Persistent Data and Storage

Persistent Data and Storage

01 Kubernetes storage abstractions

02 StatefulSets

03 Configmaps

04 Secrets

Persistent Data and Storage

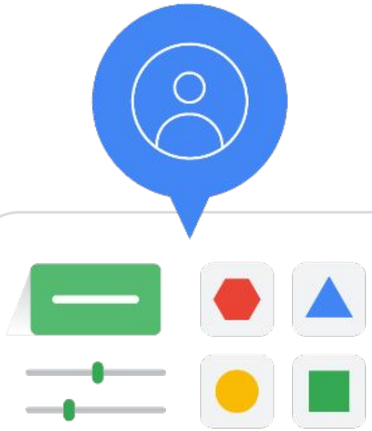
01 Kubernetes storage abstractions

02 StatefulSets

03 Configmaps

04 Secrets

Storage abstractions helps simplify provisioning and storage management



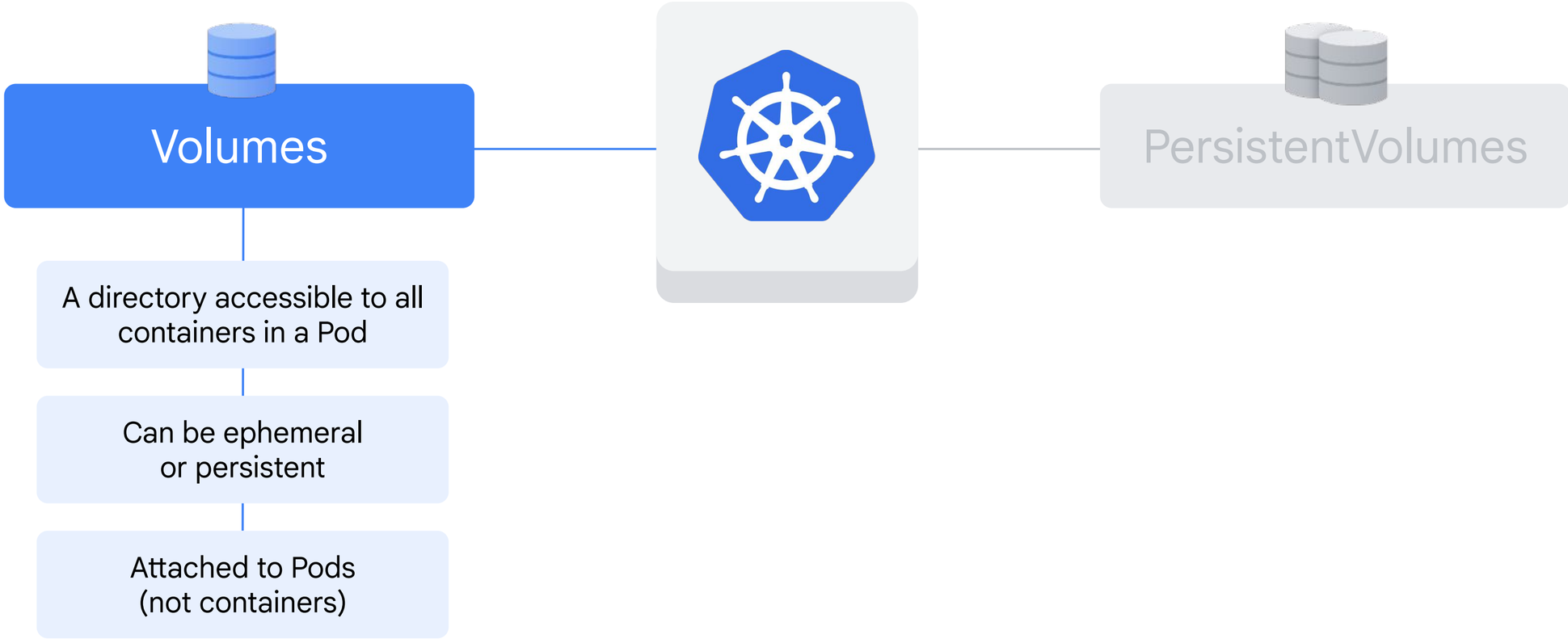
A layer of software that allows user to interact with storage in a consistent way.

GKE

Provides a consistent interface for accessing storage, and it lets users work with different storage providers.

Standard Kubernetes storage abstractions

Standard Kubernetes storage abstractions



Ephemeral Volumes

01

emptyDir

02

DownwardAPI

03

ConfigMap

04

Secret

emptyDir

01

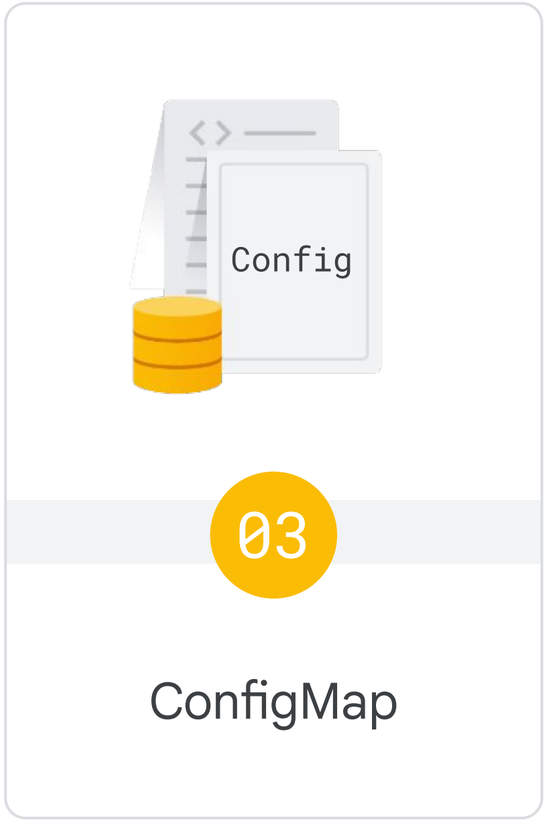
emptyDir

- It creates an empty directory within the Pod's filesystem.
- It will exist as long as that Pod is running on that node.
- It is commonly used for storing temporary files or data that doesn't need to persist.
- When a Pod is removed, the data is permanently deleted.
- If a container crashes, the Pod will not be removed from a node.

