

**Lab Manual- Kubernetes Thinking**

**Prepared for**:

**Date:** 18th Dec 2023

**Prepared by:**

Document Name: Lab Manual **Document Number** AZLabn916

**Contributor:**

Contents

[1. Objective 3](#_Toc162871884)

[1. POD 3](#_Toc162871885)

[2. Problems with Pods (Why Use ReplicaSets instead?) 4](#_Toc162871886)

[3. Kubernetes Objects 4](#_Toc162871887)

# Objective

1. Pods manage containers
2. ReplicaSets manage Pods
3. Services expose pod processes to the outside world
4. ConfigMaps and Secrets help you configure Pods
5. Labels are the plumbing that ties everything together (not covered in detail here)
6. Deployments manage the change from one set of replicasets/pods to another (e.g. for a software release)

## POD

Containers that belong together should be grouped into a pod. It's tempting to think of your Kubernetes **nodes** in the way you thought about 'machines' in the past, but that's not correct. **Pods** are the closest analogue to the traditional **machine** concept:

Containers in the same pod get scheduled together on the same node and have access to:

* shared network namespace
* shared hostname (UTS namespace)
* shared IPC namespace
* storage
* the only private thing containers in the same pod have is their own cgroup

The big difference is that different Pods, **EVEN THOSE THAT ARE SCHEDULED ON THE SAME NODE**, are isolated from each other. They don't share a hostname, IP, or IPC namespace.

All of these components will need to be scaled separately, and in different ways (a stateless **webapp** can easily scale **horizontally** -- run **20 extra copies of it!** -- while your **database** might need to scale **vertically** -- just give **more memory and CPU** time to your single copy of the postgres process).

## Problems with Pods (Why Use ReplicaSets instead?)

* if a node fails after a pod is scheduled, that pod is not rescheduled
* you can't launch multiple running versions of the same pod
* once a pod is scheduled onto a node, it doesn't move

## Kubernetes Objects

* represented as JSON or YAML files
* you create these and then push them to the kubernetes API with kubectl
* (or receive them as output from kubectl after it hits the kubernetes API for you)

Regardless of the type of object you're dealing with, the basic syntax is:

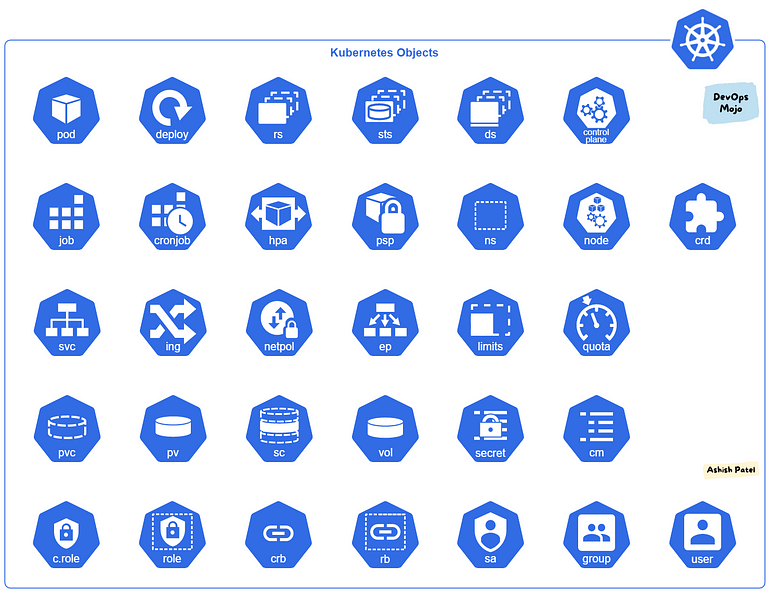
**kubectl apply -f obj.yaml**

You can **interactively** edit using

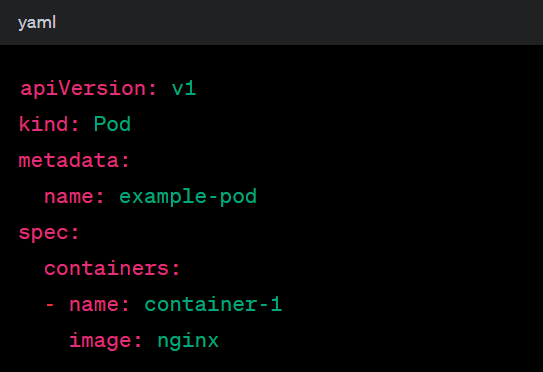
**kubectl edit <resource-name> <obj-name>**

This will:

1. download a yaml representation of the object using the kubernetes API
2. launch your $EDITOR on that textfile
3. re-upload your saved changed using the Kubernetes API, which will re-evaluate that object definition and make the necessary changes.



