AWS re:Invent

NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

CON311-R

Best practices for deploying microservices on Amazon ECS

Uttara Sridhar
Software Development Manager
Amazon Web Services

Vibhav Agarwal
Senior Product Manager
Amazon Web Services

Agenda

Microservices: What and why

Building microservices with Amazon ECS

Best practices for Amazon ECS deployments:

- Continuous deployment
- Safety
- Speed

Amazon Elastic Container Service (Amazon ECS) overview

AWS Cloud

Amazon ECS

Fully managed container service
Highly performant and scalable

Amazon EC2

AWS Regions, AWS Wavelength,
AWS Local Zones, AWS Outposts

AWS Fargate

Serverless container
compute engine

Amazon ECS Anywhere

Customer owned
container hosts

Amazon ECS is a serverless container orchestrator

Managed by AWS, no cluster versions, no data store to maintain or scale, free to use with Amazon EC2 or AWS Fargate

Works with different “compute” types, including AWS Fargate

Fargate is a serverless compute engine for containers

Managed by AWS, no AMIs to maintain, no Amazon EC2 instances to provision, scale or manage, pay for what you need

Fargate is where your containers will run; you don't interact with it directly

Amazon ECS customers innovate quickly and efficiently

Discover and distribute COVID-19 vaccines

Address exponential demand for delivering essentials

Stream entertainment globally

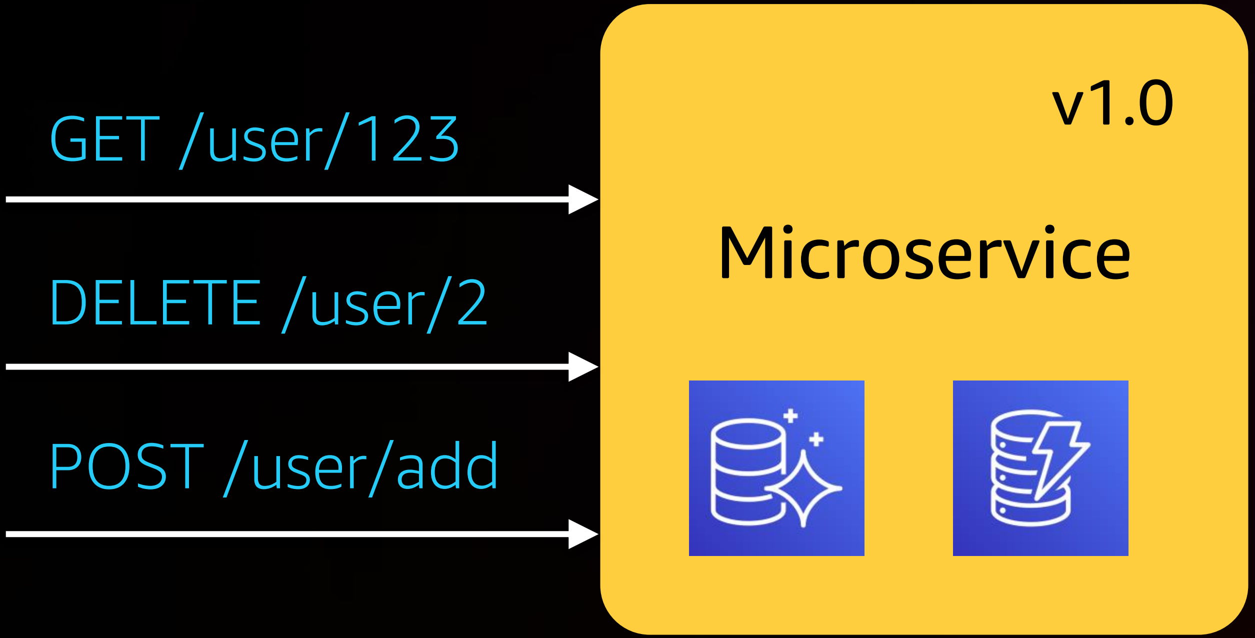
Microservices: What and why

What are microservices?

"A microservice architectural style is an approach to developing a single application as a suite of small, isolated services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API."

