Mohit Narang

2103124

Experiment 9

Aim: To study and Implement Containerization using Docker

Theory:

What is Docker and Explain its architecture?

Docker is a popular platform used for building, shipping, and running applications in containers. Containers provide a lightweight, portable, and consistent environment for running applications, enabling developers to package all the necessary components (such as code, runtime, libraries, and dependencies) into a single unit. Docker has become a standard tool in modern software development and deployment due to its efficiency, scalability, and ease of use.

Here's an overview of Docker's architecture:

1. Docker Engine:
    - At the core of Docker is the Docker Engine, which is a client-server application with these main components:
        - Docker Daemon: This is a background service running on the host machine, responsible for managing Docker objects such as images, containers, networks, and volumes. The daemon listens for Docker API requests and manages container lifecycles.
        - Docker CLI (Command Line Interface): Users interact with Docker through the CLI, issuing commands to the Docker daemon to perform various tasks like building, running, and managing containers.
        - REST API: Docker Engine exposes a RESTful API that allows external tools and applications to communicate with the Docker daemon and perform operations programmatically.
2. Docker Images:
    - Docker images are read-only templates used to create containers. An image typically contains the filesystem snapshot of the application, along with all the dependencies and configuration needed to run it.
    - Images are built using Docker files, which are text files containing a series of instructions for assembling the image layers.

- Images are stored in a registry, such as Docker Hub, which serves as a centralized repository for sharing and distributing images.

3. Docker Containers:
    - Containers are lightweight, portable, and executable environments that encapsulate an application and its dependencies.
    - Each container is created from a Docker image and runs as an isolated process on the host machine.
    - Containers are ephemeral, meaning they can be easily started, stopped, deleted, and replaced without affecting the underlying infrastructure or other containers.
    - Docker provides networking and storage options for connecting containers together and persisting data.
4. Docker Registry:
    - A Docker registry is a service responsible for storing and distributing Docker images.
    - Docker Hub is the default public registry provided by Docker, where users can find and share pre-built images.
    - Organizations often set up private registries to store proprietary or sensitive images within their own infrastructure.

Benefits of Containerization

Containerization offers several benefits for software development, deployment, and operations. Here are some of the key advantages:

1. Portability: Containers encapsulate applications and their dependencies, ensuring consistency across different environments, such as development, testing, staging, and production. Developers can build once and run anywhere, eliminating issues related to "it works on my machine" scenarios.
2. Isolation: Containers provide lightweight, isolated environments for running applications. Each container has its own filesystem, processes, networking, and resources, ensuring that applications do not interfere with each other and minimizing the risk of dependency conflicts.
3. Scalability: Containers are highly scalable, allowing organizations to easily deploy and manage applications at scale. Containers can be rapidly instantiated or destroyed to accommodate fluctuations in workload demand, enabling efficient resource utilization and cost savings.

4. Efficiency: Compared to traditional virtual machines (VMs), containers are lightweight and have minimal overhead, as they share the host operating system's kernel. This results in faster startup times, reduced memory footprint, and higher resource efficiency, enabling organizations to run more containers on the same infrastructure.
5. Consistency: Containerization promotes consistency in development, deployment, and operations workflows. By packaging applications and dependencies into standardized containers, organizations can ensure that every instance of an application behaves identically, regardless of the underlying infrastructure.
6. Version Control: Docker images, which serve as templates for containers, can be version-controlled and shared across teams. This enables developers to track changes, roll back to previous versions if necessary, and collaborate more effectively on building and deploying applications.
7. DevOps Enablement: Containers are a foundational technology for DevOps practices, facilitating continuous integration, continuous delivery (CI/CD), and automation. Containers streamline the deployment pipeline, allowing for faster release cycles, increased agility, and improved collaboration between development and operations teams.
8. Resource Isolation: Containers use namespaces and control groups (cgroups) to isolate resources such as CPU, memory, and network bandwidth. This isolation ensures that applications running in containers are protected from resource contention and can operate reliably even in multi-tenant environments.
9. Microservices Architecture: Containerization aligns well with microservices architecture, where applications are decomposed into small, loosely coupled services. Each microservice can be packaged and deployed in its own container, enabling independent development, scaling, and deployment of individual components.
10. Security: While containers provide isolation, they also offer security benefits such as reduced attack surface, improved vulnerability management, and easier application sandboxing. However, it's important to implement additional security measures, such as image scanning, network segmentation, and access controls, to enhance container security further.

Explain following w.r.t Docker.

1. **Container:**
    - In Docker, a container is a lightweight, standalone, and executable package that includes everything needed to run a piece of software, including the code, runtime, libraries, dependencies, and configuration files.

- Containers are isolated environments that run on top of a single host operating system, sharing the kernel with other containers. They provide a consistent runtime environment, ensuring that applications behave predictably across different environments.
- Docker containers are created from Docker images, which serve as read-only templates. Each container is an instance of an image, and multiple containers can be instantiated from the same image.
- Containers offer several benefits, including portability, scalability, efficiency, and isolation, making them well-suited for modern software development and deployment workflows.

