

Day-2

step-by-step guide to setting up a simple Python "Hello, Docker!" Flask application using Docker and Docker Compose.

1. Install Docker

First, install Docker to get the Docker engine running on your system:

```
sudo apt install -y docker.io
```

- **Explanation:** Installs Docker on your system using the apt package manager. The -y flag auto-confirms any prompts.
-

2. Start and Enable Docker Service

Start the Docker service and enable it to start automatically at boot time:

```
sudo systemctl start docker
```

```
sudo systemctl enable docker
```

- **Explanation:** The start command starts the Docker daemon, and enable ensures Docker runs on startup.
-

3. Verify Docker Installation

Verify that Docker was installed correctly by checking its version:

```
docker --version
```

- **Explanation:** Displays the installed Docker version to confirm the installation.
-

4. Install Docker Compose

Now, install Docker Compose, a tool to define and manage multi-container Docker applications:

```
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
```

- **Explanation:** The first command downloads the latest Docker Compose binary, and the second command makes it executable.
-

5. Verify Docker Compose Installation

Check the installed version of Docker Compose:

```
docker-compose --version
```

- **Explanation:** Displays the installed Docker Compose version to verify the installation.
-

6. Create Project Directory

Create a directory for your project and navigate into it:

```
mkdir ~/docker-python-app
```

```
cd ~/docker-python-app
```

- **Explanation:** Creates a directory for your project and navigates into it.
-

7. Create the app.py file

Create a Python file app.py for the Flask application:

```
nano app.py
```

Paste the following Flask application code:

```
from flask import Flask
```

```
app = Flask(__name__)
```

```
@app.route('/')
```

```
def hello_world():
```

```
    return 'Hello, world Running inside the docker!'
```

```
if __name__ == '__main__':
```

```
    app.run(host='0.0.0.0', port=5000)
```

- **Explanation:** A simple Flask app with one route (/) that returns a greeting message. The Flask server listens on all interfaces (0.0.0.0) and port 5000.
-

8. Create requirements.txt

Create a requirements.txt file to list Python dependencies:

```
nano requirements.txt
```

Add the following content:

```
flask
```

- **Explanation:** Lists the Flask library as the required dependency for your project.
-

9. Install pip (if not already installed)

Ensure pip is installed to handle Python package installations:

```
sudo apt update
```

```
sudo apt install python3-pip
```

- **Explanation:** Updates the package list and installs pip to handle Python packages.
-

10. Create Dockerfile

Create a Dockerfile that defines how the Docker image should be built:

```
nano Dockerfile
```

Add the following content:

```
# Use the official Python image from Docker Hub
```

```
FROM python:3.9-slim
```

```
# Set the working directory inside the container
```

```
WORKDIR /app
```

```
# Copy the current directory contents into the container at /app
```

```
COPY . /app
```

```
# Install any needed packages specified in requirements.txt
```

```
RUN pip install --no-cache-dir -r requirements.txt
```

```
# Make port 5000 available to the world outside the container
```

```
EXPOSE 5000
```

```
# Define the environment variable for Flask to run in production mode
```

ENV FLASK_ENV=production

Run app.py when the container launches

CMD ["python", "app.py"]

- **Explanation:** This Dockerfile defines the Python environment, installs dependencies, exposes port 5000, and starts the Flask app inside the container.
-

11. Create docker-compose.yml

Create a docker-compose.yml file to manage the application's services:

nano docker-compose.yml

Add the following content:

version: '3.8'

services:

web:

build: .

ports:

- "5000:5000"

environment:

- FLASK_ENV=development

volumes:

- ./app

restart: always

- **Explanation:** This Compose file:
 - o Defines the web service.
 - o Builds the image from the current directory.
 - o Maps port 5000 from the host to the container.
 - o Mounts the current directory (.) into the container to enable live code reloading.
 - o Restarts the container if it crashes.
-

