RT0904 – Cloud Programming

Episode 2 – Hiding the mess

Luiz Angelo Steffenel – RT0904

Recall on (micro)services

Services

- Services are software design approaches to transform an application in something less monolythical and more manageable/upgradable

- Definition of Services (**opengroups.org**):

  - *It logically represents a business activity with a specified outcome*
  - *It is self-contained*
  - *It is a black box for its consumers*
  - *It may consist of other underlying services*

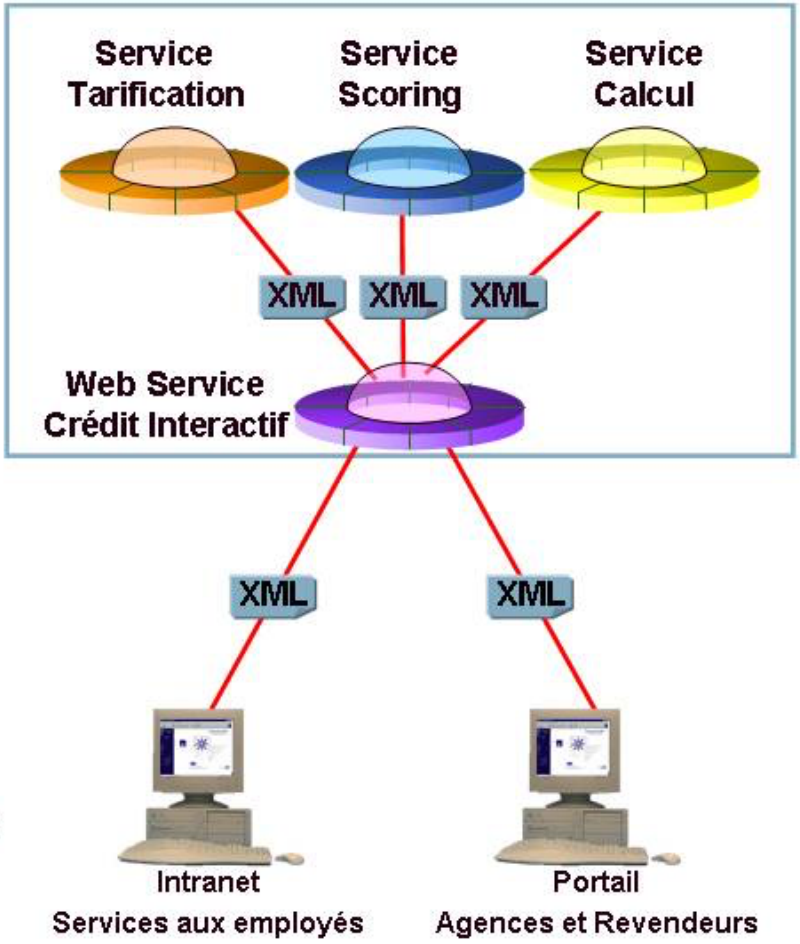
- In the 2000's, SOA (Service Oriented Architectures) and WebServices have become a buzzword

La définition du WebService

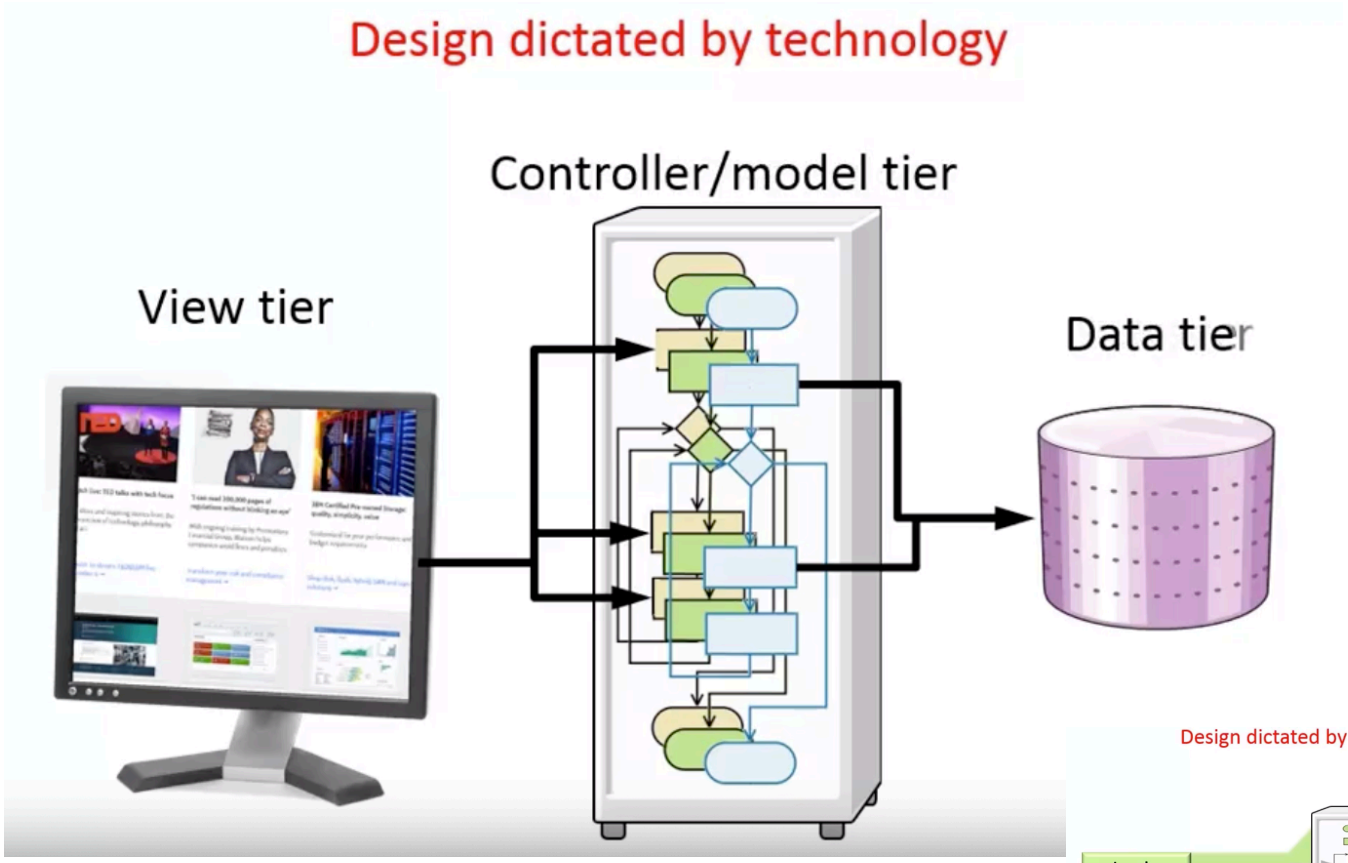
"A Web service is a software application or component that can be accessed over the Internet using a platform/language-neutral data interchange format to invoke the service and supply the response, using a rigorously defined message exchange pattern, and producing a result that is sufficiently well-defined to be processed by a software application."

