SWE 645

Component-based Software Development

Assignment 3

By:

Abhishek Samuel Daniel

Keerthan Srinivas

Navaneeth Krishna

Thanuj Pathapati

Part 1: Creating the student survey web application.

- Configure the application using springboot initializer. The initializer cab be found at https://start.spring.io/.
- Configure the application with the following settings:
  - Project -> Maven
  - Spring Boot -> 3.1.2 (your choice)
  - Set the Project Metadata -> Name of the project, Name for the application, etc.
  - Select the Packaging type. For this project I chose a .jar file and in the JDK version I chose JDK 17.
  - In the Dependencies Section, choose the following dependencies:
    - Lombok: Helps reduce Boilerplate code.
    - Spring Data JPA: Used to connect and interact with the database.
    - MySQL Driver
    - Spring Web
    - Thymeleaf: To send data to and from the web browser.
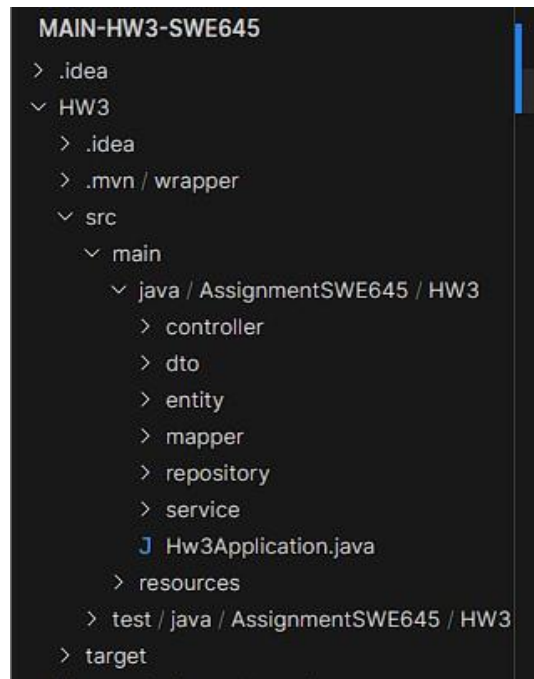 - Click on Generate: This will generate your springboot maven application.

- Next create new packages, under PROJECT_NAME-> src -> main -> java /METADA, to put the respective java classes in. After creation the structure should look like this.

- The Controller package contains the controller class. This class handles the interaction between the web application and the java application and the database.

- The DTO object is used to store information received from the browser.

- The Entity package is used to store the Student Entity. This is a replica to the DTO object but is used to store and interact with the database.

- The Mapper package contains a mapping class that maps the Student DTO to the Student Data. Basically, what it does is it transfers the data it receives from the survey to the student entity that stores it in the database.

- The Repository package contains the repository interface that stores all the data received from Student entity.

- Finally, the Service Package contains two things:

