Kubernetes For Beginners – An Introduction

Charles Wimmer

5 October 2019

This document provides an introduction to Kubernetes for the uninitiated. It is intended to give application developers enough information to get them started with Kubernetes deployments.

1 *What is Kubernetes?*

Kubernetes is a Greek word that means Captain or Pilot. It is also the root of the word cybernetic and gubernatorial. Kubernetes, the software, is designed to deploy, scale, and monitor distributed platforms. It abstracts Machines, Networks, and Storage into a declarative API. Kubernetes components running on nodes actively manage workloads to ensure the actual state meets the user's declared intentions. Kubernetes deploys and heals distributed applications for the user.

κῠβερνάω (*kubernáō*, "to steer") + -της (*-tēs*)

Figure 1: Kubernetes
Noun:
1. a captain, a steersman, a pilot, a navigator
2. (figuratively) a guide

THE PRIMARY INTERACTION between the application author and Kubernetes is through the Kubernetes API. The bulk of this document introduces the Kubernetes objects used to describe the desired state of application deployment. Objects are created by calls to the Kubernetes API.

2 *Pods*

Pods are the fundamental deployed unit in Kubernetes.[1] In comparison to previous technologies, a pod feels a little like a virtual machine. A pod is composed of one or more containers — all processes in a pod run in a container.

[1] Pods - kubernetes. `https://kubernetes.io/docs/concepts/workloads/pods/pod/`. Accessed: 2019-10-05

A container inside a pod is both a container image as well as all runtime configuration information needed to start the container. A container is a high-level concept wrapped around some primitive Linux features. Each container runs processes inside a Linux namespace. Each of the Linux namespaces[2] (Cgroup, Network, Mount, PID, IPC, UTS, as well as User) is used to isolate each running process from one another. An ENTRYPOINT is either defined in the container image or declared in the pod definition.

[2] namespaces(7) - linux manual page. `http://man7.org/linux/man-pages/man7/namespaces.7.html`. Accessed: 2019-10-05

Containers running in a pod share some characteristics with processes running in virtual machines. Each container can see the other when running commands like `ps`. Each container in a pod can see daemons running in the same pod on `localhost`. All processes that open connections outside of the pod share the same external IP address. Containers may optionally share filesystems. Application authors may define filesystems that are private to individual containers, and they may also define filesystems to share between containers.

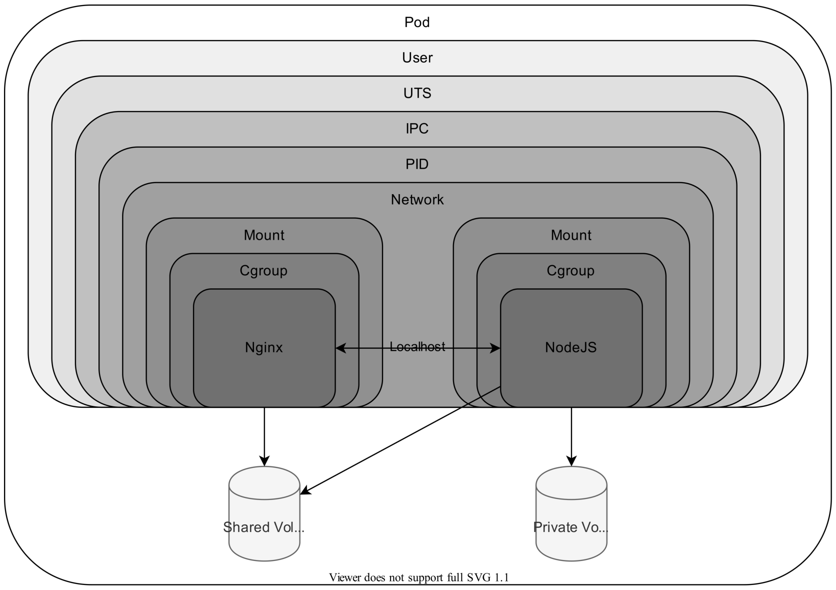

Figure 2: Namespaces and Volumes in a Pod

TODO: Insert an image of pod networking here

Figure 3: Simple Pod example

```
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
spec:
  containers:
  - name: ubuntu
    image: ubuntu:trusty
    command: ["echo"]
    args: ["Hello World"]
```

3 *Labels*

Labels are key-value pairs attached to objects (Pods, for example)[3].

Labels allow application authors to imply meaning to Kubernetes objects. For example, they allow us to express which pods are in development or which pods are in the front-end tier of an application.

