Kubernetes

Part 3

Headless Service

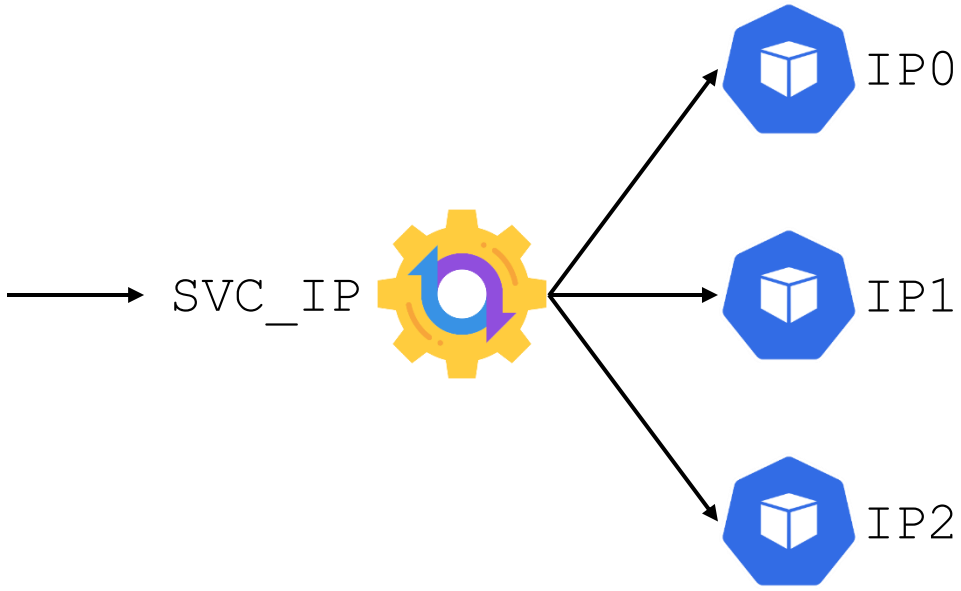
- A service that has no cluster IP
  - Service selector selects all matching pods
  - These are the pods that will be exposed via the service
  - Client has to select which pod it wants to connect to
  - Headless service exposes the pods directly to the client
- A ‘headfull’ service provides a service IP address
  - Service load balance one or more upstream (pods) over a single service IP address
  - Access the service’s IP address, the service will proxy the request to one of the endpoints defined by the pod selector

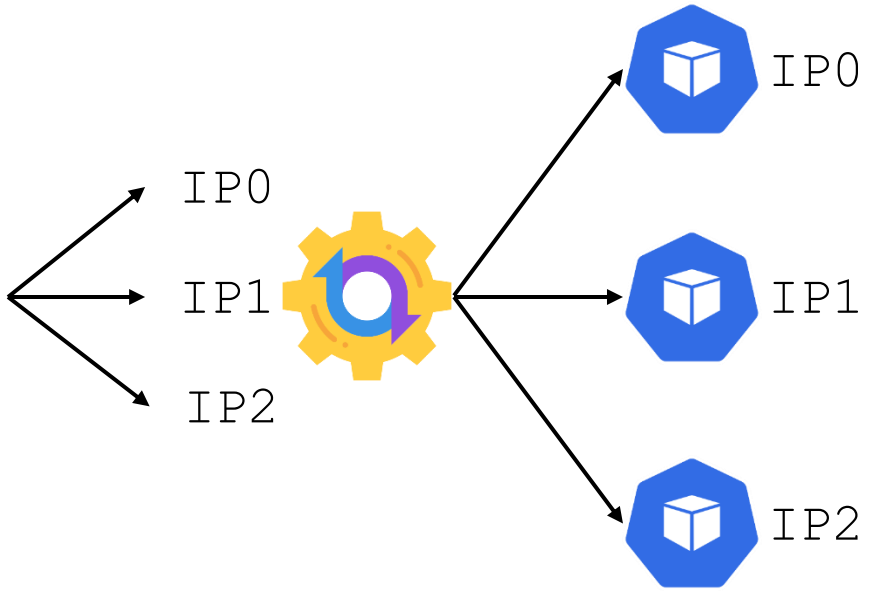
Headless Service

Set the `custerIP` field to `None`

```
apiVersion: v1
kind: Service
metadata:
   name: mysql-hsvc
spec:
   clusterIP: None
   selector:
      name: mysql-po
   ports:
   - port: 3306
     targetPort: 3306
```

