AWS re:Invent

CON212-R

# Running Kubernetes at Amazon scale using Amazon EKS

**Kevin Dillane**
Sr. Software Engineer
Amazon.com

**Cristian Munoz Urbancic**
Sr. Software Engineer
Amazon.com

**Bob Wise**
General Manager
Elastic Kubernetes Service (EKS)
Amazon Web Services

AWS re:Invent



aws

# Agenda

About velocity

From monoliths to microservices: The rendering problem

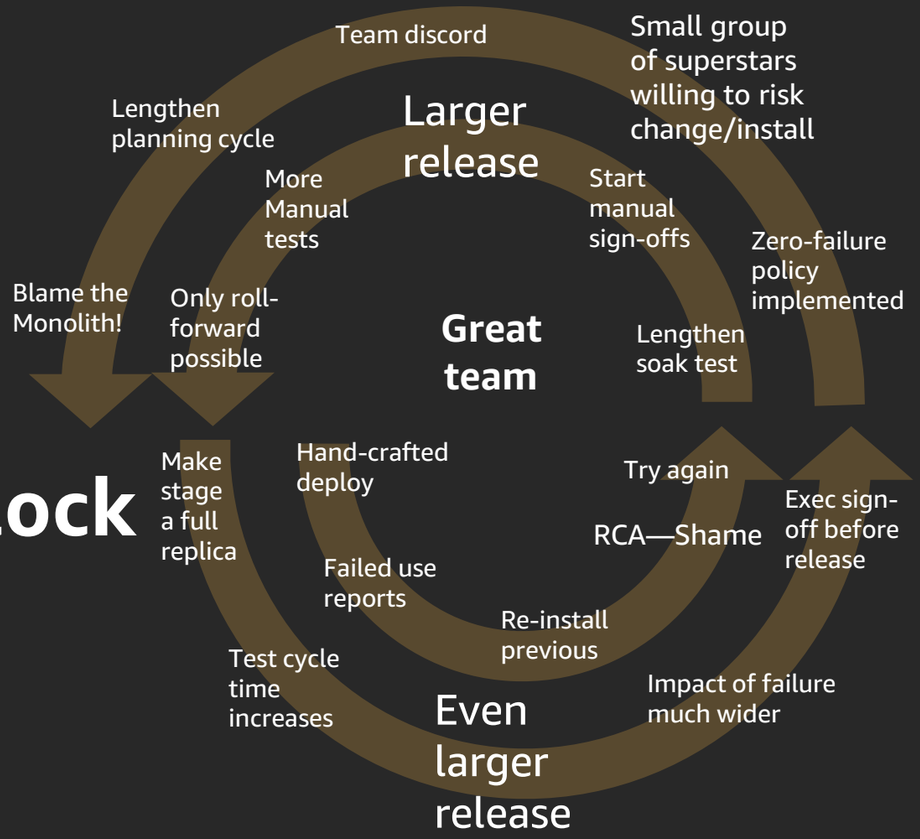
Improving ownership by owning less

CI/CD: Release features all the time!

How we learned to stop worrying and ❤️Amazon scale

# About velocity

# Good intentions… Painful result




Spiral to slowness

# Path to higher velocity…

**First steps…**


Virtuous flywheel

# Evolved teams

**5x**

Lower change failure rate

**440x**

Faster from commit to deploy

**46x**

More frequent deployments

**44%**

More time spent on new features & code

Source: Puppet 2017 State of DevOps Report

# Containers on hosts

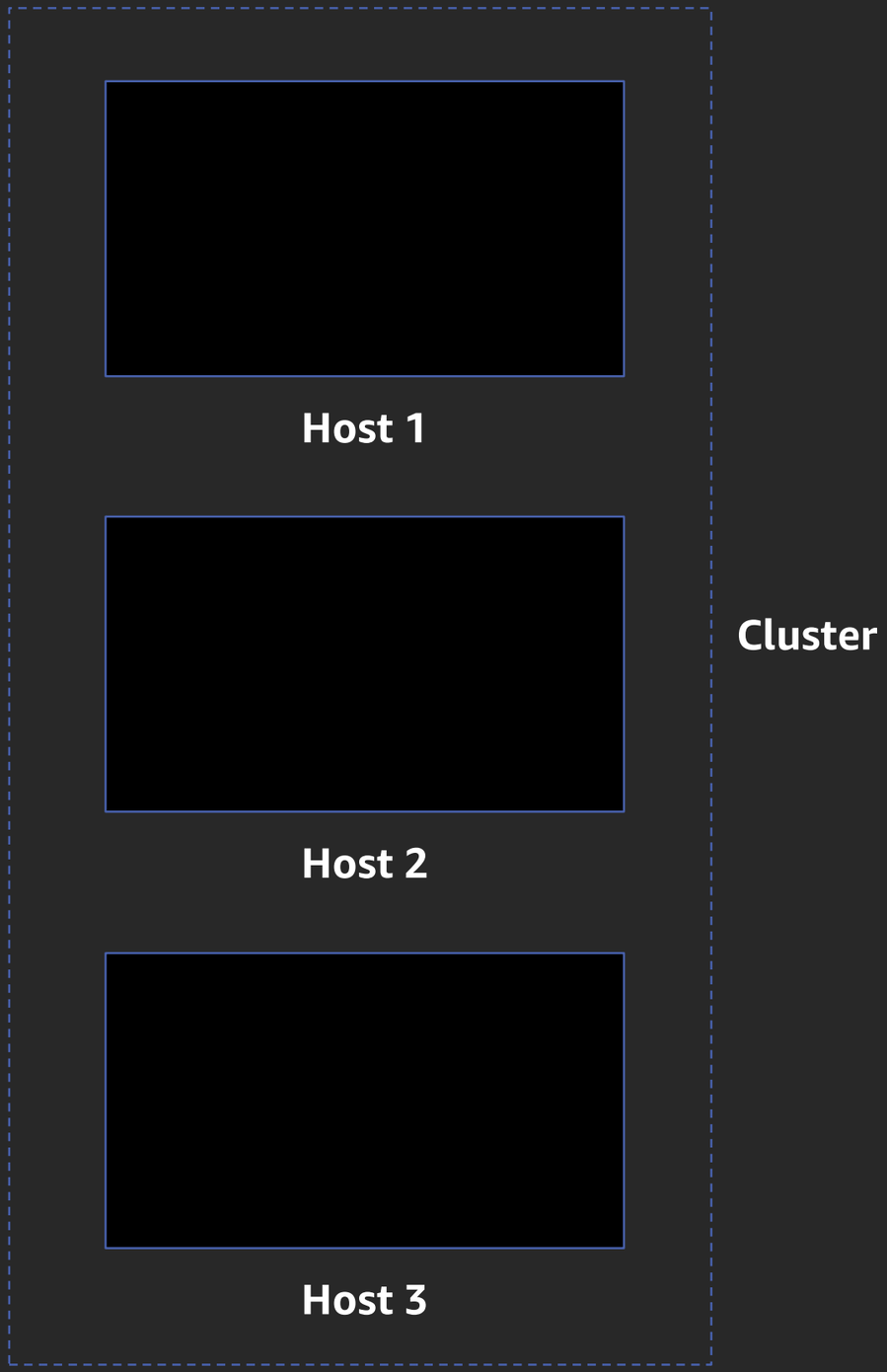
A host is a server, e.g., Amazon Elastic Compute Cloud (Amazon EC2) virtual machine. We run these hosts together as a cluster.

Our simple example web application is already containerized

3x

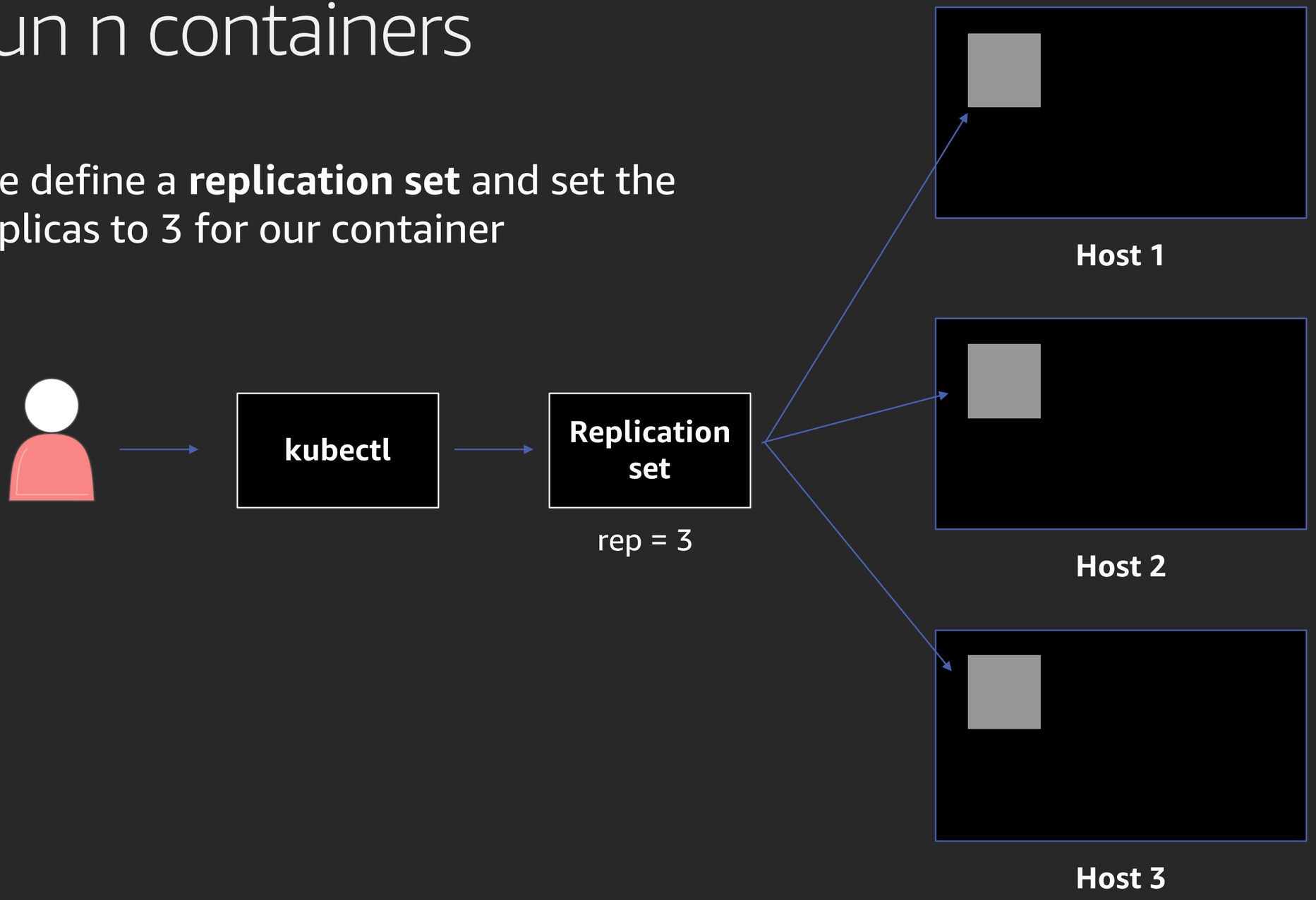
**Web application**

To start let’s run 3 copies of our web application across our cluster of Amazon EC2 hosts



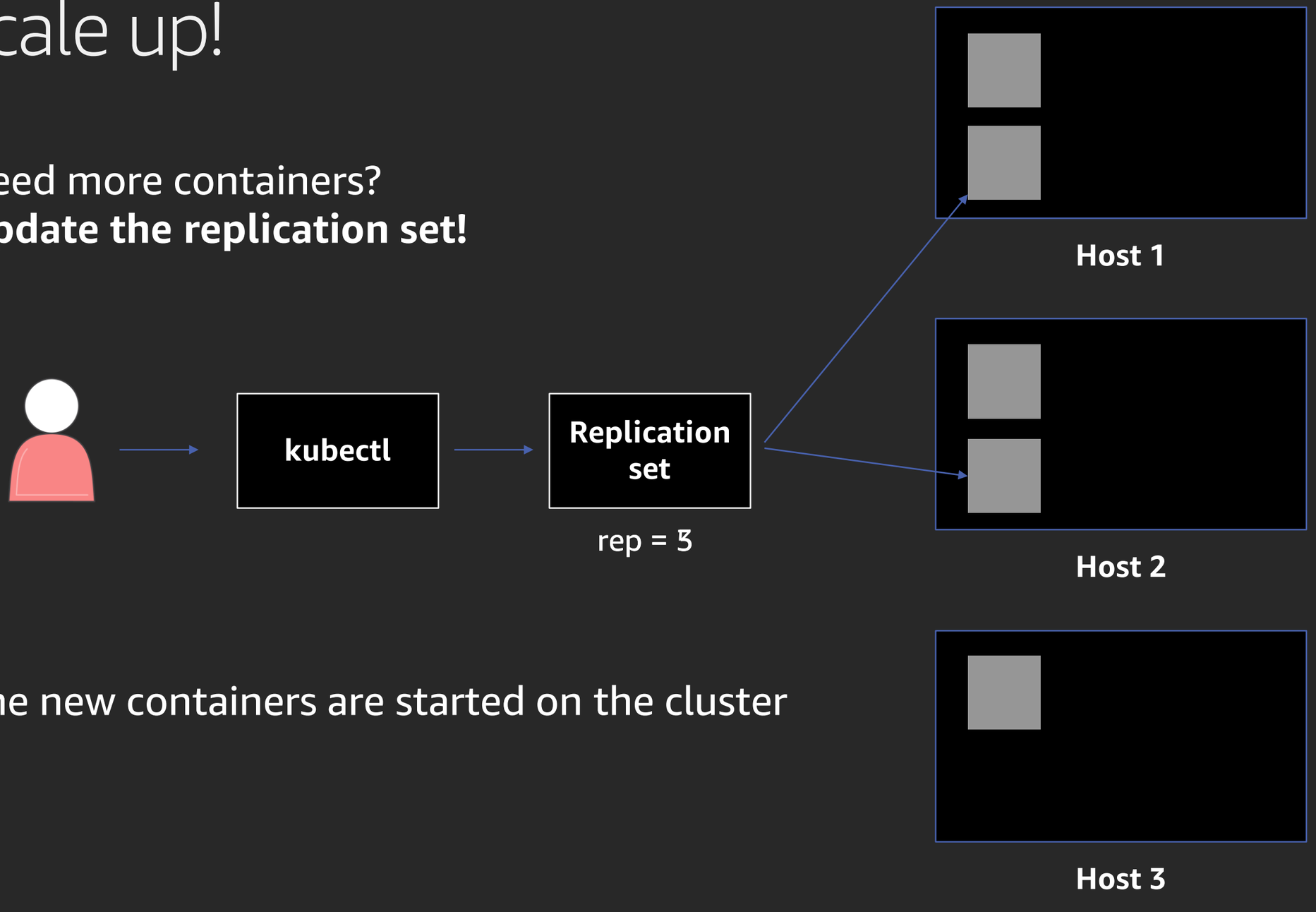
# Run n containers

We define a **replication set** and set the replicas to 3 for our container



# Scale up!

Need more containers?
**Update the replication set!**




The new containers are started on the cluster

# Untimely termination

Oh no! Our host has died!


Kubernetes notices only 3 of the 5 containers are running and starts 2 additional containers on the remaining hosts

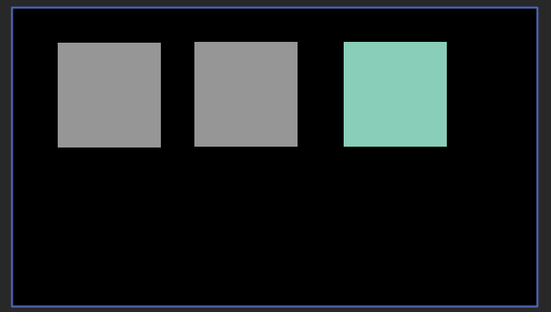
# Containers IRL

In production, we want to do more complex things like:

- Run a **service** to route traffic to a set of running containers
- Manage the **deployment** of containers to our cluster
- Run multiple containers **together** and specify how they run

**Host 1**

**Host 2**



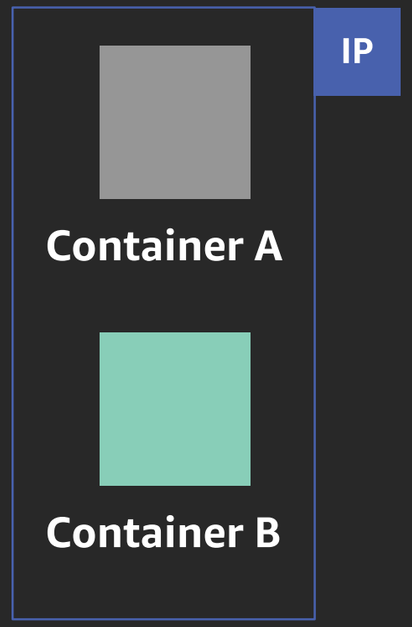
**Host 3**

# Pods

- Define how your containers should run
- Allow you to run 1 to n containers together

**Containers in pods have**
- Shared IP space
- Shared volumes
- Shared scaling (you scale pods, not individual containers)

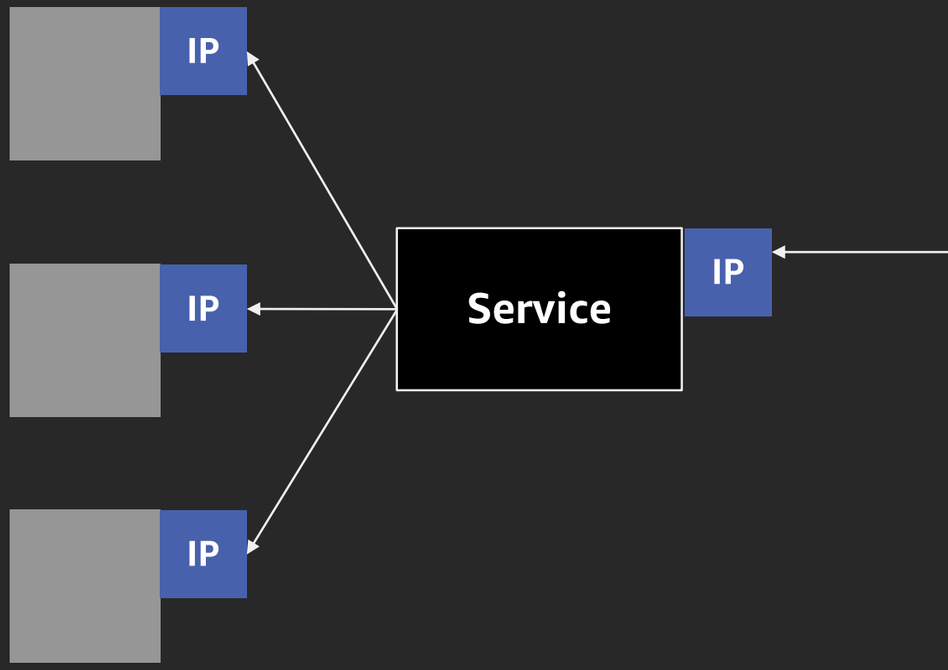
When containers are started on our cluster, they are always part of a pod
*(even if it's a pod of 1)*



# Services

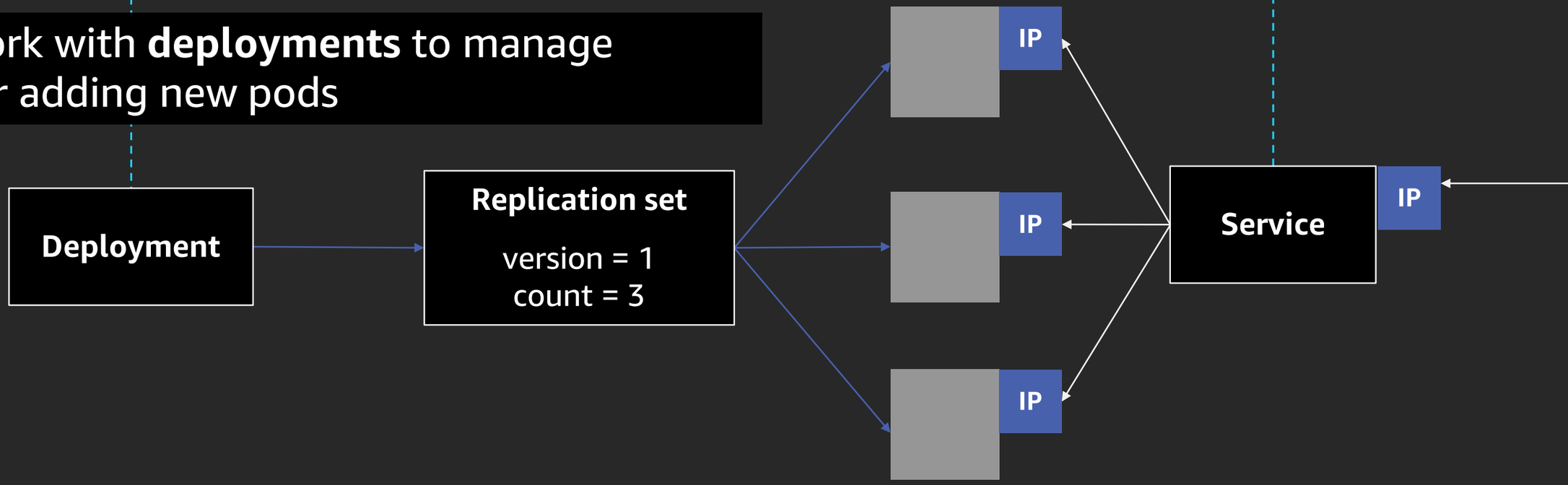
One of the ways traffic gets to your containers

- Internal IP addresses are assigned to each container
- Services are connected to containers and use labels to reference which containers to route requests to



