DOCKER

LESSON 02

SW4BED-01

AGENDA

- What is Docker?
- Creating images
- Build and deploy containers
- Persisting data
- Multi-container applications

WHAT IS DOCKER?

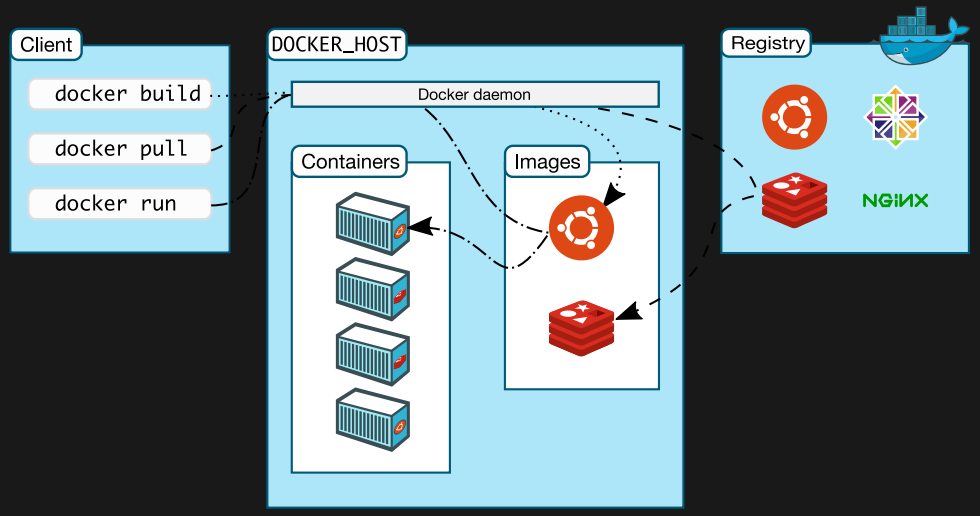
OVERVIEW

- Docker is an open platform for developing, shipping and running applications
- Docker separates applications from infrastructure
- Docker let you manage your infrastructure in the same way you manage your applications

DOCKER ARCHITECTURE

- Docker daemon
  - Listens for Docker API requests
  - Manages Docker objects
- Docker client
  - The primary way users interact with Docker
- Docker registries
  - Stores Docker images
  - Docker Hub is a public registry (used by default). You can run a private registry
  - `docker pull/run` fetches required images, and `docker push` publishes images to registries