DownwardAPI

- It can be used to make data available to applications.
- It’s useful for configuring applications based on their deployment context.
- It’s a way for containers to learn about their Pod.

ConfigMap



- It can be used to inject configuration data into the Pod's environment.
- ConfigMap data is more structured.
- The data can be shared across multiple Pods.
- The data can be referenced in a volume, and applications can then consume the data.
- If a container crashes, the Pod will not be removed from a node.

Secret

04

Secret

- It is specifically designed for storing sensitive data.
- Google encrypts the data at rest and ensures secure access within the Pod.
- They are never written to non-volatile storage.

Managing non-sensitive and sensitive Pod configuration data

ConfigMap

Manages **non-sensitive** string data

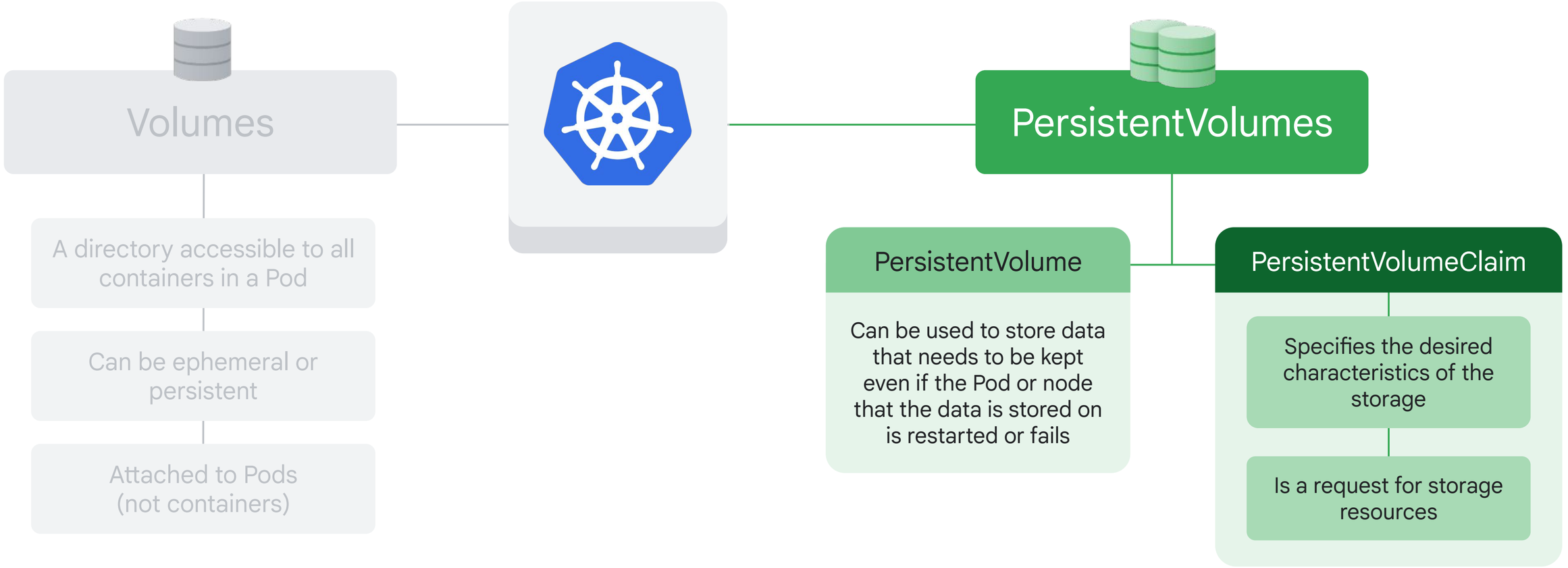
Pod

Volume types that are coupled to the life of the Pod. They get deleted when the Pod ceases to exist.

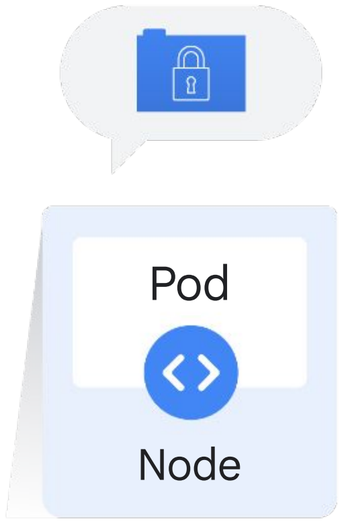
Secret

Manages **sensitive** encrypted data

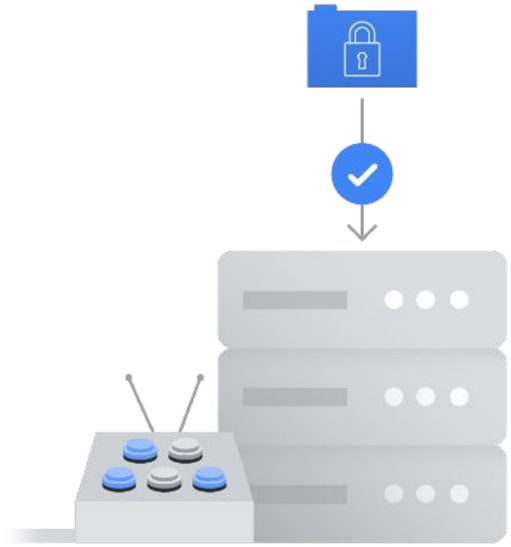
Durable Volume types



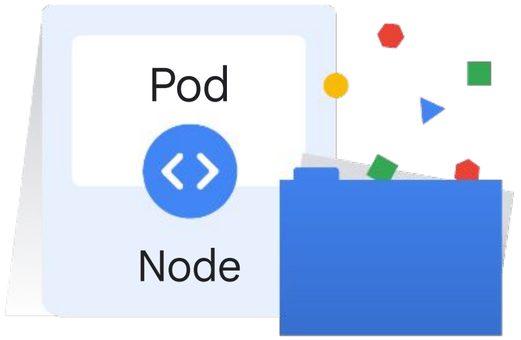
Matching a PVC to a PersistentVolume



A Pod requests a PersistentVolumeClaim



The Kubernetes controller matches the PersistentVolumeClaim to an available PersistentVolume.



