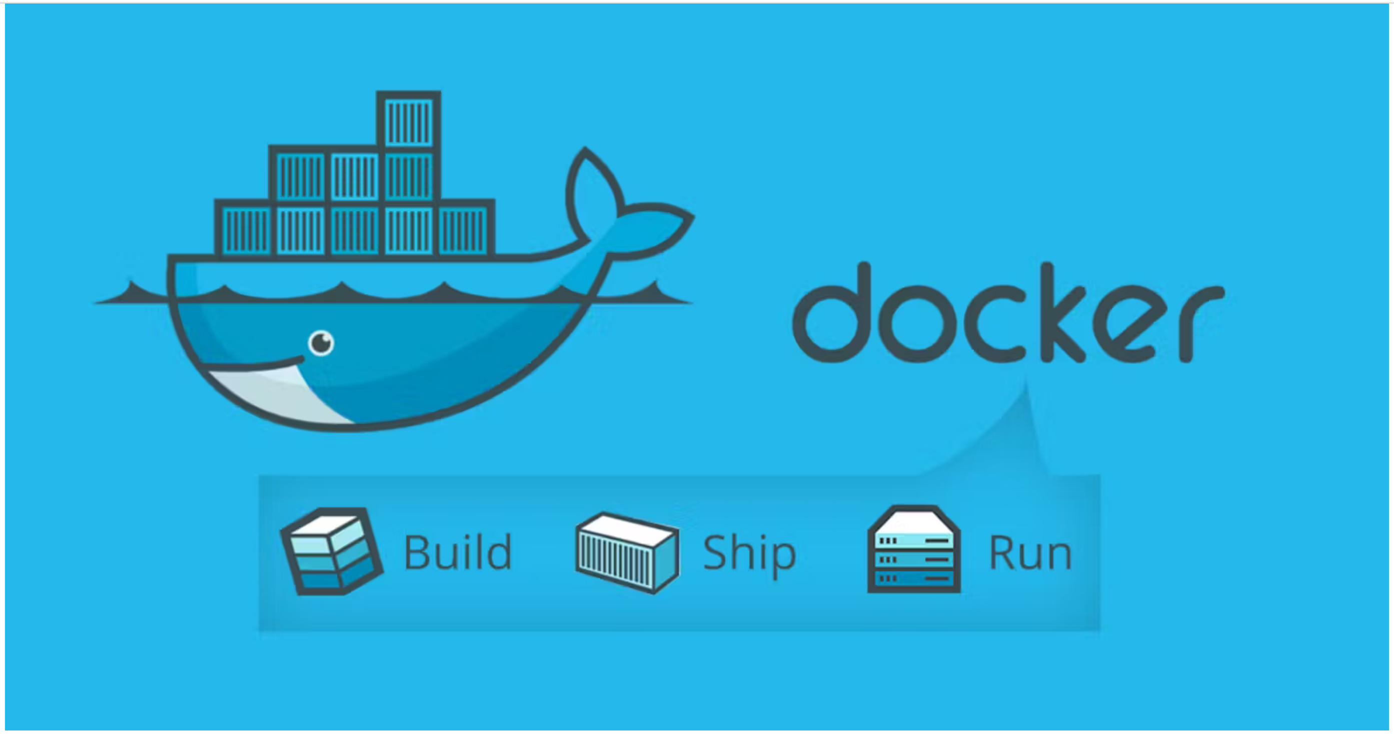


Docker for DevOps Engineers

Day 16 : 90Days of DevOps Challenge

SS Shivraj Salunkhe · Apr 24, 2023 · 9 min read

TABLE OF CONTENTS

Task :

Creating Docker Image , Docker Container Using Docker.

Important Docker Commands

Docker is a popular tool used by DevOps Engineers for containerization and managing application deployments.