Headfull vs Headless Service





Stateless and Stateful Application

Stateless

- Stateless applications are applications that have no context when processing a client's transaction
  - Eg. CDN, web application, search engine, queues
- Client need to supply all the required information
  - Stateless application cannot collate related transactions
- Scales horizontally by adding more instances

Stateful

- Stateful applications are those that have context or knowledge of previous transactions
  - Eg. email servers, databases,
- Achieve statefulness by storing client's context and associating it with a transaction
  - Transactions need to be processed by the application that has the correct context
- Scales vertically by increasing the resources of individual instance

Stateful Applications in Kubernetes

- Created by **`StatefulSet`** resource
- Pods are created sequentially and are removed in reverse order
- Each instance (pod) is unique and has an identity
  - Each stateful pod is suffix with a zero based index eg. `mysql-0`
  - Stateful pods maintain their identities; remains consistent across rescheduling or restarts
- Stateful pod has its own volume, if you chose to provision volumes
  - Volumes are not shared and will always remounted to the same stateful pod
- Require a headless service for accessing individual pods
  - `<pod-name>.<service-name>.<namespace>.svc.cluster.local`

StatefulSet

Create a `StatefulSet`

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql-sts
spec:
  replicas: 1
  selector:
    matchLabels:
      name: mysql-po
  serviceName: mysql-hsvc
```

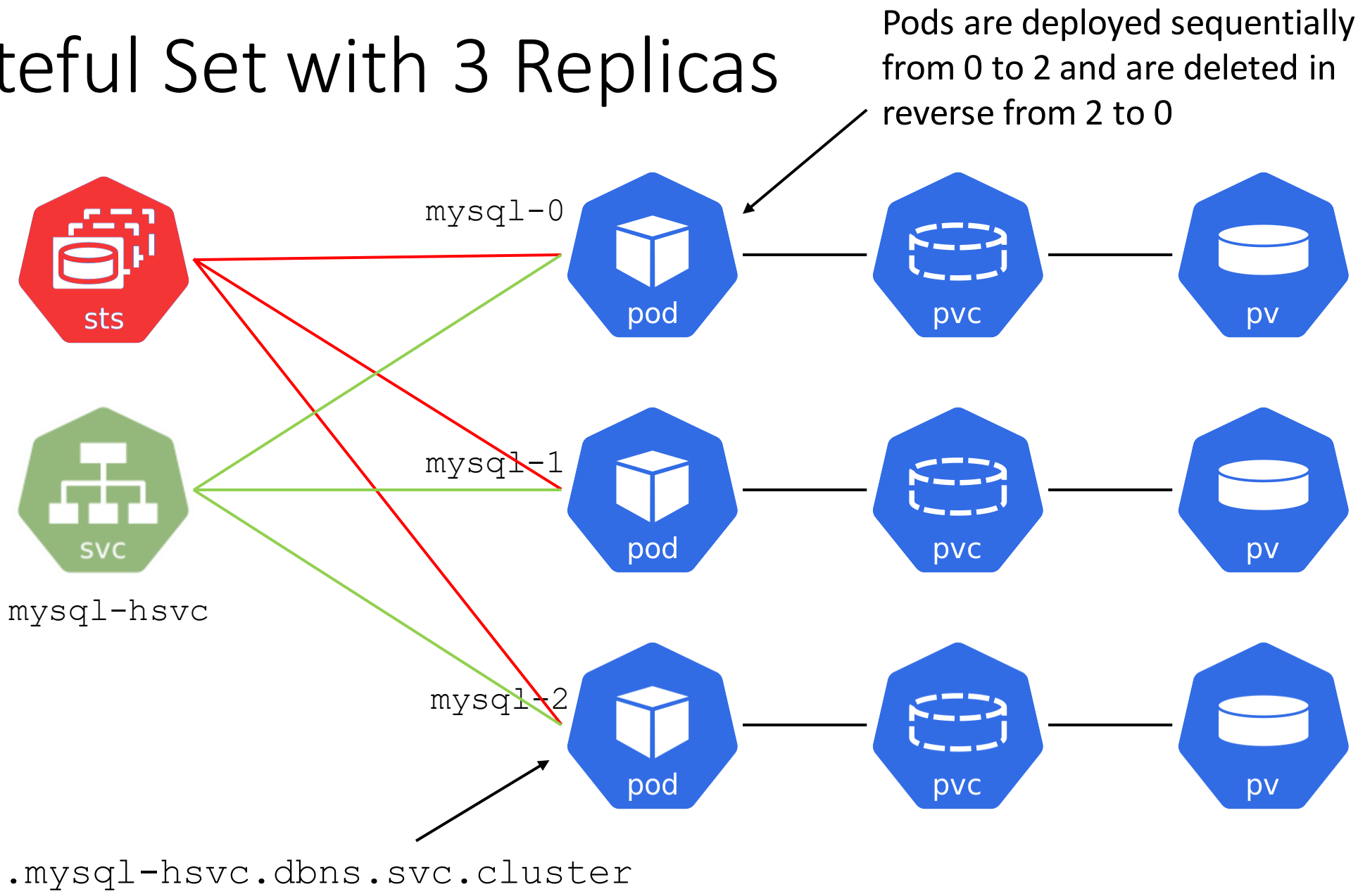
```
volumeClaimTemplates:
- metadata:
    name: data-vol
  spec:
  accessModes: [ "ReadWriteOnce" ]
  resources:
    requests:
      storage: 1Gi
  storageClassName: standard
```

Volume definition

