Introduction and Setting Docker Context

Docker context refers to the set of configurations and settings that Docker uses to determine where and how to build, run, and manage containers.

Purpose: A Docker context allows you to switch between different Docker environments easily, such as local development, remote servers, or cloud platforms.

Components: A context typically includes:

- The Docker endpoint (e.g., local Docker daemon, remote Docker host)
- Authentication details
- Orchestrator settings (e.g., Kubernetes cluster configurations)

Use cases:

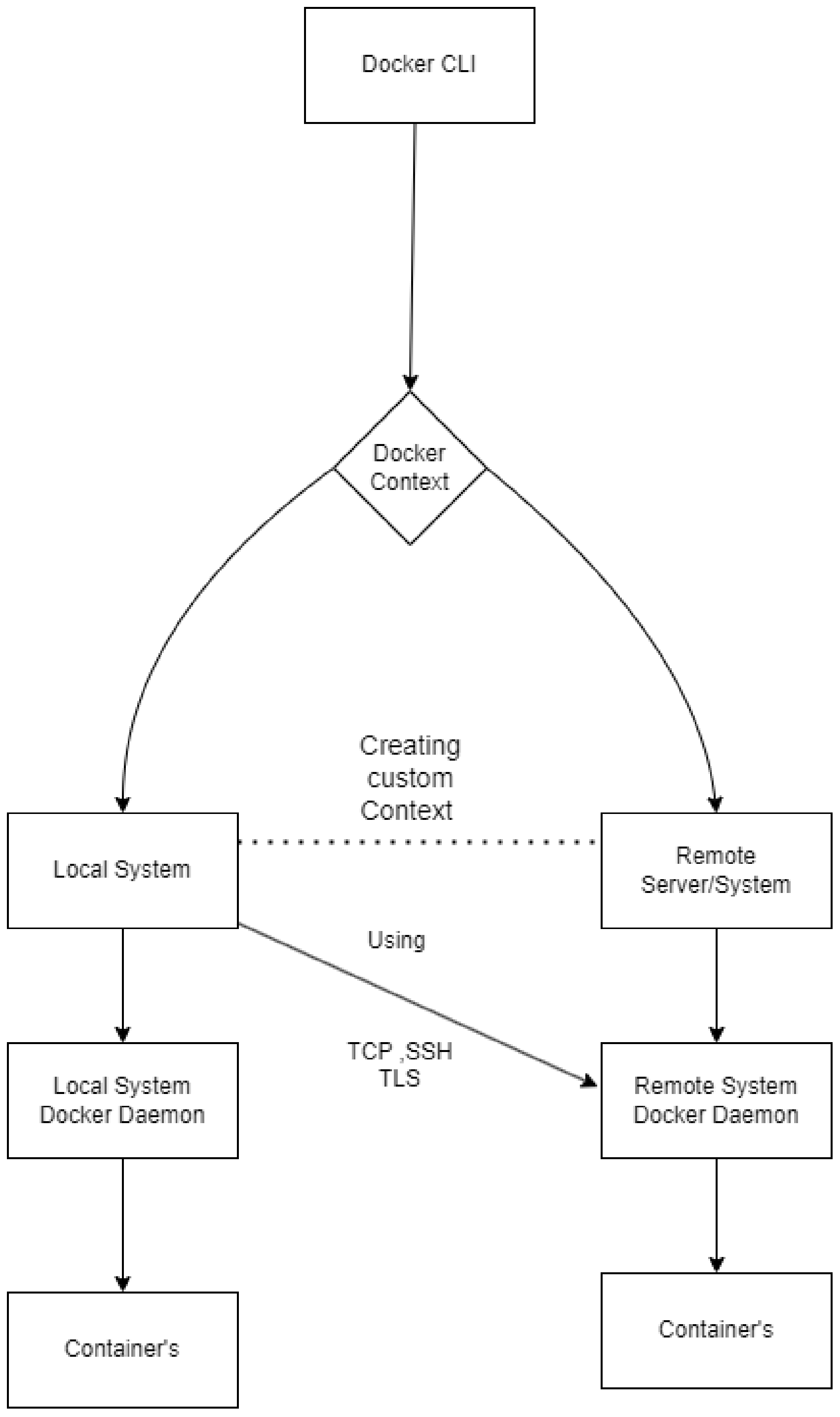
- Managing multiple Docker environments from a single CLI
- Switching between local and remote Docker instances
- Simplifying the management of Docker resources across different machines or cloud providers

