Kubernetes Workshop

by USAC Authors presented by <Presenters Names...>

<UniversityName>,<Course Name>

#CNCFStudents

Part 1

Introduction

kubernetes

¿Qué es Kubernetes?

Es un orquestador de containers, que te permite manejar crear sistemas escalables implementando mejores prácticas de Cloud Native.

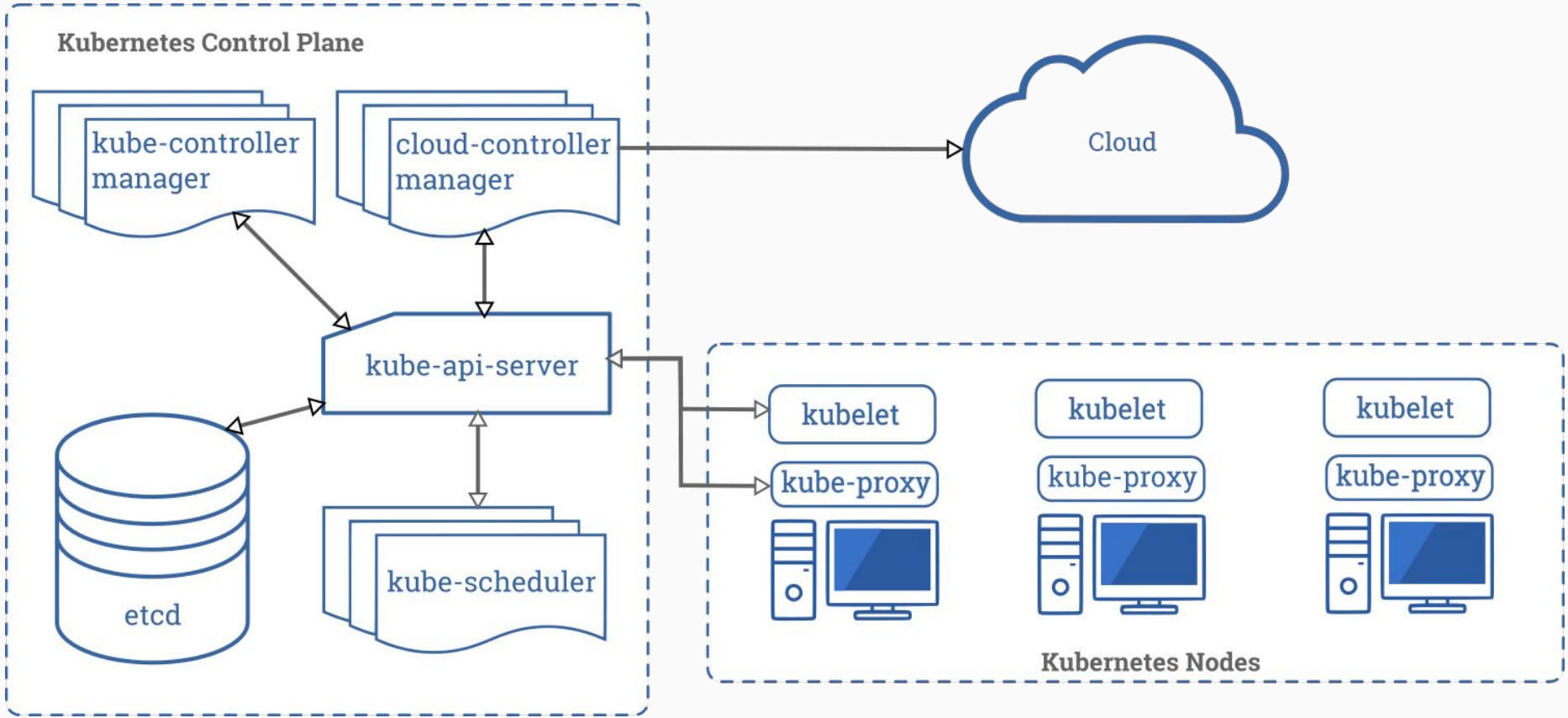
Anteriormente fue llamado Borg y era un proyecto de Google el cual fue liberado como open source.