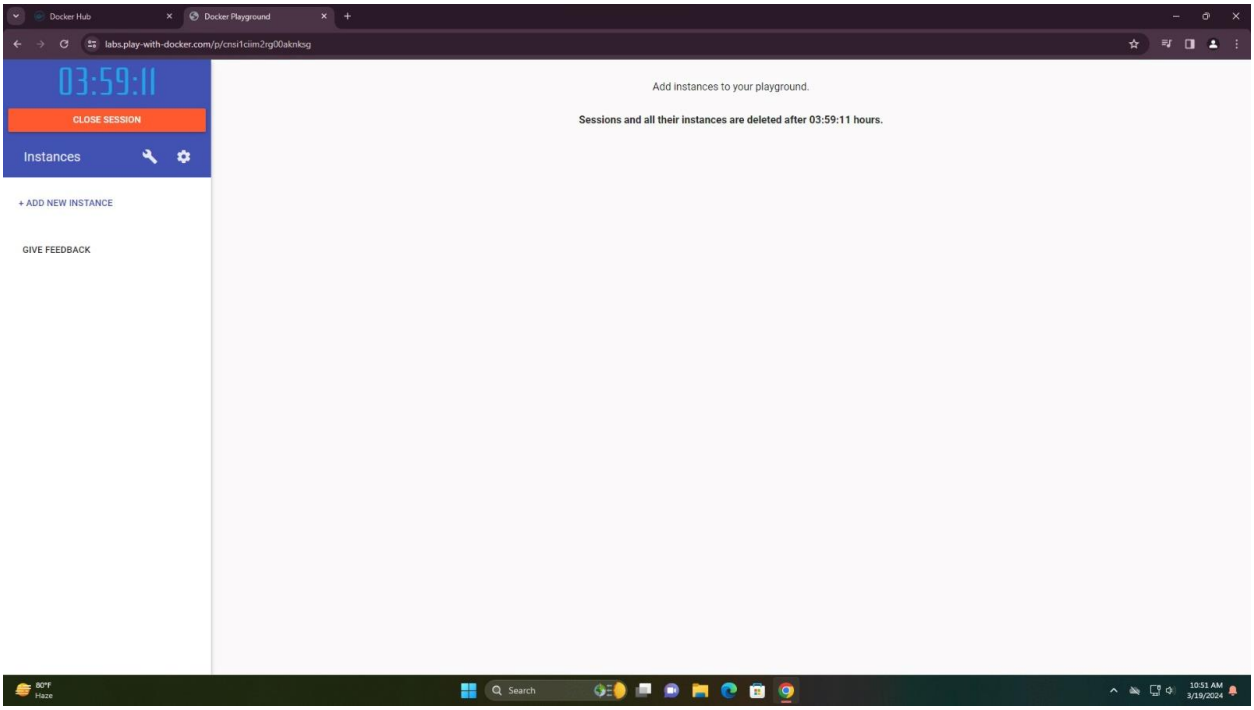
2. **Images:**
    - In Docker, an image is a lightweight, portable, and immutable snapshot of a filesystem that contains all the necessary components to run an application, including the code, runtime, libraries, dependencies, and configuration files.
    - Docker images are built using a declarative syntax called Dockerfile, which specifies a series of instructions for assembling the image layers.
    - Images are stored in a registry, such as Docker Hub, where they can be version-controlled, shared, and distributed across different environments.
    - Images are read-only, meaning they cannot be modified once created. However, they can be used to create writable containers, which allow applications to write data during runtime.
    - Docker images follow a layered architecture, where each instruction in the Dockerfile adds a new layer to the image. This layering mechanism enables efficient image sharing and incremental updates.
3. **Dockerfile:**
    - A Dockerfile is a text file that contains a set of instructions for building a Docker image. It provides a declarative and reproducible way to define the steps needed to assemble the image layers.
    - Dockerfile instructions include commands such as FROM, RUN, COPY, ADD, CMD, ENTRYPOINT, EXPOSE, and WORKDIR, among others, each serving a specific purpose in the image-building process.
    - Developers write Dockerfiles to describe the environment and dependencies required by their applications. Dockerfiles can be version-controlled alongside application code, enabling reproducible builds and continuous integration practices.

