Kubernetes Training

Travis Dent

Xebia Academy

Certified Kubernetes Application Developer (CKAD)

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

Logistics

- Clone the Github Repo
- You'll need a text editor and a terminal (for Windows, PowerShell)
- You'll be using Google Kubernetes Engine
- Topic exercises and a use case
- This is a very practical training ⌨️

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

Instructions in the repo `README.md`

Setting up access to GKE

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

Clusters have been created in a GCP Project "[Insert project name]", project ID [Insert project ID].

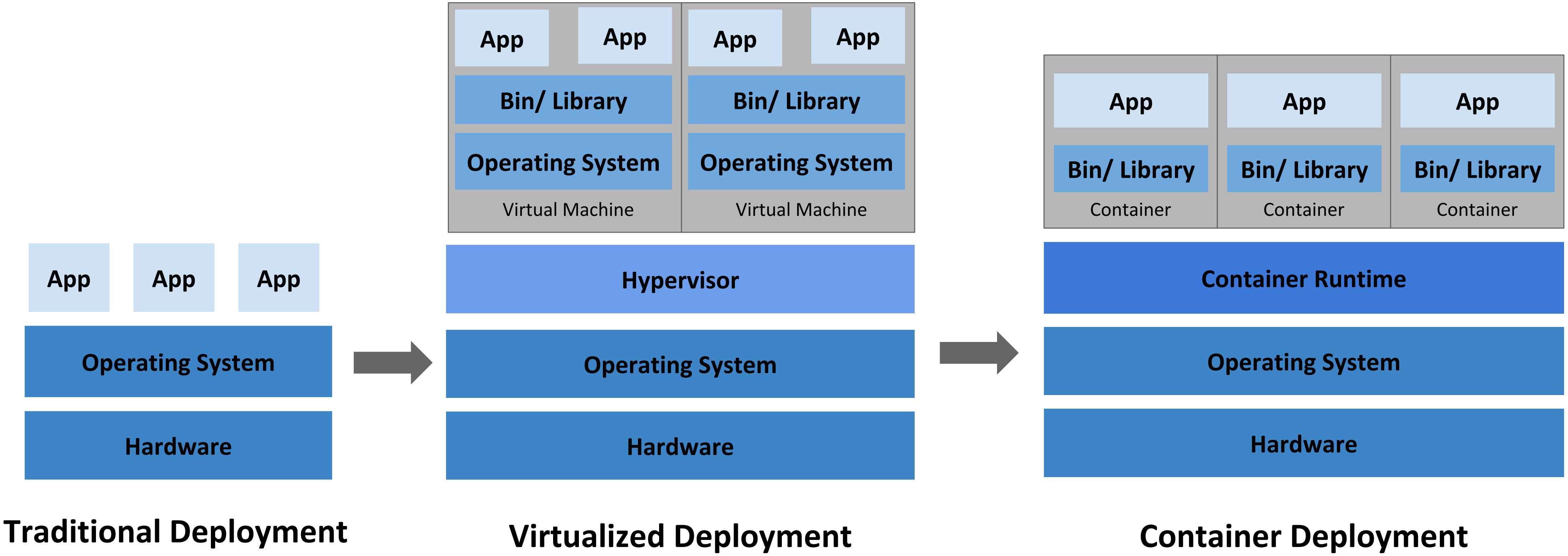
Your cluster will be called `training-<your name>`.

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

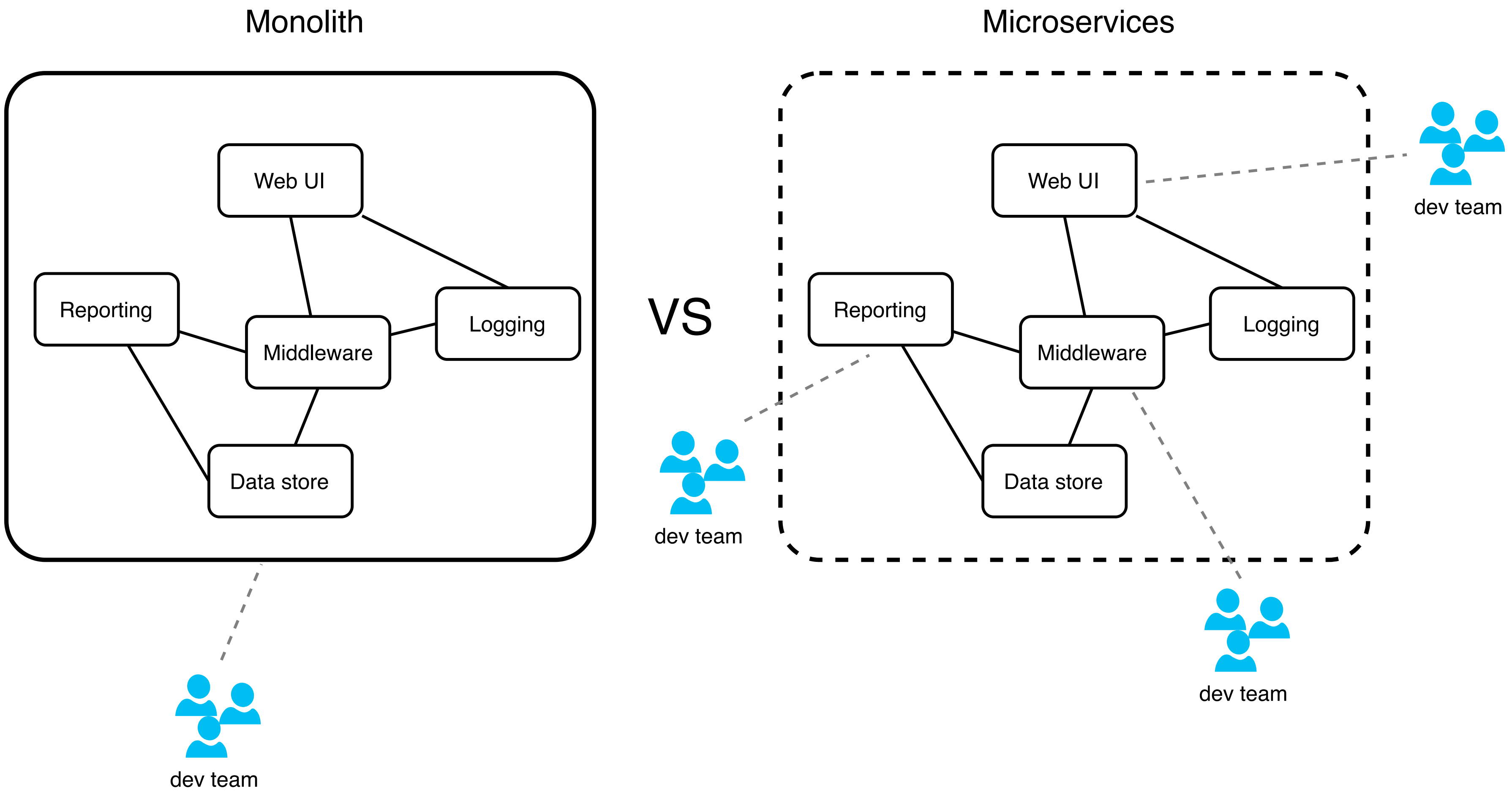
What would you like to learn?

Introduction and theory

Containers FTW



Monoliths vs Microservices



Some history

- Selling servers, network switches, monolith app licenses = big 💰
- 2006: Amazon launches AWS - the birth of cloud computing! 💥
- 2013: Docker initial release
- 2014: Google introduces Kubernetes
- 2015: Kubernetes donated to Cloud Native Computing Foundation
- 2018: Kubernetes GA on AWS (also Azure, Digital Ocean, more..)

