Kubernetes

Kubernetes Fundamentals 2

Kubernetes Storage

Kubernetes volumes

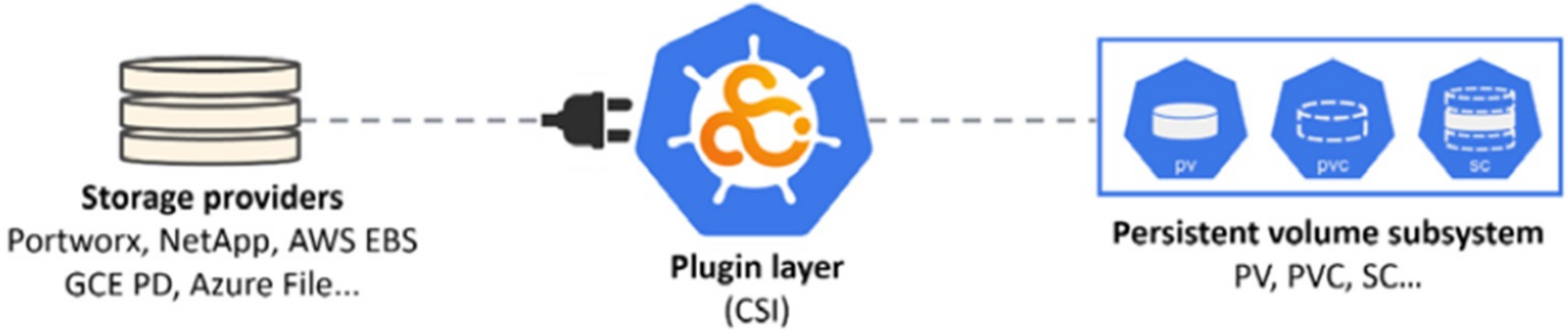
Kubernetes **supports lots of types of storage**.

For example, Internet Small Computer System Interface (iSCSI), Server Message Block (SMB), Network File System (NFS), AWS Elastic Block Store (EBS), etc.

All storage on a Kubernetes cluster is called a volume.

Container Storage Interface (CSI)

Container Storage Interface (CSI) is an open standard aimed at providing a clean **interface for plugins**.



Storage providers
Portworx, NetApp, AWS EBS
GCE PD, Azure File...

Plugin layer
(CSI)

Persistent volume subsystem
PV, PVC, SC...