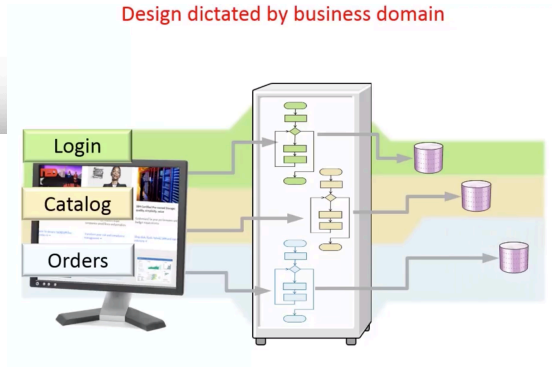
W3C Architecture domain

- Les Web Services sont:
  - des composants métiers
  - auto suffisants
  - auto descriptifs
  - exécutés sur Internet ou Intranet
  - respectant des contrats de qualité



Services main idea



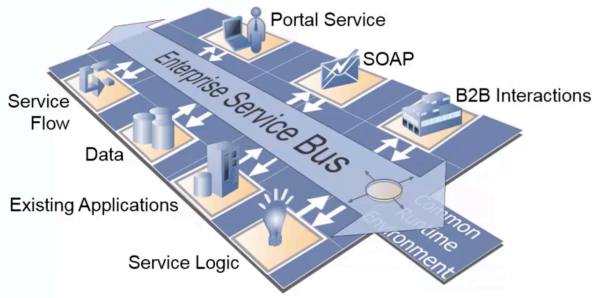


Source: IBM Training / Coursera

Drawbacks of 2000's Services

- Adoption of Services was impacted by some factors
  - Not easy to break monolithic applications into loosely tied services
    - Internal dependencies are often strong
  - Too much focus on Webservices
    - It's only one possibility
  - Most services platforms are "heavy"
    - Ex: Java Tomcat
  - Service discovery – never worked really
    - + no clear business model

- At the end, we get mostly a multi-layered application around an ESB



THE EVOLUTION OF

SOFTWARE ARCHITECTURE

1990's

SPAGHETTI-ORIENTED
ARCHITECTURE
(aka Copy & Paste)

2000's

LASAGNA-ORIENTED
ARCHITECTURE
(aka Layered Monolith)

2010's

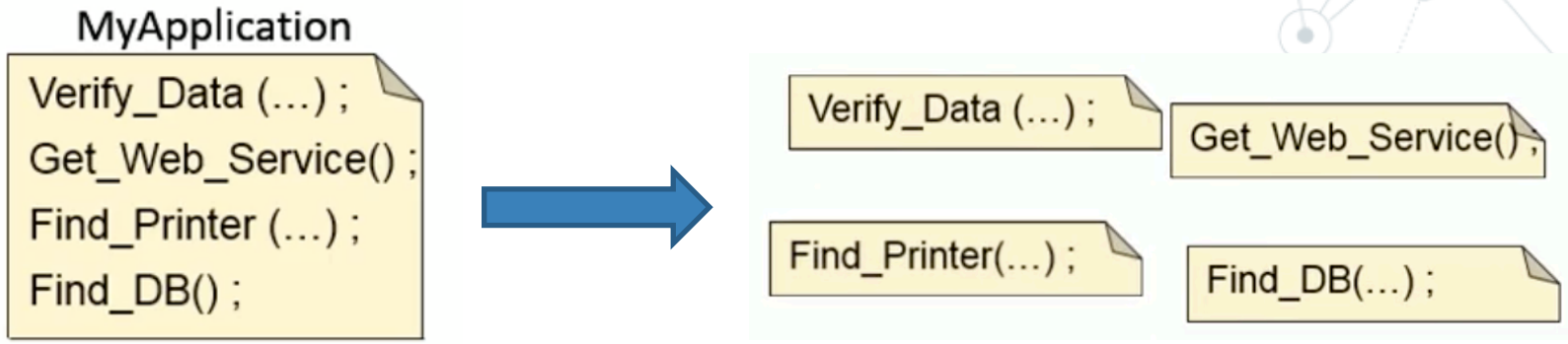
RAVIOLI-ORIENTED
ARCHITECTURE
(aka Microservices)

Source: @benorama

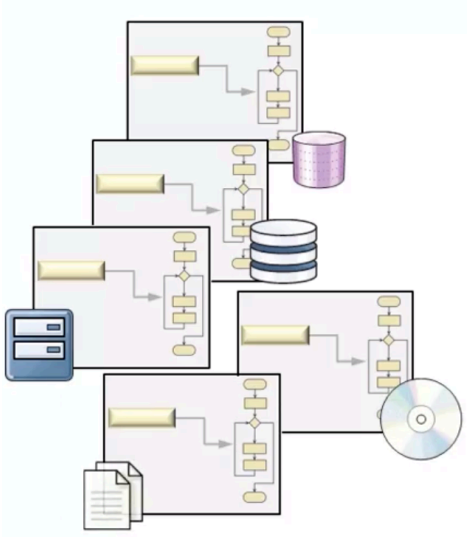
Microservices

- Approach focused on the decomposition of applications
  - single-function modules
  - totally independent
  - Communicate through well-defined interfaces
  - Operated by small teams that manage the entire lifecycle
  - Minimize communication
  - Reduce the scope of change
- Virtualization adds some extras
  - Simplified scalability/downsizing
  - CI/CD (Continuous Integration, Continuous Delivery)

Single Function, Totally Independent



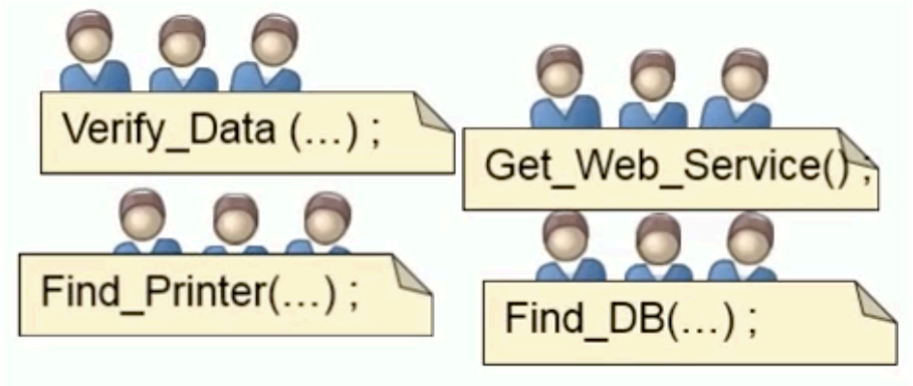
Ideally, each Microservice is fully independent from each other (even if they can be "linked" at runtime)



Well-defined interfaces, Small teams

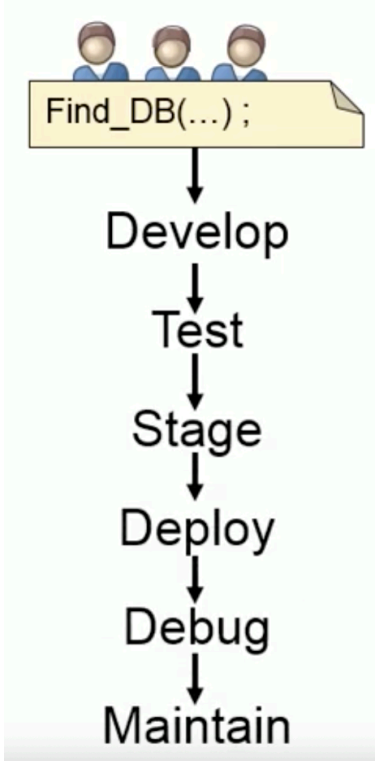
- With Single functions, the interfaces (API) are mostly stable
  - We know
    - the require parameters (IN)
    - the expected output (OUT)
  - Standard communication protocols
    - HTTP/REST for queries, JSON for data
  - Independent of the implementation language

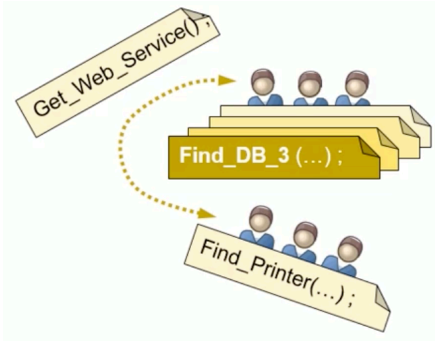
- Small teams keep things simple -> **the Pizza strategy**
  - *If you can't feed a team with two pizzas, it's too large. That limits a task force to five to seven people, depending on their appetites — Jeff Bezos*