DOCKER ARCHITECTURE



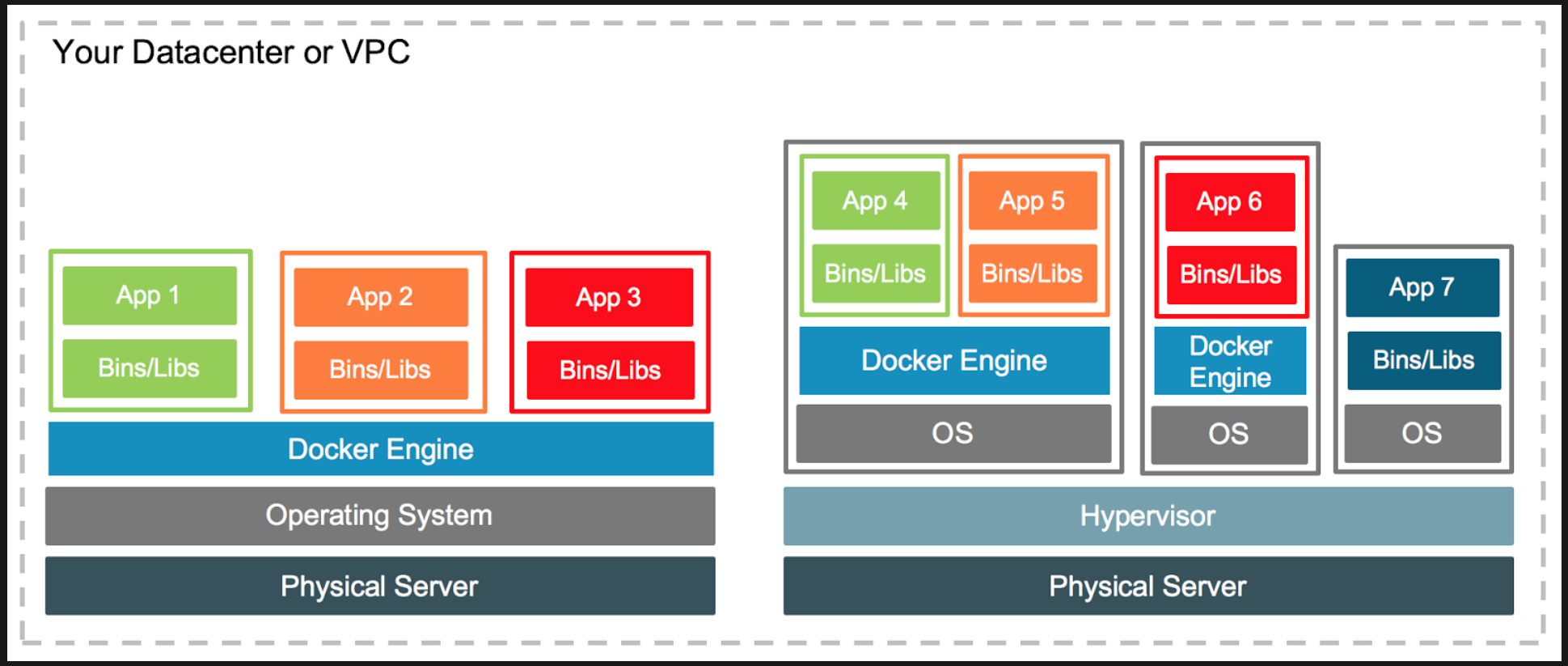
DOCKER IMAGES

- Images are read-only templates with instructions for creating containers
  - Images are typically based on other images, with some customization
- To build an image, you create a Dockerfile
  - A simple syntax defines the steps needed to produce an image and run it
  - Each instruction create a layer in the image
- Layers are what makes Docker so lightweight, small, and fast when compared with other virtualization technologies
  - When rebuilding an image, only those layers which have changed are rebuilt

DOCKER CONTAINERS

- A container is a runnable instance of an image
- A container is relatively well isolated from other containers and its host machine
- A container is defined by its image as well as any configuration you provide to it when you create or start it
  - When a container is removed, any changes to its state that are not stored in persistent storage disappear

DOCKER DEPLOYMENT



Containers and VMs Together - Docker Blog

CREATING IMAGES

OVERVIEW

- Dockerfiles
- Building images
  - Using `docker build`
  - Multi-stage builds
- Publishing images to registries

DOCKERFILES

```
# https://hub.docker.com/_/microsoft-dotnet
FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /source

# copy csproj and restore as distinct layers
COPY *.sln .
COPY api/*.csproj ./api/
RUN dotnet restore

# copy everything else and build app
COPY api/. ./api/
WORKDIR /source/api
RUN dotnet publish -c release -o /app

# final stage/image
FROM mcr.microsoft.com/dotnet/aspnet:6.0
WORKDIR /app
COPY --from=build /app ./
ENTRYPOINT ["dotnet", "api.dll"]
```

examples/lesson-02-docker/hello-docker/Dockerfile

.NET CORE DOCKER IMAGES

- Challenging to keep image size small
- Multi-stage build
  - Build application in container optimized for development, debugging, and testing
  - Run application in optimized from production
- Restore in distinct layer to use caching features
  - If the project file(s) have not changed, use cached layers

DOCKER BUILD

- The `docker build` command builds an image from a Dockerfile and a context
  - A build's context is a set of files located in a `PATH` or `URL`
  - When the `URL` points to a Git repository, that acts as the context
- Options
  - `--pull` —Always attempt to pull a newer version of the image
  - `--file, -f` —Name of the Dockerfile (Default is 'PATH/Dockerfile')
  - `--tag , -t` —Name and optionally a tag in the 'name:tag' format
- Use a `.dockerignore` file to exclude files from being sent to the Docker daemon

