

## Day-2 SUJITH CS

### 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:
    - Defines the web service.
    - Builds the image from the current directory.
    - Maps port 5000 from the host to the container.
    - Mounts the current directory (.) into the container to enable live code reloading.
    - 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>.

```
nisanth@LAPTOP-G3U578JM:/mnt/c/Users/NISANTH$ sudo apt install -y docker.io
[sudo] password for nisanth:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base iptables libip4tc2 libip6tc2 libnetfilter-contrack3
  libnfnetlink0 libnftables1 libnftnl11 nftables pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools btrfs-progs cgroupfs-mount | cgroup-lite debootstrap docker-buildx docker-compose-v2 docker-doc
  rinse zfs-fuse | zfsutils firewalld
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io iptables libip4tc2 libip6tc2 libnetfilter-contrack3
  libnfnetlink0 libnftables1 libnftnl11 nftables pigz runc ubuntu-fan
0 upgraded, 16 newly installed, 0 to remove and 56 not upgraded.
Need to get 79.6 MB of archives.
After this operation, 306 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu noble/universe amd64 pigz amd64 2.8-1 [65.6 kB]
Get:2 http://archive.ubuntu.com/ubuntu noble/main amd64 libip4tc2 amd64 1.8.10-3ubuntu2 [23.3 kB]
Get:3 http://archive.ubuntu.com/ubuntu noble/main amd64 libip6tc2 amd64 1.8.10-3ubuntu2 [23.7 kB]
Get:4 http://archive.ubuntu.com/ubuntu noble/main amd64 libnfnetlink0 amd64 1.0.2-2build1 [14.8 kB]
Get:5 http://archive.ubuntu.com/ubuntu noble/main amd64 libnetfilter-contrack3 amd64 1.0.9-6build1 [45.2 kB]
Get:6 http://archive.ubuntu.com/ubuntu noble/main amd64 libnftnl11 amd64 1.2.6-2build1 [66.0 kB]
Get:7 http://archive.ubuntu.com/ubuntu noble/main amd64 iptables amd64 1.8.10-3ubuntu2 [381 kB]
Get:8 http://archive.ubuntu.com/ubuntu noble/main amd64 libnftables1 amd64 1.0.9-1build1 [358 kB]
Get:9 http://archive.ubuntu.com/ubuntu noble/main amd64 nftables amd64 1.0.9-1build1 [69.8 kB]
Get:10 http://archive.ubuntu.com/ubuntu noble/main amd64 bridge-utils amd64 1.7.1-1ubuntu2 [33.9 kB]
Get:11 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 runc amd64 1.1.12-0ubuntu3.1 [8599 kB]
Get:12 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 containerd amd64 1.7.24-0ubuntu1~24.04.1 [37.0 MB]
Ign:12 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 containerd amd64 1.7.24-0ubuntu1~24.04.1
```

```

nisanth@LAPTOP-G3U57BJM:/mnt/c/Users/NISANTH$ sudo apt install -y docker.io
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base iptables libip4tc2 libip6tc2 libnetfilter-contrack3
  libnftnlk0 libnftables1 libnftnl11 nftables pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools btrfs-progs cgroupfs-mount | cgroup-lite debootstrap docker-buildx docker-compose-v2 docker-doc
  rinse zfs-fuse | zfsutils firewallld
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io iptables libip4tc2 libip6tc2 libnetfilter-contrack3
  libnftnlk0 libnftables1 libnftnl11 nftables pigz runc ubuntu-fan
0 upgraded, 16 newly installed, 0 to remove and 56 not upgraded.
Need to get 69.9 MB/79.6 MB of archives.
After this operation, 306 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 containerd amd64 1.7.24-0ubuntu1~24.04.1 [37.0 MB]
Get:2 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 dns-root-data all 2024071801-ubuntu0.24.04.1 [5918 B]
Get:3 http://archive.ubuntu.com/ubuntu noble/main amd64 dnsmasq-base amd64 2.90-2build2 [375 kB]
Get:4 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 docker.io amd64 26.1.3-0ubuntu1~24.04.1 [32.4 MB]
Ign:4 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 docker.io amd64 26.1.3-0ubuntu1~24.04.1
Get:5 http://archive.ubuntu.com/ubuntu noble/universe amd64 ubuntu-fan all 0.12.16 [35.2 kB]
Get:4 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 docker.io amd64 26.1.3-0ubuntu1~24.04.1 [32.4 MB]
Fetched 62.8 MB in 4min 14s (247 kB/s)
Preconfiguring packages ...
Selecting previously unselected package pigz.
(Reading database ... 42940 files and directories currently installed.)
Preparing to unpack .../00-pigz_2.8-1_amd64.deb ...
Unpacking pigz (2.8-1) ...
Selecting previously unselected package libip4tc2:amd64.
Preparing to unpack .../01-libip4tc2_1.8.10-3ubuntu2_amd64.deb ...
Unpacking libip4tc2:amd64 (1.8.10-3ubuntu2) ...
Selecting previously unselected package libip6tc2:amd64.
Preparing to unpack .../02-libip6tc2_1.8.10-3ubuntu2_amd64.deb ...
Unpacking libip6tc2:amd64 (1.8.10-3ubuntu2) ...
Selecting previously unselected package libnftnlk0:amd64.
Preparing to unpack .../03-libnftnlk0_1.0.2-2build1_amd64.deb ...
Unpacking libnftnlk0:amd64 (1.0.2-2build1) ...
Selecting previously unselected package libnetfilter-contrack3:amd64.
Preparing to unpack .../04-libnetfilter-contrack3_1.0.9-6build1_amd64.deb ...

