Kubernetes KBTG

- Training Day -

Jumpbox®

Days 2

Get Slide

Get Slide (PDF)

Jumpbox®

Tech Passion | Sharing | Society

"เราเชื่อว่าการเรียนรู้ทำให้ชีวิตคุณดีขึ้น"

-Jumpbox Team-

Jumpbox®

Agenda

Day 2

- Configmap (Lab)
- Secret
- Overview private registry
- Volume
- Kustomize (Lab)
- Helm
- Namespace
- RBAC
- Healthcheck (Lab)
- Graceful

- Kubernetes Components
- Kubernetes Custom Resource Definitions
- Resource Management
- Quality of Service for Pods
- Observability
- HPA (Demo)
- Cluster Autoscaler
- Affinity
- Taint & Toleration
- Cert-manager
- Ingress nginx (Demo)

Command and Arguments

- Inject Data Into Applications -

- Command and Arguments
- Environment Variables

Command and Arguments

Command and Arguments (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: command-demo
  labels:
    purpose: demonstrate-command
spec:
  containers:
  - name: command-demo-container
    image: debian
    command: ["printenv"]
    args: ["HOSTNAME", "KUBERNETES_PORT"]
  restartPolicy: OnFailure
```

Reference:

- Command and Argument

Command and Arguments (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: command-demo
  labels:
    purpose: demonstrate-command
spec:
  containers:
  - name: command-demo-container
    image: debian
    command: ["printenv"]
    args: ["HOSTNAME", "KUBERNETES_PORT"]
  restartPolicy: OnFailure
```

Reference:

- Command and Argument

Command and Arguments (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: command-demo
  labels:
    purpose: demonstrate-command
spec:
  containers:
  - name: command-demo-container
    image: debian
    command: ["printenv"]
    args: ["HOSTNAME", "KUBERNETES_PORT"]
  restartPolicy: OnFailure
```

- When you create a Pod, you can define a command and arguments for the containers that run in the Pod.

Reference:

- Command and Argument

Command and Arguments (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: command-demo
  labels:
    purpose: demonstrate-command
spec:
  containers:
  - name: command-demo-container
    image: debian
    command: ["printenv"]
    args: ["HOSTNAME", "KUBERNETES_PORT"]
  restartPolicy: OnFailure
```

- When you create a Pod, you can define a command and arguments for the containers that run in the Pod.

- To define a command, include the command field in the configuration file.

Reference:

- Command and Argument

Command and Arguments (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: command-demo
  labels:
    purpose: demonstrate-command
spec:
  containers:
  - name: command-demo-container
    image: debian
    command: ["printenv"]
    args: ["HOSTNAME", "KUBERNETES_PORT"]
  restartPolicy: OnFailure
```

- When you create a Pod, you can define a command and arguments for the containers that run in the Pod.
- To define a command, include the command field in the configuration file.
- To define arguments for the command, include the args field in the configuration file.

Reference:

- Command and Argument

Command and Arguments (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: command-demo
  labels:
    purpose: demonstrate-command
spec:
  containers:
  - name: command-demo-container
    image: debian
    command: ["printenv"]
    args: ["HOSTNAME", "KUBERNETES_PORT"]
  restartPolicy: OnFailure
```

- When you create a Pod, you can define a command and arguments for the containers that run in the Pod.
- To define a command, include the command field in the configuration file.
- To define arguments for the command, include the args field in the configuration file.
- The command and arguments that you define **cannot be changed after the Pod is created**.

Reference:

- Command and Argument

Command and Arguments (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: command-demo
  labels:
    purpose: demonstrate-command
spec:
  containers:
  - name: command-demo-container
    image: debian
    command: ["printenv"]
    args: ["HOSTNAME", "KUBERNETES_PORT"]
  restartPolicy: OnFailure
```

- When you create a Pod, you can define a command and arguments for the containers that run in the Pod.
- To define a command, include the command field in the configuration file.
- To define arguments for the command, include the args field in the configuration file.
- The command and arguments that you define **cannot be changed after the Pod is created**.

Reference:

- Command and Argument

Environment Variables

- Inject Data Into Applications -

Jumpbox®

Define an environment variable for a container

Define an environment variable for a container (Cont.)

- When you create a Pod, you can set environment variables for the containers that run in the Pod.
- To set environment variables, include the env or envFrom field in the configuration file.

Reference:

- environment variable

Define an environment variable for a container (Cont.)

- When you create a Pod, you can set environment variables for the containers that run in the Pod.
- To set environment variables, include the env or envFrom field in the configuration file.

env:

- allows you to set environment variables for a container, specifying a value directly for each variable that you name.

Reference:

- environment variable

Define an environment variable for a container (Cont.)

- When you create a Pod, you can set environment variables for the containers that run in the Pod.
- To set environment variables, include the env or envFrom field in the configuration file.

env:

- allows you to set environment variables for a container, specifying a value directly for each variable that you name.

envFrom:

- allows you to set environment variables for a container by referencing either a ConfigMap or a Secret.
- When you use envFrom, all the key-value pairs in the referenced ConfigMap or Secret are set as environment variables for the container.
- You can also specify a common prefix string.

Reference:

- environment variable

Define an environment variable for a container (Cont.)

- When you create a Pod, you can set environment variables for the containers that run in the Pod.
- To set environment variables, include the env or envFrom field in the configuration file.

env:

- allows you to set environment variables for a container, specifying a value directly for each variable that you name.

envFrom:

- allows you to set environment variables for a container by referencing either a ConfigMap or a Secret.
- When you use envFrom, all the key-value pairs in the referenced ConfigMap or Secret are set as environment variables for the container.
- You can also specify a common prefix string.

Reference:

- environment variable

Define an environment variable for a container (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
  labels:
    purpose: demonstrate-envars
spec:
  containers:
  - name: envar-demo-container
    image: gcr.io/google-samples/node-hello:1.0
    env:
    - name: DEMO_GREETING
      value: "Hello from the environment"
    - name: DEMO_JUMPBOX
      value: "Jumpbox is Here"
```

Reference:

- environment variable

Define an environment variable for a container (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
  labels:
    purpose: demonstrate-envars
spec:
  containers:
  - name: envar-demo-container
    image: gcr.io/google-samples/node-hello:1.0
    env:
    - name: DEMO_GREETING
      value: "Hello from the environment"
    - name: DEMO_JUMPBOX
      value: "Jumpbox is Here"
```

Pod: envar-demo

Reference:

- environment variable

Define an environment variable for a container (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
  labels:
    purpose: demonstrate-envars
spec:
  containers:
  - name: envar-demo-container
    image: gcr.io/google-samples/node-hello:1.0
    env:
    - name: DEMO_GREETING
      value: "Hello from the environment"
    - name: DEMO_JUMPBOX
      value: "Jumpbox is Here"
```

Reference:

- environment variable

Define an environment variable for a container (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
  labels:
    purpose: demonstrate-envars
spec:
  containers:
  - name: envar-demo-container
    image: gcr.io/google-samples/node-hello:1.0
    env:
    - name: DEMO_GREETING
      value: "Hello from the environment"
    - name: DEMO_JUMPBOX
      value: "Jumpbox is Here"
```

Reference:

- environment variable

Define an environment variable for a container (Cont.)

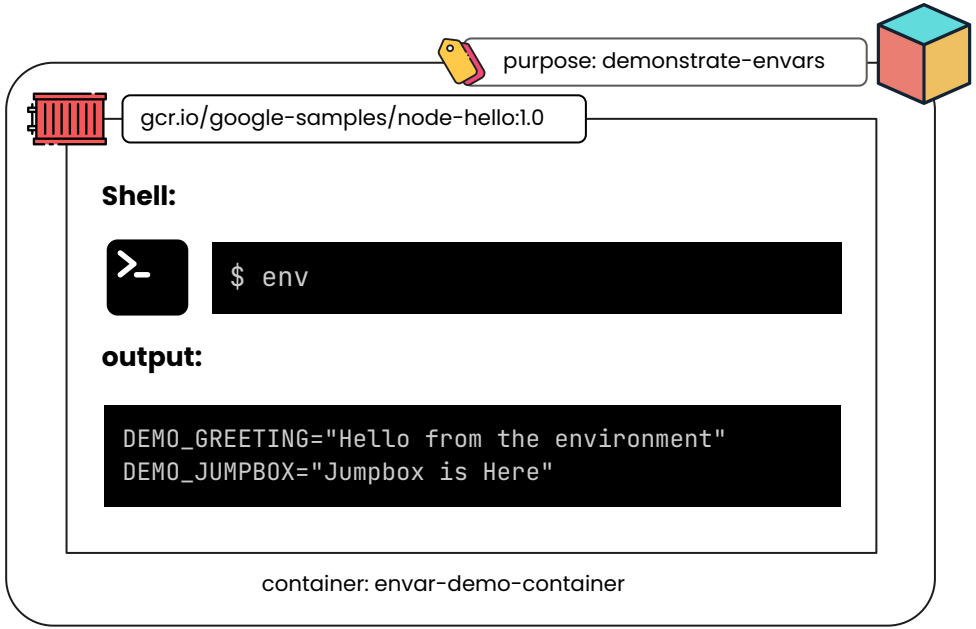
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
  labels:
    purpose: demonstrate-envars
spec:
  containers:
  - name: envar-demo-container
    image: gcr.io/google-samples/node-hello:1.0
    env:
    - name: DEMO_GREETING
      value: "Hello from the environment"
    - name: DEMO_JUMPBOX
      value: "Jumpbox is Here"
```

Reference:

- environment variable

Define an environment variable for a container (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
  labels:
    purpose: demonstrate-envars
spec:
  containers:
  - name: envar-demo-container
    image: gcr.io/google-samples/node-hello:1.0
    env:
    - name: DEMO_GREETING
      value: "Hello from the environment"
    - name: DEMO_JUMPBOX
      value: "Jumpbox is Here"
```



Pod: envar-demo

Reference:

- environment variable

Define an dependent variable for a container

Define an dependent variable for a container (Cont.)

```yaml
spec:
  containers:
  ...
    env:
    - name: SERVICE_PORT
      value: "80"
    - name: SERVICE_IP
      value: "172.17.0.1"
    - name: UNCHANGED_REFERENCE
      value: "$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
    - name: PROTOCOL
      value: "https"
    - name: SERVICE_ADDRESS
      value: "$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
    - name: ESCAPED_REFERENCE
      value: "$$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
```

Reference:

- dependent variable

Define an dependent variable for a container (Cont.)

```yaml
spec:
  containers:
  ...
    env:
    - name: SERVICE_PORT
      value: "80"
    - name: SERVICE_IP
      value: "172.17.0.1"
    - name: UNCHANGED_REFERENCE
      value: "$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
    - name: PROTOCOL
      value: "https"
    - name: SERVICE_ADDRESS
      value: "$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
    - name: ESCAPED_REFERENCE
      value: "$$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
```

Reference:

- dependent variable

Define an dependent variable for a container (Cont.)

```yaml
spec:
  containers:
  ...
    env:
    - name: SERVICE_PORT
      value: "80"
    - name: SERVICE_IP
      value: "172.17.0.1"
    - name: UNCHANGED_REFERENCE
      value: "$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
    - name: PROTOCOL
      value: "https"
    - name: SERVICE_ADDRESS
      value: "$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
    - name: ESCAPED_REFERENCE
      value: "$$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
```

Reference:

- dependent variable

Define an dependent variable for a container (Cont.)

```yaml
spec:
  containers:
  ...
    env:
    - name: SERVICE_PORT
      value: "80"
    - name: SERVICE_IP
      value: "172.17.0.1"
    - name: UNCHANGED_REFERENCE
      value: "$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
    - name: PROTOCOL
      value: "https"
    - name: SERVICE_ADDRESS
      value: "$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
    - name: ESCAPED_REFERENCE
      value: "$$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
```

Shell:

```
$ echo
"$UNCHANGED_REFERENCE\\n$SERVICE_ADDRESS\\n$ESCA
PED_REFERENCE"
```

Reference:

- dependent variable

Define an dependent variable for a container (Cont.)

```yaml
spec:
  containers:
  ...
    env:
    - name: SERVICE_PORT
      value: "80"
    - name: SERVICE_IP
      value: "172.17.0.1"
    - name: UNCHANGED_REFERENCE
      value: "$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
    - name: PROTOCOL
      value: "https"
    - name: SERVICE_ADDRESS
      value: "$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
    - name: ESCAPED_REFERENCE
      value: "$$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
```

Shell:

```
$ echo
"$UNCHANGED_REFERENCE\\n$SERVICE_ADDRESS\\n$ESCA
PED_REFERENCE"
```

output:

```
UNCHANGED_REFERENCE=$(PROTOCOL)://172.17.0.1:80
SERVICE_ADDRESS=https://172.17.0.1:80
ESCAPED_REFERENCE=$(PROTOCOL)://172.17.0.1:80
```

Reference:

- dependent variable

Pod fields as values for environment variables

Pod fields as values for environment variables (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  namespace: kai
  name: envar-demo
spec:
  containers:
  - env:
    - name: MY_POD_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: MY_POD_NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
```

Reference:

- Expose Pod Information

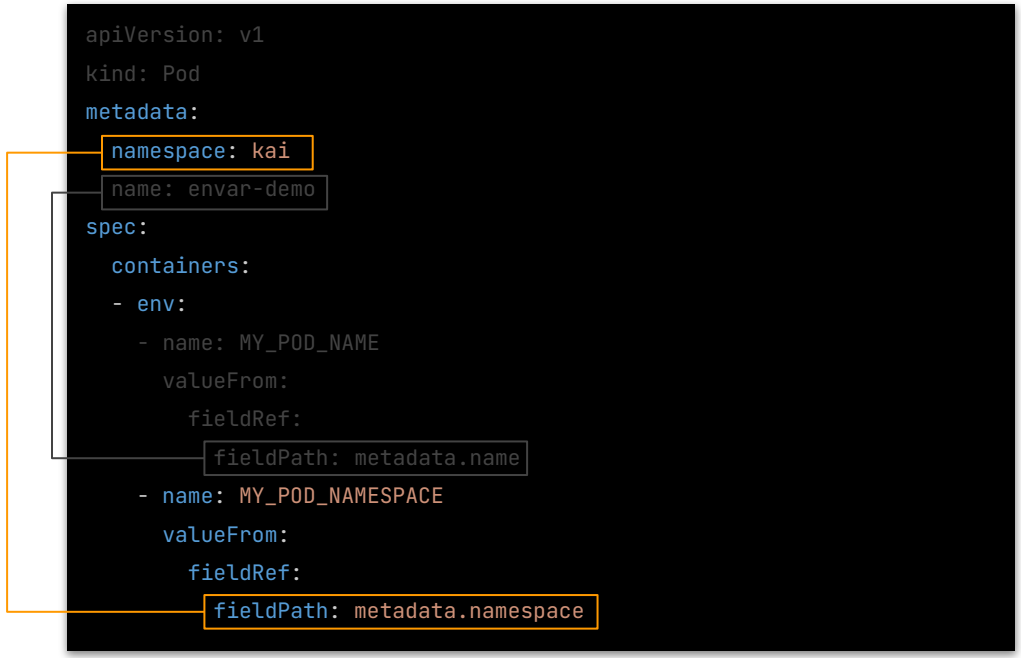
Pod fields as values for environment variables (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  namespace: kai
  name: envar-demo
spec:
  containers:
  - env:
    - name: MY_POD_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: MY_POD_NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
```

Reference:

- Expose Pod Information

Pod fields as values for environment variables (Cont.)



Reference:

- Expose Pod Information

Pod fields as values for environment variables (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  namespace: kai
  name: envar-demo
spec:
  containers:
  - env:
    - name: MY_POD_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: MY_POD_NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
```

Shell:

```
$ echo  "$MY_POD_NAME\\n$MY_POD_NAMESPACE
```

Reference:

- Expose Pod Information

Pod fields as values for environment variables (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  namespace: kai
  name: envar-demo
spec:
  containers:
  - env:
    - name: MY_POD_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: MY_POD_NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
```

Shell:

```
$ echo  "$MY_POD_NAME\\n$MY_POD_NAMESPACE
```

output:

```
MY_POD_NAME=envar-demo
MY_POD_NAMESPACE=kai
```

Reference:

- Expose Pod Information

Environment

With Real Action Simulator

[Click Here]

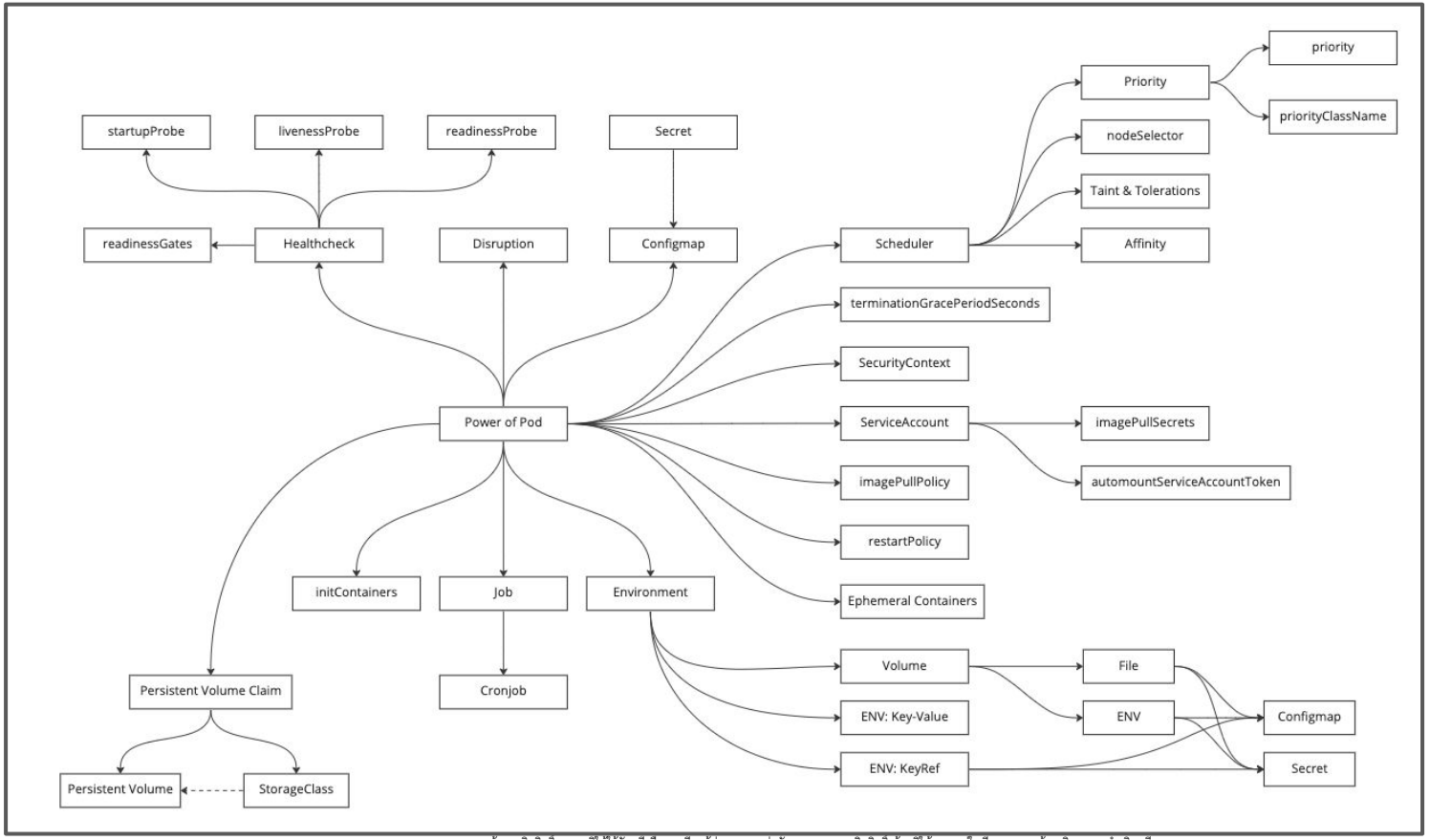
ConfigMap

- Working with Kubernetes -

- ENV
- ENV: KeyRef
- Volume

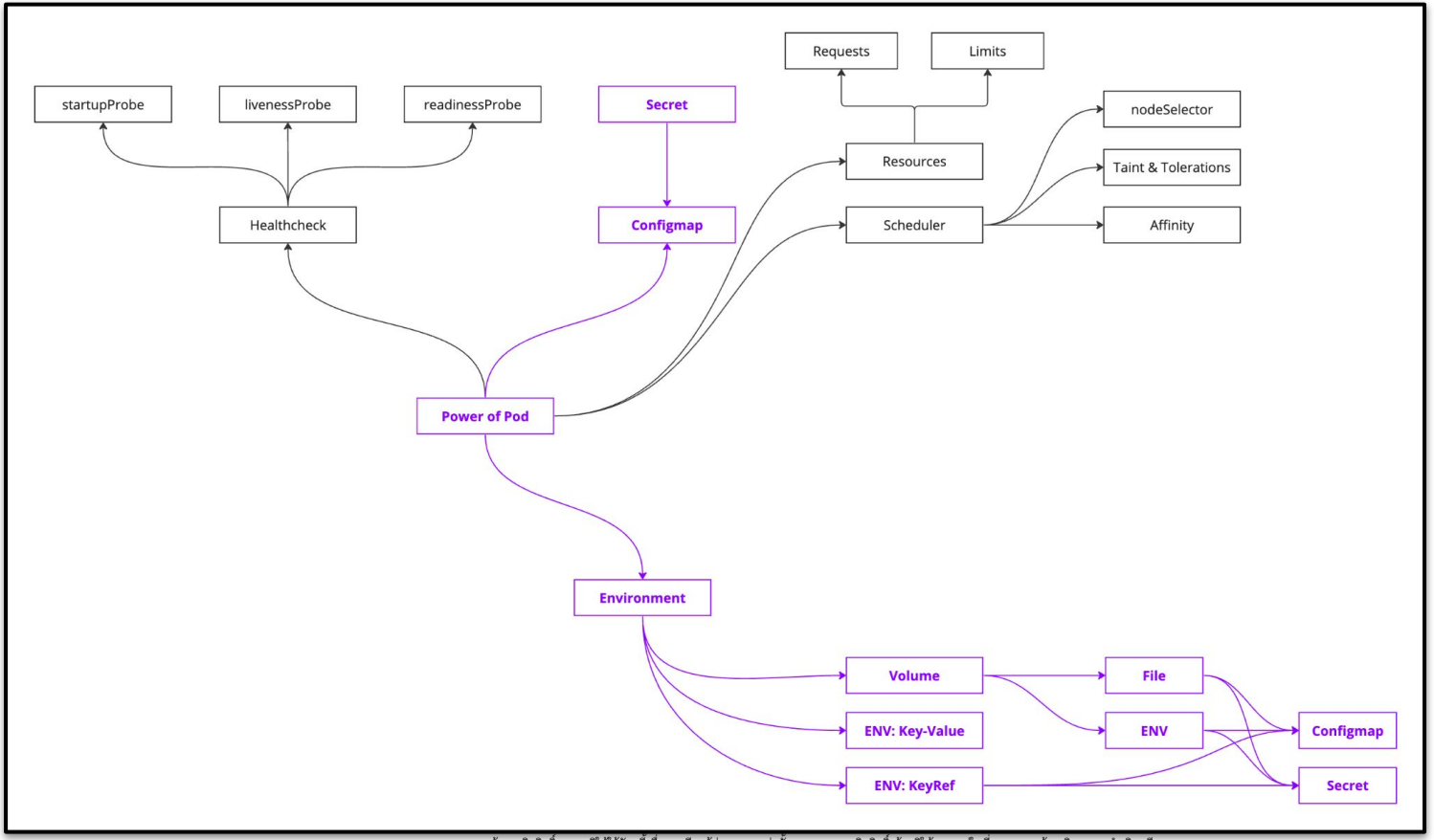
Jumpbox®

Power of Pod Mapping



Jumpbox®

Power of Pod Mapping



Jumpbox®

Configure a Pod to Use a ConfigMap

Many applications rely on configuration which is used during either application initialization or runtime. Most times, there is a requirement to adjust values assigned to configuration parameters. ConfigMaps are a Kubernetes mechanism that let you inject configuration data into application pods.

The ConfigMap concept allow you to decouple configuration artifacts from image content to keep containerized applications portable. For example, you can download and run the same container image to spin up containers for the purposes of local development, system test, or running a live end-user workload.

This page provides a series of usage examples demonstrating how to create ConfigMaps and configure Pods using data stored in ConfigMaps.

Reference Pictures:

- Configure a Pod to Use a ConfigMap

ConfigMaps (Cont.)

- Kubernetes objects for inject in containers with configuration data
- Configuration data can be consumed in pods in a variety of ways
- ConfigMap can be used to:
  - Populate the value of environment variables.
  - Set command-line arguments in a container.
  - Populate configuration files in a volume.

Reference Pictures:

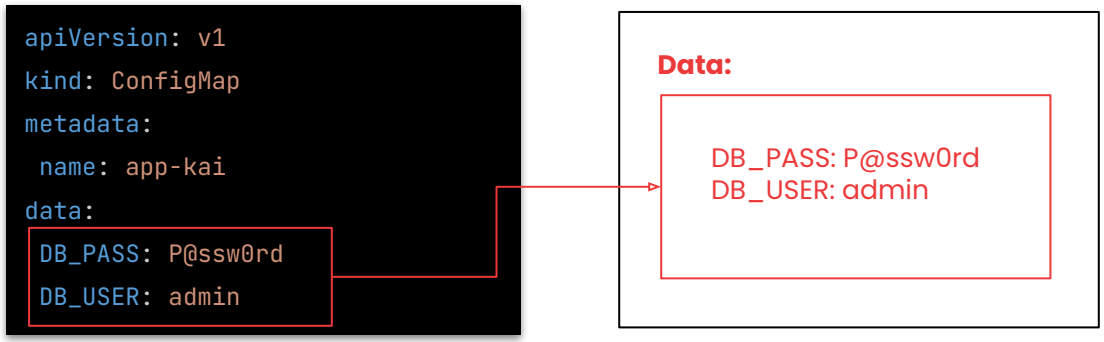
- Kubernetes Webinar - Using ConfigMaps & Secrets

ConfigMaps Structure

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
 name: app-kai
data:
 DB_PASS: P@ssw0rd
 DB_USER: admin
```

ConfigMaps Structure (Cont.)

- Allow you to decouple configuration from image content to keep containerized applications portable

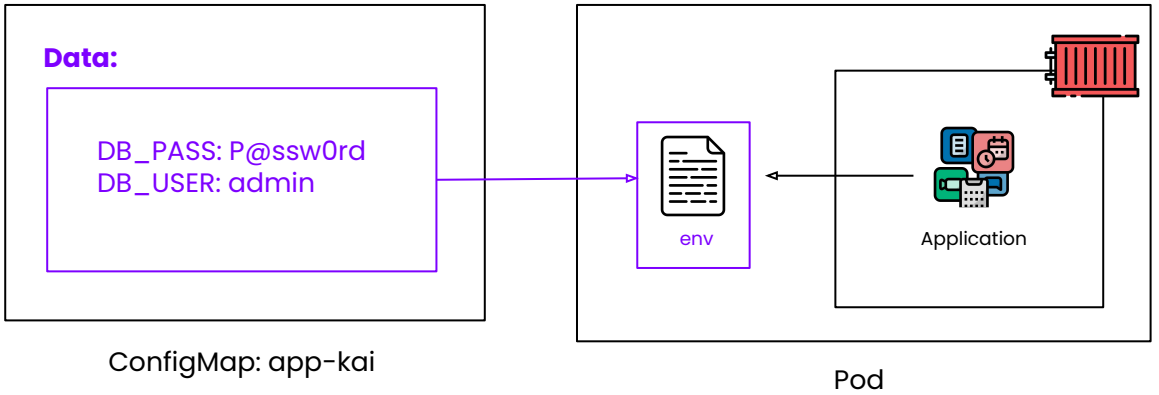


ConfigMap: app-kai

ConfigMaps Structure (Cont.)

