Kubernetes Training

Yashesh Mankad

kubernetes

Agenda

Course Content

- ★ Day 1
  - Container Fundamentals
  - K8s Fundamentals, Architecture, kubectl
- ★ Day 2
  - Cluster, Pods, Service, ReplicaSets
- ★ Day 3
  - Deployments, Rolling Update, Namespaces, ConfigMaps
  - Hackathon
- ★ Day 4
  - Persistent Storage, Volume, Claim, StatefulSets
  - Container Resources, Auto Scaling, Ingress
- ★ Additional Topics
  - K8s plugins, CSI
  - Probes, Secrets, Jobs

Container Fundamentals

Container Fundamentals

- A container provides compute level abstraction across applications
- Containers share the operating system's kernel with other containers
- Since they don't contain an entire OS, its resource footprint is tiny and boots up instantly
- Container resource consumption is a fraction of VM resources
- Hundreds of containers can run within a single VM/server
- cgroups provided the first level of abstraction in the Linux kernel in 2007
- LXC (LinuX Containers) was the first complete container manager in 2008
- Container Runtime Interface (CRI) is the accepted standard for containers
- Docker containers is the most popular implementation of CRI since 2015

Container Fundamentals

- Like VM images, there are container images. Container images can be tagged for versioning. Default tag is `latest`
- Container images are defined declaratively through Dockerfiles
- During container creation, the tag can be optional specified to pick a specific version. If not specified, default tag is `latest`
- Containers heavily restrict user access by default (because the runtime is shared with other containers). Privileged mode has to be enabled during container creation for basic OS requests like mount/umount
- Following cheat sheet is a good reference for Docker commands: https://dockerlabs.collabnix.com/docker/cheatsheet/
- Docker Desktop on Mac is a great sandbox to get going!

Elastic Cloud is a Mindset Change

bowzer.company.com
(scale-up)

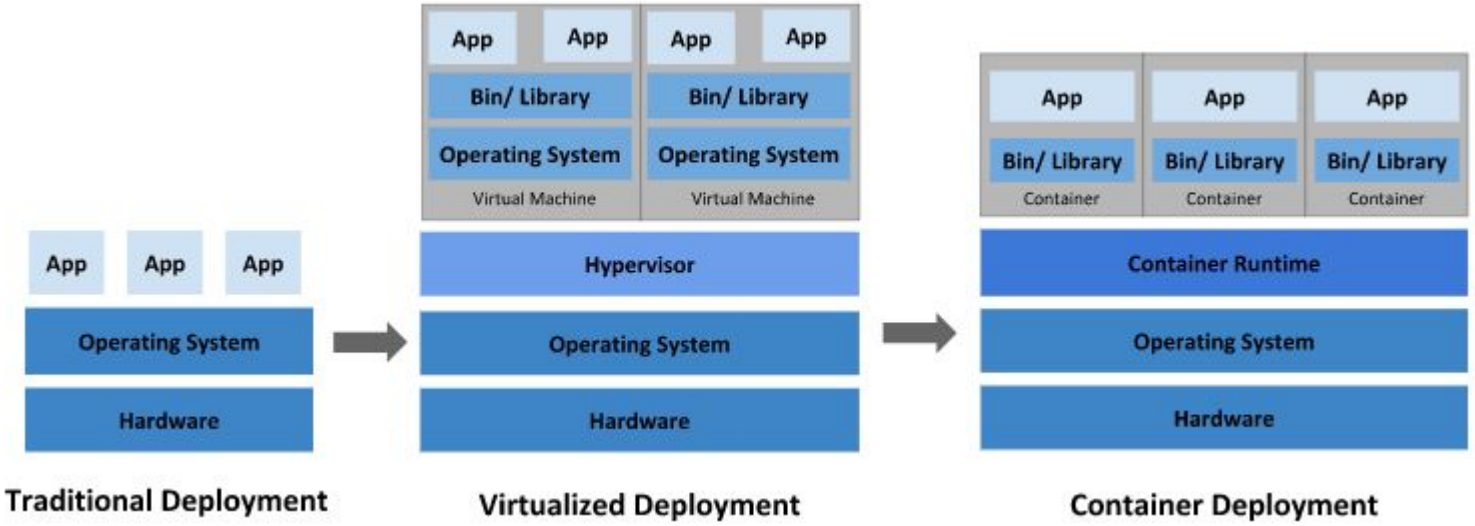
web001.company.com
(scale-out)