The Pod mounts the PersistentVolume and can access the storage.

The benefit of PersistentVolumes

Application developers can claim and use provisioned storage using **PersistentVolumeClaims**

- No need to create and maintain storage volumes directly.
- Allows for the separation of roles:
  - Job of administrators to make persistent volumes available.
  - Job of developers to use those volumes in applications.

How to create a PersistentVolume manifest

```
apiVersion: v1
kind: PersistentVolume
metadata:
   name: pd-volume
spec:
   storageClassName: "standard"
   capacity:
      storage: 100G
   accessModes:
   - ReadWriteOnce:
   gcePersistentDisk:
      pdName: demo-disk
      fsType: ext4
```

1. Specify the StorageClassName, which is a resource used to implement PersistentVolumes.
2. Decide if you want to use the Compute Engine Standard Persistent Disk type. The persistent disk must be created first, or you will encounter an error.
3. Specify volume capacity, which is the storage capacity required for your PersistentVolume.

Using an SSD persistent disk

To use an SSD persistent disk, create a new StorageClass, and give it an appropriate name, such as “ssd.”

```
apiVersion: v1
kind: PersistentVolume
metadata:
   name: pd-volume
spec:
   storageClassName: "ssd"
   capacity:
      storage: 100G
   accessModes:
   - ReadWriteOnce:
   gcePersistentDisk:
      pdName: demo-disk
      fsType: ext4
```

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: ssd
provisioner:
kubernetes.io/gce-pd
parameters:
  type: pd-ssd
```

A PVC that uses this new StorageClass named “ssd” will only use a PV that also has a StorageClass named “ssd.”

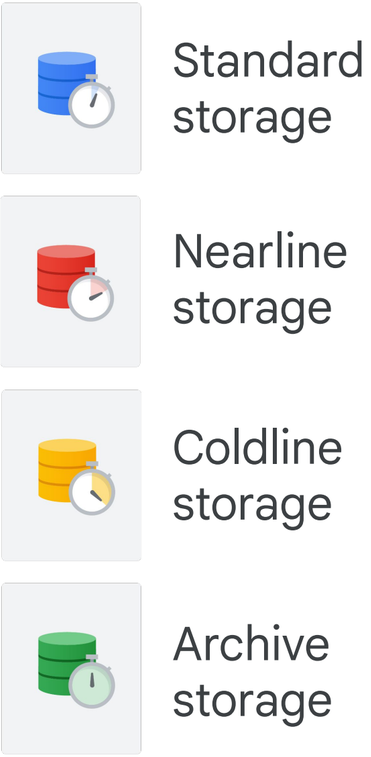
Kubernetes StorageClass versus Google Cloud Storage classes

Kubernetes StorageClasses

Choices for how PersistentVolumes are backed.

Cloud Storage

Object storage for the web.



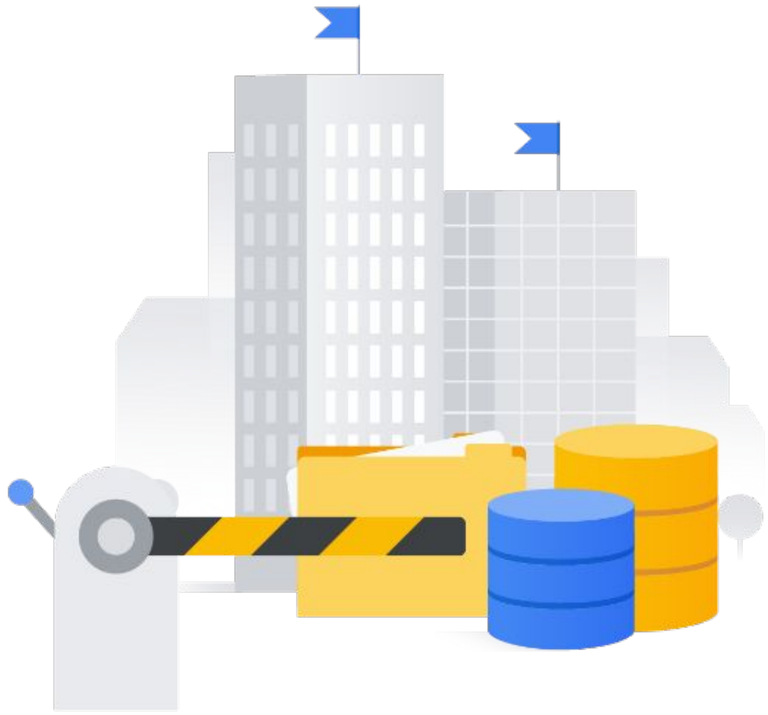
Standard storage

Nearline storage

Coldline storage

Archive storage

AccessModes

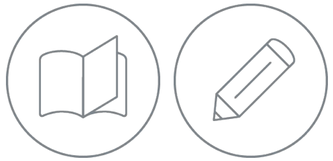


Read | Write | Once | Many

- Define how Pods can mount and access the storage provided by the PersistentVolume.
- Determine the level of access and the number of Pods that can simultaneously mount the PersistentVolume.

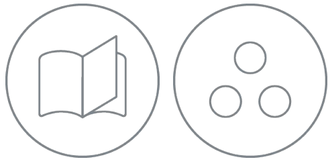
Types of AccessModes

ReadWriteOnce



The PersistentVolume can be mounted read-write by a single Node in the cluster.

ReadOnlyMany



The PersistentVolume can be mounted read-only by multiple nodes simultaneously.

ReadWriteMany

The PersistentVolume can be mounted read-write by multiple nodes simultaneously.

Creating a PV and PVC from a YAML manifest

When the Pod is started, GKE will look for a matching PV with the same storageClassName, accessModes, and sufficient capacity.

Pod requesting storage:

```
apiVersion: v1
kind: PersistentVolume
metadata:
   name: pd-volume
spec:
   storageClassName: "standard"
   capacity:
      storage: 100G
   accessModes:
   - ReadWriteOnce:
   gcePersistentDisk:
      pdName: demo-disk
      fsType: ext4
```

PVC specifying storage requirements:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
   name: pd-volume-claim
spec:
   storageClassName: “standard"
   accessModes:
   - ReadWriteOnce:
   resources:
      requests:
         storage: 100G
```

What happens when a PVC is added to a Pod

Pod requesting storage:

```
kind: Pod
apiVersion: v1
metadata:
  name: pod-demo
spec:
  volumes:
    - name: pvc-demo-vol
      persistentVolumeClaim:
        claimName: pd-volume-claim
  containers:
    - name: pod-demo
      image: nginx
```

PVC specifying storage requirements:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
   name: pd-volume-claim
spec:
   storageClassName: “standard"
   accessModes:
   - ReadWriteOnce:
   resources:
      requests:
         storage: 100G
```

Dynamic provisioning

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
   name: pd-volume-claim
spec:
   storageClassName: “standard"
   accessModes:
   - ReadWriteOnce:
   resources:
      requests:
         storage: 100G
```

If application developers claim more storage than has already been allocated to PersistentVolumes, Kubernetes will try to provision a new PV dynamically.

