Kubernetes Technical Primer

Ben Coleman

@BenCodeGeek

Feb 2020, v1.4

Content Notes

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

It can used as a learning guide or hand out

Alternatively take out 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

MAIN SECTIONS

Introduction To Kubernetes

Core Components

Using Kubernetes

Beyond The Basics

DevOps

Azure Kubernetes Service

Introduction To Kubernetes

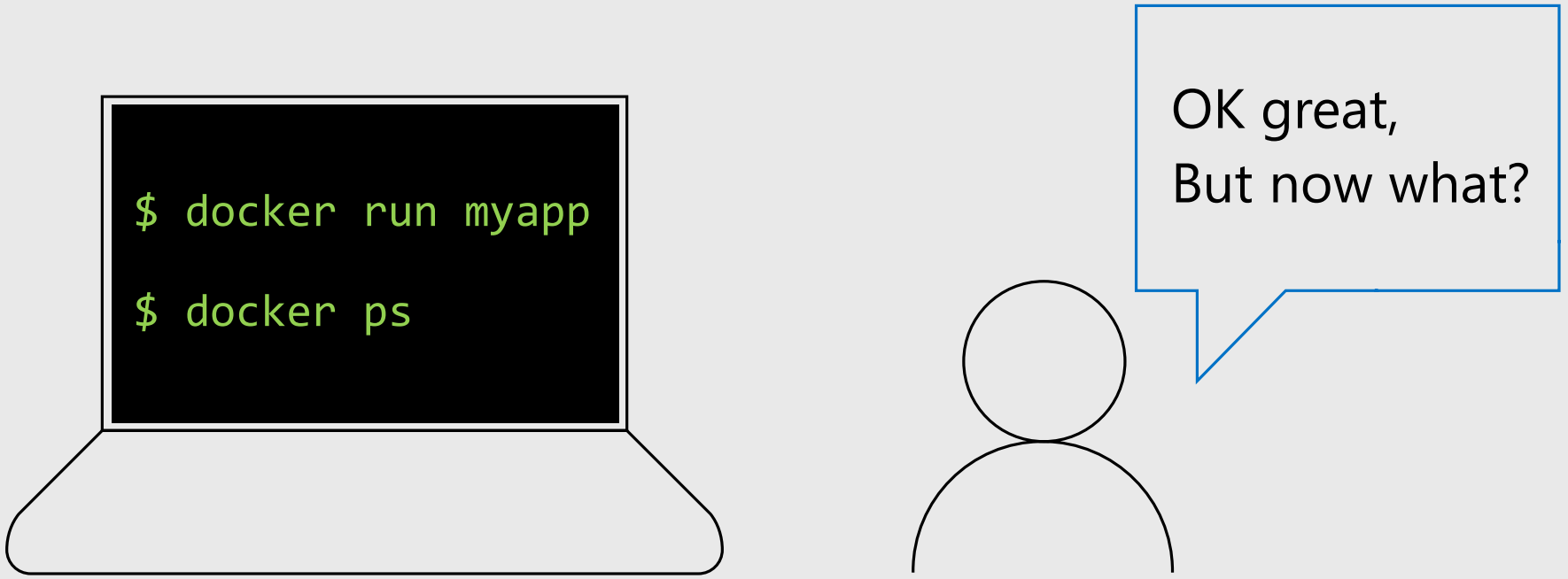
What is Cloud Native?

"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.

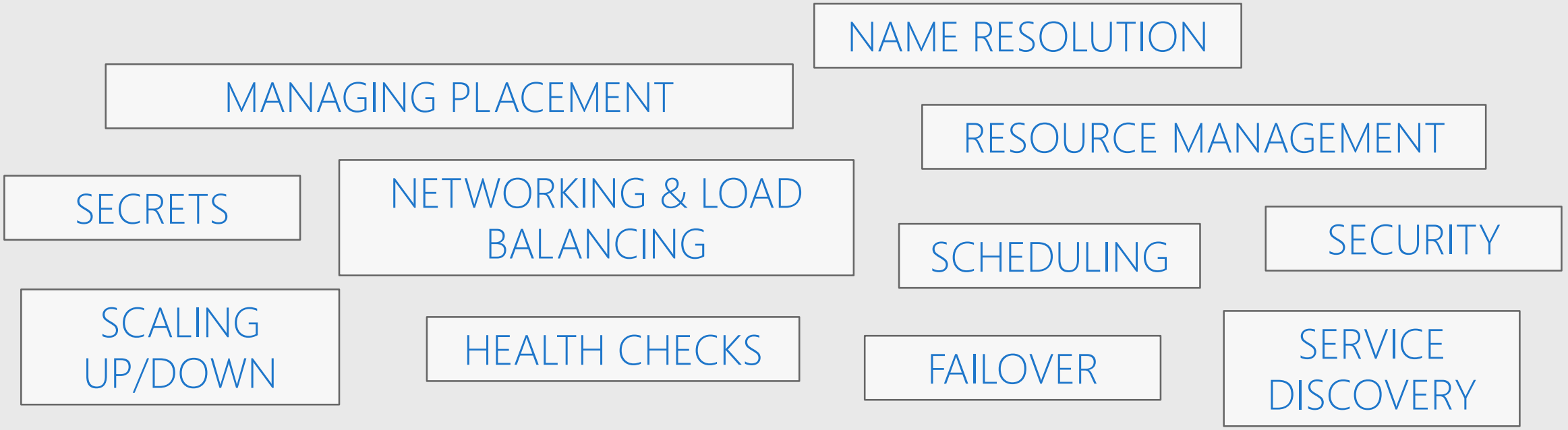
These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil."

CLOUD NATIVE
COMPUTING FOUNDATION

The need for orchestration

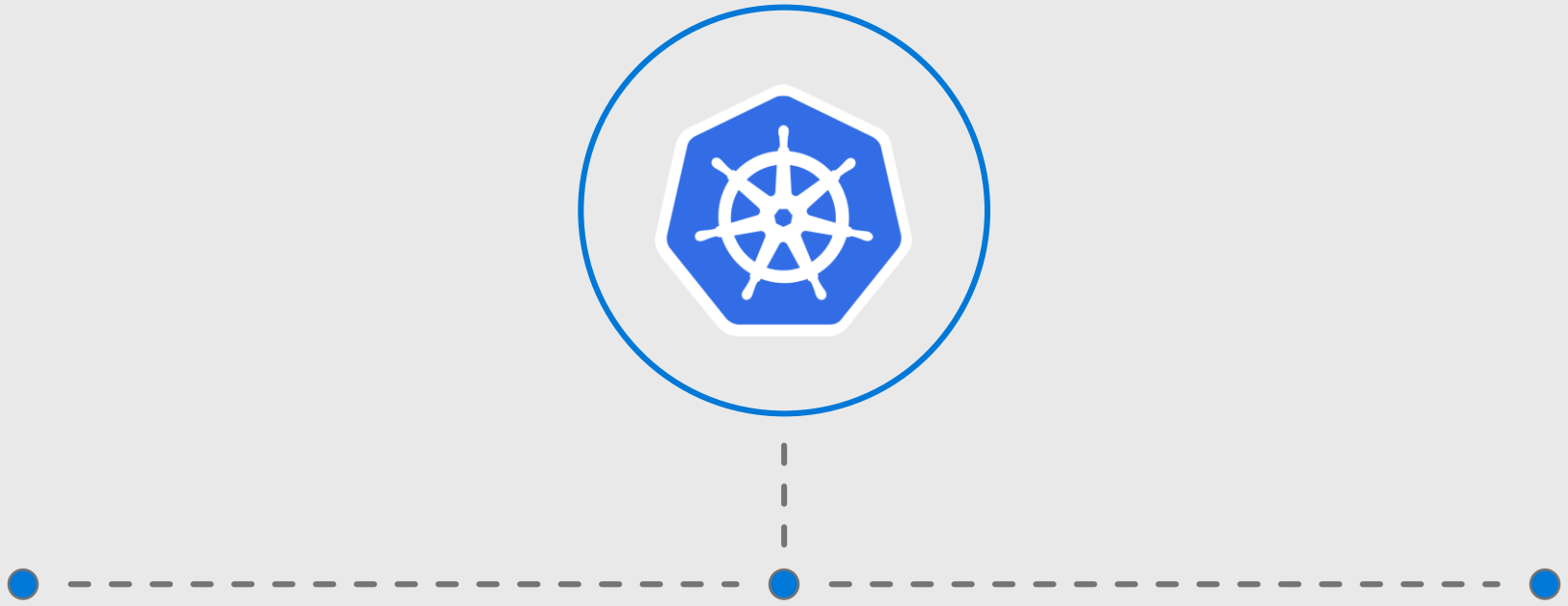


The need for orchestration



Production And At Scale

Kubernetes - production grade orchestration



Portable

Public, private, hybrid, multi-cloud

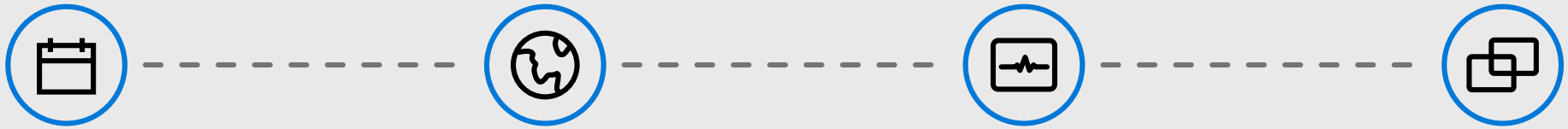
Extensible

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

Run anywhere

Industry adoption

Open Source with high degree of support

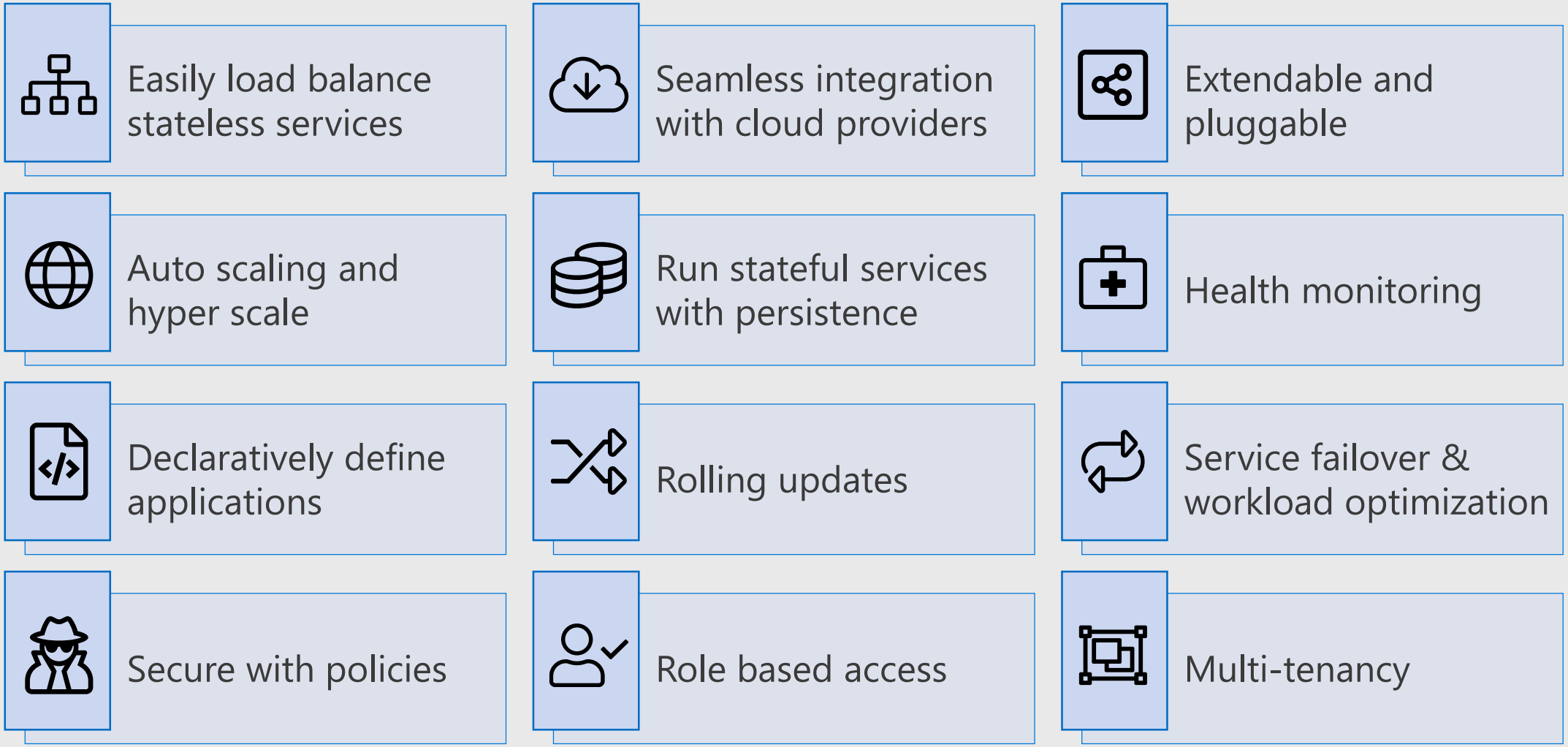
Avoids lock-in

Skills availability

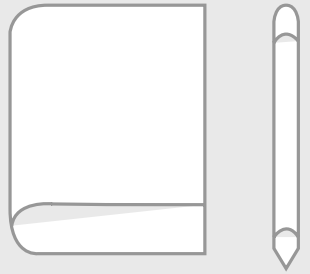
Large & growing ecosystem

Rise of microservices & containers

Kubernetes – A Modern Orchestrator



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)

