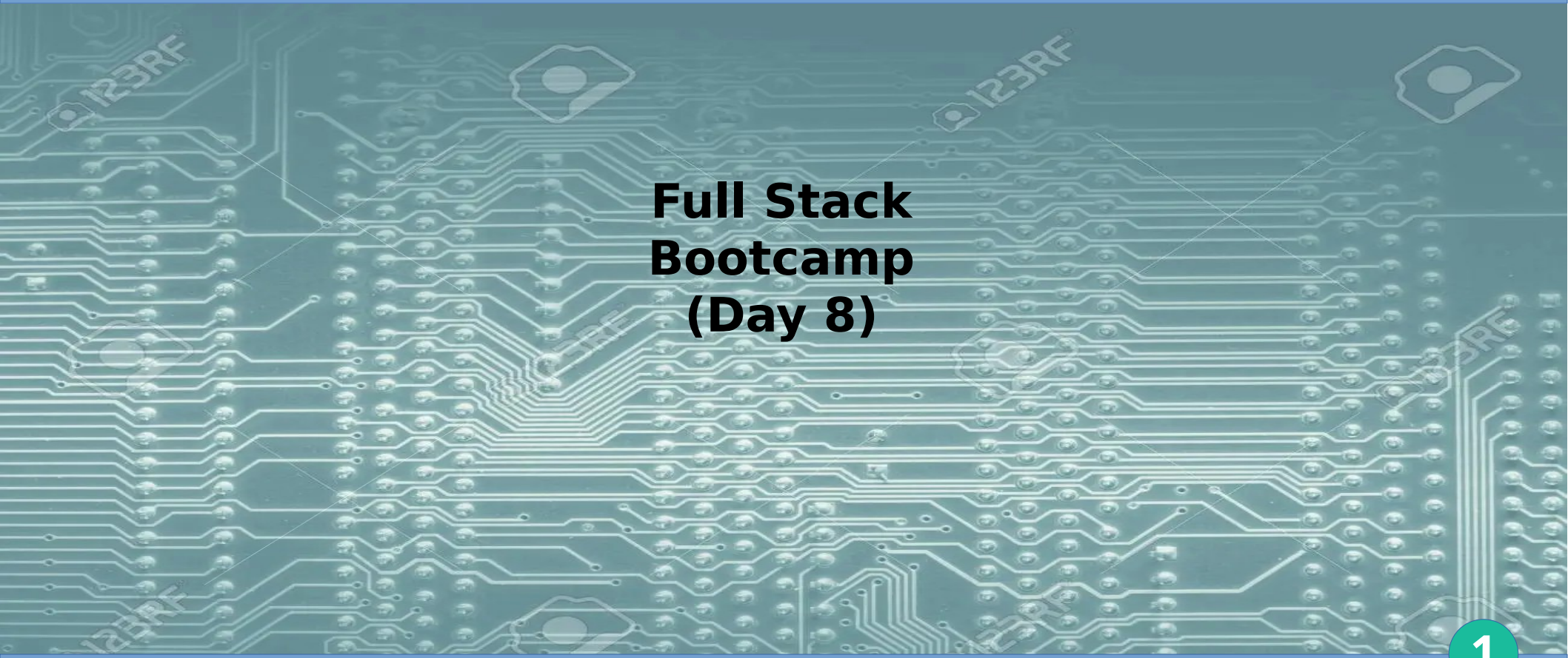
Docker Fundamentals



Agenda

- Day 1
  - ES6+
- Day 2
  - React Native
- Day 3
  - Angular
- Day 4
  - Springboot
  - SpringData
- Day 5
  - JSON
  - NoSQL
- Day 6
  - Relational
- Day 7
  - Junit
  - Mockito
- Day 8
  - Docker
- Day 9
  - Kubernetes
- Day 10
  - Images and tips

Virtual machines

- 70s and 80s mainframes (Virtual spaces)
- 80s Unix user management
- 90s OS/2 Virtual machines (DOS and Windows emulation)
- VmWare, VirtualBox, DOSBox

Containers

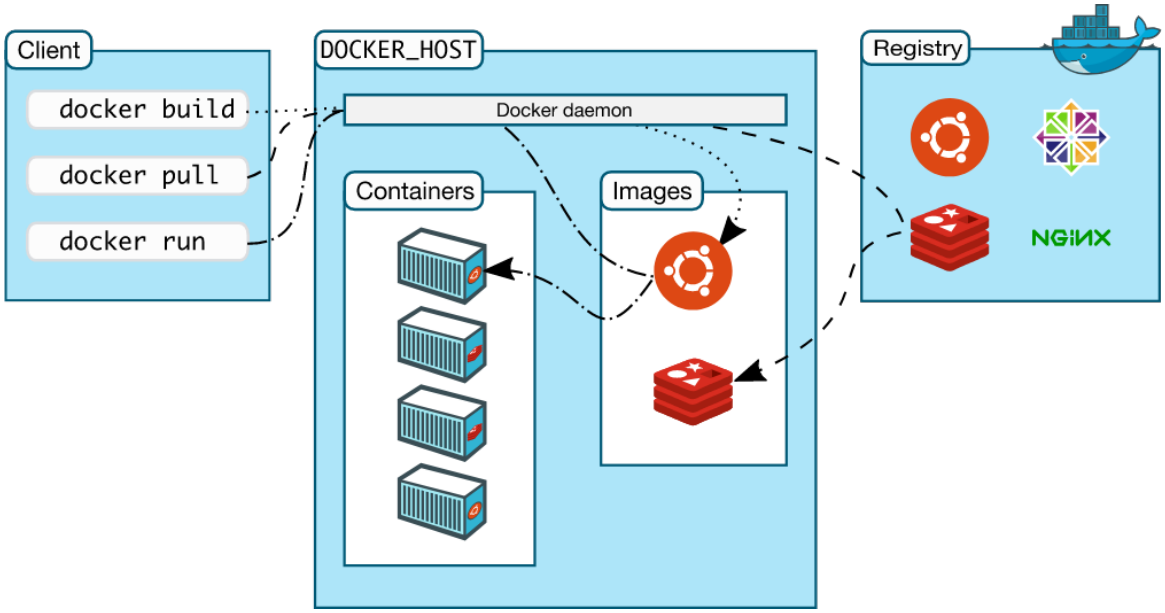
- A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

