AWS re:Invent

NOV. 27 – DEC. 1, 2023 | LAS VEGAS, NV

GAM306

Modernization of Nintendo eShop: Microservice and platform engineering

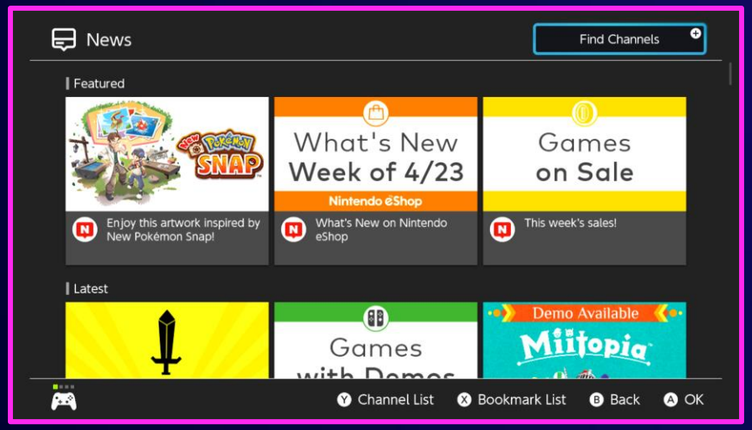
Taiji Iwai

Senior Solutions Architect
Amazon Web Services

Agenda

1. Nintendo eShop
2. Microservice
3. Platform engineering
4. E-commerce API Platform

What we do at Nintendo Systems



Game server

Online matching, leader board, messaging, etc.

Game news

Promotion of the contents

Nintendo eShop

Browse and purchase digital software

and more...

Nintendo eShop

What Nintendo eShop provides

- Browse catalog of game titles
  (product page, price, image, video clips, etc.)
- Various payment methods
  (credit card, point-of-sales cards, PayPal, etc.)
- Purchase/redeem game titles
- Distribute digital contents
- Digital rights management (DRM)

and more...

Basic of Nintendo eShop systems

Web-based application

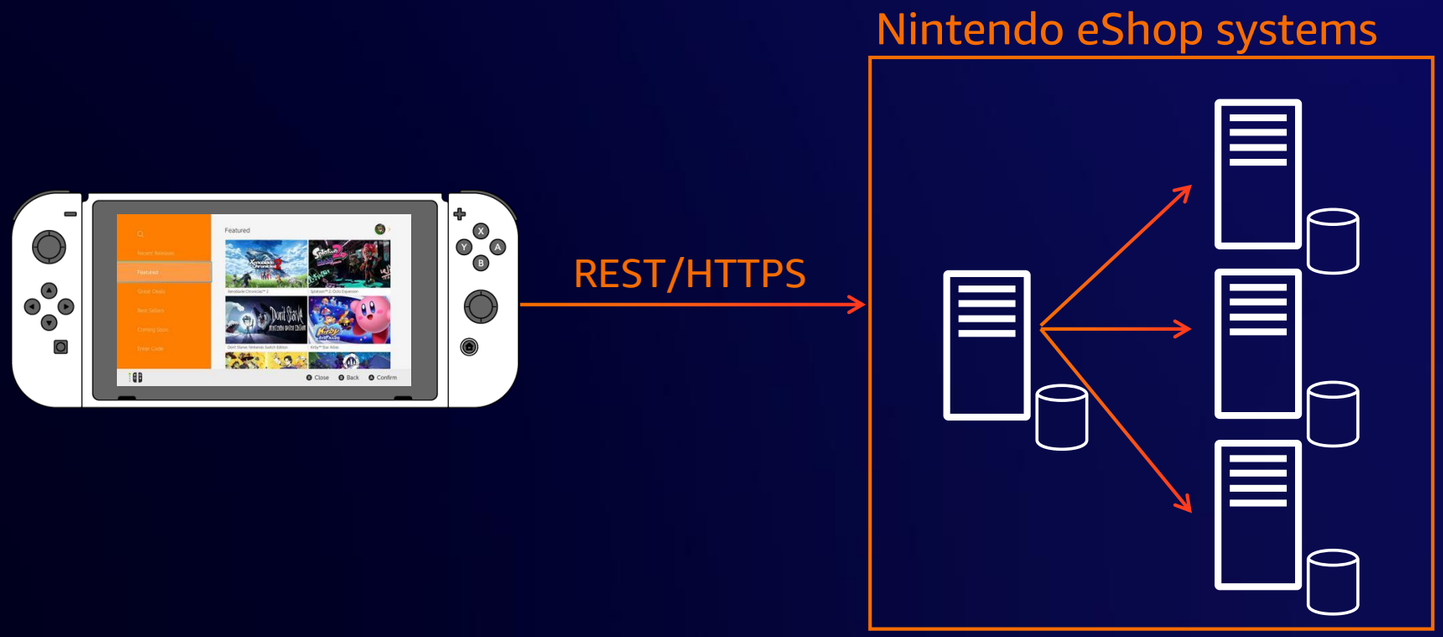
- Frontend is a single-page application built on React
- Runs on built-in browser

Global and 24/365

- 40+ countries
- 20+ currencies
- 120M+ Nintendo Switch units

Amount of data

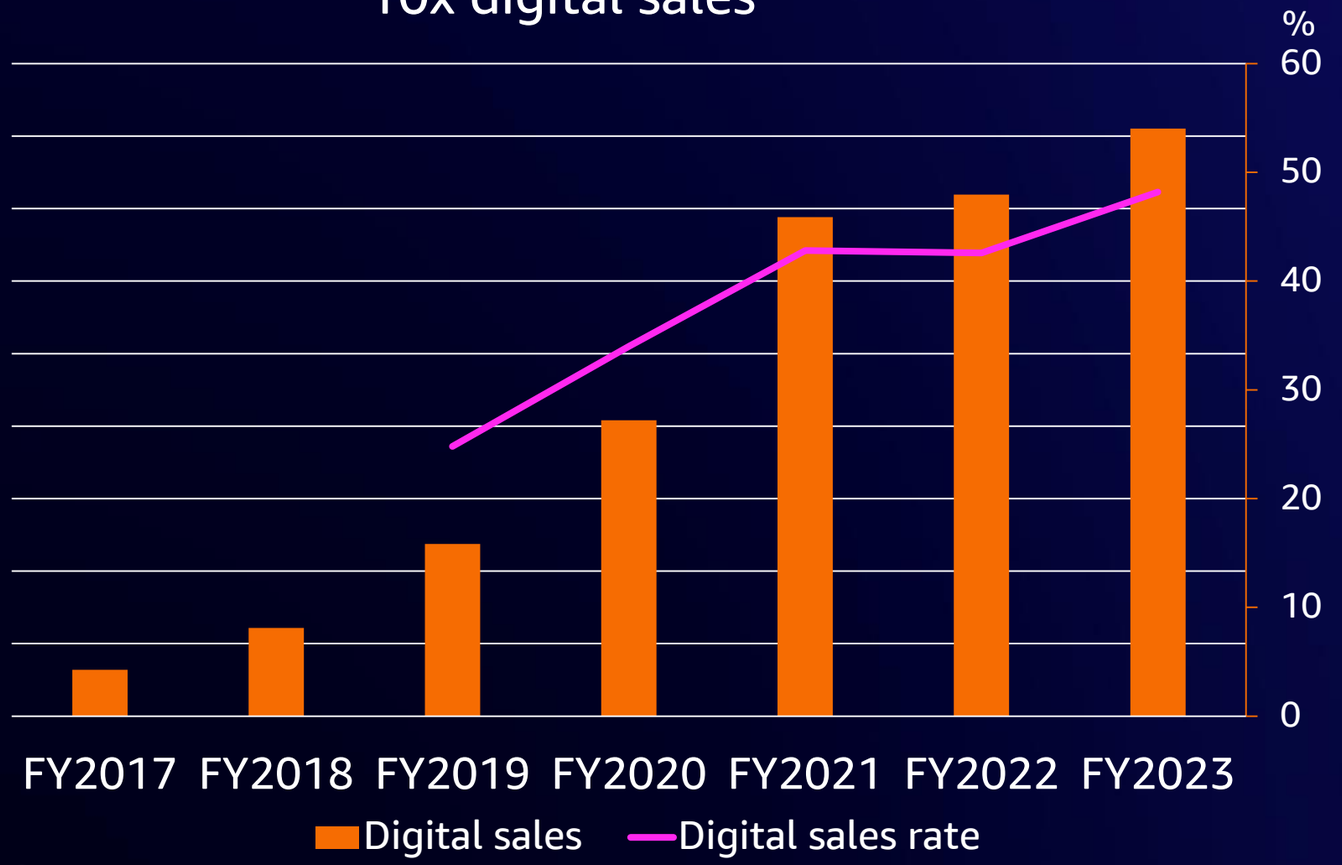
- 500+ Tables
- 100M+ account records
- 3B+ transaction records