- Allow you to decouple configuration from image content to keep containerized applications portable
- It is used to setting configuration data separately from application code

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
 name: app-kai
data:
 DB_PASS: P@ssw0rd
 DB_USER: admin
```

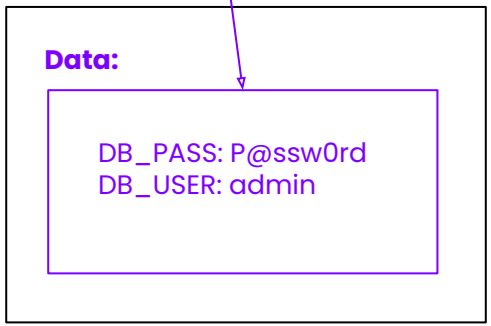


ConfigMaps Structure (Cont.)

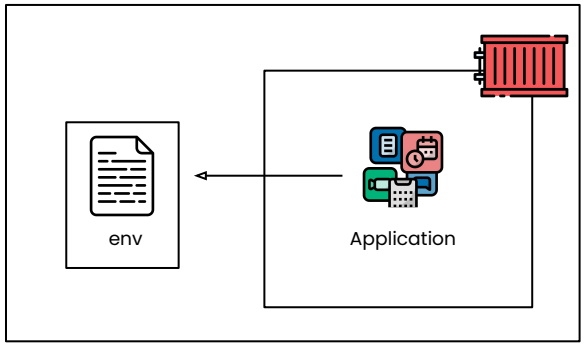
- Allow you to decouple configuration from image content to keep containerized applications portable
- It is used to setting configuration data separately from application code
- It is like .env file which can vary on environments (dev, uat, and production)

.env, application.yaml,
appsettings.json

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
 name: app-kai
data:
 DB_PASS: P@ssw0rd
 DB_USER: admin
```



ConfigMap: app-kai

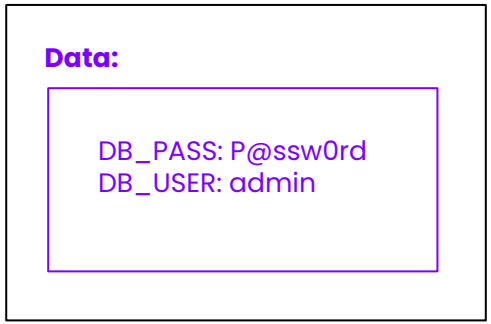


Pod

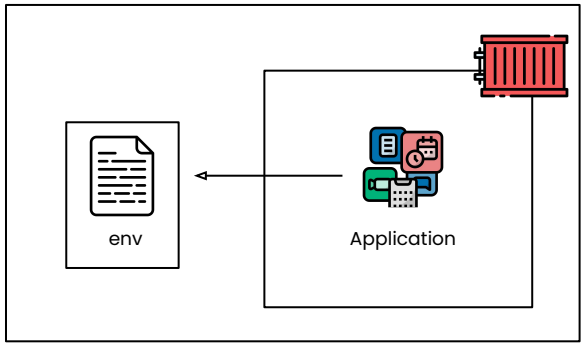
ConfigMaps Structure (Cont.)

- Allow you to decouple configuration from image content to keep containerized applications portable
- It is used to setting configuration data separately from application code
- It is like .env file which can vary on environments (dev, uat, and production)
- Does not support large chunks of data.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
 name: app-kai
data:
 DB_PASS: P@ssw0rd
 DB_USER: admin
```



ConfigMap: app-kai

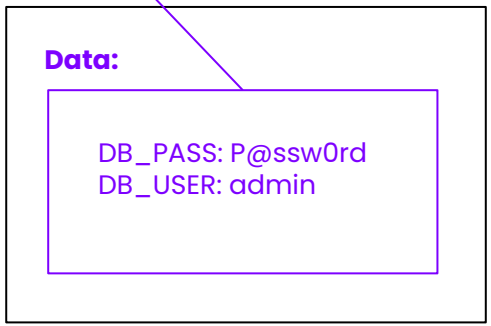


Pod

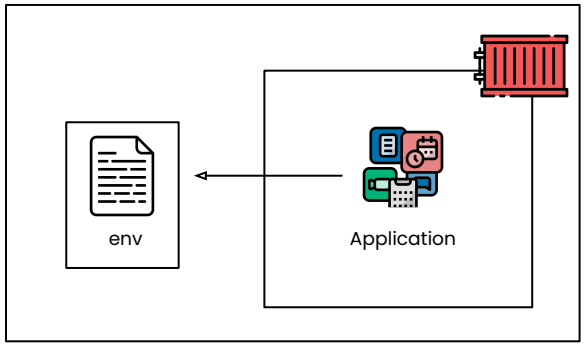
ConfigMaps Structure (Cont.)

- Allow you to decouple configuration from image content to keep containerized applications portable
- It is used to setting configuration data separately from application code
- It is like .env file which can vary on environments (dev, uat, and production)
- Does not support large chunks of data.
- The size of configmap cannot exceed 1MB.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
 name: app-kai
data:
 DB_PASS: P@ssw0rd
 DB_USER: admin
```



ConfigMap: app-kai

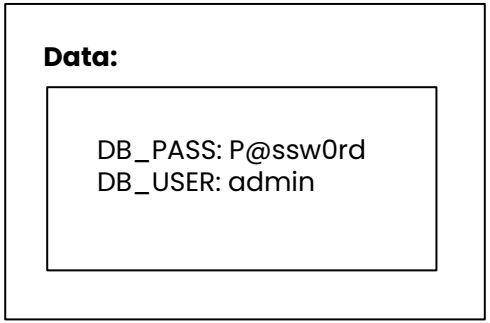


Pod

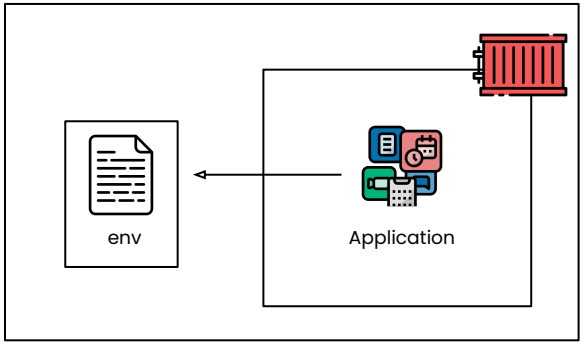
ConfigMaps Structure (Cont.)

- Allow you to decouple configuration from image content to keep containerized applications portable
- It is used to setting configuration data separately from application code
- It is like .env file which can vary on environments (dev, uat, and production)
- Does not support large chunks of data.
- The size of configmap cannot exceed 1MB.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
 name: app-kai
data:
 DB_PASS: P@ssw0rd
 DB_USER: admin
```



ConfigMap: app-kai



Pod

Environment Variable

- Configure a Pod to Use a ConfigMap -

Jumpbox®

Configure a Pod to Use a ConfigMap - Environment Variable

Configure a Pod to Use a ConfigMap - Environment Variable (Cont.)

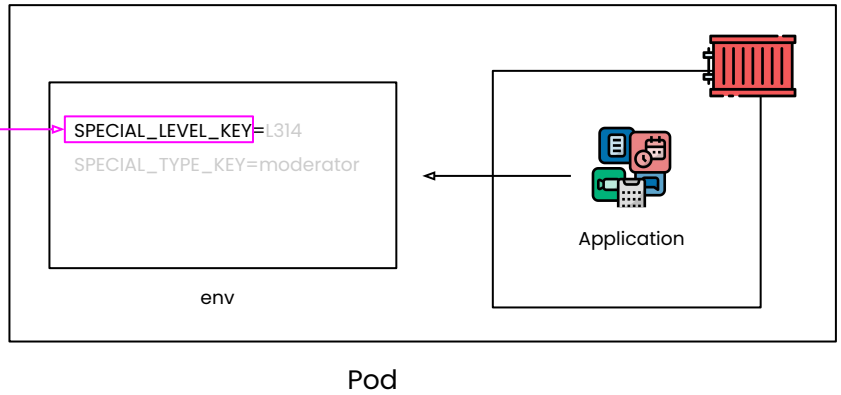
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: registry.k8s.io/busybox
    command: [ "/bin/echo", "$(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)" ]
    env:
    - name: SPECIAL_LEVEL_KEY
      valueFrom:
        configMapKeyRef:
          name: special-config
          key: SPECIAL_LEVEL
    - name: SPECIAL_TYPE_KEY
      valueFrom:
        configMapKeyRef:
          name: special-type
          key: SPECIAL_TYPE
restartPolicy: Never
```

Configure a Pod to Use a ConfigMap - Environment Variable (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: registry.k8s.io/busybox
    command: [ "/bin/echo", "$(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)" ]
    env:
    - name: SPECIAL_LEVEL_KEY
      valueFrom:
        configMapKeyRef:
          name: special-config
          key: SPECIAL_LEVEL
    - name: SPECIAL_TYPE_KEY
      valueFrom:
        configMapKeyRef:
          name: special-type
          key: SPECIAL_TYPE
restartPolicy: Never
```

Configure a Pod to Use a ConfigMap - Environment Variable (Cont.)

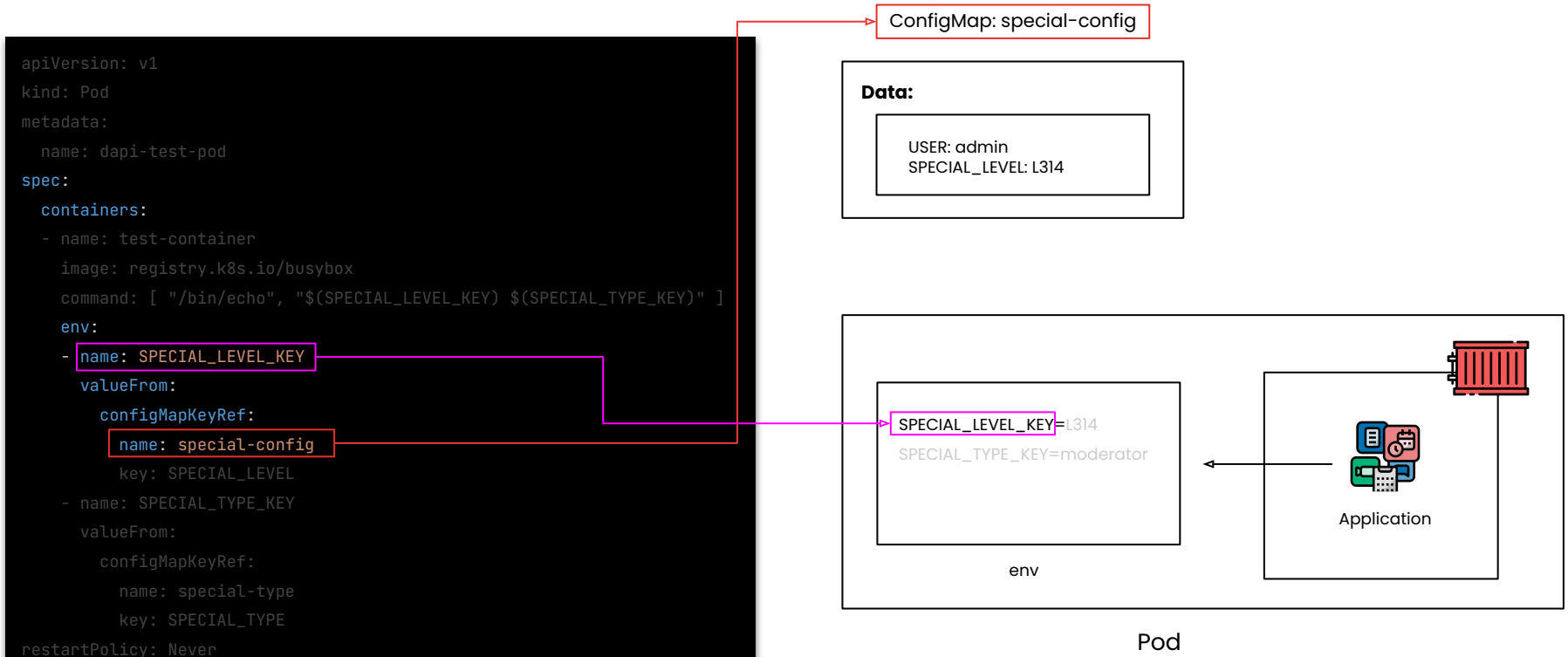
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: registry.k8s.io/busybox
    command: [ "/bin/echo", "$(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)" ]
    env:
    - name: SPECIAL_LEVEL_KEY
      valueFrom:
        configMapKeyRef:
          name: special-config
          key: SPECIAL_LEVEL
    - name: SPECIAL_TYPE_KEY
      valueFrom:
        configMapKeyRef:
          name: special-type
          key: SPECIAL_TYPE
restartPolicy: Never
```



Jumpbox®

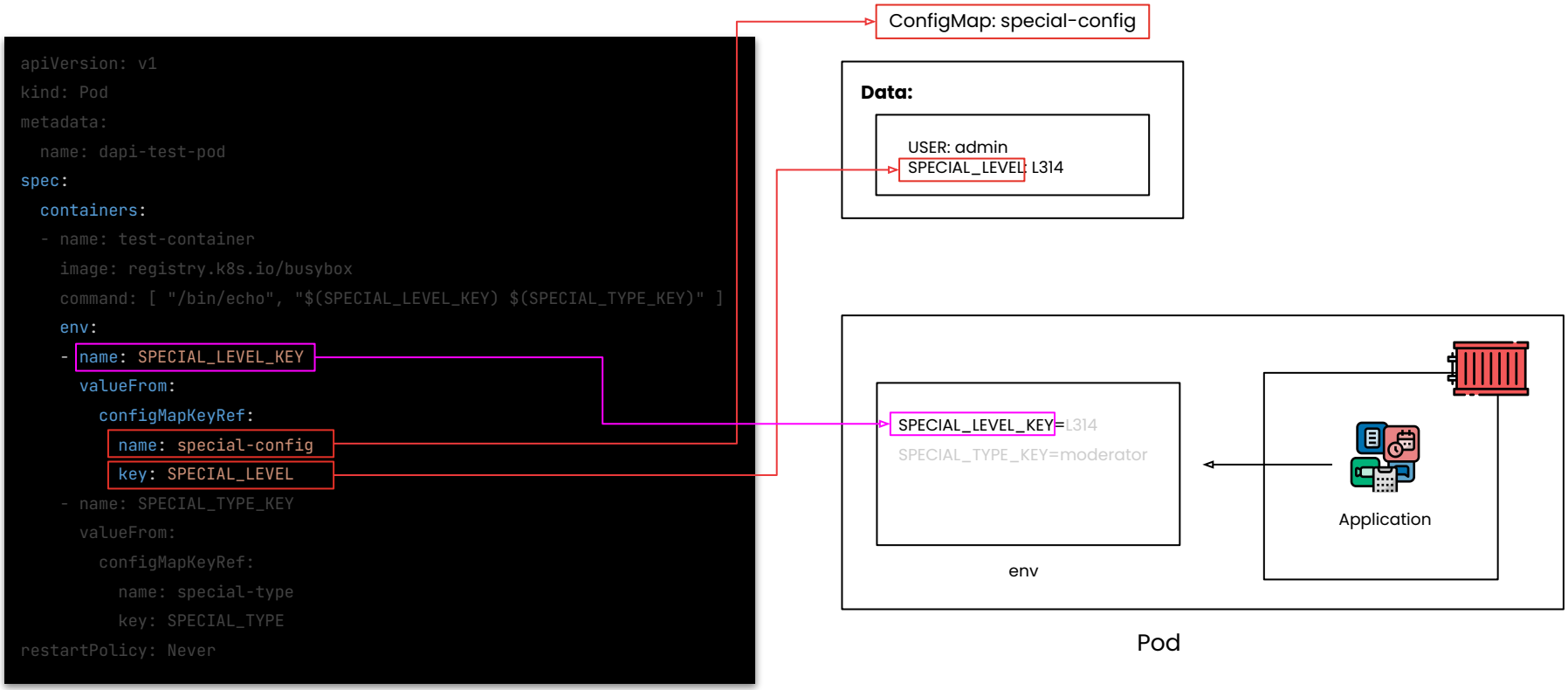
Configure a Pod to Use a ConfigMap - Environment Variable (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: registry.k8s.io/busybox
    command: [ "/bin/echo", "$(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)" ]
    env:
    - name: SPECIAL_LEVEL_KEY
      valueFrom:
        configMapKeyRef:
          name: special-config
          key: SPECIAL_LEVEL
    - name: SPECIAL_TYPE_KEY
      valueFrom:
        configMapKeyRef:
          name: special-type
          key: SPECIAL_TYPE
restartPolicy: Never
```



Configure a Pod to Use a ConfigMap - Environment Variable (Cont.)

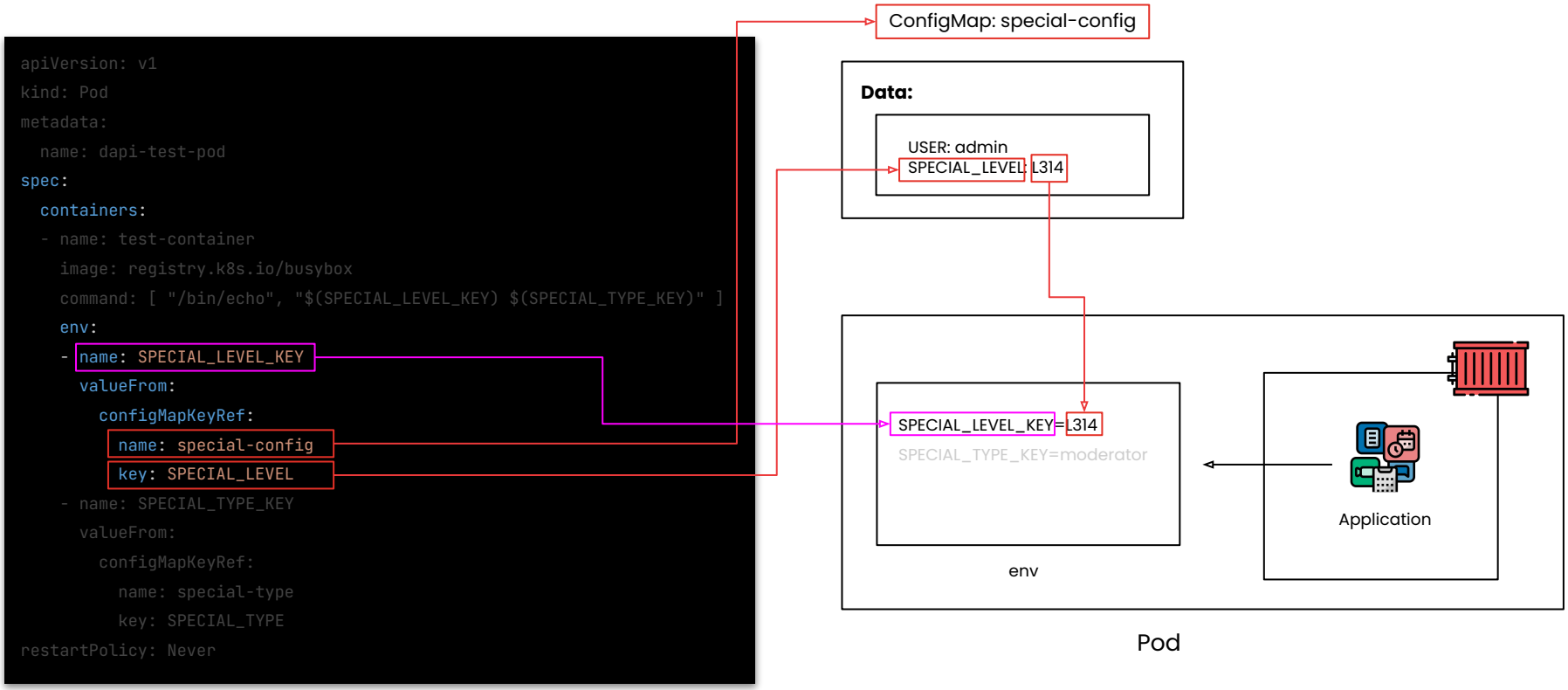
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: registry.k8s.io/busybox
    command: [ "/bin/echo", "$(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)" ]
    env:
    - name: SPECIAL_LEVEL_KEY
      valueFrom:
        configMapKeyRef:
          name: special-config
          key: SPECIAL_LEVEL
    - name: SPECIAL_TYPE_KEY
      valueFrom:
        configMapKeyRef:
          name: special-type
          key: SPECIAL_TYPE
restartPolicy: Never
```



Jumpbox®

Configure a Pod to Use a ConfigMap - Environment Variable (Cont.)

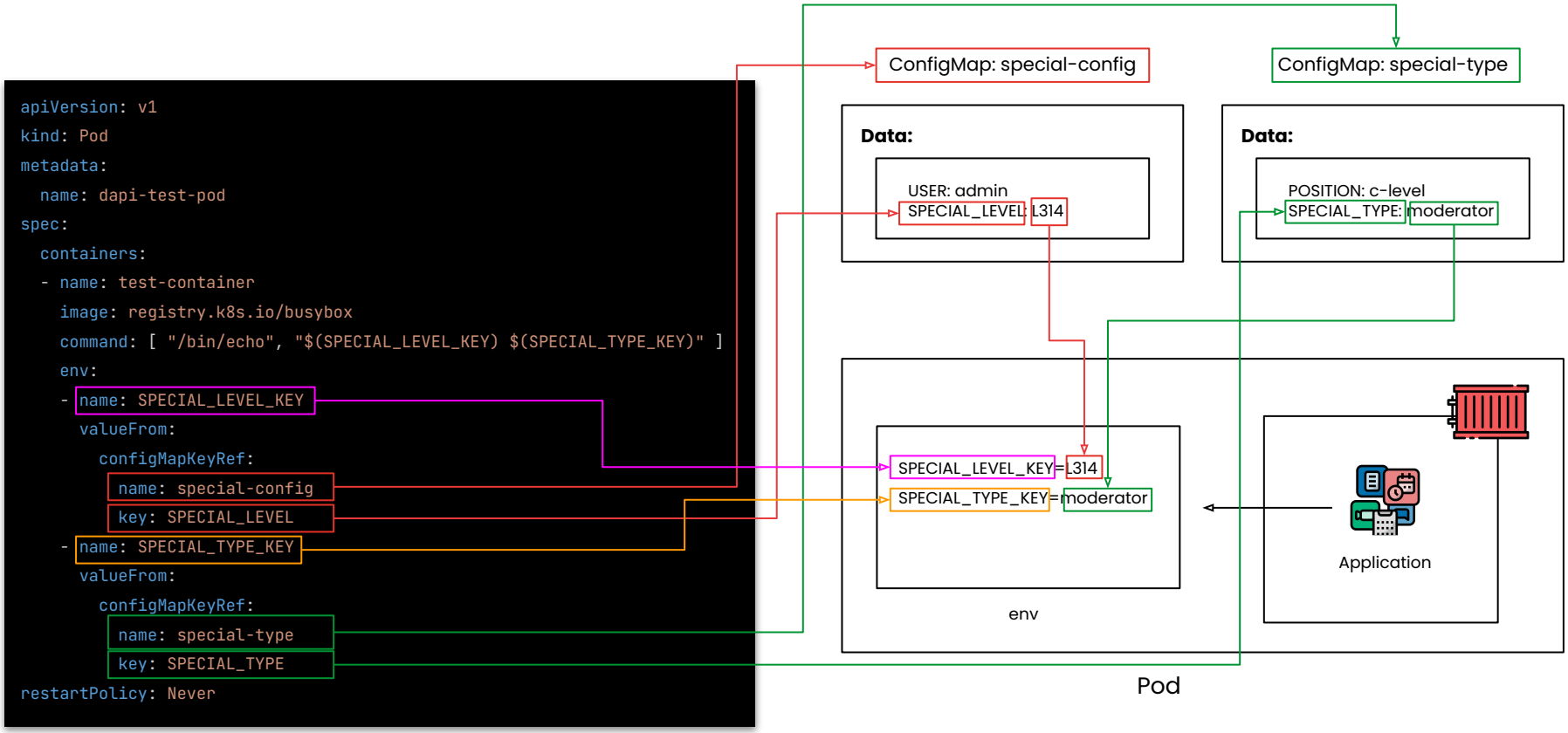
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: registry.k8s.io/busybox
    command: [ "/bin/echo", "$(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)" ]
    env:
    - name: SPECIAL_LEVEL_KEY
      valueFrom:
        configMapKeyRef:
          name: special-config
          key: SPECIAL_LEVEL
    - name: SPECIAL_TYPE_KEY
      valueFrom:
        configMapKeyRef:
          name: special-type
          key: SPECIAL_TYPE
restartPolicy: Never
```



Jumpbox®

Configure a Pod to Use a ConfigMap - Environment Variable (Cont.)

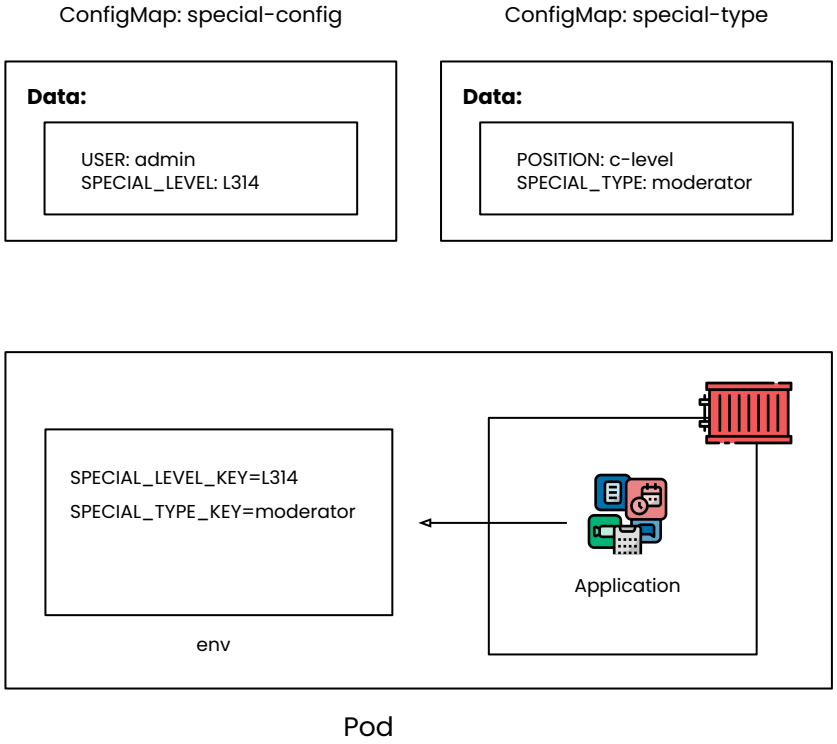
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: registry.k8s.io/busybox
    command: [ "/bin/echo", "$(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)" ]
    env:
    - name: SPECIAL_LEVEL_KEY
      valueFrom:
        configMapKeyRef:
          name: special-config
          key: SPECIAL_LEVEL
    - name: SPECIAL_TYPE_KEY
      valueFrom:
        configMapKeyRef:
          name: special-type
          key: SPECIAL_TYPE
restartPolicy: Never
```



Jumpbox®

Configure a Pod to Use a ConfigMap - Environment Variable (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: registry.k8s.io/busybox
    command: [ "/bin/echo", "$(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)" ]
    env:
    - name: SPECIAL_LEVEL_KEY
      valueFrom:
        configMapKeyRef:
          name: special-config
          key: SPECIAL_LEVEL
    - name: SPECIAL_TYPE_KEY
      valueFrom:
        configMapKeyRef:
          name: special-type
          key: SPECIAL_TYPE
restartPolicy: Never
```



All key-value as Environment

- Configure a Pod to Use a ConfigMap -

Jumpbox®

Configure a Pod to Use a ConfigMap - All key-value as Environment

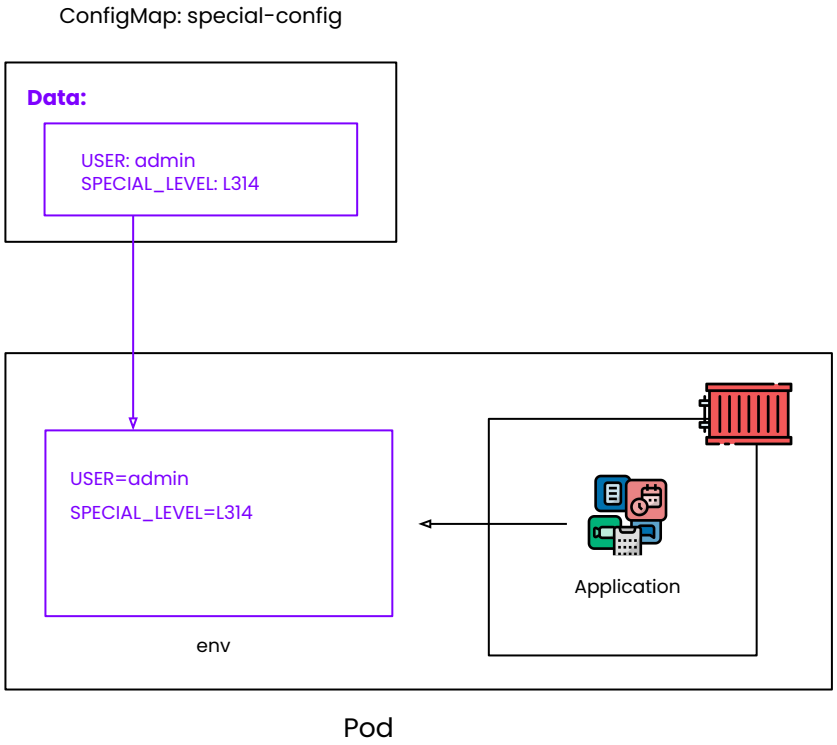
Configure a Pod to Use a ConfigMap - All key-value as Environment (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: registry.k8s.io/busybox
    command: [ "/bin/sh", "-c", "env" ]
    envFrom:
    - configMapRef:
        name: special-config
restartPolicy: Never
```

