Kubernetes

Part 2

Kubernetes

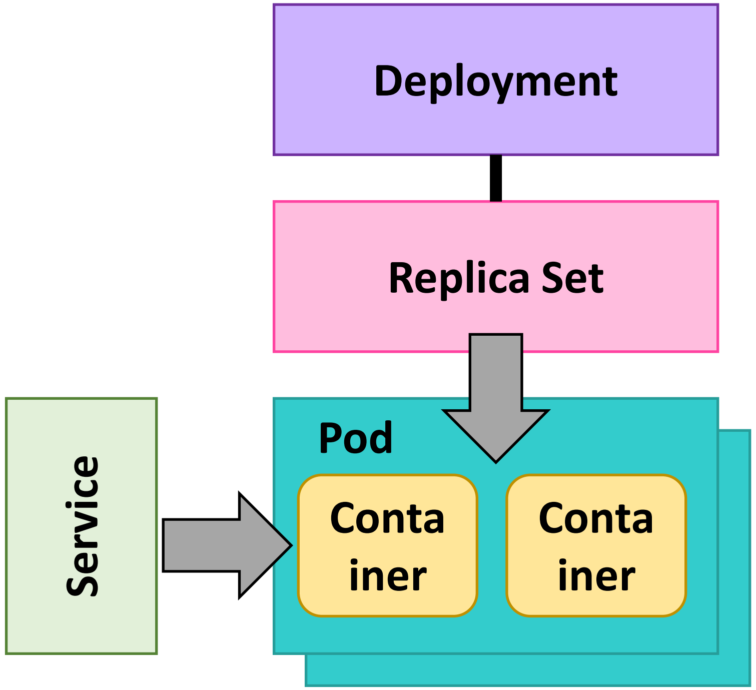


Image idea from https://www.slideshare.net/imesh/deep-dive-into-kubernetes-part-1

Volumes

- Volumes are storage that are shared by containers in a Pod
  - Allocated by the Pod, usually a shared directory in the Pod
  - Not visible outside of Pod
- Tied to the lifecycle of a Pod viz. its removed when the Pod is delete
  - Unlike Docker volumes where they are durable
- Different types of volumes
  - Eg. hostPath, NFS, iSCSI, fibre channel, empty directory, etc.
- **`hostPath`** and **`emptyDir`** type is good for sharing data between containers in a Pod
  - Eg. The example of file puller and web server

emptyDir will be deleted when pod is deleted hostPath is similar to the concept of docker bind mount

Defining a Volume

Same syntax for creating for Pod templates in deployment objects

```
kind: Pod
...
spec:
  volumes:
    - name: data-volume
      hostPath:
        path: /data
        type: DirectoryOrCreate
  containers:
    - name: webserver-pod
      volumeMounts:
        - name: data-volume
          mountPath: /var/www/html
          ...
```

Specify the volumes in this Pod

Volume’s name

Configuration details of this volume

Directory in the node’s filesystem

Type of path

Specify where to mount the volume(s)

Mount details

Mounting `ConfigMaps` and `Secrets`

- `configMap` and `secrets` can be mounted as volumes
  - Keys becomes the filename, the value is the file content
- Use cases
  - Passing configuration files eg `nginx.conf`

```
spec:
  volumes:
  - name: html-vol
    configMap:
      name: html-assets
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - name: html-vol
      mountPath: /usr/share/nginx/html
      readOnly: true
```

Kubernetes

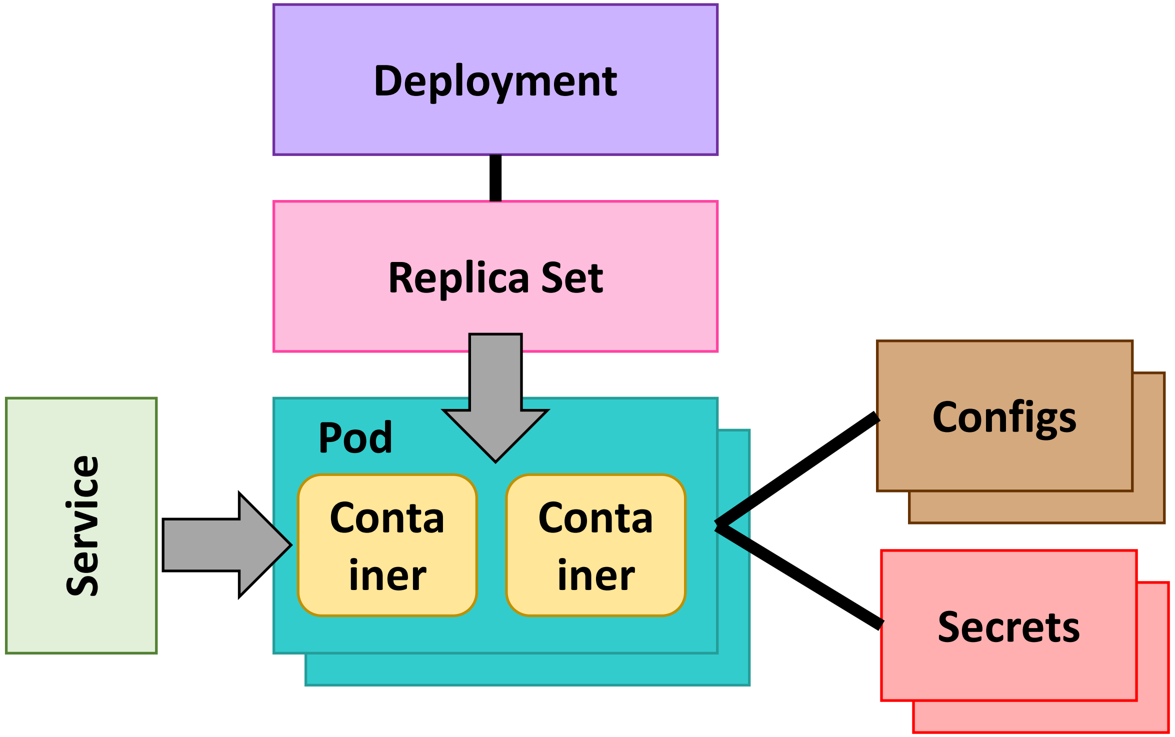


Image idea from https://www.slideshare.net/imesh/deep-dive-into-kubernetes-part-1