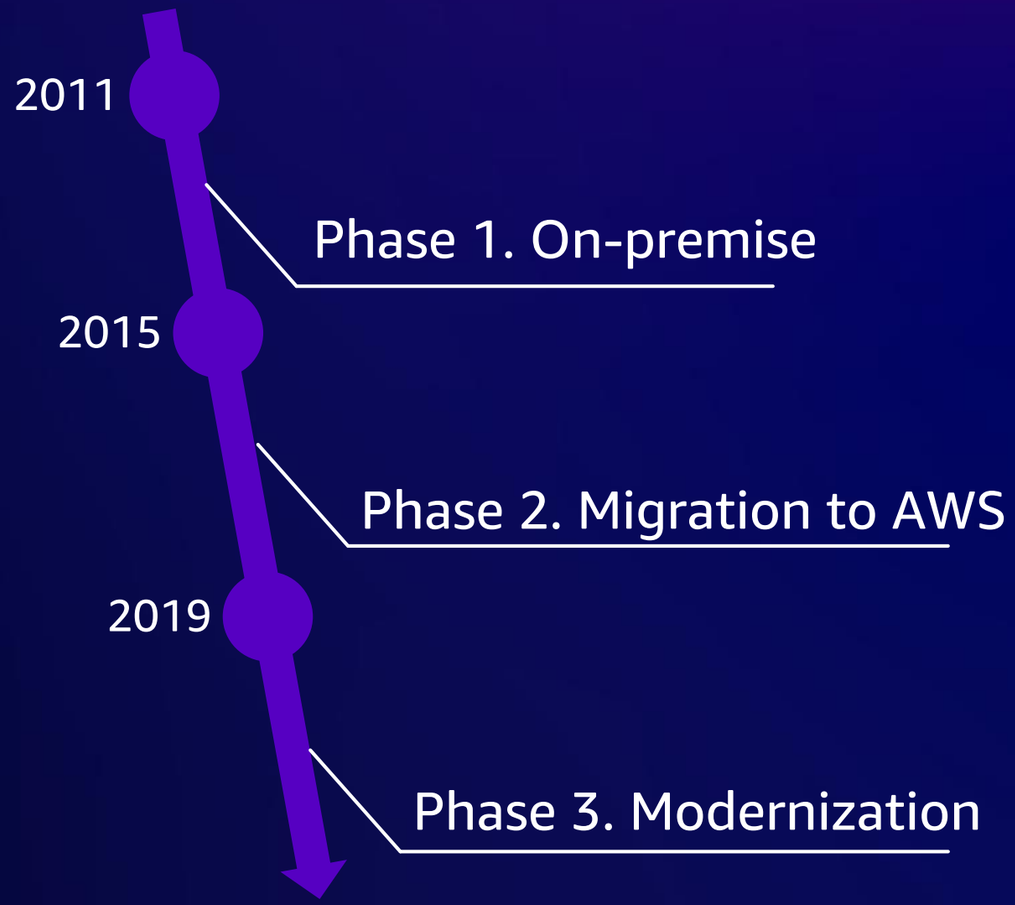
Transition of Nintendo eShop systems

Transition of digital sales over years

10+ years in service
10x digital sales



Refactoring process



Phase 1. On-premise

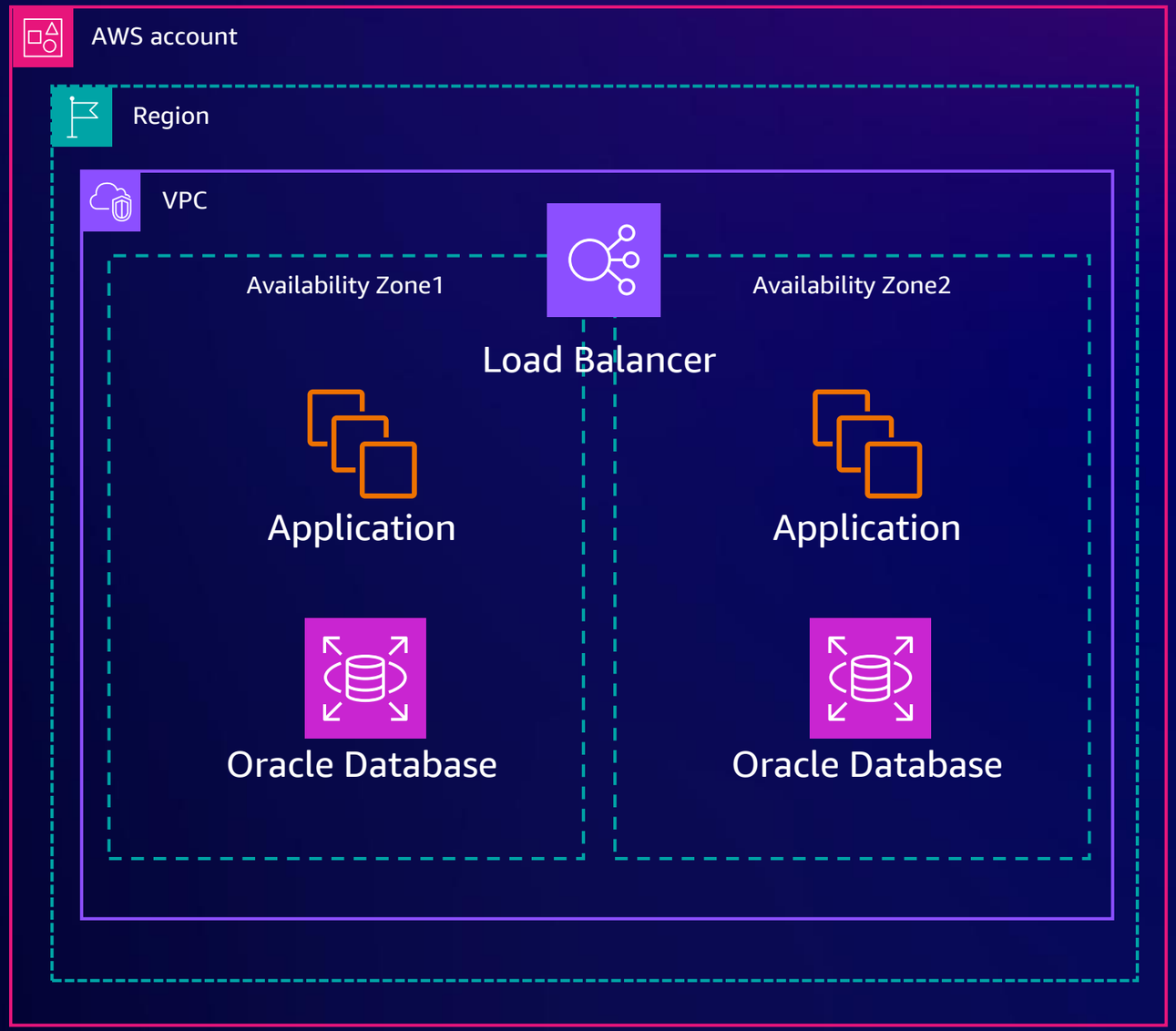
2011 TO 2015

- All the components deployed to on-premise infrastructure
- Monolithic application and database built on Java/Tomcat
- Growing number of functional/non-functional requests

Phase 2. Migration to AWS

2015 TO 2018

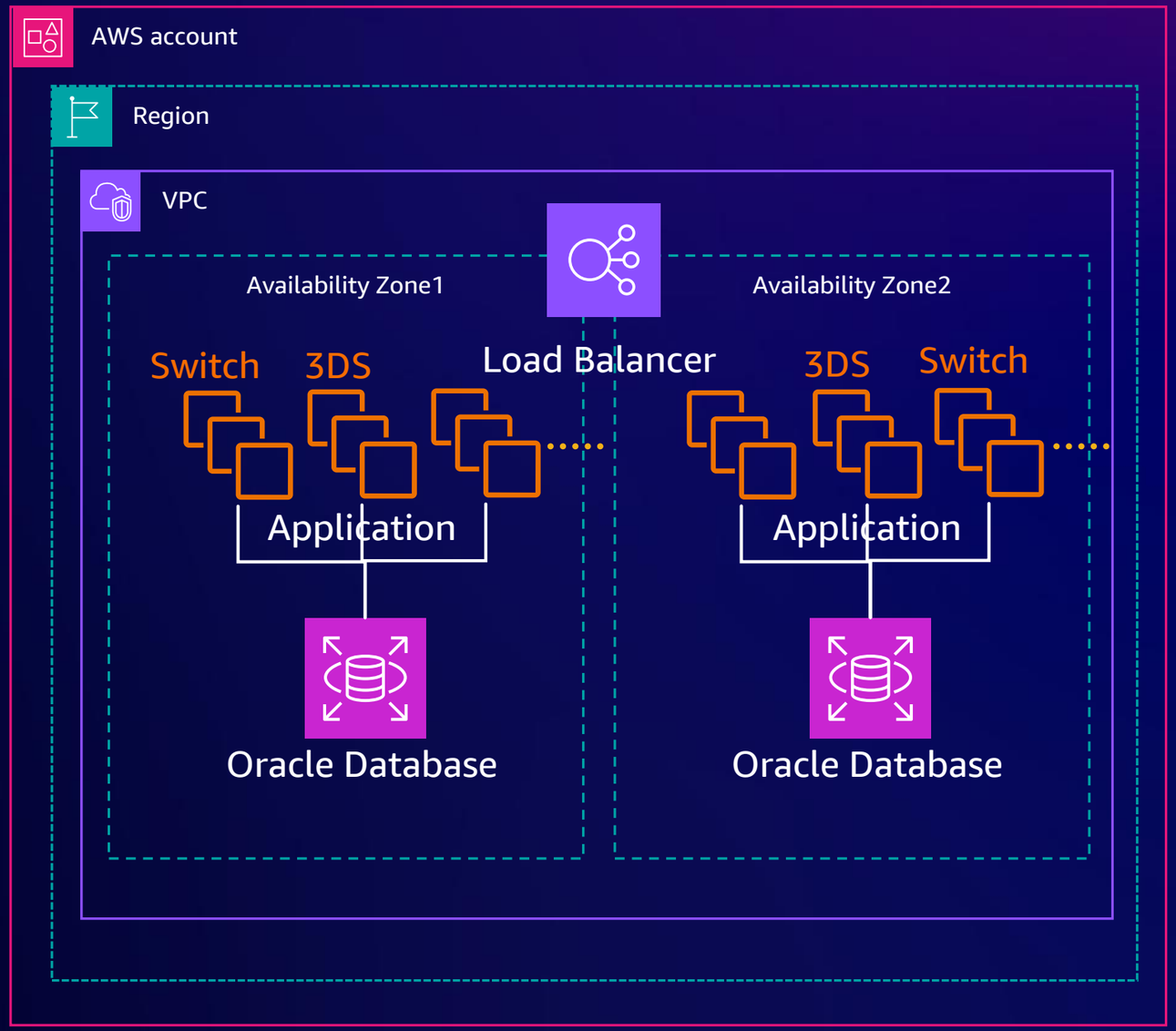
- Migrated eShop servers to AWS
- Single AWS account with single VPC
- Agility and scalability from AWS