Here are some key points to know about Docker for DevOps:

1. **Containerization**: Docker uses containerization to package applications and their dependencies into a single package that can run consistently across different environments. Containers are isolated from the host operating system and other containers, making them a lightweight and portable way to package and deploy applications.
2. **Standardization**: Docker provides a standardized way to package and deploy applications, making it easier to manage and scale applications across different environments. With Docker, DevOps Engineers can ensure that applications run consistently across development, testing, and production environments.
3. **Microservices**: Docker is often used in conjunction with microservices architecture to deploy small, independent services that work together to form a larger application. Docker makes it easy to package and deploy microservices as separate containers, allowing for greater flexibility and scalability.
4. **Continuous Integration and Deployment (CI/CD)**: Docker is commonly used in CI/CD pipelines to automate the build, test, and deployment process of

applications. DevOps Engineers can use Docker to build and test applications in isolated environments, ensuring that they are ready for deployment.

5. **Infrastructure as Code (IaC)**: Docker can be used as a part of IaC practices, allowing DevOps Engineers to manage infrastructure in a declarative way. With Docker, infrastructure can be defined and deployed as code, making it easier to manage and scale infrastructure across different environments.

These are some of the ways that Docker is used by DevOps Engineers. With its focus on containerization, standardization, microservices, CI/CD, and IaC, Docker has become an essential tool for managing modern application deployments.

Task :

Creating Docker Image , Docker Container Using Docker.

Steps to make Docker Image :

-> first of all connect with the aws ec2 instance by using ssh

Go to your aws account >> go to connect instance >> select ssh client >> copy that ssh key.

For that use Below command :

```
sudo (your_ssh_key)
```

-> Create a new Repository

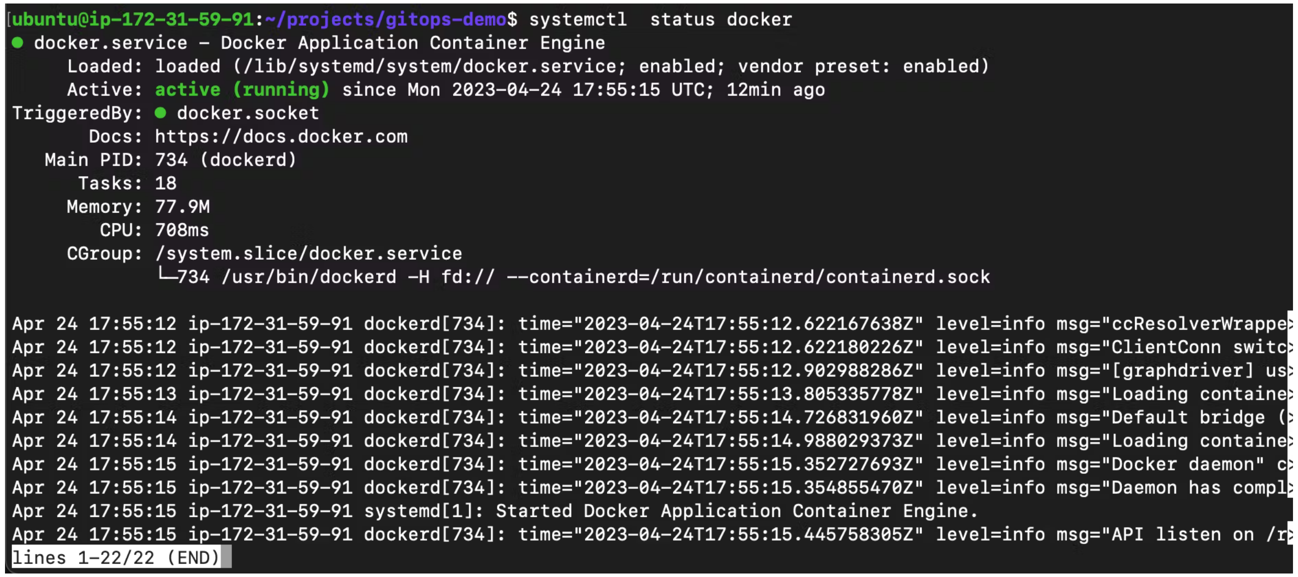
For that, use >> mkdir projects then >> cd projects

-> Clone the Project from Github url for creation of the automation process ie.Docker image.

