# Container Orchestrators

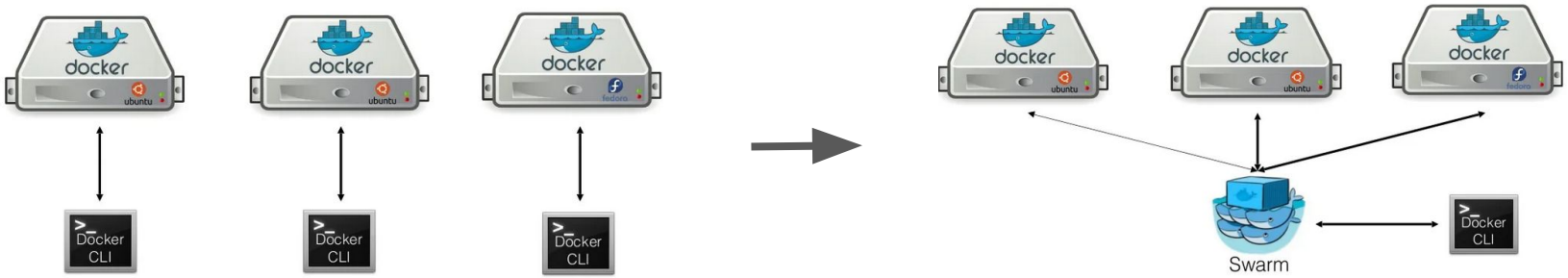
**Boris TEABE**
boris.teabe@inp-toulouse.fr

# Goals

1. **Docker Swarm**
2. Kubernetes
3. Serverless computing

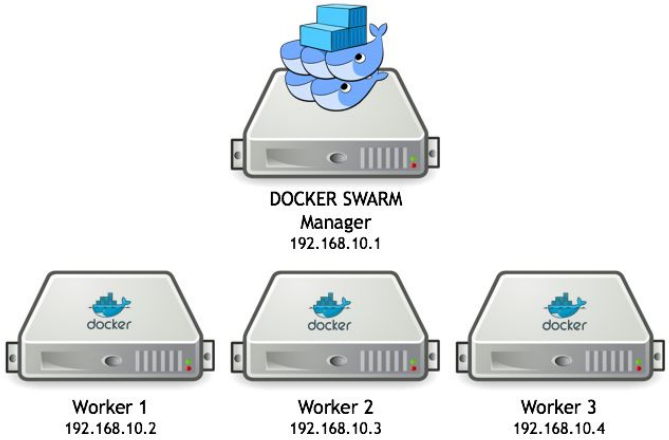
# Docker Swarm

- Native solution of Docker for clustering
  - Turn a cluster into a unique virtual host
  - Use the same API as Docker
- Allow to manage and Schedule containers on a cluster



# Docker Swarm

- A Docker Swarm is a group of either physical or virtual machines that are running the Docker application and that have been configured to join together in a cluster.  machines that have joined the cluster are referred to as nodes or worker.
- Extremely easy to get started



# Docker Swarm

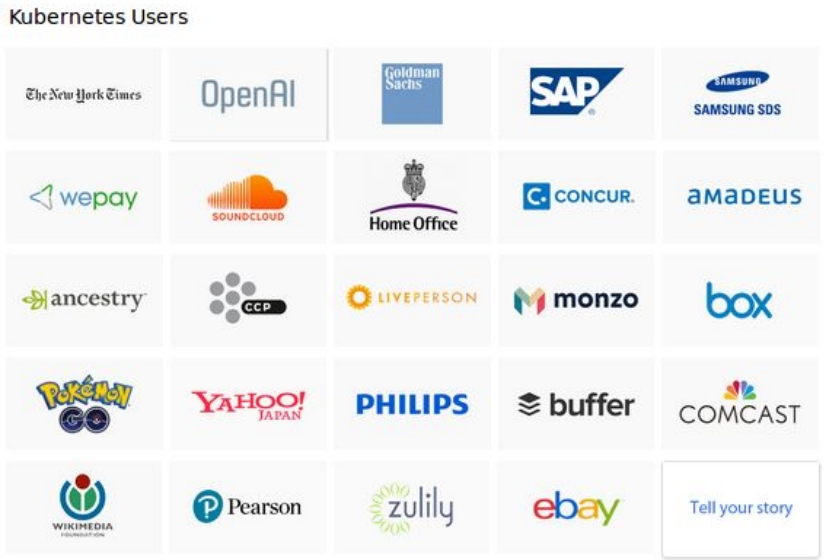
- **Installation**
  - Create a cluster
    - *docker swarm init*
  - Join a cluster
    - *docker swarm join --token ……*
- **Deployment**
  - *docker stack deploy -c ….*

