Introduction to Data Analytics

Docker

András Varga
IBM Consulting

Bratislava, 2022

Contents

Docker

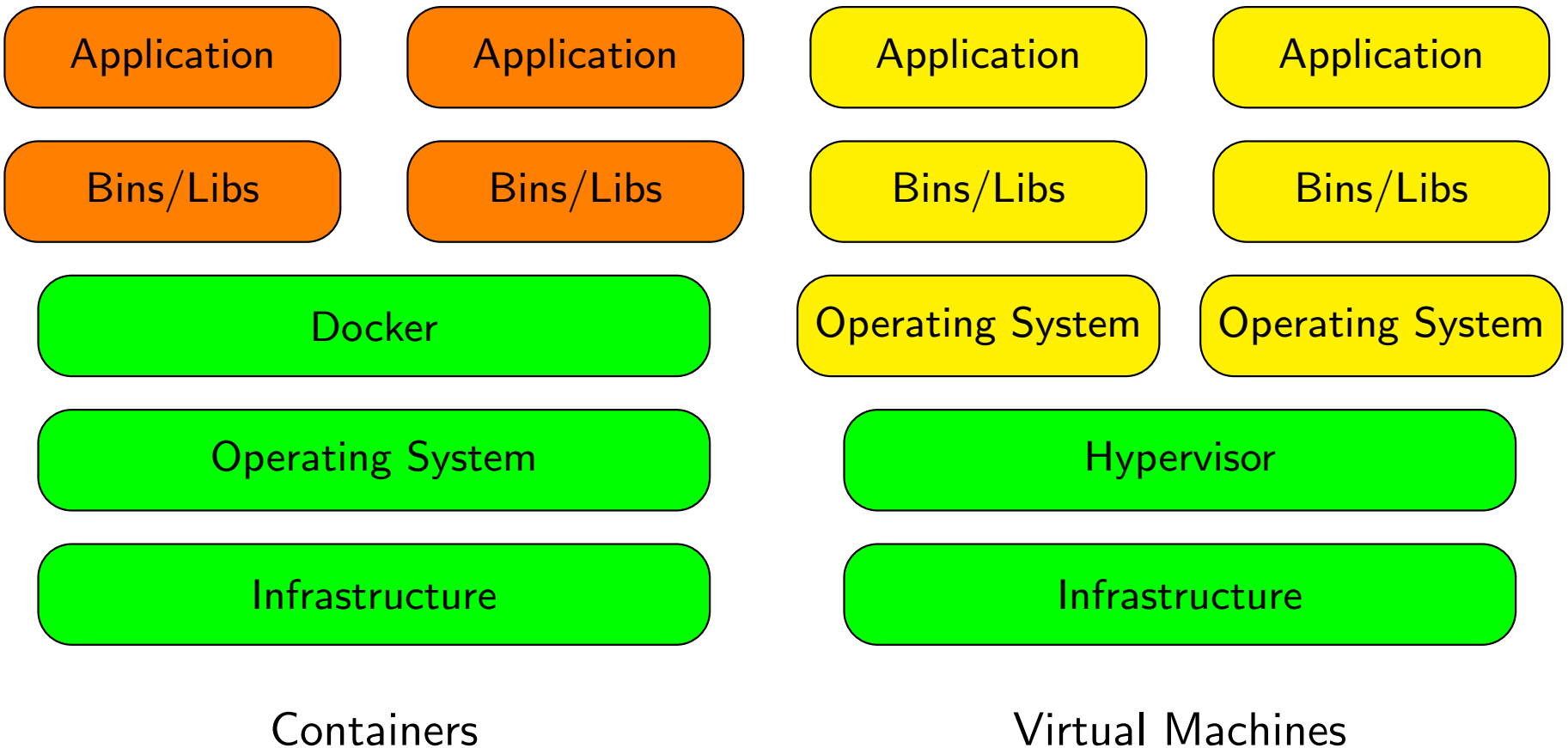
Definition

A container is a sandboxed process on your machine that is isolated from all other processes on the host machine. Docker is a particular solution to create and execute containers.

