**Will Gane – Project 4 – CB FSD Integration and Deployment**

**OVERVIEW OF PROJECT CHARTER**

DESCRIPTION

**Project Agenda:**

* To Host the Application on AWS EC2 Instance
* To automate the build and deployment process using Jenkins on AWS EC2
* To Containerize the application using docker on AWS EC2

**Tools:**

* Git and Github
* Java
* SpringBoot
* Angular
* Maven
* Jenkins
* Docker
* AWS [EC2, RDS, S3]

**Description:**

This assignment is designed to help understand how to plan and develop the back end for a given problem. Further, to gain hands-on experience building the CI/CD Pipeline using Jenkins and then containerizing the application on the AWS cloud platform.

**Problem Statement:**

Dr. Shawn runs a pet clinic. He needs to record the visits and other details associated with the pets and their owners visiting his clinic. He has software developed by Bella Solutions, a software company, to manage the same.

Bella Solutions aims to host the software solution for Dr. Shawn on AWS EC2 instance to have online access from anywhere by building CI CD Pipeline and containerizing the solution using Docker on AWS EC2.

UPDATE: TOLD BY PROGRAM STAFF TO LEVERAGE MY PROJECT 3 OUTPUT INSTEAD OF A PROVIDED FILE FOR ANGULAR & MAVEN THAT PERTAINS TO A PET CLINIC.

**Tasks:**

As a task, you need to design and develop the front end for the problem statement along with database design:

1. Import the given Spring Boot project with the generated code in Eclipse [DONE]
   1. Configure the project with Dockerfile [DONE]
   2. Configure the project with Jenkinsfile [DONE]
2. Build the project using the maven package [DONE]
3. Create and Launch AWS EC2 Instance [DONE]
4. Configure EC2 Instance with JDK 11 [DONE]
   1. Install jdk8 [DONE]
   2. Install jdk11 [DONE]
5. Configure EC2 Instance with Docker [DONE]
   1. Install the docker on EC2 [DONE]
6. Configure EC2 Instance with Jenkins [DONE]
   1. Install the Jenkins on EC2 [DONE]
   2. Create the Admin User [DONE]
7. Upload the given code to git repo [DONE]
8. Create Jenkins Pipeline on EC2 with SCM as git [DONE]
9. Build the Pipeline to dockerize the application [DONE]

**My Github repository for Project 4:** <https://github.com/datafriend0/Project4.git>

**OVERVIEW OF PROJECT EXECUTION STEPS**

Rough order of operations for the project:

1. GitHub repository created **>**
2. VSC – Setup in existing Angular project **>**
   1. Jenkins (Angular front-end setup: Jenkinsfile + Jenkins on localhost:8080) **>**
   2. Docker (Angular front-end setup: Dockerfile) **>**
3. Eclipse – Setup in existing Spring Boot project **>**
   1. Docker (Spring Boot back-end setup: Dockerfile) **>**
4. AWS S3 bucket creation / AWS RDS MySQL creation **>** 
   1. Update front end and backend project with production database details **>**
5. Jenkins (Spring Boot back-end setup: Jenkinsfile + Jenkins on localhost:8080) **>**
6. AWS EC2 environment setup and installation **>**
   1. Simultaneously clone GitHub repo to EC2 instance to enable certain installation steps **>**
7. Verify success.

**GitHub**

* Create new Repository
* For existing Project 3 into Project 4 Repo: Git push -u origin master
* Push from local directory to GitHub:
  + Open terminal window from the files you wish to upload (e.g. Desktop/project4)

git init

git add .

git commit -m "push all core files"

git remote add origin https://github.com/datafriend0/Project4.git

git push -u origin main

**Visual Studio Code – Front End Project – Angular** (*Between steps, may need other configs first*)

* Upload Project 3: Import the Angular Project from previous GitHub repository
* Create a Jenkinsfile | For Jenkins to build the project as a Pipeline in Jenkins
  + Create Jenkinsfile in the root directory of the project and push to GitHub so that Jenkins can use it for the pipeline.
  + Stages: Source, Test, Build, Containerize, Deploy
* Build the image in Jenkins on localhost:8080 “Build Now”
  + Run the image as a container in detached mode by exposing the ports
* Create a Dockerfile in root directory for the project
  + Add stages for containerizing the Angular web admin that is building and running the container

FROM nginx

COPY /dist/[angularProjectName]/ /usr/share/nginx/html

* + - Nginx is the Docker image
    - Example:

