

Serial.println("degree Celsius");

Devops

Week 3 - Jenkins Setup on AWS.

When you set up Jenkins on an AWS EC2 instance, it's like creating your own helper robot on the cloud.

This robot can:

- Watch your code (for example, from GitHub).
- Whenever you make changes, it automatically builds it, tests it, and can even put it live on a server, without you doing anything manually.

→ Using EC2 (a virtual computer in AWS) gives you full control over where and how your robot (Jenkins) works.

→ Jenkins on EC2 helps you save time and avoid mistakes by automatically building, testing, and launching your projects.

Steps:-

1. Open AWS account and select EC2 instance.
2. Click on launch instance
3. Select Ubuntu OS. → 22
4. Select "free tier" in Amazon Machine Image (AMI).
5. Select instance type "t2.micro".
6. Create a key pair of your choice.
7. In Network settings click on "Edit". Change type to "All traffic".
8. Now click on launch instance

9. Go to .pem file location (i.e. key pair) and generate gitbash from there)

10. After that to get ssh id goto aws account and click on connect and goto ssh client and copy the ssh id command.

11. Now paste this ssh id in gitbash.

12. Now you need to install the Jenkins. ~~Before~~ So, first you need to update the apt repository.
\$ sudo apt update

13. To install java.

sudo apt install openjdk-21-jdk -y.

14. To ~~set~~ install git and maven and check it

sudo apt-get install -y git maven.

git --version

mvn --version

15. Copy and paste link address

\$ wget https://get.jenkins.io/war-stable/2.479.3/jenkins.war

16) To start the Jenkins service

\$ java -jar jenkins.war

17) How can we access the Jenkins.

Take the public ip address
copy and paste in browser add 8080 as port number


```
Serial.print("Temperature:");  
Serial.print(f);  
Serial.print("degree celcius; Humidity: ");  
Serial.print(h);
```

→ Now give the administrator password i.e in gitlab

→ Now select the install suggested pluggins

→ click on ~~save~~ and continue

→ You can give all admin

→ click on save and finish

→ Now jenkins is ready.

Experiment-1

1. Start devops with a workflow that includes four phases: to do, in progress, code review, and done

→ open Jira → login

→ click on +create project. Give name for project and click next.

→ Add emails of your teammates.

→ In our created project

*Go to timeline.

→ click on +createEpic and give a name

→ for Eg: cmRIT website.

→ click on the epic, a prompt will open.

→ Add some description & click on save.

*Go to Backlog.

→ Add stories to your sprint and assign stories to any team members or else keep it unassigned.

→ start sprint.

→ A dashboard with columns (ToDo, in progress, Done) appears.

→ You can add another "code review" by clicking on + at the right side.

→ move tasks from one to another column.

→ click on complete sprint

```
Serial.print("Temperature:");  
Serial.print(f);  
Serial.print("degree celsius, Humidity:");
```

* Go to summary.

→ we can see the report.


```
Serial.print(f);  
Serial.print("degree celsius; Humidity: ");  
Serial.print("The...");
```

(Experiment-2)

Setup Eclipse for DevOps

Experiment-4:

Aim: Build WAR file in DevOps.

→ In DevOps, building a WAR (Web Application Archive) file and using Jenkins are crucial steps in automating the deployment of Java-based web applications.

In simple terms.

→ Building a WAR file means packaging your Java web app so it can run on a server like Tomcat.

→ Jenkins automates this process - it builds the WAR file every time you update your code, tests it, and can even deploy it - saving time and avoiding mistakes.

Steps:-

Follow the same steps as 'Jenkins setup'.

Build war file:

- * Check Java is installed or not.
- * Check Maven is installed or not.
- * Check Jenkins is installed or not.

Maven download

Go to root directory.

sudo su -

cd /opt

→ do this in another terminal.
i.e. go to connect in instance
and first one connect.

Serial
3 Serial.println(" ");
Serial.println("WiFi connected");
4 loop() ...();

→ open the browser and type maven download

→ right click on `apache-maven-3.9.9-bin.tar.gz` → copy the link.

use link address of maven

→ Now to unzip the maven zip file.

