

Configuring a CD pipeline for your Jenkins CI

Overview

[Jenkins](#) is a very popular Java-based open source continuous integration (CI) server that allows teams to continuously build applications across platforms. Azure Pipeline includes the ability to build any application on any platform including Windows, Linux and Mac. However, it also integrates well with Jenkins for teams who already use or prefer to use Jenkins for CI.

There are two ways to integrate Jenkins with Azure Pipelines:

- One way is to **run CI jobs in Jenkins** separately. This involves the configuration of a CI pipeline in Jenkins and a webhook in Azure DevOps that invokes the CI process when source code is pushed to a repository or a branch.
- The alternate way is to **wrap a Jenkins CI job inside an Azure pipeline**. In this approach, a build definition will be configured in Azure Pipelines to use the **Jenkins** tasks to invoke a CI job in Jenkins, download and publish the artifacts produced by Jenkins.

An Azure CD pipeline can be configured to pick these build artifacts, irrespective of the approach, for deployment. While there are pros and cons with both the approaches, the latter approach has multiple benefits:

1. End-to-end traceability from work item to source code to build and release
2. Triggering of a Continuous Deployment (CD) when the build is completed successfully
3. Execution of the build as part of the branching strategy

What's covered in this lab

This lab covers both the approaches and the following tasks will be performed

- Provision Jenkins on Azure VM using the Jenkins template available on the Azure Marketplace
- Configure Jenkins to work with Maven and Azure DevOps
- Create a build job in Jenkins
- Configure Azure Pipeline to integrate with Jenkins
- Configure a CD pipeline in the Azure Pipelines to deploy the build artifacts

Before you begin

1. **Microsoft Azure Account:** You will need a valid and active Azure account for the Azure labs. If you do not have one, you can sign up for a [free trial](#)
2. [Putty](#) a free SSH and Telnet client

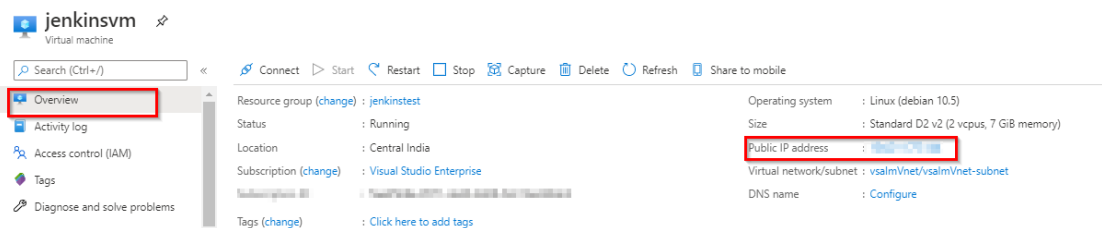
3. Set up your Azure DevOps project using the **MyShuttle** template in the [Azure DevOps Demo Generator](#). We will use a Java web app that connects to a MySQL backend.

Setting up the Jenkins VM

1. To configure Jenkins, the Jenkins VM image available on the Azure Marketplace will be used. This will install the latest stable Jenkins version on an Ubuntu Linux VM along with the tools and plugins configured to work with the Azure. Click on the **Deploy to Azure** button below to get started. Note: After clicking create, choose Password as the Authentication Type. Make note of the username and password.

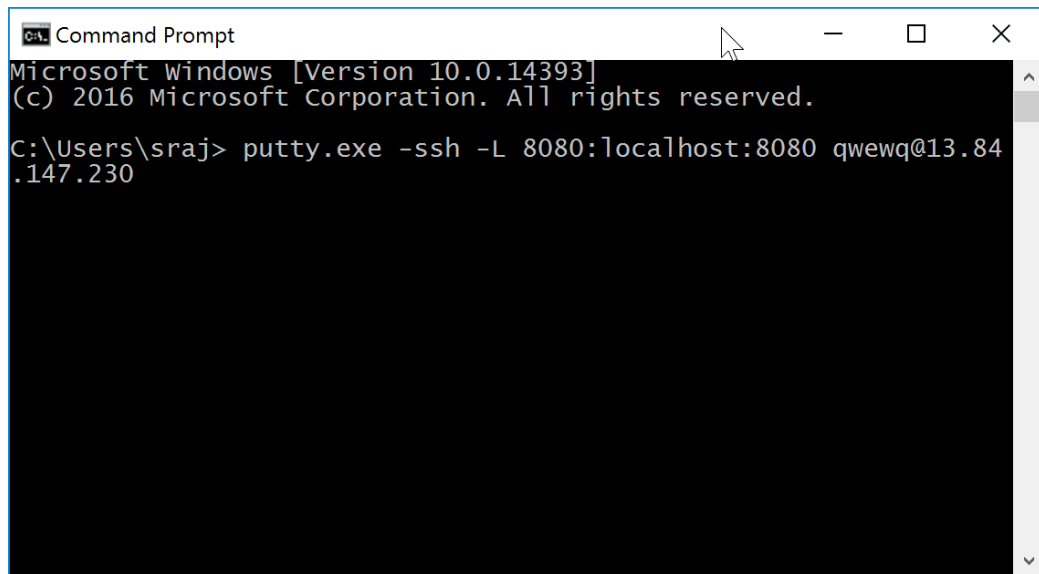


2. Once the Jenkins VM is provisioned, navigate to the VM's overview page and make a note of the **public IP address**. This information will be required to connect to the Jenkins VM from **Putty**



Note: Jenkins, by default, listens on port 8080 using HTTP. To configure a secure HTTPS connection, an SSL certificate will be required. If HTTPS communication is not being configured, the best way to ensure that the sign-in credentials are not leaked due to a "Man-in-the-middle" attack is by logging-in using the SSH tunnelling. An SSH tunnel is an encrypted tunnel created through an SSH protocol connection, that can be used to transfer unencrypted traffic over an unsecured network.

