Jenkins

- Jenkins
    - Java Maven Basics
    - Jenkins with Maven Build
        - Setup maven
        - Maven build phases
            - pom.xml definitions
    - Jenkins Build using Jenkins Job
        - pom.xml parameters
        - Artifacts Archive
    - Jenkins Build and Deploy
        - Setup Apache Tomcat on Amazon Linux
        - Tomcat War file deployment Configs
            - Tomcat Users Configuration
            - Tomcat Network Configuration
        - Apache Tomcat Terminology
        - Jenkins Job to deploy war file
            - Jenkins Plugin installation
        - Jenkinsfile Deployment for Build and Deployment
        - Assignment
            - Execution of Java Build and Deployment using AWS CICD.
        - Java Database Connection
        - Using Mysql Database Server
            - Configure Mysql DB AWS RDS
                - Connecting to an RDS
        - Multiple Environment Configuration
            - SSM Parameter Store.
            - AWS Secrets Manager
        - Infra Creation using IAC-Assignment
            - Execution of Java Build and Deployment using AWS CICD.
    - Jenkins Backup
        - Manual Backup
        - Automated Backup
        - Scripted Backup
    - Working with Jenkins
    - Troubleshooting
    - Jenkins Do's & Dont's
        - Do's
        - Dont's

Java Maven Basics

- Check for Java Maven Basics Steps.

Jenkins with Maven Build

Setup maven

- Go to ***Jenkins Dashboard -> Manage Jenkins -> Under System Configuration select Tools > Maven > Maven installations > Add Maven > Give a Name Maven_Local > Check Install Automatically > Install from Apache (specify a version) > Save***
- You can give a logical name to identify the correct version while configuring a build job.

--

Maven build phases

- Maven itself requires Java installed on your machine.

- You can verify if Maven is installed on your machine by running **mvn -v** in your command line/terminal.

- Maven is based on the `Project Object Model (POM)` configuration, which is stored in the XML file called the same – `pom.xml`. It is a structured format that describes the project, it's dependencies, plugins, and goals.

- `pom.xml` file should be present in your project directory

- Below are the Maven Build Phases

  - **Validate** : Validate Project is correct & all necessary information is available.
  - **Compile** : Compile the Source Code
  - **test** : Test the Compiled Source Code using suitable unit Testing Framework (like JUnit)
  - **package** : Take the compiled code and package it.
  - **install** : Install package in Local Repo, for use as a dependency in other project locally.
  - **Deploy** : Copy the final package to the remote repository for sharing with other developers.

- The above are always are sequential, if you specify `install`, all the phases before `install` are executed.

```
mvn install -f pom.xml
```

--

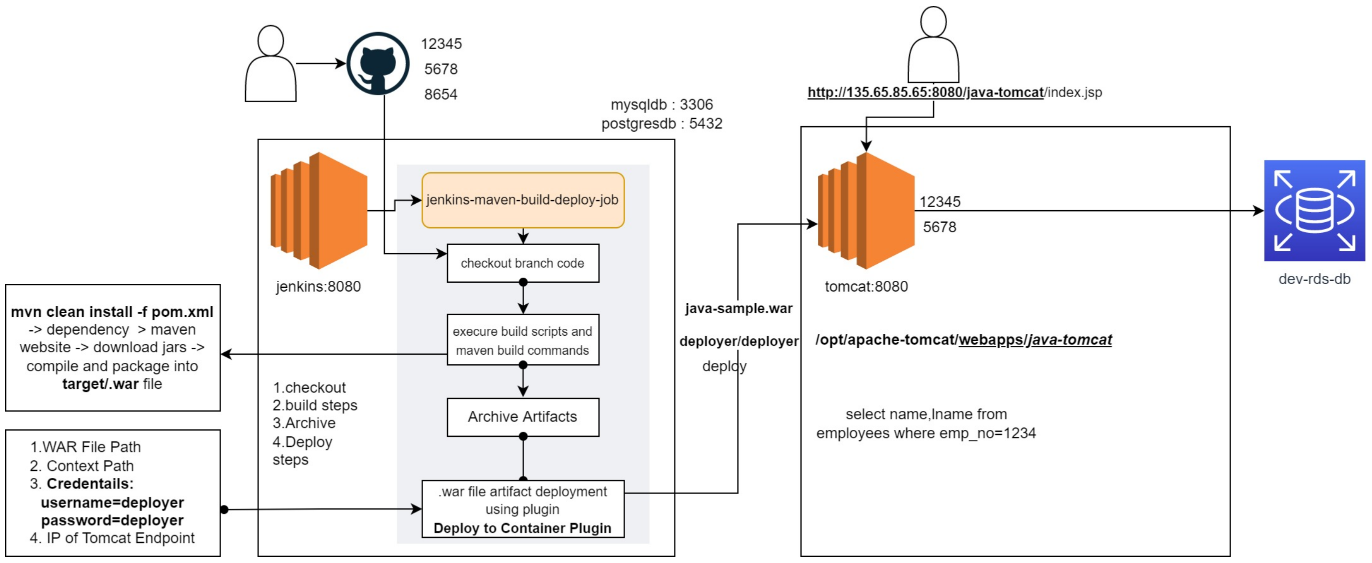
pom.xml definitions

- `<modelVersion>` : POM model version (always 4.0.0).
- `<groupId>` : Group or organization that the project belongs to. Often expressed as an inverted domain name.
- `<artifactId>` : Name to be given to the project's library artifact (for example, the name of its JAR or WAR file).
- `<version>` : Version of the project that is being built.

- `<packaging>` : How the project should be packaged. Defaults to **jar** for JAR file packaging. Use **war** for WAR file packaging.
- `<dependencies>` : External JAR File dependencies

--

Jenkins Build using Jenkins Job


12345
5678
8654
mysqldb : 3306
postgresdb : 5432
http://135.65.85.65:8080/java-tomcat/index.jsp
jenkins-maven-build-deploy-job
checkout branch code
execure build scripts and maven build commands
Archive Artifacts
.war file artifact deployment using plugin
Deploy to Container Plugin
jenkins:8080
mvn clean install -f pom.xml -> dependency > maven website -> download jars -> compile and package into target/.war file
1.WAR File Path
2. Context Path
3. Credentails: username=deployer password=deployer
4. IP of Tomcat Endpoint
1.checkout
2.build steps
3.Archive
4.Deploy steps
java-sample.war
deployer/deployer
deploy
12345
5678
tomcat:8080
/opt/apache-tomcat/webapps/java-tomcat
select name,lname from employees where emp_no=1234
dev-rds-db

