

Day-2 DevOps-Training

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>.

```
pk@PraveenKumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ git config --global user.name "viratpk18"
pk@PraveenKumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ git commit -m "Jenkin-Docker Created"
[main 395d204] Jenkin-Docker Created
 4 files changed, 38 insertions(+)
  create mode 100644 Dockerfile
  create mode 100644 app.py
  create mode 100644 docker-compose.yml
  create mode 100644 requirements.txt
pk@PraveenKumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ git branch -M main
pk@PraveenKumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ git remote add origin https://github.com/viratpk18/Jenkins-Docker
error: remote origin already exists.
pk@PraveenKumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ git push -u origin main
Username for 'https://github.com': viratpk18
Password for 'https://viratpk18@github.com':
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for information on currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/viratpk18/Jenkins-Docker/'
pk@PraveenKumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ git push -u origin main
Username for 'https://github.com': viratpk18
Password for 'https://viratpk18@github.com':
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for information on currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/viratpk18/Jenkins-Docker/'
pk@PraveenKumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ git push -u origin main
Username for 'https://github.com': viratpk18
Password for 'https://viratpk18@github.com':
pk@PraveenKumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ git push https://viratpk18:ghp_1rQv0vU82kIHcxCQjd2Hi2aL2x6ZY2FVask@github.com/viratpk18:https://github.com/viratpk18/Jenkins-Docker.git
remote: Not Found
fatal: repository 'https://github.com/viratpk18/https://github.com/viratpk18/Jenkins-Docker.git/' not found
pk@PraveenKumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ git push https://viratpk18:ghp_1rQv0vU82kIHcxCQjd2Hi2aL2x6ZY2FVask@github.com/viratpk18/Jenkins-Docker.git
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 883 bytes | 58.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/viratpk18/Jenkins-Docker.git
 905c366..395d204  main -> main
```

```
C:\Users\Praveenkumar>wsl.exe -d Ubuntu
pk@Praveenkumar:/mnt/c/Users/Praveenkumar$ sudo apt update
[sudo] password for pk:
Hit:1 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:2 http://archive.ubuntu.com/ubuntu noble InRelease
Get:3 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Ign:4 https://pkg.jenkins.io/debianstable binary/ InRelease
Hit:5 https://ppa.launchpadcontent.net/openjdk-r/ppa/ubuntu noble InRelease
Err:6 https://pkg.jenkins.io/debianstable binary/ Release
  404  Not Found [IP: 151.101.158.133 443]
Hit:7 http://archive.ubuntu.com/ubuntu noble-backports InRelease
Get:8 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [921 kB]
Get:9 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [1040 kB]
Reading package lists... Done
E: The repository 'https://pkg.jenkins.io/debianstable binary/ Release' does not have a Release file.
N: Updating from such a repository can't be done securely, and is therefore disabled by default.
N: See apt-secure(8) manpage for repository creation and user configuration details.
pk@Praveenkumar:/mnt/c/Users/Praveenkumar$ 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
  libnfnetlink0 libnftables1 libnftnl1 libnftnl11 nftables pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroups cgroups-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 0 not upgraded.
Need to get 79.6 MB of archives.
After this operation, 386 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 [23.7 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]
```

```
update-alternatives: using /usr/sbin/arptables-nft to provide /usr/sbin/arptables (arptables) in auto mode
update-alternatives: using /usr/sbin/ebttables-nft to provide /usr/sbin/ebttables (ebtables) in auto mode
Setting up docker.io (26.1.3-0ubuntu1-24.04.1) ...
info: Selecting GID from range 100 to 999 ...
info: Adding group 'docker' (GID 109) ...
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /usr/lib/systemd/system/docker.socket.
Setting up dnsmasq-base (2.90-2build2) ...
Setting up ubuntu-fan (0.12.16) ...
Created symlink /etc/systemd/system/multi-user.target.wants/ubuntu-fan.service → /usr/lib/systemd/system/ubuntu-fan.service.
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for dbus (1.14.18-4ubuntu4.1) ...