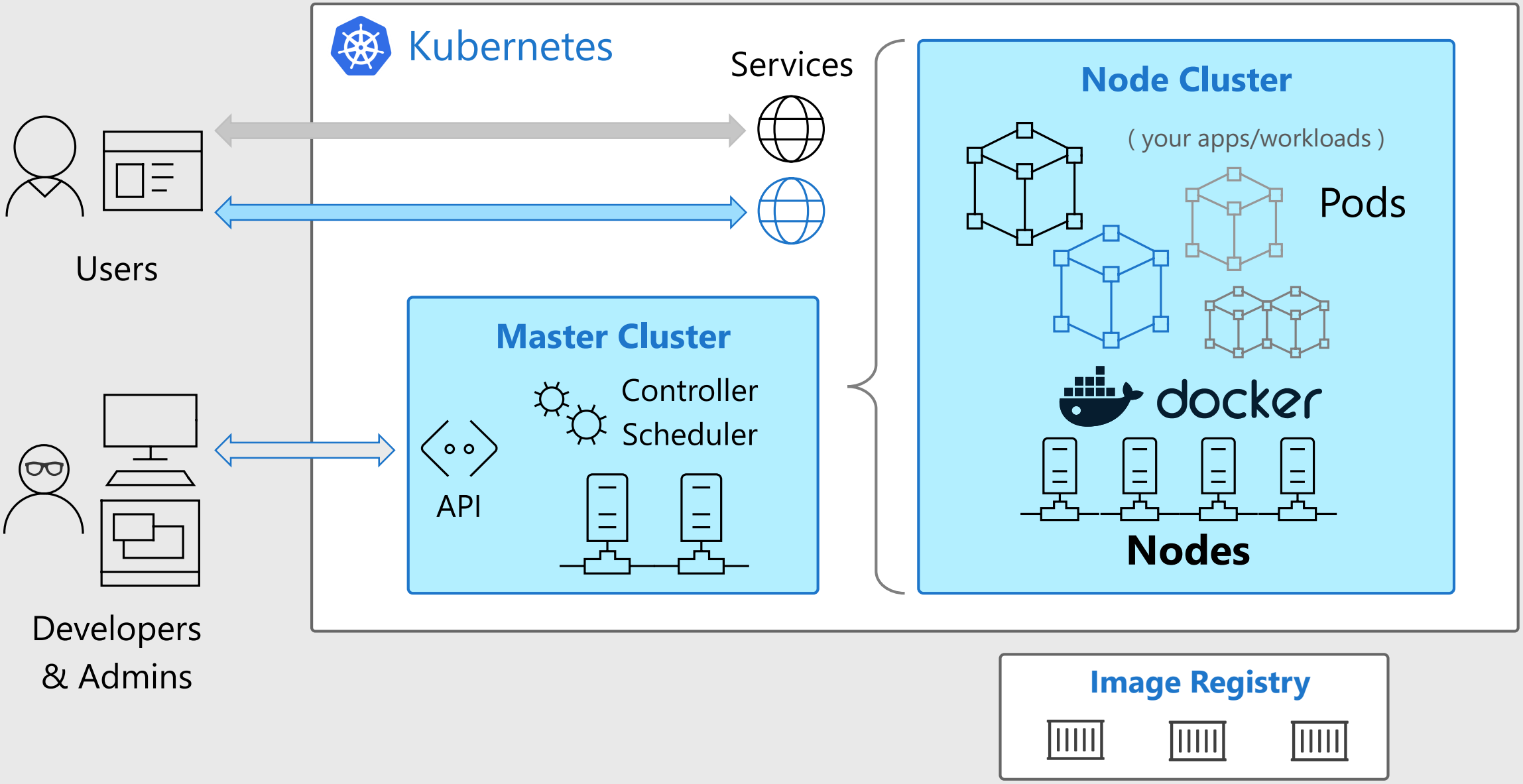
Label
Metadata attached to any object for configuration and selection

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

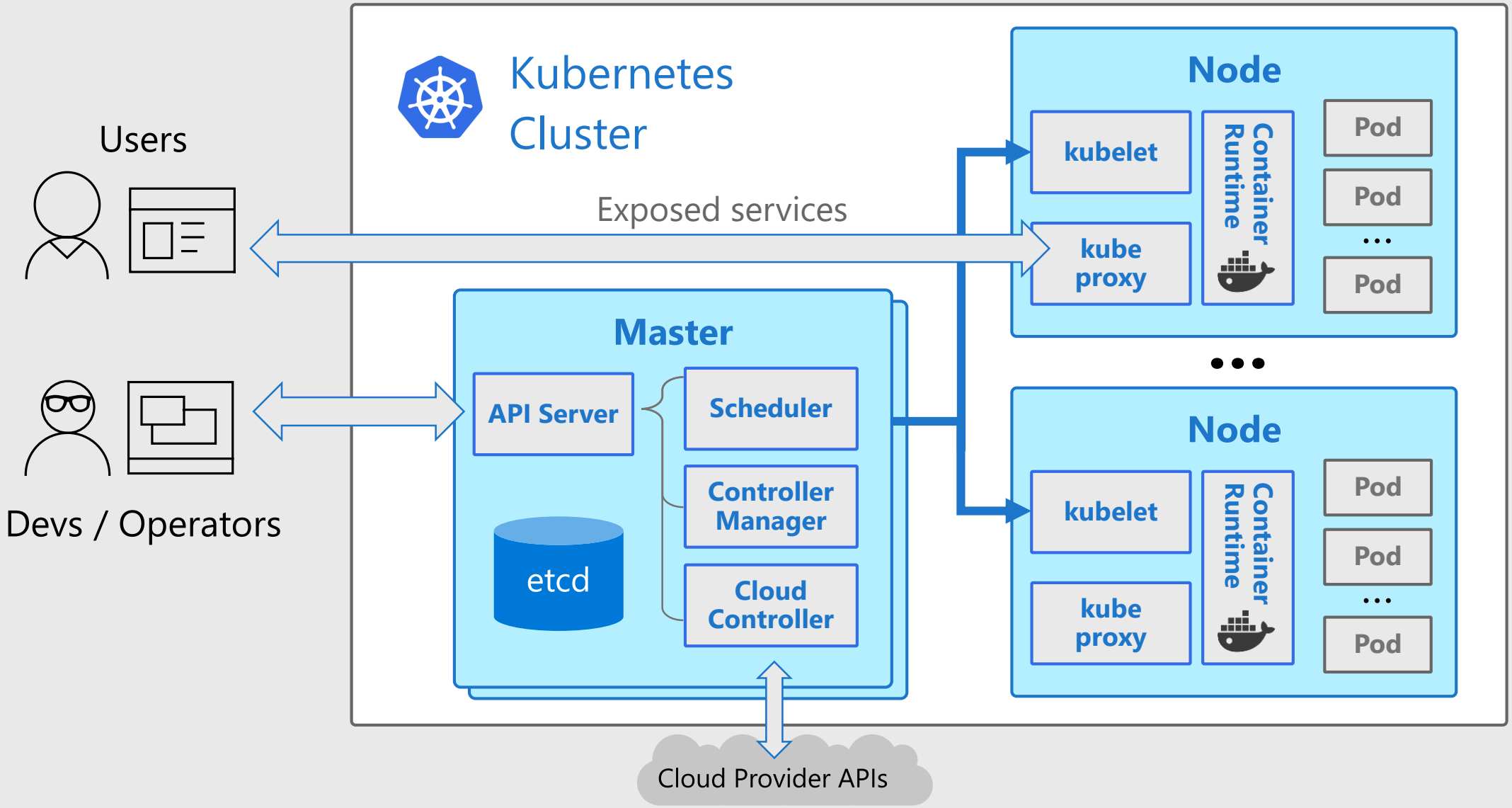
Service
Network access to a resource, e.g. pod or port. Typically load balanced

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

Simplified Architecture

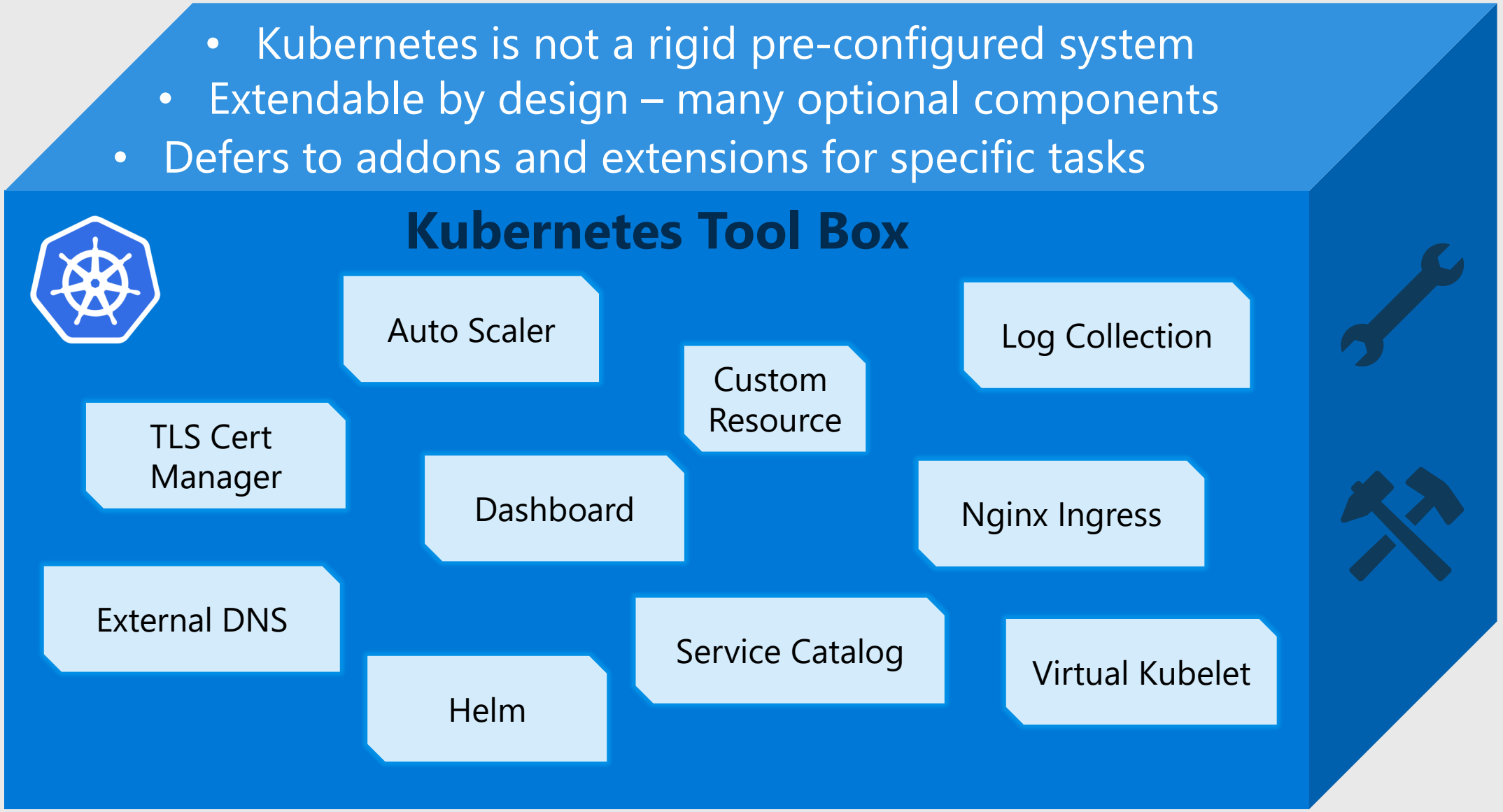


Internal Architecture



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



Core Components

Pods

Fundamental building block of Kubernetes

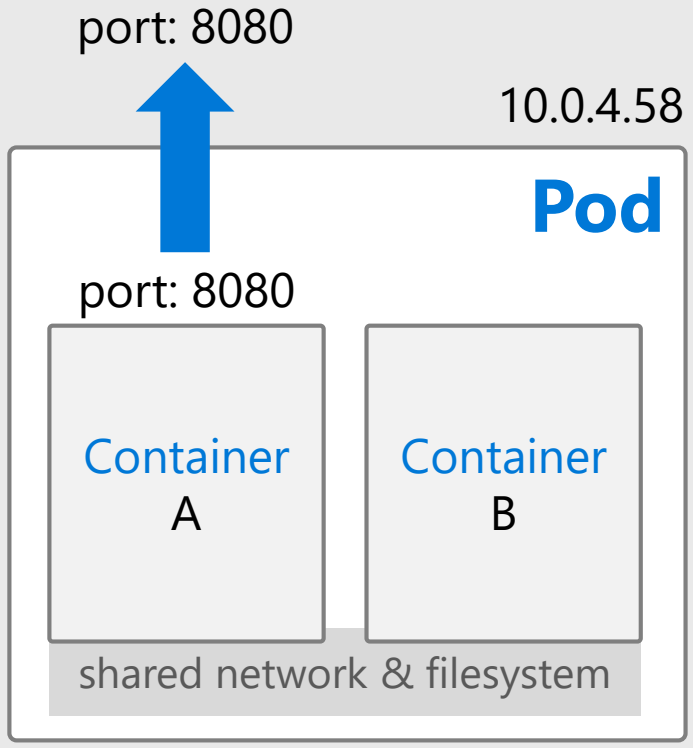
Pods run **one or more containers**

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

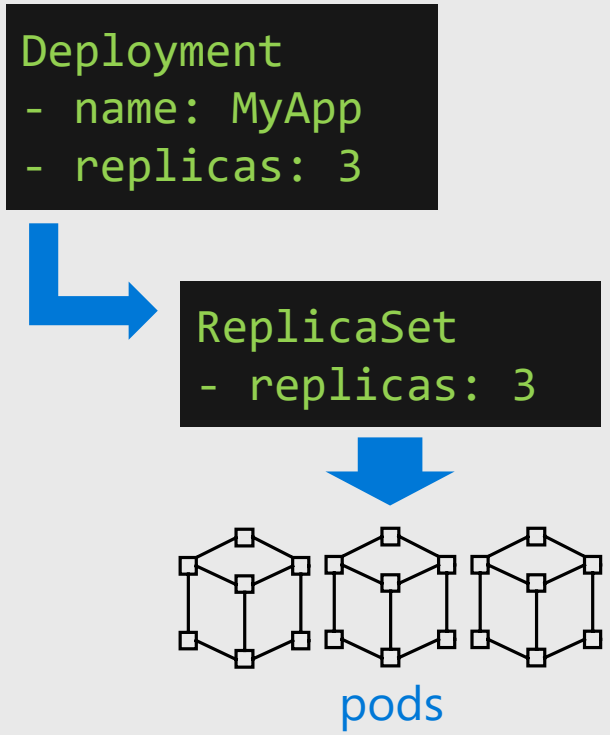
Deployments describe a **replicated** set of *Pods*

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*



Deployments let you run & scale stateless workloads in Kubernetes

Services

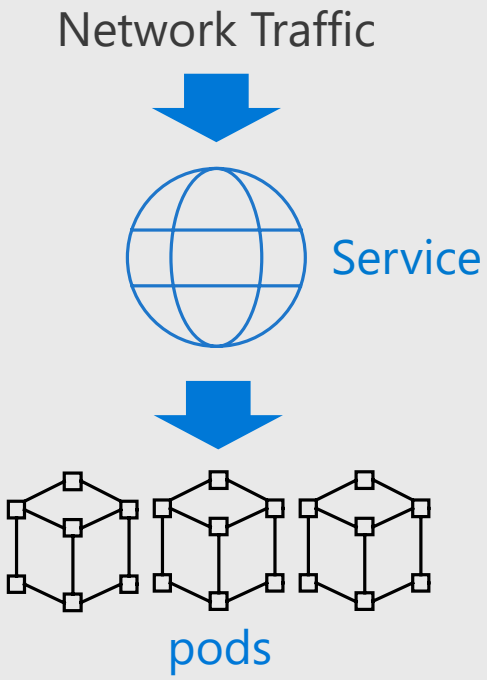
Network access to Pods

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*

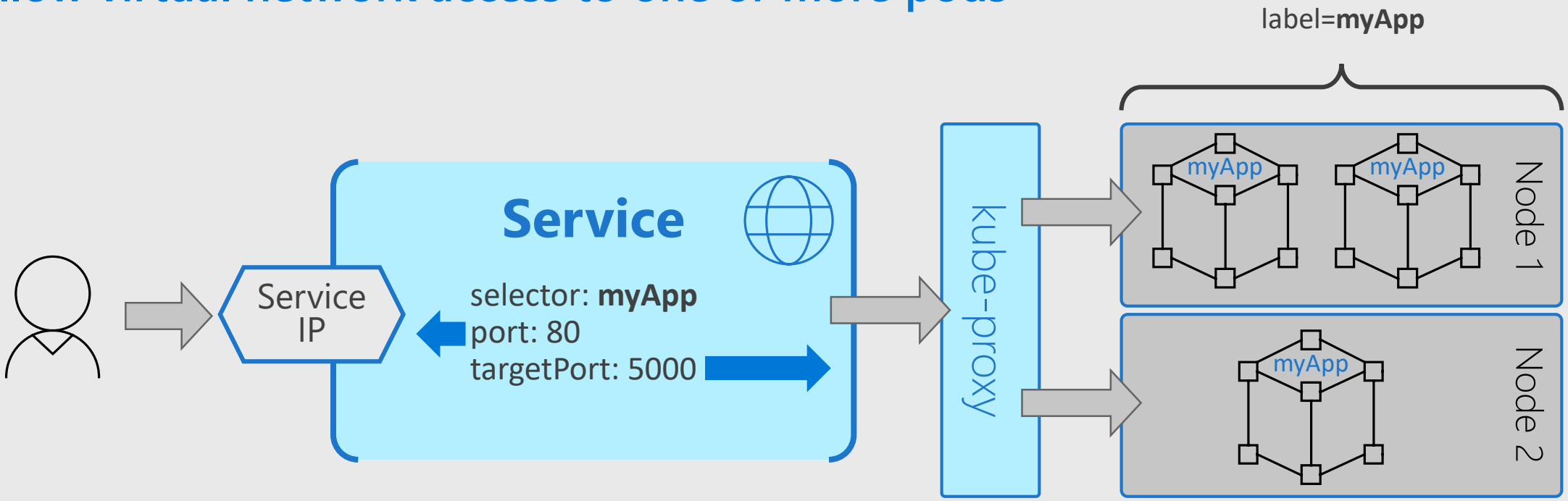
Services are assigned IP addresses and DNS names