3. 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. Note: If you have a website on 8080, you may need to choose another port.
4. `putty.exe -ssh -L 8080:localhost:8080 <username>@<ip address>`



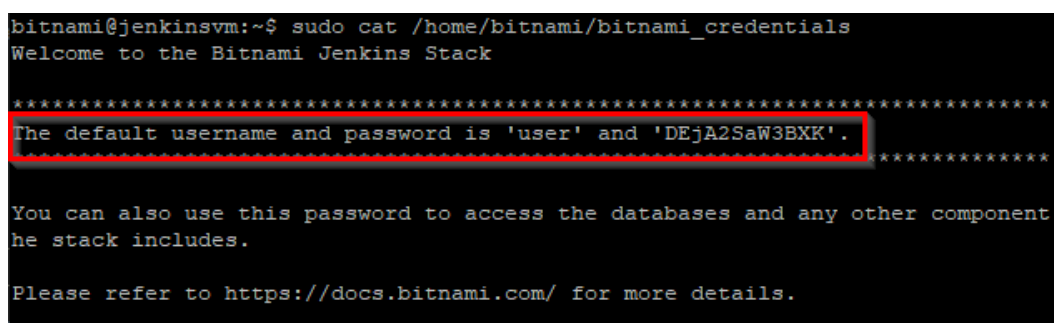
```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\sraj> putty.exe -ssh -L 8080:localhost:8080 qwewq@13.84
.147.230
```

Note: To run the above command, either the Putty.exe needs to be placed in the path selected in the Command Prompt or the full path of the Putty.exe need to be provided in the command.

5. Log in with the user name and password that you have provided while provisioning the Jenkins VM.
6. Once the connection is successful, open a browser on the host machine and navigate to the URL <http://localhost:8080>. The **Sign in** page for Jenkins will be displayed.
7. For security reasons, the application credentials are stored in a standalone file in the VM. This credentials will need to be retrieved and provided to sign in to Jenkins. Return to the **Putty** terminal and type the following command to open the credentials file and copy the user name and password.

```
sudo cat /home/bitnami/bitnami_credentials
```



```
bitnami@jenkinsvm:~$ sudo cat /home/bitnami/bitnami_credentials
Welcome to the Bitnami Jenkins Stack

*****
The default username and password is 'user' and 'DEjA2SaW3BXK'.
*****

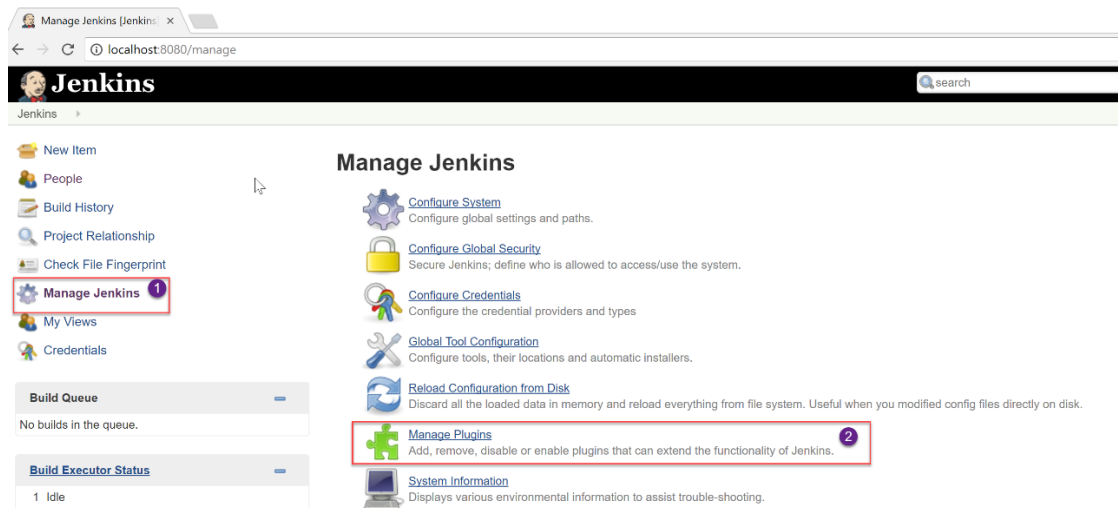
You can also use this password to access the databases and any other component
he stack includes.

Please refer to https://docs.bitnami.com/ for more details.
```

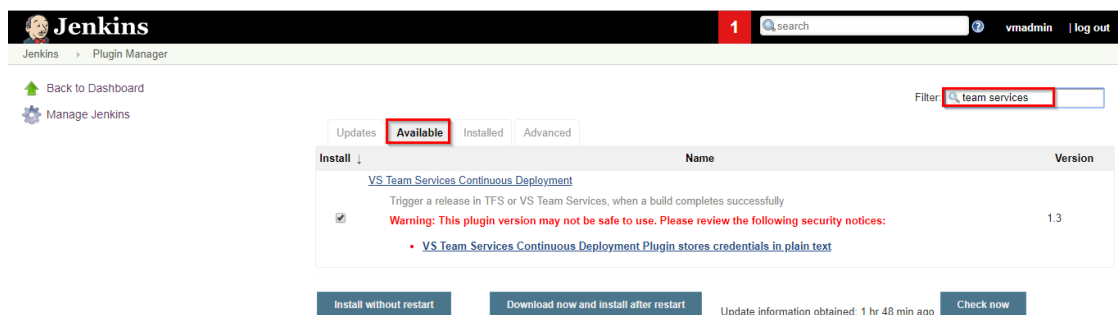
8. Return to the browser, paste the copied Username and Password to sign in to the Jenkins.

