Kubernetes Fundmentals

.NET CORE

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services It facilitates both declarative configuration and automation.

https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

What is Kubernetes?

https://developer.ibm.com/technologies/microservices/articles/why-should-we-use-microservices-and-containers/
https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
https://github.com/kubernetes/community/blob/master/contributors/design-proposals/architecture/architecture.md#kubernetes-design-and-architecture

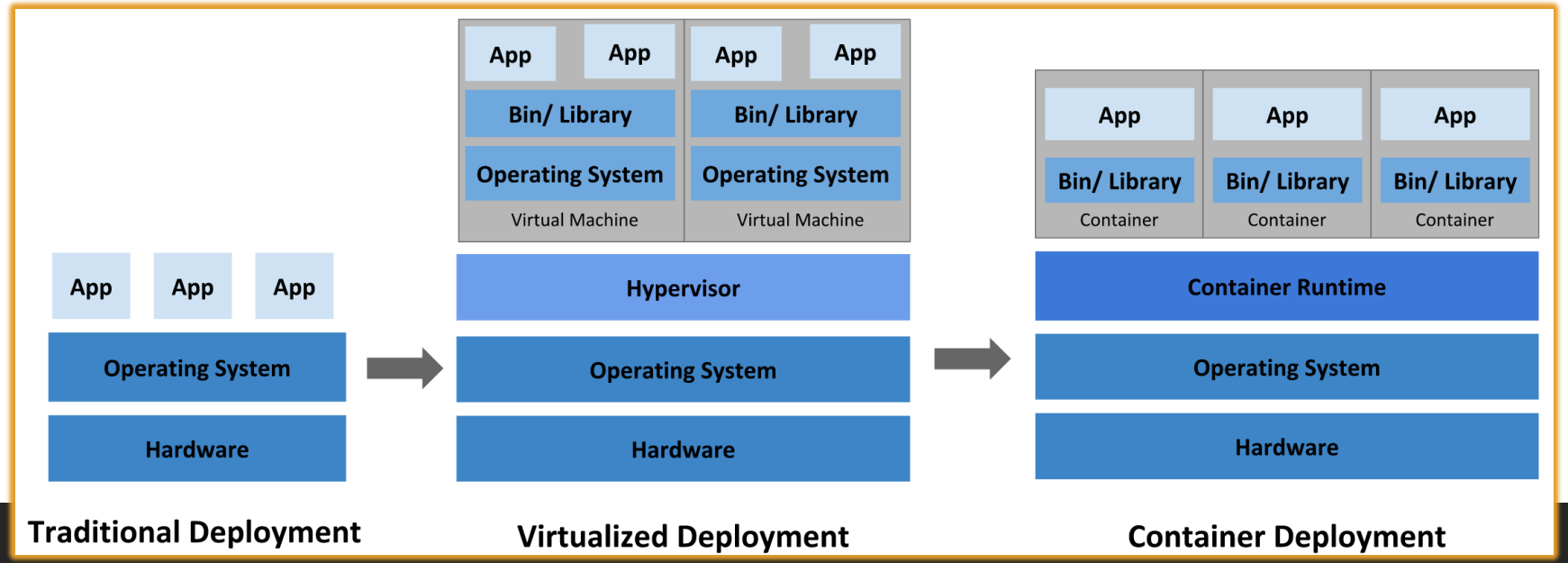
---

Kubernetes is a production-grade, open-source infrastructure for the deployment, scaling, management, and composition of application containers across clusters of hosts. It is inspired by previous work at Google ***Kubernetes project***. The name ***Kubernetes*** originates from Greek, meaning helmsman or pilot.

Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, etc. It allows you to automate the deployment of your containerized microservices. This makes it easier to manage all of the components and microservices of your application.

Containers allow you to:

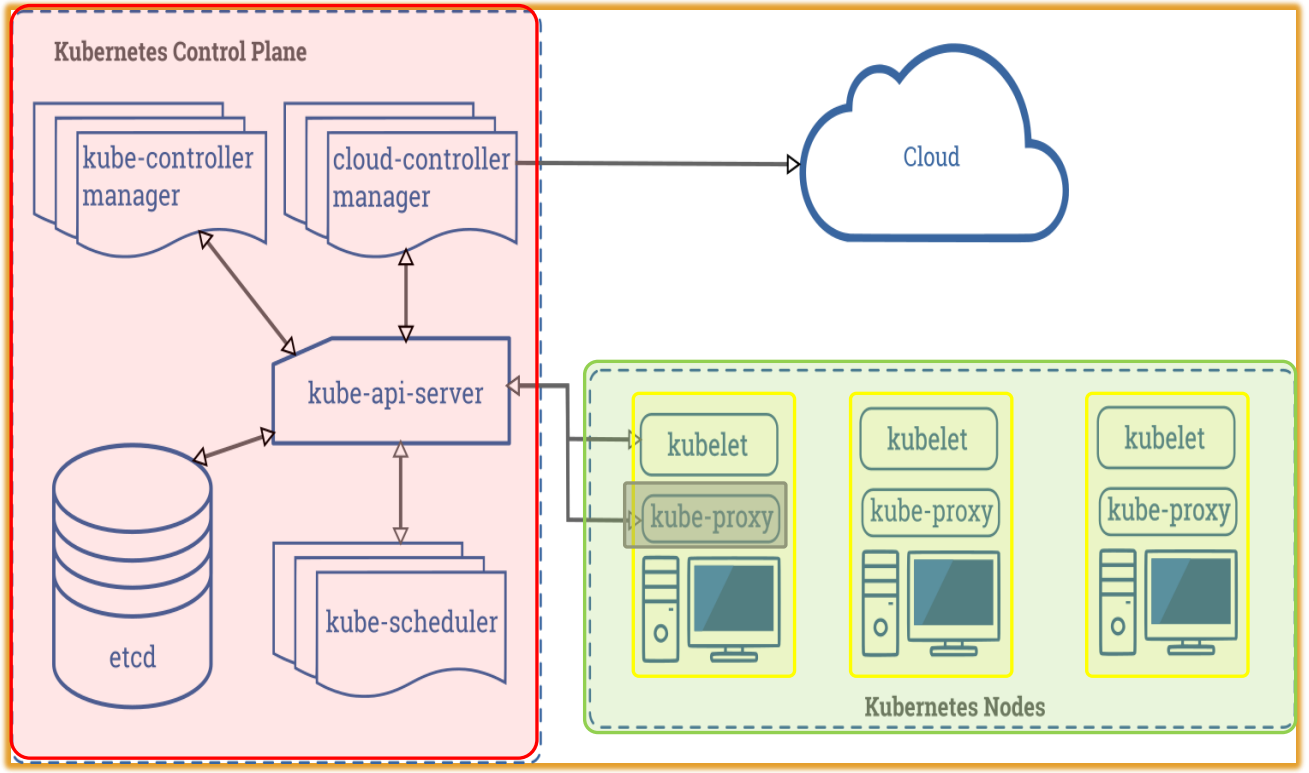
- Deploy images quickly
- Maintain continuous deployment
- Enhance Separation of Concerns
- Run your (portable) application anywhere, on any platform
- Have an elastic, scalable MSA
- Isolate resources
- Utilize resources more effectively



Kubernetes Architecture – Overview

https://kubernetes.io/docs/concepts/overview/components/

- ***Kubernetes*** is not a traditional, all-inclusive PaaS (Platform as a Service) system. ***Kubernetes*** operates at the container level rather than at the hardware level.
- When you deploy ***Kubernetes***, you get a cluster.
- A Cluster consists of worker machines (nodes), that run ***containerized*** applications.
- The worker node(s) host the Pods that are the components of the application workload.
- The control plane manages the worker nodes and the Pods in the cluster.
- In production environments, the control plane usually operates across multiple computers and a cluster usually runs multiple nodes. This provides fault-tolerance and high availability.