Benefits:

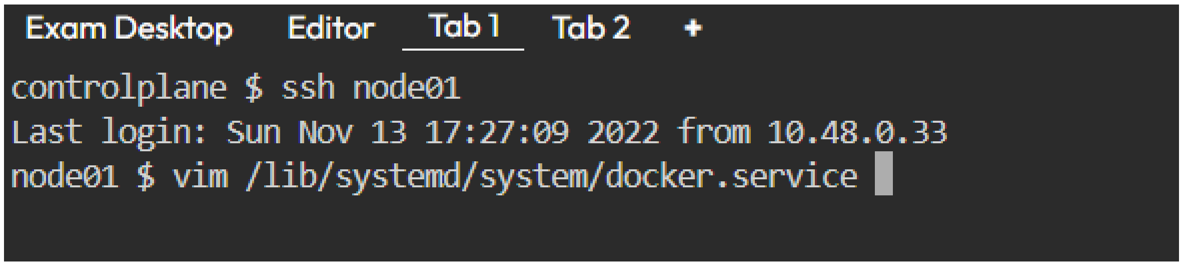
  - Improved workflow efficiency
  - Easier management of multiple Docker environments
  - Enhanced security by isolating different Docker setups

Docker contexts help streamline the process of working with multiple Docker environments, making it easier to manage containers across various setups without constantly reconfiguring your Docker CLI.

Conceptual Diagram For Docker Context

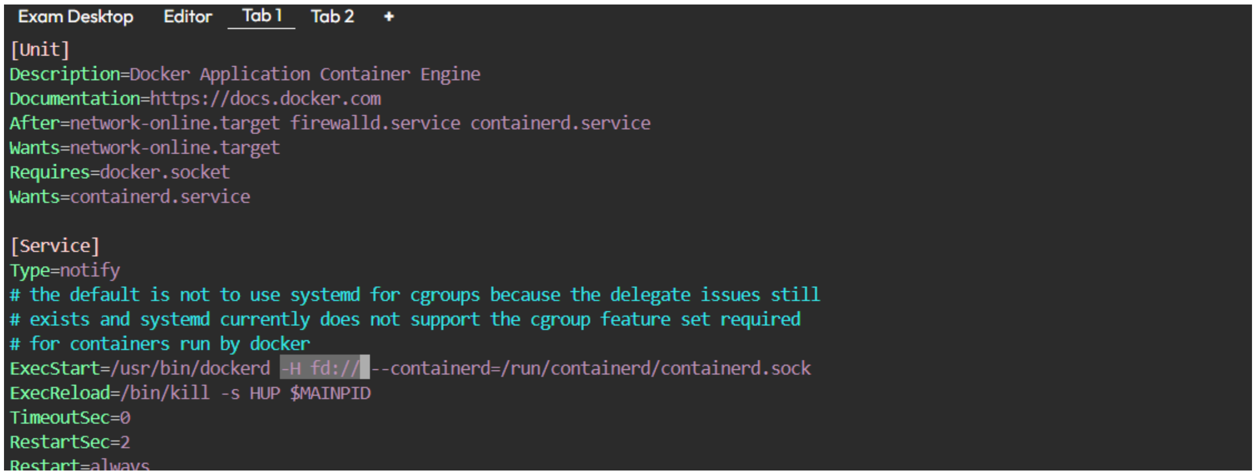


Step 1 – SSH into remote user



```
controlplane $ ssh node01
Last login: Sun Nov 13 17:27:09 2022 from 10.48.0.33
node01 $ vim /lib/systemd/system/docker.service
```

