Ex. No.:2	Docker Containers and Images
Date:23-07-25	

Objective

The objective is to verify Docker installation, create a custom Docker image using Nginx, and run a container to serve the webpage.

Tools/ Software Required

- Docker (installed and running)
- Command Line / Terminal
- Code Editor (VS Code / Sublime / Notepad++)
- Web Browser

Industry Relevance and Applications

- Containerization: Runs applications in isolated environments.
- Microservices Architecture: Deploying services in lightweight containers.
- DevOps and CI/CD: Automates build, test, and deployment workflows.
- Portability: Same image runs on multiple environments without compatibility issues.

Description

Docker is an open-source platform that enables containerization, a method of packaging applications along with all their dependencies into lightweight, portable containers. Unlike virtual machines, containers share the host operating system kernel, making them faster, more efficient, and less resource-intensive.

Working of Docker

- Image and Container Concept:
  - A Docker Image is a read-only template that contains the application code, runtime, libraries, and configuration files.
  - A Docker Container is a running instance of an image that provides an isolated execution environment.
- Portability: Docker containers ensure the application runs the same way in any environment — development, testing, or production — eliminating compatibility issues.

Typical Workflow

1. **Developer Stage**
   - Developers write and test code locally. The application code, dependencies, and configurations are defined in a Dockerfile.

2. **Docker Build Stage**
   - The docker build command creates an image from the Dockerfile. This image serves as the base for creating containers.
3. **CI/CD Pipeline Integration**
   - Continuous Integration and Continuous Deployment (CI/CD) pipelines automate the build and deployment process.
   - **Build:** The pipeline creates the image whenever new code is pushed to the repository.
   - **Test:** Automated tests validate the container before deployment.
   - **Push:** The tested image is pushed to a container registry like Docker Hub.
4. **Deployment Anywhere**
   - Once the image is in the registry, it can be deployed on any infrastructure:
   - Cloud Platforms: AWS, Azure, Google Cloud
   - On-Premise Systems: Local servers
   - Orchestrators: Kubernetes, Docker Swarm for managing multiple containers
   - Tools like Ansible, Helm, and Terraform are used for automated deployment and scaling.

This experiment demonstrates:

1. Checking Docker Installation
2. Running a Test Container
3. Building a Custom Docker Image
4. Running a Web Application in a Docker Container

Command list with explanation:

docker –version : Checks installed Docker version.

docker run hello-world : Runs a test container to verify Docker setup.

mkdir docker-demo : Creates a new folder for the project.

docker build -t myweb:v1 . : Builds a Docker image named myweb with tag v1.

docker run -d -p 8080:80 myweb:v1 : Runs container in detached mode.

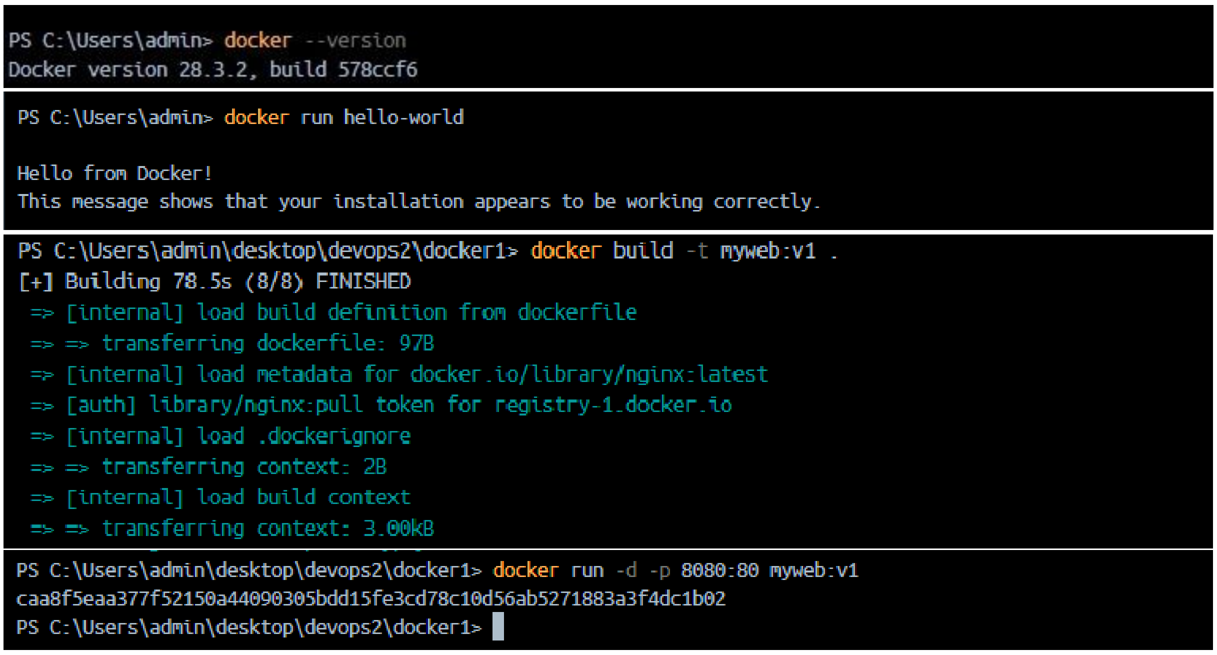
Procedure

1. Check Docker Installation by running : docker --version
2. Run Test Container using : docker run hello-world
3. Create Project Folder

```
mkdir docker-demo
cd docker-demo
```

4. Create a Web Page (index.html)
5. Create a Dockerfile
6. Build Docker Image using: docker build -t myweb:v1 .
7. Run Docker Container using: docker run -d -p 8080:80 myweb:v1

Sample Inputs:



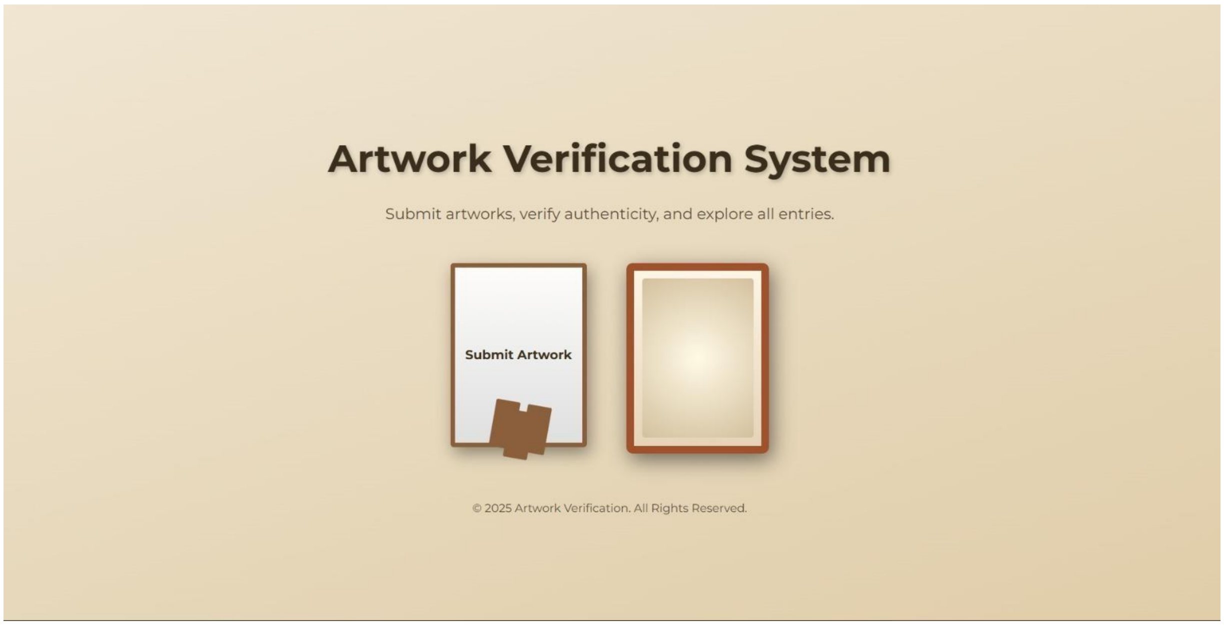
```
PS C:\Users\admin> docker --version
Docker version 28.3.2, build 578ccf6
```

```
PS C:\Users\admin> docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

```
PS C:\Users\admin\desktop\devops2\docker1> docker build -t myweb:v1 .
[+] Building 78.5s (8/8) FINISHED
 => [internal] load build definition from dockerfile
 => => transferring dockerfile: 97B
 => [internal] load metadata for docker.io/library/nginx:latest
 => [auth] library/nginx:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build context
 => => transferring context: 3.00kB
```

```
PS C:\Users\admin\desktop\devops2\docker1> docker run -d -p 8080:80 myweb:v1
caa8f5eaa377f52150a44090305bdd15fe3cd78c10d56ab5271883a3f4dc1b02
PS C:\Users\admin\desktop\devops2\docker1>
```

Output:



Artwork Verification System

Submit artworks, verify authenticity, and explore all entries.

© 2025 Artwork Verification. All Rights Reserved.

Result:

Thus Docker installation was verified, a custom Docker image was built using Nginx, and a container was successfully run to serve a simple web page.

Ex. No.:3	Flask App Containerization and Docker Hub Deployment
Date:06-08-25	

Objective

To create a Flask web application, containerize it using Docker, and deploy it to Docker Hub.

Tools/ Software Required

- Python 3.x – For running Flask applications
- Flask Framework – Web framework for Python
- Docker – For containerization
- Docker Hub Account – For image repository and distribution
- Code Editor – VS Code / Sublime / PyCharm
- Web Browser – To access the running application

Industry Relevance and Applications

- Containerization ensures applications run consistently across different environments.
- Docker is widely used in DevOps pipelines, cloud deployments, and microservices architecture.
- Flask is commonly used for lightweight web applications, APIs, and prototyping.
- Deploying containerized apps to Docker Hub enables easy sharing and collaboration in enterprise environments.

Description

Flask is a lightweight and open-source Python web framework designed for building web applications and RESTful APIs. Its micro-framework architecture provides developers with flexibility and simplicity, allowing them to create dynamic web pages using Jinja2 templates without imposing unnecessary dependencies.

Flask supports intuitive URL routing, making it easy to map web requests to Python functions, and can run locally on any system with Python installed.

Docker, is a containerization platform that packages applications along with all their dependencies, libraries, and configuration files into portable containers. This ensures that applications run consistently across different environments, eliminating conflicts between development, testing, and production setups.

By containerizing a Flask application with Docker, developers can guarantee that it operates reliably regardless of the underlying host system. Additionally, these Docker images can be shared and deployed globally via Docker Hub, enabling easy distribution, version control, and rapid deployment.

Command list with explanation:

- **pip install -r requirements.txt** : Installs Flask dependencies.
- **docker build -t flask-docker-app .** : Builds a Docker image named flask-docker-app from the Dockerfile.
- **docker run -p 5000:5000 flask-docker-app** : Runs the Docker container and maps host port 5000 to container port 5000.

Procedure

1. Create the Flask App, write app.py with a simple Flask application displaying an HTML page using render_template_string.
2. Create requirements.txt

```
Flask==2.2.5
```

3. Create Dockerfile

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python", "app.py"]
```

4. Build Docker Image

```
docker build -t flask-docker-app .
```

5. Run Docker Container

```
docker run -p 5000:5000 flask-docker-app
```

6. Test Local Application. Open browser: http://localhost:5000 (or http://localhost:5050 if using alternate port).
7. Push Image to Docker Hub

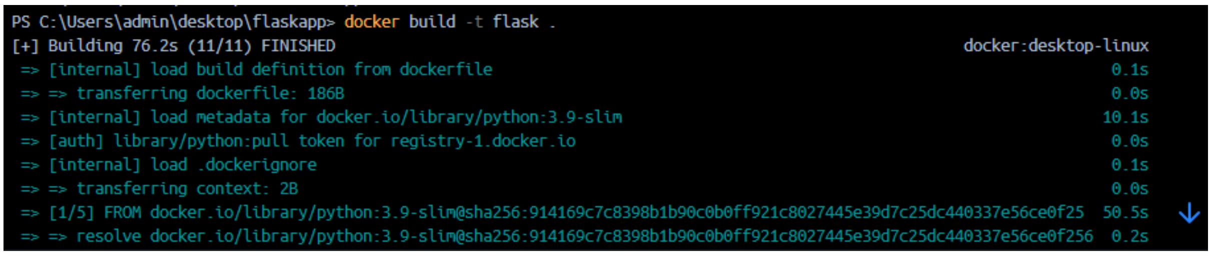
```
docker login

docker tag flask-docker-app <username>/flask-docker-app:latest

docker push <username>/flask-docker-app:latest
```

8. Run Image on Any Machine

```
docker pull <username>/flask-docker-app:latest

docker run -p 5050:5000 <username>/flask-docker-app:latest
```

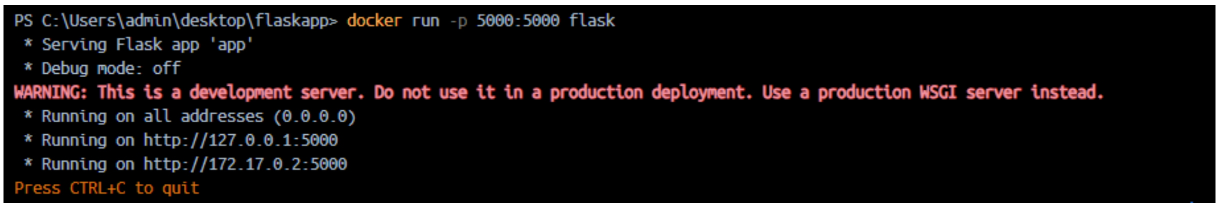
Sample Inputs:

Build Docker Image:



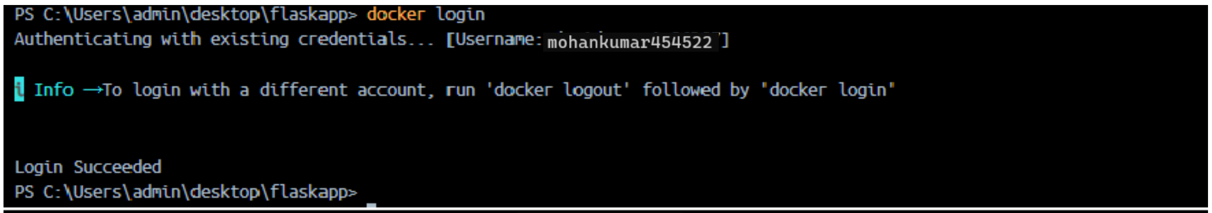
```
PS C:\Users\admin\desktop\flaskapp> docker build -t flask .
[+] Building 76.2s (11/11) FINISHED                                                              docker:desktop-linux
 => [internal] load build definition from dockerfile                                                             0.1s
 => => transferring dockerfile: 186B                                                                             0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim                                              10.1s
 => [auth] library/python:pull token for registry-1.docker.io                                                    0.0s
 => [internal] load .dockerignore                                                                                0.1s
 => => transferring context: 2B                                                                                  0.0s
 => [1/5] FROM docker.io/library/python:3.9-slim@sha256:914169c7c8398b1b90c0b0ff921c8027445e39d7c25dc440337e56ce0f25  50.5s
 => => resolve docker.io/library/python:3.9-slim@sha256:914169c7c8398b1b90c0b0ff921c8027445e39d7c25dc440337e56ce0f256  0.2s
```

Run Docker Container:

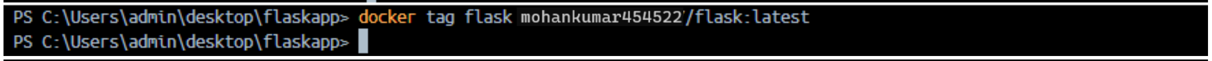


```
PS C:\Users\admin\desktop\flaskapp> docker run -p 5000:5000 flask
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
```

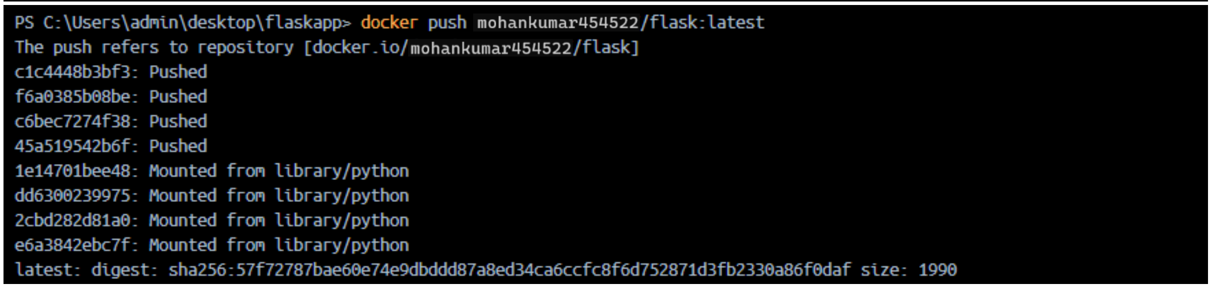
Push Image to Docker Hub:



```
PS C:\Users\admin\desktop\flaskapp> docker login
Authenticating with existing credentials... [Username: mohankumar454522 ]

i Info →To login with a different account, run 'docker logout' followed by 'docker login'


Login Succeeded
PS C:\Users\admin\desktop\flaskapp>
```

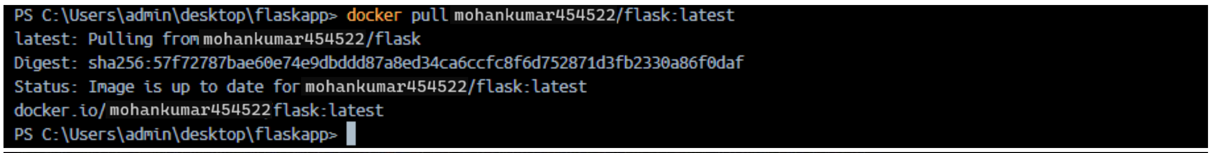


```
PS C:\Users\admin\desktop\flaskapp> docker tag flask mohankumar454522/flask:latest
PS C:\Users\admin\desktop\flaskapp>
```

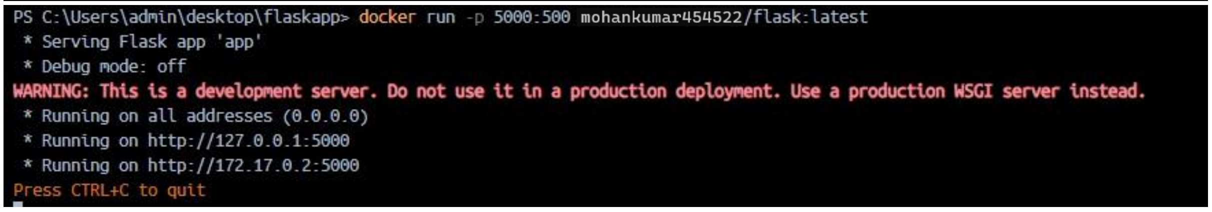


```
PS C:\Users\admin\desktop\flaskapp> docker push mohankumar454522/flask:latest
The push refers to repository [docker.io/mohankumar454522/flask]
c1c4448b3bf3: Pushed
f6a0385b08be: Pushed
c6bec7274f38: Pushed
45a519542b6f: Pushed
1e14701bee48: Mounted from library/python
dd6300239975: Mounted from library/python
2cbd282d81a0: Mounted from library/python
e6a3842ebc7f: Mounted from library/python
latest: digest: sha256:57f72787bae60e74e9dbddd87a8ed34ca6ccfc8f6d752871d3fb2330a86f0daf size: 1990
```

Run Image on Any Machine:

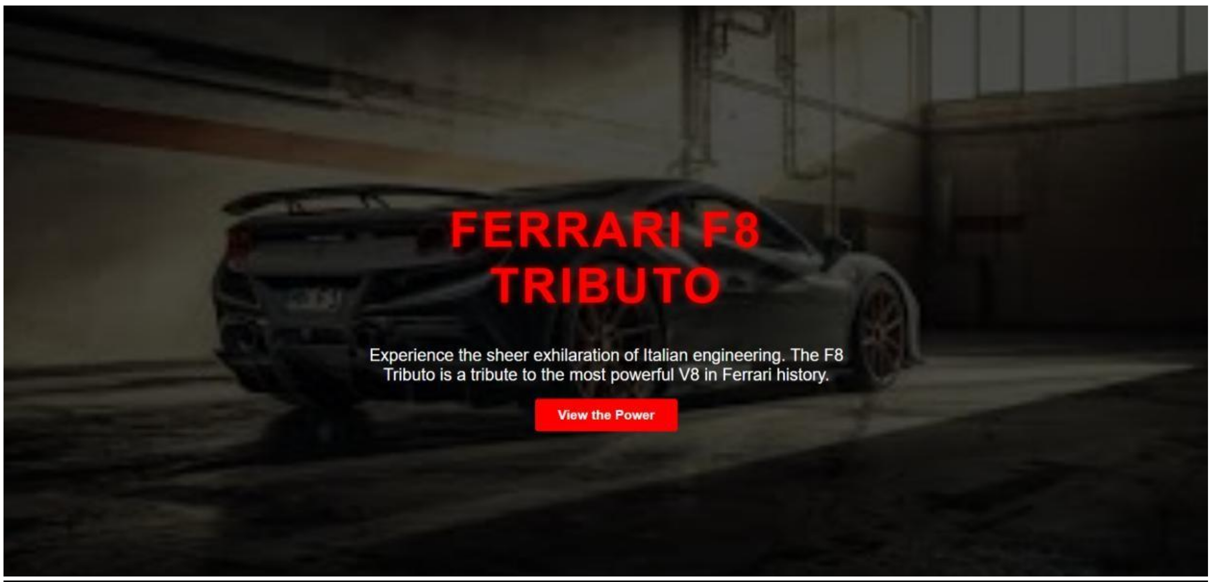


```
PS C:\Users\admin\desktop\flaskapp> docker pull mohankumar454522/flask:latest
latest: Pulling from mohankumar454522/flask
Digest: sha256:57f72787bae60e74e9dbddd87a8ed34ca6ccfc8f6d752871d3fb2330a86f0daf
Status: Image is up to date for mohankumar454522/flask:latest
docker.io/mohankumar454522flask:latest
PS C:\Users\admin\desktop\flaskapp>
```



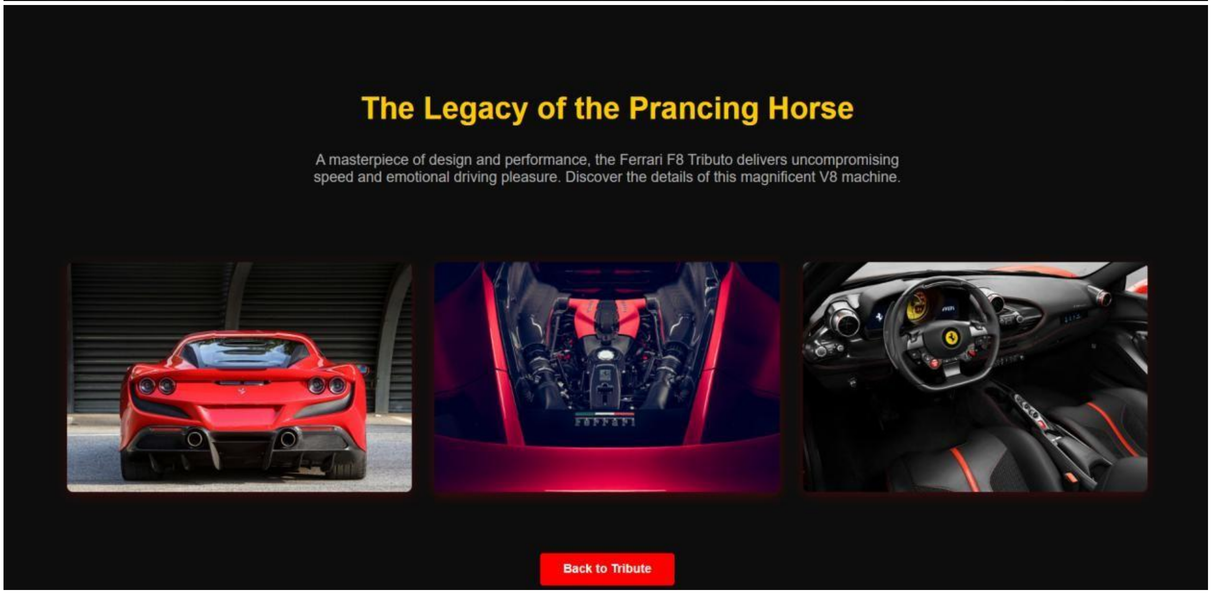
```
PS C:\Users\admin\desktop\flaskapp> docker run -p 5000:500 mohankumar454522/flask:latest
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
```

Output:



FERRARI F8 TRIBUTO

Experience the sheer exhilaration of Italian engineering. The F8 Tributo is a tribute to the most powerful V8 in Ferrari history.