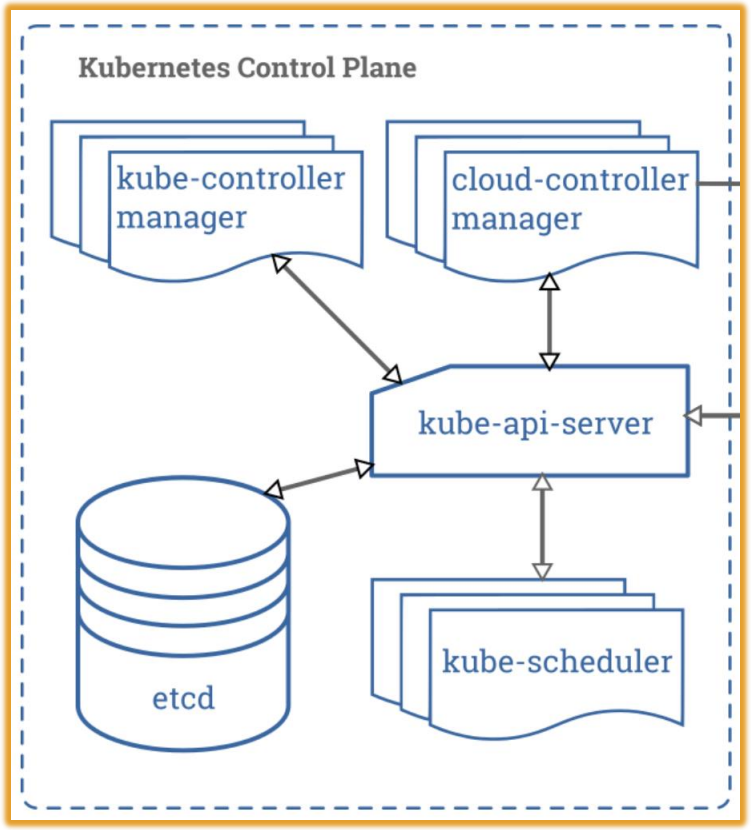


Kubernetes Control Plane (Master)

https://kubernetes.io/docs/concepts/overview/components/#control-plane-components

The ***control plane's*** components make global decisions about the cluster, as well as detecting and responding to ***cluster*** events (for example, starting up a new pod when a deployment's 'replicas' field is unsatisfied).

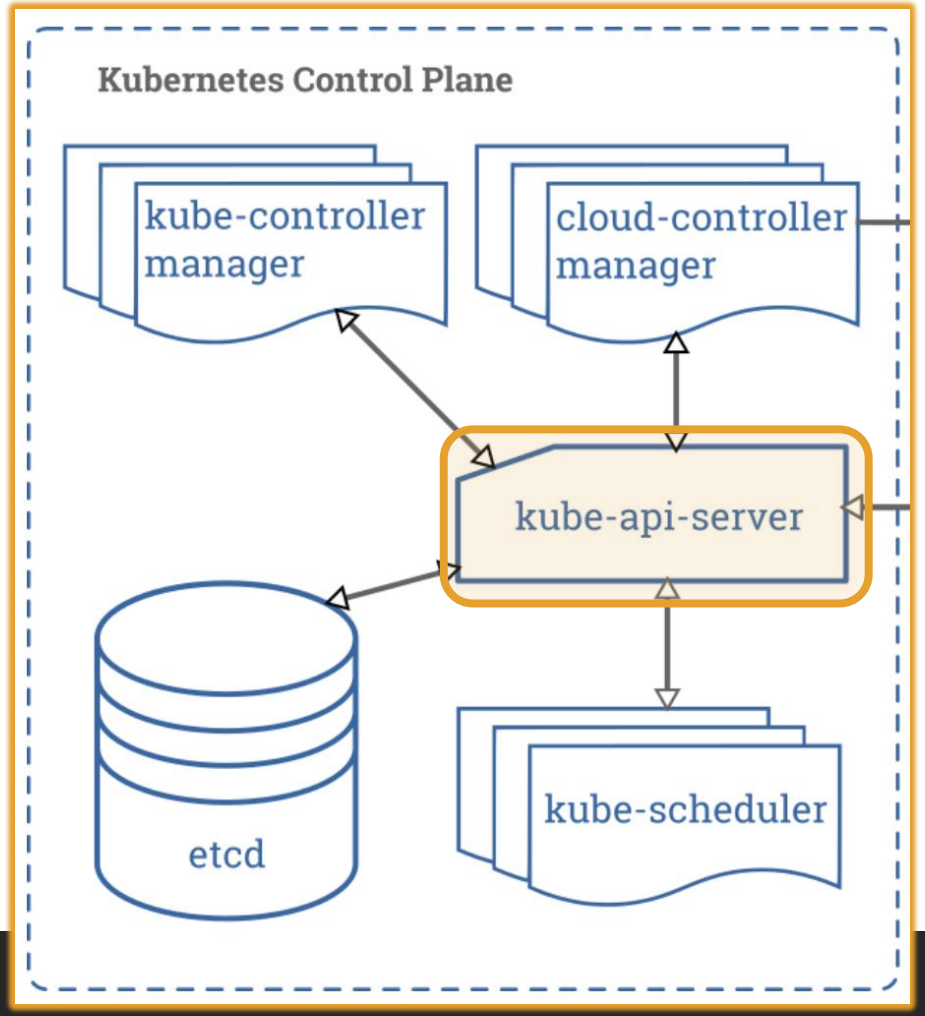
Control plane components can be run on any machine in the cluster, but typically set-up scripts start all ***control plane*** components on the same machine, and do not run user containers on that machine.



Control Plane – kube-apiserver

https://kubernetes.io/docs/concepts/overview/components/#kube-apiserver

- The ***API server*** exposes the Kubernetes API. The API server is the front end for the Kubernetes ***control plane***.
- The main implementation of a Kubernetes API server is ***kube-apiserver***.
- ***kube-apiserver*** is designed to scale horizontally (deploying more instances).
- You can run several instances of ***kube-apiserver*** and balance traffic between those instances.

