Continuous Integration (CI) Pipeline

Part 1: preparing the system

1. Overview

This portion of the document describes the infrastructure setup necessary for a CI pipeline with Jenkins as the build configuration tool. You will create an AWS EC2 instance, connect to it through secure shell (SSH), and install Java, Tomcat, Maven, Git, and Jenkins. Part 2 will deal with Jenkins configuration.

2. Environment

SSH client: PuTTy or the built-in clients in a Linux command prompt (or using the new Windows ssh client feature)
https://www.howtogeek.com/336775/how-to-enable-and-use-windows-10s-built-in-ssh-commands/

Java: jdk-8u111-linux-x64.rpm (or later)

AWS EC2: Amazon Linux

Web container: Tomcat 8.5.28 or 9.0.5 (9 looks stable now), get the tar.gz format
https://tomcat.apache.org/download-80.cgi

Jenkins: v2.89, get the “generic Java package”
https://jenkins.io/download/

Maven: v3.5.2, get the tar.gz format

https://maven.apache.org/download.cgi

Version control: Git 64-bit

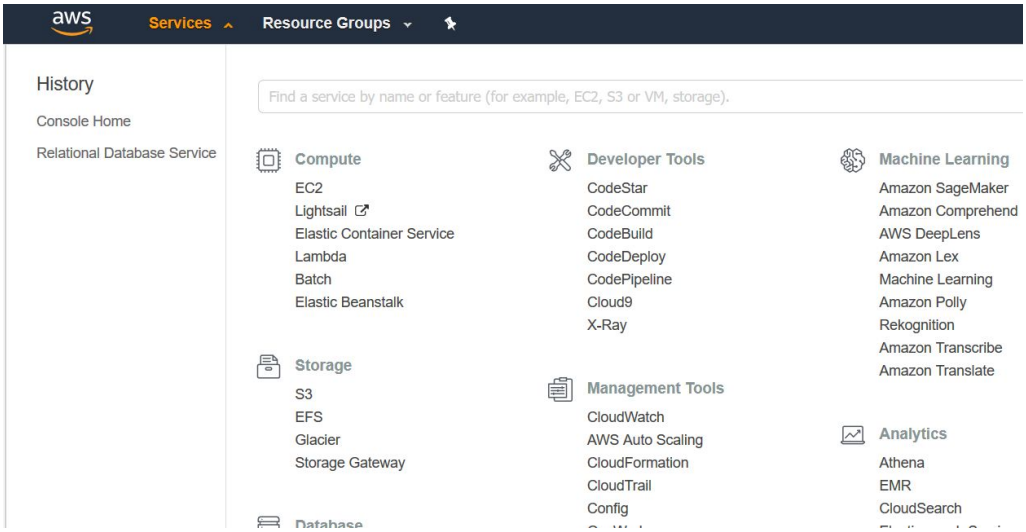
Cloud repository: GitHub

For your convenience, put all of the following files somewhere you'll be able to easily find them (a directory under your C:/ drive called, “Pipeline_Setup,” or similar is not a bad idea).

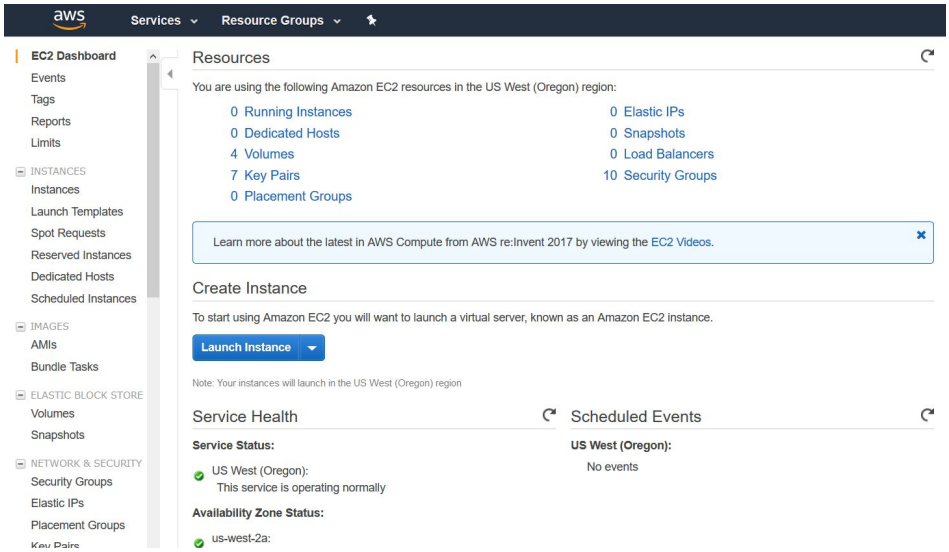
Note: versioning changes often - DON’T PANIC. Most releases are backwards compatible, but don’t go any older than the versions shown above. Specifically, earlier releases of Tomcat and Jenkins are known to cause issues in this pipeline structure.

