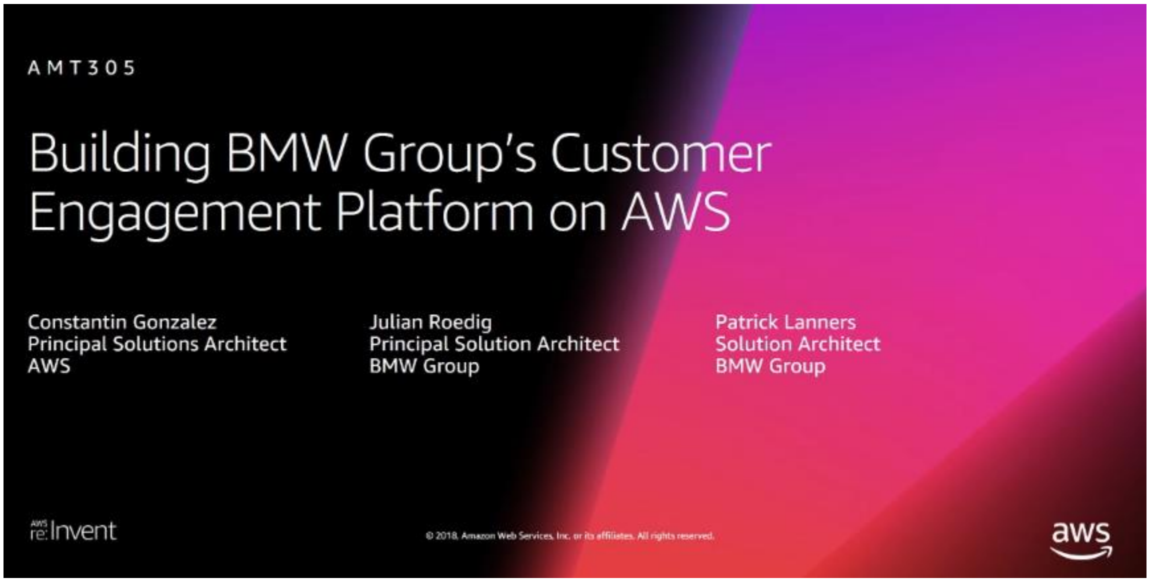


AMT305

Building BMW Group's Customer Engagement Platform on AWS

Constantin Gonzalez
Principal Solutions Architect
AWS

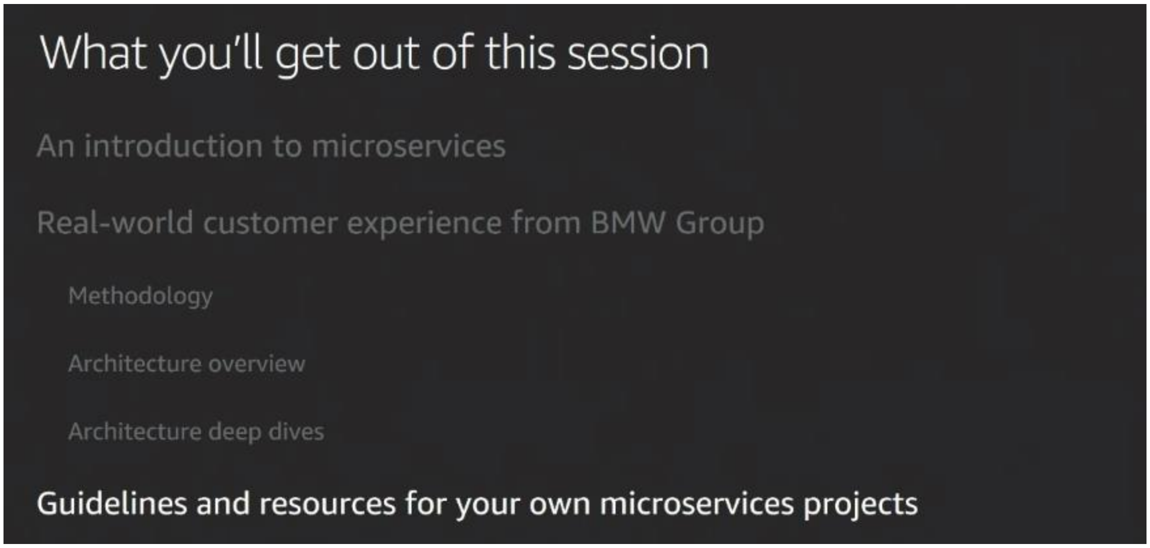
Julian Roedig
Principal Solution Architect
BMW Group

Patrick Lanners
Solution Architect
BMW Group

re:Invent

aws

In today's "always connected" world, brands must find unique ways to engage customers anywhere, anytime and across an ever-changing variety of formats. Large enterprises are often challenged by aging, monolithic applications that limit their ability to adapt quickly to changes. In this session, the BMW Group discusses how it is using microservices on the AWS Cloud to transform its customer engagement platform. Learn how the company built its Unified Configurator Platform (UCP) to serve 30+ branded customer-facing applications with over 300 RESTful API endpoints using services such as Amazon API Gateway, AWS Lambda, Amazon Elastic Beanstalk, and AWS Elastic Container Service. Additionally, the BMW Group discusses how Game Days and Chaos Monkey methodologies led to the success of the overall program.



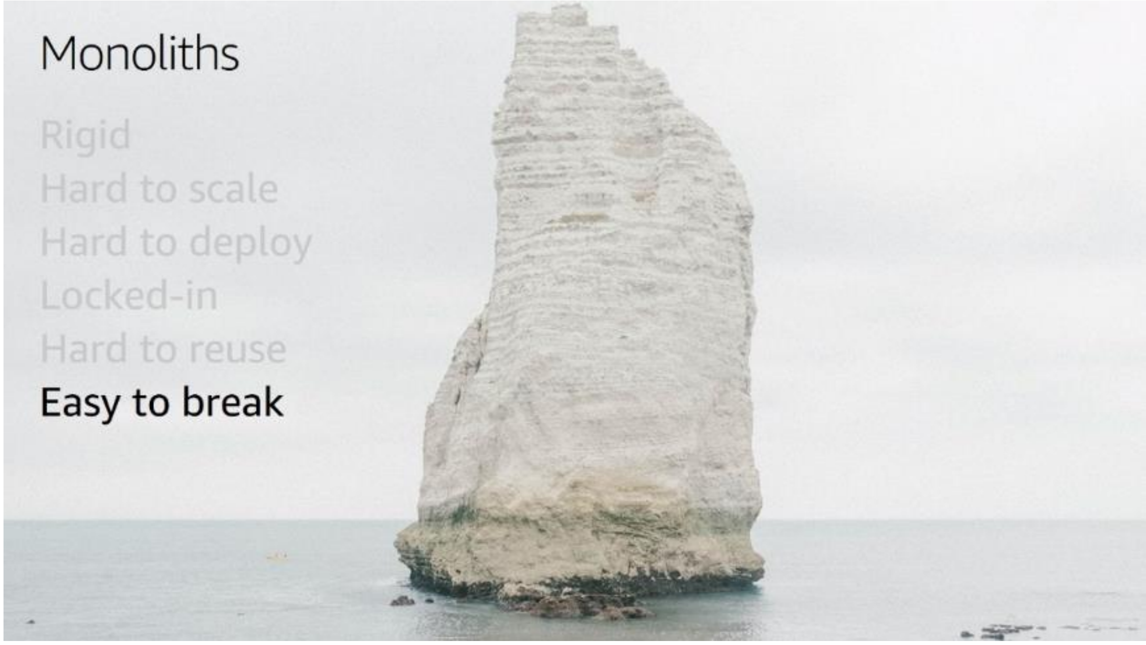
What you'll get out of this session

An introduction to microservices

Real-world customer experience from BMW Group

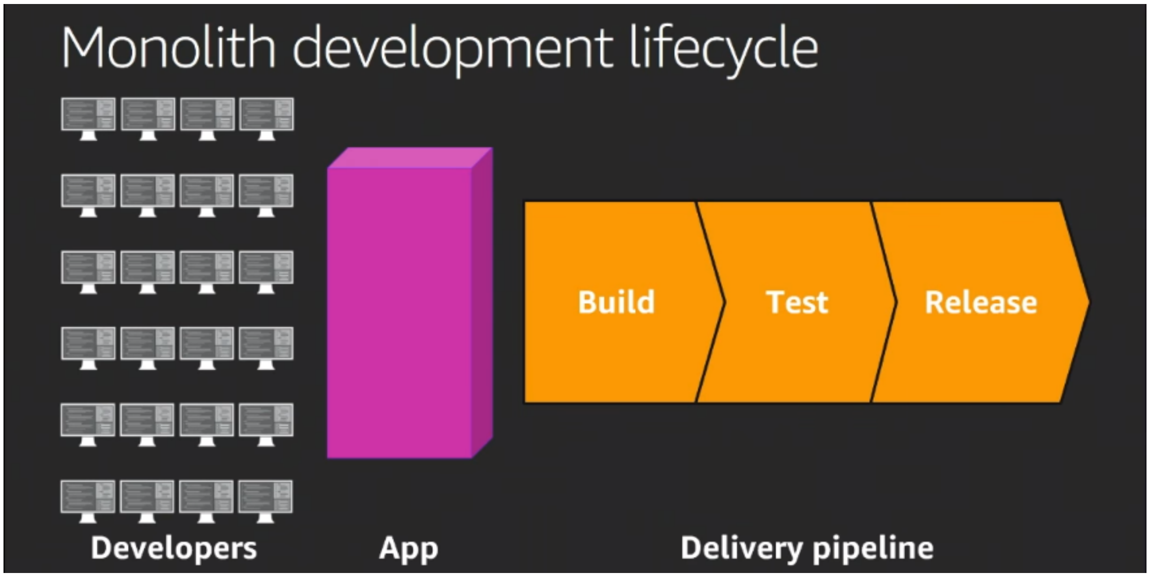
- Methodology
- Architecture overview
- Architecture deep dives

