Kubernetes Recap

Credit: Rich Wellum (Genesis Cloud)

Speaker: Cao Xuan Hoang

Staff Software Engineer - Genesis Cloud

Email: hoang@genesiscloud.com

Vietkubers 1st meetup - 2018/01/05

Genesis Cloud

Goals

- Provide a comparison between Virtualization and Containerization, where is the industry going
- Provide info on docker, what docker does and doesn't do, adoption rates
- Provide info on Kubernetes, what it is, why it's needed, why it's a red hot technology
- Workshops to create a docker image and a kubernetes cluster (TBD)

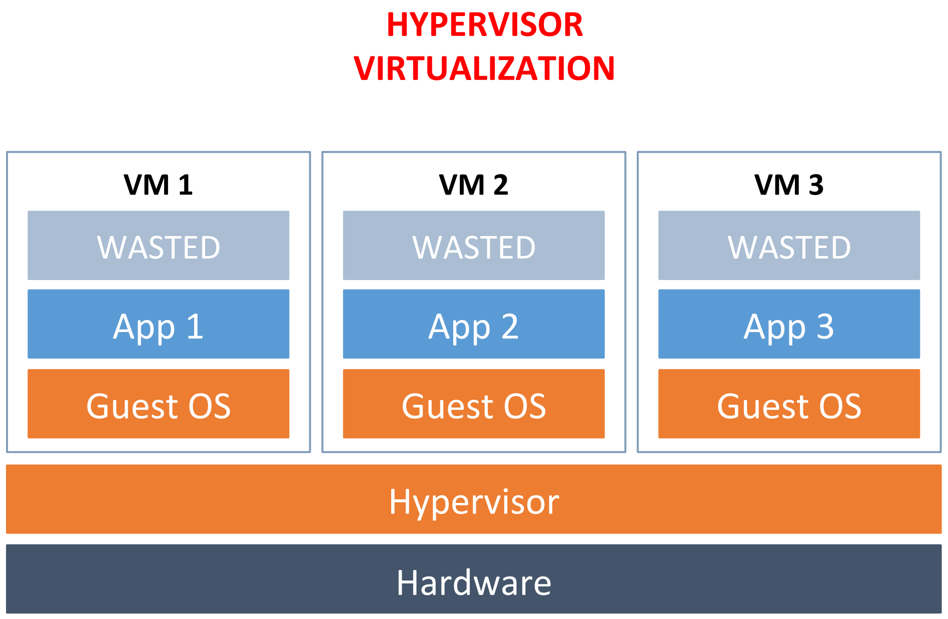
Genesis Cloud

Old School / New School

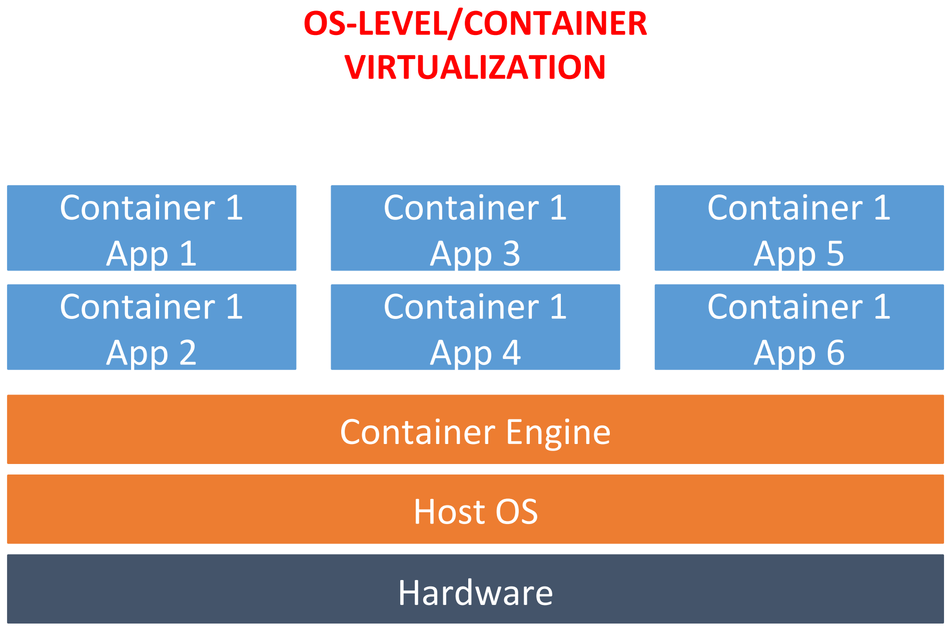
Comparing Virtualization and Containerization

Genesis Cloud

A Refresher: Differences between Virtualization and Containerization



VS



- Emulation of a computer system
- Operating system and code together packaged together
- Hypervisor provides a layer of emulation between OS and Hardware
- The guest virtual machines run on emulated servers, and that creates a lot of overhead

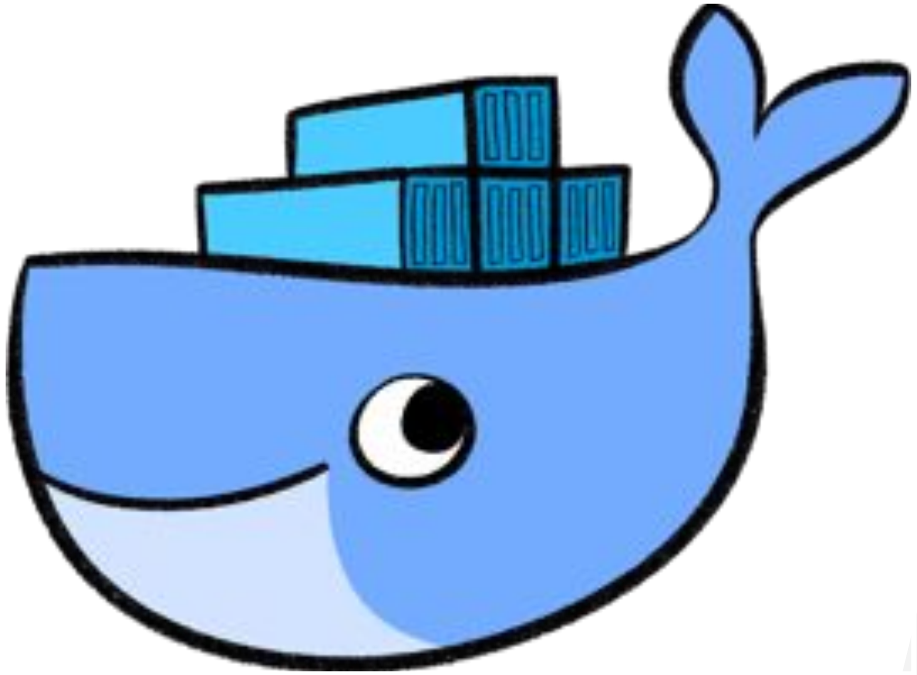
- Code and its dependencies packed into a container that can then run independently.
- Lots of them on a single host operating system
- Easier for developers to know that their software will run, no matter where it is deployed
- They also enable what's often called "microservices". Allowing development independence.

Genesis Cloud

Docker the good, bad and ugly

What docker is, what it provides and what it doesn't do that you probably think it does.

[docker white paper]()



Genesis Cloud

Docker - What is it?

- Before container technologies, deploying an application usually took quite a long time.
- The deployment had to be done manually, which cost the company time and resources.
- When container technologies came more popular with Docker and Kubernetes, the whole process became more streamlined and standardized.
- The container technologies can be used to automate the deployment process quite effortlessly and therefore the value of a well configured container platform is intangible.
- **Docker is a tool to create an image of an application and the dependencies needed to run it. The image can then later be used on a containerization platform such as Kubernetes.**

Genesis Cloud

Docker Images - Dockerfile?

- Docker is used to create a Docker (Container) Image of the application by using a Dockerfile, which is a bundle of an application along with it's runtime and dependencies.
- A Dockerfile has all the instructions on how to build the final image for deployment and distribution.
- The Docker/Container Images that are made are reusable perpetually, and are an isolated executable environment that the container image runs in.
- The benefits of Docker are for example the reusability of once created resources and the fast setup of the target environment, whether it is for testing or production purposes.

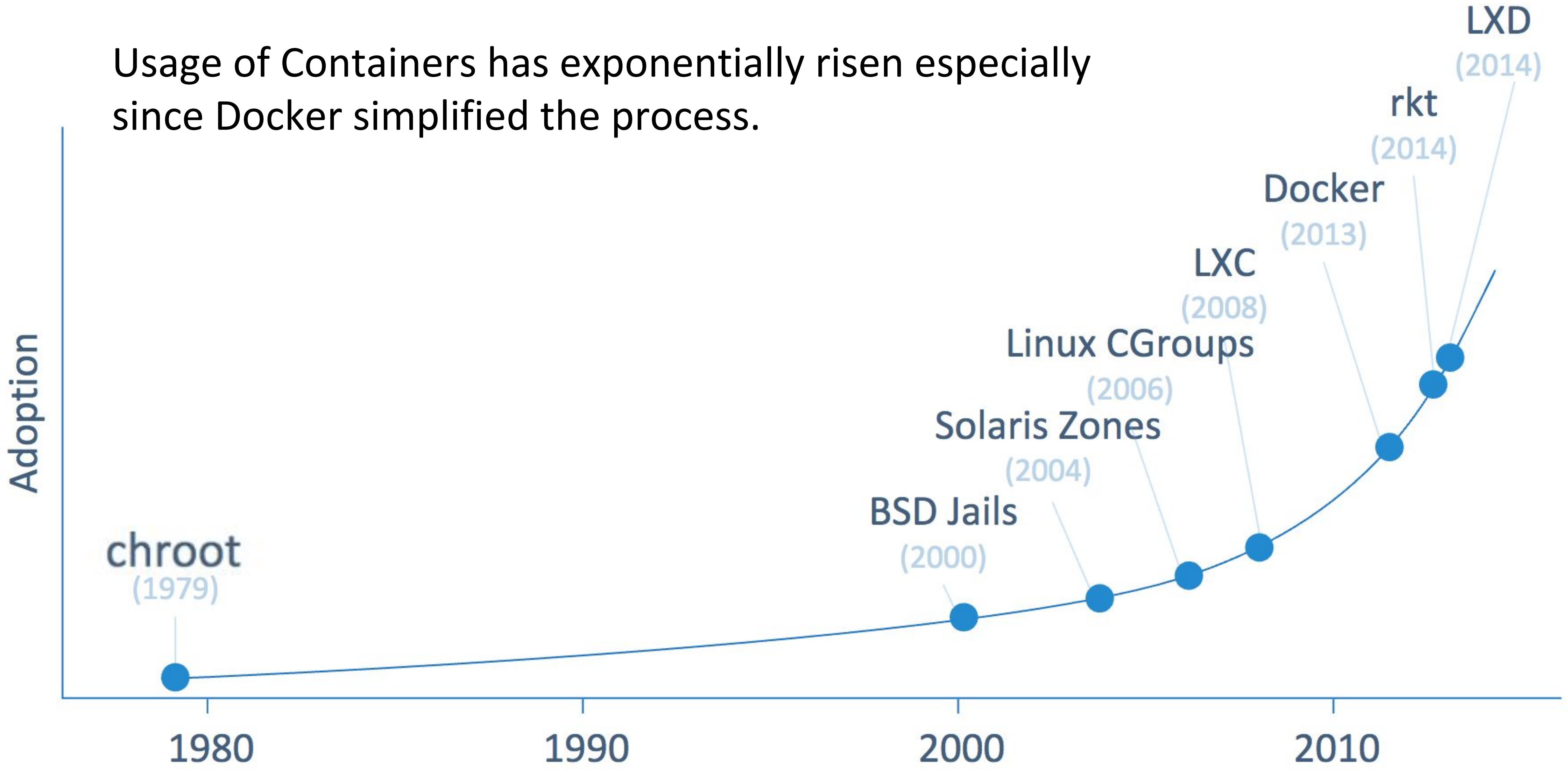
Docker - What is it?

- Docker images are layered. When building an image Docker creates a new intermediate container for each instruction described in the Dockerfile.
- When the commands are chained together to form a coherent line of build instructions, it reduces the build time and resources the Dockerfile might use.
- The field of Development and Operations (DevOps) benefit greatly from containerization in the form of automating the deployment.
- There are several types of software to create a continuous integration and deployment pipeline (CI/CD). This enables the DevOps team to deploy an application seamlessly to the targeted environment

Genesis Cloud

Docker and the Growth of Containerization

Usage of Containers has exponentially risen especially since Docker simplified the process.



Genesis Cloud

Docker - the good

Source: [why-docker?]

Docker is not very old, release 1.0 release in May of 2014. Why has it grown so quickly?

- Docker Portability.
  - Allows applications to be isolated into containers with instructions to be easily ported from machine to machine.
  - Docker and Microservice architecture goes hand-in-hand.

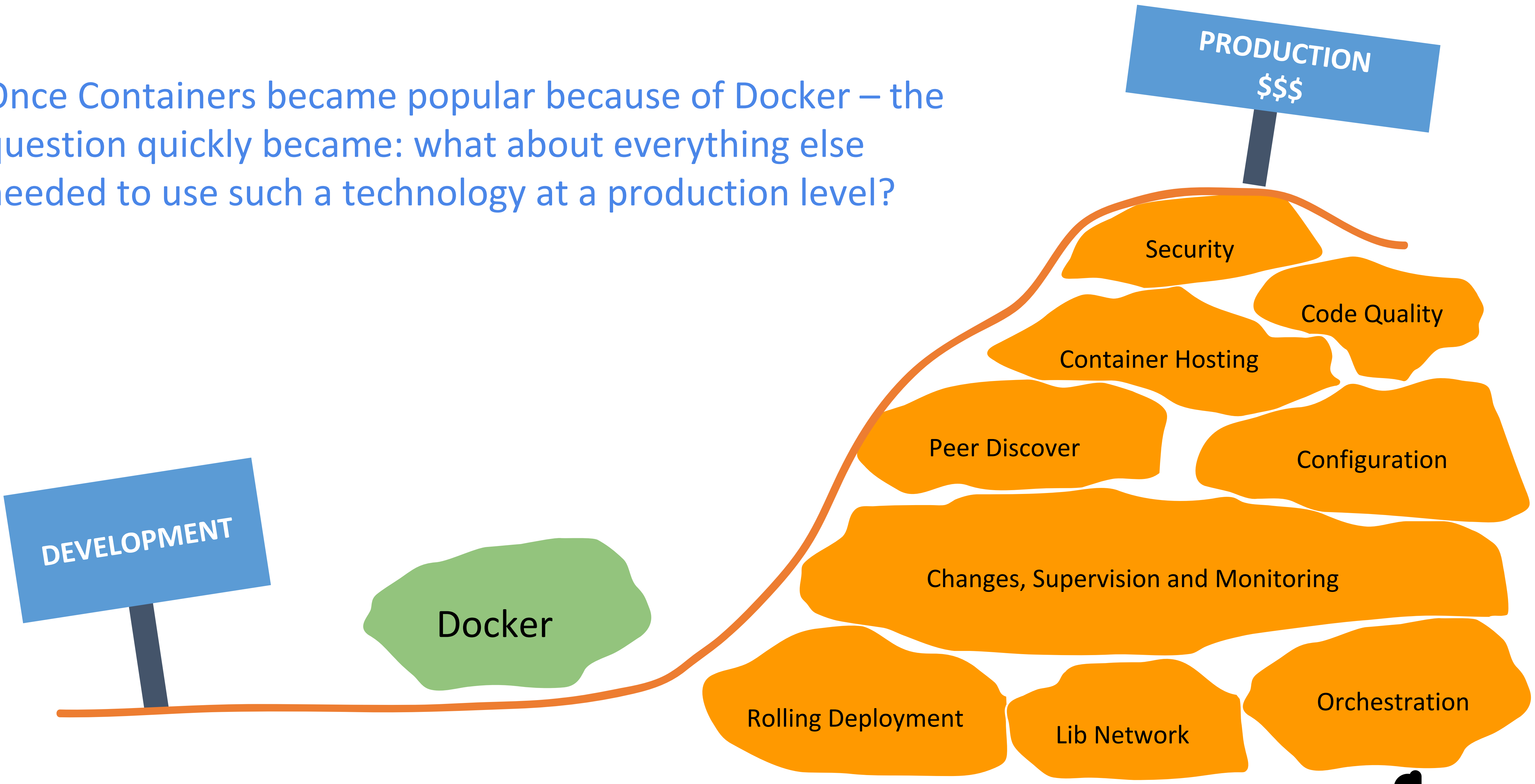
- Resource efficiency.
  - If you have 30 Docker containers that you want to run, you can run them all on a single server.
  - To run 30 virtual machines, you’ve got to boot 30 operating systems with at least minimum resource requirements available before factoring the hypervisor for them to run on with the base OS.
  - Just assuming you’re going to go with a minimum 256M VM you’d be looking at 7.5G of RAM with 30 different OS kernels managing resources.
  - With Docker you could allocate a chunk of RAM to one server and have a single OS managing those competing resources… and you could do all of that on the base operating system without a costly hypervisor needing to be involved at all.

- Tight integration with Linux Kernel.
  - It allows for significant efficiency at a low level

Genesis Cloud

Docker - the bad: what doesn't Docker do?

Once Containers became popular because of Docker – the question quickly became: what about everything else needed to use such a technology at a production level?



Genesis Cloud

Microservices

A brief guide...

Genesis Cloud

What are MicroServices

- Microservices are services that are generally containerized and are easily distributable through some form of a network to be easily replaceable parts.
- These services are defined by a specification where they do a single task, expose some endpoint or API and are composable with other microservices.
- There is a need for these services to talk to each other, and to external services and to do this they use some form of a meshing network.
- These networks right now are done through container networks, such as the Docker Overlay Network, the Kubernetes "POD" system, linkerd, serf, and a multitude of other systems like istio.

Advantages

- Isolation of technologies
- Quicker ramp-up time for developers
- Quicker, simpler Upgrades
- Roll-out acceleration
- Enables Containerization

Genesis Cloud

How to Productize: Enter Kubernetes...

The red hot Container Orchestration Engine (COE) technology that everyone is talking about.

- The History of Kubernetes
- Official Boot Camp
- Magic Sandbox

Genesis Cloud

First - what is a COE?

- Container Orchestrators group hosts together to form a cluster, and help fulfill the requirements of production level applications.
- The requirements of production level applications are:
  - Fault tolerance
  - Optimal resource usage
  - Auto Discovery
  - Communication with other applications
  - High Accessibility
  - Zero Downtime
  - Update
  - Rollback
  - ….

- The benefits of using a COE are:
  - Groups hosts to form clusters
  - Schedule containers to run on different hosts
  - facilitate communications between hosts
  - Bind containers and storage
  - Bind containers of similar type to a higher-level construct, like services for better management
  - Monitor and optimize resource usage
  - Allow secure access to applications running inside containers

Genesis Cloud

Kubernetes - what it is

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

The open source project is hosted by the Cloud Native Computing Foundation (CNCF).

Kubernetes has a number of features. It can be thought of as:

- a container platform
- a microservices platform
- a portable cloud platform and a lot more.

You interact with Kubernetes by declaring a state in which you want your system to be in, and various K8s mechanisms work towards reaching and maintaining that state.

- ❖ Kubernetes focuses on orchestration of application containers. While it abstracts and supports various hardware environments, you shouldn’t think of it as a traditional PaaS (Platform as a Service).
- ❖ Kubernetes operates at the Container level, rather than at the hardware level.
- ❖ As long as you can package your app in containers, you should be able to deploy it on Kubernetes.
- ❖ Kubernetes provides some generally applicable features common to PaaS offerings, such as deployment, scaling, load balancing, logging, and monitoring.
- ❖ However, Kubernetes is not monolithic, and these default solutions are optional and pluggable.
- ❖ Kubernetes provides the building blocks for building developer platforms, but preserves user choice and flexibility where it is important.

Kubernetes - what features does it provide

The kubernetes.io website lists the following features of Kubernetes:

- **Automatic binpacking -** Automatically places containers based on their resource requirements and other constraints, while not sacrificing availability. Mix critical and best-effort workloads in order to drive up utilization and save even more resources.
- **Self-healing -** Restarts containers that fail, replaces and reschedules containers when nodes die, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
- **Horizontal scaling -** Scale your application up and down with a simple command, with a UI, or automatically based on CPU usage.
- **Service discovery and load balancing -** No need to modify your application to use an unfamiliar service discovery mechanism. Kubernetes gives containers their own IP addresses and a single DNS name for a set of containers, and can load-balance across them.
- **Automated rollouts and rollbacks -**Kubernetes progressively rolls out changes to your application or its configuration, while monitoring application health to ensure it doesn't kill all your instances at the same time
- **Secret and configuration management -** Deploy and update secrets and application configuration without rebuilding your image and without exposing secrets in your stack configuration.
- **Storage orchestration -** Automatically mount the storage system of your choice, whether from local storage, a public cloud provider such as GCP or AWS, or a network storage system such as NFS, iSCSI, Gluster, Ceph, Cinder, or Flocker.
- **Batch execution -** In addition to services, Kubernetes can manage your batch and CI workloads, replacing containers that fail, if desired.

Genesis Cloud

Kubernetes - what it is not

- Kubernetes does not limit the types of applications supported. If an application can run in a container, it should run great on Kubernetes.

