Docker for Data Science Cheat Sheet

Advantages and Disadvantages of Containers

Role	Native Software	Virtual Machine	Docker or other container
Complexity	Low	High	Medium
Portability	Low	Medium	High
Reproducibility	Low	High	High
Startup time	Low	High	Medium
Scalability	Low	High	High
Resource Efficiency	High	Low	High
Isolation(Securty)	Low	High	Medium
Monogaing Dependecies	Low	Medium	High

Definitions

- **Docker image:** A read-only template containing all the necessary files, libraries, and dependencies required to run an application in a Docker container.
- **Docker container:** A running instance of an image. That is, an executable package including the code, libraries, and dependencies needed to run the application.
- **Dockerfile:** A script containing instructions to create the image.
- **Docker registry:** A repository for storing, sharing, and managing Docker images. These include Docker Hub, Amazon Elastic Container Registry, Microsoft Azure Container Registry, and Google Cloud Container Registry.
- **Docker Engine:** An application for managing Docker containers. It includes a server (the "Docker daemon"), a command line tool (the Docker client), and an API for other software to interact with the Docker Daemon.
- **Docker client:** A command-line tool to interact with Docker Engine to manage Docker images and containers .
- **Docker daemon (a.ka. Docker server):** A background process that manages Docker images and containers according to the commands sent from the Docker client.

Getting Help

```
# Display Docker version with docker --version
docker --version

# Display Docker system info with docker info
docker info

# Get help on Docker with docker --help
docker--help

# Get help on Docker command usage with docker {command}
docker run -help
```

Running Containers

```
# Run a container with docker run {image}
docker run hello-world  # Runs a test container to check your installation works

# Run a container then use it to run a command with docker run {image}
{command}
docker run python python -c "print('Python in Docker')" # Run Python & print
text
docker run rocker/r-base r -e "print(lm(dist~speed, cars))" # Run R & print a
model

# Run a container interactively with docker run --interactive --tty
docker run --interactive --tty rocker/r-base  # Run R interactively

# Run a container, and remove it once you've finished with docker run --
rm docker run --rm mysql  # Run MySQL, then clean up

# Run an image in the background with docker run --
detach docker run --detach postgres

# Run an image, assigning a name, with docker --name {name} run
docker run --name red1 redis # Run redis, naming the container as red1

# Run an image as a user with docker run --user {username} docker run --user
doctordocker mongo
```

Inspecting Containers

```
# List all running containers with docker ps
docker ps

# List all containers with docker ps --all
docker ps--all

# List all containers matching a conditions with docker ps --filter {key}={value}
docker ps --filter 'name=red1'

# Show container log output with docker logs --follow {container}
docker run --name bb busybox sh - "$(echo date)" # Print current datetime
docker logs --follow bb # Print what bb container printed
```

Managing Containers

```
# docker run is equivalent to docker create + docker start
# Create a container from an image with docker create {image}
# Start a container with docker start {container}
docker create --name py --interactive --tty python
docker start --interactive --attach py
# Same as docker run --name py --interactive --tty python

# Create a new image from a container with docker container commit
{container}
docker container commit

# Stop a container with docker stop {container}
# Container has option to save state or ignore request
docker stop py

# Kill a container with docker kill {container}# Container process finished
immediately
docker kill  py

# Kill and remove a container with docker rm --force {container}
docker rm --force py

# Stop then start a container with docker restart {container}
docker restart py


# Delete stopped containers with docker container prune
docker container prune

# Create an image from a container with docker container commit
{container_id} {image} # Find the container ID with docker ps --all
docker container commit 123456789abc newimage
```

Docker for Data Science Cheat Sheet

Building Images

```
# Build an image with docker build {path}
docker build


# Build a tagged image with docker build --tag {name:tag} {path} .
docker build  --tag myimage: 2023-edition


# Build an image without using the cache docker build -no-cache {path}  . docker
build --no-cache .
```

Image Management

```
# List all local images with docker images
docker images

# Show Docker disk usage with docker system df
docker system df

# Show image creation steps from intermediate layers with docker history {image}
docker history alpine

# Save an image to a file with docker save --output {filename} # Usually combined
with a compression tool like gzip
docker save julia | gzip > julia.tar.gz

# Load an image from a file with docker load --input {filename}
docker load --input julia.tar.gz

# Delete an image with docker rmi {image}
docker rmi rocker/r-base
```

Working with Registries

```
# Log in to Docker with docker login --username {username}
docker login --username doctordocker

# Pull an image from a registry with docker pull {image}
docker pull julia

# Pull a version of an image from a registry with docker pull {image}:{tag}
docker pull julia:1.8.5-bullseye

# Tag an image to a repo with docker tag {image} {user}/{repo}
docker tag python doctordocker/myrepo

# Push an image to a registry with docker push {repo_tag}
docker push doctordocker/myrepo

# Search for an image with docker search "{image-search-text}
"docker search "  "py"
```

Image Management

```
# Derive image from another image with FROM{image}
FROM ubuntu:jammy-20230301

# Set a build and runtime environment variable with ENV {name}={value}
EVN TZ ="America/New_York"

# Set a build-time variable with ARG {name}={default_value}
ARG VERBOSE=1

# Set the working directory with WORKDIR {path}
WORKDIR/home

# Switch to the user with USER {username}
USER doctordocker

# Copy a local file into the image with COPY {existing_path} {image_path}
COPY ./settings/config.yml ./settings/config.yml

# Run a shell command during the build step with RUN {command}
# \ lets commands continue across multiple lines
# && means run this command only if the preceding command succeeded
RUN apt-get update \
      && install -y libxml2-dev ,

# Run a shell command on launch with CMD ["{executable}", "{param1}"]
# Each Dockerfile should only have 1 CMD statement
 CDM [ python" "-i] # Start Python interactively
```