Pod

Configure a Pod to Use a ConfigMap - All key-value as Environment (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: registry.k8s.io/busybox
    command: [ "/bin/sh", "-c", "env" ]
    envFrom:
    - configMapRef:
        name: special-config
restartPolicy: Never
```



All key-value as file(s)

- Configure a Pod to Use a ConfigMap -

Configure a Pod to Use a ConfigMap - All key-value as file(s)

Configure a Pod to Use a ConfigMap - All key-value as file(s) (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: registry.k8s.io/busybox
    command: [ "/bin/sh", "-c", "ls /etc/config/" ]
    volumeMounts:
    - name: config-volume
      mountPath: /etc/config
  volumes:
  - name: config-volume
    configMap:
      name: special-config
  restartPolicy: Never
```

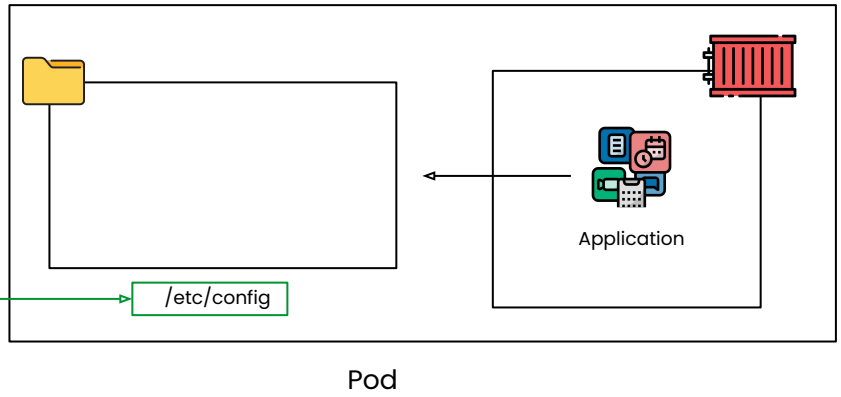
Jumpbox®

Configure a Pod to Use a ConfigMap - All key-value as file(s) (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: registry.k8s.io/busybox
    command: [ "/bin/sh", "-c", "ls /etc/config/" ]
    volumeMounts:
    - name: config-volume
      mountPath: /etc/config
  volumes:
  - name: config-volume
    configMap:
      name: special-config
  restartPolicy: Never
```

Configure a Pod to Use a ConfigMap - All key-value as file(s) (Cont.)

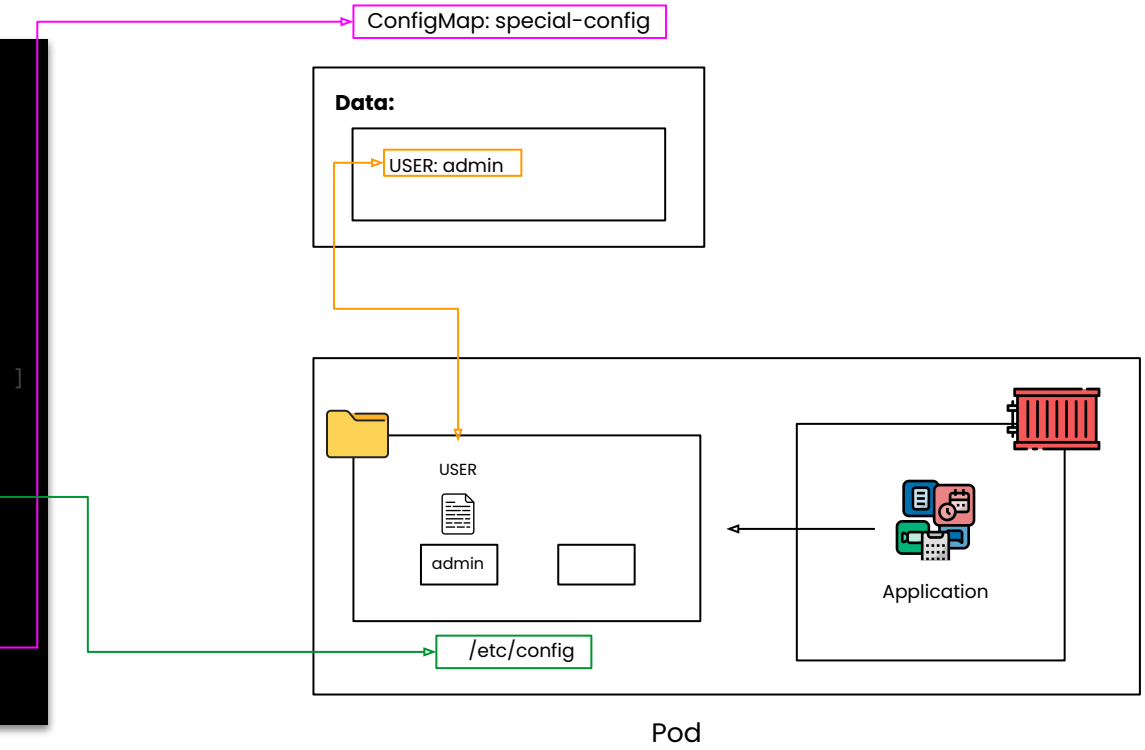
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: registry.k8s.io/busybox
    command: [ "/bin/sh", "-c", "ls /etc/config/" ]
    volumeMounts:
    - name: config-volume
      mountPath: /etc/config
  volumes:
  - name: config-volume
    configMap:
      name: special-config
  restartPolicy: Never
```



Jumpbox®

Configure a Pod to Use a ConfigMap - All key-value as file(s) (Cont.)

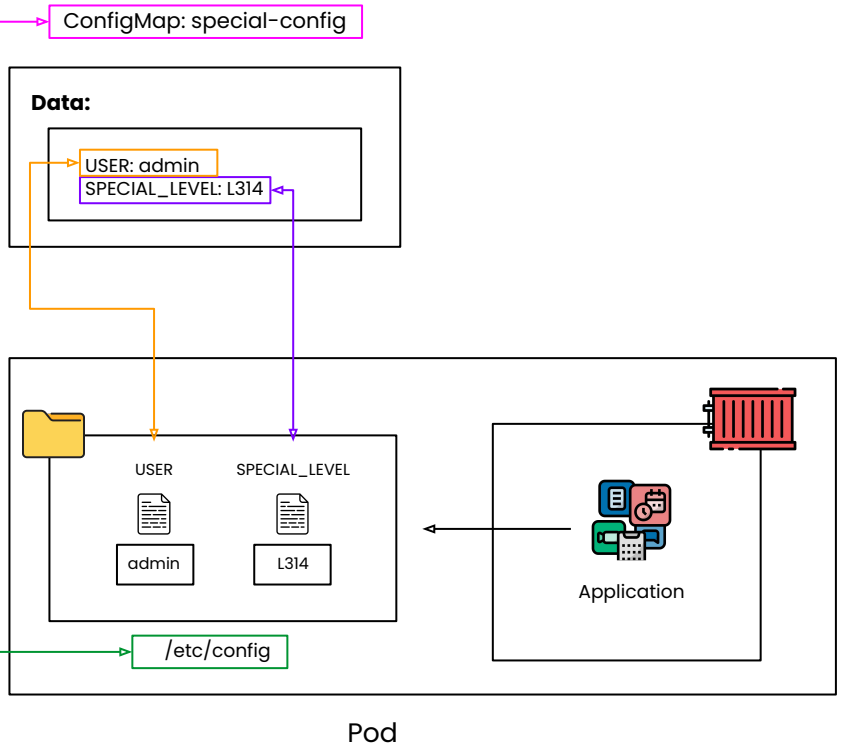
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: registry.k8s.io/busybox
    command: [ "/bin/sh", "-c", "ls /etc/config/" ]
    volumeMounts:
    - name: config-volume
      mountPath: /etc/config
  volumes:
  - name: config-volume
    configMap:
      name: special-config
  restartPolicy: Never
```



Jumpbox®

Configure a Pod to Use a ConfigMap - All key-value as file(s) (Cont.)

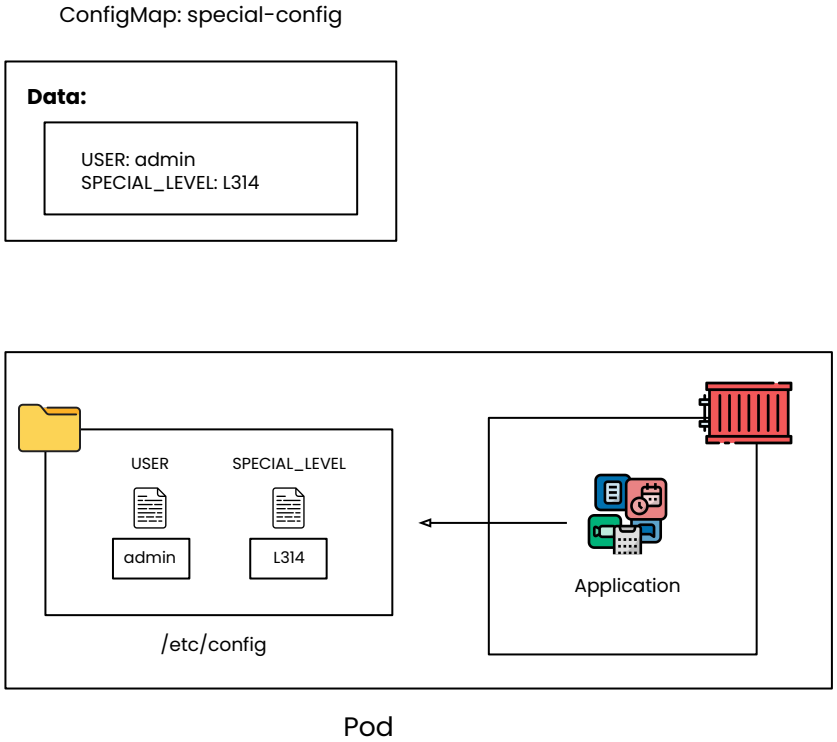
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: registry.k8s.io/busybox
    command: [ "/bin/sh", "-c", "ls /etc/config/" ]
    volumeMounts:
    - name: config-volume
      mountPath: /etc/config
  volumes:
  - name: config-volume
    configMap:
      name: special-config
  restartPolicy: Never
```



Jumpbox®

Configure a Pod to Use a ConfigMap - All key-value as file(s) (Cont.)

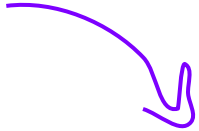
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: registry.k8s.io/busybox
    command: [ "/bin/sh", "-c", "ls /etc/config/" ]
    volumeMounts:
    - name: config-volume
      mountPath: /etc/config
  volumes:
  - name: config-volume
    configMap:
      name: special-config
  restartPolicy: Never
```



Jumpbox®

Action Dojo

- Kata: small exercise to do by yourself



Activity Time !!

- Kata -

Configmap

With Real Action Simulator

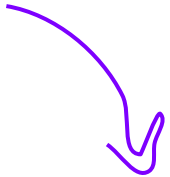
[Click Here]

Action Dojo

- Kata: small exercise to do by yourself

Activity Time !!

- kata -



Then Break !!

- 10 minutes -

Jumpbox®

Secret

- Working with Kubernetes -

- ENV
- ENV: KeyRef
- Volume

Jumpbox®

Secret

Jumpbox®

Secret (Cont.)

Uses for Secrets

You can use Secrets for purposes such as the following:

- Set environment variables for a container.
- Provide credentials such as SSH keys or passwords to Pods.
- Allow the kubelet to pull container images from private registries.

The Kubernetes control plane also uses Secrets; for example, bootstrap token Secrets are a mechanism to help automate node registration.

Reference Pictures:
- Secrets

Secret (Cont.)

- Kubernetes secret objects let you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys.

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysql
data:
  username: YWRtaW4=
  password: UEBzc3cwcmQ=
```

Secret (Cont.)

- Kubernetes secret objects let you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys.
- Secrets are similar to ConfigMaps but are specifically intended to hold confidential data.

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysql
data:
  username: YWRtaW4=
  password: UEBzc3cwcmQ=
```

Secret (Cont.)

> **Caution:**
> Kubernetes Secrets are, by default, stored unencrypted in the API server's underlying data store (etcd). Anyone with API access can retrieve or modify a Secret, and so can anyone with access to etcd. Additionally, anyone who is authorized to create a Pod in a namespace can use that access to read any Secret in that namespace; this includes indirect access such as the ability to create a Deployment.

Secret (Cont.)

> **Caution:**
> Kubernetes Secrets are, by default, stored unencrypted in the API server's underlying data store (etcd). Anyone with API access can retrieve or modify a Secret, and so can anyone with access to etcd. Additionally, anyone who is authorized to create a Pod in a namespace can use that access to read any Secret in that namespace; this includes indirect access such as the ability to create a Deployment.

In order to safely use Secrets, take at least the following steps:

1. Enable Encryption at Rest for Secrets.
2. Enable or configure RBAC rules with least-privilege access to Secrets.
3. Restrict Secret access to specific containers.
4. Consider using external Secret store providers.

For more guidelines to manage and improve the security of your Secrets, refer to Good practices for Kubernetes Secrets.

Environment Variable

- Configure a Pod to Use a Secret -

Jumpbox®

Configure a Pod to Use a Secret - Environment Variable

Configure a Pod to Use a Secret - Environment Variable (Cont.)

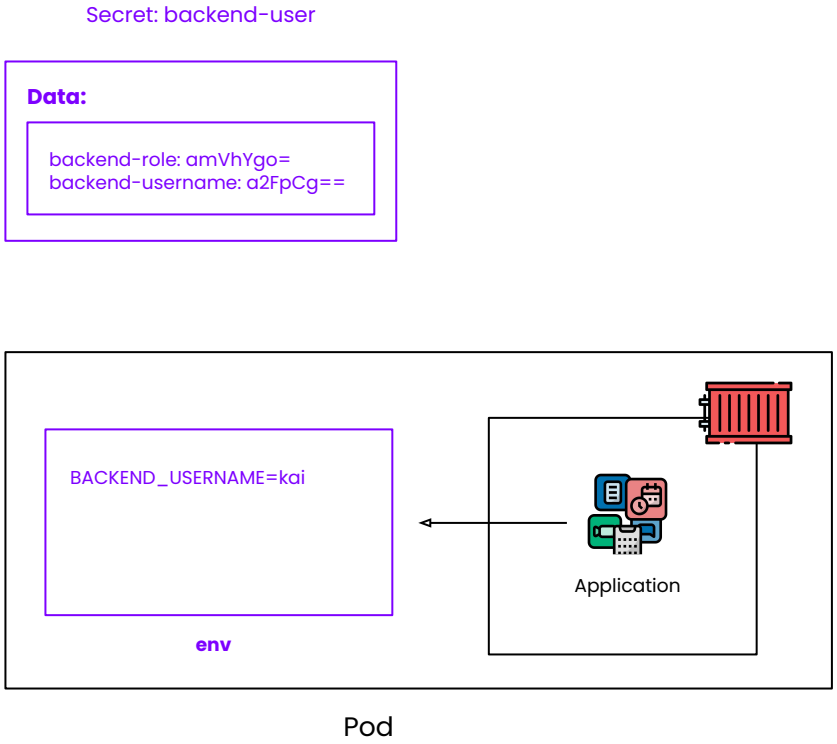
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envvars-multiple-secrets
spec:
  containers:
  - name: envars-test-container
    image: nginx
    env:
    - name: BACKEND_USERNAME
      valueFrom:
        secretKeyRef:
          name: backend-user
          key: backend-username
    - name: DB_USERNAME
      valueFrom:
        secretKeyRef:
          name: db-user
          key: db-username
```

Configure a Pod to Use a Secret - Environment Variable (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envvars-multiple-secrets
spec:
  containers:
  - name: envars-test-container
    image: nginx
    env:
    - name: BACKEND_USERNAME
      valueFrom:
        secretKeyRef:
          name: backend-user
          key: backend-username
    - name: DB_USERNAME
      valueFrom:
        secretKeyRef:
          name: db-user
          key: db-username
```

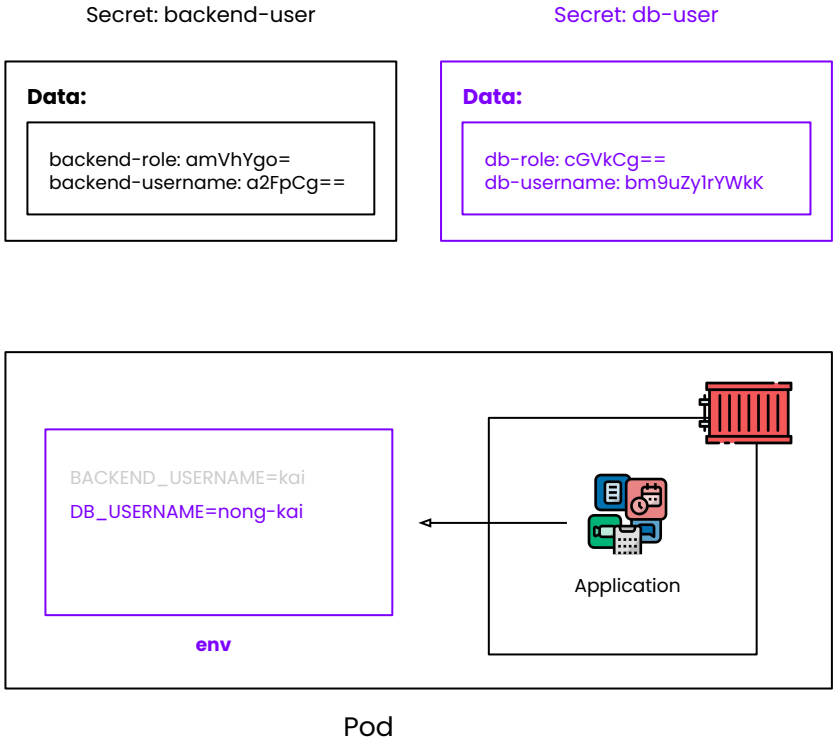
Configure a Pod to Use a Secret - Environment Variable (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envvars-multiple-secrets
spec:
  containers:
  - name: envars-test-container
    image: nginx
    env:
    - name: BACKEND_USERNAME
      valueFrom:
        secretKeyRef:
          name: backend-user
          key: backend-username
    - name: DB_USERNAME
      valueFrom:
        secretKeyRef:
          name: db-user
          key: db-username
```



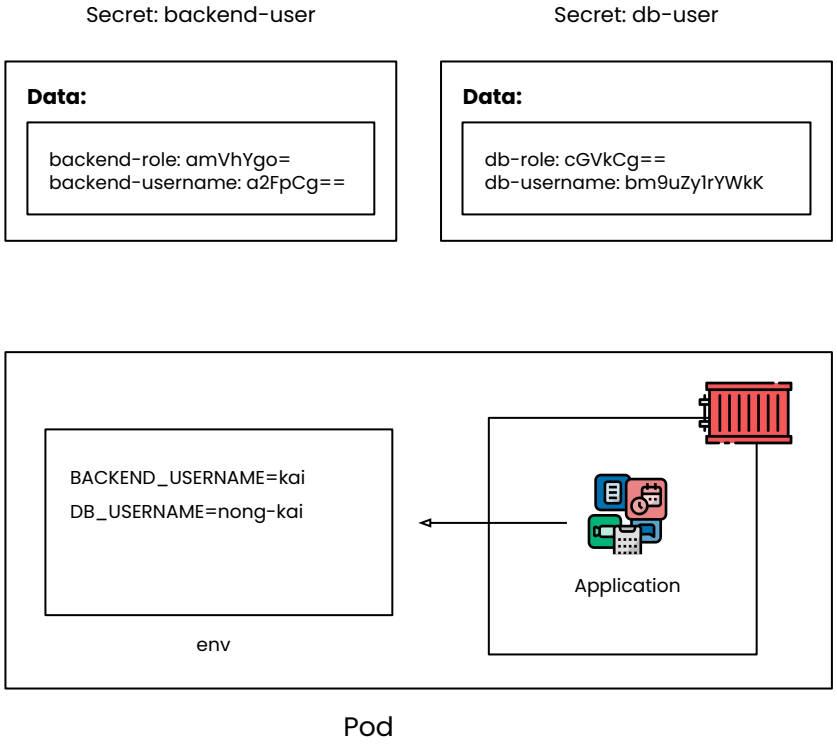
Configure a Pod to Use a Secret - Environment Variable (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envvars-multiple-secrets
spec:
  containers:
  - name: envars-test-container
    image: nginx
    env:
    - name: BACKEND_USERNAME
      valueFrom:
        secretKeyRef:
          name: backend-user
          key: backend-username
    - name: DB_USERNAME
      valueFrom:
        secretKeyRef:
          name: db-user
          key: db-username
```



Configure a Pod to Use a Secret - Environment Variable (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envvars-multiple-secrets
spec:
  containers:
  - name: envars-test-container
    image: nginx
    env:
    - name: BACKEND_USERNAME
      valueFrom:
        secretKeyRef:
          name: backend-user
          key: backend-username
    - name: DB_USERNAME
      valueFrom:
        secretKeyRef:
          name: db-user
          key: db-username
```



Jumpbox®

All key-value as Environment

- Configure a Pod to Use a Secret -

Jumpbox®

Configure a Pod to Use a Secret - All key-value (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envfrom-secret
spec:
  containers:
  - name: envars-test-container
    image: nginx
    envFrom:
    - secretRef:
        name: test-secret
```

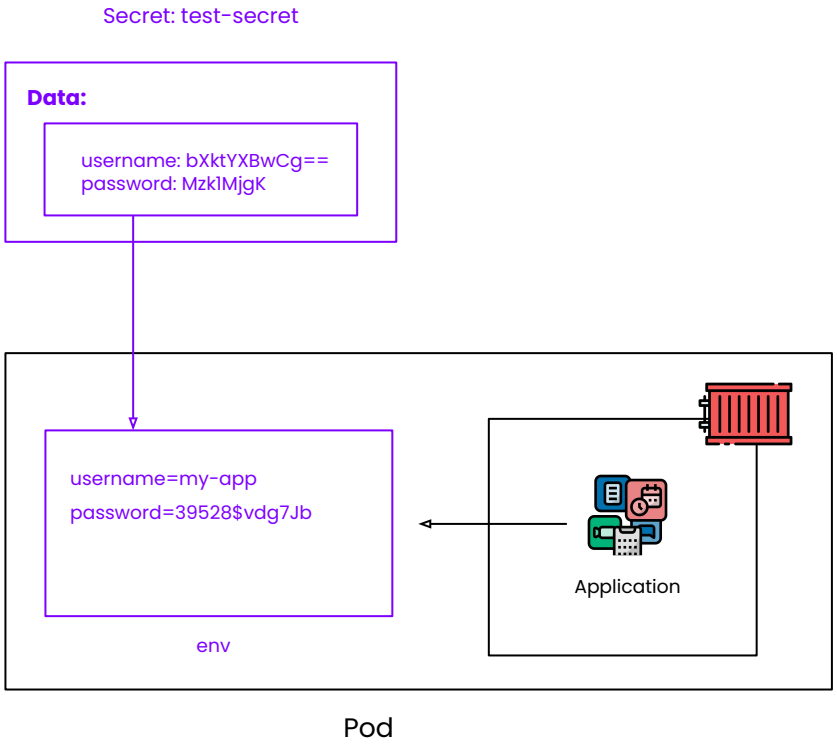
Configure a Pod to Use a Secret - All key-value (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envfrom-secret
spec:
  containers:
  - name: envars-test-container
    image: nginx
    envFrom:
    - secretRef:
        name: test-secret
```

Pod

Configure a Pod to Use a Secret - All key-value (Cont.)

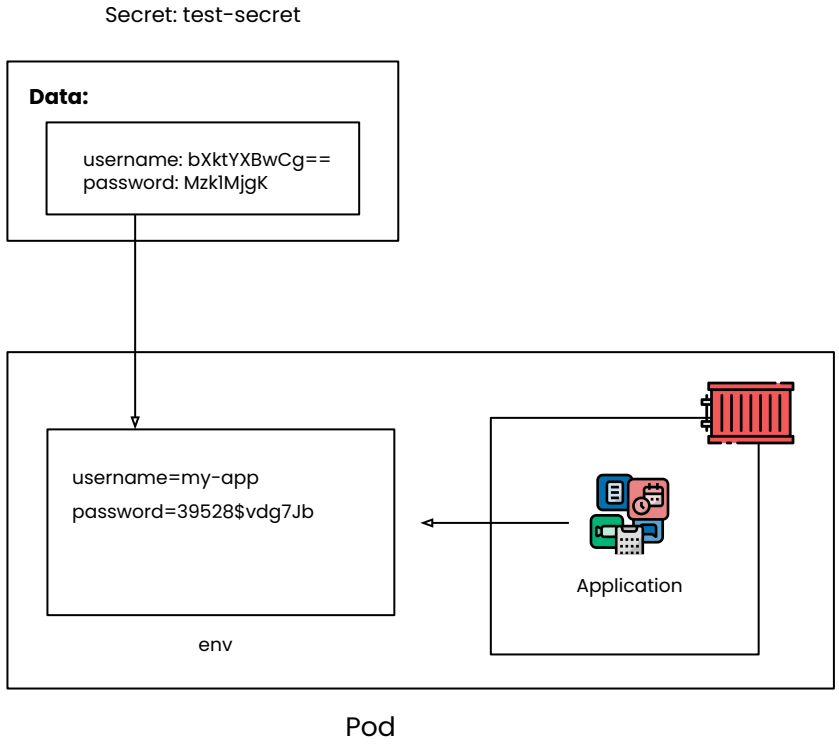
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envfrom-secret
spec:
  containers:
  - name: envars-test-container
    image: nginx
    envFrom:
    - secretRef:
        name: test-secret
```



Jumpbox®

Configure a Pod to Use a Secret - All key-value (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envfrom-secret
spec:
  containers:
  - name: envars-test-container
    image: nginx
    envFrom:
    - secretRef:
        name: test-secret
```



Jumpbox®

All key-value as file(s)

- Configure a Pod to Use a Secret -

Jumpbox®

Configure a Pod to Use a Secret - All key-value as file(s) (Cont.)

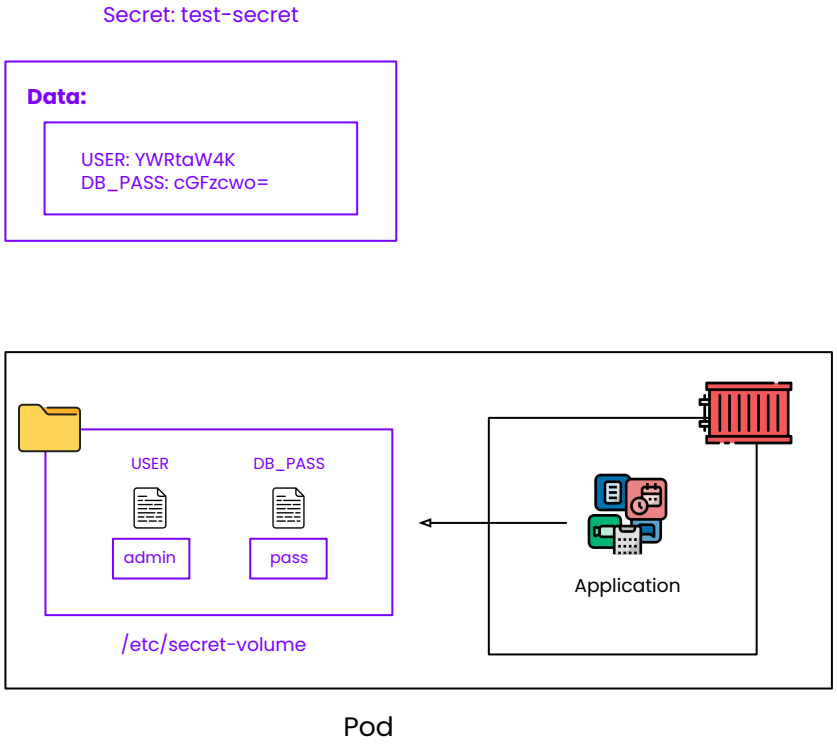
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secret-test-pod
spec:
  containers:
  - name: test-container
    image: nginx
    volumeMounts:
    - name: secret-volume
      mountPath: /etc/secret-volume
      readOnly: true
  volumes:
  - name: secret-volume
    secret:
      secretName: test-secret
```

Configure a Pod to Use a Secret - All key-value as file(s) (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secret-test-pod
spec:
  containers:
  - name: test-container
    image: nginx
    volumeMounts:
    - name: secret-volume
      mountPath: /etc/secret-volume
      readOnly: true
  volumes:
  - name: secret-volume
    secret:
      secretName: test-secret
```

Configure a Pod to Use a Secret - All key-value as file(s) (Cont.)