- It does not deploy source code and does not build your application. Continuous Integration, Delivery, and Deployment (CI/CD) workflows are determined by you.

- It does not provide application-level services, such as middleware (e.g., message buses), data-processing frameworks (for example, Spark), databases (e.g., mysql), caches, nor cluster storage systems (e.g., Ceph)

- It does not dictate logging, monitoring, or alerting solutions. It provides some integrations as proof of concept, and mechanisms to collect and export metrics.

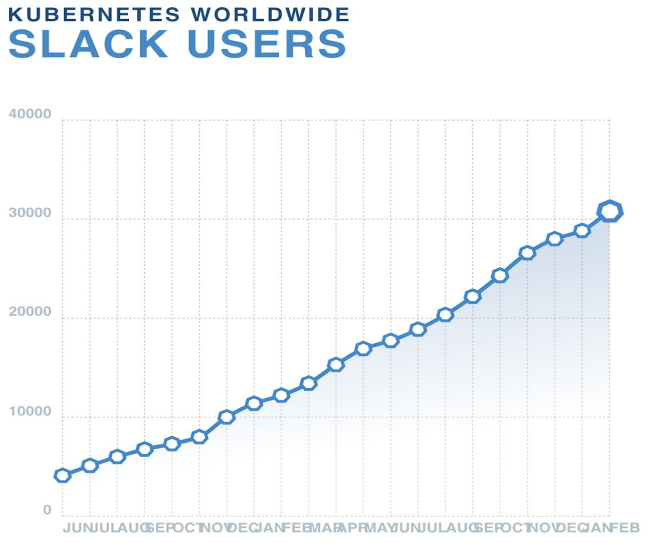
- It does not provide nor mandate a configuration language/system (e.g., jsonnet). It provides a declarative API that may be targeted by arbitrary forms of declarative specifications.

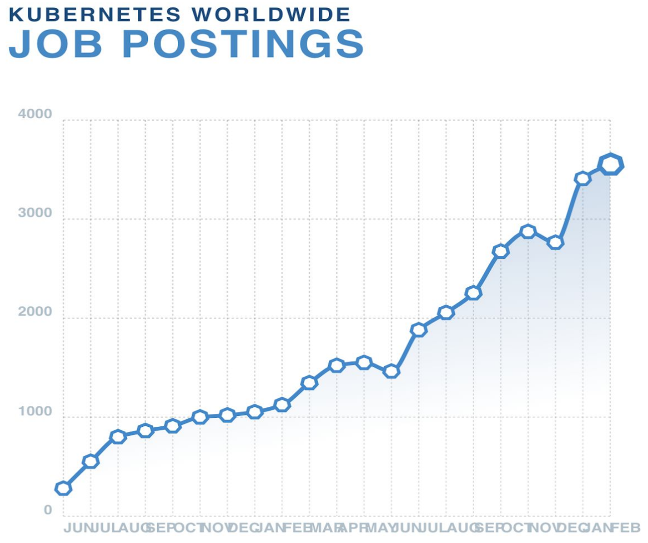
- It does not provide nor adopt any comprehensive machine configuration, maintenance, management, or self-healing systems.

Genesis Cloud

Kubernetes - where does it stand in OpenSource?

- Kubernetes is the open source standard for managing application containers from local to planetary scale. Originally developed by Google using 15 years of R&D, Kubernetes now flourishes in the open source community and has become the foundation for production cloud native solutions.
- Kubernetes is the largest and fastest growing open source software solution focused on democratizing distributed system patterns. In its first two years of general availability, more production users have committed to Kubernetes than any other comparable solution





Genesis Cloud

Who's Using Kubernetes Today?

Kubernetes is the largest and fastest growing open source software solution focused on democratizing distributed system patterns. In its first year of general availability, more production users have committed to Kubernetes than any other comparable solution.

Walmart · CONCUR · VIACOM · buffer · Home Office

Arkena · Goldman Sachs · monzo · EVE ONLINE · Pearson

The New York Times · box · OpenAI · ticketmaster · SKY

COMCAST · Bloomberg · Pokémon GO · SOUNDCLOUD · unacast.

There are other COE’s

	kubernetes	MESOS	docker	CLOUDFOUNDRY
FOUNDATION	Cloud native computing foundation (part of linux foundation)	Apache	None	Cloud foundry foundation
SOFTWARE VENDORS (PRODUCT NAME)	Apprenda (Kismatic), CoreOS (Tectonic), Engine Yard (Deis), Red Hat (OpenShift)*, Mesosphere (DCOS)*, Rancher Labs (Rancher)* *Kubernetes is a component of a larger product ** Incomplete list	Mesosphere (DCOS)	Docker Inc (Swarm)	Pivotal (PCF), HPE (Helion)
PUBLIC CLOUD SERVICE PROVIDERS	Google Container Engine, Red Hat OpenShift, and many more currently in development.	A number of public cloud providers, including Azure, use Mesos as part of its underlying architecture.	Azure	IBM, Predix (GE's IoT PaaS)

Kubernetes Introduction

What is a kubernetes cluster, what are common terms, basic architecture etc.

Genesis Cloud

Kubernetes Introduction: Official Statement

Kubernetes (commonly stylized as **K8s**[3]) is an open-source container-orchestration system for automating deployment, scaling and management of containerized applications.[4] It was originally designed by Google and is now maintained by the Cloud Native Computing Foundation. It aims to provide a "platform for automating deployment, scaling, and operations of application containers across clusters of hosts".[3] It works with a range of container tools, including Docker.

Genesis Cloud

Kubernetes Introduction: Self Healing

- Kubernetes automatically replaces and reschedules containers from failed nodes.
- It also kills and restarts the containers which do not respond to health checks based on existing rules/policy

Kubernetes Introduction: Horizontal Scaling

- Kubernetes can automatically scale applications based on resource usage like CPU and memory.
- In some cases it also supports dynamic scaling based on customer metrics.

Kubernetes Introduction: Roll , Secrets

- Kubernetes can roll out and roll back new versions/configurations of an application without introducing any downtime.
- Kubernetes can manage secrets and configuration details for an application without re-building the respective images.
- With secrets we can share confidential information to our application without exposing it to the stack configuration.

Kubernetes Introduction: Storage orchestration

- Kubernetes and its plugins we can automatically mount local and external storage solutions to the containers in a seamless manner, based on Software Defined Storage (SDS)

Kubernetes Introduction: Service Discovery and Load balancing

- Kubernetes groups sets of containers and refers to them via a DNS name.
- This DNS name is also called a **Kubernetes Service**.
- Kubernetes can discover these services automatically and load-balance requests between containers of a given Service.

Kubernetes Introduction: Declarative

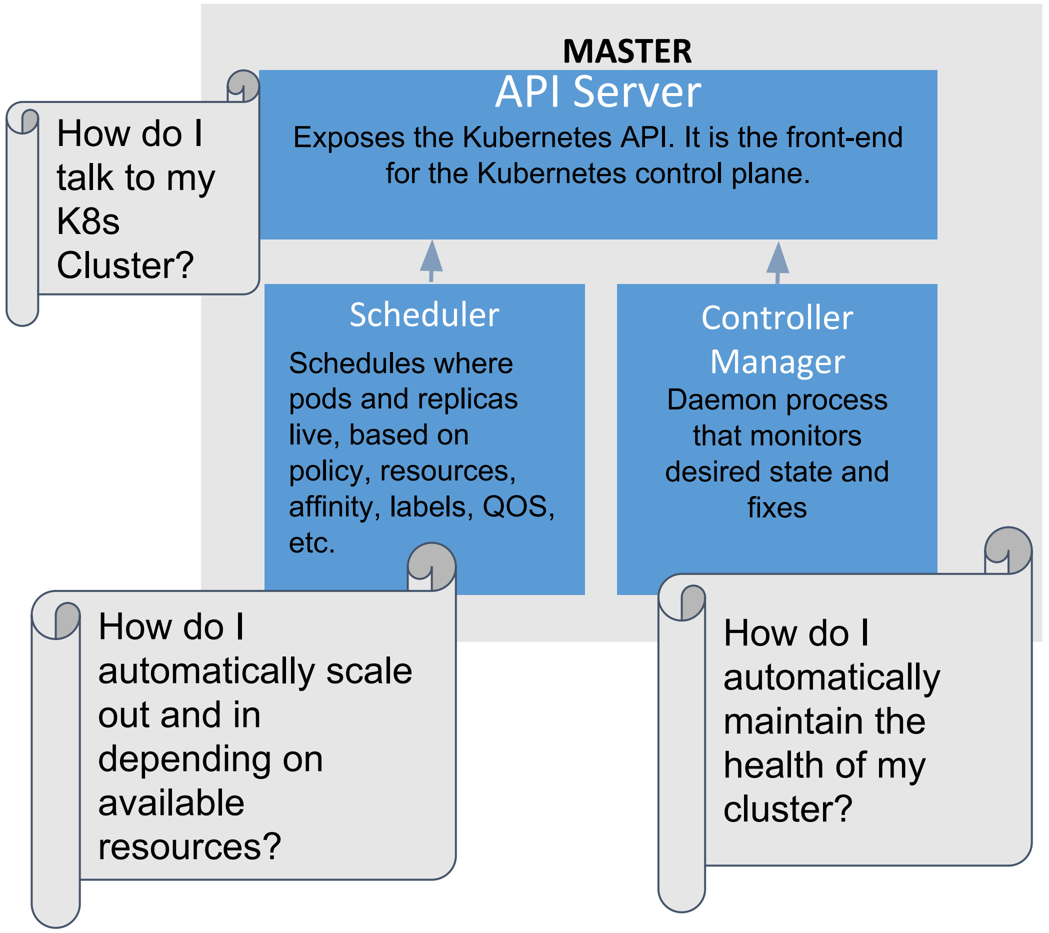
- To work with Kubernetes, you use ***Kubernetes API*** *objects* to describe your cluster's ***desired*** *state*: what applications or other workloads you want to run, what container images they use, the number of replicas, what network and disk resources you want to make available, and more. **It is value driven.**
  - You set your desired state by creating objects using the Kubernetes API, typically via the command-line interface, **kubectl**, which can be expressed in YAML.

- Once you've set your desired state, the ***Kubernetes Control Plane*** works to make the cluster's current state match the desired state.
  - To do so, Kubernetes performs a variety of tasks automatically – such as starting or restarting containers, scaling the number of replicas of a given application, and more.
  - The Kubernetes Control Plane consists of a collection of processes running on your cluster

- The Kubernetes control plane will accept and absorb new hardware (or VMs) into it's Cluster to scale up (or down)

- Replica sets (or **Deployments**) provide constant docker HA. Crucial containers are replicated in case of failure

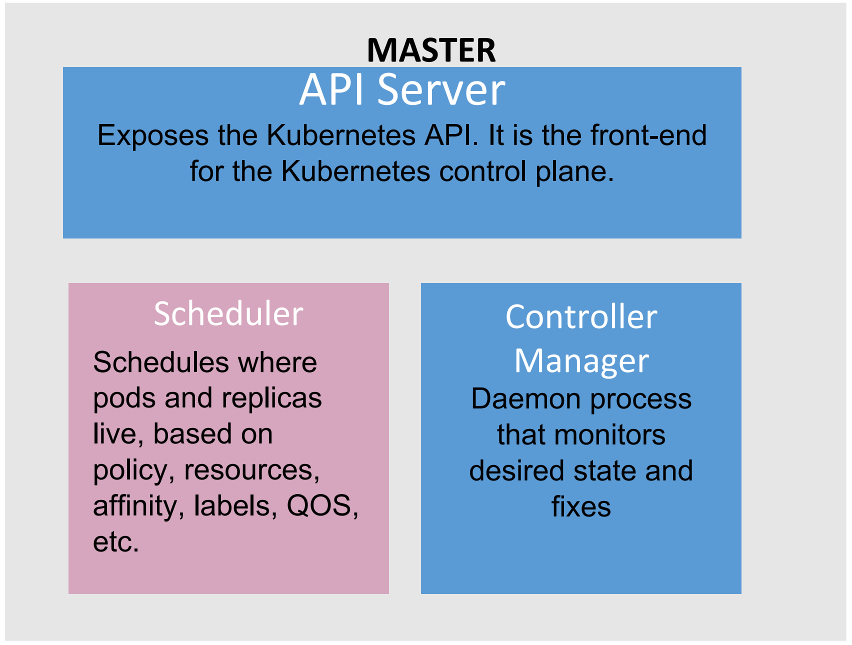
Kubernetes Introduction: Kubernetes Master Node(1)


MASTER
API Server
Exposes the Kubernetes API. It is the front-end for the Kubernetes control plane.
Scheduler
Schedules where pods and replicas live, based on policy, resources, affinity, labels, QOS, etc.
Controller Manager
Daemon process that monitors desired state and fixes
How do I talk to my K8s Cluster?
How do I automatically scale out and in depending on available resources?
How do I automatically maintain the health of my cluster?

- The Kubernetes master(s) is responsible for maintaining the desired state for your cluster.
- The entry point for all Administrative tasks.
- Communication via CLi, GUI (Dashboard) or via API's (Curl).
- The Kubernetes Master is a collection of three processes that run on a single node in your cluster, which is designated as the master node. Those processes are: kube-apiserver, kube-controller-manager and kube-scheduler.
- Multiple Master nodes are run in HA mode, one being the leader, the rest followers.

Genesis Cloud

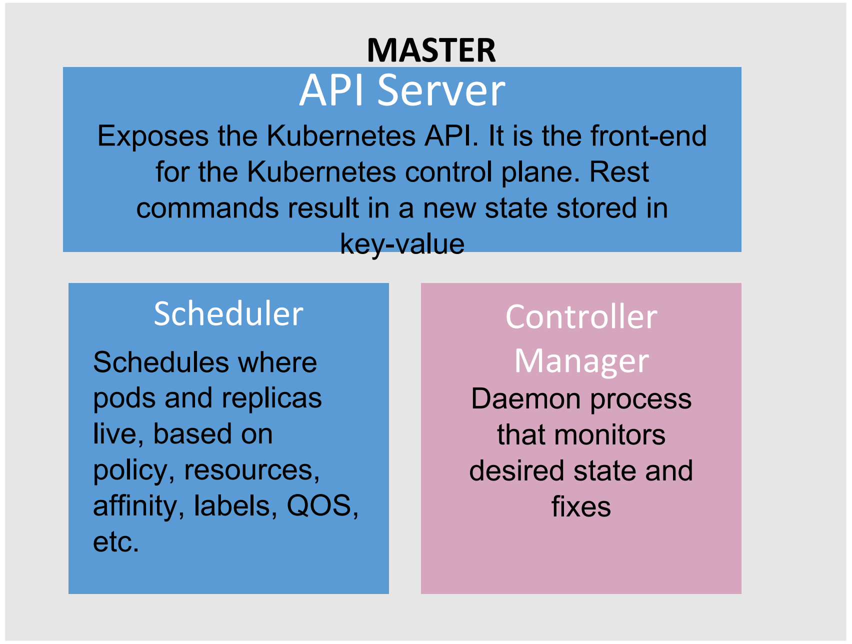
Kubernetes Introduction Master (2)


MASTER
API Server
Exposes the Kubernetes API. It is the front-end for the Kubernetes control plane.
Scheduler
Schedules where pods and replicas live, based on policy, resources, affinity, labels, QOS, etc.
Controller Manager
Daemon process that monitors desired state and fixes

- **kube-scheduler -** Schedules the work to different Worker Nodes. Work is scheduled in terms of Pods and Services.
- Has the resource usage info for each Worker Node and any user defined constraints.
- Factors taken into account for scheduling decisions include individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specification s, data locality, inter-workload interference and deadlines.

Genesis Cloud

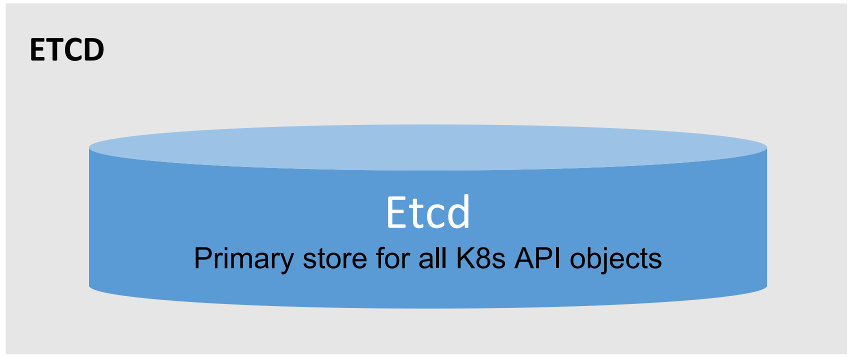
Kubernetes Introduction Master (3)


MASTER
API Server
Exposes the Kubernetes API. It is the front-end for the Kubernetes control plane. Rest commands result in a new state stored in key-value
Scheduler
Schedules where pods and replicas live, based on policy, resources, affinity, labels, QOS, etc.
Controller Manager
Daemon process that monitors desired state and fixes

- **kube-controller-manage**r - Component on the master that runs controllers.
- A control loop that watches the shared state of the cluster through the api server and makes changes attempting to move the current state towards the desired state.
- Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process.
- Controllers include:
  - Node Controller: Responsible for noticing and responding when nodes go down.
  - Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system
  - Endpoints Controller: Populates the Endpoints object (that is, joins Services & Pods).
  - Service Account & Token Controllers: Create default accounts and API access tokens for new namespaces.

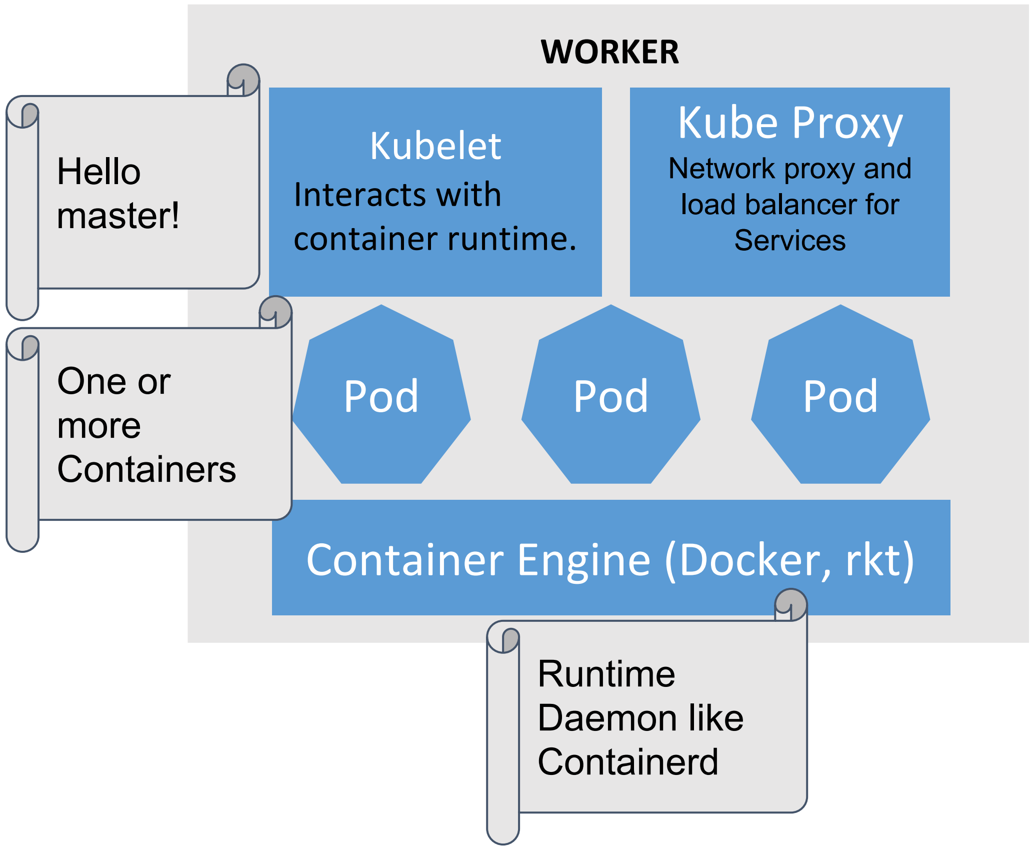
Genesis Cloud

Kubernetes Introduction Storage: etcd



- Consistent and highly-available key value store used as Kubernetes backing store for all cluster data.
- Used to manage the cluster state, and ALL master Nodes connect to it.
- Can be part of the Master Node or be configured externally.
- Also used to store configuration details such as subnets, ConfigMaps, Secrets etc.
- Always have a backup plan for etcd’s data for your Kubernetes cluster.

