ackstorm

Kubernetes Training

2019 / Barcelona

kubernetes

Advanced Controllers

1.- ReplicationController: A ReplicationController ensures that a specified number of pod replicas are running at any time. ReplicationController is one of the first controllers. It now is deprecated, and has been replaced by ReplicaSet.

2.- ReplicaSet is the next-generation Replication Controller. The only difference between a ReplicaSet and a Replication Controller is the selector support. ReplicaSet supports set-based selector requirements, whereas a Replication Controller only supports equality-based selector requirements.

3.- Deployments are the default way of creating pods. Using any other type of controller will require use case specifications. Just like a ReplicaSet, it ensures that a specified number of pod replicas are running at any time.

Note: Deployments do not create pods. They rather create ReplicaSets that create the pods.

4.- StatefulSets are objects used to manage stateful applications. They manage the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods; that are created from the same spec, but are not interchangeable since each has a persistent identifier that it maintains across any rescheduling.

5.- A DaemonSet ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected.

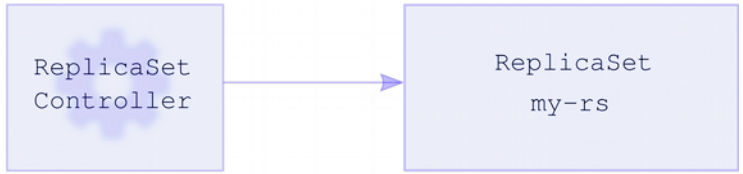
6.- A job creates one or more pods and ensures that a specified number of them successfully terminate. As pods successfully complete, the job tracks the successful completions. When a specified number of successful completions is reached, the job itself is complete.

7.- A CronJob creates Jobs on a time-based schedule. A CronJob object is like one line of a crontab (cron table) file. It runs a job periodically on a given schedule, written in Cron format. A CronJob creates a job object about once per execution time of its schedule.

ReplicaSet

```
name: my-rs
replicas: 3
Template:
```

Cluster

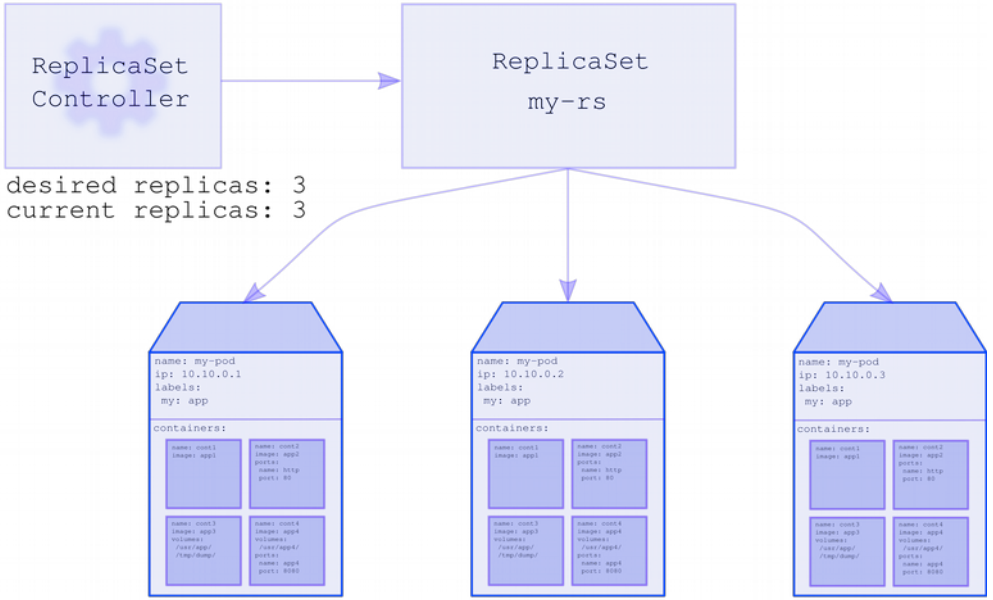



```
desired replicas: 3
current replicas: 0
```

ReplicaSet

name: my-rs
replicas: 3
Template:

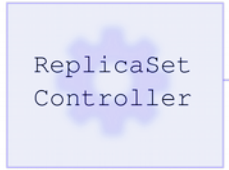
Cluster



ReplicaSet

```
name: my-rs
replicas: 3
Template:
```

Cluster

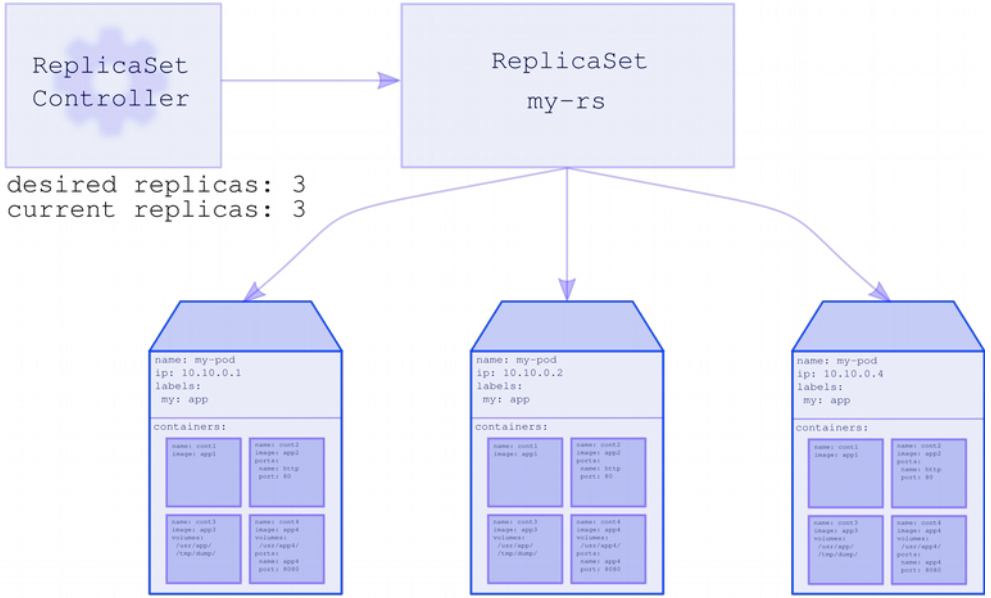



```
desired replicas: 3
current replicas: 2
```

ReplicaSet

```
name: my-rs
replicas: 3
Template:
```

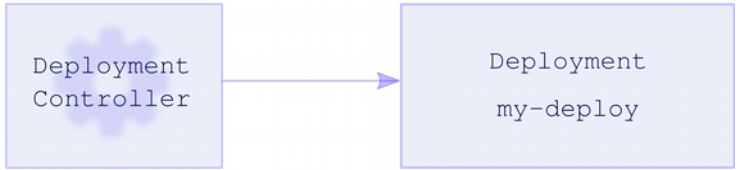
Cluster



Deployment

```
name: my-deploy
replicas: 3
strategy: RollingUpdate
Template:
```

Cluster

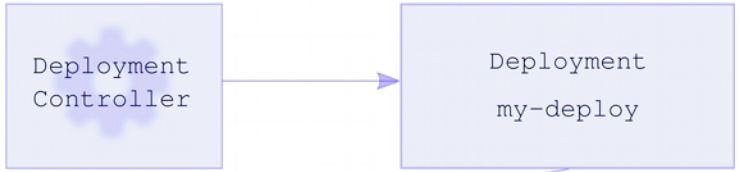



```
desired replicas: 3
current replicas: 0
```

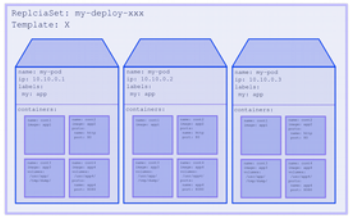
Deployment

```
name: my-deploy
replicas: 3
strategy: RollingUpdate
Template:
```

Cluster



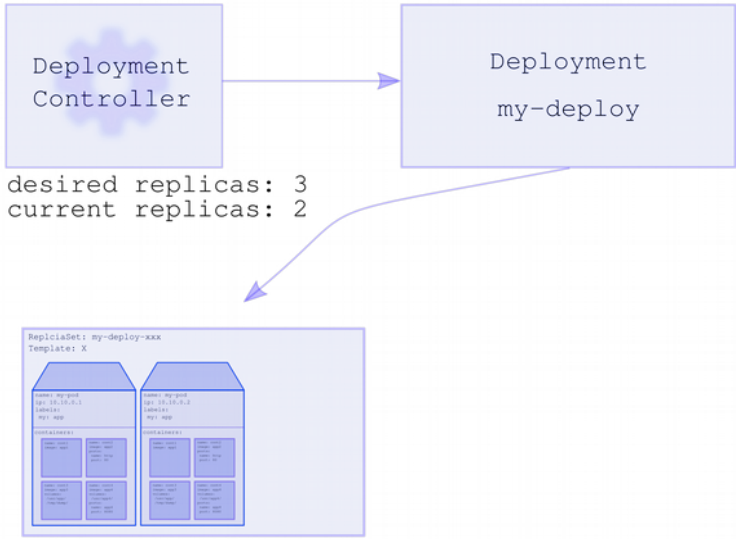
desired replicas: 3
current replicas: 3



Deployment

```
name: my-deploy
replicas: 3
strategy: RollingUpdate
Template:
```

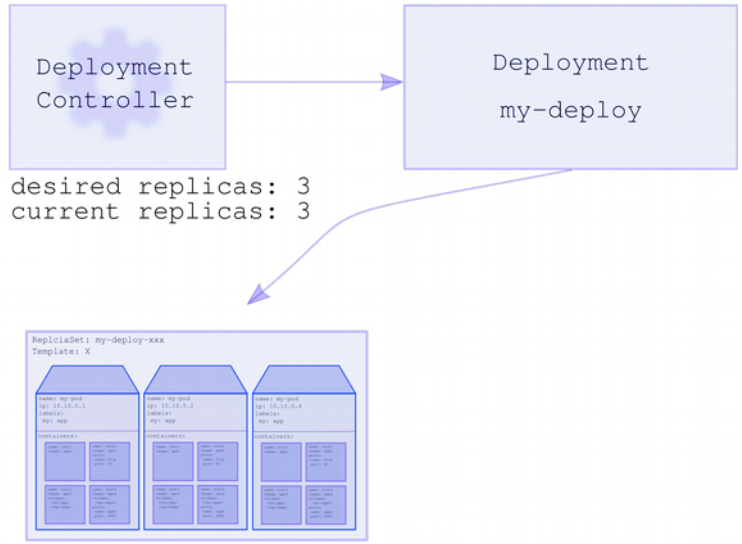
Cluster



Deployment

```
name: my-deploy
replicas: 3
strategy: RollingUpdate
Template:
```