Lifecycle, communication, scope of change

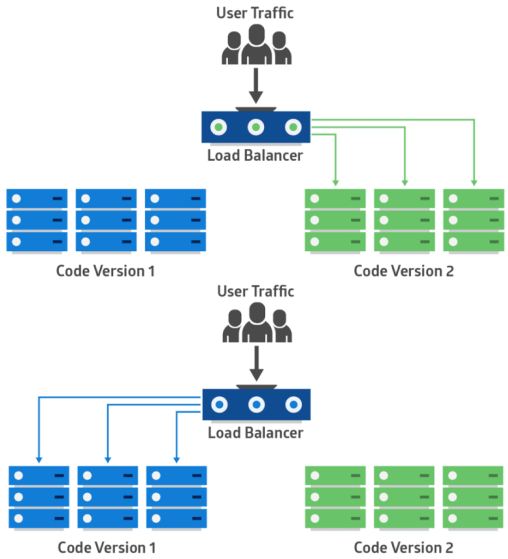
- A small team becomes responsible for all microservice lifecycle
  - Development
  - Testing
  - Deployment
  - Upgrade
- Communication is simplified
  - **Agile** inside a team
  - Interaction with other is minimized
- Each microservice can be upgraded on its own pace
  - Interfaces keep compatibility
  - No need for a huge recompiling/ redeployment interruption
  - Continuous integration (CI/CD)

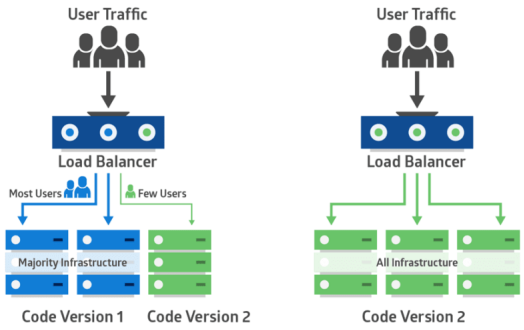




A note about CI/CD

- Virtualization (containers) and Microservices make CI/CD easier
- Different strategies:
  - **Big Bang (major redeployment)**
  - **Rolling (gradual deployment)**
  - **Blue-Green, Red-Black, A/B**
  - **Canary**





Do I need to be a DevOps?

- Microservices belong in a DevOps environment
- The development team is responsible for the code in all its lifecycle
  - Who knows how to launch the service?
  - Who knows how to scale the service? And its limitations?
  - Who will get the logs and check for problems?
- In this course we will only get some hints on how to deploy/control services with Kubernetes

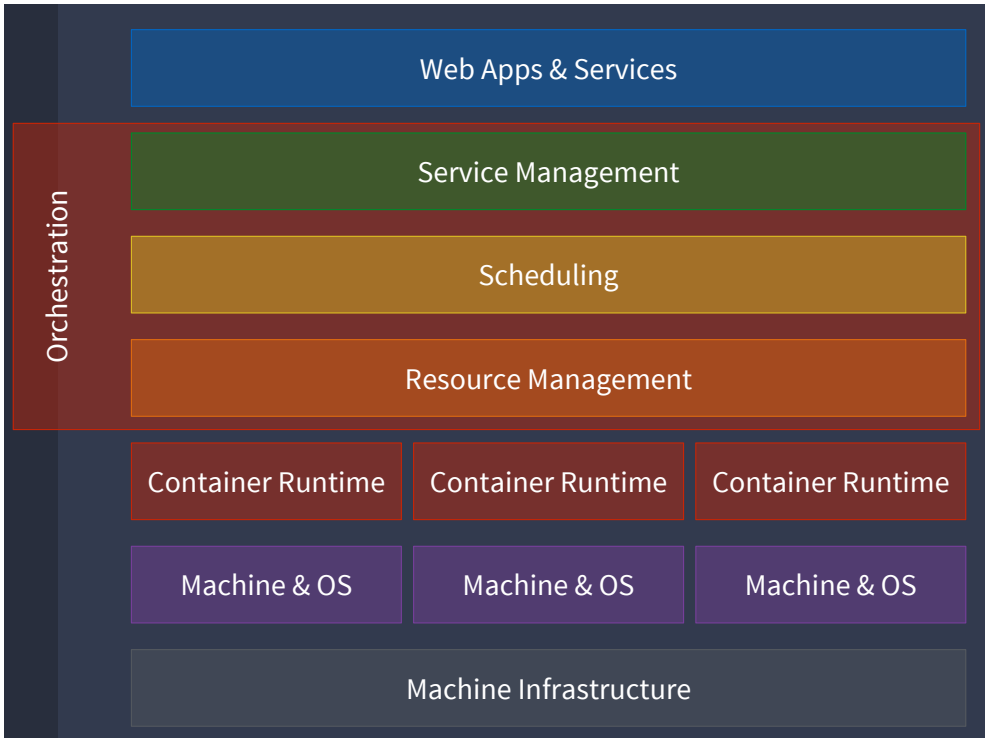
Orchestrators

What has Docker brought to us?

- Isolation and Security
  - Containers help isolate each part of your system and provides better control of each component of your system
- Run anything, anywhere
  - All languages, all databases, all operating systems
  - Any distribution, any cloud, any machine
- Reproducibility
  - Reduces the times we say “it worked on my machine”
- But this is not enough...

Orchestration 101

- “The automated arrangement, coordination, and management of complex computer systems, middleware and services." - Wikipedia



Functional Capabilities

- SCHEDULING
  - Placement
  - Replication/Scaling
  - Resurrection
  - Rescheduling
  - Rolling Deployment
  - Upgrades
  - Downgrades
  - Collocation
- RESOURCE MANAGEMENT
  - Memory
  - CPU
  - GPU
  - Volumes
  - Ports
  - IPs
- SERVICE MANAGEMENT
  - Labels
  - Groups/Namespaces
  - Dependencies
  - Load Balancing
  - Readiness Checking

