSAP2018**	

10. This is a list of links that trigger various ODATA operations. Click the **All Temps** link to see the ODATA json result of the data in to table. Note that the results will be opened in a new tab in the browser.



11. Notice that there will initially only be one row as we noted above when we were exploring the data in the Web IDE.

12. User the browser's back button to return to the list of links. Click on the **Post Temp** link to invoke an AJAX script that inserts a random temp value into the table via an ODATA Post.



13. In a new browser tab, you'll see what the request looks like in the browser and the response(not pictured).	===Begin Request===
	
	POST to sensors.xsodata/temp
	Request Header: Content-Type = application/json
	Request Header: Accept = application/json
	
	Request Body:
	
	{
	"tempId": 0,
	"tempVal": 100,
	"ts": "/Date(1533578628614)/",
	"created": "/Date(1234567890123)/"
	}
	
	===End Request===

Note that none of the python links will work at this point since we haven't build the python module yet(You'll get a "Not Found" message. We'll get into that in the next exercise.

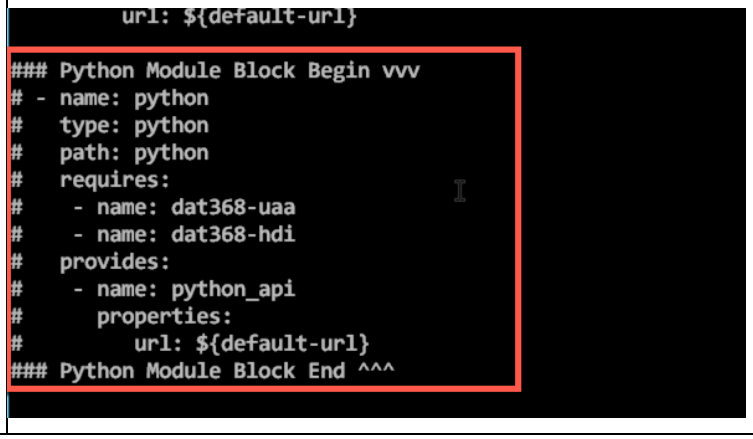
EXERCISE 3

Note: This exercise takes approximately 60 minutes to complete.

In this exercise we will turn our attention getting python running on our system and exercising the python runtime with our application project.

Exercise 3.1: Compile the Python Runtime

Return to the Putty console window.

Explanation	Screenshot
1. Examine the contents of the mta.yaml file by catenating it to the console. **cat mta.yaml** Scroll and notice that lines pertaining to python modules are commented out with #. This is to allow the project to work in the Web IDE while you may be editing other files, etc.	

At this point we would uncomment the python module related lines and adjust the application's app-router table to accommodate enabling the python module. In order to keep things moving and avoid typo's we will instead switch to another branch in the repo that has the python related changes already made.

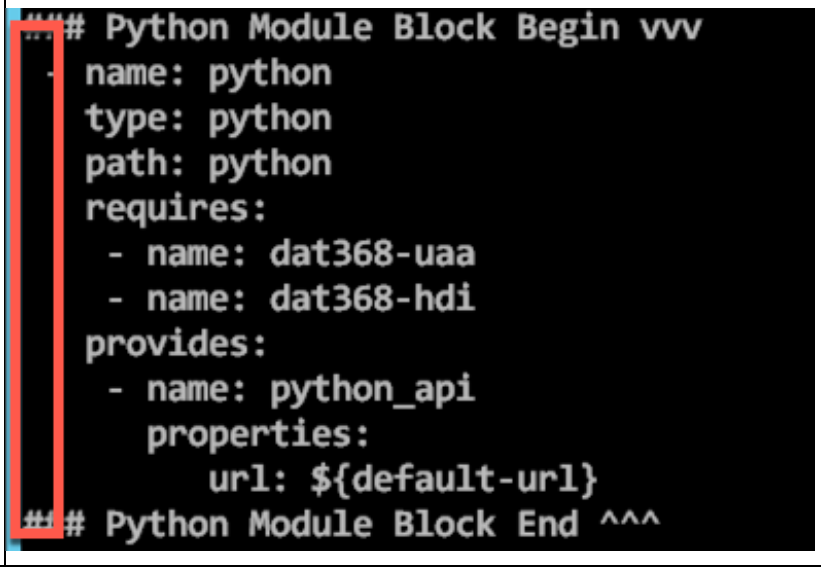
2. Switch to the solution branch.

git checkout solution

3. Again show the mta.yaml file.

cat mta.yaml

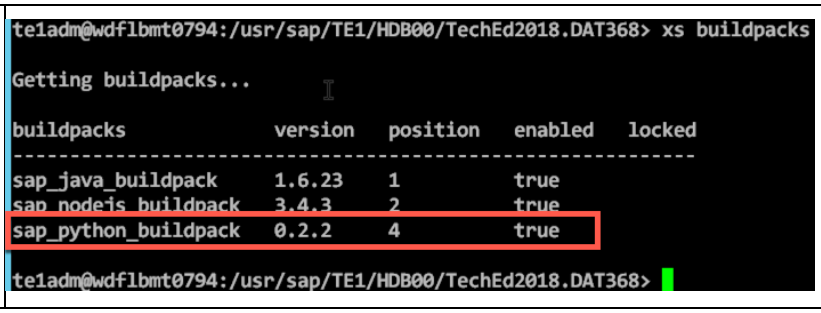
Notice that the comments have been removed.



The now enabled python module requires Python runtime version 3.6.5. This was the most current version at the time this workshop was created.
As of HANA2 SPS03, a python buildpack is available in XS Advanced. List the installed buildpacks with the xs buildpacks command.

4. Run the buildpacks command.

xs buildpacks



The buildpack coordinates the assembling of the native language files of a module with its dependencies and runtime environment. While runtimes are available for java and nodejs, by default there are no python runtimes available.

5. Run the runtimes command.

xs runtimes

Notice that while a buildpack exists for coordinating python module deployment, a runtime for python doesn't exist yet.

```
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00/TechEd2018.DAT368> xs runtimes

Getting runtimes...

type            version   id   resolved   active   description                                  bound apps
-------------------------------------------------------------------------------------------------------------
hanajdbc1       120.35    7    true       true     SAP HANA JDBC Driver 1.120.35                0
hanajdbc2       3.33      6    true       true     SAP HANA JDBC Driver 2.3.33                  6
node6.14        0.1       5    true       true     Node.js 6.14.0.1 for Linux x86-64            5
node8.9         3.6       4    true       true     Node.js 8.9.3.6 for Linux x86-64             19
sapjvm8         1.38      0    true       true     SAP JVM 8 Patchlevel 38 for Linux x86-64     1
sapjvm8_jre     1.38      3    true       true     SAP JVM JRE 8 Patchlevel 38 for Linux x86-64 8
tomcat8         5.23      1    true       true     Apache Tomcat Web Container 8.5.23           9
tomee1.7_jaxrs  5         2    true       true     Apache TomEE jaxrs 1.7.5                     0

te1adm@wdflbmt0794:/usr/sap/TE1/HDB00/TechEd2018.DAT368>
```

Normally you would download the python source using wget, but we have done that for you.