# Goals

1. Docker Swarm
2. **Kubernetes**
3. Serverless computing

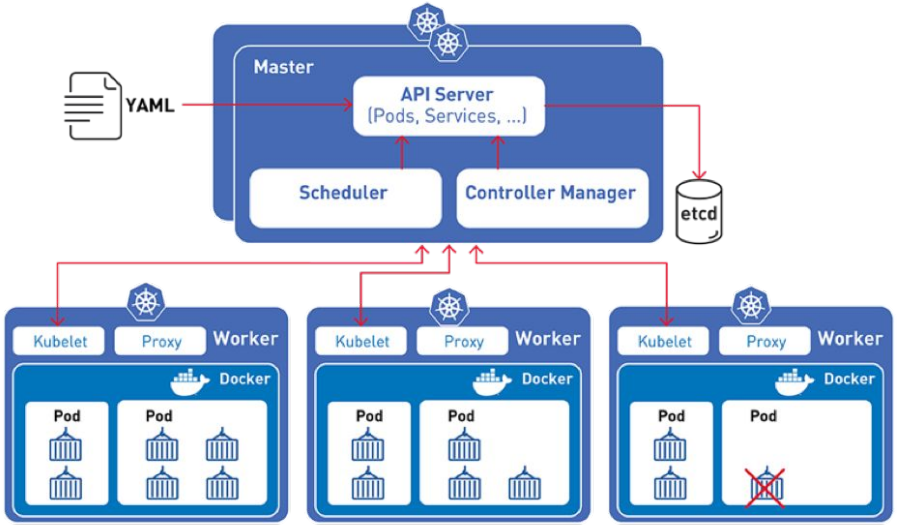
# Kubernetes

- A container orchestration system
- Kubernetes abstracts the thousands of nodes in a cluster and provides industry methods to manage applications. Administrator describes and declares the “desired state”, and kubernetes converts the “current state” to “desired state”.



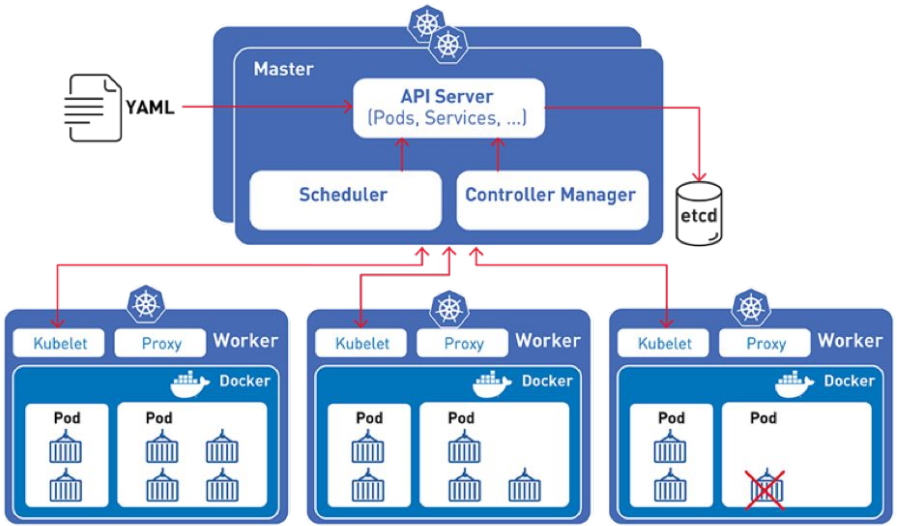
# Kubernetes

- **Architecture**
  - Master
  - Nodes
  - Pod
  - Service and Labels
  - Container
  - Node
    - Kubelet
    - KuberntesProxy



