Learning Objectives

By the end of this chapter, you should be able to:

- Discuss Kubernetes.
- Learn the basic Kubernetes terminology.
- Discuss the configuration tools.
- Learn what community resources are available.

Basics of Kubernetes

What Is Kubernetes?

Running a container on a laptop is relatively simple. But, connecting containers **across multiple hosts**, **scaling** them, **deploying applications without downtime**, and **service discovery among several aspects**, can be difficult.

Kubernetes addresses those challenges from the start with a set of primitives and a powerful open and extensible API. The ability to add new objects and controllers allows easy customization for various production needs.

According to the kubernetes.io website, Kubernetes is:

```
"an open-source system for automating deployment, scaling, and management of
containerized applications".
```

A key aspect of Kubernetes is that it builds on 15 years of experience at Google in a project called borg.

Google's infrastructure started reaching high scale before virtual machines became pervasive in the datacenter, and containers provided a fine-grained solution for packing clusters efficiently. Efficiency in using clusters and managing distributed applications has been at the core of Google challenges.

Components of Kubernetes

Deploying containers and using Kubernetes may require a change in the development and the system administration approach to deploying applications. In a traditional environment, an application (such as a web server) would be a monolithic application placed on a dedicated server. As the web traffic increases, the application would be tuned, and perhaps moved to bigger and bigger hardware. After a couple of years, a lot of customization may have been done in order to meet the current web traffic needs.

Instead of using a large server, Kubernetes approaches the same issue by deploying a large number of small web servers, or microservices. The server and client sides of the application expect that there are many possible agents available to respond to a request. It is also important that clients expect the server processes to die and be replaced, leading to a transient server deployment. Instead of a large Apache web server with many httpd daemons responding to page requests, there would be many nginx servers, each responding.

The transient nature of smaller services also allows for decoupling. Each aspect of the traditional application is replaced with a dedicated, but transient, microservice or agent. To join these agents, or their replacements together, we use services and API calls. A service ties traffic from one agent to another (for example, a frontend web server to a backend database) and handles new IP or other information, should either one die and be replaced.

Communication to, as well as internally, between components is API call-driven, which allows for flexibility. Configuration information is stored in a JSON format, but is most often written in YAML. Kubernetes agents convert the YAML to JSON prior to persistence to the database.

Challenges

Containers have seen a huge rejuvenation in the past few years. They provide a great way to package, ship, and run applications - that is the Docker motto.

The developer experience has been boosted tremendously thanks to containers. Containers, and Docker specifically, have empowered developers with ease of building container images, simplicity of sharing images via Docker registries, and providing a powerful user experience to manage containers.

However, managing containers at scale and architecting a distributed application based on microservices' principles is still challenging.

You first need a continuous integration pipeline to build your container images, test them, and verify them. Then, you need a cluster of machines acting as your base infrastructure on which to run your containers. You also need a system to launch your containers, and watch over them when things fail and self-heal. You must be able to perform rolling updates and rollbacks, and eventually tear down the resource when no longer needed.

All of these actions require flexible, scalable, and easy-to-use network and storage. As containers are launched on any worker node, the network must join the resource to other containers, while still keeping the traffic secure from others. We also need a storage structure which provides and keeps or recycles storage in a seamless manner.

One of the biggest challenges to adoption is the applications themselves, inside the container. They need to be written, or re-written, to be truly transient. If you were to deploy Chaos Monkey, which would terminate *any* containers, would your customers notice?

Other Solutions

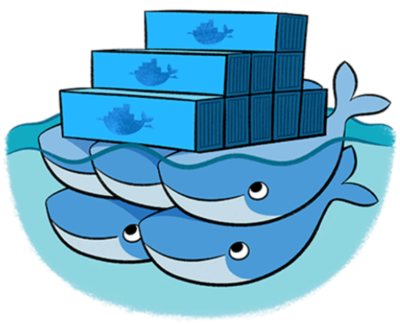
Built on open source and easily extensible, Kubernetes is definitely a solution to manage containerized applications. There are other solutions as well.

Managing Containerized Applications

Close ^

Docker Swarm

Docker Swarm is the solution provided by Docker Inc. It has been re-architected recently and is based on SwarmKit. It is embedded with the Docker Engine.



Close ^

Apache Mesos

Apache Mesos is a data center scheduler, which can run containers through the use of *frameworks*. Marathon is the framework that lets you orchestrate containers.

Close ^

Nomad

Nomad from HashiCorp, the makers of Vagrant and Consul, is another solution for managing containerized applications. Nomad schedules tasks defined in *Jobs*. It has a Docker driver which lets you define a running container as a task.

