# Kubernetes Design Patterns

SoC Summer Workshop
Cloud Computing with Big Data

**Richard T. B. Ma**
School of Computing
National University of Singapore

# Roadmap

- Behavioral Patterns
  - Stateful service -- StatefulSet
  - Self-awareness -- Downward API
- Structural Patterns
  - Init container
  - Sidecar container
- Lifecycle Patterns
  - Health probes
  - Managed lifecycle

# Servers in the DevOps World

## Pets

- nonfungible servers
  - every instance is unique
- require individual care
  - repair
  - vertical scaling
- stateful, persistent and permanent

## Cattle

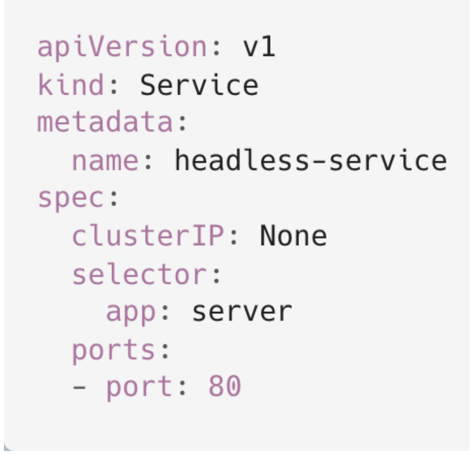
- identical servers
  - all instances are the same
- not need individual care
  - replaced
  - horizontal scaling
- stateless, ephemeral, and transient

# StatefulSet (STS)

- Distributed stateful apps require
  - persistent storage, identity, networking, and ordinality
  - every instance is unique & has long-lived characteristics
  - e.g., big data frameworks such as Map-Reduce
- Solution: `StatefulSets` provides
  - stable, unique network identifiers: each Pod in a STS gets a unique hostname based on its ordinal index.
  - stable, persistent storage: each Pod can be associated with a PersistentVolume.
  - ordered, automated rolling updates: STS manages the deployment and scaling in an ordered & deterministic fashion

# StatefulSet - how to use?

- Step 1: create a headless service
  - a ClusterIP Service without a virtual IP



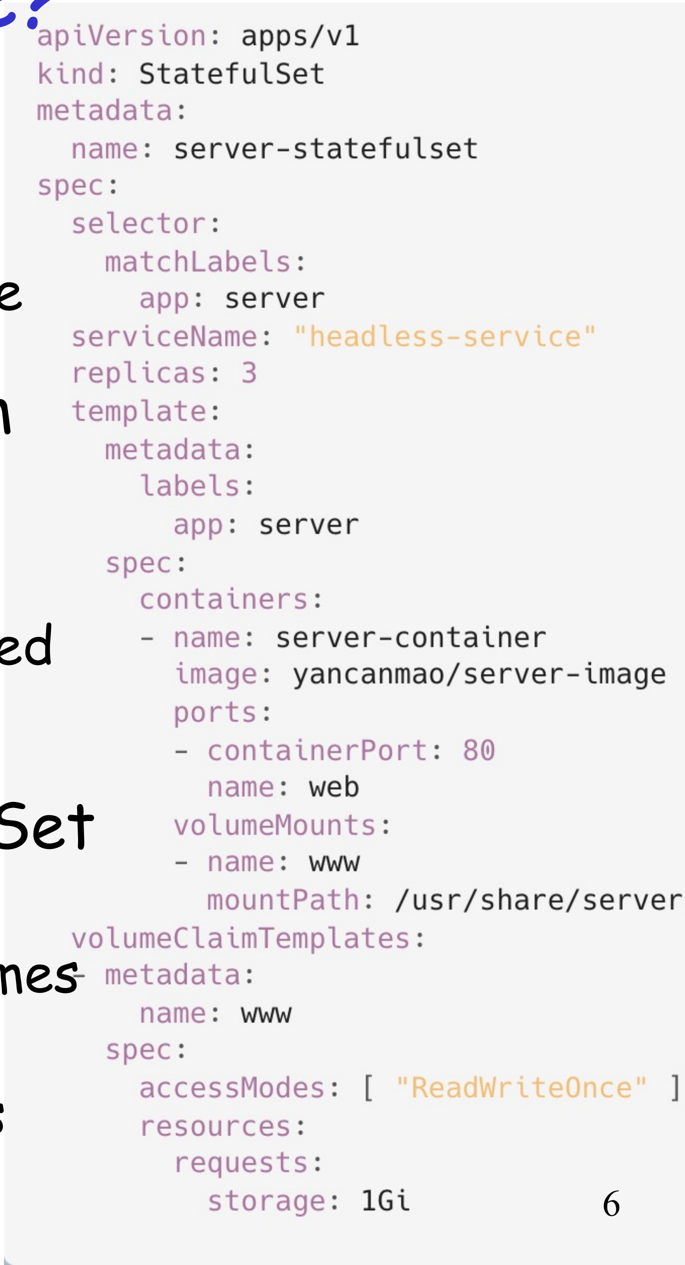
```yaml
apiVersion: v1
kind: Service
metadata:
  name: headless-service
spec:
  clusterIP: None
  selector:
    app: server
  ports:
  - port: 80
```