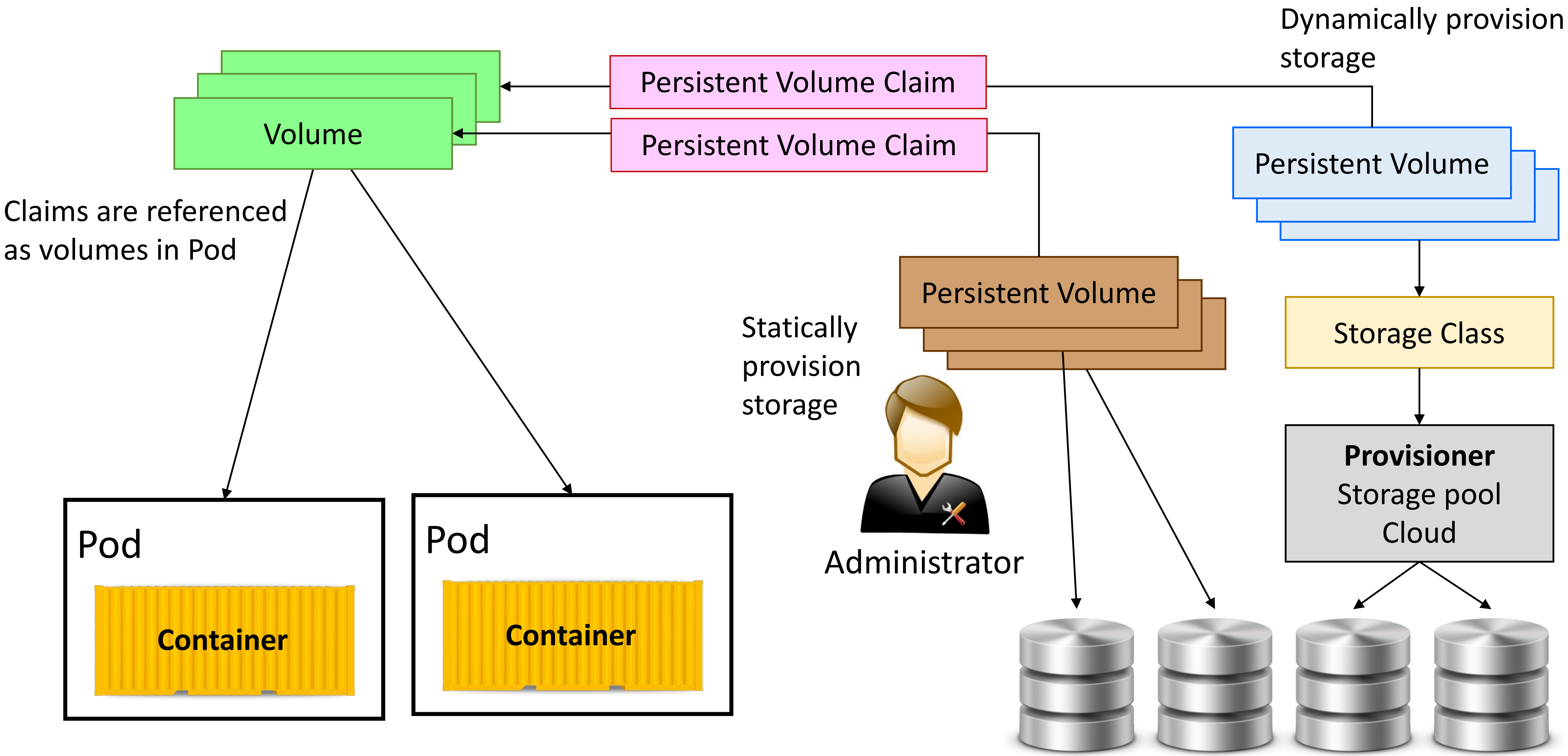
Persistent Storage

- Kubernetes can dynamically provision storage
  - Eg. User ask for 50GB volume to caching images
- Kubernetes allows storage to be either statically or dynamically provisioned
  - Static provision - an administrator will need to first provision the storage manually
  - Dynamic provision - the user describes the type of storage that is required; Kubernetes will attempt to provision based on the user's requirements
- Once a persistent storage has been allocated and claimed/reserved, a Pod can mount the volume like any regular volume
- Persistent volumes lifecycle are not tied to the Pod's lifecycle
  - Unlike volumes, persistent volumes will not be deleted when a Pod is deleted
  - This behaviour can be configured

Key Concepts

- Storage class - a type of storage
  - Who the provisioner, storage specific details, retention policy, etc.
- Persistent volume - the actual storage
  - A piece of storage provisioned by an administrator or thru storage class
  - Supports may different storage type
    - AWS EBS, Azure File Service, Cinder, fibre channel, GCP Disk, NFS, etc.
  - Different type of access mode - exclusive or shared
- Persistent Volume claim - when a persistent volume has been allocated for use, the volume is staid to be claimed

Persistent Volume



Static vs Dynamic

Static

- Administrator has to manually allocate storage and map it to a persistent volume
- Users can then claim this volume

Dynamic

- When Kubernetes tries to resolve a claim and the persistent volume is unavailable
- It looks for a storage class that best matches the request storage
- Dynamically creates the persistent volume using the provisioner

Defining a Persistent Volume Claim

```
apiVersion: v1
kind: PersistentVolumeClaim

meta-data:
  name: myapp-pvc

spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: standard
```

List of access modes

```
ReadOnlyMany
ReadWriteMany
```

Get storage class name(s) with

```
kubectl get storageclass
```

Mounting a Persistent Volume

```
apiVersion: v1
kind: Pod
meta-data:
  name: myapp

spec:
  volumes:
     - name: data-volume
       persistentVolumeClaim:
          claimName: myapp-pvc
  containers:
     - name: myapp
       volumeMounts:
          - mountPath: /app/public
            name: data-volume
       ...
```