12. Add User to Docker Group (if needed)

To avoid using sudo with Docker commands, add your user to the Docker group:

```
sudo usermod -aG docker $USER
```

```
newgrp docker
```

- **Explanation:** The first command adds your user to the Docker group, and the second command applies the changes to your current session.
-

13. Build and Run the Application

Now, you can build and start the Flask app container using Docker Compose:

```
docker-compose up --build
```

- **Explanation:** This command builds the Docker image and starts the container based on the docker-compose.yml configuration. The --build flag forces a rebuild of the Docker image.
-

14. Access the Application

Once the container is running, open your browser and navigate to:

```
http://localhost:5000
```

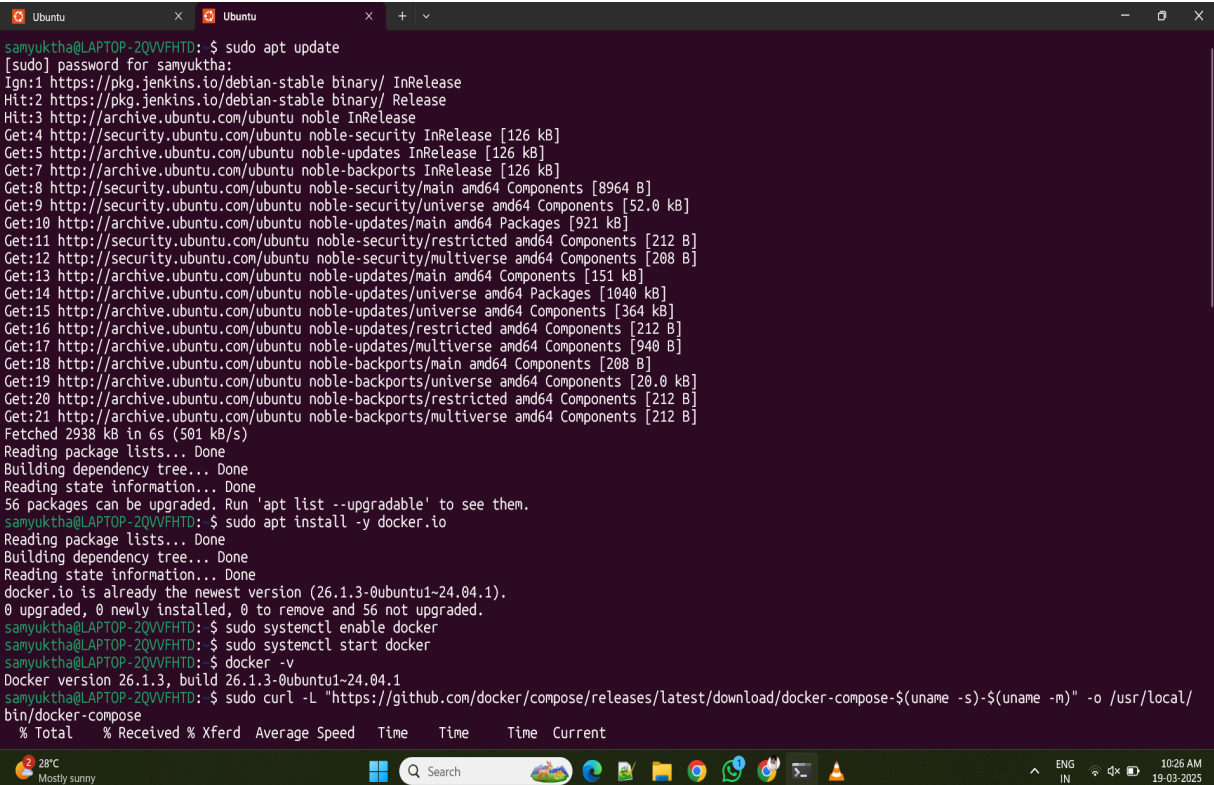
You should see the message: **"Hello, Docker Python App!"**