1. **Pod:**
   * Represents the smallest deployable units in the cluster. It encapsulates one or more containers.



1. **ReplicaSet:**
   * Maintains a stable set of replica Pods running at any given time.
2. **Deployment:**
   * Manages the deployment of a set of identical pods. It manages the creation and scaling of Pods.



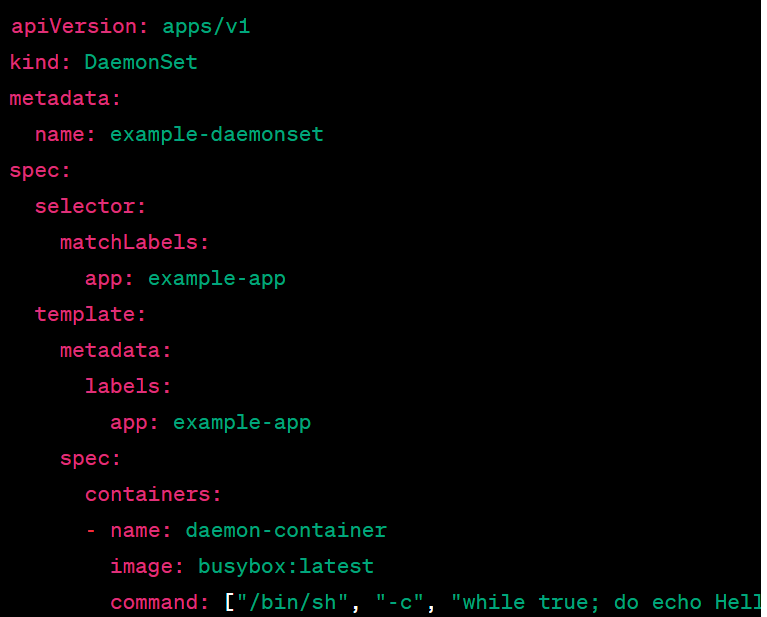
1. **StatefulSet:**
   * Manages stateful applications by maintaining a unique identifier for each Pod.

In this example, the StatefulSet (**example-statefulset**) ensures that three replicas of the Nginx container are running. It uses a volume claim template to create PersistentVolumeClaims with unique names for each replica.

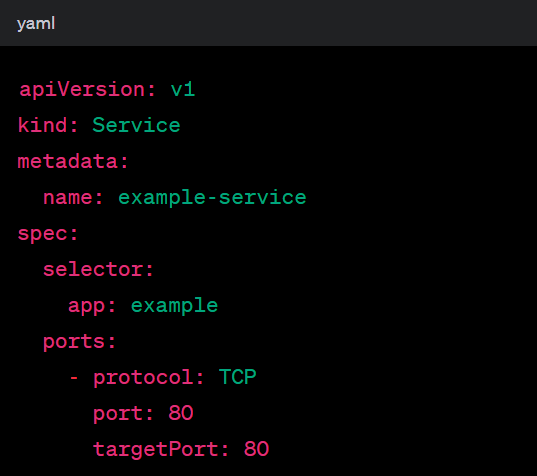


1. **DaemonSet:**
   * Ensures that all (or some) nodes run a copy of a Pod.

In this DaemonSet example (**example-daemonset**), it ensures that a copy of the **busybox** container runs on each node in the cluster. The container continuously outputs a message every 10 seconds.



1. **Service:**
   * Exposes a set of Pods as a network service with a stable IP address and DNS name.

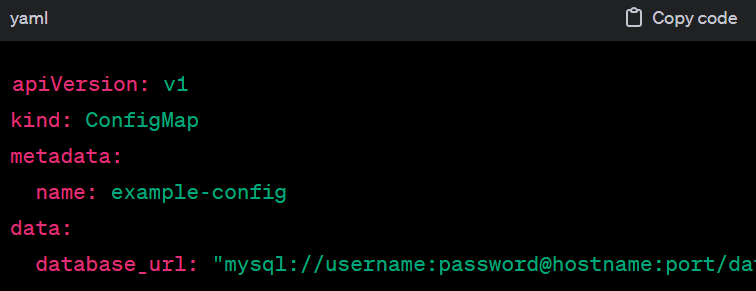


1. **Ingress:**
   * Manages external access to services, typically HTTP.