Kubernetes Fundamentals

Kubernetes Fundamentals

- Open-source container orchestration and service management platform. Highly extensible
- Originated as @ Google in 2003 to manage Search. Was overhauled in 2014 to work with Docker containers and renamed to Kubernetes
- Kubernetes 1.0 was launched in 2015 under the Cloud Native Computing Foundation (CNCF)
- It is now the de-facto standard for containers and microservices
- Other similar orchestrators: Mesos, Docker Swarm, Nomad, etc
- Supports access via CLI, API and SDK

Kubernetes Fundamentals



Deploy Kubernetes Cluster

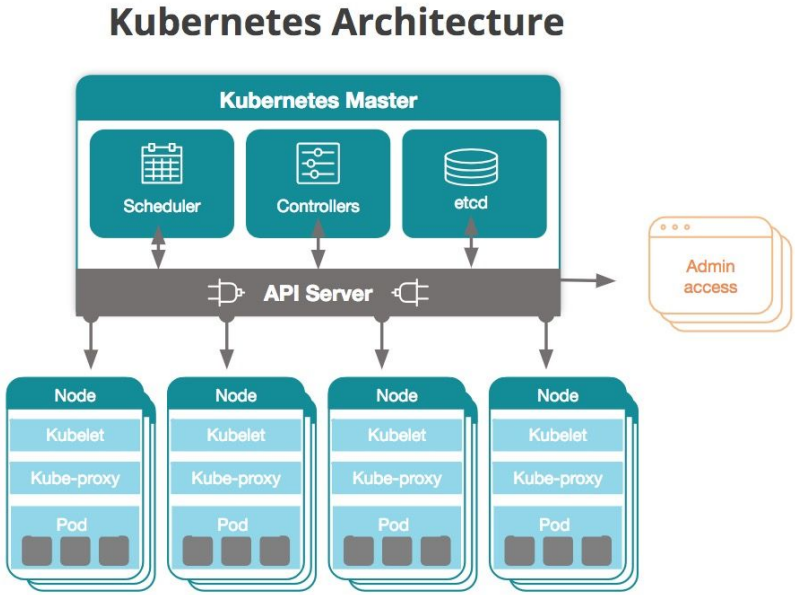
- Kubernetes clusters can be manually installed on a host/VM or accessed via managed services like Karbon, EKS, etc
- Docker Desktop for Mac offers a single node K8s cluster
- Preferences -> Resources: 2 cores, 8GB RAM, 2GB swap, 32GB storage
- Get the training material:
  git clone https://github.com/yashmankad/k8s-training.git

Kubernetes Architecture

Kubernetes Architecture

★ Kubernetes Master
- kube-apiserver
- etcd
- kube-scheduler
- kube-controller-manager