- Kubernetes will try to dynamically provision a PV if the PVC’s storageClassName is defined and an appropriate PV does not already exist.
- If a matching PV already exists, Kubernetes will bind it to the claim.
- If storageClassName is omitted, the PVC will use the default StorageClassName of “standard.”

How to retain the PersistentVolumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
   name: pd-volume-claim
spec:
   storageClassName: “standard"
   accessModes:
   - ReadWriteOnce:
   resources:
      requests:
         storage: 100G
   persistentVolumeReclaimPolicy: Retain
```

Deleting the PVC will also delete the provisioned PV.

To retain the PV, set its persistentVolumeReclaimPolicy to “Retain” in the YAML file.

Persistent Data and Storage

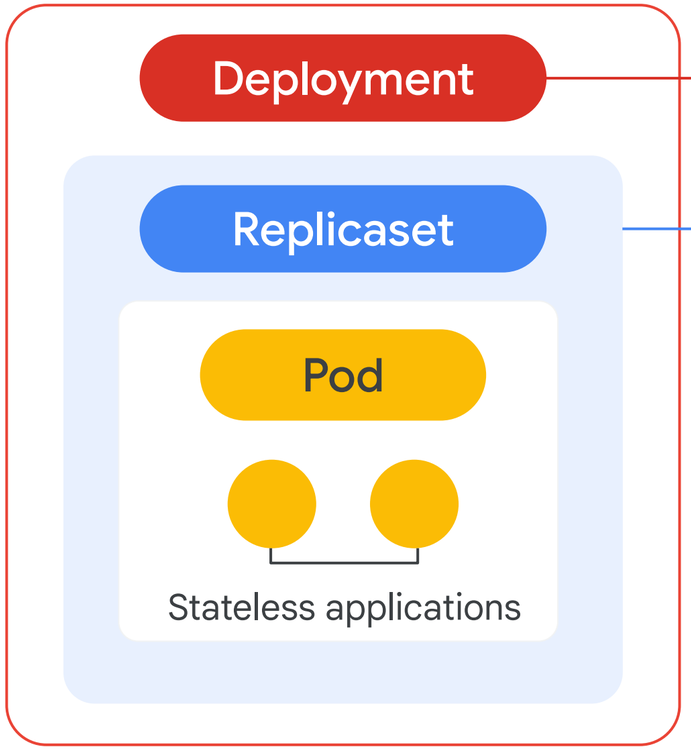
01 Kubernetes storage abstractions

02 StatefulSets

03 Configmaps

04 Secrets

PersistentVolumes can also be used for Deployments and StatefulSets



A Pod template used for stateless applications.

Runs and maintains a set of identical Pods, known as replicas.

Replicas need to attach and reattach to PersistentVolumes dynamically, which might cause a deadlock.

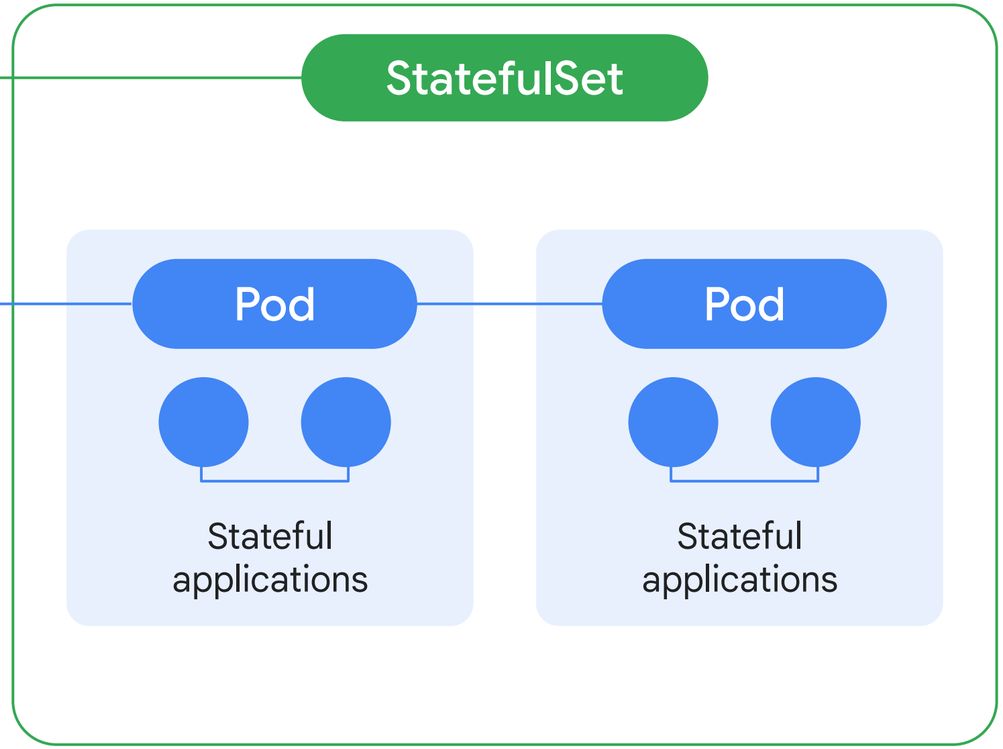
Maintain state in PersistentVolumes with a StatefulSet

Defines a desired state, and its controller achieves it.

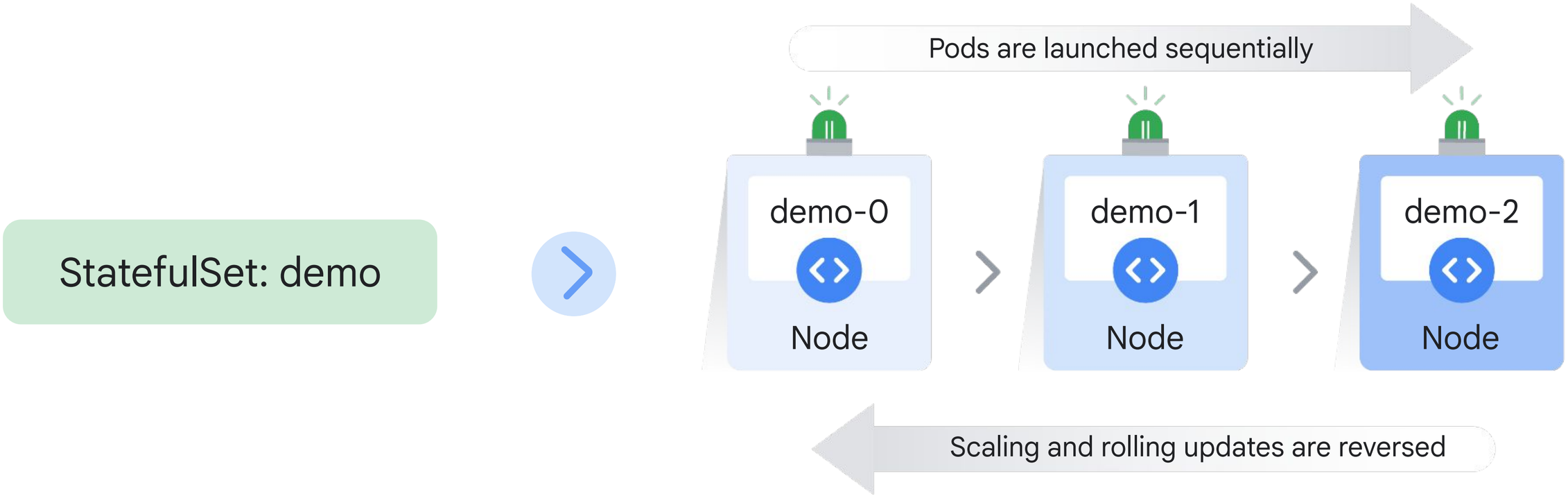
Maintains a persistent identity for each Pod.

Each Pod has an **ordinal index** (a unique sequential number) with:

- Relevant Pod name.
- A stable hostname.
- Stably identified persistent storage that is linked to the ordinal index.

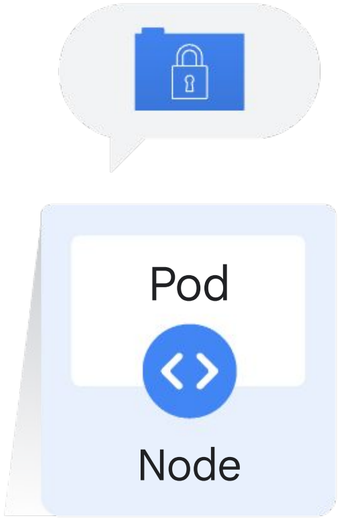


Deployment, scaling, and updates are ordered by using the ordinal index



StatefulSets

StatefulSets are useful for stateful applications.



With stable storage, StatefulSets use a unique PersistentVolumeClaim for each Pod.

These PersistentVolumeClaims use ReadWriteOnce access mode for applications.

StatefulSets require a service to control networking