wget https://www.python.org/ftp/python/3.6.5/Python-3.6.5.tgz

6. Change up to the parent directory and list the files.

cd ..
ls -1

```
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00/TechEd2018.DAT368> cd ..
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00> ls -1
HDB
HDBAdmin.sh
HDBSettings.csh
HDBSettings.sh
Python-3.6.5.tgz
TechEd2018.DAT368
XS_PYTHON00_0-70003433.ZIP
backup
del_DAT368.sh
exe
hdbenv.csh
hdbenv.sh
set_python_env.sh
wdflbmt0794.wdf.sap.corp
work
xterms
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00>
```

7. Untar the tar-zipped source file.

tar xzvf Python-3.6.5.tgz

```
Python-3.6.5/Objects/object.c
Python-3.6.5/Objects/abstract.c
Python-3.6.5/Objects/listobject.c
Python-3.6.5/Objects/bytes_methods.c
Python-3.6.5/Objects/dictnotes.txt
Python-3.6.5/Objects/typeslots.inc
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00>
```

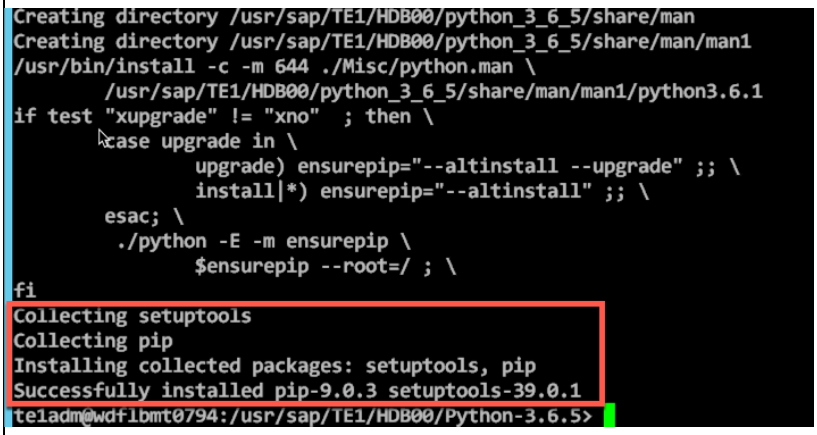
8. Create a folder that will hold the compiled python files, change into the Python source folder and run the configure script.

md python_3_6_5

cd Python-3.6.5

./configure --prefix=/usr/sap/TE1/HDB00/python_3_6_5/ --exec-prefix=/usr/sap/TE1/HDB00/python_3_6_5/ ; make -j4 ; make altinstall

This will take about 3 minutes to complete.



You should see this at the end of the output.

Successfully installed pip-9.0.3 setuptools-39.0.1

In order to perform this compilation, the server must have various development tool packages. This has already been done in this case, but your server may need to have them installed. The following is a list for your reference based on a SuSE Linux system.

zypper search -t pattern
zypper search -t pattern | grep Devel

zypper install --type pattern Basis-Devel **(182 Packages)**

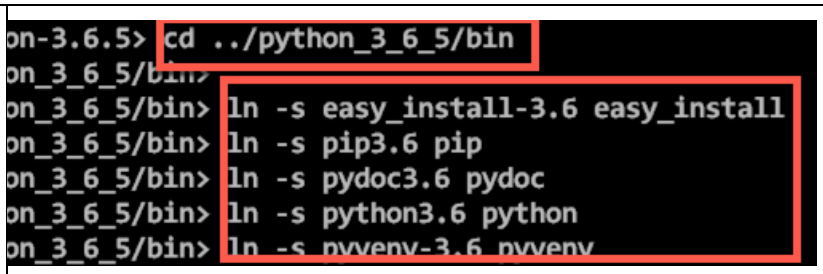
zypper install tk-devel
zypper install tcl-devel
zypper install libffi-devel
zypper install openssl-devel
zypper install readline-devel
zypper install sqlite3-devel
zypper install ncurses-devel
zypper install xz-devel
zypper install zlib-devel

The compilation left versions of the executabules with version numbers. Most tools expect the binaries to not have them. We'll fix this by creating some symbolic links to unversioned names.

9. Change into our python target directory and create the following symbolic links.

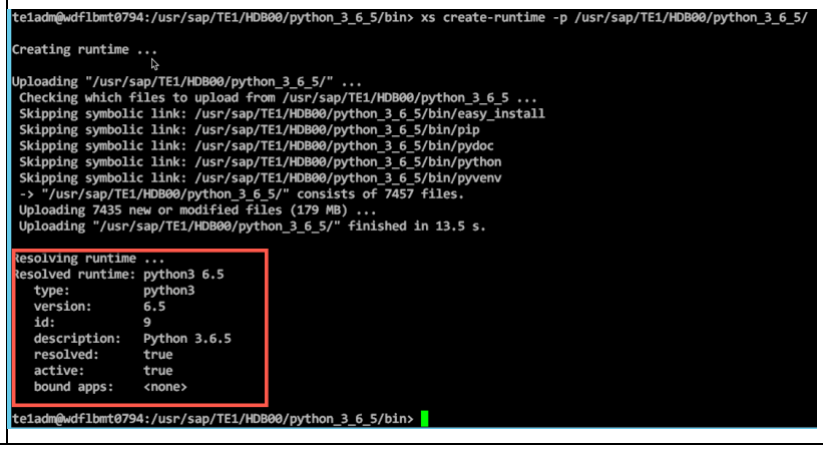
cd ../python_3_6_5/bin

ln -s easy_install-3.6 easy_install
ln -s pip3.6 pip
ln -s pydoc3.6 pydoc
ln -s python3.6 python
ln -s pyvenv-3.6 pyvenv



10. Now that python has been prepared, install it into the system as a runtime.

xs create-runtime -p /usr/sap/TE1/HDB00/python_3_6_5/



For the purposes of the TechEd workshop the following note doesn't apply. Continue with the numbered step.

Note: **Only when setting up python on your own server**: If you find that the pip command below fails with an inablility to import the _socket library, it's because the configure/build process under some variations of linux leaves some important libraries in an unexpected location. Change into the directory where the target python was installed.

cd python_3_6_5

Copy the files in the lib64 folder into the lib folder

cp -avp lib64/* lib

Uninstall the runtime. By first finding it's ID and then deleting it.

xs runtimes
xs delete-runtime -i <id> -f

Re-create it.

xs create-runtime -p python_3_6_5

11. Check that it installed correctly.

xs runtimes

```
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00/python_3_6_5/bin> xs runtimes

Getting runtimes...

type            version   id   resolved   active   description                                  bound apps
-------------------------------------------------------------------------------------------------------------
hanajdbc1       120.35    7    true       true     SAP HANA JDBC Driver 1.120.35                0
hanajdbc2       3.33      6    true       true     SAP HANA JDBC Driver 2.3.33                  6
node6.14        0.1       5    true       true     Node.js 6.14.0.1 for Linux x86-64            5
node8.9         3.6       4    true       true     Node.js 8.9.3.6 for Linux x86-64             10
python3         6.5       9    true       true     Python 3.6.5                                 0
sapjvm8         1.38      0    true       true     SAP JVM 8 Patchlevel 38 for Linux x86-64     1
sapjvm8_jre     1.38      3    true       true     SAP JVM JRE 8 Patchlevel 38 for Linux x86-64 8
tomcat8         5.23      1    true       true     Apache Tomcat Web Container 8.5.23           9
tomee1.7_jaxrs  5         2    true       true     Apache TomEE jaxrs 1.7.5                     0

te1adm@wdflbmt0794:/usr/sap/TE1/HDB00/python_3_6_5/bin>
```

12. Move back up to the home directory.

cd ../..

```
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00/python_3_6_5/bin> cd ../..
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00>
```

Exercise 3.2: Set up Python Libraries

SAP provides a set of python libraries for validating requests, connecting to HANA DB, etc. These are typically provided by downloading them from support site.

https://launchpad.support.sap.com/#/softwarecenter/search/XS_PYTHON

To save trouble, they have been download for you.