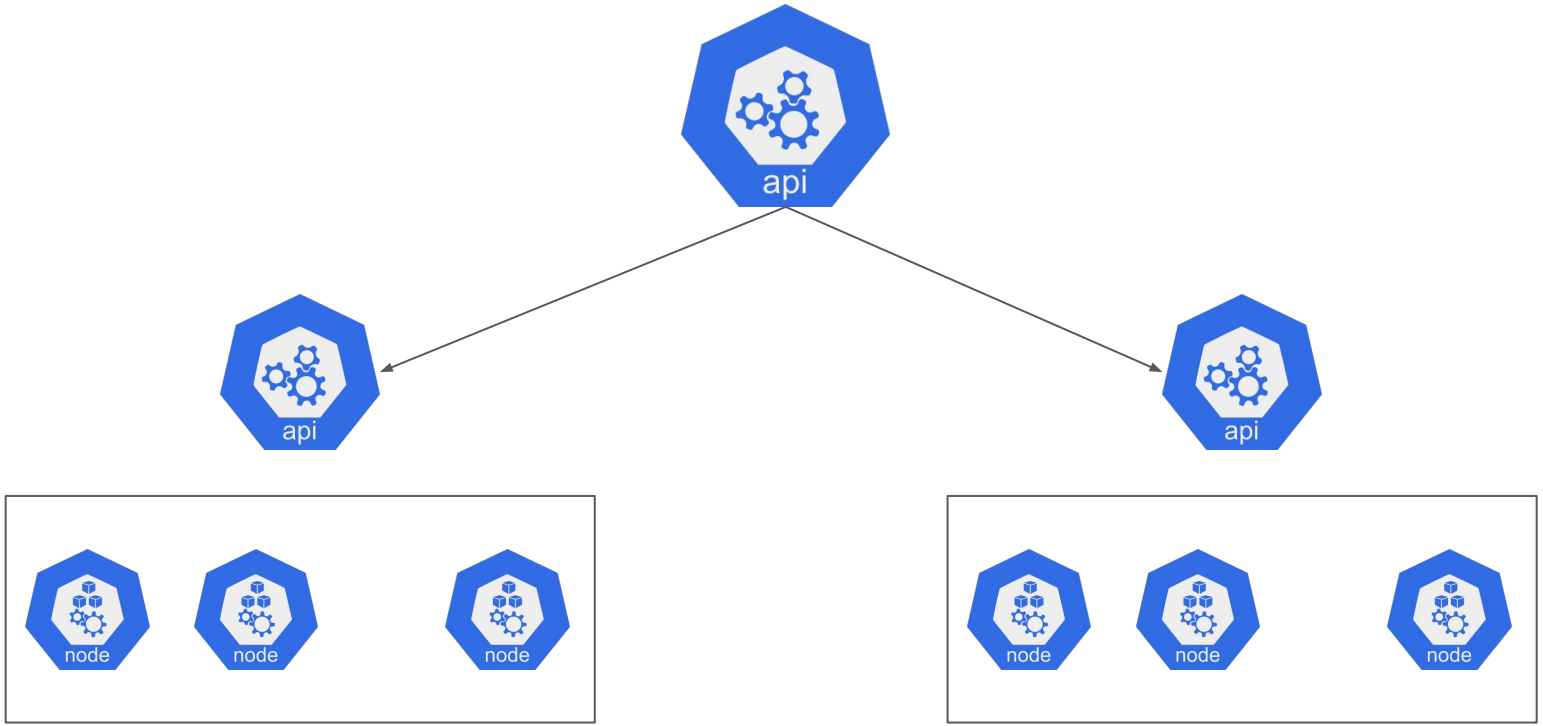
★ Kubernetes Worker Node
- kubelet
- kube-proxy



Kubernetes Architecture

- kube-apiserver: API server for internal and external access to K8s
- etcd: consistent and highly-available KV store used as Kubernetes' backing store. Generally deployed as a separate cluster
- kube-scheduler: Scheduler for Pods, containers and other workloads
- kube-controller-manager: component that runs all the control loops
  - Node Controller: detecting and responding to nodes going down
  - Replication Controller: maintains correct replica count for Pods
  - Endpoints Controller: connects Pods with Services
  - Service Account & Token Controller: Manages user accounts, access token and namespaces
- kubelet: This is a node agent. Manages Pods on that node
- kube-proxy: network proxy that ensures Pod communication inside & outside the cluster
- Container Runtime: Docker, containerd
- Kubernetes supports multi-master and cluster federation architectures (kubeadm)

