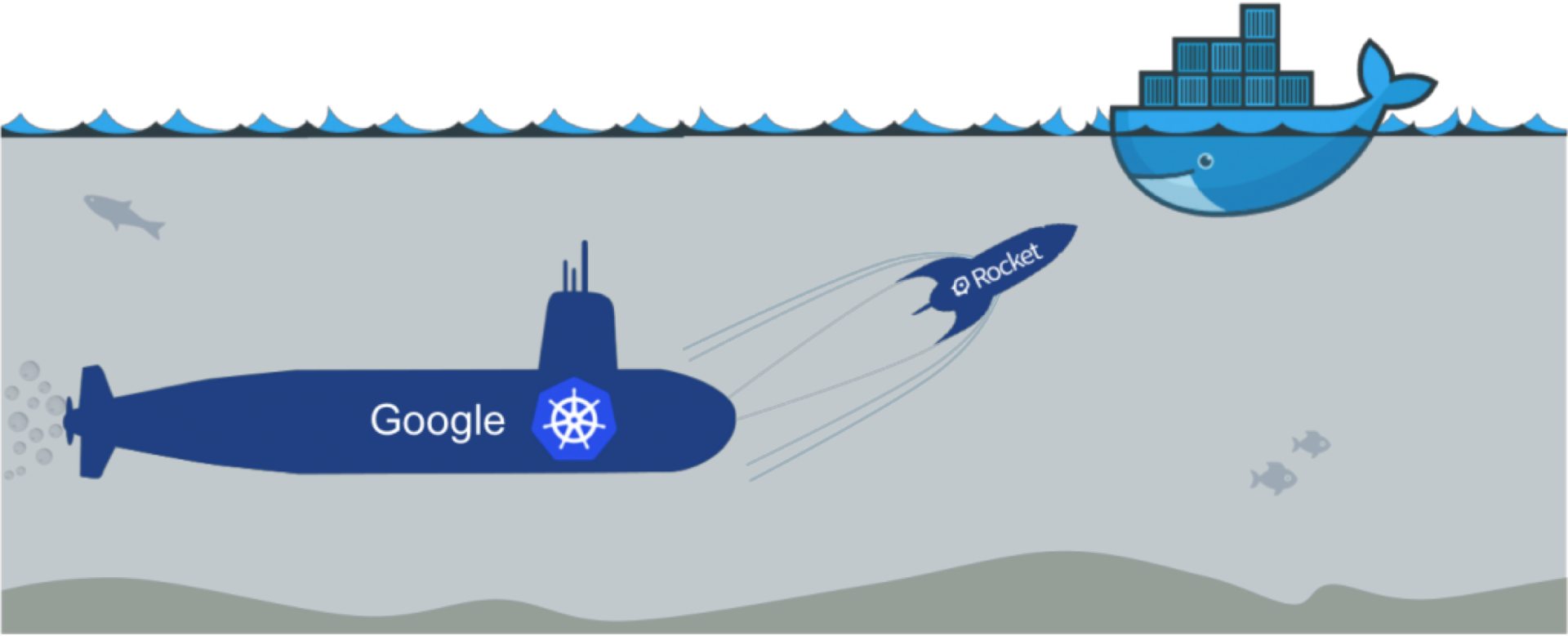
Service Orchestration

BigData from testing to production

PhD. Ronal Muresano

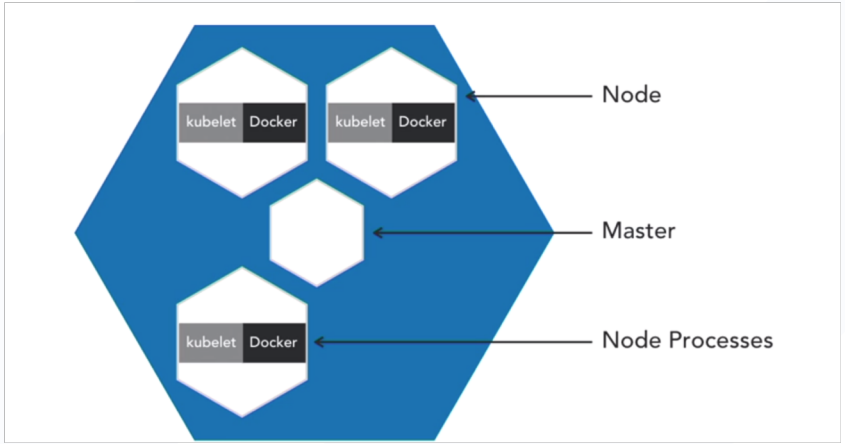

Rocket
Google

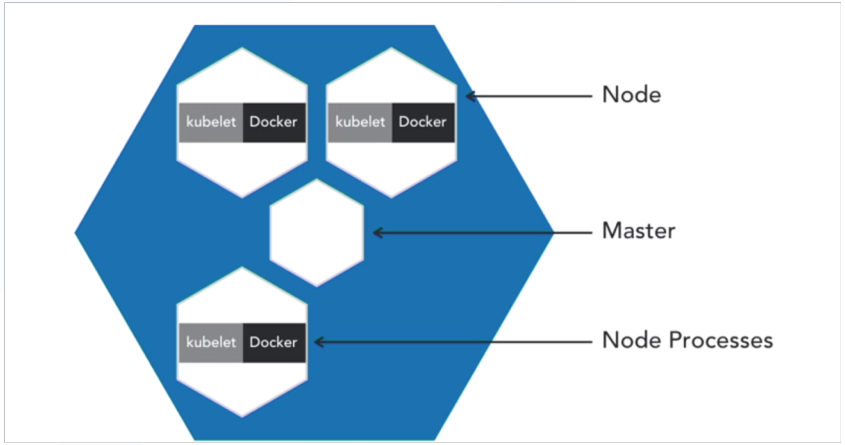
Kubernetes (Multi-container Services) Architecture

Kubernetes is **an open-source system** for managing containerized applications across multiple hosts in a cluster. Kubernetes provides mechanisms for **horizontally scale**, application deployment, **scheduling**, updating, maintenance, **scaling**, run stateless and stateful application and **ideal to execute micro-services.**

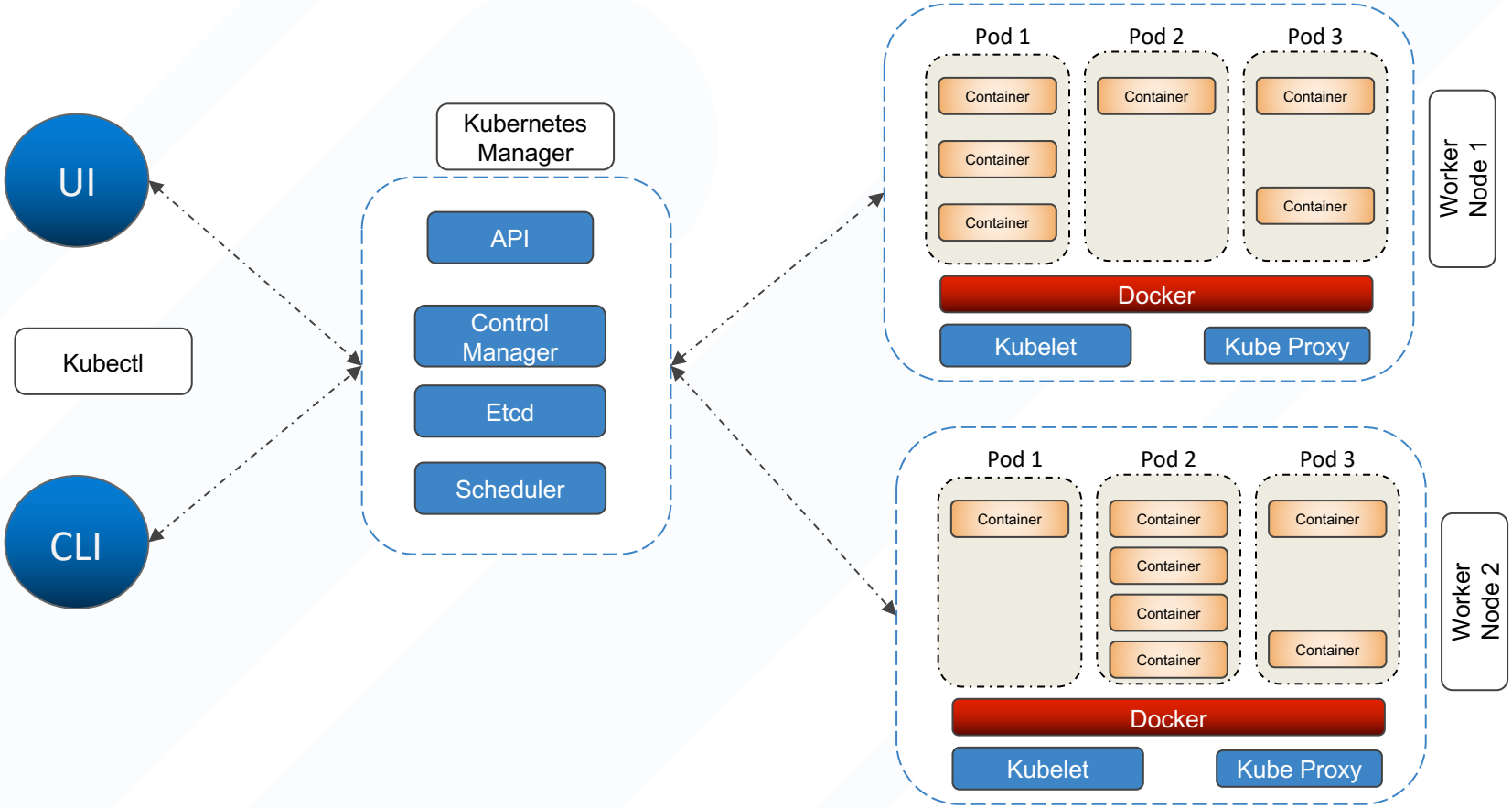
It is a declarative language for launching containers

The state of the entities in the system at any given point of time is represented by Kubernetes Objects. Kubernetes Objects also act as an additional layer of abstraction over the container interface.