Installing and Configuring Plugins

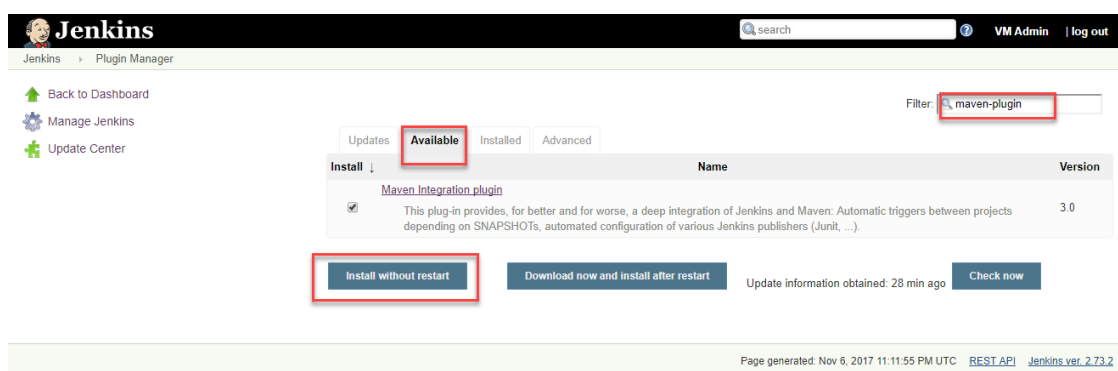
1. We will now install the Maven and VSTS (yet to be renamed Azure DevOps!) plugins that we require for this lab. Click **Manage Jenkins** on the Jenkins home page and select **Manage Plugins**. Select the **Available** tab and search for **team services**



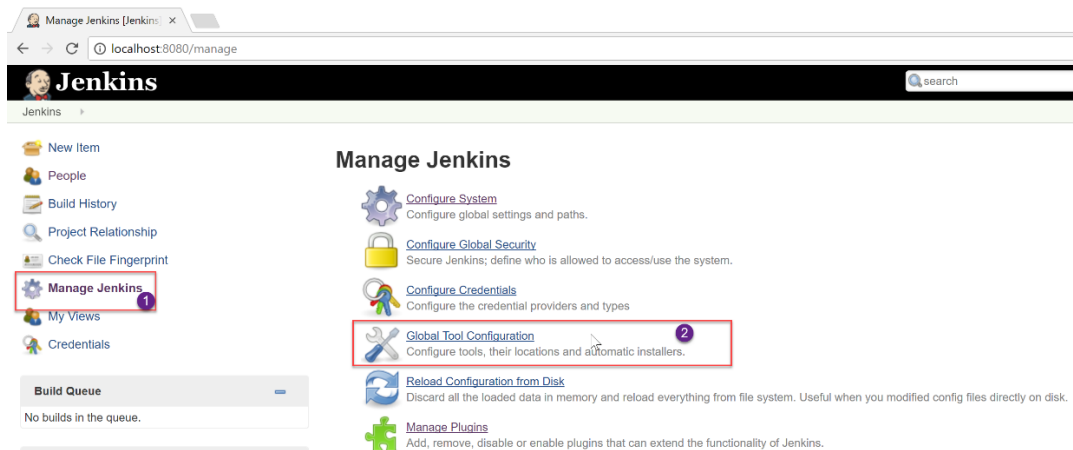
2. Select **VS Team Services Continuous Deployment** plugin and select **Install without restart**



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.

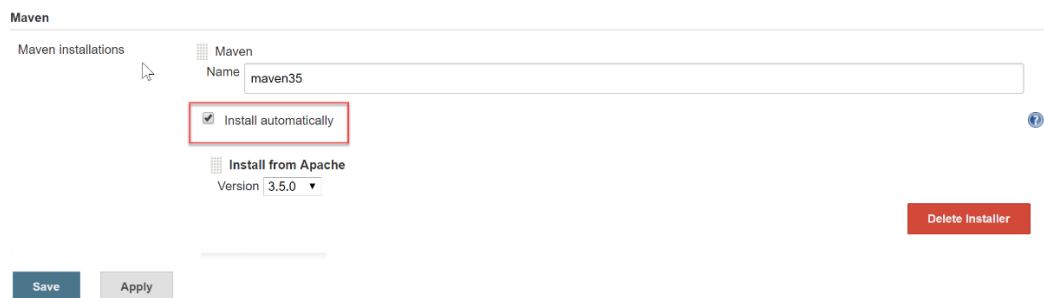


5. Once the plugin is installed, go back to **Manage Jenkins** and select the **Global Tool Configuration** option.



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.