3. EC2 creation

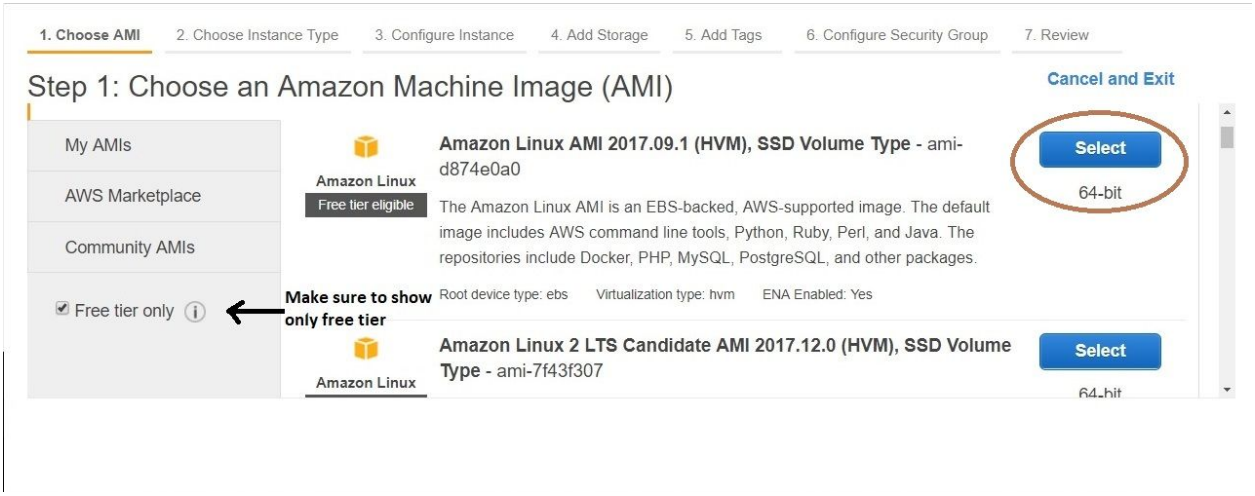
Elastic Cloud Compute (EC2) is a Virtual Machine (VM) on the Cloud, and is the basic building block for this CI pipeline. These are fully configurable with an Operating System (OS), RAM size, and security features.



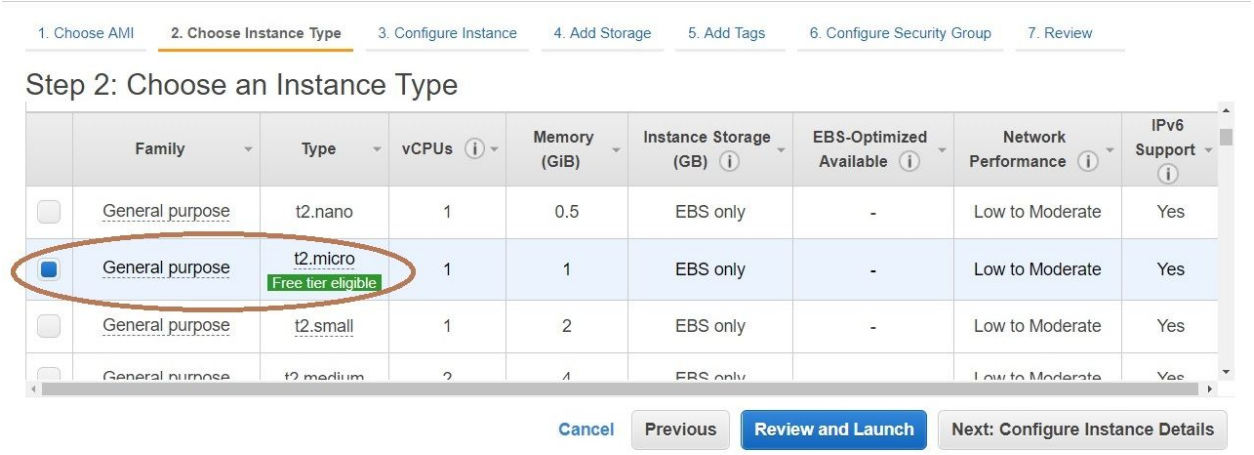
Go to Services in the upper left corner and choose EC2 to reach the EC2 dashboard. Hit the “Launch Instance” button to begin configuration.