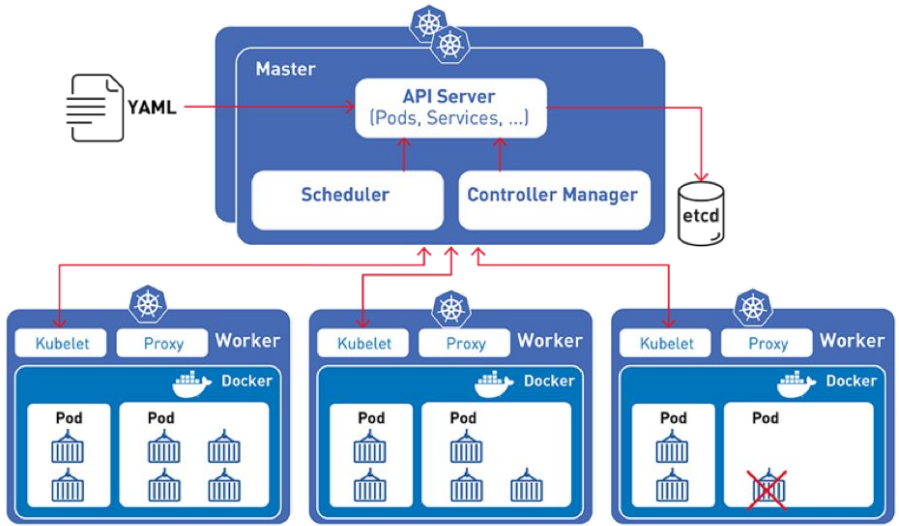
# Kubernetes

- **Master**
  - Master maintains the state of the kubernetes server runtime
  - State is maintained in the etcd backend
  - It is the point of entry for all the client calls to configure and manage kubernetes componets like Nodes, Pods, ReplicationControllers, Services



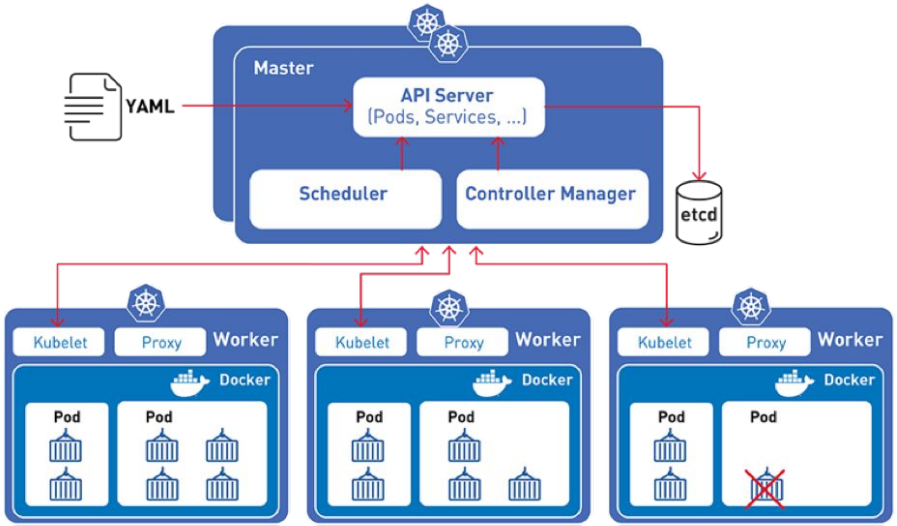
# Kubernetes

- **Node/Worker**
  - Represents the resource provided for provisioning pods
  - Node runs a docker etcd and a kubelet daemon