# Deployments

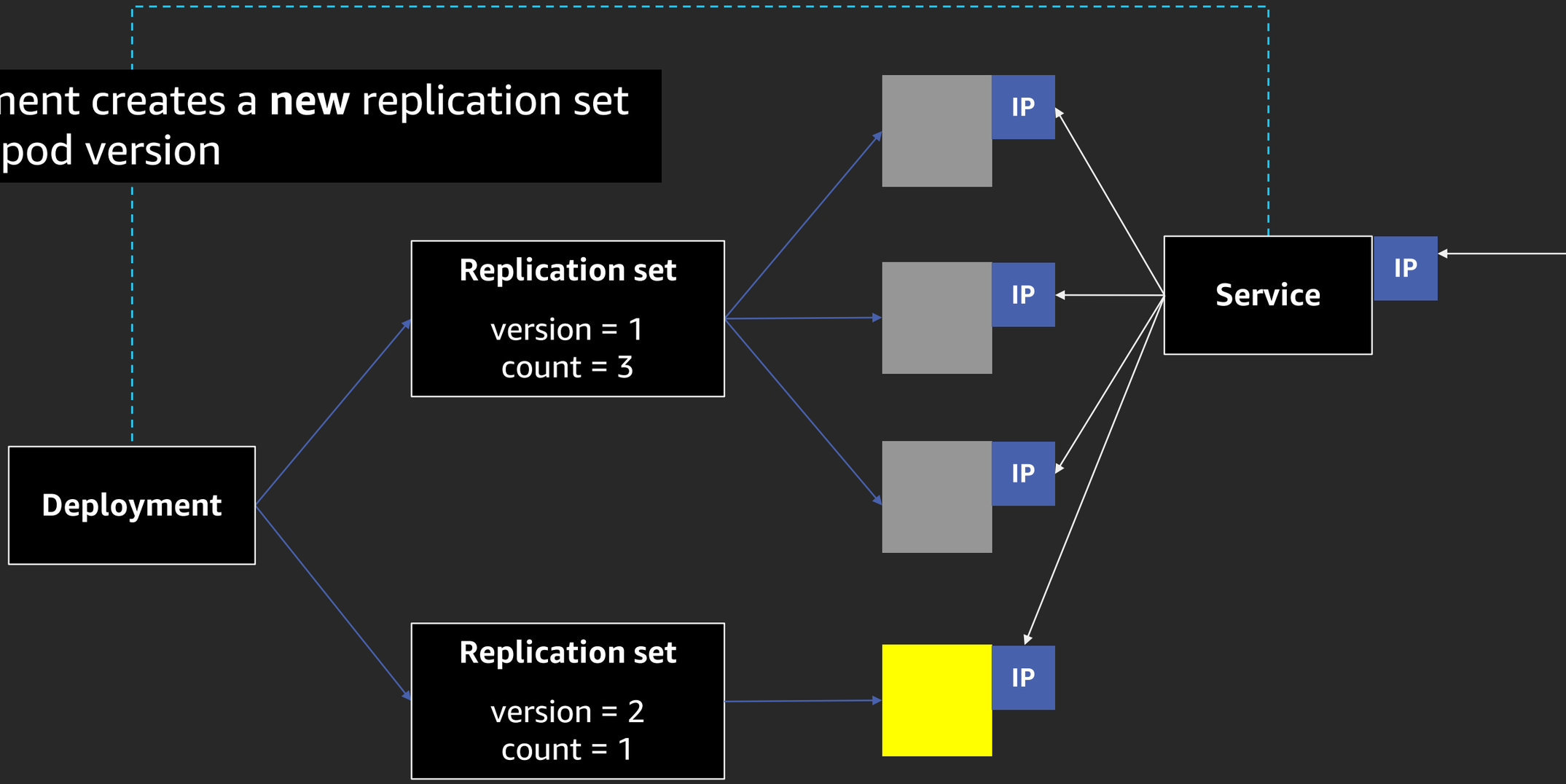
Services work with **deployments** to manage updating or adding new pods




Let’s say we want to deploy a new version of our web application as a “canary” and see how it handles traffic

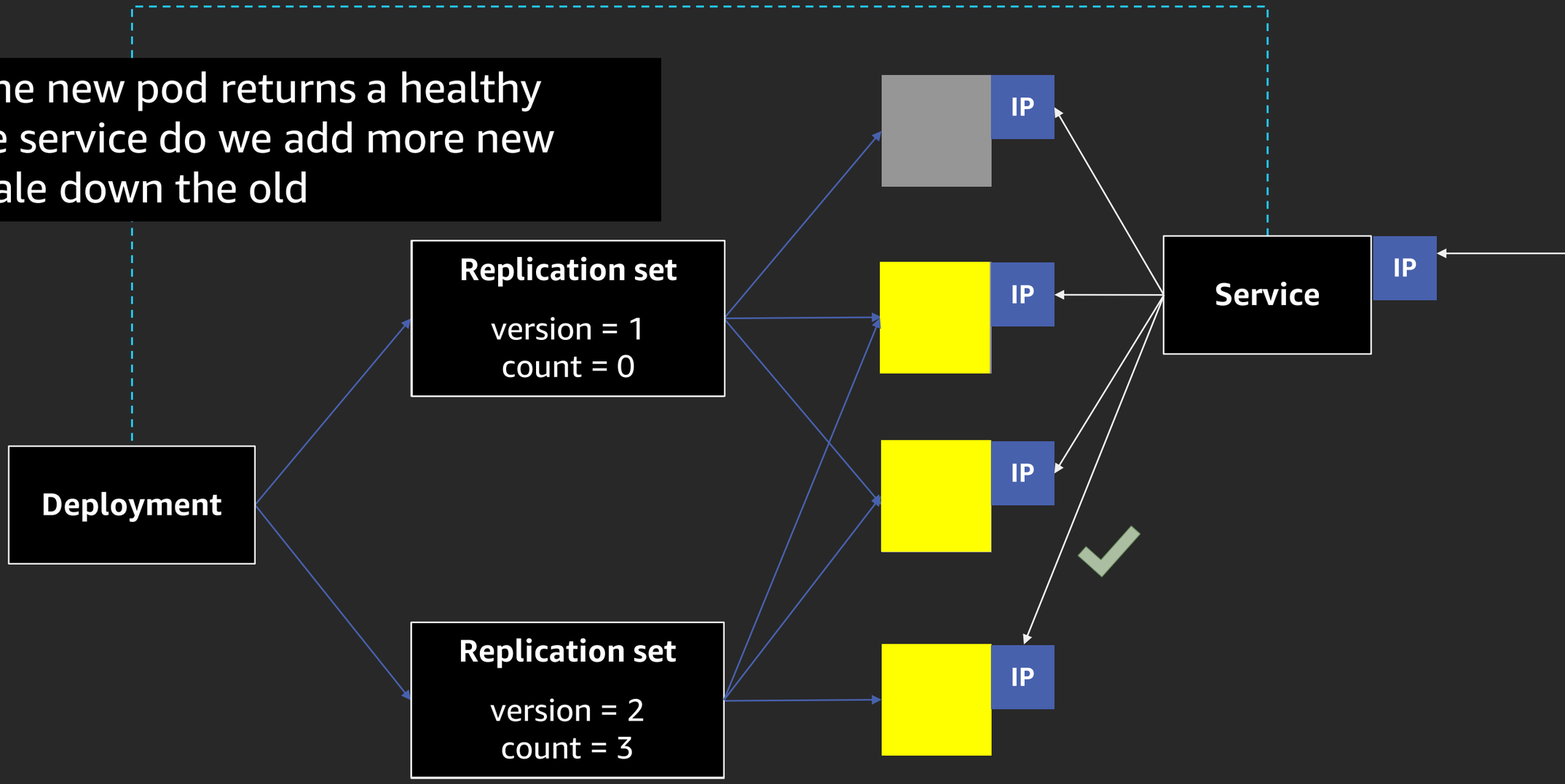
# Deployments

The deployment creates a **new** replication set for our new pod version



# Deployments

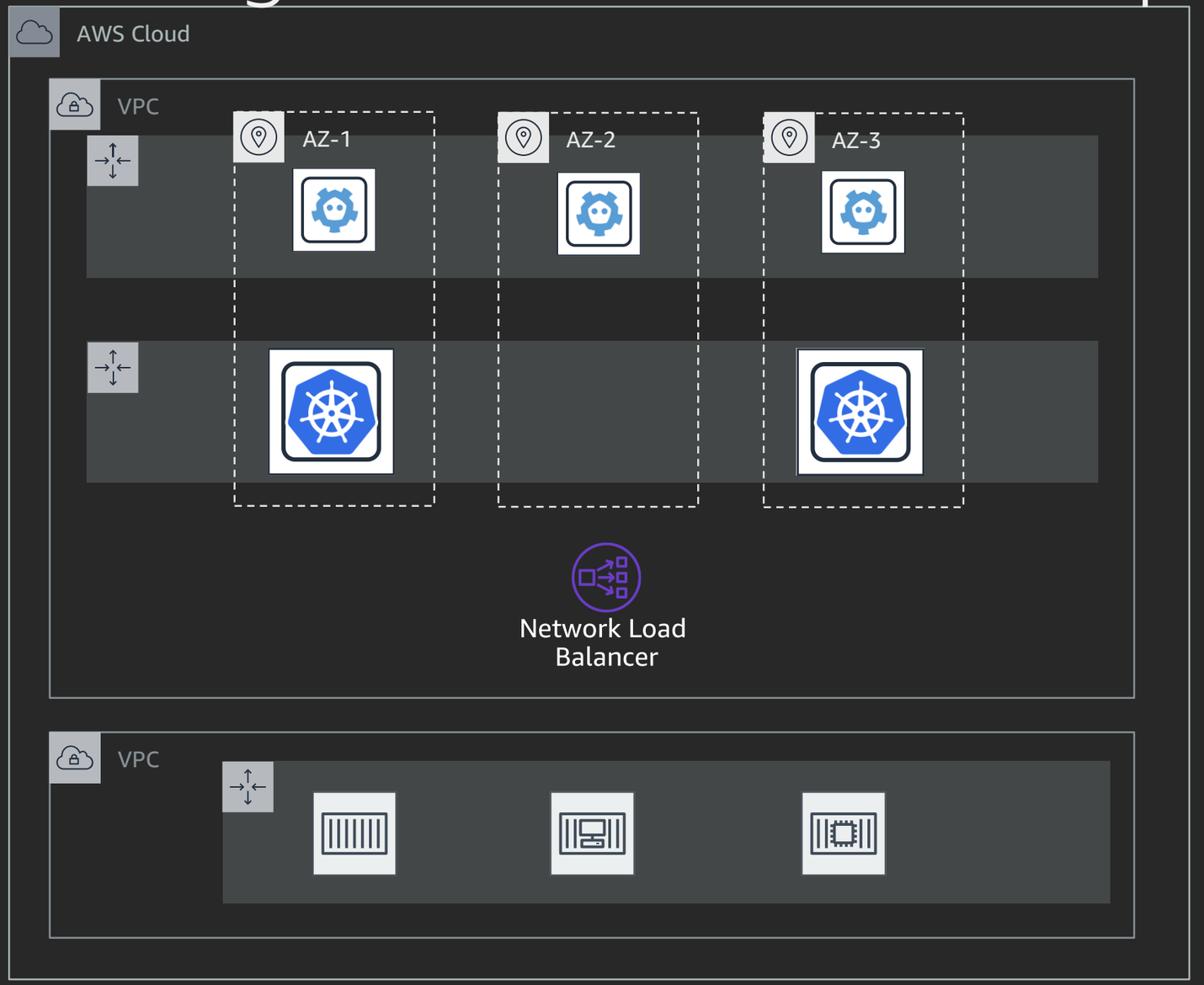
Only after the new pod returns a healthy status to the service do we add more new pods and scale down the old



# Amazon EKS: Managed Kubernetes control plane

Amazon Elastic Kubernetes Service (Amazon EKS) managed control plane

Customer owned data plane



# Amazon EKS

- **Amazon EKS** *Noun.* \ˈa-mə-ˌzänˌiːˌkeɪˌɛs\
- *Production-ready, highly-available, managed Kubernetes service for AWS*

1. Platform for enterprises to run production-grade workloads
2. Provides a native and upstream Kubernetes experience
3. AWS integrations are seamless and eliminate undifferentiated heavy lifting
4. EKS team actively contributes to the Kubernetes community

# AWS: The best place to run Kubernetes

- AWS Managed Services option: Amazon EKS
  - Eksctl: Open source command line management using gitops
- Open source DIY options:
  - Kops
  - Kubeadm
  - Kube-Aws
  - Kubespray
- Partner supported options: MANY
- Mix and match!
  - Use Amazon Linux 2 Amazon EKS AMI with DIY
  - Use commercially supported OS with Amazon EKS
  - Use EKS CNI driver on DIY or partner or open source CNI with Amazon EKS

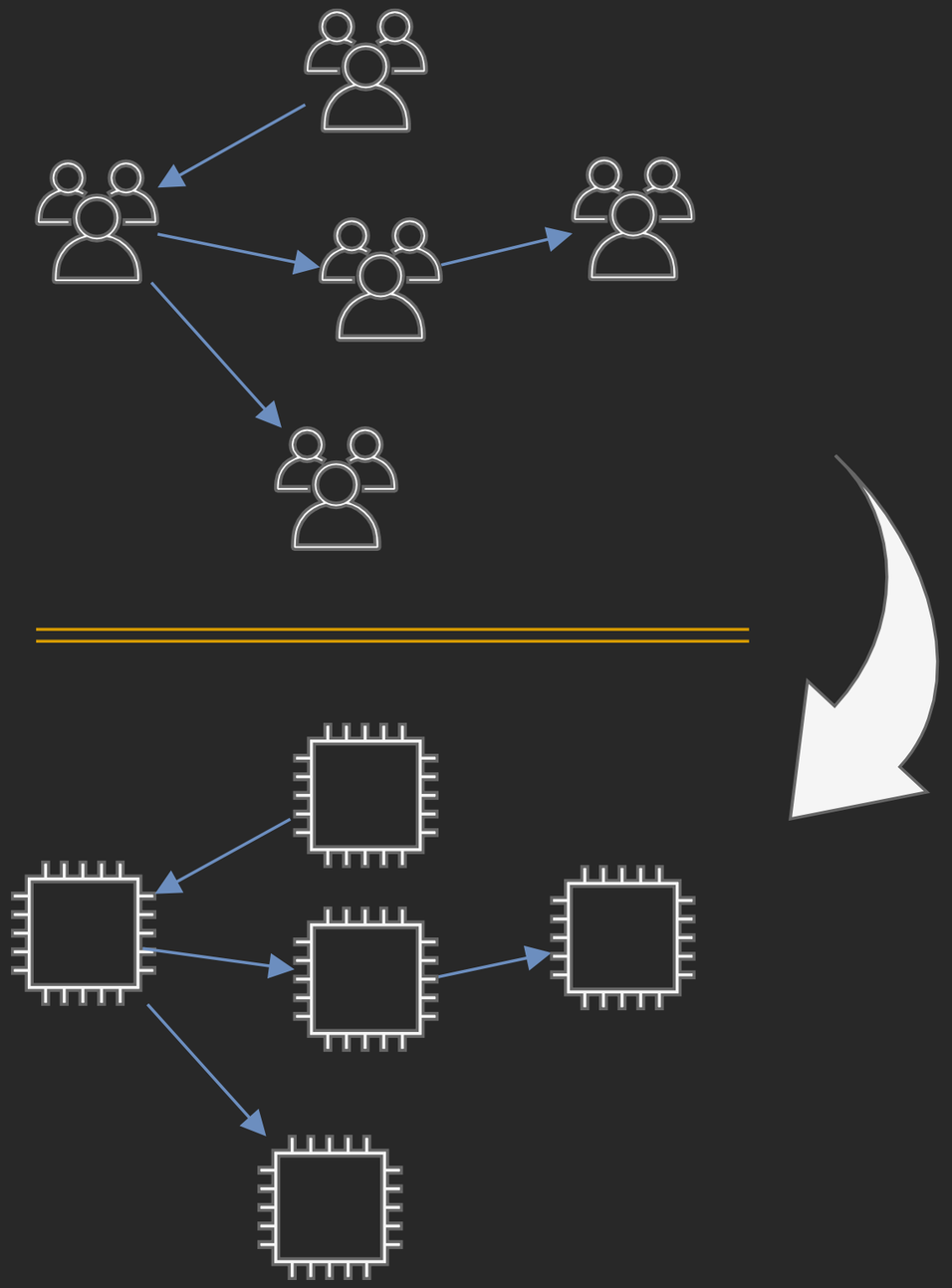
# From monoliths to microservices: The rendering problem

