

# MÓDULO 10

---

Gestión de implementacion 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...
```

- <sup>1</sup> Cantidad de réplicas de pod cuando se completa la implementación.
- <sup>2</sup> La estrategia para la implementación. Puede ser de los tipos RollingUpdate o Recreate.
- <sup>3</sup> La configuración rollingUpdate controla cómo los nuevos pods reemplazan a los pods de la versión anterior.
- <sup>4</sup> 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



<b>Name</b> dc-metro-map	<b>Latest version</b> 1
<b>Namespace</b> <b>NS</b> demo-49	<b>Message</b> config change
<b>Labels</b> <a href="#">Edit</a> 	<b>Update strategy</b> 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>	<b>Max unavailable</b> 25% of 1 pod
	<b>Max surge</b> 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.

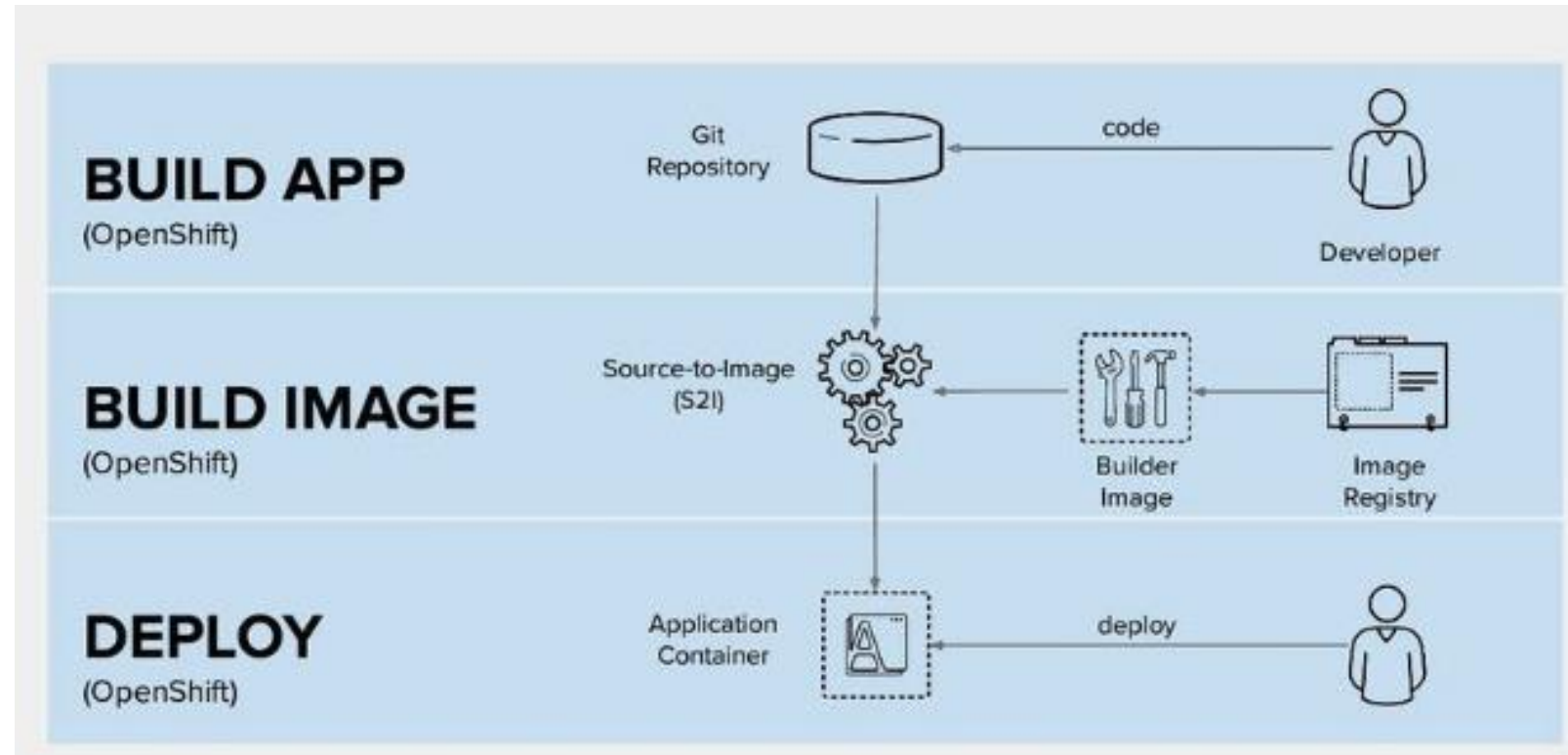


# PROCESO DE DEPLOYMENT

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
```

