Máster
Architecture & Engineering

Cristabel Talavera Arriola

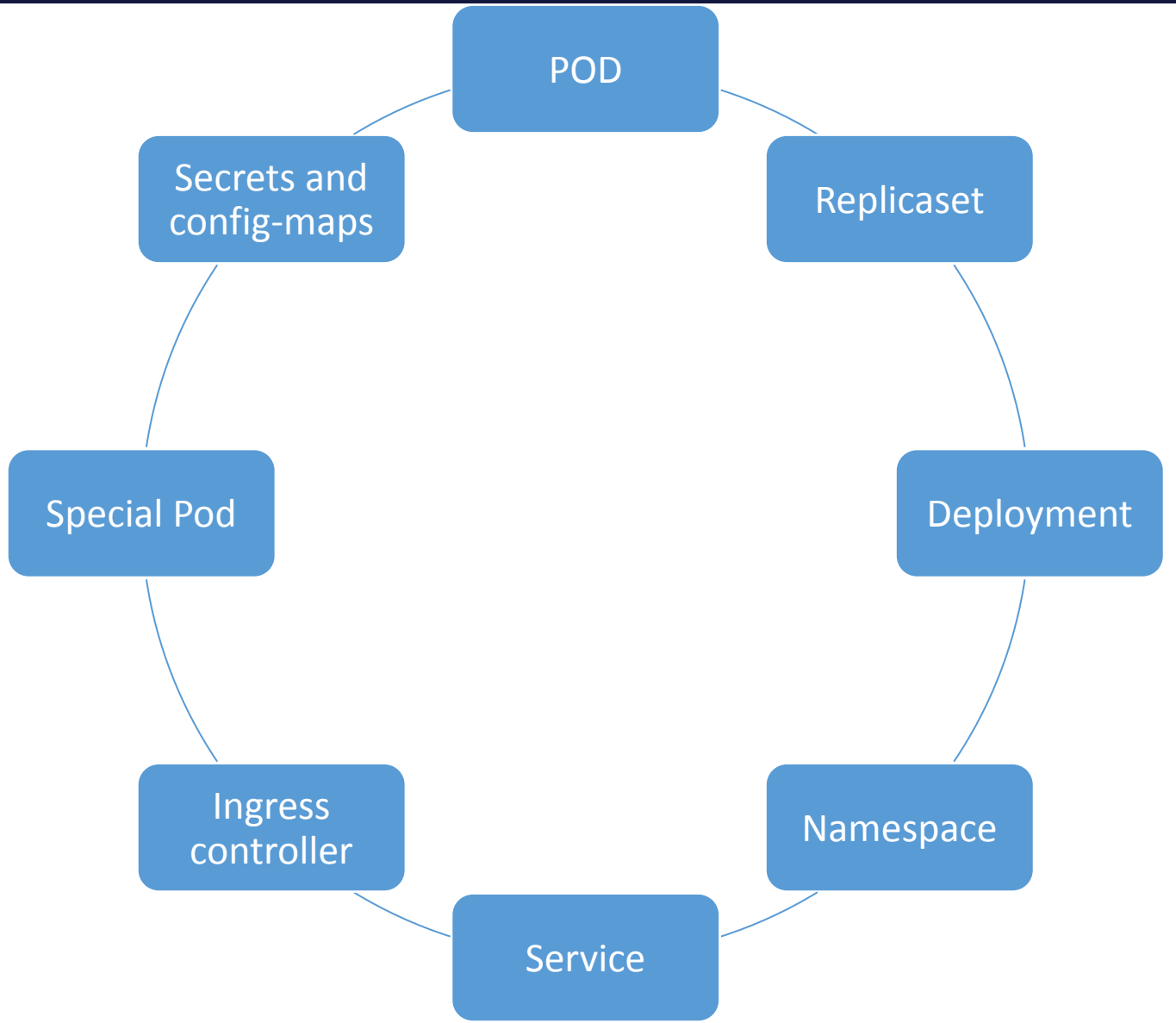
Kubernetes

Indice

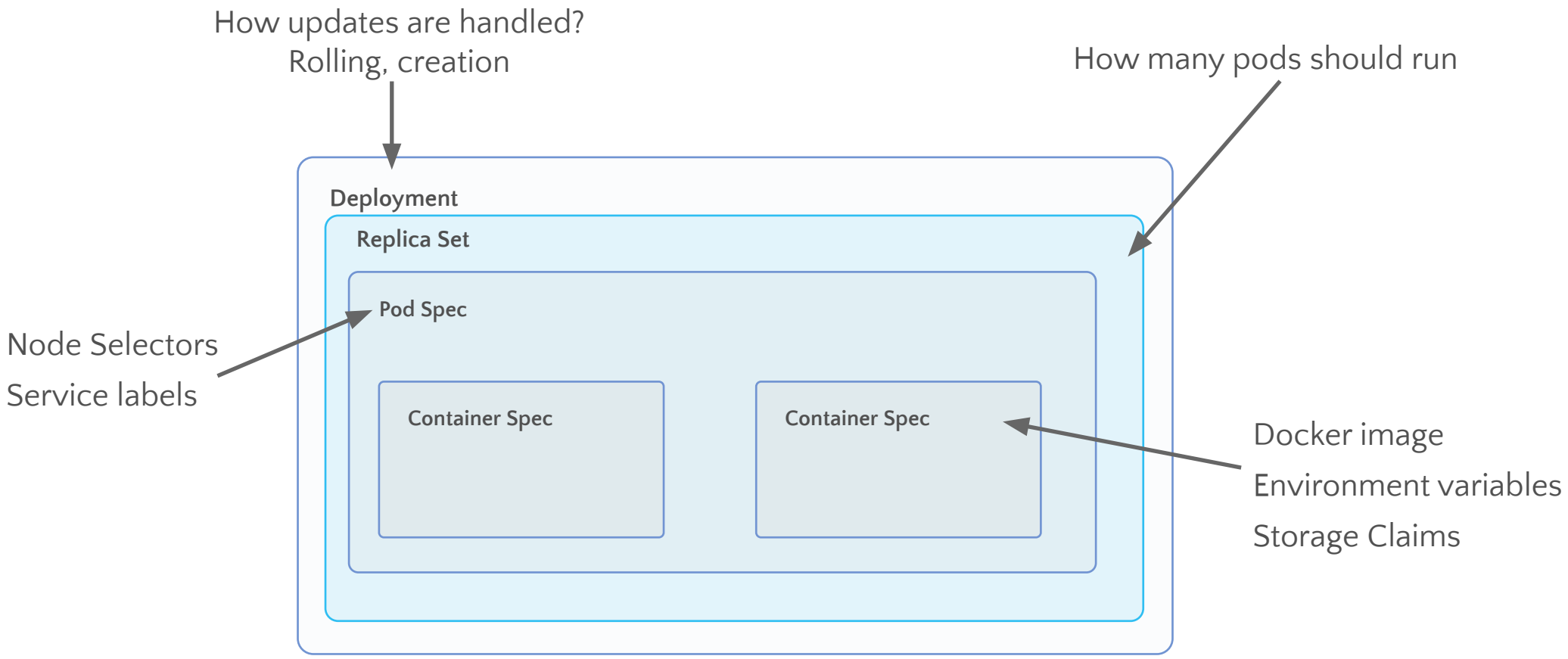
- Componentes básicos de K8S
  - PODs
  - Replica Set
  - Namespace
  - Secrets & Config Maps
- Labels

Componentes básicos de Kubernetes

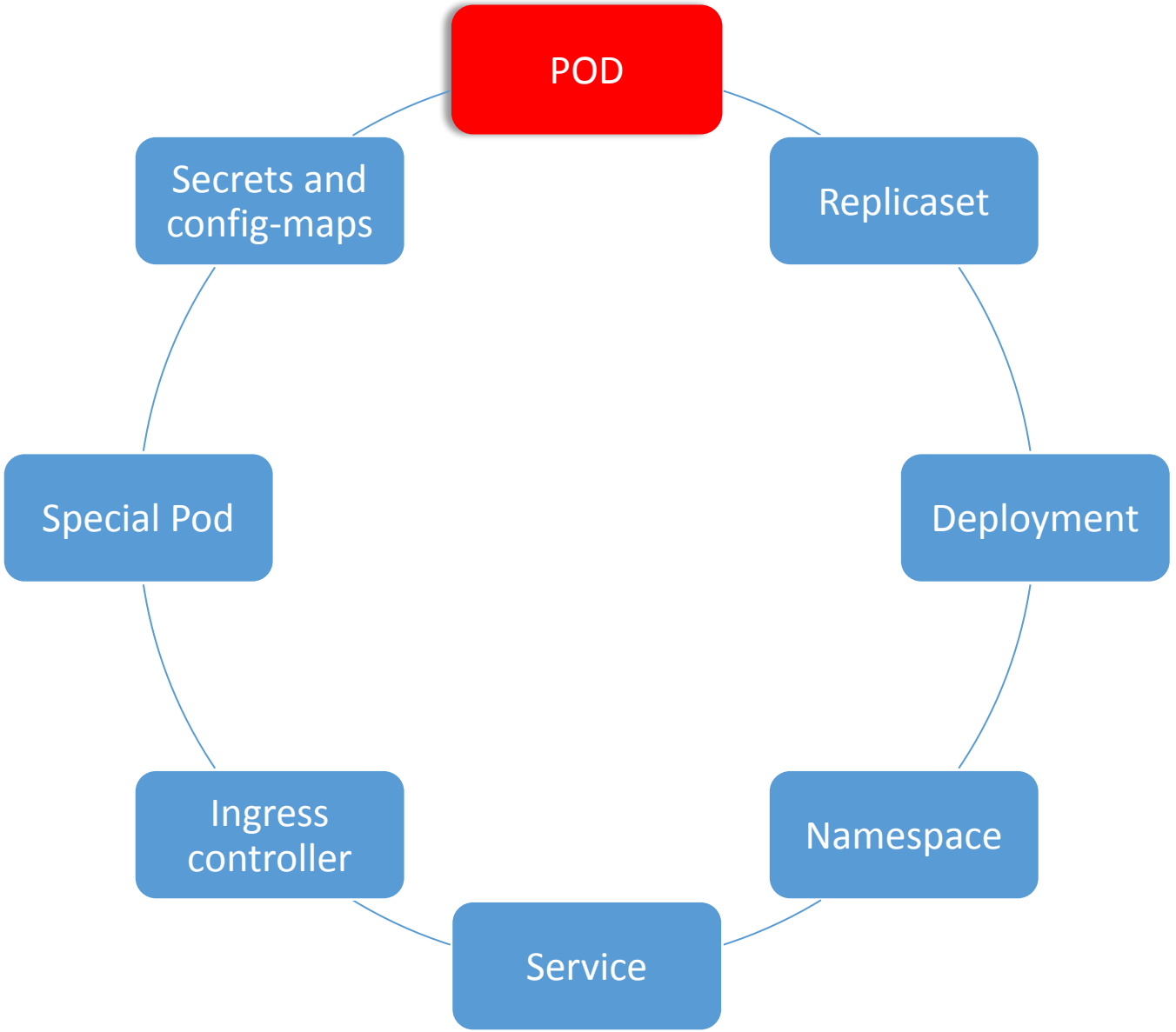
Componentes en K8S



Componentes Básicos



Componentes en K8S



¿Qué es un POD?