Guidelines and resources for your own microservices projects



Monoliths

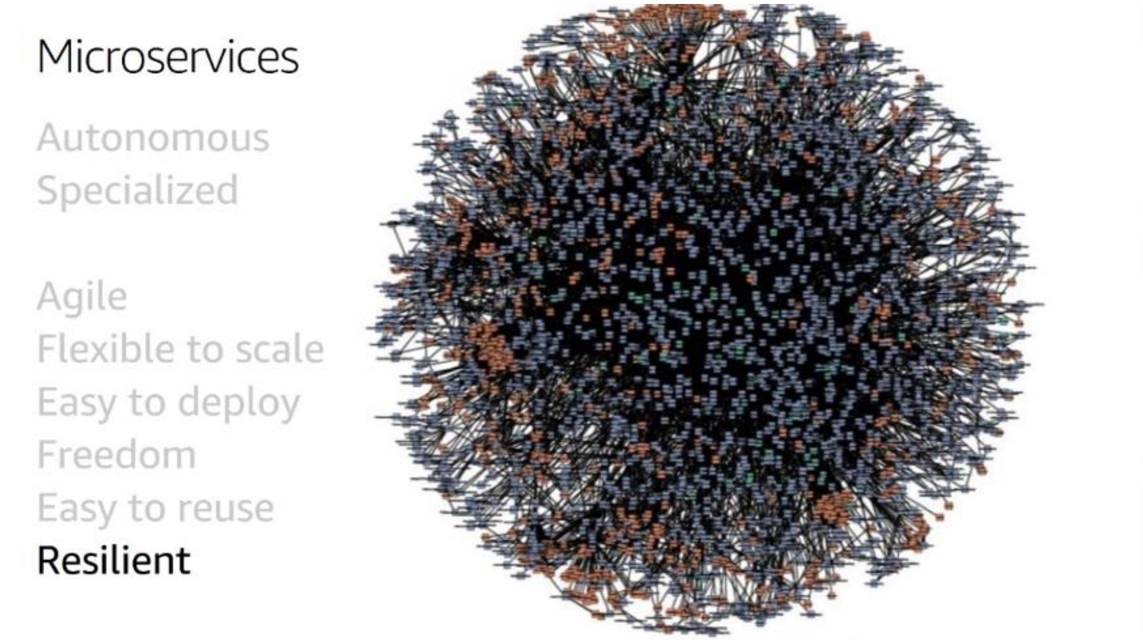
Rigid
Hard to scale
Hard to deploy
Locked-in
Hard to reuse
Easy to break



Monolith development lifecycle

Build
Test
Release

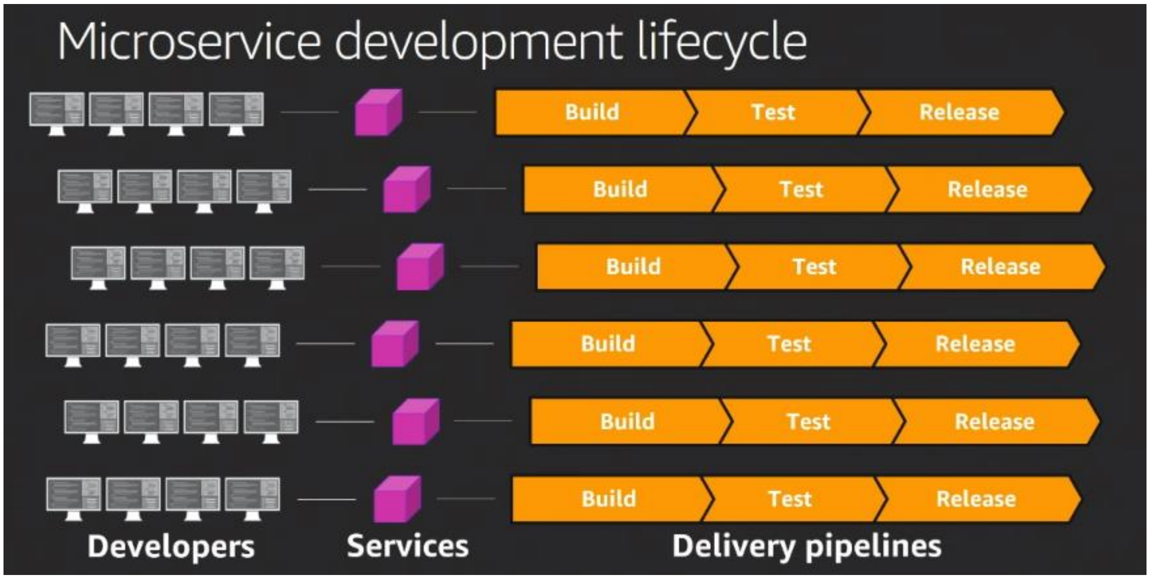
Developers
App
Delivery pipeline



Microservices

Autonomous
Specialized

Agile
Flexible to scale
Easy to deploy
Freedom
Easy to reuse
Resilient



Microservice development lifecycle

Build Test Release

Build Test Release

Build Test Release

Build Test Release

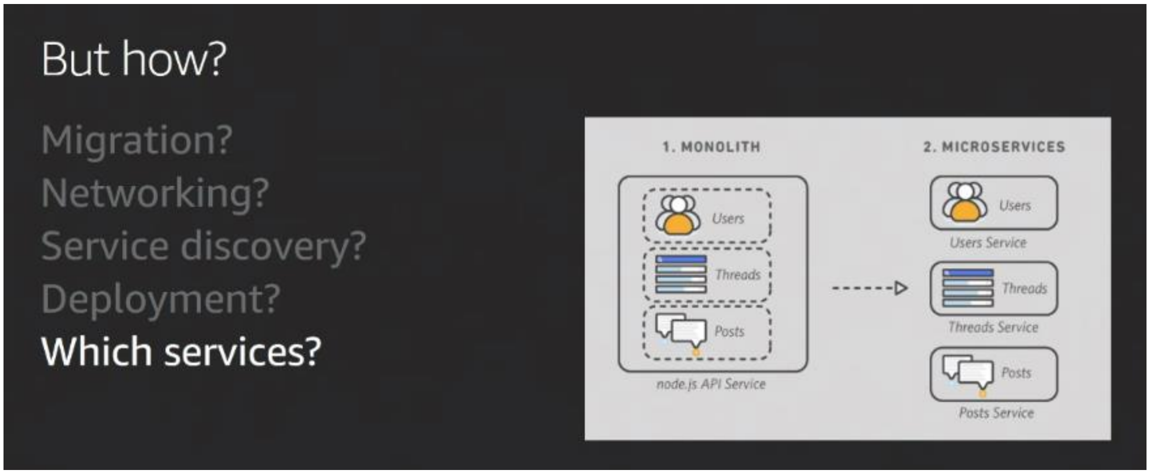
Build Test Release

Build Test Release

Developers

Services

Delivery pipelines



But how?

Migration?

Networking?

Service discovery?

Deployment?

Which services?