Services are how you connect to Pods over the network

Services – Illustrated

Allow virtual network access to one or more pods



EXTERNAL

LoadBalancer

Uses cloud provider to present an external load-balanced IP

INTERNAL

ClusterIP

Internal virtual IP, only accessible by other pods/services

Note. Uses 'round robin' to select pods

DNS and Service Discovery

Naming for Pods and Services

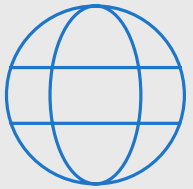
DNS in cluster is handled automatically by Kubernetes

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

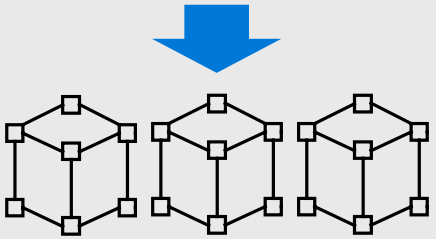
External public DNS can also be configured – See "Additional Network Services" section later

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



```
Service
- name: myapp
```

Service IP: 10.200.4.30



pods

Connect to your workloads using DNS and Services

Data Volumes & Mounts

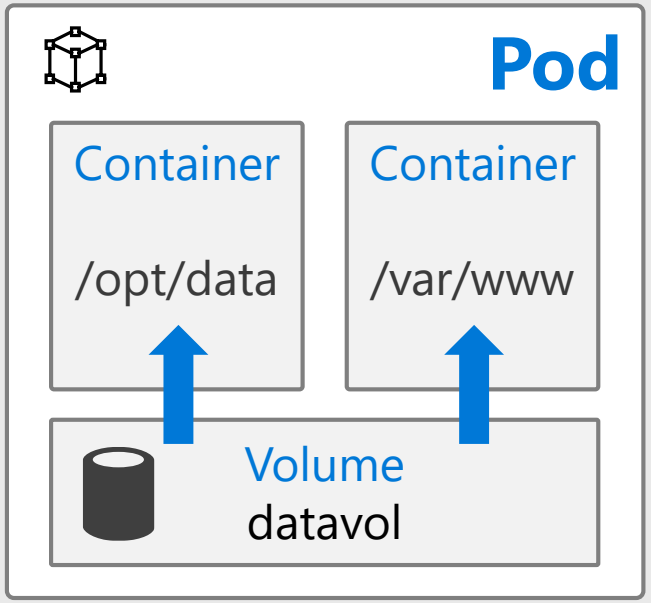
Handling State & Data

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

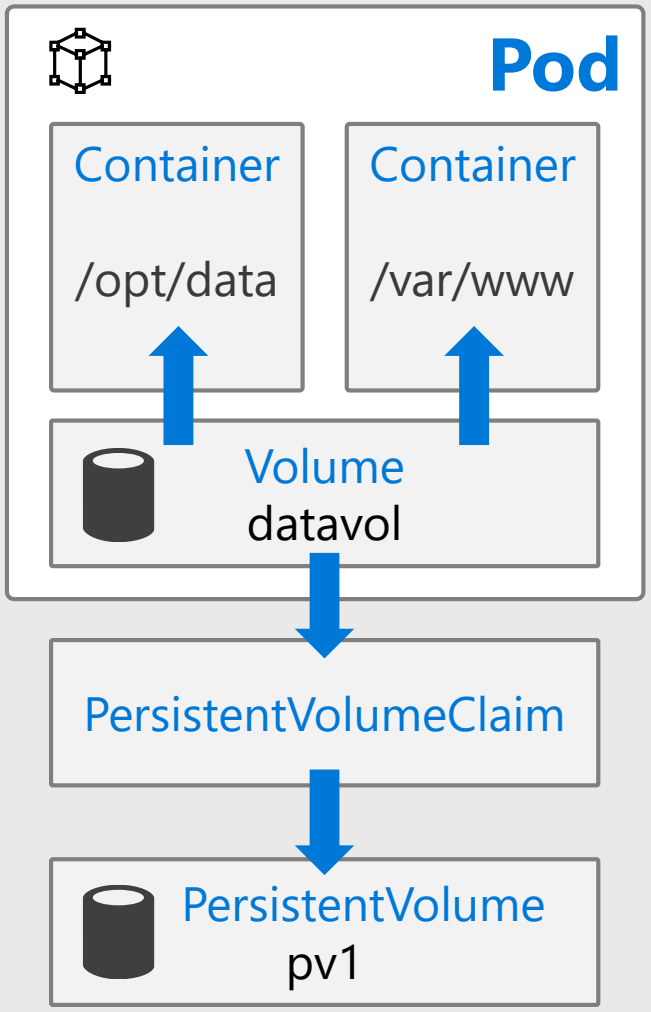
Handling State & Data

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), Flocker, Ceph, Gluster, AWS



Persistent Volumes retain data long term, outside of Pods

Stateful Sets

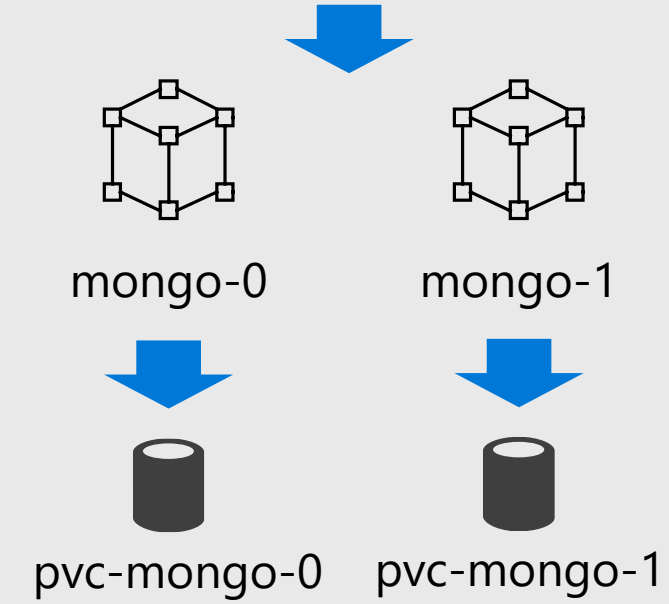
Handling Stateful workloads

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

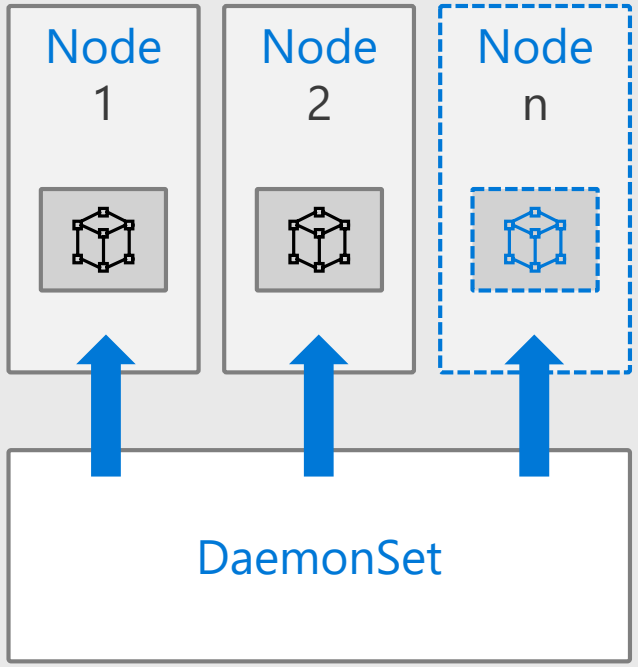
Daemon Sets

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

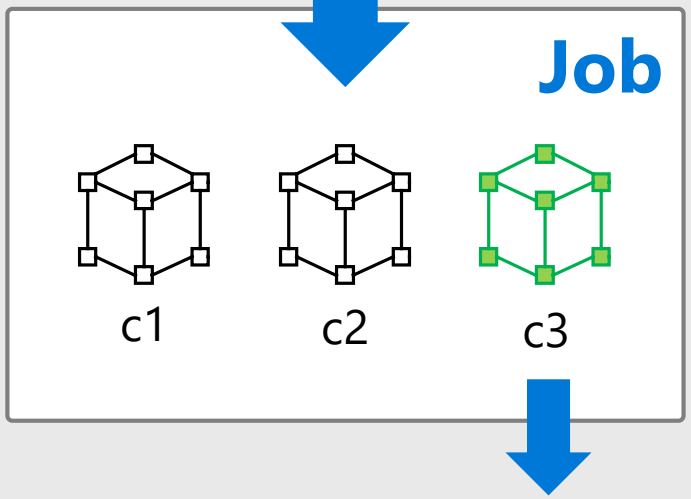
A *Job* creates one or more pods and **ensures that a specified number of them successfully terminate**

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

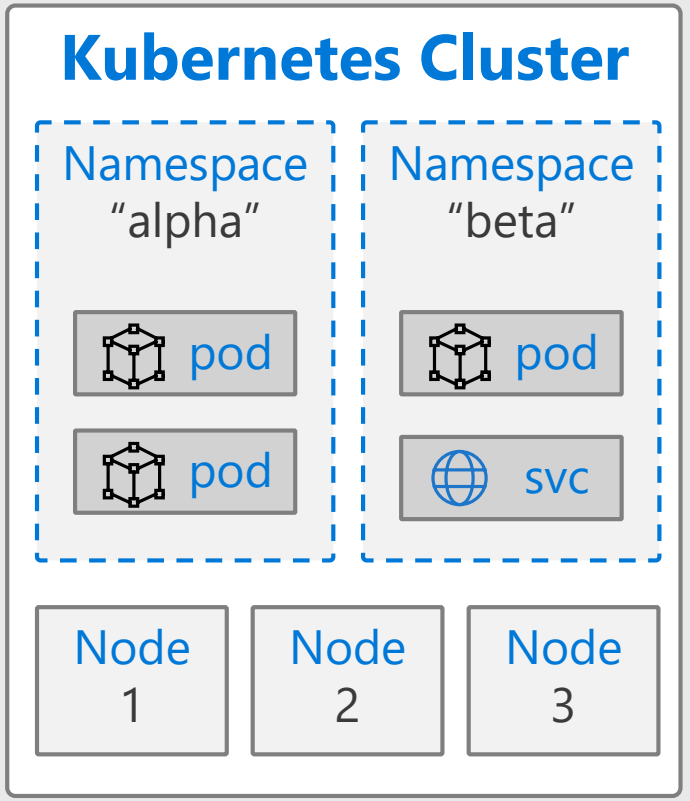
Multi tenancy

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*



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*

RoleBinding

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

Kubernetes has many authentication and authorization schemes. This complex area is deemed out of scope for this guide

