Kubernetes Training

Paolo Radaelli

Certified Kubernetes Application Developer (CKAD)

- This training is based on the exam program for CKAD
- https://www.cncf.io/certification/ckad/

Logistics

- Sunny Bikes and practice exercises
- Material starting point is no experience with Kubernetes
- Some experience with Docker required

Let's take a look at the repo! 👀

Setting up access to GKE

You need a Google Cloud Platform account.

Locally you need:

- gcloud
- kubectl

Setting up access to GKE

Tooling setup is detailed in training repo `README.md`

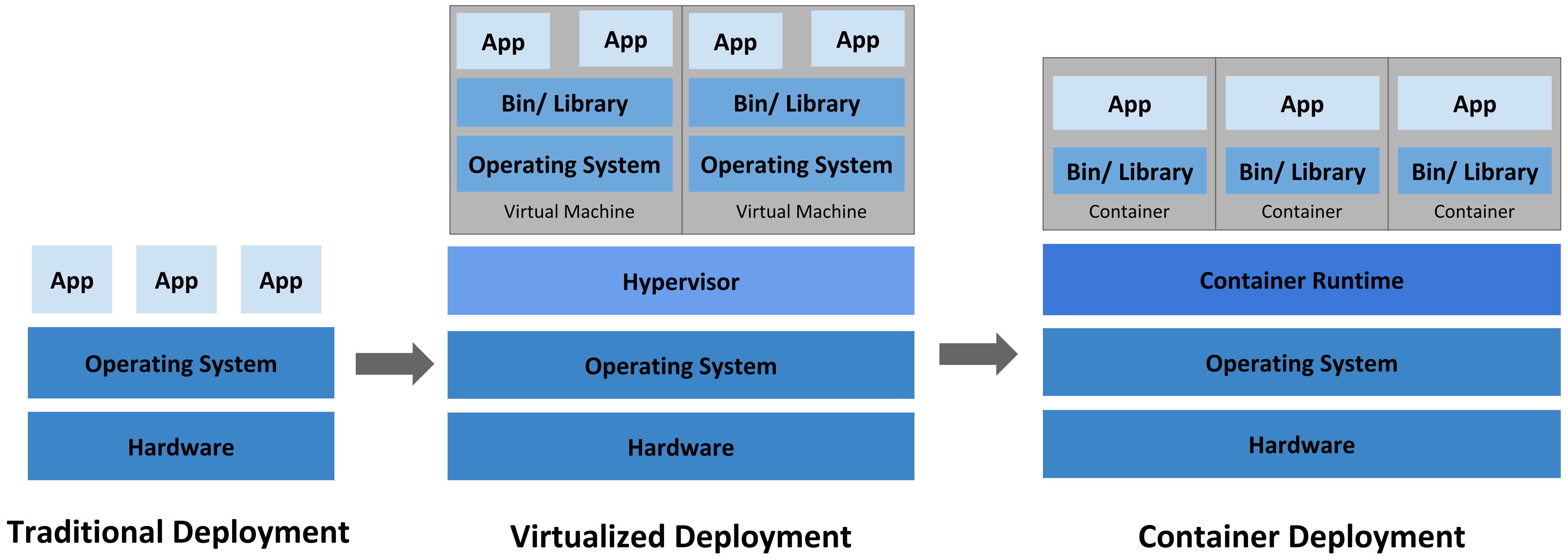
Clusters have been created in GCP Project "Training ProRail - paoloradaelli", project ID prorailk8s-3d278eed53bab6a412ad0.

Your cluster will be called `training-<your name>`.

Let's take some time to ensure everyone has access!

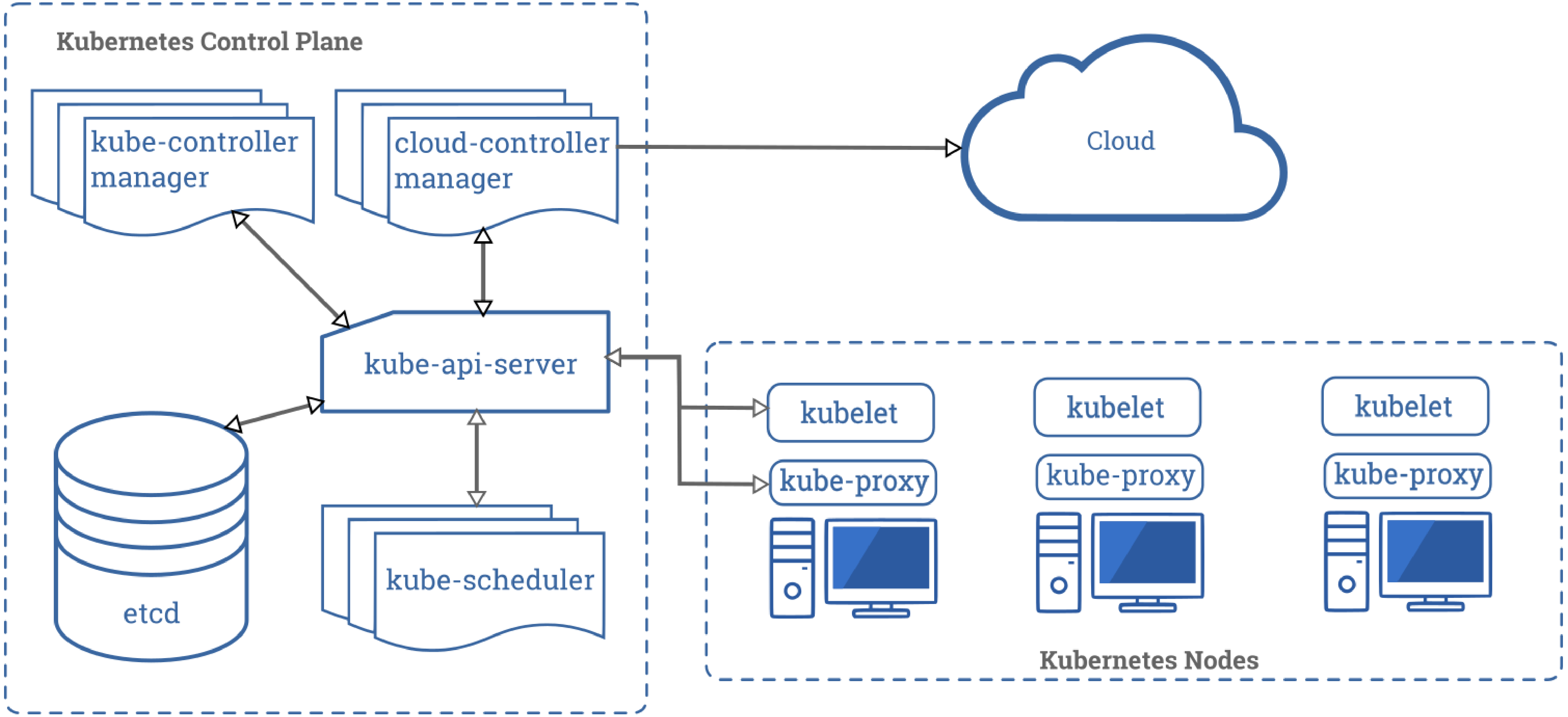
Introduction and theory

Containers FTW



Introducing Kubernetes

"Kubernetes is an open-source container orchestration system for automating the deployment, scaling and management of containerized applications."