--

- Create a Github Repo with **java-tomcat-sample** source code.
- Click on **New Item** then enter an item name, select **Freestyle project**.
- Under **Source Code Management** tab, select Git and then set the Repository URL to point to your GitHub Repository. `https://github.com/YourUserName/repo-name.git`
- Under **Build Environment > Build Step > Select Invoke top-level Maven targets** from dropdown > select the Maven Version that we just created, specify **clean install** under **Goals** Input area.
- Under Advanced tab, specifiy the **pom.xml** file relative path location from git repository.
- Click on `Save` and when you run the build, it will run

```
[test-jenkins-maven-build-freestyle-project] $
/var/lib/jenkins/tools/hudson.tasks.Maven_MavenInstallation/Maven_Local/bin/mvn -f
java-tomcat-sample/pom.xml clean install
```

- Click OK and Build a Job and you will see that a `war file` is created.
- Check the content of the **jar tf java-tomcat-maven-demo.war**

> `clean` -> Deletes `/var/lib/jenkins/workspace/jenkins-maven-build/java-tomcat-sample/target`

- To view the content of **.war** file use : **jar tf java-tomcat-sample-deploy.war**

--

pom.xml parameters

- In the `mvn` command, a parameter can be passed using `mvn clean install "-Dparameter1=value1" "-Dparameter2=value2"`
    - The above values passed in the command can be referenced in `pom.xml` definition with `${parameter1}` and `${parameter2}`.

--

Artifacts Archive

- Go to **Jenkins dashboard -> Jenkins project or build job -> Post-build Actions -> Add post-build action -> Archive the artifacts**:

- Enter details for options in **Archive the artifacts** section:

    - For `Files to archive` enter the Path of the `.war` file like : **java-tomcat-sample/target/*.war**

- `Save` the changes and `Build Now`.

- Check the directories as below to validate above information:

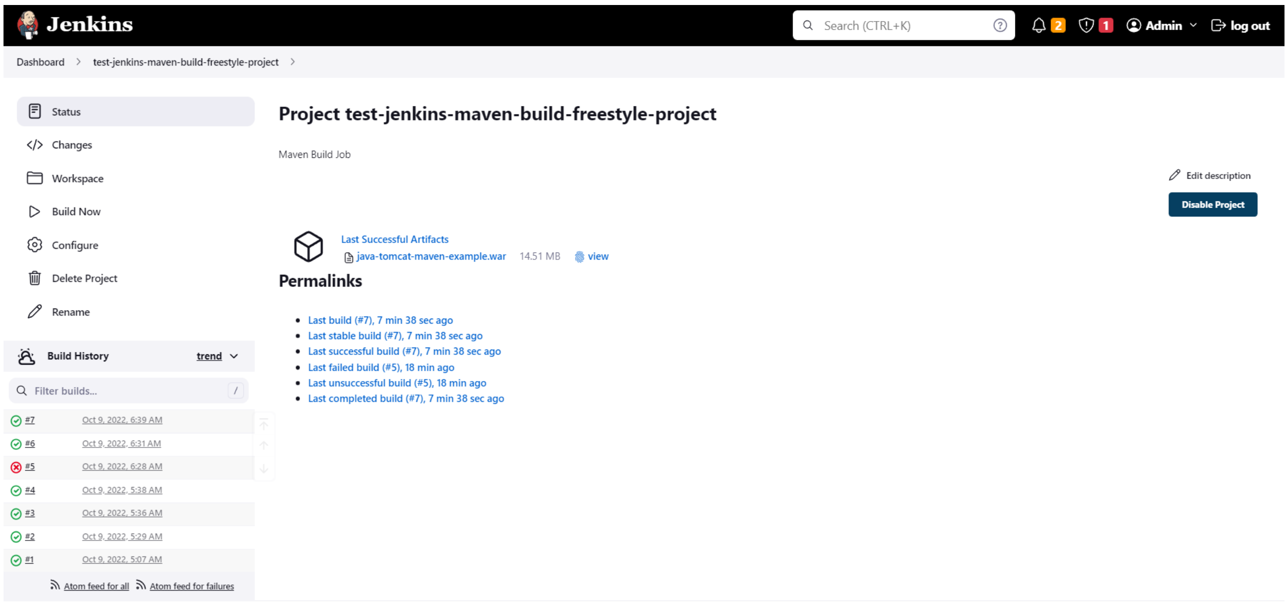
--

```
ls /var/lib/jenkins/jobs
ls /var/lib/jenkins/jobs/<JOB_NAME>
ls /var/lib/jenkins/jobs/<JOB_NAME>/builds/<BUILD_NUMBER>
ls /var/lib/jenkins/workspace/<JOB_NAME>

[ec2-user@ip-172-31-4-240 java-tomcat-sample]$ tree /var/lib/jenkins/jobs/test-
jenkins-maven-build-freestyle-project/
/var/lib/jenkins/jobs/test-jenkins-maven-build-freestyle-project/
├── builds
│   ├── 1
│   │   ├── build.xml
│   │   ├── changelog.xml
│   │   └── log
│   ├── 2
│   │   ├── build.xml
│   │   ├── changelog.xml
│   │   └── log
│   ├── 3
│   │   ├── archive
│   │   │   └── java-tomcat-sample
│   │   │       └── target
│   │   │           └── java-tomcat-maven-example.war
│   │   ├── build.xml
│   │   ├── changelog.xml
│   │   └── log
```

- If you check the directory structure, there will be `archive` directory present under the subsequent build number for which the job is executed with Post build action as `Archive the artifacts`

--

- The Build Project Summary, shows the last successful Build Artifact Name.


Jenkins | Search (CTRL+K) | 2 | 1 | Admin | log out

Dashboard > test-jenkins-maven-build-freestyle-project >

Status
Changes
Workspace
Build Now
Configure
Delete Project
Rename

Project test-jenkins-maven-build-freestyle-project

Maven Build Job

Edit description
Disable Project

Last Successful Artifacts
java-tomcat-maven-example.war 14.51 MB view

Permalinks

- Last build (#7), 7 min 38 sec ago
- Last stable build (#7), 7 min 38 sec ago
- Last successful build (#7), 7 min 38 sec ago
- Last failed build (#5), 18 min ago
- Last unsuccessful build (#5), 18 min ago
- Last completed build (#7), 7 min 38 sec ago