PUBLISHING IMAGES

- Use `docker push` to upload and publish images in registries
  - Default registry: Docker Hub
- Be sure to tag the image as `<USERNAME>/<IMAGE_TAG>`

CREATING AND RUNNING CONTAINERS

CONTAINERS

- A container a runnable instance of an image
- Containers are ephemeral
  - They can be stopped, destroyed, rebuilt and replaced with minimum set up and configuration

`docker run`

- `docker run [OPTIONS] IMAGE [COMMAND] [ARG...]`
- Options
  - `--detach, -d` —Run container in background and print container ID
  - `--rm` —Automatically remove the container when it exits
  - `--tty, -t` —Allocate a pseudo-TTY
  - `--interactive, -i` —Keep STDIN open even if not attached
  - `--volume, -v` —Bind mount a volume
  - `--name` —Assign a name to the container
  - `--publish, -p` —Publish a container's port(s) to the host
- `docker run --rm -p 5000:80 my_app`
  - Run image named `my_app` and expose port 80 in the container on 5000 on the host, and remove the container when it stops

OTHER USEFUL `docker` COMMANDS

- `docker port` —List port mappings or a specific mapping for the container
- `docker container prune` —Remove all stopped containers
- `docker ps` —List containers
- `docker rmi` —Remove one or more images
- `docker rm` —Remove one or more containers
- `docker image` List images

PERSISTING DATA

OVERVIEW

- All data is written to the container's file system by default
  - Data does not persist when a container no longer exists
- Docker has two options for containers to store files on the host machine filesystem: Volumes and Bind mounts
- Docker also supports containers storing files in-memory on the host machine with `tmpfs` mounts

STORAGE OPTIONS

- Volumes
  - Stored in a part of the host filesystem managed by Docker
  - Should only be modified by Docker processes
- Bind mounts
  - Stored anywhere on the host filesystem
  - May be modified by any process (Docker and/or Non-Docker)
- In-memory file systems
  - Stored in the host system's memory

VOLUMES

- Persist data on host filesystem managed by Docker
- Use cases
  - Sharing data across multiple containers
  - Back-up, restore and migrate data between host machines
- Use `docker volume` to create and manage volumes
- Use the `-v|--volume` flag to specify volume mount
- Can be managed directly with Docker CLI

BIND MOUNTS

- Mounts directory/file on host machine in a container
- Use cases
  - Sharing configuration files from the host machine to containers
  - Sharing source code or build artifacts between and development environment on the host machine and a container
- Use the `-v|--volume` flag to specify mount points
- Cannot be managed with Docker CLI

MULTI-CONTAINER APPLICATIONS

DOCKER COMPOSE

- Compose is a tool for defining and running multi-container applications
- Compose files are defined using YAML
- The three step process:
  - Define application environment with a Dockerfile
  - Define the services that make up the application in docker-compose.yml
  - Run `docker compose up` to start the entire application
- Concise environment description

COMPOSE FILE

```yaml
version: '3.4'

services:
  api:
    build:
      dockerfile: Dockerfile
    ports:
      - "5000:80"
    depends_on:
      - db
  db:
    image: mcr.microsoft.com/mssql/server
    user: root
    volumes:
      - hello-compose:/var/opt/mssql/data
    environment:
      SA_PASSWORD: "suchSecureVeryWordSoPassW0w!"
      ACCEPT_EULA: "Y"
    ports:
```

examples/lesson-02-docker/hello-compose/docker-compose.yaml

COMMON USE CASES

- Development environments
  - Concise environment description
  - Isolated environment
- Automated testing environments
  - Set up environment for automated test suites
  - Create and destroy isolated testing environments

ORCHESTRATION

- Infrastructure as code
- Scaling, monitoring and configuration of applications in clusters and/or clouds
- Not in the scope for SW4BED-01
  - The department offers Web Architecutre og Orchestration Practice (SWWAO-01)

WRAP-UP

- What is Docker?
- Build and deploy containers
- Creating images
- Persisting data
- Multi-container applications


REMEMBER
A CONTAINER IS NEVER BETTER
THAN THE
SOFTWARE
IT IS
CONTAINING