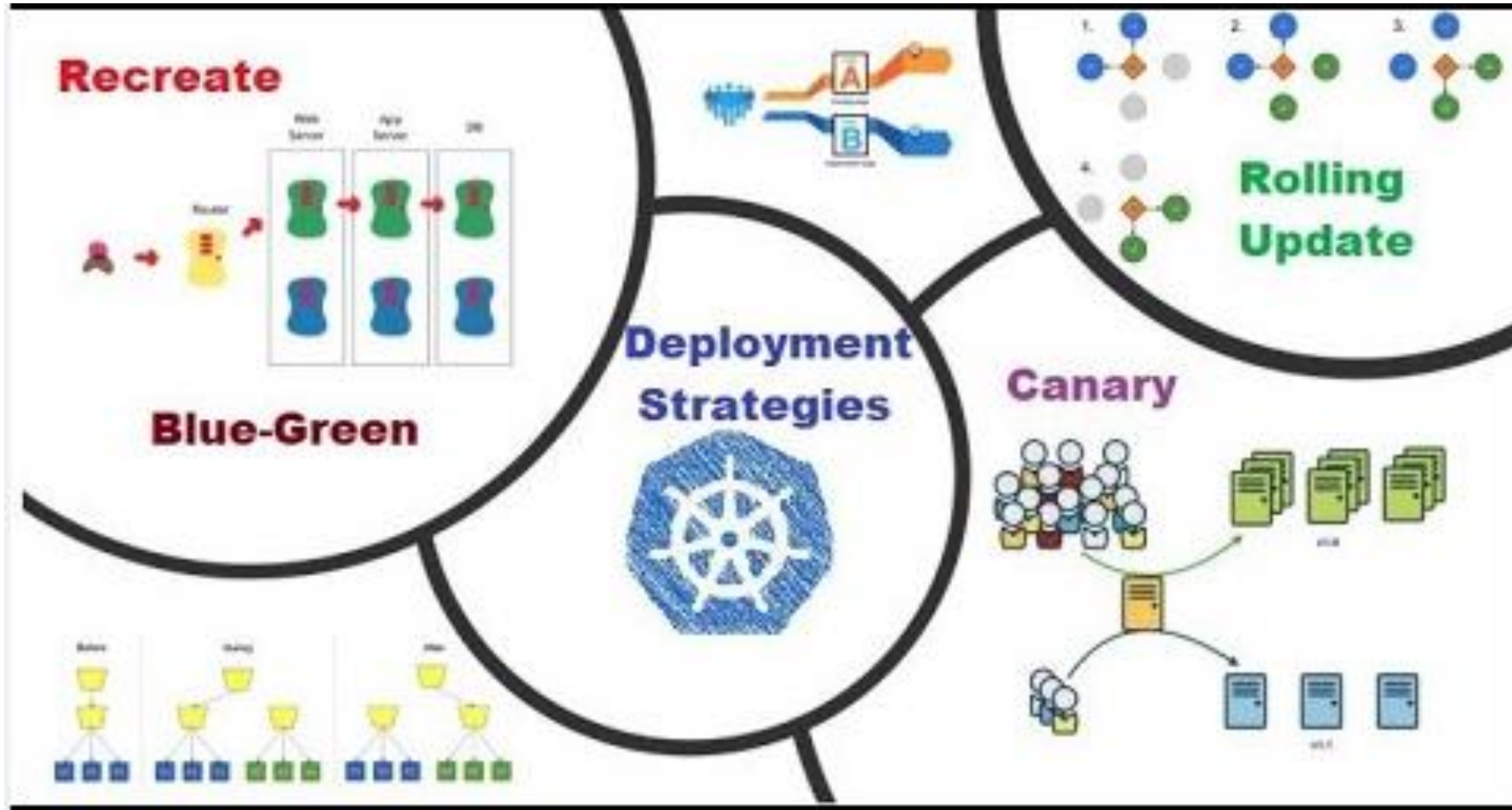
- ❶ El atributo `pre` define un enlace previo al ciclo de vida.
- ❷ Este enlace cancela nuevas implementaciones si falla el comando del enlace.
- ❸ El contenedor de la definición de plantilla de pod que crea el enlace.
- ❹ El comando que se ejecuta dentro del contenedor `schema-validator`.