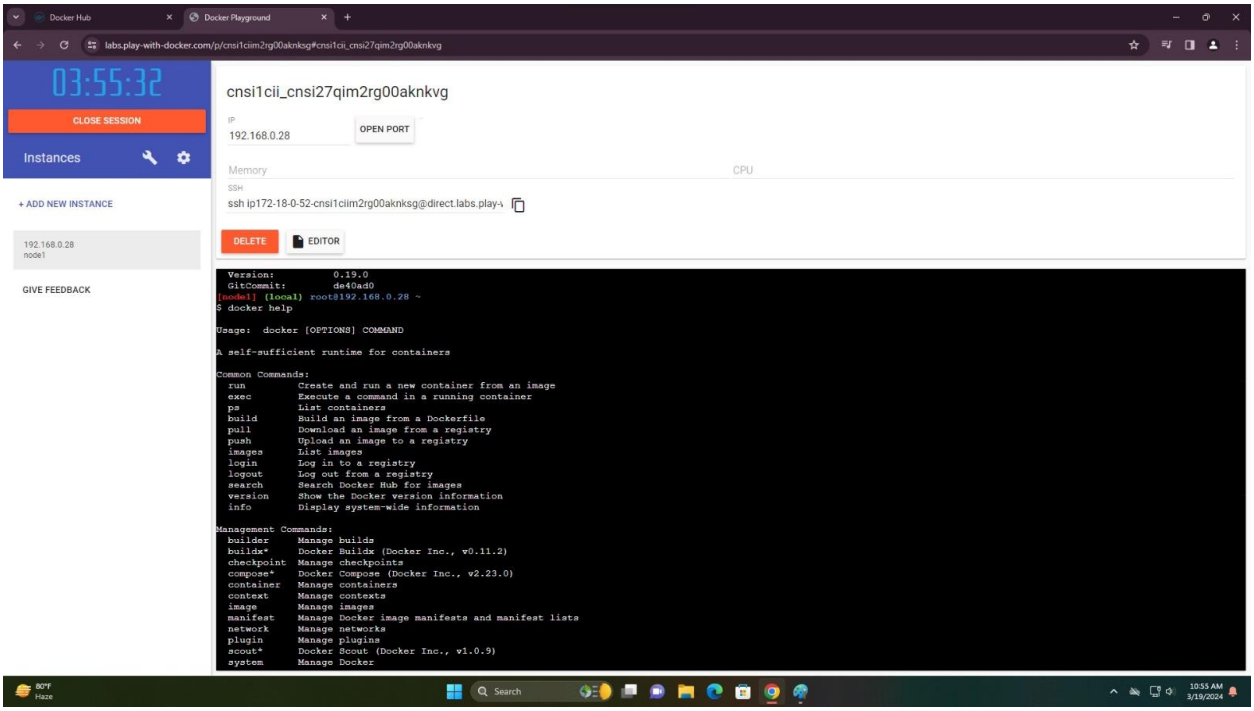
- Once a Dockerfile is defined, developers can use the docker build command to build the corresponding Docker image. The Dockerfile is typically located in the root directory of the application source code.

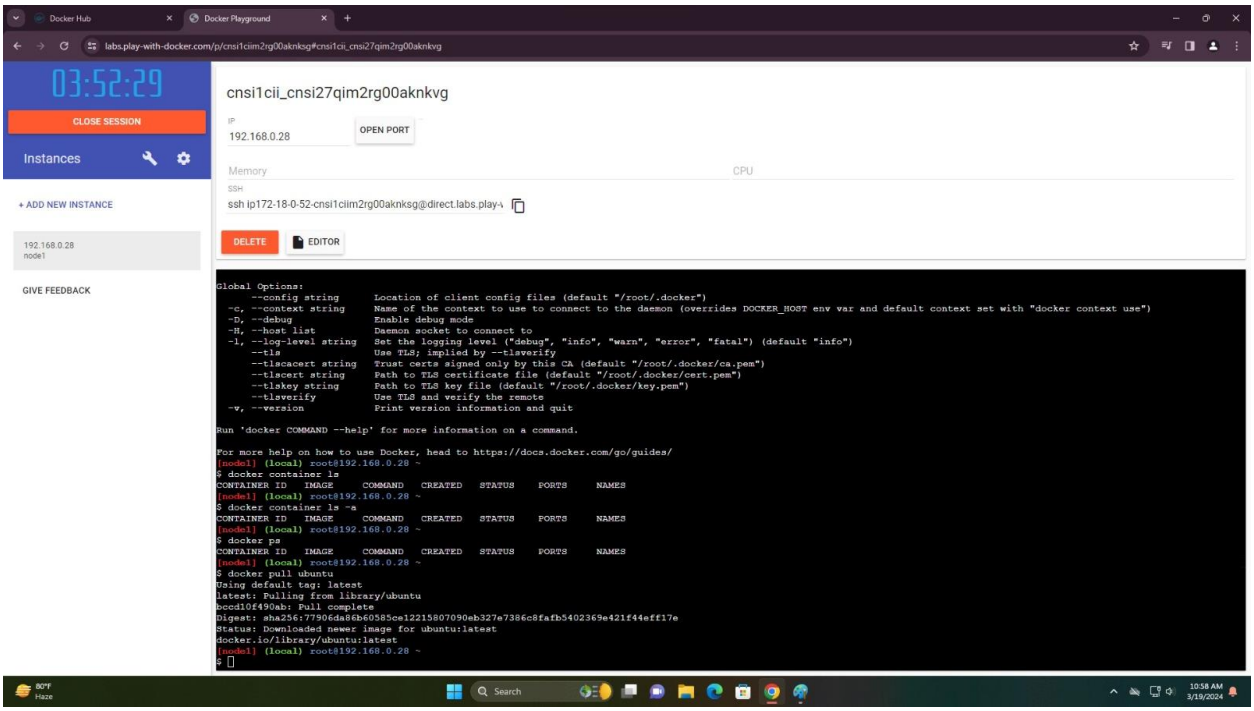
Aspect	Container	Virtual Machine (VM)
Isolation	Uses operating system-level isolation.	Uses hardware-level virtualization.
Resource Usage	Shares host kernel, more lightweight.	Each VM has its own operating system, heavier resource usage.
Startup Time	Starts quickly, in seconds.	Slower startup time, in minutes.
Resource Overhead	Lower overhead, shares host resources.	Higher overhead due to OS duplication.
Scalability	Highly scalable, fast to spin up/down.	Slower to spin up/down due to OS boot.
Density	Higher container density per host.	Lower VM density due to resource needs.
Deployment Model	Usually deployed as microservices.	Traditional monolithic application model.
Isolation Level	Less isolated, potential for more	Higher isolation, less interference.

