Final Project (Swiggy.com)

DESCRIPTION

**Project objective:**

Amazon Web Services (abbreviated AWS) is a collection of remote computing services (also called web services) that together make a cloud computing platform, offered over the Internet by Amazon for an infrastructure. Let's consider the Swiggy application which uses Devops tools like (Jenkins/Appium/Docker) for improved automated testing infrastructure, continuous integration, continuous deployment, and continuous delivery.

**Background of the problem statement:**

As a part of cloud and devops CI/CD, the user should have access to sign in to the AWS console to perform create an EC2 instance, S3 bucket, EBS volume, and mounting. Create CI/CD pipeline using AWS, Jenkins, Docker and automate the cloud-based testing for mobile application using appium.

**You must use the following:**

* AWS console access
* JAVA 1.8 version
* Appium
* Jenkins in EC2 machine
* Docker in EC2 machine

**The following requirements should be met:**

* Create Maven project for the Swiggy application. (<https://www.swiggy.com/>)
* Write an Appium script.
* Create an AWS account in console.
* Set up user/policies in IAM to perform actions with respect to service.
* Set up SSH and the user should be able to connect the ec2 machine from their local (Mac/Linux).
* Create an IAM user that can perform to create EC2 instance, S3 bucket, creating EBS volume, and mounting in AWS console.
* Set up Jenkins server by installing Jenkins in ec2 and by installing required plugins like Docker/Selenium in Jenkins.
* Set up Docker for containerizing your application on Docker.
* Create a CI/CD pipeline in Jenkins with Appium test cases and containerizing the application using Docker.

Solution

Create Maven project for the Swiggy application. (<https://www.swiggy.com/>)

o Swiggy mobile application to find the ‘Domino’s Pizza’ scenario is taken for automation.

o Simulated mobile of Android 11 version was created using Android Studio

o Device was added with desired capability on the Appium  
o Elements were identified using XPath, ID on the Appium Inspector  
o Details collected for the Swiggy mobile application is automated using JUnit, Android Driver

Graphical user interface, text, application

Description automatically generated

Appium script.

Graphical user interface, application

Description automatically generated

AWS Activities – Instance Creation, IAM, S3, Volume

Create AWC account in console

Graphical user interface, application, PowerPoint

Description automatically generated

Create EC2 instance

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text

Description automatically generated

Create Volume

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Set up SSH and the user should be able to connect the ec2 machine from their local (Mac/Linux).

Graphical user interface, text, application, email

Description automatically generated

Mount EBS Volume

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generatedGraphical user interface, application

Description automatically generated

Create S3 bucket

Graphical user interface, text

Description automatically generated

Set up Docker for containerizing application on Docker.

The following are the major steps:  
• Create & run base container

**accetto/ubuntu-vnc-xfce-chromium-g3** is selected as the base candidate as it has noVNC readily build-in

The ready-made base container can be pulled with the following command :

docker pull accetto/ubuntu-vnc-xfce-chromium-g3

Graphical user interface, website

Description automatically generated

Start the container with the following command:

docker run --privileged -d -p 6901:6901 -p 4723:4723 --name sl\_base accetto/ubuntu-

vnc-xfce-chromium-g3

Graphical user interface, text

Description automatically generated

We can either ope n a terminal from the Desktop Console, or access the shell via command below

docker exec -it <docker id> /bin/bash

Graphical user interface, text, application

Description automatically generated

• Modify running container  
 Installation of Android Emulator and Appium Server

Android SDK installation

Graphical user interface, text, application, email

Description automatically generated

Appium Server installation

Graphical user interface, text, application, email

Description automatically generated

The next milestone includes installation of Android Phone emulator and Appium server. The steps performed via the Linux shell are summarized in the tables below:

*Steps to create virtual phone SLPhone12 & install Appium*

|  |  |
| --- | --- |
| Section | Commands |
| **Create Android Virtual Device (AVD)** |  |
| Update apt & install jre | sudo apt -y update sudo apt install -y default-jre unzip |
| Download & install Android SDK | mkdir -p /home/headless/Android/sdk cd /home/headless/Android  wget https://dl.google.com/android/repository/commandlinetools-linux- 7583922\_latest.zip  unzip -d /home/headless/Android/sdk/cmdline-tools ./commandlinetools-linux- 7583922\_latest.zip  mv /home/headless/Android/sdk/cmdline-tools/cmdline-tools /home/headless/Android/sdk/cmdline-tools/tools |
| Setup environment | echo "export ANDROID\_SDK\_ROOT=/home/headless/Android/sdk" >> /home/headless/.bashrc  echo "export PATH=\$PATH:\$ANDROID\_SDK\_ROOT/cmdline- tools/tools/bin:\$ANDROID\_SDK\_ROOT/emulator" >> /home/headless/.bashrc |
| Download AVD packages | . /home/headless/.bashrc echo "y" | sdkmanager "platform-tools" "build-tools;31.0.0"  echo "y" | sdkmanager "system-images;android- 31;google\_apis\_playstore;x86\_64" |
| **Install Appium** |  |
| Install nvm & install npm | wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.37.2/install.sh | bash |
| Install latest npm | . /home/headless/.bashrc  nvm install 16.13.0 |
| Install Appium | npm install -g appium |
| Update environment | echo "export ANDROID\_HOME=/home/headless/Android/sdk" >> /home/headless/.bashrc  echo "export JAVA\_HOME=/usr/lib/jvm/java-11-openjdk-amd64" >> /home/headless/.bashrc  echo "export PATH=\$PATH:\$JAVA\_HOME/bin" >> /home/headless/.bashrc |
|  |  |

Before proceeding to finalize the container image, necessary start-up script is created in the folder /dockerstartup/.

The following script is saved as slphone\_startup.rc & sourced in the startup.sh file.

headless@629f756298c1:/dockerstartup$ cat slphone\_startup.rc sl\_start\_phone() {

## Start Phone (AVD) and Appium

. /home/headless/.bashrc

date > /dockerstartup/sl\_service.log  
echo "Starting Phone and Appium... " >> /dockerstartup/sl\_service.log

date > /dockerstartup/sl\_phone12.log  
echo "Starting SLPhone12 AVD..."  
/home/headless/Android/sdk/emulator/emulator @SLPhone12 >> /dockerstartup/sl\_phone12.log &

# wait a while for AVD to ready  
sleep 10  
date > /dockerstartup/sl\_appium.log echo "Starting Appium server..."  
appium >> /dockerstartup/sl\_appium.log &

}

• Finalize Docker image  
 Create and Tag the finalized image using docker commit command.

The following table summarize the commands to finalize the docker image ready for subsequent CI/CD pipelines.

|  |  |
| --- | --- |
| Section | Commands |
| Commit the modified base container image | [root@starport-al8 ~]# docker ps -a  CONTAINER ID IMAGE COMMANDS  CREATED STATUS PORTS NAMES  629f756298c1 accetto/ubuntu-vnc-xfce-chromium-g3 "/usr/bin/tini -- /d..." About an hour ago Exited (0) About a minute ago sl\_base [root@starport-al8 ~]# docker commit sl\_base sha256:272736efd99f77541f20f5410b6c63d847f20d26435c54532557fe93e1f97454 [root@starport-al8 ~]# docker images  REPOSITORY TAG IMAGE ID CREATED  SIZE <none> <none>. 272736efd99f 14 seconds ago 15.9GB  accetto/ubuntu-vnc-xfce-chromium-g3 678MB latest 8803787a7659. 3 days ago  678MB |
| Tag the new image | [root@starport-al8 ~]# docker tag 272736efd99f sl\_phone12  [root@starport-al8 ~]# docker images  REPOSITORY TAG IMAGE ID CREATED  SIZE  sl\_phone12 latest 272736efd99f 2 minutes ago 15.9GB  accetto/ubuntu-vnc-xfce-chromium-g3 678MB latest 8803787a7659 3 days ago  678MB |

Creating EC2 instance with EBS volume & Jenkins installation.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

After success login, additional EBS storage can be configured accordingly.