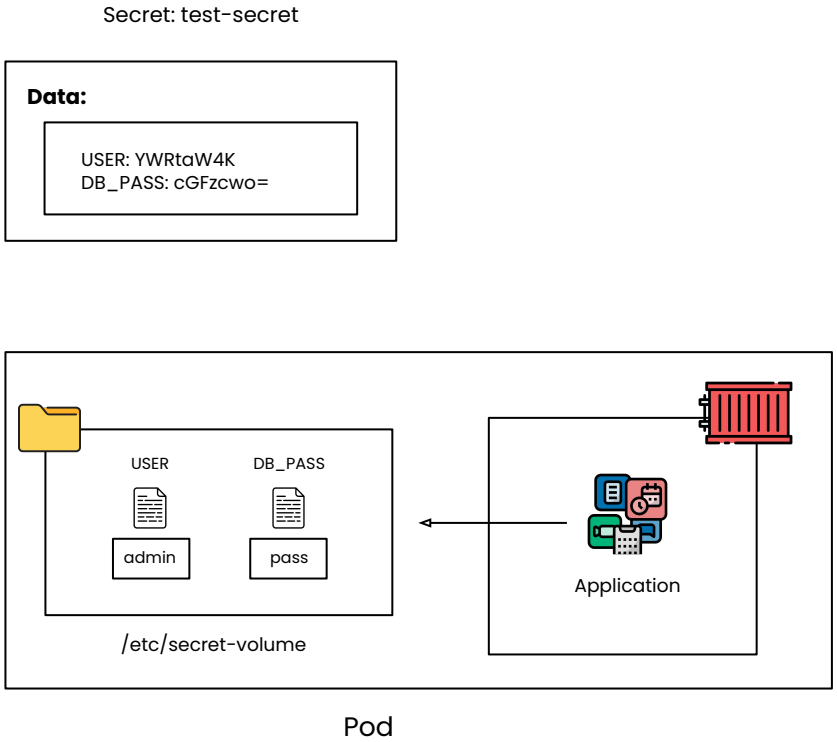
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secret-test-pod
spec:
  containers:
  - name: test-container
    image: nginx
    volumeMounts:
    - name: secret-volume
      mountPath: /etc/secret-volume
      readOnly: true
  volumes:
  - name: secret-volume
    secret:
      secretName: test-secret
```



Jumpbox®

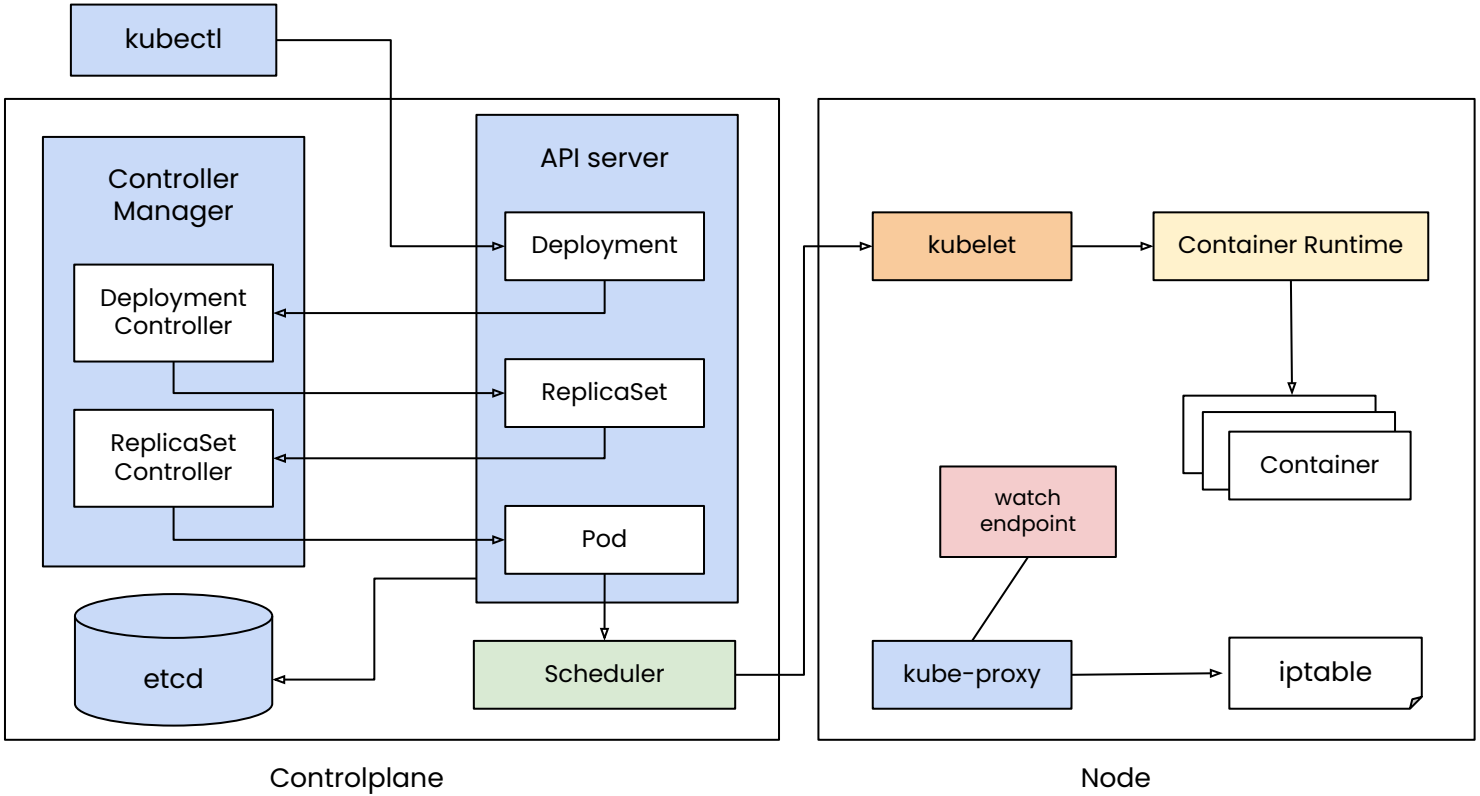
Configure a Pod to Use a Secret - All key-value as file(s) (Cont.)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secret-test-pod
spec:
  containers:
  - name: test-container
    image: nginx
    volumeMounts:
    - name: secret-volume
      mountPath: /etc/secret-volume
      readOnly: true
  volumes:
  - name: secret-volume
    secret:
      secretName: test-secret
```



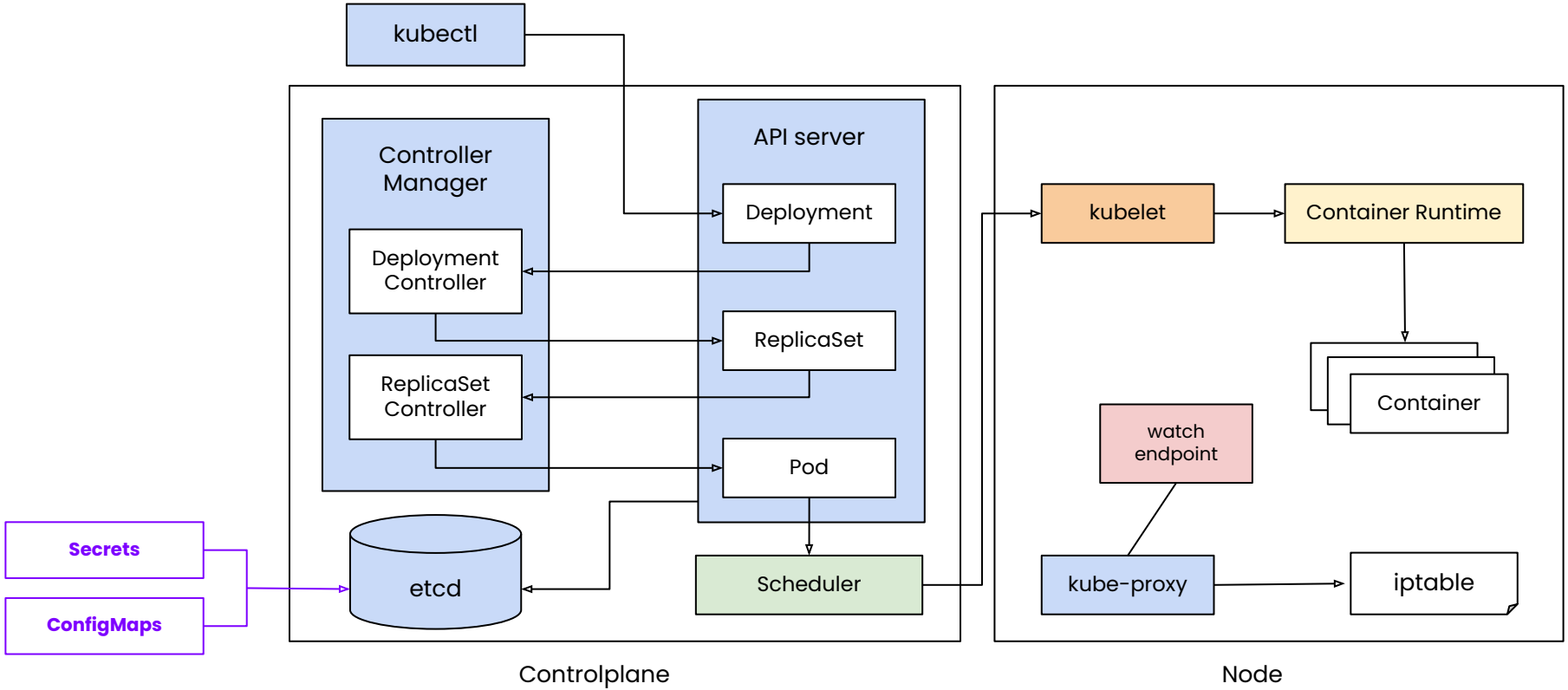
Jumpbox®

Where the ConfigMaps and Secret stored in?



Jumpbox®

Where the ConfigMaps and Secret stored in?



Jumpbox®

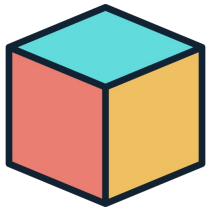
Secret

With Real Action Simulator

[Click Here]

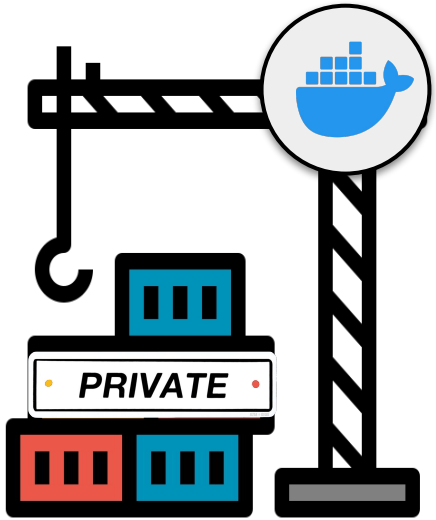
Private Registry

Private Registry (Cont.)



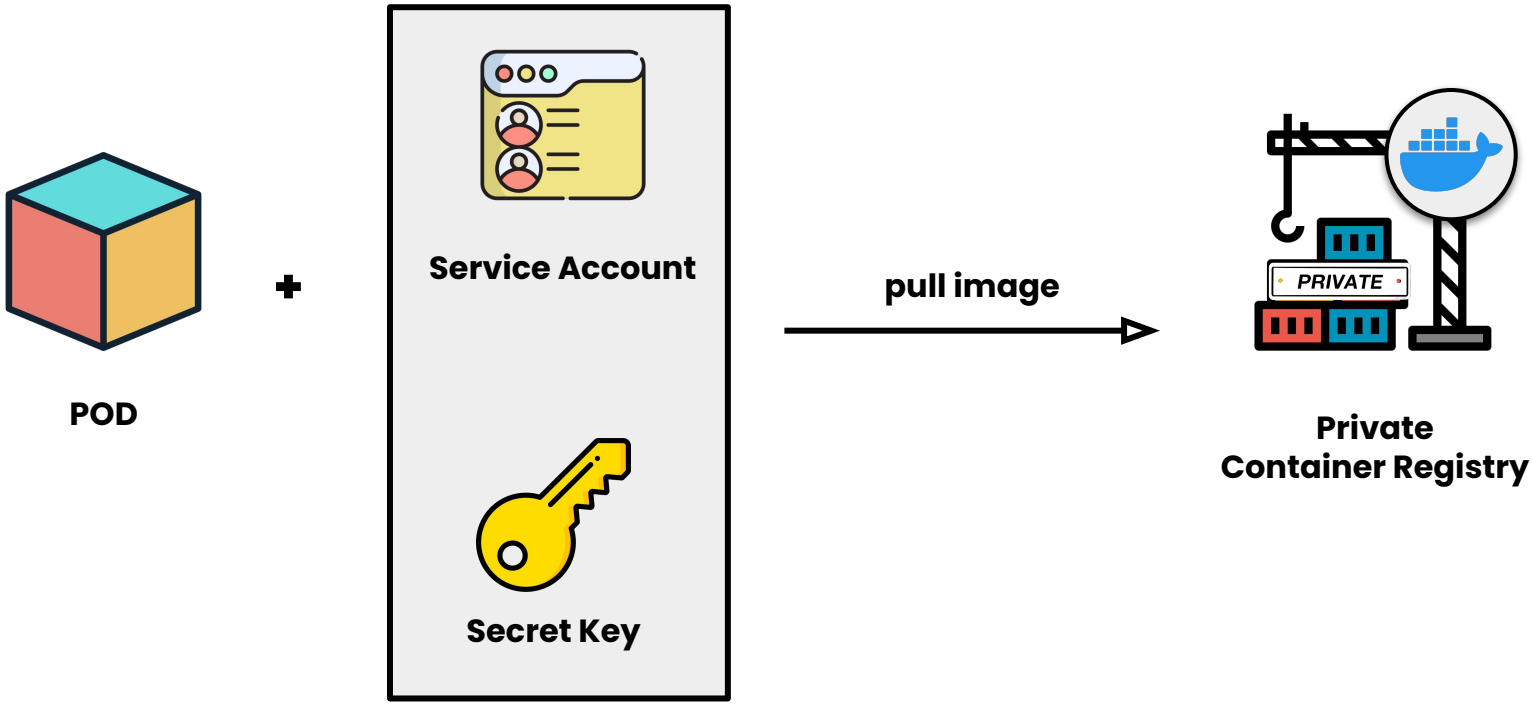
POD

pull image



Private
Container Registry

Private Registry (Cont.)



Volume

Volume

Types of Persistent Volumes

PersistentVolume types are implemented as plugins. Kubernetes currently supports the following plugins:

- `csi` - Container Storage Interface (CSI)
- `fc` - Fibre Channel (FC) storage
- `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

The following types of PersistentVolume are deprecated but still available. If you are using these volume types except for `flexVolume`, `cephfs` and `rbd`, please install corresponding CSI drivers.

- `awsElasticBlockStore` - AWS Elastic Block Store (EBS) (**migration on by default** starting v1.23)
- `azureDisk` - Azure Disk (**migration on by default** starting v1.23)
- `azureFile` - Azure File (**migration on by default** starting v1.24)
- `cephfs` - CephFS volume (**deprecated** starting v1.28, no migration plan, support will be removed in a future release)
- `cinder` - Cinder (OpenStack block storage) (**migration on by default** starting v1.21)
- `flexVolume` - FlexVolume (**deprecated** starting v1.23, no migration plan and no plan to remove support)
- `gcePersistentDisk` - GCE Persistent Disk (**migration on by default** starting v1.23)
- `portworxVolume` - Portworx volume (**deprecated** starting v1.25)
- `rbd` - Rados Block Device (RBD) volume (**deprecated** starting v1.28, no migration plan, support will be removed in a future release)
- `vsphereVolume` - vSphere VMDK volume (**migration on by default** starting v1.25)

Reference Pictures:

- Types of Persistent Volumes

Volume (Cont.)

ReadWriteOnce

the volume can be mounted as read-write by a single node. ReadWriteOnce access mode still can allow multiple pods to access the volume when the pods are running on the same node. For single pod access, please see ReadWriteOncePod.

ReadOnlyMany

the volume can be mounted as read-only by many nodes.

ReadWriteMany

the volume can be mounted as read-write by many nodes.

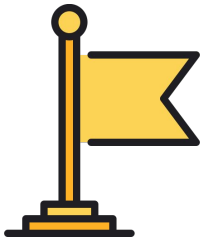
ReadWriteOncePod

ⓘ **FEATURE STATE:** `Kubernetes v1.29 [stable]`

the volume can be mounted as read-write by a single Pod. Use ReadWriteOncePod access mode if you want to ensure that only one pod across the whole cluster can read that PVC or write to it.

Reference Pictures:

- Types of Persistent Volumes



Break

Kustomize

Kustomize.io

Generate ConfigMap from file

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
configMapGenerator:
  - name: db-connection
    files:
      - db-connection.json
```

```
{
  "host": "jumpbox",
  "user": "someone",
  "password": "p39df239ds"
}
```

db-connection.json

output

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  db-connection.json: |
{
  "host": "jumpbox",
  "user": "someone",
  "password": "p39df239ds"
}
```

Generate ConfigMap from file - Custom filename

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
configMapGenerator:
  - name: db-connection
    files:
      - config.json=db-connection.json
```

```
{
  "host": "jumpbox",
  "user": "someone",
  "password": "p39df239ds"
}
```

db-connection.json

output

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  config.json: |
{
  "host": "jumpbox.co",
  "user": "someone",
  "password": "p39df239ds"
}
```

Generate ConfigMap from env

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
configMapGenerator:
  - name: jumpbox
    env: .env.jumpbox
```

```
KEY_ID: someid
REGION: us-east-1
ACCESS_KEY: somesecret
```

.env.jumpbox

output

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: jumpbox
Data:
  KEY_ID: someid
  REGION: us-east-1
  ACCESS_KEY: somesecret
```

Disable Name Suffix Hash

```yaml
# Generator Option
generatorOptions:
  disableNameSuffixHash: false
```

output

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: config-map-asd23x
```

hash

```yaml
# Generator Option
generatorOptions:
  disableNameSuffixHash: true
```

output

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: config-map
```

Activity

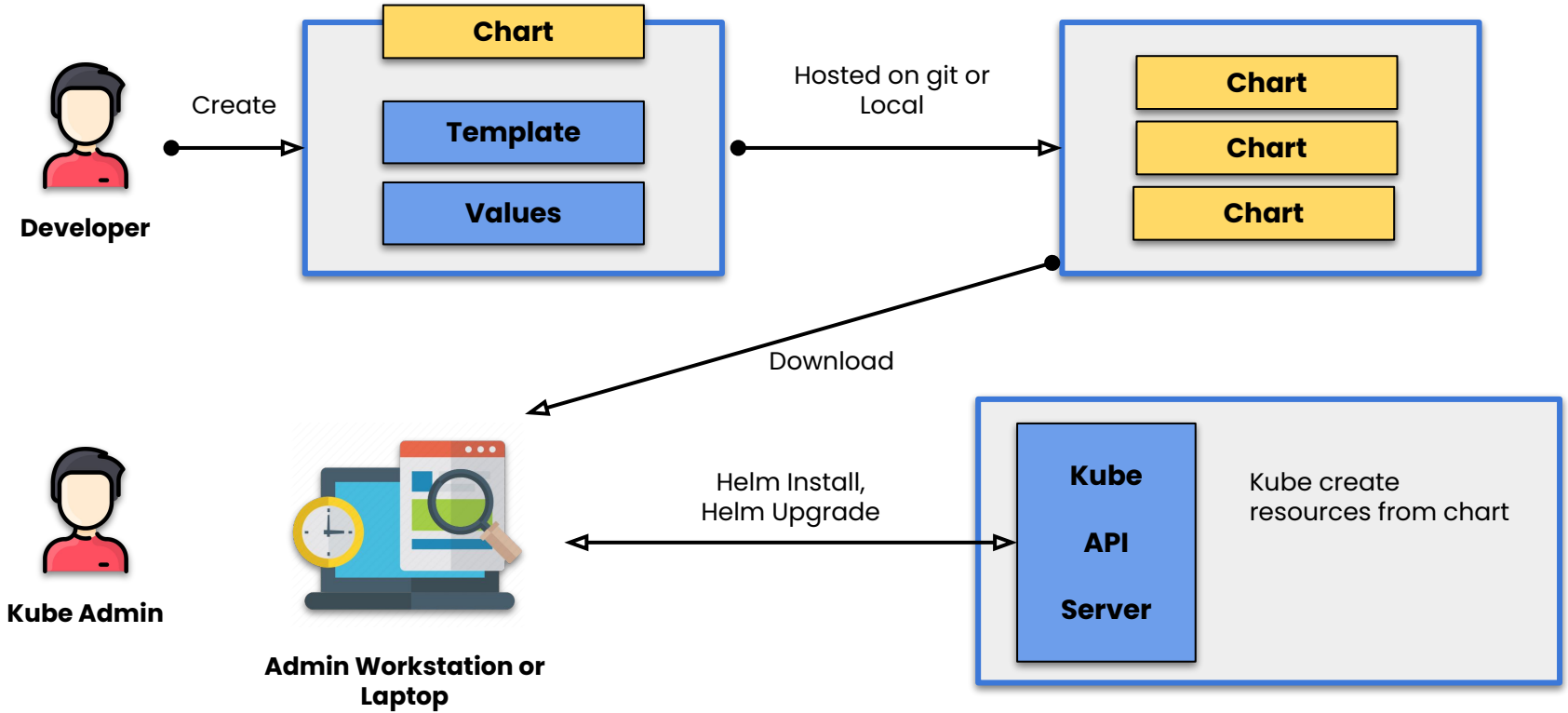
Helm

Helm: Kubernetes Package Manager (Cont.)

Helm

helps you define, install, and upgrade Kubernetes application as a chart (ig: Package) containing Kubernetes configuration files

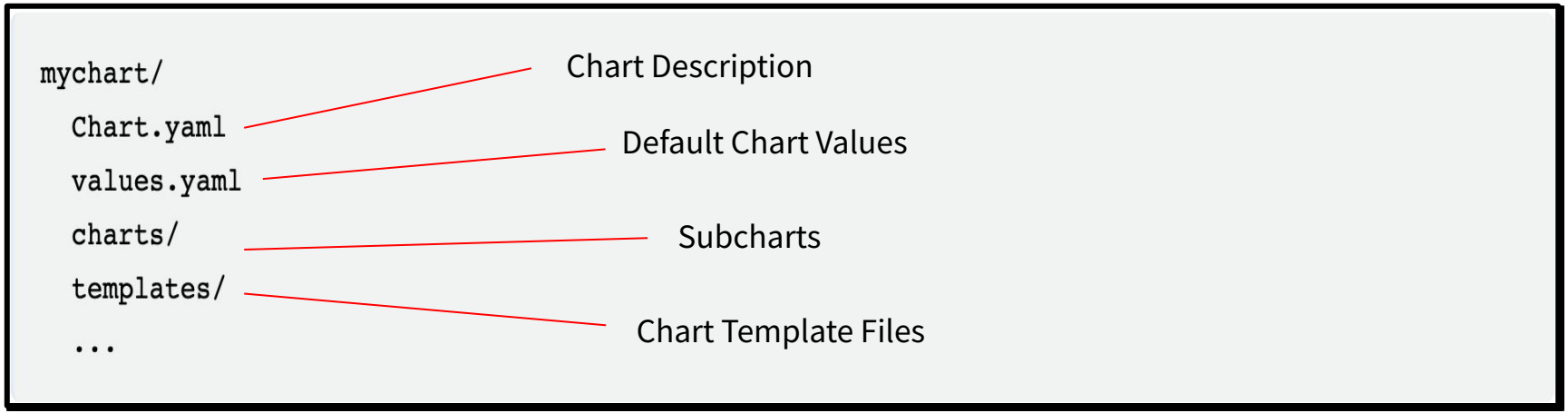
Helm: How it works?

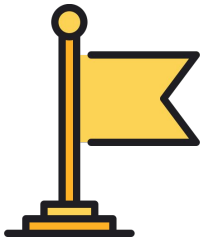


Jumpbox®

Helm: Chart Structure

Helm charts are structured like this:





Lunch break

Namespace

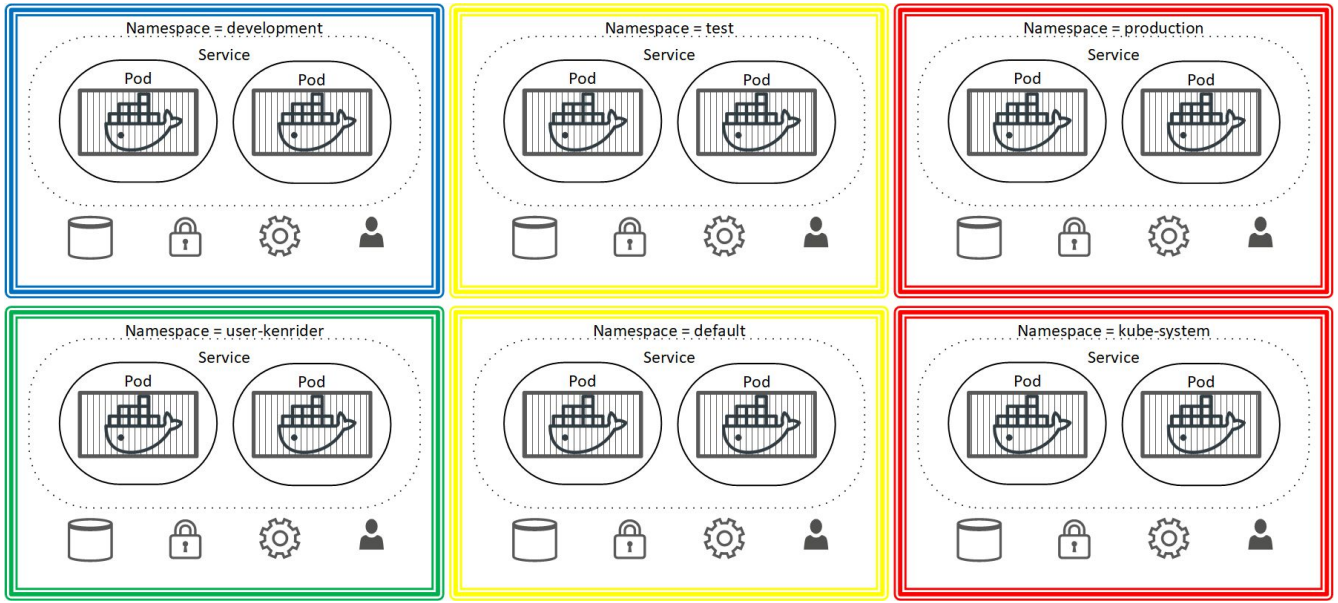
Namespace

Kubernetes namespaces help different projects, teams, or customers to share a Kubernetes cluster.

Reference:

- Namespaces Walkthrough

Namespace (Cont.)