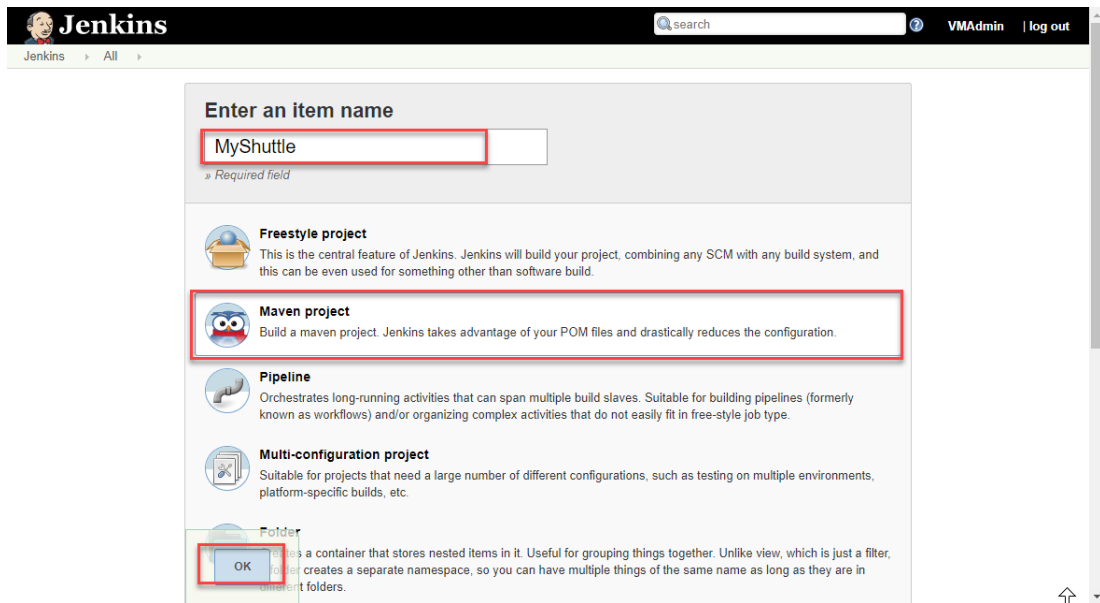
- To install Maven, select the **Install automatically** option and select the **Apply** button. The latest version of Maven at the time of writing this lab is 3.5.4



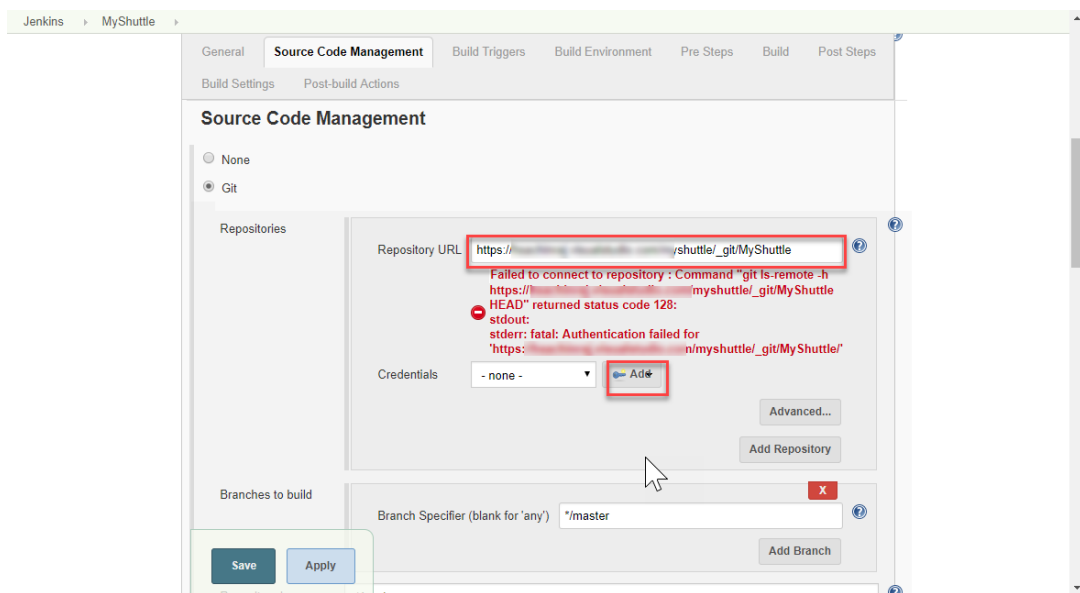
- Select the **Back to Dashboard** button to return to the home page. We are done with the setup. Let's go and create a new CI job.

Creating a new build job in Jenkins

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



- 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`. If you do not see **Git** under Source code management (not a usual thing), you will need to install/enable the Git plugin.



- 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 note down **Username** and **Password**.

Clone Repository

Command line

HTTPS

SSH

`https://sriramdasbalaji@dev.azure.com/s`


Username

Password


hwzia3kgvoaq

Copy the password now. We don't store it and you won't be able to see it again. [Learn more](#)

4. Select the **Add | Jenkins** option to add a new credential. Provide the **Username** and **Password** created in the previous step and click the **Add** button to close the wizard



Jenkins Credentials Provider: Jenkins

 **Add Credentials**

Domain

Global credentials (unrestricted)

Kind

Username with password

Scope

Global (Jenkins, nodes, items, all child items, etc)

Username

vmadmin

Password

.....

ID

Description

Add

Cancel

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

General **Source Code Management** Build Triggers Build Environment Pre Steps Build Post Steps Build Settings

Post-build Actions

Source Code Management

☐ None
☒ Git

Repositories

Repository URL

Failed to connect to repository : Command "git ls-remote -h /_git/MyShuttle HEAD" returned status code 128:
stdout:
stderr: fatal: Authentication failed for /_git/MyShuttle/'

Credentials Add

Advanced...

Add Repository

Save Apply

Branch Specifier (blank for 'any') */master

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.

Jenkins > MyShuttle >

General Source Code Management Build Triggers Build Environment Pre Steps **Build** Post Steps Build Settings Post-build Actions

Root POM

Goals and options

Advanced...

Post Steps

☐ Run only if build succeeds ☐ Run only if build succeeds or is unstable ☒ Run regardless of build result

Should the post-build steps run only for successful builds, etc.

Add post-build step

Build Settings

Save Apply

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**.