1. MONOLITH

Users

Threads

Posts

node.js API Service

2. MICROSERVICES

Users

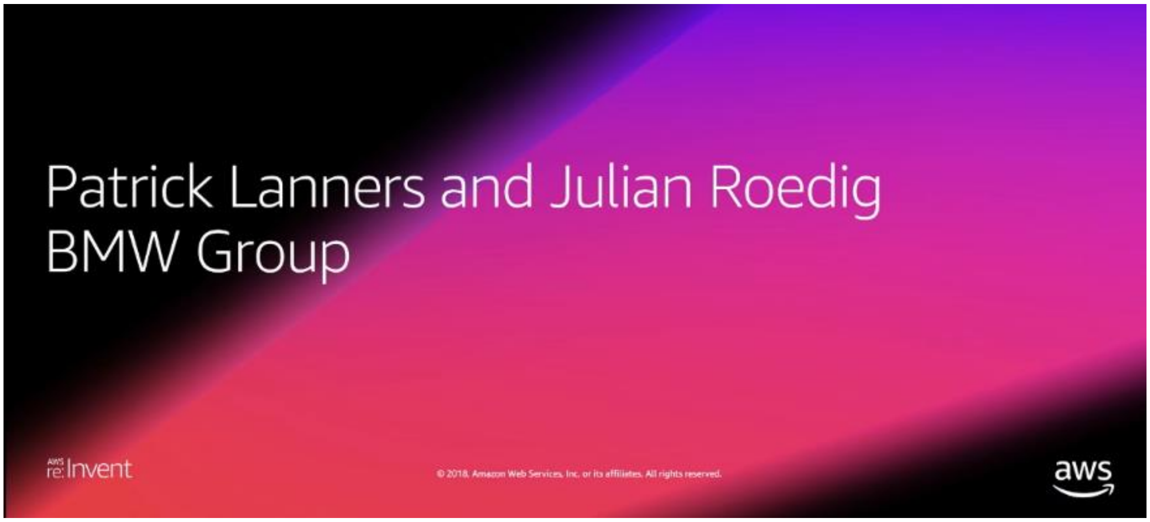
Users Service

Threads

Threads Service

Posts

Posts Service



Patrick Lanners and Julian Roedig
BMW Group

AWS re:Invent

aws

1 **Introduction**

2 Architecture overview

3 Architecture deep dives

4 Conclusion

OUR PLATFORM CONSOLIDATES THE VEHICLE PRODUCTS AS SERVICES

~7 Engines

~12 Colors

~14 Wheels

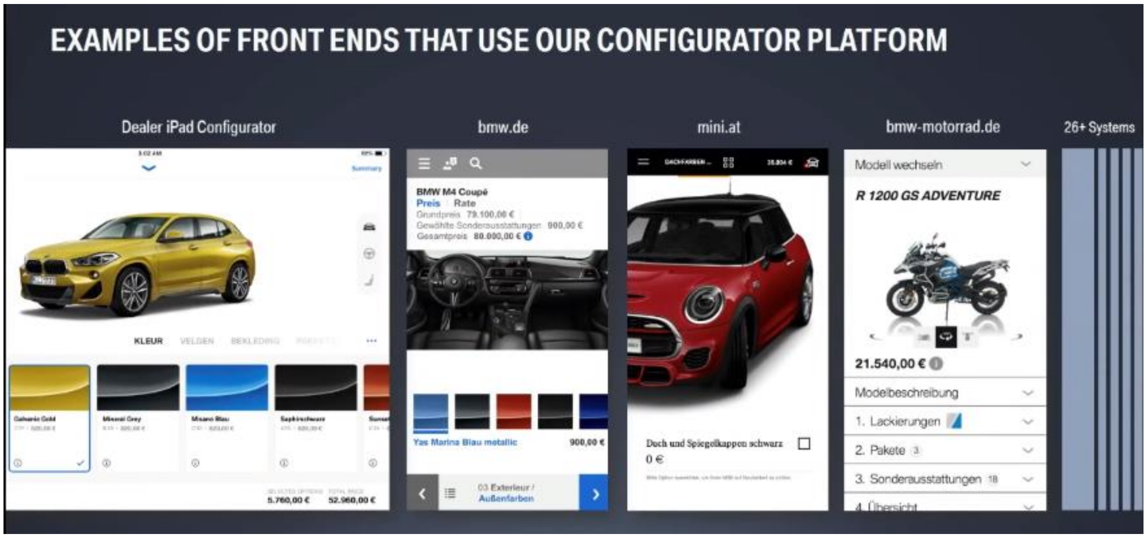
~11 Upholstery

~7 Trims

~90 Optional Equipment

=

Many possibilities to configure your dream car



EXAMPLES OF FRONT ENDS THAT USE OUR CONFIGURATOR PLATFORM

Dealer iPad Configurator

5.760,00 € 52.960,00 €

bmw.de

BMW M4 Coupé

Grundpreis 79.100,00 €

Gewählte Sonderausstattungen 900,00 €

Gesamtpreis 80.000,00 €

Yas Marina Blau metallic 900,00 €

mini.at

Dach und Spiegelkappen schwarz □

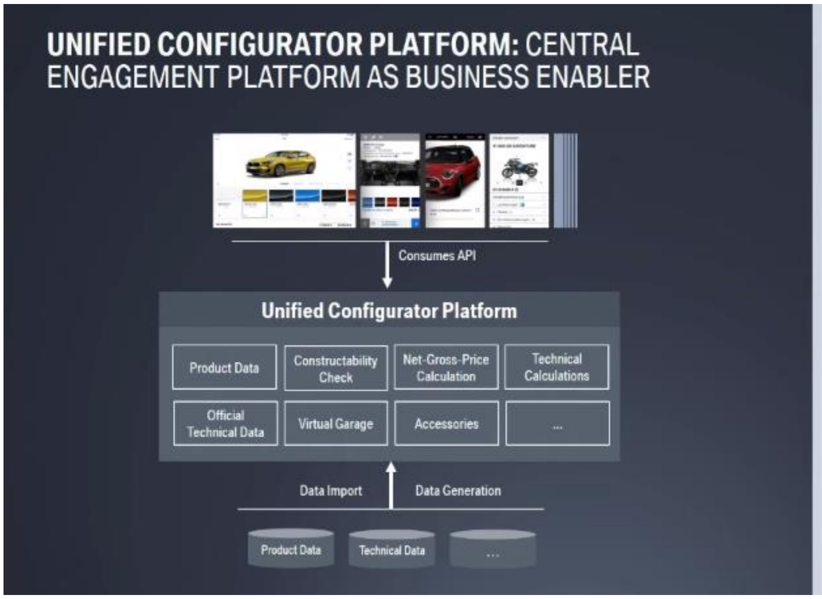
0 €

bmw-motorrad.de

R 1200 GS ADVENTURE

21.540,00 €

26+ Systems



UNIFIED CONFIGURATOR PLATFORM: CENTRAL ENGAGEMENT PLATFORM AS BUSINESS ENABLER

Consumes API

Unified Configurator Platform

Product Data | Constructability Check | Net-Gross-Price Calculation | Technical Calculations

Official Technical Data | Virtual Garage | Accessories | ...

Data Import | Data Generation

Product Data | Technical Data | ...

Key facts:

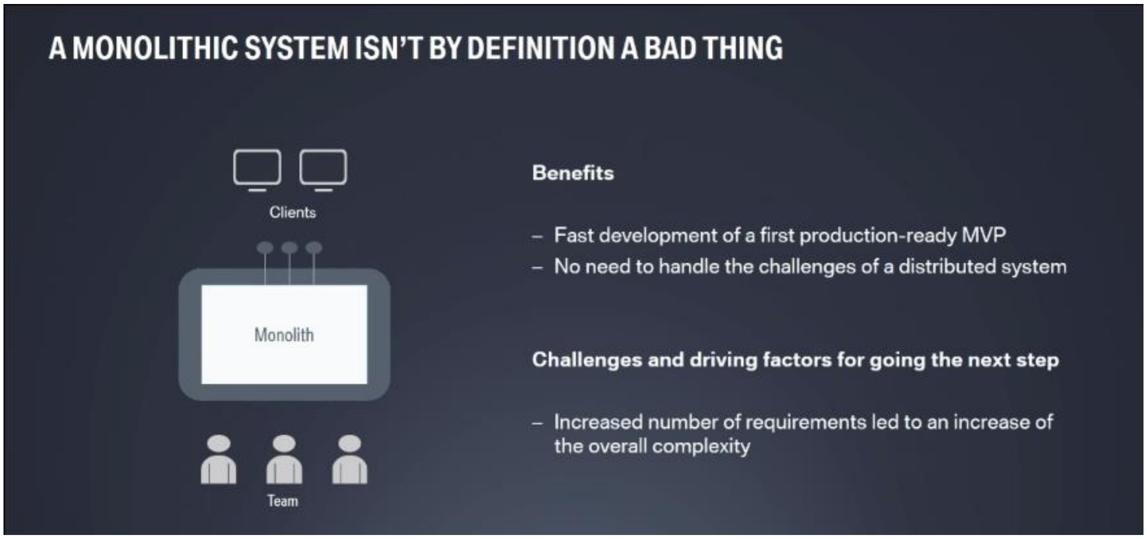
- RESTful API
- Used by ~30 products
- Hundreds of Mio API calls per month
- Supports all BMW Group brands and markets

1 Introduction

2 **Architecture overview**

3 Architecture deep dives

4 Conclusion



A MONOLITHIC SYSTEM ISN'T BY DEFINITION A BAD THING

Clients

Monolith

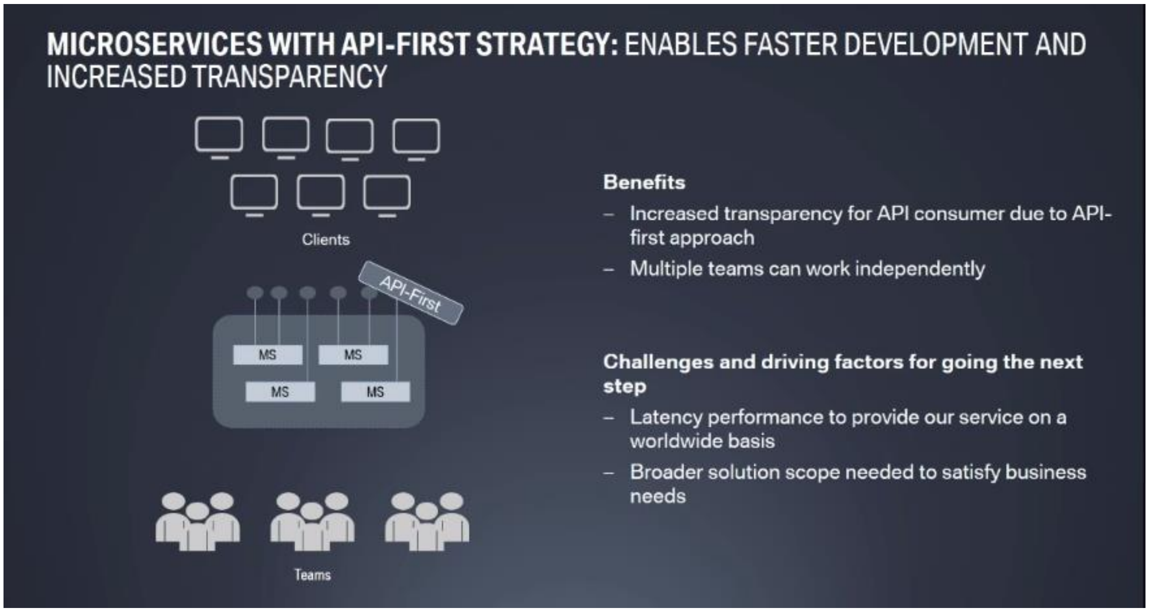
Team

Benefits

- Fast development of a first production-ready MVP
- No need to handle the challenges of a distributed system