- Un Pod es el componente básico en K8S es el “building block”
- Se usa para desplegar los contenedores
- Puede contener múltiples contenedore s(eg - side car)
- Encapsula container(s), storage, network IP, y options de ejecución
- Usa Deployment para indicar los tipos de recursos usados como POD:
  - ReplicaSet
  - StatefulSet
  - DaemonSet
  - Job
  - InitContainer

Kubernetes manifest: Pod

pod-definition.yml

```
apiVersion:
kind:
metadata:



spec:
```

pod-definition.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
      app: myapp

spec:
```

String

String

Dictionary

pod-definition.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
      app: myapp
      type: front-end
spec:
  containers:
    - name: nginx-container
      image: nginx
```

List/Array

Kubernetes manifest: Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-demo
  namespace: mem-example
spec:
  containers:
  - name: memory-demo-ctr
    image: polinux/stress
    resources:
      limits:
        memory: "200Mi"
      requests:
        memory: "100Mi"
    command: ["stress"]
    args: ["--vm", "1", "--vm-bytes", "150M", "--vm-hang", "1"]
```

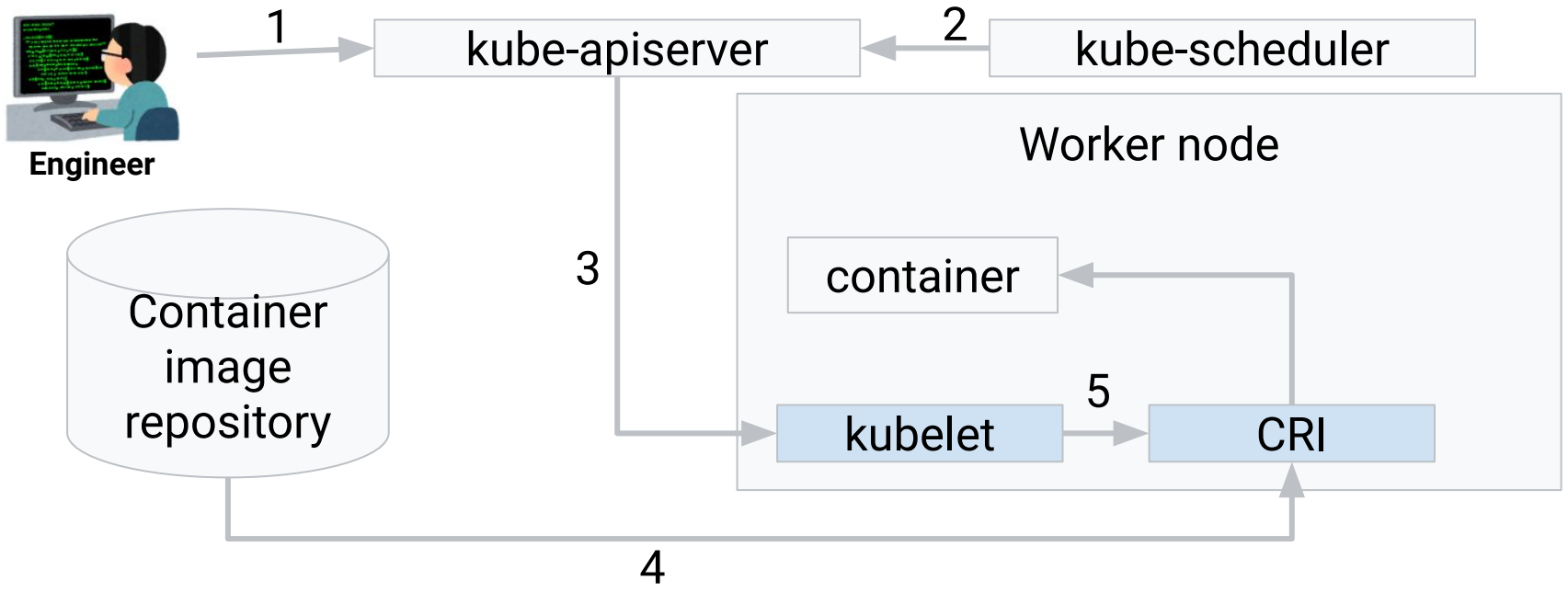
Práctica 2

- Generar un POD Manifest YAML (-o yaml). No se crea(--dry-run) para la imagen de dockerhub:

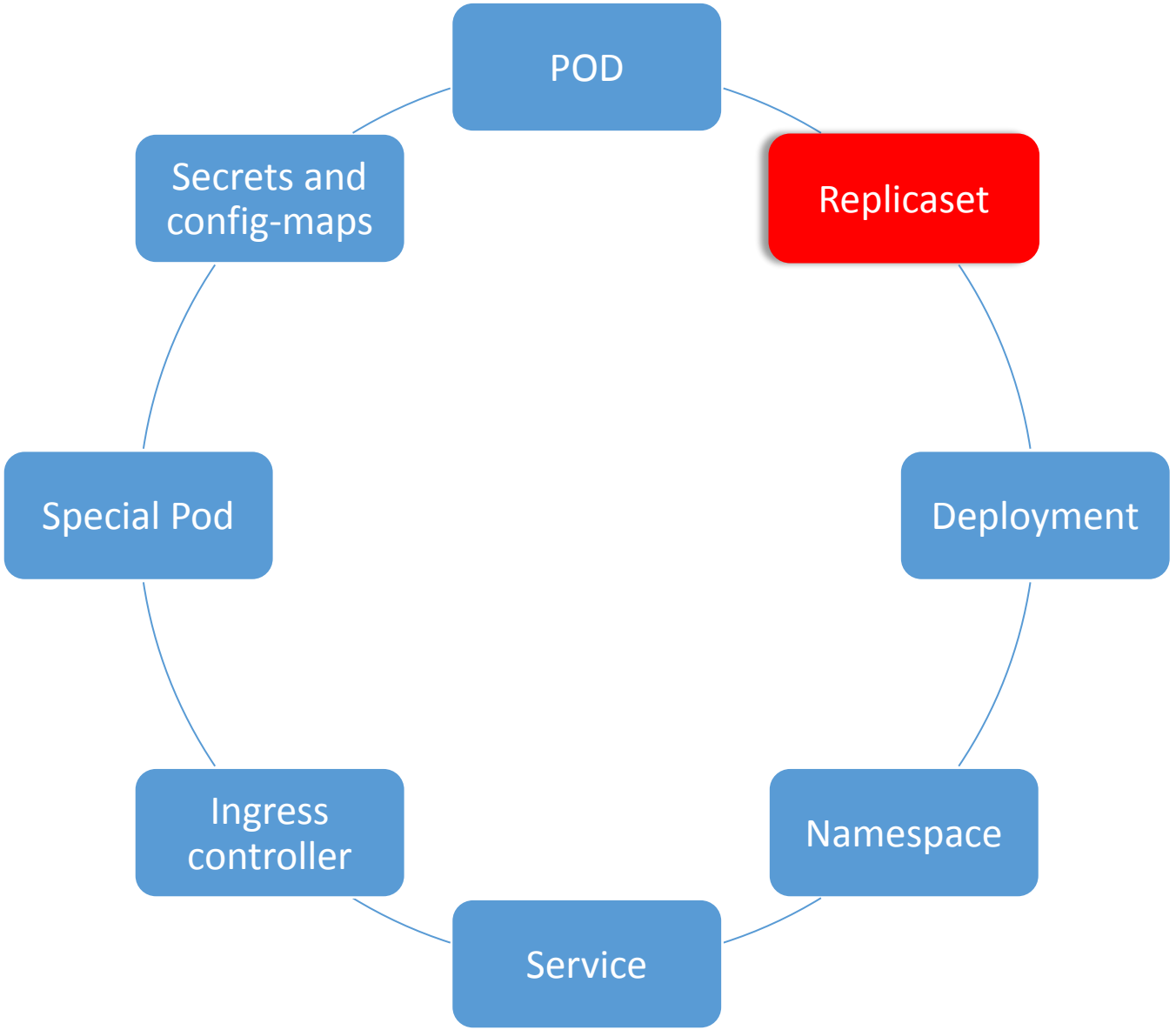
  xstabel/dotnet-core-publish
- Usar el yaml creado para desplegar la aplicación
- Listar los pods de todos los namespaces
- Listar sólo los pods de default
- Inspeccionar y describir nuestro POD
- Ejecutar la consola en un pod y ejecutar comandos bash, por ejemplo que nos sale al hacer un curl www.google.com
- Borrar todo lo creado

Lifecycle of Pod creation

1. `kubectl apply -f pod.yaml` to ask `kube-apiserver` to create a Pod resource.
2. `kube-scheduler` is monitoring Pods not assigned to any nodes.
   When `kube-scheduler` found it, it will decide which node run the containers in the Pod.
   Ask `kube-apiserver` to annotate the Pod with the node name.
3. `kubelet` is monitoring Pods assigned to the node where `kubelet` is running. When `kubelet` found it, `kubelet` will ask `CRI(containerd in GKE)` to create containers.
4. `CRI` will pull the image from specified repository.
5. `CRI` create the container in the node.

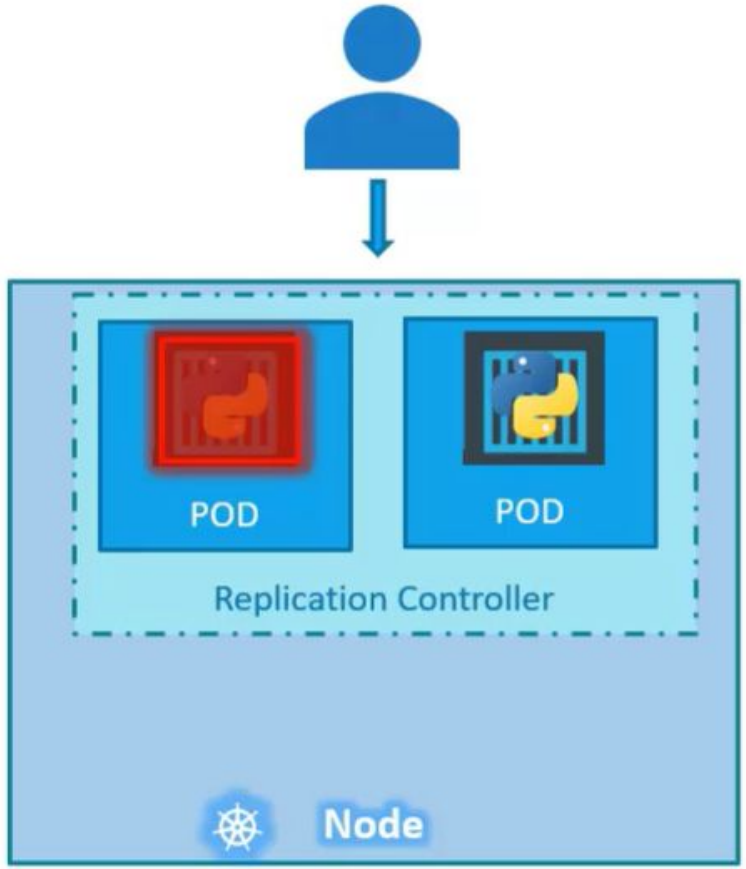


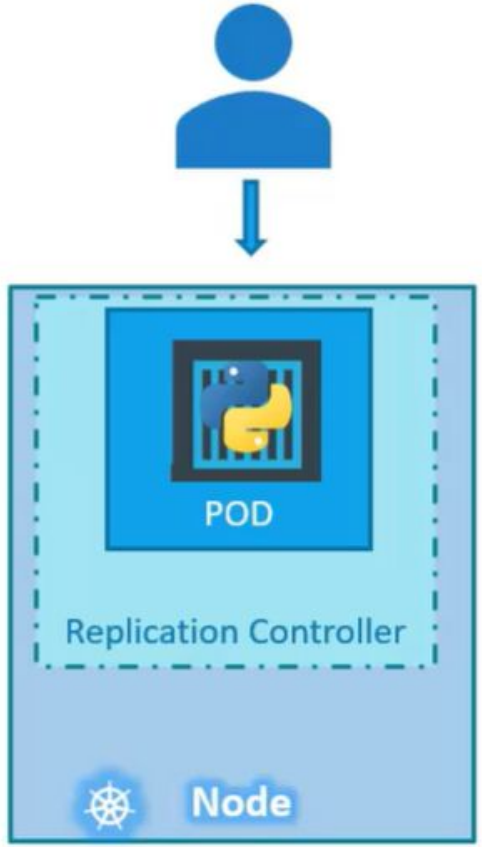
Componentes en K8S



Replica Set

- Un ReplicaSet se encarga de asegurar de que el número de PODs en ejecución sea el que se ha especificado.





Kubernetes manifest: Replica Set

replicaset-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
      app: myapp
      type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
```

