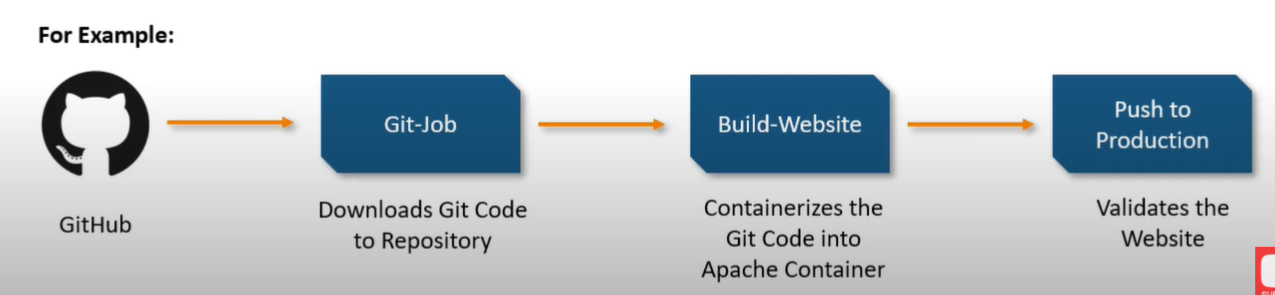
Scenario : We will push the code on **git hub** from our **Jenkins server**, as soon as that happens **gitjob** should get activated and it will make website with help of docker container. And if and only if BUID-WEBSITE stage is clear we will trigger **PUSH TO PRODUCTION** job which is final production stage.



Make three servers on AWS:

1) Jenkins : on which Jenkins will run

2) Staging : it will do “Git-Job” and “Build-Website” jobs.

3) Production : it will do “push to prod.” job

Now install Jenkins on all 3 servers as follow.

**Jenkins Installation:**

1. sudo apt-get update
2. sudo apt install openjdk-8-jdk
3. sudo apt install wget
4. wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add –
5. sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
6. sudo apt update
7. sudo apt install Jenkins
8. sudo systemctl start jenkins
9. systemctl status Jenkins

**Jenkins Server:**

1. start Jenkins on Jenkins server and install required plugins.
2. Manage Jenkins : To make nodes for the slaves

* Configure global security : Make Agent (TCP port for inbound agent) = Random
* Manage Nodes : Make two more nodes = 1) staging ,2) Production
* New Node: Remote root directory = /home/ubuntu/Jenkins

: Launch method = Launch agent by connecting it to master

* Download the .jar and .jnlp files and use filezilla to transfer them on staging and prod server respectively
* To set up Filezilla : useful to trsfer files from local host to remote servers

Host : Public IP of server

Username: ubuntu

Port : 22

Then go to edit > settings > SFTP > add .ppk file as key

|  |  |
| --- | --- |
|  |  |

1. New Item: Git-Job (freestyle project) : allow staging server to pull code from git repository

To do so : on Jenkins server type below code

* mkdir website
* cd website
* sudo nano index.html
* < html>
* <title> Hello world </title>
* <body>
* <h1>welcome viral </h1>
* </body>
* </html> (Press Ctrl + X or F2 to Exit. You will then be asked if you want to save.)
* git init : to initiate git repo in same directory
* git remote add origin “url of your git repository”
* git add .
* git commit -m “any message”
* git push origin master

Now open Jenkins Git-job

* general : Restrict where this code can run : Staging
* source code management : Git : URL : Copy URL of you repo.
* Build trigger : Github hook trigger
* Post build action : build other project : build-website (this is a second job)

**Docker Container:**

1. Now install Docker on **staging server** and **Production server** (only slave machines).
2. sudo apt-get install docker.io
3. sudo docker ps : to check docker is installed or not
4. Now create docker file on Jenkins server (master server)
5. sudo nano dockerfile
6. FROM hshar/webapp : to download an apache container
7. ADD . /var/www/html
8. Save it
9. Git add .
10. Git commit -m “added docker file”
11. git push origin master

**Now open Jenkins to create build website stage:**

* New item : build-website (freestyle project)
* general : Restrict where this code can run : Staging
* Build : Execute shell command
* sudo docker rm -f $(sudo docker ps -a -q) : to remove all existing containers
* sudo docker build /home/ubuntu/jenkins/workspace/git-job -t website : to build image
* sudo docker run -it -p 82:80 -d website : to run this image on port 8082
* Post build action : build other project : pushprod (this is a third job)
* save it.
* Build Now.

To make the first job triggered automatically when any code gets committed to git hub

* Open Github
* Settings
* Webhooks > add webhooks > URL = <http://54.174.12.231:8080/>github-webhook/ (“use Jenkins URL , don’t forget to add gihub-webhook/ at the end.”)
* Save it.

Now final thing is to push this entire thing to production,

Production

1. Open Jenkins
2. New item : pushprod (free style project)

* general : Restrick where this code can run : production
* source code management : Git : URL : Copy URL of you repo.
* Build : Execute shell command
* sudo docker rm -f $(sudo docker ps -a -q) : to remove all existing containers
* sudo docker build /home/ubuntu/jenkins/workspace/pushprod -t website : to build image
* sudo docker run -it -p 80:80 -d website : to run this image on port 8080
* save it.
* Build Now.

**Dev Pandey**

**Building Continuous Deployment on AWS with AWS CodePipeline, Jenkins and AWS Elastic Beanstalk**

Jenkins: <https://aws.amazon.com/blogs/devops/building-continuous-deployment-on-aws-with-aws-codepipeline-jenkins-and-aws-elastic-beanstalk/>

Bean stalk : <https://www.progress.com/blogs/building-sitefinity-continuous-deployment-on-aws-with-github-jenkins-and-aws-elastic-beanstalk>

apt-get update

2 apt-get upgrade

4 mkdir jenkins

5 cd jenkins/

7 wget http://www-eu.apache.org/dist/maven/maven-3/3.5.4/binaries/apache-maven-3.5.4-bin.tar.gz

8 sudo tar xzvf apache-maven-3.5.4-bin.tar.gz -C /opt/

9 rm apache-maven-3.5.4-bin.tar.gz

10 cd /opt/

12 export PATH=$PATH:/opt/apache-maven-3.5.4/bin

13 mvn archetype:generate -DgroupId=com.ebdemo -DartifactId=ebdemo -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false

If above command will not work then install jre with below command

16 apt-get install default-jre

22 cd jenkins/

23 ll

24 mvn archetype:generate -DgroupId=com.ebdemo -DartifactId=ebdemo -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false

25 ll

26 cd ebdemo/

Now we can go to Elastic Beans talk and setup below.