For that use Below command :

```
git clone https://github.com/webmagicinformatica/gitops-demo.git
```

-> To show the Docker status

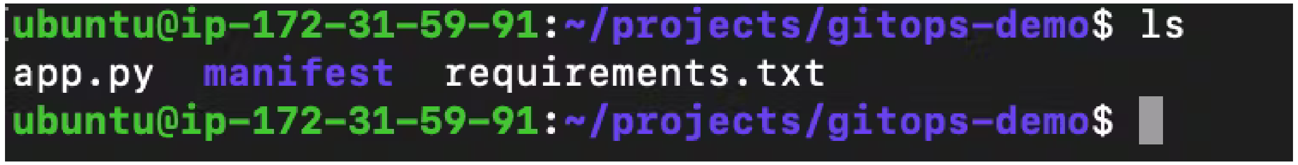


```
ubuntu@ip-172-31-59-91:~/projects/gitops-demo$ systemctl  status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2023-04-24 17:55:15 UTC; 12min ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 734 (dockerd)
      Tasks: 18
     Memory: 77.9M
        CPU: 708ms
     CGroup: /system.slice/docker.service
             └─734 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Apr 24 17:55:12 ip-172-31-59-91 dockerd[734]: time="2023-04-24T17:55:12.622167638Z" level=info msg="ccResolverWrappe
Apr 24 17:55:12 ip-172-31-59-91 dockerd[734]: time="2023-04-24T17:55:12.622180226Z" level=info msg="ClientConn switc
Apr 24 17:55:12 ip-172-31-59-91 dockerd[734]: time="2023-04-24T17:55:12.902988286Z" level=info msg="[graphdriver] us
Apr 24 17:55:13 ip-172-31-59-91 dockerd[734]: time="2023-04-24T17:55:13.805335778Z" level=info msg="Loading containe
Apr 24 17:55:14 ip-172-31-59-91 dockerd[734]: time="2023-04-24T17:55:14.726831960Z" level=info msg="Default bridge (
Apr 24 17:55:14 ip-172-31-59-91 dockerd[734]: time="2023-04-24T17:55:14.988029373Z" level=info msg="Loading containe
Apr 24 17:55:15 ip-172-31-59-91 dockerd[734]: time="2023-04-24T17:55:15.352727693Z" level=info msg="Docker daemon" c
Apr 24 17:55:15 ip-172-31-59-91 dockerd[734]: time="2023-04-24T17:55:15.354855470Z" level=info msg="Daemon has compl
Apr 24 17:55:15 ip-172-31-59-91 systemd[1]: Started Docker Application Container Engine.
Apr 24 17:55:15 ip-172-31-59-91 dockerd[734]: time="2023-04-24T17:55:15.445758305Z" level=info msg="API listen on /r
lines 1-22/22 (END)
```

checks that your docker status is showing **active(running)** if it is not showing then you need to properly install setup of docker.

-> Checking Dockerfile present or not in the Project that we are cloning from Github.



```
ubuntu@ip-172-31-59-91:~/projects/gitops-demo$ ls
app.py  manifest  requirements.txt
ubuntu@ip-172-31-59-91:~/projects/gitops-demo$ 
```

-> If not Present then Create docker file named *"Dockerfile"*

For that use Below command :

```
vim Dockerfile
```

-> Add below Commands in the Dockerfile to automate the Process.

-> Use *"Cat"* command to show the Contents in the Dockerfile.

-> To show the docker status

For that use Below command :

```
docker ps
```

-> For building the Docker Image use below Command :

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

hence, your docker image in formed successfully by using above commands.

-> To Show the Docker image is created or not.