# ESTRATEGIAS DE IMPLEMENTACIÓN

---



# ESTRATEGIAS DE IMPLEMENTACIÓN



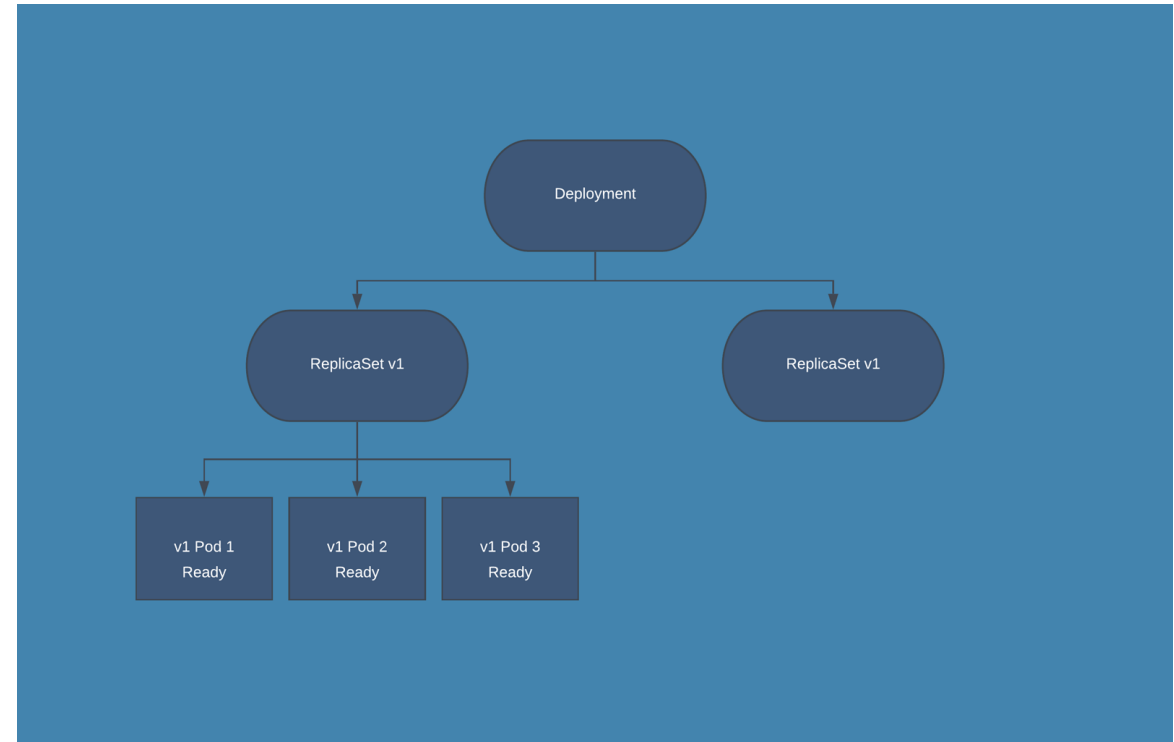
# ESTRATEGIAS DE IMPLEMENTACIÓN CON RECURSOS DE IMPLEMENTACIÓN: ROLLING UPDATE

- **Rolling Strategy:**

- Las implementaciones graduales en OpenShift son de tipo canary.
- OpenShift prueba una nueva versión, canary, antes de reemplazar todas las instancias anteriores.

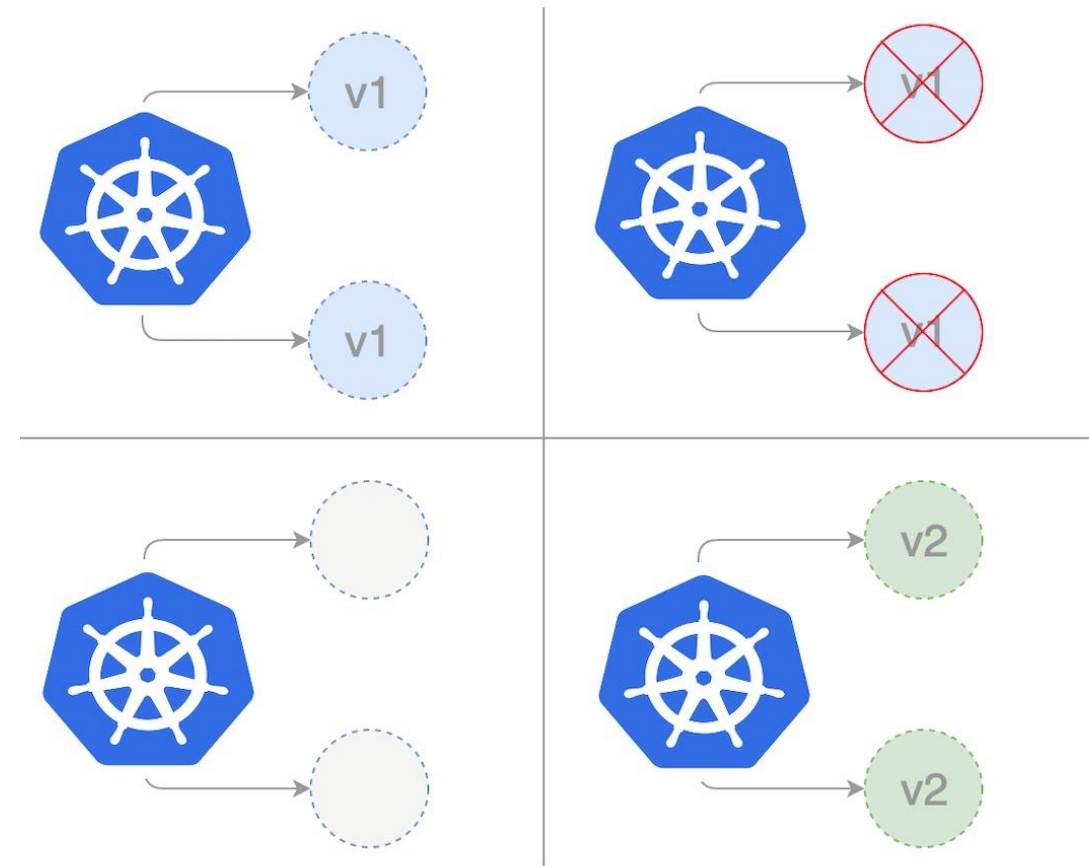
- **Recreate:**

- 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.