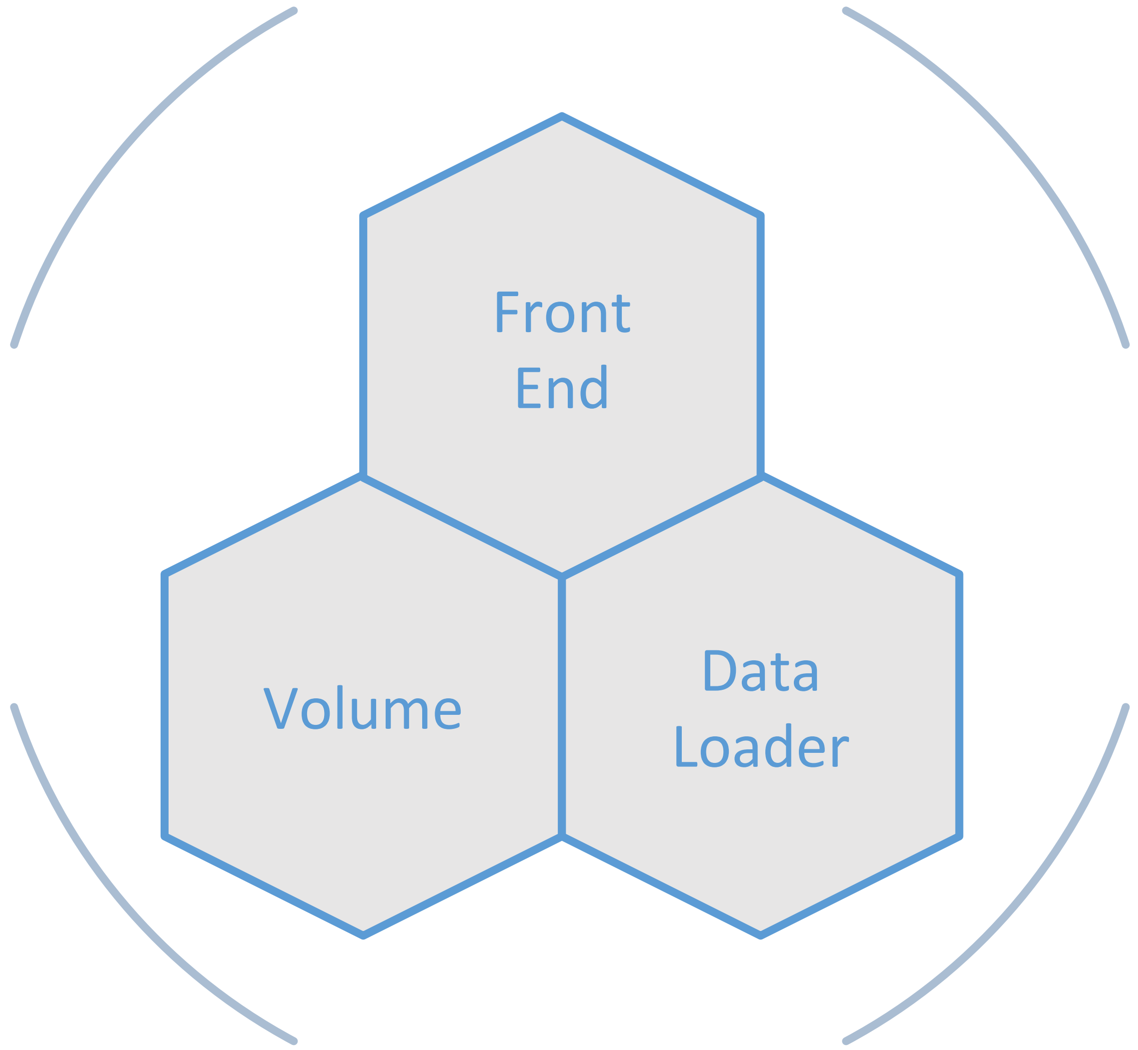
Kubernetes Introduction: Kubernetes Workers


WORKER
Hello master!
Kubelet
Interacts with container runtime.
Kube Proxy
Network proxy and load balancer for Services
One or more Containers
Pod
Pod
Pod
Container Engine (Docker, rkt)
Runtime Daemon like Containerd

- A Worker Node is a machine, VM, Physical Server etc., which runs applications using Pods and is controlled by the Master Node.
- Each individual Worker Node in your cluster runs:
  - kubelet, takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy.
  - kube-proxy, a network proxy which reflects Kubernetes networking services on each node.
  - Pods (one or more containers).
  - The container runtime is the software that is responsible for running containers. Kubernetes supports several runtimes: Docker, rkt, runc and any OCI runtime-spec implementation.
  - Addons

Genesis Cloud

Kubernetes Introduction: Pods

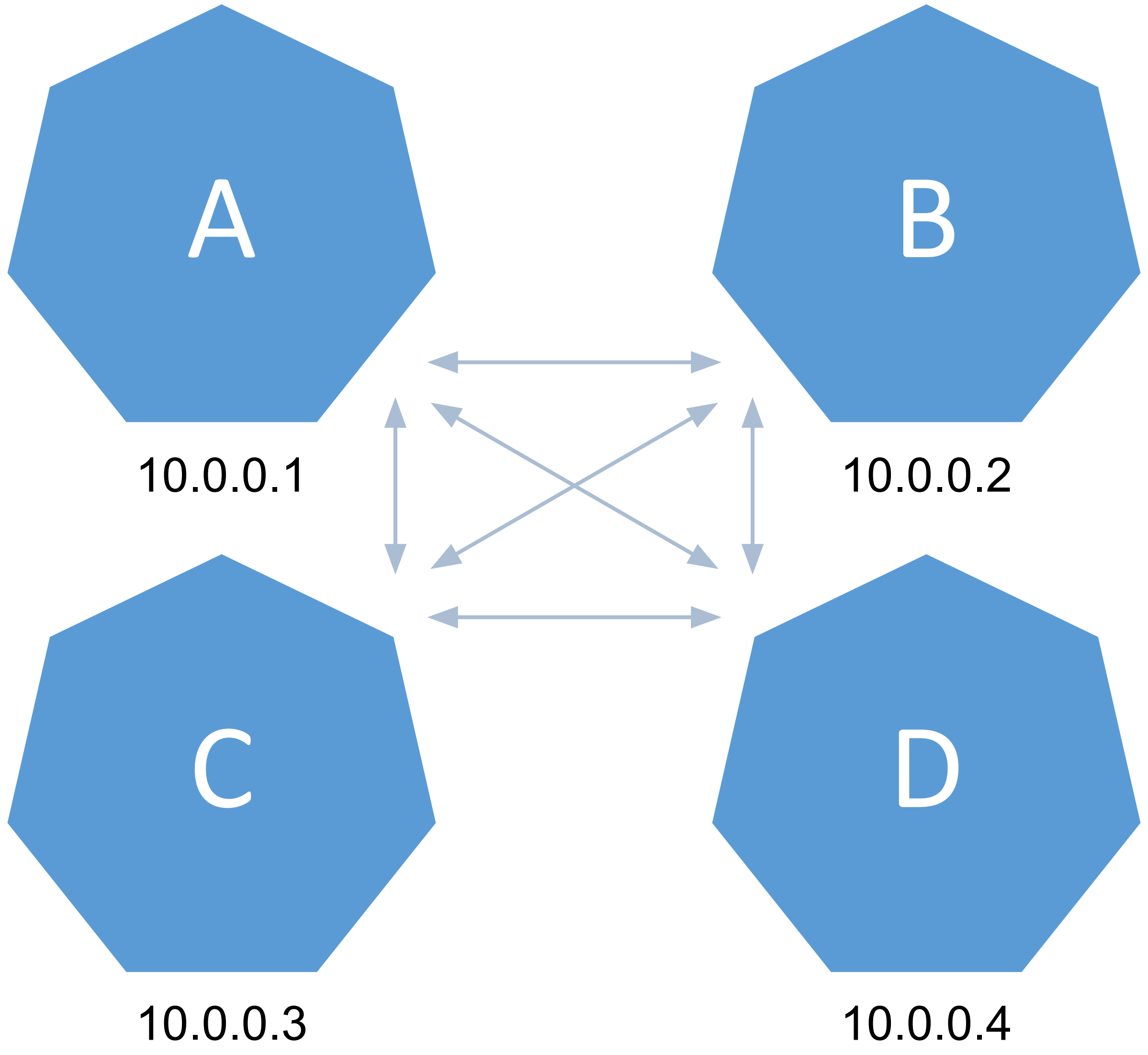


- A Pod is the scheduling unit in Kubernetes
- A logical collection if one or more containers which are always scheduled together
- Share Namespace
- Share Network

Kubernetes Introduction: Pod Networking

To do inter-pod Networking a Container Networking Infra (CNI) (SDN) applied.

Routable (layer 3)

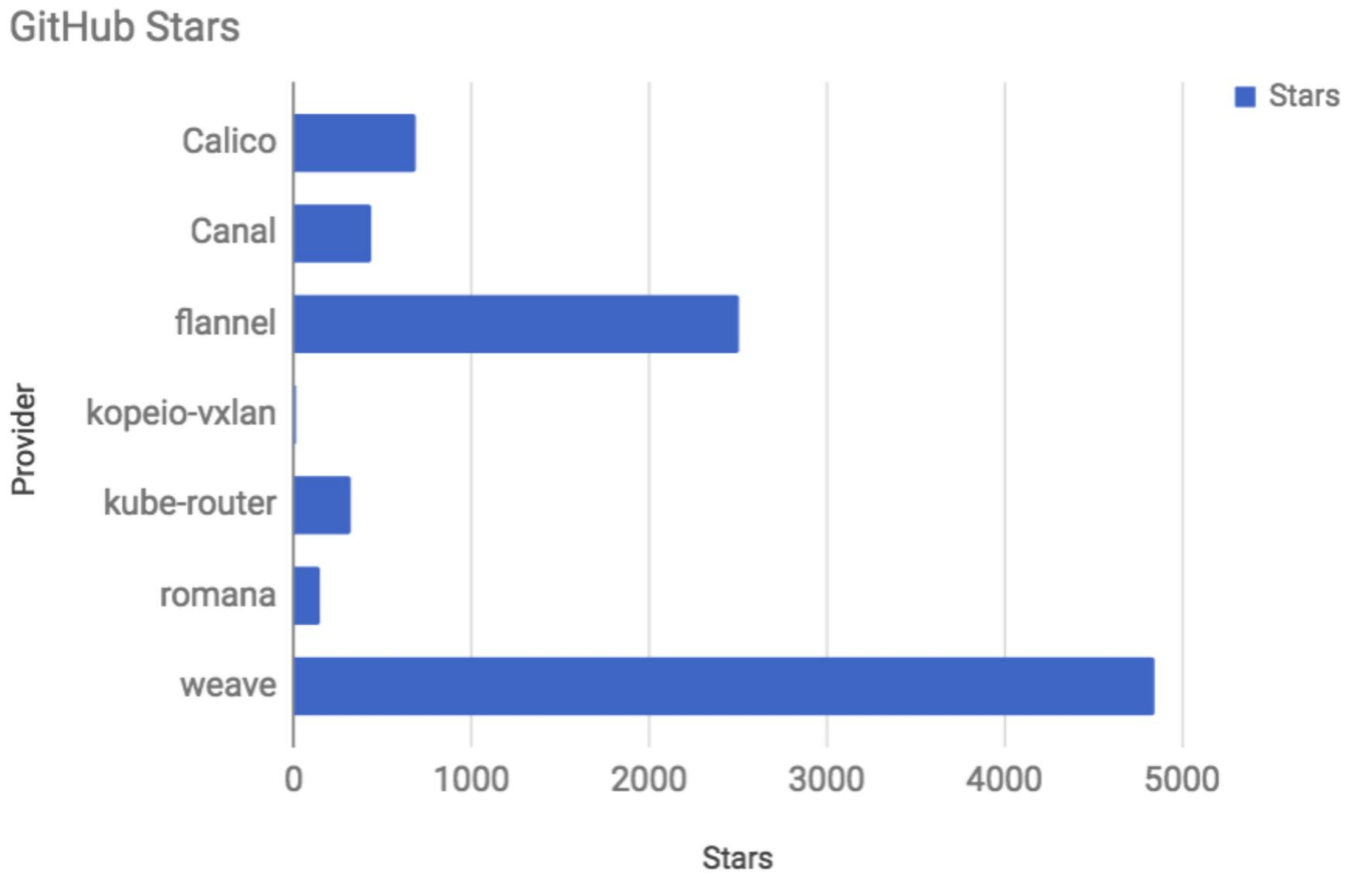


No NAT (internode)

[CNI Breakdown]

Genesis Cloud

Kubernetes Introduction: Pod Networking (CNI's)



Genesis Cloud

Kubernetes Introduction: Pod Networking

- A unique IP is assigned to each Pod.
- Containers in a Pod can communicate with each other.
- The Pod is able to communicate with other Pods in the cluster.
- If configured the application deployed inside a Pod is accessible from the external world (Ingress).
- The Container Runtime offloads the IP assignment to CNI, which connects to the underlying configured plugin. Once the IP address is given by the respective plugin the CNI forwards it back to the requested Container Runtime.
- No NAT, underlying physical infra is used to route.

Kubernetes Introduction: Kubelet

- The Kubelet is an agent running on each Worker Node and communicates with the Master Node.
- It receives Pod definitions via various means, but primarily through the API Server and runs the containers associated with the Pod.
- It also makes sure the containers which are part of the Pods are healthy at all times.
- It connects to the Container Runtimes (Docker/rkt) to run containers.

Kubernetes Introduction: Kube Proxy

- Kube-proxy is the network proxy which runs on each Worker Node and listens to the API Server for each Service endpoint creation/deletions.
- For each Service Endpoint, kube-proxy sets up the routes so that it can reach to it.

Kubernetes Introduction: Daemon Set

- A specific type of Pod running on ALL nodes at all times.
- Used to collect monitoring data from all nodes, run a storage daemon on all nodes etc

Kubernetes Introduction: RBAC

- Role-based Access Control is an authorization mechanism for managed permissions around kubernetes resources.
- Within a namespace arole is defined using a Role Object.
- For a Cluster-wide role we need to use the ClusterRole object.

Genesis Cloud

Kubernetes Introduction: Cluster Federation

- Used to manage multiple kubernetes clusters from a single control plane.
- We can sync resources across the clusters and have cross-cluster discovery.
- This allows us to do Deployments across regions and access them using a global DNS record.

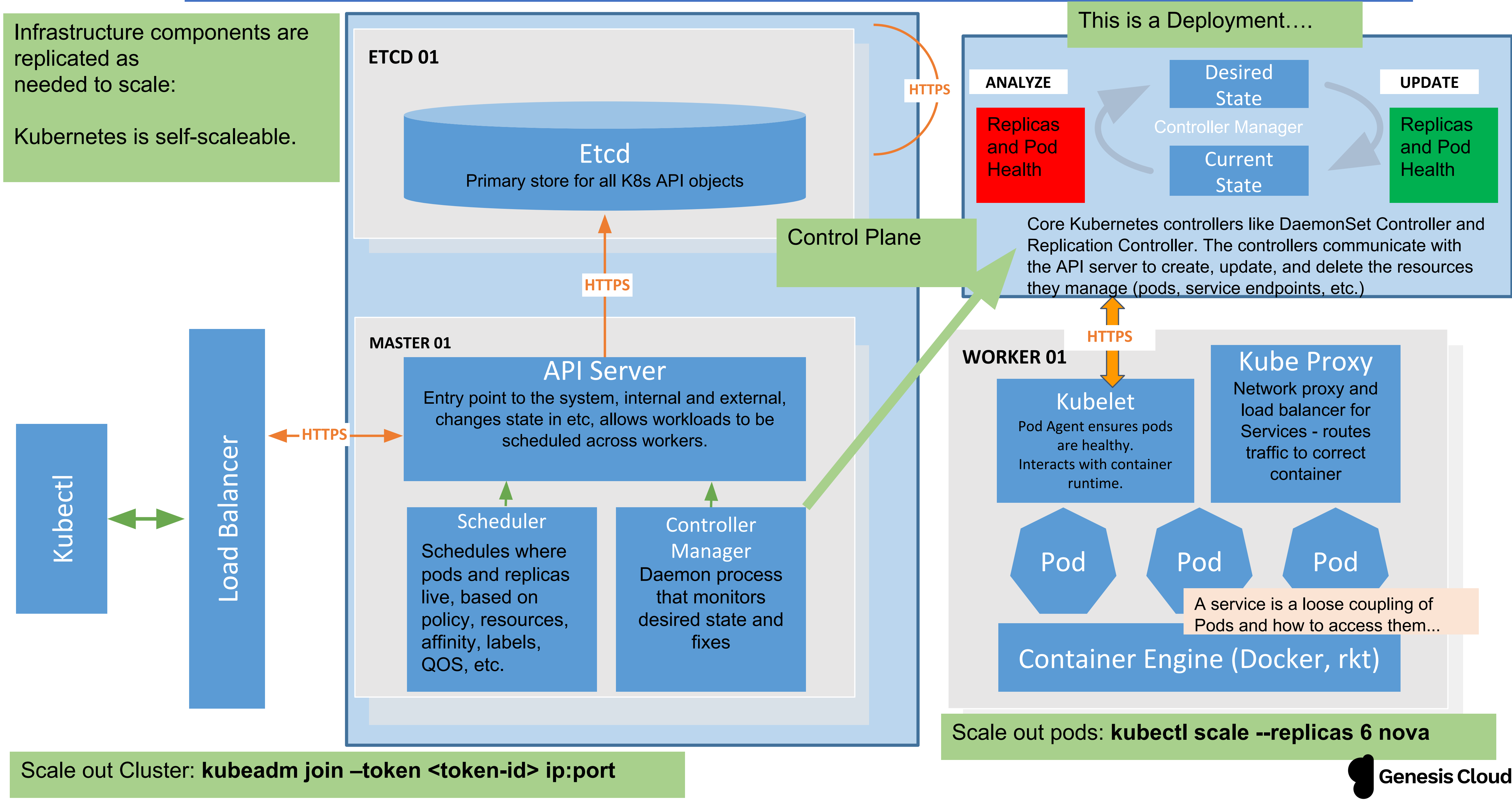
Kubernetes Introduction: Ingress

- A collection of rules that allow inbound connections to reach the cluster services
- Ingress configures a L7 HTTP load balancer for Services and provides:
  - TLS
  - named-based virtual hosting
  - path-based routing
  - custom rules
- Controlled by the Ingress Controller

Kubernetes Introduction: everything else

- Scratched the surface much?
- Suggest reading about:
  - labels
  - namespaces
  - selectors
  - service endpoints
  - service discovery mechanisms
  - NodePorts
  - LoadBalancers
  - Volumes and Volume Types, Persistant Volumes (PVs)
  - Secrets, annotations...

Putting it all together: a Kubernetes Cluster



Infrastructure components are replicated as needed to scale:

Kubernetes is self-scaleable.

This is a Deployment….

Core Kubernetes controllers like DaemonSet Controller and Replication Controller. The controllers communicate with the API server to create, update, and delete the resources they manage (pods, service endpoints, etc.)

A service is a loose coupling of Pods and how to access them...