The Legacy of the Prancing Horse

A masterpiece of design and performance, the Ferrari F8 Tributo delivers uncompromising speed and emotional driving pleasure. Discover the details of this magnificent V8 machine.

Result:

Thus successfully created a Flask web application, containerized it using Docker, and deployed it to Docker Hub.

Ex. No.:4	Cl Pipeline for Python App with GitHub and Docker Hub
Date:13-08-25	

Objective

To create a simple Python application, test it with pytest, build a Docker image, and automatically push the image to Docker Hub using a GitHub Actions CI/CD pipeline.

Tools/ Software Required

- **Python 3.11+** - for writing and testing the app
- **Git** - for version control
- **GitHub** - for hosting the repository and workflows
- **Docker & Docker Hub** - for containerization and publishing images
- **GitHub Actions** - for automation of testing and deployment

Industry Relevance and Applications

- **CI/CD Pipelines** are standard practice in modern software development.
- **Dockerization** enables consistent deployment across environments.
- **Automated Testing** ensures code quality before deployment.
- **Applications**: Web apps, APIs, microservices, data science pipelines — all use CI/CD with Docker in production systems.

Description

A CI/CD pipeline automates the processes of testing, building, and deploying software. Python applications can be tested automatically using pytest, which ensures that code changes do not introduce errors.

Containerization with Docker allows applications to run in a consistent environment across different systems, eliminating dependency issues and platform inconsistencies. GitHub Actions enables automation of workflows, including code checkout, dependency installation, test execution, Docker image building, and image deployment to container registries such as Docker Hub.

Versioning through Git tags allows each release to be uniquely identified, making it easier to track changes and maintain multiple versions of the software.

This experiment sets up a Continuous Integration (CI) pipeline that:

1. Runs unit tests with pytest.
2. Builds a Docker image of the Python app.
3. Pushes the Docker image to Docker Hub automatically on every commit or tag.

The project uses GitHub Actions workflows to achieve automation and follows best practices for DevOps.

Command list with explanation:

Command	Explanation
mkdir python-ci-docker-lab	Create a new project folder
cd python-ci-docker-lab	Navigate into the folder
python app.py	Run the Python app locally
pytest -q	Run unit tests
docker build -t yourname/python-ci-lab:local .	Build Docker image locally
docker run --rm yourname/python-ci-lab:local	Run the Docker container
git init	Initialize a new Git repository
git add .	Stage files for commit
git commit -m "message"	Commit changes
git push -u origin main	Push project to GitHub
git tag v1.0.0	Create a release tag
git push origin v1.0.0	Push the release tag to GitHub

Procedure

1. **Create project structure**

   Add app.py, __init__.py, tests/test_app.py, tests/__init__.py requirements.txt, Dockerfile, .gitignore.

   requirements.txt

   ```
   pytest==8.3.2
   ```

   Dockerfile

   ```
   FROM python:3.11-slim
   WORKDIR /app
   COPY requirements.txt .
   RUN pip install --no-cache-dir -r requirements.txt
   COPY . .
   CMD ["python", "app.py"]
   ```

   .gitignore

   ```
   __pycache__/
   .venv/
   .pytest_cache/
   .DS_Store
   *.pyc
   ```

2. **Write Python code**

   Define a simple add(a, b) function in app.py.

   app.py:

```
def add(a, b):
   return a + b
if _name___== "_main_":
   print("Hello from Python CI Lab!")
   print("2 + 3 =", add(2, 3))
```

3. **Write test cases**

   Verify correctness of add function by test cases using pytest.

   tests/test_app.py:

```
from app import add
def test_add():
   assert add(2, 3) == 5
   assert add(-1, 1) == 0
```

4. **Set up GitHub Actions workflow**

   Create .github/workflows/ci-dockerhub.yml.

   Define steps for checkout, install dependencies, run tests, build Docker image, and push to Docker Hub.

   .github/workflows/ci-dockerhub.yml.

```
name: ci-dockerhub
on:
  push:
    branches: [ "main" ]
  tags: [ "*" ]
  pull_request:
    branches: [ "main" ]
jobs:
  build-test-push:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.11'
      - name: Install deps
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
      - name: Run tests
        run: PYTHONPATH=. pytest -q
      - name: Docker meta
        id: meta
        uses: docker/metadata-action@v5
        with:
          images: ${{ secrets.DOCKERHUB_USERNAME }}/python-ci-lab
          tags: |
            type=raw,value=latest,enable={{is_default_branch}}
            type=sha,prefix=sha-,format=short
            type=ref,event=tag
      - name: Set up QEMU
        uses: docker/setup-qemu-action@v3
      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3
      - name: Login to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}
      - name: Build and push
        uses: docker/build-push-action@v6
        with:
          context: .
          push: true
          tags: ${{ steps.meta.outputs.tags }}
          labels: ${{ steps.meta.outputs.labels }}
          platforms: linux/amd64
```

5. **Test locally**

   Run Python script and Docker image on local system. Using commands:

```
python app.py
pytest -q
docker build -t yourname/python-ci-lab:local .
docker run --rm yourname/python-ci-lab:local
```

6. **Push to GitHub**

   Initialize Git repo, commit files, and push to GitHub.

7. **Configure GitHub Secrets**

   Go to your GitHub repository → click Settings → Secrets and variables → Actions → New repository secret → create two secrets:
   DOCKERHUB_USERNAME → your Docker Hub username
   DOCKERHUB_TOKEN → your Docker Hub access token

8. **Trigger Workflow**

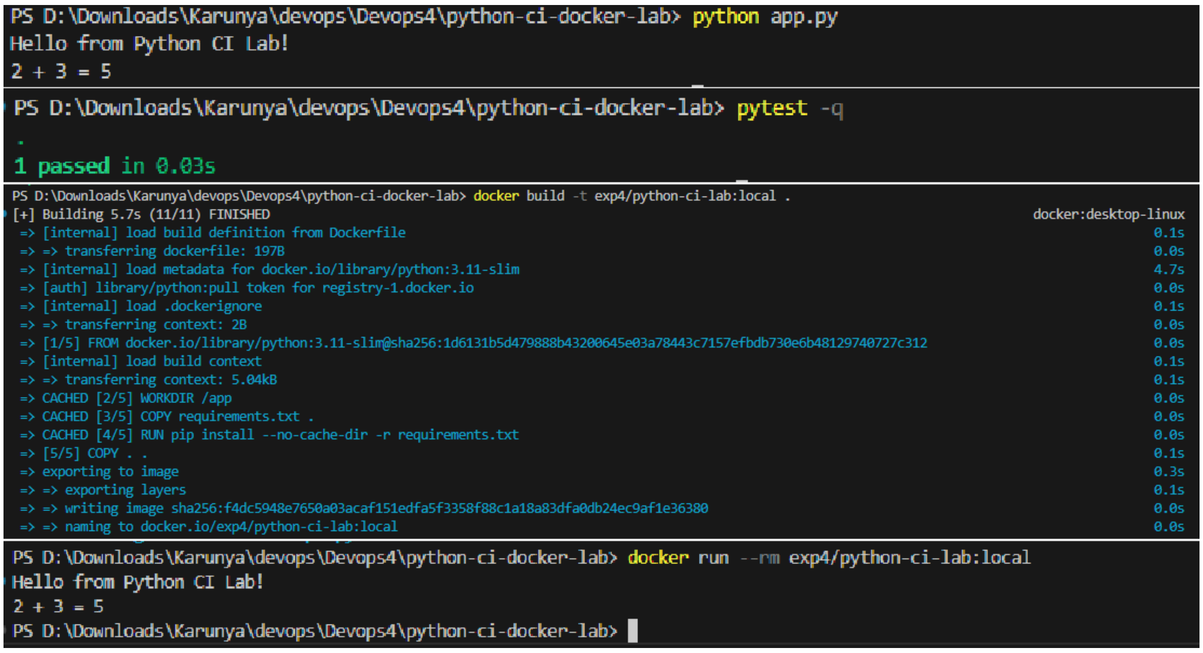
   Push code to main branch and check GitHub Actions logs.

9. **Verify Docker Hub**

   Confirm image docker.io/<DOCKERHUB_USERNAME>/python-ci-lab is available in Docker Hub.

Sample Inputs:

Local Testing:



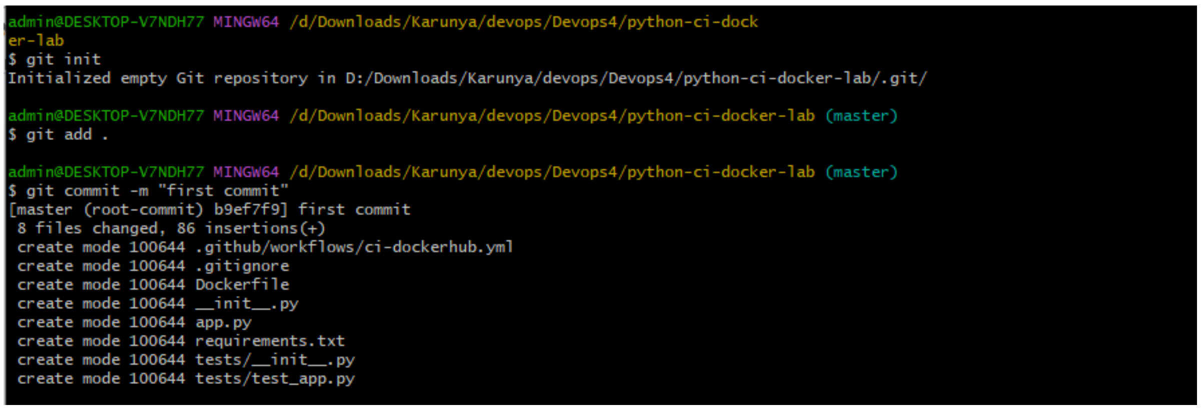
```
PS D:\Downloads\Karunya\devops\Devops4\python-ci-docker-lab> python app.py
Hello from Python CI Lab!
2 + 3 = 5
```

```
PS D:\Downloads\Karunya\devops\Devops4\python-ci-docker-lab> pytest -q
.
1 passed in 0.03s
```

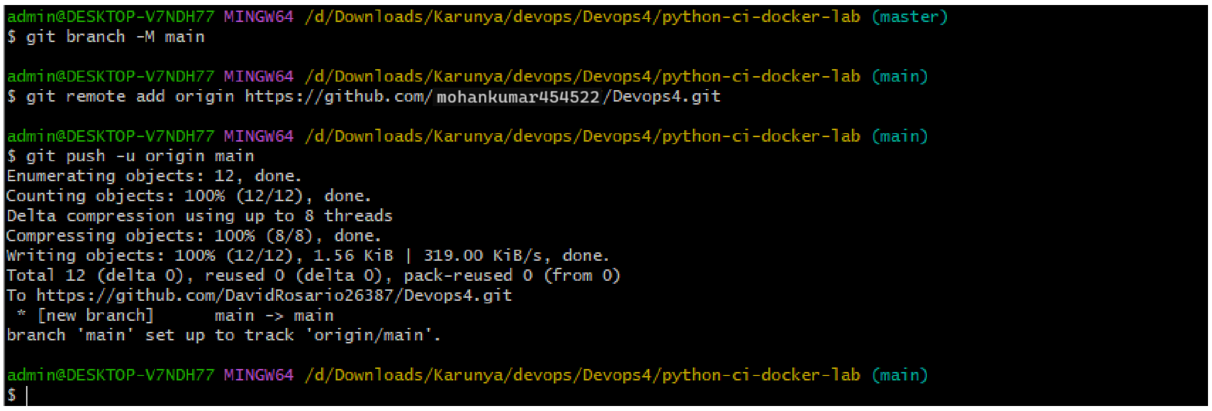
```
PS D:\Downloads\Karunya\devops\Devops4\python-ci-docker-lab> docker build -t exp4/python-ci-lab:local .
[+] Building 5.7s (11/11) FINISHED                                                         docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                       0.1s
 => => transferring dockerfile: 197B                                                                       0.0s
 => [internal] load metadata for docker.io/library/python:3.11-slim                                        4.7s
 => [auth] library/python:pull token for registry-1.docker.io                                              0.0s
 => [internal] load .dockerignore                                                                          0.1s
 => => transferring context: 2B                                                                            0.0s
 => [1/5] FROM docker.io/library/python:3.11-slim@sha256:1d6131b5d479888b43200645e03a78443c7157efbdb730e6b48129740727c312  0.0s
 => [internal] load build context                                                                          0.1s
 => => transferring context: 5.04kB                                                                        0.1s
 => CACHED [2/5] WORKDIR /app                                                                              0.0s
 => CACHED [3/5] COPY requirements.txt .                                                                   0.0s
 => CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt                                        0.0s
 => [5/5] COPY . .                                                                                         0.1s
 => exporting to image                                                                                     0.3s
 => => exporting layers                                                                                    0.1s
 => => writing image sha256:f4dc5948e7650a03acaf151edfa5f3358f88c1a18a83dfa0db24ec9af1e36380               0.0s
 => => naming to docker.io/exp4/python-ci-lab:local                                                        0.0s
```

```
PS D:\Downloads\Karunya\devops\Devops4\python-ci-docker-lab> docker run --rm exp4/python-ci-lab:local
Hello from Python CI Lab!
2 + 3 = 5
PS D:\Downloads\Karunya\devops\Devops4\python-ci-docker-lab>
```

Initialize Git & push to GitHub:

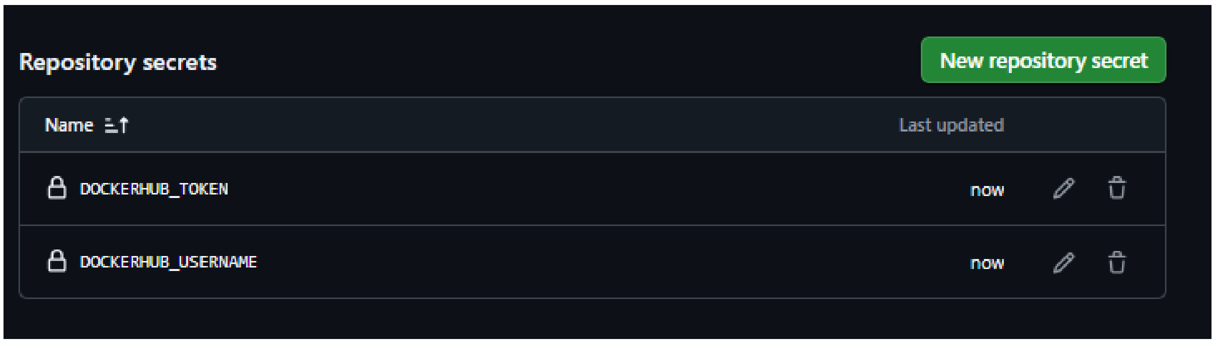


```
admin@DESKTOP-V7NDH77 MINGW64 /d/Downloads/Karunya/devops/Devops4/python-ci-dock
er-lab
$ git init
Initialized empty Git repository in D:/Downloads/Karunya/devops/Devops4/python-ci-docker-lab/.git/

admin@DESKTOP-V7NDH77 MINGW64 /d/Downloads/Karunya/devops/Devops4/python-ci-docker-lab (master)
$ git add .

admin@DESKTOP-V7NDH77 MINGW64 /d/Downloads/Karunya/devops/Devops4/python-ci-docker-lab (master)
$ git commit -m "first commit"
[master (root-commit) b9ef7f9] first commit
 8 files changed, 86 insertions(+)
 create mode 100644 .github/workflows/ci-dockerhub.yml
 create mode 100644 .gitignore
 create mode 100644 Dockerfile
 create mode 100644 __init__.py
 create mode 100644 app.py
 create mode 100644 requirements.txt
 create mode 100644 tests/__init__.py
 create mode 100644 tests/test_app.py
```



```
admin@DESKTOP-V7NDH77 MINGW64 /d/Downloads/Karunya/devops/Devops4/python-ci-docker-lab (master)
$ git branch -M main

admin@DESKTOP-V7NDH77 MINGW64 /d/Downloads/Karunya/devops/Devops4/python-ci-docker-lab (main)
$ git remote add origin https://github.com/mohankumar454522/Devops4.git

admin@DESKTOP-V7NDH77 MINGW64 /d/Downloads/Karunya/devops/Devops4/python-ci-docker-lab (main)
$ git push -u origin main
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 8 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (12/12), 1.56 KiB | 319.00 KiB/s, done.
Total 12 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/DavidRosario26387/Devops4.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

admin@DESKTOP-V7NDH77 MINGW64 /d/Downloads/Karunya/devops/Devops4/python-ci-docker-lab (main)
$ |
```

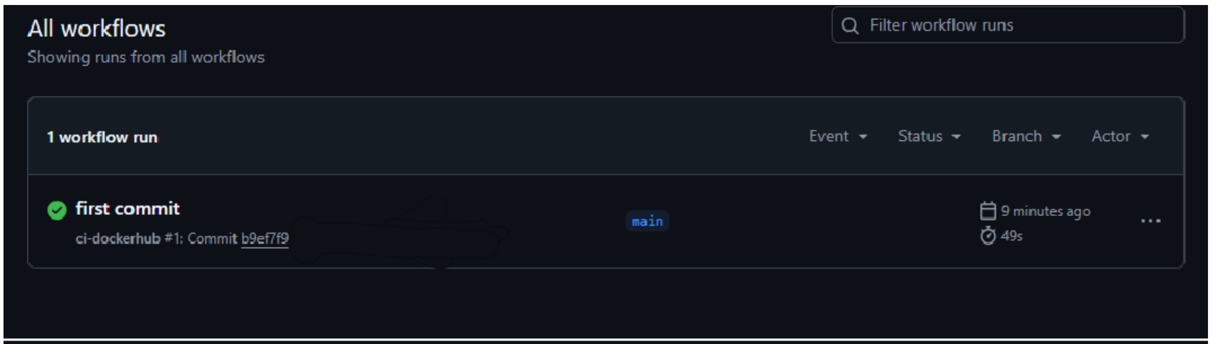
Add GitHub Secrets:



Repository secrets

Name	Last updated
DOCKERHUB_TOKEN	now
DOCKERHUB_USERNAME	now

Workflow run logs:

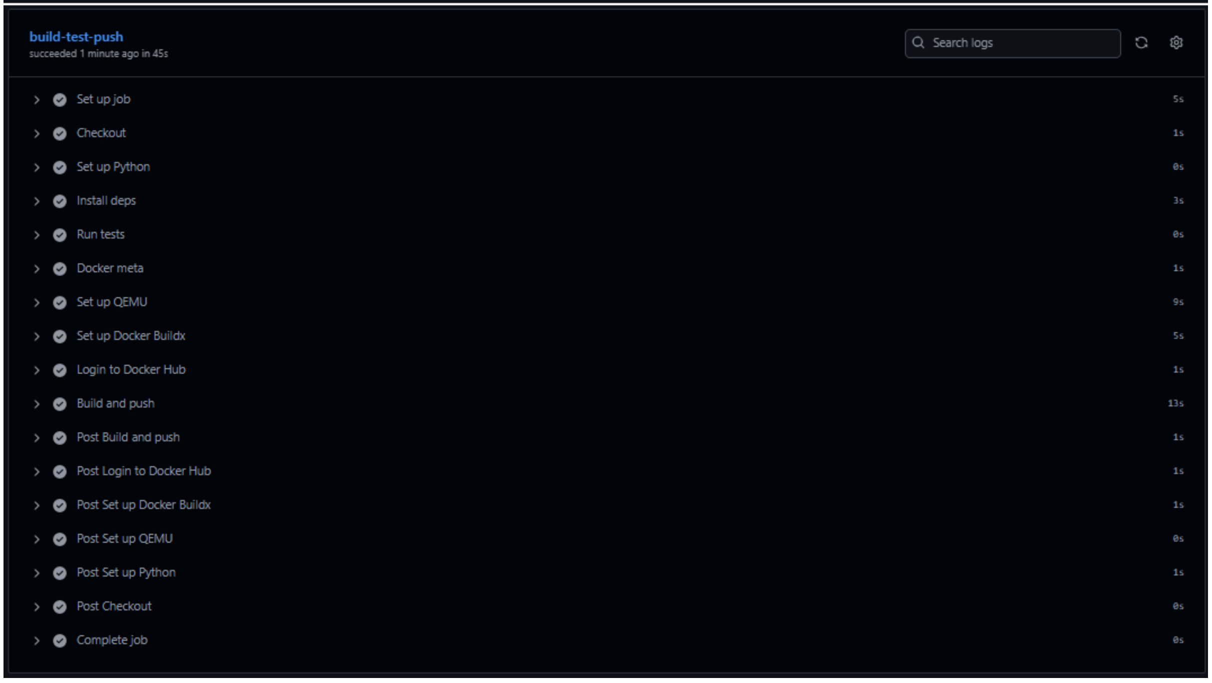


All workflows

Showing runs from all workflows

1 workflow run

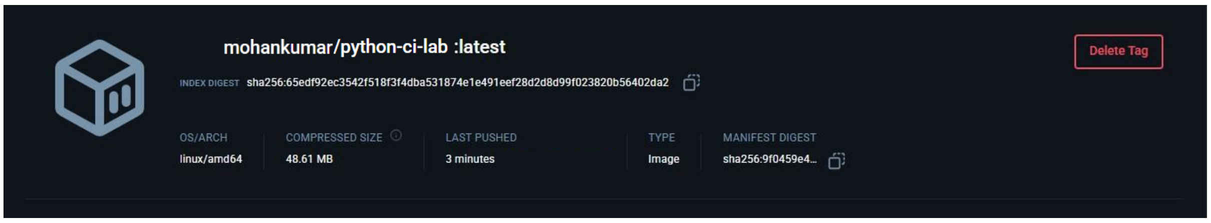
first commit main 9 minutes ago
ci-dockerhub #1: Commit b9ef7f9 49s



build-test-push
succeeded 1 minute ago in 45s

```
Set up job                 5s
Checkout                   1s
Set up Python              0s
Install deps               3s
Run tests                  0s
Docker meta                1s
Set up QEMU                9s
Set up Docker Buildx       5s
Login to Docker Hub        1s
Build and push            13s
Post Build and push        1s
Post Login to Docker Hub   1s
Post Set up Docker Buildx  1s
Post Set up QEMU           0s
Post Set up Python         1s
Post Checkout              0s
Complete job               0s
```

