A CLOUD GURU

CHAPTER ONE

Intro to Containers

A CLOUD GURU

In the beginning was the server

Tim Berry
GOOGLE CLOUD GURU

A CLOUD GURU

Deploying applications on physical servers

- Get **financial approval**, file a **purchase order**
- Take **delivery**, **unbox** it, dispose of the **packaging**
- Lift and **rack the server** – make sure you have the right **cage nuts**!
- Cable up **power** and **network**
- Install the **operating system**
- Install the **application** and **dependencies**

In the beginning was the server

Things are going great!

- Your application is getting **super popular**
- Buy **more servers**
- Repeat **all the previous steps**
- Lead time is in the **order of months**
- Large capital **expenditure**

In the beginning was the server

But what if things go wrong?

- Hard disks **fail**
- System boards **short out**
- **High availability**
- Off-peak utilization = **wasted resources**

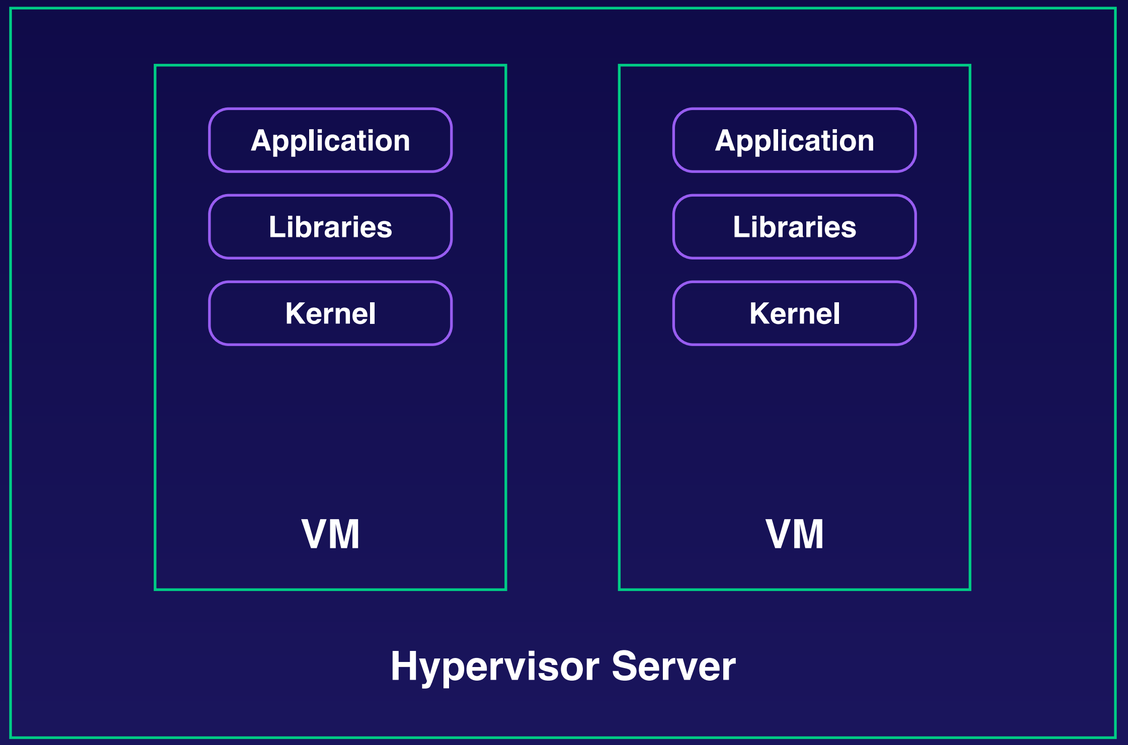
In the beginning was the server

Virtualization

- Made popular by **VMWare**, **Hyper-V** and **KVM**
- **Abstract** the hardware away
- Quickly deploy **virtual machines**
- VMs are **isolated** and can be **portable**
- Better hardware **utilization**

In the beginning was the server

In the beginning was the server



Virtual problems

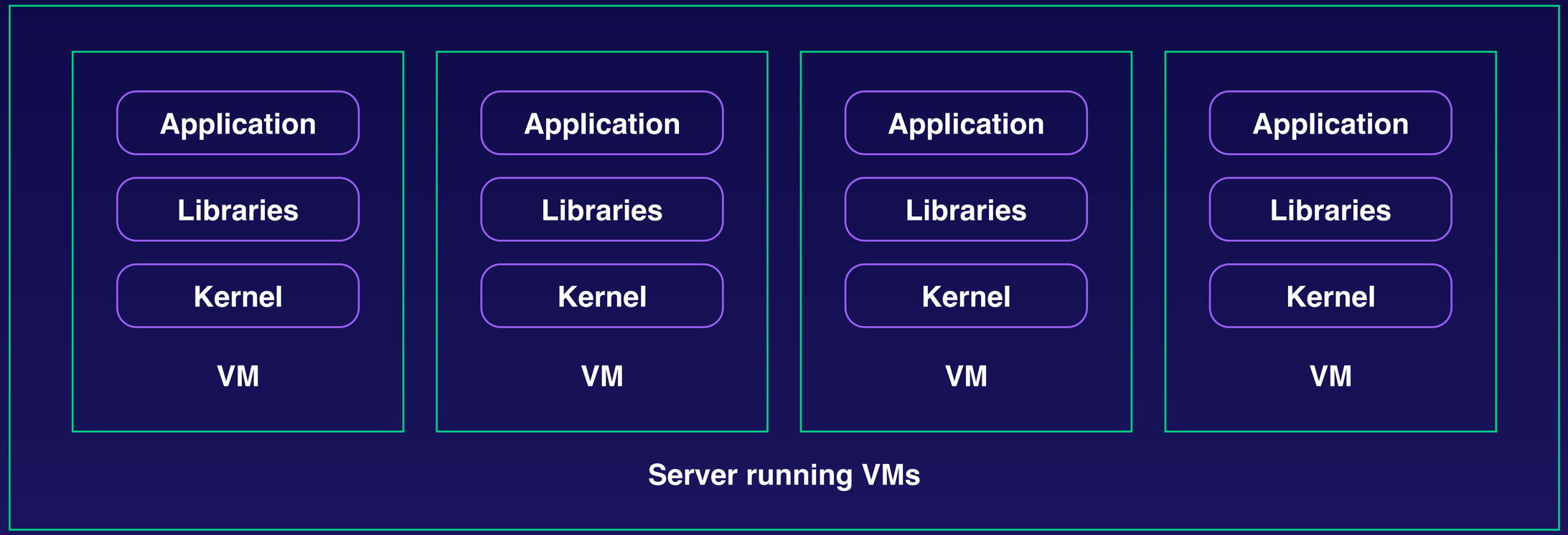
- Single operating system, libraries and dependencies **per VM**
- Difficulty separating **applications**
- Hardware utilization **suffers**
- No **automation problems** solved
- **Complex** configuration management systems

A CLOUD GURU

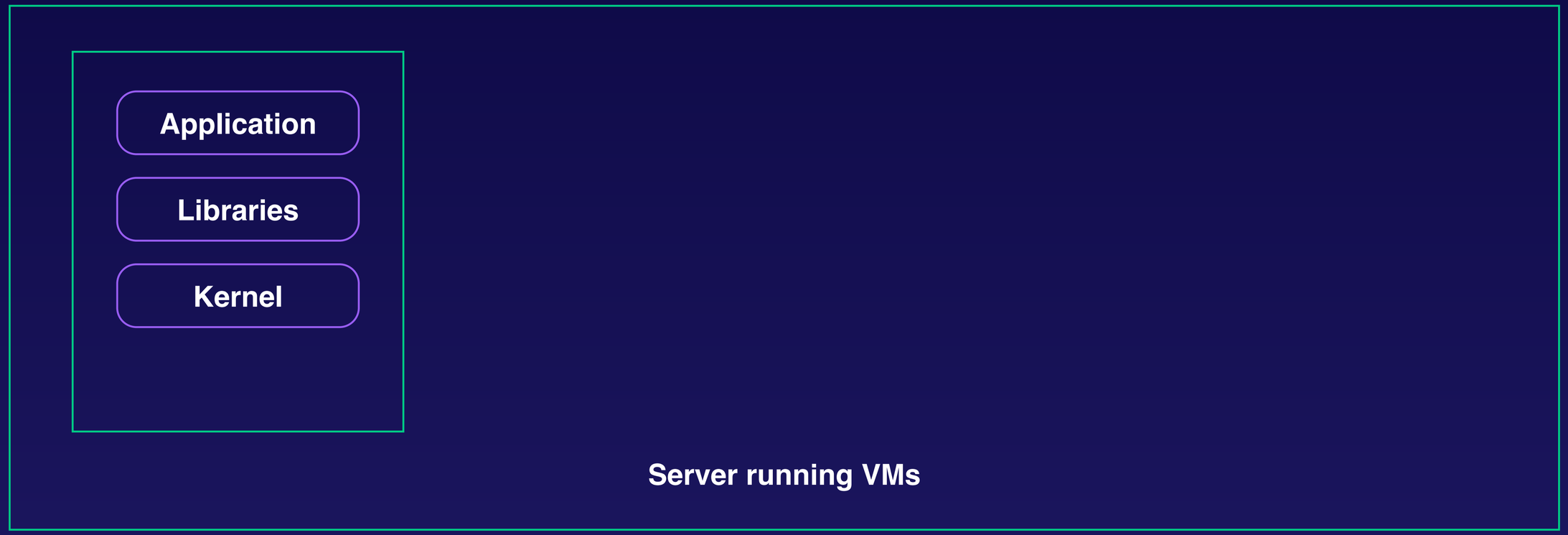
Containers to the rescue

Tim Berry
GOOGLE CLOUD GURU

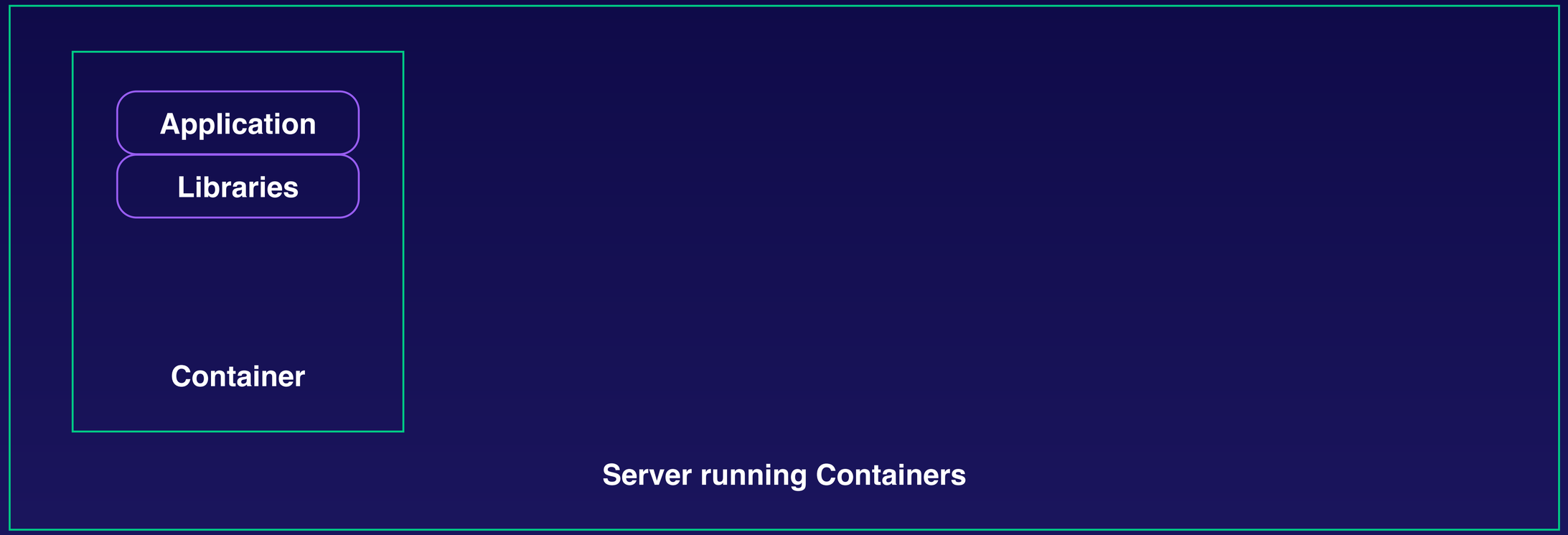
Containers to the rescue



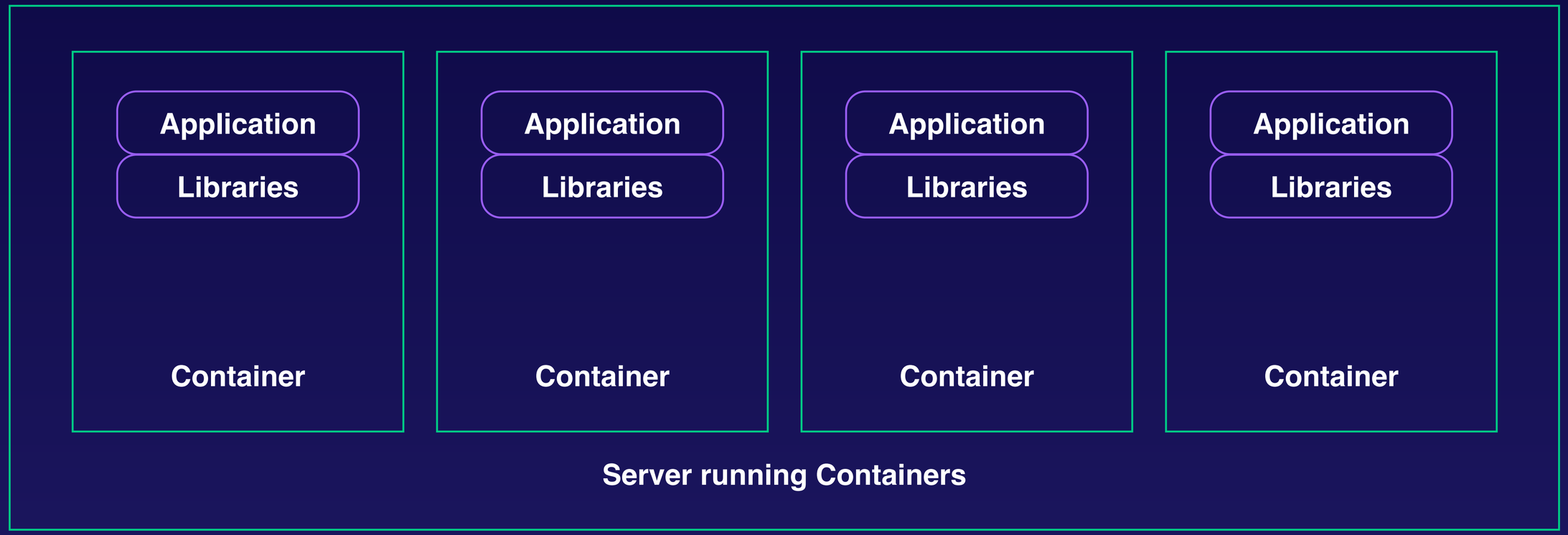
Containers to the rescue



Containers to the rescue



Containers to the rescue



Containers to the rescue

A CLOUD GURU

Containers at Google

Powering Maps, Gmail
and other services

2004

2006

cgroups

A kernel module for running
isolated user processes
(containers)

LMCTFY

Let Me Contain That For You
The foundation of Docker and
other container toolsets

2013

A CLOUD GURU

Kubernetes

Inheriting the concepts of Borg, but created as an open source project

2003-2013

2014

WEEKLY DEPLOYS

4 BILLION

Borg

Google's internal large-scale container and cluster management system

A CLOUD GURU

Containers at Google

- Used to power popular services since **2004**
- Developed **cgroups** kernel module in **2006**
- Open-sourced **LMCTFY** in **2013**
- Developed **Kubernetes** based on principles of **Borg**
- Graduated **Cloud Native Computing Foundation** project
- Over **4 billion containers** launched every week

A CLOUD GURU

Benefits of using containers

- Write **once**, run **anywhere**
- Easily deploy to **dev**, **test**, **production**
- Easily deploy to **GKE**, **VMs**, **bare metal**
- **Packaged** applications **speed** development
- Make changes **fast**
- Promote **microservices**

Docker

- Docker created a **fantastic toolset** around Linux containerization features
- **Developer-friendly** container creation and deployment
- Tools for handling container **images** and **registries**
- **Swarm** - Docker's own orchestration system
- Runtime and image specifications managed by the **Open Container Initiative**

A CLOUD GURU

Your first container

Tim Berry
GOOGLE CLOUD GURU

A CLOUD GURU

Containers start life as an image

- Just like any other **package of data**, like a zip or tarball
- Containers use a **layered read-only** filesystem
- A **running** container has a **read-write** layer on top of the image
- When the container **stops**, the read-write layer is **lost**

A CLOUD GURU

Dockerfile

- Simple **text** configuration file
- Use **existing images** as building blocks
- **Add** files, **run** commands, and perform **other actions**
- Specify what should be **running** when the container **runs**

Your first container


```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

- Inherit from an Ubuntu image
- Copy everything in local directory to /app
- Run make to compile our application
- When the container runs, execute our python app
- Save as **Dockerfile** in the local directory

Your first container


```
$ docker build -t myapp .
```

Your first container


```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

caa63f2d0bf0	1.895 KB
920af906e84d	194.5 KB
691b6fdff979	188.1Mb

myapp

Your first container


```
$ docker run -d myapp
```

Your first container


```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

8dd7a54d1cb6	0
caa63f2d0bf0	1.895 KB
920af906e84d	194.5 KB
691b6fdff979	188.1Mb

myapp

Your first container


```
$ docker ps
$ docker stop container-name
$ docker logs container-name
$ docker rm

$ docker images
$ docker tag myapp gcr.io/myapp
$ docker push gcr.io/myapp
$ docker rmi
```

- View running and stopped containers
- Stop, view logs, or delete containers

- View local images
- Tag and push to remote registries
- Remove local images

A CLOUD GURU

Lab time!

- Create a **Google Cloud Project**
- Use **Google Cloud Shell**

A CLOUD GURU

Lab:
Your first container

Tim Berry
GOOGLE CLOUD GURU

Lab: Your first container

In this lab:

- Use **Google Cloud Shell**
- Create a Python app in **Docker**
- **Run** and interact with **local containers**
- **Tag** a Docker image and **push** to a **registry**
- All you need is the **Google Cloud Console**

A CLOUD GURU

Ready to start your container journey?

Tim Berry
GOOGLE CLOUD GURU

Ready to start your container journey?

Containers in a nutshell

- An **efficient**, **developer-friendly** way of packaging an application
- Guaranteed consistency – write **once**, run **everywhere**
- Agile **creation** and **deployment**
- Isolated, elastic **microservices**

What are containers good for?

- Stateless microservices
- Frontends
- Backends
- Applications that need horizontal elasticity

Ready to start your container journey?

What are containers bad for?

- Applications that write to local disk
- Non-scaling resource – intensive monoliths
- Applications that require manual intervention to install
- Databases (unless you know what you're doing!)

Ready to start your container journey?

Containers in production

- Borg
- Kubernetes
- The **ultimate abstraction**

Ready to start your container journey?

Summary

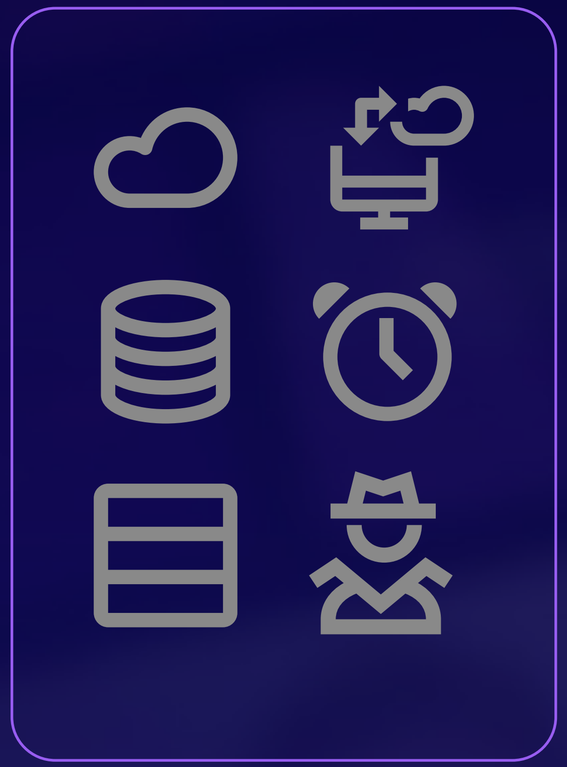
- History of application **deployment**
- History of **containers**
- **Docker**
- Production **orchestration**

A CLOUD GURU

CHAPTER TWO

Intro to GKE

A CLOUD GURU



A CLOUD GURU

The anatomy of a cluster

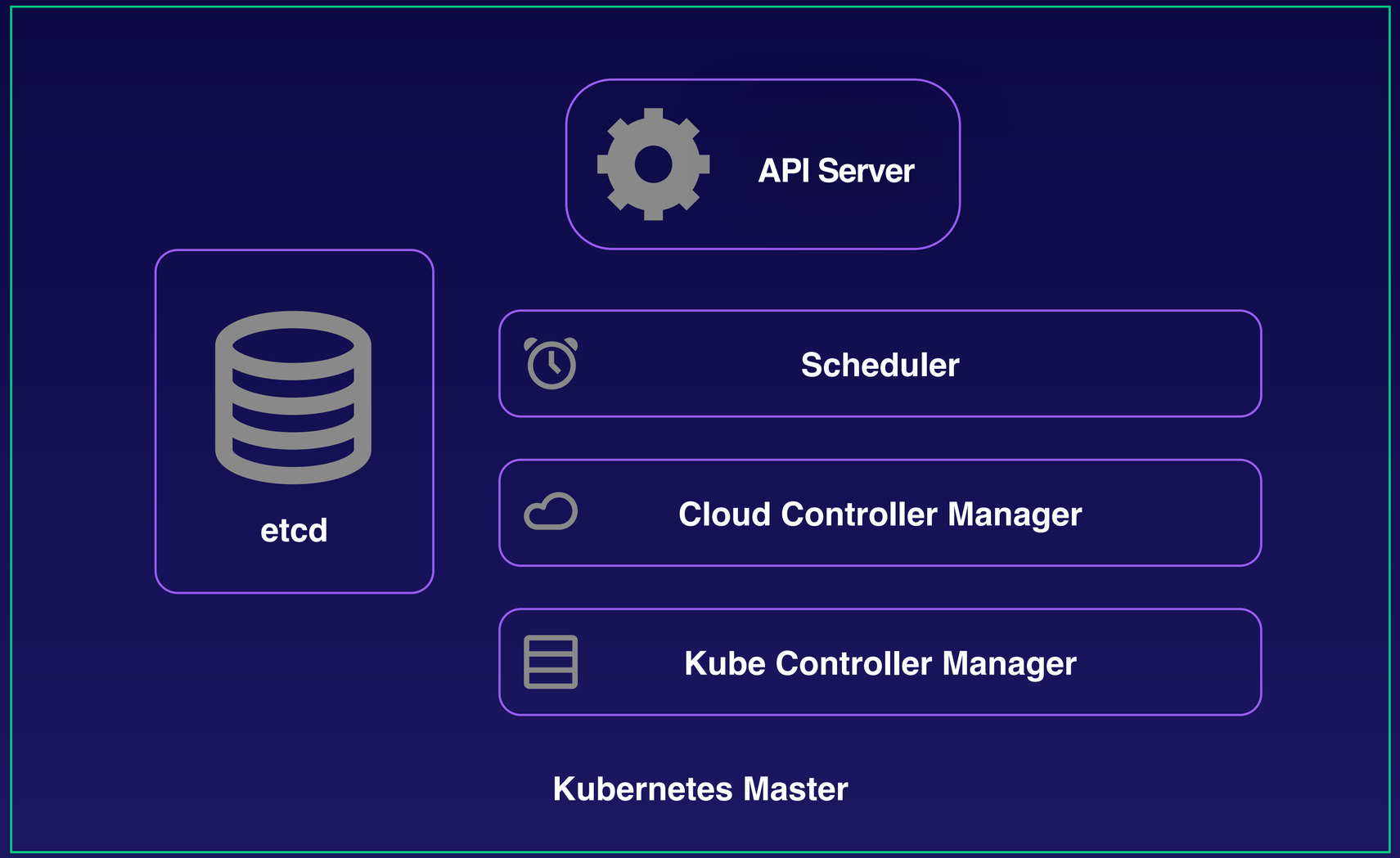
Tim Berry
GOOGLE CLOUD GURU

A CLOUD GURU

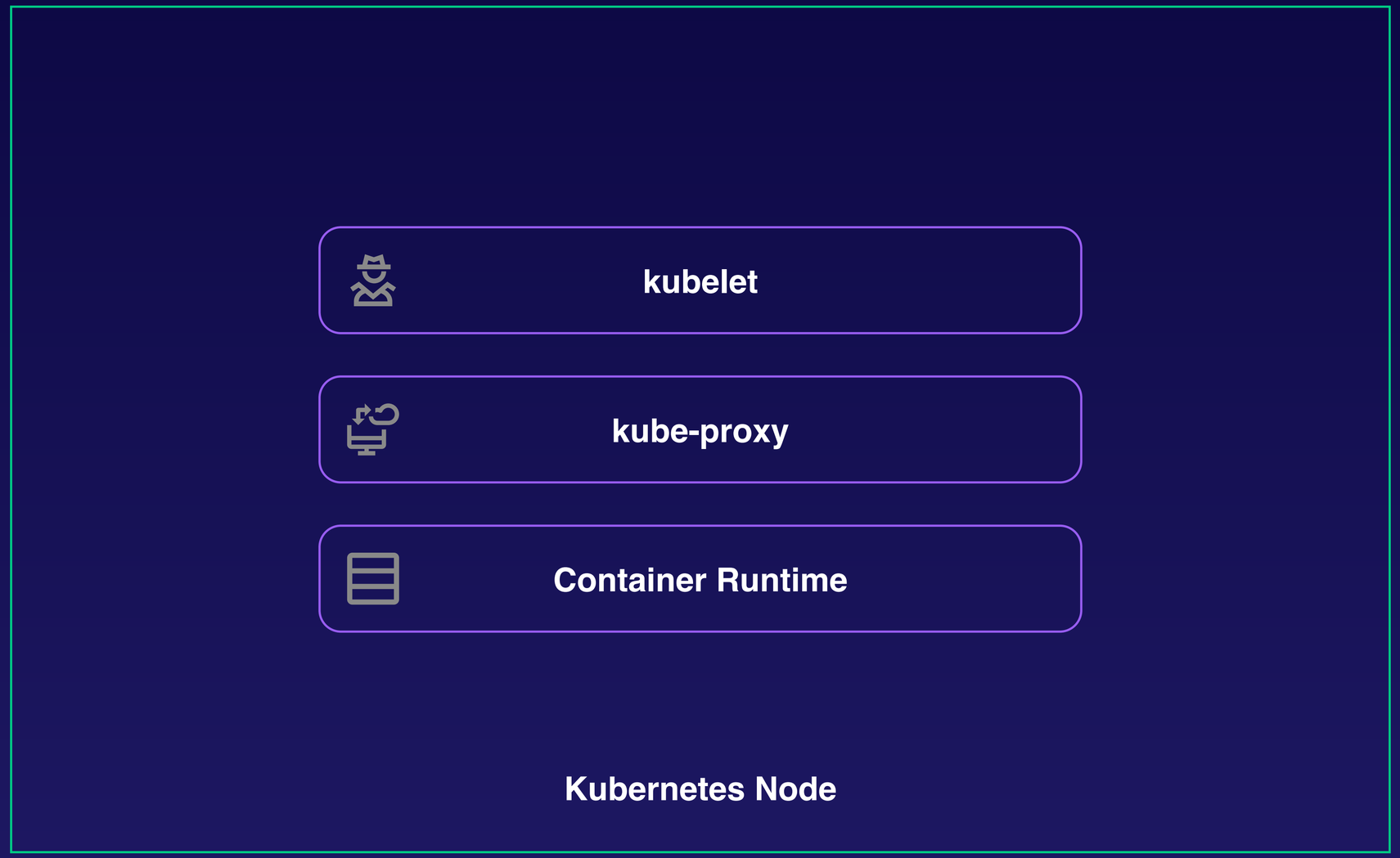
A Kubernetes Cluster

- One or more **masters**
- One or more **nodes**
- Master components provide **the control plane**
- They make **decisions** (eg. about scheduling)
- Node components provide the **runtime environment**
- They are the **resources** of the cluster

The anatomy of a cluster



The anatomy of a cluster



So how do I build a Kubernetes cluster?

- The **hard** way:
- Provision virtual machines for your **master** and **nodes**
- Install **Kubernetes**, create a **network overlay**, set up **CA**
- The **easy** way:
- **Use GKE**

A CLOUD GURU

“Google Kubernetes Engine provisions and manages the underlying cloud resources automatically.”

Google Cloud

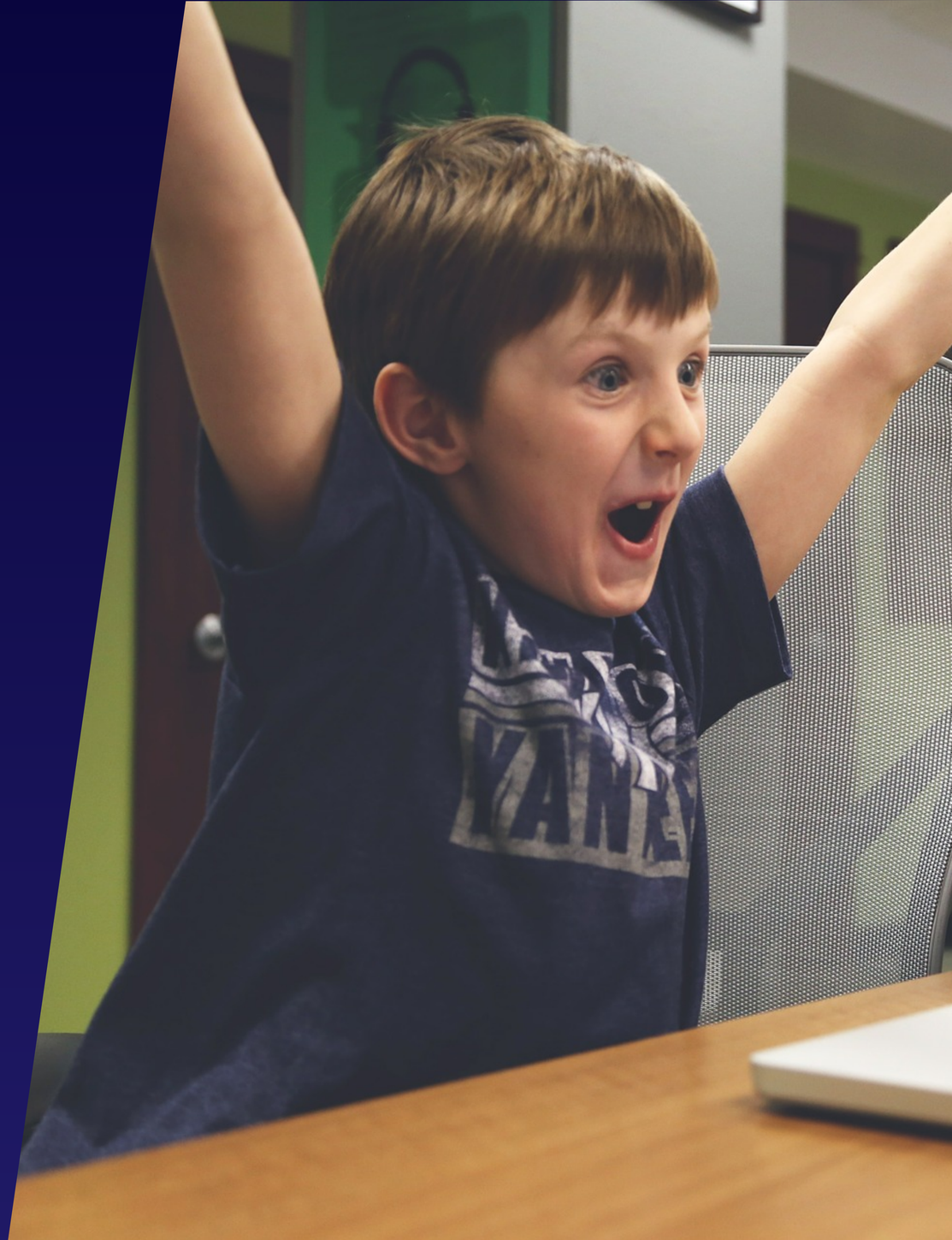
https://cloud.google.com/kubernetes-engine/

GKE: Fully Managed Kubernetes

- Creates **Masters** and **Nodes** for you
- Uses a **Container-Optimized** operating system
- Built-in automatic **upgrades** and **repairs**
- **Resource** control and cluster **scaling**
- **Integrated** with other GCP **products** and **services**

A CLOUD GURU

You don’t even touch the master control plane!



Interacting with GKE

- Google Cloud Console
- Command line
- **kubectl**
- **Create** and **manage** the lifecycle of Kubernetes **objects**
- Communicates with the **API server**

A CLOUD GURU

Lab time!

- Create a **GKE cluster**
- Deploy a **container**
- Learn **kubectl**

A CLOUD GURU

Lab: Creating a GKE cluster

Tim Berry
GOOGLE CLOUD GURU

Lab: Creating a GKE cluster

In this lab:

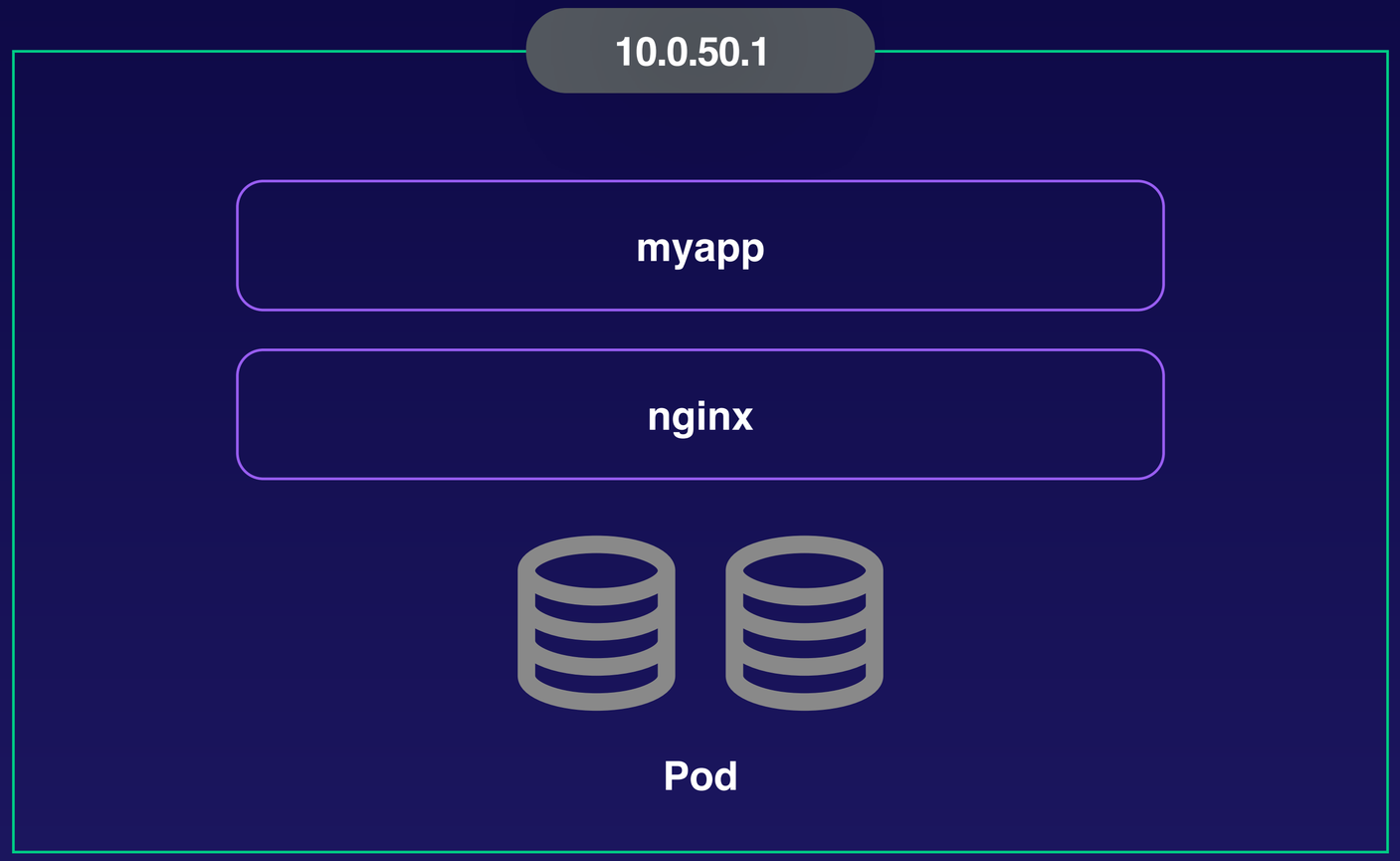
- **Options** for creating clusters with **GKE**
- Create our first **single node cluster**
- Deploy a **container** to our **cluster**
- Expose a **service** with a **load balancer**
- Experiment with **kubectl**

A CLOUD GURU

Forget containers, meet Pods!

Tim Berry
GOOGLE CLOUD GURU

Forget containers, meet Pods!



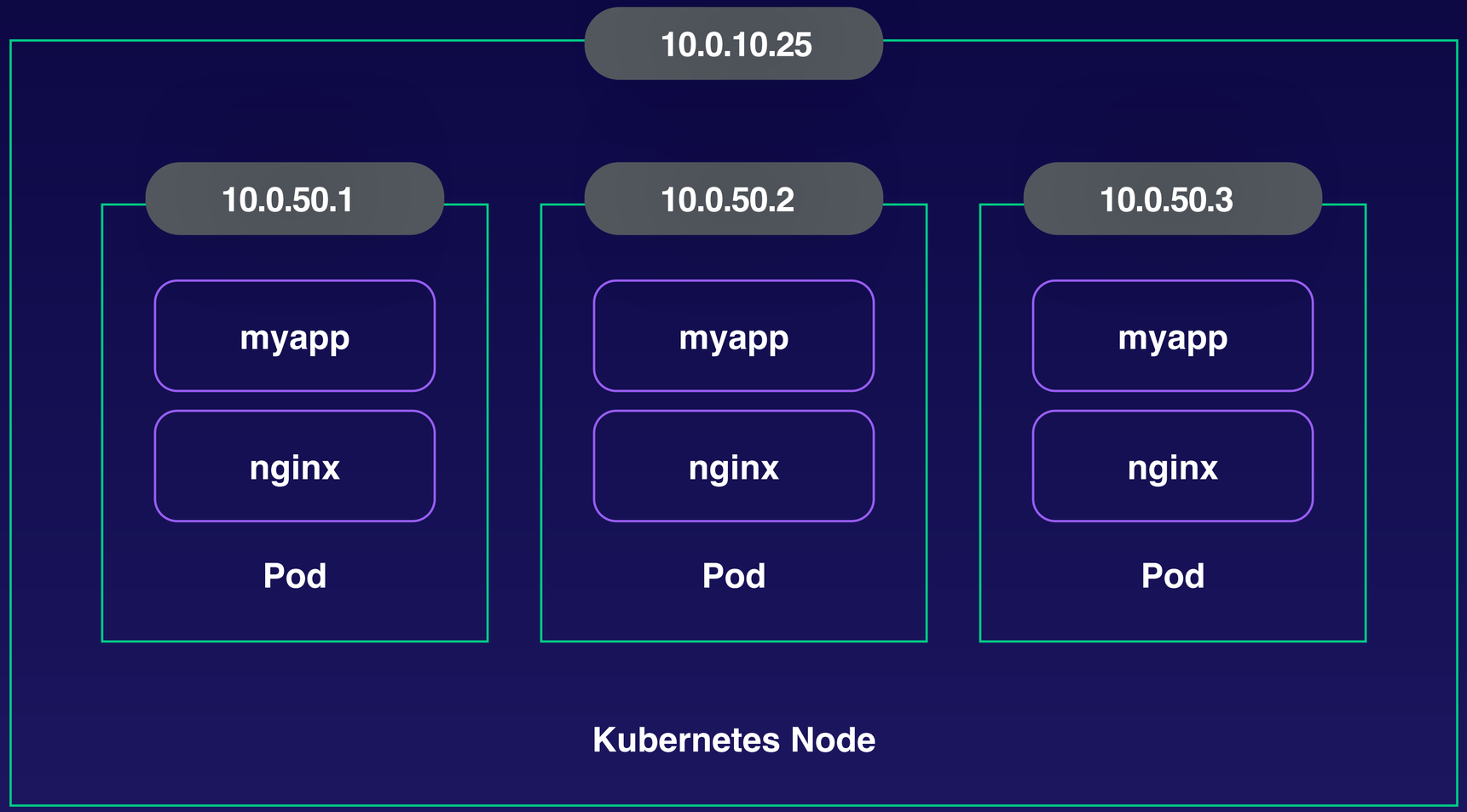
Pods are Kubernetes objects

- A logical **application-centric unit** for hosting containers
- Pods contain 1 or more **tightly-coupled** containers
- Example **design patterns** for pods:
  - A **monolith** and a **web server**
  - An **application** and a **database proxy**
  - An **application** and an **adapter** to process its output
- Pods should be designed around a **single application**

Creating a Pod

- Create objects **dynamically** with **kubectl**
- Create objects **declaratively**
- Create **specification** files in **YAML**
- **Apply** the **declarative** configuration
- Kubernetes attempts to **match the state** we declared

Forget containers, meet Pods!



Forget containers, meet Pods!


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
  - name: myapp
    image: myapp
  - name: nginx-ssl
    image: nginx
```

Forget containers, meet Pods!


