Week6 Revisit

AWS EC2 REGION AZ(Availability Zone)

Git Branching & Merging

S3 – Simple Storage Service (Persistence Storage Service)

RDS –

EC2 –

Docker – Is a DevOps Tool to manage Containers (Create, Run, Update & Remove)

Images – Application bundled as a single file along with all the required libraries and dependencies.

The main objective of docker is – To Eliminate the fact that, the code was running in my system and is not running in this system.

Containers – With the help of images – which runs the code/application in an isolated area with all the required libraries, runtime and dependencies.

Docker – Containerization Management Tool

Docker Server & Docker Client , Docker Hub (Cloud Services)

Docker Client (CUI based – run the docker commands in command prompt) (GUI based – Docker desktop)

WSL – Windows Subsystem for Linux & Virtualization Enabled (Hyper -V)

Containers VS VM (Virtual Machines)

Containers shares the OS Kernel (So Smaller in size, Light weight alternative for VM)

DockerFile – Is used to create docker image. [FROM, ENTRYPOINT/CMD]

COPY , WORKDIR

Dockerfile is a file without any extension. Docker engine will use this file to create docker image.

# Define the parent image

FROM ubuntu

# Install needed programs

RUN apt-get update

RUN apt-get -y dist-upgrade

RUN apt-get -y install default-jdk

# Within the image and thus container, set the working directory to the new directory example

WORKDIR /example

# Create Hello World Java program and save it in the appropriate file

RUN echo 'public class HiWorld{ public static void main(String[] args){System.out.println("Hi world");}}'> HiWorld.java

# Compile the Java program, creating the file that the JVM can actually run

RUN javac HiWorld.java

# Run the HelloWorld program in the container

CMD ["java", "HiWorld"]

DockerCompose.yml – To manage multiple containers

version: "3"

services:

app:

build: ./app

networks:

- backend

- frontend

web:

build: ./web

ports:

- "8000:8000"

networks:

- frontend

db:

image: postgres

ports:

- "8001:5432"

networks:

- backend

networks:

frontend:

backend:

Image 🡪 Container === Class -🡪 Object ( A Single Class can create n number of object)

(A Single docker image can create n number of containers)

Container – is a running version of a docker image. Which has unique id, name (unique) and image

Image – has a unique id and unique name which can be any software application or Operating System

Docker hub (repositories of images) === github (Repositories of source code)

Jenkins – It is a DevOps Tool

It helps to automate the Build process.

It works with many other tool like github, gitlab, bitbucket, …..

Jenkins – Is a CI/CD Tool

CI – Continuous integration (Integrating all the latest code and Testing it for any side - effects)

CD – Continuous Deployment (Running in Production Environment) / Delivery (Testing the o/p of UAT)

UAT – User Acceptance Testing

Jenkins is used to create Job, that can be automated to do any task.

Jenkins will use the build script for the automation purpose.

Installing Jenkins (WAR, Docker, Installer), Running Jenkins, Creating a Job, Adding plugin to Jenkins

8080- InitialAdminPassword, Local admin user.

Jenkins – Is an automation Server that can be configured in numerous ways.

SonarQube – Code Quality Analysing Tool (Min Req JDK11)

SonarQube – Stand – alone Edition (It uses port no 9000 as default port) – Create a project, key, select type of project and run the given command)

Sonar Lint – Is a Plugin that can be added to Eclipse/VS Code

Sonar cloud – Web Version of SonarQube

Hibernate – ORM Framework (Object Relational Mapping)

Hibernate is a JPA Implementation (Java Persistence API – Serializing the state of object to the DB Table)

Hibernate simplifies the process of interaction with any RDBMS. Using the POJO & Bean class, easily Interact with the DB Tables.

