Kubernetes Technical Primer

Ben Coleman

@BenCodeGeek

Jan 2022, v1.6

Content Notes

This guide is not intended to be delivered as a complete presentation!

It can used as a learning guide or given as a reference

Cherry pick sections or slides and use/present as needed

Target Audience

Technical architects, developers and platform engineers

Those wanting to the learn the core fundamentals of Kubernetes

Not for those trying to deploy Kubernetes from scratch or get into internals

TABLE OF CONTENTS

Introduction To Kubernetes

Core Components

Using Kubernetes

Configuring Basic Workloads

Beyond The Basics

Additional Network Services

Debugging and Troubleshooting Workloads

Advanced Pod Configuration

Scaling

Extending Kubernetes

DevOps

Azure Kubernetes Service

Additional Resources

MAIN SECTIONS

Introduction To Kubernetes

Core Components

Using Kubernetes

Beyond The Basics

DevOps

Azure Kubernetes Service

Introduction To Kubernetes

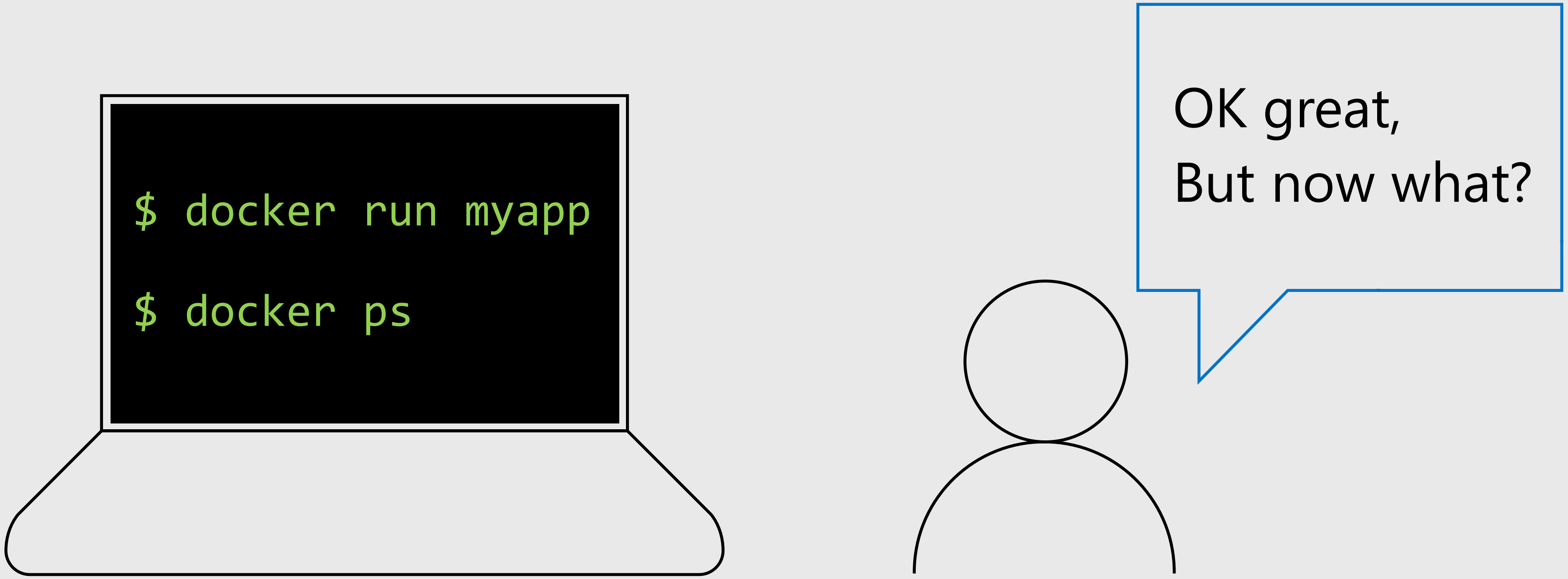
What is Cloud Native?

“Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.”

CLOUD NATIVE
COMPUTING FOUNDATION

The need for orchestration



```
$ docker run myapp

$ docker ps
```

OK great,
But now what?

The need for orchestration

NAME RESOLUTION

MANAGING PLACEMENT

RESOURCE MANAGEMENT

SECRETS

NETWORKING & LOAD BALANCING

SCHEDULING

SECURITY

SCALING UP/DOWN

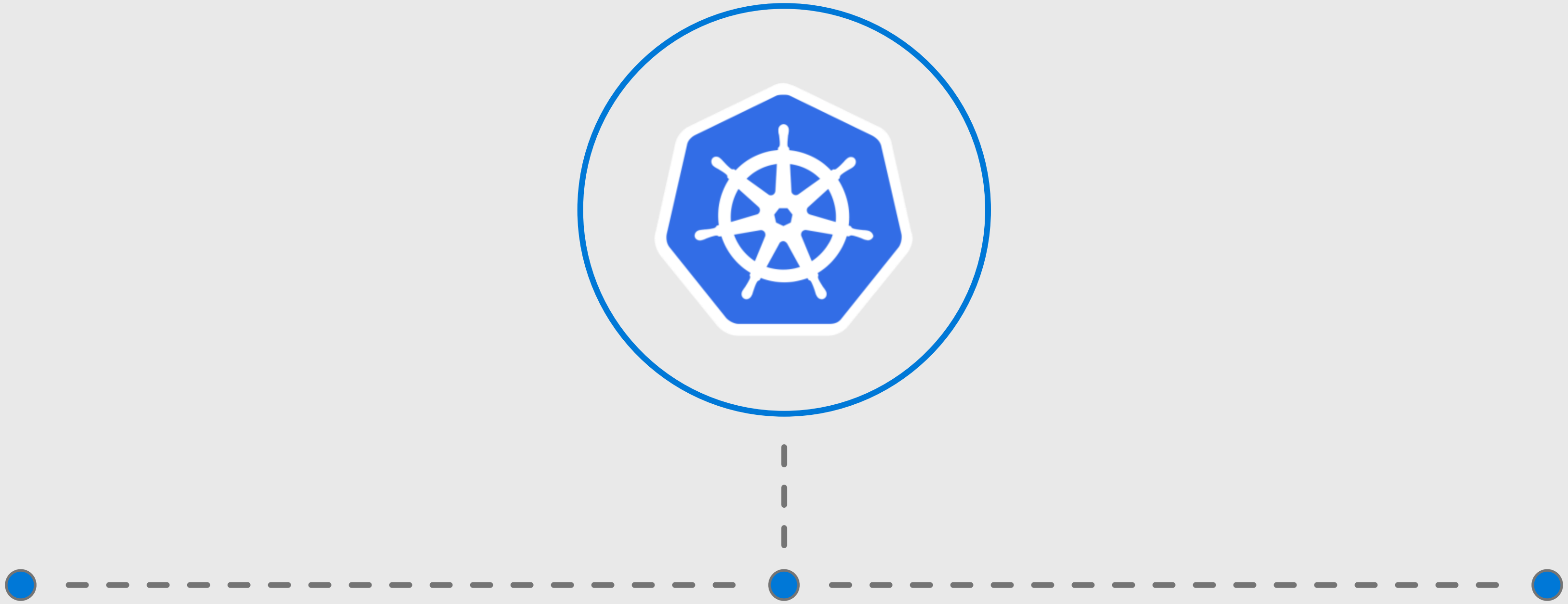
HEALTH CHECKS

FAILOVER

SERVICE DISCOVERY

Production And At Scale

Kubernetes - production grade orchestration



Portable

Public, private, hybrid,
multi-cloud

Extensible

Modular, pluggable,
hookable, composable

Self-healing

Auto-placement, auto-restart,
auto-replication, auto-scaling

The elements of orchestration

- Scheduling
- Affinity/anti-affinity
- Health monitoring
- Failover
- Scaling
- Networking
- Service discovery
- Coordinated app upgrades

Why Choose Kubernetes?

- Cornerstone of cloud native approach
- Industry adoption
- Avoids lock-in
- Large & growing ecosystem
- Run anywhere
- Open Source with high degree of support
- Skills availability
- Rise of microservices & containers

Kubernetes – A Modern Orchestrator

Easily load balance stateless services

Seamless integration with cloud providers

Extendable and pluggable

Auto scaling and hyper scale

Run stateful services with persistence

Health monitoring

Declaratively define applications

Rolling updates

Service failover & workload optimization

Secure with policies

Role based access

Multi-tenancy

Core Concepts and Terms

Node
A worker machine (VM) normally clustered, each capable of running pods

Deployment
A logical object for managing a replicated application (i.e. set of pods)

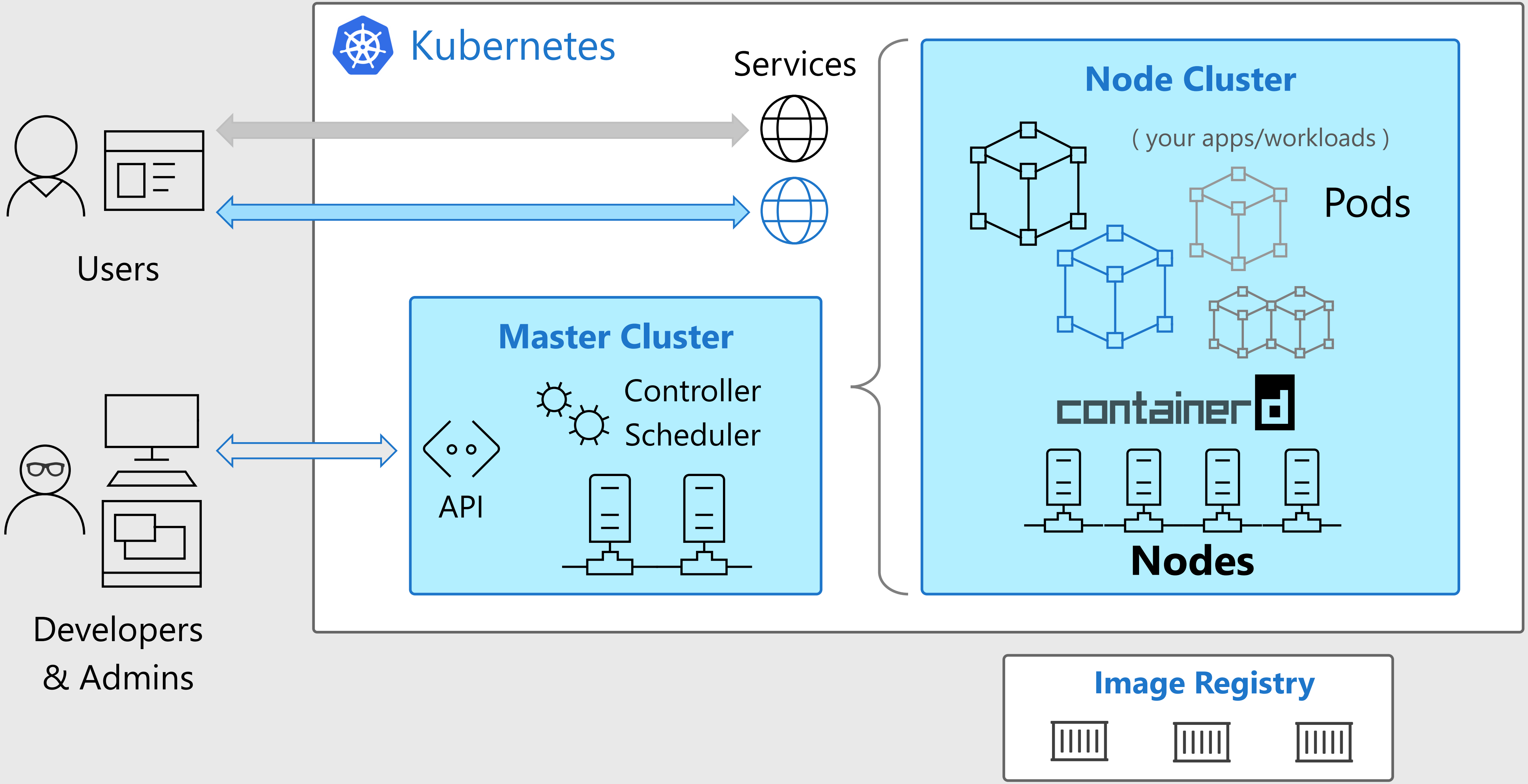
Label
Metadata attached to any object for configuration and selection

Pod
A group of one or more containers that is lifecycle managed

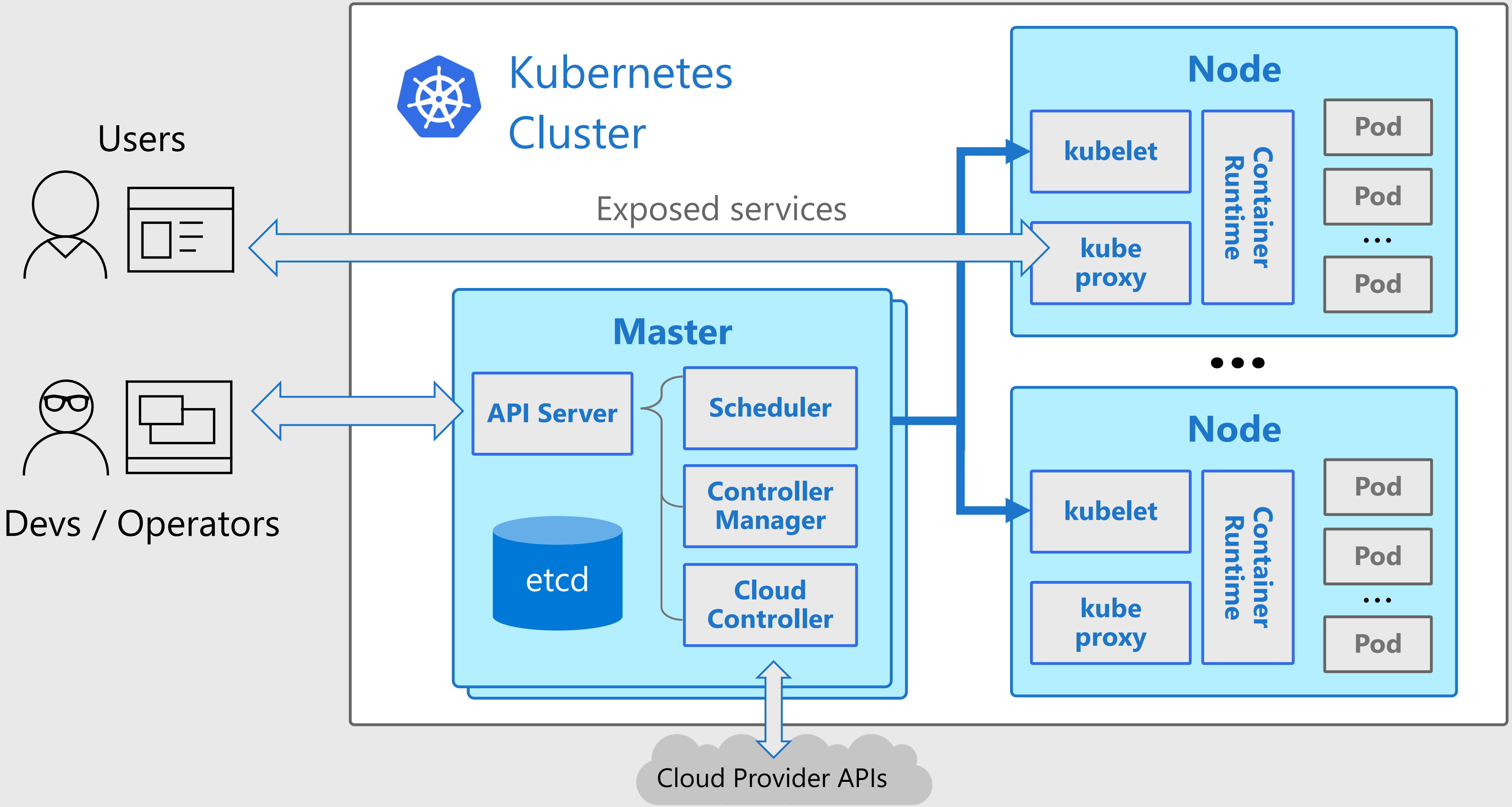
Service
Network access to a resource, e.g. pod or port. Typically load balanced

Replica Set
A set of one or more pods that is distributed and replicated across nodes

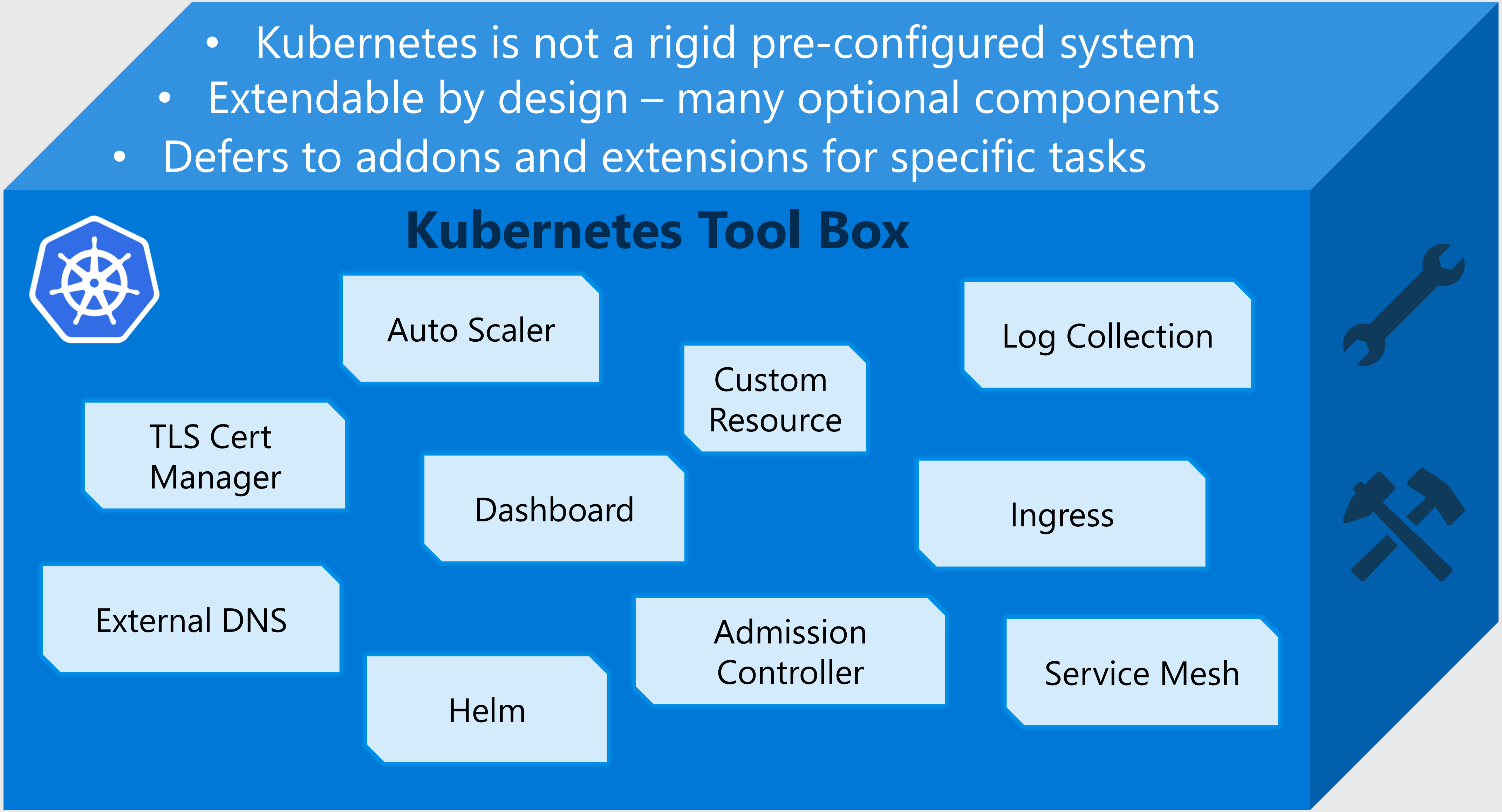
Highly Simplified Architecture



Internal Architecture



Highly Extensible / Unopinionated



- Kubernetes is not a rigid pre-configured system
- Extendable by design – many optional components
- Defers to addons and extensions for specific tasks