Scale out Cluster: **kubeadm join –token <token-id> ip:port**

Scale out pods: **kubectl scale --replicas 6 nova**

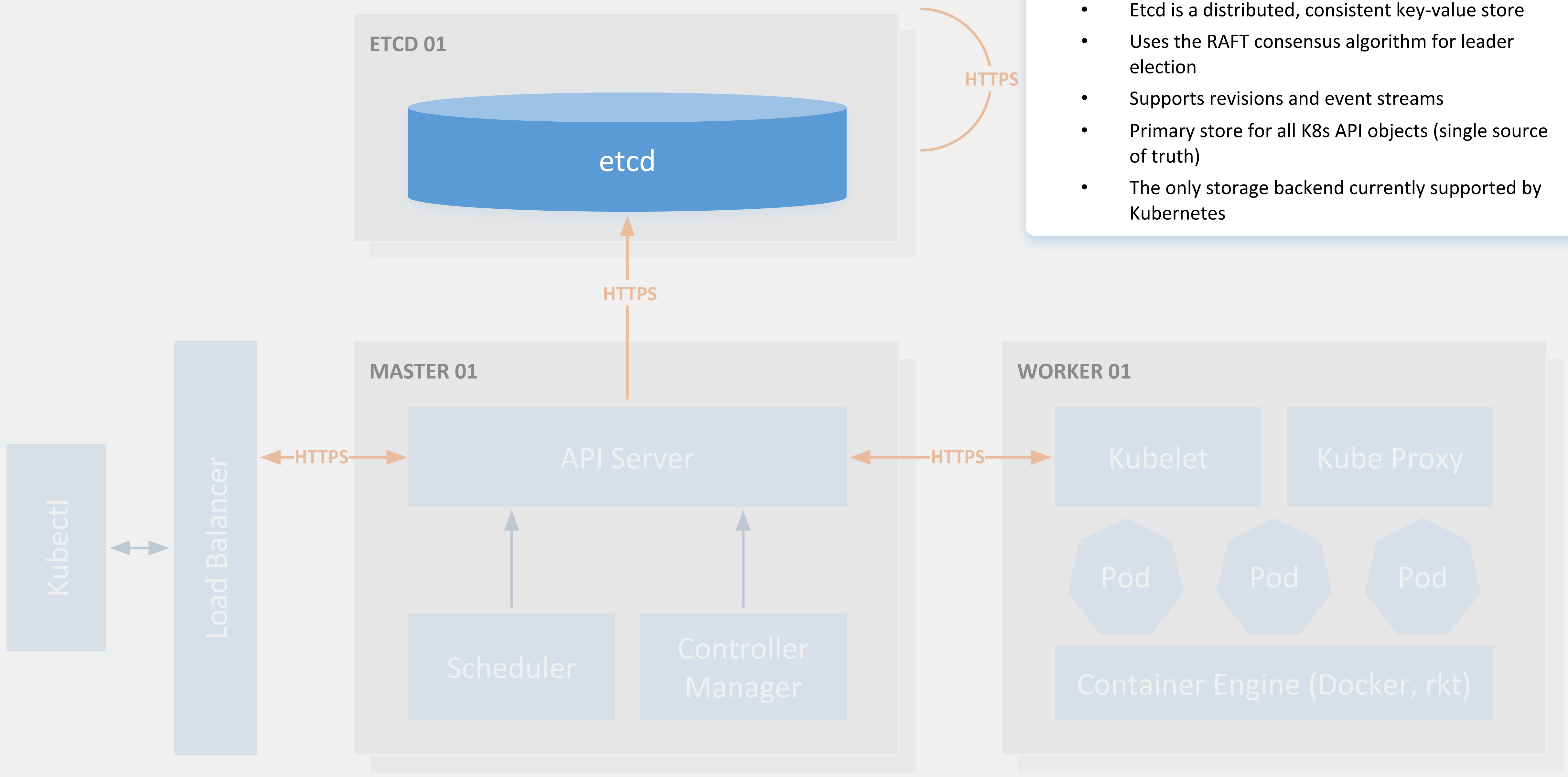
ETCD 01
Etcd
Primary store for all K8s API objects
HTTPS
HTTPS
MASTER 01
API Server
Entry point to the system, internal and external, changes state in etc, allows workloads to be scheduled across workers.
Scheduler
Schedules where pods and replicas live, based on policy, resources, affinity, labels, QOS, etc.
Controller Manager
Daemon process that monitors desired state and fixes
Control Plane
Kubectl
Load Balancer
HTTPS
ANALYZE
Replicas and Pod Health
Desired State
Controller Manager
Current State
UPDATE
Replicas and Pod Health
HTTPS
WORKER 01
Kubelet
Pod Agent ensures pods are healthy.
Interacts with container runtime.
Kube Proxy
Network proxy and load balancer for Services - routes traffic to correct container
Pod
Pod
Pod
Container Engine (Docker, rkt)

Genesis Cloud

Kubernetes Architecture Deep Dive

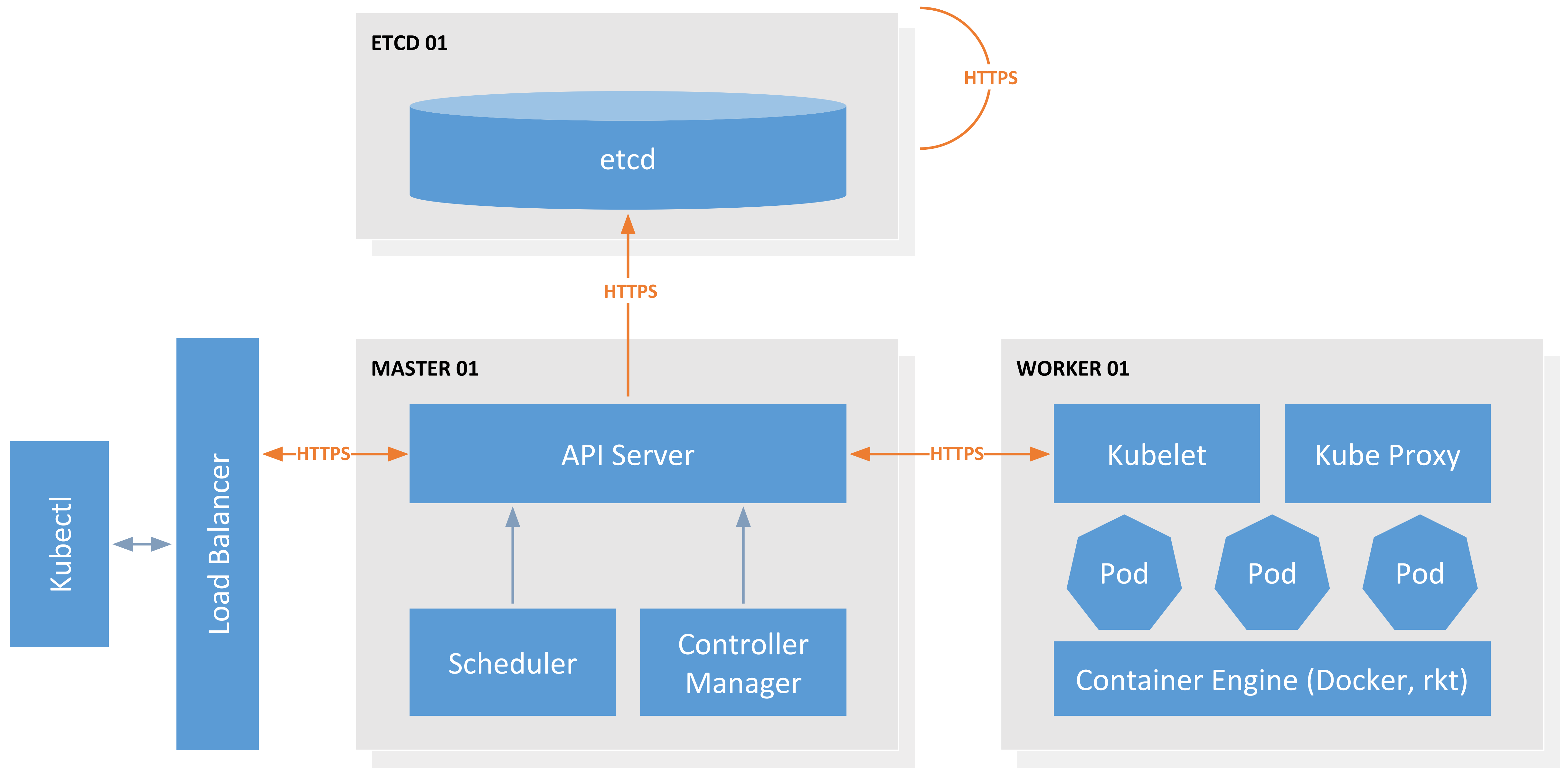
Deep dive into architecture

Genesis Cloud

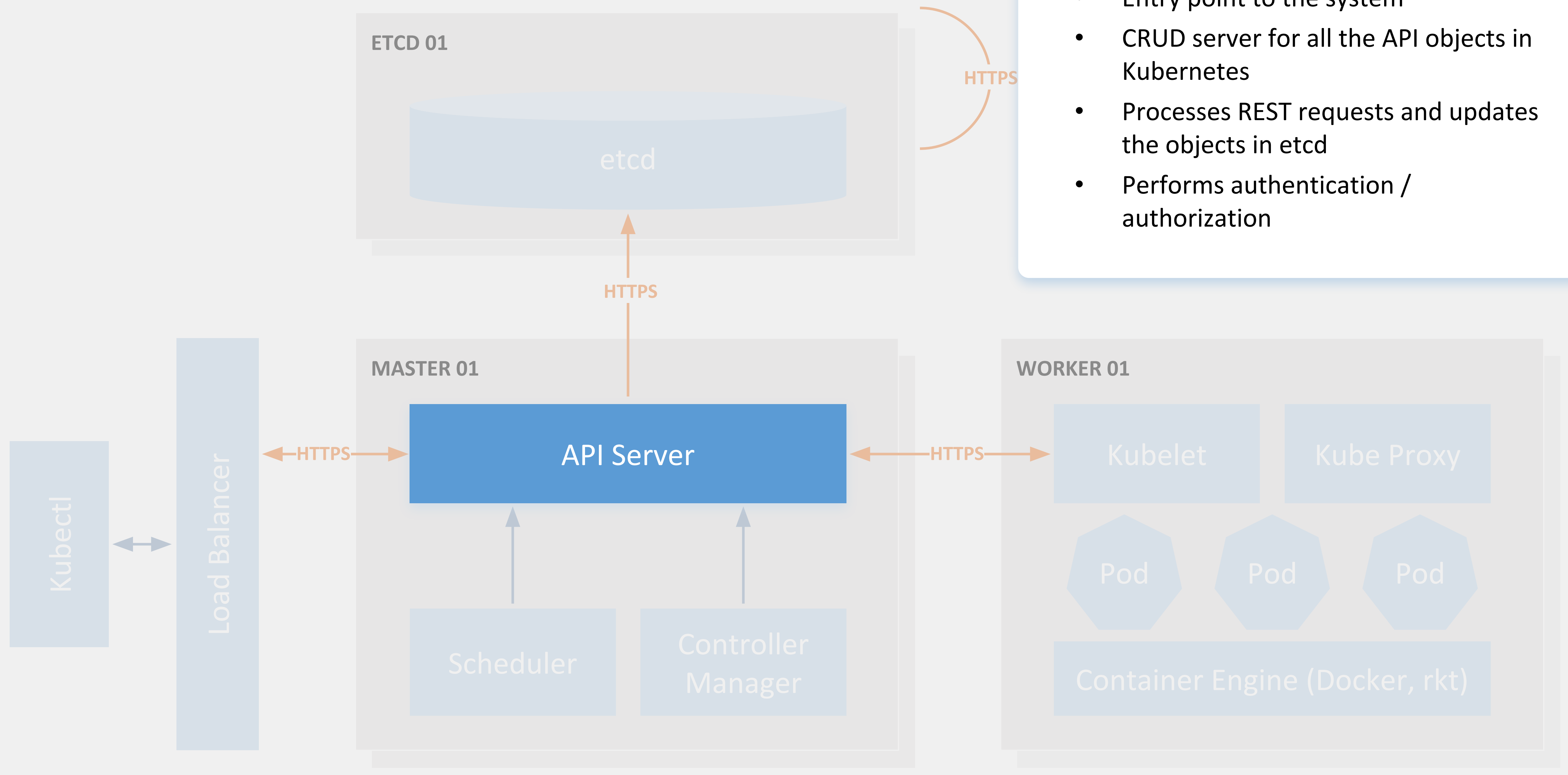


- Etcd is a distributed, consistent key-value store
- Uses the RAFT consensus algorithm for leader election
- Supports revisions and event streams
- Primary store for all K8s API objects (single source of truth)
- The only storage backend currently supported by Kubernetes

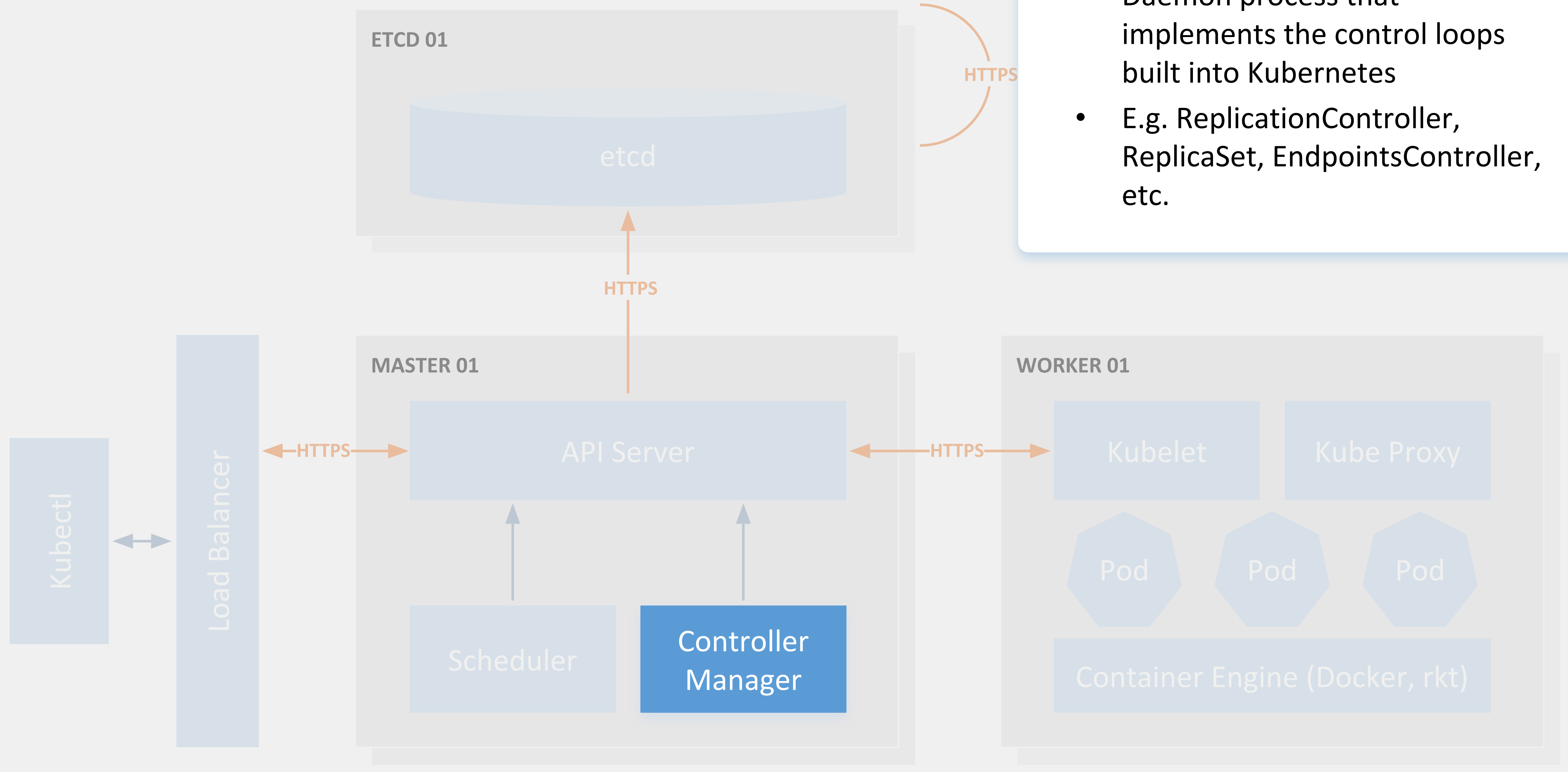
Genesis Cloud



Genesis Cloud

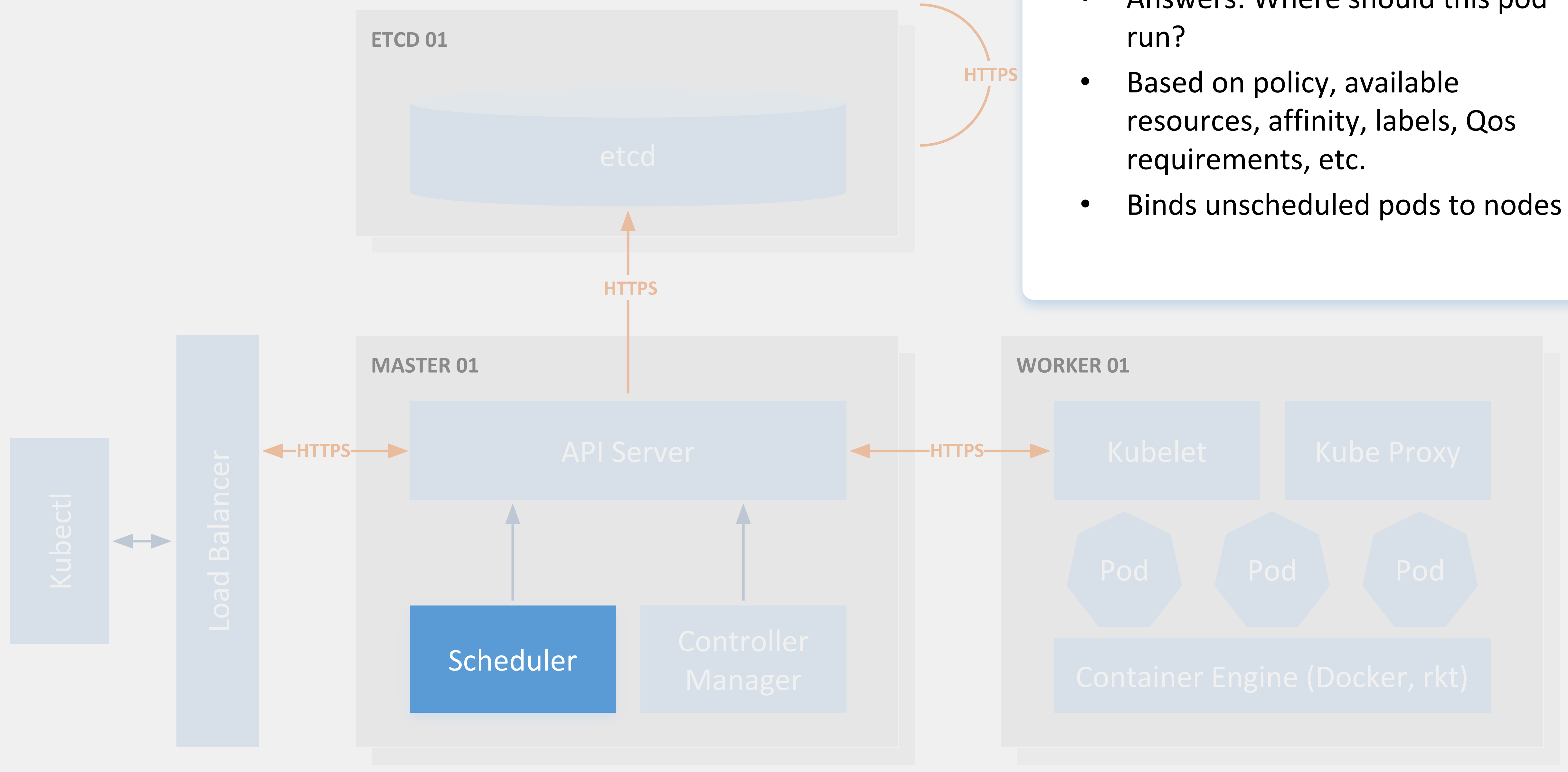


- Entry point to the system
- CRUD server for all the API objects in Kubernetes
- Processes REST requests and updates the objects in etcd
- Performs authentication / authorization



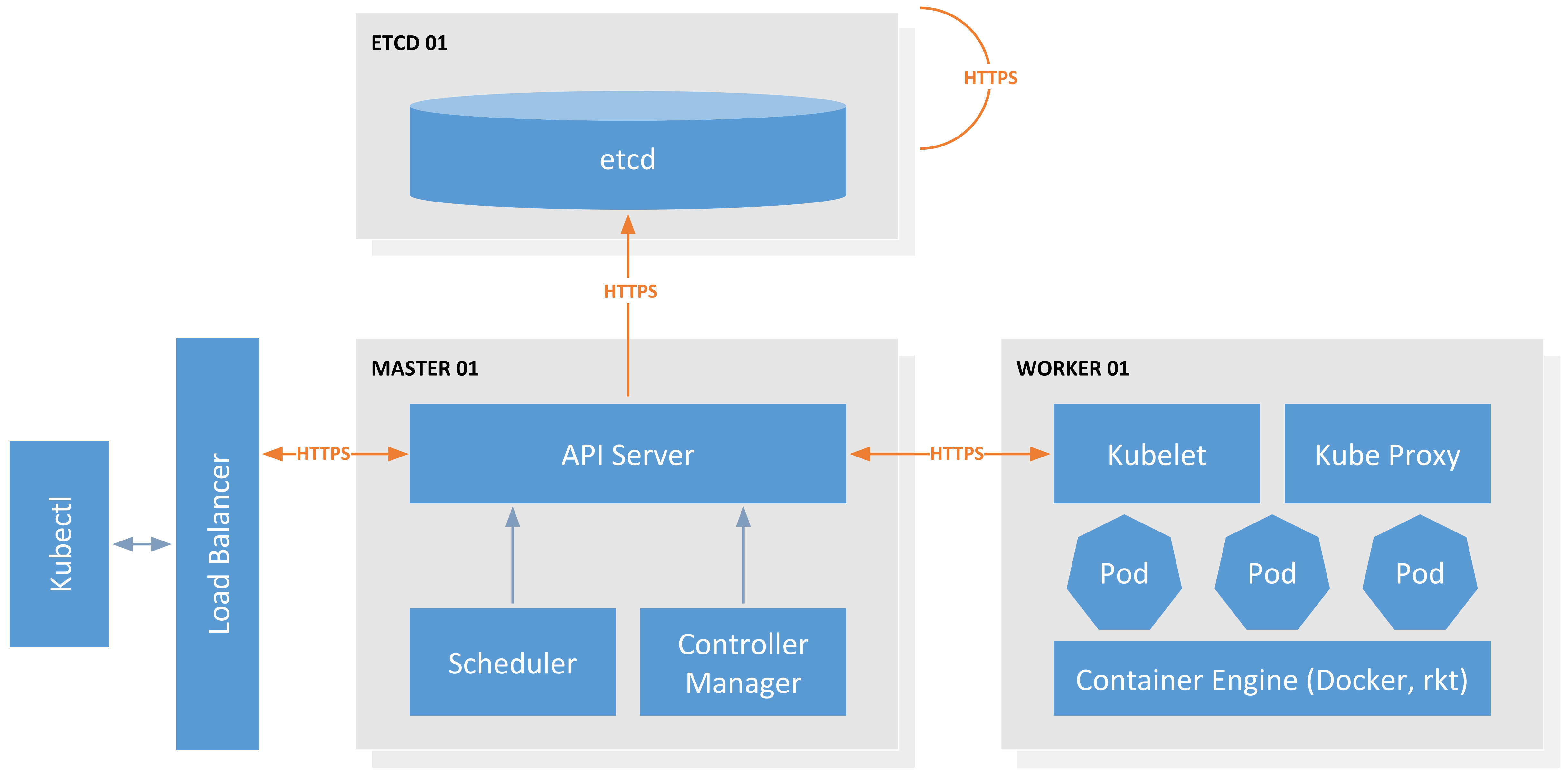
- Daemon process that implements the control loops built into Kubernetes
- E.g. ReplicationController, ReplicaSet, EndpointsController, etc.