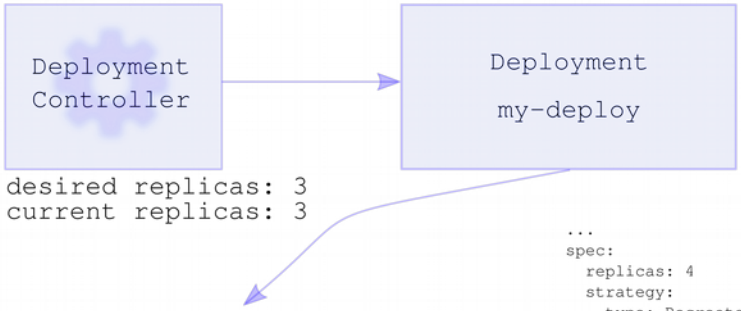
Cluster



Deployment

```
name: my-deploy
replicas: 3
strategy: RollingUpdate
Template:
```

Cluster



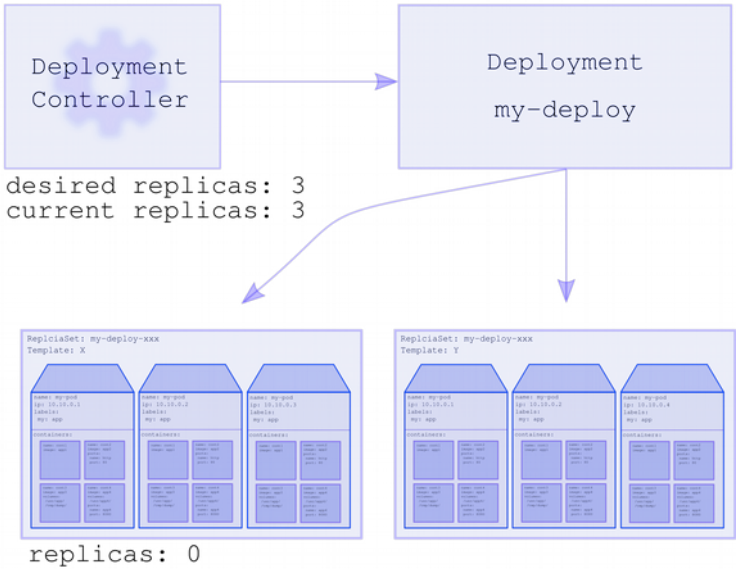
An update is triggered every time anything changes in the Pod template.

```
...
spec:
  replicas: 4
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: nginx
        new: label
    spec:
      containers:
      - image: nginx
        name: nginx
        ports:
        - containerPort: 80
          name: http
```

Deployment

```
name: my-deploy
replicas: 3
strategy: RollingUpdate
Template:
```

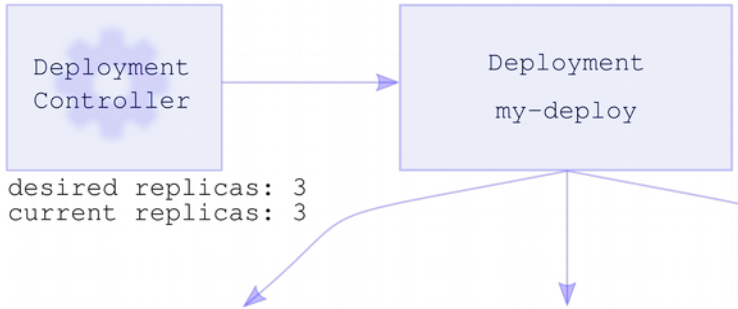
Cluster



Deployment

```
name: my-deploy
replicas: 3
strategy: RollingUpdate
Template:
```

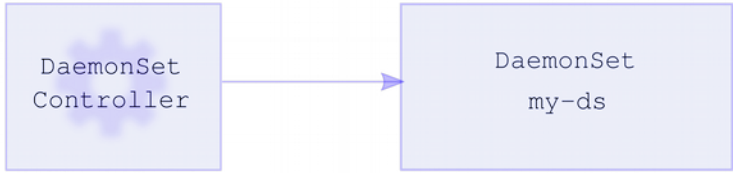
Cluster



DaemonSet

name: my-ds
Template:

Cluster

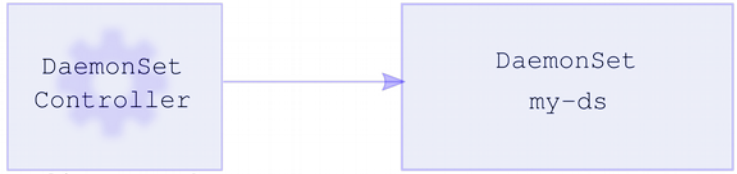


replicas = nodes

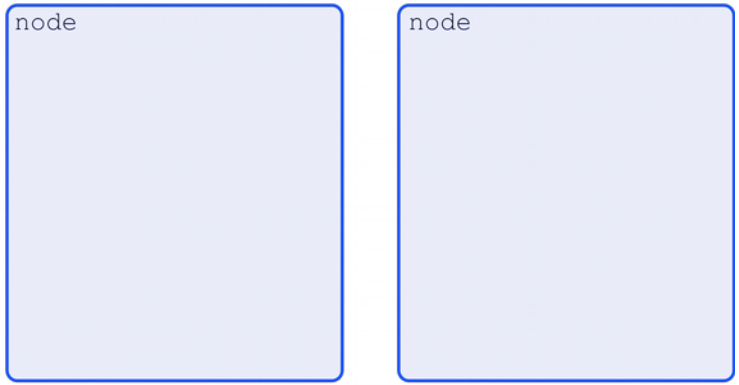
DaemonSet

name: my-ds
Template:

Cluster

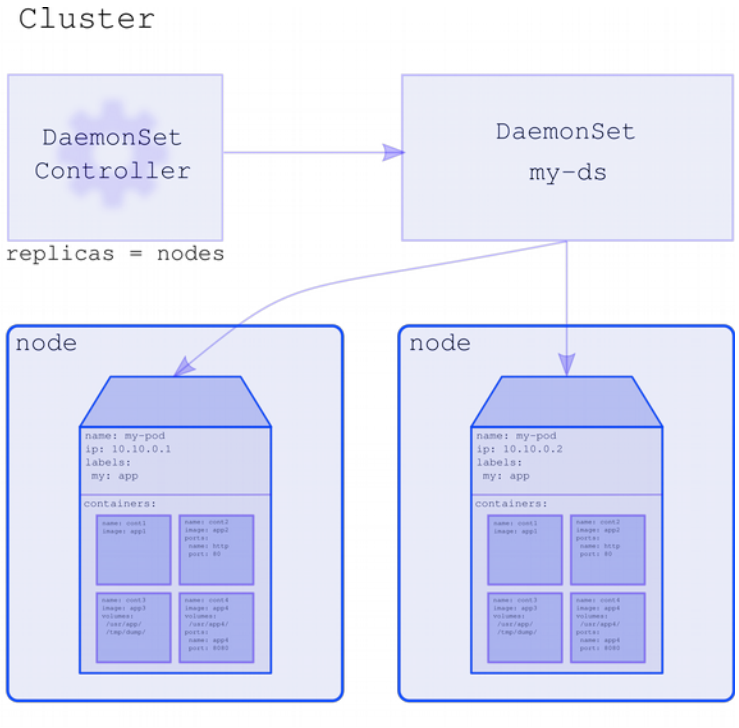


replicas = nodes



DaemonSet

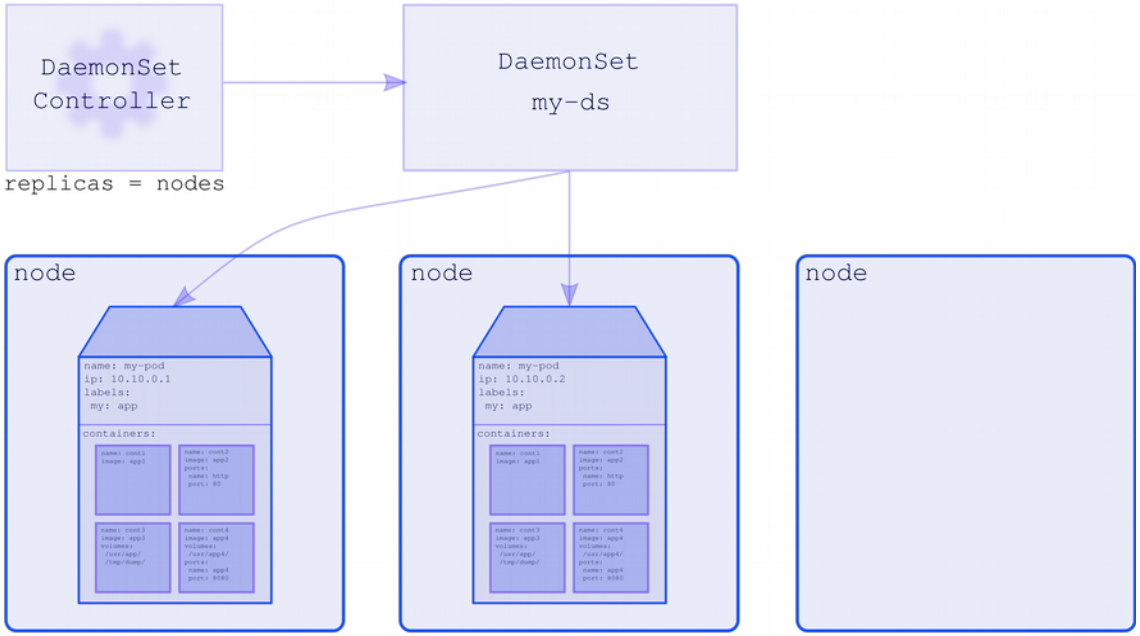
name: my-ds
Template:



DaemonSet

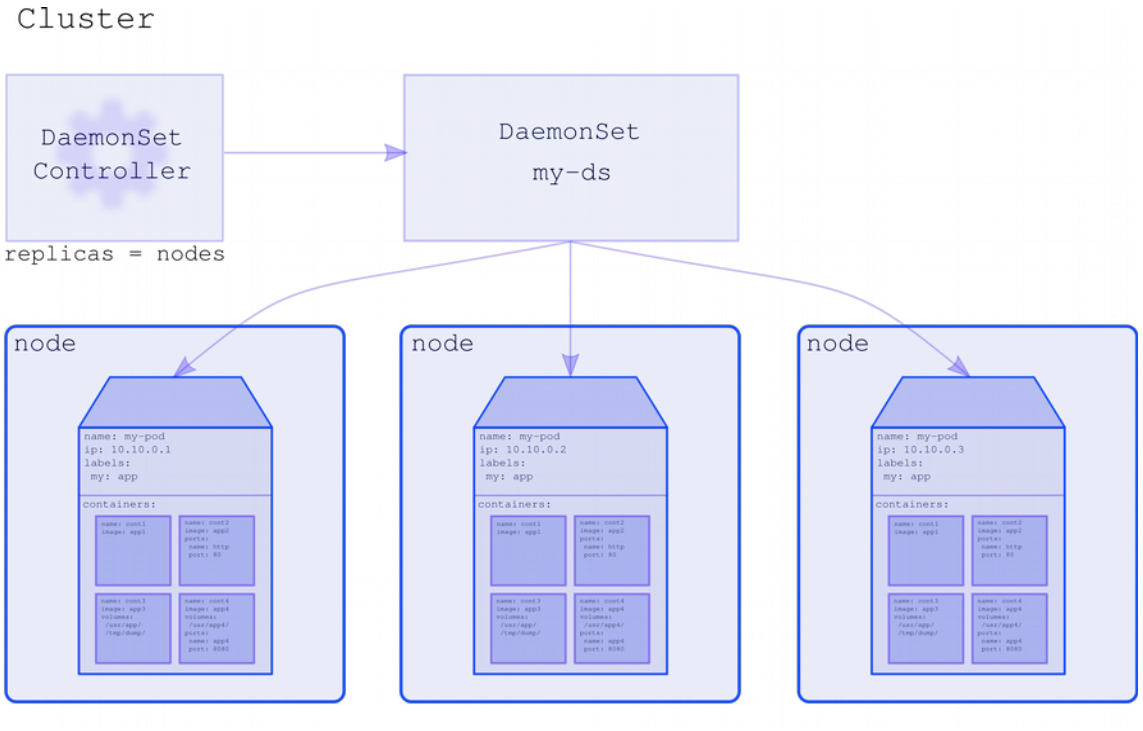
name: my-ds
Template:

Cluster



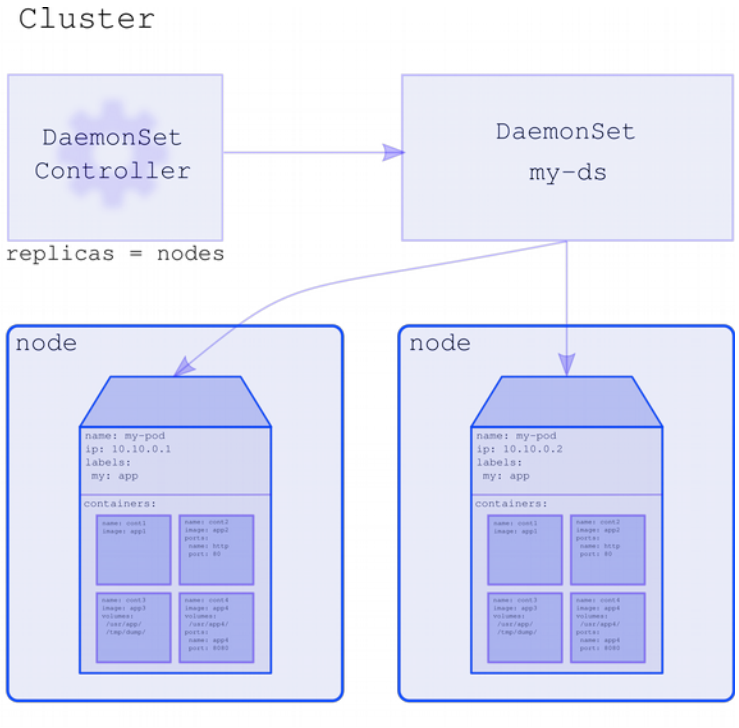
DaemonSet

name: my-ds
Template:



DaemonSet

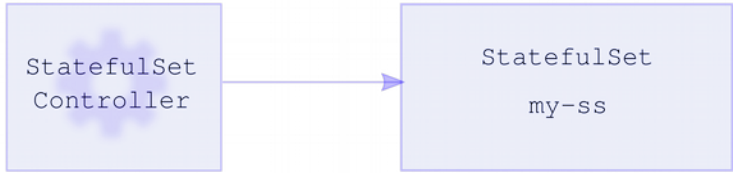
name: my-ds
Template:



StatefulSet

```
name: my-ss
replicas: 3
Template:
```

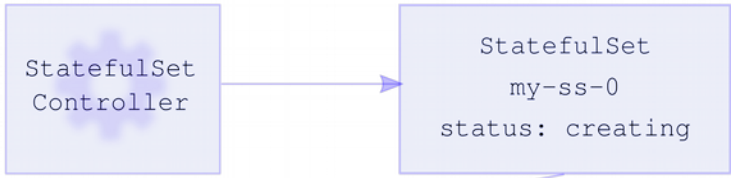
Cluster



StatefulSet

name: my-ss
replicas: 3
Template:

Cluster



StatefulSet

name: my-ss
replicas: 3
Template:

Cluster

t=0

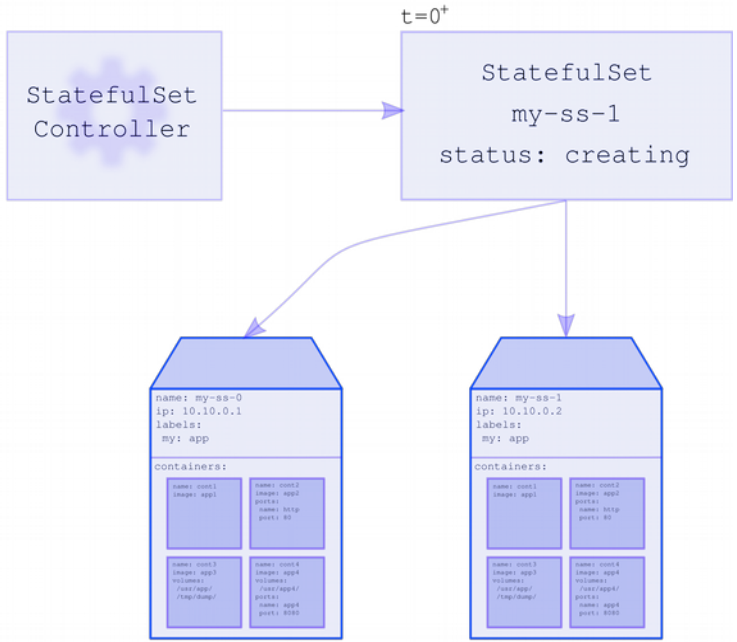
StatefulSet Controller

StatefulSet
my-ss-0
status: running

StatefulSet

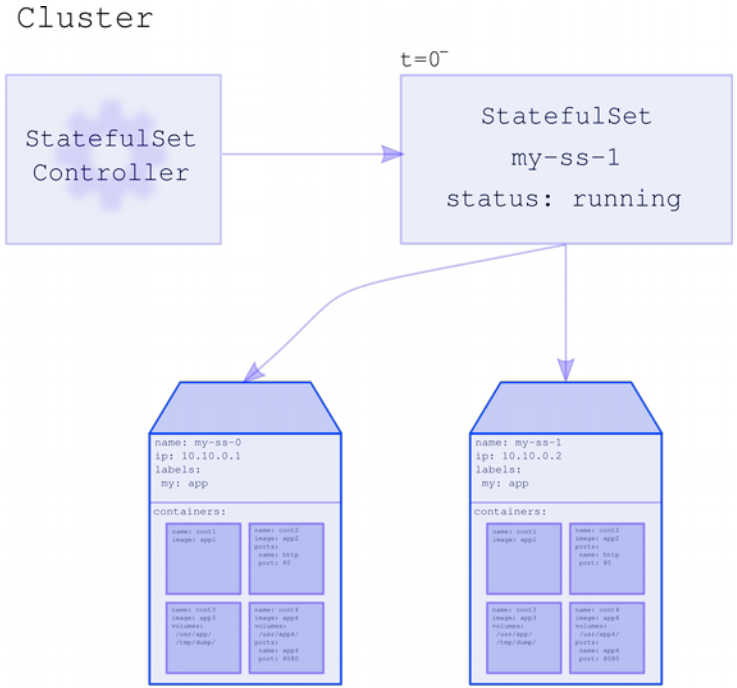
name: my-ss
replicas: 3
Template:

Cluster



StatefulSet

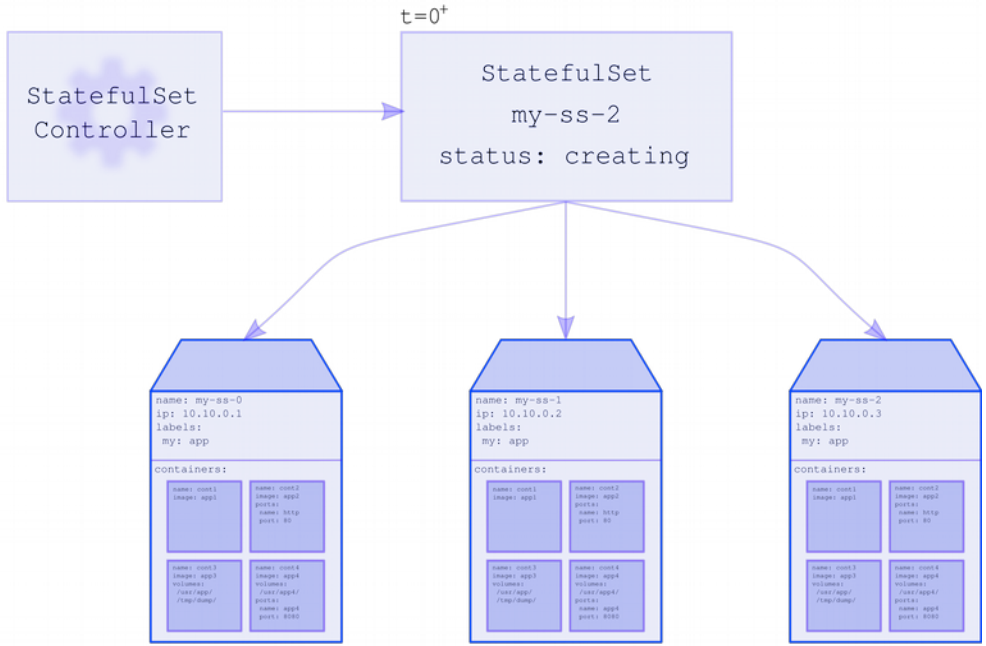
name: my-ss
replicas: 3
Template:



StatefulSet

name: my-ss
replicas: 3
Template:

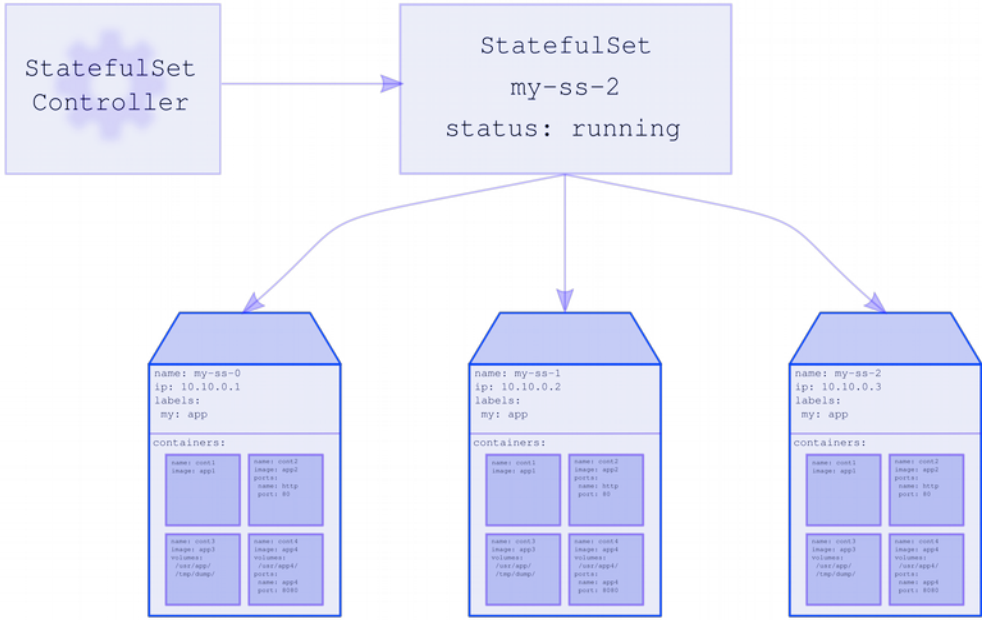
Cluster



StatefulSet

name: my-ss
replicas: 3
Template:

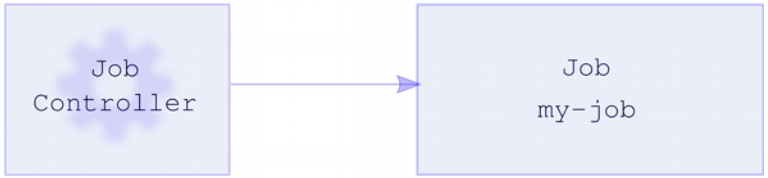
Cluster



Job/CronJob

```
name: my-job
backoffLimit: 5
activeDeadlineSeconds: 60
completions: 1
parallelism: 1
Template:
```

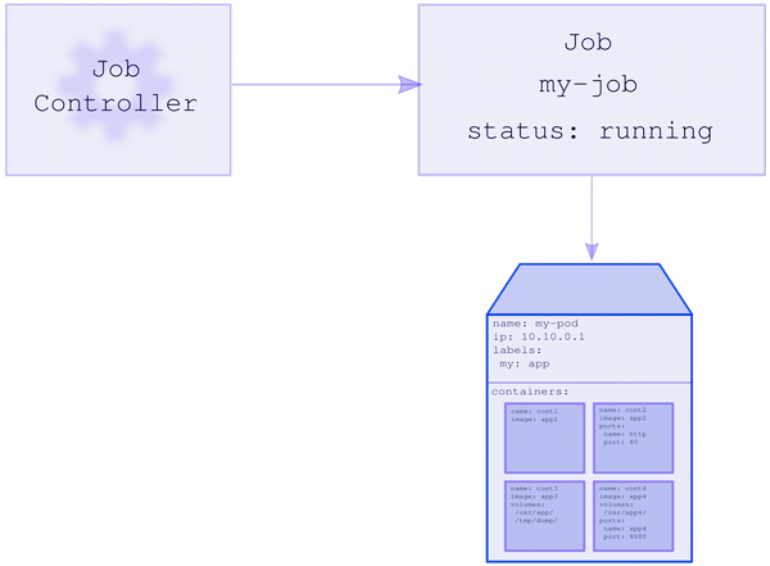
Cluster



Job/CronJob

```
name: my-job
backoffLimit: 5
activeDeadlineSeconds: 60
completions: 1
parallelism: 1
Template:
```

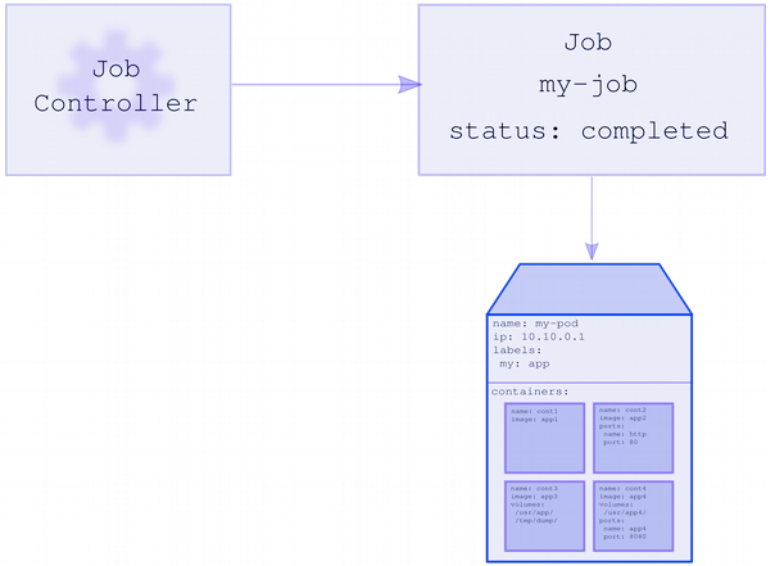
Cluster



Job/CronJob

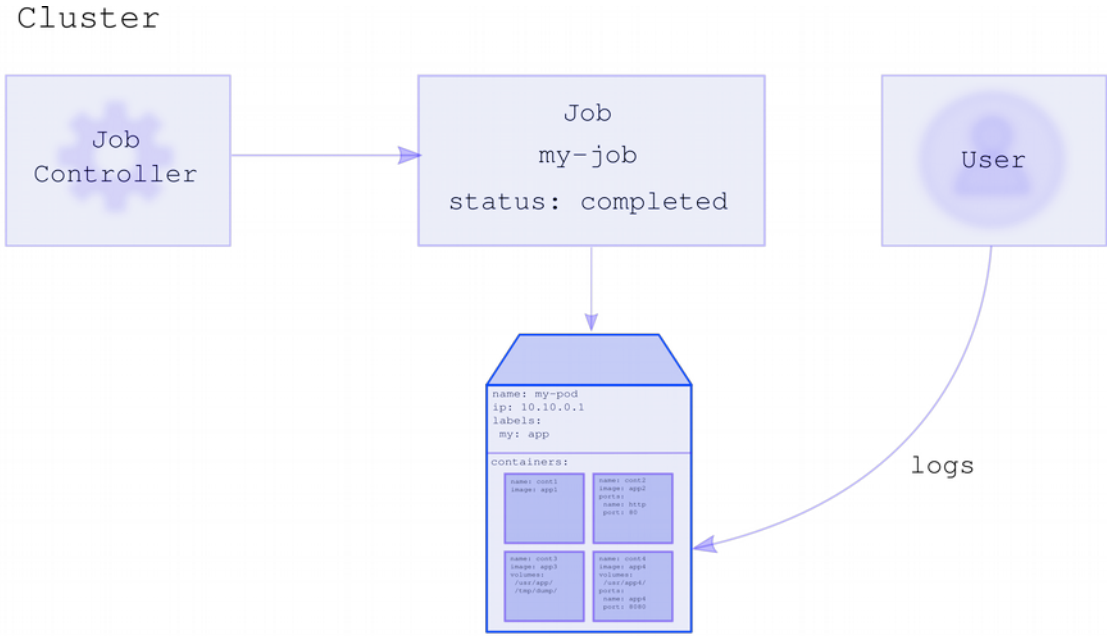
```
name: my-job
backoffLimit: 5
activeDeadlineSeconds: 60
completions: 1
parallelism: 1
Template:
```

Cluster



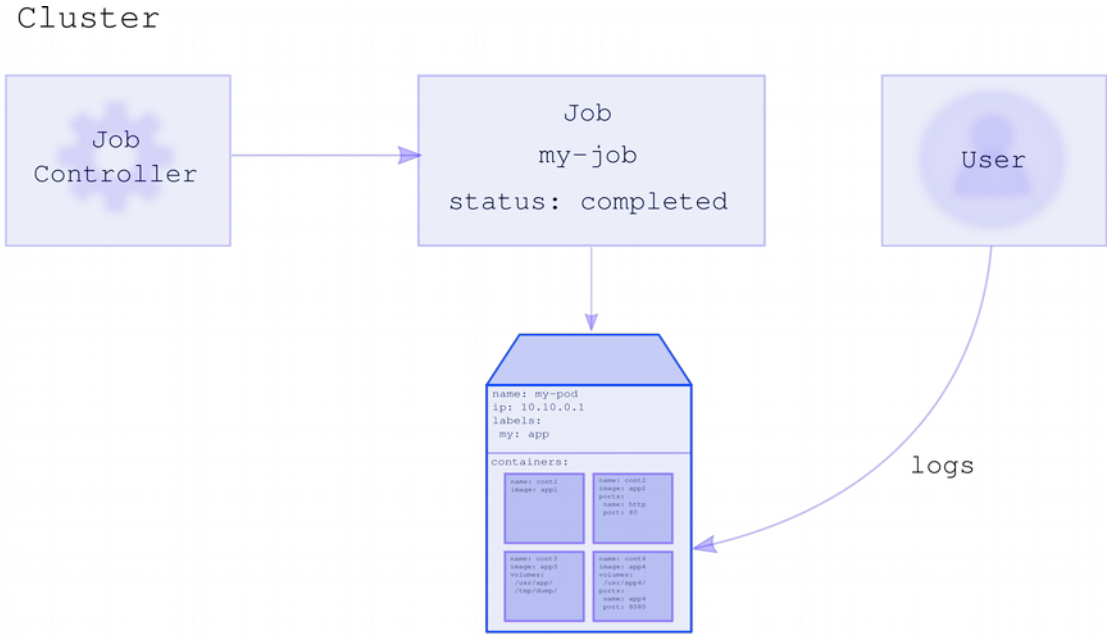
Job/CronJob

```
name: my-job
backoffLimit: 5
activeDeadlineSeconds: 60
completions: 1
parallelism: 1
Template:
```



Job - non-parallel

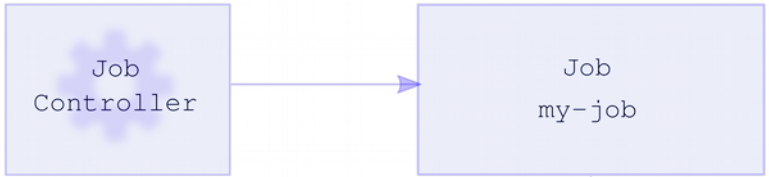
```
name: my-job
backoffLimit: 5
activeDeadlineSeconds: 60
completions: 1
parallelism: 1
Template:
```