Kubernetes Tool Box

Auto Scaler

Custom Resource

Log Collection

TLS Cert Manager

Dashboard

Ingress

External DNS

Admission Controller

Service Mesh

Helm

Core Components

Pods

Fundamental building block of Kubernetes

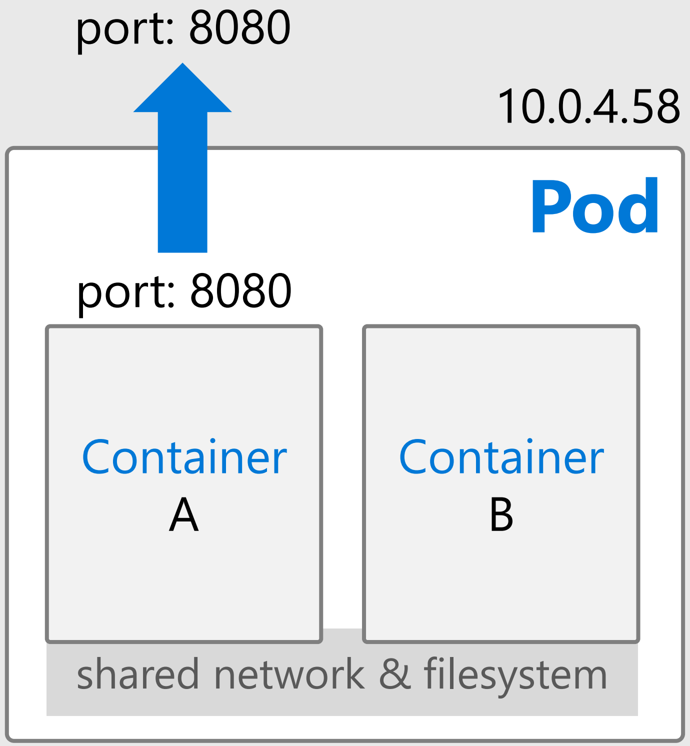
Pods run **one or more containers**

Containers in a pod **share network/storage**

Pods each have their own IP address

Pods expose **one or more ports**

Pods are scheduled and run on a *Node*



Example Pod

Pods are the primary way of running your workloads in Kubernetes

Deployments & Replica Sets

Scale and run pods across multiple nodes

Deployments describe a **replicated** set of *Pods*

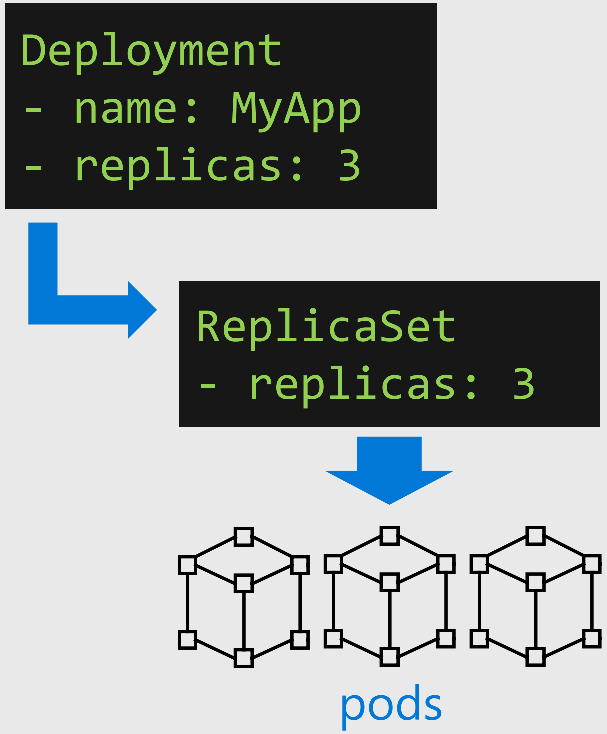
A *Deployment* represents **desired state**

- Rolling updates used to safely roll out changes

You **scale** *Deployments* up & down

Deployments typically run **stateless** workloads

Deployments use *ReplicaSets*



```
Deployment
- name: MyApp
- replicas: 3
```

```
ReplicaSet
- replicas: 3
```

Deployments let you run & scale stateless workloads in Kubernetes

Services

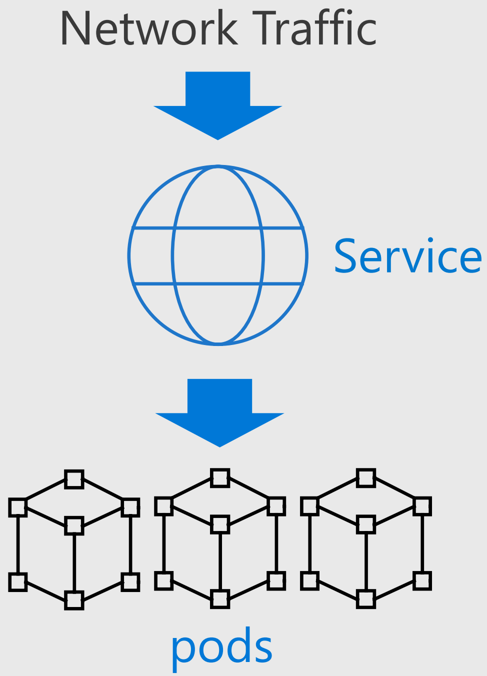
Network access to Pods

Pods are ephemeral - they can move/die/change without notice. **In general, users do not directly access *Pods***

Services are an abstraction which defines a **logical set** of *Pods* and a policy by which to access them

Services use labels and selectors to map to *Pods*

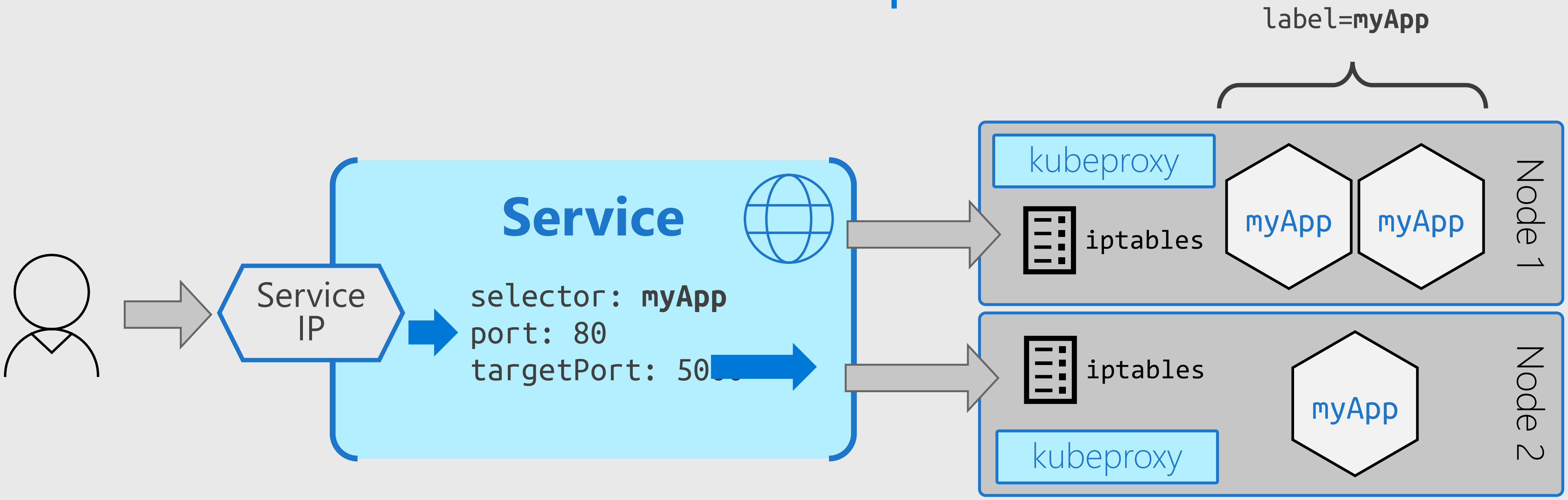
Services are assigned IP addresses and DNS names



Services are how you connect to Pods over the network

Services – Simplified Illustration

Allow virtual network access to one or more pods



label=**myApp**

Service IP

Service

```
selector: myApp
port: 80
targetPort: 50
```

kubeproxy

iptables

myApp

myApp

Node 1

iptables

kubeproxy

myApp

Node 2

EXTERNAL

LoadBalancer

Uses cloud provider to present an *external* load-balanced IP

INTERNAL

ClusterIP

Internal virtual IP, only accessible by other pods/services

DNS and Service Discovery

Naming for Pods and Services

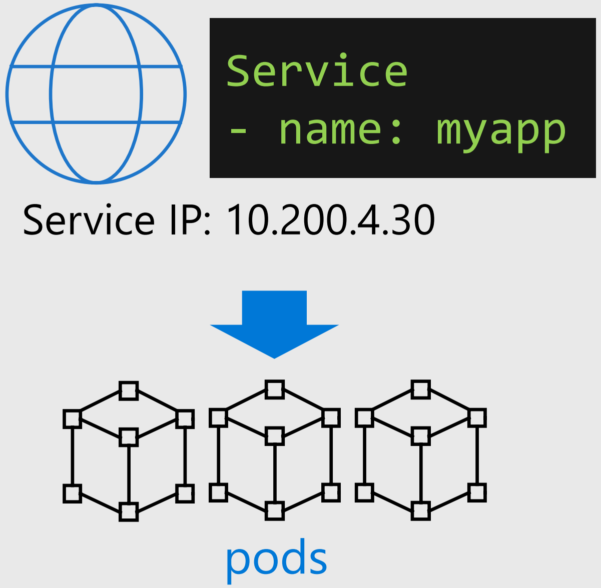
DNS in cluster is handled automatically by Kubernetes

All **ClusterIP** (internal) *Services* get assigned a **DNS record based on the service's name**

Pods also get DNS names, but this is less useful

External public DNS can also be configured – See "Additional Network Services" section later

```
$ nslookup myapp
Address: 10.200.4.30
```



```
Service
- name: myapp
```

Connect to your workloads using DNS and Services

Data Volumes & Mounts

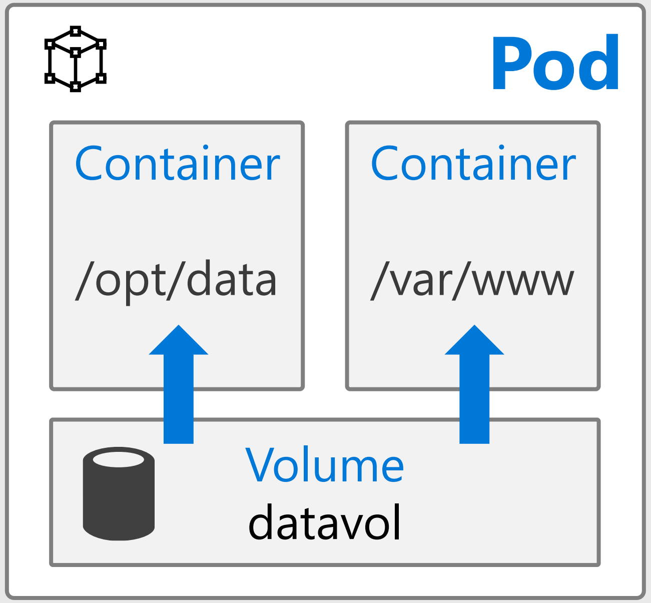
Handling State & Data

Filesystem of running containers is ephemeral. **All data written will be lost on a restart**

Use **Volumes** to hold data or state you want to keep, or to inject data into a *Pod*

Volumes are mounted into a container at a **mountPath**

Many types of storage can be used to back the Volume



Warning! A volume shares lifecycle with the Pod, so are not persistent

Volumes hold data and state for Pods and containers

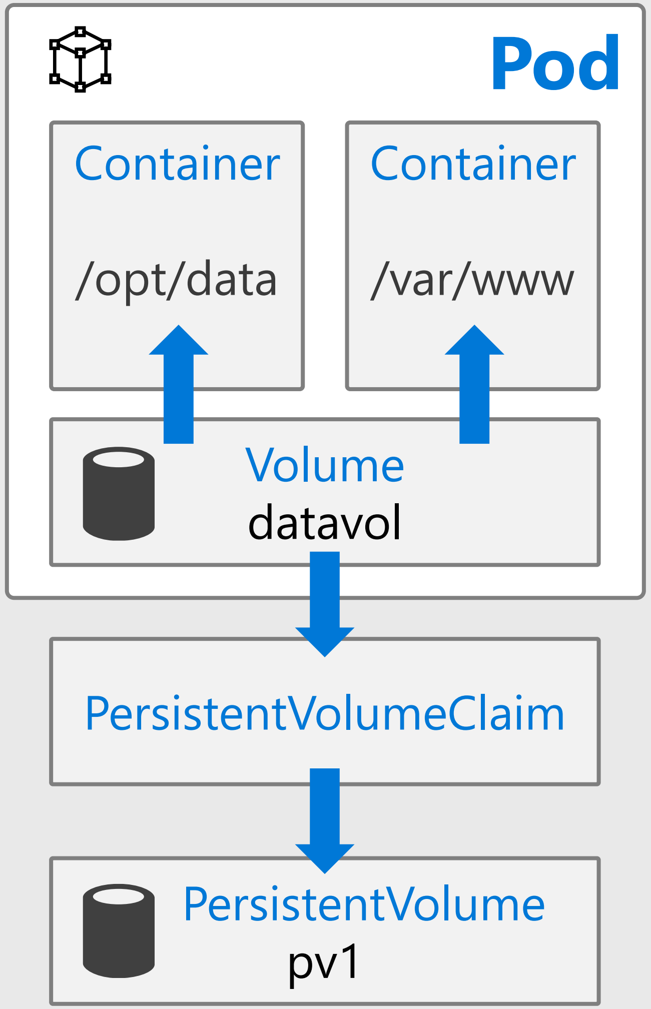
Persistent Volumes

Handling State & Data

A *PersistentVolume* allows you to **hold data independent of Pod lifecycle**

A pod uses a *PersistentVolumeClaim* to bind to a *PersistentVolume*
- **ReadWriteOnce** - mounted on a single Node (e.g. db)
- **ReadWriteMany** - mounted on multiple Nodes

Many **storage plugins** exist: NFS, iSCSI, Azure (Disk & Files), CSI, Ceph, Gluster, AWS



Persistent Volumes retain data long term, outside of Pods

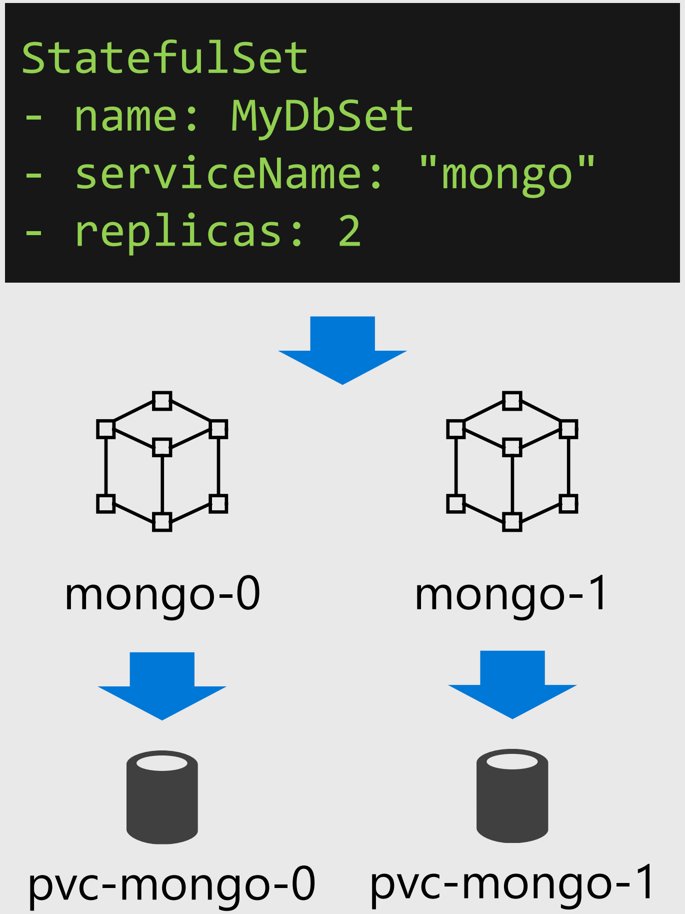
Stateful Sets

Handling Stateful workloads

A *StatefulSet* is like a *Deployment* except *Pods* get well defined names and replicas start in **ordered sequence**

StatefulSets **retain identity** regardless of which *Node* they run on

Each *Pod* in a *StatefulSet* will bind to the same defined *PersistentVolumeClaim*



```
StatefulSet
- name: MyDbSet
- serviceName: "mongo"
- replicas: 2
```

Use a StatefulSet rather than Deployment for stateful workloads

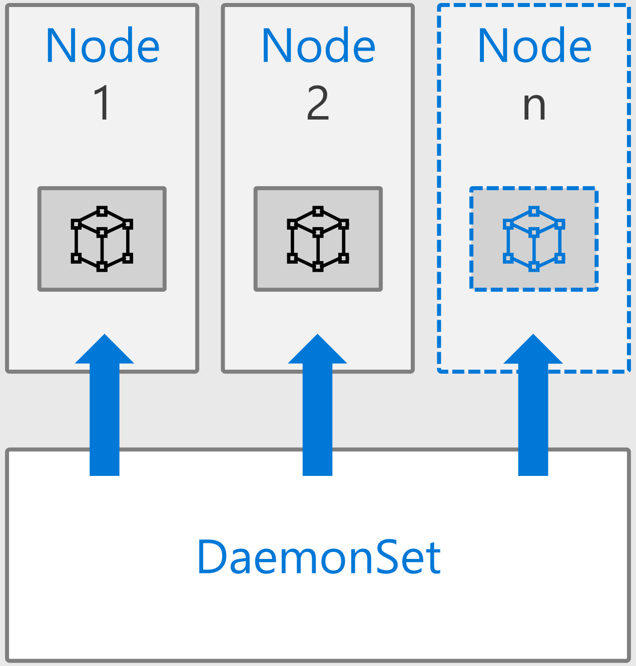
Daemon Sets

Running Pods across all Nodes

A *DaemonSet* ensures that all *Nodes* in the cluster run a given *Pod*. *Pods* will be **created/removed as *Nodes* are added/removed**

Used for special system and cluster daemons, logging, storage, etc.

DaemonSets are **not often used for normal app workloads**



DaemonSets run system Pods for monitoring & network

Jobs & CronJobs

Workloads that run to completion

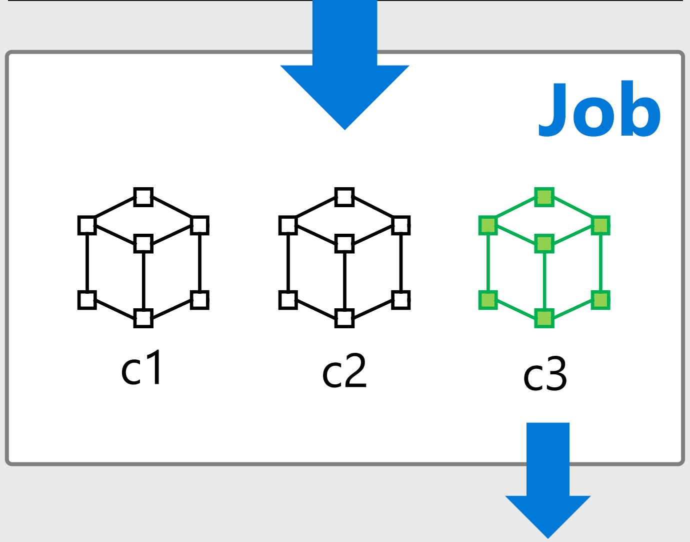
A *Job* creates one or more pods and **ensures that a specified number of them successfully terminate**

Jobs can run in **serial** or **parallel**

Control of number of failures, completions, restart policy and level of parallelism

CronJobs allows you to **schedule** *Jobs* to be run

```
kind: Job
spec:
  completions: 1
  parallelism: 3
```



c3 → Terminated: Complete
Job → Successful

Use Jobs for any workloads that run in batch or perform one off tasks

Namespaces

Logical Separation & Cluster Organisation

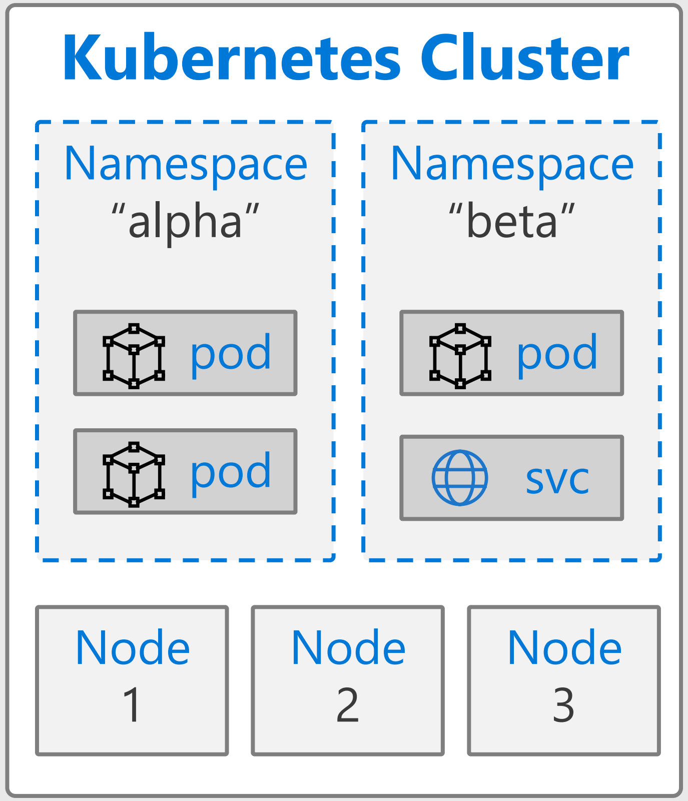
Most Kubernetes objects live inside a *Namespace*

Kubernetes starts with two *Namespaces:* `default` and `kube-system`

You can create *Namespaces* to **logically partition** a cluster, e.g. for dev/test or different customers

Nodes will be shared across *Namespaces*

A namespace does ***not*** provide isolation & multi-tenancy



When first learning Kubernetes use the default Namespace

Role Based Access Control (RBAC)

