MLOps: How to efficiently deploy and maintain machine learning solutions

Hello!

I am Hugo Leal

IT2 - Backend Data platform

You can find me at **Github, LinkedIn:** @hugolardosa

Email: hugolardosa@ua.pt

Before we start

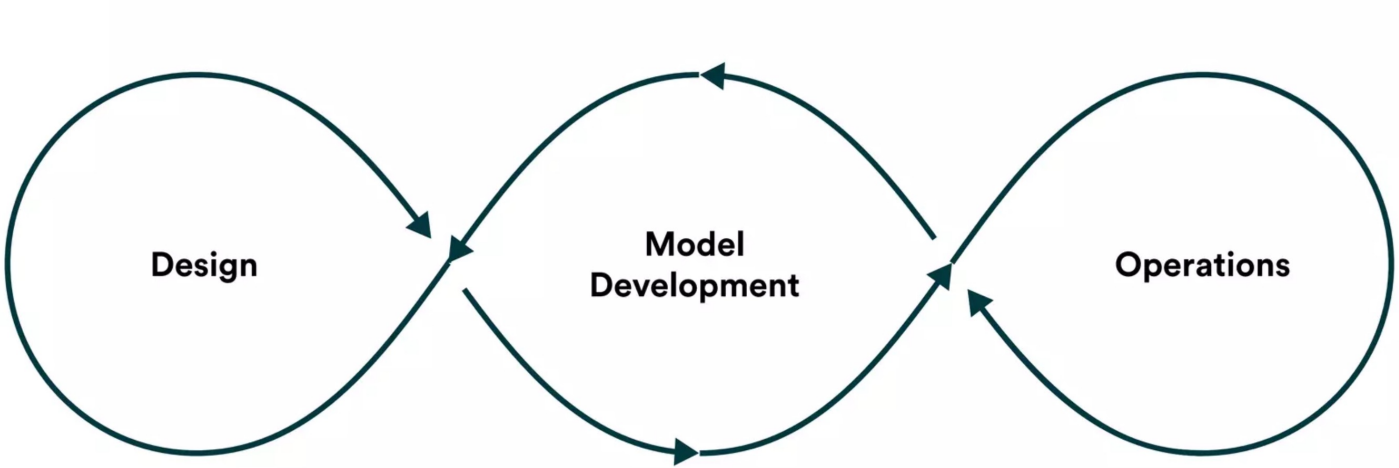
Let's check if everybody has docker installed

Any problems?

