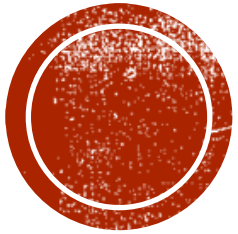
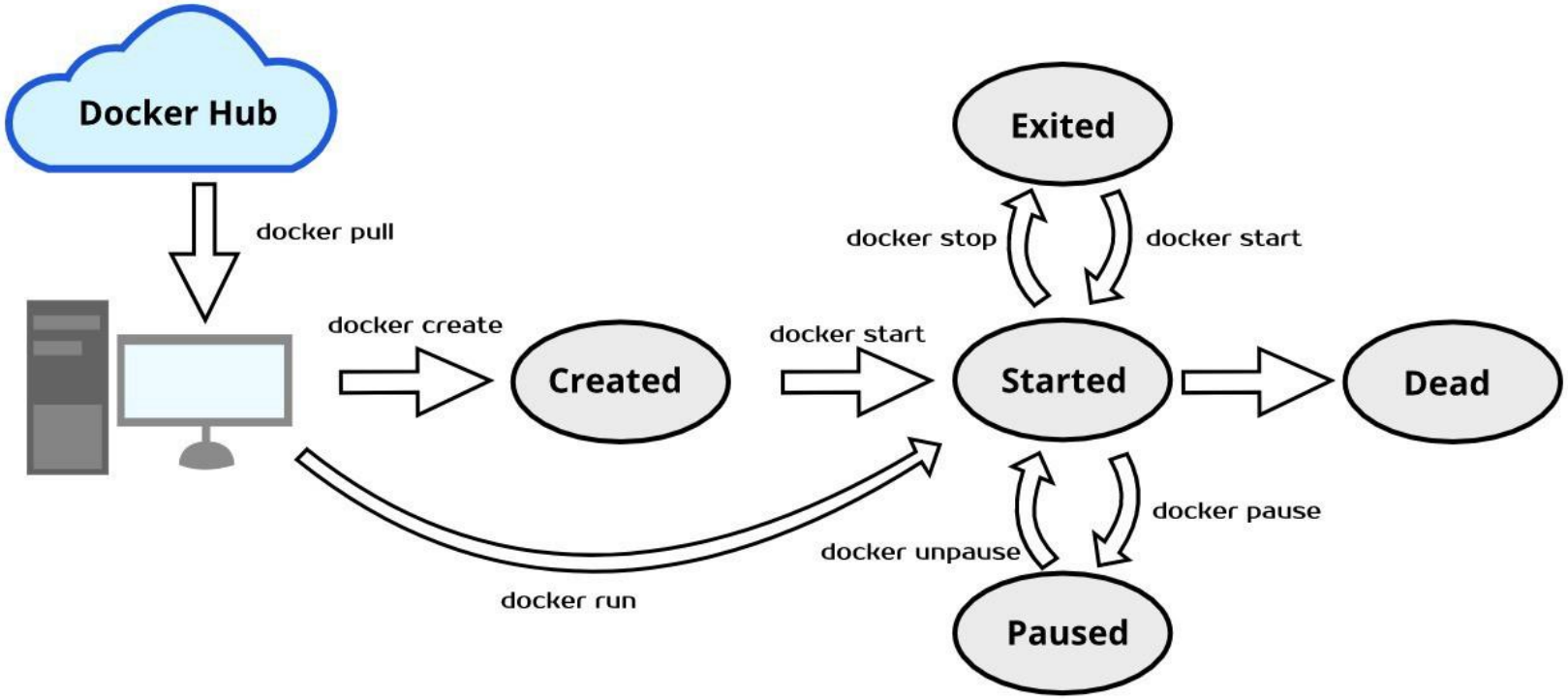


