

DevOps Capstone Project 1

CAPSTONE PROJECT

You have been Hired Sr. Devops Engineer in Abode Software. They want to implement Devops Lifecycle in their company. You have been asked to implement this lifecycle as fast as possible. Abode Softwares is a product-based company, their product is available on this GitHub link.

<https://github.com/hshar/website.git>

Following are the specifications of the lifecycle:

1. Git Workflow has to be implemented
2. Code Build should automatically be triggered once commit is made to master branch or develop branch.

If commit is made to master branch, test and push to prod

If commit is made to develop branch, just test the product, do not push to prod

3. The Code should be containerized with the help of a Dockerfile. The Dockerfile should be built every time there is a push to Git-Hub. Use the following pre-built container for your application:

hshar/webapp

The code should reside in '/var/www/html'

4. Once the website is built, you have to design a test-case, which will basically check if the website can be opened or not. If yes, the test should pass. This test has to run in headless mode, on the test server.

5. The above tasks should be defined in a Jenkins Pipeline, with the following Jobs

Job 1 - Building Website

Job 2 - Testing Website

Job 3 - Push to Production

6. Since you are setting up the server for the first time, ensure the following file exists on both Test and Prod server in /home/ubuntu/config-management/status.txt. This file will be used by a third-party tool. This should basically have the info whether apache is installed on the system or not

The content of this file, should be based on whether git is installed or not.

If apache is installed => Apache is Installed on this System"

If apache is not installed => "Apache is not installed on this System"

7. Create a Monitoring Service for the website on the Production server

Architectural Advice:

Create 3 servers on AWS "t2.micro"

Server 1 - should have Jenkins Master, Puppet Master and Nagios Installed

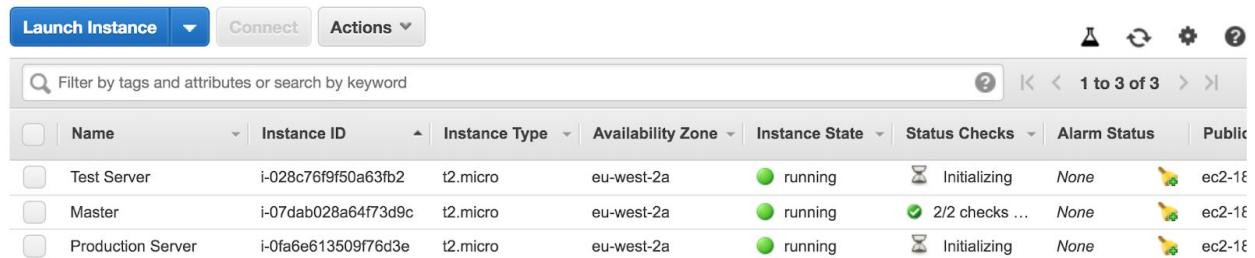
Server 2 - Testing Server, Jenkins Slave

Server 3 - Prod Server, Jenkins Slave

Solution:

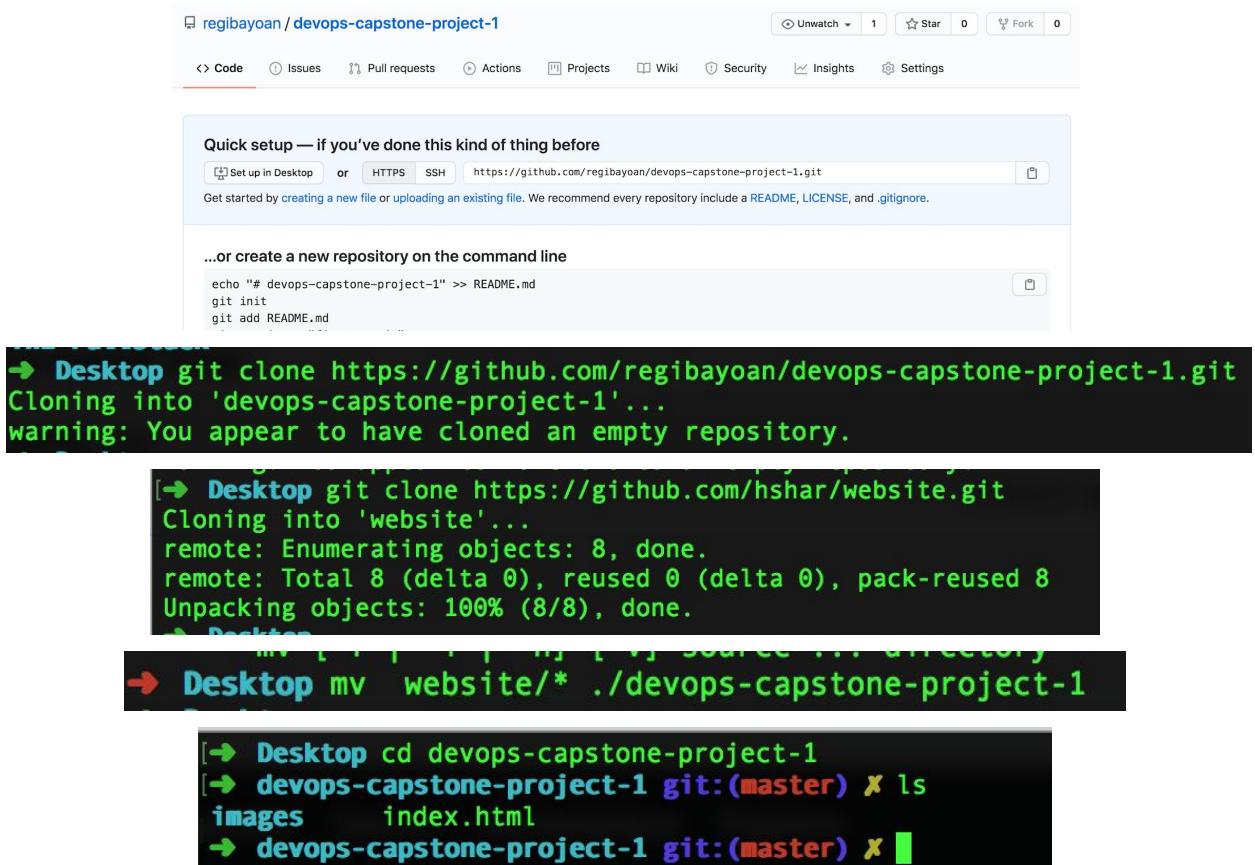
Step 1: Setup AWS architecture

- 3 servers => "t2.micro"
- Server 1 => Jenkins Master, Puppet Master, Nagios
- Server 2 => Testing Server (jenkins-slave-1)
- Server 3 => Production Server (jenkins-slave-2)



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public
Test Server	i-028c76f9f50a63fb2	t2.micro	eu-west-2a	running	Initializing	None	ec2-1€
Master	i-07dab028a64f73d9c	t2.micro	eu-west-2a	running	2/2 checks ...	None	ec2-1€
Production Server	i-0fa6e613509f76d3e	t2.micro	eu-west-2a	running	Initializing	None	ec2-1€

Step 2: Create a new GitHub repository and copy the files from the product repository



Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/regibayoan/devops-capstone-project-1.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# devops-capstone-project-1" >> README.md
git init
git add README.md
```

→ Desktop git clone https://github.com/regibayoan/devops-capstone-project-1.git
Cloning into 'devops-capstone-project-1'...
warning: You appear to have cloned an empty repository.

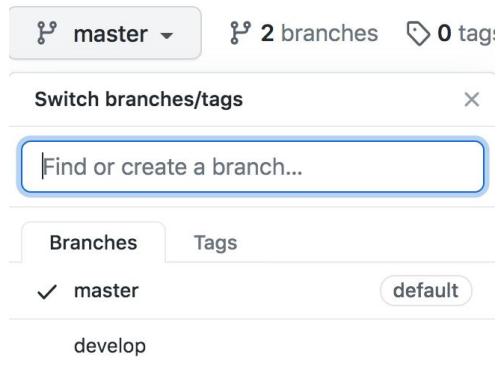
→ Desktop git clone https://github.com/hshar/website.git
Cloning into 'website'...
remote: Enumerating objects: 8, done.
remote: Total 8 (delta 0), reused 0 (delta 0), pack-reused 8
Unpacking objects: 100% (8/8), done.

→ Desktop mv website/* ./devops-capstone-project-1

→ Desktop cd devops-capstone-project-1
→ devops-capstone-project-1 git:(master) ✘ ls
images index.html
→ devops-capstone-project-1 git:(master) ✘

Step 3: Create a new ‘develop’ branch

```
[→ devops-capstone-project-1 git:(master) ✘ git checkout -b develop
Switched to a new branch 'develop'
```



Step 4: Install Jenkins on Master server

Jenkins installation Guide: <https://linuxize.com/post/how-to-install-jenkins-on-ubuntu-18-04/>

```
[ubuntu@Master:~$ systemctl status jenkins
● jenkins.service - LSB: Start Jenkins at boot time
  Loaded: loaded (/etc/init.d/jenkins; generated)
  Active: active (exited) since Thu 2020-07-30 15:25:47 UTC; 30s ago
    Docs: man:systemd-sysv-generator(8)
   Tasks: 0 (limit: 1121)
  CGroup: /system.slice/jenkins.service
```

Step 5: Setup Jenkins Master-Slave cluster

Setup guide: <https://github.com/reqibayoan/jenkins-ci-cd-pipeline>

The setup should look like this:

Step 6: Create a pipeline for when committing to **develop** branch

- Create a **Dockerfile** to build the product. Use the pre-built image, **hshar/webapp** as a base image. This image will run an Ubuntu container with Apache installed. Then add **index.html** to **/var/www/html** inside the container.
- Push to GitHub

```
GNU nano 2.0.6           File: Dockerfile           Modified

FROM hshar/webapp
ADD ./index.html /var/www/html

[→ devops-capstone-project-1 git:(develop) ✘ git status
On branch develop
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Dockerfile

nothing added to commit but untracked files present (use "git add" to track)
[→ devops-capstone-project-1 git:(develop) ✘ git add .
[→ devops-capstone-project-1 git:(develop) ✘ git commit -m "Adding Dockerfile"
[develop c1020c0] Adding Dockerfile
  1 file changed, 3 insertions(+)
  create mode 100644 Dockerfile
[→ devops-capstone-project-1 git:(develop) ✘ git push origin develop]
```

- Install Docker in Slave1

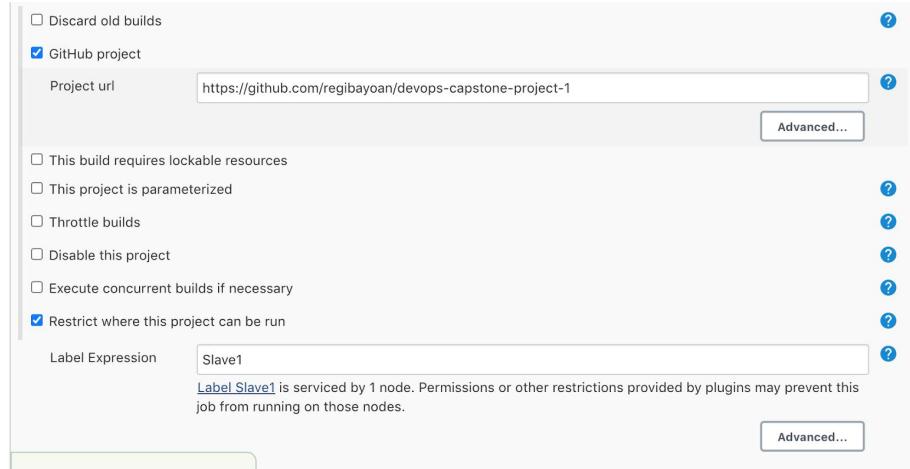
Commands: sudo apt update -> sudo apt-get remove docker docker-engine docker.io -> sudo apt install docker.io -> sudo systemctl start docker -> sudo systemctl enable docker -> sudo systemctl status docker

```
[ubuntu@Slave1:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND       CREATED
STATUS              PORTS
ubuntu@Slave1:~$
```

- Create a **Build** job in Jenkins



- Set the project repository
- Restrict project to **Slave1**



- Restrict project to build to **develop** branch

Source Code Management

Repositories

- None
- Git
 - Repository URL: https://github.com/regibayoan/devops-capstone-project-1
 - Credentials: - none -
 - Advanced...
 - Add Repository

Branches to build

- Branch Specifier (blank for 'any')
- Add Branch

Repository browser

- Build the project for now and confirm if the project files are successfully transferred to Slave1

Build History

trend —

find X

#1 Jul 30, 2020 4:20 PM

```
[ubuntu@Slave1:~/workspace/Build$ ls
Dockerfile  images  index.html
ubuntu@Slave1:~/workspace/Build$
```

- Now, try to deploy the website
- Go back to **Build job** in Jenkins -> **Configure** -> **Build** -> **Execute Shell** -> Add commands for deploying the website

Build

Execute shell

```
Command
sudo docker rm -f $(sudo docker ps -a -q)
sudo docker build /home/ubuntu/workspace/Build -t build
sudo docker run -it -p 80:80 -d build
```

See [the list of available environment variables](#)

- Run an arbitrary container in Slave1

```
[ubuntu@Slave1:~/workspace/Build$ sudo docker run -it -d ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
3ff22d22a855: Pull complete]
```

- Build project in Jenkins
- Check that the new container is successfully built
- Check in <Slave1-IPAdress>:80 to see if website is deployed

Build History

trend

find

#2 Jul 30, 2020 4:30 PM

```
[ubuntu@Slave1:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              NAMES
174a61f382aa        build              "/bin/sh -c 'apache..."   24 seconds ago    Up 23 seconds   vibrant_poincare]
```

Not Secure | 18.130.76.222

Hello world Develop!

Step 7: Next is to design a test case, which checks if the website can be opened or not

- Go to web page and **Inspect** elements

- The test case is for checking whether the title “**Intellipaat**” is present on the webpage. This should confirm that the website is up and running

```
<html>
  <head>
    <title> Intellipaat </title> == $0
  </head>
```

- Go to Eclipse and design the test case
- Make the test headless by adding “**--headless**” as an argument to ChromeDriver
- Verify the test is working locally

Code:

```
String baseUrl = "http://18.130.76.222";

// "/home/ubuntu/chromedriver"
WebDriver driver;
@BeforeTest
private void launchBrowser() {
    System.setProperty("webdriver.chrome.driver", "/home/ubuntu/chromedriver");
    ChromeOptions options = new ChromeOptions();
    options.addArguments("--headless");
    driver = new ChromeDriver(options);
    driver.get(baseUrl);
}

@Test
Run | Debug
private void verifyHomePageTitle() {
    String expectedTitle = "Intellipaat";
    String actualTitle = driver.getTitle();
    Assert.assertEquals(expectedTitle, actualTitle);
    System.out.println("Title is " + actualTitle);
    System.out.println("Website can be opened");
}
```

```

INFO: Detected dialect: W3C
Title is Intellipaat
Website can be opened
Test Complete

=====
Command line suite
Total tests run: 1, Failures: 0, Skips: 0
=====
```

- Create a runnable jar file
- Change permissions to be executable
- Send the jar file to Slave1 through Filezilla



