Cemalhan Alptekin

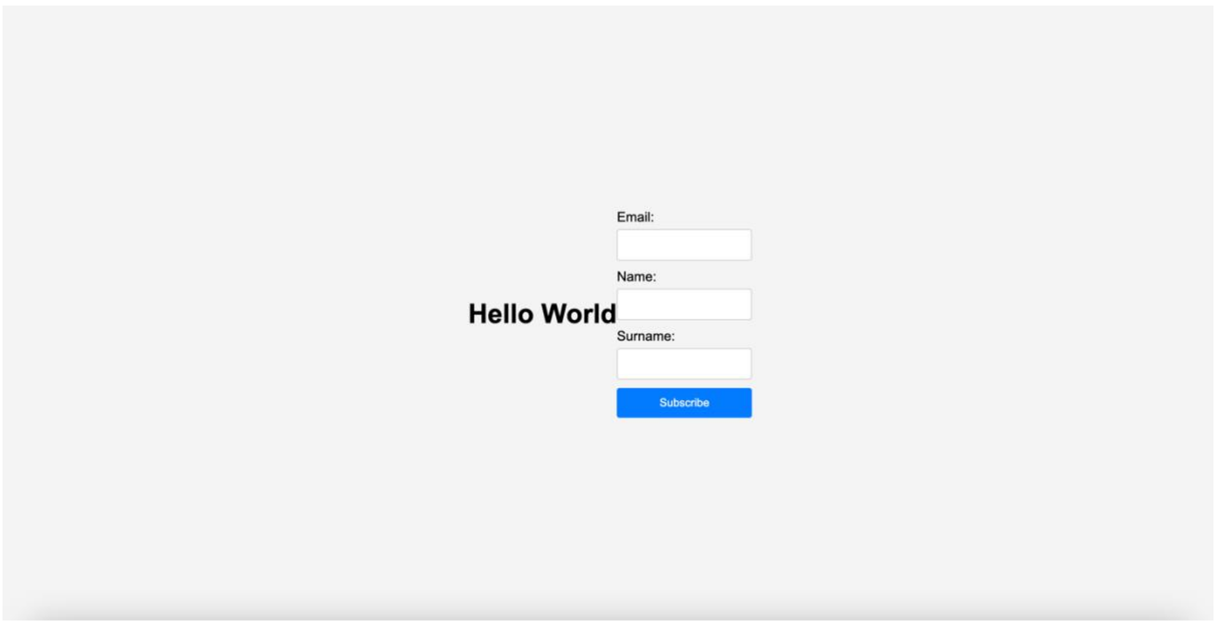
Hello World App: Overview and Installation

Application Overview

The **Hello World App** allows users to subscribe to an emailing list by entering their name, username, and email address, storing the data in a **MySQL database**. The app is containerized using **Docker** and orchestrated with **Docker Compose** to easily manage the app and database setup.

- **Frontend**: Simple HTML/CSS form.
- **Backend**: **Node.js** server that processes form submissions.
- **Database**: **MySQL** stores user information like name, username, and email.

Here's an image of the main screen of the **Hello World App**:



Local Setup Using Docker Compose

This guide explains how to set up the **Hello World App** and **MySQL** database locally using **Docker Compose**.

Prerequisites

- **Docker**: Make sure Docker is installed. You can get it here.
- **Docker Compose**: Docker Compose comes bundled with Docker Desktop, so if you've installed Docker, you should be good to go.

Step 1: Clone the Repository

```
git clone https://github.com/cemal995/hello-world-app
cd hello-world-app
```

Step 2: Docker Compose Configuration

Your project should contain a docker-compose.yml file that defines both the app and the MySQL database services. Here's an example configuration:

```
version: '3.8'

services:
  web:
    build: .
    ports:
      - "3000:3000"
    environment:
      MYSQL_HOST: db
      MYSQL_USER: user
      MYSQL_PASSWORD: password
      MYSQL_DATABASE: mydatabase
    depends_on:
      db:
        condition: service_healthy
    volumes:
      - .:/usr/src/app
    restart: always

  db:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: rootpassword
      MYSQL_DATABASE: mydatabase
      MYSQL_USER: user
      MYSQL_PASSWORD: password
    ports:
      - "3306:3306"
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "--silent"]
      interval: 10s
      retries: 5
      start_period: 30s
      timeout: 5s
    volumes:
      - ./init:/docker-entrypoint-initdb.d
    restart: always
```