# FROM VIRTUALIZATION TO CONTAINERIZATION

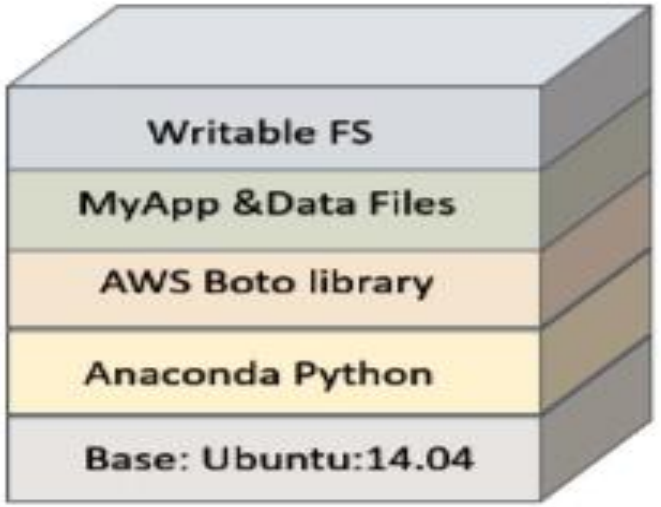
# DOCKER LIFECYCLE OVERVIEW



# DOCKER'S UNION FILE SYSTEM

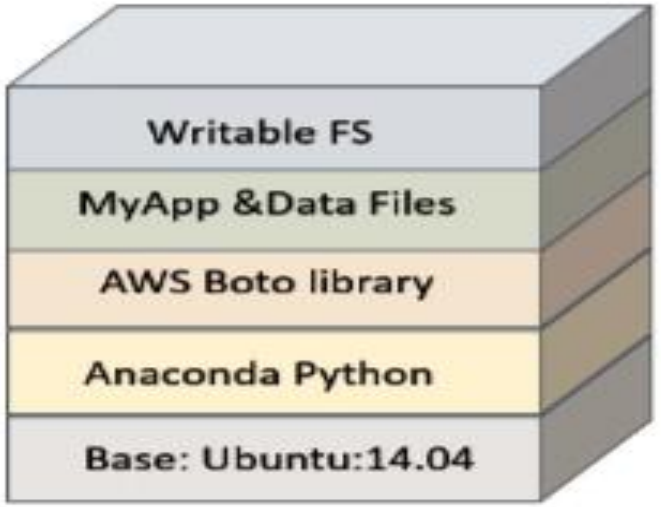
One important feature that makes docker unique is its filesystem!

- Docker's Union File Systems basically allow you to take different file systems and create a union of their contents with the top most layer superseding any similar files found in the file systems
- Docker images are composed of layers in the Union File System. The image is itself a stack of read-only directories. The base is a simplified Linux file system.
  - additional tools that the container needs are then layered on top of that base, each in its own layer.
- All containers with the same image see the same directory tree
  - Load the directory tree in the memory only once among all instances



# DOCKER'S UNION FILE SYSTEM – CONT'D

- When the container is run, a final writable file system is layered on top.
- As an application in the container executes, it uses the writable layer. If it needs to modify an object in the read-only layers, it copies those objects into the writable layer.
- Otherwise, it uses the data in the read-only layer, which is shared with other container instances.
- Thus, typically only a little of the container image needs to be actually loaded when a container is run, which means that containers can load and run much faster than virtual machines.
- In fact, launching a container typically takes less than a second, while starting a virtual machine can take minutes




Notes

- In addition to the file system layers in the container image, you can mount a host machine directory as a file system in the container's OS.
- In this way a container can share data with the host. Multiple containers can also share these mounted directories and can use them for basic communication of shared data

# GET START WITH DOCKER

- Download Docker from https://docs.docker.com/get-started/ and install on your machine
  - Once installed, the docker daemon is started. If not, manually start it.

```
## List Docker CLI commands
docker
docker container --help
## Display Docker version and info
docker –version
docker version
docker info
## Execute Docker image
docker run hello-world
## List Docker images
docker image ls
## List Docker containers (running, all, all in quiet mode)
docker container ls
docker container ls --all
docker container ls -aq
```

# DOCKER TIPS AND TRICKS

```
# List docker running process
docker ps
# List all docker processes
docker ps –a

# Remove all containers with status existed
docker rm $(docker ps -q -f status=exited)

# Stop active containers
docker stop $(docker ps -q)
# Stop all containers
docker stop $(docker ps -aq)

# Run and attach to container
# '--rm' delete container after exit
docker run -it --rm <image_name> /bin/ash
```

```
# Pass environment variables to docker
docker run -it -e TEST=1234 --env TEST1=3456 --rm alpine /bin/ash
#write in terminal
echo $TEST - you should have result '1234'

# Remove all docker images
docker rmi $(docker images -q)

# Execute command in a container
docker exec YOUR_CONTAINER echo "Hello from container!"
```

- Good to know Docker flags
  - -it connect the container's standard I/O to the shell that ran the docker command
  - -d detached mode (no interaction)
  - -v [localdir]:[containerdir] mount the localdir to the conainterdir in the container
  - docker run -it -v /tmp:/localtmp ubuntu

# DOCKERFILE CONFIG EXAMPLE

```
FROM openjdk:7
COPY . /usr/src/myapp
WORKDIR /usr/src/myapp
RUN javac Main.java
CMD ["java", "Main"]
```

Build on top of this image

Copy Contents to the following directory

Define Work Directory

Run this script when starting the image

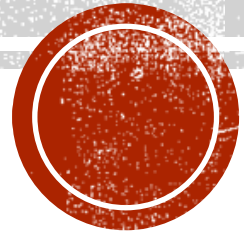
- Don't get confused with the following Docker commands:
  - RUN
    - Executes command(s) in a new layer and creates a new image. E.g., it is often used for installing software packages.
  - CMD
    - Sets default command and/or parameters, which can be overwritten from command line when docker container runs.
  - ENTRYPOINT
    - Configures a container that will run as an executable.