Kubernetes Control Plane
kube-controller manager
cloud-controller manager
Cloud
kube-api-server
etcd
kube-scheduler
kubelet
kube-proxy
kubelet
kube-proxy
kubelet
kube-proxy
Kubernetes Nodes

kubectl

- Talks to kube-api-server
- CLI for managing entire clusters
- Check out the kubectl cheatsheet (link in notes)

Kubectl [command] [type] [name] [flags]

- Command [get | create | describe]
- Type: [pod | namespace | node]
- Name: name of the resource
- Flags: optional flags
- https://kubernetes.io/docs/reference/kubectl/#operations
- https://kubernetes.io/docs/reference/kubectl/#resource-types

Kubernetes Documentation ❤️

Exercises

Remember this one? Sunny Bikes

Sunny bikes is a bike sharing company. Currently sharing bikes in Austin (Texas), New York and San Francisco.

Their system consists of several components written in Python, but they’re facing deployment issues because all components work with different versions and have different dependencies.

Sunny Bikes goes Kubernetes!

During the training the Sunny Bikes solution will be transformed into a Kubernetes implementation.

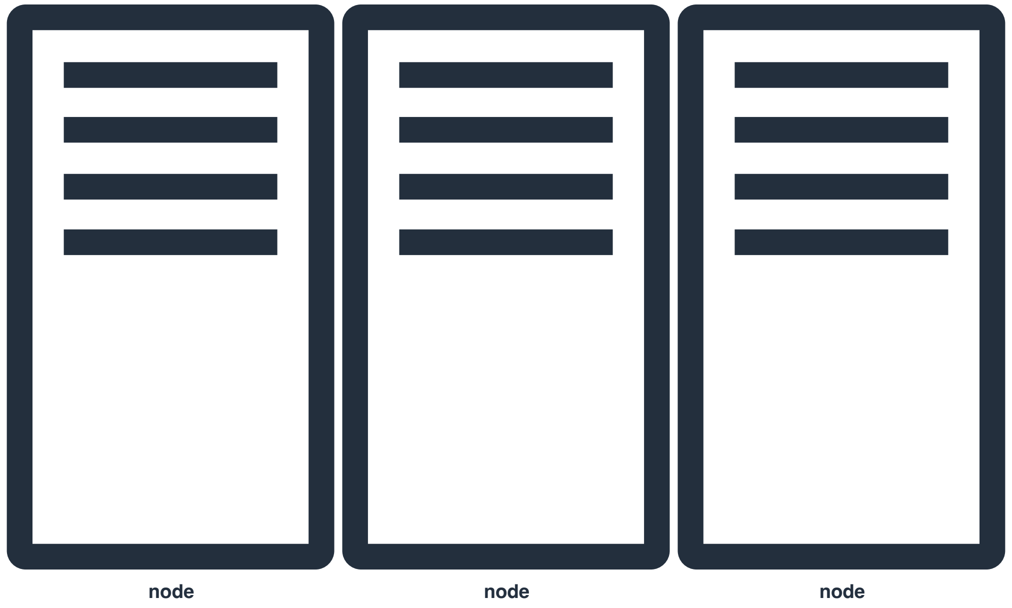
Core concepts

Kubernetes Objects

- Every interaction with kubectl works on objects
- All "things" in Kubernetes are objects in the kube-api-server:
    - Pods
    - Namespaces
    - ConfigMaps
    - Nodes
    - RBAC
    - Jobs