Reference:

- [dev test and prod k8s namepsaces]

Namespace use case

Use-case #1: Roles and Responsibilities in an Enterprise

Use-case #2: Using Namespaces to partition development landscapes

Use-case #3: Partitioning of your Customers

Reference:

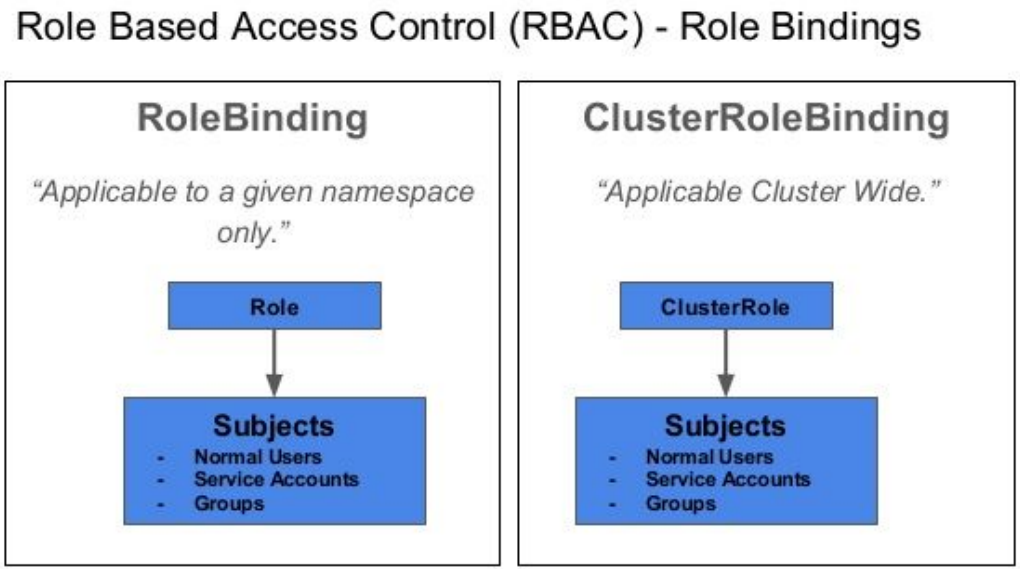
- Kubernetes Namespaces: use cases and insights

RBAC

RBAC

Using RBAC Authorization

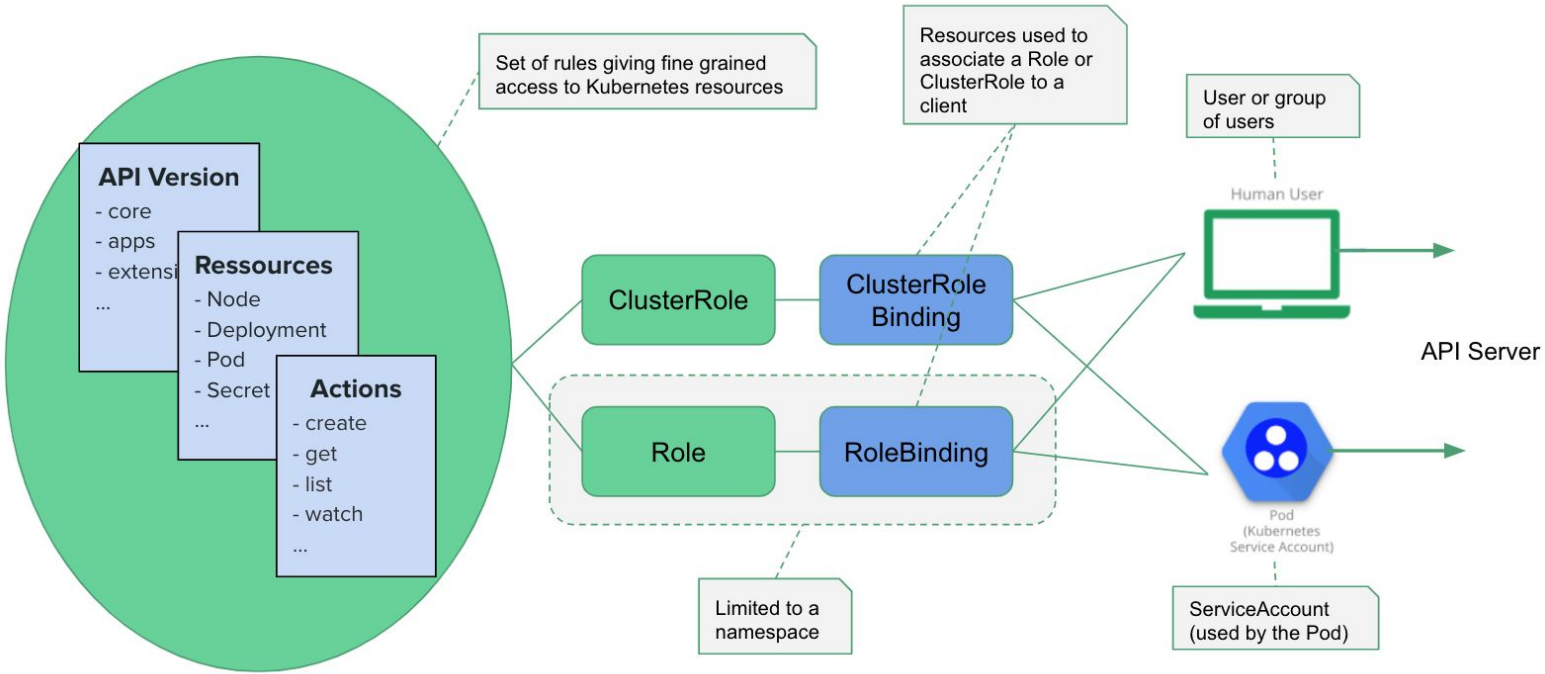
Role and RoleBinding
ClusterRole and ClusterRoleBinding



Reference:

- User authentication and authorizarion in Kubernetes

RBAC overview

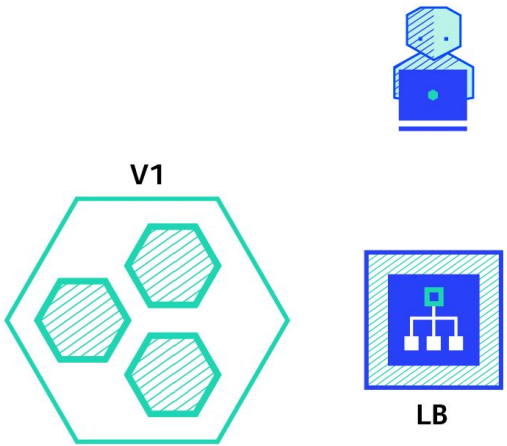


Reference:

- [Kubernetes Tips: Using a ServiceAccount]

Graceful

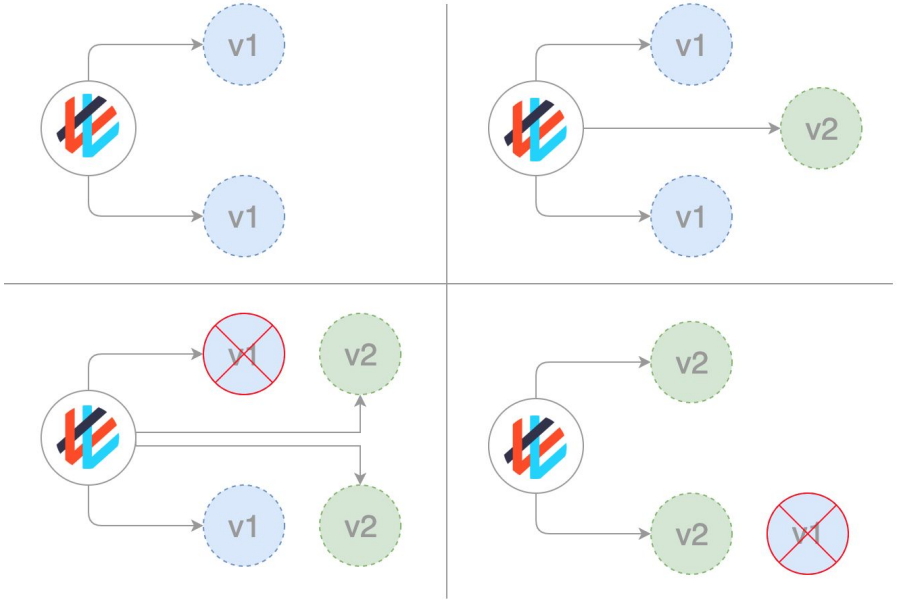
Deployment



Reference:

- Deployment Strategies

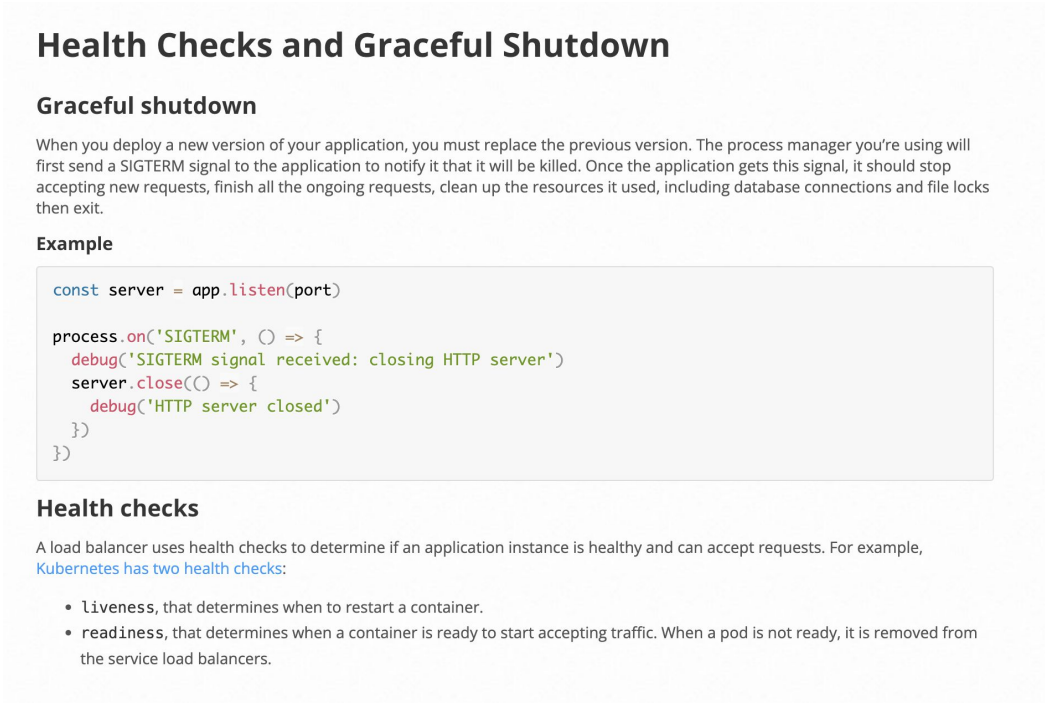
Rolling Deployment



Reference Pictures:

- [Kubernetes Deployment Strategies]

Express official Health Checks and Graceful Shutdown



Health Checks and Graceful Shutdown

Graceful shutdown

When you deploy a new version of your application, you must replace the previous version. The process manager you're using will first send a SIGTERM signal to the application to notify it that it will be killed. Once the application gets this signal, it should stop accepting new requests, finish all the ongoing requests, clean up the resources it used, including database connections and file locks then exit.

Example

```
const server = app.listen(port)

process.on('SIGTERM', () => {
  debug('SIGTERM signal received: closing HTTP server')
  server.close(() => {
    debug('HTTP server closed')
  })
})
```

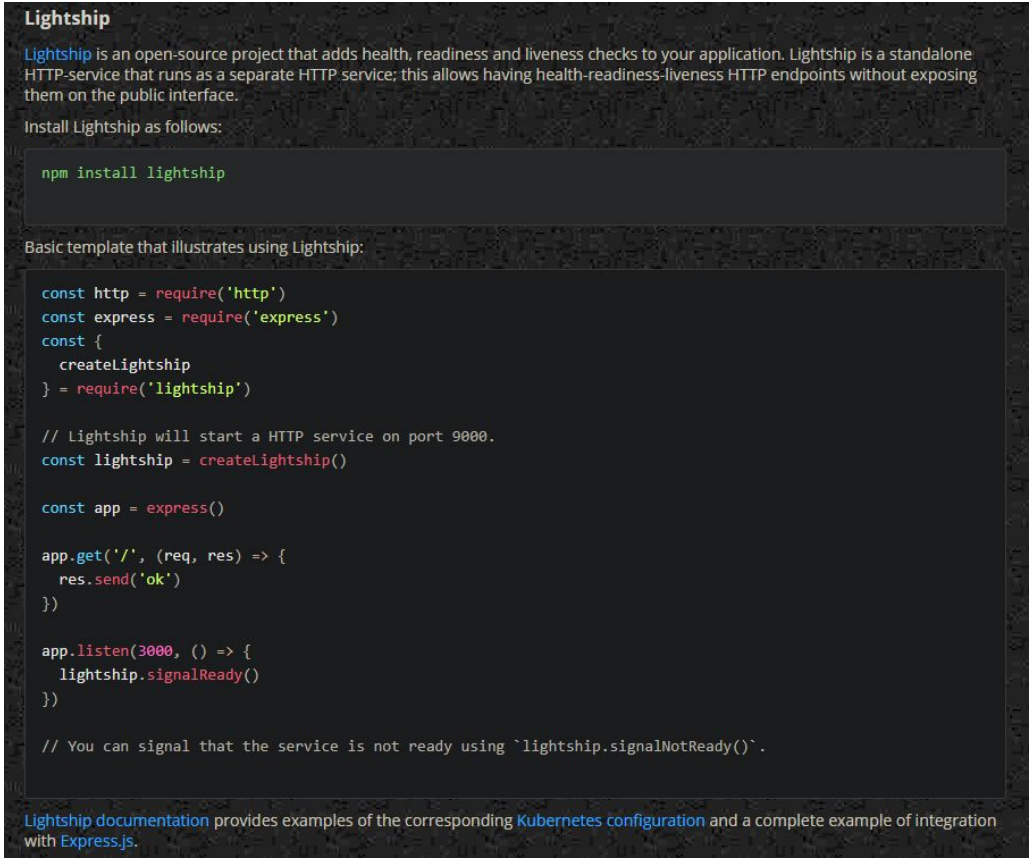
Health checks

A load balancer uses health checks to determine if an application instance is healthy and can accept requests. For example, Kubernetes has two health checks:

- `liveness`, that determines when to restart a container.
- `readiness`, that determines when a container is ready to start accepting traffic. When a pod is not ready, it is removed from the service load balancers.

Reference Pictures:

- Health Checks and Graceful Shutdown



Lightship

Lightship is an open-source project that adds health, readiness and liveness checks to your application. Lightship is a standalone HTTP-service that runs as a separate HTTP service; this allows having health-readiness-liveness HTTP endpoints without exposing them on the public interface.

Install Lightship as follows:

```
npm install lightship
```

Basic template that illustrates using Lightship:

```
const http = require('http')
const express = require('express')
const {
  createLightship
} = require('lightship')

// Lightship will start a HTTP service on port 9000.
const lightship = createLightship()

const app = express()

app.get('/', (req, res) => {
  res.send('ok')
})

app.listen(3000, () => {
  lightship.signalReady()
})

// You can signal that the service is not ready using `lightship.signalNotReady()`.
```

Lightship documentation provides examples of the corresponding Kubernetes configuration and a complete example of integration with Express.js.

Reference Pictures:

- Health Checks and Graceful Shutdown

Lightship

`/health`

`/health` endpoint describes the current state of a Node.js service.

The endpoint responds:

- `200` status code, message "SERVER_IS_READY" when server is accepting new connections.
- `500` status code, message "SERVER_IS_NOT_READY" when server is initialising.
- `500` status code, message "SERVER_IS_SHUTTING_DOWN" when server is shutting down.

Reference Pictures:
- lightship

Graceful example

```
function delay(ms: number) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

// define a route handler for the default home page
app.get("/", (_, res) => {
  res.send(`Hello world from ${uuid}`);
});

app.get("/heavy", (_, res) => {

  for (let i = 0; i < 1000000; i++) {
    console.log(`Round: ${i}`);
  }

  res.send(`Heavy from ${uuid}`);
});

app.get("/shutdown", (_, res) => {
  lightship.shutdown();
  res.send(`Starting shutdown from ${uuid}`);
});

lightship.registerShutdownHandler(async () => {
  console.log('Clear resource for shutdown.');
  await delay(shutdownTime);
  server.close();
});

// start the Express server
const server = app.listen(port, async () => {
  console.log(`server started at http://${host}:${port}`);
  await delay(readyTime);
  lightship.signalReady();
  console.log('Ready!');
});

```

Reference Pictures:

- https://github.com/Eji4h/hello-world-app

Healthcheck

- Power of Pod -

- Probe Type
- Define Probe
- Configure Probe

Jumpbox®

Probe Type

- Healthcheck -

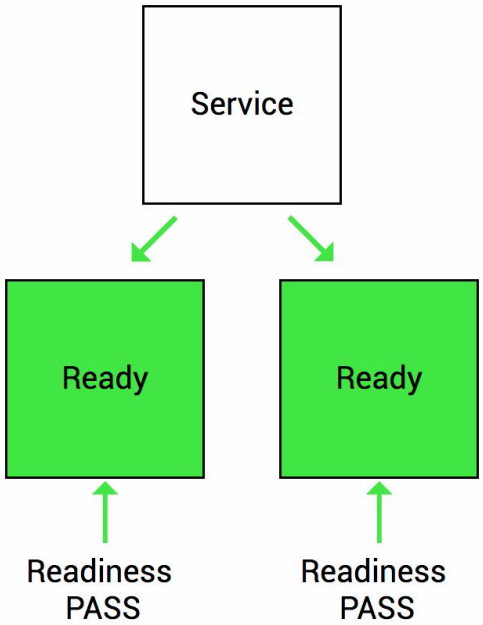
- Readiness Probe
- Liveness Probe
- Startup Probe

Readiness Probe

Readiness Probe (Cont.)

- Kubernetes uses readiness probes to know when a **container is ready to start accepting traffic**.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: readiness-exec
spec:
  containers:
  - name: readiness
    image: k8s.gcr.io/readiness
    readinessProbe:
      httpGet:
        path: /healthz
        port: 8080
      initialDelaySeconds: 3
      periodSeconds: 3
      timeoutSeconds: 1
```

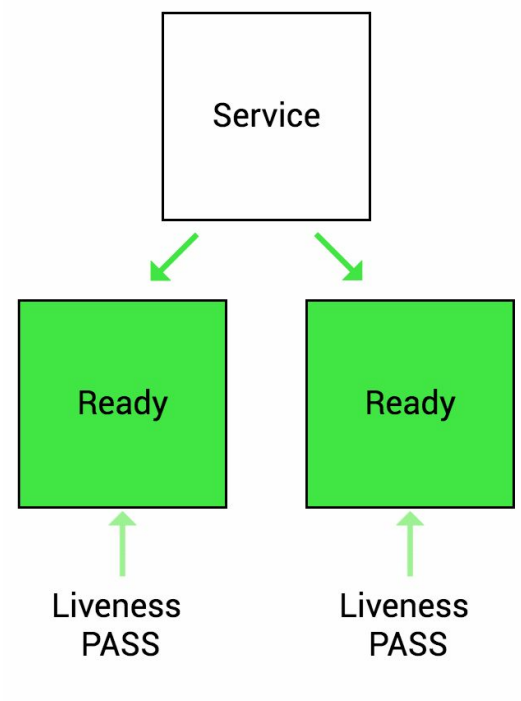


Liveness Probe

Liveness Probe (Cont.)

- Kubernetes uses liveness probes to know when to restart a container.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: liveness-exec
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/liveness
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
      initialDelaySeconds: 3
      periodSeconds: 3
      timeoutSeconds: 1
```



Startup Probe (Cont.)

Startup Probe (Cont.)

- The kubelet uses startup probes **to know when a container application has started**.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: liveness-exec
spec:
 containers:
 - name: startup
   image: nginx
   startupProbe:
     httpGet:
       path: /healthz
       port: 80
     initialDelaySeconds: 3
     periodSeconds: 10
     failureThreshold: 30
```

When should you use a startup probe?

When should you use a startup probe? (Cont.)

- Startup probes are useful for Pods that have **containers that take a long time** to come into service.

When should you use a startup probe? (Cont.)

- Startup probes are useful for Pods that have **containers that take a long time** to come into service.

- **Rather than set a long liveness interval**, you can configure a separate configuration for probing the container as it starts up, allowing a time longer than the liveness interval would allow.

When should you use a startup probe? (Cont.)

- Startup probes are useful for Pods that have **containers that take a long time** to come into service.
- **Rather than set a long liveness interval**, you can configure a separate configuration for probing the container as it starts up, allowing a time longer than the liveness interval would allow.
- Sometimes, you have to deal with legacy applications that might require an **additional startup time** on their first initialization. In such cases, it can be tricky to set up liveness probe parameters without compromising the fast response to deadlocks that motivated such a probe.

When should you use a startup probe? (Cont.)

- Startup probes are useful for Pods that have **containers that take a long time** to come into service.
- **Rather than set a long liveness interval**, you can configure a separate configuration for probing the container as it starts up, allowing a time longer than the liveness interval would allow.
- Sometimes, you have to deal with legacy applications that might require an **additional startup time** on their first initialization. In such cases, it can be tricky to set up liveness probe parameters without compromising the fast response to deadlocks that motivated such a probe.
- The trick is to **set up a startup probe** with the same command, HTTP or TCP check, with a failureThreshold * periodSeconds long enough to cover the worst case startup time

When should you use a startup probe? (Cont.)

- Startup probes are useful for Pods that have **containers that take a long time** to come into service.
- **Rather than set a long liveness interval**, you can configure a separate configuration for probing the container as it starts up, allowing a time longer than the liveness interval would allow.
- Sometimes, you have to deal with legacy applications that might require an **additional startup time** on their first initialization. In such cases, it can be tricky to set up liveness probe parameters without compromising the fast response to deadlocks that motivated such a probe.
- The trick is to **set up a startup probe** with the same command, HTTP or TCP check, with a failureThreshold * periodSeconds long enough to cover the worst case startup time

When should you use a startup probe? (Cont.)

- If your container usually starts in more than:

initialDelaySeconds + failureThreshold × periodSeconds

When should you use a startup probe? (Cont.)

- If your container usually starts in more than:

initialDelaySeconds + failureThreshold × periodSeconds

- you should specify a startup probe that checks the same endpoint as the liveness probe.

Example Protect slow starting containers (Cont.)

```yaml
startupProbe:
  httpGet:
    path: /healthz
    port: liveness-port
  failureThreshold: 30
  periodSeconds: 10

livenessProbe:
  httpGet:
    path: /healthz
    port: liveness-port
  failureThreshold: 1
  periodSeconds: 10
```

*** default ***
initialDelaySeconds: 0

Example Protect slow starting containers (Cont.)

```yaml
startupProbe:
 httpGet:
   path: /healthz
   port: liveness-port
 failureThreshold: 30
 periodSeconds: 10
```

startupProbe

30*10 = 300sec = 5min

*** default ***
initialDelaySeconds: 0

Example Protect slow starting containers (Cont.)

```yaml
startupProbe:
 httpGet:
   path: /healthz
   port: liveness-port
 failureThreshold: 30
 periodSeconds: 10
```

startupProbe

30*10 = 300sec = 5min

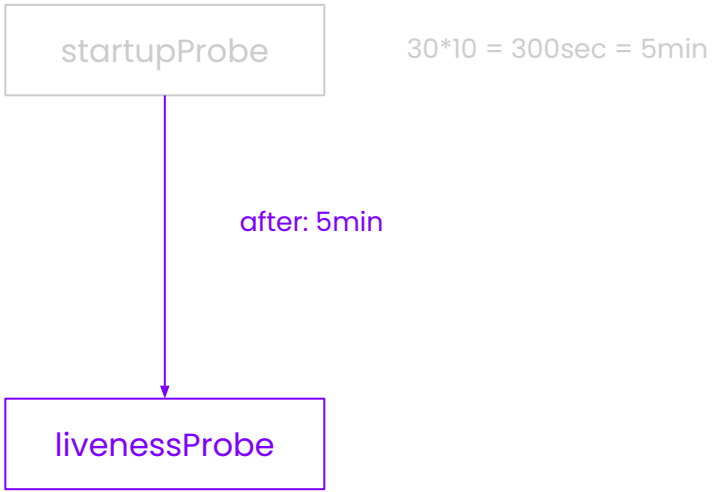
waiting: 5min

*** default ***
initialDelaySeconds: 0

Example Protect slow starting containers (Cont.)

```yaml
startupProbe:
 httpGet:
   path: /healthz
   port: liveness-port
 failureThreshold: 30
 periodSeconds: 10

livenessProbe:
 httpGet:
   path: /healthz
   port: liveness-port
 failureThreshold: 1
 periodSeconds: 10
```



*** default ***
initialDelaySeconds: 0

Define Probe

- 3. Healthcheck -

- httpGet
- tcpSocket
- exec

Define Probe (Cont.)

Define Probe (Cont.)

httpGet

- the kubelet sends an **HTTP request** to the specified **path** and **port** to perform the check.

```
Pod.spec:
containers:
- name: goproxy
  image: k8s.gcr.io/goproxy:0.1
  <readinessProbe/livenessProbe/startupProbe>:
    httpGet:
      path: /healthz
      port: 8080
```

Define Probe (Cont.)

tcpSocket

- Probe uses a TCP socket. With this configuration, the kubelet will **attempt to open a socket** to your container on the specified port

```
Pod.spec:
containers:
- name: goproxy
  image: k8s.gcr.io/goproxy:0.1
  <readinessProbe/livenessProbe/startupProbe>:
    tcpSocket:
      port: 8080
```

Define Probe (Cont.)

exec

- Probe uses a execute command. With this configuration, the kubelet will **attempt to execute command** in your container on the specified command

```
Pod.spec:
containers:
- name: goproxy
  image: k8s.gcr.io/goproxy:0.1
  <readinessProbe/livenessProbe/startupProbe>:
    exec:
      command:
      - cat
      - /tmp/healthy
```

Configure Probe

- 3. Healthcheck -

- Readiness Probe
- Liveness Probe
- Startup Probe

Configure Probes

Configure Probes (Cont.)

initialDelaySeconds:

- Number of seconds after the container has started before liveness or readiness probes are initiated.
- Defaults to 0 seconds.
- Minimum value is 0.

```
Pod.spec:
 containers:
 - name: readiness
   image: k8s.gcr.io/readiness
   readinessProbe:
     httpGet:
       path: /healthz
       port: 8080
     initialDelaySeconds: 3
     periodSeconds: 3
     timeoutSeconds: 1
```

Configure Probes (Cont.)

periodSeconds:

- How often (in seconds) to perform the probe.
- Default to 10 seconds.
- Minimum value is 1.

```
Pod.spec:
 containers:
 - name: readiness
   image: k8s.gcr.io/readiness
   readinessProbe:
     httpGet:
       path: /healthz
       port: 8080
     initialDelaySeconds: 3
     periodSeconds: 3
     timeoutSeconds: 1
```

Configure Probes (Cont.)

timeoutSeconds:

- Number of seconds after which the probe times out.
- Defaults to 1 second.
- Minimum value is 1.

```
Pod.spec:
 containers:
 - name: readiness
   image: k8s.gcr.io/readiness
   readinessProbe:
     httpGet:
       path: /healthz
       port: 8080
     initialDelaySeconds: 3
     periodSeconds: 3
     timeoutSeconds: 1
```

Configure Probes (Cont.)

successThreshold:

- Minimum consecutive successes for the probe to be considered successful after having failed.
- Defaults to 1.
- Must be 1 for liveness and startup Probes.
- Minimum value is 1.

```
Pod.spec:
 containers:
 - name: readiness
   image: k8s.gcr.io/readiness
   readinessProbe:
     httpGet:
       path: /healthz
       port: 8080
     initialDelaySeconds: 3
     periodSeconds: 3
     successThreshold: 1
```

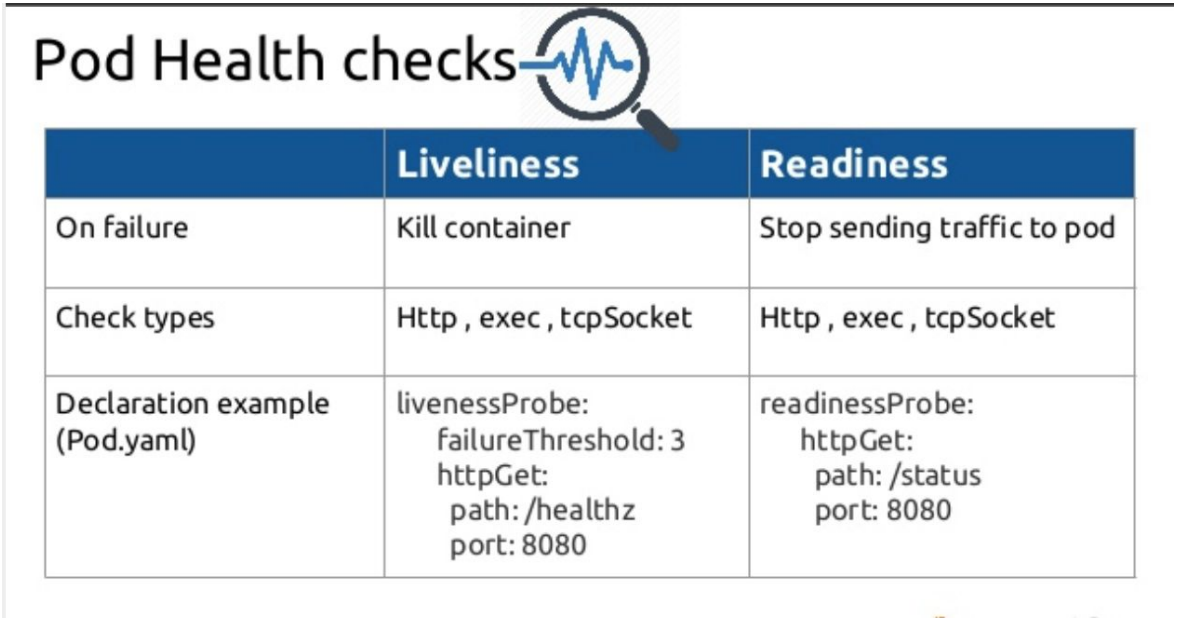
Configure Probes (Cont.)

failureThreshold:

- When a probe fails, Kubernetes will try failureThreshold times before giving up.
- Giving up in case of liveness probe means restarting the container.
- In case of readiness probe the Pod will be marked Unready.
- Defaults to 3.
- Minimum value is 1.

