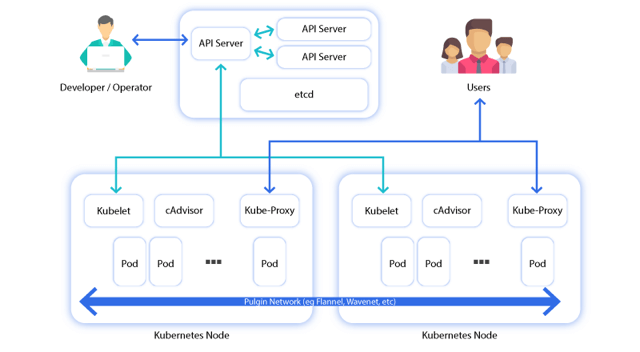
**What is Kubernetes**

* It is a system for running and coordinating containerized applications across a cluster of machines. A platform designed to completely manage the life cycle of containerized applications and services using methods that provide predictability, scalability and high availability. Open sourced in Jun 2014.
* K8s can be thought of as:

1. A container platform that manages clusters of containers
2. A Micro-services platform
3. A portable cloud platform and lot more

* K8s is organized around Master node and Worker nodes
  + K8s Master is the primary control unit that manages workload and communication across the system. Its components are:
    - Etcd storage (key-value data store for config data, accessed by all nodes of cluster)
    - API-server (central management entity that receives REST requests for modifications)
    - Scheduler (schedule the pods on various nodes based on resource utilization)
    - Controller manager (runs a number of distinct controller process in the background)



* + Worker nod, also known as minion node, contains information to manage networking between containers. Its components are
    - Kubelet (ensures all containers in the node are running and in a healthy state)
    - Container (lowest level of microservice, placed inside pods)
    - Kube Proxy (acts as a network proxy and load balancer)
    - cAdvisor (responsible for monitoring and gathering data about resource usage on a node)

**Master and slave**

* Master is the brain of k8s cluster. Here we have a core API server that maintains RESTful web services for querying and defining our desired cluster and workload state.
* Master includes a scheduler that works with the API server
* Public cloud (Anyone can use it)

**Idea behind clustered deployment using Kubernetes**

* Configuration information (that differs in respect to hosting environments e.g. QA/UAT/PROD) should be moved away from application code. Docker and Kubernetes are based on this paradigm
* Docker swarm is a cluster amangement software like Kubernetes
* Clustered environment ensures that a given app or service can be load balanced across multiple replicas and any individual app container should be able to fail with minimal or no disruption of service for the client
* To enable this horizontal, redundant scaling, applications must be designed in a stateless fashion. This means that they respond to client requests without storing persistent client and application data locally, and at any point in time if the running app container is destroyed or restarted, critical data is not lost.
* Stateless containers enable maximum portability and full use of available cloud resources, allowing the Kubernetes scheduler to quickly scale your app up and down and launch Pods wherever resources are available.
* In the Kubernetes model, the cluster control plane can be relied on to repair a broken application or service. It does this by checking the health of application Pods, and restarting or rescheduling unhealthy or unresponsive containers.
* To properly communicate application health to the Kubernetes control plane, you should implement custom application health checks that indicate when an application is both running and ready to receive traffic.
* Readiness probe – health check that lets k8s know when your application is ready to receive traffic
* Liveness probe – heath check that lets k8s know when your application is healthy and running
* The Kubelet Node agent can perform these probes on running Pods using 3 different methods:
  + HTTP:

The Kubelet probe performs an HTTP GET request against an endpoint (like /health), and succeeds if the response status is between 200 and 399