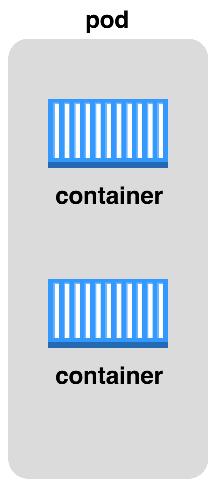
Nodes

- virtual or physical machine
- contains services for running pods

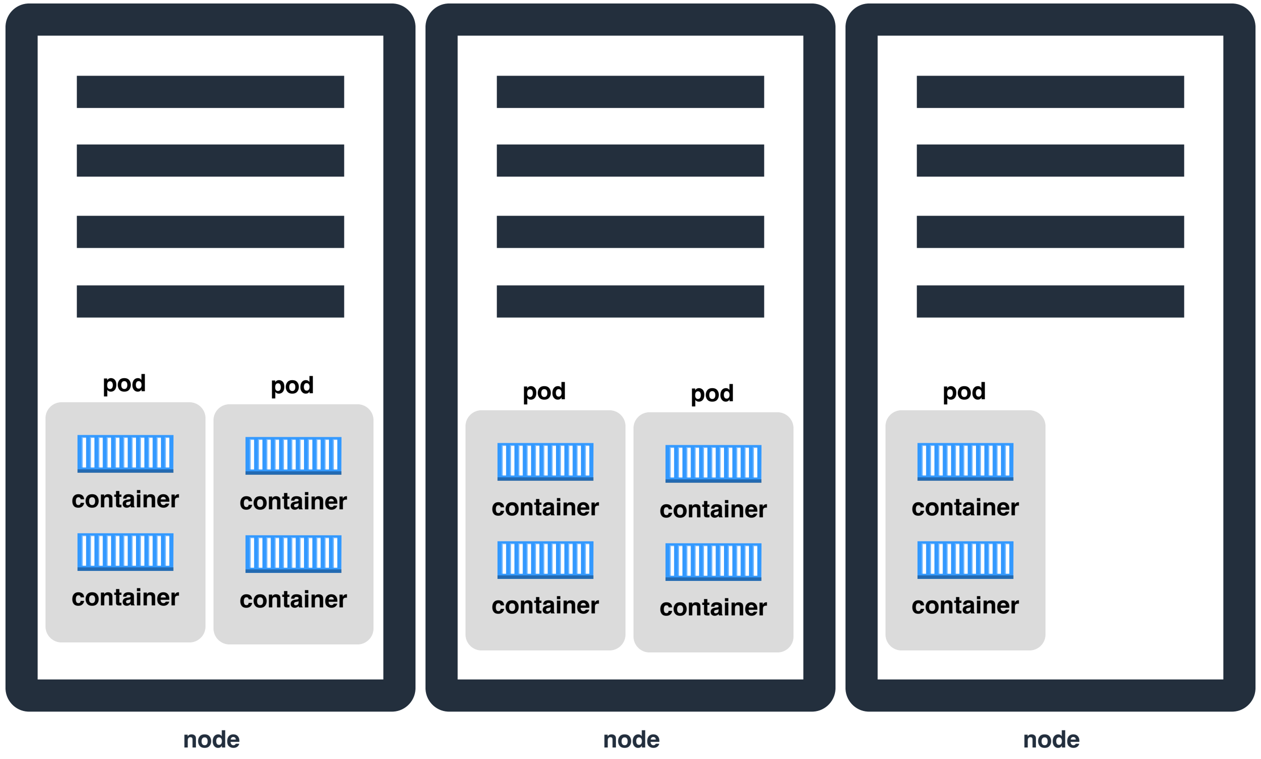


Pods

- basic unit of Kubernetes
- keep container(s) running
- shared storage and network
- usually one main application
- each pod has own IP



Pods run on Nodes



Pod

- Create yaml file "pod.yaml" (or any other name)
- Deploy to Kubernetes with "kubectl apply –f pod.yaml"

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: myapp-container
    image: busybox:stable
    command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']
```

Namespace

- "group" for objects
- default is `default`
- allows resource quotas and access isolation

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: my-namespace
```

- Ports
- restartPolicy

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  restartPolicy: Never
  containers:
  - name: myapp-container
    image: busybox:stable
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo hello; sleep 10;done"]
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  restartPolicy: Always
  containers:
  - name: myapp-container
```

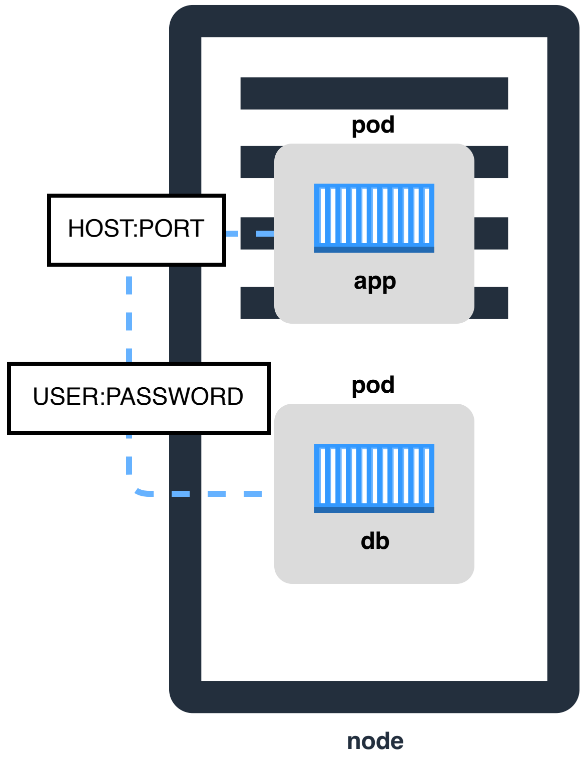
Exercise 1

1. Create a yaml file with a namespace definition for sunnybikes
2. Create a yaml file with a pod definition for the sunnybikes application. For this, use the docker image `pugillum/sunnybikes:stable`.
3. Create a yaml file with a pod definition for the postgres application. For this, use the docker image `postgres:11-alpine`.
4. Apply the configurations and check the logs of the pods with: `kubectl -n sunnybikes logs <podname>`
   Check the logs to see if everything is started correctly!

Configuration

Config Maps & Secrets

- configuration with ConfigMap
- Secret for secret data
- environment variables or properties file



Environment variables

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod-env
spec:
  restartPolicy: Never
  containers:
    - name: busybox
      image: busybox:stable
      command: ["sh", "-c", "echo $MESSAGE;"]
      env:
        - name: MESSAGE
          value: "Hello!"
```

```
kind: ConfigMap
metadata:
  name: my-configmap
data:
  # key value style
  message: hello
  name: John

  # file style
  app.cfg: |
    key1=value1
    key2=value2
```

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod-cf
spec:
  restartPolicy: Never
  containers:
    - name: busybox
      image: busybox:stable
      command: ["sh", "-c", "echo $MESSAGE $NAME;"]
      env:
        - name: MESSAGE
          valueFrom:
```

```
  name: my-configmap-volume-pod-1
spec:
  restartPolicy: Never
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', "echo $(cat /etc/config/hello)"]
    volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: my-config-map
```

```
apiVersion: v1
kind: Pod
metadata:
  name: my-configmap-volume-pod-2
spec:
  restartPolicy: Never
  containers:
  - name: busybox
    image: busybox:stable
    command: ["sh", "-c", "cat /config/app.cfg"]
      volumeMounts:
        - name: config
```

- ⚠️ beware of caveats - see k8s documentation

```
echo Secret Stuff! -n | base64
U2VjcmV0IFN0dWZmMQo=
```

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
data:
  myKey1: U2VjcmV0IFN0dWZmMQo=
stringData:
  myKey2: myPassword
```

```
apiVersion: v1
kind: Pod
metadata:
  name: my-secret-pod
spec:
  containers:
    - name: myapp-container
      image: nginx:stable
      env:
        - name: MY_PASSWORD
```

Resource requirements

- `limits` : Restriction on resources for pods
- `requests` : Requests on resource for pods

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-resource-pod
spec:
  restartPolicy: Never
  containers:
    - name: nginx
      image: nginx:stable
      resources:
        requests:
          memory: "64Mi"
          cpu: "250m"
        limits:
          memory: "128Mi"
          cpu: "500m"
```

- Role: define what is allowed
- RoleBinding: tie Roles to users or service accounts
- Varies per cloud provider

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-reader
  namespace: default
rules:
  - apiGroups: [""] # don't need special API group
    resources: ["pods"] # type of k8s object
    verbs: ["get","watch","list"] # what we can do
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pod-reader-binding
subjects: # provides list of accounts that this role binding is addressing
  - kind: User
    name: imauser@k8sworkshop.com
    namespace: default
roleRef: # which role we're binding subjects to
```