Set Up the Deployment Environment

This environment is where CodePipeline will deploy our code. Please note that you need to create the Elastic Beanstalk environment in a [region](http://docs.aws.amazon.com/general/latest/gr/rande.html#codepipeline_region) that supports CodePipeline.

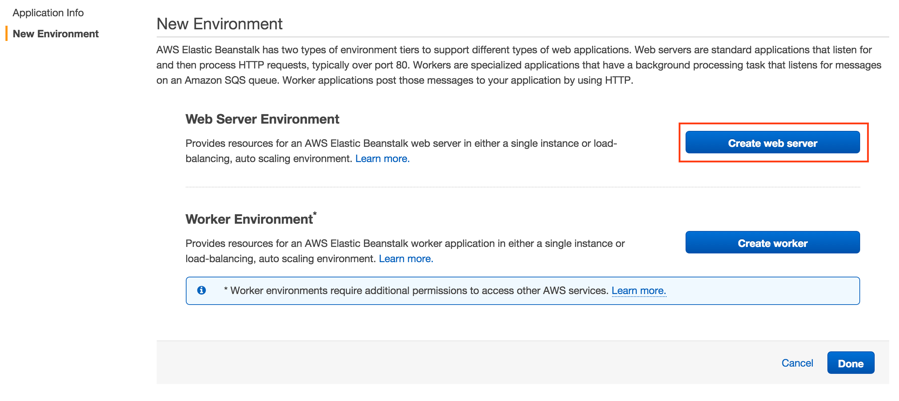
1. Create an Elastic Beanstalk application named ebdemo.

<http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.deployment.newapp.html>

2. Create an Elastic Beanstalk environment named ebdemo with the following parameters:

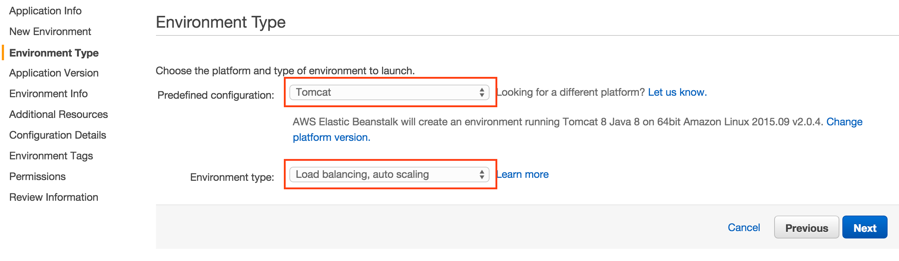
<http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.environments.html>

a.  For Web Server Environment, choose Create web server.

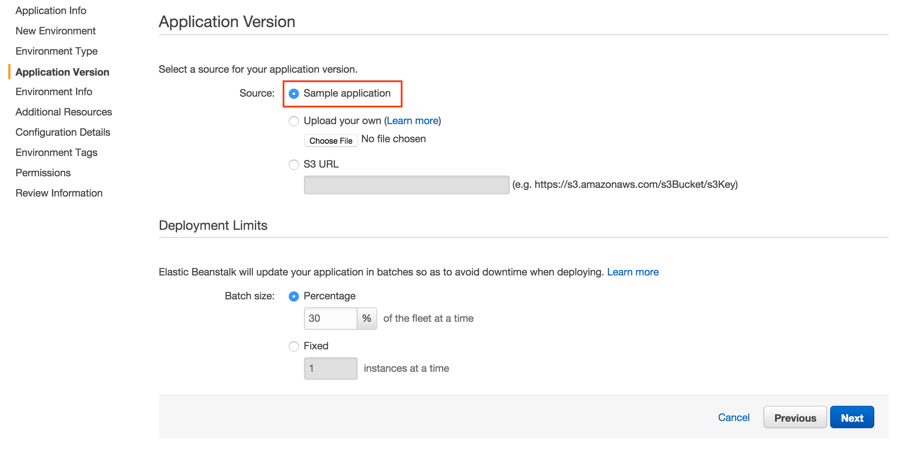


b. For Predefined configuration, choose Tomcat.

c. For Environment type, choose Load Balancing, auto scaling.

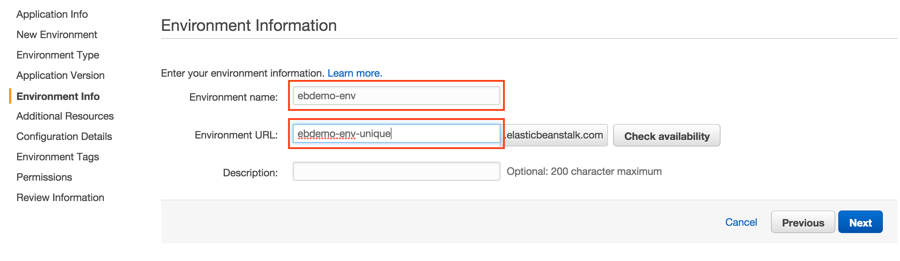


d. For Source, choose Sample application.

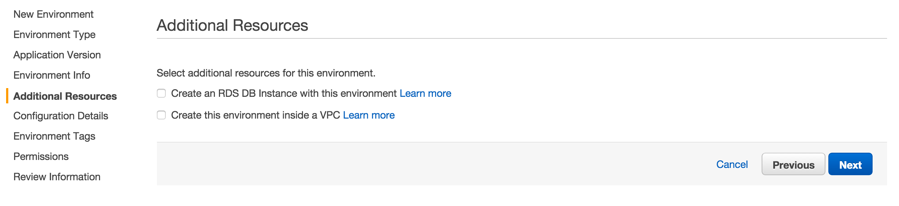


e. For Environment name, type EBDemo-env.

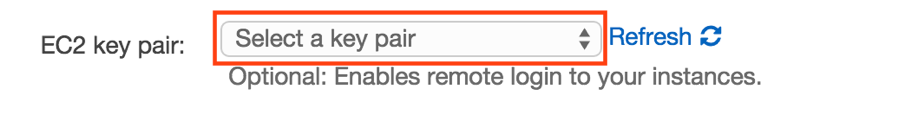
f. For Environment URL, enter a URL that is specific to your environment. For example, we’ve named ours ebdemo-env-unique.