Phase 2. Migration to AWS

2015 TO 2018

- Migrated eShop servers to AWS
- Single AWS account with single VPC
- Agility and scalability from AWS
- Numbers of applications were then added to the VPC as more eShop functionalities are implemented



Phase 3. Modernization

2019 -

Core application servers with large and complex data and business logic

- Productivity does not scale due to the complexity of the application and the database
- Low testability of the application

Distributed monolith with multiple application servers accessing a single, large database

- Scalability limited by the size of RDS
- Major feature release deliverable every three months

Modernize the system to achieve agility and long-term maintainability

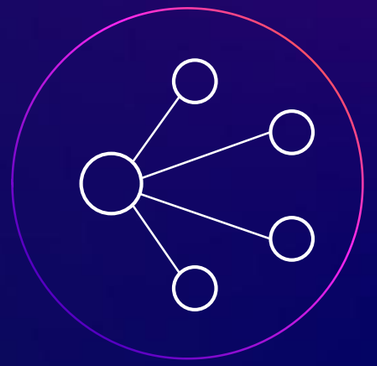
Modernizing Nintendo eShop systems

Microservice

Split the application and the database to achieve long-term maintainability and agility

Platform engineering

Bring self-service capabilities and mitigate the cognitive load of developers and operators



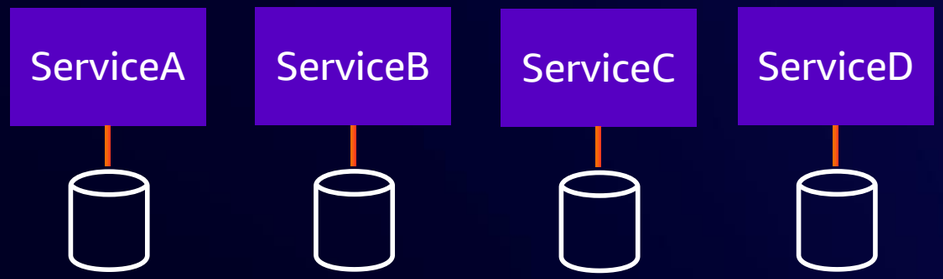
E-commerce API platform

Provide and facilitate the use of e-commerce functionalities (APIs) from various services

Microservice

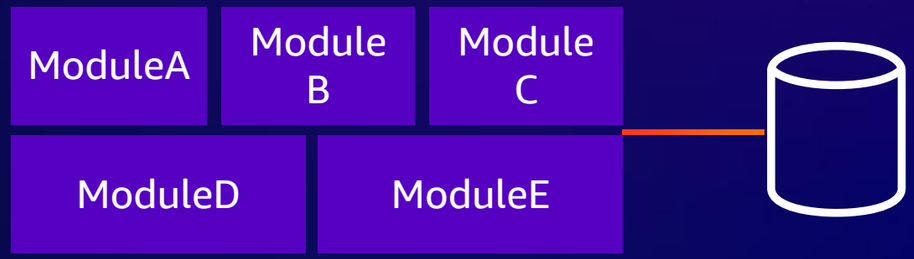
Microservice vs. modular monolith

Microservice



- Each service can be developed and deployed independently
- Each service can utilize the technologies that best suit the service
- Difficult to define a boundary to make the services loosely coupled

Modular monolith



- Each module can be developed independently (requires effort)
- Lower complexity than microservice
- Requires big effort when modernizing architecture/technology
- Limits the technology choices

Why we chose microservice

Long term maintainability

- 20+ years
- Embrace new technologies

Improve productivity

- Make the productivity scale as the team grows
- Minimize the required domain knowledge when joining the team

Shorten the release cycle

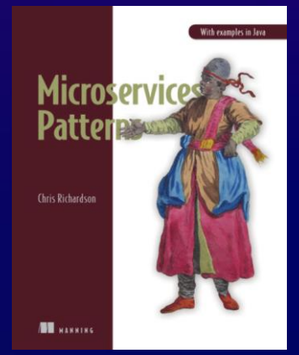
- Service can be deployed independently

Cost-effective scalability

- Necessary portion of the system can be scaled

“Microservice is not a silver bullet”

Microservices patterns

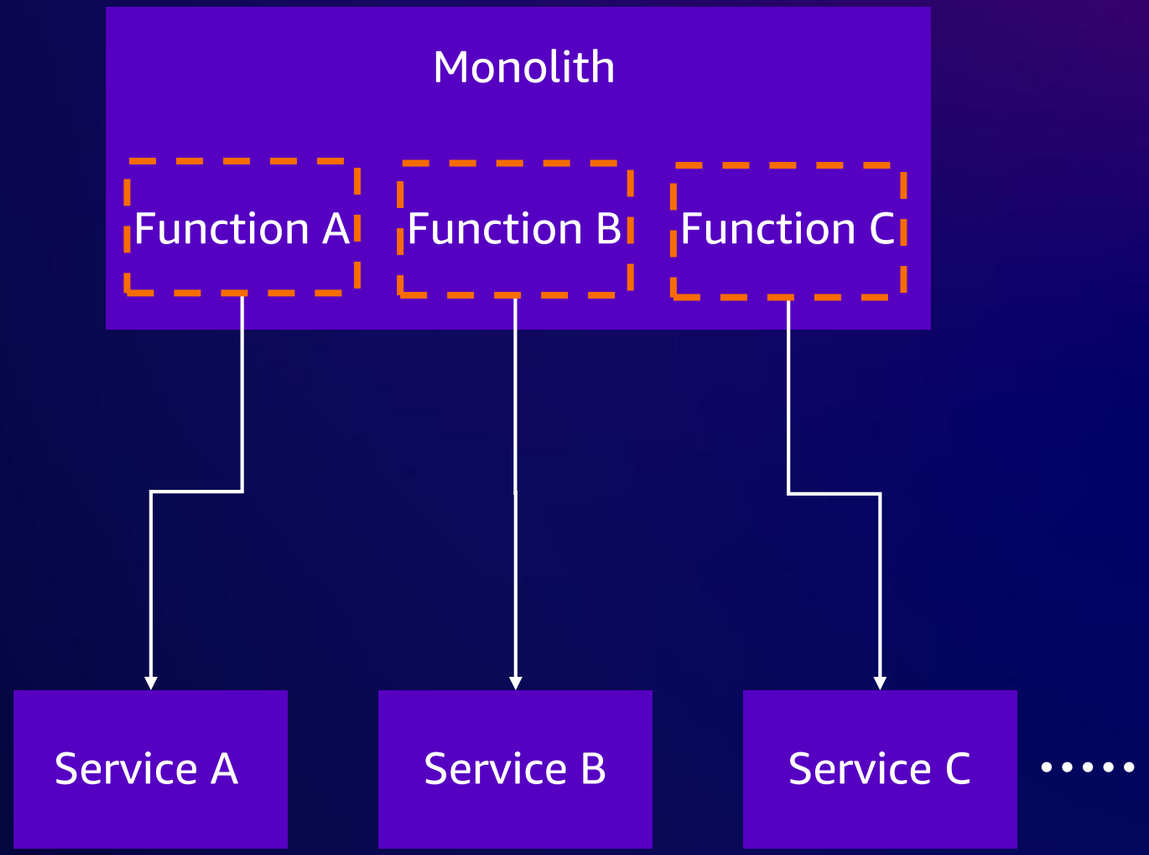


Platform engineering to mitigate the pain points of microservice

How do we define microservice boundaries?

Single responsibility principle

- Nintendo eShop balance, transaction histories, …
- Difficult to estimate the total number of services
- Only few services were easily decoupled



