

MÓDULO 10

Gestión de implementación de Red Hat OpenShift

AUTOMATIZACIÓN DE IMPLEMENTACIONES DE APLICACIONES

- La implementación de nuevas versiones de su aplicación implica un proceso que debe repetir en cada actualización de la aplicación
- Este proceso requiere una estrategia de implementación que minimice el tiempo de inactividad de las aplicaciones y considere una serie de aspectos, como los siguientes:
 - Versiones de implementación nuevas y anteriores que se ejecutan simultáneamente.
 - La cantidad de recursos informáticos que requiere la implementación de una nueva versión de su aplicación
 - El tiempo de inactividad aceptable durante el proceso de implementación



RECURSO DEPLOYMENT

- El recurso Deployment es una API nativa de Kubernetes que usa el recurso ReplicaSet para garantizar que haya varios pods que ejecuten su aplicación en un momento dado.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-openshift
spec:
  selector:
    matchLabels:
      app: hello-openshift
  replicas: 3 1
  strategy:
    type: RollingUpdate 2
    rollingUpdate: 3
      maxSurge: 1
      maxUnavailable: 1
  template: 4
    ...output omitted...
```

- ¹ Cantidad de réplicas de pod cuando se completa la implementación.
- ² La estrategia para la implementación. Puede ser de los tipos RollingUpdate o Recreate.
- ³ La configuración rollingUpdate controla cómo los nuevos pods reemplazan a los pods de la versión anterior.
- ⁴ La plantilla para definir los pods de la aplicación. El campo `.spec.template.spec` corresponde al campo `.spec` del objeto Pod.

EL RECURSO DEPLOYMENTCONFIG

- El recurso DeploymentConfig, que es específico de OpenShift, usa un recurso ReplicationController para garantizar que haya varios pods que ejecuten su aplicación en un momento dado.
- Un pod específico llamado Deployment Pod realiza el proceso de implementación.
- Puede eliminar el pod de implementación después de que finalice una implementación y el pod tenga el estado Completed.


Project: demo-49 ▼


[DeploymentConfigs](#) > [DeploymentConfig details](#)

DC dc-metro-map Actions ▼

[Details](#) [YAML](#) [ReplicationControllers](#) [Pods](#) [Environment](#) [Events](#)

DeploymentConfig details



Name dc-metro-map	Latest version 1
Namespace NS demo-49	Message config change
Labels Edit 	Update strategy Rolling
<div><div>app=dc-metro-map</div><div>app.kubernetes.io/component=dc-metro-map</div><div>app.kubernetes.io/instance=dc-metro-map</div><div>app.kubernetes.io/name=dc-metro-map</div><div>app.kubernetes.io/part-of=dc-metro-map</div><div>app.openshift.io/runtime=nodejs</div></div>	Max unavailable 25% of 1 pod
	Max surge 25% greater than 1 pod

DEPLOYMENT VS DEPLOYMENTCONFIG

- En comparación con el recurso Deployment, el recurso DeploymentConfig proporciona características adicionales, como las siguientes:
 - Triggers que inician un proceso de implementación.
 - Estrategias personalizadas de implementación que se ejecutan dentro de un pod de implementación.
 - Enlaces de ciclo de vida para definir el comportamiento personalizado durante la implementación.
 - Capacidades de reversión en caso de falla de implementación.
 - Favorece la coherencia sobre la disponibilidad.

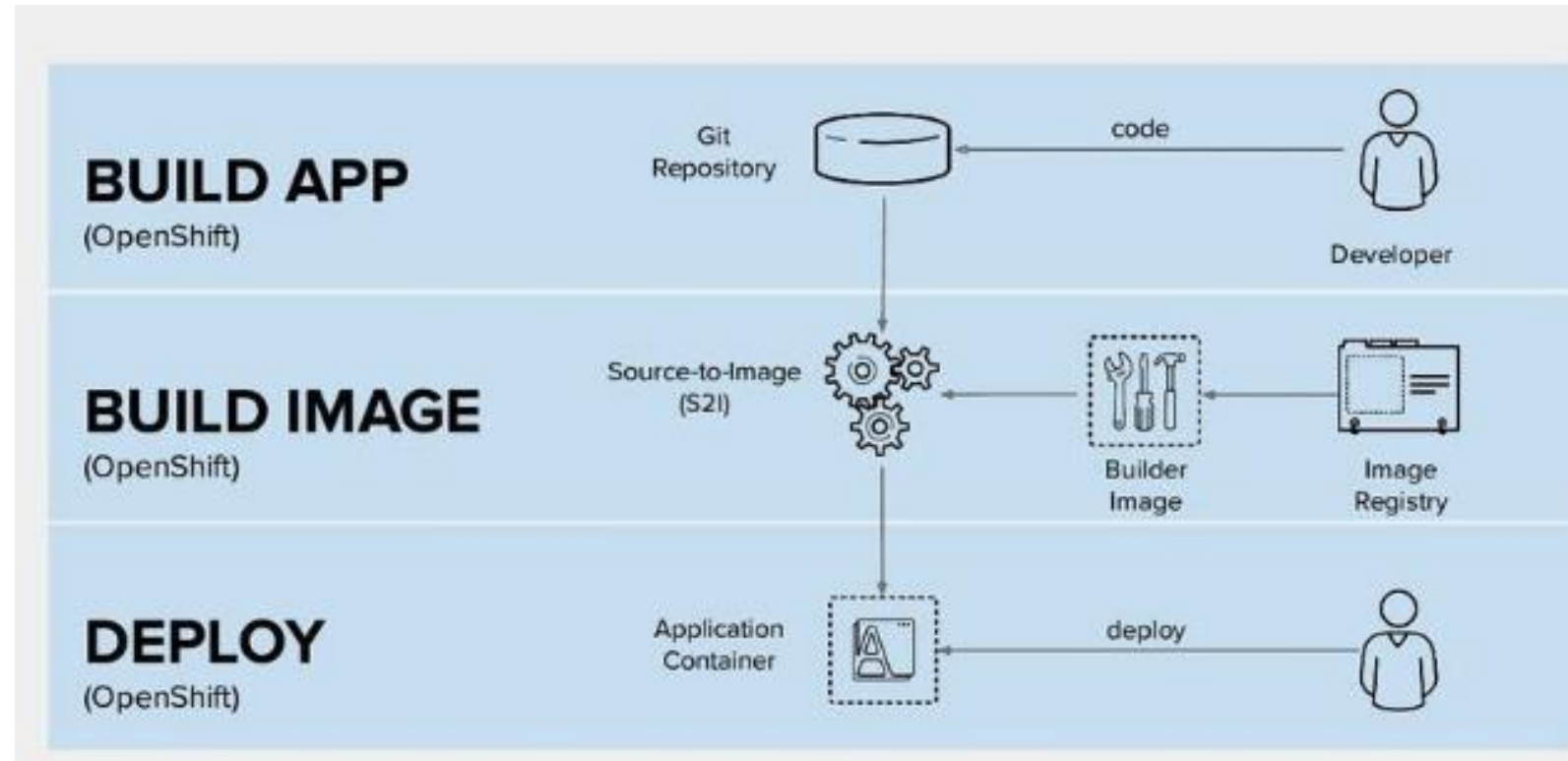


PROCESO DE DEPLOYMENT

1.- Integración de código

2.- Implementación

3.- Lanzamiento



DEPLOYMENTCONFY YAML

