

Cloud Computing Exercise – 2

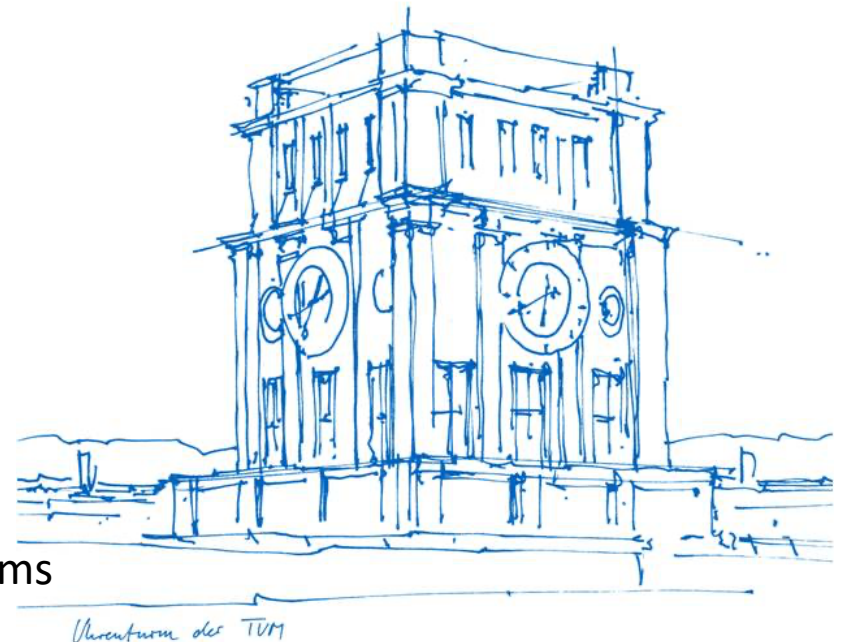
Application Deployment Using Docker

Anshul Jindal (M.Sc. Informatics)

anshul.jindal@tum.de

Chair of Computer Architecture and Parallel Systems

Technical University of Munich (TUM), Germany



Introduction to Docker and Docker Hub

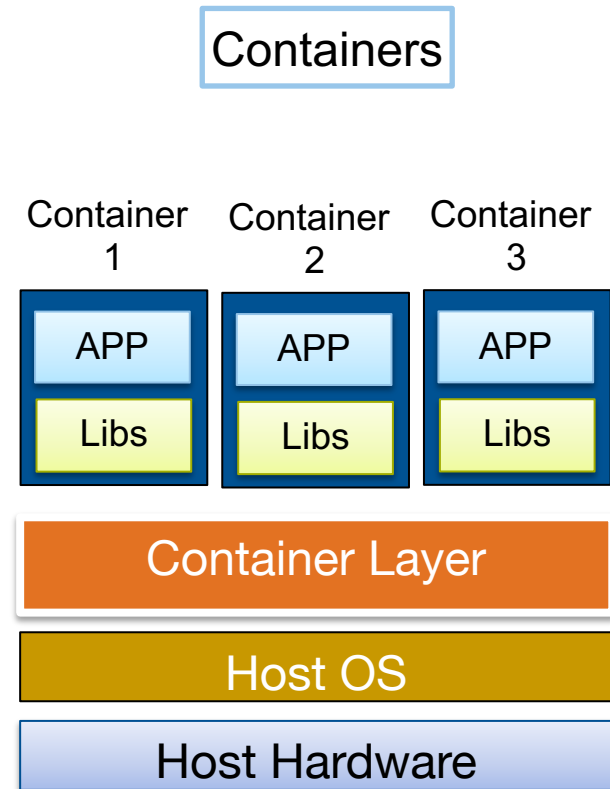
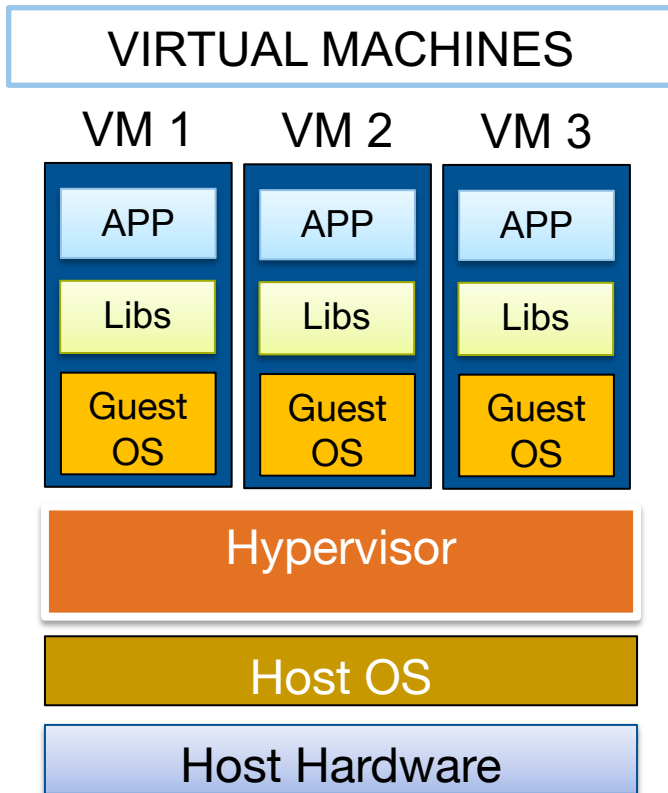
Problems with the deployment method of 1st Exercise

- **OS Dependent:** Different deployment procedure and requirements for different OS.
- **Not-Scalable:** Run on more Laptops or VMs ? Not an Ideal Solution
- **Not-Portable:** Running the same procedure from starting again on the new machine ? Time wastage.
- And many more....

Containerization (container-based virtualization)

What?

- is an OS-level virtualization method for deploying and running distributed applications without launching an entire VM for each application.
- share the same OS kernel as the host.