Processing triggers for liblc-bin (2.39-0ubuntu8.4) ...
pk@Praveenkumar:/mnt/c/Users/Praveenkumar$ sudo systemctl enable docker
pk@Praveenkumar:/mnt/c/Users/Praveenkumar$ sudo systemctl start docker
pk@Praveenkumar:/mnt/c/Users/Praveenkumar$ docker --version
Docker version 26.1.3, build 26.1.3-0ubuntu1-24.04.1
pk@Praveenkumar:/mnt/c/Users/Praveenkumar$ 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 =Xfer Average Speed   Time     Time      Current
          %:--:--  %:--:--  :--:--:--  :--:--:--  :--:--:--  :--:--:--  Speed
  0     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
  0     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
100  71.4M  100  71.4M  0    0  1595k  0    0:00:45  0:00:45  0:00:45  0:00:45  2230K
pk@Praveenkumar:/mnt/c/Users/Praveenkumar$ sudo chmod +x /usr/local/bin/docker-compose
pk@Praveenkumar:/mnt/c/Users/Praveenkumar$ docker-compose --version
Docker Compose version v2.34.0
pk@Praveenkumar:/mnt/c/Users/Praveenkumar$ mkdir docker-python
pk@Praveenkumar:/mnt/c/Users/Praveenkumar$ ls
AppData          Pictures/
'Application Data'  Music
'Contacts'        NTUSER.DAT
Cookies          NTUSER.DAT[cb182e0-be13-11ef-9689-8f535641f038].TM.blf
Documents         NTUSER.DAT[cb182e0-be13-11ef-9689-8f535641f038].TMContainer000000000000000000000001.regrtrans-ms
Downloads         NTUSER.DAT[cb182e0-be13-11ef-9689-8f535641f038].TMContainer000000000000000000000002.regrtrans-ms
'FakeNewsDetection' NTUSER.DAT[cb182e0-be13-11ef-9689-8f535641f038].TMContainer000000000000000000000003.regrtrans-ms
Favorites        NetHood
Links            OneDrive
'Local Settings' Oracle
pk@Praveenkumar:/mnt/c/Users/Praveenkumar$ cd docker-python
pk@Praveenkumar:/mnt/c/Users/Praveenkumar$ ls
Postman          docker-python
PrintHood        eclipse
Recent          eclipse-workspace
Saved Games      node_modules
Searches         ntuser.dat.LOG1
SendTo           ntuser.dat.LOG2
'Start Menu'    ntuser.ini
Templates       package-lock.json
Videos           package.json
'VirtualBox VMs'
```

```
pk@Praveenkumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ git add .
pk@Praveenkumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ git commit -m "Jenkinsfile created"
[main c93ce18] Jenkinsfile created
 1 file changed, 67 insertions(+)
 create mode 100644 Jenkinsfile
pk@Praveenkumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ git push