```
apiVersion: v1
kind: Service
metadata:
  name: demo-service
  labels:
    app: demo
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: demo
```

If load-balancing and a single service IP are not needed, a headless service can be created by specifying “None” for the cluster IP in the Service definition.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: demo-statefulset
spec:
  selector:
    matchLabels:
      app: demo
  serviceName: demo-service
  replicas: 3
  updateStrategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: demo
```

To ensure a specific service gets used for a StatefulSet, add that service to the serviceName field.

A label selector is required for the Service

A label selector is required for the Service, and this must match the template's labels defined in the template section of the StatefulSet definition.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: demo-statefulset
spec:
  selector:
    matchLabels:
      app: demo
  serviceName: demo-service
  replicas: 3
  updateStrategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: demo
    […]
      spec:
        containers:
        - name: demo-container
          image: k8s.gcr.io/demo:0.1
            […]
        volumeClaimTemplates:
        - metadata:
          name: demo-pvc
          spec:
            accessModes:
          ["ReadWriteOnce" ]
            resources:
              requests:
               storage: 1Gi
```

Container details must be defined

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: demo-statefulset
spec:
  selector:
    matchLabels:
      app: demo
  serviceName: demo-service
  replicas: 3
  updateStrategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: demo
```

```
[…]
    spec:
      containers:
    - name: demo-container
      image: k8s.gcr.io/demo:0.1
      ports:
      - containerPort: 80
        name: web
      volumeMounts:
      - name: www
        mountPath: /usr/share/web
    […]
    volumeClaimTemplates:
    - metadata:
```

The container details must also be defined, including the image, containerPort for the Service, and Volume mounts.

Specify VolumeClaimTemplates under the template

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: demo-statefulset
spec:
  selector:
    matchLabels:
      app: demo
  serviceName: demo-service
  replicas: 3
  updateStrategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: demo
[…]
    spec:
       containers:
    - name: demo-container
       image: k8s.gcr.io/demo:0.1
          […]
    volumeClaimTemplates:
    - metadata:
       name: demo-pvc
    spec:
          accessModes:
["ReadWriteOnce"]
       resources:
         requests:
          storage: 1Gi
```

The VolumeClaimTemplate must be named, and the spec needs to be the same as the PersistentVolumeClaim that is required by the Pods in this StatefulSet.

Persistent Data and Storage

01 Kubernetes storage abstractions

02 StatefulSets

03 Configmaps

04 Secrets

A ConfigMap stores configuration data

A ConfigMap in Kubernetes is an API object that stores configuration data as key-value pairs.

It provides a mechanism to decouple application configuration from Pods.

It stores and maintains a Pod's specifications in one place.

A ConfigMap can be used to store:

- Config files
- Command-line arguments
- Environment variables
- Port numbers

Creating a ConfigMap using literal values

From-literal syntax

