Kubernetes - Part 3

Solar Team

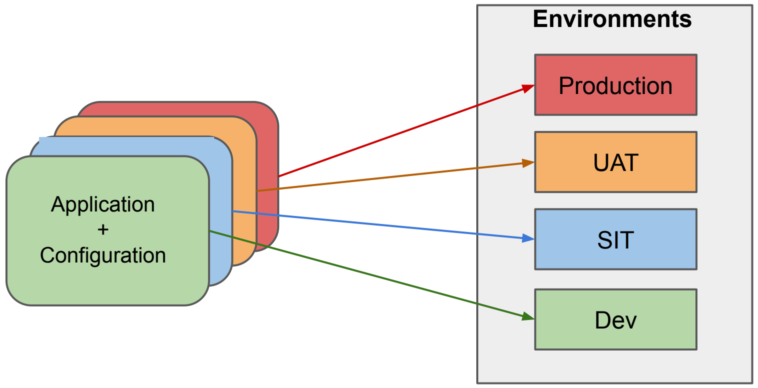
Last Updated: 29/03/2021

Agenda

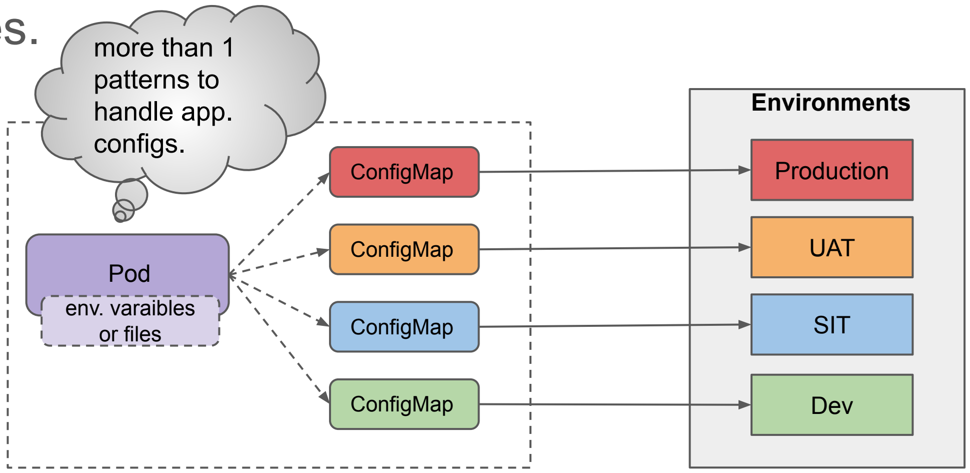
- ConfigMaps
- Secrets
- Volumes
- Persistent Volumes
- Persistent Volume Claims

Application Configuration

- An app's config is everything that is likely to vary between deploys (staging, production, developer environments, etc).(The Twelve-Factor - Config)



- Use ConfigMap - decouple environment-specific configuration from your container images.

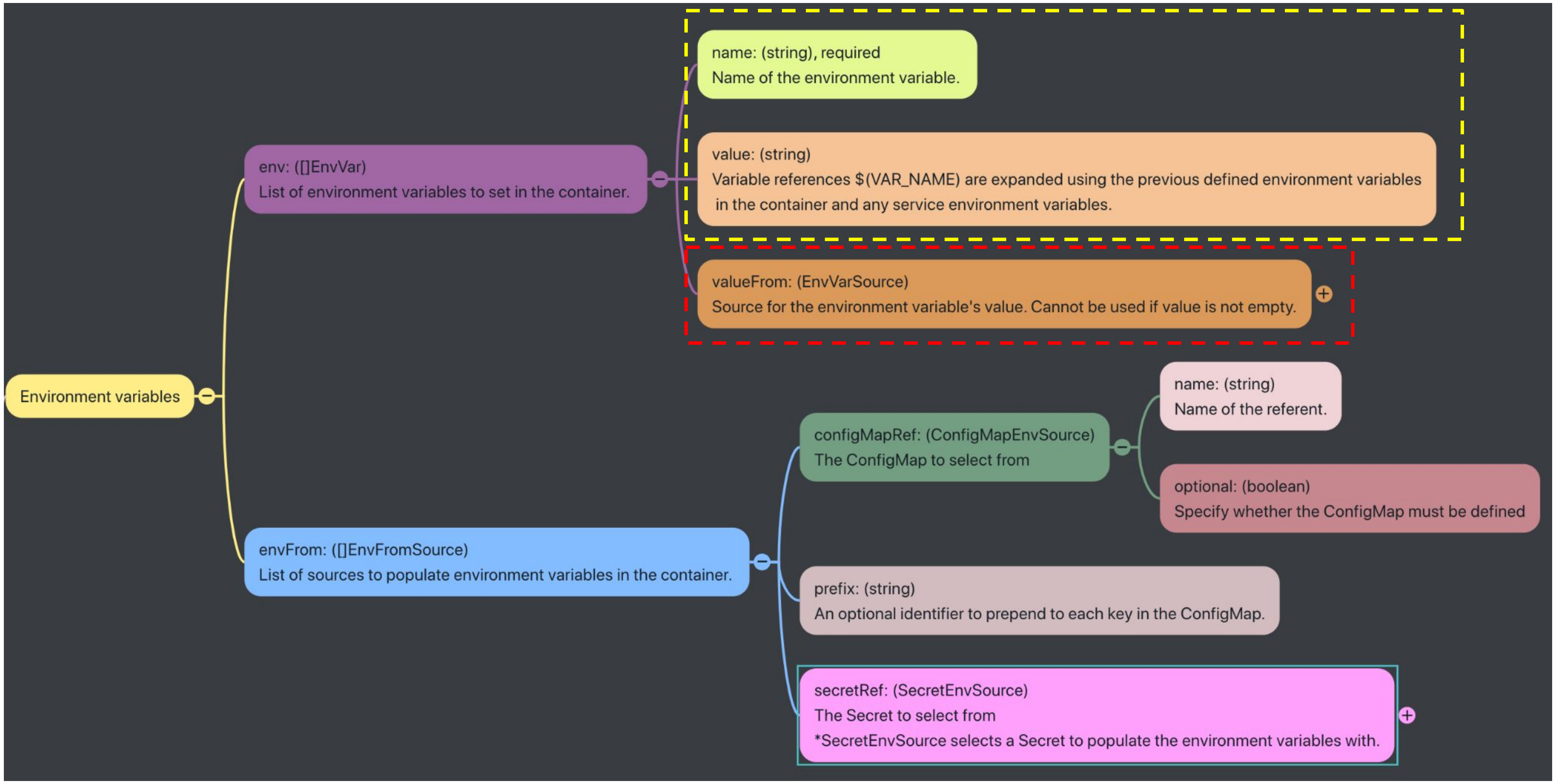


- **Environment Variable**
- **ConfigMap**
- **Volume**

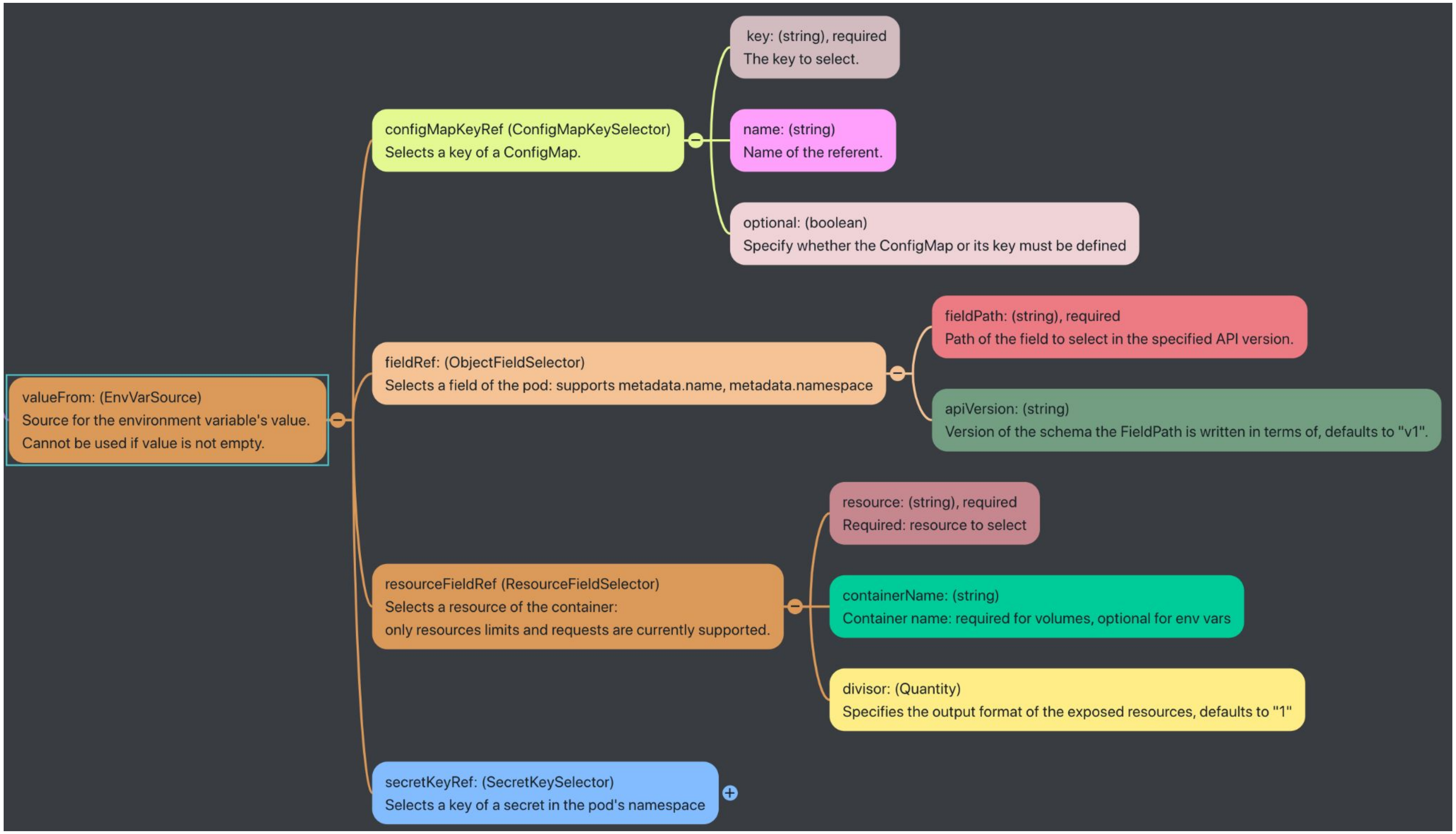
Container Environment Variable

- Can set environment variables for the containers that run in the Pod by including the env or envFrom field in the configuration file.
- Variable Reference $(VAR_NAME).
- Environment variables may reference each other, however ordering is important.
- It override any environment variables specified in the container image.