-> To create a Docker Container use below command :

This will create your docker container and shows by using below command:

```
sudo docker ps
```

by using above command you can easily see that your docker container is succesfully created or not.

Important Docker Commands

1. **docker inspect**
   Use:
   `docker inspect` is a Docker command that is used to retrieve low-level information about a Docker object. This command can be used to get detailed information about a container, image, volume, or network, including their configuration, status, and metadata.
   Command Syntax :

   ```
   docker inspect [OPTIONS] OBJECT [OBJECT...]
   ```

- `OPTIONS`: Additional options that modify the behavior of the `docker inspect` command.
- `OBJECT`: One or more Docker objects (containers, images, volumes, or networks) that you want to inspect.

Example:

To inspect a running container with the name "my-container", you can use the following command:

```
docker inspect my-container
```

This will return a large JSON object that contains detailed information about the container, including its configuration, network settings, and storage volumes. You can also use additional options with the `docker inspect` command to customize the output or retrieve specific information about the container. For example, you can use the `--format` option to output the results in a specific format, or the `--type` option to filter the output by object type.

1. **docker port**
   Use:
   `docker port` is a Docker command that is used to list the public-facing port mappings for a container. This command can be used to get information about

which ports are exposed by a container and how they are mapped to the host system.

Command Syntax:

```
docker port CONTAINER [PRIVATE_PORT[/PROTO]]
```

- `CONTAINER`: The name or ID of the container for which you want to list the port mappings.
- `PRIVATE_PORT`: The port number inside the container that you want to map to a host port.
- `PROTO`: The protocol to use (TCP or UDP).

Example:

To list the port mappings for a container with the name "my-container", you can use the following command:

```
docker port my-container
```

This will list all the public-facing port mappings for the container, including the container port and the corresponding host port. If you want to get information about a specific port mapping, you can specify the `PRIVATE_PORT` and `PROTO` parameters in the command. For example, to get the mapping for port 80, you can use the following command:

```
docker port my-container 80/tcp
```

This will return the host IP address and port number to which port 80 inside the container is mapped.

1. **docker stats**
   Use:
   `docker stats` is a Docker command that is used to display a live stream of resource usage statistics for all running containers. This command can be used to monitor the CPU, memory, and network usage of Docker containers in real-time.
   Command Syntax:

   ```
   docker stats [OPTIONS] [CONTAINER...]
   ```

   - `OPTIONS`: Additional options that modify the behavior of the `docker stats` command.
   - `CONTAINER`: One or more container names or IDs for which you want to display resource usage statistics. If no container is specified, this command will display statistics for all running containers.

Example:

To display resource usage statistics for all running containers, you can use the following command:

```
docker stats
```

This will display a live stream of resource usage statistics for all running containers, including the CPU usage, memory usage, and network I/O usage. If you want to display statistics for specific containers, you can specify their names or IDs as parameters in the command. For example, to display statistics for two containers with the names "webapp1" and "webapp2", you can use the following command:

```
docker stats webapp1 webapp2
```

This will display resource usage statistics for only the two specified containers. You can also use additional options with the `docker stats` command to customize the output or filter the containers by specific criteria, such as labels or network.

1. **docker top**
   Use:

`docker top` is a Docker command that is used to display the running processes of one or more containers. This command can be used to get information about the processes that are running inside a container and their resource usage.

Command Syntax:

```
docker top CONTAINER [ps OPTIONS]
```

- `CONTAINER`: The name or ID of the container for which you want to list the running processes.
- `ps OPTIONS`: Additional options that modify the behavior of the `ps` command that is executed inside the container.

Example:

To display the running processes of a container with the name "my-container", you can use the following command:

```
docker top my-container
```

This will display a list of the running processes inside the container, including the process ID, the user that owns the process, the CPU usage, and the memory usage. If you want to get more information about the running processes, you can use additional options with the `ps` command. For example, you can use the `-ef` option to display the full command line and environment variables of each process:

COPY

```
docker top my-container ps -ef
```

This will display the same list of running processes, but with additional information about each process. If you want to display the running processes of multiple containers at once, you can specify their names or IDs as parameters in the command.