Some more history

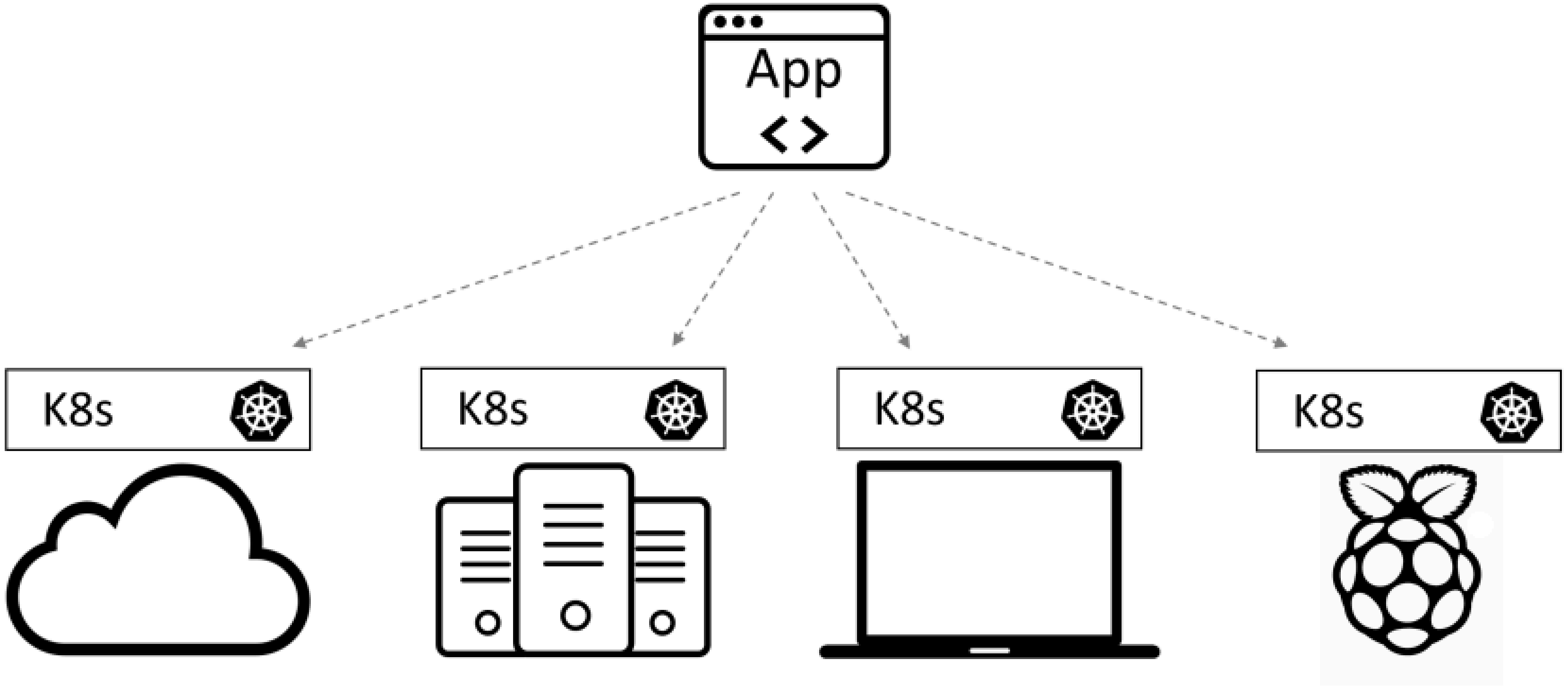
- Google used Linux containers to run servers
- Billions per week
- Used a technology called *Borg* and then *Omega*
- Same engineers built *Kubernetes*
- *Kubernetes* based on κυβερνήτης: helmsman

Kubernetes

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

What makes K8s so attractive?

The OS of the cloud

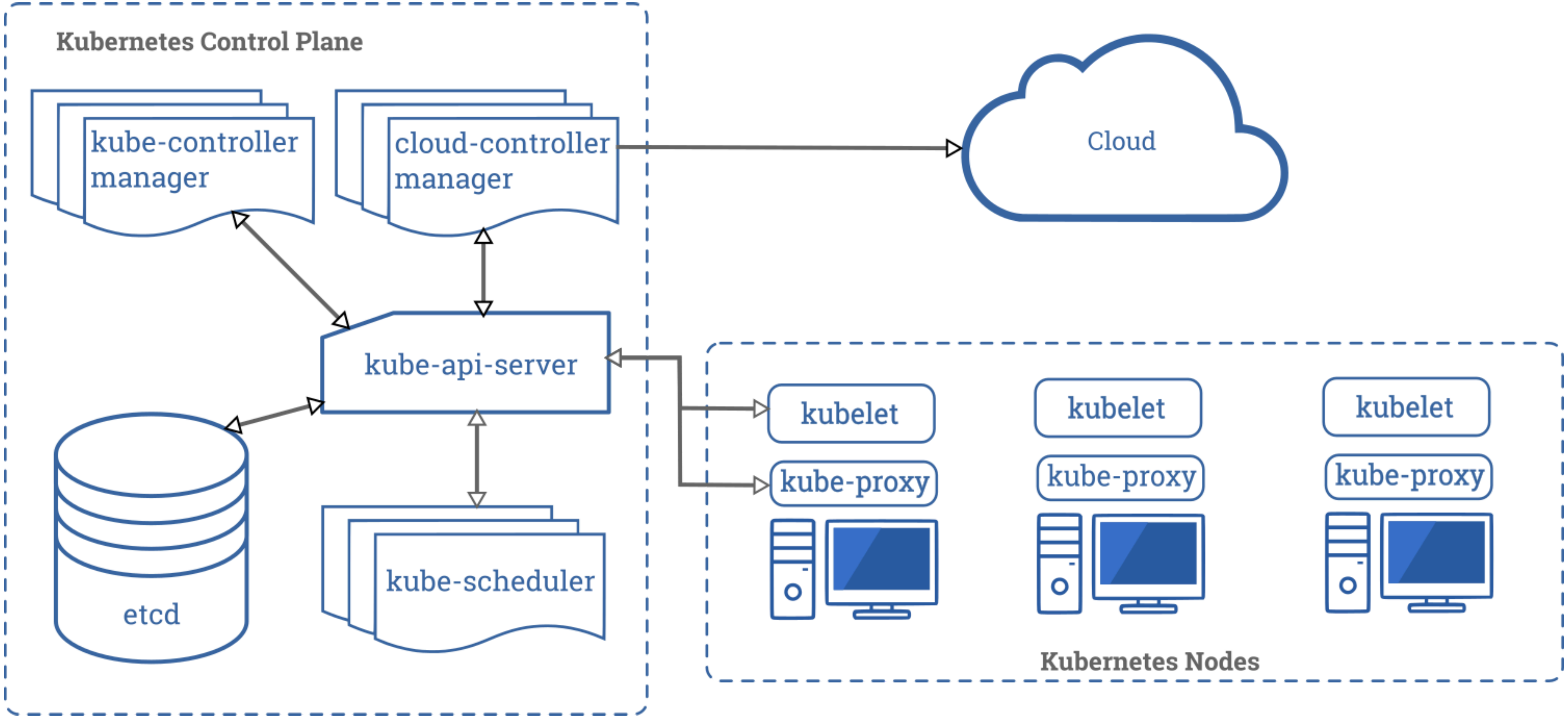


Kubernetes

- Written in Go
- Service discovery and load balancing
- Storage orchestration
- Automated rollouts and rollbacks
- Best use of resources

- Self healing
- Secret and configuration management
- Horizontal scaling
- IPv4/IPv6 dual-stack
- Designed for extensibility

Under the hood



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 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

Kubernetes manifests

- A pod manifest:

```
apiVersion: v1 # schema version
kind: Pod # what kind of object, mapped to REST resource
metaData: # object's metadata
    name: my-pod
    namespace: my-namespace
spec: # Specification of desired behaviour
    containers:
    - name: nginx
      image: nginx:stable
```

- YAML or JSON
- Declarative way to define desired state
- Multiple objects possible (separate by `---`)

Kubernetes Documentation ❤

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.