Job - parallel, with fixed completion

```
name: my-job
backoffLimit: 5
activeDeadlineSeconds: 60
completions: X
parallelism: Y
Template:
```

Cluster



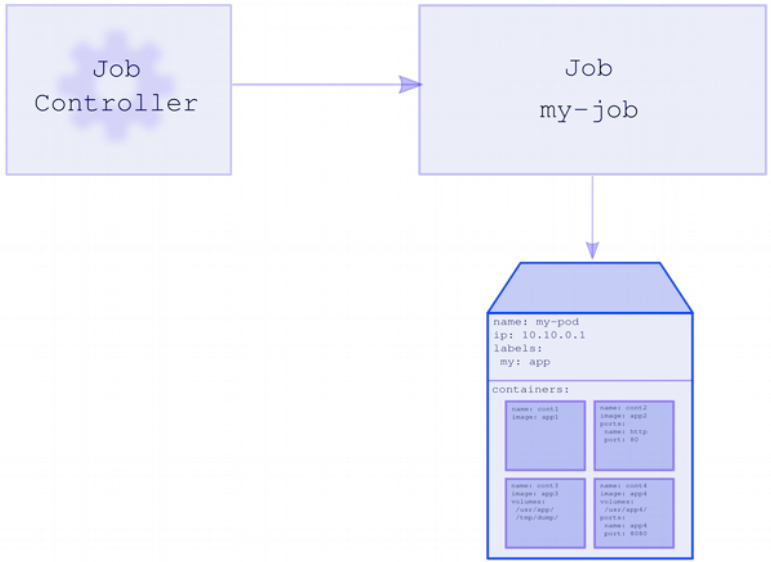
The Job will be considered finished, when X pods successfully complete.

Job Controller will spin up one Job; with Y pods in parallel, when running in this mode

Job - parallel, with work queue

```
name: my-job
backoffLimit: 5
activeDeadlineSeconds: 60
completions: null
parallelism: > 0
Template:
```

Cluster



To run in this mode, it requires running a queue service.

Each pod is independent to others.

Several outcomes are possible (depending on the app).

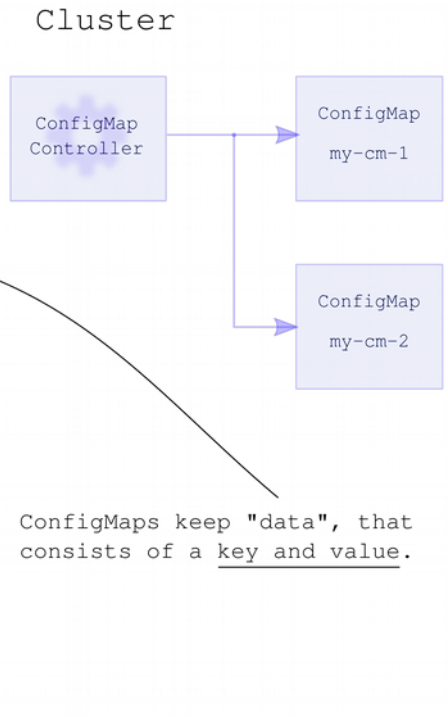
Lab4

1.- Create a Deployment with deploy.yaml file.
    $ kubectl create -f deploy.yaml
2.- Create a LoadBalancer type service with svc-lb.yaml.
    $ kubectl create -f svc-lb.yaml
    Note: Observe the particularity of the service.
3.- Access each pod, go to /usr/share/nginx/html, and change the content of indeh.html to "pod-1" and "pod-2".
    $ kubectl exec -it POD bash
    $ echo "pod-X" > /usr/share/nginx/html/index.html
4.- SSH into the VM called lab4
    $ gcloud compute ssh lab4 --zone ZONE
5.- Run a for loop with 20 requests to the IP address of the load balancer.
    $ for i in {1..20}; \
      do curl IP_ADDR; \
      done;
6.- Delete all the pods
    $ kubectl delete pod --all
    Note: Observe K8s reaction
7.- To clean up, delete the deployment
    $ kubectl delete deploy nginx

8.- Create a Job, with job.yaml
    $ kubectl create -f job.yaml
9.- When the Job has completed, check the logs
    $ kubectl logs POD
    Note: What do you think it is?

ConfigMap

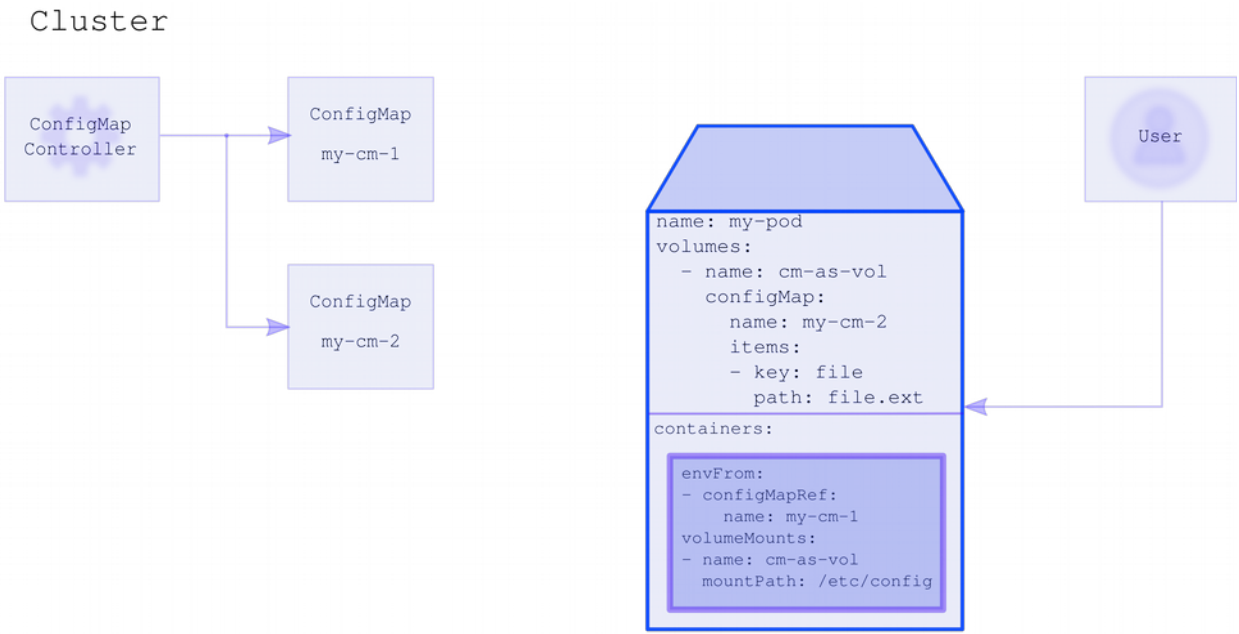
```
name: my-cm-1
data:
 CITY: BARCELONA
 ACKSTORM: TRUE
---
name: my-cm-2
data:
 file: my_data
```



ConfigMaps allow to decouple config artifacts from image content to keep containerized applications portable.

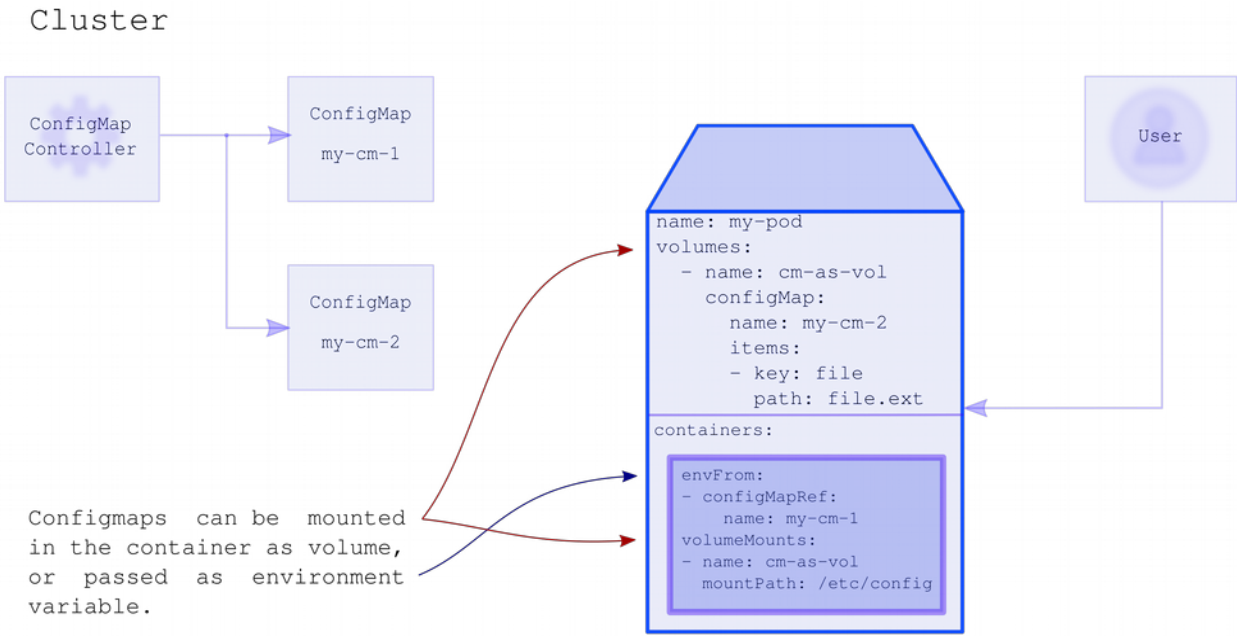
ConfigMap

```
name: my-cm-1
data:
 CITY: BARCELONA
 ACKSTORM: TRUE
---
name: my-cm-2
data:
 file: my_data
```