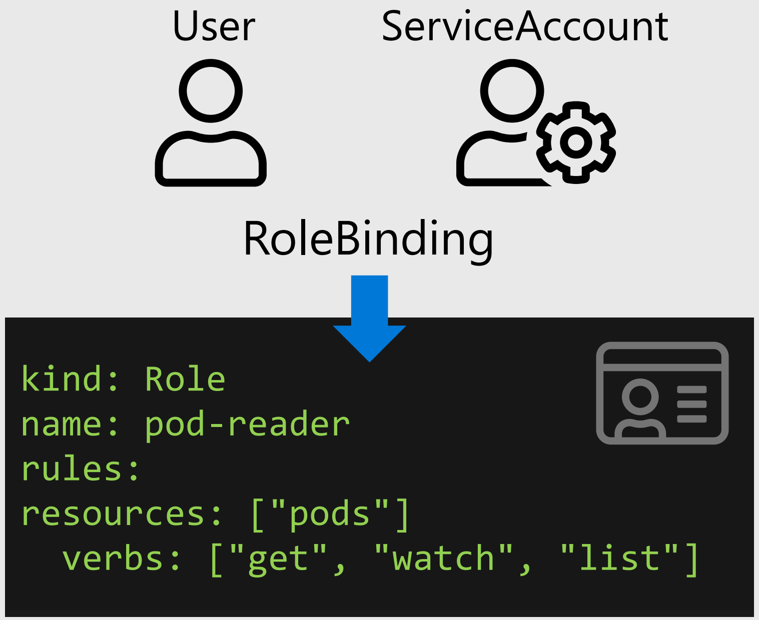
Regulating & governing access

RBAC **controls user and system access** to the API and Kubernetes resources

Roles define privileges as sets of **verbs** and **API resources**

RoleBinding grants *Roles* to *Users* and *ServiceAccounts*

Note. Kubernetes doesn't include a native identity system or a way to manage end user accounts



```
kind: Role
name: pod-reader
rules:
resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

Verb:	**get**, **delete**, **create**, **list**, **update**, **watch**
API:	**pods**, **secrets, services, jobs, nodes, ingresses**

RBAC is optional, but a standard feature for any new cluster

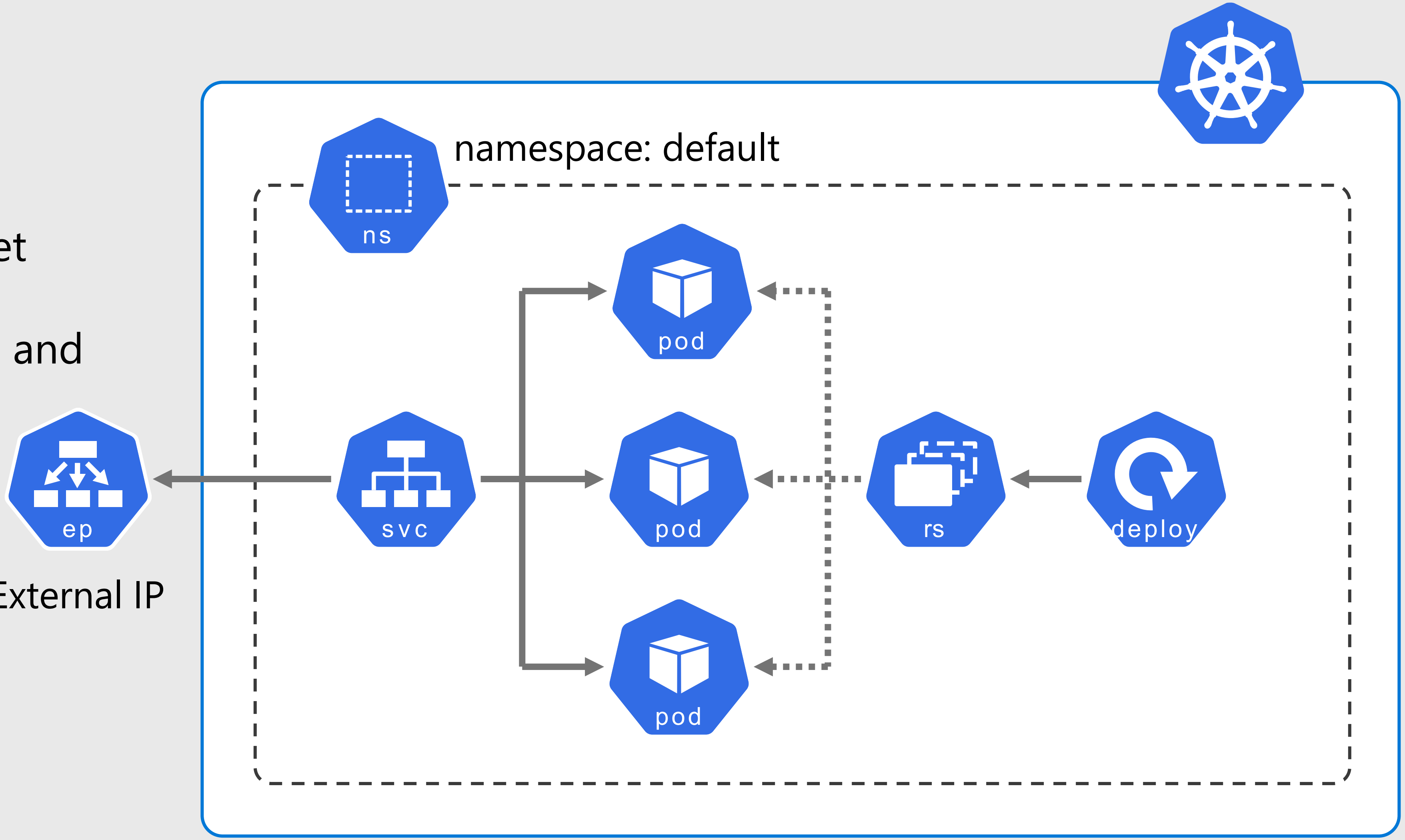
Putting It All Together

Sample Architecture – Simple App

Basic web application

Running in 3 pods via a Deployment and ReplicaSet

With LoadBalancer service and external IP



Using Kubernetes

Command Line - kubectl

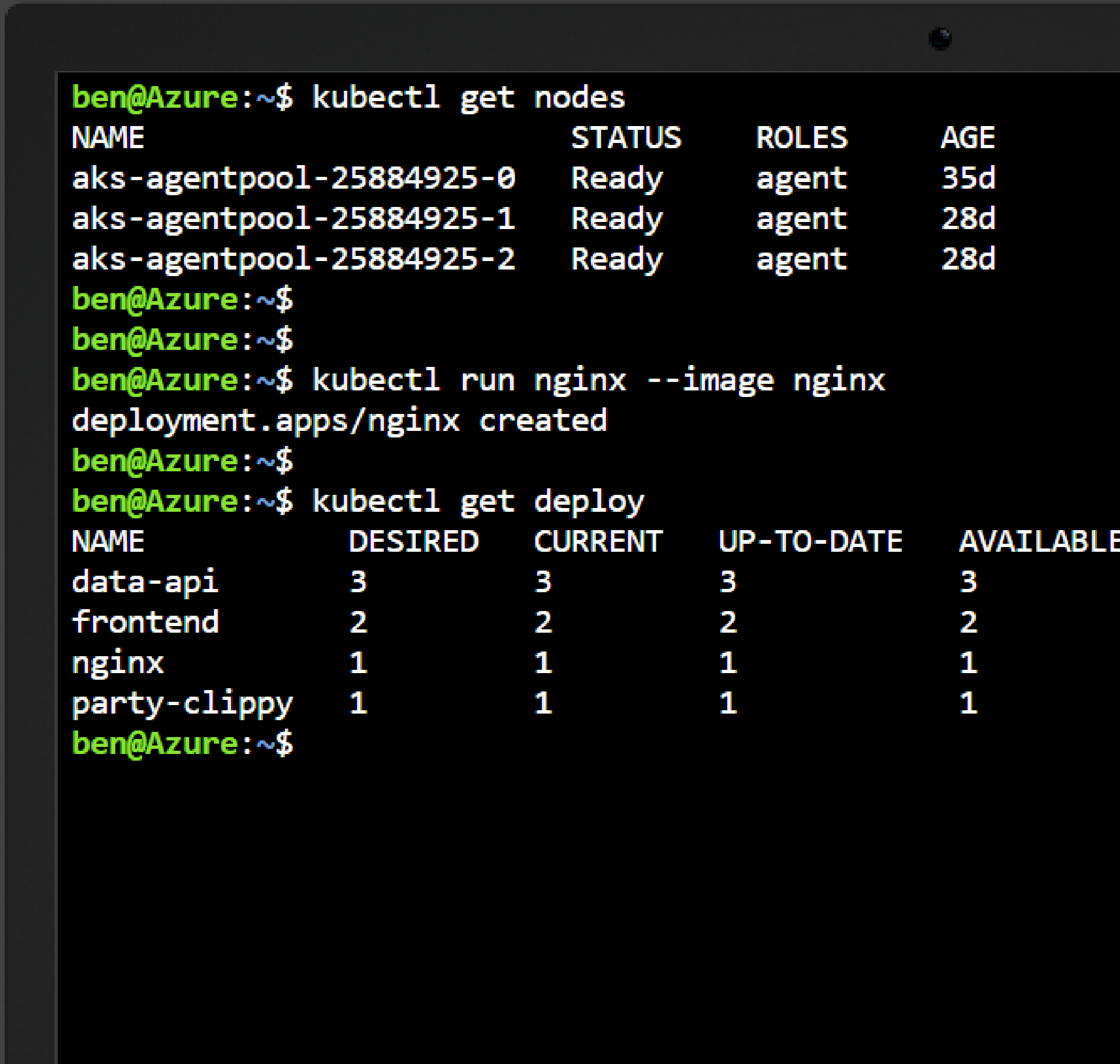
Main management & control interface to Kubernetes

Single executable binary

Can manage multiple clusters

Secure interface to Kubernetes API

Used for cluster ops and application deployment & management



```
ben@Azure:~$ kubectl get nodes
NAME                        STATUS    ROLES     AGE
aks-agentpool-25884925-0    Ready     agent     35d
aks-agentpool-25884925-1    Ready     agent     28d
aks-agentpool-25884925-2    Ready     agent     28d
ben@Azure:~$
ben@Azure:~$
ben@Azure:~$ kubectl run nginx --image nginx
deployment.apps/nginx created
ben@Azure:~$
ben@Azure:~$ kubectl get deploy
NAME            DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
data-api        3         3         3            3
frontend        2         2         2            2
nginx           1         1         1            1
party-clippy    1         1         1            1
ben@Azure:~$
```

Kubectl – Common Commands

get	delete	apply	describe	...others...
Display one or many resources	Delete resources	Create resources from manifests	Get details of any resource	Some other commands

`get nodes`
`get pods`
`get all`
`get pod pod123`
`get nodes -w`
	`delete deploy mydeploy`
	`delete -f myapp.yaml`
	`delete -l app=foo`
		`apply -f myapp.yaml`
		`apply –f ./stuff`
		`apply –f http://example.net/app.yaml`
			`describe pod pod138`
			`describe svc myservice`
			`describe pod -l app=myapp`
				`logs pod138`
				`rollout restart deploy mydeploy`
				`exec pod138 -- ls`

kubernetes.io/docs/reference/kubectl/cheatsheet

kubernetes.io/docs/reference/kubectl/overview

Set up aliases & autocomplete!

Dashboard

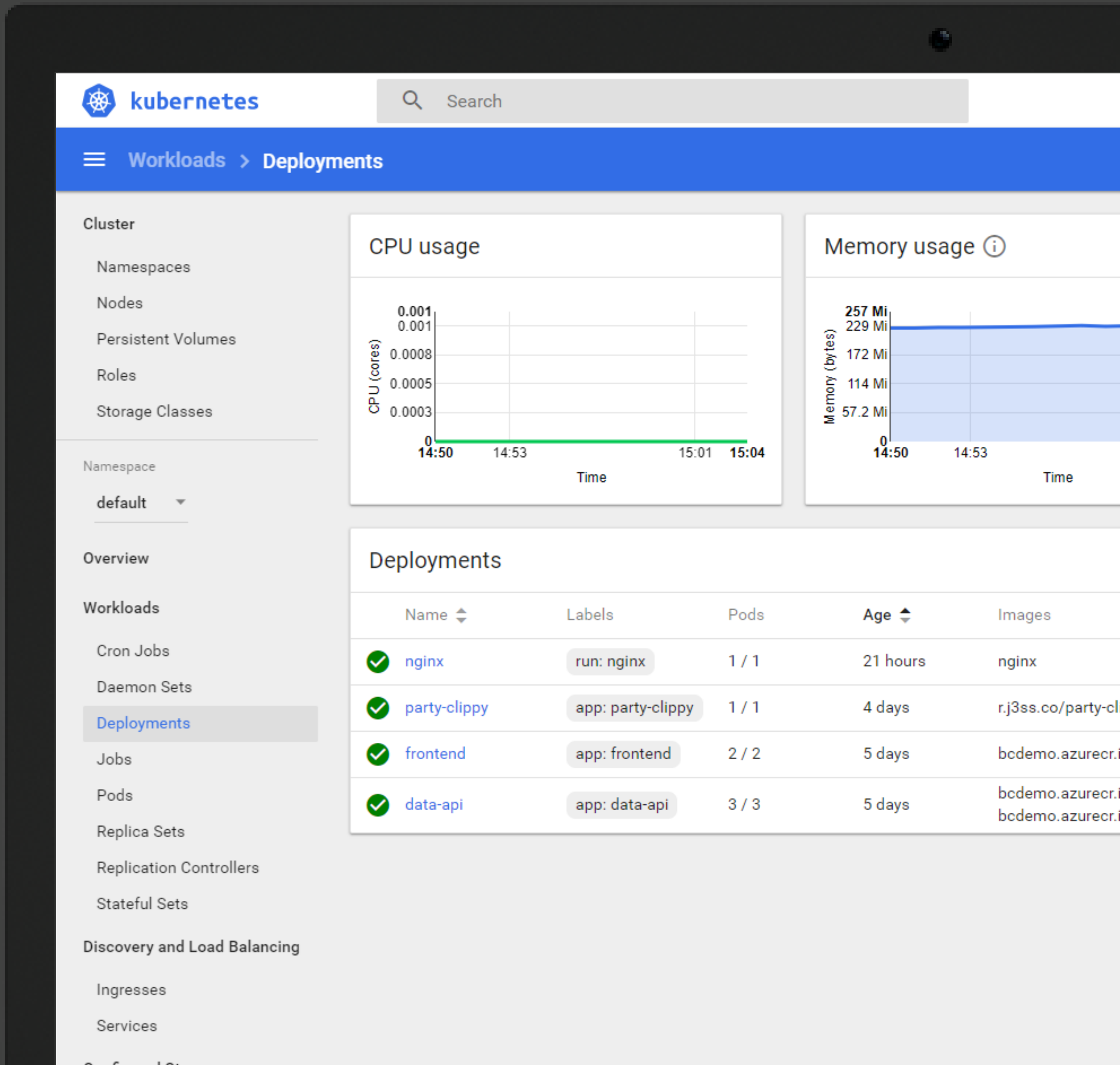
Official management web UI

Useful when learning

Not secure, should never be exposed publicly

Optional, not deployed by default

https://github.com/kubernetes/dashboard



Deployments

	Name	Labels	Pods	Age	Images
✓	nginx	run: nginx	1 / 1	21 hours	nginx
✓	party-clippy	app: party-clippy	1 / 1	4 days	r.j3ss.co/party-cl
✓	frontend	app: frontend	2 / 2	5 days	bcdemo.azurecr.i
✓	data-api	app: data-api	3 / 3	5 days	bcdemo.azurecr.i bcdemo.azurecr.i

Kubernetes Object Management

Three methods of managing Kubernetes

kubernetes.io/docs/concepts/overview/object-management-kubectl

Management Technique	Operates On	Recommended Environment	Learning Curve	Infrastructure As Code
Imperative commands	Live objects	Dev projects	Lowest	No
Imperative object configuration	Individual files	Production use	Moderate	Limited
Declarative object configuration	Individual & multiple files	Production use	Highest	Yes

File based declarative object configuration is the most common approach used

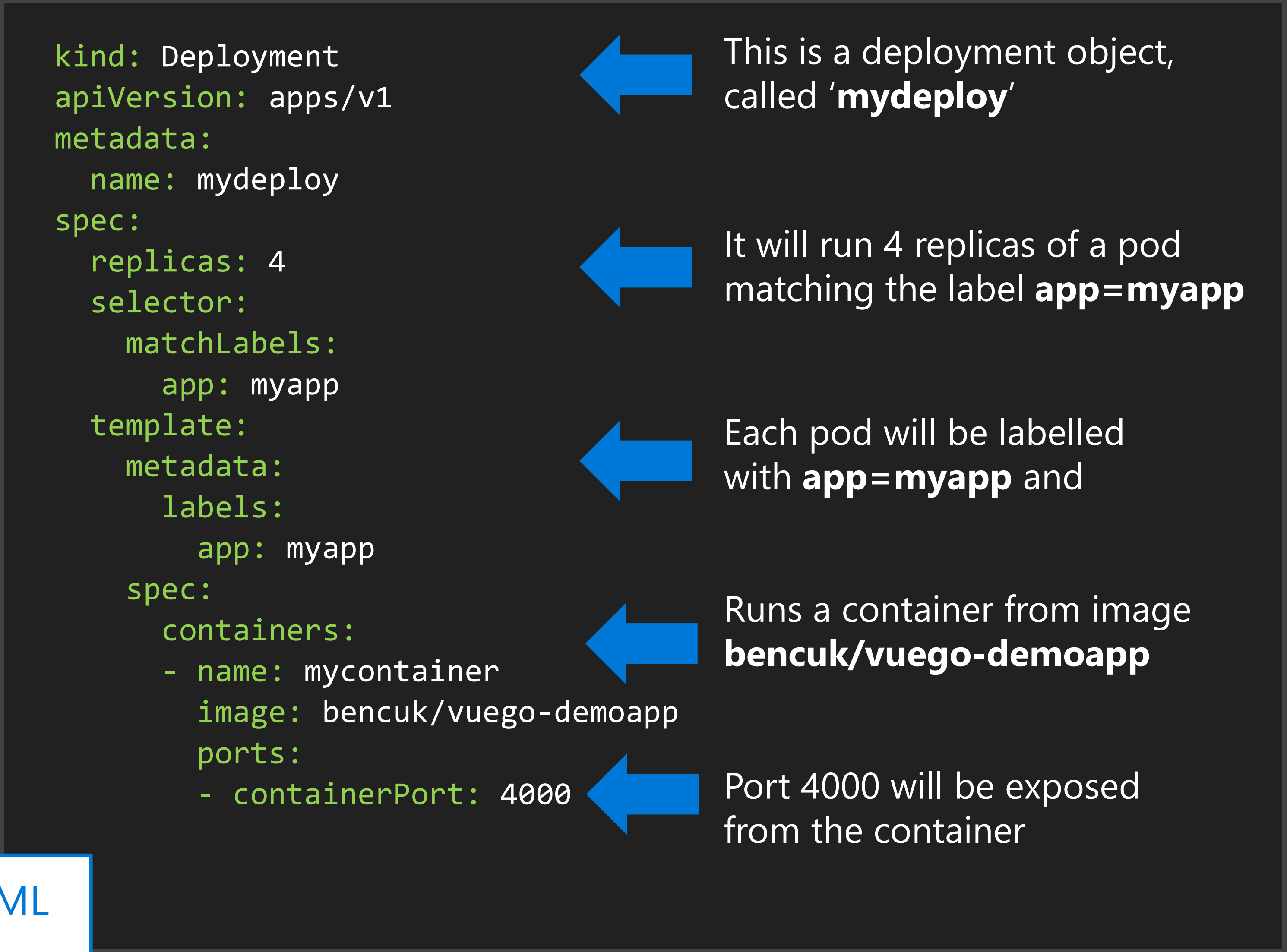
Introduction to the Declarative Model

YAML or JSON documents

Describe any Kubernetes object

Objects & properties map directly to the Kubernetes API

You can combine multiple objects into a single file (separate with ---)