Unzip the libraries into a directory called sap_dependencies (matches the official docs)

Note: We will be using two of the SAP supplied python libraries (sap_xssec and hdbcli). There are some additional libraries they you might consider, but we won't be covering their use in these exercises.

sap_instance_manager (doc page)
sap_audit_logging (doc page)
sap_cf_logging (doc page) Cloud Foundry specific logging.

We would like to see an XSA service for application logging that follows the ELK stack compatible with the Cloud Foundry app-logging service, but this is not available as of the creation of this workshop.

13. Unzip

unzip XS_PYTHON00_0-70003433.ZIP -d sap_dependencies

```
  inflating: sap_dependencies/hdbcli-2.3.14-cp36-cp36m-linux_x86_64.
  inflating: sap_dependencies/hdbcli-2.3.14-cp36-cp36m-macosx_10_6_x
  inflating: sap_dependencies/hdbcli-2.3.14-cp36-cp36m-win_amd64.whl
  inflating: sap_dependencies/README.md
  inflating: sap_dependencies/sap_audit_logging-0.3.0-py2.py3-none-a
  inflating: sap_dependencies/sap_instance_manager-0.2.1-py2.py3-nor
  inflating: sap_dependencies/sap_py_jwt-1.1.0-py2.py3-none-any.whl
  inflating: sap_dependencies/sap_xssec-1.1.5-py2.py3-none-any.whl
  inflating: sap_dependencies/SIGNATURE.SMF
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00>
```

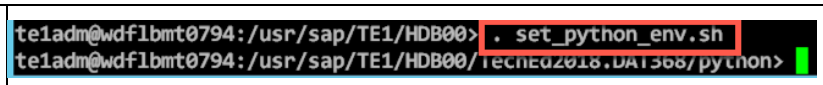
When your module is deployed, it runs in its own isolated container with this specific version of python available to it.

You can develop and test your python module locally before deploying by setting up the python environment to use the version of python just installed.

14. Run this command to set up the environment.

. set_python_env.sh

Note: That's dot space set_python_env.sh



This will relocate you into the python module of your project.

Create a vendor directory to hold the python libraries your module needs.

As mentioned in the presentation, it's important to bring all the libraries that your python module requires into the project so that you can control the versions of the python dependencies.

15. Run mkdir.

mkdir -p vendor

The python dependencies are specified in the requirements.txt file in the project. While most of these dependencies will be provided by the public python repository PyPi some are SAP libraries made available when you unzipped the XS_PYTHON00_0-70003433.ZIP file above. This is why specifying the --find-links is important, otherwise the SAP libraries won't be found when trying to gather all the needed libraries and their dependencies.

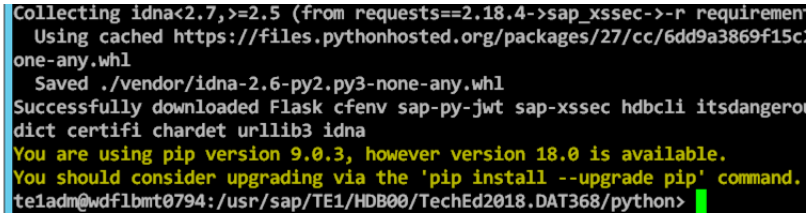
We use a folder called vendor because the python buildpack looks for this folder and assumes that everything the python module needs will be found in there. During deployment, the deployer will not make any additional attempts to find missing dependencies and will fail otherwise. This is an important thing to consider when working with python modules.

Use the python packaging tool(pip) to download the dependencies found in the requirements.txt

16. Run pip.

pip download -d vendor -r requirements.txt --find-links ../../sap_dependencies

You can safely ignore the yellow upgrade warning.



17. See what pip downloaded, change into the vendor folder.

cd vendor

ls -1

You should see a bunch of files with .whl .gz extensions.

```
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00/TechEd2018.DAT368/python> cd vendor
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00/TechEd2018.DAT368/python/vendor> ls -1
Flask-1.0.2-py2.py3-none-any.whl
Jinja2-2.10-py2.py3-none-any.whl
MarkupSafe-1.0.tar.gz
Werkzeug-0.14.1-py2.py3-none-any.whl
certifi-2018.4.16-py2.py3-none-any.whl
cfenv-0.5.3-py2.py3-none-any.whl
chardet-3.0.4-py2.py3-none-any.whl
click-6.7-py2.py3-none-any.whl
furl-1.2.tar.gz
hdbcli-2.3.14-cp36-cp36m-linux_x86_64.whl
idna-2.6-py2.py3-none-any.whl
itsdangerous-0.24.tar.gz
orderedmultidict-1.0.tar.gz
requests-2.18.4-py2.py3-none-any.whl
sap_py_jwt-1.1.0-py2.py3-none-any.whl
sap_xssec-1.1.5-py2.py3-none-any.whl
six-1.11.0-py2.py3-none-any.whl
urllib3-1.22-py2.py3-none-any.whl
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00/TechEd2018.DAT368/python/vendor>
```

Exercise 3.3: Build the Python Module and Redeploy

Now when we run the mta builder again, it will bundle these python dependencies up with the rest of your application.

18. Return to the project directory.

cd ../..

Run the following commands to build and deploy your project's mtar again(or for the first time if you skipped here to save time).

```
DB00/TechEd2018.DAT368/python/vendor> cd ../..
DB00/TechEd2018.DAT368>
```

19. Run the mta command again.

mta --build-target XSA --mtar target/dat368_xsa.mtar build

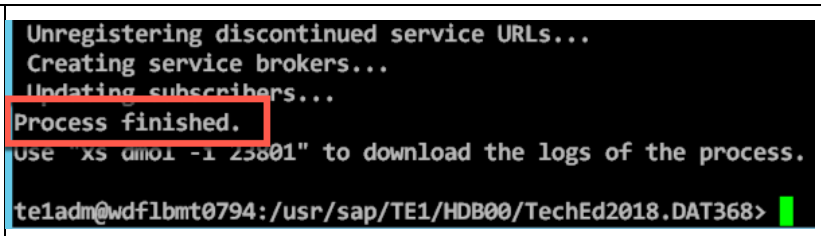
Notice that the python folder is just zipped and no dependencies are pulled in during the mtar assembly. Also the nodejs npm commands don't take as much time since they've already pulled their dependencies. This can take a couple minutes to complete.

```
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00/TechEd2018.DAT368> mta --build-target XSA --mtar target/dat368_xsa.mtar build
MTA Build Starting
SAP Multitarget Application Archive Builder 1.1.2
Module "db": /hana/shared/TE1/HDB00/TechEd2018.DAT368/db/package.json already exists
Module "python": zipping directory python
Module "xsjs": invoking npm
Module "web": invoking npm
Module "db": command output
>  npm WARN deploy@ No description
>  npm WARN deploy@ No repository field.
>  npm WARN deploy@ No license field.
Module "db": zipping directory db
Module "web": command output
>  npm WARN web-approuter@ No description
>  npm WARN web-approuter@ No repository field.
>  npm WARN web-approuter@ No license field.
Module "web": zipping directory web
Module "xsjs": command output
>  npm WARN xsjs@1.0.0 No repository field.
>  npm WARN xsjs@1.0.0 No license field.
Module "xsjs": zipping directory xsjs
Generating archive /hana/shared/TE1/HDB00/TechEd2018.DAT368/target/dat368_xsa.mtar
Done
MTA Build Finished
te1adm@wdflbmt0794:/usr/sap/TE1/HDB00/TechEd2018.DAT368>
```

20. And run the deploy command again.

xs deploy target/dat368_xsa.mtar --use-namespaces --no-namespaces-for-services -e deploy_xsa.mtaext

Note this will take about 12 minutes this time so you may want to take another break.