Namespaces

Namespace

- "group" for objects
- Default is `default`
- Allows resource quotas and access isolation
- Not all objects are "namespaced", try `kubectl api-resources`

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

Pods

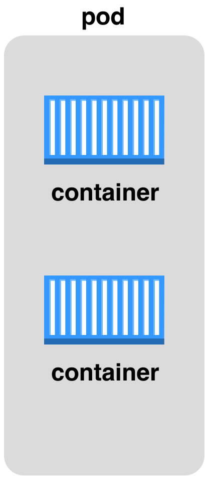
Nodes

- virtual or physical machine
- contains services for running pods

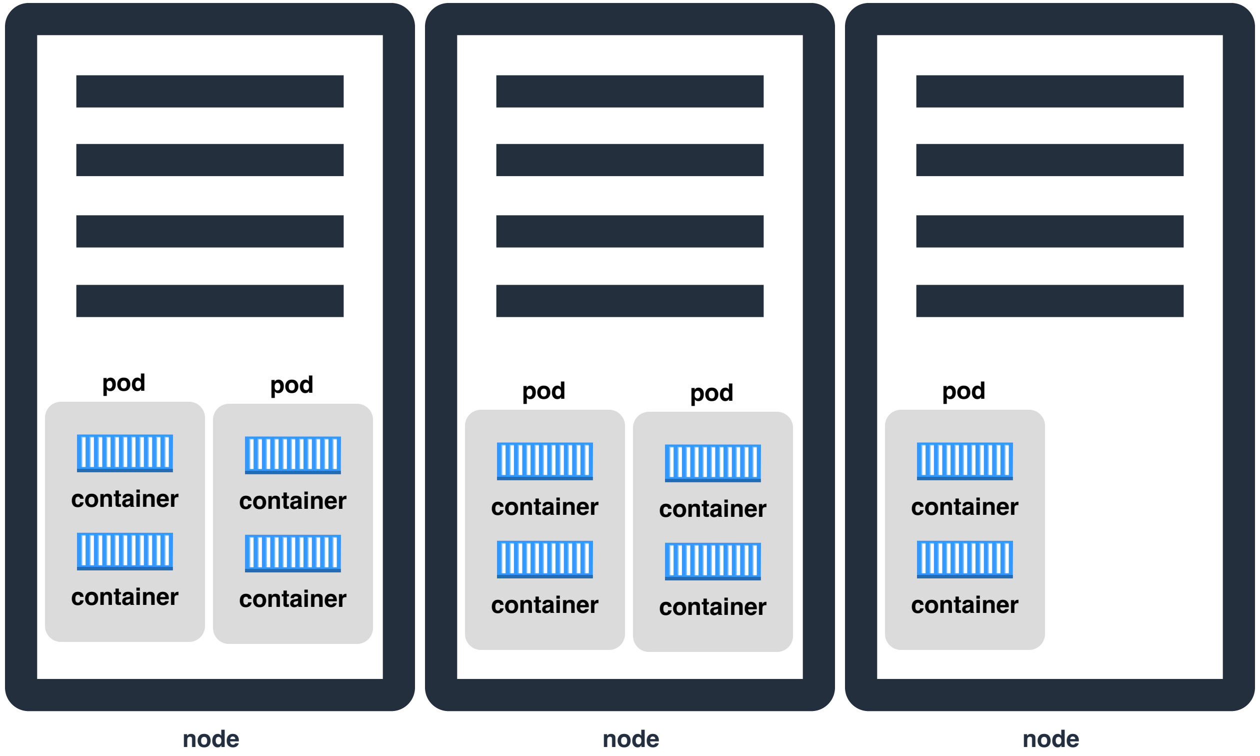
node
node
node

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']
```

Container configuration

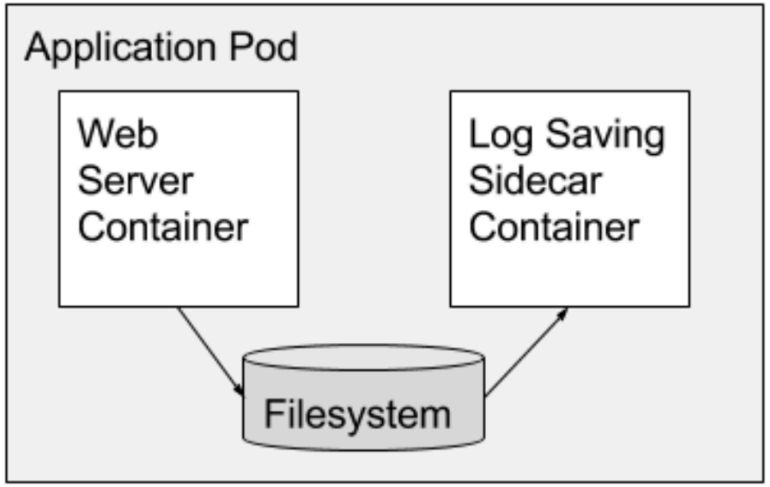
- `command` and `args`
- Ports
- restartPolicy

```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
    image: busybox:stable
    ports:
      - containerPort: 80
```

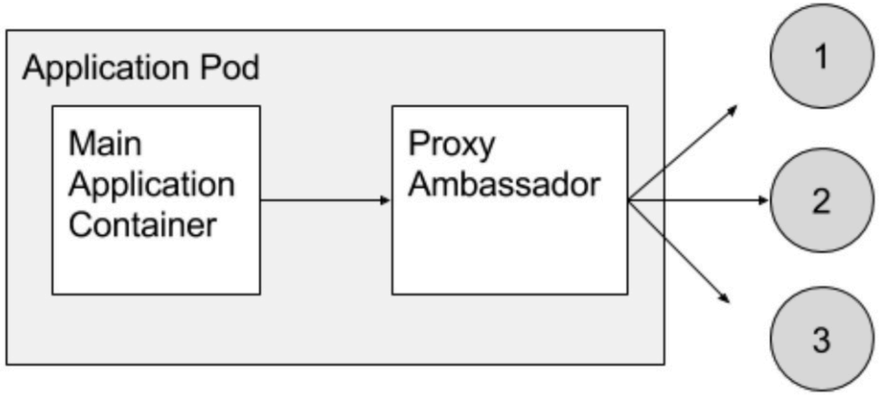
Multi-container Pods - Sidecar

- Additional container for support functionality allowing separation of concerns
- E.g. A log saving container that forwards logs to a central monitoring system



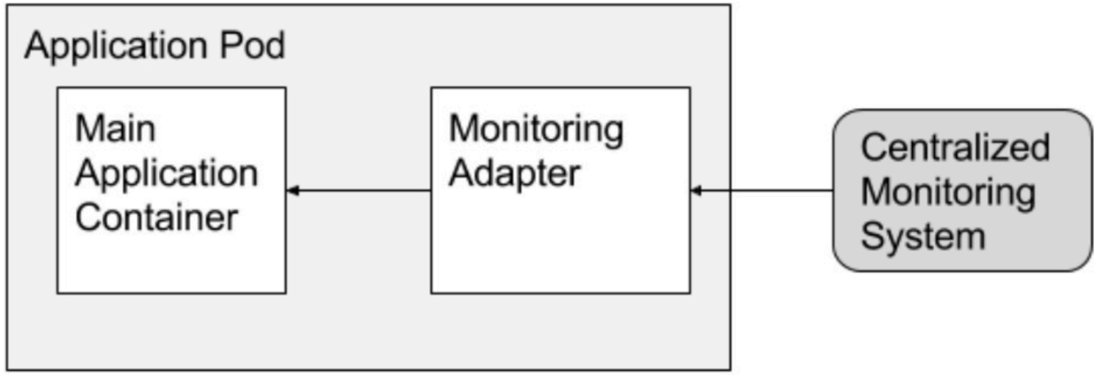
Multi-container Pods - Ambassador

- Proxy a local connection to the world.
- E.g. using a Redis server with one main write and multiple replicas for read. The ambassador proxy handles which redis instance to connect to, developers of main container don't need to add additional logic.



Multi-container Pods - Adapter

- Adapt or modify behaviour of main container
- E.g. Consider the task of monitoring N different applications, each with different monitoring data. Adapter can modify data to standardize for central monitoring system.



More about Pods 🐳

Labels

- 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
```

```bash
# 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
kubectl get pods -l environment!=production
kubectl get pods -l 'environment in (development,production)'
kubectl get pods -l app=my-app,environment=production
```

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
```

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 (already on GKE)

Debugging

- Kubectl commands like `get`, `describe`, `edit` are your friend 🥰