  - An implementation sub package that stores the implementation class used to implement all the set and get operations used to interact with the database.
  - The other part is the Service interface that resides in the main package and is used to initialize all the methods used in the Implementation class.
- At the end you can see the Main application that brings all the components together and is used to trigger and run the application.

```
HW3  C:\Users\endso\Desktop\MAIN-HW3-SWE645\HW3
  > .idea
  > .mvn
  v src
    v main
      v java
        v AssignmentSWE645.HW3
          v controller
              StudentController
          v dto
              StudentDto
          v entity
              Student
          v mapper
              StudentMapper
          v repository
              SurveyRepository
          v service
            v impl
                StudentServiceImpl
              StudentService
          Hw3Application
```
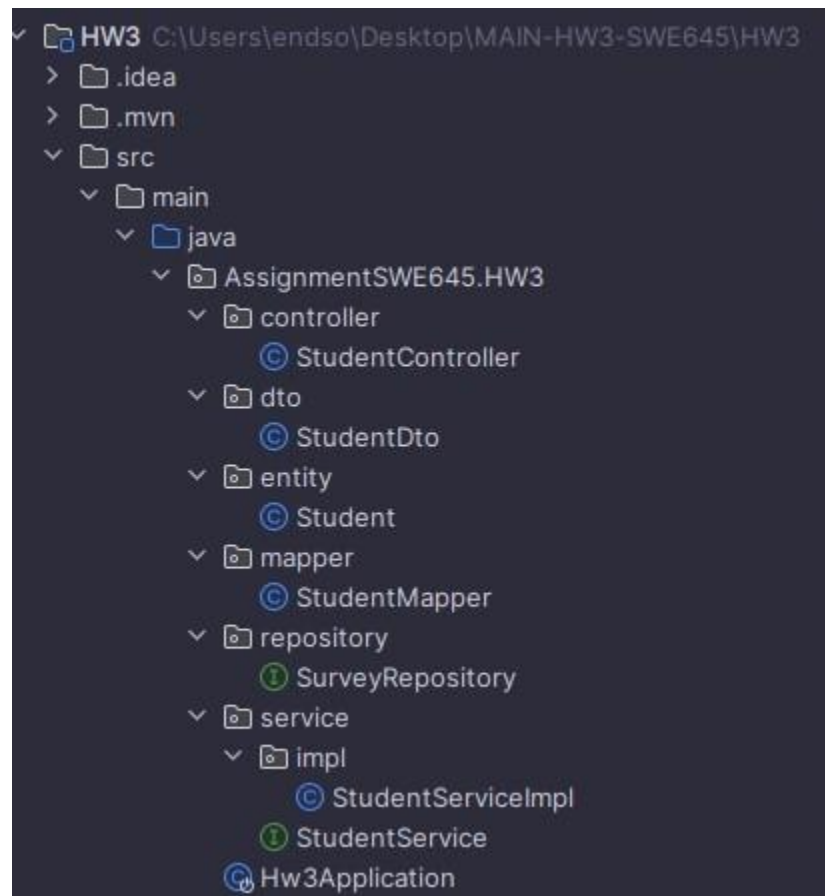
Part 1.3: Creating the HTML templates.

- We need to create an HTML template for each step of the implementation. One for the database entries. One for the survey form.
- In my project I implemented four:

```
v ⊡ resources
  v ⊡ templates
      <> new_survey.html
      <> student_survey.html
      <> update_survey.html
      <> view_student.html
```
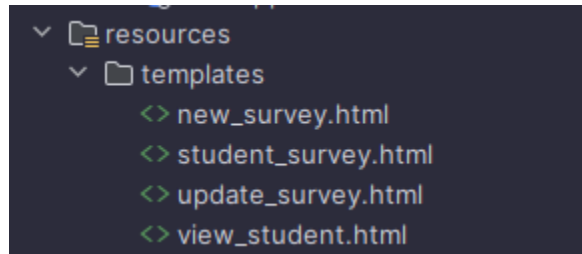
- One for creating the new survey.
- One for viewing the survey entries.
- One for updating an entry.
- And finally, one for viewing an entry.

- The next part is the application.properties file. This file contains the configuration for connecting to the MySQL database.

```yaml
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://${DB_HOST}/${DB_NAME}?allowPublicKeyRetrieval=TRUE
    username: ${DB_USERNAME}
    password: ${DB_PASSWORD}
    hikari:
      initialization-fail-timeout: 0

  jpa:
    database-platform: org.hibernate.dialect.MySQLDialect
    generate-ddl: true
    show-sql: true
    hibernate:
      ddl-auto: update
```

- Data source contains the driver definition. The url to the database and the login details to establish the connections.
- JPA contains the hibernate dialect definition and enables table auto creation.

Part 2: Creating the jar file.