Verify Image on Docker Hub:

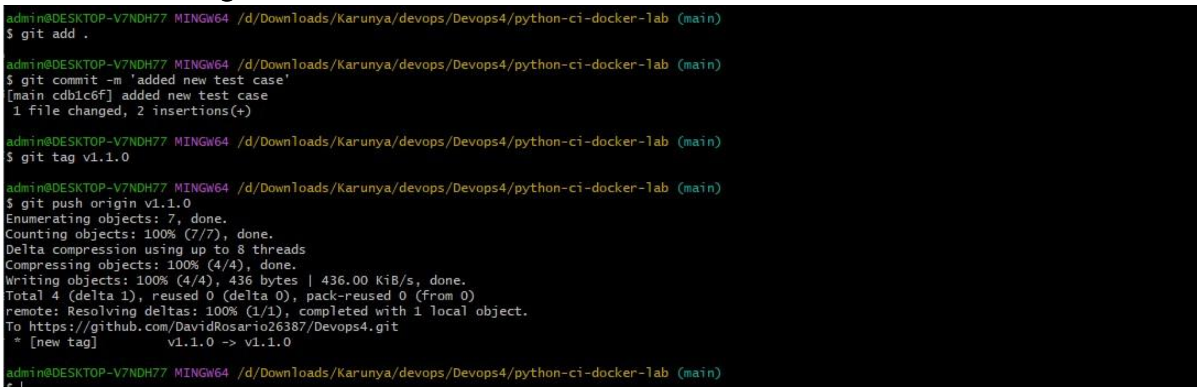


mohankumar/python-ci-lab :latest

INDEX DIGEST sha256:65edf92ec3542f518f3f4dba531874e1e491eef28d2d8d99f023820b56402da2

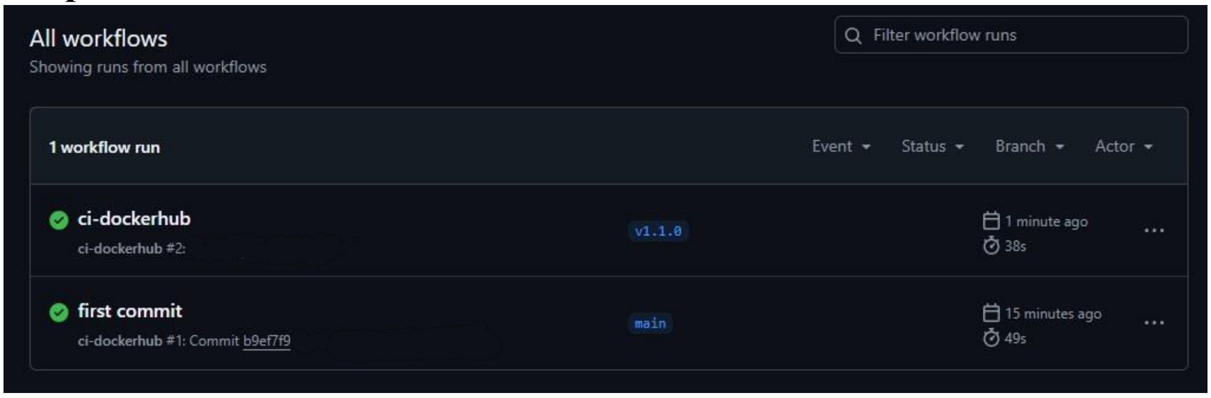
OS/ARCH	COMPRESSED SIZE	LAST PUSHED	TYPE	MANIFEST DIGEST
linux/amd64	48.61 MB	3 minutes	Image	sha256:9f0459e4…

Release with a tag:



```
admin@DESKTOP-V7NDH77 MINGW64 /d/Downloads/Karunya/devops/Devops4/python-ci-docker-lab (main)
$ git add .

admin@DESKTOP-V7NDH77 MINGW64 /d/Downloads/Karunya/devops/Devops4/python-ci-docker-lab (main)
$ git commit -m 'added new test case'
[main cdb1c6f] added new test case
 1 file changed, 2 insertions(+)

admin@DESKTOP-V7NDH77 MINGW64 /d/Downloads/Karunya/devops/Devops4/python-ci-docker-lab (main)
$ git tag v1.1.0

admin@DESKTOP-V7NDH77 MINGW64 /d/Downloads/Karunya/devops/Devops4/python-ci-docker-lab (main)
$ git push origin v1.1.0
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 436 bytes | 436.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/DavidRosario26387/Devops4.git
 * [new tag]         v1.1.0 -> v1.1.0

admin@DESKTOP-V7NDH77 MINGW64 /d/Downloads/Karunya/devops/Devops4/python-ci-docker-lab (main)
```

Output:



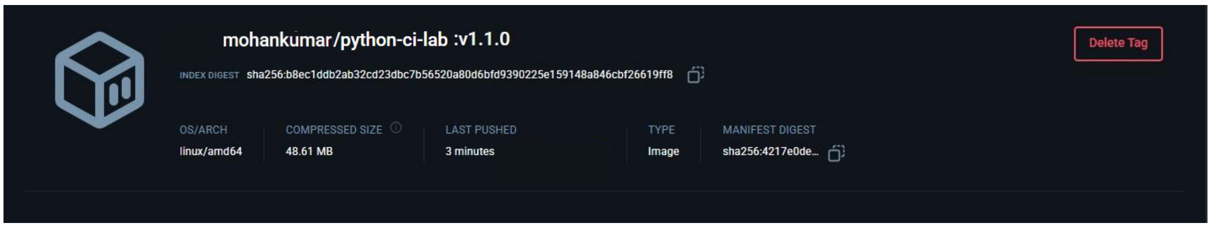
All workflows

Showing runs from all workflows

1 workflow run

ci-dockerhub
ci-dockerhub #2: v1.1.0 1 minute ago 38s

first commit
ci-dockerhub #1: Commit b9ef7f9 main 15 minutes ago 49s



mohankumar/python-ci-lab :v1.1.0

INDEX DIGEST sha256:b8ec1ddb2ab32cd23dbc7b56520a80d6bfd9390225e159148a846cbf26619ff8

OS/ARCH	COMPRESSED SIZE	LAST PUSHED	TYPE	MANIFEST DIGEST
linux/amd64	48.61 MB	3 minutes	Image	sha256:4217e0de…

Result:

The CI/CD pipeline was successfully implemented, with automated testing of the Python application using pytest, automatic building of a Docker image, and deployment of the image to Docker Hub through GitHub Actions.

Ex. No.:5	Jenkins deployment Using Docker
Date:03-09-25	

Objective

The objective of this lab is to deploy and manage Jenkins inside Docker and use it to automate the generation and publication of a simple static website.

Tools/ Software Required

- Laptop or VM with Docker and Docker Compose installed.
- Internet access to pull Docker images.
- Jenkins running inside Docker (jenkins/jenkins:lts).
- Basic knowledge of Git and command line.

Industry Relevance and Applications

- **CI/CD Automation**: Automating build, test, and deployment pipelines.
- **Infrastructure as Code**: Using Docker and Docker Compose for reproducible Jenkins setups.
- **Artifact Management**: Storing and archiving generated files for future builds.
- **DevOps Practices**: Integrating build pipelines with visualization (HTML Publisher).
- **Web Deployment**: Demonstrating how Jenkins can generate and serve a static website.

Description

- **Jenkins**
  - Open-source automation server for CI/CD
  - Automates development tasks: build, test, deployment, and artifact management.
  - Pipelines are scripted using Groovy, organized into stages for workflow control.
- **Docker**
  - Platform for containerization, enabling applications to run consistently across environments.
  - Isolates dependencies and simplifies deployment.
- **Docker Compose**
  - Tool for defining and managing multi-container applications via a YAML configuration.
  - Facilitates orchestration, volume management, and service networking for reproducible setups.
- **Benefits**
  - Provides reproducible, automated, and traceable software delivery.
  - Simplifies setup and scaling of CI/CD pipelines in containerized environments.

Command list with explanation:

Command	Explanation
mkdir folder_name	Create a new project folder
cd folder_name	Navigate into the folder
docker compose up -d	Run Jenkins in detached mode
Docker exec jenkins cat /var/jenkins_home/ secrets/ initialAdminPassword	Get Jenkins admin password

Procedure

Set up Jenkins with Docker Compose.

1. Create a folder for the lab using the commands

```
mkdir ~/jenkins-static-site
cd ~/jenkins-static-site
```

2. Create a docker-compose.yml file

```
version: '3.8'
services:
  jenkins:
    image: jenkins/jenkins:lts
    container_name: jenkins
    restart: unless-stopped
    ports:
      - "8080:8080"
      - "50000:50000"
    volumes:
      - jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
volumes:
  jenkins_home:
    driver: local
```

3. Start Jenkins using the command:

```
docker compose up -d
```

4. Get the initial admin password using the command:

```
docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

5. Open Jenkins in the browser -> http://localhost:8080
6. Complete setup → install suggested plugins → create admin user.

Install HTML Publisher Plugin

1. In Jenkins: Manage Jenkins → Plugins → Available Plugins.
2. Search for HTML Publisher and install it.
3. Restart Jenkins if prompted.

Create the Pipeline Job

1. In Jenkins, click New Item → select Pipeline → name it karunya-site-pipeline.
2. Scroll to Pipeline → Definition and select Pipeline script.
3. Paste the following script:

```
pipeline {
    agent any
    stages {
        stage('Generate site') {
            steps {
                script {
                    def currentYear = new Date().format('yyyy')
                    sh 'rm -rf site || true'
                    sh 'mkdir -p site/assets'
                    writeFile file: 'site/index.html', text: '''
<meta charset="utf-8"/>
<meta name="viewport" content="width=device-width, initial-scale=1"/>
<title>Karunya University — Simple Site</title>
<link rel="stylesheet" href="assets/style.css"/>
</head>
<body>
<header class="site-header">
<div class="container">
<h1>Karunya University</h1>
<p class="tagline">Values — Knowledge — Service</p>
</div>
</header>
<main class="container">
<section class="card">
<h2>About Karunya University</h2>
<p>Karunya Institute of Technology and Sciences is a leading
centre for higher education in India.</p>
</section>
</main>
<footer class="site-footer container">
<p>&copy; ''' + currentYear + ''' Karunya University —
Generated by Jenkins</p>
</footer>
</body>
</html>
'''
                    writeFile file: 'site/assets/style.css', text: '''
body { font-family: Arial, sans-serif; margin:0; }
.container { max-width:800px; margin:0 auto; padding:20px; }
.site-header { background:#0b3d91; color:#fff; padding:20px; }
.tagline { font-size:14px; opacity:0.8; }
.card { background:#fff; border:1px solid #ddd; padding:15px; margin:
15px 0; border-radius:6px; }
.site-footer { text-align:center; font-size:12px; color:#555; margin-top:20px; }
'''
                }
            }
        }
        stage('Publish site') {
            steps {
                publishHTML([
                    allowMissing: false,
                    alwaysLinkToLastBuild: true,
                    keepAll: true,
                    reportDir: 'site',
                    reportFiles: 'index.html',
                    reportName: 'Karunya University - Simple Site'
                ])
            }
        }
    }
    post {
        always {
            archiveArtifacts artifacts: 'site/**', fingerprint: true
        }
    }
}
```

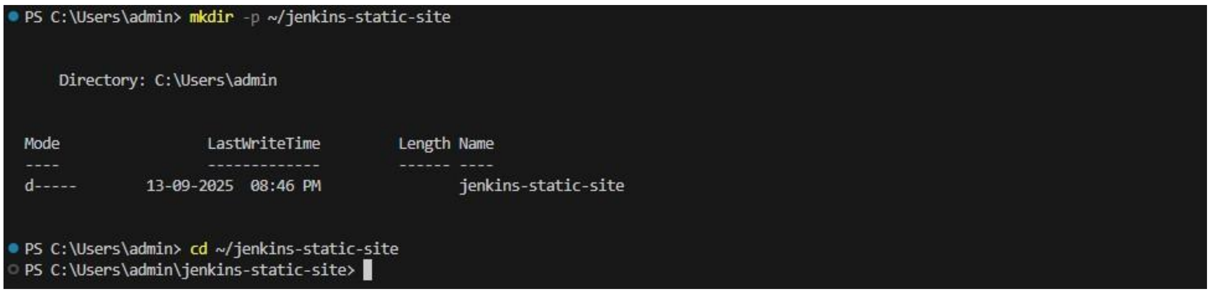
4. Save the pipeline

Run and View the Website

1. Click Build Now.
2. After success, go to the build page.
3. On the left menu, click Karunya University - Simple Site.
4. You should see the static site rendered inside Jenkins.
5. Alternatively, download the archived artifact (the site/ folder) and open index.html locally.

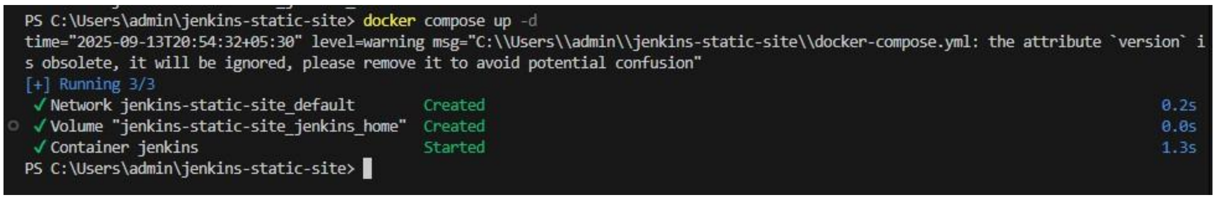
Sample Inputs:

Create project folder and navigate to it



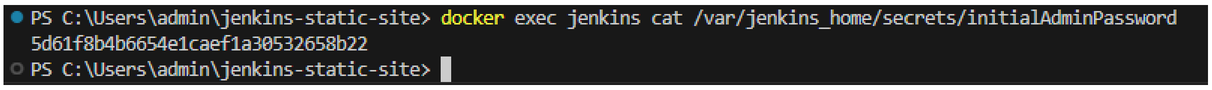
```
PS C:\Users\admin> mkdir -p ~/jenkins-static-site


    Directory: C:\Users\admin


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        13-09-2025  08:46 PM                jenkins-static-site


PS C:\Users\admin> cd ~/jenkins-static-site
PS C:\Users\admin\jenkins-static-site>
```

Start jenkins



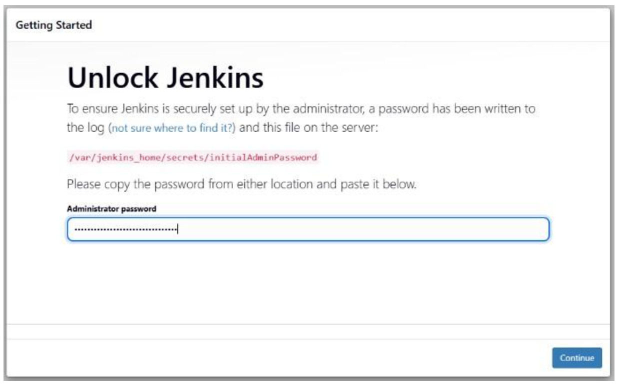
```
PS C:\Users\admin\jenkins-static-site> docker compose up -d
time="2025-09-13T20:54:32+05:30" level=warning msg="C:\\Users\\admin\\jenkins-static-site\\docker-compose.yml: the attribute `version` i
s obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 3/3
 ✓Network jenkins-static-site_default       Created                                                                              0.2s
 ✓Volume "jenkins-static-site_jenkins_home"  Created                                                                              0.0s
 ✓Container jenkins                         Started                                                                              1.3s
PS C:\Users\admin\jenkins-static-site>
```

Get the initial admin password



```
PS C:\Users\admin\jenkins-static-site> docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword
5d61f8b4b6654e1caef1a30532658b22
PS C:\Users\admin\jenkins-static-site>
```

Finish the setup



Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

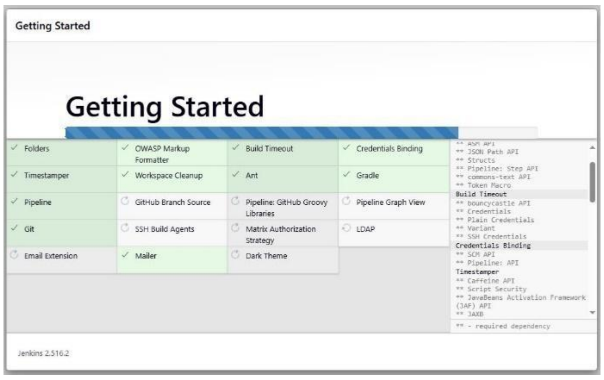
/var/jenkins_home/secrets/initialAdminPassword

Please copy the password from either location and paste it below.

Administrator password

································

Continue



Getting Started

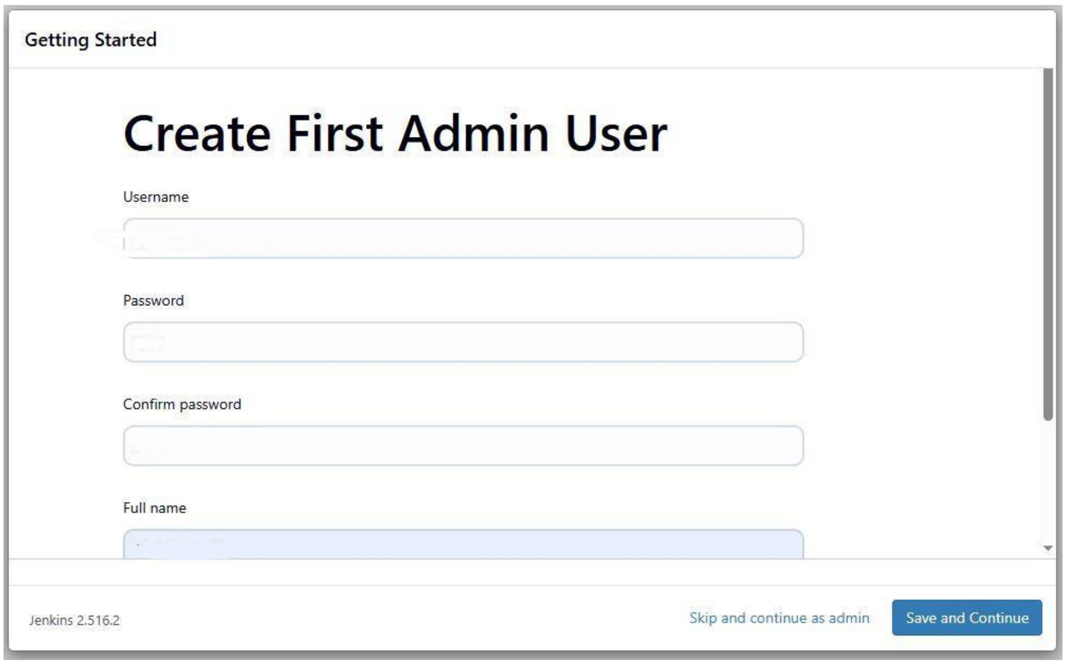
Getting Started

✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✓ Credentials Binding
✓ Timestamper	✓ Workspace Cleanup	✓ Ant	✓ Gradle
✓ Pipeline	GitHub Branch Source	Pipeline: GitHub Groovy Libraries	Pipeline Graph View
✓ Git	SSH Build Agents	Matrix Authorization Strategy	LDAP
Email Extension	✓ Mailer	Dark Theme	