Kubernetes Federation



Kubernetes Architecture

- Kubernetes offers a declarative paradigm of programming through YAML
- Users create resource templates or manifests that can be applied to a K8s cluster and reused
- Sample resource template:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
  - name: nginx-container
    image: nginx
```

Kubernetes CLI (kubectl)

- Kubernetes offers CLI access to a cluster via ***kubectl***
- Kubectl is the most common way to access a cluster and deploy K8s resources
- Common format: *kubectl <verb> <resource> <resource-name>*
- Examples:
  - Get K8s cluster info: *kubectl get cluster-info*
  - Apply a template: *kubectl apply -f <filename.yaml>*
  - Fetch a Service: *kubectl get service <service name>*
  - Describe a Pod: *kubectl describe pod <pod name>*
  - Fetch console logs from a Pod: *kubectl logs <pod name>*
- kubectl cheat sheet: https://kubernetes.io/docs/reference/kubectl/cheatsheet/

Pods

Pods

- A Pod is the smallest building block in K8s
- It encapsulates a container. In some rare cases, more than one container
- Users rarely deal with Pods. Services and Deployments are more common
- If a Pod dies, it is not automatically re-created unless it was created through a Deployment or ReplicaSet
- Pod access is usually limited to the K8s cluster unless frontended by a Service
- One or more labels can be assigned to a Pod for association with other K8s resources
- Check logs for a Pod: *kubectl logs <pod name>*
- Exec into a Pod: *kubectl exec -it <pod name>*
- Sidecar Pattern: Usually a Pod contains a single container but in some cases an auxiliary container is added to a Pod to perform some routine task, like log shipping or health monitoring. For instance, checkout how Istio monitors service interactions here

Pods Exercise

Services

Services

- Services provide access to Pods outside the K8s cluster
- Service can be associated to one or more Pods through `selectors` and `labels`
- If there are multiple `selectors` for a service, all must match on the Pods
- Clients access the Service and not the Pods directly to withstand Pod failures and restarts providing a stable point of access to Pods/workloads
- Load-balances traffic across Pods associated to the Service
- By default, service limits access with the cluster. A public service can be created through `NodePort` option
- Services are accessed through their DNS name (service name) and not their IP. For instance, a service can be accessed on port 8080 as http://service-name:8080

Services Exercise

ReplicaSets (rs)

ReplicaSets

- ReplicaSets ensure a certain minimum number of replicas (Pods) are always running for an application
- If a Pod dies, ReplicaSet creates a new Pod to ensure minimum cardinality
- Pod IPs and hostnames are not retained if they are recreated
- ReplicaSets are prefered over directly dealing with Pods
- How does it work?
    - Replication Controller on the K8s master ensures minimum number of replicas are always running for a ReplicaSet
    - If the replica count goes down, it creates new Pod records in etcd
    - Scheduler looks at these new records and finds an appropriate node to place the pods
    - kubelet on that node handles actual Pod and container creation
- A ReplicaSet is independent from a Service defined for the Pods. They can coexist and removing one will not remove the other
