Intro to Kubernetes

In order to set the stage to dive deep into Kubernetes terms it helps if we understand some terms that aren't Kubernetes specific, but lay the groundwork for deploying a Kubernetes cluster.

cgroup (control group)

Cgroups are a Linux kernel feature. Combined with another Linux kernel feature, namespaces, you can isolate processes from each other. That is basically how containers are created. Cgroups specifically limit, account for, and isolates the resource usage of a collection of processes.

Cloud Native Computing Foundation (CNCF)

The Cloud Native Computing Foundation is part of the nonprofit Linux Foundation. According to their site they support the growth and health of cloud native open source software. A few of the projects under CNCF include Kubernetes, Prometheus, and Envoy. Google is one of the top contributors to CNCF.

Container

Containers are often compared to virtual machines (VMs) as they both offer a way to package an application together with libraries and other dependencies. However, instead of virtualizing the hardware stack like VMs, containers do it at the operating system level. Containers wrap the code of an application as well as the other dependencies it needs when running. An app container is specifically used for running an application as opposed to an init container that runs separately with initialization instructions for an entire workload.

Container Runtime

The container runtime is software that runs the containers. Kubernetes supports several container runtimes: Docker, containerd, CRI-O, and any implementation of the Kubernetes CRI (Container Runtime Interface). The most popular container runtime today is Docker.

Containerd

An open-source container runtime that is part of CNCF. It is a daemon for Linux and Windows that manages the container lifecycle of its host system. It takes care of image transfer and storage, container execution and supervision, providing network access, and more. You can read more about it here https://containerd.io/.

Docker

Is the most popular container runtime, a software that runs containerization of applications. Using the resource isolation features of the Linux kernel, namespaces and cgroups, Docker allows independent containers to run within a single Linux instance. When Kubernetes schedules a pod to a node, Docker is instructed by the kubelet running on that node to launch the containers.

Microservices

This is a strategy for developing software where an application is organized as a group of loosely connected services. It is focused on building simpler modules that make up the entirety of a software application. Implementing microservices allows changes that need to be made to be done in smaller chunks where some of the modules don’t even need to be touched. You can also scale specific pieces of an application instead of the entire thing.

Kubernetes Basics

Cluster

A cluster is a set of nodes, or worker machines, that containerized applications run on. These nodes can be physical machines or virtual machines. If you are running Kubernetes you are running a cluster. A cluster will have at least one node.

Controller

Kubernetes has a group of control loops that monitor the state of your cluster and try to make adjustments to keep the cluster as close as possible to the desired state. These are called controllers in Kubernetes. A few of these controllers are located in the control plane and are core to operating Kubernetes. Some examples of these controllers are the deployment controller, daemonset controller, the namespace controller, and the persistent volume controller.

Manifest

A manifest in Kubernetes is typically a JSON or YAML file that is used to specify a desired state of a Kubernetes API object. Once a manifest is applied Kubernetes will maintain the specifications set of the Kubernetes object configured. They are used to create, modify, and delete objects like pods, deployments, services or ingresses.

Node

Nodes are the machines, either physical or virtual, in your cluster that your applications run on. Nodes perform work dictated by the Kubernetes control plane. Each node runs kubelet to communicate with the master nodes and regulate itself, kube-proxy (or equivalent) to handle networking, and the container runtime.

Pod

The most basic Kubernetes object. A pod consists of a group of containers that are running in your cluster. Most commonly, a Pod runs a single primary container. However, many teams run optional sidecar containers in order to add more functionality like monitoring.

Volume

A Kubernetes volume is a directory that contains data for containers in a Pod to access.

Workload

An application running on Kubernetes. They are divided into different types, the most popular ones being: Deployments, StatefulSets, DaemonSets, Jobs, and CronJobs.

Kubernetes Architecture & Core Objects

Kubernetes API

Core to Kubernetes is a RESTful API that is used to handle every operation involving changing objects in your cluster. It is responsible for storing the state of your cluster and provides a declarative way for users to configure their cluster objects. Everything in Kubernetes is treated as an API object. The Kubernetes API can be interacted with directly, but the more popular way of interacting with it is using the command line interface to Kubernetes called Kubectl.

Aggregation Layer

If you are needing more functionality than what is offered in the core Kubernetes APIs the aggregation layer lets you build additional APIs within your cluster. You can create your own, or find some offered by third parties.

