Kubernetes Architecture

Wednesday, March 18, 2020 3:02 PM

What is Kubernetes?
Portable, extensible, open-source platform for managing containerized workloads and services.

Deployment Schemes
Traditional

- Applications ran on physical servers
- Couldn't define resource boundaries
- Large investment in hardware to run multiple applications cleanly

Virtualized

- Multiple VMs on one physical host
- Allows application isolation between VMs
- Increased security between applications/VMs
- Full machine with all components and OS running on virtual hardware

Containerized

- Similar to VMs but each container shares the same OS
- Run natively on the host machine
- Has own FS, CPU, memory, process space, and more
- Decoupled from the infrastructure, portable, lightweight

Why Kubernetes
Containers are good to run applications, but you will still need to manage and monitor containers to ensure uptime. Kubernetes provides the framework for running resilient, distributed systems.

- Service discovery and load balancing
  - dns/ip routing
  - traffic load balancing
- Storage orchestration
  - automatically mount storage providers of choice (local, cloud, etc)
- Automated rollouts and rollbacks
  - declarative deployments allow kubernetes to match deployments to a desired state and resource usage
- Automatic bin packing
  - Kubernetes will deploy containers across a node cluster to best utilize the available resources
- Self-healing
  - restarts failing containers, replaces containers, kill non-responsive containers to user-defined health check, doesn't advertise until service is ready and stable
- Secret and config management
  - Kubernetes will manage sensitive information like passwords, OAuth tokens, and SSH keys. Management configuration without rebuilding container images, and w/o exposing secrets in your stack configuration.

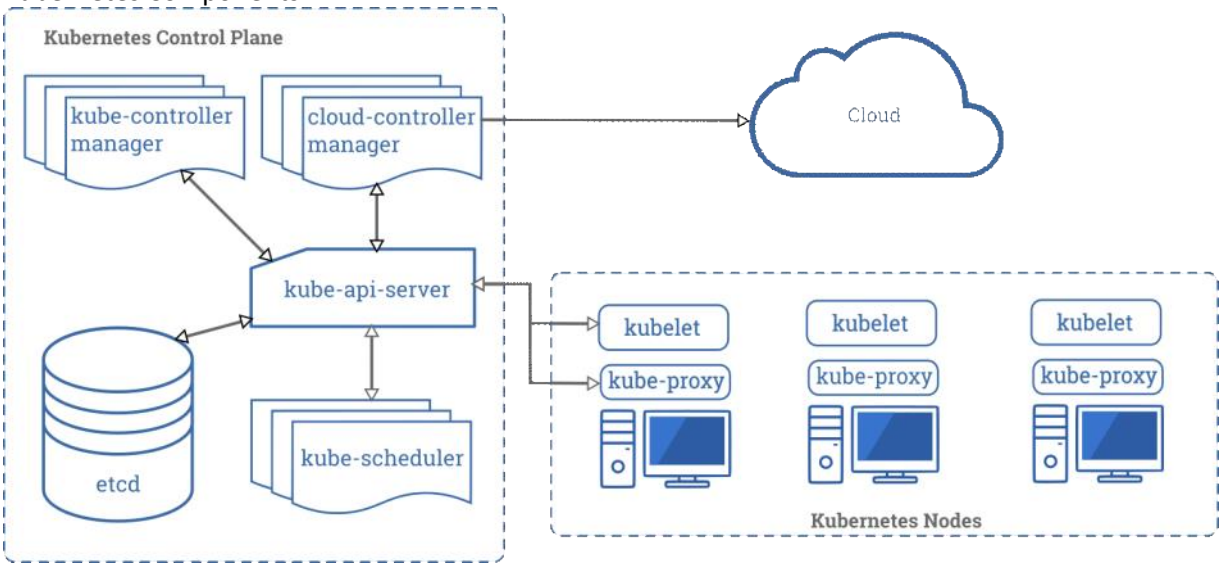
What Kubernetes is not?
K8S is not PaaS. K8S has some features similar to PaaS (deployment, scaling, load balancing, logging, monitoring), but it operates at the container level with the default solutions being pluggable.