Challenges and driving factors for going the next step

- Increased number of requirements led to an increase of the overall complexity



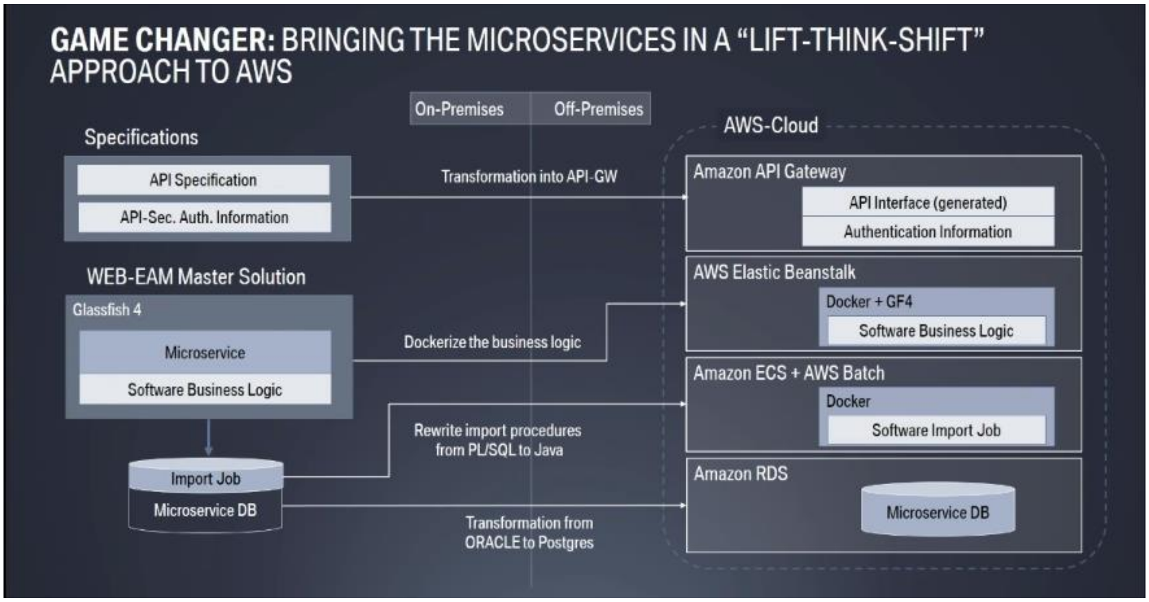
MICROSERVICES WITH API-FIRST STRATEGY: ENABLES FASTER DEVELOPMENT AND INCREASED TRANSPARENCY

Benefits

- Increased transparency for API consumer due to API-first approach
- Multiple teams can work independently

Challenges and driving factors for going the next step

- Latency performance to provide our service on a worldwide basis
- Broader solution scope needed to satisfy business needs



GAME CHANGER: BRINGING THE MICROSERVICES IN A "LIFT-THINK-SHIFT" APPROACH TO AWS

On-Premises | Off-Premises

Specifications

API Specification

API-Sec. Auth. Information

WEB-EAM Master Solution

Glassfish 4

Microservice

Software Business Logic

Import Job

Microservice DB

Transformation into API-GW

Dockerize the business logic

Rewrite import procedures from PL/SQL to Java

Transformation from ORACLE to Postgres

AWS-Cloud

Amazon API Gateway

API Interface (generated)