```
** ASM API
** JSON Path API
** Structs
** Pipeline: Step API
** commons-text API
** Token Macro
Build Timeout
** bouncycastle API
** Credentials
** Plain Credentials
** Variant
** SSH Credentials
Credentials Binding
** SCM API
** Pipeline: API
Timestamper
** Caffeine API
** Script Security
** JavaBeans Activation Framework (JAF) API
** JAXB
** - required dependency
```

Jenkins 2.516.2

Create Admin user


Getting Started

Create First Admin User

Username

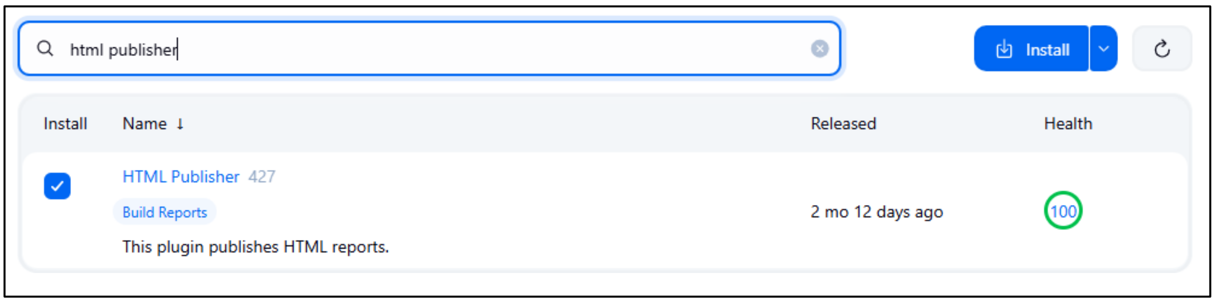
Password

Confirm password

Full name

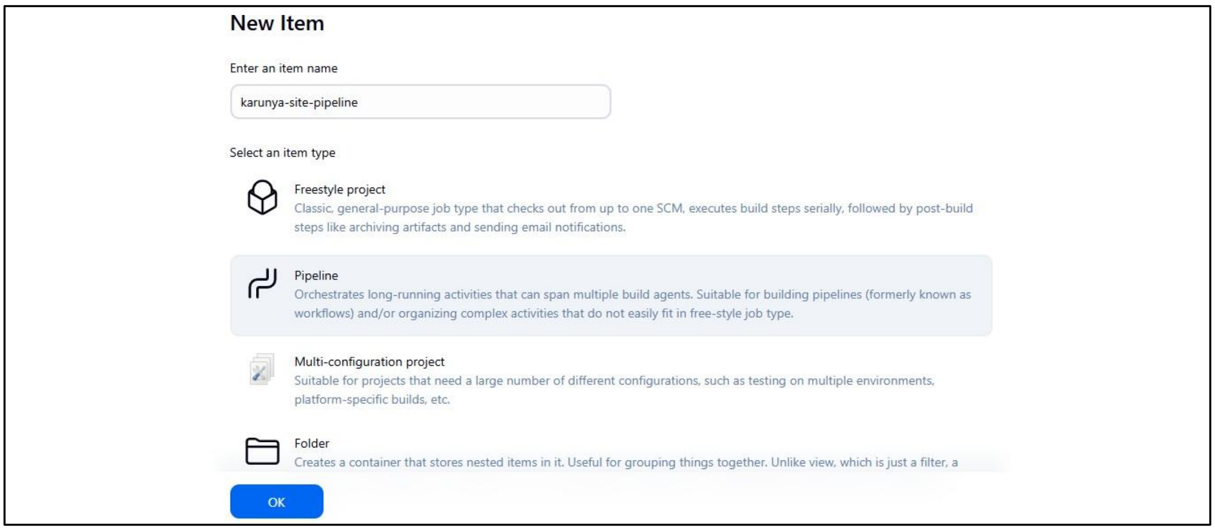
Jenkins 2.516.2

Install HTML Publisher plugin


html publisher

Install	Name ↓	Released	Health
✓	HTML Publisher 427 Build Reports This plugin publishes HTML reports.	2 mo 12 days ago	100

Create New Pipeline item


New Item

Enter an item name

karunya-site-pipeline

Select an item type

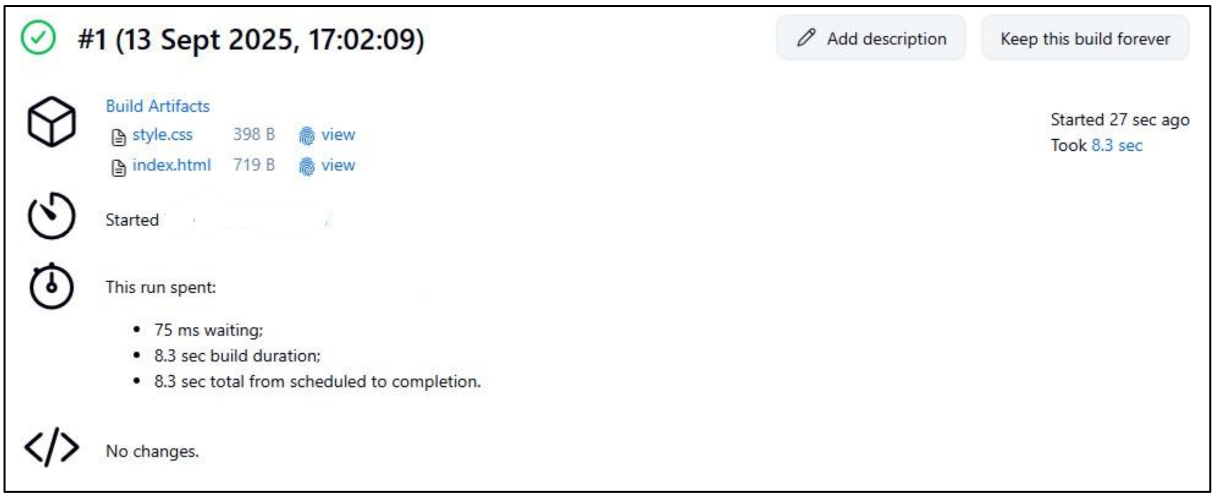
Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

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

Build the Pipeline



#1 (13 Sept 2025, 17:02:09)

Add description | Keep this build forever

Build Artifacts
- style.css 398 B view
- index.html 719 B view

Started 27 sec ago
Took 8.3 sec

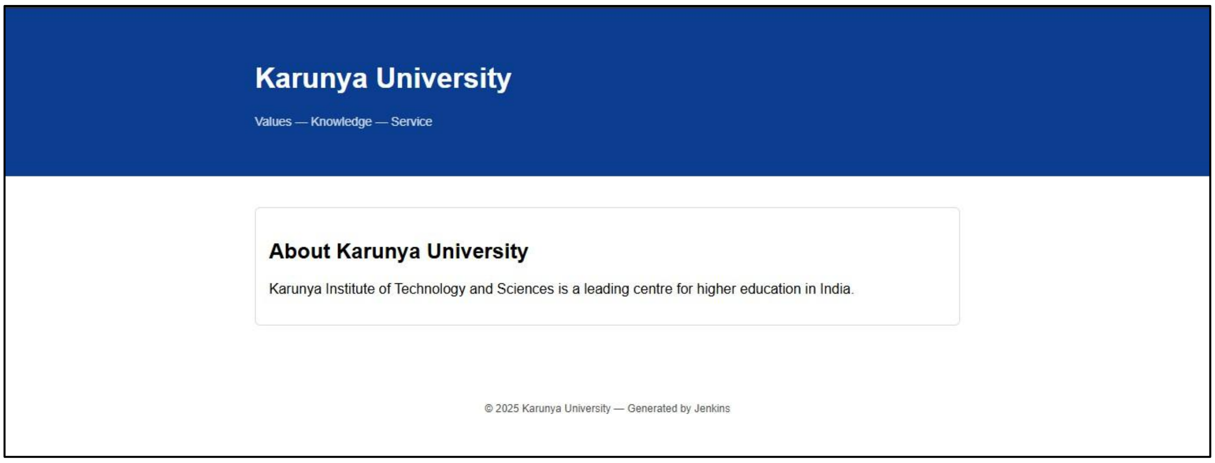
Started

This run spent:
- 75 ms waiting;
- 8.3 sec build duration;
- 8.3 sec total from scheduled to completion.

No changes.

Output:

Static Site rendered inside Jenkins



Karunya University

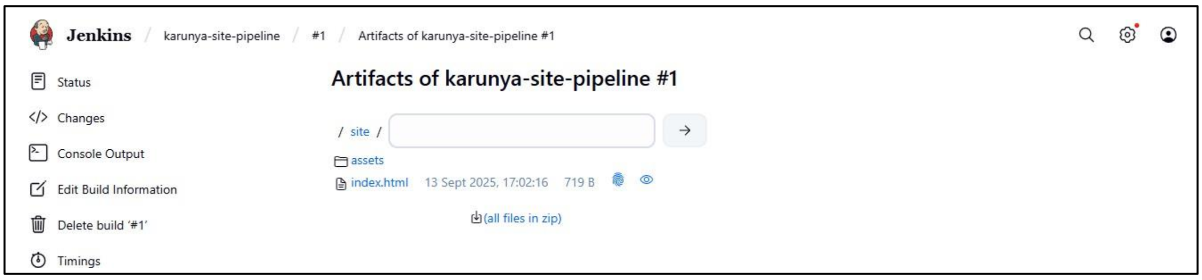
Values — Knowledge — Service

About Karunya University

Karunya Institute of Technology and Sciences is a leading centre for higher education in India.

© 2025 Karunya University — Generated by Jenkins

Download the archieved Artifact



Jenkins / karunya-site-pipeline / #1 / Artifacts of karunya-site-pipeline #1

Artifacts of karunya-site-pipeline #1

/ site /

assets
index.html 13 Sept 2025, 17:02:16 719 B

(all files in zip)

Result:

Jenkins was successfully deployed inside Docker, and a static website was generated, published, and archived using a Jenkins pipeline.

Ex. No.:6	Manage Kubernetes Resources Using CLI
Date:10-09-25	

Objective

The objective of this lab is to install the tools and manage basic Kubernetes Resources Using kubectl.

Tools/ Software Required

- Laptop/desktop with at least 4 GB RAM and internet access.
- One of these OS options: Ubuntu/Debian Linux, macOS, or Windows 10/11.
- Basic terminal or PowerShell familiarity.

Industry Relevance and Applications

- **Cloud-Native Management:** Kubernetes enables deployment and orchestration of microservices on cloud platforms like AWS, Azure, and GCP.
- **Scalability & Reliability:** Supports easy scaling and ensures high availability for production workloads.
- **DevOps Integration:** Works seamlessly with CI/CD tools like Jenkins and ArgoCD for automated deployments.
- **Infrastructure Standardization:** Promotes Infrastructure as Code (IaC) for consistent, automated, and portable environments.

Description

- **Docker:** Acts as the containerization platform to package and run applications consistently across environments.
- **Minikube:** Provides a lightweight, single-node Kubernetes cluster for local development and experimentation.
- **kubectl:** The primary CLI tool for interacting with the Kubernetes API to create, view, and manage cluster resources.
- **Pods:** The basic deployable units in Kubernetes, each encapsulating one or more containers running an application.

- **Deployments:** Manage multiple replicas of Pods, ensuring high availability and enabling rolling updates and rollbacks.
- **Services (NodePort):** Allow applications running inside the cluster to be accessed externally through a stable network endpoint.
- **Scaling:** Demonstrates horizontal scaling by increasing or decreasing the number of running Pod replicas based on load.
- **Cluster Management:** Involves verifying node status, monitoring workloads, and cleaning up resources to maintain cluster stability.
- **Automation and Reliability:** Shows how Kubernetes automates deployment, scaling, and recovery—key principles of cloud-native and DevOps workflows.

Command list with explanation:

Command	**Explanation**
kubectl version --client	Checks if the Kubernetes CLI tool (kubectl) is installed and displays the client version.
minikube start --driver=docker	Starts a local Kubernetes cluster using Docker as the driver.
minikube status	Displays the current status of the Minikube cluster components.
kubectl get nodes	Lists all nodes in the Kubernetes cluster and shows their readiness state.
kubectl apply -f nginx-pod.yaml	Creates resources (like Pods) in the cluster from a YAML configuration file.
kubectl get pods	Displays all running Pods and their current statuses.
kubectl describe pod nginx-pod	Shows detailed information about a specific Pod, including events, containers, and status.
kubectl port-forward pod/nginx-pod 8080:80	Forwards local port 8080 to the Pod’s port 80, allowing local browser access.
kubectl delete pod nginx-pod	Deletes the specified Pod from the cluster.
kubectl create deployment my-nginx --image=nginx	Creates a Deployment resource running the Nginx image.
kubectl get deployments	Lists all existing Deployments and their replica status.
kubectl get pods -l app=my-nginx	Filters and lists Pods with the label app=my-nginx.

Command	Explanation
kubectl scale deployment my-nginx --replicas=3	Scales the Deployment to run three replicas of the Pod.
kubectl rollout status deployment/my-nginx	Monitors the progress of the rolling update for the Deployment.
kubectl rollout undo deployment/my-nginx	Rolls back the Deployment to the previous working version if an update fails.
kubectl expose deployment my-nginx --type=NodePort --port=80	Exposes the Deployment as a Service, making it accessible externally via a node port.
kubectl get svc	Lists all Services running in the cluster and their corresponding ports.
minikube service my-nginx --url	Retrieves the accessible URL for the exposed service running in Minikube.
kubectl delete svc my-nginx	Deletes the Service resource created for the Deployment.
kubectl delete deployment my-nginx	Removes the Deployment and its associated Pods.
kubectl delete all --all -n default	Deletes all resources (Pods, Services, Deployments, etc.) in the default namespace.
minikube stop	Stops the Minikube cluster to free up system resources.
minikube delete	Completely removes the Minikube cluster and all its configurations.

Procedure:

Setup and installation:

1. Install **Docker Desktop** (Windows/macOS) or **Docker Engine** (Linux). And Verify installation using the command:

```
docker --version
```

2. Install **kubectl** (Kubernetes CLI). On Windows (via Chocolatey) using the command :

```
choco install kubernetes-cli
```

3. Verify installation:

```
kubectl version --client
```

4. Install and start Minikube:

```
choco install minikube

minikube start --driver=docker
```

5. Check cluster status using the command:

```
minikube status

kubectl get nodes
```

Create and manage a pod:

1. Create a new YAML file named **nginx-pod.yaml** with the following configuration:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```

2. Apply the configuration to create the Pod:

```
kubectl apply -f nginx-pod.yaml
```

3. Check the Pod status:

```
kubectl get pods
```

4. View detailed information about the Pod:

```
kubectl describe pod nginx-pod
```

5. Access the Pod in your browser by port forwarding:

```
kubectl port-forward pod/nginx-pod 8080:80
```

   Then open **http://localhost:8080**.

6. After testing, delete the Pod:

```
kubectl delete pod nginx-pod
```

Create and manage a Deployment:

1. Create a Deployment for Nginx:

```
kubectl create deployment my-nginx --image=nginx
```

2. View the running Deployments and Pods:

```
kubectl get deployments

kubectl get pods -l app=my-nginx
```

3. Scale the Deployment to 3 replicas:

```
kubectl scale deployment my-nginx --replicas=3

kubectl get pods
```

4. Update the Nginx image version (Rolling Update):

```
kubectl set image deployment/my-nginx nginx=nginx:1.25

kubectl rollout status deployment/my-nginx
```

5. If the update fails, rollback to the previous version:

```
kubectl rollout undo deployment/my-nginx
```

Expose the deployment as a service:

1. Expose the deployment externally using a NodePort service:

```
kubectl expose deployment my-nginx --type=NodePort --port=80
```

2. Check the service details:

```
kubectl get svc
```

3. Get the accessible service URL using Minikube:

```
minikube service my-nginx --url
```

Open the displayed URL in your browser to confirm successful deployment.

4. To remove the service:

```
kubectl delete svc my-nginx
```

Cleanup:

1. Delete all created resources to reset the cluster:

   kubectl delete deployment my-nginx

   kubectl delete svc my-nginx

   kubectl delete all --all -n default

2. Stop and delete the Minikube cluster:

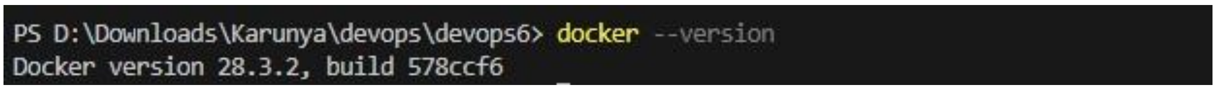
   minikube stop

   minikube delete

Sample Inputs:

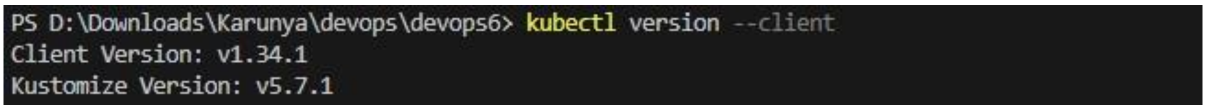
Verify installations:

Docker verification



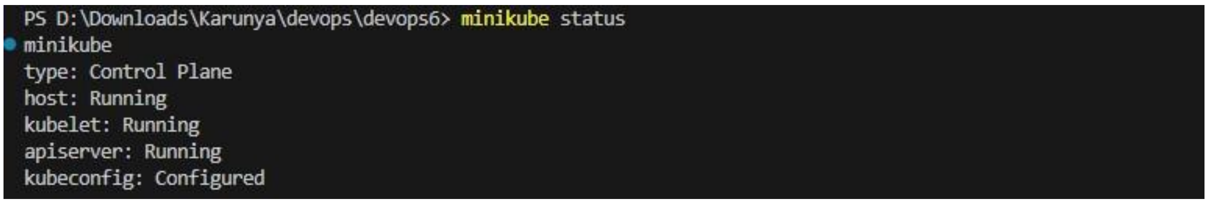
```
PS D:\Downloads\Karunya\devops\devops6> docker --version
Docker version 28.3.2, build 578ccf6
```

Kubectl verification



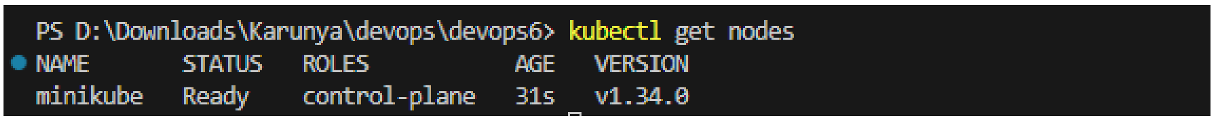
```
PS D:\Downloads\Karunya\devops\devops6> kubectl version --client
Client Version: v1.34.1
Kustomize Version: v5.7.1
```

Minikube cluster status



```
PS D:\Downloads\Karunya\devops\devops6> minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

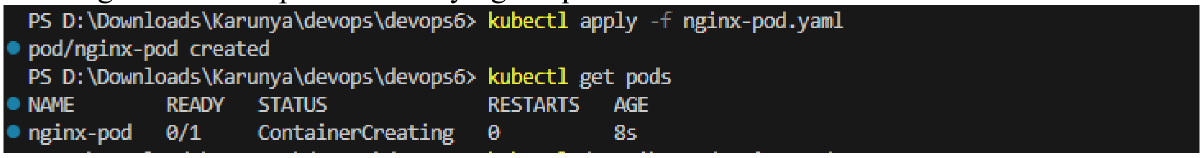
Cluster and node verification



```
PS D:\Downloads\Karunya\devops\devops6> kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
minikube   Ready    control-plane   31s   v1.34.0
```

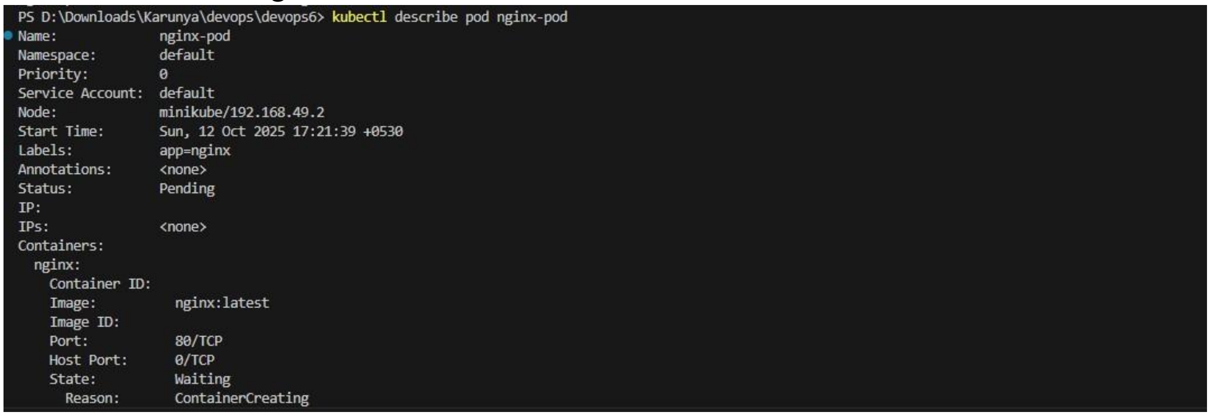
Pod creation and verification

Creating the NGINX pod and verifying the pod status



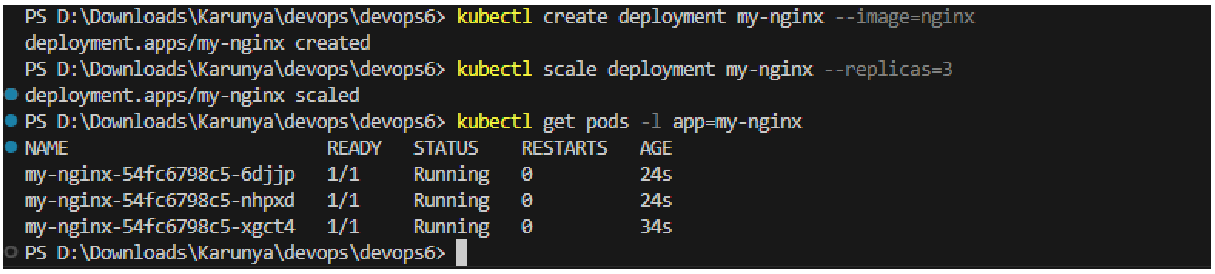
```
PS D:\Downloads\Karunya\devops\devops6> kubectl apply -f nginx-pod.yaml
pod/nginx-pod created
PS D:\Downloads\Karunya\devops\devops6> kubectl get pods
NAME        READY   STATUS              RESTARTS   AGE
nginx-pod   0/1     ContainerCreating   0          8s
```

Pod details and configuration



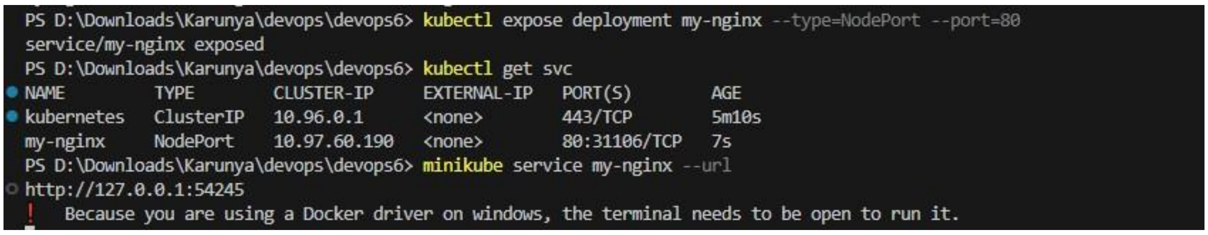
```
PS D:\Downloads\Karunya\devops\devops6> kubectl describe pod nginx-pod
Name:             nginx-pod
Namespace:        default
Priority:         0
Service Account:  default
Node:             minikube/192.168.49.2
Start Time:       Sun, 12 Oct 2025 17:21:39 +0530
Labels:           app=nginx
Annotations:      <none>
Status:           Pending
IP:
IPs:              <none>
Containers:
  nginx:
    Container ID:
    Image:          nginx:latest
    Image ID:
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Waiting
      Reason:       ContainerCreating
```

Deployment and Scaling



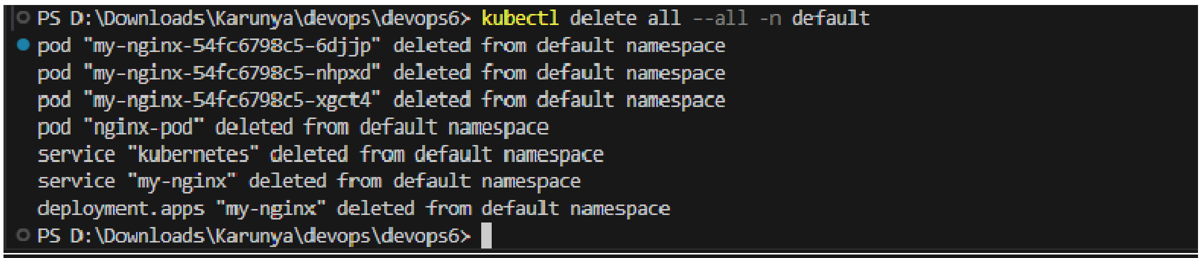
```
PS D:\Downloads\Karunya\devops\devops6> kubectl create deployment my-nginx --image=nginx
deployment.apps/my-nginx created
PS D:\Downloads\Karunya\devops\devops6> kubectl scale deployment my-nginx --replicas=3
deployment.apps/my-nginx scaled
PS D:\Downloads\Karunya\devops\devops6> kubectl get pods -l app=my-nginx
NAME                       READY   STATUS    RESTARTS   AGE
my-nginx-54fc6798c5-6djjp  1/1     Running   0          24s
my-nginx-54fc6798c5-nhpxd  1/1     Running   0          24s
my-nginx-54fc6798c5-xgct4  1/1     Running   0          34s
PS D:\Downloads\Karunya\devops\devops6>
```