Genesis Cloud

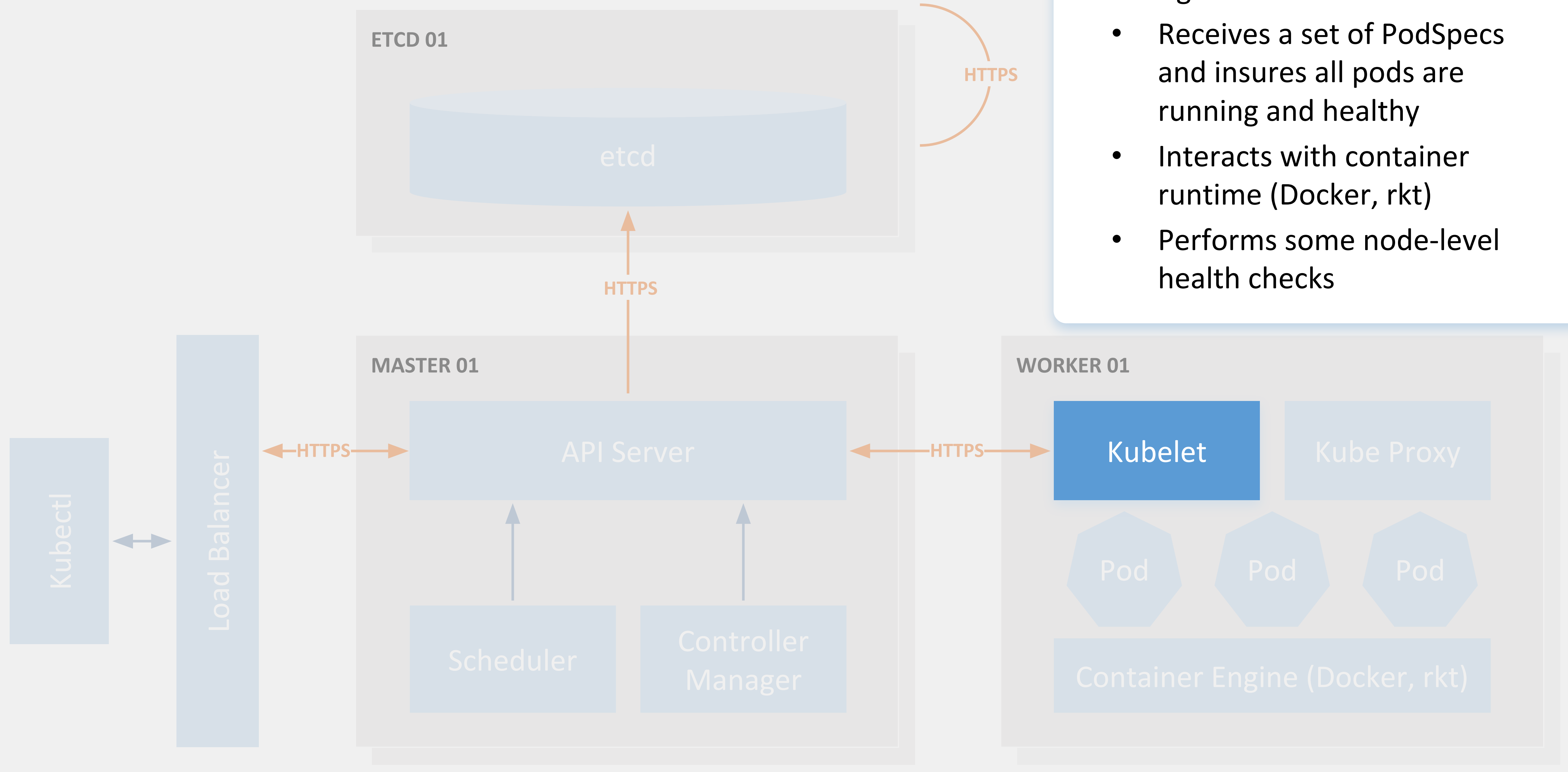


- Answers: Where should this pod run?
- Based on policy, available resources, affinity, labels, Qos requirements, etc.
- Binds unscheduled pods to nodes

Genesis Cloud

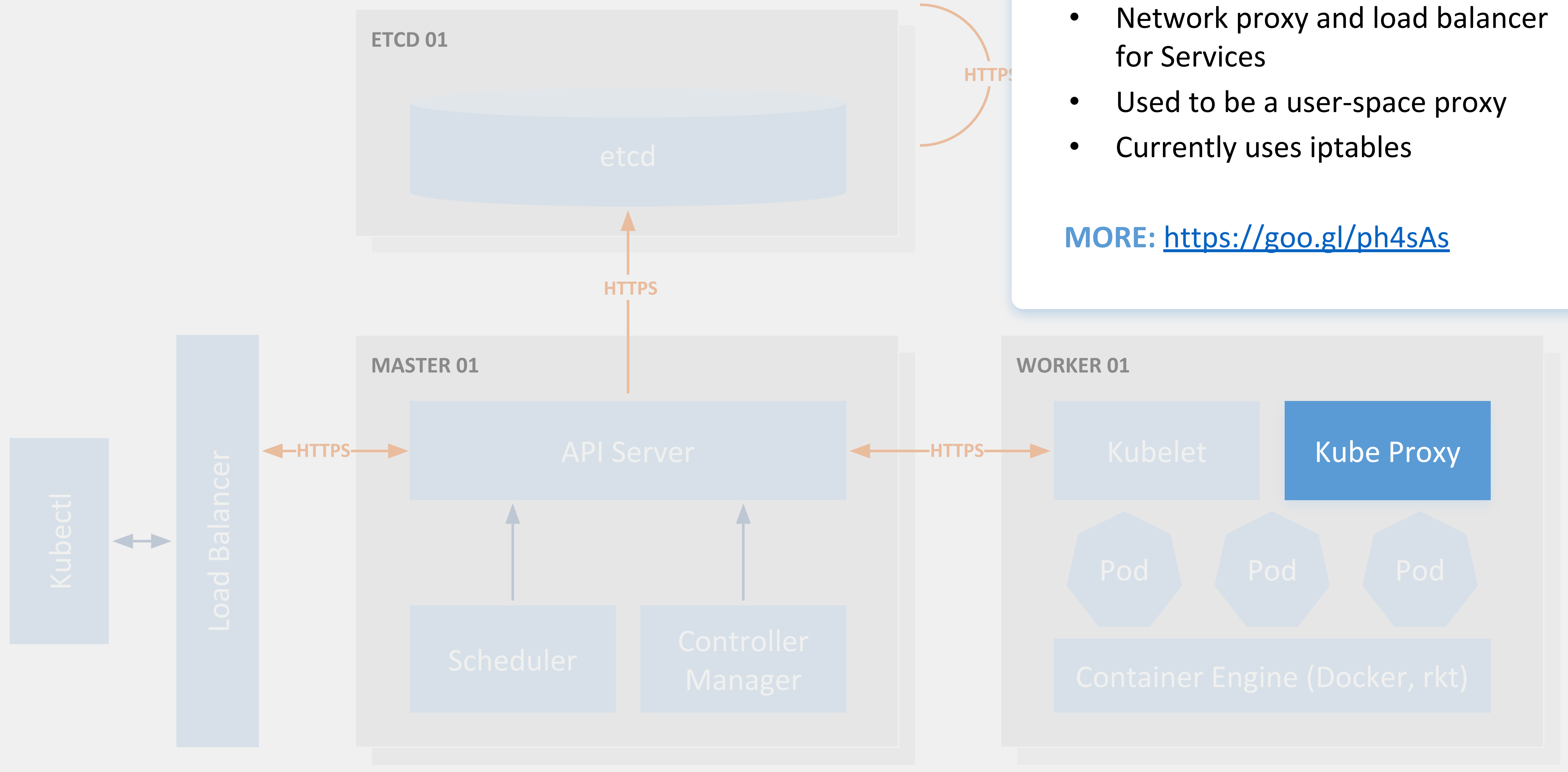


Genesis Cloud



- Agent that runs on each node
- Receives a set of PodSpecs and insures all pods are running and healthy
- Interacts with container runtime (Docker, rkt)
- Performs some node-level health checks

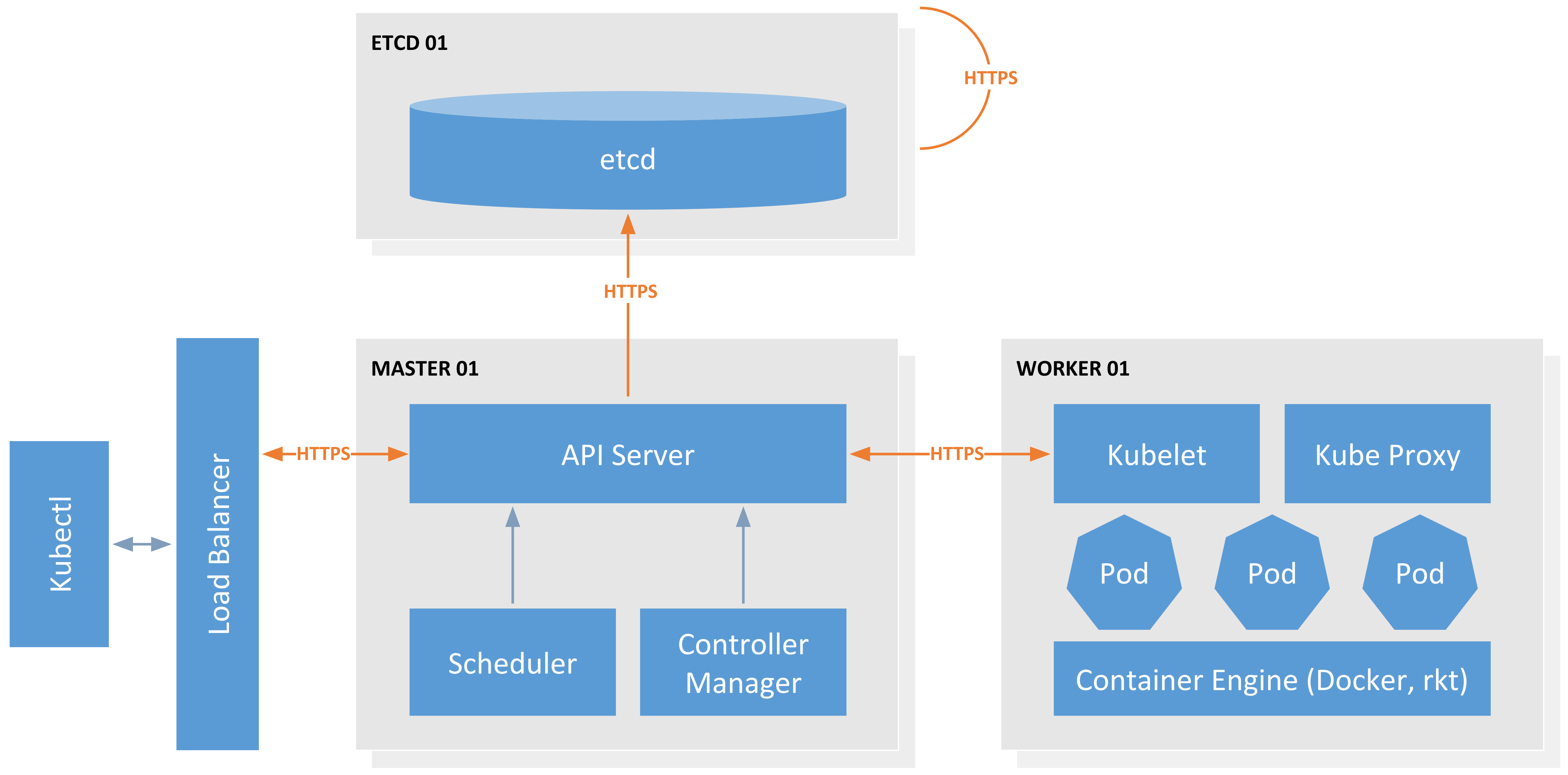
Genesis Cloud



- Network proxy and load balancer for Services
- Used to be a user-space proxy
- Currently uses iptables

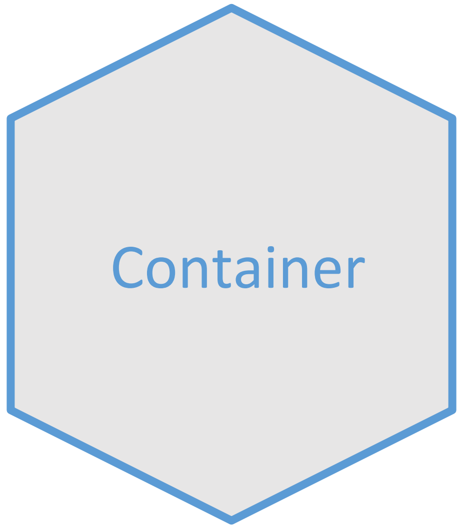
MORE: https://goo.gl/ph4sAs

Genesis Cloud



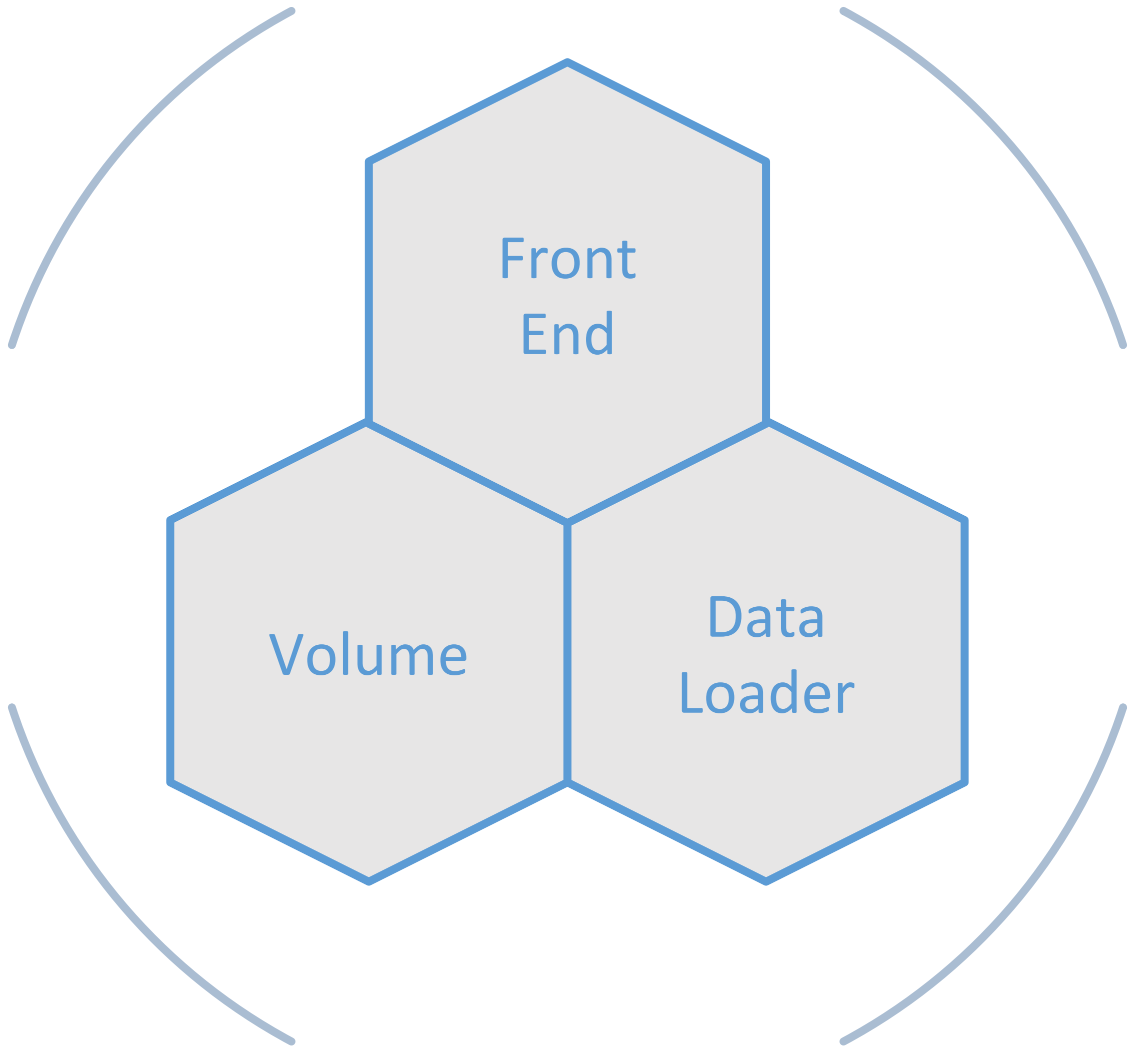
Genesis Cloud

Pods



Genesis Cloud

Pods



- Level of abstraction around one or more containers

Pods

Set of Related Containers

Tightly Coupled

Ephemeral/Stateless

Share Namespace

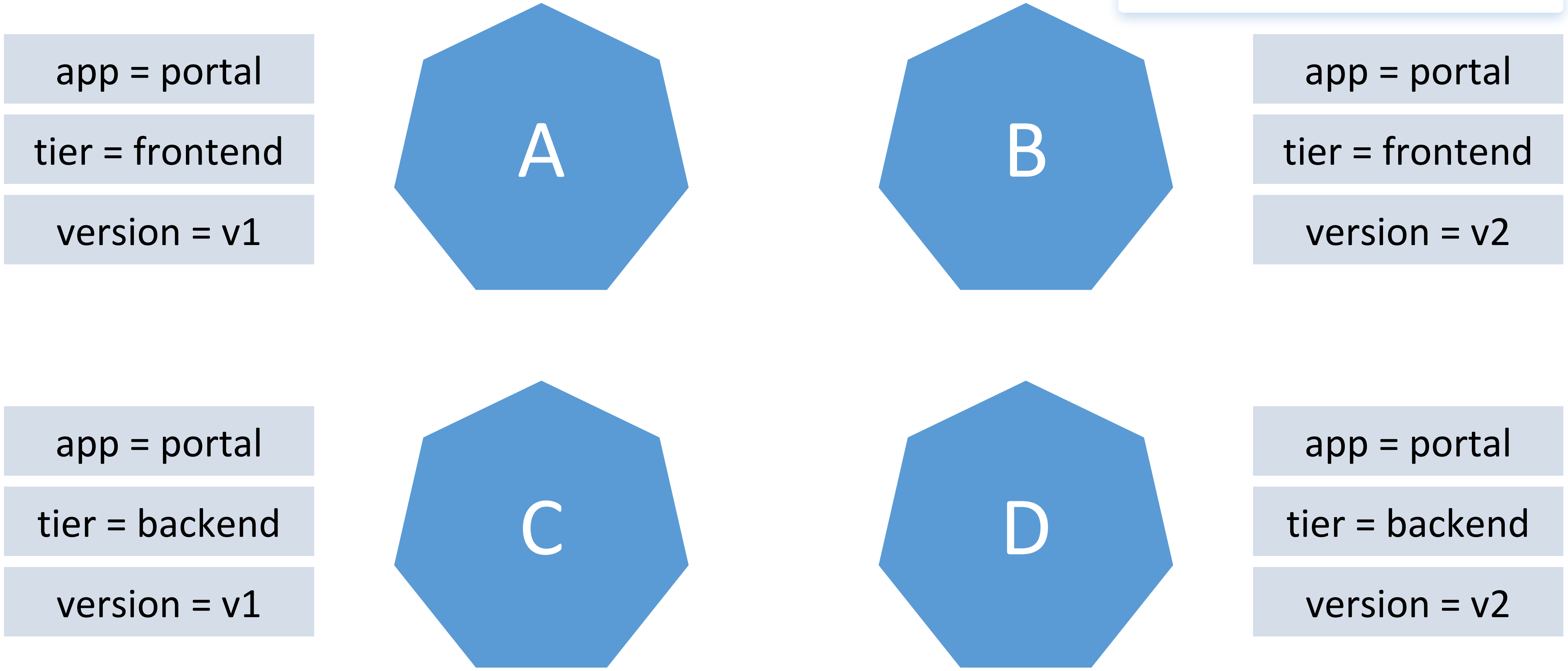
Share Network

Unit of Management

Genesis Cloud

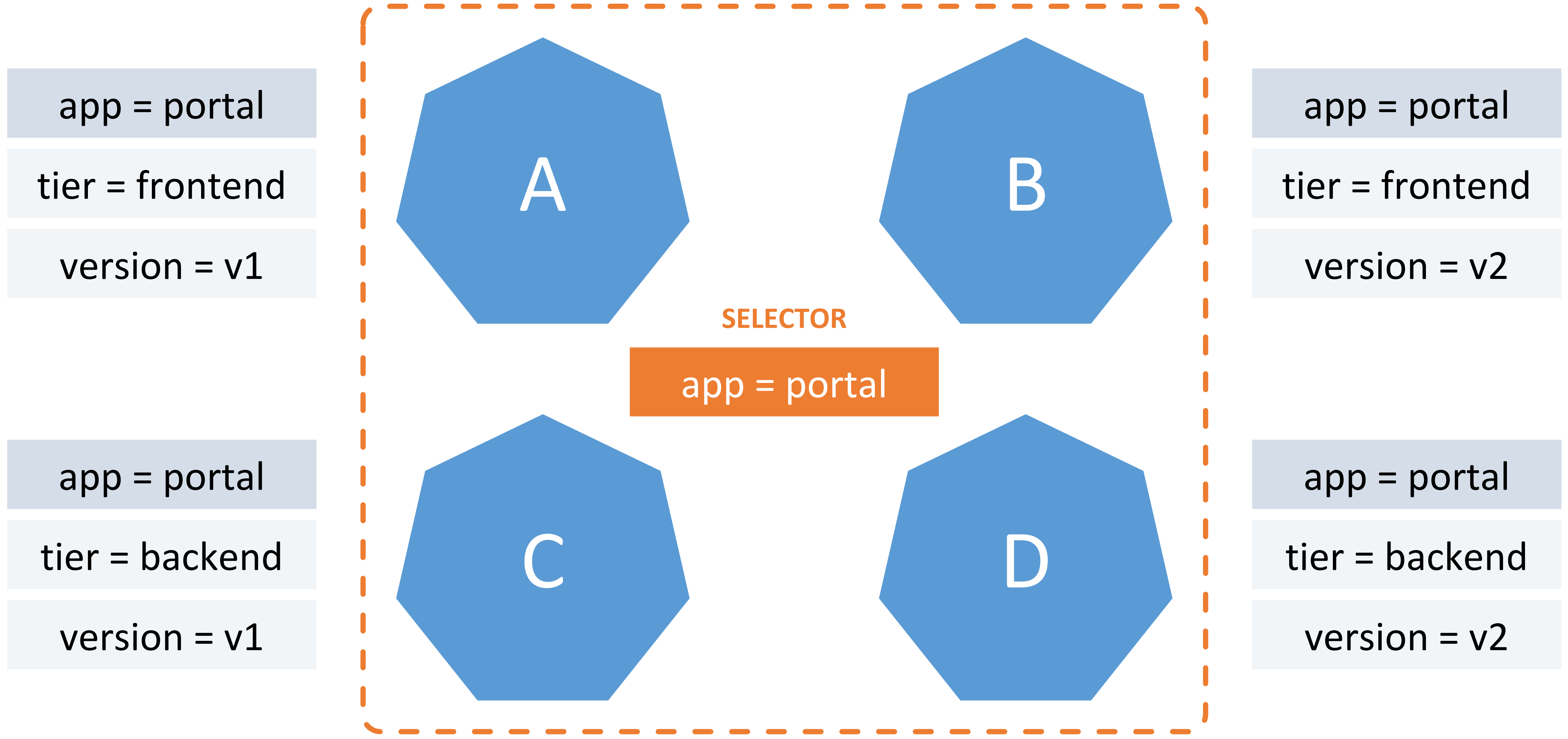
Labels

- Helps to identify pods.