Pre Steps

Add pre-build step ▾

Build

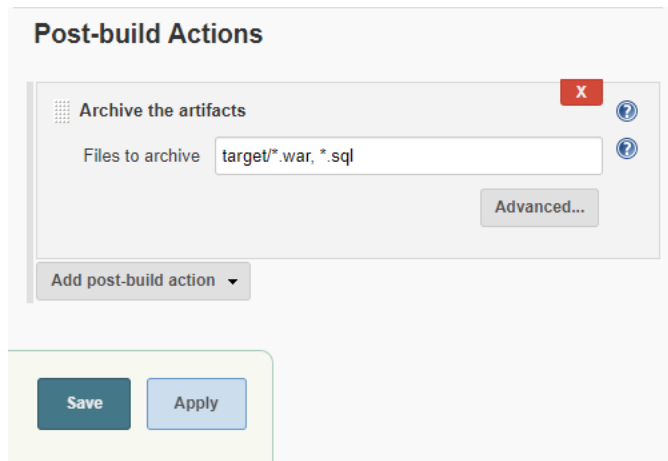
- Aggregate downstream test results
- Archive the artifacts**
- Build other projects
- Collect results for TFS/Team Services
- Deploy artifacts to Maven repository
- Perform an action if the job was performed on an Azure VM Agent.
- Publish an Azure Web App
- Record fingerprints of files to track usage
- Upload artifacts to Microsoft Azure Storage
- Git Publisher
- Add link to associated work items in TFS/Team Services
- Create a label in TFVC
- Editable Email Notification
- Set GitHub commit status (universal)
- Set build completion status in TFS/Team Services
- Set build status on GitHub commit [deprecated]
- Trigger release in TFS/Team Services
- Delete workspace when build is done

Add post-build action ▾

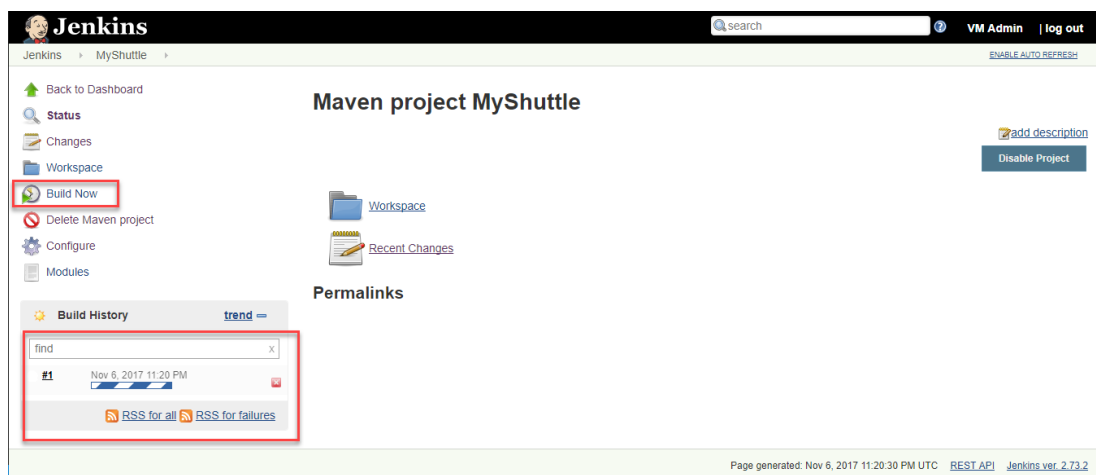
Save Apply

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.

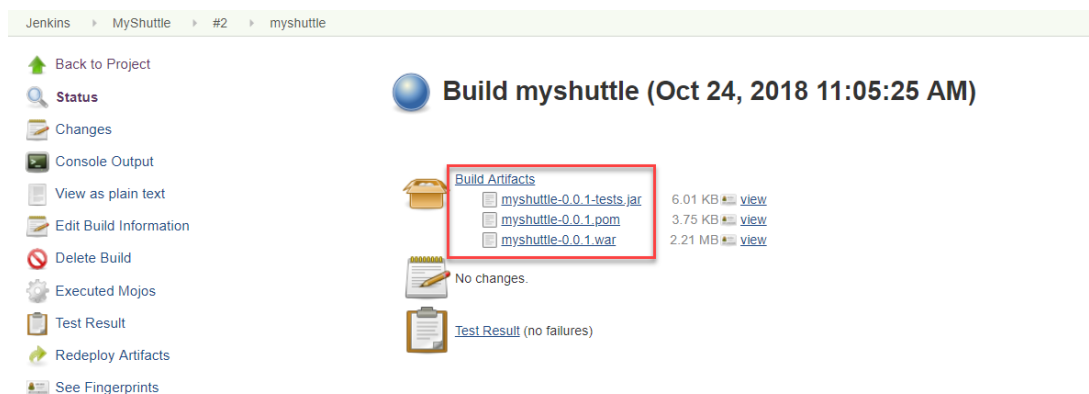
8. Enter **target/*.war, *.sql** in the **Files to archive** text box. Select the **Save** button to save the settings and return to the project page.



- 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



- To view the build details and the list of build artifacts, select the build number displayed in the **Build History** section.



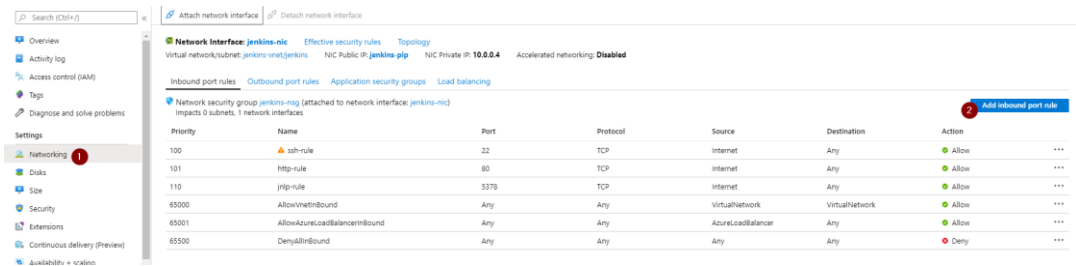
- Select the **Test Result** option to view the results of the unit tests that were included in the build definition.