- The first step is building the spring boot application. Most IDEs have a way to build maven projects and if like me you used IntelliJ, you can fin the build option on the right side of the IDE.
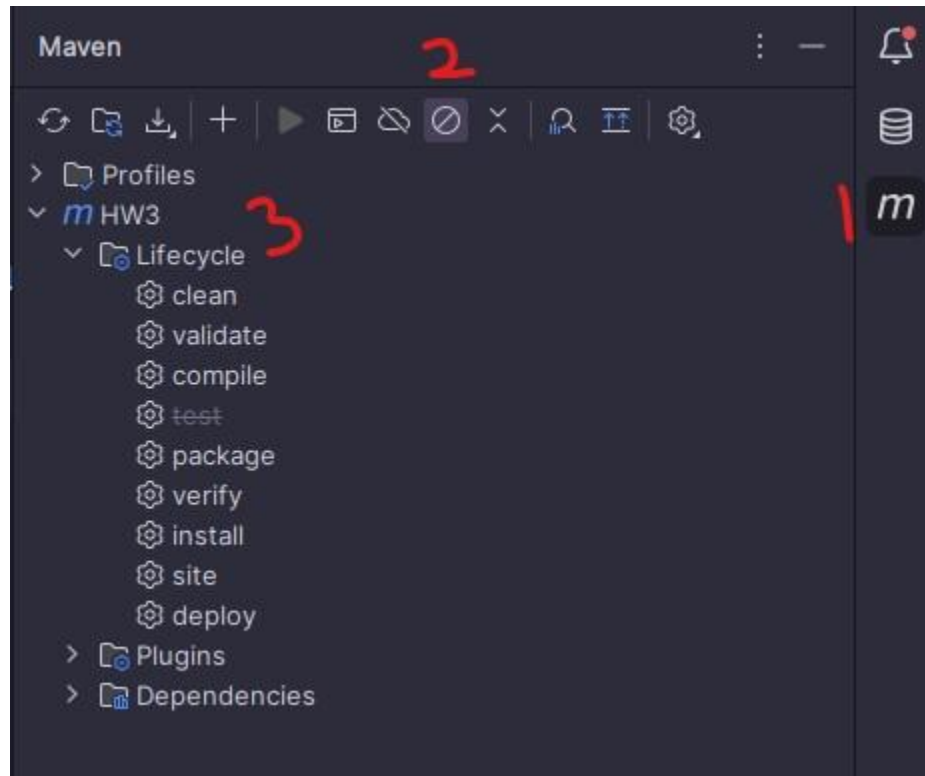
- Select the M on the right side of the screen -> Select the no tests option to skip the testing during building -> Lifecycle -> Select clean and package.
- This will start the building process. After the build has completed successfully. You can find the jar file in the target folder.
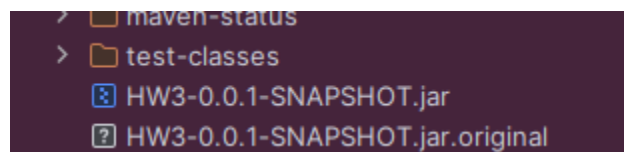
Part3: Building and Pushing the docker image.

- Before building the image, we need to create a Dockerfile. The file needs to be called "Dockerfile". Your dockerfile should look like this.

```
FROM eclipse-temurin:17
LABEL MAINTAINER="asdpkp@gmail.com"
WORKDIR /app
COPY target/HW3-0.0.1-SNAPSHOT.jar /app/swe645hw3.jar
ENTRYPOINT ["java", "-jar", "swe645hw3.jar"]
```

- These are the commands to build and push the image into your docker hub registry.

  - **docker login -u USERNAME -p PASSWORD**
  - **docker build -t IMAGE_NAME .**
  - **docker tag IMAGE_NAME REPOSITORY_NAME/IMAGE_NAME:tag**
  - **docker push REPOSITORY_NAME/IMAGE_NAME:tag**
- The image has been successfully pushed into your docker hub registry.

Part 4: Creating a GitHub Repository and pushing our Spring Boot project.

- Using the github account created for HW2, create a new repository.
- Next go to your folder that contains the files related to the project and initialize a repository.
- Then add all the files pertaining to the project into the repository and commit these changes to the repository with a message.
- Finally push these changes into the repository.
- These are the commands to enter into the command propt:
  - **git init -> This initializes the repository.**
  - **git add –all**
  - **git commit -am "YOUR_MESSAGE"**
  - **git push <YOUR_GITHUB_REPOSITORY_URL>**

Part 5: Creation of Cluster.