```
apiVersion: apps.openshift.io/v1
kind: DeploymentConfig
metadata:
  name: frontend
spec:
  selector:
    name: frontend
  replicas: 3 ❶
  strategy:
    type: Rolling ❷
  triggers: ❸
  - type: ConfigChange
  - imageChangeParams:
      automatic: true
      containerNames:
        - helloworld
      from:
        kind: ImageStreamTag
        name: hello-openshift:latest
      type: ImageChange
  template: ❹
    metadata:
      labels:
        name: frontend
    spec:
      containers:
        - name: helloworld
          image: openshift/origin-ruby-sample
```

- ❶ Cantidad de réplicas de pod cuando se completa la implementación.
- ❷ La estrategia para la implementación. Puede ser de los tipos Rolling, Recreate o Custom.
- ❸ La lista de desencadenadores que inician una nueva implementación.
- ❹ La plantilla para el pod de la aplicación. El campo `.spec.template.spec` corresponde al campo `.spec` del objeto Pod.

USO DE DEPLOYMENTCONFIG CON HOOKS (ENLACES) DE CICLO DE VIDA

- Las estrategias Recreate y Rolling admiten enlaces de ciclo de vida.
- Las implementaciones de OpenShift cuentan con tres enlaces de ciclo de vida:
 - Hook previo al ciclo de vida
 - **Hook en medio del ciclo de vida**
 - Hook posterior al ciclo de vida



HOOKS DE CICLO DE VIDA

- Estos enlaces de ciclo de vida se definen en la propiedad `spec.strategy` en uno de los siguientes atributos:
 - `rollingParams` para las estrategias Rolling. Este atributo acepta las propiedades pre y post según el tipo de enlace.
 - `recreateParams` para las estrategias Recreate. Este atributo acepta las propiedades pre, mid y post según el tipo de enlace.
- Políticas **failurePolicy**, que define la acción a tomar cuando se encuentra una falla en el enlace.
- Existen tres políticas: Cancelar, Reintentar e Ignorar

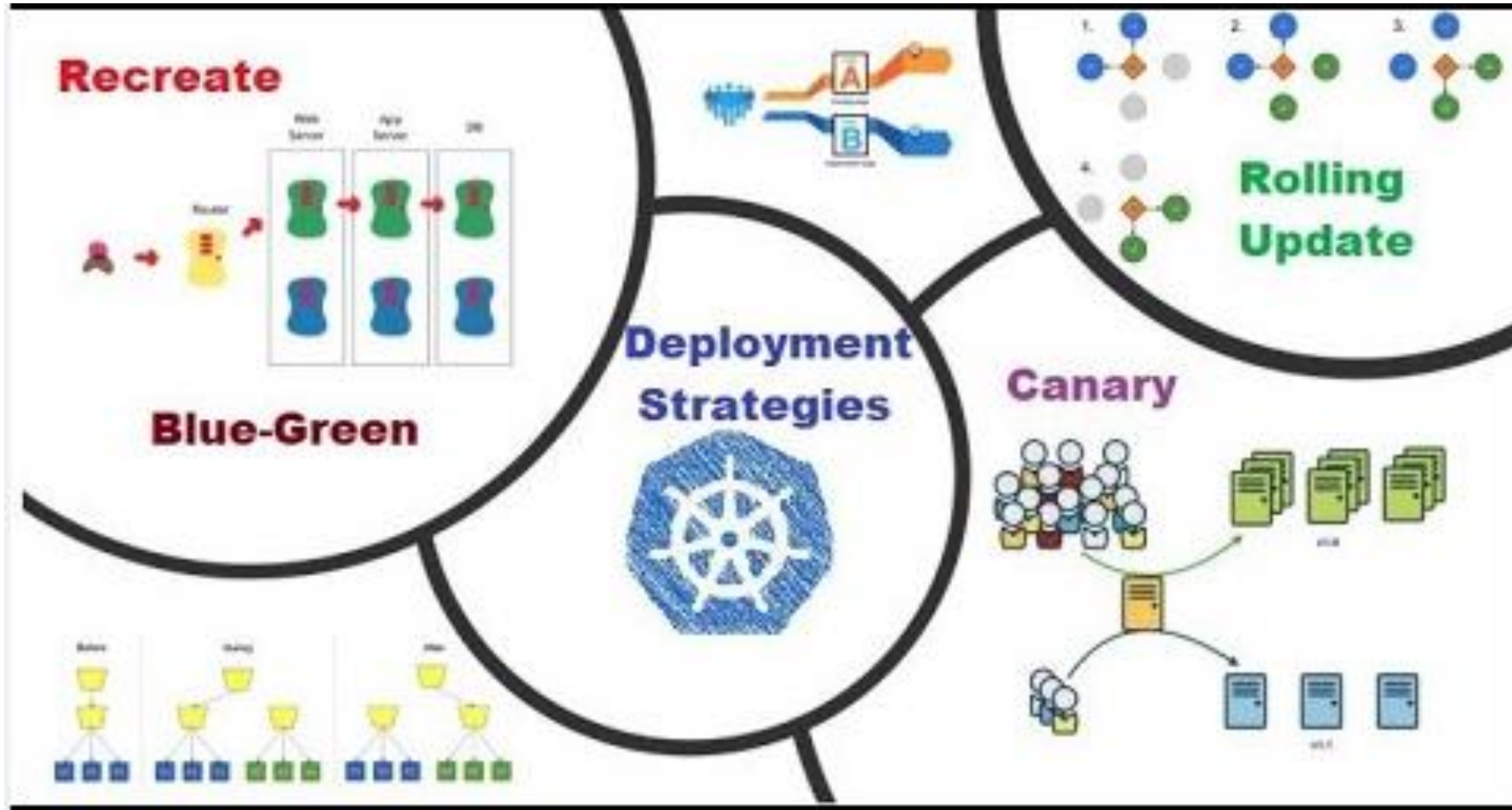
```
apiVersion: apps.openshift.io/v1
kind: DeploymentConfig
metadata:
  name: frontend
spec:
  ...output omitted...
  strategy:
    type: Rolling
    rollingParams:
      pre: ❶
        failurePolicy: Abort ❷
        execNewPod:
          containerName: schema-validator ❸
          command: [ "/bin/bash", "-c", "/opt/validate_schema.sh" ] ❹
```

- ❶ El atributo `pre` define un enlace previo al ciclo de vida.
- ❷ Este enlace cancela nuevas implementaciones si falla el comando del enlace.
- ❸ El contenedor de la definición de plantilla de pod que crea el enlace.
- ❹ El comando que se ejecuta dentro del contenedor `schema-validator`.

ESTRATEGIAS DE IMPLEMENTACIÓN

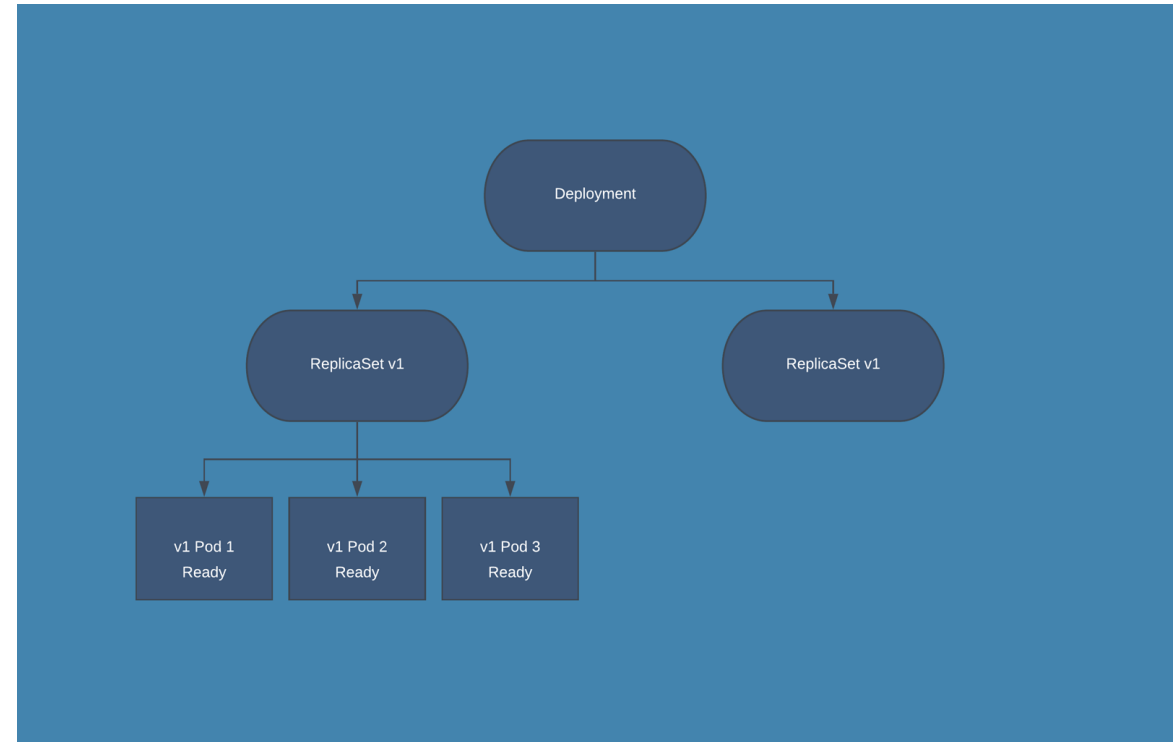


ESTRATEGIAS DE IMPLEMENTACIÓN



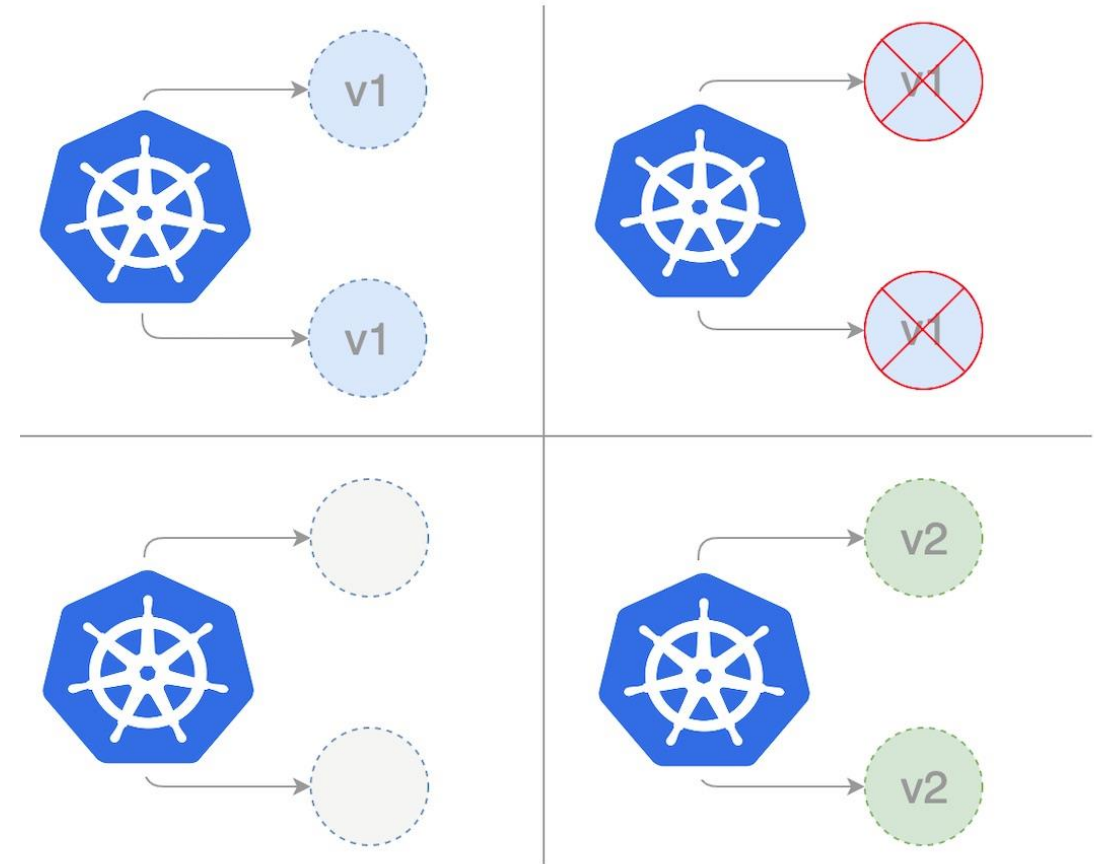