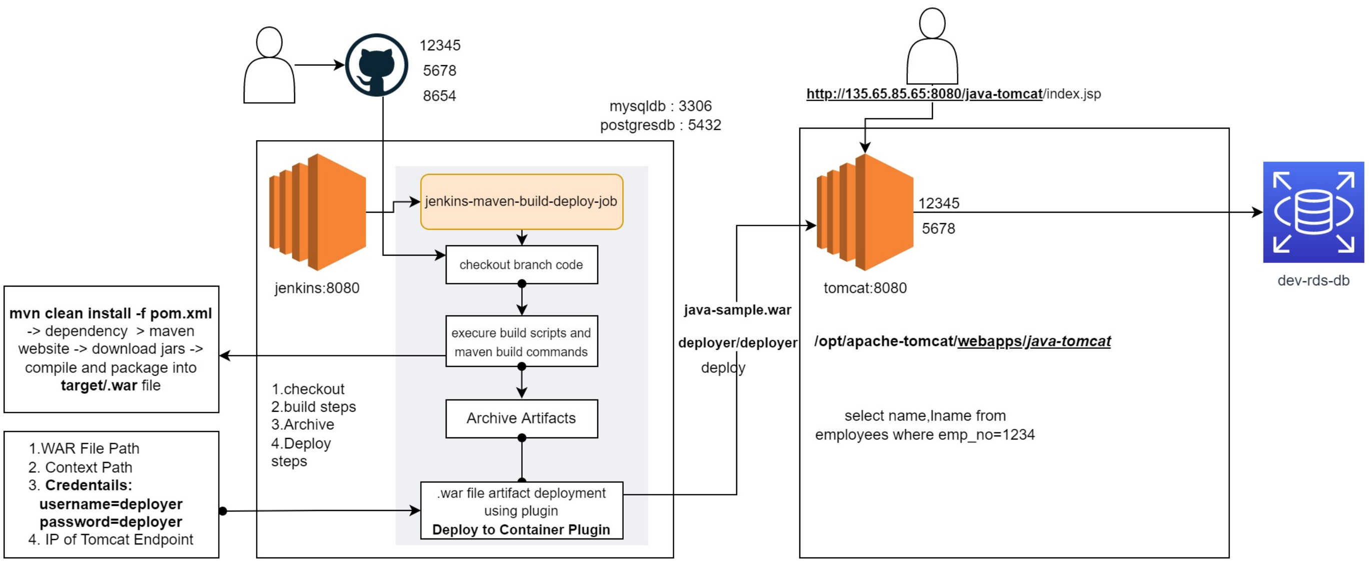
Build History trend
Filter builds...

#7	Oct 9, 2022, 6:39 AM
#6	Oct 9, 2022, 6:31 AM
#5	Oct 9, 2022, 6:28 AM
#4	Oct 9, 2022, 5:38 AM
#3	Oct 9, 2022, 5:36 AM
#2	Oct 9, 2022, 5:29 AM
#1	Oct 9, 2022, 5:07 AM

Atom feed for all   Atom feed for failures

-

Jenkins Build and Deploy


12345
5678
8654
mysqldb : 3306
postgresdb : 5432
http://135.65.85.65:8080/java-tomcat/index.jsp
jenkins-maven-build-deploy-job
checkout branch code
execure build scripts and maven build commands
Archive Artifacts
.war file artifact deployment using plugin Deploy to Container Plugin
jenkins:8080
1.checkout
2.build steps
3.Archive
4.Deploy steps
mvn clean install -f pom.xml -> dependency > maven website -> download jars -> compile and package into target/.war file
1.WAR File Path
2. Context Path
3. Credentails: username=deployer password=deployer
4. IP of Tomcat Endpoint
java-sample.war
deployer/deployer
deploy
12345
5678
tomcat:8080
/opt/apache-tomcat/webapps/java-tomcat
select name,lname from employees where emp_no=1234
dev-rds-db

--

- Below steps assume that, you have a Jenkins Server Up and Running on one of the EC2 instance.

--

Setup Apache Tomcat on Amazon Linux

- Launch a new EC2 Instance with Amazon Linux 2 for Webserver Configuration

```
sudo hostnamectl set-hostname tomcat.example.com
sudo yum install java-1.8.0 java-devel -y
cd /opt/
sudo wget https://archive.apache.org/dist/tomcat/tomcat-9/v9.0.35/bin/apache-
```

```
tomcat-9.0.35.tar.gz
sudo tar -zvxf apache-tomcat-9.0.35.tar.gz
-----------------------------
z – The file is a “gzipped” file
v – Verbose, print the file names as they are extracted one by one
x –  Extract files
f – Use the following tar archive for the operation
-----------------------------
sudo ls -ltr /opt/apache-tomcat-9.0.35/bin
```

- To Start Apache Tomcat : Run the **./startup.sh** file in **/opt/apache-tomcat-9.0.35/bin**
- We can make the scripts executable and then create a symbolic link for this scripts.

```
sudo chmod +x /opt/apache-tomcat-9.0.35/bin/startup.sh
sudo chmod +x /opt/apache-tomcat-9.0.35/bin/shutdown.sh
```

--

- Create symbolic link to these below files so that tomcat server start and stop can be executed from any directory.

```
echo $PATH
sudo ln -s /opt/apache-tomcat-9.0.35/bin/startup.sh /usr/bin/tomcatup
sudo ln -s /opt/apache-tomcat-9.0.35/bin/shutdown.sh /usr/bin/tomcatdown
sudo tomcatup
netstat -nltp | grep 8080
```

> If you want to run Apache Tomcat on same Machine where Jenkins is Installed, then change the port of Apache Tomcat in : **/opt/apache-tomcat-9.0.35/conf/server.xml** file to `8090` as below,

```
 <Connector port="8090" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
```

- If above changes are made, execute the command **tomcatdown** and **tomcatup**.

--

Tomcat War file deployment Configs

Tomcat Users Configuration

- To have access to the dashboard the admin user needs the manager-gui role. Later, we will need to deploy a WAR file using Maven, for this, we need the **manager-script** role too.

- In order for Tomcat to accept remote deployments, we have to add a user with the role **manager-script**. To do so, edit the file **../conf/tomcat-users.xml** and add the following lines:
- In this case : add below configurations in the file ***/opt/apache-tomcat-9.0.35/conf/tomcat-users.xml*** under XML Section of

```
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<user username="admin" password="admin" roles="manager-gui, manager-script"/>
<user username="deployer" password="deployer" roles="manager-script" />
```

```
In XML, lines between '<!-- and -->' are commented lines
```

--

Tomcat Network Configuration

- Edit the **RemoteAddrValve** under this file to allow all.
    - **/opt/apache-tomcat-9.0.35/webapps/manager/META-INF/context.xml**
- `Before`

```
        <Valve className="org.apache.catalina.valves.RemoteAddrValve"
         allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />
```

- `After`

```
        <Valve className="org.apache.catalina.valves.RemoteAddrValve"
         allow=".*" />
```

- Restart the tomcat server using **tomcatdown** and **tomcatup**

--

Apache Tomcat Terminology

- **Document root** : This is the top-level directory of a web application where all the application resources are located, like JSP files, HTML pages, Java classes, and images, artifact libraries.
- **Context path** : This refers to the location that's relative to the server's address and represents the name of the web application. For e.g, If the WAR File is kept under the **$CATALINA_HOME\webapps\myapp** directory, it'll be accessed by the URL **http://TOMCAT_IP:PORT/myapp** , and its context path will be ***/myapp***.
- **WAR** – Web Archive. It's the extension of a file that packages a web application directory hierarchy in ZIP format.