- For this Assignment I decided to use Google Cloud Platform (GCP) as my clous service provider. The main reason for doing this is to try something new.
- The first step is to create a free account in GCP. Google gives you $300 dollars as free credit in your free account to use.
- Next navigate to the Kubernetes Engine page. During the initial startup google might redirect you to a services page that asks you to enable the Kubernetes Engine. Here click on enable. This is crucial as without enabling this service you will not be able to create the cluster.
- After enabling the Kubernetes Engine, you will be redirected to the main page of the Kubernetes Engine service.
- Here click on create, found on the top of the page.
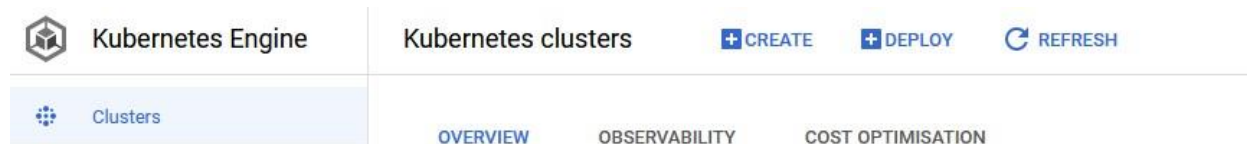
- Clicking on create will redirect you to the cluster creation page. On this page you will be asked to choose whether you want to manually configure the cluster or whether you want to google to automatically scale the cluster as per your needs.
- Here I chose the Standard option.
- The first step after this is naming your cluster. Next in the location type we select regional and, in the dropdown, select a region that is closest to us.
- On the left side choose the Node Pools sections and choose the number of nodes you want your cluster to have. For my assignment I chose 3.
- Next in the configure node setting section select the Image type of your cluster, this is similar to AMI on an EC2 instance in AWS. Here I left the default option as my setting.
- The next step is to configure the performance of your machine. To do this in the machine configuration section under the Configure node setting page select the series of machine you want and select the configuration of the machine.

- Leave all the other settings as default and go to the Security section under the Node Pools category. In here under the access scopes sub sections choose "**Allow full access to all cloud APIs"** . This will give both read and write permissions to your clusters.
- This is it, click on create and your cluster is created.

Part 5: Configuring Jenkins.

- To configure Jenkins, we first need to create a VM instance. All VM instances can be found in the Compute Engine service of Google Cloud. To search for the compute engine service, you can do so in the search bar at the top of the GCP homepage.
- As a side note you will see three instances on this VM Instances page, these are the 3 nodes you created in your cluster, if you see them this means that your cluster was successfully created.
- On this page you will see a "CREATE INSTANCE" button at the top of the page. Click on it.
- First name your instance, something intuitive would be good. Next select the region, it is crucial you select the same region you set for the cluster. The zone you choose is up to you.
- Under the machine configuration choose E2 as its operational cost is low and since it is more than capable of handling our project's needs.
- Next in the Machine Type choose e2 medium as our nodes will need atleast 3 CPU cores to run efficiently.
- Under the Identity and API access section in the Access scopes section choose the Allow full access to all Cloud APIs.
- In the Firewall section choose to Allow HTTP traffic and Allow HTTPS traffic.
- Your configuration is done, and you can click on create.

