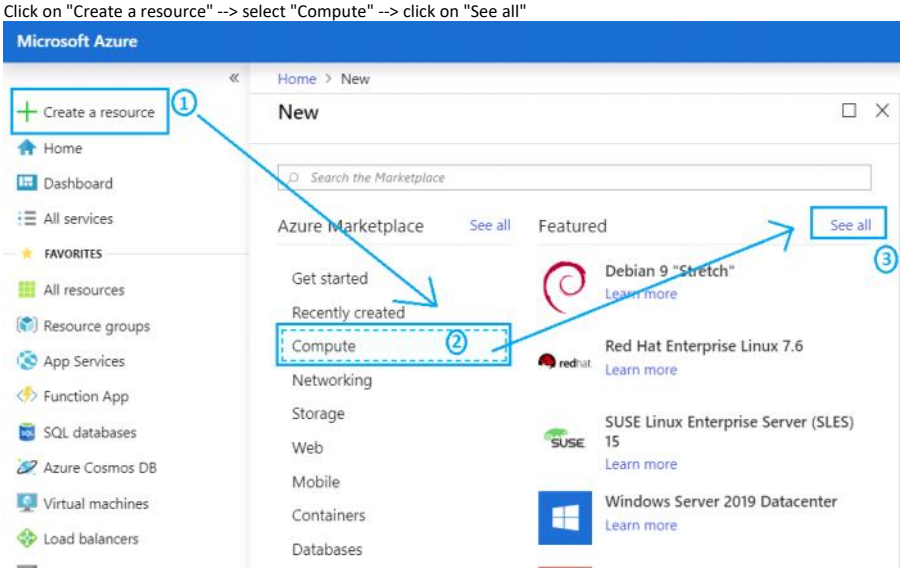
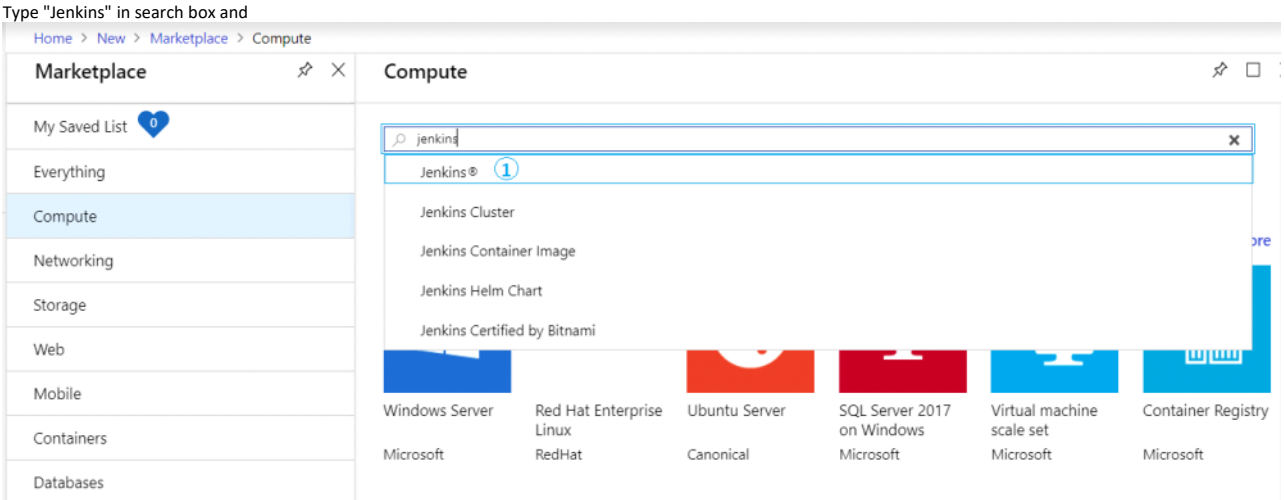
Provisioning an Azure Jenkins VM

Tuesday, April 16, 2019 10:58 AM

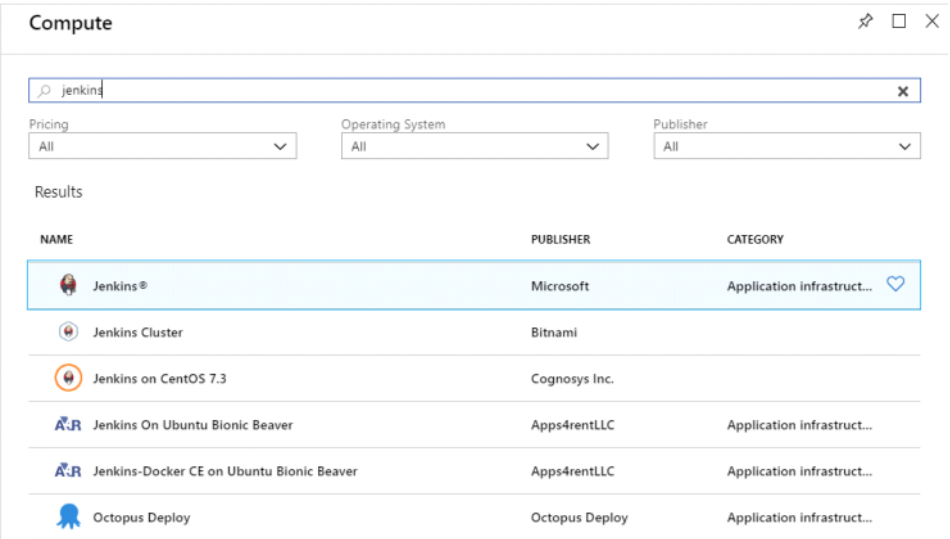
1. Using your account to access to Azure portal https://portal.azure.com/. If you do not have one, you can sign up for a free trial
2. Click on "Create a resource" --> select "Compute" --> click on "See all"



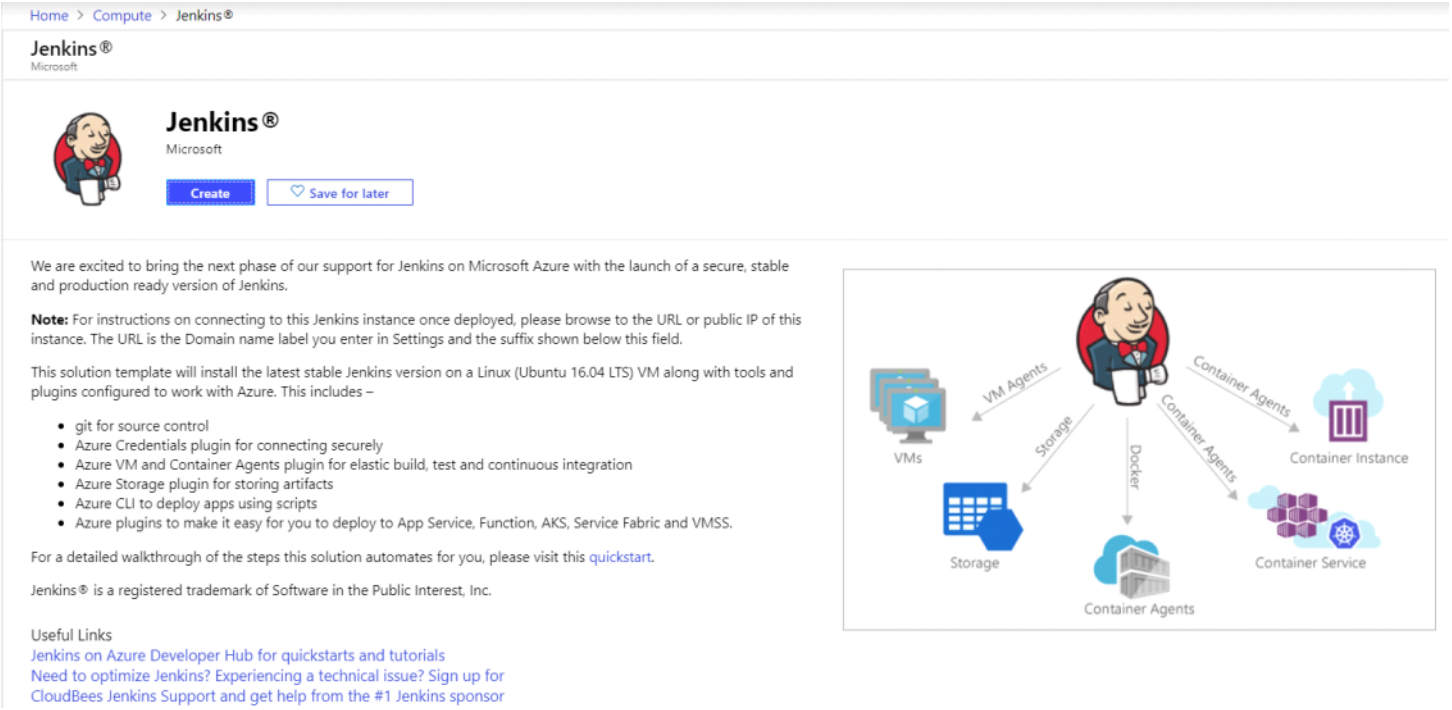
3. Type "Jenkins" in search box and



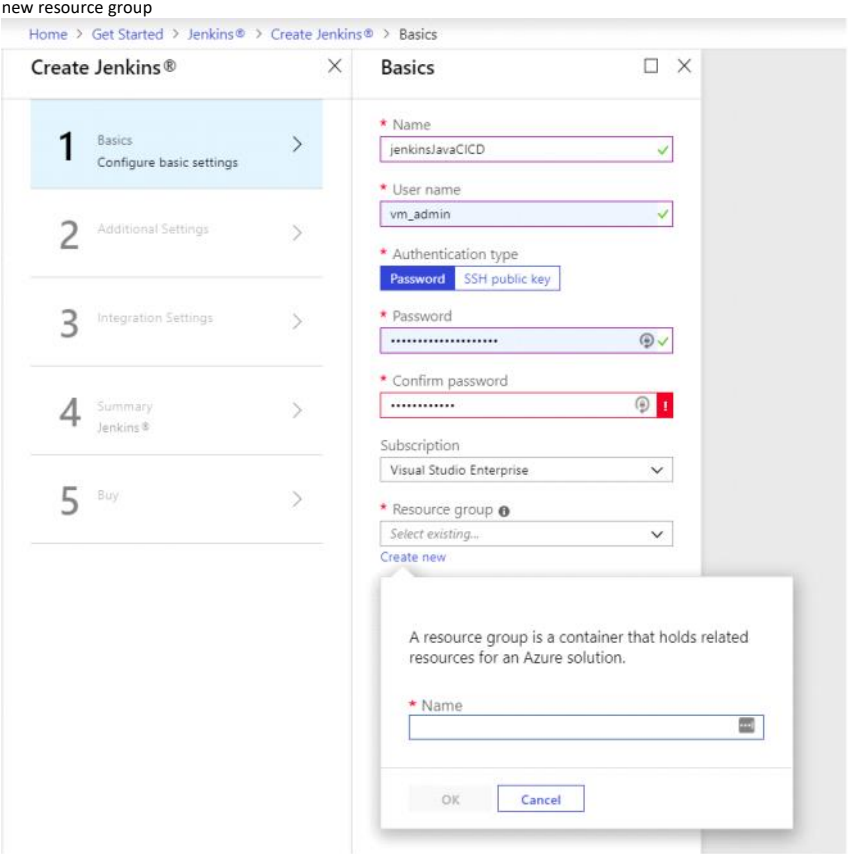
Select Jenkins® item in search result



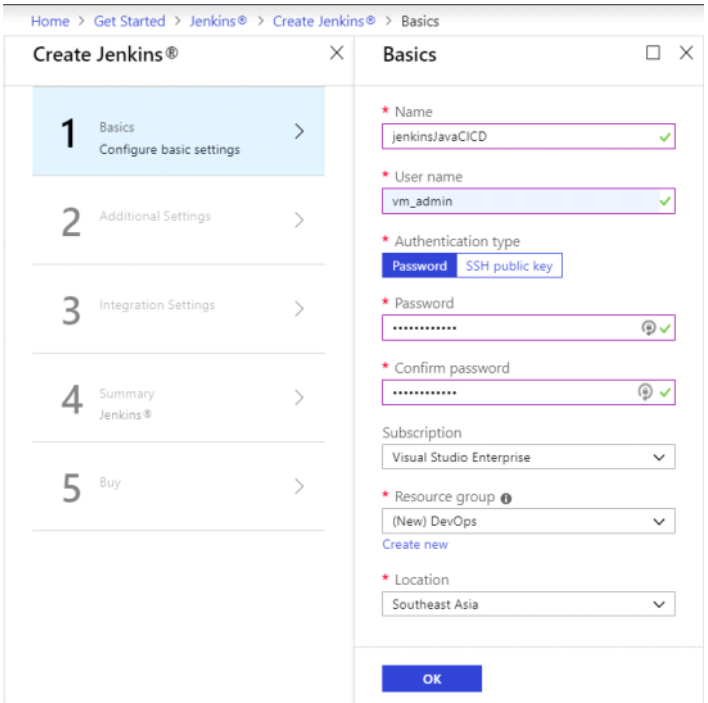
5. An Azure Marketplace > Compute > Jenkins® will be displayed, click on "Create" button to create



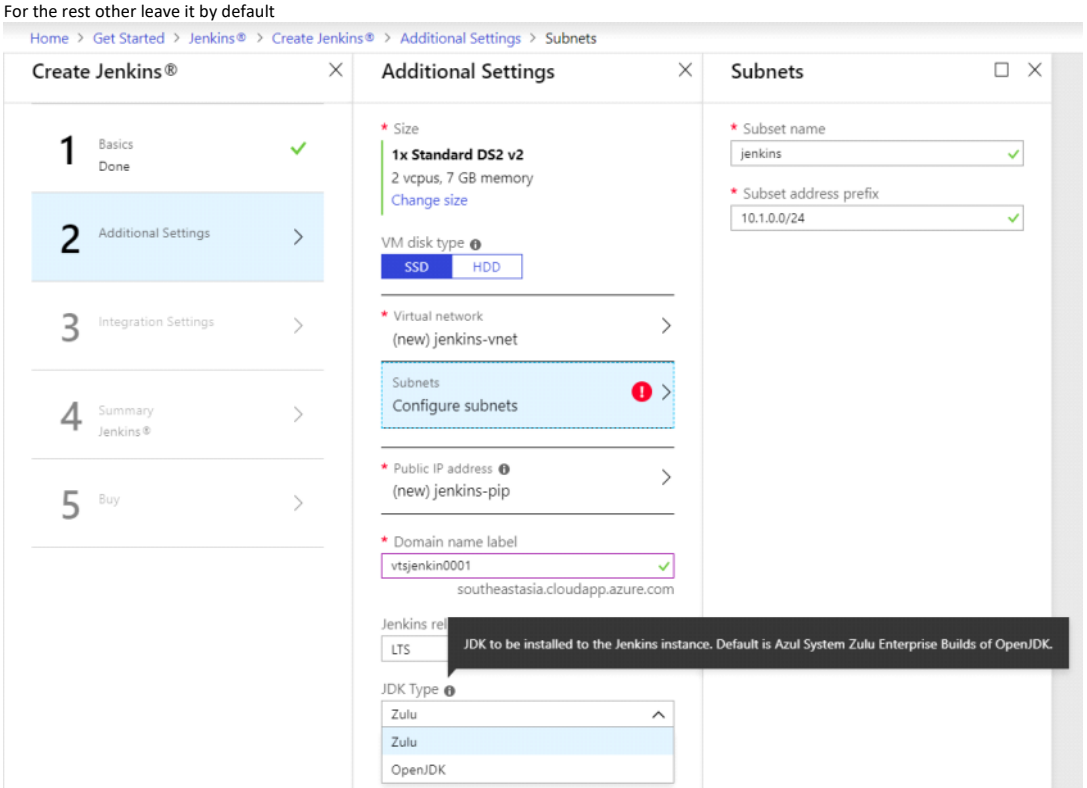
6. The Configure basic settings screen will be display and input your VM Name, Username/password, create new resource group