```yaml
kind: Deployment
apiVersion: apps/v1
metadata:
  name: mydeploy
spec:
  replicas: 4
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: mycontainer
        image: bencuk/vuego-demoapp
        ports:
        - containerPort: 4000
```

This is a deployment object, called ‘**mydeploy**’

It will run 4 replicas of a pod matching the label **app=myapp**

Each pod will be labelled with **app=myapp** and

Runs a container from image **bencuk/vuego-demoapp**

Port 4000 will be exposed from the container

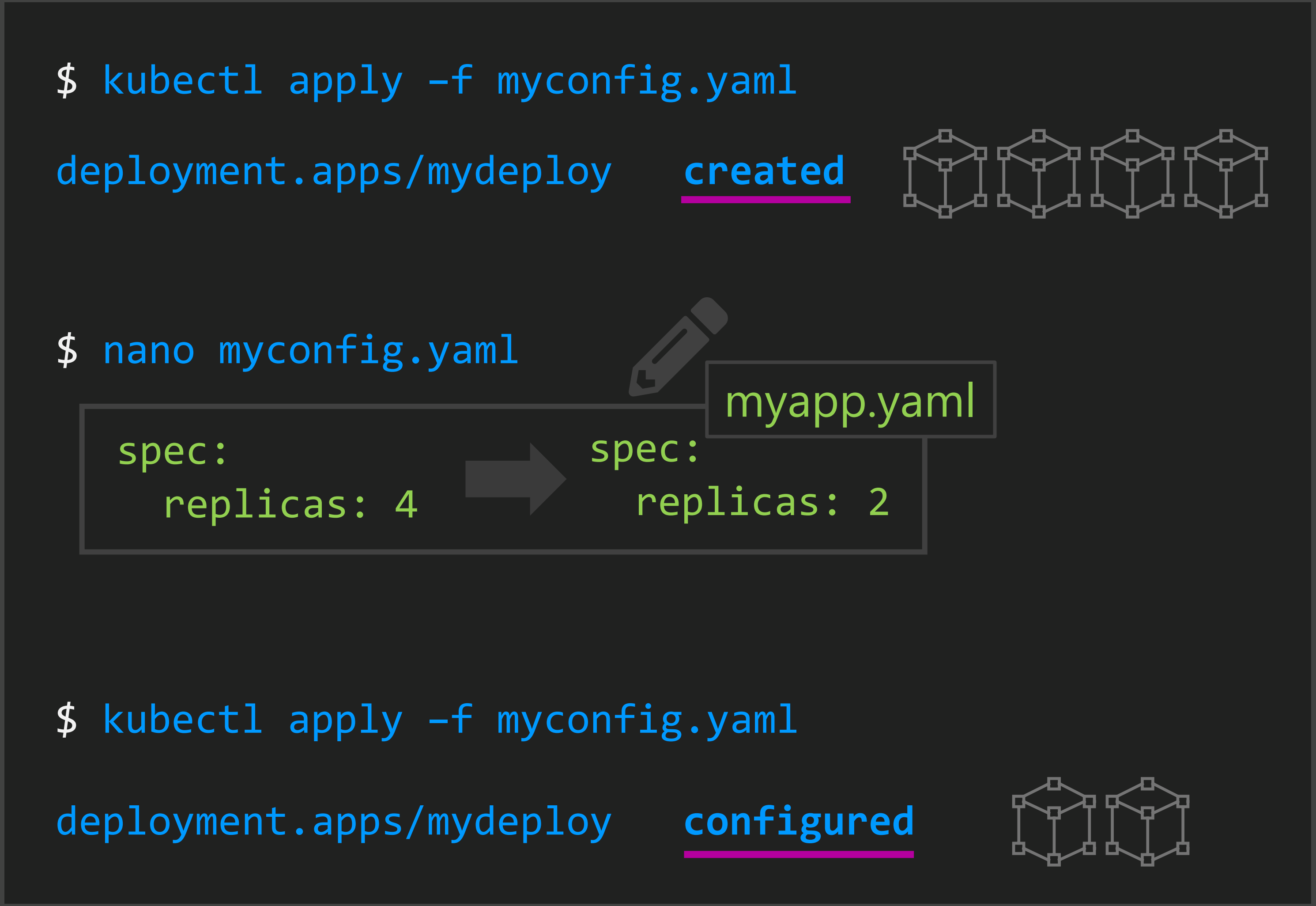
Note. JSON is also supported, but YAML is recommended for readability

Idempotent Updates & Desired State

Files describe desired state of the resources you configuring

```
kubectl apply
```

Kubernetes applies updates in idempotent way, modifying objects only if needed



```
$ kubectl apply -f myconfig.yaml

deployment.apps/mydeploy   created
```

```
$ nano myconfig.yaml
```

myapp.yaml

```
spec:
  replicas: 4
```

```
spec:
  replicas: 2
```

```
$ kubectl apply -f myconfig.yaml

deployment.apps/mydeploy   configured
```

Idempotency: *"The definition of the target state can be applied multiple times and if the system's state is unchanged, no changes are made to the system"*

Labels & Selectors

kubernetes.io/docs/concepts/overview/working-with-objects/labels

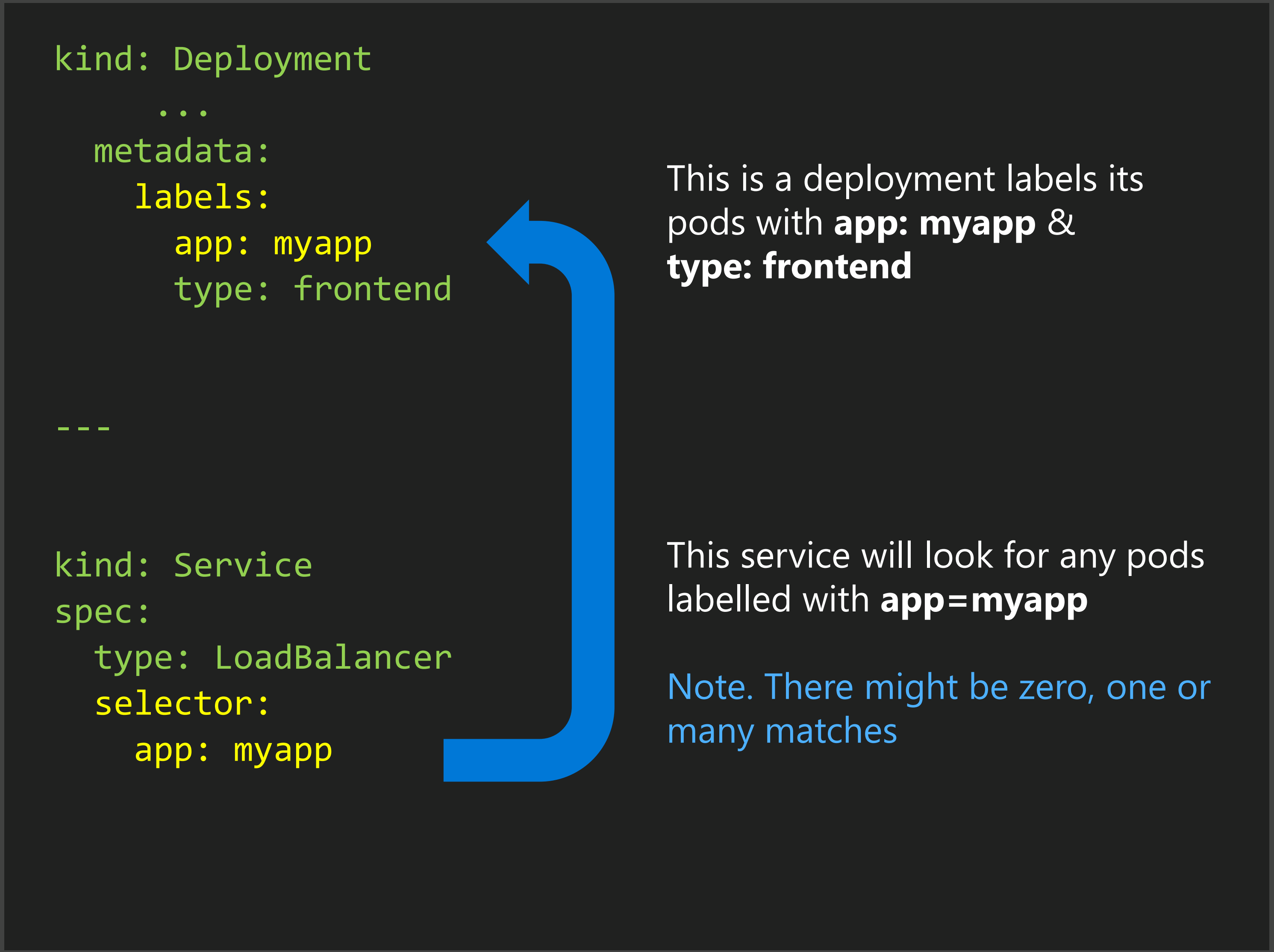
Kubernetes makes extensive use of labels and selectors

Labels are metadata on any object and are just `key:value` pairs of your choosing

Selectors are lookups that match one or more objects based on their **labels**

Uses

- Which pods are in a service
- Which nodes to run a pod on
- Which pods are in a deployment
- Logically group & tag resources



```
kind: Deployment
     ...
  metadata:
    labels:
      app: myapp
      type: frontend


---


kind: Service
spec:
  type: LoadBalancer
  selector:
    app: myapp
```

This is a deployment labels its pods with **app: myapp** & **type: frontend**

This service will look for any pods labelled with **app=myapp**

Note. There might be zero, one or many matches

Configuring Basic Workloads

Environmental Variables

kubernetes.io/docs/tasks/inject-data-application/define-environment-variable-container/

Environmental variables are the standard way to configure containers at runtime

Containerized app consumes environmental variables in OS standard way

Key value pairs

Application/container specific

Uses
- Application configuration - Parameter passing

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
  labels:
    purpose: demonstrate-envars
spec:
  containers:
  - name: envar-demo-container
    image: gcr.io/google-samples/node-hello:1.0
    env:
    - name: DEMO_GREETING
      value: "Hello from the environment"
    - name: DEMO_FAREWELL
      value: "Such a sweet sorrow"
```

Secrets

kubernetes.io/docs/concepts/configuration /secret

Hold sensitive information such as passwords, certs and API keys

Don’t place sensitive values as plain text in deployment files

Don’t “bake” secrets into your container images with config files

Can be mounted in pods as files or environmental variables

Uses

- TLS certificates
- Application configuration
- Authentication with private registry

```
$ kubectl create secret generic my-secret
--from-literal connString='admin:superSecret@some-host'

secret/my-secret created
```

myapp.yaml

```
containers:
    - name: my-web-server
    env:
    - name: DATABASE_CONNECTION_STRING
      valueFrom:
        secretKeyRef:
          name: my-secret
          key: connString
```

ConfigMaps

kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap

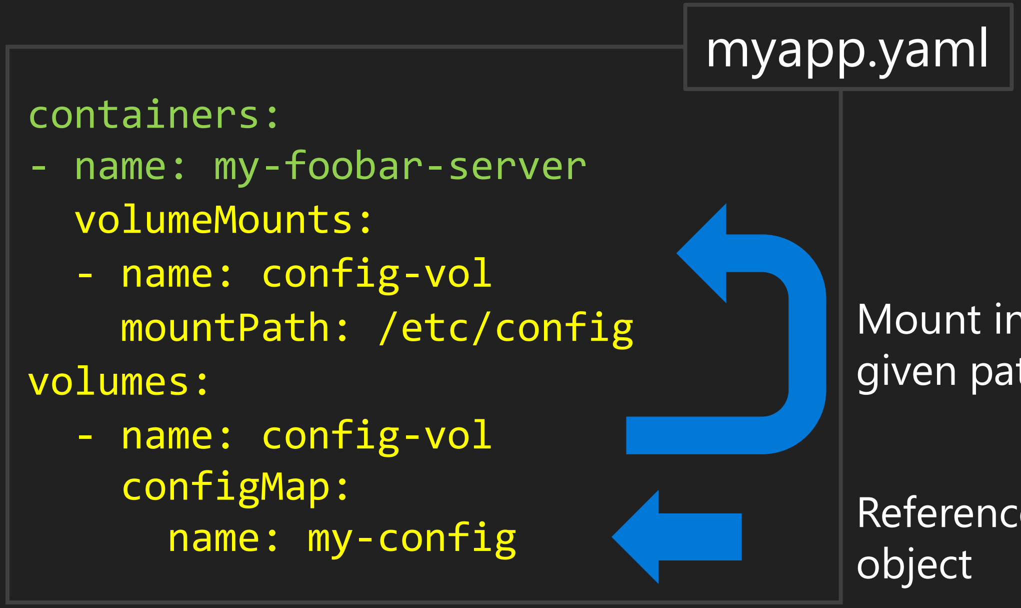
Hold application configuration data

Key value pairs (like secrets), YAML or free format (e.g. XML, conf)

Pass to containers as env vars or mount as volume

```
$ kubectl create configmap my-config
--from-file=/path/to/foobar.conf

configmap/my-config created
```



myapp.yaml

```yaml
containers:
- name: my-foobar-server
  volumeMounts:
  - name: config-vol
    mountPath: /etc/config
volumes:
  - name: config-vol
    configMap:
      name: my-config
```

Mount into container at given path

Reference to config map object

Uses
- Application configuration

Resource Management

kubernetes.io/docs/concepts/configuration/manage-compute-resources-container

Define compute (CPU & memory) resource limits and requests for containers

Allows Kubernetes to make better scheduling placement decisions

Limits are enforced, requests aren’t

CPU resources are fractions of 1 vCore

Uses

- Efficient use of cluster resources
- Prevent rogue workloads starving the cluster
- Good practice

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: demo-app
spec:
  containers:
  - name: db
    image: mysql
    resources:
      requests:
        memory: "64Mi"
        cpu: "0.25"
      limits:
        memory: "512Mi"
        cpu: "2.0"
```

Certain capabilities such as auto scaling are dependant on setting resources

Specifying limits & requests is optional but **STRONGLY** recommended

Liveness & Readiness Probes

kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes

Liveness probes tell Kubernetes your container is “alive”

Readiness probes tell Kubernetes your container is ready to accept traffic

Liveness probe failure can restart the container if it has hung

HTTP, TCP and command checks

Uses
- Maintain availability - Restart/termination of unhealthy containers - Efficient traffic routing

```
...

livenessProbe:
  httpGet:
    path: /status
    port: 8080
  initialDelaySeconds: 25
  periodSeconds: 10
  failureThreshold: 3
```

```
...

readinessProbe:
  exec:
    command: ["mysqladmin", "ping"]
  initialDelaySeconds: 30
  periodSeconds: 20
```

Specifying a liveness probe is optional but recommended

Commands & Arguments

kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/

Using the **command** property pass a starting command to a container

Use **args** to pass arguments to the container, pass an array of strings

Note. These correspond to the Docker Entrypoint and Cmd parameters

```
kind: Pod
metadata:
  name: command-demo
spec:
  containers:
  - name: command-demo-container
    image: debian
    command: ["printenv"]
    args: ["HOSTNAME", "KUBERNETES_PORT"]
```

Uses
- Use base images to run utilities & scripts - Debugging & trouble shooting - Application configuration - Parameter passing

Description	Docker field name	Kubernetes field name
The command run by the container	Entrypoint	command
The arguments passed to the command	Cmd	args

Beyond The Basics

https://servicemesh.io/

Service Mesh

Facilitates services to service calls inside Kubernetes

Features:

- Observability / tracing
- Error handling / retries / backoff
- Encryption / mTLS
- Routing / balancing
- Traffic splitting: blue-green / canary

Typically runs as a proxy sidecar in all your pods

Adds complexity & overhead

LINKERD

HashiCorp Consul

Istio

Open Service Mesh

Kuma

Kubernetes Ecosystem – *Many* Projects

Security / Policy

aqua

Gatekeeper

DevOps

HELM

Helm

Flux

Serverless

KEDA

dapr

Kubeless

Monitoring

Prometheus
Grafana
Opentracing

Network Plugins

Calico
Weave Net
Flannel

Load Balancing & Discovery

Envoy
Linkerd
CoreDNS

Storage

Portworx

Rook

landscape.cncf.io

The Kubernetes API

It’s Kind Of Important!

kubernetes.io/docs/reference/
#api-reference

Every object in Kubernetes and any interactions with the cluster are shaped by the API and the API spec

- YAML manifests schema
- `kubectl` commands

Served by the API server running on master node(s)

Kubernetes version dependant

Aggregated - hosts multiple APIs at multiple versions

`kubectl proxy` – Create a local tunnel to the API server

Additional Network Services

Ingress

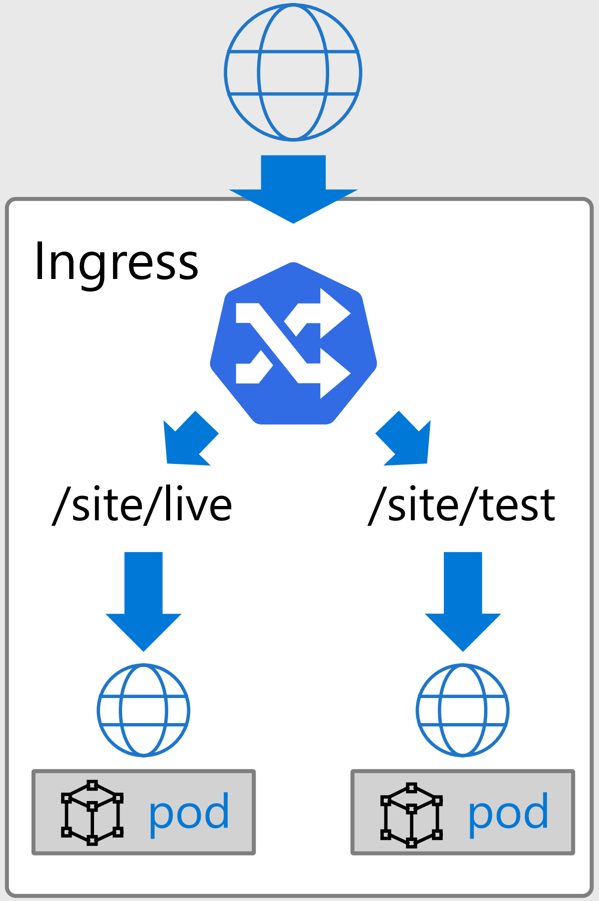
External access for HTTP and HTTPS

An *Ingress* allows you to **route** HTTP/HTTPS traffic to **services** based on URL and/or domain host name

Ingress object is a **set of rules** picked up and implemented by the *Ingress Controller*

Ingress Controller has a **public IP** and *LoadBalancer* service, it routes traffic to internal *ClusterIP* services

A large choice of controllers are available, e.g. NGINX



Use an Ingress when you want to route HTTP(S) traffic into your workloads and pods

External DNS

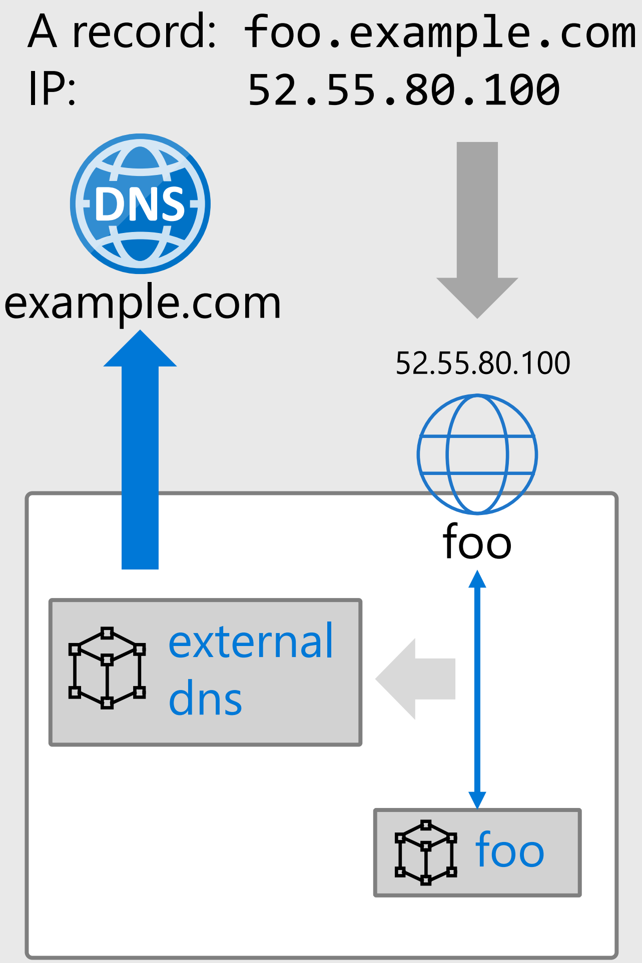
Optional Addon – Auto configuration of public DNS

Allows for **dynamic configuration** of DNS records

Seamlessly keep **public DNS** in sync with your Ingress and external services

Supports Azure DNS, AWS, CloudFlare, Google DNS etc

SIG project: github.com/kubernetes-sig/external-dns



Commonly used with an Ingress for host based external routing

Cert Manager

github.com/jetstack/cert-manager

Optional Addon – Automate issuing of TLS certs

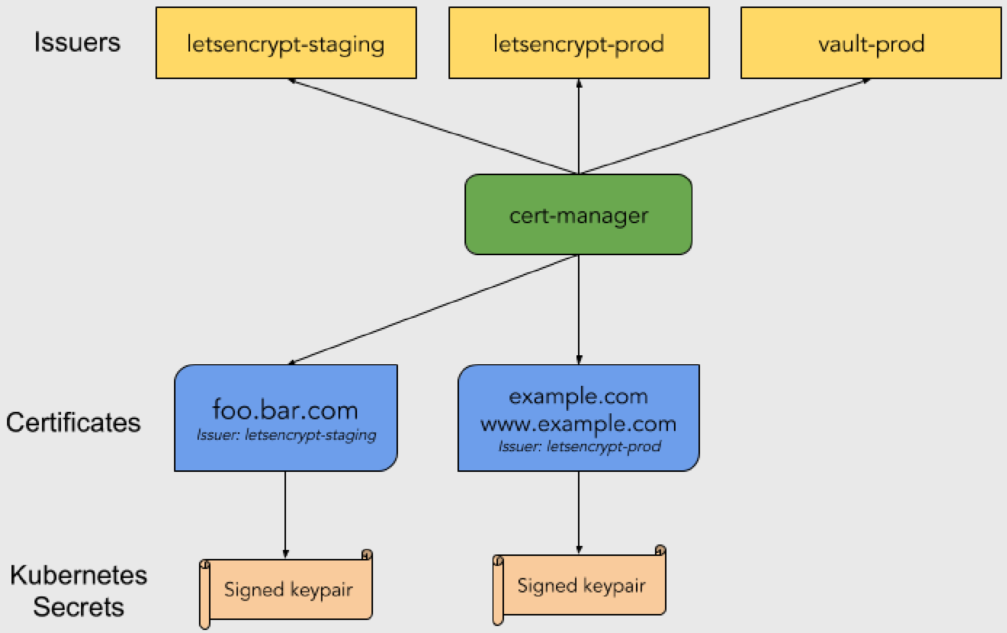
Ensures **certificates are valid** and up to date

Tightly coupled to *Ingress*, e.g. host rules

Renew certificates before expiry

Uses ACME issuers, i.e. **Let's Encrypt**

Project: https://cert-manager.io



Issue TLS certs for HTTPS access to services & Ingress

Putting It All Together

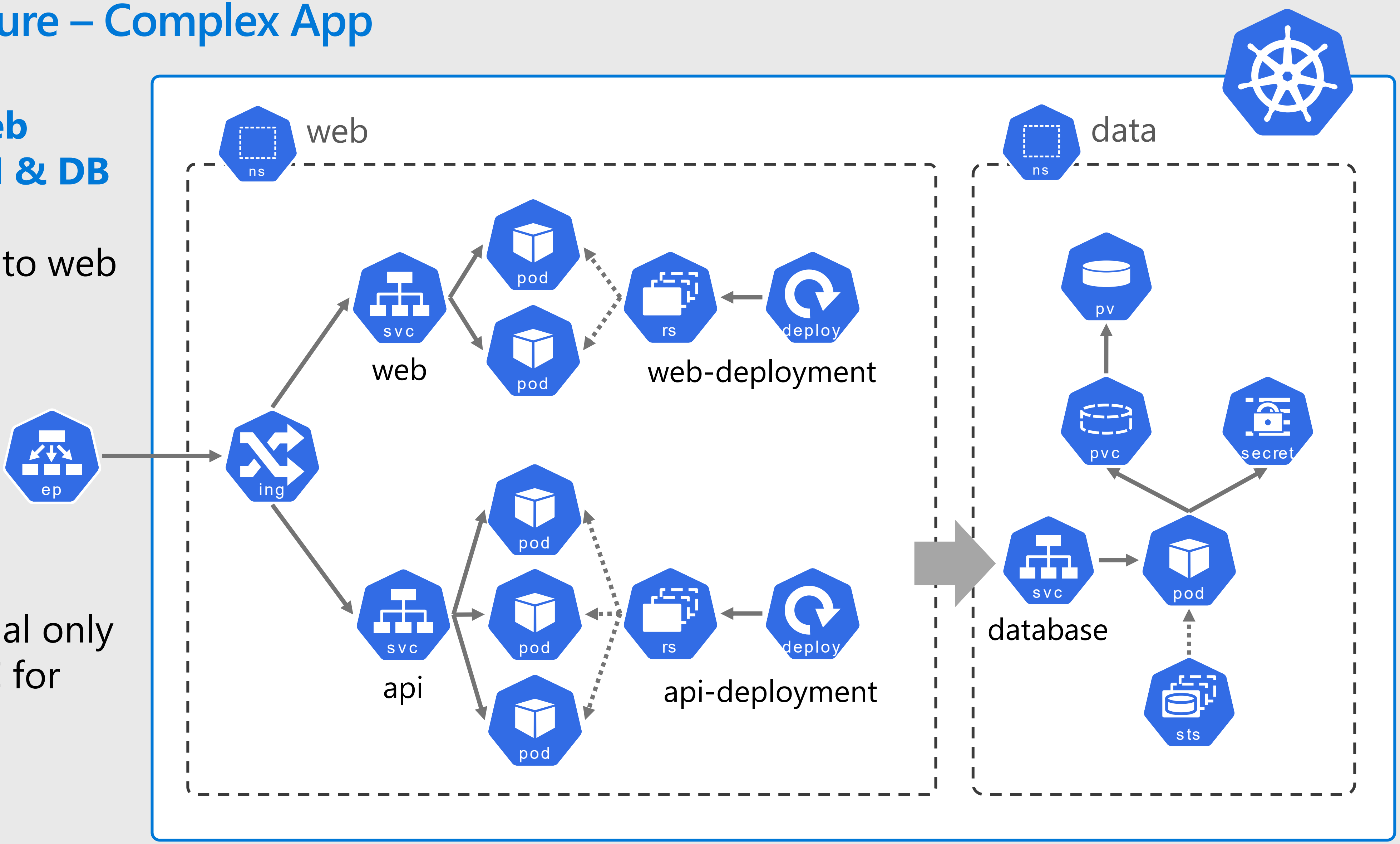
Sample Architecture – Complex App

Multi component web application, with API & DB

Ingress routing traffic to web and API pods

Ingress is exposed to the internet

Database running in StatefulSet with internal only service and using PVC for storage, plus secrets



Debugging and Troubleshooting Workloads

Describing Objects

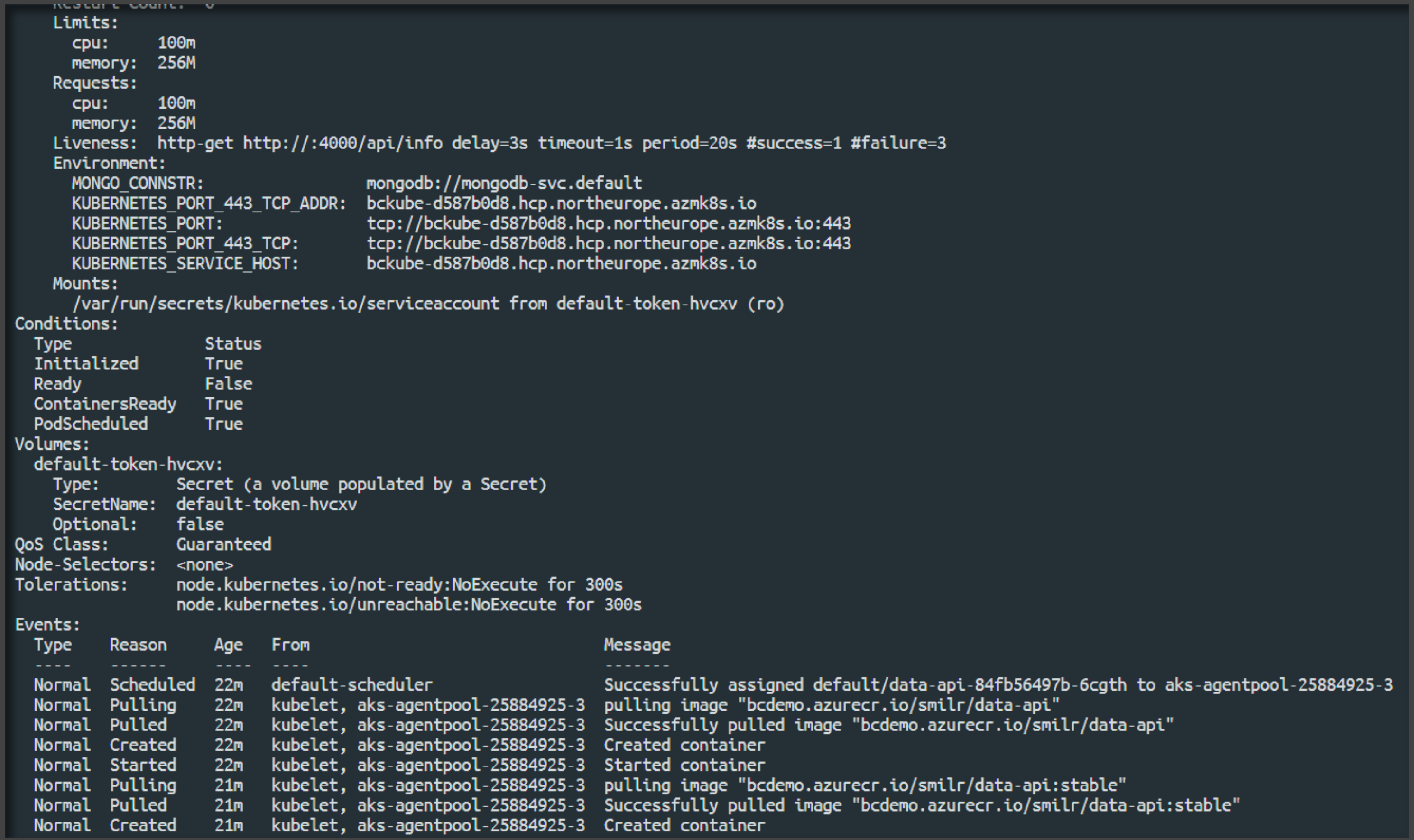
kubernetes.io/docs/tasks/debug-application-cluster

Use `kubectl describe` to inspect status of any object in your cluster

Returns events and all properties & status details

Use label selectors to query multiple objects