```
Pod.spec:
 containers:
 - name: readiness
   image: k8s.gcr.io/readiness
   readinessProbe:
     httpGet:
       path: /healthz
       port: 8080
     initialDelaySeconds: 3
     periodSeconds: 3
     failureThreshold: 3
```

Pod Health checks



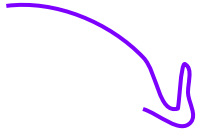
	Liveliness	Readiness
On failure	Kill container	Stop sending traffic to pod
Check types	Http , exec , tcpSocket	Http , exec , tcpSocket
Declaration example (Pod.yaml)	livenessProbe: failureThreshold: 3 httpGet: path: /healthz port: 8080	readinessProbe: httpGet: path: /status port: 8080

Reference Picture:

- https://www.weave.works/blog/resilient-apps-with-liveness-and-readiness-probes-in-kubernetes

Action Dojo

- Kata: small exercise to do by yourself



Activity Time !!

- Kata -

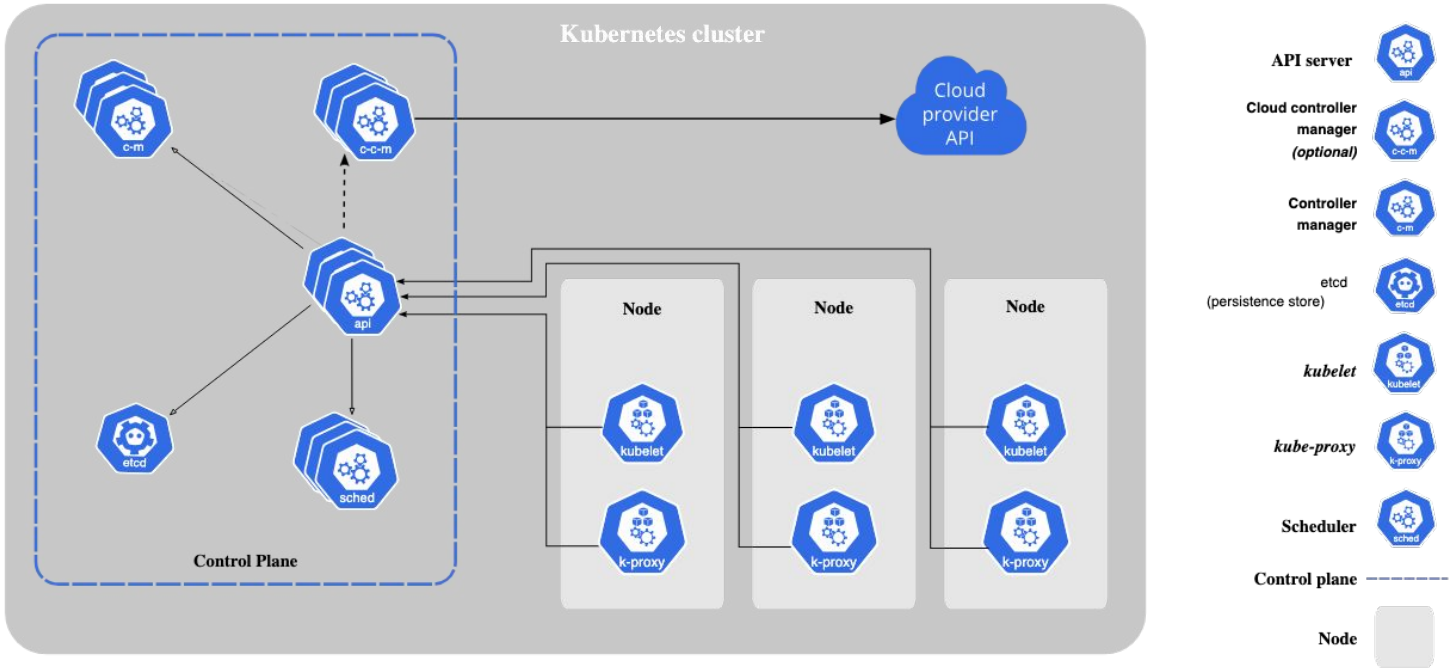
Healthcheck

With Real Action Simulator

[Click Here]

Kubernetes Components

Kubernetes Components

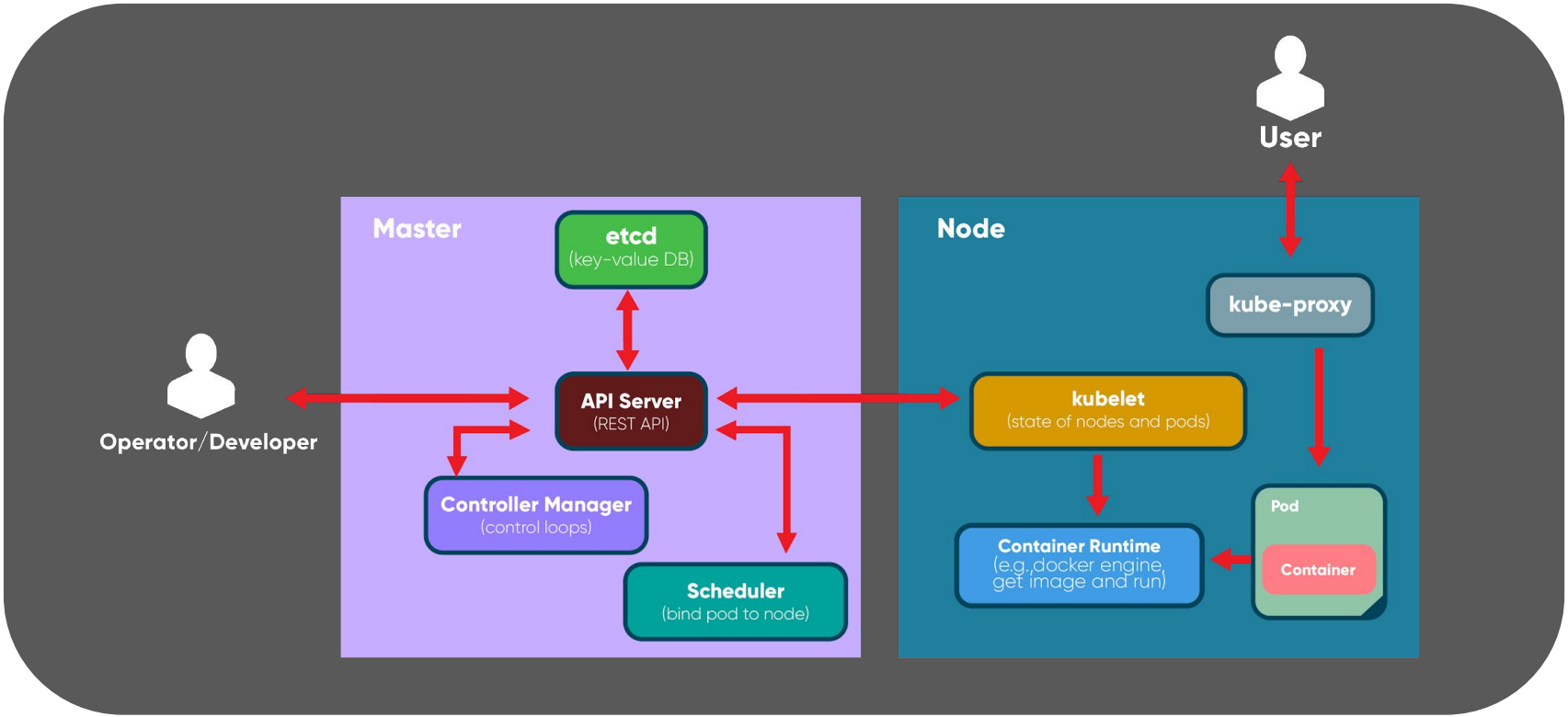


Reference Picture:

- Kubernetes Components

Jumpbox®

Kubernetes API flow



Extend API

Extend Kubernetes

Configure the Aggregation Layer

Use Custom Resources

Set up an Extension API Server

Configure Multiple Schedulers

Use an HTTP Proxy to Access the Kubernetes API

Use a SOCKS5 Proxy to Access the Kubernetes API

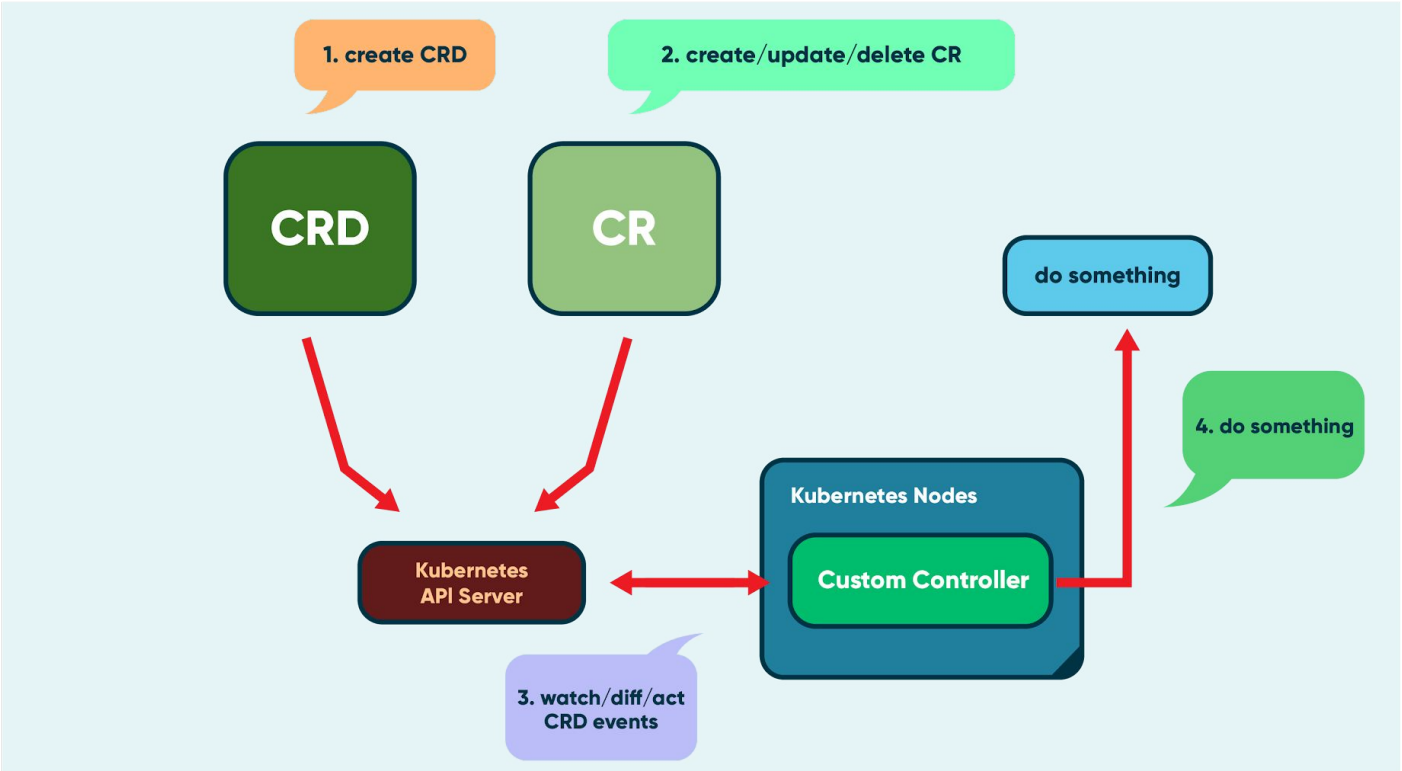
Set up Konnectivity service

Reference Picture:

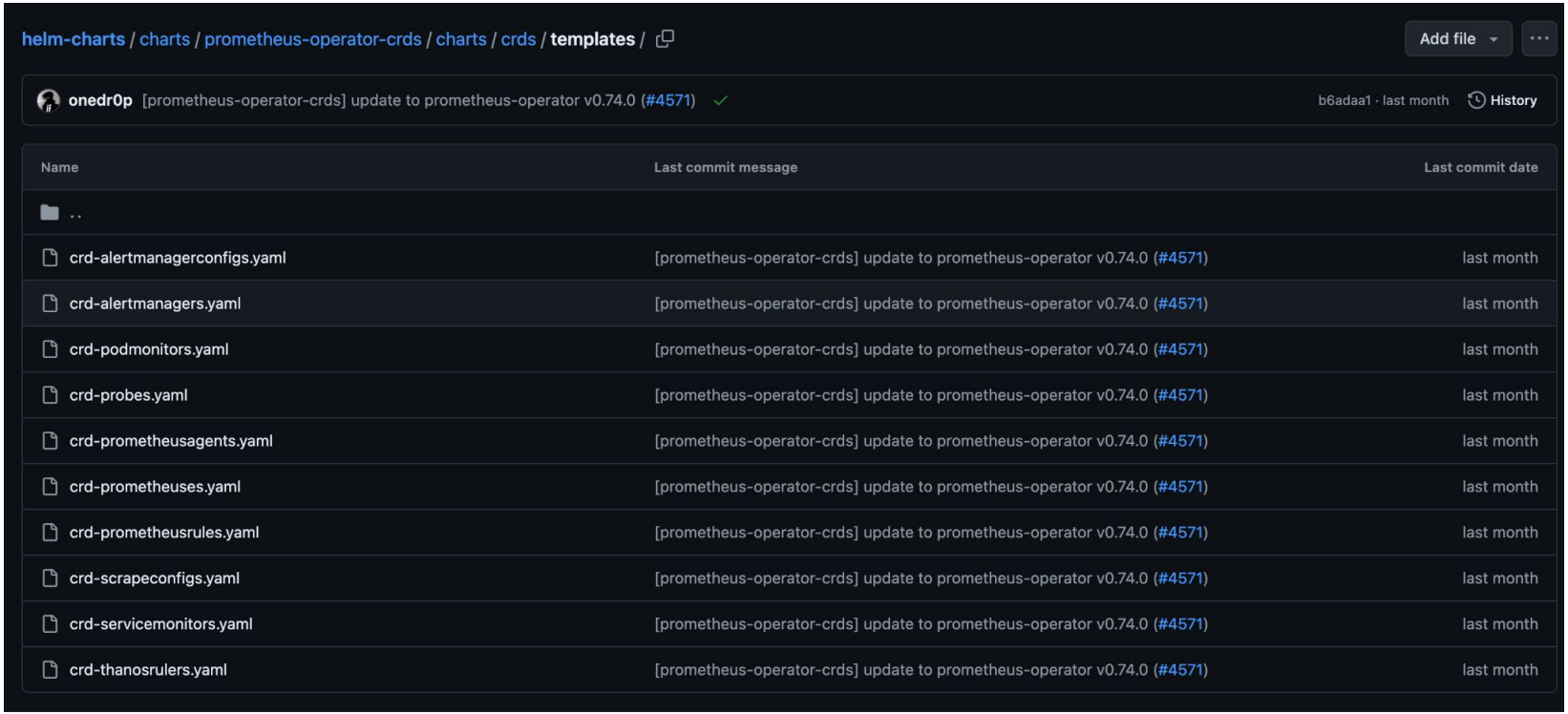
- Kubernetes Components

Kubernetes Custom Resource Definitions

CRD Simple Flow



Helm wrap it



Reference Picture:

- Prometheus Operator CRDs

Resource Management

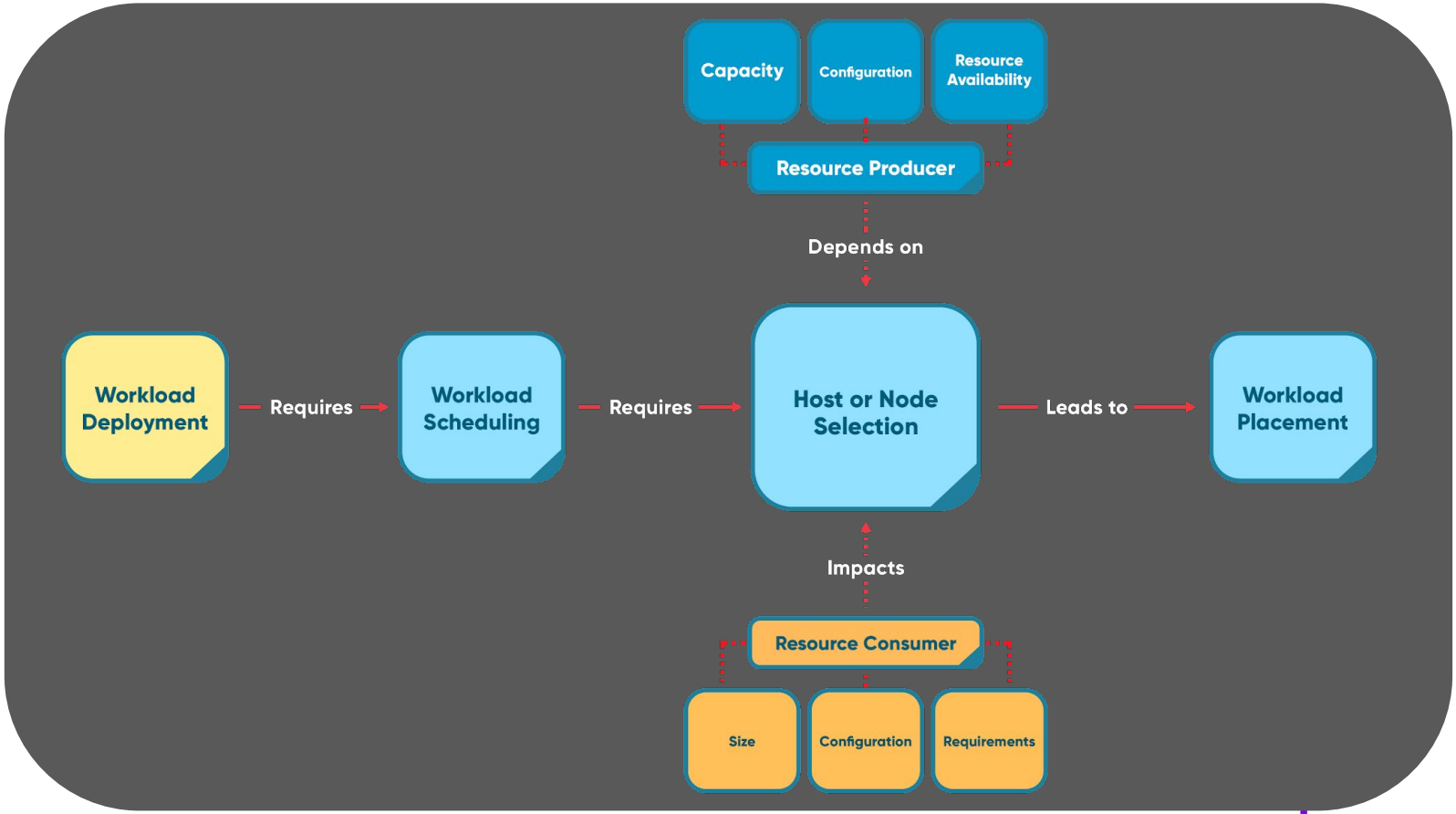
Resource Quotas

- Compute Resource Quota
- Storage Resource Quota
- Object Count Quota
- Quota Scope
- Quota Per PriorityClass
- Cross-namespace Pod Affinity Quota

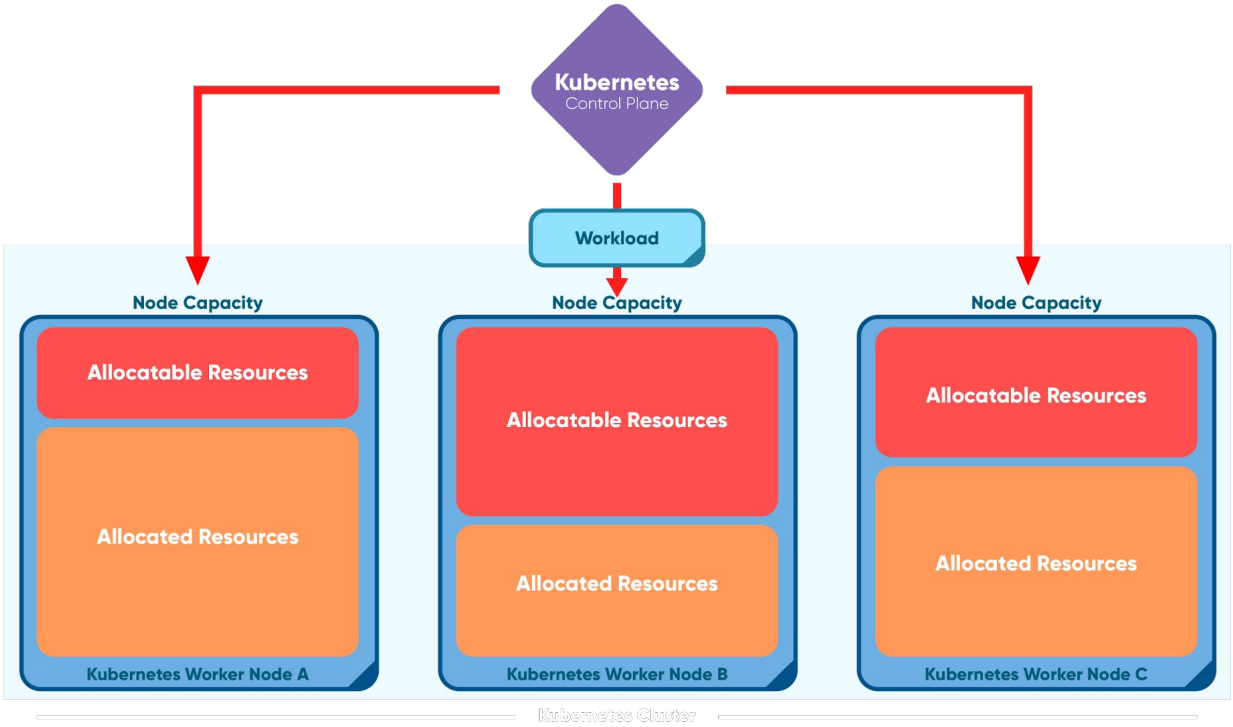
Reference:

- Resource Quotas

Workload Deployment Process



Kubernetes Scheduling



Jumpbox®

Compute Resource Quota

Resource Name	Description
`limits.cpu`	Across all pods in a non-terminal state, the sum of CPU limits cannot exceed this value.
`limits.memory`	Across all pods in a non-terminal state, the sum of memory limits cannot exceed this value.
`requests.cpu`	Across all pods in a non-terminal state, the sum of CPU requests cannot exceed this value.
`requests.memory`	Across all pods in a non-terminal state, the sum of memory requests cannot exceed this value.
`hugepages-<size>`	Across all pods in a non-terminal state, the number of huge page requests of the specified size cannot exceed this value.
`cpu`	Same as `requests.cpu`
`memory`	Same as `requests.memory`

Reference:

- Resource Quotas

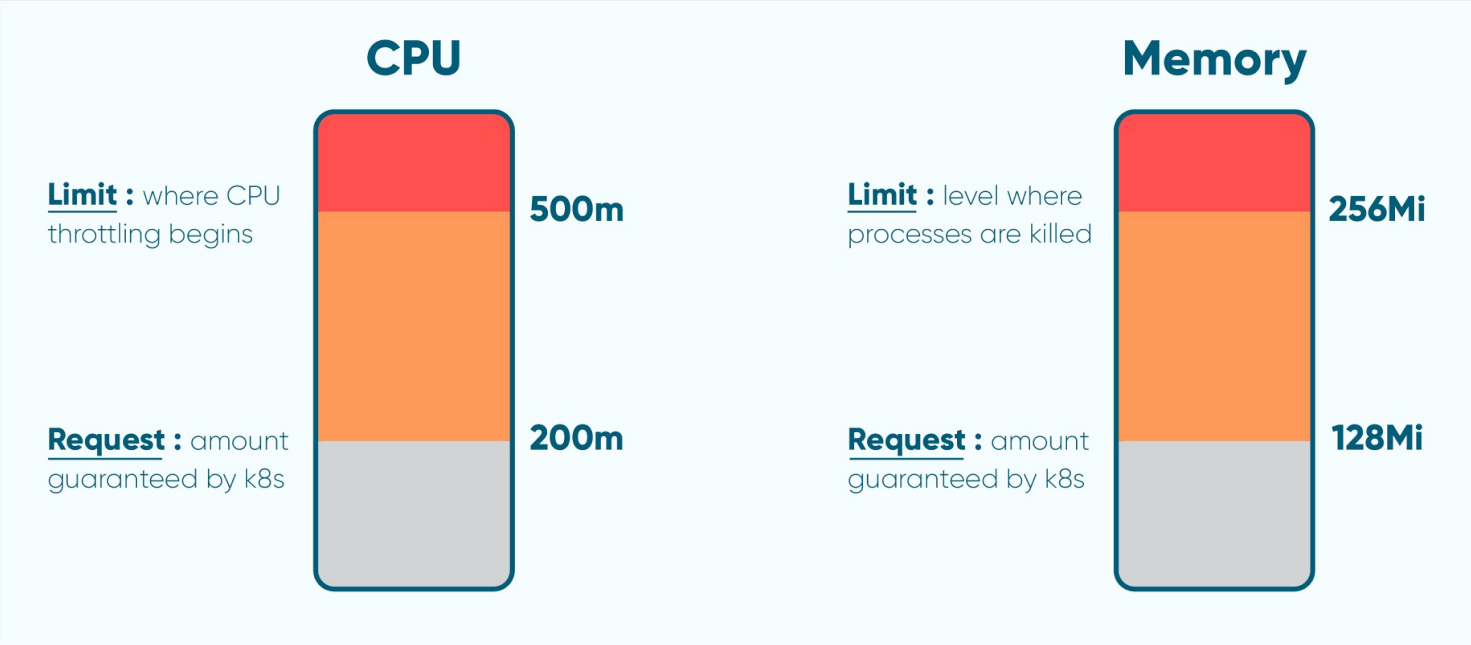
Pod Resource Quotas

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-no-conflict-with-limitrange-cpu
spec:
  containers:
  - name: demo
    image: registry.k8s.io/pause:2.0
    resources:
      requests:
        cpu: 700m
      limits:
        cpu: 700m
```

Reference:

- Limit Ranges

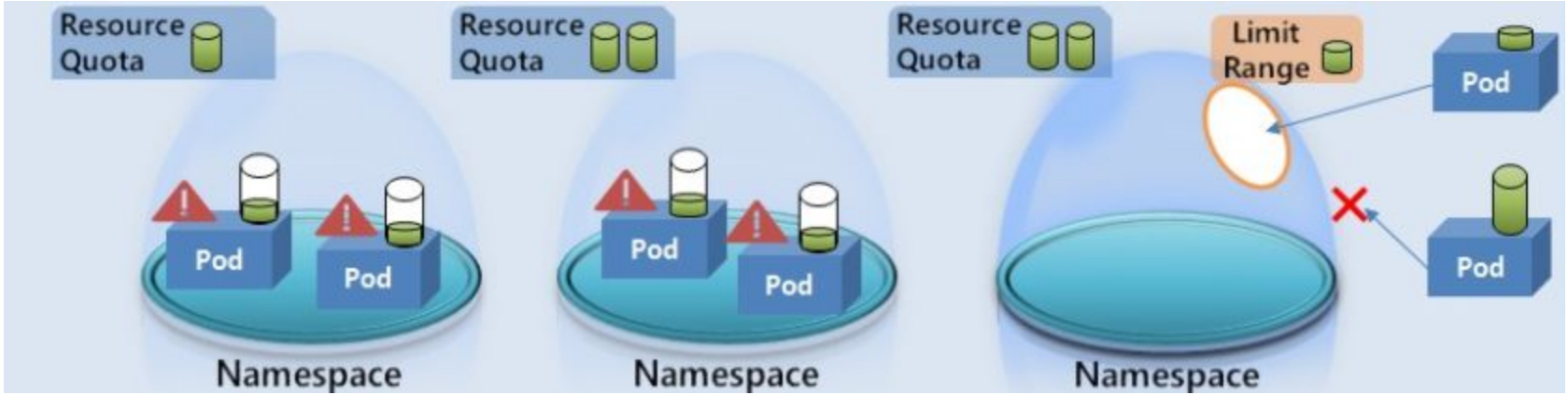
Pod Resource Quotas (Cont.)