# From monoliths…

Two-pizza teams

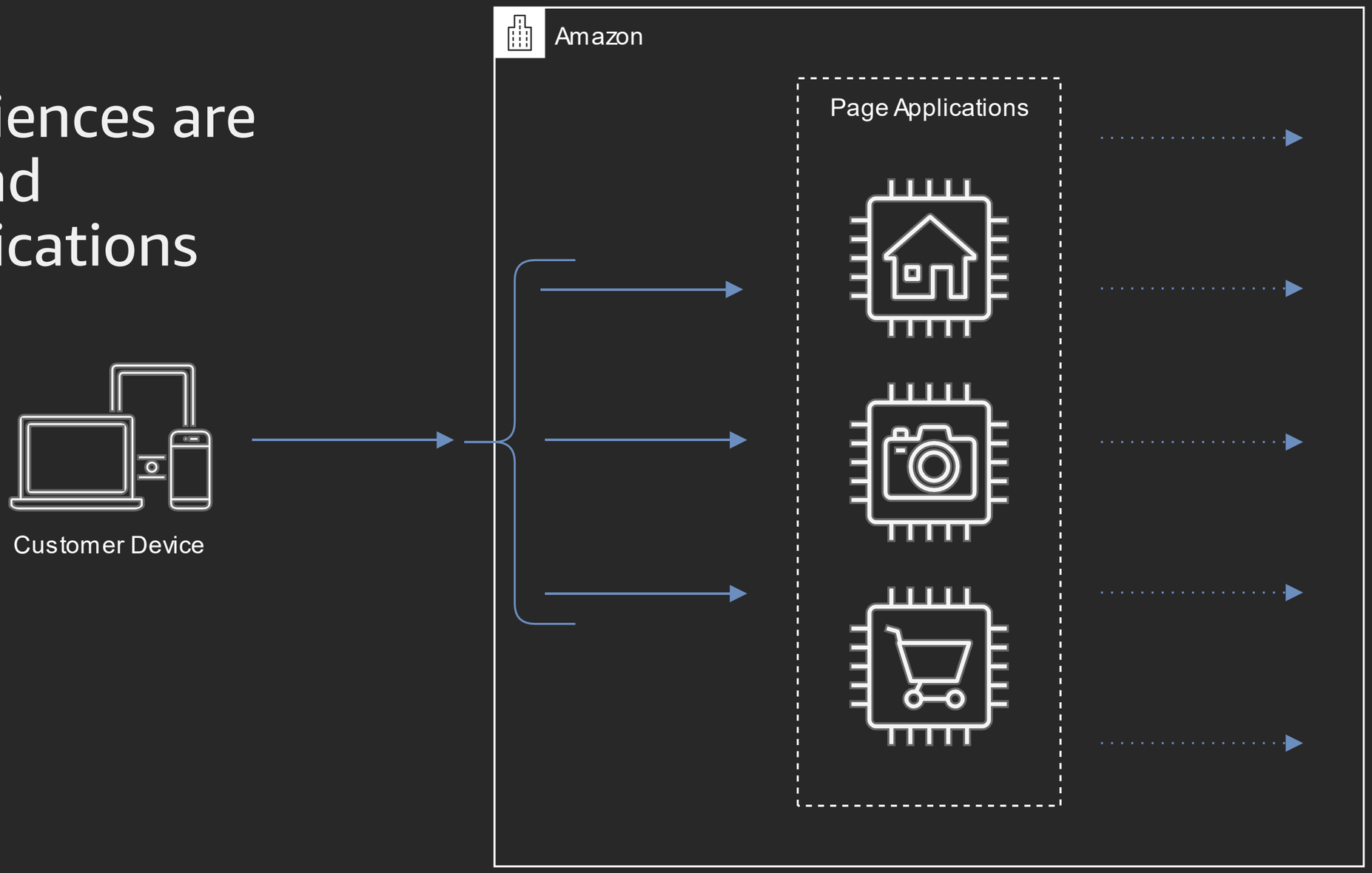
Service-oriented architecture

Traditionally data-oriented services



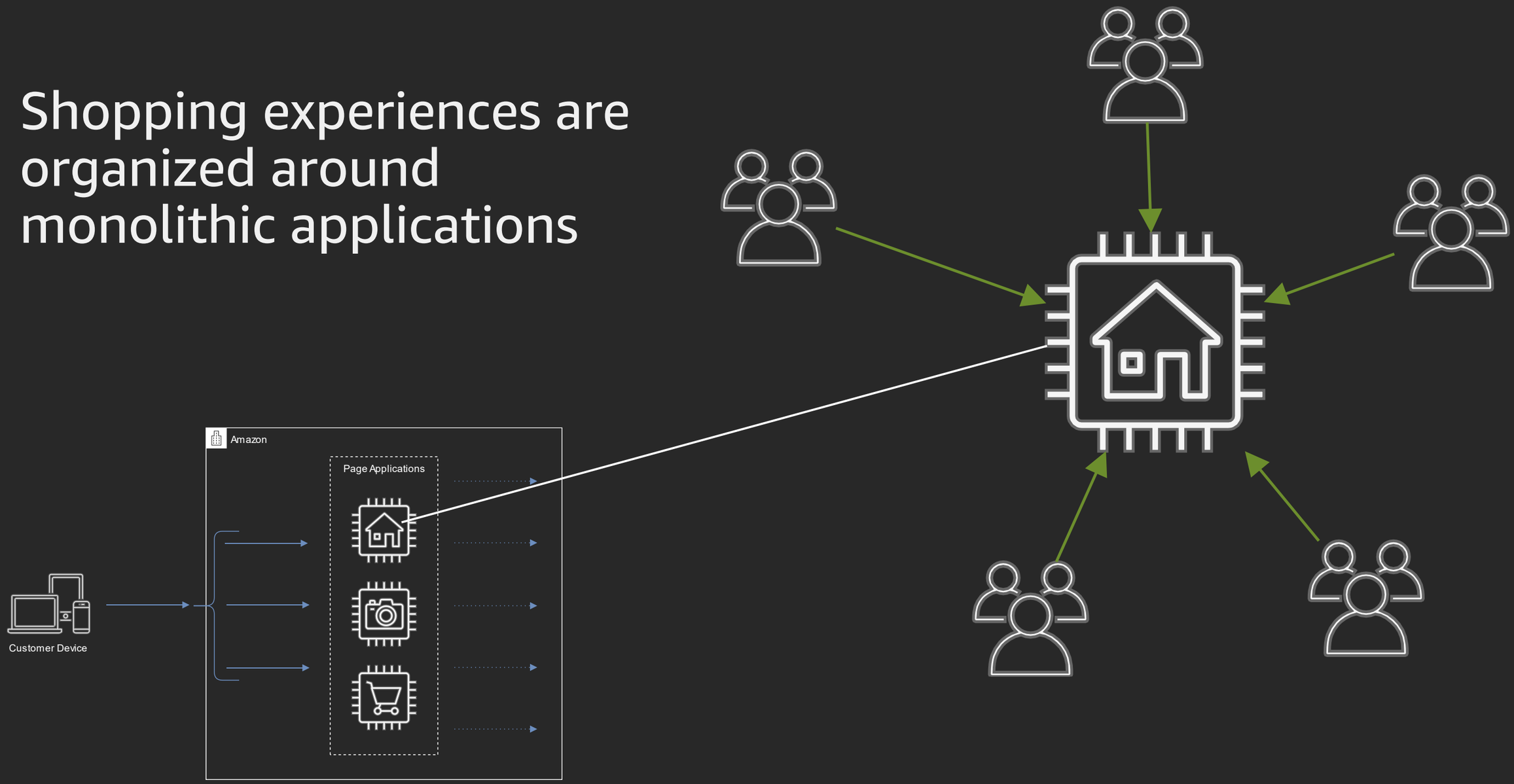
# From monoliths…

Shopping experiences are organized around monolithic applications



# From monoliths…

Shopping experiences are organized around monolithic applications




# Break it apart

# …To microservices

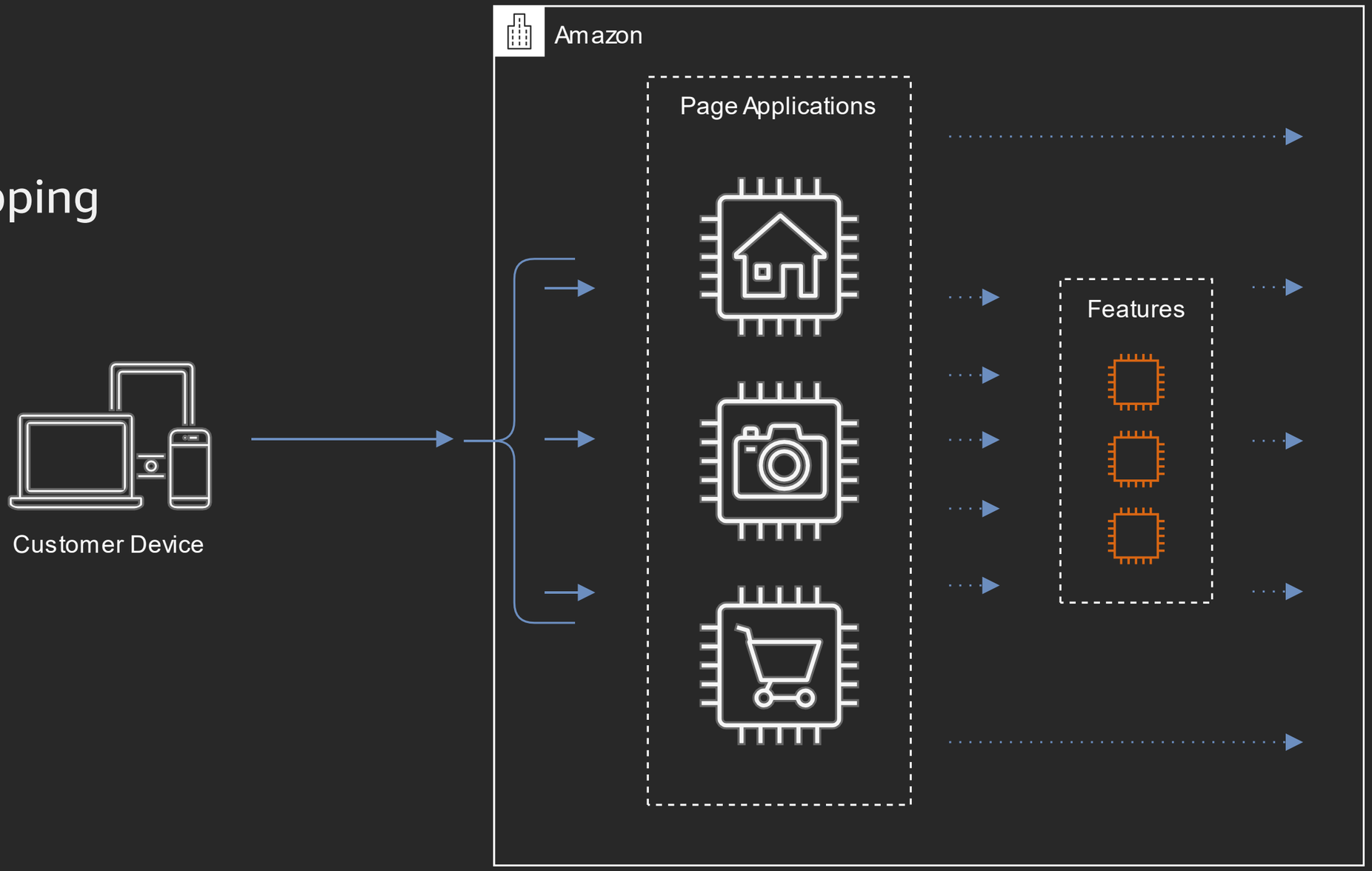
Independent release cycle

Rapid experimentation

Isolation across features

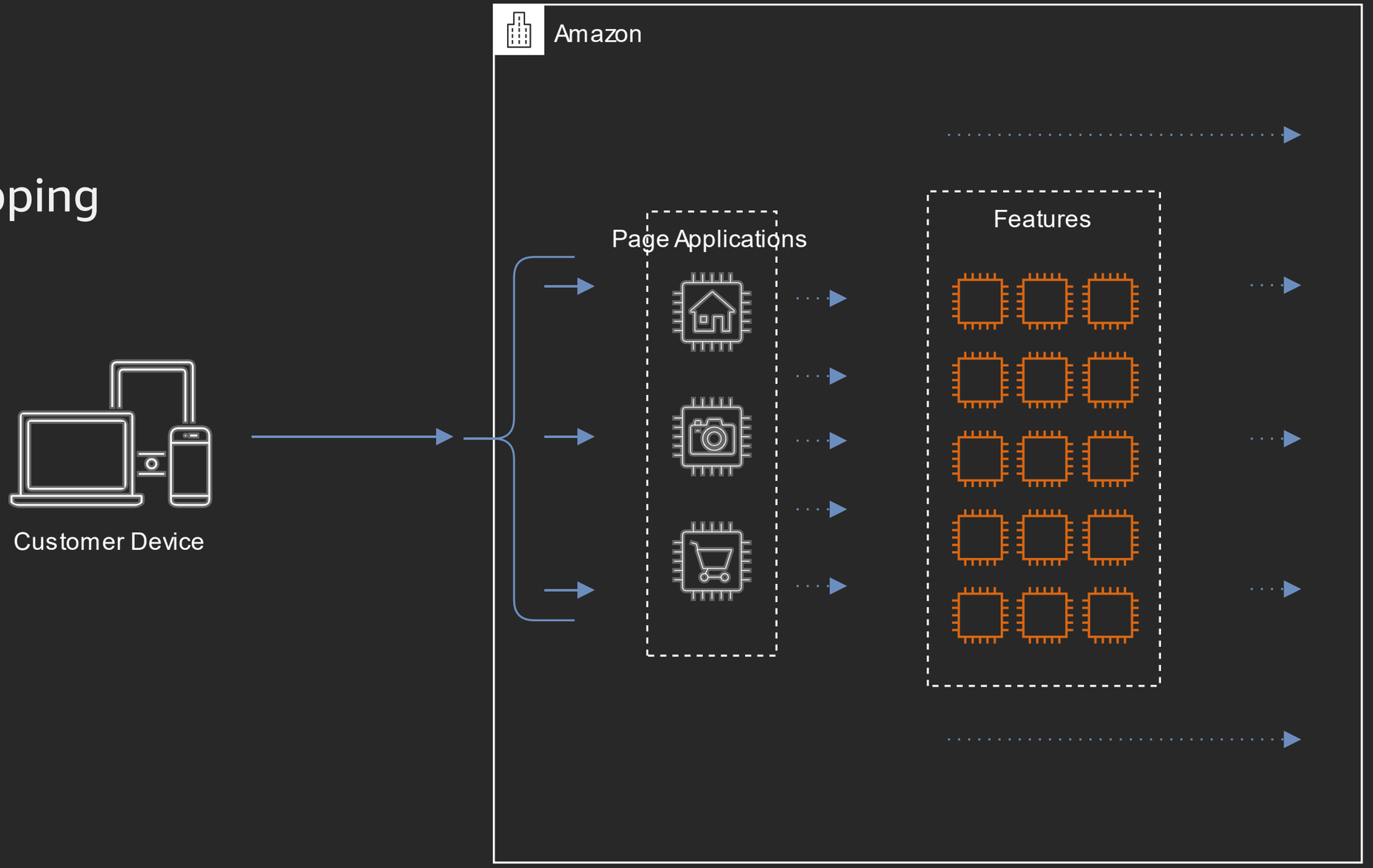
# …To microservices

Decompose the shopping experience



# …To microservices

Decompose the shopping experience



amazon Try Prime

All

EN

Hello, Sign in
Account & Lists

Orders

Try Prime

0 Cart

Hello
Select your address

Best Sellers | Gift Ideas | New Releases | Whole Foods | Today's Deals

Shop Halloween décor

# Deals, movies and music
all in one membership
prime

## Deal of the Day


## Women's shop by style


Cool


Casual


Classic


Boho


Bissell ICONpet Cordless Stick Vacuum

4.0 out of 5 stars 27

$299.99 prime

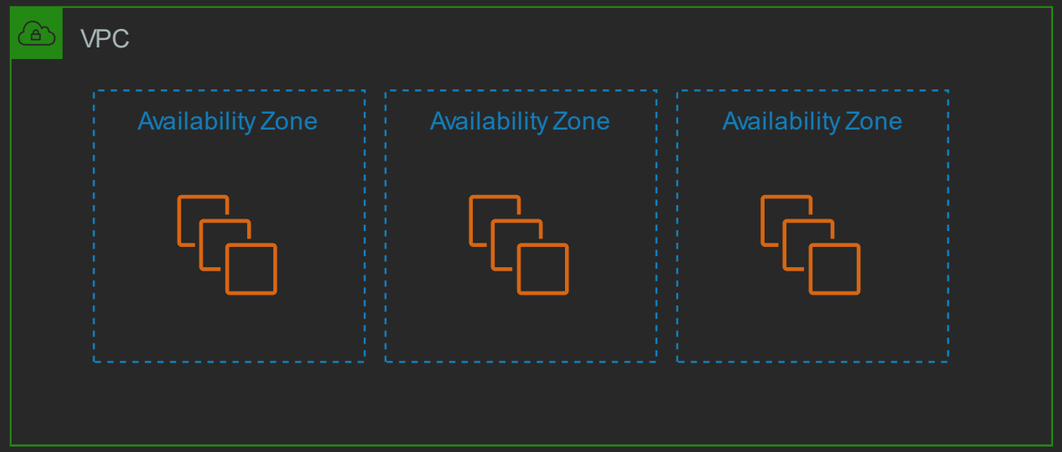
Add to Cart

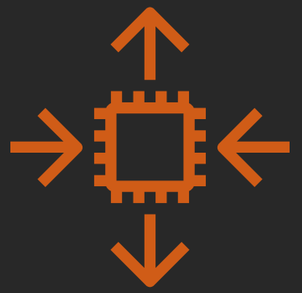
Ad feedback

# …To microservices

Hosted environment requirements

- Availability
- Scalability
- Reliability

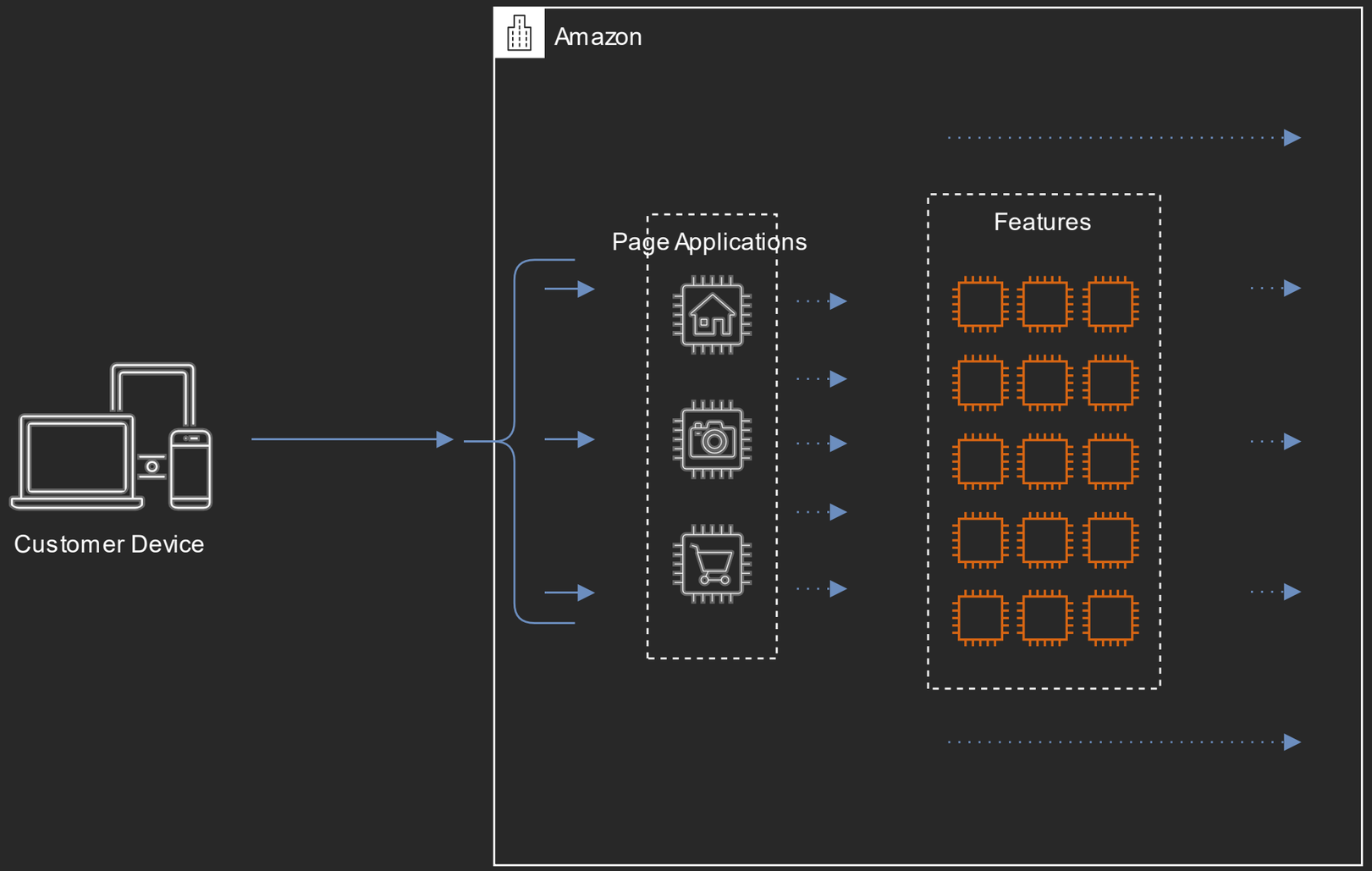


# …To microservices

Docker containers

Kubernetes clusters

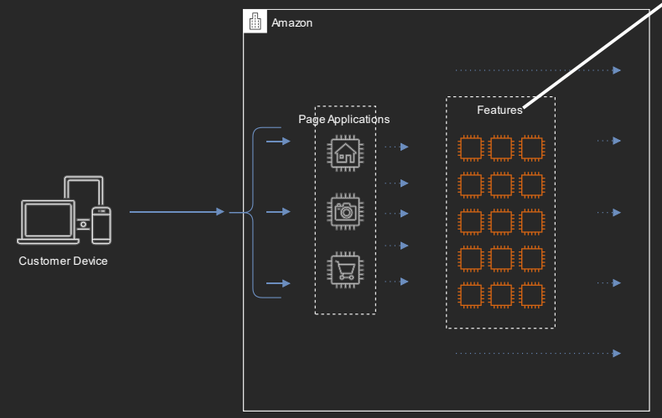


# Improving ownership by owning less

# Improving ownership by owning less

## How did Amazon EKS help?

- Managed updates
- etcd management
- Masters scaling




# Improving ownership by owning less

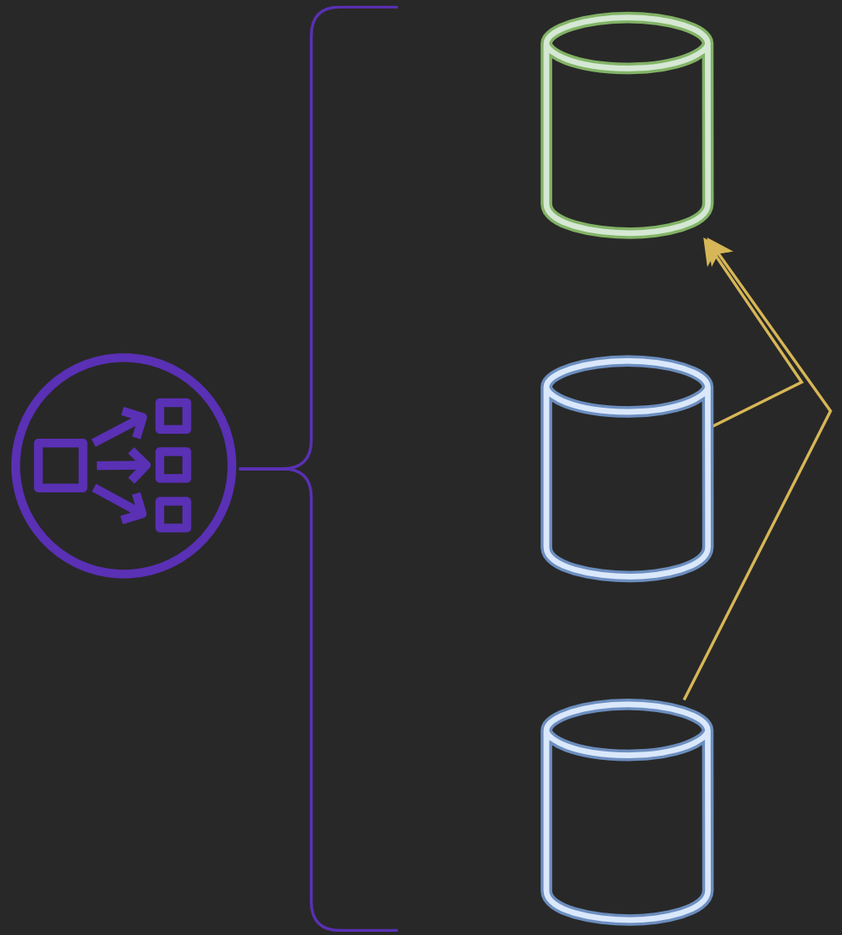
## Managed updates

Common vulnerabilities and exposures (CVE)