Jenkins Job to deploy war file

Jenkins Plugin installation

- To install the Plugin **Deploy to container** navigate to **Manage Jenkins > Manage Plugins**, search **Deploy to container** under **Available** tab, Install.

- Create a New Job/Clone an existing Job.Click on **New Item** then enter an item name, select **Freestyle project**.

- Select the GitHub project checkbox and set the Project URL to point to your GitHub Repository. **https://github.com/YourUserName/** keep the branch as **main**.

- Go to Jenkins Project -> Configure -> Under Build Environment Build Step > Select `Invoke top-level Maven targets` from dropdown > select the `Maven Version` that is configured > Enter **clean install**

--

- After the **Archive the Artifacts** Post Build Action.
- Under **Post-build Actions**, from the **Add post-build action** dropdown button select the option **Deploy war/ear to a container**
- Enter details of the War file that will be created as:
  - For **WAR/EAR files** you can use wild cards, e.g. `**/*.war`.
  - The `context path` is the part of the URL under which your application will be published in Tomcat. Enter **java-sample-webapp**
  - Select the appropriate Tomcat version from the Container dropdown box (note that you can also deploy to Glassfish or JBoss using this Jenkins plugin).
  - Under the **Credentials**, Add **username and password** value that is entered in the **tomcat-users.xml** file.
  - Specify the ID of the credentials as **tomcat_creds**. This will be used later in Pipeline Script.
  - The Tomcat URL is the base URL through which your Tomcat instance can be reached (e.g `http://172.31.67.85:8080`)
    - `Save` the changes and `Build Now`.

> Make Sure network is open on specific port by checking the Security Group attached to EC2 Instance.

--

- Once Jenkins Job is build, if there is a Success for deploy, verify the deployment files on Tomcat Server under **/opt/apache-tomcat-9.0.35/webapps/** path.
- Access the Application in Browser with specific Home Page present in `src/webapp` i.e **<TOMCAT_SERVER_IP>:<TOMCAT_PORT>/<CONTEXT_PATH>/index.jsp**
- Make some changes in the code on the github configured branch in the Jenkins Job and build the Job again to verify the Artifact Deployment on Tomcat Path.

--

Jenkinsfile Deployment for Build and Deployment

- Create a new Jenkins Pipeline Job and execute the **Jenkinsfile_Maven_Build_Deploy** to run the same Build Steps from Freestyle Project using Jenkinsfile Pipeline Code.
- Modify the necessary Pipeline Code inside the Jenkinsfile.
  - Modify the IP address of the Jenkinsfile Code.

Assignment

Execution of Java Build and Deployment using AWS CICD.

- Configure CodeCodePipeline, CodeBuild, CodeDeploy to use the existing Java Code Files and create a .war file using maven and deploy the .war file on Tomcat Servers.

```
Jenkins:
- Integration with Github.
- Build Step for Maven Command Execution:
    - mvn clean install -f pom.xml
- Archiving the artifacts ( .war file )
- Deploy to Container Plugin:
    - .war file path, tomcat url, context path.

AWS CICD:
Environment Pre-requisite : EC2 instance ( codedeploy agent, tomcat installed )
CodePipeline Integration with Github.
- CodeBuild buildspec.yml command execution:
  - install maven inside buildspec.yml
    - mvn clean install -f pom.xml
    - artifacts: section , .war file to be copied on S3.
- CodeDeploy for .war deployment:
    - appspec.yml
      src: path/to/java-tomcat.war
      dest: /opt/apache-tomcat/webapps/
```

Java Database Connection

- The Java Project Object Model file i.e `pom.xml` file contains `dependency` specified as below:

```
  <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.24</version>
  </dependency>
```

- During the Java Build, the above mentioned dependency packages i.e jar files are downloaded from Maven Website Portal and added inside the **lib** path inside the Artifact **.war** file.
- The **mysql-connector-java-8.0.24.jar** file can be found under **/opt/apache-tomcat-9.0.35/webapps/java-tomcat-sample-deploy/WEB-INF/lib/mysql-connector-java-8.0.24.jar**

- Jar files contain **.class** files already compiled and can be used as **import packages in .java programs**.
- To view the content of the Jar files use command as : **jar tf FILENAME.jar**
  - **jar tf /opt/apache-tomcat-9.0.35/webapps/java-tomcat-sample-deploy/WEB-INF/lib/mysql-connector-java-8.0.24.jar**
- This will display all the **.class** files present.
- The **mysql-connector-java-8.0.24.jar** file is used in the **java-tomcat-sample/src/main/java/dao/GetDao.java** file to import **methods/functions** and use objects to make connection to mysql database.

--

Using Mysql Database Server

Configure Mysql DB AWS RDS

- Go to RDS service and click on create database, Chose standard create, Select Mysql.
- On templates select **Free tier**.
- Enter the DB instance identifier as **java-tomcat-db**
- Keep the username and password as desired.
- For DB instance size, let us keep the default value of db.t3.micro.
- For allocated storage we can keep the default 20 GB as storage.
- The RDS similar to EC2 is launched in the VPC.Under connectivity select the VPC and click on "additional connectivity configuration ".
- Select the subnet group by default.
- Always make sure the "Public Access" setting is always set to No. As we'd never want our database to be accessed by internet.
- Select the security group. Make sure EC2 instance IP is whitelisted on port 3306 on this security group.
- Keep other default configuration same.