Service Exposure

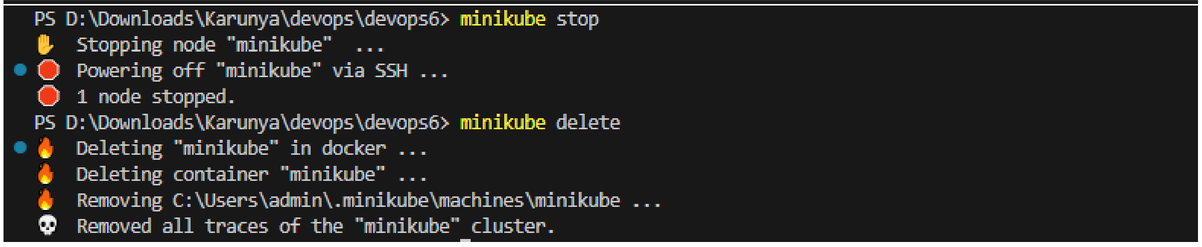


```
PS D:\Downloads\Karunya\devops\devops6> kubectl expose deployment my-nginx --type=NodePort --port=80
service/my-nginx exposed
PS D:\Downloads\Karunya\devops\devops6> kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP        5m10s
my-nginx     NodePort    10.97.60.190   <none>        80:31106/TCP   7s
PS D:\Downloads\Karunya\devops\devops6> minikube service my-nginx --url
http://127.0.0.1:54245
❗  Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

Cleanup



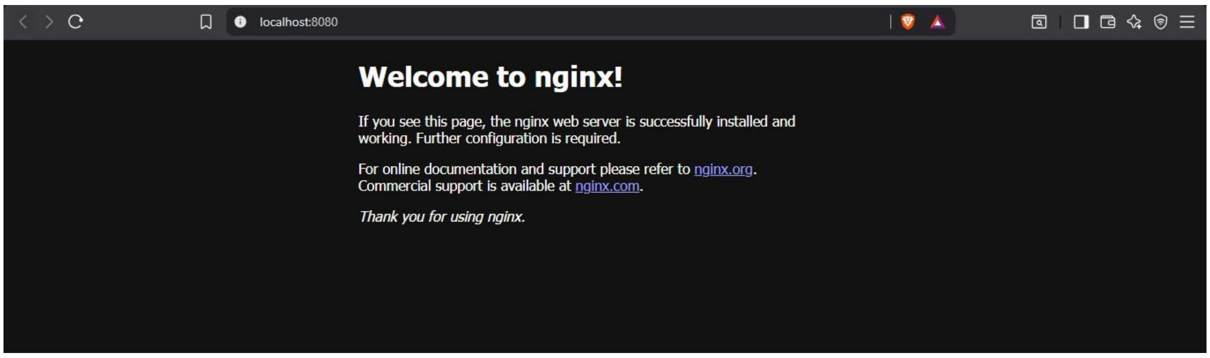
```
PS D:\Downloads\Karunya\devops\devops6> kubectl delete all --all -n default
pod "my-nginx-54fc6798c5-6djjp" deleted from default namespace
pod "my-nginx-54fc6798c5-nhpxd" deleted from default namespace
pod "my-nginx-54fc6798c5-xgct4" deleted from default namespace
pod "nginx-pod" deleted from default namespace
service "kubernetes" deleted from default namespace
service "my-nginx" deleted from default namespace
deployment.apps "my-nginx" deleted from default namespace
PS D:\Downloads\Karunya\devops\devops6>
```



```
PS D:\Downloads\Karunya\devops\devops6> minikube stop
✋  Stopping node "minikube"  ...
🛑  Powering off "minikube" via SSH ...
🛑  1 node stopped.
PS D:\Downloads\Karunya\devops\devops6> minikube delete
🔥  Deleting "minikube" in docker ...
🔥  Deleting container "minikube" ...
🔥  Removing C:\Users\admin\.minikube\machines\minikube ...
💀  Removed all traces of the "minikube" cluster.
```

Output:

Nginx Welcome Page accessed via Kubernetes Pod Port Forwarding


localhost:8080

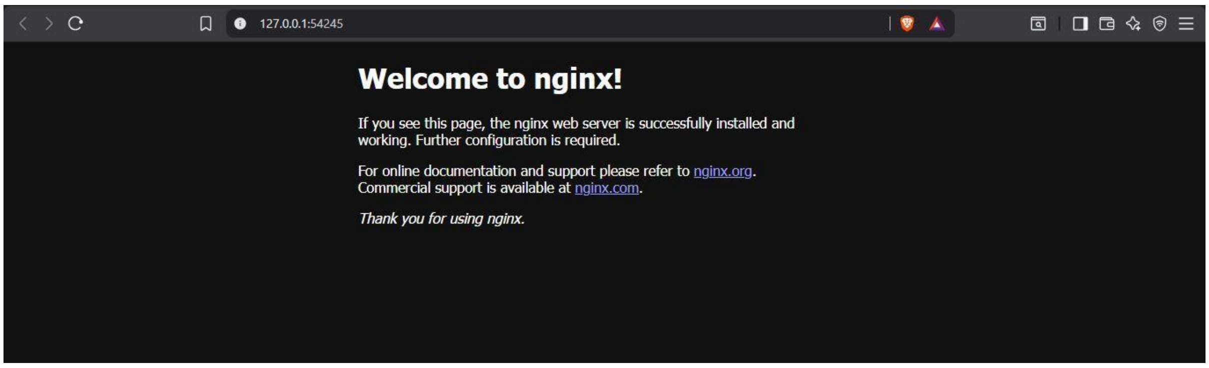
Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Nginx Welcome Page accessed via Kubernetes NodePort Service


127.0.0.1:54245

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Result:

Thus successfully installed Docker, kubectl, and Minikube; created and managed Pods; deployed and scaled applications; exposed Deployments as services; and cleaned up resources safely.

Ex. No.:7	Kubernetes Deployment and Service for Python App from Docker Hub
Date:17-09-25	

Objective

To deploy a simple Python web application stored in Docker Hub to a Kubernetes cluster using a Deployment and Service, and access it through Minikube.

Tools/ Software Required

- Minikube and kubectl installed
- Docker Desktop running
- Internet connectivity to pull the Docker image from Docker Hub.

Industry Relevance and Applications

- **Cloud-Native Deployment & Microservices:** Kubernetes enables scalable, resilient, and automated deployment of containerized applications, illustrating modern microservices workflows.
- **Automation & Scalability:** Supports automated scaling, rolling updates, and service discovery, ensuring applications remain available and efficient.
- **Local Testing Environment:** Minikube allows safe local validation of Kubernetes deployments before moving to production.

Description

- **Docker:** Builds and runs containerized Python web app images.
- **Python Web App:** Flask app packaged inside a Docker container.
- **Minikube:** Local single-node Kubernetes cluster for testing deployments.
- **kubectl:** CLI to create and manage Kubernetes resources (Pods, Deployments, Services).
- **Deployment:** Manages multiple replicas of the app and handles updates.
- **Pod:** Smallest deployable unit in Kubernetes running a container instance.
- **Service (NodePort):** Exposes the app externally and provides stable access.

- **Scaling & Updates:** Demonstrates replicating Pods and rolling updates for zero-downtime deployments.
- **Local Testing:** Access the running app via Minikube service URL in the browser.

Command list with explanation:

Command	Explanation
minikube start --driver=docker	Starts a local Kubernetes cluster using Docker.
kubectl get nodes	Lists cluster nodes and checks if they are Ready.
docker pull python:3.9-slim	Pulls the official Python 3.9 slim image from Docker Hub.
docker pull dockersamples/flaskapp	Pulls a prebuilt Flask app image (replace if unavailable).
kubectl apply -f python-deployment.yaml	Creates a Deployment in Kubernetes as defined in the YAML file.
kubectl get deployments	Lists Deployments and shows their status.
kubectl get pods	Lists all Pods; shows how many are running for the Deployment.
kubectl describe deployment python-web-deployment	Shows detailed information about the Deployment, including replica status, events, and container details.
kubectl apply -f python-service.yaml	Creates a Service (NodePort) to expose the Deployment externally.
kubectl get svc	Lists all Services and their assigned NodePorts.
minikube service python-web-service --url	Retrieves the URL to access the running Python app in a browser.
kubectl delete deployment python-web-deployment	Deletes the Deployment and all associated Pods.
kubectl delete svc python-web-service	Deletes the Service exposing the app.
minikube stop	Stops the local Minikube cluster.
minikube delete	Deletes the Minikube cluster completely.

Procedure:

Setup and Verification:

1. Ensure Minikube, kubectl, and Docker Desktop are installed and running.

2. Start Minikube cluster:

```
minikube start --driver=docker
```

3. Verify node status:

```
kubectl get nodes
```

Create your flask docker image:

App.py

```
from flask import Flask, render_template
app = Flask(_name_)
@app.route('/')
def home():
   return render_template('index.html')
if _name___== '_main_':
   app.run(host='0.0.0.0', port=5000)
```

Templates/index.html

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>My Kubernetes Flask App</title>
   <style>
      body { font-family: Arial, sans-serif; text-align: center; background-color: #f0f8ff; }
      h1 { color: #0267ff; }
      p { font-size: 18px; color: #555; }
      .box { margin: 50px auto; padding: 20px; border: 2px solid #6ba3f7; width: 50%; border-
radius: 10px; background-color: #e0f7fa; }
   </style>
</head>
<body>
   <div class="box">
      <h1>Welcome to My Flask App!</h1>
      <p>This is running in Kubernetes via Minikube.</p>
   </div>
```

```
</body>
</html>
```

requirements.txt

```
flask
```

Dockerfile

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py .
COPY templates/ ./templates/
EXPOSE 5000
CMD ["python", "app.py"]
```

Build Docker image

```
docker build -t mohankumar454522/flaskimage:1.0 .
```

Run locally

```
docker run -p 5000:5000 mohankumar454522/flaskimage:1.0
```

Push image to Dockerhub

```
docker login
```

```
docker push mohankumar454522/flaskimage:1.0
```

Create a Kubernetes Deployment:

1. Create python-deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: python-web-deployment
  labels:
    app: python-web
spec:
  replicas: 2
  selector:
  matchLabels:
      app: python-web
  template:
    metadata:
      labels:
        app: python-web
    spec:
      containers:
      - name: python-web
        image: flaskimage:1.0
        ports:
        - containerPort: 5000
```

2. Apply the deployment

```
kubectl apply -f python-deployment.yaml
```

3. Check status

```
kubectl get deployments

kubectl get pods
```

4. View details

```
kubectl describe deployment python-web-deployment
```

Expose the deployment as a service:

1. Now expose your Python app using a NodePort Service.
   Create a file python-service.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: python-web-service
spec:
  type: NodePort
  selector:
    app: python-web
  ports:
  - protocol: TCP
    port: 5000
    targetPort: 5000
```

2. Apply it:

```
kubectl apply -f python-service.yaml
```

3. Check if the service is created:

```
kubectl get svc
```

4. Get Minikube service URL:

   minikube service python-web-service –url

5. Open the displayed URL in your browser. You should see the Python web app page.

Clean up:

1. clean up all created resources:

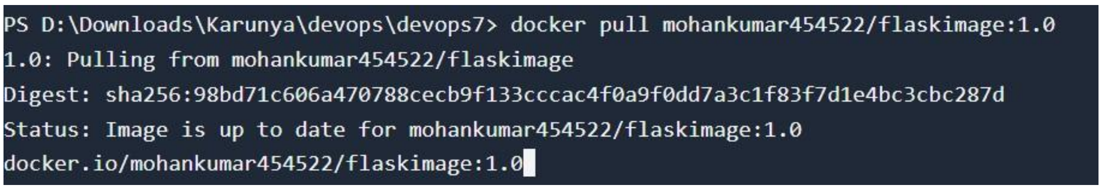
   minikube stop

   minikube delete

Sample Inputs:

Choose a Python Docker Image:

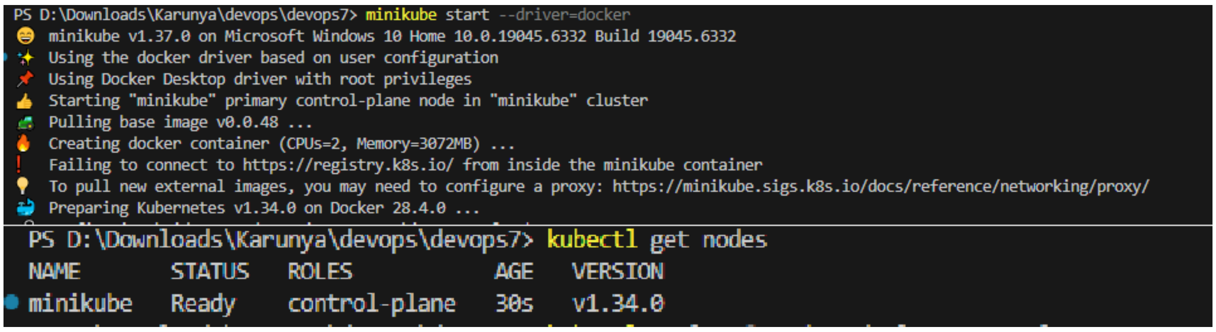
Pull Docker image



```
PS D:\Downloads\Karunya\devops\devops7> docker pull mohankumar454522/flaskimage:1.0
1.0: Pulling from mohankumar454522/flaskimage
Digest: sha256:98bd71c606a470788cecb9f133cccac4f0a9f0dd7a3c1f83f7d1e4bc3cbc287d
Status: Image is up to date for mohankumar454522/flaskimage:1.0
docker.io/mohankumar454522/flaskimage:1.0
```

Create a Kubernetes Deployment

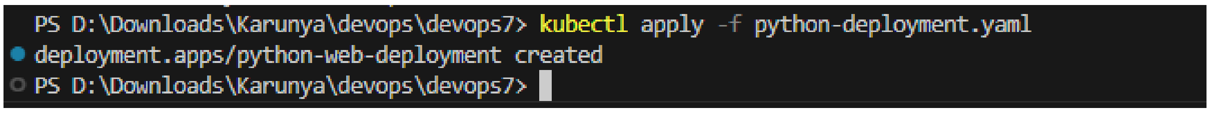
Minikube cluster setup



```
PS D:\Downloads\Karunya\devops\devops7> minikube start --driver=docker
😄  minikube v1.37.0 on Microsoft Windows 10 Home 10.0.19045.6332 Build 19045.6332
✨  Using the docker driver based on user configuration
📌  Using Docker Desktop driver with root privileges
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.48 ...
🔥  Creating docker container (CPUs=2, Memory=3072MB) ...
❗  Failing to connect to https://registry.k8s.io/ from inside the minikube container
💡  To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
🐳  Preparing Kubernetes v1.34.0 on Docker 28.4.0 ...
```

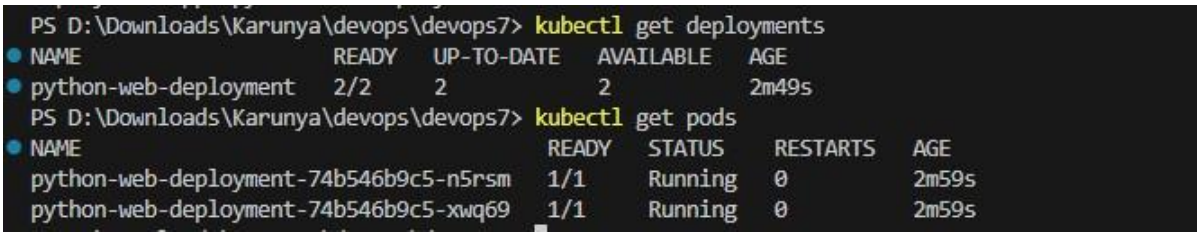
```
PS D:\Downloads\Karunya\devops\devops7> kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
minikube   Ready    control-plane   30s   v1.34.0
```

Apply the deployment:



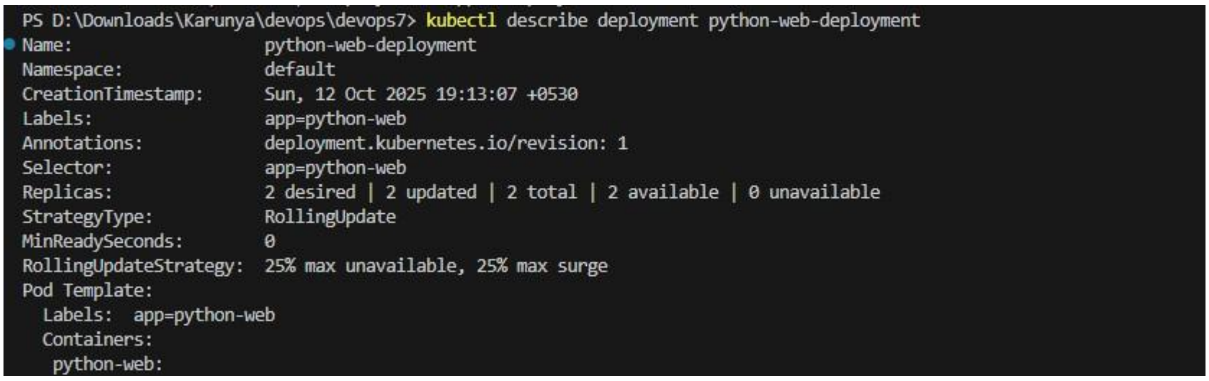
```
PS D:\Downloads\Karunya\devops\devops7> kubectl apply -f python-deployment.yaml
deployment.apps/python-web-deployment created
PS D:\Downloads\Karunya\devops\devops7>
```

Check status



```
PS D:\Downloads\Karunya\devops\devops7> kubectl get deployments
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
python-web-deployment   2/2     2            2           2m49s
PS D:\Downloads\Karunya\devops\devops7> kubectl get pods
NAME                                     READY   STATUS    RESTARTS   AGE
python-web-deployment-74b546b9c5-n5rsm   1/1     Running   0          2m59s
python-web-deployment-74b546b9c5-xwq69   1/1     Running   0          2m59s
```

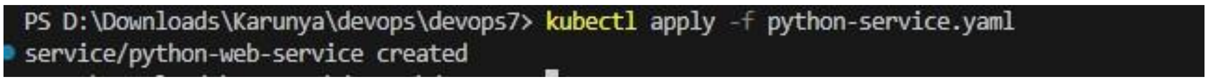
View details:



```
PS D:\Downloads\Karunya\devops\devops7> kubectl describe deployment python-web-deployment
Name:                   python-web-deployment
Namespace:              default
CreationTimestamp:      Sun, 12 Oct 2025 19:13:07 +0530
Labels:                 app=python-web
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=python-web
Replicas:               2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=python-web
  Containers:
   python-web:
```

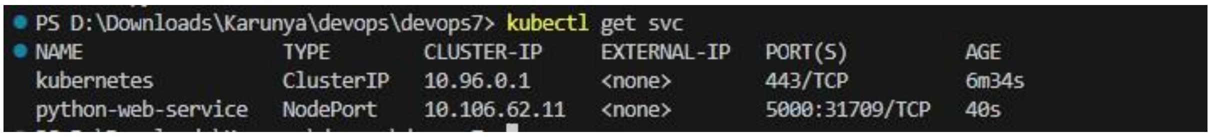
Expose Deployment as a Service

Apply it:



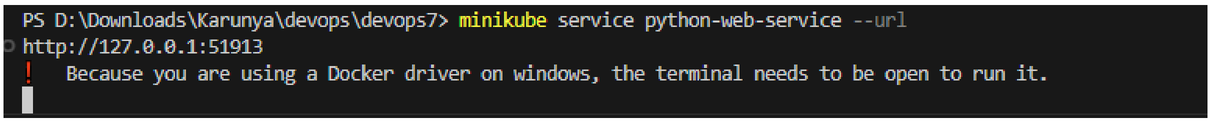
```
PS D:\Downloads\Karunya\devops\devops7> kubectl apply -f python-service.yaml
service/python-web-service created
```

Check if the service is created:



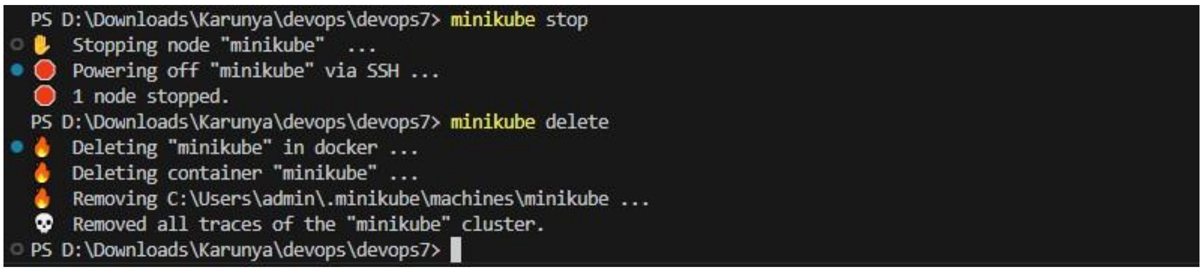
```
PS D:\Downloads\Karunya\devops\devops7> kubectl get svc
NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
kubernetes           ClusterIP   10.96.0.1      <none>        443/TCP          6m34s
python-web-service   NodePort    10.106.62.11   <none>        5000:31709/TCP   40s
```

Get Minikube service URL:



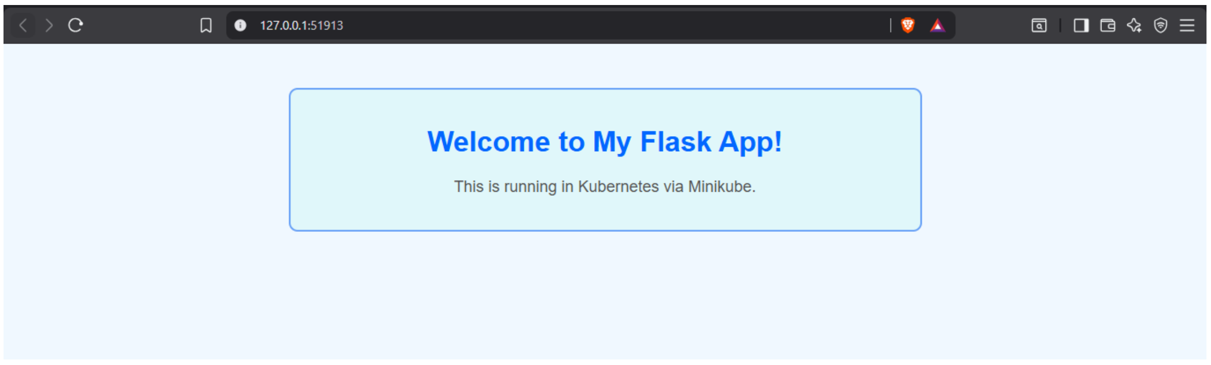
```
PS D:\Downloads\Karunya\devops\devops7> minikube service python-web-service --url
http://127.0.0.1:51913
!   Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

Cleanup



```
PS D:\Downloads\Karunya\devops\devops7> minikube stop
✋  Stopping node "minikube"  ...
🛑  Powering off "minikube" via SSH ...
🛑  1 node stopped.
PS D:\Downloads\Karunya\devops\devops7> minikube delete
🔥  Deleting "minikube" in docker ...
🔥  Deleting container "minikube" ...
🔥  Removing C:\Users\admin\.minikube\machines\minikube ...
💀  Removed all traces of the "minikube" cluster.
PS D:\Downloads\Karunya\devops\devops7>
```

Output:


127.0.0.1:51913

Welcome to My Flask App!

This is running in Kubernetes via Minikube.

Result:

Thus, a containerized Python web application was successfully pulled from Docker Hub, deployed on a Kubernetes cluster using a Deployment, exposed via a NodePort Service, and accessed through Minikube.

Ex. No.:8	Ansible Integration with Kubernetes Cluster
Date:08-10-25	

Objective

To integrate and configure Ansible with a Kubernetes cluster to automate Kubernetes administration tasks using Ansible modules and collections.

Tools/ Software Required

- OS: Windows 10/11 with WSL Ubuntu or any Linux VM
- Ansible: Automation tool (sudo apt install ansible)
- Kubernetes: Minikube cluster for local setup
- kubectl: CLI to manage Kubernetes
- Docker: Minikube driver and container runtime
- Ansible Collection: community.kubernetes for Kubernetes modules

Industry Relevance and Applications

- **Automation of Kubernetes tasks:** Reduces manual cluster management and operational errors.
- **DevOps and CI/CD integration:** Enables seamless deployment, scaling, and monitoring of applications.
- **Cloud-native and enterprise adoption:** Widely used in organizations for managing containerized workloads efficiently.

Description

This experiment demonstrates using Ansible with a Kubernetes cluster to automate administrative tasks, managing resources via playbooks and modules instead of manual kubectl commands.

1. **Ansible:**
    - An open-source automation and configuration management tool.
    - Used to write **playbooks** that define tasks like creating, listing, or scaling Kubernetes resources.