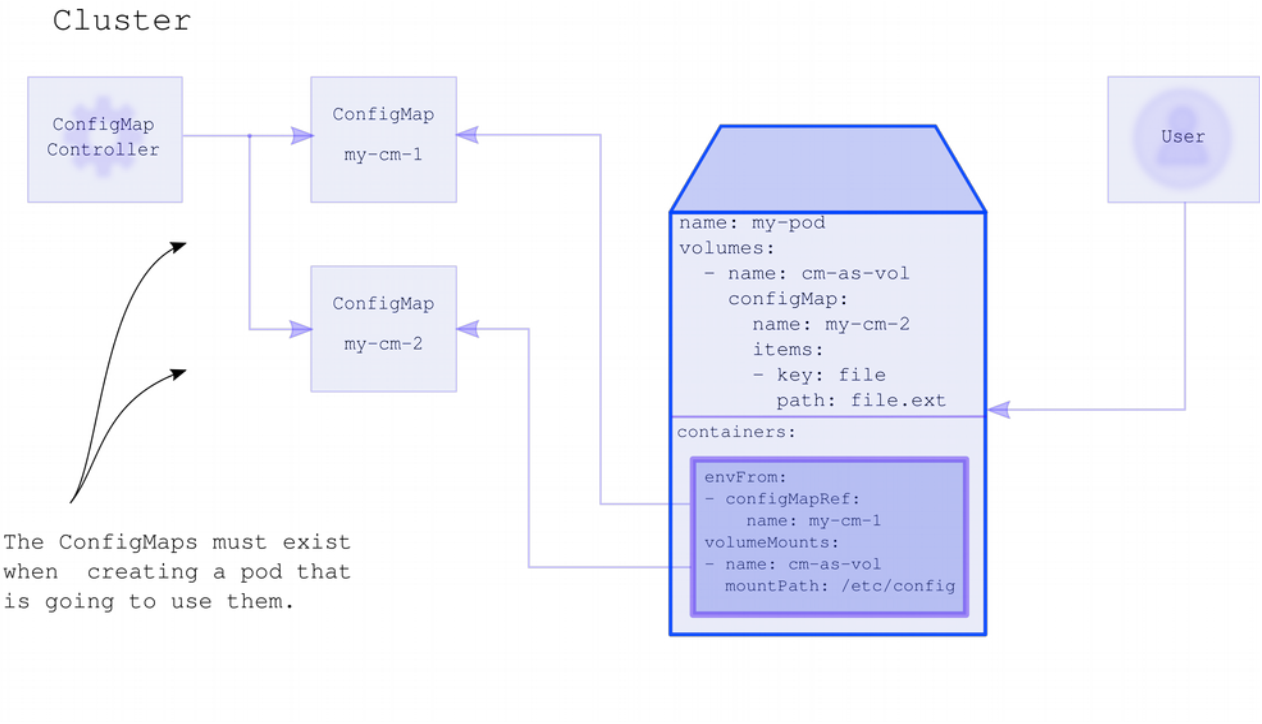
ConfigMap

```
name: my-cm-1
data:
 CITY: BARCELONA
 ACKSTORM: TRUE
---
name: my-cm-2
data:
 file: my_data
```



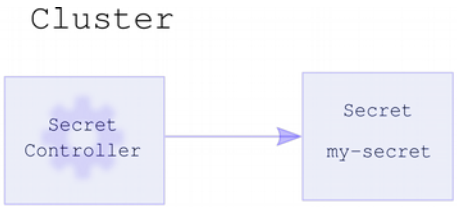
ConfigMap

```
name: my-cm-1
data:
 CITY: BARCELONA
 ACKSTORM: TRUE
---
name: my-cm-2
data:
 file: my_data
```



Secret

```
name: my-secret
data:
 username: user
 password: pass
```



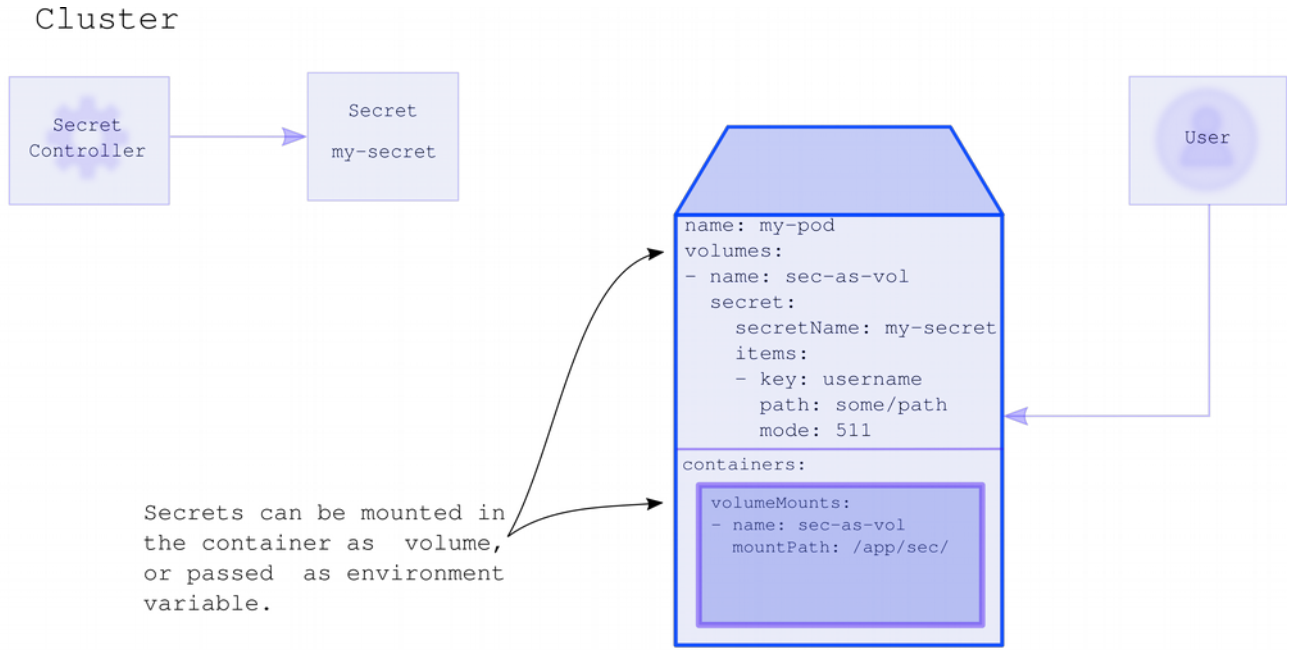
Secret objects allow to store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys.

Similar to ConfigMaps, Secrets too allow decoupling some data from the application.

By decoupling sensitive data from the image or Pod specifications, we gain more control over how is it used.

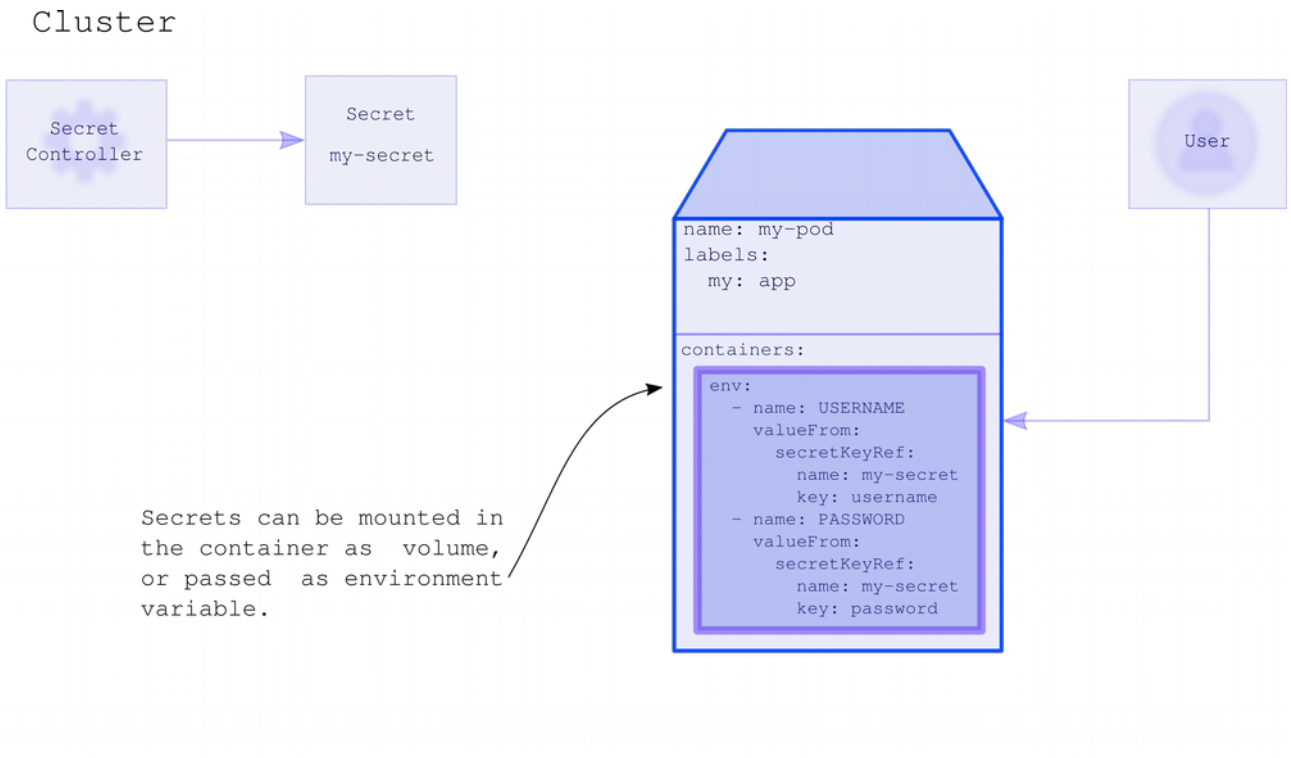
Secret

```
name: my-secret
data:
 username: user
 password: pass
```



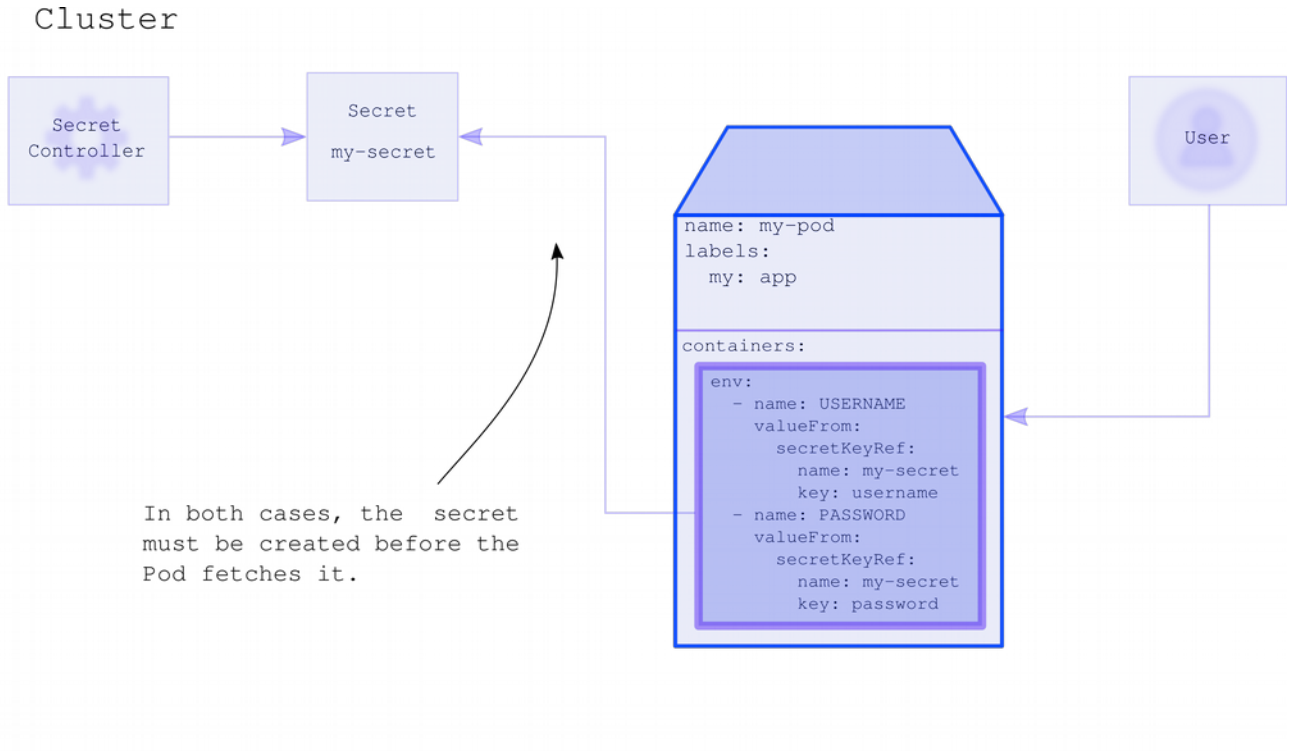
Secret

```
name: my-secret
data:
 username: user
 password: pass
```