ESTRATEGIAS DE IMPLEMENTACIÓN CON RECURSOS DE IMPLEMENTACIÓN: ROLLING UPDATE

- **Rolling Strategy:**
 - Las implementaciones graduales en OpenShift son de tipo canary.
 - OpenShift prueba una nueva versión, canary, antes de reemplazar todas las instancias anteriores.
- **Recreate:**
 - En esta estrategia, OpenShift primero detiene todos los pods que actualmente están en ejecución e inicia los pods con la nueva versión de la aplicación.
 - Esta estrategia provoca tiempo de inactividad.



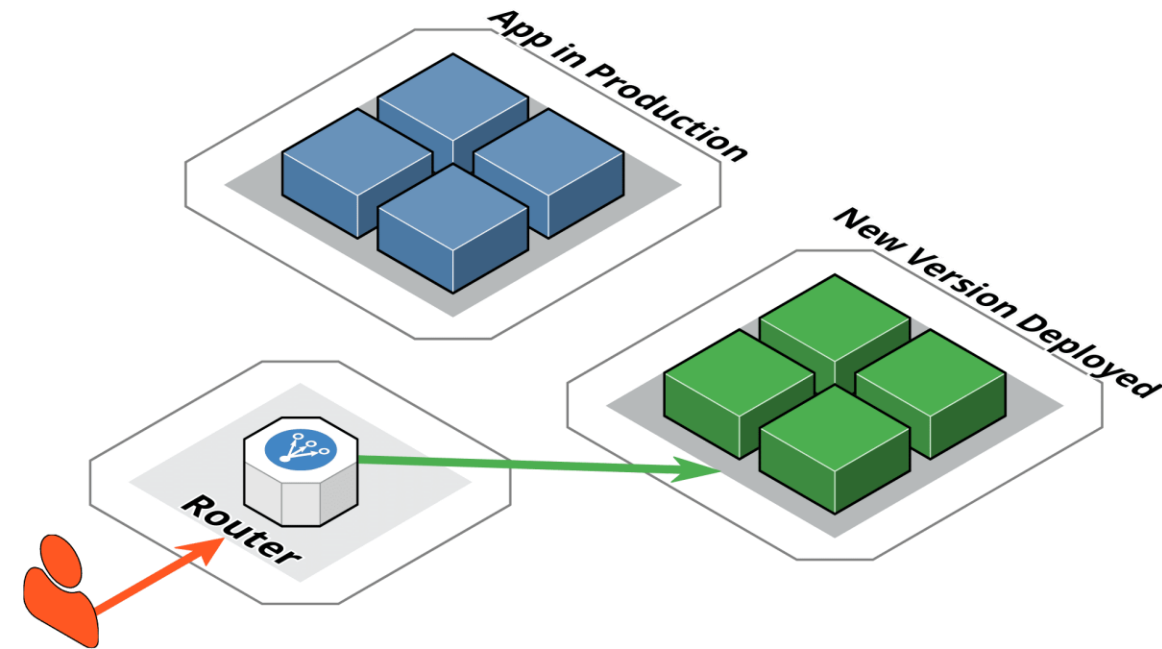
ESTRATEGIAS DE IMPLEMENTACIÓN CON RECURSOS DE IMPLEMENTACIÓN: RECREATE

- **Recreate:**
 - En esta estrategia, OpenShift primero detiene todos los pods que actualmente están en ejecución e inicia los pods con la nueva versión de la aplicación.
 - Esta estrategia provoca tiempo de inactividad.



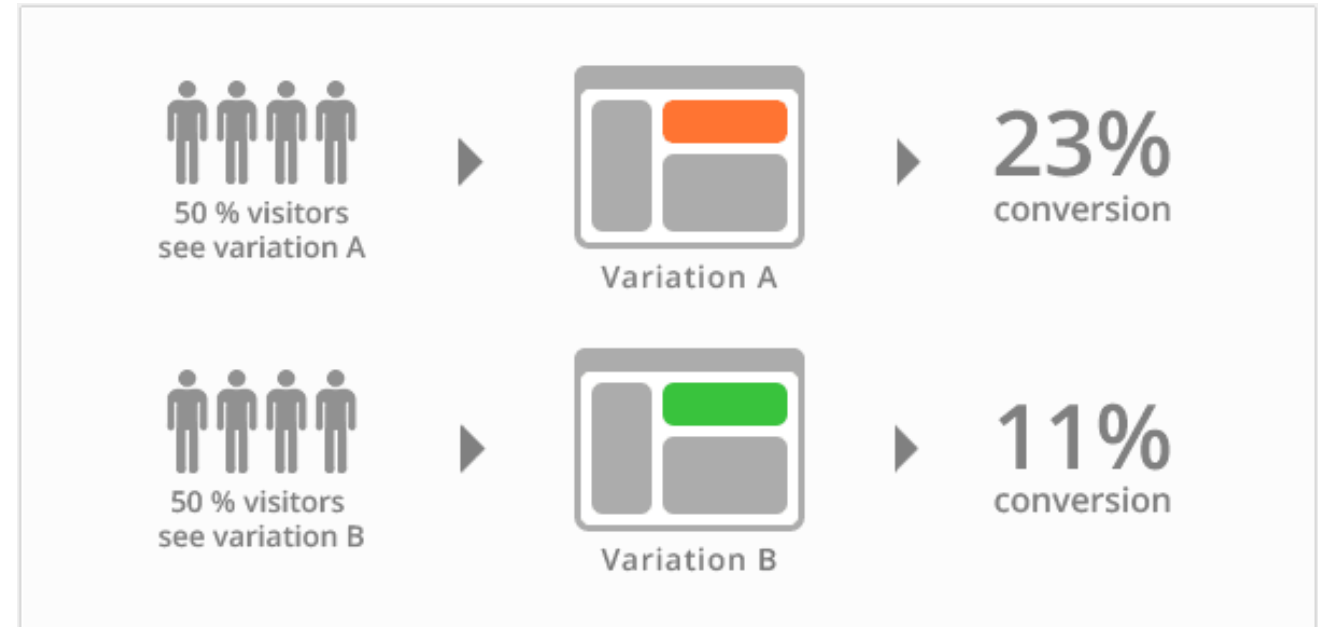
ESTRATEGIAS DE IMPLEMENTACIÓN CON EL ENRUTADOR DE RED HAT OPENSIFT: DEPLOYMENT BLUE/GREEN

- En las implementaciones azul-verde se cuenta con dos entornos que se ejecutan de manera concurrente; cada uno con una versión diferente de la aplicación.
- El enrutador OpenShift se utiliza para dirigir tráfico de la versión actualmente en producción (Verde) a la versión nueva (Azul).



ESTRATEGIAS DE IMPLEMENTACIÓN CON EL ENRUTADOR DE RED HAT OPENSIFT: DEPLOYMENT A/B

- La estrategia de implementación A/B le permite implementar una nueva versión de la aplicación para un conjunto limitado de usuarios.
- Puede configurar OpenShift para enrutar la mayoría de las solicitudes a la versión anterior de la aplicación y un número limitado de solicitudes a la nueva versión.

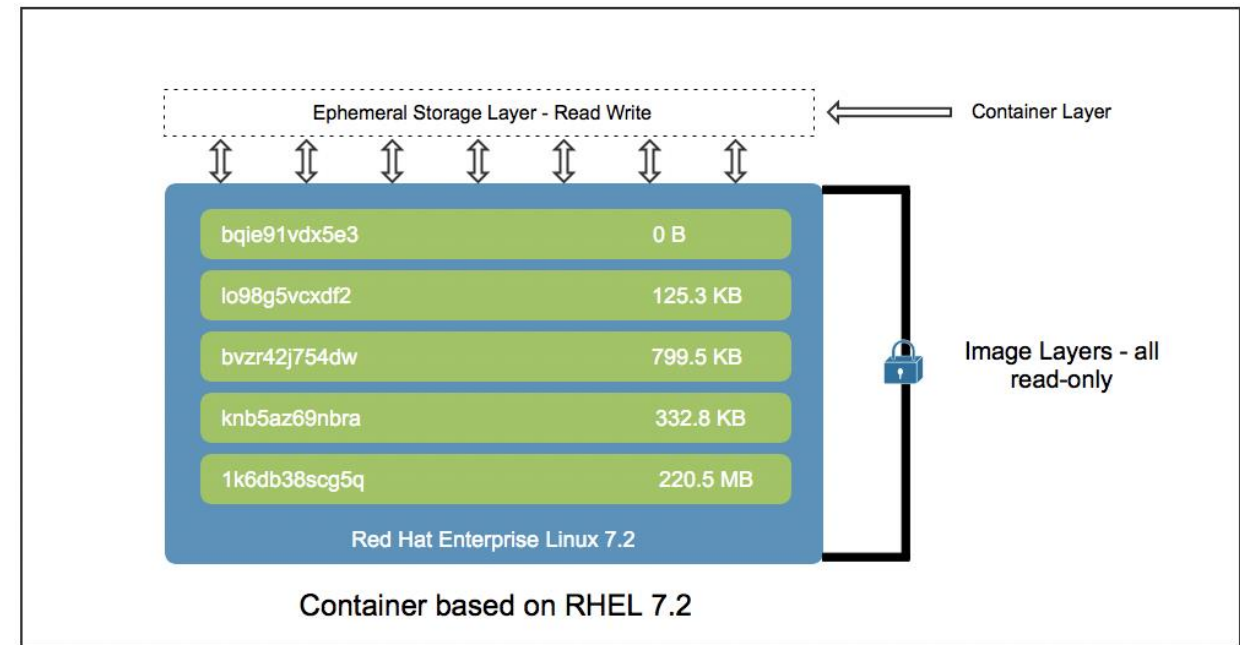


IMPLEMENTACIÓN DE APLICACIONES CON ESTADO

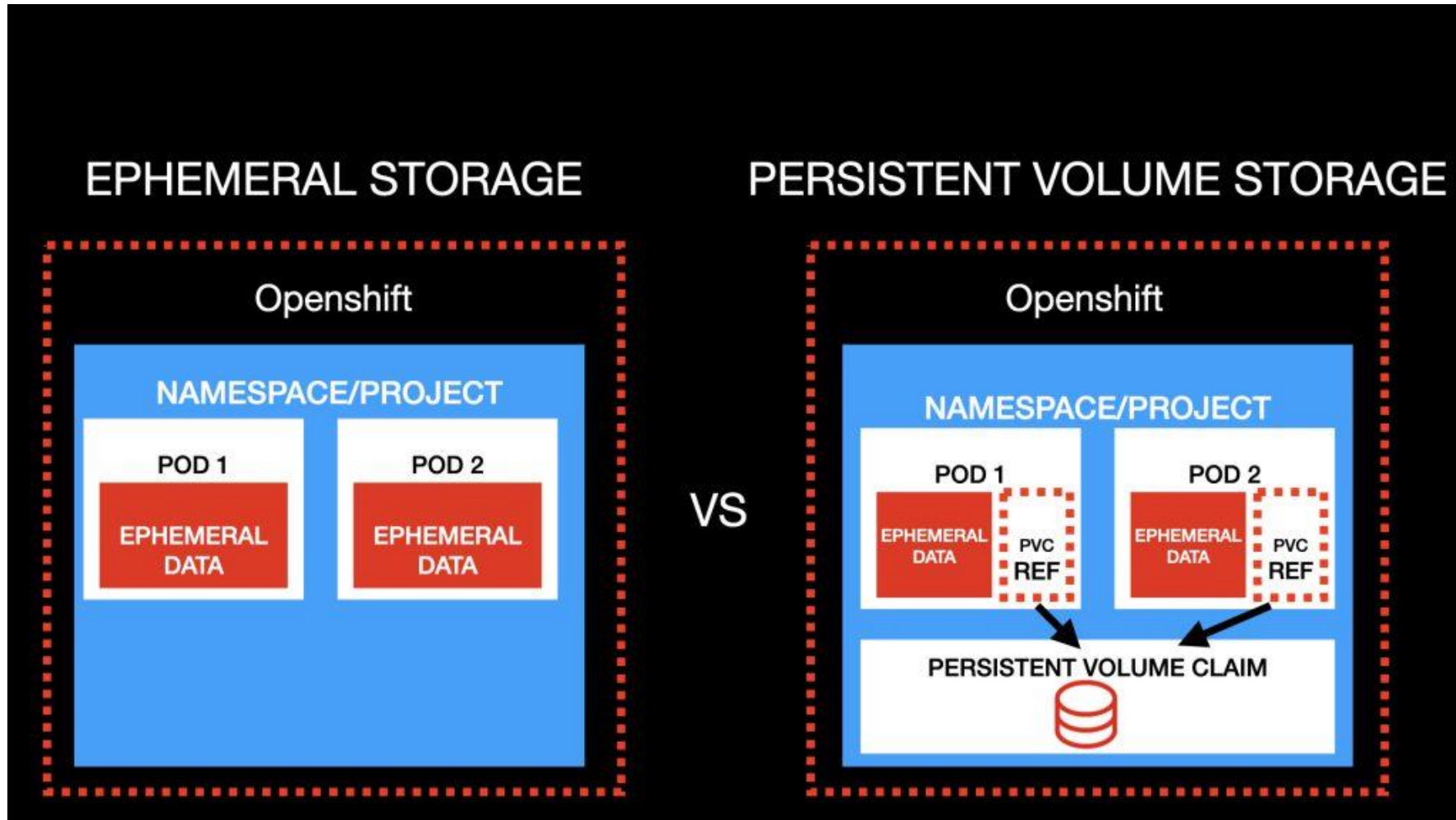


UBICACIONES DE ALMACENAMIENTO PERSISTENTE

- El almacenamiento de contenedores es efímero.
- Las aplicaciones en contenedores funcionan asumiendo que siempre comienzan con un almacenamiento vacío.
- Un contenedor en ejecución obtiene una nueva capa sobre su imagen de contenedor base, y esta capa es el almacenamiento del contenedor.
- La capa de almacenamiento del contenedor es exclusiva del contenedor en ejecución.

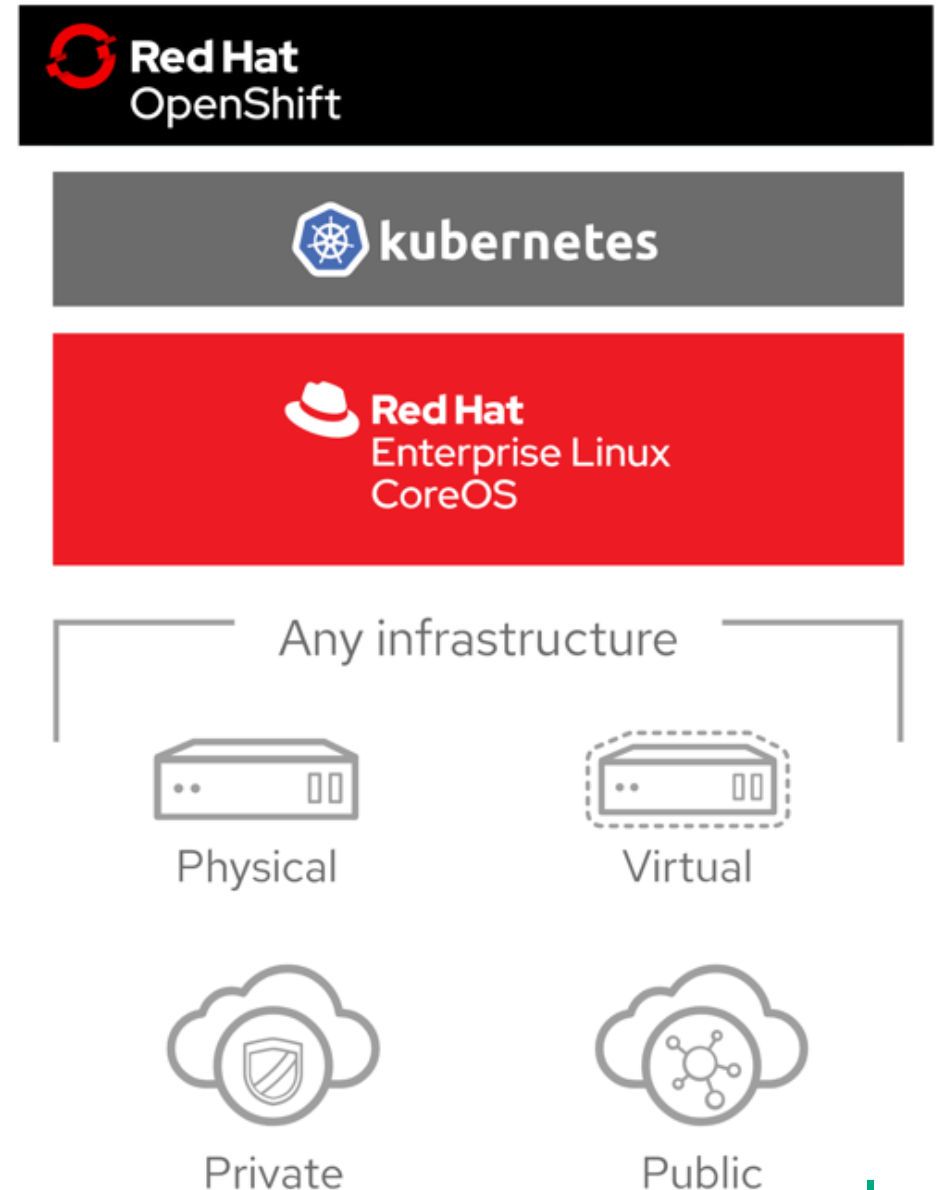


EPHEMERAL VS PERSISTENT VOLUME



PERSISTENT STORAGE

- De forma predeterminada, los contenedores en ejecución utilizan almacenamiento efímero dentro del contenedor.
- Las pods constan de uno o más contenedores que se implementan juntos, comparten el mismo almacenamiento y otros recursos, y pueden ser creados, iniciados, detenidos o destruidos en cualquier momento.