21. At this point you should see your application's modules.

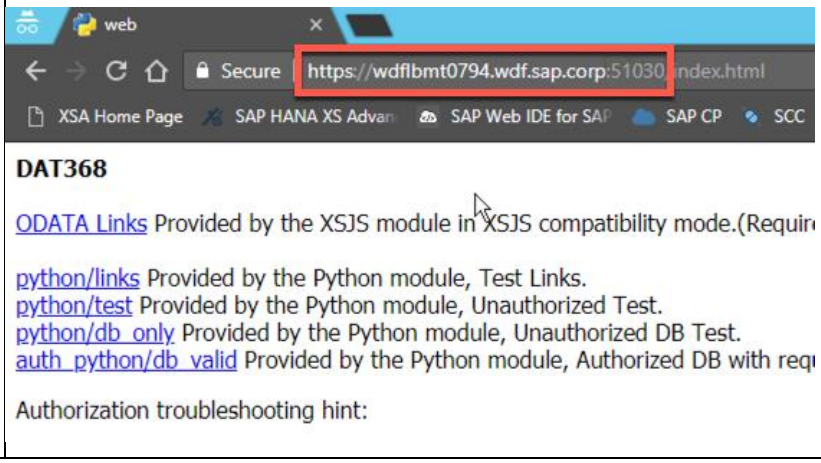
xs a | grep DAT368

This includes your python module. Also check that all but the db module are running.

22. Get the app-router URL.

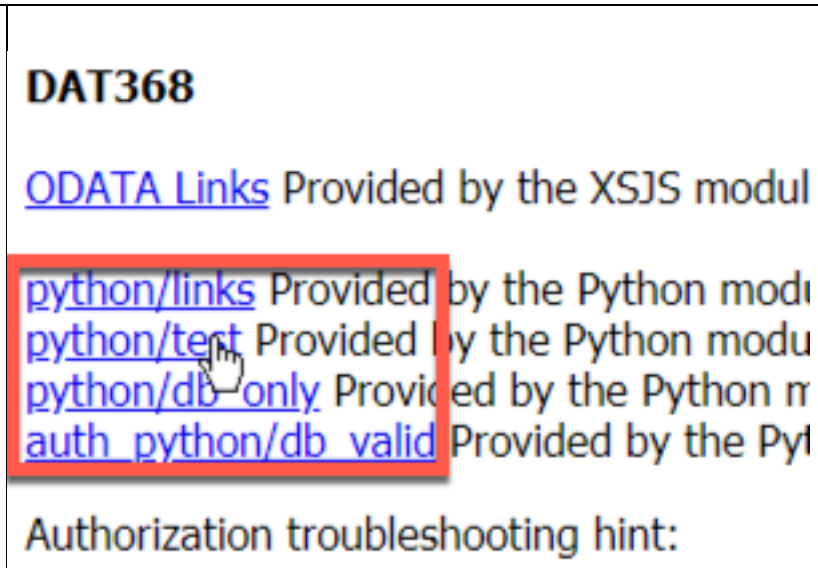
xs app DAT368.web --urls

Paste it an incognito browser window.

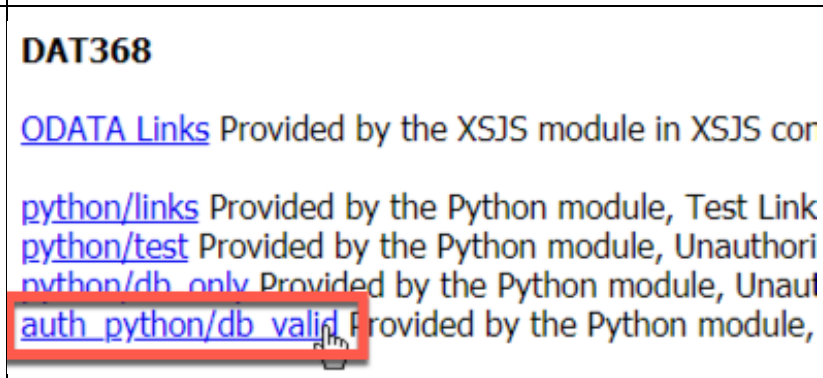


You can click on the ODATA Links which will open up a new browser tab and present you with a list of links to trigger various ODATA operations. If you want to quickly add a few lines of data to the database, click on the Post Temp link and reload the page a few times. This will add rows of data to the table with random temperature values.

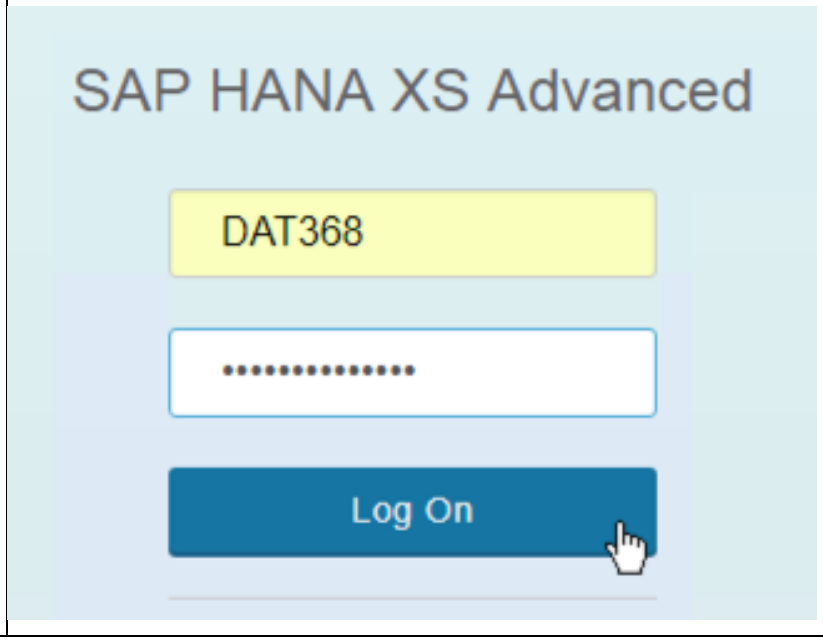
23. Click on any of the python links. The first three don't require authorization. Use the browser's back button to return to this list.