```

[→ Desktop chmod +x test.jar
[→ Desktop ls -al test.jar
-rwxr-xr-x 1 RegiBayoan staff 11318054 30 Jul 18:02 test.jar
```

Server/Local file	Direction	Remote file	Size	Priority	Status
sftp://ubuntu@18.130.7...					
/Users/RegiBayoan/D...	-->	/home/ubuntu/test.jar	11,318,054	Normal	Transferring

00:00:06 elapsed 00:00:06 left 56.0% 6,340,608 bytes (1.2 MiB/s)

- Install chromedriver in Slave1



[chromedriver_linux64.zip](#) 2020-05-28 21:05:07 5.06MB beffb1bca07d8f4fd23213b292ef963b

```

ubuntu@Slave1:~$ wget https://chromedriver.storage.googleapis.com/84.0.4147.30/chromedriver_linux64.zip
```

```
[ubuntu@Slave1:~$ unzip chromedriver_linux64.zip
Archive: chromedriver_linux64.zip
  inflating: chromedriver
ubuntu@Slave1:~$
```

- Install **Google Chrome** browser in **Slave1** using this guide
<https://linuxize.com/post/how-to-install-google-chrome-web-browser-on-ubuntu-18-04/>

- Run the test using **java -jar test.jar**

```
ubuntu@Slave1:~$ java -jar test.jar
Starting ChromeDriver 84.0.4147.30 (48b3e868b4cc0aa7e8149519690b6f6949e110a8-refs/branch-heads/4147@{#310}) on port 25697
Only local connections are allowed.
Please see https://chromedriver.chromium.org/security-considerations for suggestions on keeping ChromeDriver safe.
ChromeDriver was started successfully.
Jul 30, 2020 5:11:59 PM org.openqa.selenium.remote.ProtocolHandshake createSession
[INFO: Detected dialect: W3C
Title is Intellipaat
Website can be opened
Test Complete

=====
Command line suite
Total tests run: 1, Failures: 0, Skips: 0
=====
```

- Verify the test is actually valid by disabling the website's container, which will take down the website.

```
[ubuntu@Slave1:~$ sudo docker container stop 174a61f382aa
174a61f382aa
```

C ① 18.130.76.222



This site can't be reached

18.130.76.222 refused to connect.

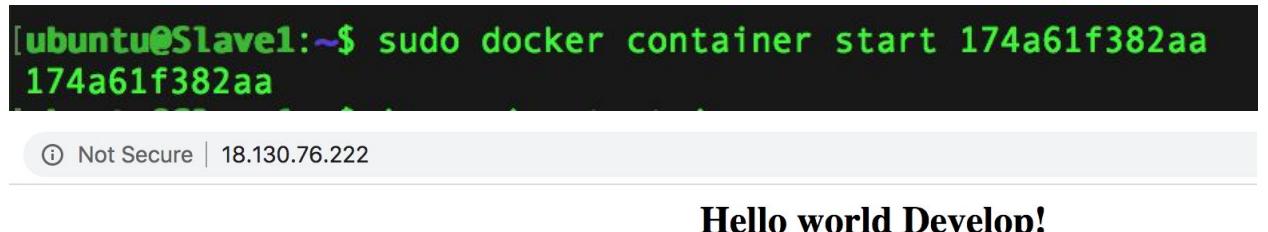
Try:

- Run the test again. The test should fail since the website will not be up anymore

```
[ubuntu@Slave1:~$ java -jar test.jar
Starting ChromeDriver 84.0.4147.30 (48b3e868b4cc6
s/4147@{#310}) on port 4582
Only local connections are allowed.
Please see https://chromedriver.chromium.org/secu
ng ChromeDriver safe.
ChromeDriver was started successfully.
Jul 30, 2020 5:18:52 PM org.openqa.selenium.remote
INFO: Detected dialect: W3C
Test Complete

=====
Command line suite
Total tests run: 1, Failures: 1, Skips: 0
=====
```

- Now restart the container. The website should be back and running



- Run the test again

```
[ubuntu@Slave1:~$ java -jar test.jar
Starting ChromeDriver 84.0.4147.30 (48b3e868b4cc6
s/4147@{#310}) on port 14553
Only local connections are allowed.
Please see https://chromedriver.chromium.org/secu
ng ChromeDriver safe.
ChromeDriver was started successfully.
Jul 30, 2020 5:19:39 PM org.openqa.selenium.remote
INFO: Detected dialect: W3C
Title is Intellipaat
Website can be opened
Test Complete

=====
Command line suite
Total tests run: 1, Failures: 0, Skips: 0
=====
```

- This verifies that the test is valid

Step 8: In Jenkins, create a **Test** job for testing the website

Enter an item name

Test
» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

- Restrict to Slave1

Restrict where this project can be run

Label Expression Slave1
[Label Slave1](#) is serviced by 1 node. Permissions or other restrictions provided by plugins may prevent this

- Restrict to **develop** branch

Branches to build

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

Add Branch

- Add commands in **Build -> Save**

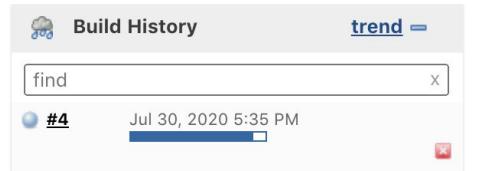
Build

Execute shell

Command `cd /home/ubuntu
java -jar test.jar`

See [the list of available environment variables](#)

- Build Test



```
INFO: Detected dialect: W3C
Title is Intellipaat
Website can be opened
Test Complete

=====
Command line suite
Total tests run: 1, Failures: 0, Skips: 0
=====

Finished: SUCCESS
```

- Problem encountered is that even when the test is failed, the Jenkins job is still “success”
- We want the job to fail as well if the test fails
- Modify the test code in Eclipse by adding a try-catch block. The catch block will return a value of 1 if the test fails. As a result, the Jenkins job will also fail. This is what we want so the pipeline will not continue to the next job if the **Test** job fails (**Ex: In production**)

```
@Test
Run | Debug
private void verifyHomePageTitle() {
    String expectedTitle = "Intellipaat";
    String actualTitle = driver.getTitle();
    try {
        Assert.assertEquals(expectedTitle, actualTitle);
    } catch(AssertionError e){
        System.out.println("Website not present");
        System.exit(1);
    }
    System.out.println("Title is " + actualTitle);
    System.out.println("Website can be opened");
}
```

- Verify this works by exporting a new jar file -> **testModified.jar** -> send to Slave 1 through Filezilla -> Verify if transferred in Slave1 -> Change permissions to be executable

```
[ubuntu@Slave1:~$ chmod +x testModified.jar
```

- Modify the **Build** section in **Test** job



- Build **Test** Job when website is up

```
+ java -jar testModified.jar
Starting ChromeDriver 84.0.4147.30 (48b3e868b4cc0aa7e8149519690b6f6949e110a8-refs/branch-heads/4147@{#310}) on port 1103
Only local connections are allowed.
Please see https://chromedriver.chromium.org/security-considerations for suggestions on keeping ChromeDriver safe.
ChromeDriver was started successfully.
Jul 30, 2020 6:35:25 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: W3C
Title is Intellipaat
Website can be opened

=====
Command line suite
Total tests run: 1, Failures: 0, Skips: 0
=====

Finished: SUCCESS
```

- Now stop the container again so the website is down. The Jenkins build should fail