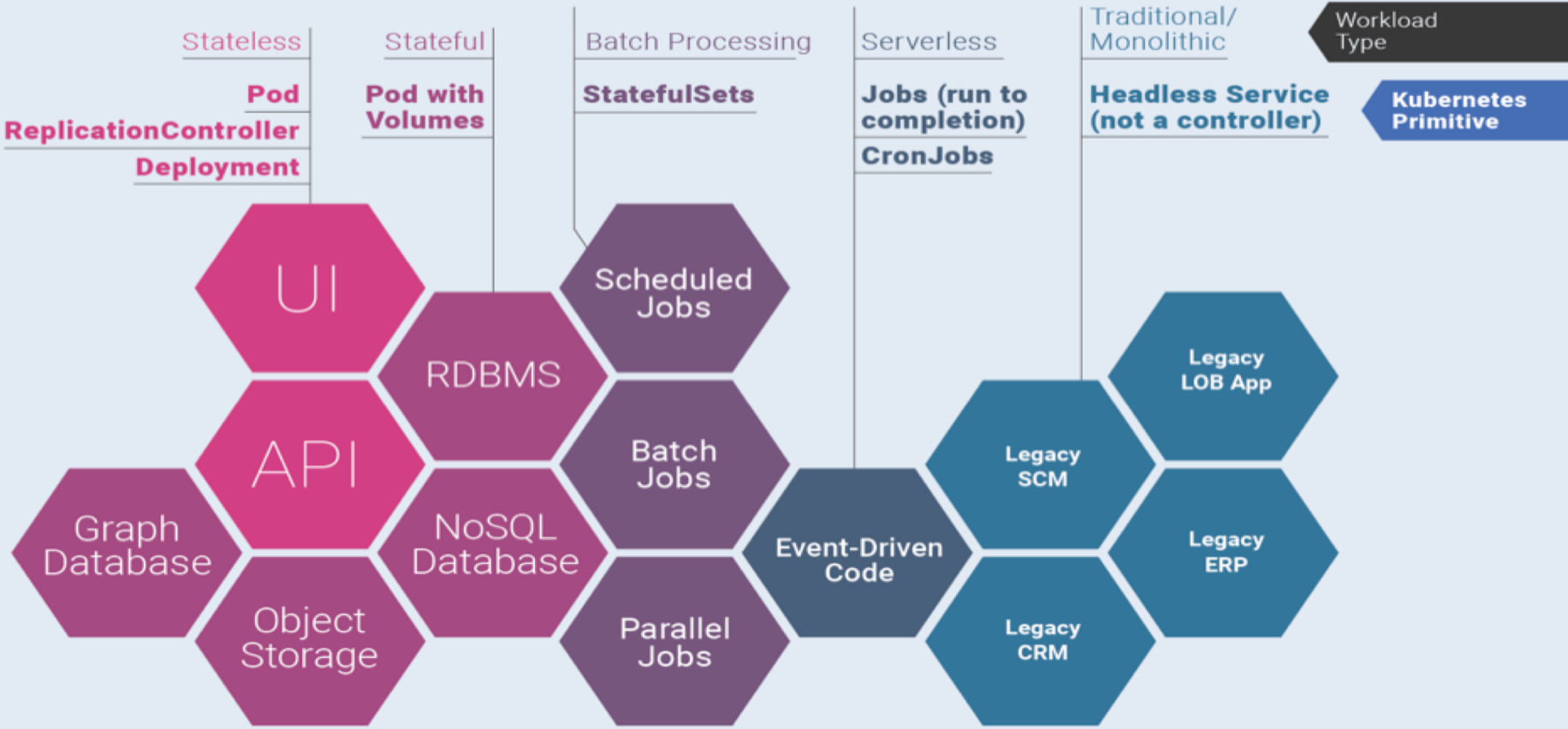




Kubernetes (Multi-container Services) Architecture



Mapping Workloads to Kubernetes Primitives



Source: Janakiram MSV

Kubernetes (Multi-container Services) Cluster Example

Creating Kubernetes Cluster

Pre-requisites

- Vagrant 2.1.4+
- Virtualbox 5.2.18+

Vagrant up

Vagrant File

https://github.com/mureron/bdaasmicroservices/blob/master/demos/kubernetes/cluster/Vagrantfile

Kubernetes (Multi-container Services) Cluster Example

Getting the kubernetes Cluster Info

```
vagrant@k8-manager-bdaas:~/spark$ kubectl get nodes
NAME                STATUS   ROLES    AGE     VERSION
k8-manager-bdaas    Ready    master   2d11h   v1.15.0
k8-worker-bdaas-1   Ready    <none>   2d11h   v1.15.0
k8-worker-bdaas-2   Ready    <none>   2d11h   v1.15.0
vagrant@k8-manager-bdaas:~/spark$ kubectl get pods
NAME                                     READY   STATUS    RESTARTS   AGE
jupyterspark-notebook-5566f85d95-qzk5z   1/1     Running   0          19h
spark-master-86d944466b-nnwbs            1/1     Running   0          19h
spark-worker-cfbb8dfdc-q9fbf             1/1     Running   0          19h
vagrant@k8-manager-bdaas:~/spark$ kubectl get deployments
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
jupyterspark-notebook   1/1     1            1           19h
spark-master            1/1     1            1           19h
spark-worker            1/1     1            1           19h
vagrant@k8-manager-bdaas:~/spark$
```

Main Kubernetes objects (Namespaces)

They are virtual clusters backed by the physical cluster. It is great concept to use for large enterprises where there are many users and teams and you want to give access to different teams but at the same time have a rough idea of who owns what in the Kubernetes environment.

- A great way to divide cluster resources between multiple users and this can be done using resource quotas.
- Names of resources, like deployments and pods, must be unique within the namespace, but not necessarily across separate namespaces.
- There is a **default** namespace where all our objects get placed

namespace.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: big-data-service
```

```
[vagrant@k8-manager-bdaas: $ kubectl get namespaces
NAME                STATUS   AGE
big-data-service    Active   10s
default             Active   13d
kube-node-lease     Active   13d
kube-public         Active   13d
kube-system         Active   13d
```

Main Kubernetes objects (Pods)

Pods: is the smallest deployable unit on a Node. It's a group of containers which must run together. Quite often, but not necessarily, a Pod usually contains one container.

Pod States

- Pending
- Running
- Succeeded
- Failed
- CrashLoopBackOff

pod.yaml

Image
Ports
Volume

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gluster-pod1
  labels:
    name: gluster-pod1
spec:
  containers:
  - name: gluster-pod1
    image: gcr.io/google_containers/nginx-slim:0.8
    ports:
    - name: web
      containerPort: 80
    securityContext:
      privileged: true
    volumeMounts:
    - name: gluster-vol1
      mountPath: /usr/share/nginx/html
  volumes:
  - name: gluster-vol1
    persistentVolumeClaim:
      claimName: gluster1
```

Main Kubernetes objects (Services)

Service is used to define a logical set of Pods and related policies used to access them.

Internal services, where an IP is only reachable from within the cluster.

External services where services running web servers, or publicly accessible pods, are exposed through an external endpoint. These endpoints are available on each node through a specific port. This is called a NodePort

Load balancer. This is for use cases when you want to expose your application to the public internet.

service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: jupyter-notebook
  labels:
    name: jupyter-notebook
spec:
  ports:
    - protocol: TCP
      port: 8888 #
      targetPort: 8888
  selector:
    name: jupyter-notebook
```

```
vagrant@k8-manager-bdaas:~/examples/namespaces$ kubectl get services -o wide
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE   SELECTOR
jupyter-notebook   ClusterIP   10.110.120.30   <none>        8888/TCP   42m   name=jupyter-notebook
kubernetes         ClusterIP   10.96.0.1       <none>        443/TCP    13d   <none>
```

Main Kubernetes objects (Volume)

It is essentially a directory accessible to all containers running in a Pod. Lifetimes are not managed and until very recently there were only local-disk-backed volumes

storage-class.yaml

```yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

Persistent-Volume-Claim.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: jupyter-pv-claim
  labels:
    app: jupyter
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

```yaml
template:
  metadata:
    labels:
      app: jupyter-notebook
  spec:
    containers:
    - name: rmuresano-jupyter
      image: rmuresano/bdaasjupyter:0_0_4
      env:
      - name: STANDALONE
        value: "YES"
      - name: CASSANDRA
        value: "NO"
      - name: HDFS
        value: "NO"
      - name: JUPYTERPORT
        value: "8888"
      ports:
      - containerPort: 8888
      volumeMounts:
      - name: jupyter-storage
        mountPath: /data/jupyter
    volumes:
    - name: jupyter-storage
      persistentVolumeClaim:
        claimName: jupyter-pv-claim
```

Kubernetes Controllers Objects

These are used to help us in the following tasks **Application Reliability**, **Scaling** and **Load Balancing.**

There are a number of Controllers provided by Kubernetes.