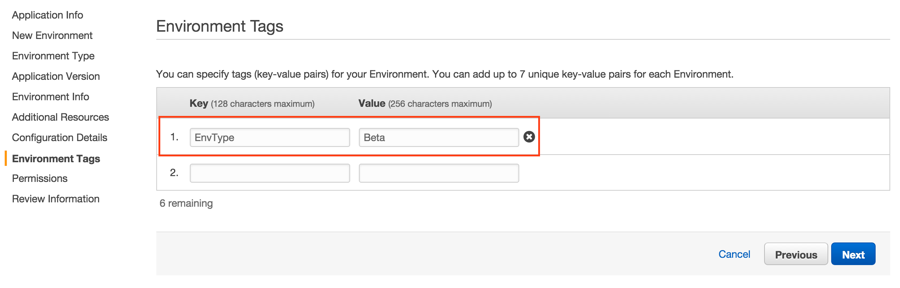
g. Leave the boxes cleared unless you want to launch into a VPC.



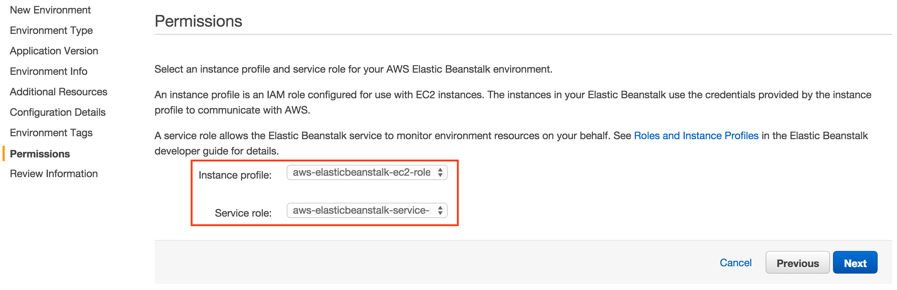
h. If you like, you can choose a key pair. You will use this key pair to connect to your Elastic Beanstalk instances by using SSH.



i.  Although tags are optional, the AWS best practice is to tag everything to provide better visibility into your resources. We’ll tag this environment with an EnvType of Beta.



j. If the instance profile and service role have not been created yet, create and then select them.



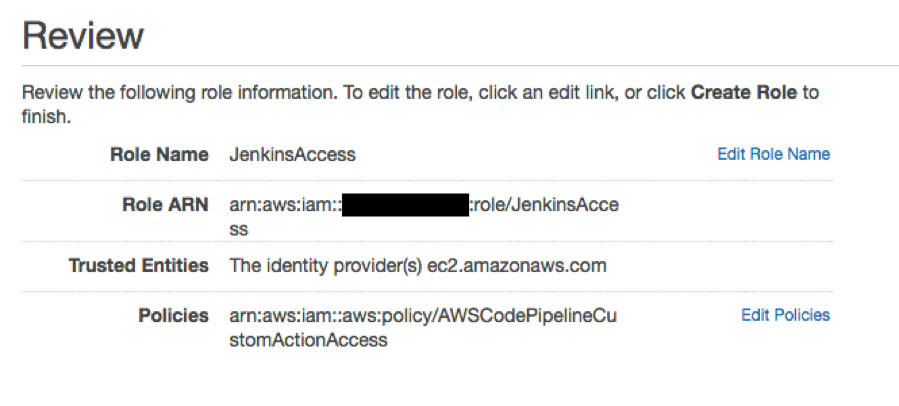
k. Click Next and Launch on the next page. Elastic Beanstalk will begin provisioning your environment

**Set up the Build Environment:**

**First, we’ll set up a new Jenkins EC2 instance. To do this, we’ll deploy Jenkins onto a standalone EC2 instance running Amazon Linux. We’ll also need to authorize ports 80 and 22 to access your Jenkins server.**

In order for the CodePipeline Jenkins plugin to access CodePipeline, you have a few options:

* Access/Secret Keys specified in the Jenkins console
* Environment Variables
* Java System Properties
* Credentials Profile File
* EC2 Instance Profile



Now go and attach the IAM role with Jenkins server..

We are good to setup a Jenkins server now on DEVlopment / Jenkins server.

Belo step will only apply when you are using different DEV server else skip this one :-

Installing Maven

In the same way as the development environment, [download](https://maven.apache.org/download.cgi) the binary tar.gz archive and [install](https://maven.apache.org/install.html) maven in your Jenkins build server.

wget http://mirror.cogentco.com/pub/apache/maven/maven-3/3.5.3/binaries/apache-maven-3.5.4-bin.tar.gz

sudo tar xzvf apache-maven-3.5.4-bin.tar.gz -C /opt/

rm apache-maven-3.5.4-bin.tar.gz

Maven is now installed at /opt/apache-maven-3.5.4/ in this example.

Good to proceed with Jenkins installation now.

Jenkins comamnds

to fix the Java issue and not startign Jenkns issue.

sudo add-apt-repository ppa:webupd8team/java

sudo apt-get update

sudo apt-get install oracle-java8-installer

installing jenikin

wget -q -O - https://jenkins-ci.org/debian/jenkins-ci.org.key | sudo apt-key add -

sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian binary/ > /etc/apt/sources.list.d/jenkins.list'

sudo apt-get update

sudo apt-get install jenkins

ps -ef | grep jenkins

http://localhost:8080

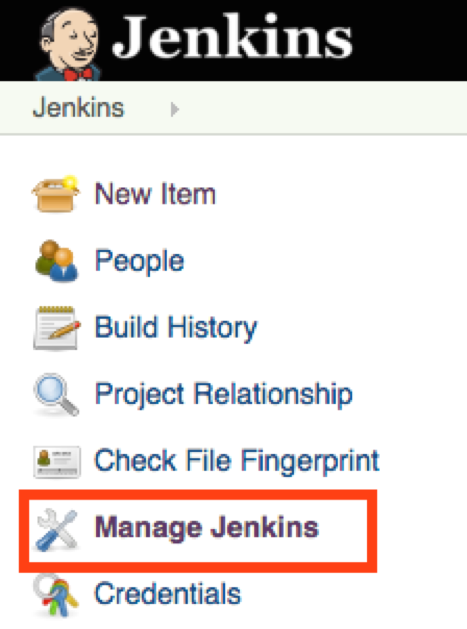
sudo cat /var/lib/jenkins/secrets/initialAdminPassword (to get admin password)

Open a browser to access the Jenkins portal. Navigate directly to the IP/DNS name of your instance. In the earlier step, we routed traffic on port 80 to redirect to port 8080. At this point, Jenkins is up and running and can be accessed on port 80. That means anyone can access it on port 80 if the security group allows it.