Environment Variables


Environment variables
env: ([]EnvVar)
List of environment variables to set in the container.
name: (string), required
Name of the environment variable.
value: (string)
Variable references $(VAR_NAME) are expanded using the previous defined environment variables in the container and any service environment variables.
valueFrom: (EnvVarSource)
Source for the environment variable's value. Cannot be used if value is not empty.
envFrom: ([]EnvFromSource)
List of sources to populate environment variables in the container.
configMapRef: (ConfigMapEnvSource)
The ConfigMap to select from
name: (string)
Name of the referent.
optional: (boolean)
Specify whether the ConfigMap must be defined
prefix: (string)
An optional identifier to prepend to each key in the ConfigMap.
secretRef: (SecretEnvSource)
The Secret to select from
*SecretEnvSource selects a Secret to populate the environment variables with.

Environment Variables - valueFrom(EnvVarSource Object)


valueFrom: (EnvVarSource)
Source for the environment variable's value.
Cannot be used if value is not empty.
configMapKeyRef (ConfigMapKeySelector)
Selects a key of a ConfigMap.
key: (string), required
The key to select.
name: (string)
Name of the referent.
optional: (boolean)
Specify whether the ConfigMap or its key must be defined
fieldRef: (ObjectFieldSelector)
Selects a field of the pod: supports metadata.name, metadata.namespace
fieldPath: (string), required
Path of the field to select in the specified API version.
apiVersion: (string)
Version of the schema the FieldPath is written in terms of, defaults to "v1".
resourceFieldRef (ResourceFieldSelector)
Selects a resource of the container:
only resources limits and requests are currently supported.
resource: (string), required
Required: resource to select
containerName: (string)
Container name: required for volumes, optional for env vars
divisor: (Quantity)
Specifies the output format of the exposed resources, defaults to "1"
secretKeyRef: (SecretKeySelector)
Selects a key of a secret in the pod's namespace

Container Environment Variable - Example

```
containers:
- name: env
  image: alpine
  imagePullPolicy: IfNotPresent
  env:
    - name: MESSAGE
      value: "hello world"
    - name: VERSION
      value: "v1.0.0"
    - name: LABEL1
      value: "App version: $(VERSION)"
  command: ["/bin/sh"]
  args: ["-c", "while true; do echo $(MESSAGE) $(LABEL1); sleep 10;done"]
  ...
```

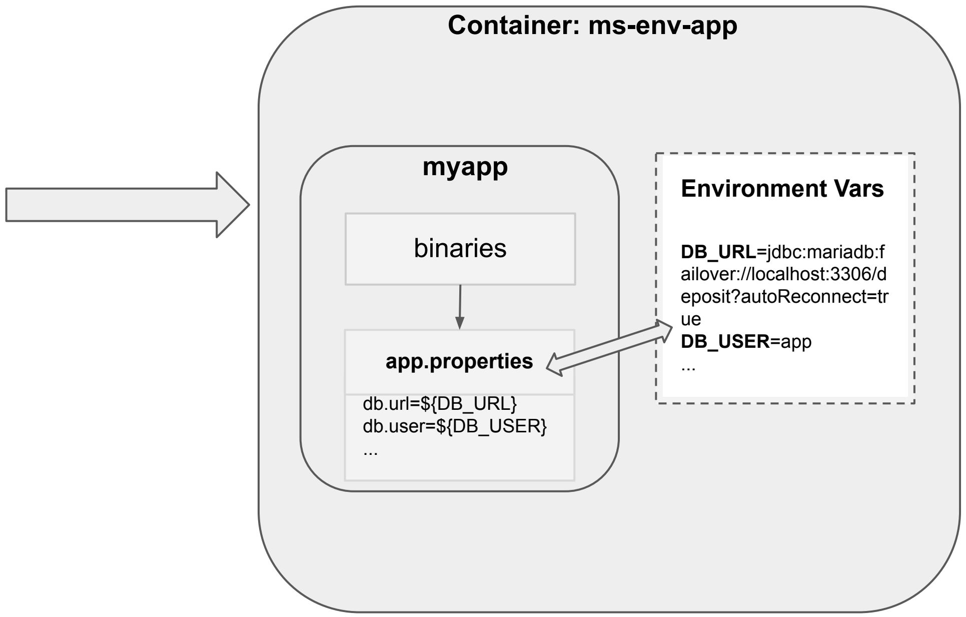
env-pod.yml

- Pod and Container Fields (Kubernetes Variables)
  - Pod fields (fieldRef: (ObjectFieldSelector)) - The values from its metadata, name, service account name, and IP addresses, but the list may grow in the future.
  - Container fields (resourceFieldRef: (ResourceFieldSelector)) - The values for resources are limited to container CPU, memory, and storage limits and requests, but the list may grow in the future.

Using Environment Variables

```
  containers:
  - name: ms-env-app
    image: myapp
    imagePullPolicy: IfNotPresent
    env:
      - name: DB_URL
        value:
"jdbc:mariadb:failover://localhost:3306/deposit
?autoReconnect=true"
      - name: DB_USER
        value: "app"
      - name: DB_PASSWORD
        value: "password"
```

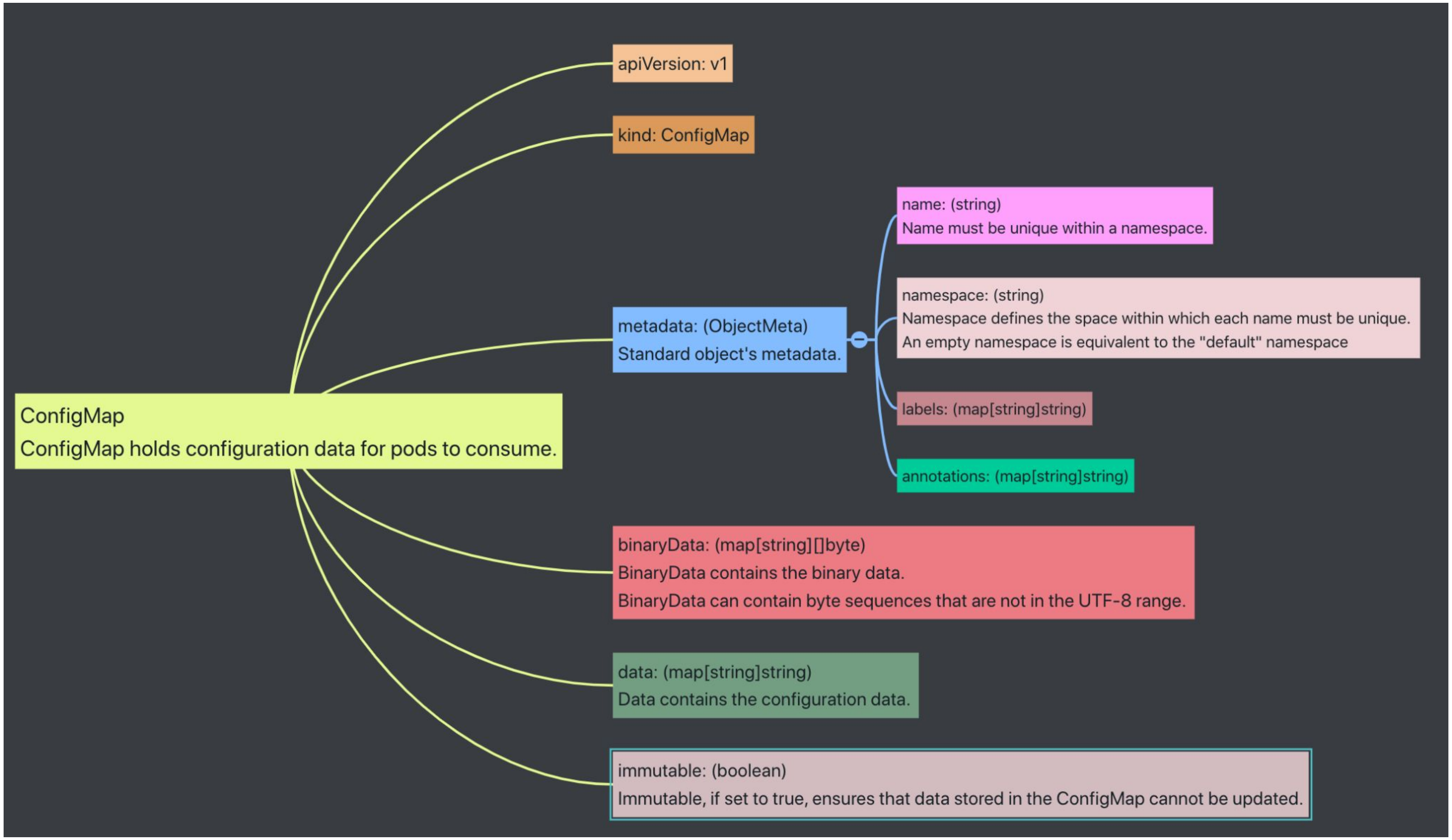
ms-env-pod.yml



ConfigMap Overview

- A ConfigMap is an API object used to store **non-confidential data** in key-value pairs.
- Help to decoupling your configuration from your image, which ensures your application is more portable.
- Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume.
- A ConfigMap is not designed to hold large chunks of data (cannot exceed 1 MiB).

ConfigMap Object



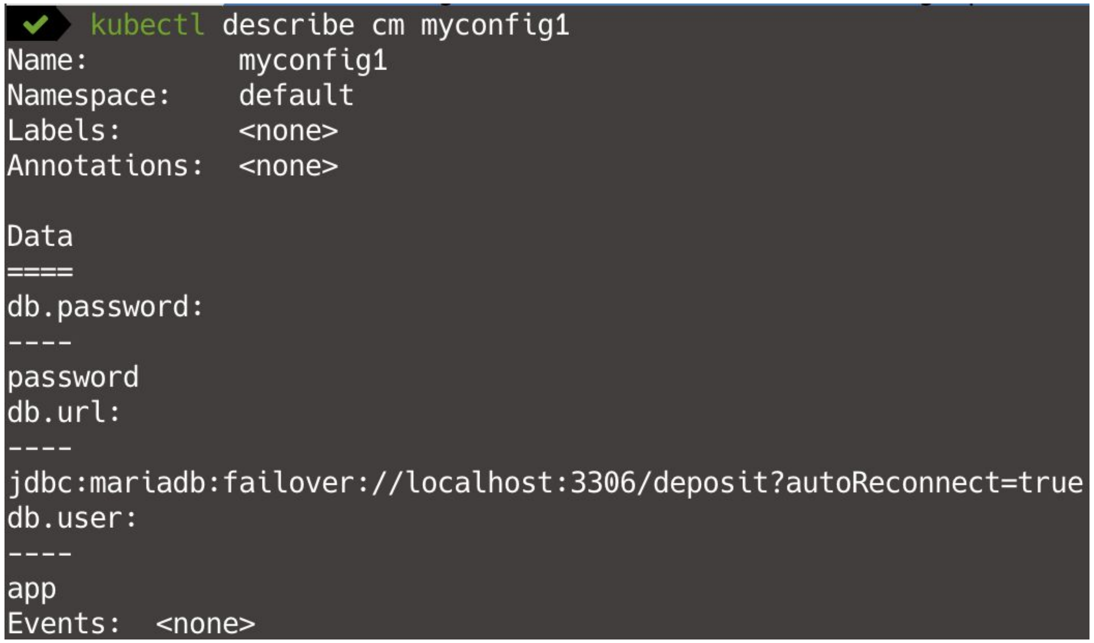
ConfigMap
ConfigMap holds configuration data for pods to consume.
apiVersion: v1
kind: ConfigMap
metadata: (ObjectMeta)
Standard object's metadata.
name: (string)
Name must be unique within a namespace.
namespace: (string)
Namespace defines the space within which each name must be unique.
An empty namespace is equivalent to the "default" namespace
labels: (map[string]string)
annotations: (map[string]string)
binaryData: (map[string][]byte)
BinaryData contains the binary data.
BinaryData can contain byte sequences that are not in the UTF-8 range.
data: (map[string]string)
Data contains the configuration data.
immutable: (boolean)
Immutable, if set to true, ensures that data stored in the ConfigMap cannot be updated.