7. Click on button "OK"

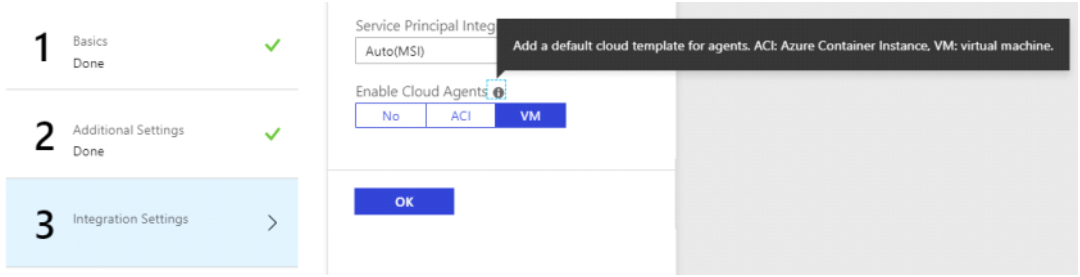


8. Additional Settings screen will be displayed
   Select your appropriate Size or click on "Change size" to change into other VM size that you want to
   Type the unique "Domain Name label" that does not exist with other
   For the rest other leave it by default

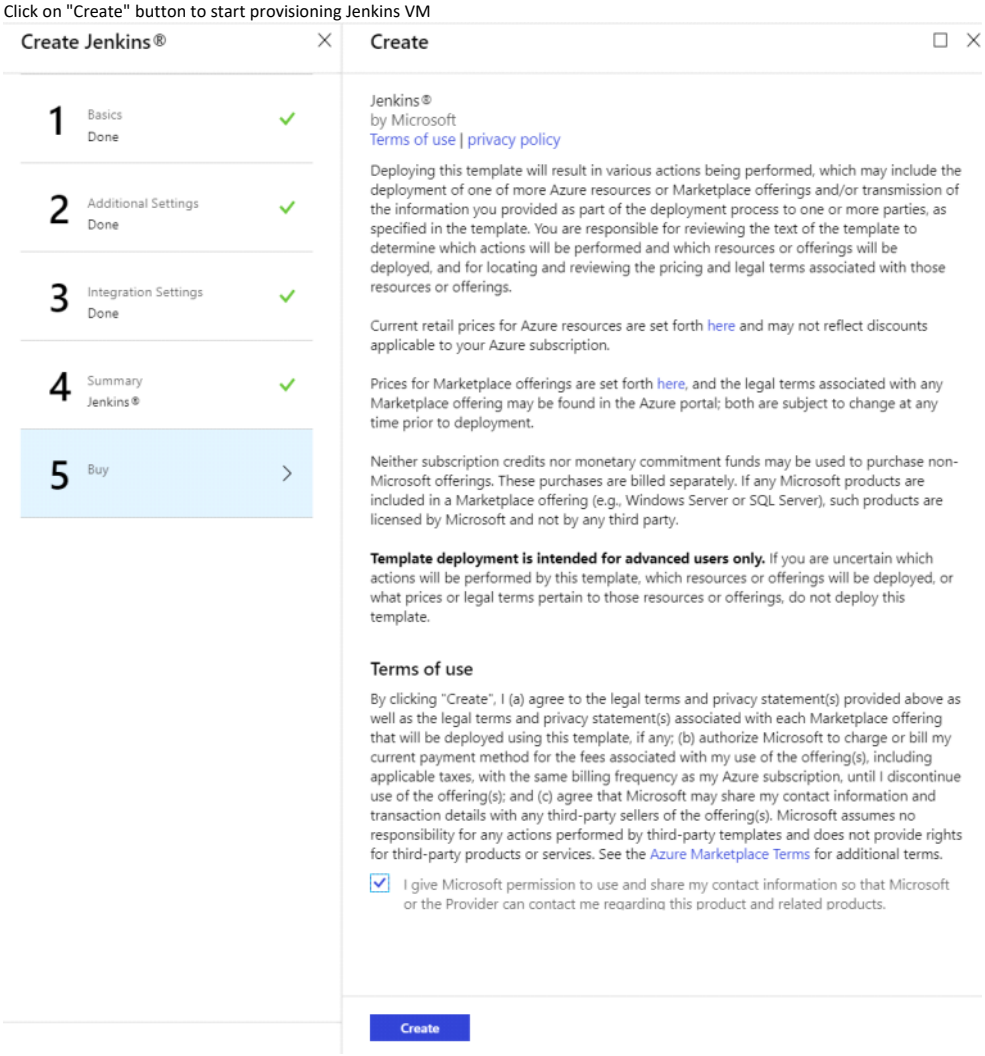


   Click "OK"

9. The Integration Settings screen will be displayed. Leave all items as default and click on button "OK" to continues



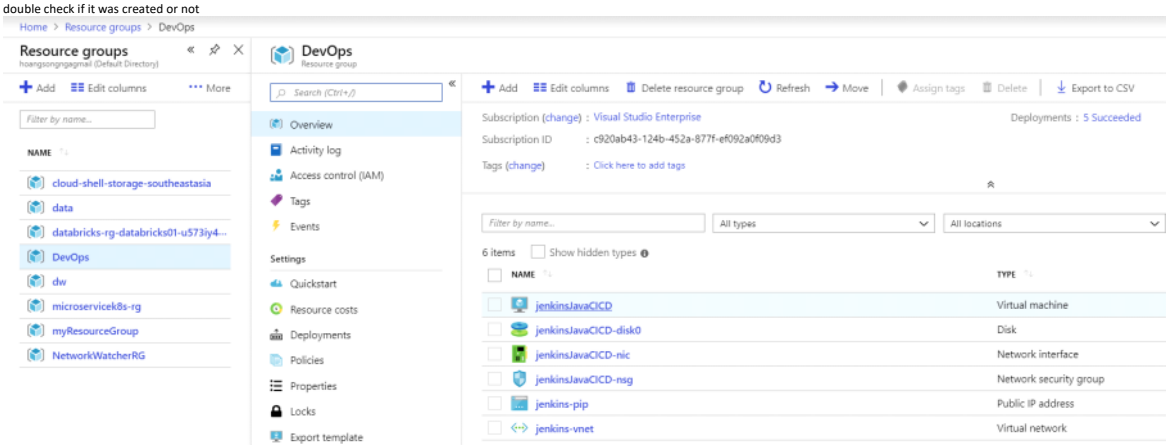
10. The Summary will be displayed to validate all previous step, if everything okay the "OK" button will be enabled and you should click on "OK" to continue
11. The last screen will be display to require you accept the condition and term …
12. Click on "Create" button to start provisioning Jenkins VM



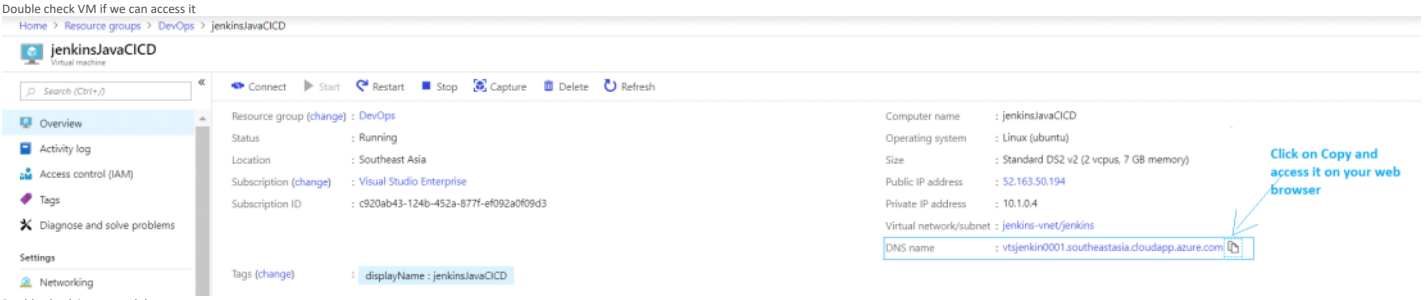
Setting up the Jenkins VM

Tuesday, April 16, 2019 11:47 AM

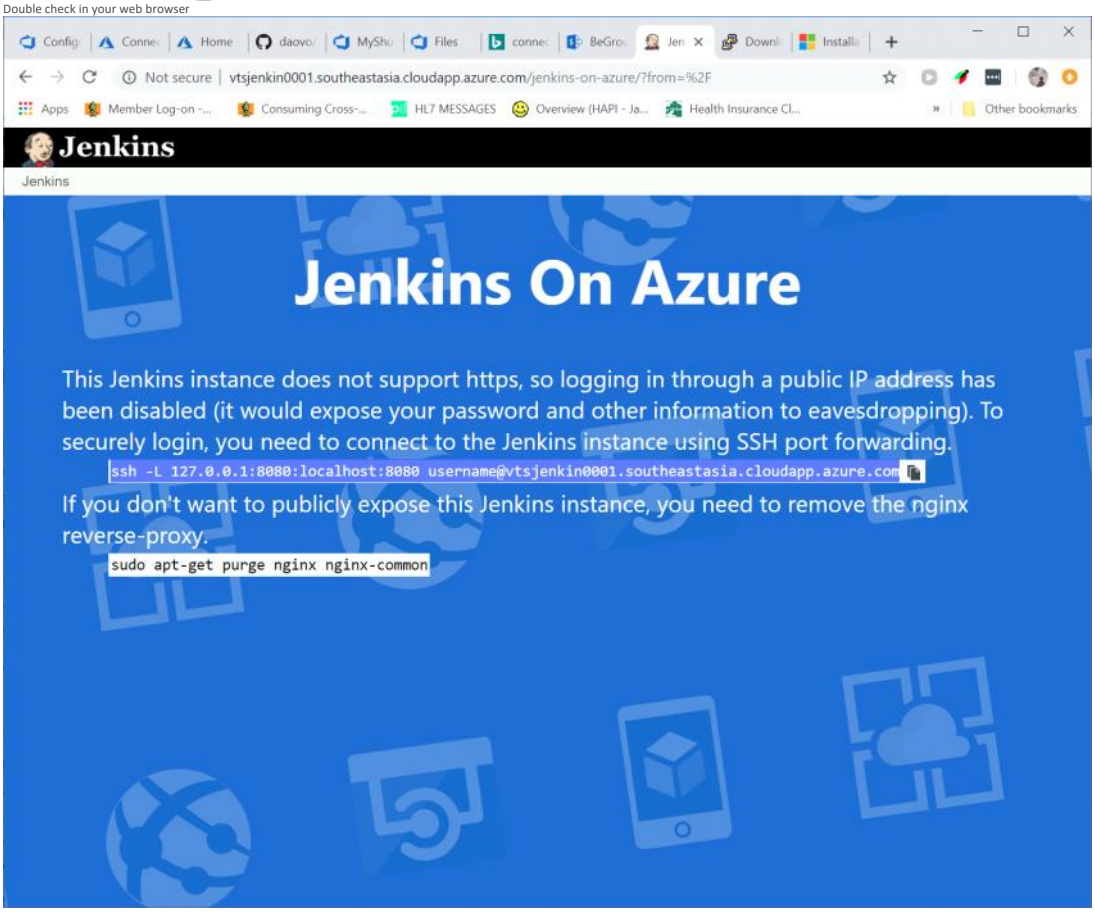
1. Once the Jenkins VM is provisioned, please go to resource group to find out VM name that you named to double check if it was created or not



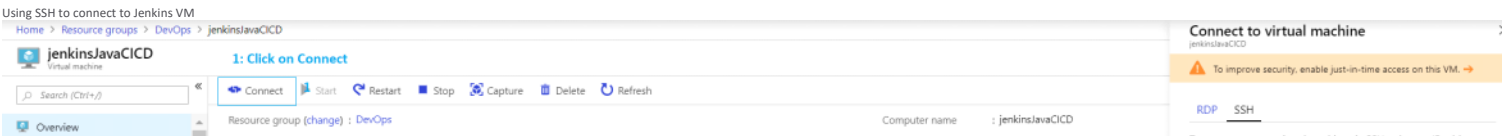
2. Double check VM if we can access it

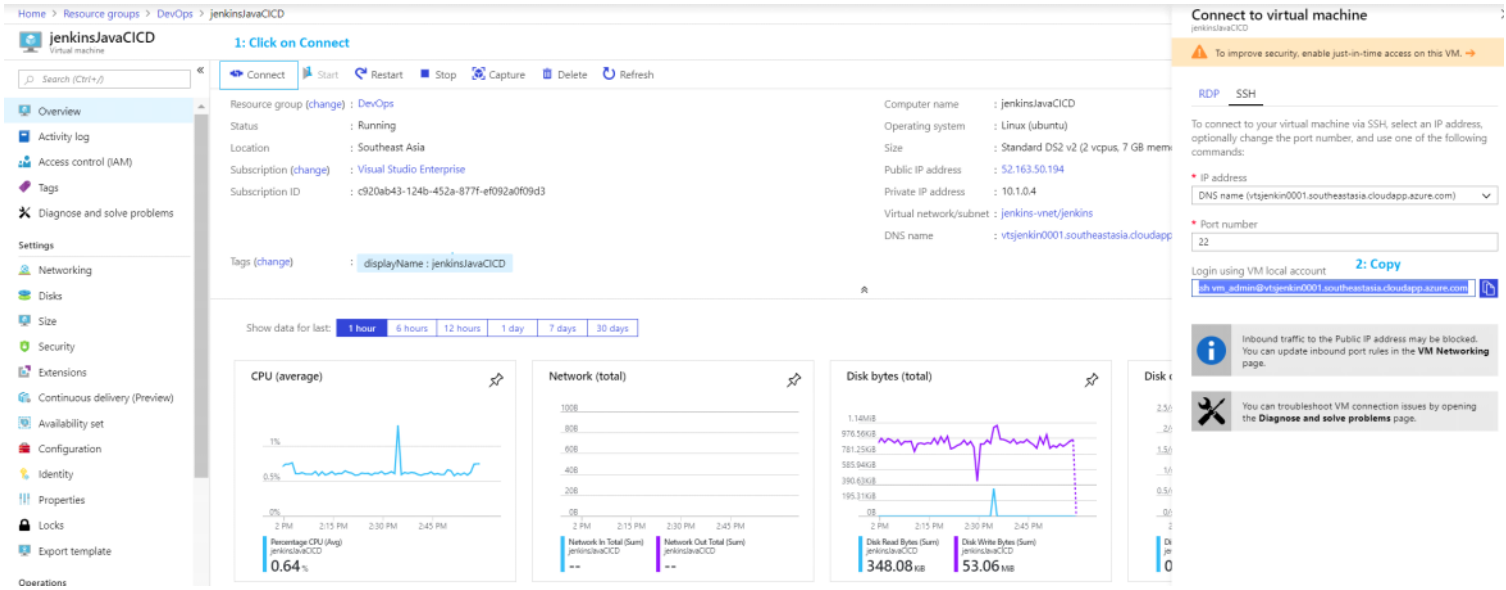
3. 