Verb: **get**, **delete**, **create**, **list**, **update**, **watch**

API: **pods**, **secrets**, **services**, **jobs**, **nodes**, **ingresses**

RBAC is optional, but is becoming standard for any cluster

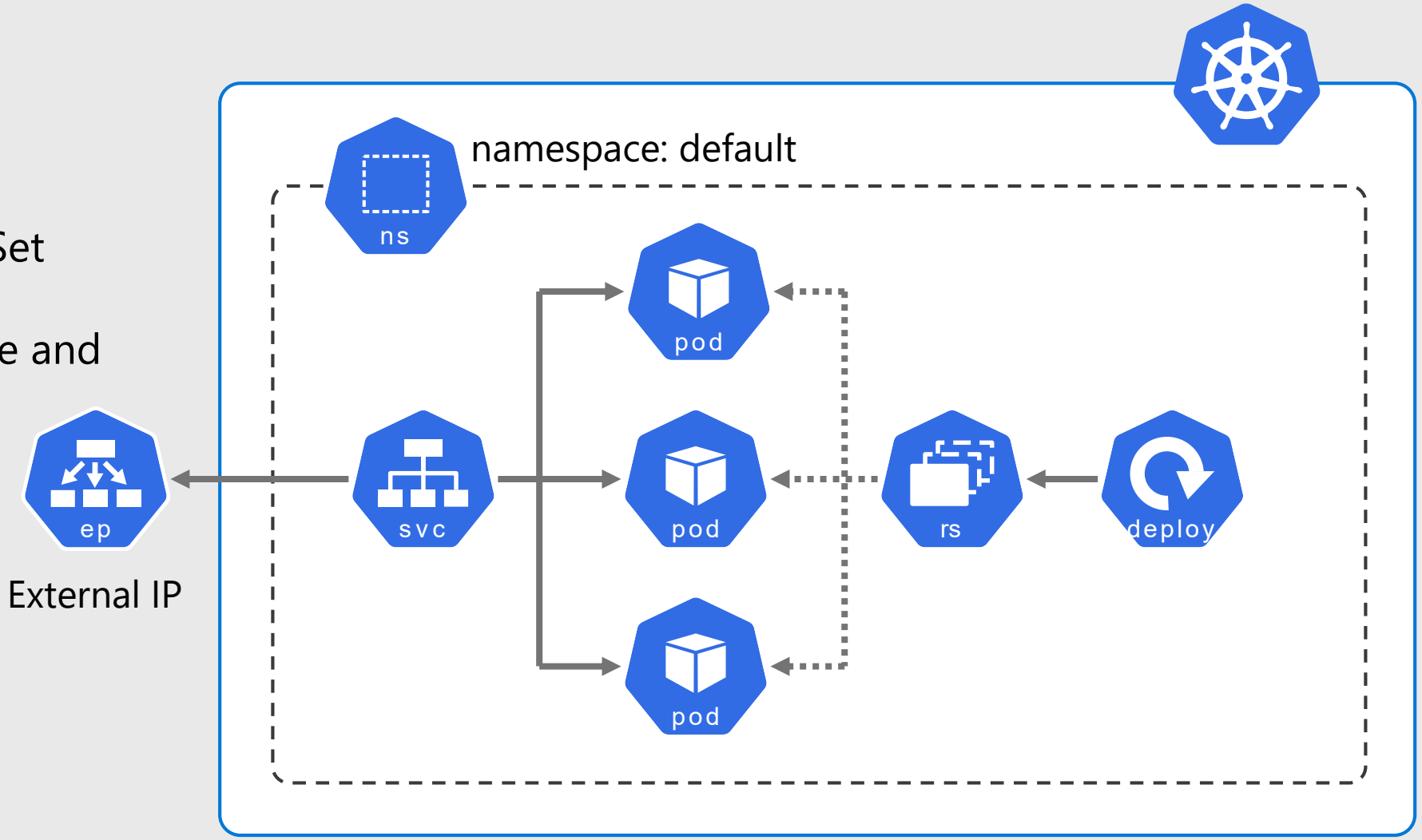
Putting It All Together

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

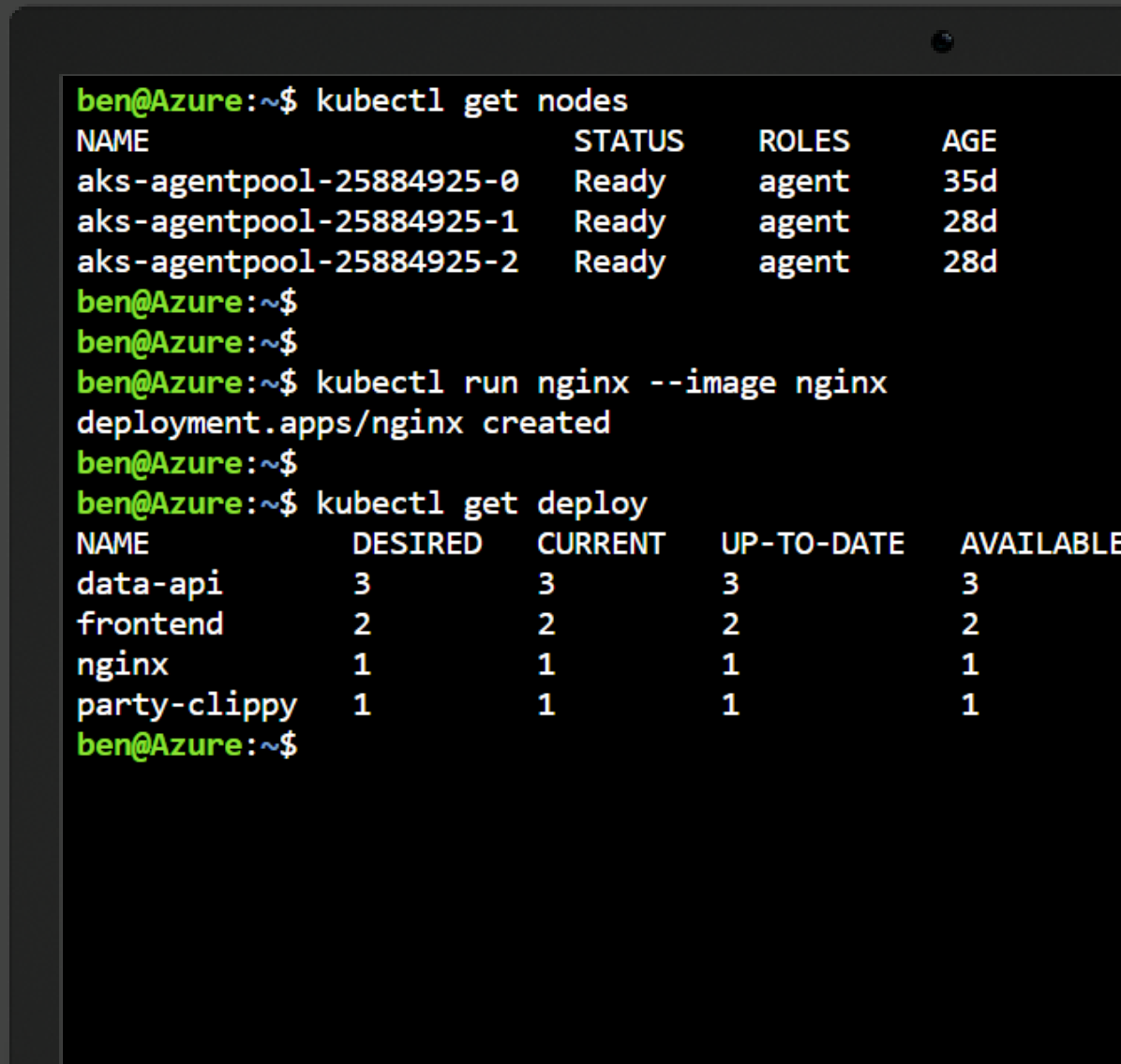
Main management & control interface to Kubernetes

Single executable binary

Can manage several clusters

Secure interface to Kubernetes API

Used for cluster ops and application deployment & management



Kubectl – Common Commands

get	delete	apply	run	describe
Display one or many resources	Delete resources	Create resources from YAML	Directly start and run pods	Get details of any resource

`get nodes`
 	
`get pods`
 	
`get all`
 	
`get pod/myPod`
 	
`get nodes -w`
	`delete deploy/mydeploy`
	
	`delete -f myapp.yaml`
	
	`delete -l app=foo`
		`apply -f myapp.yaml`	`run nginx --image=nginx --replicas=3`
			
			`run myapp --image=foo/img --port=8080`
				`describe pod/pod138`
				
				`describe svc/myservice`
				
				`describe pod -l app=myapp`

kubernetes.io/docs/reference/kubectl/cheatsheet

kubernetes.io/docs/reference/kubectl/overview

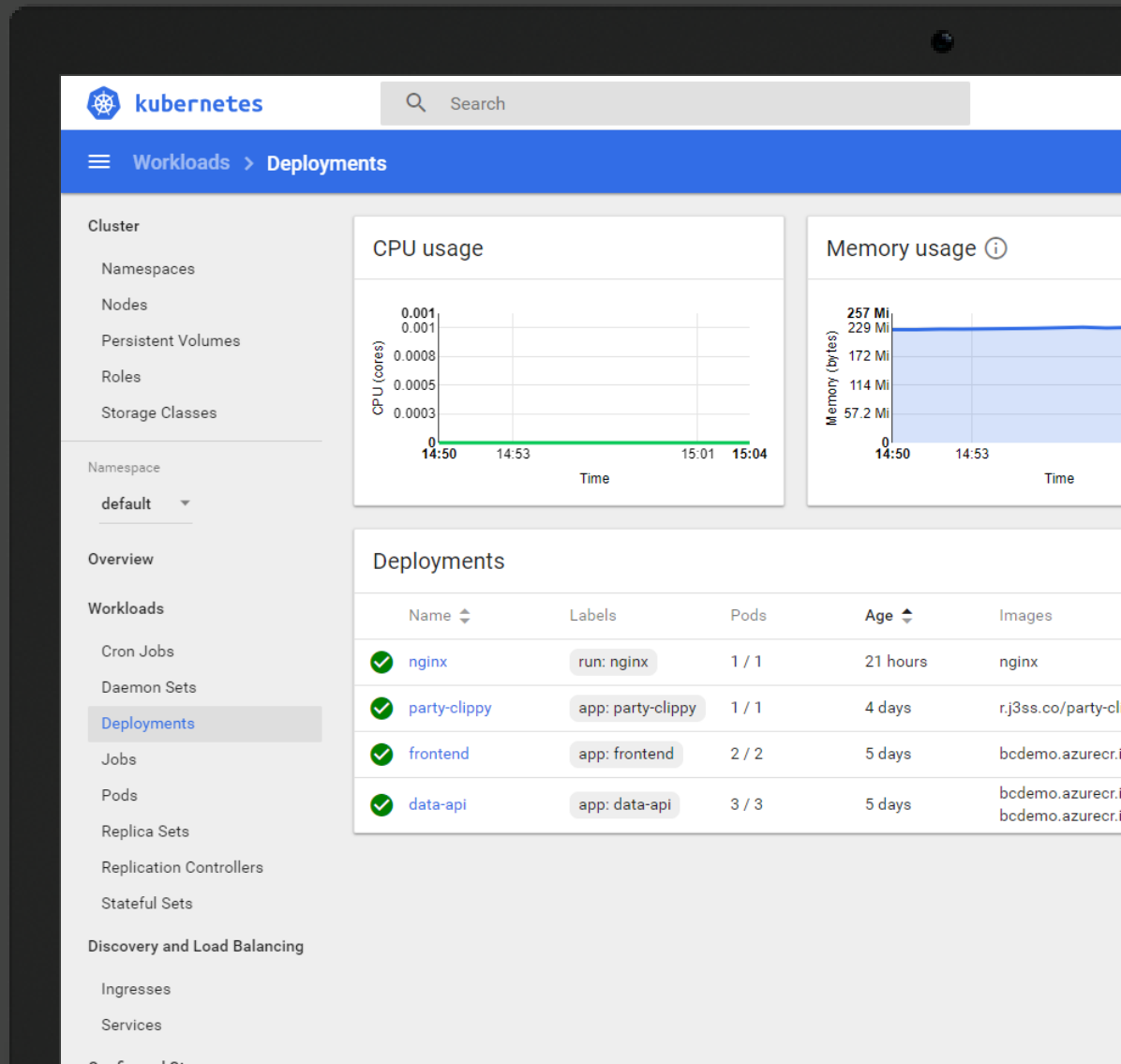
Dashboard

Management web UI

Runs inside Kubernetes as a pod

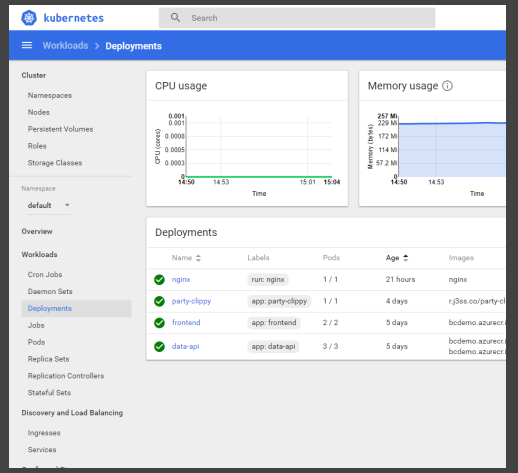
Not secure, should never be exposed externally

Optional, not required to run Kubernetes



Accessing the Dashboard

- Open proxy to Kubernetes API server
- Run command: `kubectl proxy`
- Access this URL
  http://localhost:8001/api/v1/namespaces/kube-system/services/kubernetes-dashboard/proxy



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 ---)

```
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

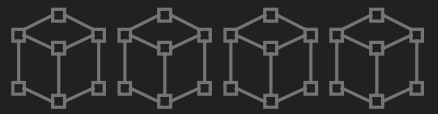
Files can describe desired state of the objects you configuring

`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
```

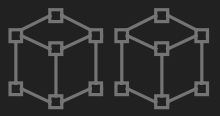
myapp.yaml

```
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 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

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='blahblah_this_is_secret'

secret/my-secret created
```

myapp.yaml

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

ConfigMaps

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

Uses
- Application configuration

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

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

configmap/my-config created
```

myapp.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

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: "64M"
        cpu: "0.25"
      limits:
        memory: "512M"
        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 accepting traffic

Liveness probe failure can restart the container

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

```yaml
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

Service Mesh

Facilitates services to service calls inside Kubernetes

Features:

- Observability / tracing
- Error handling / retries / backoff
- Encryption / mTLS
- Routing / balancing

Typically runs as a proxy sidecar in all your pods

Adds complexity & overhead

Read more
https://servicemesh.io/

Kubernetes Ecosystem – *Many* Projects

Security / Policy

Gatekeeper

DevOps

Helm

Flux

Serverless

Kubeless

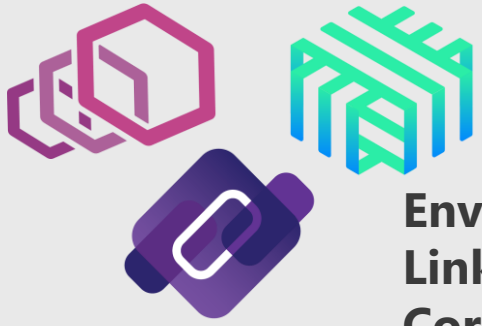
Monitoring

Prometheus
Grafana
Opentracing

Network Plugins

Calico
Weave Net
Flannel

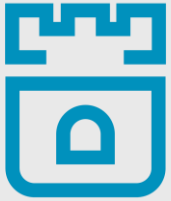
Load Balancing & Discovery



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

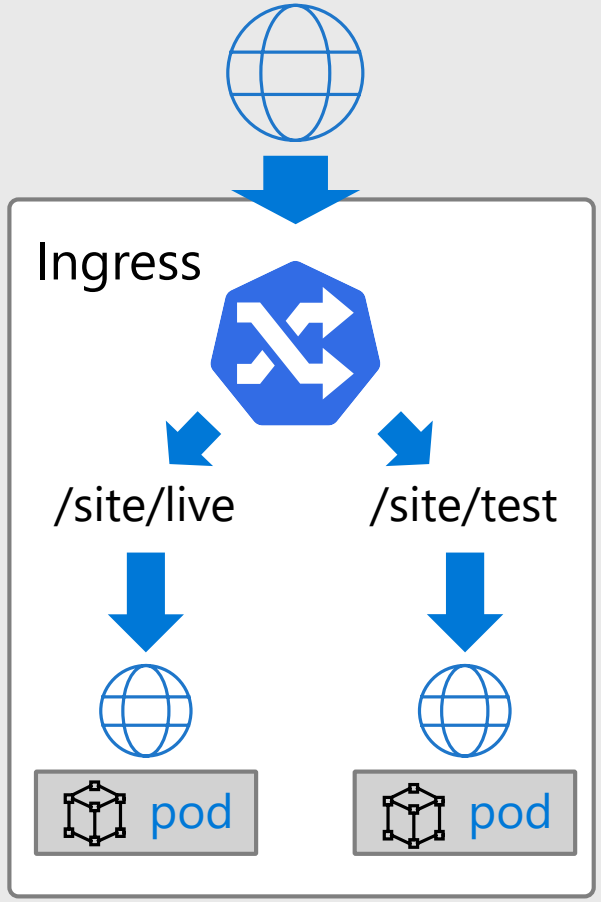
External access for HTTP and HTTPS

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

Various controllers exist, NGINX is commonly used



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

External DNS

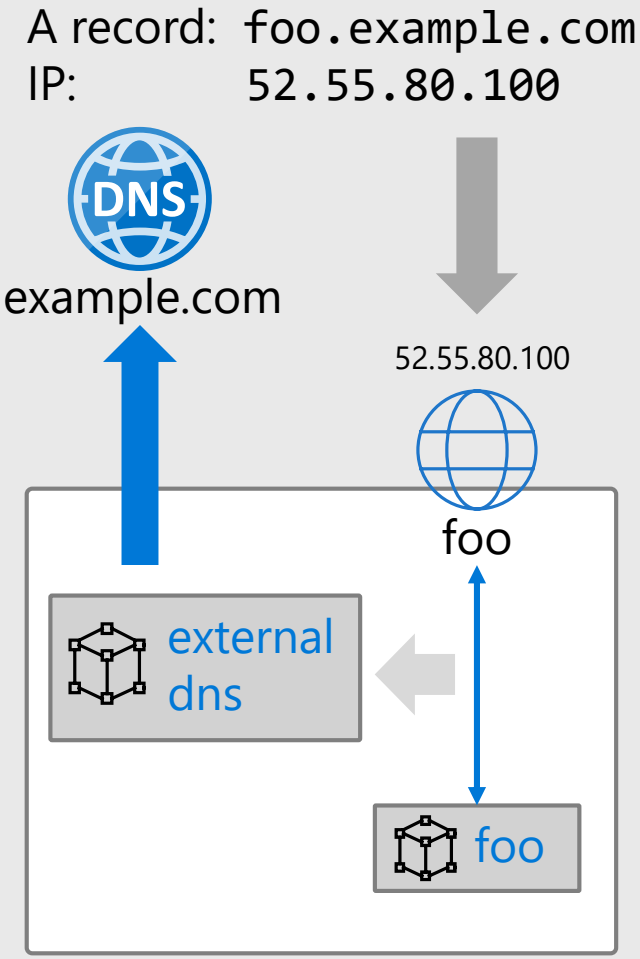
Optional Addon – Auto configuration of public DNS