Na:me* -- — — - -

[instanoe-2

Region * ━━━━━━━━━━   :ljO'JlJe"' -
us-west4 (Las Vegas)   ···· **f1)** [U!H'il'Hst4-b   ····

Region is permanent   Zone is permanent

Machine ieonfilguration

General p111rJH)se   Compute-optimised   Memory-optimised   **GPUs**

Machine types for common workloads, optirnis-ed *for* cost and flex   mty

	Series **f1**	Description	vCPUs **f1**	Mem4
O	,C3	Consistently higll perifformance	4-176	8 -1,
®	E2	Low-cost day-,to-day computing	(125-32	1 - 1:
O	**NI2**	Bala:nced p:rice and performance	2-128	2-81
O	**NI2D**	Balanced price and performance	2-224	2-8¹
O	**T2A**	Scale-Out workloads	1 -48	4-1¹
O	T2D	Scale--0ut workloads	1 - 60	**4-2,**
O	**Nii]**	Balanced price and performance	R25-96	0.6 -

Machiae ty**1**pe

Olloose a machi111e type **with** p:reset amounts of vC Us am! memoiy that S!Uit most workloads. Or, you ca:n oreate a oostom madhine for your worklload's particUJlar needs. Learn more1£

PRESET   **CUSTOM**

e2-medlium (2 **VC U.1** core, 4 GB memory)   ····

                    vCPU   Memo:ry
                    1-'2 vCPU (1 1 shared core)   4! 6B

Identity and API access ❓

Service accounts ❓

Service account
Compute Engine default service account ▾

Requires the Service Account User role (roles/iam.serviceAccountUser) to be set for users who want to access VMs with this service account. Learn more ⧉

Access scopes ❓

○ Allow default access
⦿ Allow full access to all Cloud APIs
○ Set access for each API

Firewall ❓

Add tags and firewall rules to allow specific network traffic from the Internet
☑ Allow HTTP traffic
☑ Allow HTTPS traffic

- GCP will take a few minutes to set up your instance. Once the instance is created you can see a green check mark under the status column.

☐	✓	jenkins-instamce	us-west4-b		10.182.0.2 (nic0)	SSH ▾

- You can directly SSH into your instance using the SSH option under the connect column. This will directly take you into the terminal of the VM instance.
- First, we need to update all the packages in the system and to do this we give the following command.
  - **sudo apt-get update**
- Next, we need to install java onto the system to do this we give the fallowing command.
  - **sudo apt install default-jre**
- Next, we install Jenkins onto the system as well. To do this we enter the following commands.

```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null
```

```
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
```

- Next, we update all the packages on our system. To do this we type.
  - **sudo apt-get update**
- Now that we have imported all the necessary packages, we can move on to installing Jenkins. To do this we type in.
  - **sudo apt-get install jenkins**
- Follow the onscreen instructions and Jenkins will be installed onto the instance.
- To check the status of Jenkins we can type.
  - **sudo systemctl status jenkins**

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Aug  3 03:21:03 2023 from 35.235.242.210
atnk1180@jenkins-instamce:~$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2023-08-01 19:10:43 UTC; 3 days ago
   Main PID: 107451 (java)
      Tasks: 63 (limit: 4691)
     Memory: 1.7G
        CPU: 53min 48.670s
     CGroup: /system.slice/jenkins.service
             └─107451 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins▶
```

- If the status of your Jenkins is like mine, then there is no need to manually start Jenkins. If it is inactive, you can simply start Jenkins using the following command.
  - **sudo systemctl start jenkins**
- Next, we need to install Docker onto the system, so that Jenkins can build and push images to and from our docker hub repository. To do this we give the following command.
  - **sudo apt install docker.io**
- Follow the onscreen instructions, till docker is installed onto your computer. It is best we update all the packages on our system using the apt-get update command.
- You can check whether docker is running on the system using **sudo systemctl status docker** and if it isn't running, we can manually start it using **sudo systemctl start docker**.
- Next, we need to let Jenkins communicate with docker. To do this we give the following command.

- o **sudo usermod -a -G docker jenkins**
- After installing both Jenkins and docker on to our system we need to install kubectl onto the system. This enables us to communicate with the cluster from our VM instance. To do this we type
  - o **sudo apt install snapd**
  - o **sudo snap install kubectl --classic**
- Once this is done, we can boot up the Jenkins user interface. But before we do this let us check if we have the kubeconfig and that we can execute it.
- First to check if we have the kubeconfig we need to give the following command.
  - o **kubectl config view**
- This will give the following output.

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://35.186.162.157
  name: gke_premium-griffin-394400_us-east4-b_microservices-hw3
contexts:
- context:
    cluster: gke_premium-griffin-394400_us-east4-b_microservices-hw3
    user: gke_premium-griffin-394400_us-east4-b_microservices-hw3
  name: gke_premium-griffin-394400_us-east4-b_microservices-hw3
current-context: gke_premium-griffin-394400_us-east4-b_microservices-hw3
kind: Config
preferences: {}
users:
- name: gke_premium-griffin-394400_us-east4-b_microservices-hw3
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1beta1
      args: null
      command: gke-gcloud-auth-plugin
      env: null
      installHint: Install gke-gcloud-auth-plugin for use with kubectl by following
        https://cloud.google.com/blog/products/containers-kubernetes/kubectl-auth-changes-in-gke
      interactiveMode: IfAvailable
      provideClusterInfo: true
```