FROM nginx

COPY /dist/angular-docker/ /usr/share/nginx/html

* + Build as multi-stage Docker with two stages:
    - Angular project into “dist directory” for production
    - Serve the application using NGINX web server
  + Once configurations are complete, build the Angular project in EC2 CLI by moving into the front end folder and running ng build
  + Push to Github, then run the Jenkins Pipeline Project
    - Within Jenkins on port 8080, the stages from Jenkinsfile are visible
    - Verify successful container run: docker ps -a
    - Verify project working on localhost:9090
* UPDATE URL FROM PRE-PROD TO PRODUCTION:
  + OPTION 1 - VSC) Update [services ts folder e.g. product.services.ts] baseurl under Product Service url to localhost:9090 and replace with: EC2 IP address to be:
    - “http://X.XX.XX.XX:9090/”
  + OPTION 2 - CLI) Within CLI for EC2> cd until in app/ > ls
    - Run vi command to open file and edit: vi [service.ts file name e.g. product.service.ts]
    - vi command to switch to command mode: Esc key
    - vi command to insert new information: i (then edit the necessaries)
    - vi command to save and exit: Esc key > :wq!

**Eclipse – Back End Project – Spring Boot** (*Between steps, may need other configs first*)

* Upload Project 3: Import the Spring Boot Project [Related to EC2 Steps]
  + Application.properties file > set server.port=9090
    - Change front end project to match the new port number for backend communications
  + Open Terminal to where the Spring Boot project is stored > mvn package
  + Separately in AWS S3 bucket, upload this generated .jar for Spring Boot project
* Create a Jenkinsfile in Root Directory for the Spring Boot Project
  + Add stages for containerization: Build, Test, Package, Containerize, Deploy
  + Push to Github so that Jenkins can leverage the Jenkinsfile for Pipeline Development
* In Jenkins, create a new Pipeline for the Backend (see steps in Jenkins section)
* Run Jenkins Pipeline: “Build Now”
  + Verify in CLI: docker ps -a
  + Verify in localhost
  + Verify successful services using Postman
* Create a Dockerfile in Root Directory for the Spring Boot Project
  + This will allow the backend project to dockerize Java
  + Write the Dockerfile:
    - Build the jar file of the Java Backend project for Production (3 lines of code)

FROM openjdk:11

COPY ./target/[paste springboot project jar file name]

* + - * + Filename looks like: spring-boot-with-aws-0.0.1-SNAPSHOT.jar
        + Note that no brackets required
        + Verify that jar file is, in fact, within the target file of the backend project

CMD[“java”,”-jar”,”[paste springboot project jar file name]”]

* + - * + Note that no brackets around the jar file name required, but both other brackets required
        + Example:

FROM openjdk:11

COPY ./target/spring-boot-with-docker-0.0.1-SNAPSHOT.jar

CMD ["java","-jar","spring-boot-with-docker-0.0.1-SNAPSHOT.jar"]

* + - Serve the application by executing the Jar on port 9090
* Run Jenkins Pipeline: “Build Now”
  + Notice stages in Jenkinsfile are now deployed in Jenkins
  + Verify in CLI: docker ps -a
  + Verify in localhost 9090
  + Verify successful services using Postman

**Jenkins**

* Initial Configurations
  + Download and leverage localhost:8080 for configuration
  + Create a “Pipeline” project in Jenkins
  + Pass GitHub Repository URL under “SCM”
  + Default branch as “ \*/master ”
  + Script Path = Jenkinsfile
    - This will leverage the already created Jenkinsfile in the root directory of the project
  + Build Triggers > “GitHub hook trigger for GITScm Polling” [For Production URL]
  + Save + “Build Now”
* Settings and Use
  + Verify the container is running the Jenkins build: docker ps -a
  + Check project is running on localhost:8080

**Docker** (Setup in projects [Angular & Spring Boot] + EC2)

* Need to Know for using Docker Images
  + Once Jenkins is configured, configure Dockerfile in the base root project
  + Docker image for Frontend and Docker image for Backend = Two (2) Docker images total
  + Docker Compose will allow for communication between the two images
* **Docker-compose.yml** Located: app’s main folder where frontend and backend files are stored

version: "3.7"

services:

database:

image: mysql:8

container\_name: database

environment:

MYSQL\_ROOT\_PASSWORD: root

MYSQL\_DATABASE: [DB name e.g. dockerinfo\_image]

MYSQL\_USER: [user]

MYSQL\_PASSWORD: [pass]

ports:

- 3307:3306

networks:

- backend

restart: always

spring-boot-app:

build:

context: ./spring-boot-with-docker/

dockerfile: Dockerfile

container\_name: spring-boot-app

depends\_on:

- database

restart: always

ports:

- "9292:9292"

networks:

- backend

- frontend

angular-app:

build:

context: ./[angularProjectName]/

dockerfile: Dockerfile

#image: [imageName e.g. product-frontend-app]

container\_name: [containerName e.g. angular-app]

ports:

- "80:80"

Depends\_on: [backEndProjectName e.g. spring-boot-app]

networks:

- frontend

networks:

frontend:

backend:

**Amazon Web Services (AWS)**

EC2 | Elastic Compute Cloud – Create a virtual machine for running the project; requires installation and configuration of additional tools in order to run the projects in their Production version

**EC2 SETUP - STEP 1 | AWS Console > Open EC2 Dashboard > Launch an instance**

* Name instance > Select OS as “Amazon Linux” with all free tier levels for architecture > t1.micro free tier instance type
* Create New Key Pair
  + Key Pair Name: my-keys
  + Key Pair Type: RSA
  + Private Key File Format: .pem
  + “Create Key Pair” – store key pair .pem file on Desktop
* Launch instance
* Back on Instances dashboard > see new instance listed > wait for Instance State to show green check “Running” state
* Click left check box to select instance > Select “Connect” > Connect to Instance [SSH]
  + Copy “Example:” ssh CLI code listed below the instance ID and 4 steps
  + Open new Terminal window and navigate to where .pem is stored: cd Desktop
  + Paste “Example:” code into Terminal (ssh -I “my-keys.pem” ec2-user…) > Return > yes > Return
* Now within the EC2 instance (EC2 Shell), run the following code for configurations of EC2
  + [Related to initial step for Back End Spring Boot project – upload .jar to S3]
  + Confirm success of access for this .jar file by running: ls

**EC2 SETUP - STEP 2 | Configure integrations for Angular Apps: Java, Maven, Angular CLI, Node JS, Git, Mongo DB, Docker, & Jenkins in CLI**

**STEP 2A) Java configurations for EC2 virtual machine**

* Verify Java not installed: Java --version
* Install Java 8: sudo yum install java-1.8.0-openjdk
* Install Java 11: sudo amazon-linux-extras install java-openjdk11
* Verify installation: Java --version 🡨 should see openjdk 11.0.16
* Allow for switching between versions:

sudo alternatives --config java

Java -jar [paste springboot project jar file name]

* + Filename: java -jar bookCab-0.0.1-SNAPSHOT.jar
  + Note that no brackets required
* Go back to EC2 dashboard in web browser > select current instance > Select “Security”
  + Inbound rules > click “launch wizard” > new window launched
    - Select “Inbound rules”
    - Scroll and select “Edit inbound rules”
    - “Add rule” where:
      * Type: Custom TCP
      * Protocol greyed out with TCP
      * Port range: 9090
      * Source: Anywhere-IPv4
    - Save the rule
  + Back on EC2 Dashboard with current instance selected, observe “Public IPv4 address” > click “open address” > append “ :9090 ” to the end of the IP address
    - Should now see the project running successfully
* *When not actively using the instance, stop the instance using the EC2 dashboard “current state”*

**STEP 2B) Git configurations for EC2 virtual machine**

* Install Git: sudo yum install git -y
* Confirm Installation: git --version
* Bring in Github project via Git (optional to bring in via S3 bucket instead):

git clone [githubURL]

* + e.g. https://github.com/Kaleakash/product\_management\_with\_aws.git
* Verify project is present in EC2 instance: ls
  + Confirm both frontend and backend: cd [SpringBootprojectFileName]
  + Visually confirm both projects + docker-compose.yml file present (3 total)
    - Verify all root files present: cd to spring boot project files
      * If “target” folder **IS NOT** present then install Java (step 2A) + Maven (Step 2C)
      * If “target” folder **IS** present, you are all set
        + Run the mvn package as target folder hosts the .jar file for Spring Boot
    - Verify “target” folder has .jar file present once Java and Maven are both installed

**STEP 2C) Maven & Angular configurations for EC2 virtual machine**

* Install Maven with 3 commands while inside Spring Boot Project directory:
  + Installation command 1: sudo wget https://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo
  + Installation command 2: sudo sed -i s/\$releasever/6/g /etc/yum.repos.d/epel-apache-maven.repo
  + Installation command 3: sudo yum install -y apache-maven
* Verify maven installation successful: mvn --version
* Able to now run spring boot project: mvn package