```

```

nisanth@LAPTOP-G3U57BJM:/mnt/c/Users/NISANTH$ sudo systemctl start docker
nisanth@LAPTOP-G3U57BJM:/mnt/c/Users/NISANTH$ sudo systemctl enable docker
nisanth@LAPTOP-G3U57BJM:/mnt/c/Users/NISANTH$ docker --version
Docker version 26.1.3, build 26.1.3-0ubuntu1~24.04.1
nisanth@LAPTOP-G3U57BJM:/mnt/c/Users/NISANTH$ 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
Dload Upload Total Spent Left Speed
0 0 0 0 0 0 0 0 0:00:00 0:00:00 0:00:00 0
nisanth@LAPTOP-G3U57BJM:/mnt/c/Users/NISANTH$ 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
Dload Upload Total Spent Left Speed
0 0 0 0 0 0 0 0 0:00:03 0:00:03 0:00:00 0
0 0 0 0 0 0 0 0 0:00:04 0:00:04 0:00:00 0
100 71.4M 100 71.4M 0 0 514k 0 0:02:22 0:02:22 0:00:00 789k
nisanth@LAPTOP-G3U57BJM:/mnt/c/Users/NISANTH$ sudo chmod +x /usr/local/bin/docker-compose
chmod: cannot access '/usr/local/bin/docker-compose': No such file or directory
nisanth@LAPTOP-G3U57BJM:/mnt/c/Users/NISANTH$ sudo chmod +x /usr/local/bin/docker-compose
nisanth@LAPTOP-G3U57BJM:/mnt/c/Users/NISANTH$ sudo chmod +x /usr/local/bin/docker-compose
nisanth@LAPTOP-G3U57BJM:/mnt/c/Users/NISANTH$ docker-compose --version
Docker Compose version v2.34.0
nisanth@LAPTOP-G3U57BJM:/mnt/c/Users/NISANTH$ mkdir ~/docker-python-app
mkdir: invalid option -- '/'
Try 'mkdir --help' for more information.
nisanth@LAPTOP-G3U57BJM:/mnt/c/Users/NISANTH$ mkdir ~/docker-python-app
nisanth@LAPTOP-G3U57BJM:/mnt/c/Users/NISANTH$ cd ~/docker-python-app
-bash: cd: -: invalid option
cd: usage: cd [-L|-P [-e]] [-e] [dir]
nisanth@LAPTOP-G3U57BJM:/mnt/c/Users/NISANTH$ cd ~/docker-python-app
nisanth@LAPTOP-G3U57BJM:/mnt/c/Users/NISANTH$ nano app.py
nisanth@LAPTOP-G3U57BJM:/mnt/c/Users/NISANTH$ from flask import Flask