Authentication Information

AWS Elastic Beanstalk

Docker + GF4

Software Business Logic

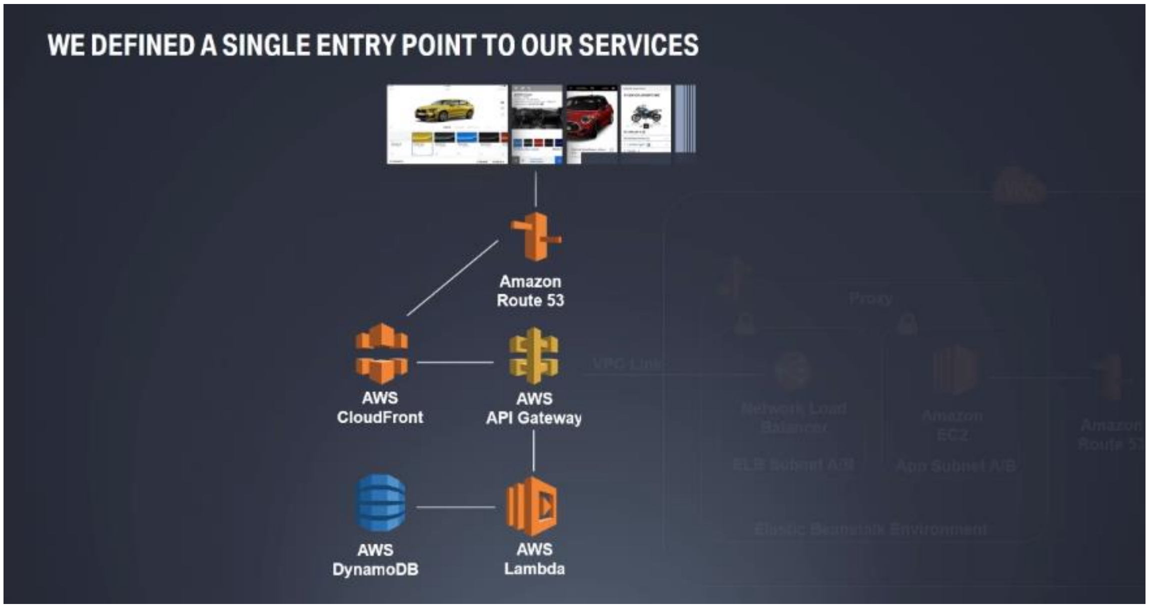
Amazon ECS + AWS Batch

Docker

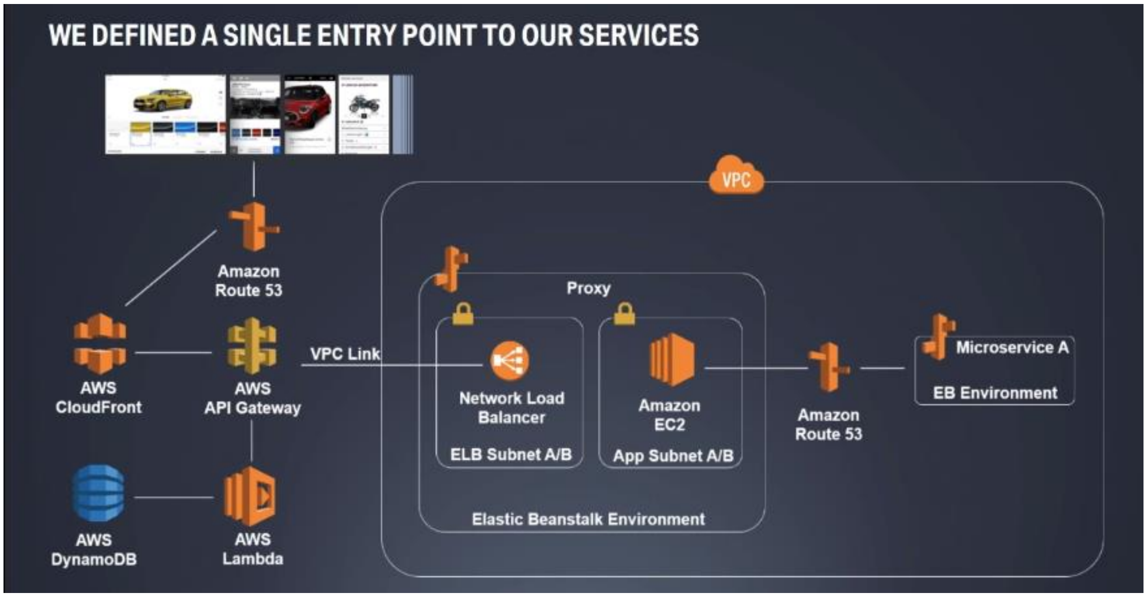
Software Import Job

Amazon RDS

Microservice DB



WE DEFINED A SINGLE ENTRY POINT TO OUR SERVICES



WE DEFINED A SINGLE ENTRY POINT TO OUR SERVICES

Amazon Route 53

AWS CloudFront

AWS API Gateway

VPC Link

AWS DynamoDB

AWS Lambda

VPC

Proxy

Network Load Balancer

ELB Subnet A/B

Amazon EC2

App Subnet A/B

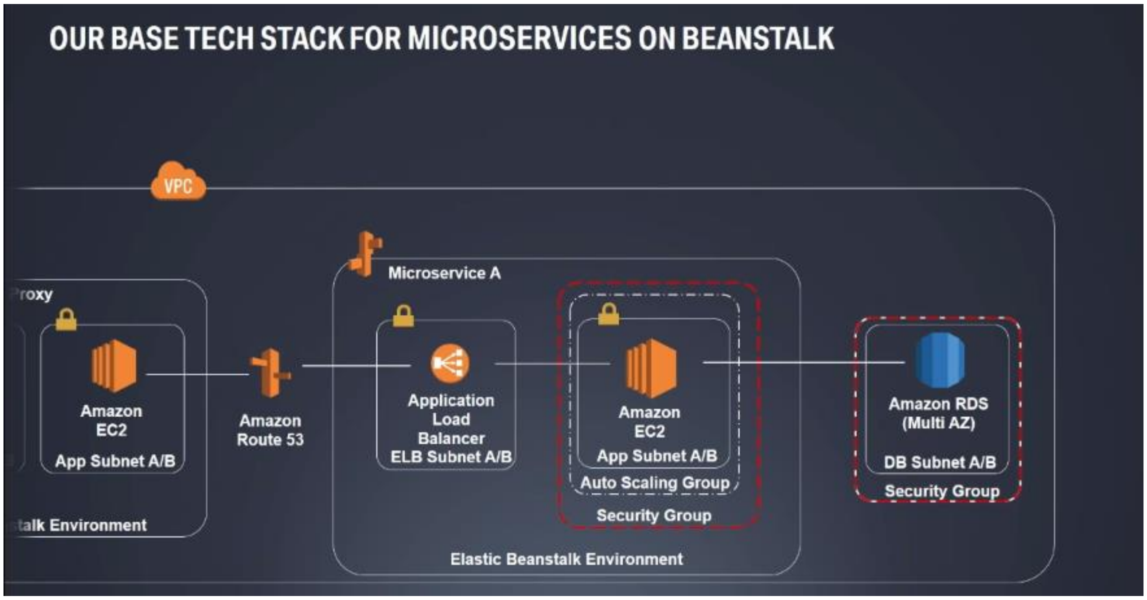
Elastic Beanstalk Environment

Amazon Route 53

Microservice A

EB Environment

Our VPC consists of 3 different subnets for our load balancers, our applications, and our databases respectively. For our microservice applications, we used Route53 routing with private hosted zones and register the microservice endpoints.



OUR BASE TECH STACK FOR MICROSERVICES ON BEANSTALK

VPC

roxy

Amazon EC2

App Subnet A/B

talk Environment

Amazon Route 53

Microservice A

Application Load Balancer

ELB Subnet A/B

Amazon EC2

App Subnet A/B

Auto Scaling Group

Security Group

Amazon RDS (Multi AZ)

DB Subnet A/B

Security Group

Elastic Beanstalk Environment

On ELB, we deploy our Java web app microservices as Docker containers

1 Introduction

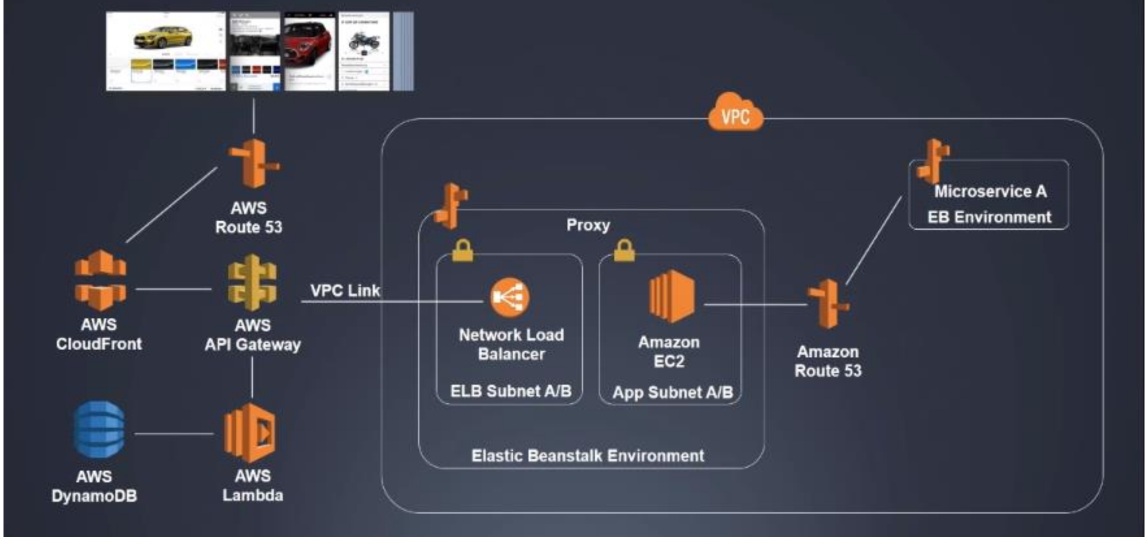
2 Architecture overview

3 Architecture deep dives

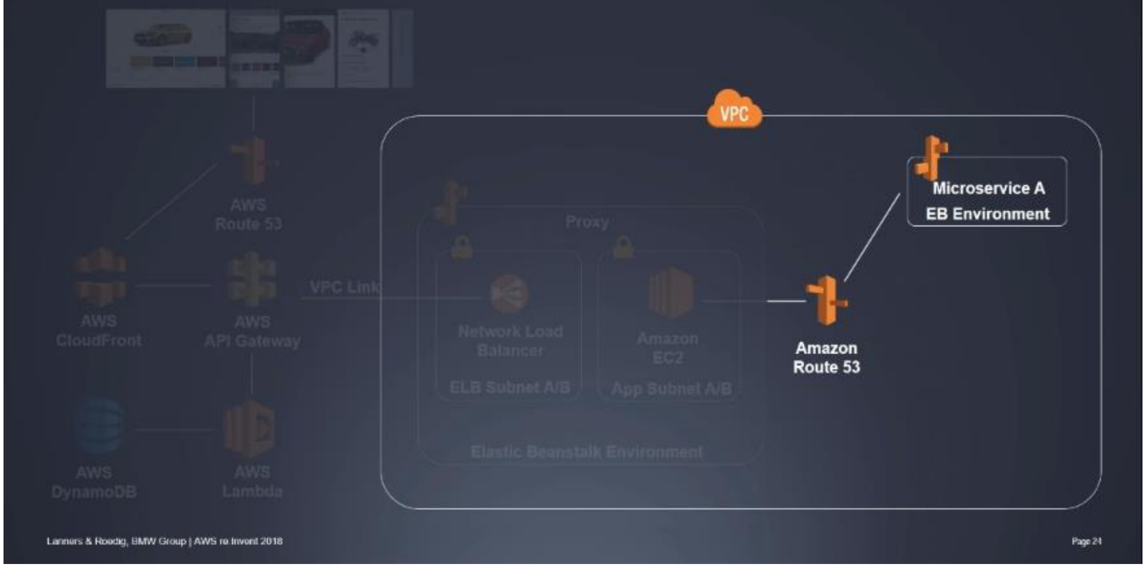
- Service discovery
- AWS Lambda CI/CD
- Offline data processing
- Advanced Lambda use case
- Environments on demand

4 Conclusion

ENHANCED FLEXIBILITY THROUGH DNS-BASED SERVICE DISCOVERY



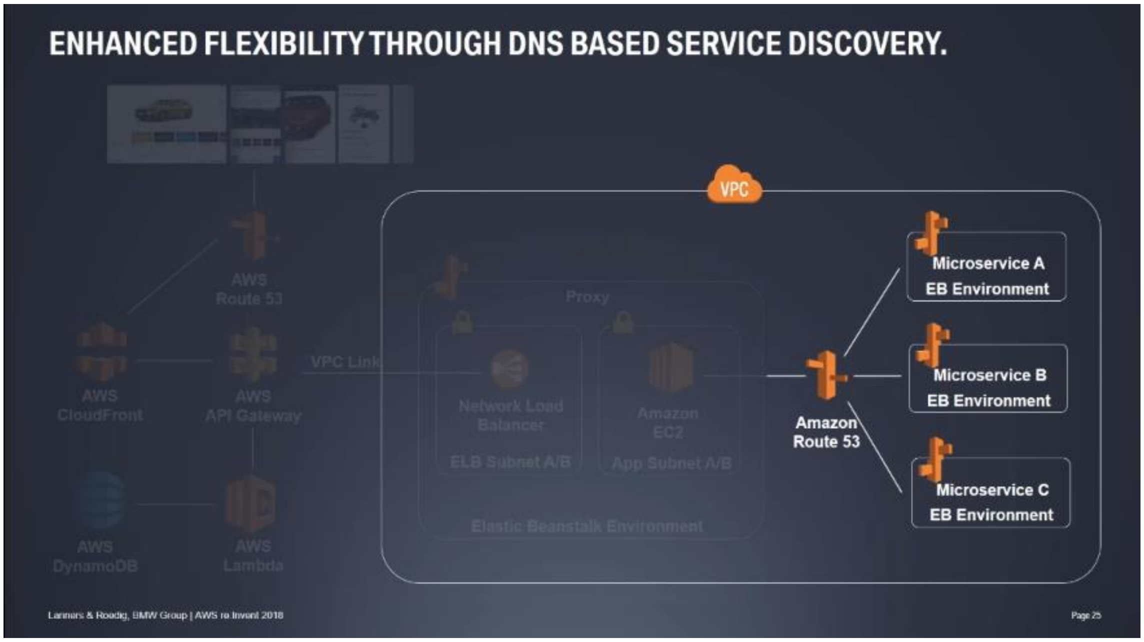
ENHANCED FLEXIBILITY THROUGH DNS-BASED SERVICE DISCOVERY



Lanners & Roedig, BMW Group | AWS re:Invent 2018

Page 24

Within our VPC, we have a private hosted zone that allows each microservice to be accessible via an internal domain name like **microservice-a.bmw.com**, **microservice-b.bmw.com** that lay behind a LB and with a Route53 A-record.