--

Connecting to an RDS

- Since we have selected public access as no , this RDS can be only accessed from within a VPC.
- On one of the EC2 machine, install mysql client to connect to RDS.

```
#Install the mysql client on EC2
sudo yum install mysql mysql-connector-java.noarch -y
```

- These commands will install the mysql client which helps us connect to any mysql database
- Now to connect to the RDS that we have created, navigate to the RDS console. Click on the RDS and copy the endpoint.

```
# command to connect to mysql
mysql -h rds-endpoint-name -P 3306 -u username -p
mysql -h java-tomcat-db.cgsel8pj8zsp.ap-south-1.rds.amazonaws.com -P 3306 -u admin
```

```
-p
```

--

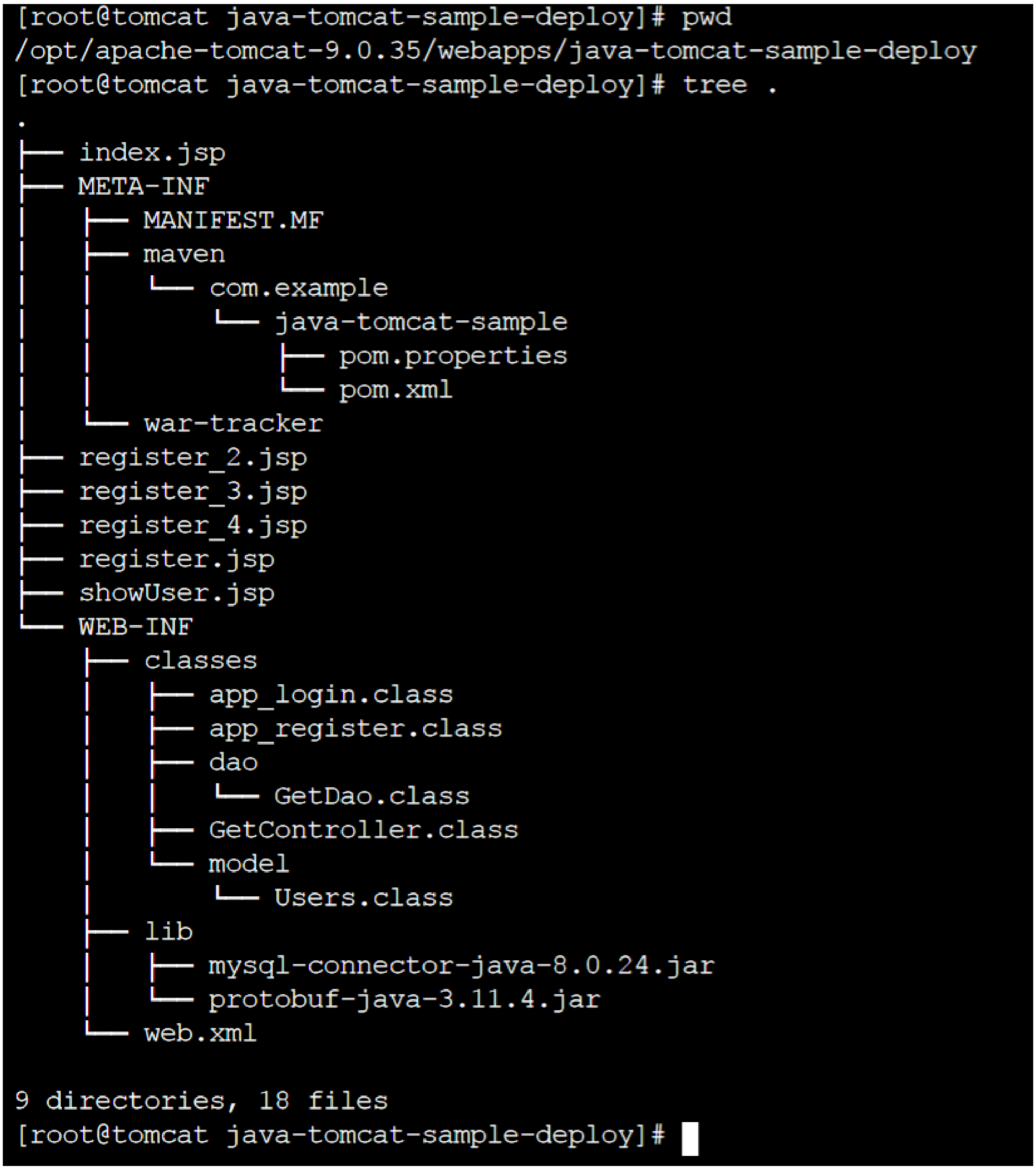
- Use below mysql commands to create database and tables.

```
mysql> show databases;
mysql> create database db;
mysql> use db;

# Create employees table
CREATE TABLE employees (
    emp_no      INT             NOT NULL,
    birth_date  DATE            NOT NULL,
    first_name  VARCHAR(14)     NOT NULL,
    last_name   VARCHAR(16)     NOT NULL,
    gender      ENUM ('M','F')  NOT NULL,
    hire_date   DATE            NOT NULL,
    PRIMARY KEY (emp_no)
);

show tables;
desc employees;

select * from employees;

INSERT INTO `employees` VALUES (10001,'1953-09-02','Georgi','Facello','M','1986-
06-26'),
(10002,'1964-06-02','Bezalel','Simmel','F','1985-11-21'),
(10003,'1959-12-03','Parto','Bamford','M','1986-08-28'),
(10004,'1954-05-01','Chirstian','Koblick','M','1986-12-01'),
(10005,'1955-01-21','Kyoichi','Maliniak','M','1989-09-12'),
(10006,'1953-04-20','Anneke','Preusig','F','1989-06-02'),
(10007,'1957-05-23','Tzvetan','Zielinski','F','1989-02-10'),
(10008,'1958-02-19','Saniya','Kalloufi','M','1994-09-15'),
(10009,'1952-04-19','Sumant','Peac','F','1985-02-18'),
(10010,'1963-06-01','Duangkaew','Piveteau','F','1989-08-24');

# Create Title Table
CREATE TABLE titles (
    emp_no      INT             NOT NULL,
    title       VARCHAR(50)     NOT NULL,
    from_date   DATE            NOT NULL,
    to_date     DATE,
    # FOREIGN KEY (emp_no) REFERENCES employees (emp_no) ON DELETE CASCADE,
    PRIMARY KEY (emp_no,title, from_date)
);

INSERT INTO `titles` VALUES (10001,'Senior Engineer','1986-06-26','9999-01-01'),
(10002,'Staff','1996-08-03','9999-01-01'),
(10003,'Senior Engineer','1995-12-03','9999-01-01'),
(10004,'Engineer','1986-12-01','1995-12-01'),
(10004,'Senior Engineer','1995-12-01','9999-01-01'),
(10005,'Senior Staff','1996-09-12','9999-01-01'),
(10005,'Staff','1989-09-12','1996-09-12'),
(10006,'Senior Engineer','1990-08-05','9999-01-01'),
(10007,'Senior Staff','1996-02-11','9999-01-01'),
(10007,'Staff','1989-02-10','1996-02-11'),
(10008,'Assistant Engineer','1998-03-11','2000-07-31'),
(10009,'Assistant Engineer','1985-02-18','1990-02-18'),
(10009,'Engineer','1990-02-18','1995-02-18'),
(10009,'Senior Engineer','1995-02-18','9999-01-01'),
(10010,'Engineer','1996-11-24','9999-01-01');

select * from titles;

SELECT employees.emp_no,employees.first_name, employees.last_name,
employees.hire_date, titles.title, titles.from_date, titles.to_date FROM employees
left JOIN titles ON employees.emp_no=titles.emp_no;
```

--

- Once above DB and Tables are created, validate the details of the **DB Hostname, Database, DB UserName, DB Password values** in file **java-tomcat-sample/src/main/java/dao/GetDao.java**
- Modify the above DB Password that is set in the above file and build and deploy artifact WAR File again.
- Execute the Jenkins Job to build above artifact and deploy the same on the Webserver Context Path.
- A directory with name of the `.war` file is present on the `webapps` path.