Enable Angular CLI to run Front End Angular project

* Return to main file for project (should match the directory name from GitHub)
* Move to frontend project: cd [AngularprojectFileName]
* Verify if **Node JS** is installed or not: node --version
  + If no, jump to STEP 2E before progressing
  + If yes, skip this step and progress with Angular installation
* Verify if **Mongo DB** is installed or not:
  + If no, jump to STEP 2G
  + If yes, skip this step and progress with Angular installation
* Verify: npm --version
* Install Angular CLI: npm install @angular/cli
* Ensure on latest version of Angular CLI: npm install -g @angular/cli
  + CLI install asks about autocompletion: yes
  + CLI install asks about data sharing: no
* Verify: ng version
* Angular CLI should then show a ‘landing page’ post install

**STEP 2D) Jenkins configurations for EC2 virtual machine**

* Download latest Jenkins package: sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat/jenkins.repo
* Import the key from Jenkins CI: sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key
* Install Jenkins: sudo yum install jenkins
* Start Jenkins Service: sudo service jenkins start
* Verify Jenkins Service status: sudo systemctl status jenkins
* Retrieve default password: sudo cat /var/lib/jenkins/secrets/initialAdminPassword

**STEP 2E) Node JS configurations for EC2 virtual machine**

* Install node – command 1: curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh | bash
* Install node – command 2: . ~/.nvm/nvm.sh
* Install node – command 3: nvm install --lts
* Install node – command 4: nvm install 16
* Verify node installation: node --version

**STEP 2F) Docker configurations for EC2 virtual machine**

Move within Spring Boot project; verify that target folder is present before installation, if not, install Java + Maven

* Install Docker: sudo yum -y install docker
* Start Docker service: sudo service docker start
* Get information: sudo docker info
* Get images: sudo docker images
* Enable Docker user pt1: sudo usermod enable --now docker
* Enable Docker user pt2: sudo usermod -a -G docker ec2-user
* Install Docker user: sudo usermod -aG docker ec2-user
* Enable Jenkins user pt1: sudo usermod -a -G docker jenkins
* Install Jenkins user pt2: sudo usermod -aG docker jenkins
* Get information: docker info
* Get version detail: docker -v
* Install Docker compose: sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
* Docker setting (apply executable permissions to the binary): sudo chmod +x /usr/local/bin/docker-compose
* Reboot Docker after a few minutes:
  + sudo systemctl daemon-reload
  + sudo systemctl restart docker
  + docker ps -a
* Verify successful Docker Compose installation: docker-compose version

**STEP 2G) Mongo DB configurations for EC2 virtual machine**

Move within Spring Boot project and verify that target folder is present before installation, if not, install Java + Maven.

* (if necessary) Create Mongo file: sudo vi /etc/yum.repos.d/mongodb-org-5.0.repo
  + name = MongoDB Repository
  + baseurl = https://repo.mongodb.org/yum/amazon/2/mongodb-org/5.0/x86\_64/
  + gpgcheck = 1
  + enabled = 1
  + gpgkey = https://www.mongodb.org/static/pgp/server-5.0.asc
  + save changes and quit edit mode :wq!
* Install Mongo: sudo yum install -y mongodb-org
* Start the Mongo Server: sudo systemctl start mongod
* Reload the Mongo Server: sudo systemctl daemon-reload
* Check the Mongo Status: sudo systemctl status mongod
* Run application: mongo

**STEP 2H) Post Configuration Verification**

* After Configuring, Exit.
* Re-Login to verify: sudo usermod -a -G docker ec2-user

**STEP 2I) EC2 Initial Installation and Configurations are now complete.**

*Be sure to stop EC2 instance when not in use.*

**AWS RDS - STEP 3 | Configure Relational Database Services (RDS) for MySQL**

* AWS Console > Open Amazon RDS > Create Database > MySQL (Free tier template)
  + Establish credentials for DB
  + Connectivity: Public = “yes”
* DB Connectivity & Security – use end point and port #s in application.properties file in backend project
* Connect MySQL DB to backend project and manually populate to validate connection

**AWS S3 - STEP 4 | Configure AWS Simple Storage Services (S3) Bucket**

* AWS Console > Open S3 Dashboard > Create Bucket
* (Optional) Per setup in EC2 instructions and Spring Boot instructions above, upload Spring Boot backend project .jar file to S3 for access in EC2

**OVERVIEW OF PROJECT OUTPUTS & VISUALS**

**Graphical user interface, application

Description automatically generated**

Graphical user interface, text, application

Description automatically generated

A picture containing text

Description automatically generated

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application

Description automatically generated