Summary of Commands

1. Install Docker:
2. `sudo apt install -y docker.io`
3. Start and enable Docker service:
4. `sudo systemctl start docker`
5. `sudo systemctl enable docker`
6. Install Docker Compose:
7. `sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose`
8. `sudo chmod +x /usr/local/bin/docker-compose`
9. Create project directory:
10. `mkdir ~/docker-python-app`
11. `cd ~/docker-python-app`

12. Create app.py with Flask code.
13. Create requirements.txt with flask.
14. Install pip (if needed):
15. sudo apt update
16. sudo apt install python3-pip
17. Create Dockerfile with the configuration.
18. Create docker-compose.yml with service definition.
19. Add your user to the Docker group (if necessary):
20. sudo usermod -aG docker \$USER
21. newgrp docker
22. Build and run the app:
23. docker-compose up --build

Now your "Hello, Docker!" Flask app should be running inside a Docker container, accessible at <http://localhost:5000>.



```
samyuktha@LAPTOP-2QVVFHTD: $ sudo apt update
[sudo] password for samyuktha:
Ign:1 https://pkg.jenkins.io/debian-stable binary/ InRelease
Hit:2 https://pkg.jenkins.io/debian-stable binary/ Release
Hit:3 http://archive.ubuntu.com/ubuntu noble InRelease
Get:4 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:5 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:7 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:8 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [8964 B]
Get:9 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Components [52.0 kB]
Get:10 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [921 kB]
Get:11 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Components [212 B]
Get:12 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Components [208 B]
Get:13 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 Components [151 kB]
Get:14 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [1040 kB]
Get:15 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 Components [364 kB]
Get:16 http://archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Components [212 B]
Get:17 http://archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Components [940 B]
Get:18 http://archive.ubuntu.com/ubuntu noble-backports/main amd64 Components [208 B]
Get:19 http://archive.ubuntu.com/ubuntu noble-backports/universe amd64 Components [20.0 kB]
Get:20 http://archive.ubuntu.com/ubuntu noble-backports/restricted amd64 Components [212 B]
Get:21 http://archive.ubuntu.com/ubuntu noble-backports/multiverse amd64 Components [212 B]
Fetched 2938 kB in 6s (501 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
56 packages can be upgraded. Run 'apt list --upgradable' to see them.
samyuktha@LAPTOP-2QVVFHTD: $ sudo apt install -y docker.io
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
docker.io is already the newest version (26.1.3-0ubuntu1-24.04.1).
0 upgraded, 0 newly installed, 0 to remove and 56 not upgraded.