- **Replication Controller / Replica Set** ensures that a specified number of Pod replicas are running at any given time.
- **Deployment** is used to change the current state to the desired state.
- **StatefulSet** is used to ensure control over the deployment ordering and access to volumes, etc.
- **Job** is used to perform some task and exit after successfully completing their work or after a given period of time.
- **DaemonSet** is used to run a copy of a Pod on all the nodes of a cluster or on specified nodes.

Replication Controllers

- A ***ReplicationController*** ensures that a specified number of pod replicas are running at any one time.
- It makes sure that a pod or a homogeneous set of pods is always up and available.
- In case of a Pod Fail it automatically starts a new Pod.

Service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: jupyter-notebook
  labels:
    name: jupyter-notebook
spec:
  ports:
    - protocol: TCP
      port: 8888 #
      targetPort: 8888
  selector:
    name: jupyter-notebook
```

ReplicationController.yaml

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: jupyter-notebook
  labels:
    name: jupyter-notebook
spec:
  replicas: 1
  selector:
    name: jupyter-notebook
  template:
    metadata:
      labels:
        name: jupyter-notebook
    spec:
      containers:
      - name: rmuresano-jupyter
        image: rmuresano/bdaasjupyter:0_0_5
        env:
        - name: STANDALONE
          value: "YES"
        - name: CASSANDRA
          value: "NO"
        - name: HDFS
          value: "NO"
        - name: JUPYTERPORT
          value: "8888"
        ports:
        - containerPort: 8888
```

Replication Controllers

```
vagrant@k8-manager-bdaas:~/examples/replicationController$ kubectl get rc,pods,services
NAME                                    DESIRED   CURRENT   READY   AGE
replicationcontroller/jupyter-notebook   1         1         0       87s

NAME                            READY   STATUS              RESTARTS   AGE
pod/jupyter-notebook-gjpj6      0/1     ContainerCreating   0          87s
pod/pod-env-var                 1/1     Running             0          50m

NAME                         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/jupyter-notebook     ClusterIP   10.110.120.30   <none>        8888/TCP   6m52s
service/kubernetes           ClusterIP   10.96.0.1       <none>        443/TCP    13d
vagrant@k8-manager-bdaas:~/examples/replicationController$ kubectl get rc,pods,services
NAME                                    DESIRED   CURRENT   READY   AGE
replicationcontroller/jupyter-notebook   1         1         1       5m16s

NAME                            READY   STATUS    RESTARTS   AGE
pod/jupyter-notebook-gjpj6      1/1     Running   0          5m16s
pod/pod-env-var                 1/1     Running   0          54m

NAME                         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/jupyter-notebook     ClusterIP   10.110.120.30   <none>        8888/TCP   10m
service/kubernetes           ClusterIP   10.96.0.1       <none>        443/TCP    13d
vagrant@k8-manager-bdaas:~/examples/replicationController$ kubectl delete pods/jupyter-notebook-gjpj6
pod "jupyter-notebook-gjpj6" deleted
```

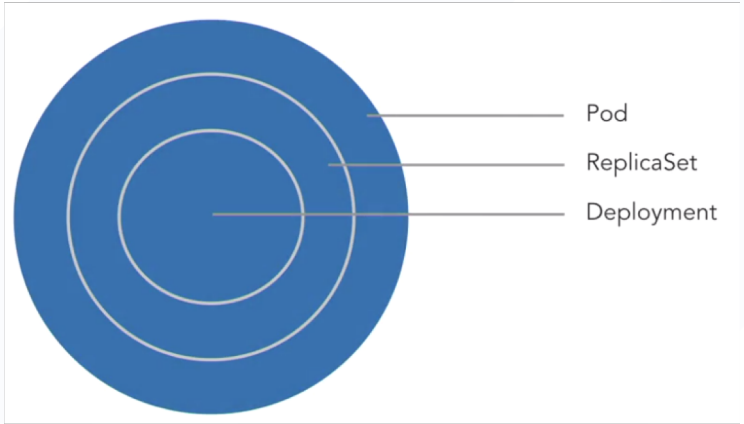
```
vagrant@k8-manager-bdaas:~/examples/replicationController$ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
jupyter-notebook-d5hx4   1/1     Running   0          57s
pod-env-var         1/1     Running   0          55m
```

Deployments

A *Deployment* controller provides declarative updates for Pods and ReplicaSets. It can be defined to create new ReplicaSets or replace existing ones with new ones.

They are higher-level constructs that were introduced to solve specific issues.

- Pod management.
- Scaling a ReplicaSet
- Pod updates and role-backs



Deployment definition

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jupyter-notebook
  labels:
    app: jupyter-notebook
spec:
  replicas: 1
  selector:
    matchLabels:
      app: jupyter-notebook
  template:
    metadata:
      labels:
        app: jupyter-notebook
    spec:
      containers:
      - name: rmuresano-jupyter
        image: rmuresano/bdaasjupyter:0_0_2
        env:
        - name: STANDALONE
          value: "YES"
        - name: CASSANDRA
          value: "NO"
        - name: HDFS
          value: "NO"
        - name: JUPYTERPORT
          value: "8888"
        ports:
        - containerPort: 8888
        volumeMounts:
        - name: jupyter-storage
          mountPath: /data/jupyter
```

Kubernetes (Multi-container Services) Architecture

Deployment definition

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jupyter-notebook
  labels:
    app: jupyter-notebook
spec:
  replicas: 1
  selector:
    matchLabels:
      app: jupyter-notebook
  template:
    metadata:
      labels:
        app: jupyter-notebook
    spec:
      containers:
      - name: rmuresano-jupyter
        image: rmuresano/bdaasjupyter:0_0_2
        env:
        - name: STANDALONE
          value: "YES"
        - name: CASSANDRA
          value: "NO"
        - name: HDFS
          value: "NO"
        - name: JUPYTERPORT
          value: "8888"
        ports:
        - containerPort: 8888
        volumeMounts:
        - name: jupyter-storage
          mountPath: /data/jupyter
```

Volumen definition

```yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/data/jupyter"
```

Example:

https://github.com/mureron/bdaasmicroservices/blob/master/demos/kubernetes/Configuration/services/jupyter/jupyterServiceDepl.yaml

Stateful

Manages the deployment and scaling of a set of Pods, *and provides guarantees about the ordering and uniqueness* of these Pods.

A StatefulSet maintains a sticky identity for each of their Pods.

These pods are created from the same spec, but are not interchangeable: each has a persistent identifier that it maintains across any rescheduling.

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx # has to match .spec.template.metadata.labels
  serviceName: "nginx"
  replicas: 3 # by default is 1
  template:
    metadata:
      labels:
        app: nginx # has to match .spec.selector.matchLabels
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "my-storage-class"
      resources:
        requests:
          storage: 1Gi
```

Jobs

Jobs are a construct that run a pod once and then stop. By itself, jobs aren't really that useful.But a lot of times people end up using CronJobs which are just like jobs, but they run periodically.

jobs.yaml

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: finalcountdown
spec:
  template:
    metadata:
      name: finalcountdown
    spec:
      containers:
        - name: counter
          image: busybox
          command:
          - bin/sh
          - -c
          - "for i in 9 8 7 6 5 4 3 2 1 ; do echo $i ; done"
      restartPolicy: Never
```

```
vagrant@k8-manager-bdaas:~/examples/jobs$ kubectl apply -f jobs.yaml
job.batch/finalcountdown unchanged
vagrant@k8-manager-bdaas:~/examples/jobs$ kubectl get jobs -o wide
NAME              COMPLETIONS   DURATION   AGE   CONTAINERS   IMAGES    SELECTOR
finalcountdown    1/1           5s         85s   counter      busybox   controller-uid=55b
vagrant@k8-manager-bdaas:~/examples/jobs$ kubectl get pods
NAME                    READY   STATUS      RESTARTS   AGE
finalcountdown-m5dgx    0/1     Completed   0          91s
jupyter-notebook-d5hx4  1/1     Running     0          151m
pod-env-var             1/1     Running     0          3h26m
vagrant@k8-manager-bdaas:~/examples/jobs$ kubectl logs finalcountdown-m5dgx
9
8
7
6
5
4
3
2
1
```

ConfigMaps

- It allows to decouple configuration artifacts from image content to keep containerized applications portable.
- One of the benefits it brings ConfigMap it that containers can be portable
- It can be used to pass ENV variables values to a Pod.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: example-configmap
data:
  # Configuration values can be set as key-value properties
  database: mongodb
  database_uri: mongodb://localhost:27017
  TEST_VARIABLE: HOLA
  # Or set as complete file contents (even JSON!)
  keys: |
    image.public.key=771
    rsa.public.key=42

```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-env-var
spec:
  containers:
    - name: env-var-configmap
      image: nginx:1.7.9
      envFrom:
        - configMapRef:
            name: example-configmap
```

ConfigMaps

```
vagrant@k8-manager-bdaas:~/examples/configmap$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
pod-env-var   1/1     Running   0          6m48s
vagrant@k8-manager-bdaas:~/examples/configmap$ kubectl exec -it pod-env-var  bash
root@pod-env-var:/# env
HOSTNAME=pod-env-var
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT=tcp://10.96.0.1:443
TERM=xterm
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_HOST=10.96.0.1
TEST_VARIABLE=HOLA
database_uri=mongodb://localhost:27017
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
database=mongodb
PWD=/
NGINX_VERSION=1.7.9-1~wheezy
keys=image.public.key=771
rsa.public.key=42

SHLVL=1
HOME=/root
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
_=/usr/bin/env
```