```
+ java -jar testModified.jar
Starting ChromeDriver 84.0.4147.30 (48b3e868b4cc0aa7e8149519690b6f6949e110a8-refs/branc]
Only local connections are allowed.
Please see https://chromedriver.chromium.org/security-considerations for suggestions on
ChromeDriver was started successfully.
Jul 30, 2020 6:36:02 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: W3C
Website not present
Build step 'Execute shell' marked build as failure
Finished: FAILURE
```

Step 9: Next, we will setup our pipeline to where **Test** job is only triggered if **Build** job is successful

- Go to **Build** job -> **Configure** -> Add **Post-Build Actions** -> **Build Other Projects** -> Project to Build = **Test** -> Select **Trigger only if build is stable** -> **Save**
- Go to **Manage Jenkins** -> **Manage Plugins** -> **Available** -> Search for **Build Pipeline** -> **Install without restart**

Loading plugin extensions	Success
Javadoc	Success
Maven Integration	Success
Run Condition	Success
Conditional BuildStep	Success
Parameterized Trigger	Success
jQuery	Success
Build Pipeline	Success
Loading plugin extensions	Running

[Go back to the top page](#)
(you can start using the installed plugins right away)

- Go back to Jenkins Dashboard and click the + sign -> Enter view name -> **Build Pipeline View** -> **OK**

All +

View name

Build Pipeline View
Shows the jobs in a build pipeline view. The complete pip

List View
Shows items in a simple list format. You can choose whic

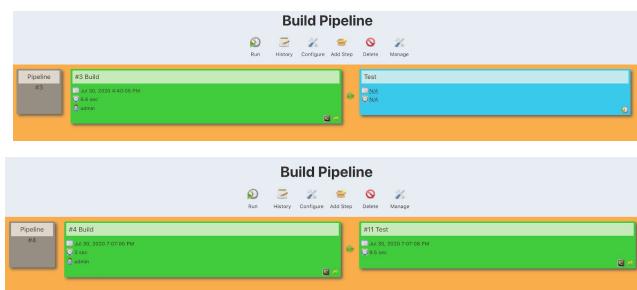
My View
This view automatically displays all the jobs that the curr

OK

- There add the Build Pipeline View title -> Select **Initial Job as Build** -> **OK**

Upstream / downstream config
Select Initial Job

- **Run pipeline**

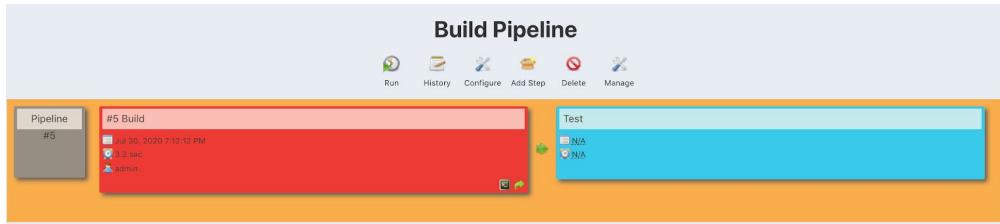


- Verify if pipeline will not continue to **Test** job if the **Build** job fails
- We do this by modifying the Dockerfile to build from a non-existent base image

```
GNU nano 2.0.6           File: Dockerfile

FROM hsha/webapp
ADD ./index.html /var/www/html
```

- Push this to **develop** branch
- Run pipeline. The pipeline should fail because the container will not be created successfully



```
Step 1/2 : FROM hsha/webapp
pull access denied for hsha/webapp, repository does not exist or may
resource is denied
Build step 'Execute shell' marked build as failure
Finished: FAILURE
```

Step 10: Now we configure **GitHub Webhook** so the pipeline will automatically build the application when there is a commit to **develop** branch

- Go to Jenkins Dashboard -> **Build** -> **Configure** -> **Build Triggers** -> Check the **GitHub hook trigger for GITScm polling** -> **Save**
- Next in **GitHub** -> **Settings** -> Click on **Webhooks** -> **Add Webhooks** -> Insert Jenkins server address as shown **JenkinsServerIPAdress:8080/github-webhook/**

The screenshot shows the 'Manage webhook' configuration page on GitHub. It has a header 'Webhooks / Manage webhook' and a descriptive text: 'We'll send a POST request to the URL below with details of any subscribers you'd like to receive (JSON, x-www-form-urlencoded, etc). More information'. Below this is a 'Payload URL *' field containing the Jenkins URL: 'http://18.133.77.181:8080/github-webhook/'.

- Go to the master terminal to trigger a build

```
[ubuntu@Master:~$ git clone https://github.com/regibayoan/devops-capstone-project-1.git
Cloning into 'devops-capstone-project-1'...
remote: Enumerating objects: 15, done.
remote: Counting objects: 100% (15/15), done.
remote: Compressing objects: 100% (13/13), done.
```

- Check webhook status again , there should be a check mark

Webhooks

Add webhook

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our [Webhooks Guide](#).

[http://18.133.77.181:8080/github-webhook/ \(push\)](http://18.133.77.181:8080/github-webhook/)

Edit Delete

- Now try to commit to **develop** branch and verify if pipeline will start automatically
- First, edit the **index.html** file

```
<h2 ALIGN= CENTER>Hello! This is from automatic build</h2>
</body>
```

- Build Pipeline should automatically start once changes is committed to GitHub

Build Pipeline

Run History Configure Add Step Delete Manage

Pipeline #11 #11 Build Jul 30, 2020 7:36:43 PM 3.1 sec

#13 Test Jul 30, 2020 7:36:53 PM 3.9 sec and counting

- Verify the web page has changed

C ⓘ Not Secure | 18.130.76.222

Hello! This is from automatic build

Step 11: Now, we create the pipeline for when pushing to **master** branch

- Create a new job -> **Build-Prod**

Enter an item name

Build-Prod
» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

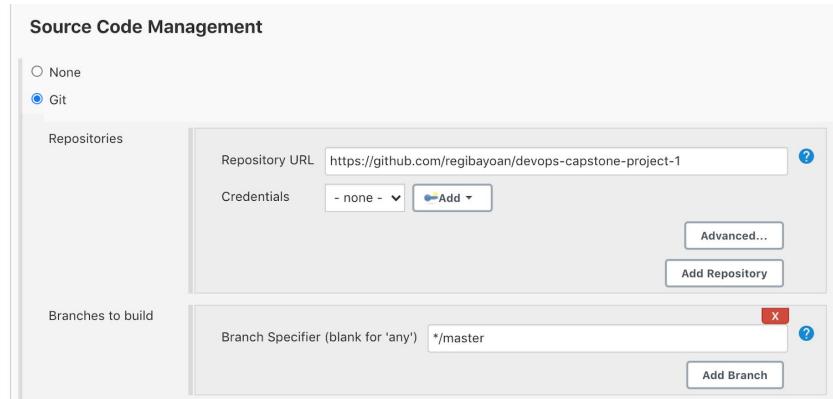
- Restrict to **Slave1** again since we want to do our tests in the **Test** server

Restrict where this project can be run

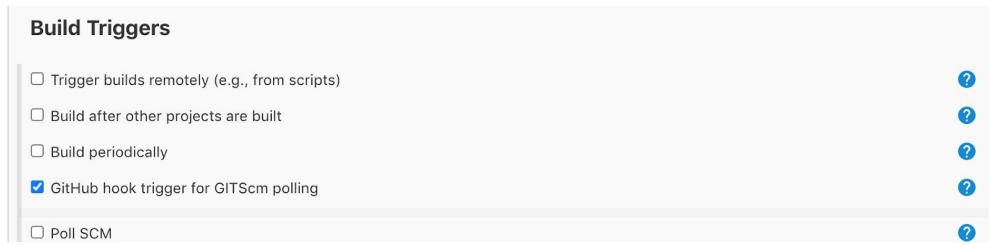
Label Expression Slave1

Label Slave1 is serviced by 1 node. Permissions or other restrictions provided by plugins may prevent this job from running on those nodes.

- Restrict to **master** branch this time



- Select **GitSCM polling** in **Build Triggers** -> **Save**



- Build project just to confirm if the **Build-Prod** workspace is created in **Slave1**



- The items in the **Build-Prod** workspace are correct. The Dockerfile is not present since we only pushed the Dockerfile to 'develop' branch. This time it's for the **master** branch