Create ConfigMap

- Use the kubectl create configmap command to create ConfigMaps
  - kubectl create configmap <map-name> <data-source>
  - <map-name> is the name you want to assign to the ConfigMap and <data-source> is the literal value, directory, or file to draw the data from.
- ConfigMap can be create in different ways (**cm** is a short name of configmap):
  - **Literal Values (--from-literal option)**
    - kubectl create cm aconfig **--from-literal**=**special.how=very**
    - kubectl create cm aconfig --from-literal=special.how=very --from-literal=special.type=charm
  - **Files (--from-file option)**
    - kubectl create cm aconnfig **--from-file**=config-file.conf
    - kubectl create configmap <name> --from-file=<my-key-name>=<path-to-file>
  - **Directories (--from-file option)**
    - kubectl create cm aconfig **--from-file**=configdir/
- You can use them together.
  - kubectl create cm aconfig **--from-literal**=special.how=very **--from-file**=config.conf **--dry-run -o yaml > cfm.yml**

Create ConfigMap - From Literal Values Example

- kubectl create cm myconfig1 **--from-literal=db.url**=jdbc:mariadb:failover://localhost:3306/deposit?autoReconnect=true **--from-literal=db.user**=app **--from-literal=db.password**=password
- kubectl describe cm myconfig1



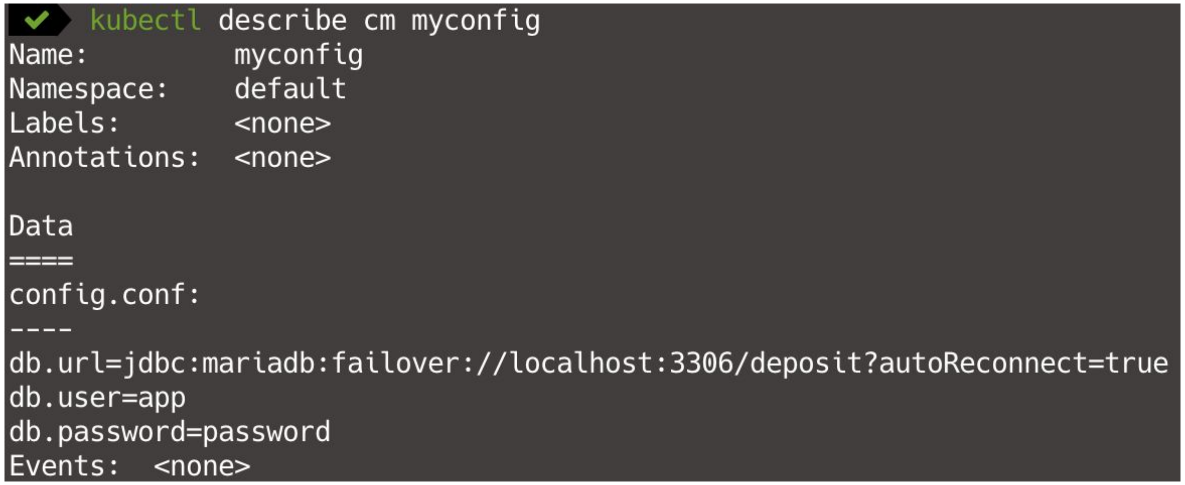
```
✔ kubectl describe cm myconfig1
Name:         myconfig1
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
db.password:
----
password
db.url:
----
jdbc:mariadb:failover://localhost:3306/deposit?autoReconnect=true
db.user:
----
app
Events:  <none>
```

Create ConfigMap - From File Example

```
db.url=jdbc:mariadb:failover://localhost:3306/deposit?autoReconnect=true
db.user=app
db.password=password
```

config.conf

- kubectl create cm myconfig2 --from-file=config.conf
- kubectl describe cm myconfig2



```
 ✓  kubectl describe cm myconfig
Name:         myconfig
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
config.conf:
----
db.url=jdbc:mariadb:failover://localhost:3306/deposit?autoReconnect=true
db.user=app
db.password=password
Events:  <none>
```

Create ConfigMap - From Directory Example

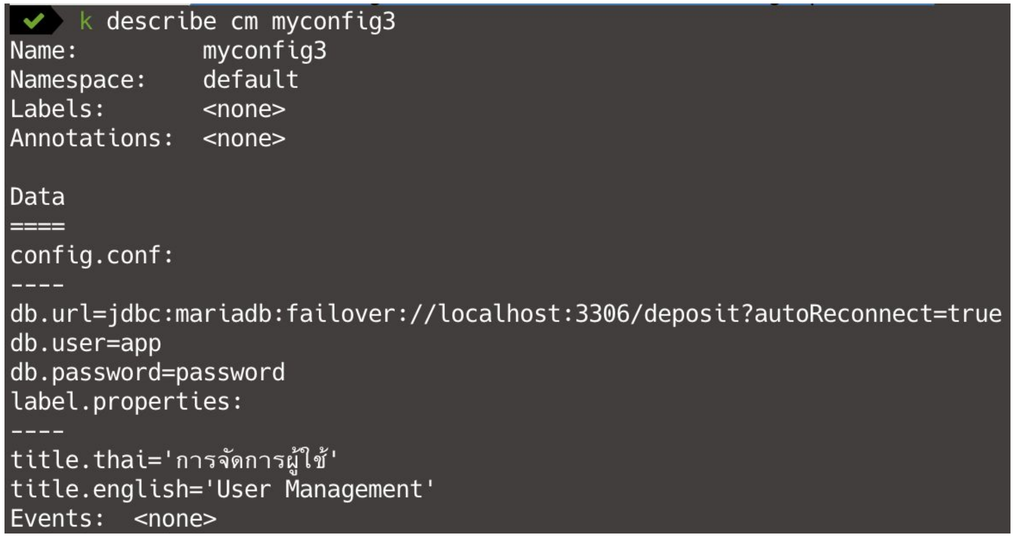
```
db.url=jdbc:mariadb:failover://localhost:3306/d
eposit?autoReconnect=true
db.user=app
db.password=password
```

config.conf

```
title.thai='การจัดการผู้ใช้'
title.english='User Management'
```

label.properties

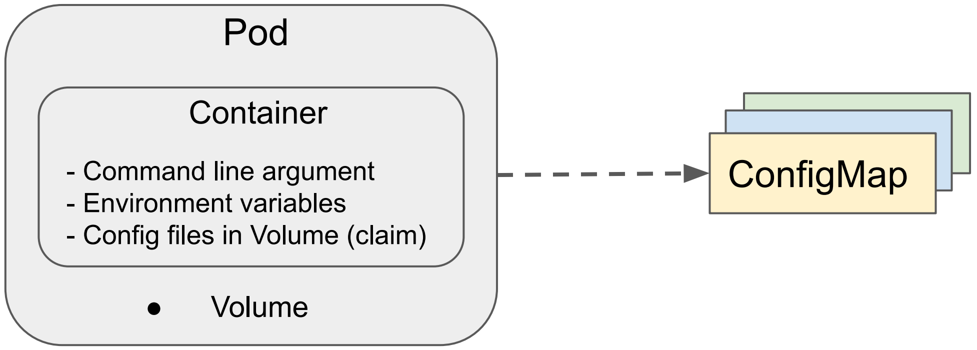
- kubectl create cm myconfig3 --from-file=./config-demo/
- kubectl describe cm myconfig3



```
 ✔  k describe cm myconfig3
Name:         myconfig3
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
config.conf:
----
db.url=jdbc:mariadb:failover://localhost:3306/deposit?autoReconnect=true
db.user=app
db.password=password
label.properties:
----
title.thai='การจัดการผู้ใช้'
title.english='User Management'
Events:  <none>
```

Using ConfigMap

- Four different ways to use a ConfigMap.
  - Inside a container **command-line args.**
  - **Environment variables** for a container.
  - Add a file in **volume**, for the application to read.
  - Write code to run inside the Pod that uses the Kubernetes API to read a ConfigMap.



Using ConfigMap - Inside a container command and args

- Using the ConfigMap in a container with **env** and **envFrom**.

args-pod.yml

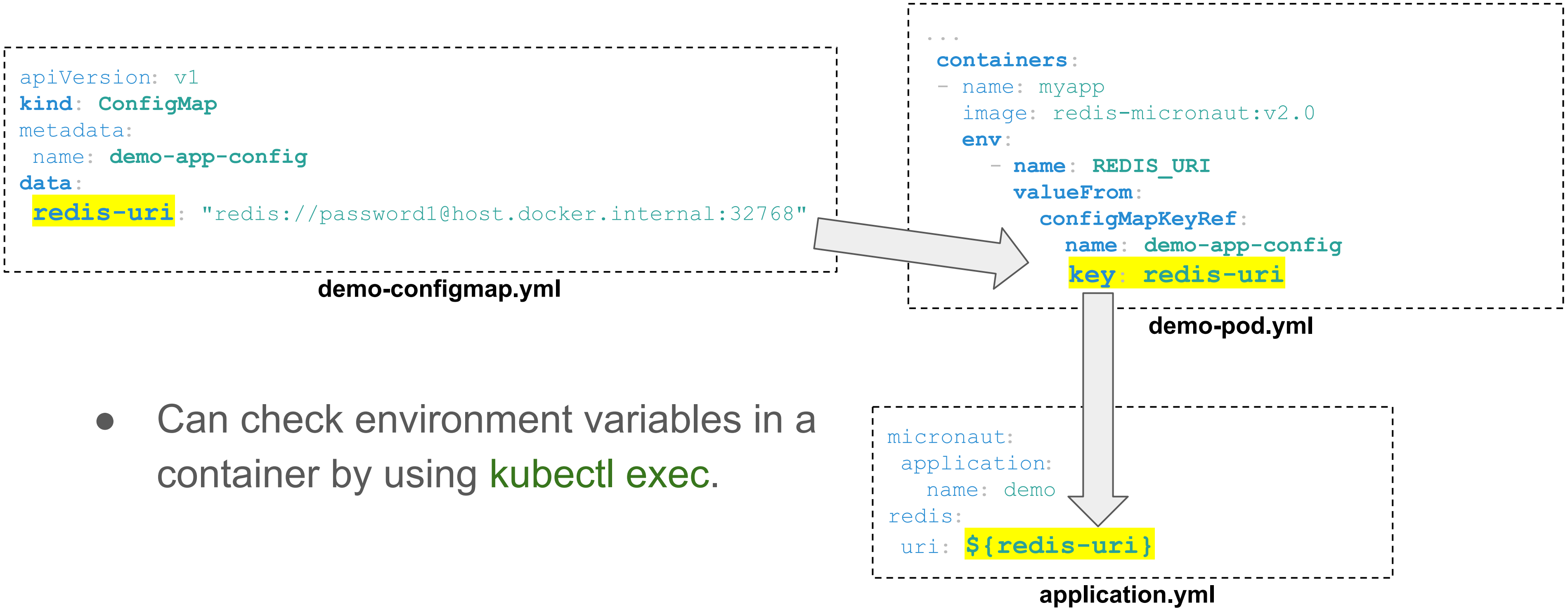
```
containers:
- name: args-app
  image: alpine
  imagePullPolicy: IfNotPresent
  env:
    - name: DB_URL
      valueFrom:
        configMapKeyRef:
          name: myconfig1
          key: db.url
    - name: DB_USER
      valueFrom:
        configMapKeyRef:
          name: myconfig1
          key: db.user
    - name: DB_PASSWORD
      valueFrom:
        configMapKeyRef:
          name: myconfig1
          key: db.password
  command: ["/bin/sh"]
  args: ["-c", "while true; do echo
database=$(DB_URL), user=$(DB_USER) and
password=$(DB_PASSWORD); sleep 10;done"]
```

```
containers:
- name: args-envfrom-app
  image: alpine
  imagePullPolicy: IfNotPresent
  envFrom:
    - configMapRef:
        name: myconfig1
  command: ["/bin/sh"]
  args: ["-c", "while true; do echo
database=$(db.url), user=$(db.user)
and password=$(db.password); sleep
10;done"]
```