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

Incubation project:
github.com/kubernetes-incubator/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

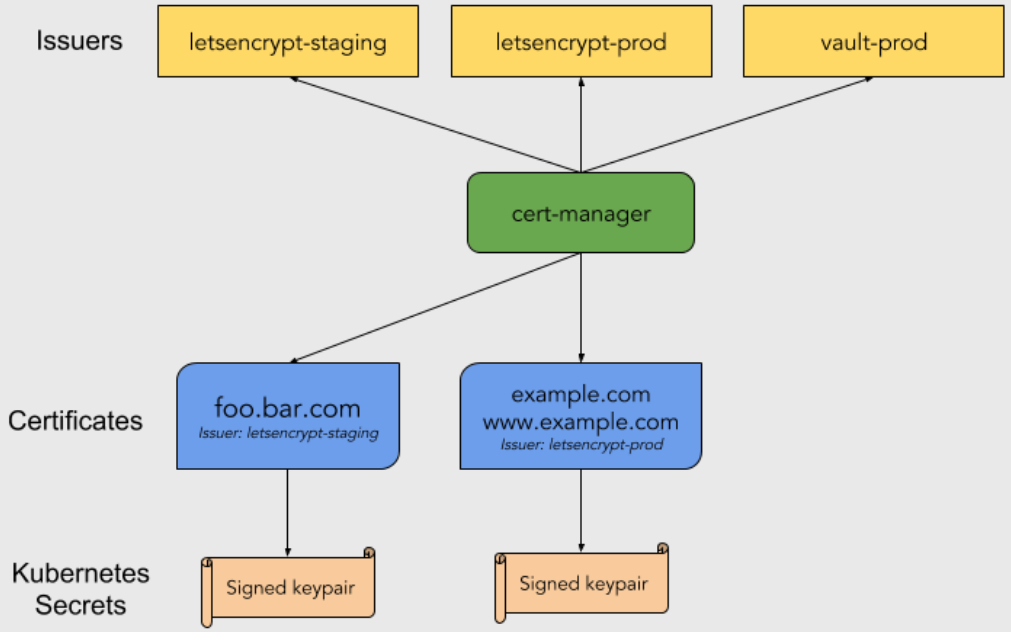
Ensures **certificates are valid** and up to date

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

Renew certificates before expiry

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

Project:
github.com/jetstack/cert-manager



Issue TLS certs for HTTPS access to services & Ingress

Putting It All Together

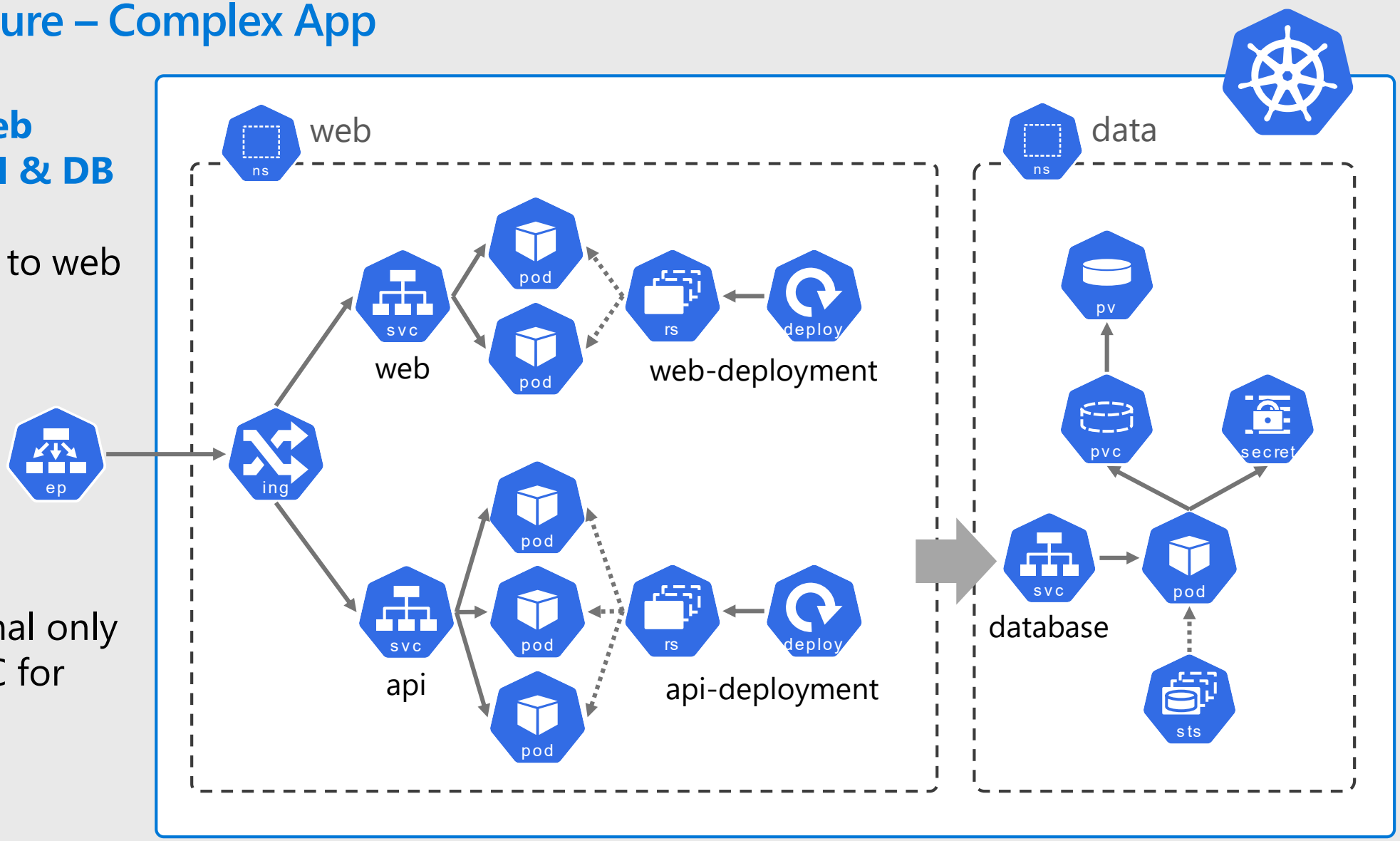
Sample Architecture – Complex App

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
```

```
    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`

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

```
$ 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
```

Get Shell Access

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

Create interactive shell into running containers

Use `kubectl exec`

Use `-it` switch and sh or bash for shell

Can also run single non-interactive command

Uses
- Low level debugging - Interactive troubleshooting

```
$ 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
```

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 ??

```
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

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

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

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

```
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

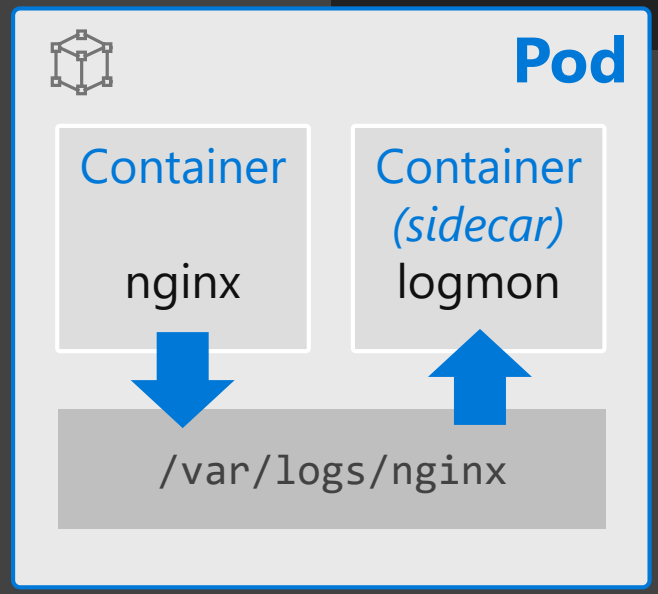
Pods co-locate multiple containers together, sharing network and storage

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)

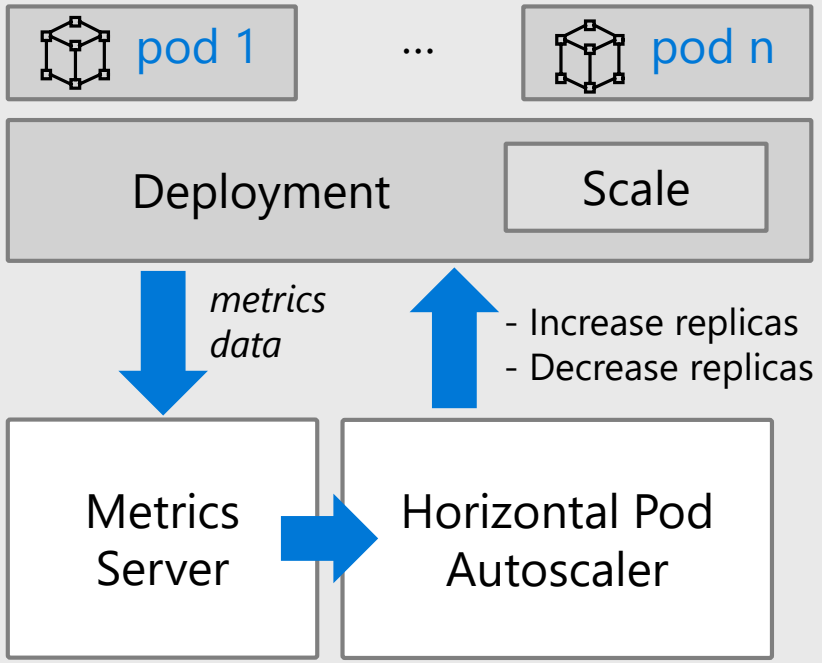
Automatically scale stateless workloads

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

Rules define desired *Pod* replicas based on **observed metrics**

Takes metrics from the metrics API fed from the *Metrics Server* (Kubernetes 1.8+)

Supports extension via custom metrics



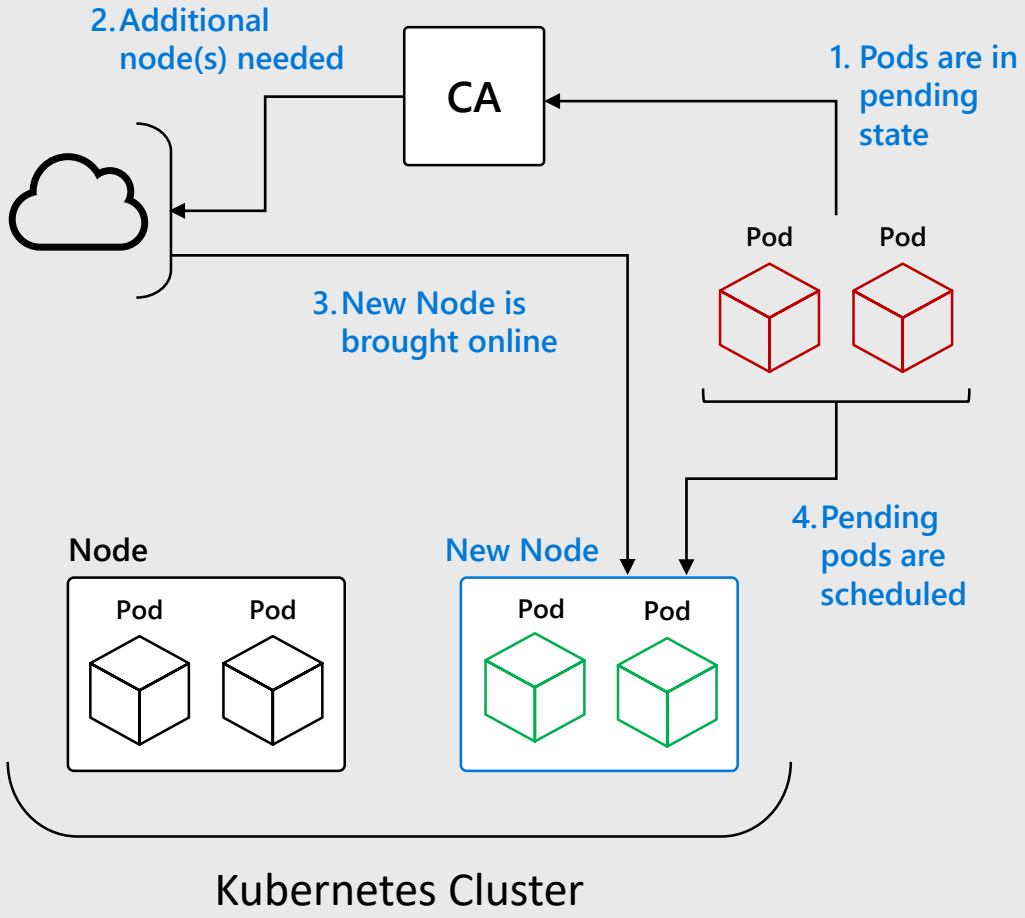
Dynamically scale stateless workloads across available nodes