```
[ubuntu@Slave1:~/workspace$ ls
Build Build-Prod Test
[ubuntu@Slave1:~/workspace$ cd Build-Prod/
[ubuntu@Slave1:~/workspace/Build-Prod$ ls
images index.html
ubuntu@Slave1:~/workspace/Build-Prod$
```

- Next, we go back to the **Build-Prod** project -> **Configure** -> **Build** -> Add commands

Build

Execute shell

```
Command: sudo docker rm -f $(sudo docker ps -a -q)
sudo docker build /home/ubuntu/workspace/Build-Prod -t build-prod
sudo docker run -it -p 80:80 -d build-prod
```

See [the list of available environment variables](#)

- Set Post-build Actions -> Test-Prod -> Select Trigger only if build is stable -> Save

Post-build Actions

Build other projects

Projects to build: Test-Prod

Trigger only if build is stable

Trigger even if the build is unstable

Trigger even if the build fails

Step 12: Create a new job -> Test-Prod

Enter an item name

» Required field

Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

- Restrict where project can be run to **Slave1**

Restrict where this project can be run

Label Expression: Slave1

Label Slave1 is serviced by 1 node. Permissions or other restrictions provided by plugins may prevent this job from running on those nodes.

- Branches to build -> **master**

Branches to build

Branch Specifier (blank for 'any')

Add Branch

- Select Build Trigger

GitHub hook trigger for GITScm polling

- Add commands in **Build -> Execute Shell -> Save**

The screenshot shows the Jenkins 'Build' configuration page. Under the 'Execute shell' section, there is a command input field containing:

```
cd /home/ubuntu
java -jar testModified.jar
```

Below the command field, a link says "See [the list of available environment variables](#)".

- Install **Publish over SSH** plugin
- Go to **Manage Jenkins -> Configure System** -> Go down to **Publish over SSH** -> Add details
- We need the private key that we use to SSH into Slave2. This is usually provided by AWS or by ssh-keygen.
- Finding this will depend on where you saved your .pem file. Copy from the beginning to the end of the key file.

```
→ ~ cat Downloads/intellipaat-london.pem
-----BEGIN RSA PRIVATE KEY-----
-----END RSA PRIVATE KEY-----%
```

The screenshot shows the Jenkins 'Manage Plugins' screen under the 'Available' tab. It lists the 'Publish Over SSH' plugin.

Install	Name	Version	Released
Install	Publish Over SSH	1.20.1	1 yr 10 mo ago

Details for the 'Publish Over SSH' plugin:

- Send build artifacts over SSH
- This plugin is up for adoption! We are looking for new maintainers. Visit our [Adopt a Plugin](#) initiative for more information.

The screenshot shows the Jenkins 'Publish over SSH' configuration screen.

Setting	Value	Help
Jenkins SSH Key	Configure	?
Passphrase	<input type="password"/> Concealed	Change Password ?
Path to key	<input type="text"/>	?
Key	<input type="text"/> -----BEGIN RSA PRIVATE KEY-----	?

SSH Servers

	SSH Server
Name	Slave2
Hostname	18.132.119.90
Username	ubuntu
Remote Directory	/home/ubuntu

[Advanced...](#)

Success [Test Configuration](#)

- Create another **Build** step -> **Send files or execute commands over SSH**
- Add the SSH credentials for Slave2

Send files or execute commands over SSH

SSH Publishers	SSH Server
Name	Slave2
<input type="checkbox"/> Verbose output in console	
<input checked="" type="checkbox"/> Credentials	
Username	ubuntu
Passphrase / Password	<input type="password"/> Concealed Change Password
Path to key	
Key	-----BEGIN RSA PRIVATE KEY-----

- For the **transfers** -> the files will only transfer using relative paths. Using absolute paths somehow does not work.
- Source files = * (Copy all files in /home/ubuntu/workspace/Test-Build) which is the Dockerfile and index.html
- Remote directory = where you want the files to be transferred. In this case, we create a new folder called **production**

Transfers

Transfer Set	
Source files	*
Remove prefix	
Remote directory	production
Exec command	

All of the transfer

- Set Post-build Actions -> PushToProduction -> Select Trigger only if build is stable
-> Save

Post-build Actions

Build other projects

Projects to build: PushToProduction

Trigger only if build is stable

Trigger even if the build is unstable

Trigger even if the build fails

Step 13: Create a new job -> PushToProduction

Enter an item name

PushToProduction
» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

- Restrict where this project can be run -> Slave2

Restrict where this project can be run

Label Expression: Slave2

Label Slave2 is serviced by 1 node. Permissions or other restrictions provided by plugins may prevent this job from running on those nodes.

- Branches to build -> master

Branches to build

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

Add Branch

- Set build trigger

GitHub hook trigger for GITScm polling

- Build -> Execute Shell -> Add commands

Build

Execute shell

Command

```
sudo docker rm -f $(sudo docker ps -a -q)
sudo docker build /home/ubuntu/production -t production
sudo docker run -it -d -p 80:80 production
```

See [the list of available environment variables](#)

[Advanced...](#)

[Add build step ▾](#)

- Install Docker in **Slave2**
- Run an arbitrary container in **Slave2**

Commands: sudo apt update -> sudo apt-get remove docker docker-engine docker.io -> sudo apt install docker.io -> sudo systemctl start docker -> sudo systemctl enable docker -> sudo systemctl status docker

```
ubuntu@Slave2:~$ sudo docker run -it -d ubuntu
526eae6d630115a83527ee51826ee16b3ac7bdc2b2458d0699ddb31c8cf0a669
[...]
```

Step 14: Create a new Build Pipeline

View name: Production Environment

Build Pipeline View
 Shows the jobs in a build pipeline view. The complete pipeline of jobs that a version propagates through are shown as a row in the view.

List View
 Shows items in a simple list format. You can choose which jobs are to be displayed in which view.

My View
 This view automatically displays all the jobs that the current user has an access to.

Step 15: First, make a commit to **develop** -> Merge with **master** -> Test if **Test** and **Production** pipelines run separately

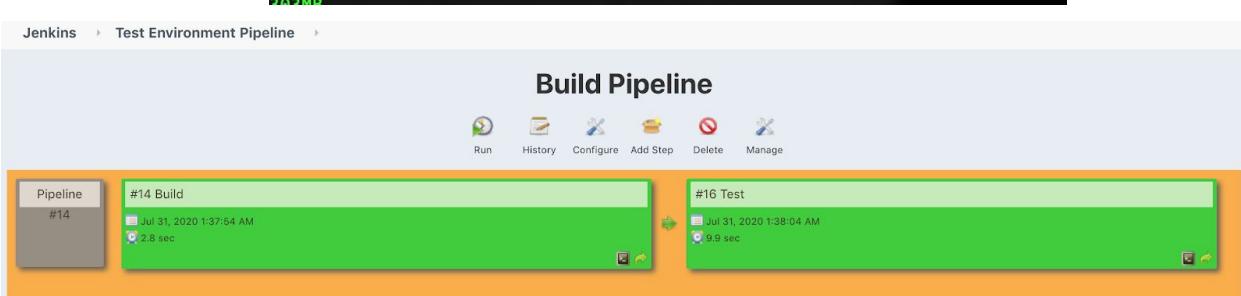
```
GNU nano 2.0.6          File: index.html