24. Click on the last link **auth_python/db_valid**.



25. This link requires authorization. Enter **DAT368** with password **WelcomeSAP2018**

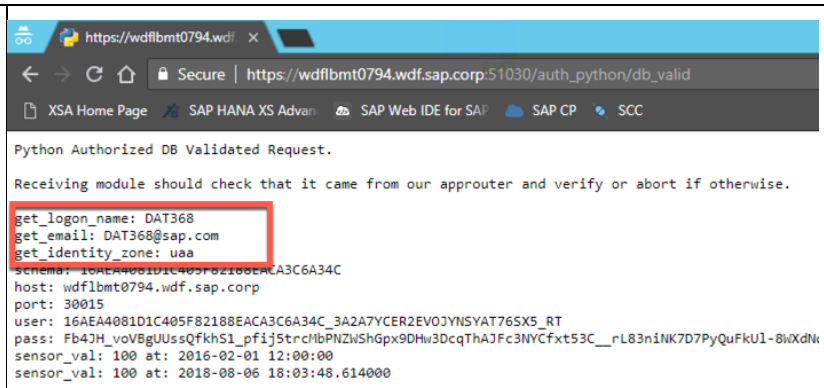


26. Note the following output.

See that the user is authorized for this operation.

Here is the code that does the authorization check.

Notice that the module will abort if the proper authorization isn't provided in the request.

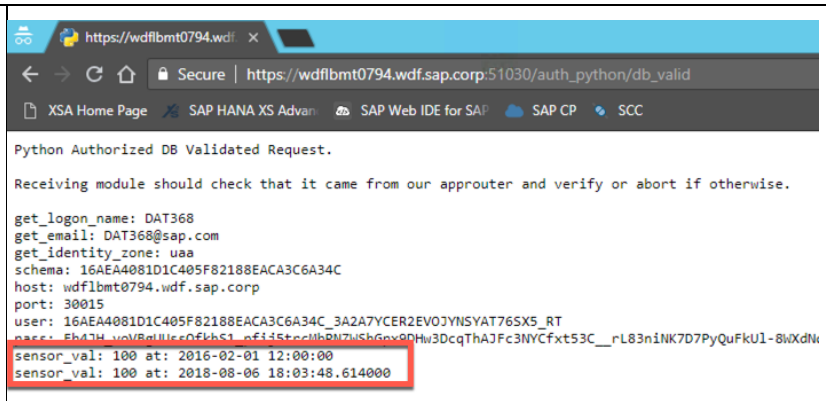


Here is an excerpt of the server.py code with the request verification code in yellow.

If there is a request for a auth_python/db_valid, return Testing message and then check JWT and connect to the data service and retrieve some data

```
@app.route('/auth_python/db_valid')
def auth_db_valid():
    output = 'Python Authorized DB Validated Request. \n'
    output += '\n'
    output += 'Receiving module should check that it came from our approuter and verify or abort if otherwise.\n'
    output += '\n'
    svcs_json = str(os.getenv("VCAP_SERVICES", 0))
    svcs = json.loads(svcs_json)

    # Verify the JWT before proceeding. or refuse to process the request.
    # https://jwt.io/ JWT Debugger Tool and libs for all languages

    uaa_service = env.get_service(label='xsuaa').credentials
    access_token = request.headers.get('authorization')[7:]

    security_context = xssec.create_security_context(access_token, uaa_service)
    isAuthorized = security_context.check_scope('openid')
    if not isAuthorized:
        abort(403)

...
```

27. Also notice that the python module had connected to the DB and returned the rows of data from the table. The code that follows illustrates this.




```
# If there is a request for a auth_python/db_valid, return Testing message and then check JWT and connect to the data service and retrieve some data

@app.route('/auth_python/db_valid')
def auth_db_valid():

...


#   # Connect to the python HANA DB driver using the connection info
    connection = dbapi.connect(host,int(port),user,password)
#   # Prep a cursor for SQL execution
    cursor = connection.cursor()
#   # Form an SQL statement to retrieve some data
    cursor.execute('SELECT "tempId", "tempVal", "ts", "created" FROM "' + schema +
'"."DAT368.db.data::sensors.temp"')
#   # Execute the SQL and capture the result set
    sensor_vals = cursor.fetchall()

#   # Loop through the result set and output
    for sensor_val in sensor_vals:
        output += 'sensor_val: ' + str(sensor_val[1]) + ' at: ' + str(sensor_val[2]) + '\n'
#
#   # Close the DB connection
    connection.close()
#
    # Return the results
    return output
```

EXERCISE 4 (OPTIONAL)

Note: This exercise takes approximately 30 minutes to complete.

Deploying to Cloud Foundry is actually not much different from deploying to XSA as long as you pay attention to MTA best practices and abstract away specifics of the deployment environment. You must also keep in the back of your mind that in Cloud Foundry based landscapes you are in a shared environment and that this situation has implications for the uniqueness of naming of urls and role definitions and authentication mechanisms.

In the case of this workshop we will use a single Cloud Foundry account that will be shared across all participants. The reason this isn't a problem is because each participant will deploy into a unique space. All the isolation is provided by the HDI deploy process. However, there are a few things you will need to customize in order to avoid name collisions. We recommend that you replicate the workshop project on your own HANA Express system and adjust it to deploy with your own SAP Cloud Platform Cloud Foundry account when you get the opportunity.

Note: In the following instructions, xx or XX has been replaced with a variable ${space} that will be resolved by the deployer as the current space name. This resolved the name collisions issue just mentioned.

https://www.sap.com/developer/topics/sap-hana-express.html

At the time this workshop was created, the HANA database could be made available in SAP Cloud Platform Cloud Foundry in one of two ways. The account could be provisioned with a HANA database instance or you as the account owner could create a HANA as a Service(HaaS) instance yourself. When you use provisioned HANA instances the python libraries will work properly because these instances don't require the connection to HANA to be encrypted. However, when you use HaaS created instances, they are configured to require encrypted connections. The python libraries provided as of this date don't have the ability to create encrypted connections. In order to overcome this limitation a more updated python library must be used.

Exercise 4.1: Build and Deploy to Cloud Foundry

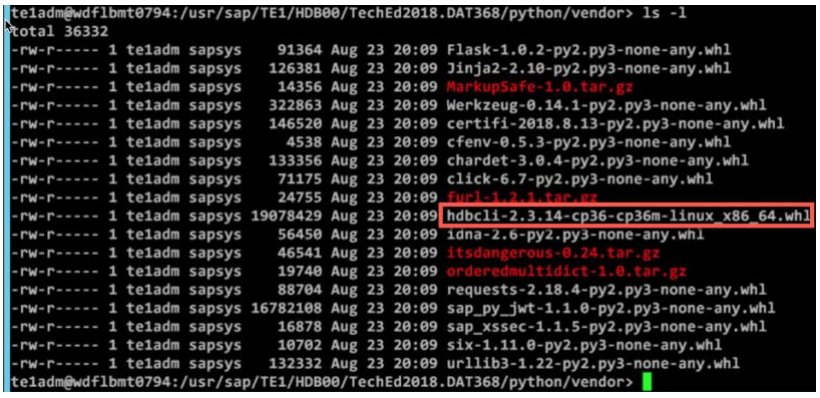
Return to the Putty console window.

