Demo Project: CD - Deploy Application from Jenkins Pipeline on EC2 Instance (automatically with docker-compose)

This guide demonstrates how to:

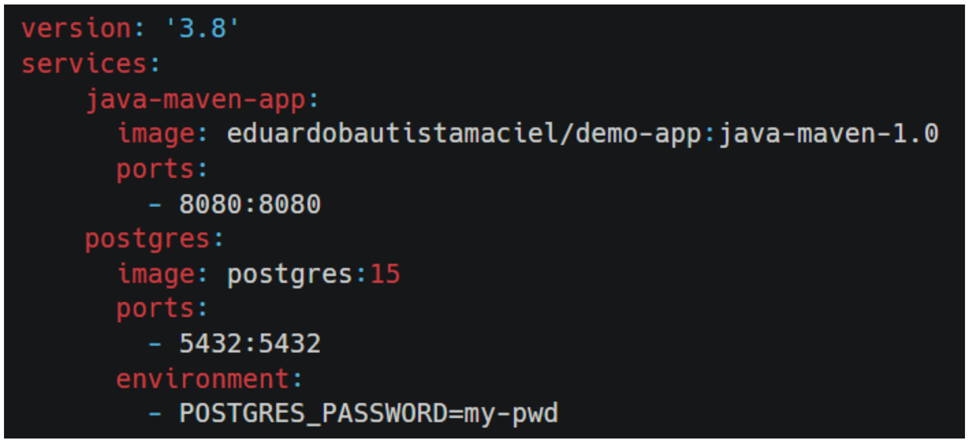
1. Install Docker Compose on an EC2 instance.
2. Create and use a `docker-compose.yaml` file for a multi-service deployment.
3. Extend the Jenkins pipeline to deploy applications on the EC2 instance using Docker Compose.
4. Enhance the pipeline to handle parameterized Docker images.
5. Test the pipeline and deployment process.

Step 1: Install docker-compose on EC2 Instance

1. **On EC2 server install Docker compose:**
    - `sudo curl -L https://github.com/docker/compose/releases/latest/download/docker-compose-$ (uname -s)-$(uname -m) -o /usr/local/bin/docker-compose`
    - `sudo chmod +x /usr/local/bin/docker-compose`
2. **Verify installation:** `docker-compose --version`

Step 2: Create docker-compose.yaml file

1. **Prepare the EC2 Instance**:
    - Remove any existing containers: `docker ps`
    - Stop and remove all running containers if necessary.
2. **Switch to the `jenkins-jobs` branch.**
3. **Create the `docker-compose.yaml`: Define a `docker-compose.yaml` file for a Java Maven application and a PostgreSQL database:**

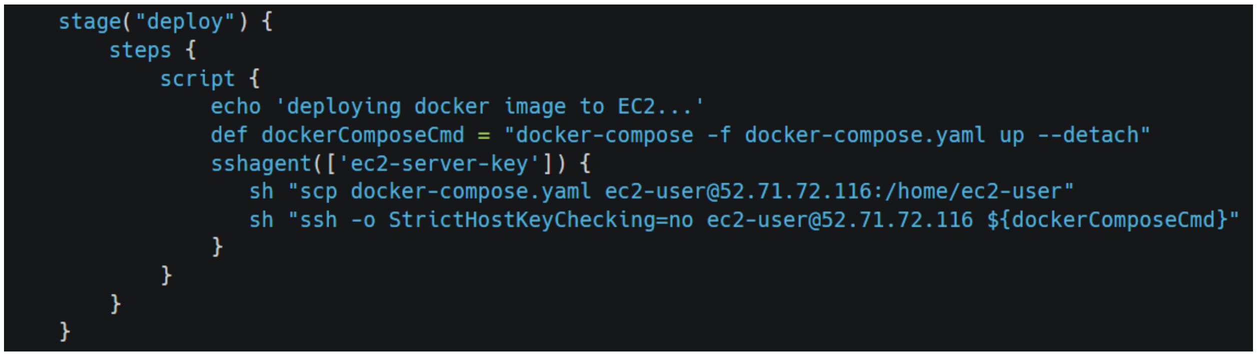


```yaml
version: '3.8'
services:
    java-maven-app:
      image: eduardobautistamaciel/demo-app:java-maven-1.0
      ports:
        - 8080:8080
    postgres:
      image: postgres:15
      ports:
        - 5432:5432
      environment:
        - POSTGRES_PASSWORD=my-pwd
```

Step 3: Adjust Jenkinsfile to Deploy with Docker Compose

Update the Jenkinsfile to execute the Docker Compose deployment on the EC2 instance.