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



Scale cluster wide by adding/removing Nodes

DevOps

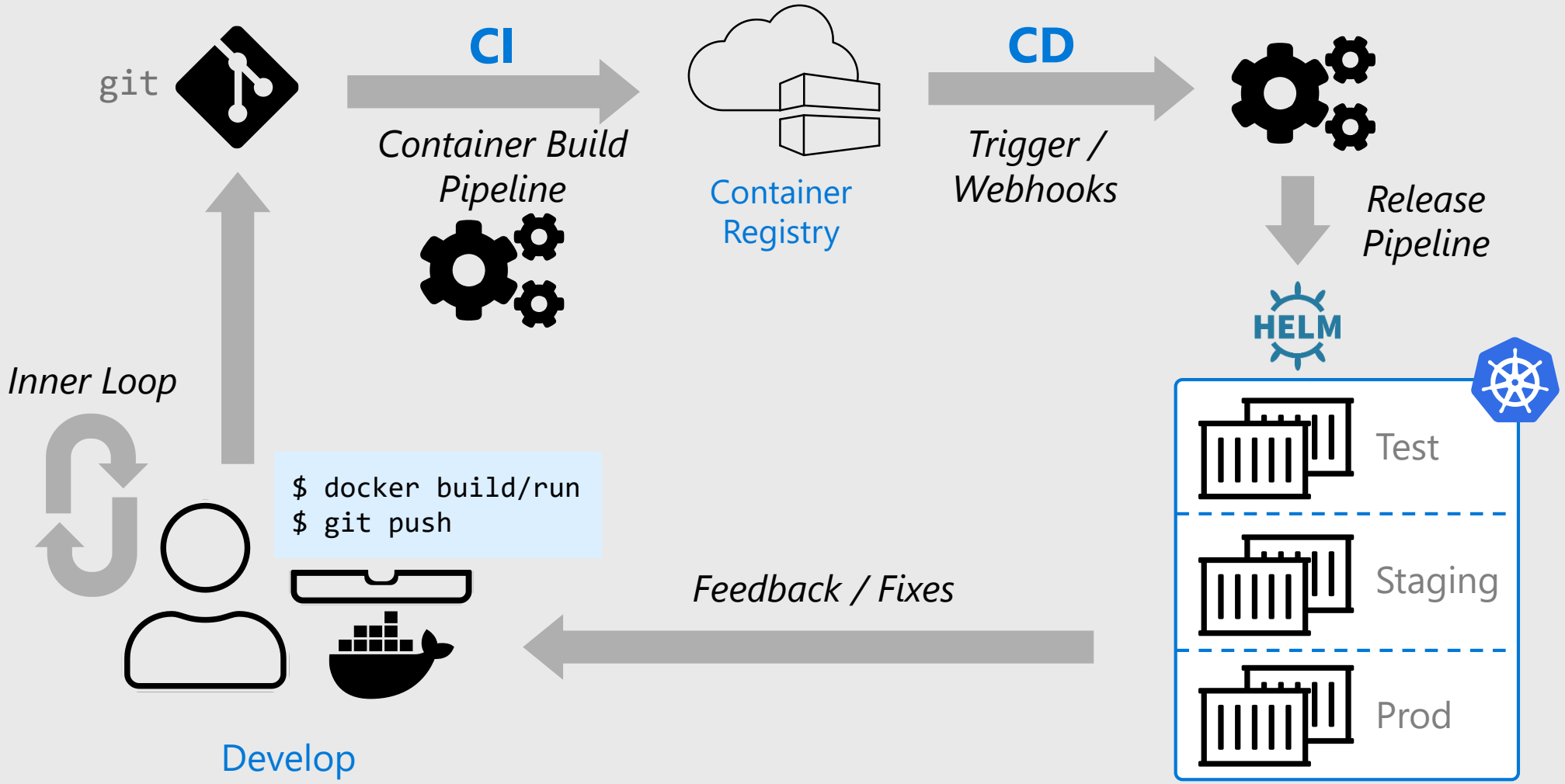
Virtual Machines

Pets Legacy Infrastructure 	**Cattle** Modern Infrastructure 
• Pets are given names like cutecat.petshop.com • They are unique, lovingly raised & cared for • When they get ill, you nurse them back to health	• Cattle are given numbers like 10200713.cattlerancher.com • They are almost identical to other cattle • When they get ill, you replace them and get another
Infrastructure is a **permanent** fixture in the data centre	Infrastructure is stateless, **ephemeral**, and **transient**
Infrastructure takes days to create, are **serviced weekly, maintained** for years, and requires migration project to move	Infrastructure is instantiated, modified, destroyed and recreated in minutes **from scratch** using **automated** scripts
Infrastructure is modified & patched in place and generally requires **specialist config management tools**	Infrastructure is **IMMUTABLE** Modify via **source control** & **automation pipelines** to rebuild and redeploy
Infrastructure requires **several different teams** to coordinate and provision the full environment	Infrastructure is **self-service** with the ability to provision computing, network and storage services with a single click
Infrastructure is **static**, requiring excess capacity to be dormant for use during peak periods of demands	Infrastructure is **elastic and scales automatically**, expanding and contracting on-demand to service peak usage periods