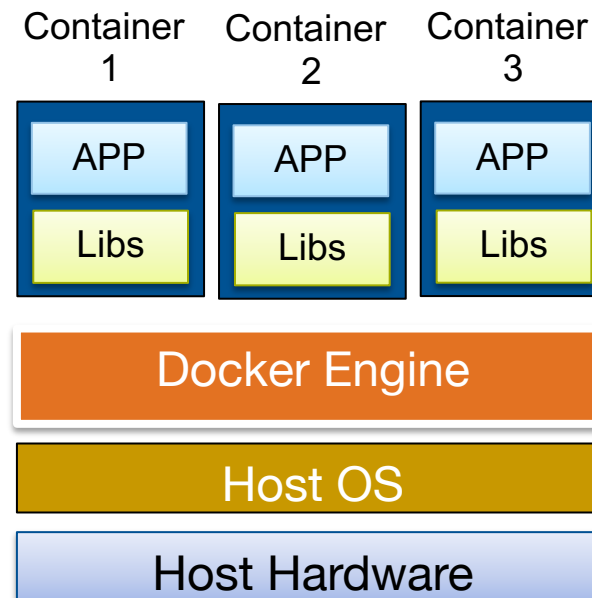


Docker

Docker provides a unified access to

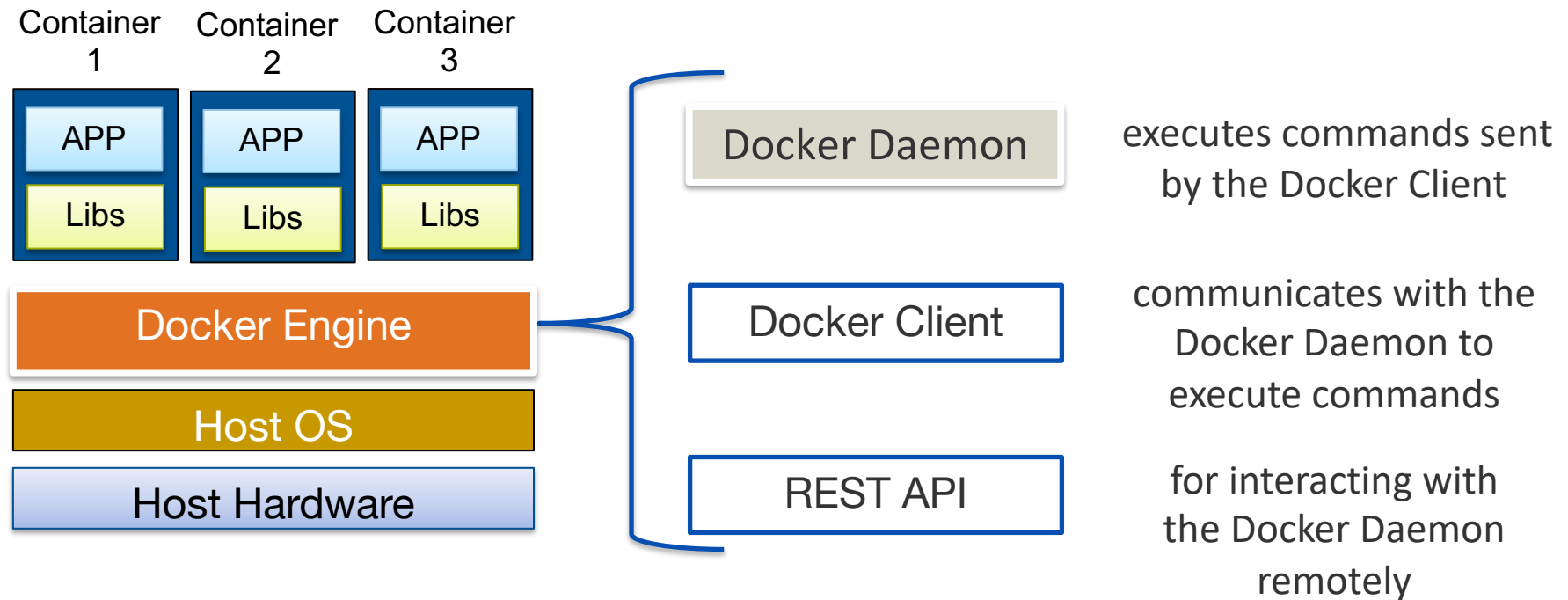
- Linux container technology (cgroups, namespaces)
- Various container implementations (lxc, libvirt, libcontainer, etc.)