Kubectl

This is the command line configuration tool for Kubernetes that communicates with a Kubernetes API server. Using kubectl allows you to create, inspect, update and delete Kubernetes objects. If you want to learn more about Kubectl commands take a look at our Kubectl cheatsheet.

Control Plane

Kubernetes clusters are divided into two parts: the control plane run by a master node or nodes, and the worker machines/nodes. The Control plane is the container orchestration layer where the Master Node resides managing the Worker Nodes. The processes running on the master node make up the control

plane. The Master Node includes Kubernetes objects that control the cluster, as well as the data about the cluster's state and configuration. These Kubernetes objects include the kube-apiserver, etcd, kube-scheduler, kube-controller-manager, and the cloud-controller manager.

API Server

The API Server, also known as kube-apiserver, is part of the Kubernetes Control Plane. When you make commands through kubectl to your Kubernetes cluster you are communicating with the API server component. It is the only component that connects to etcd and acts as the front end for the Kubernetes control plane.

Etcd

Data about the configuration and state of the cluster lives in a key value store database called etcd. It is the ultimate source of truth about your current cluster state.

Kube-scheduler

Part of the Kubernetes components that manage your cluster in the Control Plane. The Kube-scheduler checks for unscheduled pods and assigns them to worker nodes based on resource availability and other specifications. When a pod is assigned to a node the kubelet on that node is triggered and the pod and containers are created.

Kube-controller-manager

Part of the Kubernetes master components that manage your cluster in the Control Plane. The Kube-controller-manager is a daemon that manages the core control loops, or controllers, in Kubernetes. There are many controllers in Kubernetes a few of them are: Node controller, Replication controller, Endpoints controller, and the Service Account & Token controller.

Cloud-controller-manager

A control plane component that embeds cloud-specific control logic. This helps you link your cloud provider's API into your cluster, and separates what the cloud provider can interact with in your cluster versus those that shouldn't.

Data Plane

The layer of software that processes the data requests. In Kubernetes it is the layer controlled by the Control Plane and provides resources that allow Containers to run. It consists of Worker Nodes.

Kubelet

Kubelet is a tiny application that runs on each node in the cluster. It takes a set of specifications, PodSpecs, and ensures containers are working. The Kubelet executes actions sent from the master node.

Kube-proxy

Kube-proxy runs on each worker node in your cluster. It manages network communication going to your Pods from inside or outside of your cluster.

Kubernetes Operation

DaemonSet

A DeamonSet ensures that all nodes (or a specific subset of them) are running exactly one copy of a pod. For example, you might want to run fluentd on all your nodes to collect logs, or you might want to run a monitoring process like the Datadog agent. You can learn even more on our blog post An introduction to Kubernetes DaemonSets.

Deployment

A deployment is a resource object in Kubernetes that provides declarative updates to applications. It manages the scheduling and lifecycle of pods. They provide several key features for managing pods, including rolling updates of pods, the ability to rollback, and easily scaling pods horizontally.

Dynamic Volume Provisioning

Utilizing a Kubernetes API object, StorageClass, storage volumes can be automatically provisioned by user request instead of being pre-provisioned.

EndpointSlice

Part of the Kubernetes 1.16 release. Endpoint Slices is a feature that allows you to group network endpoints with Kubernetes resources. A Kubernetes endpoint stores all the pods that match a particular service. When you have a lot of (hundreds) Pods in a service and changes the endpoints objects can get really big. Before Endpoint Slices whenever you made any change in your cluster that impacted your endpoints objects the entire endpoints object would change and shift to every node in your cluster, causing performance issues. With Endpoint Slices you slice up the endpoints object and updates to the endpoints object can be updated across the entire cluster in smaller pieces which do not cause a disruption.

Ephemeral Container

These are a newer type of container that can run temporarily in an existing Pod. These shouldn’t be used to run any part of the workload, but can be very helpful for specific use cases. For example, they can be used as a foundation to attach a debugging container to your main process to investigate a problematic Pod.

Horizontal Pod Autoscaler

Also called HPA, this is an API resource that scales the number of pod replicas. Typically DevOps use CPU and memory thresholds to increase or decrease the number of pod replicas. Horizontal Pod Autoscaler can also use custom metric thresholds as well.

HostAliases

HostAliases is an optional mapping between the IP address and hostname to be injected into a Pod's host file if specified. They are only valid for non-hostNetwork Pods.