- Usage case:
  - direct access to the individual pods without load balancing

- How does it work?
  - headless-service.default.svc.cluster.local will resolve to multiple IPs, one for each Pod.
  - Pod-name.headless-service.default.svc.cluster.local will resolve to the specific Pod's IP.

# StatefulSet - how to use?

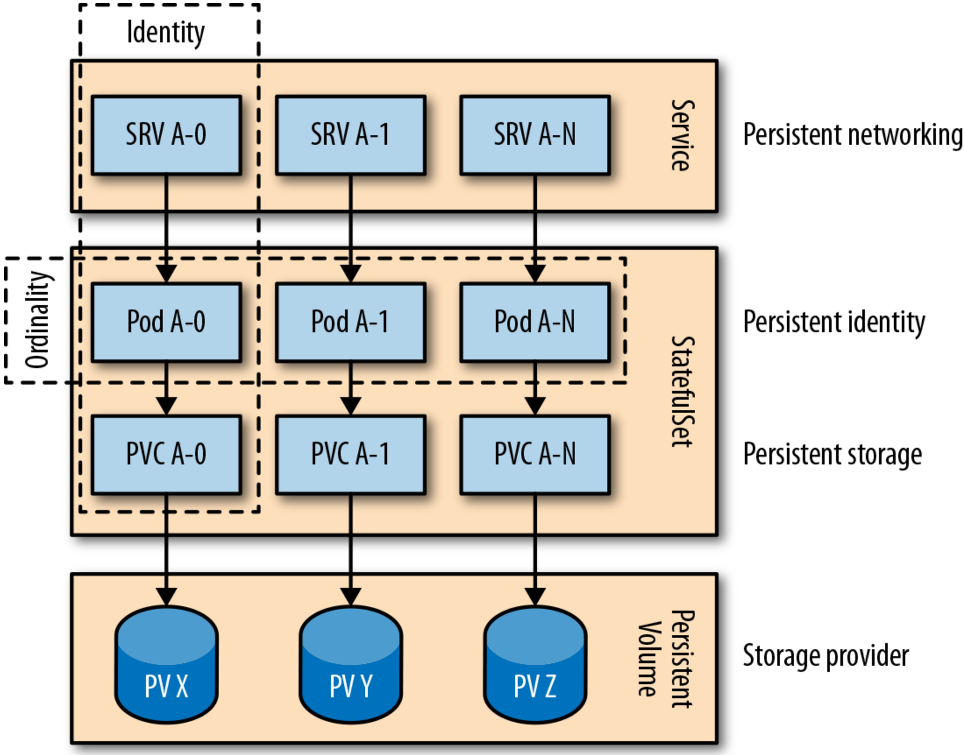
- ❑ Step 2: create a StatefulSet
  - ❖ serviceName matches headless service
- ❑ volumeClaimTemplates mechanism
  - ❖ specifies storage requirements
  - ❖ creates PVCs on the fly during
  - ❖ allows each Pod to get its own dedicated PVC during pod creation
- ❑ In contrast, Deployment & ReplicaSet
  - ❖ use a predefined PVC, suited for using ReadOnlyMany or ReadWriteMany volumes mounted on multiple replicas
  - ❖ not suited for ReadWriteOnce volumes



```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: server-statefulset
spec:
  selector:
    matchLabels:
      app: server
  serviceName: "headless-service"
  replicas: 3
  template:
    metadata:
      labels:
        app: server
    spec:
      containers:
      - name: server-container
        image: yancanmao/server-image
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/server
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi
```