2. **Kubernetes (Minikube):**
   - A container orchestration platform that manages containerized applications.
   - Minikube provides a lightweight, local Kubernetes cluster for experimentation and testing.
3. **kubectl:**
   - Command-line interface for interacting with Kubernetes clusters.
   - Provides configuration and connectivity that Ansible uses under the hood.
4. **Docker:**
   - Container runtime used by Minikube to run pods and services.
   - Enables a lightweight and reproducible environment for applications.
5. **Ansible Kubernetes Collection (community.kubernetes):**
   - Adds Kubernetes-specific modules (k8s, k8s_info, k8s_scale) to Ansible.
   - Allows automation of tasks like gathering cluster info, managing namespaces, and scaling deployments.

Command list with explanation:

Command	**Explanation**
minikube start --driver=docker	Starts a local Kubernetes cluster using Docker.
kubectl get nodes	Verifies that the cluster nodes are up and in Ready state.
sudo apt update	Updates the package list in WSL Ubuntu.
sudo apt install -y ansible	Installs Ansible automation tool.
ansible --version	Checks the installed Ansible version.
ansible-galaxy collection install community.kubernetes	Installs Kubernetes collection with modules (k8s, k8s_info, k8s_scale).
kubectl config view	Displays the kubeconfig file; ensures Ansible can connect to the cluster.
echo "…playbook content…" > test-k8s.yaml	Creates a test Ansible playbook to list pods in the default namespace.
ansible-playbook test-k8s.yaml	Runs the test playbook to verify Ansible-Kubernetes connection.

Command	Explanation
echo "…playbook content…" > get-k8s-info.yaml	Creates a playbook to gather cluster information (namespaces and nodes).
ansible-playbook get-k8s-info.yaml	Runs the playbook to display namespaces and nodes in the cluster.
kubectl delete namespace ansible-lab	Deletes any test namespace created during the experiment.
minikube stop	Stops the Minikube cluster.
minikube delete	Deletes the Minikube cluster and frees resources.

Procedure:

1. Verify Minikube Setup

- Start the Minikube cluster using Docker:

```
minikube start --driver=docker
```

- Check the status of cluster nodes:

```
kubectl get nodes
```

- Ensure all nodes show STATUS = Ready

2. Install Ansible

- Update the package index in WSL Ubuntu:

```
sudo apt update
```

- Install Ansible:

```
sudo apt install -y ansible
```

- Verify the installation:

```
ansible --version
```

3. Install Kubernetes Collection for Ansible

- Install the official collection:

```
ansible-galaxy collection install community.kubernetes
```

- This adds modules such as:

  k8s → manage Kubernetes resources

  k8s_info → gather resource information

  k8s_scale → scale deployments

4. **Verify Kubernetes Connection via Ansible**

- Check your kubeconfig file:

```
kubectl config view
```

- If kubectl get pods works correctly, then Ansible can connect to the same cluster automatically. Test with a simple Ansible playbook:
- test-k8s.yaml:

```
---
- name: Test Kubernetes Connection
  hosts: localhost
  connection: local
  gather_facts: no
  vars:
    ansible_python_interpreter: /home/user1/.ansible-env/bin/python
  tasks:
    - name: List all pods in default namespace
      community.kubernetes.k8s_info:
        kind: Pod
      register: pod_list
    - name: Show pod names only
      debug:
        msg: "{{ pod_list.resources | map(attribute='metadata.name') | list }}"
```

- Run the playbook:

```
ansible-playbook test-k8s.yaml
```

- Confirm that Ansible successfully retrieves pod information from the cluster.

5. **Create Kubernetes Info Playbook**

- Create get-k8s-info.yaml to gather cluster details:
  - List all namespaces
  - List all nodes

- Create the playbook file get-k8s-info.yaml with the following content:

```
---
- name: Gather Kubernetes Information
  hosts: localhost
  connection: local
  gather_facts: no
  tasks:
    - name: Get all namespaces
      community.kubernetes.k8s_info:
        kind: Namespace
      register: ns_info
    - debug:
        var: ns_info.resources
    - name: Get all nodes
      community.kubernetes.k8s_info:
        kind: Node
      register: node_info
    - debug:
        var: node_info.resources
```

- Run the playbook:

```
ansible-playbook get-k8s-info.yaml
```

- Verify the output shows namespaces and nodes in the Minikube cluster

6. Cleanup

- Delete any test namespace created:

```
kubectl delete namespace ansible-lab
```

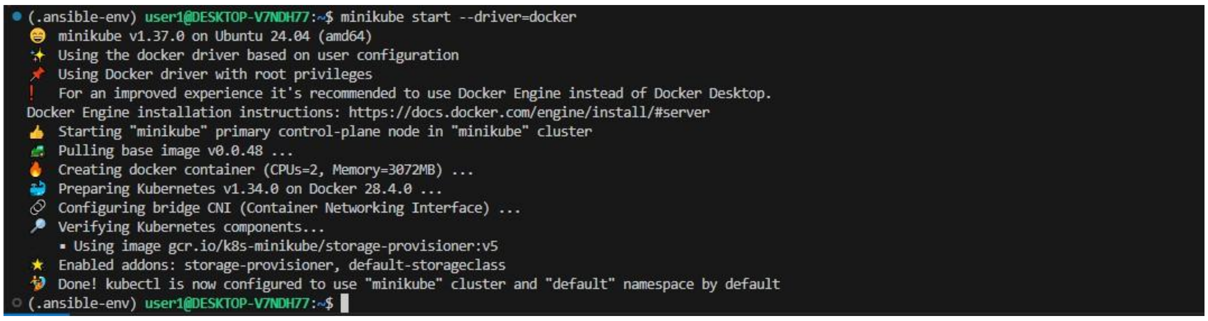
- Stop and delete Minikube cluster:

```
minikube stop

minikube delete
```

Sample Inputs:

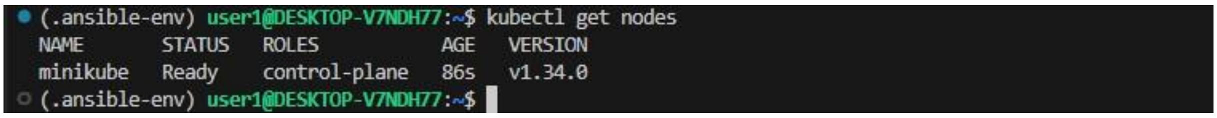
Verify Minikube cluster:

Start the local Kubernetes cluster



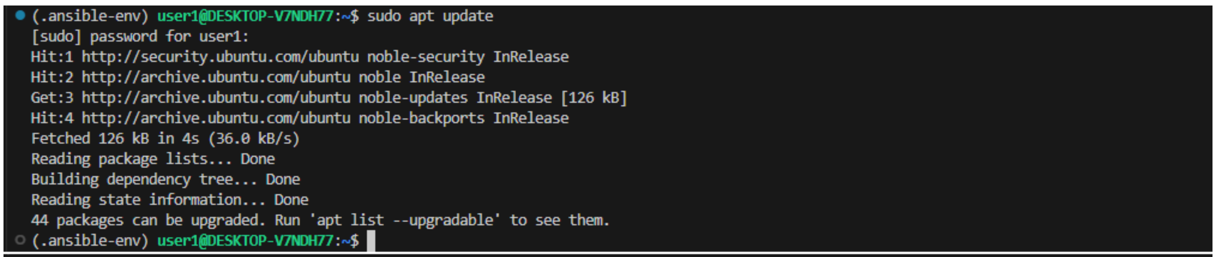
```
(.ansible-env) user1@DESKTOP-V7NDH77:~$ minikube start --driver=docker
  minikube v1.37.0 on Ubuntu 24.04 (amd64)
  Using the docker driver based on user configuration
  Using Docker driver with root privileges
!  For an improved experience it's recommended to use Docker Engine instead of Docker Desktop.
Docker Engine installation instructions: https://docs.docker.com/engine/install/#server
  Starting "minikube" primary control-plane node in "minikube" cluster
  Pulling base image v0.0.48 ...
  Creating docker container (CPUs=2, Memory=3072MB) ...
  Preparing Kubernetes v1.34.0 on Docker 28.4.0 ...
  Configuring bridge CNI (Container Networking Interface) ...
  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
  Enabled addons: storage-provisioner, default-storageclass
  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
(.ansible-env) user1@DESKTOP-V7NDH77:~$
```

Get Minikube node and their status

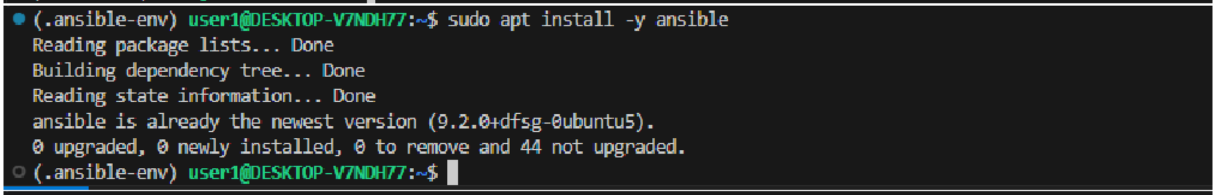


```
(.ansible-env) user1@DESKTOP-V7NDH77:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
minikube   Ready    control-plane   86s   v1.34.0
(.ansible-env) user1@DESKTOP-V7NDH77:~$
```

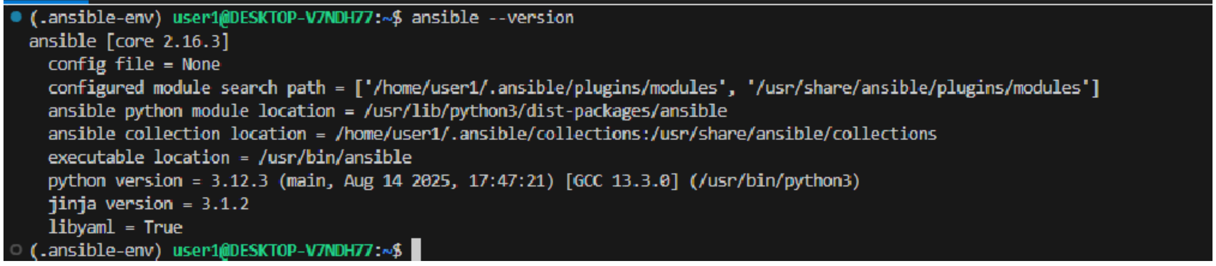
Install Ansible and verify Installation



```
(.ansible-env) user1@DESKTOP-V7NDH77:~$ sudo apt update
[sudo] password for user1:
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]
Hit:4 http://archive.ubuntu.com/ubuntu noble-backports InRelease
Fetched 126 kB in 4s (36.0 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
44 packages can be upgraded. Run 'apt list --upgradable' to see them.
(.ansible-env) user1@DESKTOP-V7NDH77:~$
```

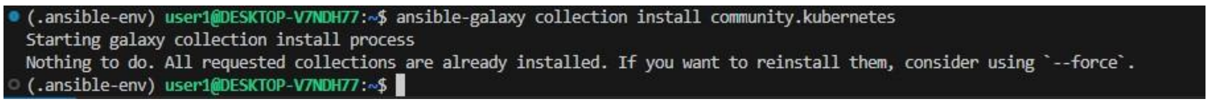


```
(.ansible-env) user1@DESKTOP-V7NDH77:~$ sudo apt install -y ansible
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ansible is already the newest version (9.2.0+dfsg-0ubuntu5).
0 upgraded, 0 newly installed, 0 to remove and 44 not upgraded.
(.ansible-env) user1@DESKTOP-V7NDH77:~$
```



```
(.ansible-env) user1@DESKTOP-V7NDH77:~$ ansible --version
ansible [core 2.16.3]
  config file = None
  configured module search path = ['/home/user1/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/user1/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.12.3 (main, Aug 14 2025, 17:47:21) [GCC 13.3.0] (/usr/bin/python3)
  jinja version = 3.1.2
  libyaml = True
(.ansible-env) user1@DESKTOP-V7NDH77:~$
```

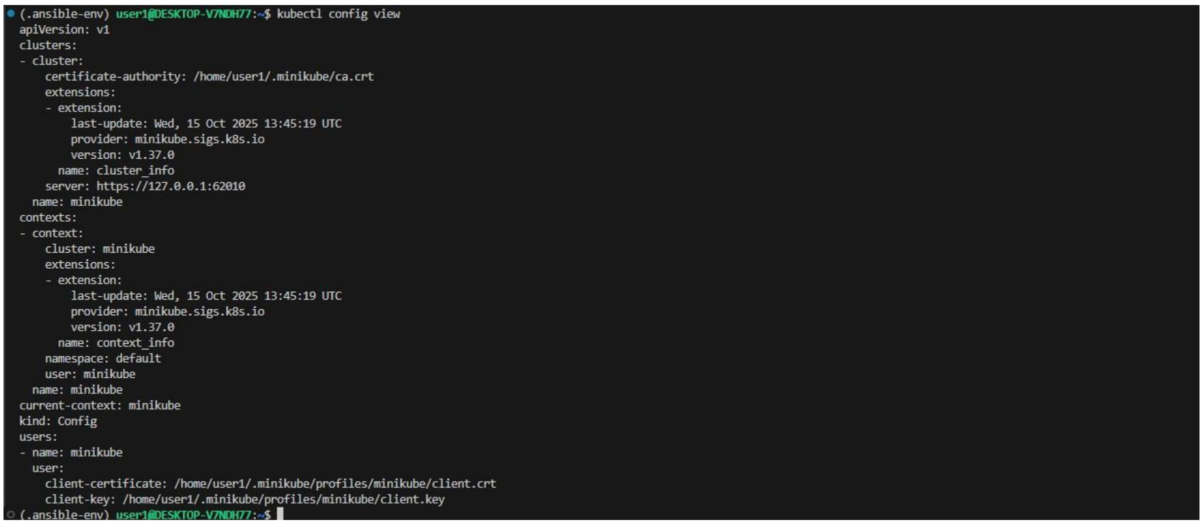
Install Kubernetes Collection for Ansible



```
(.ansible-env) user1@DESKTOP-V7NDH77:~$ ansible-galaxy collection install community.kubernetes
Starting galaxy collection install process
Nothing to do. All requested collections are already installed. If you want to reinstall them, consider using `--force`.
(.ansible-env) user1@DESKTOP-V7NDH77:~$
```

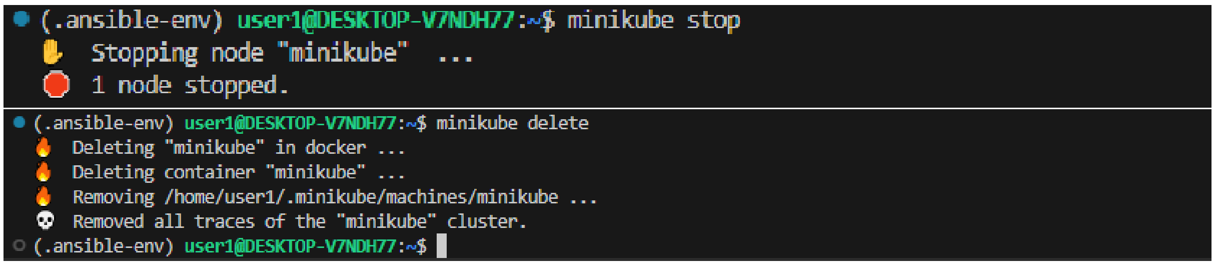
Verify Kubernetes Connection for Ansible

Check your configuration:



```
● (.ansible-env) user1@DESKTOP-V7NDH77:~$ kubectl config view
  apiVersion: v1
  clusters:
  - cluster:
      certificate-authority: /home/user1/.minikube/ca.crt
      extensions:
      - extension:
          last-update: Wed, 15 Oct 2025 13:45:19 UTC
          provider: minikube.sigs.k8s.io
          version: v1.37.0
        name: cluster_info
      server: https://127.0.0.1:62010
    name: minikube
  contexts:
  - context:
      cluster: minikube
      extensions:
      - extension:
          last-update: Wed, 15 Oct 2025 13:45:19 UTC
          provider: minikube.sigs.k8s.io
          version: v1.37.0
        name: context_info
      namespace: default
      user: minikube
    name: minikube
  current-context: minikube
  kind: Config
  users:
  - name: minikube
    user:
      client-certificate: /home/user1/.minikube/profiles/minikube/client.crt
      client-key: /home/user1/.minikube/profiles/minikube/client.key
○ (.ansible-env) user1@DESKTOP-V7NDH77:~$
```

Cleanup

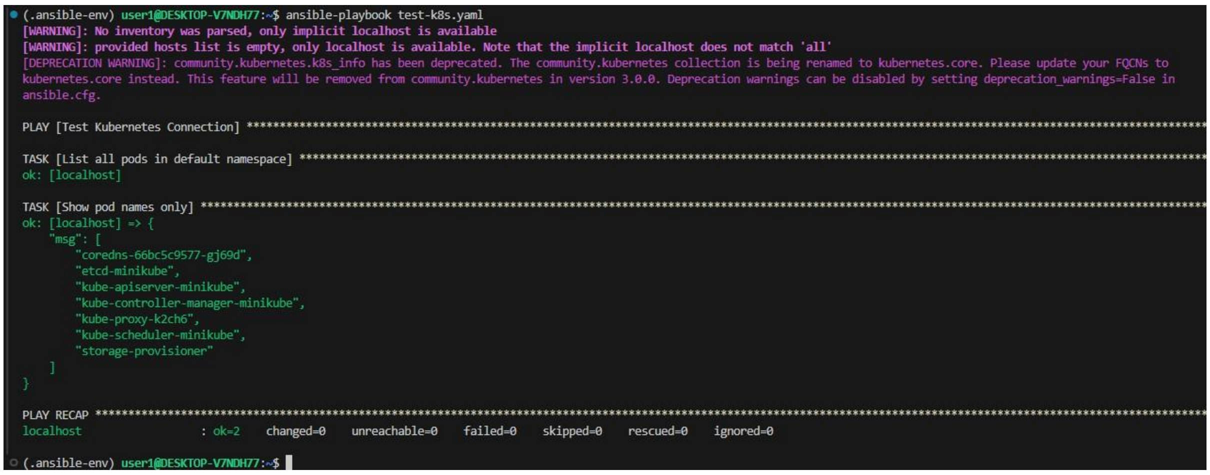


```
● (.ansible-env) user1@DESKTOP-V7NDH77:~$ minikube stop
  ✋  Stopping node "minikube"  ...
  🛑  1 node stopped.
● (.ansible-env) user1@DESKTOP-V7NDH77:~$ minikube delete
  🔥  Deleting "minikube" in docker ...
  🔥  Deleting container "minikube" ...
  🔥  Removing /home/user1/.minikube/machines/minikube ...
  💀  Removed all traces of the "minikube" cluster.
○ (.ansible-env) user1@DESKTOP-V7NDH77:~$
```

Output:

Run the playbook: test-k8s.yaml

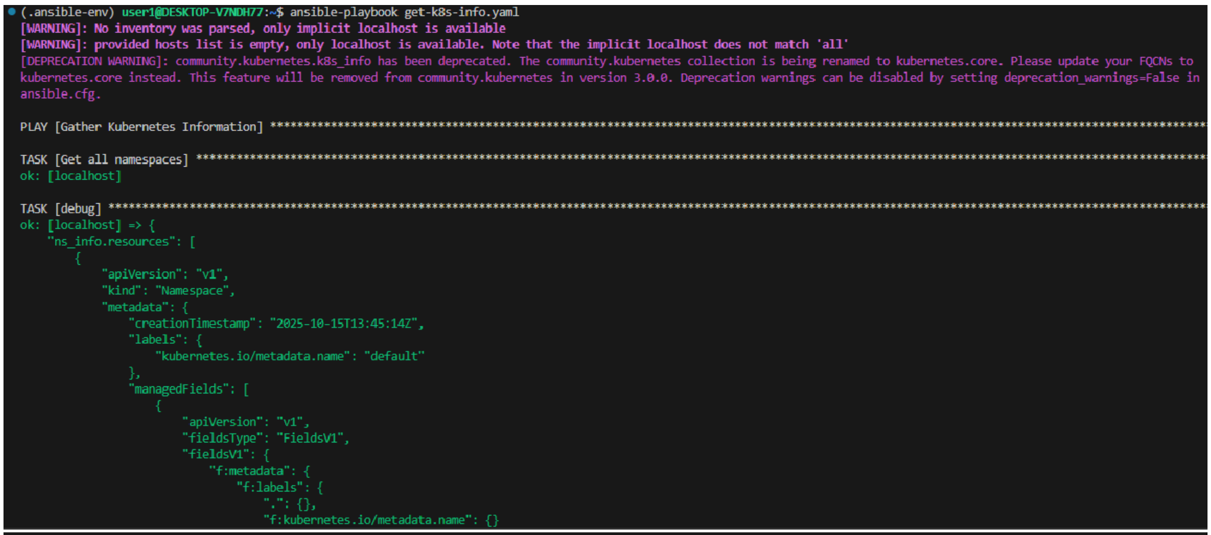
List of all pods in the default Kubernetes namespace.