POD

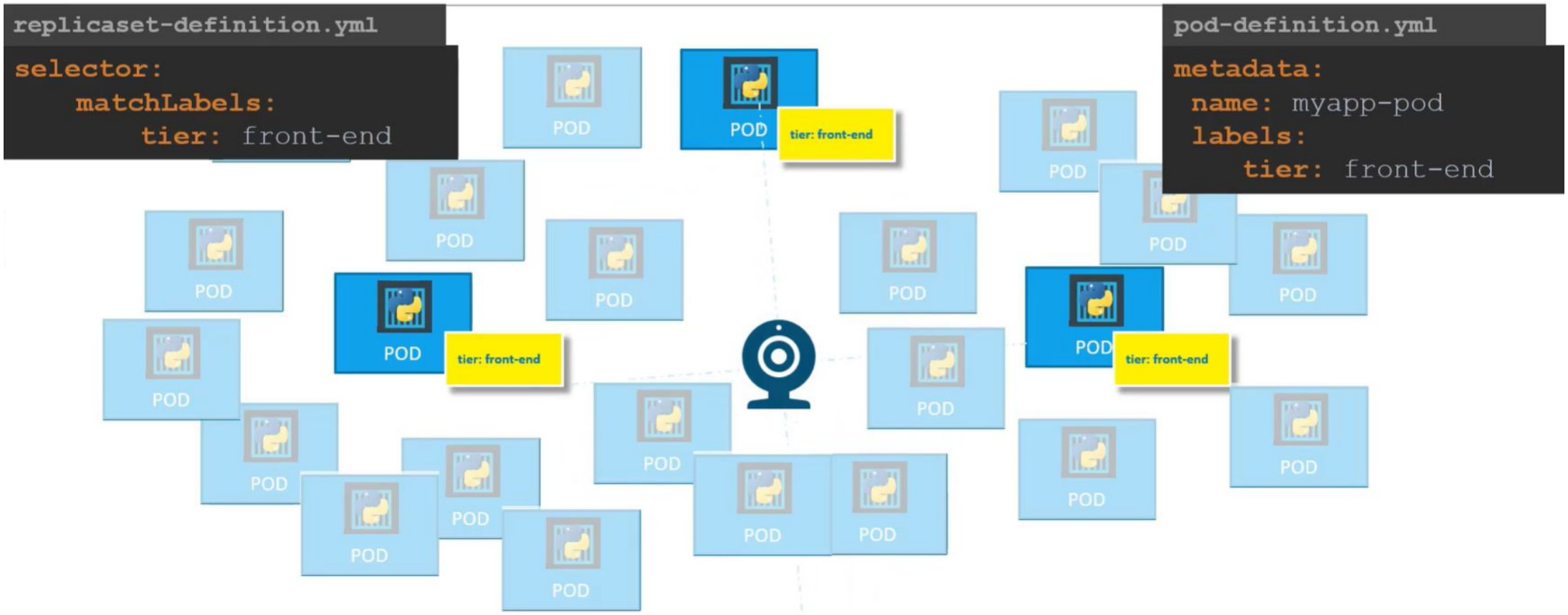
replicaset-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
      app: myapp
      type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx

  replicas: 3
```

replicaset-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
      app: myapp
      type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx

  replicas: 3
  selector:
    matchLabels:
      type: front-end
```

Replica Set


replicaset-definition.yml
selector:
matchLabels:
tier: front-end
pod-definition.yml
metadata:
name: myapp-pod
labels:
tier: front-end
POD
tier: front-end

Replica Set nos sirve para escalar instancias

```
kubectl scale --replicas=6 replicaset myapp-replicaset
```


```
kubectl create -f replicaset-definition.yml
```

```
kubectl get replicaset
```

```
kubectl delete replicaset myapp-replicaset
```

*Also deletes all underlying PODs

```
kubectl replace -f replicaset-definition.yml
```

```
kubectl scale -replicas=6 -f replicaset-defin
```

Práctica 3

- Crear Replicaset YAML de la aplicación de la práctica 2 con 3 réplicas y aplicar el yaml. Llamamos a nuestra rs **front (**image xstabel/dotnet-core-publish**)**
- Eliminamos los pods, ¿qué ocurre?
- Escalar hacia abajo la misma aplicación, ahora reducir a 1 réplica con línea de comandos o yaml
- Ver todos los objetos replica sets que existen
- Eliminar replica set ¿qué ocurre?
- *Comprobar eliminación y qué pasa con mi aplicación*

Field selectors

We use field selectors and selectors in general to let you select Kubernetes resources based on the value of one or more resource fields. So what are resource fields?

Here are some examples of field selector queries:

```
kubectl get pods --field-selector metadata.namespace!=default
kubectl get pods --field-selector metadata.name=my-service
kubectl get pods --field-selector status.phase=Running
```

Supported field selectors vary by Kubernetes resource type. All resource types support the metadata.name and metadata.namespace fields. Using unsupported field selectors produces an error.

Field selectors

Supported operators are =, ==, and != with field selectors (= and == mean the same thing). This kubectl command, for example, selects all Kubernetes Services that aren't in the default namespace:

```
kubectl get services  --all-namespaces --field-selector metadata.namespace!=default
```

Chained selectors

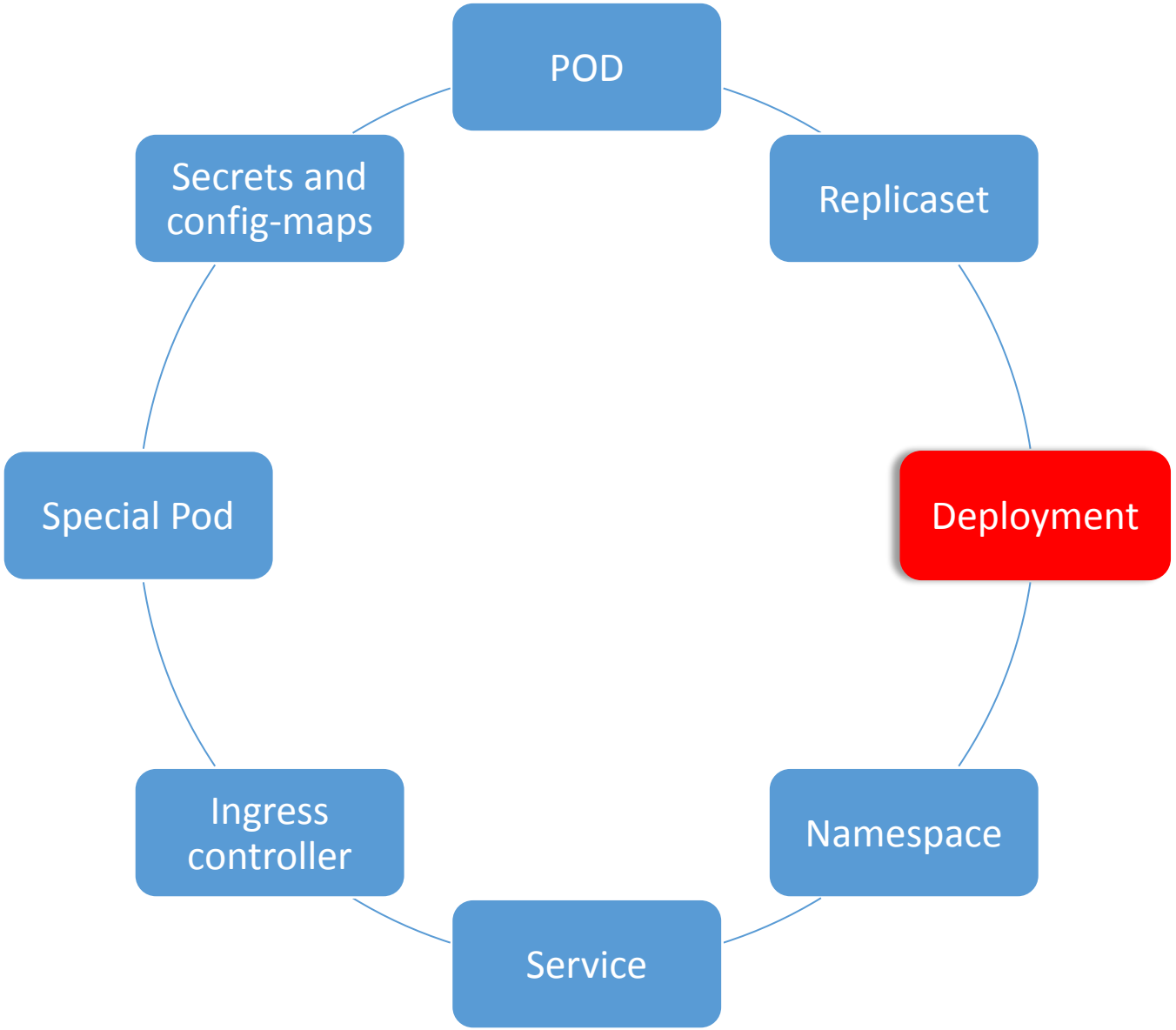
```
kubectl get pods --field-selector=status.phase!=Running,spec.restartPolicy=Always
```