'libcontainer' is Docker's implementation of container technology

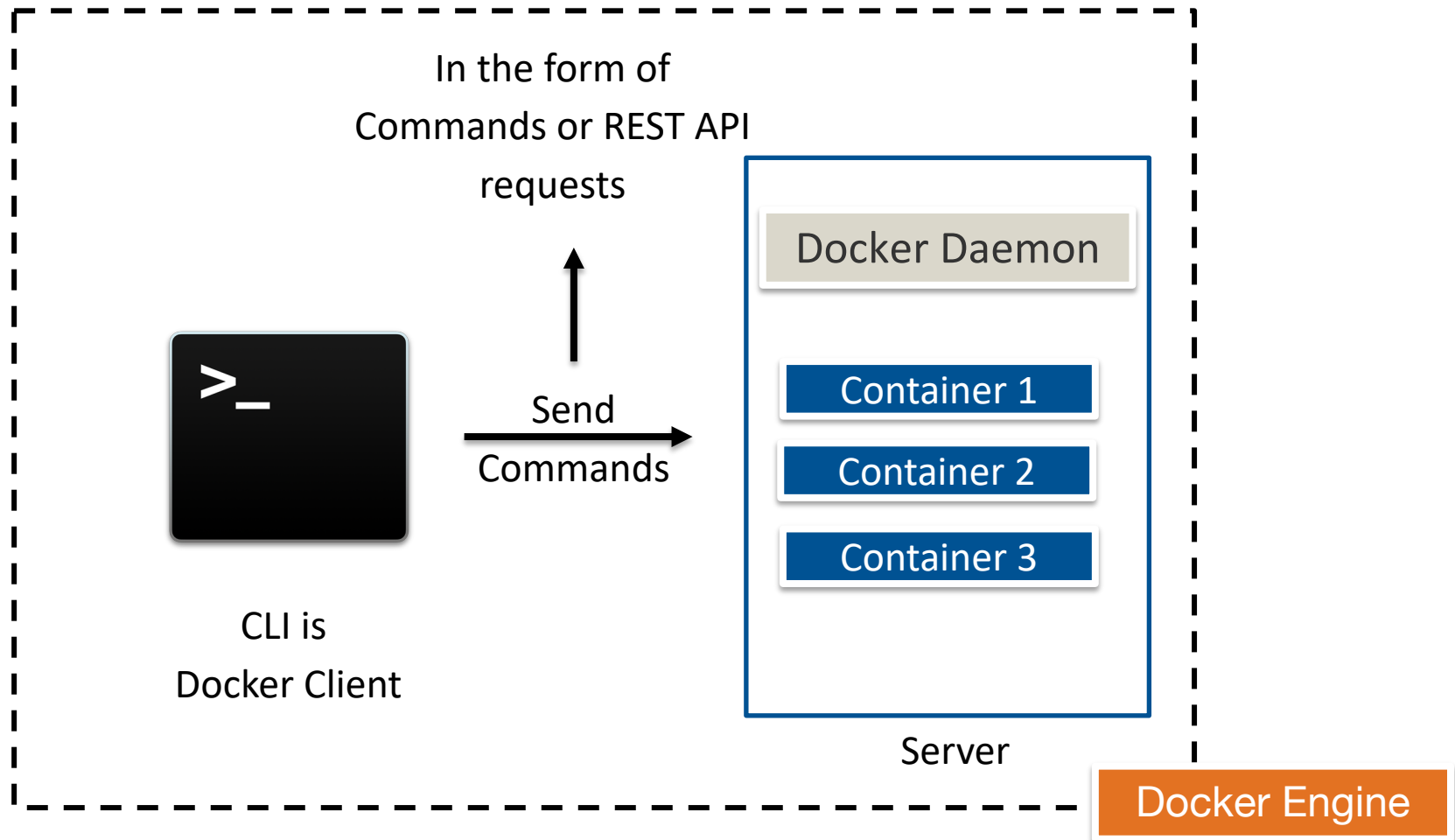


Docker Continued...

The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers.



Docker client-server Architecture



Docker client and daemon can be present on the same or different host machines

Advantages of containers



- packs your application in a container with all of your application's bins\libs and dependencies.
- makes it fully isolated from external environments regardless of where it is running.
- we can ship that container to any where i.e. to any other OS, to Docker Registry (such as Docker Hub) or to the cloud.

OS Independent

Scalable

Portable

Dockerfile

```
FROM node:alpine
```

```
RUN mkdir -p /usr/src/server
```

```
WORKDIR /usr/src/server
```

```
COPY package.json /usr/src/server/
```

```
RUN npm install
```

```
COPY . /usr/src/server
```

```
EXPOSE 3000
```

```
CMD [ "node", "server.js" ]
```

Use a Docker base Image

Image based on **Alpine Linux**. Node is the **repository name (<your-username>/my-first-repo)** in dockerhub and **alpine** is the version

Create Application Directory

Set the working directory of the container for all the RUN commands

Copy the **package.json** file which contain all the dependencies required for application

This command will install all the dependencies listed in **package.json**

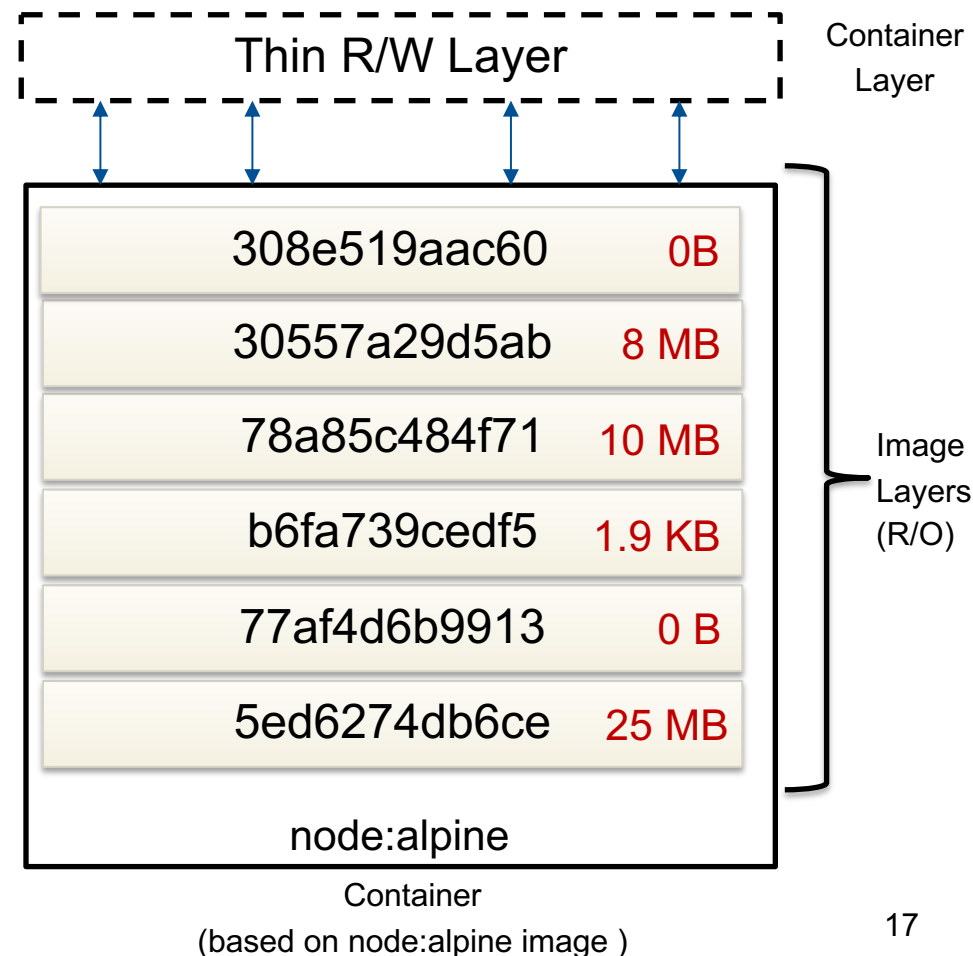
Copy all other files from local machine to container

Expose container port to the host machine

A start command to run the application

Images and layers

- A Docker image is built up from a series of layers.
- Each layer represents an instruction in the image's Dockerfile.
- A new writable layer on top of the underlying layers often called as the "container layer" is added when a new container is created.