Close ^

Rancher

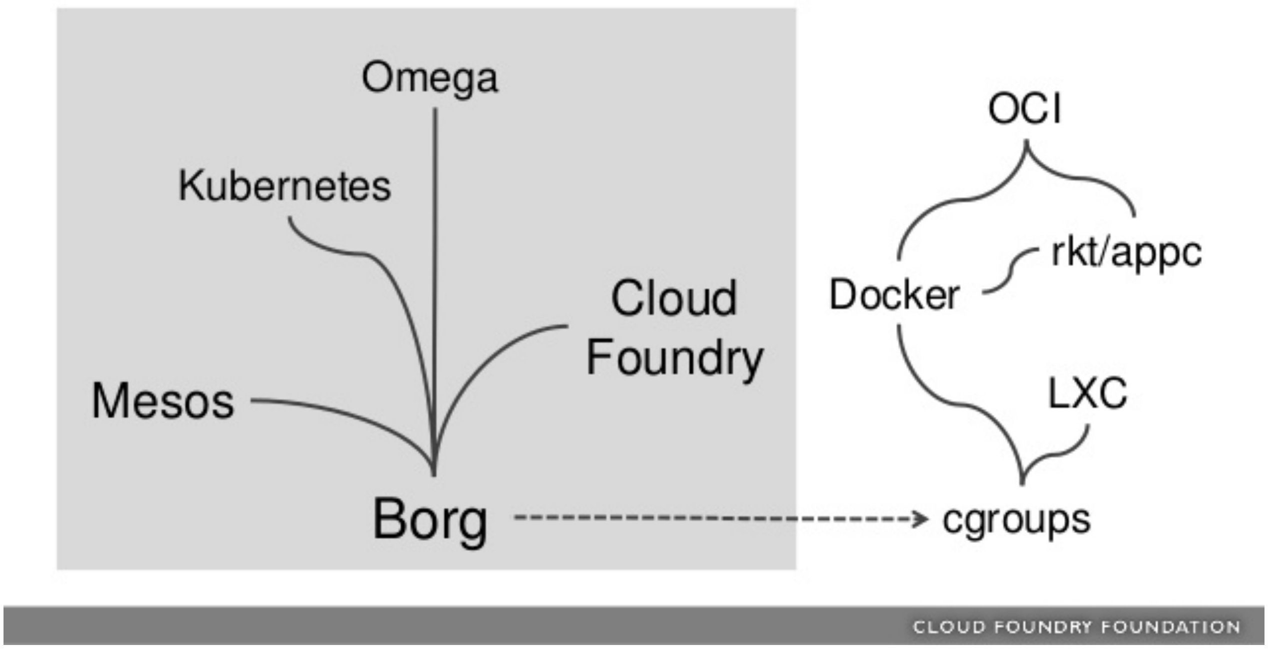
Rancher is a container orchestrator-agnostic system, which provides a single pane of glass interface to manage applications. It supports Mesos, Swarm, Kubernetes.

Borg Heritage

Borg Heritage

What primarily distinguishes Kubernetes from other systems is its heritage. Kubernetes is inspired by Borg - the internal system used by Google to manage its applications (e.g. Gmail, Apps, GCE).

With Google pouring the valuable lessons they learned from writing and operating Borg for over 15 years into Kubernetes, this makes Kubernetes a safe choice when having to decide on what system to use to manage containers. While a powerful tool, part of the current growth in Kubernetes is making it easier to work with and handle workloads not found in a Google data center.