- Check whether the name of the cluster you are trying to access matches the one you deployed previously. If not, you can import the kubeconfig file of the desired cluster using the following command.

  - o **gcloud container clusters get-credentials CLUSTER_NAME – region=REGION_NAME**

  Note: - If gloud does not come pre-installed onto your system you can manually install by following the instructions on this link: [Google CLI install](.).

- To check if the kubeconfig file has the necessary permissions for execution we need to pass the following command to the shell.
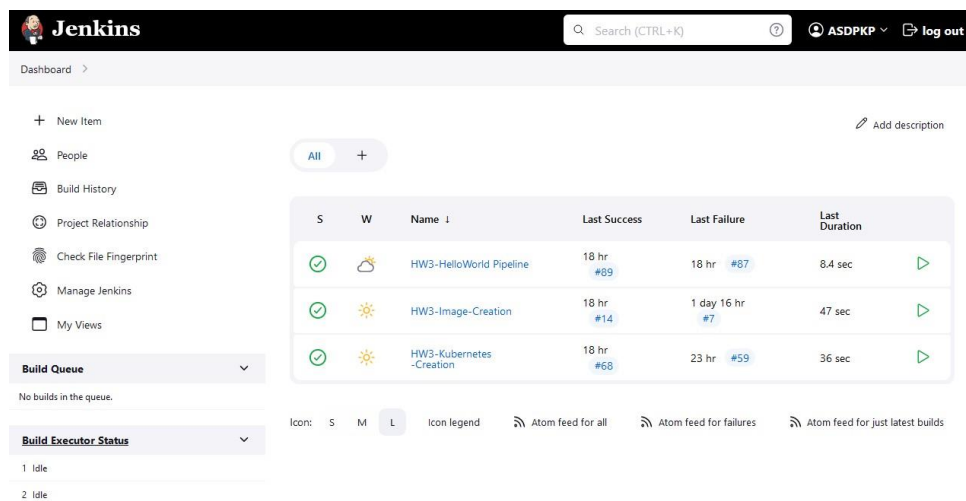
- o **ls -l /var/lib/jenkins/.kube/config**

  Note: - in case no such file or directory error pops up you can move the kubeconfig file from wherever it is residing (usually the home folder) to the Jenkins folder using the following commands.
  - o **sudo mkdir /var/lib/jenkins/.kube**
  - o **sudo mv ~/.kube/config /var/lib/jenkins/.kube/config**
- With this setting up our instance done we can finally login to out jenkins homepage. To do this we grab the external IP address of our machine, which can be found on the VM Instances home page and paste into the url tab of our browser like so: - *<EXTERNAL_IP>*:**8080**

Part 6: Creating the CI/CD Pipeline.

- On the first startup of Jenkins, it asks us for a one time password and also specifies the path the where it can be found, to get this password we need to give the following command:
  - o sudo cat *<PATH_TO_FILE>*
- This will print a one-time password. All you need to do is to copy this password and paste it into the password field.
- After this follow the onscreen instructions until you reach the homepage of Jenkins which looks like this.