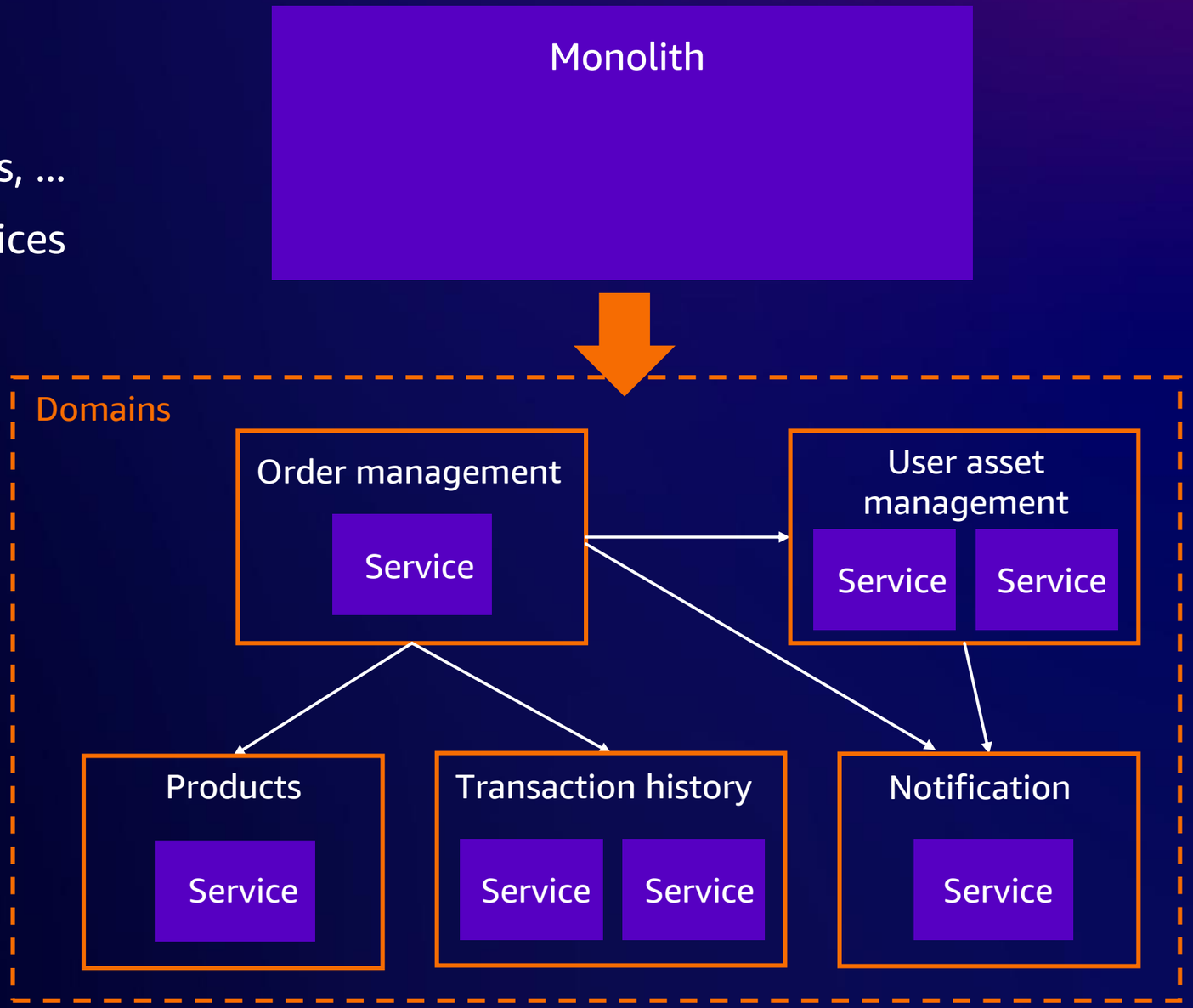
How do we define microservice boundaries?

Single responsibility principle

- Nintendo eShop balance, transaction histories, …
- Difficult to estimate the total number of services
- Only few services were easily decoupled

Domain Driven Design principle

- Divide data and functionalities into domains
- Decompose domain into acceptable number of microservices