CMD ["node", "server.js"]

COPY . /usr/src/server

RUN npm install

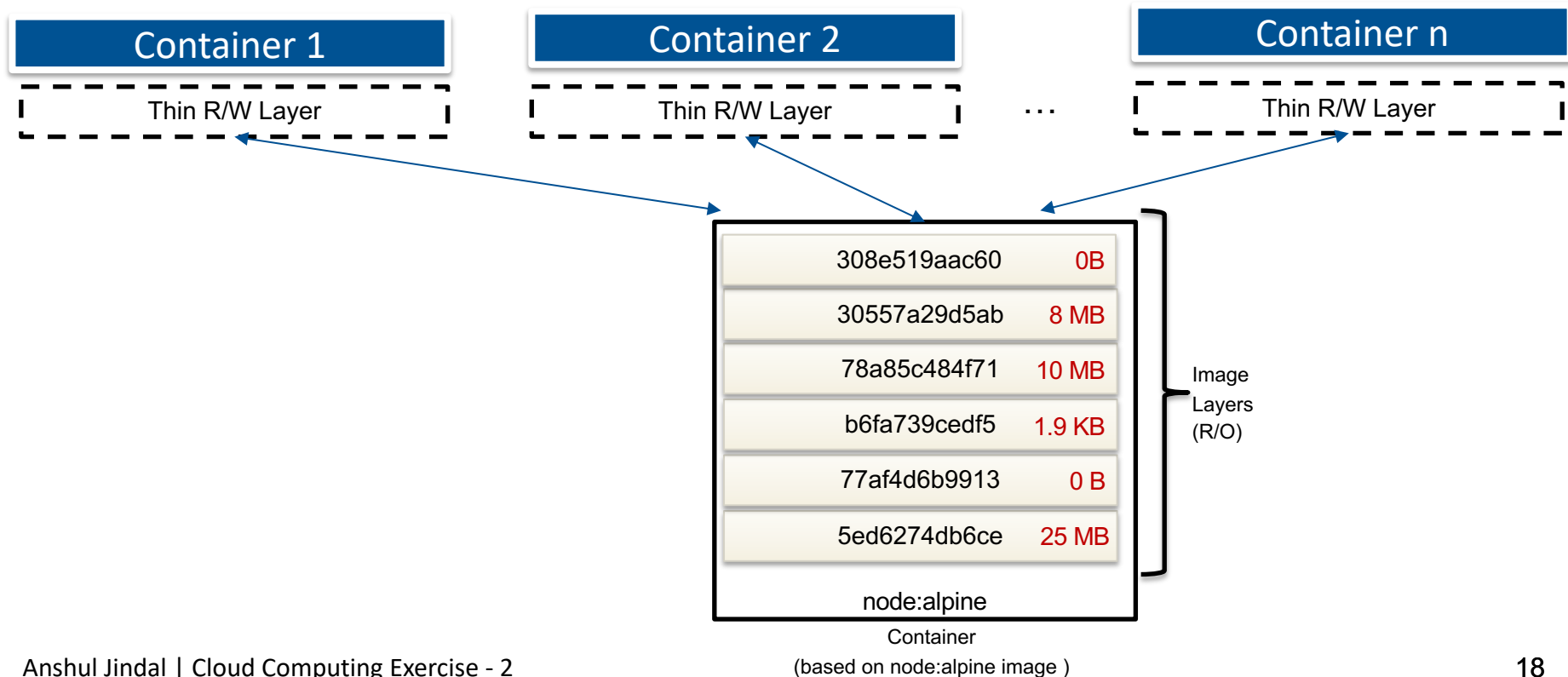
COPY package.json /usr/src/server/

RUN mkdir -p /usr/src/server

FROM node:alpine

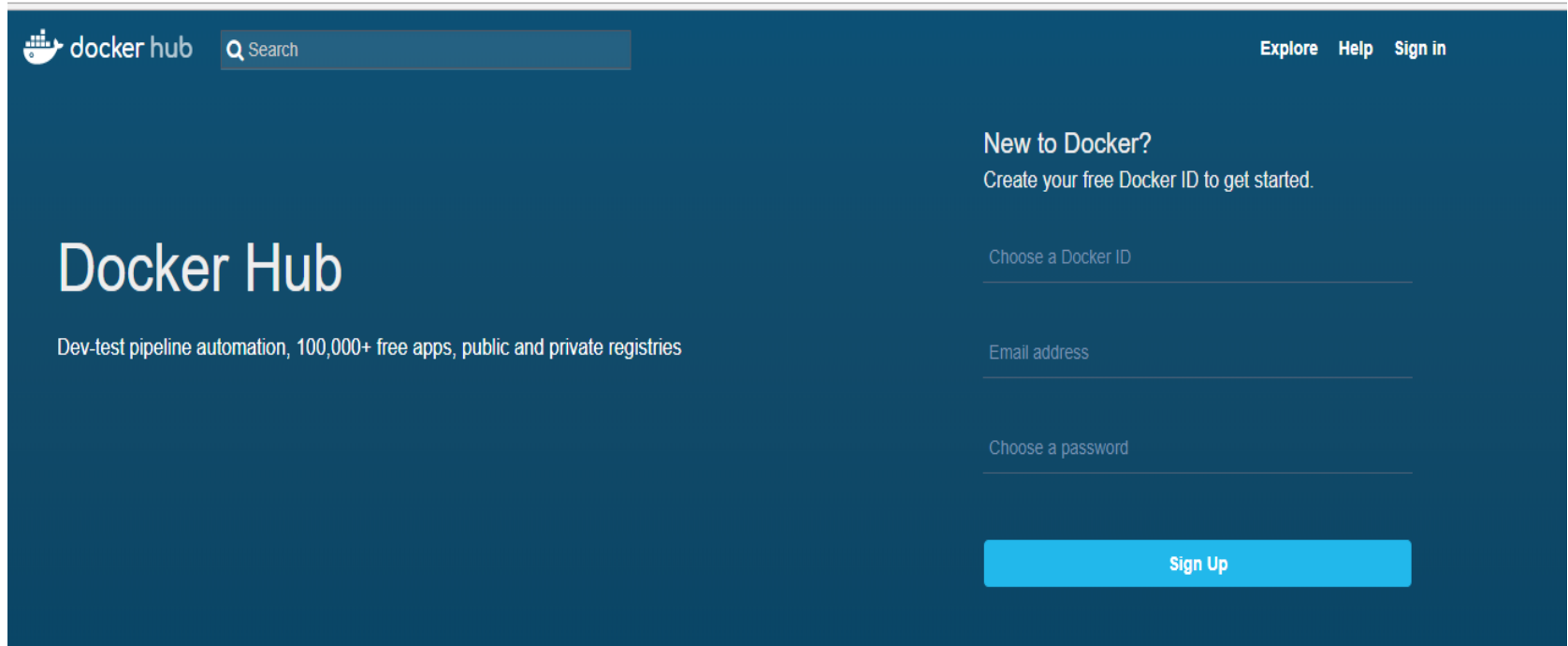
Container and layers

- The major difference between a container and an image is the top writable layer.
- All new/modification writes to the container are stored in this writable layer.