- El uso de almacenamiento efímero significa que los datos escritos en el sistema de archivos dentro del contenedor se pierden cuando se detiene el contenedor.
- Al implementar aplicaciones que requieren datos persistentes cuando el contenedor está detenido, OpenShift utiliza volúmenes persistentes (PV) de Kubernetes para aprovisionar almacenamiento persistente para pods.



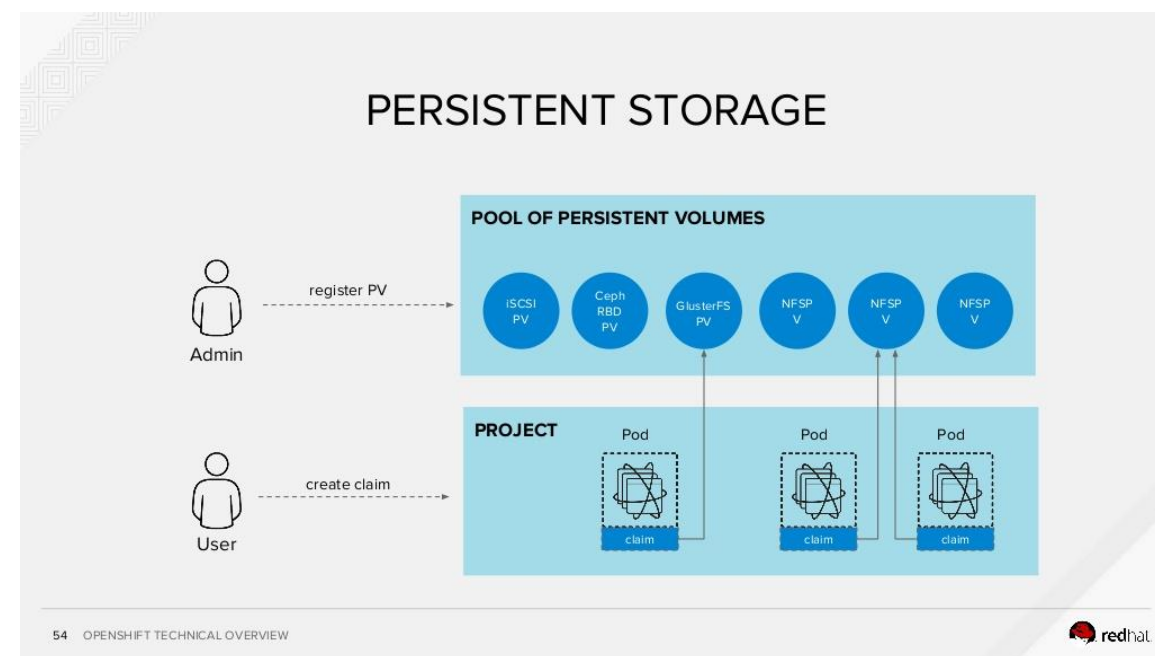
COMPONENTES DE PERSISTENT STORAGE

- Persistent Volume:

Un PV es un recurso en el clúster de OpenShift, definido por un objeto de API PersistentVolume, que representa una pieza de almacenamiento en red existente en el clúster que se ha provisionado por un administrador. Es un recurso en el clúster al igual que un nodo es un clúster recurso. Los PV tienen un ciclo de vida independiente de cualquier pod individual que utilice el PV.

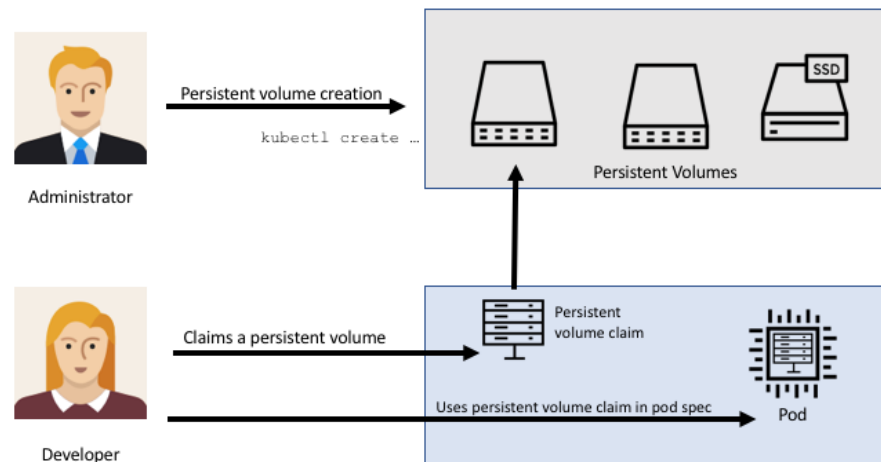
- Persistent Volume Claim :

Los PVC se definen mediante un objeto API PersistentVolumeClaim, que representa una solicitud de almacenamiento por un desarrollador. Es similar a un pod en el que los pods consumen recursos de nodo y PVC consume recursos fotovoltaicos.



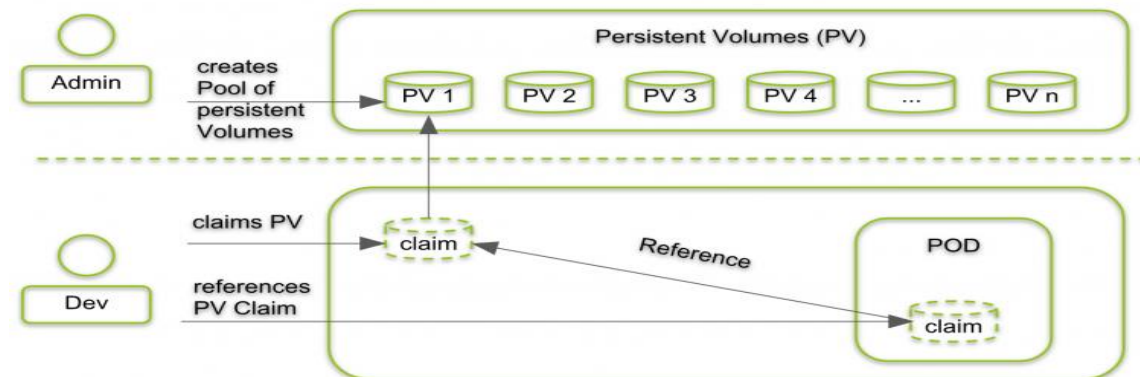
ALMACENAMIENTO DE PERSISTENT VOLUME CLAIMS

- Para que los pods comiencen a usar los volúmenes, deben reclamarse (a través de un reclamo de volumen persistente) y hacer referencia al reclamo en la especificación de un pod.
- Un Reclamo de volumen persistente describe la cantidad y las características del almacenamiento requerido por el pod, encuentra los volúmenes persistentes coincidentes y los reclama. Las clases de almacenamiento describen información de volumen predeterminada (sistema de archivos, tamaño, tamaño de bloque, etc.).



PERSISTENT VOLUME CLAIMS

- Si nuestro pod necesita un volumen, necesitamos hacer una solicitud de almacenamiento usando un objeto del tipo *PersistentVolumenCliams*.
- Cuando creamos un *PersistentVolumenClaims*, se asignará un *PersistentVolumen* que se ajuste a la petición. Esta asignación se puede configurar de dos maneras distintas:
 - Estática:** Primero se crean todos los *PersistentVolumenCliams* por parte del administrador, que se irán asignando conforme se vayan creando los *PersistentVolumen*.
 - Dinámica:** En este caso necesitamos una "provision de almacenamiento", de tal manera que cada vez que se cree un *PersistentVolumenClaim*, se creará bajo demanda un *PersistentVolumen* que se ajuste a las características seleccionadas.



PLUGINGS SOPORTADOS CON OPENSIFT PARA ALMACENAMIENTO PERSISTENTE

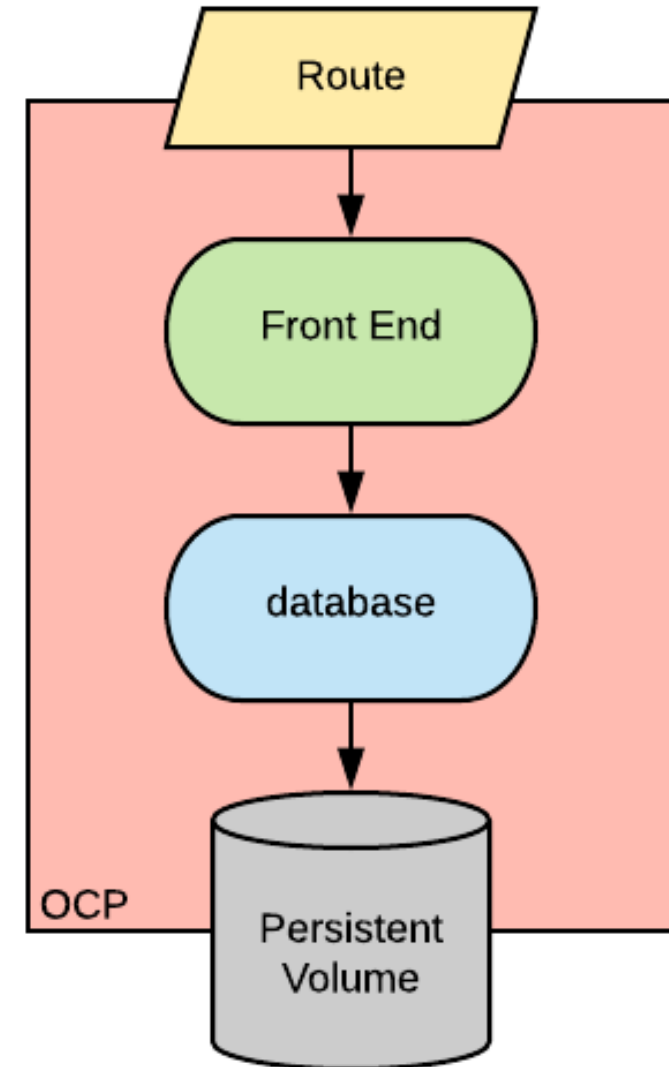