```
# 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
kubectl edit pod nginx -n nginx-ns
```

Cattle not pets 🐄

✨ Pods are intended to be immutable. Some fields can be updated (like `image`) but correct practice is to replace pods.

Tune in for more when we talk about Deployments. 📻

Exercise 1

1. Create a YAML file with a namespace definition for `sunnybikes` and apply it
2. Create a YAML file with a pod definition for the sunnybikes application. For this, use the docker image `pugillum/sunnybikeslite:stable`. Apply this manifest.
3. Apply the configurations and check the logs of the pods with: `kubectl –n sunnybikes logs <podname>`
4. Port-forward using `kubectl port-forward pod/sunnybikes 8000:5000` and access http://localhost:8000 in your browser. Now try http://localhost:8000/docs

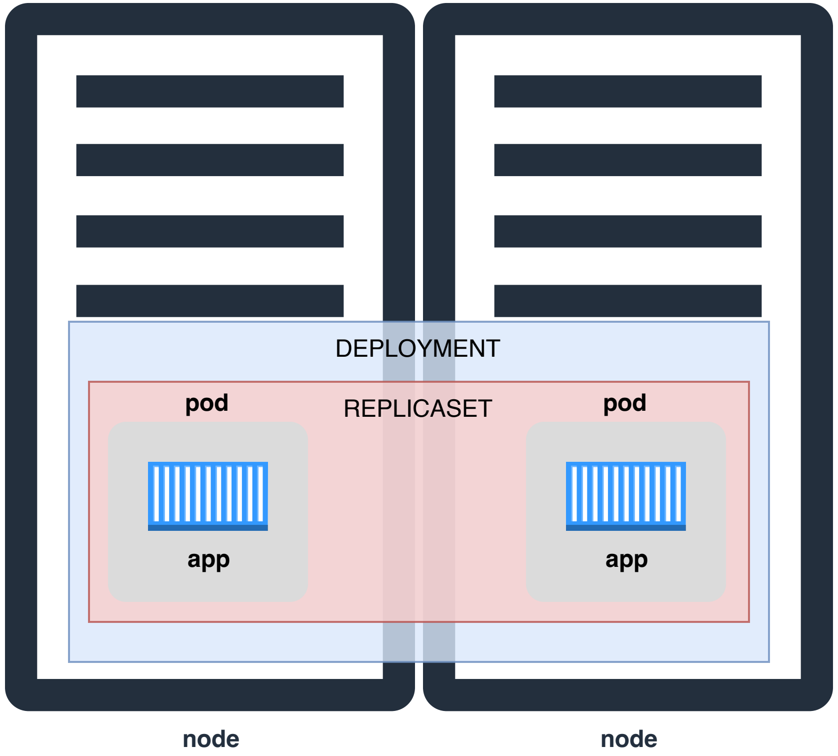
Deployments

Pods

- Cannot self-heal ❤️‍🩹
- Don't scale ⚖️
- Updates and rollbacks -> hard 🎢
- Enter Deployments 🏆

Deployments

- Replicasets manage Pods
  - self-healing
  - scaling
- Deployments manage Replicasets
  - rollouts
  - rollbacks



Deployments - State

- Desired state - declared
- Observed state - monitored by Controllers (ReplicaSet and Deployment)
- Reconciliation - bring in sync

Deployments

```yaml
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
```

Rolling updates

- Update with zero downtime
- Various strategies, default RollingUpdate

```
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
```

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

# View the rollout history of a deployment
kubectl rollout history deployment/rolling-deployment

# View the details of revision 2
kubectl rollout history deployment/rolling-deployment --revision=2
```

Deployment strategy - standard rolling

If `strategy` left out, the default looks like this:

```
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 25% # No more than 25% above desired state
      maxUnavailable: 25% # No more than 25% below desired state
```

Deployment strategy - recreate

- terminate all running instances then recreate with newer version.
- this means downtime

```
spec:
  replicas: 3
  strategy:
    type: Recreate
```

Deployment strategy - ramped

- slow rollout
- for stateful apps that need to rebalance data

```yaml
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 2 # No more than 2 pods above desired state
      maxUnavailable: 0 # No pods below desired state
```

Deployment strategy - Blue/Green

- Have separate deployments for old and new (blue/green)
- Use a service to shift traffic from blue to green
- Can quickly roll back to blue if green has issues

Deployment strategy - Canary

- Similar to Blue/Green but only portion of users directed to new "Canary" deployment
- Rest of traffic shifted once real users have verified the new one

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 2

1. Migrate the pod definitions for `sunnybikes` to a deployment (called `sunnybikes`)
2. The `sunnybikes` pods must be replicated at least 3 times
3. Make sure that when a new update of `sunnybikes` is deployed at least 1 pod is always available during upgrading and the number of pods can go up to a maximum of 6.

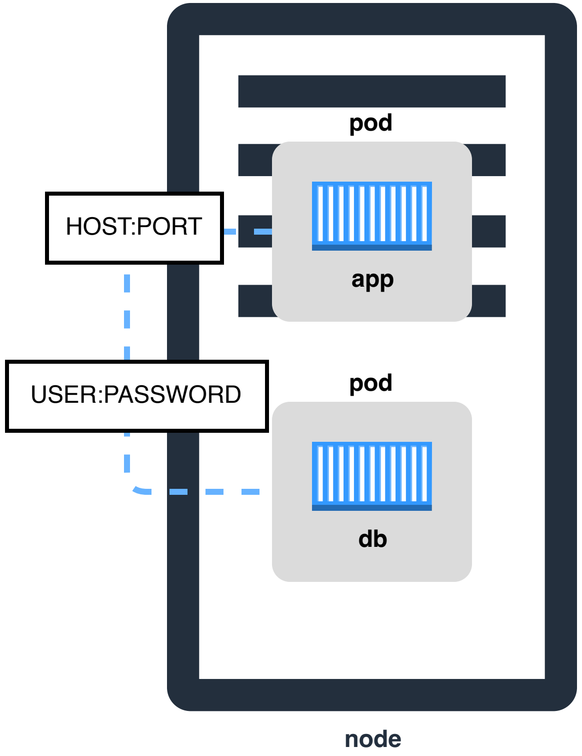
Configuration

Environment variables

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod-env
spec:
  containers:
    - name: busybox
      image: busybox:stable
      command: ["sh", "-c", "echo $MESSAGE;"]
      env:
        - name: MESSAGE
          value: "Hello!"
```

Config Maps & Secrets

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



ConfigMaps

- Sharing configuration between pods / containers
- Mountable as environment variables

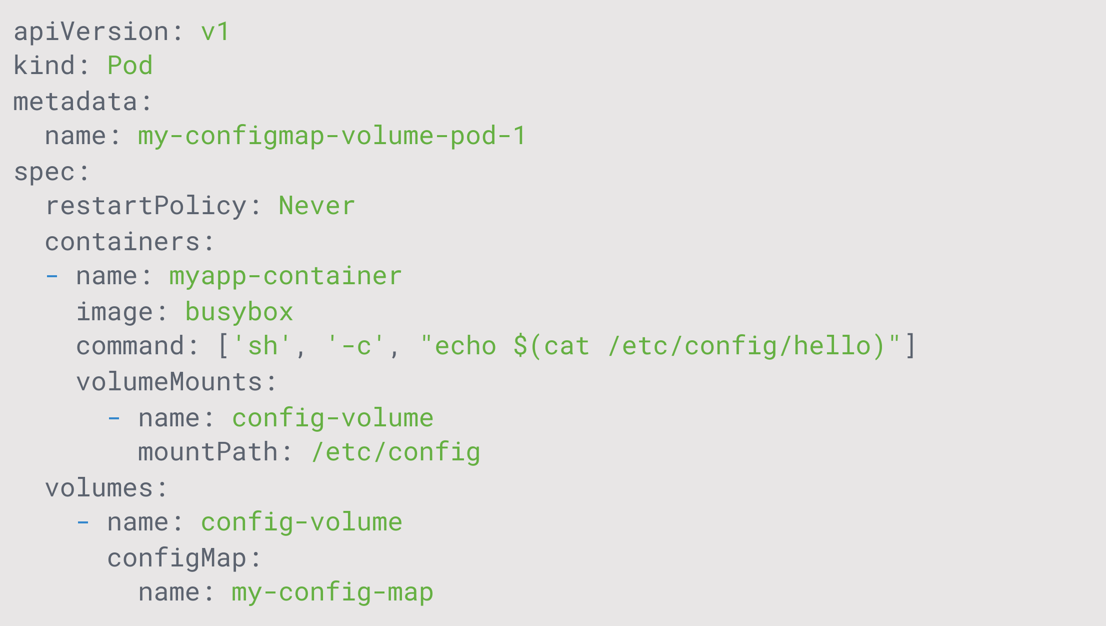
```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-configmap
data:
  # key value style
  message: hello
  name: John