Cluster nodes

Master Node: etcd, kube-api, kube-controller, kube-scheduler, kubectl, kubelet, core-dns, network-driver

Worker Nodes: kubelet, kubectl, kube-proxy

Client: kubectl



Ref: https://kubernetes.io/docs/concepts/overview/components/

Objects abstraction(Object types)

Pod: Varios containers agrupados en 1 pod

ReplicaSet: Mantener varias copias del container, no versionable

Deployment: Mantiene varias copias del container y es versionable

Init Container: Un container que se ejecuta cuando se crea un Pod

DaemonSet: Un pod que corre en todos los nodos del cluster

Objects abstraction(Object types)

Service: Una forma de exponer un objeto, sus tipos son ClusterIP, NodePort, LoadBalancer

Ingress: Una forma de exponer un servicio por un LoadBalancer capa 7

Config: Almacenamiento de configuraciones en formato clave-valor

Secret: Configuraciones de información sensibles, Config Encriptado

PersistentVolume: Reserva Espacio para almacenamiento

Objects abstraction(Object types)

ClusterRoles, Roles: Acceso a recursos por Namespace o Nodos

Taints: Tolerancias para ejecutar pods en nodos

Labels: Etiquetas en servicios

Ingress, Egress: Reglas de tráfico en Red

Namespace: Agrupar recursos en espacios de trabajo

HPA y CRD: Autoescalamiento de Servicios, tipos custom en K8s

Antes de empezar

1. Crear cuenta en GCP

2. Instalar y configurar gcloud

install gcloud

```
curl -O
https://dl.google.com/dl/cloudsdk/channels/rapid/downloads/google-cloud-sdk-297.0.0-linux-x86_64.tar.gz

tar zxvf [ARCHIVE_FILE] google-cloud-sdk

./google-cloud-sdk/install.sh

logout and login to reload the new gcloud command

gcloud init<<Seguir instrucciones de pantalla>>
```

Ref: https://cloud.google.com/sdk/docs/quickstart-linux

3. Crear reglas de firewall de entrada/salida para aplicar a los nodos del cluster

Instrucciones para crear reglas de firewall

Visitar esta página para crear la regla de firewall

https://console.cloud.google.com/networking/firewalls/add

4. Instalar kubectl

Instalar kubectl versión 1

https://kubernetes.io/docs/tasks/tools/install-kubectl/

```
wget curl -LO
https://storage.googleapis.com/kubernetes-release/release/v1.18.0/bin/linu
x/amd64/kubectl

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin/kubectl
```

Instalar kubectl con gcloud versión 2

```
gcloud components install kubectl
```

5. Crear Cluster en Google

Crear Cluster

```
gcloud config set project yourproject

gcloud config set compute/zone us-central1-a

v1:  gcloud container clusters create k8s-demo --num-nodes=1
--tags=allin,allout --enable-legacy-authorization --enable-basic-auth
--issue-client-certificate --machine-type=n1-standard-2
--no-enable-network-policy

v2:  gcloud container clusters create k8s-demo --num-nodes=1
--tags=allin,allout --machine-type=n1-standard-2
--no-enable-network-policy
```

6. Configurar kubectl para acceder al cluster

Obtener credenciales para kubectl

Clusters->MyCluster->Details->Show Cluster Certificate

```
gcloud container clusters get-credentials k8s-demo --zone=us-central1-c
```

kubectl get nodes

Ref:
https://cloud.google.com/sdk/gcloud/reference/container/clusters/get-credentials

6. App demo

app demo instructions v1 cluster default

```
kubectl run mipod --image=nginx --restart=Never

kubectl get pods

kubectl port-forward pod/mipod 8080:80

kubectl delete pods mipod
```

