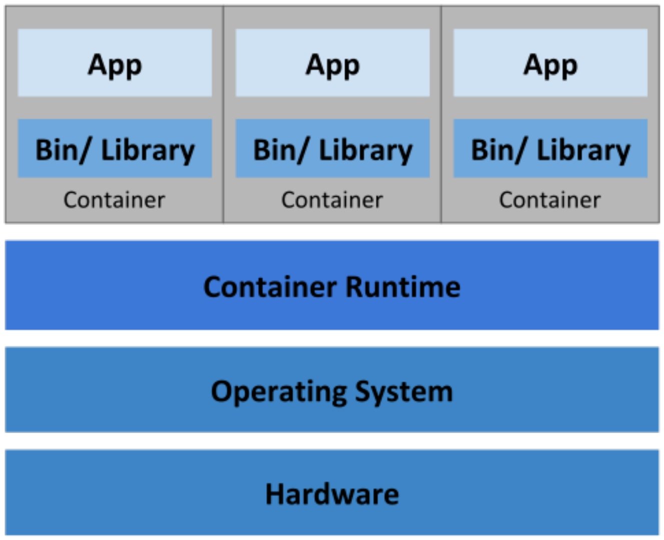
Microservices:

- Microservices-based architecture is aligned with Event-driven Architecture and Service-Oriented Architecture (SOA) principles
- Seamless upgrades and patching processes are other benefits of microservices architecture

....

- **<u>container runtimes</u>** like **runC**, **containerd**, or **cri-o** we can use those pre-packaged images, to create one or more containers.



Container Deployment

- **Containers** are an application-centric(Focusing on the application as the foundation or starting point) method to deliver high-performing, scalable applications on any infrastructure of your choice.

- **Microservices** are lightweight applications written in various modern programming languages, with specific dependencies, libraries and environmental requirements.

- Containers **encapsulate** microservices and their dependencies but do not run them directly. Containers run container images.

- A **container image** bundles the application along with its runtime, libraries, and dependencies, and it represents the source of a container deployed to offer an isolated executable environment for the application

Kubernetes

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

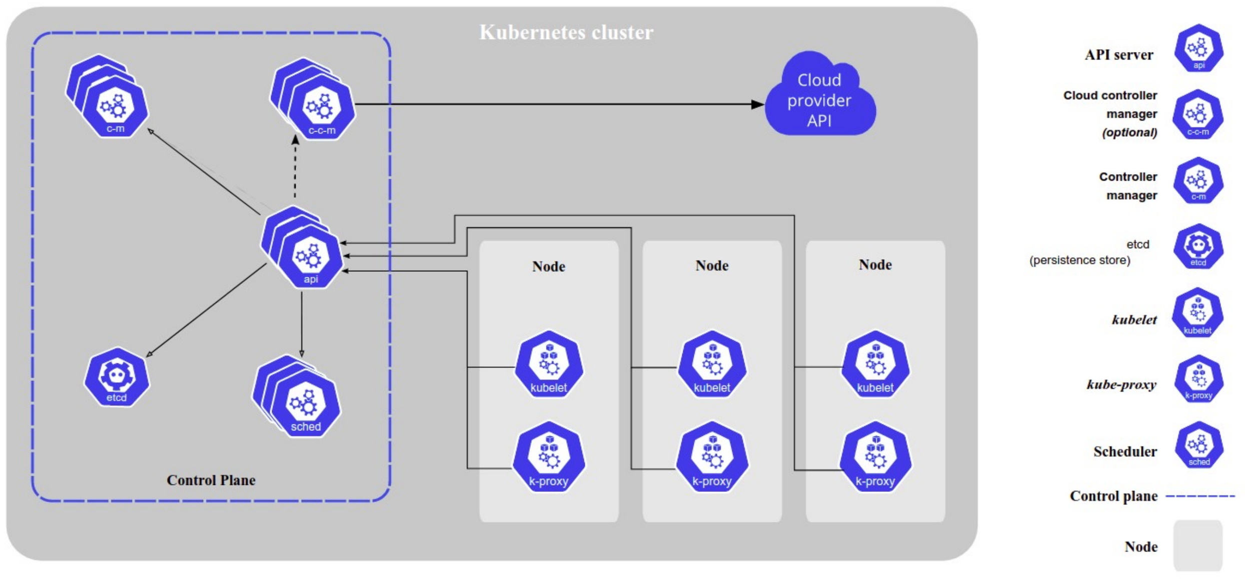
  Kubernetes comes from the Greek word **κυβερνήτης**, which means *helmsman* or *ship pilot*. With this analogy in mind, we can think of Kubernetes as the pilot on a ship of containers.