```
template:
  metadata:
    name: mysql-po
    labels:
      name: mysql-po
  spec:
    containers:
    - name: mysql
      image: mysql:8.0
      volumeMounts:
      - name: data-vol
        mountPath: /var/lib/mysql
```

Pod definition

Stateful Set with 3 Replicas



Differences Between **`StatefulSet`** and **`Deployment`**

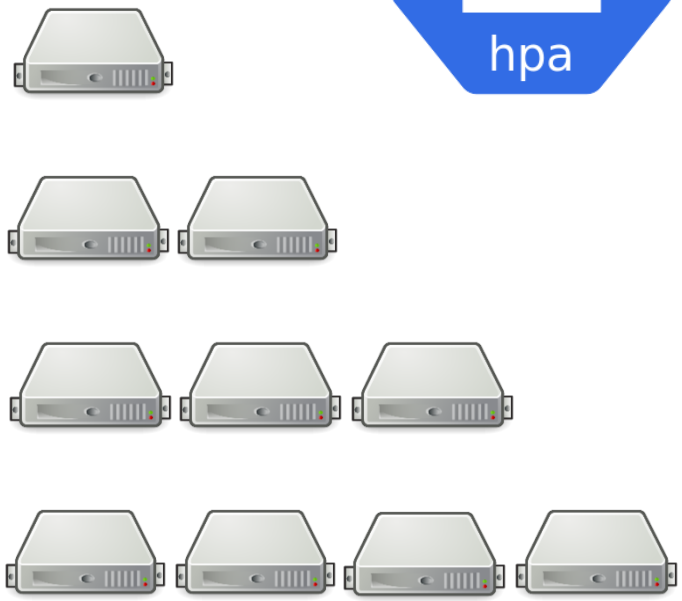
`StatefulSet`

- For stateful applications
- Each pod has a unique name, is maintained across restarts
- Created in sequential order, destroyed in reverse order
- Every pod can be different
- Pods do not share volumes
- Headless service exposes individual pod directly to the client
- Scales by increase the resources of individual instance with `VerticalPodAutoscaler`

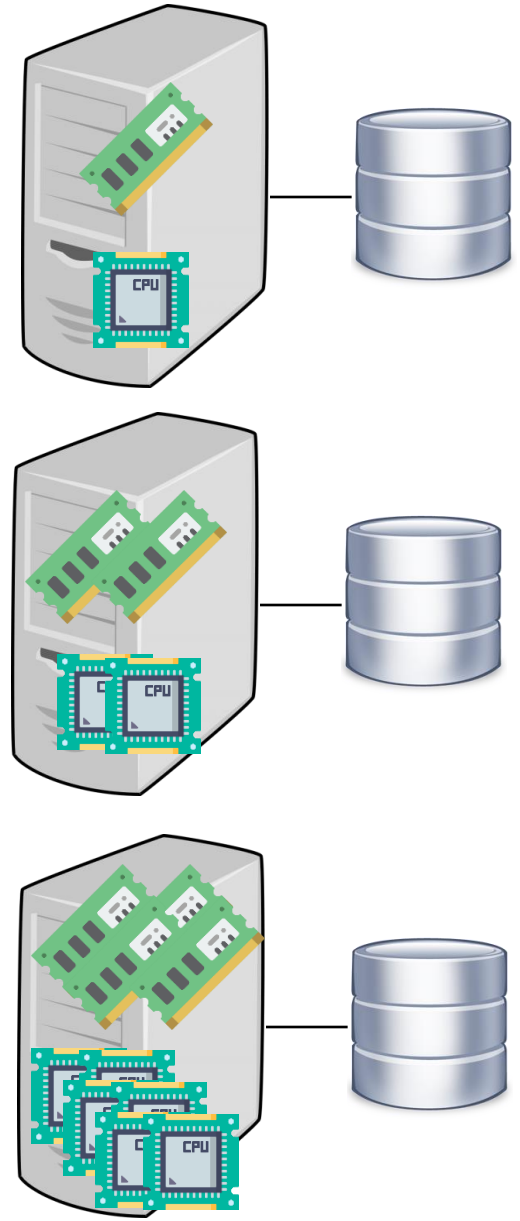
`Deployment`

- For stateless applications
- Pod's name have random generated suffix
- Created and destroyed randomly in parallel
- Every pod is the same
- Pods can share a volume
- Services exposes all pods to the client
- Scales by adding multiple instances with `HorizontalPodAutoscaler`

Scaling Stateless vs Stateful