Acceder el http://127.0.0.1:8080 en el browser

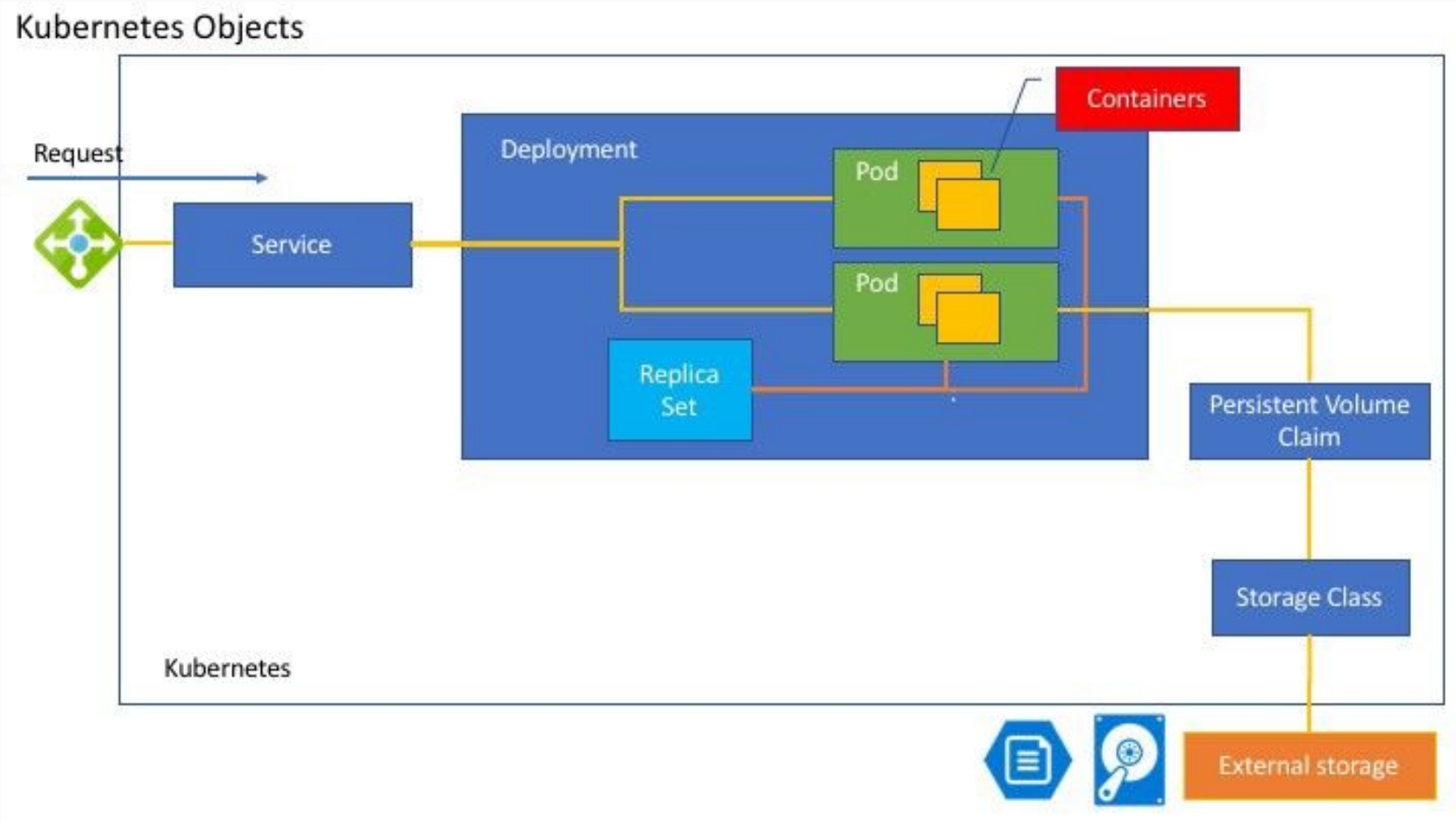
app demo instructions v2 cluster custom

```
kubectl run mipod --image=nginx --restart=Never

kubectl get pods

kubectl expose pod/mipod --target-port=80 --port=80 --type=LoadBalancer --name=mipod-svc

kubectl get services mipod-svc

kubectl get pods -o wide

kubectl get nodes -o wide

kubectl delete pods mipod;kubectl delete services mipod-svc

kubectl describe nodes NODE_NAME | grep ExternalIP
```

app demo instructions v3 cluster custom

```
kubectl run mipod --image=nginx --restart=Never

kubectl get pods

kubectl expose pod/mipod --target-port=80 --port=80 --type=NodePort
--name=mipod-svc

kubectl get services

kubectl get pods -o wide

kubectl get nodes -o wide

kubectl describe nodes NODE_NAME | grep ExternalIP
```

Kubernetes más a detalle



Ref: https://medium.com/@tsuyoshiushio/kubernetes-in-three-diagrams-6aba8432541c

Operaciones válidas con todos los objetos

```
kubectl get objetos

kubectl get objetos objeto

kubectl delete objetos objeto

kubectl describe objetos objeto

KUBE_EDITOR=nano kubectl edit
objetos objeto
```

```
kubectl get objetos objeto -o yaml
> objeto.yaml

kubectl create -f objeto.yaml

kubectl delete -f objeto.yaml

kubectl apply -f objeto.yaml
```

POD

-- create & update pods --

```
kubectl run podx --image=YOURIMAGE
--restart=Never
```

kubectl run podx --image=YOURIMAGE
***--restart=Never ***

--dry-run -o yaml > podx.yaml

```
kubectl create -f podx.yaml
```

```
kubectl apply -f podx.yaml
```