Google's Borg

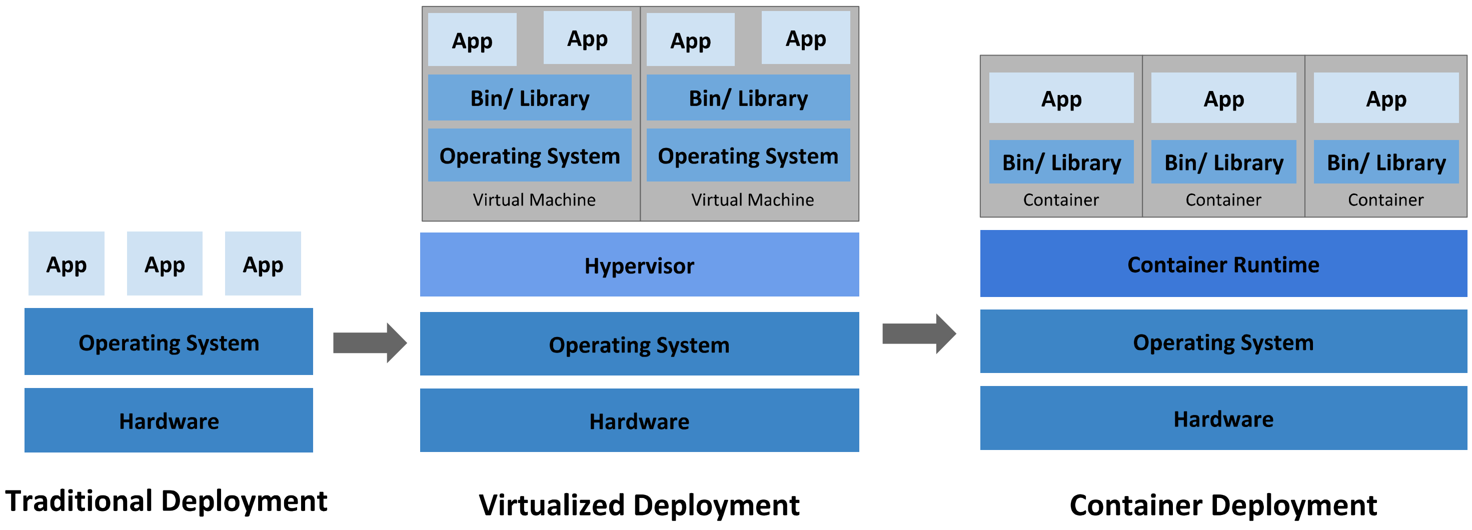
- *"Google's Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines"*

Kubernetes has the following main components:

- One or more **master nodes**, part of the **control plane**
- One or more **worker nodes**.



Components of Kubernetes Architecture



The technical definition of orchestration is execution of a defined workflow: first do A, then B, then C

Controle Plane:

components→:

- API server
- c-c-m (cloud controller manager)
- c-m (controller manager) : where the core kubernetes logic happen
- etcd : all cluster configuration data is saved to **etcd,** is a distributed key-value store which only holds cluster state related data.
- sched (schedular)

The **data plane** forwards traffic to other services via sidecars,
while the **control plane** handles configuration, administrative, security and monitoring related functions.