WSL

- The Windows Subsystem for Linux lets developers run a GNU/Linux environment, including most command-line tools, utilities, and applications, directly on Windows, unmodified, without the overhead of a traditional virtual machine or dualboot setup. You can:
- Choose your favorite GNU/Linux distributions from the Microsoft Store.
- Run common command-line tools such as grep, sed, awk, or other ELF-64 binaries.
- Run Bash shell scripts and GNU/Linux command-line applications including:
  - Tools: vim, emacs, tmux
  - Languages: NodeJS, Javascript, Python, Ruby, C/C++, C# & F#, Rust, Go, etc.
  - Services: SSHD, MySQL, Apache, lighttpd, MongoDB, PostgreSQL.
- Install additional software using your own GNU/Linux distribution package manager.
- Invoke Windows applications using a Unix-like command-line shell.
- Invoke GNU/Linux applications on Windows.

Docker Architecture

- Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.



Setup

- Your Windows machine must meet the following requirements to successfully install Docker Desktop.
- Requirements:
  - WSL 2 backend
  - Hyper-V backend and Windows containers
- Run the proper installer for the Docker Desktop that applies for your system
  - Mac installation is different if Mac or Intel chip is used

Download

- https://docs.docker.com/desktop/install/windows-install/ for windows
- https://docs.docker.com/desktop/install/mac-install/ for Mac
- https://docs.docker.com/desktop/install/linux-install/ (be aware that Ubuntu users can use apt-get install docker or snap install docker, other distributions could use yum or other tool)

Installing

- The following hardware prerequisites are required to successfully run WSL 2 on Windows 10 or Windows 11:
  - 64-bit processor with Second Level Address Translation (SLAT)
  - 4GB system RAM
  - BIOS-level hardware virtualization support must be enabled in the BIOS settings. For more information, see Virtualization.
- Download and install the Linux kernel update package.

Importance of Linux

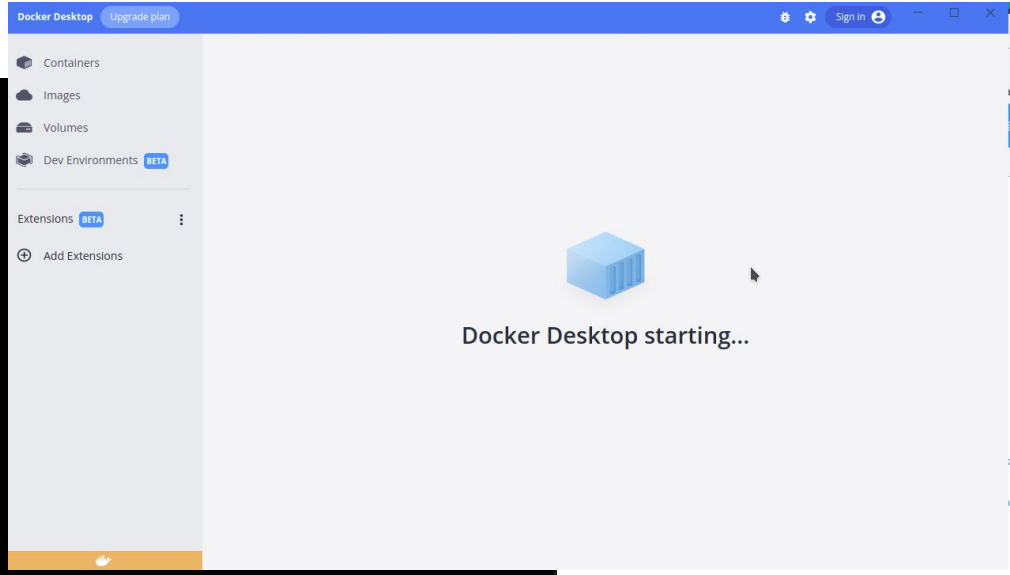
- Linux came from the UNIX family Operating Systems
- UNIX has been the most used servers for the last 40 years
- Linux is the most widely used OS,even in final user market:
  - As of April 2022, Android, an operating system using the Linux kernel, is the world's most-used operating system when judged by web use. It has 43% of the global market, followed by Windows with 30%, Apple iOS with 17%, macOS with 6%, then (desktop) Linux at 0.98% also using the Linux kernel.
- Most cloud solutions were born in Linux