-- delete pods --

```
kubectl delete -f podx.yaml
```

```
kubectl delete pods podx
```

-- get all pods in default namespace --

```
kubectl get pods
```

-- show more information --

```
kubectl get pods -o wide
```

Estructura de un archivo YAML

- apiVersion
- kind
- metadata
- spec

```
== podx.yaml ==

apiVersion: v1
kind: Pod
metadata:
  labels:
    run: podx
  namespace: minamespace
  name: podx
spec:
  containers:
  - image: czdev/python-flask-distroless
    name: podx


kubectl apply -f podx.yaml
```

Comandos para agilizar tu trabajo

- `comando --dry-run`
- `comando -o yaml > archivo.yaml`
- `comando --dry-run -o yaml > archivo.yaml`

Namespaces

== NAMESPACES ==

```
kubectl create namespace minamespace

kubectl get ns

kubectl delete ns minamespace
```

== Namespaces por defecto ==

```
default

kube-public

kube-system
```

Namespaces

== minamespace.yaml ==

```
apiVersion: v1
kind: Namespace
metadata:
  name: minamespace
```

```
kubectl apply -f minamespace.yaml
```

Ejemplos con namespaces

```
kubectl run nginx --image=nginx --restart=Never -n minamespace

kubectl get pods -n minamespace
```

Nota: *Agregar -n minamespace para limitar la búsqueda de objetos a ese namespace*

Deployments

== DEPLOYMENTS ==

```
kubectl create deployment app1
--image=YOURIMAGE

kubectl get deployments

kubectl describe deployment app1

kubectl edit deployments app1

kubectl delete deployment app1

kubectl scale --replicas=3
deployment/app1

kubectl apply -f app1.yaml
```

Deployments

```
== app1.yaml ==

apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: app1
  name: app1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: app1
  template:
    metadata:
      labels:
        app: app1
    spec:
      containers:
      - image: nginx
        name: nginx


kubectl apply -f app1.yaml
```

Troubleshooting pods

```
kubectl run -it client --rm --image=busybox \

 --restart=Never -n minamespace -- sh

-- Access inside pod --

kubectl exec -it YOURPOD -n minamespace -- [bash|sh]

kubectl exec -it YOURPOD -c container -- [bash|sh]

kubectl describe pods YOURPOD -n mynamespace
```

Monitoring & TroubleShooting

== MONITOREO & TROUBLESHOOTING ==

```
kubectl logs -f pod/podname

kubectl logs -f deployment/deployname

kubectl describe deployments app1

kubectl run terminal --image=busybox --restart=Never --rm -it -n
mynamespace -- sh
```

Services

```
kubectl expose deployment app1 --port=80 --target-port=80 --type=ClusterIP
```

```
kubectl expose deployment app1 --port=80 --target-port=80 --type=NodePort
```

```
kubectl expose deployment app1 --port=80 --target-port=80
--type=LoadBalancer
```

Services

```
#edit service file

kubectl get services miservicio -o yaml > service.yaml

kubectl apply -f service.yaml

#set LoadBalancer

kubectl expose deployment app1 --port=80 --target-port=80
--type=LoadBalancer
```

Ejemplo de ClusterIP

Este crea un DNS interno para acceder un deployment o Pod por nombre

```
== service.yaml ==

apiVersion: v1
kind: Service
metadata:
  labels:
    run: mipod
  name: mipod-svc2
spec:
  ports:
  - port: 5000
    protocol: TCP
    targetPort: 5000
  selector:
    run: mipod
  type: ClusterIP

kubectl apply -f service.yaml
```

Ejemplo de NodePort

Se debe tomar en cuenta que con un ingress muchas veces es necesario hacer configuraciones de DNS

```
== service.yaml ==

apiVersion: v1
kind: Service
metadata:
  labels:
    run: mipod
  name: mipod-svc2
spec:
  ports:
  - port: 5000
    protocol: TCP
    targetPort: 5000
    nodePort: 31111
  selector:
    run: mipod
  type: NodePort

kubectl apply -f service.yaml
```

Ejemplo de Load Balancer

Se debe tomar en cuenta que con un ingress muchas veces es necesario hacer configuraciones de DNS

```
== service.yaml ==

apiVersion: v1
kind: Service
metadata:
  labels:
    run: mipod
  name: mipod-svc
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 5000
  selector:
    run: mipod
  type: LoadBalancer

kubectl apply -f service.yaml
```