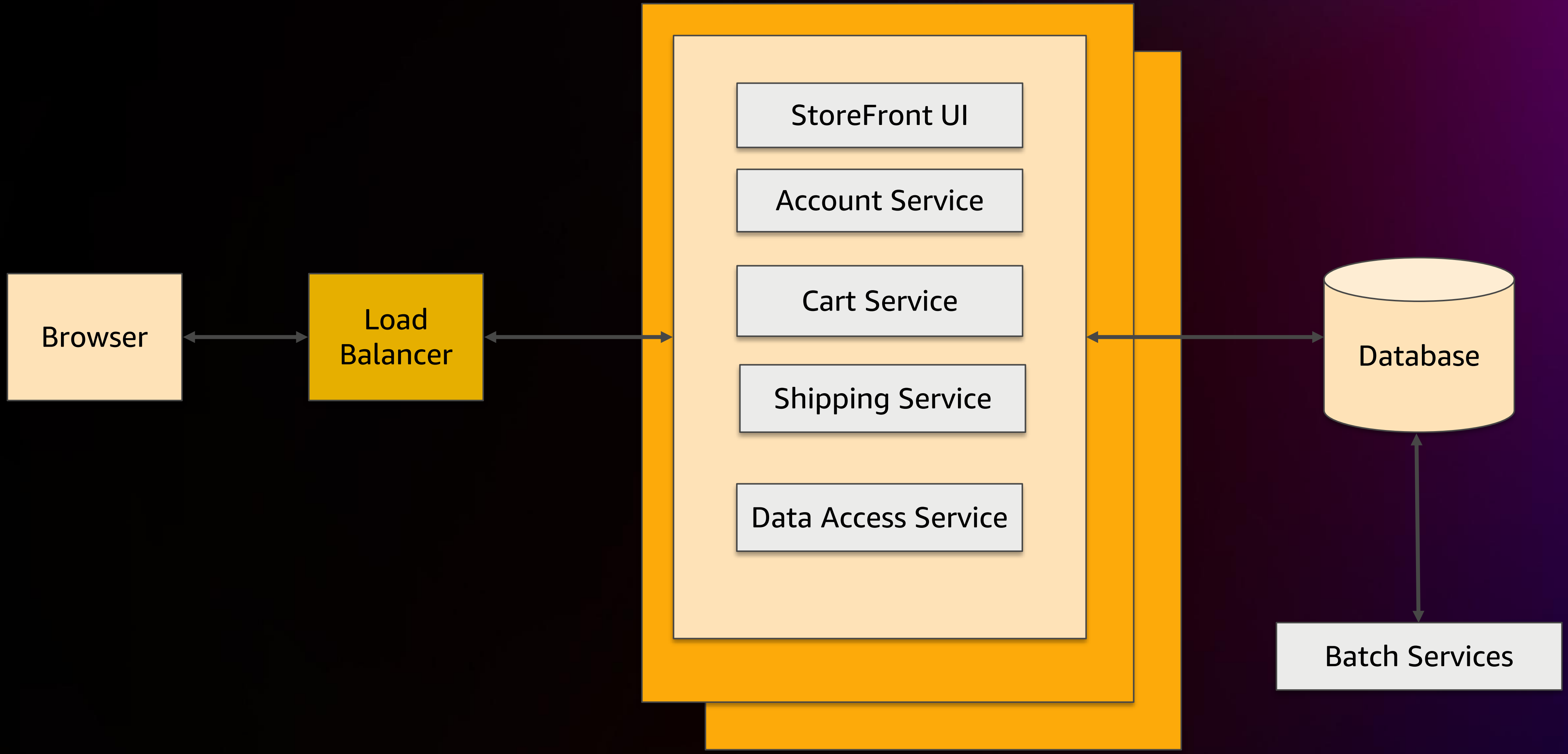
James Lewis and Martin Fowler

Microservice tenets

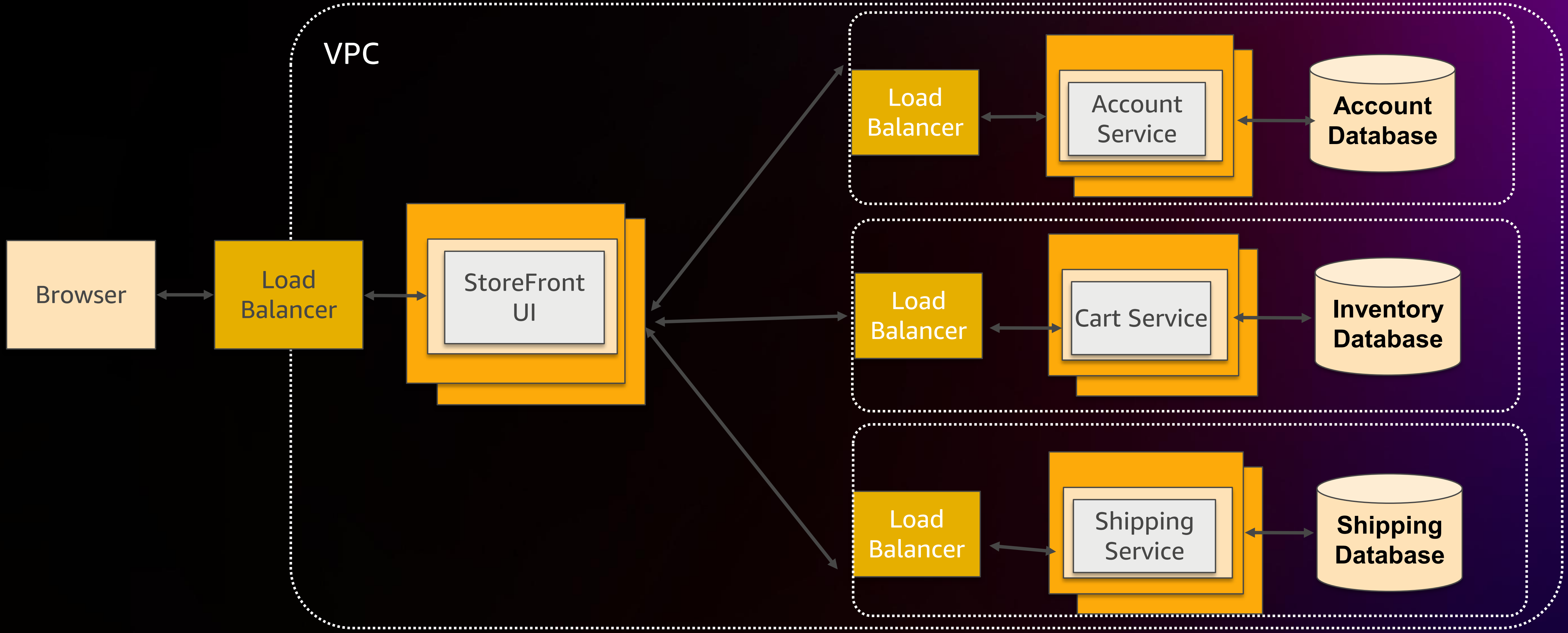


- Well-defined contract
- Autonomous
- Deployed separately
- Scaled independently
- Single responsibility
- Owns, manages, and encapsulates its data
- Decentralized ownership

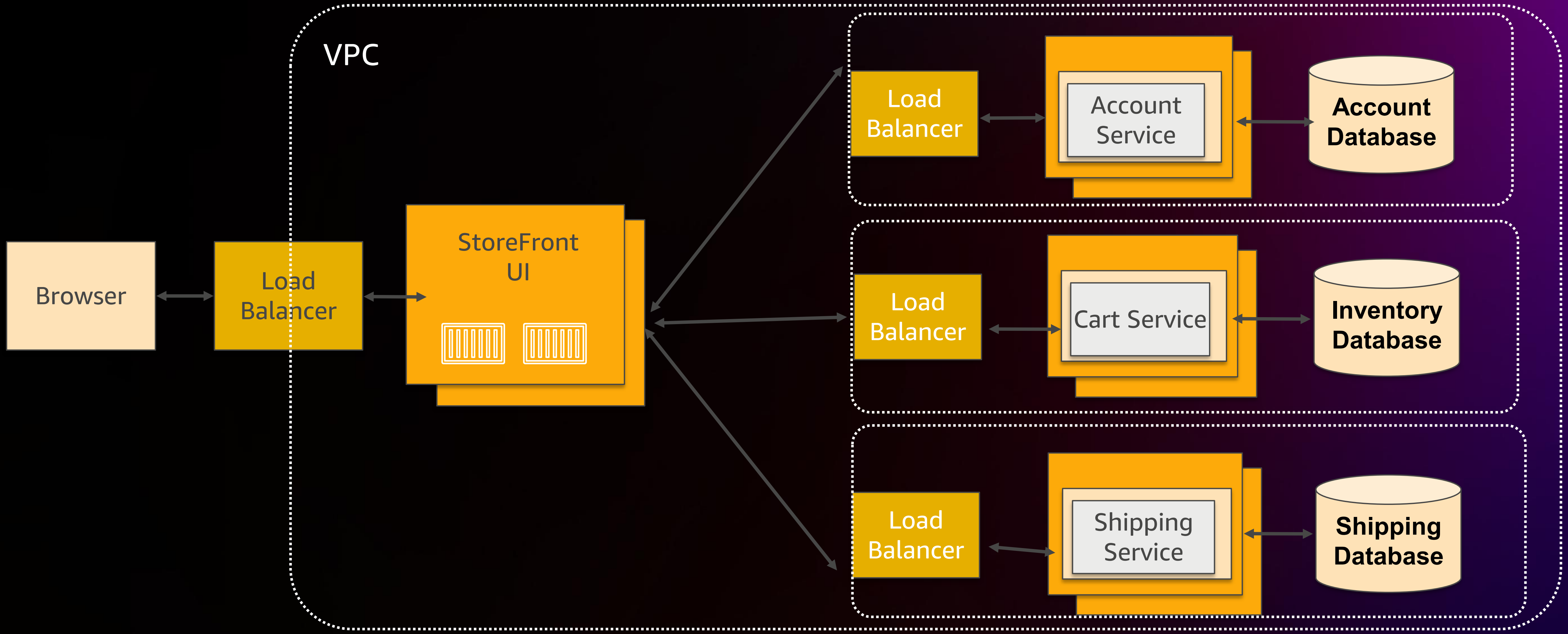
Monolithic architectures are tightly coupled



Microservices architecture

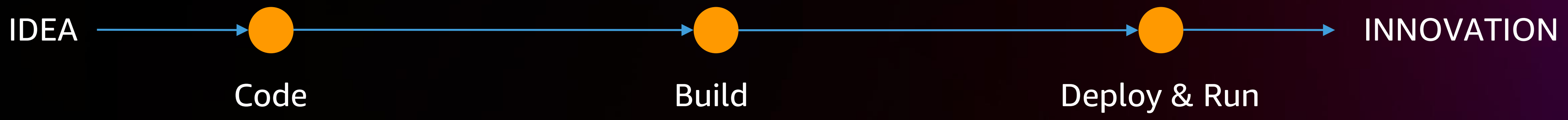


How to run microservices? Containers!

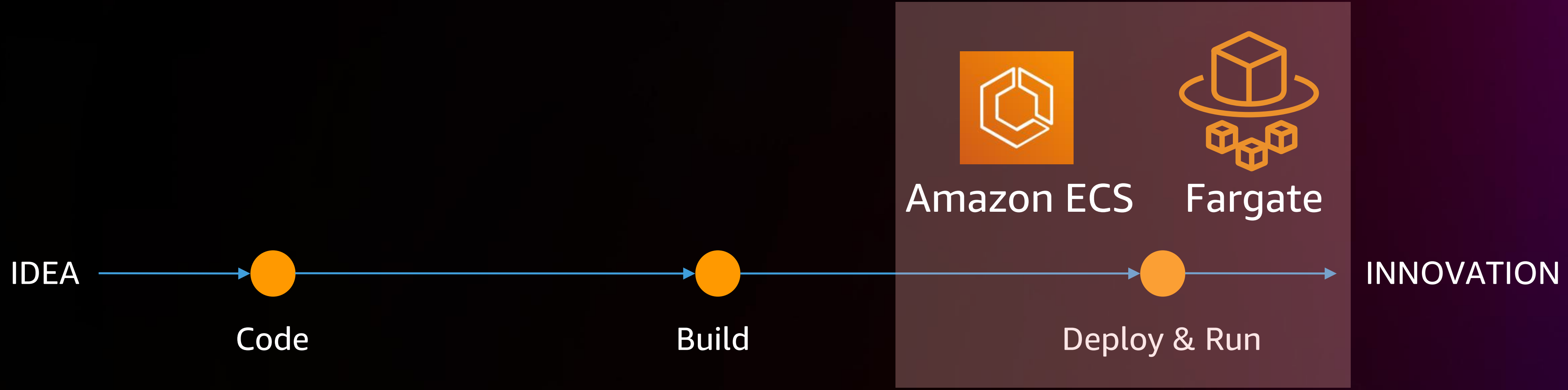


Containers + microservices: Modern digital assembly line for software delivery

Deploying applications as containerized microservices accelerates innovation while maintaining reliability and cost efficiency