Explanation	Screenshot
1. Change into the python/vendor folder. **cd python/vendor**	te1adm@wdflbmt0794:/usr/sap/TE1/HDB00/TechEd2018.DAT368> cd python/vendor te1adm@wdflbmt0794:/usr/sap/TE1/HDB00/TechEd2018.DAT368/python/vendor>

2. List the library package files.

ls -l

Notice the hdblci-...whl file is version 2.3.14



3. Delete this file.

rm -f hdbcli-2.3.14-cp36-cp36m-linux_x86_64.whl

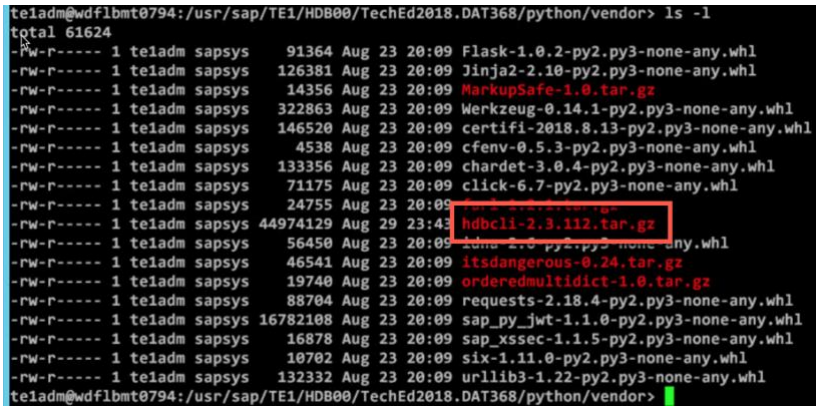
4. Copy the newer python lib file from the update folder to the current folder.

cp ../../update/hdbcli-2.3.112.tar.gz .

Don't worry that the new file has an extension of tar.gz instead of .whl, both variations are allowed.

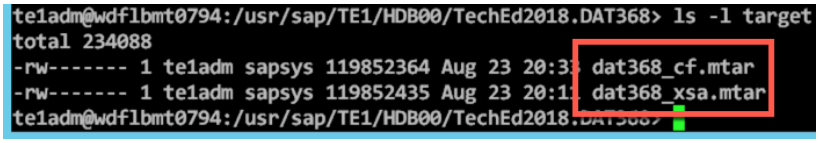
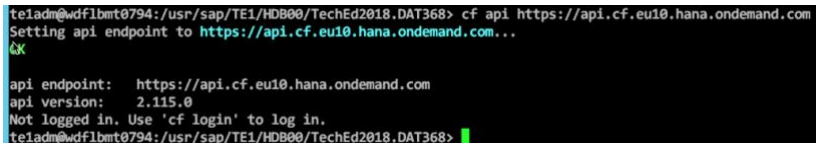
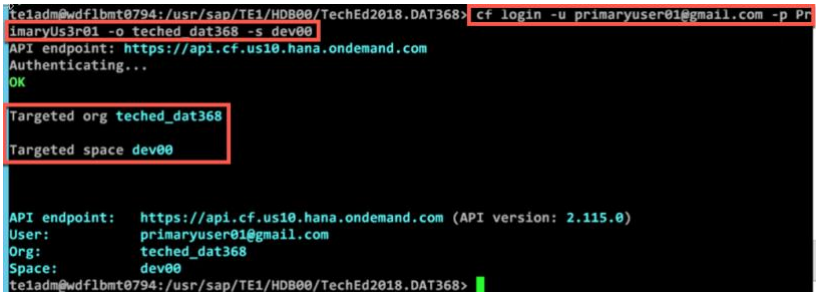
5. Verify that the 2.3.112 version is available and the 2.3.14 version is deleted.

ls -l



6. Change back to the project folder.

cd ../..

7. Run the MTA command again but this time specify that the output is targeted to Cloud Foundry. **mta --build-target CF --mtar target/dat368_cf.mtar build**	
8. See that there are now 2 mtar files in the target folder. **ls -l target**	
9. Connect to the US10 Cloud Foundry landscape. **cf api https://api.cf.us10.hana.ondemand.com**	
10. Log in as this user. User: **primaryuser01@gmail.com** Password: **PrimaryUs3r01** **cf login -u primaryuser01@gmail.com -p PrimaryUs3r01 -o teched_dat368 -s dev00**	
11. Target the space that corresponds to the number you were assigned at the beginning of this workshop by replacing the "XX" with your number. If you don't know your number, raise your hand and ask for assistance **cf t -s devXX**	
12. Check to see that a service called dat368-hdi is available in this space. **cf s**	

When deploying to Cloud Foundry, you need to coordinate your application with the space and sub-account that you are deploying into. This is done in the deploy_cf.mtaext file that is passed into the deploy operation.

Note: In the following instructions, xx or XX has been replaced with a variable ${space} that will be resolved by the deployer as the current space name.

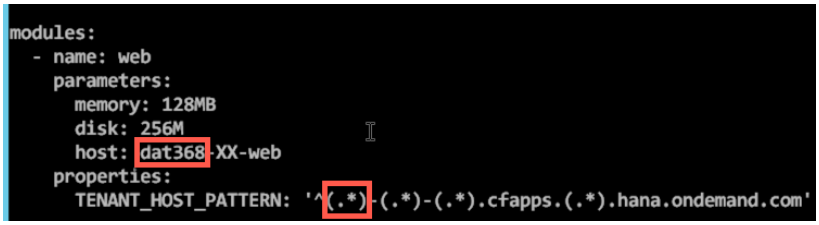
In this case, we are deploying into a sub-account that has a sub-domain called dat368. The url endpoint of the approuter module(dat368-xx-web.cfapps.us10.hana.ondemand.com) must include the sub-domain name. This is enabled with a special environment variable called TENANT_HOST_PATTERN which contains a regular expression. The first matching group needs to match the same string as the sub-domain.

^(.*)-(.*)-(.*).cfapps.(.*).hana.ondemand.com

dat368-00-web.cfapps.eu10.hana.ondemand.com

13. Examine the deploy_cf.mtaext file to confirm that the regular expression matches and that the host parameter of the web module starts with dat368.

cat deploy_cf.mtaext



As mentioned above, in order to avoid resource name collisions we need to edit the deploy_cf.mtaext so that once the mtar is deployed into the Cloud Foundry account, it will have unique urls.

If you've forgotten your two digit number, raise your hand and we'll help you find it.

The deploy_cf.mtaext file includes the space name that includes your two digit number.

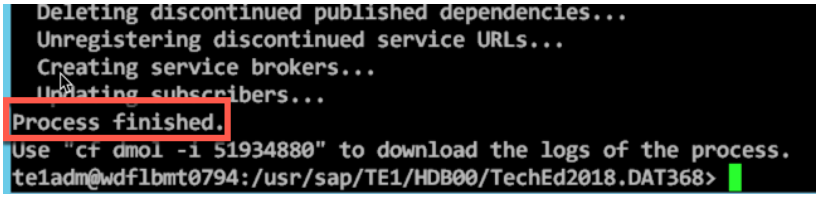
14. Now build the mtar file but this time targeting Cloud Foundry.

mta --build-target CF --mtar target/dat368_cf.mtar build

Note: The following deploy operation will take approximately 12 minutes. The initial file upload takes at nearly 3 minutes without displaying any progress indication so don't worry that the process has hung. Once you see some output from the deploy process you may want to take a break and return in about 10 minutes.