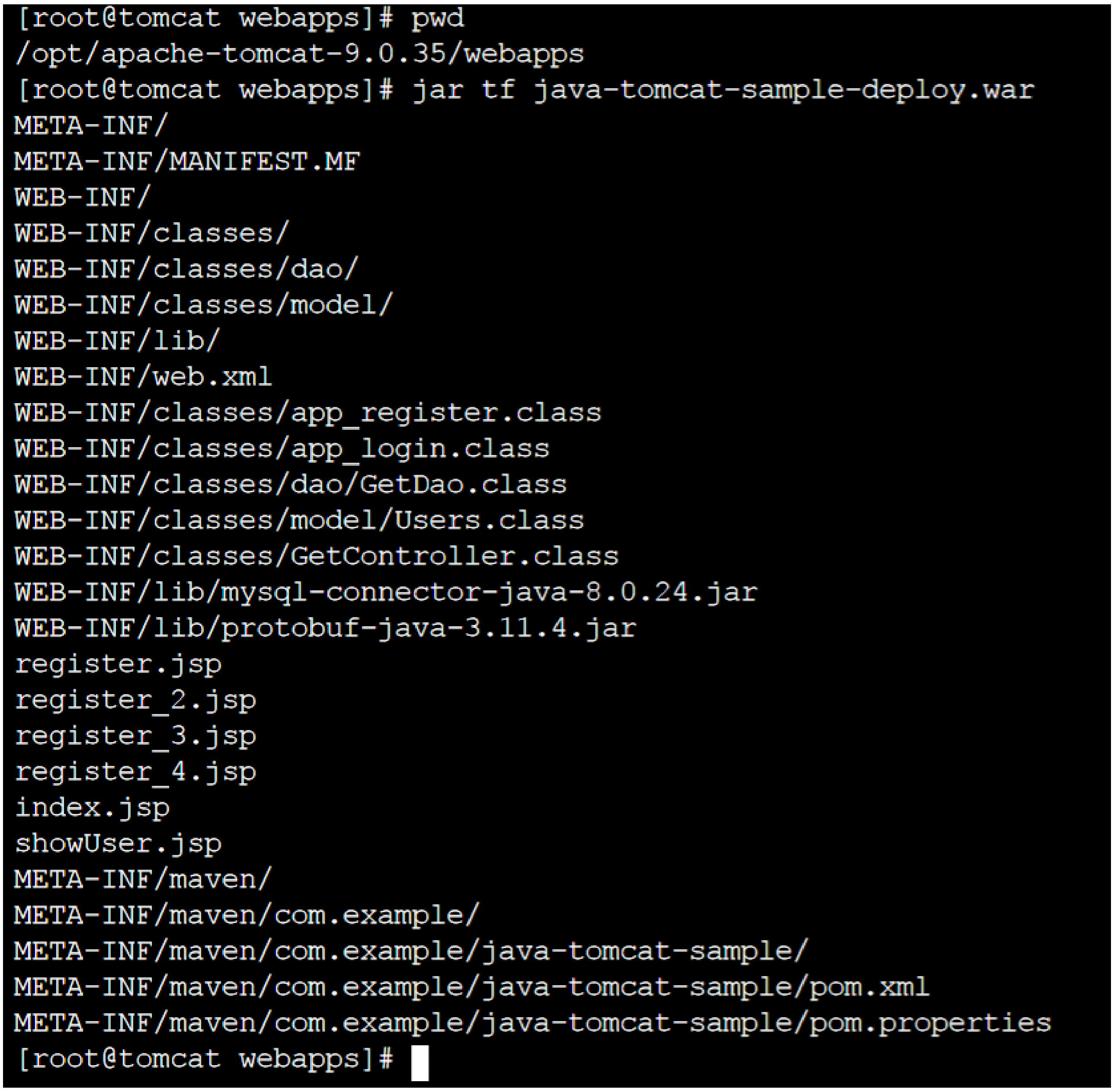
--

- 

```
[root@tomcat java-tomcat-sample-deploy]# pwd
/opt/apache-tomcat-9.0.35/webapps/java-tomcat-sample-deploy
[root@tomcat java-tomcat-sample-deploy]# tree .
.
├── index.jsp
├── META-INF
│   ├── MANIFEST.MF
│   ├── maven
│   │   └── com.example
│   │       └── java-tomcat-sample
│   │           ├── pom.properties
│   │           └── pom.xml
│   └── war-tracker
├── register_2.jsp
├── register_3.jsp
├── register_4.jsp
├── register.jsp
├── showUser.jsp
└── WEB-INF
    ├── classes
    │   ├── app_login.class
    │   ├── app_register.class
    │   ├── dao
    │   │   └── GetDao.class
    │   ├── GetController.class
    │   └── model
    │       └── Users.class
    ├── lib
    │   ├── mysql-connector-java-8.0.24.jar
    │   └── protobuf-java-3.11.4.jar
    └── web.xml

9 directories, 18 files
[root@tomcat java-tomcat-sample-deploy]# 
```

--

- To view the content of `.war` file use : `jar tf java-tomcat-sample-deploy.war`

- 

```
[root@tomcat webapps]# pwd
/opt/apache-tomcat-9.0.35/webapps
[root@tomcat webapps]# jar tf java-tomcat-sample-deploy.war
META-INF/
META-INF/MANIFEST.MF
WEB-INF/
WEB-INF/classes/
WEB-INF/classes/dao/
WEB-INF/classes/model/
WEB-INF/lib/
WEB-INF/web.xml
WEB-INF/classes/app_register.class
WEB-INF/classes/app_login.class
WEB-INF/classes/dao/GetDao.class
WEB-INF/classes/model/Users.class
WEB-INF/classes/GetController.class
WEB-INF/lib/mysql-connector-java-8.0.24.jar
WEB-INF/lib/protobuf-java-3.11.4.jar
register.jsp
register_2.jsp
register_3.jsp
register_4.jsp
index.jsp
showUser.jsp
META-INF/maven/
META-INF/maven/com.example/
META-INF/maven/com.example/java-tomcat-sample/
META-INF/maven/com.example/java-tomcat-sample/pom.xml
META-INF/maven/com.example/java-tomcat-sample/pom.properties
[root@tomcat webapps]#
```

--

- Also, validate the details retured in the WebPage from Database.
  - Enter the Employee ID details in the UI and check if you are getting response from the values in the Database Tables.

