Sesión 8 : kubernetes

devops, cursos, k8s

katacoda

- https://www.katacoda.com/
- CoreDNS : DNS simple para trabajar como editando `/etc/hosts` y lo que no, a google

kubernetes

- master
  - todo a través de la API
  - etcd (lo que más se muere en entorno multimaster)
    - en algunos casos pierde la coherencia, no se replica, hay que borrar los ficheros de etcd del nodo
  - scheduler: distribuye
  - controllers
  - google no instala cluster de más de 100 nodos, escalan con balanceadores
- namespaces:
  - comparten servicios
  - como un «stack»
- pod:
  - contenedores compartiendo IP (se comunican como «localhost»)
- secrets & configMaps
  - comparten (o pueden) volúmenes
- uso de etiquetas para relacionar los diferentes objetos
- redes:
  - swarm: si cmparten red, se ven
  - kubernetes: si comparte *namespace*, se ven
- conceptos kubernetes
  - deployment
    - apartado *template*: definición del *pod* o *pods*
    - estrategias de autoescalado
    - histórico (roll-back)
  - service
    - clusterIP
    - nodePort
    - LoadBalancer
  - kubelet:
    - agente en los nodos workers
    - obtiene el estado de los pods
    - healtcheck en Dockerfile (para repartir la carga)... si no lo pasa, se suicida
  - kube-proxy:
    - proxy de puertos en el nodo worker
  - volumeclaim
    - casa contendores/pods con los volúmenes
    - se puede afinar su asignación
  - Estratégias despliegue:
    - https://container-solutions.com/kubernetes-deployment-strategies/
    - RollingUpdate
    - Blue/Green

        - Canary
        - A/B testing
    - ventajas
        - autoescalado
        - gestión de volúmenes
        - gestión de red
        - docker-enterprise apuesta por kubernetes, futuro de docker-swarm?
        - kubernetes=linux, swarm=macintosh
    - deployment vs daemonset (swarm «global»)
        - daemontset → recolectores

labs

- https://training.play-with-kubernetes.com/
- helm.sh (no temario) : recetas (Charts) para desplegar sobre kubernetes
    - microsoft/azure/google...
    - gitlab using helm
- duffle = creación de kubernetes + helm
- traefik kubernetes = https://docs.traefik.io/user-guide/kubernetes/
    - istio
    - kong = capa + plugin ram lua + nginx

katacoda

lab1

- https://www.katacoda.com/courses/kubernetes/launch-single-node-cluster
    - `minikube version`
    - `minikube start`
    - `kubectl cluster-info`
    - `kubectl get nodes`
    - `kubectl run first-deployment –image=katacoda/docker-http-server –port=80` → `kubectl create`
    - `kubectl expose deployment first-deployment –port=80 –type=NodePort`
    - `kubectl describe deployments`
    - `export PORT=$(kubectl get svc first-deployment -o go-template='range.spec.portsif_.nodeportnodeportnendend'); echo «Accessing host01:$PORT»; curl host01:$PORT`

lab2

- https://www.katacoda.com/courses/kubernetes/getting-started-with-kubeadm
    - kubeadm : aprovisionamiento de nodos
    - `kubeadm init –token=102952.1a7dd4cc8d1f4cc5 –kubernetes-version $(kubeadm version -o short)` ← en producción no pasar el token para se genere
    - `sudo cp /etc/kubernetes/admin.conf $HOME/; sudo chown $(id -u):$(id -g) $HOME/admin.conf; export KUBECONFIG=$HOME/admin.conf` : copia certificados y configuración en el $HOME del usuario en curso para su uso
    - `kubeadm token list`
    - `kubeadm join –discovery-token-unsafe-skip-ca-verification –token=102952.1a7dd4cc8d1f4cc5 172.17.0.44:6443`
        - **–discovery-token-unsafe-skip-ca-verification** : bybass Discovery Token verification

- `kubectl get nodes` (on master)
- CNI: Container Network Interface : networks providers
- `kubectl apply -f /opt/weave-kube` : deploy del WeaveWorks
  - https://www.weave.works/docs/net/latest/kube-addon/
- `kubectl get pod -n kube-system`
- `kubectl create deployment http –image=katacoda/docker-http-server:latest`
- `kubectl get pods`
- `docker ps | grep docker-http-server` (on node)
- `kubectl apply -f dashboard.yaml`
- `kubectl get pods -n kube-system`
- Creacion **ServiceAccount**:

```
cat <<EOF | kubectl create -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kube-system
EOF
```

- get token: `kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk «{print $1}»)`
- `kubeadm token list`

lab3

- https://www.katacoda.com/courses/kubernetes/kubectl-run-containers
  - `minikube start`
  - `kubectl get nodes`
  - `kubectl run http –image=katacoda/docker-http-server:latest –replicas=1`
  - `kubectl get deployments`
  - `kubectl describe deployment http`
  - `kubectl expose deployment http –external-ip=«172.17.0.28» –port=8000 –target-port=80`
  - en un solo comando: `kubectl run httpexposed –image=katacoda/docker-http-server:latest –replicas=1 –port=80 –hostport=8001` → expone el puerto a nivel de Docker, por lo tanto no se muestra así: `kubectl get svc`, si no, así: `docker ps | grep httpexposed`
  - escalar contenedores:
    - `kubectl scale –replicas=3 deployment http`
      - cada nuevo pod creado se añade al LB
    - `kubectl get pods` : lista de pods en ejecución