Playing with Kubernetes

Exercise 1: Deploy a Jupyter using the volumen claim and services. Time 10-15 min

Steps to follow:

1) **Git clone the class repo in case you don't have it**

   git clone git@github.com:mureron/bdaasmicroservices.git

2) **Enter in the folder Kubernetes/Services/kubclt/jupyter**

3) kubectl apply -f StorageClass.yaml

4) kubectl apply -f jupyterServiceDepl.yaml (This file has the Volume Claim, Service and Deployment)

5) Make the port Forwarding : kubectl port-forward service/<name> 8003:8888

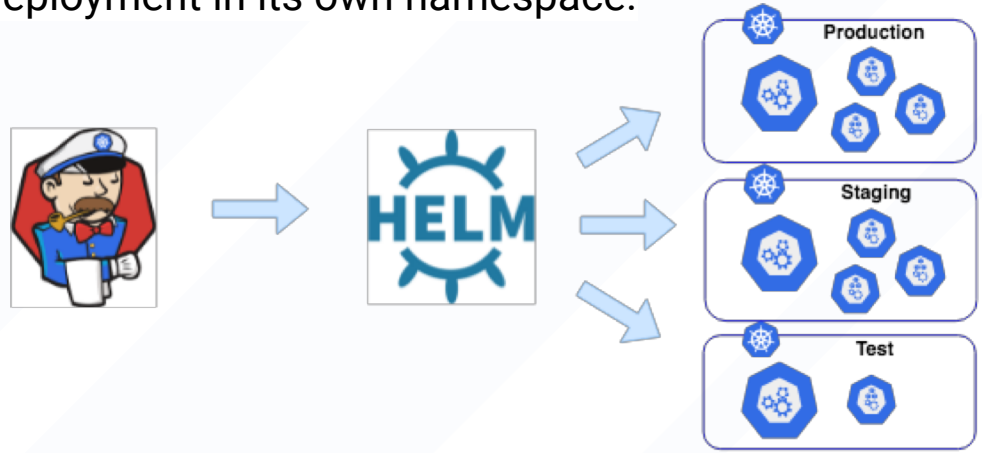
6) Create a ssh Tunnel using in a new terminal ssh -L 8003:localhost:8003 vagrant@192.168.205.10
   password : vagrant

7) Enter in the service using localhost:8002

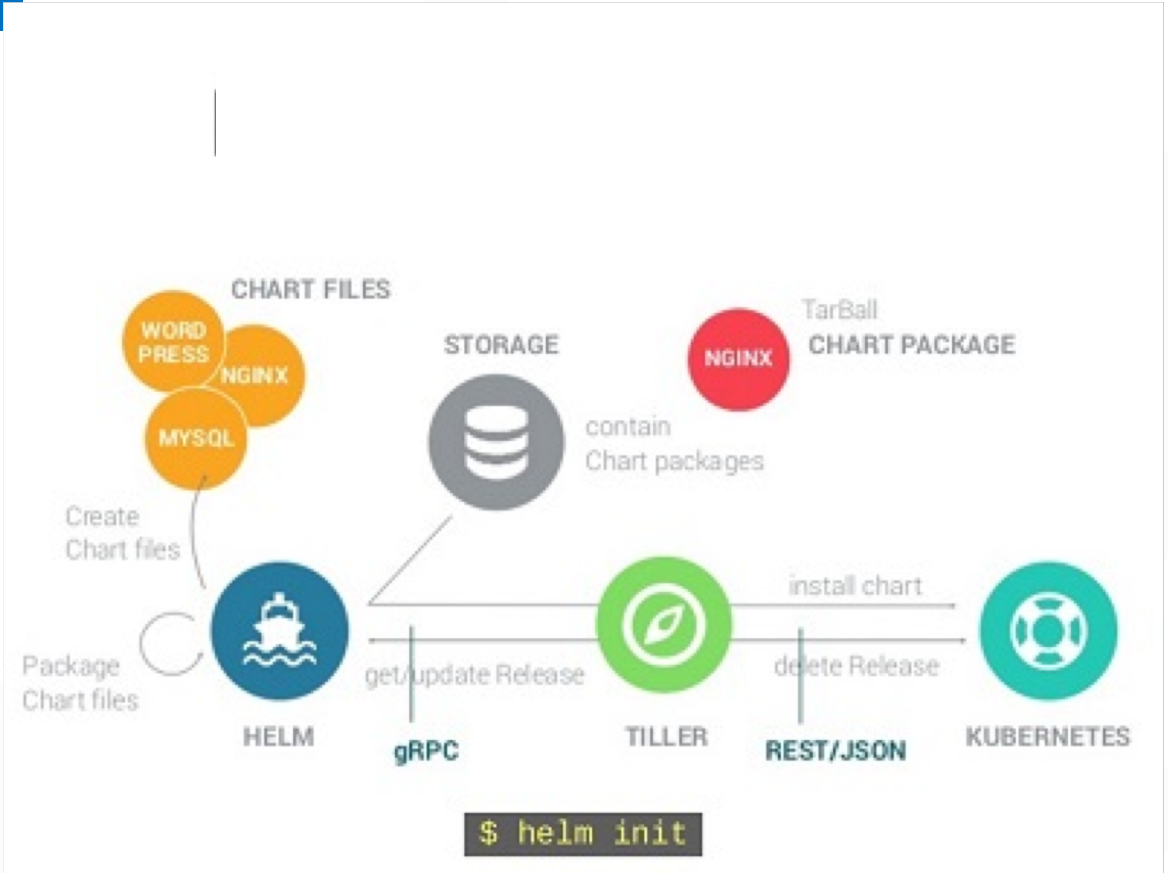
Helm as a Kubernetes manager application

It's a tool that coordinates the installation and management of Kubernetes applications.

Helm manages charts, which are packages of preconfigured Kubernetes resources. It uses a client server architecture where the Helm client runs on your local machine and the server component, called Tiller, runs in your Kubernetes cluster as a deployment in its own namespace.



Helm architecture



A Helm chart must contain:

Chart.yaml: contains high-level metadata information about your chart.

Templates: The template file contains your Kubernetes resources and the components that will make up your application

Values.yaml: a file that contains the deployment parameters values.

Helm Chart (Zeppelin-Spark Example)

Chart.yaml

```yaml
apiVersion: v1
name: spark
version: 1.0.0
appVersion: 1.5.1
description: Fast and general-purpose cluster computing system.
home: http://spark.apache.org
icon: http://spark.apache.org/images/spark-logo-trademark.png
sources:
  - https://github.com/kubernetes/kubernetes/tree/master/examples/spark
  - https://github.com/apache/spark
maintainers:
  - name: lachie83
    email: lachlan.evenson@gmail.com
```

Helm Chart (Zeppelin-Spark Example)

values.yaml

```yaml
# Default values for spark.
# This is a YAML-formatted file.
# Declare name/value pairs to be passed
# name: value

Spark:
  Path: "/opt/spark"

Master:
  Name: master
  Image: "k8s.gcr.io/spark"
  ImageTag: "1.5.1_v3"
  Replicas: 1
  Component: "spark-master"
  Cpu: "100m"
  Memory: "512Mi"
  ServicePort: 7077
  ContainerPort: 7077
  # Set Master JVM memory. Default 1g
  # DaemonMemory: 1g
  ServiceType: LoadBalancer

WebUi:
  Name: webui
  ServicePort: 8080
  ContainerPort: 8080

Worker:
  Name: worker
  Image: "k8s.gcr.io/spark"
  ImageTag: "1.5.1_v3"
  Replicas: 3
  Component: "spark-worker"
  Cpu: "100m"
  Memory: "512Mi"
  ContainerPort: 8081
  # Set Worker JVM memory. Default 1g
  # DaemonMemory: 1g
  # Set how much total memory workers have to give executors
  # ExecutorMemory: 1g
  Autoscaling:
    Enabled: false
  ReplicasMax: 10
  CpuTargetPercentage: 50

Zeppelin:
  Name: zeppelin
```

template/spark-master-deployment.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: {{ template "webui-fullname" . }}
  labels:
    heritage: {{ .Release.Service | quote }}
    release: {{ .Release.Name | quote }}
    chart: "{{ .Chart.Name }}-{{ .Chart.Version }}"
    component: "{{ .Release.Name }}-{{ .Values.Master.Component }}"
spec:
  ports:
    - port: {{ .Values.WebUi.ServicePort }}
      targetPort: {{ .Values.WebUi.ContainerPort }}
  selector:
    component: "{{ .Release.Name }}-{{ .Values.Master.Component }}"
  type: {{ .Values.Master.ServiceType }}
```

Helm Chart (Zeppelin-Spark Example)

Template Folder