`tar xvf apache-maven-3.9.9-bin.tar.gz`

`ll`

↳ `apache-maven-3.9.9/`

Rename `apache-maven-3.9.9` to `maven`

`mv apache-maven-3.9.9 maven`

`cd maven`

`ll`

↳ `bin/`

`boot/`

`conf/`

`lib/`

note down the maven path: `/opt/maven`

`pwd`

`cd bin`

`ll`

`pwd`

↳ note down the bin path `/opt/maven/bin`

→ Now check the maven is install or not

`mvn --version`

Serial.print("degree

→ It shows 'mvn' not found
→ so, by the below command

→ copy

./mvn --version

↳ It show the maven version.

→ Now go back to the root directory

cd /

→ Check the maven version → ~~mvn~~

mvn --version

→ Again it shows mvn not found

→ Go to the root directory with below command

cd ~

pwd (present working directory)
/root

ll

→ Now you can see the .profile file.

→ open the file with the below command:

sudo vim .profile

→ Go to insert mode (click on I) and give maven,
java home and m2 paths here

M2_HOME=/opt/maven.

M2=/opt/maven/bin.

JAVA_HOME=/usr/lib/jvm/java-21-openjdk-amd64

PATH=\$PATH:\$HOME/bin:\$JAVA_HOME:\$M2_HOME:\$M2

→ to exit enter (Esc + :wq)

→ To get the java-home path,

find / -name java-21*

echo \$PATH

→ you can't see the java and maven path for the above command so you need to restart the .profile file with below command.

source .profile

→ Now you can see the java and maven paths.

→ Now check the maven version

mvn --version

→ Now go to the Jenkins dashboard you need to install one plugin (maven integration).

↳ Install.

→ Manage Jenkins → plugins → available plugins → maven integration plugin

→ without this plugin we can't able to see the maven project

→ once installed click on restart Jenkins.

→ you need to add java and maven paths to Jenkins.

→ Go to Jenkins dashboard → manage Jenkins → tools

Add JDK and Add Maven.

Add JDK

↳ Name: JAVA-21

JAVA-HOME: /usr/lib/jvm/java-21-openjdk-amd64

→ uncheck install automatically

Add Ma

↳ Name

MAVE

→ click on

→ Now w

→ Dashboard

project cli

→ Now go

and part

→ Now go

Git an

→ Now go t

Root po

Goals

→ then

→ click on

→ If job

restart

→ Run the

Now y

Add Maven

↳ Name: maven.

MAVEN_HOME: /opt/maven.

→ click on apply and save it.

→ Now we can create job for war file.

→ Dashboard → newItem → war (give any name) select maven project click on ok!

→ Now go to github account and copy the https path and paste it in the jenkins repository url.

→ Now go to source code management and check the Git and paste the github url in the 'repository url'.

→ Now go to build

Root POM - pom.xml

Goals & options → clean install

→ then click on apply and save.

→ Click on Build Now

→ If job execution takes lot of time

restart the log out the jenkins

stop instance.

restart instance

start jenkins server.

→ Run the job

Now your job will be executed.

Experiment - 6:

im: Deploy the artifact on the Test Server.

→ setup tomcat9 got to tomcat vlt docs

→ copy the url

→ It is continuation of 'real file'

→ connect to sit machine with git bash

wget tomcatlinkaddress

→ update the apt repository.

\$ sudo apt-get update

→ Install tomcat9

\$ sudo apt-get install -y tomcat9

→ Now install tomcat admin as well.

\$ sudo apt-get install tomcat9-admin

→ To access the sit see tomcat

Take publicip of sit server and add 8080 (copy paste the command on the browser)

(you will get a page

It works!

→ Now goto below path in git bash

cd /etc/tomcat9

ll

↳ tomcat-users.xml

Serial.print("degree celsius: ");
Serial.print(h);
Serial.println("send to Thingspeak:");
Serial.println(" ");

→ Now we need to add user to this tomcat-users.xml

\$ sudo vim tomcat-users.xml

→ Goto insert mode and add user below

```
<user username="keerthana" password="keer123" roles="manager-script, manager-status, manager-gui"/>
```

→ to exit :wq

→ Now restart the tomcat service.

\$ sudo service tomcat9 restart

→ Now add one plugin in the jenkins (deploy to container)

→ Manage jenkins → plugins → available plugins → select deploy to container and install it and restart

→ Now goto the jenkins dashboard

Select the war job

Select the configuration steps.