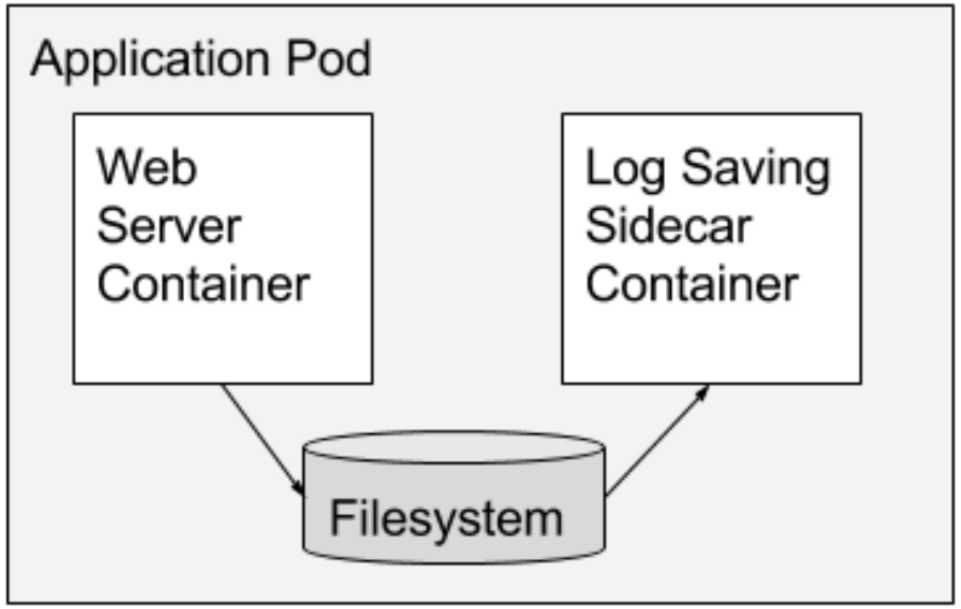
Exercise 2

1. Define an environment variable for the app pod `PG_PORT` with value `"5432"`
2. Create a yaml file with a secret definition for the postgres password. Add the secret as environment variable to the sunny and postgres pods. SunnyBikes needs a `PG_PASSWORD` env variable and postgres needs a `POSTGRES_PASSWORD` env variable
3. Create a yaml file with a config map definition for the postgres init schema. Mount the init script in the postgres pod
   Full details in sunny_bikes_exercises.md
4. [Optional] Create a role and binding that only allows "list" on secrets in the `sunnybikes` namespace for your user.

Multi-container pods

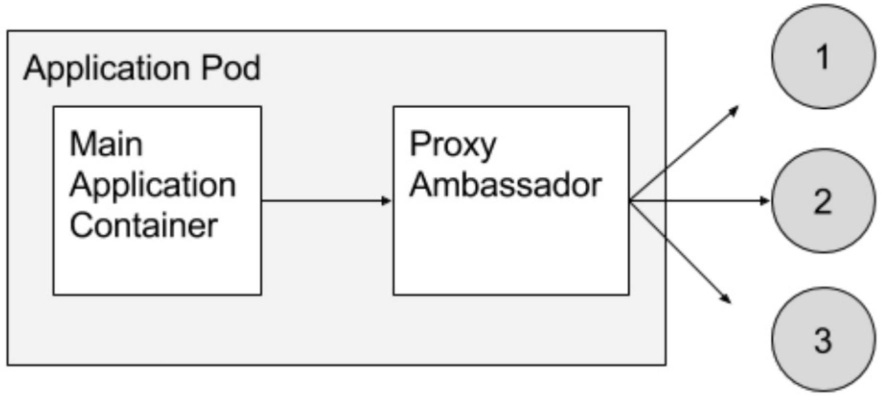
Design patterns

- Sidecar containers



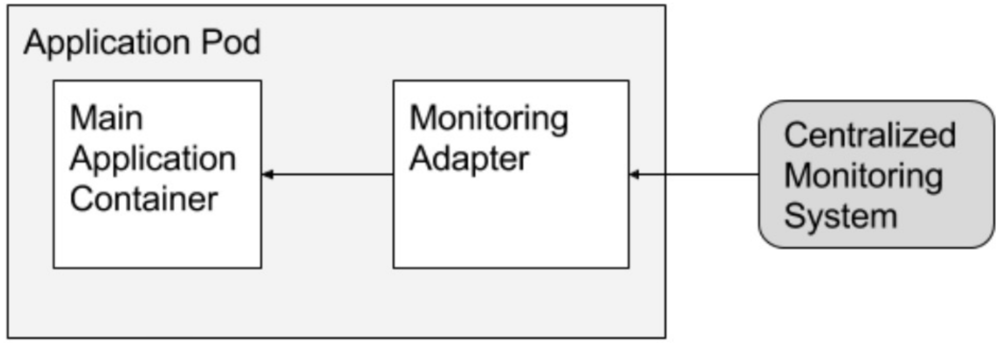
Design patterns

- Ambassador containers
- Proxy a local connection to the world.



Design patterns

- Adapter containers
- Standardize and normalize outputs



Observability

Liveness probe

- Can customize how Kubernetes detects the status of **containers**
- Indicates if a container is running properly and governs when the cluster will automatically stop or restart the container.
- Health status

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-liveness-pod
spec:
  containers:
  - name: myapp-container:stable
    command: ["sh", "-c", "while true; do sleep 10; done"]
    livenessProbe:
      exec:
        command:
        - echo
        - testing
      initialDelaySeconds: 5
      periodSeconds: 5
```

Readiness probe

- Can customize how Kubernetes detects the status of **containers**
- Indicates whether a container is ready to service requests and governs whether requests will be forwarded to the pod.
- Ready to serve status

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-readiness-pod
spec:
  containers:
  - name: nginx
    image: nginx:1.20.1
    readinessProbe:
      httpGet:
        path: /
        port: 80
      initialDelaySeconds: 5
      periodSeconds: 5
```

Container logs

```
# read logs from a specific container
kubectl logs my-pod -c my-container

# output to a file
kubectl logs my-pod -c my-container > my-container.log

# stream logs
kubectl logs -f my-pod
```

Metrics

```
# see CPU and memory per pod
kubectl top pods

# specific pod
kubectl top pod my-resource-pod

# kube-system
kubectl top pods -n kube-system

# the nodes
kubectl top nodes
```

✨ This requires installation of metrics server

- Kubectl commands like `get`, `describe`, `edit`

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: nginx-ns
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: nginx-ns
spec:
  containers:
    - name: nginx
      image: nginx:1.20.1
```

```
# list pods from all namespaces
kubectl get pods --all-namespaces

# get details about a pod
kubectl describe pod nginx -n nginx-ns

# edit a pod manifest
```

Extra note

✨ Some specifications are immutable and cannot be edited or updated. Delete and recreate the pod if this happens.

Exercise 3

1. Sunny Bikes has a "/healthz" endpoint that returns status code 200 if the service is up. Use that for a readiness Probe

Pod design

- Useful for increasing search efficiency

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-production-label-pod
  labels:
    app: my-app
    environment: production
spec:
  containers:
    - name: nginx
      image: nginx
---
apiVersion: v1
kind: Pod
metadata:
  name: my-development-label-pod
  labels:
    app: my-app
    environment: development
spec:
  containers:
    - name: nginx
      image: nginx
```

```
# get pods with specific label filters
kubectl get pods -l app=my-app
kubectl get pods -l environment=production
kubectl get pods -l environment=development
```