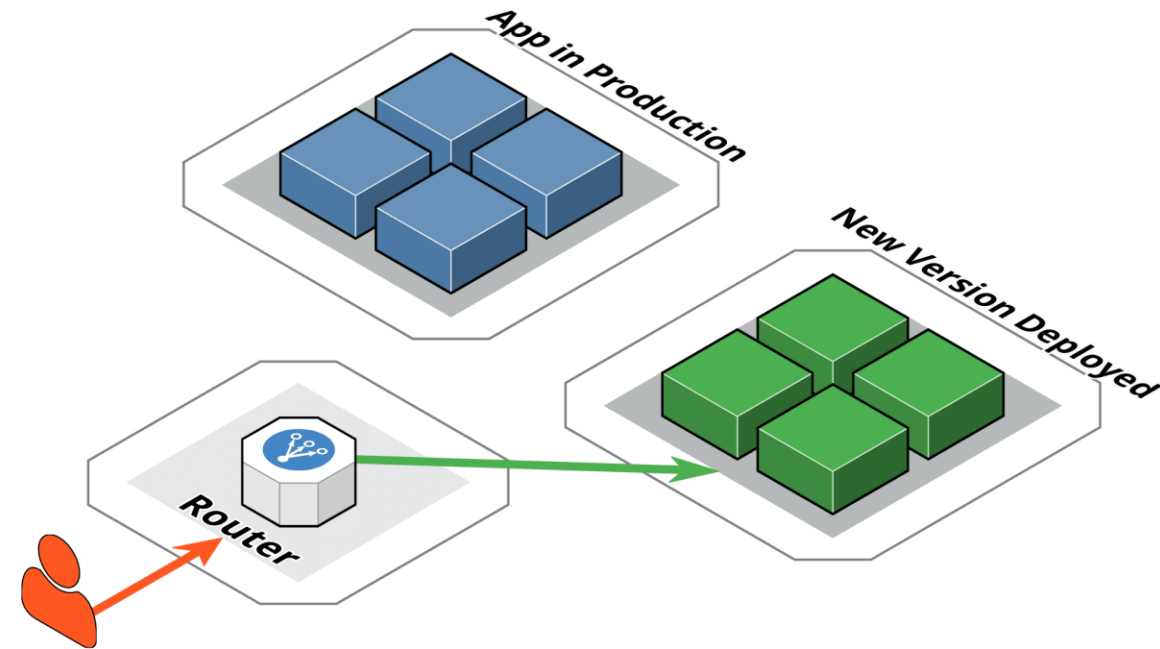
# ESTRATEGIAS DE IMPLEMENTACIÓN CON RECURSOS DE IMPLEMENTACIÓN: RECREATE

- **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.



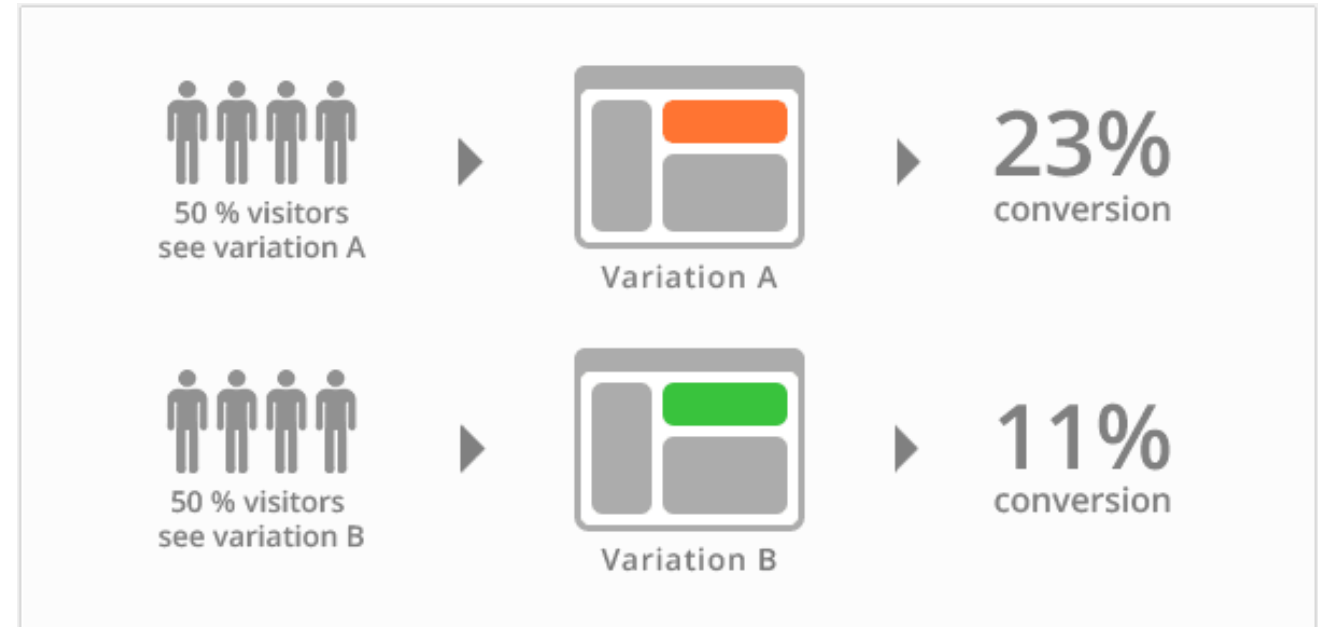
# ESTRATEGIAS DE IMPLEMENTACIÓN CON EL ENRUTADOR DE RED HAT OPENSIFT: DEPLOYMENT BLUE/GREEN