Jenkinsfile Example:



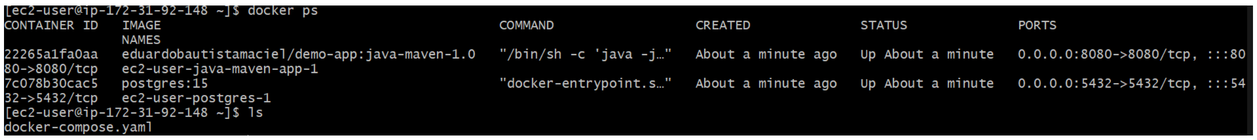
```
    stage("deploy") {
        steps {
            script {
                echo 'deploying docker image to EC2...'
                def dockerComposeCmd = "docker-compose -f docker-compose.yaml up --detach"
                sshagent(['ec2-server-key']) {
                   sh "scp docker-compose.yaml ec2-user@52.71.72.116:/home/ec2-user"
                   sh "ssh -o StrictHostKeyChecking=no ec2-user@52.71.72.116 ${dockerComposeCmd}"
                }
            }
        }
    }
```

Explanation:

- `scp docker-compose.yaml`: Copies the `docker-compose.yaml` file to the EC2 instance.
- `docker-compose up`: Brings up the services in the background (`-detach`).

Step 4: Test and Confirm

1. **Run the Pipeline**:
    - Trigger the Jenkins pipeline and monitor logs for a successful run.
2. **Verify on EC2**:
    - Check running containers: `docker ps`
    - Confirm the `docker-compose.yaml` file exists on the EC2 instance: `ls`



```
[ec2-user@ip-172-31-92-148 ~]$ docker ps
CONTAINER ID   IMAGE                                       COMMAND                  CREATED              STATUS              PORTS
               NAMES
22265a1fa0aa   eduardobautistamaciel/demo-app:java-maven-1.0   "/bin/sh -c 'java -j…"   About a minute ago   Up About a minute   0.0.0.0:8080->8080/tcp, :::80
80->8080/tcp   ec2-user-java-maven-app-1
7c078b30cac5   postgres:15                                 "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:5432->5432/tcp, :::54
32->5432/tcp   ec2-user-postgres-1
[ec2-user@ip-172-31-92-148 ~]$ ls
docker-compose.yaml
```

Step 5: Improvement: Extract to Shell Script

Why This is Necessary:

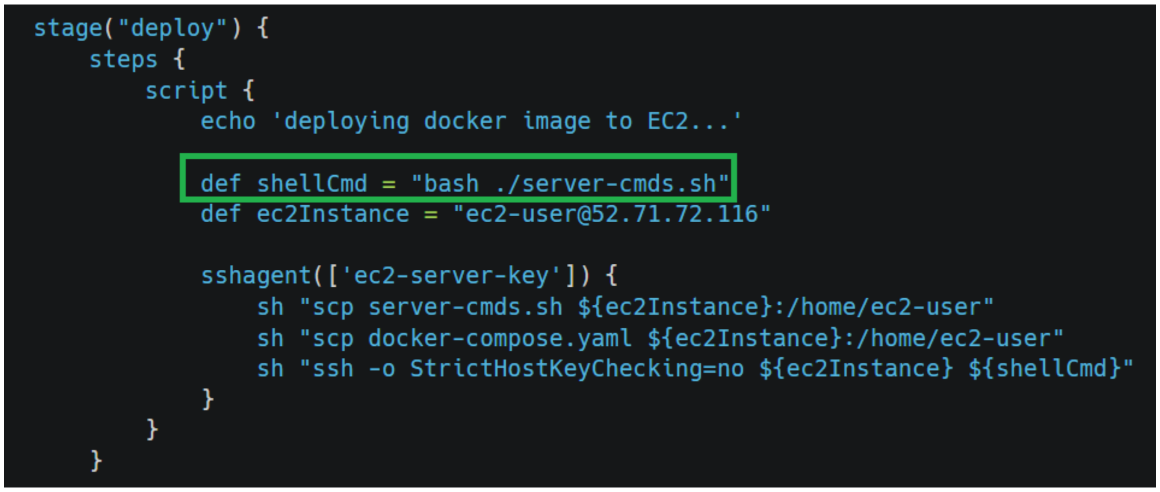
We can execute multiple commands, also setting variables by grouping them into a shell script and then executing a shell script from the Jenkinsfile.