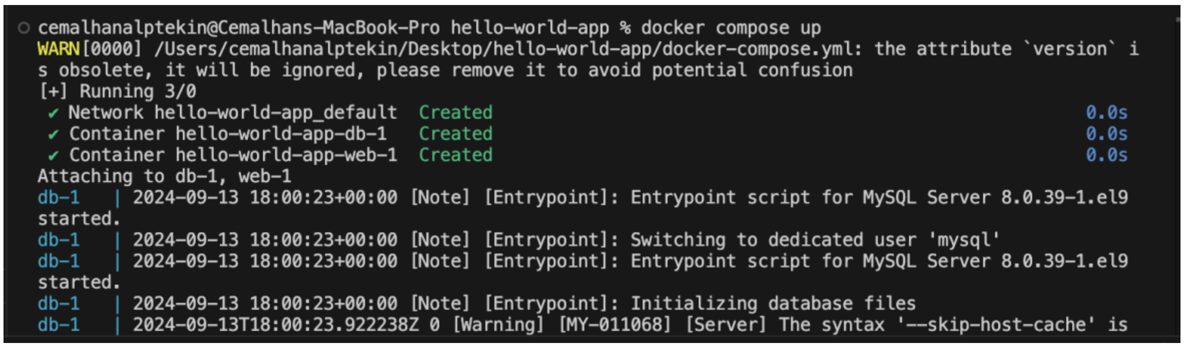
Breakdown of Services

- web: This service represents the Hello World app.
  - build: The Dockerfile will be used to build the app.
  - ports: The app will be accessible via port 3000.
  - environment: The app relies on environment variables to connect to the database (MYSQL_HOST, MYSQL_USER, etc.).
  - depends_on: Ensures that the database service (db) is up and healthy before the app starts.
  - volumes: Mounts the current directory to /usr/src/app in the container.
  - restart: Always restarts the container in case of failure.
- db: This service runs the MySQL database.
  - image: Uses the MySQL 8.0 Docker image.
  - environment: Defines environment variables such as MYSQL_ROOT_PASSWORD, MYSQL_USER, and MYSQL_PASSWORD to set up the MySQL database.
  - healthcheck: Monitors the MySQL service's health by pinging the database.
  - volumes: Mounts an initialization script from the ./init folder to initialize the database when the container is created.
  - restart: Always restarts in case of failure.

<u>Step 3: Start the App and Database</u>

Run the following command to bring up both services:

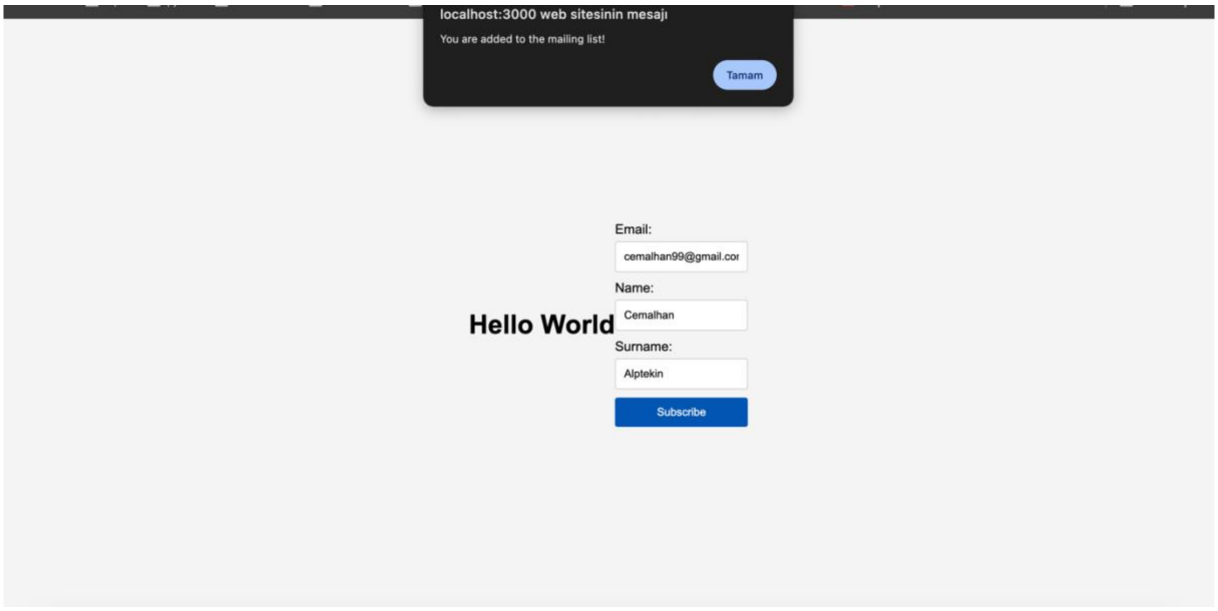
docker-compose up



```
cemalhanalptekin@Cemalhans-MacBook-Pro hello-world-app % docker compose up
WARN[0000] /Users/cemalhanalptekin/Desktop/hello-world-app/docker-compose.yml: the attribute `version` i
s obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 3/0
 ✔ Network hello-world-app_default  Created                                                   0.0s
 ✔ Container hello-world-app-db-1   Created                                                   0.0s
 ✔ Container hello-world-app-web-1  Created                                                   0.0s
Attaching to db-1, web-1
db-1   | 2024-09-13 18:00:23+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.39-1.el9
started.
db-1   | 2024-09-13 18:00:23+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
db-1   | 2024-09-13 18:00:23+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.39-1.el9
started.
db-1   | 2024-09-13 18:00:23+00:00 [Note] [Entrypoint]: Initializing database files
db-1   | 2024-09-13T18:00:23.922238Z 0 [Warning] [MY-011068] [Server] The syntax '--skip-host-cache' is
```