args-envfrom- pod.yml

Using ConfigMap - Environment variables for a container

- Create a ConfigMap using literal values.
  - kubectl create cm demo-app-config --from-literal=redis-uri="redis://password1@host.docker.internal:32768"
- Use the ConfigMap in a container.



```yaml
apiVersion: v1
kind: ConfigMap
metadata:
 name: demo-app-config
data:
 redis-uri: "redis://password1@host.docker.internal:32768"
```

demo-configmap.yml

```yaml
...
 containers:
 - name: myapp
   image: redis-micronaut:v2.0
   env:
     - name: REDIS_URI
       valueFrom:
         configMapKeyRef:
           name: demo-app-config
           key: redis-uri
```

demo-pod.yml

```yaml
micronaut:
 application:
   name: demo
redis:
 uri: ${redis-uri}
```

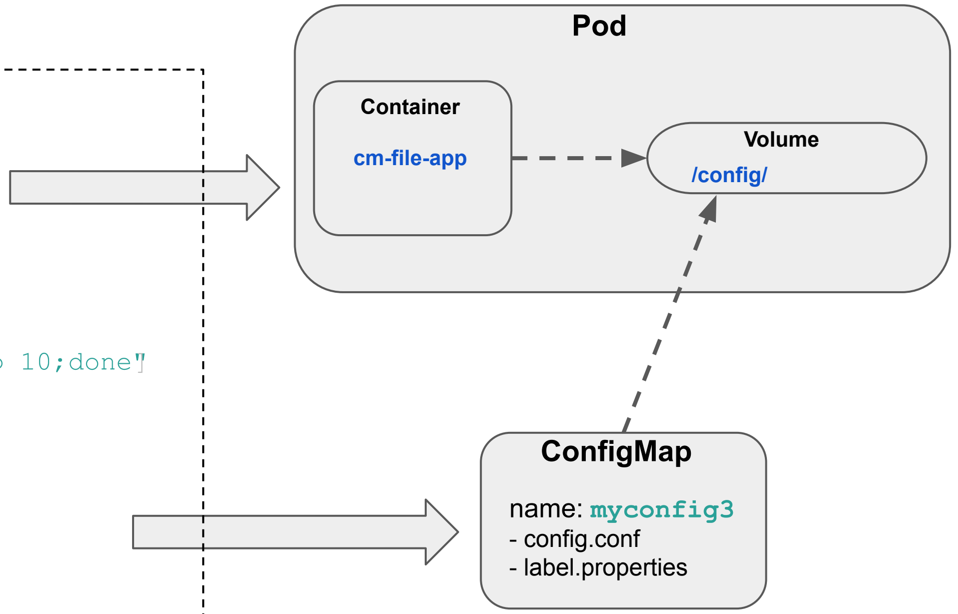
application.yml

- Can check environment variables in a container by using kubectl exec.

Using ConfigMap - File in Volume

```yaml
containers:
- name: cm-file-app
  image: alpine
  imagePullPolicy: IfNotPresent
  volumeMounts:
    - name: config-volume
      mountPath: /config/
  command: ["/bin/sh"]
  args: ["-c", "while true; do date; sleep 10;done"]
  resources:
    limits:
      memory: "32Mi"
      cpu: "500m"
volumes:
  - name: config-volume
    configMap:
      name: myconfig3
```

cm-file-pod.yml

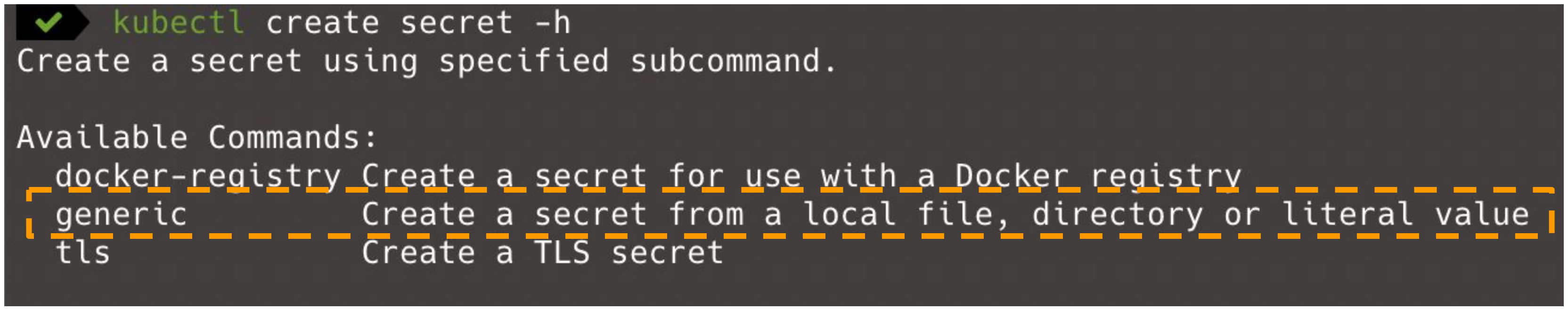


- Can check config files in a container by using kubectl exec.

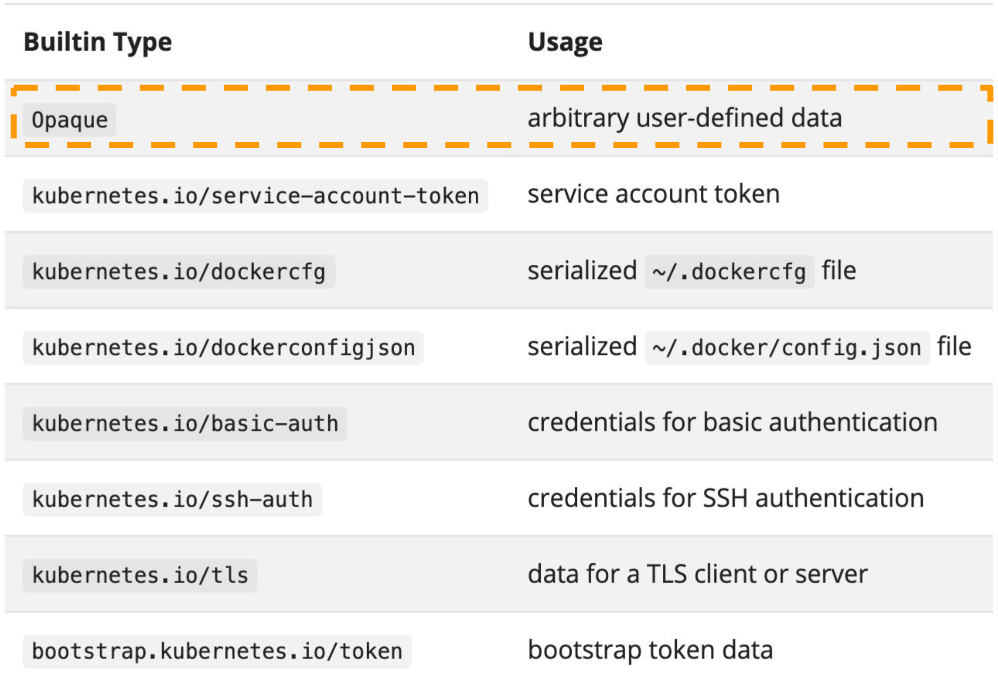
Secrets

- Secrets let you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys.
- Storing confidential information in a Secret is safer and more flexible than putting it verbatim in a Pod definition or in a container image.
- Secrets are **unencrypted** base64-encoded strings and stored as plaintext in etcd.
- Secrets are created and used by Pod in the same way as ConfigMap.
- Some Secrets are autometically created by Kubernetes itself.
- Secrets are encoded, not encrypted.

Secret Types



```
✓ kubectl create secret -h
Create a secret using specified subcommand.

Available Commands:
  docker-registry Create a secret for use with a Docker registry
  generic         Create a secret from a local file, directory or literal value
  tls             Create a TLS secret
```

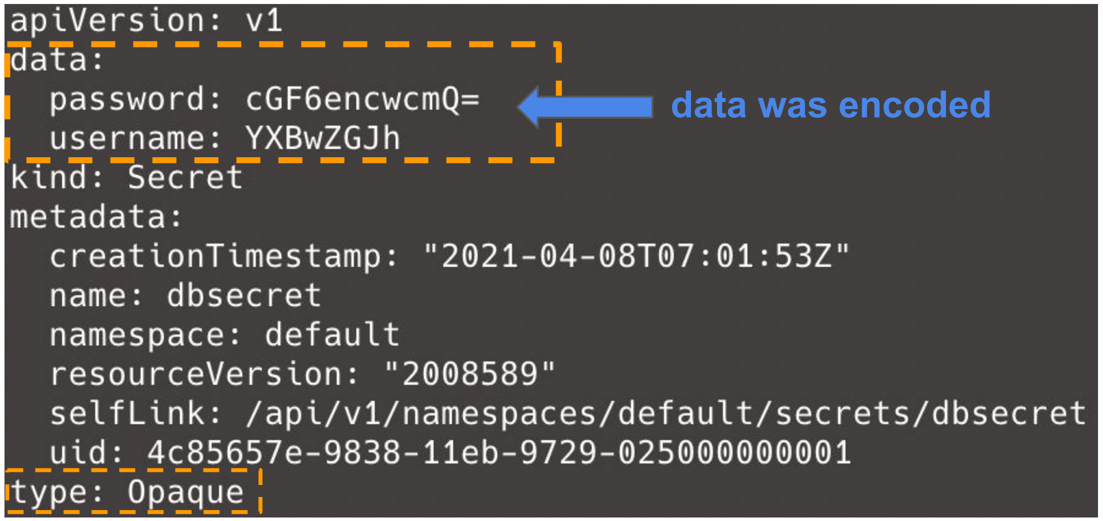


Builtin Type	Usage
`Opaque`	arbitrary user-defined data
`kubernetes.io/service-account-token`	service account token
`kubernetes.io/dockercfg`	serialized `~/.dockercfg` file
`kubernetes.io/dockerconfigjson`	serialized `~/.docker/config.json` file
`kubernetes.io/basic-auth`	credentials for basic authentication
`kubernetes.io/ssh-auth`	credentials for SSH authentication
`kubernetes.io/tls`	data for a TLS client or server
`bootstrap.kubernetes.io/token`	bootstrap token data