Choose Amazon Linux for the operating system (AMI type).

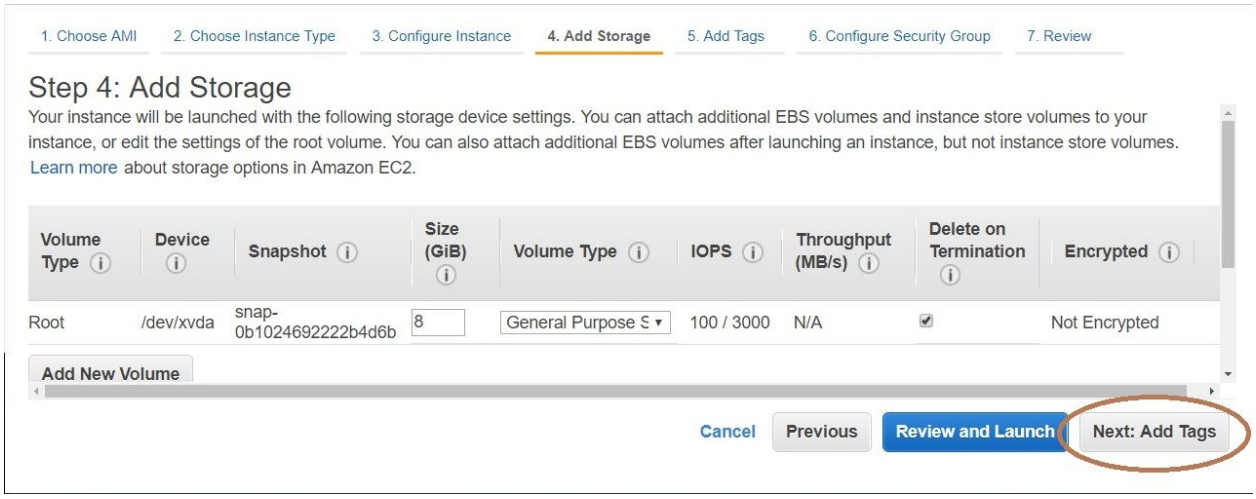


Choose the size of the VM (nano, micro, small, or medium). The t2.micro designation, which is in the Free Tier, represents a single, shared core so performance is limited and dependent on the load from other users.

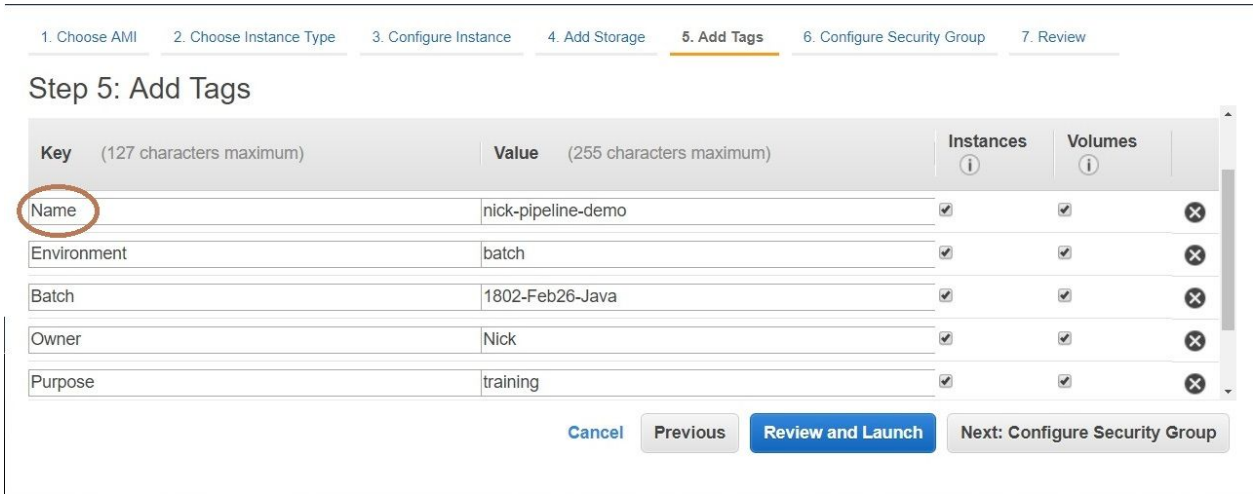


For the “configure instance details” step, simply keep the defaults then move on to the next step.