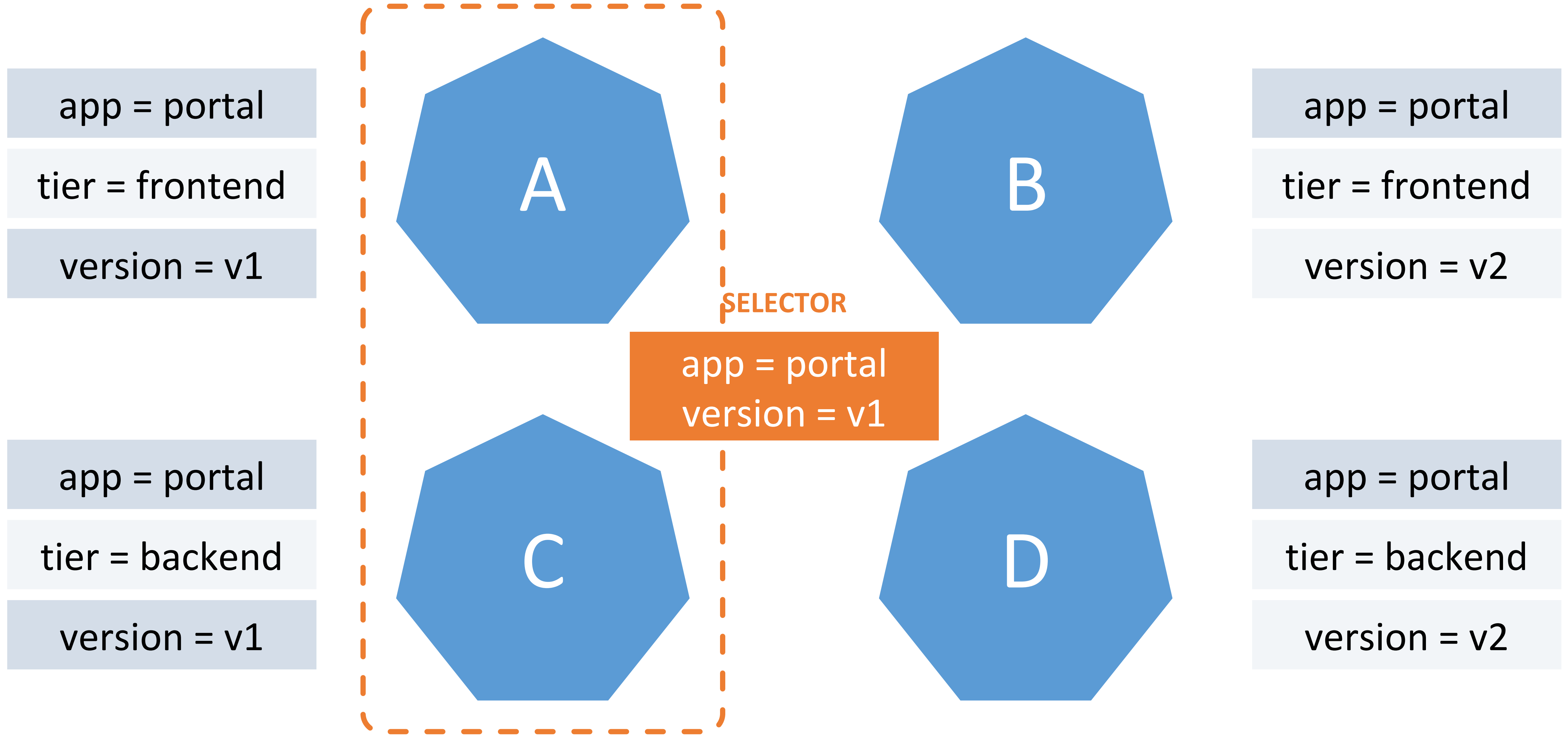
Genesis Cloud

Label Selectors



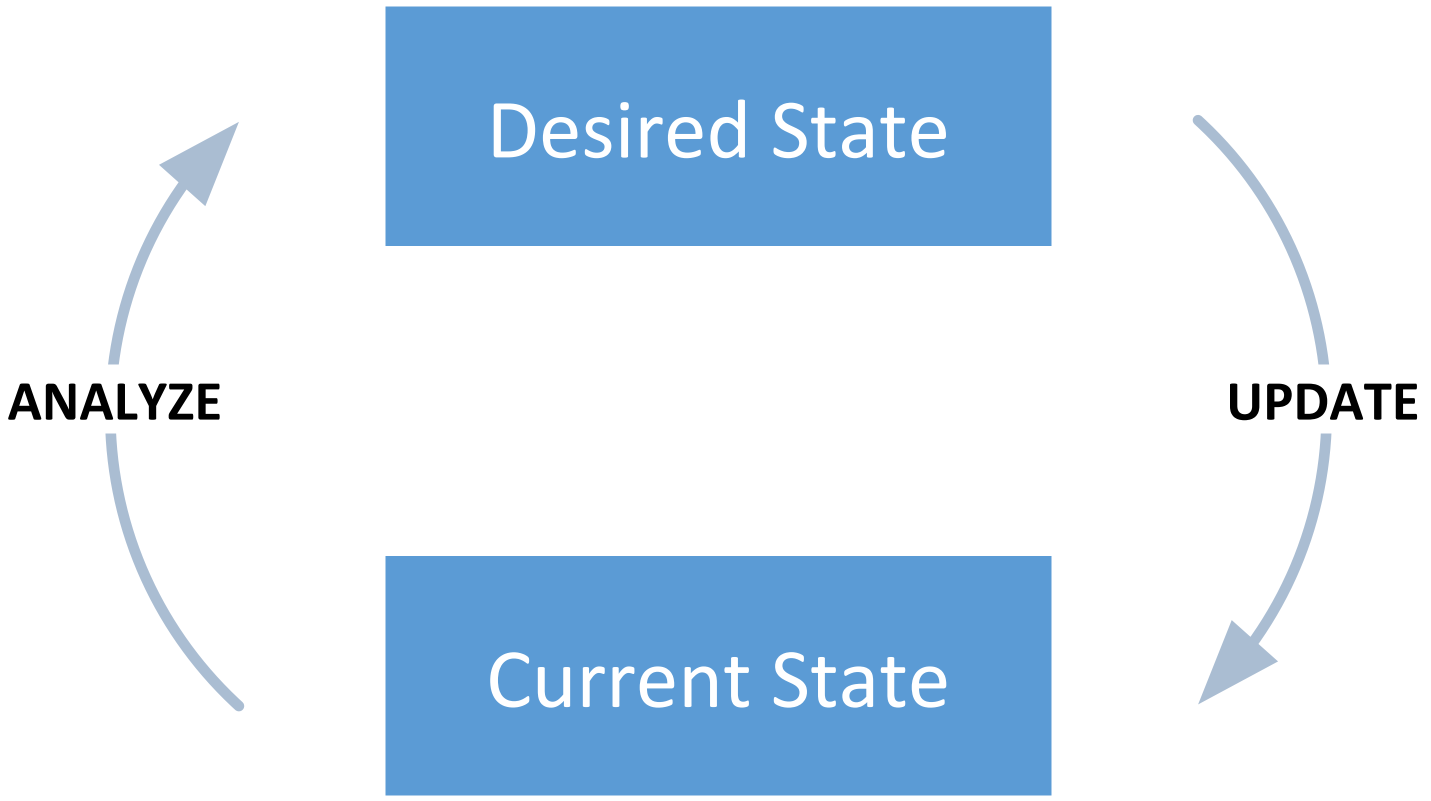
Genesis Cloud

Label Selectors



Genesis Cloud

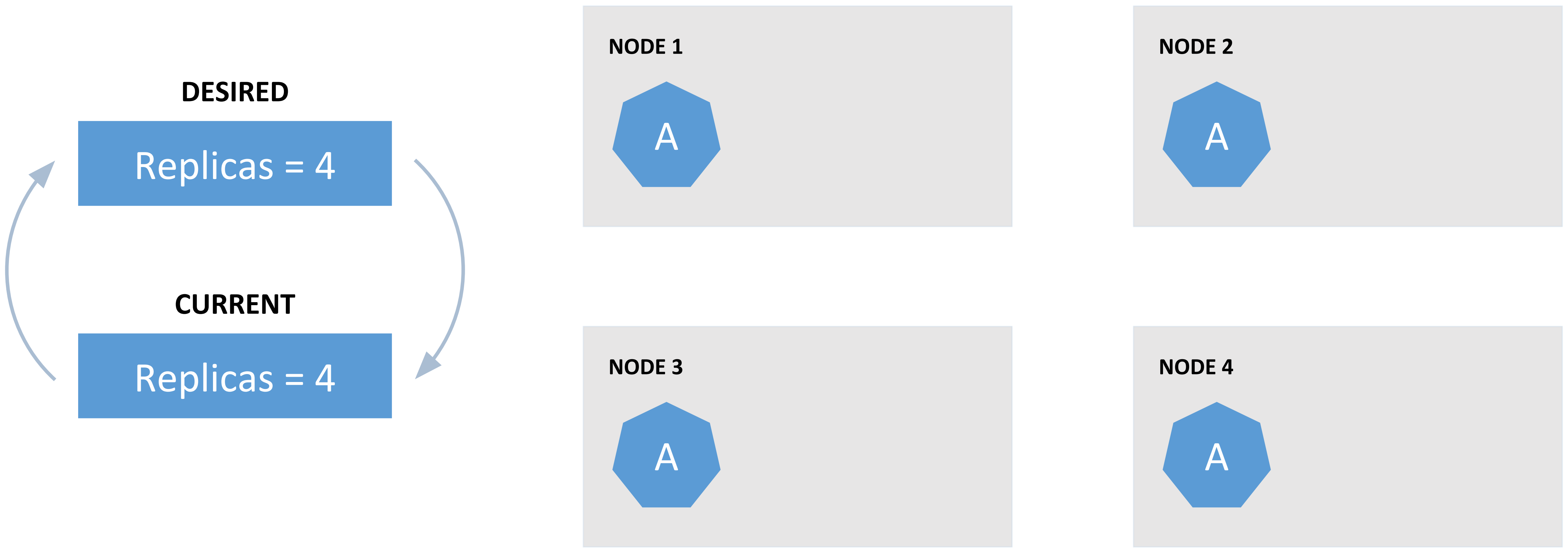
Controllers



Genesis Cloud

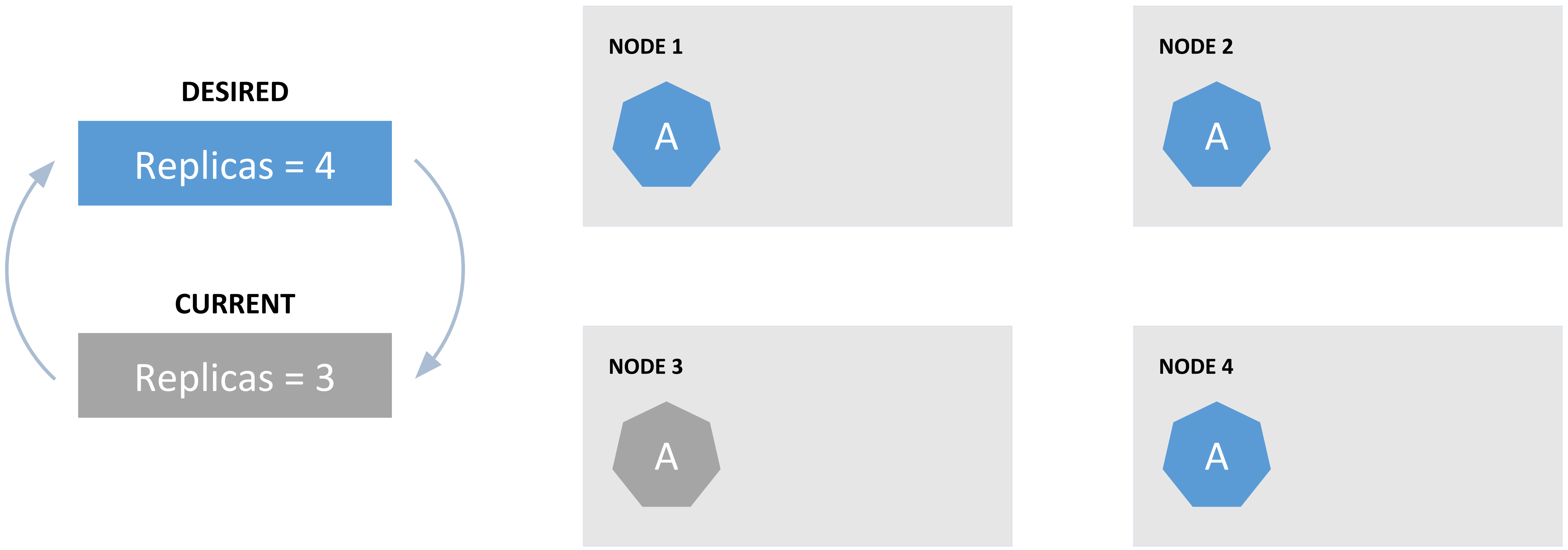
Replica Set (Controller)

- Replicas = Resilience = Scalability



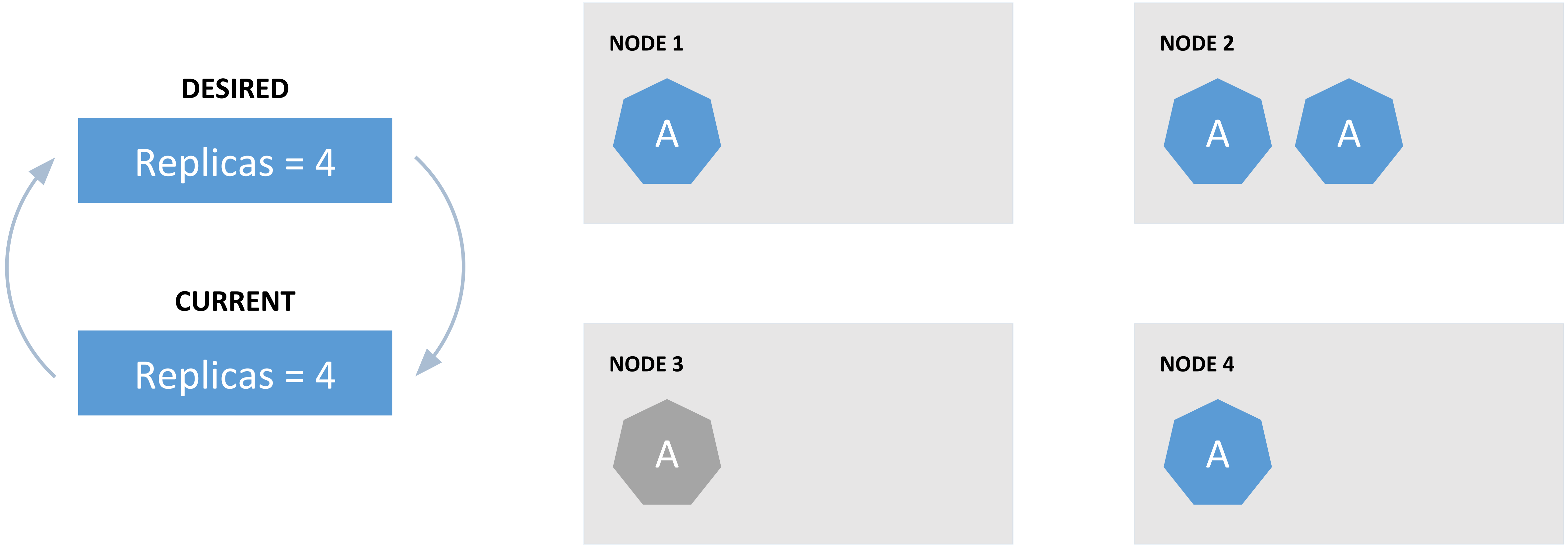
Genesis Cloud

Replica Set (Controller)



Genesis Cloud

Replica Set (Controller)

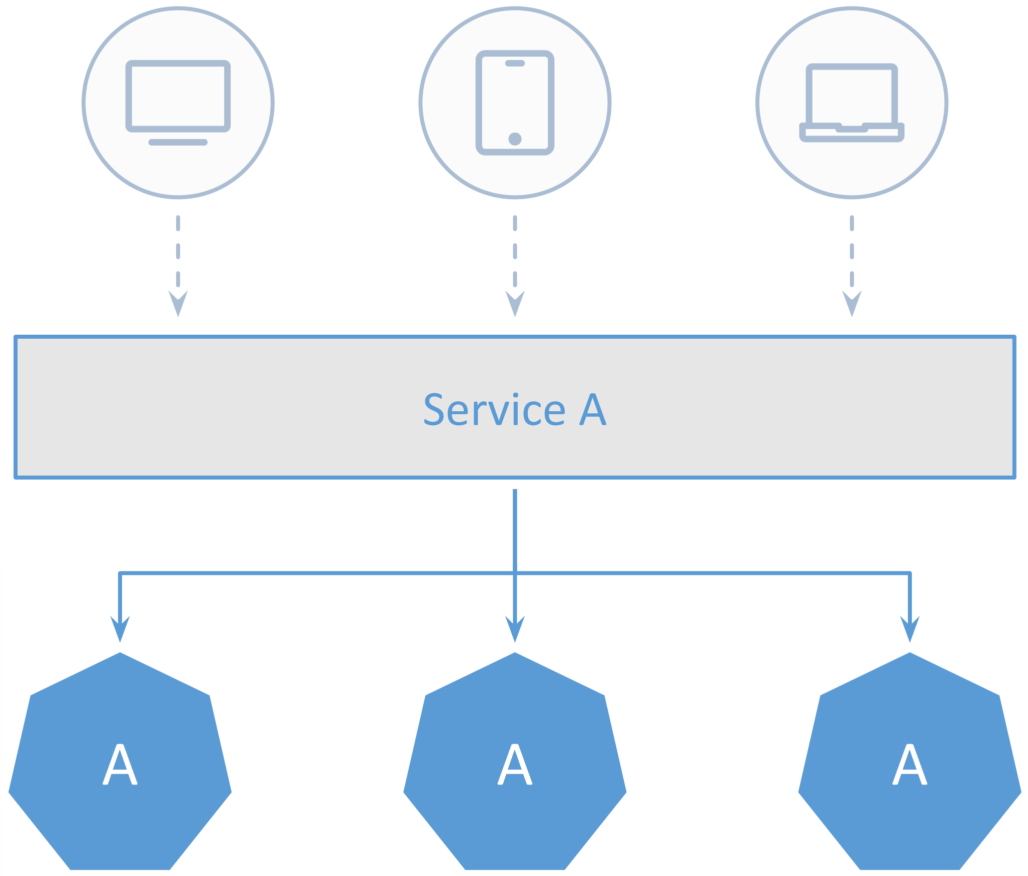


Genesis Cloud

(µ)Services

SELECTOR

app = portal
tier = frontend
version = v1



- Services provide abstraction to pods and replicas.
- Unique entry point into pod.