Step 4: Access the App

http://localhost:3000


localhost:3000 web sitesinin mesajı

You are added to the mailing list!

Hello World

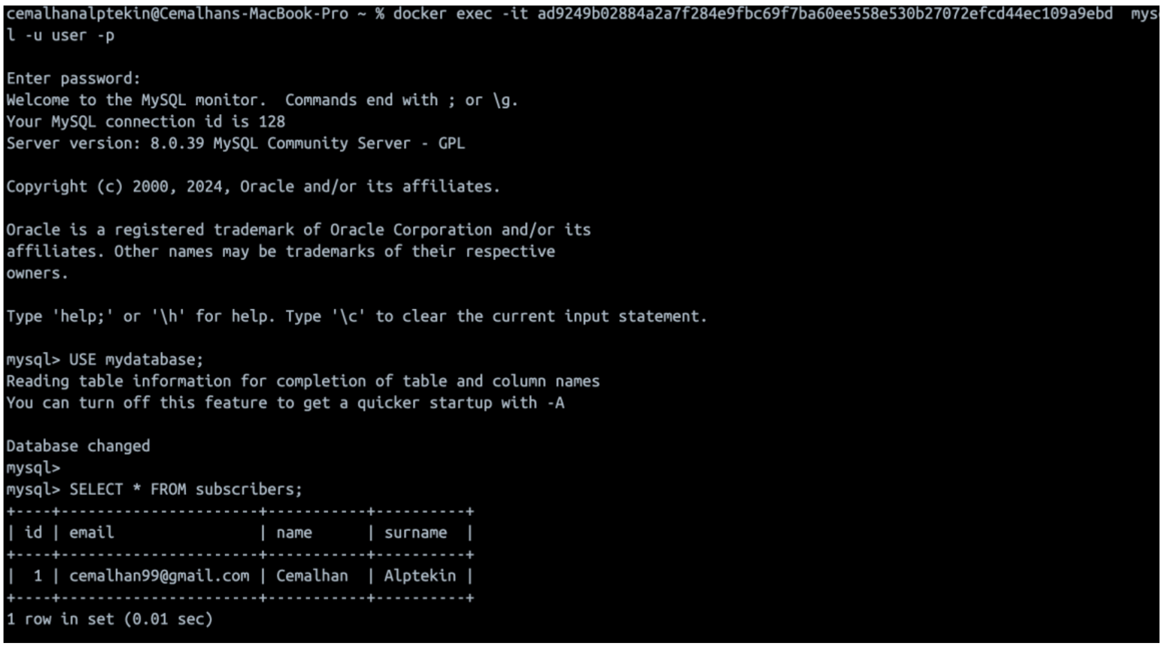
Email: cemalhan99@gmail.cor

Name: Cemalhan

Surname: Alptekin

Querying the MySQL Database

Once users submit their name, username, and email through the subscription form, the data gets saved to the MySQL database. You can query the database to view or manage these records using the MySQL command-line client or a MySQL GUI tool like phpMyAdmin, MySQL Workbench, or any other MySQL management tool.



```
cemalhanalptekin@Cemalhans-MacBook-Pro ~ % docker exec -it ad9249b02884a2a7f284e9fbc69f7ba60ee558e530b27072efcd44ec109a9ebd  mys
l -u user -p

Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 128
Server version: 8.0.39 MySQL Community Server - GPL

Copyright (c) 2000, 2024, 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> USE mydatabase;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql>
mysql> SELECT * FROM subscribers;
+----+---------------------+----------+---------+
| id | email               | name     | surname |
+----+---------------------+----------+---------+
|  1 | cemalhan99@gmail.com | Cemalhan | Alptekin |
+----+---------------------+----------+---------+
1 row in set (0.01 sec)
```

Setting Up Jenkins with Docker

This section outlines the process of setting up Jenkins on Docker and installing Docker within the Jenkins container to enable Docker-based operations in Jenkins jobs.

Step 1: Running the Jenkins Container

The Jenkins container can be run locally using Docker. The following command exposes Jenkins on port 8080 and the Jenkins worker communication port on 50000. The Docker socket and Jenkins home directory are mounted as volumes to allow Docker commands within the container.

Command to run the Jenkins container:

```
docker run -p 8080:8080 -p 50000:50000 -d \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v jenkins_home:/var/jenkins_home jenkins/jenkins:lts
```

- Ports:
  - 8080:8080 maps Jenkins to the local machine's port 8080.
  - 50000:50000 allows for master-slave communication in Jenkins.
- Volumes:
  - /var/run/docker.sock:/var/run/docker.sock enables Docker operations within Jenkins.
  - jenkins_home:/var/jenkins_home stores Jenkins jobs, configurations, and plugins.

Jenkins is then started in detached mode (-d).

Step 2: Installing Docker Inside Jenkins

To allow Jenkins jobs to execute Docker commands, Docker must be installed inside the running Jenkins container.

An interactive shell is started in the Jenkins container using the following command:

```
docker exec -it --user root <container_id> bash
```

The <container_id> can be obtained by running:

```
docker ps -a
```

Once inside the container, the following commands install Docker:

```
curl https://get.docker.com/ > dockerinstall && chmod 777 dockerinstall && ./dockerinstall
```

After installation, exit the interactive shell:

```
exit
```

Step 3: Adjusting Docker Socket Permissions

Permissions on the docker.sock file need to be adjusted to ensure that Jenkins can execute Docker commands:

```
su chmod 666 /var/run/docker.sock
```

This provides Jenkins with the necessary access to the Docker daemon.

Step 4: Installing Docker Compose Inside Jenkins

To enable running multi-container applications, Docker Compose must also be installed inside the Jenkins container. This can be done with the following commands:

Re-enter the Jenkins container with root privileges:

```
docker exec -it --user root <container_id> bash
```

Download and install Docker Compose by running:

```
curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Set the appropriate executable permissions:

```
chmod +x /usr/local/bin/docker-compose
```

Verify the installation:

```
docker-compose –version
```

This installs Docker Compose inside the Jenkins container, allowing Jenkins jobs to orchestrate multi-container setups.

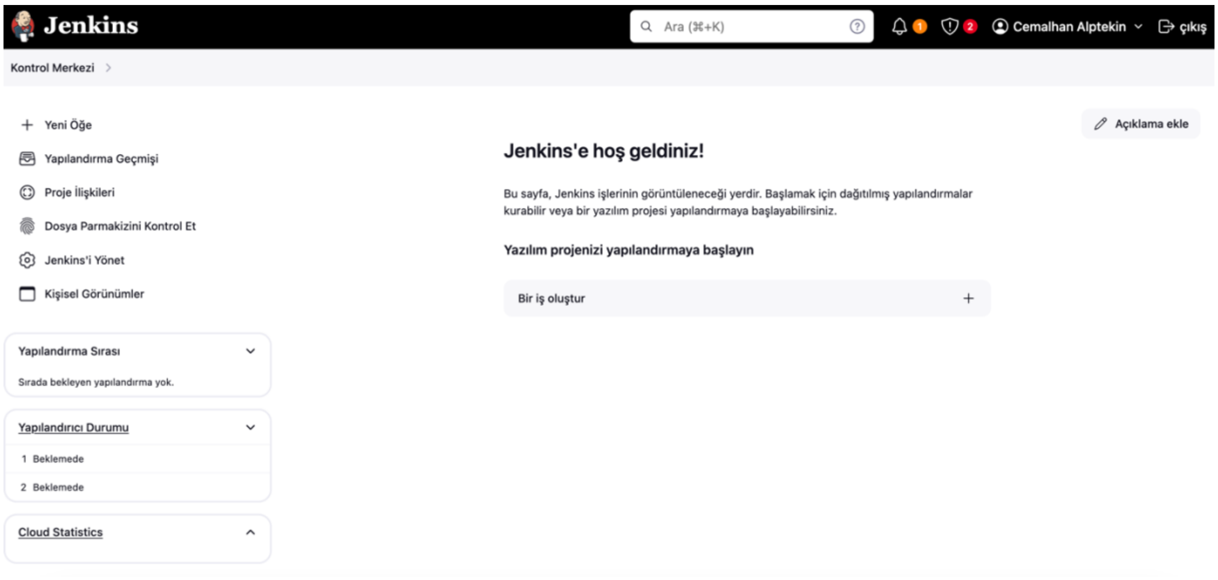
Step 5: Accessing Jenkins and Completing Setup

Jenkins can be accessed at http://localhost:8080. At first launch, Jenkins requests an initial admin password, stored inside the container. The password can be retrieved with this command:

```
docker exec -it <container_id> cat /var/jenkins_home/secrets/initialAdminPassword
```

The password is copied and entered on the Jenkins unlock page. Afterward, selecting Install Suggested Plugins installs the required plugins.

Finally, an admin user account is created, and Jenkins is ready for use.



Jenkins'e hoş geldiniz!

Bu sayfa, Jenkins işlerinin görüntüleneceği yerdir. Başlamak için dağıtılmış yapılandırmalar kurabilir veya bir yazılım projesi yapılandırmaya başlayabilirsiniz.

Yazılım projenizi yapılandırmaya başlayın

Sırada bekleyen yapılandırma yok.

Docker and Docker Compose are now installed inside Jenkins, and the environment is ready for Docker-based jobs, including multi-container applications.

Creating the Jenkins Pipeline

Once Jenkins is set up with Docker and Docker Compose, the next step is to create a Jenkins pipeline that integrates with Git, Jira, and Docker. This process involves downloading the necessary plugins and configuring credentials for Jira.

Step 1: Installing Required Jenkins Plugins

To enable integration with Git, Jira, and Docker within the pipeline, the following plugins must be installed:

- **Git Plugin**: For pulling source code from repositories like GitHub or GitLab.
- **Jira Plugin**: For connecting Jenkins with Jira and updating issues or creating comments automatically.
- **Docker Pipeline Plugin**: For executing Docker commands within Jenkins pipelines.

Navigate to **Manage Jenkins** > **Manage Plugins** and install these plugins by searching for them under the **Available** tab.

Step 2: Configuring Jira Credentials

Jenkins requires a Jira API token for authentication and to perform actions like creating or updating Jira tickets.