4. Double check in your web browser

5. 

5. Using SSH to connect to Jenkins VM

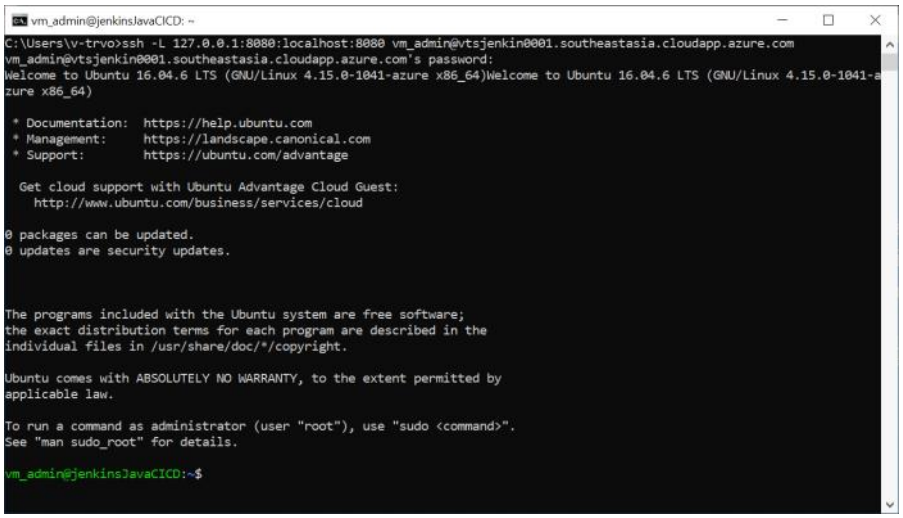




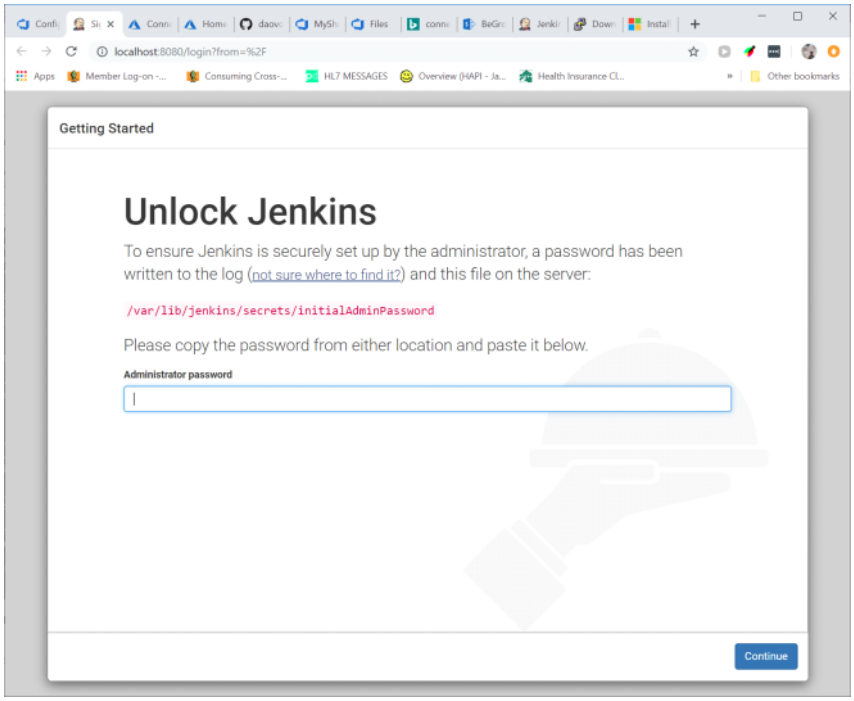
6. To initiate an SSH tunnel, the following command needs to be run from a Command Prompt. An SSH tunnel creates a secure connection between your host and remote computer through which services can be relayed. If this command is successful, you should be able to access the remote Jenkins on port 8080 on your local machine.

   putty.exe -ssh -L 8080:localhost:8080 <username>@<ip address>

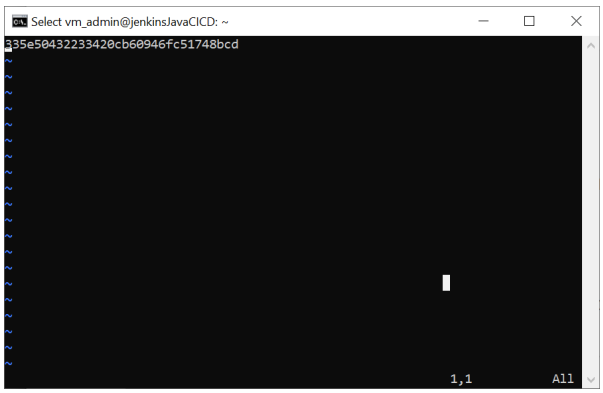
   Open your CMD from Run on window dialog to access to the VM by using you username/password from previous steps



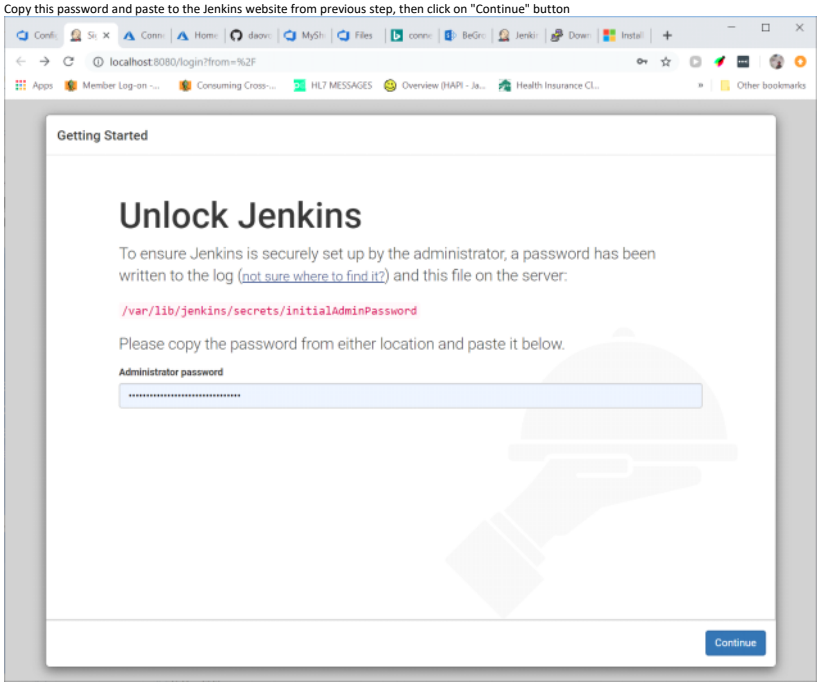
1. Once the connection is successful, open a browser on the host machine and navigate to the URL http://localhost:8080. The Getting Started page for Jenkins will be displayed.

2. 

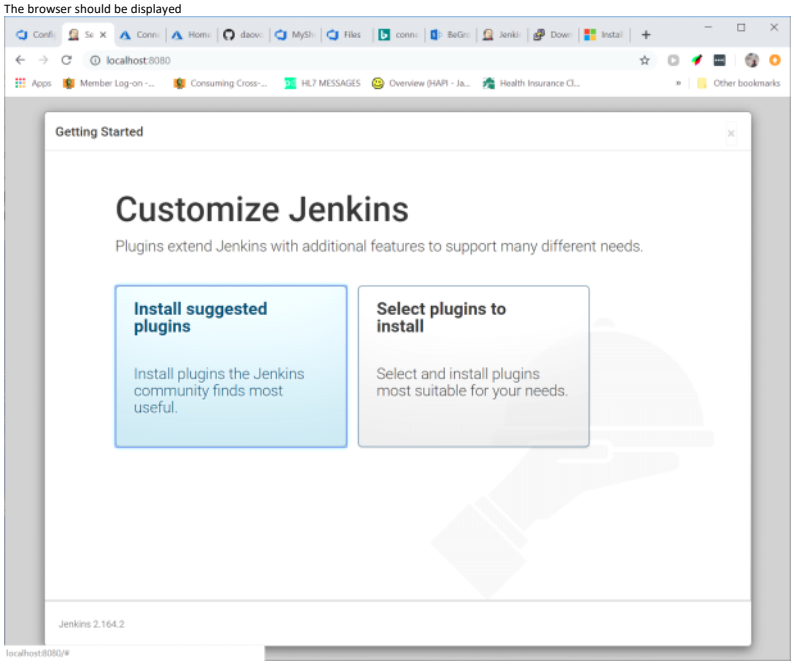
3. For security reasons, Jenkins generates an initial password and save it in a file on the server. This password will need to be retrieved and provided to unlock Jenkins. Return to the **Putty** terminal and type the following command to open the password file and copy the password. You can double click on the password text and use **CTRL+C** to copy the text and place it in the clipboard. Press the **Esc** button and then type **:q!** at the prompt to exit the vi editor without saving the file.
sudo vi /var/lib/jenkins/secrets/initialAdminPassword



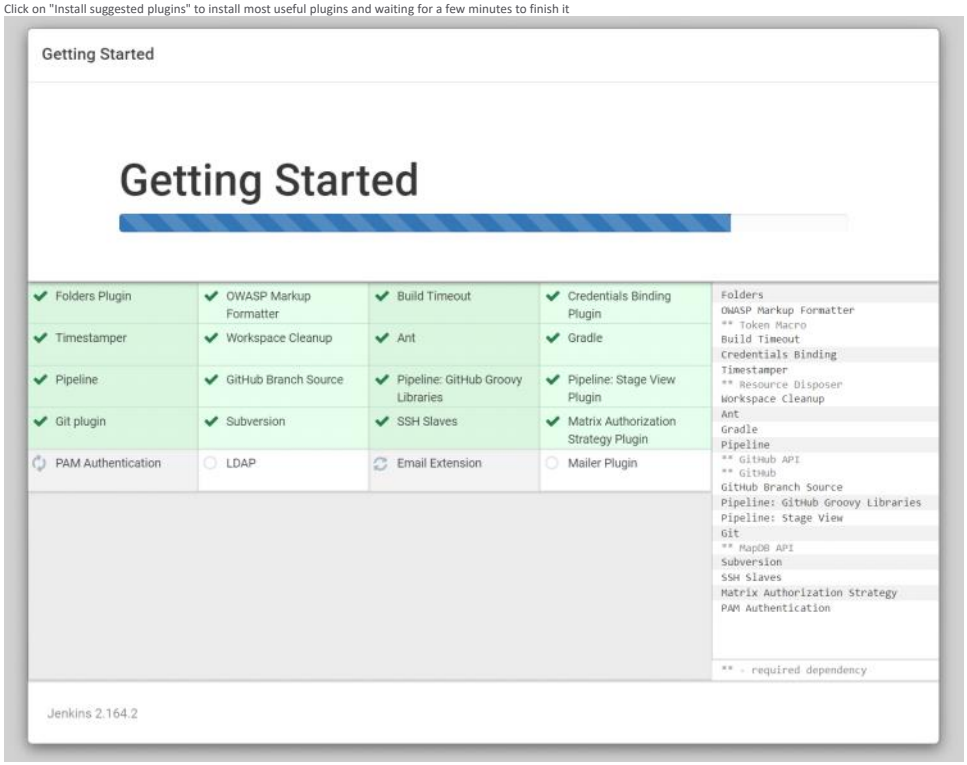
4. Copy this password and paste to the Jenkins website from previous step, then click on "Continue" button

5. 

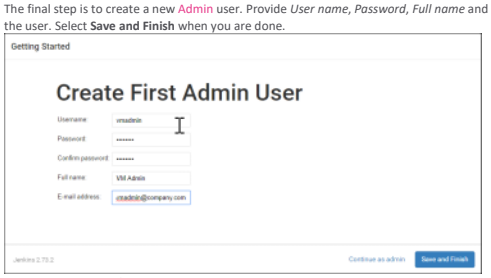
6. The browser should be displayed

7. 

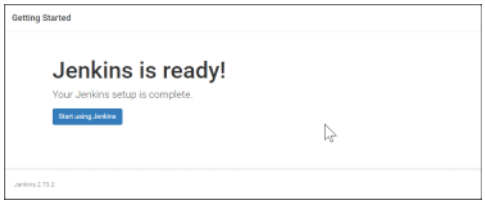
8. Click on "Install suggested plugins" to install most useful plugins and waiting for a few minutes to finish it

9. 

10. The final step is to create a new Admin user. Provide *User name*, *Password*, *Full name* and *Email address* for the user. Select **Save and Finish** when you are done.