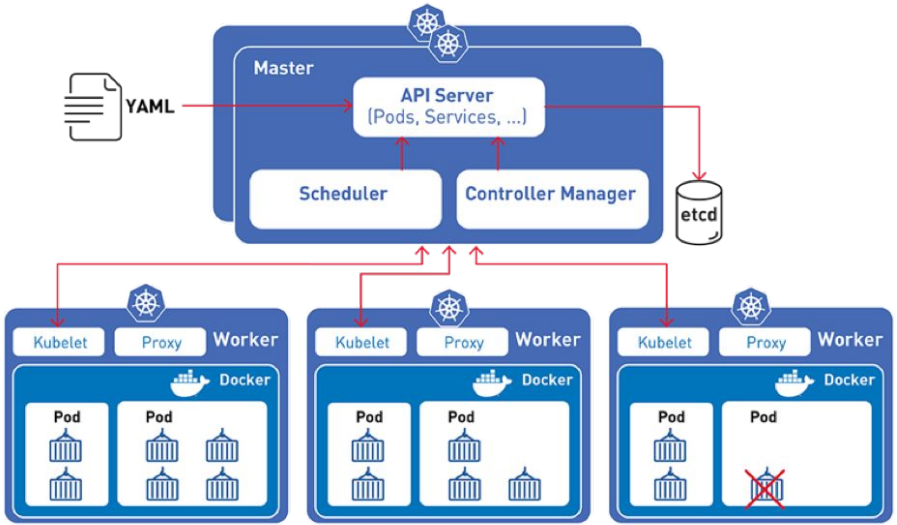
# Kubernetes

- **Kubelet**
  - Component which runs on each nodes and manages the pod and container lifecycle



# Kubernetes

- **Proxy**
  - Manages the network rules on the node and performs connection forwarding or load balancing for kubernetes cluster services

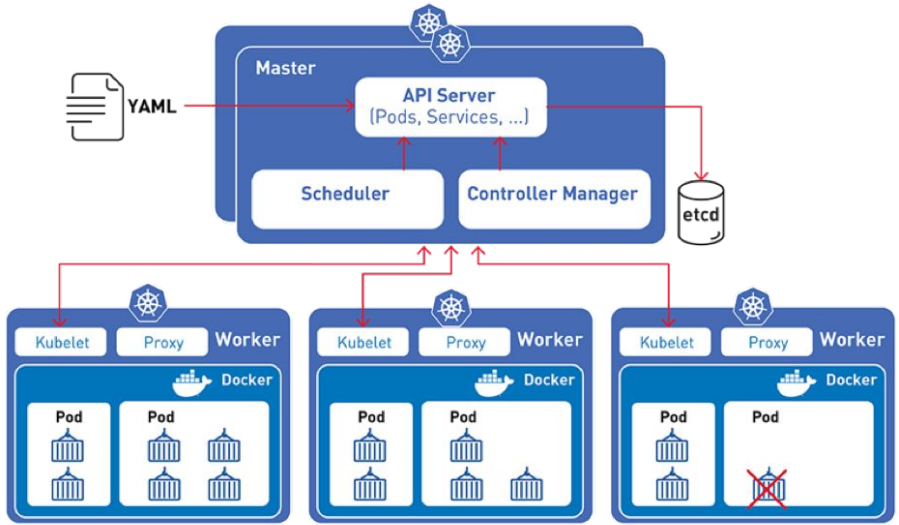


# Kubernetes

- **Concepts-core**
  - **Cluster**. A collection of hosts that aggregate their available resources including CPU, ram, disk and their devices into a usable pool
  - **Master**. Represent a collection of components that make up the control plane of Kubernetes. These components are responsible for all cluster decisions including both scheduling and responding to cluster events
  - **Node**. A single host physical or virtual capable of running pods.
  - **Namespace**. A logical cluster or environment. A method of dividing a cluster.