Username for 'https://github.com': viratpk18
Password for 'https://viratpk18@github.com':
pk@Praveenkumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ git push https://viratpk18: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@github.com/viratpk18/Jenkins-Docker
r.git
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 998 bytes | 61.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/viratpk18/Jenkins-Docker.git
 395d2b4..c93ce18  main -> main
pk@Praveenkumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ sudo systemctl restart jenkins
[sudo] password for pk:
pk@Praveenkumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ sudo systemctl enable jenkins
Synchronizing state of jenkins.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable jenkins
pk@Praveenkumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ sudo usermod -aG docker jenkins
pk@Praveenkumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ sudo systemctl restart jenkins
pk@Praveenkumar:/mnt/c/Windows/System32/docker-python/Jenkins-Docker$ sudo systemctl restart docker
```

```
nisanth@LAPTOP-GJUB7BJM:~/dockers-app$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
a9ayd9673cb3        docker-python-app-web   "python app.py"   14 hours ago      Up 26 minutes   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp
NAMES
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:
 - 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

Enabled

General

Triggers

Pipeline

Advanced

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

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

General

Triggers

Pipeline

Advanced

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

localhost:8080/job/Jenkins-Pipelines/configure

Dashboard > Jenkins-Pipelines > Configuration

Configure Pipeline

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

Pipeline

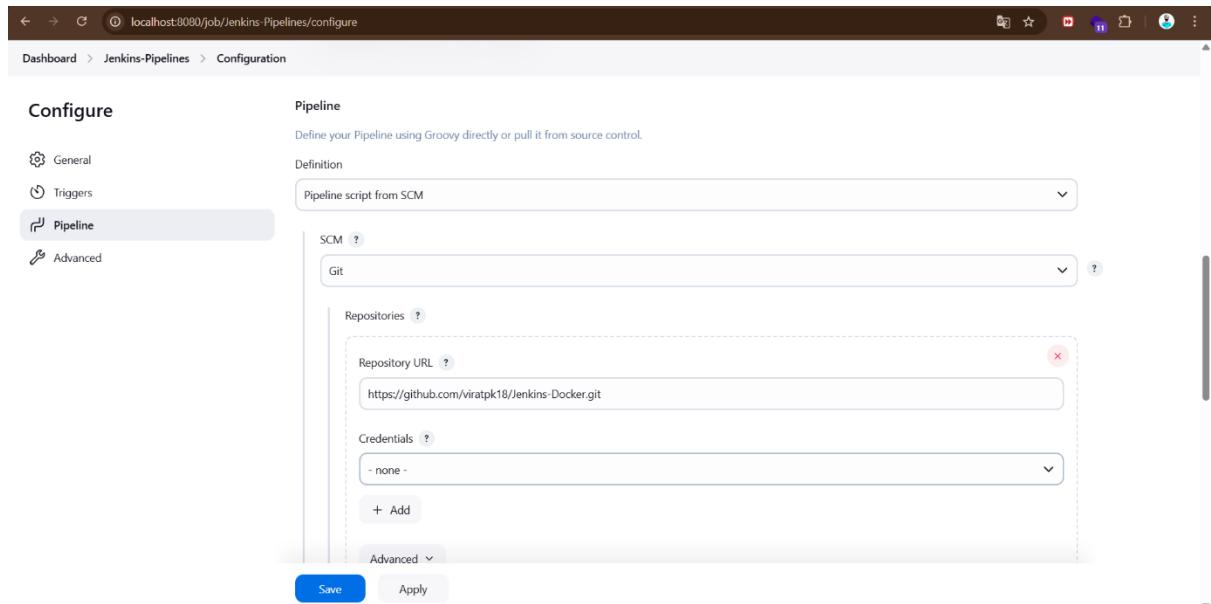
SCM ?

Repository URL ?
https://github.com/viratpk18/Jenkins-Docker.git

Credentials ?
- none -

+ Add

Save Apply



Configure Jenkins Credentials Provider: Jenkins

Add Credentials

Domain
Global credentials (unrestricted)

Kind
Username with password

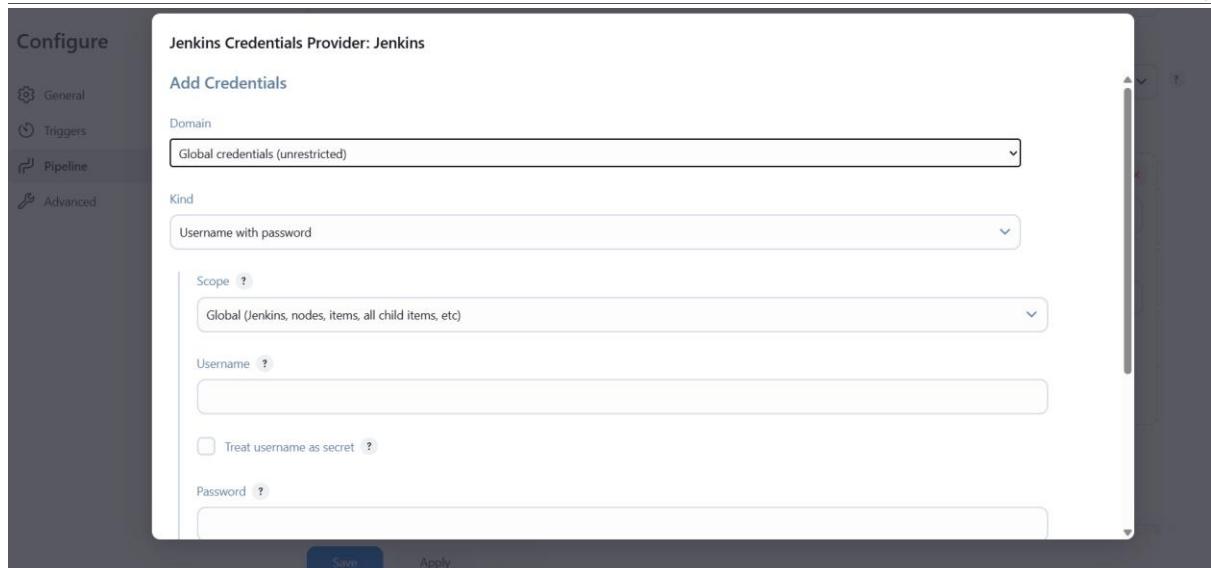
Scope ?
Global (Jenkins, nodes, items, all child items, etc)

Username ?

Treat username as secret ?

Password ?