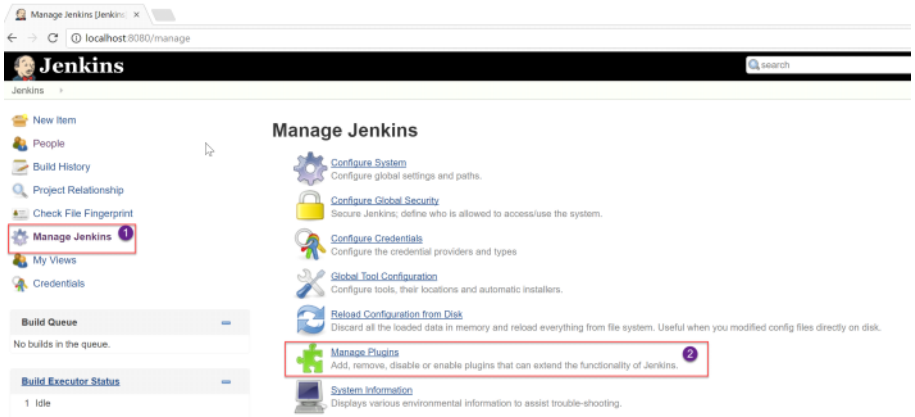


11. Jenkins is now ready for use. Select **Start using Jenkins** to start using it.

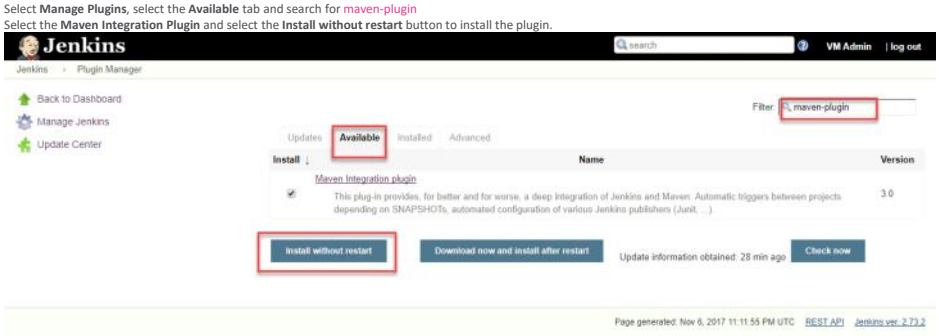


Installing and Configuring Plugins

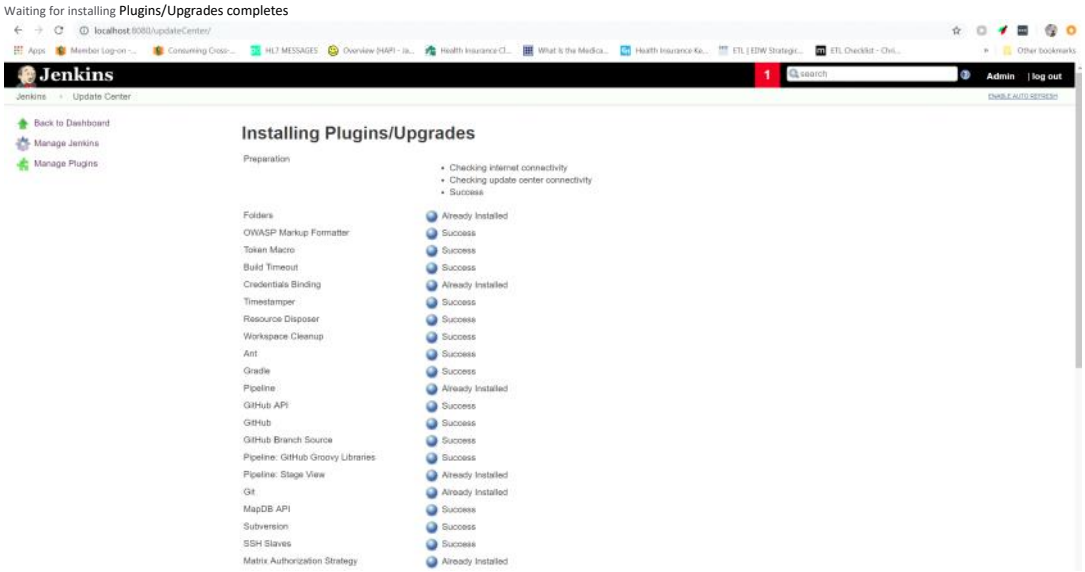
1. We will now install the Maven and VSTS (yet to be renamed Azure DevOps!) plugins that we require for this lab. Click **Manage Jenkins** on the Jenkins home page and select **Manage Plugins**. Select the **Available** tab and search for team services



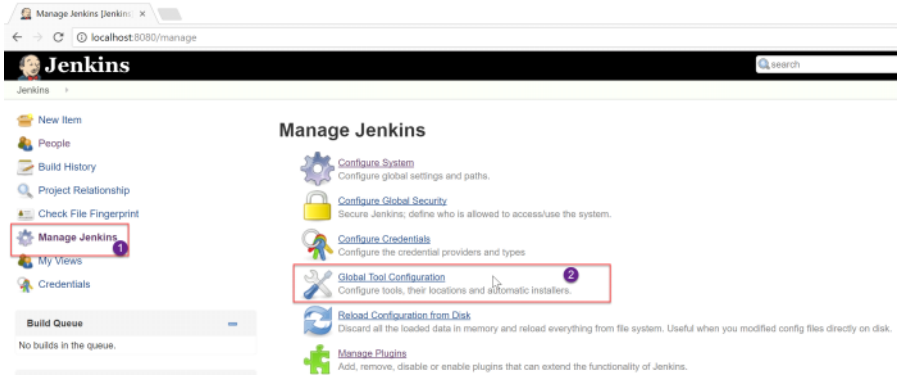
2. Select **VS Team Services Continuous Deployment/ GitHub Authentication/ GitLab Authentication** plugin and select **Install without restart**
3. Select **Manage Plugins**, select the **Available** tab and search for maven-plugin
4. Select the **Maven Integration Plugin** and select the **Install without restart** button to install the plugin.



5. Waiting for installing Plugins/Upgrades completes

6. 

7. Once the plugin is installed, go back to **Manage Jenkins** and select the **Global Tool Configuration** option.



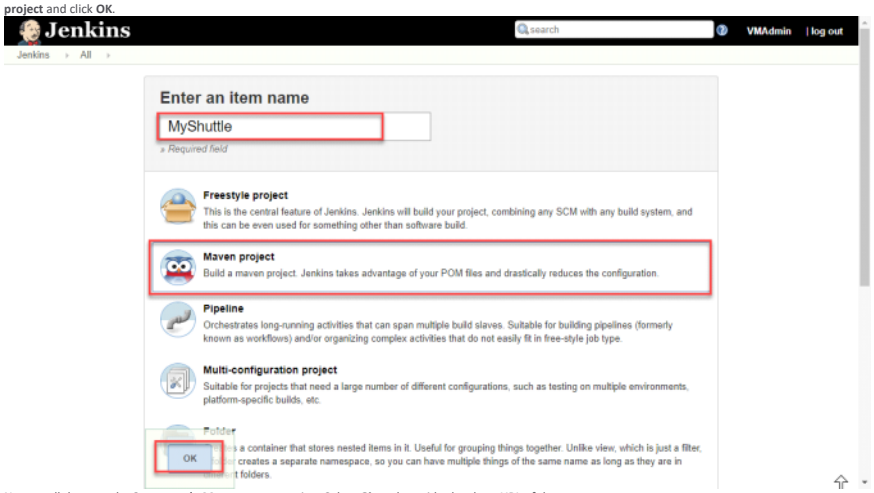
Note: Jenkins provides great out-of-the-box support for Maven. Since Maven is not yet installed, it can be manually installed by extracting the tar file located in a shared folder. Alternatively, when the **Install automatically** option is selected in the **Global Tool Configuration** screen, Jenkins will download and install Maven from the Apache website when a build job requires it.

8. To install Maven, select the **Install automatically** option and select the **Apply** button. The latest version of Maven at the time of writing this lab is 3.5.4 (latest version is **3.6.1)**

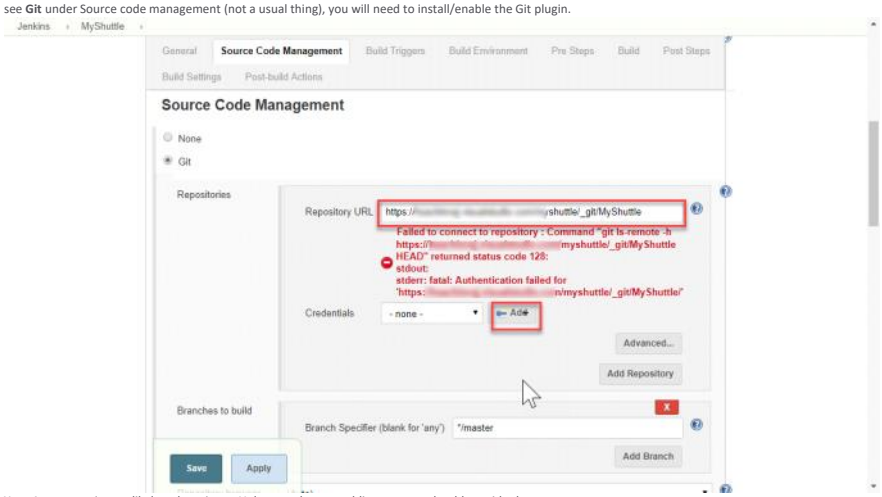
9. Select the **Back to Dashboard** button to return to the home page. We are done with the setup. Let's go and create a new CI job.

Creating a new build job in Jenkins

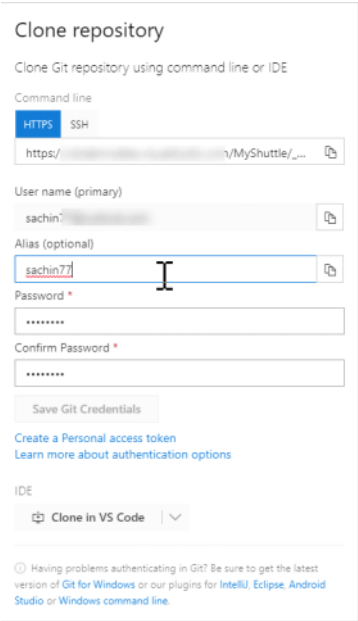
1. From the Jenkins home page, click the **New Item** link. Provide a name for the build definition, select **Maven project** and click **OK**.



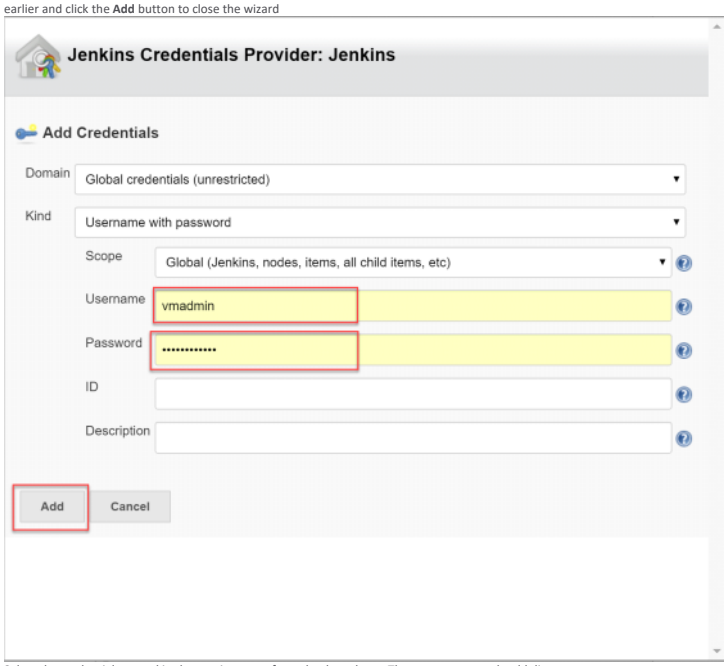
2. Now scroll down to the **Source code Management** section. Select **Git** and provide the clone URL of the Azure DevOps Git repo in the format http://dev.azure.com/{your org name}/{team project name}/_git/MyShuttle (or github or gitlab url but my suggestion we should use dev.azure.com repos). If you do not see **Git** under Source code management (not a usual thing), you will need to install/enable the Git plugin.



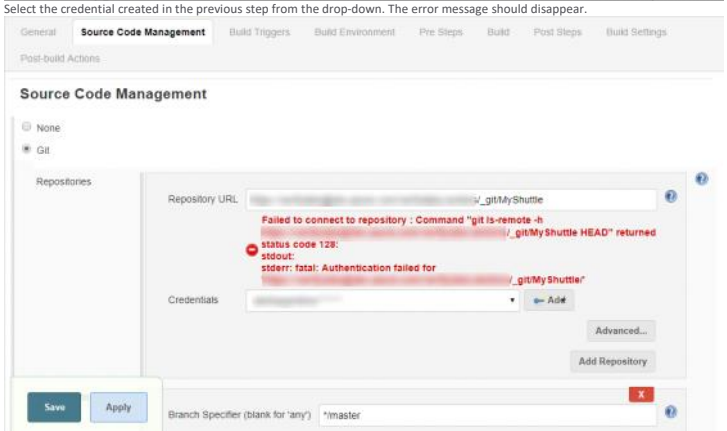
3. Your Azure repo is very likely to be private. Unless you have a public repo, you should provide the credentials to access the repository. If you do not have one or don't remember the credentials, go to your Azure Repos and select the **Clone** option. Select **Generate Credentials** and enter a User name and Password. Click **Save Git Credentials** to save.



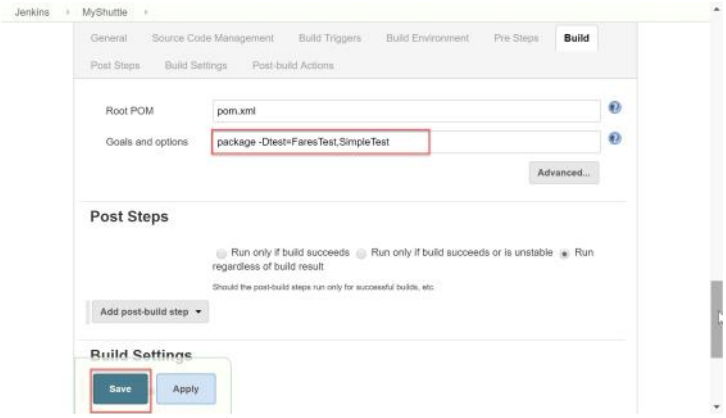
4. Select the **Add | Jenkins** option to add a new credential. Provide the User name and Password created earlier and click the **Add** button to close the wizard



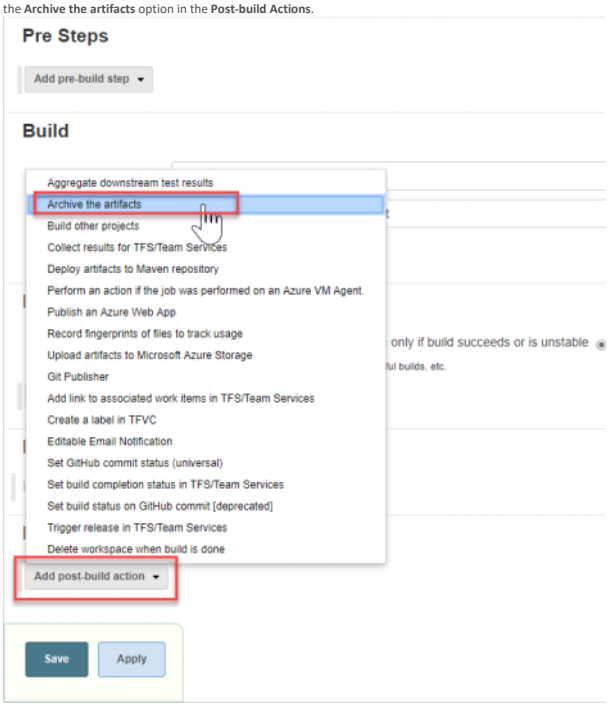
5. Select the credential created in the previous step from the drop-down. The error message should disappear.



6. The source code for this application has both unit tests and UI tests. We are not ready to run the UI test at this point. So, we will specify to run only the unit tests. Scroll down to the **Build** section and provide the text package -Dtest=FaresTest,SimpleTest in the **Goals and options** field.

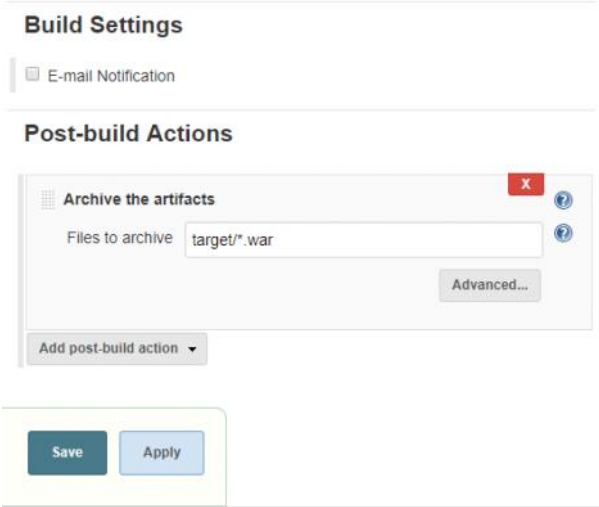


7. Once the build is complete, you can specify what action you want to take. For instance, you can archive the build artifacts, trigger an Azure CD pipeline, deploy directly to Azure App Service, etc., We will choose the **Archive the artifacts** option in the **Post-build Actions**.