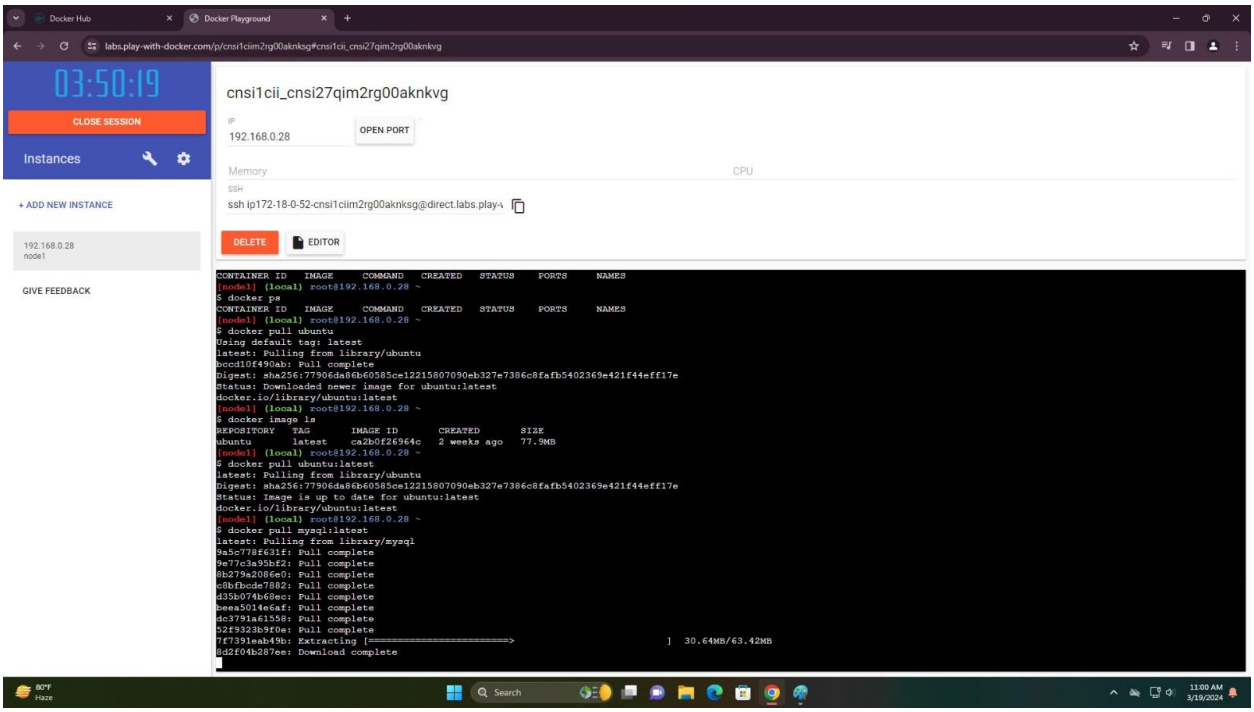
Aspect	Images	Containers
Definition	Read-only template for containers.	Runnable instance of an image.
Lifecycle	Immutable, cannot be modified.	Instantiated from images, can be modified during runtime.
Storage	Stored in registries like Docker Hub.	Instances of images stored locally or in container registries.
Usage	Used as a base for creating containers.	Running instances of applications.
Mutability	Immutable, changes require rebuilding.	Mutable, changes can be made during runtime.
Size	Generally larger due to including all dependencies.	Smaller, as they are lightweight runtime environments.
Versioning	Version-controlled, tagged with versions.	No inherent versioning, instances may be tagged.
Deployment	Built before containers are deployed.	Actively used during application runtime.
Reproducibility	Enables reproducible builds.	Instances may differ due to runtime changes.
Example Commands	`docker build`, `docker push`, `docker pull`.	`docker run`, `docker stop`, `docker restart`.

Steps to be followed:

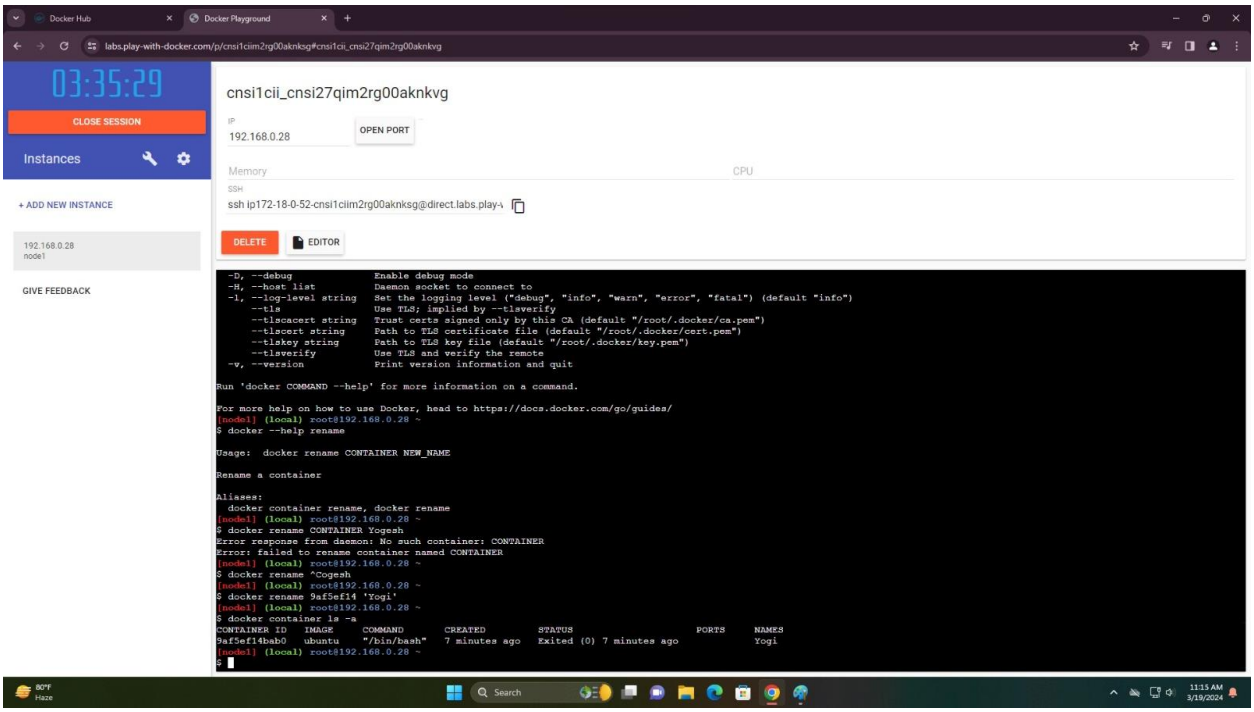





Docker Hub
Docker Playground
labs.play-with-docker.com/p/cnsi1ciim2rg00aknksg#cnsi1cii_cnsi27qim2rg00aknkvg
03:52:29
CLOSE SESSION
Instances
+ ADD NEW INSTANCE
192.168.0.28
node1
GIVE FEEDBACK
cnsi1cii_cnsi27qim2rg00aknkvg
IP
192.168.0.28
OPEN PORT
Memory
CPU
SSH
ssh ip172-18-0-52-cnsi1ciim2rg00aknksg@direct.labs.play-
DELETE
EDITOR
Global Options:
--config string Location of client config files (default "/root/.docker")
-c, --context string Name of the context to use to connect to the daemon (overrides DOCKER_HOST env var and default context set with "docker context use")
-D, --debug Enable debug mode
-H, --host list Daemon socket to connect to
-l, --log-level string Set the logging level ("debug", "info", "warn", "error", "fatal") (default "info")
--tls Use TLS; implied by --tlsverify
--tlscacert string Trust certs signed only by this CA (default "/root/.docker/ca.pem")
--tlscert string Path to TLS certificate file (default "/root/.docker/cert.pem")
--tlskey string Path to TLS key file (default "/root/.docker/key.pem")
--tlsverify Use TLS and verify the remote
-v, --version Print version information and quit
Run 'docker COMMAND --help' for more information on a command.
For more help on how to use Docker, head to https://docs.docker.com/go/guides/
[node1] (local) root@192.168.0.28 ~
$ docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
[node1] (local) root@192.168.0.28 ~
$ docker container ls -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
[node1] (local) root@192.168.0.28 ~
$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
[node1] (local) root@192.168.0.28 ~
$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
bccd10f490ab: Pull complete
Digest: sha256:77906da86b60585ce12215807090eb327e7386c8fafb5402369e421f44eff17e
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
[node1] (local) root@192.168.0.28 ~
$
80°F
Haze
Search
10:58 AM
3/19/2024


Docker Hub
Docker Playground
labs.play-with-docker.com/p/cnsi1ciim2rg00aknksg#cnsi1cii_cnsi27qim2rg00aknkvg
03:50:19
CLOSE SESSION
Instances
+ ADD NEW INSTANCE
192.168.0.28
node1
GIVE FEEDBACK
cnsi1cii_cnsi27qim2rg00aknkvg
IP
192.168.0.28
OPEN PORT
Memory
CPU
SSH
ssh ip172-18-0-52-cnsi1ciim2rg00aknksg@direct.labs.play-
DELETE
EDITOR
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
[node1] (local) root@192.168.0.28 ~
$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
[node1] (local) root@192.168.0.28 ~
$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
bccd10f490ab: Pull complete
Digest: sha256:77906da86b60585ce12215807090eb327e7386c8fafb5402369e421f44eff17e
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
[node1] (local) root@192.168.0.28 ~
$ docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
ubuntu latest ca2b0f26964c 2 weeks ago 77.9MB
[node1] (local) root@192.168.0.28 ~
$ docker pull ubuntu:latest
latest: Pulling from library/ubuntu
Digest: sha256:77906da86b60585ce12215807090eb327e7386c8fafb5402369e421f44eff17e
Status: Image is up to date for ubuntu:latest
docker.io/library/ubuntu:latest
[node1] (local) root@192.168.0.28 ~
$ docker pull mysql:latest
latest: Pulling from library/mysql
9a5c778f631f: Pull complete
9e77c3a95bf2: Pull complete
8b279a2086e0: Pull complete
c8bfbcde7882: Pull complete
d35b074b68ec: Pull complete
beaa5014e4af: Pull complete
dc3791a61558: Pull complete
52f9323b9f0e: Pull complete
7f7391eab49b: Extracting [==========>] 30.64MB/63.42MB
8d2f04b287ee: Download complete
80°F
Haze
Search
11:00 AM
3/19/2024

```
$ docker pull mysql:5.6
5.6: Pulling from library/mysql
35b2232c987e: Pull complete
fc55c00e48f2: Pull complete
0030405130e3: Pull complete
e1fef7f6a8d1: Pull complete
1c76272398bb: Pull complete
f57e698171b6: Pull complete
f5b825b269c0: Pull complete
dcb0af686073: Pull complete
27bbfeb886d1: Pull complete
6f70cc868145: Pull complete
1f6637f4600d: Pull complete
Digest: sha256:20575ecebe6216036d25dab5903808211f1e9ba63dc7825ac20cb975e34cfcae
Status: Downloaded newer image for mysql:5.6
docker.io/library/mysql:5.6
[node1] (local) root@192.168.0.28 ~
$ docker search --filter=stars=100 ubuntu
NAME                DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
ubuntu              Ubuntu is a Debian-based Linux operating sys…   16955     [OK]
websphere-liberty   WebSphere Liberty multi-architecture images …   298       [OK]
neurodebian         NeuroDebian provides neuroscience research s…   106       [OK]
ubuntu-upstart      DEPRECATED, as is Upstart (find other proces…   115       [OK]
ubuntu/nginx        Nginx, a high-performance reverse proxy & we…   112
[node1] (local) root@192.168.0.28 ~
$
```