samyuktha@LAPTOP-2QVVFHTD: $ sudo systemctl enable docker
samyuktha@LAPTOP-2QVVFHTD: $ sudo systemctl start docker
samyuktha@LAPTOP-2QVVFHTD: $ docker -v
Docker version 26.1.3, build 26.1.3-0ubuntu1-24.04.1
samyuktha@LAPTOP-2QVVFHTD: $ sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           % Done    %       0         0      0     0    0      0   0      0
28°C
Mostly sunny
```

```
0 651k 0 0:01:52 0:01:11 0:00:41 153 66 71.4M 66 47.4M 0 0 671k 0 0:01:48 0:01:12 0:00:36 172 68 71.4M 68 49.2M 0 0
687k 0 0:01:46 0:01:13 0:00:33 100 71.4M 0 0 818k 0 0:01:29 0:01:29 ---:--- 1392k711k
sanyuktha@LAPTOP-2QVVFHTD: $ sudo chmod +x /usr/local/bin/docker-compose
sanyuktha@LAPTOP-2QVVFHTD: $ docker-compose -v
Docker Compose version v2.34.0
sanyuktha@LAPTOP-2QVVFHTD: $ mkdir ~/docker-app
mkdir: cannot create directory '/home/sanyuktha/docker-app': File exists
sanyuktha@LAPTOP-2QVVFHTD: $ cd docker-app/
sanyuktha@LAPTOP-2QVVFHTD:~/docker-app$ nano app.py
sanyuktha@LAPTOP-2QVVFHTD:~/docker-app$ nano requirements.txt
sanyuktha@LAPTOP-2QVVFHTD:~/docker-app$ cat app.py
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello. World! Running inside Docker!"

if __name__ == "__main__":
    app.run(host="0.0.0.0",port=5000)

sanyuktha@LAPTOP-2QVVFHTD:~/docker-app$ nano requirements.txt
sanyuktha@LAPTOP-2QVVFHTD:~/docker-app$ cat requirements.txt
Flask
sanyuktha@LAPTOP-2QVVFHTD:~/docker-app$ nano dockerfile
sanyuktha@LAPTOP-2QVVFHTD:~/docker-app$ cat dockerfile
FROM python:3.11

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 5000

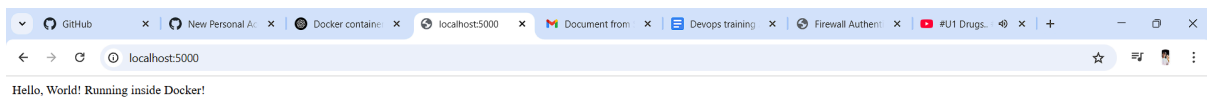
CMD ["python", "app.py"]
```

```
CMD ["python", "app.py"]
sanyuktha@LAPTOP-2QVVFHTD:~/docker-app$ nano docker-compose.yml
sanyuktha@LAPTOP-2QVVFHTD:~/docker-app$ cat docker-compose.yml
version: '3.8'

services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - ./app
    restart: always
sanyuktha@LAPTOP-2QVVFHTD:~/docker-app$ sudo docker-compose build
[sudo] password for sanyuktha:
WARN[0001] /home/sanyuktha/docker-app/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion
Compose can now delegate builds to bake for better performance.
To do so, set COMPOSE_BAKE=true.
[+] Building 198.7s (11/11) FINISHED
=> [web internal] load build definition from dockerfile 0.2s
=> => transferring dockerfile: 195B 0.0s
=> [web internal] load metadata for docker.io/library/python:3.11 4.4s
=> [web internal] load .dockerignore 0.1s
=> => transferring context: 2B 0.0s
=> [web 1/5] FROM docker.io/library/python:3.11@sha256:ebfa8696e47a68cfff 172.2s
=> => resolve docker.io/library/python:3.11@sha256:ebfa8696e47a68cfff 0.1s
=> => sha256:ebfa8696e47a68cfff31e370a93ce57c01bc7 9.08kB / 9.08kB 0.0s
=> => sha256:53218a9ae69d30aedef8f402e9a544862d3aee8 2.33kB / 2.33kB 0.0s
=> => sha256:18c0f2265fd94d4d17a0a465d6053f18dd19fdc 6.18kB / 6.18kB 0.0s