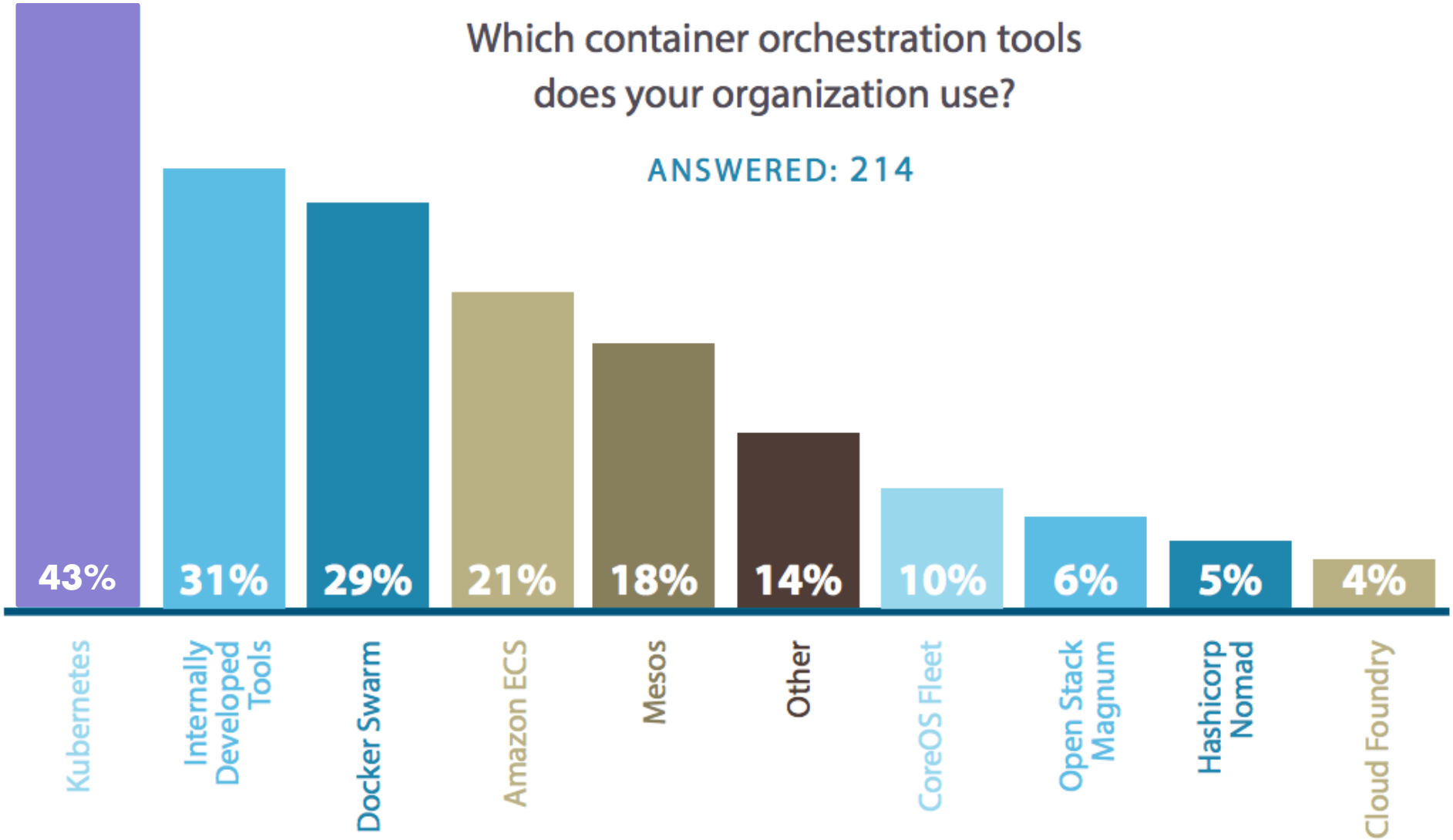
Non-Functional Qualities

- SCALABILITY
  - Performance
  - Responsiveness
  - Efficiency
- AVAILABILITY
  - Fault Tolerance
  - Robustness
  - Reliability
  - Resilience
  - Disaster Recovery
- FLEXIBILITY
  - Format Support
  - Portability
  - Interoperability
  - Extensibility
- USABILITY
  - Familiarity
  - Maintainability
  - Compatibility
  - Debuggability
- PORTABILITY
  - Container Runtimes
  - Host OS
  - Hosted
  - Cloud
  - Bare-Metal
- SECURITY
  - Auditability
  - Secrets Management
  - Encryption
  - Isolation

The competition started a few years ago

- Several competitors try to answer the orchestration needs for container execution
- Examples :
  - Amazon ECS (Orchestration)
  - **Docker Swarm** (Container Orchestration)
  - Kontena (Container Orchestration)
  - **Kubernetes** (Orchestration Platform)
  - Openshift (PaaS, based on Kubernetes)
  - DC/OS (Distributed Operating System)
- All them have good features, but one orchestrator is becoming the *de facto* environment :

KUBERNETES



Source: devops.com and clusterHQ survey 2016

Kubernetes ~~Container~~ Engines

Direct deployment of containers on the cloud

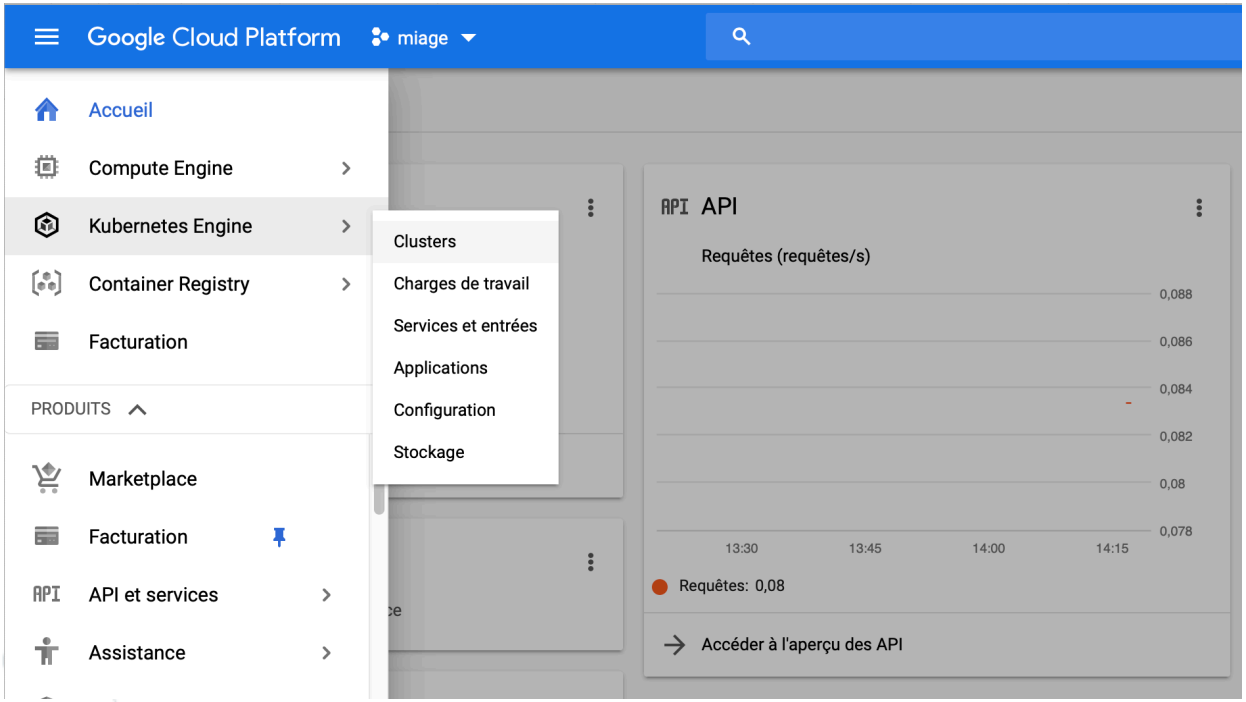
Setting free from IaaS

- Containers are becoming too important; several applications rely on containers instead of classical IaaS
- One can launch a cloud IaaS instance and install all softwares for Docker, but we miss the main point

- Several cloud providers now offer services to run containers directly
  - They "hide" the fact that they start VMs for us
  - The VMs they start are better managed than our IaaS

Using the Google Cloud

- First, you need to connect to google cloud console and accessing the project the teacher provided you
- https://console. cloud.google.com



Google Container Engine

- Select "Kubernetes Engine" in the left menu
- Click on the "Google Cloud Shell" icon in the upper right

- Now a terminal window will open below

Starting a cluster

- To simplify, let's do some pre-configuration
  - `export PROJECT_ID=` *PROJECTID*
    - You can find your *PROJECTID* on your prompt
      lsteffenel@**project-1-185121**:~$
  - `gcloud config set project $PROJECT_ID`
  - `gcloud config set compute/zone us-central1-b`

- Create a container cluster
  - `gcloud container clusters create` **`cluster-name`**
  - This action takes some time (2-5 minutes)

- Check if the cluster has been fully launched
  - `gcloud compute instances list`
  - → Your machine must have a public IP

Now let's import a docker image

- The terminal has most docker commands already available to build your own image
- In this example, we will retrieve an image stored in Docker hub
- Problem: we cannot run it directly
  - It must be on the Google repository
  - Solution: let's pull our image, retag it and then push to Google

```
docker login

docker pull lsteffenel/flask-trek

docker tag lsteffenel/flask-trek  gcr.io/${PROJECT_ID}/flask-trek

docker push gcr.io/${PROJECT_ID}/flask-trek

gcloud container images list
```

```
lsteffenel@cloudshell:~ (docker-miage)$ docker pull lsteffenel/flask-app
Using default tag: latest
latest: Pulling from lsteffenel/flask-app
cc5efb633992: Pull complete
e3590229229e: Pull complete
12e306f5327e: Pull complete
18b041126c9b: Pull complete
8f838a7d6dea: Pull complete
90fc8f167838: Pull complete
36b2c0596361: Pull complete
Digest: sha256:6393bb66219bea60ecc7c9e09ae5d649b50b32194f7a3d83318f8860376fd0ae
Status: Downloaded newer image for lsteffenel/flask-app:latest
lsteffenel@cloudshell:~ (docker-miage)$ docker tag lsteffenel/flask-app gcr.io/${PROJECT_ID}/flask-app
lsteffenel@cloudshell:~ (docker-miage)$ docker push gcr.io/${PROJECT_ID}/flask-app
The push refers to repository [gcr.io/docker-miage/flask-app]
79cf62976d7e: Pushed
4599be7213d3: Pushed
e8714cec6df0: Pushed
e54e36595a91: Pushed
29aa93171082: Pushed
f8c4b51a2b00: Pushed
51951fc332eb: Pushed
latest: digest: sha256:6393bb66219bea60ecc7c9e09ae5d649b50b32194f7a3d83318f8860376fd0ae size: 1781
lsteffenel@cloudshell:~ (docker-miage)$ gcloud container images list
NAME
gcr.io/docker-miage/flask-app
Only listing images in gcr.io/docker-miage. Use --repository to list images in other repositories.
lsteffenel@cloudshell:~ (docker-miage)$
```