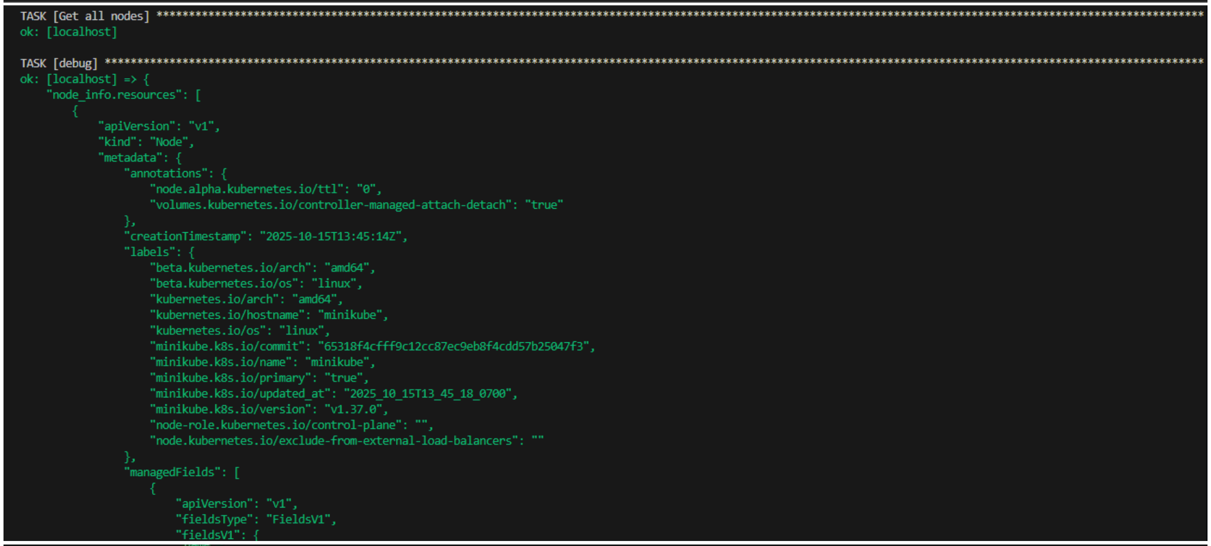
```
● (.ansible-env) user1@DESKTOP-V7NDH77:~$ ansible-playbook test-k8s.yaml
  [WARNING]: No inventory was parsed, only implicit localhost is available
  [WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'
  [DEPRECATION WARNING]: community.kubernetes.k8s_info has been deprecated. The community.kubernetes collection is being renamed to kubernetes.core. Please update your FQCNs to
  kubernetes.core instead. This feature will be removed from community.kubernetes in version 3.0.0. Deprecation warnings can be disabled by setting deprecation_warnings=False in
  ansible.cfg.

  PLAY [Test Kubernetes Connection] *******************************************************************************************************************************************

  TASK [List all pods in default namespace] ***********************************************************************************************************************************
  ok: [localhost]

  TASK [Show pod names only] **************************************************************************************************************************************************
  ok: [localhost] => {
      "msg": [
          "coredns-66bc5c9577-gj69d",
          "etcd-minikube",
          "kube-apiserver-minikube",
          "kube-controller-manager-minikube",
          "kube-proxy-k2ch6",
          "kube-scheduler-minikube",
          "storage-provisioner"
      ]
  }

  PLAY RECAP ******************************************************************************************************************************************************************
  localhost                  : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

○ (.ansible-env) user1@DESKTOP-V7NDH77:~$
```

Run the get-k8s-info.yaml file

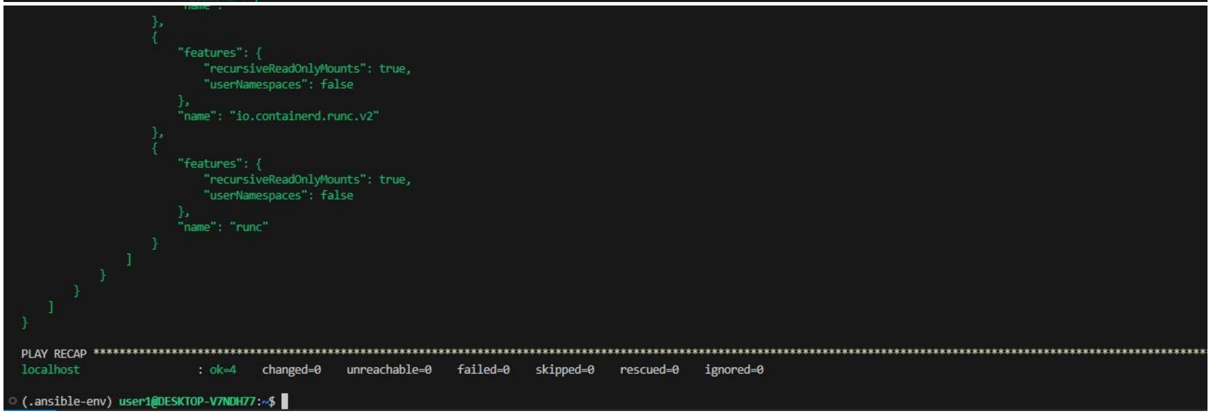
All namespaces and nodes in the Kubernetes cluster.



```
● (.ansible-env) user1@DESKTOP-V7NDH77:~$ ansible-playbook get-k8s-info.yaml
  [WARNING]: No inventory was parsed, only implicit localhost is available
  [WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'
  [DEPRECATION WARNING]: community.kubernetes.k8s_info has been deprecated. The community.kubernetes collection is being renamed to kubernetes.core. Please update your FQCNs to
  kubernetes.core instead. This feature will be removed from community.kubernetes in version 3.0.0. Deprecation warnings can be disabled by setting deprecation_warnings=False in
  ansible.cfg.

  PLAY [Gather Kubernetes Information] ***********************************************************************************************************************************

  TASK [Get all namespaces] **********************************************************************************************************************************************
  ok: [localhost]

  TASK [debug] ***********************************************************************************************************************************************************
  ok: [localhost] => {
      "ns_info.resources": [
          {
              "apiVersion": "v1",
              "kind": "Namespace",
              "metadata": {
                  "creationTimestamp": "2025-10-15T13:45:14Z",
                  "labels": {
                      "kubernetes.io/metadata.name": "default"
                  },
                  "managedFields": [
                      {
                          "apiVersion": "v1",
                          "fieldsType": "FieldsV1",
                          "fieldsV1": {
                              "f:metadata": {
                                  "f:labels": {
                                      ".": {},
                                      "f:kubernetes.io/metadata.name": {}
```



```
  TASK [Get all nodes] ***************************************************************************************************************************************************
  ok: [localhost]

  TASK [debug] ***********************************************************************************************************************************************************
  ok: [localhost] => {
      "node_info.resources": [
          {
              "apiVersion": "v1",
              "kind": "Node",
              "metadata": {
                  "annotations": {
                      "node.alpha.kubernetes.io/ttl": "0",
                      "volumes.kubernetes.io/controller-managed-attach-detach": "true"
                  },
                  "creationTimestamp": "2025-10-15T13:45:14Z",
                  "labels": {
                      "beta.kubernetes.io/arch": "amd64",
                      "beta.kubernetes.io/os": "linux",
                      "kubernetes.io/arch": "amd64",
                      "kubernetes.io/hostname": "minikube",
                      "kubernetes.io/os": "linux",
                      "minikube.k8s.io/commit": "65318f4cfff9c12cc87ec9eb8f4cdd57b25047f3",
                      "minikube.k8s.io/name": "minikube",
                      "minikube.k8s.io/primary": "true",
                      "minikube.k8s.io/updated_at": "2025_10_15T13_45_18_0700",
                      "minikube.k8s.io/version": "v1.37.0",
                      "node-role.kubernetes.io/control-plane": "",
                      "node.kubernetes.io/exclude-from-external-load-balancers": ""
                  },
                  "managedFields": [
                      {
                          "apiVersion": "v1",
                          "fieldsType": "FieldsV1",
                          "fieldsV1": {
```



```
                      },
                      {
                          "features": {
                              "recursiveReadOnlyMounts": true,
                              "userNamespaces": false
                          },
                          "name": "io.containerd.runc.v2"
                      },
                      {
                          "features": {
                              "recursiveReadOnlyMounts": true,
                              "userNamespaces": false
                          },
                          "name": "runc"
                      }
                  ]
              }
          }
      ]
  }

  PLAY RECAP *************************************************************************************************************************************************************
  localhost                  : ok=4    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

○ (.ansible-env) user1@DESKTOP-V7NDH77:~$ ▌
```

Result:

Thus, installed and configured Ansible for Kubernetes integration, used Ansible's Kubernetes modules to connect and query clusters, and automated basic Kubernetes administrative tasks.

Ex. No.:9	Deploy a Python (Flask) Web App Locally using Ansible
Date:22-10-25	

Objective

To deploy a Flask web application on the local machine using Ansible with Gunicorn and systemd, ensuring automated, idempotent configuration and service management.

Tools/ Software Required

- **OS:** Linux or macOS with Python 3.9+ (Windows users: run under WSL recommended).
- **Installed:**
    - Python pip & venv (python3 -m venv --help should work)
    - Ansible >=2.15 (ansible --version)
- **Ports:** 8000 available locally.

Industry Relevance and Applications

- **DevOps automation:** Simplifies application deployment and configuration management using Ansible.
- **Infrastructure as Code (IaC):** Ensures consistent, repeatable, and scalable deployments across environments.
- **System service management:** Uses systemd for reliable process supervision, auto-restart, and boot-time service handling.
- **Enterprise application deployment:** Commonly adopted for deploying and maintaining Python-based web services in production.

Description

This experiment demonstrates deploying a Python Flask web application locally using Ansible, automating the setup with Gunicorn and systemd to simulate production-grade deployment and service management.

Ansible:

- An open-source automation tool for configuration management and application deployment.

- Used to define playbooks that install dependencies, configure the environment, and manage the Flask service automatically.

Flask:

- A lightweight Python web framework for building web applications and APIs.
- Serves as the core application deployed and managed through Ansible automation.

Gunicorn:

- A production-grade WSGI HTTP server for running Python web applications.
- Handles multiple worker processes for better performance and reliability.

systemd:

- A Linux service manager that ensures the Flask app runs in the background and restarts automatically on failure or system reboot.
- Used here to create a persistent, managed service for the deployed web application.

Localhost Deployment:

- Simulates real-world deployment by hosting the application on http://127.0.0.1:8000.
- Validates that the Ansible playbook is idempotent and can repeatedly deploy the app without manual intervention.

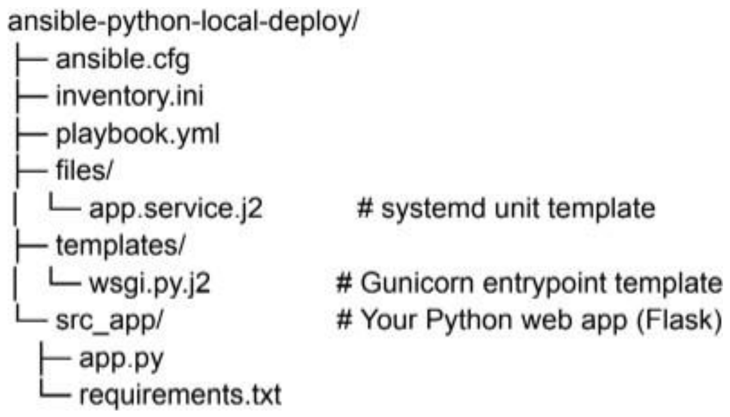
Command list with explanation:

Command	Explanation
python3 -m venv venv	Creates a virtual environment for isolating Python dependencies.
source venv/bin/activate	Activates the virtual environment to install and run project dependencies.
pip install flask gunicorn ansible	Installs Flask for the web app, Gunicorn as the WSGI server, and Ansible for deployment automation.
ansible --version	Verifies that Ansible is correctly installed and checks its version.
ansible -i inventory.ini all -m ping	Tests connectivity and configuration of hosts defined in the inventory file (localhost in this case).

Command	Explanation
ansible-playbook playbook.yml -K	Executes the Ansible playbook to deploy the Flask app using Gunicorn and systemd; -K prompts for sudo password.
sudo systemctl status ansible-flask.service	Checks the running status of the Flask systemd service deployed by Ansible.
curl http://127.0.0.1:8000	Tests if the Flask app is successfully running and serving responses locally.

Procedure:

1. **Set up the Project Directory**
   - Create a folder named ansible-python-local-deploy/.
   - Inside it, create the following structure:



```
ansible-python-local-deploy/
├─ ansible.cfg
├─ inventory.ini
├─ playbook.yml
├─ files/
│  └─ app.service.j2        # systemd unit template
├─ templates/
│  └─ wsgi.py.j2           # Gunicorn entrypoint template
└─ src_app/                # Your Python web app (Flask)
   ├─ app.py
   └─ requirements.txt
```

2. **Create the Flask Application**
   - **Open src_app/app.py and add the following code:**

```
from flask import Flask

app  =  Flask(_name_)

@app.get("/")

def home():

    return {"status": "ok", "msg": "Hello from Ansible-deployed Flask!"}

if __name__== "__main__":

    app.run(host="127.0.0.1", port=8000)
```

- **In src_app/requirements.txt, list the dependencies:**

```
flask==3.0.0
gunicorn==21.2.0
```

3. **Configure Ansible for Local Deployment**

- **In ansible.cfg, add:**

```
[defaults]
inventory = ./inventory.ini
host_key_checking = False
retry_files_enabled = False
stdout_callback = yaml
forks = 10
```

In inventory.ini, specify the local target:

```
[local]
localhost ansible_connection=local
```

4. **Create Template Files**

- **In files/app.service.j2, define the systemd unit:**

```
[Unit]
Description=Gunicorn for Flask app
After=network.target
[Service]
User={{ deploy_user }}
Group={{ deploy_user }}
WorkingDirectory={{ app_root }}
Environment="PATH={{ venv_path }}/bin"
ExecStart={{ venv_path }}/bin/gunicorn --workers {{ gunicorn_workers }} --bind 127.0.0.1:8000 wsgi:app
Restart=always
RestartSec=5
[Install]
```

```
WantedBy=multi-user.target
```

- **In templates/wsgi.py.j2, add:**

```
from app import app
# Expose `app` for Gunicorn
```

5. **Write the Ansible Playbook**

- **In playbook.yml, add the deployment steps:**

```
---
- name: Deploy Flask app to localhost with systemd
  hosts: local
  become: true
  vars:
    deploy_user: "{{ lookup('env','USER') or '$(whoami)' }}"
    app_root: "/opt/ansible_flask_app"
    venv_path: "{{ app_root }}/venv"
    gunicorn_workers: 2
  pre_tasks:
    - name: Ensure Python and system tools present (apt-based; skip if macOS)
      package:
        name: [python3, python3-venv, python3-pip]
        state: present
      when: ansible_os_family == 'Debian'
  tasks:
    - name: Create application directory
      file:
        path: "{{ app_root }}"
        state: directory
        owner: "{{ deploy_user }}"
        group: "{{ deploy_user }}"
        mode: "0755"
```

```
- name: Copy application source
  copy:
    src: "src_app/"
    dest: "{{ app_root }}/"
    owner: "{{ deploy_user }}"
    group: "{{ deploy_user }}"
    mode: "0644"
- name: Create venv
  command: "python3 -m venv {{ venv_path }}"
  args:
    creates: "{{ venv_path }}/bin/activate"
- name: Upgrade pip
  command: "{{ venv_path }}/bin/pip install --upgrade pip"
- name: Install app dependencies
  command: "{{ venv_path }}/bin/pip install -r {{ app_root }}/requirements.txt"
- name: Render wsgi entrypoint
  template:
    src: templates/wsgi.py.j2
    dest: "{{ app_root }}/wsgi.py"
    owner: "{{ deploy_user }}"
    group: "{{ deploy_user }}"
    mode: "0644"
- name: Place systemd unit
  template:
    src: files/app.service.j2
    dest: "/etc/systemd/system/ansible-flask.service"
    mode: "0644"
  notify: ["daemon-reload", "restart app"]
- name: Enable service on boot
```

```
      systemd:
        name: ansible-flask.service
        enabled: true
    - name: Start service now
      systemd:
        name: ansible-flask.service
        state: started
  handlers:
    - name: daemon-reload
      systemd:
        daemon_reload: true
    - name: restart app
      systemd:
        name: ansible-flask.service
        state: restarted
```

6. **Run the Deployment**
   - **From the project root, test Ansible connectivity:**

     ```
     ansible -i inventory.ini all -m ping
     ```
   - **Deploy the Flask app using the playbook:**

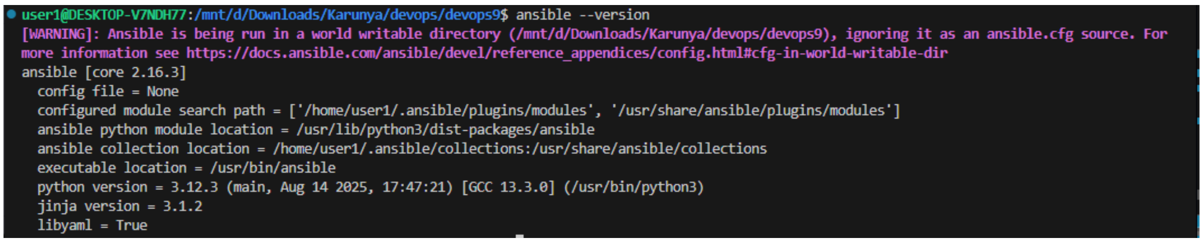
     ```
     ansible-playbook playbook.yml -K
     ```
7. **Verify the Deployment**
   - **Check the service status:**

     ```
     sudo systemctl status ansible-flask.service
     ```
   - **Test the web app in the browser or terminal:**

     ```
     curl http://127.0.0.1:8000
     ```

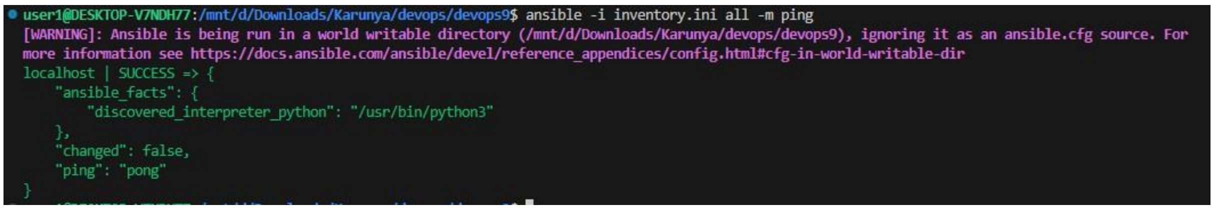
Sample Inputs:

Verifying Ansible Installation



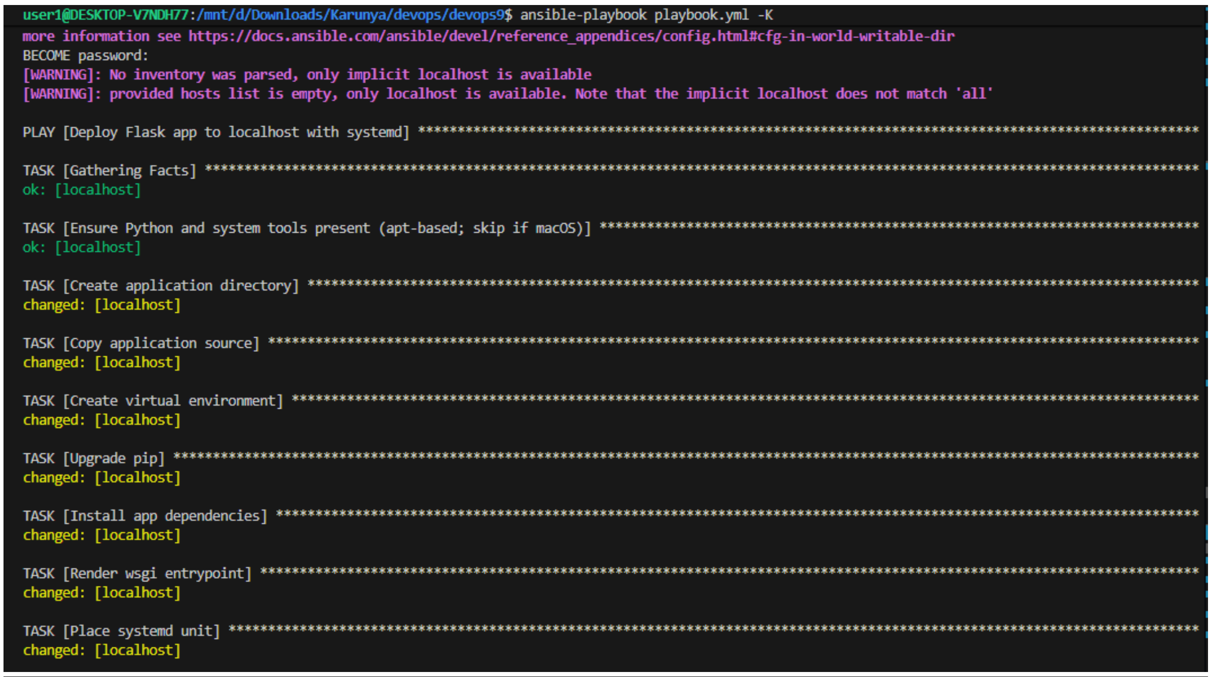
```
user1@DESKTOP-V7NDH77:/mnt/d/Downloads/Karunya/devops/devops9$ ansible --version
[WARNING]: Ansible is being run in a world writable directory (/mnt/d/Downloads/Karunya/devops/devops9), ignoring it as an ansible.cfg source. For
more information see https://docs.ansible.com/ansible/devel/reference_appendices/config.html#cfg-in-world-writable-dir
ansible [core 2.16.3]
  config file = None
  configured module search path = ['/home/user1/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/user1/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.12.3 (main, Aug 14 2025, 17:47:21) [GCC 13.3.0] (/usr/bin/python3)
  jinja version = 3.1.2
  libyaml = True
```

Testing Localhost Connectivity

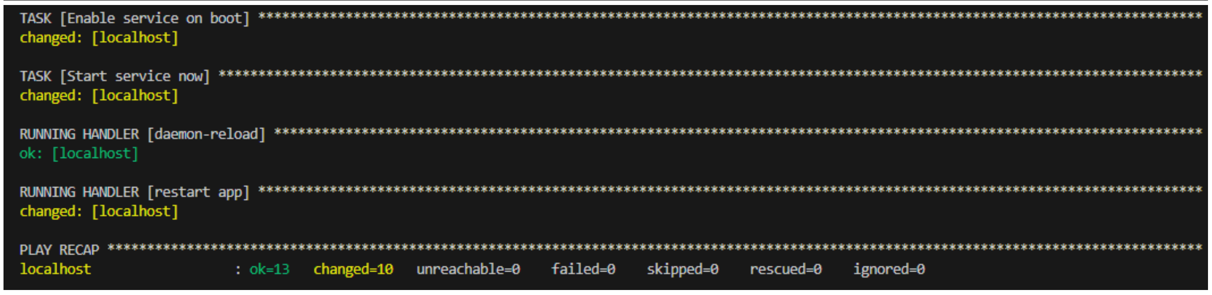


```
user1@DESKTOP-V7NDH77:/mnt/d/Downloads/Karunya/devops/devops9$ ansible -i inventory.ini all -m ping
[WARNING]: Ansible is being run in a world writable directory (/mnt/d/Downloads/Karunya/devops/devops9), ignoring it as an ansible.cfg source. For
more information see https://docs.ansible.com/ansible/devel/reference_appendices/config.html#cfg-in-world-writable-dir
localhost | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

Running the Ansible Playbook

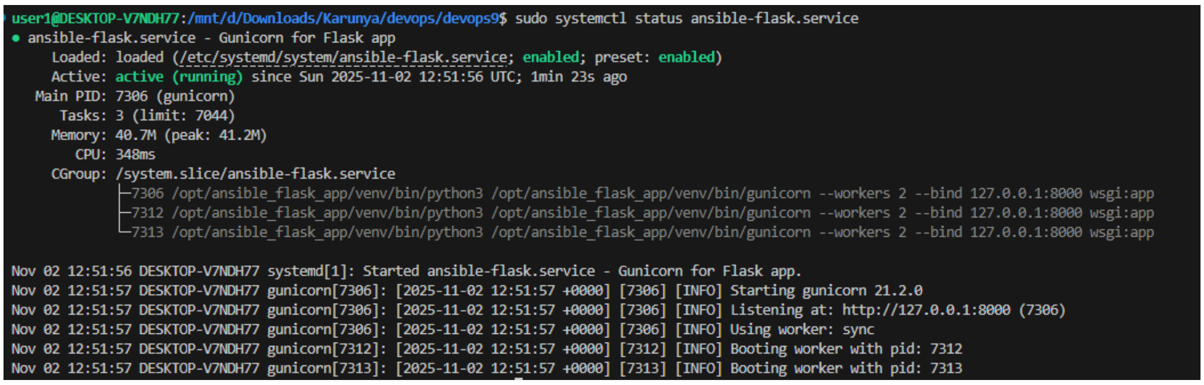


```
user1@DESKTOP-V7NDH77:/mnt/d/Downloads/Karunya/devops/devops9$ ansible-playbook playbook.yml -K
more information see https://docs.ansible.com/ansible/devel/reference_appendices/config.html#cfg-in-world-writable-dir
BECOME password:
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'

PLAY [Deploy Flask app to localhost with systemd] ****************************************************************************************

TASK [Gathering Facts] *******************************************************************************************************************
ok: [localhost]

TASK [Ensure Python and system tools present (apt-based; skip if macOS)] *****************************************************************
ok: [localhost]

TASK [Create application directory] ******************************************************************************************************
changed: [localhost]