Amazon ECS + Fargate: Simple and powerful microservice deployments



Microservice deployments in Amazon ECS

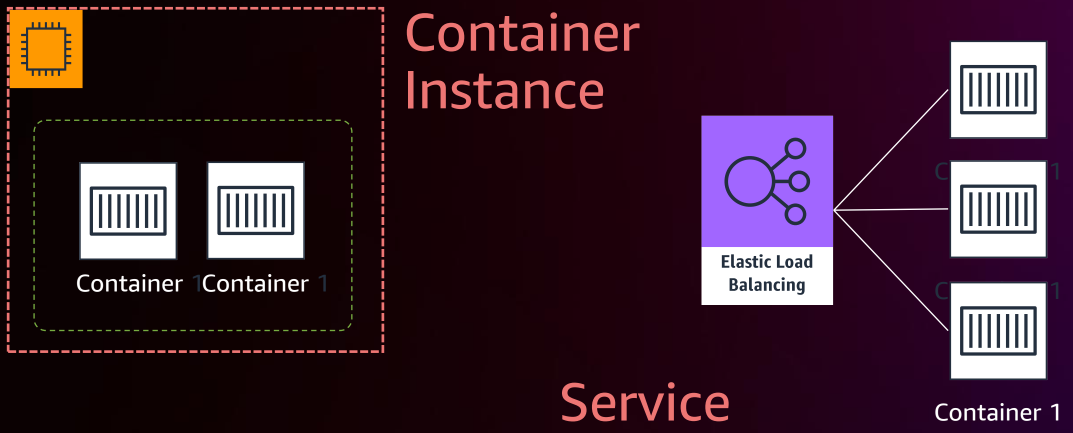
Amazon ECS constructs

Task definition

- Template used by Amazon ECS to launch tasks
- Parallels to docker run parameters
- Defines requirements:
  - CPU/memory
  - Container image(s)
  - Logging
  - IAM role
  - Etc.

Cluster

- Resource grouping and isolation
- IAM permissions boundary



Container Instance

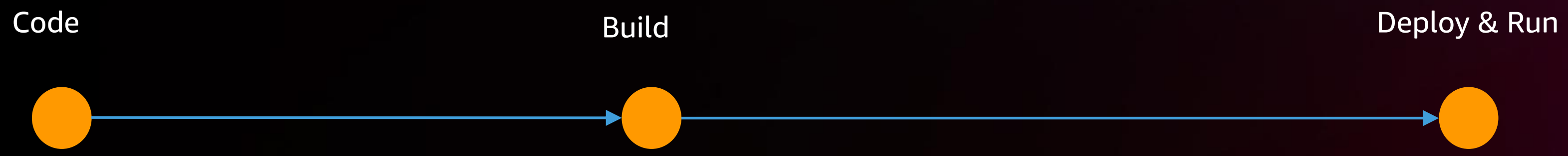
Task

- Running instance of a task definition
- One or more containers

Service

- Maintains desired # of running tasks
- Replaces unhealthy tasks
- ELB integrated

Amazon ECS services are commonly used to deploy microservices



Git

- App code
- Dockerfile

AWS CodeBuild/ Jenkins

- Container images
- Amazon ECR image repo

Amazon ECS

- Service definition
- Task definition (incl. container images for task)

Fargate

- Provision CPU, memory, networking to run task

Anatomy of an Amazon ECS service deployment

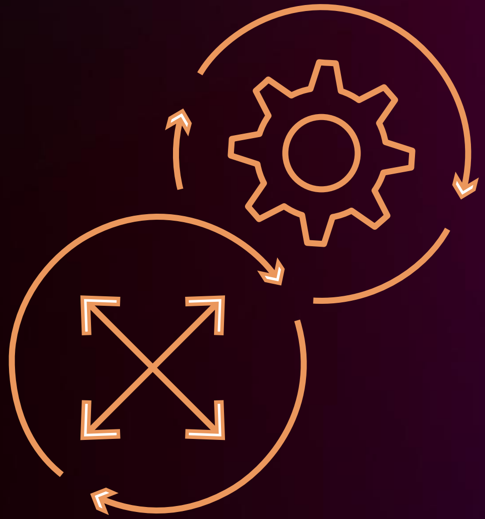
1. Task definition registration
2. Create or update a service
   1. Amazon ECS service scheduler
      - Provision compute and ENI for each task, Pull container images for task, Task runs
      - For Load Balancer service: Amazon ECS registers task to target group, Amazon ECS checks LB health check status for tasks
3. Service is deployed (desired count of tasks are running and healthy)

What is NOT an Amazon ECS deployment

Scaling a service

Autoscaling

Updating the image tag



A deployment means rolling out a new application version and replacing the old version (if any)

Two key ways to deploy to Amazon ECS

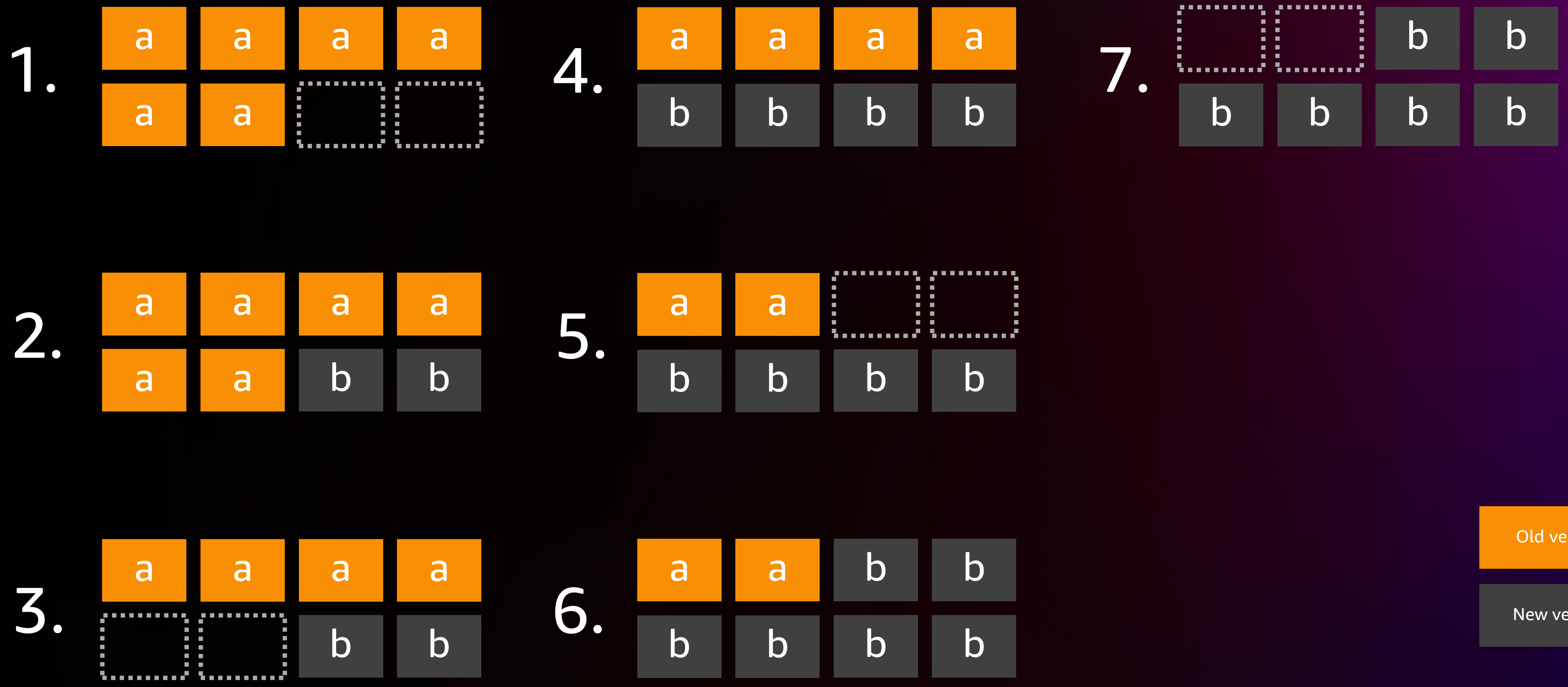
Amazon ECS orchestrated rolling deployment

Incrementally increasing percentage of requests is serviced by the newly deployed application version until 100% is reached

Blue-green and canary deployment with AWS CodeDeploy

Deploy new application version, shift requests to the new infrastructure incrementally, but retain the old infrastructure for rollback purposes