- **Opaque** is the default Secret type if omitted from a Secret configuration file.
- kubectl create secret **generic** empty-secret
- kubectl get secret empty-secret

Create Secret - Using kubectl Command

- Create Secret using kubectl command.
  - kubectl create secret generic dbsecret --from-literal=**username=appdba** --from-literal=**password=pazzw0rd**
  - kubectl get secret dbsecret -o yaml



```
apiVersion: v1
data:
  password: cGF6encwcmQ=
  username: YXBwZGJh
kind: Secret
metadata:
  creationTimestamp: "2021-04-08T07:01:53Z"
  name: dbsecret
  namespace: default
  resourceVersion: "2008589"
  selfLink: /api/v1/namespaces/default/secrets/dbsecret
  uid: 4c85657e-9838-11eb-9729-025000000001
type: Opaque
```

Create Secret - Using Config File

- Create Secret from config file.
  - kubectl create secret generic dbsecret --from-file=**dbsecret.yml**

```
apiVersion: v1
data:
 password: pazzw0rd
 username: appdba
kind: Secret
metadata:
 name: dbsecret
type: Opaque
```

dbsecret.yml

```
echo -n ‘'pazzw0rd' | base64
echo -n ‘'appdba' | base64
```

```
apiVersion: v1
data:
 password: cGF6encwcmQK
 username: YXBwZGJhCg==
kind: Secret
metadata:
 name: dbsecret
type: Opaque
```

- Data must be manually encoded as a base64 string or use **stringData field** instead.

```
stringData:
 password: pazzw0rd
 username: appdba
```

dbsecret-stringdata.yml

Using Secrets

- Secrets can be used by a container in a Pod:
  - Exposed as **environment variables** (env and envFrom)
  - Mounted as **data volumes**

```
containers:
- name: mypod
  image: radial/busyboxplus
  imagePullPolicy: IfNotPresent
  env:
    - name: USERNAME
      valueFrom:
        secretKeyRef:
          name: dbsecret
          key: username
    - name: PASSWORD
      valueFrom:
        secretKeyRef:
          name: dbsecret
          key: password
```

secret-used-as-env.yml

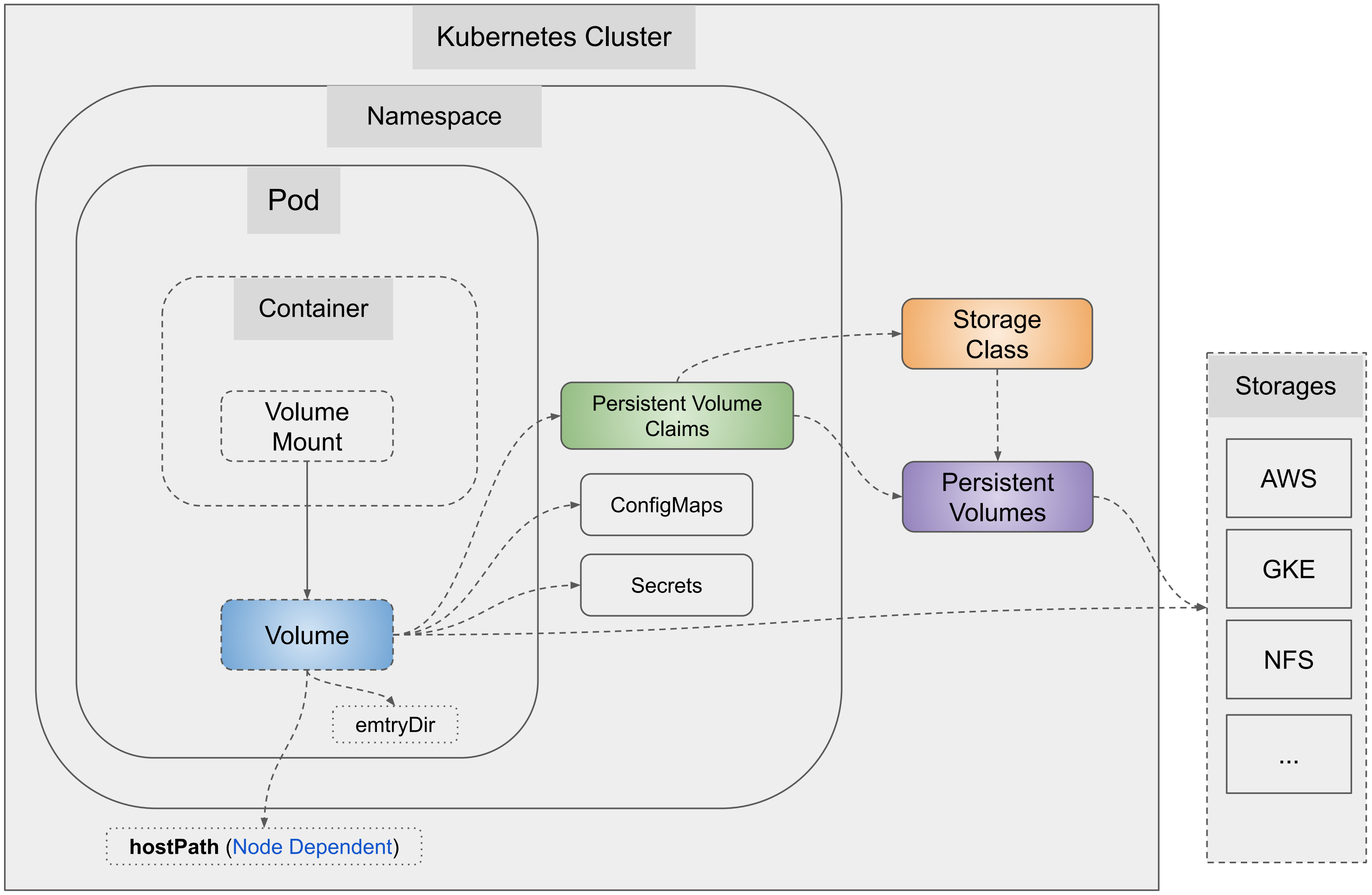
```
containers:
- name: mypod
  image: radial/busyboxplus
  imagePullPolicy: IfNotPresent
  command:
    - sleep
    - "3600"
  volumeMounts:
  - name: dbinfo
    mountPath: "/config/"
    readOnly: true
  resources: {}
volumes:
- name: dbinfo
  secret:
    secretName: dbsecret
```

secret-used-as-vol.yml

What are Kubernetes Volumes?

- Data and State.
- A volume is a directory, possibly with some data in it, which is accessible to the containers in a pod.
- Two types of volume
  - Ephemeral Volume - have a lifetime of a pod.
  - Persistent Volumes - exist beyond the lifetime of a pod.
- Volumes and Storages
  - Kubernetes volumes support both local and remote storages
- Different drivers and types.
- Create/refer volumes in a pod and then uses them in containers.
- Pod/Node independent volumes with Persistent Volumes.
- Support both file and block storages.

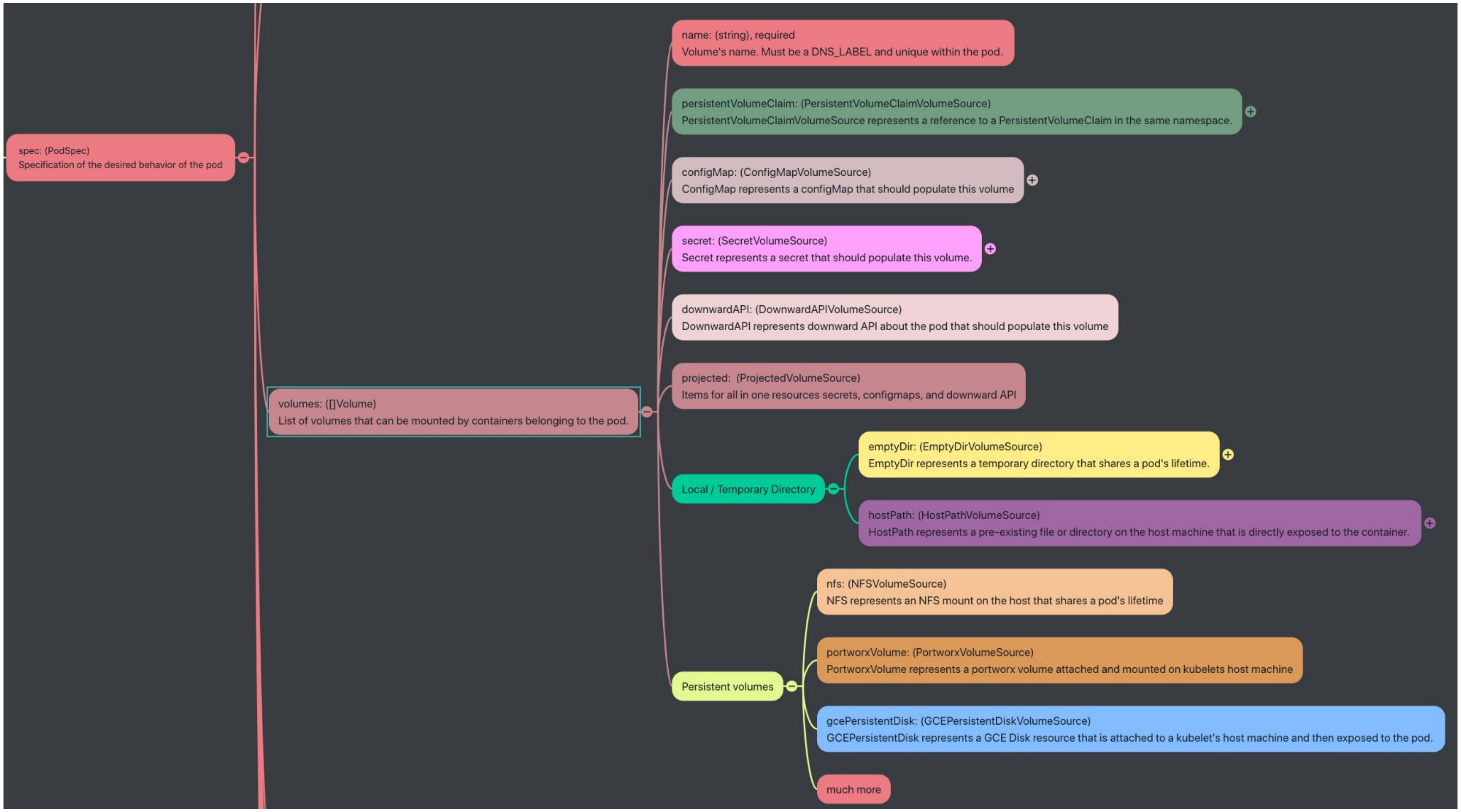
Volumes, Persistent Volumes and Persistent Volume Claims