1. Go to **Manage Jenkins** > **Manage Credentials**.
2. Under the **Global domain**, click **Add Credentials**.
3. Select **Kind: Secret text** and input the Jira API token generated from your Jira account.

4. Add a meaningful ID (e.g., jira-api-token) and save the credential.

These credentials will be referenced in the pipeline script to authenticate Jenkins actions with Jira.

Step 3: Creating a Jenkins Pipeline

With the plugins installed and Jira credentials configured, create a pipeline to automate the CI/CD process.

Creating a Jenkins Pipeline for hello-world-app Using GitHub Jenkinsfile

1. **Access Jenkins Dashboard**
   - Open Jenkins in a web browser at http://localhost:8080.
2. **Create a New Pipeline Job**
   - Click on New Item in the Jenkins dashboard.
   - Enter a name for the new pipeline job (e.g., Hello-World-App-Pipeline).
   - Select Pipeline and click OK.
3. **Configure Pipeline Settings**
   - In the pipeline configuration page, go to the Pipeline section.
   - Set Definition to Pipeline script from SCM.
4. **Configure Source Control Management**
   - Set SCM to Git.
   - In the Repository URL field, enter the GitHub repository URL: https://github.com/cemal995/hello-world-app.git.
   - If the repository is private, configure the necessary credentials.
5. **Specify the Jenkinsfile Path**
   - In the Script Path field, enter the path to the Jenkinsfile in the repository (typically Jenkinsfile if it is located in the root directory of the repo).
6. **Save and Run the Pipeline**
   - Click Save to store the pipeline configuration.
   - Click Build Now to execute the pipeline and monitor its progress on the Jenkins dashboard.

This setup will instruct Jenkins to use the Jenkinsfile located in the hello-world-app GitHub repository for pipeline execution.

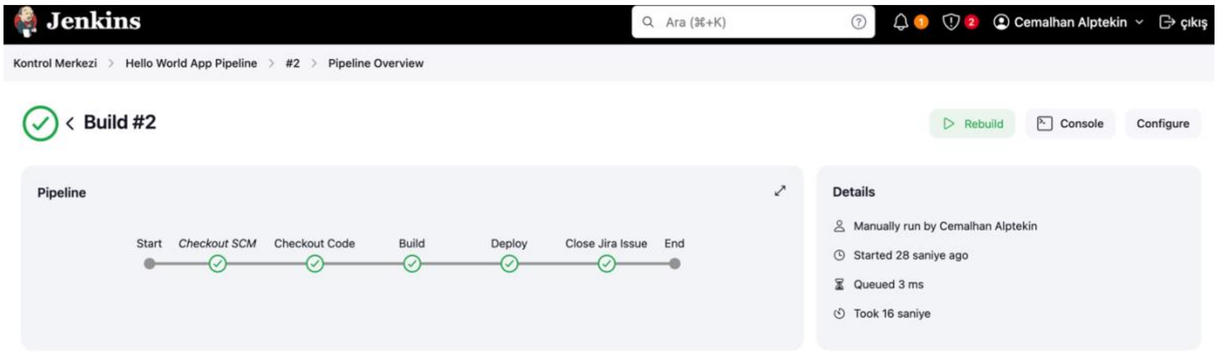
The pipeline will:

- Clone the Git repository containing the source code.
- Build a Docker image from the application’s Dockerfile.
- Deploy the Docker container.
- Update Jira with the build status or create a comment on a related ticket.

Below is an example of a Jenkinsfile that automates these tasks:

```
pipeline {
   agent any

   environment {
      JIRA_SITE = 'my-jira-site' // The name of your Jira site configuration in Jenkins
      JIRA_CREDENTIALS_ID = 'jira-api-token' // Your Jenkins credentials ID for Jira
API token
      JIRA_ISSUE_KEY = 'NODE-5'
   }

   stages {
      stage('Checkout Code') {
         steps {
            git branch: 'main', url: 'https://github.com/cemal995/hello-world-app.git'
         }
      }

      stage('Build') {
         steps {
            script {
               // Build the application using Docker Compose
               sh 'docker compose up --build -d'

               // Add a comment to the Jira issue indicating build started
               jiraAddComment site: "${env.JIRA_SITE}", idOrKey:
"${env.JIRA_ISSUE_KEY}", comment: 'Build started for the application.'
            }
         }
      }

      stage('Deploy') {
         steps {
            script {
               // Deploy the application using Docker
               sh 'docker compose up -d'

               // Add a comment to the Jira issue indicating deployment
               jiraAddComment site: "${env.JIRA_SITE}", idOrKey:
"${env.JIRA_ISSUE_KEY}", comment: 'Deployment completed successfully.'
            }
            }
        }

        stage('Close Jira Issue') {
            steps {
                script {
                    // Transition the Jira issue to "Done"
                    jiraTransitionIssue site: "${env.JIRA_SITE}", credentialsId:
"${env.JIRA_CREDENTIALS_ID}", idOrKey: "${env.JIRA_ISSUE_KEY}", input:
[transition: '31']
                }
            }
        }
    }