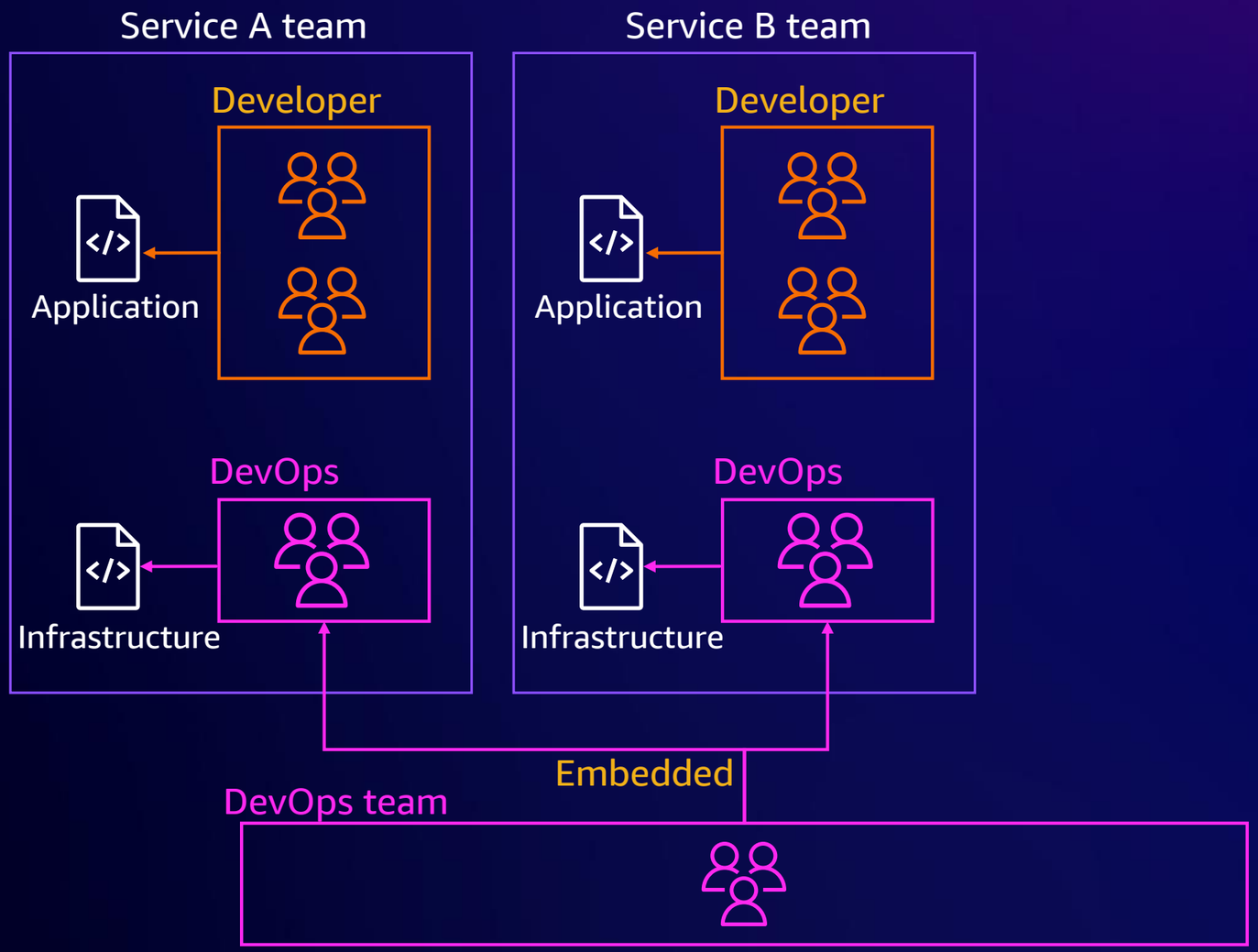
Microservice team

Developer

- Application development

Devops

- Infrastructure development
- Deployment



Microservice team

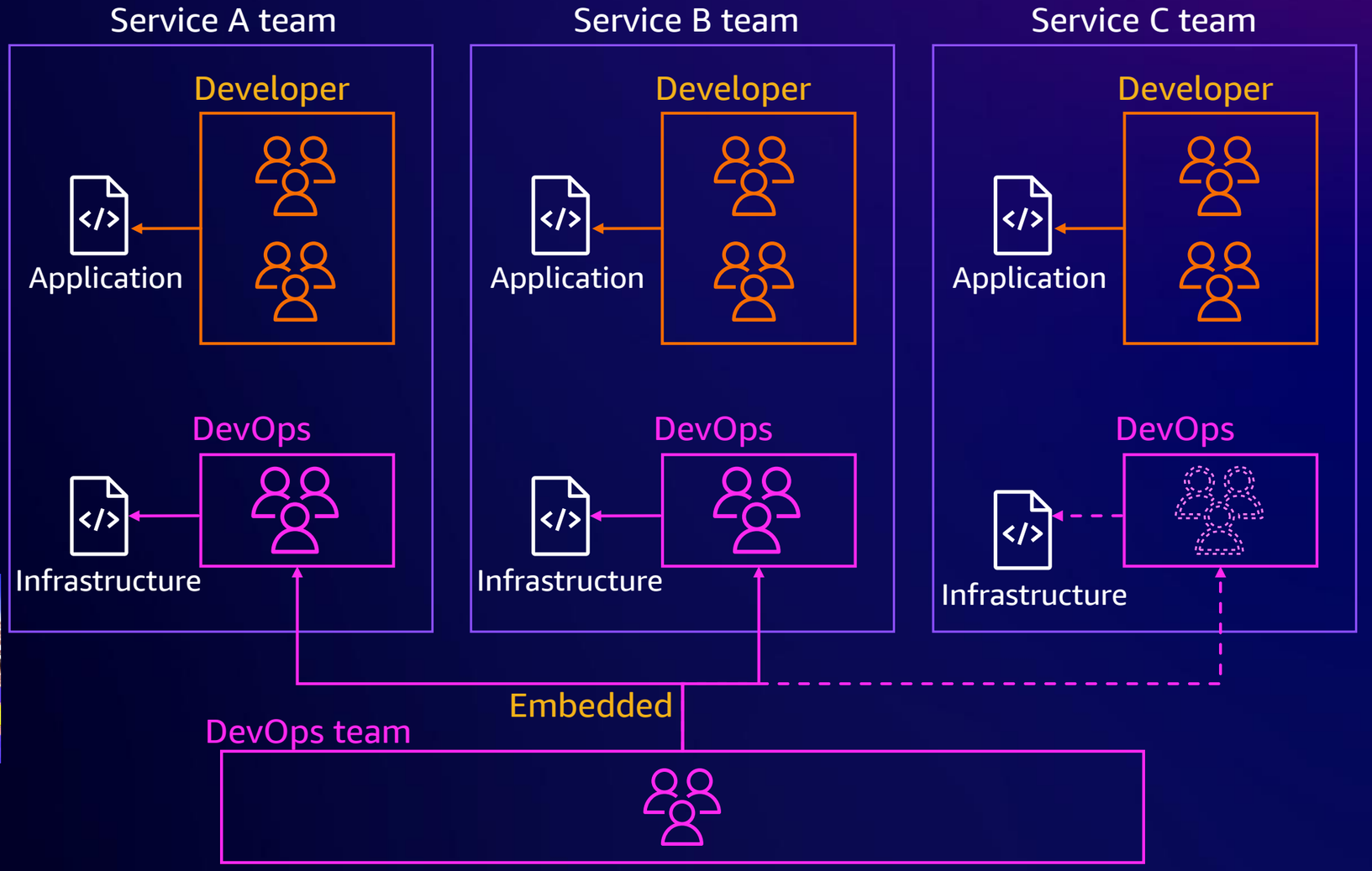
Developer

- Application development

Devops

- Infrastructure development
- Deployment

Nintendo Direct



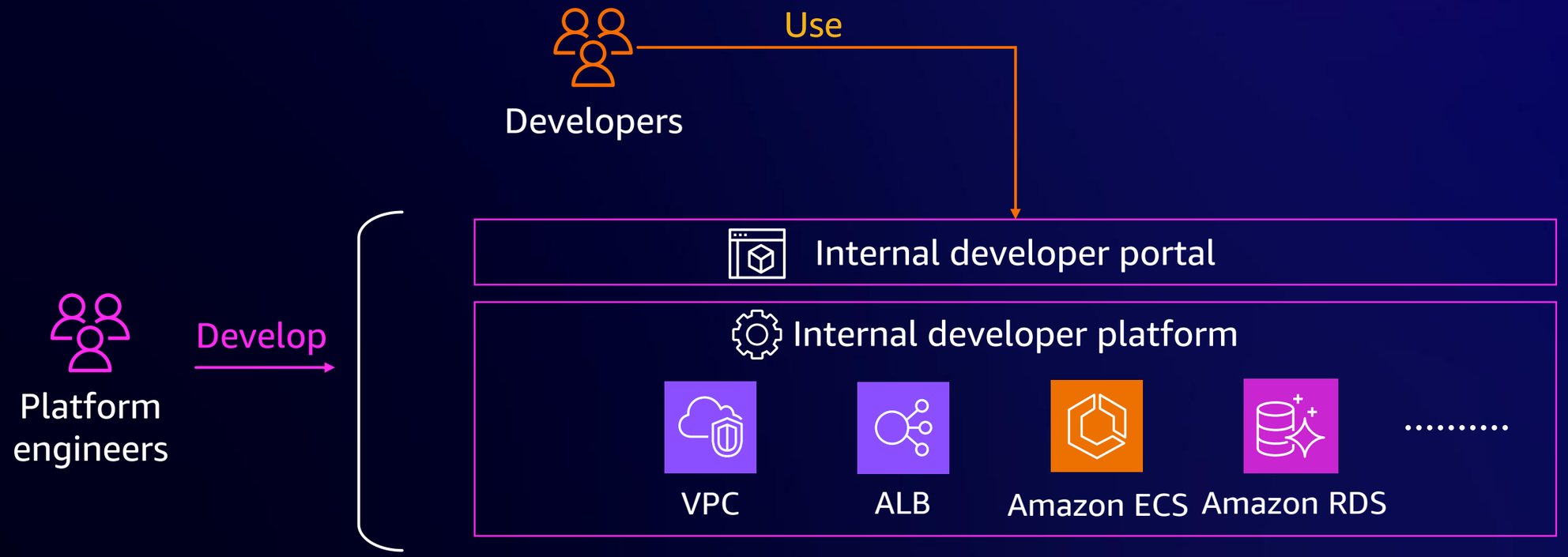
Platform engineering

What is platform engineering?

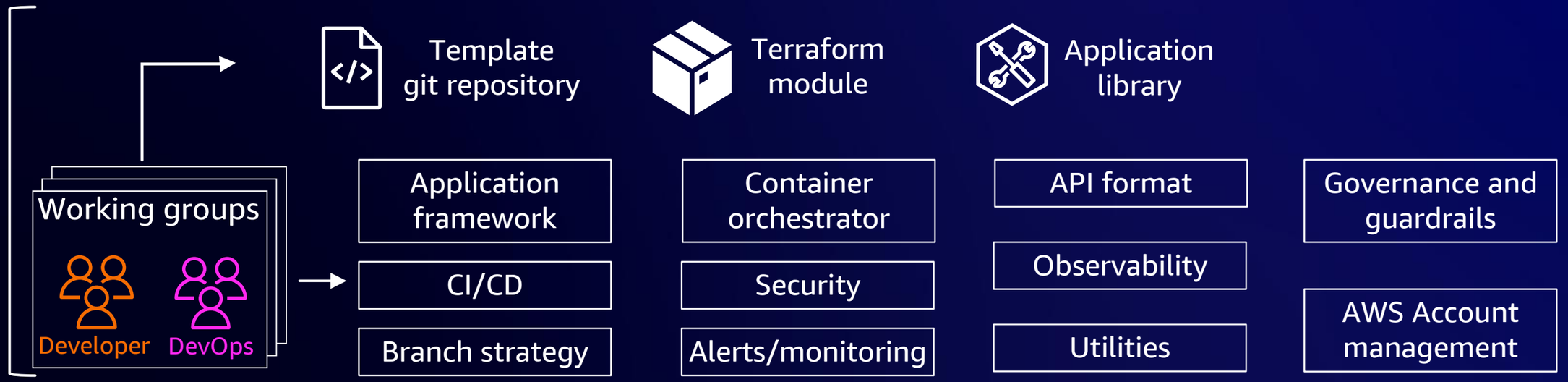
Platform Engineering

https://platformengineering.org/

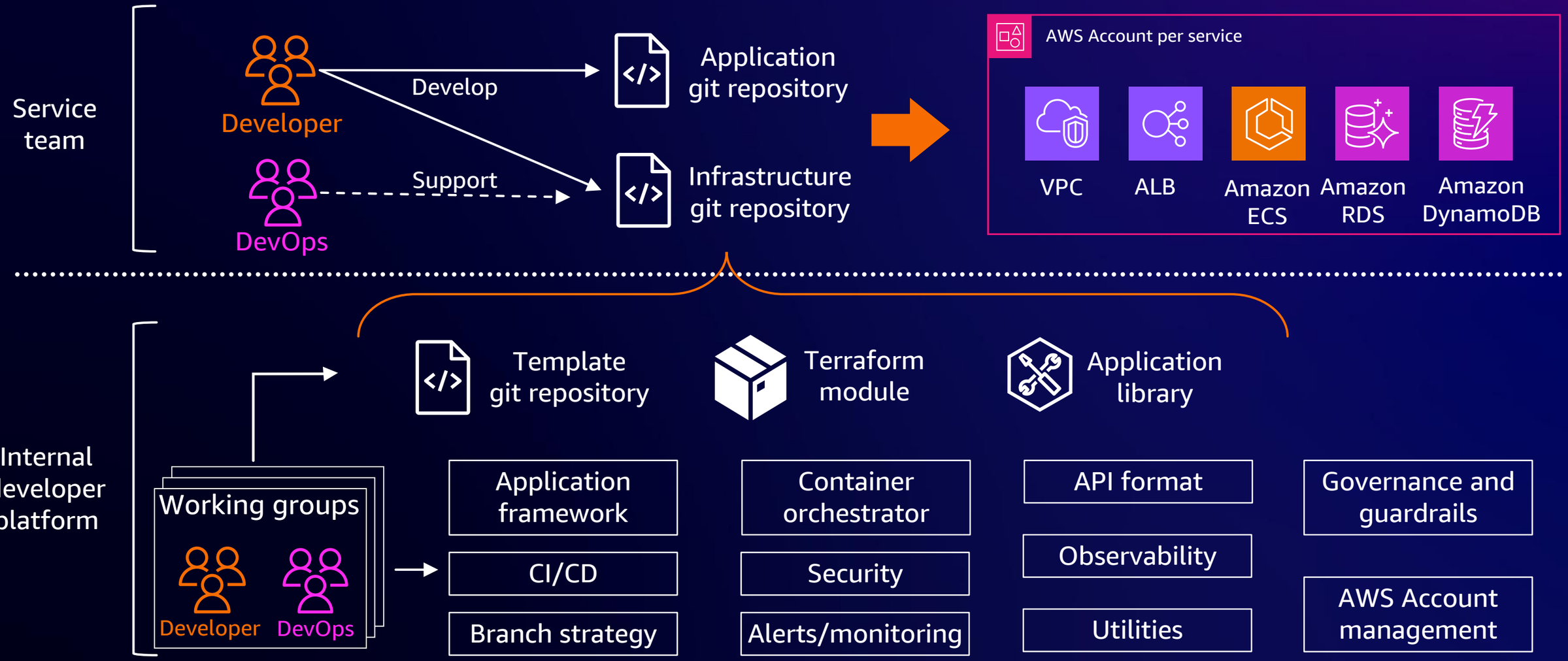
Platform engineering is the discipline of designing and building toolchains and workflows that enable self-service capabilities for software engineering organizations in the cloud-native era. Platform engineers provide an integrated product most often referred to as an internal developer platform covering the operational necessities of the entire life cycle of an application.