Supported version updates

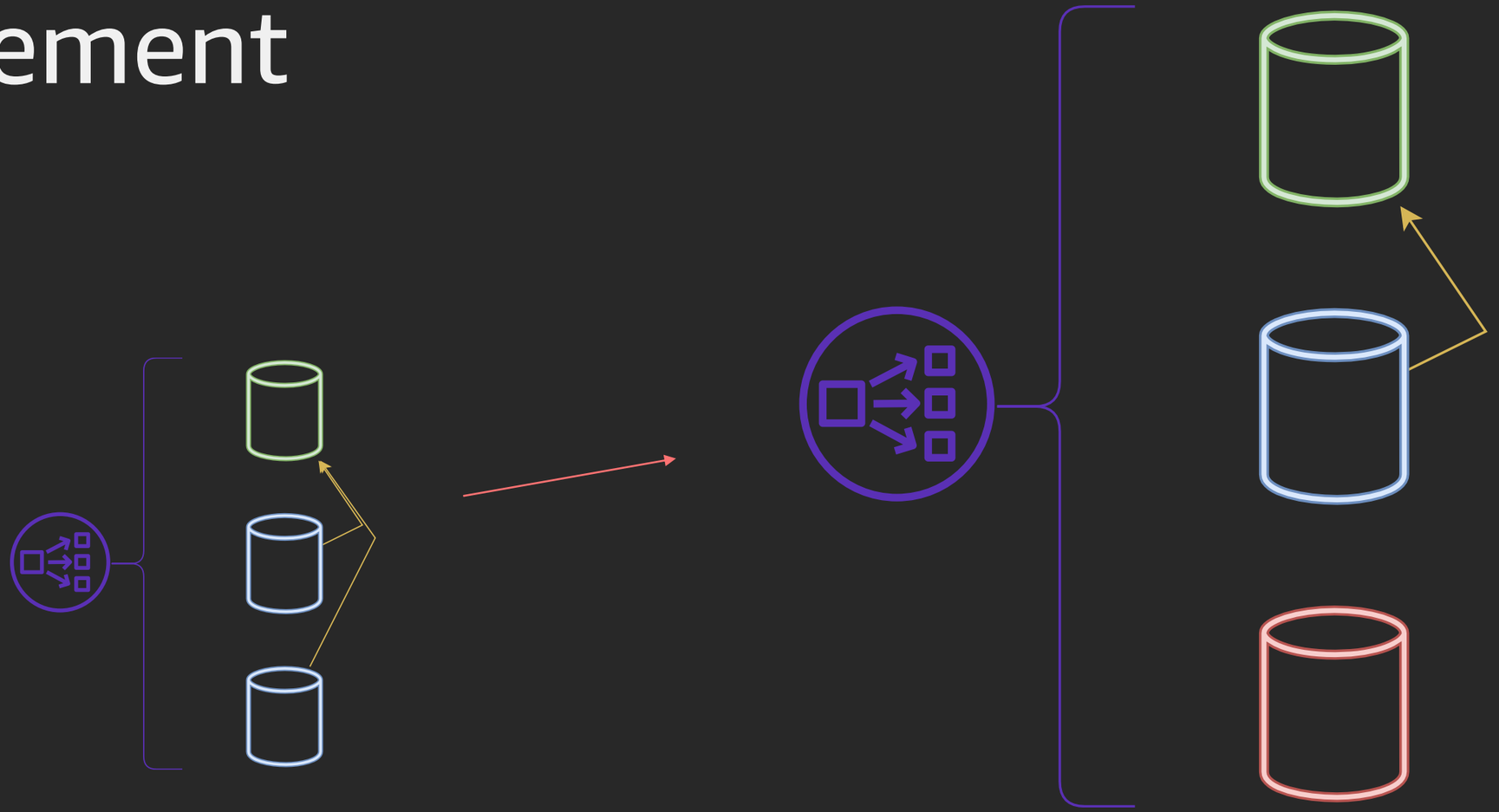
# Improving ownership by owning less

## etcd management



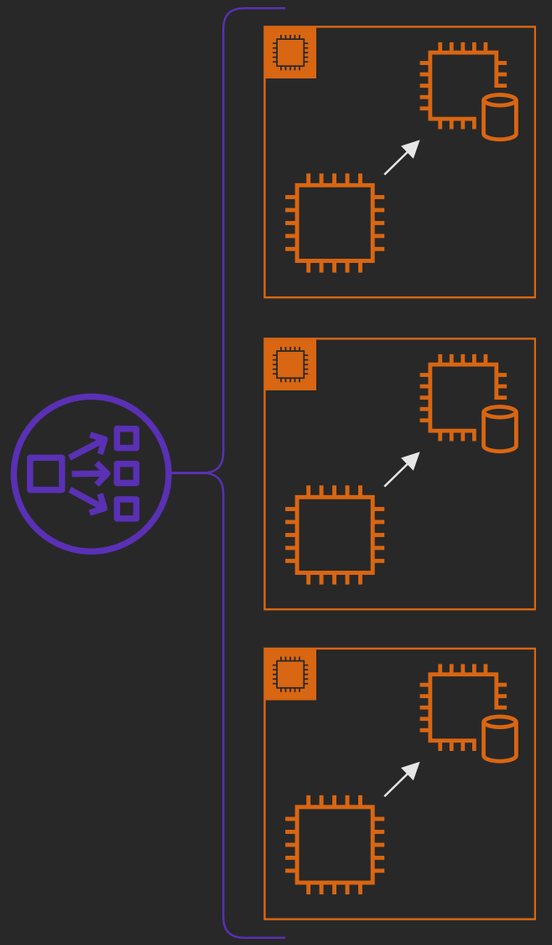
# Improving ownership by owning less

## etcd management



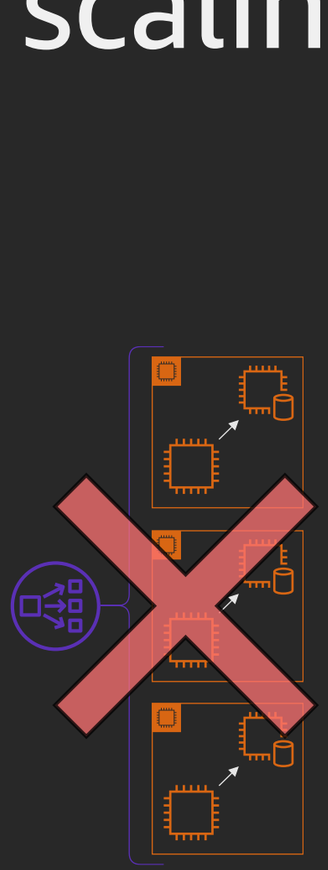
# Improving ownership by owning less

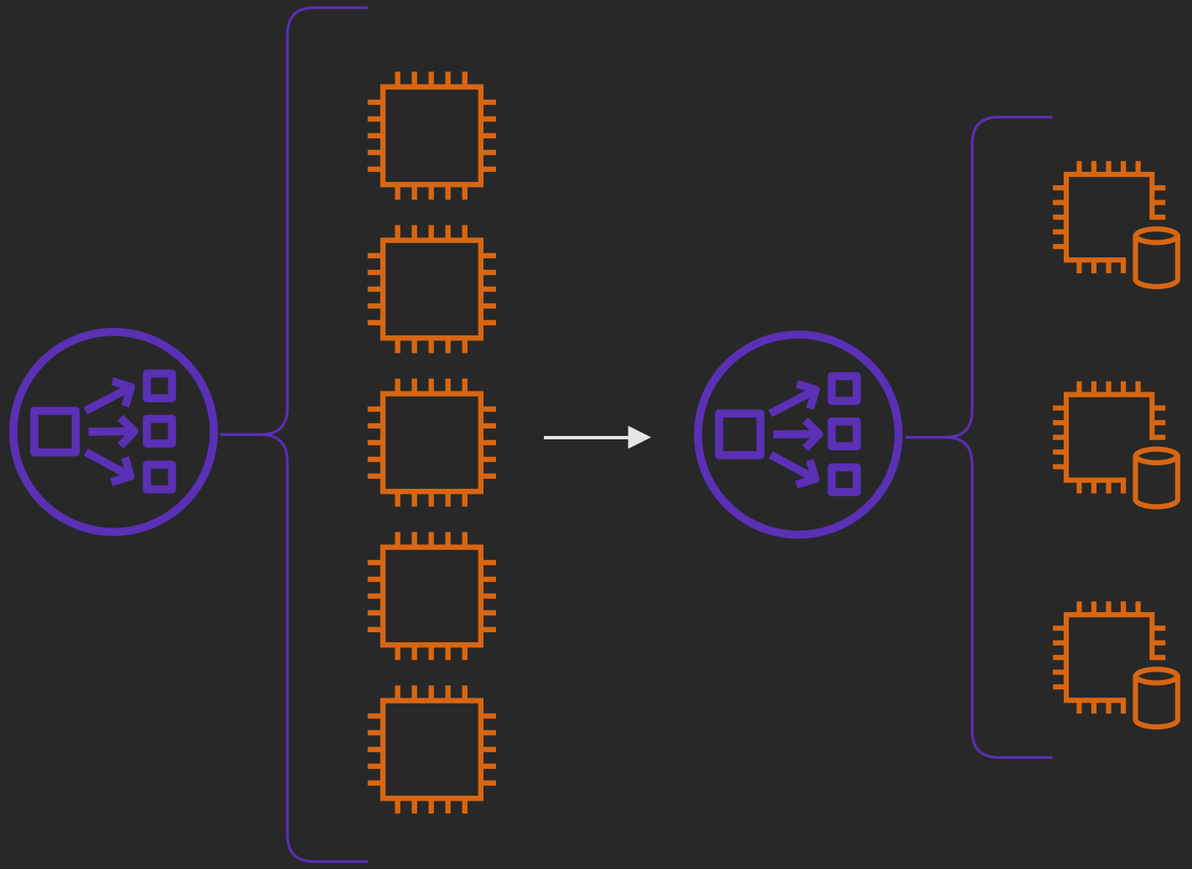
## Masters scaling



# Improving ownership by owning less

## Masters scaling

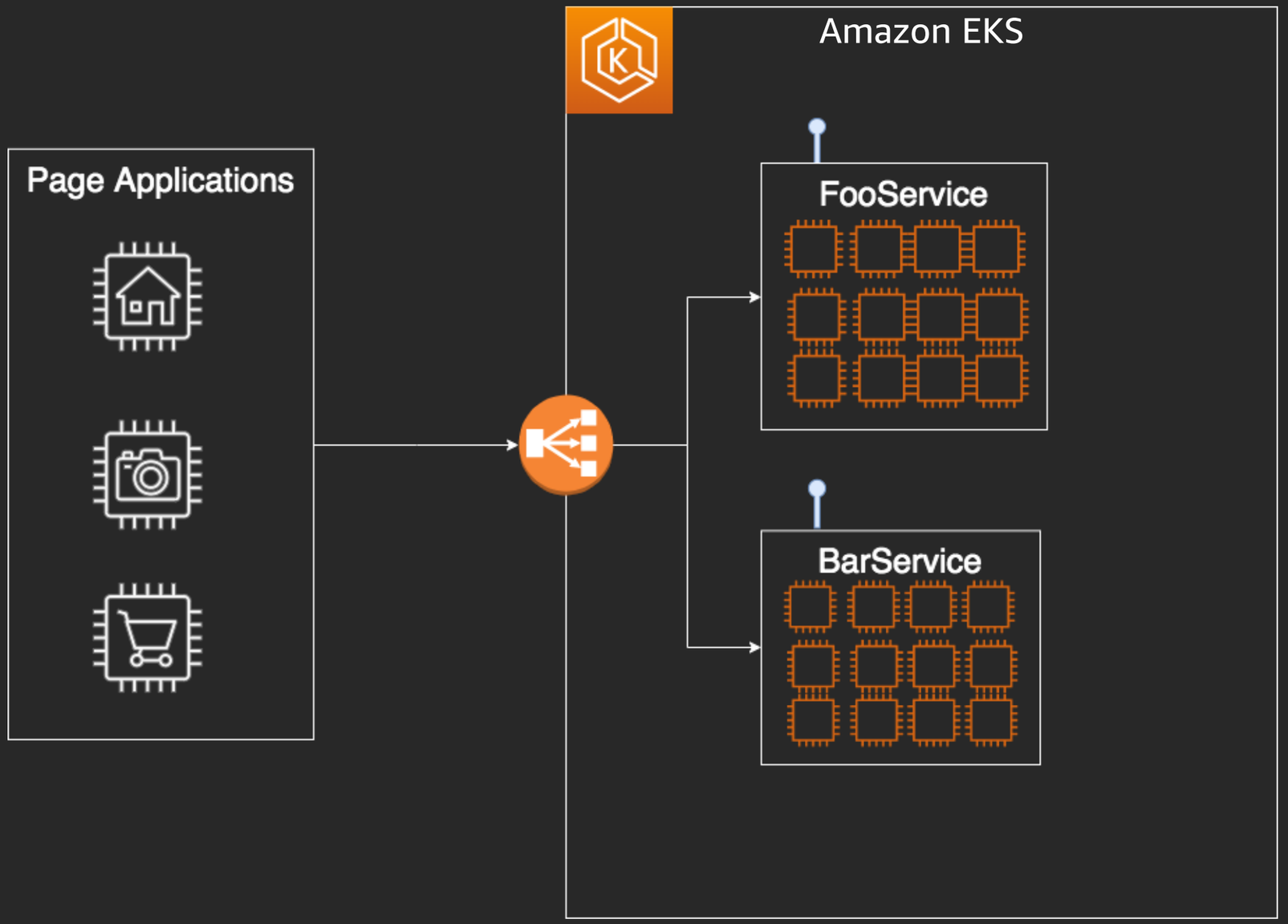




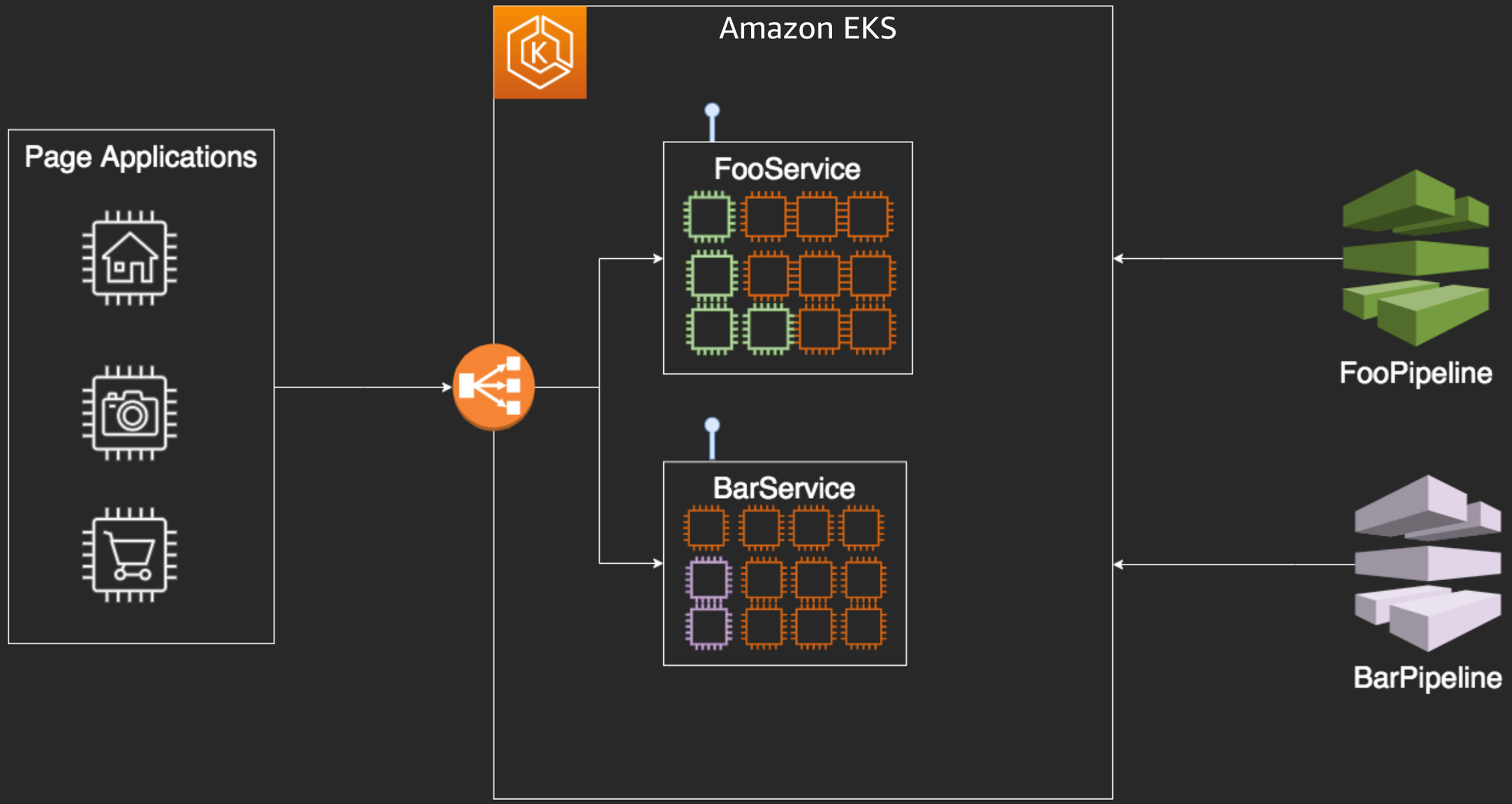

Focus on the problem

# CI/CD: Release features all the time!

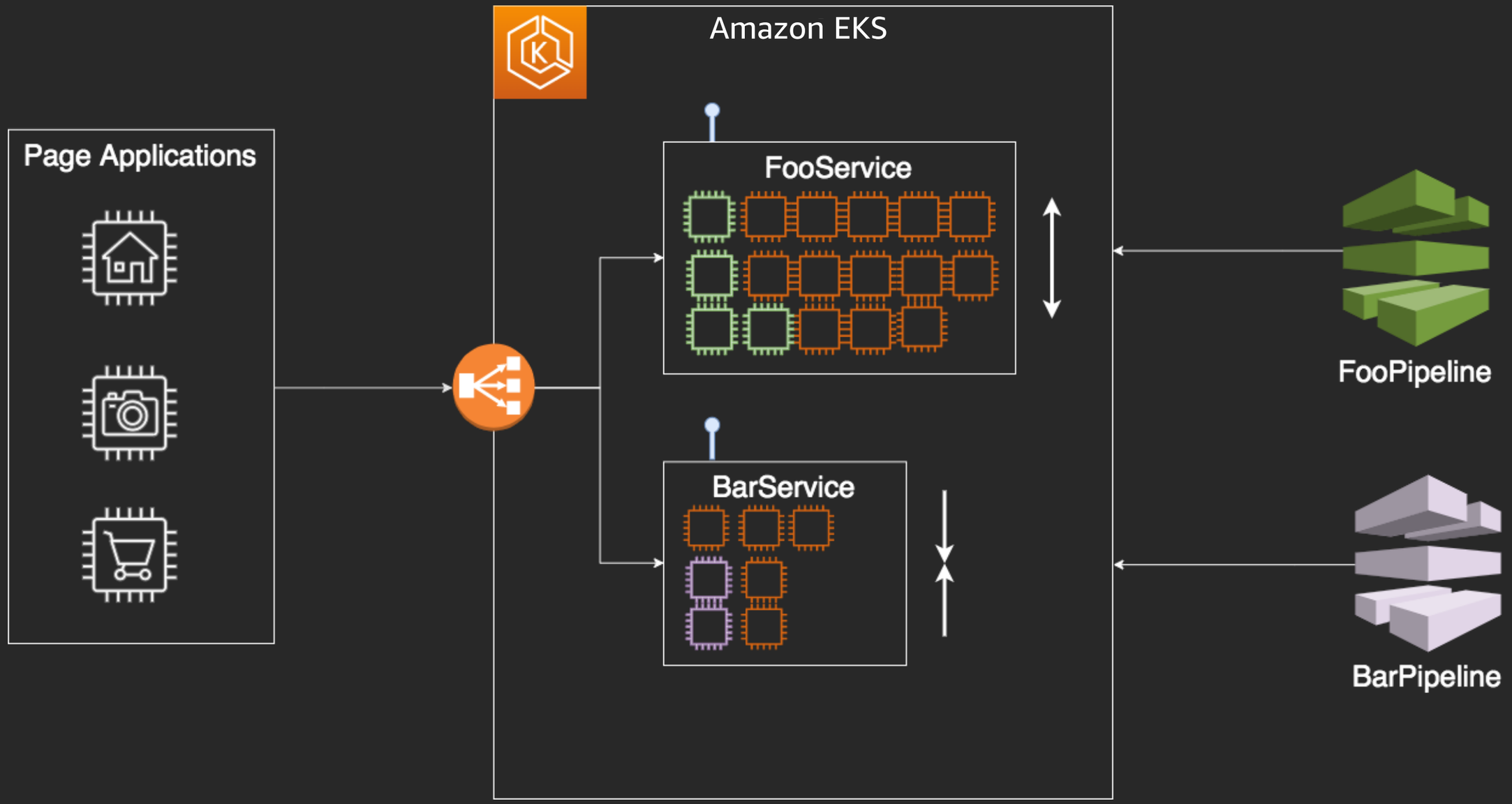