```yaml
apiVersion: v1
kind: Service
metadata:
  name: {{ template "master-fullname" . }}
  labels:
    heritage: {{ .Release.Service | quote }}
    release: {{ .Release.Name | quote }}
    chart: "{{ .Chart.Name }}-{{ .Chart.Version }}"
    component: "{{ .Release.Name }}-{{ .Values.Master.Component }}"
spec:
  ports:
    - port: {{ .Values.Master.ServicePort }}
      targetPort: {{ .Values.Master.ContainerPort }}
  selector:
    component: "{{ .Release.Name }}-{{ .Values.Master.Component }}"
---
apiVersion: v1
kind: Service
metadata:
  name: {{ template "webui-fullname" . }}
```

- NOTES.txt
- _helpers.tpl
- spark-master-deployment.yaml
- spark-sql-test.yaml
- spark-worker-deployment.yaml
- spark-worker-hpa.yaml
- spark-zeppelin-config-pvc.yaml
- spark-zeppelin-deployment.yaml
- spark-zeppelin-ingress.yaml
- spark-zeppelin-notebook-pvc.yaml

Installing Helm Client

Installation procedure:https://helm.sh/docs/using_helm/#installing-helm

From Binaries

Download your desired version

1. Unpack it (`tar -zxvf helm-v2.0.0-linux-amd64.tgz`)
2. Find the `helm` binary in the unpacked directory, and move it to its desired destination (`mv linux-amd64/helm /usr/local/bin/helm`)

From Snap (Linux)

$ sudo apt update
$ sudo apt install snapd

The Snap package for Helm is maintained by Snapcrafters.

$sudo snap install helm --classic

From Homebrew (macOS)

brew install kubernetes-helm

Set up the RBAC in kubernetes cluster for tiller Helm

- Tiller runs inside your Kubernetes cluster, and manages releases (installations) of your charts.
- Tiller needs access to the Kubernetes API and the RBAC (Role Base Access Control) block this operations.
- To able this option, we need to do the following steps:
    - Create a Service Account *tiller* for the Tiller server (in the kube-system namespace).
    - Bind the *cluster-admin* ClusterRole to this Service Account. We use ClusterRoleBindings to apply to all name-spaces.
    - Update the existing Tiller deployment

The ***cluster-admin*** **ClusterRole** exists by default in your Kubernetes cluster, and allows superuser operations in all of the cluster resources. The reason for binding this role is because with Helm charts, you can have deployments consisting of a wide variety of Kubernetes resources

Set up the RBAC in kubernetes cluster for tiller Helm

Step 1: Create The Tiller Service Account

- Create a *tiller-serviceaccount.yaml* file using *kubectl*:

```
kubectl create serviceaccount tiller --namespace kube-system
```

Step 2 : Bind The Tiller Service Account To The *Cluster-Admin* Role

- Create a clusterroleBinding.yaml
- Deploy it

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: tiller-clusterrolebinding
subjects:
- kind: ServiceAccount
  name: tiller
  namespace: kube-system
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: ""
```

```
kubectl create -f tiller-clusterrolebinding.yaml
```

Set up the RBAC in kubernetes cluster for tiller Helm

Step 3: Update The Existing Tiller Deployment

- Update the existing *tiller-deploy* deployment with the Service Account you created earlier:

```
helm init --service-account tiller --upgrade
```

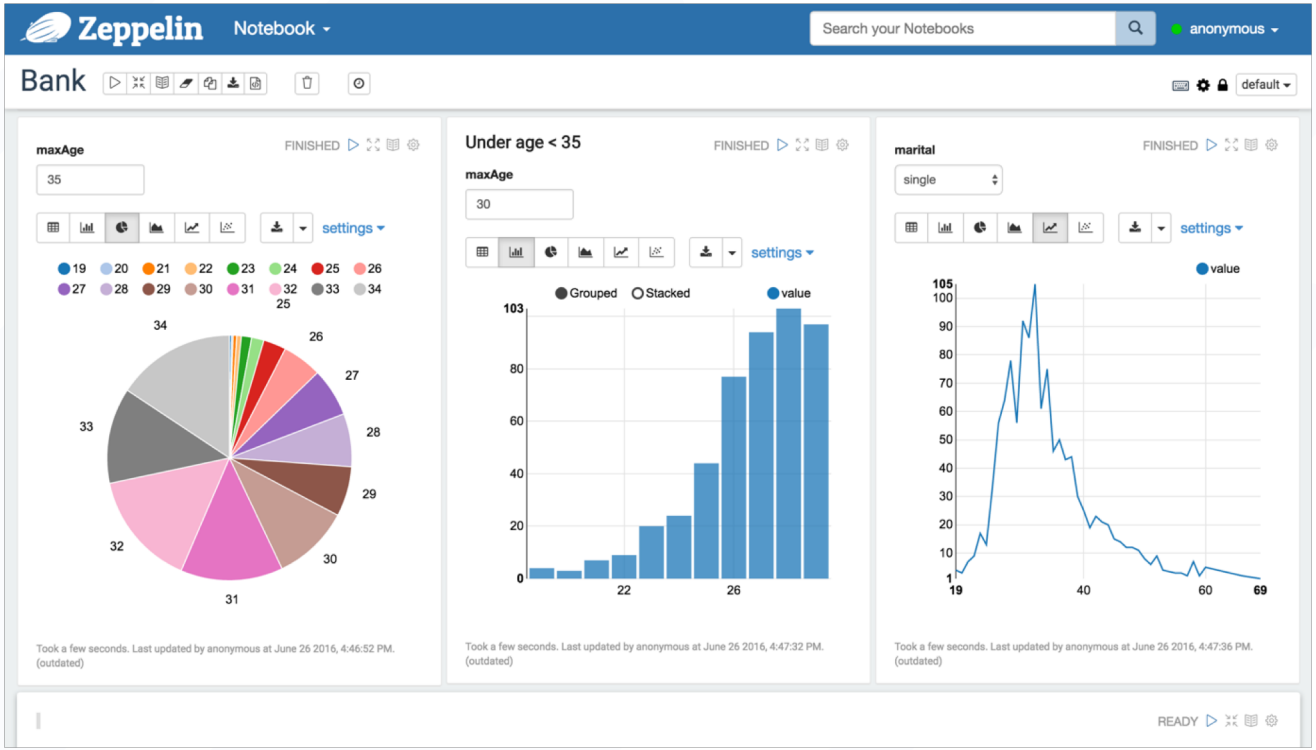
```
MacBook-Pro-de-Ronal:zeppelinspark rmuresano$ kubectl get pod --all-namespaces
NAMESPACE     NAME                                                READY     STATUS    RESTARTS   AGE
default       zeppelispark-master-64dc78f778-tln6b                1/1       Running   0          37m
default       zeppelispark-worker-79bc8df6b4-84d57                1/1       Running   0          31m
default       zeppelispark-zeppelin-68458f76c6-b2pb4              0/1       Evicted   0          8m
default       zeppelispark-zeppelin-68458f76c6-jvt2l              0/1       Evicted   0          19m
default       zeppelispark-zeppelin-68458f76c6-lvj92              0/1       Evicted   0          10m
default       zeppelispark-zeppelin-68458f76c6-n6dc5              0/1       Evicted   0          5m
default       zeppelispark-zeppelin-68458f76c6-rfw25              1/1       Running   0          49s
default       zeppelispark-zeppelin-68458f76c6-spwmc              0/1       Evicted   0          42m
default       zeppelispark-zeppelin-68458f76c6-xklgm              0/1       Evicted   0          12m
kube-system   prometheus-to-sd-fmmcg                              1/1       Running   0          27m
kube-system   prometheus-to-sd-jprkz                              1/1       Running   0          43m
kube-system   prometheus-to-sd-qrw94                              1/1       Running   0          47m
kube-system   prometheus-to-sd-v7t78                              1/1       Running   0          47m
kube-system   tiller-deploy-7b659b7fbd-jc7hh                      1/1       Running   0          31m
```

Preparing Helm

Exercise Installation procedure. Estimated time 10 minutes

Using Helm

Example running Zeppelin and Spark



Using Helm

Steps to follow:

1) Git clone the class repo in case you don't have it

   git clone git@github.com:mureron/bdaasmicroservices.git

2) Enter in the folder Kubernetes/Services/helm

3) helm install --name zeppelinspark -f values.yaml stable/spark
   (wait while the service is completed deployed)