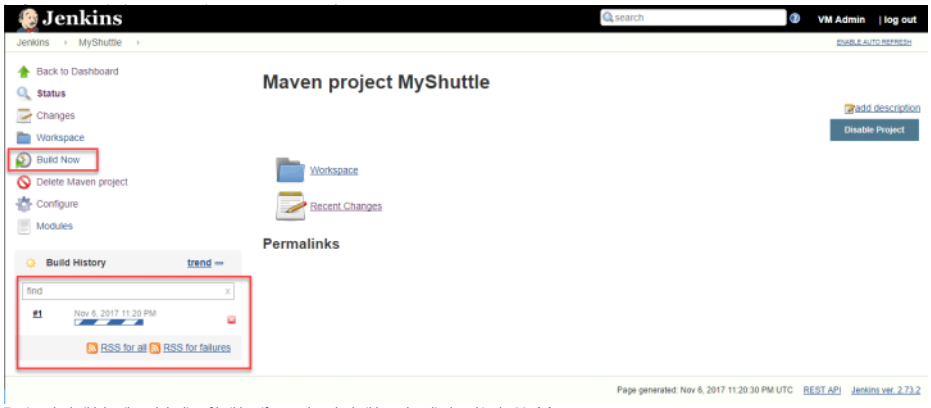


Note: Note there is also **Post-build steps** section which is very similar to the actions section. The tasks configured in the post-build steps/actions are executed after all the build steps have been executed.

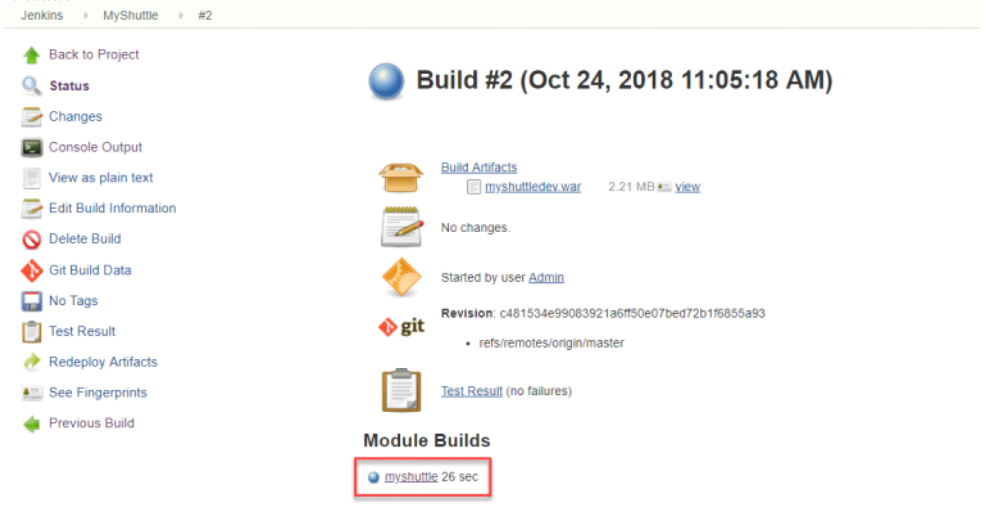
8. Enter **target/*.war** in the **Files to archive** text box. Select the **Save** button to save the settings and return to the project page.



9. The configuration is now completed, Select the **Build Now** option to initiate an Ad-hoc build. The build progress will be displayed on the left pane in the **Build History** section



10. To view the build details and the list of build artifacts, select the build number displayed in the **Module Builds** section.



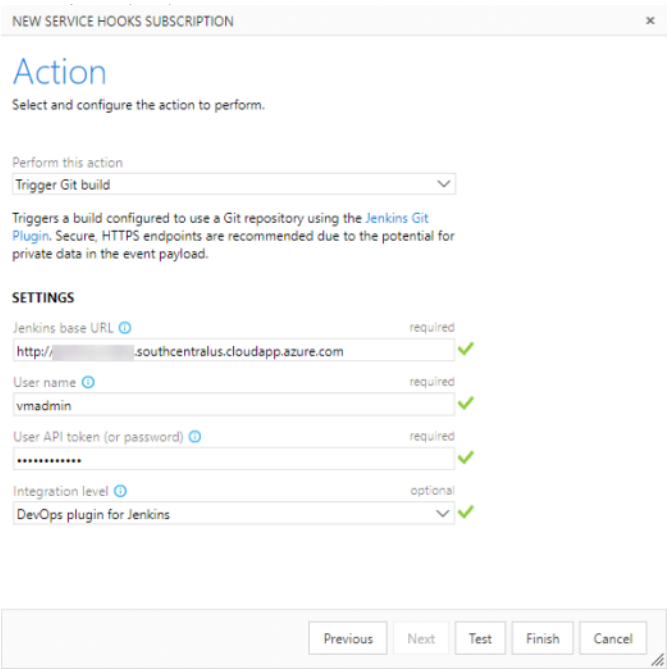
11. Select the **Test Result** option to view the results of the unit tests that were included in the build definition.

Next, we will explore the two different options available to trigger the Jenkins CI job when a code is pushed to Azure Repos.

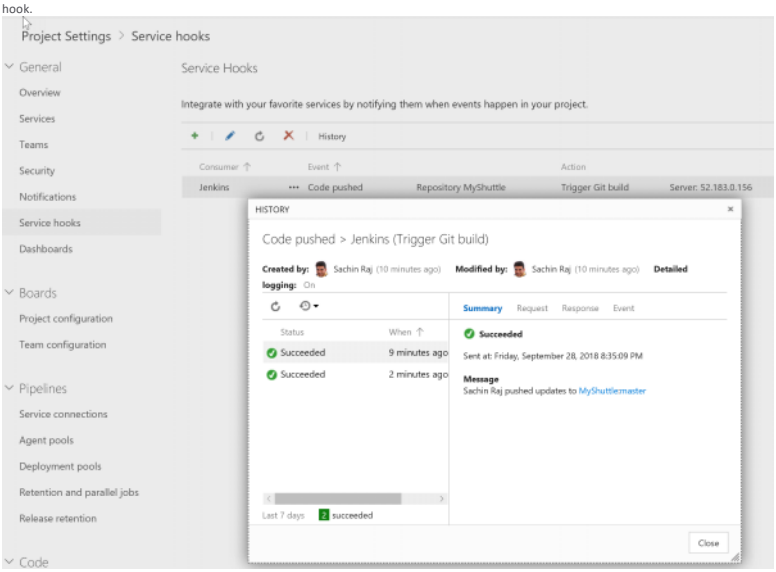
Approach 1: Triggering the CI via a service hook in Azure DevOps

In this approach, a service hook will be configured in Azure DevOps to trigger a Jenkins build upon a code commit. Service hooks enable you to perform tasks on other services when events happen in your Azure DevOps Services projects.

1. To configure the service hook, navigate to the Azure DevOps project settings page and select **Service hooks** under **General**. Select **+ Create subscription**.
2. In the *New Service Hook Subscriptions* screen, select the **Jenkins** option and then click the **Next** button. Jenkins service supports three events - **Build completed**, **Code Pushed** and **Pull request merged**. We are only interested in the code push event - so, select **Code pushed** for the **Trigger on this type of event** field. Select the **MyShuttle** repository and then click **Next**



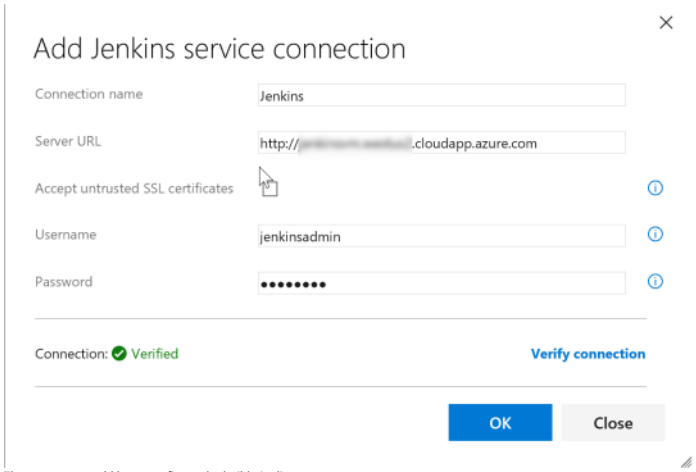
3. Provide the following details in the **Select and configure the action to perform** screen
    1. Select the **Trigger Git build** option
    2. Provide the **Jenkins base URL** in http://{ip address or the host name} format
    3. Provide the **User name** and **Password** to trigger the build. Note the username and password is the credentials of the Jenkins administrator user that you configured earlier
4. Click the **Test** button to validate the configuration and then click **Finish**. This will set the trigger to initiate the Jenkins CI build whenever a source code change is committed on the repository.
5. Try making a commit to the code - src/main/webapp/index.jsp would be a good candidate. This should trigger the MyShuttle build on Jenkins. You can confirm it by checking the history tab of the Jenkins services hook.



Approach 2: Wrapping Jenkins Job within Azure Pipelines

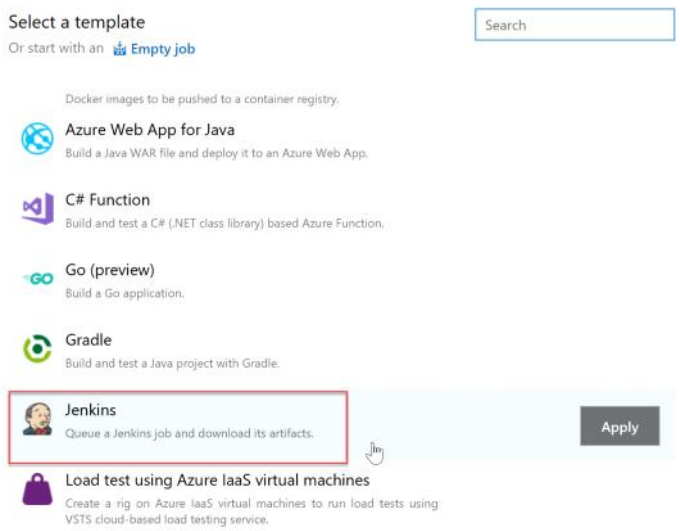
In this approach, Jenkins CI job will be nested within an Azure CI pipeline. The key benefit of this approach is you can have end-to-end traceability from work items to source code to build and release pipelines.
To begin, an endpoint to the Jenkins Server for communication with Azure DevOps will be configured.

1. Go to your project settings. Select **Pipelines** and **Service connections**, click **New service connection** and choose **Jenkins** from the dropdown.
2. Provide a connection name, Jenkins server URL in the format http://[server IP address or DNS name] and Jenkins user name with password. Select **Verify Connection** and validate the configuration. If it successful, then select **Ok**.

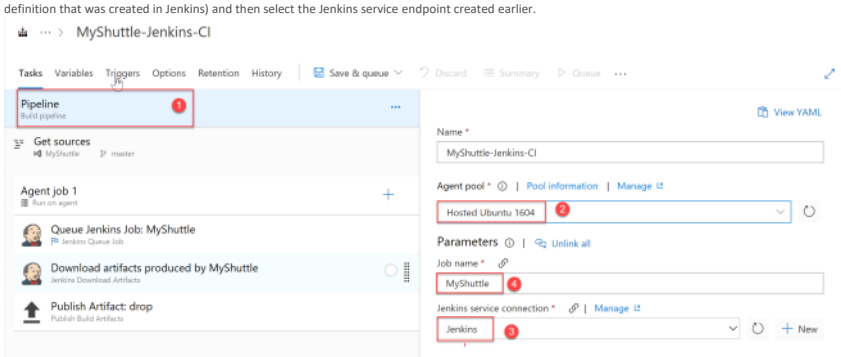


The next step would be to configure the build pipeline.

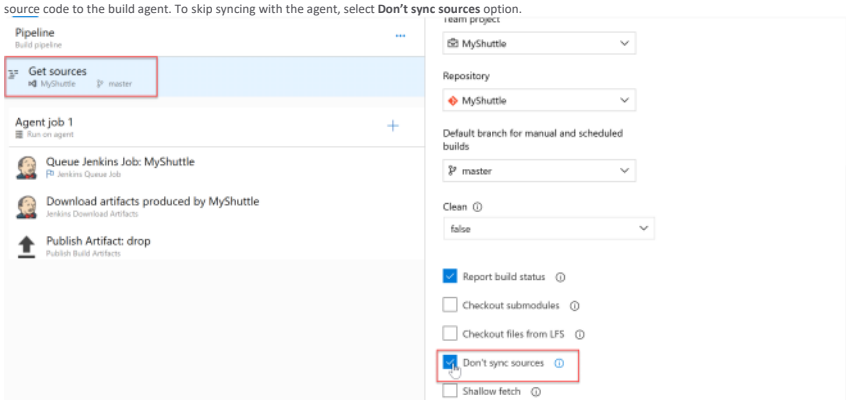
3. Go to **Azure Pipelines** and **Builds**, Click **+New** and select **New build pipeline** to create a new build definition.
4. At the time of writing this lab, Azure Pipelines did not support Jenkins in YAML. Select **Use the Visual Designer** to create a pipeline without a YAML.
5. Select **Myshuttle** project, repository and click *Continue*.
6. Scroll down and select the standard **Jenkins** template Click Apply.



7. Select **Hosted VS2017** for the Agent Queue, provide **MyShuttle** as the Job name (name of the build definition that was created in Jenkins) and then select the Jenkins service endpoint created earlier.

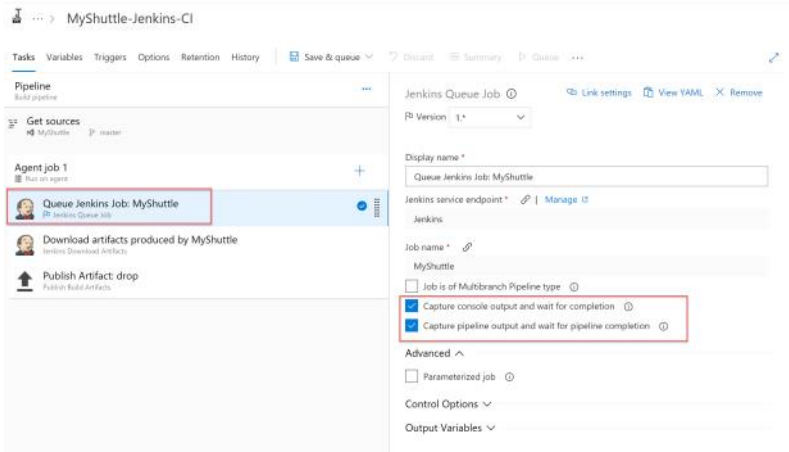


8. Next, select the **Get Sources** step. Since Jenkins is being used for the build, there is no need to download the source code to the build agent. To skip syncing with the agent, select **Don't sync sources** option.