|  |  |
| --- | --- |
| Steps | Description |
| Create partition | root@ip-172-31-23-91 ec2-user]# fdisk /dev/xvdb  Welcome to fdisk (util-linux 2.30.2). Changes will remain in memory only, until you decide to write them. Be careful before using the write command.  Device does not contain a recognized partition table. Created a new DOS disklabel with disk identifier 0x089adfd6.  Command (m for help): g Created a new GPT disklabel (GUID: EB9F0EC0-B28B-4F6A-9E12- B93AAF674786).  Command (m for help): n Partition number (1-128, default 1): First sector (2048-16777182, default 2048): Last sector, +sectors or +size{K,M,G,T,P} (2048-16777182, default 16777182):  Created a new partition 1 of type 'Linux filesystem' and of size 8 GiB.  Command (m for help): w The partition table has been altered. Calling ioctl() to re-read partition table. Syncing disks. |
| Format the partition | [root@ip-172-31-23-91 ec2-user]# mkfs -t xfs /dev/xvdb1  meta-data=/dev/xvdb1 isize=512 = sectsz=512  = crc=1 data = bsize=4096  agcount=4, agsize=524223 blks attr=2, projid32bit=1 finobt=1, sparse=0 blocks=2096891, imaxpct=25  naming log  realtime  =  =version 2  =internal log  =  =none  sunit=0  bsize=4096  bsize=4096  sectsz=512  extsz=4096  swidth=0 blks ascii-ci=0 ftype=1 blocks=2560, version=2 sunit=0 blks, lazy-count=1 blocks=0, rtextents=0 |
| Transfer existing file in /opt | [root@ip-172-31-23-91 ec2-user]# mv /opt /opt.ori && mkdir /opt  [root@ip-172-31-23-91 ec2-user]# mount /dev/xvdb1 /opt[root@ip-172-31-23-91 ec2-user]# mv /opt.ori/\* /opt/  [root@ip-172-31-23-91 ec2-user]# rmdir /opt.ori  [root@ip-172-31-23-91 ec2-user]# df -h |

With the EC2 and EBS storage configured, Jenkins server & Maven can be installed. The procedures are listed below:

|  |  |
| --- | --- |
| Steps | Description |
| Add Jenkins yum repo & epel repo | sudo yum update –y  sudo wget -O /etc/yum.repos.d/jenkins.repo \ https://pkg.jenkins.io/redhat-stable/jenkins.repo  sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key  sudo amazon-linux-extras install epel -y  sudo yum -y upgrade |
| Install Jenkins | sudo yum install jenkins java-1.8.0-openjdk-devel -y |
| Start Jenkins | sudo systemctl daemon-reload  sudo systemctl enable jenkins  sudo systemctl start jenkins |
| Install maven & git | sudo wget https://repos.fedorapeople.org/repos/dchen/apache-maven/epel- apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo  sudo sed -i s/\$releasever/6/g /etc/yum.repos.d/epel-apache-maven.repo sudo yum install -y apache-maven  sudo yum install -y git |

## Install and configure Jenkins

In this step I will deploy Jenkins on EC2 instance by completing the following tasks:

Connect to Linex instance

Using SSH to connect to my instance

specify the private key (.pem) file and ec2-user@public\_dns\_name.

$ ssh -i /path/my-key-pair.pem ec2-user@ec2-198-51-

100-1.compute-1.amazonaws.com

Download and install Jenkins

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

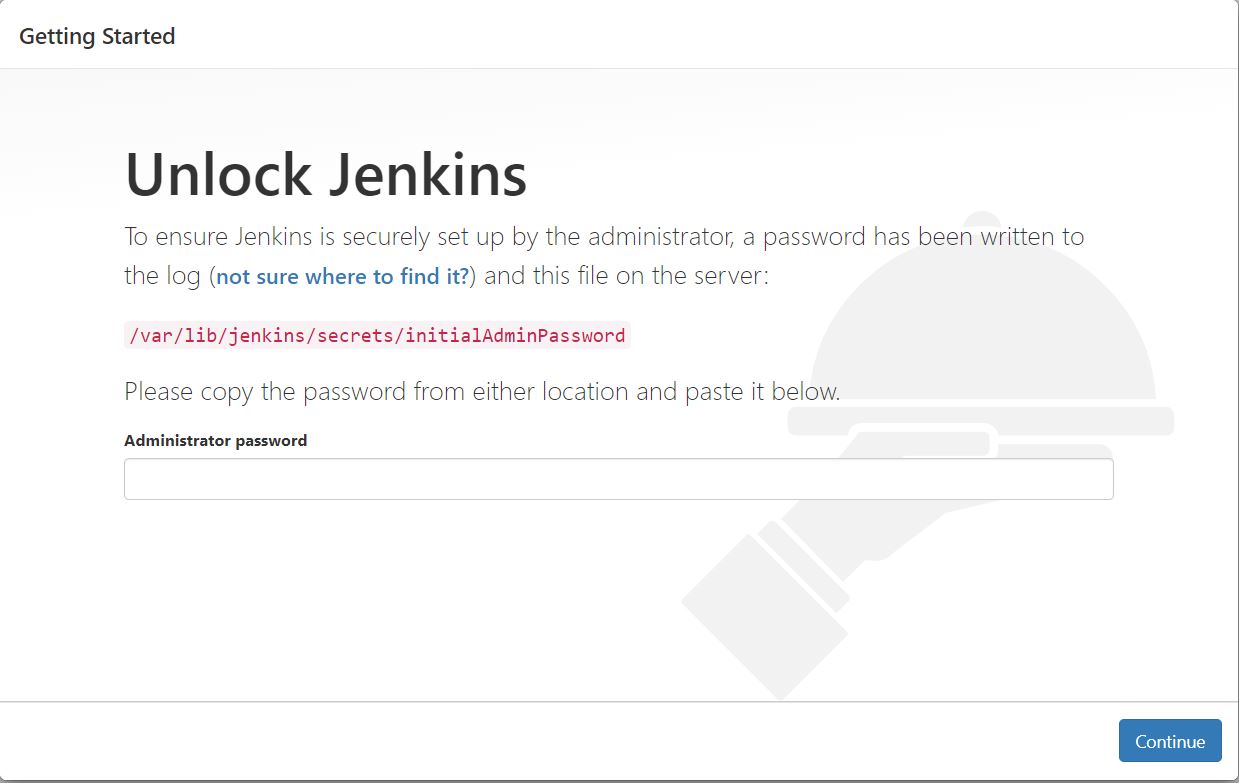
Jenkins installation Graphical user interface, text, application