Annotations

- Used to annotate objects
- Cannot be searched

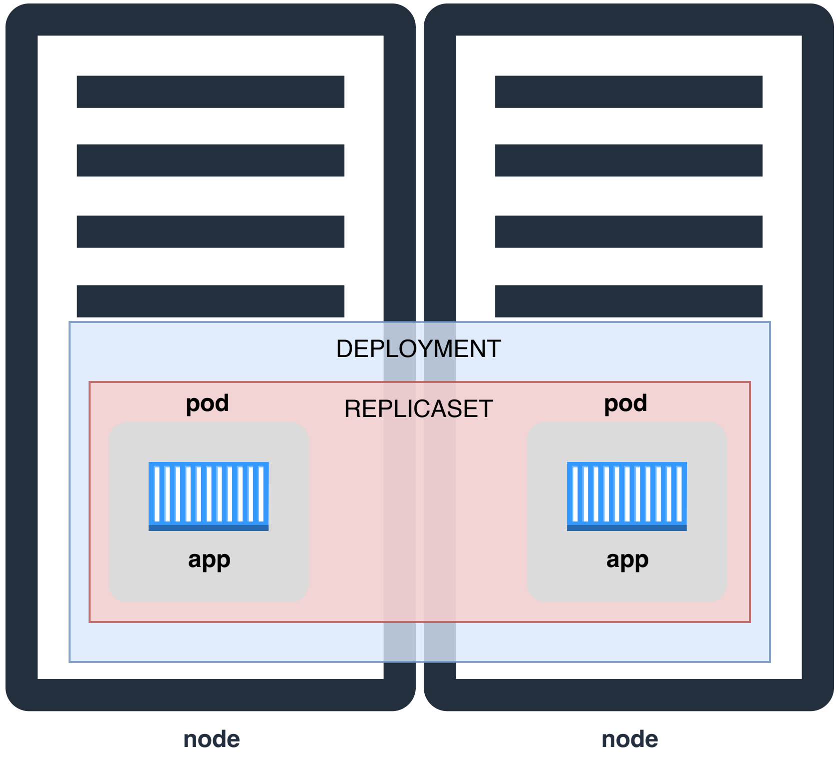
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-annotation-pod
  annotations:
    owner: myaccount@k8sisgr8.com
    git-commit: bdab0c6
spec:
  containers:
  - name: nginx
    image: nginx
```

Deployments

- abstract
- keeps Pods alive
- replicates Pods

Deployments

- creates Replicasets
- desired state configuration



Deployments

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

```
# check deployment
kubectl get deployment
```

- Various strategies, default RollingUpdate

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rolling-deployment
spec:
  strategy:
    rollingUpdate:
      maxSurge: 3 # max pods that can be scheduled above desired number
      maxUnavailable: 2 # max pods that can be unavailable during update
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.1
        ports:
        - containerPort: 80
```

```bash
# set image version for specific deployment
kubectl set image deployment/rolling-deployment nginx=nginx:1.7.9
```

Rollback

```
# rollback to previous deployment
kubectl rollout undo deployment/rolling-deployment

# rollback to a specific version
kubectl rollout undo deployment/rolling-deployment --to-revision=1
```

Exercise 4

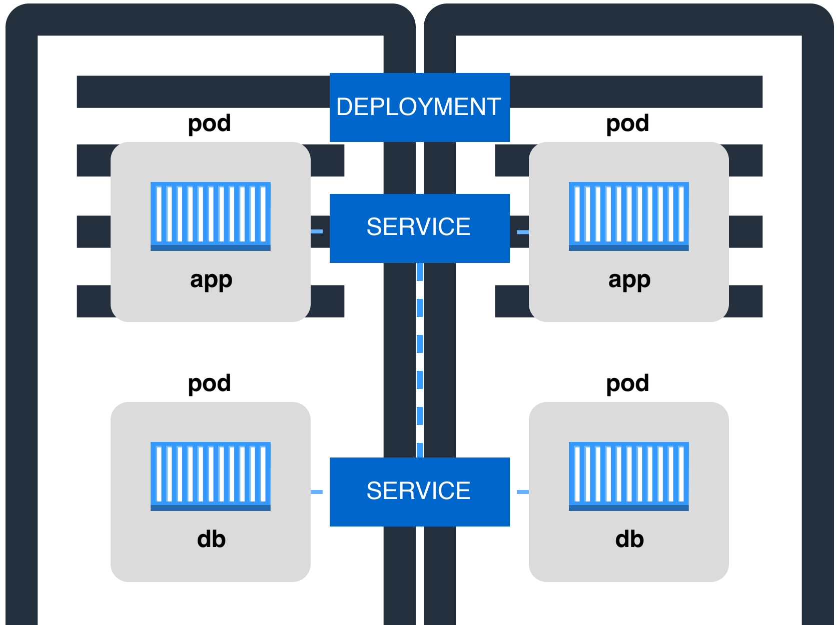
1. Migrate the pod definitions for Sunny Bikes and postgres to a deployment
2. The Sunny Bikes pods must be replicated at least 3 times
3. Make sure that when a new update of Sunny Bikes is deployed at least 1 pod is always available during upgrading
4. Add some annotations/labels that you deem useful

Services and networking

Services

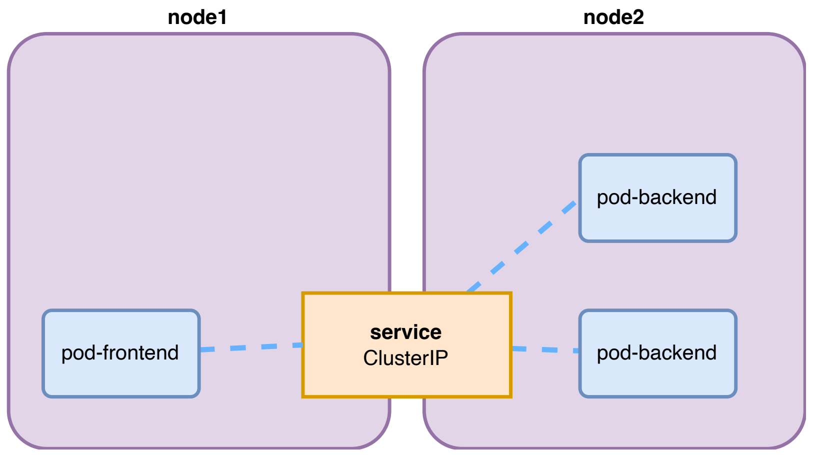
- abstract
- provide connection to Pods
- no messing with IPs