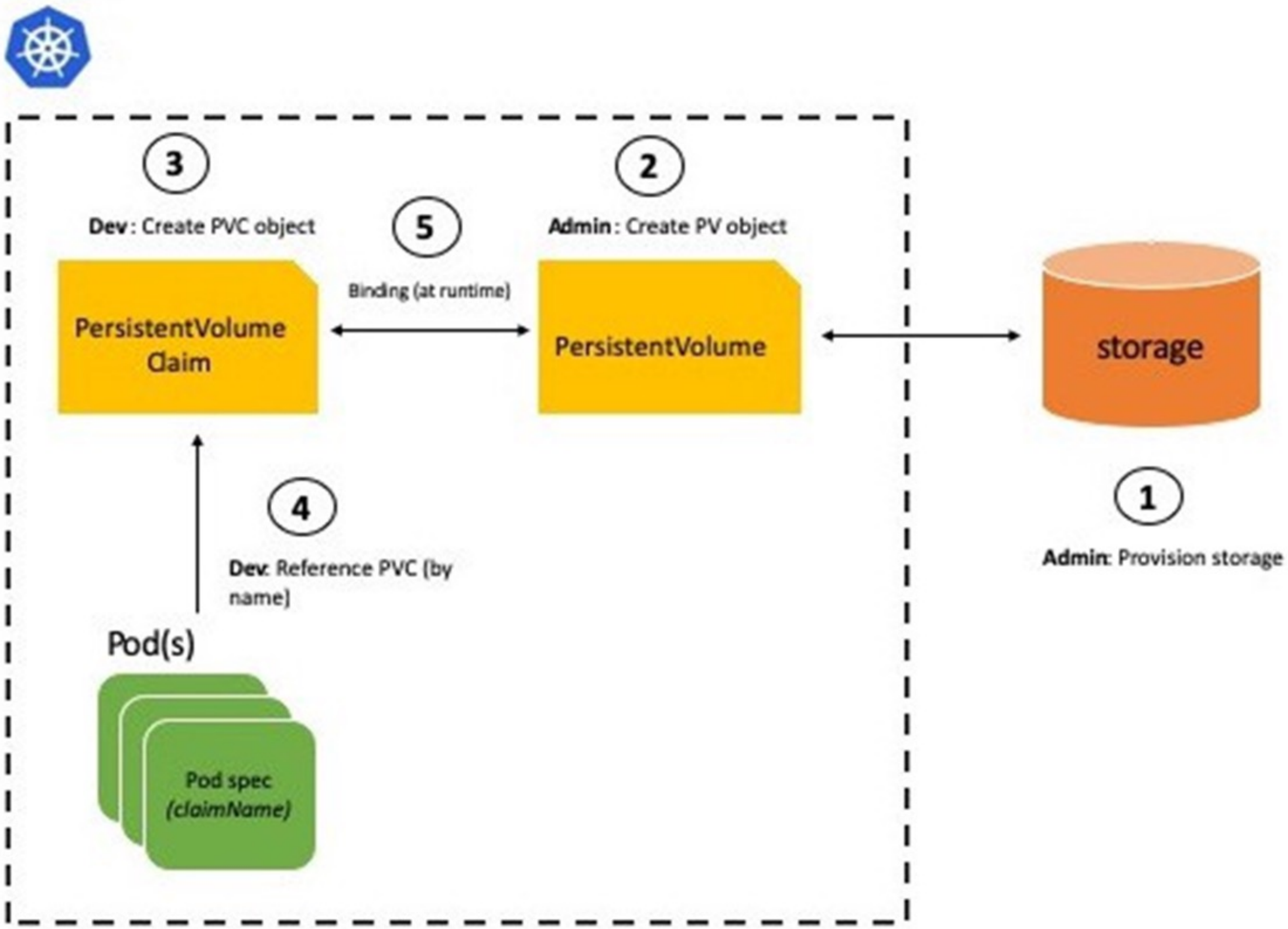
Kubernetes persistent volume subsystem

This is a set of **API objects** that allow applications to consume storage.

PersistentVolumes (PV): map external storage onto the cluster.

Persistent VolumeClaims (PVC): authorize Pods to use a PV.

Storage Classes (SC): enable dynamic provisioning.



3
Dev: Create PVC object
5
Binding (at runtime)
2
Admin: Create PV object
PersistentVolume Claim
PersistentVolume
storage
1
Admin: Provision storage
4
Dev: Reference PVC (by name)
Pod(s)
Pod spec
(claimName)

Storage access modes (spec.accessModes)

ReadWriteOnce (RWO) defines a PV that can only be mounted/bound as read/write by a single PVC. Attempts to bind it via multiple PVCs will fail.

ReadWriteMany (RWM) defines a PV that can be bound as R/W by multiple PVCs. This mode is usually only supported by file and object storage; block storage normally only supports RWO.

ReadOnlyMany (ROM) defines a PV that can be bound by multiple PVCs as read-only.

Reclaim policy (spec.persistentVolumeReclaimPolicy)

Retain – manual reclamation

Recycle – basic scrub (rm -rf /thevolume/*)

Delete

Only NFS and HostPath support recycling. AWS EBS, GCE PD, Azure Disk support deletion.

