Python Wrangling for SAP HANA Application Developers

DAT-368

Exercises / Solutions  
Andrew Lunde / SAP

Table of ContentS

[BEFORE YOU START 3](#_Toc522721538)

[Getting Help 3](#_Toc522721539)

[Source Code Solutions 3](#_Toc522721540)

[Exercise 1 4](#_Toc522721541)

[Exercise 1.1: MTA Project – WebIDE Usage and Limitations 4](#_Toc522721542)

[Exercise 1.2: Using the XS CLI 12](#_Toc522721543)

[Exercise 1.3: Cloning the project into the server’s file space 16](#_Toc522721544)

[Exercise 1.4: Role Collection Setup 17](#_Toc522721545)

[Exercise 2 24](#_Toc522721546)

[Exercise 2.1: Building an MTAR 24](#_Toc522721547)

[Exercise 2.2: Deploying the MTAR. 26](#_Toc522721548)

[Exercise 3 30](#_Toc522721549)

[Exercise 3.1: Compile the Python Runtime 30](#_Toc522721550)

[Exercise 3.2: Set up Python Libraries 34](#_Toc522721551)

[Exercise 3.3: Build the Python Module and Redeploy 36](#_Toc522721552)

[Exercise 4 40](#_Toc522721553)

[Exercise 4.1: Compile the Python Runtime 40](#_Toc522721554)

[Conclusion: 41](#_Toc522721555)

# 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>

# 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 WebIDE 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.

## Exercise 1.1: MTA Project – WebIDE 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** |  |
| 1. 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/** |  |
| 1. 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. | |
| 1. Right-click on Workspace and select **Git->Clone Repository** |  |
| 1. Use the following URL.   **<https://github.com/alundesap/TechEd2018.DAT368.git>**  and click **Clone** |  |
| 1. 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. | |
| 1. Right-click on the mta.yaml file and select **Open With -> Code Editor**. |  |
| 1. 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. |  |
| 1. 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.** |  |
| 1. Select **Space**. |  |
| 1. And select **DEV** from the **Space** dropdown. Then click the **Save** button, then **Close**. |  |
| 1. 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. |  |
| 1. 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. | |
| 1. Click on the DB Explorer icon. |  |
| 1. Since no DB connections have yet been made, click **Yes**. |  |
| 1. Enter **DAT368** in the HDI Containers field to narrow the list, and then select the line that starts with DAT368. |  |
| 1. Remove the long suggested name and enter **DAT368** in the Name to Show in Display field and then click the **OK** button. |  |
| 1. Left-click on Tables. Notice that a table named DAT368.db.data::sensors.temp appears in the list below. |  |
| 1. 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. |  |
| 1. Select the **HANA2 as teroot** line from the Saved Sessions. |  |
| 1. Then click **Load**. |  |
| 1. And finally **Open.** |  |
| 1. A putty window will open and you’ll be prompted to enter a password.   Password: **Welcome18** |  |
| 1. 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, User Ctrl-C in Windows and then right-click in the Putty console to paste the copy buffer into the command line. | |
| 1. Verify the the xs cli is set correctly.   **xs api** |  |
| 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** | |
| 1. Log in as the XSA\_ADMIN user.   **xs login -u XSA\_ADMIN -p WelcomeSAP2018** |  |
| 1. If the space is not DEV then select the DEV space with the xs target command.   **xs t -s DEV** |  |
| 1. Use the xs apps command to list all the applications(modules) in the space.   **xs apps** |  |
| 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). | |
| 1. Enter this command.   **xs a | grep xsa** |  |
| 1. 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** |  |
| 1. And finally the (r)outes that apps in this space have registered.   **xs r** |  |
| 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** | |
| 1. Use git to clone the repo.   **git clone https://github.com/alundesap/TechEd2018.DAT368.git** |  |
| 1. List the files and folders in the current directory.   **ls**  Make sure you see the **TechEd2018.DAT368** folder. |  |
| 1. 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 WebIDE 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. | |
| 1. Create the service instance.   **xs cs xsuaa default dat368-uaa -c ./xs-security.json** |  |
| 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** |  |
| 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**](https://wdflbmt0794.wdf.sap.corp:51027) | |
| 1. 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. |  |
| 1. Paste the URL into the browser making sure the port number is what was returned above. |  |
| 1. Log in but this time use the XSA\_ADMIN user.   User: **XSA\_ADMIN**  Password: **WelcomeSAP2018** |  |
| 1. Under **Security**, select **Role Collections** and **New Role Collection**. |  |
| 1. Give it a name and description and click **Save**.   Name: **DAT368\_MGR**  Description: **DAT368 Manager Role Collection** |  |
| 1. Click on the DAT368\_MGR link. |  |
| 1. Click on **Add Role**. |  |
| 1. You will get a dialog box with 3 dropdowns. Starting with the top one, select DAT368. |  |
| 1. Then for Role Template, select **DAT368Manager**. |  |
| 1. The for Role, select **DAT368Manager**. |  |
| 1. 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 and create a **New Role Collection** as before this time calling it **DAT368\_USR** with Description **DAT368 User Role Collection**. | |
| 1. Click **Save** when finished. |  |
| 1. Click on the DAT368\_USR link. |  |
| 1. 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  link near the top of the page and **User Management**.  In the search box type **DAT368**. This will limit the users being displayed. | |
| 1. In the search box type **DAT368**. |  |
| 1. On the line with User Name DAT368, select the Assign Role Collections icon. |  |
| 1. Click the **Add** button. |  |
| 1. Select the **DAT368\_MGR** Role Collection by checking the checkbox. |  |
| 1. And click **OK**. |  |
| 1. Verify that DAT368\_MGR is in the list and click the **Save** button. |  |
| 1. Close this incognito window. |  |
| This is a good time to verify that the user indeed has a proper role collection.  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](https://wdflbmt0794.wdf.sap.corp:30030)  and select the URL that follows it. | |
| 1. 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](https://wdflbmt0794.wdf.sap.corp:30030)/config?action=who | |
| 1. Replace the URL with the one above. |  |
| 1. This time log in with the following.   User: **DAT368**  Password: **WelcomeSAP2018** |  |
| 1. Verify that the **DAT368.view** and **DAT368.create** authorities are in the authorities array. This confirms that the user has the proper role collection. |  |
| 1. Close this incognito window. |  |