Reference:

- Kubernetes Requests and Limits: A Practical Guide and Solutions

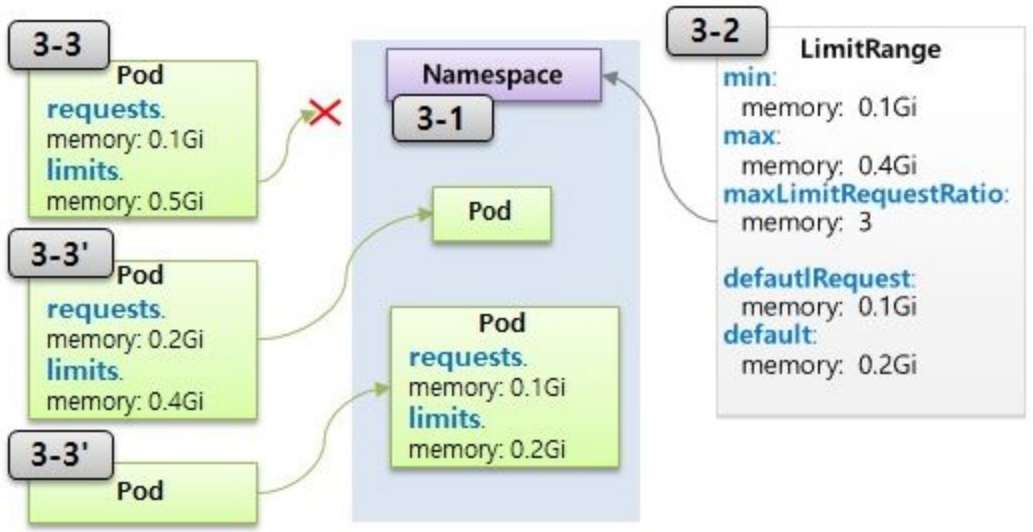
Pod Resource Quotas (Cont.)



Reference:

- [k8s] 11. Namespace, ResourceQuota, LimitRange - 실습

Limit Range Resource Quotas



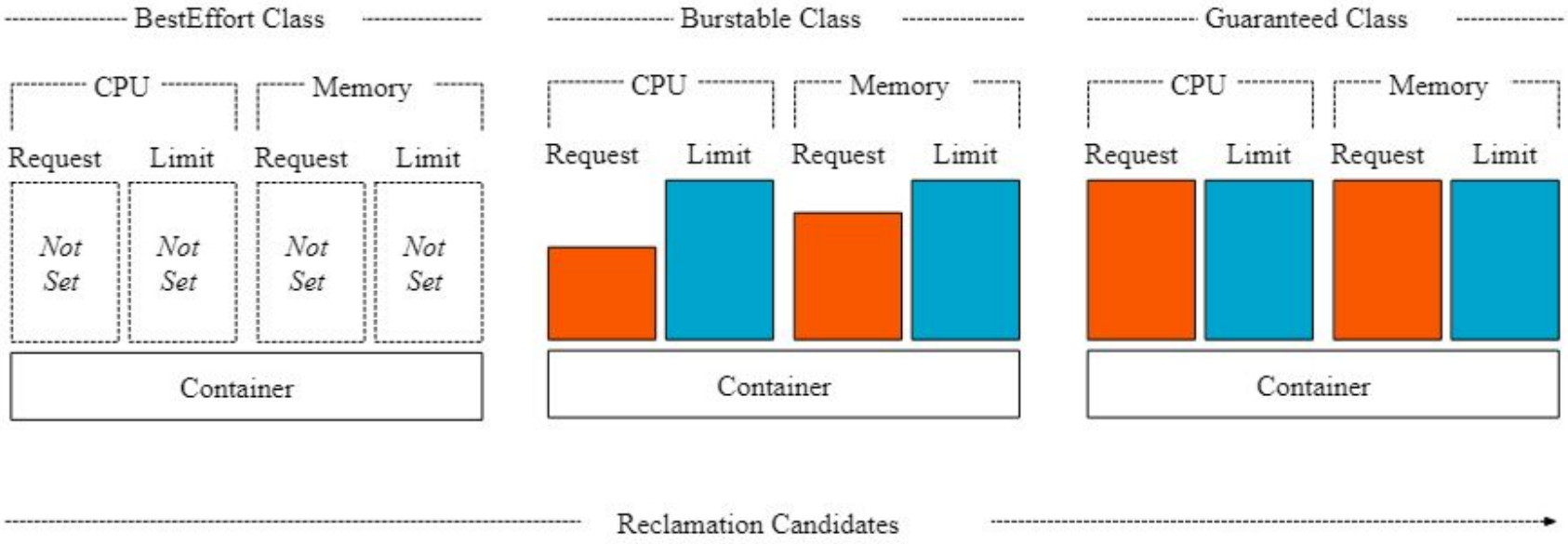
Reference:

- [k8s] 11. Namespace, ResourceQuota, LimitRange - 실습

Quality of Service for Pods

- Guaranteed
- Burstable
- BestEffort

Scheduling vSphere Pods



Reference:

- [Scheduling vSphere Pods]

Node out of memory behavior

Node out of memory behavior

If the node experiences an *out of memory* (OOM) event prior to the kubelet being able to reclaim memory, the node depends on the oom_killer to respond.

The kubelet sets an `oom_score_adj` value for each container based on the QoS for the pod.

Quality of Service	oom_score_adj
Guaranteed	-997
BestEffort	1000
Burstable	*min(max(2, 1000 - (1000 × memoryRequestBytes) / machineMemoryCapacityBytes), 999)*

Reference:

- Node out of memory behavior

Application Output

- Cloud Native Observability -

Jumpbox®

Tree Output

Tree

Tree Output (Cont.)

Tree

Gas

Tree Output (Cont.)

Tree

Gas

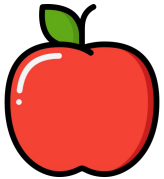
Leaf

Tree Output (Cont.)

Tree

Gas

Leaf



Fruit

Application Output

Application Output (Cont.)

App

Application Output (Cont.)

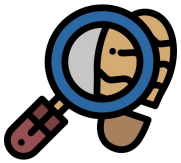
Logs

App

Application Output (Cont.)

App

Logs

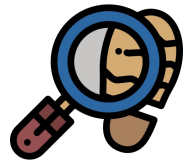


Traces

Application Output (Cont.)

App

Logs

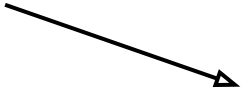


Traces

Metrics

Application Output (Cont.)

App



Logs

Traces

Metrics

Grafana loki

elasticsearch

Application Output (Cont.)

App

Logs

Traces

Metrics

Grafana loki

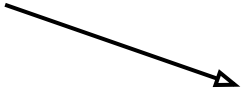
elasticsearch

Grafana Tempo

elastic apm

Application Output (Cont.)

App



Logs

Traces

Metrics

Grafana loki

elasticsearch

Grafana Tempo

elastic apm

Grafana Mimir

Prometheus

Jumpbox®

Auto Scale

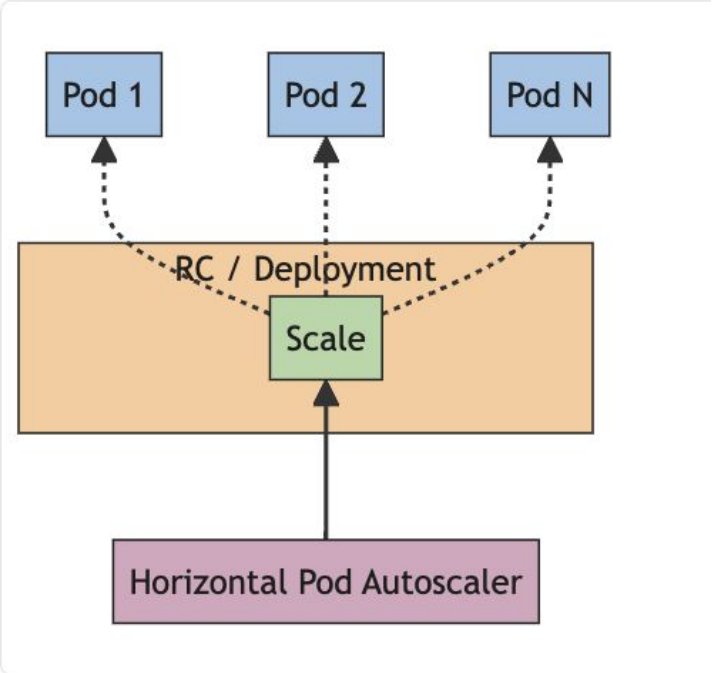
Reference Pictures:

- [Tune Your Paid Search Campaigns To Account For Holiday Volatility]

Jumpbox®

Horizontal Pod Autoscaler

How does the Horizontal Pod Autoscaler work?



Reference Pictures:

- How does a HorizontalPodAutoscaler work?

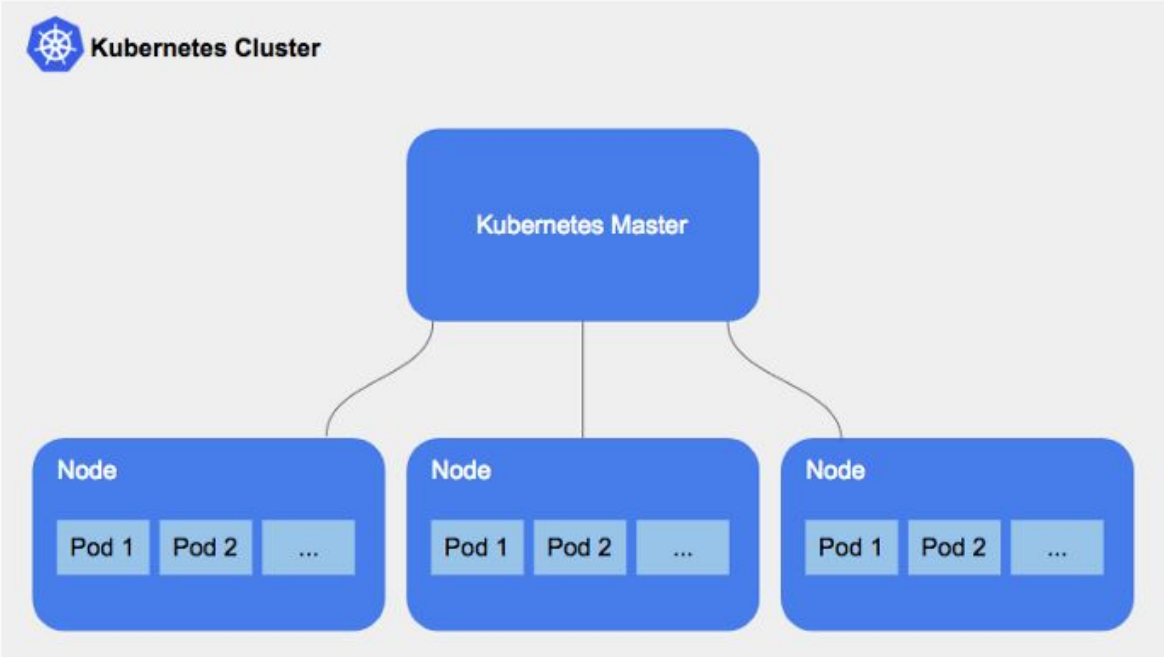
HPA

With Real Action Simulator

[Click Here]

Node Auto Scale

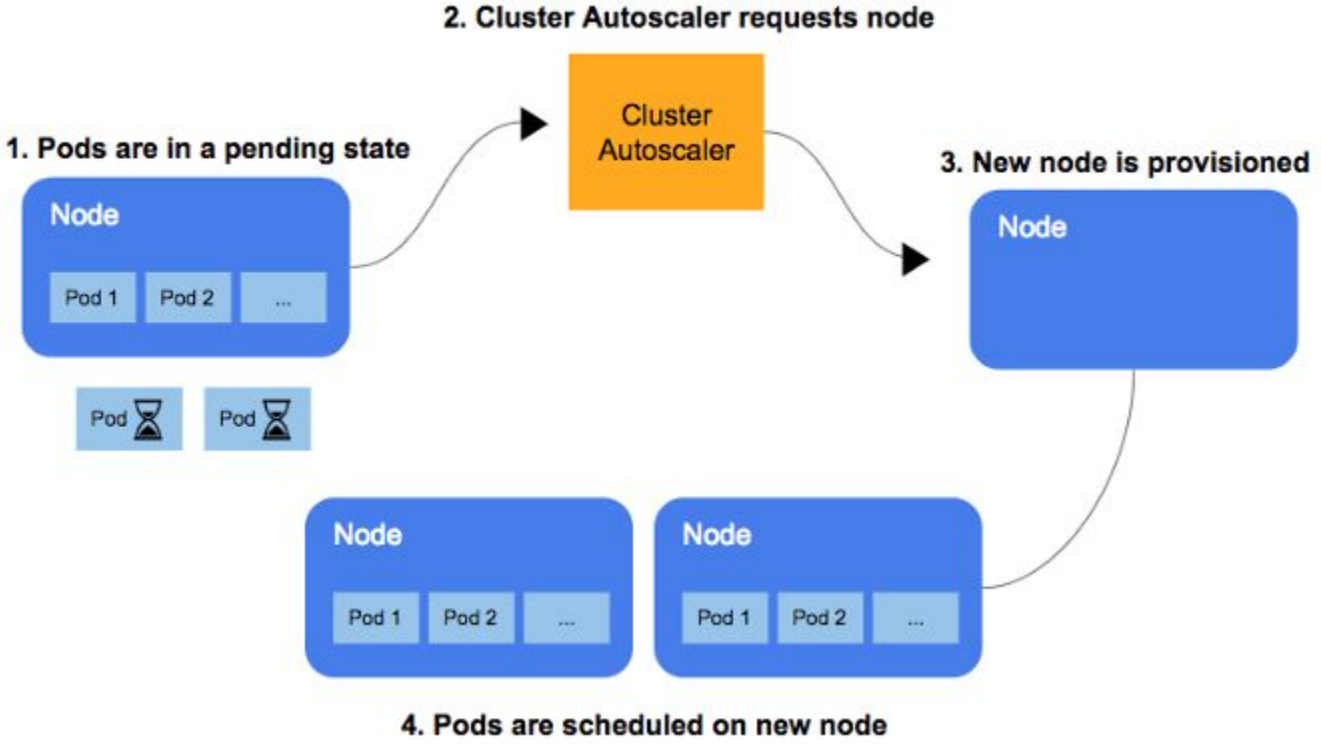
A high-level overview of a Kubernetes cluster



Reference Pictures:

- [Understanding Kubernetes Cluster Autoscaling]

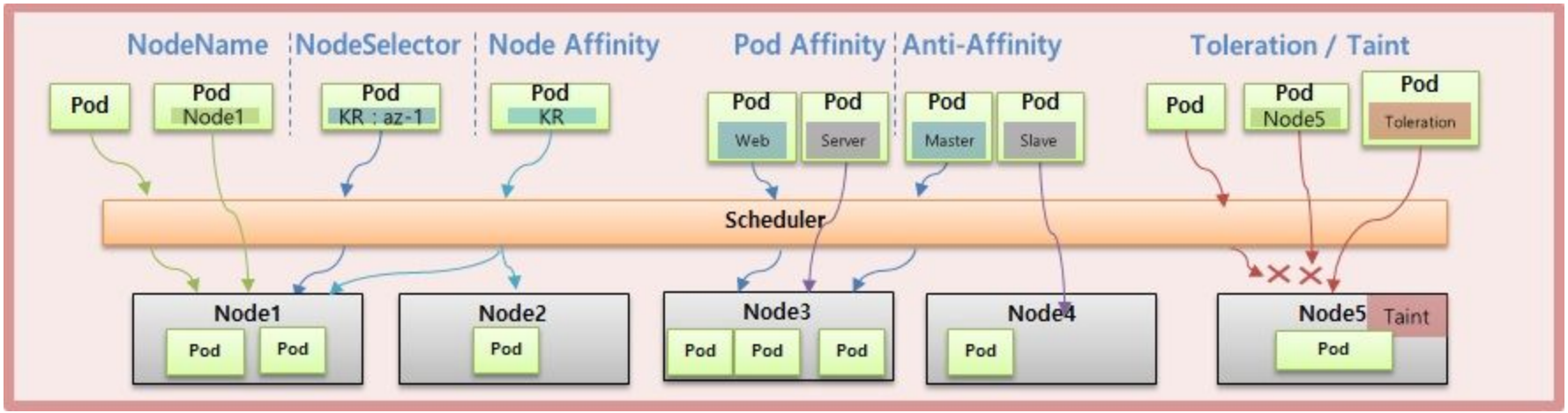
A high-level overview of a Kubernetes cluster



Reference Pictures:

- [Understanding Kubernetes Cluster Autoscaling]

Pod - Node Scheduling



Reference Pictures:

- https://kubetm.github.io/practice/intermediate/pod-node_scheduling/

Node Labels Populated By The Kubelet

Kubernetes nodes come pre-populated with a standard set of labels.

You can also set your own labels on nodes, either through the kubelet configuration or using the Kubernetes API.

Preset labels

The preset labels that Kubernetes sets on nodes are:

- `kubernetes.io/arch`
- `kubernetes.io/hostname`
- `kubernetes.io/os`
- `node.kubernetes.io/instance-type` (if known to the kubelet – Kubernetes may not have this information to set the label)
- `topology.kubernetes.io/region` (if known to the kubelet – Kubernetes may not have this information to set the label)
- `topology.kubernetes.io/zone` (if known to the kubelet – Kubernetes may not have this information to set the label)

Reference Pictures:

- Node Labels Populated By The Kubelet

Affinity and Anti-Affinity

Affinity & Anti-Affinity

โดย Affinity Rules มี 2 แบบคือ

- `Affinity` คือ pods เหล่านี้ต้องอยู่ใน location เดียวกัน (co-location) มักจะเป็น pod ที่ share data กันเป็นจำนวนมาก
- `Anti-affinity` คือ pods เหล่านี้ต้องอยู่คนละ location กัน เหมาะกับ pods ที่ต้องการ fault tolerance

Operations ที่สามารถใช้ในการระบุ labels เช่น

- `In`: เลือก pod ที่มี labels ที่มีค่าตามที่ระบุไว้
- `NotIn`: เลือก pod ที่ไม่มี labels ที่มีค่าตามที่ระบุไว้
- `Exists`: เลือก pod ที่มี labels ที่ระบุ
- `DoesNotExist`: เลือก pod ที่ไม่มี labels ที่ระบุ

นอกจากนี้ ยังมีการระบุรายละเอียดการ schedule pods ลงไปอีก 2 parameters คือ

- `requiredDuringSchedulingIgnoredDuringExecution`: ต้องมี labels ตาม conditions ที่กำหนดเท่านั้น ถ้าไม่มี ไม่ schedule (hard affinity)
- `preferredDuringSchedulingIgnoredDuringExecution`: ถ้าไม่มี labels ตาม conditions ที่กำหนด ก็ไม่เป็นไร แต่ถ้ามีก็จะ schedule ตาม conditions ที่กำหนดนั้น (soft affinity)

มากไปกว่านั้น เรายังสามารถใช้ร่วมกับ `topologyKey` ได้อีกด้วย ดังนี้

- ถ้าระบุ `Affinity` พร้อมกับ `Anti-affinity` ที่เป็น `requiredDuringSchedulingIgnoredDuringExecution` ต้องระบุ `topologyKey`
- ถ้าระบุ `Anti-affinity` ที่เป็น `requiredDuringSchedulingIgnoredDuringExecution` เท่านั้น `topologyKey` ต้องเป็น `kubernetes.io/hostname`
- ถ้าระบุ `Anti-affinity` ที่เป็น `preferredDuringSchedulingIgnoredDuringExecution` เท่านั้น `topologyKey` ต้องระบุ `topologyKey`
- นอกเหนือจากข้างบน `topologyKey` เป็นค่าอะไรก็ได้ ใน built-in Labels

Reference Pictures:

- LFS258 [10/15]: Scheduling

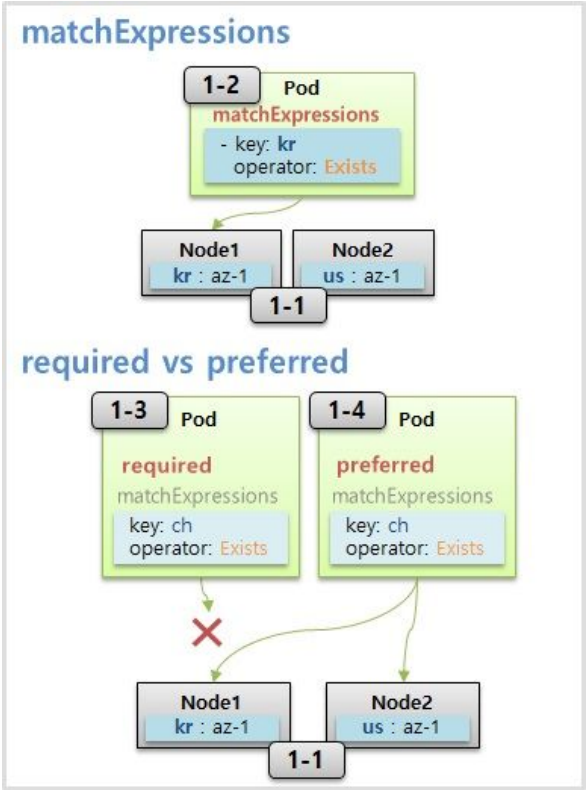
Affinity & Anti-Affinity (Cont.)

มากไปกว่านั้น เรายังสามารถใช้ร่วมกับ `topologyKey` ได้อีกด้วย ดังนี้

- ถ้าระบุ `Affinity` พร้อมกับ `Anti-affinity` ที่เป็น `requiredDuringSchedulingIgnoredDuringExecution` ต้องระบุ `topologyKey`
- ถ้าระบุ `Anti-affinity` ที่เป็น `requiredDuringSchedulingIgnoredDuringExecution` เท่านั้น `topologyKey` ต้องเป็น `kubernetes.io/hostname`
- ถ้าระบุ `Anti-affinity` ที่เป็น `preferredDuringSchedulingIgnoredDuringExecution` เท่านั้น `topologyKey` ต้องระบุ `topologyKey`
- นอกเหนือจากข้างบน `topologyKey` เป็นค่าอะไรก็ได้ ใน built-in Labels

Reference Pictures:
- LFS258 [10/15]: Scheduling

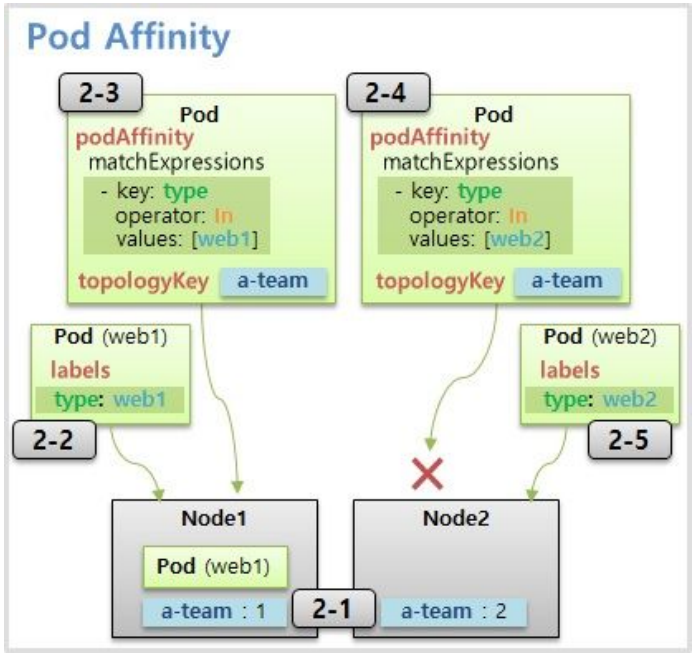
Node Affinity



Reference Pictures:

- https://kubetm.github.io/practice/intermediate/pod-node_scheduling/

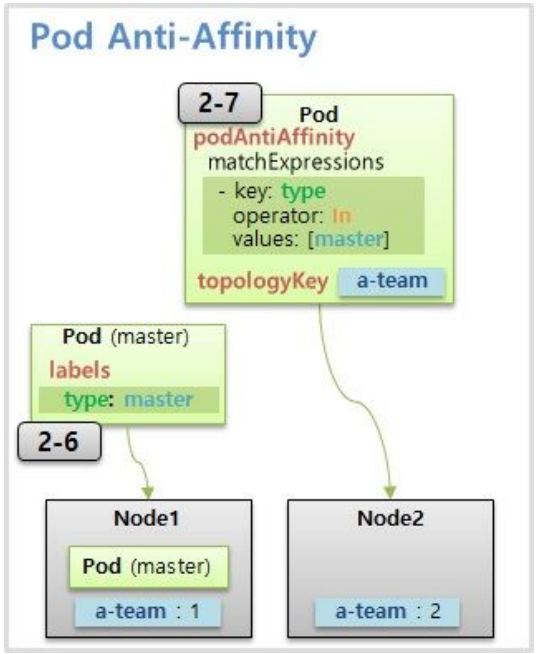
Pod Affinity



Reference Pictures:

- https://kubetm.github.io/practice/intermediate/pod-node_scheduling/

Pod Anti-Affinity

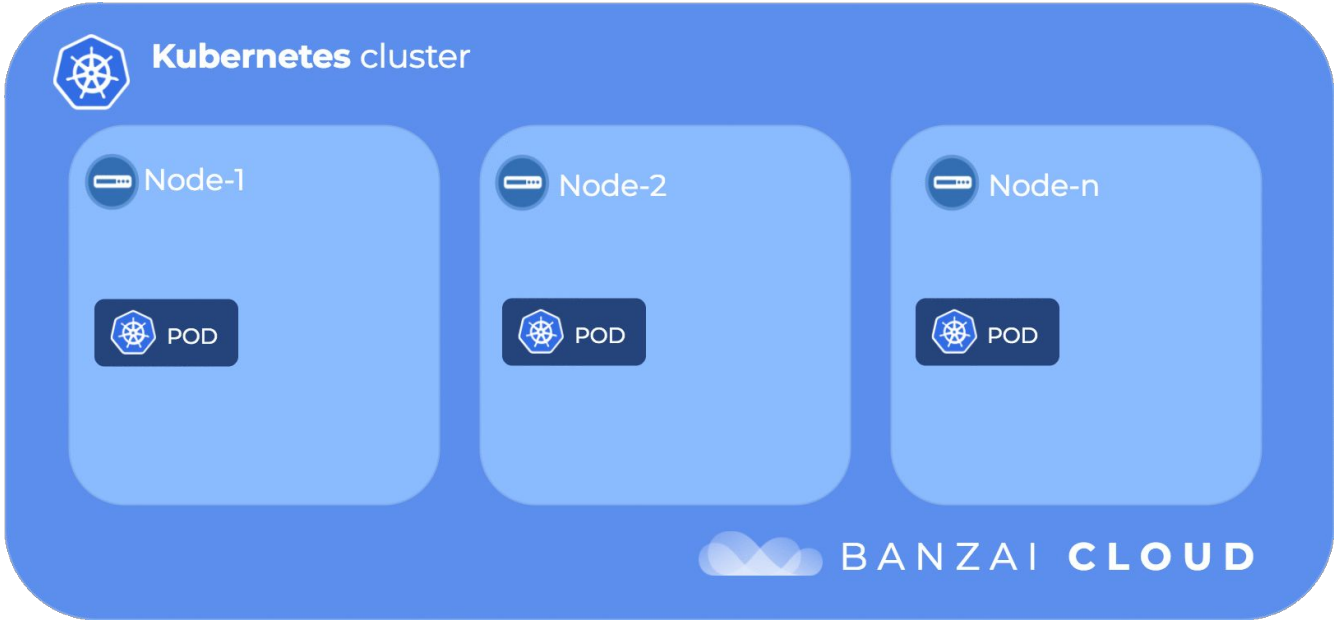


Reference Pictures:

- https://kubetm.github.io/practice/intermediate/pod-node_scheduling/

Taints and Tolerations

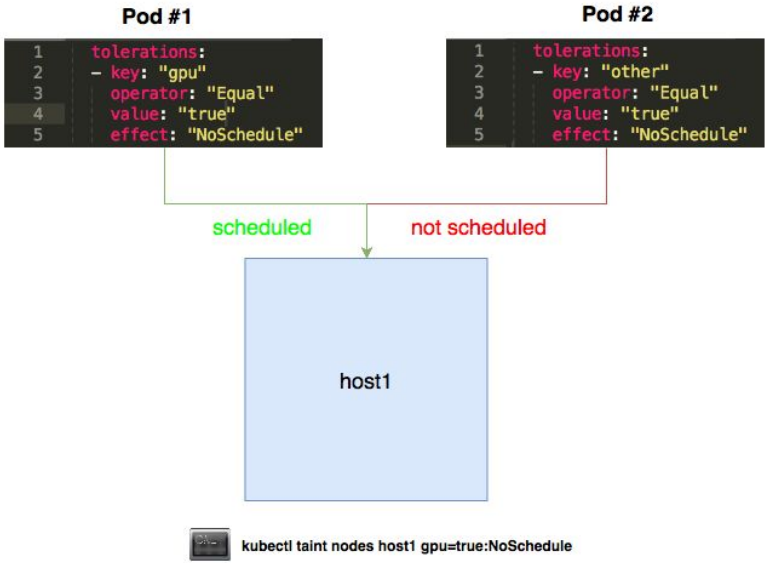
Taints and Tolerations



Reference Pictures:

- https://banzaicloud.com/blog/k8s-taints-tolerations-affinities/

Taints and Tolerations Use case



Reference Pictures:

- [Making Sense of Taints and Tolerations in Kubernetes]

Cert-manager

Now SSL from other

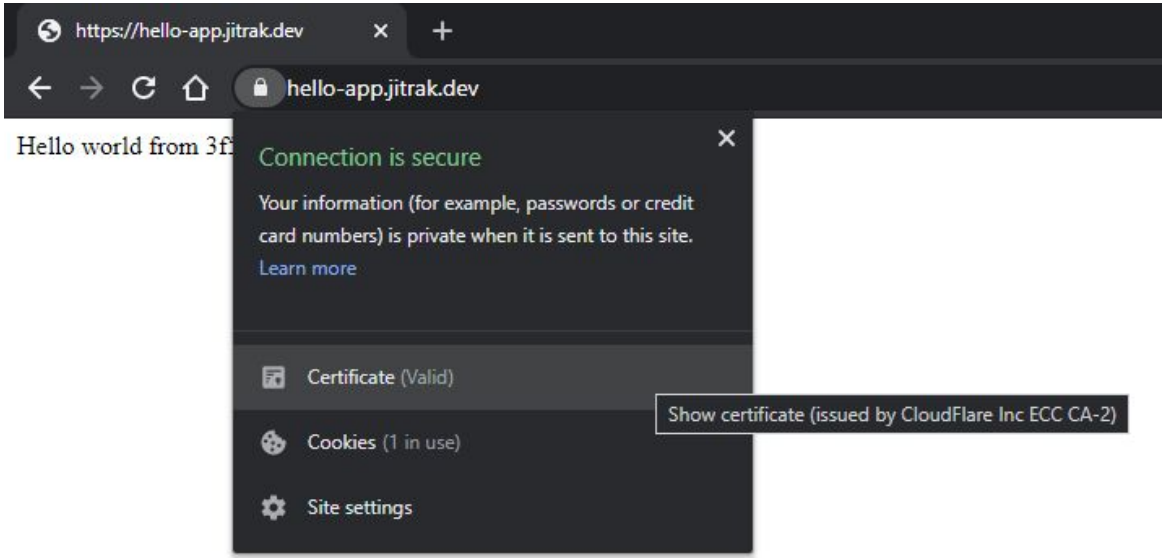
Your SSL/TLS encryption mode is Full

This setting was last changed a few seconds ago

- ○ Off (not secure)
  No encryption applied
- ○ Flexible
  Encrypts traffic between the browser and Cloudflare
- ◉ **Full**
  Encrypts end-to-end, using a self signed certificate on the server
- ○ Full (strict)
  Encrypts end-to-end, but requires a trusted CA or Cloudflare Origin CA certificate on the server

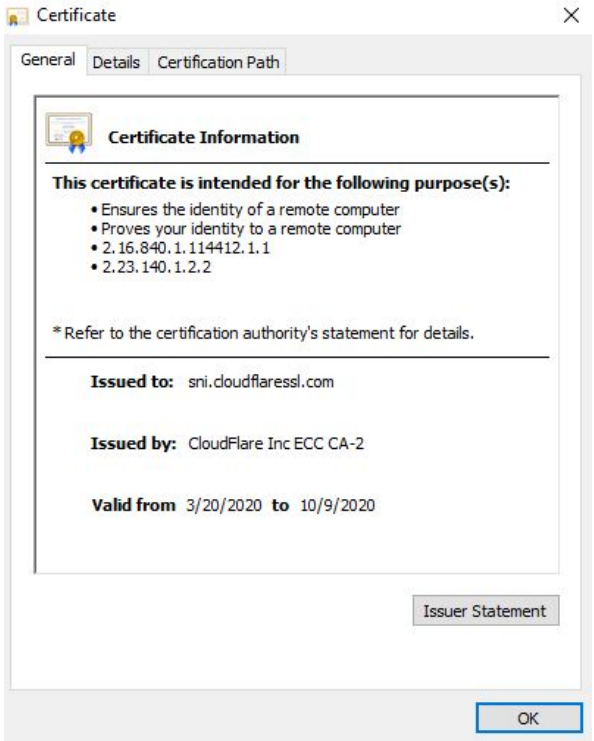
Learn more about End-to-end encryption with Cloudflare

Now SSL from other

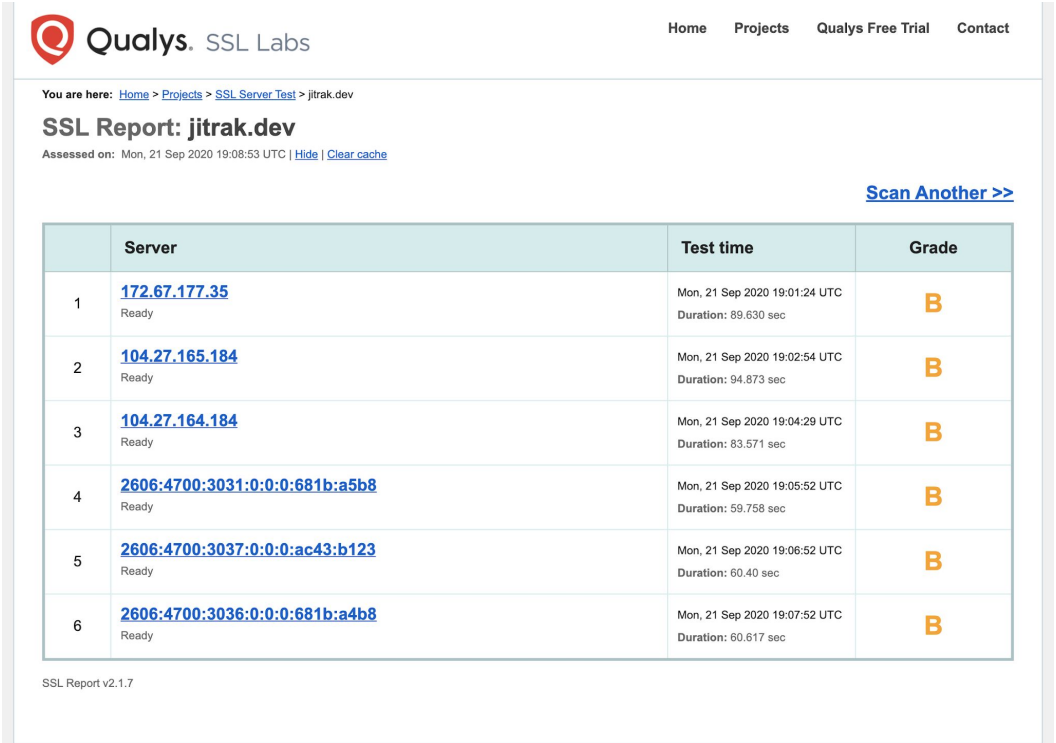


Jumpbox®

Now SSL from other (Cont.)



Cloudflare proxy is SSL Grade B



Qualys. SSL Labs

Home Projects Qualys Free Trial Contact

You are here: Home > Projects > SSL Server Test > jitrak.dev

SSL Report: jitrak.dev

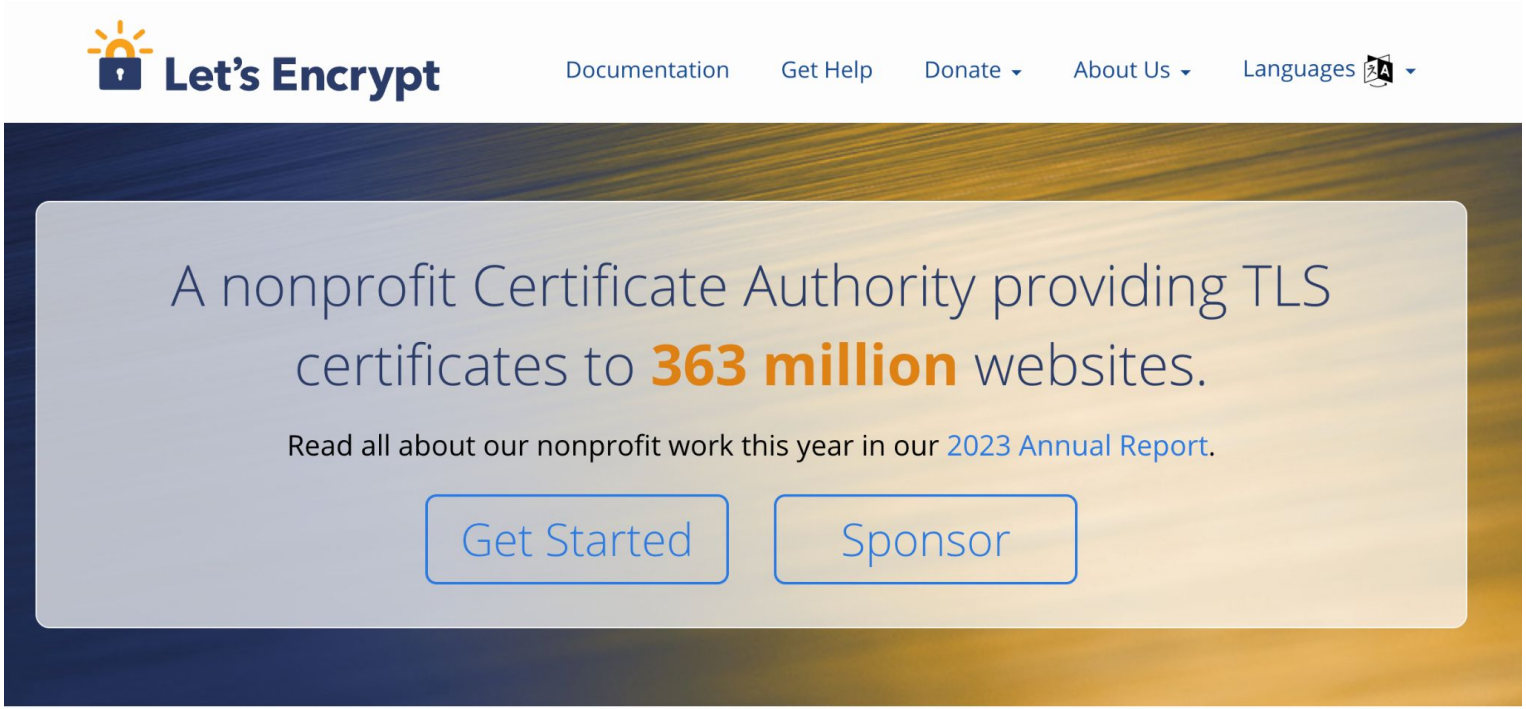
Assessed on: Mon, 21 Sep 2020 19:08:53 UTC | Hide | Clear cache

Scan Another >>

	Server	Test time	Grade
1	172.67.177.35 Ready	Mon, 21 Sep 2020 19:01:24 UTC Duration: 89.630 sec	B
2	104.27.165.184 Ready	Mon, 21 Sep 2020 19:02:54 UTC Duration: 94.873 sec	B
3	104.27.164.184 Ready	Mon, 21 Sep 2020 19:04:29 UTC Duration: 83.571 sec	B
4	2606:4700:3031:0:0:0:681b:a5b8 Ready	Mon, 21 Sep 2020 19:05:52 UTC Duration: 59.758 sec	B
5	2606:4700:3037:0:0:0:ac43:b123 Ready	Mon, 21 Sep 2020 19:06:52 UTC Duration: 60.40 sec	B
6	2606:4700:3036:0:0:0:681b:a4b8 Ready	Mon, 21 Sep 2020 19:07:52 UTC Duration: 60.617 sec	B

SSL Report v2.1.7

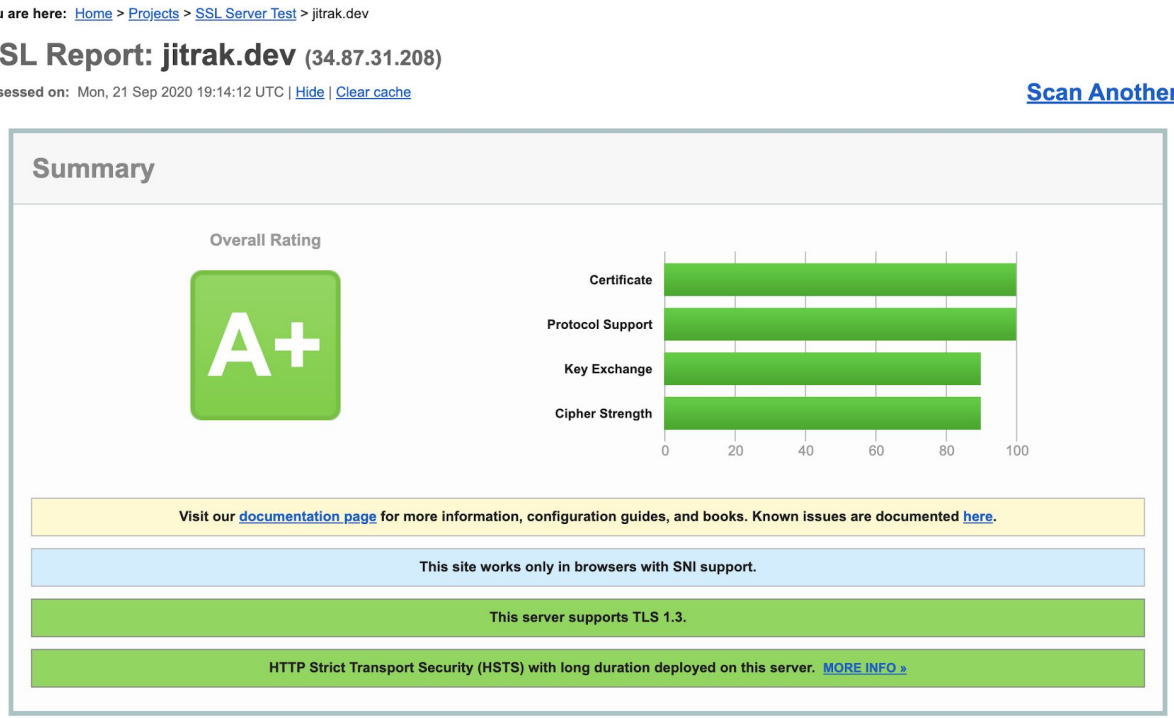
Let's Encrypt



Reference Pictures:

- Let's Encrypt

Let’s Encrypt is SSL Grade A+



SSL Grade

Grades

We wish to provide meaningful and easy-to-understand grades to rate SSL/TLS and PKI configurations. The letter-based A-F grades have proved very successful and we wish to continue using them. Although the meaning of the letter grades can largely be inferred, the original SSL Labs rating guide never defined them, and that caused inconsistencies in how the grades were applied.
We wish to correct that mistake now and specify the meaning behind each grade; they are:

- A+ - exceptional configuration
- A - strong commercial security
- B - adequate security with modern clients, with older and potentially obsolete crypto used with older clients; potentially smaller configuration problems
- C - obsolete configuration, uses obsolete crypto with modern clients; potentially bigger configuration problems
- D - configuration with security issues that are typically difficult or unlikely to be exploited, but can and should be addressed
- E - unused
- F - exploitable and/or patchable problems, misconfigured server, insecure protocols, etc.

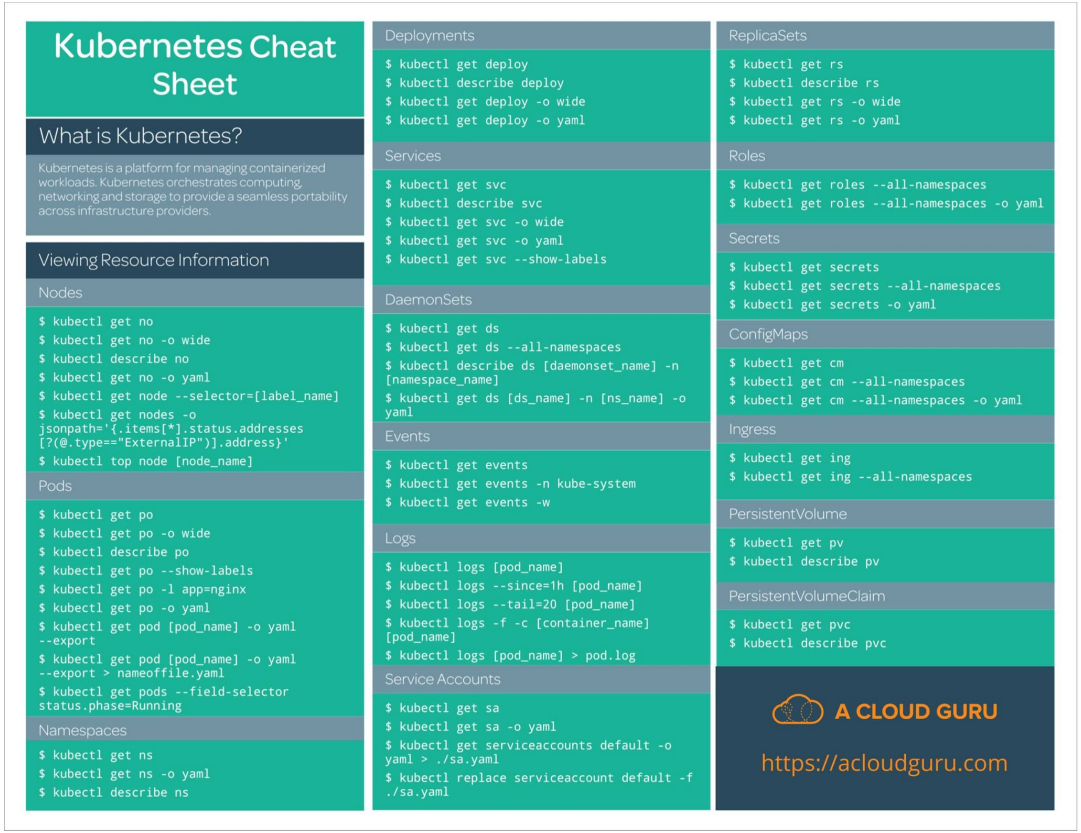
Reference Pictures:

- SSL Labs Grading 2018

GitOps

Reference Pictures:

- https://www.linkedin.com/pulse/everything-you-need-know-gitops-chaitanya-jawale/



Kubernetes Cheat Sheet

What is Kubernetes?

Kubernetes is a platform for managing containerized workloads. Kubernetes orchestrates computing, networking and storage to provide a seamless portability across infrastructure providers.

Viewing Resource Information

Nodes

```
$ kubectl get no
$ kubectl get no -o wide
$ kubectl describe no
$ kubectl get no -o yaml
$ kubectl get node --selector=[label_name]
$ kubectl get nodes -o jsonpath='{.items[*].status.addresses[?(@.type=="ExternalIP")].address}'
$ kubectl top node [node_name]
```

Pods

```
$ kubectl get po
$ kubectl get po -o wide
$ kubectl describe po
$ kubectl get po --show-labels
$ kubectl get po -l app=nginx
$ kubectl get po -o yaml
$ kubectl get pod [pod_name] -o yaml --export
$ kubectl get pod [pod_name] -o yaml --export > nameoffile.yaml
$ kubectl get pods --field-selector status.phase=Running
```

Namespaces

```
$ kubectl get ns
$ kubectl get ns -o yaml
$ kubectl describe ns
```

Deployments

```
$ kubectl get deploy
$ kubectl describe deploy
$ kubectl get deploy -o wide
$ kubectl get deploy -o yaml
```

Services

```
$ kubectl get svc
$ kubectl describe svc
$ kubectl get svc -o wide
$ kubectl get svc -o yaml
$ kubectl get svc --show-labels
```

DaemonSets

```
$ kubectl get ds
$ kubectl get ds --all-namespaces
$ kubectl describe ds [daemonset_name] -n [namespace_name]
$ kubectl get ds [ds_name] -n [ns_name] -o yaml
```

Events

```
$ kubectl get events
$ kubectl get events -n kube-system
$ kubectl get events -w
```

Logs

```
$ kubectl logs [pod_name]
$ kubectl logs --since=1h [pod_name]
$ kubectl logs --tail=20 [pod_name]
$ kubectl logs -f -c [container_name] [pod_name]
$ kubectl logs [pod_name] > pod.log
```

Service Accounts

```
$ kubectl get sa
$ kubectl get sa -o yaml
$ kubectl get serviceaccounts default -o yaml > ./sa.yaml
$ kubectl replace serviceaccount default -f ./sa.yaml
```

ReplicaSets

```
$ kubectl get rs
$ kubectl describe rs
$ kubectl get rs -o wide
$ kubectl get rs -o yaml
```

Roles

```
$ kubectl get roles --all-namespaces
$ kubectl get roles --all-namespaces -o yaml
```

Secrets

```
$ kubectl get secrets
$ kubectl get secrets --all-namespaces
$ kubectl get secrets -o yaml
```

ConfigMaps

```
$ kubectl get cm
$ kubectl get cm --all-namespaces
$ kubectl get cm --all-namespaces -o yaml
```

Ingress

```
$ kubectl get ing
$ kubectl get ing --all-namespaces
```

PersistentVolume

```
$ kubectl get pv
$ kubectl describe pv
```

PersistentVolumeClaim

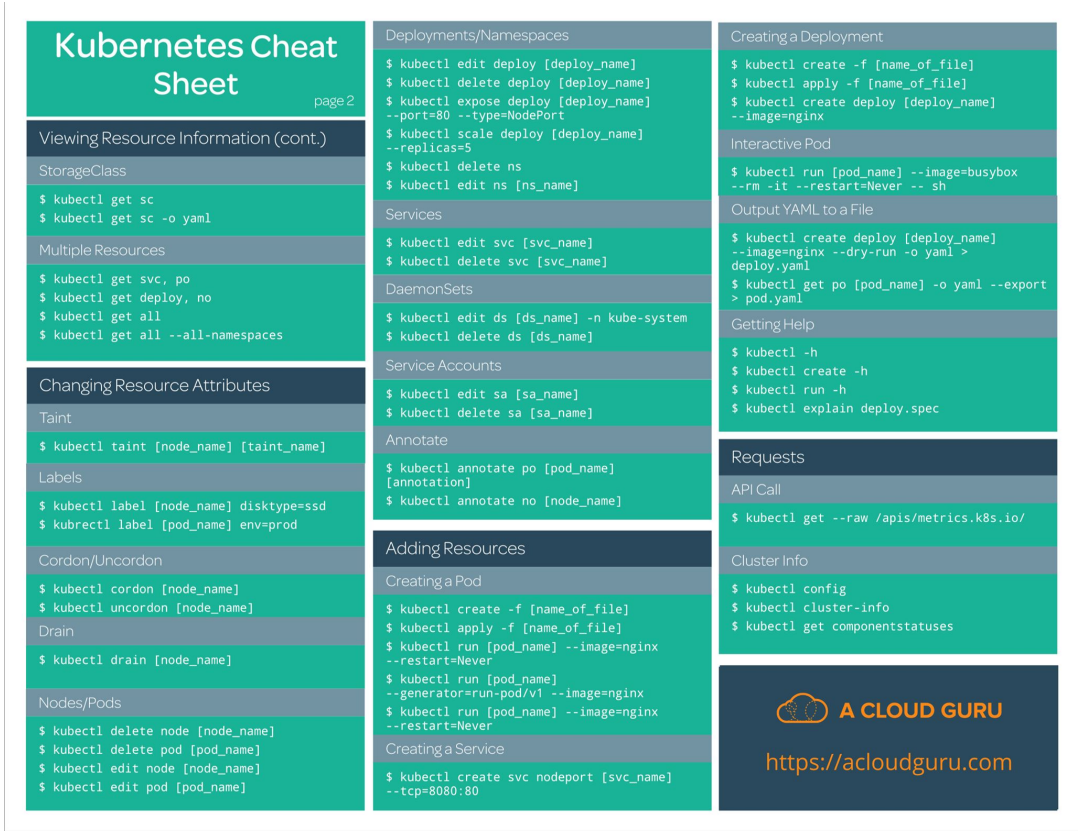
```
$ kubectl get pvc
$ kubectl describe pvc
```

A CLOUD GURU

https://acloudguru.com

Reference Pictures:

- The ultimate Kubernetes CHEAT SHEET! Save and share



Reference Pictures:

- [The ultimate Kubernetes CHEAT SHEET! Save and share]

Ingress - Logical Mapping

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 annotations:
   nginx.ingress.kubernetes.io/backend-protocol: HTTPS
   nginx.ingress.kubernetes.io/rewrite-target: /$2
 name: nginx
spec:
 ingressClassName: nginx
 rules:
 - http:
     paths:
     - backend:
         service:
           name: nginx
           port:
             number: 80
       path: /
       pathType: Prefix
```

```yaml
apiVersion: v1
kind: Service
metadata:
 name: nginx
spec:
 selector:
   app: nginx
 ports:
 - port: 80
   protocol: TCP
   targetPort: 80
 type: ClusterIP
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-<rs-hash>-<hash>
  labels:
    app: nginx
spec:
  containers:
  -  image: nginx
     name: nginx
     ports:
     - containerPort: 80
     resources: {}
     restartPolicy: Always
```

ช่วยส่ง **Feedback** ให้กับผู้สอน เพื่อเราได้พัฒนาตัวเองให้ดีขึ้นครับ... 👩‍💻

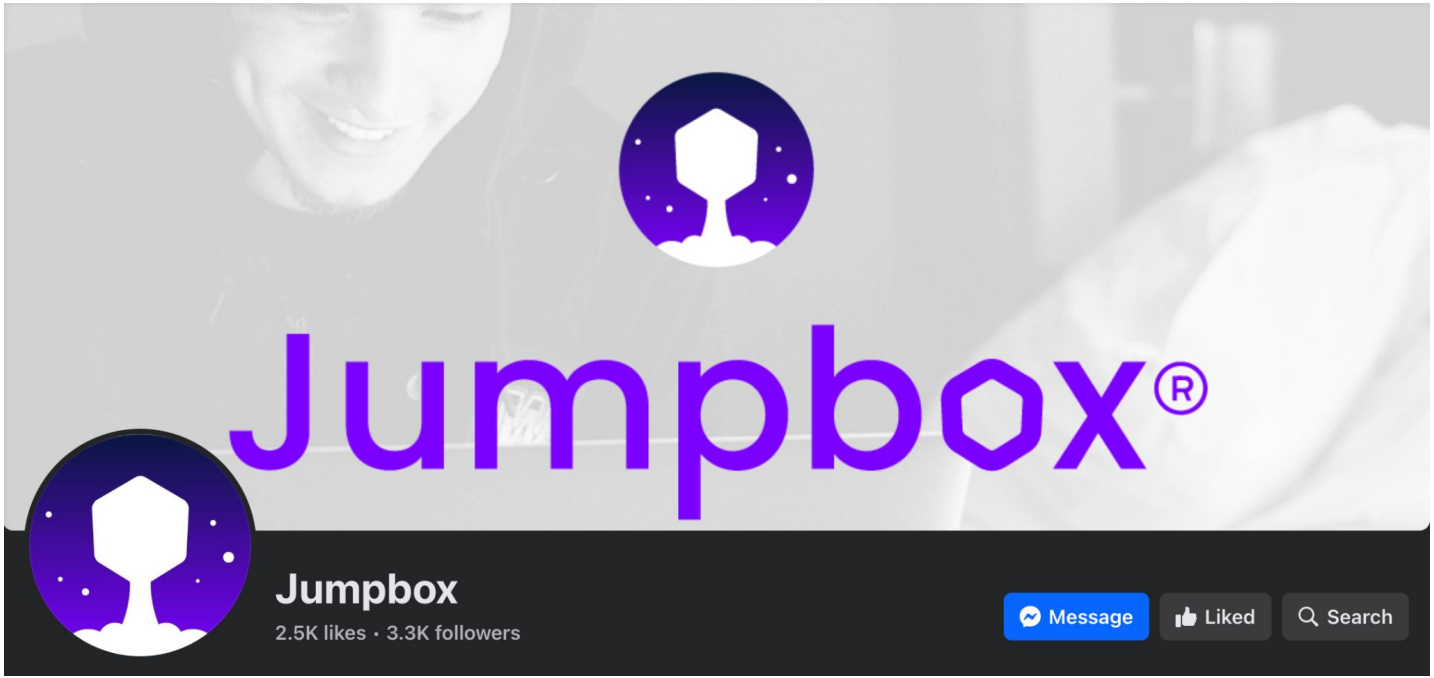
Feedback Link

Jumpbox®

Jumpbox®

Tech Passion | Sharing | Society

facebook



https://www.facebook.com/jumpbox.academy

Jumpbox®

YouTube

Jumpbox

@jumpbox.academy · 1.92K subscribers · 30 videos

More about this channel ...more

https://www.youtube.com/@jumpbox.academy

Contact Us

Jumpbox

@jumpbox

admin@jumpbox.co

063-245-2168 (JoJo)

062-796-1559 (Beau)

Jumpbox®

"เราเชื่อว่าการเรียนรู้ทำให้ชีวิตคุณดีขึ้น"

-Jumpbox Team-

Jumpbox®

Jumpbox®