starting Docker

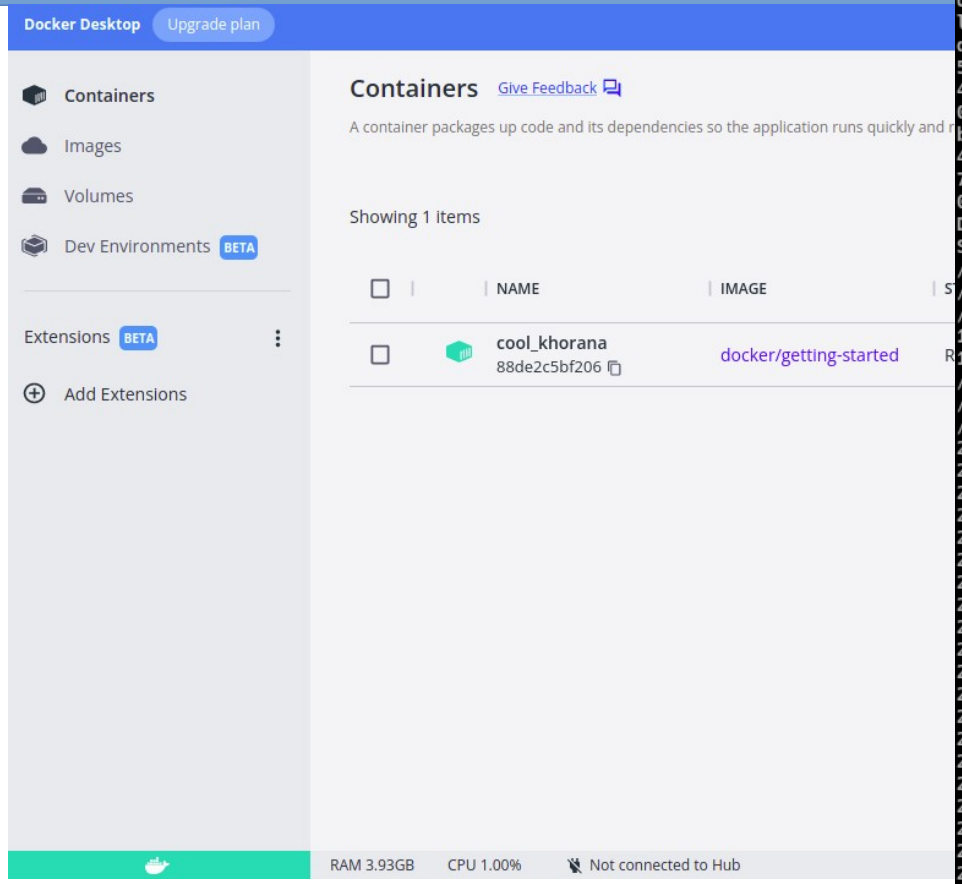
- Run the command:
- `docker run -d -p 80:80 docker/getting-started`
- You'll notice a few flags being used. Here's some more info on them:
  1. `-d` - run the container in detached mode (in the background)
  2. `-p 80:80` - map port 80 of the host to port 80 in the container
  3. `docker/getting-started` - the image to use

Dashboard

```
charly@worker01:~$ gpg --generate-key
gpg (GnuPG) 2.2.19; Copyright (C) 2019 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Nota: Usa "gpg --full-generate-key" para el diálogo completo de generación de clave.

GnuPG debe construir un ID de usuario para identificar su clave.

Nombre y apellidos: charly
Dirección de correo electrónico: cdelunasaenz@yahoo.com.mx
Ha seleccionado este ID de usuario:
    "charly <cdelunasaenz@yahoo.com.mx>"

¿Cambia (N)ombre, (D)irección o (V)ale/(S)alir? V
Es necesario generar muchos bytes aleatorios. Es una buena idea realizar
alguna otra tarea (trabajar en otra ventana/consola, mover el ratón, usar
la red y los discos) durante la generación de números primos. Esto da al
generador de números aleatorios mayor oportunidad de recoger suficiente
entropía.
Es necesario generar muchos bytes aleatorios. Es una buena idea realizar
alguna otra tarea (trabajar en otra ventana/consola, mover el ratón, usar
la red y los discos) durante la generación de números primos. Esto da al
generador de números aleatorios mayor oportunidad de recoger suficiente
entropía.
gpg: clave 2492489C777DB626 marcada como de confianza absoluta
gpg: certificado de revocación guardado como '/home/charly/.gnupg/openpgp-revocs.d/C765BBFF6AEA5F80BC9DE4B62492489C777DB626.rev'
claves pública y secreta creadas y firmadas.

pub   rsa3072 2022-07-18 [SC] [caduca: 2024-07-17]
      C765BBFF6AEA5F80BC9DE4B62492489C777DB626
uid                      charly <cdelunasaenz@yahoo.com.mx>
sub   rsa3072 2022-07-18 [E] [caduca: 2024-07-17]

charly@worker01:~$ pass init C765BBFF6AEA5F80BC9DE4B62492489C777DB626
mkdir: se ha creado el directorio '/home/charly/.password-store/'
Password store initialized for C765BBFF6AEA5F80BC9DE4B62492489C777DB626
```