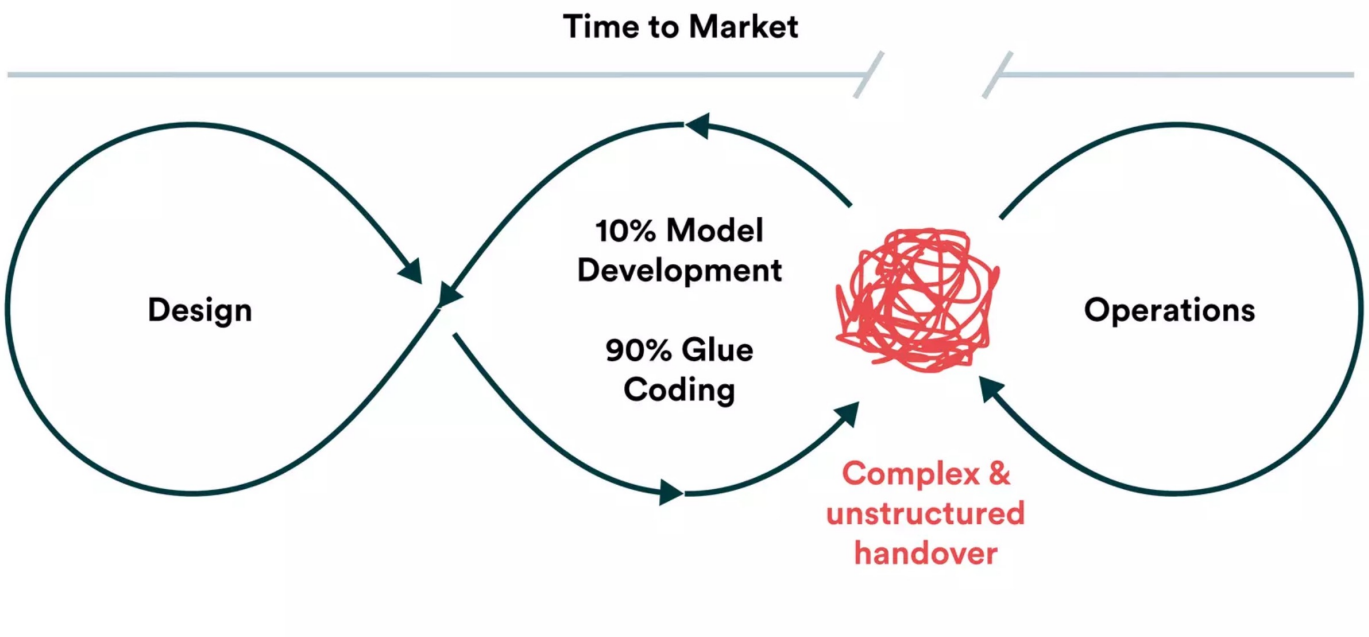
1.
What is MLOps

Machine Learning Operations

What ML development should look like



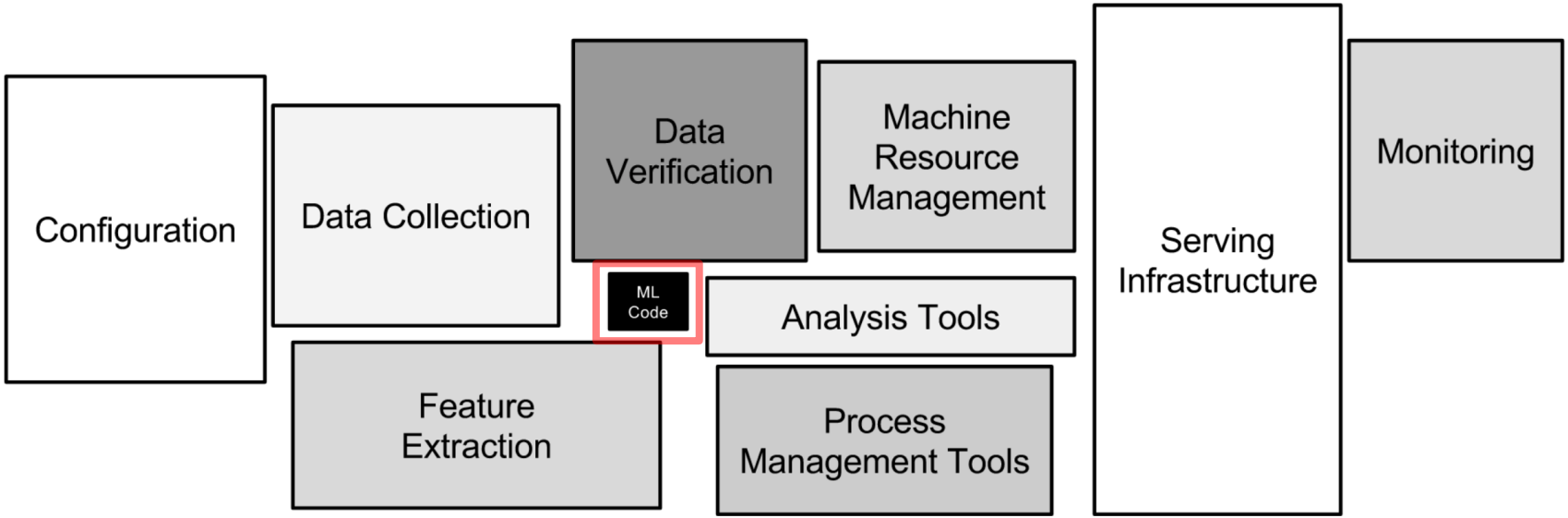
What it actually looks like



Why does this happen?

- We need to prove that our machine learn code has value in a project or a company

Because a machine learning system is more than the ML solution



“

The goal of MLOps is to **reduce technical friction** to get the model from an idea into production in the shortest possible time with as little risk as possible.

What ML development should look like

- MLOps is not a single tool or platform.
- MLOps is about agreeing to do ML the right way and then supporting it.

A few shared principles will take you a long way..

Process

ML should be collaborative	ML should be reproducible	ML should be continuous	ML should be tested & monitored
Shared Infrastructure	Versioning and Code Data and Metadate	Machine Learn Pipelines	Model Deployment and Monitoring

Tooling

Avoid the "It's too early for MLOps" trap.