```
$ kubectl describe deploy/data-api
$ kubectl describe pod/data-api-84fb56497b-6cgth
$ kubectl describe service/frontend
$ kubectl describe pod -l app=data-api
```



```
    Restart Count:  0
    Limits:
      cpu:     100m
      memory:  256M
    Requests:
      cpu:     100m
      memory:  256M
    Liveness:  http-get http://:4000/api/info delay=3s timeout=1s period=20s #success=1 #failure=3
    Environment:
      MONGO_CONNSTR:                 mongodb://mongodb-svc.default
      KUBERNETES_PORT_443_TCP_ADDR:  bckube-d587b0d8.hcp.northeurope.azmk8s.io
      KUBERNETES_PORT:               tcp://bckube-d587b0d8.hcp.northeurope.azmk8s.io:443
      KUBERNETES_PORT_443_TCP:       tcp://bckube-d587b0d8.hcp.northeurope.azmk8s.io:443
      KUBERNETES_SERVICE_HOST:       bckube-d587b0d8.hcp.northeurope.azmk8s.io
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-hvcxv (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             False
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-hvcxv:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-hvcxv
    Optional:    false
QoS Class:       Guaranteed
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age   From                               Message
  ----    ------     ----  ----                               -------
  Normal  Scheduled  22m   default-scheduler                  Successfully assigned default/data-api-84fb56497b-6cgth to aks-agentpool-25884925-3
  Normal  Pulling    22m   kubelet, aks-agentpool-25884925-3  pulling image "bcdemo.azurecr.io/smilr/data-api"
  Normal  Pulled     22m   kubelet, aks-agentpool-25884925-3  Successfully pulled image "bcdemo.azurecr.io/smilr/data-api"
  Normal  Created    22m   kubelet, aks-agentpool-25884925-3  Created container
  Normal  Started    22m   kubelet, aks-agentpool-25884925-3  Started container
  Normal  Pulling    21m   kubelet, aks-agentpool-25884925-3  pulling image "bcdemo.azurecr.io/smilr/data-api:stable"
  Normal  Pulled     21m   kubelet, aks-agentpool-25884925-3  Successfully pulled image "bcdemo.azurecr.io/smilr/data-api:stable"
  Normal  Created    21m   kubelet, aks-agentpool-25884925-3  Created container
```

Uses

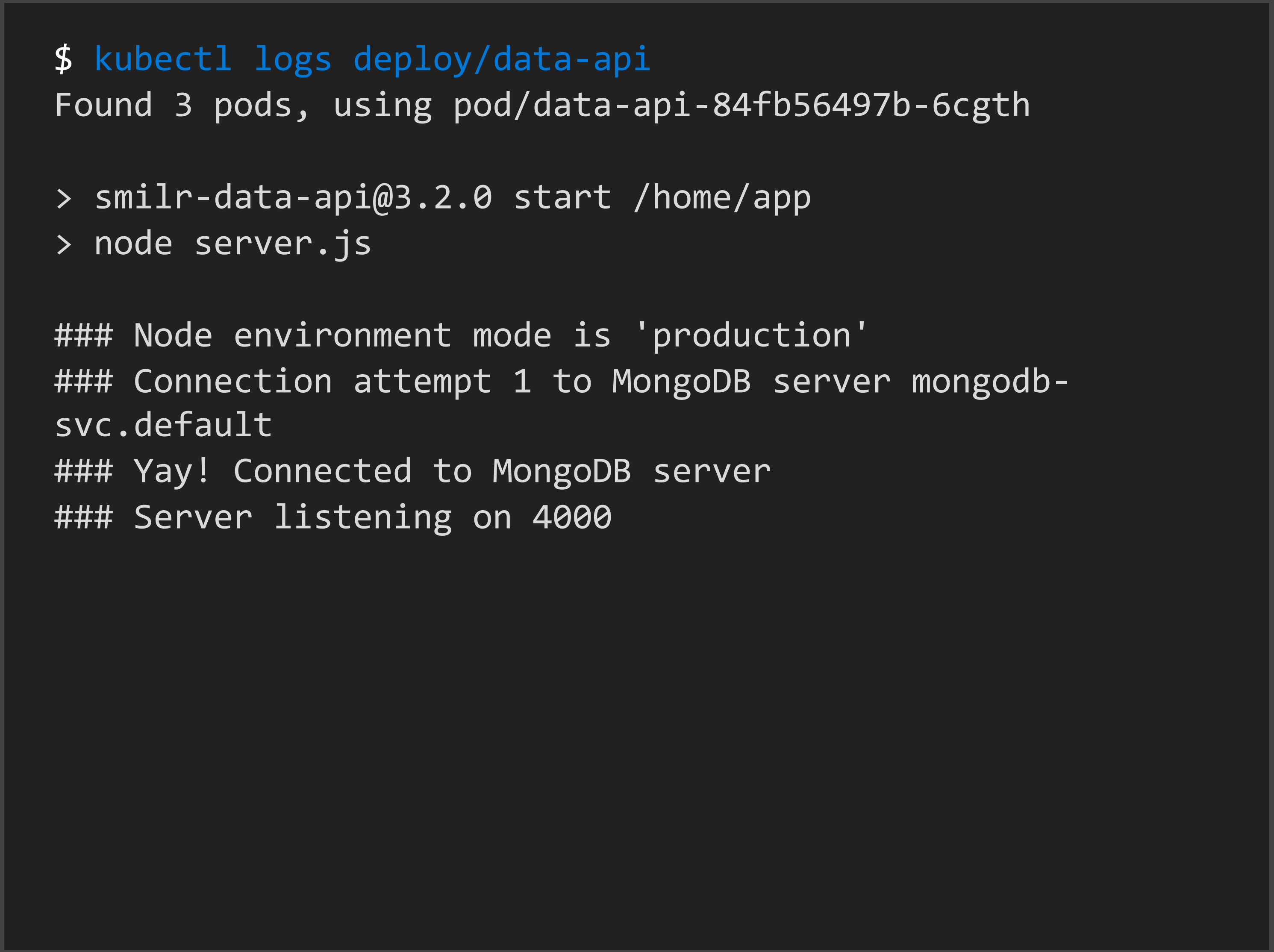
- View and understand the status of anything in your cluster
- Troubleshoot pending/failed workloads

Container Logs

Access stdout & stderr output from pods with:
kubectl logs

Get output from a deployment or pod or single container

Follow logs with -f



```
$ kubectl logs deploy/data-api
Found 3 pods, using pod/data-api-84fb56497b-6cgth

> smilr-data-api@3.2.0 start /home/app
> node server.js

### Node environment mode is 'production'
### Connection attempt 1 to MongoDB server mongodb-
svc.default
### Yay! Connected to MongoDB server
### Server listening on 4000
```

Uses

- View any errors output from containers
- See what your workloads are doing

Get Shell Access

kubernetes.io/docs/tasks/debug-application-cluster/get-shell-running-container

Create processes in running containers

Use `kubectl exec`

Use `-it` switch and sh or bash to create an interactive shell

Can also run a single non-interactive command

```
$ kubectl exec frontend-58b84f7d7-fb6rg -- ps -ef
PID   USER     TIME  COMMAND
    1 root      0:00 npm
   22 root      0:00 node server.js
   36 root      0:00 ps -ef


$ kubectl exec -it frontend-58b84f7d7-fb6rg -- bash
bash-4.4# ls -a
node_modules        package.json        server.js

bash-4.4# uname
Linux
```

Uses

- Low level debugging
- Interactive troubleshooting

Advanced Pod Configuration

Deeper Dive on Manifests

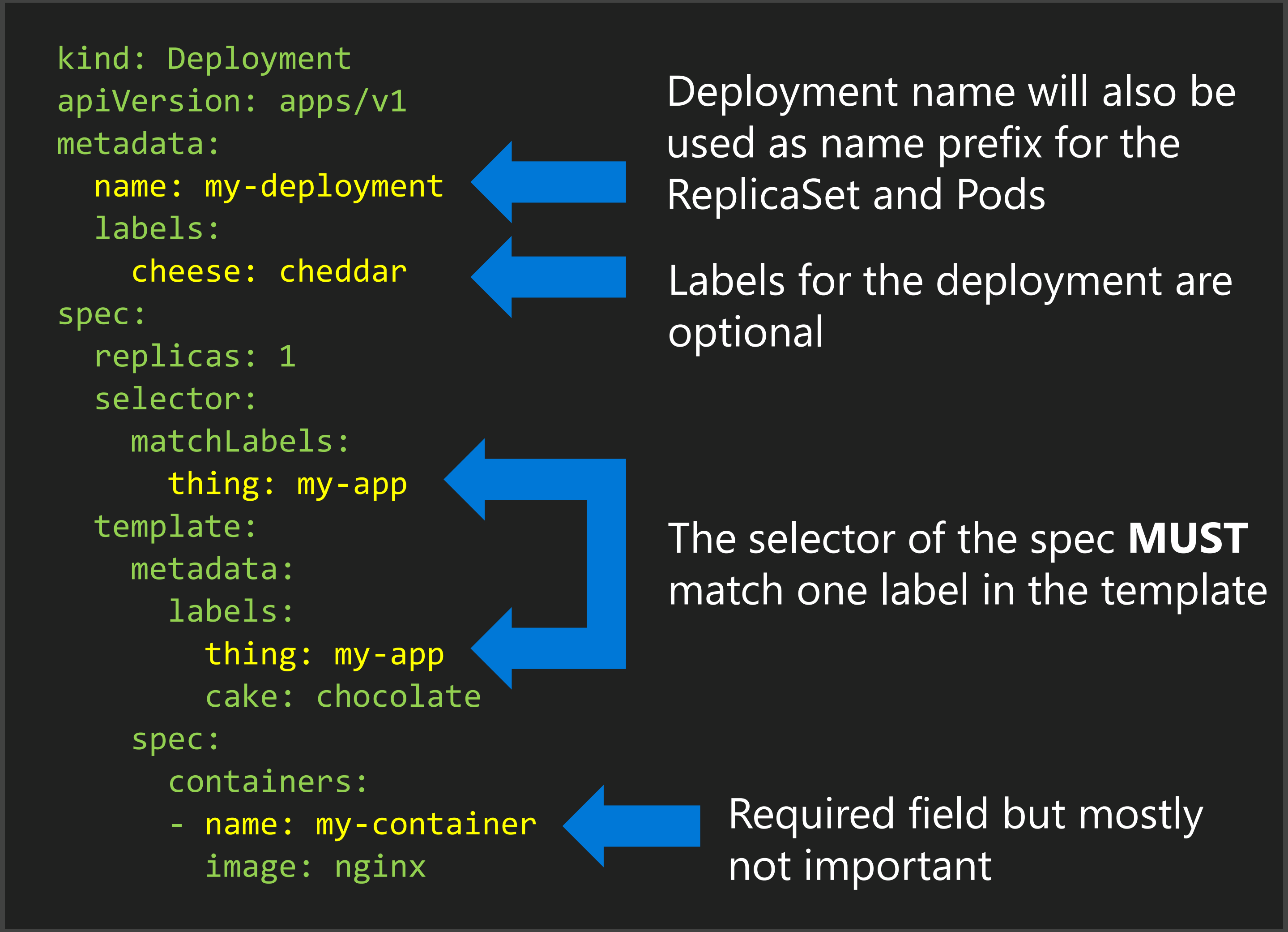
Manifests for *Deployments*, *StatefulSets* and *DaemonSets* have a similar pattern & structure

The `spec` part contains `replicas` and a `selector` and also a `template` for the objects it will replicate

The `template` will contain another `spec`, typically a *Pod* `spec`

A *Pod* `spec` contains one or more `containers`

?? So what do all these labels mean ??