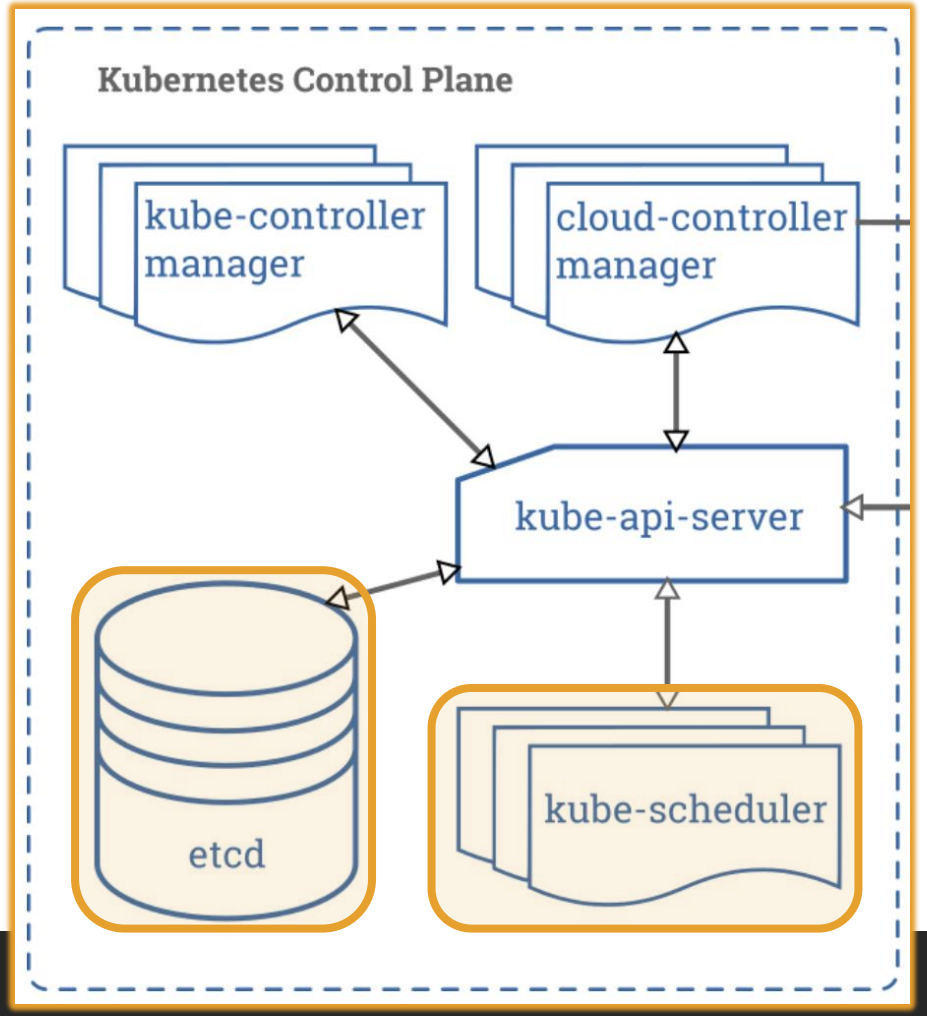


Control Plane = etcd and kube-scheduler

https://kubernetes.io/docs/concepts/overview/components/#etcd
https://kubernetes.io/docs/concepts/overview/components/#kube-scheduler
https://github.com/kubernetes/community/blob/master/contributors/design-proposals/architecture/architecture.md#scheduler

---

- ***Etcd*** is a key-value store. It maintains all the ***clusters'*** data.
- ***kube-scheduler*** watches for new ***Pods*** and assigns a ***node*** to them to run on based on predetermined requirements like:
  - hardware constraints,
  - affinity/anti-affinity specifications,
  - deadlines, and many more.

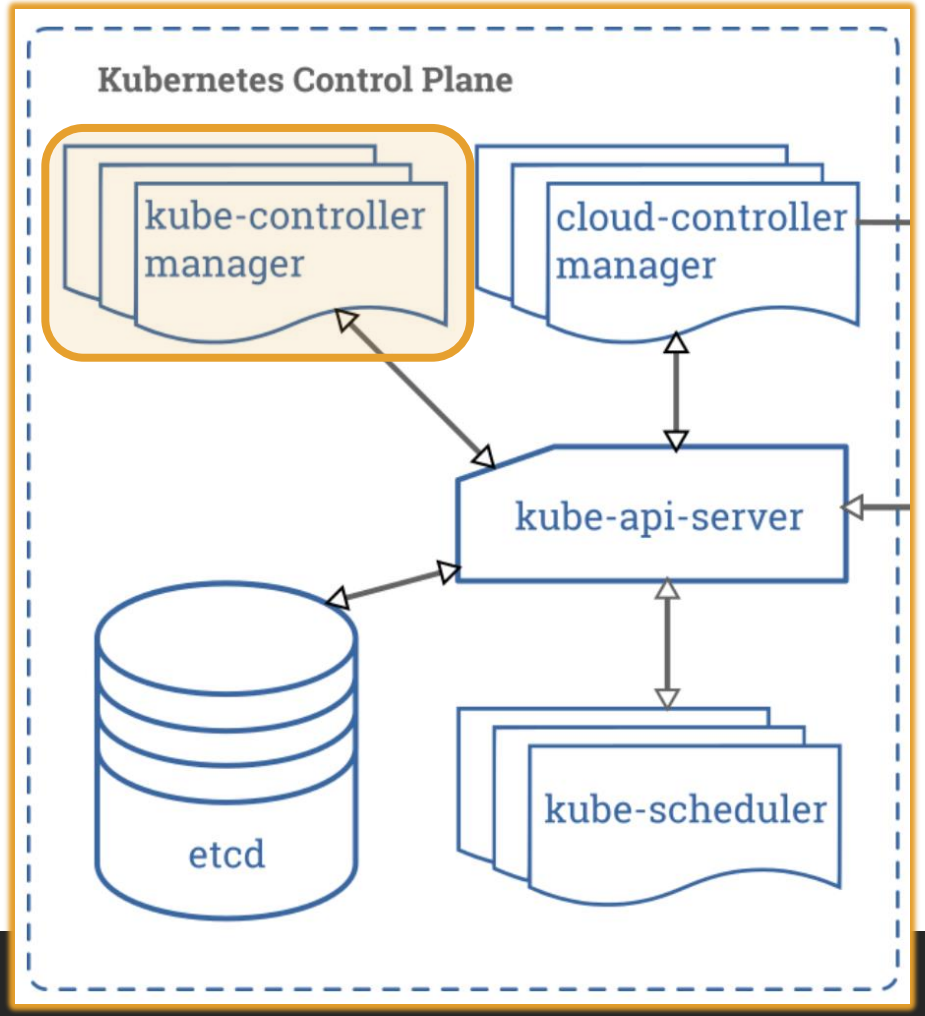


Control Plane – kube-controller-manager

https://kubernetes.io/docs/concepts/overview/components/#kube-controller-manager

Kube-manager-controller runs the ***controller processes***. There are 4 ***controllers***:

- Node controller: notices and responds when nodes go down.
- Replication controller: maintains the correct number of pods for every replication controller object in the system.
- Endpoints controller: Populates the Endpoints object (joins Services & Pods).
- Service Account & Token controllers: Create default accounts and API access tokens for new namespaces.