Details:

- Containers are portable and isolated
- Originally for Linux, currently Windows and Mac is supported as well
- Container images can be shared
- Lightweight
- Provides simpler build and deployment pipelines
- www.docker.com

Container vs. Virtual Machine



How Docker Works

Docker uses several long existing features of the Linux kernel to achieve isolation:

- cgroups - control groups provide an option to control the allocation of CPU or memory resources
- namespaces - provide users access to elevated commands inside a namespace

Security is the usual complaint, but it can be fixed by proper technique

Docker Vocabulary

- The application is built into so called *container images* or simply *images*
- Images are collected and stored into public or private *registries*
- *Docker Hub* is the largest public registry, or image library
- A *container* is a running instance of an image
- *Dockerfile* is a descriptive file providing a recipe to build a particular container image

Building an Image

- Command to use: docker build *context*
- A context can be a git repository, a tarball, or a path
- The build uses the default name for Dockerfile (PATH/Dockerfile)
- The non-default Dockerfile name can be set by using the -f option

Dockerfile Example

```
FROM ubuntu:22.04

ENV DATAPATH=/data

RUN mkdir ${DATAPATH}
RUN chmod -R 0777 ${DATAPATH}
USER testUser
ADD --chown=myuser:mygroup file_from_context $DATAPATH
CMD ["/usr/bin/wc","--help"]

LABEL version="1.0" description="dummy"
EXPOSE 80/tcp

VOLUME ["/var/www", "/var/log/apache2"]
ENTRYPOINT ["/usr/sbin/apache2ctl", "-D",
"FOREGROUND"]
```

Working With Images

- Important option of docker build: *-t myname/mytag* providing a "name", tag for the final image
- docker push myname/mytag - pushes the image into a registry
- docker pull someimage - pulls an image from a registry

Docker Image Layers

- Docker images are built in so called "layers"
- After a change is executed from a Dockerfile a new (unnamed) image - a layer is created and stored locally
- This means many RUN commands executed one after another uses up more space on the disk
  - → It is advised to execute more commands in one RUN to reduce layers
- When changing the Dockerfile or context the image is re-built from the last non-changed layer, speeding up the build process

Starting Containers

- Containers are executed based on images
- Technically, during the container startup a new layer is added to the image, which will contain the changes performed during the execution

Basic Docker Run Command

docker run **-d** **-p** 80:80 my_image *service nginx start*

- The simplest way to start a container
- Containers can be started in a foreground or in a detached mode (-d) as a "daemon"
- Mapping of ports is defined from outside the container into the inside of it using the -p option
- Persistent volumes can be mounted to allow containers to store data using the **--mount** or **-v** options

Docker Network

- Docker provides a way to isolate the network containers use
- Docker networks can spawn over a cluster of computers
- iptables are manipulated by docker to achieve the isolation
- Containers can listen to specific networks

Listing Docker Objects

- Listing all images: *docker images*
- Listing running containers: *docker container ls*
- Listing docker networks: *docker network ls*

Swarm

- A native cluster management option of Docker
- Managed by *docker swarm* command
- It runs tasks in nodes
- Handles the usual tasks, e.g., scaling and load balancing

Docker Compose

- Handles the execution of containers in a single host
- Provides isolation for whole applications
- Managed by *docker compose* command
- The environments can be described in a yaml file

Kubernetes - K8s

- Is a generic container orchestration tool
- Provides the usual set of services, e.g., self-healing, scaling, load balancing
- *Pods* are deployed into *nodes*
- Docker containers can run in such pods

Brief Introduction to Microservices

- It is an **architecture pattern** heavily utilizing containers and orchestration (and aggressive, automated testing)
- The application is broken down to individual services, each running in an isolation
- Hence these services are usually small
- Each service is usually developed and deployed separately, often by different teams
- Messaging between the services is complex
- Databases are also somewhat problematic to handle

Build and Release Pipelines

- Continuous Delivery is often split into Build and Release stages, facilitated by so called pipelines
- Using docker the build pipeline usually relies on a *docker build* command and pushing into a registry
- Release pipelines are usually handling configuration changes in the orchestration tools

Best Practices

- Create ephemeral containers (can be destroyed and re-started at will)
- Use .dockerignore
- Usually install more generic libraries first, add your code later to have more generic intermediate layers
- Decouple applications
- Sort the arguments
- Minimize the amount of layers

Best Practices - Security

- Do not install SSH server into a docker image
  - → Use the command *docker exec* or *docker attach* instead on the running container
- Never start from untrustworthy, unknown images
  - → It is better to start from a known, or even from a blank OS image
- Dockerfile is open text, do not use passwords in it
  - → E.g. use vaults
- Encrypt docker images if needed

See more https://developer.ibm.com/articles/encrypted-container-images-for-container-image-security-at-rest/

Alternatives

Docker is vastly popular, other alternatives are dwarfed in comparison:

- RedHat OpenShift is an orchestration tool
- LXD (by Ubuntu)
- Containerd

and many more...