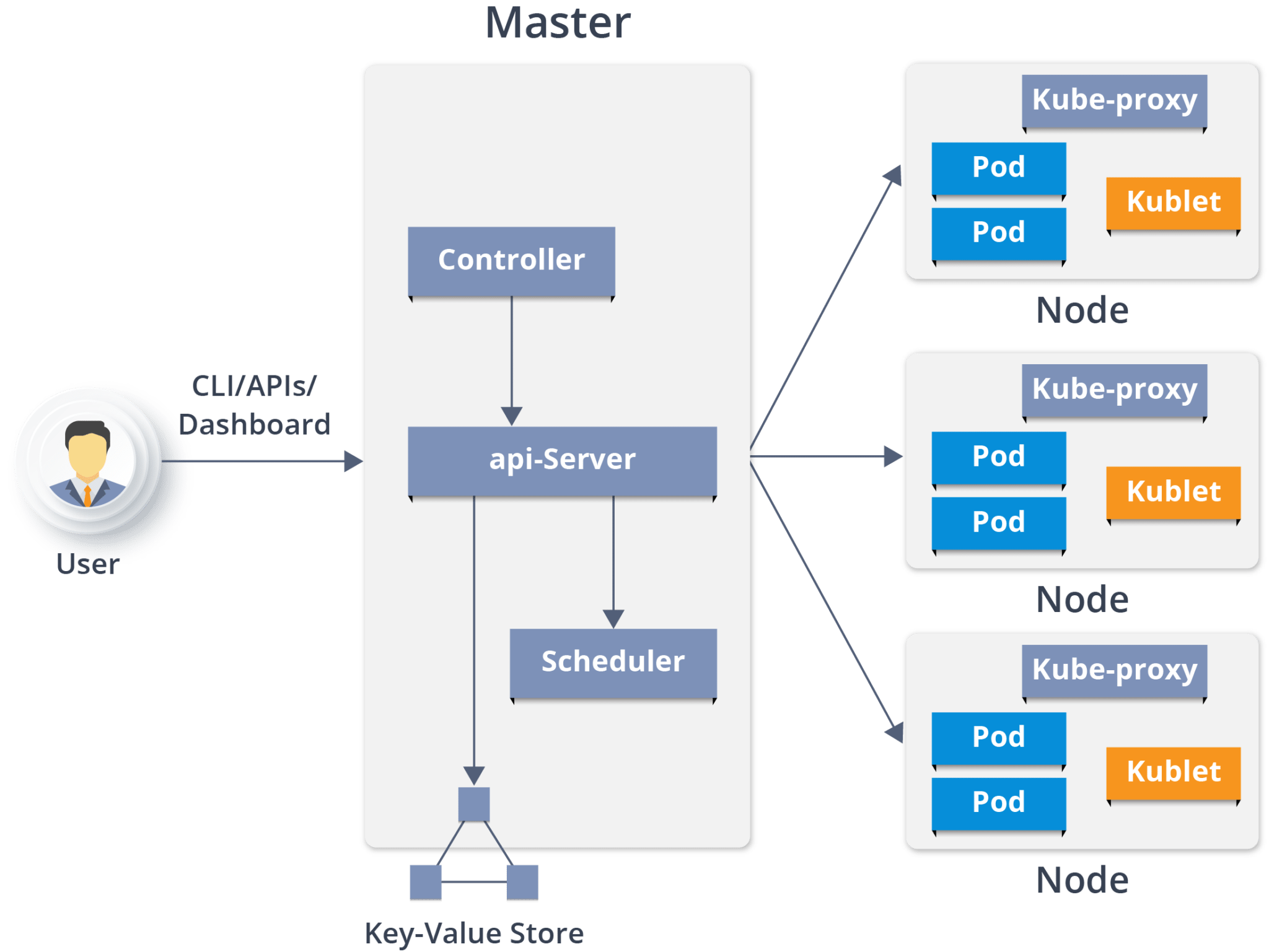
* + Container Command:

The Kubelet probe executes a command inside of the running container. If the exit code is 0, then the probe succeeds

* + TCP:
  + The Kubelet probe attempts to connect to your container on a specified port. If it can establish a TCP connection, then the probe succeeds.

**Kubernetes Architecture**

* Kubernetes architecture has the following main components:
  + Master nodes
  + Worker/Slave nodes
  + Distributed key-value store (etcd)



* Master node is the entry point for all administrative tasks which is responsible for managing the Kubernetes cluster
* There can be more than one master node in a cluster to check for fault tolerance

**API SERVER:**

* Performs all administrative tasks of K8s cluster
* Requests are sent to it as REST commands. It validates and processes it
* Store the state of cluster in key-value store

**SCHEDULER:**

* Schedules tasks to slave nodes. Stores resource usage information for each slave node
* Schedules the work in form of pods and services