- Deployments are preferred over ReplicaSets

ReplicaSet Exercise

Deployments

Deployments

- Deployments are an abstraction on top of ReplicaSets
- Offers ability to rollout and rollback new application versions
- Rollout strategy can be provided in the deployment spec under `strategy`
- With a `Recreate` strategy, all old Pods are destroyed before new ones are created. This will cause application downtime
- Deployments and Services are independent, just like with ReplicaSets

Deployment Exercise

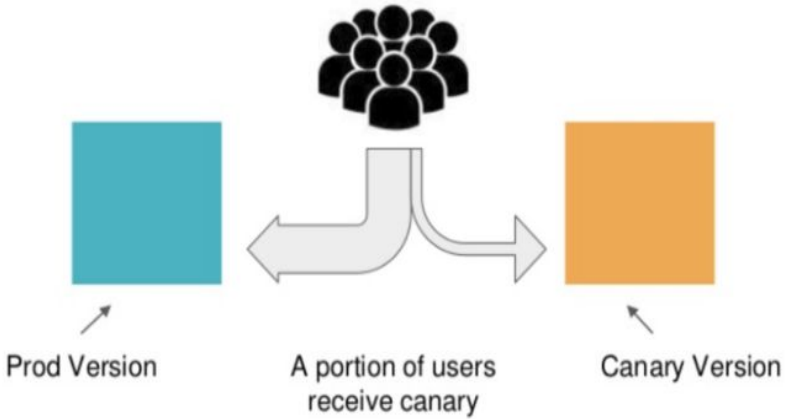
Blue/Green Deployment

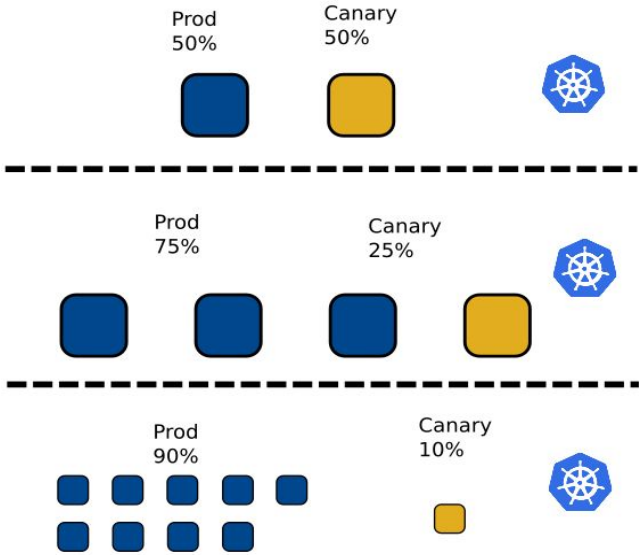
Blue/Green deployments are a common practice on-prem

Blue/Green Deployment Exercise

Canary Deployment

Canary deployments are popular with cloud services





Rolling Update

Rolling Update

- Hitless upgrades are supported by default in K8s. They are realized by managing multiple ReplicaSets under the same Deployment
- How does a rollout work under the hood?
  - A new ReplicaSet is created for the new rollout with the new Pod configuration and replica count as zero
  - The Replicas and Pod count will increase steadily for the new ReplicaSet
  - Once newer pods start launching, the older replicas/Pods will be reduced/terminated
  - This process will continue till the new ReplicaSet has all the replicas and the old ReplicaSet has zero replicas
  - The old ReplicaSet is empty and will hang around (customizable in the Deployment spec)
  - A rollback will reverse this process
- Creates a new ReplicaSet for each version of the application for hitless upgrade
- Rollouts can be paused, resumed or undone (rollback)
- Rollout Status: *kctl rollout undo deployment <deployment-name>*
- Rollback: *kctl rollout undo deployment <deployment-name>*
- Rollout History: *kctl rollout history deployment <deployment-name>*

Rolling Update Exercise

Namespaces (ns)

Namespaces

- Ability to carve out virtual K8s clusters from same physical cluster
- Same K8s cluster can host ‘prod’ and ‘staging’ namespaces
- If not specified, resources go in the `default` namespace
- All K8s services run in the *`kube-system`* namespace
- Namespaces can be created declaratively
  - Fetch all namespaces: *kctl get ns*
  - Fetch a resource in a specific namespace: *kctl get pods -n <my-namespace>*
  - Fetch resources across all namespaces: *kctl get all --all-namespaces*

Namespaces Exercise

Troubleshooting Exercise

ConfigMaps

ConfigMaps

- It is a resource object that holds configuration
- Pod/Containers often need several environment variables to function properly
- These variables are usually externalized for easy reuse and sharing. Which means these variables have to be passed into containers which is messy
- ConfigMaps allow you to decouple configurations at the cluster level and have containers refer to them. Allows configuration reuse
- In addition to storing key/value pairs for environment variables, you can inject files into containers with ConfigMaps

ConfigMap Exercise

Hackathon

kubernetes

Container Storage

Container Storage

- Containers can have two types of storage - ephemeral and persistent
- Ephemeral storage is tied to the container’s lifetime. It is unavailable once the container is gone. Container’s root (‘/’) always goes on ephemeral storage