Specify the claim name

Kubernetes

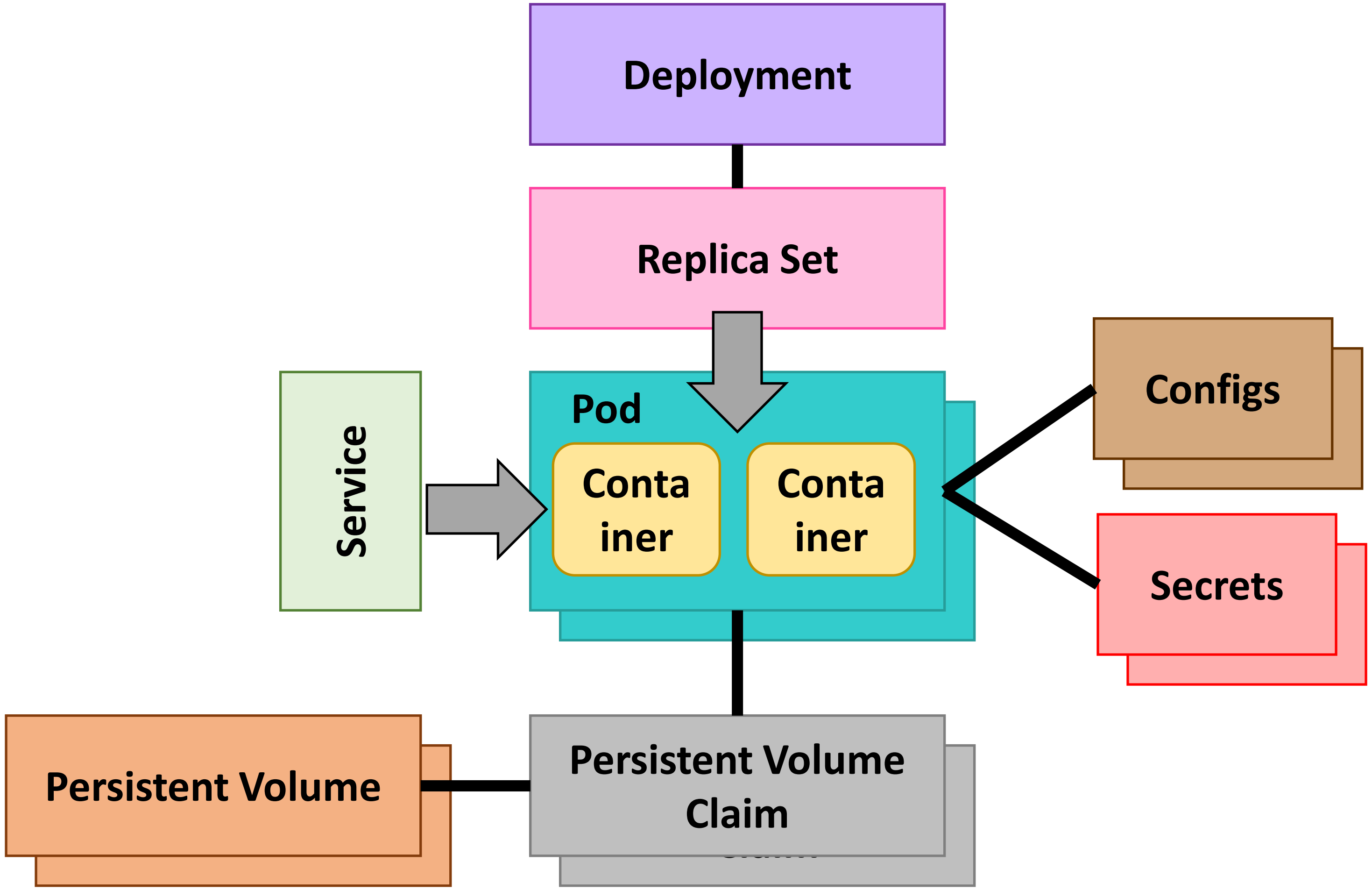


Image idea from https://www.slideshare.net/imesh/deep-dive-into-kubernetes-part-1

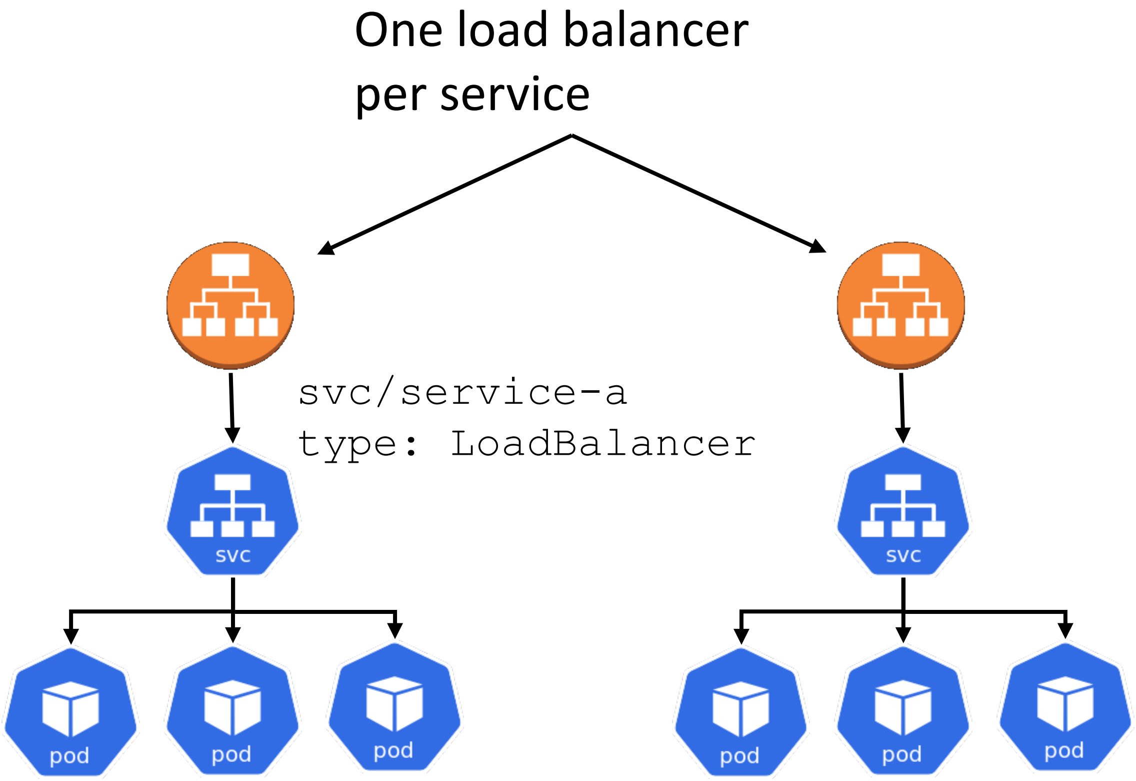
Persistence Volume Management

- Display persistence volume detail
  - Persistence volume - `kubectl get pv`
  - Persistence volume claim - `kubectl get pvc`
  - Storage classes - `kubectl get sc`
- Delete persistence volume

```
kubectl delete pvc <name>
kubectl delete pv <name>
```

Load Balancer and Ingress

- By default services are allocated a cluster IP
  - Only accessible within the cluster
- Load balancer exposes the service to the public
  - Accessible from outside of the cluster
  - Load balancer will redirect the request to pods based on its routing policy
  - Another way to allow external access is via node port
- Load balancer are resources that are provisioned from the underlying cloud platform
  - May have more features that you require
  - Also cost more



One load balancer
per service

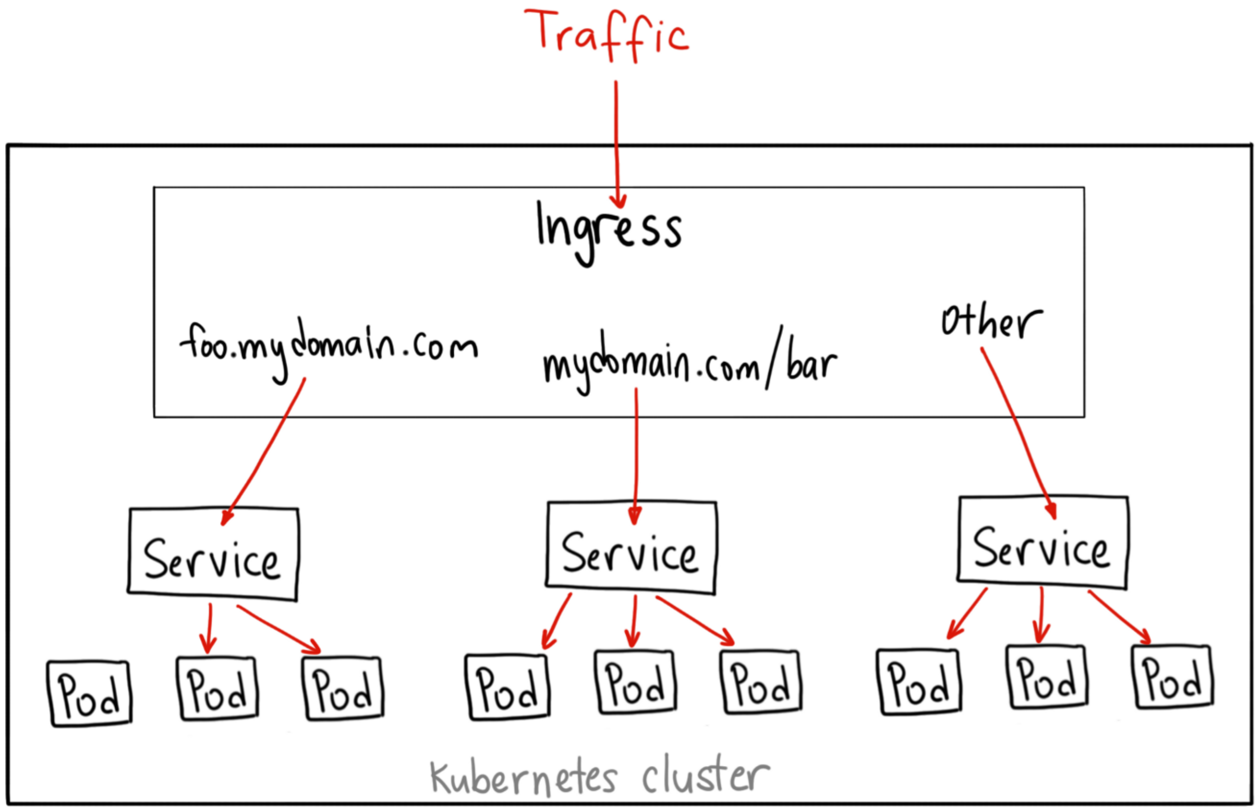
```
svc/service-a
type: LoadBalancer
```