The Kubernetes Lineage

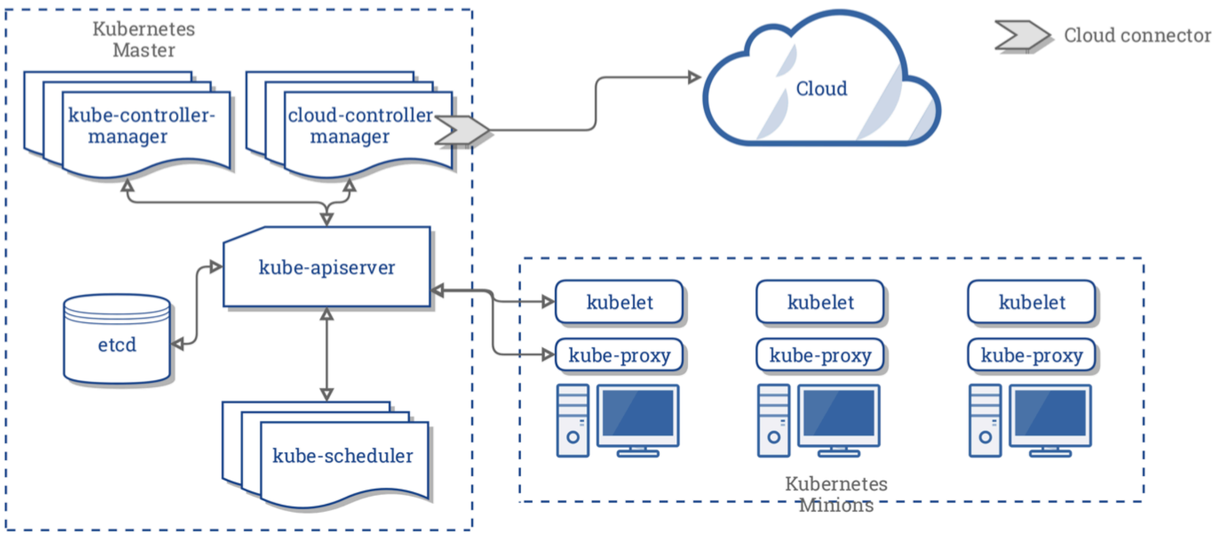
Borg has inspired current data center systems, as well as the underlying technologies used in container runtime today. Google contributed cgroups to the Linux kernel in 2007; it limits the resources used by collection of processes. Both cgroups and Linux namespaces are at the heart of containers today, including Docker.

Mesos was inspired by discussions with Google when Borg was still a secret. Indeed, Mesos builds a multi-level scheduler, which aims to better use a data center cluster.

The Cloud Foundry Foundation embraces the 12 factor application principles. These principles provide great guidance to build web applications that can scale easily, can be deployed in the cloud, and whose build is automated. Borg and Kubernetes address these principles as well.

Kubernetes Architecture

To quickly demistify Kubernetes, let's have a look at the *Kubernetes Architecture* graphic, which shows a high-level architecture diagram of the system components. Not all components are shown. Every node running a container would have **kubelet** and **kube-proxy**, for example.



Kubernetes Architecture
Retrieved from the Kubernetes documentation **-** *[Concepts Underlying the Cloud Controller Manager]()*

In its simplest form, Kubernetes is made of a central manager (aka **master**) and some **worker** nodes, once called minions (we will see in a follow-on chapter how you can actually run everything on a single node for testing purposes). The manager runs an API server, a scheduler, various controllers and a storage system to keep the state of the cluster, container settings, and the networking configuration.Kubernetes exposes an API via the API server. You can communicate with the API using a local client called **kubectl** or you can write your own client and use **curl** commands. The **kube-scheduler** is forwarded the requests for running containers coming to the API and finds a suitable node to run that container. Each node in the cluster runs two processes: a kubelet and kube-proxy. The kubelet receives requests to run the containers, manages any necessary resources and watches over them on the local node. The kubelet interacts with the local container engine, which is Docker by default, but could be rkt or cri-o, which is growing in popularity. The **kube-proxy** creates and manages networking rules to expose the container on the network.Using an API-based communication scheme allows for non-Linux worker nodes and containers. Support for Windows Server 2019 was graduated to *Stable* with the 1.14 release. Only Linux nodes can be master on a cluster.

Terminology

We have learned that Kubernetes is an orchestration system to deploy and manage containers. Containers are not managed individually; instead, they are part of a larger object called a **Pod**. A Pod consists of one or more containers which share an IP address, access to storage and namespace. Typically, one container in a Pod runs an application, while other containers support the primary application.

Orchestration is managed through a series of watch-loops, or **controllers**. Each controller interrogates the **kube-apiserver** for a particular object state, modifying the object until the declared state matches the current state. These controllers are compiled into the **kube-controller-manager**. The default, newest, and feature-filled controller for containers is a **Deployment**. A Deployment ensures that resources declared in the **PodSpec** are available, such as IP address and storage, and then deploys a ReplicaSet. The ReplicaSet is a controller which deploys and restarts pods, which declares to the container engine, Docker by default, to spawn or terminate a container until the requested number is running. Previously, the function was handled by the ReplicationController, but has been obviated by Deployments. There are also **Jobs** and **CronJobs** to handle single or recurring tasks, among others.

To manage thousands of Pods across hundreds of nodes can be a difficult task. To make management easier, we can use **labels**, arbitrary strings which become part of the object metadata. These can then be used when checking or changing the state of objects without having to know individual names or UIDs. Nodes can have **taints** to discourage Pod assignments, unless the Pod has a **toleration** in its metadata.

There is also space in metadata for **annotations** which remain with the object but cannot be used by Kubernetes commands. This information could be used by third-party agents or other tools.