- Your homepage should look like this but without the items on the homepage, basically a blank homepage.
- Before we move on, first let us install the required plugins for our pipeline to work. Click on Manage Jenkins -> Plugins -> Available plugins.
- In the search bar check for the Docker Pipeline and install. Then install the google pipeline.
- Once this is done, we need to provide the docker credentials to our pipeline. To do this we go to Manage Jenkins -> Credentials -> (global) -> Add Credentials.
- Leave the Kind as default and the Scope also as default. In the username field we give our docker username and under the password field we give our docker password. And in the ID section we give in our ID, remember the name of the id as we must pass the same value in our Jenkinsfile in order to access these credentials.
- Click on create. This will create our credentials.
- It is now time for us to create our CI/CD pipeline on the homepage select new item. In here give a name and select the pipeline option from the list of options.

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.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

Organization Folder
Creates a set of multibranch project subfolders by scanning for repositories.

- We will setup our pipeline to detect any changes in our github repo and automatically trigger a new build.
- To do this we need to select Poll SCM under the Build Triggers section and in the schedule give the following input: - * * * * *. This tells Jenkins to scan the github repo for any changes every minute and automatically start a build if any changes are detected.
- Under the pipeline section, in the definition dropdown select "Pipeline script from SCM" and in the SCM choose git.
- Now in the Repositories URL field pass the github url to your repo and in the credentials choose the credentials you just created. Finally under the branch specifier change */master to */main.
- Our pipeline setup is done we can click on Apply and then save.
- Jenkins automatically picks up the Jenkinsfile from your github repo and uses it to commence the build.

Jenkinsfile

Your Jenkins file should look like this:

```
pipeline {
    agent any
    environment {
        docker_creds = credentials('docker-credentials')
        }
```

This parts is used to start the pipeline the environment subsection is used to pass the docker credentials we defined in Jenkins to the local variable docker_creds. We will use this to pass our docker user name and password to docker.

```
stages {
    stage('Checking for updates to the GitHub master repo') {
        steps {
            checkout scm
            }
        }
    }
```

This part scours out GitHub repo for changes.

```
stage('Logging into Docker') {
    steps {
        sh "docker login -u ${docker_creds_USR} -p ${docker_creds_PSW}"
        }
    }
```

This stage is for logging into docker using the environment variable docker_creds.

```
stage('Building and Pushing Docker Image') {
    steps {
        script {
            dockerImage = docker.build("asdpkp/latest_hw3:latest")
            dockerImage.push("latest")
        }
    }
}
```

This stage is used for building the docker image and, tagging and pushing the image into the docker hub registry.

```
stage('Cleaning the cluster') {
    steps {
        //sh 'kubectl delete pvc mysql-pvc -n myapp-namespace'
        sh 'kubectl delete deployments myapp -n myapp-namespace'
        sh 'kubectl delete deployments mysqldbi -n myapp-namespace'
        sh 'kubectl delete svc mysqldb-svc -n myapp-namespace'
        sh 'kubectl delete svc myapp-svc -n myapp-namespace'
        //sh 'kubectl delete namespace myapp-namespace'
        //sh 'kubectl create namespace myapp-namespace'
    }
}
```

This stage is used to delete any previous deployments and services in the Kubernetes cluster.

```
stage('Deploying the MySQL container') {
    steps {
        sh 'kubectl apply -f mysql-deployment.yaml -n myapp-namespace'
    }
}
```

This stage is for deploying the mysql database container to the cluster in the namespace myapp-namespace.

```
stage('Deploying the Web Application container') {
    steps {
        sh 'kubectl apply -f app-deployment.yaml -n myapp-namespace'
        }
    }
}
```

This deploys the survey app onto the cluster In the same namespace as the mysql container.