Control Plane – cloud-controller-manager

https://kubernetes.io/docs/concepts/overview/components/#cloud-controller-manager

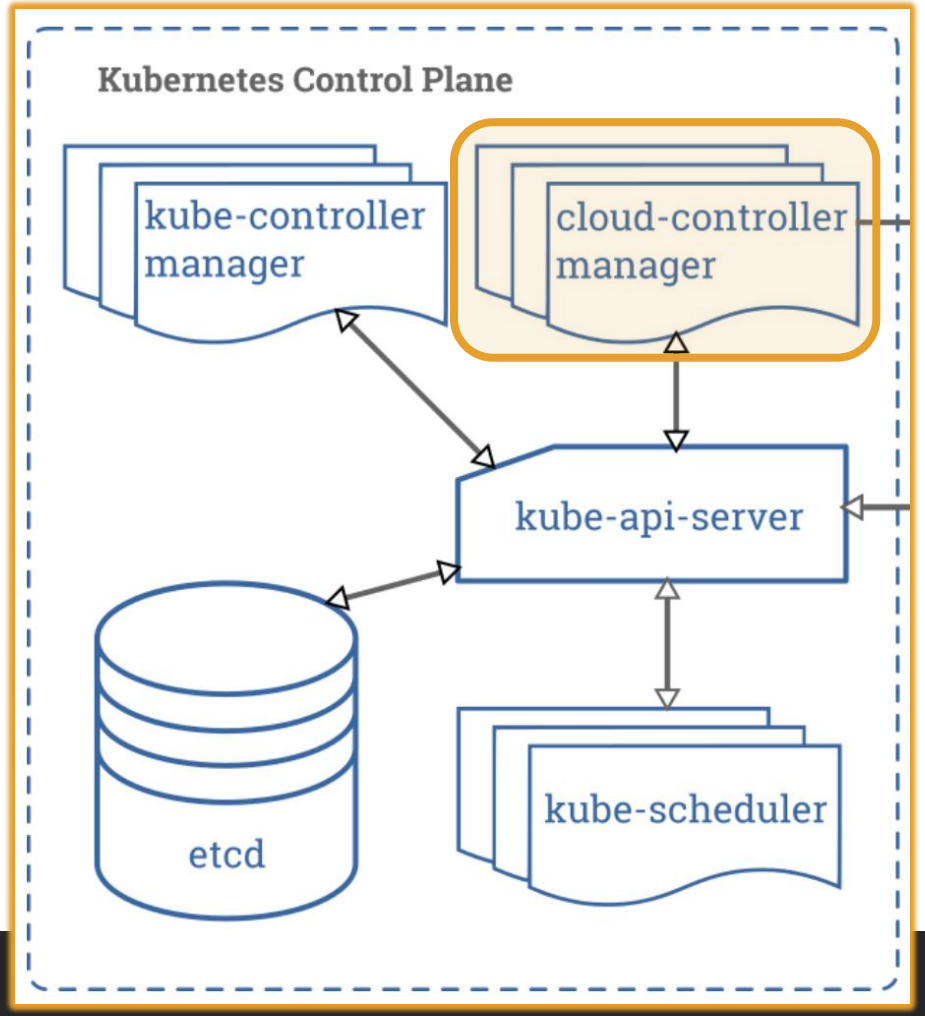
The ***cloud-controller-manager*** embeds cloud-specific control logic. It allows you to link your cluster into your cloud providers API.

The ***cloud-controller-manager*** will separate the components that interact with your cloud platform from components that only interact with your cluster.

cloud-controller-manager combines several logically independent control loops into a single binary that you run as a single process. You can scale horizontally (run more than one copy) to improve performance or to help tolerate failures.

The following controllers can have cloud provider dependencies:

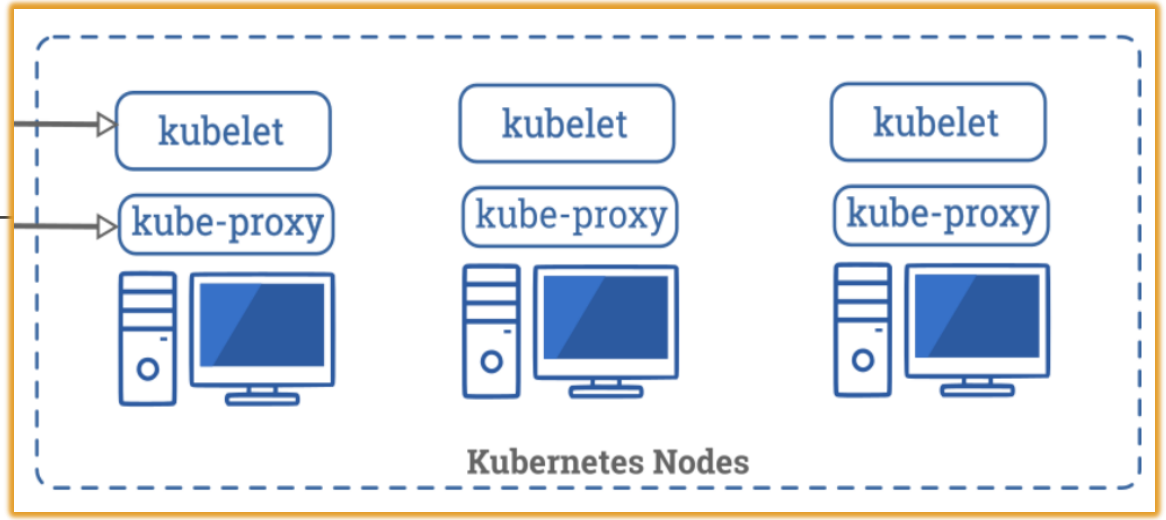
- Node controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
- Route controller: For setting up routes in the underlying cloud infrastructure
- Service controller: For creating, updating and deleting cloud provider load balancers.



Node(Worker) – Components

https://kubernetes.io/docs/concepts/overview/components/#node-components
https://github.com/kubernetes/community/blob/master/contributors/design-proposals/architecture/architecture.md#kubelet
https://github.com/kubernetes/community/blob/master/contributors/design-proposals/architecture/architecture.md#kube-proxy

Node components run on every ***node***, and maintain running ***pods***, and providing the Kubernetes runtime environment.