```yaml
kind: Deployment
apiVersion: apps/v1
metadata:
  name: my-deployment
  labels:
    cheese: cheddar
spec:
  replicas: 1
  selector:
    matchLabels:
      thing: my-app
  template:
    metadata:
      labels:
        thing: my-app
        cake: chocolate
    spec:
      containers:
      - name: my-container
        image: nginx
```

Deployment name will also be used as name prefix for the ReplicaSet and Pods

Labels for the deployment are optional

The selector of the spec **MUST** match one label in the template

Required field but mostly not important

Init Containers

kubernetes.io/docs/concepts/workloads/pods/init-containers

Init Containers are optional special containers that run only once when a pod is started

Init containers run to completion (terminate)

Main containers in a pod will not start until all Init Containers have run

Uses

- Application configuration
- Bootstrapping apps
- Running utility & start-up scripts
- Data injection

```
...

initContainers:
  - name: init-mysql
    image: bencuk/mysqldb
    command: ["./scripts/checkDB"]
```

```
...

initContainers:
  - name: init-demodata
    image: bcdemo.azurecr.io/smilr/data-api
    command: ['sh', '-c', 'cd demoData && node demodb.js']
    env:
      - name: MONGO_CONNSTR
        value: mongodb://mongodb-svc.default
      - name: WIPE_DB
        value: "true"
```

Node Selector

kubernetes.io/docs/concepts/configuration/assign-pod-node/#nodeselector

A simple constraint to which Nodes are eligible to run a Pod

Key value pairs of labels, to be matched against Node's labels

Not a 'hard rule', other Pods that have no nodeSelector can still land on the node

Pod

```
kind: Pod
metadata:
  name: machineLearning
spec:
  containers:
  - name: trainModel
    image: ml-image:latest
  nodeSelector:
    hardware: gpu
```

Can be scheduled on

Node

```
Name:         aks-nodepool1-18655374-vmss000000
Roles:        agent
Labels:       agentpool=nodepool1
              beta.kubernetes.io/arch=amd64
              hardware=gpu
```

Uses
- Assign workloads requiring special hardware or resources, e.g. GPU - Physical partitioning of cluster - Separating noisy Pods

Affinity and Taints

kubernetes.io/docs/concepts/configuration/taint-and-toleration/

kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity

Affinity and anti-affinity provide advanced ways to control Pod placement
- 'Hard' rules and 'soft' rules
- Weighting and expressions

Taints and Tolerations. Taints are applied to *Nodes* and will not accept *Pods* that don't have a matching Toleration

```
$ kubectl taint nodes myNode001 team=team1:NoSchedule
node/myNode001 tainted
```

```
kind: Pod
spec:
  tolerations:
  - key: team
    operator: Exists
    value: "team1"
    effect: NoSchedule
```

Uses
- Assign workloads requiring special hardware or resources, e.g. GPU - Physical partitioning of cluster - Separating noisy Pods

```
kind: Pod
spec:
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
    preferredDuringSchedulingIgnoredDuringExecution:
  podAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
    preferredDuringSchedulingIgnoredDuringExecution:
```

Unlike taints & nodeSelectors, affinity rules can also apply to **pods**. i.e. do or don't schedule these pods together

Sidecars

Pods co-locate multiple containers together, sharing network and storage

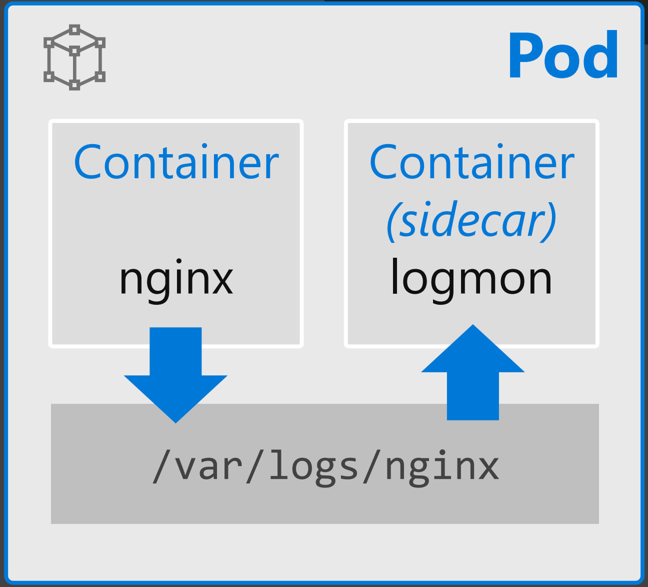
Sidecar is a pattern where additional containers provide enhancing/optional capabilities

Bind containers together to form a single cohesive unit of service

Uses

- Decompose architecture
- Build services incrementally
- Bolt on features

```yaml
kind: Pod
metadata:
  name: monitored-webapp
spec:
  containers:
  - name: webserver
    image: nginx
  - name: log-monitor
    image: my-log-monitor
    args: ["--log-dir", "/var/logs/nginx"]
```



Scaling

Manually Scaling

Scale stateless workloads by controlling the number of replicas of a Deployment

Be careful scaling StatefulSets, unless the workload/application is “cluster aware”

DaemonSets don’t require scaling

Uses

- Horizontally scale
- Distribute work around cluster
- Remove single points of failure

```
$ kubectl scale deploy/myApp --replicas=5

deployment.extensions/myApp scaled
```

OR

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: myApp
spec:
  replicas: 5
```

Horizontal Pod Autoscaler (HPA)

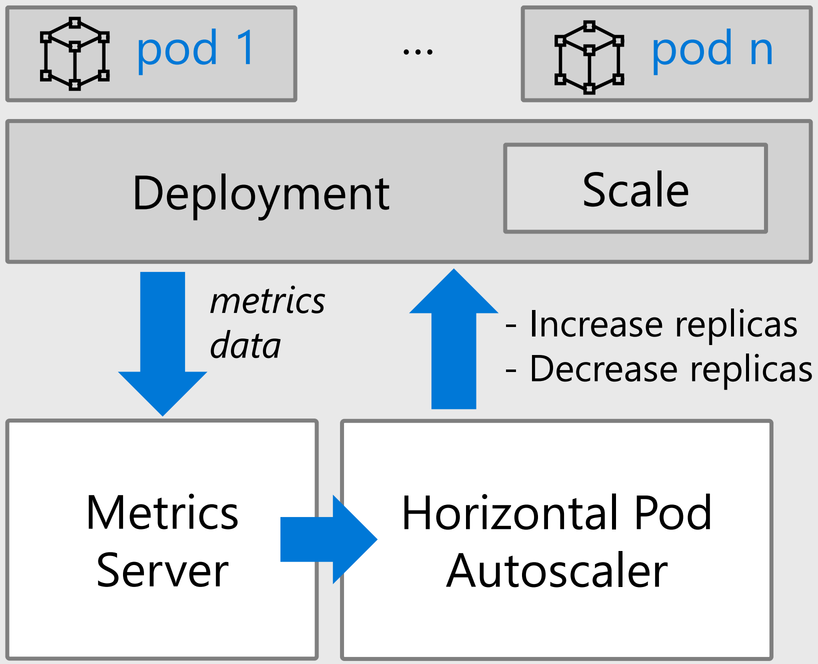
Automatically scale stateless workloads

Use to horizontally **scale stateless** *Pods* in *ReplicaSet/Deployment*

Rules define desired *Pod* replica count based on **observed metrics**

Takes metrics from the metrics API fed from the *Metrics Server (*

Supports extension via custom metrics



Dynamically scale stateless workloads across available nodes

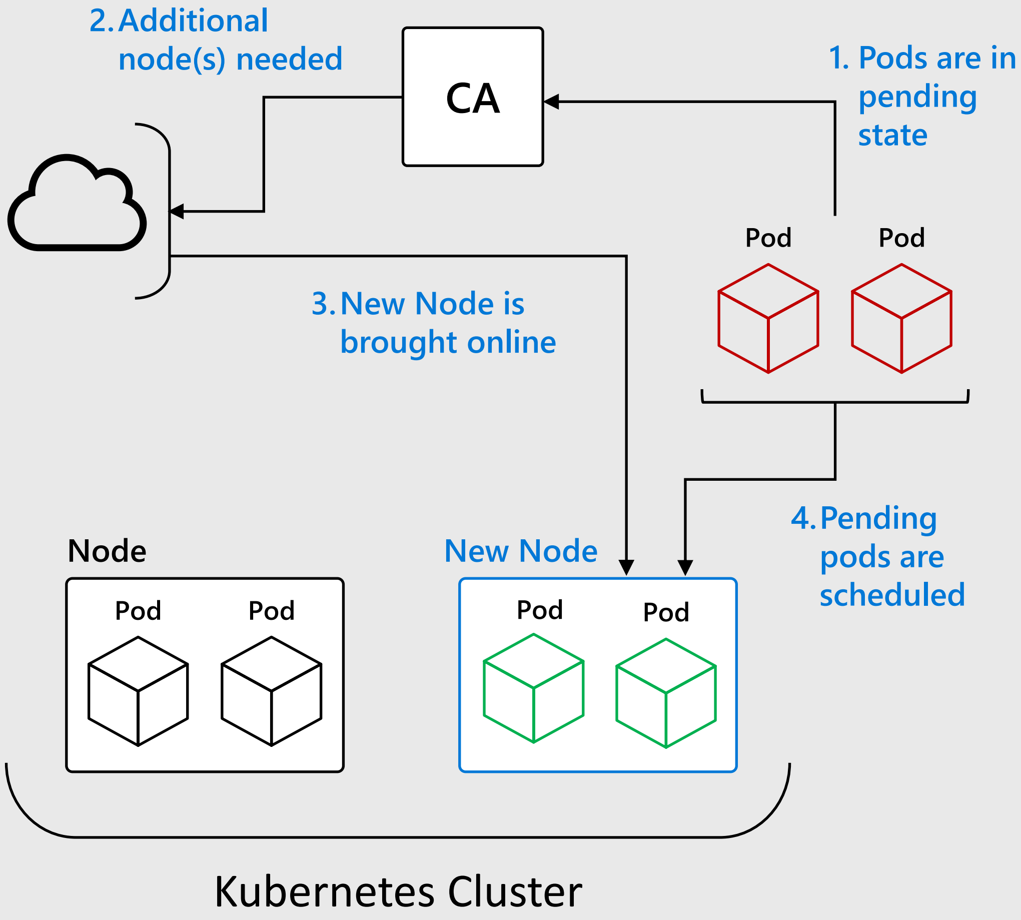
Cluster Autoscaler (CA)

Automatically scale cluster resources

Adjusts the size of the **Kubernetes cluster**, adding & removing *Nodes* when

- Pods are **pending** state due to insufficient resources (**scale out**)
- Nodes have been underutilized for a period of time (**scale in**)

Tightly coupled to the cloud and environment hosting the cluster & Nodes



2. Additional node(s) needed
CA
1. Pods are in pending state
Pod
Pod
3. New Node is brought online
Node
Pod
Pod
New Node
Pod
Pod
4. Pending pods are scheduled
Kubernetes Cluster

Scale cluster wide by adding/removing Nodes

Extending Kubernetes

Extra Advanced

Custom Resources Definitions (CRDs)

Extend the Kubernetes API and object model

An **extension** of the Kubernetes API that is not available in a default Kubernetes installation

Can represent application specific or generalised entities, e.g. *BackupJob*, *Report*

A CRD is schema & state – **doesn't define behaviour**

Manage via the API & kubectl same as standard built-in resource types

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
spec:
  names:
    plural: pets
    kind: Pet
openAPIV3Schema:
  type: object
  properties:
    petName:
      type: string
```

```
$ kubectl get pets
```

Extend Kubernetes with new entities and data models

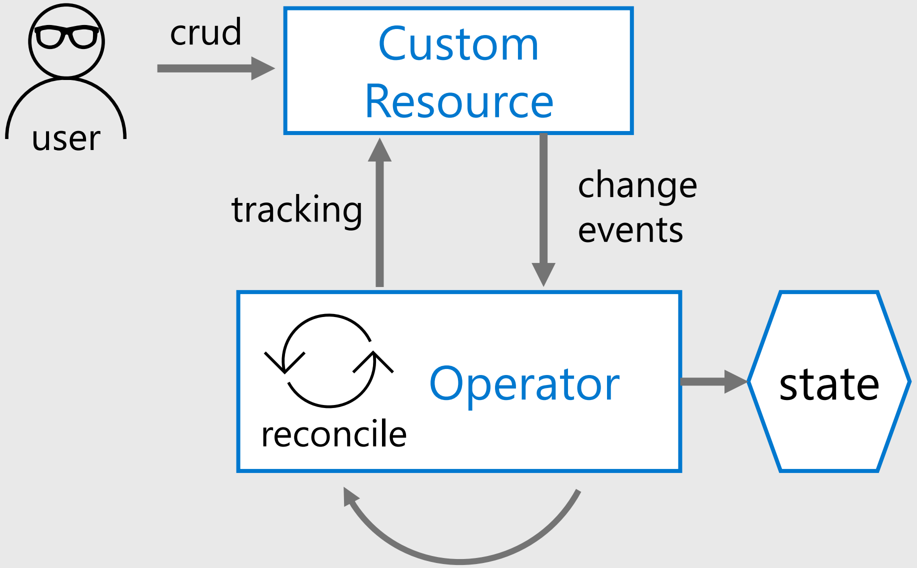
Operators

The Operator pattern: CRD + Custom Controller

An operator is custom controller that acts on a given set of CRDs – defines behaviour

Controller is run as a regular pod & deployment

Custom operator reconciler code subscribes to the Kubernetes API & watches for CRD changes

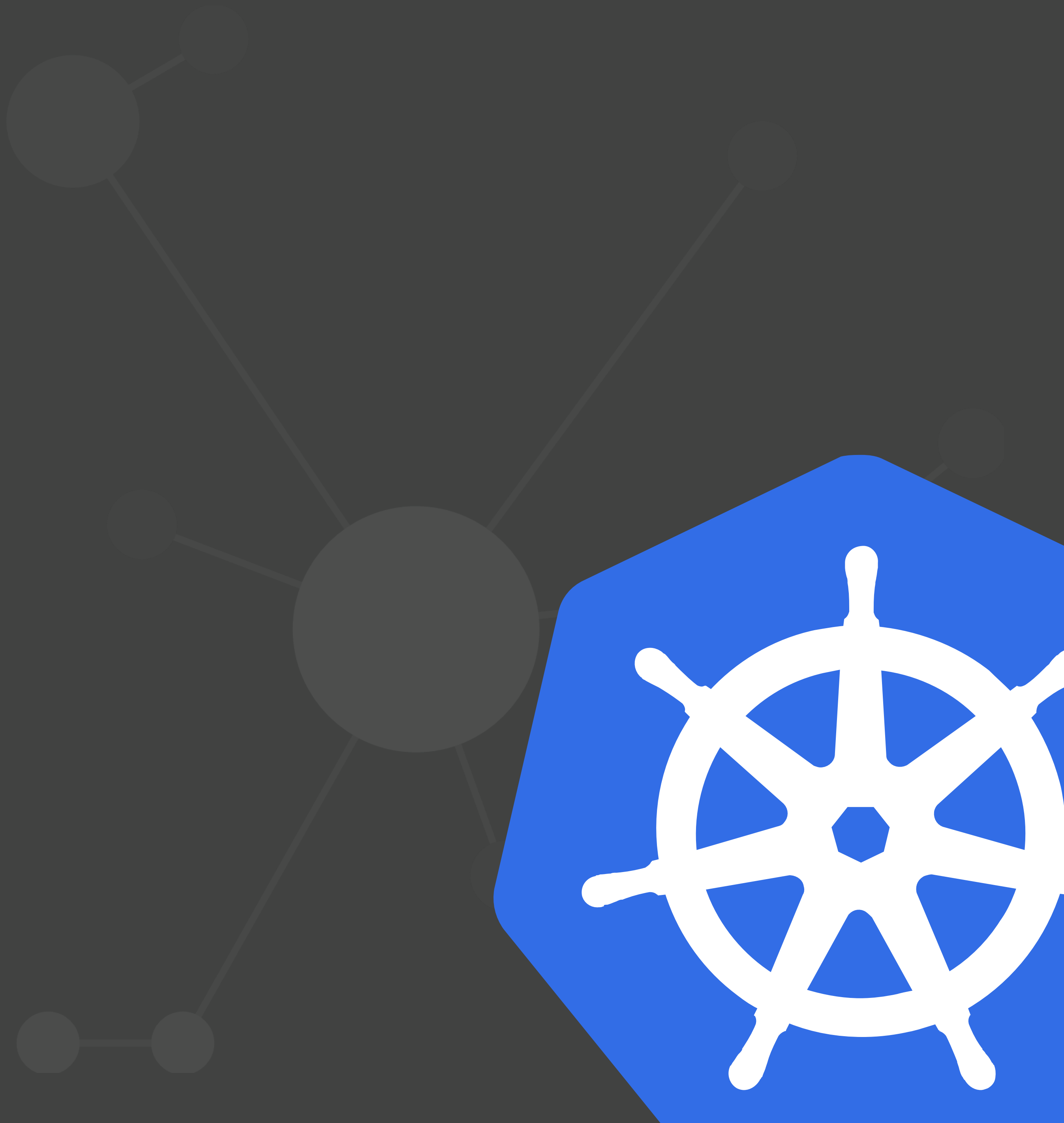


Discover & use:
- artifacthub.io
- operatorhub.io

Build your own:
- Kubebuilder
- KUDO
- Operator SDK

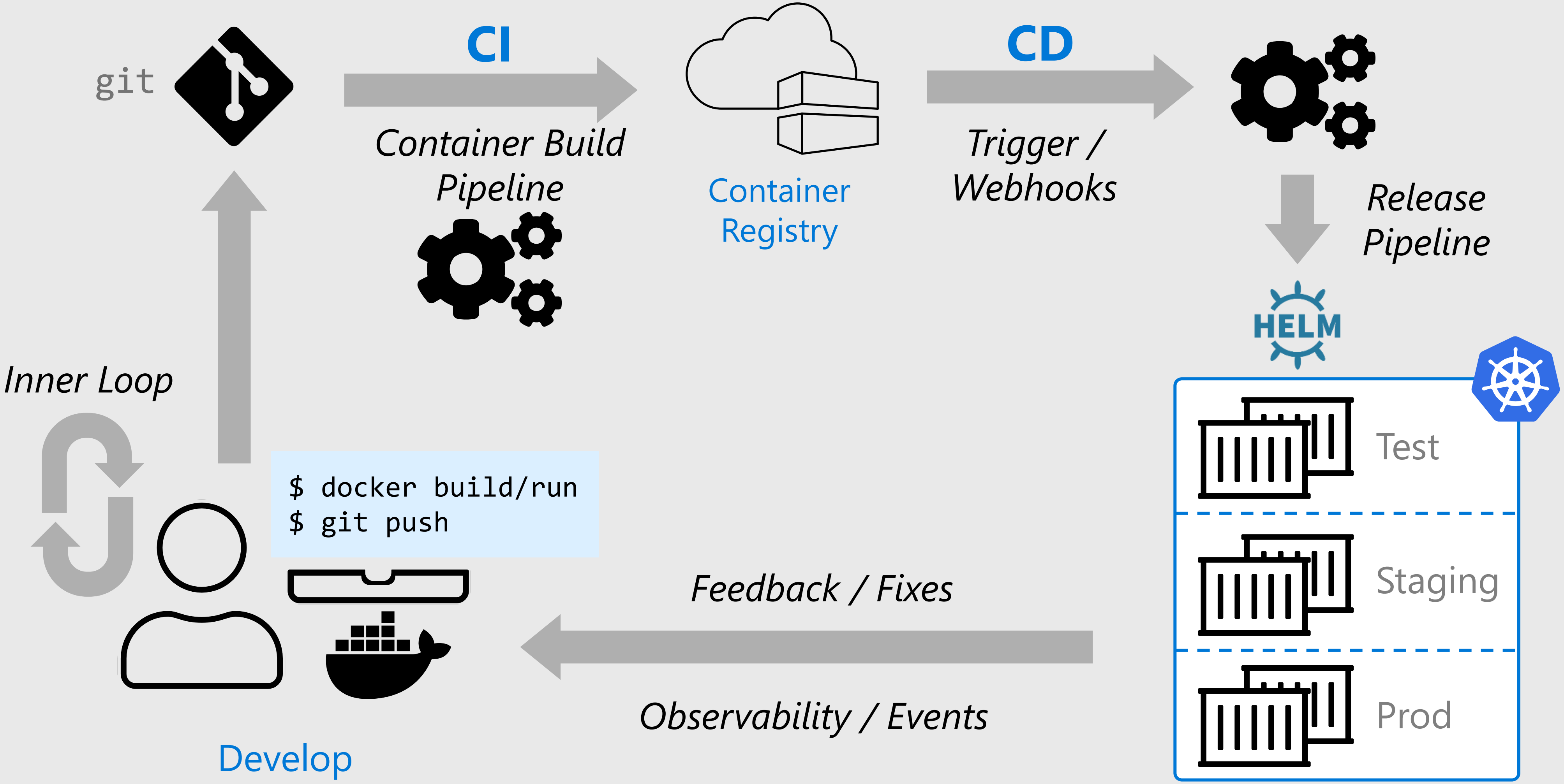
A common and easily understood way to extend Kubernetes

DevOps



Common DevOps Containers Lifecycle

CI/CD flow with pipelines/workflows and Helm