TASK [Copy application source] ***********************************************************************************************************
changed: [localhost]

TASK [Create virtual environment] ********************************************************************************************************
changed: [localhost]

TASK [Upgrade pip] ***********************************************************************************************************************
changed: [localhost]

TASK [Install app dependencies] **********************************************************************************************************
changed: [localhost]

TASK [Render wsgi entrypoint] ************************************************************************************************************
changed: [localhost]

TASK [Place systemd unit] ****************************************************************************************************************
changed: [localhost]
```



```
TASK [Enable service on boot] ************************************************************************************************************
changed: [localhost]

TASK [Start service now] *****************************************************************************************************************
changed: [localhost]

RUNNING HANDLER [daemon-reload] **********************************************************************************************************
ok: [localhost]

RUNNING HANDLER [restart app] ************************************************************************************************************
changed: [localhost]

PLAY RECAP *******************************************************************************************************************************
localhost                  : ok=13   changed=10   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

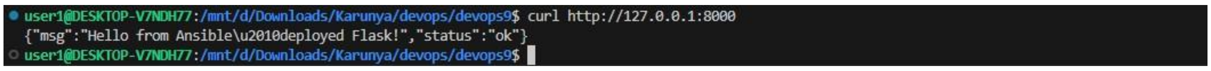
Checking the Service Status



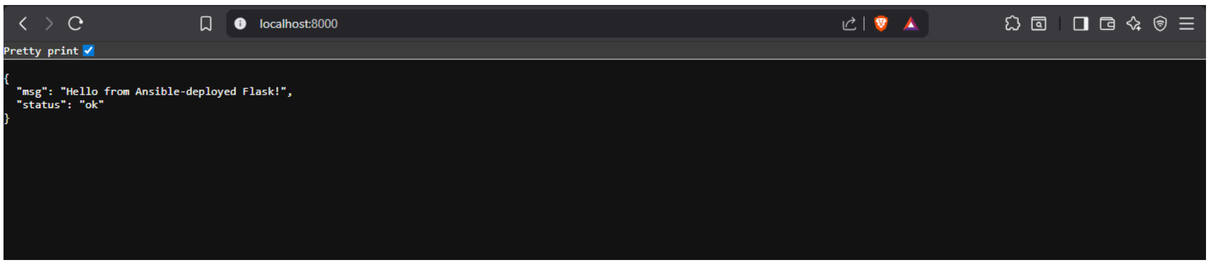
```
user1@DESKTOP-V7NDH77:/mnt/d/Downloads/Karunya/devops/devops9$ sudo systemctl status ansible-flask.service
● ansible-flask.service - Gunicorn for Flask app
     Loaded: loaded (/etc/systemd/system/ansible-flask.service; enabled; preset: enabled)
     Active: active (running) since Sun 2025-11-02 12:51:56 UTC; 1min 23s ago
   Main PID: 7306 (gunicorn)
      Tasks: 3 (limit: 7044)
     Memory: 40.7M (peak: 41.2M)
        CPU: 348ms
     CGroup: /system.slice/ansible-flask.service
             ├─7306 /opt/ansible_flask_app/venv/bin/python3 /opt/ansible_flask_app/venv/bin/gunicorn --workers 2 --bind 127.0.0.1:8000 wsgi:app
             ├─7312 /opt/ansible_flask_app/venv/bin/python3 /opt/ansible_flask_app/venv/bin/gunicorn --workers 2 --bind 127.0.0.1:8000 wsgi:app
             └─7313 /opt/ansible_flask_app/venv/bin/python3 /opt/ansible_flask_app/venv/bin/gunicorn --workers 2 --bind 127.0.0.1:8000 wsgi:app

Nov 02 12:51:56 DESKTOP-V7NDH77 systemd[1]: Started ansible-flask.service - Gunicorn for Flask app.
Nov 02 12:51:57 DESKTOP-V7NDH77 gunicorn[7306]: [2025-11-02 12:51:57 +0000] [7306] [INFO] Starting gunicorn 21.2.0
Nov 02 12:51:57 DESKTOP-V7NDH77 gunicorn[7306]: [2025-11-02 12:51:57 +0000] [7306] [INFO] Listening at: http://127.0.0.1:8000 (7306)
Nov 02 12:51:57 DESKTOP-V7NDH77 gunicorn[7306]: [2025-11-02 12:51:57 +0000] [7306] [INFO] Using worker: sync
Nov 02 12:51:57 DESKTOP-V7NDH77 gunicorn[7312]: [2025-11-02 12:51:57 +0000] [7312] [INFO] Booting worker with pid: 7312
Nov 02 12:51:57 DESKTOP-V7NDH77 gunicorn[7313]: [2025-11-02 12:51:57 +0000] [7313] [INFO] Booting worker with pid: 7313
```

Output:

Testing Flask Application Output



```
user1@DESKTOP-V7NDH77:/mnt/d/Downloads/Karunya/devops/devops9$ curl http://127.0.0.1:8000
{"msg":"Hello from Ansible\u2010deployed Flask!","status":"ok"}
user1@DESKTOP-V7NDH77:/mnt/d/Downloads/Karunya/devops/devops9$
```



```
{
  "msg": "Hello from Ansible-deployed Flask!",
  "status": "ok"
}
```

Result:

Thus, the Flask web app was successfully deployed on the local machine using Ansible with Gunicorn and systemd, ensuring automated setup and idempotent execution.

Ex. No.:10	Ansible for Docker & Kubernetes —(Local)
Date:05-11-25	

Objective

To automate the deployment of an NGINX web server using Ansible by configuring and managing containerized environments with Docker and Kubernetes (kind) on a local Ubuntu/WSL system.

Tools/ Software Required

- **Ubuntu / WSL :** for running Ansible and Docker locally
- **Ansible :** for automating Docker and Kubernetes setup and deployment
- **Python 3 :** required dependency for Ansible
- **Docker Engine / Docker Desktop :** container runtime environment
- **NGINX (Docker image: nginx:alpine) :** lightweight web server used in deployment
- **kubectl :** Kubernetes command-line tool to manage clusters and resources
- **kind (Kubernetes in Docker) :** to create and manage a local Kubernetes cluster

Industry Relevance and Applications

- **Infrastructure Automation:** Ansible is widely used in DevOps pipelines for automating infrastructure setup, configuration management, and application deployment across servers and cloud environments.
- **Containerized Deployments**: Docker and Kubernetes have become industry standards for packaging, deploying, and scaling applications in microservices-based architectures. Automating these with Ansible reflects real-world CI/CD workflows used in production systems.
- **Platform Engineering:** The experiment demonstrates how automation tools integrate with container orchestration, forming the foundation of modern platform engineering and site reliability practices.
- **Cloud-Native Environments:** The same approach used locally with kind can be extended to manage cloud Kubernetes clusters (AWS EKS, Azure AKS, Google GKE), aligning directly with enterprise DevOps operations.

Description

This experiment demonstrates deploying an NGINX web server using Ansible to automate both Docker- and Kubernetes-based setups on a local Ubuntu/WSL environment. It covers provisioning containers, creating a local Kubernetes cluster using kind, and managing services through Ansible playbooks to simulate a real-world containerized deployment workflow.

Ansible:

- An open-source automation tool used for provisioning, configuration management, and application deployment.
- In this setup, Ansible automates Docker installation, container management, Kubernetes setup, and deployment of NGINX services.

Docker:

- A containerization platform that packages applications with all dependencies for consistent execution across environments.
- Used here to deploy an NGINX container running on port 8080, managed directly via Ansible.

Kubernetes (kind):

- Kubernetes (K8s) is an orchestration platform for automating container deployment, scaling, and management.
- kind (Kubernetes in Docker) creates a lightweight local cluster for testing and development.
- Ansible automates cluster creation, deployment of the NGINX app, and service exposure via port-forwarding.

NGINX:

- A high-performance web server used to serve static content or act as a reverse proxy.
- Deployed both as a Docker container and as a Kubernetes pod to demonstrate portability and scalability across environments.

Ansible Collections:

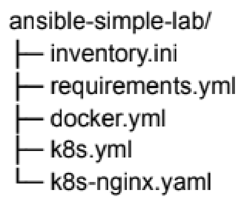
- **community.docker** – Enables Docker container management through Ansible modules.
- **kubernetes.core** – Provides Ansible modules to manage Kubernetes objects declaratively.

Command list with explanation:

Command	Explanation
ansible-galaxy collection install -r requirements.yml	Installs required Ansible collections.
ansible-playbook -i inventory.ini docker.yml	Deploys NGINX using Docker via Ansible.
sudo systemctl status docker	Checks Docker service status.
docker ps	Lists running Docker containers.
ansible-playbook -i inventory.ini k8s.yml	Deploys NGINX on Kubernetes using Ansible.
kind get clusters	Lists existing kind clusters.
kubectl get pods	Displays running pods.
kubectl get svc	Shows active services.
kubectl port-forward svc/nginx 8081:80	Forwards port 8081 to NGINX service.
curl http://localhost:8080	Tests Docker NGINX access.
curl http://localhost:8081	Tests Kubernetes NGINX access.

Procedure:

1. **Set up the Project Directory**
   - Create a folder named ansible-simple-lab/
   - Inside it, create the following structure:



```
ansible-simple-lab/
├── inventory.ini
├── requirements.yml
├── docker.yml
├── k8s.yml
└── k8s-nginx.yaml
```

2. **Set Up Inventory**
   - **Open inventory.ini and add:**

```
[local]
127.0.0.1 ansible_connection=local
```

3. **Install Required Collections**
   - **In requirements.yml, specify:**

```
collections:
  - community.docker
  - kubernetes.core
```

   - **Install the collections:**

```
ansible-galaxy collection install -r requirements.yml
```

4. **Create Docker Playbook**
   - **Create a file named docker.yml with the following content:**

```
---
- hosts: local
  become: true
  tasks:
    - name: Install Docker (quick script)
      ansible.builtin.shell: |
        set -e
        if ! command -v docker >/dev/null 2>&1; then
          curl -fsSL https://get.docker.com | sh
        fi
      args:
        creates: /usr/bin/docker
    - name: Ensure Docker is running
      ansible.builtin.service:
        name: docker
        state: started
        enabled: true
    - name: Run NGINX container on 8080
      community.docker.docker_container:
      name: web
        image: nginx:alpine
        published_ports:
          - "8080:80"
        restart_policy: always
    - name: Show URL
      ansible.builtin.debug:
        msg: "Open http://localhost:8080"
```

5. **Run Docker Playbook**

- Execute the playbook to deploy NGINX in Docker:

  ```
  ansible-playbook -i inventory.ini docker.yml
  ```

- Open http://localhost:8080 in browser

6. **Create Kubernetes Playbook**

- Create a file named k8s.yml with the following:

```
---
- hosts: local
  become: true
  tasks:
    - name: Install kubectl (once)
      ansible.builtin.shell: |
        set -e
        if ! command -v kubectl >/dev/null 2>&1; then
          curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s \
          https://storage.googleapis.com/kubernetes-
release/release/stable.txt)/bin/linux/amd64/kubectl"
        install -m 0755 kubectl /usr/local/bin/kubectl
        fi
      args:
```

```
        creates: /usr/local/bin/kubectl
    - name: Install kind (once)
      ansible.builtin.shell: |
        set -e
        if ! command -v kind >/dev/null 2>&1; then
          curl -Lo kind https://kind.sigs.k8s.io/dl/stable/kind-linux-amd64
          install -m 0755 kind /usr/local/bin/kind
        fi
      args:
        creates: /usr/local/bin/kind
    - name: Create kind cluster if missing
      ansible.builtin.shell: |
        set -e
        kind get clusters | grep -q '^simple$' || kind create cluster --name simple
      args:
        creates: /root/.kube/config
    - name: Apply NGINX Deployment & Service
      kubernetes.core.k8s:
        state: present
        definition: "{{ lookup('file', playbook_dir + '/k8s-nginx.yaml') | from_yaml_all |
list }}"
    - name: Show services
      ansible.builtin.shell: kubectl get svc
      register: svc
      changed_when: false
    - debug: var=svc.stdout_lines
ansible-playbook playbook.yml -K
```

7. Create Kubernetes Deployment File (k8s-nginx.yaml)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
```

```
spec:
  replicas: 1
  selector:
  matchLabels:
      app: nginx
  template:
  metadata:
      labels:
        app: nginx
    spec:
    containers:
        - name: nginx
          image: nginx:alpine
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  selector:
    app: nginx
  ports:
    - port: 80
      targetPort: 80
  type: NodePort
```

8. Run Kubernetes Playbook

- Execute:

```
ansible-playbook -i inventory.ini k8s.yml
```

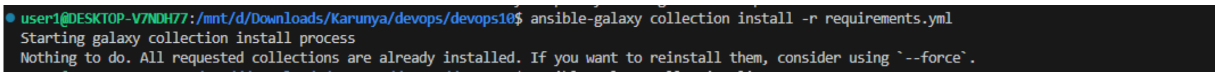
9. **Access NGINX on Kubernetes**

- Forward the service port and open it in the browser:

  kubectl port-forward svc/nginx 8081:80

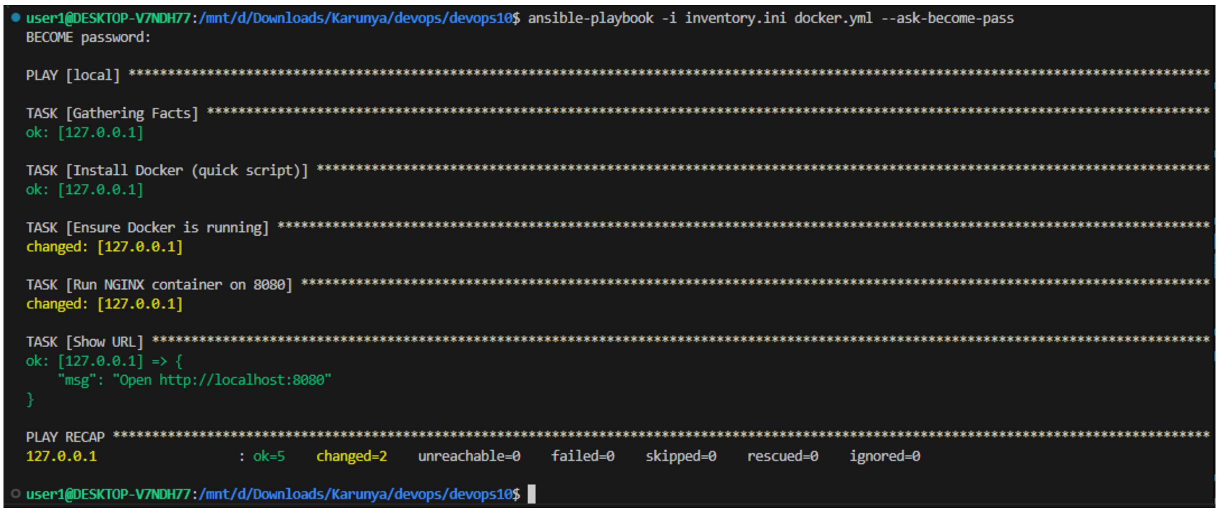
Sample Inputs:

Installing Required Ansible Collections



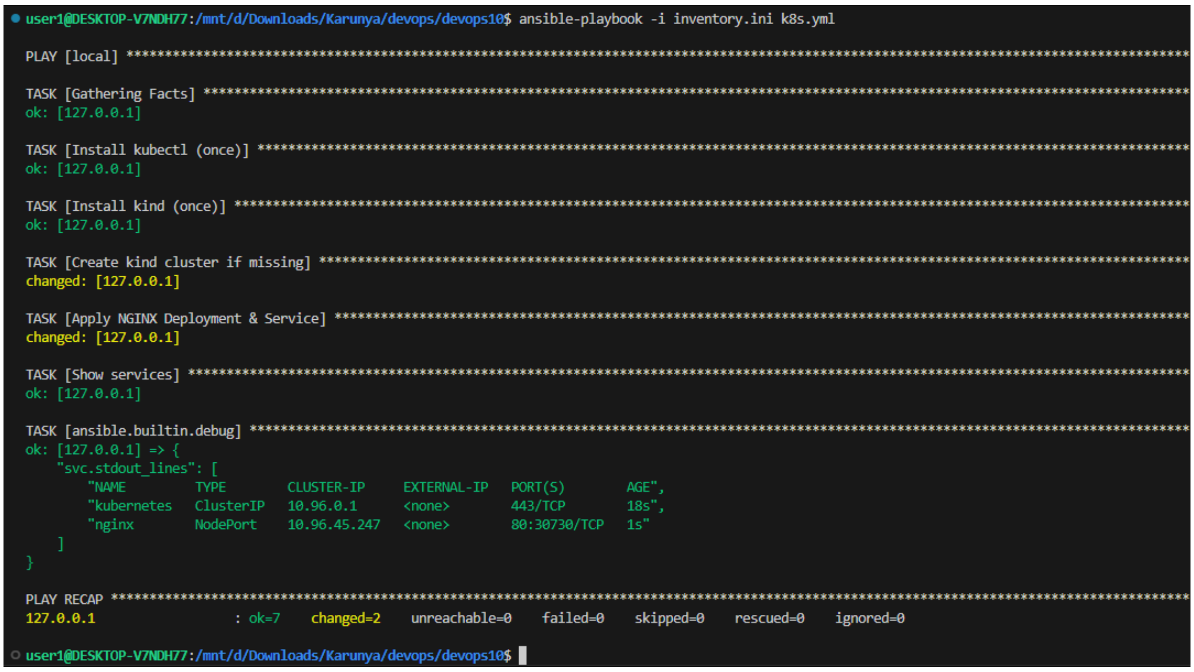
```
user1@DESKTOP-V7NDH77:/mnt/d/Downloads/Karunya/devops/devops10$ ansible-galaxy collection install -r requirements.yml
Starting galaxy collection install process
Nothing to do. All requested collections are already installed. If you want to reinstall them, consider using `--force`.
```

Running Ansible Playbook to Deploy NGINX on Docker



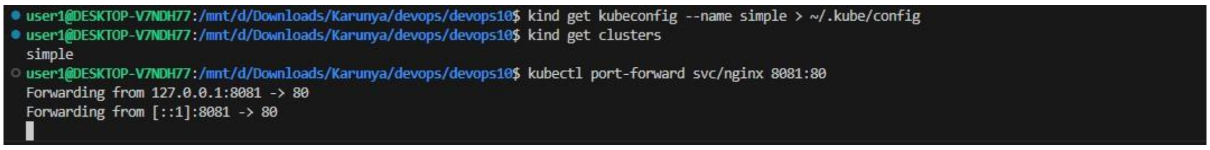
```
user1@DESKTOP-V7NDH77:/mnt/d/Downloads/Karunya/devops/devops10$ ansible-playbook -i inventory.ini docker.yml --ask-become-pass
BECOME password:

PLAY [local] ************************************************************************************************************

TASK [Gathering Facts] **************************************************************************************************
ok: [127.0.0.1]

TASK [Install Docker (quick script)] ************************************************************************************
ok: [127.0.0.1]

TASK [Ensure Docker is running] *****************************************************************************************
changed: [127.0.0.1]

TASK [Run NGINX container on 8080] **************************************************************************************
changed: [127.0.0.1]

TASK [Show URL] *********************************************************************************************************
ok: [127.0.0.1] => {
    "msg": "Open http://localhost:8080"
}

PLAY RECAP **************************************************************************************************************
127.0.0.1                  : ok=5    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

user1@DESKTOP-V7NDH77:/mnt/d/Downloads/Karunya/devops/devops10$
```

Deploying NGINX on Kubernetes using Ansible



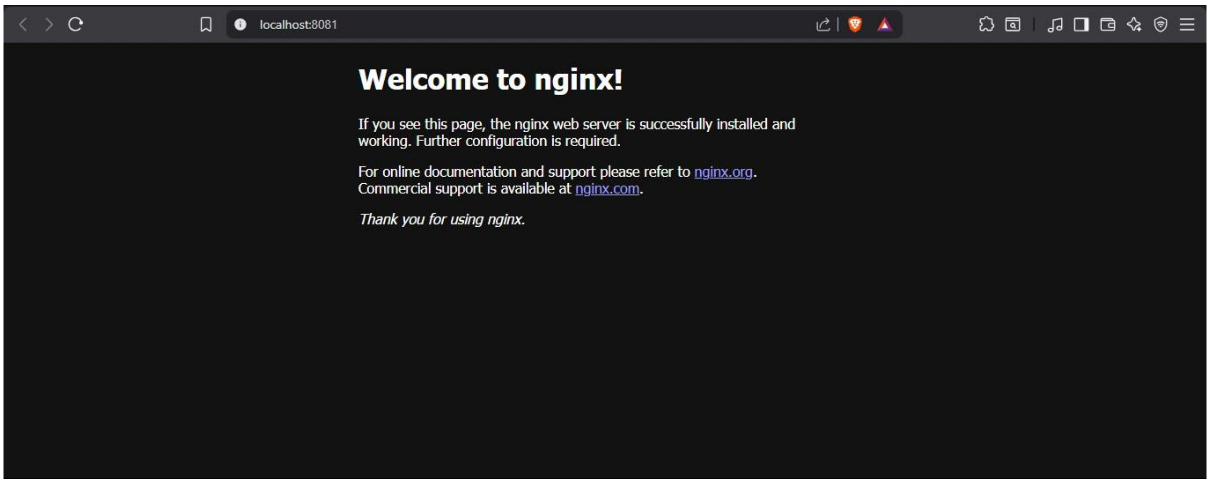
```
user1@DESKTOP-V7NDH77:/mnt/d/Downloads/Karunya/devops/devops10$ ansible-playbook -i inventory.ini k8s.yml

PLAY [local] ************************************************************************************************************

TASK [Gathering Facts] **************************************************************************************************
ok: [127.0.0.1]

TASK [Install kubectl (once)] *******************************************************************************************
ok: [127.0.0.1]

TASK [Install kind (once)] **********************************************************************************************
ok: [127.0.0.1]

TASK [Create kind cluster if missing] ***********************************************************************************
changed: [127.0.0.1]

TASK [Apply NGINX Deployment & Service] *********************************************************************************
changed: [127.0.0.1]

TASK [Show services] ****************************************************************************************************
ok: [127.0.0.1]

TASK [ansible.builtin.debug] ********************************************************************************************
ok: [127.0.0.1] => {
    "svc.stdout_lines": [
        "NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE",
        "kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP        18s",
        "nginx        NodePort    10.96.45.247   <none>        80:30730/TCP   1s"
    ]
}

PLAY RECAP **************************************************************************************************************
127.0.0.1                  : ok=7    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

user1@DESKTOP-V7NDH77:/mnt/d/Downloads/Karunya/devops/devops10$
```

Exporting kind Cluster Configuration to kubeconfig and Accessing NGINX Service via Port Forwarding



```
● user1@DESKTOP-V7NDH77:/mnt/d/Downloads/Karunya/devops/devops10$ kind get kubeconfig --name simple > ~/.kube/config
● user1@DESKTOP-V7NDH77:/mnt/d/Downloads/Karunya/devops/devops10$ kind get clusters
  simple
○ user1@DESKTOP-V7NDH77:/mnt/d/Downloads/Karunya/devops/devops10$ kubectl port-forward svc/nginx 8081:80
  Forwarding from 127.0.0.1:8081 -> 80
  Forwarding from [::1]:8081 -> 80
  ▌
```

Output:



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Result:

Thus, successfully automated the deployment of an NGINX web server using Ansible, managing containerized environments with Docker and Kubernetes (kind) on a local Ubuntu/WSL setup.