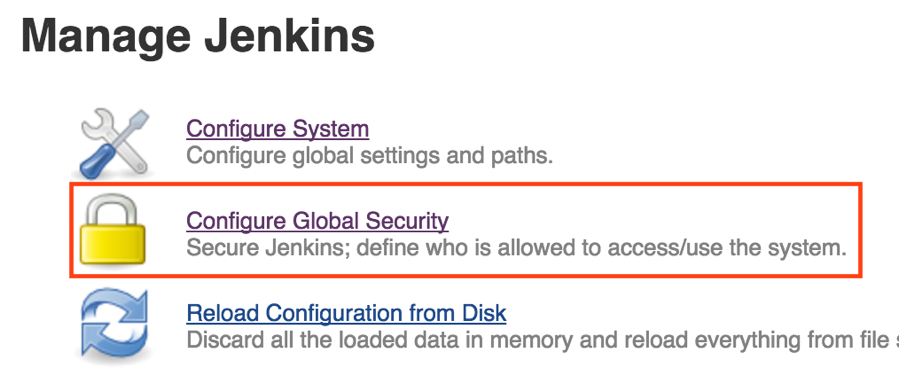
Configure Security in Jenkins

Securing Jenkins is optional, but highly recommended. We’ll set up basic security.

1. From the Jenkins landing page, choose Manage Jenkins.



2. Choose Configure Global Security.

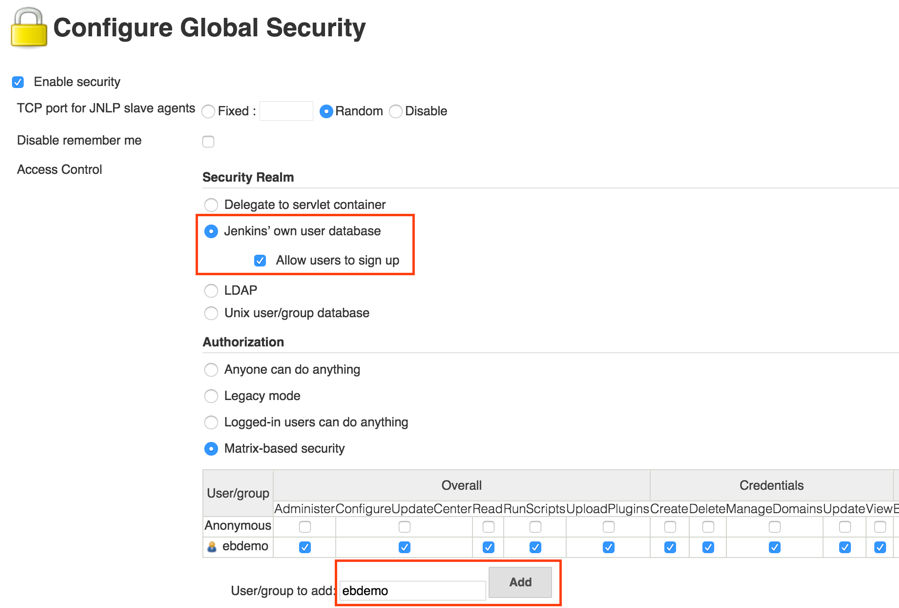


3. Check Enable Security.

4. Under Security Realm, choose Jenkin’s own user database and Allow users to sign up.

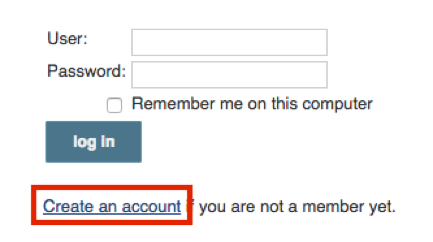
5. Under Authorization, choose Matrix-based Security.

6. To the User/group table, type ebdemo, and then choose Add. Use the  icon to give the user all permissions. This gives permissions to the account you’ll create in the next two steps.

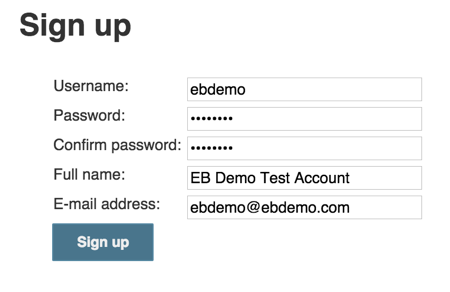


7. Choose Save.

8. Choose Create an account.



9. Create an account with a Username of ebdemo and a password of your choice.

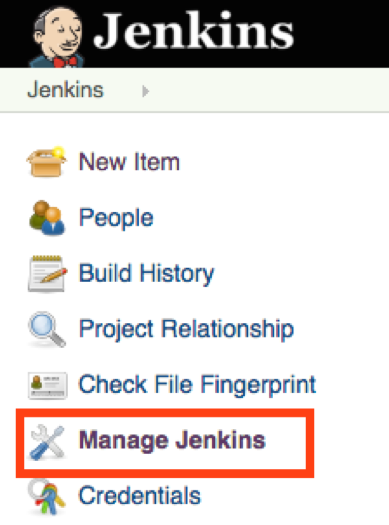


10. Sign in with your ebdemo account.

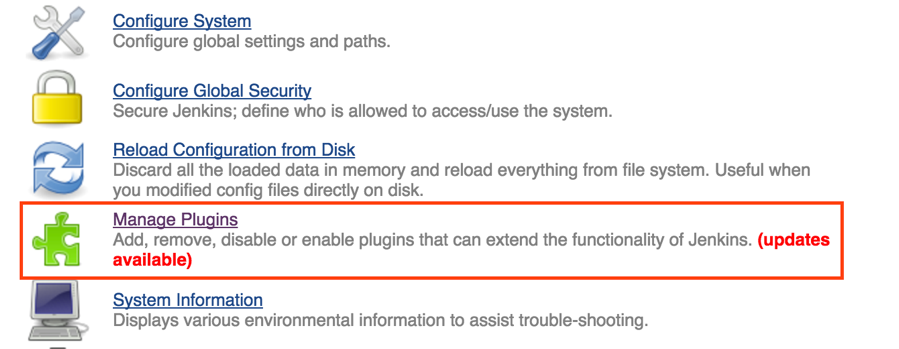
Install the AWS CodePipeline Plugin

The CodePipeline plugin is now included in the Jenkins plugin repo.