**CONTROLLER:**

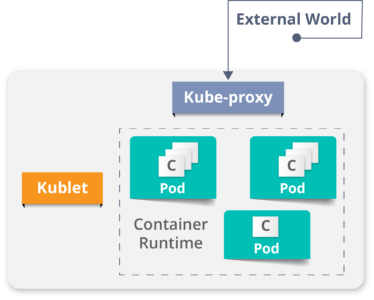
* A daemon that regulates k8s cluster which manages the different non-terminating control loops
* Performs lifecycle functions such as namespace creation and lifecycle, terminated-pod garbage collection etc.
* Watches the desired state of the objects, manages and watches their current state through API server. If the current state does not match with the desired state then the control loop takes corrective steps to make sure that they match

**ETCD:**

* A distributed key-value store that stores the cluster state
* Written in Go programming language
* Generally it is a part of the master node. But it can also be configured externally

**Worker node (formerly minions)**

* It is a physical server or you can say a VM which runs the applications using Pods (**a pod scheduling unit**) which is controlled by the master node. On a physical server (worker/slave node), pods are scheduled. For accessing the applications from the external world, we connect to nodes.



**CONTAINER RUNTIME:**

* To run and manage a container’s lifecycle on the worker node

**KUBELET:**

* It is an agent which communicates with the Master node and executes on nodes or the worker nodes.
* It gets the Pod specifications through the API server and executes the containers associated with the Pod and ensures that the containers described in those Pod are running and healthy.

**KUBE-PROXY:**

* Runs on each node to deal with individual host sub-netting and ensure that the services are available to external parties.
* Serves as a network proxy and a load balancer for a service on a single worker node and manages the network routing for TCP and UDP packets.
* It is the network proxy which runs on each worker node and listens to the API server for each Service endpoint creation/deletion.
* For each Service endpoint, kube-proxy sets up the routes so that it can reach to it.

**Container concepts**

At the core of container technology are

* Control groups (cgroups)
* Namespaces
* Additionally, Docker uses union filesystems for added benefits to the container development process.

**Docker:** A tool to create an image (Dockerimage) of an application and the dependencies needed to run it by using a configuration file (Dockerfile). A dockerfile has all the instructions on how to build the final image for deployment and distribution. The image is then used by k8s for deployment.

**Cgroup:** Restricts resources a process can consume (e.g. CPU, Memory, Disk IO etc.). It works by allowing the host to share and also limit the resources each process or container can consume. This allows several containers to share CPU and memory while staying within the predefined constraints.

**Namespace:** Change a process’s view of the system (e.g. network interfaces, PIDs, users, mount etc.) It offers another form of isolation for process interaction within operating systems. Container processes are limited to see only what is in the same namespace.

**Union filesystems:** Are also a key advantage of using Docker containers. Containers run from an image. Much like an image in the VM or cloud world, it represents state at a particular point of time. Container images snapshot the filesystem but tend to be much smaller than a VM.

Deployment of containers are based on operating-system-level virtualization rather than hardware virtualization. These containers are isolated from each other and from the host. They have their own filesystems, they cannot see each other’s processes and their computational resource usage can be bounded. They are portable across clouds and OS distributions.

**Points to remember**

* Continuous integration is a development practice that requires developers to integrate code to a shared repository several times a day. The result is that updates and bug fixes happen much faster and the overall quality improves
* Using Docker ensures that OS, packages and application versions are the same across development, staging and production environments
* Micro-service is an architectural style – an approach to develop a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms. These services are built around business capabilities and independently deployable by fully automated deployment machinery.
* Kubernetes (k8s) is a management and orchestration tool for containerized application deployment.
* K8s operates at container level rather than at hardware level. If an application can run in a container, it should run great on k8s
* K8s control script is known as ‘kubectl’. Most administrative interactions are done via this script
* The ideas of actual state and desired state is the key of how k8s manages the cluster and its workloads. All the pieces of k8s are constantly working to monitor the current actual state and synchronize it with the desired state defined by the administrators via the API server or kubectl script. There will be times when these states do not match up, but the system is always working to reconcile the two
* By default, Kubernetes uses the ‘gce’ provider for Google cloud. We can override this default by setting the KUBERNETES\_PROVIDER environment variable. Some permissible values are listed below:
  + gce (Google Compute Engine – public cloud)
  + gke (Google Conatiner Engine – public cloud)
  + aws (Amazon Web Services – public cloud)
  + azure (Microsoft Azure – public cloud)