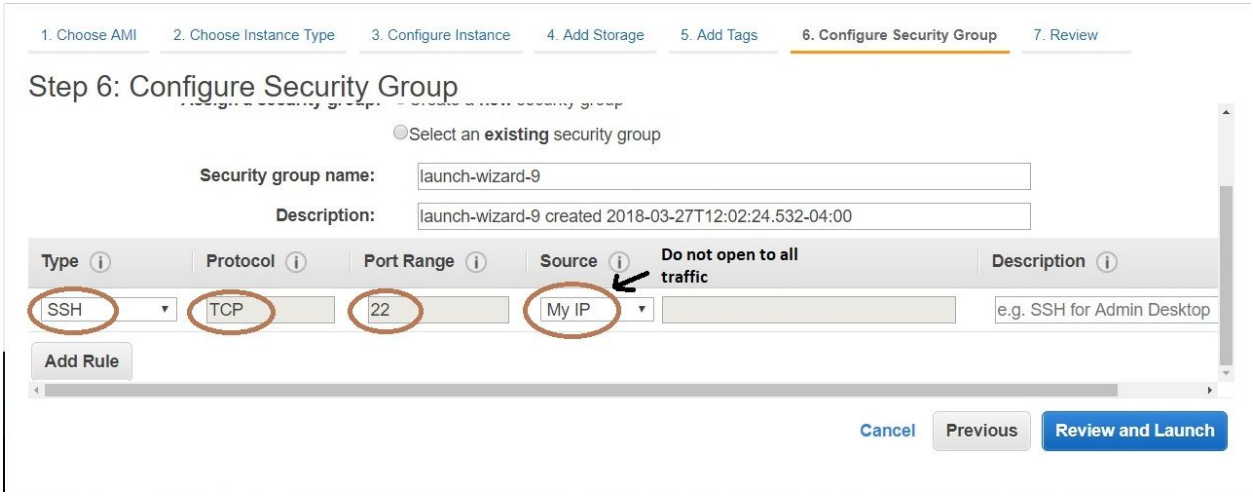
On the “add storage” step, keep the default settings to stay on the free tier. EBS is like the hard drive for your EC2 instance.



Give your instance a name with a tag called, “Name.” This is not strictly necessary, but tags can be used to organize instances across your AWS account.



Regarding security groups, leave your instance open to inbound and outbound traffic over HTTP on all ports inbound traffic over TCP on port 22, for all sources.



Finally, and this is very important, CREATE AND DOWNLOAD A NEW KEY PAIR. This is how you will access the instance via SSH.

4. Connect to EC2 via secure shell (SSH)

There are a few ways to accomplish this. If you are running Windows and like GUIs, PuTTY is a good option (method 1). If you prefer PowerShell or are running a Unix system, ignore PuTTY and go to method 2.

Method 1- PuTTY

PuTTY is Secure Shell (SSH) client that runs on a Windows machine to remotely connect it to a Linux machine in a secure manner. Using PuTTy, which opens a shell window to execute Linux commands, you can install the JDK, the web server, etc., on an EC2 instance.
PuTTY is located here: http://www.chiark.greenend.org.uk/~sgtatham/putty/. Install PuTTY on a local Windows machine by clicking on the Windows installer icon: “putty-0.67-installer”, and keeping all the defaults, which will install all the available features.

Convert the key file (.pem) downloaded from the AWS EC2 creation (Section 4.0) to a public/private key pair that PuTTY can use as follows:
• Invoke PuTTYgen from the Start menu of Windows.
• Go to the Conversions>>Import Key and find the pem file.
• Touch the “Save private key” button and then give the new file a name based on the pem file name such as: “EC2TomcatJenkinsPrivate.” You will then be prompted for a passphrase which isoptional.
• Repeat the previous step for the “Save public key” button.

Configuring PuTTy:
• Launch PuTTy from the desktop icon.