[3] Labels and selectors. `https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/`. Accessed: 2019-10-05

Labels do not change the behavior of the core system. Kubernetes stores and reports labels on objects. Kubernetes does not change its behavior based on labels.

```
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
  labels:
    application: my-app
    version: "v31"
    release: "r42"
    stage: production
spec:
  containers:
  - name: ubuntu
    image: ubuntu:trusty
    command: ["echo"]
    args: ["Hello World"]
```

Figure 4: Pod with labels

```
kubectl label pods pod-example environment=production
```

Figure 5: Label a pod

```
kubectl get pods -l environment=production
```

Figure 6: Pod query with labels

4 *ReplicationControllers*

A ReplicationController is a template for managing any number of copies of a pod[4]. It manages the pod lifecycle, including scaling up and down. It is useful for retaining targeted replication levels even after destructive maintenance like an operating system upgrade or a hard drive replacement.

[4] Replication controller. `https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller/`. Accessed: 2019-10-05

ReplicationControllers exhibit characteristics that are sometimes useful when debugging applications. Pods are associated with ReplicationControllers by labels and selectors. A ReplicationController knows it has the appropriate number of replicas when a query to the Kubernetes API returns the correct number of pods. The ReplicationController runs the equivalent of `kubectl get pods -l <labels>`.

Assume for a moment one wishes to save a pod for debugging while retaining sufficient replicas for serving production load. If one removes or replaces the label(s) observed by the Replication-Controller, it will spin up another in order to converge toward the desired number of replicas. The pod is available for later analysis; the ReplicationController no longer manages the lifecycle.

TODO: Graphic depicting debug action

```
---
apiVersion: v1
kind: ReplicationController
metadata:
  name: webserver-replicationcontroller
spec:
  replicas: 3
  selector:
    app: nginx
  template:
    metadata:
      name: webserver-pod
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx-container
        image: nginx:1.7.10
        ports:
        - containerPort: 80
```

Figure 7: Replication Controller

5 *Deployments*

Deployments provide all the features of ReplicationControllers and more[5]. Deployments orchestrate ReplicationControllers. In practice, an application author would never need to use a ReplicationController directly. They should choose Deployments instead.[6]

Unlike Replication Controllers, Deployments support rolling upgrades of container versions and config parameters. Deployments may be updated to start the rollout of a new version of a Pod. The Deployment's controller will create a new ReplicationController with the new version. It will manage the scale-up and scale-down of new and old controllers until the Pods are all upgraded.

TODO: Image to describe upgrade

[5] Deployments. `https://kubernetes.io/docs/concepts/workloads/controllers/deployment/`. Accessed: 2019-10-05

[6] As a counter-example, Spinnaker uses Replication Controllers to scale applications up and down.

Figure 8: 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:1.7.10
        ports:
        - containerPort: 80
```

6 *ConfigMaps*

ConfigMaps are used to store configuration for an application[7]. A configuration item may be expressed as either entire configuration files or as key-value pairs, such as environmental variables.

Data in ConfigMaps may be used directly to populate environmental variables for containers. Data in Configmaps may also be materialized as a file in a container by way of a ConfigMap Volume.

Beginning with Figure 9, we see some examples of ConfigMap usage.

[7] Configure a pod to use a configmap. `https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/`. Accessed: 2019-10-05

Figure 9: ConfigMap

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-config-3
data:
  game-special-key: |
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
```

Figure 10: Pod using ConfigMap as environmental variable

```
---
apiVersion: v1
kind: Pod
metadata:
  name: game-test-pod
spec:
  containers:
    - name: game-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        # Define the environment variable
        - name: SEKRET_PASSPHRASE
          valueFrom:
            configMapKeyRef:
              # The ConfigMap containing the value you want to assign
              # to SEKRET_PASSPHRASE
              name: game-config-3
              # Specify the key associated with the value
              key: secret.code.passphrase
  restartPolicy: Never
```

Figure 11: Pod using ConfigMap as file

```
---
apiVersion: v1
kind: Pod
metadata:
  name: game-test-pod
spec:
  containers:
    - name: game-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/sh", "-c", "ls /etc/config/" ]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        # Provide the name of the ConfigMap containing the files you
        # want to add to the container
        name: game-config-3
  restartPolicy: Never
```

```
---
apiVersion: v1
kind: Pod
metadata:
  name: game-test-pod
spec:
  containers:
    - name: game-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/sh","-c","cat /etc/config/keys" ]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: game-config-3
        items:
        - key: secret.code.passphrase
          path: keys
          restartPolicy: Never
```

Figure 12: Pod using ConfigMap item as file

7 *Secrets*

Application authors use Secret objects for storing items such as passwords, keys, and tokens[8]. Secrets are marginally safer and more flexible than ConfigMaps for these cases.

[8] Secrets. `https://kubernetes.io/docs/concepts/configuration/secret/`. Accessed: 2019-10-05

A Secret is similar to a ConfigMap with a few exceptions. The sensitive fields in a Secret are Base64 obfuscated when reading from the API. Secrets are encrypted with a symmetric key by the Kubernetes APIserver and stored in the etcd database.