Secret

```
name: my-secret
data:
 username: user
 password: pass
```

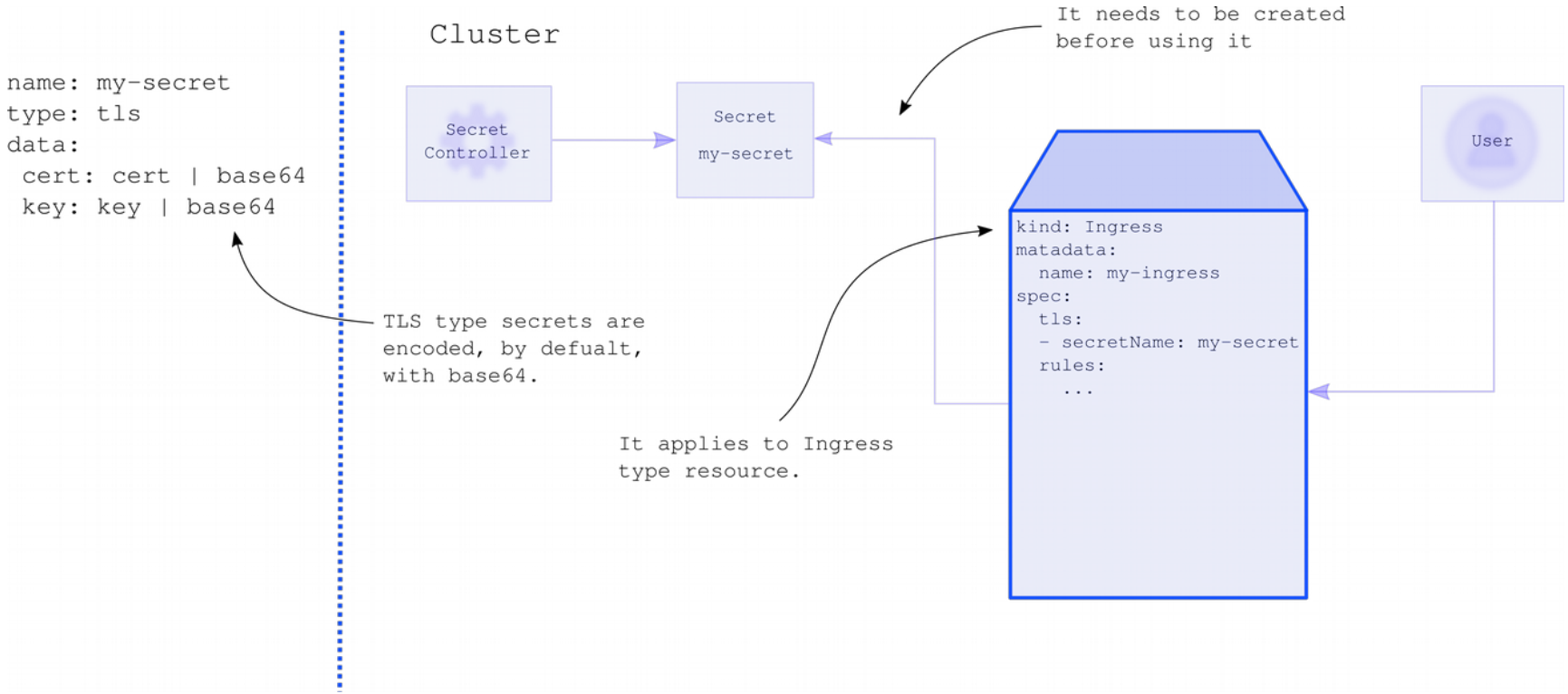


Lab4

1.- Create a Deployment with deploy.yaml file.
    $ kubectl create -f deploy.yaml
2.- Create a LoadBalancer type service with svc-lb.yaml.
    $ kubectl create -f svc-lb.yaml
    Note: Observe the particularity of the service.
3.- Access each pod, go to /usr/share/nginx/html, and change the content of indeh.html to "pod-1" and "pod-2".
    $ kubectl exec -it POD bash
    $ echo "pod-X" > /usr/share/nginx/html/index.html
4.- SSH into the VM called lab4
    $ gcloud compute ssh lab4 --zone ZONE
5.- Run a for loop with 20 requests to the IP address of the load balancer.
    $ for i in {1..20}; \
      do curl IP_ADDR; \
      done;
6.- Delete all the pods
    $ kubectl delete pod --all
    Note: Observe K8s reaction
7.- To clean up, delete the deployment
    $ kubectl delete deploy nginx

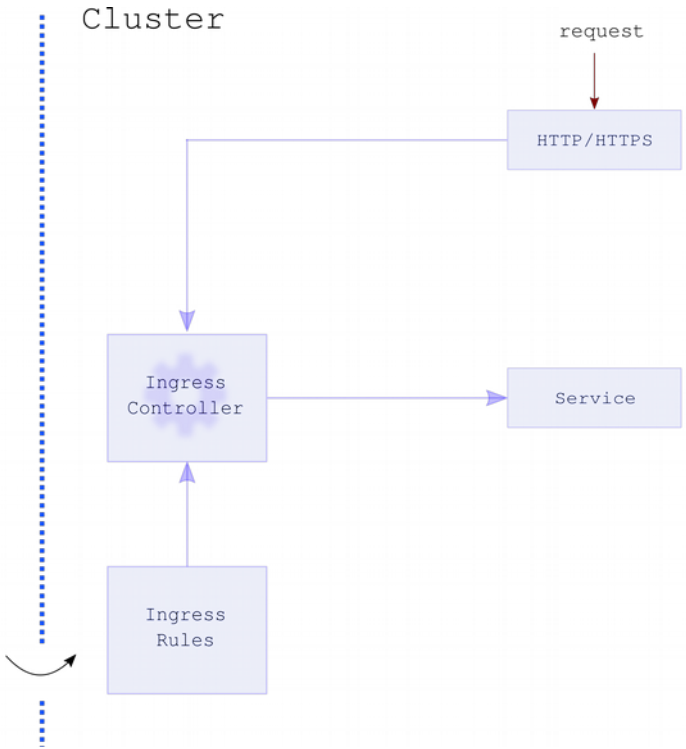
8.- Create a Job, with job.yaml
    $ kubectl create -f job.yaml
9.- When the Job has completed, check the logs
    $ kubectl logs POD
    Note: What do you think it is?

Secret - TLS


Cluster
It needs to be created
before using it
name: my-secret
type: tls
data:
cert: cert | base64
key: key | base64
Secret
Controller
Secret
my-secret
User
kind: Ingress
matadata:
name: my-ingress
spec:
tls:
- secretName: my-secret
rules:
...
TLS type secrets are
encoded, by defualt,
with base64.
It applies to Ingress
type resource.

Ingress

```
name: my-ingress
spec:
  tls:
  - secretName: tls
  rules:
  - host: site1.domain.com
    http:
      paths:
      - path: /path1/
        backend:
          serviceName: service1
          servicePort: 80
      - path: /path2/
        backend:
          serviceName: service2
          servicePort: 8080
      - path: /
        backend:
          serviceName: service3
          servicePort: 80
  - host: site2.domain.com
    http:
      paths:
      - backend:
          serviceName: service1
          servicePort: 80
```



Ingress object manages external access to the services in a cluster.

Traffic routing is controlled by rules defined on Ingress resource.

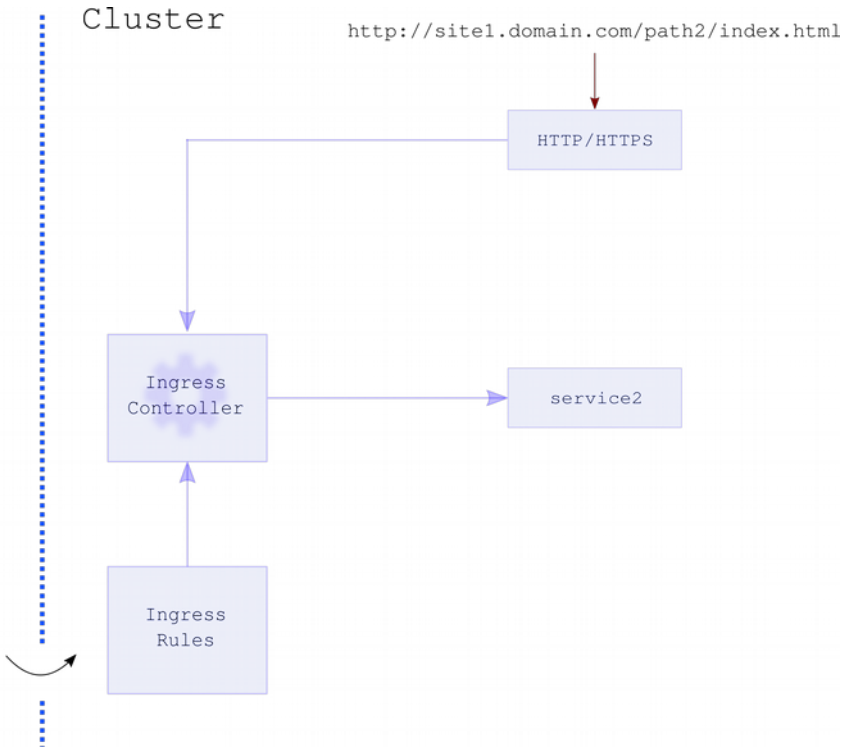
Ingress resource has no knowledge about the backends. It forwards a packet that matched an Ingress rule, to the right service.

Ingress can provide load balancing, SSL termination and name-based virtual hosting.

Ingress Controller, unlike other controllers, is NOT part of the kube-controller-manager binary.