# StatefulSet - Characteristics

- STS does not manage PV
  - but manages PVCs
  - scaling up creates new Pods and associated PVCs.
  - scaling down deletes the Pods, but it does not delete any PVCs (nor PVs)
- K8s cannot free the claimed/used PV storage
  - manual deletion is needed
  - a system behavior by design

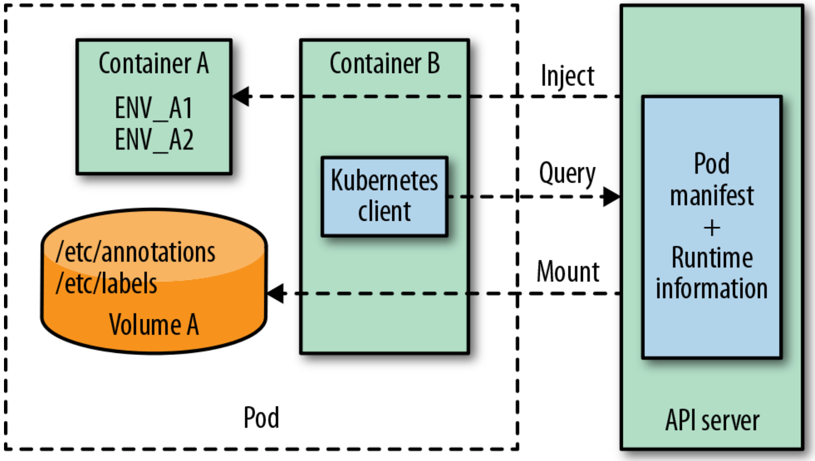


# The need of self-awareness

- Scenario: apps may need to have info about themselves and their running environment
  - runtime info: Pod's name & IP, Node's hostname
  - static info: specific resource requests & limits
  - dynamic info: annotations and labels
- Use cases:
  - log information, send metrics to a central server.
  - tune thread-pool size, GC algorithm or memory allocation based on resource limits
  - discover other pods and interact with them
- Solution: Downward API
  - allows passing metadata about the Pod to the containers and the cluster through environment variables and files