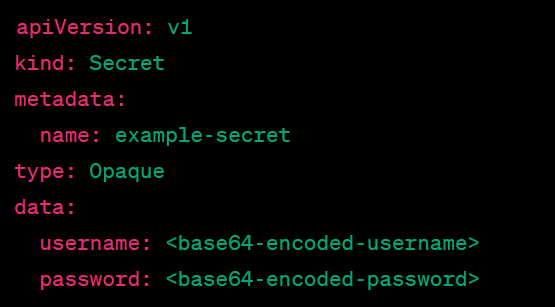
In this example, the Ingress resource defines a rule for routing traffic with the host **example.com**. Requests to **/app** will be forwarded to the service named **app-service** on port 80.



1. **ConfigMap:**
   * Holds configuration data as key-value pairs for Pods to consume.

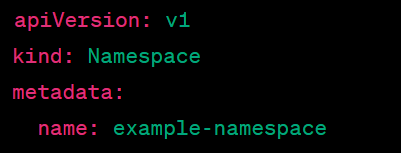


1. **Secret:**
   * Stores sensitive information, such as authentication credentials, in base64-encoded format.



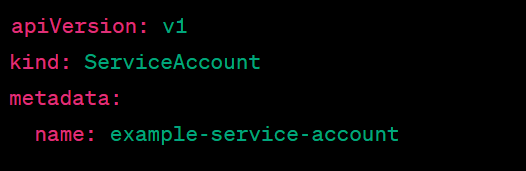
1. **Namespace:**
   * Provides a way to divide cluster resources between multiple users or teams.

This simple YAML file creates a Kubernetes Namespace named **example-namespace**.



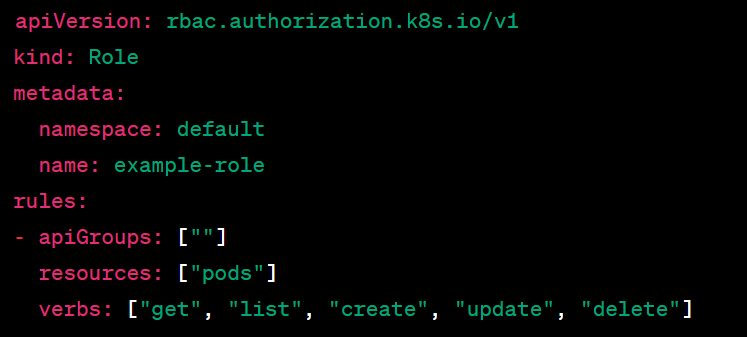
1. **ServiceAccount:**
   * Defines a set of permissions for accessing cluster resources.

This YAML file creates a Kubernetes **ServiceAccount** named **example-service-account**. ServiceAccounts are used to provide an identity to a Pod.



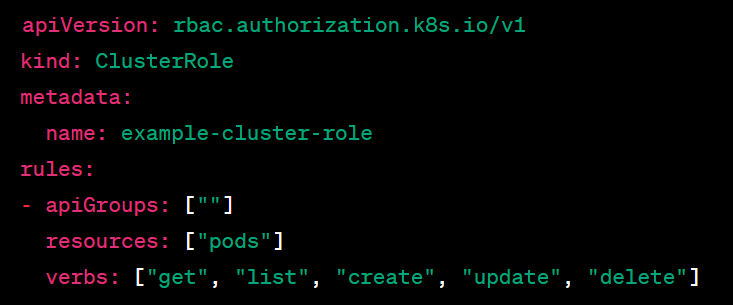
1. **Role:**
   * Defines a set of permissions within a namespace.

This Role example grants permissions in the default namespace to **get, list, create, update**, and **delete** Pods.



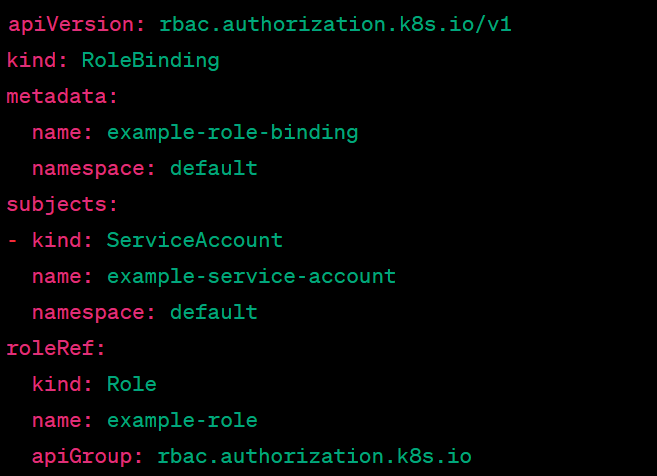
1. **ClusterRole:**
   * Defines a set of permissions cluster-wide.