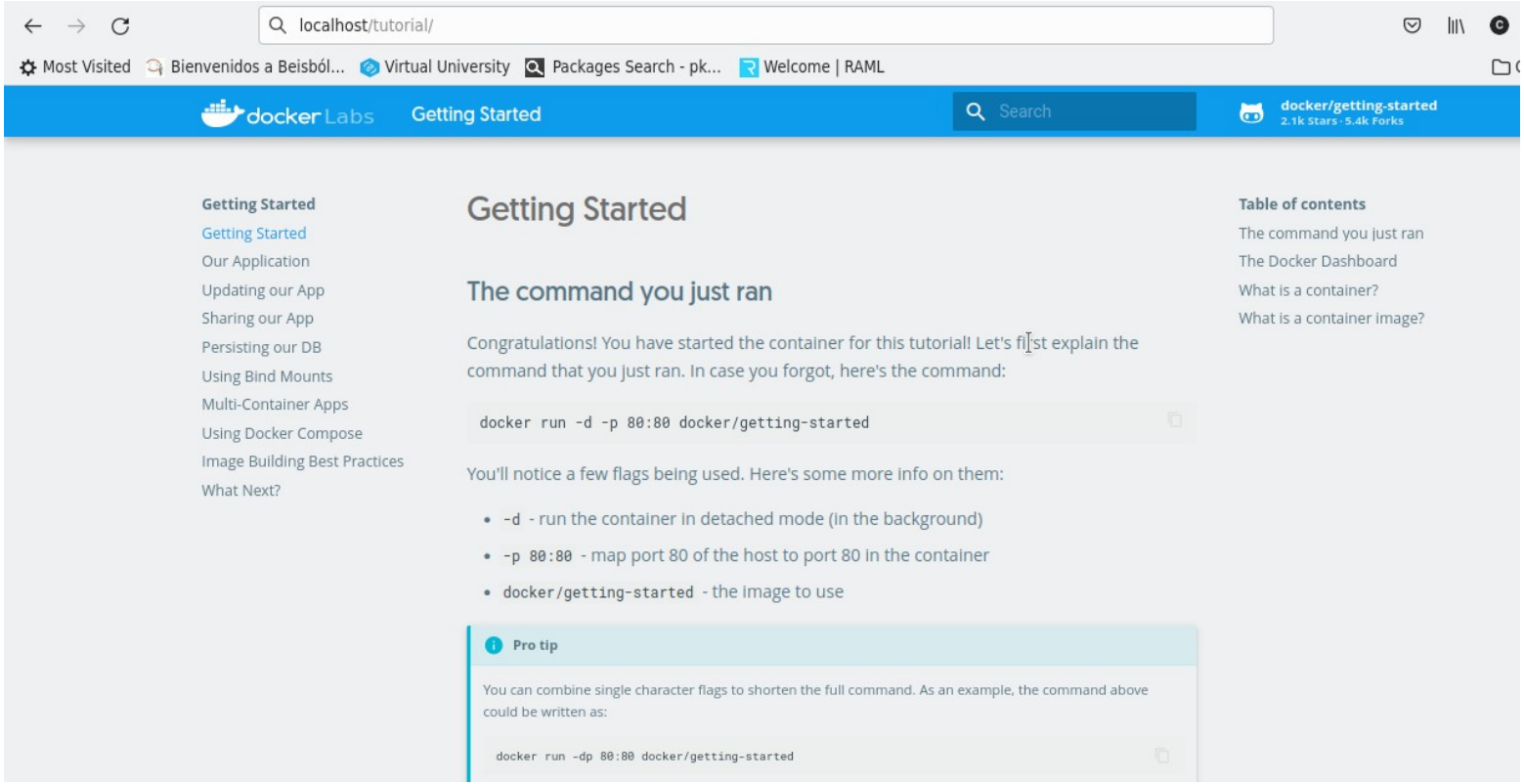


Dashboard



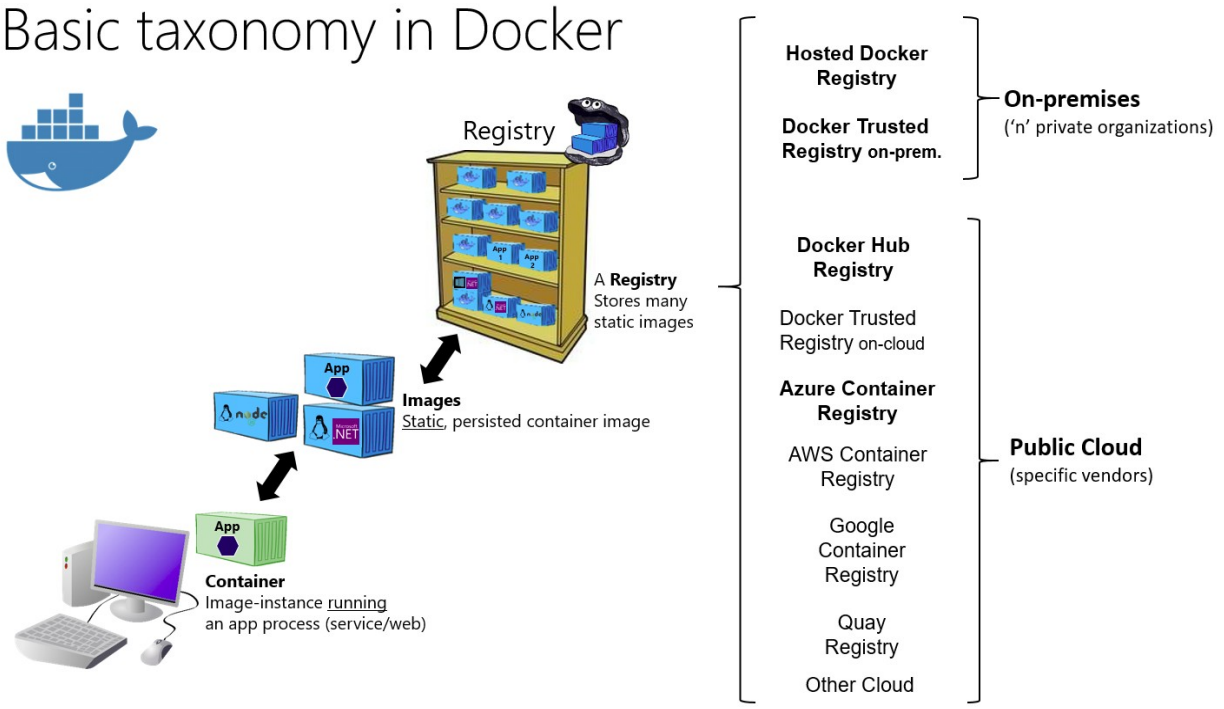

```
charly@worker01:~$ docker run docker/getting-started
Unable to find image 'docker/getting-started:latest' locally
latest: Pulling from docker/getting-started
df9b9388f04a: Pull complete
5867cba5fcbd: Pull complete
4b639e65cb3b: Pull complete
061ed9e2b976: Pull complete
bc19f3e8eeb1: Pull complete
4071be97c256: Pull complete
79b586f1a54b: Pull complete
0c9732f525d6: Pull complete
Digest: sha256:b558be874169471bd4e65bd6eac8c303b271a7ee8553ba47481b73b2bf597aae
Status: Downloaded newer image for docker/getting-started:latest
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2022/07/18 00:48:31 [notice] 1#1: using the "epoll" event method
2022/07/18 00:48:31 [notice] 1#1: nginx/1.21.6
2022/07/18 00:48:31 [notice] 1#1: built by gcc 10.3.1 20211027 (Alpine 10.3.1_git20211027)
2022/07/18 00:48:31 [notice] 1#1: OS: Linux 5.10.104-linuxkit
2022/07/18 00:48:31 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2022/07/18 00:48:31 [notice] 1#1: start worker processes
2022/07/18 00:48:31 [notice] 1#1: start worker process 32
2022/07/18 00:48:31 [notice] 1#1: start worker process 33
2022/07/18 00:49:18 [notice] 1#1: signal 28 (SIGWINCH) received
2022/07/18 00:49:18 [notice] 1#1: signal 28 (SIGWINCH) received
2022/07/18 00:49:18 [notice] 1#1: signal 28 (SIGWINCH) received
2022/07/18 00:49:18 [notice] 1#1: signal 28 (SIGWINCH) received
2022/07/18 00:49:19 [notice] 1#1: signal 28 (SIGWINCH) received
2022/07/18 00:49:19 [notice] 1#1: signal 28 (SIGWINCH) received
2022/07/18 00:49:19 [notice] 1#1: signal 28 (SIGWINCH) received
2022/07/18 00:49:19 [notice] 1#1: signal 28 (SIGWINCH) received
2022/07/18 00:49:19 [notice] 1#1: signal 28 (SIGWINCH) received
2022/07/18 00:49:19 [notice] 1#1: signal 28 (SIGWINCH) received
2022/07/18 00:49:19 [notice] 1#1: signal 28 (SIGWINCH) received
2022/07/18 00:49:20 [notice] 1#1: signal 28 (SIGWINCH) received
```