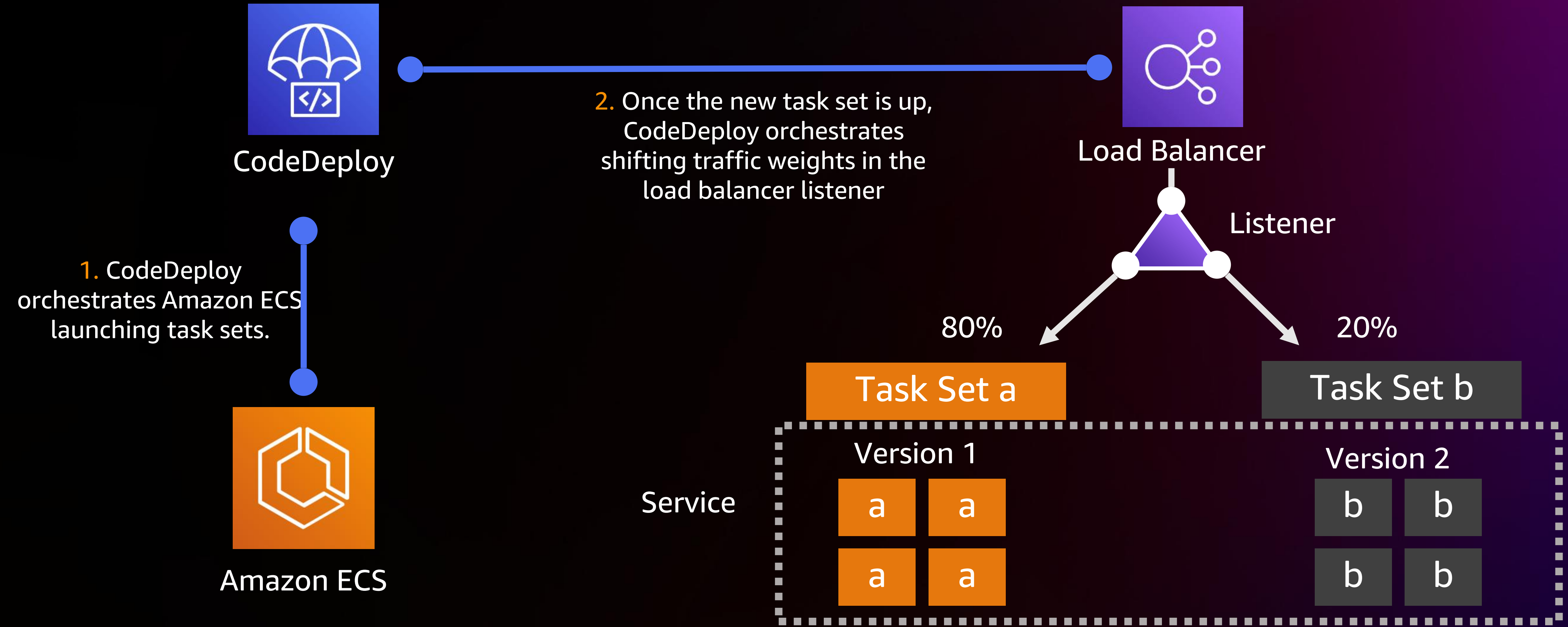
Amazon ECS rolling deployment iteratively creates new tasks and removes old



🟧 Old version

⬛ New version

CodeDeploy orchestrates blue/green deployment with Amazon ECS and ELB



CodeDeploy
2. Once the new task set is up, CodeDeploy orchestrates shifting traffic weights in the load balancer listener
Load Balancer
Listener
1. CodeDeploy orchestrates Amazon ECS launching task sets.
80%
20%
Task Set a
Task Set b
Version 1
Version 2
Service
a
a
a
a
b
b
b
b
Amazon ECS

There are many ways to deploy to Amazon ECS

Console

API

Copilot

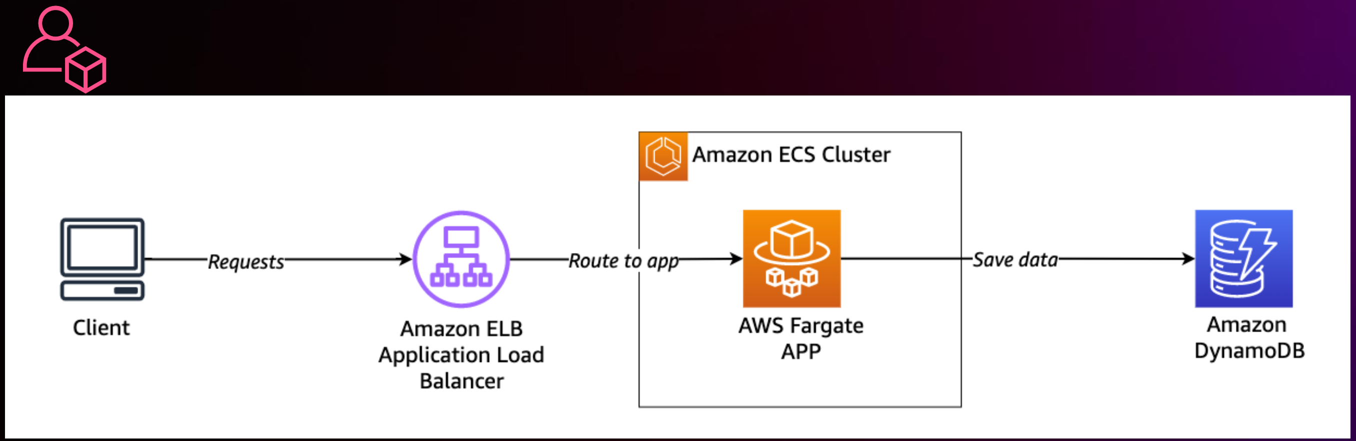
CloudFormation

CDK

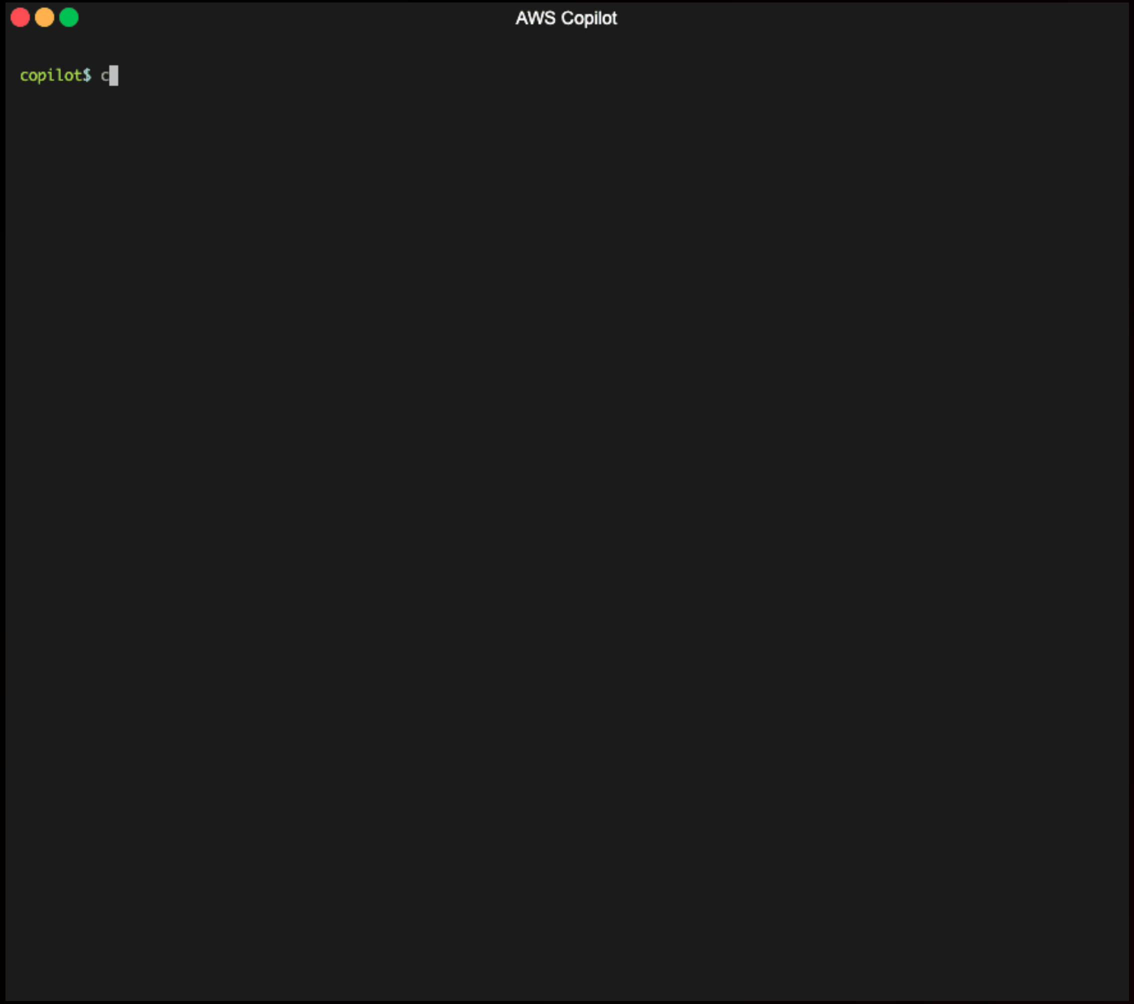
Terraform

Github actions

And many more . . .



