

Containers and Docker

footer: font_size: \$base_font_size_small font_color: \$base_font_color # NOTE if background_color is set, background and border will span width of page border_color: dddddd border_width: 0.25 height: \$base_line_height_length * 2.5 padding: [\$base_line_height_length / 2, 1, 0, 1] valign: top recto_content: right: {section-or-chapter-title} | {page-number} verso_content: left: {page-number} | {chapter-title}

header: Containers and Docker - ECSE 321 Tutorial 7

#0. XaaS

<https://www.ibm.com/blogs/cloud-computing/2014/10/xaas-family-iaas-paas-saas-explained/>

#1. Virtualization types

Using a virtualization method and building an image provides consistent environments. The virtualization method can be categorized based on how it mimics hardware to a guest operating system and emulates guest operating environment. Primarily, there are two main types of virtualization:

1. **Emulation and paravirtualization** is based on some type of hypervisor which is responsible for translating guest OS kernel code to software instructions or directly executes it on the bare-metal hardware. A virtual machine provides a computing environment with dedicated resources, requests for CPU, memory, hard disk, network and other hardware resources that are managed by a virtualization layer. A virtual machine has its own OS and full software stack.
2. **Container-based virtualization**, also known as operating system-level virtualization, enables multiple isolated executions within a single operating system kernel. It has the best possible performance and density, while featuring dynamic resource management. The isolated virtual execution environment provided by this type of virtualization is called a *container* and can be viewed as a well-defined group of processes.

![Platform- and OS- level virtualization differences](virtualization-platform-os.png)

#2. Docker concepts

The core concepts of Docker are **images** and **containers**.

A **Docker image** is a read-only template with instructions for creating a Docker container. For example, an image might contain an Ubuntu operating system with an Apache web server and your web application installed. You can build or update images from scratch or download and use images created by others. An image may be based on, or may extend, one or more other images. A Docker image is described in text file called a Dockerfile, which has a simple, well-defined syntax.

A **Docker container** is a runnable instance of a Docker image. You can run, start, stop, move, or delete a container using Docker API or CLI commands. When you run a container, you can provide configuration metadata such as networking information or environment variables.

#3. Docker basics + hands-on

- **Dockerfile:** An image is defined in a **Dockerfile**. Every image starts from a base image, e.g. from **ubuntu**, a base Ubuntu image. The Docker image is built from the base image using a simple, descriptive set of steps we call instructions, which are stored in a **Dockerfile**. Main dockerfile instructions:

- FROM
- RUN
- ADD
- VOLUME
- CMD
- WORKDIR

[More details here](<https://docs.docker.com/engine/reference/builder/>); example for tomcat is here: <https://blog.lukaspradel.com/dockerizing-a-tomcat-postgresql-java-web-application/>

- **Command `docker build .`:** it builds an **image** from a **Dockerfile** and a **context**. The build's **context** is the set of files at a specified location PATH or URL (in this case the current directory, **.**). The PATH is a directory on your local filesystem. The URL is a Git repository location. Add the **-f** switch to specify the Dockerfile location, if it is not present in the root context.
- **Command `docker run`:** Run a command in a new container. Images come to life with the **docker run** command, which creates a container by adding a read-write layer on top of the image. This combination of read-only layers topped with a read-write layer is known as a [union file system](<https://en.wikipedia.org/wiki/UnionFS>). Changes exist only within an individual container instance. When a container is deleted, any changes are lost unless steps are taken to preserve them. Parameters to show: **-v, -i, -t, -a**
- **Command `docker start/stop`:** Start/stop a container.
- **Command `docker exec`:** Execute a command in a running container.
- Managing images and containers:

`docker ps --all:`

CONTAINER ID	IMAGE	COMMAND	STATUS
9bba8a2a3f81	makisyu/texlive-2016	"/bin/bash -c 'sleep...'"	Exited (0) 4 days ago
cd005b9af0af	makisyu/texlive-2016	"/bin/bash -c 'sleep...'"	Exited (0) 4 days ago
b92dd4d5886d	eclipse/che	"/scripts/entrypoint..."	Exited (2) 5 days ago

`docker images ls --all:`

REPOSITORY	IMAGE ID	CREATED	SIZE
eclipse/che	8956a46aa7e3	10 days ago	51.3MB
gradle	e7f185032db8	2 months ago	820MB
busybox	6ad733544a63	3 months ago	1.13MB
makisyu/texlive-2016	bb92f3e57f6b	9 months ago	5.42GB

#4. DockerHub + demo

Dockerhub and its usage. Related commands:

- **Command** `docker pull`
- **Command** `docker push`
- **Command** `docker save`
- **Command** `docker load`

#5 Exercises:

pull busybox ping a server with busybox (show ping, ping from docker) show interactive mode

```
docker attach
docker run -d
docker run -it

- if [[ -e docker/texbuilder.tar.gz ]] ; then gzip -dc docker/texbuilder.tar.gz |
docker load ; else docker pull makisyu/texlive-2016 ; docker save makisyu/texlive-
2016 | gzip > docker/texbuilder.tar.gz ; fi

docker run -d --name texbuilder -v `pwd`::/mnt/ makisyu/texlive-2016 /bin/bash -c
"sleep 10"
docker ps -a
docker exec -ti texbuilder /bin/bash -c "cd /mnt && pdflatex /mnt/calendar_1.tex"
ls -la

docker run -it -v `pwd`::/mnt/ openjdk
```

Miscellaneous

Running docker without sudo

Taking the steps from this answer on askubuntu

1. Add the docker group if it doesn't already exist: `sudo groupadd docker`
2. Add the connected user "\$USER" to the docker group. Change the user name to match your preferred user if you do not want to use your current user: `sudo gpasswd -a $USER docker`
3. Either do a `newgrp docker` or log out/in to activate the changes to groups. You can use `docker run hello-world` to check if you can run docker without sudo.