Our approach to platform engineering



Our approach to platform engineering

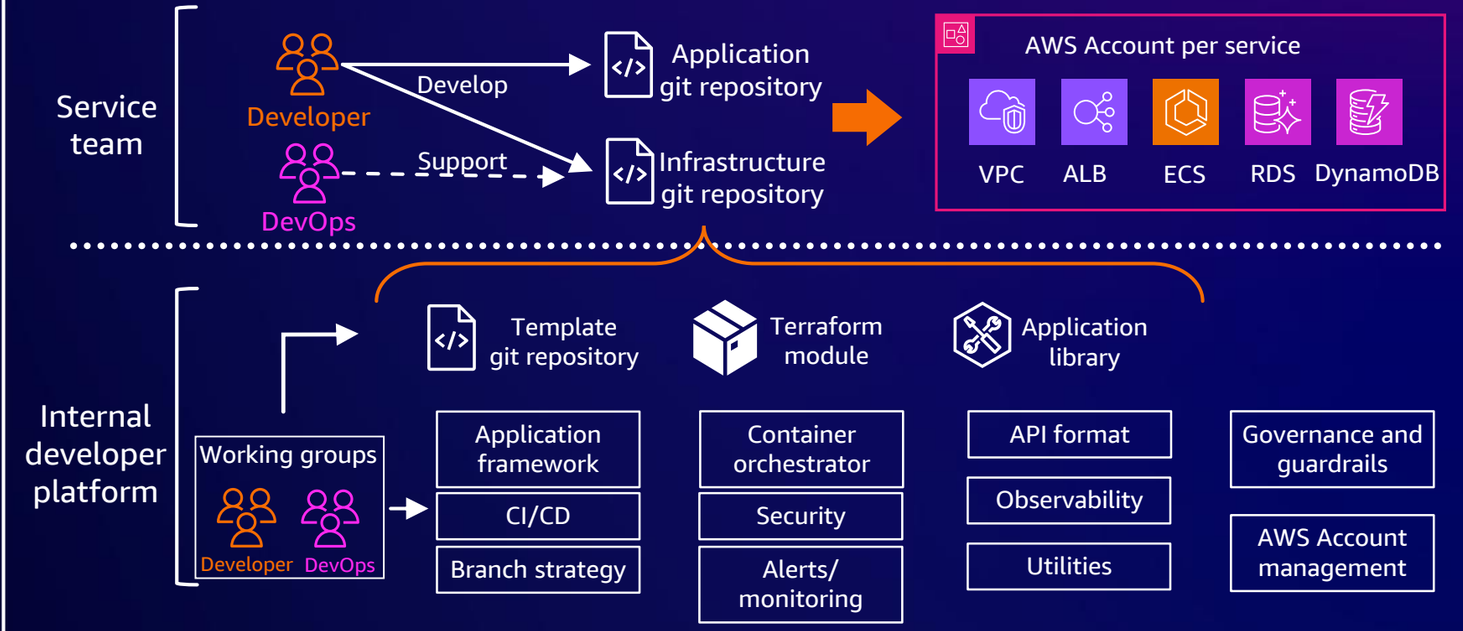


Our approach to platform engineering

No explicit platform engineers
- Lightweight and sufficient platform for developers

Developers participate in working group
- Able to define components necessary for self-service and improvement of productivity and maintainability
- Common libraries; circuit breaker, health check

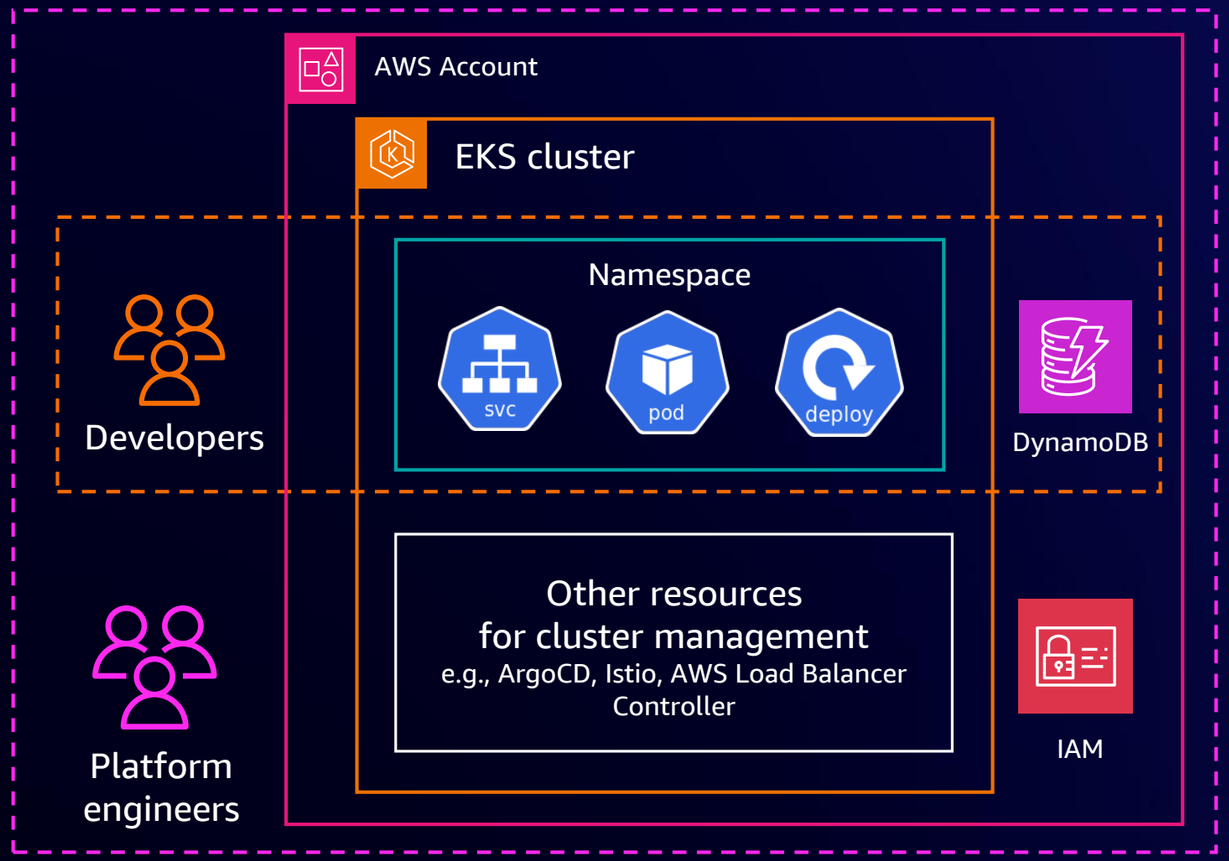
IDP does not have to cover all development use cases
- It’s the business logic that matters
- Provide and nurture developer experience