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

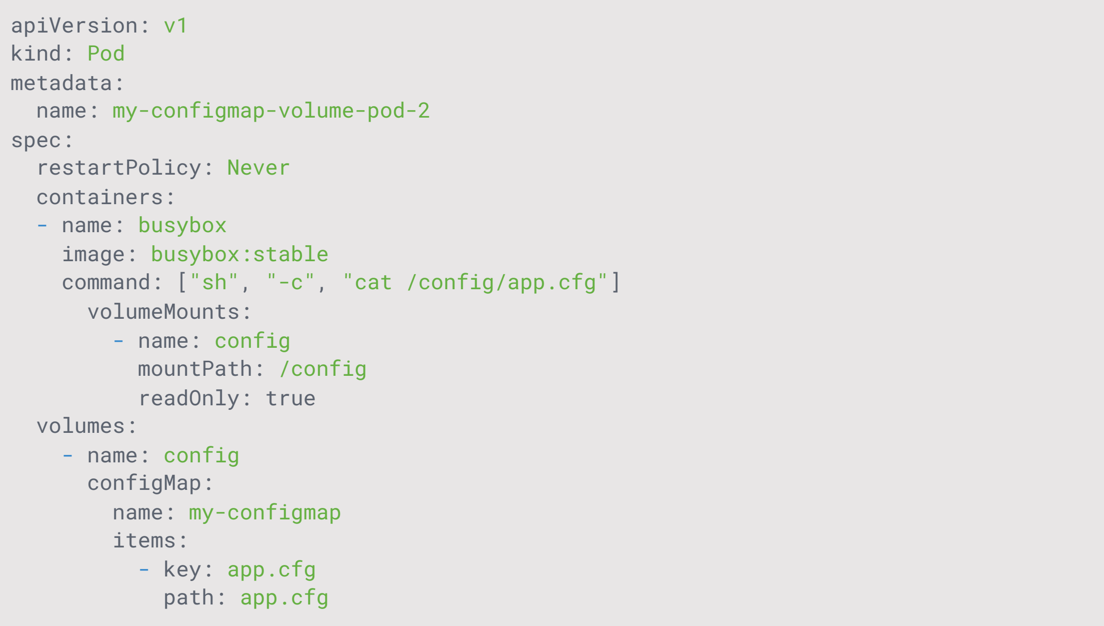
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod-cf
spec:
  restartPolicy: Never
  containers:
    - name: busybox
      image: busybox:stable
      command: ["sh", "-c", "echo $MESSAGE $NAME;"]
      env:
        - name: MESSAGE
          valueFrom:
            configMapKeyRef:
           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

ConfigMaps

- Mountable as volume



```
apiVersion: v1
kind: Pod
metadata:
  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
          mountPath: /config
          readOnly: true
  volumes:
    - name: config
      configMap:
        name: my-configmap
        items:
          - key: app.cfg
            path: app.cfg
```

Secrets

- Manage sensitive data like passwords, tokens etc.
- ⚠️ 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
          valueFrom:
            secretKeyRef:
              name: my-secret
              key: myKey
```

Exercise 3 - Part 1

1. Add an additional deployment `postgres` based on the image `postgres:11-alpine`, only one pod should run
2. Define an environment variable for the `sunnybikes` pods `PG_PORT` with value `"5432"`
3. Create a yaml file with a secret definition for the postgres password called `secret.yaml` and apply
4. 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

Exercise 3 - Part 2

5. Create a yaml file with a config map (`configmap.yaml`) for the postgres init schema. Mount the init script in the postgres container in the folder `docker-entrypoint-initdb.d` as file `init-schema.sql`.

   The schema is as follows:

```sql
CREATE TABLE public.bike_rides (
    uuid UUID PRIMARY KEY,
    name VARCHAR(80) NOT NULL,
    location VARCHAR(80) NOT NULL,
    created TIMESTAMPTZ NOT NULL
)
```

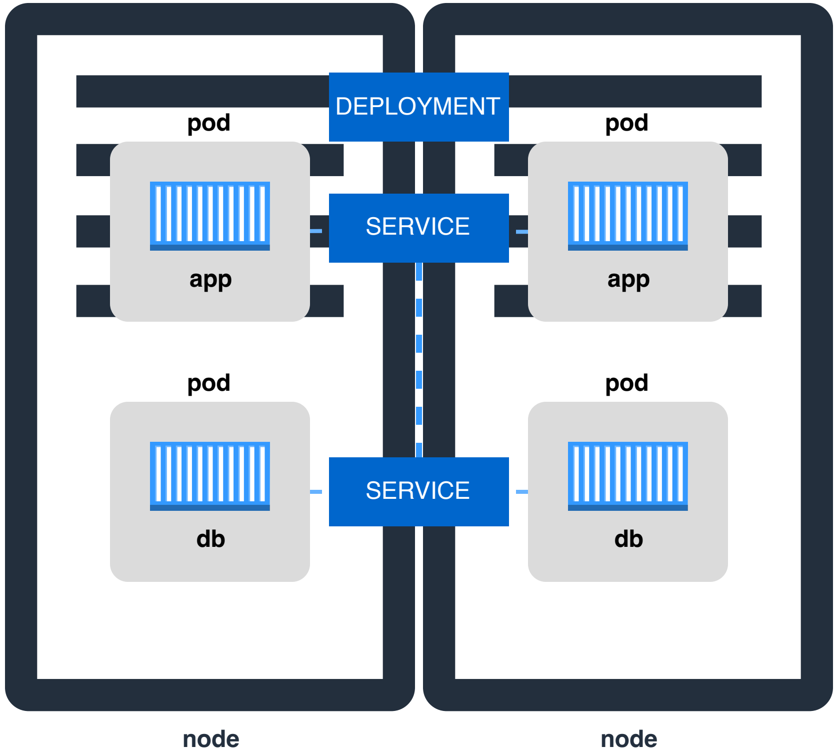
Services

Services

- New pods == new IP addresses
- IP churn!
- Services provide stable network endpoints

Service

- Stable IP address
- Stable DNS
- Seamless connection across nodes
- Lifecyle detached from pods
- Internal and external
- Load balancing across pods
- Network policies



Services - implementing

```
apiVersion: v1
kind: Service
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
```

```
apiVersion: v1
kind: Service
metadata:
  name: nodeport-service
spec:
  type: NodePort # a different type of service
  selector:
    app: nginx # determines which pods receive traffic
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 80
      nodePort: 30080 # the port to map to on the node
```

Pods linked to these services

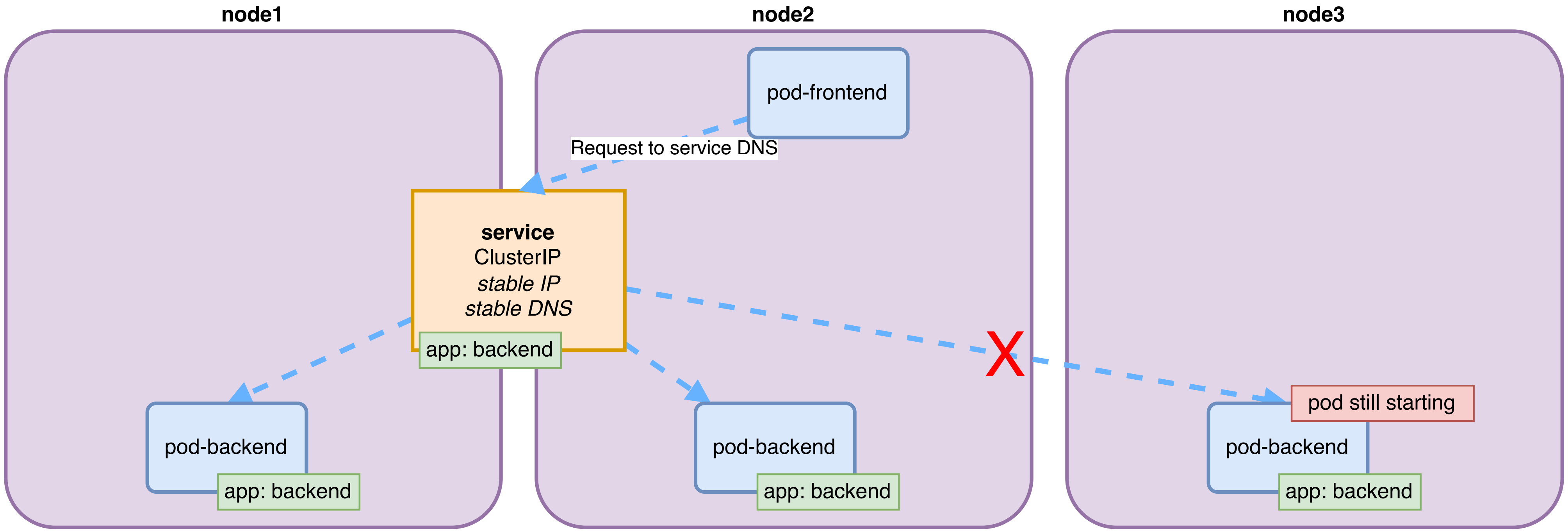
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx # note this label compared to service selector!
    spec:
      containers:
        - name: nginx
          image: nginx:1.20.1
          ports:
            - containerPort: 80
```

```
kubectl get svc
kubectl get endpoints my-service
```