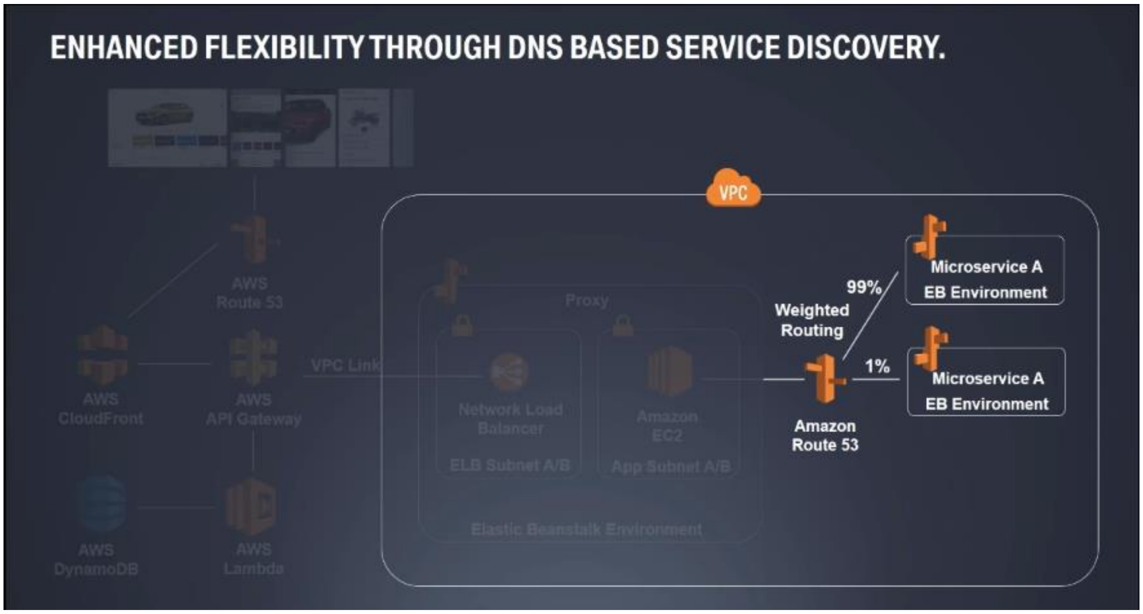


ENHANCED FLEXIBILITY THROUGH DNS BASED SERVICE DISCOVERY.

Lanners & Roedig, BMW Group | AWS re:Invent 2018

Page 25

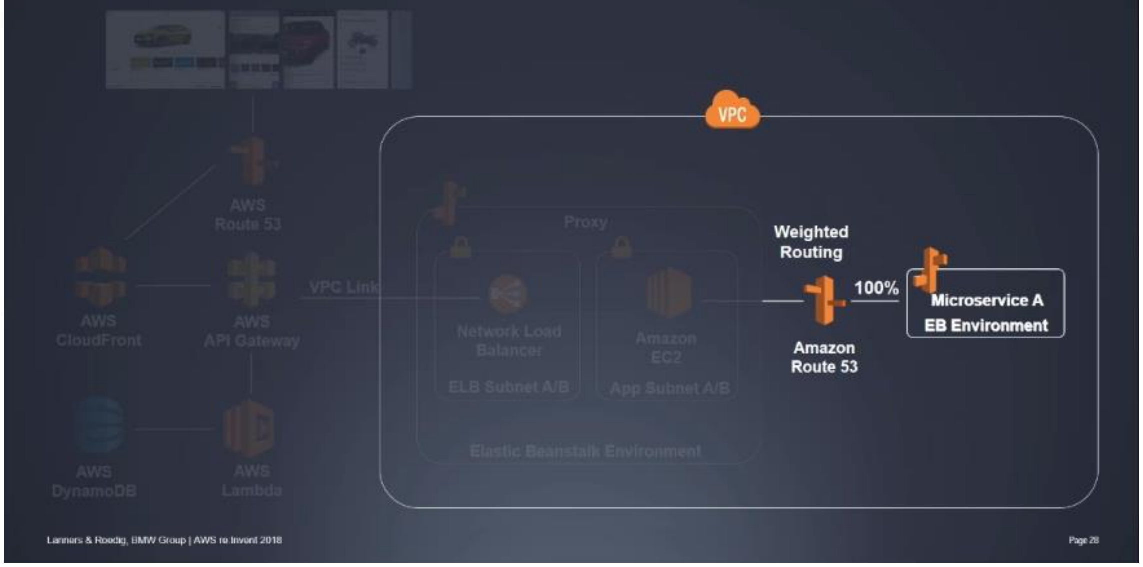
We also use Route53 as an internal service discovery mechanism to allow the microservices to talk to each other, this also allows the Proxy to route traffic to the microservices using their internal domain names.



ENHANCED FLEXIBILITY THROUGH DNS BASED SERVICE DISCOVERY.

Route53 allows us to create multiple A-records for the same domain name and route the traffic through to multiple targets, this allows us to verify new deployments.

ENHANCED FLEXIBILITY THROUGH DNS BASED SERVICE DISCOVERY.



VPC

AWS Route 53

AWS CloudFront

AWS API Gateway

VPC Link

AWS DynamoDB

AWS Lambda

Proxy

Network Load Balancer

ELB Subnet A/B

Amazon EC2

App Subnet A/B

Elastic Beanstalk Environment

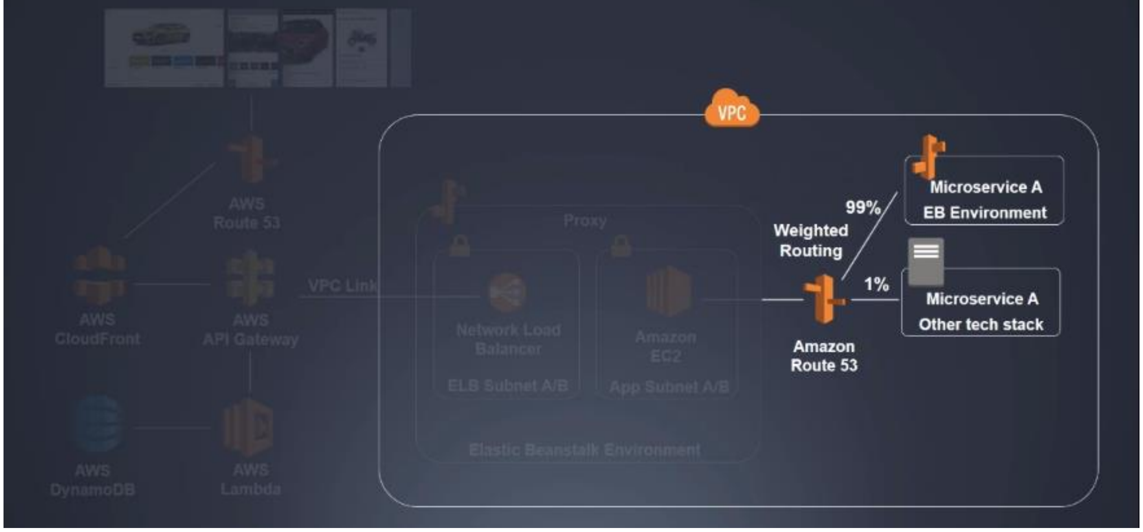
Weighted Routing

100%

Amazon Route 53

Microservice A
EB Environment

ENHANCED FLEXIBILITY THROUGH DNS BASED SERVICE DISCOVERY.