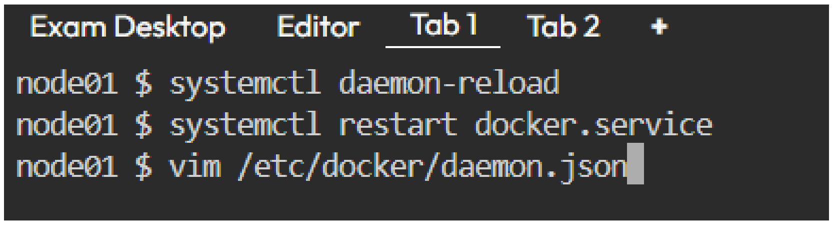
Step 2 – Open the file /lib/system/system/docker.service AND remove the -H fd:// from [Service] ExecStart Variable.



```
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
After=network-online.target firewalld.service containerd.service
Wants=network-online.target
Requires=docker.socket
Wants=containerd.service

[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always
```

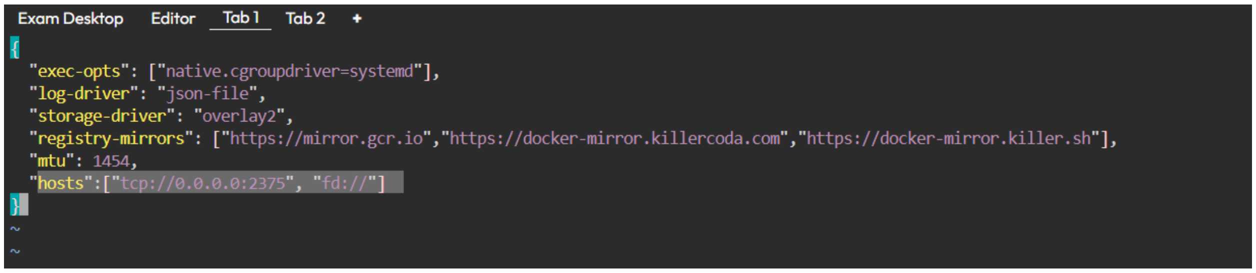
Step 3 – Reload the Docker Service and Daemon



```
node01 $ systemctl daemon-reload
node01 $ systemctl restart docker.service
node01 $ vim /etc/docker/daemon.json
```

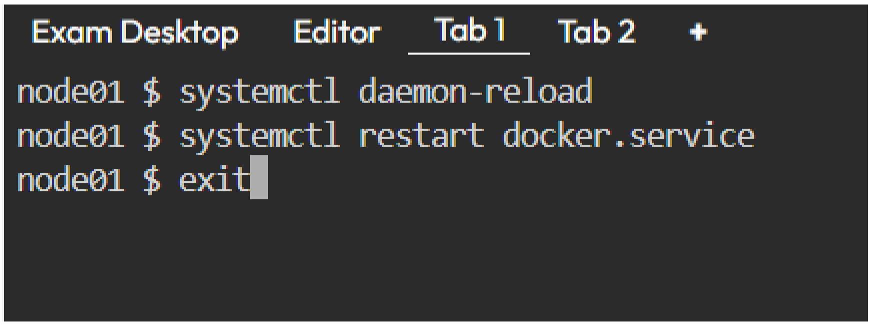
Step 4- Open /etc/docker/daemon.json file and add the following snippet into the file

“hosts”:[“tcp://0.0.0.2375,fd://”]



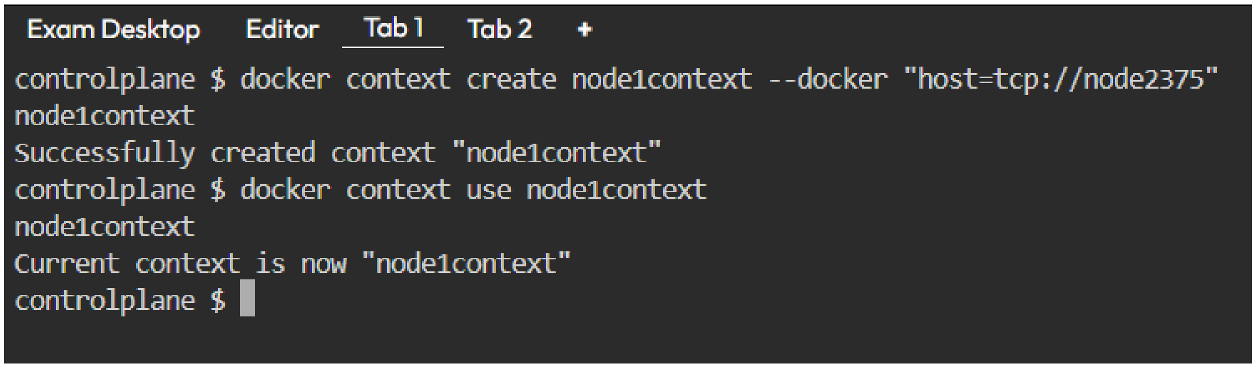
```
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "storage-driver": "overlay2",
  "registry-mirrors": ["https://mirror.gcr.io","https://docker-mirror.killercoda.com","https://docker-mirror.killer.sh"],
  "mtu": 1454,
  "hosts":["tcp://0.0.0.0:2375", "fd://"]
}
~
~
```