- Persistent storage is not tied to the container’s lifetime. It can be mounted onto the container and then re-mounted to a different container if needed
- Data is preserved on persistent storage and can be shared across multiple containers
- Persistent storage can be achieved by creating storage volumes during Pod creation or through a first class resource called Persistent Volumes (PVs)

Persistent Volumes (PVs)

StorageClass

- StorageClass is a resource useful for defining different “classes” of storage
- These classes may be created for different quality of service or policies (primary vs. backup) or related to the cluster where applications are running
- For instance, a K8s cluster can have two storage classes: a class backed by NVMe devices for prod and an SSD based class for backup/staging environments
- Users can carve out persistent volumes (PVs) from storage classes which can then be further carved for consumption by Pods (more on this later)
- Users need to choose the following while configuring a StorageClass
    - Storage technology: Underlying storage technology that the class should use (NFS, iSCSI, FC, etc)
    - Provisioner/Provider: Storage vendor specific configuration (AWS EBS, Azure File, vSphere, etc)
    - Reclaim Policy: Determines what should be done with volumes that are no longer used in this StorageClass (delete them, clean then up for reuse or do nothing)

Persistent Volumes (PVs)

- PVs can be created outside the scope of a Pod and used across Pods/Services
- PVs can be provisioned from a certain storage classes based on the desired quality of service. Or a Storage Class can be created at runtime
- If a PV is not attached to a Pod, its status is ‘Available’. If an active Pod is using it, the the status changes to ‘Bound’ and once released by the Pod, it changes to ‘Released’
- Life-cycle of a PV is not associated to a Pod. Users can configure a reclaim policy to determine what happens to the PV once it is released (retain or delete)
- For dynamically created PVs, reclaim policy is inherited from the StorageClass
- If not specified, the StorageClass is ‘default’ backed by the hostpath
- kubectl: *kubectl get pv*

Persistent Volume Claims (PVCs)

Persistent Volume Claims (PVCs)

- PVC is a request for storage by a user that can be used in Pods
- Pods consume node resources and PVCs consume PV resources
- PVC requests specify the desired size of storage and access mode (ReadWriteOnce, ReadOnlyMany and ReadWriteMany)
- For instance, a PV can be provisioned for 10GB and one or more Pods can request storage through a PVC of variable sizes (< 10GB)
- Like PVs, PVCs are also managed as a first-class object and outside the scope of a Pod
- Storage features for PVCs depend on capabilities offered by the underlying storage vendor (resizing, snapshotting, clone, etc)
- Kubectl: *kubectl get pvc*

StatefulSets

StatefulSets

- StatefulSets provide ordered, graceful deployment and scaling of Pods
- Each Pod is uniquely identified by a sticky identifier that does not change even if the Pod is recreated
- Rolling updates and rollbacks are also ordered
- Only PVs/PVCs can be used for storage with StatefulSets and the PV/PVC is not deleted if the Pod is removed (when the StatefulSet is scaled down)
- Stickiness is maintained for the persistent storage attached to the Pods