VPC

AWS Route 53

AWS CloudFront

AWS API Gateway

VPC Link

AWS DynamoDB

AWS Lambda

Proxy

Network Load Balancer

ELB Subnet A/B

Amazon EC2

App Subnet A/B

Elastic Beanstalk Environment

Weighted Routing

99%

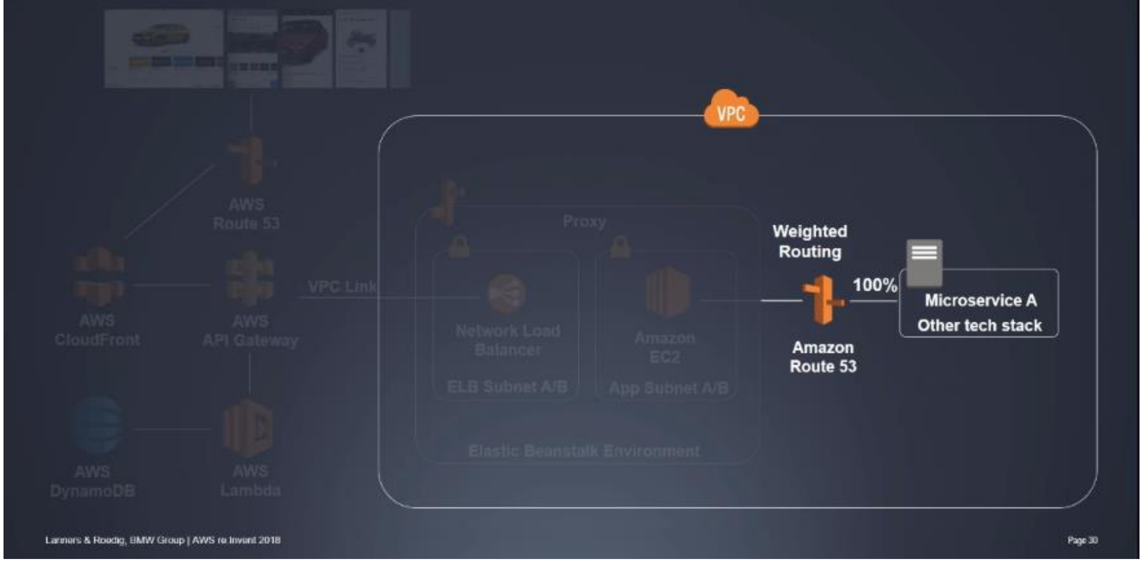
1%

Amazon Route 53

Microservice A
EB Environment

Microservice A
Other tech stack

ENHANCED FLEXIBILITY THROUGH DNS BASED SERVICE DISCOVERY.



VPC

AWS Route 53

AWS CloudFront

AWS API Gateway

VPC Link

AWS DynamoDB

AWS Lambda

Proxy

Network Load Balancer

ELB Subnet A/B

Amazon EC2

App Subnet A/B

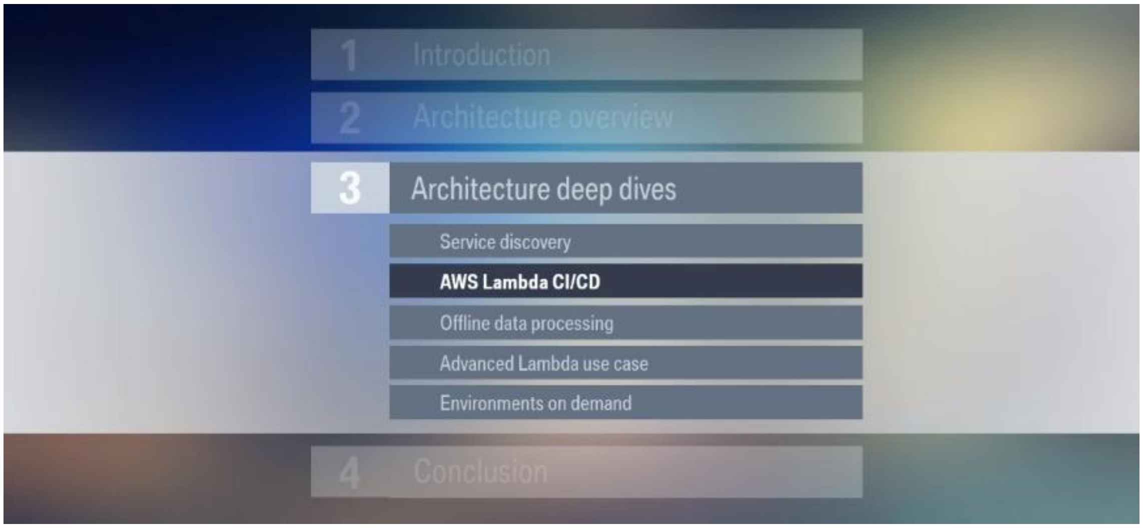
Elastic Beanstalk Environment

Weighted Routing

100%

Amazon Route 53

Microservice A
Other tech stack



1 Introduction

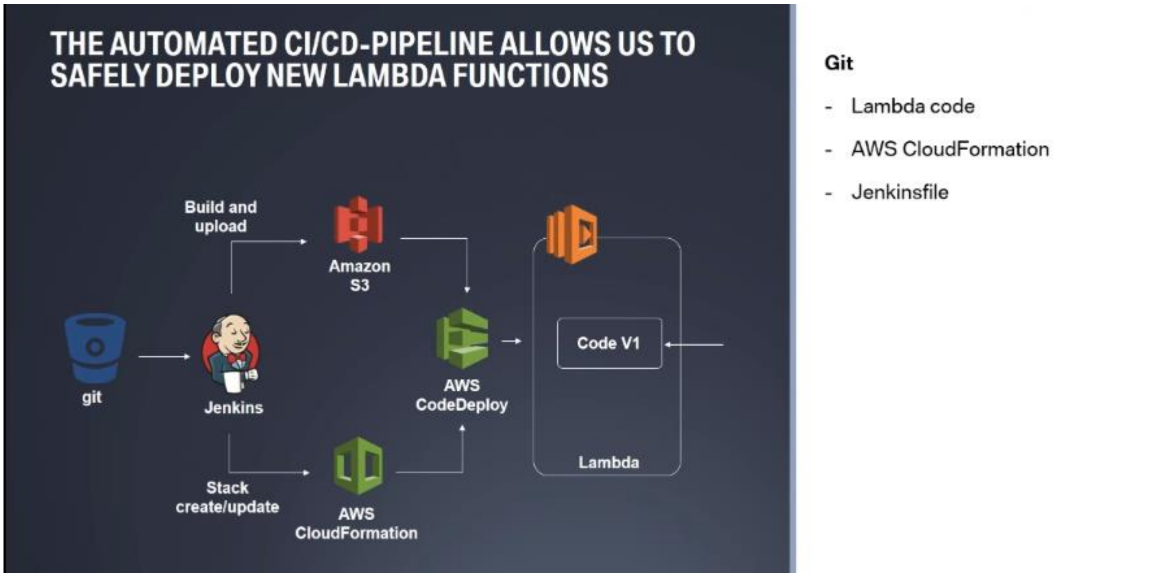
2 Architecture overview

3 Architecture deep dives

- Service discovery
- **AWS Lambda CI/CD**
- Offline data processing
- Advanced Lambda use case
- Environments on demand

4 Conclusion

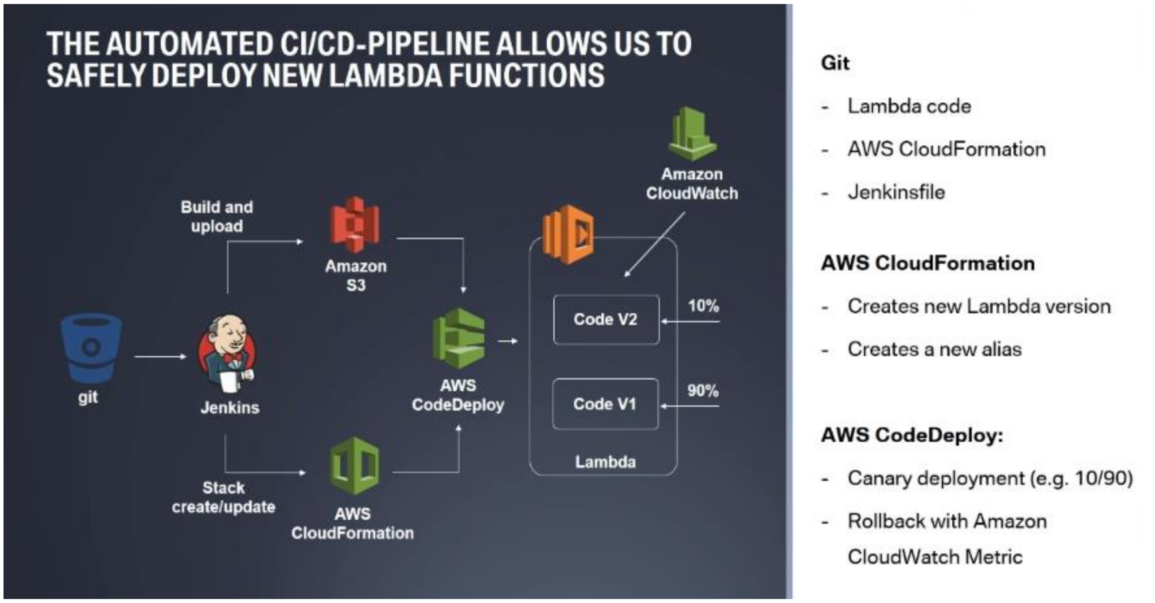
How do we deploy the different Lambda functions that we use?