1. Create a Shell Script (`server-cmds.sh`):

```
#!/usr/bin/env bash

docker-compose -f docker-compose.yaml up --detach
echo "success"
```

2. Update the Jenkinsfile:

Update the deploy stage to execute the shell script on the EC2 instance:

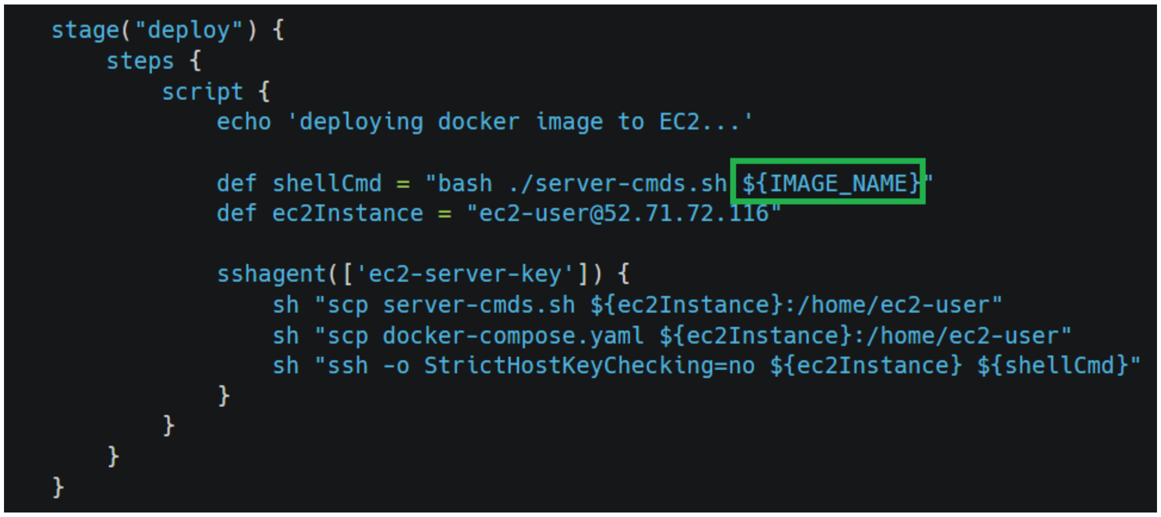


```
stage("deploy") {
    steps {
        script {
            echo 'deploying docker image to EC2...'

            def shellCmd = "bash ./server-cmds.sh"
            def ec2Instance = "ec2-user@52.71.72.116"

            sshagent(['ec2-server-key']) {
                sh "scp server-cmds.sh ${ec2Instance}:/home/ec2-user"
                sh "scp docker-compose.yaml ${ec2Instance}:/home/ec2-user"
                sh "ssh -o StrictHostKeyChecking=no ${ec2Instance} ${shellCmd}"
            }
        }
    }
```

Step 6: Improvement: Replace Docker Image with newly build version

1. Update Jenkinsfile:

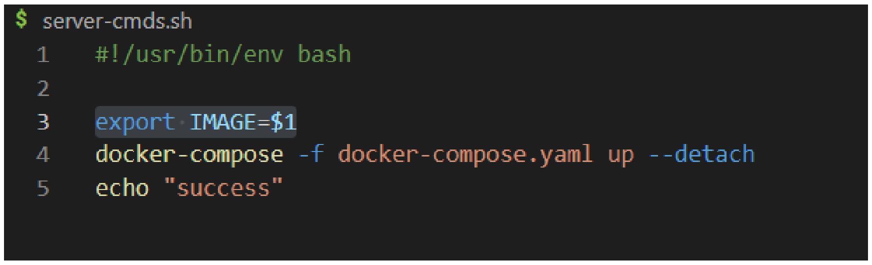
Pass the Docker image name and tag as a parameter to the shell script:



```
stage("deploy") {
    steps {
        script {
            echo 'deploying docker image to EC2...'

            def shellCmd = "bash ./server-cmds.sh ${IMAGE_NAME}"
            def ec2Instance = "ec2-user@52.71.72.116"

            sshagent(['ec2-server-key']) {
                sh "scp server-cmds.sh ${ec2Instance}:/home/ec2-user"
                sh "scp docker-compose.yaml ${ec2Instance}:/home/ec2-user"
                sh "ssh -o StrictHostKeyChecking=no ${ec2Instance} ${shellCmd}"
            }
        }
    }
}
```