```
Command: $ kubectl create
configmap demo \
--from-literal=
lab.difficulty=easy \
--from-literal=
lab.resolution=high
```

Google Cloud console

demo

Cluster	projectdemo
Namespace	default
Created	Mar 22, 2024, 4:43:27 PM
Labels	*No labels set*
Annotations	*Not set*

Data

Lab.difficulty	easy
Lab.resolution	high

kubectl

```
$ kubectl describe configMaps
demo
Name:         demo
Namespace:    default
...
Data
====
lab.difficulty:
----
easy
lab.resolution:
----
high
```

Creating a ConfigMap using files

from-file syntax

```
Command: $ kubectl create
configmap demo

 --from-file=
demo/color.properties \

--from-file=
demo/ui.properties
```

Google Cloud console

demo

Cluster	projectdemo
Namespace	default
Created	Mar 22, 2024, 4:43:27 PM
Labels	*No labels set*
Annotations	*Not set*

Data

color.properties	color.good=green color.bad=red
ui.properties	resolution=high AAO=enabled

kubectl

```
$ kubectl describe configMaps
demo
Name:         demo
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
color.properties: |-
  color.good=green
  color.bad=red
ui.properties: |-
  resolution=high
  AAO=enabled
```

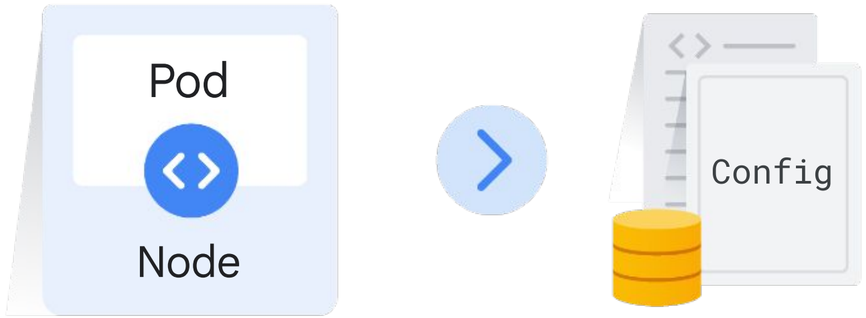
Creating ConfigMap from a manifest

Command: `$ kubectl apply`

```
apiVersion: v1
data:
  color.properties: |-
    color.good=green
    color.bad=red
  ui.properties: |-
    resolution=high
    AAO=enabled
kind: ConfigMap
metadata:
  name: demo
```

The `kubectl apply` command can be added to the manifest file and it will create the ConfigMap.

Pods refer to ConfigMaps in three ways



01 As a container environment variable

02 In Pod commands

03 By creating a Volume

Using a ConfigMap as a container environment variable

```
apiVersion: v1
kind: Pod
metadata:
   name: demo-pod
spec:
   containers:
   - name: test-container
     image:/busybox
     env:
      - name: VARIABLE_DEMO
        valueFrom:
          configMapKeyRef:
             name: demo
             key:lab.difficulty
```

demo

Cluster	projectdemo
Namespace	default
Created	Mar 22, 2024, 4:43:27 PM
Labels	*No labels set*
Annotations	*Not set*

Data

lab.difficulty	easy
lab.resolution	high

Container environment variables can be used inside manifest commands

```
apiVersion: v1
kind: Pod
metadata:
  name: demo-pod
spec:
  containers:
    - name: demo-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/sh", "-c", "echo $(VARIABLE_DEMO)"]
      env:
      - name: VARIABLE_DEMO
        valueFrom:
          configMapKeyRef:
             name: demo
             key: lab.difficulty
```

A dollar sign $ and an opening parenthesis (is put in front of the environment variable's name, and a closing parenthesis) after it.

This allows configuration artifacts to be decoupled from image content to keep containerized applications portable.

Add ConfigMap data to an ephemeral volume

```
[…]
Kind: Pod
spec:
  containers:
    - name: demo-container
      image: k8s.gcr.io/busybox
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: demo
```

All the data from the ConfigMap is stored in this ConfigMap Volume as files, and then this Volume is mounted to the container by using the mountPath directory.

When a ConfigMap Volume is already mounted and the source ConfigMap is changed, the projected keys are eventually updated.

Persistent Data and Storage

01 Kubernetes storage abstractions

02 StatefulSets

03 Configmaps

04 Secrets

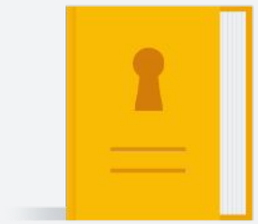
Secrets store sensitive information

Secret

Passes information to Pods.

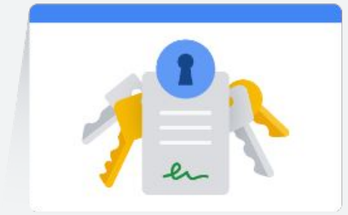
- They are used to store sensitive information like passwords, tokens, and SSH keys.
- They help to ensure Kubernetes doesn't accidentally output this data to logs.
- If an application is managing high-value assets, key management systems should be used.

There are three types of Secrets



Generic

Used for creating Secrets from files, directories, or literal values.



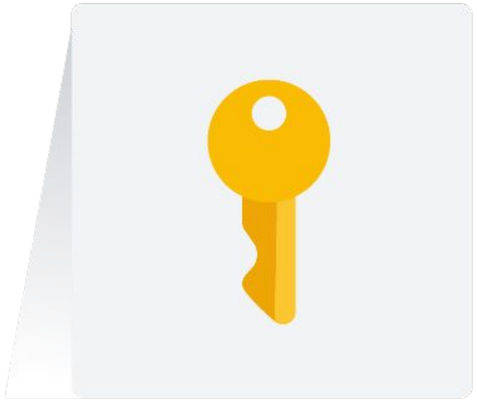
TLS

Designed to securely store transport layer security (TLS) certificates and their associated private keys.

Docker-Registry

Designed to store the credentials needed to authenticate with private Docker registries.

Generic Secrets



Stored in key-value pairs

```
$ echo -n 'admin' | base64
YWRtaW4=
$ echo -n 'kubernetes' |
base64 a3ViZXJuZXRlcw==
```

Base-64–encoded strings represent binary data (e.g., images, audio, or code) using text characters.

This is not a form of encryption.