- A ***Kubelet*** agent runs on each ***node*** in the cluster. It is the primary implementer of the ***Pod*** and Node APIs that drive the container execution layer. The ***Kubelet*** uses ***PodSpecs*** to verify that containers described in ***PodSpecs*** are running in the ***Pods***. The ***kubelet*** doesn't manage containers which were not created by ***Kubernetes***.
- A ***kube-proxy*** is a network proxy that runs on each ***node*** in your cluster. ***kube-proxy*** provides a way to group pods under a common access policy (e.g., ***load-balanced***). This creates a virtual IP which clients can access, and which is transparently proxied (forwarded) to the ***pods*** in a Service. Every ***node*** runs a ***kube-proxy*** process. ***Kube-proxy*** programs IpTables rules to trap access to service IPs and redirect them to the correct backends.

Node – Components

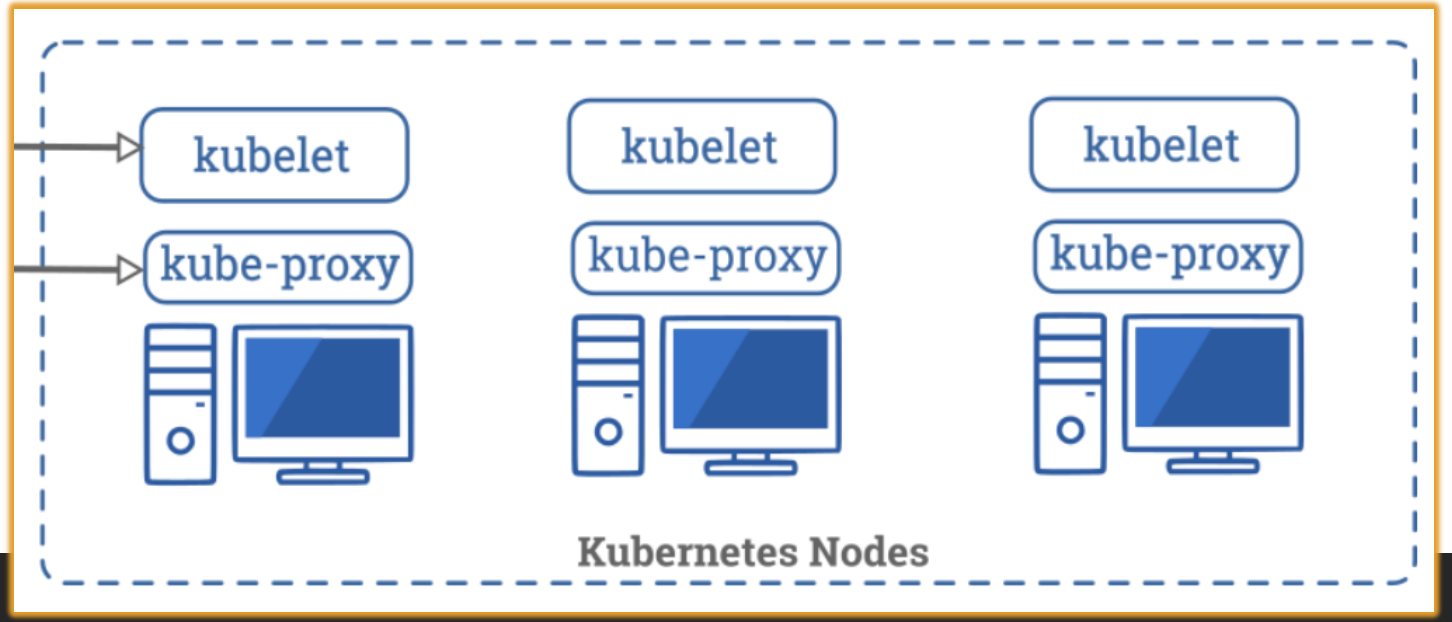
https://kubernetes.io/docs/concepts/overview/components/#node-components
https://kubernetes.io/docs/concepts/architecture/nodes/#management

---

The ***container runtime*** is the software that is responsible for running containers.

Kubernetes supports several container runtimes. Some are:

- Docker
- containerd
- CRI-O
- Kubernetes CRI
  - (Container Runtime Interface).



