

**07-08-2024**

## **DAILY TASK**

**1. Mention the actions of following comments: git remote add origin**

**"http://github/a.git" Git pull origin master Git push origin dev**

**git remote add origin "http://github/a.git"**

- This command adds a new remote repository to your local git repository and names it origin. The URL "http://github/a.git" is the address of the remote repository. After executing this command, you can use origin as a shorthand reference to interact with this remote repository.

**git pull origin master**

- This command fetches changes from the master branch of the remote repository named origin and merges them into your current branch. It combines git fetch (to get the latest changes) and git merge (to integrate those changes into your local branch) in one step.

**git push origin dev**

- This command pushes the commits from your local dev branch to the remote repository named origin, updating the remote dev branch to match your local branch. If the dev branch does not exist on the remote repository, it will be created.

**2. What are the functions of following Docker objects and key components:**

**Dockerd:**

**Dockerfile**

**Docker-compose.**

**yaml Docker**

## **Registries DockerHost**

### **1. Dockerd:**

- dockerd is the Docker daemon, a server-side application that manages Docker containers on your system. It listens for Docker API requests and handles container management operations such as building, running, and distributing Docker containers. It is responsible for all the actions taken by the Docker client commands (docker), such as starting, stopping, and creating containers.

### **2. Dockerfile:**

- A Dockerfile is a text file that contains a set of instructions to build a Docker image. Each instruction in a Dockerfile creates a layer in the image. Common instructions include specifying the base image (FROM), copying files (COPY), running commands (RUN), and setting environment variables (ENV). The Dockerfile allows for the automation of image creation, ensuring consistency and reproducibility.

### **3. Docker-compose.yaml:**

- A docker-compose.yaml file is used with Docker Compose to define and manage multi-container Docker applications. It allows you to specify the services, networks, and volumes needed for your application in a single file. By using the docker-compose up command, Docker Compose will create and start all the services defined in the docker-compose.yaml file. This is useful for defining the entire architecture of your application, including dependencies between services.

#### **4. Docker Registries:**

- Docker registries are storage and distribution systems for Docker images. The most commonly used registry is Docker Hub, but there are other public and private registries available. A registry allows you to push and pull Docker images to and from a central repository, making it easier to share images with others or deploy them to different environments.

#### **5. DockerHost:**

- A Docker Host is a physical or virtual machine configured to run Docker containers. It runs the Docker daemon (dockerd) and provides the necessary environment for container execution. The Docker Host manages the lifecycle of containers, including starting, stopping, and scaling them. It also handles networking, storage, and other resources needed for containers to run.

Each of these components plays a crucial role in the Docker ecosystem, contributing to the process of containerization, orchestration, and management of applications.

### **3.What's the isolation in Docker container?**

Isolation in Docker containers refers to the ability to run applications in an environment that is separate from the host system and other containers. This is achieved through various mechanisms provided by Docker and underlying technologies like Linux namespaces and control groups (cgroups). Here are the key aspects of isolation in Docker containers:

#### **1. Process Isolation:**

- Each container runs its own set of processes, separate from other containers and the host system. This is achieved using Linux namespaces, which provide isolated process trees, user IDs, and interprocess communication (IPC) resources.

## **2. Filesystem Isolation:**

- Containers have their own isolated filesystems. The container's filesystem is created from the Docker image and can include additional layers specific to the container. This isolation ensures that changes made in the container's filesystem do not affect the host system or other containers.

## **3. Network Isolation:**

- Containers can have their own network stacks, including isolated IP addresses, network interfaces, and routing tables. Docker uses network namespaces to provide this isolation. Containers can be connected to different networks or share a network depending on the configuration.

## **4. Resource Isolation:**

- Docker uses control groups (cgroups) to allocate and limit resources such as CPU, memory, block I/O, and network bandwidth for each container. This ensures that one container cannot consume all the resources of the host system, providing better performance and stability.

## **5. Security Isolation:**

- Docker employs several security mechanisms to isolate containers and protect the host system. These include user namespaces, which can map container users to different users on the host system, and security modules like AppArmor, SELinux, and seccomp, which provide additional access controls and sandboxing.

## **6. Namespace Isolation:**

- **Docker uses various types of namespaces to provide different levels of isolation:**

- **PID Namespace:** Isolates the process ID space, so processes in different containers have separate PID trees.