    - `kubectl describe svc http` : muestra, entre otras cosas, los **endpoints**

lab4

- https://www.katacoda.com/courses/kubernetes/creating-kubernetes-yaml-definitions
    - deployment.yaml

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: webapp1
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: webapp1
    spec:
      containers:
      - name: webapp1
        image: katacoda/docker-http-server:latest
        ports:
        - containerPort: 80
```

    - `kubectl create -f deployment.yaml`
    - `kubectl get deployment` : muestra todos los deploys
    - `kubectl describe deployment webapp1` : descripción de un deploy concreto

    - service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: webapp1-svc
  labels:
    app: webapp1
spec:
  type: NodePort
  ports:
  - port: 80
    nodePort: 30080
  selector:
    app: webapp1
```

    - `kubectl create -f service.yaml`
    - `kubectl get svc`
    - `kubectl describe svc webapp1-svc`
    - (modificación de las réplicas del deployment.yaml) → `kubectl apply -f deployment.yaml`
    - `kubectl get deployment`, `kubectl get pods`

lab5

- https://www.katacoda.com/courses/kubernetes/guestbook
  - https://github.com/katacoda/kubernetes-guestbook.git
  - arrancan un cluster k8s con 1 master y 1 nodo:
    - API
    - Master
    - Proxy
    - DNS
  - k8s service deployment:
    - replication controller: cuantas instancias, la imagen docker a usar, el nombre que lo identifica (más otras de configuración y discovery)
      - `kubectl create -f redis-master-controller.yaml`
      - redis-master-controller.yaml

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: redis-master
  labels:
    name: redis-master
spec:
  replicas: 1
  selector:
    name: redis-master
  template:
    metadata:
      labels:
        name: redis-master
    spec:
      containers:
      - name: master
        image: redis:3.0.7-alpine
        ports:
        - containerPort: 6379
```

    - service:
      - load balancer que reenvia el tráfico a 1 o más contenedores (aunque estén en nodos diferentes)
      - comunica dentro del clustes, rara vez expone puertos al exterior
      - para comunicarnos desde fuera del cluster se recomienda el uso de un LoadBalancer
      - `kubectl create -f redis-master-service.yaml`
      - redis-master-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: redis-master
  labels:
    name: redis-master
spec:
  ports:
```

```
    # the port that this service should serve on
  - port: 6379
    targetPort: 6379
  selector:
    name: redis-master
```

- kubectl get services
- kubectl describe services redis-master

- replication slave pods
  - http://redis.io/topics/replication
  - uso de variables de entorno para el descubrimiento de otros PODs
  - kubectl create -f redis-slave-controller.yaml
  - redis-slave-controller.yaml

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: redis-slave
  labels:
    name: redis-slave
spec:
  replicas: 2
  selector:
    name: redis-slave
  template:
    metadata:
      labels:
        name: redis-slave
    spec:
      containers:
      - name: worker
        image: gcr.io/google_samples/gb-redisslave:v1
        env:
        - name: GET_HOSTS_FROM
          value: dns
          # If your cluster config does not include a dns
service, then to
          # instead access an environment variable to find the
master
          # service's host, comment out the 'value: dns' line
above, and
          # uncomment the line below.
          # value: env
        ports:
        - containerPort: 6379
```

  - kubectl get rc
- service redis slaves
  - kubectl create -f redis-slave-service.yaml
  - redis-slave-service.yaml

```
apiVersion: v1
```

```
kind: Service
metadata:
  name: redis-slave
  labels:
    name: redis-slave
spec:
  ports:
    # the port that this service should serve on
  - port: 6379
  selector:
    name: redis-slave
```

- kubectl get services
- **PQ NO TIENE UN TARGETPORT COMO EL REDIS-MASTER-SERVICE.YAML**

◦ deploy de la web app

- kubectl create -f frontend-controller.yaml

- frontend-controller.yaml

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: frontend
  labels:
    name: frontend
spec:
  replicas: 3
  selector:
    name: frontend
  template:
    metadata:
      labels:
        name: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
        env:
        - name: GET_HOSTS_FROM
          value: dns
          # If your cluster config does not include a dns
service, then to
          # instead access environment variables to find
service host
          # info, comment out the 'value: dns' line above, and
uncomment the
          # line below.
          # value: env
        ports:
        - containerPort: 80
```

- kubectl get rc
- kubectl get pods

◦ hacemos accesible el frontend

        - NodePort: expone un puerto
        - `kubectl create -f frontend-service.yaml`

        - frontend-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    name: frontend
spec:
  # if your cluster supports it, uncomment the following to
automatically create
  # an external load-balanced IP for the frontend service.
  # type: LoadBalancer
  type: NodePort
  ports:
    # the port that this service should serve on
    - port: 80
      nodePort: 30080
  selector:
    name: frontend
```

        - `kubectl get services`
    - estado de los pods: `kubectl get services`
    - por si no hemos asignado el puerto y queremos ver cual ha sido asignado: `kubectl describe service frontend | grep NodePort`

otros

- abreviaturas:
    - kubernetes = k8s
    - internationalization = i18n
    - localization = l10n
    - https://wiki.mageia.org/en/What_is_i18n,_what_is_l10n
- minikube
- komposer (swarm - k8s)
- tibco

From:
https://matebcn.eu/ - **miguel angel torres egea**

Permanent link:
https://matebcn.eu/info:cursos:pue:devops:sesion8

Last update: **08/04/2020 15:57**