- Doesn't limit the type of application. If it runs in a container, it should run on K8S
- Doesn't build your application. Doesn't deploy source code
- Doesn't provide application-level services like middleware (messaging buses), data-

processing (Spark), databases (PostgreSQL), caches.

- Doesn't dictate logging, monitoring, or alerting solutions. It provides some integrations as PoC and mechanisms to collect and export metrics
- Doesn't provide or mandate a config language/system. It provides a declarative API that may be targeted by arbitrary forms of declarative specifications.
- Does not provide nor adopt any comprehensive machine configuration, maintenance, management, or self-healing systems
- Is not just orchestration. K8S doesn't impose centralized control, but rather is a set of composable control processes that continuously drive the current state towards the provided desired state.

Kubernetes Components



Control Plane Components

The Control Plane makes the global decisions for the cluster (scheduling), as well as detecting and responding to cluster events (starting new pods when needed). Control plane components can run on any machine in the cluster, but for simplicity sake they the components are typically run on a single master and no user containers are run on this machine.

- kube-apiserver
  - exposes the k8s api, the frontend for the control plane
- etcd
  - key/value store for all cluster data
- kube-scheduler
  - watches for newly createad Pods with no assigned node and chooses the optimal node to run it on.
  - considers individual and collective resource requirements like hardware, software, policy constraints, affinity and anti-affinity specs, data locality, inter-workload interference, and deadlines
- kube-controller-manager
  - Collection of controllers that run controller processes
    - Node Controller
      - Notices and responds when nodes go down
    - Replication Controller
      - Responsible for maintaining the correct number of pods for every replication controller object in the system
    - Endpoints Controller

        - Populates the Endpoints object (joins Services & Pods)
    - Service Account & Token Controllers
        - create default accounts and API access tokens for new namespaces
- cloud-controller-manager
    - runs controllers that interact with the underlying cloud providers.
    - Provides abstraction between cloud provider evolution and k8s.
        - Node Controller
            - checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
        - Route Controller
            - Setting up routes in the underlying cloud infrastructure
        - Service Controller
            - creating, updating, and deleting cloud provider load balancers
        - Volume Controller
            - creating, attaching, and mounting volumes, and interacting with cloud provider to orchestrate volumes

Node Components

Node components run on every node, maintaining pods and providing the K8S environment

- kubelet
    - agent that ensures all containers are running in a Pod
    - Handles only containers that appear in a PodSpec
    - doesn't manage containers not created by k8s
- kube-proxy
    - implements part of the k8s Service concept
        - exposes an application running on a set of Pods as a network service
        - Allow network communication between Pods from session inside or outside of your cluster
- Container Runtime
    - software that is responsible for running containers
    - Supports Docker, containerd, CRI-O and any impl of Kubernetes CRI (Container Runtime Interface)

Objects

Wednesday, March 18, 2020 3:03 PM

Kubernetes contains a number of abstractions that represent the state of your system. The abstractions represent deployed containerized applications and workloads, their associated network and disk resources, and other information about the cluster and what it's doing. These abstractions are represented by objects in the Kubernetes API

Basic Objects

- Pod
- Service
- Volume
- Namespace

High level Objects

- Deployment
- DaemonSet
- StatefulSet
- ReplicaSet
- Job

The high level objects rely on controllers to build upon the basic objects.

Pods

Wednesday, March 18, 2020 3:32 PM

What is a Pod
Pods are the basic execution unit of Kubernetes. A pod is the simplest unit in K8S object model that you create or deploy.

Pods are encapsulations of an application's container, storage resources, unique IP, and other container options. A pod is a single instance of an application in K8S which may contain one or more running containers

- Single container pods
  - Most common pattern
  - One container per pod
- Multi-Container Pod
  - Sidecar pattern
  - Tightly coupled containers
  - Typically one container supplies the interface while the others provide support

Working with Pods
Each Pod is meant to run a single instance of an application. Don't replicate containers in a single pod. Horizontal scaling is accomplished by replicating containers in separate pods, one for each replica. Replicated pods are usually created and managed as a group by a Controller

When Pods are created the kube-scheduler is responsible for finding a node in the cluster that has the resources needed. By their nature Pods are ephemeral, disposable entities. Pods will run on the it's assigned node until terminated, deleted, or evicted. Pods don't self-heal and any event that causes the Pod to fail or it's node to fail will result in the Pod being effected permanently.