Services - ClusterIP

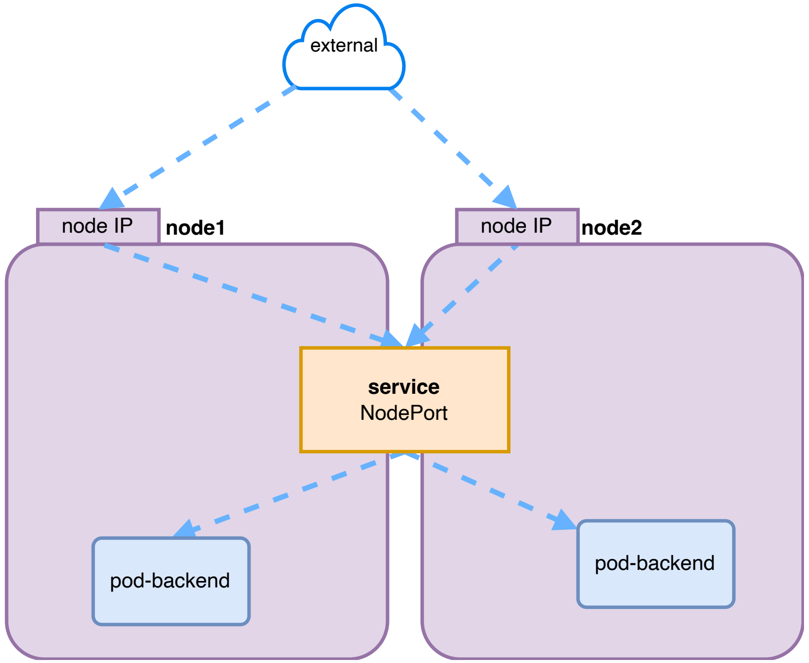
- The default type
- Exposes the Service on a cluster-internal IP
- Only reachable from within the cluster
- Has a DNS entry `<service-name>.<namespace>.svc.cluster.local`

Services - ClusterIP



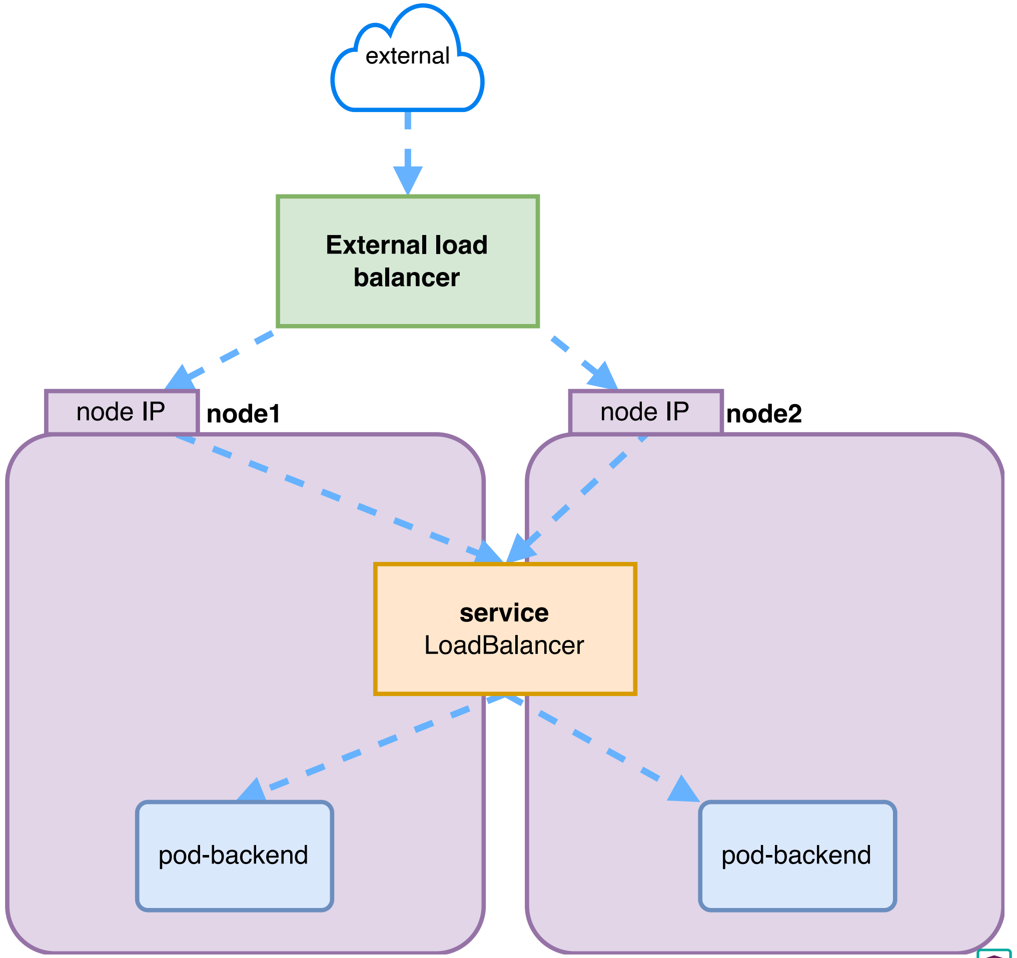
Services - NodePort

- 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

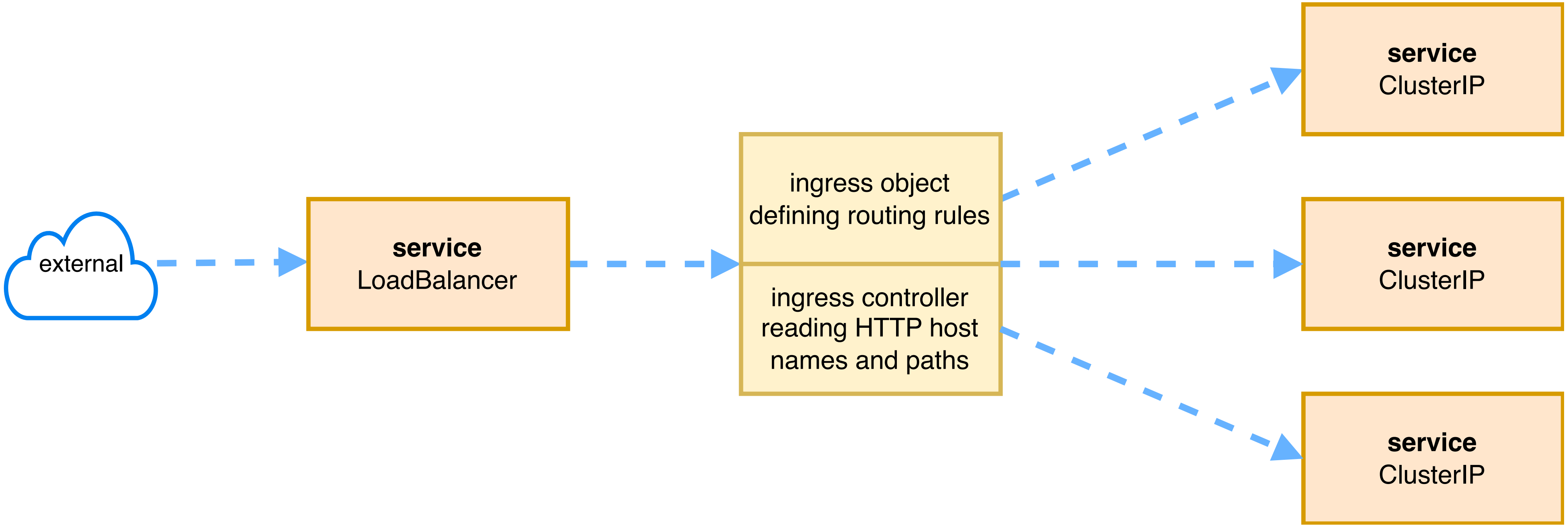
- Uses external load balancer (often from cloud provider)
- NodePort and ClusterIP Services, to which the external load balancer routes, are automatically created