• Go to Session and copy the Public DNS to the Host Name field. Do not include a protocol prefix, and make sure the port is 22 and the SSH are selected.
• Go to Connection>>SSH>>Auth and navigate to the private key field at the bottom of the dialogue box. Browse to the private key created in Section 5.1.
• Go to Connection>>Data and enter “ec2-user” in the “Auto-login username” field. Note, this field is fixed by the type of EC2 operating system in use: Amazon Linux, Red Hat, Windows, etc.
• Return to Session and touch and create a name for this setup in the “Saved Sessions” field, and
touch “Save.” In the future, you can connect quickly using the connection setup of your choice and touching the “Load” button.
• Open a connection to the EC2 by touching the “Open” button at the bottom of the page. The first-time login screen then displays a warning.
• Click “Yes” to connect to the EC2 instance. For added security, you can manually check the fingerprint against the one for the AWS instance to make sure that they match. This PuTTY warning will not be displayed again for this connection setup.

Method 2 - Command-line SSH client

No one says it better than AWS themselves:
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstancesLinux.html

5. Set up directories in EC2

Note, opening a PuTTY session automatically lands the user in the /ec2-user/home, or /~. Use sudo to create the following directories as you will not be able to login as root:

For the JDK:	/usr/java
Tomcat/Maven:	/usr/apache
Maven repo:	/~/.m2
Git:	/~/.git

Example to create a new directory called java from inside the current directory:
sudo mkdir java

To give access to that directory:
sudo chmod 777 /usr/java
(see https://www.maketecheasier.com/file-permissions-what-does-chmod-777-means/ for reference)

To check if the permission applied:
ls -la
The folder should have drwxrwxrwx next to it if its permissions were set to 777 properly.

6. Transfer components to EC2

Now that you've established an ssh connection and set up the internal structure, you can copy the software we gathered in section 2 over to the EC2.

Transfer the JDK, Tomcat, and Maven archives from the local Windows machine to the /tmp of the EC2 instance. Open a command prompt and use the following putty command:
pscp -i path_to_private_key path_to_source_file
userid@server_name:path_to_destination_file_name

Note: make sure that if you're using PuTTy, you should make this command from within the PuTTy installation folder.

In other SSH clients, "scp" works because the "p" prefix refers to PuTTy.

Example to transfer the JDK archive to /tmp of the AWS EC2 instance:
pscp -i C:\pipeline_tools\[filename].ppk C:\pipeline_tools\jdk-8u111-linux-x64.rpm
ec2-user@[hostname]:/tmp/jdk-8u111-linux-x64.rpm

7. Installation on EC2

The files transferred in the last section aren't usable yet sitting in the tmp folder, they need to be installed.

7.1 Java

From /usr/java on the EC2 instance run the following:

sudo rpm -ivh /tmp/jdk-8u111-linux-x64.rpm

This will create three directories under /usr/java: default, latest, and jdk1.8.0_111. Each directory
contains the unpacked JDK files.

7.2 Git

Go to /~/.git and install the Git tool with the following command which places git.exe there:

sudo yum install git

7.3 Maven

Unpack Maven in /tmp on the EC2 instance by running the following:
Cd
tar xvf apache-maven-3.5.2-bin.tar.gz

Move the Maven folder to the Maven directory with this:

sudo mv apache-maven-3.5.2 /usr/apache
sudo chmod 777 /usr/apache/apache-maven-3.5.2

7.4 Tomcat

Unpack Tomcat in /tmp on the EC2 instance by running the following:
tar xvf apache-tomcat-8.5.28.tar.gz
Move the Tomcat folder to the Tomcat directory with this:
sudo mv apache-tomcat-8.5.28 /usr/apache
sudo chmod 777 /usr/apache/apache-tomcat-8.5.28

The Tomcat installation location, called CATALINA_HOME will be set in the next section.
Change the default CATALINA_HOME/conf/tomcat-users.xml to include two new passwords: one for the HTML login page at http://localhost:8080 to be used by the User and one for a text cd login to be used by Maven and Jenkins.
These can also be stored with encryption as well which is not discussed here.

```
<role rolename="manager-gui"/>
 <role rolename="manager-script"/>
 <user username="manager-gui" password="tomcat" roles="manager-gui"/>
 <user username="manager-script" password="script" roles="manager-script"/>
```

To relax the protection against CSRF, comment-out the <Valve className="..."/> element in CATALINA_HOME/webapps/manager/META-INF/context.xml. This will allow Jenkins to log into Tomcat's Manager Console from an IP address that is different than Tomcat's. Strictly speaking this step is only necessary for more advanced pipelines that have multiple servers on different EC2 instances.

Increase the file upload size from 50 MB to 134 MB by editing the multipart-config tags
ind/.project

CATALINA_HOME/webapps/manager/WEB-INF/web.xml. Change <max-file-size> and <max-request-size> to 134217728. This allows Jenkins to push a moderately sized .war file to the server.