Nodes

```
== NODOS ==

kubectl get nodes

kubectl get nodes -o wide

kubectl describe nodes

kubectl top nodes
```

Helm

```
== HELM ==

curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3

chmod 700 get_helm.sh

./get_helm.sh

helm repo add bitnami https://charts.bitnami.com/bitnami
```

Helm+Nginx Ingress

```
kubectl create ns nginx-ingress

helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

helm repo update

helm install nginx-ingress ingress-nginx/ingress-nginx -n nginx-ingress

helm list -n nginx

helm uninstall nginx-ingress -n nginx-ingress

kubectl get services -n nginx-ingress

https://kubernetes.github.io/ingress-nginx/deploy/#using-helm
```

Ingress Contour Instalación

```
== INGRESS ==

https://projectcontour.io/

kubectl apply -f https://projectcontour.io/quickstart/contour.yaml

https://kubernetes.io/docs/concepts/services-networking/ingress/
```

Ejemplo #1 Ingress

nginx-ingress

Se debe tomar en cuenta que con un ingress muchas veces es necesario hacer configuraciones de DNS

```
== ingress.yaml ==

kind: Ingress
apiVersion: networking.k8s.io/v1beta1
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
    ingress.kubernetes.io/rewrite-target: /
  name: api-ingress
  namespace: dev
spec:
  rules:
    - host: subdomain.domain.tld
      http:
        paths:
          - backend:
              serviceName: api-srv
              servicePort: YOUR_PORT

kubectl apply -f ingress.yaml
```

Ejemplo #2 Ingress

nginx-ingress

Se debe tomar en cuenta que con un ingress muchas veces es necesario hacer configuraciones de DNS

```
== ingress.yaml ==

apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /$1
  name: appx-ingress
spec:
  rules:
    - host: subdomain.domain.tld
      http:
        paths:
          - backend:
              serviceName: api-srv
              servicePort: YOUR_PORT
            path: /YOUR_PATH/(.*)
            pathType: Prefix

kubectl apply -f ingress.yaml
```

Ejemplo #3 Ingress

nginx-ingress

Se debe tomar en cuenta que con un ingress muchas veces es necesario hacer configuraciones de DNS

```
== ingress.yaml ==

apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /$1
  name: appx-ingress
spec:
  rules:
    - host: INGRESS_CONTROLLER_IP.nip.io
      http:
        paths:
          - backend:
              serviceName: api-srv
              servicePort: YOUR_PORT
            path: /YOUR_PATH/(.*)
            pathType: Prefix

kubectl apply -f ingress.yaml
```

Ejemplo ConfigMap

```
== config.yaml ==

apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  # property-like keys; each key maps to a simple value
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"

  # file-like keys
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true

kubectl apply -f config.yaml
```

Ejemplo ConfigMap

== config.yaml ==

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: configmap-demo-pod
spec:
  containers:
    - name: demo
      image: alpine
      command: ["sleep", "3600"]
      env:
        # Define the environment variable
        - name: PLAYER_INITIAL_LIVES  # Notice that the case is different here
                                      # from the key name in the ConfigMap.
          valueFrom:
            configMapKeyRef:
              name: game-demo           # The ConfigMap this value comes from.
              key: player_initial_lives  # The key to fetch.
        - name: UI_PROPERTIES_FILE_NAME
          valueFrom:
            configMapKeyRef:
              name: game-demo
              key: ui_properties_file_name
      volumeMounts:
      - name: config
        mountPath: "/config"
        readOnly: true
  volumes:
    # You set volumes at the Pod level, then mount them into containers inside that Pod
    - name: config
      configMap:
        # Provide the name of the ConfigMap you want to mount.
        name: game-demo
        # An array of keys from the ConfigMap to create as files
        items:
        - key: "game.properties"
          path: "game.properties"
        - key: "user-interface.properties"
          path: "user-interface.properties"
```

kubectl apply -f config.yaml

Secrets

```
kubectl create secret docker-registry secret-tiger-docker \
  --docker-username=tiger \
  --docker-password=pass113 \
  --docker-email=tiger@acme.com
```

Ejemplo Secrets

== config.yaml ==

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-sa-sample
  annotations:
    kubernetes.io/service-account.name: "sa-name"
type: kubernetes.io/service-account-token
data:
  # You can include additional key value pairs as you do with Opaque Secrets
  extra: YmFyCg==
```

kubectl apply -f ingress.yaml