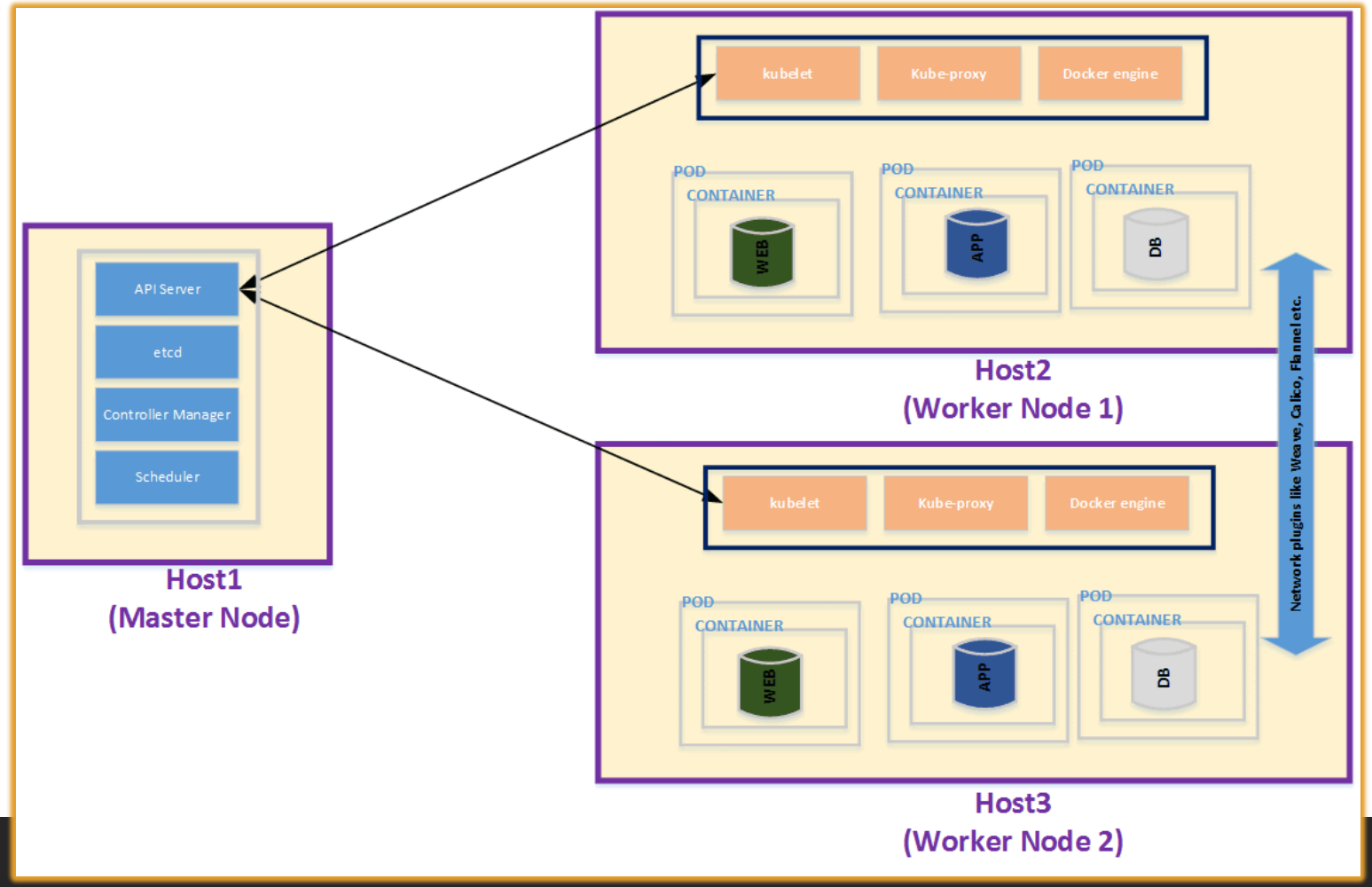
Node Structure

https://kubernetes.io/docs/concepts/architecture/nodes/

- A ***container*** represents your containerized application.
- The ***container*** is placed into a ***Pod***.
- A ***Pod*** runs on a ***Node***.
- One or more ***Nodes*** make up a ***Cluster***.
- The ***Cluster*** is your ***Kubernetes Workload***.

Each ***node*** contains the services necessary to run the ***Pods*** on it, which are managed by the ***control plane***.

A ***node*** may be a virtual or physical machine.



Kubectl (say, “Cube CTL”)

https://kubernetes.io/docs/reference/kubectl/overview/
https://kubernetes.io/docs/tasks/kubectl/install/

Kubectl is the command line tool for controlling Kubernetes clusters. ***Kubectl*** looks for a file named config in the $HOME/.kube directory. You can specify other kubeconfig files by setting the KUBECONFIG environment variable or by setting the --kubeconfig flag.

Kubectl uses the Kubernetes API to interact with the cluster. The following syntax is used in command line to communicate through kubectl:

- kubectl [command] [TYPE] [NAME] [flags]

Command	Usage
[command]	Specifies the operation to perform on one or more resources (create, get, describe, delete.)
[type]	Specifies the (case-insensitive) resource type. Singular, plural, or abbreviated forms can be specified.
[name]	Specifies the name of the resource. Names are case-sensitive. If the name is omitted, details for all resources are displayed.
[flags]	Specifies optional flags.

Deployment

https://kubernetes.io/docs/tutorials/hello-minikube/

A Deployment provides declarative updates for *Pods* and *ReplicaSets*.

Describe a desired state in a *Deployment YAML*, and the *Deployment Controller* changes the actual state to the desired state at a controlled rate. You can define *Deployments* to create new *ReplicaSets*, or to remove existing *Deployments* and adopt all their resources with new *Deployments*.

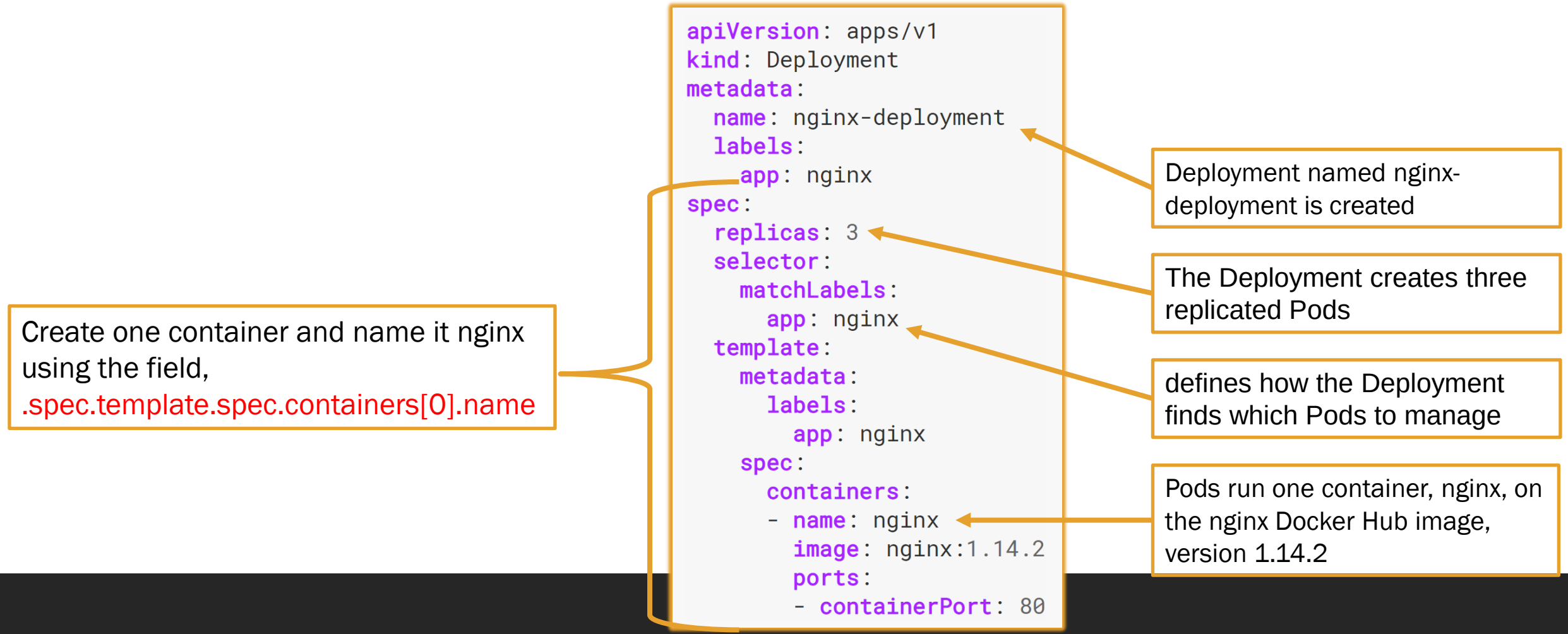
A Kubernetes *Deployment* checks on the health of your *Pod* and restarts the *Pod's* Container if it terminates.

Deployments are the recommended way to manage the creation and scaling of *Pods*. A *Deployment* is a higher-level concept that manages *ReplicaSets* and provides declarative updates to *Pods* along with a lot of other useful features. Therefore, we recommend using *Deployments* instead of directly using *ReplicaSets*, unless you require custom update orchestration or don't require updates at all.

```yaml
apiVersion: apps/v1
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:1.14.2
        ports:
        - containerPort: 80
```

Deployment

https://kubernetes.io/docs/tutorials/hello-minikube/



ReplicaSet

https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

It is recommended to use ***Deployments*** instead of directly using ***ReplicaSets***, unless you require custom update orchestration or don't require updates at all.