8. Environment variables

Using the vi or nano editor, change .bashrc, a hidden file located in ~/, and add the following contents:

```
JAVA_HOME=/usr/java/latest
export JAVA_HOME
CATALINA_HOME=/usr/apache/apache-tomcat-8.5.28
export CATALINA_HOME
M2_HOME=/usr/apache/apache-maven-3.5.2
export M2_HOME

M2=$M2_HOME/bin
export M2
export PATH=$M2:$PATH
```

This file will be invoked after a new session is created with PuTTY and an interactive Bash shell is used.

Here is an excerpt from settings.xml showing the path to the local repository, which was edited by the user.

```
<!--
| This is the configuration file for Maven. It can be specified at two levels:
|
| 1. User Level. This settings.xml file provides configuration for a single user,
| and is normally provided in ${user.home}/.m2/settings.xml.
|
| NOTE: This location can be overridden with the CLI option:
|
| -s /path/to/user/settings.xml
|
| 2. Global Level. This settings.xml file provides configuration for all Maven
| users on a machine (assuming they're all using the same Maven
| installation). It's normally provided in
| ${maven.home}/conf/settings.xml.
|
| NOTE: This location can be overridden with the CLI option:
|
| -gs /path/to/global/settings.xml
| The sections in this sample file are intended to give you a running start at
| getting the most out of your Maven installation. Where appropriate, the default
| values (values used when the setting is not specified) are provided.
|
|-->
```

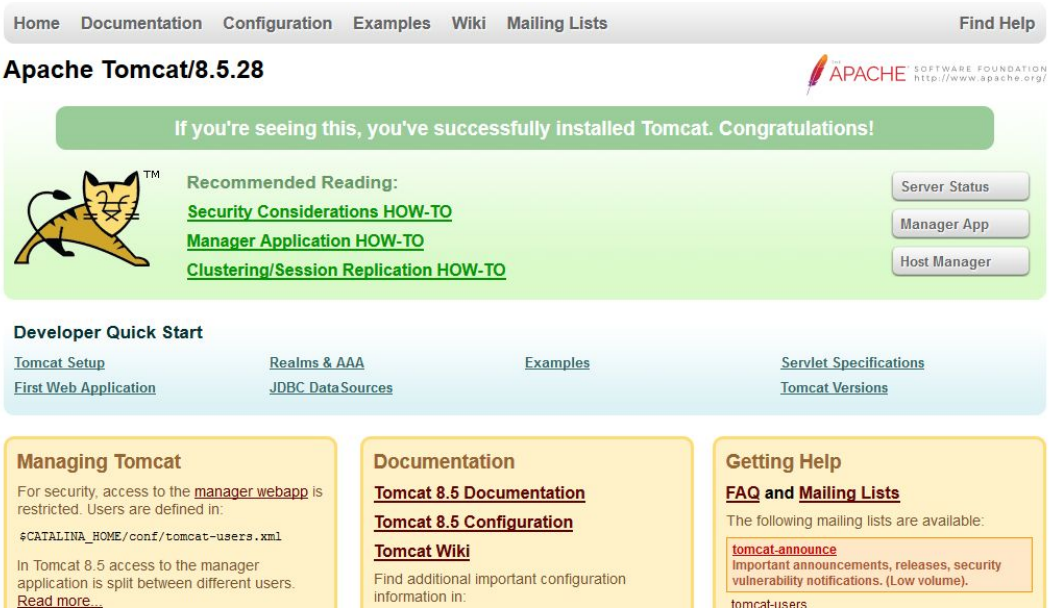
```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">

<!-- localRepository
| The path to the local repository maven will use to store artifacts.
|
-->
<localRepository>/home/ec2-user/.m2</localRepository>
```

Now, let's check whether Tomcat has successfully installed.