Ingress

An API object that exposes HTTP and HTTPS routes from outside the cluster to services within the cluster giving external access to Services running in the cluster. There are five ingress types of a single service ingress, simple fanout, name based virtual hosting, TLS termination, and load balancing. Deploying an ingress controller that is responsible for the ingress is the only way to create one.

Init Container

Sometimes in your cluster it will be important to prepare a pod before app containers deploy to it and start running your application. An initialization container, or init container for short, are perfect for this. They are similar to app containers except that they must complete successfully before app containers are allowed to run. Some use cases for using init containers are waiting a determined amount of time before starting an app container, cloning a git repository into a volume, or registering a Pod with a remote server.

Job

A Job in Kubernetes is an object that supervises the completion of a process by tracking one or more Pod objects carrying out the process and making sure the correct number of them successfully terminate.

Kubernetes Events

A resource type in Kubernetes that are automatically created when other resources have state changes, errors, or other messages that should broadcast to the system. There isn't a lot of documentation available for these events, but they are very helpful when debugging issues with your cluster. You can read more about them on our Types of Kubernetes Events blog post.

Container Image

A container image is an instance of a container that has been stored. The container is packaged with a set of specified software making it ready to execute a desired application. The image can be stored in a container registry, pulled to a local system, or run as an application. Think of it as a container template with all the necessary pieces to spin up container instances. You can also edit the metadata of these images to indicate what it is used for, when it was created, who built it, and more.

LimitRange

Pods run without constraints on the amount of CPU and memory they'll consume by default. Meaning any pod in the cluster can use as much CPU and memory on the node that executes that pod. LimitRange allows a user to place resource consumption constraints on containers or pods within a namespace.

Logging

Logs are the list of events that are recorded by a cluster or application. They help us understand how data is flowing through applications as well as spot when and where errors occur. In Kubernetes your application should output logs to stdout and stderr. The kubelet installed on each node will collect them and combine them into a log file that is managed by Kubernetes. You can read more about basic log management in Kubernetes here.

Mirror Pod

When a static pod is created the kubelet that the static pod is bound to will automatically try and create a pod object on the Kubernetes API server. This is only for visibility, the static pod cannot be controlled from the Kubernetes API server (kube-apiserver).

Network Policy

By default, pods in Kubernetes can communicate with each other and receive traffic from any source. Network Policy is a Kubernetes resource that specifies

how Pod groups are allowed to communicate with each other and with other network endpoints. Using labels, Pods are selected and traffic is configured to determine what is allowed.

Persistent Volume

In addition to managing containers that are running an application, Kubernetes can manage the application data used by a cluster. Persistent Volume, or PV for short, is a storage resource in the cluster that can outlive the life of a pod. Persistent volumes abstract the storage of data from the underlying storage infrastructure (physical disks, cloud storage, etc.). Since they outlive pods, PVs can be used to create stateful pods.

Persistent Volume Claim

Or PVC, for short, is a request for storage that can be claimed as defined in a PersistentVolume. The PVC becomes mounted as a volume in a container and specifies the amount of storage resources it needs, how it will be accessed, and how the storage resource will be reclaimed.

Pod Disruption Budget

This is also called PDB for short. It allows you to configure the amount of concurrent disruptions that can be tolerated for a class of pods ensuring you always have a determined number of pods available when disruption events occur. Whenever a disruption to a class of pods in a cluster causes the cluster to drop below the budget, the operation is paused until the budget can be maintained again.

Pod Phase

The Pod phase is a high level summary of where a Pod is at in its lifecycle. It is shown in the phase field included in the status field of a PodStatus object. There are five possible phases for a Pod:

1. Pending - Kubernetes has accepted the Pod, but is waiting for the required number of Container images to be created.

2. Running - All of the Containers have been created and the Pod has been attached to a Node. At least one of the Containers in the Pod is either running, or working on starting or restarting.
3. Succeeded - Every Container in the Pod has successfully terminated and are not going to restart.
4. Failed - Every Container in the Pod has terminated, but at least one of them terminated in failure. It either exited with a non-zero status or was terminated by the system.
5. Unknown - The state of the Pod can't be determined, most likely due to a failure in communicating with the Pod host.

Pod Lifecycle

There are five possible Pod phases: Pending, Running, Succeeded, Failed, and Unknown. The Pod Lifecycle is the sequence of these states that the Pod enters while it exists.