- static IP address
- seamless connection across nodes
- lifecyle detached from pods
- internal and external
- load balancing across pods
- single DNS name
- network policies



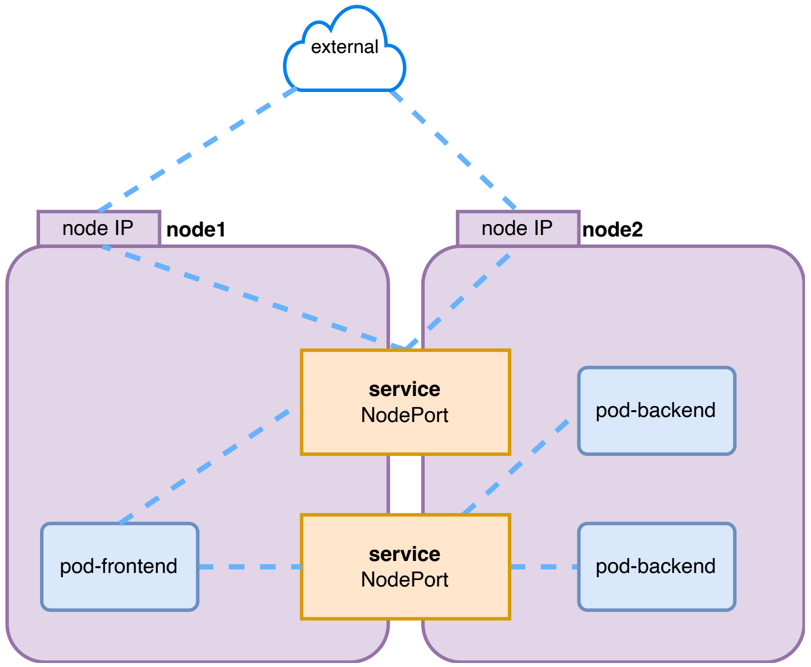
Services - ClusterIP

- Exposes the Service on a cluster-internal IP
- Choosing this value makes the Service only reachable from within the cluster
- This is the default ServiceType



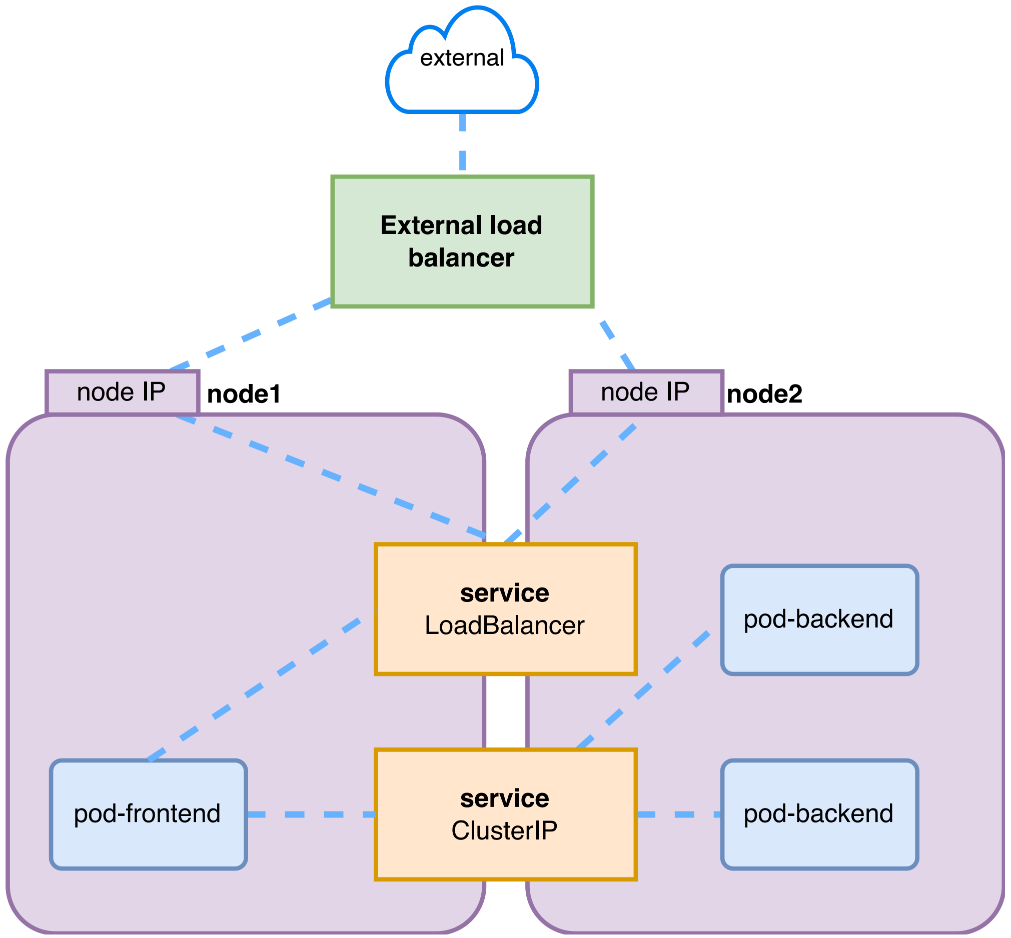
Services - NodePort

- Exposes the Service on each Node's IP at a static port (the NodePort)
- Automatically creates a ClusterIP Service
- From external connect to `<NodeIP>:<NodePort>`



Services - LoadBalancer

Exposes the Service externally using a cloud provider's load balancer. NodePort and ClusterIP Services, to which the external load balancer routes, are automatically created



```yaml
metadata:
  name: my-service
spec:
  type: ClusterIP
  selector:
    app: nginx # determines which pods receive traffic
  ports:
  - protocol: TCP
    port: 8080 # port that services listens on
    targetPort: 80 # port on pod to route traffic to
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nodeport-service
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 80
      nodePort: 30080 # the port to map to on the node
```

Ingress

- can be used to expose *multiple* services
- not a Service type, but acts as entry point
- allows consolidating your routing rules into a single resource
- most powerful, but can also be the most complicated