Containers


localhost/tutorial/
Most Visited
Bienvenidos a Beisból...
Virtual University
Packages Search - pk...
Welcome | RAML
docker Labs
Getting Started
Search
docker/getting-started
2.1k Stars · 5.4k Forks
Getting Started
Getting Started
Our Application
Updating our App
Sharing our App
Persisting our DB
Using Bind Mounts
Multi-Container Apps
Using Docker Compose
Image Building Best Practices
What Next?
Getting Started
The command you just ran
Congratulations! You have started the container for this tutorial! Let's first explain the command that you just ran. In case you forgot, here's the command:
docker run -d -p 80:80 docker/getting-started
You'll notice a few flags being used. Here's some more info on them:
-d - run the container in detached mode (in the background)
-p 80:80 - map port 80 of the host to port 80 in the container
docker/getting-started - the image to use
Pro tip
You can combine single character flags to shorten the full command. As an example, the command above could be written as:
docker run -dp 80:80 docker/getting-started
Table of contents
The command you just ran
The Docker Dashboard
What is a container?
What is a container image?

Images

- A Docker image is a file used to execute code in a Docker container. Docker images act as a set of instructions to build a Docker container, like a template. Docker images also act as the starting point when using Docker. An image is comparable to a snapshot in virtual machine (VM) environments.