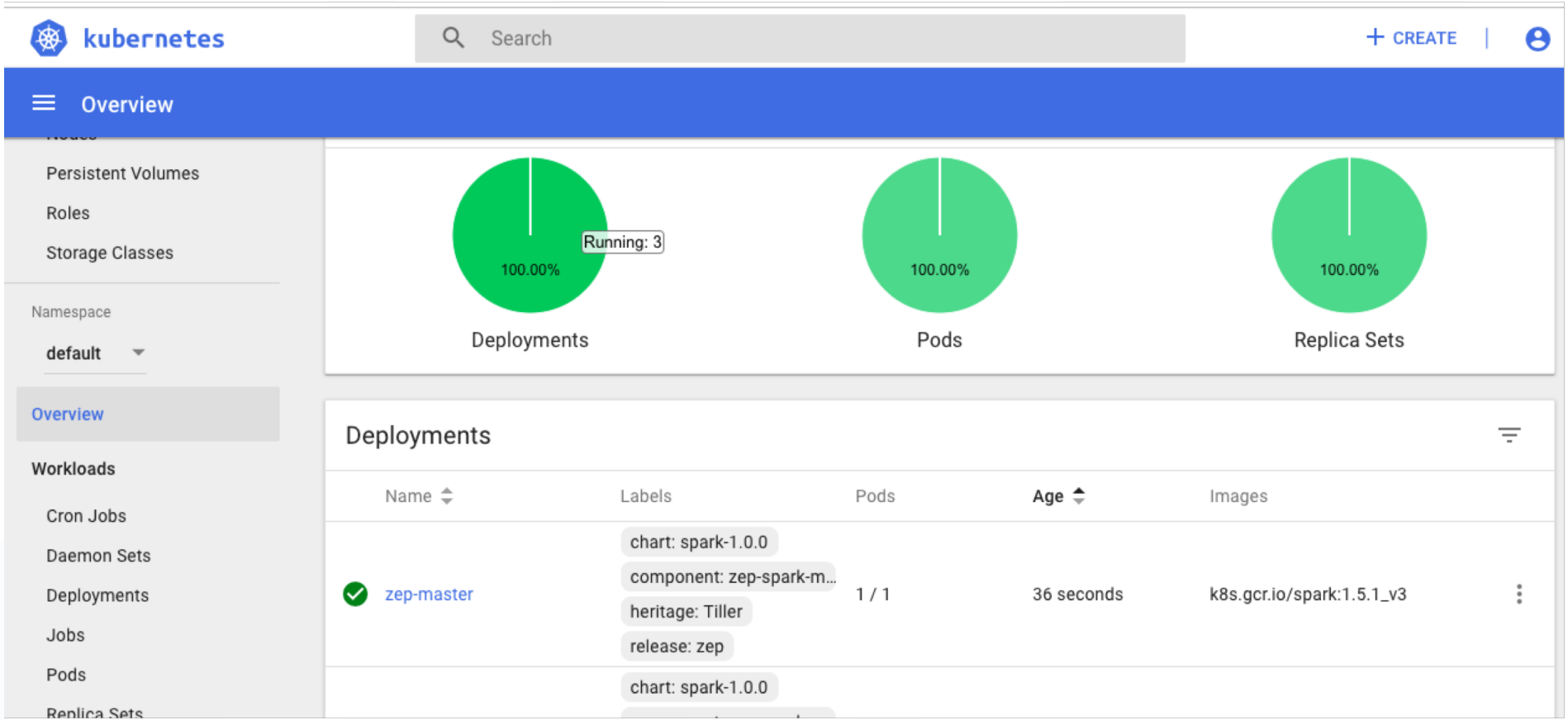
4) kubectl get pods,deployment,services

5) Make the port Forwarding : kubectl port-forward service/<name> 8002:8888

6) Create a ssh Tunnel using in a new terminal ssh -L 8002:localhost:8002 vagrant@192.168.205.10
   password : vagrant

7) Enter in the service using localhost:8002

Zeppelin Spark Helm Deployment on Dashboard




WELCOME
TO THE
HELM WORLD
imgflip.com

Playing with kubernetes and Helm

Exercise: Deploy Grafana and Node Red using Helm. Time 15 minutes

The values.yaml file are in the folder and play with the parameters and the dashboard

Course Roadmap

Docker Swarm

Orchestration in a real private cluster (Deploying and Creating services)

Kubernetes

Production Orchestration. Automating deployment, scaling, and management of containerized applications

Orchestration for developing

Docker and Docker-Compose.

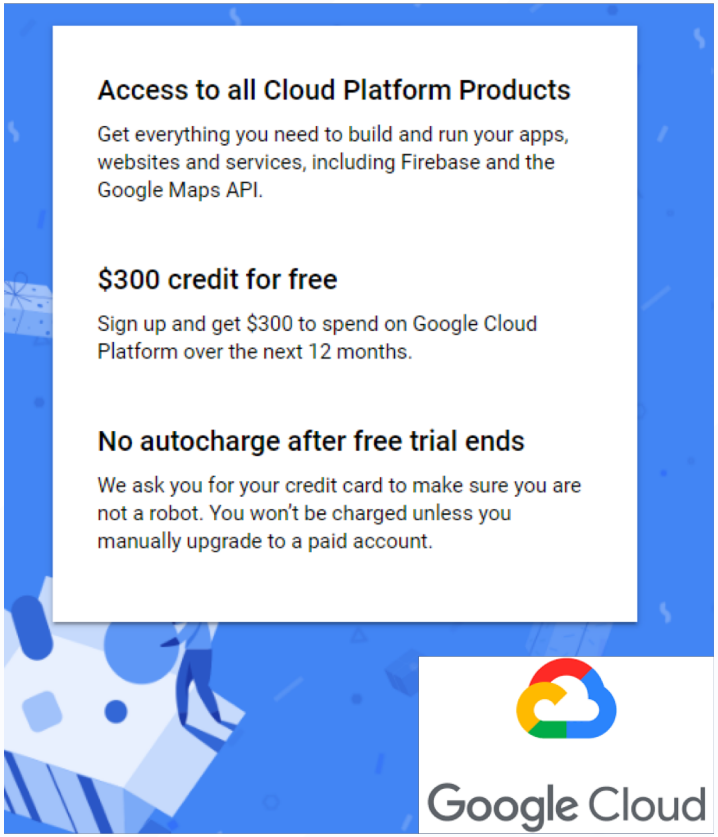
Big Data Challenges

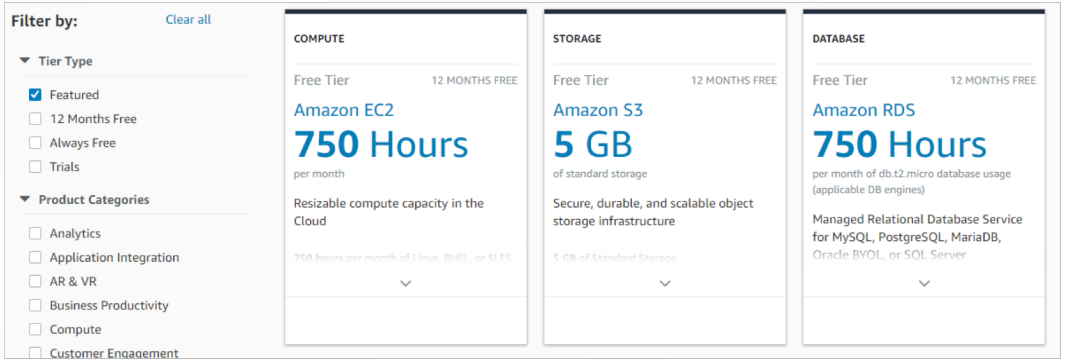
An overview applied to the SMEs. Orchestration Philosophy