Load Balancer and Ingress

- An ingress, like a LoadBalancer service, allows traffic into the cluster
  - Provisions an ‘external’ load balancer
  - Shared by many services within the cluster
  - More cost effective
- May require to provision an Ingress controller
  - May not be available
- NGINX Ingress controller is a popular ingress controller
  - Deploys NGINX as Ingress
  - https://github.com/kubernetes/ingress-nginx

Ingress

- Application layer (L7) router that sits in front of multiple services
- Define a set of routing rules on how services are access externally
  - Eg. 2 services, one for search one for checkout. Might map to `/search` and `/checkout`
- Rules are applied to ingress controllers which performs the actual routing
  - Controllers might be a cloud provider's load balancer or Nginx reverse-proxy



Defining an Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
   name: myapp
   annotations:
     nginx.ingress.kubernetes.io/rewrite-target: “/”
     nginx.ingress.kubernetes.io/ssl-redirect: “false”
spec:
   ingressClassName: nginx
   rules:
   - host: acme.com
     http:
        paths:
        - path: /hello
          pathType: Prefix
           backend:
             service:
                 name: myapp
                 port:
                    number: 8080
```

Change/rewrite a matched resource to its root e.g `/hello` to `/`

Used to configure NGINX ingress controller

Select the ingress controller to use

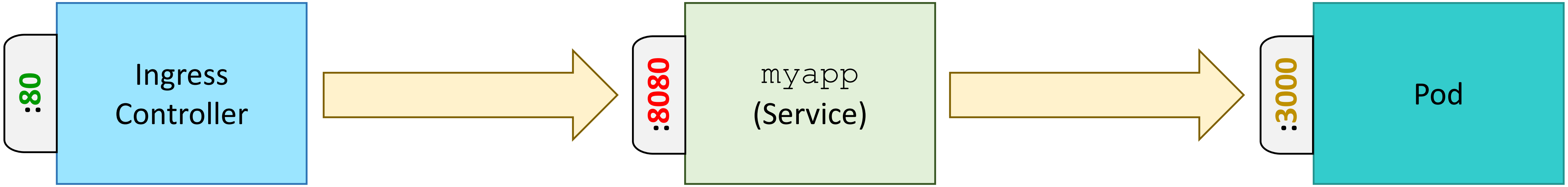
One or more of these rules to specify which services to handle what resource

Ingress Ports

```
kind: Ingress
spec:
   rules:
   - host: acme.com
     http:
        paths:
        - path: /
          pathType: Prefix
          backend:
           service:
              name: mysvc
              port:
                 number: 8088
```

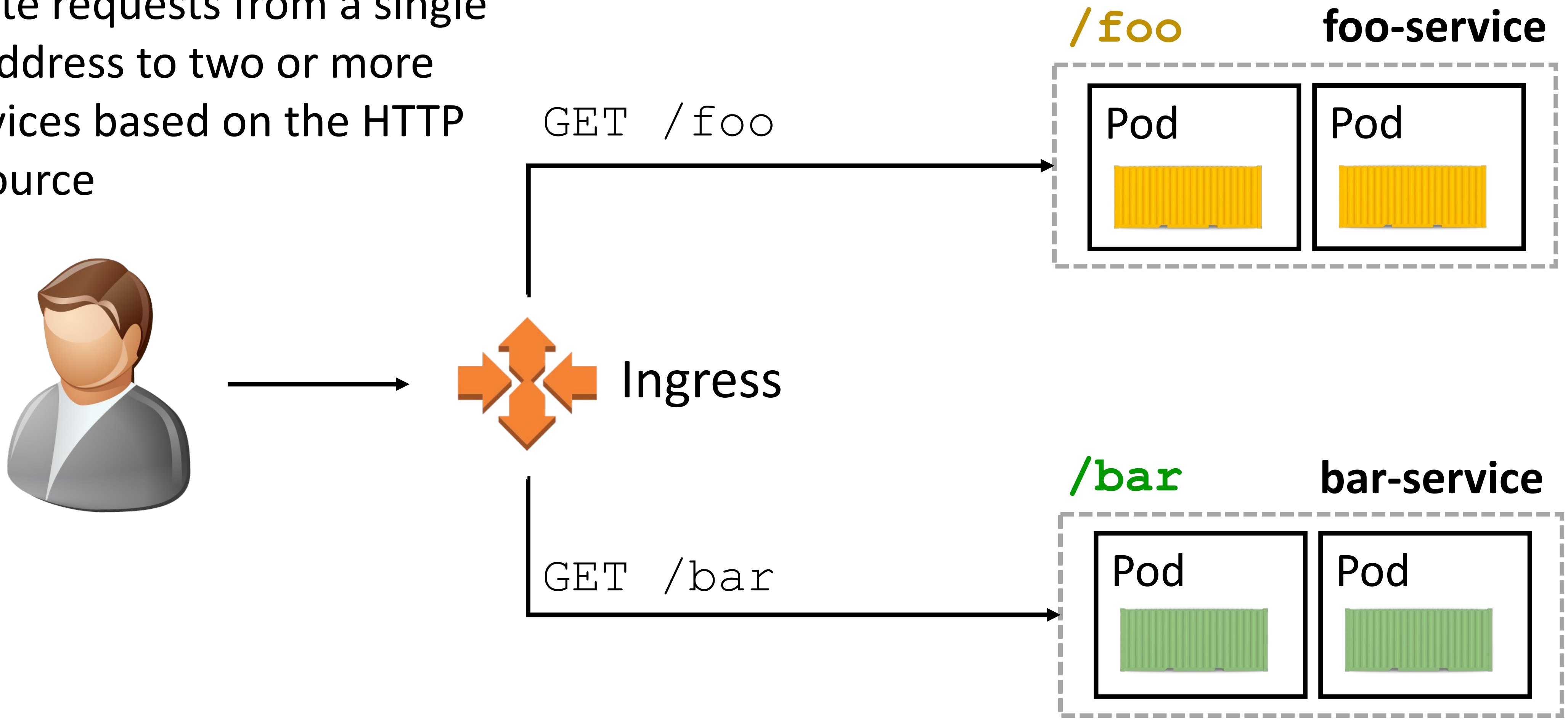
```
kind: Service
spec:
   ports:
   - port: 8080
     targetPort: 3000
```