# Containerized applications



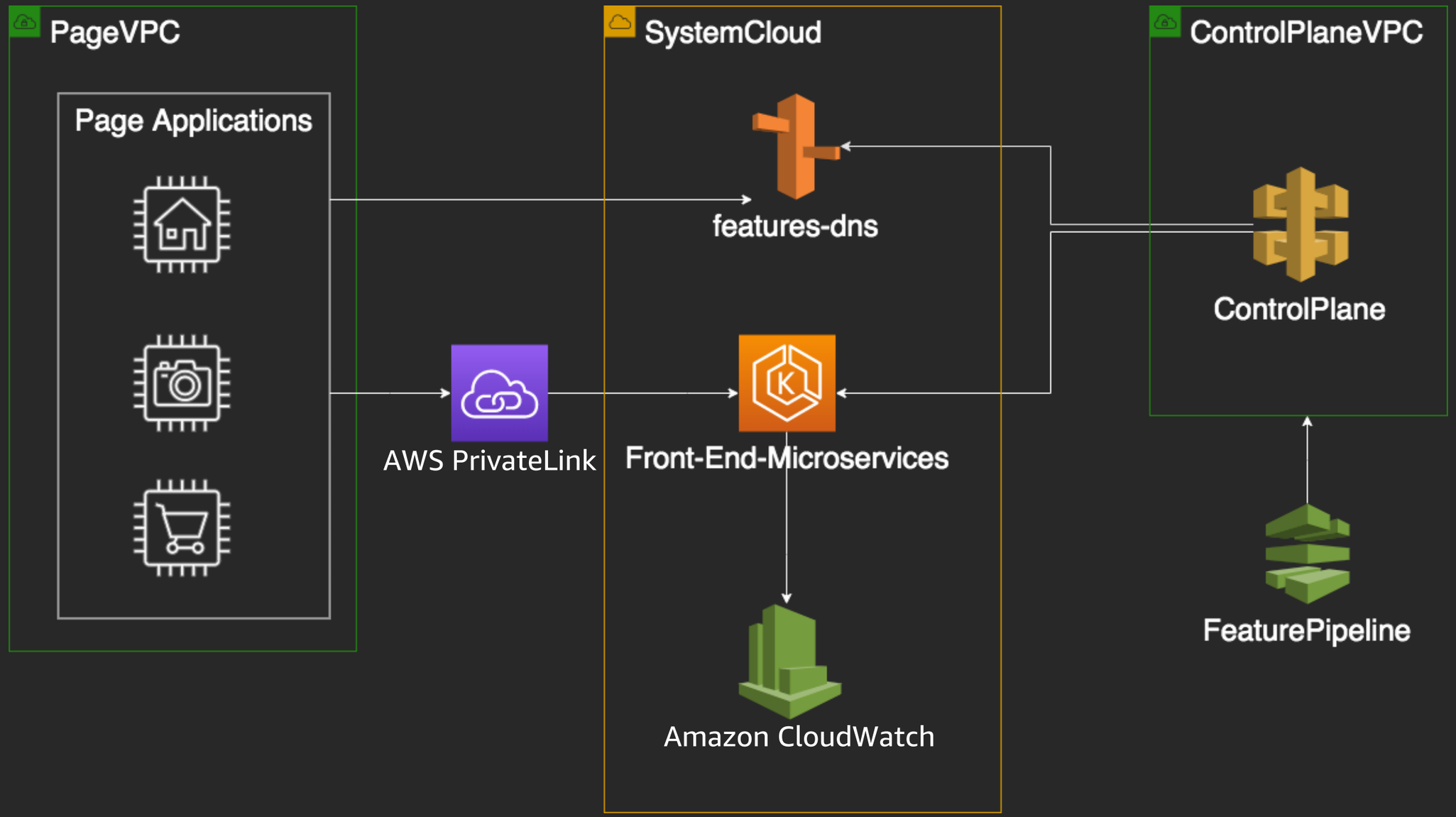
# Gradual deployments



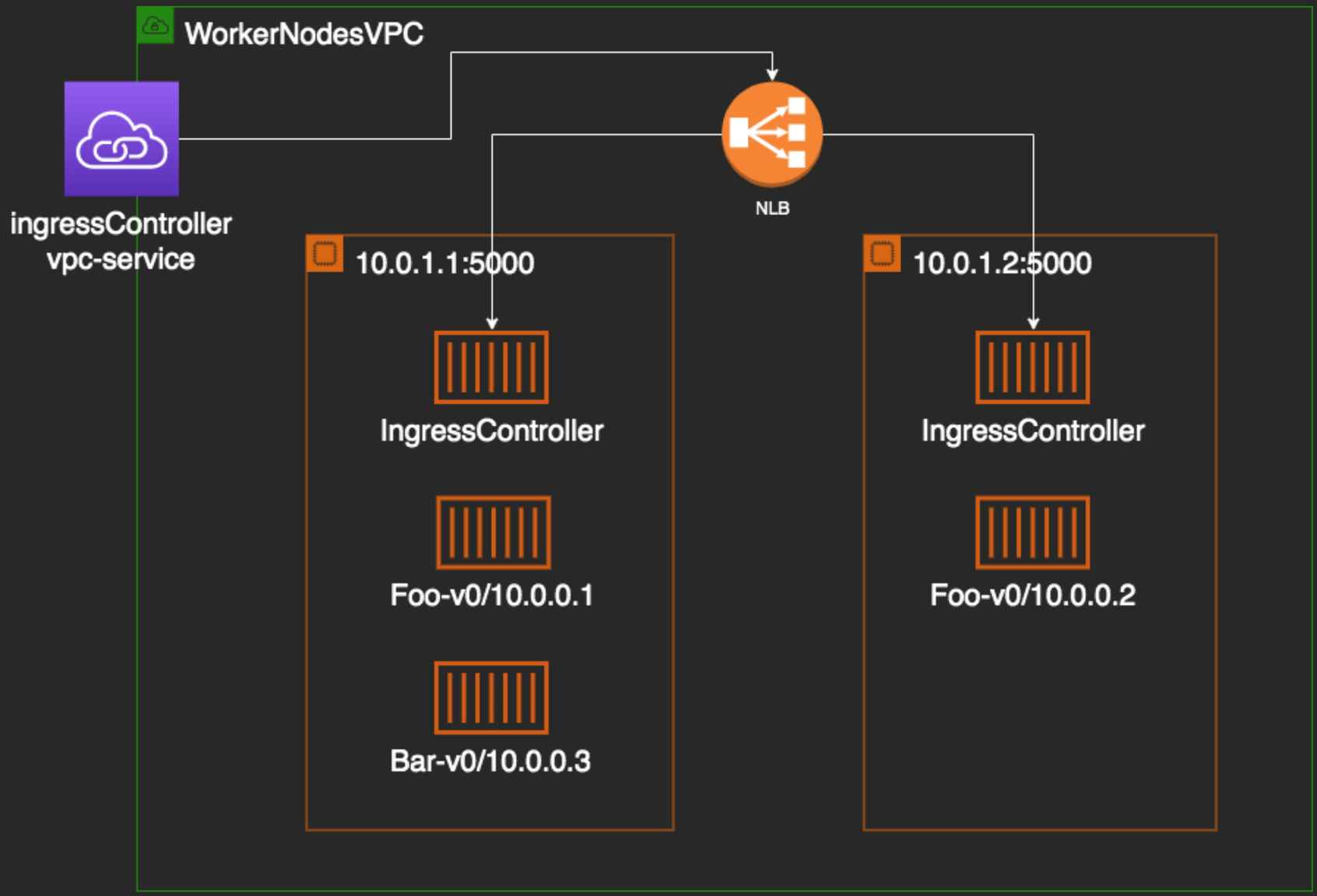
# Highly available elasticity



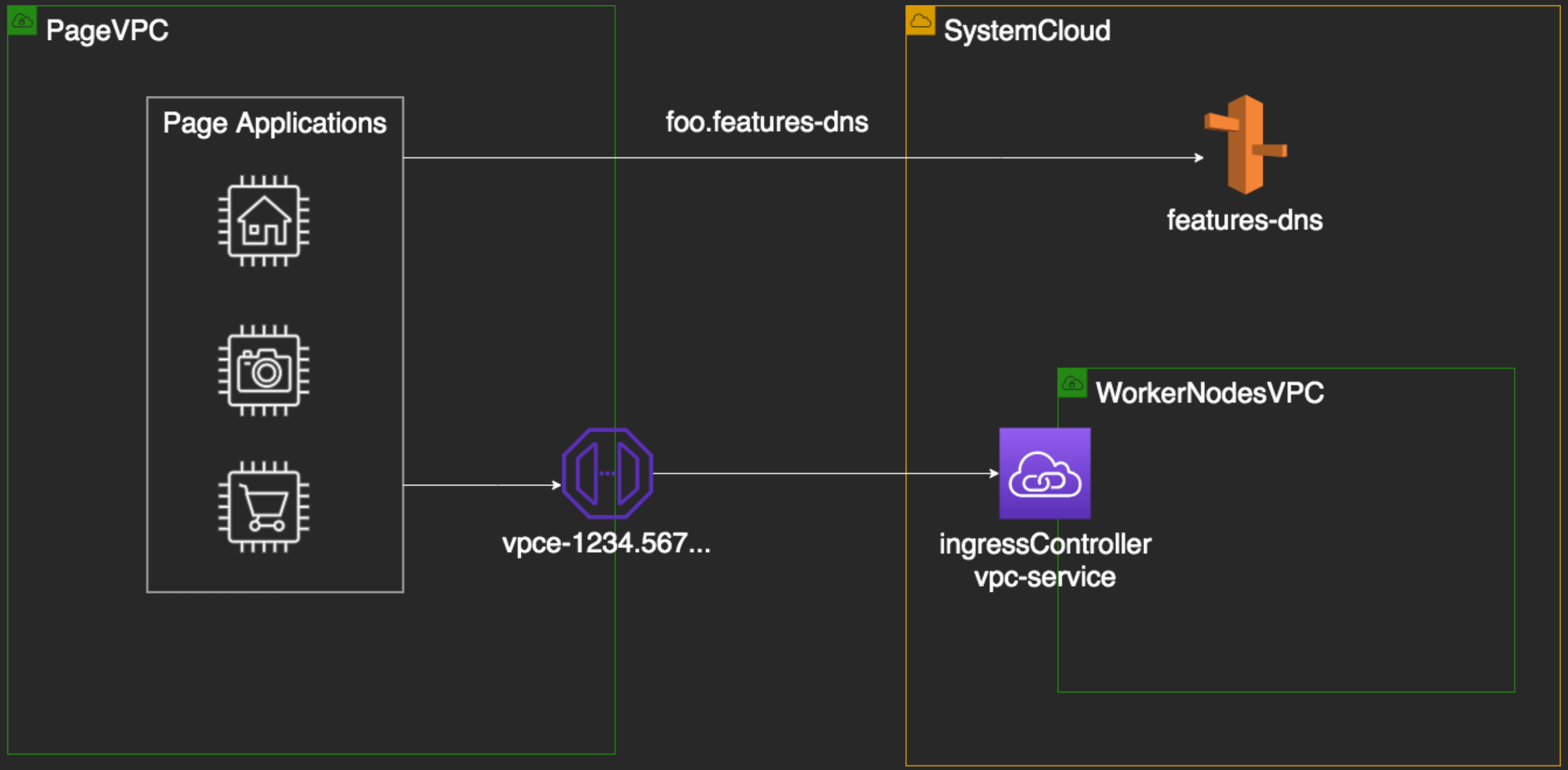
# High-level architecture



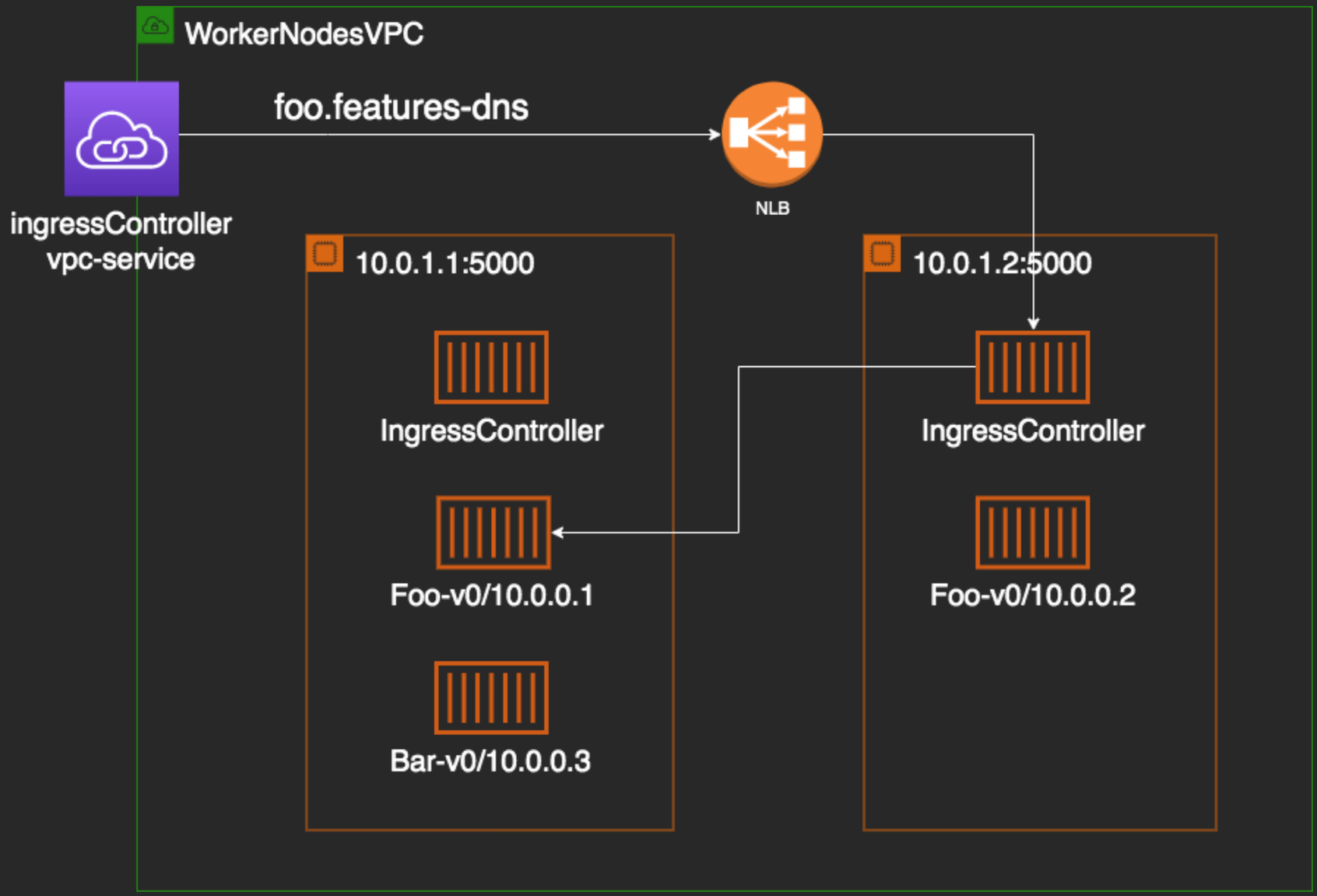
# Running Foo/Bar in Amazon EKS



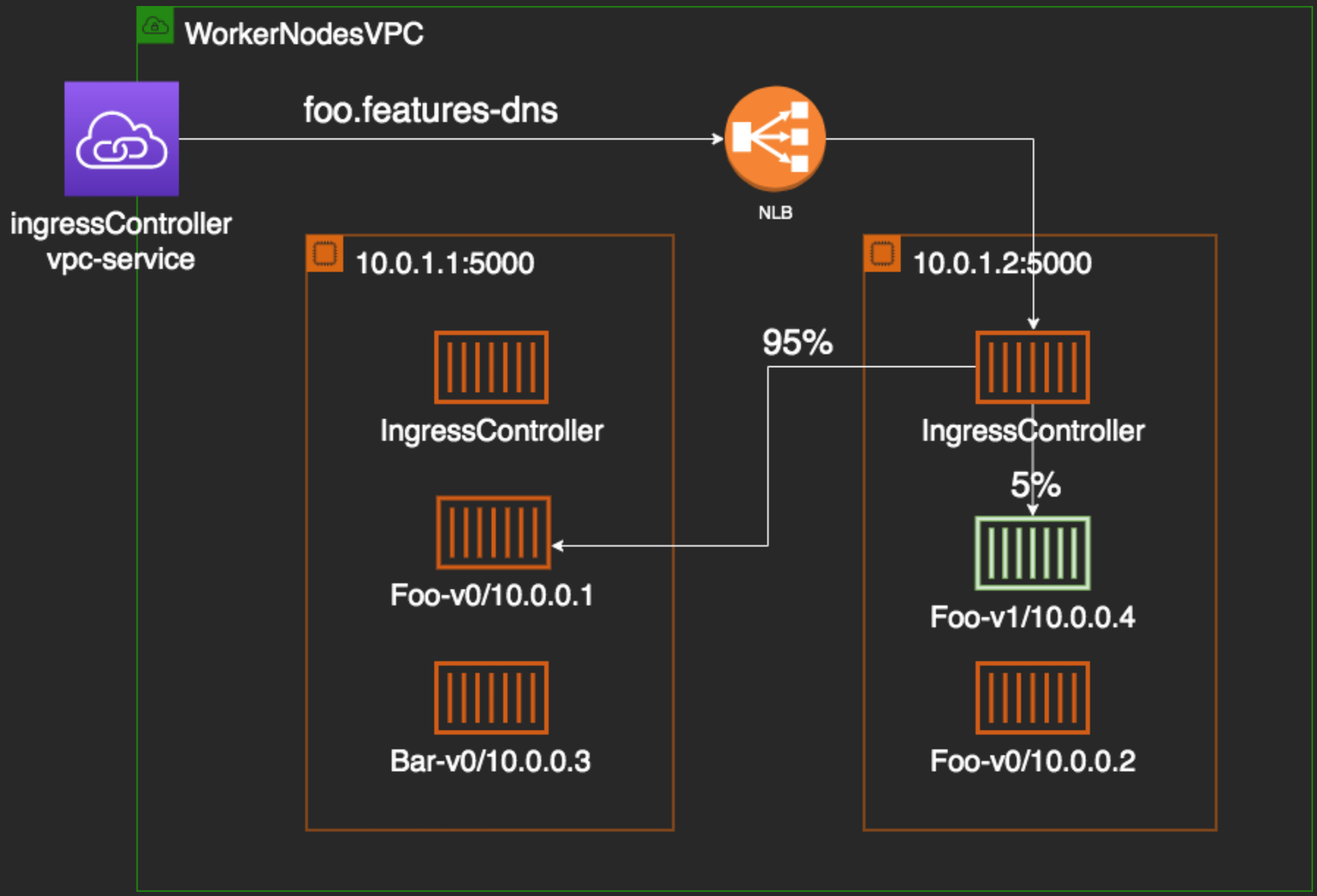
# Routing requests to Foo



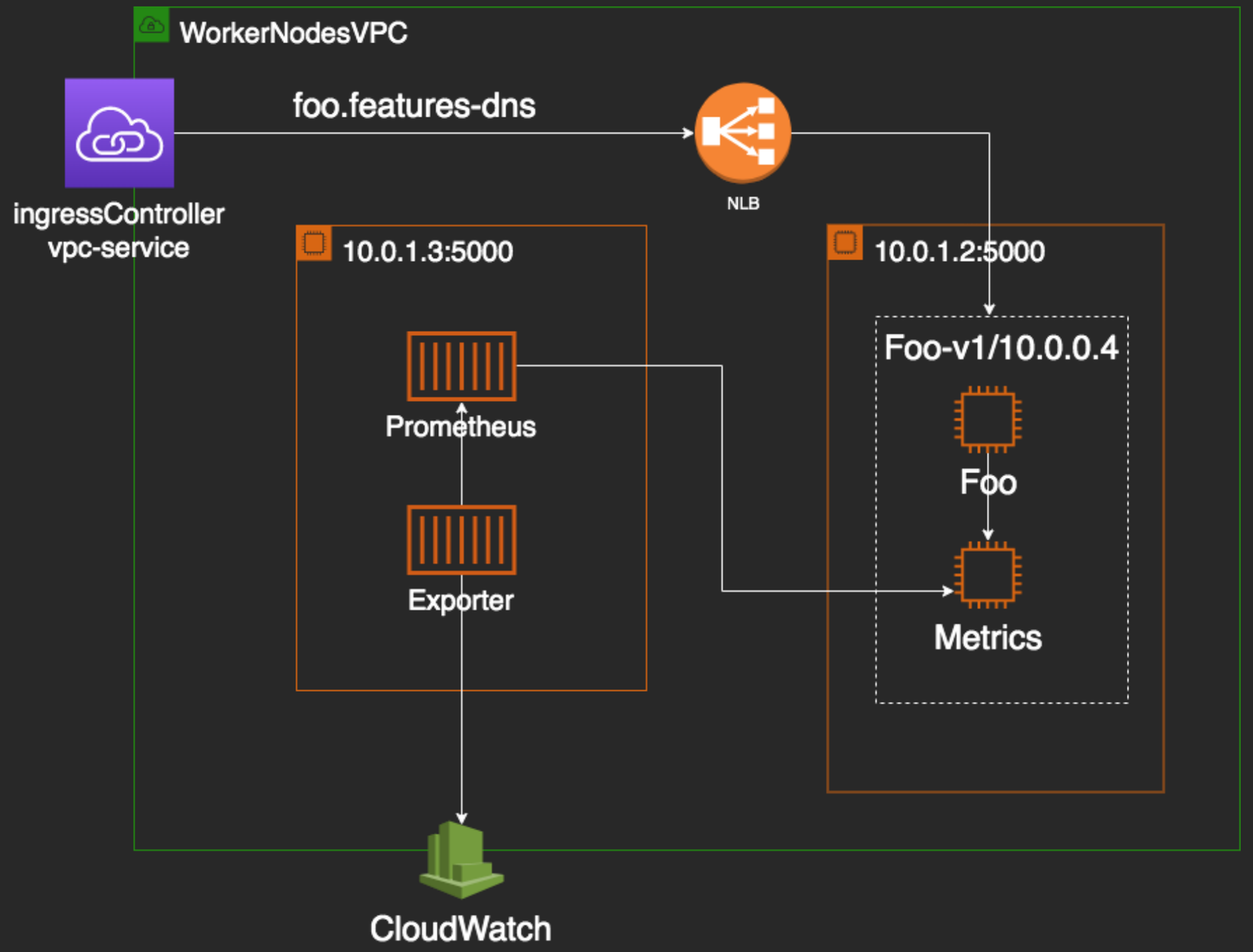
# Routing requests to Foo



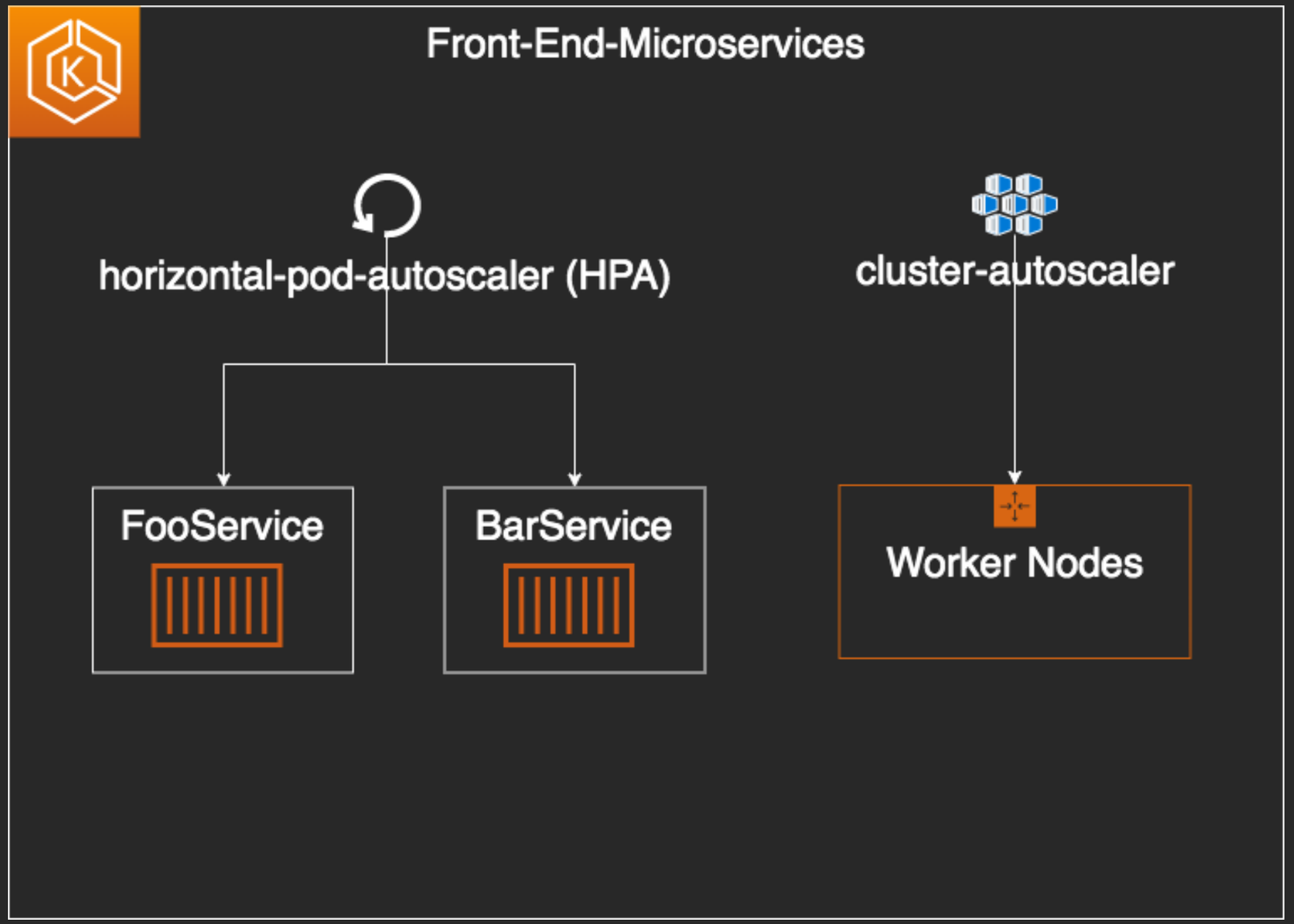
# Gradually deploy Foo



# Gradually deploy Foo



# Highly available elasticity



# Baby steps

- ✓ Running services
- ✓ Exposed endpoints
- ✓ Gradual deployments
- ❖ Elastic scaling