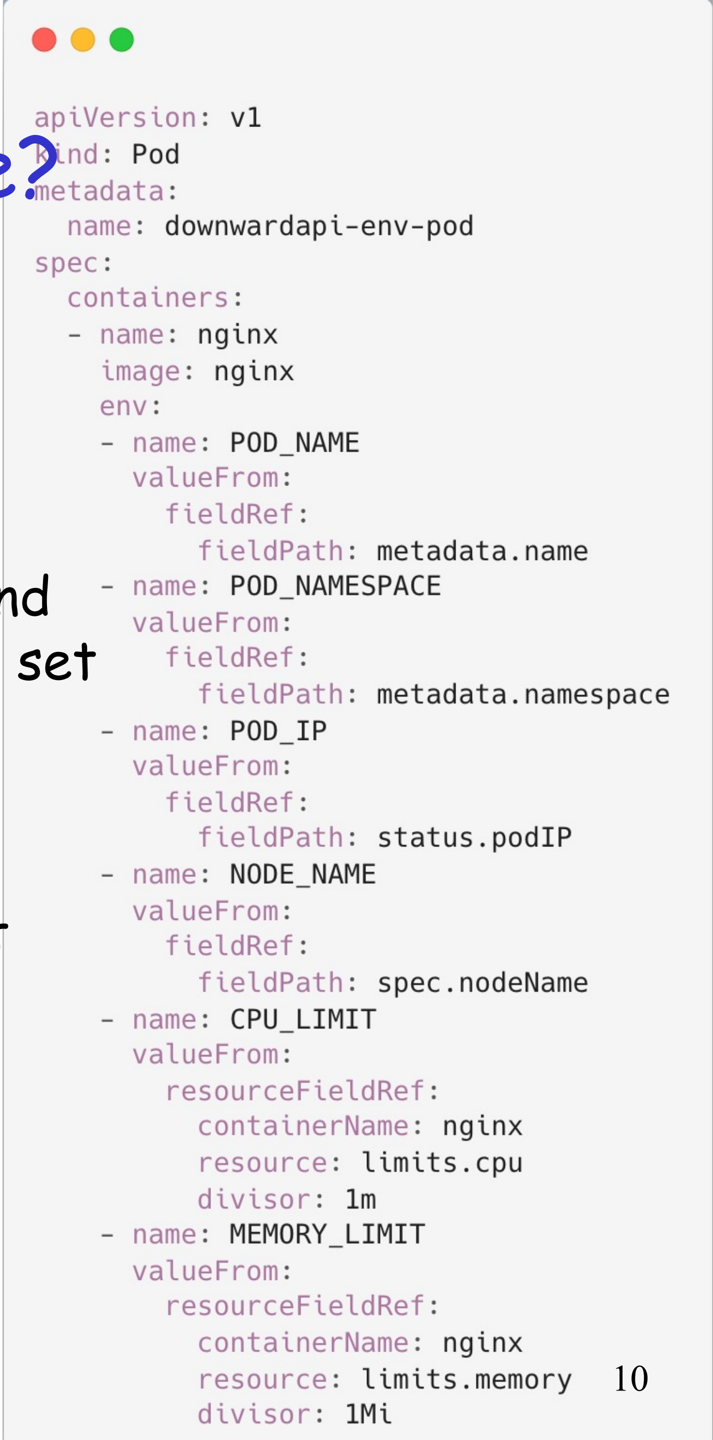
# Downward API - how does it work?

- Same mechanisms for passing data from ConfigMaps
  - data is not created by developers
  - specify the keys that interests us, and K8s populates the values dynamically
- Main advantage:
  - metadata is injected into Pod and made available locally
  - no need to use a client and interact with the API server
  - nonintrusive introspection and metadata injection, remain K8s-agnostic



# Downward API - how to use?

- ❑ Import as environment variables
- ❑ fieldPath:fieldPath option:
  - ❖ POD_NAME, POD_NAMESPACE, POD_IP, and NODE_NAME environment variables are set using the Downward API.
- ❑ ResourceFieldRef option:
  - ❖ CPU_LIMIT and MEMORY_LIMIT are set using container resource limits.