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

Preparing Jenkins machine to use service hook in Azure DevOps

For the service hook in Azure DevOps to work - Jenkins machine should accept incoming connections to port 8080.

1. Open <https://portal.azure.com/> and access your virtual machine with Jenkins.
2. Click the "Networking" link in the Settings tab and click "Add inbound port rule" button



3. Set an inbound rule for port 8080

The screenshot shows the 'Add inbound security rule' form in the Azure portal. The form is titled 'Add inbound security rule' and has a 'Basic' tab selected. The fields are as follows:

- Source: Any
- Source port ranges: *
- Destination: Any
- Destination port ranges: 8080
- Protocol: Any
- Action: Allow
- Priority: 120
- Name: Port_8080
- Description: (empty)

The 'Add' button at the bottom is highlighted with a red circle and a number 2.

4. Navigate to your Jenkins page and go to **User | Configure**. Click on **Add new token** under API Token section and give some name and click **Generate**. Make a note of the Token generated. We would be using this Token as Jenkins password in Azure DevOps

The screenshot shows the Jenkins web interface. At the top, the Jenkins logo and a search bar are visible. The user 'user' is logged in, indicated by a red box and a red circle with the number 1. The left sidebar contains navigation links: People, Status, Builds, Configure (highlighted with a red box and a red circle with the number 2), My Views, and Credentials. The main content area shows the user configuration for 'user'. The 'Full Name' field contains 'user'. The 'Description' field is empty. The 'API Token' section shows 'Current token(s)' as 'There are no registered tokens for this user.' and an 'Add new Token' button (highlighted with a red box and a red circle with the number 3). Below this, a detailed view of the 'API Token' section is shown, with a text input field containing 'azure devops', 'Generate' and 'Cancel' buttons, and an 'Add new Token' button.

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**

NEW SERVICE HOOKS SUBSCRIPTION

Trigger

Select an event to trigger on and configure any filters.

Trigger on this type of event

Code pushed

Remember that selected events are visible to users of the target service, even if they don't have permission to view the related artifact.

FILTERS

Repository optional

MyShuttle

Branch optional

[Any]

Pushed by a member of group: optional

[Any]

Previous
Next
Test
Finish
Cancel

3. Provide the following details in the **Select and configure the action to perform** screen
 - a. Select the **Trigger generic build** option
 - b. Provide the **Jenkins base URL** in `Http://{ip address}:8080` format
 - c. Provide the **User name** and **User API Token** to trigger the build. Note that the username and Token are the credentials of the Jenkins user. For Token use, the API Token generated earlier from Jenkins
 - d. Select the **Build** job you created in Jenkins.

NEW SERVICE HOOKS SUBSCRIPTION

Action

Select and configure the action to perform.

Perform this action

Trigger generic build

Triggers a generic Jenkins build, invoking the Jenkins build URL. Secure, HTTPS endpoints are recommended due to the potential for private data in the event payload. [Learn more.](#)

SETTINGS

Jenkins base URL required

http://104.211.75.144:8080/

User name required

user

User API token (or password) required

.....

Build required

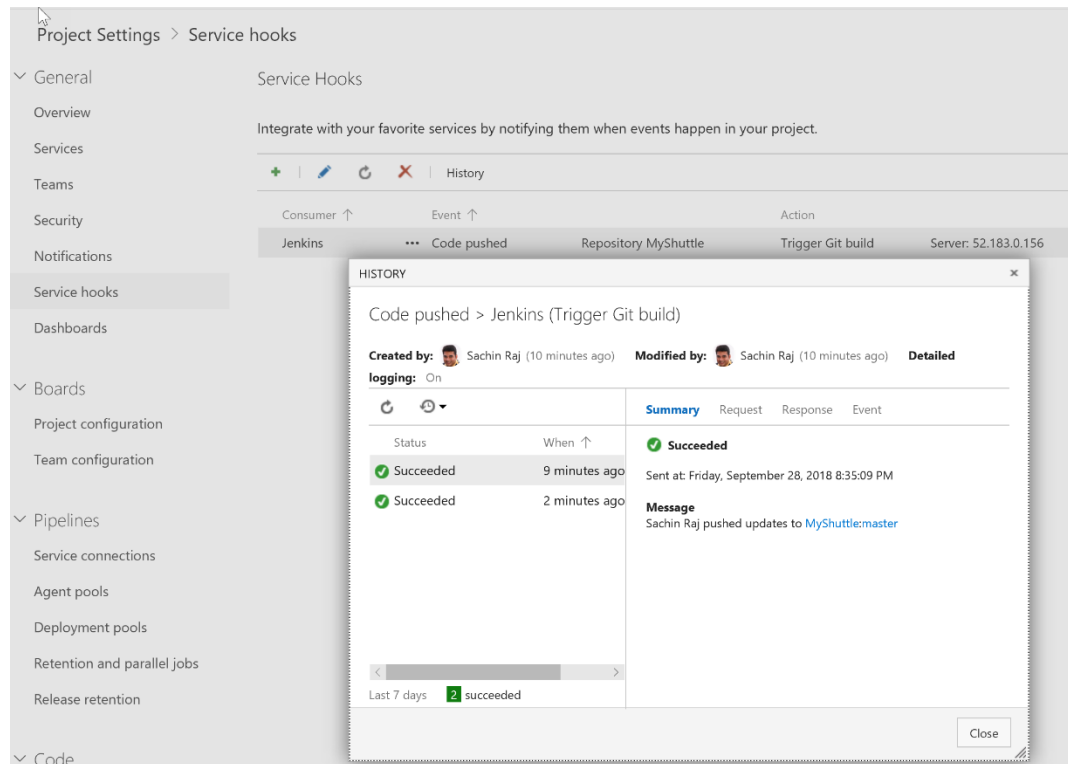
Myshuttle

Integration level optional

Run in Jenkins API

Previous
Next
Test
Finish
Cancel

- 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.
- 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 that 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.