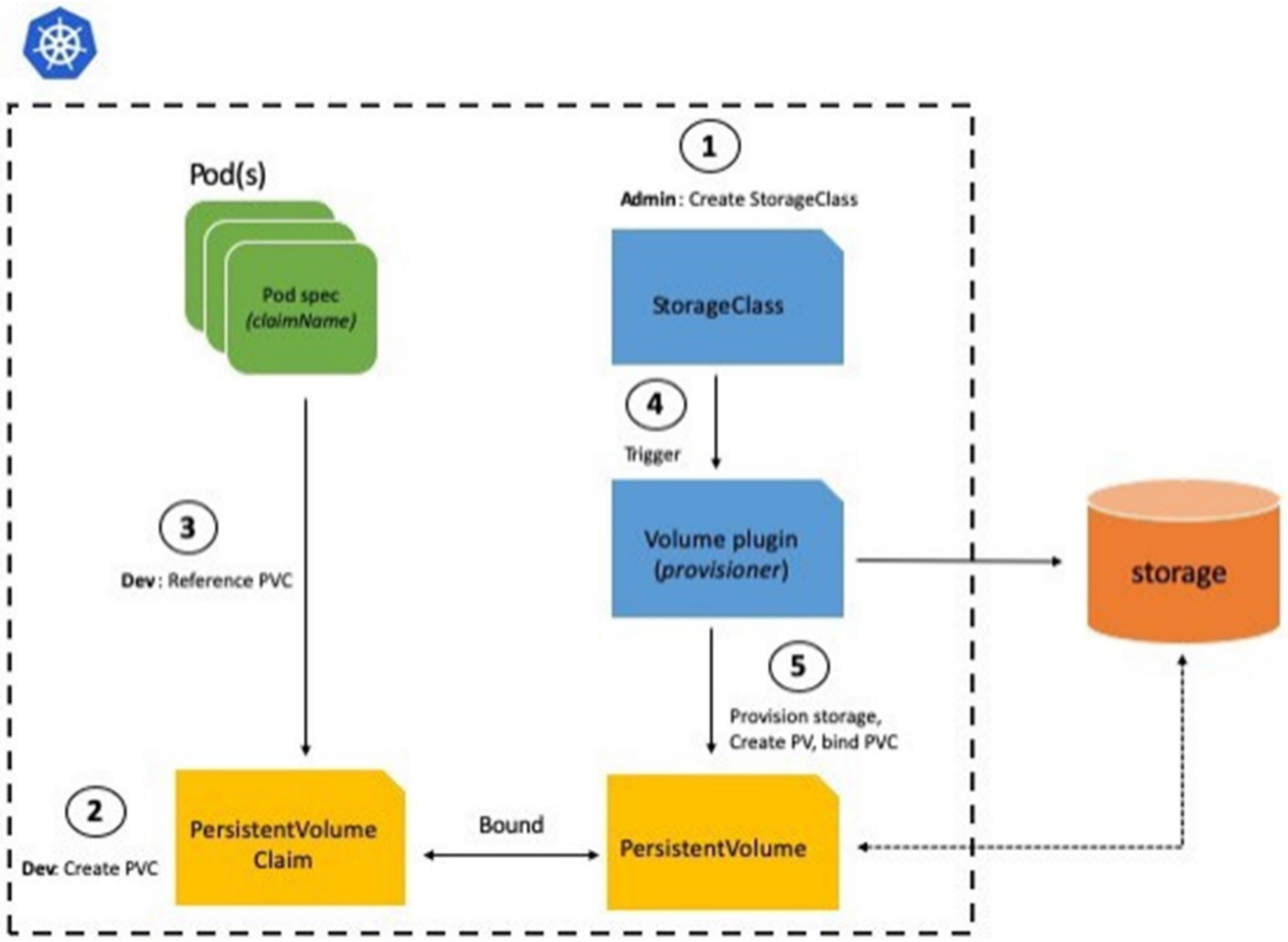
Storage Providers

Kubernetes can use storage from a wide range of external systems.

For example:

Native cloud services such as **AWSElasticBlockStore** or **AzureDisk**.

On-premises storage arrays providing **iSCSI** or **NFS** volumes


Pod(s)
Pod spec
(claimName)
1
Admin : Create StorageClass
StorageClass
4
Trigger
Volume plugin
(provisioner)
storage
3
Dev : Reference PVC
5
Provision storage,
Create PV, bind PVC
2
Dev: Create PVC
PersistentVolume
Claim
Bound
PersistentVolume

Lab: Storage

1) Create PV and PVC and use it in your deployment

2) Install and use SMB CSI driver

3) Install and use NFS provisioner

4) Optional: Install and configure Longhorn distributed block storage

Helm

Helm

Helm the **package manager for Kubernetes**. Helps you manage Kubernetes applications.

Helm Charts help you define, install, and upgrade even the most complex Kubernetes application. See: https://helm.sh/

Artifact Hub is a web-based application that enables finding, installing, and publishing packages and configurations for CNCF projects.
See: https://artifacthub.io/

Lab: Helm

1) Create an empty chart and check the files and structure

2) Install an app with helm

Kubernetes Secrets

A secret volume is **used to pass sensitive information**, such as passwords, to Pods. You can **store secrets in the Kubernetes API and mount them as files for use by Pods**.

Secret volumes are backed by tmpfs (a RAM-backed filesystem) so they are never written to non-volatile storage.

You **must create a secret in the Kubernetes API before you can use it**.

Ref.: https://kubernetes.io/docs/concepts/configuration/secret

Lab: Secrets

1) Create a Secret
2) Use a secret in your deployment

API Access Control

Authenticating

Kubernetes clusters have **two categories of users: service accounts managed by Kubernetes, and normal users**.

Kubernetes does not have objects which represent normal user accounts.

It is assumed that a **cluster-independent service manages normal users.**

Normal user

Even though a normal user cannot be added via an API call, **any user that presents a valid certificate signed by the cluster's certificate authority (CA) is considered authenticated.**

In this configuration, **Kubernetes determines the username from the common name field in the 'subject' of the cert (e.g., "/CN=bob").** From there, the role based access control (RBAC) sub-system would determine whether the user is authorized to perform a specific operation on a resource.

Role-based access control (RBAC)

Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within your organization.

RBAC authorization uses the **rbac.authorization.k8s.io** API group to drive authorization decisions.

The RBAC API declares four kinds of Kubernetes object: **Role**, **ClusterRole**, **RoleBinding** and **ClusterRoleBinding**.