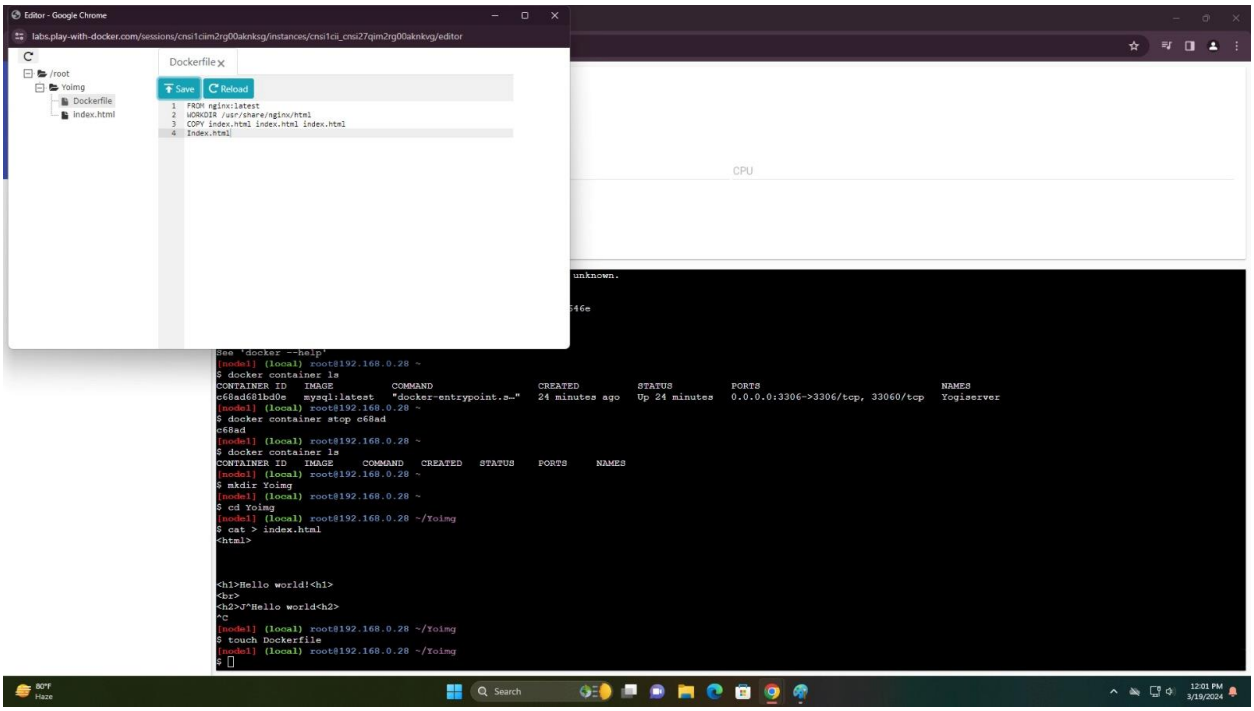
```
$ docker container ls -a
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS                     PORTS     NAMES
9af5ef14bab0   ubuntu    "/bin/bash"   7 minutes ago   Exited (0) 7 minutes ago             Yogi
[node1] (local) root@192.168.0.28 ~
$ docker run -it ubuntu /bin/bash
root@f94261579bc9:/# ls -1
bin
boot
dev
etc
home
lib
lib32
lib64
libx32
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
root@f94261579bc9:/#
```

```
$ docker run --publish 8080:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
8a1e25ce7c4f: Pull complete
e78b137be355: Pull complete
39fc875bd2b2: Pull complete
035788421403: Pull complete
87c3fb37cbf2: Pull complete
c5cdd1ce752d: Pull complete
33952c599532: Pull complete
Digest: sha256:6db391d1c0cfb30588ba0bf72ea999404f2764febf0f1f196acd5867ac7efa7e
Status: Downloaded newer image for nginx:latest
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/03/19 05:52:30 [notice] 1#1: using the "epoll" event method
2024/03/19 05:52:30 [notice] 1#1: nginx/1.25.4
2024/03/19 05:52:30 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/03/19 05:52:30 [notice] 1#1: OS: Linux 4.4.0-210-generic
2024/03/19 05:52:30 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/03/19 05:52:30 [notice] 1#1: start worker processes
2024/03/19 05:52:30 [notice] 1#1: start worker process 29
2024/03/19 05:52:30 [notice] 1#1: start worker process 30
2024/03/19 05:52:30 [notice] 1#1: start worker process 31
2024/03/19 05:52:30 [notice] 1#1: start worker process 32
2024/03/19 05:52:30 [notice] 1#1: start worker process 33
2024/03/19 05:52:30 [notice] 1#1: start worker process 34
2024/03/19 05:52:30 [notice] 1#1: start worker process 35
2024/03/19 05:52:30 [notice] 1#1: start worker process 36
```