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
  - name: myapp
    image: myapp
  - name: nginx-ssl
    image: nginx
```

Forget containers, meet Pods!


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
  - name: myapp
    image: myapp
  - name: nginx-ssl
    image: nginx
```

Forget containers, meet Pods!


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
  - name: myapp
    image: myapp
  - name: nginx-ssl
    image: nginx
```

Forget containers, meet Pods!


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
  - name: myapp
    image: myapp
  - name: nginx-ssl
    image: nginx
```

Forget containers, meet Pods!


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
  - name: myapp
    image: myapp
  - name: mginx-ssl
    image: nginx
```

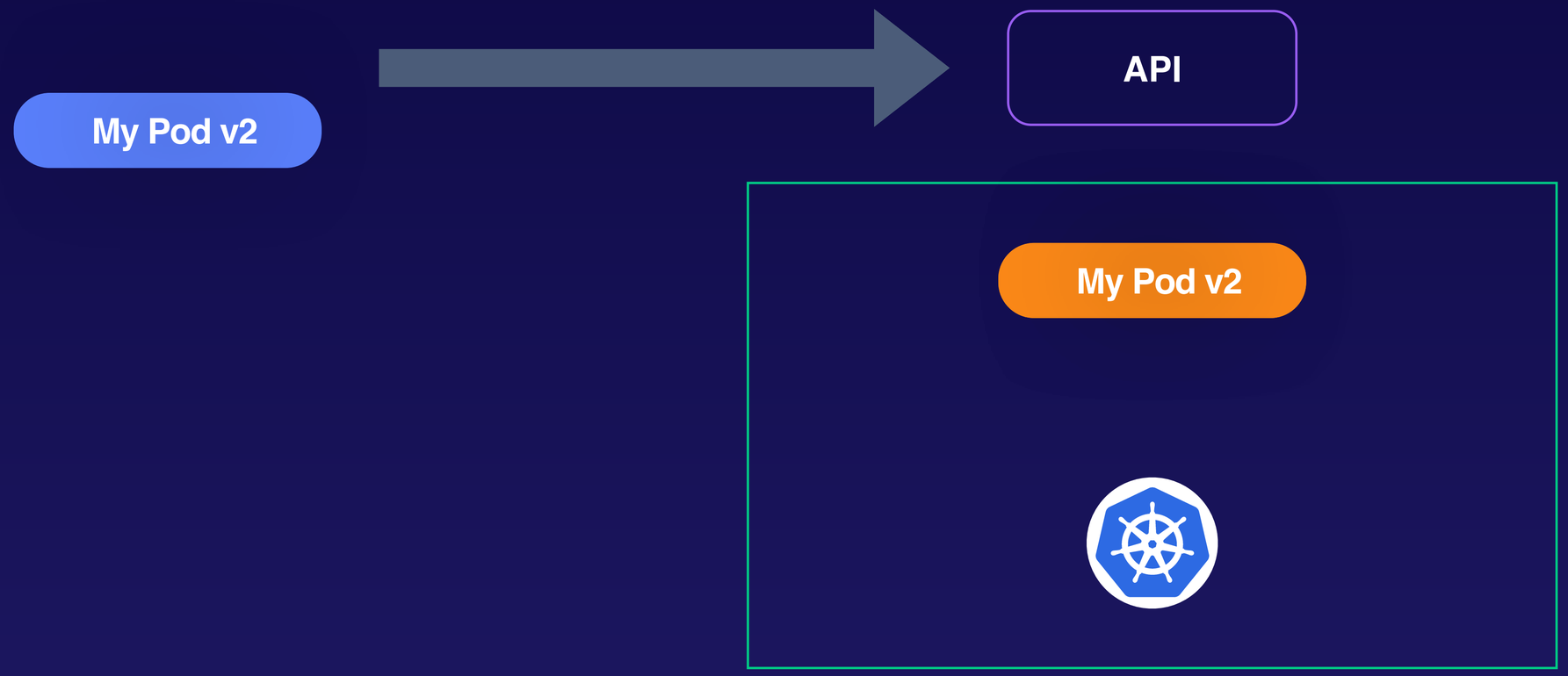
Forget containers, meet Pods!


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
  - name: myapp
    image: myapp
  - name: nginx-ssl
    image: nginx
```

Forget containers, meet Pods!


```
$ kubectl apply -f mypod.yaml
```

Forget containers, meet Pods!



Lab time!

- Create some **Pods** with **YAML**
- Create a **multi-container** Pod
- Explore pod **specifications** and **resources**

A CLOUD GURU

Lab: Creating Pods

Tim Berry
GOOGLE CLOUD GURU

Lab: Creating Pods

In this lab:

- Create some **Pods** with **YAML**
- Create a **multi-container** Pod
- Explore pod **specification options**

A CLOUD GURU

ReplicaSets and Deployments

Tim Berry
GOOGLE CLOUD GURU

ReplicaSets

- A **Pod** is logical application-centric **unit of containers**
- A **ReplicaSet** is a Kubernetes **object** that provides:
  - A stable set of Pod **replicas**
  - A **specified number** of Pods
  - Logic to ensure **availability**
- **Not recommended** to create ReplicaSets on their own

Deployments

- An **object** for logically managing **Pods** and **ReplicaSets**
- Desired state is **enforced** by the **Controller**
- Logic for **updating**, **rolling back** and **scaling** deployments
- Proportional **scaling** and **error checking** for rollouts
- The **preferred object** for deploying compute workloads

ReplicaSets and Deployments


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    containers:
    - name: nginx
      image: nginx:1.7.9
      ports:
      - containerPort: 80
```

ReplicaSets and Deployments


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    containers:
    - name: nginx
      image: nginx:1.7.9
      ports:
      - containerPort: 80
```

ReplicaSets and Deployments


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    containers:
    - name: nginx
      image: nginx:1.7.9
      ports:
      - containerPort: 80
```

ReplicaSets and Deployments


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    containers:
    - name: nginx
      image: nginx:1.7.9
      ports:
      - containerPort: 80
```

ReplicaSets and Deployments


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    containers:
    - name: nginx
      image: nginx:1.7.9
      ports:
      - containerPort: 80
```

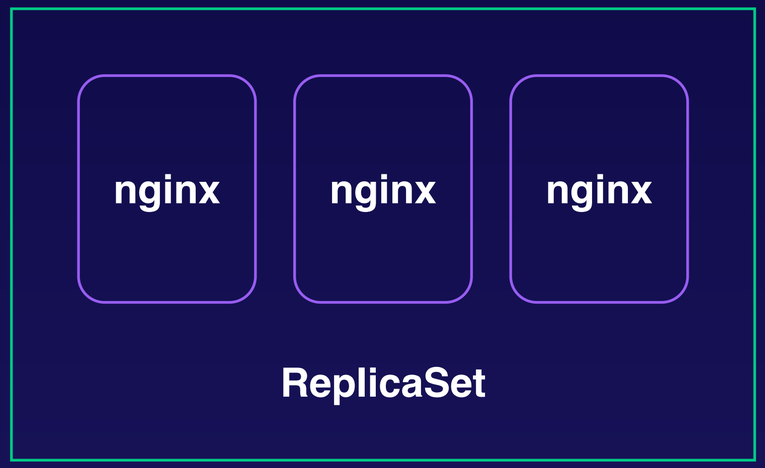
ReplicaSets and Deployments


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    containers:
    - name: nginx
      image: nginx:1.7.9
      ports:
      - containerPort: 80
```

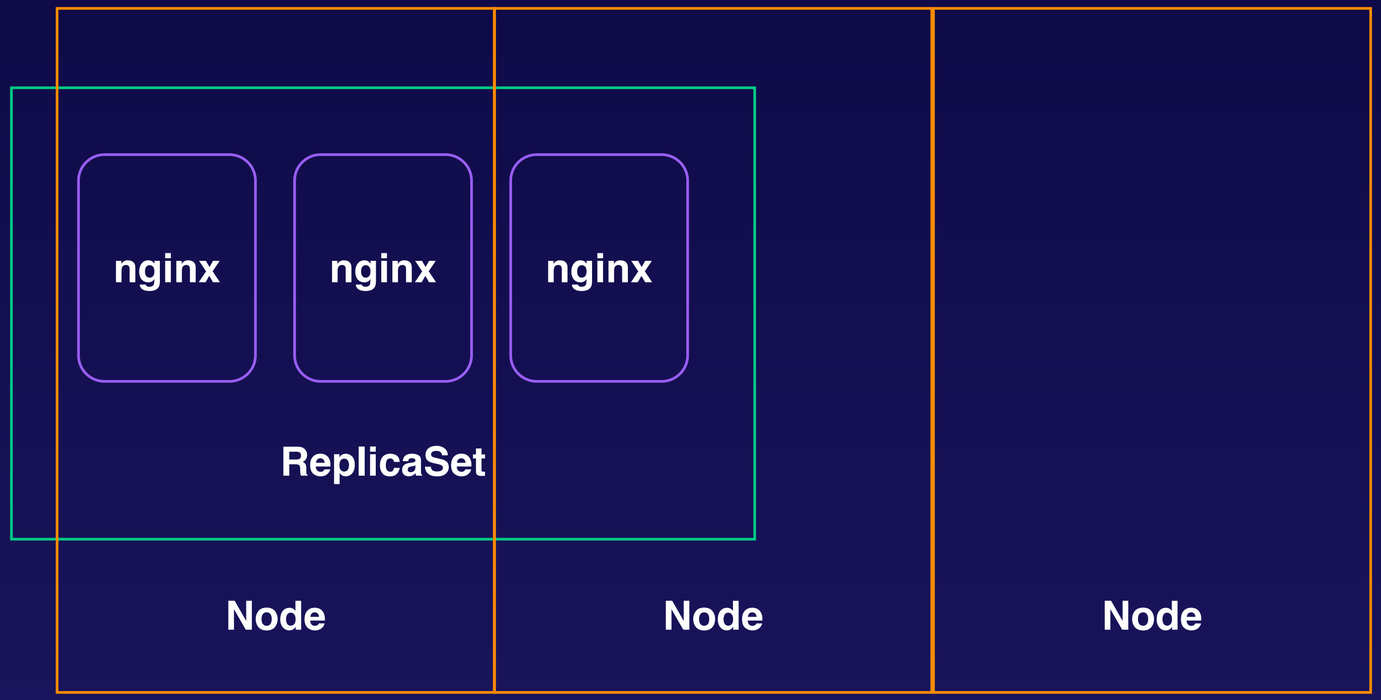
ReplicaSets and Deployments


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    containers:
    - name: nginx
      image: nginx:1.7.9
      ports:
      - containerPort: 80
```

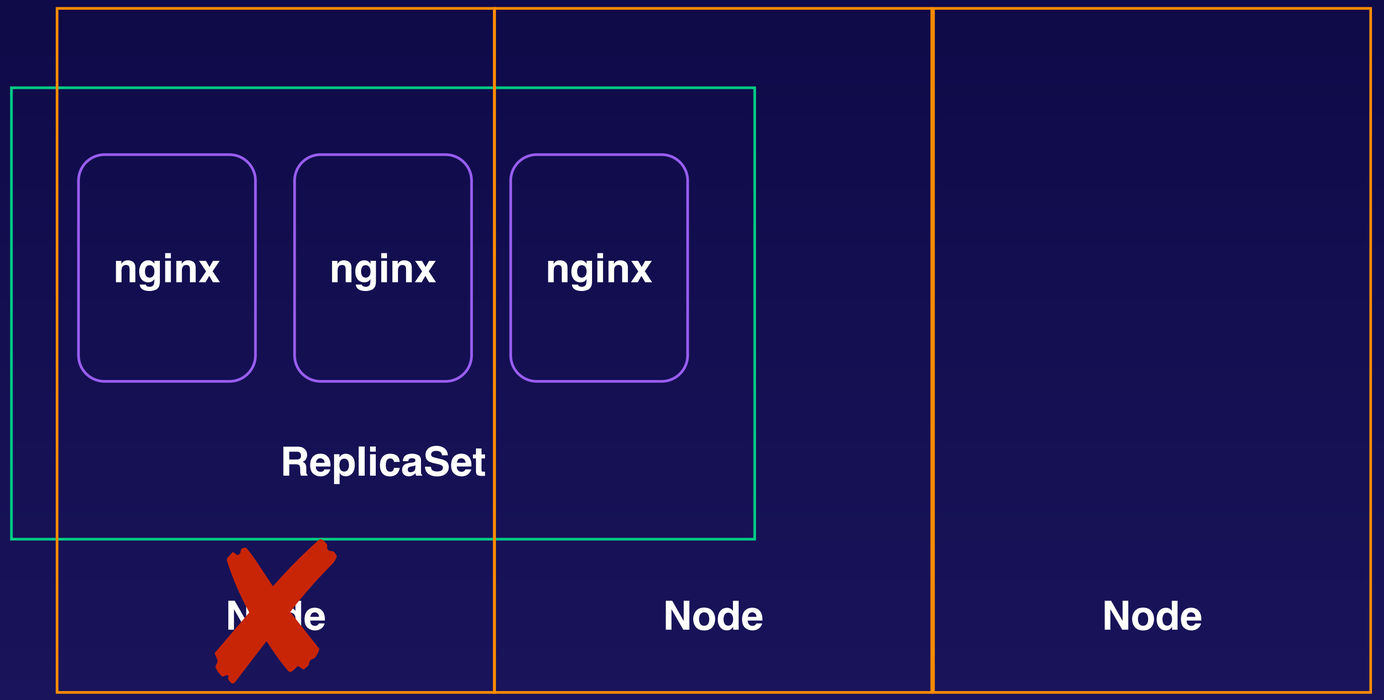
ReplicaSets and Deployments



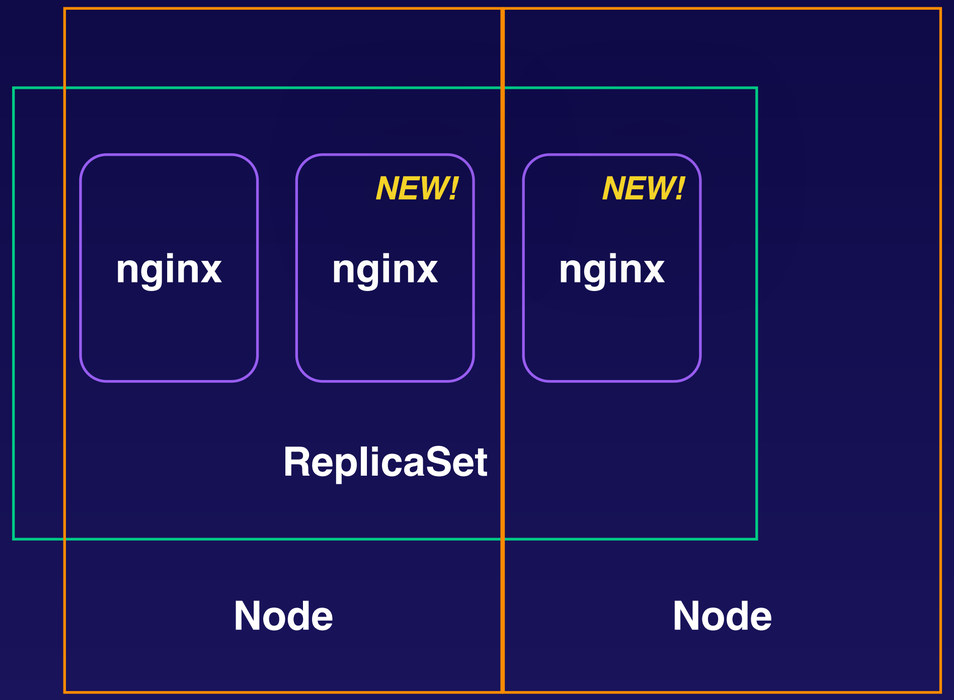
ReplicaSets and Deployments



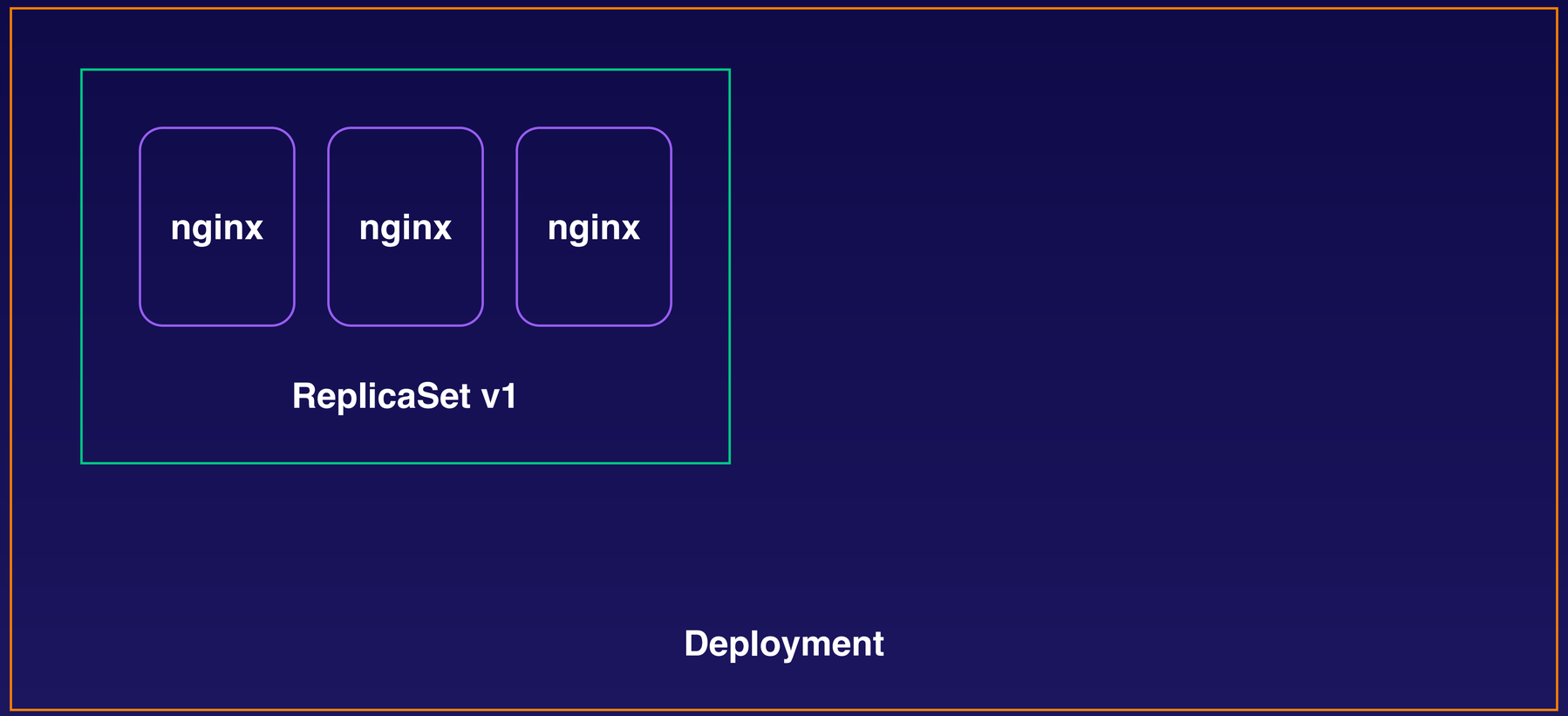
ReplicaSets and Deployments



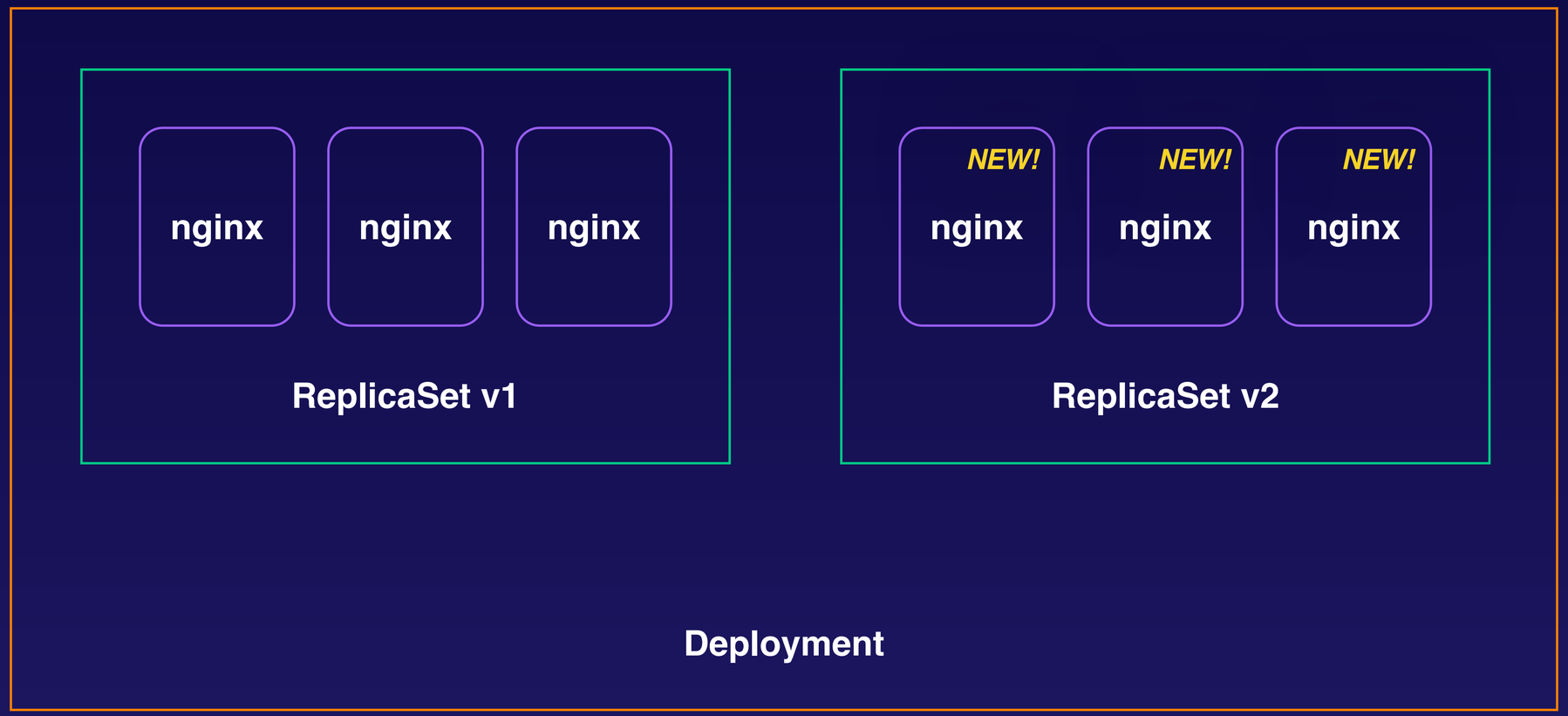
ReplicaSets and Deployments



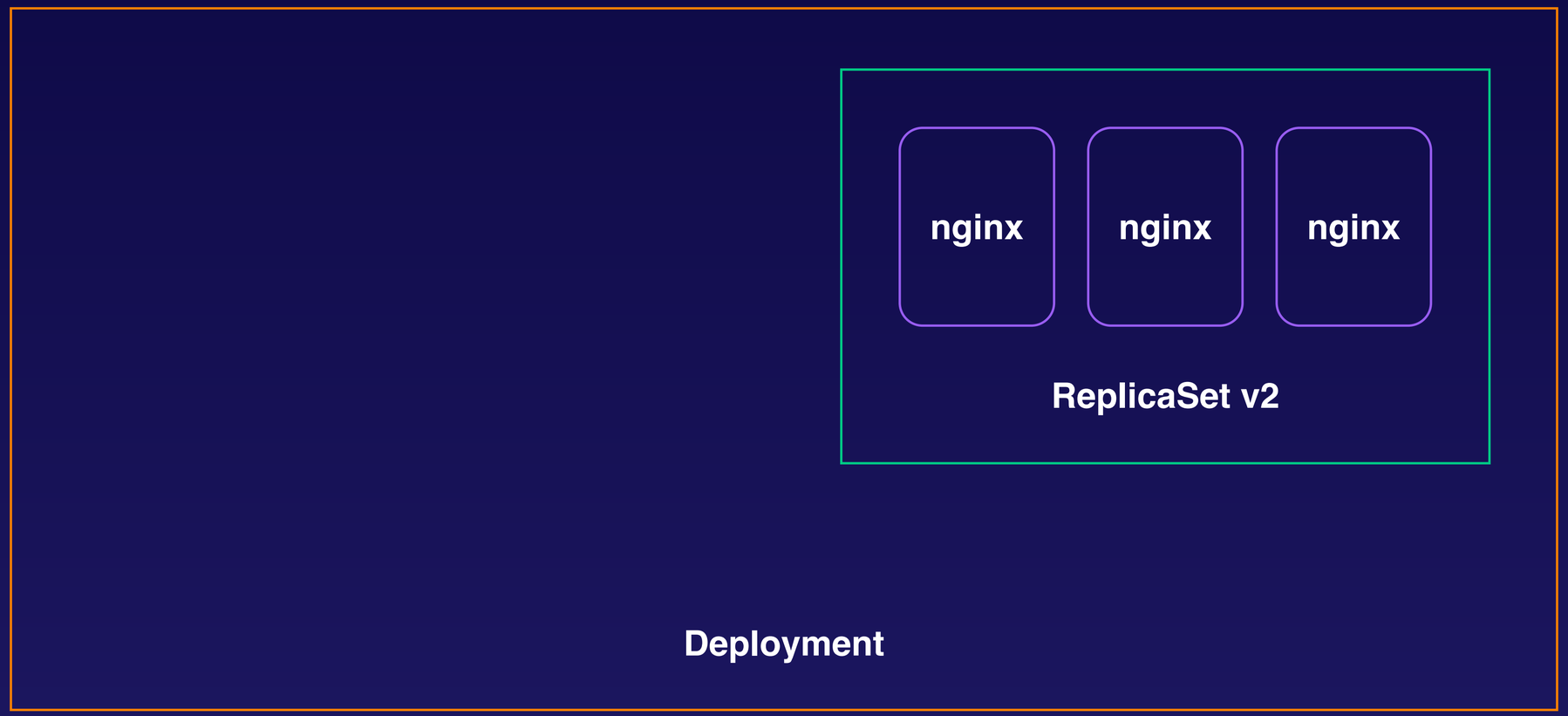
ReplicaSets and Deployments



ReplicaSets and Deployments



ReplicaSets and Deployments



A CLOUD GURU

Accessing Applications

Tim Berry
GOOGLE CLOUD GURU

A CLOUD GURU

Services

- A **Service** exposes a **set of Pods** to the **network**
- A **Service** assigns a **fixed IP** to your **Pod replicas**
- Three main types of **Service**:
  - **ClusterIP**
  - **NodePort**
  - **LoadBalancer**

A CLOUD GURU

Selectors

- Labels are **key-value pairs** in object **metadata**
- Selectors search for **groups of labels**
  - `app=nginx`
- Pods that **match** selectors become part of the **Service**

Accessing Applications


```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
 selector:
   app: nginx
 ports:
 - protocol: TCP
   port: 80
   targetPort: 80
```

Accessing Applications


```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
 selector:
   app: nginx
 ports:
 - protocol: TCP
   port: 80
   targetPort: 80
```

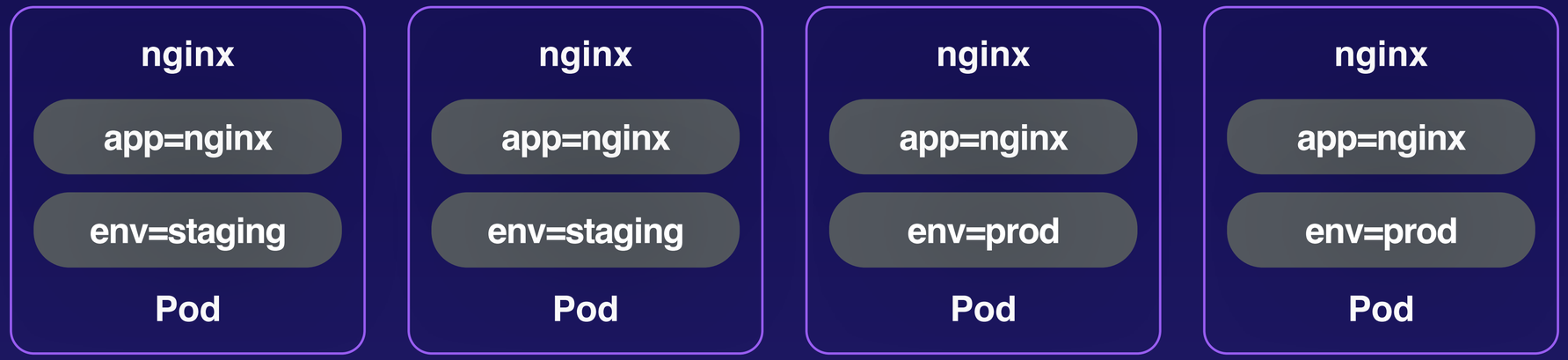
Accessing Applications


```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
 selector:
   app: nginx
 ports:
 - protocol: TCP
   port: 80
   targetPort: 80
```

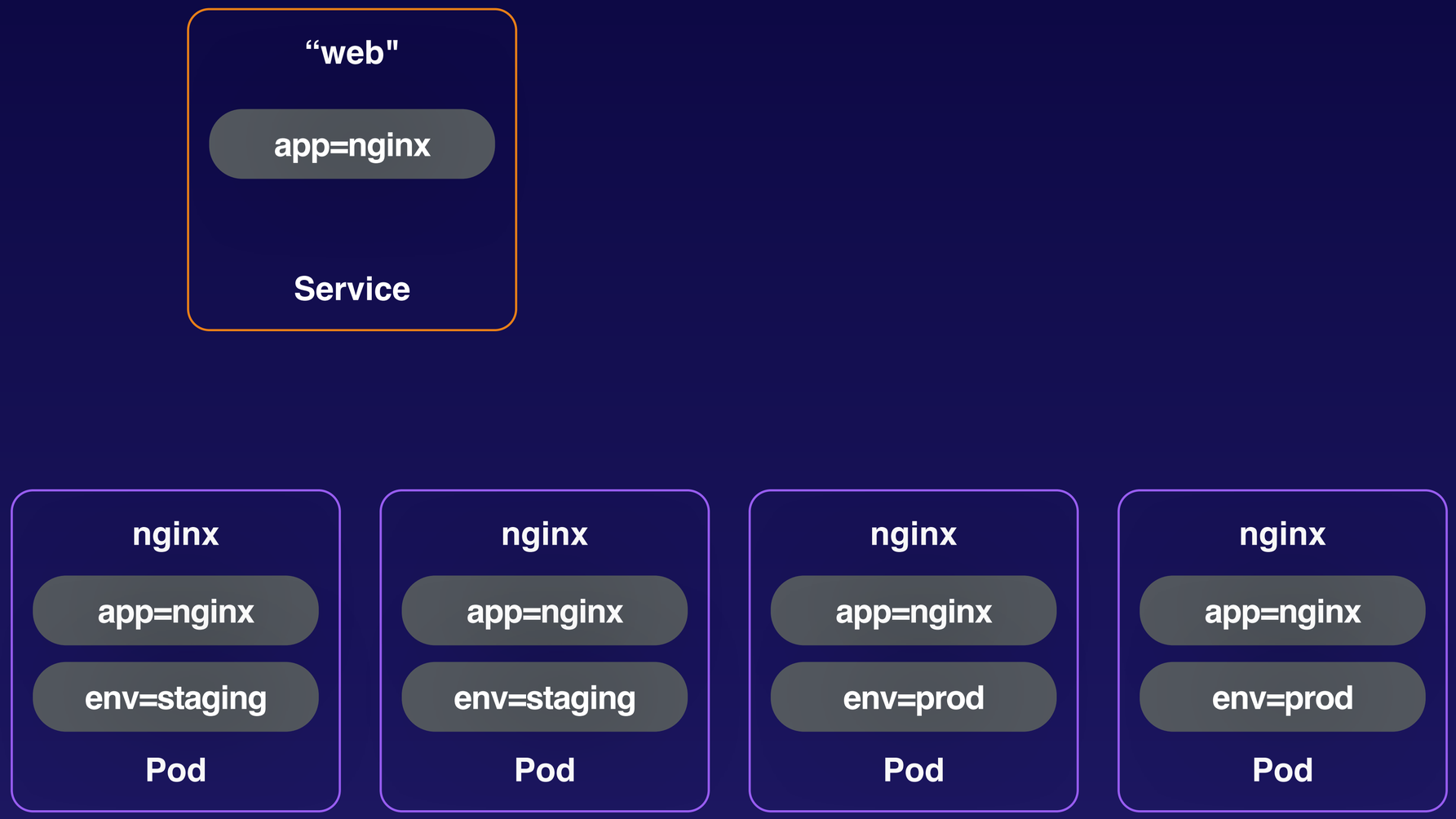
Accessing Applications