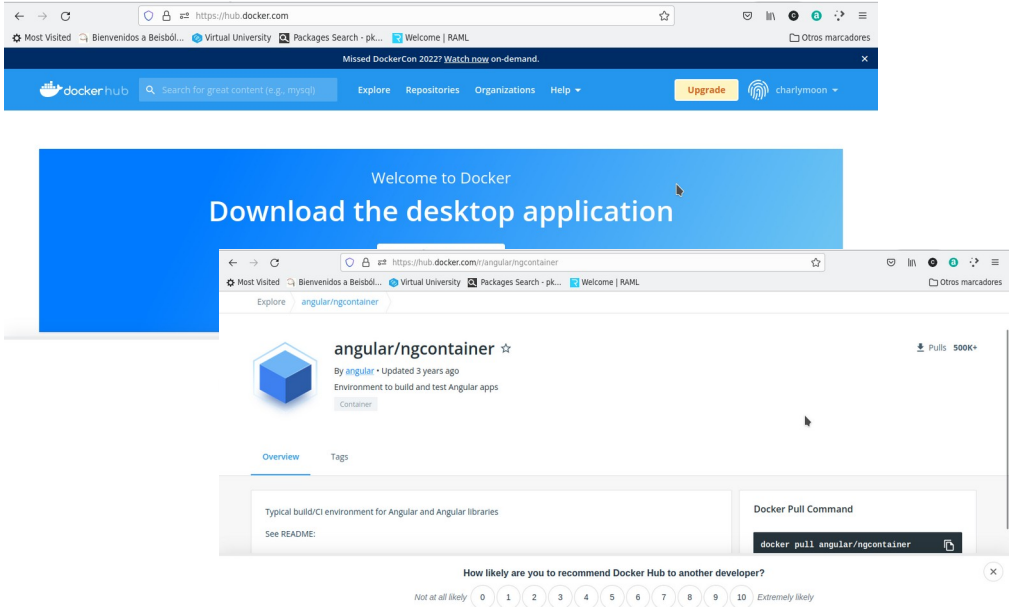


Exercise 1

- Installation and first image

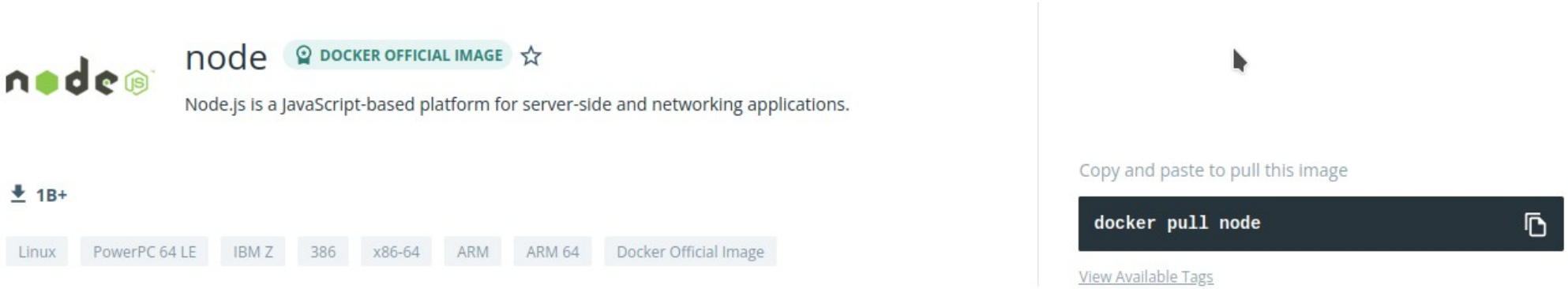
Install Angular

- Login to docker hub
- Search for Angular
- Run the **`docker pull angular/ngcontainer`**



Install React

- Download NodeJS image
- Modify Image
- ¡You have React!
- React already have an image but with android specific features based



Customizing images

- Generate configuration files
  - Docker file
- Build Docker image
  - `docker build -t MyAppAndImageTitle DirectoryForDockerFile`
- Pull docker image
  - Build will do it by default
- Run image
  - The already known `docker run` command
- Review the status
  - `docker ps` or use the docker desktop

docker file

- FROM
  - Specifies what docker image will be used as base
- ENV
  - Will set an environment variable
- COPY
  - Will copy local files to the image
- RUN
  - Will run commands on the image
- # for comments
- CMD
  - Sets an array of strings with a shell command and the parameters
- EXPOSE
  - Documents which port does the application in the image uses

Sample Applications

- Store application on Angular
- Calculator in React
- Baseball database bank

Exercise 2

- Container access

Updating the app

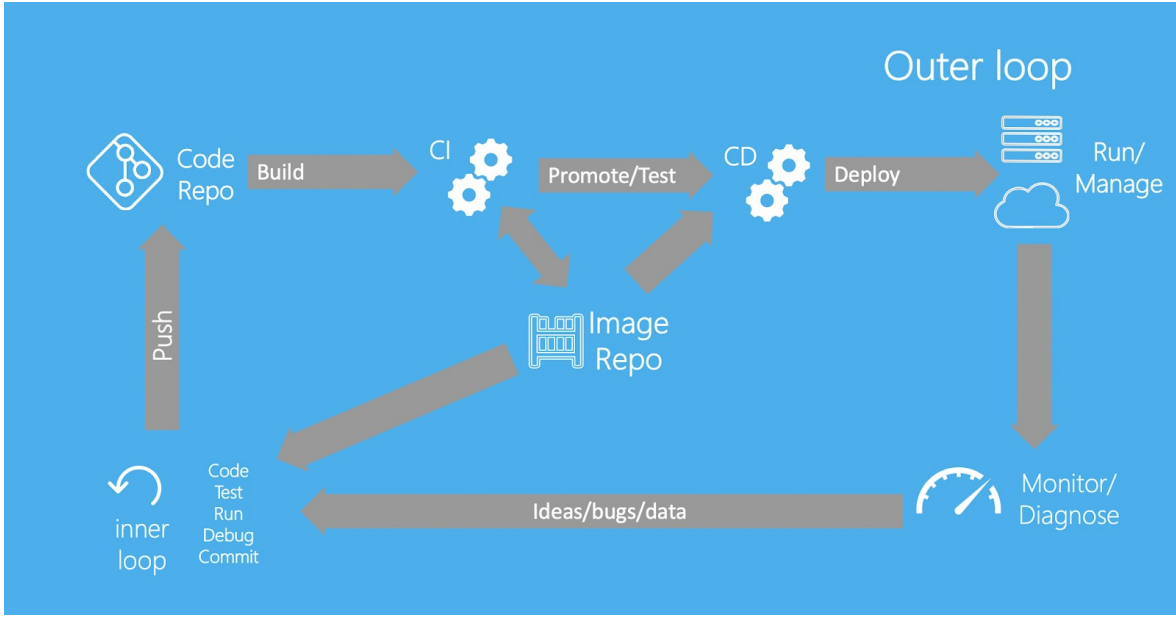
- Stateless images
- Rebuild your images
- Keep your persistence volumes

Sharing the app

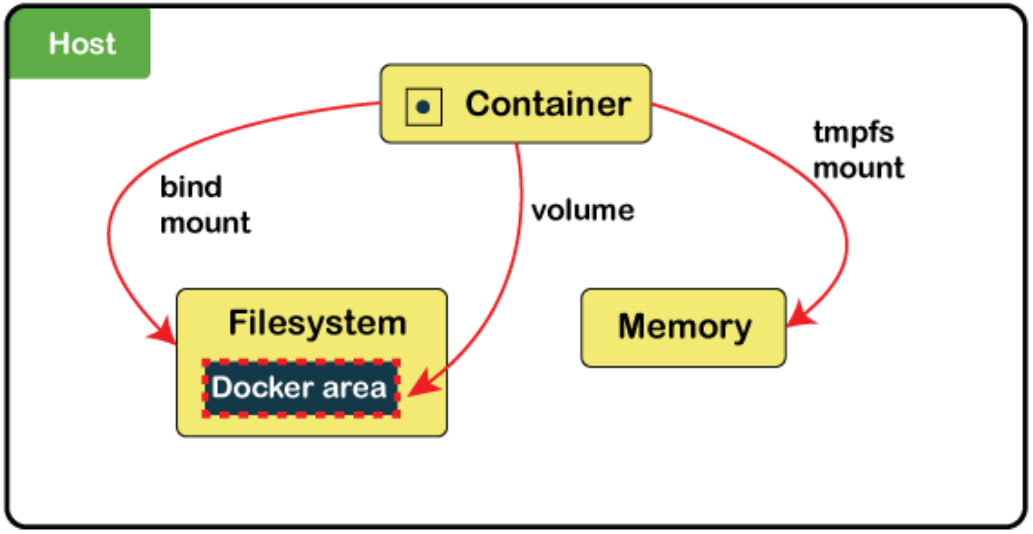
- Deployment
- Jenkins
  - docker compose
  - Deployed on containers managers
- ¿Docker Hub?