15. Deploy the mtar using your modified deploy_cf.mtaext file.

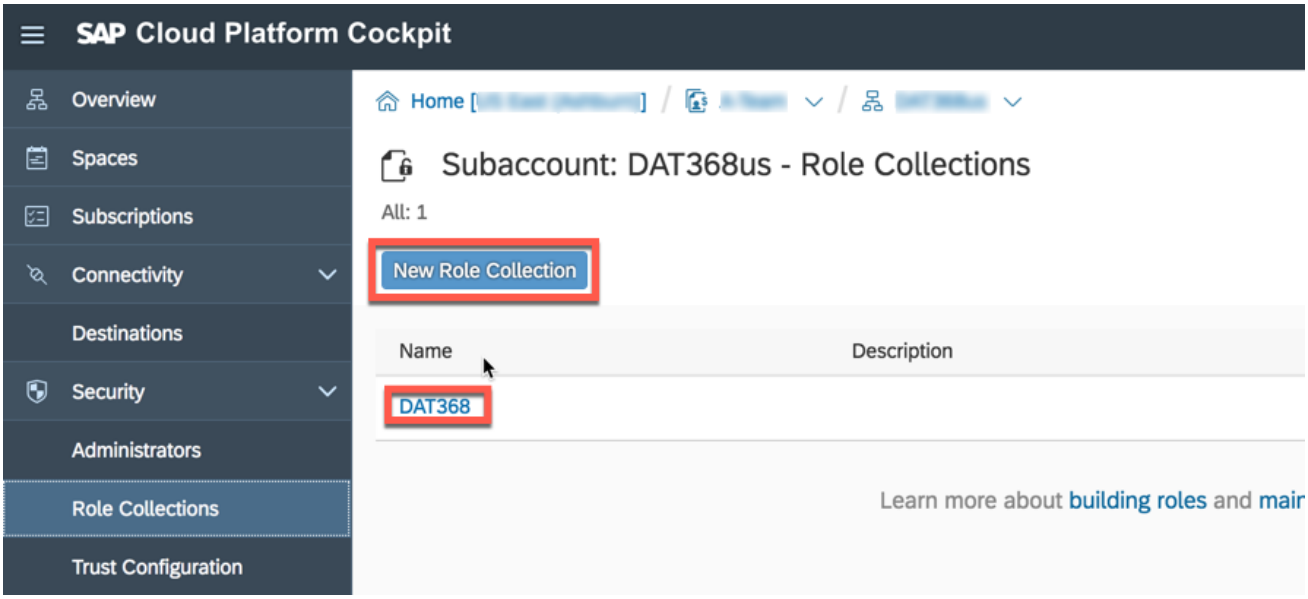
cf deploy target/dat368_cf.mtar --use-namespaces --no-namespaces-for-services -e deploy_cf.mtaext



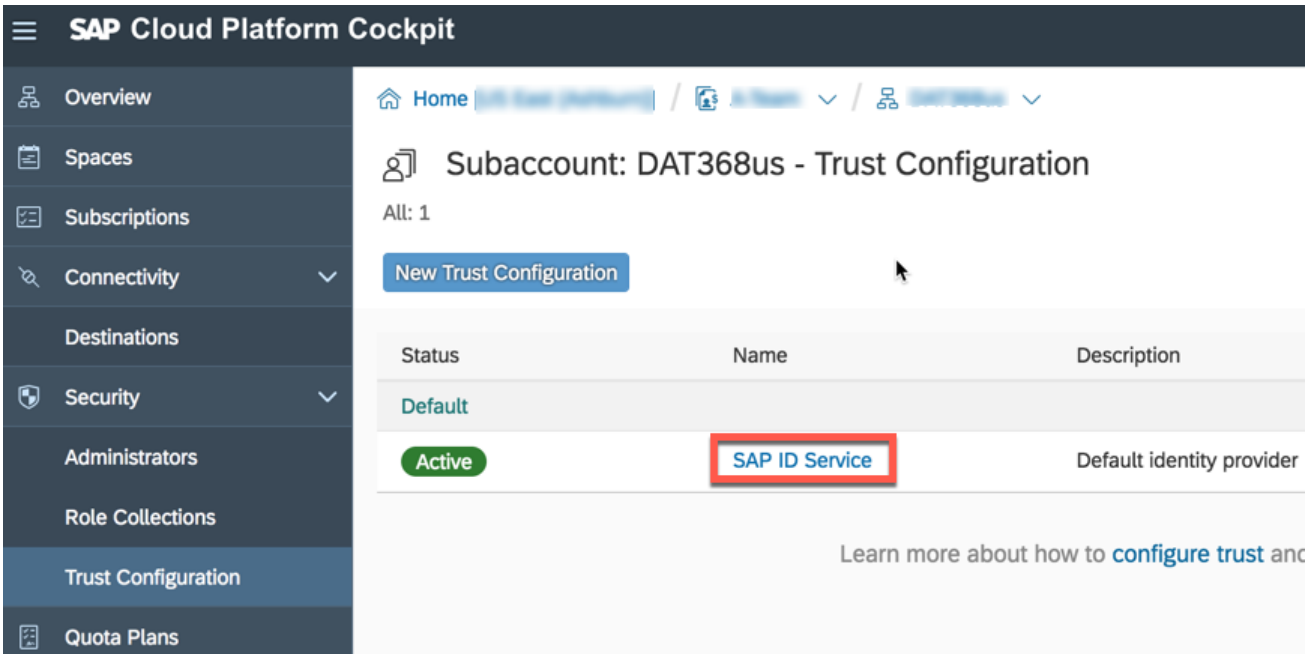
Once deployed, the system will understand the roles that your application has defined. As we did earlier, you would need to log into your Cloud Foundry account and create role collections and assign roles to them. You then would need to assign the created role collection to your users. Because we can't give you access to the Cloud Foundry account we're using, we will have to do this for you.

If you get to this point in the exercise, raise your hand and a facilitator will add your role to the role collection for the primaryuser01@gmail.com user.

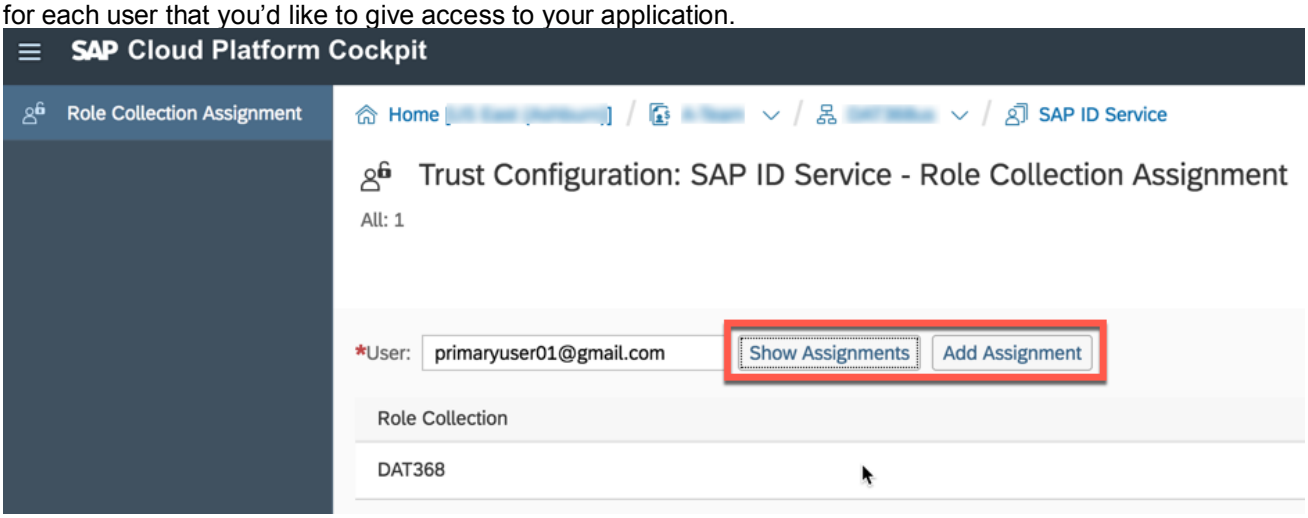
If you were doing this in your own account, you would first create a new DAT368 role collection and add the DAT368Manager and DAT368_XX User roles to it(not pictured).