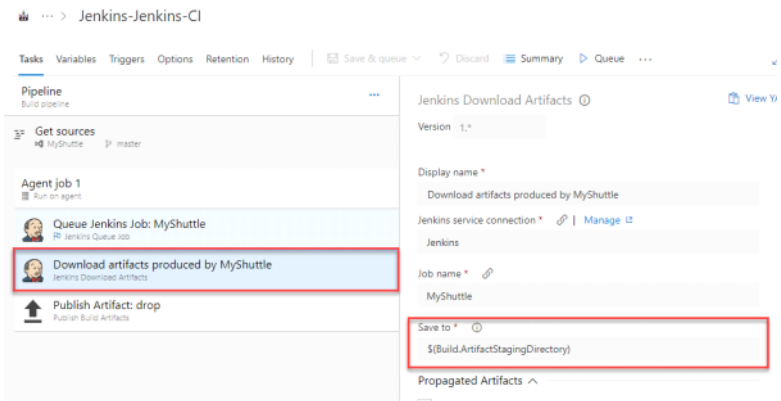


9. Next, select the **Queue Jenkins Job** step. This task queues the job on the Jenkins server. Make sure that the

services endpoint and the job name are correct. The **Capture console output** and the **Capture pipeline output** options available at this step will be selected.
The **Capture console output and wait for completion** option, when selected, will capture the output of the Jenkins build console when the Azure build pipeline runs. The build will wait until the Jenkins Job is completed. The **Capture pipeline output and wait for pipeline completion** option is very similar but applies to Jenkins pipelines (a build that has more than one job nested together).



10. The **Jenkins Download Artifacts** task will download the build artifacts from the Jenkins job to the staging directory.

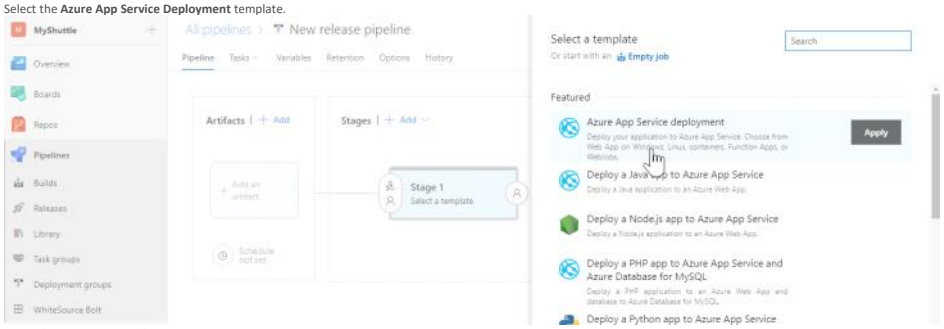


11. The **Publish Artifact drop** will publish to Azure Pipelines.
12. Click **Save & queue** button to save and initiate a new build.

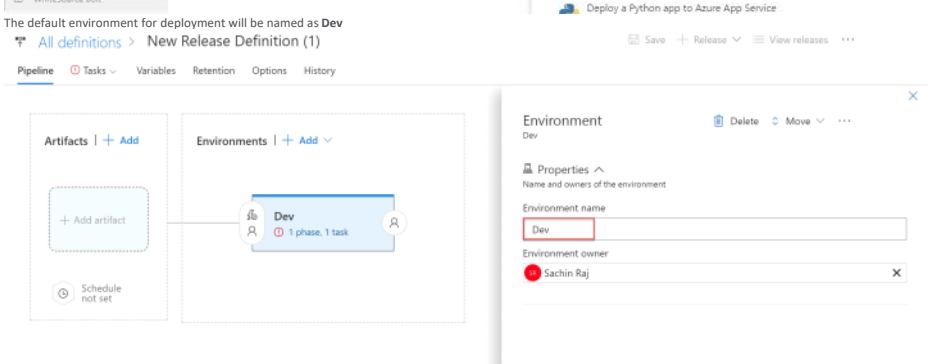
Linking the build artifact for deployment in a CD pipeline

Next, you will configure an Azure CD pipeline to fetch and deploy the artifacts produced by the build. Since the deployment is being done to Azure, an endpoint to Azure will be configured. An endpoint to Jenkins server will also be configured, if not configured earlier.

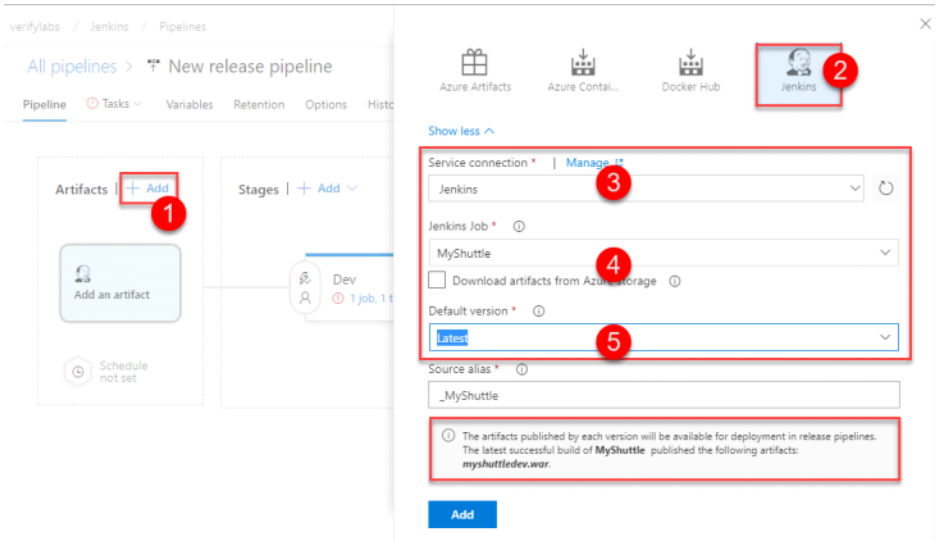
1. After the endpoint creation, go to the **Releases** tab in **Azure Pipelines**. Open the + drop-down in the list of release pipelines, and choose **Create release pipeline**.
2. Select the **Azure App Service Deployment** template.



3. The default environment for deployment will be named as **Dev**



4. In the **Artifacts** section in the **Pipeline** tab, choose the **+ Add** link to select your build artifact.
   1. If you have used the first approach, select **Jenkins** as the *Source type*, select the Jenkins endpoint configured earlier and provide **MyShuttle** for the *Source(Job)*, choose the **Default version** as *Latest*.
      The Source(Job) should map to the project name configured in Jenkins.
      If the Jenkins server and the source location is configured correctly, once the publishing of the artifacts is completed, a message with the output file name **myshuttledev.war** will be displayed.



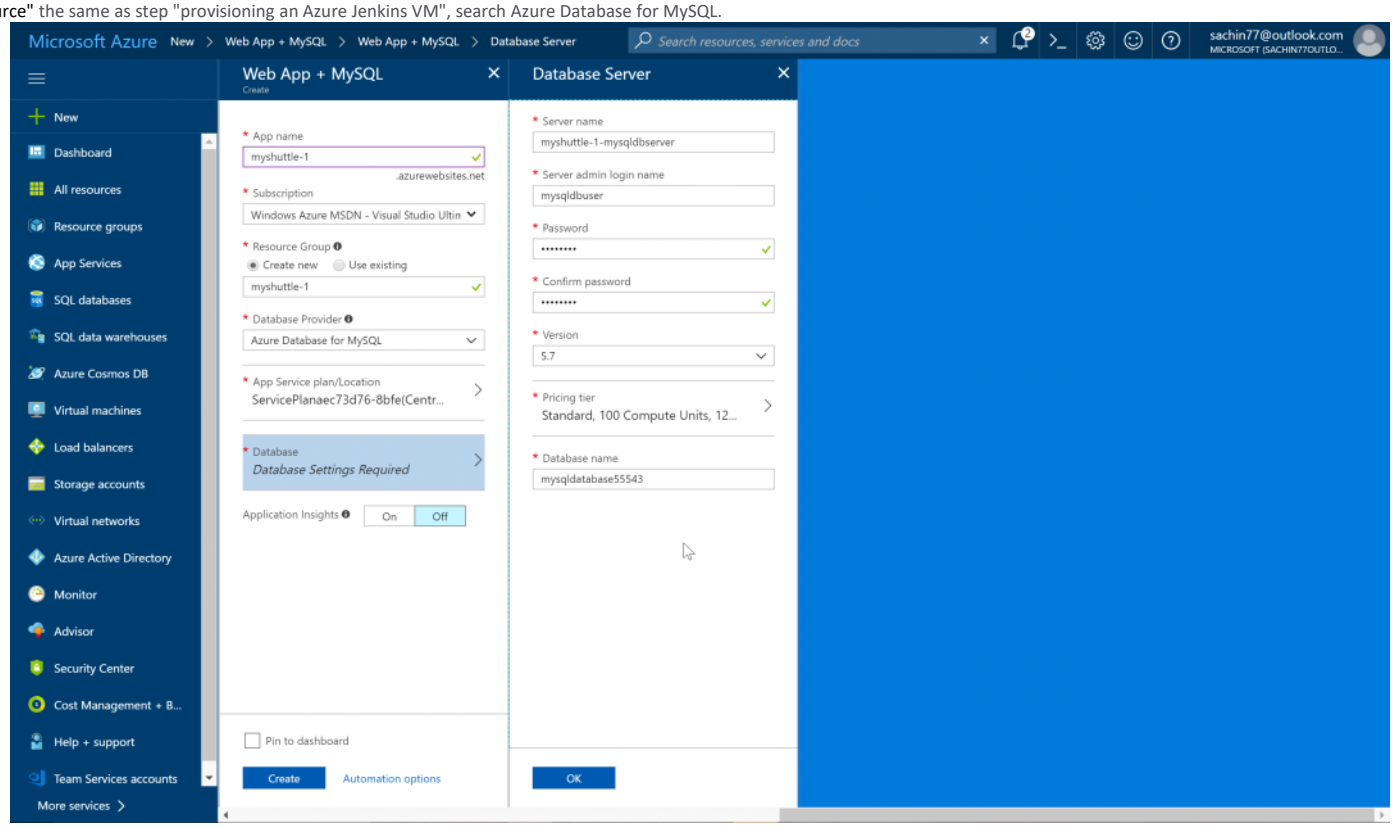
2. Otherwise, point this to the Azure CI build pipeline from which the Jenkins CI is executed.

5. Now, the artifact is linked for deployment. Please refer the [Deploying a MySQL Backed Tomcat app on Azure Web App]lab for deploying the WAR file to Azure App Service.

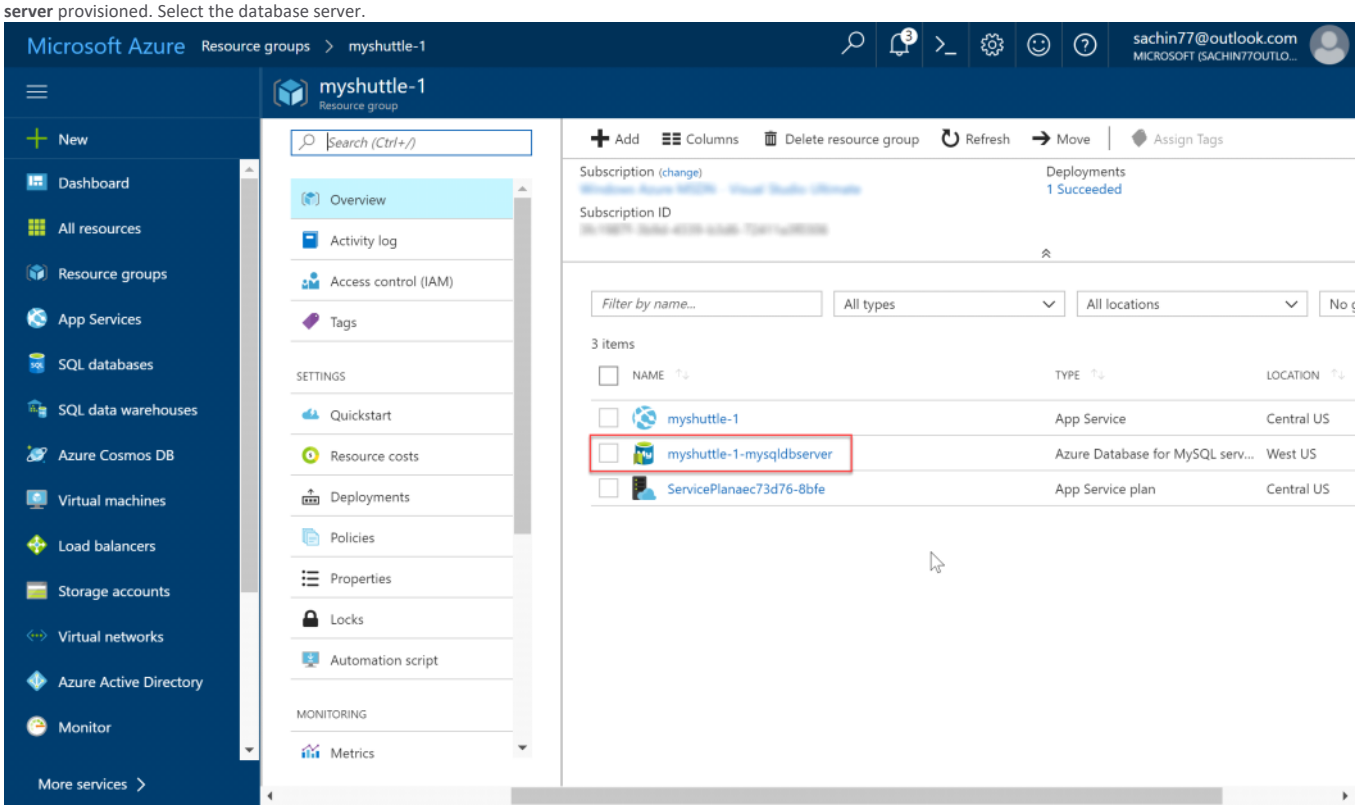
Creating Azure Web App and MySQL database