CI/CD and Docker

- To get started, one of the most important things when working with Docker and any CI/CD is to understand when you need to test with the CI, and when you can do this locally. At Docker, we think about how developers work in terms of their inner loop (code, build, run, test) and their outer loop (push changes, CI build, CI test, deployment).



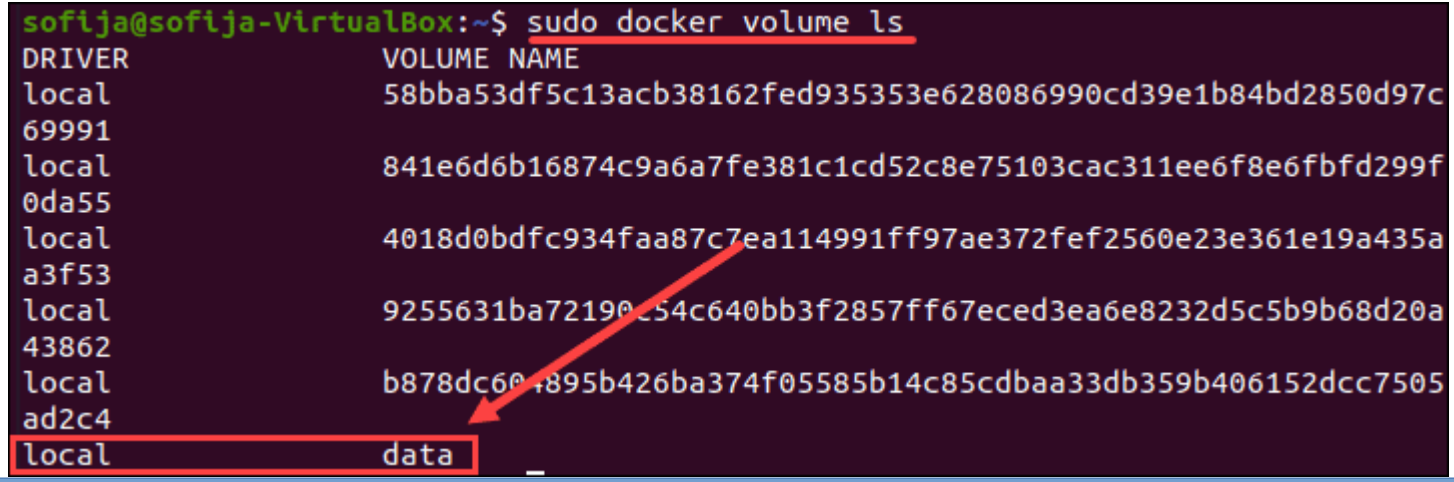
Persistence



- **Volumes**
  - To create a Docker Volume use the command:
  - **`docker volume create [volume_name]`**
  - Docker automatically creates a directory for the volume on the host under the `/var/lib/docker/volume/ path`. You can now mount this volume on a container, ensuring data persistence and data sharing among multiple containers.
  - For example, to create a volume under the name data, you would run the command:
  - **`docker volume create data`**

Volumes

- List Docker Volumes
  - To verify you have successfully created a Docker volume, prompt Docker to list all available volumes with:
  - **`docker volume list`**
- The output displays a list of volumes, specifying their location (DRIVER) and their VOLUME NAME. In the image below, you can see the volume data created in the previous section.



Volumes

- Inspecting Docker Volumes
  - To see more information about a Docker volume, use the inspect command:
  - **docker volume inspect [volume_name]**
  - It lists details of a volume, including its location on the host file (Mountpoint). Everything stored within the data volume can also be found in the directory listed under the mountpoint path.

• Mounting a Data Volume

- To mount a data volume to a container add the --mount flag to the docker run command. It adds the volume to the specified container, where it stores the data produced inside the virtual environment.
- To run a container and mount a data volume to it, follow the basic syntax:
- **`docker run --mount source=[volume_name],destination=[path_in_container] [docker_image]`**
- Replace [path_in_container] with the path where you want to place the data volume in the container. Everything stored in that directory automatically gets saved on the data volume on the host as well.

Mounting a Host Directory as a Data volume

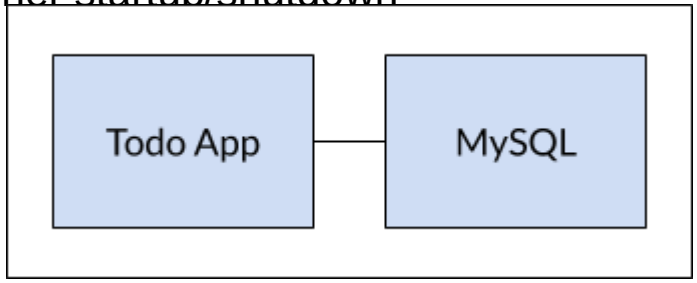
- You can also mount an existing directory from the host machine to a container. This type of volume is called Host Volumes.
- You can mount host volumes by using the -v flag and specifying the name of the host directory.
- Everything within the host directory is then available in the container. What's more, all the data generated inside the container and placed in the data volume is safely stored on the host directory.
- The basic syntax for mounting a host directory is:
- `docker run -v "$(pwd)":[volume_name] [docker_image]`
- The "$(pwd)" attribute instructs Docker to mount the directory the user is currently in.