Deploy your app

- Google uses Kubernetes to deploy images to a cluster
  - We will study Kubernetes in detail in a few slides

```
kubectl run flask-app --image=gcr.io/${PROJECT_ID}/flask-trek --port 80
```

```
lsteffenel@cloudshell:~ (docker-miage)$ kubectl run flask-app --image=gcr.io/${PROJECT_ID}/flask-app --port 80
deployment.apps "flask-app" created
lsteffenel@cloudshell:~ (docker-miage)$ kubectl get pods
NAME                         READY     STATUS    RESTARTS   AGE
flask-app-85b4fbd7bf-8x21k   1/1       Running   0          44s
lsteffenel@cloudshell:~ (docker-miage)$ kubectl get deploy
NAME        DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
flask-app   1         1         1            1           49s
```

- We can see that a "Pod" is running and was deployed
  - `kubectl get pods`

Scaling your app

- Pods are "instances" of your application
- We can augment (or reduce) the number of running instances using the **kubectl scale deployment** command

```
kubectl scale deployment flask-app --replicas=3
```

```
lsteffenel@cloudshell:~ (docker-miage)$ kubectl run flask-app --image=gcr.io/${PROJECT_ID}/flask-app --port 80
deployment.apps "flask-app" created
lsteffenel@cloudshell:~ (docker-miage)$ kubectl get pods
NAME                         READY     STATUS    RESTARTS   AGE
flask-app-85b4fbd7bf-8x21k   1/1       Running   0          44s
lsteffenel@cloudshell:~ (docker-miage)$ kubectl get deploy
NAME        DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
flask-app   1         1         1            1           49s
```

- We can see that more copies of the "Pod" have been added

Exposing your container

- The image is running but is not open to the external world
  - You need to instruct how to do this
  - It can still be accessed from other pods in the same cluster

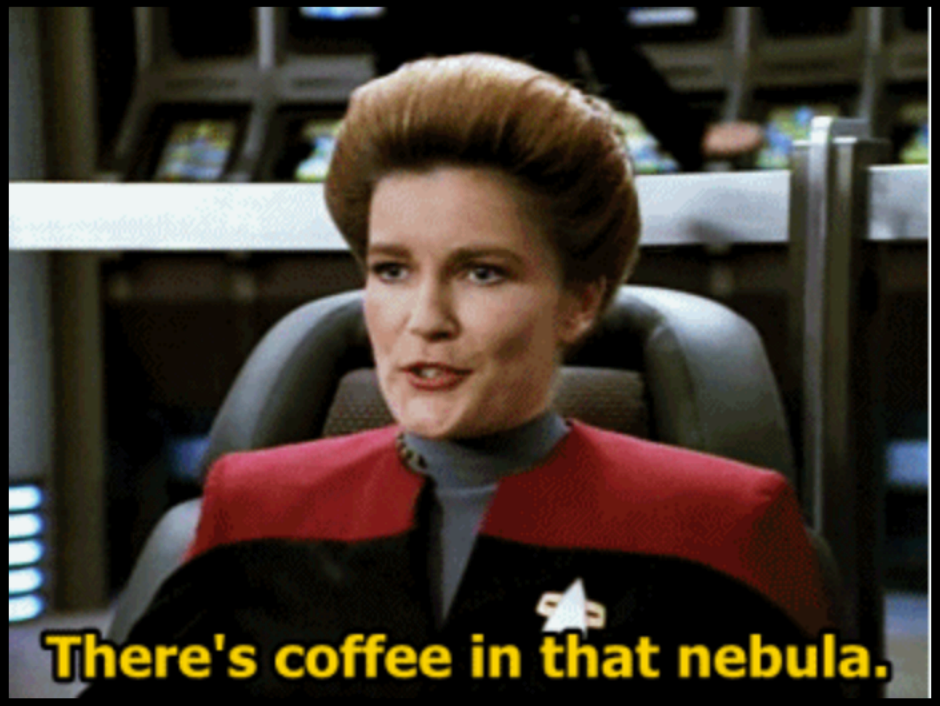
- We can open the image to the world by exposing the ports

```
kubectl expose deployment flask-app --type=LoadBalancer
--port 80 --target-port 80
```

- The service soon will receive an external IP address
- `kubectl get services`

```
lsteffenel@cloudshell:~ (docker-miage)$ kubectl expose deployment flask-app --type=LoadBalancer --port 80 --target-port 80
service "flask-app" exposed
lsteffenel@cloudshell:~ (docker-miage)$ kubectl get services
NAME         TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)        AGE
flask-app    LoadBalancer   10.23.251.2   <pending>     80:32097/TCP   9s
kubernetes   ClusterIP      10.23.240.1   <none>        443/TCP        32m
lsteffenel@cloudshell:~ (docker-miage)$ kubectl get services
NAME         TYPE           CLUSTER-IP    EXTERNAL-IP       PORT(S)        AGE
flask-app    LoadBalancer   10.23.251.2   104.198.145.171   80:32097/TCP   1m
kubernetes   ClusterIP      10.23.240.1   <none>            443/TCP        33m
```

STAR TREK GIF OF THE DAY



Cleanup

- Remove the service
- `kubectl delete service flask-app`

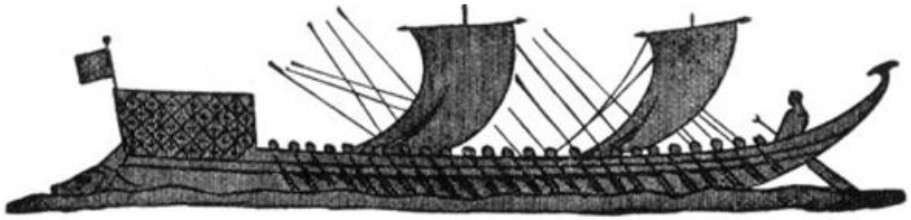
- Stop the container
- `kubectl delete deploy flask-app`

Kubernetes

One orchestrator to rule them all

What is Kubernetes

- "Kubernetes" in Greek is the word for “pilot” or “Helmsman of a ship”



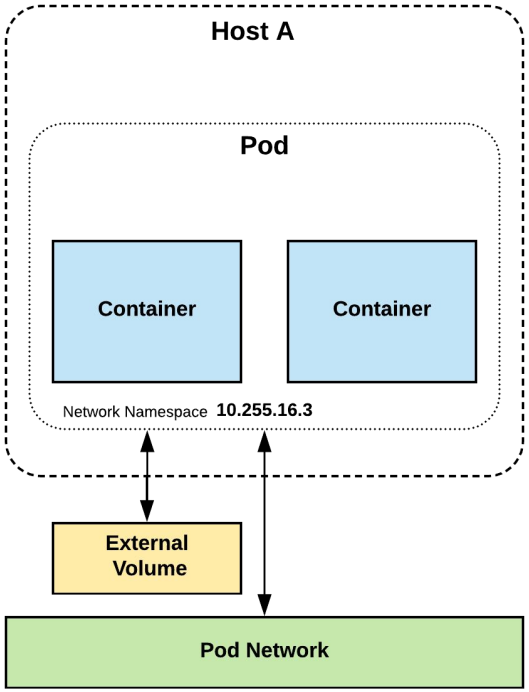
- Project started by Google then released as an open source container orchestration platform
  - Now directed by the "Cloud Native Computing Foundation"
- Designed as a **loosely coupled** collection of components centered around deploying, maintaining and scaling workloads

What Kubernetes can do?