Application-first interfaces to deploy with Amazon ECS



```
copilot$ c
```

1. Amazon ECS Blueprints

Terraform templates to deploy apps in minutes

2. Amazon ECS CDK extensions

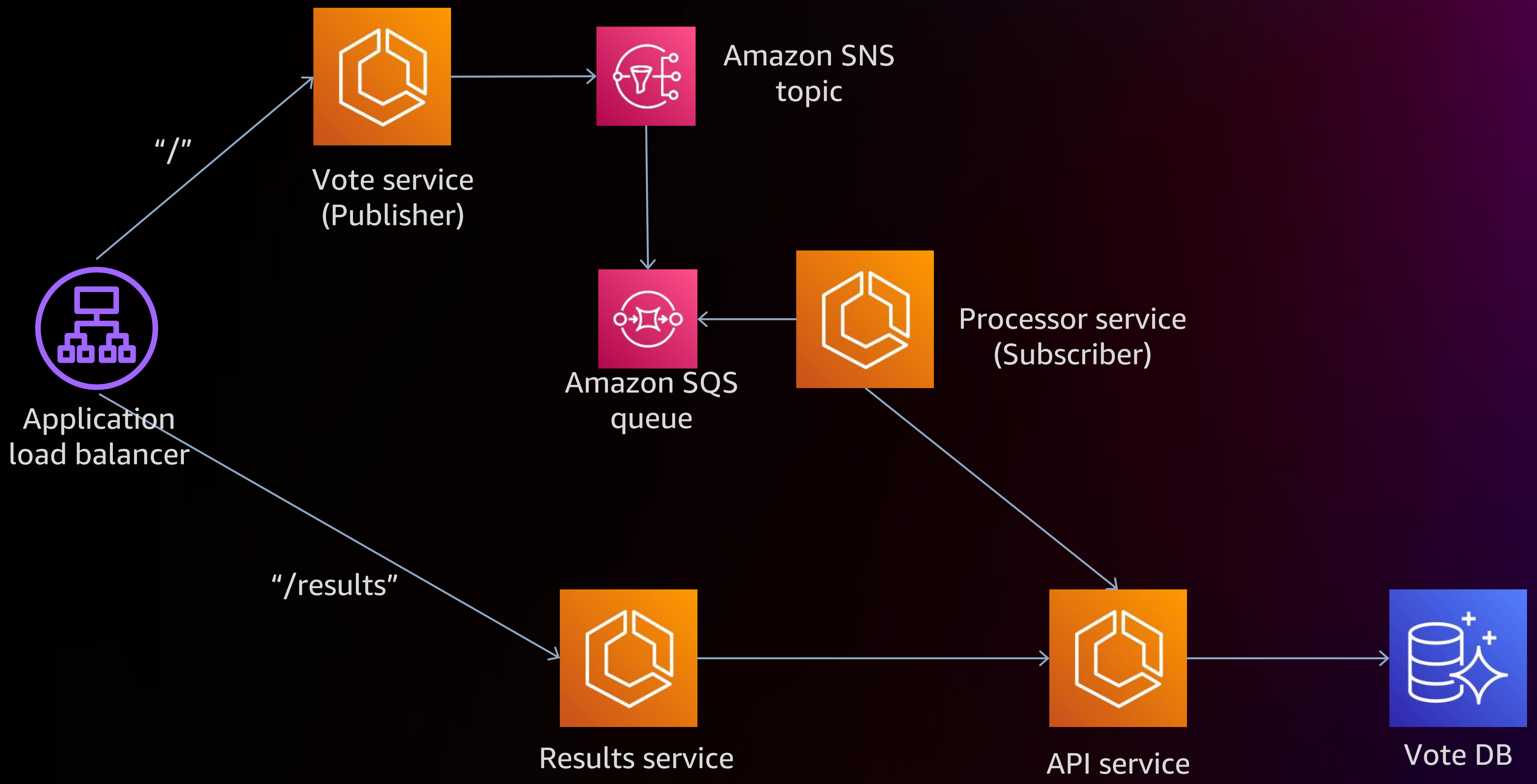
Program architecture patterns in your choice language

3. AWS Copilot CLI

Configuration-ready experience for common cloud architectures. Preconfigured continuous delivery pipeline with auto scaling, metrics, alarms, and logs.

Let’s see an Amazon ECS deployment in action!

First, let’s look at the application architecture



Deployment best practices

3 key dimensions for deployments

Continuous deployment

Safety

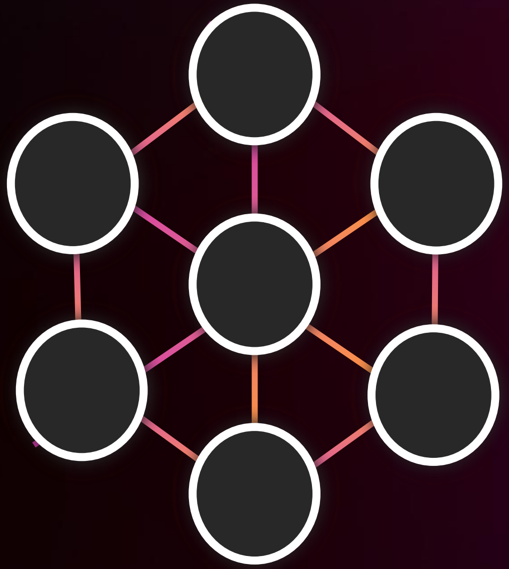
Speed

Best practices: Continuous deployment

When the impact of change is small, release velocity can increase

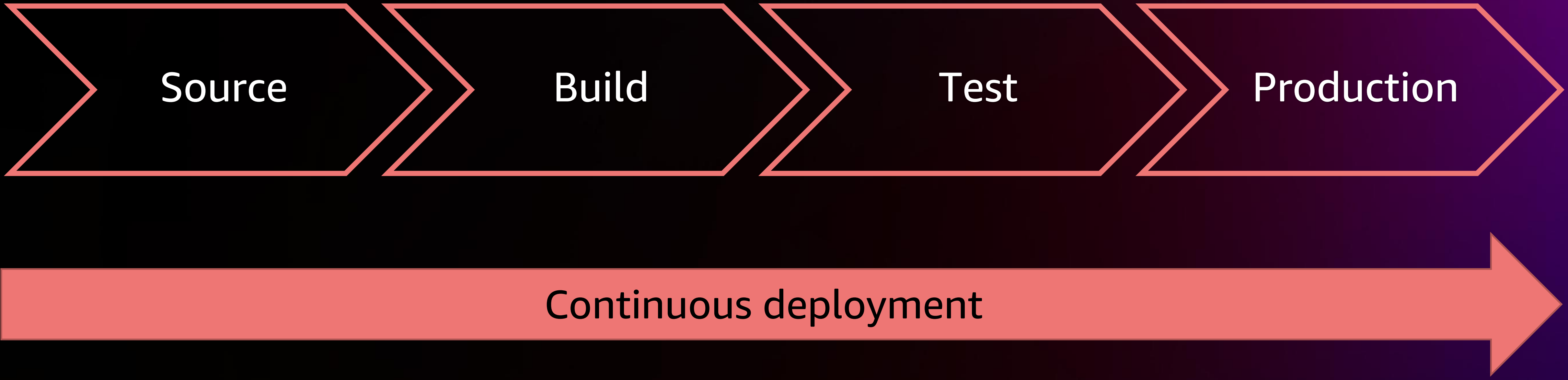
Monolith

Does everything



Microservices

Does one thing


Source
Build
Test
Production
Continuous deployment

Safe continuous deployments at Amazon

Test

Code

- App Code
- Configuration
- Infra-as-code
- Dependency libraries
- Static Assets

Build

- Compile code
- Unit tests
- Code coverage checks
- Static analysis
- Artifact

Beta

- Health checks
- Integration tests

Gamma

- Health checks
- Integration tests
- Automatic rollback
- Bake time

Canary tests
(run continuously)

Production

Wave 1 One Box Region A

- Alarm Blocker
- Time window blocker
- Health Checks
- Automatic rollback
- Bake time