Step 5- Restart Docker Service and Daemon and Exit from the remote user



```
node01 $ systemctl daemon-reload
node01 $ systemctl restart docker.service
node01 $ exit
```

Step 6 – Create Docker Context for the Remote user and change the Context from default to newly created context.

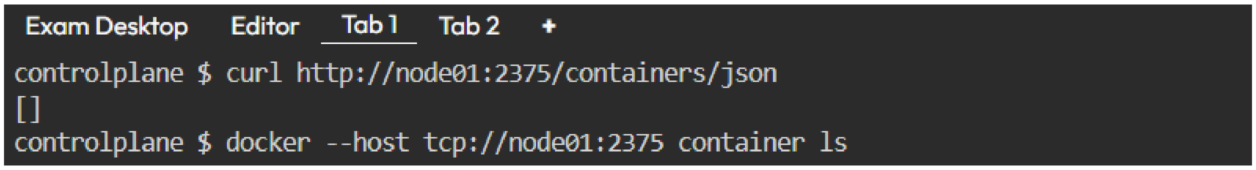


```
controlplane $ docker context create node1context --docker "host=tcp://node2375"
node1context
Successfully created context "node1context"
controlplane $ docker context use node1context
node1context
Current context is now "node1context"
controlplane $ 
```

Step 7 – To check the Successful connect you can run the following command

Curl http://node01:2375/containers/json

If the following command gives [] as output it means you have successfully connected to remote user



```
controlplane $ curl http://node01:2375/containers/json
[]
controlplane $ docker --host tcp://node01:2375 container ls
```

Step 8- Now you access the remote user and control the container's on remote user

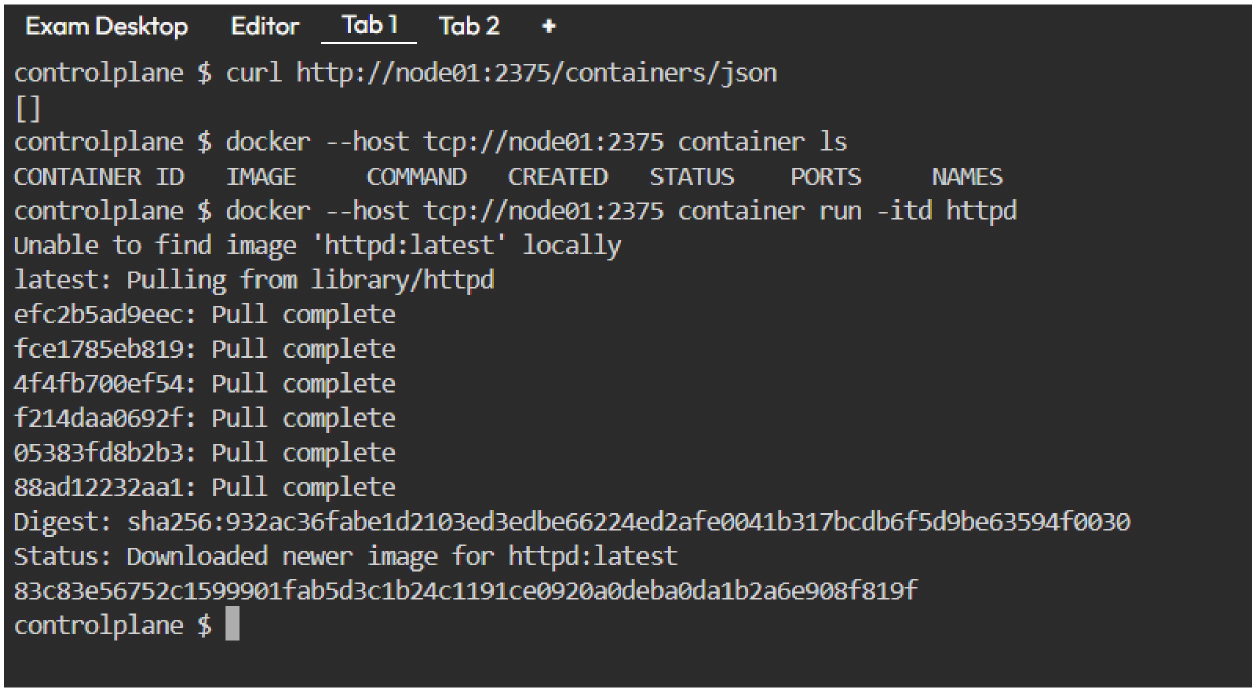
You can perform various tasks such as List, Run, Delete Etc.