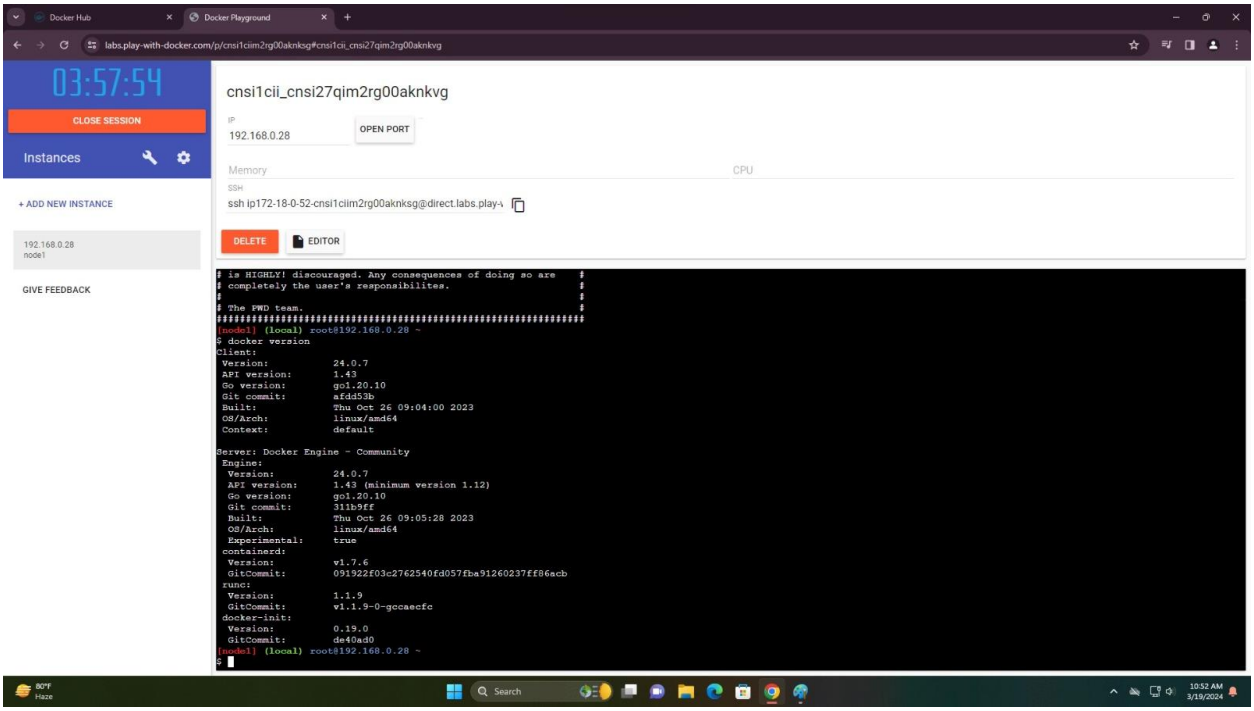
```
[node1] (local) root@192.168.0.28 ~
$ docker run --name Yogiserver -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=123 mysql:latest
c68ad681bd0eb86226b379cebe492cfd0104ee611340c71e44872ab1ecc3c3a2
[node1] (local) root@192.168.0.28 ~
$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                               NAMES
c68ad681bd0e   mysql:latest   "docker-entrypoint.s…"   29 seconds ago   Up 28 seconds   0.0.0.0:3306->3306/tcp, 33060/tcp   Yogiserver
[node1] (local) root@192.168.0.28 ~
$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS                      PORTS                               NAMES
c68ad681bd0e   mysql:latest   "docker-entrypoint.s…"   38 seconds ago   Up 37 seconds               0.0.0.0:3306->3306/tcp, 33060/tcp   Yogiserver
0e30574bfea9   nginx          "/docker-entrypoint.…"   5 minutes ago    Exited (0) 2 minutes ago                                        angry_neumann
f94261579bc9   ubuntu         "/bin/bash"              11 minutes ago   Exited (0) 6 minutes ago                                        youthful_williamson
9af5ef14bab0   ubuntu         "/bin/bash"              22 minutes ago   Exited (0) 22 minutes ago                                       Yogi
[node1] (local) root@192.168.0.28 ~
$
```

```
$ ps -a
PID   USER     TIME  COMMAND
    1 root      0:00 /bin/sh -c cat /etc/hosts >/etc/hosts.bak &&     sed 's/^::1.*//' /etc/hosts.bak > /etc/hosts &&     sed -i "s/\PWD_IP_ADDRESS/$PWD_IP_ADDRESS/" /etc/doc
   16 root      1:03 dockerd
   17 root      0:00 script -q -c /bin/bash -l /dev/null
   19 root      0:00 /bin/bash -l
   32 root      0:00 sshd: /usr/sbin/sshd -o PermitRootLogin=yes -o PrintMotd=no [listener] 0 of 10-100 startups
   47 root      0:24 containerd --config /var/run/docker/containerd/containerd.toml --log-level debug
  935 root      0:00 /usr/local/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 3306 -container-ip 172.17.0.2 -container-port 3306
  949 root      0:01 /usr/local/bin/containerd-shim-runc-v2 -namespace moby -id c68ad681bd0eb86226b379cebe492cfd0104ee611340c71e44872ab1ecc3c3a2 -address /var/run/docker/cont
  968 999       0:12 mysqld
 1512 root      0:00 ps -a
[node1] (local) root@192.168.0.28 ~
$ docker run -itd ubuntu sleep20
3e769994db29678f16b0f9db54fa9ed82a7dbb2f7d3e964a82089fe0dbd641b4
docker: Error response from daemon: failed to create task for container: failed to create shim task: OCI runtime create failed: runc create failed: unable to start container p
rocess: exec: "sleep20": executable file not found in $PATH: unknown.
[node1] (local) root@192.168.0.28 ~
$ docker run -itd ubuntu sleep 20
8e04e2bd7b62c88df8d412238f256bf1756cfd898ecc1ae1005d45997879646e
[node1] (local) root@192.168.0.28 ~
$
```

```
$ docker ls
docker: 'ls' is not a docker command.
See 'docker --help'
[node1] (local) root@192.168.0.28 ~
$ docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                               NAMES
c68ad681bd0e   mysql:latest   "docker-entrypoint.s…"   24 minutes ago   Up 24 minutes   0.0.0.0:3306->3306/tcp, 33060/tcp   Yogiserver
[node1] (local) root@192.168.0.28 ~
$ docker container stop c68ad
c68ad
[node1] (local) root@192.168.0.28 ~
$ docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[node1] (local) root@192.168.0.28 ~
$
```