1. **docker save**
   Use:
   `docker save` is a Docker command that is used to save one or more Docker images to a tar archive. This command can be used to backup Docker images, move them to another Docker host or registry, or share them with others.
   Command Syntax:

   COPY

   ```
   docker save [OPTIONS] IMAGE [IMAGE...]
   ```

   - `OPTIONS`: Additional options that modify the behavior of the `docker save` command.
   - `IMAGE`: One or more image names or IDs that you want to save to a tar archive.

Example:

To save a Docker image with the name "my-image" to a tar archive, you can use the following command:

COPY

```
docker save my-image > my-image.tar
```

This will save the "my-image" Docker image to a tar archive named "my-image.tar" in the current working directory. You can also save multiple Docker images to a single tar archive by specifying their names or IDs as parameters in the command:

COPY

```
docker save my-image1 my-image2 > my-images.tar
```

This will save the "my-image1" and "my-image2" Docker images to a single tar archive named "my-images.tar" in the current working directory.

You can use additional options with the `docker save` command to customize the output format, exclude certain image layers or tags, or compress the tar archive. For example, you can use the `-o` option to specify the output file name, the `--exclude` option to exclude certain layers or tags, or the `-z` option to compress the tar archive with gzip.

1. **docker load**

Use:

`docker load` is a Docker command that is used to load one or more Docker images from a tar archive. This command can be used to restore Docker images that have been previously saved with the `docker save` command, or to import Docker images from another host or registry.

Command Syntax:

```
docker load [OPTIONS] < FILE
```

- `OPTIONS`: Additional options that modify the behavior of the `docker load` command.
- `FILE`: The path to the tar archive containing the Docker image or images that you want to load.

Example:

To load a Docker image from a tar archive named "my-image.tar", you can use the following command:

```
docker load < my-image.tar
```

This will load the Docker image from the "my-image.tar" tar archive into the local Docker image repository. You can also load multiple Docker images from a single tar archive by specifying the path to the archive as a parameter in the command:

```
docker load < my-images.tar
```

This will load all the Docker images contained in the "my-images.tar" tar archive into the local Docker image repository.

You can use additional options with the `docker load` command to customize the behavior of the command, such as excluding certain layers or tags, or specifying a different image repository or tag. For example, you can use the `--input` option to specify the path to the tar archive, the `--exclude` option to exclude certain layers or tags, or the `-q` option to suppress the progress output.

Thank You for Reading My Blog.. 👍

Connect with me : linkedin.com/in/shivraj-salunkhe-5881141a4

Follow my Blog channel : **shivrajofficial.hashnode.dev**

10

Subscribe to my newsletter

Read articles from directly inside your inbox.
Subscribe to the newsletter, and don't miss out.

Enter your email address SUBSCRIBE

Docker ci-cd Devops learning docker images

WRITTEN BY

Shivraj Salunkhe

Follow

Dedicated and hardworking undergraduate student pursuing a degree in Information Technology with a passion for learning, leadership experience, and real-world skills through internships and part-time jobs in IT sector. Actively looking for new Opportunities and committed to do personal and professional growth.

MORE ARTICLES

Shivraj Salunkhe

Dockerfile in Docker for DevOps

A Dockerfile is a text file that contains instructions for building a Docker image. The Dockerfile d...

Shivraj Salunkhe

Create CI/CD Pipeline of a web-app using Github, Jenkins, AWS and Docker.

This Blog helps you to create the CI/CD Pipeline of Web-Application named "Todo-List" to your Github...

Shivraj Salunkhe

Python Libraries for DevOps

-> Reading JSON in Python To read JSON files in Python, you can use the json module. import json # ...