- Multiple containers can share access to the same underlying image and yet have their own data state.
- Multiple containers sharing the same image:



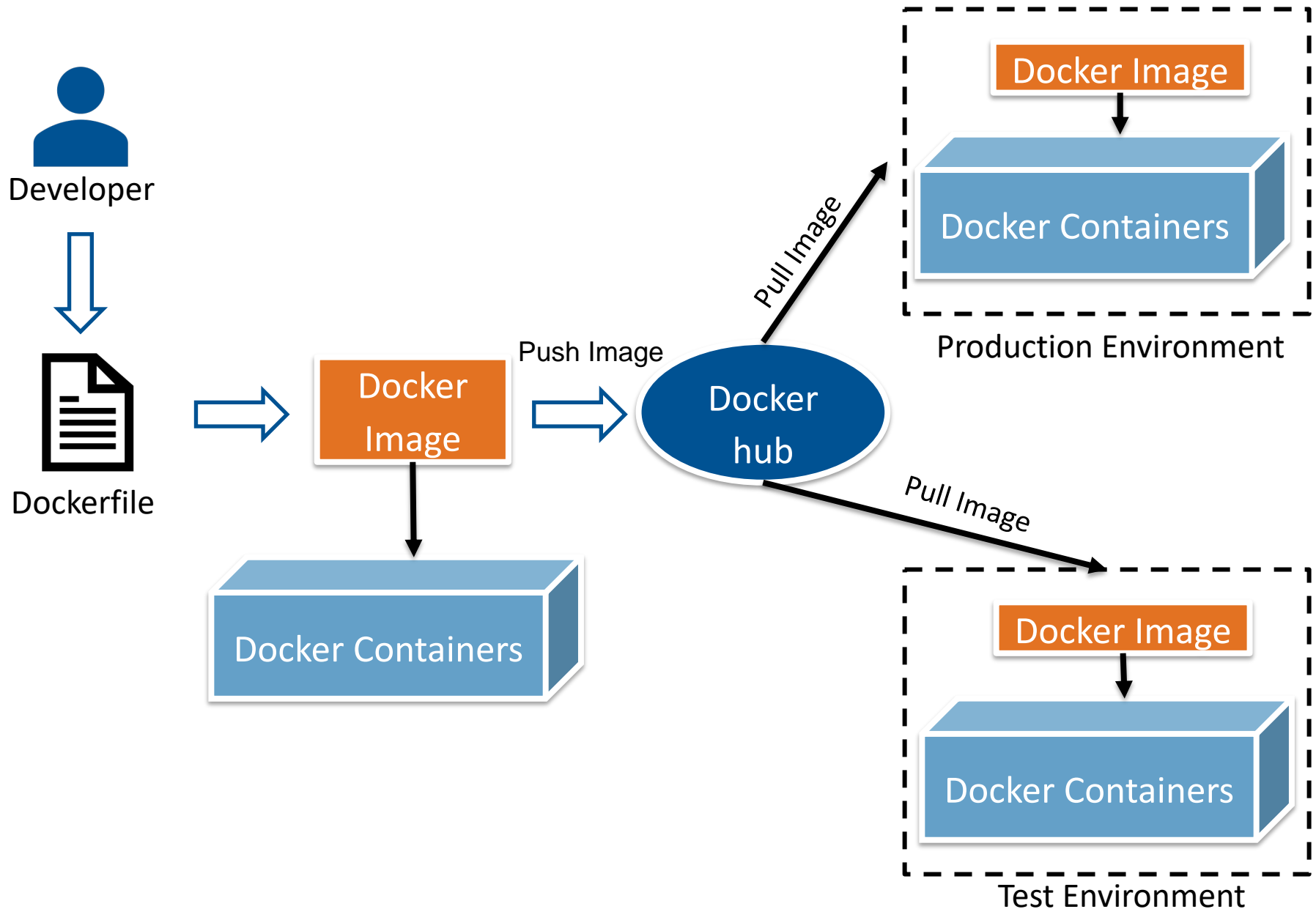
Docker Hub

- Cloud-based registry service. [Link](#)
- Allows you to link to code repositories, build your images, stores manually pushed images, and links to Docker Cloud so you can deploy images to your hosts.
- It provides a centralized resource for container image discovery, distribution and management