Stable Cluster-wide IP

Cluster-wide DNS name

Layer 3 Load Balancer

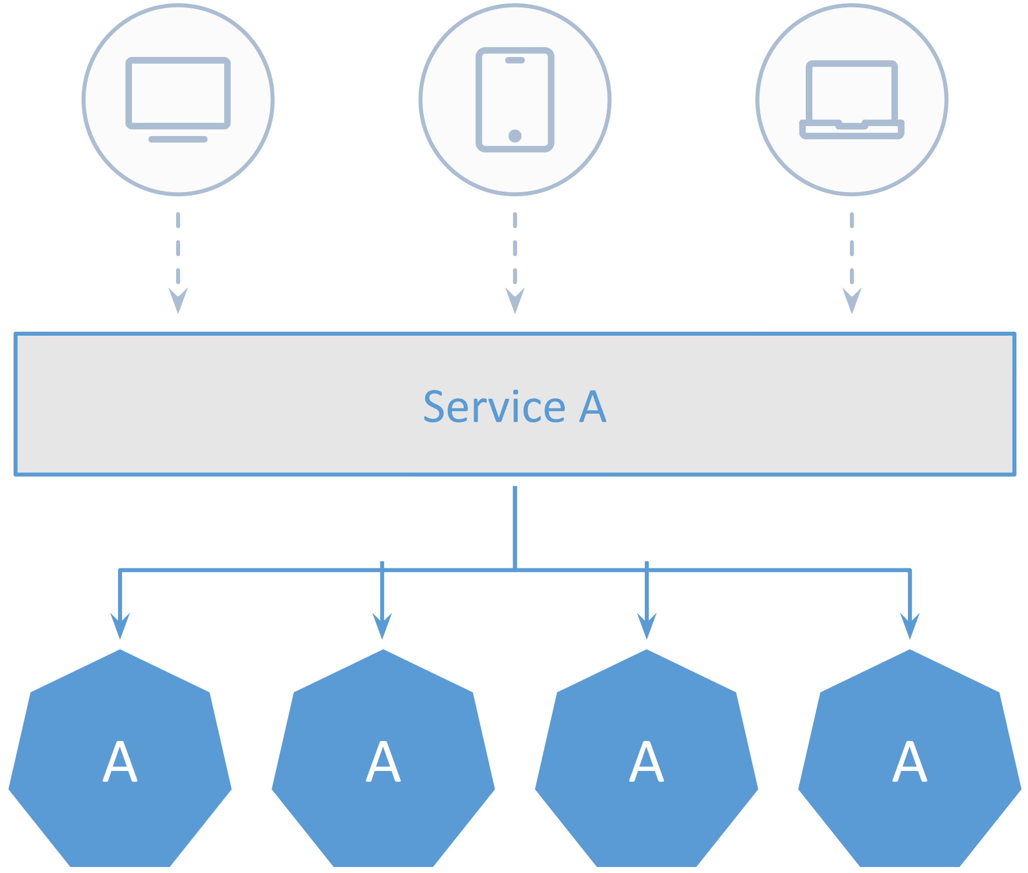
Updates continuously

Genesis Cloud

(µ)Services

SELECTOR

app = portal
tier = frontend
version = v1



New Pod added automatically based on its label selector

Genesis Cloud

Kubernetes Objects

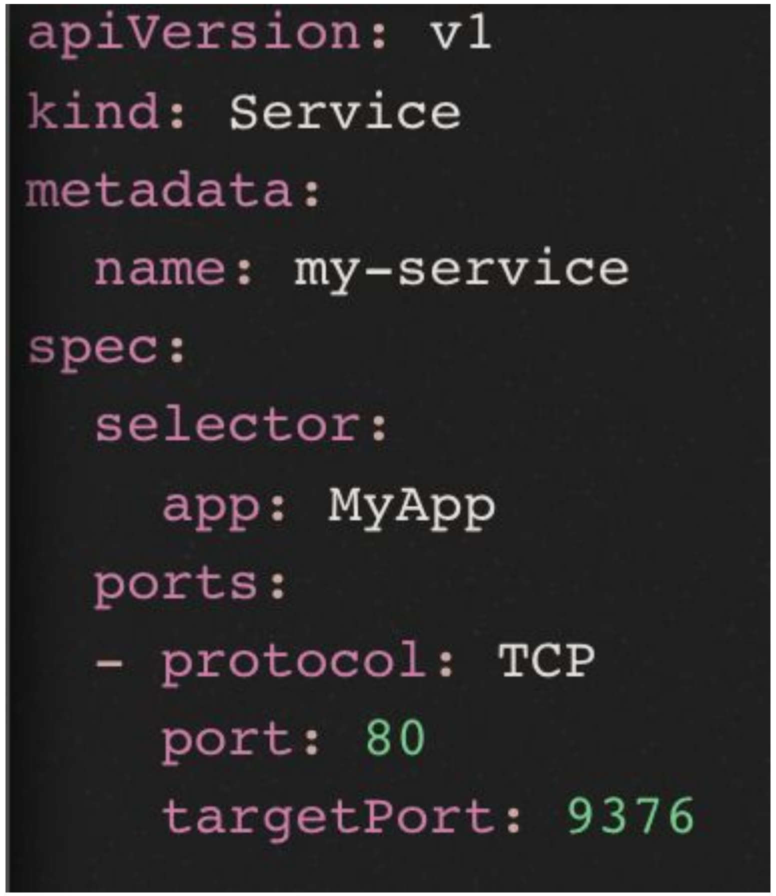
- *Example of defining objects in YAML*
- Let’s have a look at a typical kubernetes yaml structure.
- If you’re not familiar with general yaml file format you can have a quick look here

Genesis Cloud

Kubernetes Objects

- The Kubernetes Object Model represents containerized applications, app resource consumptions and app policies.
- A Spec. field declares the intent or desired state of objects.
- Kubernetes system manages the status field of objects and tried to match the actual state with the desired state.
- Examples of Objects are: Pods, Deployments, ReplicaSets etc.
- A YAML spec file is created, which is converted by kubectl into a JSON payload and sent to the API Server.
- Qn: how do deploy an application defined by Object Yaml?
- Ans: ***kubectl create -f application.yaml***

Kubernetes Object YAML

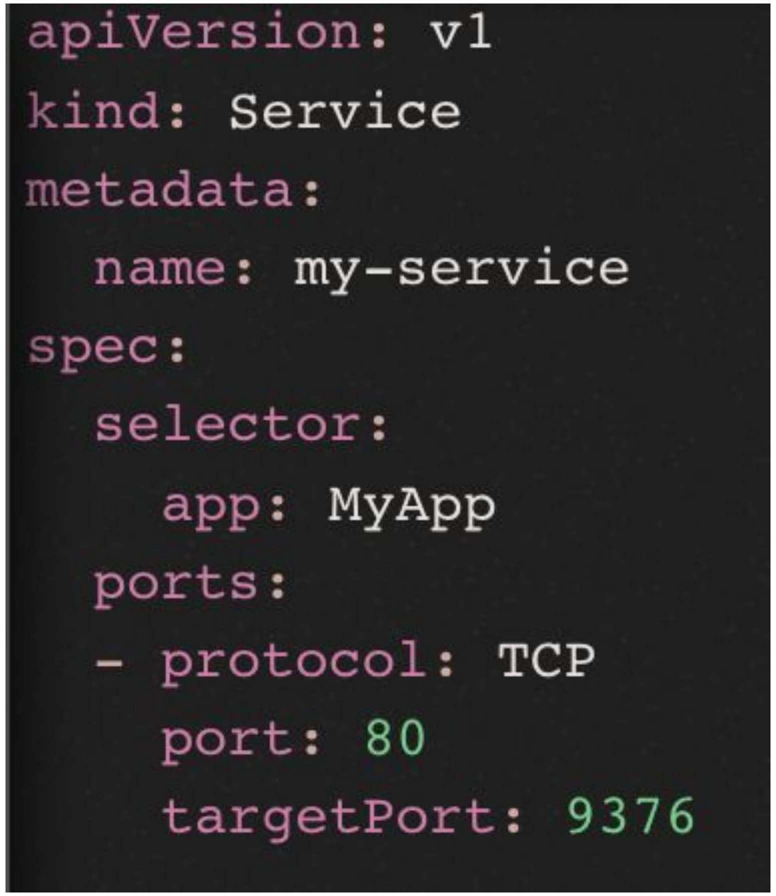


```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

The fields required to describe a k8s object in a yaml file are:

- apiVersion
- kind
- metadata and
- spec

Kubernetes Object YAML

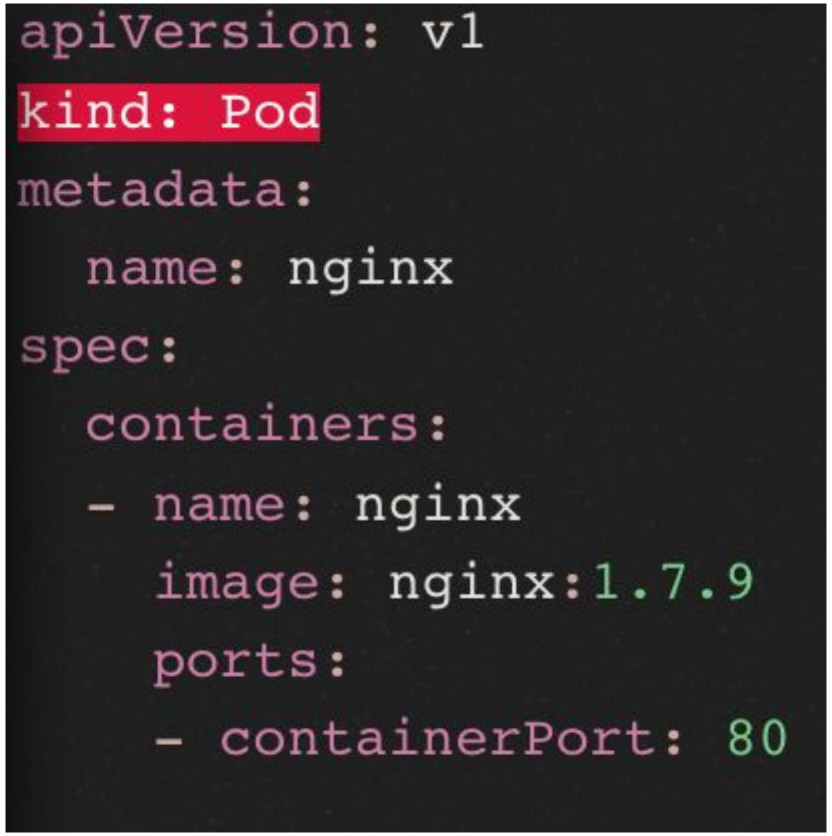


```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

- The **apiVersion** defines the API version of k8s used to create this object
- The **kind** defines what kind of object you want to create, such as Pod, Deployment, Service etc
- The **metadata** defines data that uniquely identifies the object, like a name
- The **spec** specifies the details of each object you create. The precise format of the spec field is different for every k8s object.
  - For details you can always have a look at the k8s API Reference

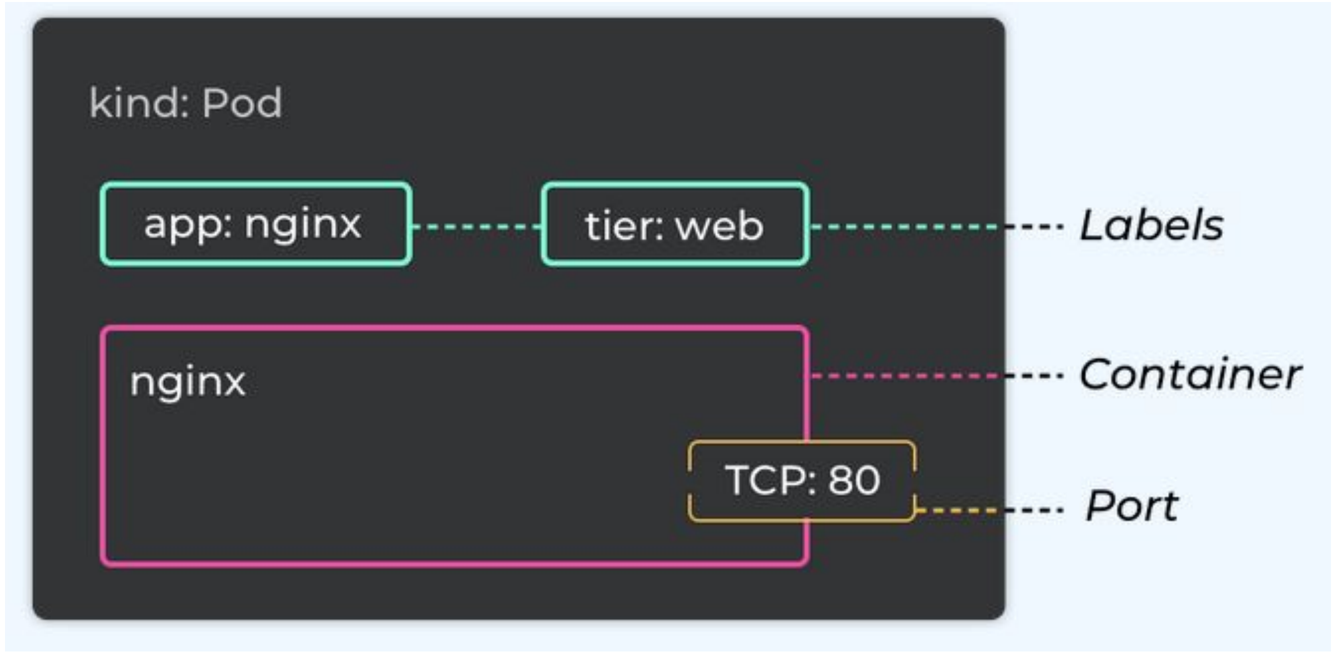
Genesis Cloud

Kubernetes Object YAML - a POD (1)



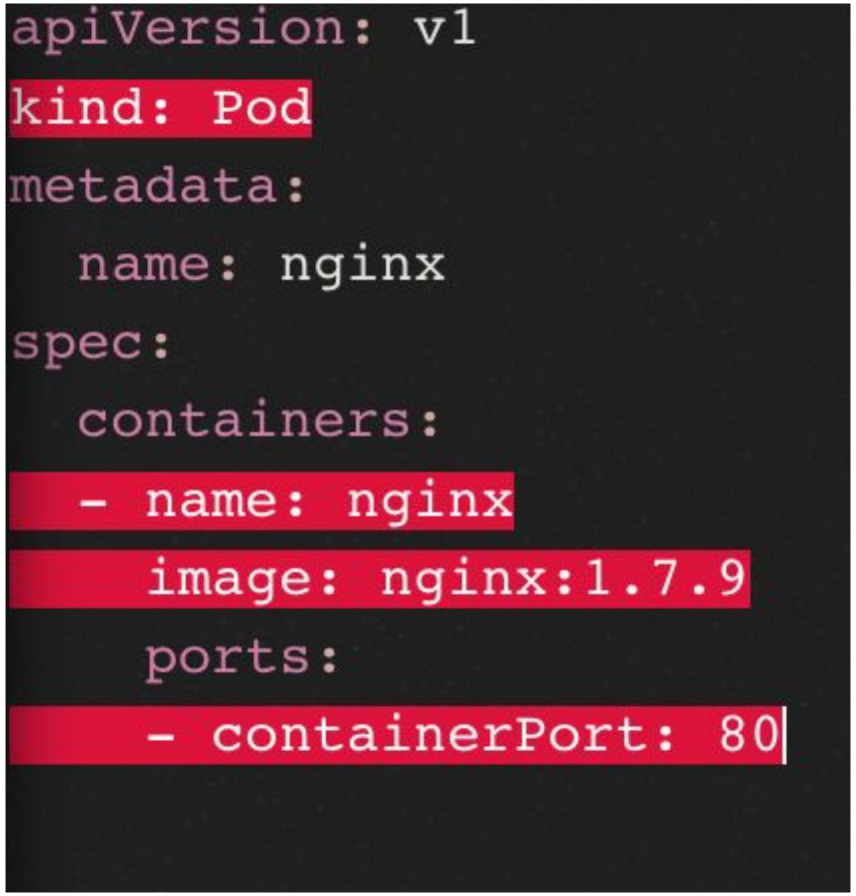
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

- A Pod is a grouping of Docker containers running as a unit and one of the main building blocks of kubernetes.



Genesis Cloud

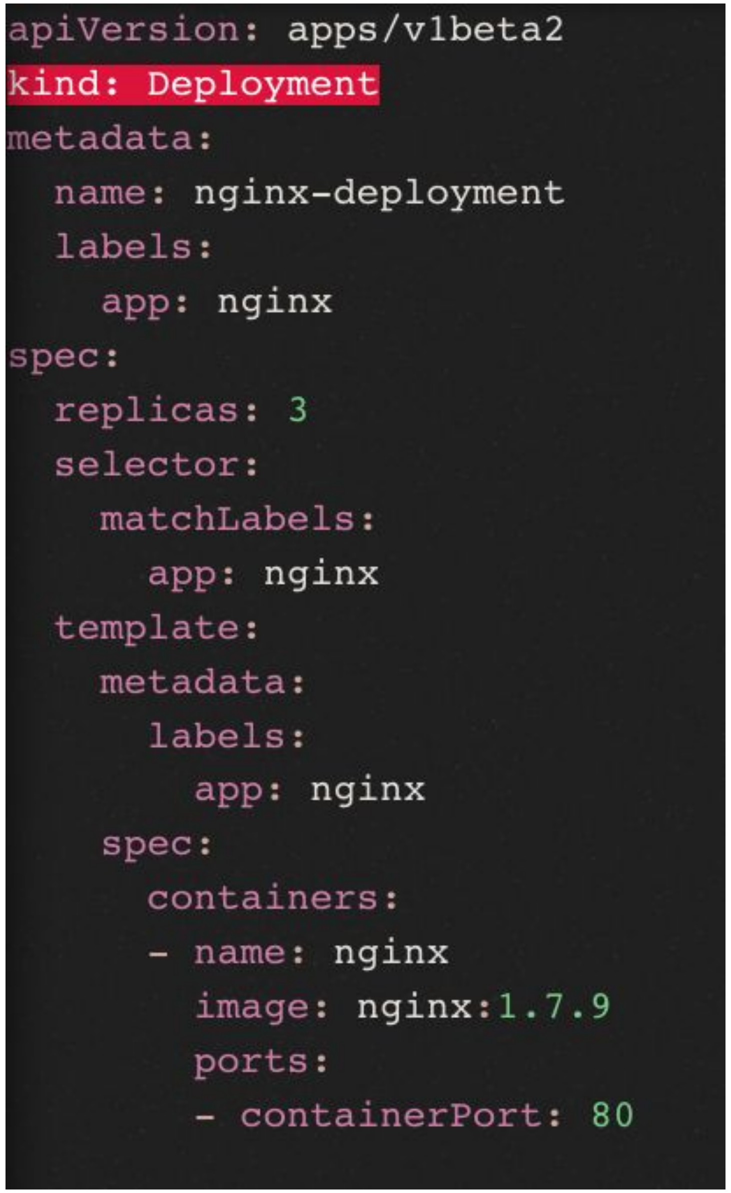
Kubernetes Object YAML - a POD (2)



```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

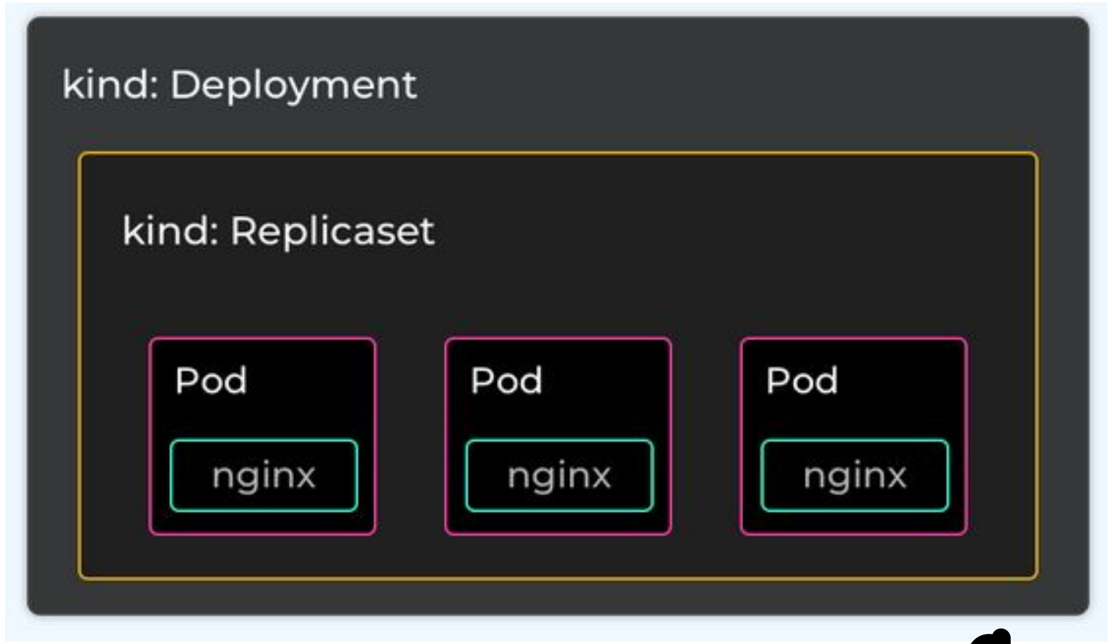
- In this example we see a Pod running one container.
- We've named the container nginx, defined the docker image to be ran as nginx:1.7.9 and opened the port 80

Kubernetes Object YAML - a Deployment (1)

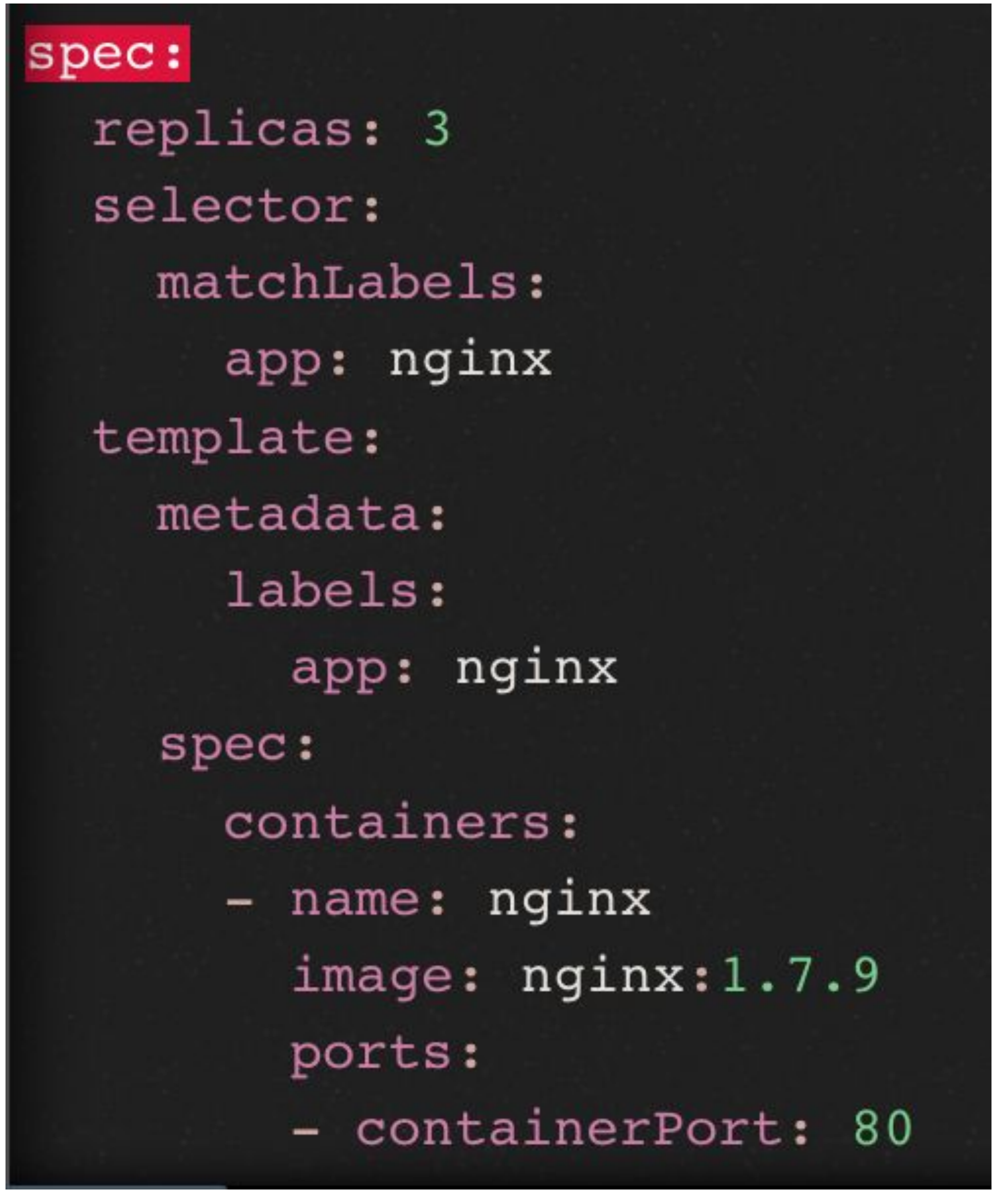