Cloud Infraestructure

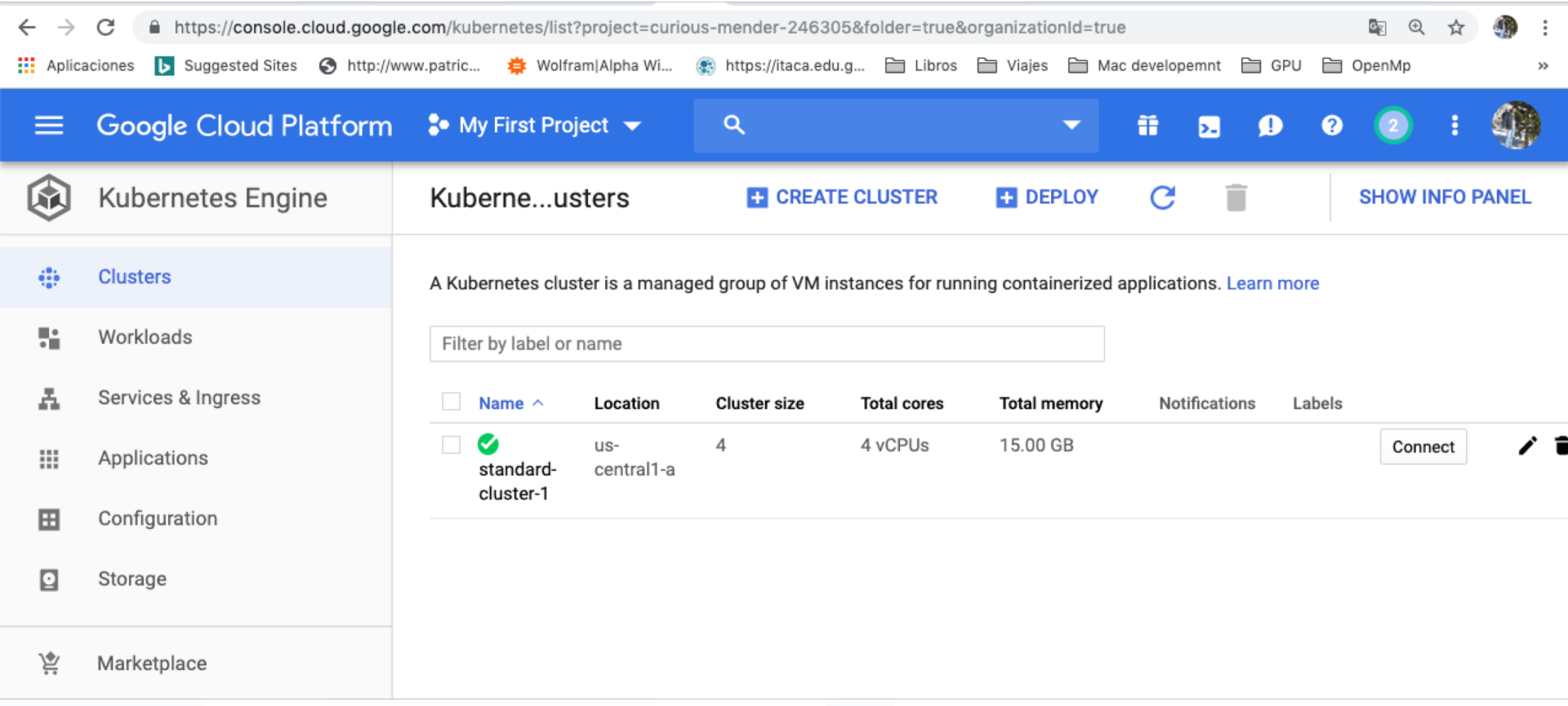
Google Cloud and AWS deployments

Texting in real Cloud Environment

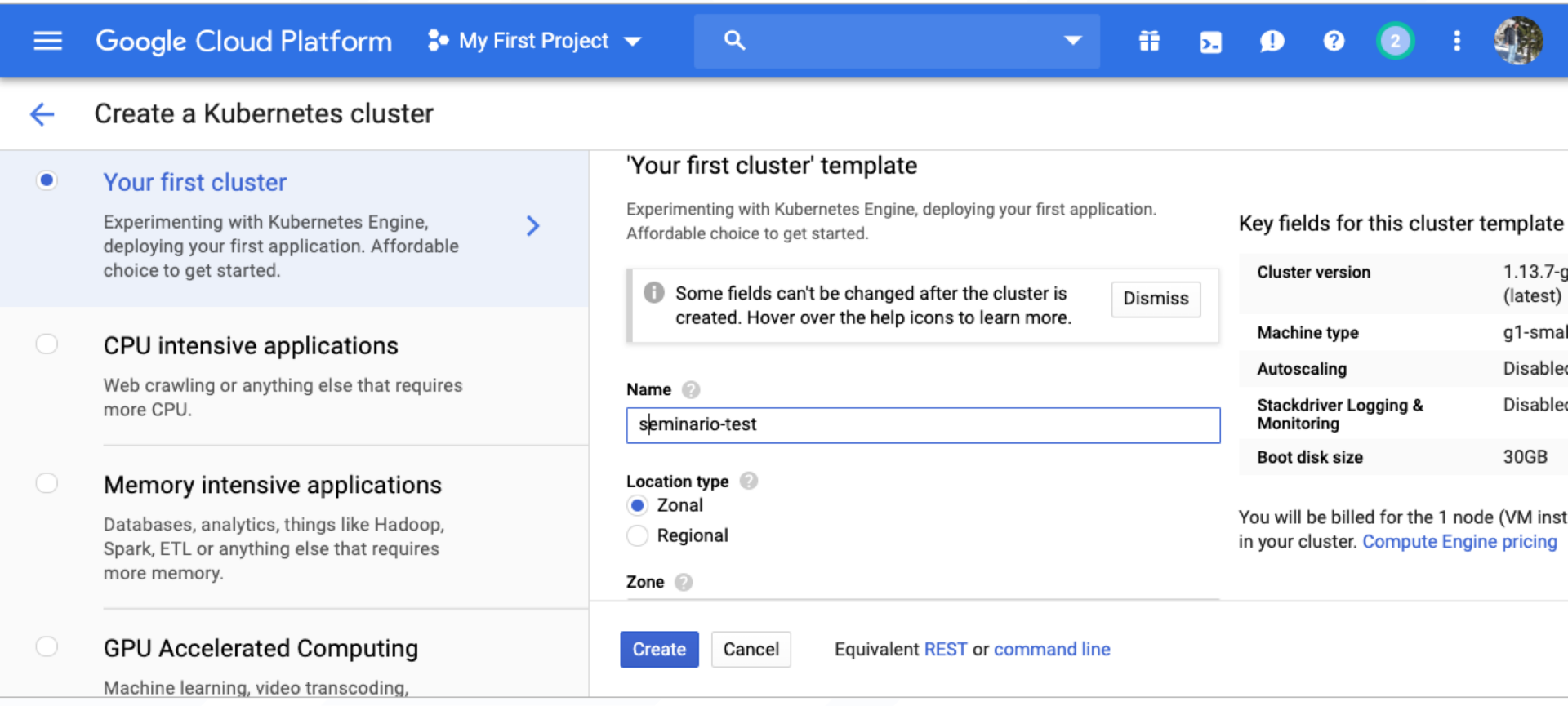




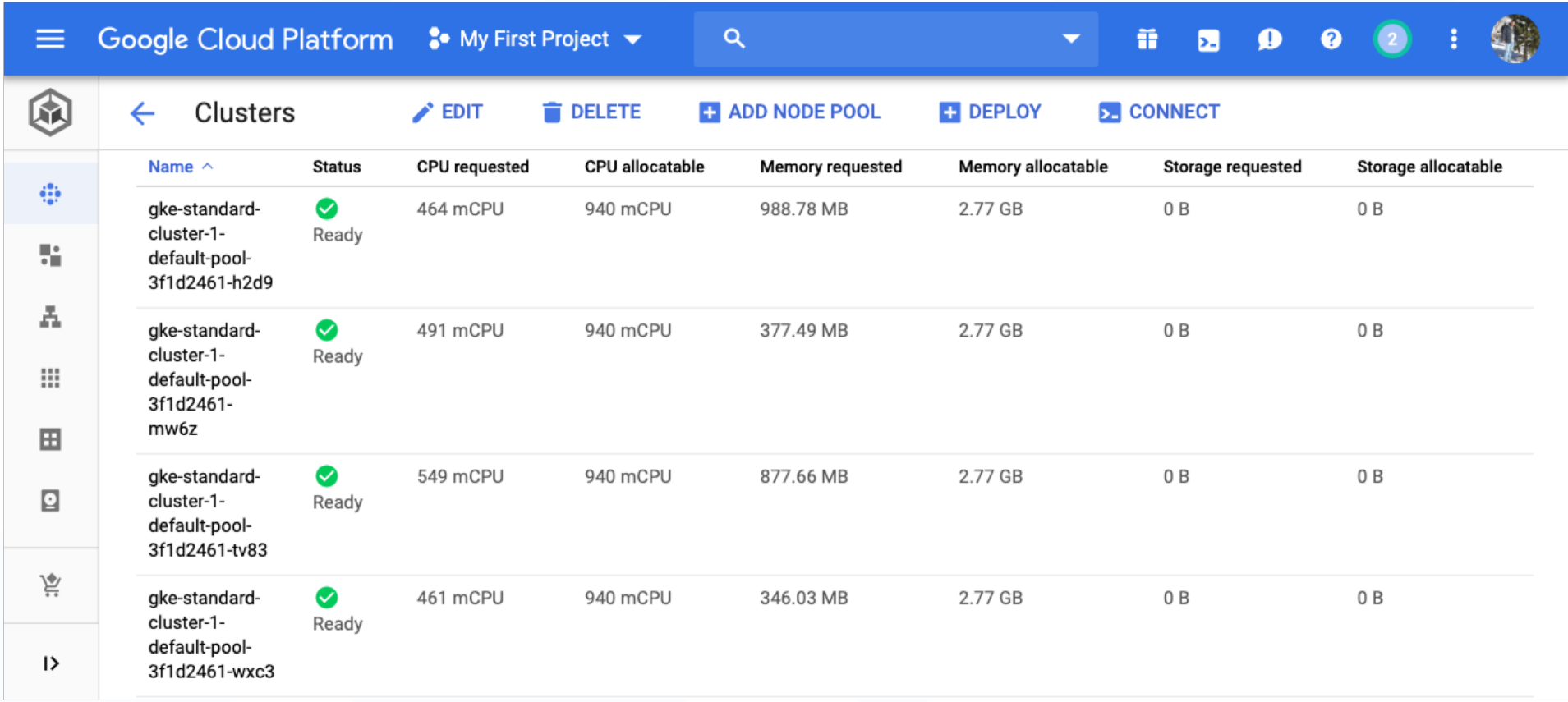
Create a Kubernetes Cluster



Create a Kubernetes Cluster



Create a Kubernetes Cluster (Ready to Use)



Installing the Gcloud command line tool (Ubuntu)

Crea una variable de entorno para lograr la distribución correcta:

```
export CLOUD_SDK_REPO="cloud-sdk-$(lsb_release -c -s)"
```

Agrega el URI de distribución del SDK de Cloud como una fuente de paquete:

```
echo "deb http://packages.cloud.google.com/apt $CLOUD_SDK_REPO main" | sudo tee -a
/etc/apt/sources.list.d/google-cloud-sdk.list
```

Nota: Si tienes instalado apt-transport-https, puedes usar “https” en lugar de “http” en este paso.

Importa la clave pública de Google Cloud:

```
curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
```

Sugerencia para la solución de problemas: Si no puedes obtener las últimas actualizaciones debido a una clave vencida, obtiene el archivo de claves apt-get.gpg más reciente.