Multiple resource types

```
kubectl get statefulsets,services --all-namespaces --field-selector metadata.namespace!=default
```

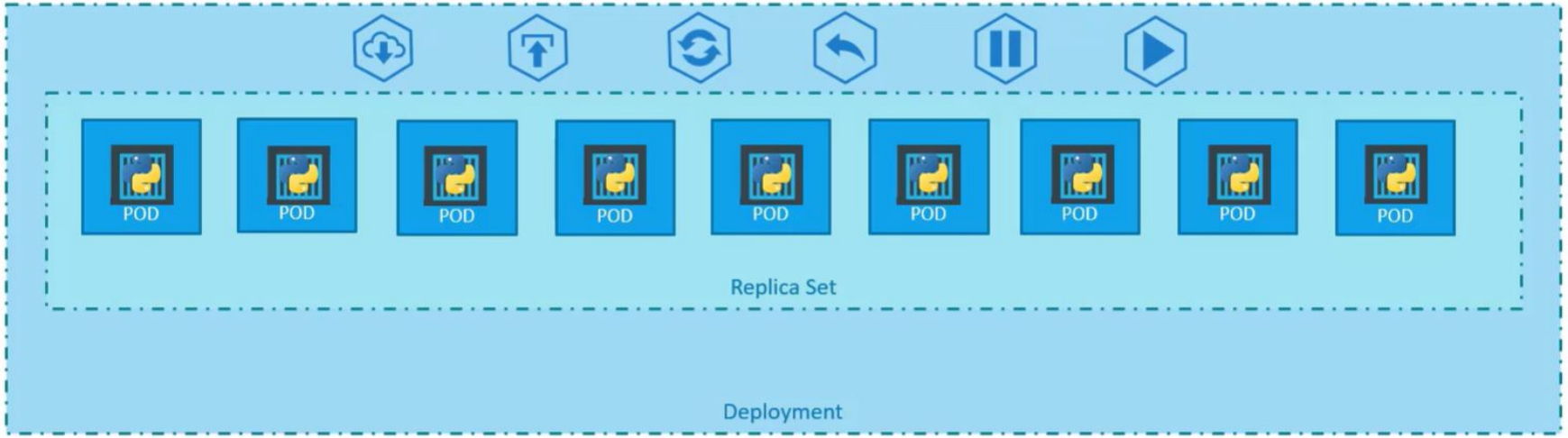
LET’S PRACTICE :-)

Componentes en K8S



Deployments

- A Deployment is a higher-level concept that manages ReplicaSets and provides declarative updates to Pods along with a lot of other useful features.
- Therefore, It’s recommended to use Deployments instead of directly using ReplicaSets, unless you require custom update orchestration or don’t require updates at all.



Kubernetes manifest: Deployment

deployment-definition.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
  labels:
      app: myapp
      type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx

  replicas: 3
  selector:
    matchLabels:
      type: front-end
```

- Create deployment

  kubectl create –f deployment-definition.yaml

- Listar despliegues:

  kubectl get deployments

- Ver Replica Set

  Kubectl get replicaset

- Ver Replica POD

  Kubectl get pods

Desplegando aplicaciones en Kubernetes

Deployment objects: crear nuevos despliegues, actualizar, aplicar rollback a versiones previas. Algunos comandos útiles para trabajar con despliegues:

- Listar despliegues:

  kubectl get deployments or kubectl get deploy

- Ver estado del roll out de despliegues

  Kubectl rollout status <deployment>

- Configurar la imagen de un despliegue

  Kubectl set image <deployment> <image>

- Ver la historia de un rollout, incluyendo revisiones previas

  Kubectl rollout history <deployment>

Desplegando aplicaciones en Kubernetes

- Pausar un despliegue

  Kubectl rollout pause <deployment>

- Reanudar un despliegue

  Kubectl rollout resume <deployment>

- Rolling Back

  kubectl rollout undo <deployment> --to-revision=<Num-Revision>

Deployment Strategy

Práctica 4 – Despliegues y PODs

- Crear un despliegue en K8S de una aplicación stateless a través de un fichero YAML. Usar un Nginx
  - En el despliegue se debe indicar 2 réplicas y dar el nombre “único” a nuestro container de la siguiente manera: NGINX-PRIMER APELLIDO
- Listar los despliegues
- Inspeccionar el despliegue, ver los eventos
- Se puede buscar nuestra aplicación a través de su nombre? … Cómo y por qué?

Práctica 4 – Despliegues y PODs

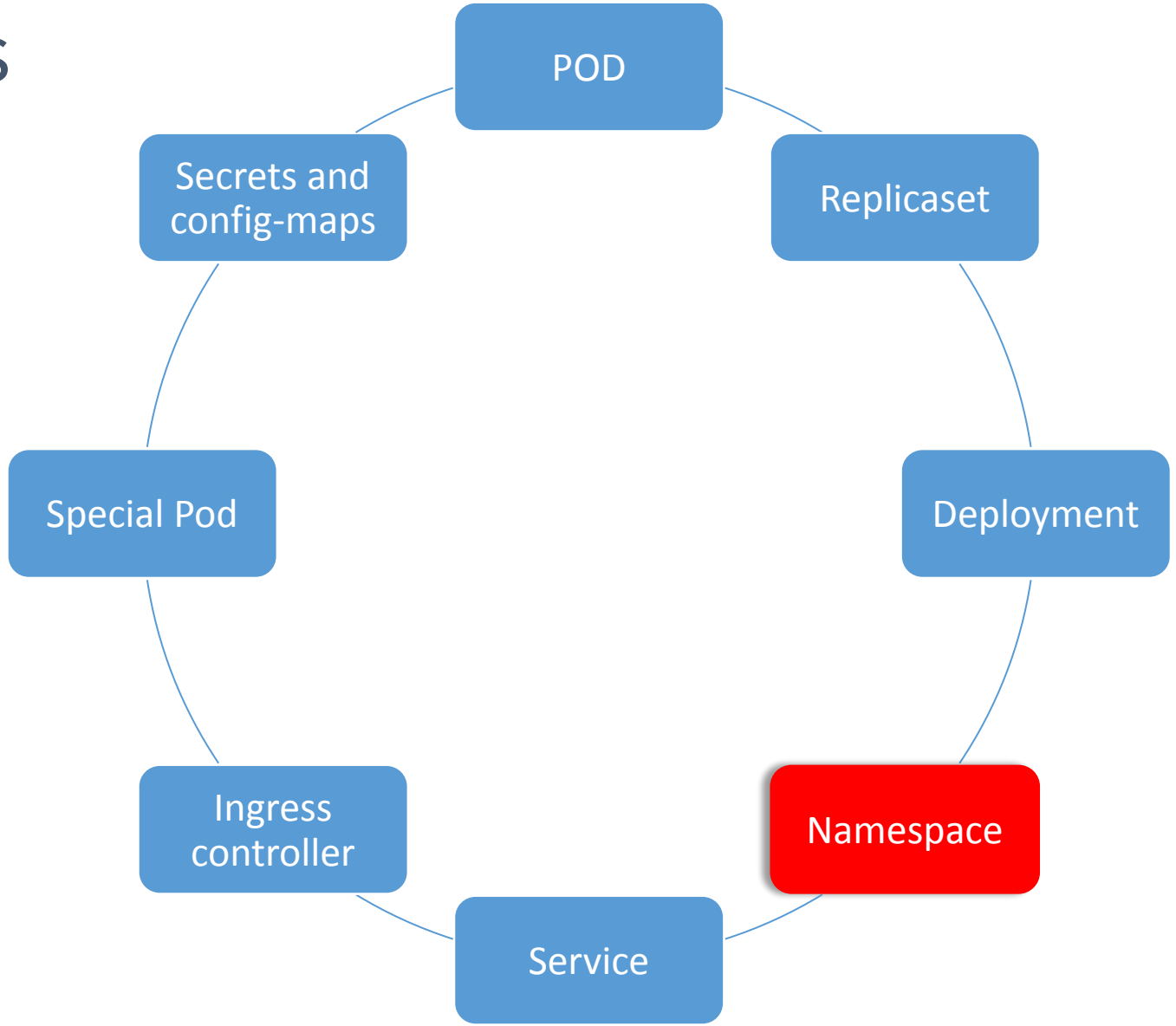
Y si creamos nuestra app?