* OpenShift utiliza complementos que admiten los siguientes back-ends para el almacenamiento persistente:

- NFS
- GlusterFS
- OpenStack Cinder
- Ceph RBD
- AWS Elastic Block Store (EBS)
- GCE Persistent Disk
- iSCSI
- Fibre Channel
- Azure Disk and Azure File
- FlexVolume (allows for the extension of storage back-ends that do not have a built-in plug-in)
- VMWare vSphere
- Dynamic Provisioning and Creating Storage Classes
- Volume Security
- Selector-Label Volume Binding



ALMACENAMIENTO PERSISTENTE

- Un PersistentVolumen es un objeto que representa los volúmenes disponibles en el cluster.
- En él se van a definir los detalles del [back-end](#) de almacenamiento que vamos a utilizar, el tamaño disponible, los [modos de acceso](#), las [políticas de reciclaje](#), etc.



MODOS DE ACCESO A VOLÚMENES PERSISTENTES



- La siguiente tabla describe los modos de acceso:

ACCESS MODE	CLI ABBREVIATION	DESCRIPTION
ReadWriteOnce	RWO	The volume can be mounted as read/write by a single node.
ReadOnlyMany	ROX	The volume can be mounted read-only by many nodes.
ReadWriteMany	RWX	The volume can be mounted as read/write by many nodes.

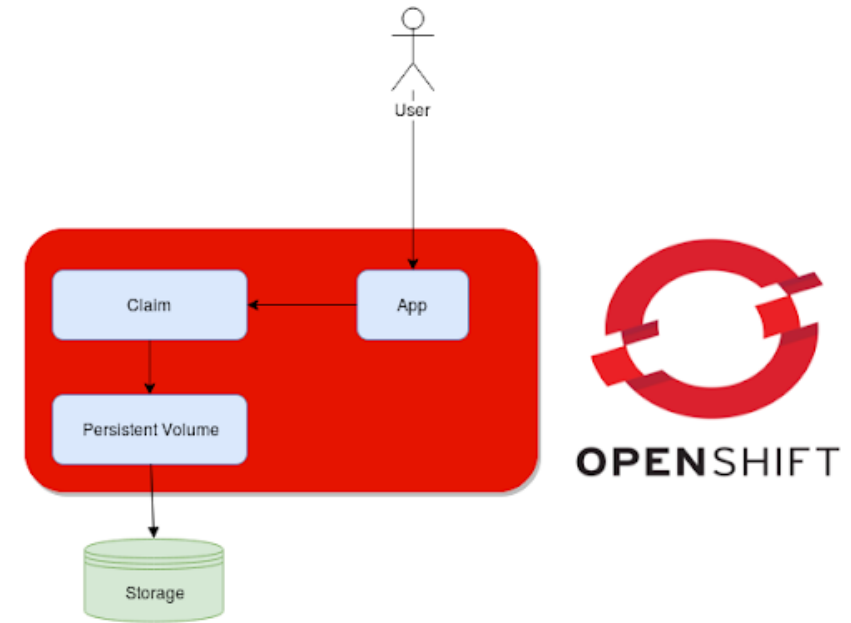
- Clases de almacenamiento de volumen persistente **PV Claims** puede solicitar opcionalmente una clase de almacenamiento específica especificando su nombre en el Atributo **storageClassName**.



CREACIÓN DE RECURSOS DE PV Y PVC

La interacción entre PV y PVC tiene el siguiente ciclo de vida:

- Crea el volumen persistente
- Define una declaración de volumen persistente
- Usa almacenamiento persistente



POLÍTICAS DE RECLAMACIÓN: RECYCLING & RETAIN

- De forma predeterminada, los volúmenes persistentes se establecen en Retain.
- La política de recuperación de retención permite Recuperación del recurso.
- Cuando se elimina la reclamación de volumen persistente, el volumen persistente todavía existe y el volumen se considera liberado
- Un administrador puede recuperar manualmente el volumen.
- Los volúmenes NFS con su política de reclamación establecida en Reciclar se eliminan después de liberarse de su reclamo



SOLICITUD DE VOLÚMENES PERSISTENTES

- Cuando una aplicación requiere almacenamiento, crea un objeto PersistentVolumeClaim (PVC) para solicitar un recurso de almacenamiento dedicado del grupo de clústeres.
- El siguiente contenido de un archivo llamado pvc.yaml es una definición de ejemplo para un PVC:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myapp
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

- El PVC define los requisitos de almacenamiento para la aplicación, como la capacidad o el rendimiento. Cuando crees el PVC, usa el comando **oc create**:

```
[admin@host ~]$ oc create -f pvc.yaml
```

RECURSOS PERSISTENT VOLUME

- Si OpenShift encuentra una coincidencia, vincula el objeto PersistentVolume al objeto PersistentVolumeClaim.
- Para enumerar los PVC en un proyecto, usa el comando **oc get pvc**:

```
[admin@host ~]$ oc get pvc
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
myapp	Bound	pv0001	1Gi	RWO		6s

PERSISTENT VOLUMEN EN UNA APLICACIÓN

```
apiVersion: "v1"
kind: "Pod"
metadata:
  name: "myapp"
  labels:
    name: "myapp"
spec:
  containers:
    - name: "myapp"
      image: openshift/myapp
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/var/www/html"
          name: "pvol" ❶
  volumes:
    - name: "pvol" ❷
      persistentVolumeClaim:
        claimName: "myapp" ❸
```

- ❶ This section declares that the **pvol** volume mounts at **/var/www/html** in the container file system.
- ❷ This section defines the **pvol** volume.
- ❸ The **pvol** volume references the **myapp** PVC. If OpenShift associates an available persistent volume to the **myapp** PVC, then the **pvol** volume refers to this associated volume.

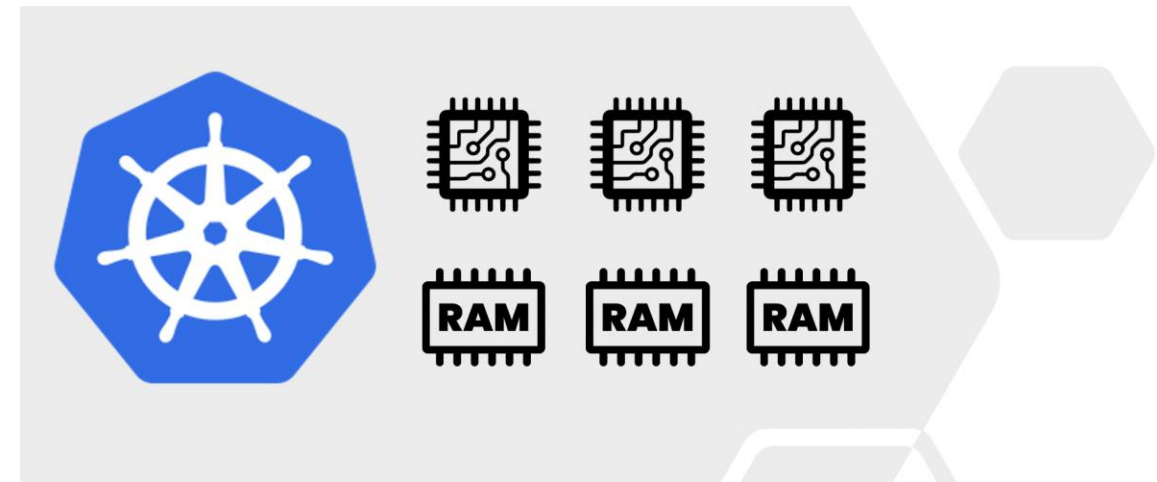


GESTIÓN DE RECURSOS DE LA APLICACION



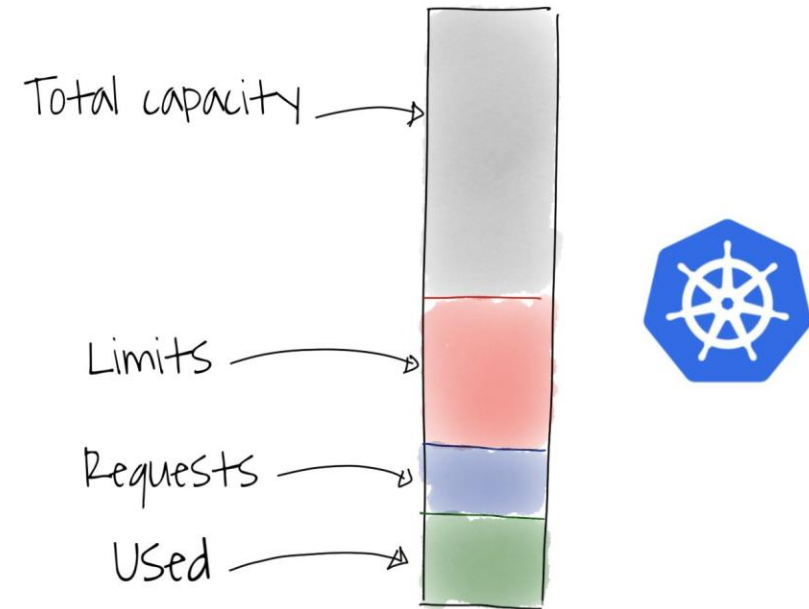
GESTIÓN DE RECURSOS PARA PODS Y CONTENEDORES

- Es importante controlar los recursos de los Pods ya que si no es así los pods pueden llegar a destruir o tirar a bajo un nodo.
- Cuando especifica la solicitud de recursos para contenedores en un Pod, el [kube-scheduler](#) usa esta información para decidir en qué nodo colocar el Pod.
- Cuando especifica un *límite* de recursos para un contenedor, el kubelet impone esos límites para que el contenedor en ejecución no pueda usar más de ese recurso que el límite establecido



LIMITS & REQUEST

- Las solicitudes y los límites son los mecanismos que utiliza Kubernetes para controlar recursos como la CPU y la memoria.
- Si un contenedor solicita un recurso, **Kubernetes** solo lo programará en un nodo que pueda darle ese recurso..
- Las **solicitudes** son lo que el contenedor está garantizado. Si un contenedor solicita un recurso, Kubernetes solo lo programará en un nodo que pueda darle ese recurso.
- Los **límites**, por otro lado, aseguran que un contenedor nunca supere un cierto valor. El contenedor solo puede subir hasta el límite y luego está restringido.



LIMITAR RECURSOS RAM

- La estructura de un fichero con Limits (RAM):

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: memory-demo
5  spec:
6    containers:
7    - name: memory-demo-ctr
8      image: polinux/stress
9      resources:
10       limits:
11         memory: "200Mi"
12       requests:
13         memory: "100Mi"
14      command: ["stress"]
15      args: ["--vm", "1", "--vm-bytes", "150M", "--vm-hang", "1"]
```

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: memory-demo
5  spec:
6    containers:
7    - name: memory-demo-ctr
8      image: polinux/stress
9      resources:
10       limits:
11         memory: "200Mi"
12       requests:
13         memory: "100Mi"
14      command: ["stress"]
15      args: ["--vm", "1", "--vm-bytes", "250M", "--vm-hang", "1"]
```

- Superando Limites lo que pasaría es que no llegaría a crear el pod.

```
MacBook-Pro-Juanlle:limits-requests Juanlle$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
deployment-test-b7c99d94b-j628z     1/1     Running   0           22h
deployment-test-b7c99d94b-lsrrk     1/1     Running   0           22h
deployment-test-b7c99d94b-wpr6n     1/1     Running   0           22h
memory-demo                         0/1     OOMKilled 0            6s
```

SUPERAR LIMITES DEL NODO

- ¿Que pasaría si superar los limites de recursos del nodo ?

```
apiVersion: v1
kind: Pod
metadata:
  name: memory-demo
spec:
  containers:
  - name: memory-demo-ctr
    image: polinux/stress
    resources:
      limits:
        memory: "1000Gi"
      requests:
        memory: "1000Gi"
    command: ["stress"]
    args: ["--vm", "1", "--vm-bytes", "250M", "--vm-hang", "1"]
```

- El pod quedaría siempre en estado pending.

NAME	READY	STATUS	RESTARTS	AGE
deployment-test-b7c99d94b-j628z	1/1	Running	0	22h
deployment-test-b7c99d94b-lsrrk	1/1	Running	0	22h
deployment-test-b7c99d94b-wpr6n	1/1	Running	0	22h
memory-demo	0/1	Pending	0	6s