Increase traffic



Increase the number of instances

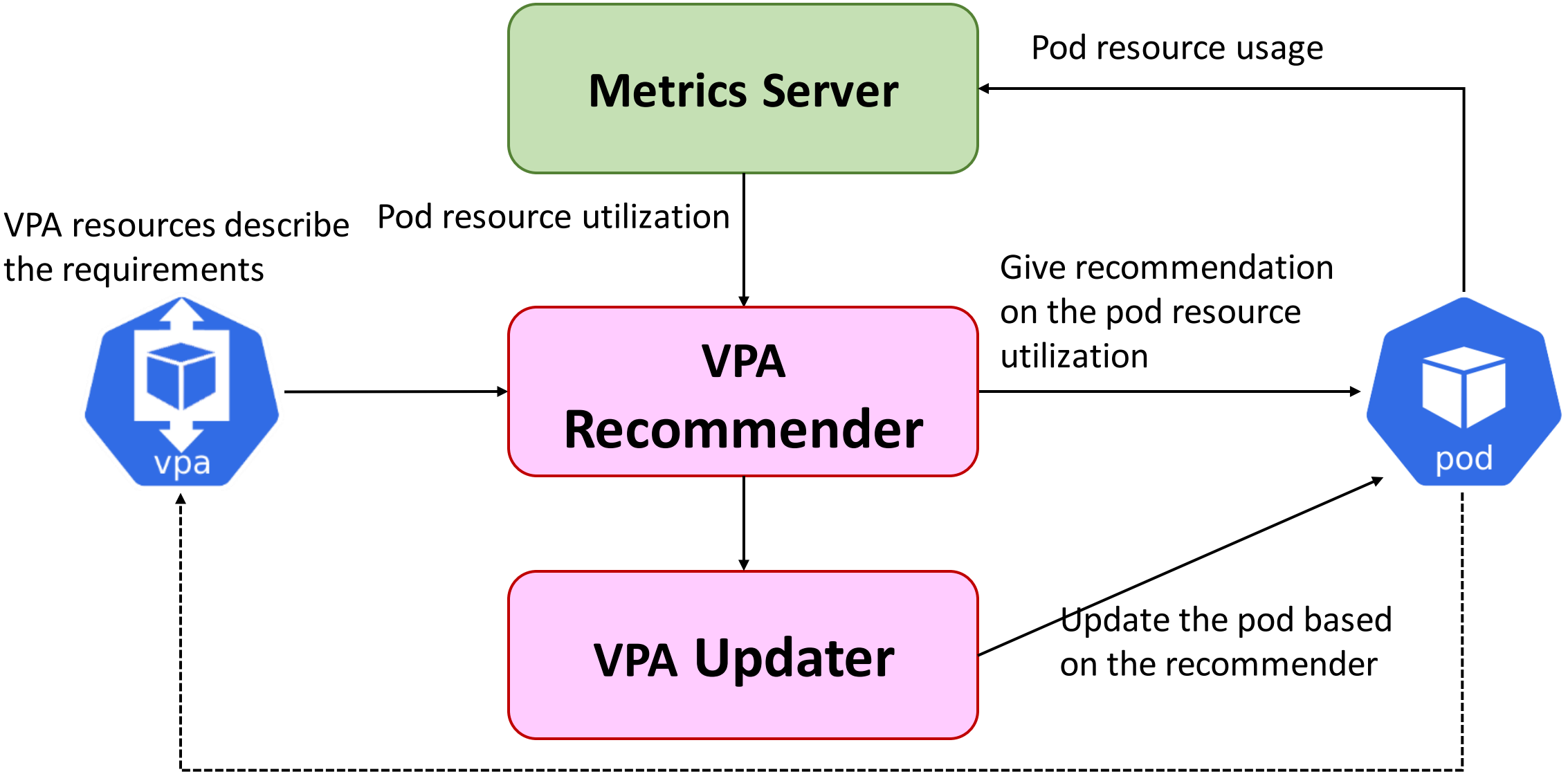
Increase the resource of each instance

Unused

Vertical Pod Autoscaler

- Automatically adjust the CPU and memory reservation of pods
  - Often pods resource requests and limits are based on ‘guesstimate’
  - VPA right size these values to free up resources for other pods
- Kubernetes does not re-evaluate these values after pods have been deployed
  - One of VPA’s functionality is to continuously evaluate pods resource requirements
  - Can automatically adjust these requirements
  - Important for stateful workloads or vertically scaled applications
- VPA cannot be used together with HPA in a single pod

Vertical Pod Autoscaler Architecture



Vertical Pod Autoscaler

- VPA observes the pod in question and determine the baseline resource requirements
  - The resource lower, upper and recommended targets
- Based on observation, VPA can
  - Make changes by applying the recommended resource targets to the pods
  - Report the lower, upper and recommended targets but not make any changes

Defining a Vertical Pod Autoscaler