Select ^{In post steps} Post-Build-Actions and search deploy war/ear to a container.

→ Now go to Post-Build-Actions

↳ WAR/EAR files → **/*.war

Context path → /sit

Containers → tomcat9

- Add credentials:

↳ Username - keerthana

password - keer123

Tomcat URL - http://3.110.55.250:8080

public ip of sit server with 8080 port

- to check the artifact is deployed or not.
- Take the sit public ip address and port no and give the content path name.

Ex:- `http://3.110.35.250:8080/sit`

```
Humidity();  
temperature();  
...)
```

```
Serial.print(temperature);  
Serial.print(f);  
Serial.print("degree celsius; Humidity: ");  
Serial.print(h);
```

Experiment-7

Aim: Perform automation using Jenkins.

Whenever you made some changes in source file jenkins job run automatically fetch the code from github to dev server and build the war file and deployed into st.

→ Now open gitbash

```
$ git clone https://github.com/Mahi-Repalle/practice.git
```

```
$ ll
```

```
└─ practice/
```

```
$ cd practice
```

```
$ ll
```

```
└─ src/
```

```
$ cd src
```

```
$ ll
```

```
└─ main/
```

```
$ cd main
```

```
$ ll
```

```
└─ webapp/
```

```
$ cd webapp
```

```
$ ll
```

```
└─ index.jsp
```

```
$ vim index.jsp
```

```
$ git status
```

```
$ git add .
```


\$ git status

\$ git commit -m "ab"

\$ git push origin master

→ Now goto dashboard.

↳ New item → war → configuration → Triggers

↓
Select poll scm
and give cron tab values

→ click on apply and save the job.

→ Don't run the build now

→ Jenkins automatically trigger the job.

→ Now refresh the sit server latest code changes are reflected in the sit

ad Temperature();
(1))

Serial.print(N);
Serial.println("%", send to Thingspeak:");

Experiment-8 for

Aim: Build and deploy a grid Chrome and Firefox based testing.

Procedure:

Steps to build and deploy selenium grid.

Prerequisites:

1. Java Development Kit (JDK installed on the system).

2. Selenium server JAR file (latest version)

→ Selenium grid is a tool within the selenium framework that enables parallel execution of automated test across multiple browser, operating systems and machines.

→ create one aws instances in aws account.

Commands in git bash:

\$ sudo su -

always docker run on root user (##)

curl -fsSL https://get.docker.com -o install-docker.sh

Download and install docker

sh install-docker.sh

Search in chrome as hub.docker.com and search for selenium

Select selenium/hub

1. We need to create the docker network in git bash

docker network create grid

Start the hub using the created network.

docker run -d -p 4445:4444 --net grid
--name selenium-hub selenium/hub

docker run -d -p 4447:7900 --net grid -e
SE_EVENT_BUS_HOST=selenium-hub --name edge
selenium

docker run -d -p 4446:5900 -p 4447:7900 --net
grid -e SE_EVENT_BUS_HOST=selenium-hub -e SE_EVENT
BUS_PUBLISH=^{PORT}4442 -e SE_EVENT_BUS_SUBSCRIBE_PORT
=4443 --shm-size="2g" --name edge selenium/
node-edge:latest.

→ after running this command in git bash open aws
account take the public ip address copy the ip address
and with port number as 4445.

Ex: 13.272.77.07:4445 search this ip address
in edge with get selenium in edge.

Aim: Create

→ Deployment
managing
Kubernetes

→ It run

→ Create

→ Once

cloud con

→ Create

Kubernat

→ select

→ It

→ After

now g

→ Open

→ To se

\$ gcc

→ You

\$ gcc

→ or el

dur

Experiment - 9

Aim: Create deployment resource using Kubernetes

- Deployment resource in Kubernetes is essential for managing the lifecycle of applications running in a Kubernetes cluster
- It runs your app by creating and managing pods.
- Create the google cloud console free account.
- Once account is created you can login to google cloud console.
- Create the Kubernetes cluster. (by going to Kubernetes Engine).
- select create
- It take 5-10 mins to create
- After creating select details and select connect. now you can copy the command line access.
- Open the cloud shell.
- To see the cluster list run the below command.
\$ gcloud container clusters list.
- You can create the cluster with below command.
\$ gcloud container clusters create my-cluster --zone us-central1-a
- or else paste the copied command it creates the cluster.

```
void loop()
{
  h = dht.read Humidity();
  Temperature();
}
```

→ Now go to Kubernetes Engine → cluster, you can see the my-cluster in running.