<html>
<head>
<title> Intellipaat </title>
</head>
<body style = "background-image:url('images/github3.jpg'); background-size: 100% 100%;">
<h2 ALIGN=CENTER>Hello World Congratulations! You've pushed the website to prod$</h2>
</body>
</html>

→ devops-capstone-project-1 git:(develop) ✘ git add .
→ devops-capstone-project-1 git:(develop) ✘ git commit -m "Commit to dev first"

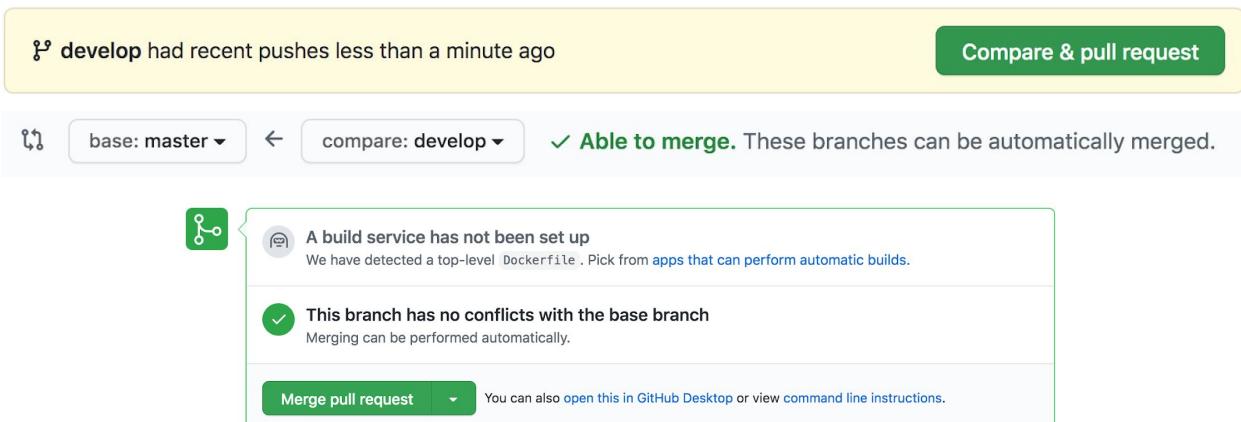
[develop 19b8028] Commit to dev first
 1 file changed, 1 insertion(+), 1 deletion(-)
→ devops-capstone-project-1 git:(develop) git push origin develop

- Confirm the Test Environment Pipeline runs and a new image is built
  ubuntu@Slave1:~/workspace/Test-Prod$ sudo docker image ls
  REPOSITORY          TAG      IMAGE ID      CREATED
  SIZE
  build               latest   321e43206ec9   5 minutes ago
  202MB

Jenkins > Test Environment Pipeline >


```

- Now, merge the **develop** branch to **master** branch. This should trigger the **Production Environment Pipeline** and push the application to **Production** server



⚠ develop had recent pushes less than a minute ago

Compare & pull request

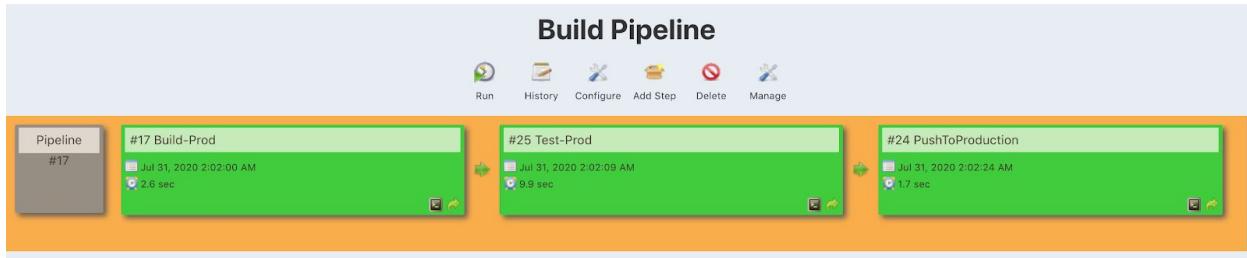
base: master ▾ ← compare: develop ▾ ✓ Able to merge. These branches can be automatically merged.

A build service has not been set up
We have detected a top-level Dockerfile . Pick from apps that can perform automatic builds.

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request ▾ You can also open this in GitHub Desktop or view command line instructions.

- The pipeline runs successfully



- Confirm by visiting <Slave2-IP-Address>:80



Step 16: Once I finished **Points 1-5** of the project, I decided to take the entire architecture down and started again with Configuration Management and Monitoring (**Points 6 & 7**).

Step 17: Create 3 new instances, same setup

Step 18: Install **Puppet** on Master using the commands below

```
On Master :
sudo apt-get update
sudo apt-get install wget
wget https://apt.puppetlabs.com/puppet-release-bionic.deb
sudo dpkg -i puppet-release-bionic.deb
sudo apt-get install puppetmaster
```

- Check if puppet is installed

```
ubuntu@master:~$ apt policy puppetmaster
puppetmaster:
  Installed: 5.4.0-2ubuntu3
  Candidate: 5.4.0-2ubuntu3
  Version table:
 *** 5.4.0-2ubuntu3 500
      500 http://eu-west-2.ec2.archive.ubuntu.com/ubuntu bionic/universe amd64 Packages
      100 /var/lib/dpkg/status
ubuntu@master:~$
```

- Check status of puppet service

```
[ubuntu@master:~$ sudo systemctl status puppet-master.service
● puppet-master.service - Puppet master
  Loaded: loaded (/lib/systemd/system/puppet-master.service; enabled; vendor preset:
  Active: active (running) since Fri 2020-07-31 16:46:48 UTC; 7min ago
    Docs: man:puppet-master(8)
   Main PID: 3015 (puppet)
      Tasks: 3 (limit: 1121)
         ...
          
```

- Next is to fine tune some settings. Edit the **/etc/default/puppet-master** file

```
$ sudo nano /etc/default/puppet-master
```

- Add this line. The reason for this is our Puppet master by default uses 1GB of memory. But since our system also has 1GB of memory, we're going to mention that the Puppet master could only use 512 MB of the memory. Puppet master can work in a 512MB memory as well if we're dealing with less amounts of nodes. For this particular setup, it should be enough.

```
JAVA_ARGS="-Xms512m Xmx512m"
```

```
GNU nano 2.9.3          /etc/default/puppet-master          Modified

# Defaults for puppetmaster - sourced by /etc/init.d/puppet-master
JAVA_ARGS="-Xms512m Xmx512m"
# Start puppetmaster on boot?
START=yes

# Startup options.
DAEMON_OPTS=""
```

- To apply the defaults, we restart the puppet master

```
$ sudo systemctl restart puppet-master.service
```

- We open Port **8140** which is the port where Puppet communicates with the slaves

```
sudo ufw allow 8140/tcp
```

- Puppet master is now set

Step 19: Now setup the slaves as well

- First edit the **/etc/hosts** file. Add the Puppet Master IP address and name it “**puppet**”. Do this for the Master and all the slaves as well

```
GNU nano 2.9.3          /etc/hosts

127.0.0.1 localhost
52.56.116.254 puppet
# The following lines are desirable for IPv6 capable hosts
```

- Now start the puppet agent

```

sudo apt-get update
sudo apt-get install wget
wget https://apt.puppetlabs.com/puppet-release-bionic.deb
sudo dpkg -i puppet-release-bionic.deb
sudo apt-get install puppet

sudo systemctl start puppet
sudo systemctl enable puppet

```

Step 20: On Master, list all the certificates

- Sudo puppet cert list