```yaml
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
    name: myapp-vpa
spec:
    targetRef:
      apiVersion: apps/v1
      kind: StatefulSet
      name: myapp-sts
    updatePolicy:
       updateMode: "Off"
```

The `StatefulSet` that the VPA is monitoring

Recommendation only
No updates to pods

```
[k8s-test] k get vpa -nbggns -owide
NAME         MODE   CPU   MEM         PROVIDED   AGE
bggdb-vpa    Off    25m   716711186   True       4m59s
```

Recommended resources settings for each pod in the `StatefulSet`

Recommendations

- Get more details with ‘describe’
- Lower bound is the estimated minimum required resources
  - Lower values may cause the pod to die
- Upper bound is the estimated maximum required resources
  - Any values above these will be wasted
- Target is the recommended optimal resource setting
- Uncapped would be the resources settings if the VPA resource have not been configured with upper or lower limits

```
Status:
  Conditions:
    Last Transition Time:  2022-08-29T08:42:51Z
    Status:                True
    Type:                  RecommendationProvided
  Recommendation:
    Container Recommendations:
      Container Name:  bggdb-container
      Lower Bound:
        Cpu:     25m
        Memory:  614548542
      Target:
        Cpu:     25m
        Memory:  716711186
      Uncapped Target:
        Cpu:     25m
        Memory:  716711186
      Upper Bound:
        Cpu:     5098m
        Memory:  58000565460
```

VPA Updater

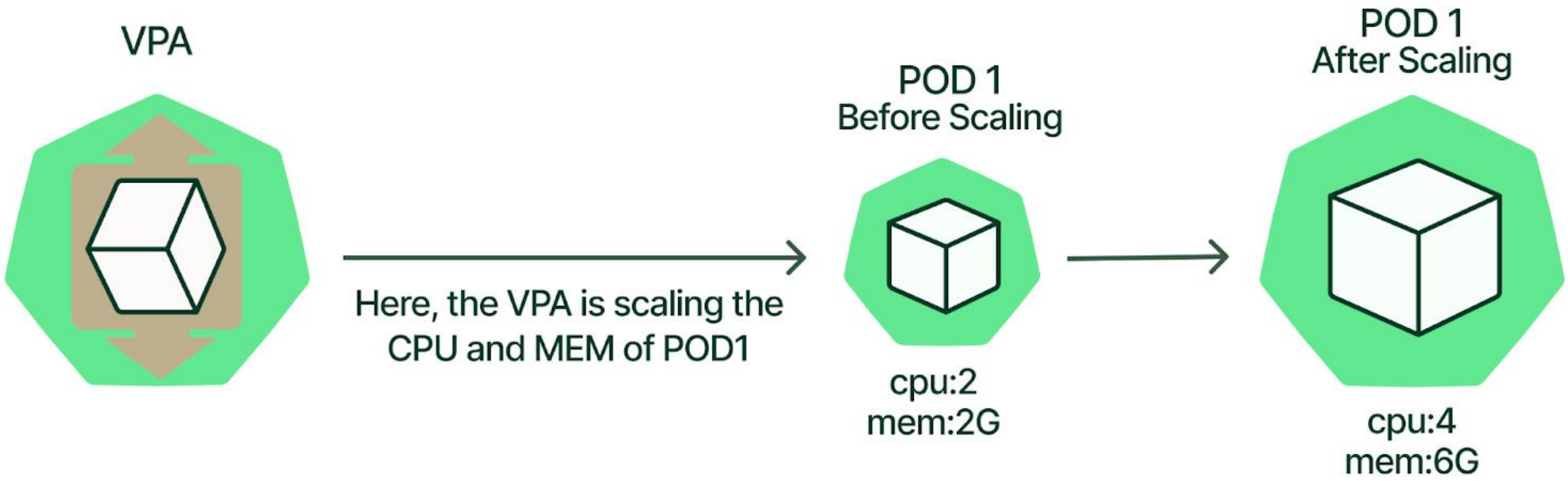


Image from https://www.kubecost.com/kubernetes-autoscaling/kubernetes-vpa

Enable VPA to Update Pod

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
    name: myapp-vpa
spec:
    targetRef:
        apiVersion: apps/v1
        kind: StatefulSet
        name: myapp-sts
    updatePolicy:
        updateMode: "Auto"
    containerPolicies:
        ...
```