HQL – Hibernate Query Language (It syntax are little similar to SQL but it helps to write DB independent Query

|  |  |
| --- | --- |
| SQL | HQL |
| Select \* from Student; (table name) | FROM Student; (Student in Entity Bean Class name) |
| Insert into Student values (……); | Save/persist (EntityBean object) |
| Update Student set ……. Where id = . | getForId(), upsert(), Save/Persist (Object) |
| Delete from Student where id = ? | Remove(Object obj) |
|  |  |

Hibernate expect two imp config files

1. Hibernate.cfg.xml – (DB details, hbm2ddl, show\_sql, format\_sql—sessionFactory is the root tag)
2. <entity\_bean\_name>.hbm.xml ( Mapping file – optional – use annotations in Entity Bean class instead)

* Configuration Class
* SessionFactory Interface
* Session Interface
* Transaction Interface
* Query Interfaces

Criteria is an OO implementation of DB Operation (CRUD Operation)

HQL – Hibernate Query Lang

JPQL – Java Persistence Query Lang

**HQL Examples**

* HQL **Select** Query example to retrieve a student details whose id is 101.

TypedQuery<Student> query = session.createQuery("FROM Student WHERE id = '101' ", Student.class);

List<Student> students = query.getResultList();

* HQL **Update** Query example to update the name to "John" whose id is 105.

Query query = session.createQuery("UPDATE Student SET name = :stud\_name WHERE id = :stud\_id");

query.setParameter("stud\_name", "John");

query.setParameter("stud\_id", "105");

int result = query.executeUpdate();

Please note that HQL should only be used to batch-update records. If you are updating a single record, it's preferable to update the actual Java object's properties, and then persist that object and its changes back to the database with session.update(object) or session.flush()

* HQL **Delete** Query example to delete a student whose id is 108.

Query query = session.createQuery("DELETE Student WHERE id = :stud\_id");

query.setParameter("stud\_id", "108");

int result = query.executeUpdate();

Similarly to updates, HQL should only be used to batch-delete records. If you are deleting a single record, it's preferable to use session.delete(object).

* HQL **Insert** statement cannot insert values directly in to a table. It is only used to insert rows from another table. It supports only INSERT INTO … SELECT … .

Query query = session.createQuery("INSERT INTO Student(id, name) SELECT s\_id, s\_name FROM NewStudent");

int result = query.executeUpdate();

Native SQL

## Hibernate NativeSQL

Hibernate provides the option to execute native SQL queries through the use of the **SQLQuery** object. The Session.createNativeQuery(String query) method is used to create the [NativeQuery](https://docs.jboss.org/hibernate/orm/5.4/javadocs/org/hibernate/query/NativeQuery.html) object and execute it.

**Example:**

Query query = session.createNativeQuery("SELECT id, name, age FROM student");

// Get All Students

List<Object[]> rows = query.getResultList();

for(Object[] row : rows){

Student stud = new Student();

stud.setId(Long.parseLong(row[0].toString()));

stud.setName(row[1].toString());

stud.setAge(Long.parseLong(row[2].toString()));

System.out.println(stud);

}

The getResultList() method returns a List of Object arrays, which we need to explicitly parse to double, long etc. Hibernate will use **ResultSetMetadata** to deduce the actual order and types of the returned scalar values.

To avoid the overhead of using ResultSetMetadata, or simply to be more explicit in what is returned, one can use addScalar():

session.createNativeQuery("SELECT \* FROM student")

.addScalar("id", LongType.INSTANCE)

.addScalar("name", StringType.INSTANCE)

.addScalar("age", LongType.INSTANCE);

The addEntity() method used to get entity objects from a Native SQL query.

Query<Student> studentSQLQuery = session.createNativeQuery("SELECT id, name, age FROM student").addEntity(Student.class);

Alternatively, this can be provided as an argument to the overloaded method:

Query<Student> studentSQLQuery = session.createNativeQuery("SELECT id, name, age FROM student", Student.class);

The addJoin() method used to fetch the data from associated table using tables join.

Query<Student> query = session.createNativeQuery("SELECT s.id, s.name, a.\* FROM student s JOIN address a ON s.id = a.stud\_id")

.addEntity("e", Student.class)

.addJoin("a", "student.address");

XML/JSON – Language/Platform/Architectural independent way of storing and exchanging the data.

XML

1. Well formed (Checking XML rules – tag needs to be closed, nested properly, proper case and space
2. Valid (Checking against XSD/DTD)

XSD – XML Schema Definition

DTD – Document Type Definition/Description

Parser – Is a tool to read/extract the data from XML file

Parsing – Is a process that parser will do

Marshalling – Reading XML data and converting to Java Object

Un Marshalling – reverse of marshalling

JAXP – Java API for XML Parsing

JAXB – Java And XML Binding