```

```

nisanth@LAPTOP-G3U57BJM:~/docker-python-app$ ls
nisanth@LAPTOP-G3U57BJM:~/docker-python-app$ nano app.py
nisanth@LAPTOP-G3U57BJM:~/docker-python-app$ cat app.py
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, Docker Python App!'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
nisanth@LAPTOP-G3U57BJM:~/docker-python-app$ nano requirements.txt
nisanth@LAPTOP-G3U57BJM:~/docker-python-app$ cat app.py
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, Docker Python App!'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
nisanth@LAPTOP-G3U57BJM:~/docker-python-app$ pip install -r requirements.txt
Command 'pip' not found, but can be installed with:
sudo apt install python3-pip
nisanth@LAPTOP-G3U57BJM:~/docker-python-app$ sudo apt update
sudo apt install python3-pip
[sudo] password for nisanth:
Hit:1 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:2 http://archive.ubuntu.com/ubuntu noble InRelease
Hit:3 http://archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:4 http://archive.ubuntu.com/ubuntu noble-backports InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done

```

```

2Fvar%2Frun%2Fdocker.sock/v1.48/images/docker-python-app-web/json": dial unix /var/run/docker.sock: connect: permission denied
nisanth@LAPTOP-G3U57BJM:~/docker-python-app$ sudo usermod -sG docker $USER
nisanth@LAPTOP-G3U57BJM:~/docker-python-app$ newgrp docker
nisanth@LAPTOP-G3U57BJM:~/docker-python-app$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES
nisanth@LAPTOP-G3U57BJM:~/docker-python-app$ docker-compose up --build
WARN[0000] /home/nisanth/docker-python-app/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid
usion
Compose can now delegate builds to bake for better performance.
To do so, set COMPOSE_BAKE=true.
[+] Building 34.3s (10/10) FINISHED                                docker:default
=> [web internal] load build definition from Dockerfile           0.1s
=> => transferring dockerfile: 599B                               0.0s
=> [web internal] load metadata for docker.io/library/python:3.9-slim 4.1s
=> [web internal] load .dockerignore                               0.0s
=> => transferring context: 2B                                       0.0s
=> [web 1/4] FROM docker.io/library/python:3.9-slim@sha256:e52ca5f579cc58fed41efcbb55a8ed5dccc6c7a156c8ba76acfb4 20.7s
=> => resolve docker.io/library/python:3.9-slim@sha256:e52ca5f579cc58fed41efcbb55a8ed5dccc6c7a156c8ba76acfb4ab42f 0.0s
=> => sha256:e52ca5f579cc58fed41efcbb55a8ed5dccc6c7a156c8ba76acfb4ab42f 10.4kB / 10.4kB 0.0s
=> => sha256:246ee88fdd0bae1efcd3f6abbda800618093520b3aec0810f30e27dc0d609e0b0 1.75kB / 1.75kB 0.0s
=> => sha256:b6a33dea87178741208fa17205b317e46dd3f37b4fb1982d6022200726219b9bc 5.28kB / 5.28kB 0.0s
=> => sha256:6e909acdb790c8a1989d9cfc795fda5a246ad6664bb27b5c688e2b734b2c5fad 28.20MB / 28.20MB 12.9s
=> => sha256:cec49b84de9df5777ee4858eb2fd933b7ae87ec6746f4aa181d6893175331a21 3.51MB / 3.51MB 1.4s
=> => sha256:dalcbe8d584f421af08256ce9d3d64b361a433126b700d76ac42c23da4259346 14.93MB / 14.93MB 2.2s
=> => sha256:9a95d174077ce9cae854b9f5822065468265f57a58dae4d8fcec30f1e60216f 249B / 249B 0.0s
=> => extracting sha256:6e909acdb790c8a1989d9cfc795fda5a246ad6664bb27b5c688e2b734b2c5fad 4.3s
=> => extracting sha256:cec49b84de9df5777ee4858eb2fd933b7ae87ec6746f4aa181d6893175331a21 0.6s
=> => extracting sha256:dalcbe8d584f421af08256ce9d3d64b361a433126b700d76ac42c23da4259346 2.3s
=> => extracting sha256:9a95d174077ce9cae854b9f5822065468265f57a58dae4d8fcec30f1e60216f 0.0s
=> [web internal] load build context                               0.0s
=> => transferring context: 1.10kB                                    0.0s
=> [web 2/4] WORKDIR /app                                         0.4s
=> [web 3/4] COPY . /app                                          0.1s
=> [web 4/4] RUN pip install --no-cache-dir -r requirements.txt 8.5s
=> [web] exporting to image                                       0.3s
=> => exporting layers                                              0.2s
=> => writing image sha256:8296e4528d065435479b82d6ca5020b95967eef5dccc0465fe381ccd43eca88f7 0.0s
=> => naming to docker.io/library/docker-python-app-web         0.0s
=> [web] resolving provenance for metadata file                  0.0s
[+] Running 3/3