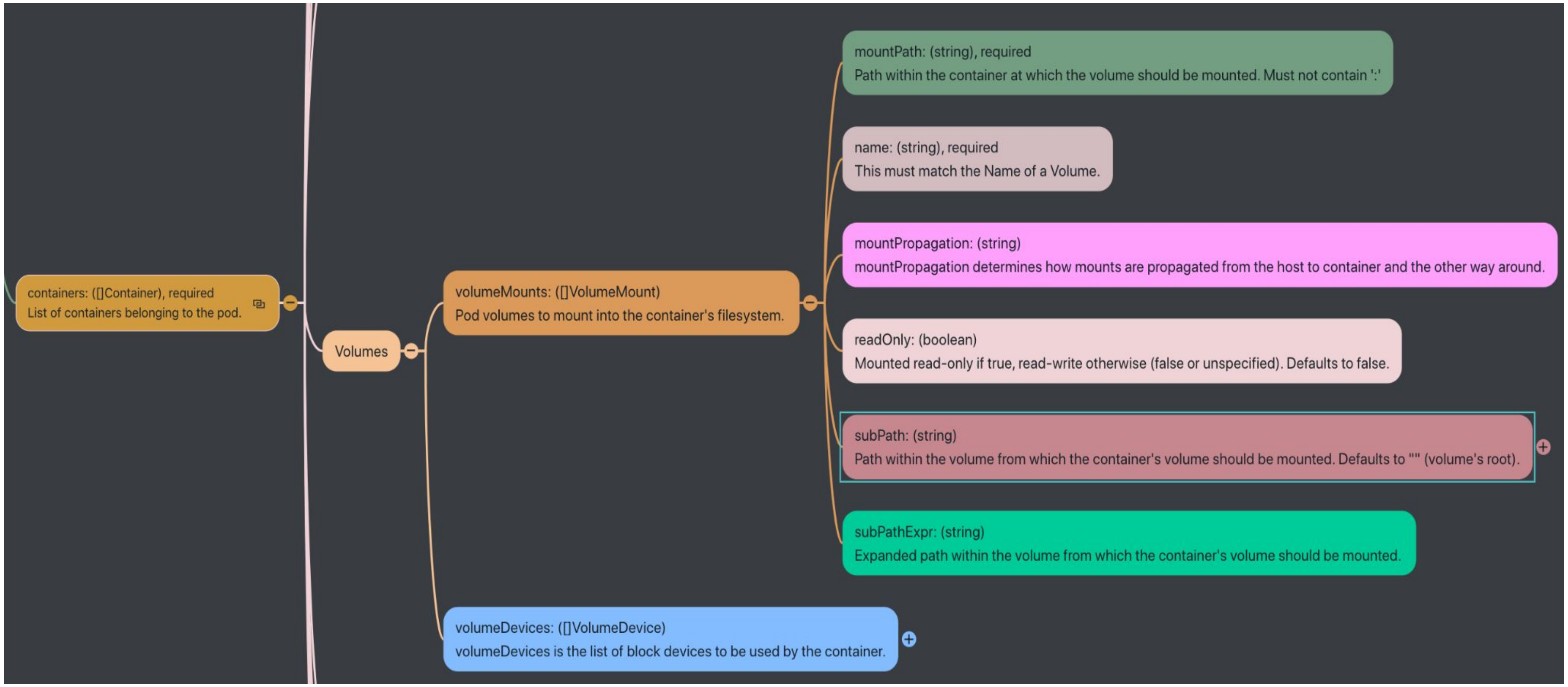
Volumes

- Ephemeral volume types.
- Volumes are pod/node specific.
- Volume lifetime depends on the Pod lifetime.
- Kubernetes supports many types of volumes.
- A Pod can use any number of volume types simultaneously.
- To use a volume:
    - Specify the volumes to provide for the Pod in *.spec.volumes*.
    - Declare where to mount those volumes into containers in *.spec.containers[*].volumeMounts*.

Volume Object


spec: (PodSpec)
Specification of the desired behavior of the pod
volumes: ([]Volume)
List of volumes that can be mounted by containers belonging to the pod.
name: (string), required
Volume's name. Must be a DNS_LABEL and unique within the pod.
persistentVolumeClaim: (PersistentVolumeClaimVolumeSource)
PersistentVolumeClaimVolumeSource represents a reference to a PersistentVolumeClaim in the same namespace.
configMap: (ConfigMapVolumeSource)
ConfigMap represents a configMap that should populate this volume
secret: (SecretVolumeSource)
Secret represents a secret that should populate this volume.
downwardAPI: (DownwardAPIVolumeSource)
DownwardAPI represents downward API about the pod that should populate this volume
projected: (ProjectedVolumeSource)
Items for all in one resources secrets, configmaps, and downward API
Local / Temporary Directory
emptyDir: (EmptyDirVolumeSource)
EmptyDir represents a temporary directory that shares a pod's lifetime.
hostPath: (HostPathVolumeSource)
HostPath represents a pre-existing file or directory on the host machine that is directly exposed to the container.
Persistent volumes
nfs: (NFSVolumeSource)
NFS represents an NFS mount on the host that shares a pod's lifetime
portworxVolume: (PortworxVolumeSource)
PortworxVolume represents a portworx volume attached and mounted on kubelets host machine
gcePersistentDisk: (GCEPersistentDiskVolumeSource)
GCEPersistentDisk represents a GCE Disk resource that is attached to a kubelet's host machine and then exposed to the pod.
much more

VolumeMount Object in Container



containers: ([]Container), required
List of containers belonging to the pod.

volumeMounts: ([]VolumeMount)
Pod volumes to mount into the container's filesystem.

mountPath: (string), required
Path within the container at which the volume should be mounted. Must not contain ':'

name: (string), required
This must match the Name of a Volume.

mountPropagation: (string)
mountPropagation determines how mounts are propagated from the host to container and the other way around.

readOnly: (boolean)
Mounted read-only if true, read-write otherwise (false or unspecified). Defaults to false.

subPath: (string)
Path within the volume from which the container's volume should be mounted. Defaults to "" (volume's root).

subPathExpr: (string)
Expanded path within the volume from which the container's volume should be mounted.

volumeDevices: ([]VolumeDevice)
volumeDevices is the list of block devices to be used by the container.

Volume Types

- awsElasticBlockStore
- azureDisk
- azureFile
- cephfs
- cinder
- configMap
- downwardAPI
- emptyDir
- fc (fibre channel)
- flocker (deprecated)
- gcePersistentDisk
- gitRepo (deprecated)
- glusterfs
- hostPath
- iscsi
- local
- nfs
- persistentVolumeClaim
- portworxVolume
- projected
- quobyte
- rbd
- scaleIO (deprecated)
- secret
- storageOS
- vsphereVolume

https://kubernetes.io/docs/concepts/storage/volumes/

Volumes - Example

```
apiVersion: v1
kind: Pod
metadata:
 name: hostpath-emptydir-pod
 labels:
   name: hostpath-emptydir-pod
spec:
 terminationGracePeriodSeconds: 0
 containers:
 - name: hostpath-emptydir-app
   image: radial/busyboxplus
   imagePullPolicy: IfNotPresent
   volumeMounts:
     - name: config-volume
       mountPath: /config/
     - name: emptydir-volume
       mountPath: /emptydir
   command:
     - "sleep"
     - "3600"
   resources:
     limits:
       memory: "32Mi"
       cpu: "500m"
 volumes:
   - name: config-volume
     hostPath:
       path: /Users/ktb_user/temp/redisconfig
       type: Directory
   - name: emptydir-volume
     emptyDir: {}
```