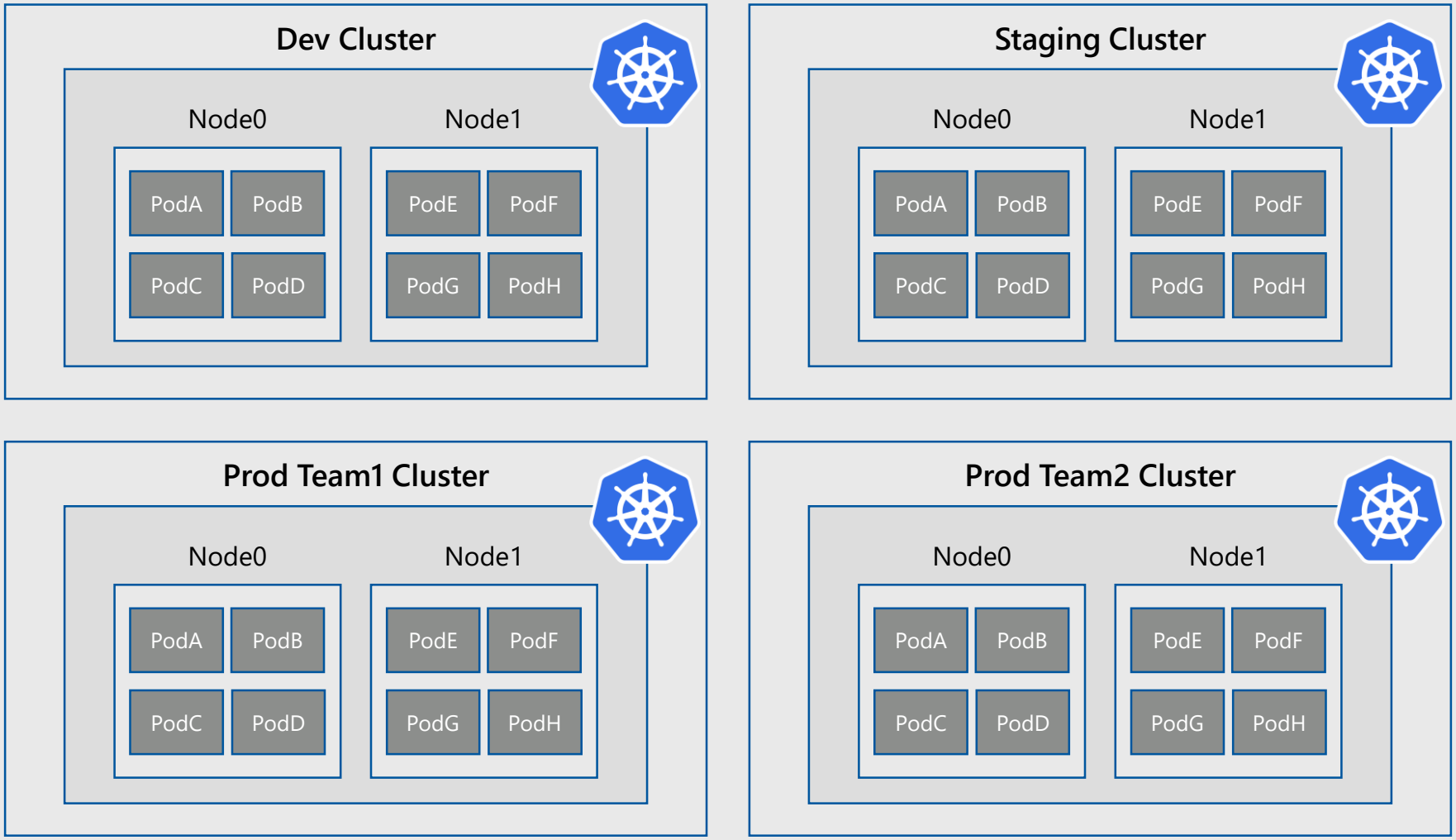
Containers

DevOps Containers Lifecycle Loop

Complete CI/CD Pipeline Loop

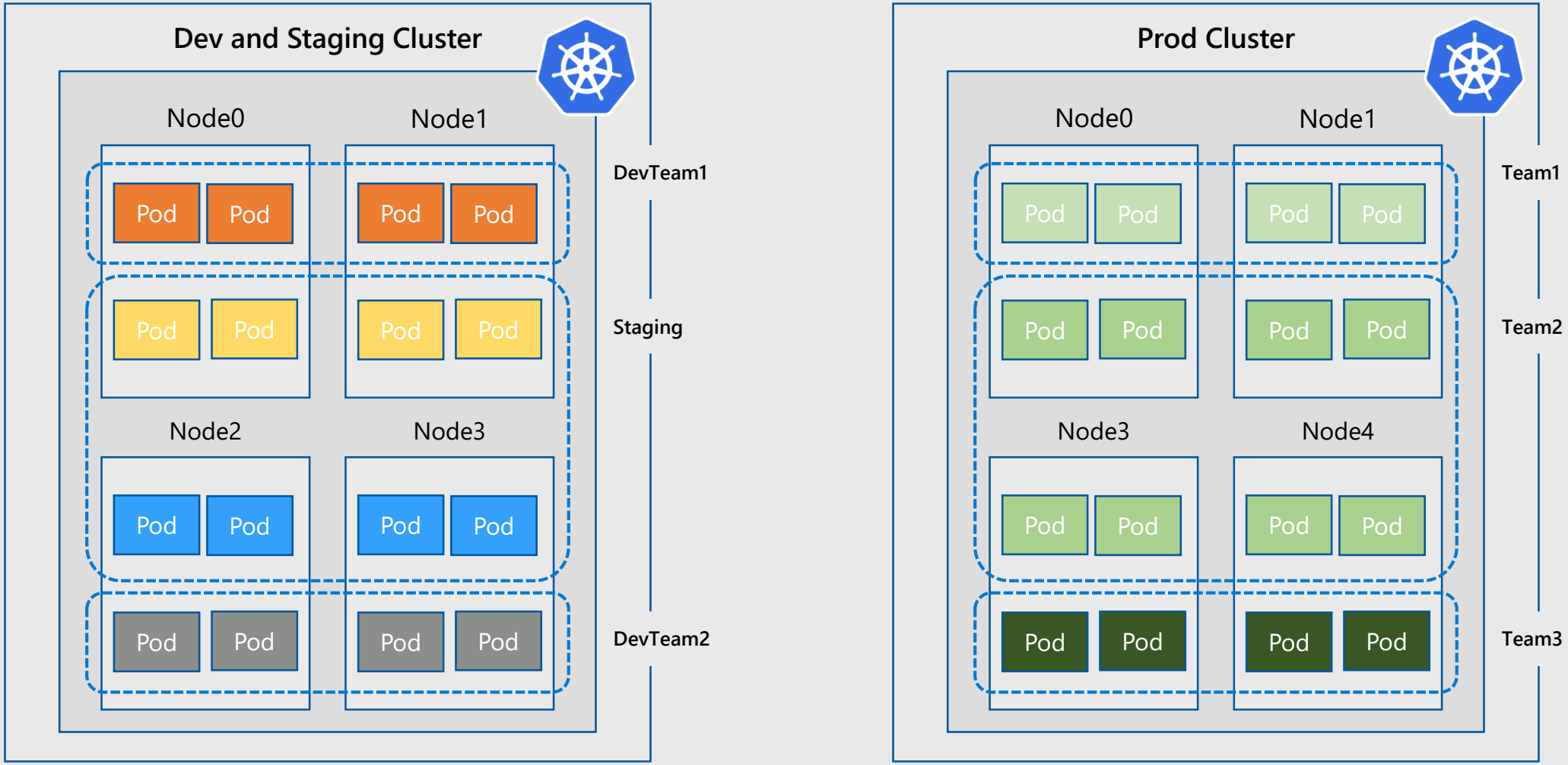


Cluster Isolation Patterns: Physical Isolation



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

240+ charts exist for standard software/tools/packages github.com/helm/charts

https://helm.sh

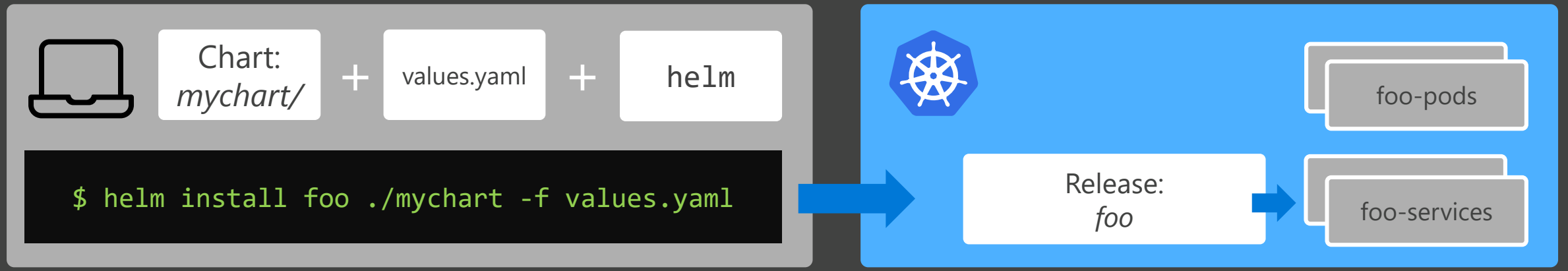
CLOUD NATIVE COMPUTING FOUNDATION

- 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

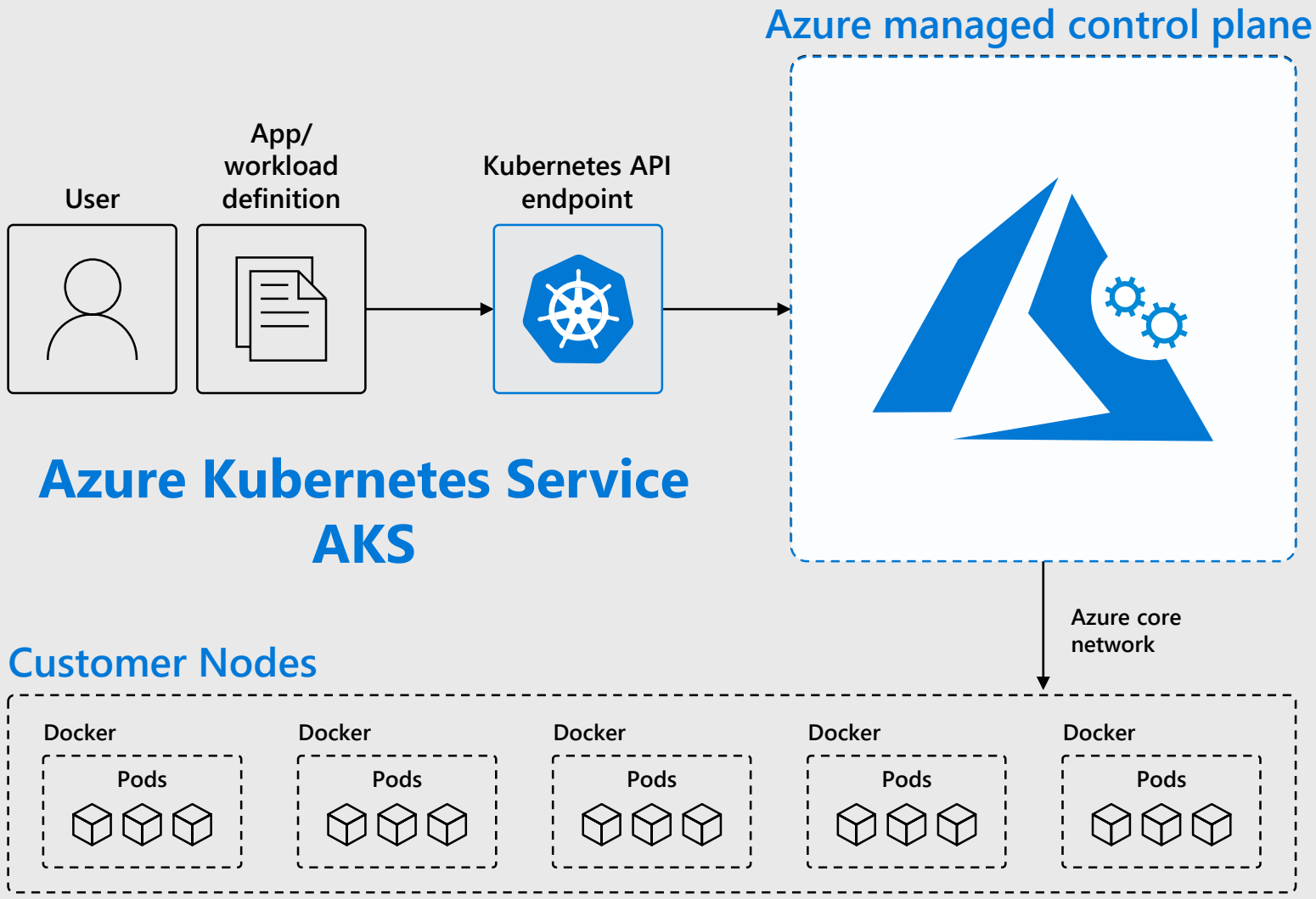
`helm`	Client tool to manage and work with Helm
Chart	Package of Kubernetes resources in template form
Template	Kubernetes YAML with directives in Go template language format, e.g. {{ blah }}
Release	When installing a chart into Kubernetes it becomes a release
Values	Used at install time to customise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



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 Virtual Nodes

Elastically provision compute capacity in seconds

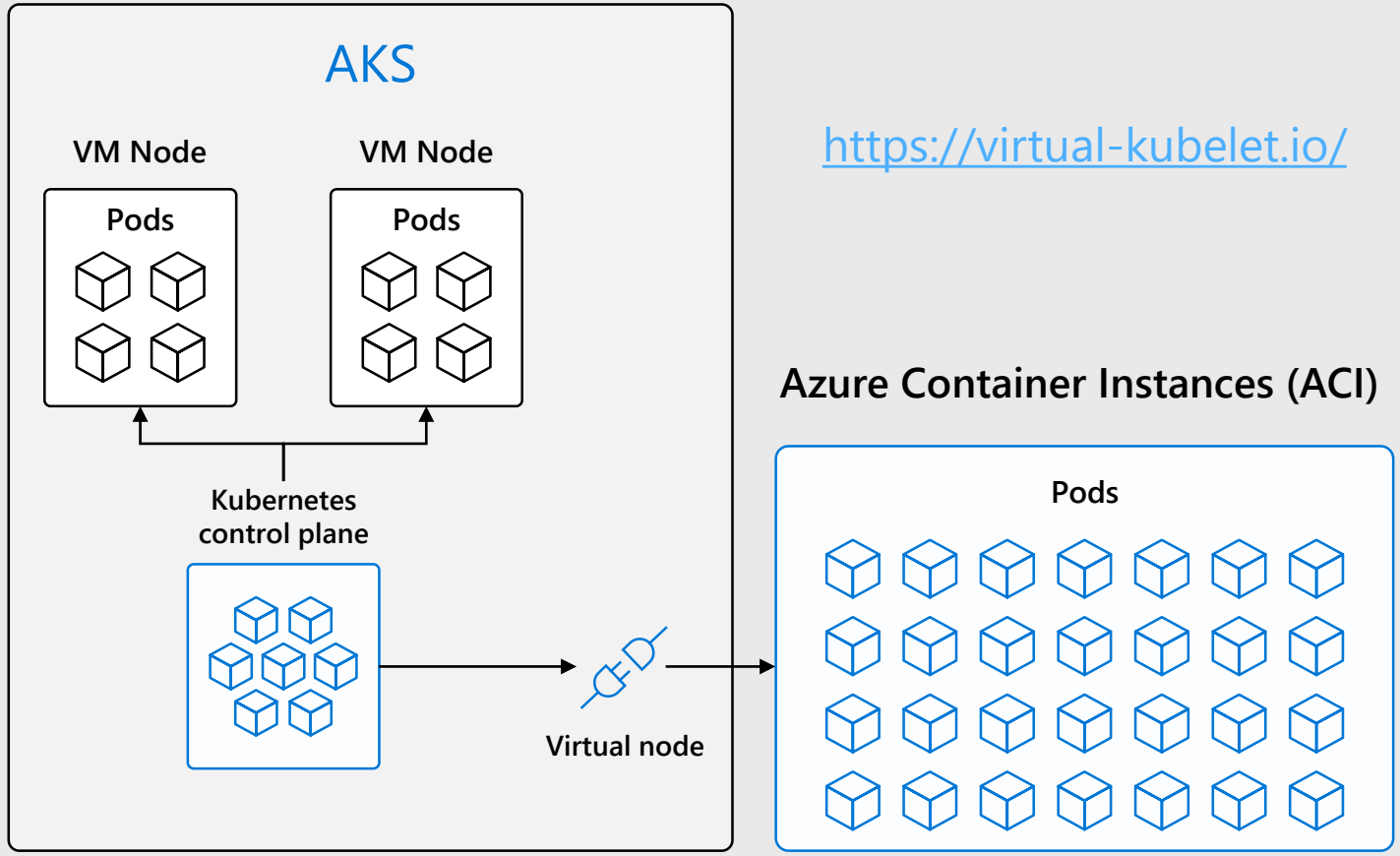
No infrastructure to manage

Builds on top of Azure Container Instances (ACI)

Uses Virtual Kubelet

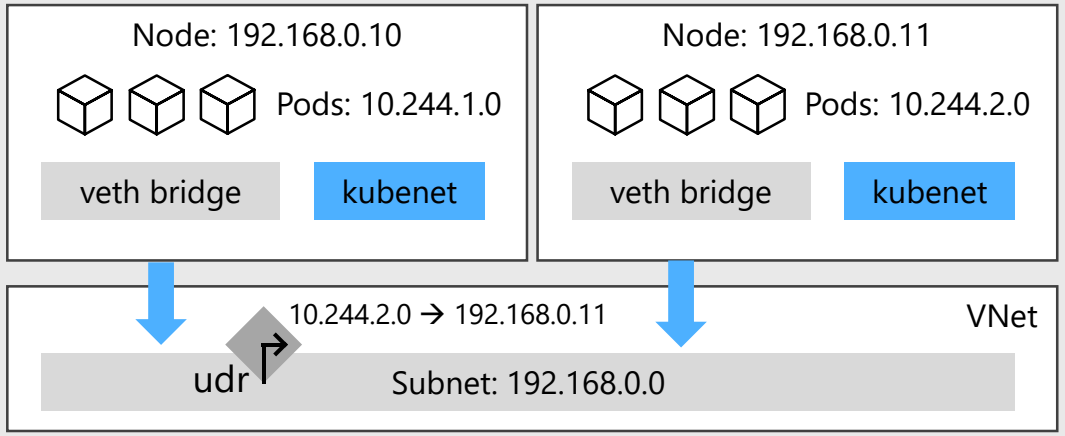
Scenarios:

- Bursting
- CI/CD
- “Serverless Kubernetes”



https://virtual-kubelet.io/

AKS Networking Models



Basic

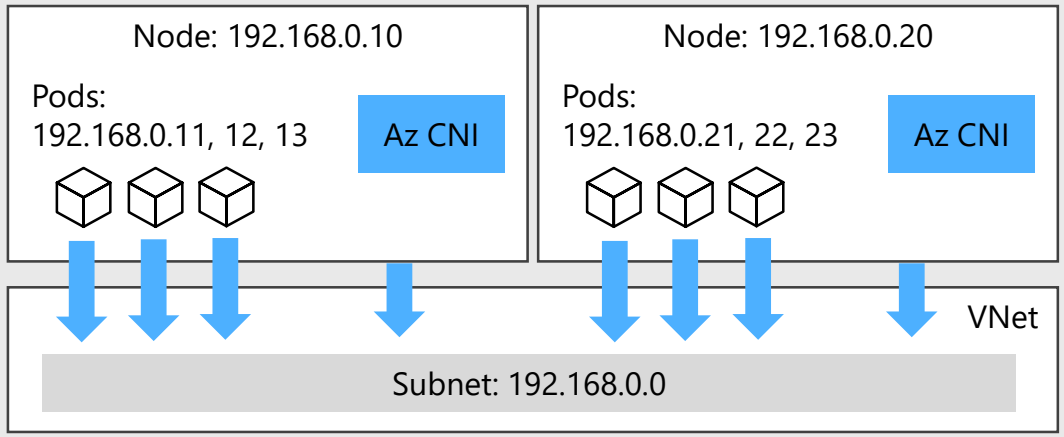
Uses **kubenet** network plugin

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

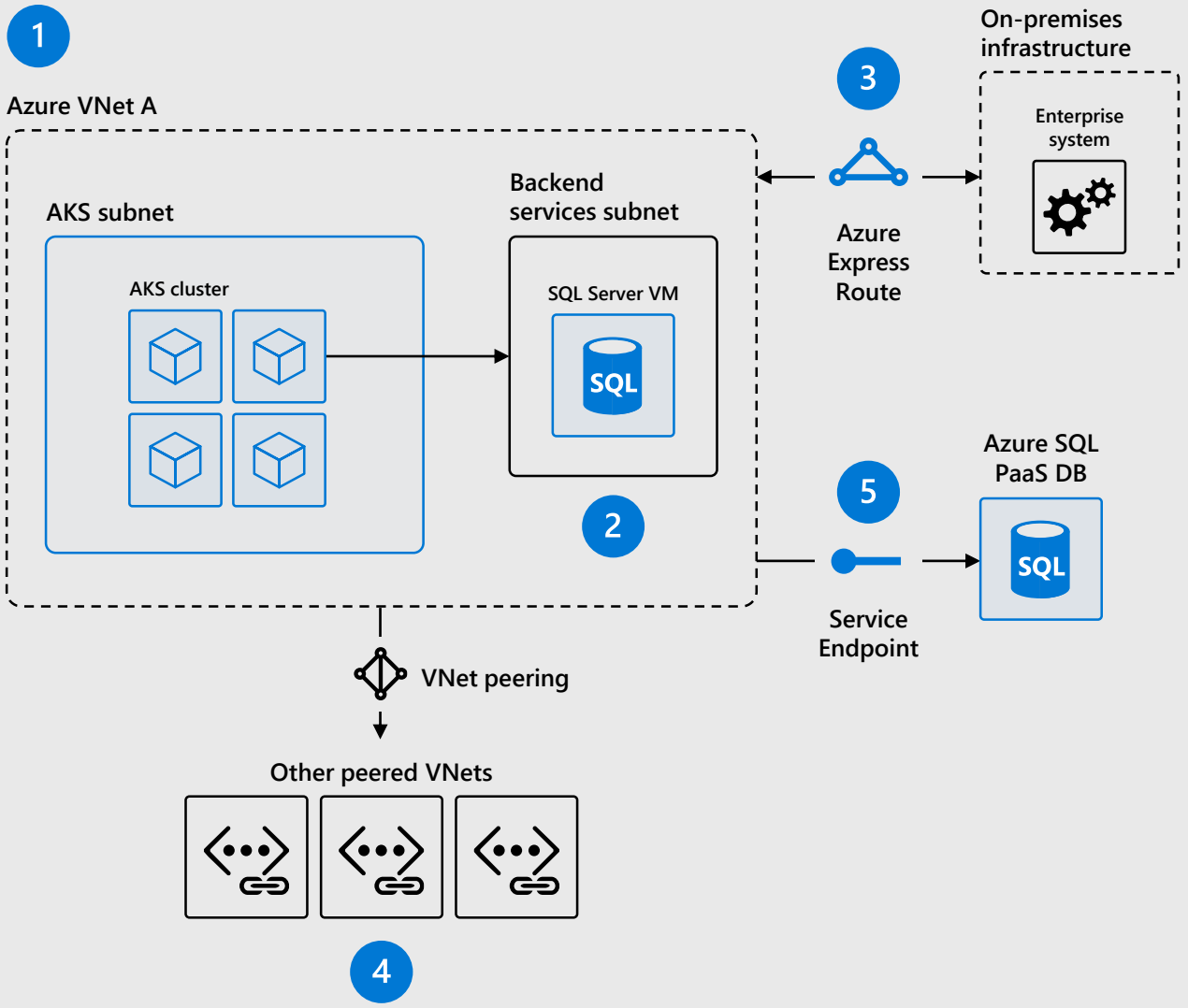
Works with peering, service endpoints, ExpressRoute