Pod Priority

Some Pods are more important than others in production workloads. This feature allows you to determine the scheduling priority of a Pod giving it preference over less important Pods.

PodPreset

PodPreset is an API object that lets you inject information into Pods when they are created. The presets are managed by Admission Control which applies the Pod presets when creation requests are made. Some examples of information injected into pods with this object are secrets, volumes, volume mounts, and environment variables.

Preemption

After a pod is created it is placed in a queue. With the Pod Priority and Preemption features turned on the Kubernetes Scheduler grabs a pod from the queue and attempts to schedule the pod on a node in the cluster. If a pod cannot be scheduled the Kubernetes Scheduler uses Preemption logic to terminate lower priority Pods to allow the scheduling of the pending Pod.

Proxy

In computing, a proxy is a server that acts as an intermediary for a remote service. A client interacts with the proxy, the proxy copies the client's data to the actual server, the actual server replies to the proxy, the proxy sends the actual server's reply to the client. In Kubernetes, kube-proxy is a network proxy for performing Kubernetes networking services. It handles network communications in or out of your cluster. Each node contains kube-proxy or an equivalent implementation.

QoS Class

QoS Class or Quality of Service Class is a concept in Kubernetes that helps determine the priority of pods when it comes to scheduling and eviction. Kubernetes assigns these classes by itself when a Pod is created. It is determined by the Pods resource requests and limits. There are three QoS classes Kubernetes can assign: guaranteed, burstable, and besteffort.

ReplicaSet

One of the core Kubernetes controllers that is used to make sure the specified number of pod replicas are running.

ReplicationController

The control plane manages the number of Pods running in a cluster. This number is defined and the control plane keeps the cluster in adherence to it even when Pods fail, are deleted, or if somehow too many were started. The ReplicationController controls how many exact copies of a pod should be running in the cluster.

Resource Quotas

Defined by a ResourceQuota object, a Resource Quota declares restrictions that limit the total resource consumption per Namespace. The total quantity of objects allowed in a namespace can be set, in addition to limiting the total

amount of memory or CPU consumption in a Namespace. This helps teams only use their fair share of the resources in a cluster.

Service

A Kubernetes Service groups a set of pods together and offers a way to make them accessible as a network service through the DNS name of the Service. One way you can configure your Service is by using a YAML manifest. There are four types of Services in Kubernetes: ClusterIP, NodePort, LoadBalancer, and ExternalName. You can get more in depth information on our post about Kubernetes Services.

ServiceAccount

Provides an identity for processes that run in a Pod. When a process inside a Pod wants to interact with the API server it will authenticate through a particular service account. By default a Pod is automatically assigned the same service account in the same Namespace. Service accounts are granted API permissions via RBAC.

Sidecar Container

Extending support of a main container with a utility container in a Pod. Sidecar containers are paired with one or more main containers and they enhance the functionality of those main containers. An example would be using a sidecar container specifically to process system logs, or for monitoring.

StatefulSet

This is a controller in Kubernetes that manages the deployment and scaling of a set of Pods. In a StatefulSet Pods that are managed by the controller are given a unique identity. It uses this same identity when it needs to reschedule Pods. We've written more in depth about Kubernetes StatefulSet here.

Static Pod

Static Pods are similar to regular Kubernetes Pods, except that they are managed directly by the kubelet on a node and not the API server. This means they are ignored by the kube-scheduler and won't be evicted when the node is drained or fails.

Storage Class

A StorageClass gives admins a way to describe the different available storage options. The key fields in a StorageClass specification are the provisioner, parameters, reclaimPolicy, and the volumeBindingMode. These can be used when a Persistent Volume belonging to the specified class needs to be dynamically provisioned. The name of a StorageClass object and be used to request the preferred storage option.

Sysctl

Linux provides an administrator a way to modify attributes of the kernel at runtime through the sysctl interface. It is both the system call used as well as the name of the tool used to modify kernel settings.

Taint

This is used when you don't want Pods to run on a specific node or node group. You might want to do this if nodes are reserved with specialized hardware, they aren't licensed for software running in the cluster, or other reasons. Taints tell the Kubernetes Scheduler that specific nodes are not available. Taints work hand in hand with Tolerations providing a way to take more control over which pods are scheduled to which nodes, if you find yourself needing that functionality.