Container orchestrator

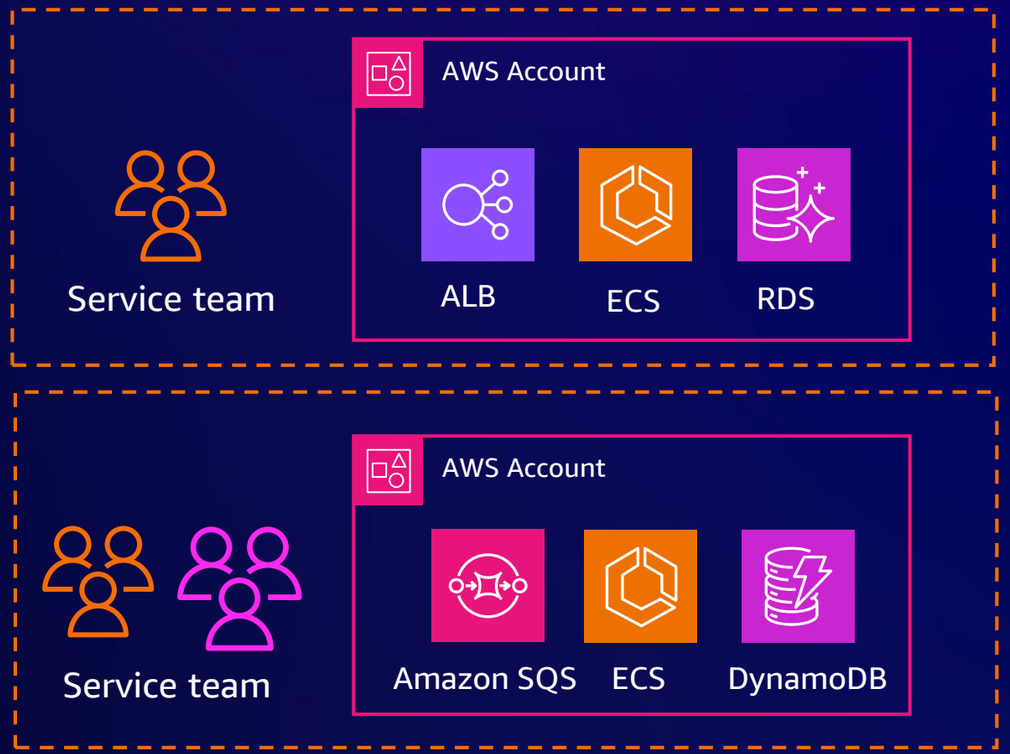
Amazon EKS

- Managed kubernetes cluster
- Powerful kubernetes ecosystem



Amazon ECS

- Managed container orchestrator
- Native integration with other AWS products



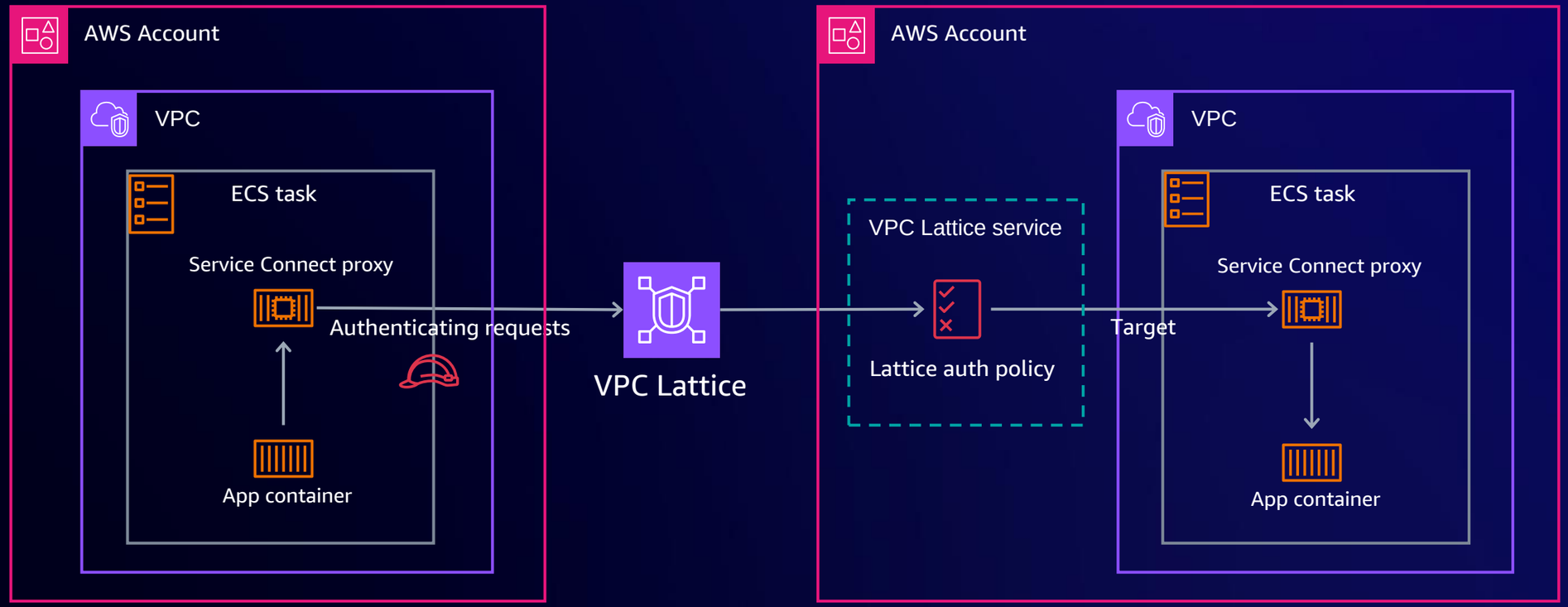
Pushing self-service further with Amazon ECS

Future road map of Amazon ECS

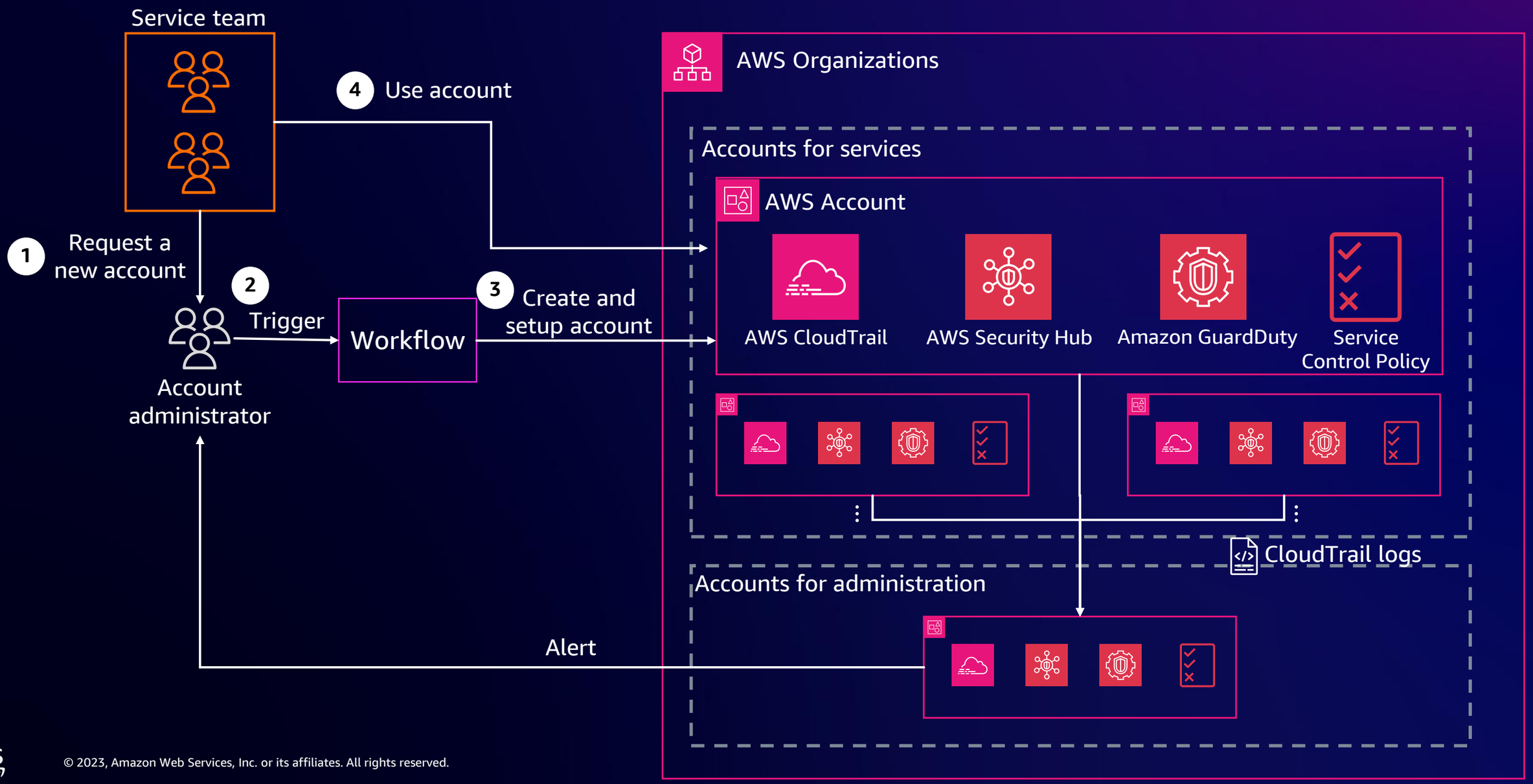
https://github.com/aws/containers-roadmap/

- ECS Service Connect with VPC Lattice integration
- Cross account and cross VPC access with IAM Policy-based authorization