```
$ docker build/run
$ git push
```

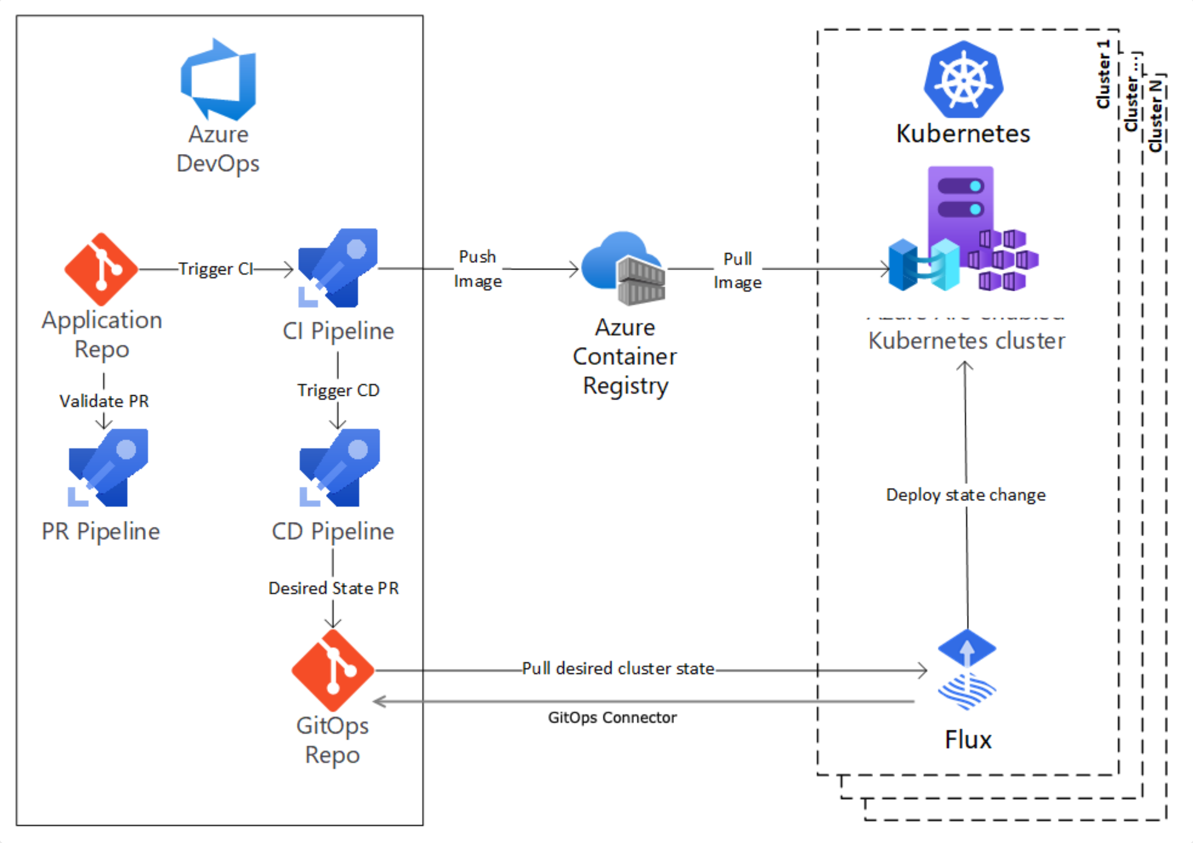
GitOps

Declarative approach to IaC & continuous deployment

GitOps is new approach & a set of practices & tools to manage infrastructure & application configurations using Git.

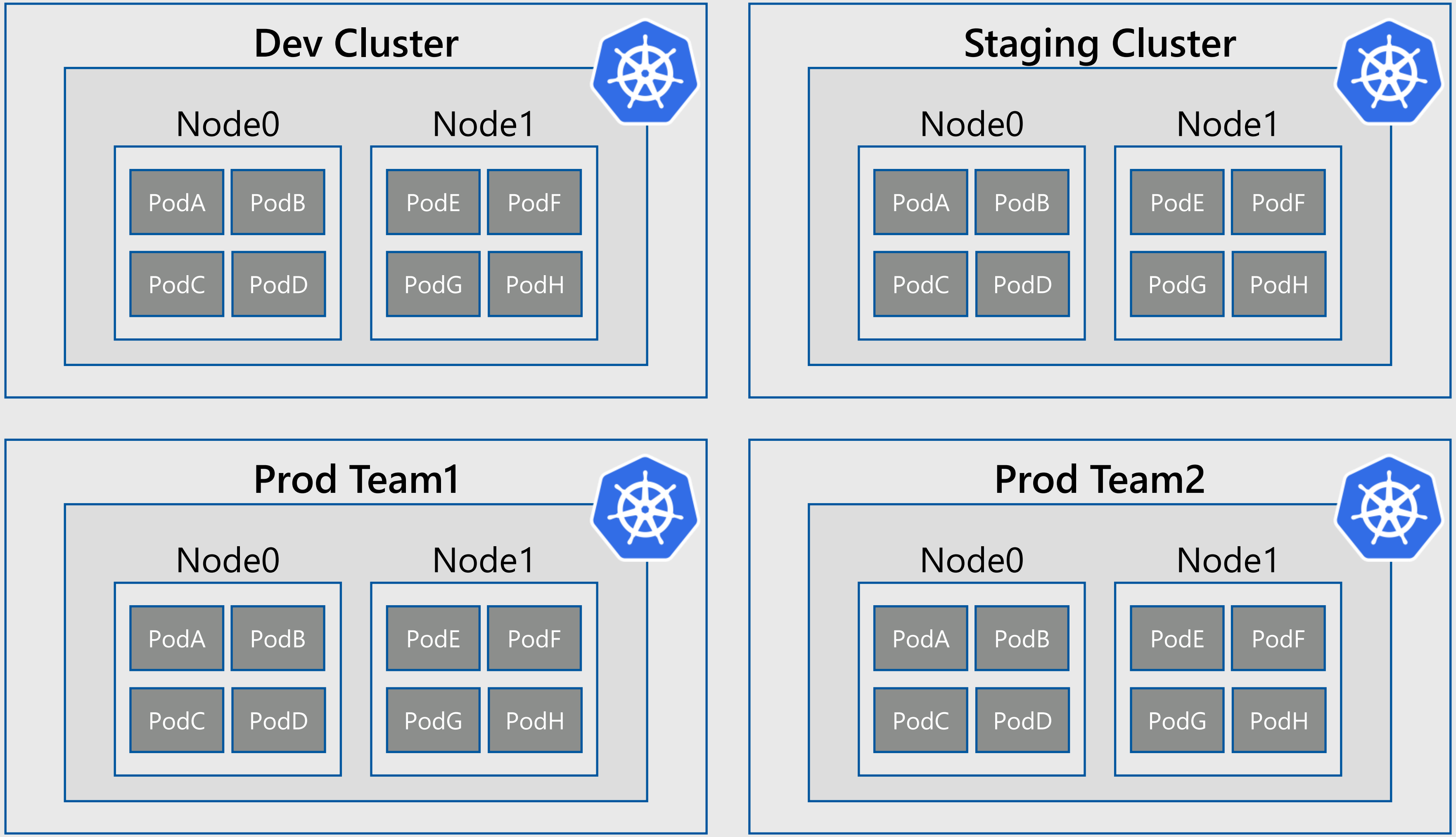
A "pull" approach vs "push" – special controllers (e.g. Flux) in Kubernetes watch and poll for changes and reconcile the updates.

More complex & not as well adopted as the Helm/kubectl push CD flow but has some advantanges.



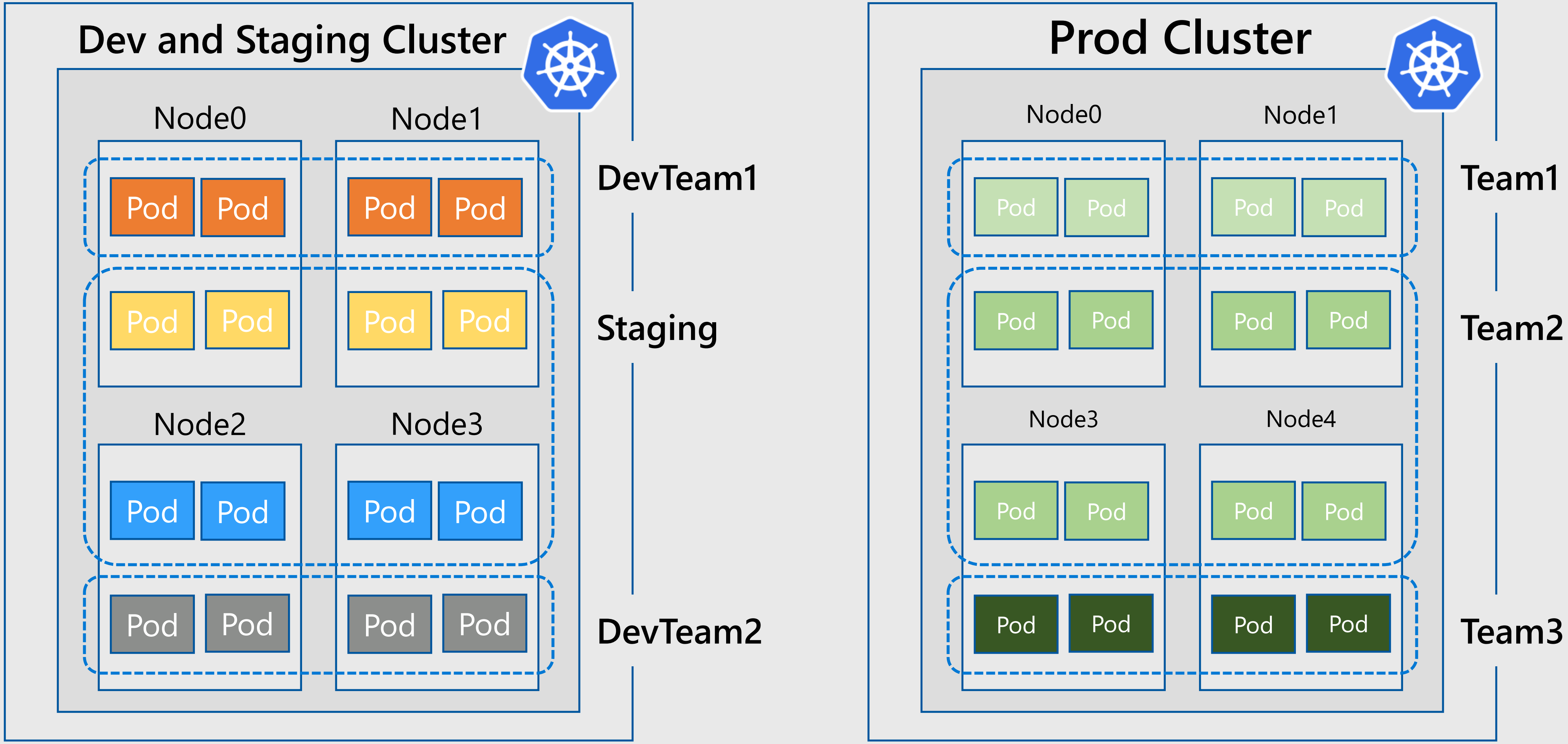
Alternative approach for managing and deploying applications to Kubernetes

Cluster Isolation Patterns: Physical Isolation



Simple but can waste resources, management overhead

Cluster Isolation Patterns: Logical Isolation



Requires planning & some Kubernetes expertise

Helm

Package Manager for Kubernetes

Helm simplifies **deployment** into Kubernetes using *charts*

A chart consists of one or more Kubernetes YAML **templates** + supporting files

Helm charts support dynamic **parameters & functions** important for automated pipeline deployments

Thousands of charts exist for standard software, tools & packages

https://helm.sh

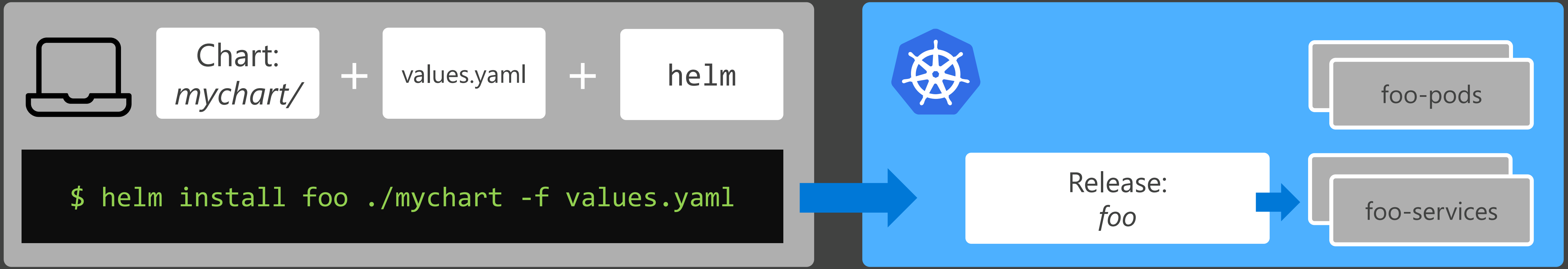
CLOUD NATIVE
COMPUTING FOUNDATION

- Use Helm to install software/apps in your cluster
- Create Helm charts for your own apps, for CI/CD releases

Helm – The Basics

docs.helm.sh/glossary

`helm`	Client tool to manage and work with Helm
Chart	Package of Kubernetes resources in template form
Template	Kubernetes YAML with directives in Go template language format, e.g. {{ blah }}
Release	When installing a chart into Kubernetes it becomes a release
Values	Used at install time to customise the release, either from CLI or file
Dependency	A chart can require other external charts, Helm will automatically pull/update

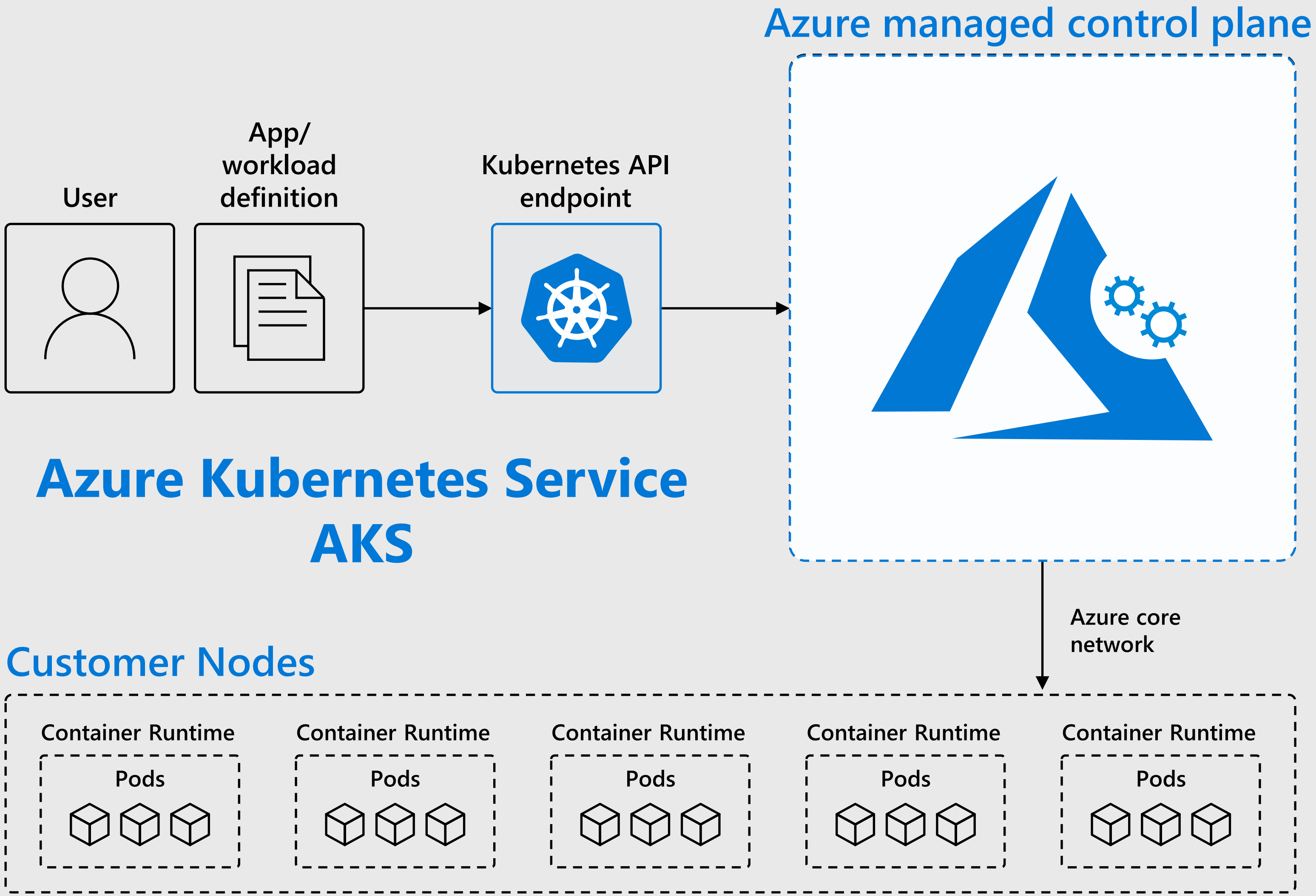


```
$ helm install foo ./mychart -f values.yaml
```

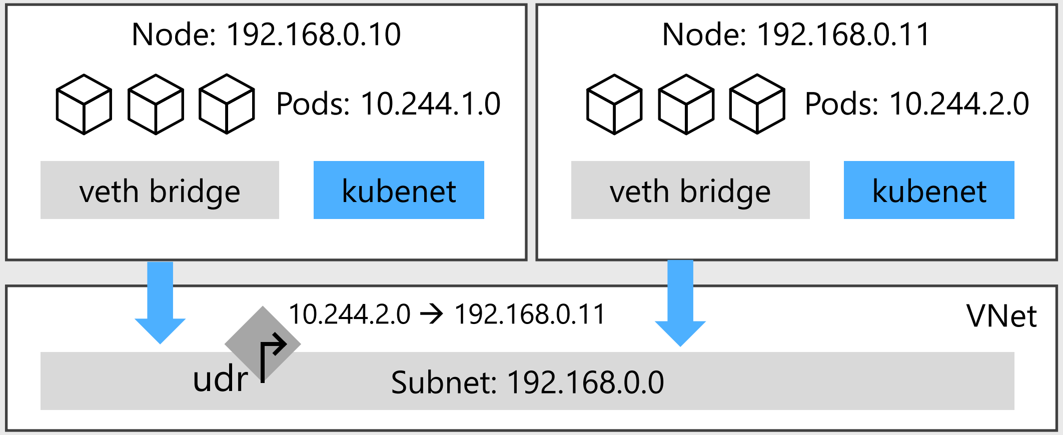
Azure Kubernetes Service

Managed Kubernetes on Azure – AKS

- Automated upgrades, patches
- High reliability, availability
- Easy, secure cluster scaling
- Self-healing
- API server monitoring
- At no charge



AKS Networking Models



Basic

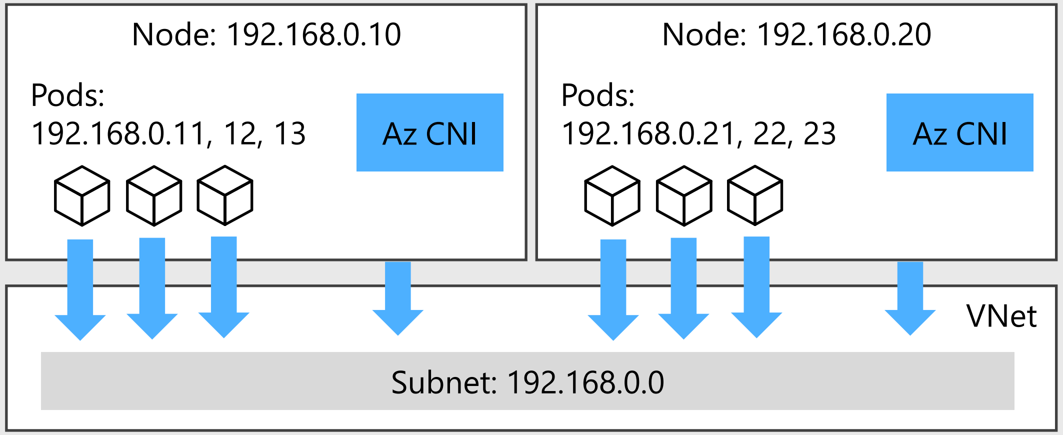
Uses **kubenet** network plugin

Pods are not in a VNet or same subnet as Nodes

Pods live behind bridge, and UDR is used

Simple but has limitations & performance impact

No need to allocate/reserve IPs for Pods



Advanced

Uses **Azure CNI** network plugin

Both Nodes and Pods on same subnet + VNet

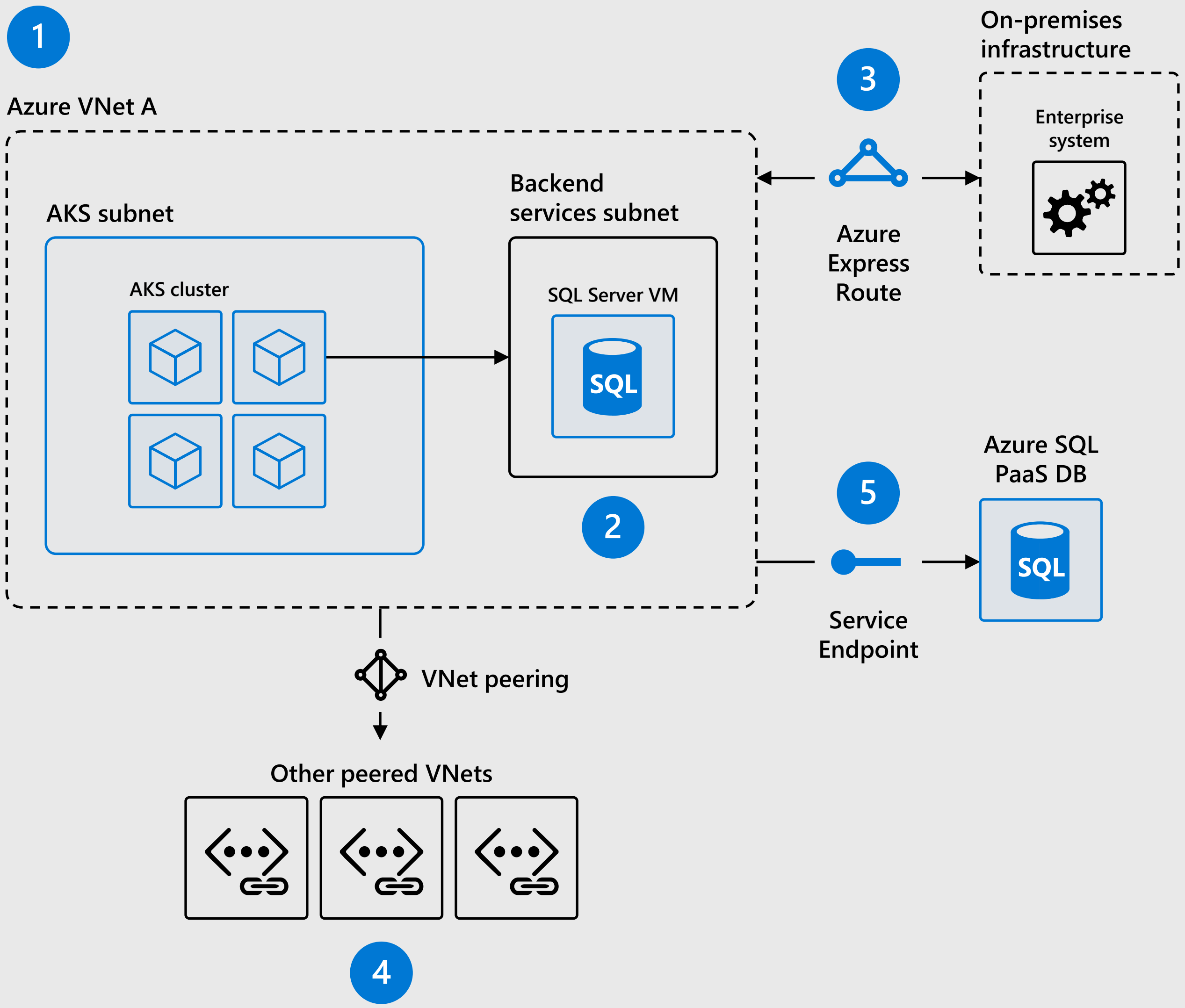
Works with peering, service endpoints, ExpressRoute

Better performance

Each pod requires an IP allocated to it

AKS Advanced Networking

1. Uses Azure subnet for both your pods and cluster VMs
2. Allows for connectivity to existing Azure IaaS/VMs in the same VNet
3. Use ExpressRoute to connect to on-premises infrastructure
4. Use VNet peering to connect to other VNets
5. Connect AKS cluster securely and privately to Azure PaaS using VNet endpoints



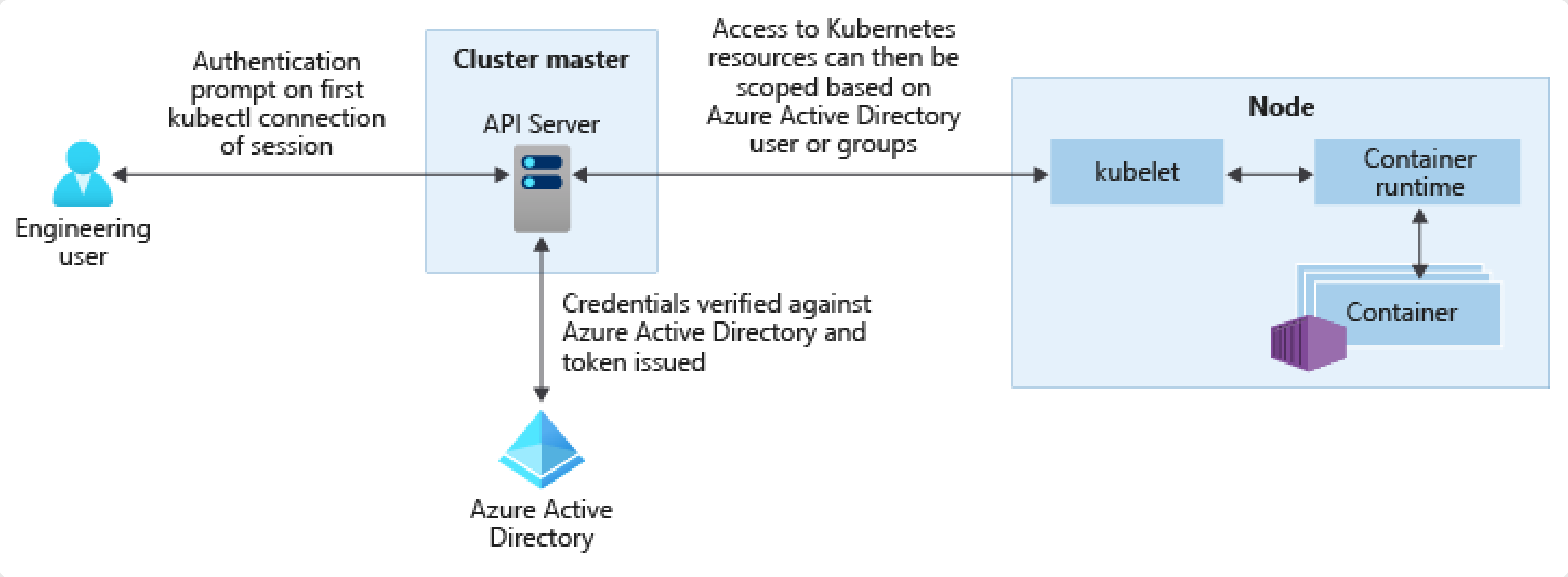
Azure Active Directory

https://docs.microsoft.com/azure/aks/managed-aad

Use Azure Active Directory to authenticate AKS access

Assign roles & permissions in Kubernetes RBAC, based on AAD users & groups

Roles & groups in AAD define who has access to the cluster


Authentication prompt on first kubectl connection of session
Cluster master
API Server
Access to Kubernetes resources can then be scoped based on Azure Active Directory user or groups
Node
kubelet
Container runtime
Container
Engineering user
Credentials verified against Azure Active Directory and token issued
Azure Active Directory

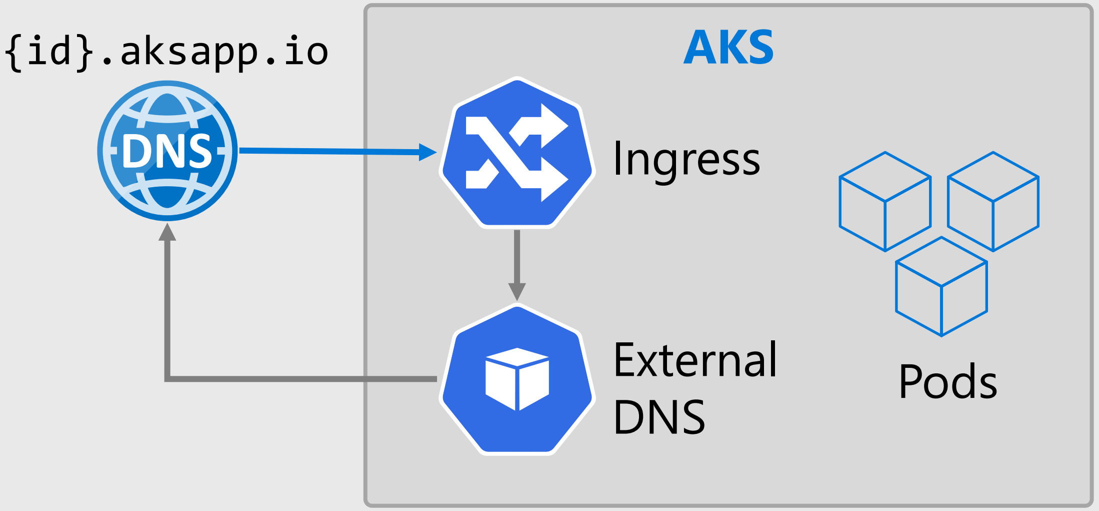
‘HTTP application routing’ Add-On

docs.microsoft.com/azure/aks/http-application-routing

Simplifies Ingress and DNS for external access to HTTP services in cluster

Creates:
- NGINX Ingress controller (in cluster)
- External DNS helper (in cluster)
- aksapp.io DNS Zone (in Azure)