# BUILD, TEST, AND SHARE A DOCKER FILE

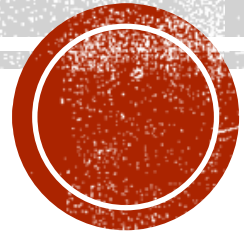
- Build Your Docker Image (and tag it)
  - docker build –t *yourDockerHubId/chooseName* .
- Test Your Image (you may not need the arguments depending on the case)
  - docker run –d –p 8000:8000 *yourDockerHubId/chooseName*

  *Or*
  - *docker run yourDockerHubId/chooseName*
- Upload to the Docker Hub to share with others
  - docker push yourname/bottlesamp

# DEMO FOR DOCKER AND GCP



# DON'T FORGET TO DISABLE BILLING AFTER YOU FINISH!!



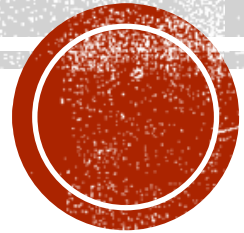
# DEPLOYING DOCKER HUB IMAGES TO GCP

- You will need extra effort to deploy your Docker hub image to GCP.
- Check the following articles:
  - https://cloud.google.com/build/docs/interacting-with-dockerhub-images
  - https://cloud.google.com/build/docs/configuring-builds/create-basic-configuration

# VIRTUAL MACHINES VS CONTAINERS

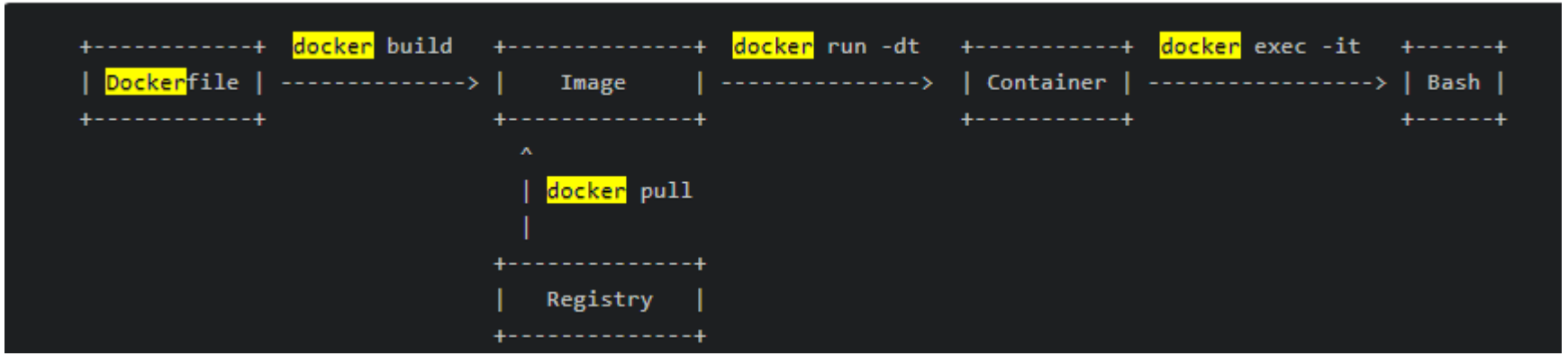
| Virtual machines | Containers |
| --- | --- |
| Heavyweight<br>Fully isolated; hence more secure<br>No automation for configuration<br>Slow deployment<br>Easy port and IP address mapping Custom images not portable across clouds | Lightweight<br>Process-level isolation; hence less secure<br>Script-driven configuration<br>Rapid deployment<br>More abstract port and IP mappings<br>Completely portable |
| Citrix Xen,<br>Microsoft Hyper-V,<br>VMWare ESXi,<br>VirtualBox,<br>KVM | Docker,<br>Google container,<br>LXC – Linux kernel container,<br>FreeBSD jails,<br>Solaris Zones |

# DOCKER COMMON INTERVIEW QUESTIONS



# EXPLAIN BASIC DOCKER USAGE WORKFLOW

- Everything starts with the **Dockerfile**. The Dockerfile is the source code of the Image.
- Once the Dockerfile is created, you build it to create the **image** of the container. The image is just the "compiled version" of the "source code" which is the Dockerfile.
- Once you have the image of the container, you should redistribute it using the **registry**. The registry is like a git repository -- you can push and pull images.
- Next, you can use the image to run **containers**. A running container is very similar, in many aspects, to a virtual machine (but without the hypervisor).



# WHAT TYPE OF APPLICATIONS - STATELESS OR STATEFUL ARE MORE SUITABLE FOR DOCKER CONTAINER?

- It is preferable to create Stateless application for Docker Container. We can create a container out of our application and take out the configurable state parameters from application. Now we can run same container in Production as well as QA environments with different parameters. This helps in reusing the same Image in different scenarios. Also a stateless application is much easier to scale with Docker Containers than a stateful application.

# OTHER QUESTIONS

- How is Docker different from a virtual machine?
- How will you monitor Docker in production?
- What is Docker Swarm?
- What is Docker image?
- What is Docker container?

# TODO

- Homework-1