Innovation

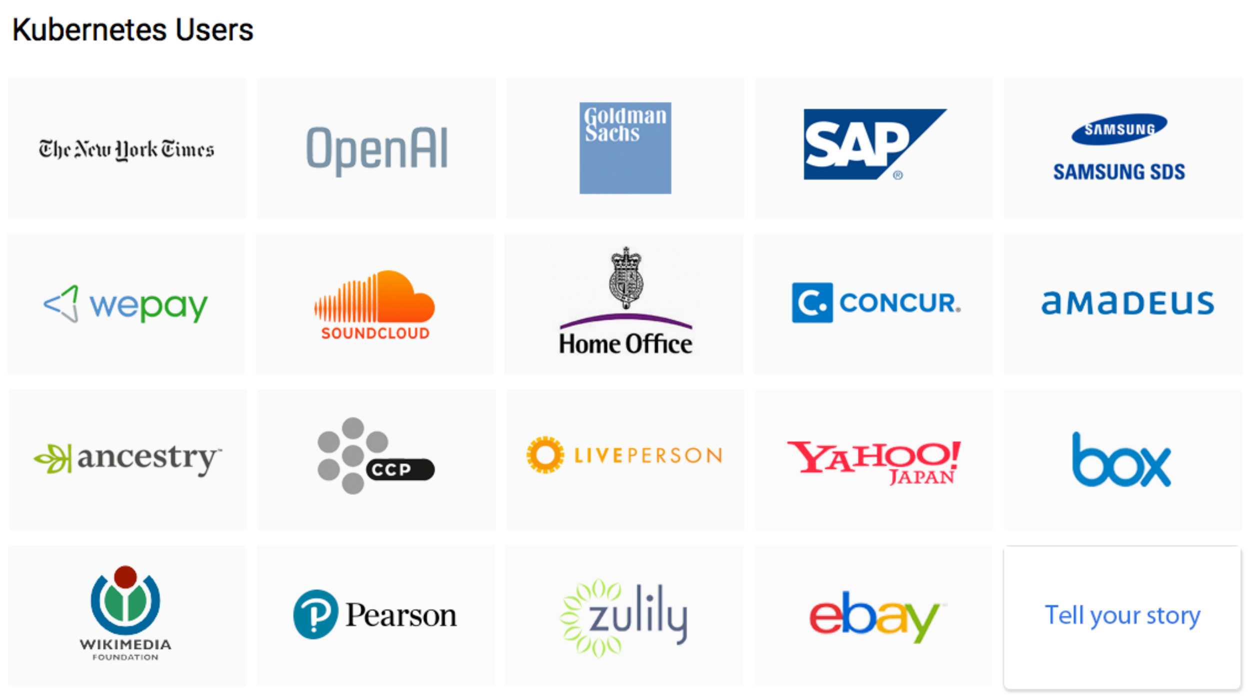
Since Its inception, Kubernetes has seen a terrific pace of innovation and adoption. The community of developers, users, testers, and advocates is continuously growing every day. The software is also moving at an extremely fast pace, which is even putting GitHub to the test:

- Given to open source in June 2014
- Thousands of contributors
- More than 83k commits
- More than 28k on Slack
- Currently, on a three month major release cycle
- Constant changes.

User Community

Kubernetes is being adopted at a very rapid pace. To learn more, you should check out the case studies presented on the Kubernetes website. Ebay, Box, Pearson and Wikimedia have all shared their stories.

Pokemon Go, the fastest growing mobile game, also runs on Google Container Engine (GKE), the Kubernetes service from Google Cloud Platform (GCP).



Retrieved from the Kubernetes website

Tools

There are several tools you can use to work with Kubernetes. As the project has grown, new tools are made available, while old ones are being deprecated. **Minikube** is a very simple tool meant to run inside of VirtualBox. If you have limited resources and do not want much hassle, it is the easiest way to get up and running. We mention it for those who are not interested in a typical production environment, but want to use the tool.

Our labs will focus on the use of **kubeadm** and **kubectl**, which are very powerful and complex tools.

There are third-party tools as well, such as Helm, an easy tool for using Kubernetes charts, and Kompose to translate Docker Compose files into Kubernetes objects. Expect these tools to change often!

Cloud Native Computing Foundation (CNCF)

Kubernetes is an open source software with an Apache license. Google donated Kubernetes to a newly formed collaborative project within The Linux Foundation in July 2015, when Kubernetes reached the v1.0 release. This project is known as the Cloud Native Computing Foundation (CNCF).

CNCF is not just about Kubernetes, it serves as the governing body for open source software that solves specific issues faced by cloud native applications (i.e. applications that are written specifically for a cloud environment).

CNCF has many corporate members that collaborate, such as Cisco, the Cloud Foundry Foundation, AT&T, Box, Goldman Sachs, and many others.