=> => sha256:7cd785773db44407e20a679ce5439222e505 48.47MB / 48.47MB 32.2s
=> => sha256:091eb8249475f42de217265c501e0186f0a3 24.01MB / 24.01MB 24.6s
=> => sha256:255774e0627b7d2327719e78dbad5ad8c9c 64.40MB / 64.40MB 32.6s
=> => sha256:363e14e5ce47664fba714e7d820801d90 211.35MB / 211.35MB 85.6s
=> => sha256:963091970bc2ac7e8b231ef98df8c79a8cb44a 6.16MB / 6.16MB 39.5s
=> => sha256:e7235c43f7e31b18eb2e528d9c2e6b7935d9 24.31MB / 24.31MB 39.4s
=> => extracting sha256:7cd785773db44407e20a679ce5439222e505475eed5 11.9s
=> => sha256:7f221c50e407e12492524c32985601495973634af6 250B / 250B 39.8s
=> => extracting sha256:091eb8249475f42de217265c501e0186f0a3ea7490ef 4.4s
```

```
Ubuntu
=> sha256:091eb8249475f42de217265c501e0186f0a3 24.01MB / 24.01MB 24.6s
=> sha256:255774e0027b72d2327719e78dbad5ad8c9c 64.40MB / 64.40MB 32.6s
=> sha256:353e14e5cc47664fba714a7da288001d90 211.35MB / 211.35MB 85.6s
=> sha256:963091970bc2ac7e8b231ef98df8c29a8cb44a 6.16MB / 6.16MB 39.5s
=> sha256:e7235c43f7e31b18eb2e528d9c2eb7935d9 24.31MB / 24.31MB 39.4s
=> extracting sha256:7cd785773db44407e20a679ce5439222e505475eed5 11.9s
=> sha256:7f221c50e407e12492524c32985601495973634af6 250B / 250B 39.8s
=> extracting sha256:091eb8249475f42de217265c501e0186f0a3ea7490ef 4.4s
=> extracting sha256:255774e0027b72d2327719e78dbad5ad8c9cf446d05 18.3s
=> extracting sha256:353e14e5cc47664fba714a7da288001d90427c70549 72.7s
=> extracting sha256:963091970bc2ac7e8b231ef98df8c29a8cb44a20a26 2.2s
=> extracting sha256:e7235c43f7e31b18eb2e528d9c2eb7935d99979a261 8.5s
=> extracting sha256:7f221c50e407e12492524c32985601495973634af6da 0.0s
[web internal] load build context
=> transferring context: 748B 0.0s
[web 2/5] WORKDIR /app 1.3s
[web 3/5] COPY requirements.txt . 0.2s
[web 4/5] RUN pip install --no-cache-dir -r requirements.txt 18.7s
[web 5/5] COPY . . 0.2s
[web] exporting to image 0.6s
=> exporting layers 0.5s
=> writing image sha256:7c97f28e1ff084378646804a3fb3ca9830d8a3579 0.0s
=> naming to docker.io/library/docker-app-web 0.0s
[web] resolving provenance for metadata file 0.1s
[+] Building 1/1
✔web Built 0.0s
sanyuktha@LAPTOP-2QVVFHTD:~/docker-app$ sudo docker-compose up -d
WARN[0000] /home/sanyuktha/docker-app/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 2/2
✔Network docker-app_default Created 0.4s
✔Container docker-app-web-1 Started 2.8s
sanyuktha@LAPTOP-2QVVFHTD:~/docker-app$ sudo docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
f774a01c8ad1 docker-app-web "python app.py" About a minute ago Restarting (1) 23 seconds ago docker-app-web-1
sanyuktha@LAPTOP-2QVVFHTD:~/docker-app$
```

```
✔Network docker-app_default Created 0.4s
✔Container docker-app-web-1 Started 2.8s
sanyuktha@LAPTOP-2QVVFHTD:~/docker-app$ sudo docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
f774a01c8ad1 docker-app-web "python app.py" About a minute ago Restarting (1) 23 seconds ago docker-app-web-1
sanyuktha@LAPTOP-2QVVFHTD:~/docker-app$
```



