

### ****Step 1: Setting Up Docker on Windows****

### 

### ****Step 2: Prepare Your Application with Docker****

#### **2.1. Create a Dockerfile**

The **Dockerfile** is a script that defines the environment and steps required to build a Docker image for your application. Here's an example of a simple Dockerfile for a Python-based web app using Flask.

# Use the official Python image as the base image

FROM python:3.9-slim

# Set the working directory in the container

WORKDIR /app

# Copy the application files into the container

COPY . /app

# Install the required dependencies

RUN pip install -r requirements.txt

# Expose the port the app will run on

EXPOSE 5000

# Command to run the application

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

1. Place the Dockerfile in the root of your web app project.
2. The requirements.txt file should list all necessary Python dependencies (e.g., Flask, Gunicorn).

#### 2.2. **Build the Docker Image**

To build the Docker image for your app, navigate to the directory containing your Dockerfile and run the following command:

docker build -t my-web-app .

This will create a Docker image called my-web-app. You can verify the image was created using:

docker images

### ****Step 3: Running Docker Containers****

After building the Docker image, you can run it as a container:

docker run -d -p 5000:5000 --name web-app my-web-app

* -d runs the container in detached mode.
* -p 5000:5000 maps port 5000 from the container to port 5000 on your host.
* --name web-app assigns a name to the running container.

To check if the container is running:

docker ps

### ****Step 4: Automate Deployment Using Ansible and Docker****

Now that your application is containerized, you can use **Ansible** to automate the deployment of the Docker container across multiple servers (whether on Windows or Linux). This can be done by leveraging **Docker-related Ansible modules**.

#### 4.1. **Install Ansible and Docker on Windows (WSL)**

1. **Install Ansible** on your WSL environment (as mentioned in previous steps).
2. Install **Docker** on your target machines (Linux or Windows-based systems) using Docker Desktop or Docker Engine, depending on the environment.

#### 4.2. **Ansible Docker Modules**

Ansible provides several Docker-related modules that can help you manage Docker containers, images, and volumes.

* **docker\_image**: Build or pull Docker images.
* **docker\_container**: Create, start, and manage Docker containers.

For example, to deploy a Docker container, you can use Ansible’s docker\_container module.

### ****Step 5: Create the Ansible Playbook for Docker Deployment****

You can now write an **Ansible playbook** to automate the Docker container deployment. Here’s an example playbook (deploy\_docker\_app.yml) that automates the deployment of a Docker container for your web application.

#### Example Playbook:

---

- name: Deploy web app using Docker

hosts: all

become: yes

tasks:

- name: Pull the Docker image

docker\_image:

name: my-web-app

source: build

path: /path/to/your/Dockerfile # Path where Dockerfile is located

force: yes

- name: Stop and remove any existing container

docker\_container:

name: web-app

state: absent

- name: Run the web app container

docker\_container:

name: web-app

image: my-web-app

state: started

restart\_policy: always

published\_ports:

- "5000:5000"

#### Explanation:

1. **docker\_image**: This task pulls the my-web-app image or builds it from a local Dockerfile located at /path/to/your/Dockerfile. You can use source: pull to fetch the image from a Docker registry instead of building it locally.
2. **docker\_container**: This task ensures that any previously running container with the name web-app is stopped and removed. Then it starts a new container with the my-web-app image, mapping the container’s port 5000 to the host’s port 5000.

### ****Step 6: Run the Ansible Playbook****

To run this playbook, you’ll need to specify your inventory file and run the following command:

ansible-playbook -i inventory/hosts.yml deploy\_docker\_app.yml

Where the inventory/hosts.yml file contains the list of your target hosts (either Linux or Windows servers) where Docker is installed.

#### Example inventory file (hosts.yml):

[web\_servers]

linux-server-1.example.com

windows-server-1.example.com

[web\_servers:vars]

ansible\_user=your\_user

ansible\_password=your\_password

ansible\_connection=winrm # for Windows

ansible\_winrm\_transport=ntlm # for Windows

### ****Step 7: Continuous Integration/Continuous Deployment (CI/CD)****

To integrate this deployment process into a CI/CD pipeline (e.g., using **GitLab CI**, **Jenkins**, or **GitHub Actions**), you can:

1. Trigger the CI/CD pipeline after a successful commit or merge.
2. Build the Docker image.
3. Run the Ansible playbook to deploy the updated container to your servers.

Here’s an example of a .gitlab-ci.yml file for GitLab CI:

stages:

- build

- deploy

build:

stage: build

script:

- docker build -t my-web-app .

deploy:

stage: deploy

script:

- ansible-playbook -i inventory/hosts.yml deploy\_docker\_app.yml

### ****Conclusion****

Using **Docker** in combination with **Ansible** provides a highly efficient way to deploy web applications across multiple servers, whether they are Windows or Linux-based. Docker allows you to containerize your application, making it portable and ensuring consistent environments across all deployment targets. Ansible, on the other hand, automates the setup, management, and orchestration of Docker containers, ensuring a smooth and repeatable deployment process. This approach is perfect for environments where you need flexibility, consistency, and ease of scaling.