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



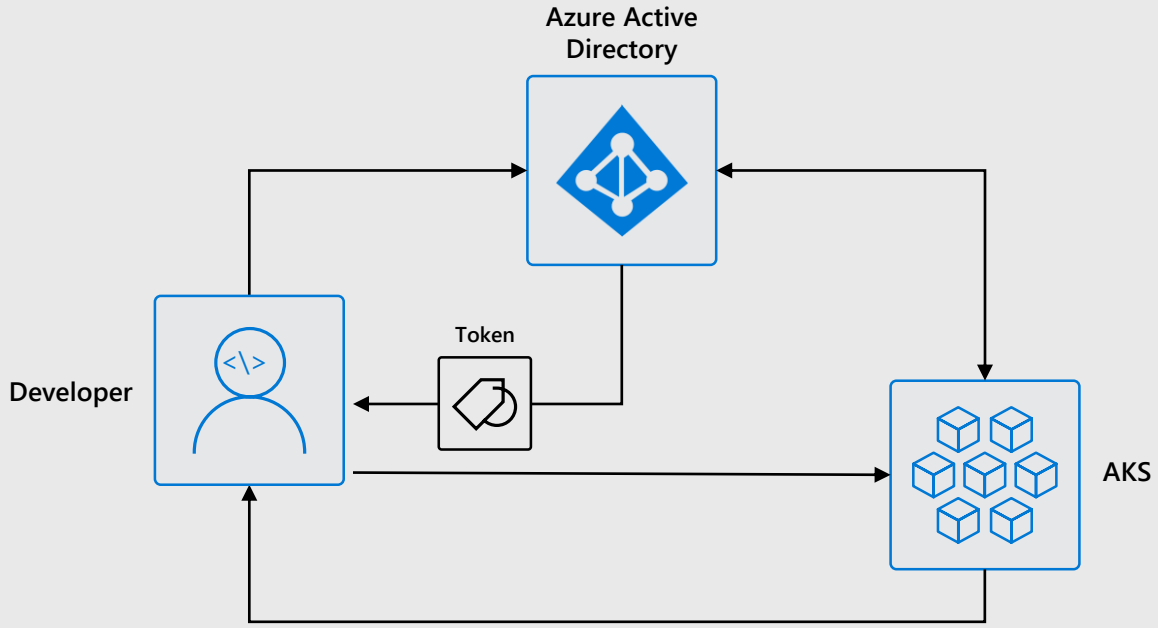
Azure Active Directory

docs.microsoft.com/azure/aks/aad-integration

Use Azure Active Directory to authenticate AKS users

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

Roles in AAD define who has access to the cluster



Azure Kubernetes Service Cluster Admin Role

Azure Kubernetes Service Cluster User Role

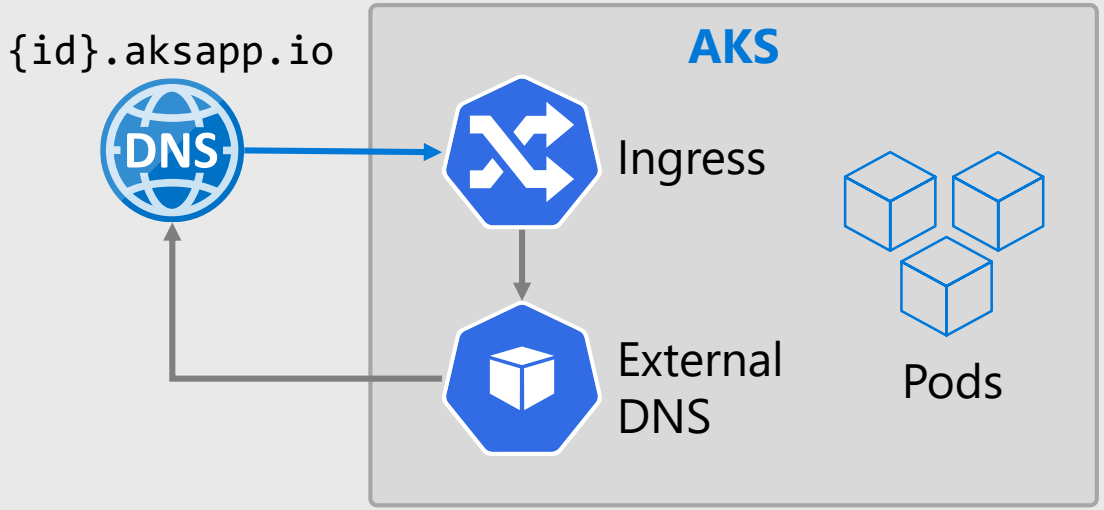
Note. Can only be enabled at cluster creation time

‘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

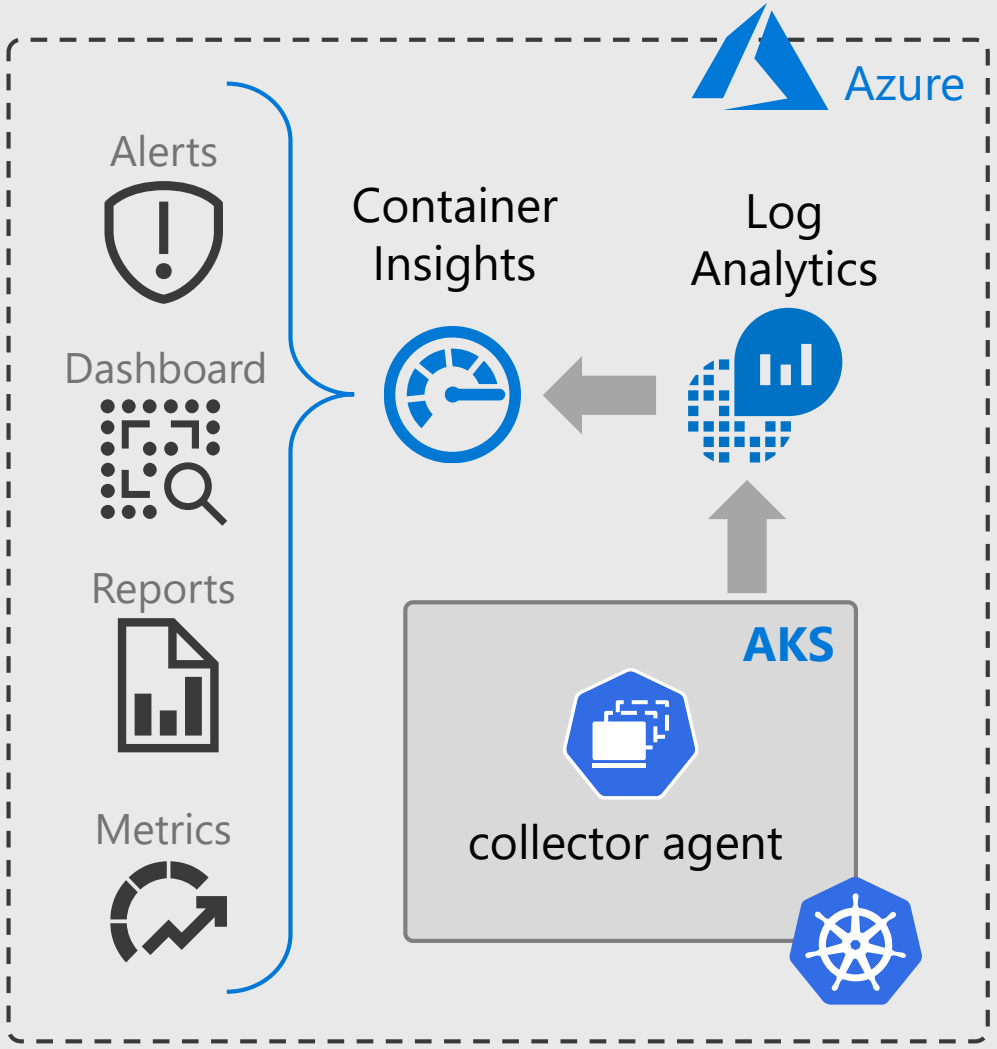
Insights
Provide insights with multi-cluster health roll up view

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}
```



Cluster Auto Scaler

Autoscaling with Azure Scale Sets

PREVIEW

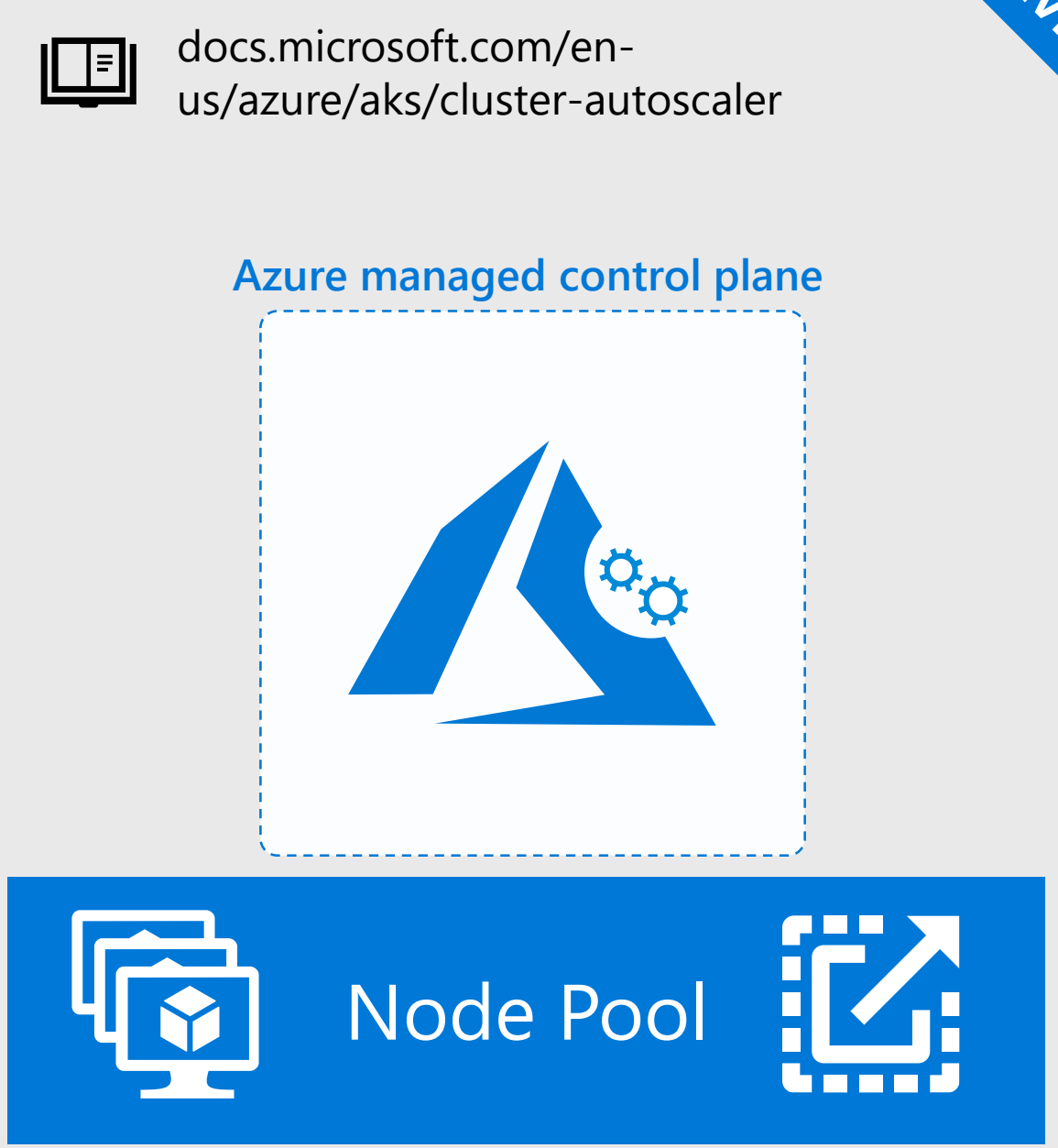
docs.microsoft.com/en-us/azure/aks/cluster-autoscaler

Enables the standard Kubernetes Cluster Autoscaler (CA)

Uses VM Scale Sets (Preview)

While in preview - hidden behind CLI extension

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



Azure Dev Spaces

Simplifying Development in Kubernetes

PREVIEW

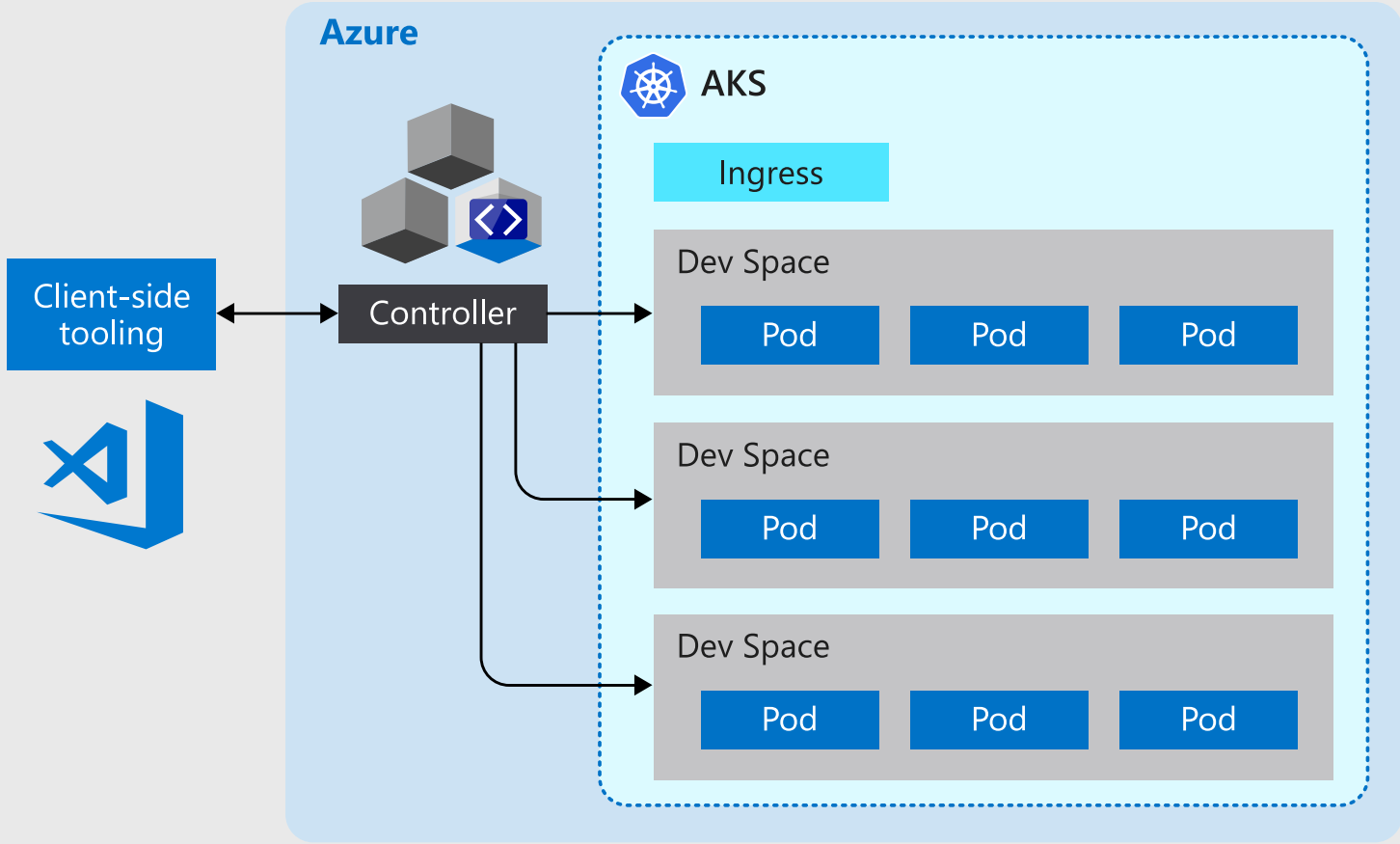
docs.microsoft.com/en-us/azure/dev-spaces/

Minimize local dev machine setup and work directly in AKS

Rapidly iterate & debug directly in AKS using Visual Studio 2017 or Visual Studio Code.

Generate Docker and Kubernetes configuration-as-code assets

Share a Kubernetes cluster with your team



Additional Resources

Kubernetes Docs

API Reference

Kubernetes Hands On Lab

Kubernetes Workshop

Katacoda Courses

Udemy Course

THANK YOU

Change Log

1.0 – Initial release

1.1 – Typos and fixes, added final additional resources slide

1.2 – Added AKS section & taints/affinity

1.3 – Extra sections on AKS

1.4 – Minor refresh, ConfigMaps and service mesh

Hidden Slide