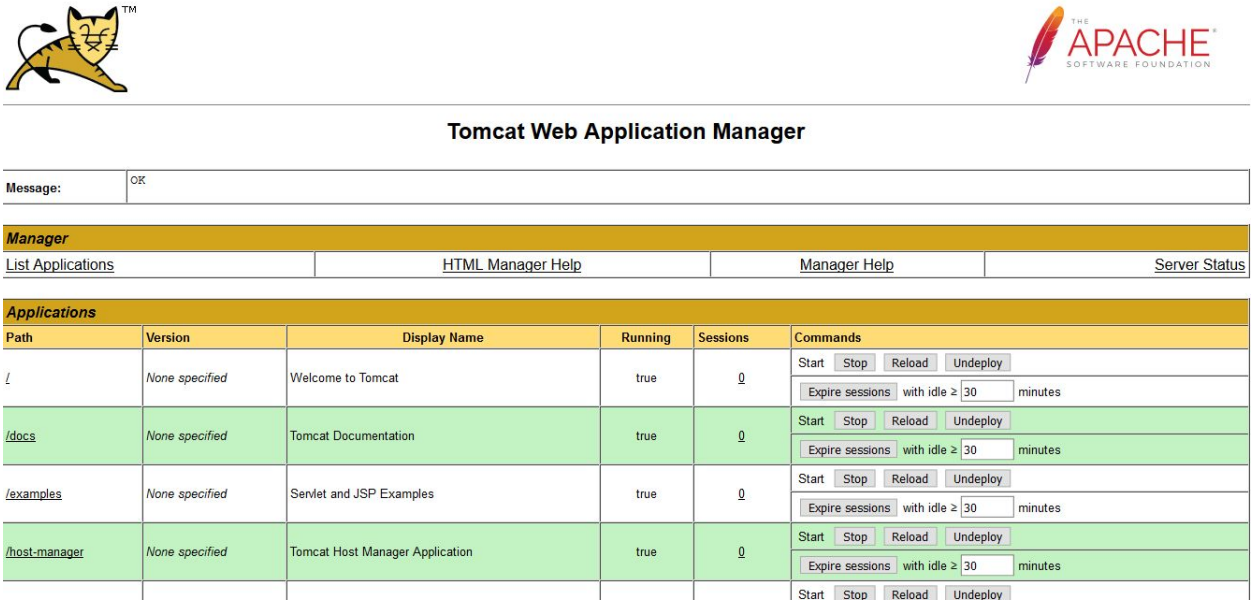
Start Tomcat by navigating to CATALINA_HOME/bin and execute the following:
./startup.sh

Verify the server installation by pointing your browser to: http://<EC2 public DNS or public IP>:8080
(Where DNS is the DNS of your EC2 instance.)

You should see the following Tomcat homepage :



Use the "manager-gui" credentials you set while configuring Tomcat to log in to the Manager App (see the button on the right-hand side of the homepage). You should see something like this:



Tomcat Web Application Manager

Message:	OK

Manager			
List Applications	HTML Manager Help	Manager Help	Server Status

Applications					
Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
					Start Stop Reload Undeploy

This page shows every deployed app in your web container, and gives you some GUI controls for interacting with them. You can also manually add a .war from here, but there's a more elegant way to do that (see step 9).

If you're not sure whether an app is deployed, check the Manager App and see if everything is where you expect it to be.

To stop Tomcat, go to CATALINA_HOME/bin and execute the following:
./shutdown.sh

9. Deploying Jenkins

Transfer jenkins.war to the /tmp folder of the EC2 instance using the same process as for Java, Maven, and Tomcat. Move jenkins.war to CATALINA_HOME/webapps with sudo mve.

The deployment scanner will automatically detect and unpack the war and deploy it within seconds.

Verify the Jenkins installation by pointing your browser to: http://<EC2 public DNS or public IP>:8080/jenkins

The Jenkins splash page shown below should be visible.

Unlock Jenkins by supplying the initial administrator password from the hidden file location shown on the splash page: /home/ec2-user/.jenkins/secrets/initialAdminPassword

On the next page choose the Install Suggested Plugins option, and create an admin user on the following page. (REMEMBER THESE CREDENTIALS!)

Now start Jenkins and make sure you can see the dashboard.

Part 2: Configuring Jenkins

1. Installing plugins

From your Jenkins dashboard, go to “Manage Jenkins” >> “Manage Plugins” and install the GitHub, Maven, Slack Notification, and Deploy to Container plugins.

2. Configure global tools

From your Jenkins dashboard, go to “Manage Jenkins” >> “Global Tool Configuration”.

For “Maven Configuration”, specify “Settings file in filesystem” as the “default settings provider” and give the file path to the /conf/settings.xml in your Maven installation (should be /usr/apache/apache-maven-3.5.2/conf/settings.xml).

For “JDK”, specify “java-8” as the name and “/usr/java/latest” as JAVA_HOME.

For “Maven,” specify “maven-352” as the name and “/usr/apache/apache-maven-3.5.2” as MAVEN_HOME.