→ Now run the below command

```
$ gcloud container clusters get-credentials my-cluster --zone us-central1-a
```

→ To see the list of nodes.

```
$ kubectl get nodes
```

→ Create the pods

```
$ kubectl run --image tomcat webserver
```

→ To see pods list

```
$ kubectl get pods
```

→ To get the list of pods along with ip address.

```
$ kubectl get pods -o wide
```

→ Actually you can create pod using definition file
create pd-df1.yaml

```
vim pd-df1.yaml
```

→ For accessing the appⁿ you need to open the port
To open the port.

```
$ gcloud compute firewall-rules create rule2 --allow tcp:8080
```

```
$ kubectl create -f pd-df1.yaml
```

```
$ kubectl get pods -o wide
```

```
$ kubectl get nodes -o wide
```



```
Serial.println("send to Thingspeak");
```

→ To access the pod, take the external ip add
add the port no 8080.

→ open the browser paste `ipaddress:8080`.

→ Now you can see the jenkins.

Experiment-11

Continuation

Aim: Setup Grafana for Devops.

```
$ kubectl get secret prometheus-grafana -n monitoring -o  
  jsonpath='{$.data.admin-user}' | base64 --decode ; echo
```

→ If you run the above command u can see the username for grafana (admin)

```
$ kubectl get secret prometheus-grafana -n monitoring -o  
  jsonpath='{$.data.admin-password}' | base64 --decode ; echo
```

→ If you run above command u can see the password for grafana (prom-operator)

→ PORT FORWARDING

```
$ kubectl port-forward svc/prometheus-grafana 3000:80 -n  
  monitoring.
```

→ click on webpreview and change port no to 3000 and click on change and preview.

→ Now u can see grafana.

→ Enter username, password

```
Serial.print(n);  
Serial.println("%s send to Thingspeak:");
```

Experiment -12

Aim: Setup Prometheus for Devops.

```
$ helm repo add prometheus https://prometheus-community.github.io/helm-charts
```

```
$ helm repo update
```

```
$ helm install prometheus prometheus-community/kube-prometheus-stack --namespace monitoring --create-namespace
```

→ This will install prometheus, alertmanager and grafana.

→ check the prometheus pods and services.

```
$ kubectl get pods -n monitoring
```

```
$ kubectl get svc -n monitoring
```

→ Access prometheus and port forwarding.

```
$ kubectl port-forward svc/prometheus-kube-prometheus-prometheus 9090:9090 -n monitoring
```

→ click on web preview change port to 9090.

→ Now you can able to see the prometheus in the browser.

Experiment - 20

Aim: Create a docker image for any application using Docker file and push it to Docker hub.

- Docker is a containerization tool
 - Docker Image is combination of binaries or libraries which are necessary for software application.
 - Docker container: when image is installed and comes into running condition is called container.
- Image → run → container

- Docker Host: Machine on which docker is installed is called as docker host.
- Docker client: terminal which is used to run docker commands (gitbash)

Practical:-

- connect to a/c and create an instance.

- open browser (get-docker.com)

→ ~~git~~

\$ sudo su -

always docker run on root user

- Download and install docker.

```
$ curl -fsSL https://get.docker.com -o install-docker.sh
```

```
$ sh install-docker.sh
```

Vim doctex file

FROM openjdk.

WORKDIR /app

COPY . /app.

RUN javac sample.java

CMD ["java", "sample"]

run sample.java

Write or copy paste calculator program

→ Install java

\$apt-get install openjdk-21-jdk -y

compile and run java program.

javac sample.java

java sample

→ Create docker image for java application.

docker build -t maheedhar45/javacalculator

docker image ls

→ Now push the docker image in to docker hub

→ So create dockerhub account (hub.docker.com)

Login

docker login -u maheedhar45 (username)

password :

docker push maheedhar45/javacalculator

→ Now go and check in the docker hub in repository.

→ Accessing the image.

docker run --name myjava -it maheedhar45/javacalculator