**1)C:\Users\VijayPrasanthBurgula>ssh -i "C:\Users\VijayPrasanthBurgula\OneDrive - ValueMomentum, Inc\Documents\docker.pem"** [**ec2-user@13.50.112.255**](mailto:ec2-user@13.50.112.255)

**(Connects to your EC2 Linux instance using your prublic key)**

If it get connection timedout then check connection has security group attached with ssh and ACL with inbound all traffic etc then you can connect

2) **sudo yum update -y (Updates all existing packages on your Amazon Linux machine to their latest versions using yum.)**

**3)** **sudo yum install -y yum-utils(Adds extra yum tools like yum-config-manager, which helps manage package repositories.)**

**4)** **docker –version**

**5)** **sudo dnf install -y docker**

**6)** **sudo systemctl start docker**

**7)** **sudo systemctl enable docker**

**8)** **sudo usermod -aG docker(Gives your current user (ec2-user) permission to run Docker without sudo)**

**9)** **sudo yum update**

**10)** **sudo dnf update**

**11)** **sudo yum install vim -y**

**12)** **sudo yum install git -y**

**13) docker build -t abhishekf5/my-first-docker-image:latest .(** ** Builds a Docker image from the current directory (.) using the Dockerfile found there.**

** Tags it as abhishekf5/my-first-docker-image:latest (you can later push it to Docker Hub).**

**14) docker push vijay086/my-first-docker-image:latest**

**15)** **docker pull vijay086/my-first-docker-image:latest**

**16)** **docker run --rm -v "%cd%":/app -w /app golang:latest go run calculator.go**

**Unable to find image 'golang:latest' locally**

**latest: Pulling from library/golang**

**17) dock**

**er images | head -5(to check how much size calculator is)**

**18) docker build -t simplecalculat-multistage .(to reduce the size of calculator)**

**19) Docker network ls (to see all mnetworks)**

**20)** **When a container is running in detached mode, it means:**

**🟢 The container runs in the background, and you don’t see its logs or output in your terminal directly.**

**🛠️ Syntax:**

**bash**

**CopyEdit**

**docker run -d <image-name>**

**21) 🧪 Example:**

**bash**

**CopyEdit**

**docker run -d --name my-nginx nginx**

**22)apt-get install iputls-ping -y**

**23)Docker ps**

**24)Docker network rm test (to delete)**

**25) To connect in a isolated way we use**

**Docker run -d –name –network secure-network finance nginx:latest**

**25)docker run -d –name host-demo –network=host nginx:latest(it is used to create with host networking as it is completelt different idp address fro other networks(login and log out) because it is using host network**

**26)docker inspect host-demo(ito inspect the specific networking)**

**26)** **docker image ls(to see all images)**

**27)docker run -d –mount source abshik,target=/app nginx:latest**

**27) docker inspect “name of container”**

**BIND MOUNTS AND DOCKER VOLUMES**

Great questions! You're touching on real-world **Docker use cases and problems**. Let’s break each one down, explain the issue, and suggest the correct Docker solution.

**✅ Use Case 1:**

*“There is an application and it contains NGINX, it is in Docker. If the application goes down, all data will be lost.”*

**🔍 Problem:**

* Docker containers are **ephemeral** — when stopped or removed, **data inside them is lost** unless it’s stored externally.

**🛠️ Solution: Use Docker Volumes**

* Volumes allow containers to **store data outside the container** (on the host or external storage).
* Even if the container is removed, the data persists.

**✅ Fix:**

docker run -d -v nginx-data:/usr/share/nginx/html nginx

Here, nginx-data is a named volume. It stores the website data separately.

**✅ Use Case 2:**

*“There is a front end and back end. Backend always creates files. If the backend goes down, how will frontend access the files?”*

**🔍 Problem:**

* The backend dynamically generates files.
* If it's down, the frontend can't retrieve those files.

**🛠️ Solution: Use Shared Volumes or Object Storage**

* Mount a **shared volume** between frontend and backend.
* Or better: Store files in a **central storage like AWS S3**, so both services are loosely coupled.

**✅ Fix (Shared Volume Example):**

docker network create app-net

docker run -d --name backend --network app-net -v shared-data:/app/data backend-image

docker run -d --name frontend --network app-net -v shared-data:/app/data frontend-image

Both containers share shared-data. Files are created by backend and read by frontend.

**✅ Use Case 3:**

*“Your app is trying to read some files from the host OS but doesn’t know how to read them.”*

**🔍 Problem:**

* The container cannot see files from the host OS by default.
* Containers are isolated from the host file system.

**🛠️ Solution: Use Bind Mounts**

* You can **bind a host directory into the container** so it can read/write files from it.

**✅ Fix:**

docker run -d -v /path/on/host:/app/files my-app-image

This allows your app inside the container to access /app/files, which is mapped to the host's directory.

**✅ Summary Table:**

| **Use Case** | **Problem** | **Solution** | **Docker Concept** |
| --- | --- | --- | --- |
| 1 | Data lost on container stop | Use Docker volumes | docker volume |
| 2 | Frontend can't access backend data | Shared volume or S3 | docker volume, object storage |
| 3 | App can't access host files | Bind host directory | bind mount (-v /host:/container) |

Would you like diagrams or real file examples for any of these?

**Commands**

**1)Docker volume ls**

**2)docker volume create Vijay(to create volume)**

**3)docker volume inspect abhishek(we get details of the docker volume)**