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

Ingress



external

service
LoadBalancer

ingress object
defining routing rules

ingress controller
reading HTTP host
names and paths

service
ClusterIP

service
ClusterIP

service
ClusterIP

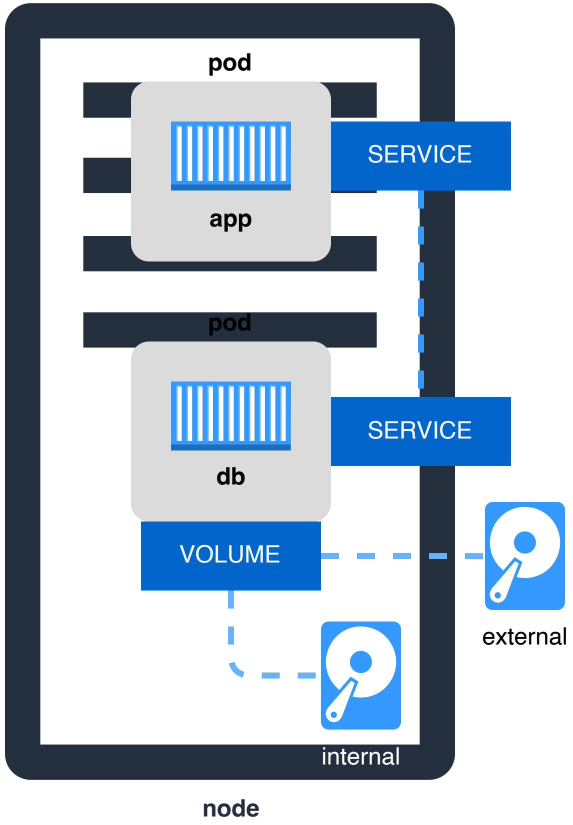
Exercise 4

1. Make Postgres available to the `sunnybikes` pods from only within the cluster
2. `sunnybikes` should be adjusted to use a new image `sunnybikes:stable`
3. `sunnybikes` takes 2 environment variables to configure the postgres host and port, `PG_HOST` - `<service name>.<namespace>.svc.cluster.local` and `PG_PORT` - `"5432"` respectively.
4. Make `sunnybikes` available on port 80 from the outside world

Storage

Volumes

- Pod disk storage is ephemeral 😕
- Volume - a directory that is accessible to all containers in a pod
- Ephemeral volumes - lifetime of the pod
- Persistent volumes - beyond pod lifetime
- Abstract a variety of different storage types - portability FTW!



Volumes in Pod definition

- 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
```

Kubernetes Storage Objects

PersistentVolume (PV)
Storage resources setup in the cluster

PersistentVolumeClaim (PVC)
Connect pods to storage via request

StorageClass (SC)
Dynamically provision PVs with specific characteristics

Container Storage Interface

- A standardized interface for storage integration
- Allows separate development of storage plugins

Persistent Volume

- Represent storage resources in the cluster
- Static PVs are provisioned by cluster admin
- Dynamic PVs are provisioned via StorageClass
- Independent of Pod lifecycle
- Local or Remote
- There are many types - each have a specific configuration
- Not linked to Namespace

Persistent Volume - types

- `local` – Data is stored on devices mounted locally to your cluster's Nodes.
- `hostPath` – Stores data within a named directory on a Node (designed for testing purposes)
- `nfs` – Used to access Network File System (NFS) mounts.
- `csi` – Allows integration with storage providers that support the Container Storage Interface (CSI) specification, such as the block storage services provided by cloud platforms.
- more...

Storage Class

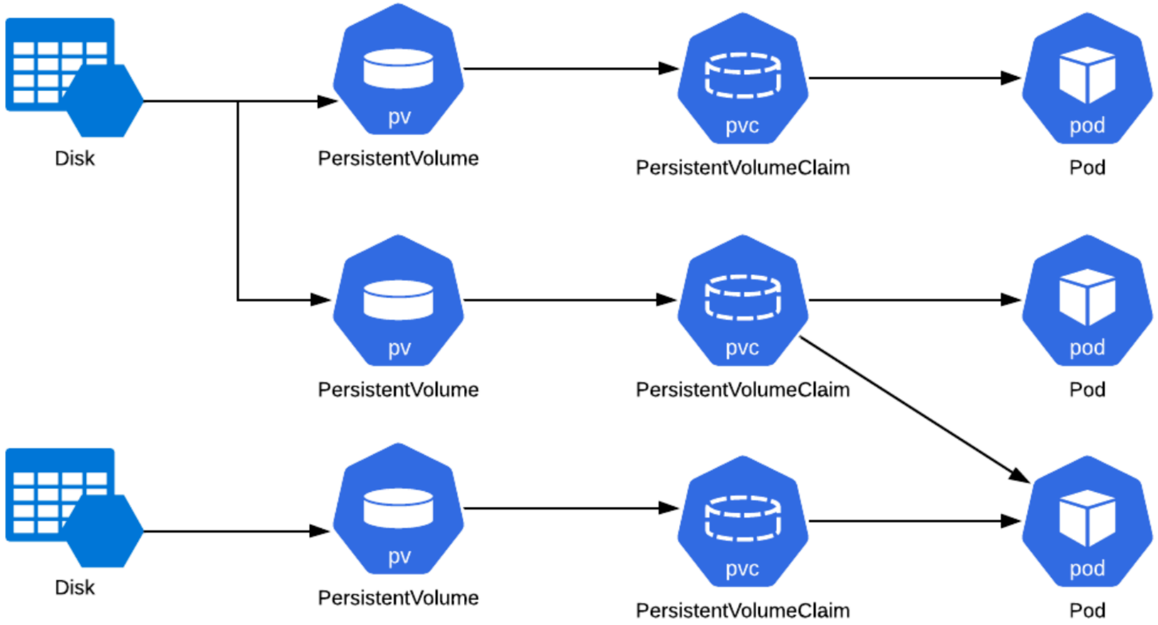
- Handle storage provisioning operations
- Managed by cluster admins
- There are built-in storage classes for each of the supported PV types
- Additional storage classes can be added by installing plugins

Persistent Volume Claim

- Defines a request for storage
- Includes details on type of storage needed (capacity, access mode and storage class)
- Automatically binds to available PersistentVolume that meets requirements
- Mounted in a pod like a volume (available to all containers)

Persistent storage

- A pod may have many PersistentVolumeClaims (PVC)
- A PVC may be be used by many pods
- A PersistentVolume may only be claimed by one PVC
- A disk may have many PersistentVolumes requesting parts of storage



Persistent storage

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  storageClassName: manual # used for binding PVC
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce # volume can be mounted as read-write by a single Node
  hostPath:
    path: "/mnt/data" # maps to this path on node
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  storageClassName: manual # matches PV storage class
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 512Mi
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pvc-pod
spec:
  containers:
  - name: nginx
    image: nginx:stable
    ports:
      - containerPort: 80
    volumeMounts:
    - mountPath: "/usr/share/nginx/html"
      name: my-storage
  volumes:
  - name: my-storage
    persistentVolumeClaim:
      claimName: my-pvc
```

Exercise 5

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 of the API
Hint: Postgres stores its data in `/var/lib/postgresql/data`

Verify the volume is working.