```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
 selector:
   app: nginx
 ports:
 - protocol: TCP
   port: 80
   targetPort: 80
```

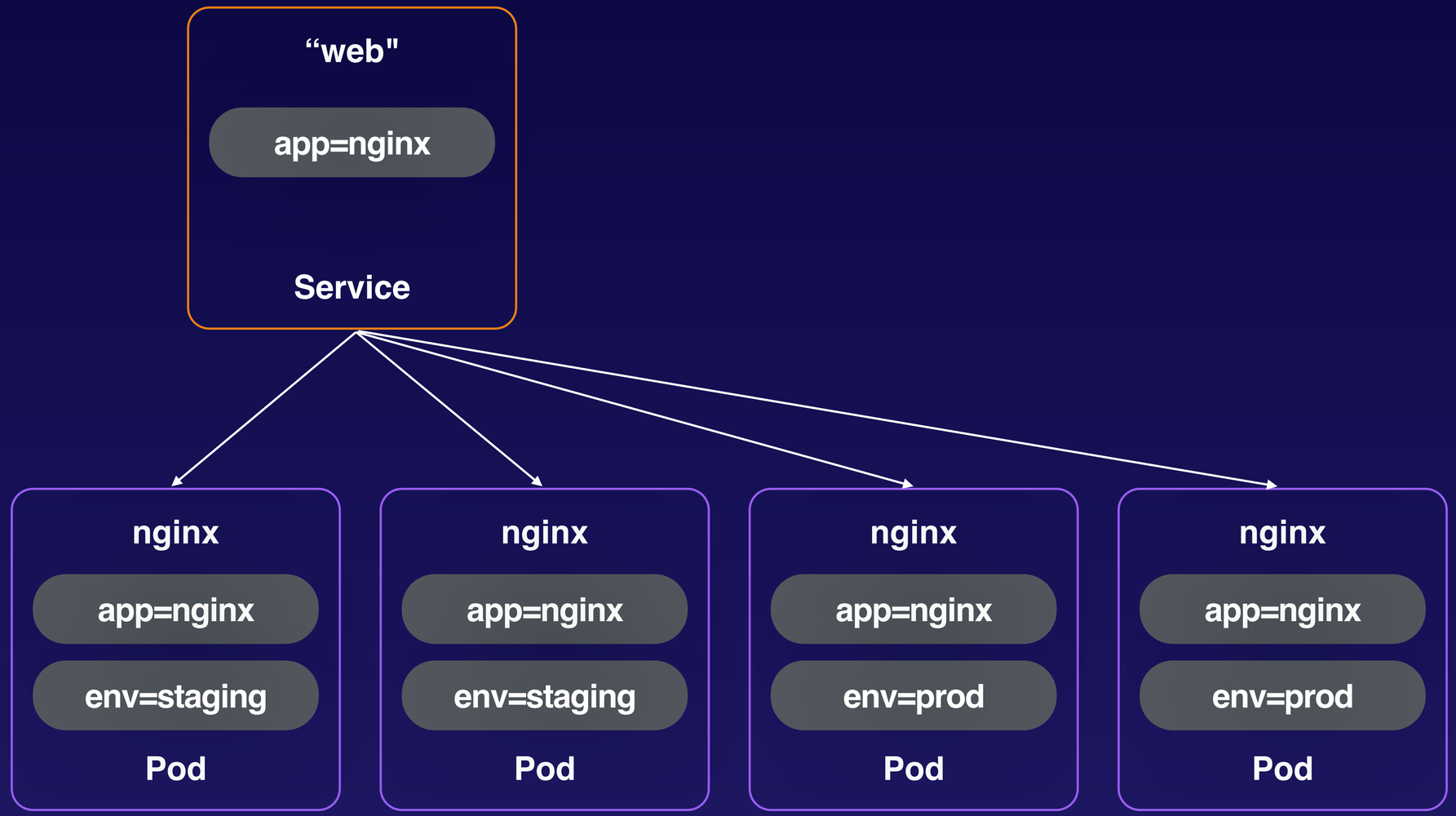
Accessing Applications



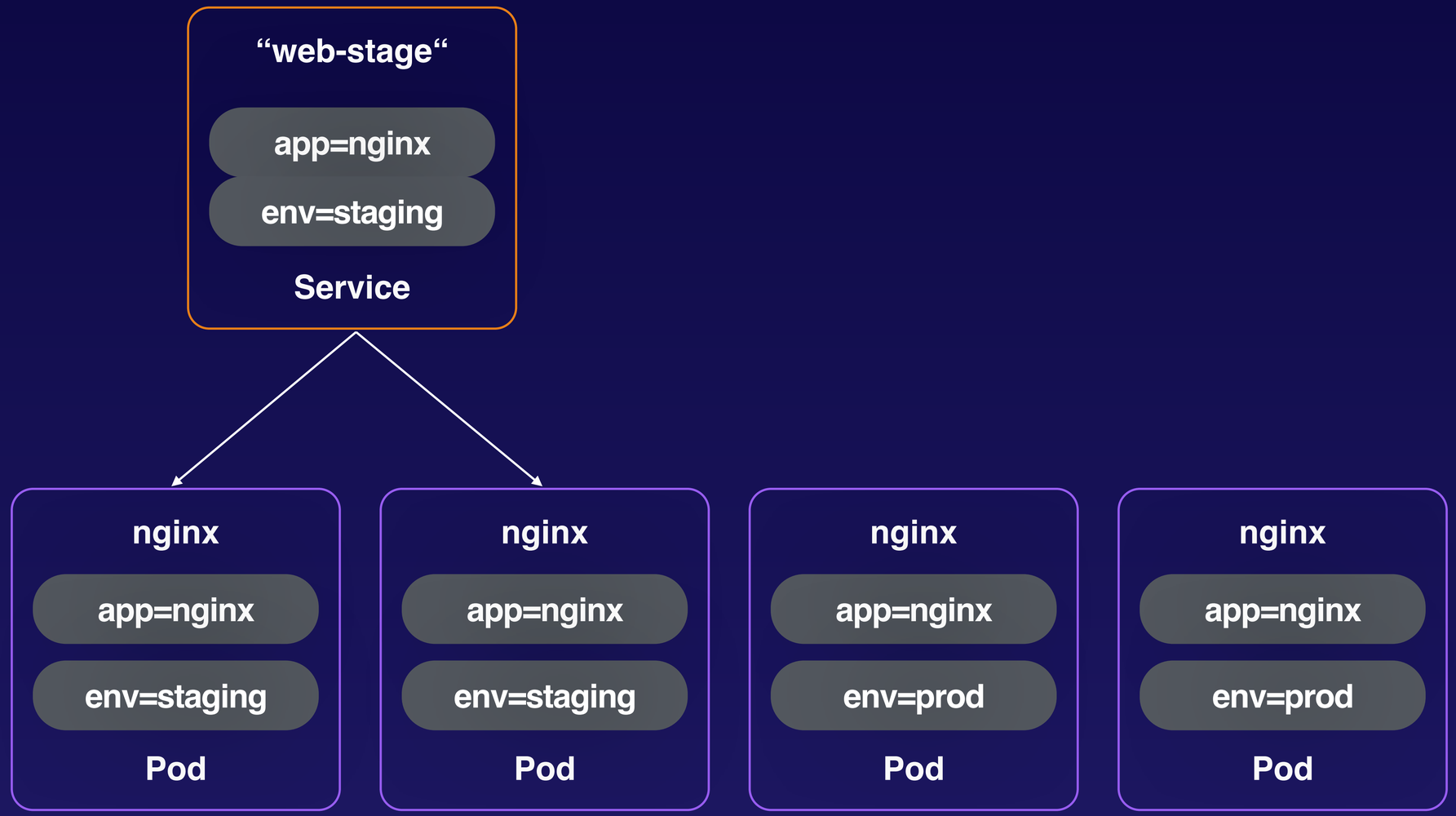
Accessing Applications



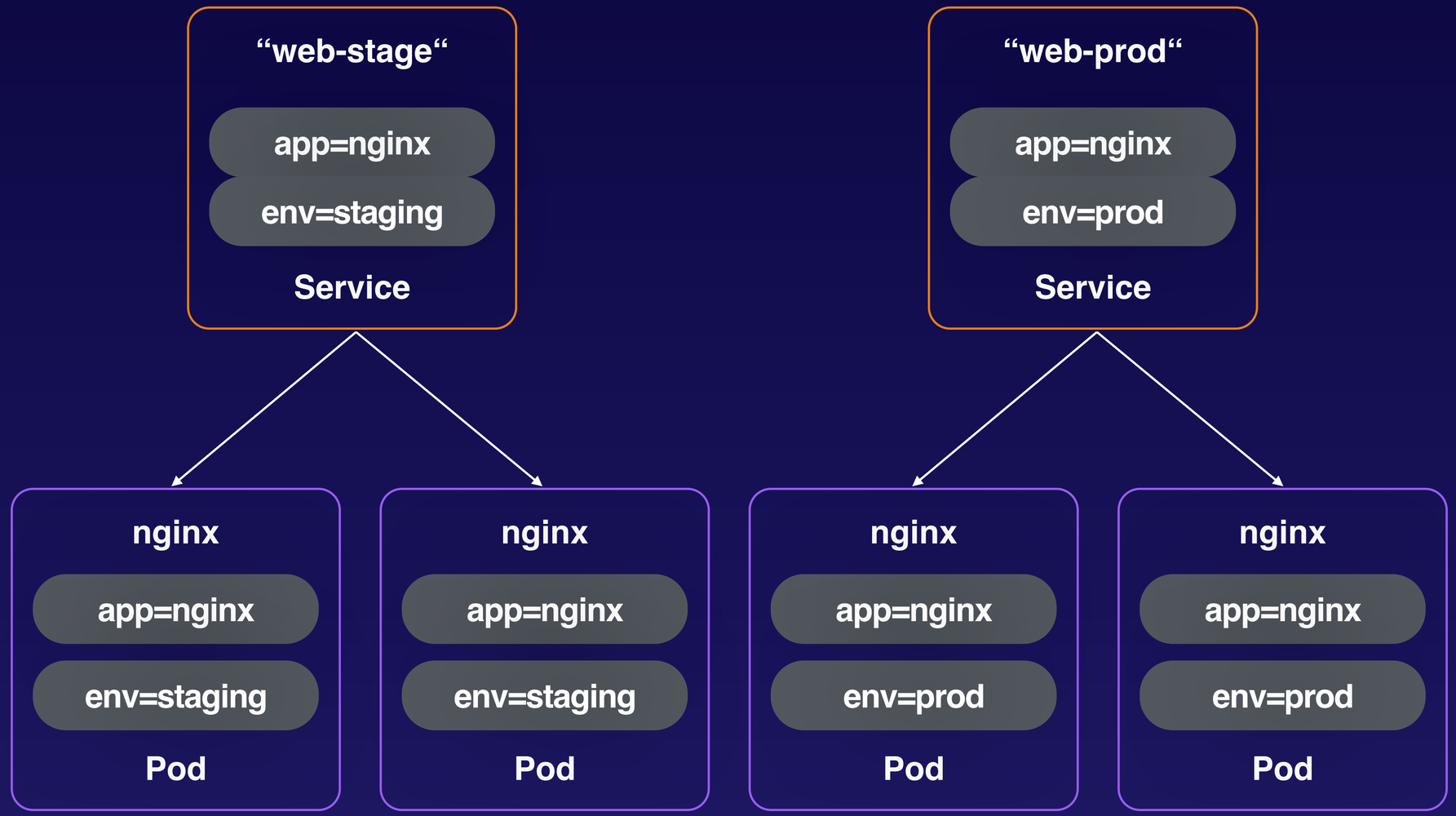
Accessing Applications



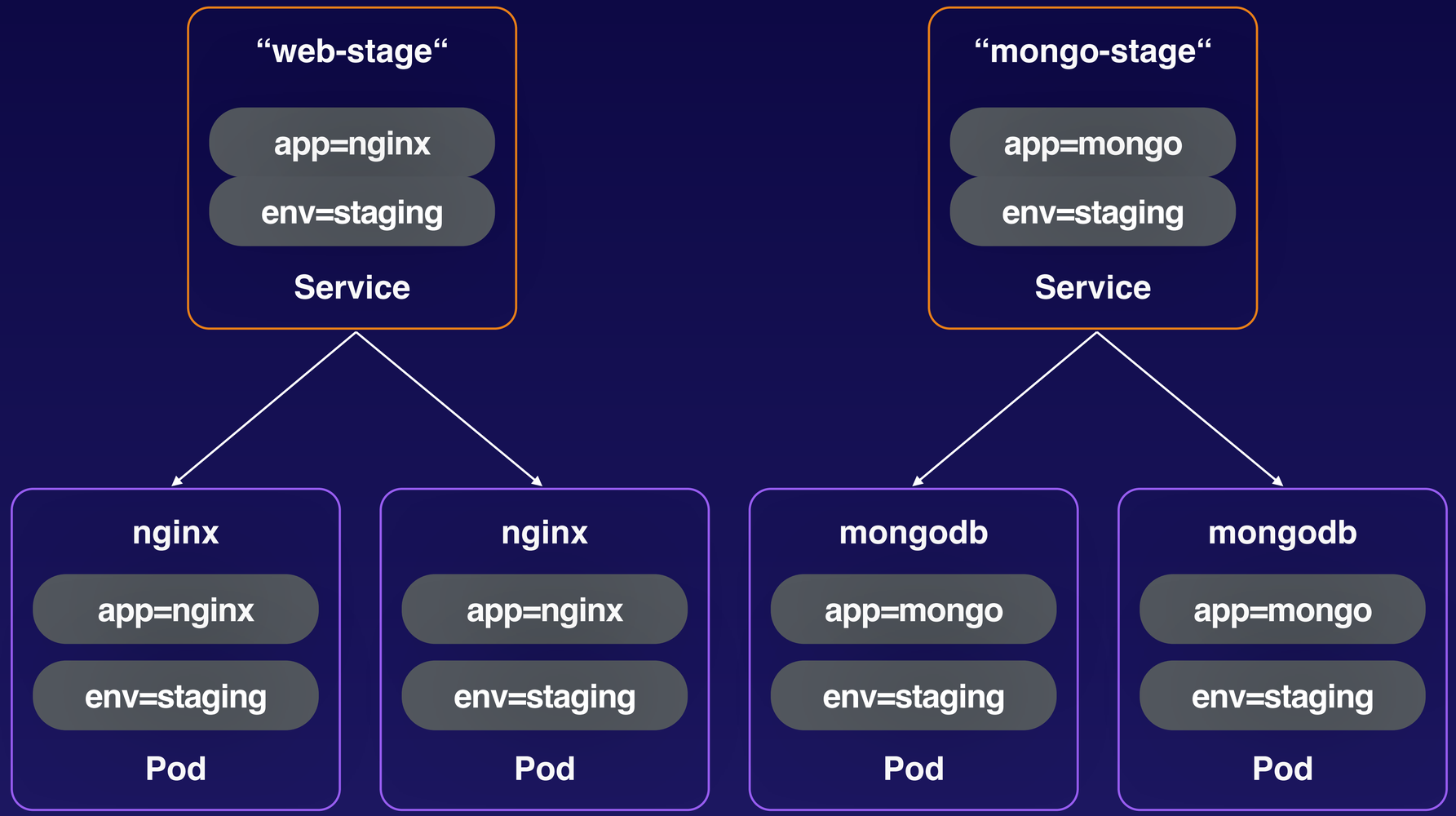
Accessing Applications



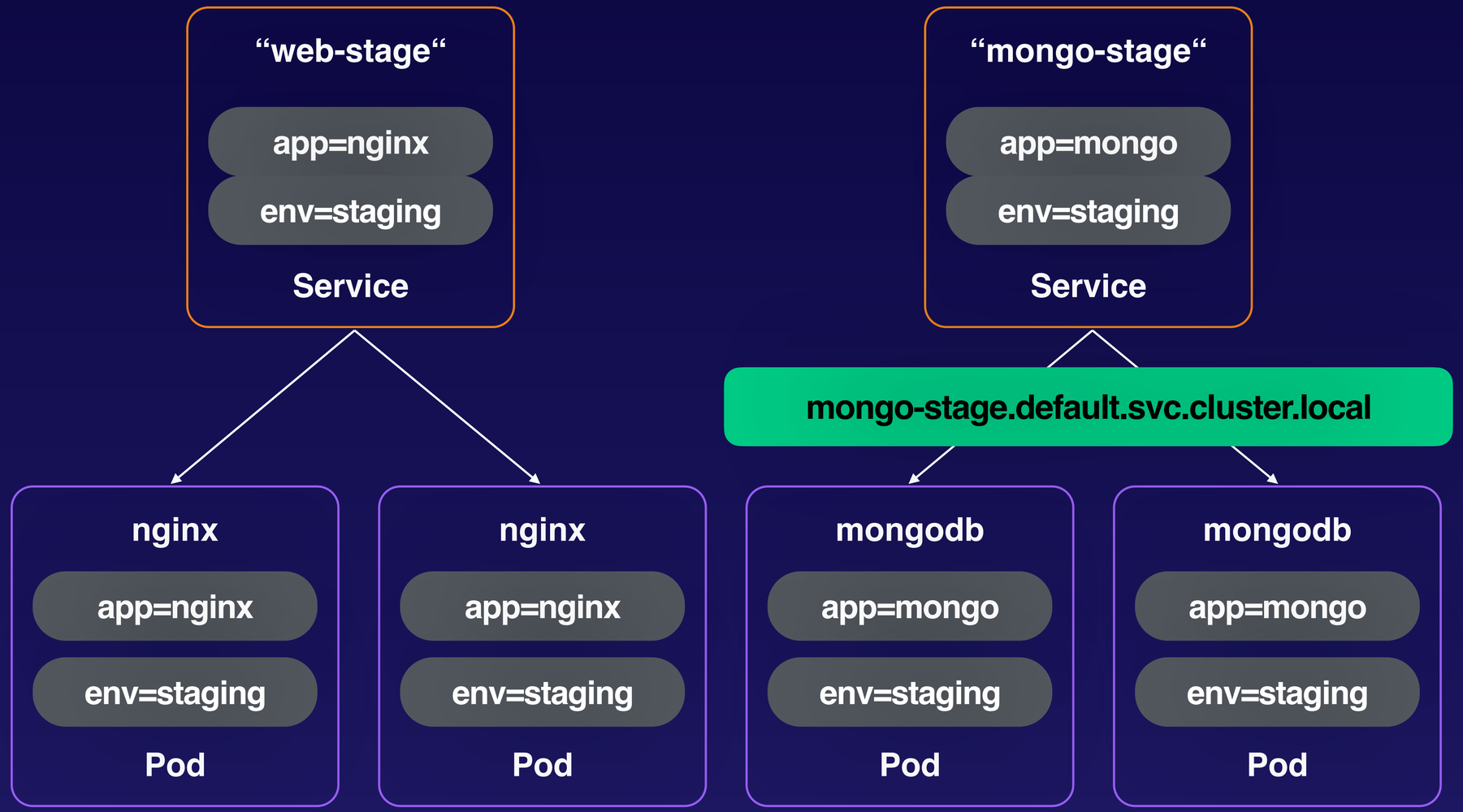
Accessing Applications



Accessing Applications



Accessing Applications



A CLOUD GURU

Lab time!

- Create a **Deployment** for and expose it via a **Service**
- View traffic as we **Rollout** and **Rollback** application changes
- Observe a **Deployment** dealing with **node errors**
- **Monitoring** and **logging**
- **Debugging** failed deployments

A CLOUD GURU

Lab: Deployments & Services

Tim Berry
GOOGLE CLOUD GURU

Lab: Deployments & Services

In this lab:

- Create a **Deployment** for our application
- Simulate a **failed node**
- **Rollout changes** to our application and **roll them back**

A CLOUD GURU

Lab: Monitoring, Logging and Debugging

Tim Berry
GOOGLE CLOUD GURU

Lab: Monitoring, Logging and Debugging

In this lab:

- **Monitoring** and **logging**
- **GCP Console** and **Stackdriver**
- **Debugging** failed deployments
- Common Pod **failures**

A CLOUD GURU

CHAPTER THREE

Deploying Applications

A CLOUD GURU

Putting it all together: Stateless application example

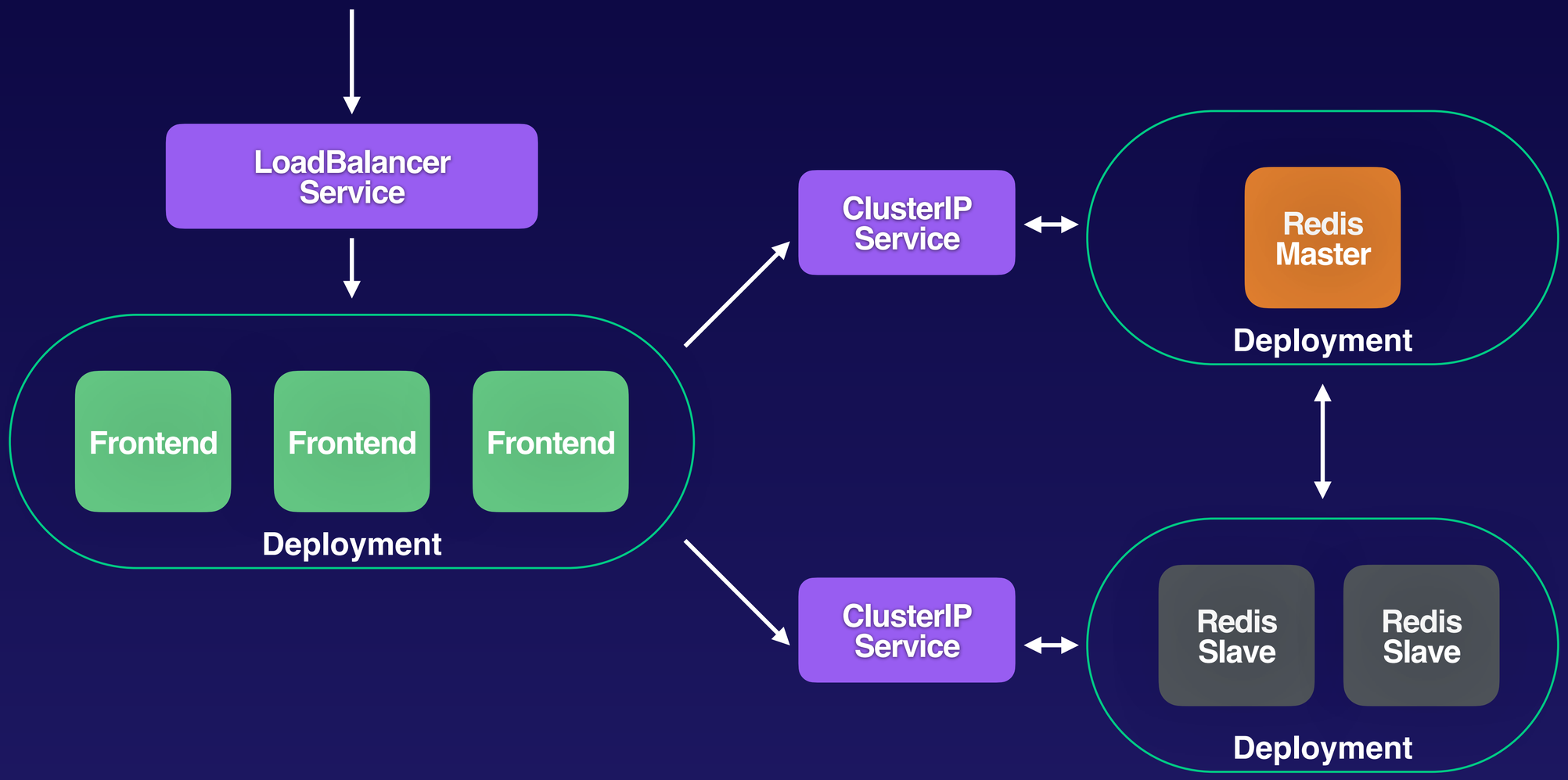
Tim Berry
GOOGLE CLOUD GURU

Stateless application example

This slide not shown

- Instead, screen-capture montage goes here, from:
  - https://kubernetes.io/docs/tutorials/stateless-application/guestbook
  - https://www.hover.com/blog/10-things-that-used-to-be-on-every-website-that-you-totally-forgot-about/

Stateless application example



Stateless application example

Lab objectives:

- Create a **standard 3-node** GKE cluster
- Use all **default** settings
- Authenticate **kubectl** with **gcloud**:

```
gcloud container clusters get-credentials <cluster-name> --zone=<zone>
```

A CLOUD GURU

Pod reliability with health checks

Tim Berry
GOOGLE CLOUD GURU

Pod reliability with health checks

Liveness Probes

- A check performed by **kubelet**
- Built-in **monitoring**
- Can check an **HTTP** endpoint, **TCP** socket, or run a **command**
- A **failed probe** will cause a Pod to be **replaced**

Pod reliability with health checks


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: liveness-test
spec:
  containers:
  - name: liveness-test
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 3
```

Pod reliability with health checks


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: liveness-test
spec:
  containers:
  - name: liveness-test
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 3
```

Readiness Probes

- A similar check performed by **kubelet**
- Define when a Pod is **ready to start** serving
- **Buys time** for Pods that must perform **startup tasks**
- Traffic **won't be directed** to a Pod until it is **ready**
- Use in **conjunction** with a **Liveness** Probe

Pod reliability with health checks


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: goproxy
  labels:
    app: goproxy
spec:
  containers:
  - name: goproxy
    image: k8s.gcr.io/goproxy:0.1
    ports:
    - containerPort: 8080
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 30
    livenessProbe:
      tcpSocket:
        port: 8080
     periodSeconds: 10
```

Pod reliability with health checks


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: goproxy
  labels:
    app: goproxy
spec:
  containers:
  - name: goproxy
    image: k8s.gcr.io/goproxy:0.1
    ports:
    - containerPort: 8080
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 30
    livenessProbe:
      tcpSocket:
        port: 8080
     periodSeconds: 10
```

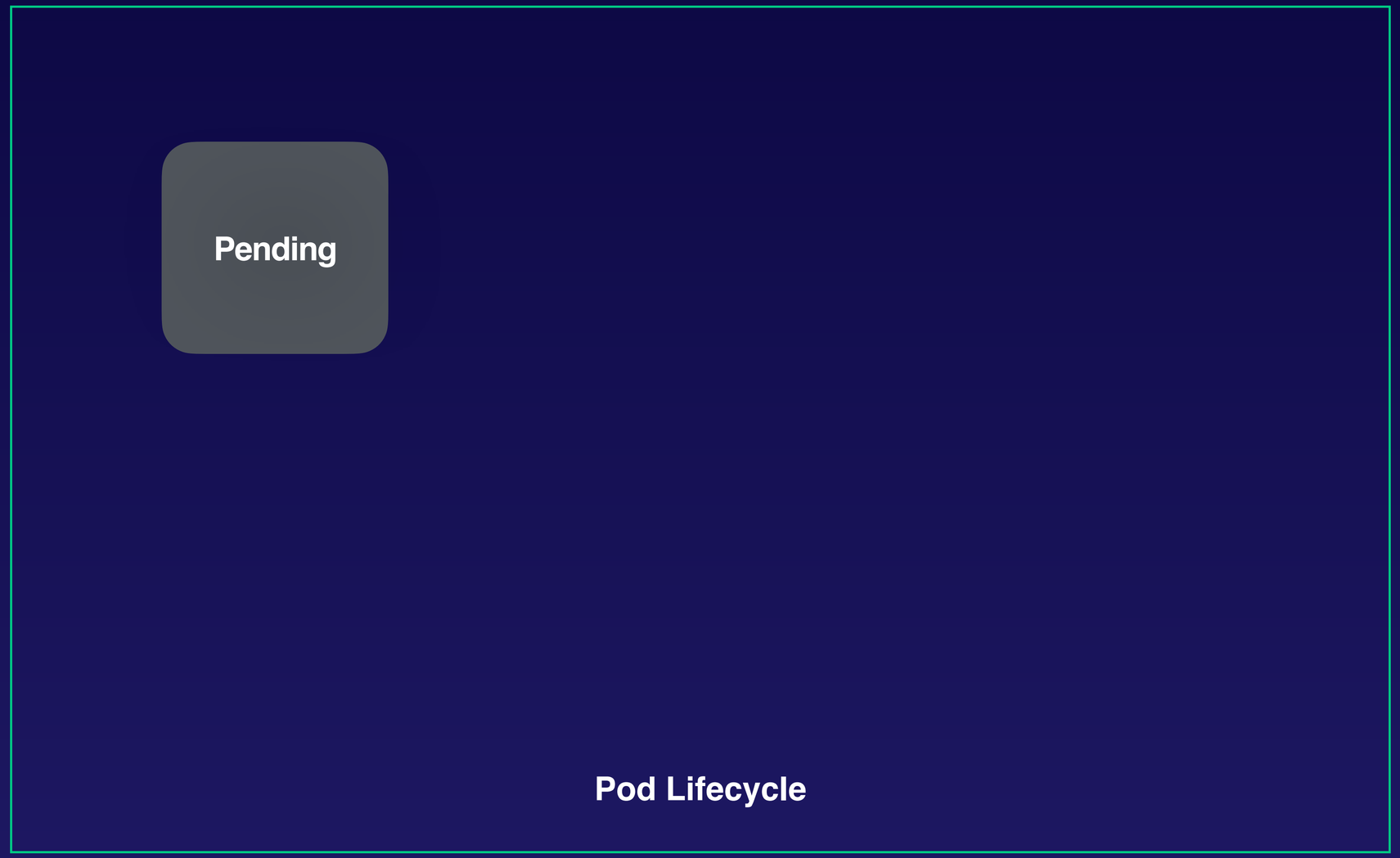
Pod reliability with health checks


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: goproxy
  labels:
    app: goproxy
spec:
  containers:
  - name: goproxy
    image: k8s.gcr.io/goproxy:0.1
    ports:
    - containerPort: 8080
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 30
    livenessProbe:
      tcpSocket:
        port: 8080
     periodSeconds: 10
```

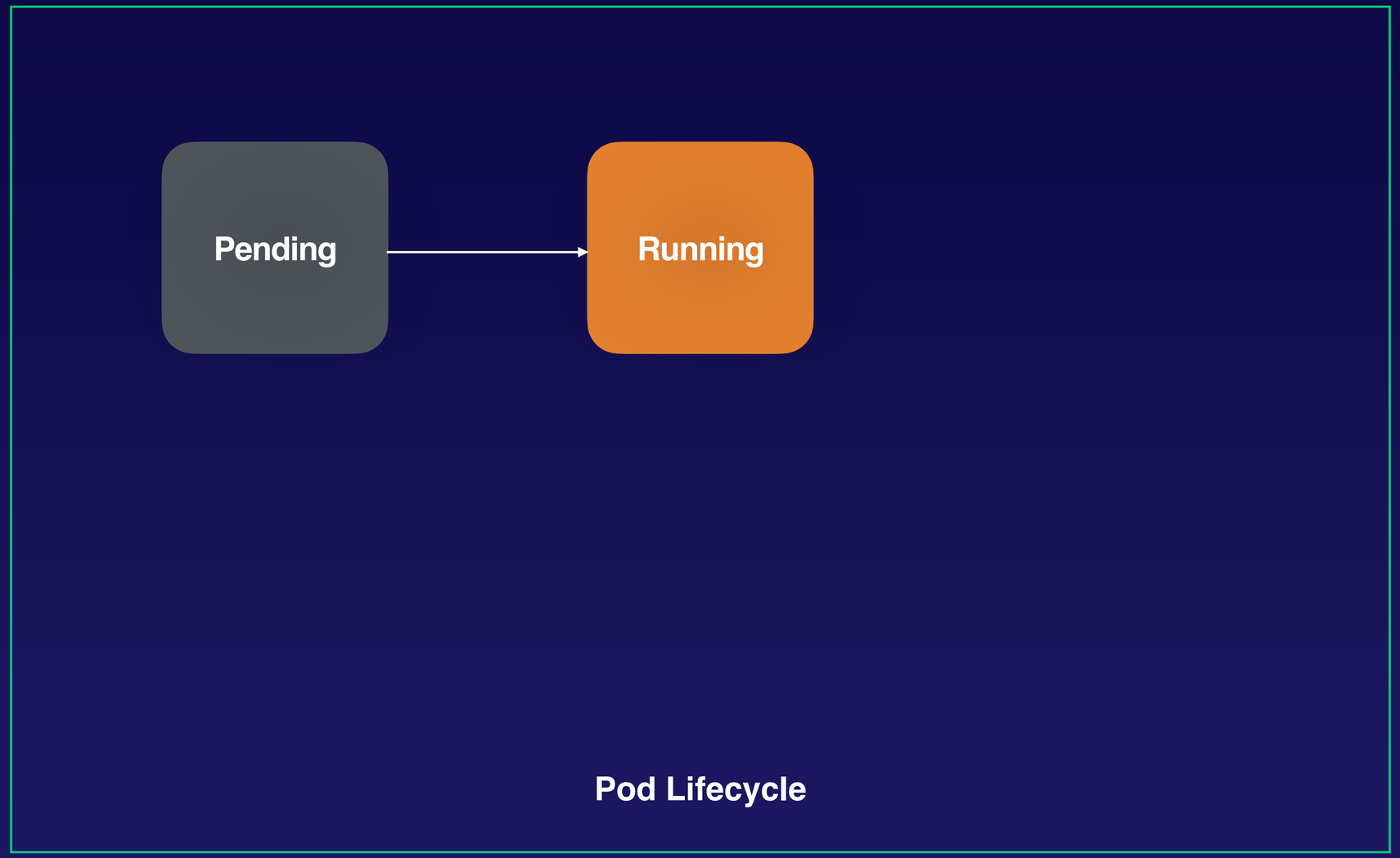
Probe Configuration

- **Probes** are performed by a **handler**:
  - ExecAction
  - TCPSocketAction
  - HTTPGetAction
- initialDelaySeconds, periodSeconds, timeoutSeconds
- successThreshold, failureThreshold