Tuesday, April 16, 2019 3:23 PM

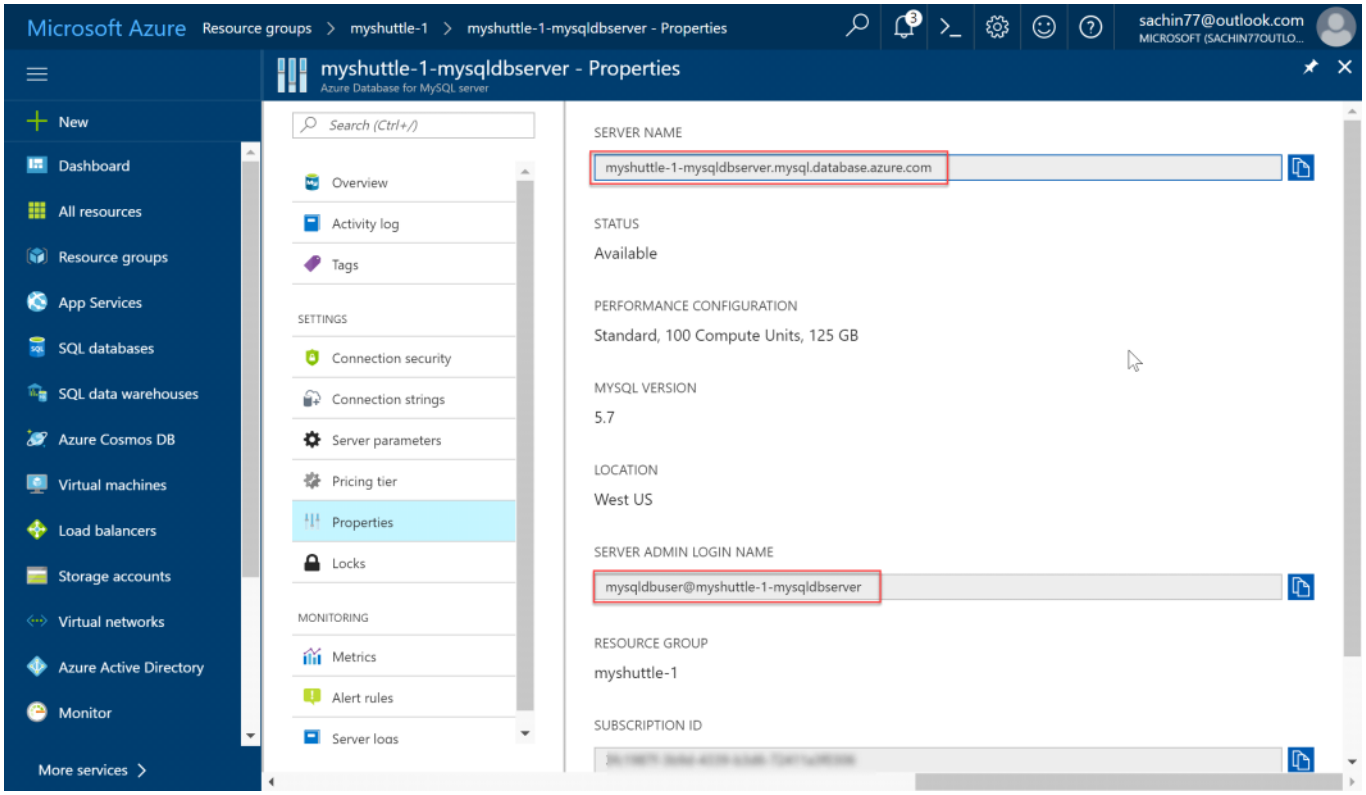
1. Click on create new the below button to provision a Website app + MySQL Database together to "Create a resource" the same as step "provisioning an Azure Jenkins VM", search Azure Database for MySQL.



2. Wait for the Web App and the database to be provisioned. It roughly takes 3-5 minutes.
3. Navigate to the resource group that you have created. You should see a **Azure Database for MySQL server** provisioned. Select the database server.



4. Select **Properties**. Save the **SERVER NAME** and **SERVER ADMIN LOGIN NAME** to a notepad.

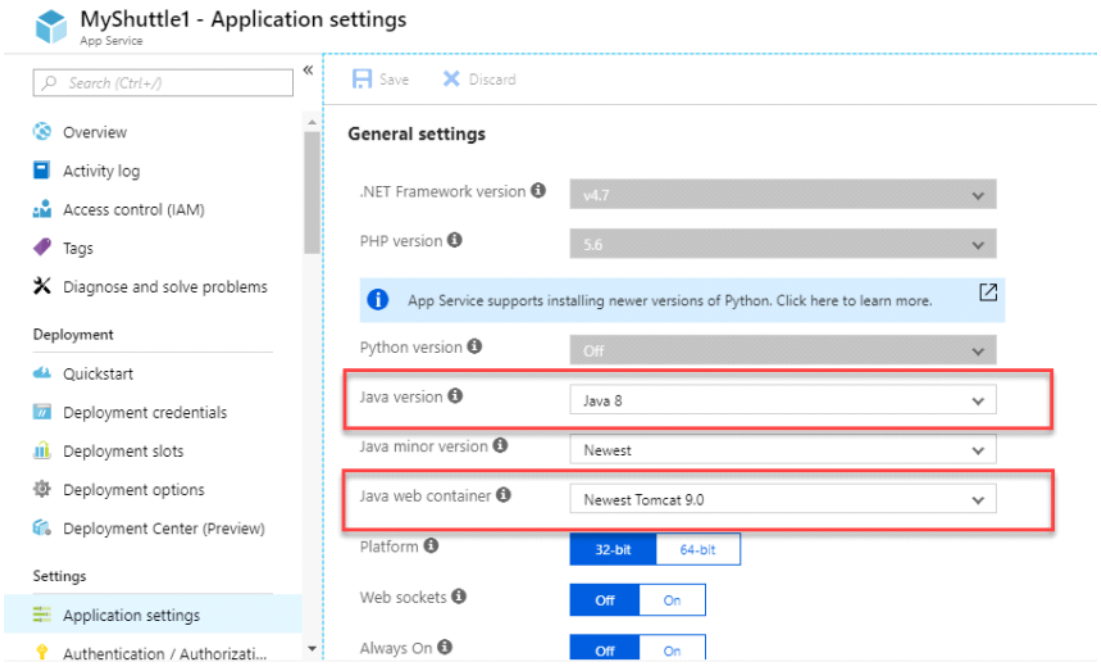


In this example, the server name is **myshuttle-1-mysqldbserver.mysql.database.azure.com** and the admin user name is **mysqldbuser@myshuttle-1-mysqldbserver**.

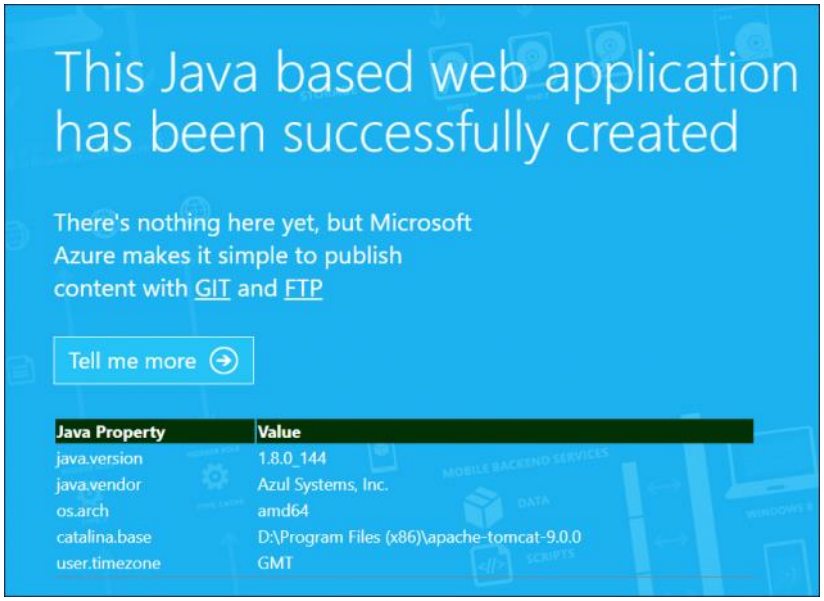
Updating the App Settings for the Web App

Next, navigate to the Web app that you have created. As you are deploying a Java application, you need to change the web app's web container to Apache Tomcat.

1. Click **Application Settings**. To change it to Tomcat, you will first need to install Java. Select a **Java Version** to install and then change **Web container** to use Apache Tomcat. For this purpose of the lab, you will choose ***Java 8*** and ***Apache Tomcat 9.0*** though the version number would not matter much for the simple app that we are deploying.

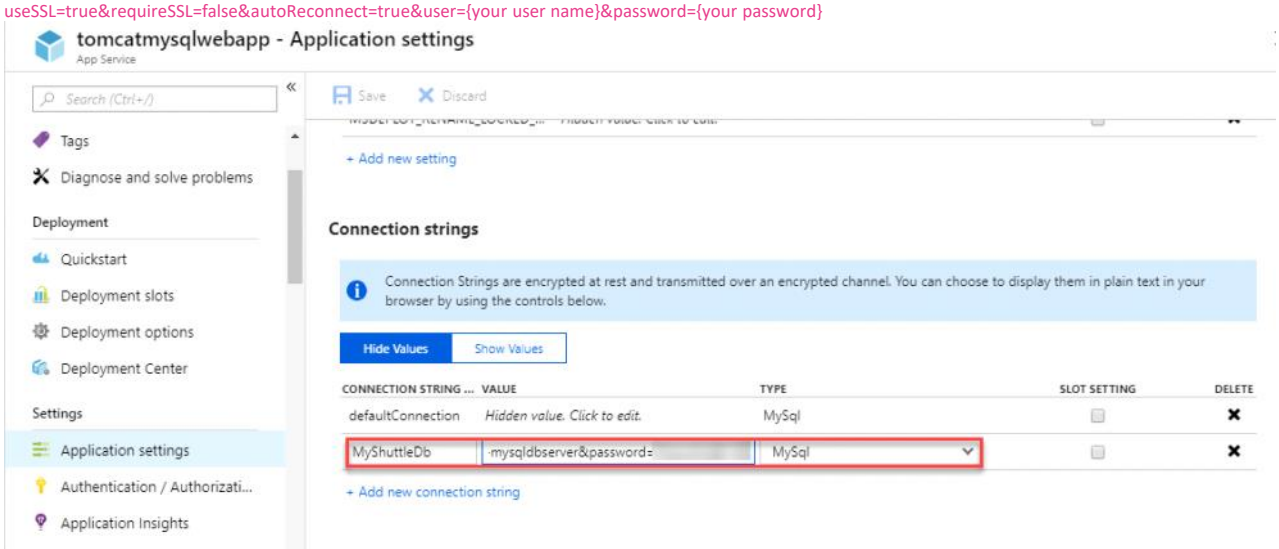


2. Click on **Save** and wait for the update to be applied. The web page will now look like the below image.



Next, you need to update the connection strings for the web app to connect to the database correctly. There are multiple ways you can do this - but for the purpose of this lab, you will take a simple approach by updating it directly on the Azure portal.

3. From the Azure portal, select the Web app you provisioned. Select **Application Settings** and scroll down to the **Connection Strings** section.
4. Add a new **MySQL** connection string with **MyShuttleDb** as the name, paste the following string for the value and replace **MySQL Server Name**, **your user name** and **your password** with the appropriate values - jdbc:mysql://{MySQL Server Name}:3306/alm?useSSL=true&requireSSL=false&autoReconnect=true&user={your user name}&password={your password}



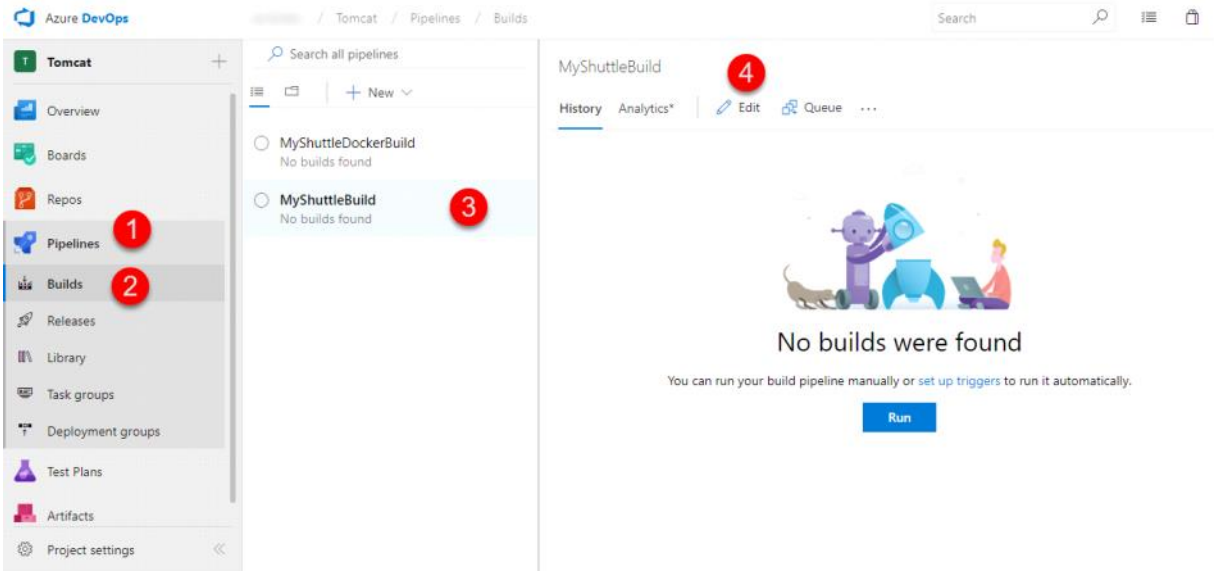
   - MySQL Server Name : Value that you copied previously from the MySQL server Properties.
   - your user name : Value that you copied previously from the MySQL server Properties.
   - your password : Value that you provided during the creation of MYSQL database server in the *Deploy to Azure* phase.

5. Click on **Save** to save the connection string.
   Note: Connection Strings configured here will be available as environment variables, prefixed with connection type for Java apps (also for PHP, Python and Node apps). In the **DataAccess.java** file under **src/main/java/com/microsoft/example** folder, we retrieve the connection string using the following code
   String conStr = System.getenv("MYSQLCONNSTR_MyShuttleDb");

You have now setup and configured all the resources that is needed to deploy and run the MyShuttle application.