- 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

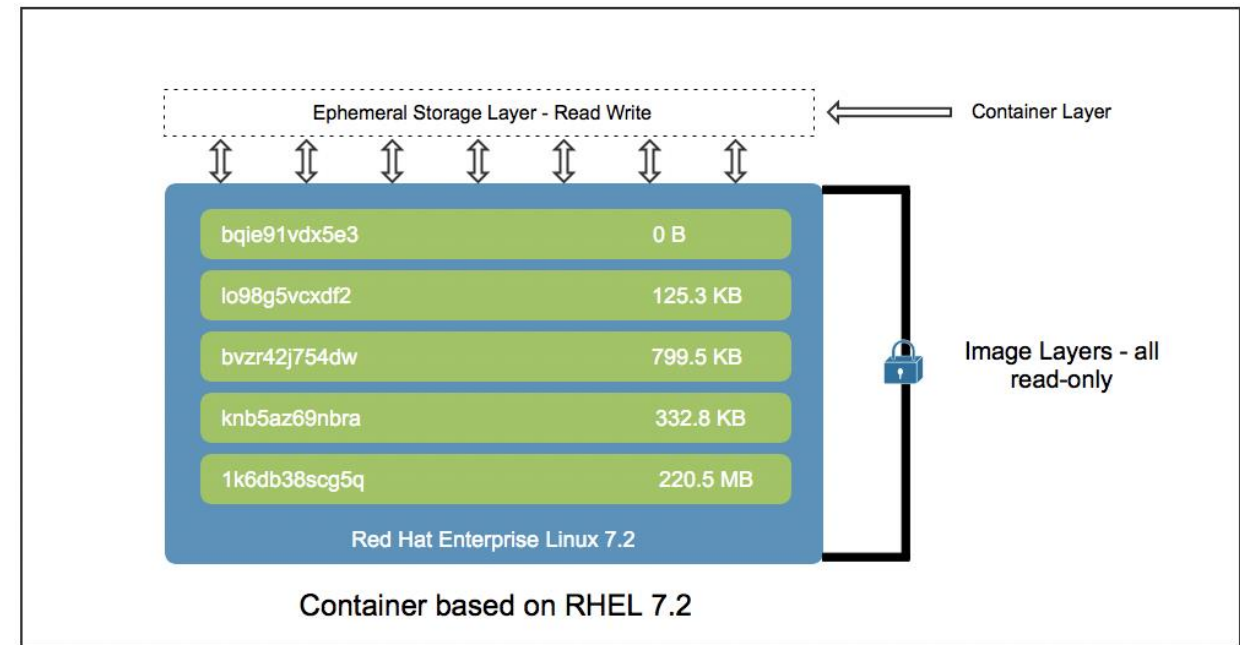
---



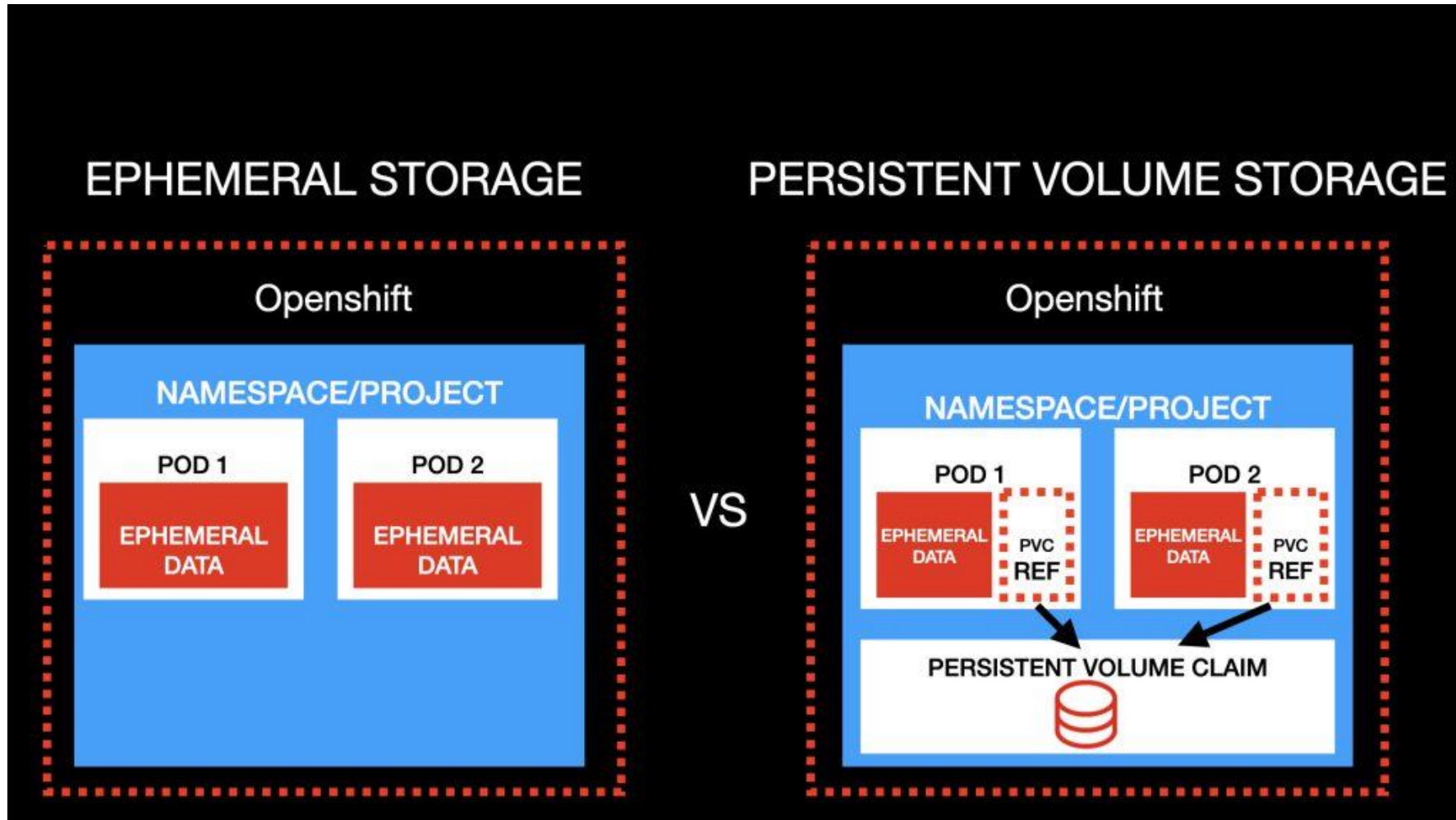


# UBICACIONES DE ALMACENAMIENTO PERSISTENTE

- 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.