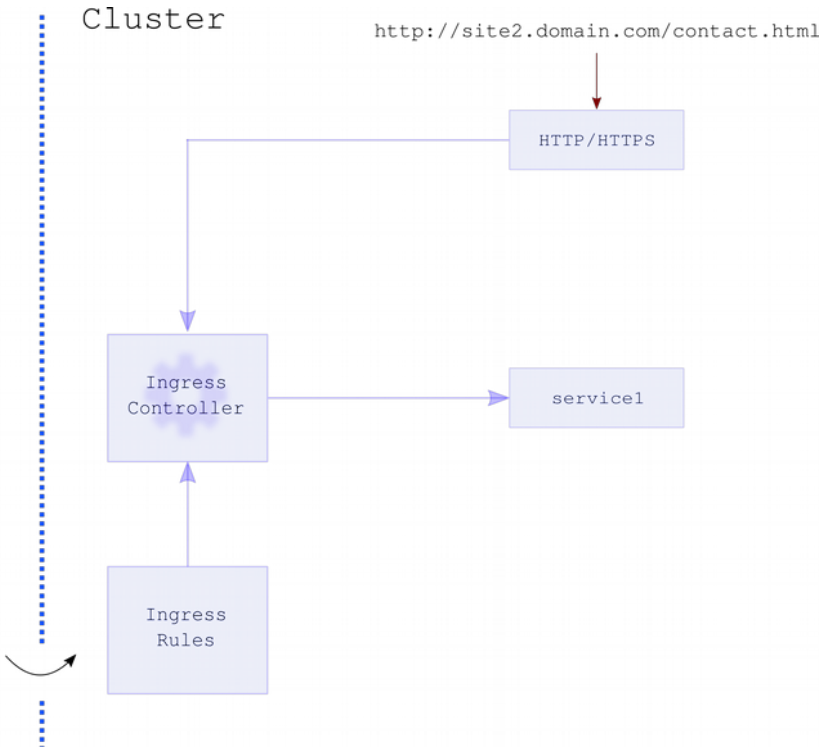
Ingress

```
name: my-ingress
spec:
  tls:
  - secretName: tls
  rules:
  - host: site1.domain.com
    http:
      paths:
      - path: /path1/
        backend:
          serviceName: service1
          servicePort: 80
      - path: /path2/
        backend:
          serviceName: service2
          servicePort: 8080
      - path: /
        backend:
          serviceName: service3
          servicePort: 80
  - host: site2.domain.com
    http:
      paths:
      - backend:
          serviceName: service1
          servicePort: 80
```



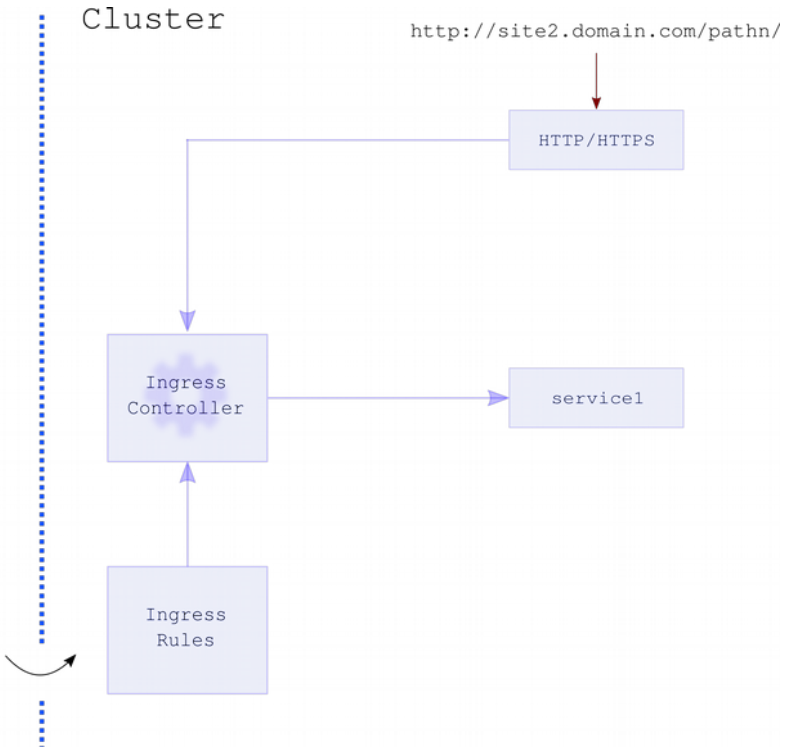
Ingress

```
name: my-ingress
spec:
  tls:
  - secretName: tls
  rules:
  - host: site1.domain.com
    http:
      paths:
      - path: /path1/
        backend:
          serviceName: service1
          servicePort: 80
      - path: /path2/
        backend:
          serviceName: service2
          servicePort: 8080
      - path: /
        backend:
          serviceName: service3
          servicePort: 80
  - host: site2.domain.com
    http:
      paths:
      - backend:
          serviceName: service1
          servicePort: 80
```



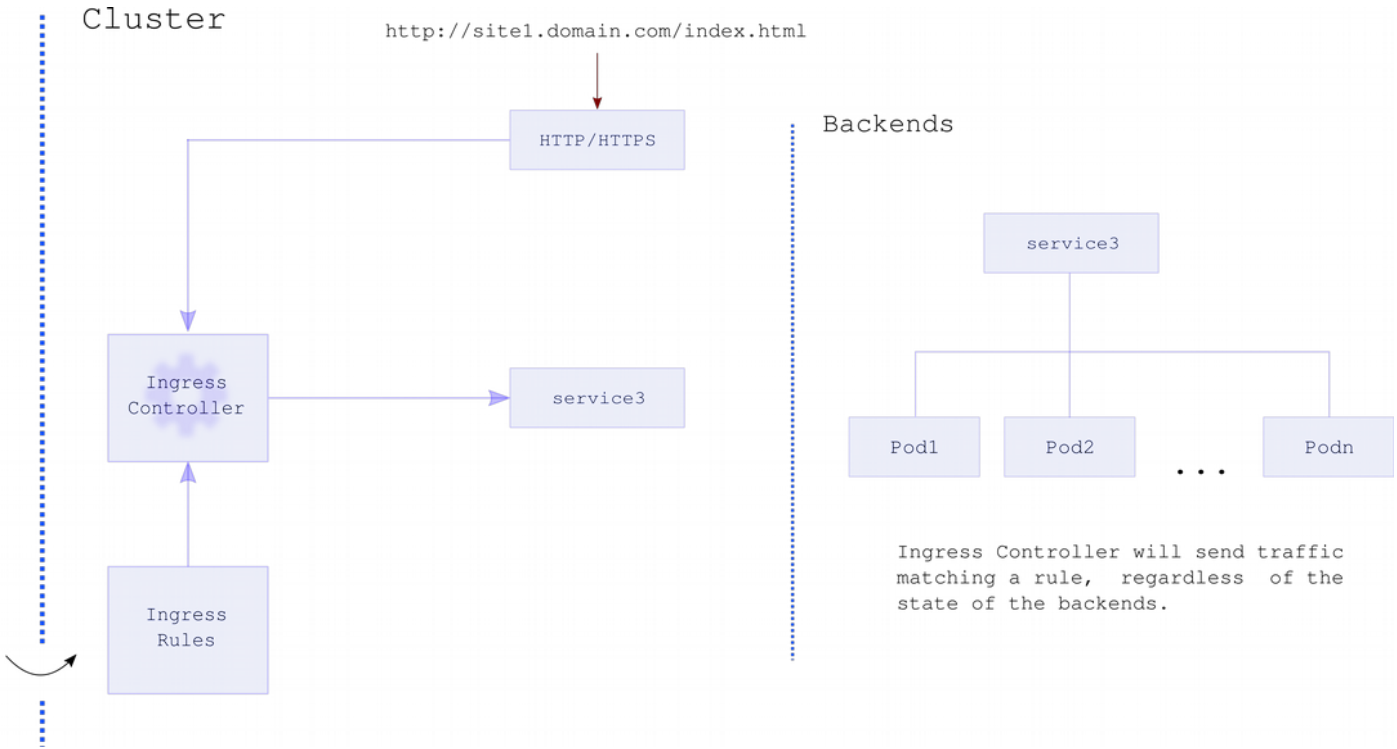
Ingress

```
name: my-ingress
spec:
  tls:
  - secretName: tls
  rules:
  - host: site1.domain.com
    http:
      paths:
      - path: /path1/
        backend:
          serviceName: service1
          servicePort: 80
      - path: /path2/
        backend:
          serviceName: service2
          servicePort: 8080
      - path: /
        backend:
          serviceName: service3
          servicePort: 80
  - host: site2.domain.com
    http:
      paths:
      - backend:
          serviceName: service1
          servicePort: 80
```



Ingress

```
name: my-ingress
spec:
  tls:
  - secretName: tls
  rules:
  - host: site1.domain.com
    http:
      paths:
      - path: /path1/
        backend:
          serviceName: service1
          servicePort: 80
      - path: /path2/
        backend:
          serviceName: service2
          servicePort: 8080
      - path: /
        backend:
          serviceName: service3
          servicePort: 80
  - host: site2.domain.com
    http:
      paths:
      - backend:
          serviceName: service1
          servicePort: 80
```



Lab 5

Configure Redis login with a ConfigMap, and pass the password in a secret.

1.- Create the ConfigMap with cm.yaml file
    $ kubectl create -f cm.yaml
2.- Create the secret with secret.yaml file
    $ kubectl create -f secret.yaml
3.- Create the Redis pod, with pod.yaml file
    $ kubectl create -f pod.yaml
4.- Execute "ping" command on redis container.
    $ kubectl exec redis -- sh -c 'redis-cli ping'
    Note: The command should fail with "NOAUTH Authentication required."
5.- Execute the same command, but now with the password; passed as an environment variable from the secret.
    $ kubectl exec redis -- sh -c 2>/dev/null 'redis-cli -a $REDIS_PASSWORD ping'

Create a deployment with certain pods targeted by an HTTPS Load Balancer, secured with TLS type secret.

1.- Create the landing page and nginx configuration ConfigMaps with cm-lp.yaml and cm-nginx.yaml
2.- Create 4 Nginx pods, and a NodePort type service targeting them.
3.- Create the certificates to be applied to the Load Balancer
    $ openssl req -newkey rsa:2048 -nodes -keyout tls.key -out tls.csr
    $ openssl x509 -in tls.csr -out tls.pem -req -signkey tls.key
4.- Create the secret from the certificates
    $ kubectl create secret tls certs --cert tls.pem --key tls.key
5.- Create the Ingress object, with ing.yaml file.
    $ kubectl create -f ing.yaml
6.- Once the Load balancer is created, and with healthy backends, curl it.
    $ curl https://LB_IP_ADDR/
7.- Modify the Ingress rules (from ing.yaml file), to accept requests for "training.com" host.

end