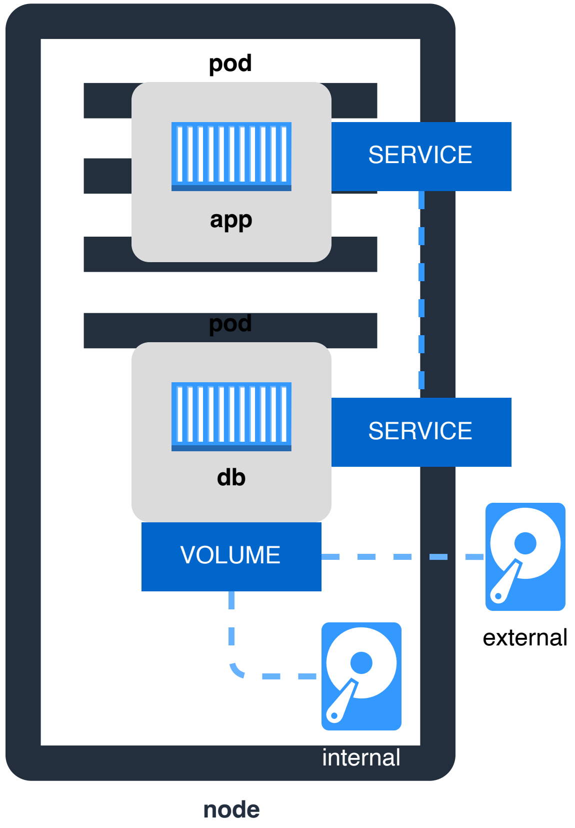
Exercise 5

1. Make Postgres available from only within the cluster to the Sunny Bikes pod.
2. SunnyBikes takes 2 environment variables to configure the postgres host and port, `PG_HOST` and `PG_PORT` respectively.
   Hint: DNS of service will be in form `<service name>.<namespace>.svc.cluster.local`
3. Make Sunny Bikes available on port 80 from outside world
   To test: Swagger docs are available on `http://<ip>:80/docs`

State persistence

Volumes

- pod storage is ephemeral 😕
- volume links to physical storage
- ⚠️ k8s doesn't manage storage



Volumes

- Containers can share storage

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: volume-pod
spec:
  containers:
    - name: busybox1
      image: busybox:stable
      command: ["/bin/sh", "-c", 'echo "The writer wrote this!" > /output/data.txt; while true; do sleep 5; done']
      volumeMounts:
        - mountPath: /output
          name: my-volume
    - name: busybox2
      image: busybox:stable
      command: ["/bin/sh", "-c", "while true; do cat /input/data.txt; sleep 5; done"]
      volumeMounts:
        - mountPath: /input
          name: my-volume
  volumes:
    - name: my-volume
      emptyDir: {} # exists for lifetime of Pod
```

Persistent storage

Persistent Volume:

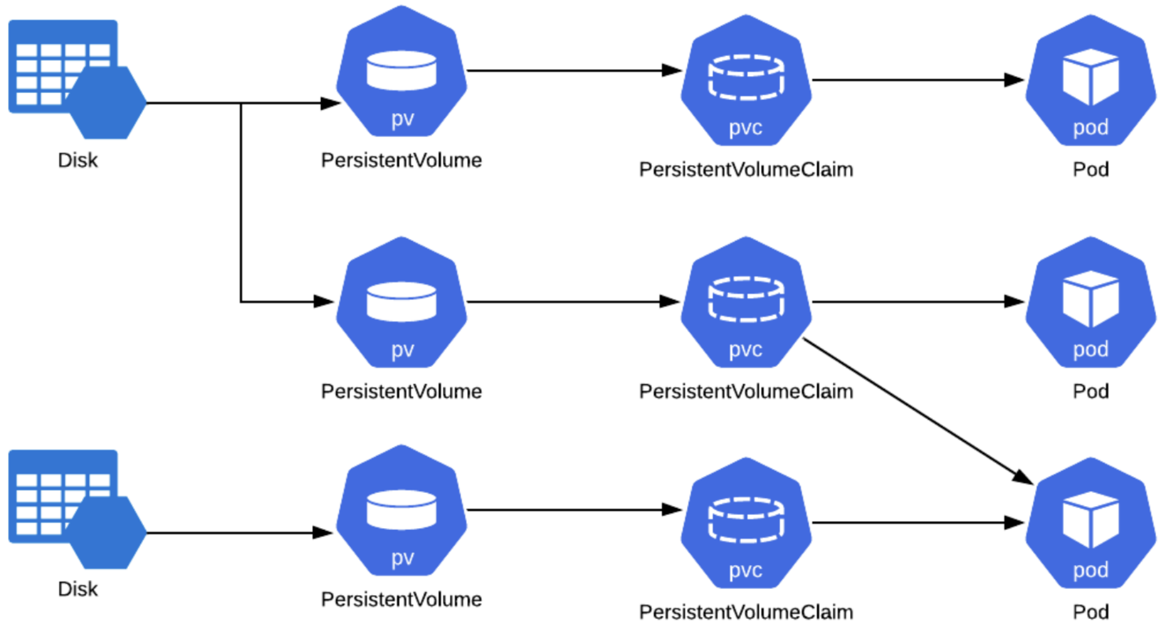
- Piece of storage in the cluster
- Not dependent on Pod lifecycle
- Typically provided by cluster Admin
- Local or Remote - Use remote
- Many types - have specific config
- Not namespace specific

Persistent Volume Claim

- defines a request for storage including details on type of storage needed
- automatically binds to available PersistentVolume that meets provided requirements
- mounted in a pod like a volume (available to all containers)

Persistent storage

- A pod may claim many PersistentVolumeClaims
- A PersistentVolumeClaims may be be used by many pods
- A PersistentVolume may only be claimed by one PersistentVolumeClaim and vice versa
- A disk may have many PersistentVolumes requesting parts of storage