    post {
        failure {
            script {
                // In case of a failure, comment on the Jira issue
                jiraAddComment site: "${env.JIRA_SITE}", idOrKey:
"${env.JIRA_ISSUE_KEY}", comment: 'The build or deployment has failed.'
            }
        }
    }
}
```

Pipeline Overview:



Details

Manually run by Cemalhan Alptekin

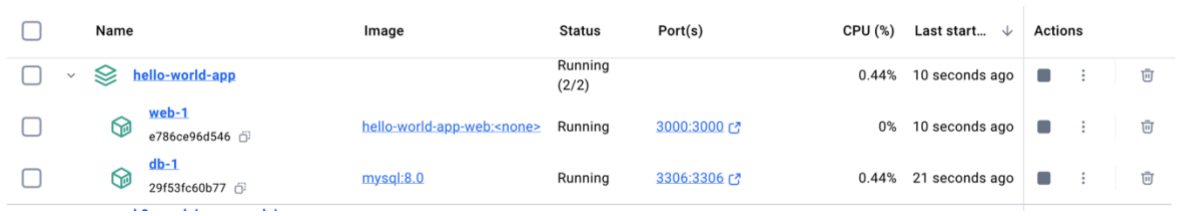
Started 28 saniye ago

Queued 3 ms

Took 16 saniye

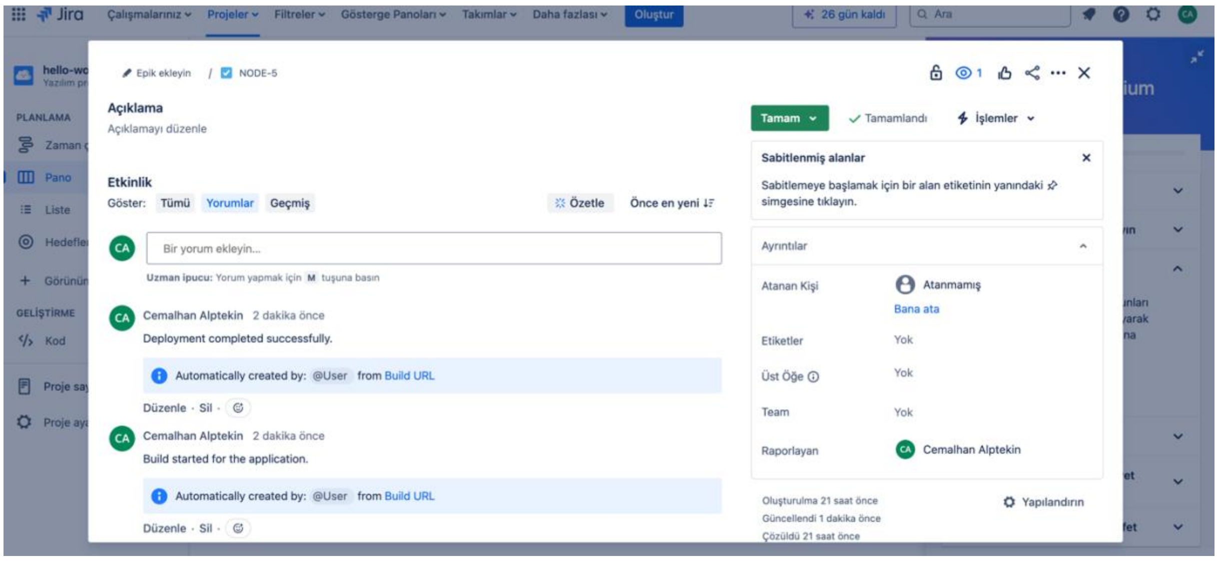
Cemalhan Alptekin

Running Docker Compose Application:



Name	Image	Status	Port(s)	CPU (%)	Last start...	Actions
hello-world-app		Running (2/2)		0.44%	10 seconds ago	
web-1 e786ce96d546	hello-world-app-web:<none>	Running	3000:3000	0%	10 seconds ago	
db-1 29f53fc60b77	mysql:8.0	Running	3306:3306	0.44%	21 seconds ago	

Jira Ticket that The Pipeline is Closed:



Açıklama

Açıklamayı düzenle

Etkinlik

Bir yorum ekleyin...

Uzman ipucu: Yorum yapmak için M tuşuna basın

Cemalhan Alptekin 2 dakika önce

Deployment completed successfully.

Automatically created by: @User from Build URL

Düzenle · Sil

Cemalhan Alptekin 2 dakika önce

Build started for the application.

Automatically created by: @User from Build URL

Düzenle · Sil

Sabitlenmiş alanlar

Sabitlemeye başlamak için bir alan etiketinin yanındaki simgesine tıklayın.

Ayrıntılar

Atanan Kişi	Atanmamış Bana ata
Etiketler	Yok
Üst Öğe	Yok
Team	Yok
Raporlayan	Cemalhan Alptekin

Oluşturulma 21 saat önce
Güncellendi 1 dakika önce
Çözüldü 21 saat önce