- **Abstracts the underlying hardware** of the nodes
- Works as an engine for resolving state by converging actual and the **desired state** of the system
  - Autoscale Workloads
  - Manage Stateless and Stateful Applications
  - **Native methods of service discovery**
- Provides a uniform interface for workloads
  - support 3rd party apps (highly extensible)
- **Use the SAME API**, no matter if running on bare metal or on a cloud provider!!!

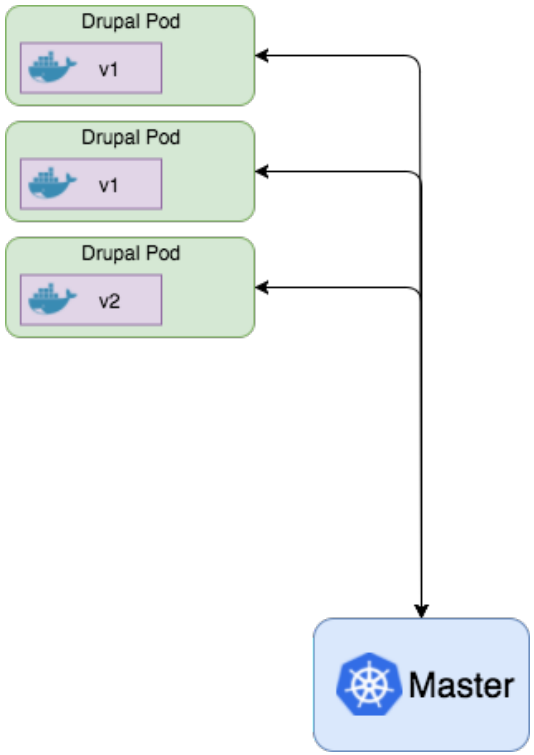
Key Concepts - Pods

- Pods are **atomic units**
  - smallest “unit of work” of Kubernetes
- Pods are **ONE OR MORE containers** that share volumes, a network namespace, and are a part of a single context
  - Like a "service" in docker compose
- Pods are **ephemeral**



Key Concepts - Pods

- Once Kubernetes understands what is in a pod, pod management can include
- System Performance
  - Number of replicas
  - Scale up/down the number of pods based on CPU load or other criteria
- System Monitoring
  - Check the health of each pod
  - Unhealthy pods are killed and replaced
- Deployment options
  - Deploy new versions of the container
  - Control traffic to the new pods to test the new version
    - Blue/Green deployments
    - Rolling deployments



Defining pods

- Before we just launched a single image
  - One single container, default parameters
- A pod may have more than one container and extra parameters (like volumes to mount)
  - Description in a .yaml file

```
lsteffenel@cloudshell:~ (docker-miage)$ cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: myfirstpod
  labels:
    env: test
spec:
  containers:
  - name: myfirstflask
    image: gcr.io/docker-miage/flask-app
    imagePullPolicy: IfNotPresent
    ports:
    - containerPort: 80
```

Launching a pod (standalone)

- kubectl create -f pod.yaml

```
lsteffenel@cloudshell:~ (docker-miage)$ kubectl create -f pod.yaml
pod "myfirstpod" created
lsteffenel@cloudshell:~ (docker-miage)$ kubectl get pods
NAME         READY     STATUS    RESTARTS   AGE
myfirstpod   1/1       Running   0          8s
lsteffenel@cloudshell:~ (docker-miage)$ kubectl describe pods myfirstpod
Name:         myfirstpod
Namespace:    default
Node:         gke-cats-cluster-nom-default-pool-518c3472-rlmr/10.128.0.2
Start Time:   Sun, 14 Oct 2018 21:06:23 +0200
Labels:       env=test
Annotations:  kubernetes.io/limit-ranger=LimitRanger plugin set: cpu request for container myfirstflask
Status:       Running
IP:           10.20.0.27
Containers:
  myfirstflask:
    Container ID:   docker://57f634cef73af2677fb2c742ce3ba031970cfe45d913b32ad3443a088524832c
    Image:          gcr.io/docker-miage/flask-app
    Image ID:       docker-pullable://gcr.io/docker-miage/flask-app@sha256:020ca6b9d64e8005b7729d09f28cb086
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sun, 14 Oct 2018 21:06:24 +0200
    Ready:          True
    Restart Count:  0
    Requests:
      cpu:        100m
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-2bjh4 (ro)
Conditions:
```

Why not launching pods individually?

- Pods are "single units"
  - They do not scale
  - They don't resurrect in case of problems

- Instead of single pods, we must prefer "Deployments"

Key Concepts - Deployment

- A deployment is a way to control the execution of pods in your application
  - It's like a "template" for pods

- Help define parameters such as number of replicas, versions to launch, conditions to scale up/down, ports to expose, etc.
  - It also includes all informations from pods (images, etc.)
- We already created a deployment before :

- `kubectl expose deployment flask-app --type=LoadBalancer --port 80 --target-port 80`

- We can enter many more parameters in a .yaml file

Key Concepts - Deployment

```
lsteffenel@cloudshell:~ (docker-miage)$ cat deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-deployment
spec:
  selector:
    matchLabels:
      app: myfirstpod
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: myfirstpod
    spec:
      containers:
      - name: myfirstpod
        image: gcr.io/docker-miage/flask-app
        ports:
        - containerPort: 80
lsteffenel@cloudshell:~ (docker-miage)$ kubectl create -f deploy.yaml
deployment.apps "flask-deployment" created
lsteffenel@cloudshell:~ (docker-miage)$ kubectl get deploy
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
flask-deployment   2         2         2            2           19s
lsteffenel@cloudshell:~ (docker-miage)$ kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
flask-deployment-64d586c4bf-j52st   1/1       Running   0          26s
flask-deployment-64d586c4bf-nskn7   1/1       Running   0          26s
```

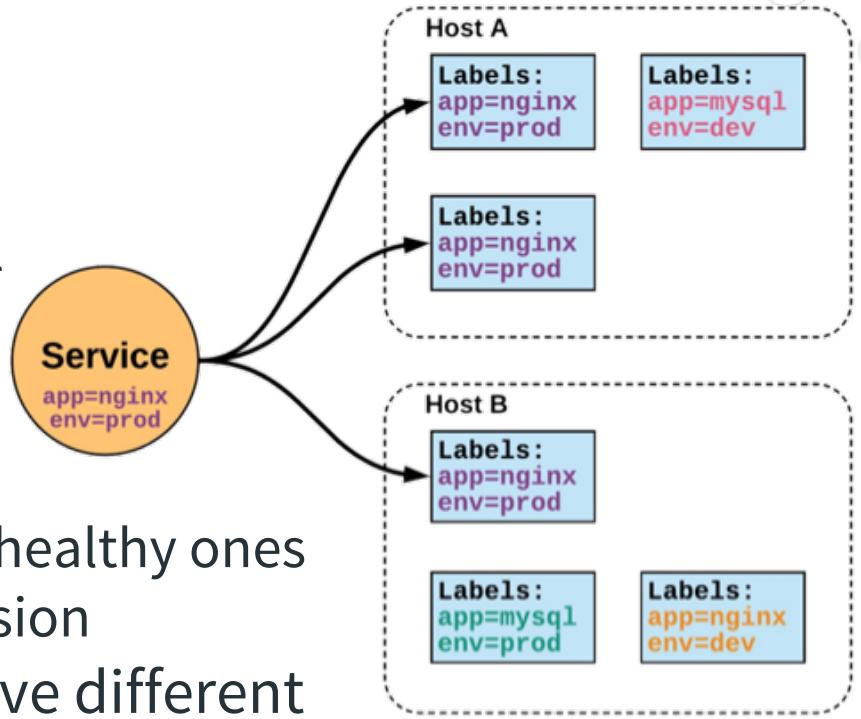
Key Concepts - Deployment

- A deployment can be updated!!!!
  - Here, we scale down to 1 pod and change the image

```
lsteffenel@cloudshell:~ (docker-miage)$ cat depl-update.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-deployment
spec:
  selector:
    matchLabels:
      app: myfirstpod
  replicas: 1 # tells deployment to run 1 pods matching the template
  template:
    metadata:
      labels:
        app: myfirstpod
    spec:
      containers:
      - name: myfirstpod
        image: gcr.io/docker-miage/flask-trek # update to a new image
        ports:
        - containerPort: 80
lsteffenel@cloudshell:~ (docker-miage)$ kubectl apply -f depl-update.yaml
Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
deployment.apps "flask-deployment" configured
lsteffenel@cloudshell:~ (docker-miage)$ kubectl get pods
NAME                                READY     STATUS        RESTARTS   AGE
flask-deployment-57cb889fbb-55hp6   0/1       Pending       0          7s
flask-deployment-64d586c4bf-2fznz   1/1       Terminating   0          24s
flask-deployment-64d586c4bf-bmkdt   1/1       Running       0          24s
lsteffenel@cloudshell:~ (docker-miage)$ kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
flask-deployment-57cb889fbb-55hp6   1/1       Running   0          1m
lsteffenel@cloudshell:~ (docker-miage)$
```