Update the resources by recreating the pod with new resource settings

```
containerPolicies:
- containerName: '*'
  minAllowed:
    cpu: 250mi
    memory: 250Mi
  maxAllowed:
    cpu: 1mi
    memory: 1000Mi
  controlledResources: [ "cpu", "memory" ]
```

Name of the container to scale

Capped the resource setting. If these are not set, VPA will update these as it sees fit

Resources to get recommendation for. CPU and memory is the default

VPA Update Modes

- VPA can update pod resources based on the following update modes
  - **`Initial`** - only apply the recommended resources targets to newly created pods, will not change during the lifetime of the pod
  - **`Auto`** – assign requested resources to pod; will terminate and update existing pod if the requested resources differs 'significantly' from the recommended.
  - **`Recreate`** – assign requested resources to pod; terminate and recreate pods when recommendation changes
- Recreate mode will not be enabled if a **`StatefulSet`** or **`Deployment`** does not have at least 2 replicas
  - Use **`minReplicas`** attribute to control the number of available replicas when pods are recreated; like **`maxUnavailable`** in **`Deployment`** upgrades

How VPA Updates Pod

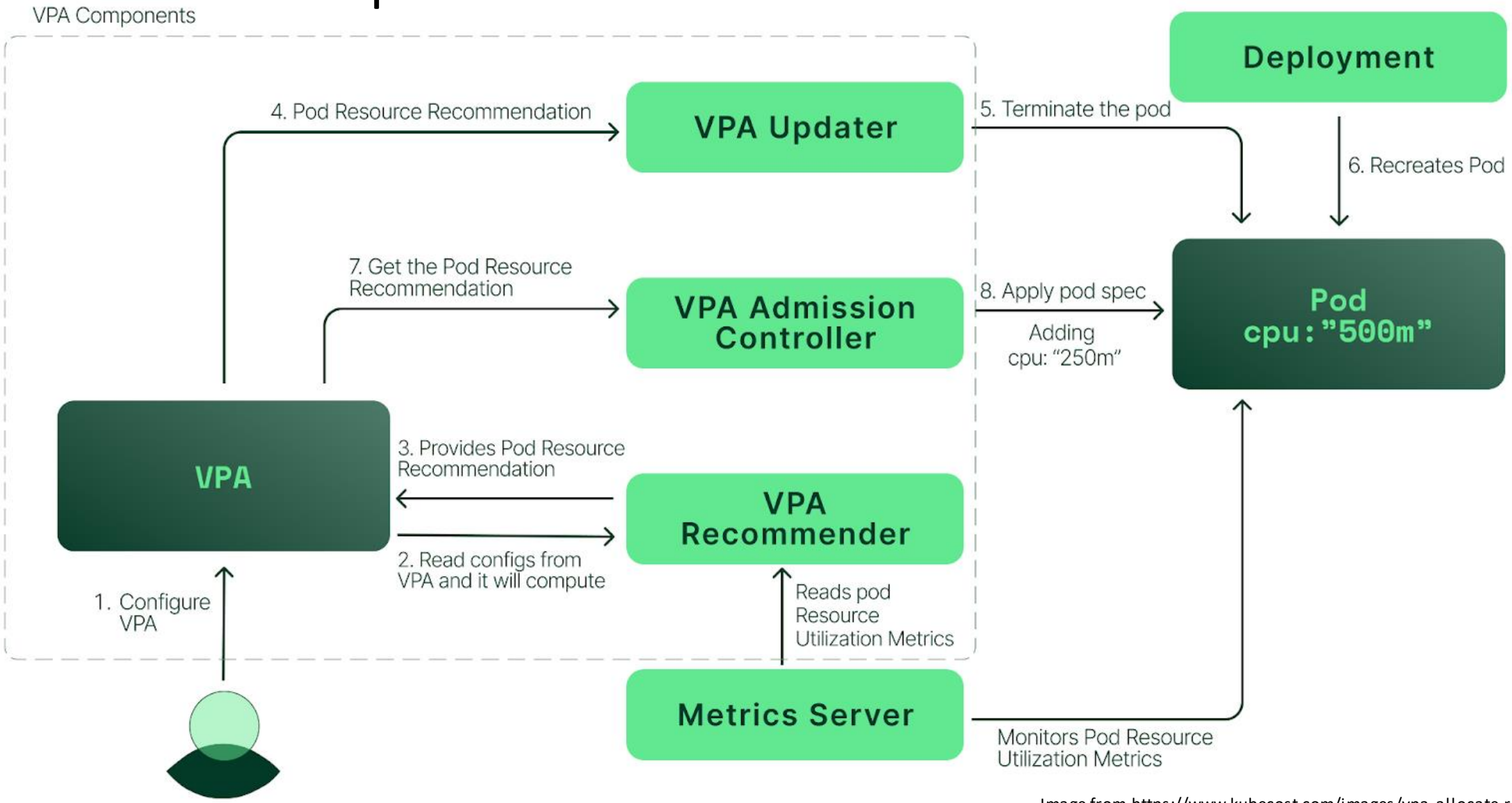


Image from https://www.kubecost.com/images/vpa-allocate-resources.png

Use VPA in Recommender Mode

- VPA can cause disruption to service in Recreate update mode
  - When CPU and/or memory fluctuates
- Practice is get resource recommendations and set the resources to the pods manually
- See https://povilasv.me/vertical-pod-autoscaling-the-definitive-guide for more detailed guide on better resource estimation