# Exercise 2

## Exercise 2.1: Building an MTAR

At this point we've been using the WebIDE to build the Database container and the XS CLI to create the authorization service instance.  If the WebIDE could support the Python module type then we would be able to use it to build the python module from within the WebIDE.  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, let’s remove the HDI container we created earlier.

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** |  |
| 1. 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/**](https://npm.sap.com/)**" ; npm config set registry "**[**https://registry.npmjs.org/**](https://registry.npmjs.org/)**" ; npm config set strict-ssl true** | |
| 1. Run the npm commands.   **npm config set @sap:registry "**[**https://npm.sap.com/**](https://npm.sap.com/)**"**  **npm config set registry "**[**https://registry.npmjs.org/**](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. | |
| 1. Create a target folder to store the mtar file that will get generated.   **mkdir -p target**  **ls -1** |  |
| 1. 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. | |
| 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. | |
| 1. Verify the deploy completed successfully by seeing output similar to the following. |  |
| 1. Check that’s it’s listed in the mta list.   **xs mtas** |  |
| 1. List it’s app modules |  |
| 1. And it’s services.   **xs s | grep dat368**  Notice that the HDI container has been re-created and that the uaa service still exists. |  |
| 1. Get the web module’s URL. Note the port number may differ for you.   **xs app DAT368.web --urls** |  |
| 1. Cut the URL and paste it into a new Chrome incognito browser window. |  |
| 1. Click on the **ODATA Links** link. This will prompt you for login since the nodejs module requires authentication. |  |
| 1. Use **DAT368** with password **WelcomeSAP2018** |  |
| 1. 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. |  |
| 1. Notice that there will initially only be one row as we noted above when we were exploring the data in the WebIDE. |  |
| 1. 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. |  |
| 1. In a new browser tab, you’ll see what the request looks like in the browser and the response(not pictured). |  |
| 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

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

But first, let’s remove the HDI container we created earlier.