Toleration

Tolerations provide a way for a pod to ignore a taint that has been applied to nodes. Tolerations are a property of the PodSpec. It is in the format of a key name, operator, a value, and an effect. To set a toleration to a specific taint the

key name needs to be set to the same value as the taint's key. You can also create a Toleration that will ignore any taint by setting operator equal to exists.

Vertical Pod Autoscaler

Also called VPA, is an infrastructure service that scales your cluster by allocating more or less cpu or memory to existing pods as opposed to the Horizontal Pod Autoscaler that increases or decreases the amount of pods to scale. It can dynamically adjust based on analysis of historical resource utilization, amount of resources available in the cluster and real-time events like OOMs. Here is the Github repository for it.

Kubernetes Security and Organization

Admission Controller

A native Kubernetes security feature that allows you to configure what objects are allowed to run on your cluster. The admission controllers are plugins that regulate how the cluster is used. They check requests to the Kubernetes API server before allowing objects to run.

Annotation

In Kubernetes you can use either labels or annotations to give Kubernetes objects metadata. An annotation is used to apply arbitrary non-identifying metadata to objects using a key-value pair.

Label

A Label in Kubernetes is a key/value pair attached to Kubernetes objects that specify meaningful information about those objects to users. Using Labels will help you organize objects and select subsets of objects.

Name

Every Kubernetes object in your cluster has a unique Name that is a string provided by the user at creation time. These Names are used in resource URLs

and can only be used by one object of a given kind at a time. Most pod names are generated by their controller.

Namespace

These act as virtual clusters. A further abstraction that provides a way to divide a physical cluster into multiple virtual clusters. It is also a way to provide organization to objects in a cluster.

Pod Security Policy

A Pod Security Policy is a cluster-level resource implemented as an optional admission controller in the cluster. These policies define the conditions a Pod must meet to be allowed into the system.

RBAC

Role-Based Access Control, or RBAC for short, provides a way to manage different levels of access in your Kubernetes cluster through the Kubernetes API. With multiple users or services accessing the cluster they will have different permission needs and should be limited to only the permissions required to perform their function. RBAC, as the name implies, leverages roles which grant the necessary level of access to sets of users in your cluster.

Secret

Often a Kubernetes Pod will need to leverage sensitive information like passwords, API keys, ssh keys, OAuth tokens, and more. A Secret is a Kubernetes object that stores this confidential information so that Pods can use it without the data being shown. Secrets are exposed to containers either as a file in a volume mount or through environment variables.

Security Context

Setting the securityContext field defines the access control settings in Kubernetes. This field configures the user that processes run as, the group that

processes run as, and privilege settings.Two levels of Security Context exist in Kubernetes: pod level security context, and container level security context. Setting the security context on the pod level applies to all containers in the Pod. Whereas container level security context settings apply to the specific container.

Selector

A Label in Kubernetes is a key/value pair that is attached to Kubernetes objects that specify meaningful information about those objects to users. Selectors are used when querying or filtering a list of Kubernetes objects based on the labels that have been created.

UID

Short for unique ID, the UID in Kubernetes is a string generated by the system that uniquely identifies every object created in the entire lifetime of the Kubernetes cluster.

Kubernetes Tools & Extensions

Add-ons

These are tools that give Kubernetes additional features. Examples of some popular ones include: Calico, Flannel, Dashboard, and Weave Scope.

Amazon EKS

Amazon EKS stands for Amazon Elastic Kubernetes Service. It launched in 2018 and is AWS's managed service of the Kubernetes control plane in your cluster. We have a lot more content about running Kubernetes on AWS.

API Group

An API Group in Kubernetes is a set of related APIs. Grouping them in sets of related paths makes it easier to extend the Kubernetes API. The group is specified in a REST path and in the apiVersion field of a serialized object.

Azure Kubernetes Service (AKS)

The managed Kubernetes service offered by Microsoft in Azure. You can learn more about AKS here.

Container Network Interface (CNI)

Container network interface (CNI) is a container networking specification proposed by CoreOS and adopted by projects like Apache Mesos, Kubernetes, Kurma, and rkt. It is the simplest possible interface between container runtime and network implementation. CNI plugins are a type of network plugin that adheres to the appc/CNI specification.

Container Runtime Interface (CRI)

The container runtime interface (CRI) is an API which enables kubelet to use a variety of container runtimes without the need to recompile.