Actualiza el SDK de Cloud y también instálalo:

```
sudo apt-get update && sudo apt-get install google-cloud-sdk
```

Installing the Gcloud command line tool (mac)

1. Crea un proyecto de Google Cloud Platform, si todavía no tienes uno.
2. Asegúrate de que Python 2.7 esté instalado en tu sistema: $ python -v
3. Download the package: $wget google-cloud-sdk-231.0.0-darwin-x86_64.tar.gz
4. Descomprimir el fichero $ tar -xzvf google-cloud-sdk-231.0.0-darwin-x86_64.tar.gz
5. `./google-cloud-sdk/install.sh`

Playing with Kubernetes and Google Cloud

Exercise: Estimated time 30-40 min

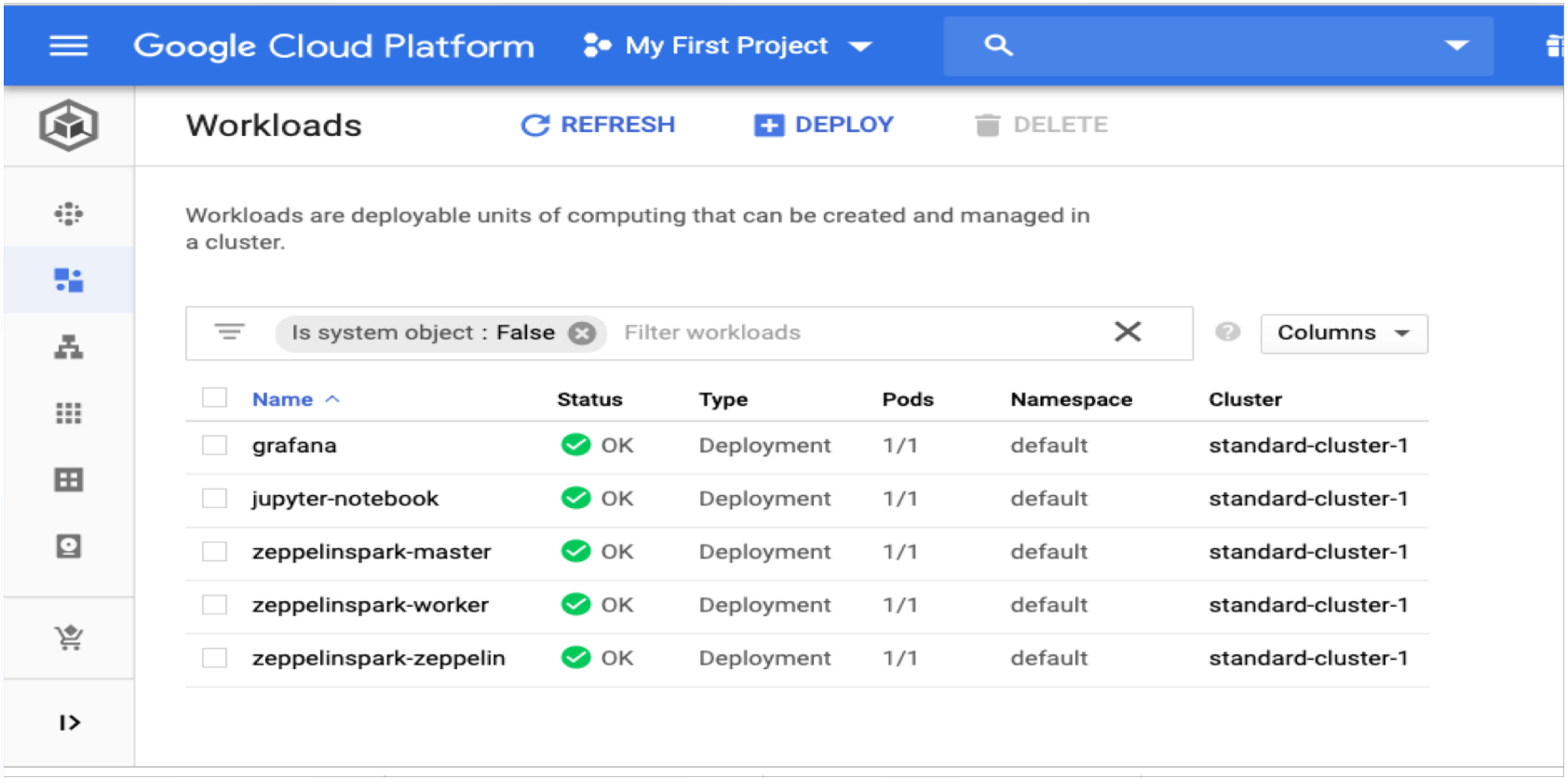
- Install the Helm
  - Deploy Jupyter (Kubectl)
  - Deploy the Zeppelin Spark (using Helm)
- Deploy the Grafana (using Helm)
- Deploy service de grafana (kubectl)

Deployment and Services on Google Cloud

```
MacBook-Pro-de-Ronal:zeppelinspark rmuresano$ kubectl get services,pods,deployments -o wide
NAME                                  TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)          AGE       SELECTOR
service/grafana                       ClusterIP      10.12.13.88    <none>           80/TCP           2h        app=grafana,release=grafana
service/grafana-service-deployment    LoadBalancer   10.12.10.115   35.239.124.107   3000:31889/TCP   2h        app=grafana
service/jupyter-service-deployment    LoadBalancer   10.12.7.79     35.232.42.194    8888:31454/TCP   2h        app=jupyter-notebook
service/kubernetes                    ClusterIP      10.12.0.1      <none>           443/TCP          2h        <none>
service/zeppelinspark-master          ClusterIP      10.12.14.53    <none>           7077/TCP         1h        component=zeppelinspark-spark-
master
service/zeppelinspark-webui           LoadBalancer   10.12.0.109    34.67.56.201     8080:31303/TCP   1h        component=zeppelinspark-spark-
master
service/zeppelinspark-zeppelin        LoadBalancer   10.12.6.57     35.232.19.39     8080:31617/TCP   1h        component=zeppelinspark-zeppel
in

NAME                                      READY     STATUS    RESTARTS   AGE       IP          NODE
pod/grafana-7fb95c4d47-2z4mn              1/1       Running   0          2h        10.8.3.4    gke-standard-cluster-1-default-pool-3f1d246
1-tv83
pod/jupyter-notebook-865f46bdc6-kx5ff     1/1       Running   0          2h        10.8.2.3    gke-standard-cluster-1-default-pool-3f1d246
1-h2d9
pod/zeppelinspark-master-86457dfbf8-x62g2 1/1       Running   0          1h        10.8.2.4    gke-standard-cluster-1-default-pool-3f1d246
1-h2d9
pod/zeppelinspark-worker-856486cb64-zhdkz 1/1       Running   0          1h        10.8.3.6    gke-standard-cluster-1-default-pool-3f1d246
1-tv83
pod/zeppelinspark-zeppelin-566947c958-756zb 1/1     Running   0          1h        10.8.3.5    gke-standard-cluster-1-default-pool-3f1d246
1-tv83
```

Kubernetes and Google Cloud



Radiatus (A Big Data Analytics Platform)

It is a BDaaS platform over a PaaS which allows data scientist to setup and deploy Big Data Infrastructure

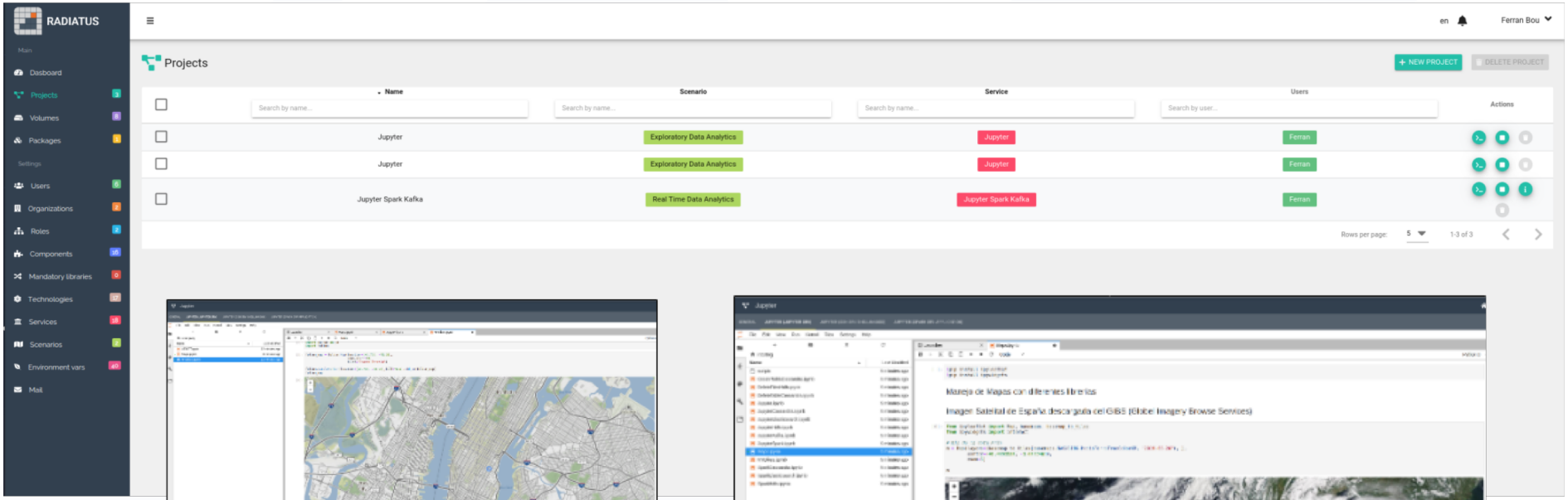
Create an efficient Big Data ecosystem to be executed on Cloud.

Simplify the service creation by automating the technological stack integration.

Focused on the Data Scientist to simplify the deployment procedure.


DRIFT

Radiatus (A Big Data Analytics Platform)



Demo

Radiatus (A Big Data Analytics Platform)

Big Data as a Service: A new approach for the SMEs

Muchas gracias por su atención

Ronal Muresano
rmuresano@gmail.com
rmuresano@iti.es

Windows Installation

Install docker windows

https://github.com/docker/toolbox/releases/tag/v18.09.3