```

ubuntu@master:~$ sudo puppet cert list
  "slave1.eu-west-2.compute.internal" (SHA256) E4:C2:32:52:A7:15:EA:9F:82:
BB:FB:E2:A2:81:4F:0A:C8:9D:22:62:23:E9:8D:AB:F9:6D:63:E7:F6:39:E3:74
  "slave2.eu-west-2.compute.internal" (SHA256) 9A:31:79:0C:15:22:14:56:78:
8F:D6:85:6B:71:E7:D6:0C:91:2B:86:B7:57:4C:82:C9:F7:0F:B4:B8:A2:31:78
ubuntu@master:~$ 

```

- Sign all the certificates -> **sudo puppet cert sign --all**

```

ubuntu@master:~$ sudo puppet cert sign --all
Signing Certificate Request for:
  "slave2.eu-west-2.compute.internal" (SHA256) 9A:31:79:0C:15:22:14:56:78:8
F:D6:85:6B:71:E7:D6:0C:91:2B:86:B7:57:4C:82:C9:F7:0F:B4:B8:A2:31:78
Notice: Signed certificate request for slave2.eu-west-2.compute.internal
Notice: Removing file Puppet::SSL::CertificateRequest slave2.eu-west-2.comp
ute.internal at '/var/lib/puppet/ssl/ca/requests/slave2.eu-west-2.compute.i
nternal.pem'
Signing Certificate Request for:
  "slave1.eu-west-2.compute.internal" (SHA256) E4:C2:32:52:A7:15:EA:9F:82:B
B:FB:E2:A2:81:4F:0A:C8:9D:22:62:23:E9:8D:AB:F9:6D:63:E7:F6:39:E3:74
Notice: Signed certificate request for slave1.eu-west-2.compute.internal
Notice: Removing file Puppet::SSL::CertificateRequest slave1.eu-west-2.comp
ute.internal at '/var/lib/puppet/ssl/ca/requests/slave1.eu-west-2.compute.i
nternal.pem'
ubuntu@master:~$ 

```

- Our master and slaves can now interact with each other

Step 21: To verify that slaves can communicate with the master, we will create a **Manifest**

- First we create a path on the **master**

```

sudo mkdir -p /etc/puppet/code/environments/production/manifests/

```

- We edit the **Manifest** file on the **master**

```

sudo nano /etc/puppet/code/environments/production/manifests/site.pp

```

- Add the content of the file. This code will ensure that a file '**/tmp/it_works.txt**' is present
- This will also set the file permission to **0644**

```

file {'/tmp/it_works.txt':                                     # resource type file and filer
  ensure  => present,                                         # make sure it exists
  mode    => '0644',                                           # file permissions
  content => "It works on ${ipaddress_eth0}!\n",   # Print the eth0 IP fact
}

```

- **Save** -> Restart puppet master -> **sudo systemctl restart puppet-master**
- Now the slave has to request for the changes
- Before running the command, verify that the file “**it_works.txt**” is not inside /tmp

<pre>[ubuntu@Slave1:/tmp\$ ls systemd-private-a0017bc220a54ca9a3a systemd-private-a0017bc220a54ca9a3a R ubuntu@Slave1:/tmp\$]</pre>	<pre>[ubuntu@Slave2:/tmp\$ ls systemd-private-5664e974eca54af38 systemd-private-5664e974eca54af38 4</pre>
--	---

- Now run the command in slaves -> **sudo puppet agent --test**

<pre>[ubuntu@Slave1:/tmp\$ sudo puppet agent --test Info: Using configured environment 'production' Info: Retrieving pluginfacts Info: Retrieving plugin Info: Retrieving locales Info: Caching catalog for slave1.eu-west-2.compute.internal Info: Applying configuration version '1596218474' Notice: /Stage[main]/Main/File[/tmp/it_works.txt]/ensure: defined content as '{m d5}0d3235d52fb64f5b1d8a0ab10510ffd1' Notice: Applied catalog in 0.02 seconds [ubuntu@Slave1:/tmp\$ ls it_works.txt</pre>

- View the contents inside of the file

<pre>[ubuntu@Slave1:/tmp\$ cat it_works.txt It works on 172.31.45.75! ubuntu@Slave1:/tmp\$]</pre>
--

- Verify the permissions are right too

<pre>-rw-r--r-- 1 root root 26 Jul 31 18:01 it_works.txt</pre>
--

Step 22: Next, we modify the **site.pp** file to where we check if Apache is installed in the system or not.

Code:

- **node default** => any node that requests the change from **master**
- **file** => creates a new directory '**config-management**' since the directory is not present by default.
- **exec** => separate the two conditions using **onlyif** and **unless**

```
GNU nano 2.9.3                                         site.pp                                         Modified

node default {
  file { '/home/ubuntu/config-management/':
    ensure  => 'directory',
    mode   => '0644',
    content => " ",
  }

  exec{'Conditions1':
    command=> '/bin/echo "Apache is installed on this system" > /home/ubuntu/config-management/status.txt',
    onlyif=> '/bin/which apache2',
  }
  exec{'Conditions2':
    command=> '/bin/echo "Apache is not installed on this system" > /home/ubuntu/config-management/status.txt',
    unless=> '/bin/which apache2',
  }
}
```

- Run the command in slaves -> **sudo puppet agent --test**
- Since the servers are new, they will not have Apache installed therefore we're expecting "**Apache is not installed on this system**" to be in **status.txt**

```
[ubuntu@Slave1:~$ sudo puppet agent --test
Info: Using configured environment 'production'
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Retrieving locales
Info: Caching catalog for slave1.eu-west-2.compute.internal
Info: Applying configuration version '1596221122'
Notice: /Stage[main]/Main/Node[default]/File[/home/ubuntu/config-management/]/ensure: created
Notice: /Stage[main]/Main/Node[default]/Exec[Conditions2]/returns: executed successfully
Notice: Applied catalog in 0.04 seconds
[ubuntu@Slave1:~$ ls
config-management  puppet-release-bionic.deb
[ubuntu@Slave1:~$ cd config-management/
[ubuntu@Slave1:~/config-management$ ls
status.txt
[ubuntu@Slave1:~/config-management$ cat status.txt
Apache is not installed on this system
```

- Verify if the **onlyif** block works by installing Apache in the servers

```
[ubuntu@Slave1:~$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
   Drop-In: /lib/systemd/system/apache2.service.d
             └─apache2-systemd.conf
     Active: active (running) since Fri 2020-07-31 19:01:08 UTC; 21s ago
```

- We should see the **status.txt** change once we send a request again

```
[ubuntu@Slave1:~$ sudo puppet agent --test
Info: Using configured environment 'production'
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Retrieving locales
Info: Caching catalog for slave1.eu-west-2.compute.internal
Info: Applying configuration version '159622214'
Notice: /Stage[main]/Main/Node[default]/Exec[Conditions1]/returns: executed successfully
Notice: Applied catalog in 0.04 seconds
[ubuntu@Slave1:~$ cat config-management/status.txt
Apache is installed on this system
```

Step 23: Now we set up a Nagios monitoring service on our **Production** server

- Setup the master
Installation guide: <https://www.itzgeek.com/post/how-to-install-nagios-on-ubuntu-20-04/>
- Confirm master is set up successfully by visiting <Master-IP-Address>/nagios
- We should see the login prompt -> Nagios home page

Step 24: Next is to add the host

- In **Slave2** (Production server):
 - **sudo apt update** -> **sudo apt install nagios-nrpe-server nagios-plugins**
- Edit **nrpe.cfg**
 - **sudo nano /etc/nagios/nrpe.cfg**
 - Uncomment **server_address** and add the address of your Nagios master