Description automatically generated

Configure Jenkins

Jenkins is now installed and running on my EC2 instance. To configure Jenkins:

* Connect to http://<your\_server\_public\_DNS>:8080 from a favorite browser. I will be able to access Jenkins through its management interface:



Use the following command to display this password:

[ec2-user ~]$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword

Jenkins CI/CD

With the previous work of Creating Appium test script, installing Docker and AWS EC2 instance creating, the Jenkins CI/CD automation could be performed. The following additional steps on EC2 and Docker’s host are required before creation of Pipeline job.

|  |  |
| --- | --- |
| Steps | Description |
| Adding ssh plugin & pipeline maven integration plugin to Jenkins | page24image12041456page24image12040000 |
| Create a specific user on Docker’s host & edit sudoers accordingly. | * A special user “slearn\_docker” is created on Docker’s host. * Create start / stop Docker container script & enable user   “slearn\_docker” to run related sudo command without password. This will ease the automation.  page24image11862432 |
| Create credential in Jenkins (select ssh with key) | page24image11863888 |

Jenkins Pipeline

New Jenkins pipeline job with following stages is created to automate the test.

|  |
| --- |
| **pipeline {**  **agent any**  **stages {**  **stage('PREPARE: start-remote-docker') { steps{**  **// remote to docker host & start phone emulator container withCredentials([sshUserPrivateKey(credentialsId: "sl-docker-jenkins-key", keyFileVariable:**  **'keyfile')]) {**  **sh 'ssh -p 8898 -i ${keyfile} slearn\_docker@stargate.erengu.info sudo**  **/usr/sbin/sl\_start\_phone12.sh'**  **// add small delay to ensure emulator and appium ready sh 'sleep 10'**  **} }**  **}**  **stage('BUILD: source-from-git') { steps{**  **// Obtain code from GitHub**  **git 'https://github.com/behcs-qm2u/SL\_Phase4Project.git' }**  **}**  **stage('TEST: mvn-test') {**  **steps{**  **} }**  **// Note: Need Pipeline Maven Integration Plugin withMaven {**  **// run mvn test**  **sh 'mvn test'**  **}**  **}**  **}**  **stage('CLEAN-UP: stop-remote-docker') { steps{**  **// remote to docker host & stop / remove phone emulator container withCredentials([sshUserPrivateKey(credentialsId: "sl-docker-jenkins-key", keyFileVariable:**  **'keyfile')]){**  **sh 'ssh -p 8898 -i ${keyfile} slearn\_docker@stargate.erengu.info sudo**  **/usr/sbin/sl\_stop\_phone12.sh' }**  }  }  }  }  *Jenkins pipeline script* |

Conclusion

The project conclusively achieved the integration of efforts ranging from the creation of Jenkins and Android emulators on a Docker container to the development of mobile apps and Appium automation tests.

Although the majority of the work appears to fall within the scope of a DevOps engineer or infrastructure engineer in a specific organization, it is believed that the knowledge gained from this project will be extremely useful for learners pursuing the path of SDET rather than a typical software QA tester.