```
kind: Deployment
spec:
   containers:
      ports:
      - containerPort: 3000
```



Ingress - Fan Out

Route requests from a single IP address to two or more services based on the HTTP resource



Ingress Fan Out Example

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
   name: myapp-ingress
   annotations:
       nginx.ingress.kubernetes.io/rewrite-target: /
spec:
   ingressClassName: nginx
   rules:
   - host: acme.com
     http:
          paths:
         - path: /foo
           pathType: Prefix
           backend:
              service:
                 name: foo-service
                 port:
                    number: 8000
         - path: /bar
           pathType: Prefix
           backend:
              service:
                 name: bar-service
                 port:
                    number: 8001
```

Note: If the 2 services are web application, then all resources references (eg in HTML) are now rooted under `/foo` or `/bar`

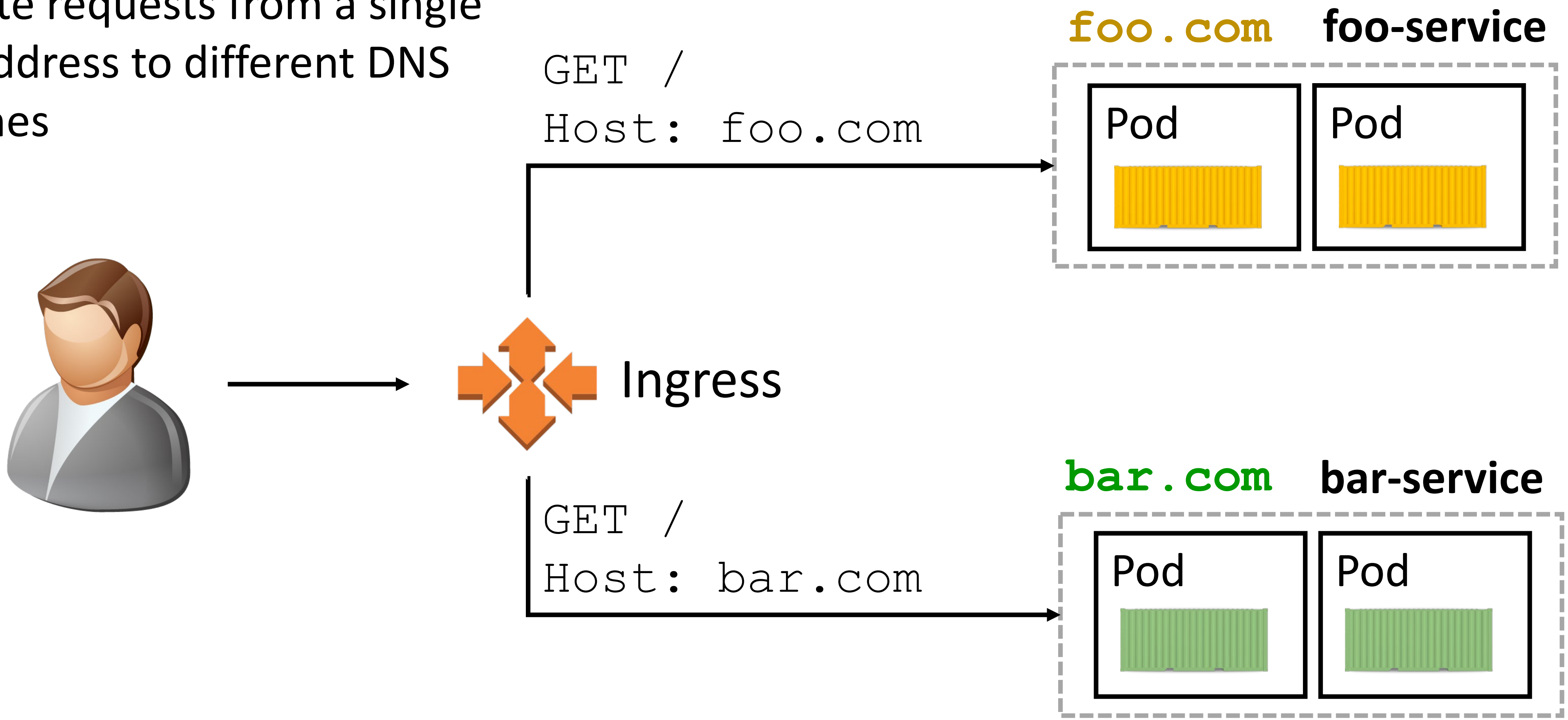
So the web application must take this into account

One option is to use relative reference or use `<base>`

Request is routed to these 2 services
`acme.com/foo` or `acme.com/bar`

Ingress - Virtual Host

Route requests from a single IP address to different DNS names



Ingress Virtual Host Example

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
   name: myapp-ingress
spec:
   ingressClassName: nginx
   rules:
   - host: foo.com
     http:
      paths:
      - pathType: Prefix
        backend:
         name: foo-service
         port:
            number: 8080
   - host: bar.com
     http:
      paths:
      - pathType: Prefix
        backend:
         name: bar-service
         port:
            number: 8080
```

`foo.com` **host**

`bar.com` **host**

Request is routed to these 2 services depending on the `Host` attribute.

Kubernetes Annotations

- Annotations are additional/proprietary metadata passed to a controller
  - Usually configuration information
- Allow additional configuration not part of the spec:
  - Eg. ingress resources does not support canary deployment. Can configure canary if using ingress-ngnix thru annotations
- https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations

ingress-nginx Features

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: myapp-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite: /$2
spec:
  ingressClassName: nginx
  rules:
  - host: acme.com
    http:
      paths:
      - path: /foo(/|$)(.*)
        pathType: Prefix
        backend:
          ...
      - path: /bar(/|$)(.*)
        pathType: Prefix
        backend:
```

Remove the first segment of the resource from the request

Original URL	Rewrote URL
/foo	/
/foo/debug	/debug
/foo/api/customer/123	/api/customer/123

Handling Errors

- If no rules matches the incoming request, then the traffic can be routed to a default backend if it is configured
  - The default backend service can be any of your application
  - Eg. Help page, chatbot

Traffic will be routed to this service if no rule matches
If the default backend is not specified, then will fallback to the ingress controller

```
apiVersion: v1
kind: Ingress
metadata:
    name: myapp-ingress
    annotations:
        nginx.ingress.kubernetes.io/rewrite: /$2
spec:
    defaultBackend:
        service:
            name: defaultBackendService
            port:
                number: 8080
    rules:
    - host: acme.com
      http:
        ...
```

NGINX

stable/nginx-ingress

Handling Errors

- An alternative is to install a global default service
  - This feature is specific to the ingress controller
- For `stable/nginx-ingress`, this is done during deployment
  - With helm
- Can only have a single default backend

values.yaml

```
defaultBackend:
    name: default-backend
    port: 3000
    image:
        repository: myrepo/backend-image
        tag: ‘v1'
```