Short term thinking

Time

Do it fast

Risk

Do it safe

Long term thinking

Risk

Do it right

Time

So it will be faster next time

2.
Docker

The main tool on DevOps/MLOps

What is Docker?

- Ever you ever used a virtual machine? Like virtualbox, VMWare?
  - These technologies emulate a computer and run an OS

With docker, a containered solution you have a isolation layer from the host OS

… Don't worry this will make more sense during the workshop

Docker

- **Core Concept:** Container
  - App level construct (unrelated to infrastructure)
  - One Container runs One Application (which can spawn child apps)

- Containers pack an application to run in any underlying hardware
  - including configurations, dependencies, auxiliary data, etc...

Docker Concepts

- **Image:** the data of a container (application, libraries, images, etc...)
- **Container:** A running instance of an application.
  - Composed by one or more images. Each image adds a specific functionality
- **Engine:** Software the executes commands for containers
- **Registry:** Repository of docker images
- **Control Plane:** infrastructure to manage containers and images

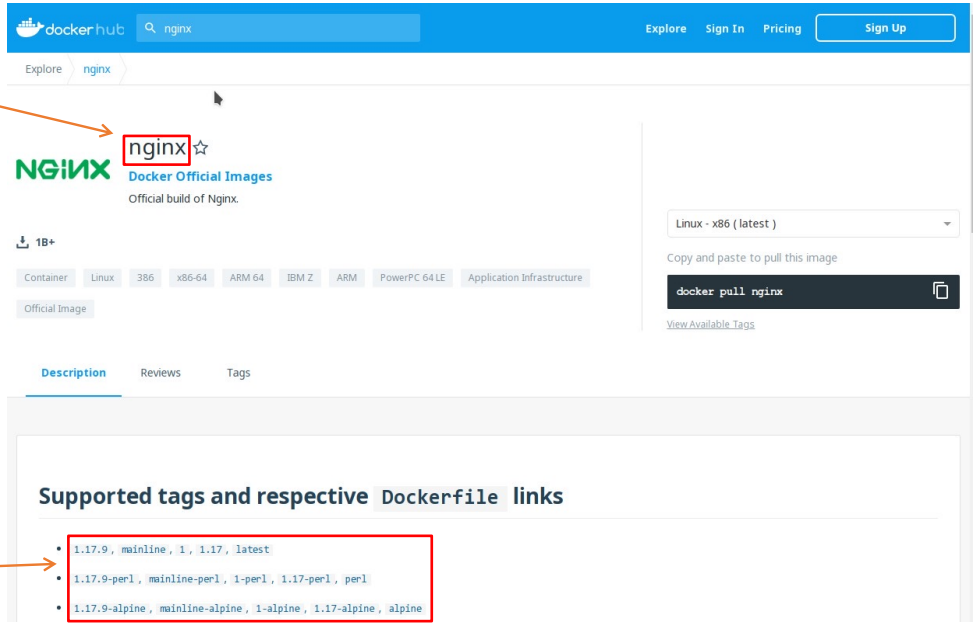
Docker Registry

- Central repository to store and deliver images
  - indexed by name and tag
  - can be private to an infraestruture or public (Docker Hub)
- Clients push images to the registry
  - Local client
- Docker Engine pulls images and executes the container
  - Running on servers
- Layers are hashed and indexed allowing reuse
  - Pulling an image only requires the layers that do not exist locally

Docker Hub – Largest Public Regsitry

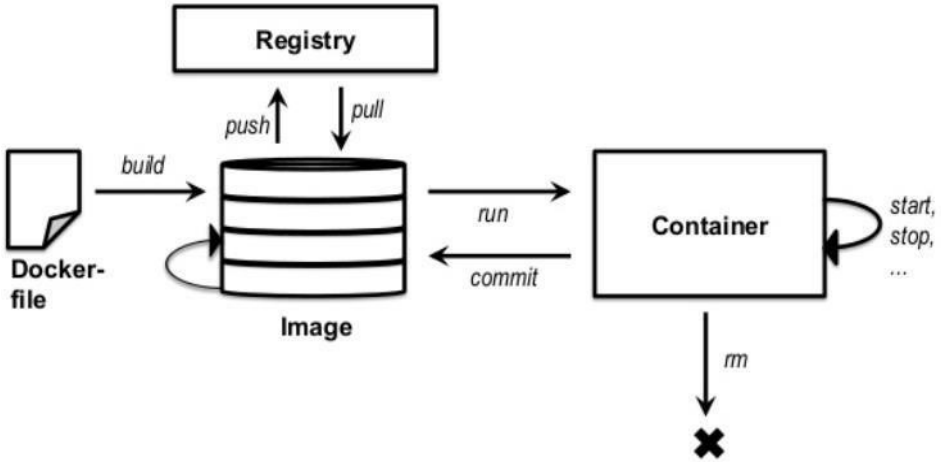
Name

Tags



Workflow

- Search docker image
  - $ docker search nginx
- Pull docker image
  - $ docker pull nginx
- List local docker images
  - $ docker image ls
- Run container
  - $ docker run -d --name frontend-server nginx