```
apiVersion: v1
kind: Secret
metadata:
  name: demo-secret
type: Opaque
data:
  username: YWRtaW4=
  password: a3ViZXJuZXRlcw==
```

Creating generic Secrets using kubectl create

1. Create a Secret using files

```
$ kubectl create secret generic demo \
   --from-file=./username.txt\
   --from-file=./password.txt
```

2. Create a Secret using literal values

```
$ kubectl create secret generic demo \
   --from-literal user=admin \
   --from-literal password=kubernetes
```

3. Create a Secret using naming keys

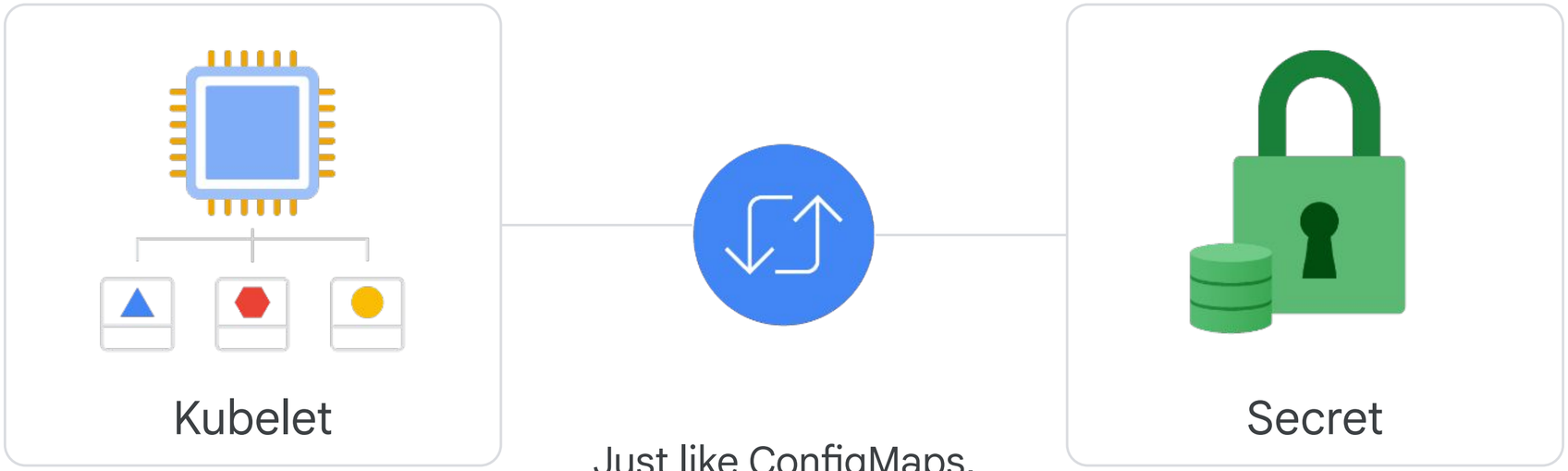
```
$ kubectl create secret generic demo \
   --from-file=User=./username.txt \
   --from-file=Password=./password.txt
```

Separate control planes for Configmaps and Secrets