```
--enable-addons
http_application_routing
```



The HTTP application routing add-on is designed to let you quickly create an ingress controller and access your applications. This add-on is not recommended for production use ⚠

‘Monitoring’ Add-On

Azure Monitor – Container Insights

docs.microsoft.com/azure/azure-monitor/insights/container-insights-overview

Visualization

Visualize overall health and performance from clusters to containers with drill downs and filters

Insights

Provide insights with multi-cluster health roll up view

Monitor & Analyze

Monitor and analyze Kubernetes and container deployment performance, events, health, and logs

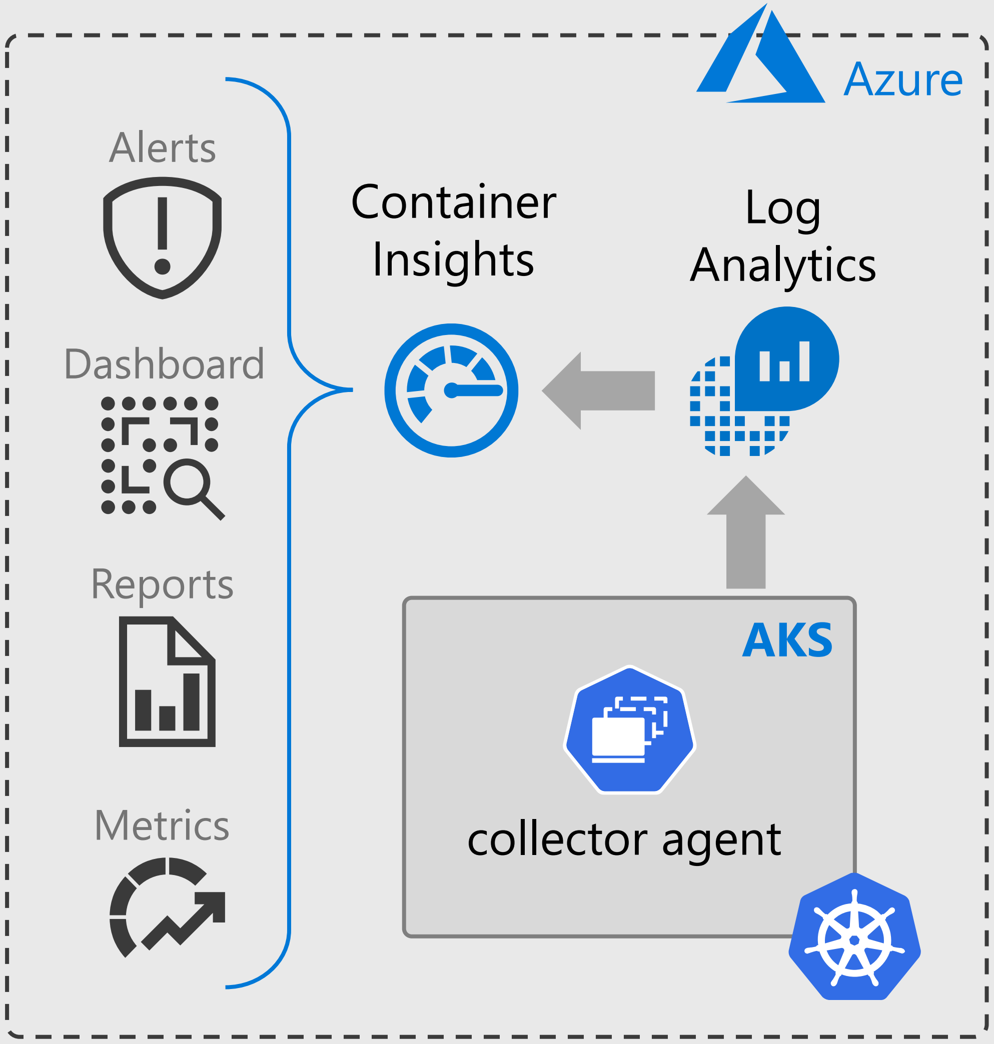
Response

Native alerting with integration to issue managements and ITSM tools

Observability

Observe live container logs on container deployment status

```
--enable-addons monitoring
--workspace-resource-id
  {my-azure-monitor-logs-ws}
```



Dapr

APIs for building portable and reliable microservices

dapr.io

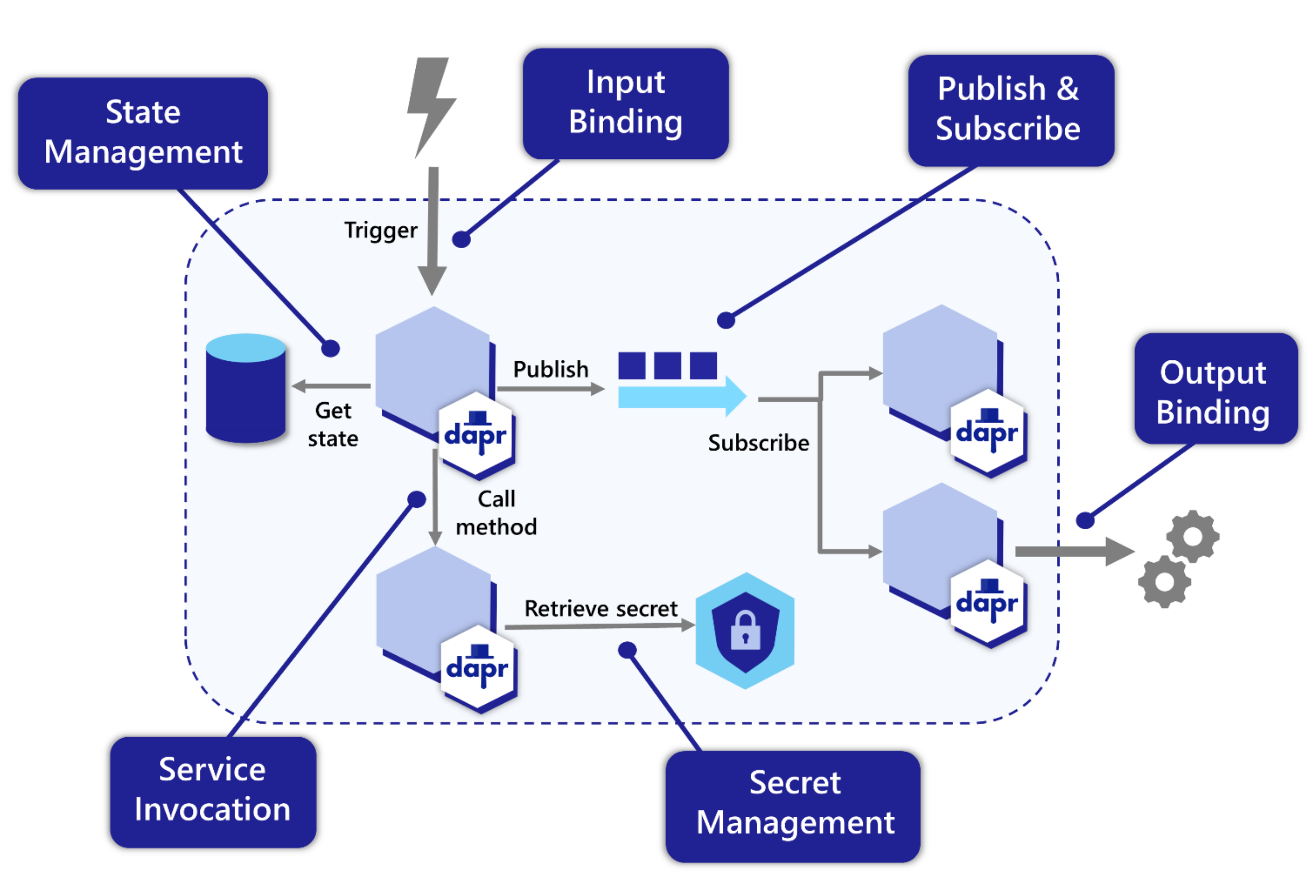
Dapr is a portable, event-driven runtime to simplify building microservices

A wide range of APIs and building blocks for common patterns such as pub sub, service invocation and storing state

Use triggers to build event driven applications

Out of the box components & connectors for many 3rd party systems

```
az k8s-extension create
--cluster-name my-cluster
--extension-type Microsoft.Dapr
```



https://docs.dapr.io/concepts/building-blocks-concept/

AKS Virtual Nodes

Elastically provision compute capacity in seconds

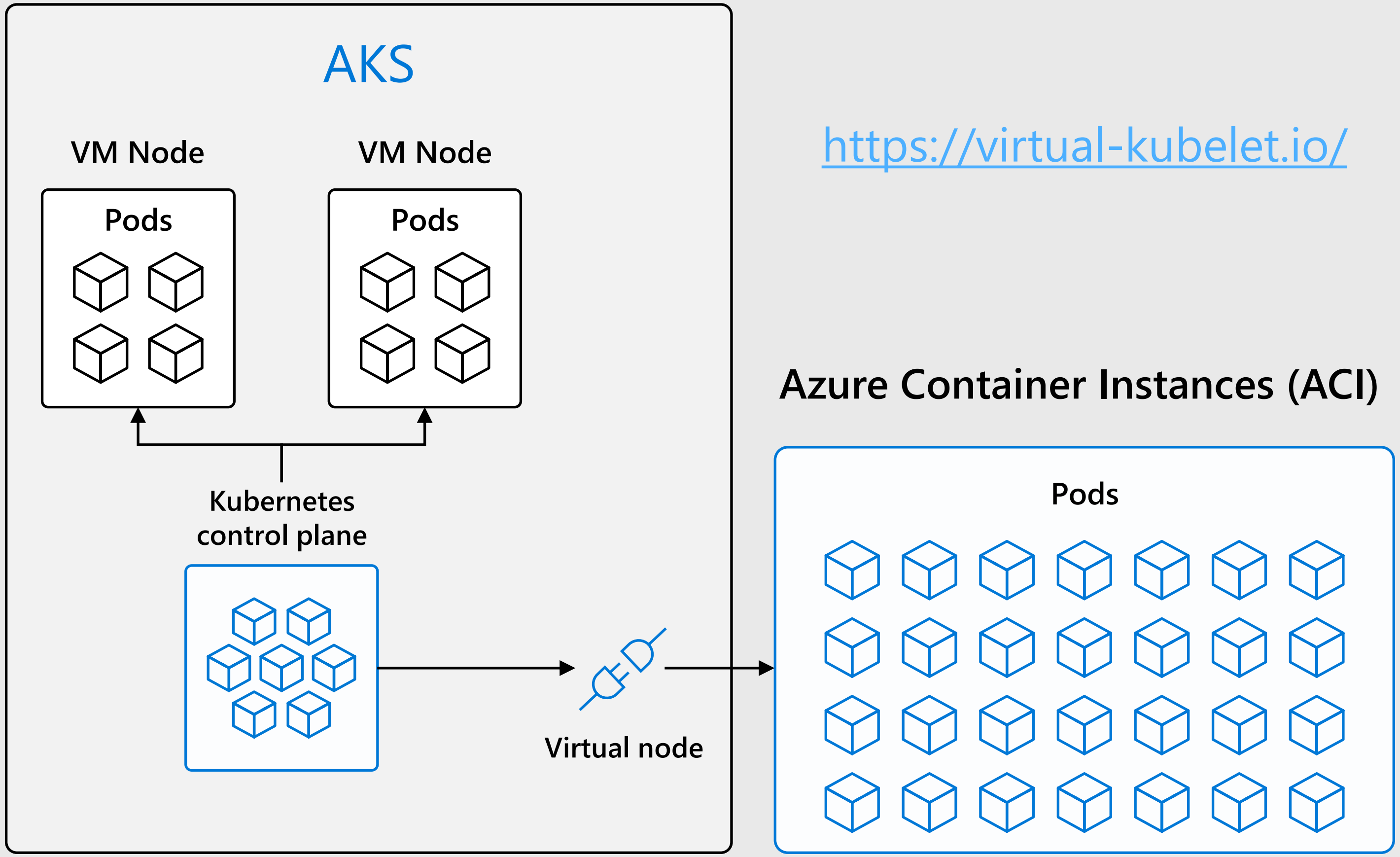
No infrastructure to manage

Builds on top of Azure Container Instances (ACI)

Uses Virtual Kubelet

Scenarios:

- Bursting
- CI/CD
- “Serverless Kubernetes”



https://virtual-kubelet.io/

Cluster Auto Scaler

Autoscaling with Azure Scale Sets

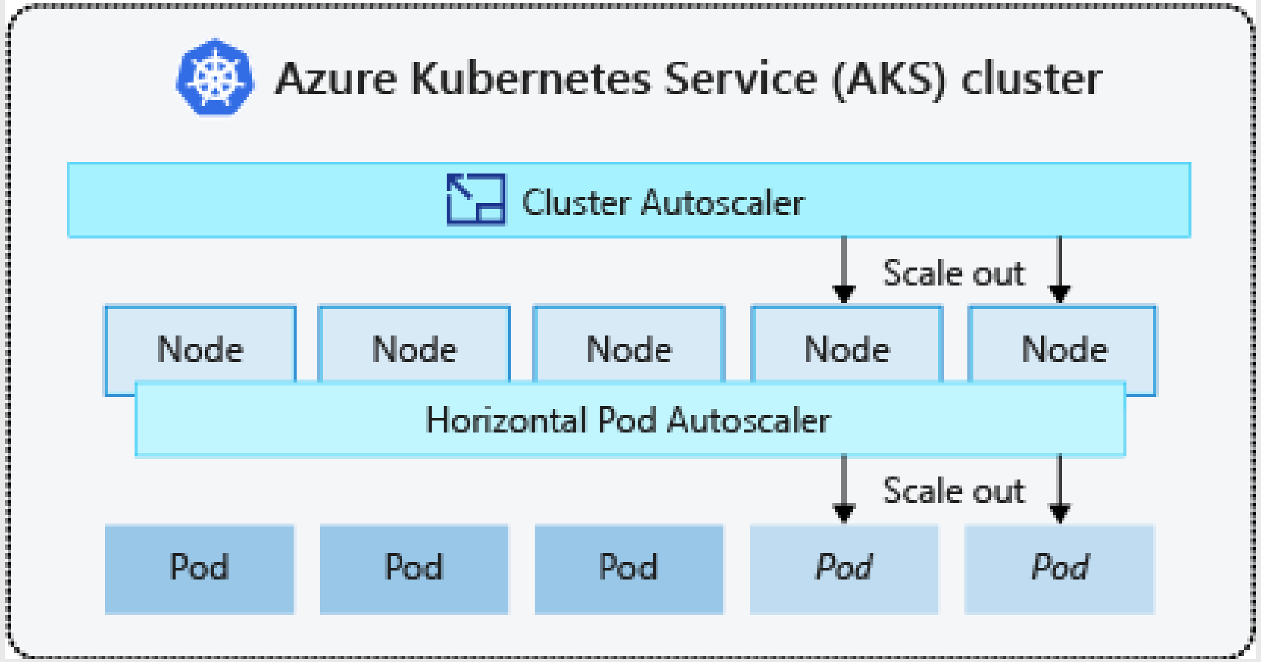
docs.microsoft.com/azure/aks/cluster-autoscaler

Enables the standard Kubernetes Cluster Autoscaler (CA)

Leverages VM Scale Sets

Settable per node pool if using multiple pools

```
--enable-cluster-autoscaler
--min-count 1
--max-count 8
```



Node Pool

Additional Resources

Kubernetes Learning Path

Kubernetes Docs

API Reference

Kubernetes Hands On Lab

Kubernetes Workshop

Katacoda Courses

Udemy Course

THANK YOU