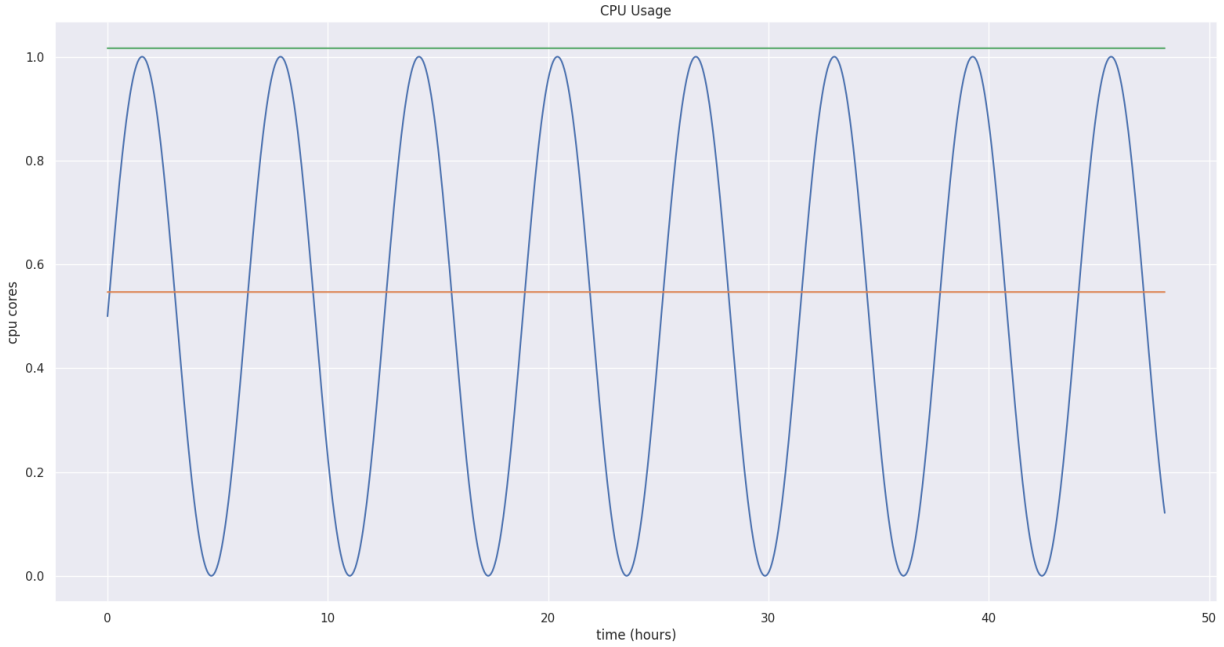


Image from https://povilasv.me/wp-content/uploads/2021/01/cpu-usage-1536x749.png

Kubernetes

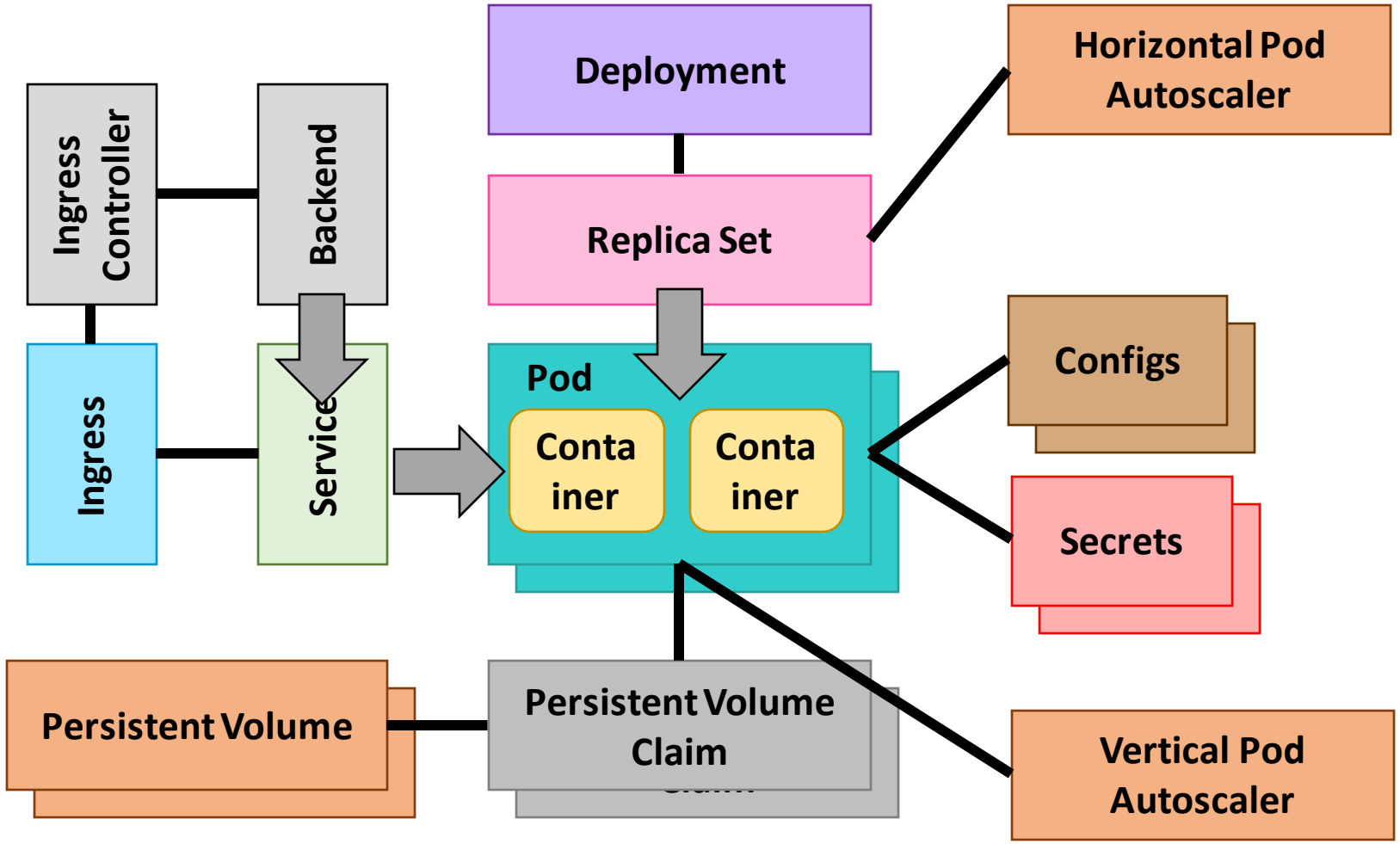


Image idea from https://www.slideshare.net/imesh/deep-dive-into-kubernetes-part-1

Stateless Application in Kubernetes

- Replica set is used to create multiple instances of the same pod
  - Typically created via a Deployment
  - Replica set maintains the number pod instances, creating new instances when an existing instance dies
- Each pod instance name is the pod's name plus a random hash suffix
  - When a new pod is provisioned, the new pod will get a new hash suffix
  - Different from the one it replaces
- A service routes request to the replica set
  - Since all pods in the replica are the same, does not matter which instance serves the request
  - No advantage in requesting for a specific pod to process request
- Scales by increasing the number of instances in the replica set