```
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  storageClassName: local-storage
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 512Mi
```

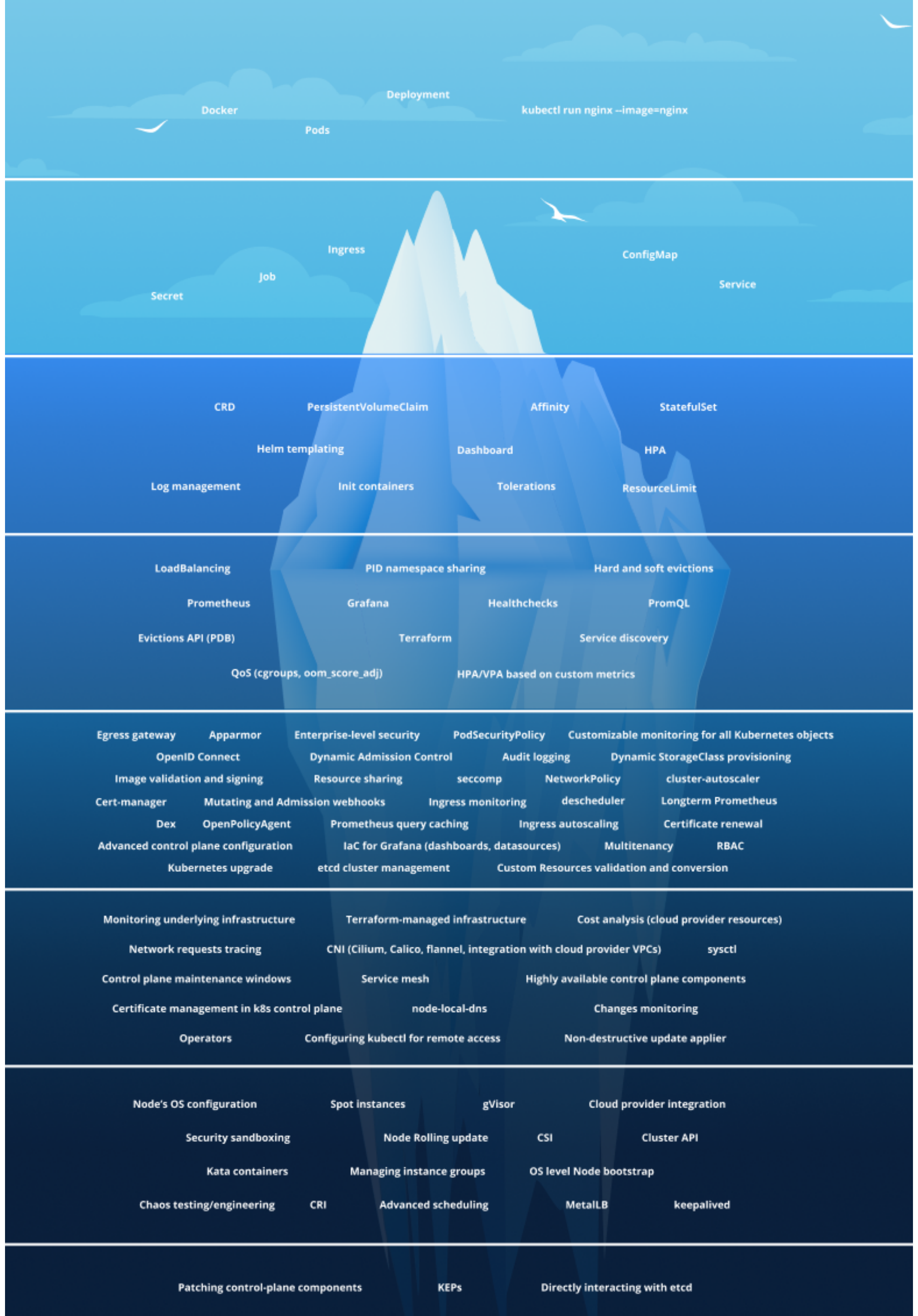
```
apiVersion: v1
kind: Pod
metadata:
  name: my-pvc-pod
spec:
  containers:
```

Exercise 6

If the postgres pod dies,all the data is lost. 😭 Make sure all the postgres data is persistent even throughout pod restarts

Hint: Add data via the Swagger interface
Hint: Postgres stores its data in `/var/lib/postgresql/data`

Verify the volume is working.



So much to learn 🤯

Advanced topics

Security

- security flaws sometimes exist
- there are practices to be safe
- run as non-root as much as possible
- security is hard
- align with IT

Non-root - Python Dockerfile

- run `pip` as root
- `USER` configures user for subsequent `RUN`, `CMD` and `ENTRYPOINT`

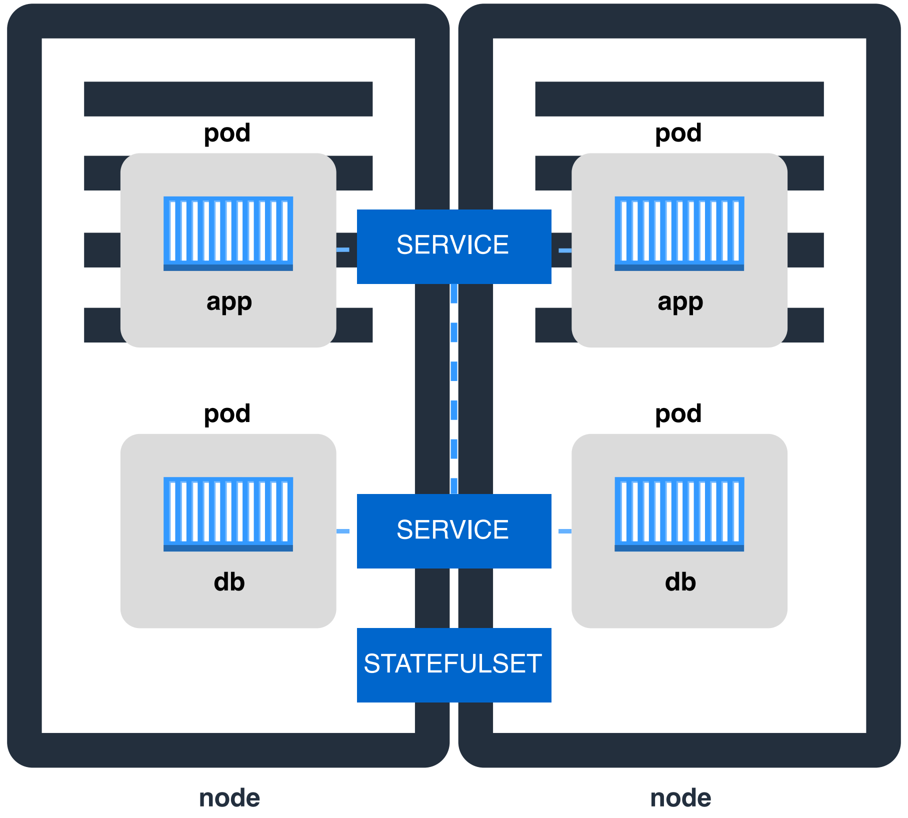
```
FROM python:3.7.2-alpine

RUN pip install --upgrade pip

RUN adduser -D worker
USER worker
WORKDIR /home/worker

COPY --chown=worker:worker requirements.txt requirements.txt
RUN pip install --user -r requirements.txt

ENV PATH="/home/worker/.local/bin:${PATH}"

COPY --chown=worker:worker . .

ENTRYPOINT ["python"]
```

```
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: myapp-container
    image: busybox:stable
    securityContext:
        runAsNonRoot: True
```

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: sec-ctx-vol
    emptyDir: {}
  containers:
  - name: sec-ctx-demo
    image: busybox:1.28
```

StatefulSet

- for stateFUL
- synchronizes reads/writes
- ensures data consistency
- ⚠️ not advised - external storage better



HELM

The package manager for Kubernetes

Helm is the best way to find, share, and use software built for Kubernetes.