Pods and Controllers
Controllers can easily create and manage multiple pods, handling replication and rollout, provide self-healing.

Examples of Controllers with one or more Pods
- Deployment
- DaemonSet
- StatefulSet

Pod Templates
In general Controllers use pod templates to create Pod replicas. A template provides a description of the desired state. There is no entanglement between the template and the pods, any changes to the template will have no direct effect on the pods already created.

```
apiVersion: v1
kind: Pod
metadata:
    name: myapp-pod
```

```
    labels:
        app: myapp
spec:
    containers:
      ○ name: myapp-container
        image: busybox
        command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']
```

Deployment

Wednesday, March 18, 2020 4:08 PM

What is a Deployment
A deployment provided declarative updates to Pods and ReplicaSets. Simply put a Deployment is a description of Pod templates that are being controlled by a ReplicaSet. Deployments are the easiest way to create and orchestrate replicas of application containers is Kubernetes.

Creating a Deployment

controllers/nginx-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

This example

- Creates a deployment named 'nginx-deployment
- Creates 3 replicated Pods
- The selector field defines how the deployment finds it's Pods
- The template field describes the Pods

- Pods are labeled 'app: nginx'
- Each Pod runs a container named 'nginx' using the image nginx:1.14.2

Complete Deployment
Kubernetes considers a deployment to be complete when

- All replicas have been updated
- All replicas are available
- No old replicas are running

Failed Deployment
Deployments can fail and never complete when"

- Insufficient quota
- Readiness probe failures
- Image pull errors
- Insufficient permissions

Service

Wednesday, March 18, 2020 3:33 PM

What is a Service
A service is an abstraction which defines a logical set of Pods and a policy for accessing them. Services provide the abstraction so clients don't have to memorize ephemeral IP addresses.

Service Discovery
The API server Endpoints controller can be queried for service information. In non-native environments, Kubernetes offers ways to place a network port or load balancer between your application and backend Pods.

Defining a Service

```
apiVersion: v1
kind: Service
metadata:
    name: my-service
spec:
    selector:
        app: MyApp
    ports:
      - protocol: TCP
        port: 80
        targetPort: 9376
```

This create a service named my-service (must be valid DNS name). When created Kubernetes will assign it an IP address. The controller for the service continously scans for Pods that match it selector and POSTs any updates to an Endpoint object also named "my-service". Traffic incoming on port 80 will be mapped to pod port 9376

Service Types
There are several types of services used for exposing your application externally.

- ClusterIP: Exposes the Service on a cluster-internal IP. This makes the service reachable from inside the cluster. (Default)
- NodePort: Exposes the Service on each Node's IP at a static port. A ClusterIP Service to which the NodePort Service routes is automatically created. You can contact the NodePort Service from outside the cluster by contacting <NodeIP>:<NodePort>
- LoadBalancer: Exposes the Service externally using a cloud provider's load balancer. NodePort and ClusterIP Services, to which the external load balancer routes, are automatically created.
- ExternalName: Maps the Service to the contents of the externalName field (foo.bar.example.com) by returning a CNAME record with its value. No proxying of any kind is setup.

Ingress can also be used to expose a Service. Ingress is not a service type, but it acts as an entry point to the cluster. It can consolidate routing rules into one resource, it can expose multiple services under the

same IP address.