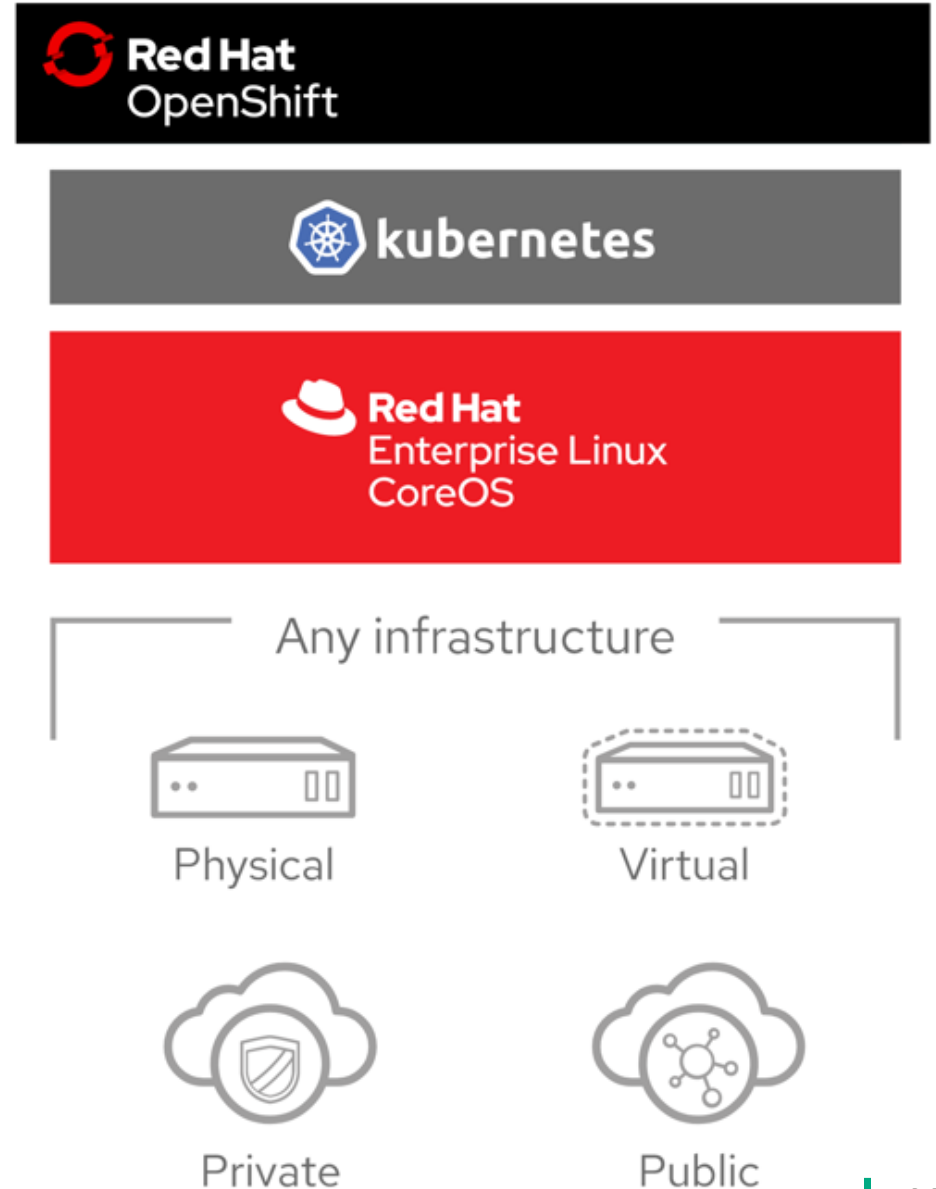


# EPHEMERAL VS PERSISTENT VOLUME



# 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.



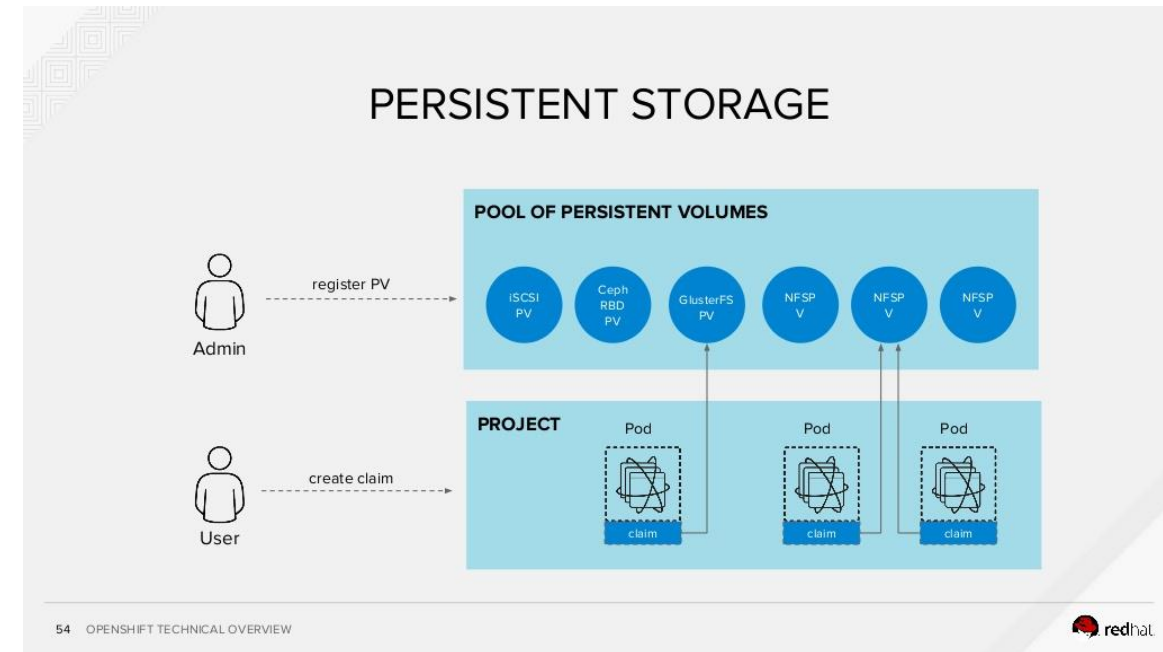
# COMPONENTES DE PERSISTENT STORAGE

## - 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.