- Pod IPs are not sticky - IPs can change if a Pod is recreated. As a result, applications use DNS names (pod name) instead of IPs for communication
- Scaling down a StatefulSet is ordered but deleting it entirely is not
- Parallel Pod management is possible through additional config
- Requires a headless service as part of its deployment

PV, PVCs and StatefulSet Exercise

Container Resources

Container Resources

- Containers can request minimum number of desired resources
- Most common resources requested are CPU, memory and storage
- A container can consume more than its minimum requirement of resources if available upto a certain maximum (if specified)
- It is recommended to set upper bounds on resources (`limits`)
- During Pod creation, scheduler picks a K8s node that satisfies all the minimum resources requested by the container
- If a container exceeds its resources, the Pod can be evicted/destroyed
- Evicted Pods are not recreated unless deployed through a ReplicaSet
- Container resources can be viewed via:
  - *kubectl describe pod <pod-name>*
  - *kubectl top*

Container Resource Exercise

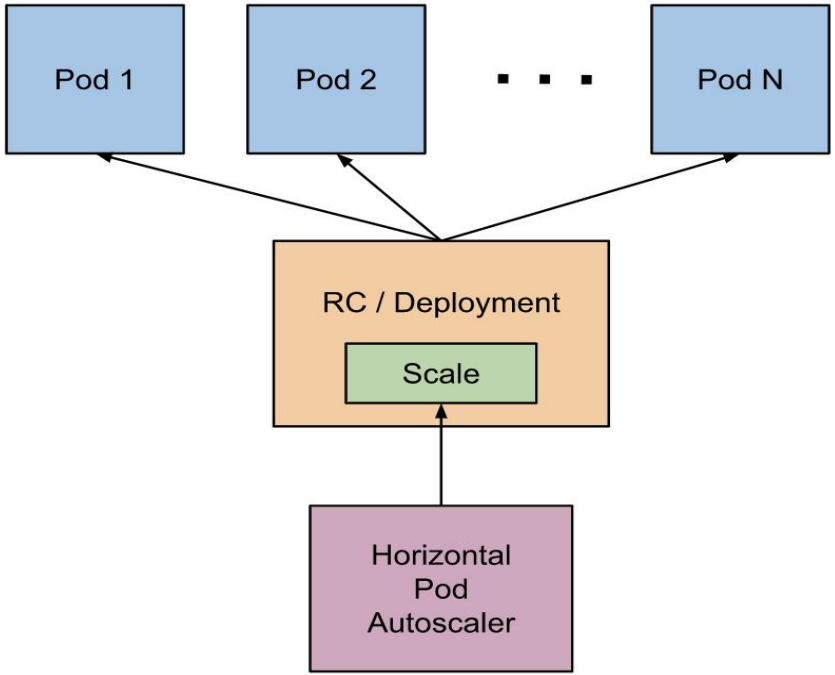
Auto Scaling

Auto Scaling

- Kubernetes offers three choices for auto-scaling resources
- Cluster Autoscaler
  - Adds nodes in a K8s cluster based on configured policy
  - Policy examples include max. Pods on each node, CPU based expansion, etc
- Horizontal Pod Autoscaler
  - Automatically scales number of Pods in a ReplicaSet, Deployment or StatefulSet based on policy configuration
  - Policies are based on CPU, memory and/or custom metric consumption
- Vertical Pod Autoscaler
  - Rarely used. Vertically scales a Pod by destroying and recreating it with higher resources
  - `updateMode` is almost always Off to get recommendations but not autoscale
- All three auto scaling options are policy driven

Horizontal Pod Autoscaling (hpa)

Horizontal Pod Autoscaler



Horizontal Pod Autoscaler

- Scales the number of Pods both Up and Down based on existing and desired resource consumption
- `desiredReplicas = ceil[currentReplicas * (currentMetricValue / desiredMetricValue)]`
- Metric values requested can be absolute numbers or percentages
- Requires a metric server to be running to extract usage (*kctl top*)
- Most commonly used for CPU and memory scaling
- Custom resources are also supported but require a custom metric server
- CLI to enable HPA imperatively: *kubectl autoscale deployment <name> --cpu-percent=50*

Horizontal Pod Autoscaler

- If multiple resources specified, then the higher scale count is considered. For instance, if CPU scaling requires 4 replicas versus memory requires 2, the final replica count is 2
- Scaling can be limited by providing the `minReplicas` and `maxReplicas` counts in the spec
- Rate of scaling can be controlled by defining scale up/down `behavior` in the spec
- If multiple behaviors are defined, we can configure weather to go with the behavior resulting in min versus max Pod count during scale up/down
- By default, scale ups are immediate and scale downs are gradual

HPA Exercise

Ingress (ing)

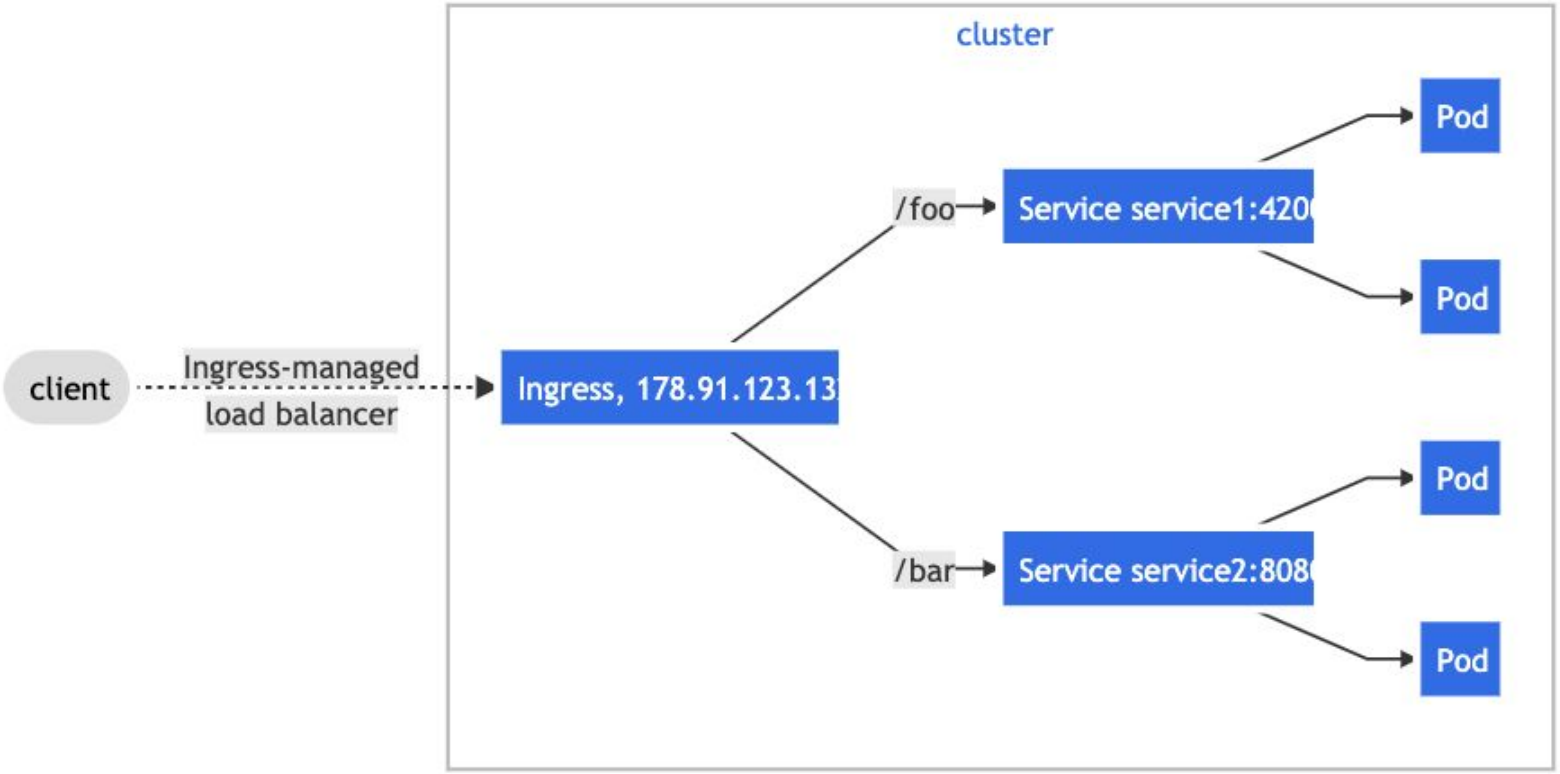
Ingress

- An API object that manages external access to services in a cluster
- It exposes HTTP/HTTPS routes from outside the cluster to services within the cluster