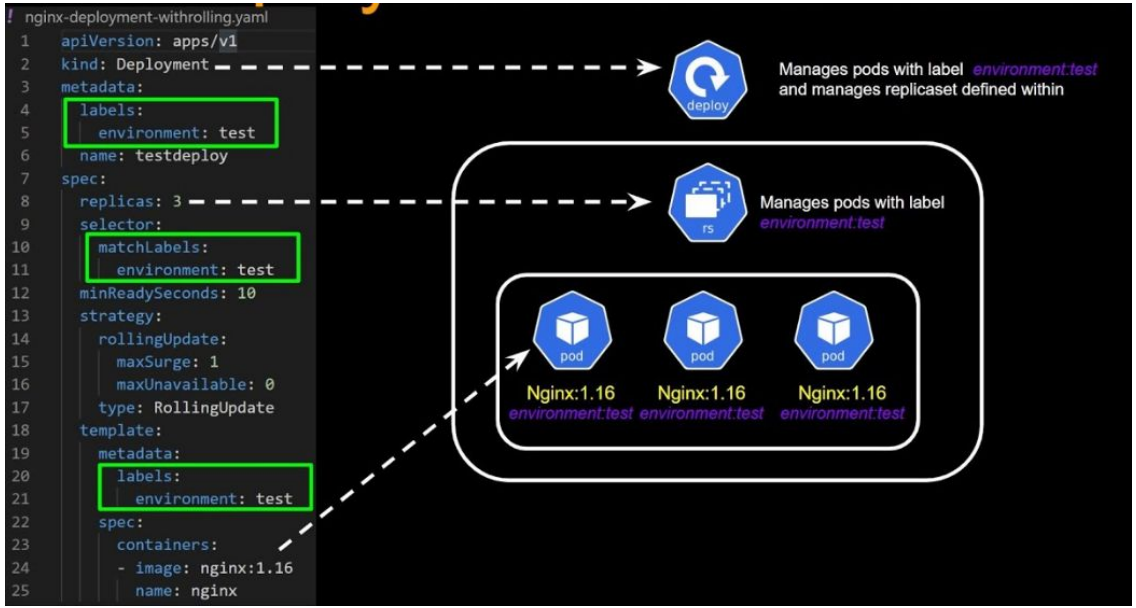
# Kubernetes

- **Concepts-workloads**
  - **Pods**
    - Collection of containers that can run on a host, represent an application
    - In Kubernetes a pod represents a bundle of containers with shared volumes



# Kubernetes

- **Concepts-workloads**
  - **Deployment.** A declarative method of managing stateless Pods.



# Kubernetes

- **Concepts-network**
  - **Service**
    - An abstraction which defines  a logical set of pods and a policy by which to access them
    - The set of Pods targeted by a service is (usually) determined by a label Selector
    - A service defines a TCP or UDP port reservation
    - Allows for abstracted configuration and for mobility and load balancing of the providing containers

# Kubernetes

- **Concepts-Storage**
  - **Volumes**
    - Container’s disks a ephemeral in nature
    - Everytime container restarts ephemeral disks are restarted
    - They are just a mount point or host dir
    - Several kind of volumes exist:
      - empty dir, AWS EBS, GCE Persitent etc.

# Goals

1. Docker Swarm
2. Kubernetes
3. **Serverless computing**

# Serverless computing

- Real cloud-native applications: only provide code for the busines core features
- All management and execution provided by the cloud platform
  - From execution environment to service availability
- Serverless

**Backend-as-a-Service + Function-as-a-Service**

# Backend-as-a-Service

- Common backend components in application architectures
  - Database servers, message queues, (object) storage…
- Better served by the cloud provider
  - Mutualized, no overhead for the user, available
  - Provides an ecosystem of components
- Elasticity requirement: scale quickly, up and down, with the FaaS workload