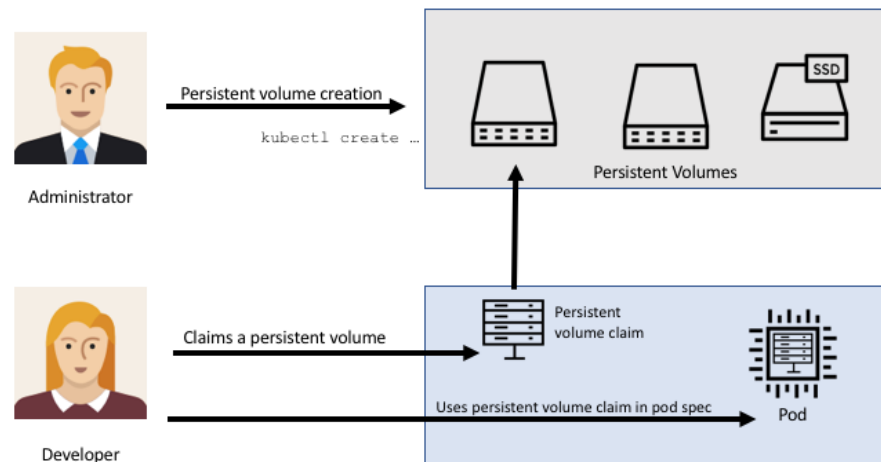
## - Persistent Volume Claim :

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.



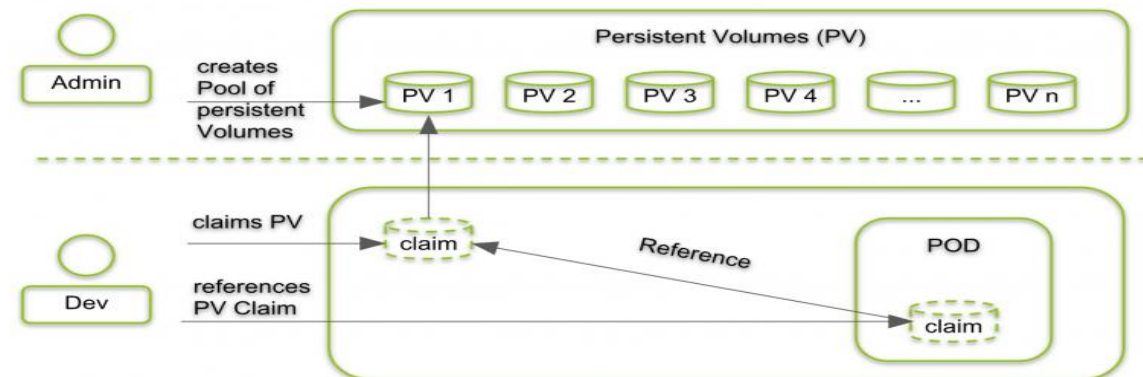
# ALMACENAMIENTO DE PERSISTENT VOLUME CLAIMS