```



```

nisanth@LAPTOP-G3U57BJM:~/docker-app$ docker-compose up --build
Compose can now delegate builds to bake for better performance.
To do so, set COMPOSE_BAKE=true.
[+] Building 57.5s (10/10) FINISHED                                docker:default
=> [web internal] load build definition from Dockerfile            0.1s
=> => transferring dockerfile: 599B                               0.0s
=> [web internal] load metadata for docker.io/library/python:3.9-sli 0.0s
=> [web internal] load .dockerignore                              0.1s
=> => transferring context: 2B                                     0.0s
=> [web 1/4] FROM docker.io/library/python:3.9-slim              0.0s
=> [web internal] load build context                             0.1s
=> => transferring context: 1.10kB                                0.0s
=> CACHED [web 2/4] WORKDIR /app                                  0.0s
=> [web 3/4] COPY . /app                                          0.1s
=> [web 4/4] RUN pip install --no-cache-dir -r requirements.txt  56.7s
=> [web] exporting to image                                       0.4s
=> => exporting layers                                           0.3s
=> => writing image sha256:6952612da343bd0f62a8d2a753a4dc6e49284cab4 0.0s
=> => naming to docker.io/library/docker-app-web                0.0s
=> [web] resolving provenance for metadata file                  0.0s
[+] Running 3/3
  ✓web                               Built                        0.0s
  ✓Network docker-app_default        Created                   0.2s
  ✓Container docker-app-web-1        Created                   0.1s
Attaching to web-1
Gracefully stopping... (press Ctrl+C again to force)
Error response from daemon: driver failed programming external connectivity
on endpoint docker-app-web-1 (2f1d179af4d8d97a06799df2eaadc398d562a16bb3963b
a3c19cd77f72706f49): Bind for 0.0.0.0:5000 failed: port is already allocated

```

```

nisanth@LAPTOP-G3U57BJM:~/docker-app$ sudo docker ps
CONTAINER ID   IMAGE               COMMAND                  CREATED        STATUS        PORTS                               NAMES
a9a9d9673cb3   docker-python-app-web  "python app.py"         14 hours ago   Up 26 minutes   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp  docker-python-app-web-1

```

Hello, Docker Python 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:

- repo (full control of private repositories)
  - 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:**

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

## 2. Configure the Pipeline:

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


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

🔍

🔒

🔴

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

NewIntroducing our new CEO Don Johnson - Read More →

docker

Create your username

Continue with your Google account or [choose another](#).

Nisanth Gurusamy

nisanthg1010@gmail.com

Username

Nisanth1010

☐ Send me occasional product updates and announcements.

Sign up



By creating an account I agree to the [Subscription Service Agreement](#), [Privacy Policy](#), [Data Processing Terms](#).

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
 <a href="#">github-nisanthg1010</a>	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
 <a href="#">github-nisanthg1010</a>	nisanthg1010/***** (Nisanth)	Username with password	Nisanth 
 <a href="#">docker_id</a>	Nisanth1010/***** (jenkis file configure)	Username with password	jenkis file configure 

Icon: S M L

REST API

Jenkins 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:

- Checkout Code** - Fetch the latest code from the GitHub repository.
- Build Docker Image** - Create a Docker image for the application.
- Login to Docker Registry** - Authenticate to the container registry.
- Push to Container Registry** - Upload the built image to a Docker registry.
- Stop & Remove Existing Container** - Stop and remove any existing container with the same name.
- Run Docker Container** - Deploy a new container with the updated image.
- 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!