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. | |
| 1. Switch to the solution branch.   **git checkout solution** |  |
| 1. 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. | |
| 1. 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. | |
| 1. 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. |  |
| 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> | |
| 1. Change up to the parent directory and list the files.   **cd ..**  **ls -1** |  |
| 1. Untar the tar-zipped source file.   **tar xzvf Python-3.6.5.tgz** |  |
| 1. 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.  *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. | |
| 1. 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** |  |
| 1. 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/** |  |
| Note 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**  Now running pip should not cause an error.  **pip --version**  pip 9.0.3 from /usr/sap/TE1/HDB00/python\_3\_6\_5/lib/python3.6/site-packages (python 3.6) | |
| 1. Check that it installed correctly.   **xs runtimes** |  |
| 1. Move back up to the home directory.   **cd ../..** |  |
| Exercise 3.2: Set up Python Libraries SAP provides a set of python libraries for validating request, 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](https://help.sap.com/viewer/4505d0bdaf4948449b7f7379d24d0f0d/2.0.03/en-US/842824f04d654ceeaf5168da663a65ce.html))  Note: We will be using two of the SAP supplied python libraries ([sap\_xssec](https://help.sap.com/viewer/4505d0bdaf4948449b7f7379d24d0f0d/2.0.03/en-US/8732609bd5314b51a17d6a3cc09110c3.html#loio8732609bd5314b51a17d6a3cc09110c3__section_atx_2vt_vt), [hdbcli](https://help.sap.com/viewer/0eec0d68141541d1b07893a39944924e/2.0.03/en-US/ee592e89dcce4480a99571a4ae7a702f.html)). 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](https://help.sap.com/viewer/4505d0bdaf4948449b7f7379d24d0f0d/2.0.03/en-US/8732609bd5314b51a17d6a3cc09110c3.html#loio8732609bd5314b51a17d6a3cc09110c3__section_vpc_qrj_ycb))  sap\_audit\_logging ([doc page](https://help.sap.com/viewer/4505d0bdaf4948449b7f7379d24d0f0d/2.0.03/en-US/8732609bd5314b51a17d6a3cc09110c3.html#loio8732609bd5314b51a17d6a3cc09110c3__section_cgq_w5l_mv))  sap\_cf\_logging ([doc page](https://help.sap.com/viewer/4505d0bdaf4948449b7f7379d24d0f0d/2.0.03/en-US/8732609bd5314b51a17d6a3cc09110c3.html#loio8732609bd5314b51a17d6a3cc09110c3__section_dhv_x21_cdb)) Cloud Foundry specific logging.  We would like to see an XSA service for application logging that follows the [ELK](https://www.elastic.co/elk-stack) stack compatible with the Cloud Foundry app-logging service, but this is not available as of the creation of this workshop. | |
| 1. Unzip   **unzip XS\_PYTHON00\_0-70003433.ZIP -d sap\_dependencies** |  |
| 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. | |
| 1. 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. | |
| 1. Run mkdir.   **mkdir -p vendor** |  |
| Use the python packaging tool(pip) to download the dependencies found in the requirements.txt | |
| 1. Run pip.   **pip download -d vendor -r requirements.txt --find-links ../../sap\_dependencies**  You can safely ignore the yellow upgrade warning. |  |
| 1. See what pip downloaded, change into the vendor folder.   **cd vendor**  **ls -1**  You should see a bunch of file with .whl .gz extensions. |  |
| 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. | |
| 1. Return to the project directory.   **cd ../..**  Run the following commands to build and deploy your project's mtar again. |  |
| 1. 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. Also the nodejs npm commands don’t take as much time since they’ve already pulled their dependencies. |  |
| 1. 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. |  |
| 1. At this point you should see your applications modules.   **xs a | grep DAT368**  This includes your python module. Also check that all but the db module are running. |  |
| 1. Get the app-router URL.   **xs app DAT368.web --urls**  Paste it an incognito browser window. |  |
| 1. Click on any of the python links. The first three don’t require authorization. |  |
| 1. Click on the last link **auth\_python/db\_valid**. |  |
| 1. This link requires authorization. Enter **DAT368** with password **WelcomeSAP2018** |  |
| 1. 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)  ... | |
| 1. 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) 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.  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.   Exercise 4.1: Build and Deploy to Cloud Foundry Return to the Putty console window.   |  |  | | --- | --- | | Explanation | **Screenshot** | | 1. 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** |  | | 1. See that there are now 2 mtar files in the target folder.   **ls -l target** |  | | 1. Connect to the US10 Cloud Foundry landscape.   **cf api https://api.cf.us10.hana.ondemand.com** |  | | 1. Log in as this user.   User: **primaryuser01@gmail.com**  Password: **PrimaryUs3r01**  **cf login -u primaryuser01@gmail.com -p PrimaryUs3r01** |  | | 1. Pick the space that corresponds to the number you were assigned at the beginning of this workshop. |  | | 1. Check to see that a service called dat368-hdi and dat368-uaa are 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.  In this case, we are deploying into a sub-account that has a sub-domain called dat368. The url endpoint of the approuter module must include the sub-domain name. A special environment variable called TENANT\_HOST\_PATTERN which contains a regular expression. The first matching group needs to extract the same string as the sub-domain.  ^(.\*)-(.\*)-(.\*).cfapps.(.\*).hana.ondemand.com  dat368-00-web.cfapps.eu10.hana.ondemand.com | | | 1. 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** |  | | 1. Deploy the mtar.   **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 will need to log into your cloud foundry account and create role collections and assign roles to them.  You then 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’ve done this for you already.  If you were doing this in your own account, you would first create a new DAT368 role collection and add the DAT368Manager and DAT368 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. | | | 1. Verify that the url of the approuter(web) module starts with dat368.   cf a |  | | 1. Also verify that the approuter(web) module has the TENANT\_HOST\_PATTERN regex set properly.   **cf env DAT368.web** |  | | 1. Get the url of the approuter(web) module.   **cf a | grep DAT368.web** |  | | 1. Paste it into an incognito browser window click on the and login with   User: **primaryuser01@gmail.com**  Password: **PrimaryUs3r01** |  | | |
|  | |
|  | |
| 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 is to log the module’s activity to either the audit log and/or application log service.  This concludes the python wrangling for today. Please fill out the survey in the TechEd mobile app for this workshop session. | |

**www.sap.com/contactsap**



© 2018 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE’s or its affiliated companies’ strategy and possible future developments, products, and/or platforms, directions, and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, and they should not be relied upon in making purchasing decisions.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

See <https://www.sap.com/copyright> for additional trademark information and notices.

**www.sap.com/contactsap**