- Traffic routing can be controlled by setting rules on the ingress resource
- It can additionally load-balance traffic and terminate SSL/TLS
- Ingress capabilities are implemented by an ingress-controller. Unlike other controllers, K8s does not run ingress controllers by default. Users can run one of the following open-source controllers listed here. Nginx, HAProxy and Istio are quite popular
- Forwarding rules can be added for a specific service, host, API endpoint and request type
- Furthermore, API endpoints matching can be exact or prefix based
- Wildcard characters can be used for hostnames

Ingress


cluster
client
Ingress-managed
load balancer
Ingress
routing rule
Service
Pod
Pod

Ingress


cluster
Pod
/foo
Service service1:420
Pod
client
Ingress-managed
load balancer
Ingress, 178.91.123.13
Pod
/bar
Service service2:808
Pod

Additional Topics

K8s Plugins

K8s Plugins

- Kubernetes has a highly pluggable architecture
- Compute Plugins
  - Docker
  - LXC
  - Custom Scheduler
- Network Plugins
  - Flannel
  - Weave
  - Calico
- Storage Plugins
  - Supports various storage vendors out-of-box
  - Container Storage Interface (CSI)

Container Storage Interface (CSI)

Container Storage Interface (CSI)

- Open source standard to interface with external storage vendors
- Vendors implement this interface based on their APIs and capabilities
- Functionality gets invoked when PVs or PVCs are requested by applications
- Latest interface version defines standards for basic storage capabilities
- Advanced features like replication, raw block storage, etc are not defined yet
- Available storage feature set depends on a combination of the CSI spec and your storage vendor’s level of implementation/support

Probes

Probes

- Probes allow us to detect unhealthy Pod/applications and take corrective action
- They can also be used to determine if a Pod is ready before forwarding traffic
- There are 3 kinds of probes: liveness, readiness and startup
- Liveness probes determine if a Pod is healthy and if not restart it. They can be of type TCP, HTTP or command based
- Readiness probes signal when a Pod and all its containers are Ready to receive traffic. It communicates this to the K8s API
- Startup probes ping containers to confirm they have completed ‘Startup’
- Probes can be combined to build robust Pods

Probes Exercise

Secrets

Secrets

- Secrets are used when Pods want to access sensitive information at runtime (passwords, ssh keys, etc)
- Encoded versions of a ConfigMap (base64 encoding by default)
- Encrypted version of sensitive information can be accessed at runtime as well
- They can be mounted as volumes or exposed as environment variables
- Prevents need to bake sensitive information into the container image

Secrets Exercise

Jobs

Jobs

- Jobs are short-lived processes that create Pods to perform work and then cleanup the Pods once done
- Jobs can be run periodically through the CronJob resource
- Best used for asynchronous and offline tasks like garbage collection, cleanup, offline data compression, periodic scans, etc

Jobs Exercise

Additional Learning

Additional Learning

- Microservice Architecture & Design
- Helm Charts (service/application deployment)
- kOps (Kubernetes Operations)
- K8s ecosystem (Envoy, Nats, Istio, etc)