```
FROM node:10
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 8080
CMD ["node", "app.js"]
```

```
docker build …

docker push …

Kubectl apply -f node-myapp-deployment.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: node-deployment
spec:
  selector:
    matchLabels:
      app: node
  replicas: 2
  template:
    metadata:
      labels:
        app: node
    spec:
      containers:
      - name: node
        image: node:10
        ports:
        - containerPort: 80
```

Componentes en K8S



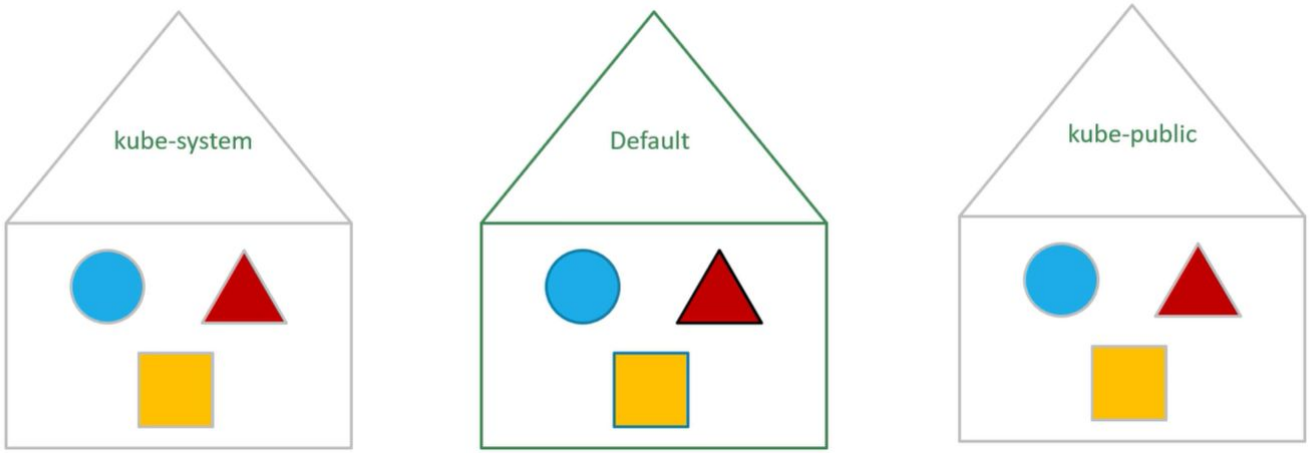
Kubernetes Namespaces

- Namespaces Object is the logical Isolation boundary
- Kubernetes has features to help us safely isolate tenants
  - Scheduling: Resource Quota
  - Network Isolation using Network Policies
  - Authentication and Authorization: Roles and Pod Security Policy

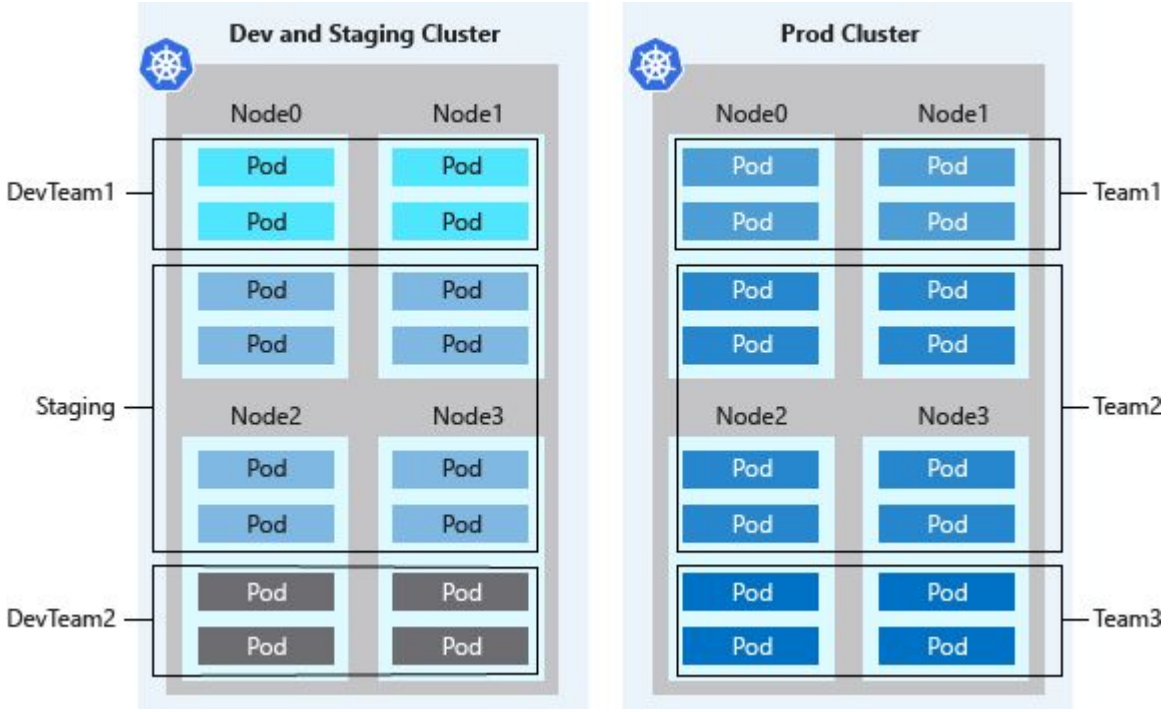
- Note: Container Level isolation still need to be done to achieve hard Isolation

Namespaces

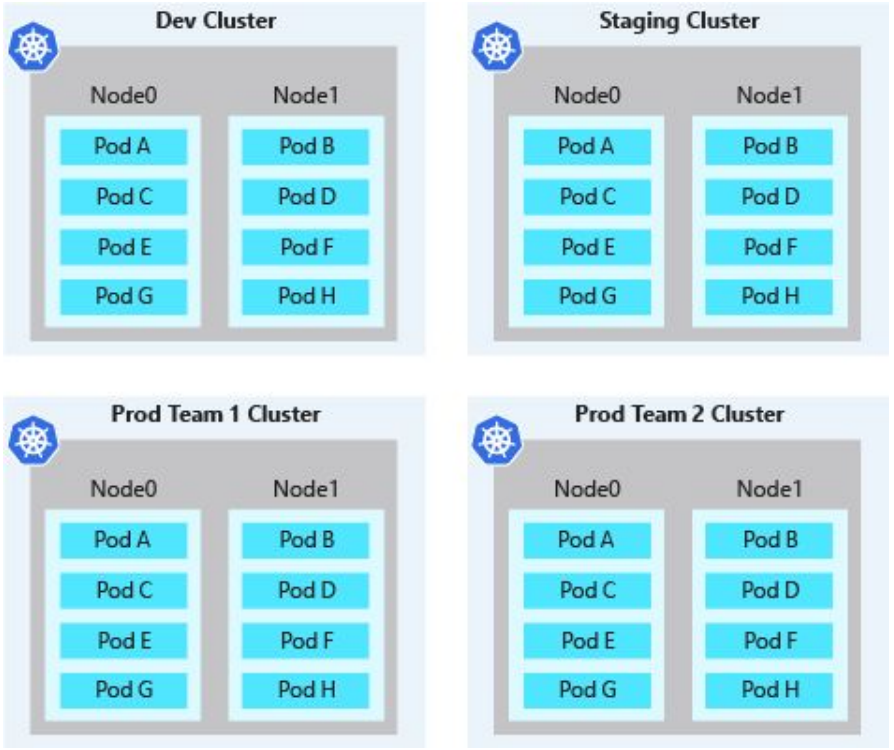
- A single cluster should be able to satisfy the needs of multiple users or groups of users.
- Kubernetes *namespaces* help different projects, teams, or customers to share a Kubernetes cluster.
- The Namespace provides a unique scope for:
  1. named resources (to avoid basic naming collisions)
  2. delegated management authority to trusted users
  3. ability to limit community resource consumption



Namespaces - Isolation



Logically isolate clusters



Physically isolate clusters

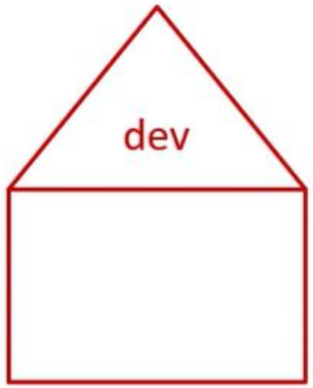
Namespace definition

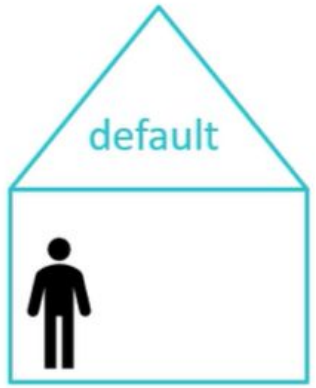
pod-definition.yml

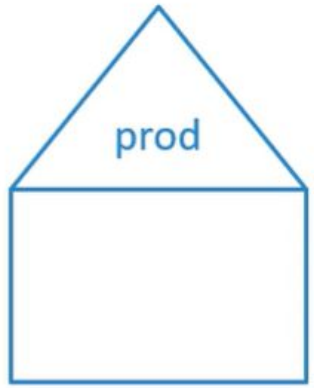
```yaml
apiVersion: v1
kind: Pod

metadata:
 name: myapp-pod
 namespace: dev

 labels:
    app: myapp
    type: front-end
spec:
  containers:
  - name: nginx-container
    image: nginx
```

```yaml
kind: Namespace
apiVersion: v1
metadata:
  name: test
  labels:
      name: Dev
```

```
kubectl apply -f pod.yaml --namespace=dev
```








```
> kubectl get pods --namespace=dev
```

```
> kubectl get pods
```

```
> kubectl get pods --namespace=prod
```

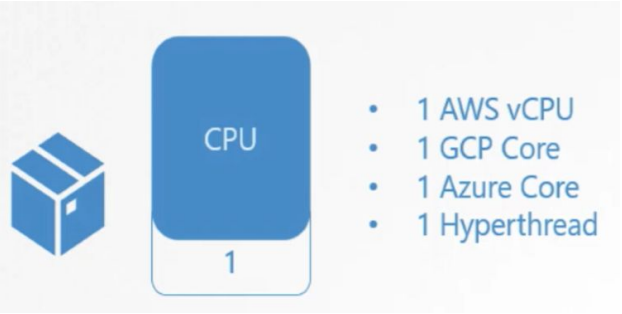
Namespace Quotas

Compute-quota.yaml

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
    name: compute-quota
    namespace: dev

spec:
  hard:
    pods: "10"
    requests.cpu: "4"
    requests.memory: 5Gi
    limits.cpu: "10"
    limits.memory: 10Gi
```

Resource Requirements

Se puede especificar la cantidad de recursos mínimos que requieren los pods y también los límites de estos.