Docker file

- **Sequence of commands** to prepare the container for execution
  - Lists dependencies
  - Lists commands to be executed inside container
  - Sets the command to execute on start
- Used locally or distributed in registry
  - Public registry: Docker hub
- May define versions for same software
  - nginx:latest, nginx:perl, nginx:alpine...

```
FROM debian:stretch-slim

LABEL maintainer="NGINX Docker Maintainers <docker-maint@nginx.com>"

ENV NGINX_VERSION 1.15.9-1~stretch
ENV NJS_VERSION     1.15.9.0.2.8-1~stretch

RUN set -x \
        && apt-get update \
        && apt-get install --no-install-recommends --no-install-
suggests -y gnupg1 apt-transport-https ca-certificates \
        && \




...OMMITED...

RUN ln -sf /dev/stdout /var/log/nginx/access.log \
        && ln -sf /dev/stderr /var/log/nginx/error.log


EXPOSE 80

STOPSIGNAL SIGTERM

CMD ["nginx", "-g", "daemon off;"]
```

Images (I)

- Containers have their initial data in images
  - Contain data, applications, libraries
  - Contain an entry point to be executed when the container is initiated
- Composed by multiple layers
  - A base image
  - Several images, changing the content (with differences)
  - Exploits overlayfs (a unionfs similar to AUFS)
    - Can use others like btrfs, ZFS

Images (II)

- Read Only and Non persistent
  - Serve as templates for containers to start
  - Multiple containers will use the same images (containers are ephemeral)

- It's important to keep images small
  - Dont: base your image on a full blown distro (e.g. debian)
  - Do: base your image on a small distro (e.g. Alpine)

Containers – Important!

Containers do not store anything, they are volatile

We need to create volumes!

Container persistence – Bind

- Bind mounts:
  - mount an existing path into a path inside the container
  - !!! – Use an absolute path with docker run

```
docker run  -d -v /app:/var/www --name frontend-server nginx
```

Container persistence – Bind

- Bind mounts:
  - mount a specific storage object (managed by docker) at a given path
  - volumes are not deleted with containers

```
$ docker run -d\ -v app:/var/www:ro --name frontend-server nginx
```

Container Network Ports

- Services inside container cannot provide services to other applications
  - No network services by default
- Run command must expose ports
  - Outside ports mapped to inside ports
- E.g. map external port 8000 to internal (inside container) port 80

```
$ docker run -d -p 8000:80 --name frontend-server nginx
```

Sensitive data

- Frequently, there is sensitive data that must be available to containers
  - Database/ext service access credentials, certificates, admin credentials
  - Sensitive data is container specific, or shareable among several containers

- Direct (**no so correct**) approach: Build images with credentials
  - must build different images for each credential pair
  - must rebuild images when credentials change
  - credentials frequently end in the teams source control (e.g. Git repository)

Docker Compose

- Compose services created from multiple containers
  - that is… consider the specification of a complete service, and not
  - docker-compose.yml file

- Example:
  - specifies service name: nginx
  - container name and build ref
  - volumes to add
    - uses an environmental variable
  - ports to expose to outside

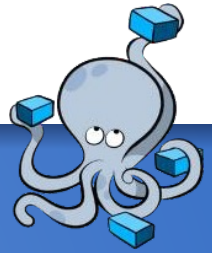
```
version: 3

services:
    nginx:
        container_name: nginx
        build:
            context: ./nginx
            dockerfile: Dockerfile
        volumes:
              app:/var/www
              ${NGINX_LOG_PATH}:/var/log/nginx
        ports:
            - 80:80
            - 443:443
```

Sensitive data: Docker Secrets

- Allows creating blobs of data with restricted access
  - Exist independently of the containers
  - Allow changing the secret without changing the container
    - ex: database access creds for dev, stagging and prod
  - Only accessed inside specific containers
    - at /run/secrets/secret_name