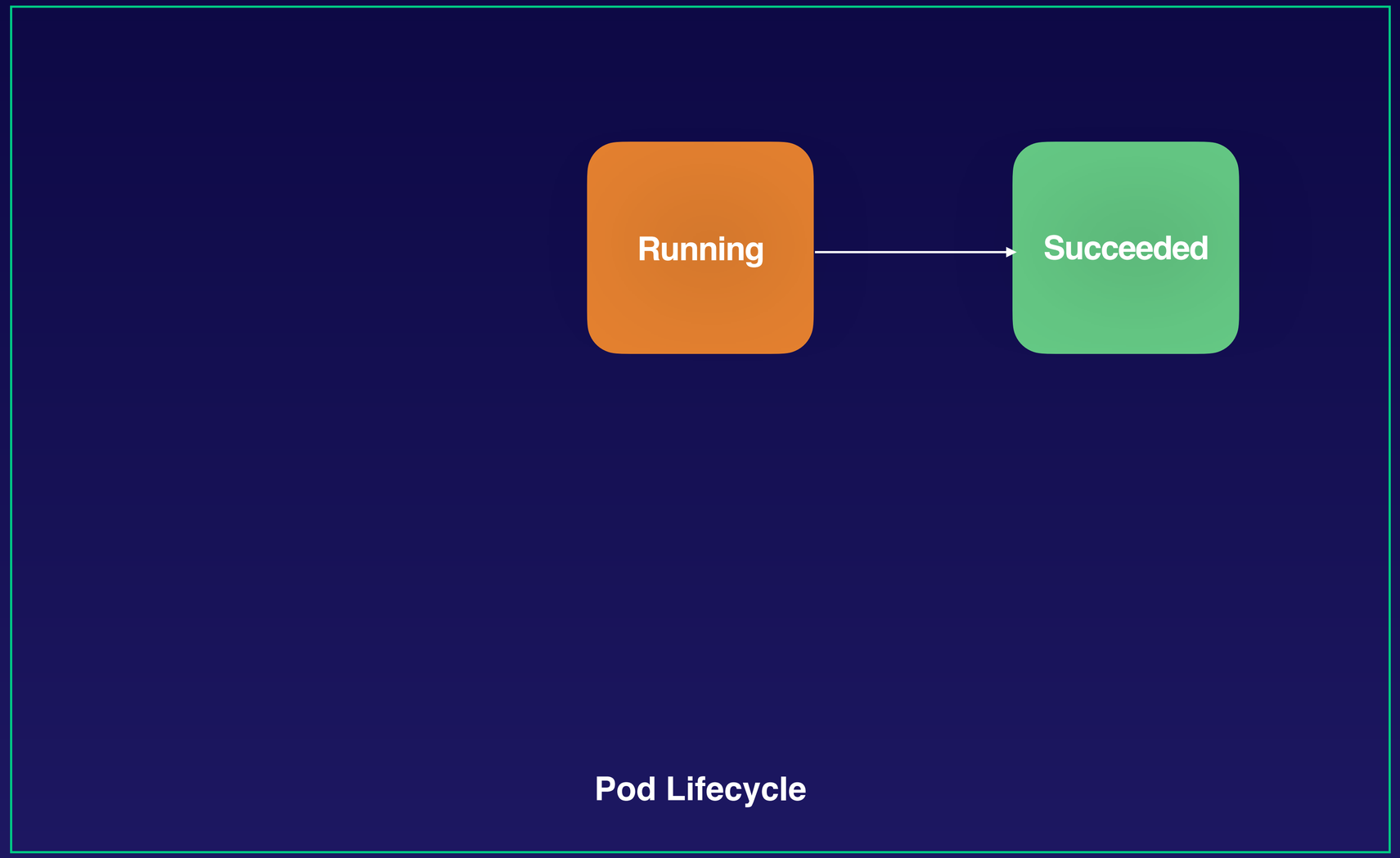
Pod reliability with health checks



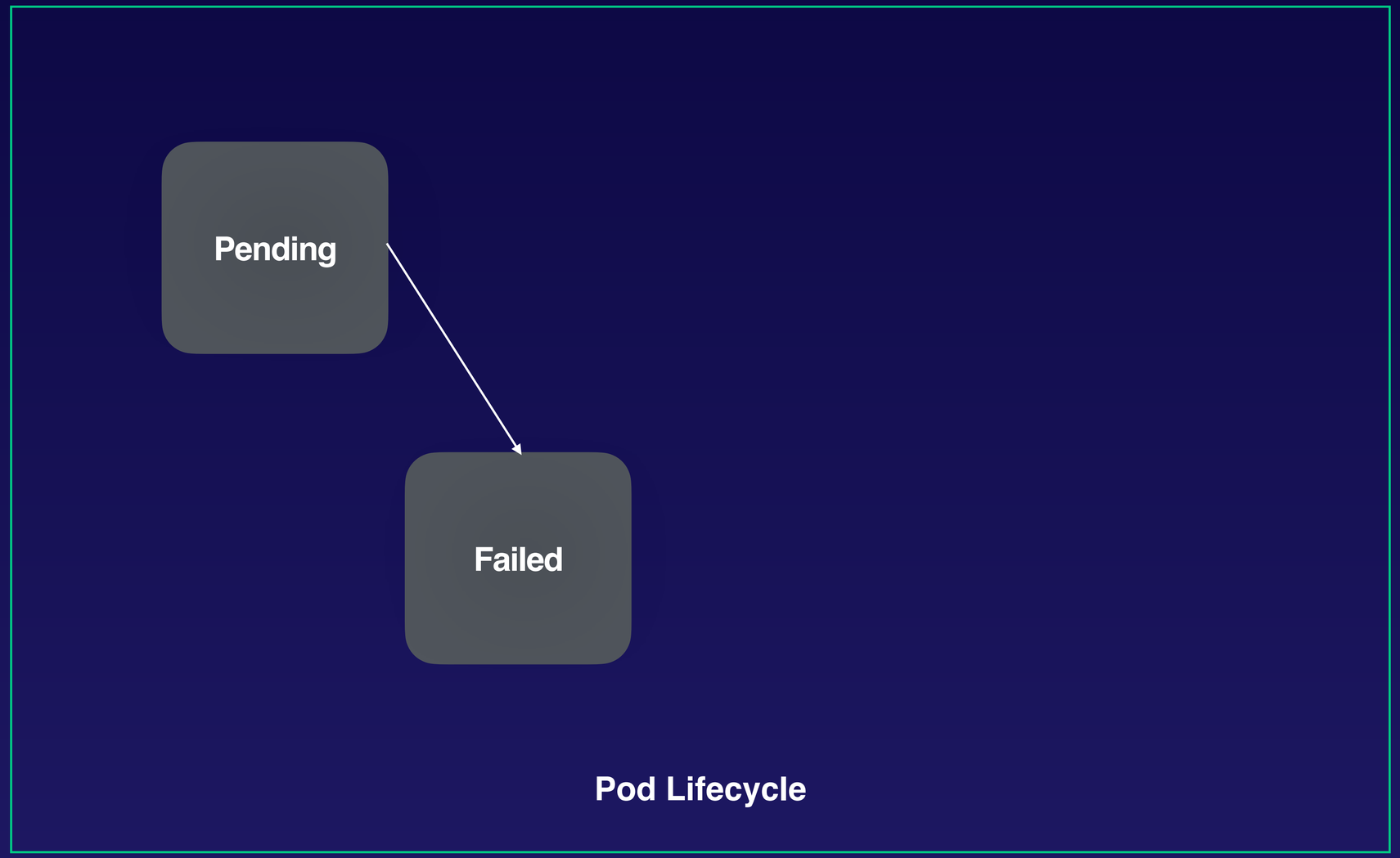
Pod reliability with health checks



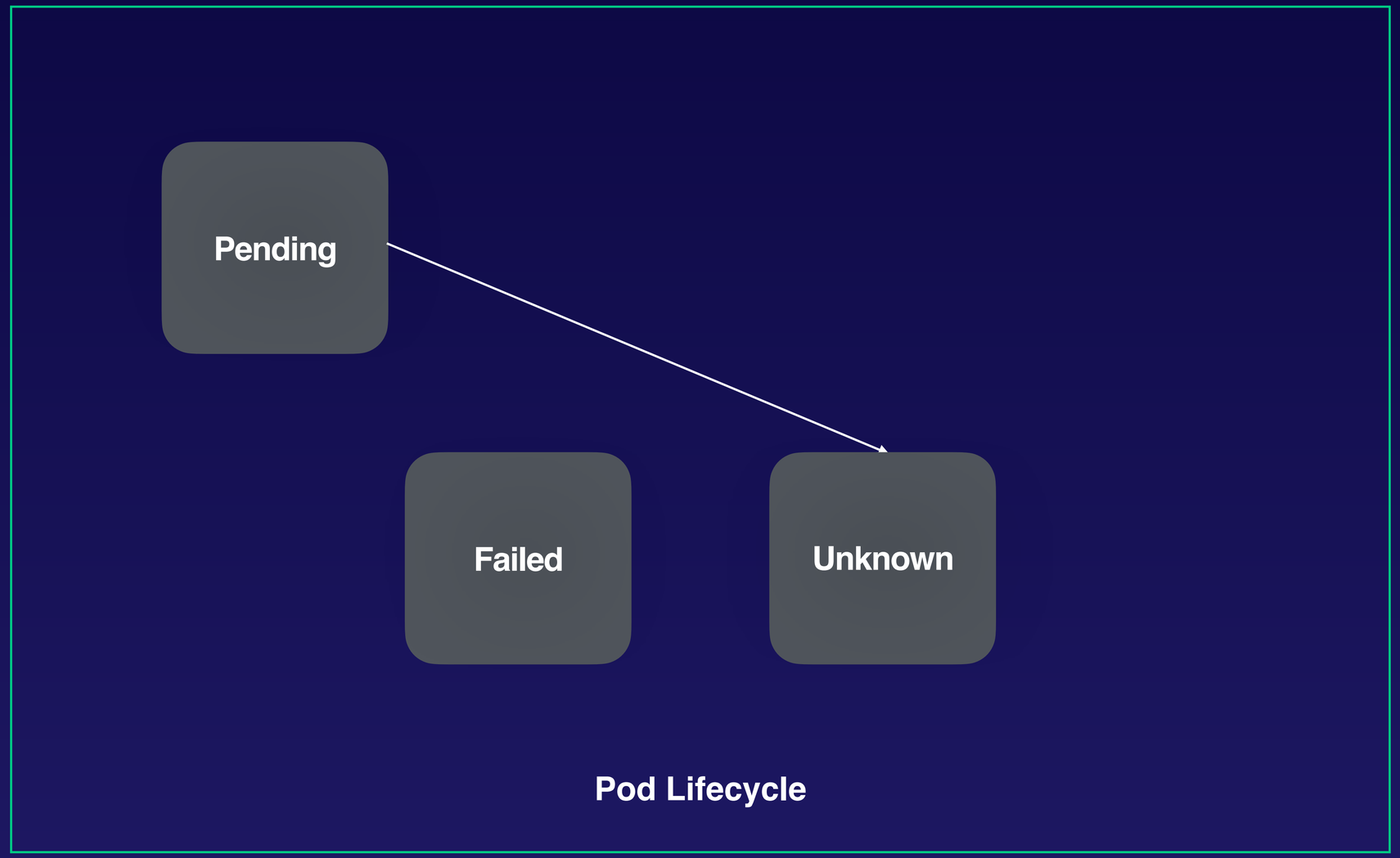
Pod reliability with health checks



Pod reliability with health checks



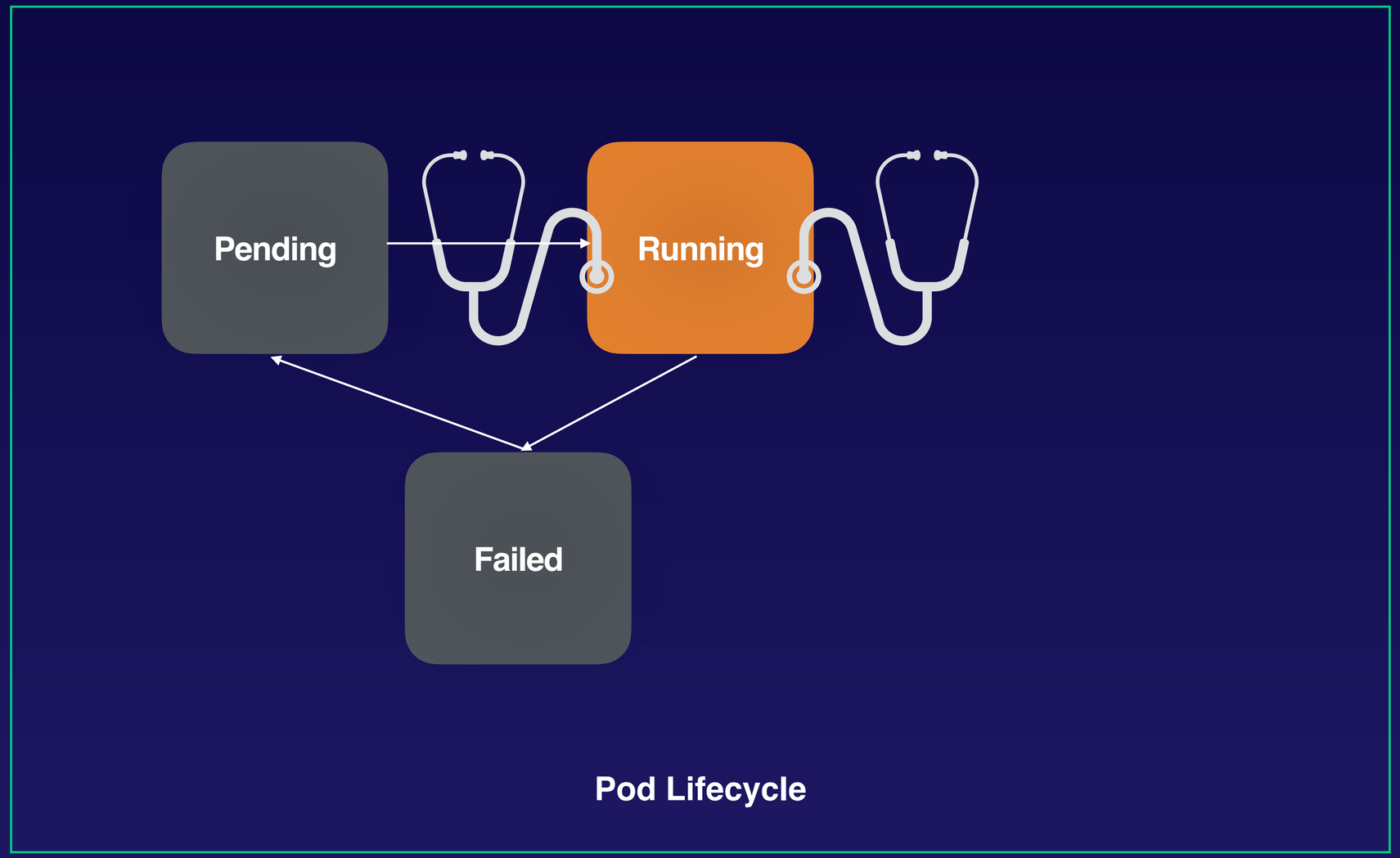
Pod reliability with health checks



Pod reliability with health checks

Pod Lifecycle

Pod reliability with health checks



A CLOUD GURU

Accessing external services

Tim Berry
GOOGLE CLOUD GURU

A CLOUD GURU

Service Endpoints

- Create a **Service** object with **no Selector**
- Kubernetes looks for an **Endpoint** with the same **name**
- Endpoints map **external services** to Service objects
- Service discovery via **internal DNS**

Accessing external services


```yaml
apiVersion: v1
kind: Service
  metadata: mongo
spec:
  type: ClusterIP
  ports:
  - port: 27017
    targetPort: 27017
```

no selector!

Accessing external services


```yaml
apiVersion: v1
kind: Service
  metadata: mongo
spec:
  type: ClusterIP
  ports:
  - port: 27017
    targetPort: 27017
```

```yaml
apiVersion: v1
kind: Endpoints
metadata:
  name: mongo
subsets:
  - addresses:
    - ip: 10.240.0.4
    ports:
    - port: 27017
```

Accessing external services


```yaml
apiVersion: v1
kind: Service
  metadata: mongo
spec:
  type: ClusterIP
  ports:
  - port: 27017
    targetPort: 27017
```

```yaml
apiVersion: v1
kind: Endpoints
metadata:
  name: mongo
subsets:
  - addresses:
    - ip: 10.240.0.4
    ports:
    - port: 27017
```

mongodb://mongo.default.svc.cluster.local

Accessing external services


```yaml
apiVersion: v1
kind: Service
  metadata: mongo
spec:
  type: ExternalName
  externalName: db12.mymdb.com
```

- Services can also map to **external FQDNs**
- Use **ExternalName** type
- No **EndPoints** required

mongodb://mongo.default.svc.cluster.local

Accessing external services


```yaml
apiVersion: v1
kind: Endpoints
metadata:
  name: rabbitmq
subsets:
  - addresses:
    - ip: 10.245.0.1
    ports:
    - port: 31000
  - addresses:
    - ip: 10.245.0.2
    ports:
    - port: 31000
```

- **Endpoints** can map to **multiple IPs**
- The Service will **round-robin** each IP

A CLOUD GURU

Sidecars

- **Sidecar** provides the **connection** to the external service
- Application container accesses the service on **localhost** / **127.0.0.1**
- Useful for proxies, such as **MySQL Proxy** for **Cloud SQL**

A CLOUD GURU

Lab: Maintaining a service with unhealthy pods

Tim Berry
GOOGLE CLOUD GURU

Lab: Maintaining a service with unhealthy pods

Lab objectives:

- Create a **standard 3-node** GKE cluster
- Authenticate **kubectl** with **gcloud**
- Create a **deployment** with a **sidecar container** and **health checks**
- **Update** a deployment with a **bad change**
- **Readiness checks** will keep our service running

A CLOUD GURU

Volumes and persistent storage

Tim Berry
GOOGLE CLOUD GURU

A CLOUD GURU

Volumes

- Container storage is **ephemeral**
- A **volume** is an independent **object**
- A **directory** mounted in a container to access files
- Many different **types** - some of which are **persistent**

Volumes and persistent storage

emptyDir

Scratch-space that can be shared by multiple containers in the same Pod. Deleted forever when a Pod is removed from a node.

gcePersistentDisk

A GCP persistent disk. Must be created beforehand, can be pre-populated with data, and mounted read-only by multiple consumers. Will only be unmounted when a Pod is removed.

PersistentVolumeClaim

Used to mount a PersistentVolume into a Pod - a way to “claim” durable storage. Requires a matching PersistentVolume object.

A CLOUD GURU

Persistent Volumes

- **PersistentVolume** defines a piece of storage in the cluster
- Manually or dynamically provisioned using **plugins**
- Configured with a **Storage Class**
- **PersistentVolumeClaim** is a request to consume a **PersistentVolume**
- Access Modes define how a volume may be accessed by multiple consumers

Volumes and persistent storage


```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: myapp-pv
spec:
  storageClassName: ""
  capacity:
    storage: 10G
  accessModes:
    - ReadOnlyMany
  gcePersistentDisk:
    pdName: myapp-disk
    fsType: ext4
```

- Manually define a PersistentVolume using an existing GCP persistent disk
- Persistent disk must contain a filesystem

Volumes and persistent storage


```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myapp-pvc
spec:
  accessModes:
    - ReadOnlyMany
  resources:
    requests:
      storage: 10G
```

- Manually define a PersistentVolumeClaim
- Kubernetes matches the claim to an available PersistentVolume object

Volumes and persistent storage


```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myapp-dynamic
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 30Gi
```

- Alternatively - make a claim without creating a volume beforehand
- GKE dynamically creates a disk and volume using the Storage Class provisioner

Volumes and persistent storage


```yaml
volumes:
- name: myapp-vol
  persistentVolumeClaim:
    claimName: myapp-dynamic
    readOnly: true
```

- Consume the volume claim when defining volumes in a Pod spec

Volumes and persistent storage


```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gcp-ssd
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
  replication-type: none
```

- GKE default storage class uses standard persistent disks
- Custom Storage Class objects change the parameters

Access Modes

- **ReadWriteOnce** - a single node can mount the volume **read/write**
- **ReadOnlyMany** - any node can mount the volume **read-only**
- **ReadWriteMany** - any node can mount the volume **read/write**
  - Note: This mode is **not supported** by GCP persistent disks

Constraints

- **Deployments** are designed for **stateless** applications
- All replicas will share the same **PersistentVolumeClaim**
- Volume mode must be **ReadOnlyMany**
- Deployment strategy may cause volume **deadlocks**

A CLOUD GURU

Alternatives

- Abstract away your data!
- **Object storage** (GCS) and managed **databases**
- Separate file service in **GKE** or **GCE**
- Cloud Filestore

A CLOUD GURU

Lab: Volumes and persistent storage

Tim Berry
GOOGLE CLOUD GURU

Lab: Volumes and persistent storage

Lab objectives:

- Create a **standard 3-node** GKE cluster
- Authenticate **kubectl** with **gcloud**
- Create **dynamic** persistent volume claims
- Deploy **MySQL** and **Wordpress** Pods
- Test data **persistence** on Pod **failure**

Caveats

- **Deployments** are designed for **stateless** applications
- When using a **persistent disk** you can only run a **single Pod**
- **Stateful** workloads are covered in the **next chapter!**

A CLOUD GURU

ConfigMaps and Secrets

Tim Berry
GOOGLE CLOUD GURU

Secrets

- Designed to **obfuscate** the storage of **sensitive data**
- Consumed as **environment variables** or **volumes**
- Secrets are **encoded** only - **not encrypted** by default
- Secrets can be encrypted with **Cloud KMS** (beta)

A CLOUD GURU

ConfigMaps

- **Decouple** configuration from image content
- Created from **files**, **directories** or **literal values**
- Values can be referenced as **environment** variables
- ConfigMap data may also be mounted as a **Volume**

ConfigMaps and Secrets


```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: myconfig
data:
  make: ford
  model: prefect
  color: blue
```

- Basic ConfigMaps can be defined in YAML

ConfigMaps and Secrets


```yaml
containers:
  - name: my-container
    image: my-image
    env:
      - name: CAR_MAKE
        valueFrom:
          configMapKeyRef:
            name: myconfig
            key: make
      - name: CAR_MODEL
        valueFrom:
          configMapKeyRef:
            name: myconfig
            key: model
```

- Keys and values can be referenced and passed to environment variables inside the container
- Use individual keys to assign values

ConfigMaps and Secrets


```
containers:
  - name: my-container
    image: my-image
    envFrom:
    - configMapRef:
        name: myconfig
```

- Alternatively map all keys and values as environment variables inside a container
- For convention, key names should be CAPITALIZED

ConfigMaps and Secrets


```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-nginx-conf
data:
  nginx.conf: |
    user nginx;
    worker_processes 3;
    server {
      listen 80
      server name _;
      location / {

      etc. etc.
```

- ConfigMaps can also store standard configuration files
- Anything after the I is treated as a string

ConfigMaps and Secrets


```yaml
containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - name: nginx-config
      mountPath: /etc/nginx
volumes:
  - name: nginx-config
    configMap:
      name: my-nginx-config
```

- ConfigMaps are then used as Volume objects
- They are mounted in the container filesystem in the same way as any other Volume

A CLOUD GURU

Deployment patterns

Tim Berry
GOOGLE CLOUD GURU

A CLOUD GURU

Stateless Application Updates

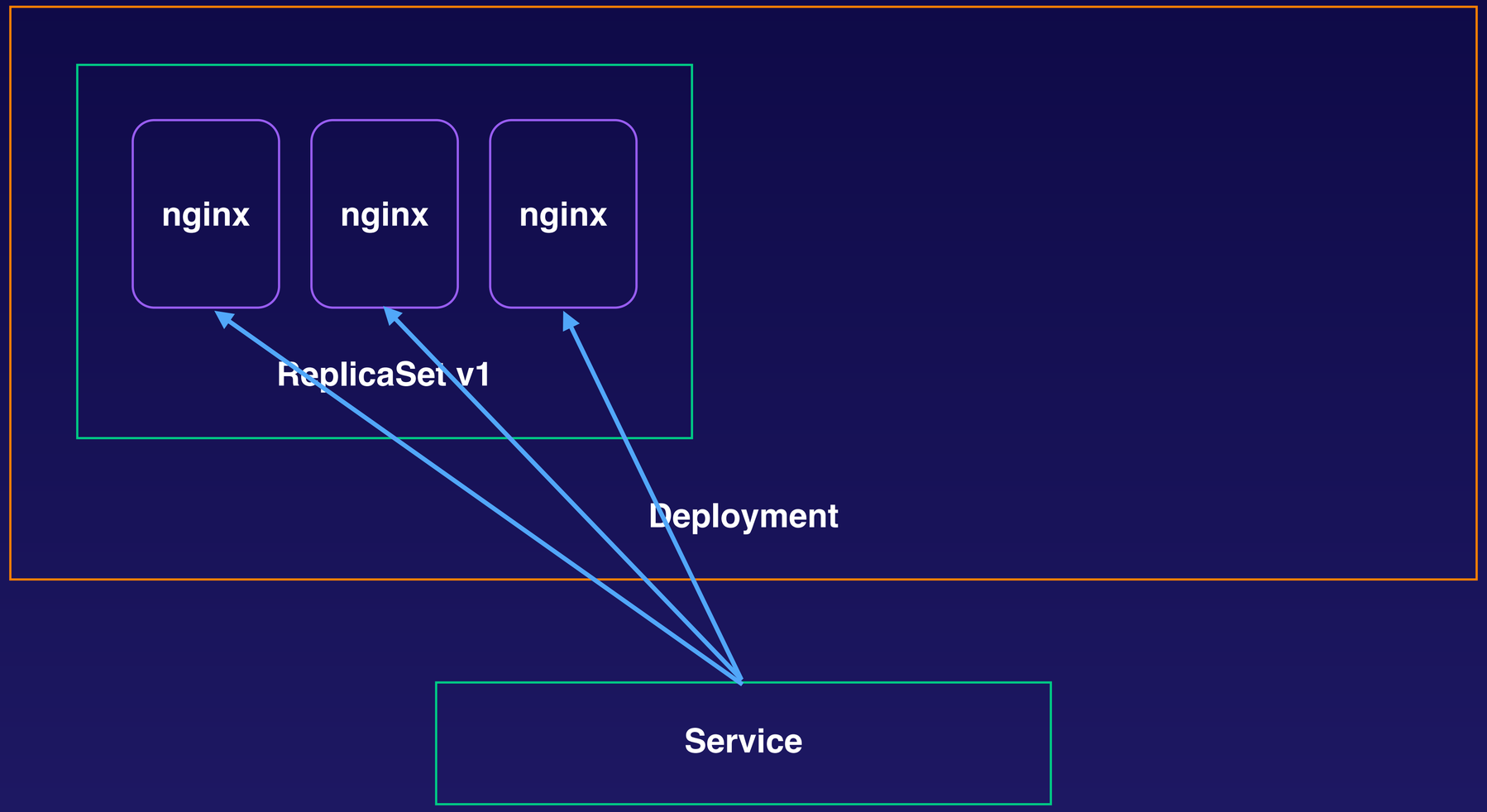
- **Rolling** Updates
- **Canary** Deployments
- **Blue-Green** Deployments

Rolling Updates

- Gradually **replace** Pods with an **updated spec**
- **Control** how many **additional** Pods may be created
- Specify **threshold** for **failed** Pods