- Go to your project settings. Select **Pipelines** and **Service connections**, click **New service connection** and choose **Jenkins** from the dropdown.
- Provide a connection name, Jenkins server URL in the format `http://[server IP address]:8080` and Jenkins user name with password (Use Jenkins User API Token as password). Select **Verify Connection** and validate the configuration. If it is successful, then select **Ok**.

New Jenkins service connection



Server URL

http://10.10.10.10:8080

☐ Accept untrusted SSL certificates (optional)

Allows the Jenkins clients to accept self-signed SSL server certificates without installing them into the TFS service role and/or Build Agent computers.

Authentication

Username

User

Username for connecting to the endpoint

Password

.....

Password for connecting to the endpoint

Verify

✓ Verification Succeeded

Details

Service connection name

MyJenkinsSerevr

Description (optional)

[Learn more](#)

[Troubleshoot](#)

Back

Verify and save



The next step would be to configure the build pipeline.

3. Go to **Pipelines | Pipelines**, Click **New Pipeline** to create a new build definition.
4. Select **Use the classic editor** to create a pipeline without a YAML.
5. Select **Myshuttle** project, repository and click *Continue*.
6. Scroll down and select the standard **Jenkins** template and then click on **Apply**.

Select a template

Or start with an **Empty job**

Search

Docker images to be pushed to a container registry.



Azure Web App for Java

Build a Java WAR file and deploy it to an Azure Web App.



C# Function

Build and test a C# (.NET class library) based Azure Function.



Go (preview)

Build a Go application.



Gradle

Build and test a Java project with Gradle.



Jenkins

Queue a Jenkins job and download its artifacts.



Load test using Azure IaaS virtual machines

Create a rig on Azure IaaS virtual machines to run load tests using VSTS cloud-based load testing service.

Apply

7. Select **vs2017-win2016** for the Agent specification, 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.

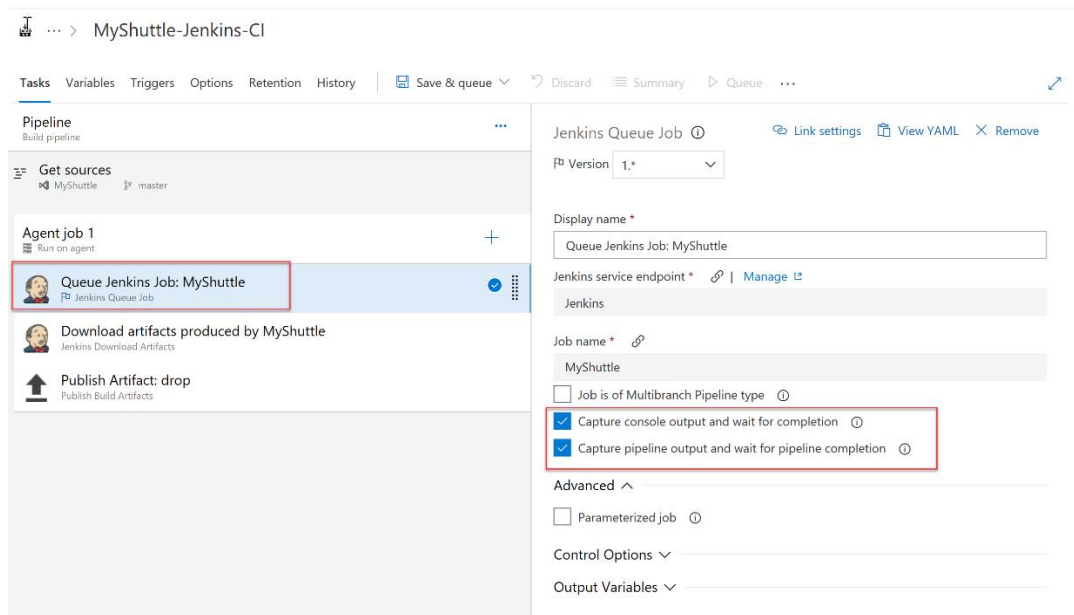
The screenshot shows the Jenkins pipeline configuration interface. The 'Pipeline' tab is active. The 'Agent Specification' dropdown is set to 'vs2017-win2016'. The 'Job name' field is 'MyShuttle'. The 'Jenkins service connection' dropdown is set to 'MyJenkinsServer'.

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.