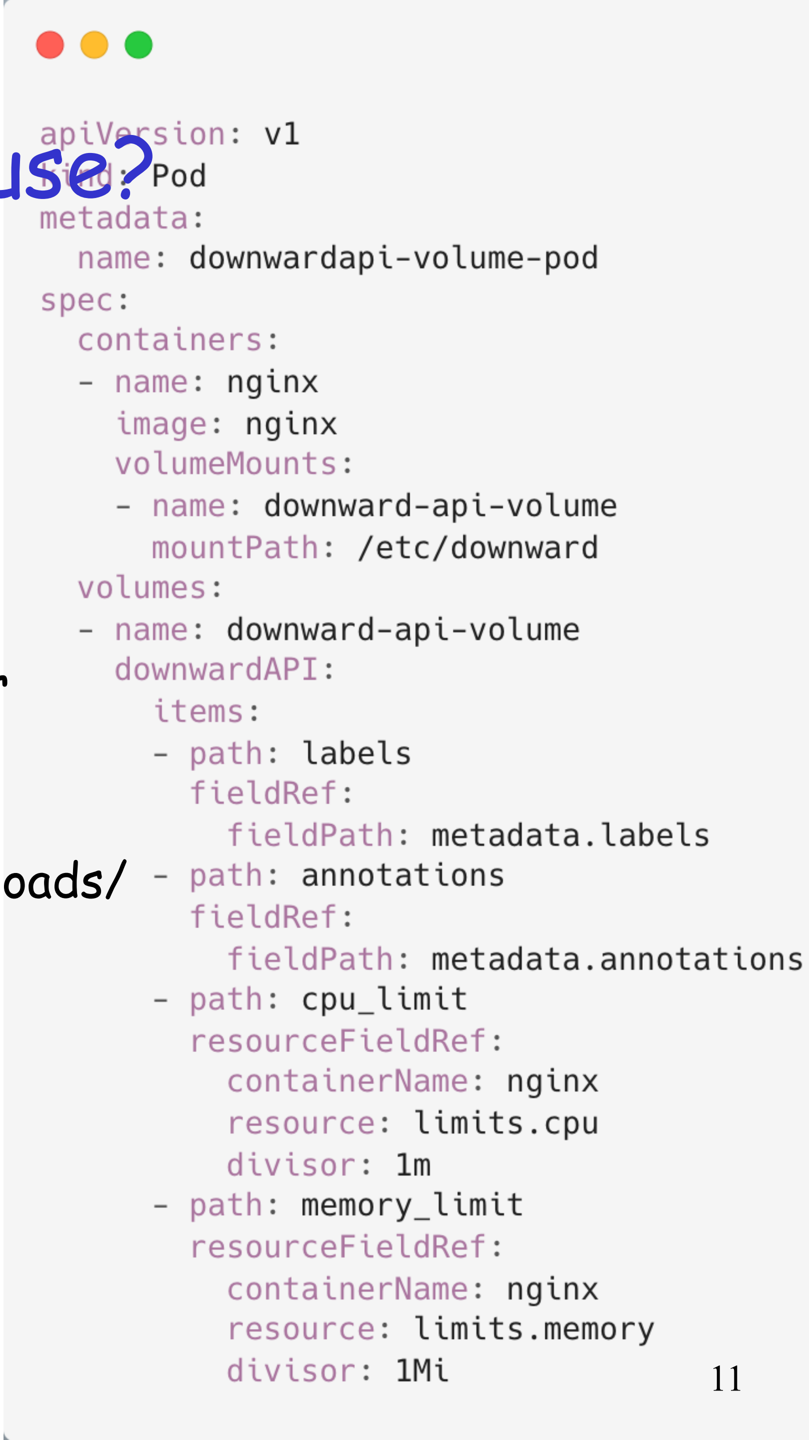
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: downwardapi-env-pod
spec:
  containers:
  - name: nginx
    image: nginx
    env:
    - name: POD_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: POD_NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
    - name: POD_IP
      valueFrom:
        fieldRef:
          fieldPath: status.podIP
    - name: NODE_NAME
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
    - name: CPU_LIMIT
      valueFrom:
        resourceFieldRef:
          containerName: nginx
          resource: limits.cpu
          divisor: 1m
    - name: MEMORY_LIMIT
      valueFrom:
        resourceFieldRef:
          containerName: nginx
          resource: limits.memory
          divisor: 1Mi
```

# Downward API - how to use?

- Import as a volume
  - downwardAPI type of volume
  - all information written into files
  - all the labels and annotations retrieved as files, not for EnvVar
- Available information:

https://kubernetes.io/docs/concepts/workloads/pods/downward-api/

- Limitations of downward API:
  - limited info, some can only be accessed by one method



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: downwardapi-volume-pod
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - name: downward-api-volume
      mountPath: /etc/downward
  volumes:
  - name: downward-api-volume
    downwardAPI:
      items:
      - path: labels
        fieldRef:
          fieldPath: metadata.labels
      - path: annotations
        fieldRef:
          fieldPath: metadata.annotations
      - path: cpu_limit
        resourceFieldRef:
          containerName: nginx
          resource: limits.cpu
          divisor: 1m
      - path: memory_limit
        resourceFieldRef:
          containerName: nginx
          resource: limits.memory
          divisor: 1Mi
```