> Once Environment setup and deployment is working, delete all AWS Resources created to avoid cost in AWS Account.

Multiple Environment Configuration

- In Application Code, the details that will change as per Environments (**dev/qa/prod**), those config changes, should not be hardcoded in the program code.
- No cross environment resources should be connecting or accessing each other.

SSM Parameter Store.

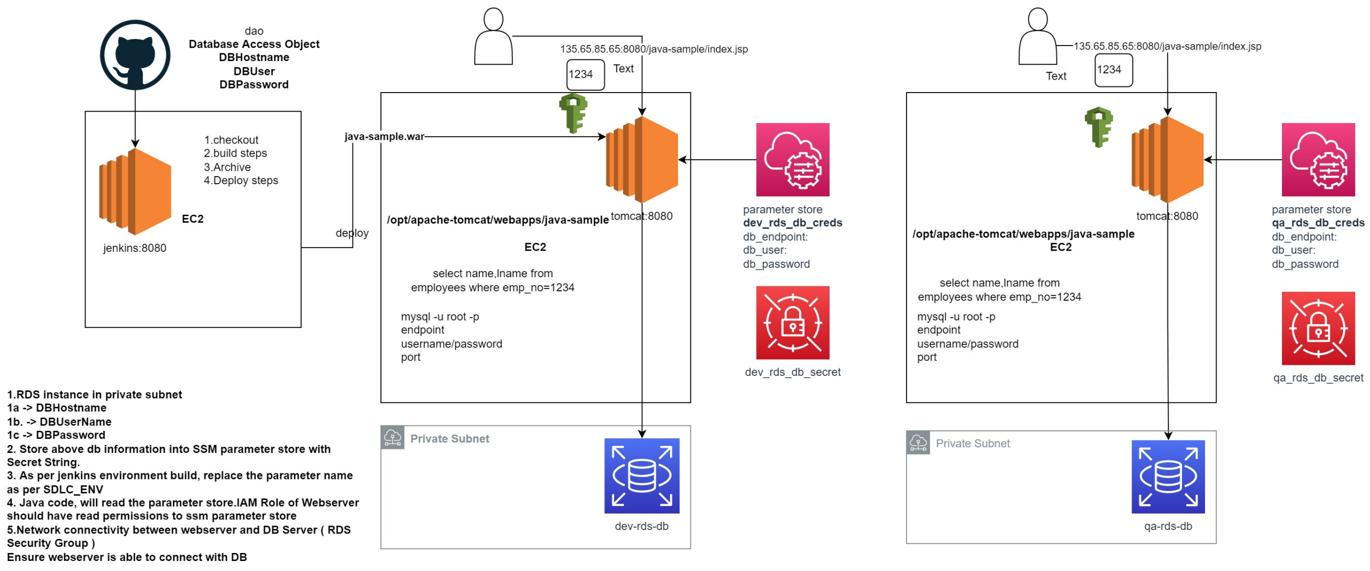
- Create Parameter for each environment **qa_db_rds_password** as **SecureString** Type, store the DB Password.
- In Java, there is **aws-sdk**, add the dependency in your **pom.xml** file.
- Dynamically as per users input for specific deployment
  - **sh "sed -i /s/SDLC_ENV/${params.SDLC_ENV}/ java-tomcat-sample/src/main/java/dao/GetDao.java"** , during the build and packaging, the code is replaced with correct environment variables, so that, credentials are retrieved specifically for that environment.

--

AWS Secrets Manager

- Create a secret in AWS Secret Manager Service and store the DB Endpoint, Username, Password.
- In Java, there is `aws-sdk`, add the dependency in your pom.xml file.
- Dynamically as per users input for specific deployment, **sh "sed -i /s/SDLC_ENV/${params.SDLC_ENV}/ java-tomcat-sample/src/main/java/dao/GetDao.java"** , during the build and packaging, the code is replaced with correct environment variables, so that, credentials are retrieved specifically for that environment.

--



1.RDS instance in private subnet
1a -> DBHostname
1b. -> DBUserName
1c -> DBPassword
2. Store above db information into SSM parameter store with Secret String.
3. As per jenkins environment build, replace the parameter name as per SDLC_ENV
4. Java code, will read the parameter store.IAM Role of Webserver should have read permissions to ssm parameter store
5.Network connectivity between webserver and DB Server (RDS Security Group)
Ensure webserver is able to connect with DB

--

Infra Creation using IAC-Assignment

- Tomcat Server EC2 instances, VPC, IAM Roles, Secrets, RDS all the resources required for application code to run, should be created using IAC framework.
  - Terraform
  - CloudFormation

> Use Terraform modules for infra creation as per SDLC Environments.

--

Execution of Java Build and Deployment using AWS CICD.

- Configure CodePipeline, CodeBuild, CodeDeploy to use the existing Java Code Files and create a .war file using maven and deploy the .war file on Tomcat Servers.

```
Jenkins:
- Integration with Github.
- Build Step for Maven Command Execution:
    - mvn clean install -f pom.xml
- Archiving the artifacts ( .war file )
- Deploy to Container Plugin:
    - .war file path, tomcat url, context path.

AWS CICD:
Environment Pre-requisite : EC2 instance ( codedeploy agent, tomcat installed )
CodePipeline Integration with Github.
- CodeBuild buildspec.yml command execution:
  - install maven inside buildspec.yml
    - mvn clean install -f pom.xml
    - artifacts: section , .war file to be copied on S3.
- CodeDeploy for .war deployment:
    - appspec.yml
        src: path/to/java-tomcat.war
        dest: /opt/apache-tomcat/webapps/
```

--

Jenkins Backup

- Taking backups of your Jenkins Server can be important to ensure that you can recover your configuration, jobs, and data in case of system failures
- Jenkins provides several ways to take backups, including manual and automated methods. Here's how you can take backups in Jenkins:

Manual Backup

- **Backup Jobs:**
    - For **Freestyle and Declarative Pipeline jobs**, manually copy the job configurations from the Jenkins web interface.
    - For Pipeline jobs defined in a **Jenkinsfile**, ensure your pipeline scripts are versioned in your VCS (Github)
- **Backup Data Directory:**
    - The Jenkins data directory i.e **/var/lib/jenkins** contains important files and configurations.
    - Create a backup of this directory, including the jobs directory, plugins directory, and other configuration files.