Editor - Google Chrome
labs.play-with-docker.com/sessions/cnsi1ciim2rg00aknksg/instances/cnsi1cii_cnsi27qim2rg00aknkvg/editor
Dockerfile
/root
Yoimg
Dockerfile
index.html
Save
Reload
1 FROM nginx:latest
2 WORKDIR /usr/share/nginx/html
3 COPY index.html index.html index.html
4 index.html
CPU
unknown.
See 'docker --help'
[node1] (local) root@192.168.0.28 ~
$ docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
c68ad681bd0e mysql:latest "docker-entrypoint.s…" 24 minutes ago Up 24 minutes 0.0.0.0:3306->3306/tcp, 33060/tcp Yogiserver
[node1] (local) root@192.168.0.28 ~
$ docker container stop c68ad
c68ad
[node1] (local) root@192.168.0.28 ~
$ docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
[node1] (local) root@192.168.0.28 ~
$ mkdir Yoimg
[node1] (local) root@192.168.0.28 ~
$ cd Yoimg
[node1] (local) root@192.168.0.28 ~/Yoimg
$ cat > index.html
<html>
<h1>Hello world!<h1>

<h2>J^Hello world<h2>
^C
[node1] (local) root@192.168.0.28 ~/Yoimg
$ touch Dockerfile
[node1] (local) root@192.168.0.28 ~/Yoimg
$
80°F Haze
Search
12:01 PM 3/19/2024


Docker Hub
Docker Playground
labs.play-with-docker.com/p/cnsi1ciim2rg00aknksg#cnsi1cii_cnsi27qim2rg00aknkvg
03:57:54
CLOSE SESSION
Instances
+ ADD NEW INSTANCE
192.168.0.28
node1
GIVE FEEDBACK
cnsi1cii_cnsi27qim2rg00aknkvg
IP
192.168.0.28
OPEN PORT
Memory
CPU
SSH
ssh ip172-18-0-52-cnsi1ciim2rg00aknksg@direct.labs.play-
DELETE
EDITOR
is HIGHLY! discouraged. Any consequences of doing so are
completely the user's responsibilites.
# #
The PWD team.
[node1] (local) root@192.168.0.28 ~
$ docker version
Client:
Version: 24.0.7
API version: 1.43
Go version: go1.20.10
Git commit: afdd53b
Built: Thu Oct 26 09:04:00 2023
OS/Arch: linux/amd64
Context: default
Server: Docker Engine - Community
Engine:
Version: 24.0.7
API version: 1.43 (minimum version 1.12)
Go version: go1.20.10
Git commit: 311b9ff
Built: Thu Oct 26 09:05:28 2023
OS/Arch: linux/amd64
Experimental: true
containerd:
Version: v1.7.6
GitCommit: 091922f03c2762540fd057fba91260237ff86acb
runc:
Version: 1.1.9
GitCommit: v1.1.9-0-gccaecfc
docker-init:
Version: 0.19.0
GitCommit: de40ad0
[node1] (local) root@192.168.0.28 ~
$
80°F Haze
Search
10:52 AM 3/19/2024