using volumes

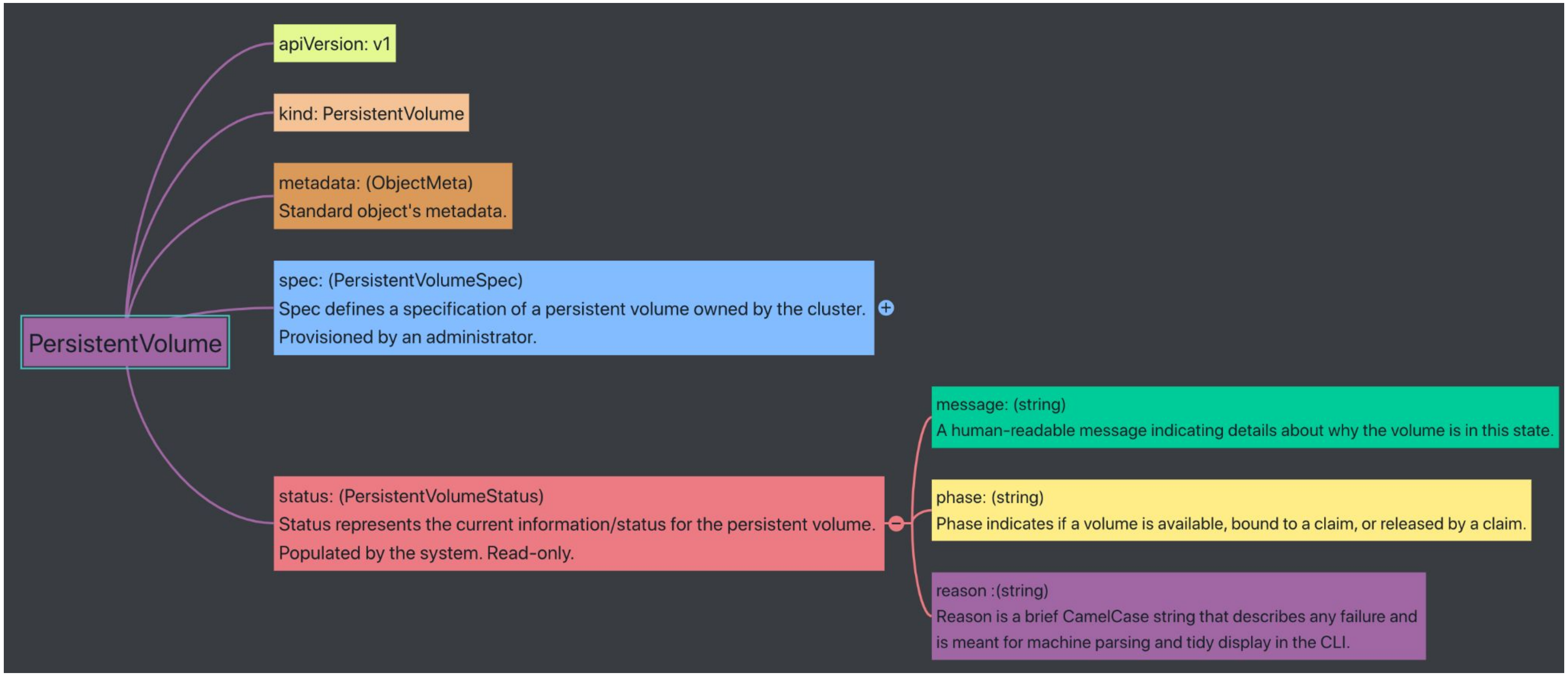
create volumes

hostpath-emptydir-pod.yml

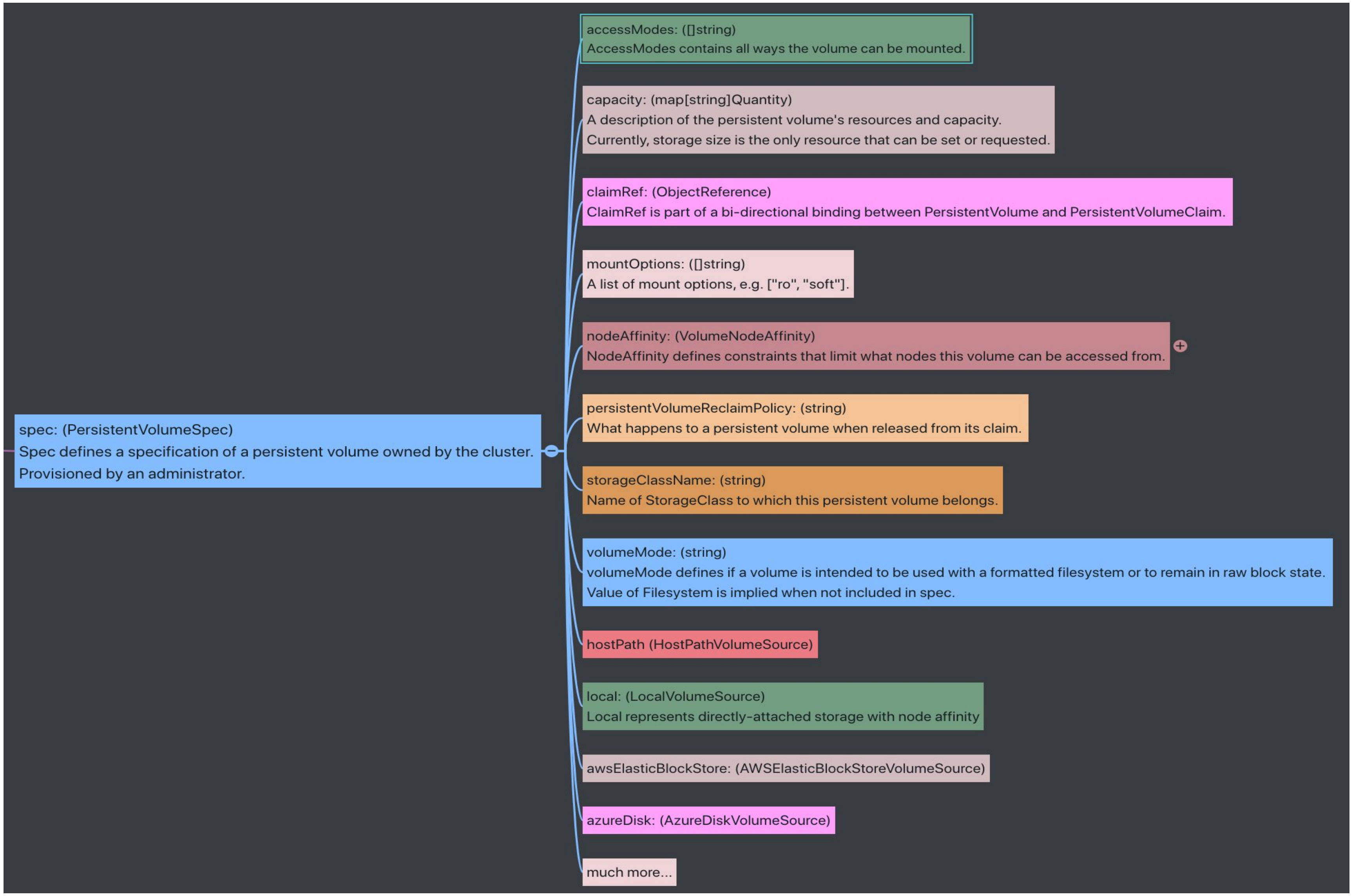
Persistent Volumes

- A PersistentVolume (PV) is a piece of storage in the cluster.
- Generally provisioned by an administrator.
- One of Kubernetes primitive type.
- Pod/Node independent.
- Defines it once and then use it in multiple Pods.
- Provisioned by an administrator (static provisioned) or dynamically provisioned using Storage Classes.
    - kubectl get sc;
    - kubectl describe sc <storage-class name>
- It types almost similar to the types of regular volumes.
- Many features likes, Capacity, Volume Mode, Access Modes, Reclaim Policy, etc.

PersistentVolume Object


PersistentVolume
apiVersion: v1
kind: PersistentVolume
metadata: (ObjectMeta)
Standard object's metadata.
spec: (PersistentVolumeSpec)
Spec defines a specification of a persistent volume owned by the cluster.
Provisioned by an administrator.
status: (PersistentVolumeStatus)
Status represents the current information/status for the persistent volume.
Populated by the system. Read-only.
message: (string)
A human-readable message indicating details about why the volume is in this state.
phase: (string)
Phase indicates if a volume is available, bound to a claim, or released by a claim.
reason :(string)
Reason is a brief CamelCase string that describes any failure and
is meant for machine parsing and tidy display in the CLI.

PersistentVolume Object - Spec



spec: (PersistentVolumeSpec)
Spec defines a specification of a persistent volume owned by the cluster.
Provisioned by an administrator.

accessModes: ([]string)
AccessModes contains all ways the volume can be mounted.

capacity: (map[string]Quantity)
A description of the persistent volume's resources and capacity.
Currently, storage size is the only resource that can be set or requested.

claimRef: (ObjectReference)
ClaimRef is part of a bi-directional binding between PersistentVolume and PersistentVolumeClaim.

mountOptions: ([]string)
A list of mount options, e.g. ["ro", "soft"].

nodeAffinity: (VolumeNodeAffinity)
NodeAffinity defines constraints that limit what nodes this volume can be accessed from.

persistentVolumeReclaimPolicy: (string)
What happens to a persistent volume when released from its claim.

storageClassName: (string)
Name of StorageClass to which this persistent volume belongs.

volumeMode: (string)
volumeMode defines if a volume is intended to be used with a formatted filesystem or to remain in raw block state.
Value of Filesystem is implied when not included in spec.

hostPath (HostPathVolumeSource)

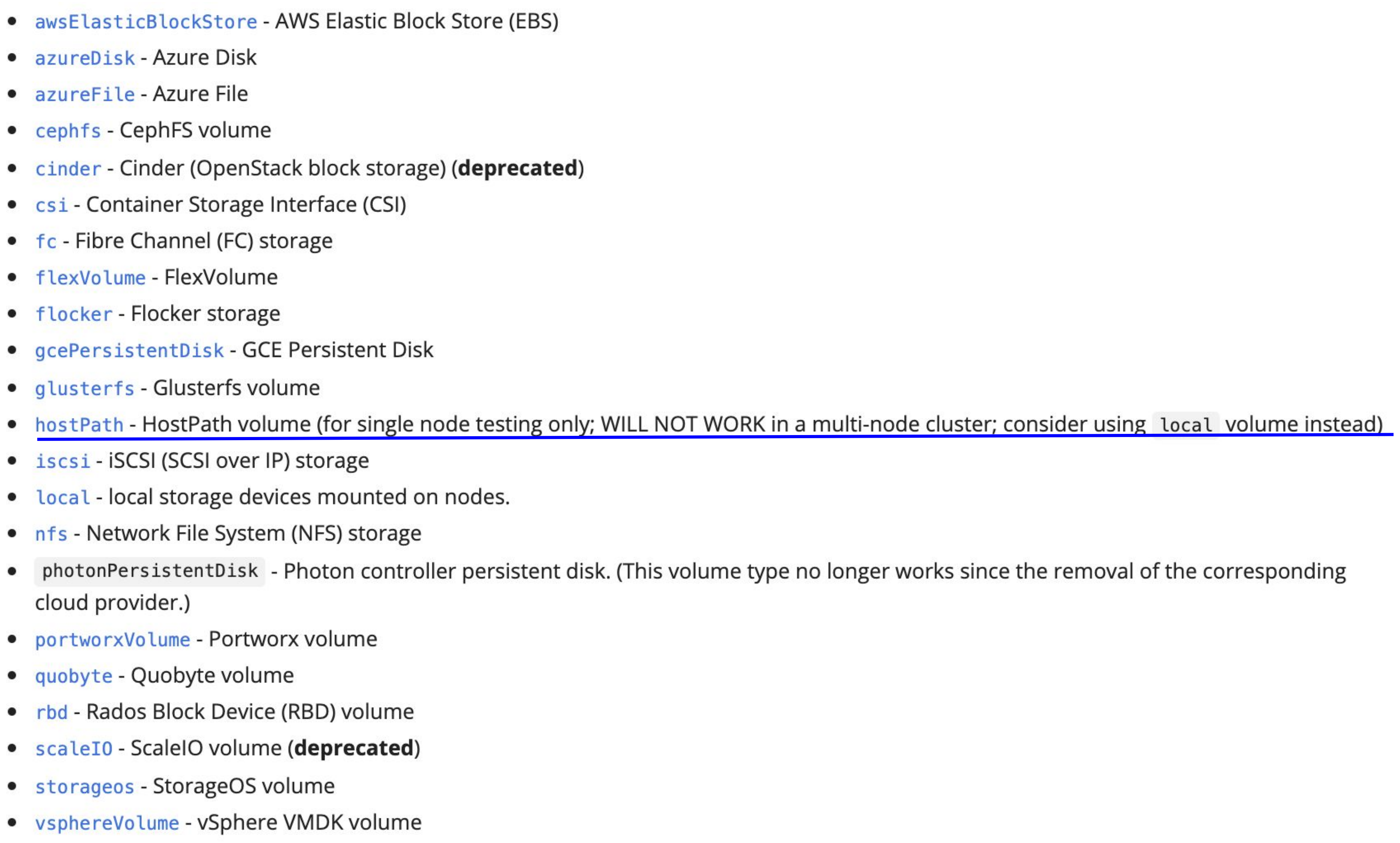
local: (LocalVolumeSource)
Local represents directly-attached storage with node affinity

awsElasticBlockStore: (AWSElasticBlockStoreVolumeSource)

azureDisk: (AzureDiskVolumeSource)

much more...