HW3 Application

MyApp Deployment file:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  namespace: myapp-namespace
spec:
  selector:
    matchLabels:
      app: myapp
  replicas: 3
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
        - name: swe645hw3
          image: asdpkp/latest_hw3:latest
          ports:
            - containerPort: 8080
          env:   # Setting Enviornmental Variables
            - name: DB_HOST   # Setting Database host address from configMap
              value: mysqldb-svc

            - name: DB_NAME  # Setting Database name from configMap
              value: ssdb

            - name: DB_USERNAME  # Setting Database username from Secret
              value: root

            - name: DB_PASSWORD # Setting Database password from Secret
              value: root
```

The deployment is created in the myapp-namespace and is deployed into all the three nodes (replicas 3). The name of the container is swe645hw3 and the image being used is asdpkp/latest_hw3: latest. The container port is 8080. The DB_HOST is mysqldb_svc

(service name), DB_NAME is ssdb, DB_USERNAME is root and, DB_PASSWORD is root. The above values are passed to the application.yaml file (application.properties) file. Which are then used to connect the database.

MyApp Service:

```yaml
apiVersion: v1 # Kubernetes API version
kind: Service # Kubernetes resource kind we are creating
metadata: # Metadata of the resource kind we are creating
  name: myapp-svc
  namespace: nginx-app-namespace
  annotations:
spec:
  selector:
    app: myapp
  ports:
    - protocol: "TCP"
      port: 8080 # The port that the service is running on in the cluster
      targetPort: 8080 # The port exposed by the service
  type: NodePort # type of the service.
  #loadBalancerIP: 34.86.124.111
```

Service is also deployed in the same namespace as the deployment. The protocol used is TCP. It is deployed on nodes with the name myapp. It runs on port 80 and exposes port 8080 for communications. And the type of service is a Load Balancer. I also provided a static IP address for the Load Balancer which is used in the extra credit.

MySQL

MySQL Persistence Volume Clain:

Since the database is running on the cloud, we need to allocate some of the storage to the database to do these we need to raise a persistent volume claim to the nodes.

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pvc
  namespace: myapp-namespace
  labels:
    app: mysqldbi
    tier: database
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 256Mi
```

The volume is also deployed in the same namespace as myapp. AccessMode is read write once and the storage being allocated is 256 MB. The name of the volume is mysql-pvc.

MySQL Deployment:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysqldbi
  namespace: myapp-namespace
  labels:
    app: mysqldbi
    tier: database
spec:
  selector:
    matchLabels:
      app: mysqldbi
      tier: database
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysqldbi
        tier: database
    spec:
      containers:
        - name: mysqldbi-cont
          image: mysql
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: root

            - name: MYSQL_DATABASE
              value: ssdb

          ports:
            - containerPort: 3306
              name: mysqldbi-cont

          volumeMounts:
            - name: mysql-persistent-storage
              mountPath: /var/lib/mysql

      volumes:
        - name: mysql-persistent-storage
          persistentVolumeClaim:
            claimName: mysql-pvc
```

Name of the deployment is mysqldbi which is deployed in the myapp-namespace. The name of the container is mysqldbi-cont and the image being used is the latest version of mysql. The root password is root which is stores in MYSQL_ROOT_PASSWORD and the name given to the database is ssdb which is stored in MYSQL_DATABASE. The port used for deploying this container is 3306. The persistent is to be stored in the mount path /var/lib/mysql.

MySQL Service:

```
apiVersion: v1
kind: Service
metadata:
  name: mysqldb-svc
  namespace: myapp-namespace
  labels:
    app: mysqldbi
    tier: database
spec:
  selector:
    app: mysqldbi
    tier: database
  ports:
    - port: 3306
      targetPort: 3306
```

The service name is mysqldb-svc deployed in the namespace myapp-namespace. The pods used for deploying the container are mysqldbi. The ports used for deployment port is 3306 and exposes the port 3306.

Links:

GitHub Multi Repo: https://github.com/ASDPKP/MAIN-HW3-SWE645.git

Github Mono Repo:

1. https://github.com/ASDPKP/HW3.git
2. https://github.com/ASDPKP/HW3-kubernetes-Deployment.git

MyApp Link: 34.107.188.102/survey (Copy and Paste the URL)