Key concepts - Services

- A service is a unified method to access the exposed workload from Pods
- Durable resource
  - Static cluster IP
  - Static namespace
  - DNS name
- Services are used to control communications with the pods
  - Load balance the requests
  - Don’t send traffic to the unhealthy ones
  - Only talk to the correct version
- Deployment and Service have different responsibilities



Key concepts - Services

```
lsteffenel@cloudshell:~ (docker-miage)$ cat service.yaml
apiVersion: v1
kind: Service
metadata:
  name: flask-service
spec:
  ports:
  - port: 80 # the port that this service should serve on
    # the container on each pod to connect to, can be a name
    # (e.g. 'www') or a number (e.g. 80)
    targetPort: 80
    protocol: TCP
  # just like the selector in the deployment,
  # but this time it identifies the set of pods to load balance
  # traffic to.
  selector:
    app: myfirstpod
  type: LoadBalancer
lsteffenel@cloudshell:~ (docker-miage)$ kubectl create -f service.yaml
service "flask-service" created
lsteffenel@cloudshell:~ (docker-miage)$ kubectl get services
NAME             TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
flask-service    LoadBalancer   10.23.245.191   <pending>     80:32044/TCP   9s
kubernetes       ClusterIP      10.23.240.1     <none>        443/TCP        2h
lsteffenel@cloudshell:~ (docker-miage)$ kubectl get services
NAME             TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)        AGE
flask-service    LoadBalancer   10.23.245.191   104.198.68.27   80:32044/TCP   1m
kubernetes       ClusterIP      10.23.240.1     <none>          443/TCP        2h
```

Necessary to get an external IP address

Key concepts - Services

Key concepts : Labels

- Labels are "tags" to help us manipulate containers
- Labels can be given to several components
  - Pods
  - Volumes
  - Deployments
  - Services
- One nice use is to indicate **version** or **maturity**
- Later we can "filter" using the labels

```
lsteffenel@cloudshell:~ (docker-miage)$ cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: myfirstpod
  labels:
    env: test
spec:
  containers:
  - name: myfirstflask
    image: gcr.io/docker-miage/flask-app
    imagePullPolicy: IfNotPresent
    ports:
    - containerPort: 80
lsteffenel@cloudshell:~ (docker-miage)$ kubectl get pods -l env=test
NAME         READY     STATUS    RESTARTS   AGE
myfirstpod   1/1       Running   0          7m
```

Cleanup

- Remove the service
- `kubectl delete service flask-service`

- Stop the deployment
- `kubectl delete deploy flask-deployment`

- Check if there is a pod still running and delete if needed
- `kubectl get pods`
- `kubectl delete pod XXXX`

- You can now delete the cluster
- `gcloud container clusters delete` **`cluster-name`**

- **Think also to remove container images from the repository (we pay some cents for storage)**
  - ***`docker rmi gcr.io/project/imagename`***

AI Platform

Direct deployment of AI models on the cloud

If Docker can, Tensorflow can do it too

- Kubernetes Engine brings an interesting concept:
  - Can I just plug my _____ and run it on the cloud?

- Deep learning usually requires a lot of resources
  - Memory
  - GPUs
  - Many libraries to configure (NVIDIA, Tensorflow, etc.)
- If the provider can give us an execution environment, why to bother with IaaS configuration?

AI Platform from GCP

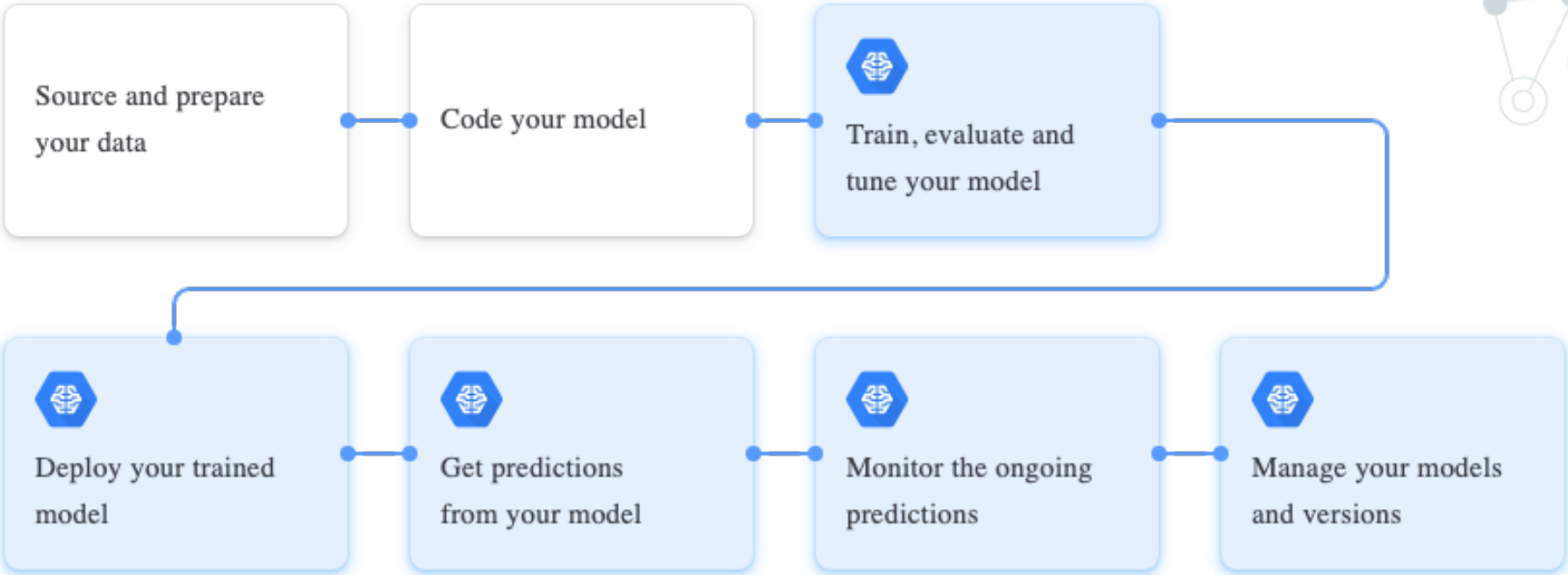
- In the case of GCP, a service called "AI Platform" allow us to rapidly test / train / deploy models on the cloud
- Main idea
  1. From command line, we can launch our code (Sklearn, TensorFlow, Keras, no matter)
     - Special commands added to the gcloud tools

       `gcloud ai-platform XXXXXX`

     - It is possible to "test" a small subset in the console (local), then submit the training to the AI Platform queue