```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

- A Deployment manages multiple pods Pods through controllers called ReplicaSets.
- You describe a desired **state** in a Deployment object, and the Deployment controller changes the actual state to the desired state at a controlled rate.
- Deployments



Kubernetes Object YAML - a Deployment (2)


```
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

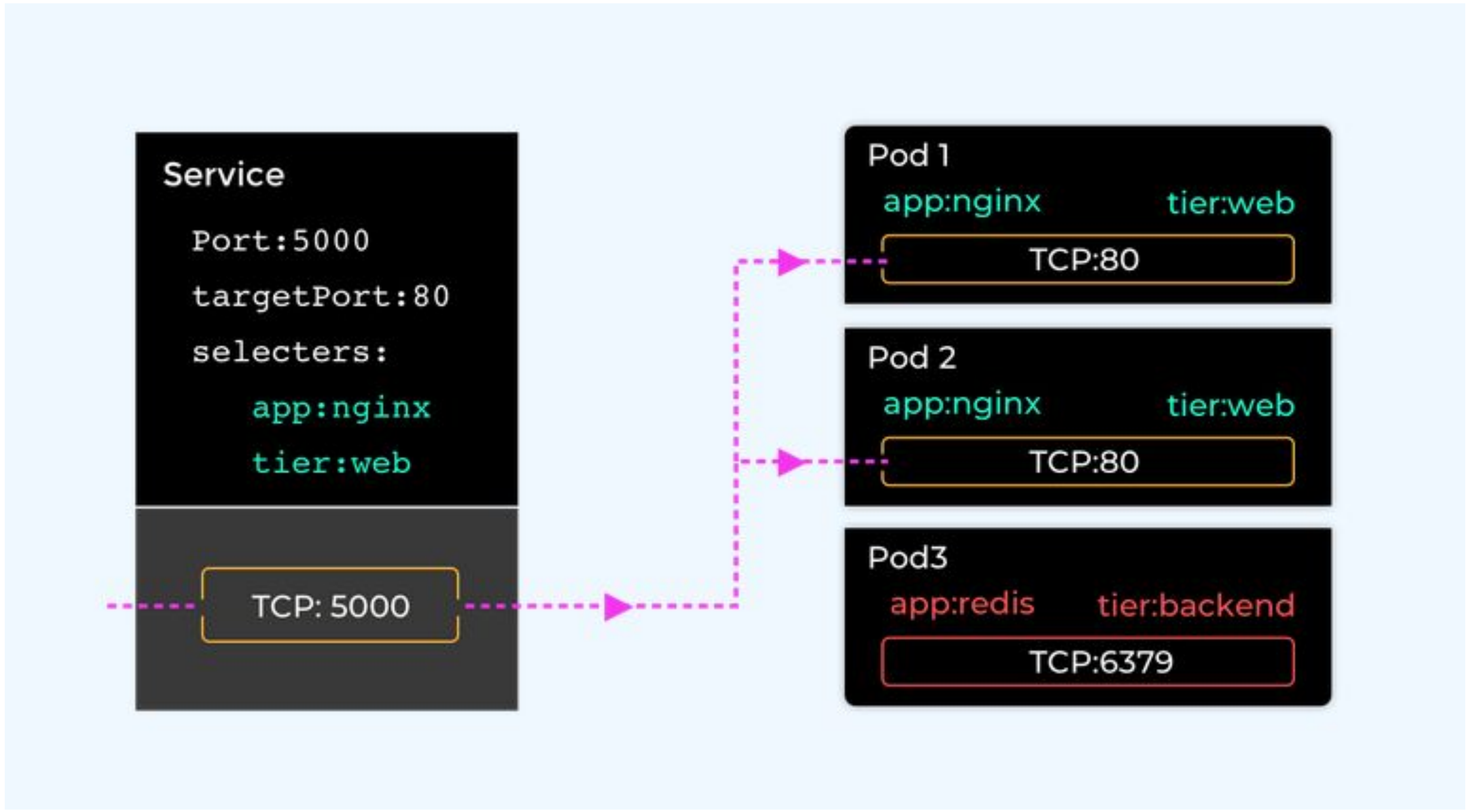
- The replicas field defines that we want 3 instances of our pod
- The selector field defines how will the deployments target which pods are under its management
- Every managed pod should have labels assigned to it, and the matchLabels selector targets the pods with the labels specified.
- The template field defines the template of the pods being created
- In this case a pod with a container named nginx, using the nginx:1.7.9 image, with the port 80 open

Genesis Cloud

Kubernetes Object YAML - a Service (1)

- Kubernetes Pods are mortal. They are born and when they die, they may not be resurrected.
- ReplicaSets in particular create and destroy Pods dynamically (e.g. when scaling up or down).
- While each Pod gets its own IP address, even those IP addresses cannot be relied upon to be stable over time.
- This leads to a problem: if some set of Pods (let’s call them backends) provides functionality to other Pods (let’s call them frontends) inside the Kubernetes cluster, how do those frontends find out and keep track of which backends are in that set?

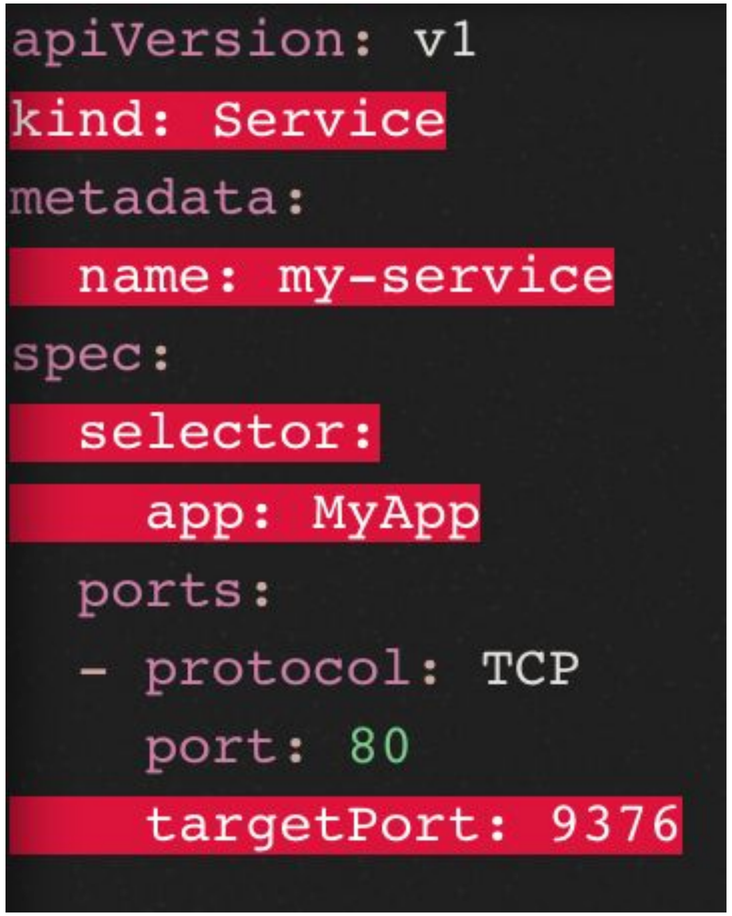
Kubernetes Object YAML - a Service (2)



- A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them - sometimes called a micro-service.
- The set of Pods targeted by a Service is (usually) determined by a Label Selector

Genesis Cloud

Kubernetes Object YAML - a Service (3)



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

- As specified by the kind: Service field, this is a Service object.
- This specification will create a new Service object named my-service which targets TCP port 9376 on any Pod with the app=MyApp label.
- The **Service** will also be assigned its own IP address (sometimes called the ‘cluster IP’.
- The service will route all traffic coming into it’s incoming port (in this case 80) to the targetPort of the pods it routes to
- The selector field is the primary mechanism of finding pods which to target.
- In this case, all pods with the label app=Myapp will be routed to.

Helm - programming Kubernetes

[Deploy apps with Helm]

You’ve created a Kubernetes cluster, how do you install Containers, Apps and more?

Genesis Cloud

How to install containers into Kubernetes: Helm

Helm is a tool that streamlines installing and managing Kubernetes applications. Think of it like apt/yum/homebrew for Kubernetes.

Helm does this via Kubernetes charts. Charts are packages of pre-configured Kubernetes resources (yaml to template GTPL).

Use Helm to:

- Find and use popular software packaged as Kubernetes charts
- Share your own applications as Kubernetes charts
- Create reproducible builds of your Kubernetes applications
- Intelligently manage your Kubernetes manifest files
- Manage releases of Helm packages (upgrade etc)

Helm Structure

- Helm has two parts:
  - Helm: A CLI client that lives outside the K8's cluster.
  - Tiller: A CLI server that runs in your K8s cluster as another Pod.
  - Install a chart with '***helm install –f values.yaml mychart***'
  - Upgrade: '***helm upgrade -f values.yaml -f overides.yaml***'
  - Note helm v3 changes this to no Tiller Server due to security risks

- A chart is a collection of files in a directory - the directory name is the name of the chart:

Chart.yaml:

```
cat <<'EOF' > ./Chart.yaml
name: hello-world
version: 1.0.0
EOF
```

Helm Templates:

GO template language
control structures.

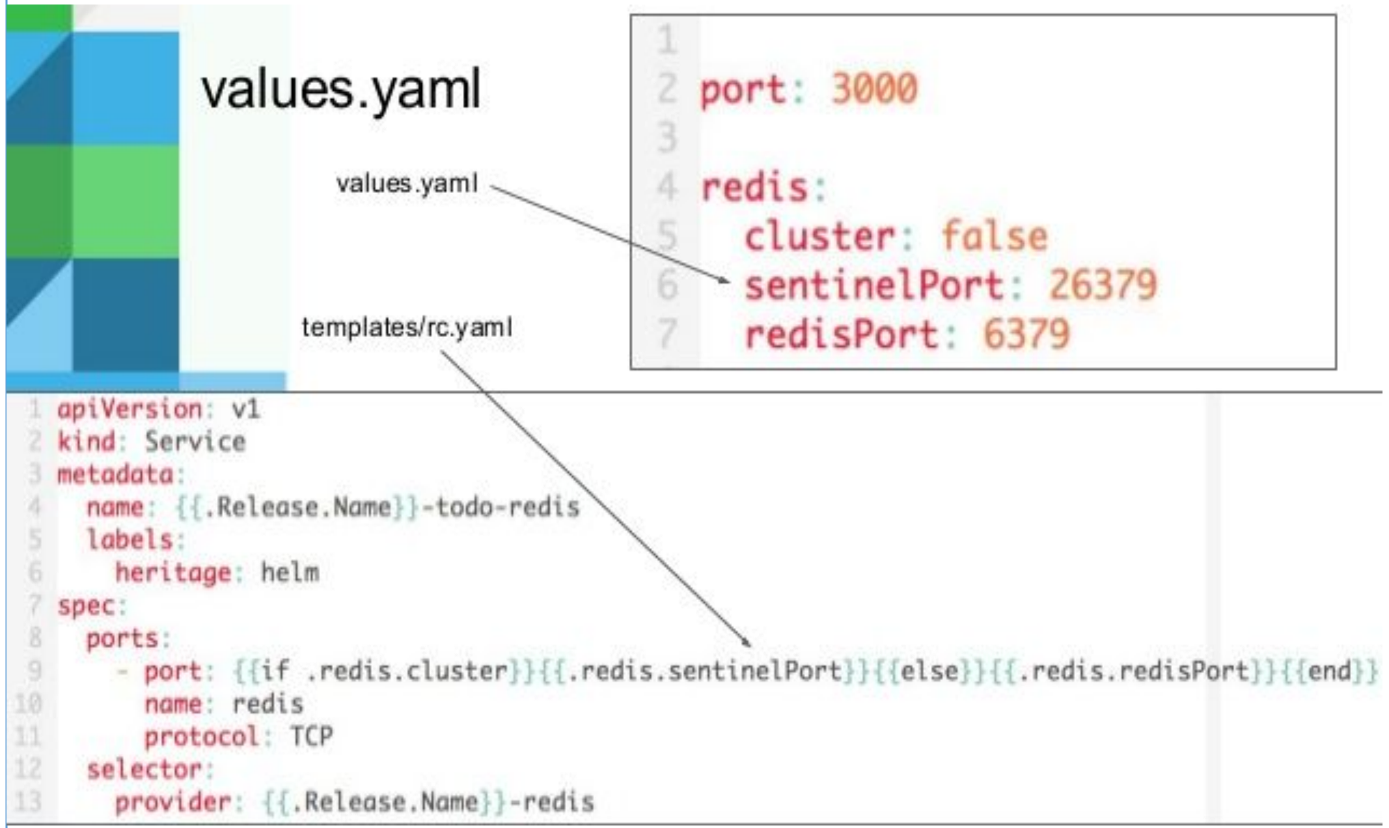
Values.yaml:

How we get values into templates
Simple yaml with namespaces
Override ports for example

Genesis Cloud

Helm in depth

- Install a chart with ‘helm install –f values.yaml mychart’



```yaml
port: 3000

redis:
  cluster: false
  sentinelPort: 26379
  redisPort: 6379
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: {{.Release.Name}}-todo-redis
  labels:
    heritage: helm
spec:
  ports:
    - port: {{if .redis.cluster}}{{.redis.sentinelPort}}{{else}}{{.redis.redisPort}}{{end}}
      name: redis
      protocol: TCP
  selector:
    provider: {{.Release.Name}}-redis
```

- Helm is the Kubernetes Package Manager, and Charts are how you tell Helm what to deploy.
- Charts are packages of pre-configured Kubernetes resources.
- Public Charts easily install apps into your cluster, e.g. OpenStack.
- Or a Private Chart for your application.

Deployers of Kubernetes

All about various deployers

Genesis Cloud

There are so many...

- Kubectl (do it by hand, doesn’t scale, gets wrapped)
- KubeADM (CLI)
- Kubespray (Ansible driven, not stateful, very slow)
- Airship (ATT - single vendor tightly controlled)
- Tectonic (CoreOs - where is it going now?)
- Rancher (Single vendor - Rancher labs)
- Kismatic (Single vendor Apprenda)
- OpenShift (RedHat - single vendor, expensive)
- Azure Kubernetes Service (AKS) - (https://azure.microsoft.com/en-us/services/kubernetes-service/)
- Google Container Platform
- Flagship (Charter), helm driven, self-hosted)
- Gardener (SAP) https://gardener.cloud/, self-hosted, k8s as a service
- Magnum (sits on top of OpenStack - complex)
- A billion more including those written by the slides author….

Genesis Cloud

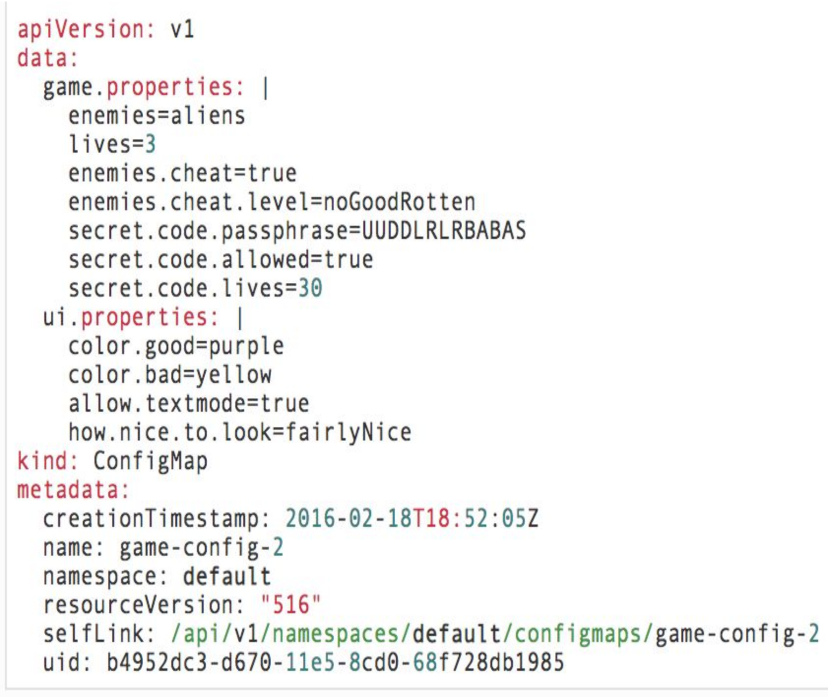
Kubernetes Introduction: ConfigMaps

Configmaps

The ConfigMap API resource provides mechanisms to inject containers with configuration data while keeping containers agnostic of Kubernetes.

```
$ kubectl create configmap game-config-2
--from-file=docs/user-guide/configmap/kubectl/game.properties
--from-file=docs/user-guide/configmap/kubectl/ui.properties
```

```
$ kubectl get configmaps game-config-2 -o yaml
```



```
apiVersion: v1
data:
  game.properties: |
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:52:05Z
  name: game-config-2
  namespace: default
  resourceVersion: "516"
  selfLink: /api/v1/namespaces/default/configmaps/game-config-2
  uid: b4952dc3-d670-11e5-8cd0-68f728db1985
```

Genesis Cloud

Kubernetes Workshop

See you next time :)

Genesis Cloud