Jenkins Pipeline Through Git Token - Setup Procedure

Step 1: Generate a Git Personal Access Token

Before configuring the Jenkins pipeline, you need to generate a **Personal Access Token (PAT)** from your Git service.

GitHub (Example)

1. **Log in to GitHub** and navigate to your profile.
2. Go to **Settings > Developer Settings > Personal Access Tokens**.
3. Click **Generate New Token**.
4. Select the necessary permissions for the token. For example, to clone repositories, select:
 - o repo (full control of private repositories)
 - o read:org (for organization repository access)
5. Generate the token and **copy it**. This token will act as the password when Jenkins connects to GitHub.

GitLab (Example)

1. **Log in to GitLab** and go to **Profile Settings > Access Tokens**.
2. Generate a new token with appropriate scopes (e.g., read_repository).
3. **Save the token** to use in Jenkins.

Bitbucket (Example)

1. **Log in to Bitbucket** and go to **Personal Settings > App Passwords**.
2. Create an app password with necessary permissions (like repository read).
3. **Save the password** to use in Jenkins.

Step 2: Store Git Token in Jenkins Credentials

Once you've generated the Git token, the next step is to store it securely in Jenkins.

1. **Log in to Jenkins** and navigate to the Jenkins dashboard.
2. In the left menu, click on **Manage Jenkins**.
3. Click on **Manage Credentials**.
4. Select the appropriate **scope** (e.g., (Global)).
5. Click on **Add Credentials**.
6. In the **Kind** dropdown, select **Username with password**.
7. In the **Username** field, enter your Git username (e.g., your-username for GitHub).
8. In the **Password** field, paste the **Git token** you generated.
9. Optionally, give it an ID (e.g., git-token-jenkins).
10. Click **OK** to save the credentials.

Step 3: Configure Jenkins Pipeline

Now that the Git token is securely stored in Jenkins, you can configure a Jenkins pipeline to use it for Git interactions.

Example Pipeline Script (Declarative Pipeline)

You'll now set up a pipeline that uses Git for the source code. Here's an example using a declarative pipeline.

1. Create a New Pipeline Job:

- o Go to Jenkins Dashboard.
- o Click **New Item**, select **Pipeline**, and name your pipeline (e.g., Git-Pipeline).
- o Click **OK**.

2. Configure the Pipeline:

- o In the pipeline configuration, scroll to the **Pipeline** section.
- o Choose **Pipeline script from SCM**.
- o Set the **SCM** dropdown to **Git**.
- o In the **Repository URL** field, enter your repository URL (e.g., <https://github.com/yourusername/your-repository.git>).
- o Select **Credentials**. Choose the credentials you created earlier (e.g., git-token-jenkins).

Step 4: Run the Jenkins Pipeline

- After configuring the pipeline, click **Save** and then **Build Now** to run the pipeline.
- Jenkins will use the credentials you provided to authenticate with Git, clone the repository, and run the pipeline steps.

Step 5: Monitor and Troubleshoot

- If the pipeline fails, check the Jenkins job's **Console Output** for debugging information. Common issues can be due to incorrect credentials, Git URL, or permission issues.

New Item

Enter an item name

Nisanth

Select an item type



Freestyle project

Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.



Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.



Multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.



Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different

OK

Configure

General

Triggers

Pipeline

Advanced

General

Enabled ☒

Description

Jenkins with github through docker

Plain text Preview

- ☐ Discard old builds ?
- ☐ Do not allow concurrent builds
- ☐ Do not allow the pipeline to resume if the controller restarts
- ☐ GitHub project
- ☐ Pipeline speed/durability override ?
- ☐ Preserve stashes from completed builds ?

Configure

General

Triggers

Pipeline

Advanced

Set up automated actions that start your build based on specific events, like code changes or scheduled times.

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☐ GitHub hook trigger for GITScm polling ?
- ☐ Poll SCM ?
- ☐ Trigger builds remotely (e.g., from scripts) ?

Pipeline

Define your Pipeline using Groovy directly or pull it from source control.

Definition

Pipeline script from SCM

SCM ?

None

Script Path ?

Jenkinsfile

Save

Apply

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/nisanthg1010/Devops_Nisanth.git

Credentials ?

- none -

+ Add

Advanced

Add Repository

Save

Apply

Configure

General

Triggers

Pipeline

Advanced

Jenkins Credentials Provider: Jenkins

Add Credentials

Domain

Global credentials (unrestricted)

Kind

Username with password

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

Username ?

☐ Treat username as secret ?

Password ?