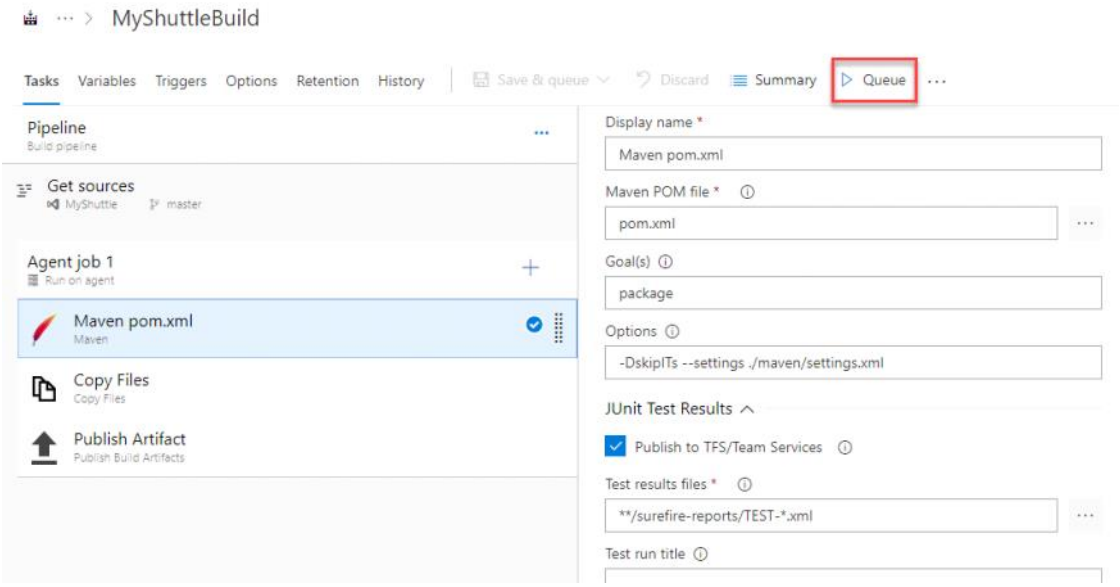
Deploy the changes to Web App

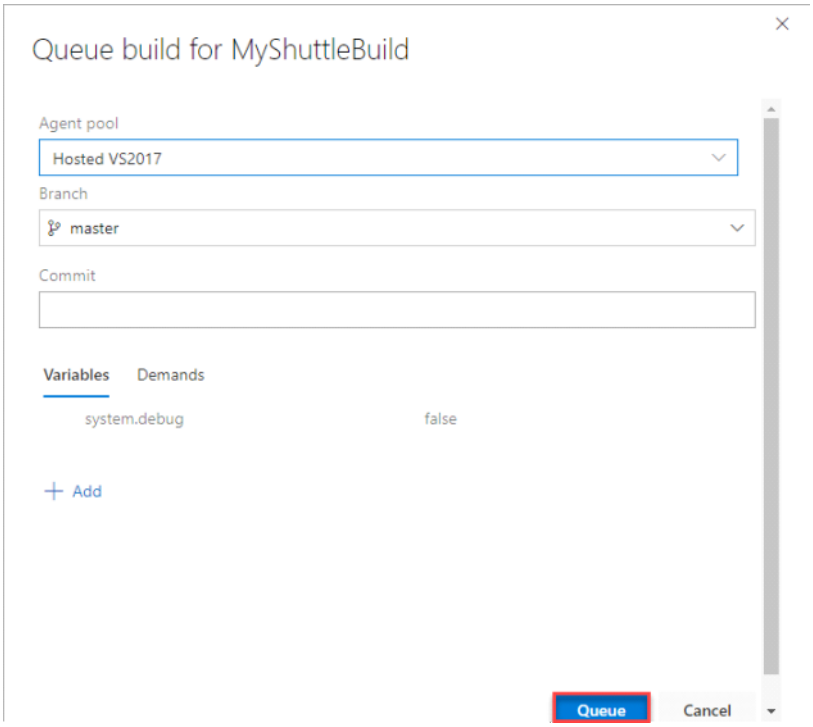
1. Select **Pipelines** and then, **Builds**. Choose the build **MyShuttleBuild** and click **Edit Pipeline** to view the build definition.



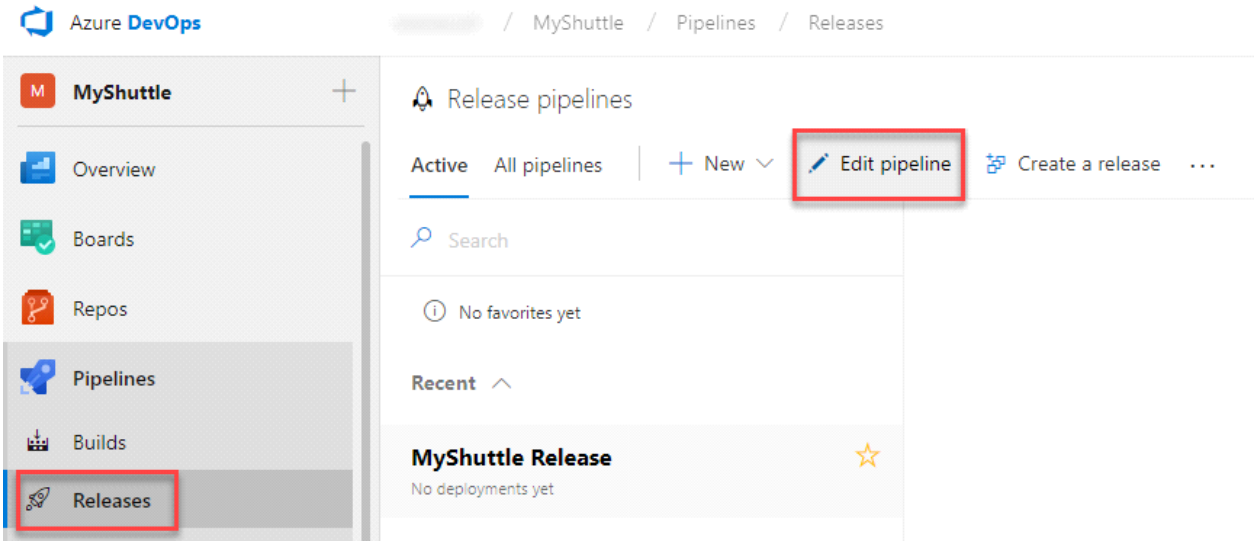
The lab uses the standard **Maven** build template to compile the code, copy and publish the resulting artifacts for deployment. An additional file which is copied here is the *CreateMYSQLDB.sql* file which creates a MYSQL database and inserts a few records into it during the deployment.

2. Click **Queue** to queue the build and wait for the build to complete.

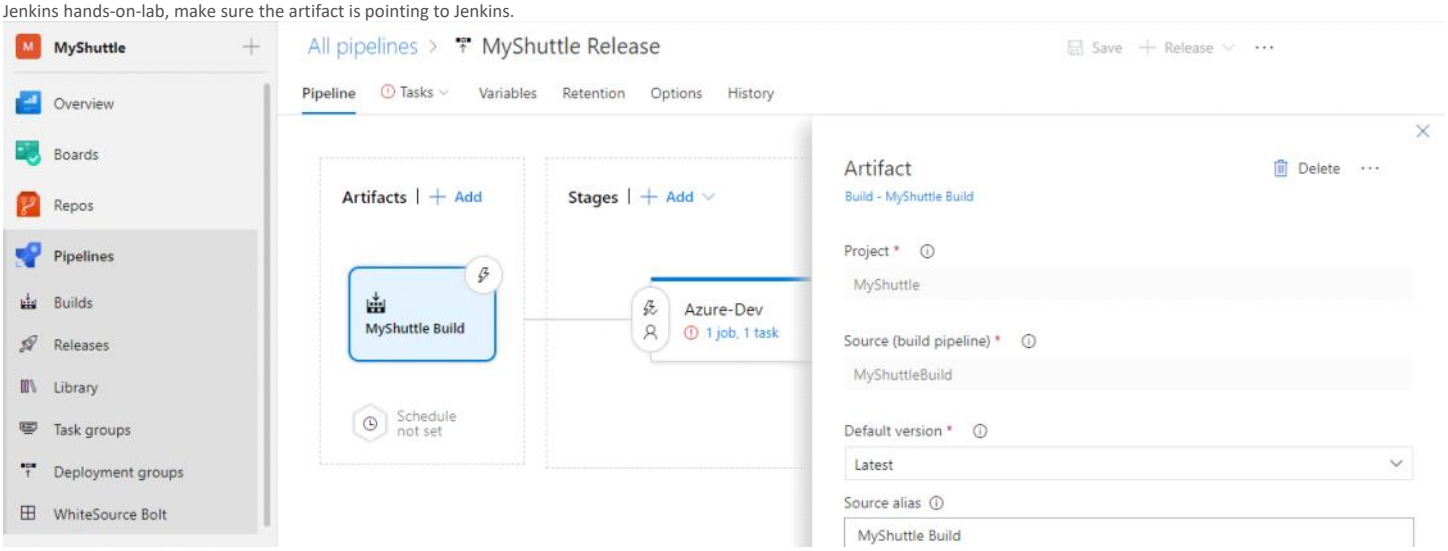




3. Once the build succeeds, Select **Releases** under **Pipelines**.
4. Select **MyShuttle Release** and click **Edit Pipeline** to open the release definition.

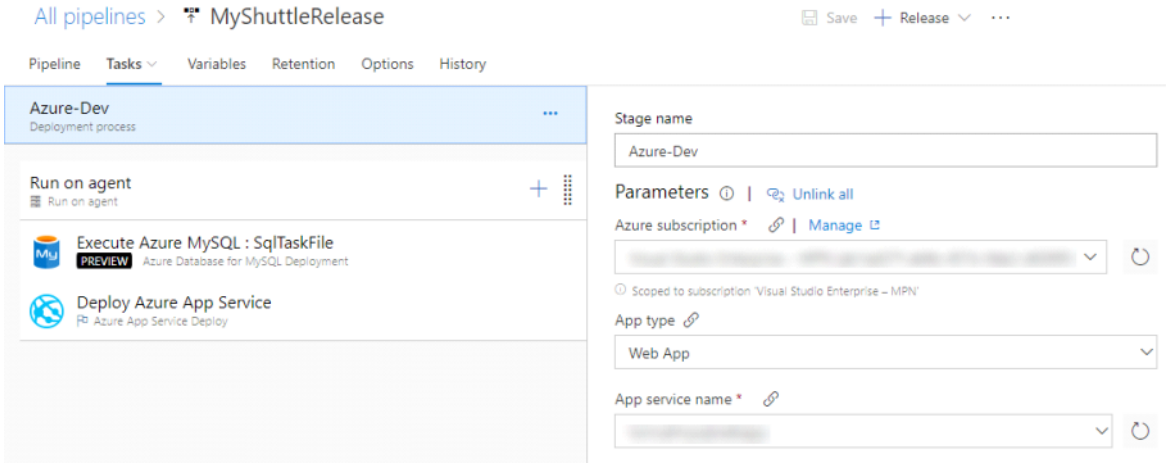


5. Make sure the artifact is pointing to the **Build** artifact as shown below. If you are following this lab from Jenkins hands-on-lab, make sure the artifact is pointing to Jenkins.

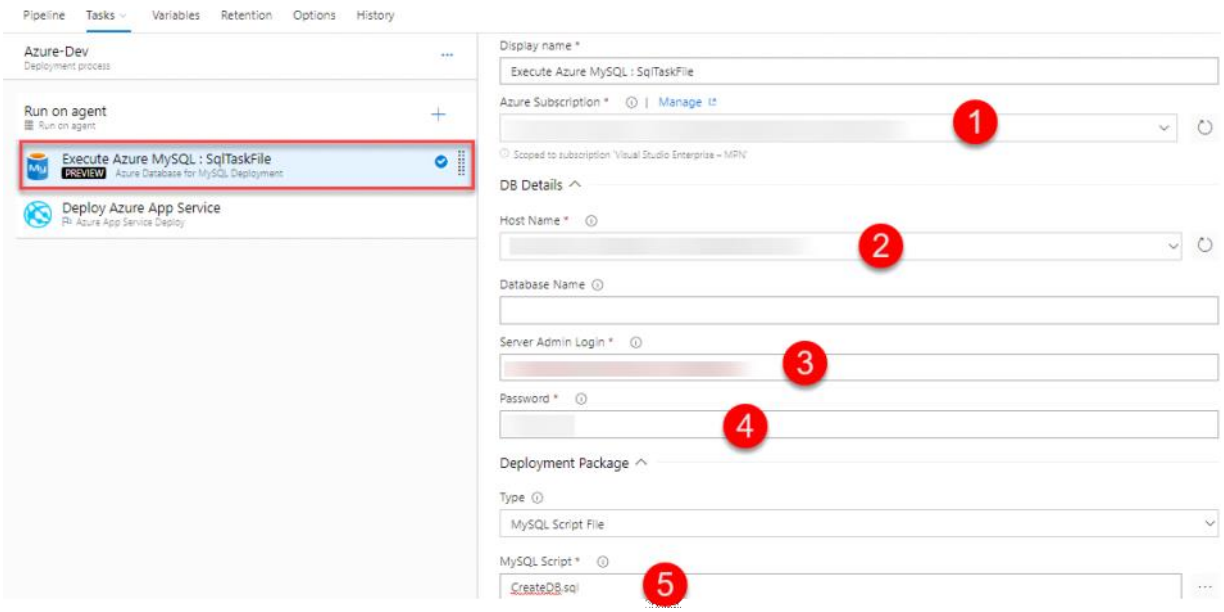


6. Click **Tasks**, select **Azure-Dev** and choose the **Azure subscription** details from the drop down.

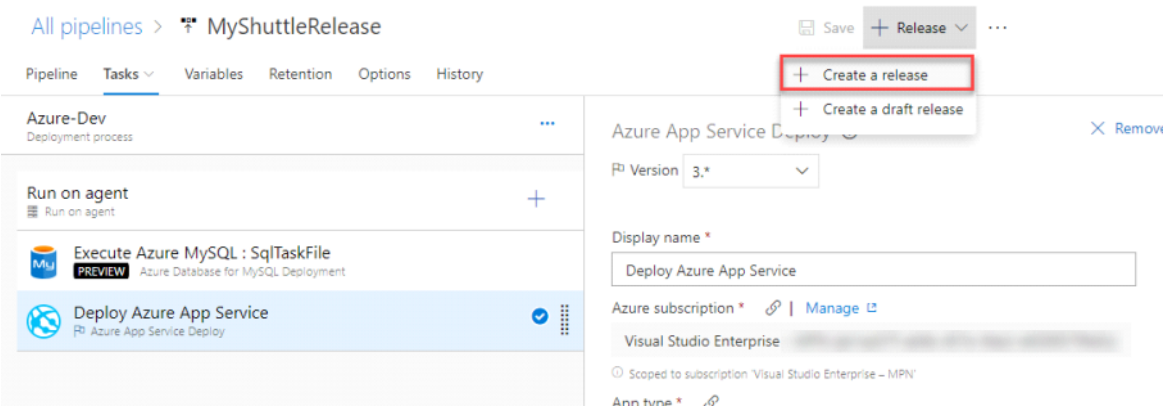
Click **Authorize** and login to your Azure subscription in the pop-up window. Provide or choose the created **App Service Name** with the web app that you created previously in this lab.



7. Click **Tasks** and select **Execute Azure MySQL : SqlTaskFile** task and provide the following details.
   - Azure Subscription Details : Select the appropriate subscription.
   - Host Name : Select the **MySQL Database server** host name that was created.
   - Server Admin Login : Provide the **SERVER ADMIN LOGIN NAME** that you noted down previously.
   - Password : Provide the password that you created during the creation of *Azure Web App + MYSQL* database server in the Azure portal.



8. Select the **Deploy Azure App Service** task and ensure that the created **App service name** is reflected correctly.
   Note: We are using the **Deploy Azure App Service** task. This task is used to update Azure App Service to deploy Web Apps and WebJobs to Azure. The task works on cross platform agents running Windows, Linux or Mac and uses the underlying deployment technologies of Web Deploy and Kudu. The task works for ASP.NET, ASP.NET Core 1 and Node.js based web applications. Note that this task works with Azure Resource Manager APIs only.
9. Click on **Save** and then **+Release | Create Release** to start a new release



10. Wait for the release to complete. Then navigate to the Web App and select the **URL** from the overview blade. Add **/myshuttledev** context to the URL. For instance - http://myshuttle1.azurewebsites.net/myshuttledev
11. Select **Login** and try logging in to the site with any one of the following credentials.

Username	Password
barney	barneypassword
fred	fredpassword