Similar to ConfigMaps, the Kubelet materializes the contents of Secret objects on a container's file system. Unlike ConfigMaps, the Kubernetes scheduler validates Secret existence at Pod scheduling time. If an application author defines a Pod with a missing Secret, the Pod will never get scheduled. When an application author defines a Pod with a missing ConfigMap, the Pod will start without the file.

Another limitation of Secrets that an application author must keep in mind is the limited storage capacity. Unlike ConfigMaps, Kubernetes limits the size of a Secret to 1MB. The limitation is in place for two reasons. First, to limit the likelihood of a DOS on the Kubernetes API. Second, to allow Kubelet implementations that never materialize the secret material on the Node's filesystem.

Figure 13: Secret

```
---
kind: Secret
apiVersion: v1
metadata:
  name: dotfile-secret
data:
  .secret-file: dmFsdWUtMg0KDQo=
---
kind: Pod
apiVersion: v1
metadata:
  name: secret-dotfiles-pod
spec:
  volumes:
  - name: secret-volume
    secret:
      secretName: dotfile-secret
  containers:
  - name: dotfile-test-container
    image: k8s.gcr.io/busybox
    command:
    - ls
    - "-l"
    - "/etc/secret-volume"
    volumeMounts:
    - name: secret-volume
      readOnly: true
      mountPath: "/etc/secret-volume"
```

8 Services

Pods are mortal. Kubernetes starts and stops Pods when a deployment does a rolling upgrade, for example. They fail when hardware fails. In a distributed system or a microservice architecture, components must be able to access consistently other components upon which they depend.

Services provide this consistency. Services are persistent references to a group of pods performing the same function[9]. Services publish DNS records as a discovery mechanism.

[9] Service. `https://kubernetes.io/docs/concepts/services-networking/service/`. Accessed: 2019-10-05

Services are associated with Pods via a Label. Application authors configure Services with a NodeSelector. Pods become members of a Service if their Labels match the Service's NodeSelector.

Application authors may consider Services as if they were load balancers internal to Kubernetes. They act as a VIP layer for an application.

In addition to referring to private pods as a Service, a Service object may refer to an external service either by IP address or by DNS record.

Figure 14: Service backed by Pods

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

The service in Figure 15 has no selector. There is no implicit endpoint object created.

```
---
kind: Service
apiVersion: v1
metadata:
  name: my-service
  namespace: qa
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
--
kind: Endpoints
apiVersion: v1
metadata:
  name: my-service
subsets:
  - addresses:
      - ip: 1.2.3.4
    ports:
      - port: 9376
```

Figure 15: Service external to cluster, defined by IP address

The service in Figure 16 has no selector. DNS will contain a CNAME to external DNS

```
---
kind: Service
apiVersion: v1
metadata:
  name: my-service
  namespace: prod
spec:
  type: ExternalName
  externalName: my.database.example.com
```

Figure 16: Service external to cluster, aliased to DNS entry

9 *Volumes*

Volumes expose persistent and ephemeral storage to pods[10]. The lifecycle of a Volume is the same as the Pod that encloses it.

Kubernetes Volumes differ from the Docker feature of the same name. It is important to understand these differences.

[10] Volumes. `https://kubernetes.io/docs/concepts/storage/volumes/`. Accessed: 2019-10-05

A Volume is a directory, possibly with some data in it, accessible to the containers in a pod.

Some example Volume implementations include:

- awsElasticBlockStore
- rds
- glusterfs
- nfs
- iscsi
- gcePersistentDisk
- hostPath

Figure 17: Volumes

```
---
apiVersion: v1
kind: Pod
metadata:
  name: test-ebs
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /test-ebs
      name: test-volume
  volumes:
  - name: test-volume
    # This AWS EBS volume must already exist.
    awsElasticBlockStore:
      volumeID: <volume-id>
      fsType: ext4
```

10 Namespaces

Namespaces provide a mechanism for logical grouping of Kubernetes objects[11]. Application authors may think of Namespaces as multiple virtual clusters backed by a single physical cluster. It allows for isolation of workloads between users, groups, projects, or development environments.

[11] Namespaces. https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/. Accessed: 2019-10-05

Namespaces provide isolation of workloads by providing a separate scope for Names. A Deployment object by the same name may exist in multiple namespaces. Each Namespace may be allocated a ResourceQuota, thus limiting the use of physical resources of Nodes.

Namespaces are also scoped in DNS. Each Namespace effectively controls its subdomain of the cluster DNS. For example, a service in a namespace has a DNS name of the form: <service>.<namespace>.svc.cluster.local.

