Selfie Shots - Tools 304 Kubernetes

Persistent University

Introduction to Kubernetes

Key learning points :

- What is Kubernetes
- Components of Kubernetes
- Challenges with container orchestration
- Similar solutions available
- Kubernetes Architecture in brief

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.
- In Greek, **κνβερνητης** means the Helmsman, or pilot of the ship. Keeping with the maritime theme of Docker containers, Kubernetes is the pilot of a ship of containers.
- Due to the difficulty in pronouncing the name, many will use a nickname, **K8s**, as Kubernetes has eight letters. The nickname is said like **Kate's**.

The “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.
- To learn more about the ideas behind Kubernetes, you can read the *Large-scale cluster management at Google with Borg paper*.
- 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.

Persistent

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.
- Kubernetes is written in **Go Language**, a portable language which is like a hybridization between C++, Python, and Java. Some claim it incorporates the best (while some claim the worst) parts of each.

Challenges

- Containers have seen a huge rejuvenation in the past three 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.

Similar Solutions available

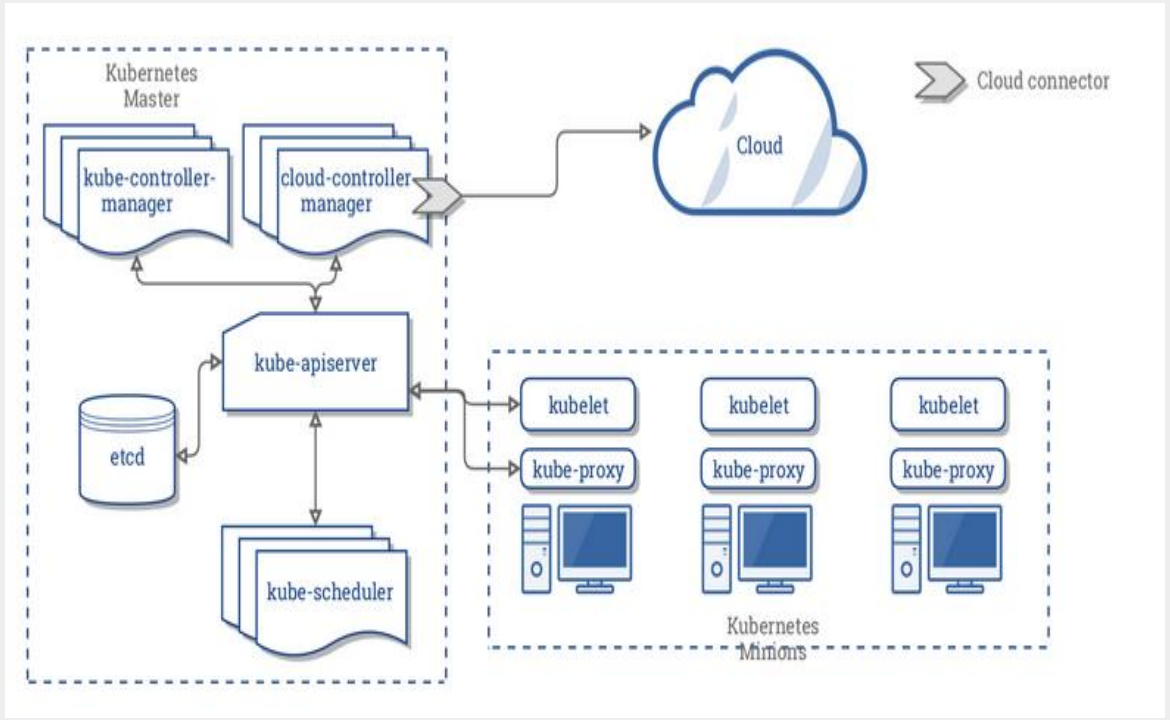
- Built on open source and easily extensible, **Kubernetes** is definitely a solution to manage containerized applications.
- There are other solutions as well, including:
- **Docker Swarm** is the Docker Inc. solution. It has been re-architected recently and is based on SwarmKit. It is embedded with the **Docker Engine**.
- **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.
- **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.
- **Rancher** is a container orchestrator-agnostic system, which provides a single pane of glass interface to managing applications. It supports **Mesos**, **Swarm**, **Kubernetes**.

Kubernetes Architecture

- In its simplest form, **Kubernetes** is made of
- a central manager (aka master) and some worker nodes.

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.



- The **kube-scheduler** sees the requests for running containers coming to the API and finds a suitable node to run that container in.
- Each node in the cluster runs two processes: a **kubelet** and a **kube-proxy**. The **kubelet** receives requests to run the containers, manages any necessary resources and watches over them on the local node.
- The **kube-proxy** creates and manages networking rules to expose the container on the network.

Summary

At the end of this session, we see that you are now able to

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

Lab Exercise

- **Visit kubernetes.io**
  - With such a fast changing project, it is important to keep track of updates. The main place to find documentation of the current version is https://kubernetes.io/.
- **Track Kubernetes Issues**
  - There are hundreds, perhaps thousands, working on Kubernetes every day. With that many people working in parallel there are good resources to see if others are experiencing a similar outage. Both the source code as well as feature and issue tracking are currently on github.com
  - To view the main page use your browser to visit https://github.com/kubernetes/kubernetes/
  - Click on various sub-directories and view the basic information available.
  - Update your URL to point to https://github.com/kubernetes/kubernetes/issues. You should see a series of issues, feature requests, and support communication.