```
[…] kind: Pod
spec:
  containers:
  - name: mycontainer
    image: redis
    volumeMounts:
    - name: storagesecrets
      mountPath: "/etc/sup"
      readOnly: true
  volumes:
  - name: storagesecrets
    secret:
      secretName: demo-secret
```

Control planes provide a mechanism to create a secure storage that can be used to store and protect Secrets.

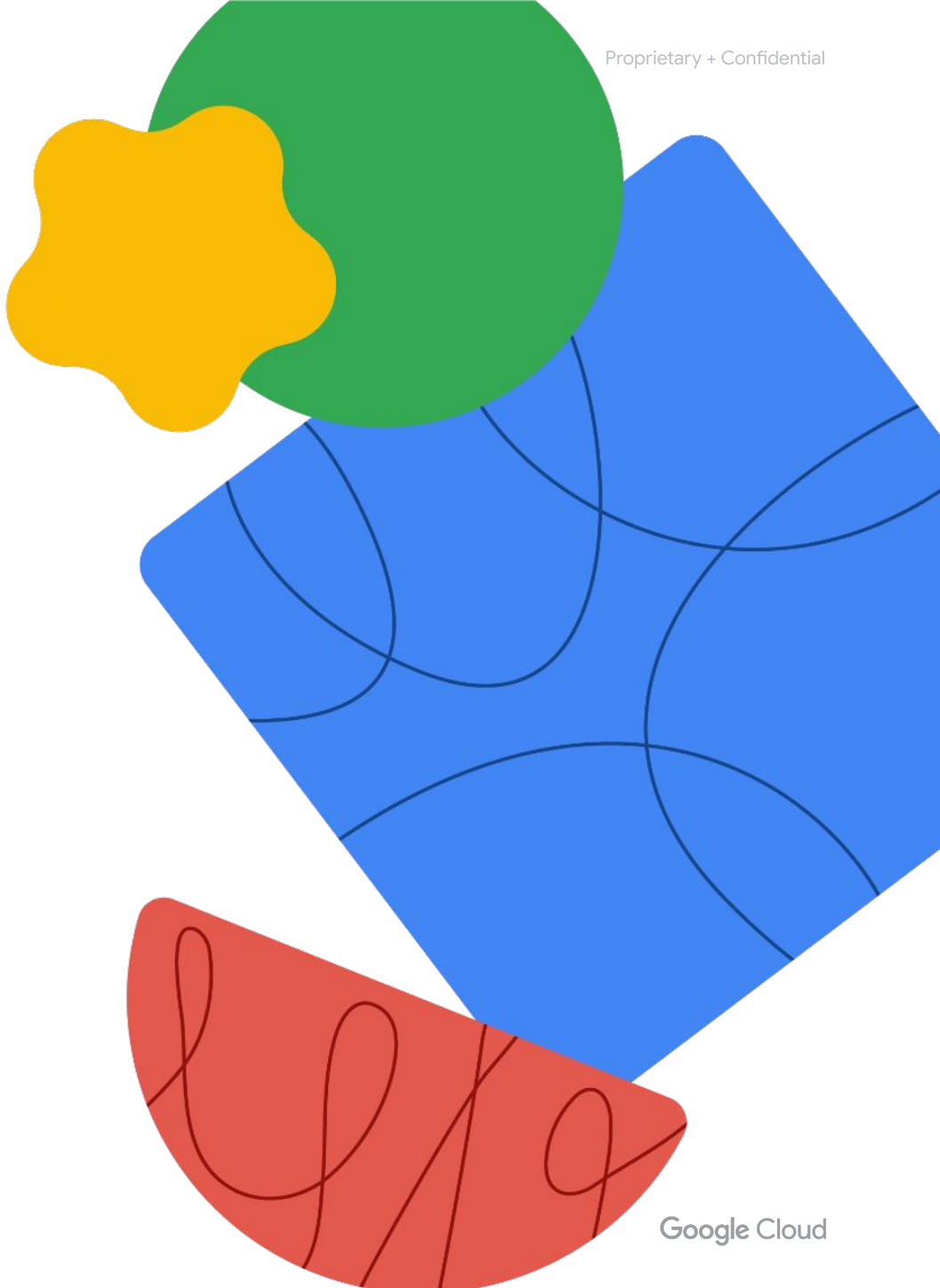
Kubelet keeps a Secret Volume updated



Just like ConfigMaps, kubelet periodically syncs with Secrets to keep a Secret Volume updated.

Quiz questions

Let’s pause for a quick check in.



Quiz | Question 1

Question

A GKE application might need persistent storage. The app owner creates a PersistentVolumeClaim (PVC) with a StorageClassName labeled "standard." What type of storage will likely be used for the volume?

A. Local volume on the node

B. Memory Backed

C. Google Persistent Disk

D. NFS Storage

Quiz | Question 1

Answer

A GKE application might need persistent storage. The app owner creates a PersistentVolumeClaim (PVC) with a StorageClassName labeled "standard." What type of storage will likely be used for the volume?

A. Local volume on the node

B. Memory Backed

C. Google Persistent Disk

D. NFS Storage

Quiz | Question 2

Question

A StatefulSet consists of four Pods that are named Demo-0, Demo-1, Demo-2 and Demo-3. The StatefulSet originally had only two replica Pods but this number was recently increased to four. An update is being rolled out to The StatefulSet is currently being updated with a rolling strategy. Which Pod in the StatefulSet will be updated last?

A. Demo-1

B. Demo-3

C. Demo-0

D. Demo-2

Quiz | Question 2

Answer

A StatefulSet consists of four Pods that are named Demo-0, Demo-1, Demo-2 and Demo-3. The StatefulSet originally had only two replica Pods but this number was recently increased to four. An update is being rolled out to The StatefulSet is currently being updated with a rolling strategy. Which Pod in the StatefulSet will be updated last?

A. Demo-1

B. Demo-3

C. Demo-0 

D. Demo-2

Quiz | Question 3

Question

You have created a ConfigMap and want to make the data available to your application. Where should you configure the directory path parameters in the Pod manifest to allow your application to access the data as files?

A. spec.containers.volumeMounts

B. spec.containers.volumes

C. spec.containers.name

D. spec.containers.env

Quiz | Question 3

Answer

You have created a ConfigMap and want to make the data available to your application. Where should you configure the directory path parameters in the Pod manifest to allow your application to access the data as files?

A. spec.containers.volumeMounts

B. spec.containers.volumes

C. spec.containers.name

D. spec.containers.env

Quiz | Question 4

Question

You need to store image registry credentials to allow Pods to pull images from a private repository. What type of Kubernetes Secret should you create?

A. A generic Secret

B. A TLS Secret

C. A JSON credential file

D. A Docker-Registry Secret

Quiz | Question 4

Answer

You need to store image registry credentials to allow Pods to pull images from a private repository. What type of Kubernetes Secret should you create?

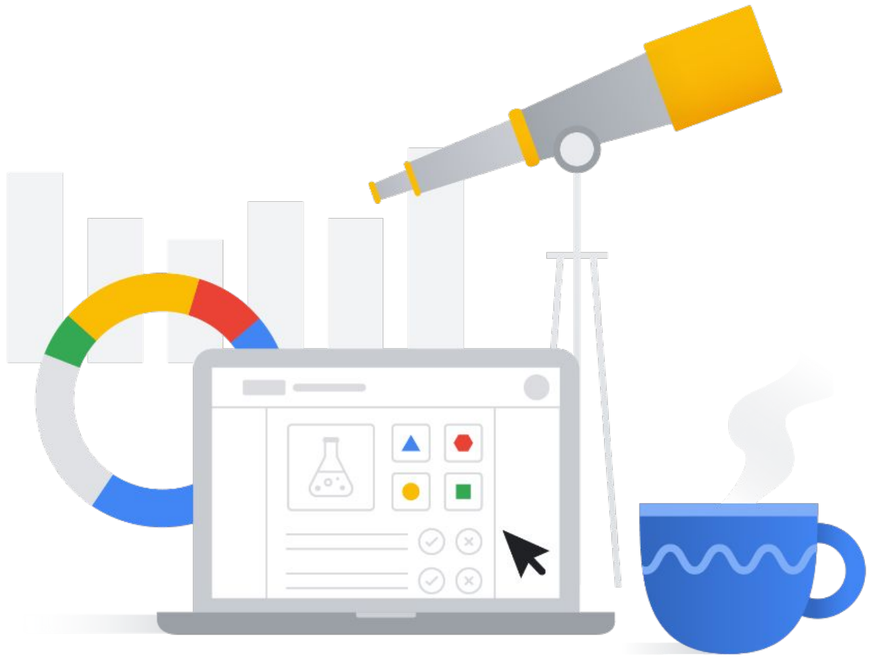
A. A generic Secret

B. A TLS Secret

C. A JSON credential file

D. A Docker-Registry Secret 

Lab: Configuring Persistent Storage for GKE



01 Create manifests for PersistentVolumes and PersistentVolumeClaims.

02 Mount Google Cloud persistent disk PVCs as volumes in Pods.

03 Use manifests to create StatefulSets.

04 Mount Google Cloud persistent disk PVCs as volumes in StatefulSets.