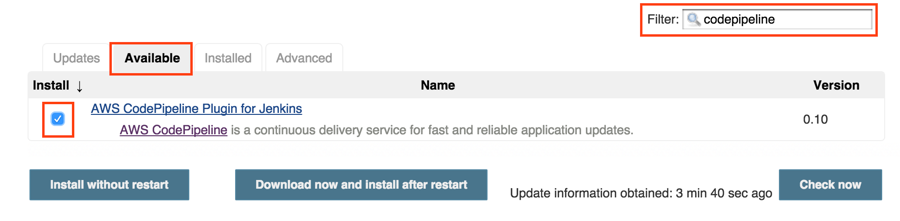
1. From the Jenkins landing page, choose Manage Jenkins.



2. Choose Manage Plugins.



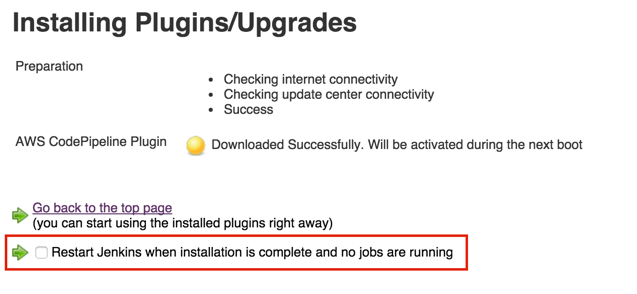
3. On the Available tab, filter for CodePipeline.



4. Choose the Download now and Install after restart button.

5. Check the Restart Jenkins when Installation is complete and no jobs are running box for Jenkins to automatically restart and install the plugin.

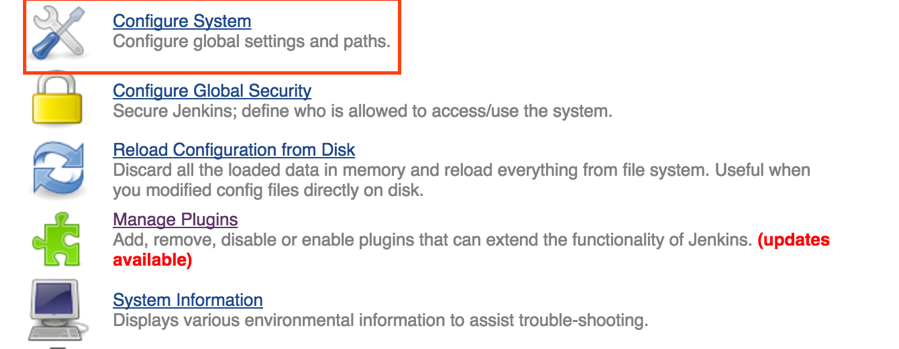
6. add one more plugin name of Maven integration. To have Maven project create.



Configure Maven in Jenkins

1. On the Jenkins landing page, choose Manage Jenkins.

2. Choose Configure System.



# 3. Go to Global Tool Configuration

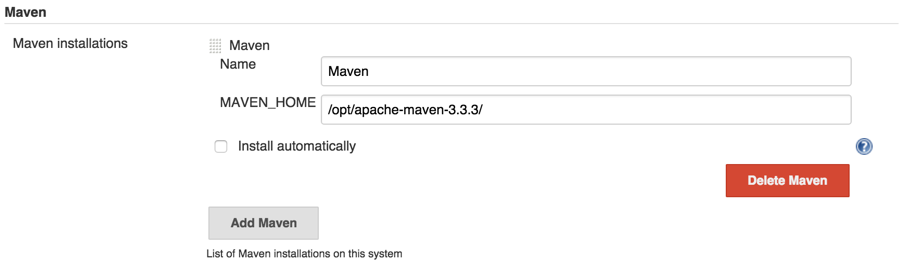
Scroll to the Maven (not Maven Configuration) section.

4. Choose the Add Maven button.



5. For Name, type Maven. Be sure the Install Automatically box is cleared.

6. Type the directory where you downloaded and extracted Maven earlier.



7. Choose Save.

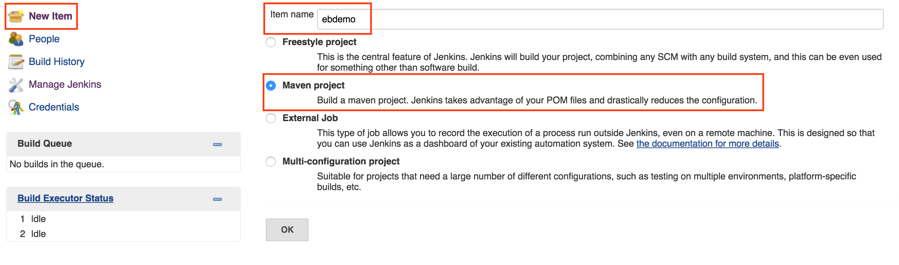
Create a Jenkins Build Job

We can now create a Jenkins build job that will poll CodePipeline for new artifacts, build them, and pass the completed build back to CodePipeline.

1. In the Jenkins console, choose New Item.

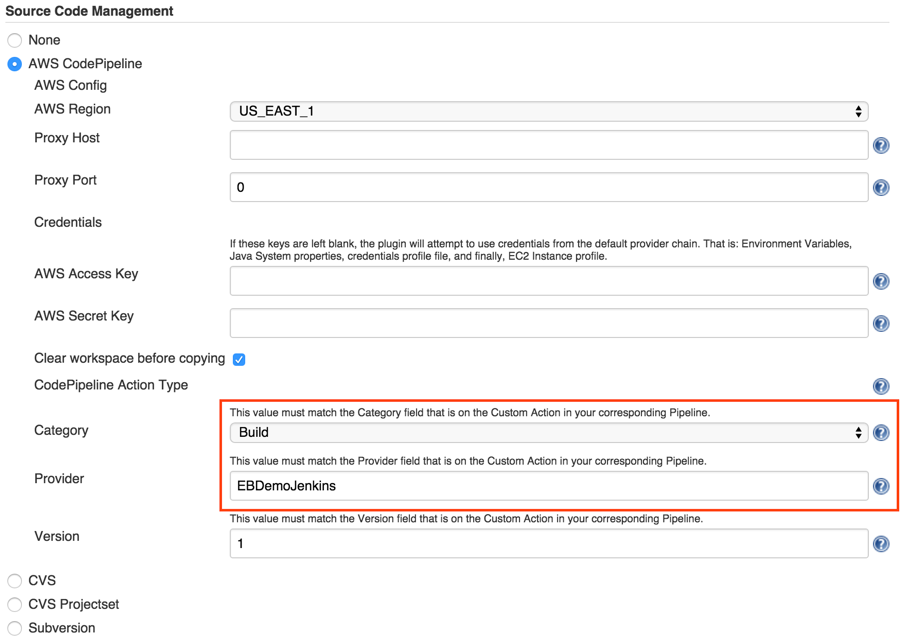
2. Choose Maven Project.