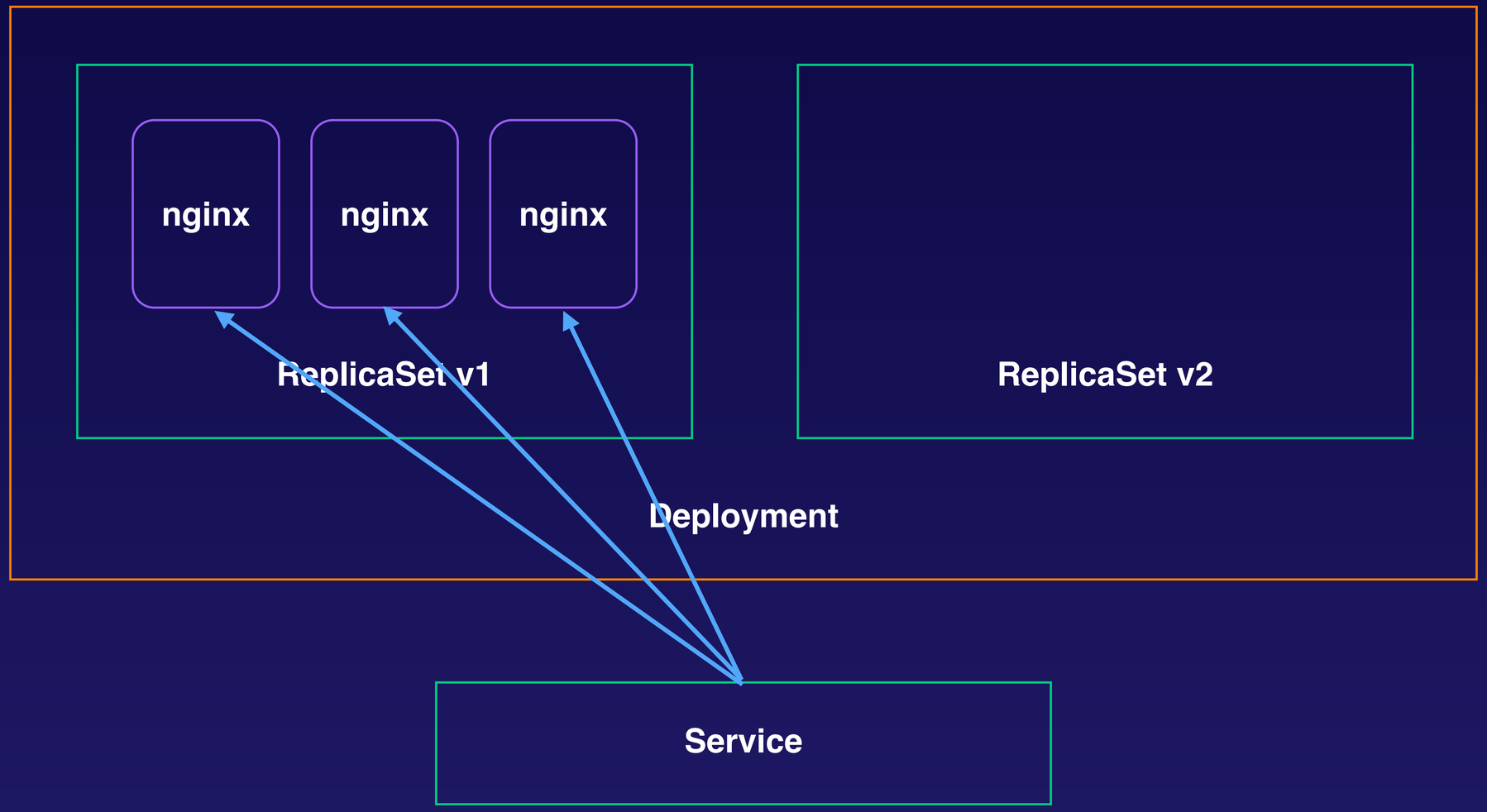
Deployment Patterns

ROLLING UPDATE



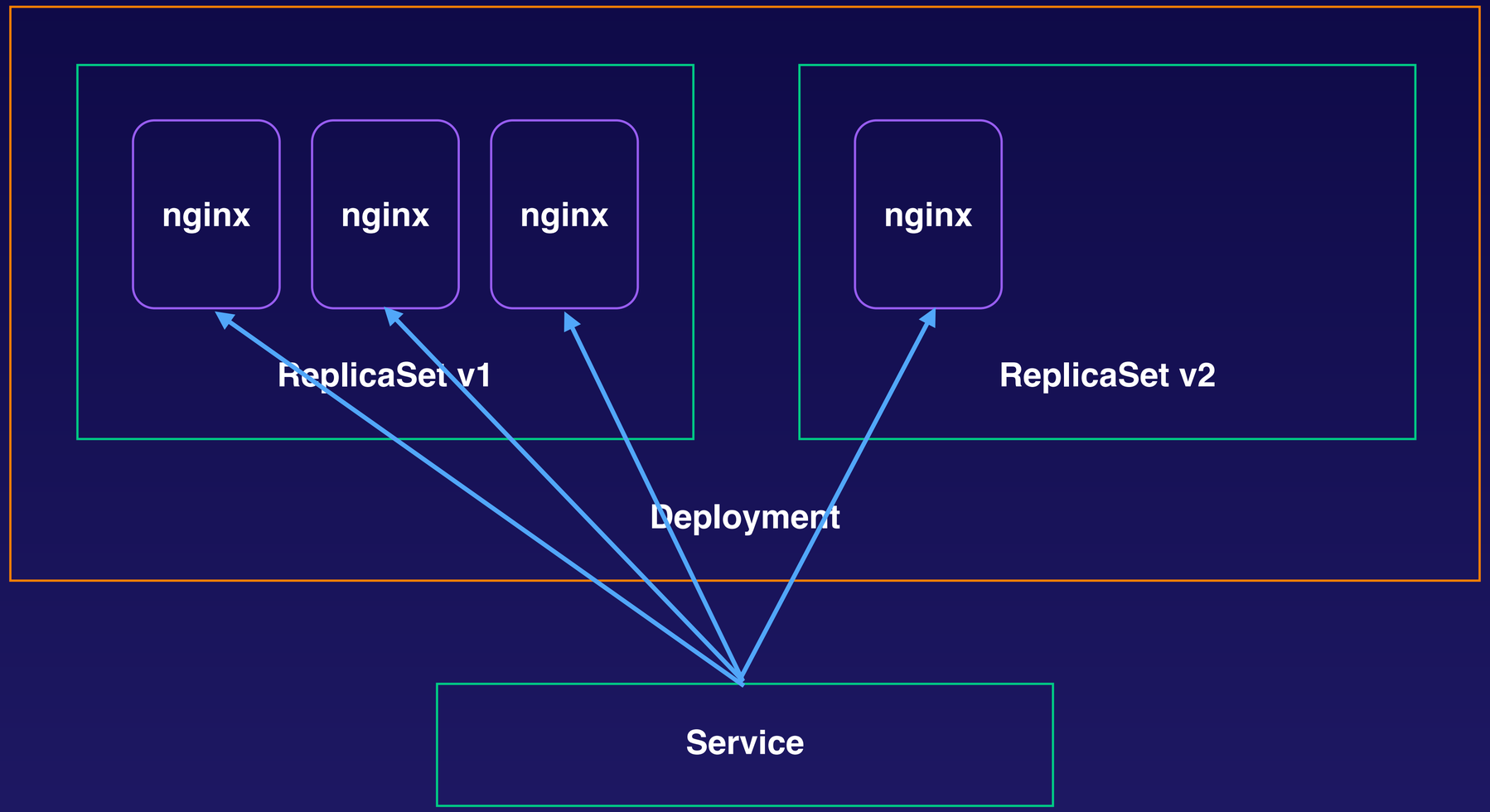
Deployment Patterns

ROLLING UPDATE



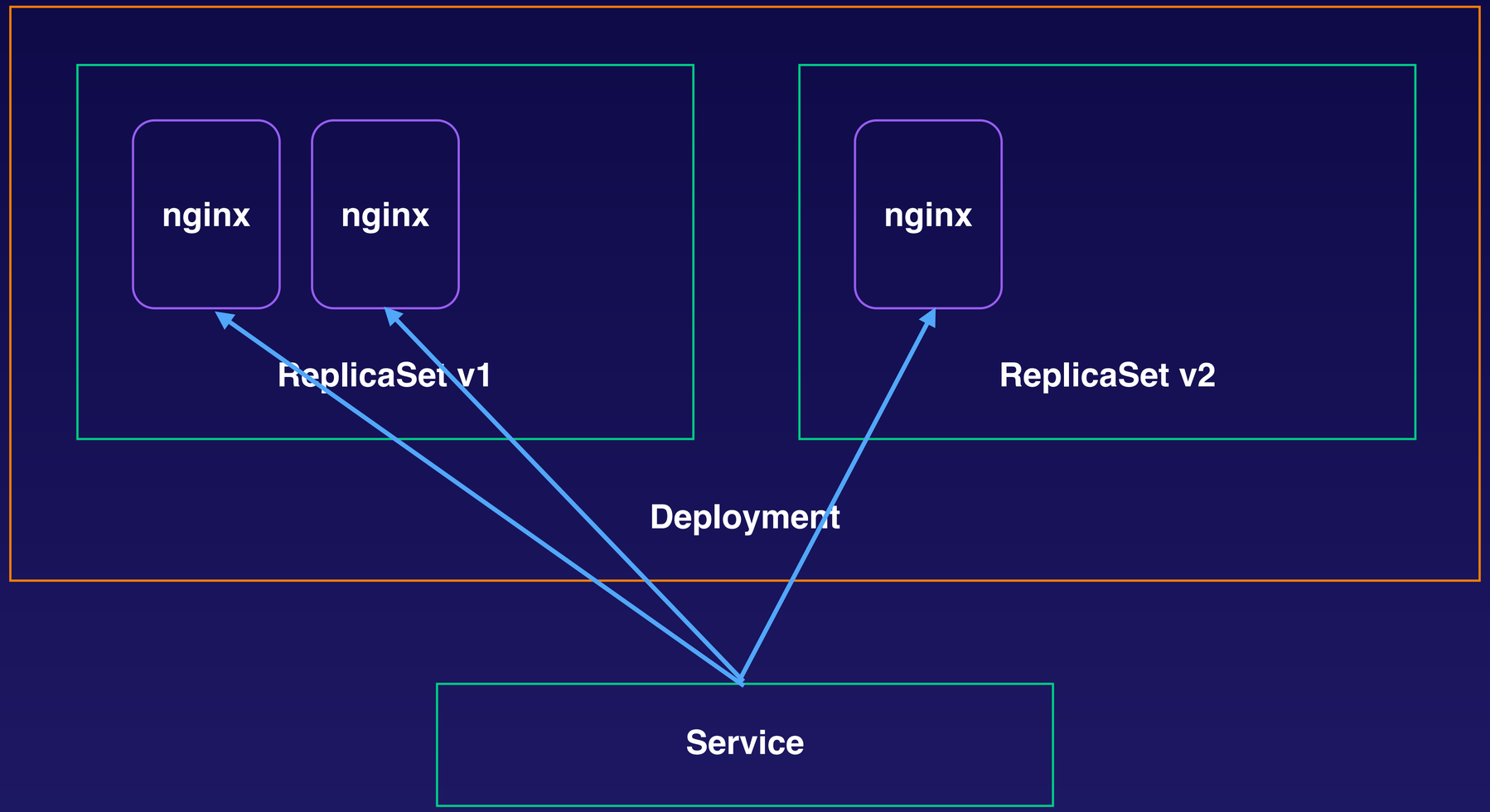
Deployment Patterns

ROLLING UPDATE



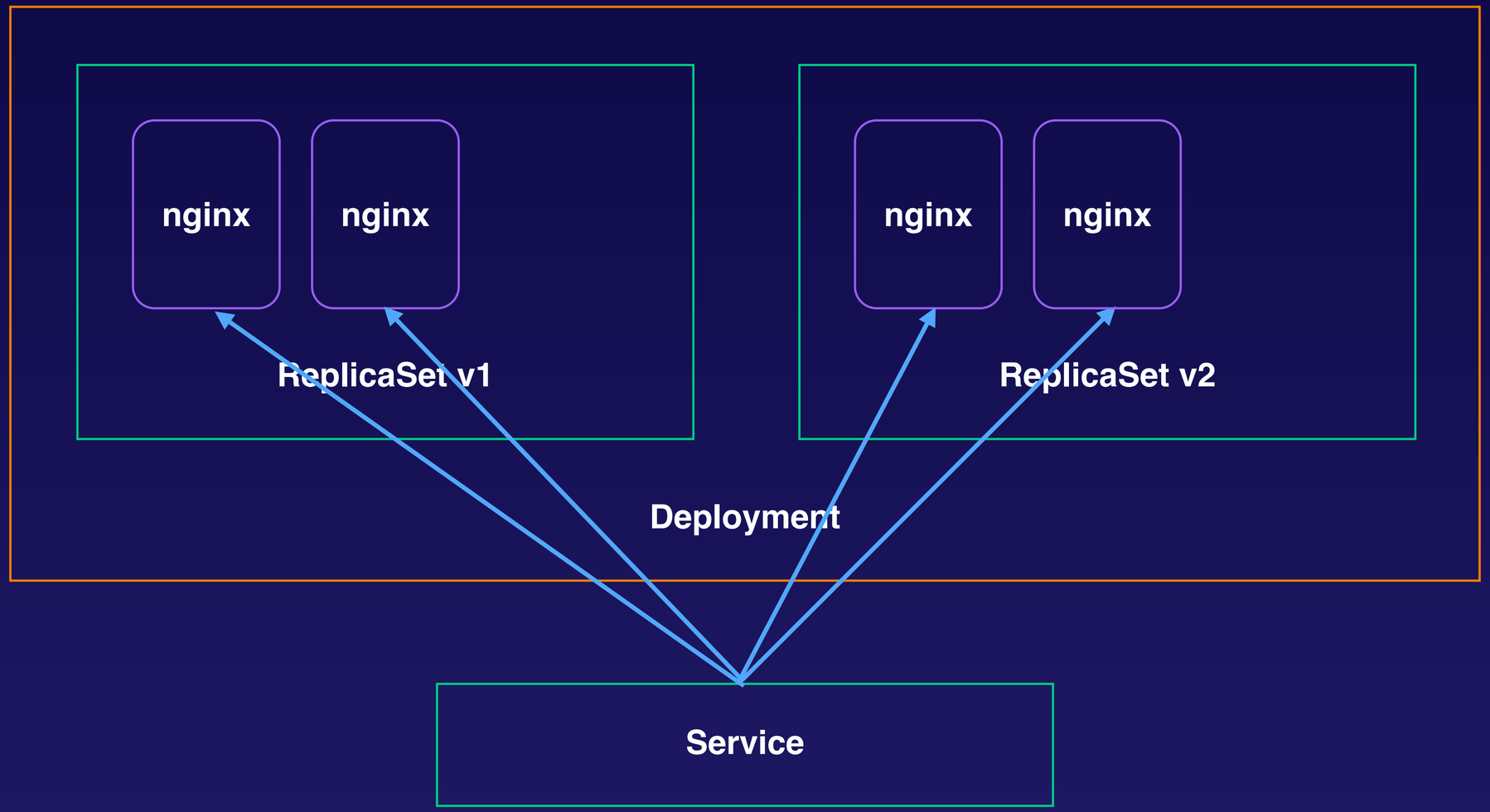
Deployment Patterns

ROLLING UPDATE



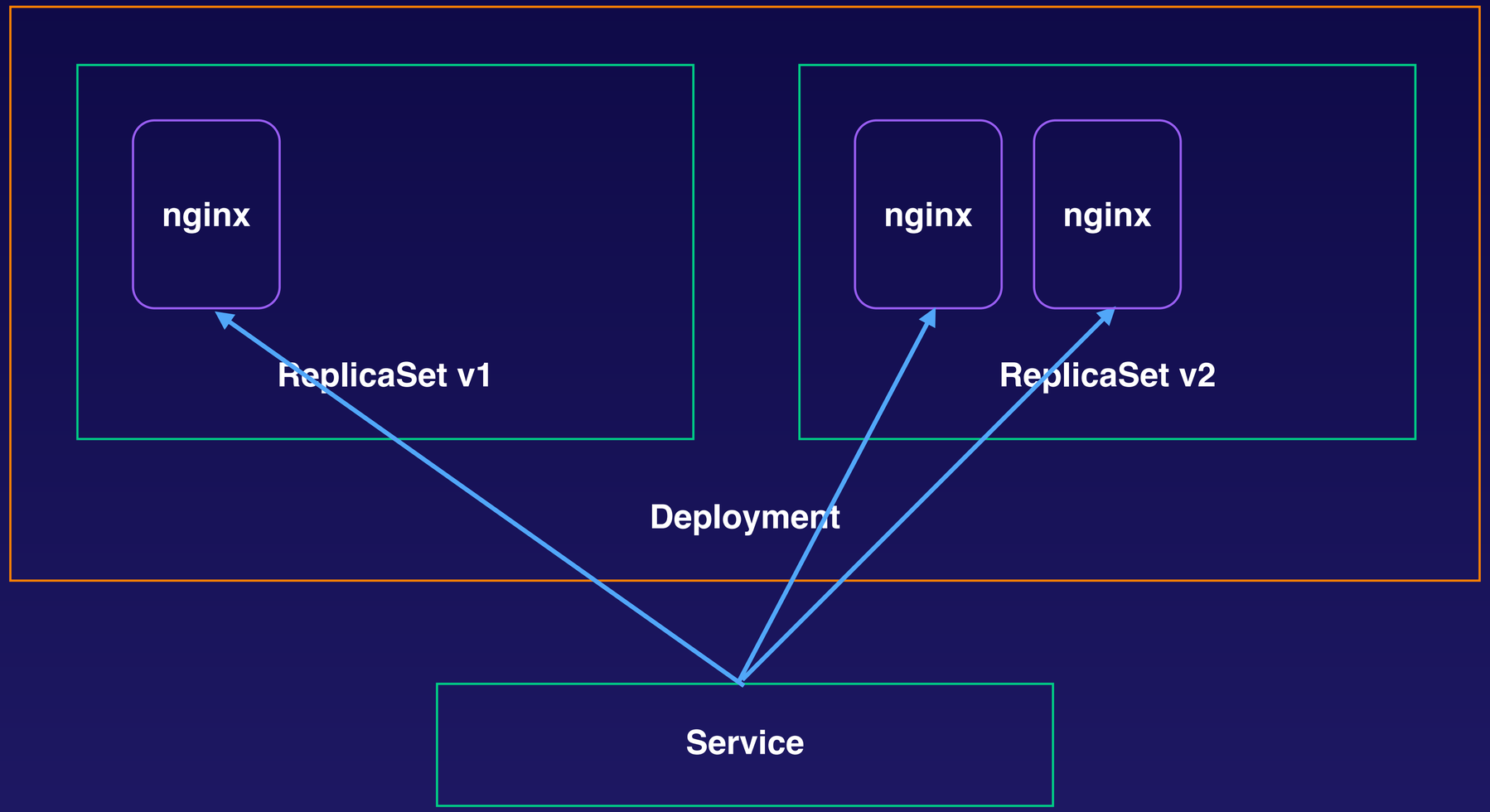
Deployment Patterns

ROLLING UPDATE



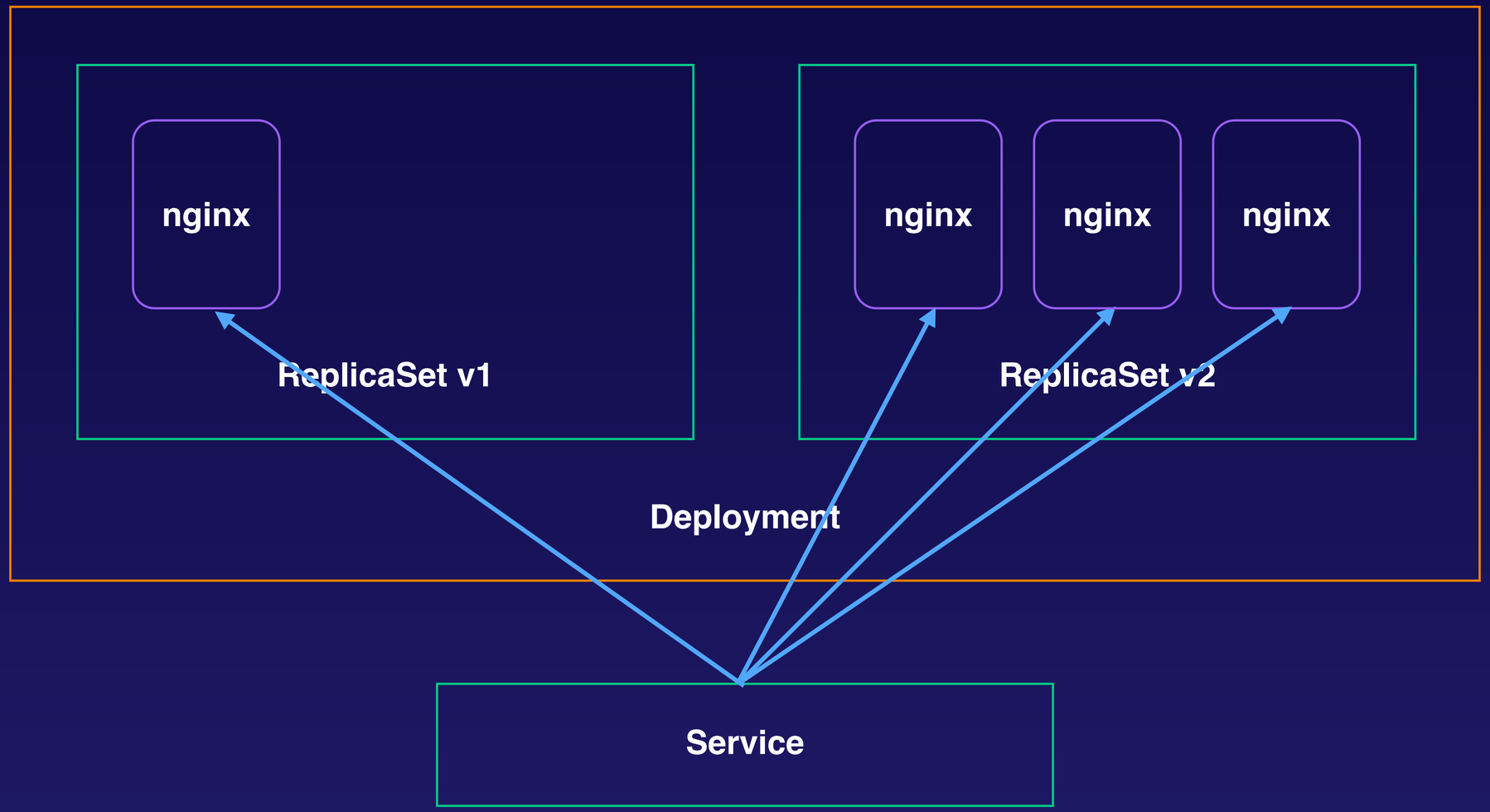
Deployment Patterns

ROLLING UPDATE



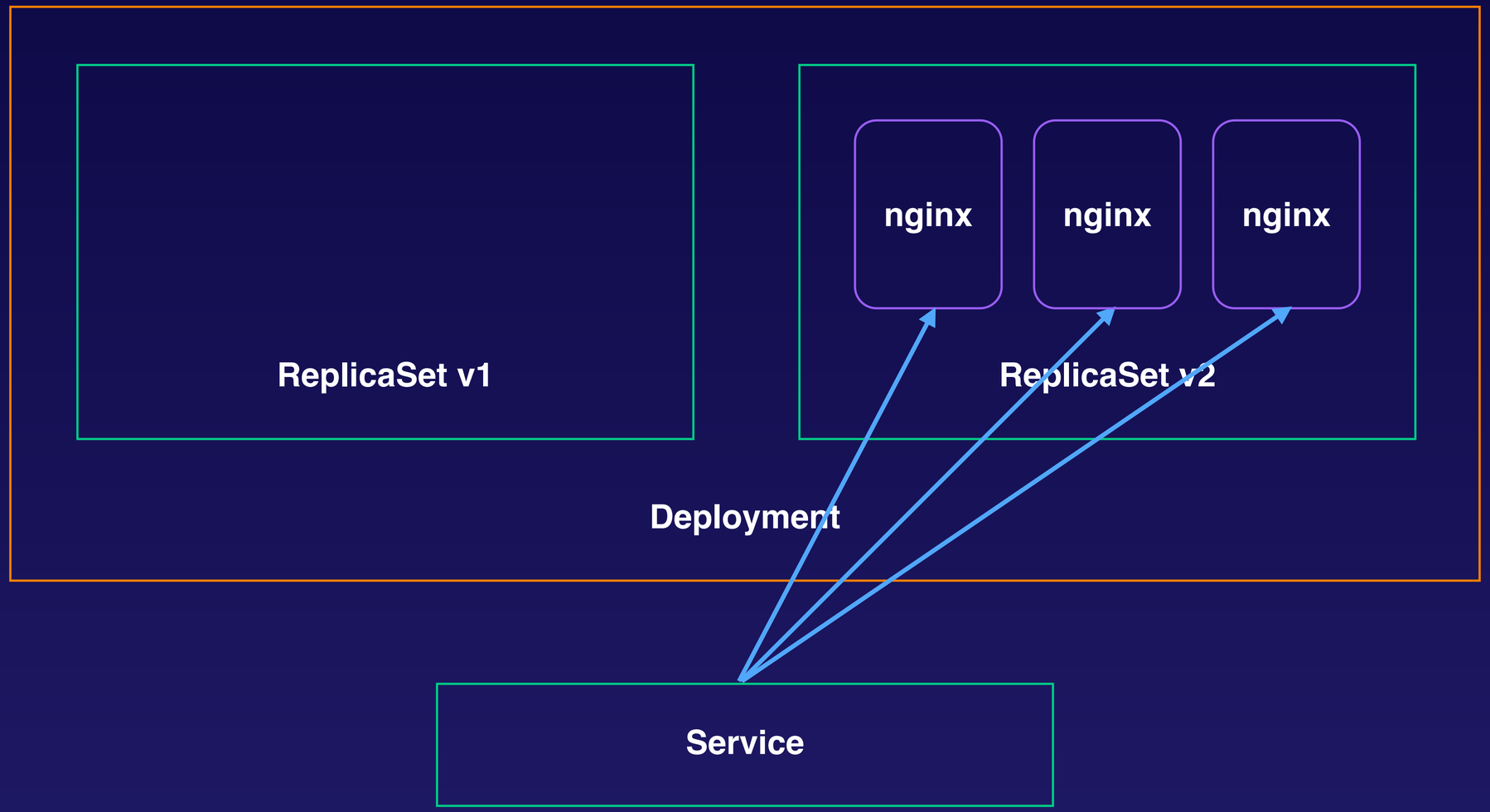
Deployment Patterns

ROLLING UPDATE



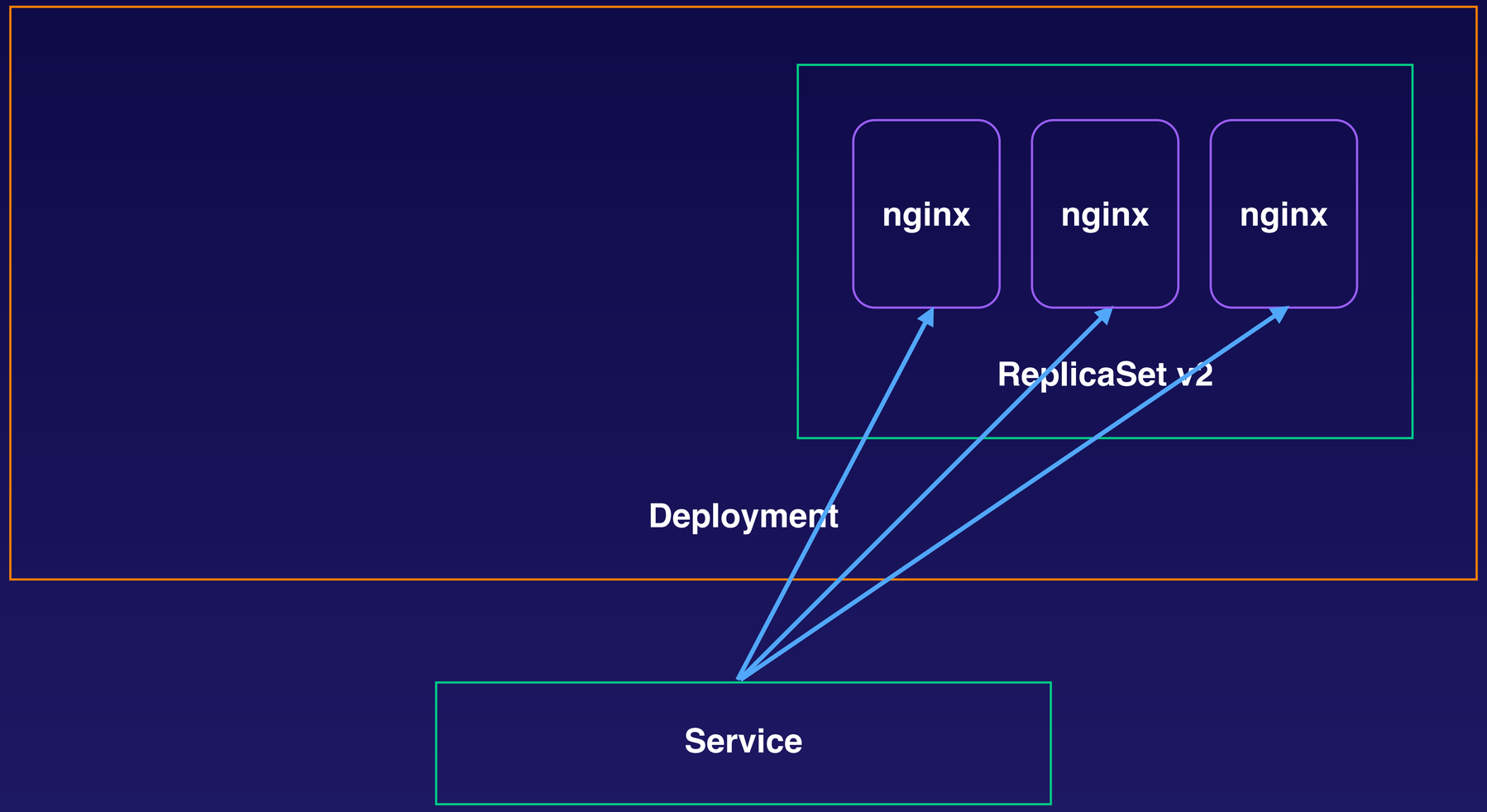
Deployment Patterns

ROLLING UPDATE



Deployment Patterns

ROLLING UPDATE



Canary Deployments

- Combines multiple **Deployments** with a single **Service**
- Deploy updates to a **small** subset of traffic
- Testing with **production** traffic

Deployment Patterns


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-prod
spec:
  replicas: 90
  template:
    metadata:
      name: frontend
      labels:
        app: frontend
        env: prod
    spec:
      containers:
      - name: frontend
        image: frontend:v1
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-canary
spec:
  replicas: 10
  template:
    metadata:
      name: frontend
      labels:
        app: frontend
        env: canary
    spec:
      containers:
      - name: frontend
        image: frontend:v2
```

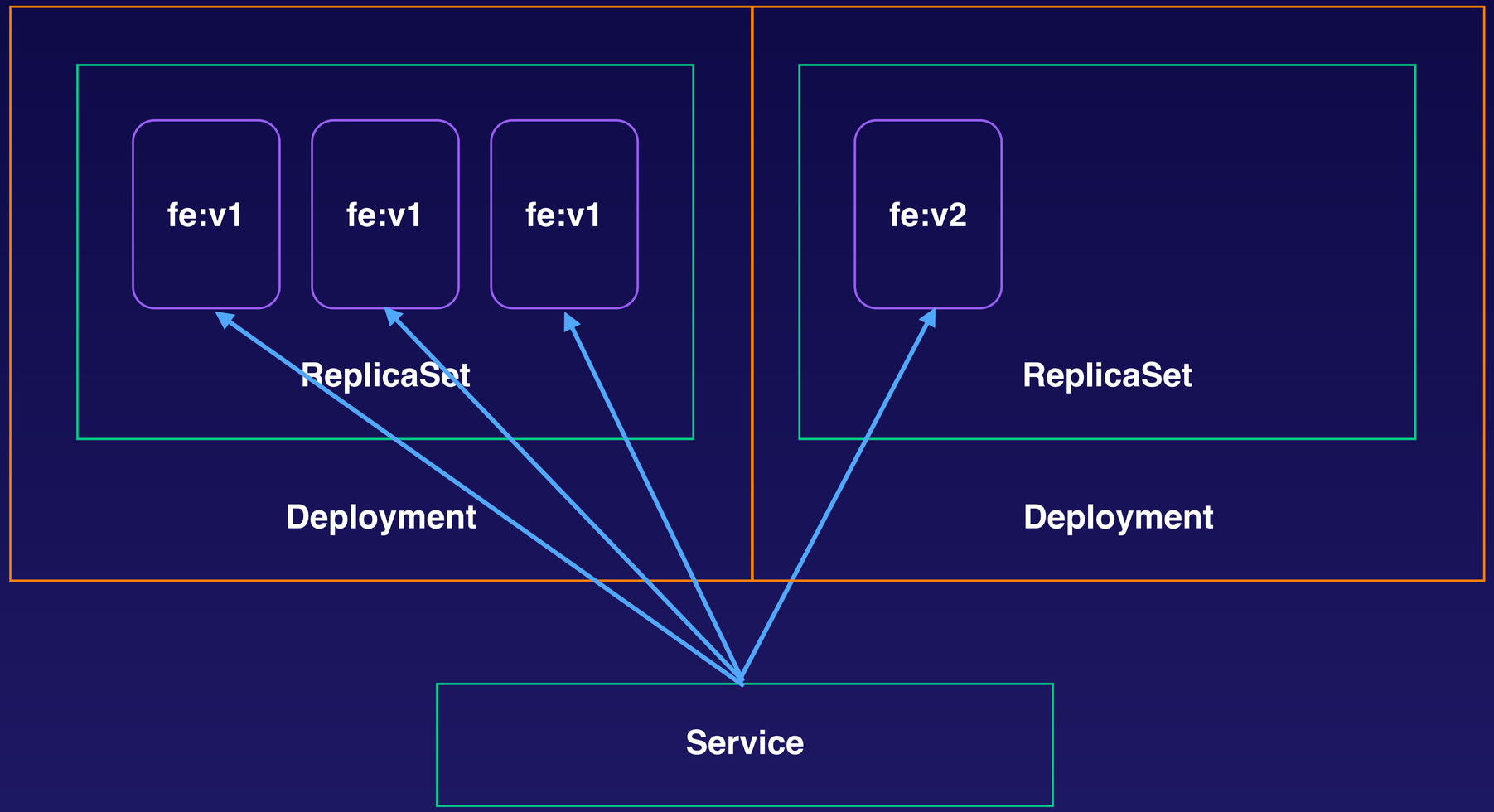
Deployment Patterns


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-prod
spec:
  replicas: 90
  template:
    metadata:
      name: frontend
      labels:
        app: frontend
        env: prod
    spec:
      containers:
      - name: frontend
        image: frontend:v1
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-canary
spec:
  replicas: 10
  template:
    metadata:
      name: frontend
      labels:
        app: frontend
        env: canary
    spec:
      containers:
      - name: frontend
        image: frontend:v2
```

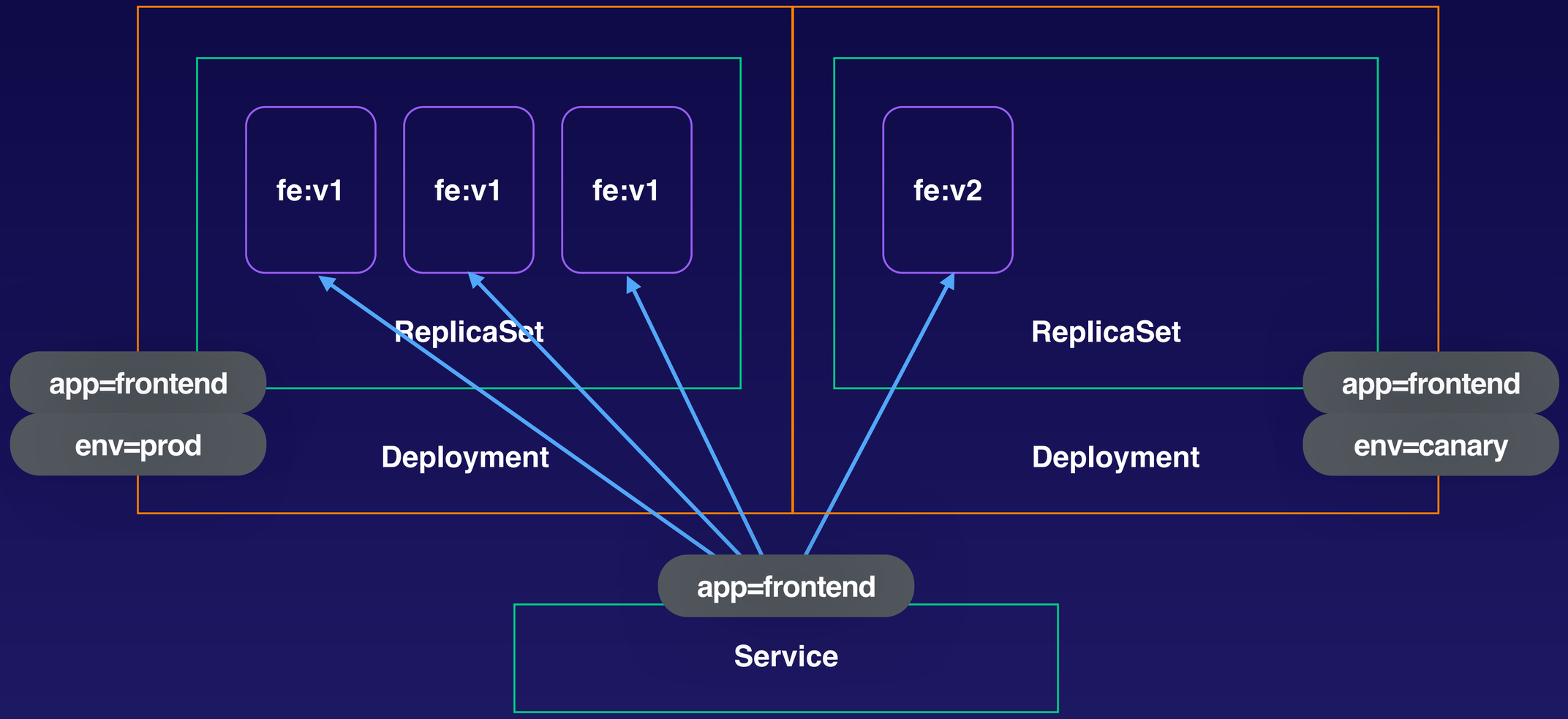
Deployment Patterns

CANARY DEPLOYMENT



Deployment Patterns

CANARY DEPLOYMENT



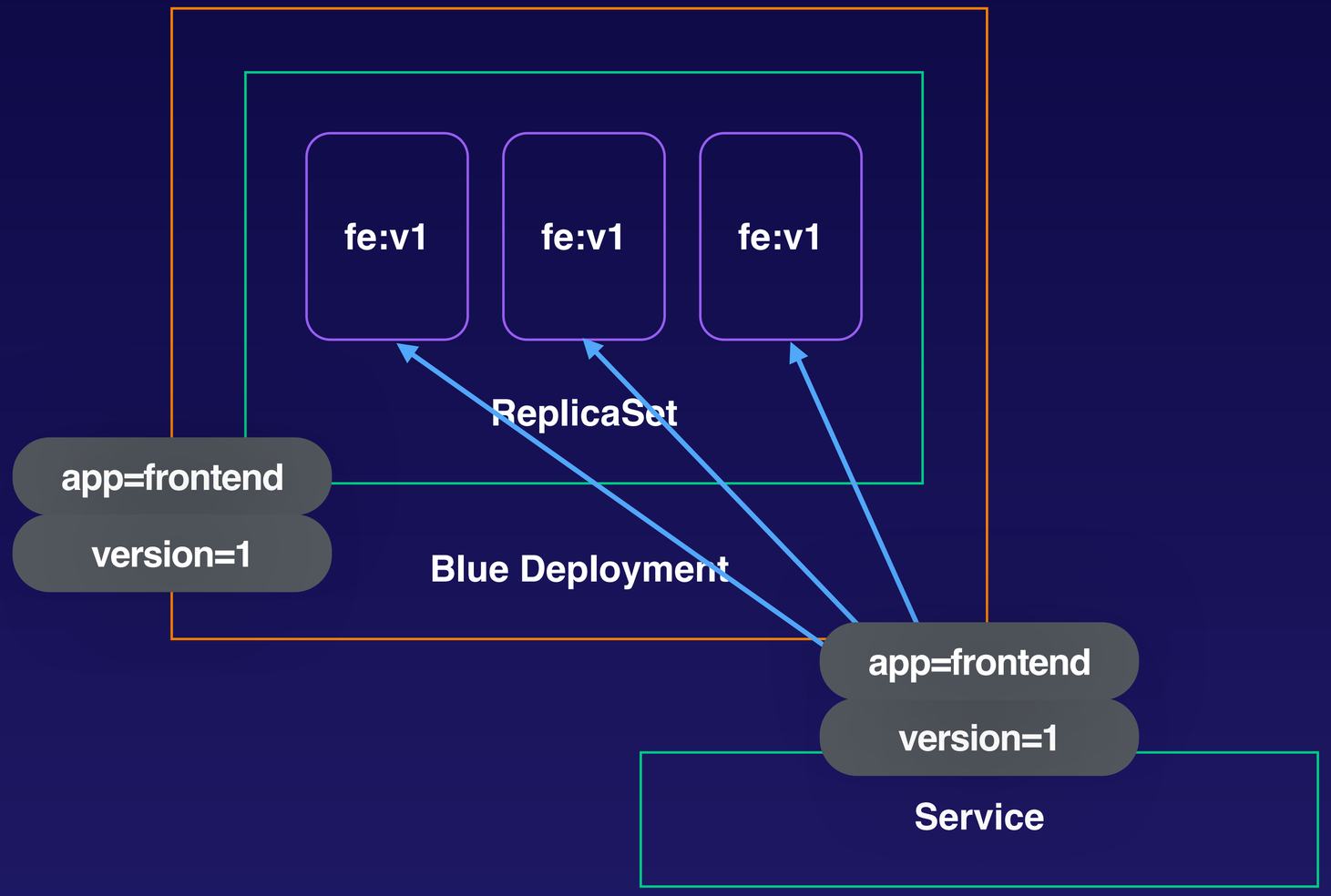
A CLOUD GURU

Blue-Green Deployments

- Maintain **two versions** of your application deployment
- Switch traffic from **blue** to **green** with **Service** selector
- **All traffic** immediately sent to **new deployment**
- **Switch back** in the event of problems

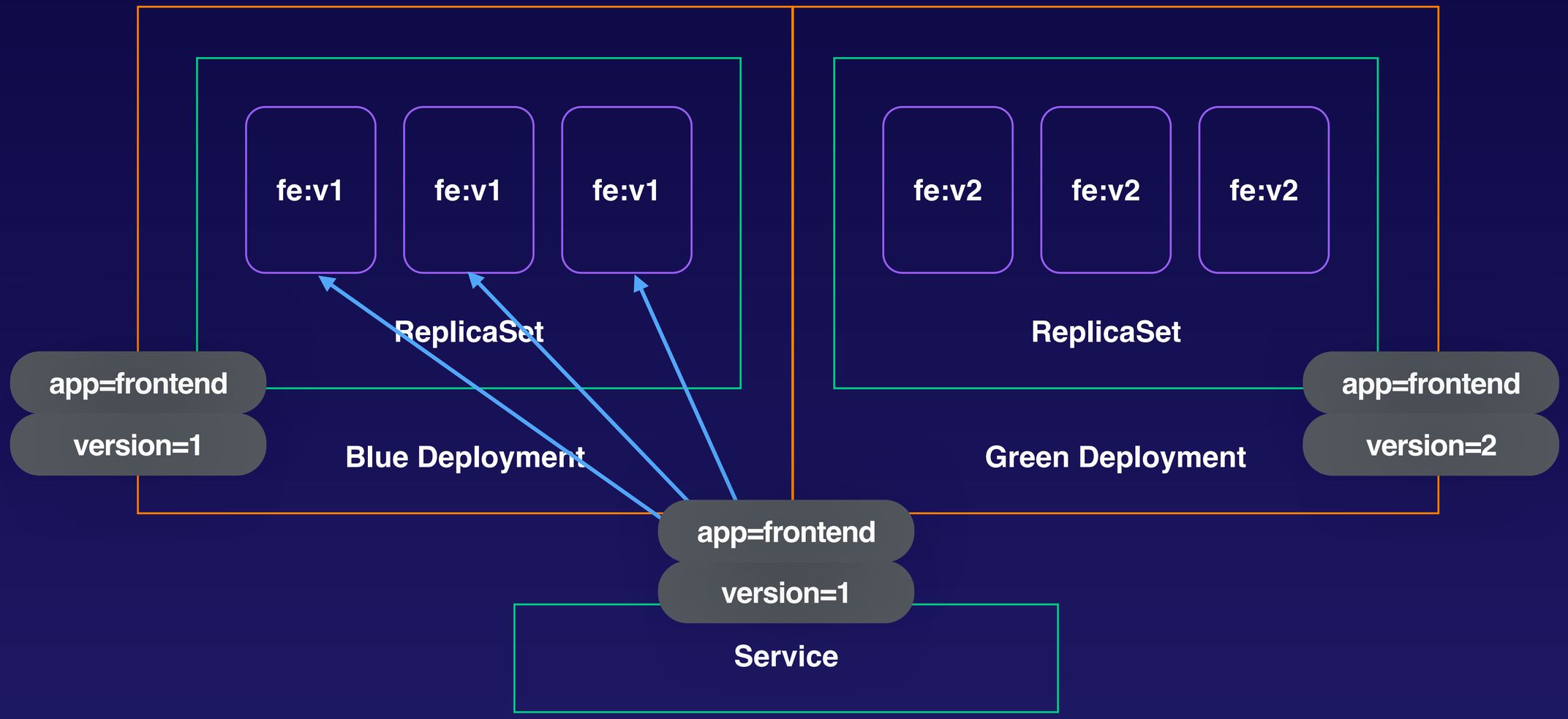
Deployment Patterns

BLUE/GREEN DEPLOYMENT

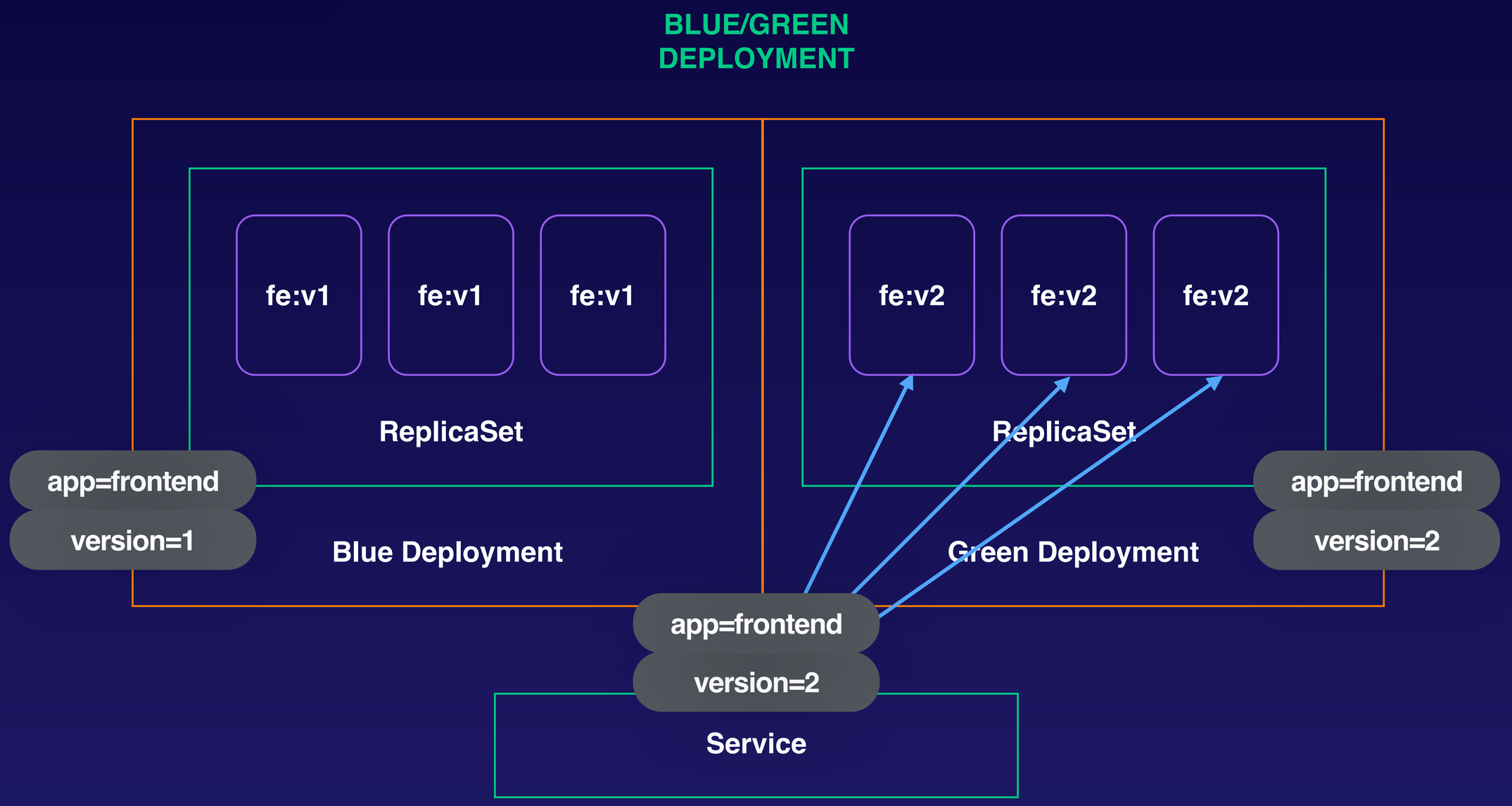


Deployment Patterns

BLUE/GREEN DEPLOYMENT

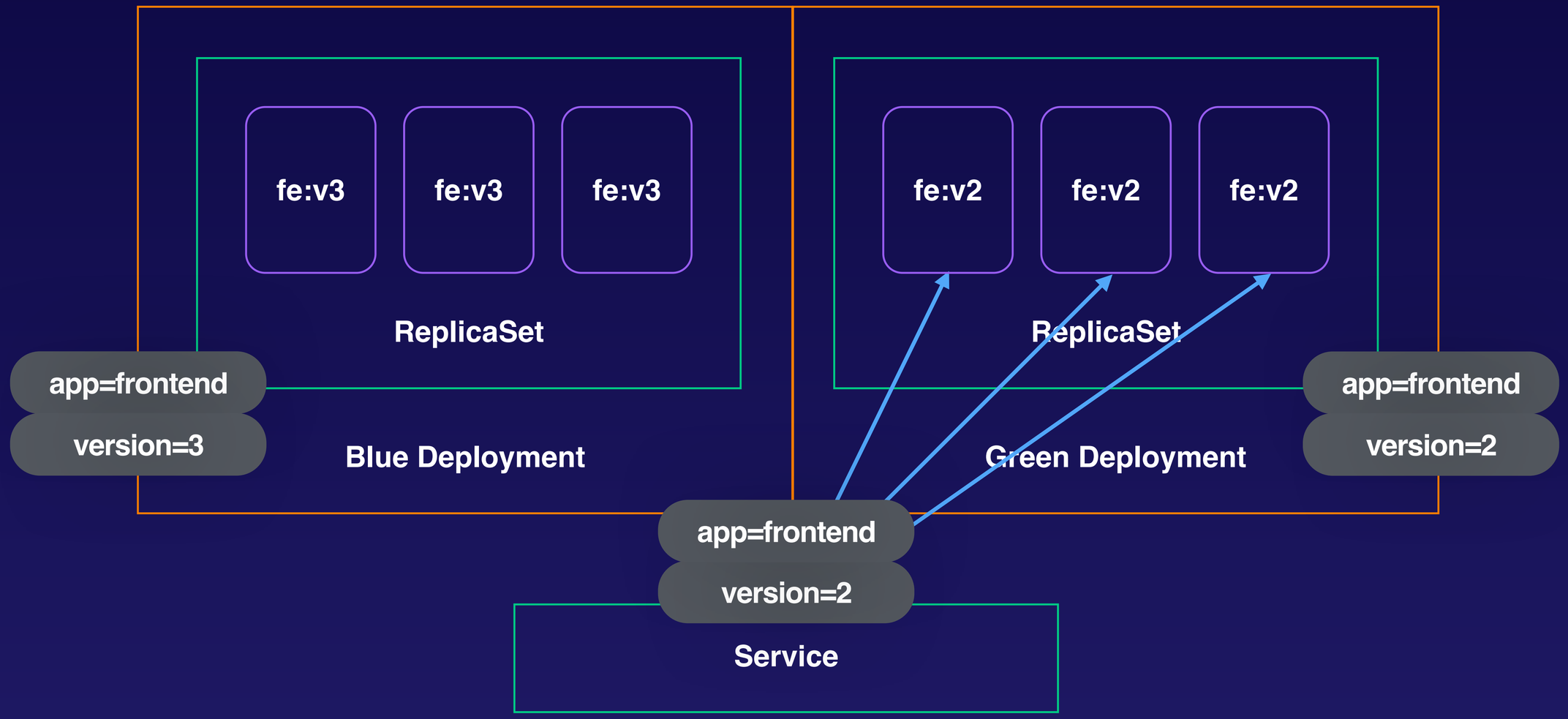


Deployment Patterns



Deployment Patterns

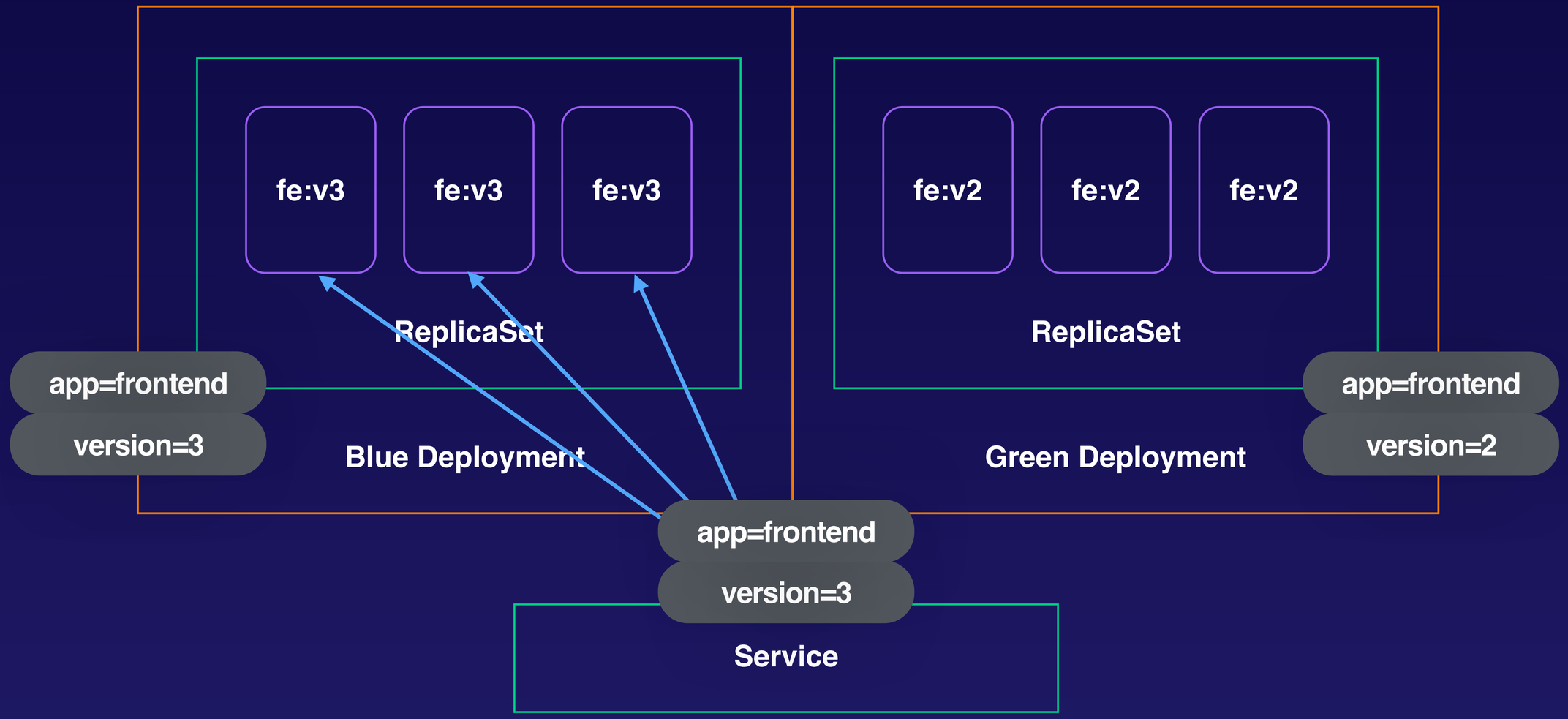
BLUE/GREEN DEPLOYMENT



Deployment Patterns

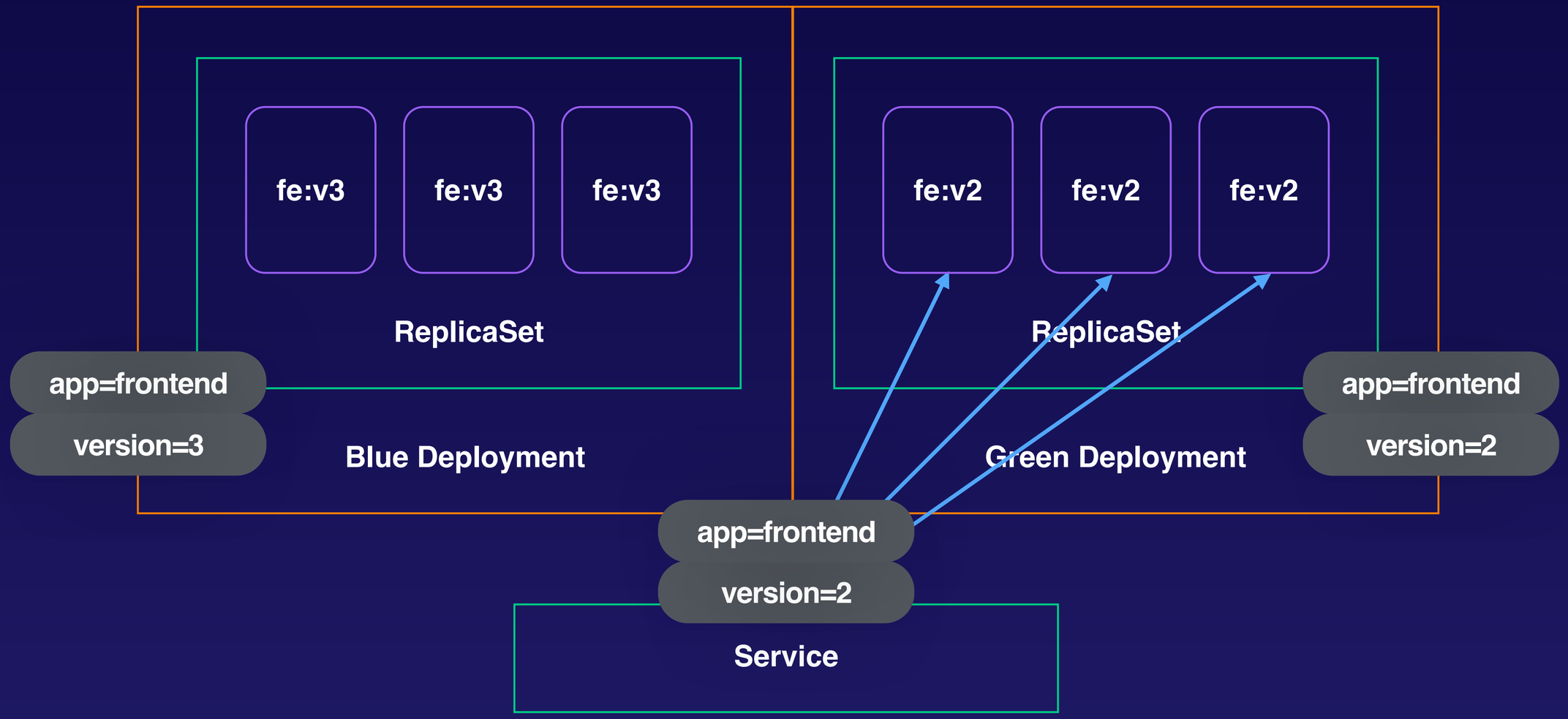
A CLOUD GURU

BLUE/GREEN DEPLOYMENT



Deployment Patterns

BLUE/GREEN DEPLOYMENT



A CLOUD GURU

Lab: Deployment patterns

Tim Berry
GOOGLE CLOUD GURU

Lab: Deployment patterns

Lab objectives:

- Create a **standard 3-node** GKE cluster
- Authenticate **kubectl** with **gcloud**
- Create a **blue/green** deployment
- **Update** application versions and **switch** traffic
- Create a **canary** deployment

A CLOUD GURU

Autoscaling all the things

Tim Berry
GOOGLE CLOUD GURU

A CLOUD GURU

Automatically scaling deployments

- **Horizontally** scale **Pods** to increase capacity when **demand** is high
  - `HorizontalPodAutoscaler`
- **Vertically** scale **Pods** to increase resources assigned to individual Pods
  - `VerticalPodAutoscaler`
- Scale **cluster nodes** when demand is so high that **extra VMs** are required
  - Node-pool cluster autoscaling in GKE

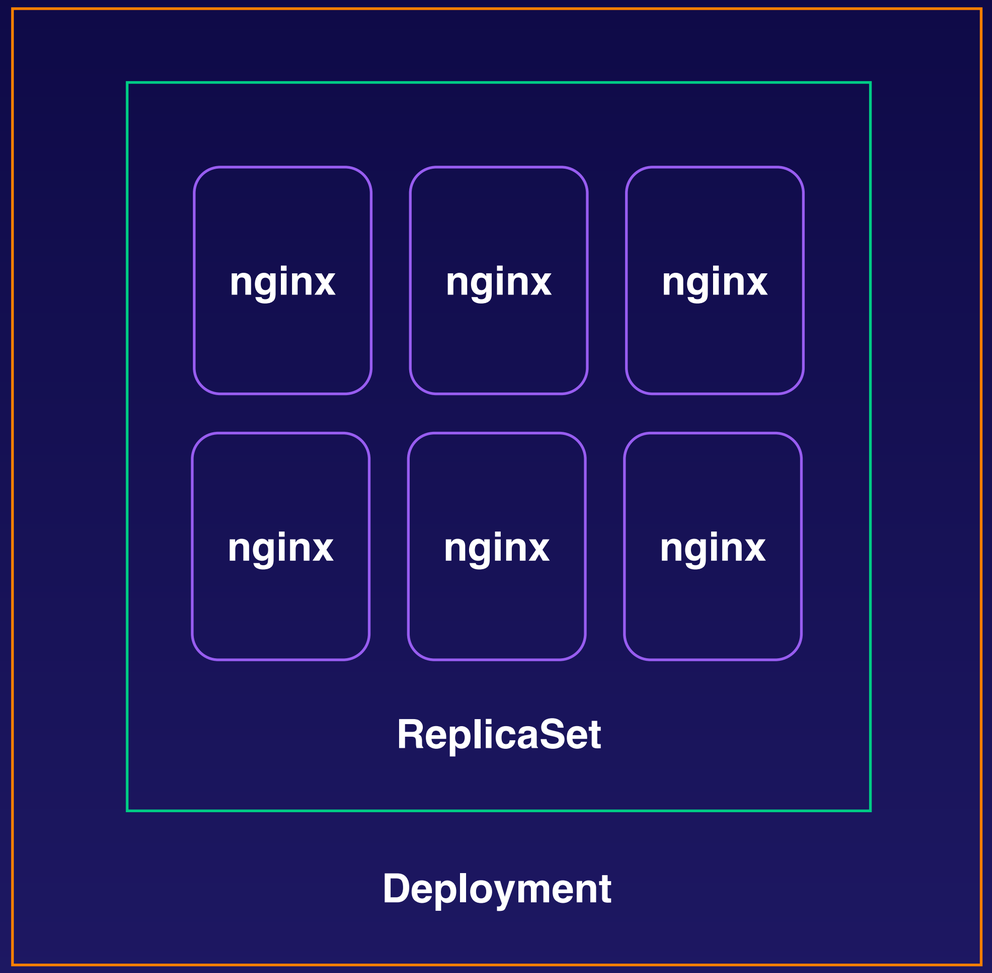
Horizontal Pod Autoscaler

- **Auto-scales** the number of **Pod replicas** in a ReplicaSet/Deployment
- **CPU** and **Memory** thresholds are observed by GKE to **trigger scaling**
- **Custom**, **multiple** and **external** metrics also supported
- **Stackdriver** metrics can also be used

Autoscaling all the things

HORIZONTAL POD AUTOSCALER

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: nginx-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx
  minReplicas: 3
  maxReplicas: 6
  targetCPUUtilizationPercentage: 60
```



A CLOUD GURU

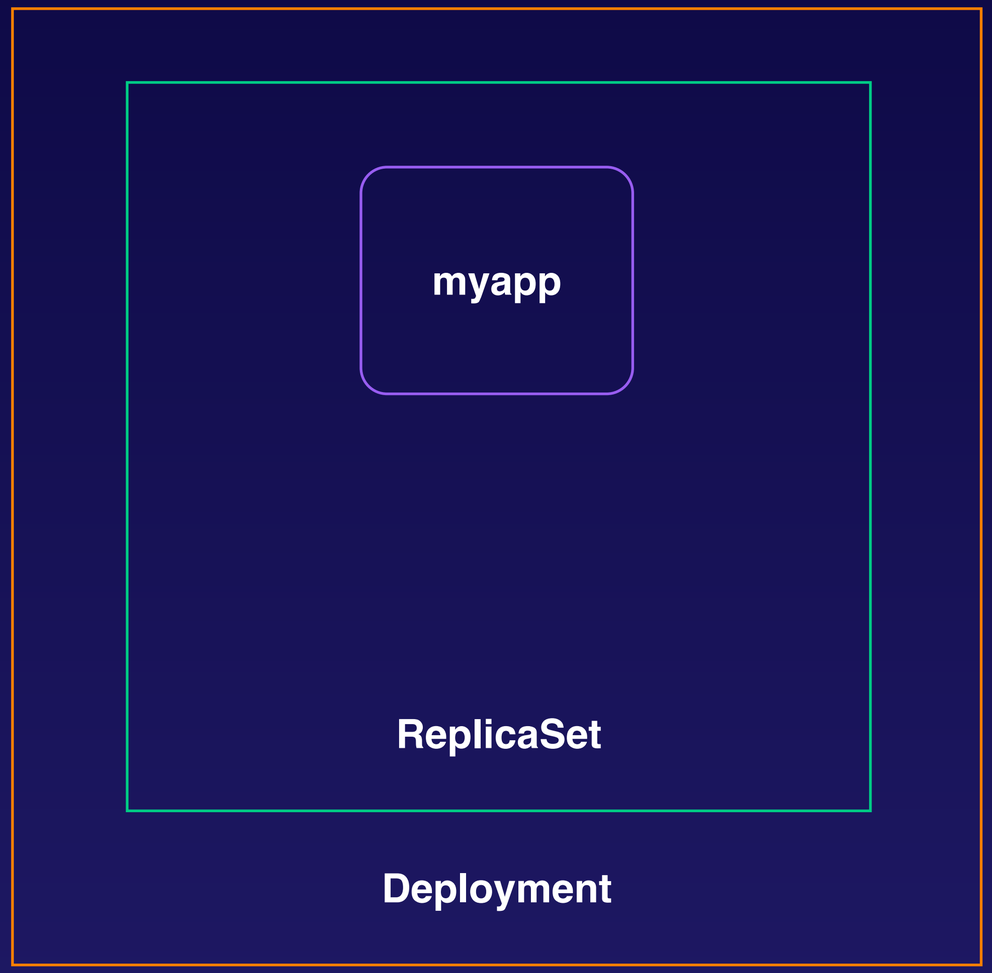
Vertical Pod Autoscaler

- **Automatically** recommends or applies **CPU** and **RAM requests** for Pods
- More suited to **stateful** deployments
- **Cannot** work alongside the Horizontal Pod Autoscaler
- Currently in **Beta** in **GKE**

Autoscaling all the things

VERTICAL POD AUTOSCALER

```yaml
apiVersion: autoscaling.k8s.io/v1beta2
kind: VeritcalPodAutoscaler
metadata:
  name: myapp-vpa
spec:
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp-deployment
 updatePolicy:
   updateMode: "auto"
```



A CLOUD GURU

Cluster Autoscaler

- **Automatically** grow or shrink your **GKE** cluster based on **demand**
- New nodes **added** when Pods don't have enough capacity to run
- Underutilized nodes automatically **deleted**
- Works best with **Node Pools**
- Group **VM nodes** of a specific type together - eg. High **CPU** or High **RAM**

Best Practices

- Design **node pools** around the requirements of **specific workloads**
- Enable **cluster autoscaling** and **horizontal Pod autoscaling**
- For example:
  - Run CI/CD, **ancillary services** on a single **fixed** node pool
  - Run web **frontend workloads** on a **generic autoscaling** node pool
  - Run **RAM intensive** Java applications in a **high RAM instance-type** node pool

A CLOUD GURU

Lab: Autoscaling all the things

Tim Berry
GOOGLE CLOUD GURU

Lab: Autoscaling all the things

Lab objectives:

- Add a **HorizontalPodAutoscaler** to myapp
- Enable GKE **cluster autoscaling**
- **Simulate** high demand and **observe** autoscaling of Pods and nodes

A CLOUD GURU

CHAPTER FOUR

Advanced GKE Operations

A CLOUD GURU

Helm: The Kubernetes package manager

Tim Berry
GOOGLE CLOUD GURU

What is Helm?

- Packages Kubernetes **object manifests** and **configuration** in a Helm **chart**
- Maintains the **lifecycle** of a **deployment** to GKE
- **Public repos** of helm charts for **popular** software
- **Other** manifest-management solutions are available

Using Helm

- Install the **helm** tool
- Search for software in the **Helm Hub**
- Add the necessary Helm **repository**
- Install the Helm **chart**
  - `helm install my-wordpress bitnami/wordpress`
- Helm applies the **templated manifests** from the chart to your cluster

Helm: The Kubernetes package manager


```
wordpress/
  Chart.yaml
  LICENSE
  README.md
  requirements.yaml
  values.yaml
  templates/
```

Helm: The Kubernetes package manager


```
wordpress/
  Chart.yaml
  LICENSE
  README.md
  requirements.yaml
  values.yaml
  templates/
```

- Chart metadata, including name, version, description, maintainers, and sources

Helm: The Kubernetes package manager


```
wordpress/
  Chart.yaml
  LICENSE
  README.md
  requirements.yaml
  values.yaml
  templates/
```

- Chart metadata, including name, version, description, maintainers, and sources
- Additional helm charts required for deployment, dynamically linked

Helm: The Kubernetes package manager


```
wordpress/
  Chart.yaml
  LICENSE
  README.md
  requirements.yaml
  values.yaml
  templates/
```

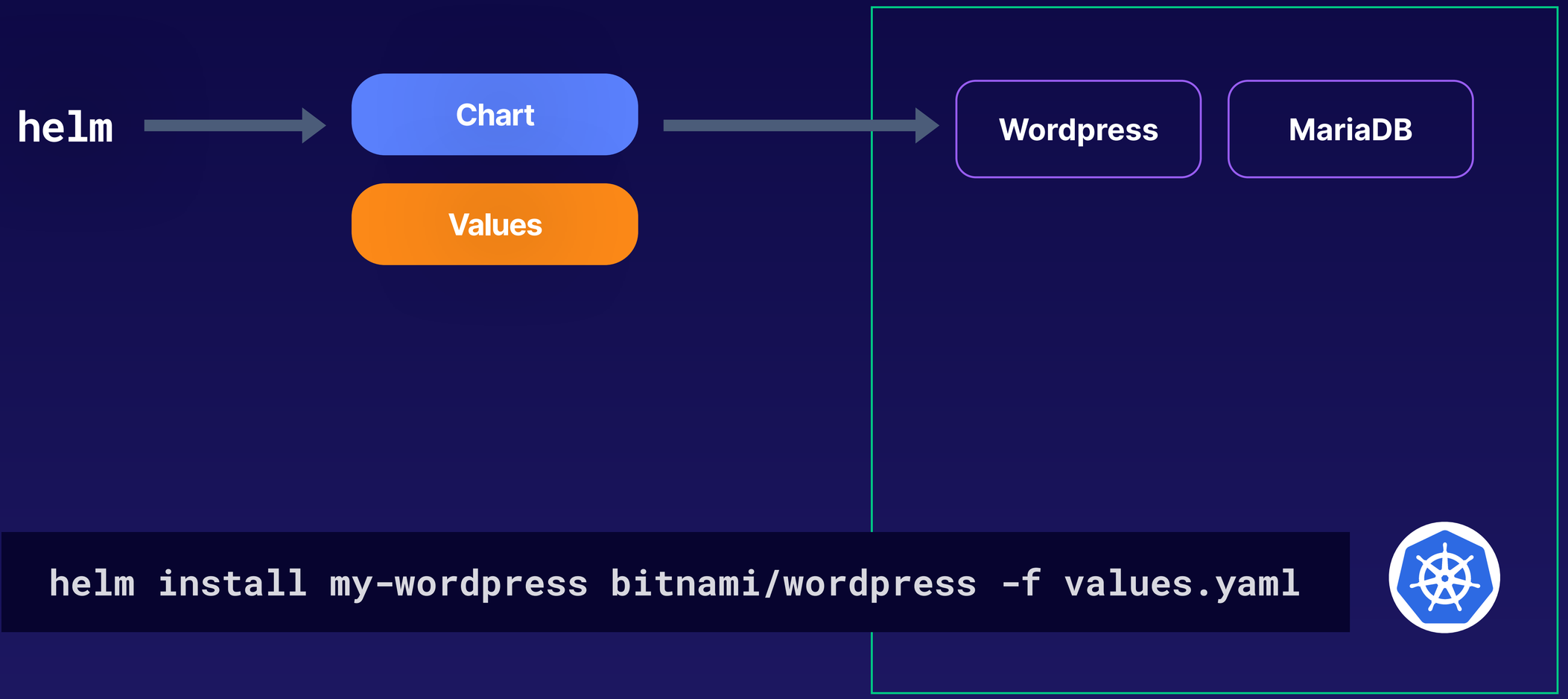
- Chart metadata, including name, version, description, maintainers, and sources
- Additional helm charts required for deployment, dynamically linked
- Customizable configuration specific to this deployment, such as image versions, resource requests and ingress options

Helm: The Kubernetes package manager


```
wordpress/
  Chart.yaml
  LICENSE
  README.md
  requirements.yaml
  values.yaml
  templates/
```

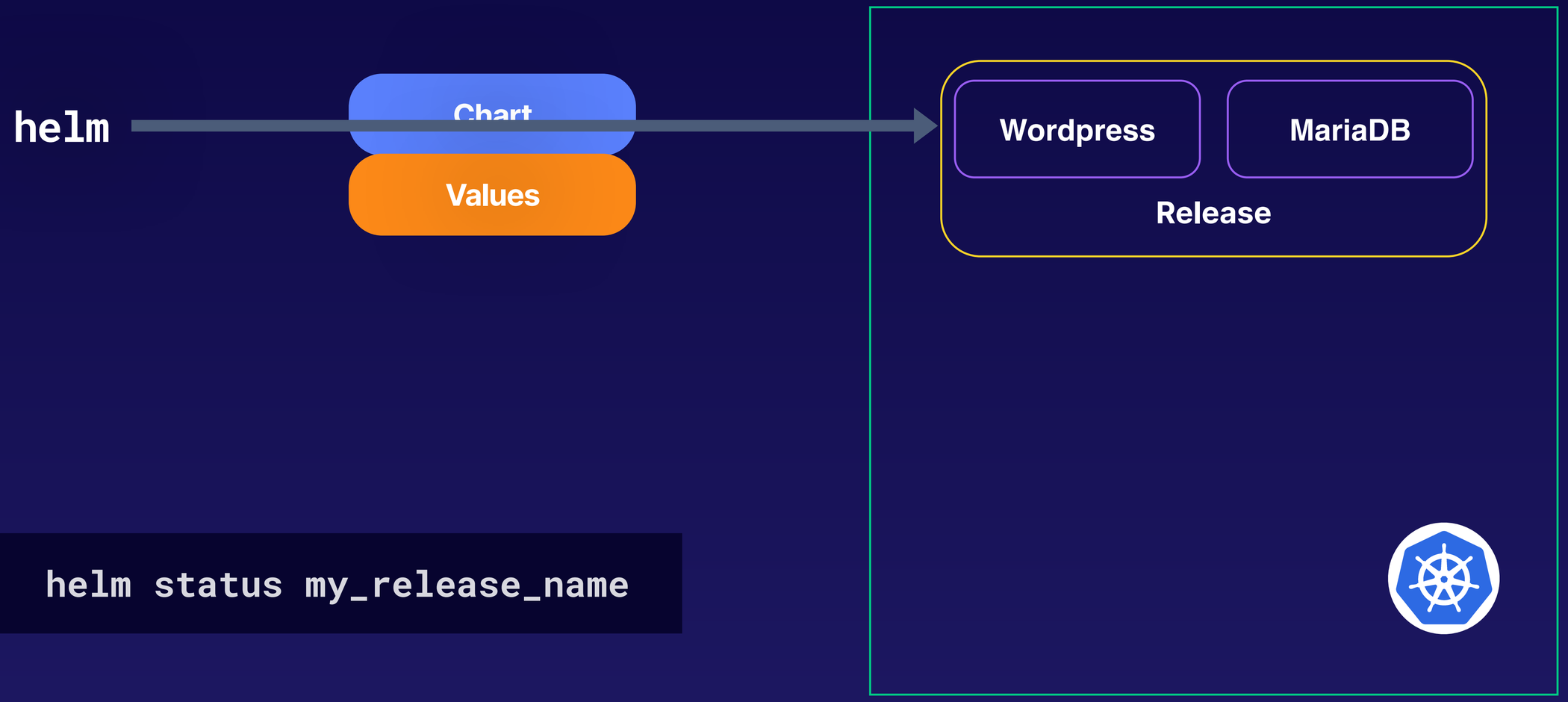
- Chart metadata, including name, version, description, maintainers, and sources
- Additional helm charts required for deployment, dynamically linked
- Customizable configuration specific to this deployment, such as image versions, resource requests, and ingress options
- Kubernetes YAML manifests, templated with Go, allowing for the dynamic use of configuration values

Helm: The Kubernetes package manager



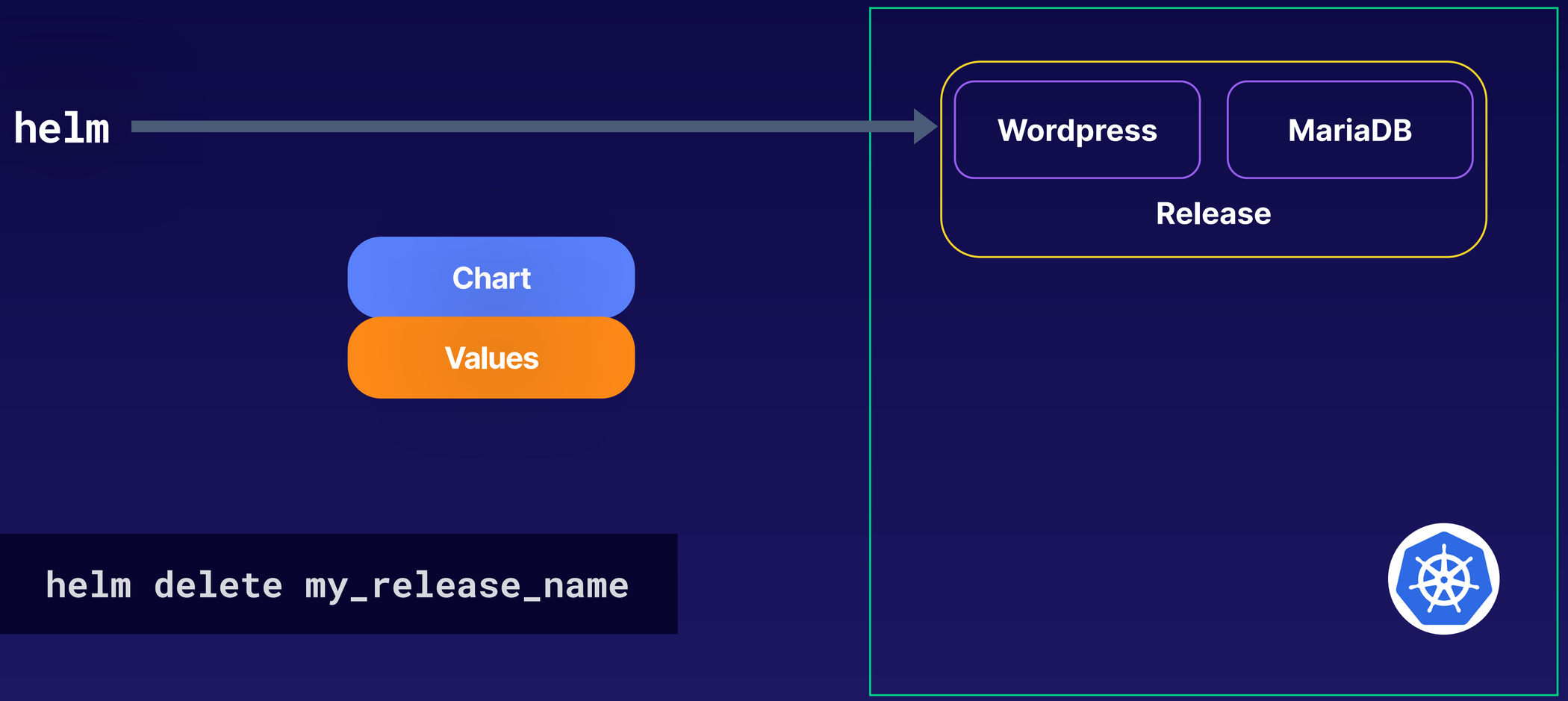

```
helm install my-wordpress bitnami/wordpress -f values.yaml
```

Helm: The Kubernetes package manager




```
helm status my_release_name
```

Helm: The Kubernetes package manager




```
helm delete my_release_name
```

A CLOUD GURU

Lab: Deploying applications with Helm

Tim Berry
GOOGLE CLOUD GURU

Lab: Deploying applications with Helm

Lab objectives:

- Create a **standard 3-node** GKE cluster
- Authenticate **kubectl** with **gcloud**
- Install **Elasticsearch** with **Helm**
- Query, update, and remove Helm releases

A CLOUD GURU

Advanced Ingress control

Tim Berry
GOOGLE CLOUD GURU

A CLOUD GURU

Ingress

- A more **customizable** way to expose traffic than a cloud load balancer
- An **Ingress object** configures access to **services** from **outside** the cluster
- Designed for **HTTP** and **HTTPS** services
- Can provide **SSL**, **name-based**, and **path-based** routing

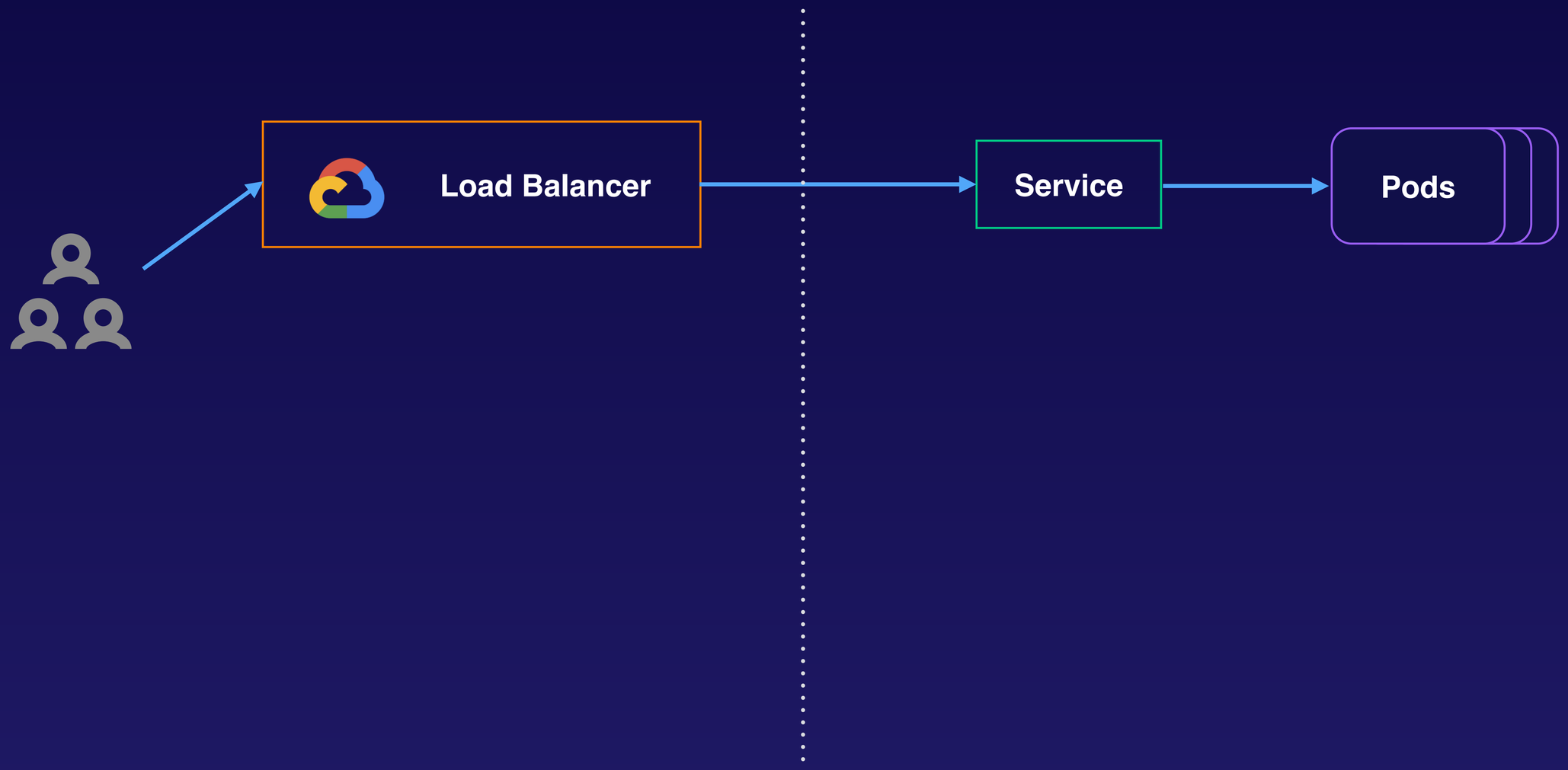
A CLOUD GURU

Ingress Controller

- Routes traffic to **services** based on **Ingress** definitions
- Usually fronted by a **cloud load balancer**
- **Consolidates** your routing through a single **resource**
- **NGINX Ingress Controller** deployment is very common
- Other ingress controllers are available!

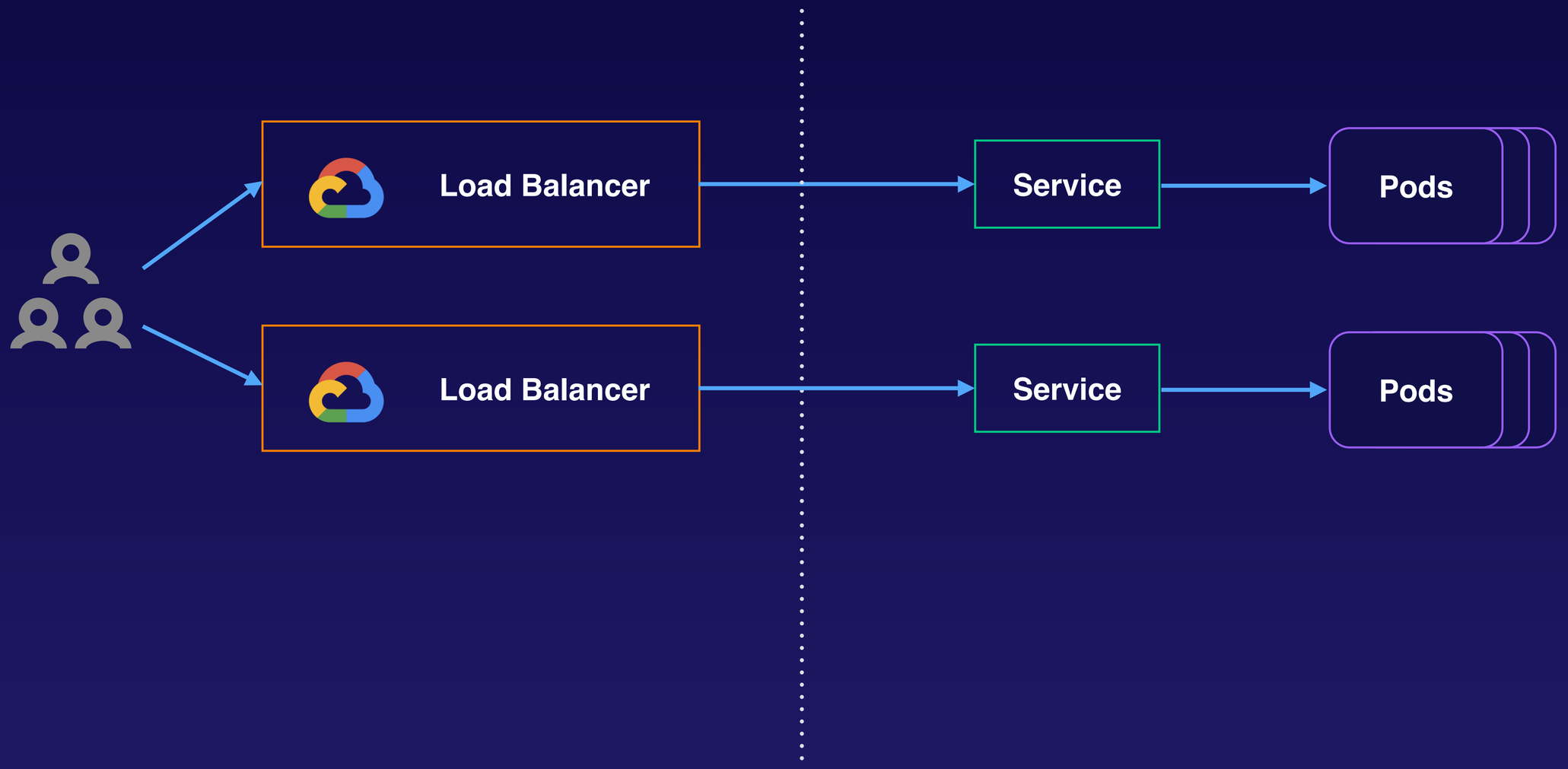
Advanced Ingress control

USING A CLOUD LOAD BALANCER



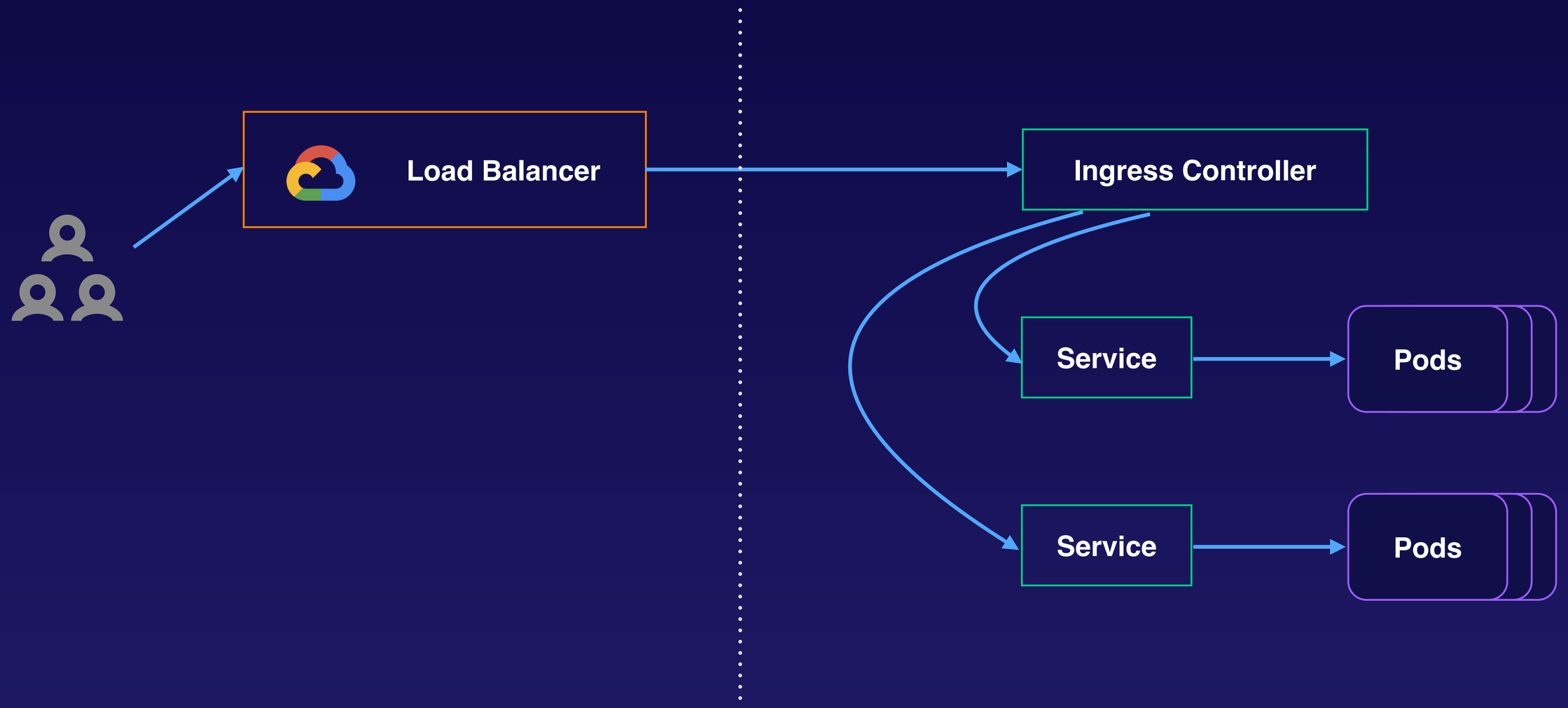
Advanced Ingress control

USING A CLOUD LOAD BALANCER



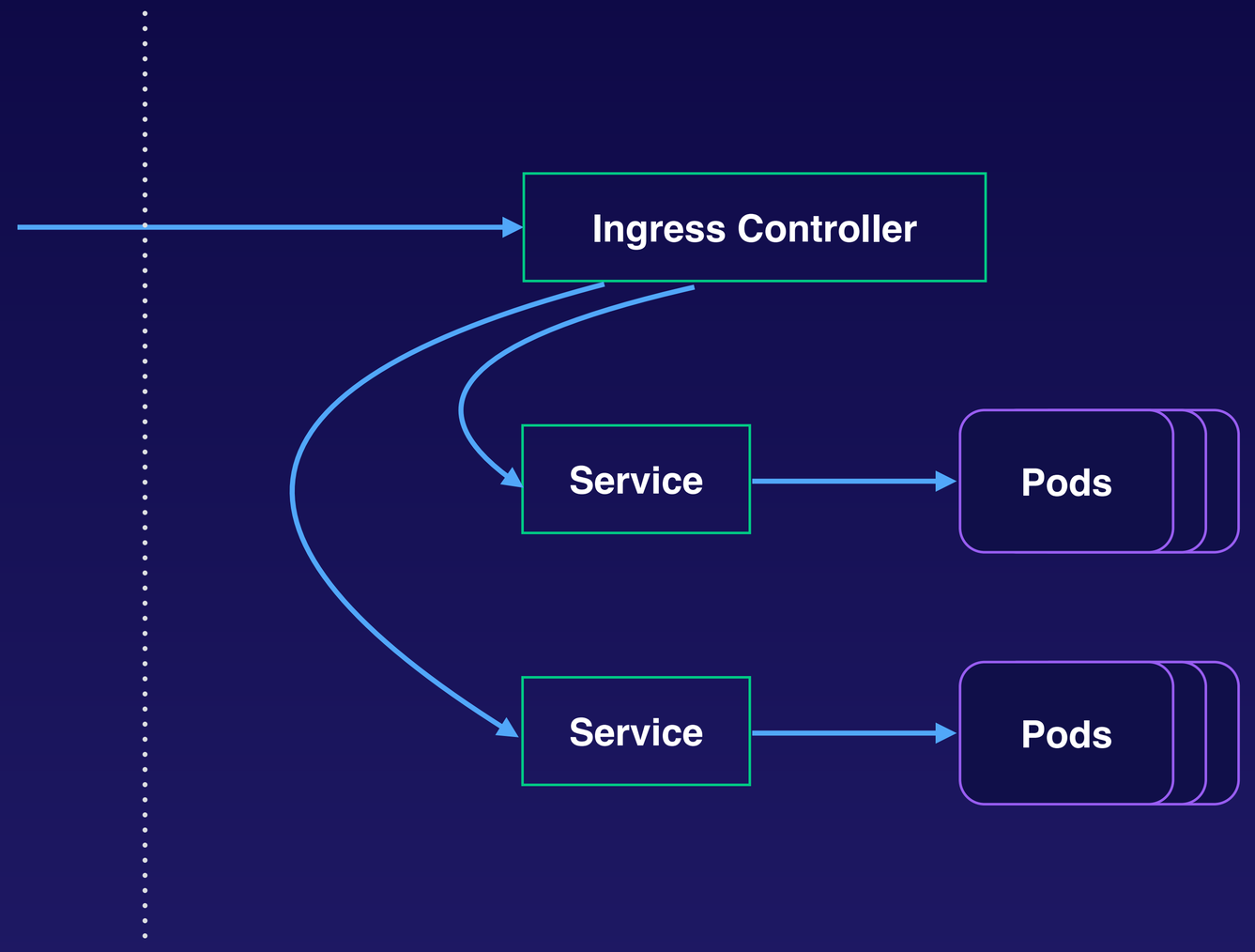
Advanced Ingress control

USING AN INGRESS CONTROLLER



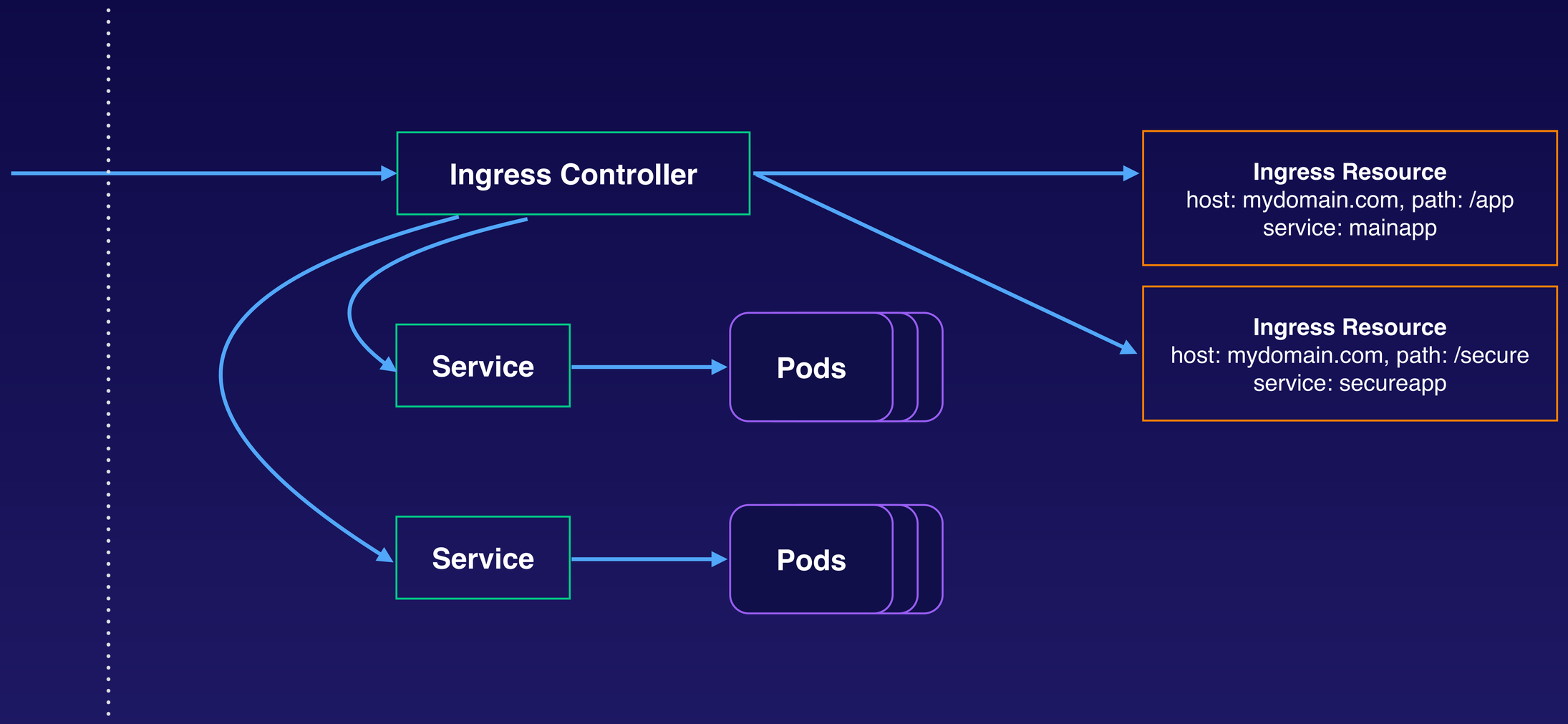
Advanced Ingress control

USING AN INGRESS CONTROLLER



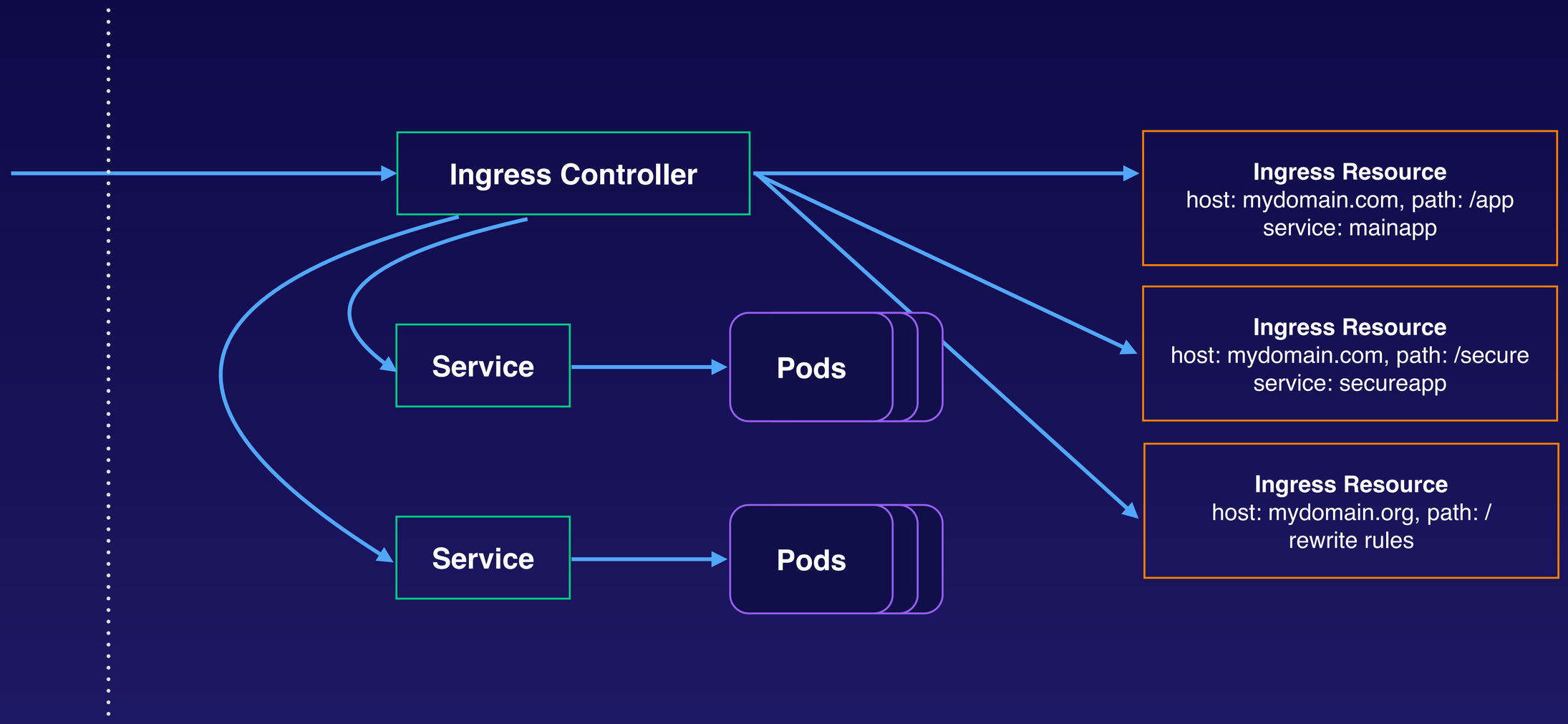
Advanced Ingress control

USING AN INGRESS CONTROLLER



Advanced Ingress control

USING AN INGRESS CONTROLLER



A CLOUD GURU

NGINX Ingress Controller

- Easily installed via **Helm**
- Runs as a **deployment** or **DaemonSet**
- Supports **autoscaling**
- **Supported** by the Kubernetes project

A CLOUD GURU

Lab: Routing traffic with Ingress controllers

Tim Berry
GOOGLE CLOUD GURU

Lab: Routing traffic with Ingress controllers

Lab objectives:

- Create a **standard 3-node** GKE cluster
- Authenticate **kubectl** with **gcloud**
- Install **NGINX** Ingress Controller with **Helm**
- Configure **Ingress** resources

A CLOUD GURU

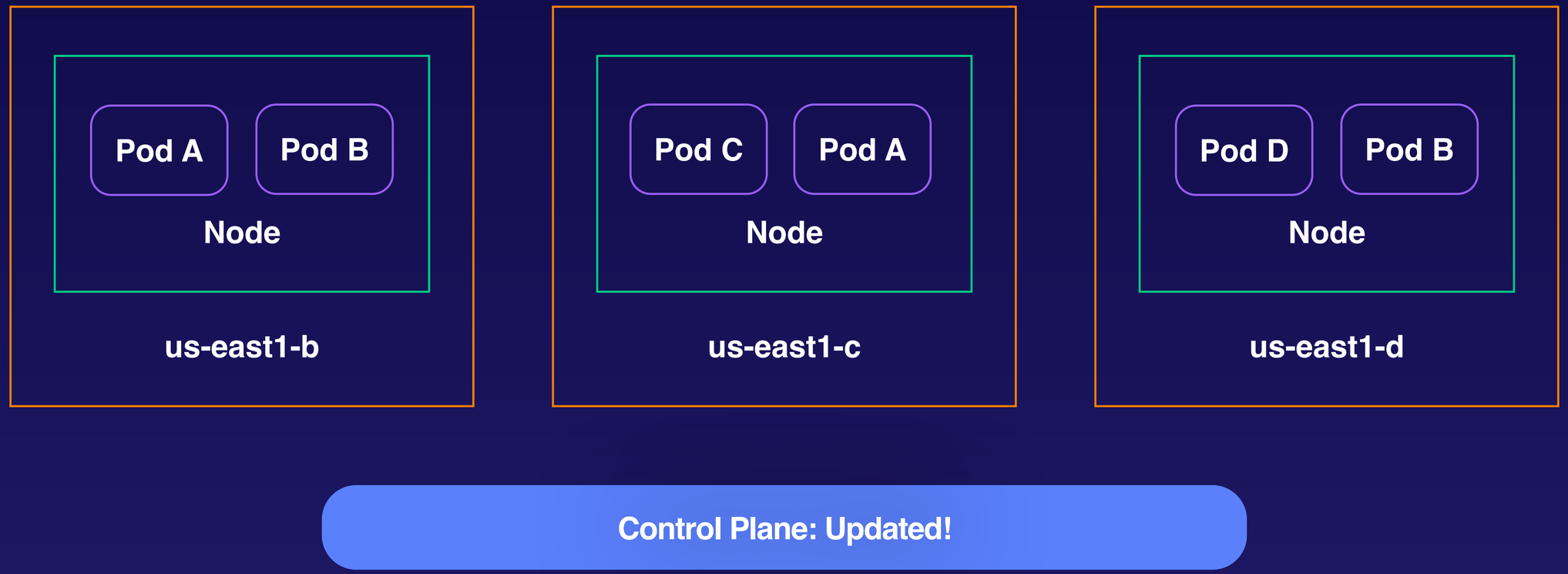
High availability clusters and workloads

Tim Berry
GOOGLE CLOUD GURU

High availability clusters and workloads

Regional GKE Clusters

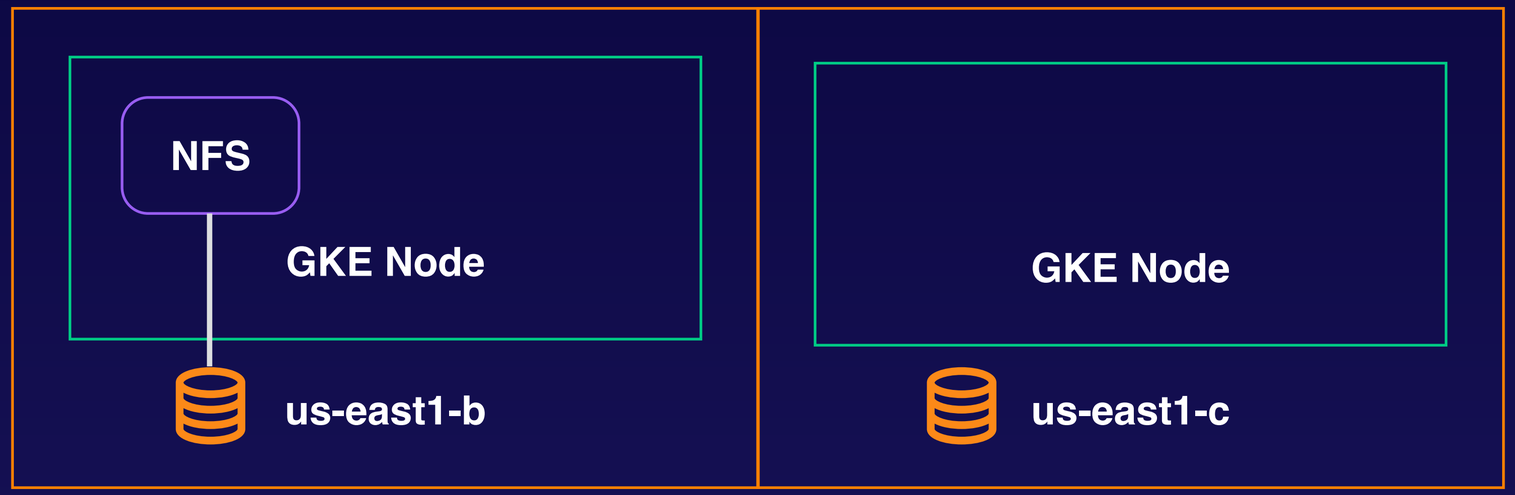
GCP REGION: US-EAST1



Regional Persistent Storage

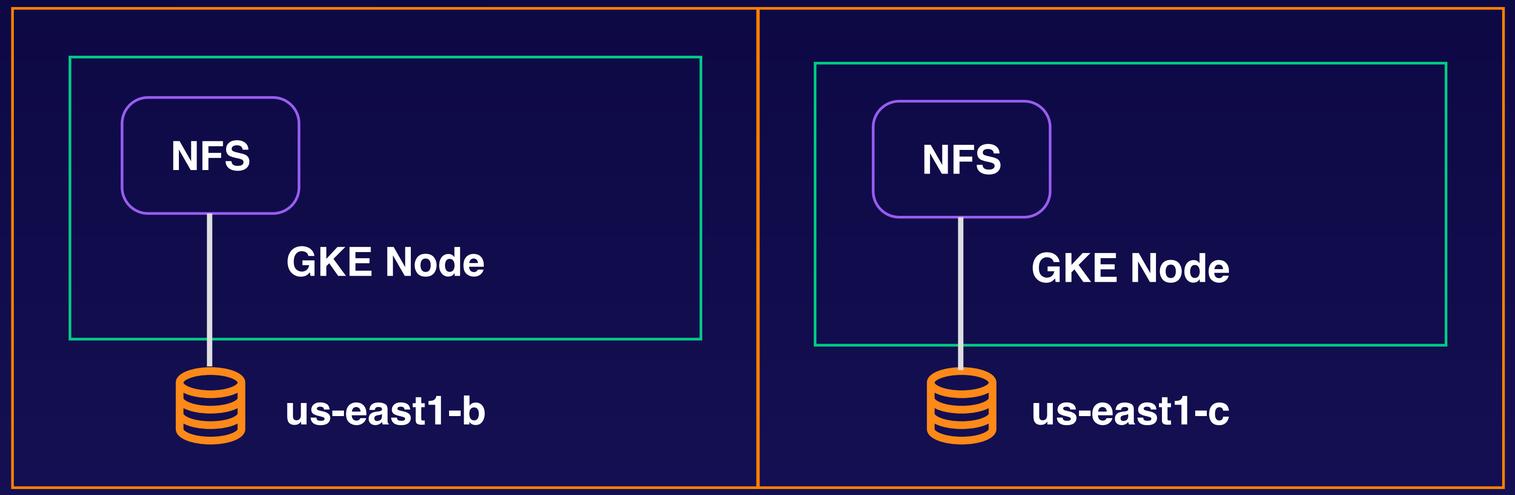
- GCP **persistent disks** are **zonal** resources
- **Stateful** Deployments can schedule a **Pod** and **disk** in the **same zone**
- GCP **regional persistent disks** replicate across **two zones**
- Use **nodeAffinity** and **nodeSelectorTerms**
- Schedule Pods and disks in the **same zone**

High availability clusters and workloads