Save Apply



localhost:8080/job/Jenkins-Pipelines/

Jenkins

Praveen Kumar Log out

Dashboard > Jenkins-Pipelines >

Jenkins-Pipelines

Status Changes Build Now Configure Delete Pipeline Stages Rename Pipeline Syntax

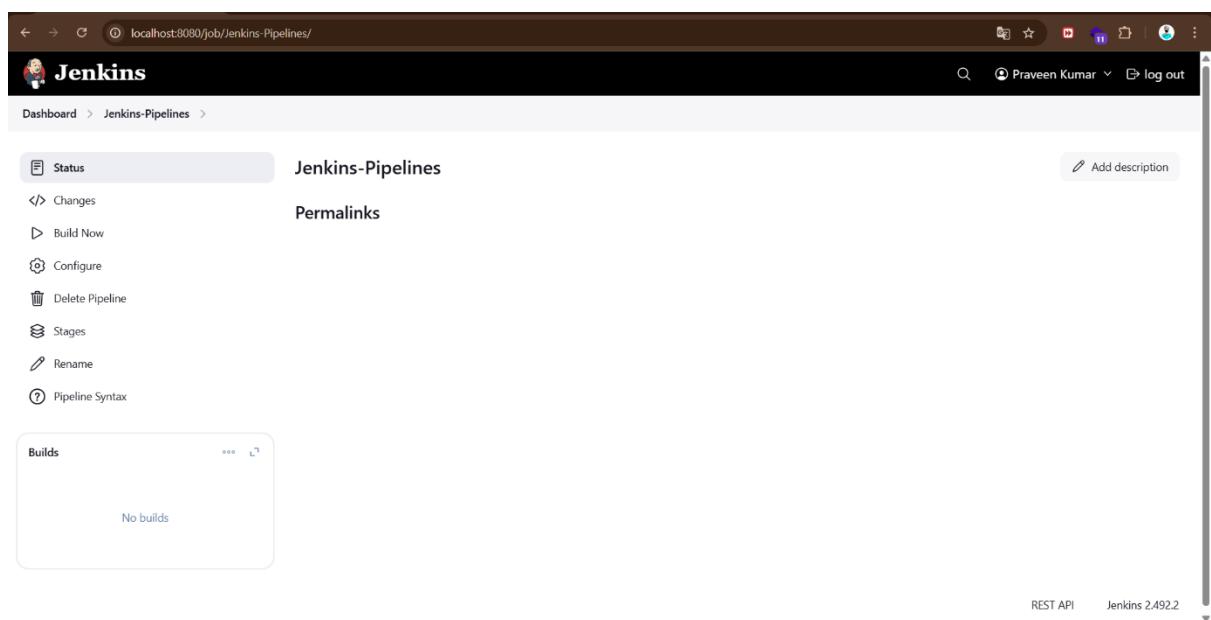
Permalinks

Builds *** L¹

No builds

Add description

REST API Jenkins 2.492.2



The screenshot shows the Jenkins Global credentials (unrestricted) page. At the top, there is a navigation bar with links to Dashboard, Manage Jenkins, Credentials, System, and Global credentials (unrestricted). A search bar and user information (Praveen Kumar) are also present. Below the navigation, the title "Global credentials (unrestricted)" is displayed, along with a note about credentials being available irrespective of domain specification. A table lists a single credential entry:

ID	Name	Kind	Description
git_viratpk18	viratpk18/*****	Username with password	

Below the table, there are icons for sorting (S, M, L) and a REST API link at the bottom right.

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!

hub.docker.com/repositories/viratpk18

New Introducing our new CEO Don Johnson - Read More →

Docker Hub Explore My Hub

Search Docker Hub

Repositories

All repositories within the viratpk18 namespace.

Search by repository name All content

Create a repository

Name	Last Pushed	Contains	Visibility	Scout
viratpk18/docker-app	7 minutes ago	IMAGE	Public	Inactive

1–1 of 1

hub.docker.com/repository/docker/viratpk18/docker-app/general

New Introducing our new CEO Don Johnson - Read More →

Docker Hub Explore My Hub

Search Docker Hub

Repositories / docker-app / General

Using 0 of 1 private repositories. Get more

Docker commands

To push a new tag to this repository:

```
docker push viratpk18/docker-app:tagname
```

viratpk18/docker-app ⓘ

Last pushed about 1 hour ago · Repository size: 366 MB

Add a description ⓘ

Add a category ⓘ

General Tags Image Management BETA Collaborators Webhooks Settings

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest	Ubuntu	Image	less than 1 day	about 1 hour

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.

Get faster builds through shared caching across your team, native multi-platform support, and encrypted data transfer - all without managing infrastructure.