A ***ReplicaSet's*** purpose is to maintain a stable set of replica ***Pods*** running at any given time. As such, it is often used to guarantee the availability of a specified number of identical ***Pods***.

A ***ReplicaSet*** will dynamically drive the ***cluster*** back to the predetermined desired state via creation of new ***Pods*** to keep your application running.

Use kubectl apply -f [URL] to apply a template.

Pods can be added without a template also.

A ReplicaSet is defined with fields, including:

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

replicas indicating how many Pods it should be maintaining

selector that specifies how to identify Pods it can acquire

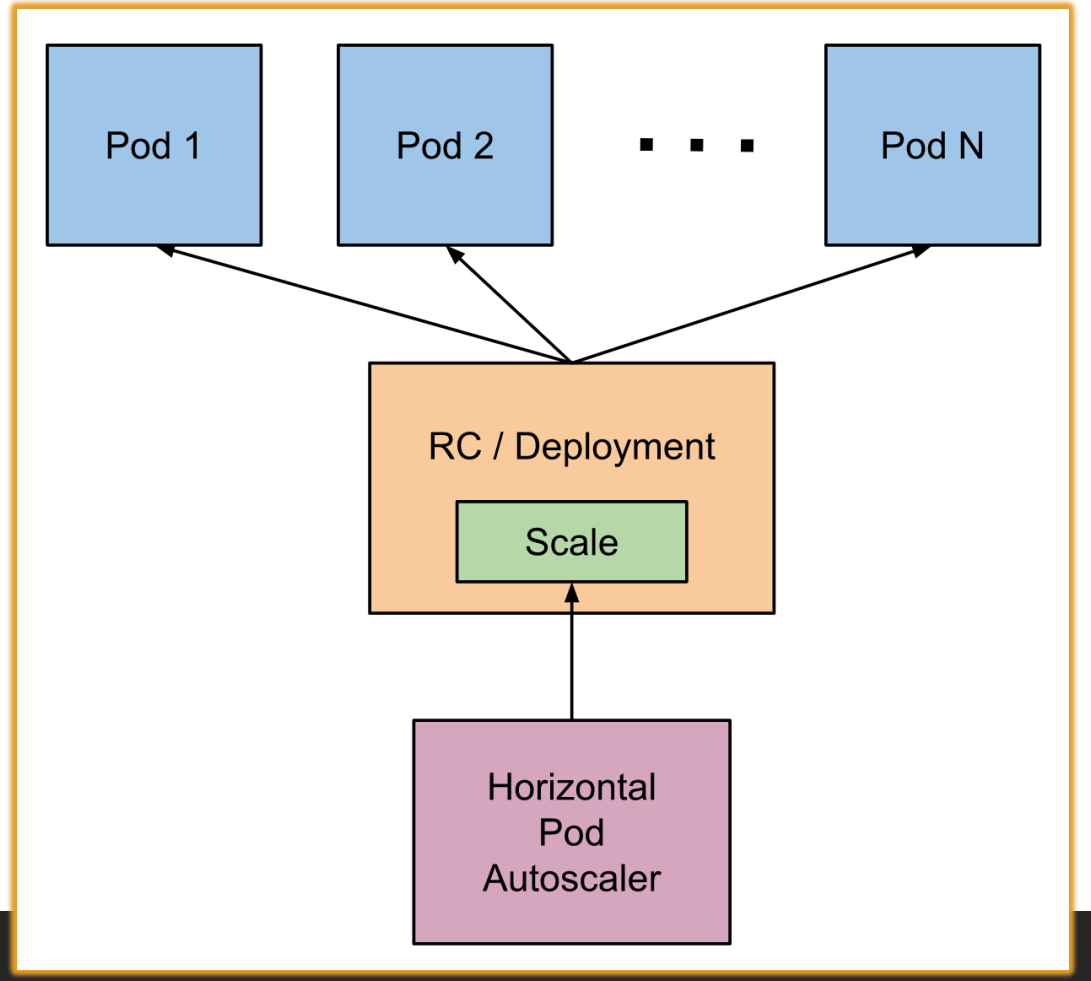
pod template specifying the data of new Pods it should create

AutoScaling

https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

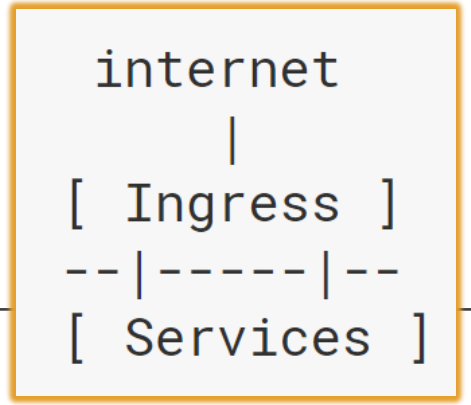
The ***Horizontal Pod Autoscaler*** is an API resource in the Kubernetes autoscaling API group which automatically scales the number of ***pods*** in a ***replication controller***, ***deployment***, ***replica set*** or ***stateful set*** based on observed CPU utilization.

The ***Horizontal Pod Autoscaler*** is implemented as a Kubernetes API resource and a controller in a control loop, with a period controlled by the controller manager. The resource determines the behavior of the controller.



Ingress

https://kubernetes.io/docs/concepts/services-networking/ingress/



The word **Ingress** refers to the right to enter a property.

Ingress in Kubernetes can be configured to give ***Services*** externally-reachable URLs, load balance traffic, terminate SSL/TLS, and offer name based virtual hosting. It exposes HTTP(S) routes from outside the ***cluster*** to services within the ***cluster***.

Traffic routing is controlled by rules defined on the ***Ingress resource***. An ***Ingress Resource*** is (usually) a YAML file defining the rules for data accessing structures in a ***cluster***.

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: test-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          serviceName: test
          servicePort: 80
```

Ingress

https://kubernetes.io/docs/concepts/services-networking/ingress/

As with all other Kubernetes resources, an Ingress needs ***apiVersion***, ***kind***, and ***metadata*** fields.

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: test-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          serviceName: test
          servicePort: 80
```

Each HTTP ***rule*** contains:

1. An optional ***host***. Here, no ***host*** is specified, so the rule applies to all inbound HTTP traffic through the IP address specified.
2. A list of ***paths***
3. A ***backend*** defined for each ***path*** with a ***serviceName*** and ***servicePort***. A backend is a combination of ***Service*** and ***port*** names.