3. Type a name for the project. For the purpose of this post, we will use ebdemo.



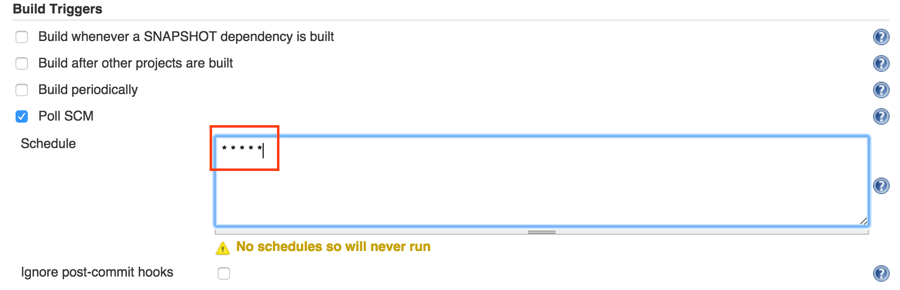
4. Choose OK.

5. On the configuration page for the job, under Source Code Management, choose AWS CodePipeline.

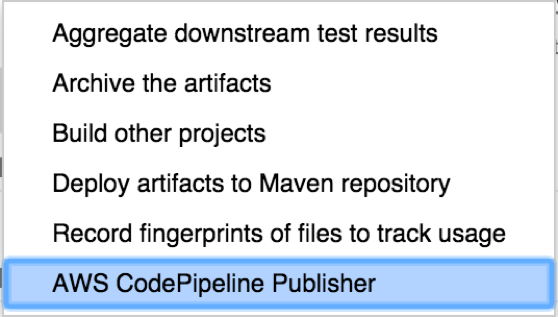


6. Under Build Triggers, select Poll SCM.

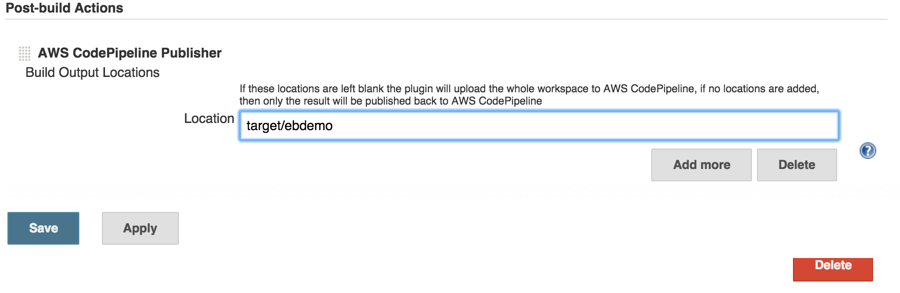
7. In the Schedule text box, type \* \* \* \* \*. This will poll CodePipeline every minute for changes.



8. In the Post-build Actions section, choose Add post-build action, and then choose AWS CodePipeline Publisher.

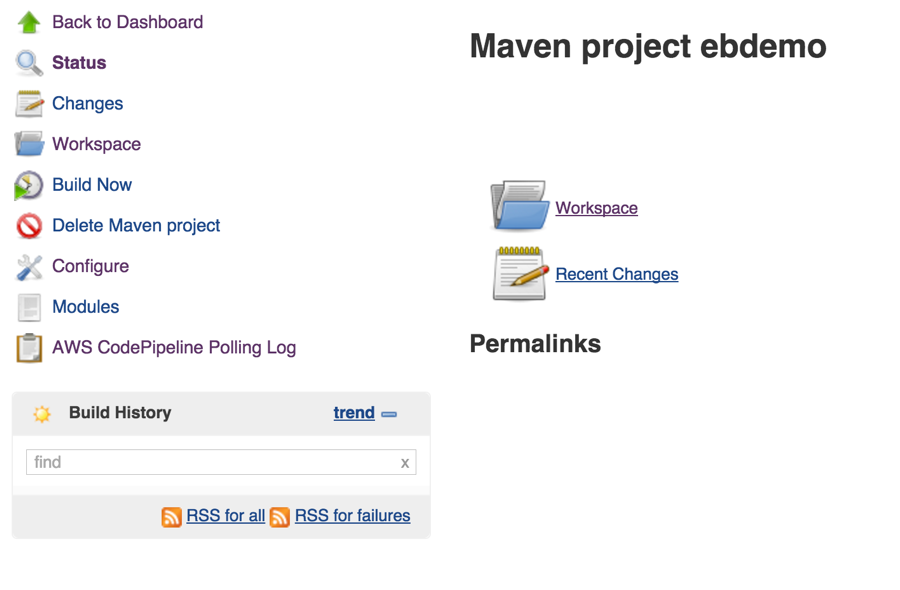


9. For Location, type the location of the built artifact. In this case, it is target/ebdemo.



10. Choose Save.

The following page should appear:

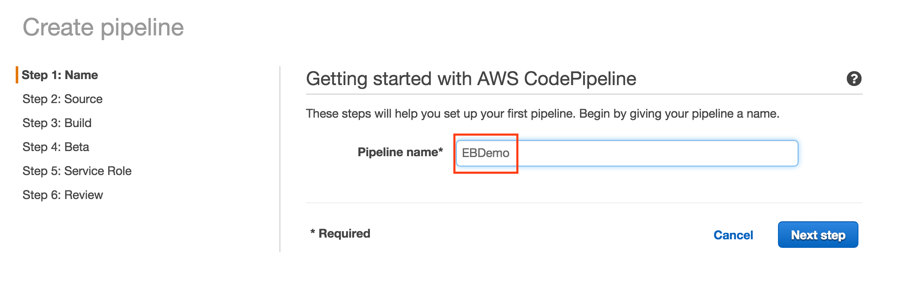


Jenkins should now be set up to listen for changes coming from CodePipeline.