Container Storage Interface (CSI)

The Container Storage Interface or CSI is an initiative defining a standard interface to expose storage systems to container orchestrator systems (COs). This allows COs to work with all storage vendors like Ceph, NetApp, Portworx, EBS, and more. It lets storage vendors create custom storage plugins for Kubernetes, and other COs, without specifically adding them to the Kubernetes repository.

CRI-O

According to their site, CRI-O is a “lightweight container runtime for Kubernetes.” It is a tool that allows you to use any Open Container Initiative (OCI) compatible runtimes. It is a lightweight alternative to Docker. You can read more about it here.

Custom Controller

If you need to extend the core Kubernetes controllers you can write your own to build upon the core functionality without changing the Kubernetes core code using the Kubernetes API.

Custom Resource

First off, a resource in Kubernetes is an endpoint in the core Kubernetes API where a group of specific API objects are found. A custom resource is essentially just that, custom code used to add to the Kubernetes API for your environment. Custom resources are installed by creating a CustomResourceDefinition.

Device Plugin

Kubernetes device plugins provide Pods on worker Nodes access to resources that have vendor-specific requirements. The device plugin can be installed as a DaemonSet, or directly installed on the specific Nodes in need. Here is an example of a device plugin installed as a DaemonSet for NVIDIA devices. It allows you to expose the number of GPUs in your cluster, keep track of the GPU health, and run RPU enabled containers in your cluster.

Extensions

Extensions are software components that extend and deeply integrate with Kubernetes to support new types of hardware. There are different types of extensions available like API extensions, and infrastructure extensions.

FlexVolume

FlexVolume allows users to mount volumes provided by vendors into their clusters. These vendor volumes are considered out-of-tree, meaning they are not part of the Kubernetes core code. You can learn more about it on its Github repository.

Google Kubernetes Engine (GKE)

GKE is a managed Kubernetes service offered in Google Cloud. You can learn more about it here.

Helm

Is an open source free package manager for Kubernetes. Think of it like Homebrew for Kubernetes. It helps you define, install, and update your Kubernetes application. Helm is maintained by the Helm community and is part of the CNCF incubator. You can read more about using Helm for managing and configuring Kubernetes here.

Helm Chart

A helm chart is the format that Helm uses to package a group of pre-configured Kubernetes resources. A chart could encompass something as simple as a single pre-configured Kubernetes Pod all the way to a full web app stack.

Istio

An open source service mesh and platform, including APIs, that helps simplify running a distributed microservice architecture. It is not specific to Kubernetes. Istio provides core features as a layer of infrastructure between a service and the network. Some of them are: traffic management, security, and observability into your service mesh deployment. More information about Istio can be found at this site.

Kops

Short for Kubernetes operations, kops is a CLI tool that was specifically created by the Kubernetes on AWS community. It allows you to install, upgrade, and manage your Kubernetes cluster. It significantly simplifies cluster creation and operations compared to manually getting started. You can learn more about it at the Kops Github repository.

Kubeadm

A toolkit for quickly setting up a minimal viable Kubernetes cluster no matter where you are running them. However, unlike a tool like Kops, it cannot create your infrastructure so is only really useful on existing infrastructure.

Managed Service

A third party maintained resource used by an application. Some examples of Managed Services are AWS EC2, AWS EKS, Azure SQL Database, and GCP Pub/Sub. Many of the managed services useful for a Kubernetes cluster can be provisioned using Service Catalog directly from the native Kubernetes tooling.

Minikube

Minikube is best utilized as a tool to test out and learn about Kubernetes. It is meant for running Kubernetes locally inside a VM on your computer. It runs a single-node cluster on this VM. Here are the docs for Minikube if you'd like to learn more about it or get started using it.

Service Catalog

This is an extension API that allows applications inside your Kubernetes cluster to leverage services available outside of the cluster. For example, if you are running your cluster on VMs with AWS using EC2 and would like to integrate RDS, AWS's relational database service, Service Catalog gives your cluster applications access to RDS without complicated amounts of manual workarounds that you would otherwise be faced with.

Volume Plugin

A Kubernetes Volume Plugin extends the Kubernetes volume interface enabling the integration of a block or file storage system. It allows you to attach and mount these storage volumes for Pod use. The plugins can either be part of the Kubernetes code repository, or developed independently. These volume plugins are built on Container Storage Interface (CSI), which is the primary volume plugin system for Kubernetes.