# Running Foo without elasticity

- Success rate within SLAs
  - Small size payload (10kb)
  - Fast response (5ms)
- Drops on availability
  - Deployments
  - Scaling

# How we enabled elasticity?

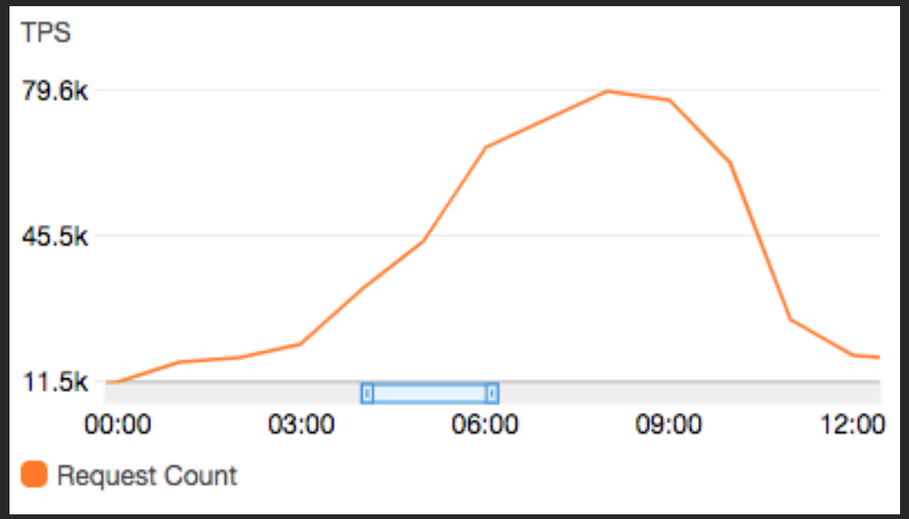
- Cold start time
- Success rate
- 60% CPU

# We did it!

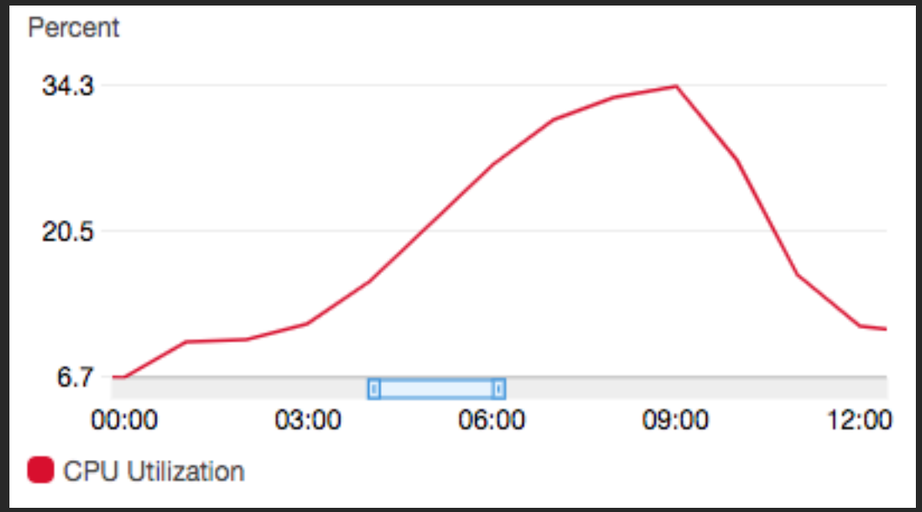
# How we learned to stop worrying and ❤ Amazon scale