--

Automated Backup

- **ThinBackup Plugin:**
    - Install the **ThinBackup** plugin from the Jenkins Plugin Manager.
    - Configure the plugin to schedule automated backups of your Jenkins instance.
    - Backups can be stored locally, on a remote server, or in cloud storage.
- **Jenkins Backup to Amazon S3:**
    - If you are using AWS, you can use the **Jenkins Backup to Amazon S3** plugin to automatically back up your Jenkins configuration and data to an S3 bucket.

--

Scripted Backup

- You can write custom scripts to automate the backup process, including copying the data directory, job configurations, and other important files to a backup location.

Important Considerations:

- Test your backup and restoration process in a some lower non-production environment to ensure it works as expected.
- Store backups securely, preferably in an offsite location or cloud storage.
- Regularly review and update backup strategies based on changes in your Jenkins configuration and infrastructure.
- Keep backup procedures and documentation up to date.
- Remember that backups are a critical part of disaster recovery planning, so ensure that you have a well-defined backup strategy that aligns with your organization's needs and requirements.

--

Working with Jenkins

- Jenkins Installation
- Jenkins Configuration
    - Jenkins Credentials
    - Jenkins Plugins
    - Jenkins User Authorization
    - Jenkins Audit trail
- Jenkins Job Creation:
    - Freestyle Job using Git integration
    - Pipeline Job
        - Writing Jenkinsfile
        - Multi Environment Deployment
- Application Build Scenario:
    - Maven Java
        - pom.xml definition
- CICD Comparison

- Infrastructure Creation using Terraform with AWS CICD and Jenkins
- Docker Image Creation Pipeline In Jenkins.
  - Image Creation shell script can be executed from Jenkins Freestyle Job or Jenkinsfile
- AWS CICD for Maven Build and Deployment.

--

Troubleshooting

- Troubleshooting in Jenkins involves identifying & resolving issues that can occur during the build & deployment processes. Below are the some common troubleshooting scenarios, along with examples & solutions:

- **1. Jenkins Job Fails to Start:**
  - **Scenario:** A Jenkins job fails to start, & you're not sure why.
  - **Solution:**

  1. Check the job's configuration for syntax errors or misconfigured settings.
  2. Review the console output for error messages that indicate the cause of the failure.
  3. Ensure that any required plugins are installed & up to date.

2. Build Errors: Scenario: A build step within a Jenkins job fails. **Solution:**

1. Review the console output for error messages or stack traces indicating the cause of the failure.
2. Verify that the build environment has the required tools & dependencies installed.
3. Check for issues related to source code, permissions, or file paths.

--

3. Job Stuck or Hanging: Scenario: A Jenkins job appears to be stuck or hanging without making progress. **Solution:**

1. Monitor the job's console output for any signs of activity or log messages.
2. Check if any resources (e.g., agents, external systems) required by the job are unavailable or experiencing issues.
3. Increase the timeout settings for build steps if applicable.

- **4. Git Checkout Failures: Scenario:** The job fails during the code checkout step from Git **Solution:**

1. Check the repository URL, Credentials, & branch/tag settings in the job's configuration.
2. Verify that the Jenkins agent running the job has access to the Git repository.
3. Validate the clone url & type of authentication used for **https** & **ssh**.

--

5. Plugin Compatibility Problems: Scenario: The job fails due to incompatibility issues with a plugin. **Solution:**

1. Review the plugin versions & check if they're compatible with your Jenkins version.
2. Update the plugin to a version that's compatible with your Jenkins version.
3. Disable or remove plugins that are causing conflicts or compatibility issues.

7. Insufficient Disk Space: Scenario: The job fails due to insufficient disk space on the Jenkins master or agent. **Solution:**

1. Check disk usage on the machine hosting Jenkins.
2. Clean up unnecessary files or artifacts to free up space.
3. Consider increasing disk space or adding additional storage if required.

8. Configuration & Credential Issues: Scenario: The job fails due to incorrect or missing configuration settings. **Solution:**

1. Double-check the job's configuration for accuracy, including URLs, paths, & credentials.
2. Use Jenkins credential management to securely store & provide credentials to jobs.

> Note : Effective troubleshooting often involves a systematic approach of isolating & identifying the root cause of the issue. Check logs, console outputs, configurations, & relevant documentation to diagnose & resolve problems. If you're unable to solve a problem on your own, don't hesitate to seek assistance from the Jenkins community or your organization's support channels.

Jenkins Do's & Dont's

- As Jenkins is a widely used open-source automation server that facilitates the process of building, testing, & deploying software projects. Following best practices with Jenkins helps ensure a smooth & efficient software development pipeline. Here are some do's & don'ts that are commonly followed in the industry:

Do's

Action	Do	Reason
Version Control Integration:	Integrate Jenkins with your version control system (e.g., Git) to trigger builds automatically when changes are pushed.	This ensures that every code change is built & tested, maintaining code quality & catching issues early.
Pipeline as Code:	Define your build & deployment pipelines using code (e.g., Jenkinsfile for Jenkins pipelines).	This approach allows versioning, code review, & simplifies pipeline maintenance.
Environment Isolation:	Use isolated environments for different stages of the pipeline (development, testing, production).	This prevents interference between stages & helps maintain consistent testing environments.

--

Dont's

Action	Dont's	Reason
Manual Steps:	Rely on manual interventions within your pipeline.	Manual steps introduce delays & potential human errors.

Action	Dont's	Reason
Complex Pipelines:	Create overly complex pipelines that are hard to understand & maintain.	Complexity hinders troubleshooting & increases the likelihood of errors.
Hardcoded Credentials:	Store credentials or sensitive information directly in pipeline scripts.	Hardcoding credentials poses a security risk. Use credential management solutions.
Inadequate Error Handling:	Neglect error handling & notification mechanisms.	Proper error handling ensures issues are promptly addressed & teams are informed.
Ignoring Monitoring & Logs:	Neglect monitoring & logging of your Jenkins server & pipelines.	Monitoring helps identify performance bottlenecks & abnormal behavior.

> Note : These best practices may vary depending on the specific needs & technologies of your organization. Regularly reviewing & updating your Jenkins practices based on lessons learned is crucial to maintaining an efficient & effective development process.