Pods - Advanced

imagePullPolicy

- `IfNotPresent` - reduce network bandwidth
- `Always` - to ensure the latest version always - the default
- `Never` - image must already be on node, for security

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mycontainer
    image: myimage:latest
    imagePullPolicy: IfNotPresent
```

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"
```

Job / CronJob

- Controller for short-lived pods (can be multiple)
- Job - one time or scheduled tasks
- CronJob - periodic tasks
- Tracks progress and ensures completion

Job / CronJob YAML

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: my-job
spec:
  template:
    spec:
      containers:
      - name: print
        image: busybox:stable
        command: ["echo","This is a test!"]
      restartPolicy: Never
  backoffLimit: 4 # Optional: number of retries
  activeDeadlineSeconds: 10 # Optional: max time until job stopped
```

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: my-cronjob
spec:
  schedule: "*/1 * * * *" #
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox:stable
            command: ["echo", "This is a test!"]
          restartPolicy: OnFailure
```

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 via execution of command or HTTPGet

```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 via exec command or HTTPGet

```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
```

Exercise 6

1. Add a liveness probe to check the `healthz` endpoint of the sunny bikes API.
2. Add a readiness probe to check that Postgres is ready by calling the command `pg_isready -U postgres` with a delay of 10 seconds, repeating at 10 second intervals.

Extra topics

Roles Based Access Control (RBAC)

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

```yaml
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
```

```yaml
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
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

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

- `USER` configures user for subsequent `RUN`, `CMD` and `ENTRYPOINT`

```
FROM python:3.12-slim

COPY requirements.txt requirements.txt
RUN pip install --user -r requirements.txt

RUN groupadd -r workers && useradd -r -g workers worker
USER worker
WORKDIR /home/worker

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

COPY --chown=worker:workers . .

ENTRYPOINT ["python"]
```

Pod Security context

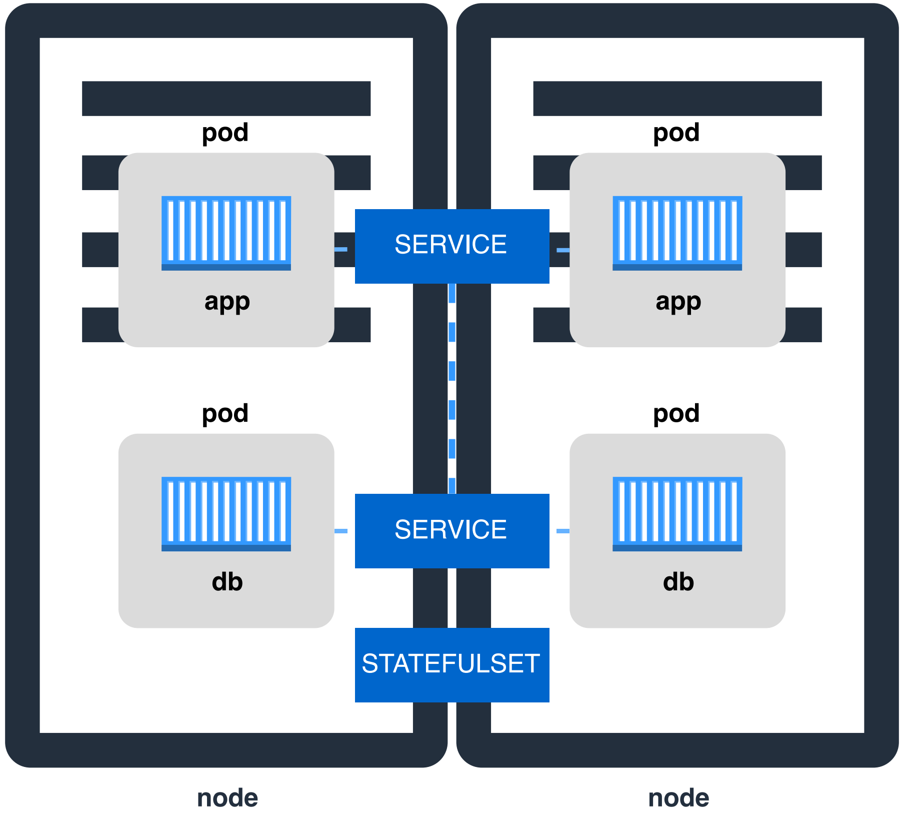
`runAsNonRoot` - simple and safe

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: myapp-container
    image: busybox:stable
    securityContext:
        runAsNonRoot: True
```

```yaml
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
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: sec-ctx-vol
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```

StatefulSet

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



Databricks Containers

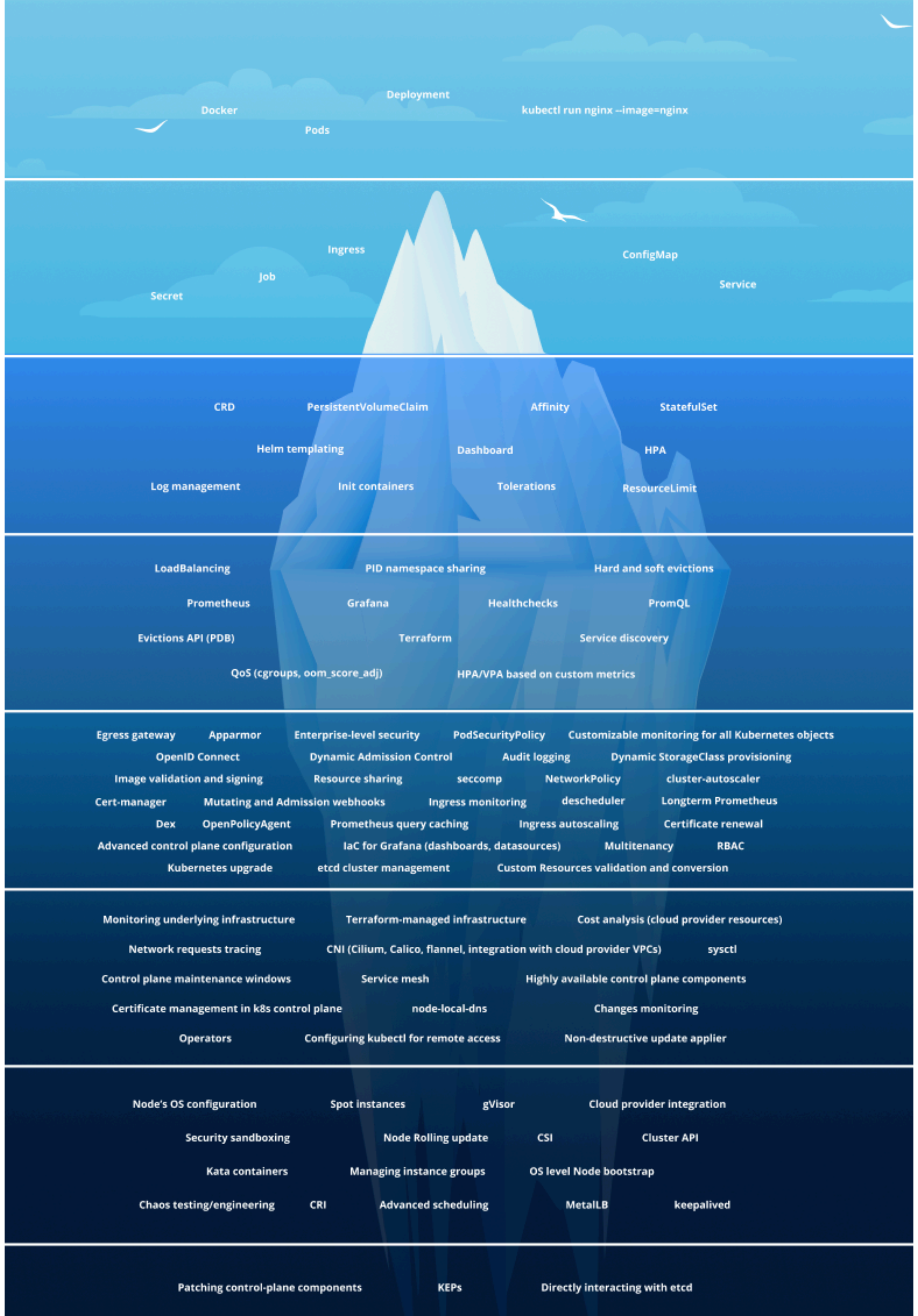
- Databricks Container Service
- Can use Databricks base images or from scratch
- Only LTS images patched regularly
- Use with caution ☢

Helm

- A package manager for Kubernetes
- Simplifies deployment and management of applications on a cluster
- Combine multiple manifests into a single chart
- Templating
- Version control
- Rollout/rollback easily
- Community maintained charts

Kustomize

- Customize YAML files without the need for templates
- Define base templates in standard manifest YAML
- Use patching to adjust settings per environment
- Manage objects with a `kustomization` file
- Installed with `kubectl`



Still more to learn 🤯

Thanks for listening! 🙂