LIMITAR RECURSOS CPU

- Fichero yaml en el que definiremos los Limits & Request de la CPU

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: cpu-demo
5  spec:
6    containers:
7    - name: cpu-demo-ctr
8      image: vish/stress
9      resources:
10       limits:
11         cpu: "1"
12       requests:
13         cpu: "0.5"
14    args:
15    - -cpus
16    - "2"
```

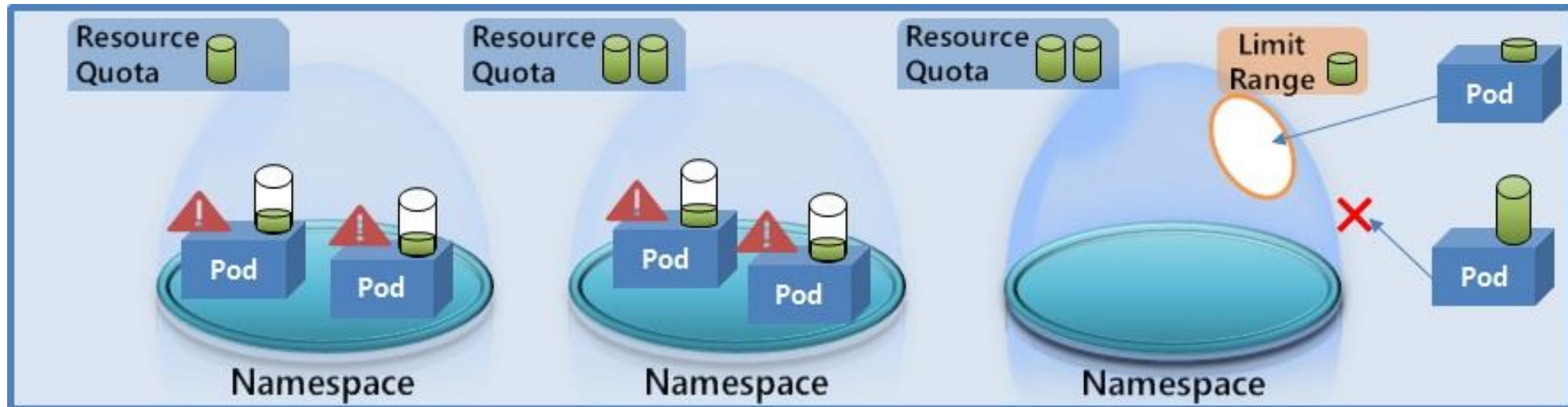
```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: cpu-demo2
5  spec:
6    containers:
7    - name: cpu-demo-ctr
8      image: vish/stress
9      resources:
10       limits:
11         cpu: "100"
12       requests:
13         cpu: "100"
14    args:
15    - -cpus
16    - "2"
```

- Kubernetes permitirá que el pod se ejecute:

NAME	READY	STATUS	RESTARTS	AGE
cpu-demo	1/1	Running	0	36s
deployment-test-b7c99d94b-j628z	1/1	Running	0	22h
deployment-test-b7c99d94b-lsrrk	1/1	Running	0	22h
deployment-test-b7c99d94b-wpr6n	1/1	Running	0	22h

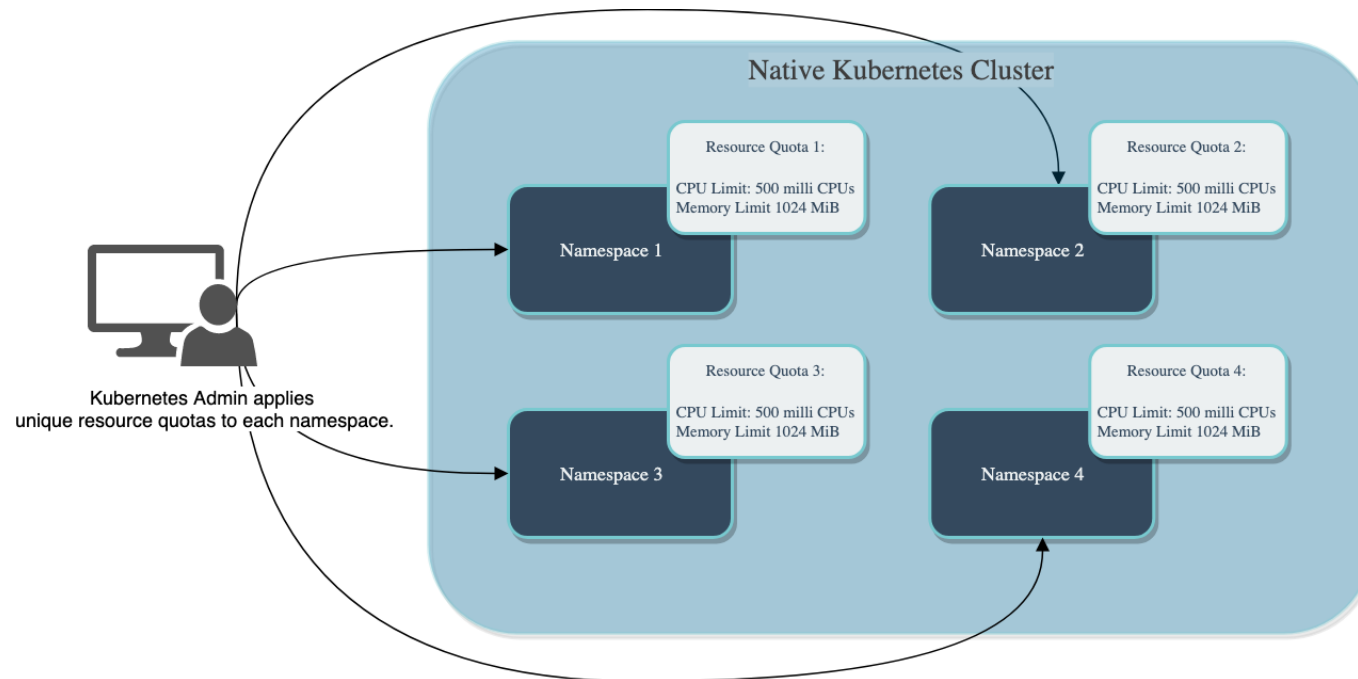
LIMIT RANGE

- El Limit Range nos permite limitar recursos nivel de objetos por namespace.
- Aplicando restricciones de asignación de recursos, los administradores de clústeres se aseguran del cumplimiento del consumo de recursos por espacio de nombre ([Namespace](#)).



RESOURCEQUOTA

- Cuando varios usuarios o equipos comparten un clúster con un número fijo de nodos, existe la preocupación de que un equipo pueda usar más recursos de los que le corresponden Y para solucionar esto están las **RESOURCEQUOTA**.
- Proporciona restricciones que limitan el consumo de recursos por **namespace**.
- Puede limitar la cantidad de objetos que se pueden crear en un espacio de nombres por tipo



RED HAT OPENSIFT APPLICATION HEALTH CHECKS



COMPROBACIONES DE ESTADO DE APLICACIONES DE RED HAT OPENS SHIFT

- Una *comprobación de estado* o *sondeo* es una verificación periódica que monitorea el estado de una aplicación.
- Existen tres tipos de sondeos en OpenShift:
 - **Sondeo de inicio:** Un sondeo de inicio verifica si se inicia la aplicación dentro de un contenedor. OpenShift finaliza el contenedor y lo reinicia según la `restartPolicy` del pod
 - **Sondeo de disponibilidad:** Los sondeos de disponibilidad determinan si un contenedor está listo o no para responder las solicitudes. Para configurar el sondeo de disponibilidad, debe agregar el atributo `spec.containers.readinessprobe` de la configuración del pod.
 - **Sondeo de ejecución:** Los sondeos de ejecución determinan si una aplicación que se está ejecutando en un contenedor tiene un estado `healthy`. Para configurar el sondeo de ejecución, debe agregar el atributo `spec.containers.livenessprobe` de la configuración del pod.



CONFIGURACION DE SONDEOS

OpenShift ofrece las siguientes opciones para configurar estos sondeos:

Nombre	Obligatorio	Descripción	Valor predeterminado
initialDelaySeconds	Sí	Determina cuánto tiempo se debe esperar después de que se inicia el contenedor y antes de comenzar el sondeo.	0
timeoutSeconds	Sí	Determina cuánto tiempo se debe esperar para que finalice el sondeo. Si se excede este tiempo, OpenShift supone que el sondeo falló.	1
periodSeconds	No	Especifica la frecuencia de las comprobaciones.	1
successThreshold	No	Especifica los mínimos aciertos consecutivos del sondeo para que se considere satisfactorio después de haber fallado.	1
failureThreshold	No	Especifica los mínimos errores consecutivos del sondeo para que se considere con errores después de haber tenido un resultado satisfactorio.	3

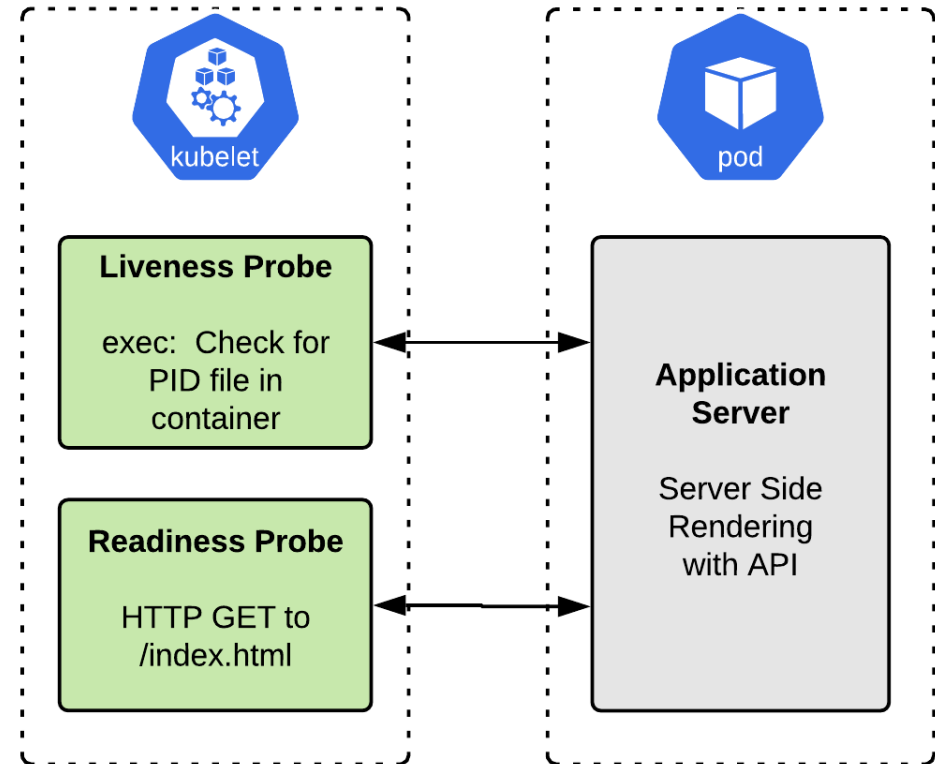


MÉTODOS PARA COMPROBAR EL ESTADO DE LA APLICACIÓN



COMPROBACIONES HTTP

- Una comprobación HTTP es ideal para aplicaciones que muestran códigos de estado HTTP, como las API REST.
- El sondeo HTTP usa solicitudes GET para garantizar el estado de una aplicación. La comprobación se considera satisfactoria si el código de respuesta HTTP es entre 200 y 399.

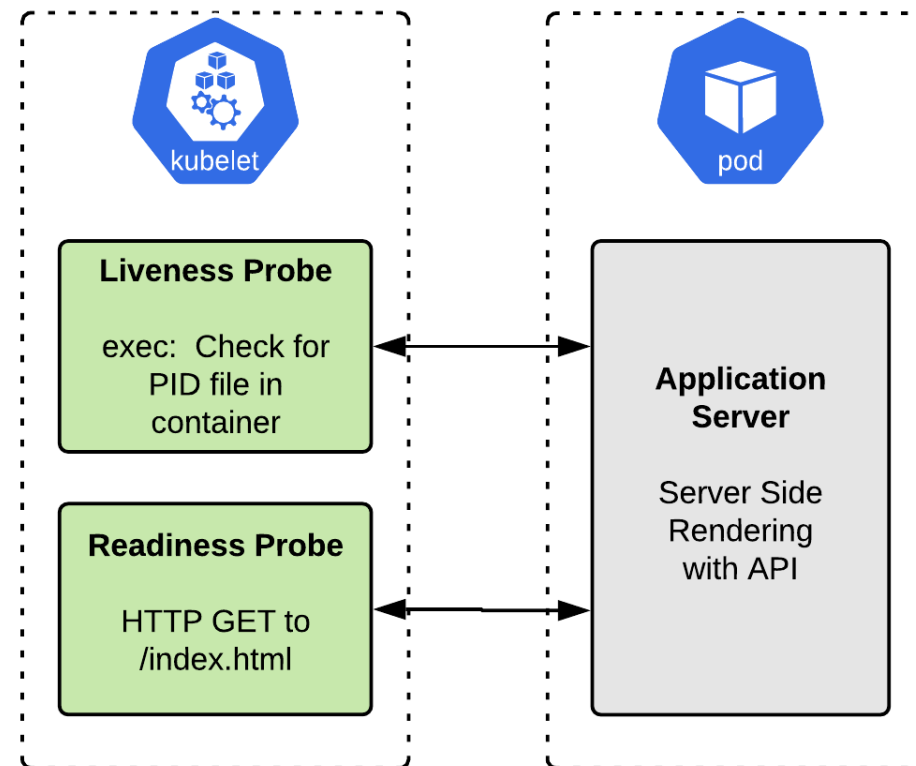


```
...contents omitted...
readinessProbe:
  httpGet:
    path: /health
    port: 8080
    initialDelaySeconds: 15
    timeoutSeconds: 1
...contents omitted...
```

- 1 El extremo del sondeo de disponibilidad.
- 2 Cuánto tiempo se debe esperar después de que se inicia el contenedor antes de comprobar su estado.
- 3 Cuánto tiempo se debe esperar para que finalice el sondeo.

COMPROBACIONES DE EJECUCIÓN DE CONTENEDORES

- Las comprobaciones de ejecución del contenedor son ideales en escenarios en los que debe determinar el estado del contenedor según el código de salida de un proceso o script de shell que se ejecuta en el contenedor.
- La comprobación tiene éxito si el contenedor devuelve el código de salida 0.

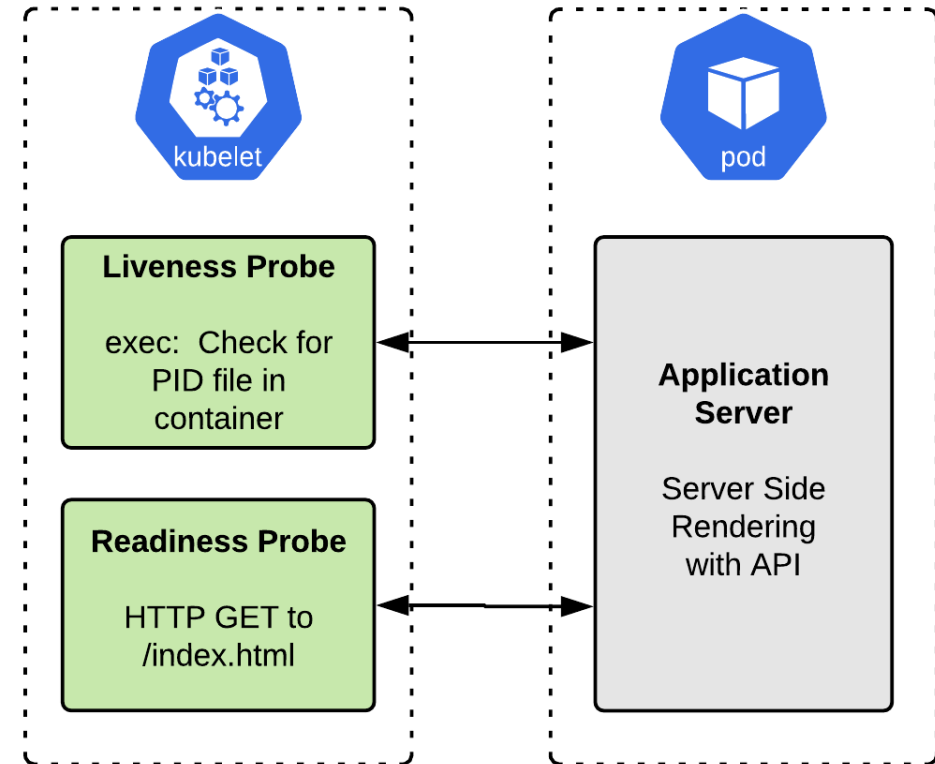