# Highly available elasticity

## Request count

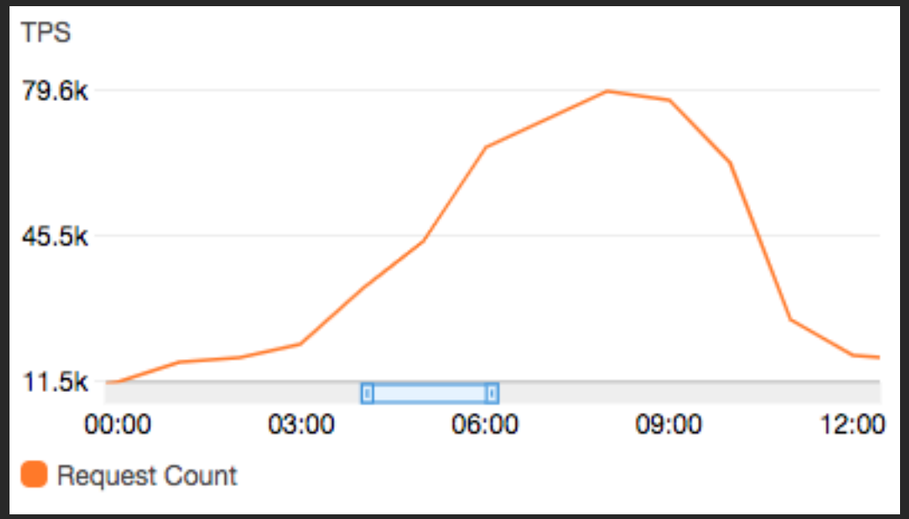



## CPU utilization

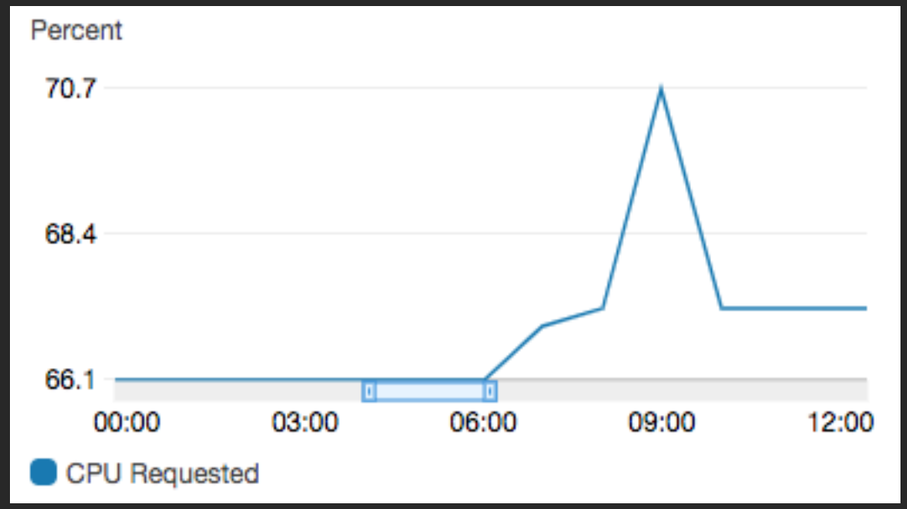


# Highly available elasticity

## Request count

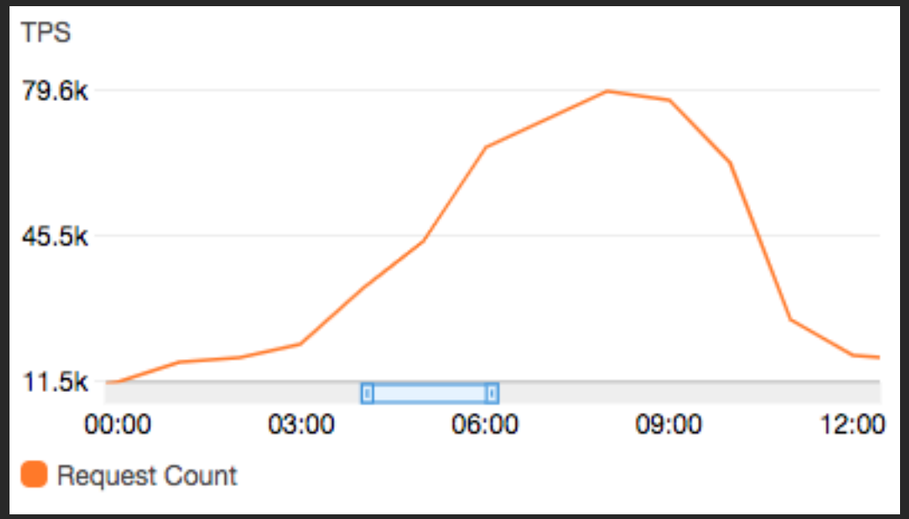



## CPU requested

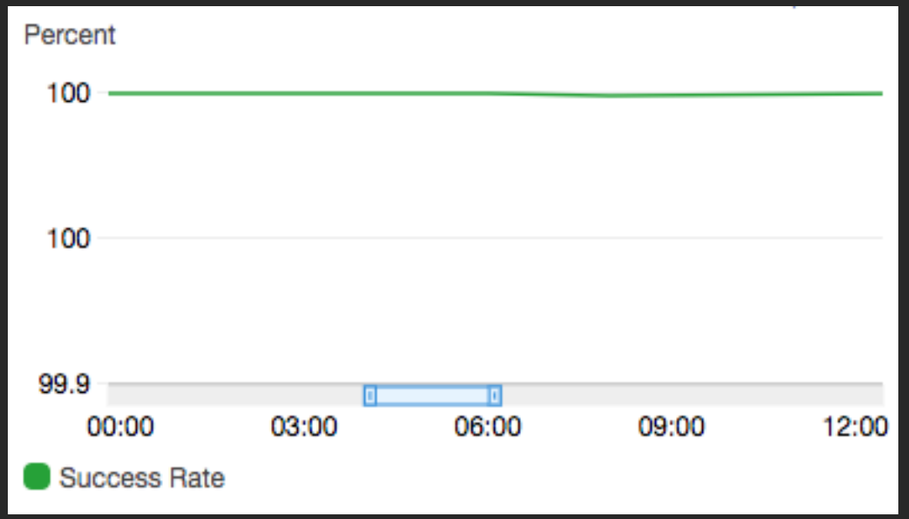


# Highly available elasticity

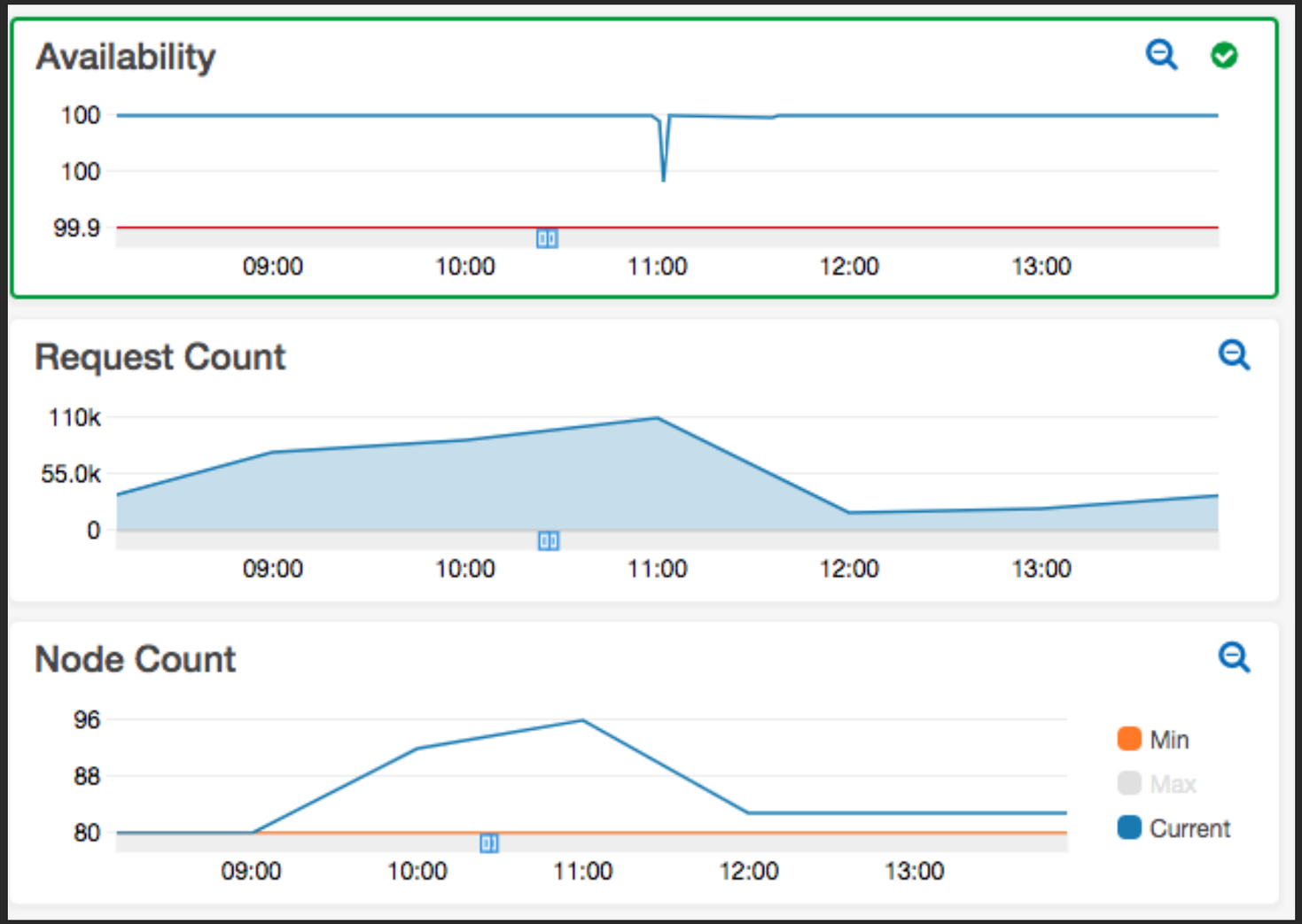
## Request count




## Success rate



# Highly available elasticity

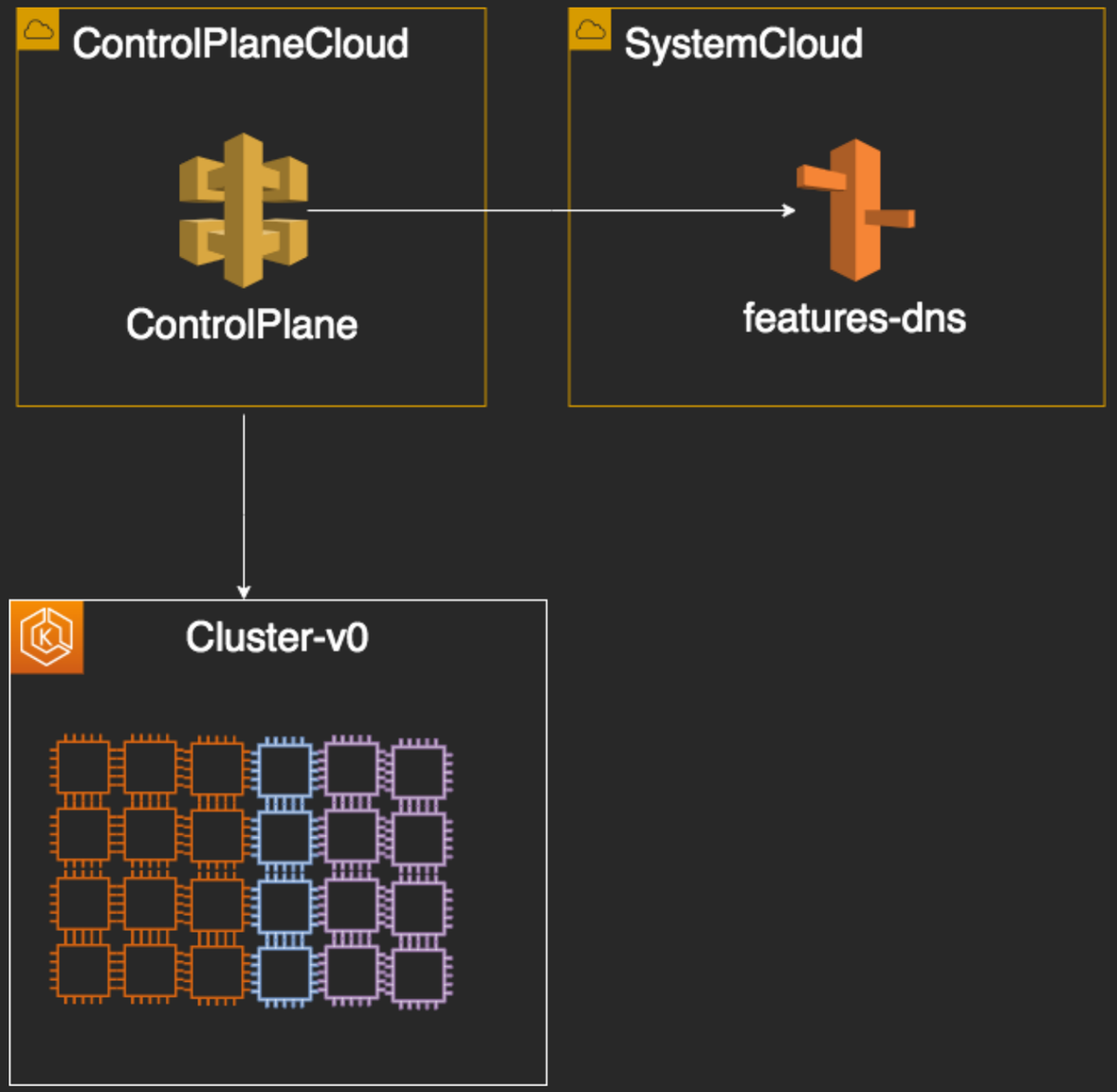


# And we updated our cluster

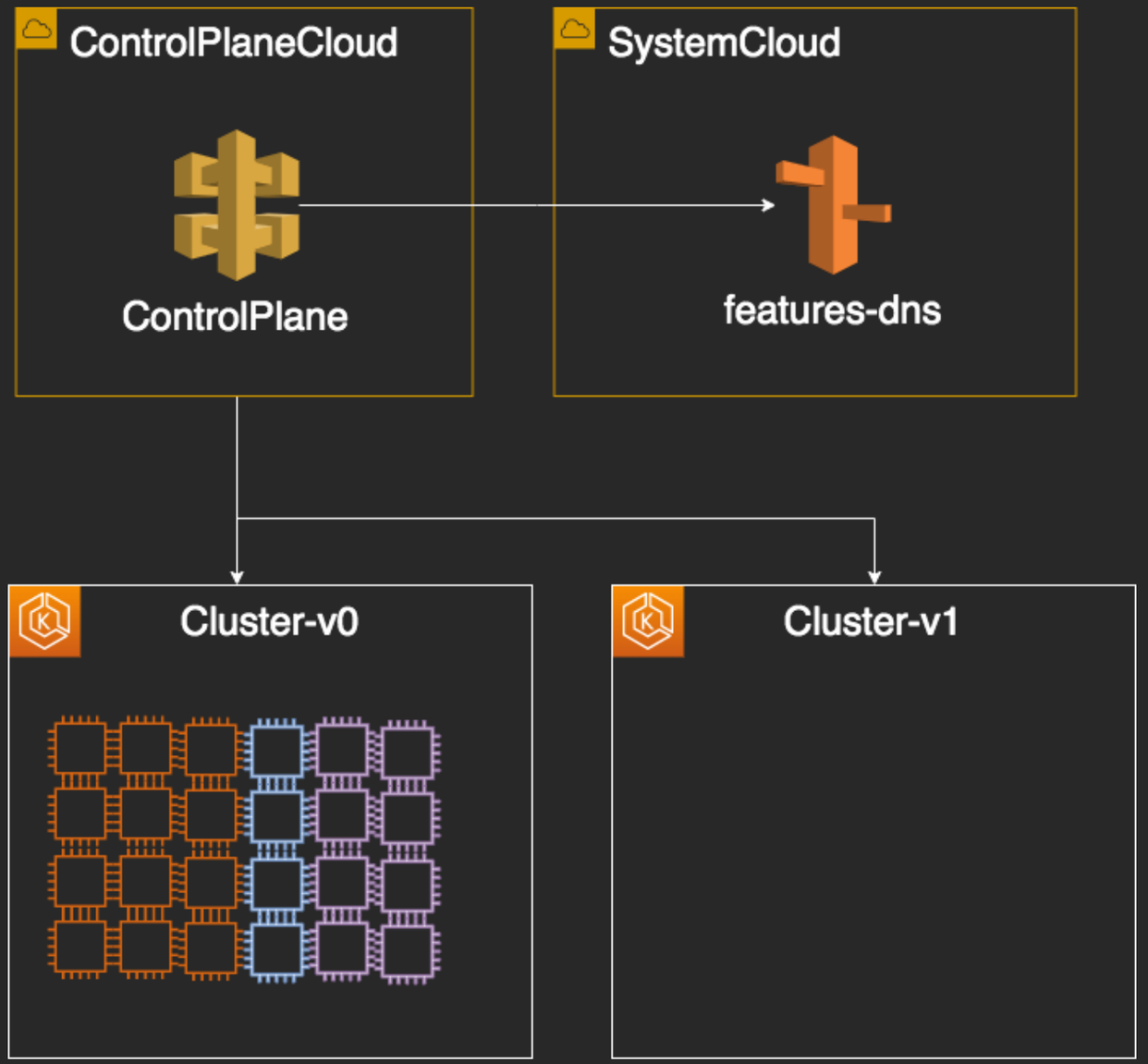
# Ephemeral clusters

- Infrastructure automation
- Cluster rollouts

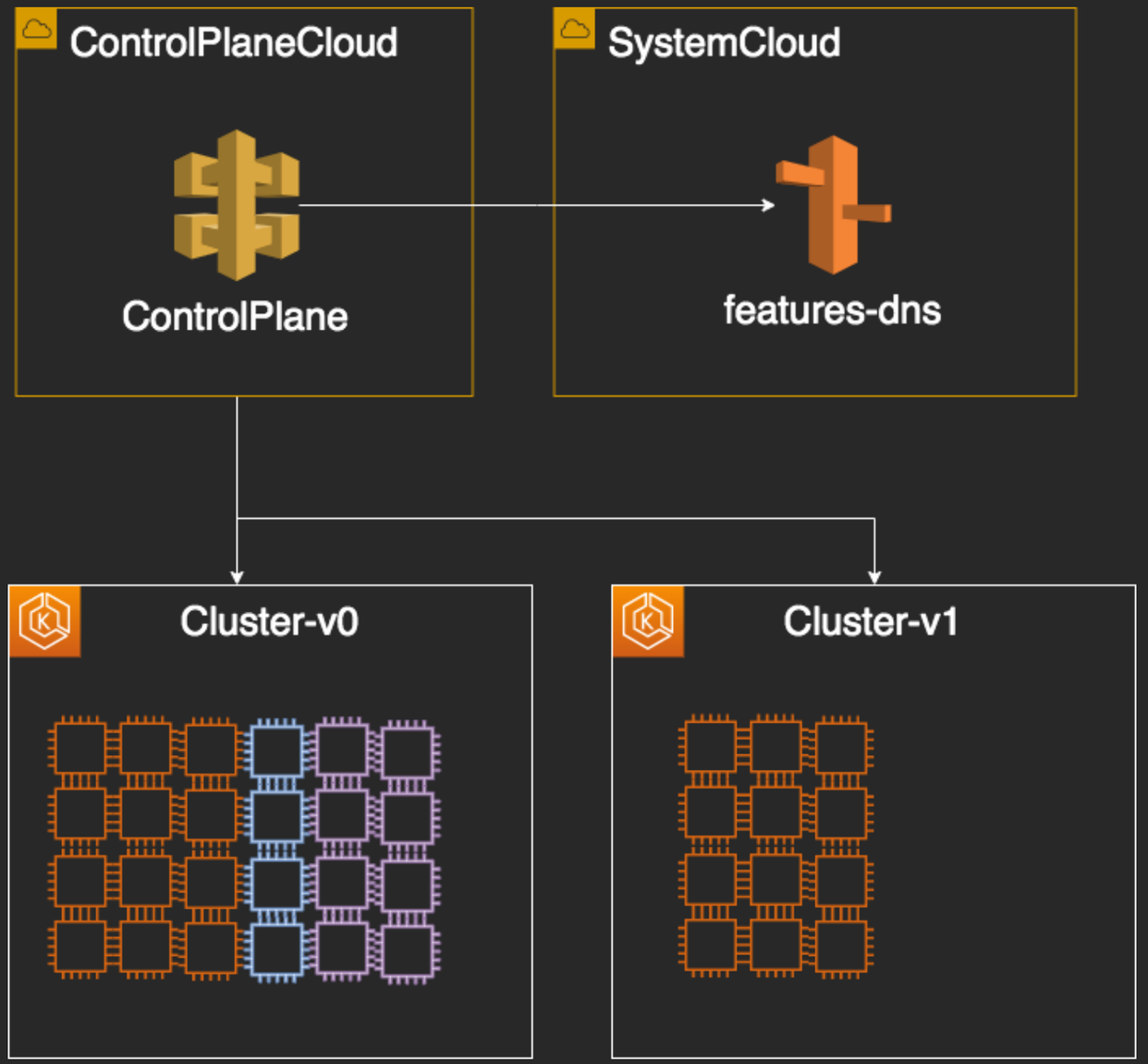
# Cluster rollouts



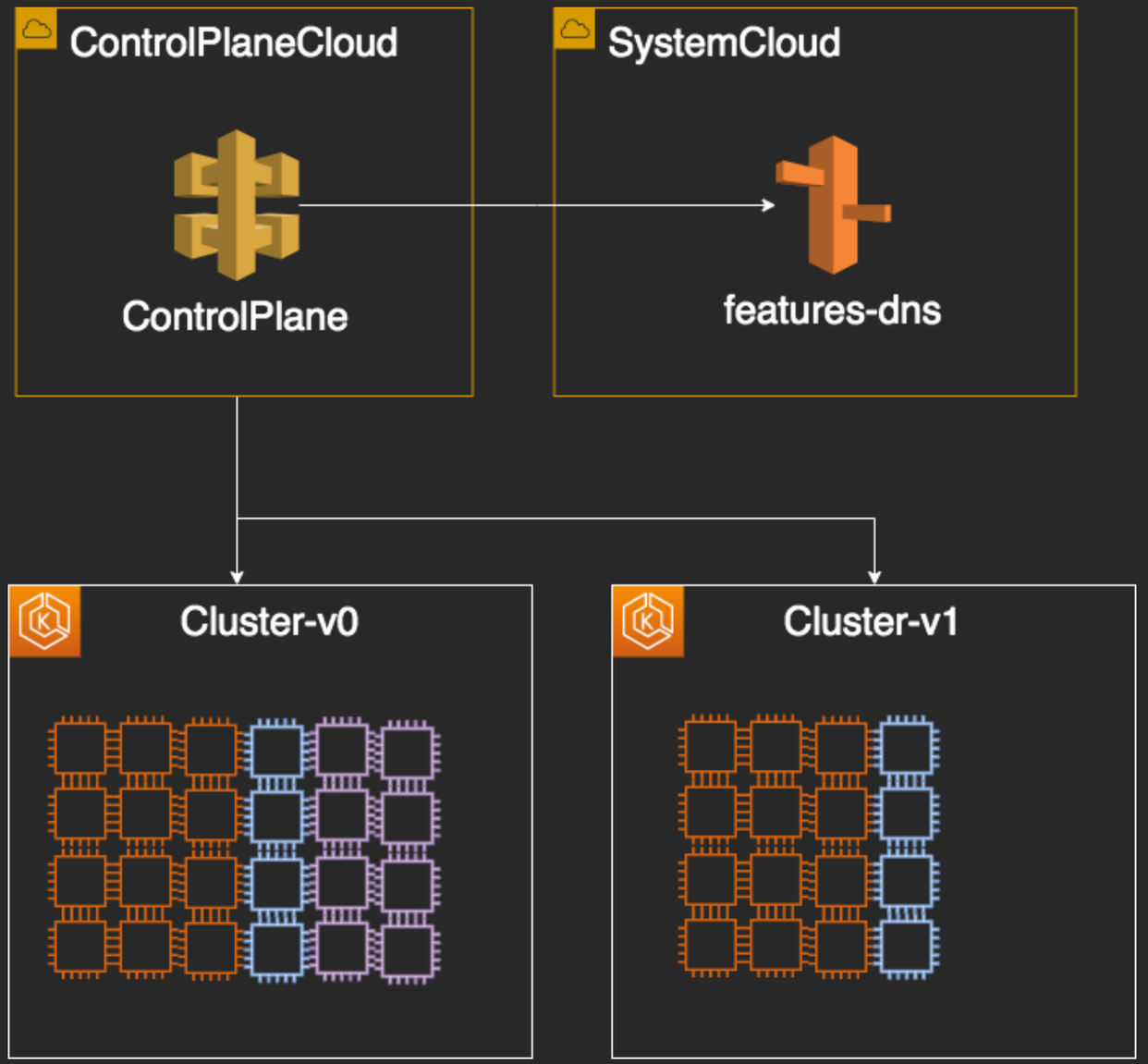
# Cluster rollouts



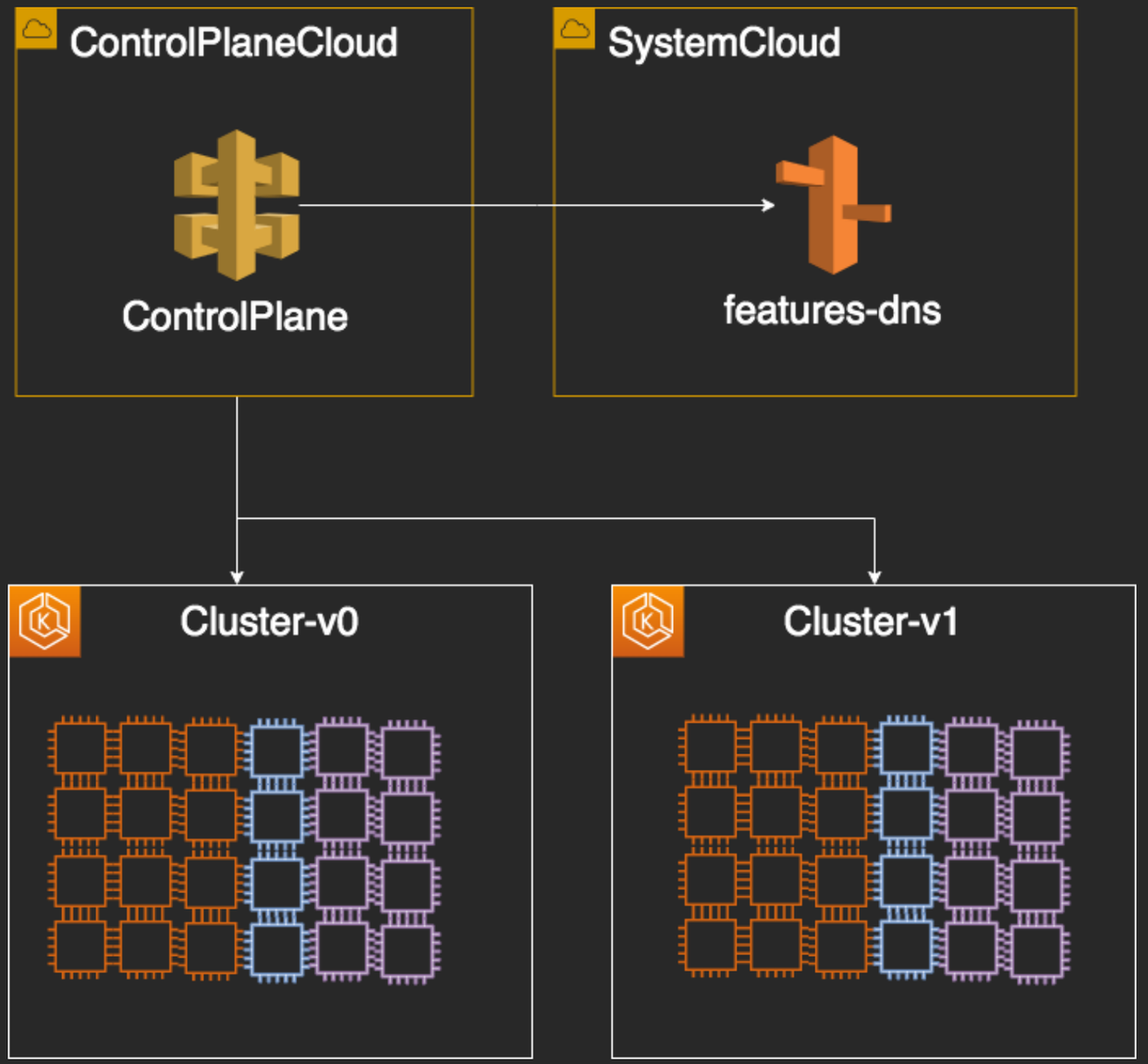
# Cluster rollouts



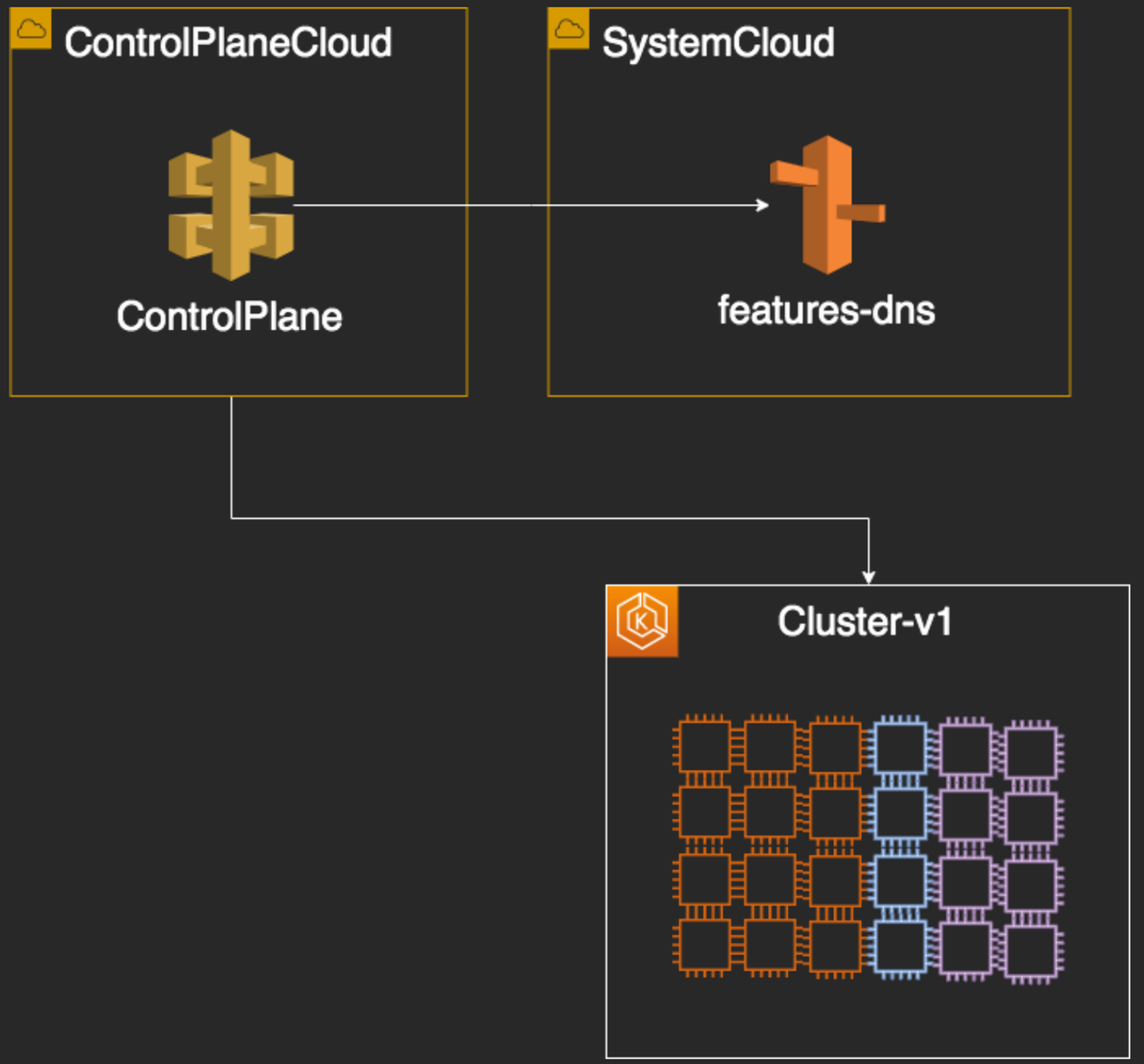
# Cluster rollouts



# Cluster rollouts



# Cluster rollouts

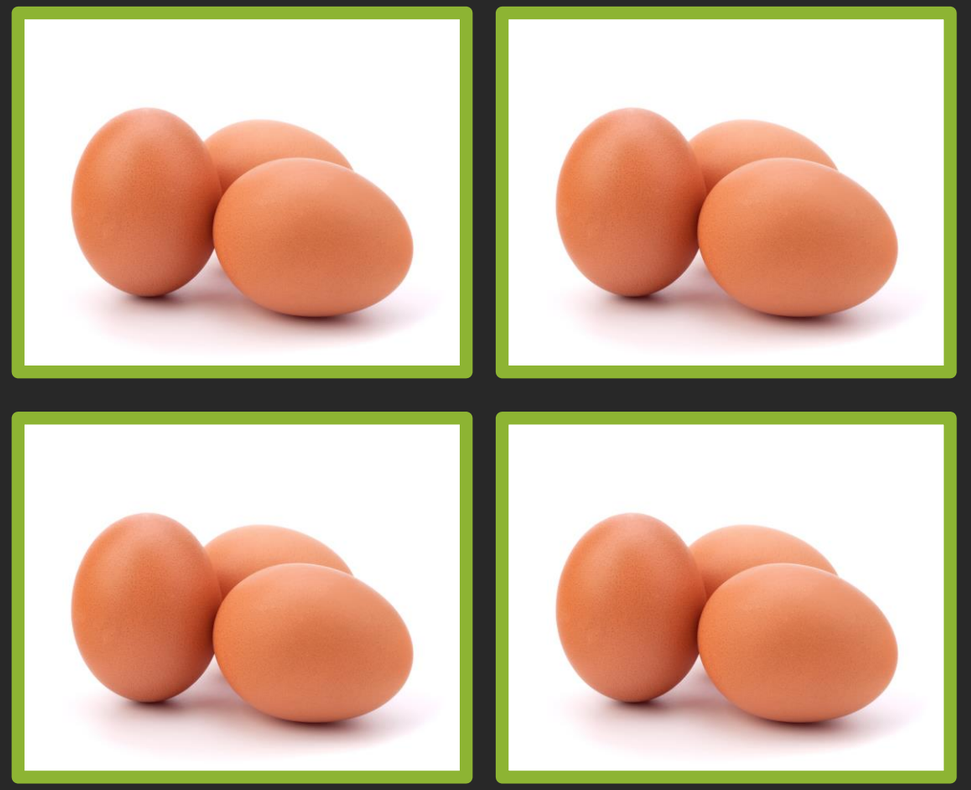


# Cell-based architecture

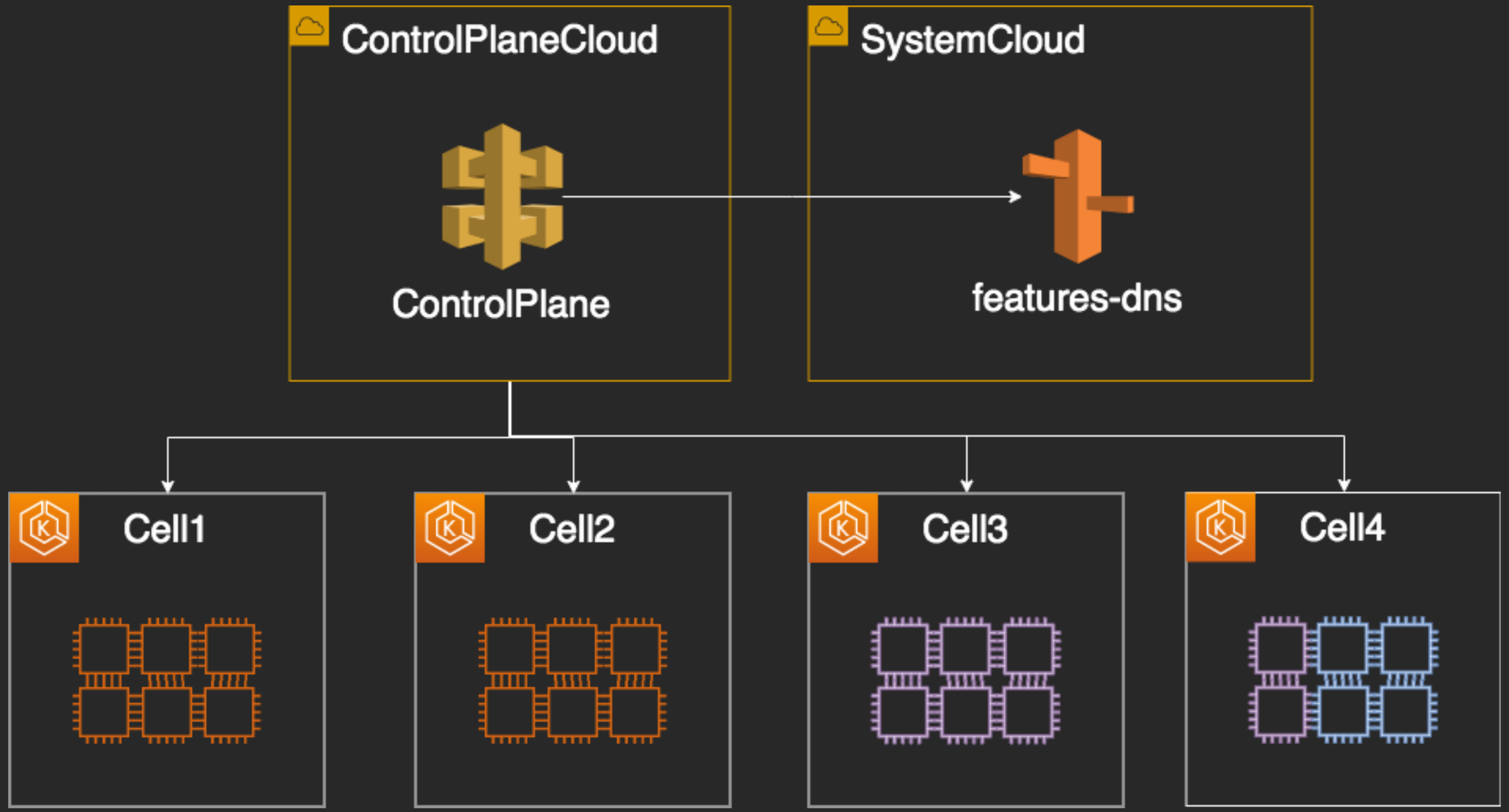
- Don’t put your eggs in one bucket

# Cell-based architecture

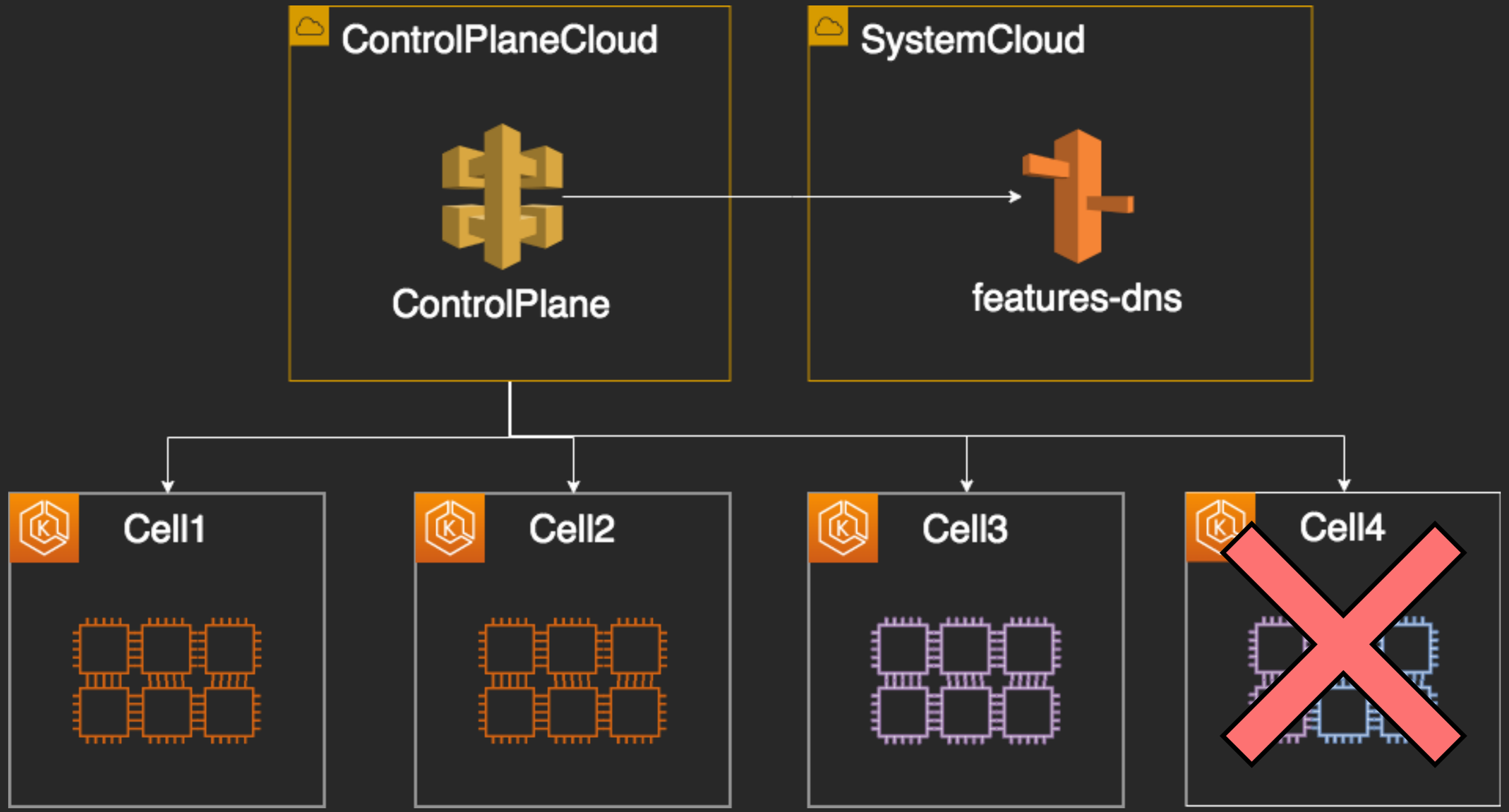
- Don’t put your eggs in one basket
- Higher testability
- Higher scalability
- Reduces blast radius



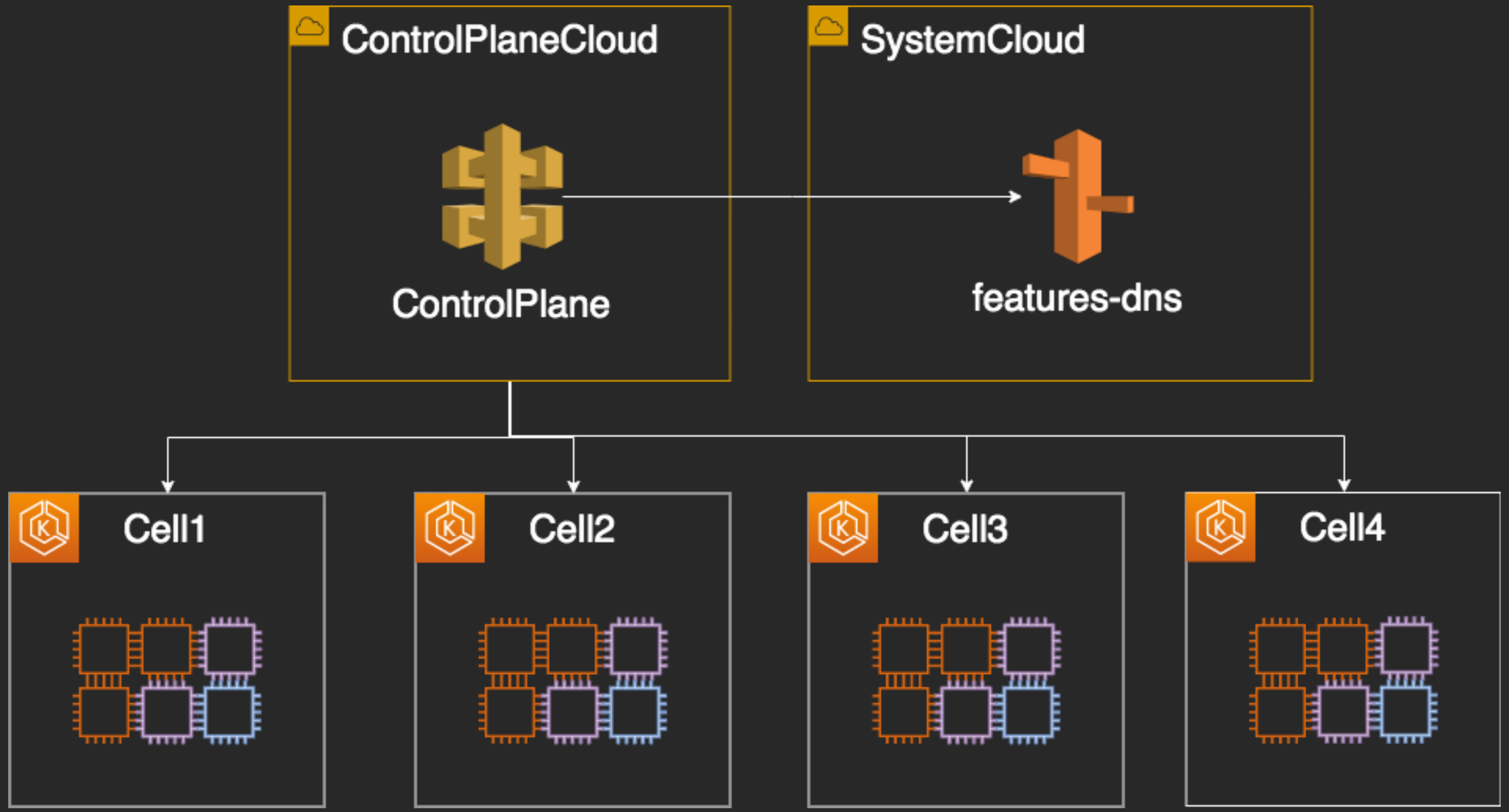
# Cell-based architecture



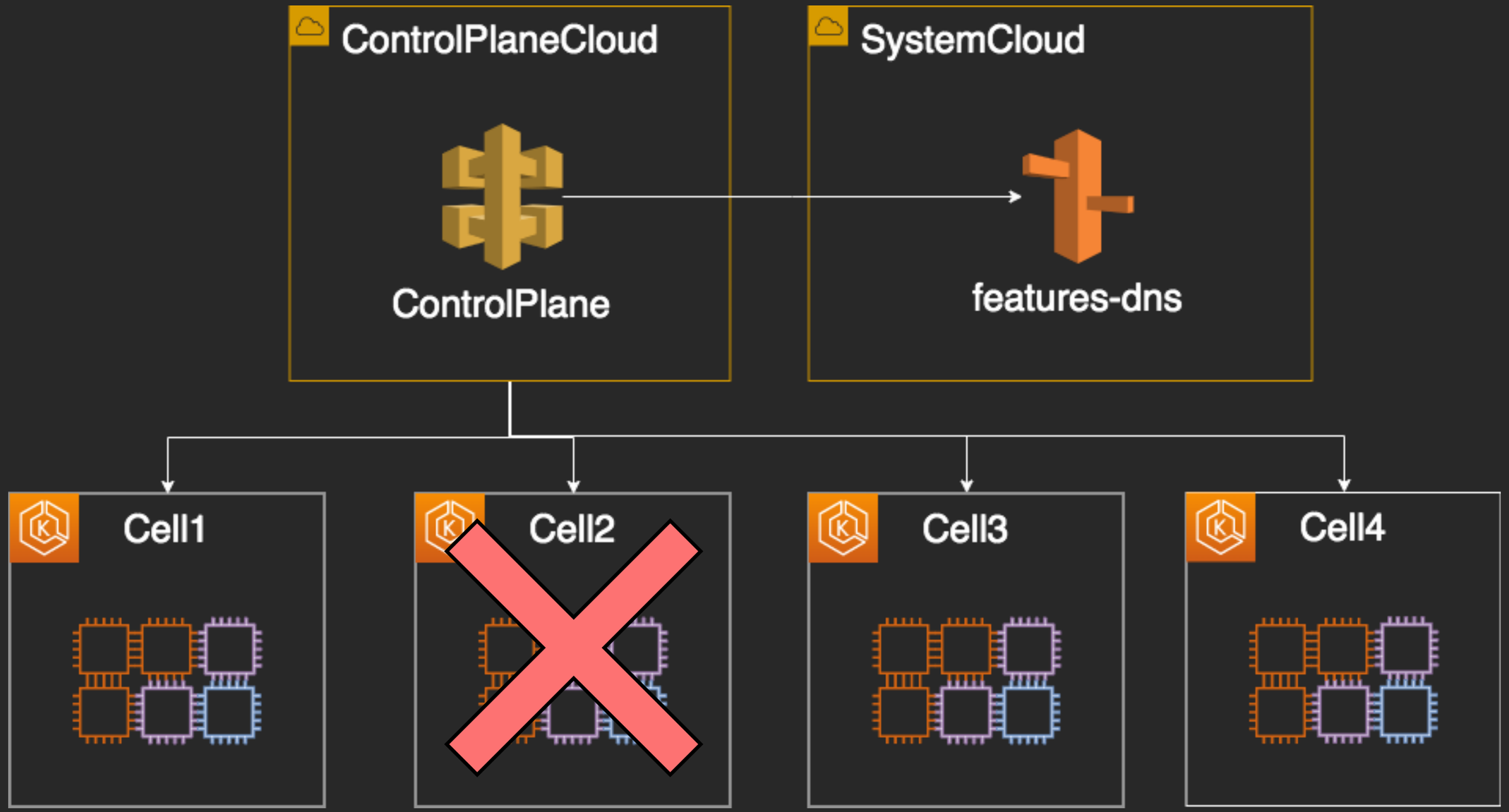
# Cell-based architecture

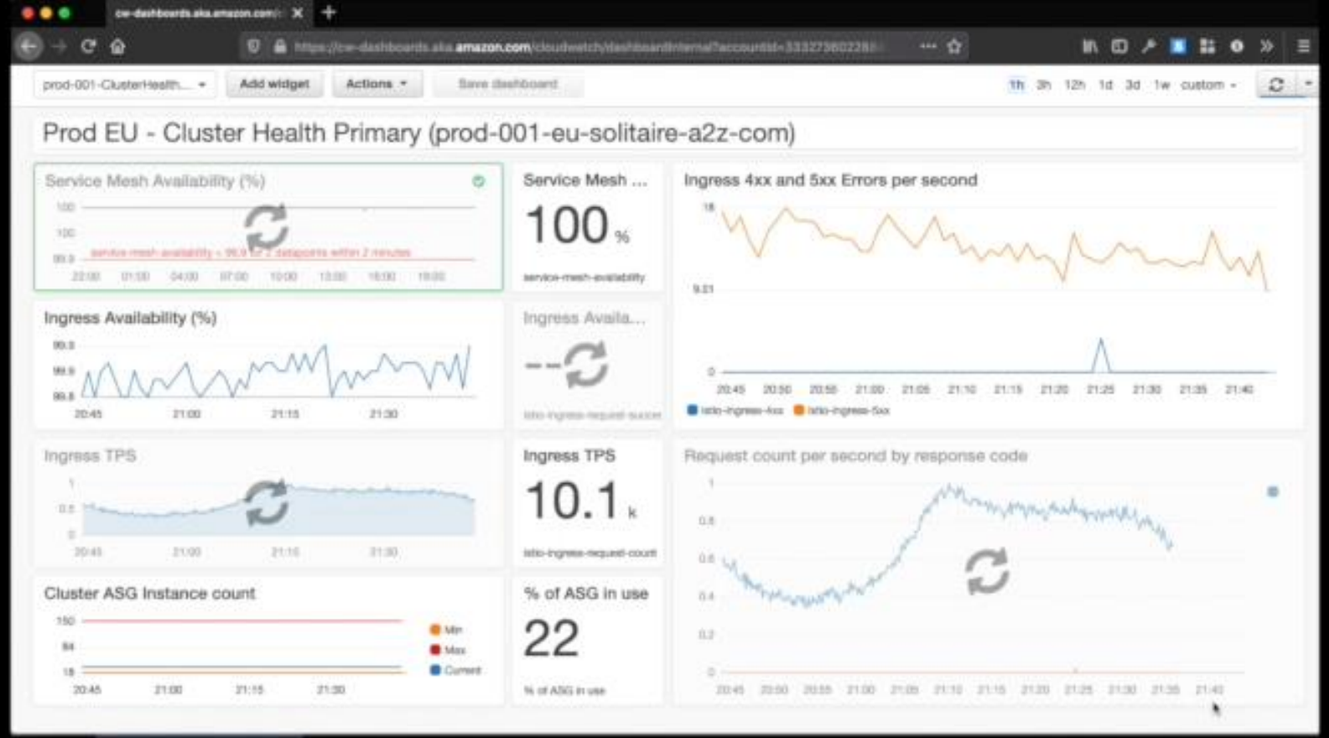