```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
    nodeSelectorTerms:
      - matchExpressions:
        - key: “failure-domain.beta-kubernetes.io/zone"
          operator: In
          values: [“us-east1-b”, “us-east1-c”]
```

High availability clusters and workloads




```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
    nodeSelectorTerms:
      - matchExpressions:
        - key: “failure-domain.beta-kubernetes.io/zone"
          operator: In
          values: [“us-east1-b”, “us-east1-c”]
```

Multi-cluster Ingress

- Replicate application deployments across **multiple** GKE **clusters**
- Distribute workloads to reduce **blast radius**
- **kubemci** creates a Global Google Cloud Load Balancer
- Direct traffic to worldwide regions based on **lowest latency**
- Currently in **beta** with **no SLA**

A CLOUD GURU

Lab: Global load balancing with multi-cluster Ingress

Tim Berry
GOOGLE CLOUD GURU

Lab: Global load balancing with multi-cluster Ingress

Lab objectives:

- Provision **GKE** clusters from the **command line**
- Deploy the **zone-printer** demo application
- Configure multi-cluster **Ingress**

A CLOUD GURU

Running a secure GKE cluster

Tim Berry
GOOGLE CLOUD GURU

Cloud Native Security

- **Cloud**
  - VPC and IAM configuration, other GCP resources
- **Cluster**
  - Securing your GKE cluster
- **Container** and **code**
  - Trusted container images, secure code, versions and updates

A CLOUD GURU

Securing your cluster

- Role-based access control (**RBAC**)
- **Namespaces** and **resource** restrictions
- **Pod** security policies
- **Network** policies
- **Workload** identity

Role-Based Access Control

- **Granular** method of regulating **access** to cluster **resources**
- Can be applied at a **namespace** or **cluster** level
- Grants a set of **actions** to specific API **groups** and **resources**
- Can be used with Pod **service accounts**

Running a secure GKE cluster


```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-and-pod-logs-reader
rules:
- apiGroups: [""]
  resources: ["pods", "pods/log"]
  verbs: ["get", "list"]
```

- Grants **get** and **list** verbs on the **pods** and **pods/log** resources
- No API group restrictions specified
- Only applied to the default namespace

Running a secure GKE cluster


```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: monitoring-endpoints
rules:
- apiGroups: [""]
  resources: [“services”, “endpoints”, “pods"]
  verbs: ["get", “list”, “watch"]
```

- Grants **get**, **list**, and **watch** verbs on the **services**, **endpoints**, and **pods** resources
- No API group restrictions specified
- A ClusterRole applies across all namespaces

Namespaces

- **Virtual clusters** used to isolate resources for multiple **teams** or **projects**
- Can divide cluster resources with **resource quotas**
- **default** and **kube-system**
- A **scope** for resource names
  - `myservice.mynamespace.svc.cluster.local`

Running a secure GKE cluster


```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: team-rocket
```

- Define a namespace in YAML

Running a secure GKE cluster


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  namespace: team-rocket
```

- Reference the namespace in object metadata

Running a secure GKE cluster


```
$ kubectl apply mypod.yaml

$ kubectl -n team-rocket get pods

$ kubectl -n kube-system get pods
```

Pod Security Policies

- Set of **controls** that **locks down** security-sensitive aspects of a Pod specification
- Policies define **conditions** that a Pod spec **must** meet in order to be **scheduled**
- Enforced by the optional **admission controller**
- Requires the creation of a **Role** or **ClusterRole** with **permission** to use the policy
- Requires a **user** or **service account** to be bound to the Role
- By default, Pods will **not** be **scheduled**

Running a secure GKE cluster


```yaml
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: noprivpods
spec:
  privileged: false
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  runAsUser:
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
  volumes:
  - '*'
```

- Prohibits the scheduling of Pods that contain privileged containers
- Some fields are required for any PodSecurityPolicy

Running a secure GKE cluster


```
$ kubectl apply -f mypod.yaml

Error creating: pods “mypod” is forbidden:
unable to validate against any security
policy: [].
```

- Default cluster user does not operate with a Role or ClusterRole that is permitted to use a PodSecurityPolicy

Network Policies

- An object that defines network **ingress** and **egress** rules for **Pods**
- Use **selectors** to restrict incoming and outgoing **traffic**
- Can **isolate** traffic between **namespaces**
- Reduce **risk** of application **vulnerabilities** affecting other cluster services
- Should be **enabled** when the GKE cluster is **built**

Running a secure GKE cluster


```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: login-service-from-fe
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: login-service
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend-service
```

- Policy is applied to all Pods matching: `app=login-service`
- Only permits ingress from Pods matching: `app=frontend-service`

Running a secure GKE cluster


```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: outbound-dns-only
spec:
  policyTypes:
  - Egress
  podSelector:
    matchLabels:
      app: restricted
  egress:
  - ports:
    - port: 53
      protocol: TCP
    - port: 53
      protocol: UDP
```

- Policy is applied to all Pods matching: `app=restricted`
- Only permits egress on TCP and UDP port 53

Workload Identity

- A **secure** way for **GKE** workloads to consume **GCP** services
- Compute Engine **default service account** used by GKE nodes
- Map custom **GCP** service accounts to **GKE** workloads
- **Kubernetes** service accounts can also be mapped to **GCP** service accounts
- Currently in **beta**

A CLOUD GURU

Enhancing cluster nodes with DaemonSets

Tim Berry
GOOGLE CLOUD GURU

DaemonSets

- A workload **object** for managing groups of replicated **Pods**
- **One Pod** per **Node** model - across the entire cluster, or a subset of nodes
- If new **nodes** are added, new **Pods** are automatically created
- Very **specific** use cases

Enhancing cluster nodes with DaemonSets

Use cases

- Any scenario where workloads require access to a service **on every node**
- **Storage** daemons
- **Logs** collection services
- **Monitoring** daemons
- Custom **drivers**

Enhancing cluster nodes with DaemonSets


```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd
spec:
  selector:
      matchLabels:
        name: fluentd
  template:
    metadata:
      labels:
        name: fluentd
    spec:
      nodeSelector:
        type: prod
      containers:
      - name: fluentd
        image: gcr.io/google-containers/fluentd-elasticsearch:1.20
```

A CLOUD GURU

Stateful applications and workloads

Tim Berry
GOOGLE CLOUD GURU

Stateless vs Stateful

- **Stateless**
  - Pods can be **added**, **removed**, or **restarted** at will
  - Individual Pods have no data to **persist**, or concept of **state**
- **Stateful**
  - Pods have **data** to store and need a **persistent identity**
  - The **deployment** and **scaling** of Pods must be logically **managed**

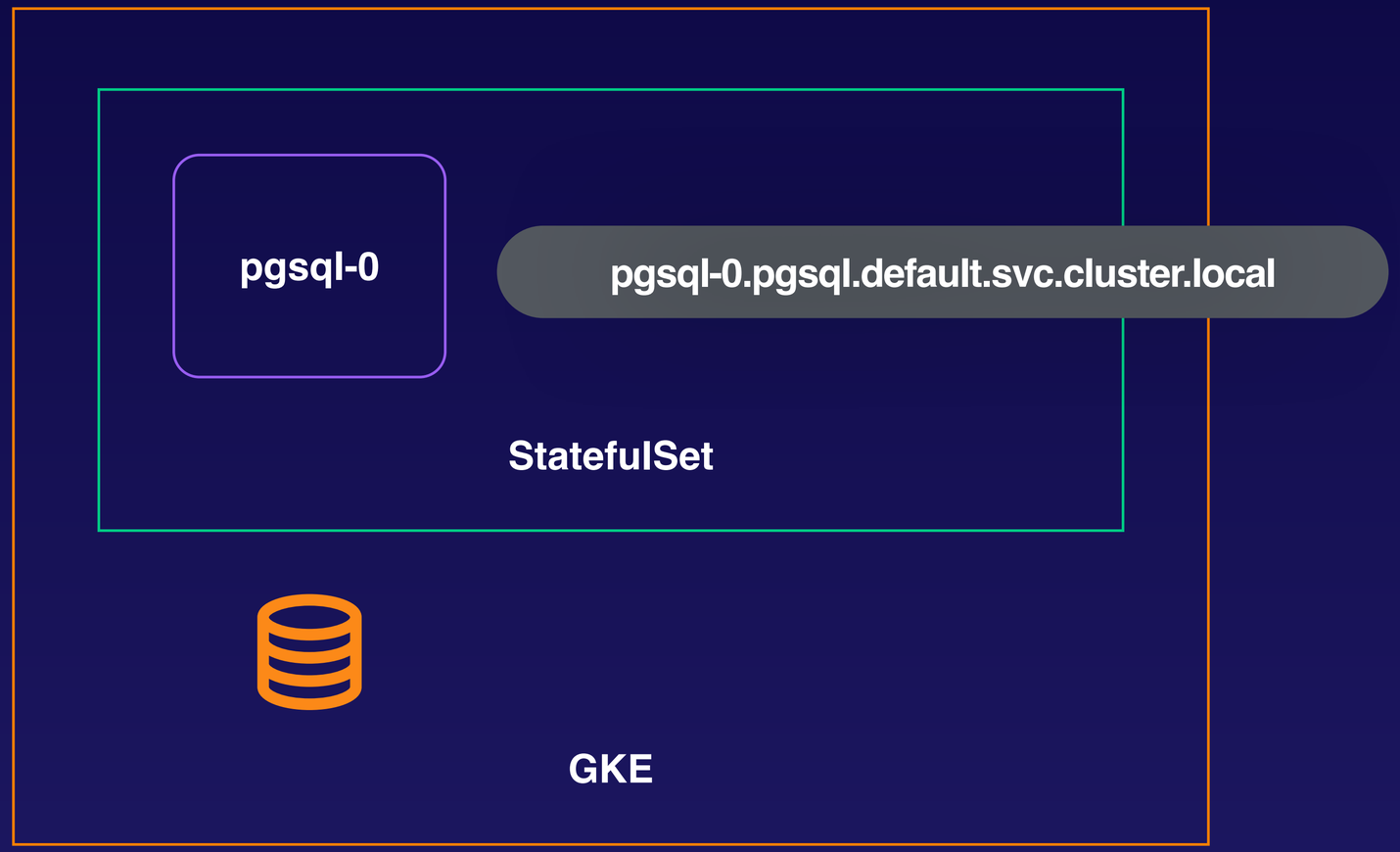
StatefulSets

- Manages **Pods** based on a container **spec**, much like a Deployment
- Maintains an **identity** for each deployed **Pod**
- Guarantees the **ordering** and **uniqueness** of Pods
- Stable **network identity** and persistent **storage**
- Ordered, graceful **deployment**, **scaling**, and **updates**

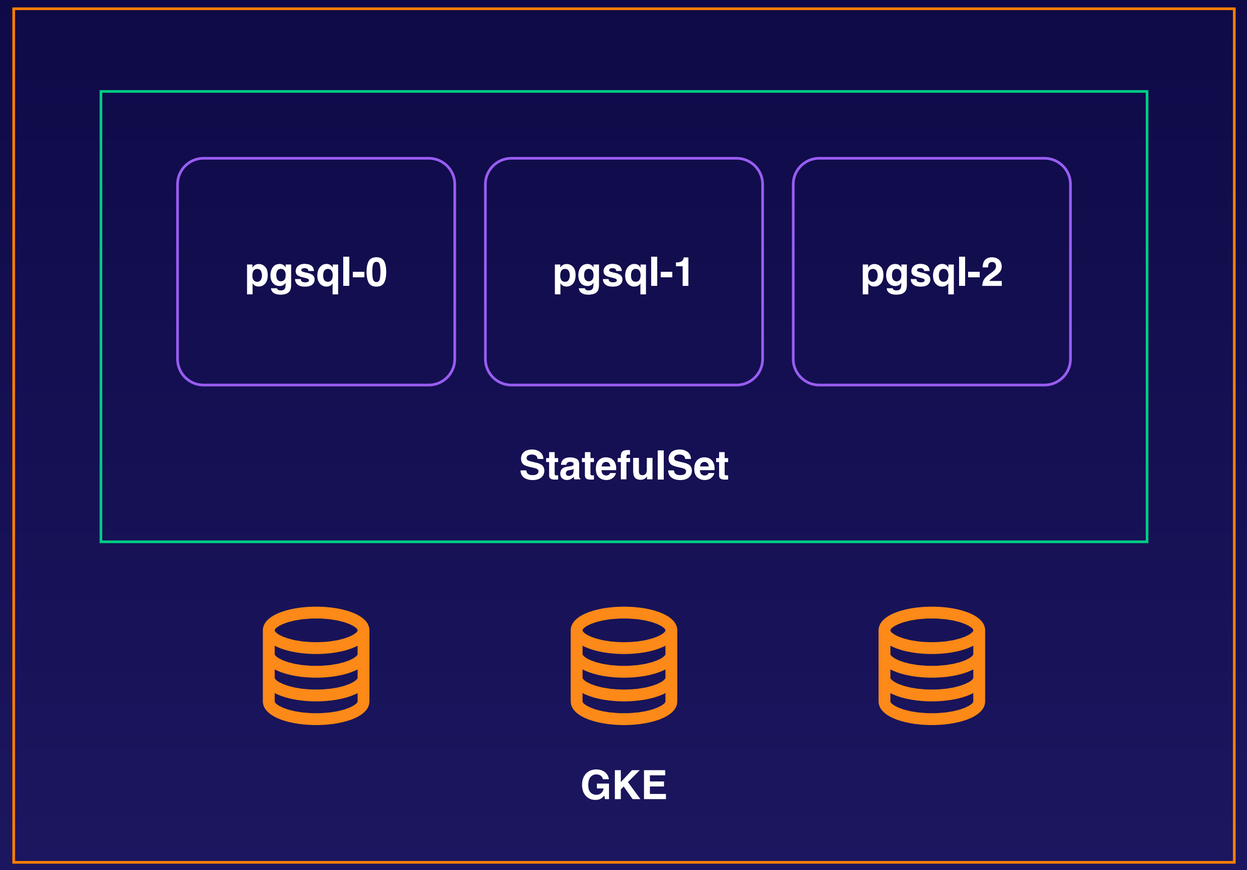
Stateful applications and workloads


```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: pgsql
spec:
  selector:
    matchLabels:
      app: pgsql
  replicas: 3
  serviceName: pgsql
  template:
    metadata:
      labels:
        app: pgsql
    spec:
      containers:
      - name: pgsql
        image: example.gcr.io/postgres:9.6
        ports:
        - containerPort: 5432
          name: pgsql
        volumeMounts:
        - name: pgdata
          mountPath: /var/lib/pgsql
```

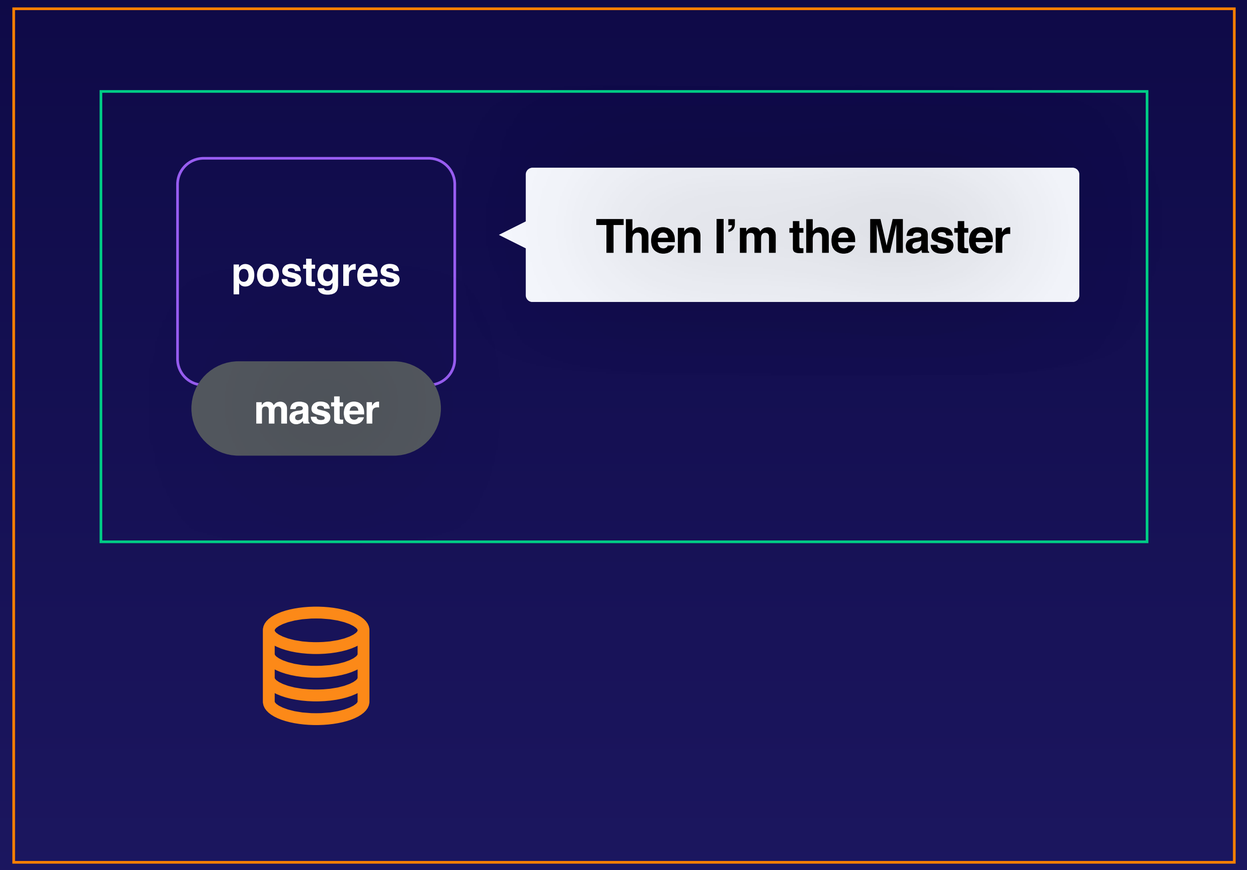
Stateful applications and workloads



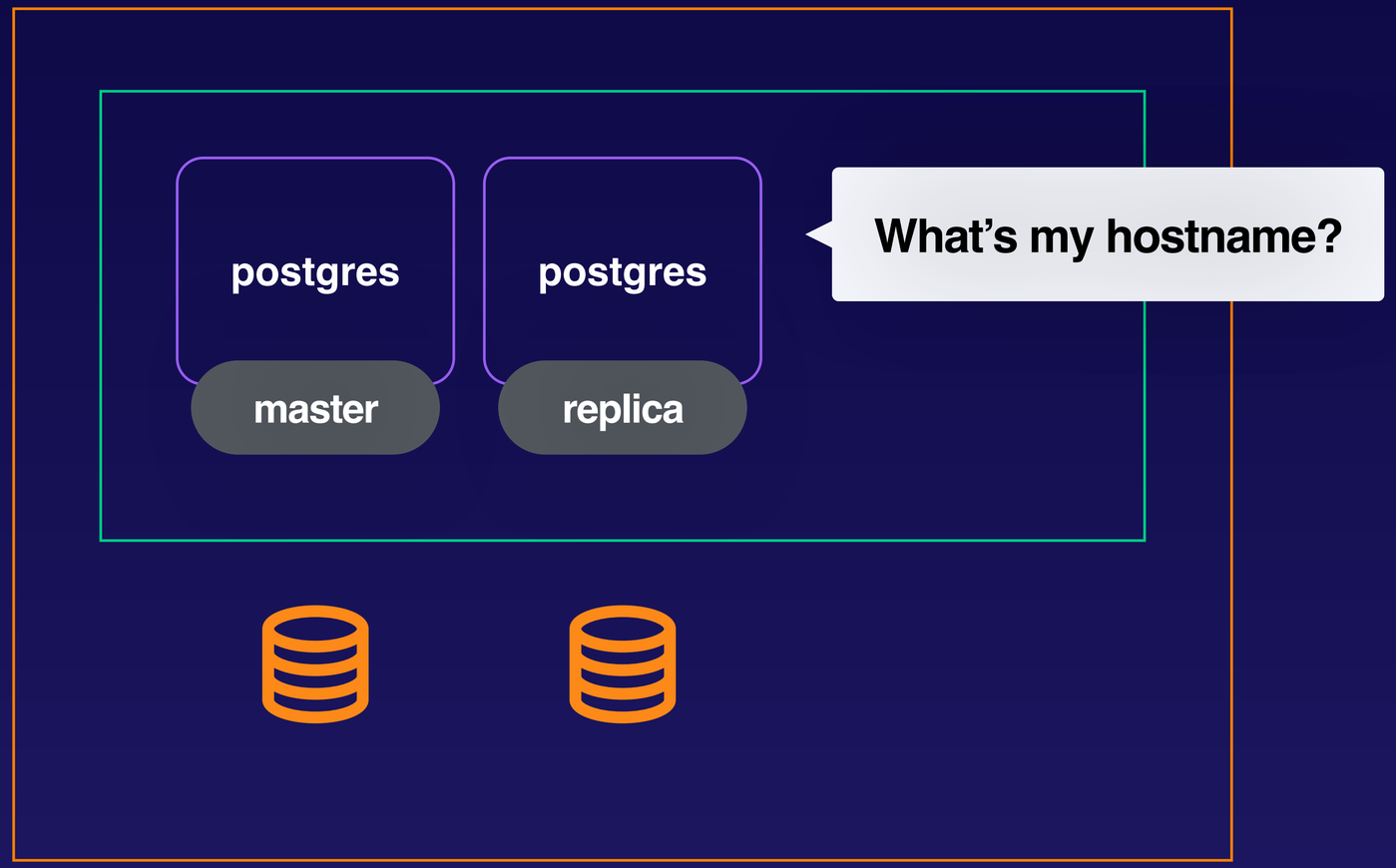
Stateful applications and workloads



Stateful applications and workloads



Stateful applications and workloads



Stateful applications and workloads

A CLOUD GURU

Pod Management Policies

- **Ordered** Pod management is the **default**
- **Parallel** Pod management is an **option**

A CLOUD GURU

Update Strategies

- **Rolling updates** delete and recreate Pods in descending order
- The scheduler waits for a Pod to be **running** and **ready** before proceeding
- Updates can also be **partitioned**
- **Manual** updates can be **enforced** with the **OnDelete** strategy

A CLOUD GURU

Finite tasks and init containers

Tim Berry
GOOGLE CLOUD GURU

A CLOUD GURU

Jobs and CronJobs

- A workload **object** that represents a finite task and manages it to completion
- A **non-parallel** job usually starts **one Pod** and is complete when the Pod terminates successfully
- A **parallel** job may have a fixed **completion** count running **multiple** Pods to completion
- Parallelism is **configurable**
- Jobs can be **scheduled** to be reoccurring with **CronJobs**

Finite tasks and init containers


```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: print-pi
spec:
  template:
    metadata:
      name: print-pi
    spec:
      containers:
      - name: print-pi
        image: perl
        command: ["perl"]
        args: ["-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
```

Finite tasks and init containers


```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: process-uploads
spec:
  completions: 8
  template:
    metadata:
      name: process-uploads
    spec:
      containers:
      - name: processor
        image: gcr.io/example-project/processor:1.12
        command: ["run"]
        args: ["--from-bucket", "mybucket", "--num", "500"]
      restartPolicy: Never
```

Finite tasks and init containers


```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: process-uploads
spec:
  completions: 8
  parallelism: 4
  template:
    metadata:
      name: process-uploads
    spec:
      containers:
      - name: processor
        image: gcr.io/example-project/processor:1.12
        command: ["run"]
        args: ["--from-bucket", "mybucket", "--num", "500"]
      restartPolicy: Never
```

Finite tasks and init containers


```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: process-uploads-schedule
spec:
  schedule: "0 0 * * 6"
  jobTemplate:
    spec:
      template:
        metadata:
          name: process-uploads
        spec:
          containers:
          - name: processor
            image: gcr.io/example-project/processor:1.12
            command: ["run"]
            args: ["--from-bucket", "mybucket", "--num", "500"]
          restartPolicy: Never
```

Finite tasks and init containers

Init Containers

- Part of the **containers** array in a **Pod** spec
- Execute **before** application or other containers in a Pod
- Init containers run **in order** and **to completion**
- Can separate custom **start-up** code from an application image
- Can contain **logic** that **delays** the start-up of an application image