3. Configure Git

IN GITHUB:

In your repository, go to "Settings" and choose "Integrations and Services"
Add the Jenkins (GitHub plugin) service, using http://<your public DNS>:8080/jenkins/github-webhook/ as the Jenkins hook URL.
Make sure "active" is checked and install the service.

Click on your profile icon and go to, "settings" in the dropdown.
On the left side of the screen, select "Personal access tokens" from "Developer settings"
Choose to generate a new user access token, and give it a description (jenkins-1 or something), and assign it repo scope.
Click "generate token" and copy the alphanumeric string which is generated.

IN JENKINS:

Go to "Manage Jenkins" and choose "Global Tool configuration".
Make sure that under "Git" the "path the Git executable" is just "git".
Hit "Apply".

Go back to "Manage Jenkins" and choose "Configure System". Under "Github", choose "Add GitHub Server".
the API URL should be, "https://api.github.com". In "Credentials", choose "Add/Jenkins" and choose "Kind" as "secret text".
Put the generated access token from GitHub into the "secret" field and give it some kind of descriptive ID. Then choose "Add".
Now, when you click "Test connection", you should see a message saying, "Credentials verified for user <your github name>... etc"
Hit "Apply".

In your **project setup** (not the global config), make sure that you choose "Git" under "Source Code Management".
Specify the repository URL and which branch you want to have deployed on your server.
Choose "GitHub hook trigger for GITScm polling" under "Build Triggers".

4. Configure Slack notifications

In the #jenkins channel in our organization, click the settings icon on the top panel and choose, “add an app.” This will bring you to your browser. Search for the “Jenkins CI” app and install it. Copy the integration token which will be generated.

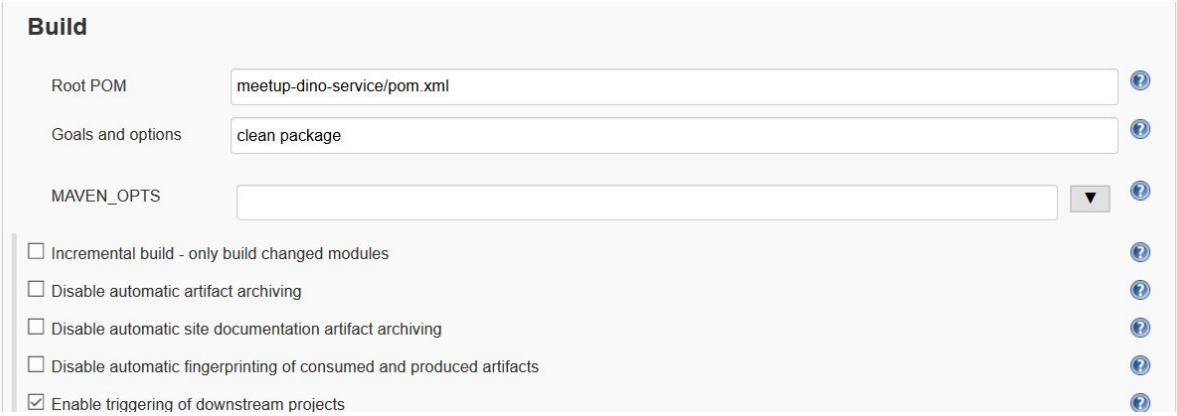
Under “Global Slack Notifier Settings,” add “jan22java” (or whatever Slack team you’re trying to use) as the Team Subdomain and paste your integration token. Test your connection, and “success” should be displayed below the form.

5. Create a project

Choose “new item”, give the item a name matching your Maven project in your repo, and choose “Maven Project” as the item type.

Select “GitHub hook trigger for GitSCM polling” for the build trigger.

In “Build”:
-add the location of your project’s POM file under “Build/Root POM”
-add “clean package” in “Goals and options”
-add your Maven version as specified in global tool settings
-add “provided settings.xml” for “settings file”



Add Slack notifications under “post-build actions” and go back to your project dashboard. Select “build now” to test whether it worked.

Go to “post-build actions” and choose, “add a post-build action.” Select “Deploy WAR/EAR to container,” and provide “**/*.war” as the “WAR/EAR files” field. Provide the context path you’d expect for local deployment (your project name prepended by a slash).

Add tomcat 8.x as a container. Provide the manager-script credentials and http://localhost:8080 as the tomcat url. Now attempt deployment by running another build.

Your app has been deployed to THIS EC2. If you wish to deploy to a different EC2, specify that url instead and make sure the credentials you provide are for THAT EC2’s Tomcat web container.