Role and ClusterRole

An RBAC **Role** or **ClusterRole contains rules that represent a set of permissions.** Permissions are purely additive (there are no "deny" rules).

A **Role always sets permissions within a particular namespace**; when you create a Role, you have to specify the namespace it belongs in.

ClusterRole, by contrast, **is a non-namespaced resource.**

RoleBinding and ClusterRoleBinding

A role binding **grants the permissions defined in a role to a user or set of users**. It holds a list of subjects (users, groups, or service accounts), and a reference to the role being granted.

A **RoleBinding grants permissions within a specific namespace** whereas a **ClusterRoleBinding grants that access cluster-wide**.

Lab: Access Control

1) Create a new „developers” namespace
2) Generate private key and CSR for user „training” and sing it by the CA
3) Create Role and RoleBinding
4) Create kubeconfig for user „training” and test it.

Kubernetes Autoscaling

Autoscaling

The **Horizontal Pod Autoscaler (HPA)** dynamically increases and decreases the number of Pods in a Deployment based on demand.
Ref.: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/

The **Cluster Autoscaler (CA)** dynamically increases and decreases the number of nodes in your cluster based on demand.

Lab: Autoscaling

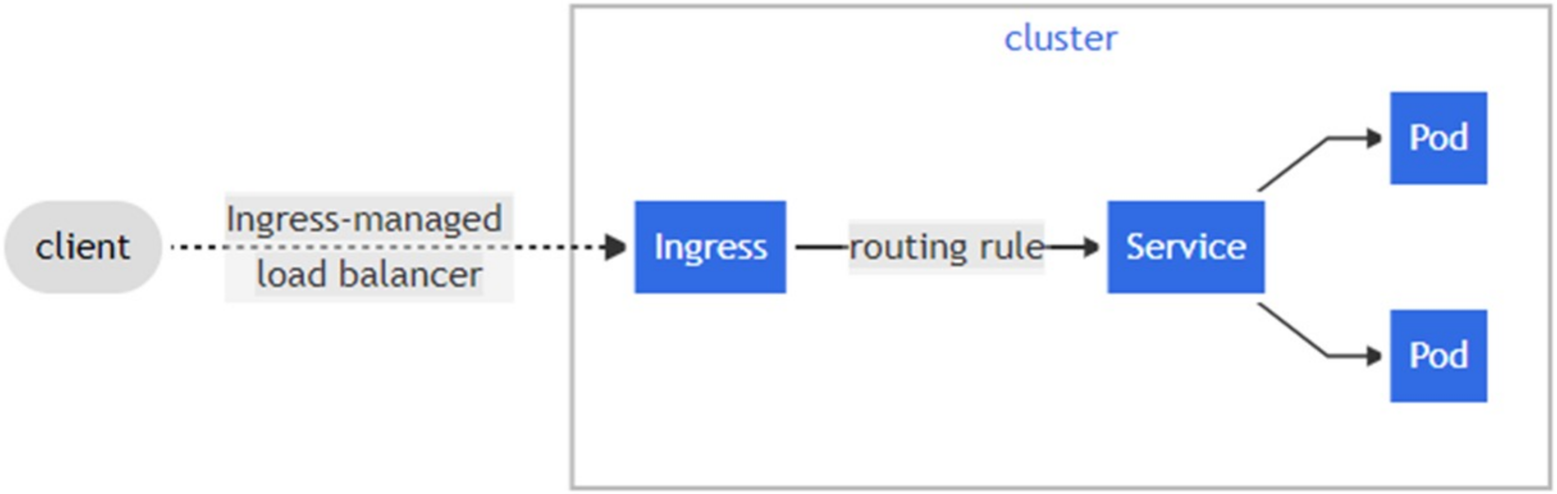
1) Use Horizontal Pod Autoscaler (HPA) for your app

2) Test it

Ingress

Ingress

An **API object that manages external access to the services in a cluster**. Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.



Lab: Ingress

1) Install NGINX Ingress controller

2) Create Ingress objects (rules) for your apps (services)

kubectl plugins

kubectl plugins

A **plugin is a standalone executable file**, whose name begins with **`kubectl-`**. To install a plugin, move its executable file to anywhere on your PATH.

You can also discover and install **`kubectl`** plugins available in the open source using **Krew**.

Ref.: https://kubernetes.io/docs/tasks/extend-kubectl/kubectl-plugins/

Krew: kubectl plugin manager

Krew is the plugin manager for **`kubectl`** command-line tool.

Krew helps you:

- discover kubectl plugins,
- install them on your machine,
- and keep the installed plugins up-to-date.

Ref.: https://krew.sigs.k8s.io/plugins/

Lab: kubectl plugins

1) Install and use one or more **`kubectl`** plugin