- UTS Namespace: Isolates hostname and domain name, allowing containers to have their own UTS (UNIX Time-sharing System) namespace.
- Mount Namespace: Isolates the filesystem mount points, so containers can have different views of the filesystem.
- IPC Namespace: Isolates IPC resources, so containers have separate shared memory, semaphores, and message queues.
- Network Namespace: Isolates network interfaces, IP addresses, port numbers, and routing tables.
- User Namespace: Isolates user and group IDs, providing different user mappings for containers and the host system. By leveraging these isolation mechanisms, Docker containers can run applications securely and efficiently, minimizing the risk of interference between containers and ensuring that the host system remains stable and unaffected by containerized workloads

## Docker Examples:

### 1.Pull image from Docker hub:

```
C:\Users\SOWNTHARI.RP>docker pull node:latest
latest: Pulling from library/node
ca4e5d672725: Pull complete
30b93c12a9c9: Pull complete
10d643a5fa82: Pull complete
d6dc1019d793: Pull complete
81bff076e6cf: Pull complete
1171ed9a56f6: Pull complete
fe9b706f3e3d: Pull complete
512b19417822: Pull complete
Digest: sha256:72314283e7a651d65a367f4e72fde18ec431a73ccfc87977f81be5dfc99c1c94
Status: Downloaded newer image for node:latest
docker.io/library/node:latest

What's next:
View a summary of image vulnerabilities and recommendations → docker scout quickview node:latest
```

### 2. Running a nginx Web Server in a Container

`docker run -d -p 8080:80 --name mynginx nginx (or)`

can be started directly from docker desktop.

docker desktop

Search for images, containers, volumes, extensions... Ctrl+K

Sign in

Containers

Images

Volumes

Builds

Docker Scout

Extensions

Containers [Give feedback](#)

Container CPU usage ⓘ  
0.00% / 800% (8 CPUs available)

Container memory usage ⓘ  
0B / 3.64GB

[Show charts](#)

Search

Only show running containers

<input type="checkbox"/>	Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	<div>mynginx</div> <div>4c4560f1645a</div>	nginx	Running	8080:80	0%	48 seconds ago	<div><div></div><div></div><div></div></div>

Showing 1 item

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.*

🔍

🔖

🔄

📄

🌟

🔒

🕒

🔊

⋮

🔧

### 3. Creating own image for python application

1. Create a Python script (app.py).
2. Create a Dockerfile to define the image.
3. Build the Docker image with docker build.
4. Run the Docker container with docker run.

#### 1.app.py

```
def dog_facts():  
    facts = [  
        "Dogs have a sense of time and miss you when you're gone.",  
        "Dogs' sense of smell is at least 40x better than humans'.",  
        "Dogs can hear frequencies that are out of the range of human hearing.",  
        "A dog's nose print is unique, much like a person's fingerprint.",  
        "Some dogs are incredible swimmers."  
    ]  
    return facts  
  
def print_dog_facts():  
    facts = dog_facts()  
    print("Here are some fun facts about dogs:")  
    for fact in facts:  
        print(f"- {fact}")  
  
if __name__ == "__main__":  
    print_dog_facts()
```

#### 2. Dockerfile

```
# Official Python runtime as a parent image  
FROM python:3.8-slim
```

# Set the working directory in the container

WORKDIR /usr/src/app

# Copy the current directory contents into the container at /usr/src/app

COPY . .

# Run the application

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

### 3.Build image for the application

**docker build -t dog .**

```
D:\Python\dog>docker build -t dog .
[+] Building 3.1s (8/8) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile              0.1s
=> => transferring dockerfile: 313B                             0.0s
=> [internal] load metadata for docker.io/library/python:3.8-slim 0.0s
=> [internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                    0.0s
=> [1/3] FROM docker.io/library/python:3.8-slim                 0.1s
=> [internal] load build context                                2.8s
=> => transferring context: 973B                                  2.4s
=> [2/3] WORKDIR /usr/src/app                                    0.0s
=> [3/3] COPY . .                                               0.1s
=> exporting to image                                           0.1s
=> => exporting layers                                           0.0s
=> => writing image sha256:975366b8d664e8b447642ae090b7365200042b05984fd9d6c09ef3cc844dee97 0.0s
=> => naming to docker.io/library/dog                          0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/jhjrjp6uufs27tetofdobrkbsw

What's next:
  View a summary of image vulnerabilities and recommendations → docker scout quickview
```