Type NodePort

```
apiVersion: v1
kind: Service
metadata:
    name: client-node-port
spec:
    type: NodePort
    ports:
        - name: serviceport
          port: 3050
          targetPort: 3000
          nodePort: 31515
          protocol: TCP
    selector:
        component: web
```

Defining a NodePort requires us to set up at least on port field. 'nodePort' is how the service will be accessed from outside the cluster. If not specified 'nodePort' will be assigned a random port in the default range of 30000-32767. 'port' is how the service is accessed internally form the clusterIP proxy. 'targetPort' is the mapped Pod port

Type ClusterIP

ClusterIP services are used for internal access only. Typically used for one of two reasons.

1. An Ingress and LoadBalancer will be routing external traffic to the service
2. Access will only be allowed to other internal K8S objects

```
apiVersion: v1
kind: Service
metadata:
    name: api-cluster-service
spec:
    type: ClusterIP
    selector:
        component: api
    ports:
        - port: 5000
          targetPort: 5000
```

Namespace

Wednesday, March 18, 2020 3:34 PM

What is a namespace
A virtual k8s cluster inside a physical cluster. K8S supports the creation and management of multiple virtual clusters in a single physical cluster.

Using multiple namespaces
Namespaces are intended for environments with multiple teams or projects. For clusters with only a few users, namespaces aren't truly necessary.

Working with namespaces
Kubernetes starts with 3 namespaces

- Default – the default namespace for object with no other namespace
- Kube-system – the namespace for object create by the Kubernetes system
- Kube-public – objects in this namespace are readable by all users (even not authenticated). This namespace is mostly reserved for cluster usage.

Namespaces and DNS
When a service is created, it will create a DNS entry in the form of <service-name>.<namespace-name>.svc.cluster.local. If a container uses <service-name> the it resolves to the service which is local to that container's Pod's namespace. If you need to access service across namespaces, you need to use the FQDN.

Creating a Namespace

```
apiVersion: v1

kind: Namespace

metadata:

    name: <insert-namespace-name-here>
```

Volume

Wednesday, March 18, 2020 3:33 PM

Containers by their nature are ephemeral. When a container crashes kubelet will restart it, but the files will be lost. If a pod is running multiple containers it is common for the containers to share file. The Volume abstraction solves the problem.

Volumes in K8S are Pod centric. This means that the volume will live as long as the Pod does providing consistent volume access to any containers running in the Pod. However, if the Pod is ever removed for any reason the Volume will be removed as well. K8S supports different types of volumes and a Pod can use any number of them simultaneously.

Volume Types
https://kubernetes.io/docs/concepts/storage/volumes/#types-of-volumes

Referencing a volume in a pod done by adding the volume mounts and volume declarations in the pod spec.

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: deployment-with-volume
spec:
    replicas: 3
    selector:
        matchLabels:
            app: demo
    template:
        metadata:
            labels:
                app: demo
        spec:
            containers:
              - name: demo-container
                image: my-image
                volumeMounts:
                  - mountPath: /path/to/container/dir
                    name: my-volume
            volumes:
              - name: my-volume
                emptyDir: {}
```

The above deployment configuration declares a volume call my-volume as an emptyDir volume and maps the containers mountPath to the volume. As mentioned earlier volumes are controlled at the pod level and will be destroyed when the pod is destroyed. More durable storage can be configured using Kubernete Persistent Volumes

Persistent Volumes

Monday, February 14, 2022 3:31 PM

Persistent volumes are Kubernetes resources used to manage durable storage in a cluster. Unlike volumes, PVs are allocated and managed at the cluster level and so will outlive the lifetime of any pod attached to the volume. As an example, on a cloud provider like AWS or GCP, a persistent volume would be allocated using the providers backing storage services like EBS or GCE Persistent Disk, respectively. Persistent volumes can be provision and managed dynamically in the cluster and don't have to be created and deleted manually.

Persistent volume claims work hand-in-hand with Persistent volumes. Pods use PVC to request a PV. A PVC specifies the size, access mode, and storage class for the PV. If a PV is found, it is bound to PVC. Pods use PVCs as volumes. PVCs and PVs increase portability as the same pod specs can be used in any cluster.

A storage class is a particular implementation of the PV. Any K8S provider will have a variety of storage classes such as gcePersistentDisk for GCP and awsElasticBlockStore for AWS.

Most of the time, PVs can be allocated dynamically, by creating a PVC with a backing PV. The K8S provider will then dynamically create the PV when one cannot be found to bind to.

StatefulSet

Wednesday, March 18, 2020 4:09 PM

What is a StatefulSet
A Stateful set is similar to a Deployment, but is used to help manage a Stateful application. Unlike Deployments though StatefulSet Pods have a sticky identities. These Pods are created from the same spec, but are not interchangeable. Each of the Pods identities is maintained across any rescheduling.

Using StatefulSets
StatefulSets are useful for

- Stable, unique network identifiers
- Stable, persistent storage
- Ordered, graceful deployment and scaling
- Ordered, automated rolling updates

Limitations

- The storage must be pre-provisioned
- Deleting or scaling will not remove the volumes
- StatefulSet require a Headless Service to be responsible for the Pod indentity. You must create the Service
- Pods may not be terminated if the StatefulSet is delete. To ensure graceful termination, scale down to 0 first
- When using OrderedReady Rolling Updates it is possible to reach a broken state that requires manual intervention to repair

Creating a StatefulSet