```
...contents omitted...
livenessProbe:
  exec:
    command: ❶
    - cat
    - /tmp/health
  initialDelaySeconds: 15
  timeoutSeconds: 1
...contents omitted...
```

❶ El comando a ejecutar y sus argumentos, como una matriz YAML.

COMPROBACIONES DE SOCKETS TCP

- Una comprobación de sockets TCP es ideal para aplicaciones que abren puertos TCP, como servidores de bases de datos, servidores de archivos, servidores web y servidores de aplicaciones.
- Cuando se usan las comprobaciones de sockets TCP, OpenShift intenta abrir un socket en el contenedor.
- Se considera que el contenedor tiene un estado correcto si la comprobación puede establecer una conexión exitosa.

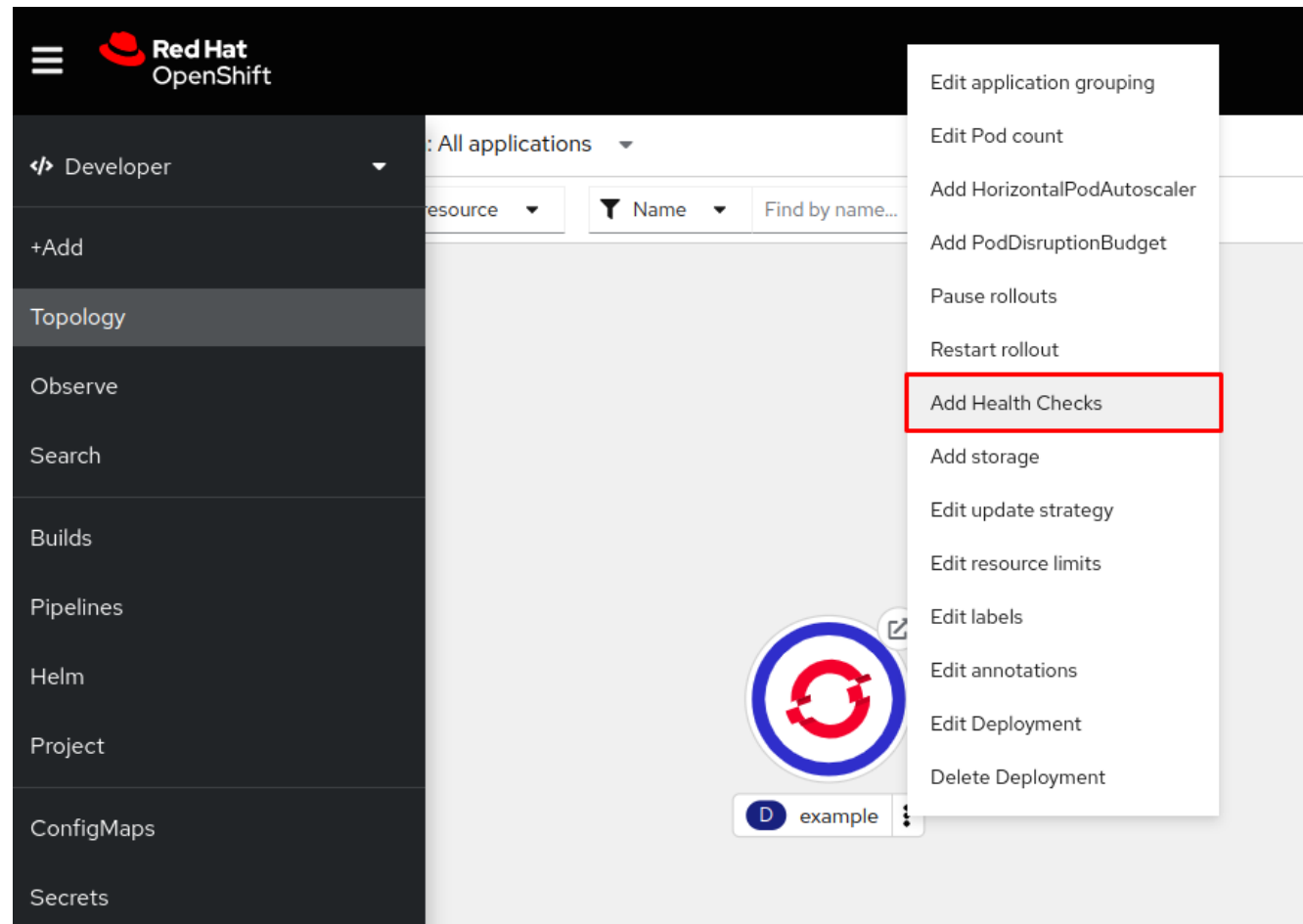


```
...contents omitted...  
livenessProbe:  
  tcpSocket:  
    port: 80801  
  initialDelaySeconds: 15  
  timeoutSeconds: 1  
...contents omitted...
```

¹ El puerto TCP que se comprobará.

GESTIÓN DE SONDEOS CON LA CONSOLA WEB

- Los desarrolladores pueden crear sondeos mediante la consola web de OpenShift. En la vista **Topology**, puede hacer clic en el menú de tres puntos para una implementación y, a continuación, hacer clic en **Add Health Checks**.



GESTIÓN DE SONDEOS CON LA CONSOLA WEB II

- Esto abre el formulario **Add health checks** donde puede agregar los sondeos para su implementación.

Project: example ▼

Path

Port *

Failure threshold

How many times the probe will try starting or restarting before giving up.

Success threshold

How many consecutive successes for the probe to be considered successful after having failed.

Initial delay

seconds

How long to wait after the Container starts before checking it's health.

Period

seconds

How often to perform the probe.

Timeout

seconds

How long to wait for the probe to finish, if the time is exceeded, the probe is considered failed.



GESTIÓN DE SONDEOS CON LA CONSOLA WEB III

- En este punto, puede agregar más sondeos antes de hacer clic en **Add**.

Add health checks [Learn more](#)

Health checks for **D** example

Container **C** example

Readiness probe [Edit Probe](#)

A readiness probe checks if the Container is ready to handle requests. A failed readiness probe means that a Container should not receive any traffic from a proxy, even if it's running.

✓ Readiness probe added -

Liveness probe

A liveness probe checks if the Container is still running. If the liveness probe fails the Container is killed.

+ Add Liveness probe

Startup probe

A startup probe checks if the application within the Container is started. If the startup probe fails the Container is killed.

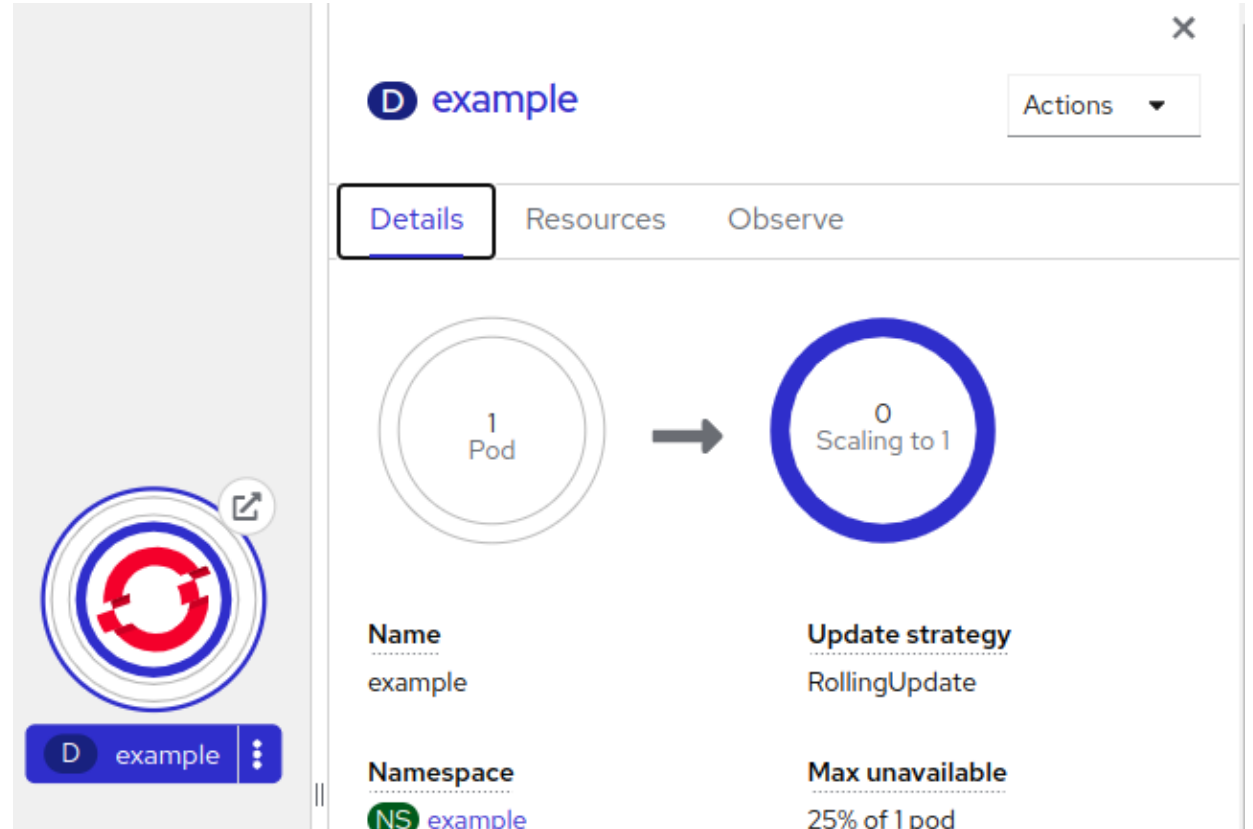
+ Add Startup probe

Add

Cancel

GESTIÓN DE SONDEOS CON LA CONSOLA WEB IV

- Al hacer clic en **Add**, OpenShift implementa los cambios.
- Puede seguir la nueva implementación en la pestaña **Details**.



The screenshot displays the OpenShift console interface for a deployment named 'example'. On the left, a sidebar shows a circular icon with a red and blue design, representing the deployment, with a button labeled 'D example' below it. The main panel has a tabbed interface with 'Details', 'Resources', and 'Observe'. The 'Details' tab is active, showing a diagram of the deployment process: a circle labeled '1 Pod' transitions via an arrow to a circle labeled '0 Scaling to 1'. Below this, the 'Name' is 'example' and the 'Namespace' is 'NS example'. The 'Update strategy' is 'RollingUpdate' and the 'Max unavailable' is '25% of 1 pod'. An 'Actions' dropdown menu is visible in the top right corner.

D example

Actions

Details Resources Observe

1 Pod → 0 Scaling to 1

Name
example

Update strategy
RollingUpdate

Namespace
NS example

Max unavailable
25% of 1 pod

CREACIÓN DE SONDEOS CON LA CLI

En los siguientes ejemplos, se muestra el uso del comando `oc set probe` con diversas opciones:

```
[user@host ~]$ oc set probe deployment myapp --readiness \
--get-url=http://:8080/readyz --period=20
```

```
[user@host ~]$ oc set probe deployment myapp --liveness \
--open-tcp=3306 --period=20 \
--timeout-seconds=1
```

```
[user@host ~]$ oc set probe deployment myapp --liveness \
--get-url=http://:8080/livez --initial-delay-seconds=30 \
--success-threshold=1 --failure-threshold=3
```



LAB 10

GESTIÓN DE IMPLEMENTACIONES DE RED HAT OPENSIFT