```
GNU nano 2.9.3          /etc/nagios/nrpe.cfg          Modified

server_port=5666

# SERVER ADDRESS
# Address that nrpe should bind to in case there are more than one interface
# and you do not want nrpe to bind on all interfaces.
# NOTE: This option is ignored if NRPE is running under either inetd or xinetd

server_address=18.132.119.90
```

- Restart the NRPE server -> **sudo /etc/init.d/nagios-nrpe-server restart**

Step 25: Now we need to setup the **Master** to interact with the client

- In **Master**:

- **cd /usr/local/nagios/etc/objects**
- **ls**
- Open **localhost.cfg**
- **sudo nano localhost.cfg**
- Copy the “**define host**” section and we will use it to define a new host

```
# Define a host for the local machine

define host {

    use            linux-server          ; Name of host template to use
                  ; This host definition will inherit$;
                  ; in (or inherited by) the $;

    host_name      localhost
    alias          localhost
    address        127.0.0.1
}
```

- Now we'll create a new file
 - **sudo nano host1.cfg**
 - We paste the code we just copied, and we edit it a little bit like below:
 - Change the **host_name** and **alias**
 - Make sure to add the **address** to whatever your host address is

```
GNU nano 2.9.3                               host1.cfg                                Modified

define host {

    use            linux-server          ; Name of host template to use
                  ; This host definition will inherit$;
                  ; in (or inherited by) the linux-se$;

    host_name      Production-Server
    alias          nagios-nrpe
    address        18.132.119.90
}
```

- Next is to edit the **nagios.cfg** file by going back one directory
 - **cd ..**
 - **sudo nano nagios.cfg**
 - Scroll down and add a **cfg file**
 - Add the line **cfg_file=/usr/local/nagios/etc/objects/host1.cfg**

```
GNU nano 2.9.3                                     nagios.cfg                                         Modified

# You can split your object definitions across several config files
# if you wish (as shown below), or keep them all in a single config file.

# You can specify individual object config files as shown below:
cfg_file=/usr/local/nagios/etc/objects/commands.cfg
cfg_file=/usr/local/nagios/etc/objects/contacts.cfg
cfg_file=/usr/local/nagios/etc/objects/timeperiods.cfg
cfg_file=/usr/local/nagios/etc/objects/templates.cfg
cfg_file=/usr/local/nagios/etc/objects/host1.cfg
# Definitions for monitoring the local (Linux) host
cfg_file=/usr/local/nagios/etc/objects/localhost.cfg
```

- Restart the Nagios server
 - **sudo systemctl restart nagios**
- Check in Nagios Dashboard -> **Hosts** -> **Production-Server** should be included

Host Status Details For All Host Groups

Host Status Details For All Host Groups				
Host	Status	Last Check	Duration	Status Information
Production-Server	UP	07-31-2020 19:36:33	0d 0h 0m 26s+	PING OK - Packet loss = 0%, RTA = 0.57 ms
localhost	UP	07-31-2020 19:35:09	0d 0h 21m 12s	PING OK - Packet loss = 0%, RTA = 0.05 ms

Step 26: Create a monitoring service

- I've recreated my infrastructure like and the website is up and running

(i) Not Secure | 18.132.119.90

You've pushed the website to prod

- Now we will create a monitoring service that will check if this IP address can be pinged at Port 80
- We go to our Nagios server and go to **localhost.cfg** and let's copy the **service** section of the configuration file

```
ubuntu@master:~$ sudo nano /usr/local/nagios/etc/objects/localhost.cfg
```

```
define service {
    use           local-service          ; Name of service template to use
    host_name     localhost
    service_description PING
    check_command  check_ping!100.0,20%!500.0,60%
}
```

- We'll edit to monitor the **http** service
- We go back and add the **service** code block to **host1.cfg**
- We use the template **generic-service**
- **host_name** would be the same name for your host (**Production-Server**)
- **service_description** - the description of the service that we will see in the GUI

- Command for this particular service is **check_http**. This is the official command for checking Port 80 for any computer.

```
GNU nano 2.9.3                               host1.cfg                                Modified

define host {
    use           linux-server      ; Name of host template to use
                        ; This host definition will inherit$;
                        ; in (or inherited by) the linux-se$;

    host_name    Production-Server
    alias        nagios-nrpe
    address     18.132.119.90
}

define service {
    use           generic-service   ; Name of service template to use
    host_name    Production-Server
    service_description Check Apache
    check_command  check_http
}
```

- Now we save and exit
- Restart the server - **sudo systemctl restart nagios**
- We should see that there is a new service called **Check Apache for Production-Server** in the pending state from the GUI

Limit Results: 100 ▾						
Host ♦♦	Service ♦♦	Status ♦♦	Last Check ♦♦	Duration ♦♦	Attempt ♦♦	Status Information
Production-Server	Check Apache	PENDING	N/A	0d 0h 0m 4s+	1/3	Service check scheduled for Fri Jul 31 21:05:35 UTC 2020
localhost	Current Load	OK	07-31-2020 21:01:24	0d 1h 48m 1s	1/4	OK - load average: 0.00, 0.00, 0.01
	Current Users	OK	07-31-2020 21:02:02	0d 1h 47m 23s	1/4	USERS OK - 1 users currently logged in
	HTTP	OK	07-31-2020 21:02:39	0d 1h 46m 46s	1/4	HTTP OK: HTTP/1.1 200 OK - 11192 bytes in 0.001 second response time

- Wait a few minutes, the status should be **OK** since the website is up and running on **Production server**

Host ♦♦	Service ♦♦	Status ♦♦	Last Check ♦♦	Duration ♦♦	Attempt ♦♦	Status Information
Production-Server	Check Apache	OK	07-31-2020 21:05:35	0d 0h 3m 8s+	1/3	HTTP OK: HTTP/1.1 200 OK - 484 bytes in 0.003 second response time

- By default, **check_http** command takes around 10mins to retry therefore will take a long time to update the status on the GUI
- To modify that, we go to **host1.cfg** in our server
- Add the **retry_interval** and **check_interval** commands. 1 means “1 minute”. The result should be that the GUI will update much faster.

```
define service {

    use           generic-service      ; Name of service template to use
    host_name    Production-Server
    service_description Check Apache
    check_command  check_http
    retry_interval 1
    check_interval 1

}
```

- Then restart server - **sudo systemctl restart nagios**

- We can see these additional commands from **templates.cfg**

Step 26: Verify that the monitoring service we've created is working properly by stopping the container running our website

```
ubuntu@Slave2:~$ sudo docker container stop 52e97bc50fdf
52e97bc50fdf
```

C ① 18.132.119.90



This site can't be reached

18.132.119.90 refused to connect.

- We should see that the status of **Check Apache** is **CRITICAL**

Host ↗⬇	Service ↗⬇	Status ↗⬇	Last Check ↗⬇	Duration ↗⬇	Attempt ↗⬇	Status Information
Production-Server	Check Apache	CRITICAL	07-31-2020 21:14:42	0d 0h 0m 13s	1/3	connect to address 18.132.119.90 and port 80: Connection refused

- Now try running the container again, the **Check Apache** status should go back to **OK**

) Not Secure | 18.132.119.90

You've pushed the website to prod

Host ↗⬇	Service ↗⬇	Status ↗⬇	Last Check ↗⬇	Duration ↗⬇	Attempt ↗⬇	Status Information
Production-Server	Check Apache	OK	07-31-2020 21:17:42	0d 0h 2m 35s	1/3	HTTP OK: HTTP/1.1 200 OK - 484 bytes in 0.002 second response time

Step 27: Finally, push to **develop** branch one more time -> Test server builds and tests the application -> Merge **develop** to **master** -> Test server builds and tests the application, then deploys to Production server -> Production server displays the website -> Production server is monitored using Nagios

18.132.119.90

Capstone Project 1