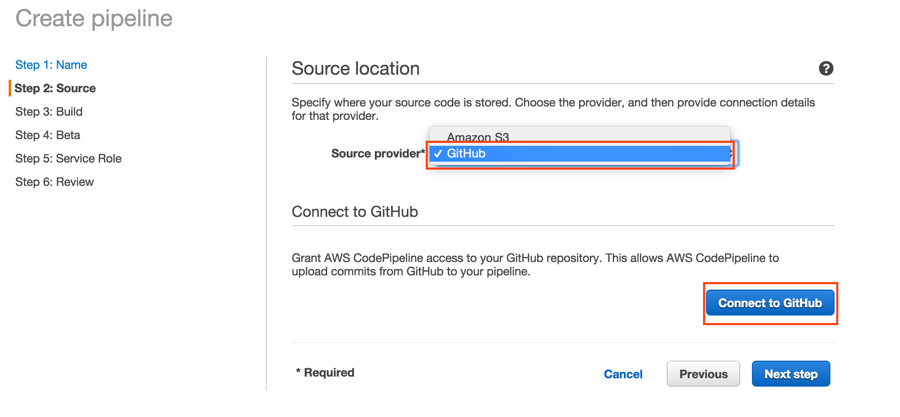
Create and Configure AWS CodePipeline

1. Sign in to the AWS Management Console and then open the CodePipeline console. Choose Get Started.

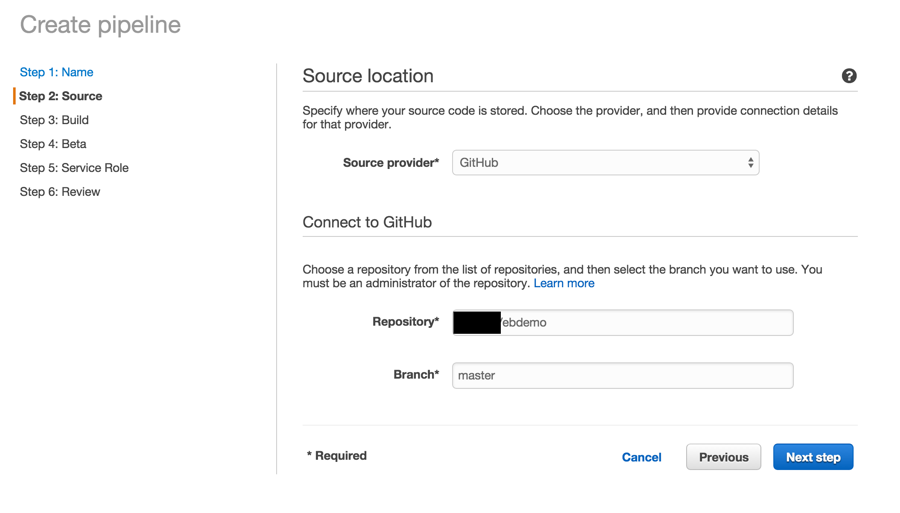
2. On the Create pipeline page, type a name your pipeline. For this post, we’ll name it EBDemo.



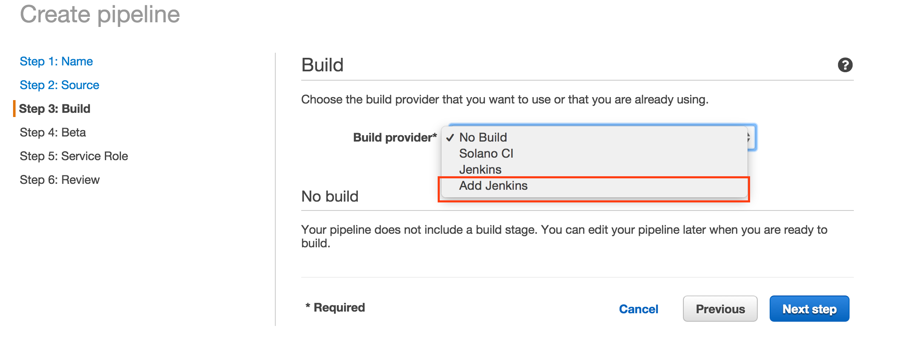
3. For Source provider, choose GitHub.