Finite tasks and init containers


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: myapp
    image: gcr.io/example-project/myapp
    volumeMounts:
    - name: myapp-files
      mountPath: /data
  initContainers:
  - name: init-files
    image: alpine/git
    command: ['git', 'clone', ‘https://github.com/myapp-files/git.git /data’]
    volumeMounts:
    - name: myapp-files
      mountPath: /data
  - name: init-mysql
    image: busybox
    command: ['sh', '-c', 'until nslookup mysql; do echo waiting for mysql; sleep 2; done;']
```

A CLOUD GURU

Lab: Running a stateful Cassandra database

Tim Berry
GOOGLE CLOUD GURU

Lab: Running a stateful Cassandra database

Lab objectives:

- Create a **standard 3-node** GKE cluster with **n1-standard-2** instances
- Authenticate **kubectl** with **gcloud**
- Deploy a **StatefulSet** to run **Cassandra**
- Use custom **security contexts**, resource **requests** and **limits**, readiness **probes** and custom **storage classes**
- Make a StatefulSet **production-ready** with **distributed** persistent **storage**

A CLOUD GURU

CHAPTER FIVE

Wrapping up

A CLOUD GURU

What have we learned so far?

Tim Berry
GOOGLE CLOUD GURU

What have we learned so far?

What's so special about Kubernetes?

- A distributed **cloud operating system** for large-scale computing
- Object abstractions for **workloads** and other **infrastructure** components
- Simplifies the **developer** experience
- Fully supports DevOps **software lifecycle**

What's so special about GKE?

- An advanced **production ready** managed Kubernetes platform
- Control plane **managed** by **Google** SREs
- Fast, flexible, and resilient **compute** powered by **GCP**
- Interoperable with all other **GCP products** and **services**

What have we learned so far?

Success stories

monzo

- A app-based startup bank with a **micro-services** core banking platform
- Moved to **Kubernetes** and benefited from **75% cost savings**
- Resilient **highly available** deployments with **chaos** engineering
- **Polyglot** codebases deployed to clusters across **multiple clouds**

Source: https://monzo.com/blog/2016/09/19/building-a-modern-bank-backend

What have we learned so far?

Success stories

- Processing data from sensors in the **Large Hadron Collider**
- Currently at **330 PB** and due to **10x** within the next few years
- Developers embraced **Kubernetes** and **Helm**
- Time to **deploy** for new applications **massively reduced**

Source: https://kubernetes.io/case-studies/cern/

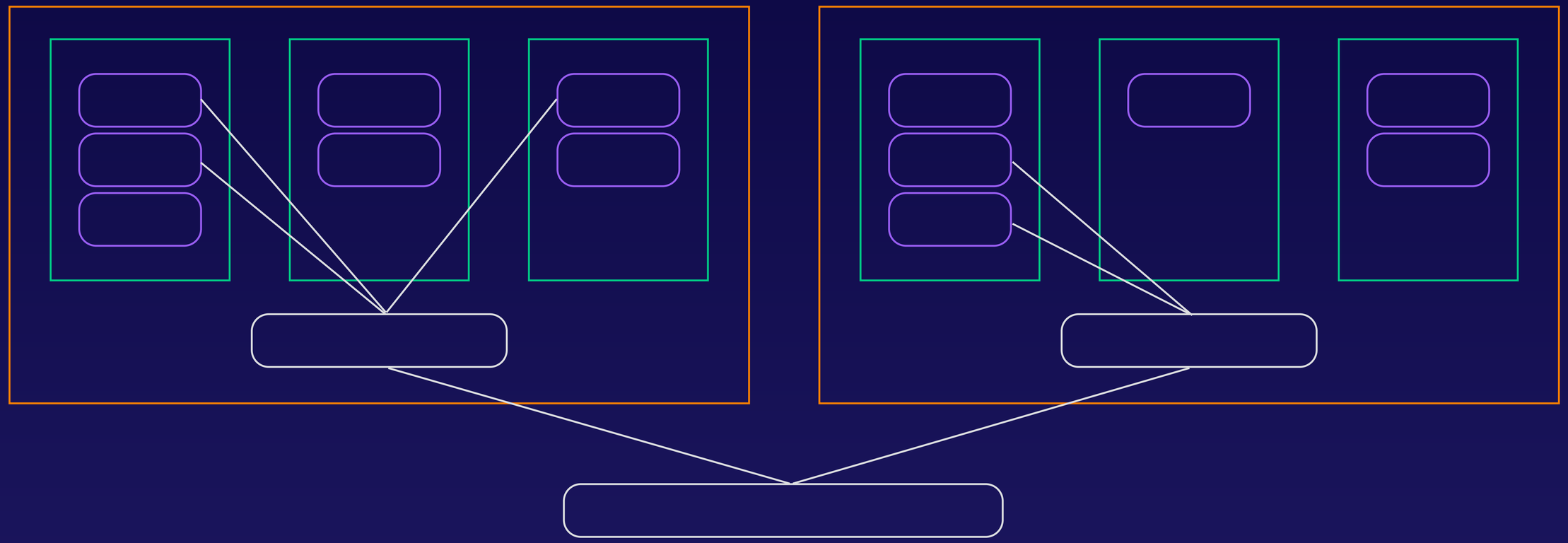
What have we learned so far?

Success stories

- Early adopters of **GKE** with the **largest** deployment ever at launch
- **10,000 cores** with **1000s** of new **nodes** added in flight
- Container-based development enables **continuous improvement**
- Directly **integrates** with GCP products: **Cloud Datastore** and **load balancing**

Source: https://cloud.google.com/blog/products/gcp/bringing-pokemon-go-to-life-on-google-cloud

What have we learned so far?



What have we learned so far?

Where do we go from here?

- Further **advanced** areas of interest to keep exercising your **GKE muscles**
- Continuous **Integration** and **Deployment** with **GitOps**
- **Operators**: an alternative way to package applications with built-in **logic**
- **Istio** and **Service Meshes**
- **Serverless**
- The **future** of **GKE**

A CLOUD GURU

Continuous integration and deployment

Tim Berry
GOOGLE CLOUD GURU

Source control

- Source control provides a **history** of **code** development and enables **collaboration** across multiple developers and teams
- **Modern** application **development** relies on source control - usually **Git**
- **GitOps** extends this model by adding operations to the commit model
- Activities in Git (commits) can trigger **operations** such as image **building** and even application **deployments**

Continuous integration (CI)

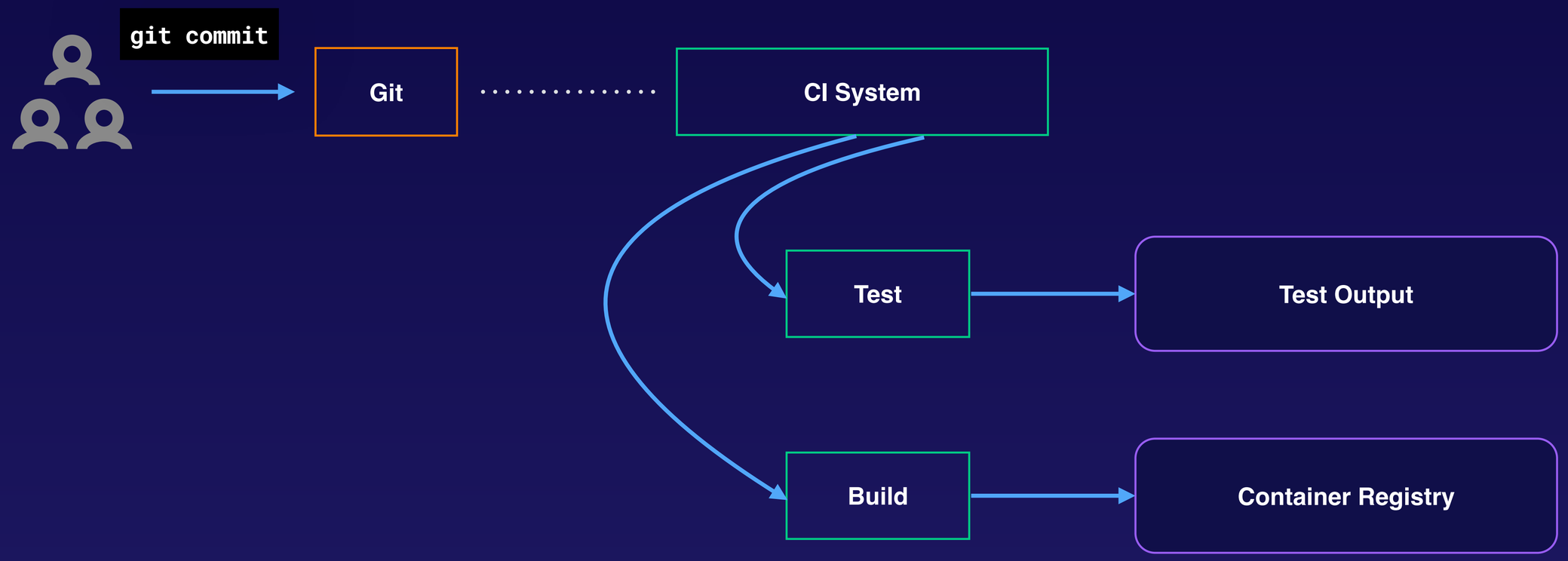
- Automate the **test** and **build** of code every time a change is **committed**
- Post-commit **hooks** can trigger CI systems
  - Perform **tests**
  - Execute docker image **builds**
- **Notify** development teams of test and build outcomes

Continuous deployment (CD)

- The next logical step in **automation** after CI
- Application updates, once tested and built, are **automatically deployed**
- Updates can be made with many types of **deployment pattern:**
  - Rolling update, blue/green
  - Development namespaces / node pools / clusters / projects
- CI/CD systems can be **fully automated** or require some **human interaction**

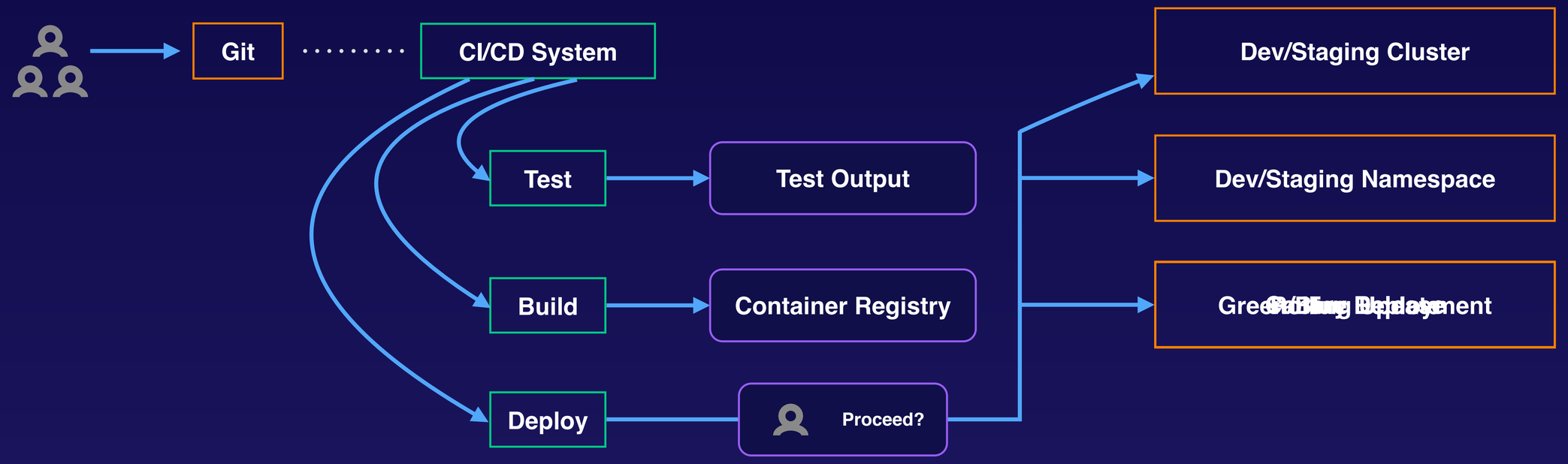
Continuous integration and deployment

GITOPS: CI



Continuous integration and deployment

GITOPS: CI/CD



CI/CD systems

- **Jenkins** is the most widely adopted CI system
- Hundreds of **plugins** and **integrations** available
- However, it can be **clunky** to install and **maintain**
- **GitLab** - free online or self-hosted
- Source control with bundled **DevOps toolchain**
- CircleCI, TravisCI, GitHub, Google Cloud Build

A CLOUD GURU

Operators: Adding logic to deployments

Tim Berry
GOOGLE CLOUD GURU

The promise of orchestrated containers

- Containers offer the holy grail of write **once**, run **anywhere**
- Kubernetes deployments are **hassle-free** for **stateless** applications
- Operations overhead can **increase** when **stateful** application containers:
  - **Scale** up and down according to **demand**
  - Perform **backup** and **restore** operations
  - Respond to cluster events like **failed nodes**

Using StatefulSets can get you half way there

- The logic of **StatefulSets** can help
  - **Ordering** of Pods
  - **Persisting** storage
- Each Pod is still based on an **immutable** container image
- Application **logic** needs to run **inside** that container

Operators provide automated logic

- Bundle **operational knowledge** to streamline and **automate** common operational functions
- Add new **custom resources** to Kubernetes
  - Custom **objects** that can be deployed with built-in logic
  - Custom **controllers** that can automate common events

Operators extend Kubernetes

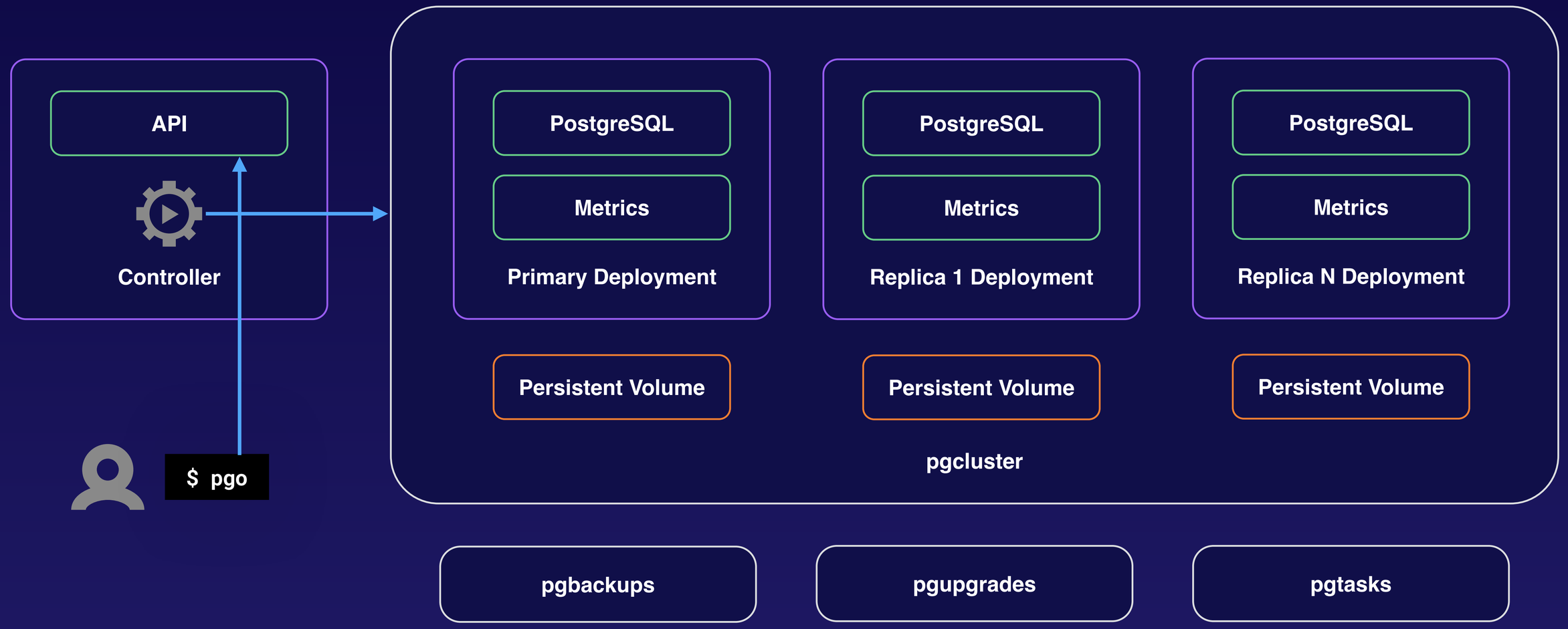
- Custom Resource Definitions (CRDs)
  - Custom **resources** are sets of structured **data**
  - Custom **controllers** are **applications** that provide an API and enforce a **declarative** state in the cluster
- Operators are gaining **traction** in the Kubernetes community
- Not all are production ready - **understand** what they are doing under the hood!

Operators: Adding logic to deployments

Operators: Adding logic to deployments

POSTGRES OPERATOR

https://github.com/CrunchyData/postgres-operator



A CLOUD GURU

Istio and service meshes

Tim Berry
GOOGLE CLOUD GURU

A CLOUD GURU

Microservices at scale

- Numerous **benefits** of container-driven **microservices**
- Kubernetes **automates** the **deployment** of applications
- Managing dozens - even **hundreds** - of microservices becomes **problematic**
  - Traffic management
  - Reliability and visibility
  - Dependency management

The service mesh solution

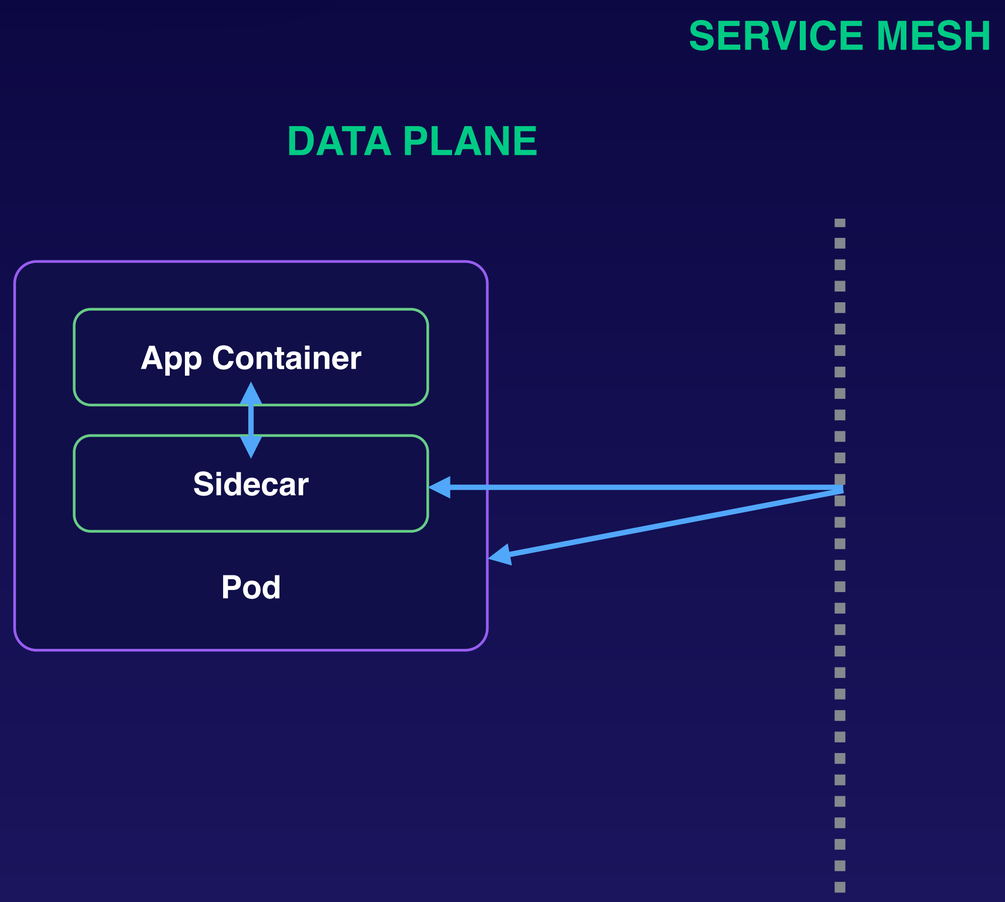
- Centralizes **communication** and **control** across all microservices
- Data plane:
  - A **sidecar** container running in every Pod that manages communication
- Control plane:
  - A deployed set of applications for **controlling** microservices

A CLOUD GURU

Data plane

- A **sidecar** container running the **Envoy** proxy
- Controls **network traffic** in and out of the **Pod**
- Communicates with the **control plane** to retrieve **routing** logic and send **metrics**

Istio and service meshes

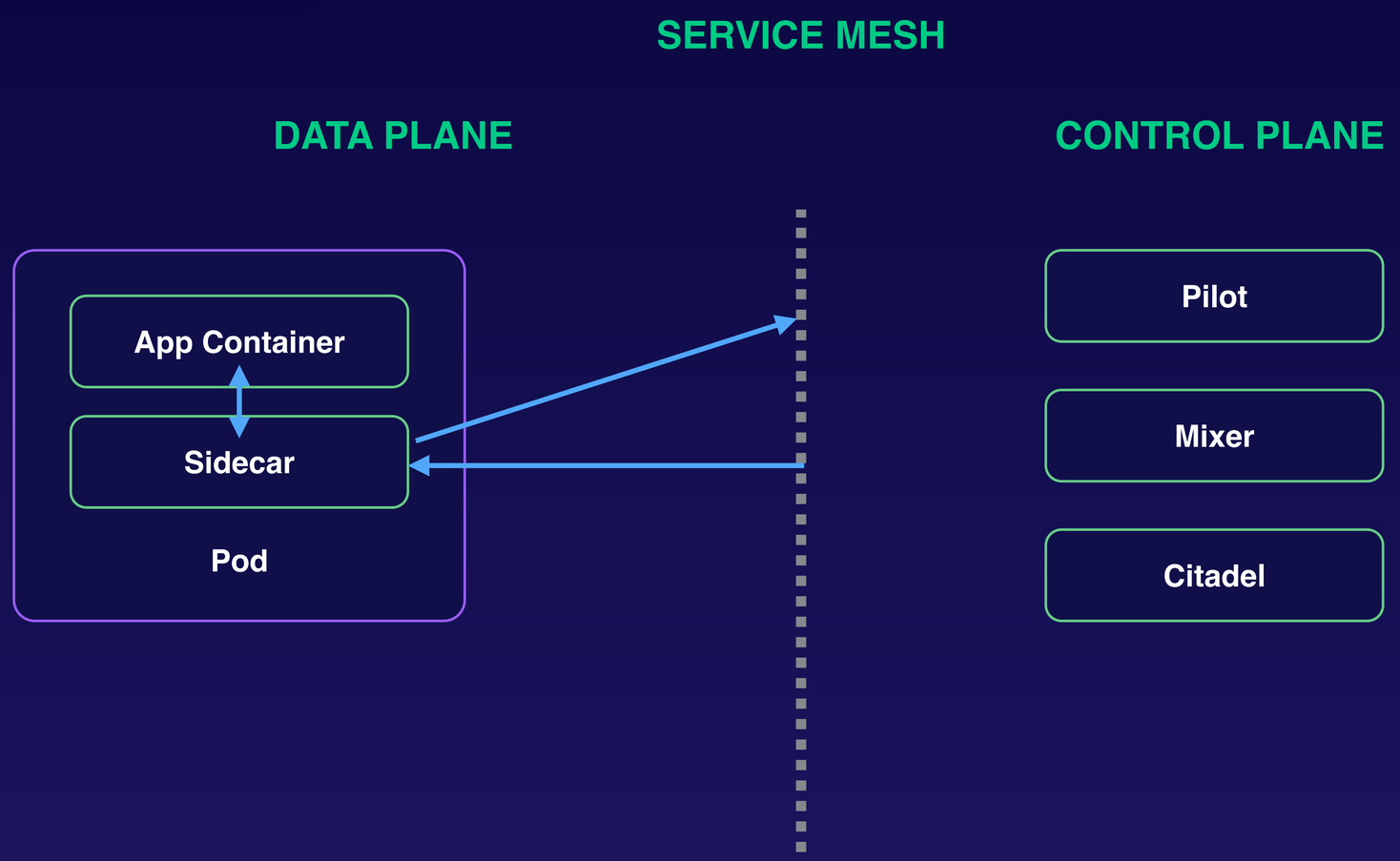


A CLOUD GURU

Control plane

- **Pilot** - configures the data plane, defines proxy rules and behaviors
- **Mixer** - collects traffic metrics and responds to authorization, access control, or quota checks
- **Citadel** - assigns TLS certificates to each service and enables end-to-end encryption

Istio and service meshes



A CLOUD GURU

Installing and operating Istio

- **Enabled** as part of a **GKE** installation
- Installed via **Helm** chart
- Installs multiple **deployments** into the **istio-system** namespace
- Istio rules and functions are **Custom Resources** in Kubernetes
- Declarative **manifests** applied to the cluster with **YAML**

A CLOUD GURU

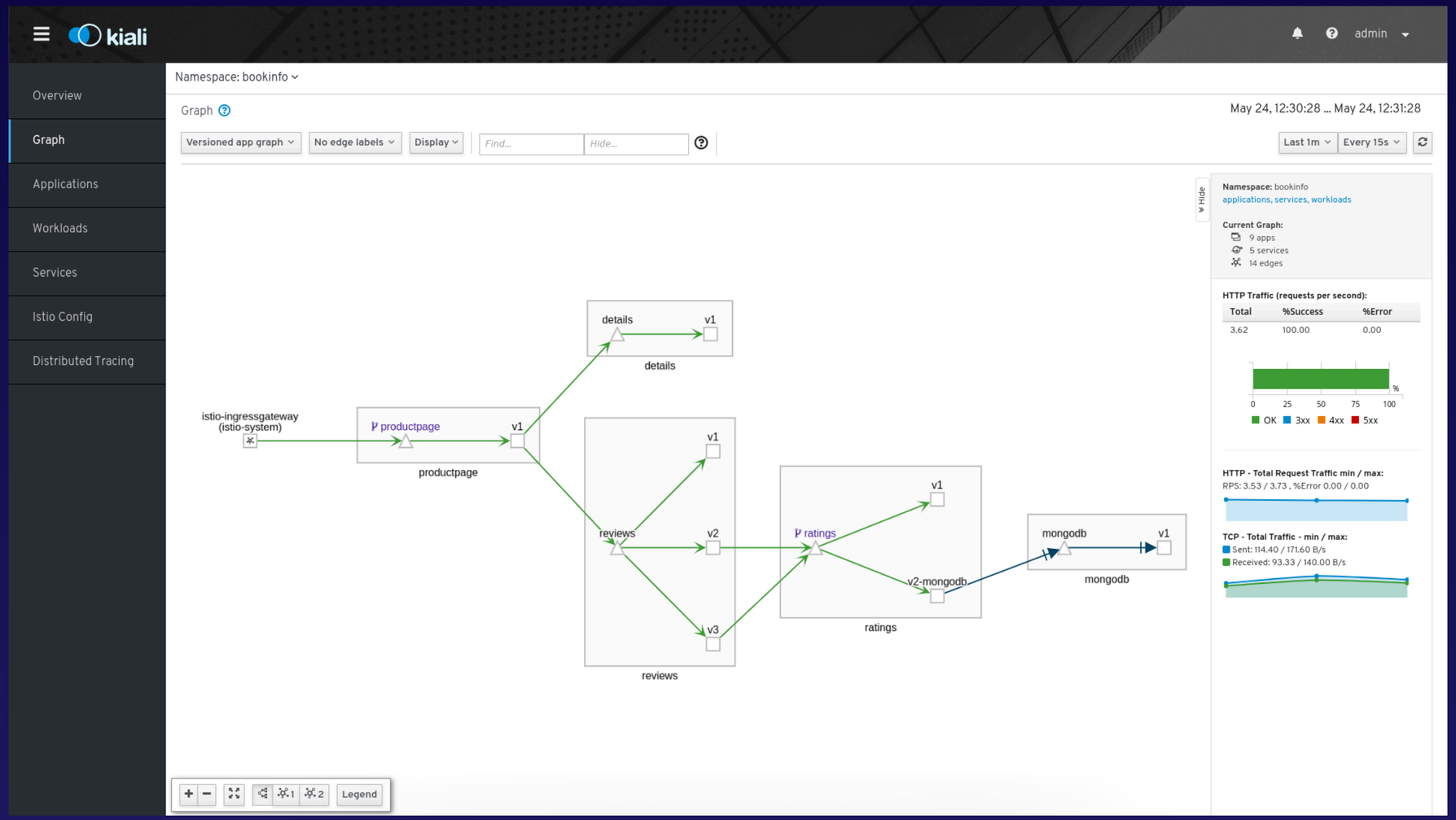
Istio features

- **Traffic** management
- **Security**
- **Telemetry**
- **Visualization**

Do I really need a service mesh?

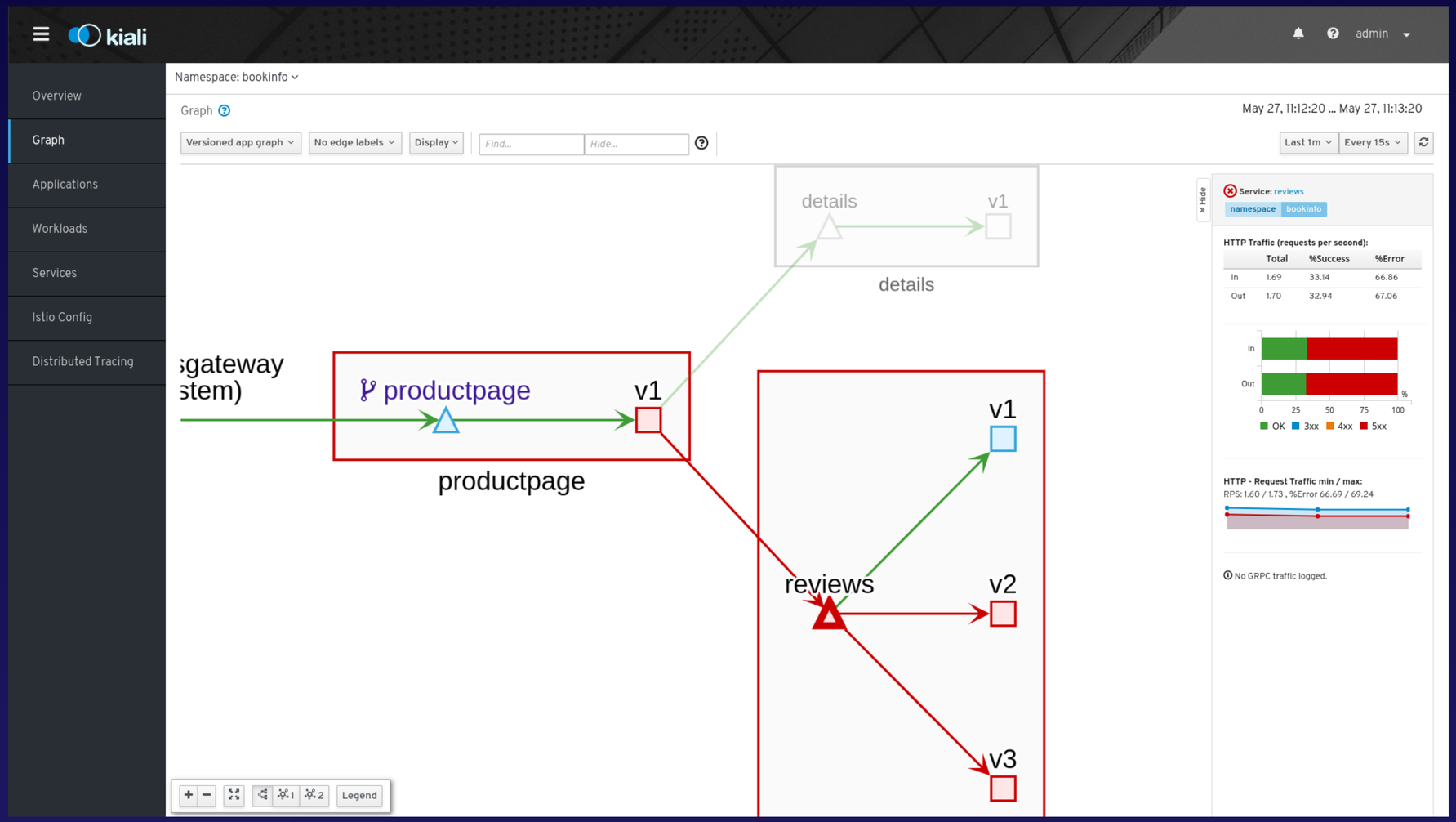
- Probably not, **at first**
- **Visualizing** a microservices topology becomes **essential** when dealing with multiple services **at scale**
- **SRE teams** responsible for application SLOs will require microservice response and latency metrics
- **Complex** problems can be troubleshooted with **distributed tracing**

Istio and service meshes



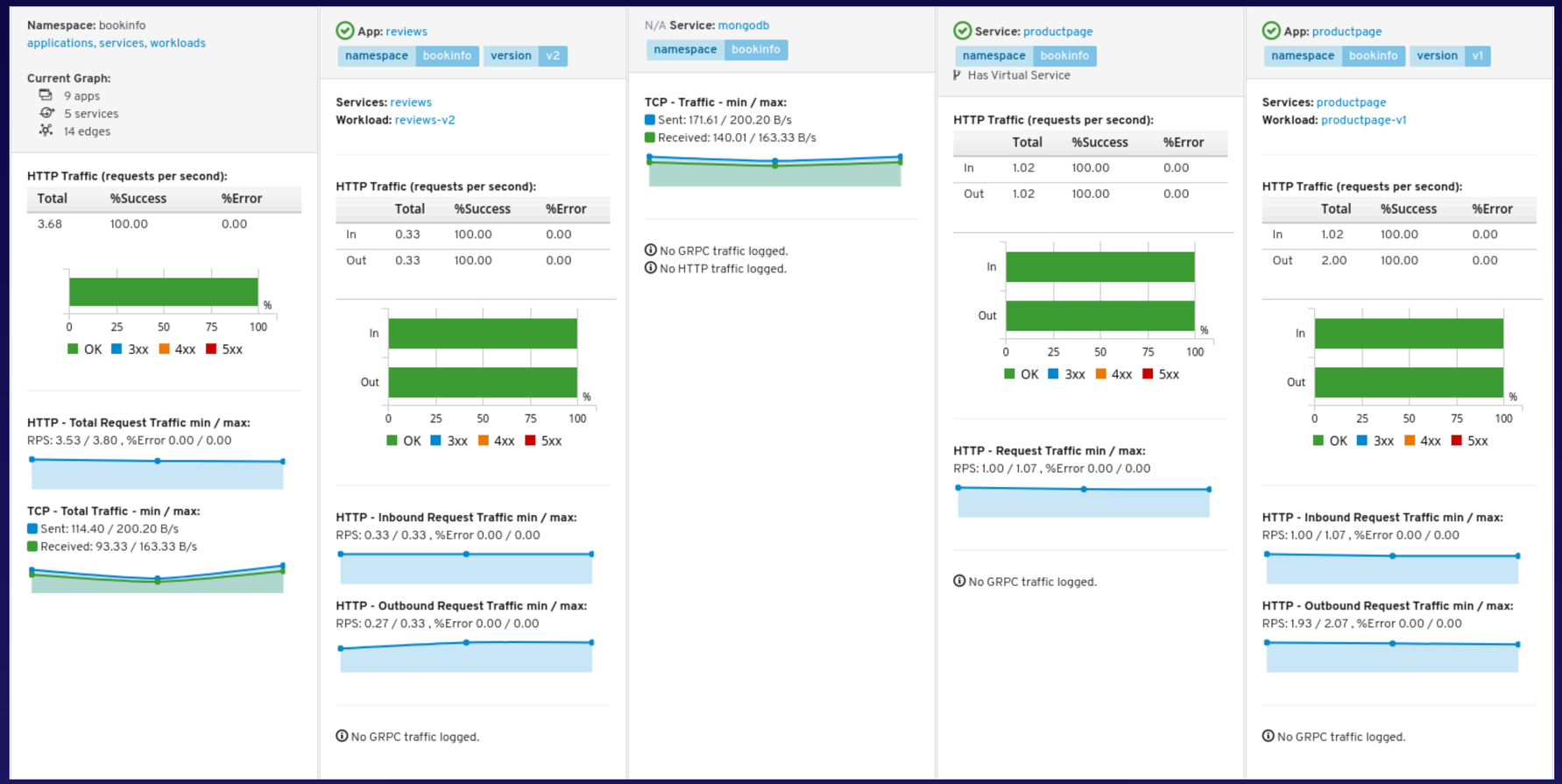
Source: https://www.kiali.io/documentation/features/

Istio and service meshes



Source: https://www.kiali.io/documentation/features/

Istio and service meshes



Source: https://www.kiali.io/documentation/features/

A CLOUD GURU

Alternatives to Istio

- Google **Cloud Service Mesh**
- **Linkerd**
- Standalone **Envoy**

A CLOUD GURU

Serverless on Kubernetes

Tim Berry
GOOGLE CLOUD GURU

What does serverless mean?

- A method of deploying code that completely **abstracts servers away**
- Arguably even more **developer friendly** than Kubernetes! 😱
- Containers as a Service: **Deploy** a **container** and let the platform automate everything else
  - Google **Cloud Run** or AWS **Fargate**
- Functions as a Service: **Snippets** of **code** run on demand
  - Google **Cloud Functions** or AWS **Lambda**

How does this relate to Kubernetes?

- Kubernetes is a distributed **platform** for **all** of your infrastructure
- **New projects** may embrace cloud-native paradigms and be built entirely with **serverless functions**
- You will still likely have to support **multiple** teams and application stacks with **different** deployment **demands** and **requirements**
- Kubernetes can run serverless **functions** and serverless **containers**, so your overall platform is still a **single** large distributed **system**

Serverless containers with Knative

- A deployment method for **developers** in the style of ‘click to deploy’
- Knative **extends** Kubernetes with new Custom Resources
- A container image is simply **built** and **deployed**
- Knative handles **networking**, **revision tracking** and **autoscaling**
- **Cloud Run** is a Knative-compatible implementation as a **managed** service
- Knative can be installed **manually** or as a ‘Cloud Run on GKE’ **package**

Serverless on Kubernetes

- Quick screen recording of Knative demo (not a lab)

Serverless functions with OpenFaaS

- An **open source** FaaS platform that runs on Kubernetes
- Deploy functions as **minimal** docker **containers**
- HTTP requests **abstracted** through a function watchdog
- Not as lightweight as Google **Cloud Functions**, but much more **customizable**

Serverless on Kubernetes

- Quick screen recording of OpenFaaS demo (not a lab)

A CLOUD GURU

What’s next for GKE?

Tim Berry
GOOGLE CLOUD GURU

Ongoing GKE development

- GKE versions close to releases from **upstream** Kubernetes
  - Latest **Kubernetes** 1.15.3, 1.14.6 - Latest **GKE**: 1.14.5
- **Native** Kubernetes **functionality** should always be available in GKE
- Custom Google **enhancements** moving to a **new tier**

What's next for GKE?

GKE Advanced

- GKE **Sandbox** based on **gVisor**
- Binary **authorization** and **shielded VM** nodes
- **Cloud Run** on GKE
- **Enhanced** Node Auto Provisioning
- New features are still being **announced** and **pricing** is **TBD**

What's next for GKE?

Anthos

- Google's big push into **hybrid cloud** and **enterprise**
- GKE **On-Prem**
- Multi-cluster **management** and **service mesh**
- Declarative **cluster config** management
- **VM** to **Container** migration
- Requires a monthly **subscription fee**