1 G (Gigabyte) = 1,000,000,000 bytes

1 M (Megabyte) = 1,000,000 bytes

1 K (Kilobyte) = 1,000 bytes

1 Gi (Gibibyte) = 1,073,741,824 bytes

1 Mi (Mebibyte) = 1,048,576 bytes

1 Ki (Kibibyte) = 1,024 bytes

pod-definition.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: simple-webapp-color
  labels:
    name: simple-webapp-color
spec:
  containers:
  - name: simple-webapp-color
    image: simple-webapp-color
    ports:
      - containerPort: 8080
    resources:
      requests:
        memory: "1Gi"
        cpu: 1
      limits:
        memory: "2Gi"
        cpu: 2
```

Viewing resources in the Namespace

All commands are running against the currently active Namespace. To find your Pod, you need to use the “namespace” flag:

```
$ kubectl get pods --namespace=dev
```

There is a really good tool called kubens (*created by Ahmet Alp Balkan*) that makes it a breeze!

When you run the `$ kubens` command, you should see all the namespaces, with the active namespace highlighted:

Práctica 5 – Name Spaces

- **Crear el namespace test con una quota de objetos y de recursos:**
  - **Recursos:**

    requests.cpu: "1"

    requests.memory: 1Gi

    limits.cpu: "2"

    limits.memory: 2Gi

  - **Objetos:**

    persistentvolumeclaims: "2"

    replicationcontrollers: "10"

    secrets: "5"

    services: "5"

    services.loadbalancers: "1"

- **Listar los pods de namespace de sistema, listar los pods de todos los namespaces**
- **Configurar namespace por defecto: test**
- **Crear nuestro NGINX en namespace test** ⟶

Usad lo siguiente como ej:
(con 3 réplicas que pasa?)

```
image: nginx:1.14.2
resources:
  requests:
    cpu: 200m
    memory: 0.5Gi
  limits:
    cpu: 200m
    memory: 0.5Gi
```

Cross Namespace communication

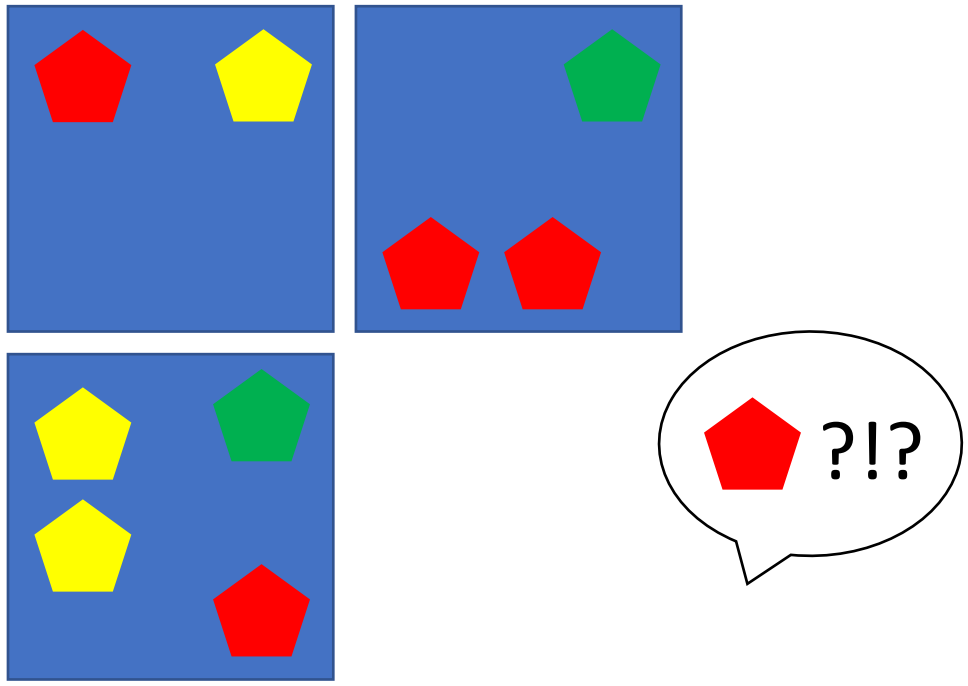
Namespaces are “hidden” from each other, but a service in one ns can talk to a service in another ns. When your app wants to access a K8S Service, you can use the built-in DNS service discovery and just point your app at the Service’s name. However, you can create a service with the same name in multiple Namespaces! It’s easy to get around this by using the expanded form of the DNS address. Services in K8S expose their endpoint using a common DNS pattern:

```
<Service Name>.<Namespace Name>.svc.cluster.local
```

What is service discovery?

Problem: How service can find and communicate with another one running into the same cluster?

- Service Registry/Naming Service
- Service Announcement
- Lookup/Discovery
- Load Balancing

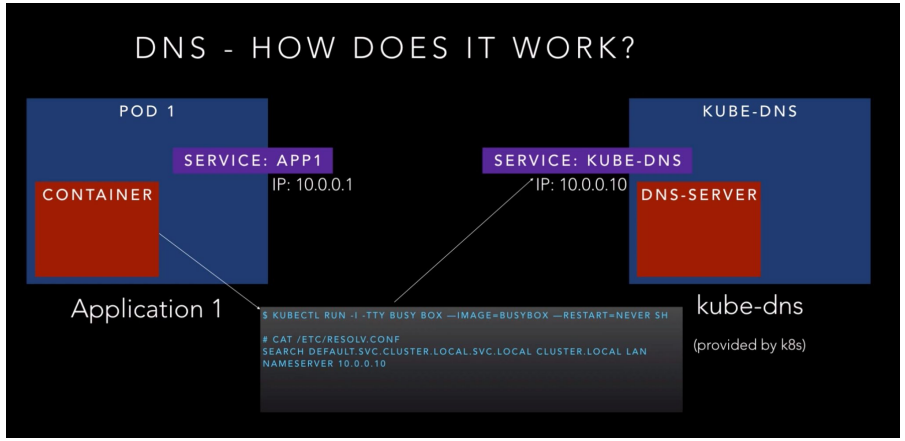


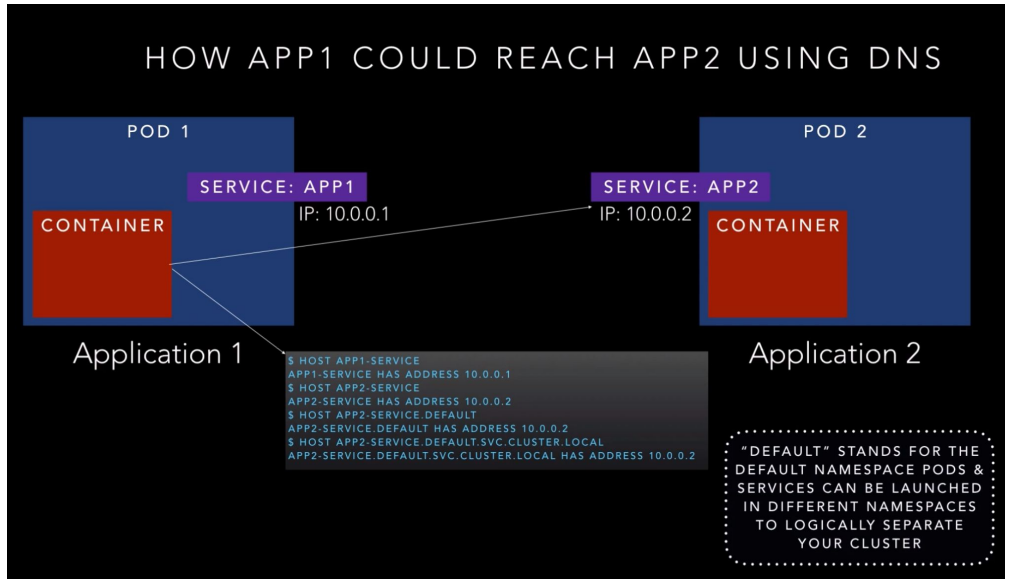
Service Discovery - DNS

Todos los servicios en Kubernetes tienen un nombre DNS. La nomenclatura utilizada para el nombre DNS:

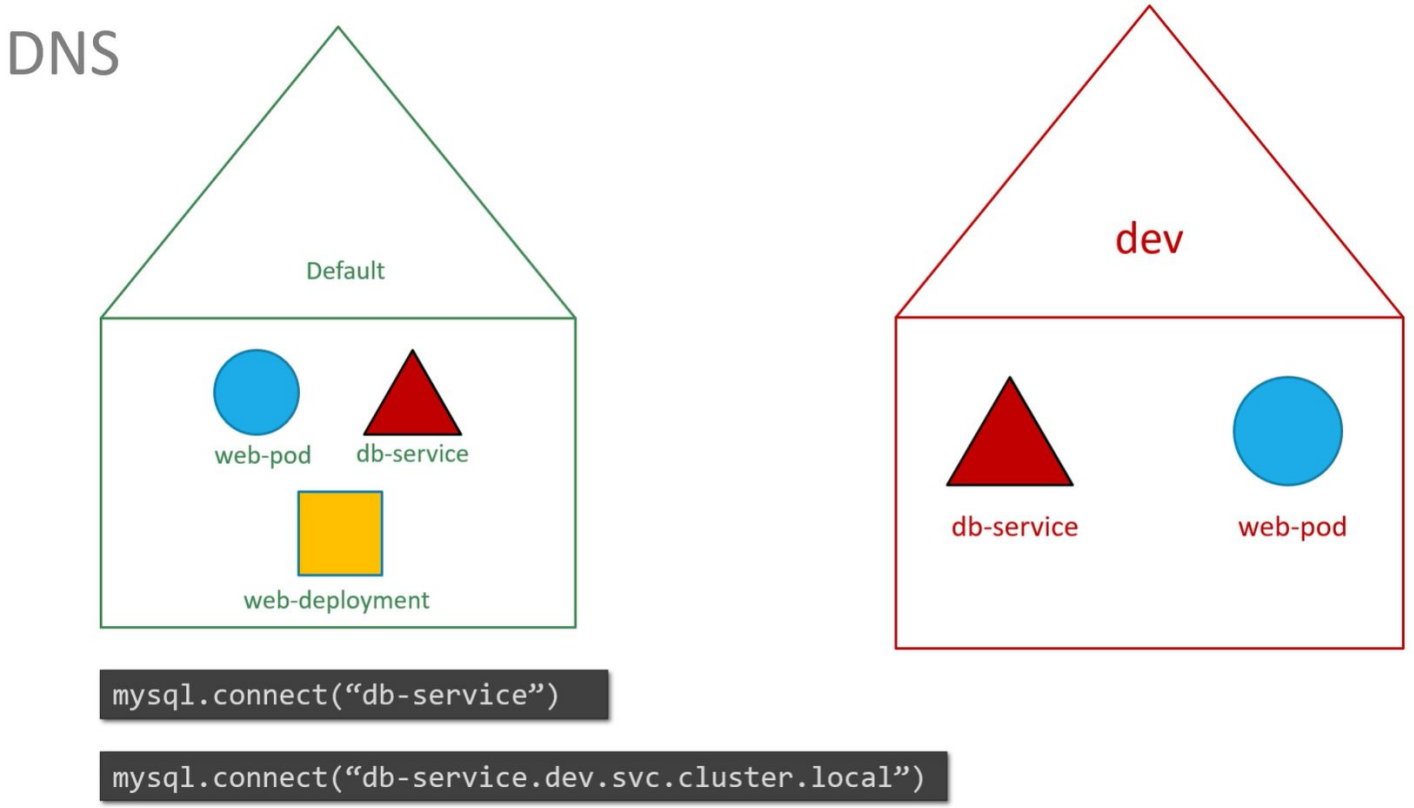
<my-Service-name>.<my-namespace>.svc.cluster.local

Esta nomenclatura facilita identificar el nombre de servicios que tu aplicación necesita.





Core DNS