Then you would go into Trust Configuration and select the SAP ID Service.



And then you would explicitly enter the username and select “Show Assignments” and then “Add Assignment” for each user that you’d like to give access to your application.

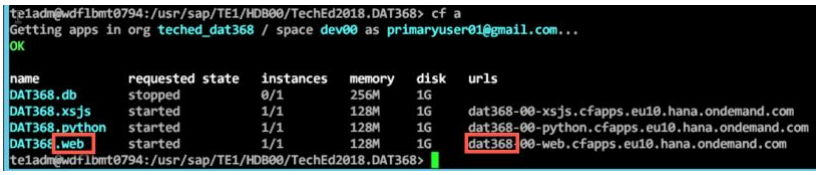


This process will be somewhat different if you have defined your own Identity Provider as you would create a group to role collection mapping instead. This is beyond the scope of this exercise.

...continuing.

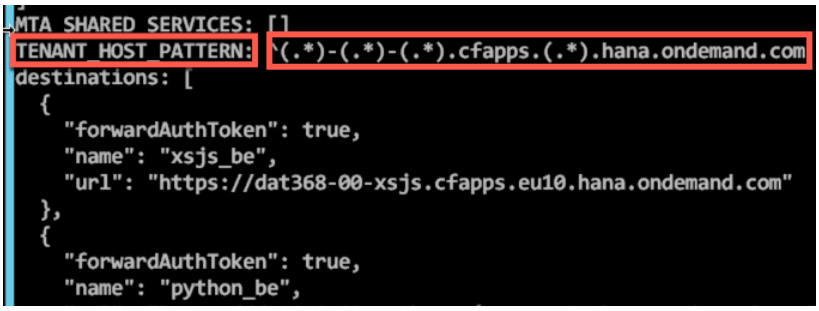
16. Verify that the url of the approuter(web) module starts with dat368.

cf a



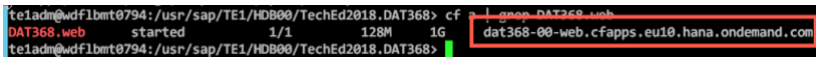
17. Also verify that the approuter(web) module has the TENANT_HOST_PATTERN regex set properly.

cf env DAT368.web

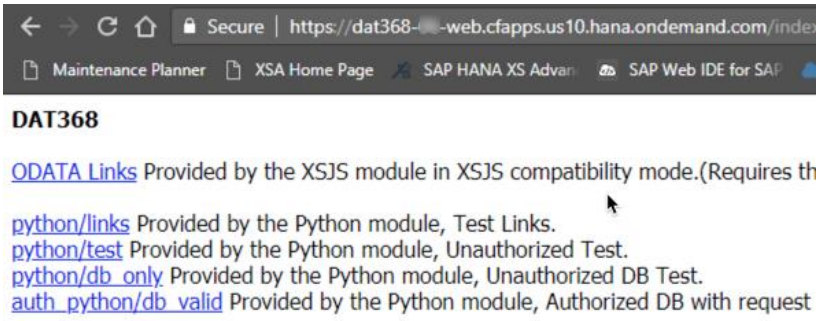


18. Get the url of the approuter(web) module.

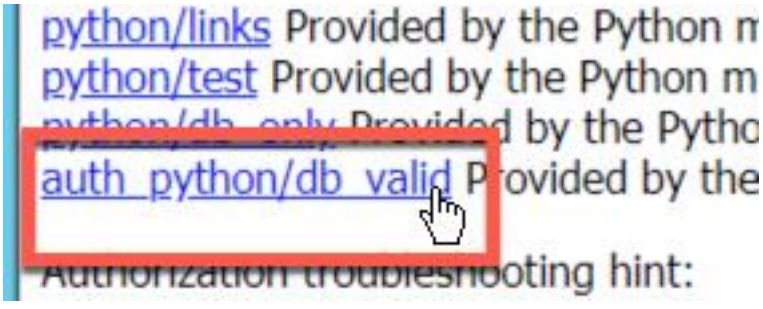
cf a | grep DAT368.web



19. Paste it into an incognito browser window.



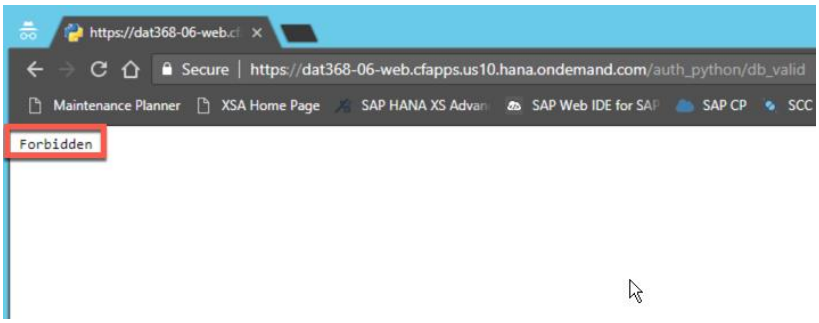
20. Click on the auth_python/db_valid link.



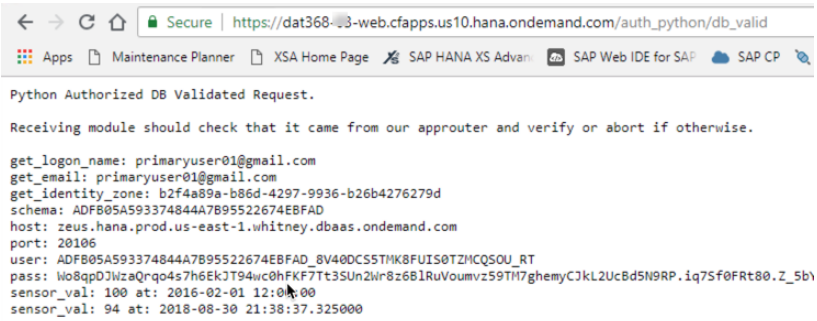
21. Login with this.

User: **primaryuser01@gmail.com**
Password: **PrimaryUs3r01**

22. If you get a white screen with "Forbidden" it's because the facilitator hasn't yet added your application's role to the role collection for the user. Raise your hand and ask that your role be added.



23. Verify that the user is logged in and that output similar to this is returned.



This completes the Cloud Foundry deployment exercise.

Conclusion:

While this example use of python is very simple, it illustrates the two most important functions that any module should provide.

1. Insure that the request that is being serviced comes from your application's app-router and contains the needed authorizations.

2. If needed, connect to the DB in order to read data to be processed and write the results back.

Another thing that is commonly done in a python application module is to log the module's activity to either the audit log and/or application log service. While we didn't cover this in these exercises, you can find details here.

sap_instance_manager (doc page)
sap_audit_logging (doc page)
sap_cf_logging (doc page) Cloud Foundry specific logging.

This concludes the python wrangling for today.

Please fill out the survey in the TechEd mobile app for this workshop session.