~~The name of an~~ Ingress object must be a valid DNS subdomain name

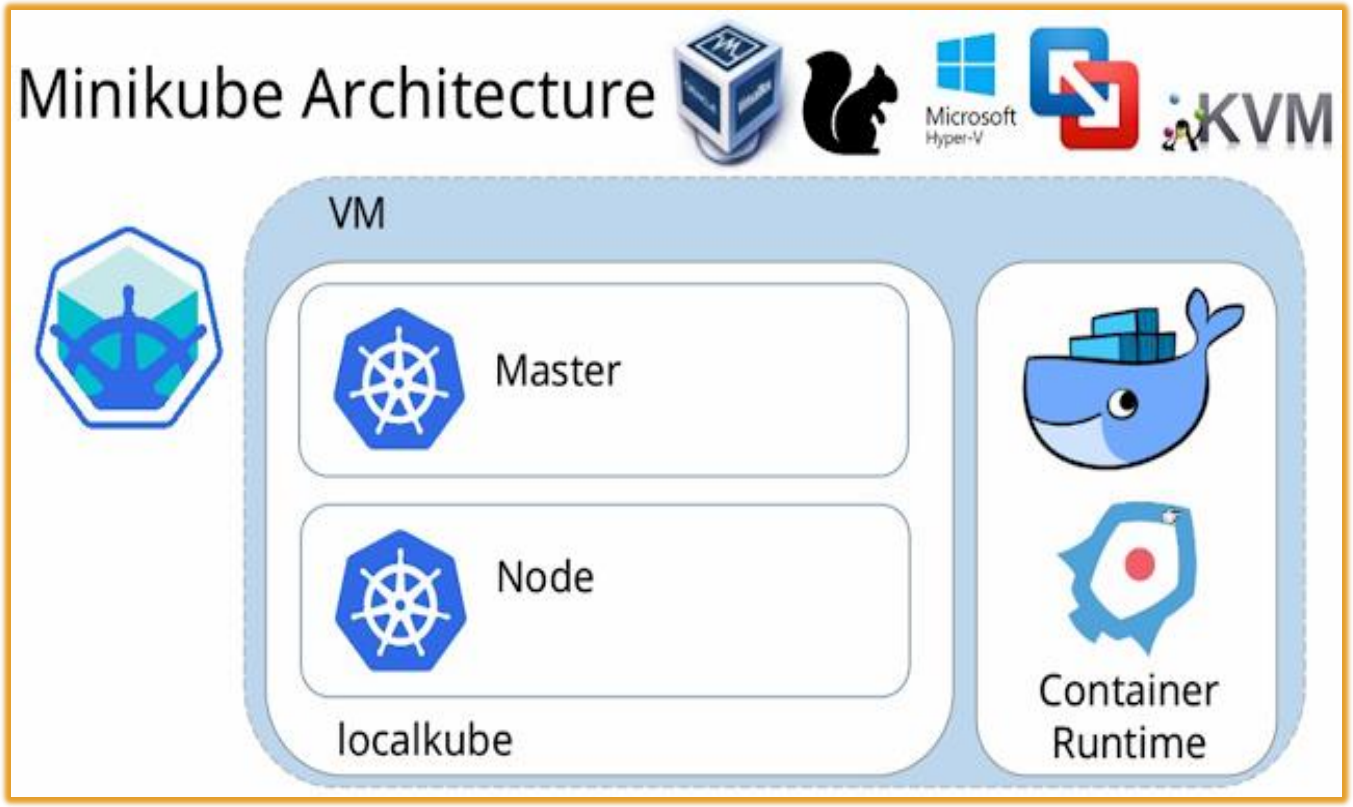
Ingress frequently uses annotations to configure some options depending on the Ingress controller

The Ingress spec has all the information needed to configure a load balancer or proxy server.

MiniKube

https://kubernetes.io/docs/setup/learning-environment/minikube/
https://kubernetes.io/docs/tasks/tools/install-minikube/

- To get started with Kubernetes development, you can use ***Minikube***.
- ***Minikube*** is a lightweight Kubernetes implementation that creates a VM on your local machine and deploys a simple ***cluster*** containing only one node.
- Minikube is available for Linux, macOS, and Windows systems.
- The ***Minikube CLI*** provides basic bootstrapping operations for working with your cluster, including ***start***, ***stop***, ***status***, and ***delete***.



AKS (Azure Kubernetes Service)

https://kubernetes.io/docs/setup/production-environment/turnkey/azure/#azure-kubernetes-service-aks
https://github.com/Azure/aks-engine/blob/master/docs/tutorials/README.md
https://docs.microsoft.com/en-us/azure/aks/intro-kubernetes

- The Azure Kubernetes Service (AKS) offers simple deployments for Kubernetes clusters.
- AKS makes it simple to deploy a managed Kubernetes cluster in Azure.
- AKS handles much of the complexity and operational overhead of managing Kubernetes.
- Azure handles critical tasks like health monitoring.
- The Kubernetes masters are managed by Azure. You only manage and maintain the agent nodes.
- AKS lets you integrate with Azure Active Directory and use Kubernetes role-based access controls.

Assignment

https://kubernetes.io/docs/tutorials/kubernetes-basics/

Create 7 quiz questions along with four plausible answers to go along with the 6 chapters of the linked tutorial and the lecture pdf. That means one question from each chapter and 1 from the pdf from each associate.

These questions will be included in a Google Forms quiz that you will take tomorrow morning.

This will serve as a primer for QC on Tuesday, so make sure to include a full range of possible questions from definitions of structural elements to process to data flow, to specific commands in the CTL.

Hello-Node Tutorial SbS(1/2)

https://kubernetes.io/docs/tutorials/hello-minikube/

- Create a Deployment that manages a Pod which will run a container based on the provided Docker Image with
  - kubectl create deployment hello-node –image=k8s.gcr.io/echoserver:1.4
- See the deployment with
  - kubectl get deployments.
- See the Pod with
  - kubectl get pods.
- See cluster events with
  - kubectl get events.
- See the kubectl configuration with
  - kubectl config view.
- The Pod is default only accessible byt an internal IP inside the Cluster. Expose the ***Pod*** as a Kubernetes ***Service*** to make is visible from outside the ***Cluster*** with the type=LoadBalancer is the expose keyword.
  - kubectl expose deployment hello-node --type=LoadBalancer --port=8080. (more on the next slide.)

Hello-Node Tutorial SbS(2/2)

https://kubernetes.io/docs/tutorials/hello-minikube/

- View the service you just created with
  - kubectl get services.
- External cloud providers get an external IP to access the service,
- Select + → Select Port to View on Host 1 → Enter the 5 digit port # after the :
- Take a look at available Add-Ons with
  - minikube addons list.
- Enable the metrics-server add-on with
  - minikube addons enable metrics-server.
- View the Pod with the service you just created with
  - kubectl get pod, svc –n kube-system.

- Disable the metrics-server with
  - minikube addons disable metrics-server.
- Delete the service with
  - kubectl delete service hello-node
- Delete the deployment with
  - kubectl delete deployment hello-node
- (optional) stop Minikube with
  - minikube stop
- (optional)Delete the Minikube VM with
  - minikube delete

Tutorials

https://docs.microsoft.com/en-us/azure/aks/tutorial-kubernetes-prepare-app

https://kubernetes.io/docs/tutorials/stateful-application/mysql-wordpress-persistent-volume/