```
mysql.connect("db-service")
```

```
mysql.connect("db-service.dev.svc.cluster.local")
```

```
mysql.connect("db-service.dev.svc.cluster.local")
```

Service Name | Namespace | Service | domain

Configurations in K8S

Comandos y Argumentos

```
FROM Ubuntu

ENTRYPOINT ["sleep"]

CMD ["5"]
```

pod-definition.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: ubuntu-sleeper-pod
spec:
  containers:
    - name: ubuntu-sleeper
      image: ubuntu-sleeper
      args: ["10"]
```

```
kubectl create -f pod-definition.yml
```

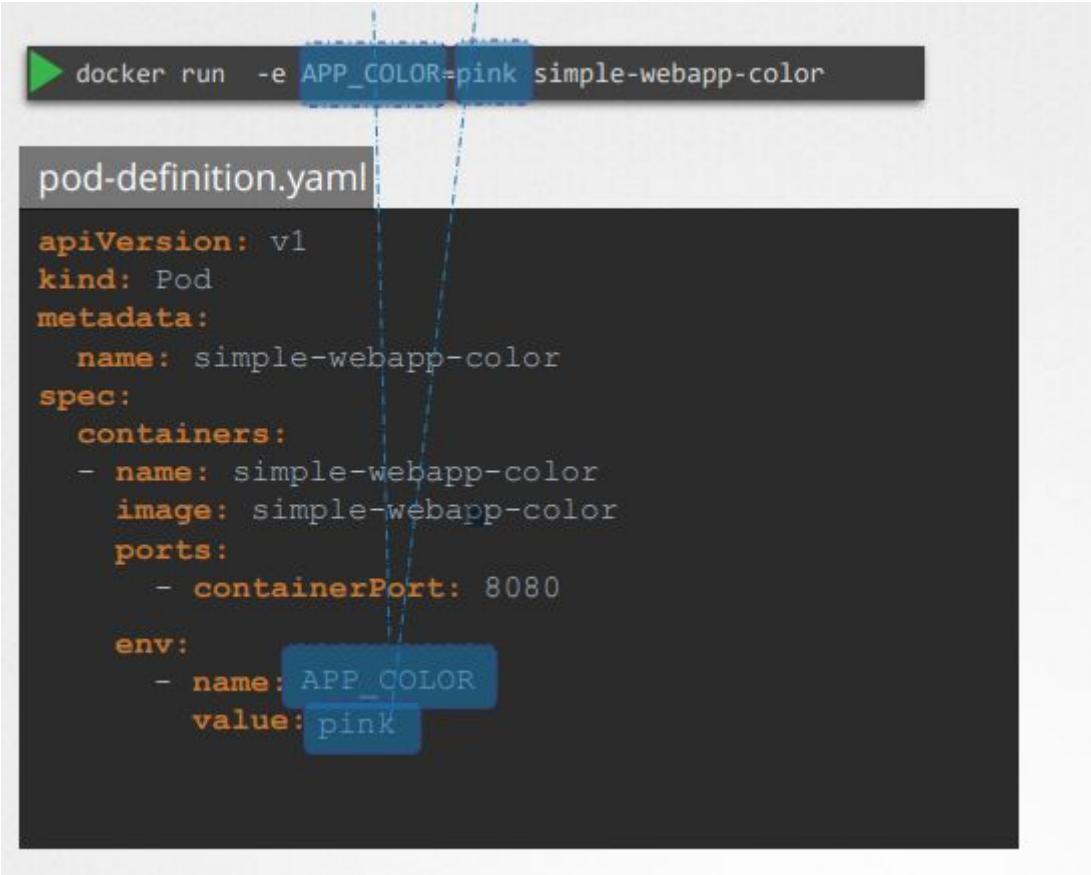
```
FROM Ubuntu

ENTRYPOINT ["sleep"]

CMD ["5"]
```

pod-definition.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: ubuntu-sleeper-pod
spec:
  containers:
    - name: ubuntu-sleeper
      image: ubuntu-sleeper
      command: ["sleep2.0"]

      args: ["10"]
```

Variables de entorno

Define the variables in pod spec using the env section:



Práctica 6 – Variables de entorno

- Crear desde fichero las variables de entorno con el nombre DEMO_GREETING y DEMO_FAREWELL con valores de prueba, para la imagen `gcr.io/google-samples/node-hello:1.0`
- veamos las variables `kubectl exec <pod> -- printenv`
- Usemos las variables de entorno con la imagen bash con el comando echo y como argumentos las variables que vayamos a crear

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envar-dhack2
spec:
  containers:
  - name: env-print-dh
    image: bash
    env:
```

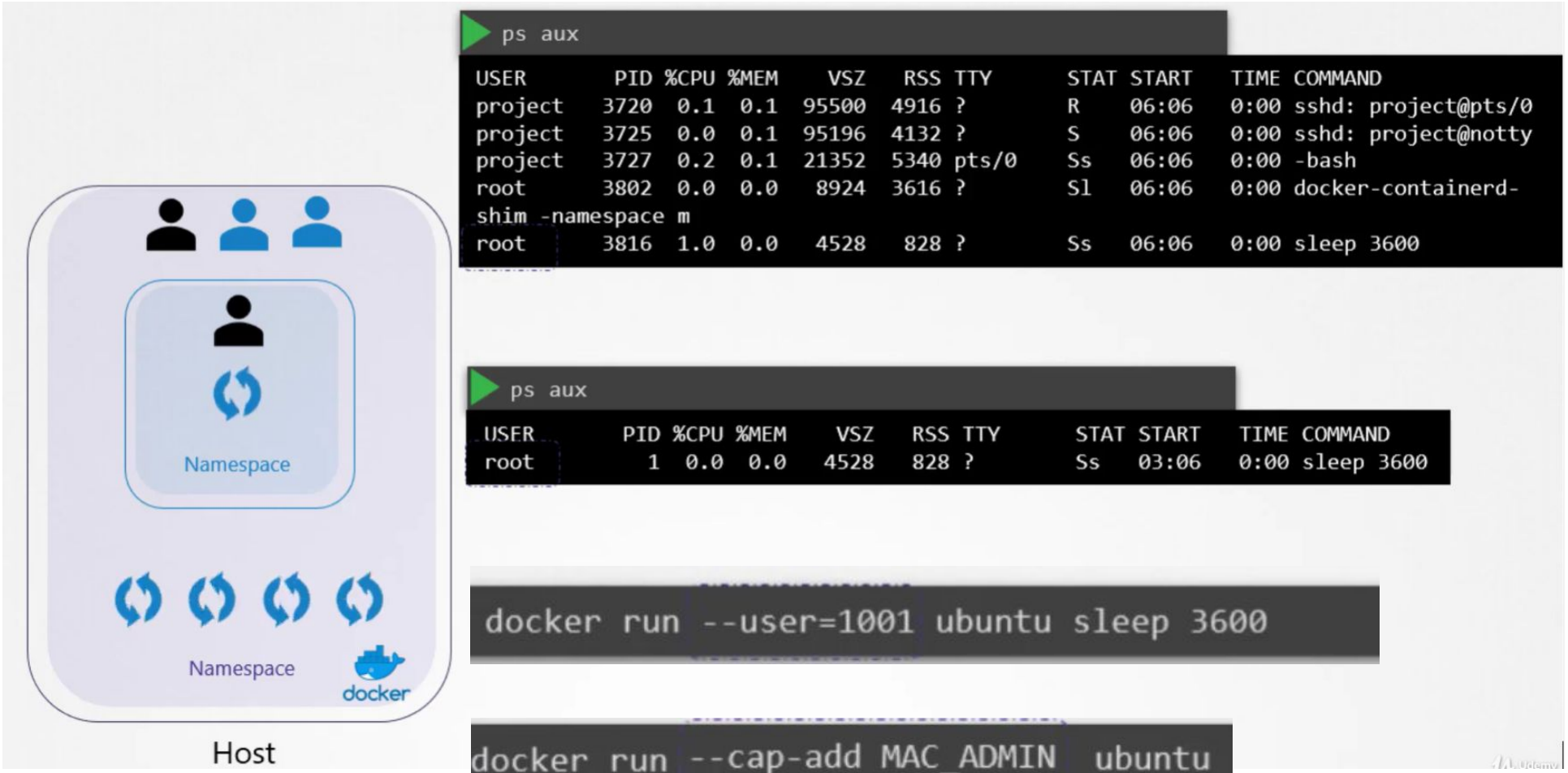
Práctica 6 – Variables de entorno

- Ahora juguemos con las variables para que nos muestren información de nuestro POD y de los contenedores.

https://kubernetes.io/docs/tasks/inject-data-application/environment-variable-expose-pod-information/