Wave 1 Prod Region A

- Alarm Blocker
- Time window blocker
- Rolling deployment
- Health Checks
- Automatic rollback
- Bake time
- Integration tests

Wave 2 One Box Region B,C

- Alarm Blocker
- Time window blocker
- Health Checks
- Automatic rollback
- Bake time

Wave 2 Prod Region B,C

- Alarm Blocker
- Time window blocker
- Rolling deployment
- Health Checks
- Automatic rollback
- Bake time
- Integration tests

…

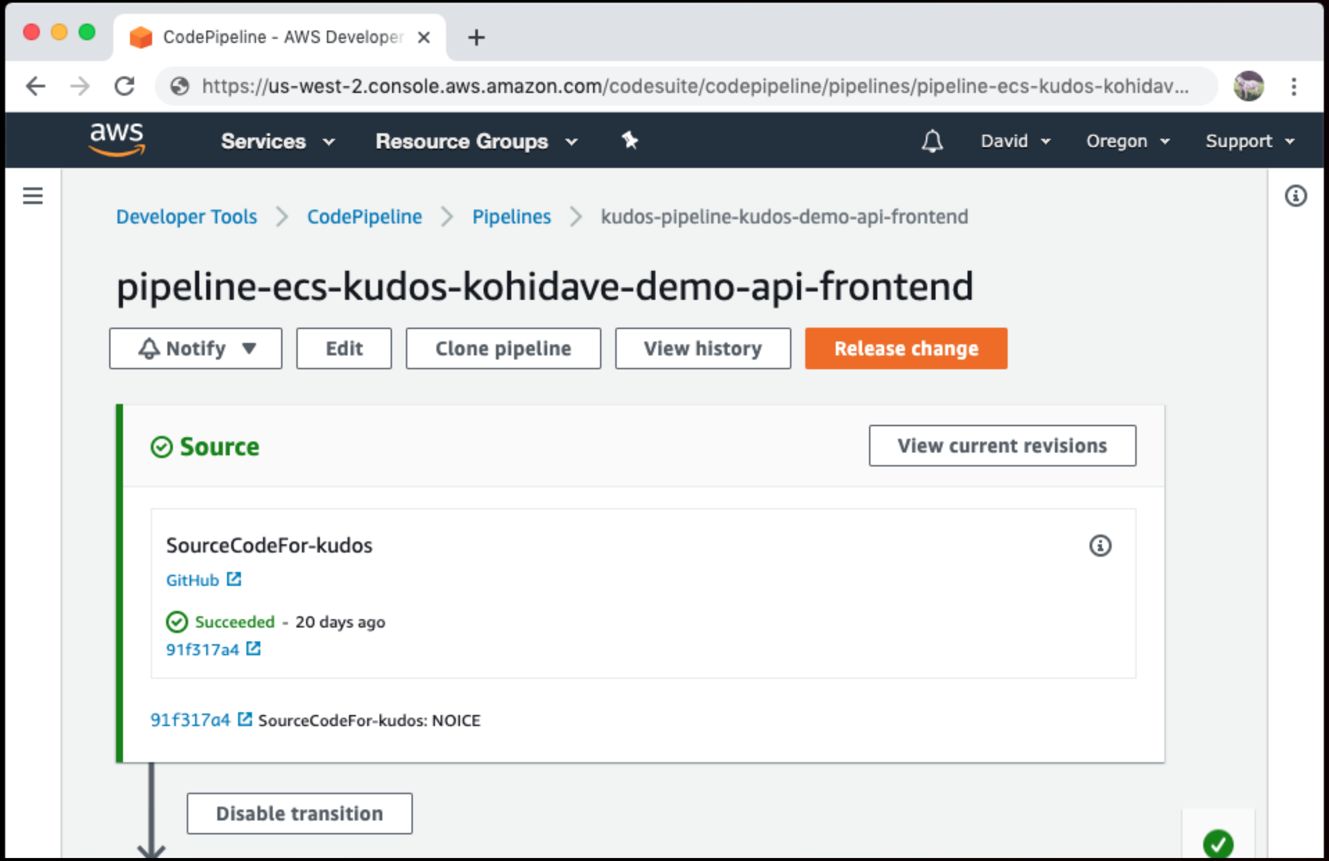
<---- Canary tests
(run continuously) ---->

Continuous deployment goals

Continuous deployment

1. Automatically deploy new changes to staging environments for testing
2. Deploy to production safely without affecting customers
3. Deliver to customers faster: Increase deployment frequency, and reduce change lead time and change failure rate

AWS Copilot simplifies continuous deployments


CodePipeline - AWS Developer
https://us-west-2.console.aws.amazon.com/codesuite/codepipeline/pipelines/pipeline-ecs-kudos-kohidav...
Services
Resource Groups
David
Oregon
Support
Developer Tools > CodePipeline > Pipelines > kudos-pipeline-kudos-demo-api-frontend
Notify
Edit
Clone pipeline
View history
Release change
View current revisions
Disable transition

pipeline-ecs-kudos-kohidave-demo-api-frontend

Source

SourceCodeFor-kudos
GitHub
Succeeded - 20 days ago
91f317a4

91f317a4 SourceCodeFor-kudos: NOICE

Uses AWS CodePipeline

Deploy to one or many environments

Can configure tests

Fully customize or use own CI/CD

Best practices: Safety

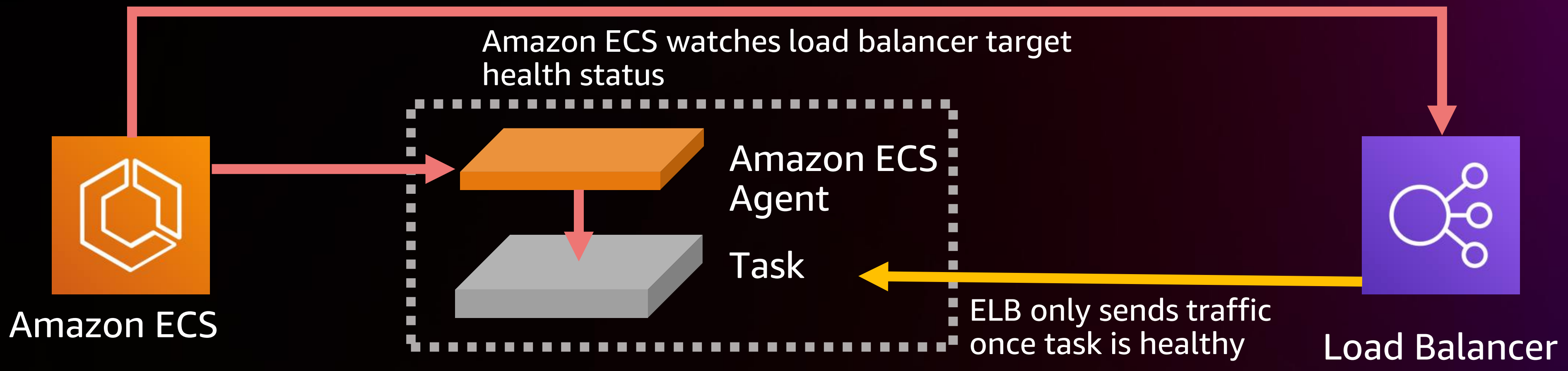
2 key goals for deployment safety

Prevent bad changes from reaching end users

Minimize downtime

- Handle shutdown gracefully
- Detect failures quickly and rollback

Amazon ECS + ELB health checks prevent bad changes from reaching production

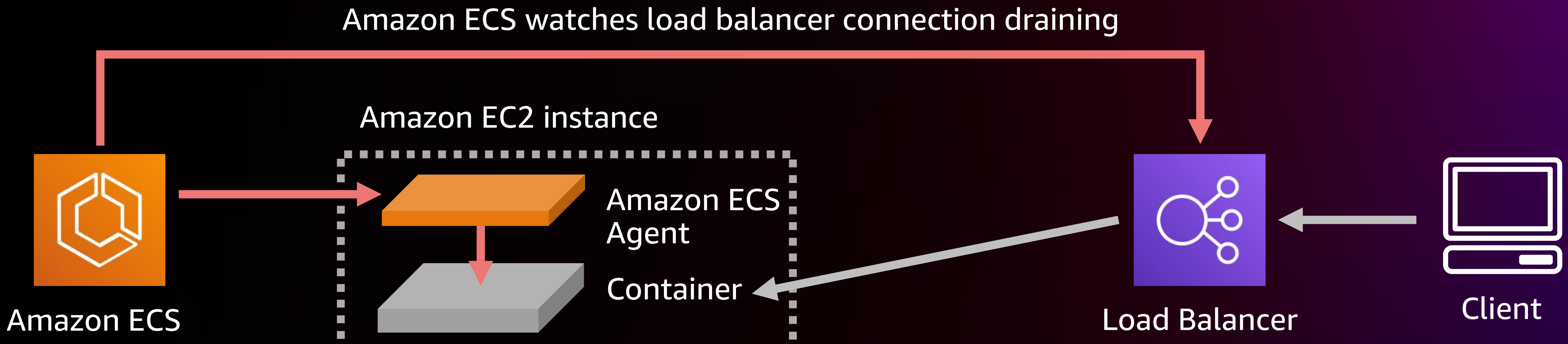


Amazon ECS watches load balancer target health status
Amazon ECS Agent
Task
Amazon ECS
ELB only sends traffic once task is healthy
Load Balancer