This ClusterRole example grants cluster-wide permissions to **get, list, create, update**, and **delete** Pods.



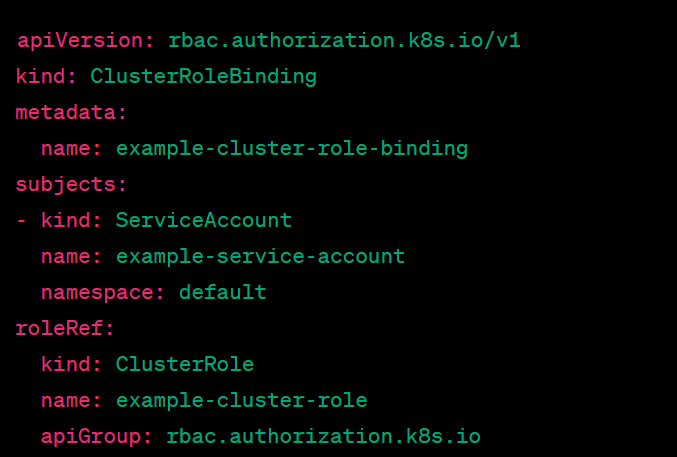
1. **RoleBinding:**
   * Binds a role to a user or group within a namespace.

This RoleBinding example binds the **example-role** to the **example-service-account** within the default namespace.



1. **ClusterRoleBinding:**
   * Binds a cluster-wide role to a user or group.

This ClusterRoleBinding example binds the **example-cluster-role** to the **example-service-account** at the cluster level.



1. **PersistentVolume (PV):**
   * Represents a piece of storage in the cluster that has been provisioned by an administrator.

This PersistentVolume example uses **hostPath** to represent a volume on the node. It has a capacity of **1Gi, ReadWriteOnce** access mode, and retains data even when the claim is deleted.



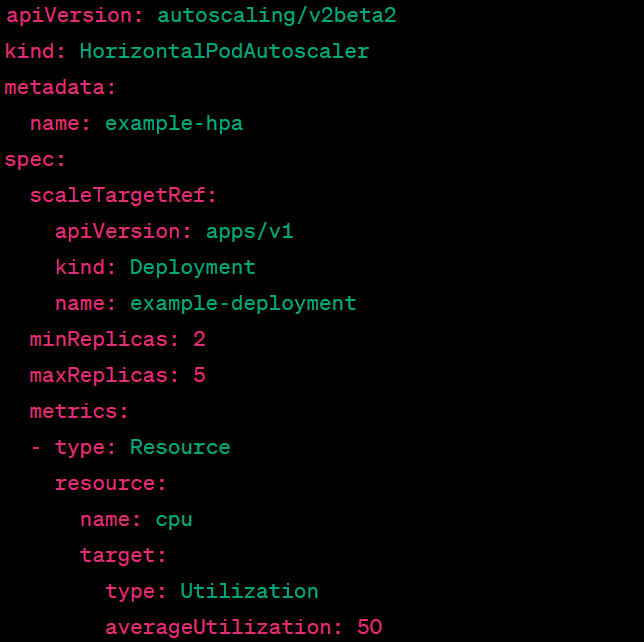
1. **PersistentVolumeClaim (PVC):**
   * Requests storage resources from a PersistentVolume.
2. **StorageClass:**
   * Defines different classes of storage.
3. **Job:**
   * Creates one or more Pods and ensures that a specified number of them successfully terminate.
4. **CronJob:**
   * Creates Jobs on a repeating schedule.

In this example, a CronJob is scheduled to run every minute, and it runs a simple command in a Pod.



1. **HorizontalPodAutoscaler (HPA):**
   * Automatically adjusts the number of Pods in a Deployment or ReplicaSet.

In this example, the HPA is targeting a Deployment (**example-deployment**). It is configured to scale between **2** and **5 replicas** based on the average CPU utilization, maintaining it around 50%.



1. **PodDisruptionBudget (PDB):**
   * Limits the number of disruptions that can happen simultaneously.

This PodDisruptionBudget example ensures that at least 2 Pods labeled with **app: example-app** are available during voluntary disruptions, like those caused by scaling down or updates.