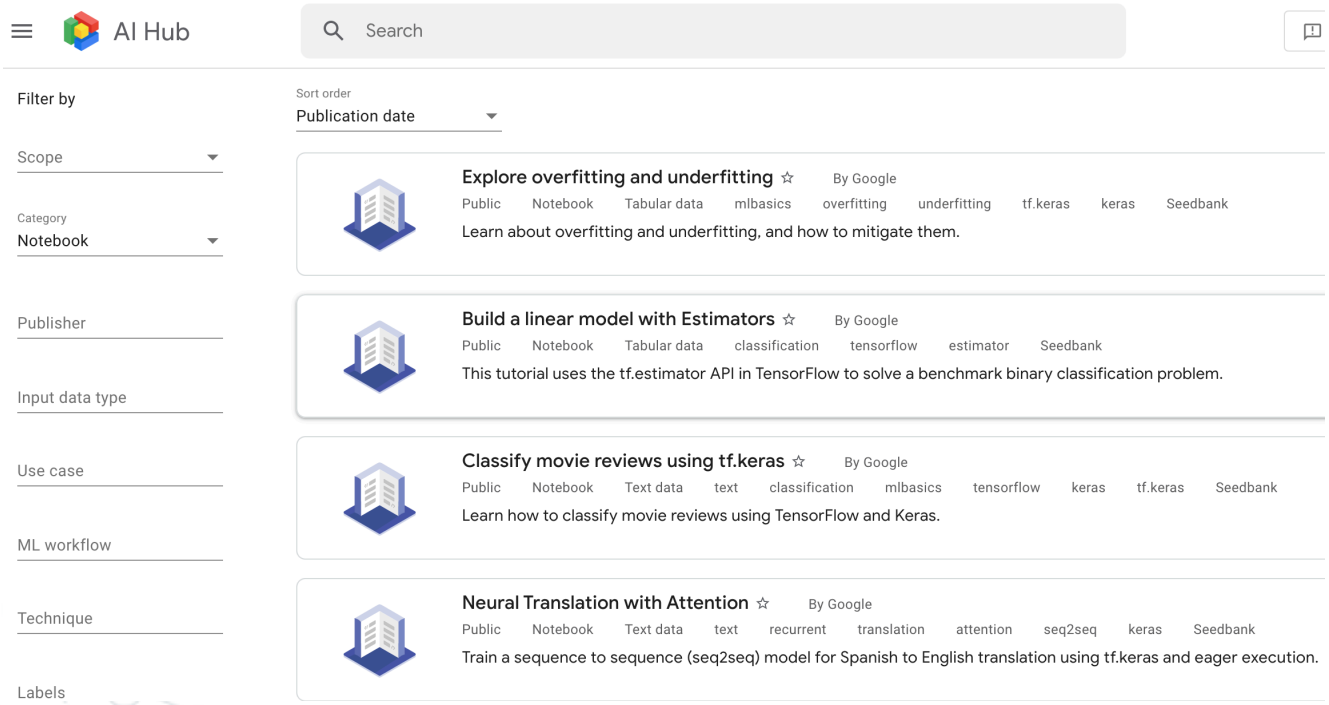
       GCP selects the resources for us, we just need to wait

  2. Once the model was trained, register on the cloud for direct access through the API


Source and prepare your data
Code your model
Train, evaluate and tune your model
Deploy your trained model
Get predictions from your model
Monitor the ongoing predictions
Manage your models and versions

AI Platform bonus

- A lot of tutorials and examples exist in the form of Jupyter notebooks
- We can "import" notebooks from AI Hub, modify, execute and generate models for our needs



Ready to go AI Apps?

Vision, Translation, etc.
(formerly AutoML Vision, ...)

Solutions for common problems

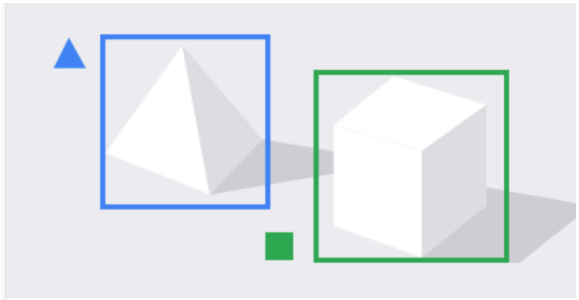
- Some AI problems are quite common
  - Ex : image classification, segmentation
  - Ex : tracking objects in videos
  - Ex : natural language processing and translation
- Instead of writing your own code, a "point and click" solutions is provided

Classification à étiquette unique

Prédit le bon libellé que vous voulez assigner à une image.

Classification multi-étiquette

Prédit toutes les bonnes étiquettes que vous souhaitez assigner à une image.

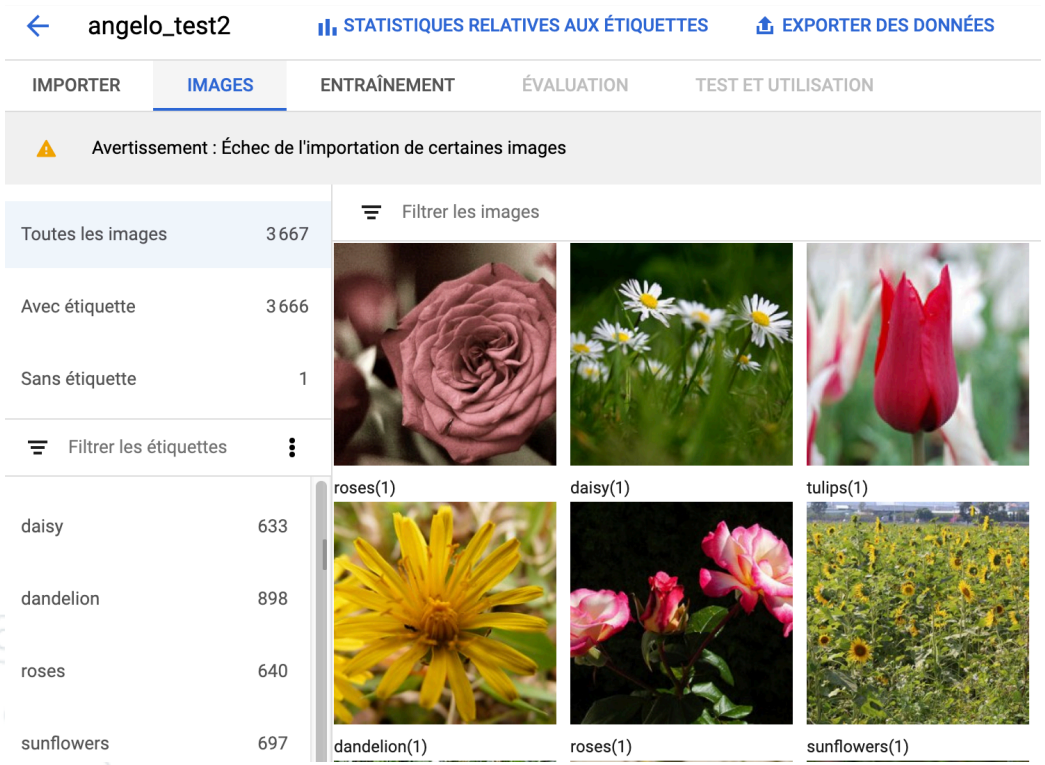


Détection d'objets

Prédit tous les emplacements des objets qui vous intéressent.

Exemple: image classification

- Using Vision, we create a dataset
  1. Import images
  2. Set labels
     - Other option: labels already in a csv file



Exemple: image classification

- Once imported, just click "Train" to launch the IA training
  - ... (it takes a bit of time depending of your dataset)
- Once trained, the model can be verified ("Evaluate") or used for new predictions
  - Select the "Predict" tab, import new images
- The model can be stored in the cloud or downloaded for offline/mobile use

In Summary

- Cloud services are plenty of useful tools
- Cost is an important factor but not only
  - How often I'll use these services?
  - How much time shall I spend to make my own servers?
- IA services have a lot of aggregate value
  - Especially those from $$ APIs (speech, etc.)

- Concerning Kubernetes
  - It's the current standard for container management
  - Even if you don't like, knowing it is a MUST for your CV
    - How to use it in your computer? Install Minikube or Docker Machine (K8s is included)

Contact

angelo.steffenel@univ-reims.fr