Exercise 3

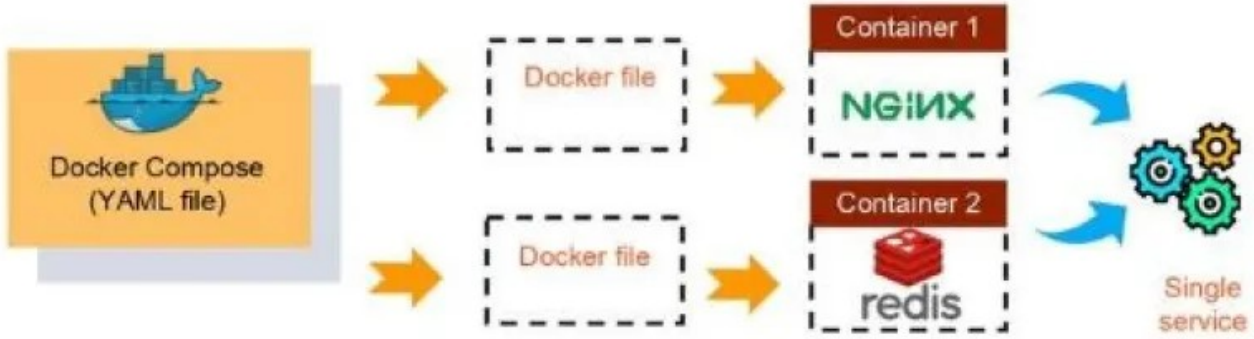
- Create a MySQL image and volume

Multicontainer application

- Up to this point, we have been working with single container apps. But, we now want to add MySQL to the application stack. The following question often arises - “Where will MySQL run? Install it in the same container or run it separately?” In general, each container should do one thing and do it well. A few reasons:
  - There’s a good chance you’d have to scale APIs and front-ends differently than databases
  - Separate containers let you version and update versions in isolation
  - While you may use a container for the database locally, you may want to use a managed service for the database in production. You don’t want to ship your database engine with your app then.
  - Running multiple processes will require a process manager (the container only starts one process), which adds complexity to container startup/shutdown



Docker compose and multicontainer




```
version: '3'
services:
  web:
    image: nginx
    ports:
    - 8080:80

  database:
    image: redis
```

version ‘3’: This denotes that we are using version 3 of the Compose file.
N This section defines all the different containers we will create. In the above example, we have two services, web and database.
web and database: These are the names of our services. Containers will be created with the names we provide.
ports: This is used to map the container’s ports to the host machine.
image: To run a service using a pre-built image, we specify the image location using the image clause.

Docker networks

- There are two ways to put a container on a network:
  - Assign it at start or
  - connect an existing container.
- For now, we will create the network first and attach the MySQL container at startup.
  - Create the network.
  - `docker network create todo-app`

Starting a container within a network

- Start a MySQL container and attach it to the network. We're also going to define a few environment variables that the database will use to initialize the database (see the "Environment Variables" section in the MySQL Docker Hub listing).
- docker run -d \
- --network todo-app --network-alias mysql \
- -v todo-mysql-data:/var/lib/mysql \
- -e MYSQL_ROOT_PASSWORD=secret \
- -e MYSQL_DATABASE=todos \
- mysql:5.7
- If you are using Windows then use this command in PowerShell.
- PS> docker run -d `
- --network todo-app --network-alias mysql `
- -v todo-mysql-data:/var/lib/mysql `
- -e MYSQL_ROOT_PASSWORD=secret `
- -e MYSQL_DATABASE=todos `
- mysql:5.7

Exercise 4

- Baseball DB

Image Building best practices

- Docker file and docker-compose to stablish parameters instead command line
- Use docker network to communicate between your apps
- Use a container manager additional to docker such as kubernetes, Openshift or Pivotal Cloud Foundry
- Use one container just for one application or set of services (APIs)
- Use already made containers as a base for your applications
- Keep nomenclature standards for all your containers and images
- Use volumes just in case there are necesarly
- Do not share persistence sources among different applications
- Do security scanning (**`docker scan`**)

Docker Hub limit

- Docker Hub limits the number of Docker image downloads (“pulls”) based on the account type of the user pulling the image. Pull rates limits are based on individual IP address. For anonymous users, the rate limit is set to 100 pulls per 6 hours per IP address. For authenticated users, it is 200 pulls per 6 hour period. Users with a paid Docker subscription get up to 5000 pulls per day. If you require a higher number of pulls, you can also purchase an Enhanced Service Account add-on.
- Some images are unlimited through our Open Source and Publisher programs.

Security scanning

- Docker has partnered with Snyk to provide the vulnerability scanning service.
- You must be logged in to Docker Hub to scan your images.
- Run the command **`docker scan --login`**, and then scan your images using **`docker scan <image-name>`**
- For example, to scan the getting-started image you created earlier in the tutorial, you can just type
- **`docker scan getting-started`**

References

- https://docs.docker.com/language/ For specific language features
- https://docs.docker.com/desktop/ Manuals
- https://docs.docker.com/reference/ Reference on cli and API
- https://docs.docker.com/get-started/overview/ guidance on creating containers

Day 7 summary

- Docker overview
- Configuration
- Image creation
- Image customization
- Multicontainer applications