Container port

Container image name and tag

```
helm install myingress \
    stable/nginx-ingress \
    -f values.yaml \
    -n kube-ingress
```

NGINX
stable/nginx-ingress

ingress-nginx Features

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
    name: search-ingress
    annotations:
        nginx.ingress.kubernetes.io/canary: "true"
        nginx.ingress.kubernetes.io/canary-weight: “20‟
spec:
    ingressClassName: nginx
    rules:
    - host: acme.com
      http:
        paths:
        - path: /search
          pathType: Prefix
          backend:
              name: search-v2
              port:
                  number: 8080
```

Redirect 20% of the traffic to /search

stable/nginx-ingress

ingress-nginx Features

- Enable CORS to response

```
nginx.ingress.kubernetes.io/enable-cors: "true"
```

- Rate limit the number of request from a given IP per minute and seconds. Returns a 503 if threshold is breached

```
nginx.ingress.kubernetes.io/limit-rps: “5“
nginx.ingress.kubernetes.io/limit-rpm: “300“
```

- Enable affinity/stickiness

```
nginx.ingress.kubernetes.io/affinity: “cookie“
nginx.ingress.kubernetes.io/affinity-mode: “persistent“
nginx.ingress.kubernetes.io/session-cookie-name: “sessionid"
```

stable/nginx-ingress

ingress-nginx Features

- Deploy ingress-nginx with modsecurity module enabled
  - Modsecurity enables web application firewall (L7 firewall)

values.yaml

```
controller:
  config:
    enable-modsecurity: “true”
    enable-owasp-modsecurity-crs: “true”
```

```
helm install my-ingress \
  stable/ingress-nginx -f values.yaml
```

- WAF is enable for all routes by default
  - Need to explicitly disable it
  - See https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/#modsecurity

```
nginx.ingress.kubernetes.io/enable-modsecurity: “false"
nginx.ingress.kubernetes.io/enable-owasp-core-rules: “false"
```

Kubernetes

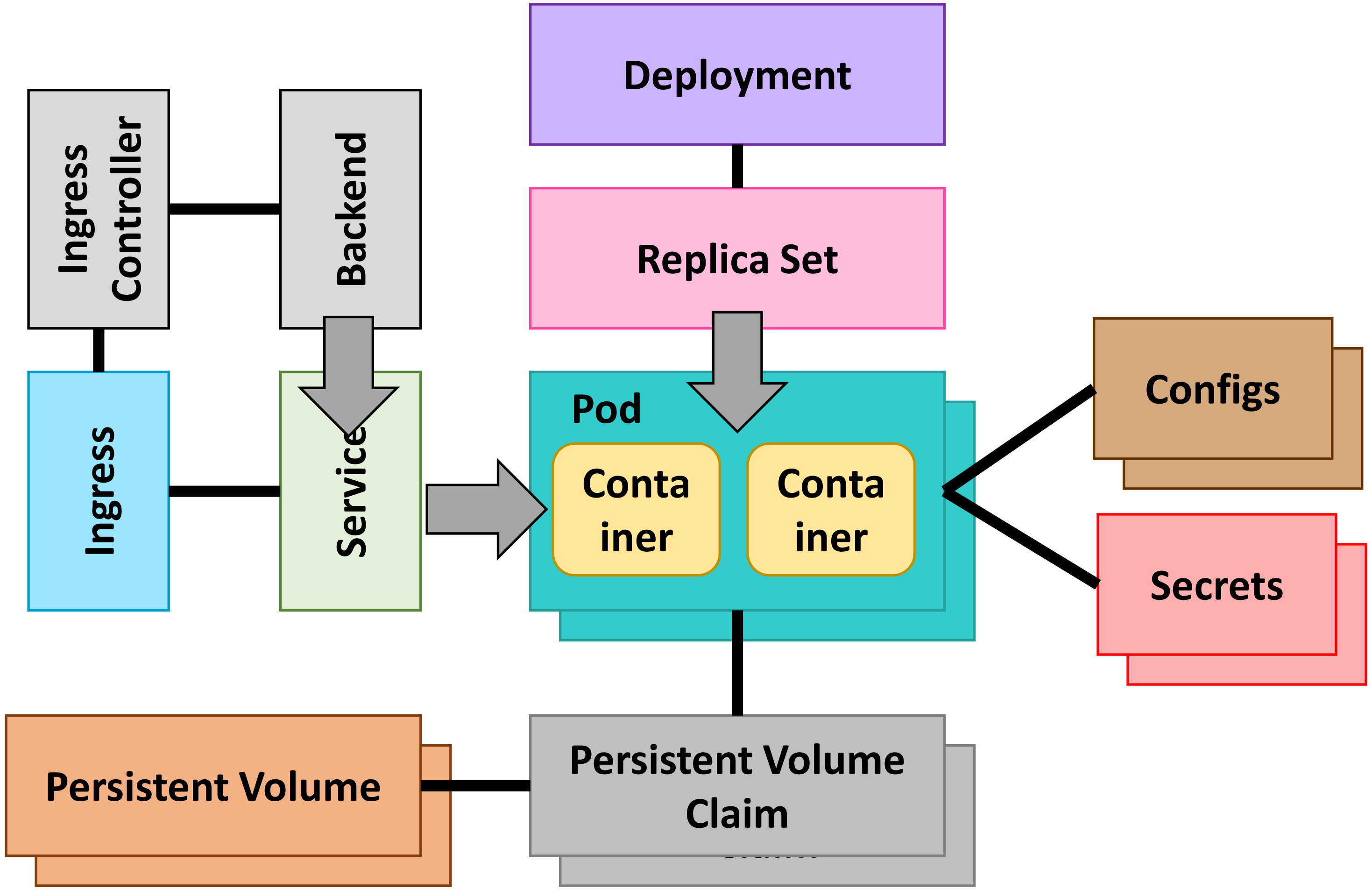
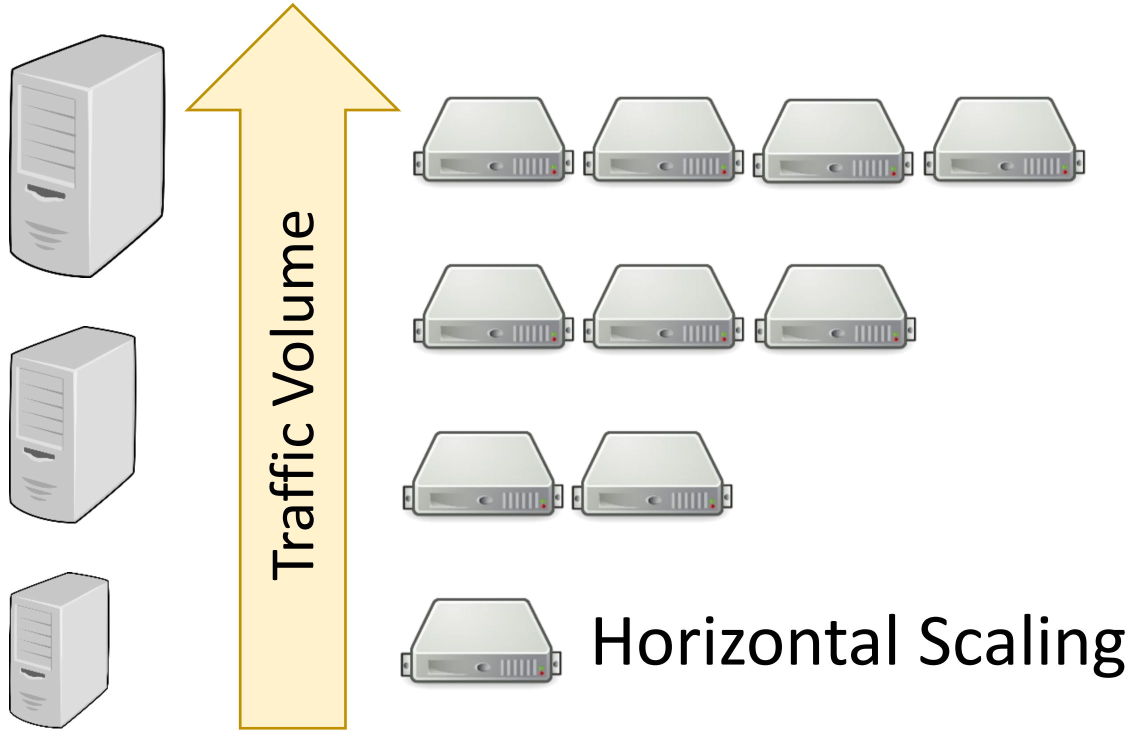


Image idea from https://www.slideshare.net/imesh/deep-dive-into-kubernetes-part-1

Scaling



Vertical Scaling

- Scaling is the capability of the system to handle more workload by provisioning more resources
- Three types of scaling
  - Horizontal scaling - scales by provision more Pods
    - Applications must be stateless allowing the ingress controller to route the request to any Pod
  - Vertical scaling - scaling by migrating the application to a ‘larger’ node
    - Application must be able to utilize the extra resources eg. more vCPUs or memory
  - Cluster scaling - scale cluster by adding nodes to the cluster
    - Cloud provider specific - typically configured when provisioning the cluster

Why Scale?

- Efficient use of resources
  - Ensure that the actual usage is on parity with the current usage
- Dynamically respond to workload fluctuation
  - Elasticity - providing an agreed on SLA
- Cost optimization
  - Pay only what you use

Horizontal Manual Scaling

- Types of scaling
  - Manual
  - Automatic - Horizontal Pod Autoscaler
- Use `kubectl` to scale up or down