- In order to communicate with the Kubernetes cluster, users send requests to the control plane via a Command Line Interface (**CLI**) tool, a Web User-Interface (**Web UI**) Dashboard, or Application Programming Interface (**API**).

- A **master node** runs the following control plane components:

1. API Server
2. Scheduler
3. Controller Managers
4. Data Store.

In addition, the master node runs:

5. Container Runtime
6. Node Agent
7. Proxy.

....

API Server:

- All the administrative tasks are coordinated by the **kube-apiserver**, a central control plane component **running** on the master node
- The API Server **intercepts RESTful calls** from users, operators and external agents, then validates and processes them. During processing the API Server reads the Kubernetes cluster's current state from the etcd data store, and after a call's execution, the resulting state of the Kubernetes cluster is saved in the distributed key-value data store for persistence
- The API Server is the **only master plane component to talk to the etcd** data store, both to read from and to save Kubernetes cluster state information - acting as a middle interface for any other control plane agent inquiring about the cluster's state.

Scheduler:

- The role of the **kube-scheduler** is to assign new workload objects, such as pods, to nodes.
- During the scheduling process, decisions are made based on current **Kubernetes cluster state and new object's requirements.**
- Once all the cluster data is available, the scheduling algorithm filters the nodes with predicates to isolate the possible node candidates which then are scored with priorities in order to select the one node that satisfies all the requirements for hosting the new workload. The **outcome of the decision process is communicated back to the API Server**, which then delegates the workload deployment with other control plane agents.

Controller Managers:

- are control plane components on the master node running controllers **to regulate the state of the Kubernetes cluster.**
- Controllers are **watch-loops** continuously running and comparing the cluster's desired state (provided by objects' configuration data) with its current state (obtained from etcd data store via the API server). In case of a mismatch corrective action is taken in the cluster until its current state matches the desired state.

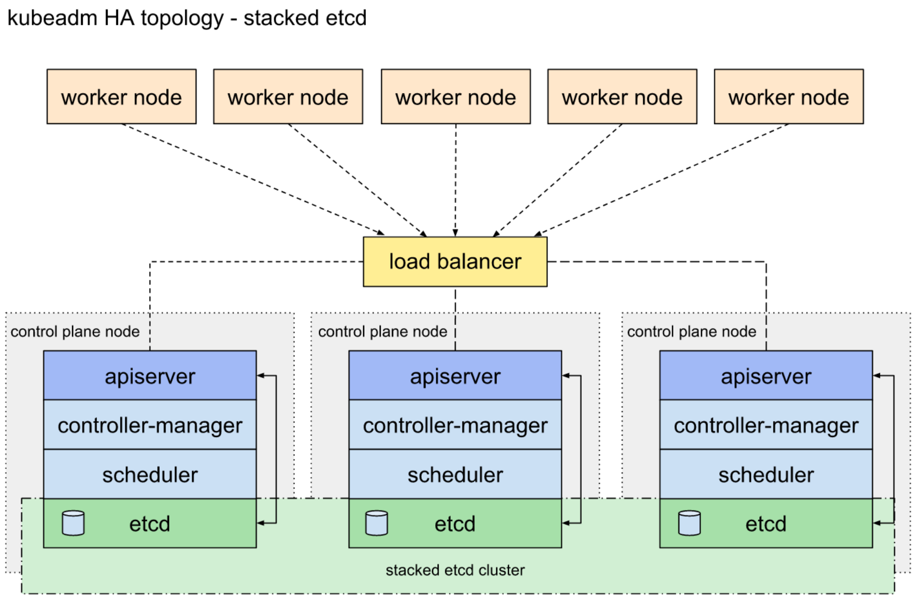
  The **kube-controller-manager** runs controllers responsible to act when nodes become unavailable, to ensure pod counts are as expected, to create endpoints, service accounts, and API access tokens.

  The **cloud-controller-manager** runs controllers responsible to interact with the underlying infrastructure of a cloud provider when nodes become unavailable, to manage storage volumes when provided by a cloud service, and to manage load balancing and routing.