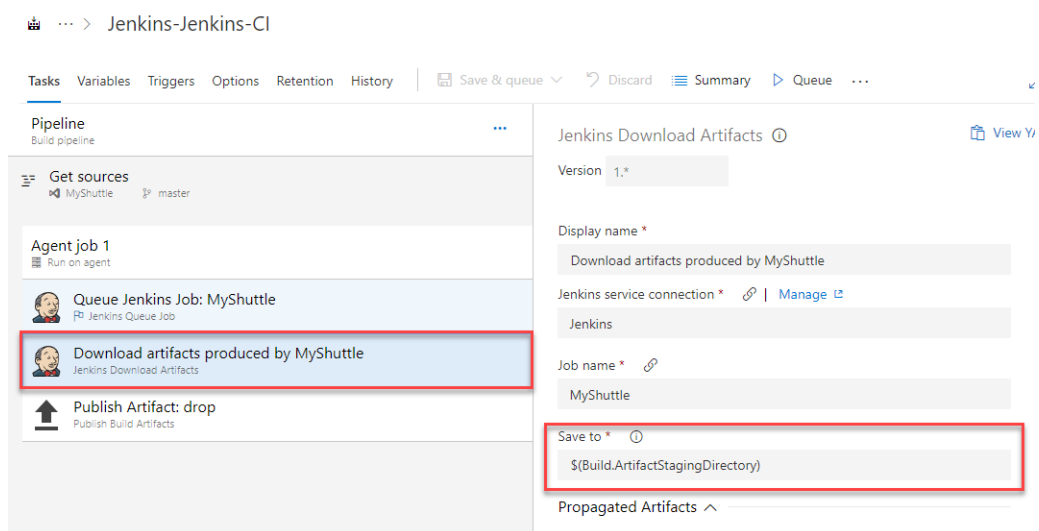
The screenshot shows the 'Get sources' step configuration. The 'Don't sync sources' checkbox is checked. The 'Repository' is set to 'MyShuttle'. The 'Default branch for manual and scheduled builds' is set to 'master'. The 'Clean' checkbox is unchecked. The 'Report build status' checkbox is checked. The 'Checkout submodules' checkbox is unchecked. The 'Checkout files from LFS' checkbox is unchecked. The 'Shallow fetch' checkbox is unchecked.

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 names 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.

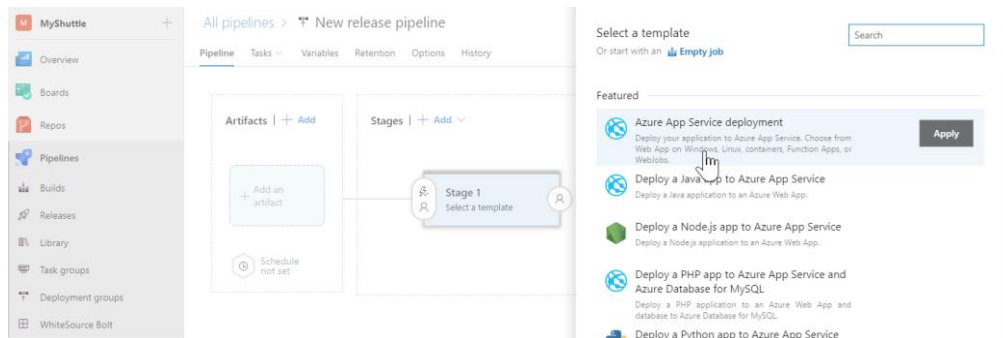


11. The **Publish Artifact drop** will publish to Azure Pipelines.
12. Click on **Save & queue** button to save and initiate a new build.

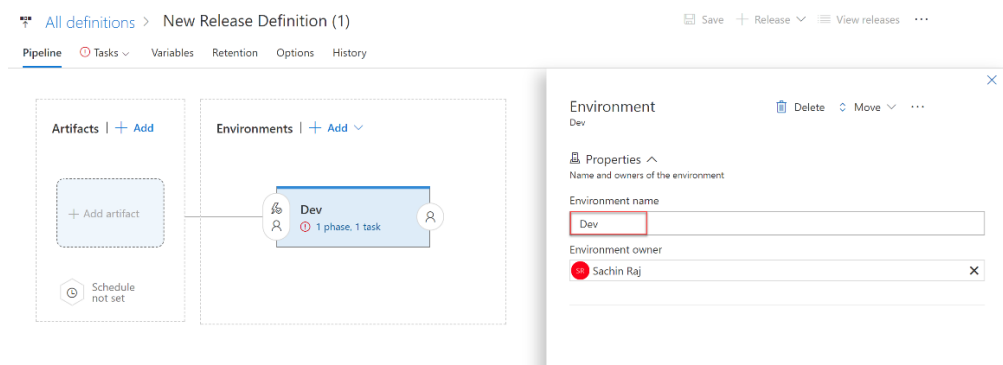
Linking the build artifact for deployment in a CD pipeline

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**. Click **+New** and select **New release pipeline** to create a new release pipeline.
2. Select the **Azure App Service Deployment** template.

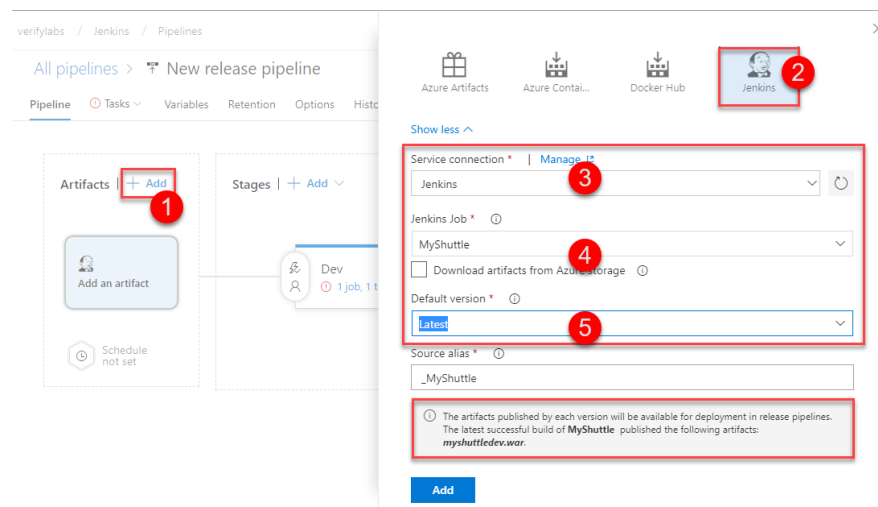


3. The default environment for deployment will be named as **Dev**



4. In the **Artifacts** section in the **Pipeline** tab, choose the **+ Add** link to select your build artifact.
 - a. 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.



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