Docker Compose: From Installation to Deployment

1. Introduction to Docker Compose

Docker Compose is a tool used for defining and running multi-container Docker applications. It allows you to configure services, networks, and volumes in a `docker-compose.yml` file and manage them with simple commands.

---

2. Installation of Docker Compose

Step 1: Install Docker

- Download and install Docker Desktop from Docker's official website.

Verify installation using:
sh
Copy code
```
docker --version
```

-

Step 2: Install Docker Compose

For Linux users, install Docker Compose using:

sh
Copy code
```
sudo curl -L
"https://github.com/docker/compose/releases/latest/download/docker-com
pose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

Verify installation:

sh
Copy code
```
docker-compose --version
```

---

3. Writing a Docker Compose File

A `docker-compose.yml` file defines services, networks, and volumes. Example:

yaml
Copy code
```
version: '3.8'
services:
  web:
    image: my-web-app
    ports:
      - "80:80"
  db:
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD: example
```

---

4. Running Docker Compose

Step 1: Start Services

Run the following command to start the defined services:

sh
Copy code
```
docker-compose up -d
```

This starts the containers in detached mode (`-d`).

Step 2: Check Running Containers

sh
Copy code

```
docker-compose ps
```

Step 3: Access a Running Container

sh
Copy code

```
docker exec -it <container_name> /bin/bash
```

Step 4: Stop and Remove Containers

sh
Copy code

```
docker-compose down
```

---

5. Continuous Deployment with Jenkins

- The second screenshot suggests a Jenkins pipeline that builds and deploys a Dockerized application.
- It pulls a `tomcat` image, builds the application, and deploys it via Docker.

---

6. Conclusion

This guide provides a step-by-step approach to installing, configuring, and deploying applications using Docker Compose.

```
ananth@DESKTOP-LQBQG2D:~$ docker-compose up -d
Creating network "ananth_default" with the default driver
Pulling web (ananth1501/simpleapplication:)...
latest: Pulling from ananth1501/simpleapplication
5a7813e071bf: Pull complete
8dbbbc6af9dc: Pull complete
a10b6847b9f1: Pull complete
dcc1c5ea3c7d: Pull complete
91e6cc55403a: Pull complete
bbbc5226420a: Pull complete
4f4fb700ef54: Pull complete
865114aeca46: Pull complete
429752ed5e84: Pull complete
Digest: sha256:1e7a64d4f2b24daf27d5154454704d07790c826c99f18266a1cb586f61d8f873
Status: Downloaded newer image for ananth1501/simpleapplication:latest
Creating ananth_web_1 ... done
Creating ananth_db_1  ... done
ananth@DESKTOP-LQBQG2D:~$ docker exec -it ananth_db_1 /bin/bash
bash-5.1# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 9.2.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> exit
Bye
bash-5.1# exit
exit
ananth@DESKTOP-LQBQG2D:~$ docker-compose images
 Container             Repository            Tag       Image Id      Size
------------------------------------------------------------------------
ananth_db_1    mysql                         latest   fa262c3a6564   797 MB
ananth_web_1   ananth1501/simpleapplication   latest   66bf48d2d3b0   520 MB
ananth@DESKTOP-LQBQG2D:~$ docker-compose ps
    Name                 Command             State                    Ports
-------------------------------------------------------------------------------------------------
ananth_db_1    docker-entrypoint.sh mysqld   Up      3306/tcp, 33060/tcp
ananth_web_1   catalina.sh run               Up      0.0.0.0:80->80/tcp,:::80->80/tcp, 8080/tcp
ananth@DESKTOP-LQBQG2D:~$ docker-compose down
Stopping ananth_db_1  ... done
Stopping ananth_web_1 ... done
Removing ananth_db_1  ... done
Removing ananth_web_1 ... done
Removing network ananth_default
ananth@DESKTOP-LQBQG2D:~$
```

CI/CD Pipeline Execution Using Docker and Jenkins

1. Introduction

Continuous Integration and Continuous Deployment (CI/CD) is a software development practice that automates the process of building, testing, and deploying applications. Jenkins, combined with Docker, helps streamline deployments by containerizing applications and running them in isolated environments.

---

2. Prerequisites

Before setting up the CI/CD pipeline, ensure you have:

- **Jenkins installed** (with Docker and required plugins)
- **Docker and Docker Compose installed**
- **A Git repository** with your application source code
- **A Dockerfile** for containerization

---

3. Setting Up the CI/CD Pipeline

Step 1: Install Required Jenkins Plugins

- Go to **Manage Jenkins** → **Manage Plugins** → **Available Plugins**
- Install the following:
    - **Pipeline**
    - **Docker Pipeline**
    - **Git**

Step 2: Create a Jenkins Pipeline Job

- Navigate to **Jenkins Dashboard** → **New Item**
- Select **Pipeline**
- Configure the job with your Git repository URL

Step 3: Define the Jenkinsfile

A `Jenkinsfile` defines the pipeline steps. Example:

groovy
Copy code

```
pipeline {
```

```
    agent any
    stages {
        stage('Clone Repository') {
            steps {
                git 'https://github.com/user/repo.git'
            }
        }
        stage('Build Docker Image') {
            steps {
                sh 'docker build -t my-app:latest .'
            }
        }
        stage('Run Container') {
            steps {
                sh 'docker run -d -p 8080:8080 my-app:latest'
            }
        }
        stage('Cleanup') {
            steps {
                sh 'docker system prune -f'
            }
        }
    }
}
```

---

4. Running the Pipeline

Once the pipeline is configured:

1. **Trigger the build** in Jenkins.
2. The console output will show pipeline execution steps.
3. If successful, the application will be deployed in a Docker container.

---

5. Verifying the Deployment

Check running containers:
sh
Copy code
```
docker ps
```

-
- Access the application via `http://localhost:8080`.

---

6. Conclusion

This guide demonstrates how to set up an automated CI/CD pipeline using Jenkins and Docker, ensuring seamless deployment and containerized execution of applications.

```
3359bc3d7a6a: Mounted from library/tomcat
f844dcf94898: Mounted from library/tomcat
4b7c01ed0534: Mounted from library/tomcat
latest: digest: sha256:1e7a64d4f2b24daf27d5154454704d07790c826c99f18266a1cb586f61d8f873 size: 2409
[Pipeline] }
[Pipeline] // withDockerRegistry
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```