2. Update the Shell Script:

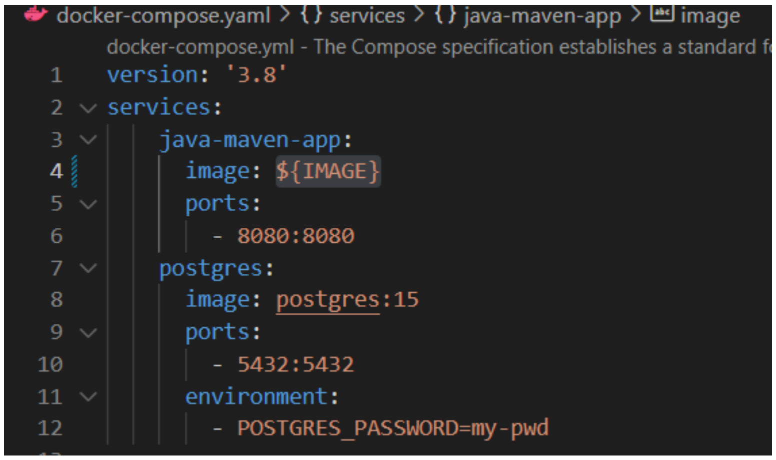
Modify the script to accept the Docker image name as an argument and update the `docker-compose.yaml` file dynamically:



```
$ server-cmds.sh
#!/usr/bin/env bash

export IMAGE=$1
docker-compose -f docker-compose.yaml up --detach
echo "success"
```

3. Update `docker-compose.yaml`:

Replace the hardcoded image name with a placeholder:



```
docker-compose.yaml > {} services > {} java-maven-app > image
docker-compose.yml - The Compose specification establishes a standard f
version: '3.8'
services:
  java-maven-app:
    image: ${IMAGE}
    ports:
      - 8080:8080
  postgres:
    image: postgres:15
    ports:
      - 5432:5432
    environment:
      - POSTGRES_PASSWORD=my-pwd
```

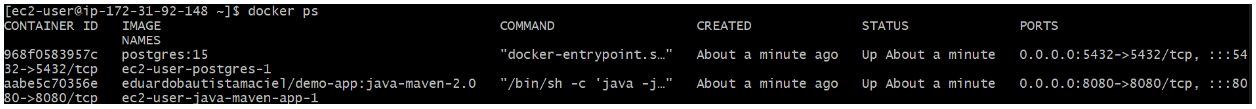
Step 7: Testing

1. **Update the Image Version**: Update the `IMAGE_NAME` environment variable in the Jenkinsfile to the new version:

```
environment {
    IMAGE_NAME = 'eduardobautistamaciel/demo-app:java-maven-2.0'
}
```

2. **Commit and Push**
3. **Clean Existing Containers**: In the server remove all existing containers to prepare for testing:

   `docker-compose -f docker-compose.yaml down`
4. **Run the Pipeline**: Trigger the pipeline and ensure the correct version is deployed.
5. **Verify on EC2**:
   - Check running containers `docker ps`
   - Confirm the newly deployed image version is correct:



```
[ec2-user@ip-172-31-92-148 ~]$ docker ps
CONTAINER ID   IMAGE                                            COMMAND                  CREATED              STATUS              PORTS
               NAMES
968f0583957c   postgres:15                                      "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:5432->5432/tcp, :::54
32->5432/tcp   ec2-user-postgres-1
aabe5c70356e   eduardobautistamaciel/demo-app:java-maven-2.0    "/bin/sh -c 'java -j…"   About a minute ago   Up About a minute   0.0.0.0:8080->8080/tcp, :::80
80->8080/tcp   ec2-user-java-maven-app-1
```