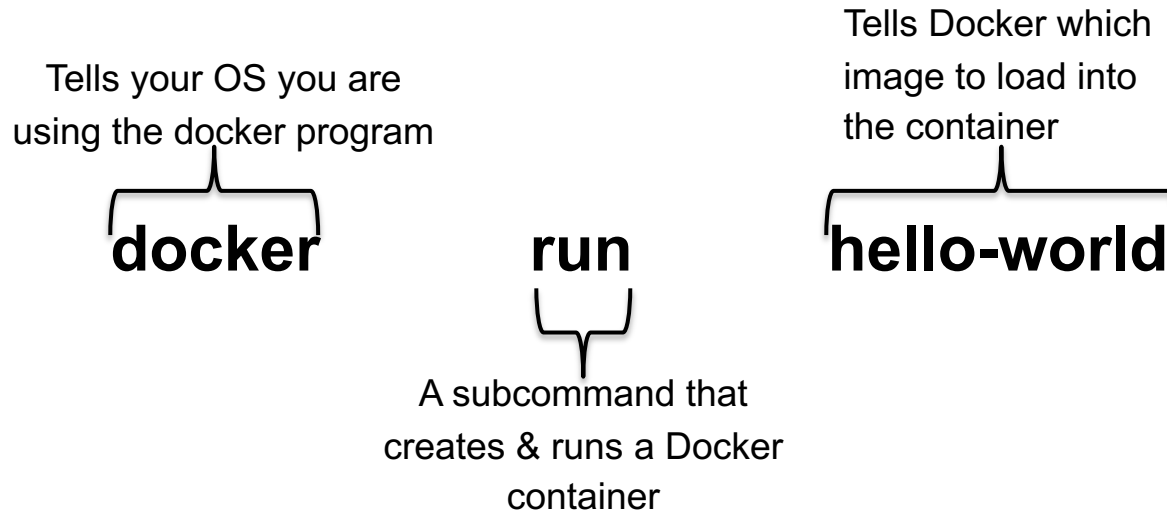
The screenshot shows the Docker Hub homepage with a dark blue background. At the top left is the Docker Hub logo and a search bar. At the top right are links for 'Explore', 'Help', and 'Sign in'. The main heading 'Docker Hub' is prominently displayed, followed by the tagline 'Dev-test pipeline automation, 100,000+ free apps, public and private registries'. On the right side, there is a 'New to Docker?' section with the text 'Create your free Docker ID to get started.' Below this are three input fields: 'Choose a Docker ID', 'Email address', and 'Choose a password'. A bright blue 'Sign Up' button is located at the bottom of this section.

Docker Containers build flow



Docker Run Command

Docker Run Command

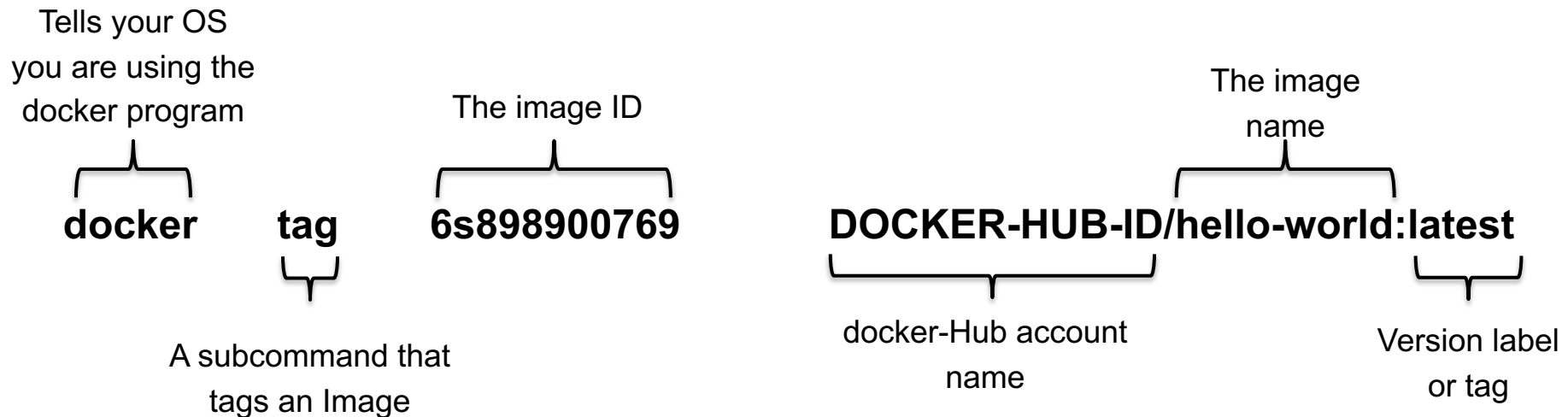


When we run the command, Docker Engine:

- check to see if we had the **hello-world** image
- download the image from the Docker Hub (if it's not present)
- load the image into the container and “run” it

Tag, Push and Pull your Image

- You can tag your image and push it to your docker hub account.
- Tags are used to identify different versions.



Other Important Docker Commands

Build an image from the Dockerfile in the directory and tag the image

`docker 1build -t <image-name>:<tag> <Directory>`

List all images that are locally stored with the Docker engine

`docker images`

List Running Containers

`docker ps`

Stop a running container

`docker stop <container-id>`

or `docker rm -f <container-id>`

Remove a running container

`docker rm <container-id>`

Delete an image from the local image store

`docker rmi <image-name>`

`docker 2run`

`--rm` remove container automatically after it exits

`-it` connect the container to terminal

`-d` This runs the container in the background (starts a container in detached mod)

`--name <nam>` name the container

`-p 5000:80` expose port 5000 externally and map to port 80

`-v ~/dev:/code` create a host mapped volume inside the container

`image:tag` the image from which the container is instantiated

`/bin/sh` the command to run inside the container

Docker-Compose

- Compose is a tool for defining and running multi-container Docker applications.
- It also helps to link multiple services.
- Uses a docker-compose.yml file, it is written in **YAML** format.
 - YAML (YAML Ain't Markup Language) is a human-readable data serialization language.
 - It uses Python-style indentation to indicate nesting, and uses [] for lists and {} for maps.
 - YAML 1.2 a superset of JSON.

docker-compose.yml file



```
version: '3'
```

```
services:
```

```
  server:
```

```
    build: ./server
```

```
    image: HUB_ID/cloudcomputinggroup#:latest
```

```
    container_name: cloudcomputinggroup#
```

```
    depends_on:
```

```
      - "mongodb"
```

```
    environment:
```

```
      - MONGODB_URI= mongodb://mongodb:27017/booksData
```

```
    ports:
```

```
      - "3000:3000"
```

```
  mongodb:
```

```
    image: mongo:latest
```

```
    container_name: "mongodb"
```

```
    environment:
```

```
      - MONGO_DATA_DIR=/data/db
```

```
    volumes:
```

```
      - ./data:/data/db
```

```
    ports:
```

```
      - "27017:27017"
```

Version of docker-compose file

Start of all services

Server service container

Path to make the image from

Image location on docker hub

Name of the container

This service depends on mongodb service.