4. Choose Connect to GitHub, and then type your GitHub credentials.

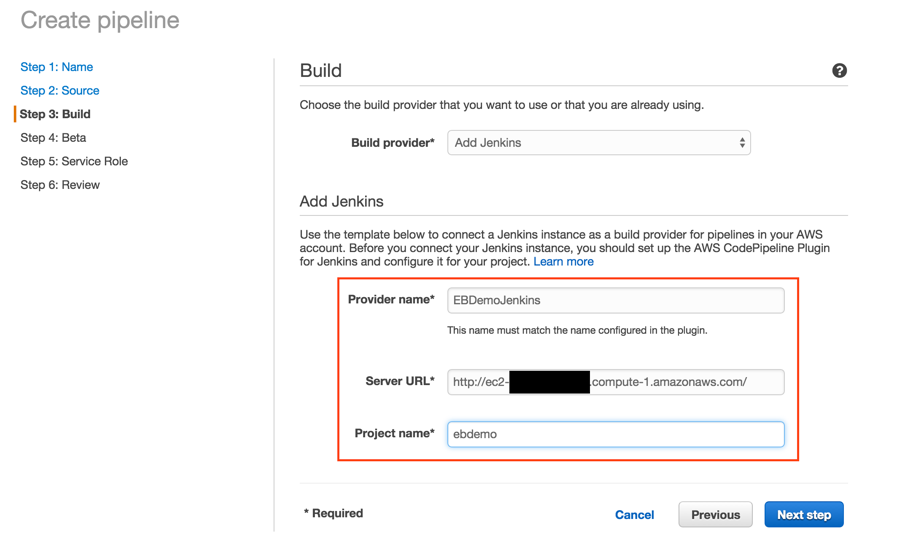


5. For Build provider, choose Add Jenkins.

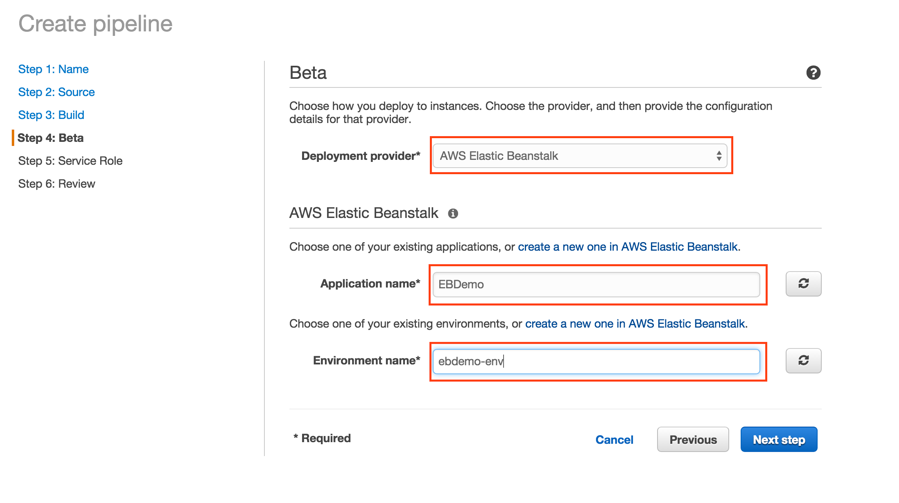


6. The provider name and project name must match the values we used to configure Jenkins:

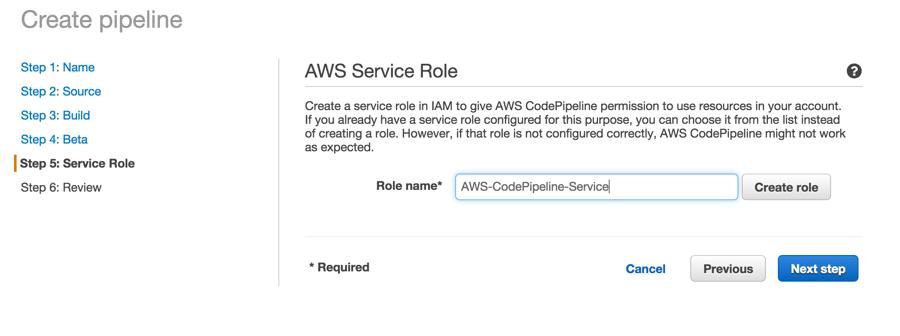
* Provider Name: EBDemoJenkins
* Server URL: <The URL of your Jenkins EC2 instance>
* Project Name: ebdemo



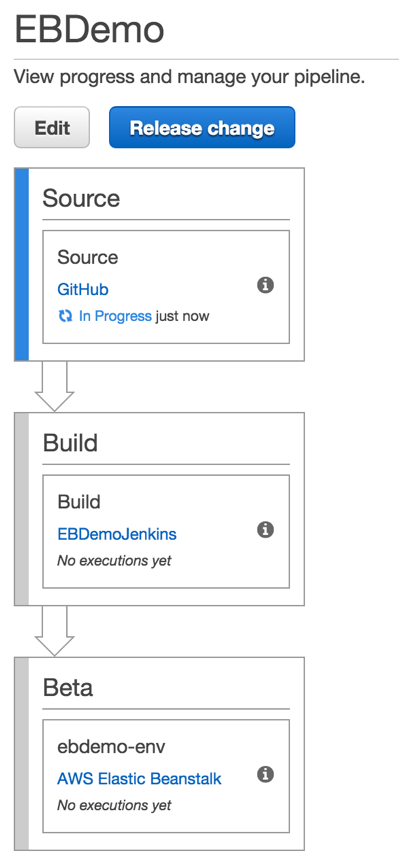
7. For Deployment provider, choose AWS Elastic Beanstalk. For Application name, type EBDemo. For Environment name, type ebdemo-env.



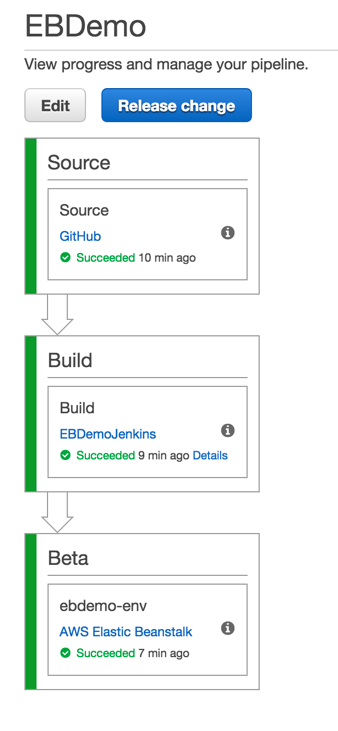
8. Under AWS Service Role, choose the AWS CodePipeline service role or create one.

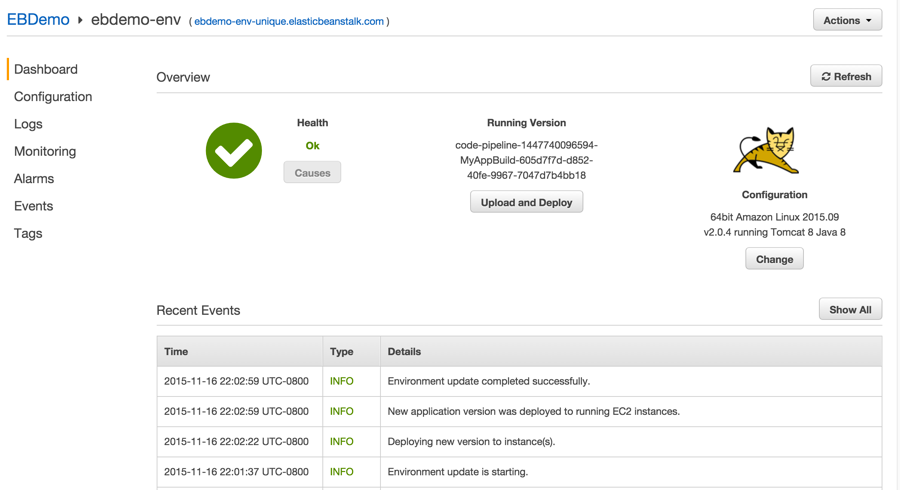


9. Review your pipeline, and then choose Go. You should see this:



In about a minute, CodePipeline will take your most recent revision from GitHub, pass it to your Jenkins EC2 instance, and then send the built code to Elastic Beanstalk. Once you see that CodePipeline has completed execution as pictured below, go to the Elastic Beanstalk console to take a look at your environment.





You now have a pipeline that will automatically build and deploy to Elastic Beanstalk each time you check in code. Happy coding!