- Manage secret: docker secret create/inspect/list/rm
- Add to service: docker service... --secret secret_name
  - Will expose secret_name inside container

Docker Compose




```
version: 3
services:
    nginx:
        container_name: nginx
        build:
            context: ./nginx
            dockerfile: Dockerfile
        ...
        secrets:
            key_file:
                file: server-key.pem
            cert_file:
                file: server-cert.pem
```

3.
Kubernets

Container Orchestration

Kubernetes

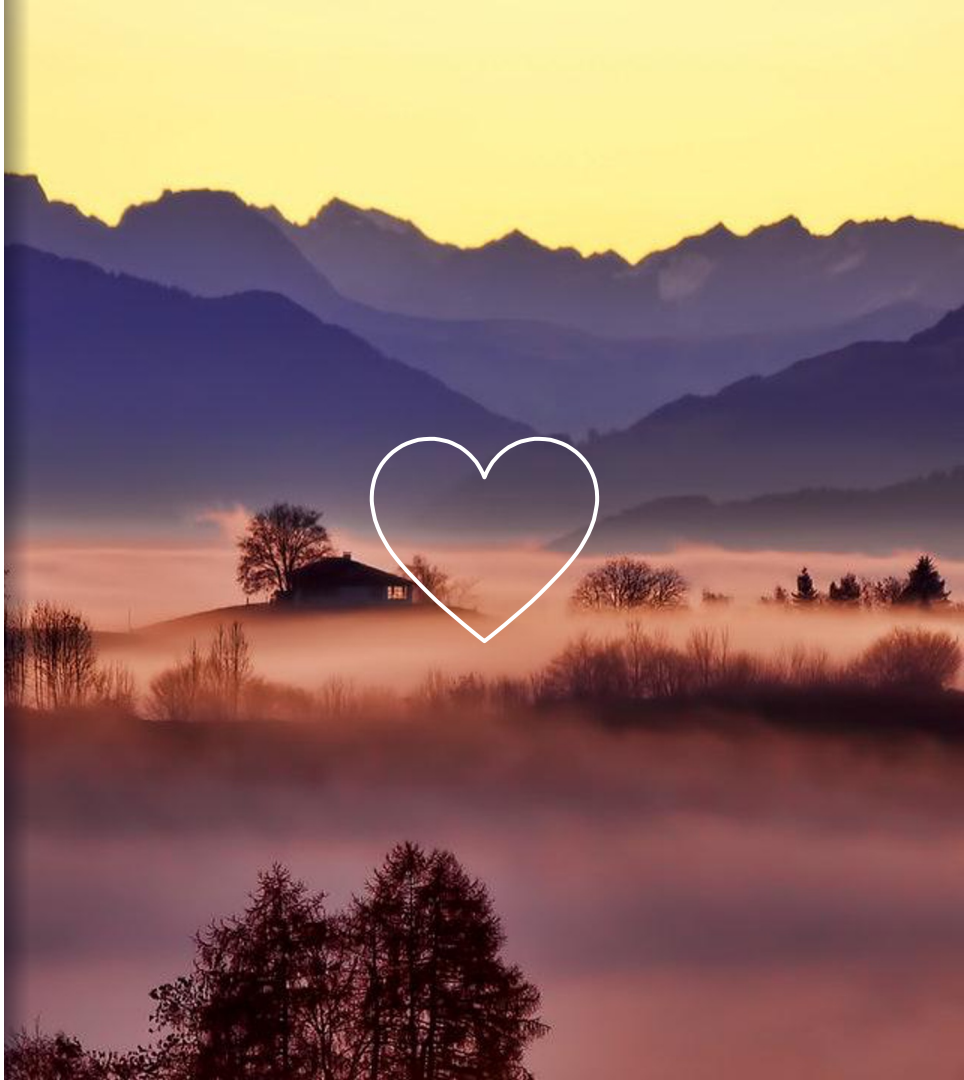
- Just an honorable mention to kubenetes… But let's start with docker
- One thing at a time :)

Thanks!

Any questions?

You can find me at:

- @hugolardosa
- hugolardosa@ua.pt



Credits

- Presentation template by SlidesCarnival
- Photographs by Unsplash

What is MLOps Henrik Skogstrorm
https://www.slideshare.net/shyssee/what-is-mlops

Virtualizations Approaches (docker) – Prof. João Barraca, UA-DETI