Tip: Add a dedicated health check endpoint for better control

Amazon ECS waits for load balancers to drain connections

deregistration_delay.timeout_seconds *(default = 300s)*

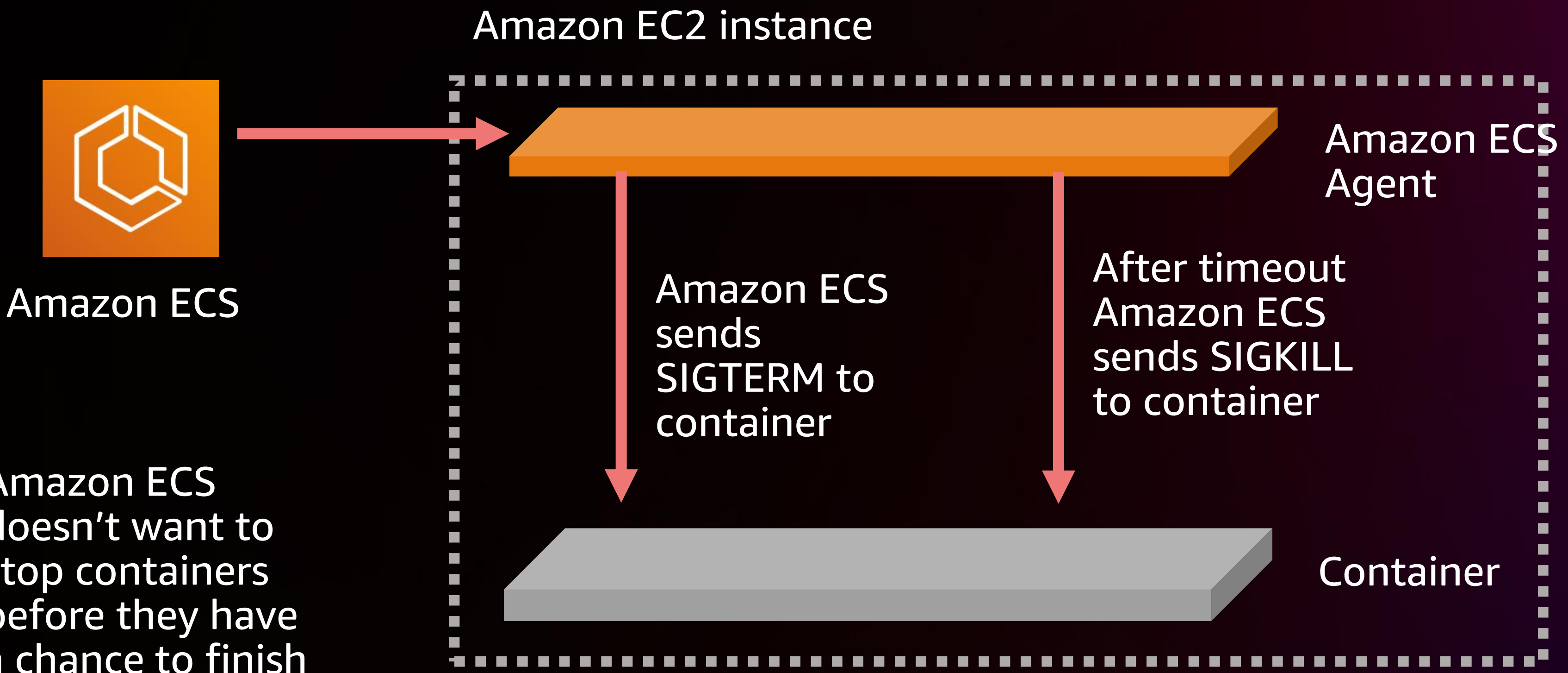


Amazon ECS doesn't send SIGTERM to container until load balancer reports that draining is completed.

Load balancer waits for client to close keep alive connection

Amazon ECS asks containers to exit nicely, before stopping them

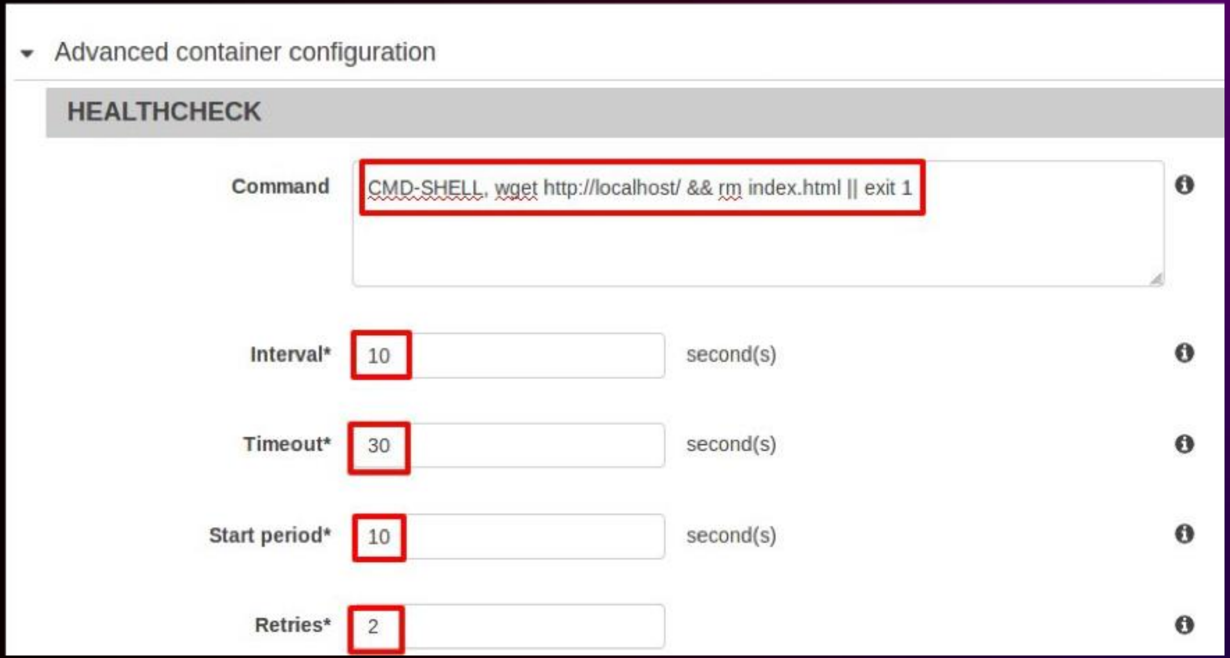
Amazon ECS_CONTAINER_STOP_TIMEOUT *(default = 30s)*


Amazon EC2 instance
Amazon ECS
Amazon ECS Agent
Amazon ECS sends SIGTERM to container
After timeout Amazon ECS sends SIGKILL to container
Container

Amazon ECS doesn't want to stop containers before they have a chance to finish in-progress work

Use container health checks for microservices that don’t use a load balancer

Amazon ECS container health check allows you to run user-configured commands in shell



Command	CMD-SHELL, wget http://localhost/ && rm index.html || exit 1	
Interval*	10	second(s)
Timeout*	30	second(s)
Start period*	10	second(s)
Retries*	2	

Other symptoms to detect bad deployments

Task Launch failures

Application errors

Abnormal CPU/ memory utilization

Amazon ECS deployment circuit breaker automatically monitors and rolls back bad deployments