Simplifies interconnection and further pushes self-service capability



AWS Account Management



Case study

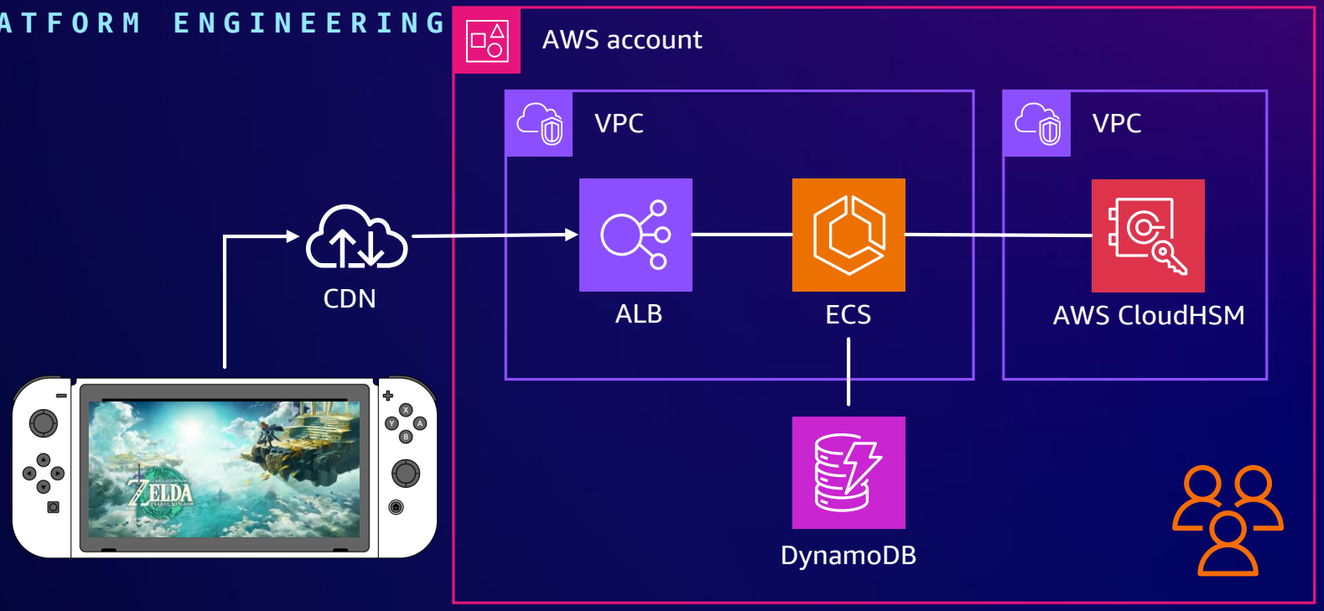
A SUCCESSFUL USE CASE OF MICROSERVICE AND PLATFORM ENGINEERING

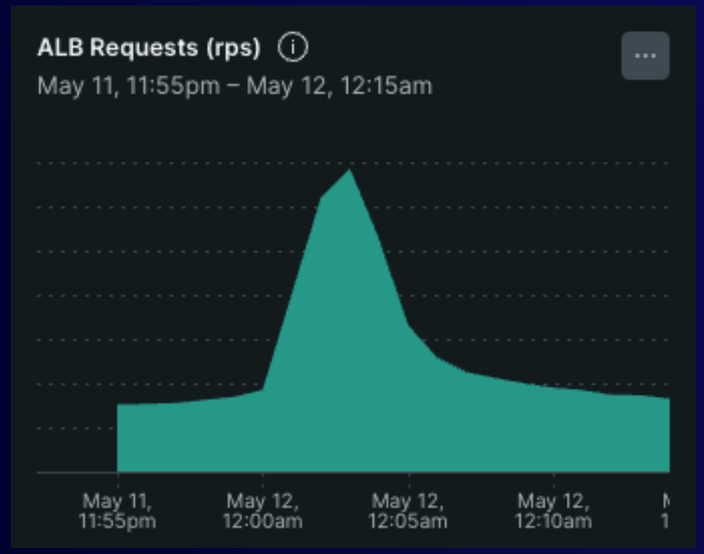
About the service

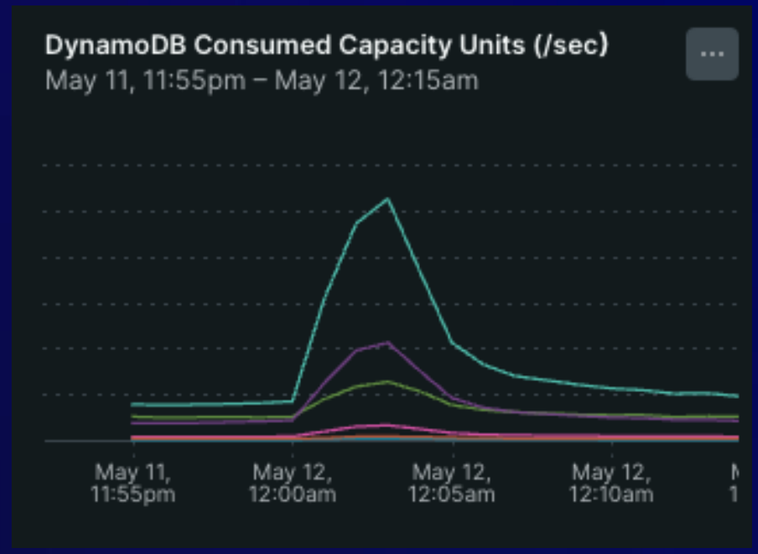
- Dynamically check the user's digital assets when user launches a game on a game console
- A large spike of traffic on the launch dates
- Built as microservice to achieve availability and scalability
- Developed and operated using IDP components

Launch of Legend of Zelda: Tears of the Kingdom

- Scaled the number of ECS tasks (120 tasks) and AWS CloudHSM instances
- Changed DynamoDB capacity mode







E-commerce API Platform

Need for e-commerce API Platform

Emerging use cases of e-commerce functionalities

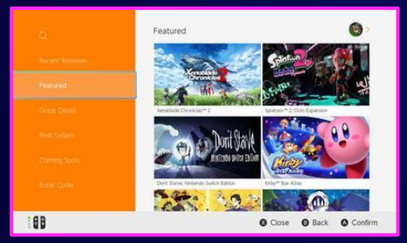
- Nintendo eShop
- My Nintendo Store
- Year In Review site
- Mobile apps, and more…

API Platform to support the use cases

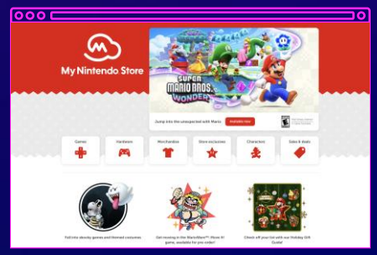
- Provide APIs to various developers/services

Year In Review

Mobile apps



Nintendo eShop



My Nintendo Store

E-commerce API Platform

Microservice

Platform engineering

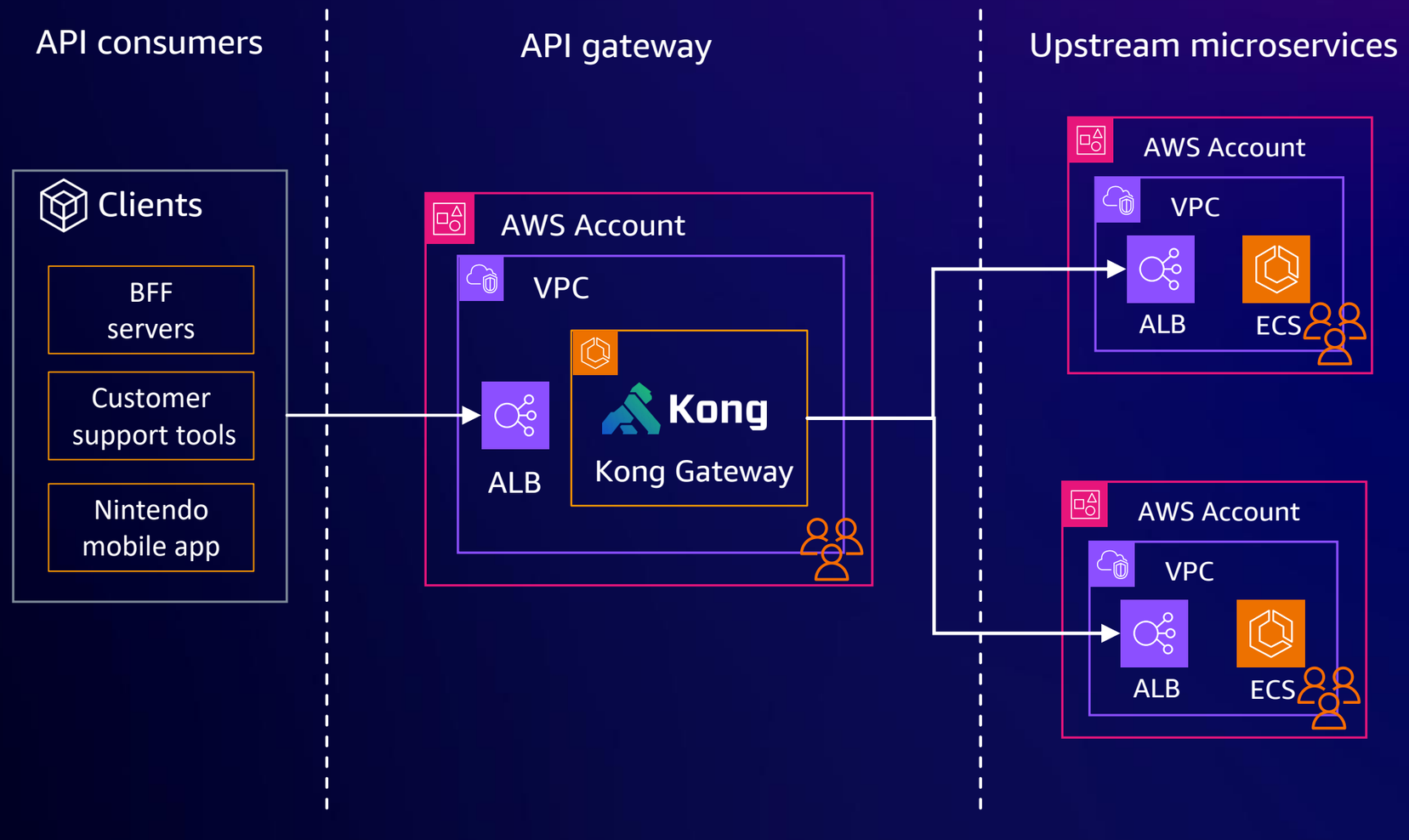
API gateway

Kong Gateway on ECS

- Lightweight, fast, and flexible cloud-native API gateway

Features provided by Kong

- Traffic control
- Authentication/authorization
- Rate limit
- Circuit breaker
- Monitoring API usage

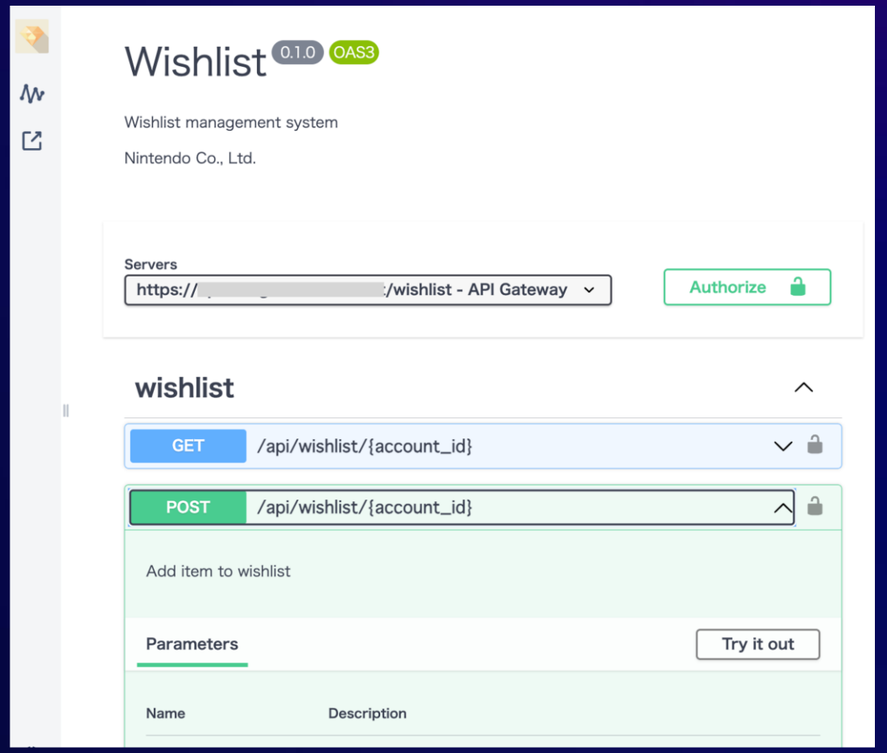


Key points of our API gateway

Monitor and control the usage

Visualize to monitor in real time how much each user is using which API

API catalog to discover and try out the APIs



Aggregate API Spec to build an API catalog

How to manage API gateway settings efficiently

Use Jsonnet
(A template language that generates JSON)

Manage differences of environment /microservice
in a “DRY“ manner

Summary and key takeaways

Introduced…

- The transition and the challenge of Nintendo eShop system
- Our approach to microservice and platform engineering
- Some of the components of our internal developer platform
- API gateway as a component of e-commerce API platform

- Microservice to achieve long term maintainability and embracement of new technologies
- Platform engineering to bring self-service capabilities
- Developers and DevOps engineers collaborate to develop efficient and developer friendly internal developer platform

- Amazon ECS fits well for self-service, independent AWS account environment
- Future prospects for ECS could further push the self-service development and operation in an independent AWS environment

Thank you!