```
kubectl scale --replicas <number> deployment <deployment>
```

Metrics Server

- Need to collect metrics for Kubernetes to make decision on scaling
- metrics-server is a set of pods running in Kubernetes
  - Collects CPU and memory utilization
  - Stores them in memory not to an external datastore
  - For viewing historical data, require more advance packages like Prometheus
- Can be installed from a YAML file or as a helm chart

Monitoring Kubernetes with top

- Node metrics

```
kubectl top node
```

- Pod metrics

```
kubectl top pod
```

- `top` will only work if metrics-server has been installed

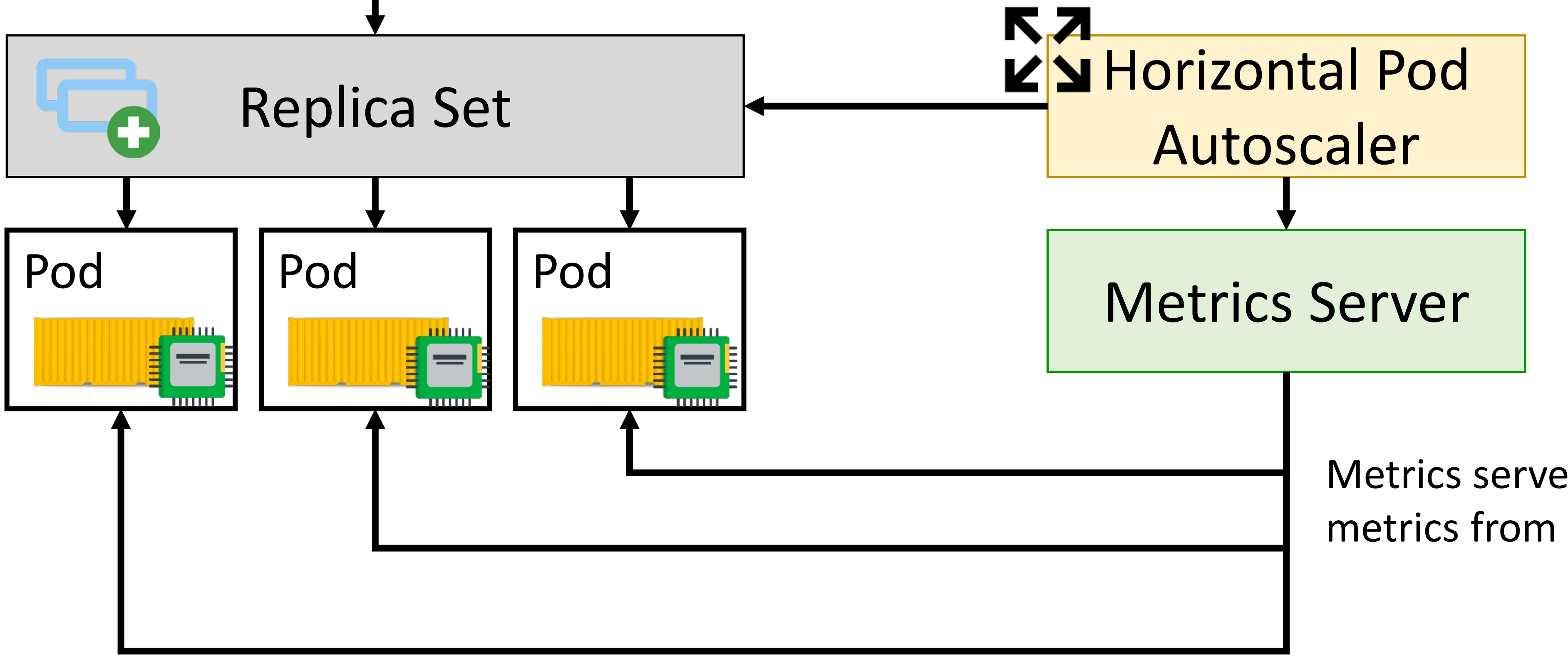
Horizontal Pod Autoscaler

- HPA scales a deployment based on one or more metrics
    - Eg. trigger scaling when CPU utilization breaches 80%
    - Metrics to scale the Pods can be
        - Build in metrics , custom metrics, external metrics
- HPA runs a control loop - runs every 30 seconds (default)
    - Queries metrics server
    - Match that against the specified threshold
    - Updates the number of replicas in a deployment if required to meet the load
    - Deployment would then perform the scaling (in or out)
- Reduces cluster size if utilization is low for a period of time
    - Scaling in

Setting Pod Request

- Horizontal scaler scales a Pod by determining if the Pod has breached a certain threshold
  - For memory and CPU
- Set the request for CPU and memory
  - Specify the minimum amount of compute resources required
- Resource type
  - CPU - measured in CPU units eg 100m is 100 millicores
    - 1 CPU in Kubernetes == 1 vCPU, Core, vCore, Hyperthread
  - Memory - 16M

Horizontal Pod Autoscaler



Autoscaler collect metrics from the Metrics server and use it to determine the scaling policy

Metrics server collects metrics from the Pods

Requesting Resources

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
    - name: myapp
      image: myapp:sha256:...
      resources:
        requests:
          cpu: 100m
          memory: 16M
        limits:
          memory: 32M
      ...
```