Environment Variables

Mapping of VM port to container port

MongoDb container

Docker hub repo/image name of mongodb

Environment Variables

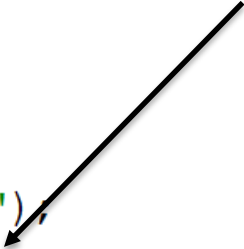
Volume to be mounted

Mapping of VM port to container port

docker-compose.yml file continued..

Usage of environment variable

```
const mongoose = require("mongoose");  
mongoose.connect( process.env.MONGODB_URI ||  
  "mongodb://localhost:27017/booksData",{ useNewUrlParser: true });
```



Docker-Compose Commands

Build all the images

`docker-compose1 build`

Build only the selected image

`docker-compose build <image-name>`

Log in to a registry (the Docker Hub by default)

`docker login`

`docker login <registry-host>`

Push images to a registry

`docker-compose push`

`docker-compose config`

To start all containers

`docker-compose2 up`  `down`

To start all containers in background

`docker-compose up -d`

Stop the containers

`docker-compose stop`

Kill the containers

`docker-compose kill`

Remove stopped containers

`docker-compose rm`

Installation and Running the application

Install docker and docker-compose

Docker Installation (use this official steps only):

<https://docs.docker.com/install/linux/docker-ce/ubuntu/>

Docker Compose Installation:

<https://docs.docker.com/compose/install/#prerequisites>

Enable Docker Remote API on the VM (Ubuntu)



1. Edit the file `nano /lib/systemd/system/docker.service`
2. Modify the line that starts with `ExecStart` to look like this

`ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock -H tcp://0.0.0.0:4243`

Where the addition is `"-H tcp://0.0.0.0:4243"`

3. Save the modified file
4. Run `sudo systemctl daemon-reload`
5. Run `sudo service docker restart`
6. Test that the Docker API is accessible:

`curl http://localhost:4243/version`

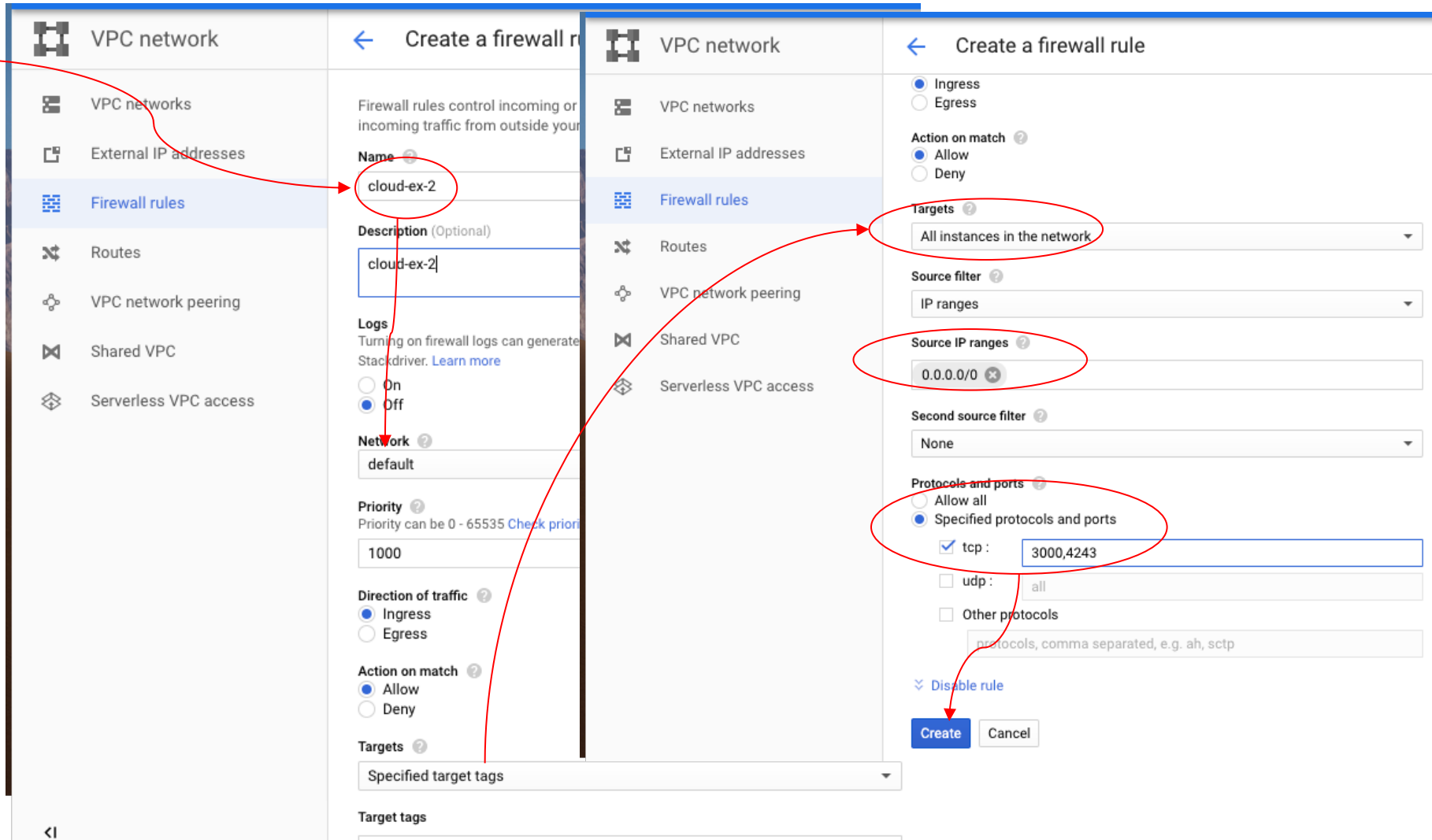
or `curl http://localhost:4243/images/json`

Remember to enable port **4243** on your VM so that docker API can be accessed from outside network using your VM IP.