Persistent Volume Types



- `awsElasticBlockStore` - AWS Elastic Block Store (EBS)
- `azureDisk` - Azure Disk
- `azureFile` - Azure File
- `cephfs` - CephFS volume
- `cinder` - Cinder (OpenStack block storage) (**deprecated**)
- `csi` - Container Storage Interface (CSI)
- `fc` - Fibre Channel (FC) storage
- `flexVolume` - FlexVolume
- `flocker` - Flocker storage
- `gcePersistentDisk` - GCE Persistent Disk
- `glusterfs` - Glusterfs volume
- `hostPath` - HostPath volume (for single node testing only; WILL NOT WORK in a multi-node cluster; consider using `local` volume instead)
- `iscsi` - iSCSI (SCSI over IP) storage
- `local` - local storage devices mounted on nodes.
- `nfs` - Network File System (NFS) storage
- `photonPersistentDisk` - Photon controller persistent disk. (This volume type no longer works since the removal of the corresponding cloud provider.)
- `portworxVolume` - Portworx volume
- `quobyte` - Quobyte volume
- `rbd` - Rados Block Device (RBD) volume
- `scaleIO` - ScaleIO volume (**deprecated**)
- `storageos` - StorageOS volume
- `vsphereVolume` - vSphere VMDK volume

https://kubernetes.io/docs/concepts/storage/persistent-volumes/

Persistent Volumes - Example

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-hostpath
 labels:
   type: hostpath
spec:
 capacity:
   storage: 100Gi
 volumeMode: Filesystem
 accessModes:
   - ReadWriteOnce
 hostPath:
   path: /Users/ktb_user/temp/redisconfig
   type: Directory
```

pv-hostpath.yml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-hostpath
 labels:
   type: hostpath
spec:
 capacity:
   storage: 100Gi
 volumeMode: Filesystem
 storageClassName: hostpath
 accessModes:
   - ReadWriteOnce
 hostPath:
   path: /Users/ktb_user/temp/redisconfig
   type: Directory
```

pv-hostpath-storageclass.yml

Phase

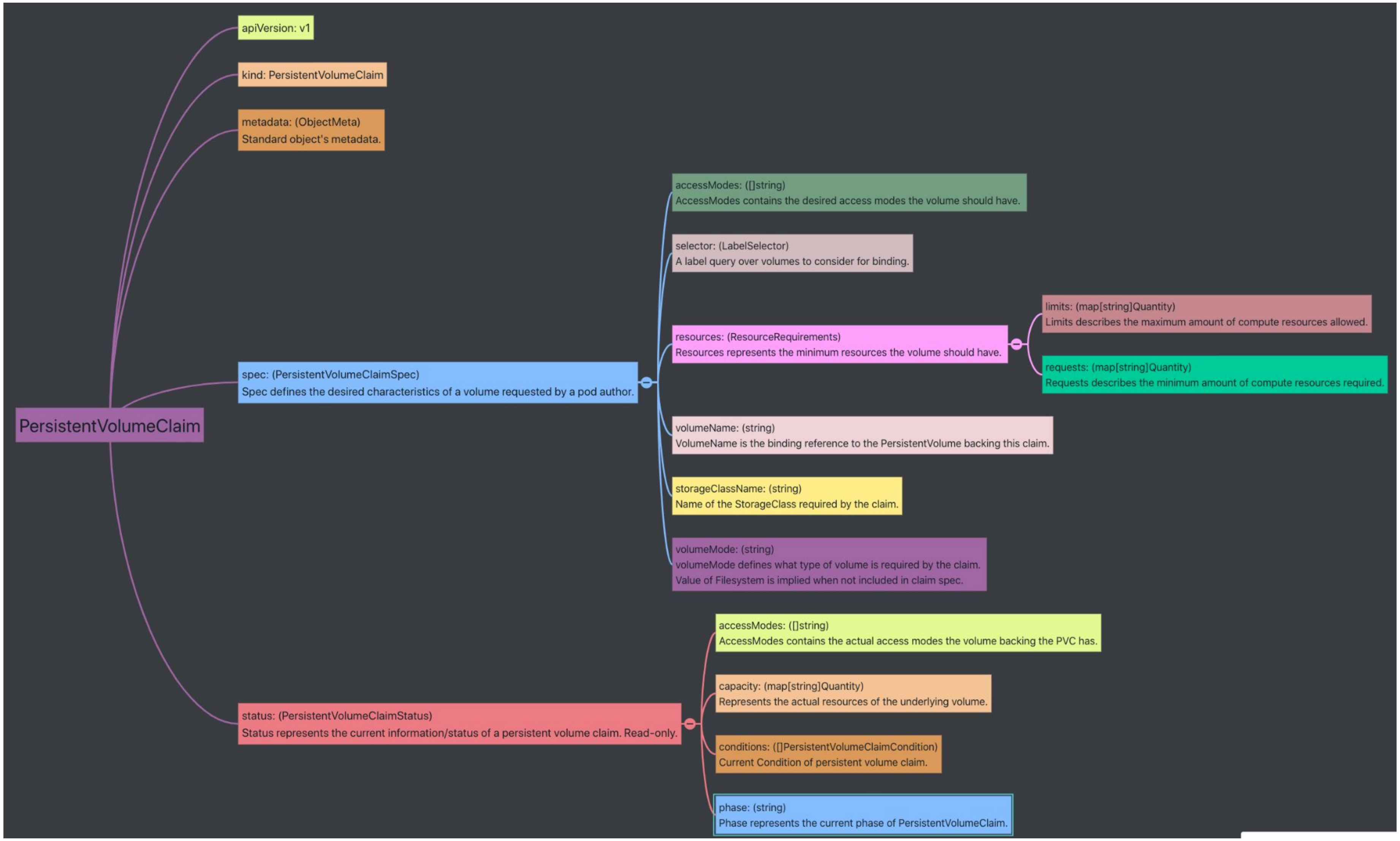
A volume will be in one of the following phases:

- Available -- a free resource that is not yet bound to a claim
- Bound -- the volume is bound to a claim
- Released -- the claim has been deleted, but the resource is not yet reclaimed by the cluster
- Failed -- the volume has failed its automatic reclamation

Persistent Volume Claims

- A PersistentVolumeClaim (PVC) is a request for storage by a user.
- Claims can request specific size and access modes.
- The master watches for new PVCs, finds a matching PV (if possible), and binds them together.
- Pods use claims as volumes.
- Claiming
  - Claims can specify a label selector (matchLabels and matchExpressions) to further filter the set of volumes. Only the volumes whose labels match the selector can be bound to the claim.
  - A claim can request a particular class by specifying the name of a StorageClass using the attribute storageClassName.
  - using volumeName attribute (binding reference).

PersistentVolumeClaim Object


PersistentVolumeClaim
apiVersion: v1
kind: PersistentVolumeClaim
metadata: (ObjectMeta)
Standard object's metadata.
spec: (PersistentVolumeClaimSpec)
Spec defines the desired characteristics of a volume requested by a pod author.
accessModes: ([]string)
AccessModes contains the desired access modes the volume should have.
selector: (LabelSelector)
A label query over volumes to consider for binding.
resources: (ResourceRequirements)
Resources represents the minimum resources the volume should have.
limits: (map[string]Quantity)
Limits describes the maximum amount of compute resources allowed.
requests: (map[string]Quantity)
Requests describes the minimum amount of compute resources required.
volumeName: (string)
VolumeName is the binding reference to the PersistentVolume backing this claim.
storageClassName: (string)
Name of the StorageClass required by the claim.
volumeMode: (string)
volumeMode defines what type of volume is required by the claim.
Value of Filesystem is implied when not included in claim spec.
status: (PersistentVolumeClaimStatus)
Status represents the current information/status of a persistent volume claim. Read-only.
accessModes: ([]string)
AccessModes contains the actual access modes the volume backing the PVC has.
capacity: (map[string]Quantity)
Represents the actual resources of the underlying volume.
conditions: ([]PersistentVolumeClaimCondition)
Current Condition of persistent volume claim.
phase: (string)
Phase represents the current phase of PersistentVolumeClaim.

Persistent Volume Claims - With Volume Name

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-hostpath
 labels:
   type: hostpath
spec:
 capacity:
   storage: 100Gi
 volumeMode: Filesystem
 accessModes:
   - ReadWriteOnce
 hostPath:
   path: /Users/ktb_user/temp/redisconfig
   type: Directory
```

pv-hostpath.yml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc-hostpath
spec:
 resources:
   requests:
     storage: 100Gi
 volumeMode: Filesystem
 accessModes:
   - ReadWriteOnce
 volumeName: pv-hostpath
```

pvc-hostpath.yml

Persistent Volume Claims - With Volume Name

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-hostpath
 labels:
   type: hostpath
spec:
 capacity:
   storage: 100Gi
 volumeMode: Filesystem
 storageClassName: hostpath
 accessModes:
   - ReadWriteOnce
 hostPath:
   path: /Users/ktb_user/temp/redisconfig
   type: Directory
```

pv-hostpath-storageclass.yml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc-hostpath
spec:
 resources:
   requests:
     storage: 100Gi
 volumeMode: Filesystem
 accessModes:
   - ReadWriteOnce
 volumeName: pv-hostpath
```

pvc-hostpath.yml

Persistent Volume Claims - With Selector

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-hostpath
 labels:
   type: hostpath
spec:
 capacity:
   storage: 100Gi
 volumeMode: Filesystem
 storageClassName: hostpath
 accessModes:
   - ReadWriteOnce
 hostPath:
   path: /Users/ktb_user/temp/redisconfig
   type: Directory
```

pv-hostpath-storageclass.yml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc-hostpath-selector
spec:
 resources:
   requests:
     storage: 100Gi
 volumeMode: Filesystem
 accessModes:
   - ReadWriteOnce
 selector:
   matchLabels:
     type: hostpath
```

pvc-hostpath-selector.yml

Using PVC in Pod

```
apiVersion: v1
kind: Pod
metadata:
 name: hostpath-pvc-pod
 labels:
   name: hostpath-pvc-pod
spec:
 terminationGracePeriodSeconds: 0
 containers:
 - name: hostpath-pvc-app
   image: radial/busyboxplus
   imagePullPolicy: IfNotPresent
   volumeMounts:
     - name: config-volume
       mountPath: /config/
     - name: emptydir-volume
       mountPath: /emptydir
   command:
     - "sleep"
     - "3600"
   resources:
     limits:
       memory: "32Mi"
       cpu: "500m"
volumes:
  - name: config-volume
    persistentVolumeClaim:
      claimName: pvc-hostpath
  - name: emptydir-volume
    emptyDir: {}
```

hostpath-pvc-pod.yml

Using ConfigMap and Secret in Spring Boot

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
 name: account-config
data:
 application.yml: |
   spring:
     datasource:
       platform: mariadb
       driverClassName: org.mariadb.jdbc.Driver
       url:
jdbc:mariadb:failover://host.docker.internal:3306/depos
it?autoReconnect=true
       username: ${user}
       password: ${password}
       dbcp2:
         ...
```

account-config.yml

```yaml
apiVersion: v1
data:
 password: cGFzc3dvcmQ=
 user: cm9vdA==
kind: Secret
metadata:
 name: account-secret
type: Opaque
```

account-secret.yml

```yaml
containers:
- image: account:v1.0
  name: account
  env:
    - name: SPRING_CONFIG_LOCATION
      value: /config/
    - name: USERNAME
      valueFrom:
        secretKeyRef:
          key: user
          name: account-secret
    - name: PASSWORD
      valueFrom:
        secretKeyRef:
          key: password
          name: account-secret
  volumeMounts:
    - mountPath: /config/application.yml
      name: app-config
      subPath: application.yml
…

volumes:
  - name: app-config
    configMap:
      name: account-config
  - name: account-secret
    secret:
      secretName: account-secret
```

account-pod-secret.yml

```
kubectl create cm account-config --from-file=application.yml
kubectl create secret generic account-secret --from-literal=password=password --from-literal=user=root
```

End.