Save

Apply

Jenkins

🔍

🛡️ 1

👤 Nisanth G

🚪 log out

Dashboard > Nisanth >

Status

</> Changes

▶ Build Now

⚙️ Configure

🗑️ Delete Pipeline

📄 Stages

✎ Rename

🔍 Pipeline Syntax

Nisanth

Jenkins with github through docker

Permalinks

Edit description

Builds

No builds

REST APIJenkins 2.492.2

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

Global credentials (unrestricted)

+ Add Credentials

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
github-nisanthg1010	nisanthg1010/***** (Nisanth)	Username with password	Nisanth

Icon: S M L

REST APIJenkins 2.492.2

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

Global credentials (unrestricted)

+ Add Credentials

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
github-nisanthg1010	nisanthg1010/***** (Nisanth)	Username with password	Nisanth
docker_id	Nisanth1010/***** (jenkis file configure)	Username with password	jenkis file configure

Icon: S M L

REST APIJenkins 2.492.2

Jenkins Pipeline for Dockerized Application Deployment

This document provides a step-by-step guide on how the Jenkins pipeline automates the process of fetching the code from GitHub, building a Docker image, pushing it to a container registry, and deploying the application in a running Docker container.

Pipeline Overview

The pipeline follows these key steps:

1. **Checkout Code** - Fetch the latest code from the GitHub repository.
 2. **Build Docker Image** - Create a Docker image for the application.
 3. **Login to Docker Registry** - Authenticate to the container registry.
 4. **Push to Container Registry** - Upload the built image to a Docker registry.
 5. **Stop & Remove Existing Container** - Stop and remove any existing container with the same name.
 6. **Run Docker Container** - Deploy a new container with the updated image.
 7. **Post Actions** - Handle success or failure messages.
-

Step-by-Step Execution

1. Checkout Code

- Uses Jenkins credentials to authenticate and fetch the latest code from GitHub.
- Ensures secure access using stored credentials instead of exposing raw tokens.

Implementation:

```
stage('Checkout Code') {  
    steps {  
        withCredentials([usernamePassword(credentialsId: 'github-nisanthg1010',  
usernameVariable: 'GIT_USER', passwordVariable: 'GIT_TOKEN')]) {  
            git url:  
"https://$GIT_USER:$GIT_TOKEN@github.com/nisanthg1010/Devops_Nisanth.git",  
branch: 'main'  
        }  
    }  
}
```

2. Build Docker Image

- Builds the Docker image using the Dockerfile present in the repository.
- Tags the image with the latest version.

Implementation:

```
stage('Build Docker Image') {
    steps {
        sh 'docker build -t $DOCKER_IMAGE .'
    }
}
```

3. Login to Docker Registry

- Uses stored Jenkins credentials to log in securely to the Docker registry.
- Prevents exposing login credentials in the script.

Implementation:

```
stage('Login to Docker Registry') {
    steps {
        withCredentials([usernamePassword(credentialsId: 'docker_nisanth', usernameVariable:
'DOCKER_USER', passwordVariable: 'DOCKER_PASS')]) {
            sh 'echo $DOCKER_PASS | docker login -u $DOCKER_USER --password-stdin'
        }
    }
}
```

4. Push to Container Registry

- Pushes the newly built Docker image to the specified container registry.
- Ensures the latest version of the application is stored and accessible.

Implementation:

```
stage('Push to Container Registry') {
    steps {
        sh 'docker push $DOCKER_IMAGE'
    }
}
```

5. Stop & Remove Existing Container

- Stops and removes the running container if it exists.

- Prevents conflicts when deploying the new version.

Implementation:

```
stage('Stop & Remove Existing Container') {
  steps {
    script {
      sh '''
        if [ "$(docker ps -aq -f name=$CONTAINER_NAME)" ]; then
          docker stop $CONTAINER_NAME || true
          docker rm $CONTAINER_NAME || true
        fi
      '''
    }
  }
}
```

6. Run Docker Container

- Starts a new Docker container with the updated image.
- Maps the internal application port 5000 to 5001 on the host machine.