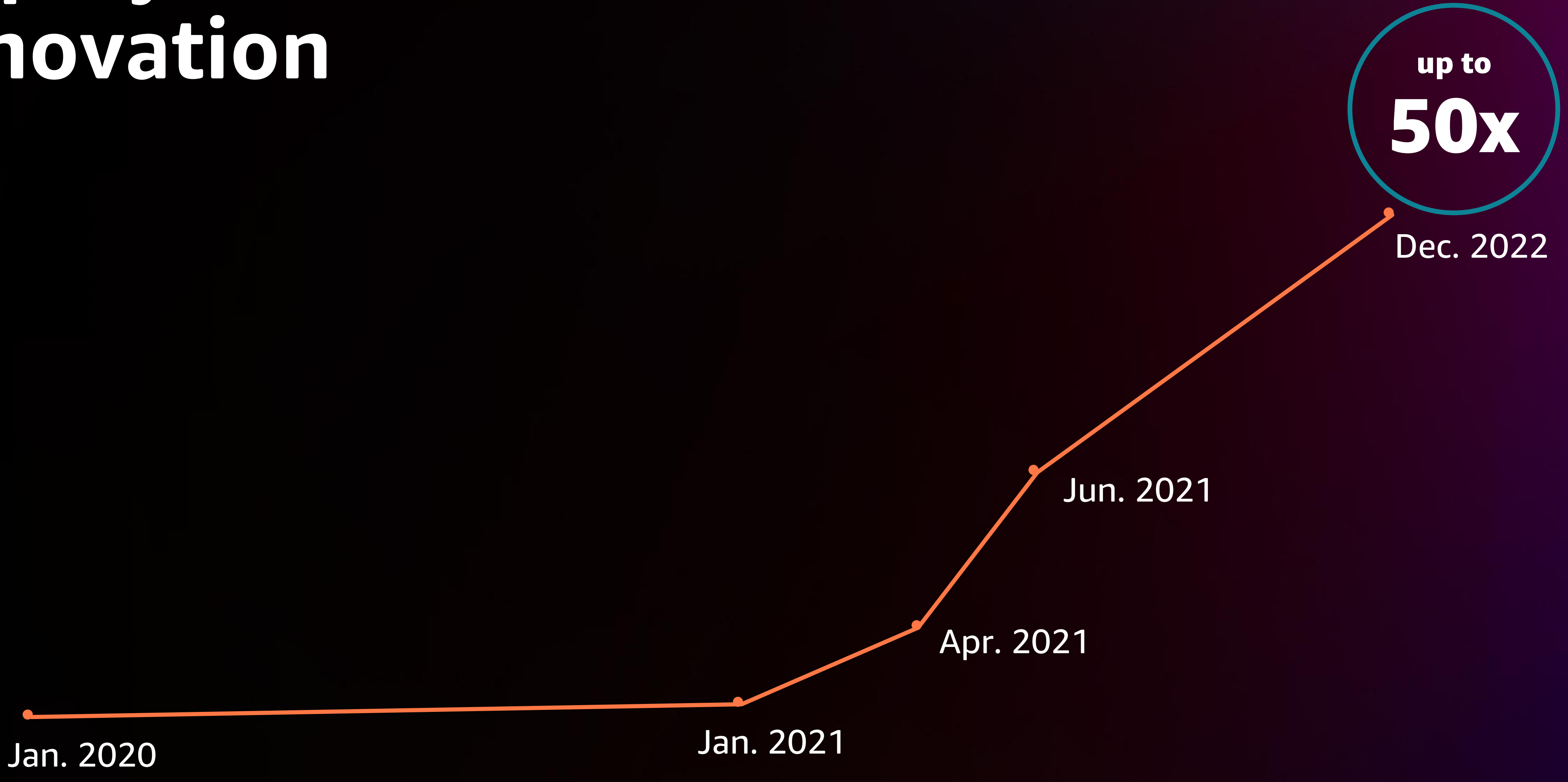
Amazon ECS deployment circuit breaker monitors task launch and health check failures

If failures exceed threshold,
Amazon ECS rolls back or stops deployment

One-click setup

Best practices: Speed

Amazon ECS is continuously making deployments faster to enable faster innovation



Amazon ECS deployment performance

Benchmarks using Amazon ECS services	Size	Duration	Rate
AWS Fargate on-demand capacity, no load balancer	1000 tasks	208 seconds	4.8 tasks/sec
AWS Fargate spot capacity, no load balancer	1000 tasks	353 seconds	2.8 tasks/sec
AWS Fargate 50/50 on-demand and spot capacity	1000 tasks	213 seconds	4.7 tasks/sec
AWS Fargate on-demand, no public IP address	1000 tasks	199 seconds	5 tasks/sec
AWS Fargate on-demand, load balanced service	1000 tasks	252 seconds	4 tasks/sec
EC2 capacity, capacity provider starts empty	1000 tasks	1118 seconds	0.8 tasks/sec
EC2 capacity, with all the EC2 instances up already	1000 tasks	270 seconds	3.7 tasks/sec
EC2 capacity, host networking instead of AWS VPC	1000 tasks	270 seconds	3.7 tasks/sec

Full blog: https://aws.amazon.com/blogs/containers/under-the-hood-amazon-elastic-container-service-and-aws-fargate-increase-task-launch-rates/

3 key controls to optimize deployment speed

Load balancer configurations

Amazon ECS service configuration

Task launch time

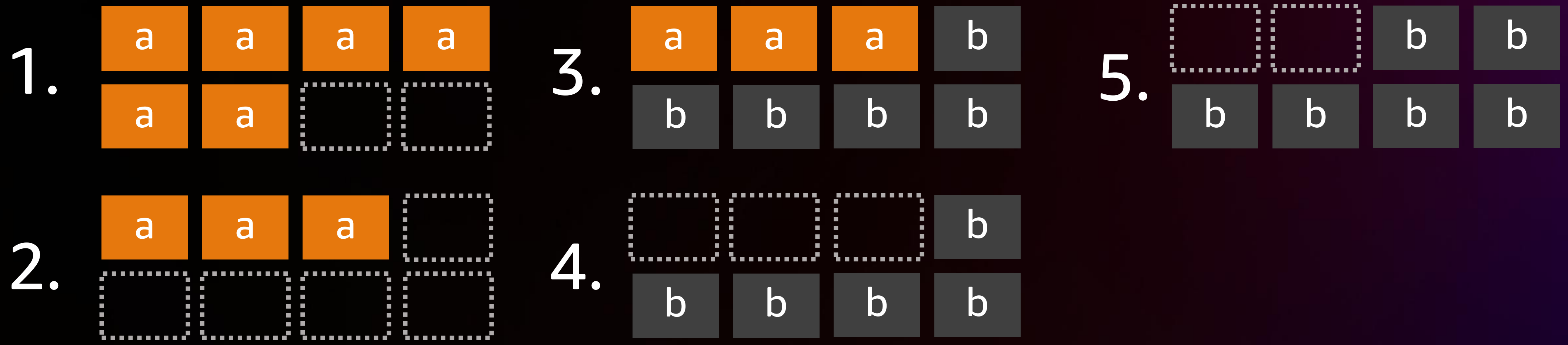
Important controls for Amazon ECS rolling deployment

minimumHealthyPercent – lower limit on the number of tasks in a service that must remain in the RUNNING

maximumPercent – upper limit on the number of tasks in a service that are allowed in the RUNNING or PENDING

These impact resources and time taken to roll out.

E.g. minimumHealthyPercent – 50% ; maximumHealthyPercent – 200%



Availability vs. speed

Availability ⟷ Speed

For production you want to maintain high availability and handle load

minimumHealthyPercent – 100%
maximumPercent – 200%

For a development environment optimize for speed of rolling a deploy

minimumHealthyPercent – 50%
maximumPercent – 200%

Load Balancer configurations impact deployment speed too

Registering new tasks

HealthCheckIntervalSeconds *(default 30s for ALB with Target as IP)*

HealthyThresholdCount *(default 5 for ALB)*

Terminating old tasks

deregistration_delay.timeout_seconds *(default = 300s)*

You can mutate these configurations to increase deployment speed

Speeding up task launches

Smaller container images

Use cached images on Amazon EC2

Amazon ECS_IMAGE_PULL_BEHAVIOR – once or prefer-cached

Compressed container images (zstd) for faster startup on Fargate **(NEW)**

Takeaways

Best practices for deployments

Configure dedicated health check endpoint for your load balanced Amazon ECS services (container health check if service doesn’t use ELB)

Use circuit breaker as a default safeguard

Mutate Amazon ECS service and ELB configurations to manage deployment speed

CDK/Copilot/Blueprints for application-focused constructs

Pipelines for automation

Keep images small and/or use caching (EC2)/compression (Fargate)

Resources

Best practices to speed up Amazon ECS deployments: https://bit.ly/3VrjEoG

Automating hands off deployments: https://go.aws/3EFwB7H

Graceful shutdowns in Amazon ECS: https://go.aws/3gFK6fz

Faster startup times on Fargate with zstd compression: https://go.aws/3F8XRwW

Sample application: https://bit.ly/3gIpUd6

Thank you!

Uttara Sridhar
Uttara@amazon.com

Vibhav Agarwal
agvibhav@amazon.com
@VibhavAgarwal22

Please complete the session survey in the **mobile app**