```
apiVersion: v1

kind: Service

metadata:

  name: nginx

  labels:

   app: nginx

spec:

  ports:

  - port: 80
    name: web
    clusterIP None
    selector:
      app: nginx
---
apiVersion: apps/v1
```

```
kind: StatefulSet
metadata:
 name: web
spec:
 selector:
  matchLabels:
   app: nginx # has to match .spec.template.metadata.labels
 serviceName: "nginx"
replicas: 3 # by default is 1
  template:
 metadata:
 labels:
  app: nginx # has to match .spec.selector.matchLabels
 spec:
  terminationGracePeriodSeconds: 10
  containers:
   - name: nginx
     image: k8s.gcr.io/nginx-slim:0.8
     ports:
     - containerPort: 80
       name: web
    volumeMounts:
     - name: www
       mountPath /usr/share/nginx/html
  volumeClaimTemplates:
 - metadata:
    name:www
   spec:
       accessModes: [ "ReadWriteOnce" ]
       storageClassName: "my-storage-class"
       resources:
         requests:
           storage: 1Gi
```

In the above configuration

- A headless service is used to control the network domain
- A StatefulSet with 3 replicated Pods is created
- VolumeClaimTemplates will provide a stable storage using PersistentVolumes provisioned by a PersistentVolume Provisioner

Deployment and Scaling

- For a StatefulSet with N replicas, when Pods are being deployed, they are created sequentially, in order from {0..N-1}.
- When Pods are being deleted, they are terminated in reverse order, from {N-1..0}.
- Before a scaling operation is applied to a Pod, all of its predecessors must be Running and Ready.
- Before a Pod is terminated, all of its successors must be completely shutdown.

DaemonSet

Wednesday, March 18, 2020 4:09 PM

A DaemonSet ensure that all (or some) Nodes run a copy of a Pod. This is useful for

- running a cluster storage daemon, such as glusterd, ceph, on each node.
- running a logs collection daemon on every node, such as fluentd or filebeat.
- running a node monitoring daemon on every node, such as Prometheus Node Exporter, Flowmill, Sysdig Agent, collectd, Dynatrace OneAgent, AppDynamics Agent, Datadog agent, New Relic agent, Ganglia gmond, Instana Agent or Elastic Metricbeat.

Ingress

Thursday, March 19, 2020 1:46 PM

What is an Ingress?
An Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is defined on the Ingress resource.

To make an effective Ingress you must

- Have created an Ingress Controller
- Don't create an Ingress Controller manually (just use one of the standard like ingress-nginx

Creating an Ingress

```
apiVersion: networking.k8s.io/v1beta1

kind: Ingress

metadata:

  name: test-ingress

  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /

spec:

  rules:

- http:
```

TLS

```
    paths:
apiVersion: v1
      - path: /testpath
kind: Secret
    backend:
metadata:
      serviceName: test
  name: testsecret-tls

  namespace: default

data:

  tls.crt: base64 encoded cert

  tls.key: base64 encoded key

type: kubernetes.io/tls

apiVersion: networking.k8s.io/v1beta1

kind: Ingress
```

```
metadata:
  name: tls-example-ingress
spec:
  tls:
  - hosts:
    - sslexample.foo.com
      secretName: testsecret-tls
  rules:
    - host: sslexample.foo.com
      http:
      paths:
      - path: /
        backend:
        serviceName: service1
        servicePort: 80
```

Command Ref

Wednesday, March 18, 2020 3:03 PM

Intro

Friday, March 20, 2020 1:26 PM

What is Helm?
Helm is a package manager, similar to Homebrew, Apt, or Yum. Helm makes it easy to install and deploy complex applications in a Kubernetes cluster.

Three Big Concepts
A Chart is a Helm package, It contains all of the resource definitions to run the workload in you cluster. A Chart is similar to a Homebrew formula, Apt dpkg, or Yum RPM file.

A Repository is the place where charts reside. Repositories make it easy to distribute charts. There are many public repositories available which you can explore at https://hub.helm.sh . There is also the Kubernetes public https://kubernetes-charts.storage.googleapis.com/ .

A Release is an instance of chart running in a K8S cluster. One chart can often be installed many times into the same cluster. Each Release can then be managed separately from each other.

Installing Helm

Thursday, February 10, 2022 2:28 PM

To install helm please follow the install directions of your choice at Helm Install.

Command Ref

Friday, March 20, 2020 1:43 PM

Finding charts

- helm search hub searches the Helm Hub, which comprises helm charts from dozens of different repositories.
- helm search repo searches the repositories that you have added to your local helm client (with helm repo add). This search is done over local data, and no public network connection is needed.

After adding a repo or simply using the public hub.helm.sh you can search for charts using the following syntax.

```
helm search hub wordpress
            or
helm search <repo-name> <chart-name>
```

Helm using fuzzy search logic so it is not necessary to know the complete chart name. You can search for parts of the name.

Installing charts

To install a chart use the helm install . The command takes in a relase name and the chart name.

```
helm install <release-name> <chart-name>
helm install mischievous-mouse stable/mariadb
```

The release name can be left off and helm will generate a name, just use –generate-name

```
helm install stable/mariadb --generate-name
```

When installing charts Helm doesn't wait for all resources to startup. Use helm status <release-name> to track the install progress of a chart.

Customizing an Install

A simple helm install will install the chart with all of its default values. It is possible to customize the install by using helm show values <chart-name>

```
helm show values stable/mariadb
```

Declarative customization

Prior to an install it is possible to place your customized values in a yaml file and pass the file to the install command.

```
echo '{mariadbUser: user0, mariadbDatabase: user0database}' > config.yaml
helm install –f config.yaml stable/mariadb --generate-name
```

Either the –f or the –values option can be used to pass in a file. These options can be used multiple times in a single install command, the right most file values will have highest precedence.

Imperative customization

Single customize values can be passed at the command line using the –set option. This option can be

used multiple time in a single install command and can be used with the –values/-f options as well. The –set option will take precedence over –values/-f options. Values overridden with –set are persisted in a ConfigMap and can be viewed using helm get values <release-name> . Those values can also be reset using helm upgrade <release-name> --reset-values

--set formats
The –set option is translated to a YAML equivalent

- --set name=value
  - name: value
- --set a=b, c=d
  - a: b
  - c: d
- --set outer.inner=value
  - outer:
    inner: value
- --set name={a,b,c}
  - name:

    - a

    - b

    - c

Upgrade and Rollback
When a new version of a chart is released or when you want to change the configuration of a release helm upgrade <release-name> . Helm will upgrade only parts of the chart that have changed to reduce interference with operation.

If during upgrade a problem occurs during an upgrade use helm rollback <release-name> <revision> .

```
helm rollback happy-panda 1
```

The previous command will rollback the release name happy-panda to its first revision. Helm will track a release's history with every upgrade. Use helm history <release-name> to view a release's history

Uninstalling a release
helm uninstall <release-name>