Security Context

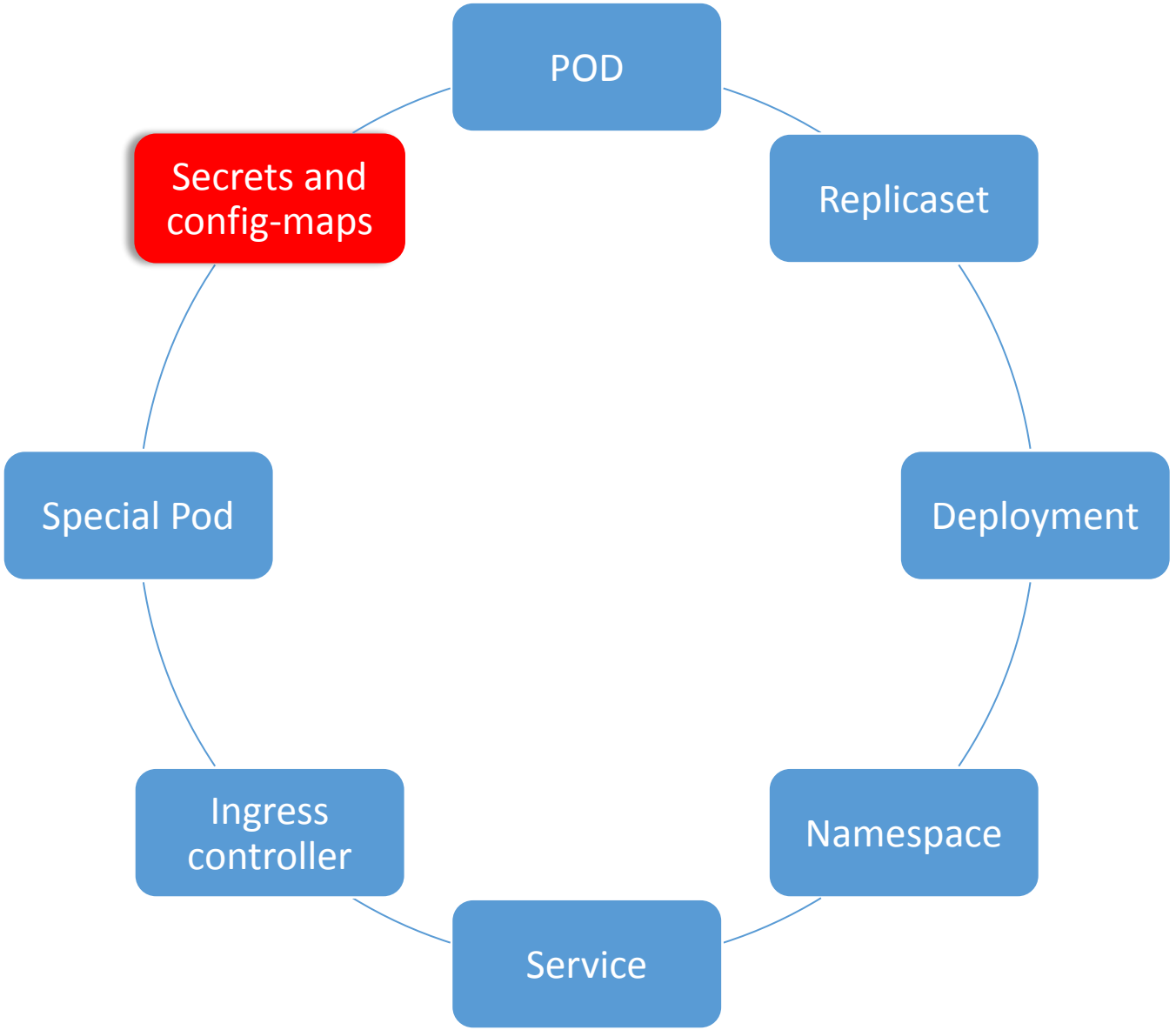
In Docker. Use another user not Root:



Pod Level – Pod Security Context

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    fsGroup: 2000
  volumes:
  - name: sec-ctx-vol
    emptyDir: {}
  containers:
  - name: sec-ctx-demo
    image: ignite.azurecr.io/nginx-demo
    volumeMounts:
    - name: sec-ctx-vol
      mountPath: /data/demo
    securityContext:
      runAsUser: 2000
      allowPrivilegeEscalation: false
      capabilities:
        add: ["NET_ADMIN", "SYS_TIME"]
      seLinuxOptions:
        level: "s0:c123,c456"
```

Componentes en K8S



ConfigMaps

- Flexible configuration model for Kubernetes
- ConfigMaps
  - ConfigMaps for configuration that doesn’t have sensitive information. However, there are pros/cons with each.
  - ConfigMaps designed to more conveniently support working with strings
  - ConfigMaps can be attached to a POD:
    - Like a file placed on a volume at runtime (useful for certificates).
    - An environment variable referenced by the POD

ConfigMaps

- Create from literal

  Kubectl create configmap <name> --from-literal=<key>=<value>

- Create from file

  Kubectl create configmap <name> --from-file=<path>

- Create from definition

config-map.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  APP_COLOR: blue
  APP_MODE: prod
```

ConfigMaps

- How to use

Práctica 7 – ConfigMaps

- Crear un configMap para indicar una variable de entorno en nuestro POD anterior

- Ver y describir los configmaps

  kubectl get configmaps

  kubectl describe configmaps <>

- Utilizar configmaps en nuestro POD

Secrets

- Secrets
  - Use Secrets for things which are actually secret like API keys, credentials, etc
  - Secret values are base64 encoded and automatically decoded for pods, but not strong encryption. You can encode it manually
  - Secrets can be attached to a POD:
    - Like a file placed on a volume at runtime (useful for certificates).
    - An environment variable referenced by the POD

Creating a Secret

- Create a secret using Kubectl
  - From a File (db-usrpassword and username)

kubectl create secret generic mysql-pass --from-file=./username.txt --from-file=./password.txt

```
spec:
  containers:
  - image: mysql:5.6
    name: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      valueFrom:
        secretKeyRef:
          name: mysql-pass
          key: password
    ports:
    - containerPort: 3306
      name: mysql
    volumeMounts:
    - name: mysql-persistent-storage
      mountPath: /var/lib/mysql
    - name: secret-storage
      mountPath: /etc/secretStore
  volumes:
  - name: mysql-persistent-storage
    persistentVolumeClaim:
      claimName: mysql-pv-claim
  - name: secret-storage
    secret:
      secretName: mysql-pass
```

Creating a Secret

- Create a secret using Kubectl
  - From the literal on the command line:

kubectl create secret generic mysql-pass
--from-literal=password=YOUR_PASS

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
    spec:
      containers:
      - image: mysql:5.6
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password
```

Práctica 8 - Secrets

- Crear un Secret para ser usado como usuario/password de BBDD en el despliegue de NGINX

```
kubectl create secret generic secret-backend-user --from-literal=backend-username='backend-admin'
kubectl create secret generic secret-db-user --from-literal=db-username='db-admin'
```

- Listar los secretos
- Ver el valor de los secretos … podemos verlos?
- Utilizar los secretos en el POD de NGINX y ver los valores dentro del bash del contenedor

Secrets in Pods

```yaml
envFrom:
  - secretRef:
      name: app-config
```

ENV

SINGLE ENV

```yaml
env:
  - name: DB_Password
    valueFrom:
      secretKeyRef:
        name: app-secret
        key: DB_Password
```

```yaml
volumes:
- name: app-secret-volume
  secret:
    secretName: app-secret
```

VOLUME

Etiquetas y Selectores

Etiquetas

Los labels o etiquetas son pares clave-valor que se adjuntan a los objetos como los Pods.

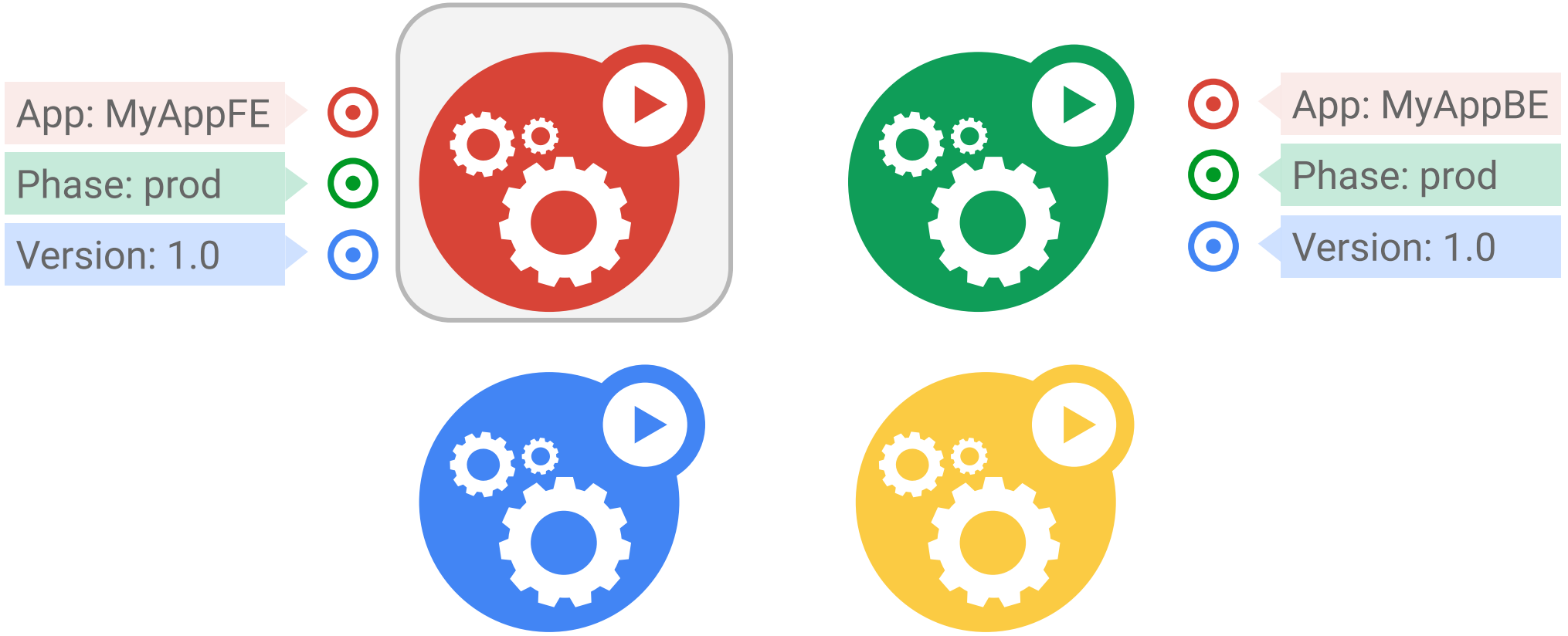
Los selectores son una forma de expresar cómo se seleccionan los objetos basados en sus labels.

Permiten **mantener organizados los elementos** en Kubernetes y ayudan a identificar los recursos que se despliegan.

Etiquetas/Labels

- Arbitrary metadata
- Attached to any API object
- Generally represent **identity**
- Queryable by **selectors**
  - Think SQL '*select ... where ...*'
- The **only** grouping mechanism
  - Pods under a ReplicationController
  - Pods in a Service
  - Capabilities of a node (constraints)

Selectores


App: MyAppFE
Phase: prod
Version: 1.0
App: MyAppBE
Phase: prod
Version: 1.0

Práctica 9 con Labels – Comandos Kubectl

- Crear un label en el pod X con un valor del label *environmet* a *dev*
- Actualizar el pod X con un nuevo valor para el label environment
- Actualizar todos los pods en el namespace default con el label status con el valor unhealthy
- Eliminar el label environment del pod X
- Ver todos los labels de los pods del namespace default
- Listar todos los recursos con un label en particular

Muchas gracias

datahack
INSPIRING BIG DATA