# Function-as-a-Service

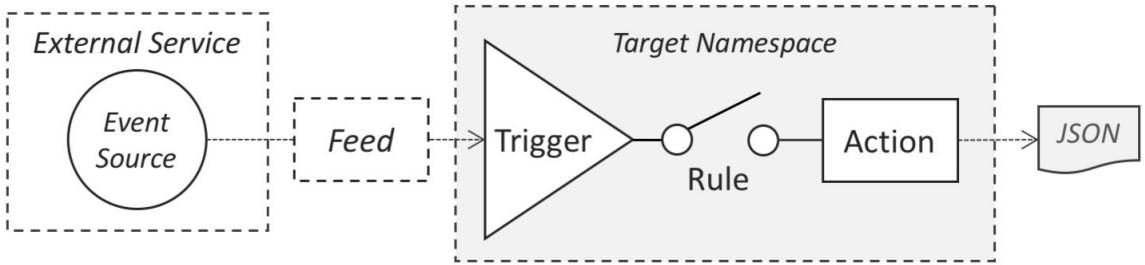
- Run backend code without long-lived servers
  - Execution environments are spawned on-demand
  - All managed by the cloud platform
- The unit of execution is a code block: the function
  - Applications are mostly event-driven
- Central feature of serverless

# Benefits of FaaS

- Elasticity: granularity of the request handler
- Deployment: just write code and upload
  - Quick experimentation, update
- Cost: pay only the compute time you need
  - No request=no function running = no resource to pay

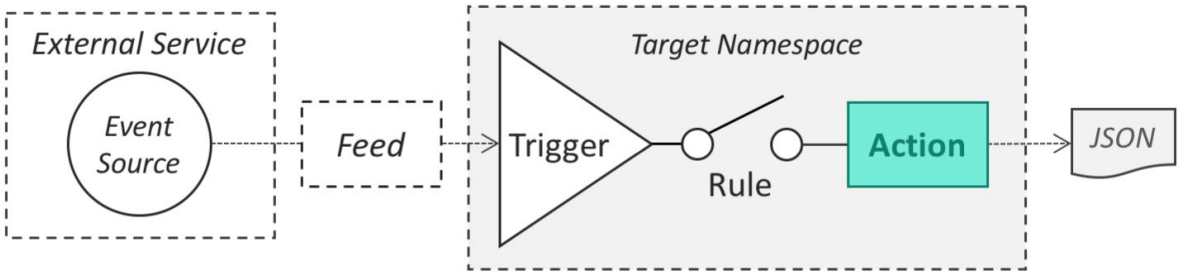
# OpenWhisk

- Distributed Serverless platform that executes functions  in response to events at any scale
- Is Event-driven
    - Events drive the Serverless execution of functional code called Actions. Events can come from any Event Source or Feed service



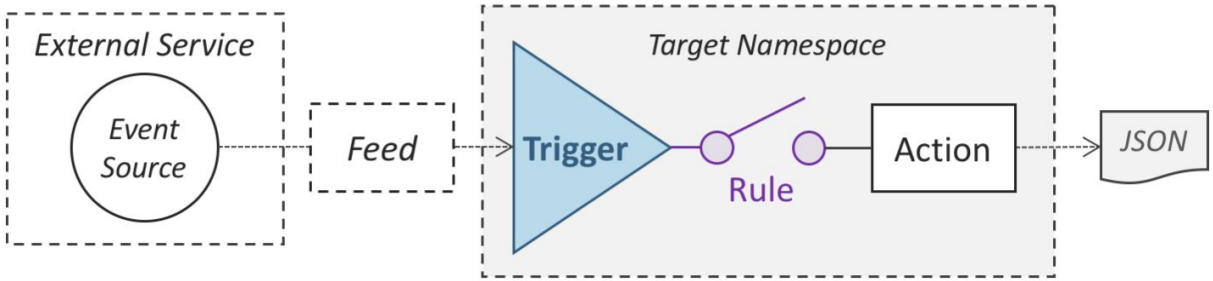
# OpenWhisk

- Action:
  - Stateless functions that run on the OpenWhisk platform. Actions encapsulate application logic to be executed in response to events. Actions can be invoked manually by the OpenWhisk REST API, OpenWhisk CLI, simple and user-created APIs or automated via Triggers



# OpenWhisk

- **Trigger**: kinds of events sent from Event Sources
- **Rule**: Rules are used to associate one trigger with one action.
- **Event Sources**: These are services that generate events that often indicate changes in data or carry data themselves.



# OpenWhisk

- **Architecture of OpenWhisk**