Important Note-

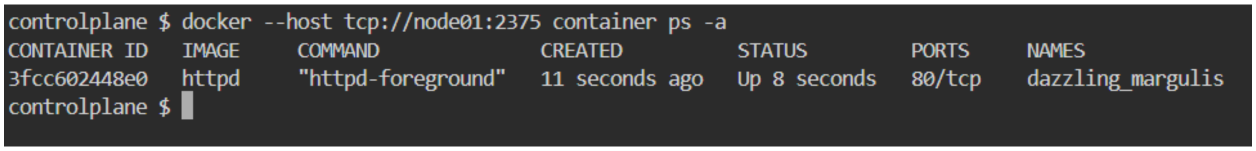
You can use 3 types of Protocol's to access Remote user and Perform Operation

1) TCP
2) TLS
3) SSH

This task's focuses Docker context Using TCP Protocol.



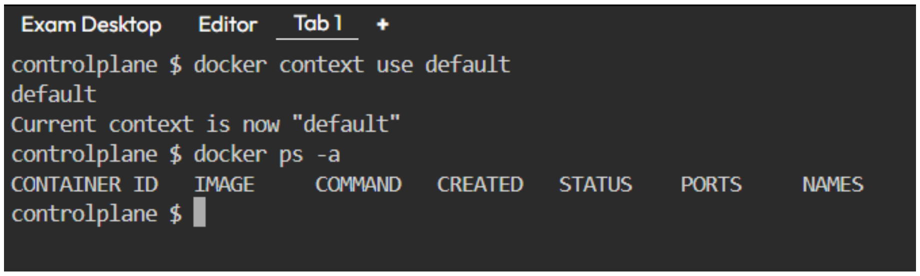
```
controlplane $ curl http://node01:2375/containers/json
[]
controlplane $ docker --host tcp://node01:2375 container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
controlplane $ docker --host tcp://node01:2375 container run -itd httpd
Unable to find image 'httpd:latest' locally
latest: Pulling from library/httpd
efc2b5ad9eec: Pull complete
fce1785eb819: Pull complete
4f4fb700ef54: Pull complete
f214daa0692f: Pull complete
05383fd8b2b3: Pull complete
88ad12232aa1: Pull complete
Digest: sha256:932ac36fabe1d2103ed3edbe66224ed2afe0041b317bcdb6f5d9be63594f0030
Status: Downloaded newer image for httpd:latest
83c83e56752c1599901fab5d3c1b24c1191ce0920a0deba0da1b2a6e908f819f
controlplane $ 
```



```
controlplane $ docker --host tcp://node01:2375 container ps -a
CONTAINER ID   IMAGE     COMMAND              CREATED          STATUS         PORTS     NAMES
3fcc602448e0   httpd     "httpd-foreground"   11 seconds ago   Up 8 seconds   80/tcp    dazzling_margulis
controlplane $ 
```

Step 9 – You can also check if the task is successful or not by changing the context to default and listing the container's

You'll be not be able to see the containers of Remote user, you can see container's of local user only.



```
controlplane $ docker context use default
default
Current context is now "default"
controlplane $ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
controlplane $ 
```

------------------------------- Thank You-------------------------------