Implementation:

```
stage('Run Docker Container') {
  steps {
    sh 'docker run -d -p 5001:5000 --name $CONTAINER_NAME $DOCKER_IMAGE'
  }
}
```

7. Post Actions

- If successful, displays a success message.
- If failed, displays an error message.

Implementation:

```
post {
  success {
    echo "Build, push, and container execution successful!"
  }
}
```

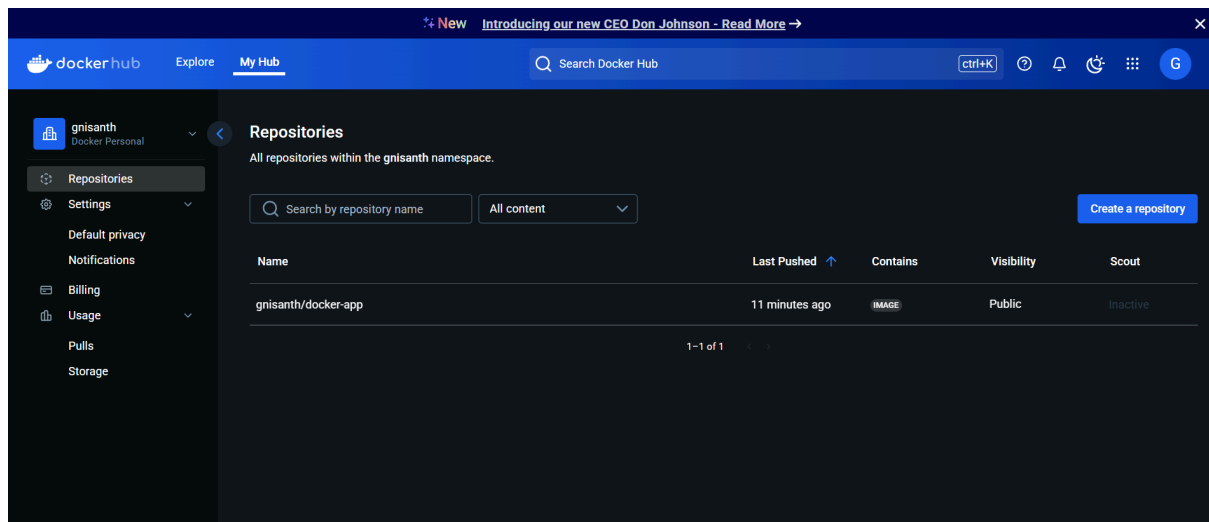


```
}  
failure {  
    echo "Build or container execution failed."  
}  
}
```

Conclusion

This Jenkins pipeline automates the entire process of fetching the code, building a Docker image, pushing it to a registry, and deploying the container. It ensures a seamless CI/CD workflow, making application updates smooth and efficient. 🚀

Hello, Docker Python App!



New

Introducing our new CEO Don Johnson - Read More

dockerhub

Explore

My Hub

Search Docker Hub

ctrl+K

G

gnisanth

Docker Personal

Repositories

Settings

Default privacy

Notifications

Billing

Usage

Pulls

Storage

Repositories / docker-app / General

Using 0 of 1 private repositories. [Get more](#)

gnisanth/docker-app

Last pushed 12 minutes ago

Add a description

Add a category

Docker commands

To push a new tag to this repository:

docker push gnisanth/docker-app:tagname

Public view

General

Tags

Image Management

Collaborators

Webhooks

Settings

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	less than 1 day	12 minutes

[See all](#)

buildcloud

Build with Docker Build Cloud

Accelerate image build times with access to cloud-based builders and shared cache.

Docker Build Cloud executes builds on optimally-dimensioned cloud infrastructure with dedicated per-organization isolation.