HPA only looks at the CPU

Request the minimum amount of compute resources

Describe the maximum amount of compute resources required

Defining a Horizontal Pod Autoscaler

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoScaler
metadata:
   name: myapp
spec:
  minReplicas: 1
  maxReplicas: 8
  scaleTargetRef:
     apiVersion: apps/v1
     kind: Deployment
     name: myapp
```

The deployment that this HPA is targeting

Minimum and maximum number of replicas. Since the HPA is managing the replica set, this setting takes precedence over the deployment setting

Scale the pod when these threshold are breached

```
metrics:
- type: Resource
   resource:
     name: cpu
     target:
        type: Utilization
        averageUtilization: 80
- type: Resource
  resource:
     name: memory
     target:
        type: Utilization
        averageUtilization: 80
```

Kubernetes

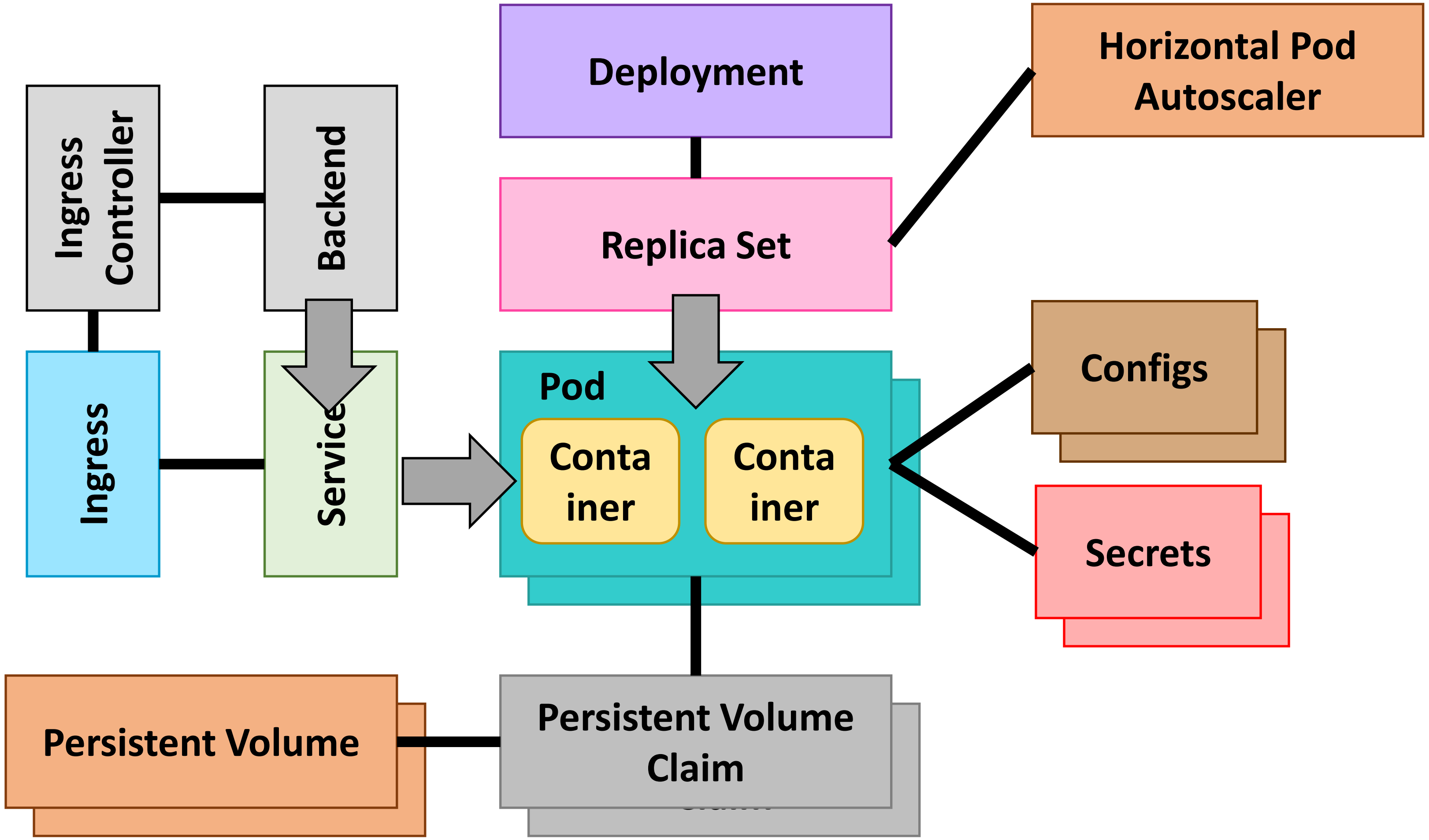


Image idea from https://www.slideshare.net/imesh/deep-dive-into-kubernetes-part-1

Appendix

Using ConfigMaps

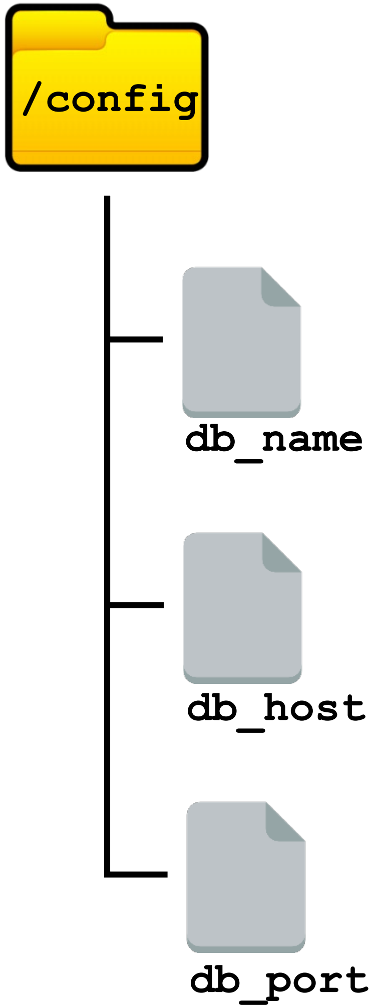
Injecting as environment variables

```
containers:
  ...
  env:
    - name: DB_NAME
      valueFrom:
        configMapKeyRef:
          name: myapp-config
          key: db_name
    - name: DB_HOST:
      valueFrom:
        configMapKeyRef:
          name: myapp-config
          key: db_host
```

Mounting as a volume

```
volumes:
  - name: config-volume
    configMap:
      name: myapp-config

containers:
  ...
  volumeMounts:
    - name: config-config
    - mountPath: /config
```



Managing Context

- For grouping access parameters under a common name
  - Like a profile
  - Set the namespace, do not need the `–n` option
- Create a context

```
kubectl config set-context <context_name> --namespace=<name> \
  --cluster=<cluster_name> --user=<user_name>
```

- View current contexts

```
kubectl config view
```

- Use a context

```
kubectl config use-context <context_name>
```