Enabling Ports on GCP

The screenshot illustrates the steps to navigate to the Firewall rules page in the Google Cloud Platform console. A red arrow originates from the left-hand navigation menu, pointing to the 'VPC network' option under the 'NETWORKING' section. A second red arrow points from the 'VPC network' option in the left menu to a sub-menu that appears after clicking, where 'Firewall rules' is highlighted. A third red arrow points from the 'Firewall rules' option in the sub-menu to the 'Firewall rules' page in the main console area. The 'Firewall rules' page shows a table with columns for 'VPC network' and 'Firewall rules', and includes buttons for '+ CREATE FIREWALL RULE', 'REFRESH', and 'DELETE'. Below the table, there is explanatory text about firewall rules and a link to 'Learn more'.

Enabling Ports on GCP



The image displays two screenshots of the Google Cloud Platform (GCP) console, illustrating the steps to create a firewall rule.

Left Screenshot: Create a firewall rule

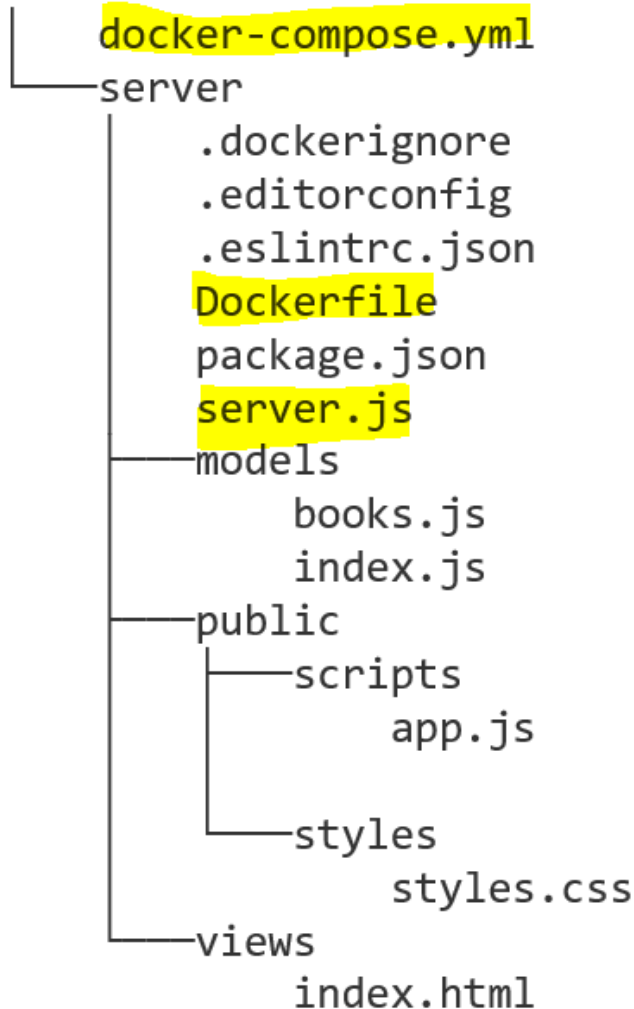
- Navigation:** The left sidebar shows the "VPC network" menu with "Firewall rules" selected.
- Name:** The name "cloud-ex-2" is entered and circled in red.
- Description:** The description "cloud-ex-2" is entered in the optional field.
- Logs:** The "On" radio button is selected.
- Network:** The "default" network is selected.
- Priority:** The priority is set to "1000".
- Direction of traffic:** The "Ingress" radio button is selected.
- Action on match:** The "Allow" radio button is selected.
- Targets:** The "Specified target tags" option is selected.

Right Screenshot: Create a firewall rule

- Direction:** The "Ingress" radio button is selected.
- Action on match:** The "Allow" radio button is selected.
- Targets:** The "All instances in the network" option is selected and circled in red.
- Source filter:** The "IP ranges" option is selected.
- Source IP ranges:** The "0.0.0.0/0" range is entered and circled in red.
- Second source filter:** The "None" option is selected.
- Protocols and ports:** The "Specified protocols and ports" radio button is selected and circled in red. Below it, "tcp" is checked, and the ports "3000,4243" are entered in the adjacent field.
- Buttons:** The "Create" button is highlighted with a red arrow.

Application Download and Make Changes

Download the provided application source zip file from Moodle.



Highlighted are the ones which need to modified or added

Running and Testing the Application

- Do the changes to the application on your local laptop/computer
- Check the application is running or not locally
`sudo docker-compose up --build`

If everything is working correctly now :

- Create a repository on docker hub
- Login to your hub account using the command on your local machine:
`sudo docker login`
- Push Images to docker hub (don't forget to add your docker hub id and image name into **docker-compose.yml** file)
`sudo docker-compose push`
- Copy the **docker-compose.yml** file to the VM and remove **build** line from it
- Run the application using docker-compose
`sudo docker-compose up`
- Test the Application is running or not by going to the URLs:
 - http://YOUR_VM_PUBLIC_IP:3000/api/exercise2
 - http://YOUR_VM_PUBLIC_IP:3000/api/profile
 - http://YOUR_VM_PUBLIC_IP:3000/api/books

Tasks to be Completed

Tasks to be completed

As part of the exercise2, there are following tasks to be completed:

1. Add an API in your application:
 1. /api/exercise2 : Which sends a message "**group # application deployed using docker**" back to the user when called the API.
 2. Make sure to use the same completed application (all the first exercise tasks).
2. Create the missing docker file of your application and then build the image.
3. Create your docker hub account and **push** your application image to it. First do a **docker login** then **docker-compose push**.
4. Name of your application image should be as **cloudcomputinggroup#**
5. Start a VM and run the provided docker-compose file. This will **pull** this application image on it along with mongo image and run them using docker.
6. Enable docker remote API

Deadline for submission: Check the exercise page on server

*Replace # with your group number in above tasks.

Submission

Submission Instructions

To submit your application results you need to follow this :

1. Open the Cloud Class server url : <https://cloudcom.caps.in.tum.de/>
2. Login with your provided username and password.
3. After logging in, you will find the button for **exercise2**
4. Click on it and a form will come up where you must provide
 - VM ip on which your application is running
 - and the **dockerhub** image path name.

Example:

10.0.23.1

dockerHubUserId/myImageName

5. Then click submit.
6. You will get the correct submission from server if everything is done correctly.

Remember no cheating and no Hacking 😊

Hints



- For sending message from the API use **res.send("message here")** method.
- First test locally then only submit on server.
- Enable ports on VM
 - 3000
 - 4243

Thank you for your attention!