There are Kubernetes components that are not scoped by Namespaces but are rather scoped at the Cluster level. Some examples of non-Namespaced objects are:

- Nodes
- PersistentVolumes

For a difinitive list of each type, use these commands:

```
# In a namespace
kubectl api-resources --namespaced=true

# Not in a namespace
kubectl api-resources --namespaced=false
```

11 Architecture

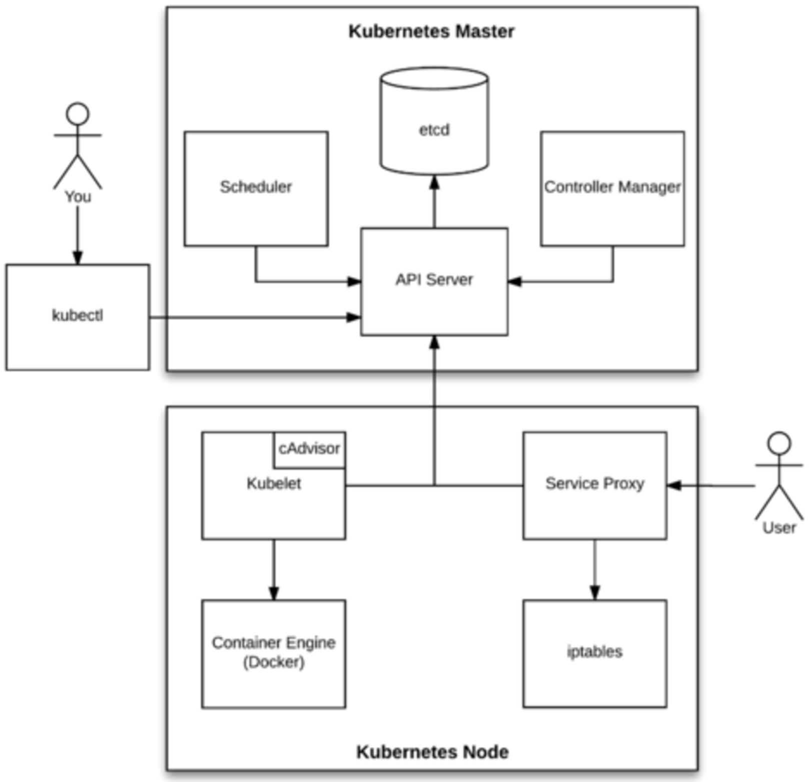

Figure 18: Kubernetes Architecture

11.1 *APIserver*

The APIserver component is the heart of Kubernetes. All components communicate to one another via the APIserver. It enforces authentication and authorization of API calls. It is the component that persists to the etcd databse.

11.2 *Scheduler*

The Scheduler component watches for workloads and decides where they should run. It is the component that binds Pods to Nodes.

11.3 *Controller-manager*

The Controller-manager component is where the core control loops are located. For example:

- Replication Controller
- Endpoints Controller
- Namespace Controller

In addition to controllers, it also manages all the life cycle functions of the cluster.

- event garbage collection
- terminated pod garbage collection
- cascading deletion garbage collection
- node garbage collection

11.4 *Service Proxy*

The ServiceProxy watches for changes to services and pods. It keeps the network configuration up to date.

11.5 *Container Runtime*

The container runtime is responsible for launching and configuring processes inside Linux namespaces on Nodes. Docker and cri-o are the currently supported Container Runtimes. rkt is also supported, but it has been deprecated.

11.6 *Kubelet*

Kubelet is the component that runs on each distributed Node that provides resources. Kubelet watches the API for the desired state of the Node. Also, it assures each of the assigned Pods is running. It communicates with the Container Runtime on the system over the provided API to affect state changes to containers and query status. Kubelet is responsible for running the health checks defined in each Pod specification. Also, Kubelet retrieves metrics about resource utilization from the kernel.

References

Configure a pod to use a configmap. `https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/`. Accessed: 2019-10-05.

Deployments. `https://kubernetes.io/docs/concepts/workloads/controllers/deployment/`. Accessed: 2019-10-05.

Labels and selectors. `https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/`. Accessed: 2019-10-05.

namespaces(7) - linux manual page. `http://man7.org/linux/man-pages/man7/namespaces.7.html`. Accessed: 2019-10-05.

Namespaces. `https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/`. Accessed: 2019-10-05.

Pods - kubernetes. `https://kubernetes.io/docs/concepts/workloads/pods/pod/`. Accessed: 2019-10-05.

Replication controller. `https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller/`. Accessed: 2019-10-05.

Secrets. `https://kubernetes.io/docs/concepts/configuration/secret/`. Accessed: 2019-10-05.

Service. `https://kubernetes.io/docs/concepts/services-networking/service/`. Accessed: 2019-10-05.

Volumes. `https://kubernetes.io/docs/concepts/storage/volumes/`. Accessed: 2019-10-05.