- 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

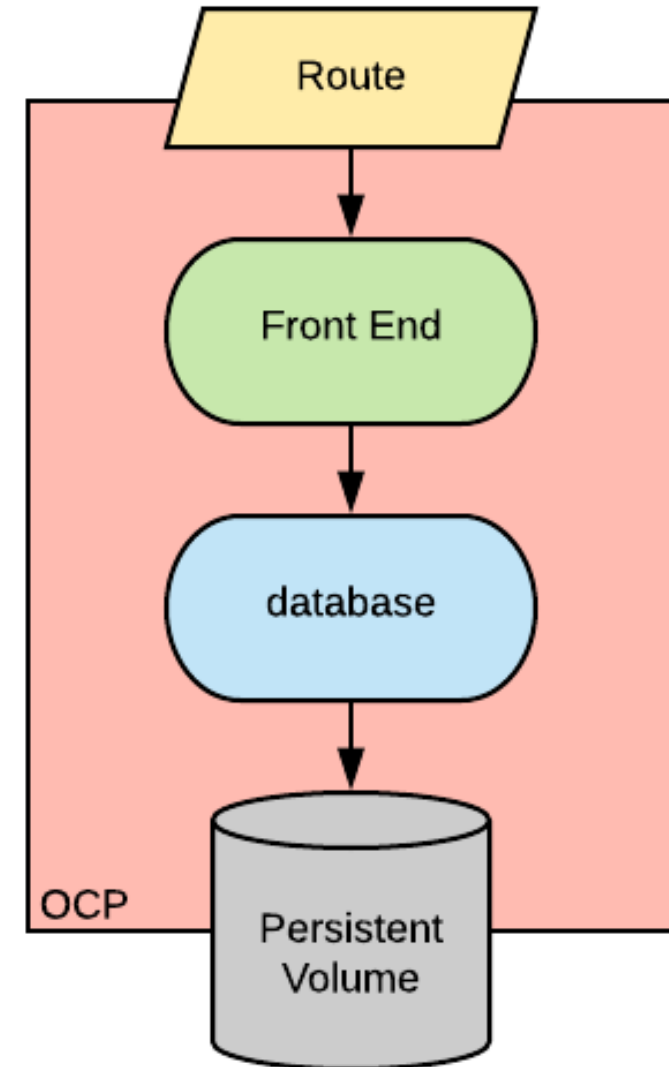
---

\* 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.

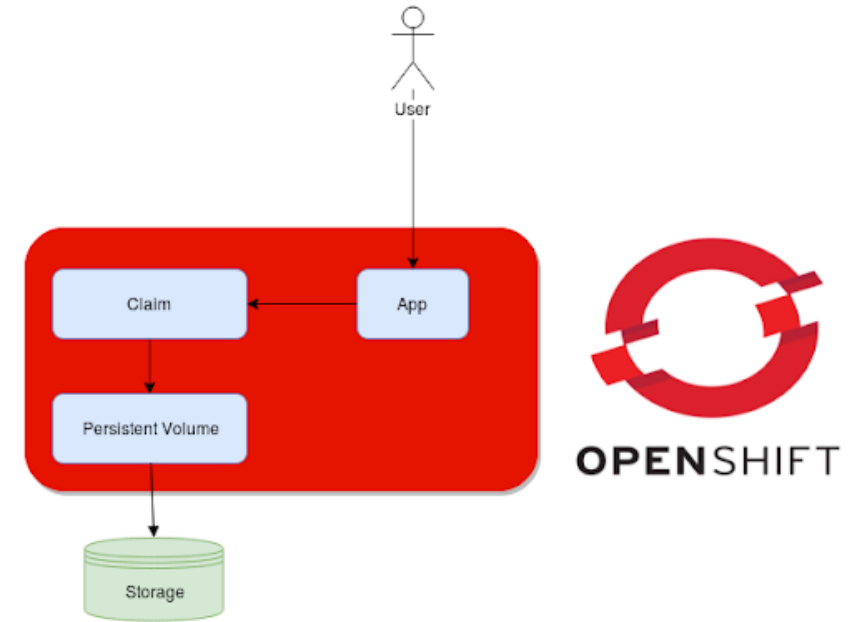
- Clases de almacenamiento de volumen persistente **PV Claims** puede solicitar opcionalmente una clase de almacenamiento específica especificando su nombre en el Atributo **storageClassName**.



# 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.



LAB 10

# GESTIÓN DE IMPLEMENTACIONES DE RED HAT OPENSIFT