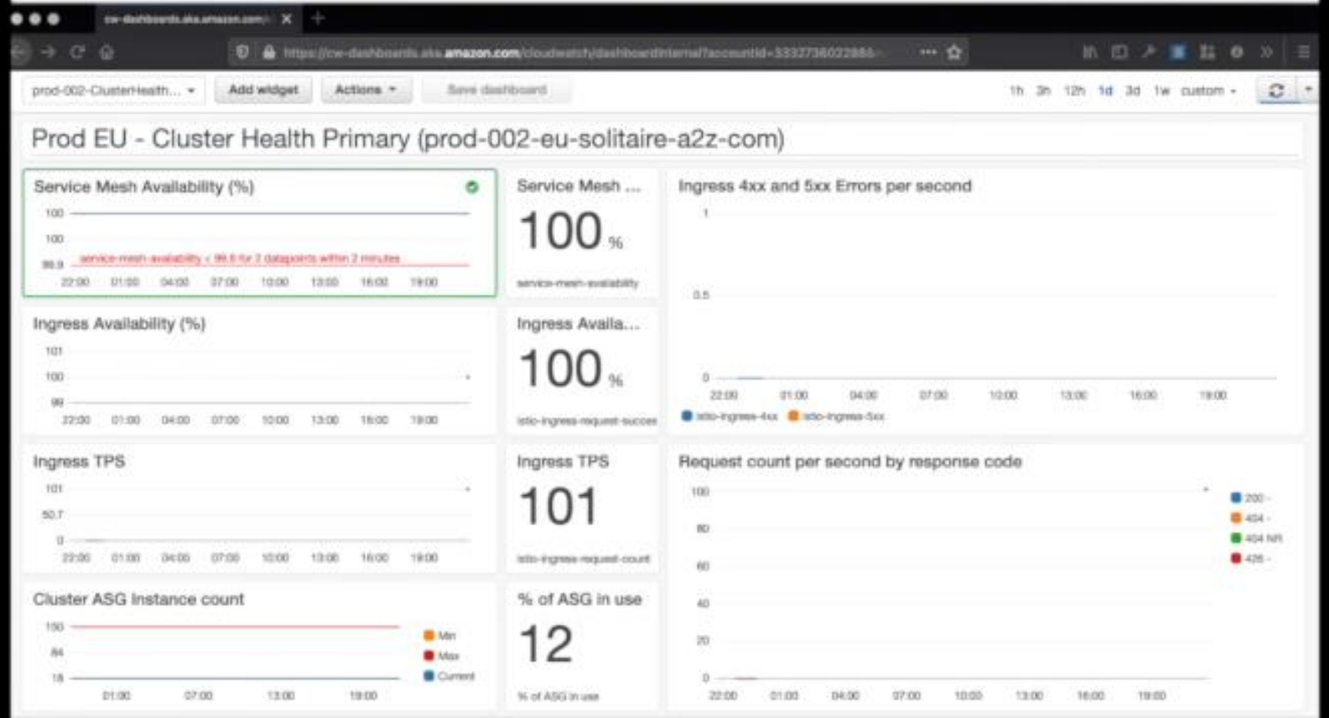
# Cell-based architecture



# Cell-based architecture




Add widget
Actions
Save dashboard
1h 3h 12h 1d 3d 1w custom
Prod EU - Cluster Health Primary (prod-001-eu-solitaire-a2z-com)
Service Mesh Availability (%)
Service Mesh ...
100 %
service-mesh-availability
Ingress 4xx and 5xx Errors per second
Ingress Availability (%)
Ingress Availa...
Ingress TPS
Ingress TPS
10.1 k
istio-ingress-request-count
Request count per second by response code
Cluster ASG Instance count
Min
Max
Current
% of ASG in use
22
% of ASG in use


Add widget
Actions
Save dashboard
1h 3h 12h 1d 3d 1w custom
Prod EU - Cluster Health Primary (prod-002-eu-solitaire-a2z-com)
Service Mesh Availability (%)
service-mesh-availability < 99.6 for 3 datapoints within 2 minutes
Service Mesh ...
100 %
service-mesh-availability
Ingress 4xx and 5xx Errors per second
Ingress Availability (%)
Ingress Availa...
100 %
Ingress TPS
Ingress TPS
101
istio-ingress-request-count
Request count per second by response code
Cluster ASG Instance count
Min
Max
Current
% of ASG in use
12
% of ASG in use

# Key takeaways

Ownership is hard, managed services make it easier

Monitor used and reserved resources to optimize over time

Understand your cold start behavior and SLAs to configure elasticity

Design for ephemeral infrastructure

# Thank you!

AWS re:Invent



aws

# Please complete the session survey in the mobile app.