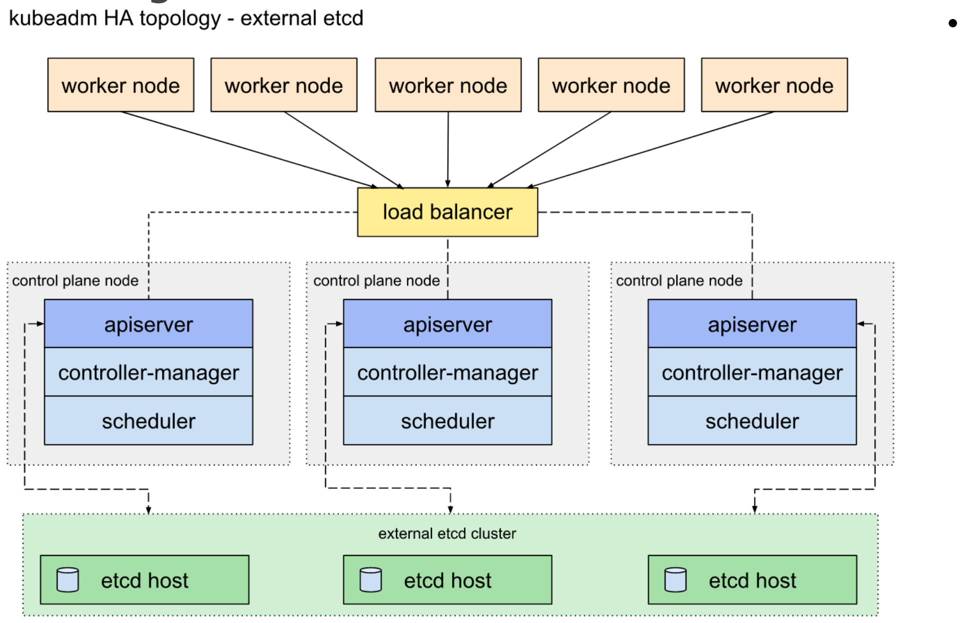
Data Store:

- **etcd** is a strongly consistent, distributed key-value **data store** used to persist a Kubernetes cluster's state. New data is written to the data store only by appending to it, data is never replaced in the data store. Obsolete data is compacted periodically to minimize the size of the data store.
- etcd is also used to store configuration details such as subnets, ConfigMaps, Secrets, etc.

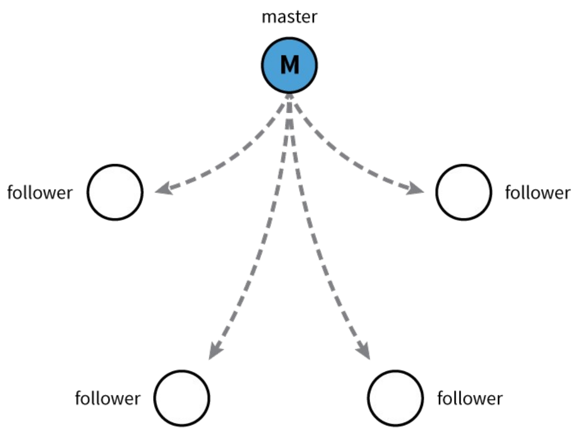
- Some Kubernetes cluster bootstrapping tools, such as **kubeadm**, by default, provision stacked etcd master nodes, where the data store runs alongside and shares resources with the other control plane components on the same master node.



- For data store isolation from the control plane components, the bootstrapping process can be configured for an external etcd topology, where the data store is provisioned on a dedicated separate host, thus reducing the chances of an etcd failure.



- Both stacked and external etcd configurations support HA configurations. etcd is based on the [Raft Consensus Algorithm](#) which allows a collection of machines to work as a coherent group that can survive the failures of some of its members. At any given time, one of the nodes in the group will be the master, and the rest of them will be the followers. etcd gracefully handles master elections and can tolerate node failure, including master node failures. Any node can be treated as a master.


master
M
follower
follower
follower
follower