THE AUTOMATED CI/CD-PIPELINE ALLOWS US TO SAFELY DEPLOY NEW LAMBDA FUNCTIONS

Git

- Lambda code
- AWS CloudFormation
- Jenkinsfile



THE AUTOMATED CI/CD-PIPELINE ALLOWS US TO SAFELY DEPLOY NEW LAMBDA FUNCTIONS

Git

- Lambda code
- AWS CloudFormation
- Jenkinsfile

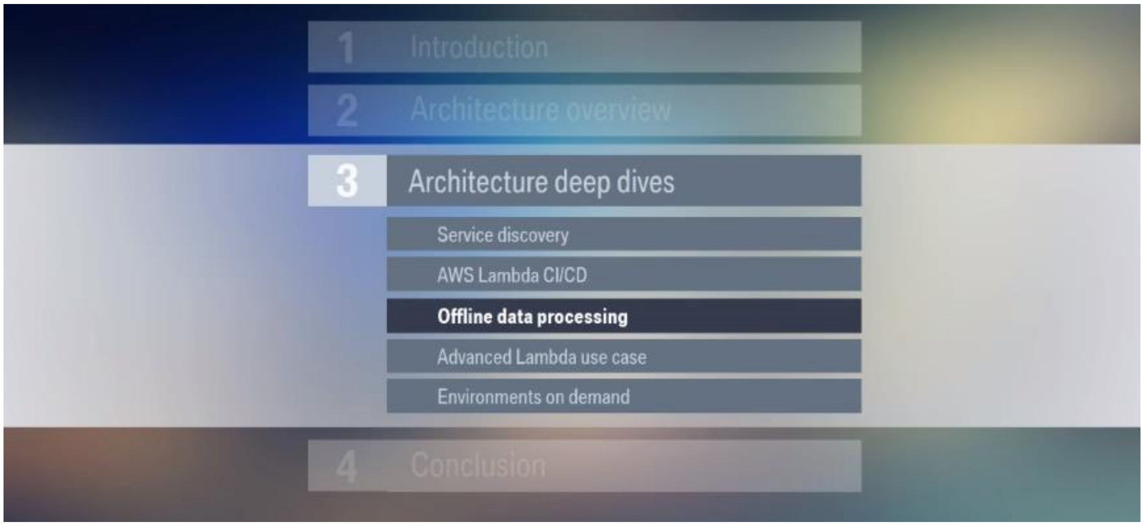
AWS CloudFormation

- Creates new Lambda version
- Creates a new alias

AWS CodeDeploy:

- Canary deployment (e.g. 10/90)
- Rollback with Amazon CloudWatch Metric

We can then set the deployment preferences in CodeDeploy like splitting traffic temporarily and switch the full traffic in 10 mins to the new version.



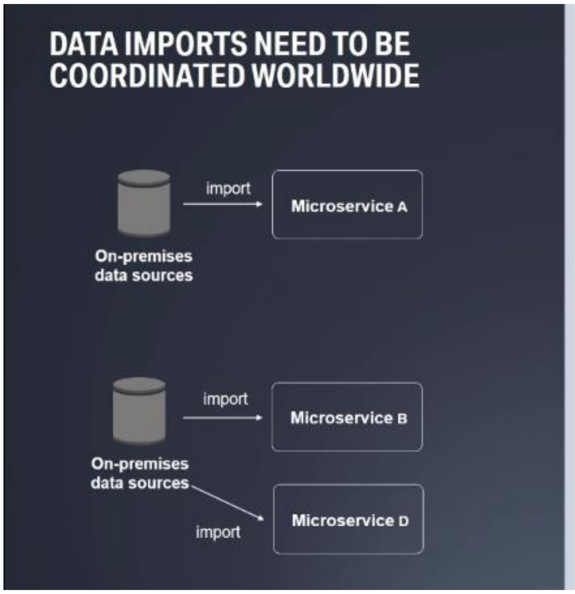
1 Introduction

2 Architecture overview

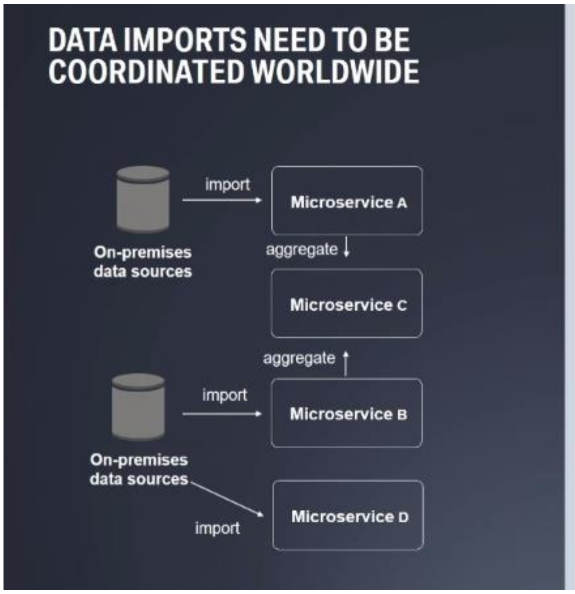
3 Architecture deep dives

- Service discovery
- AWS Lambda CI/CD
- **Offline data processing**
- Advanced Lambda use case
- Environments on demand

4 Conclusion

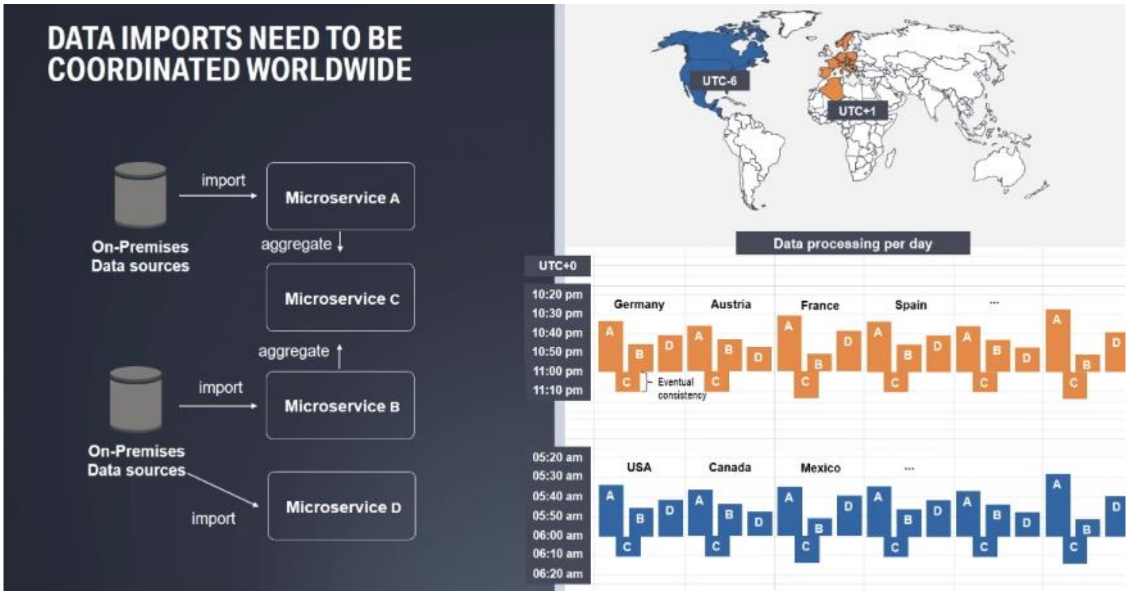


DATA IMPORTS NEED TO BE COORDINATED WORLDWIDE



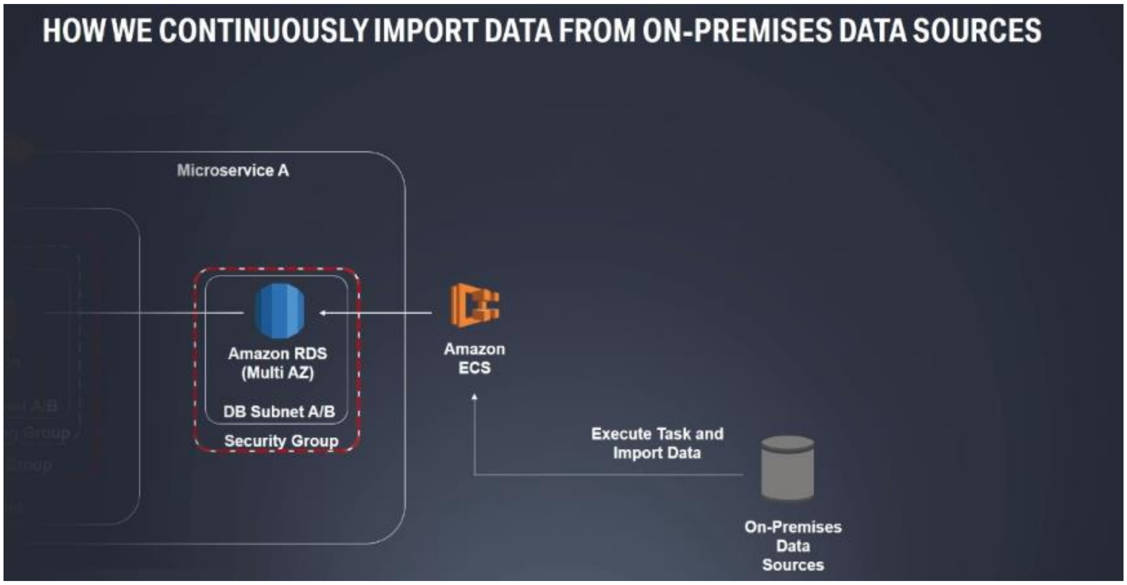
DATA IMPORTS NEED TO BE COORDINATED WORLDWIDE

We also have microservices that aggregate the results of other microservices by calling their APIs for data.

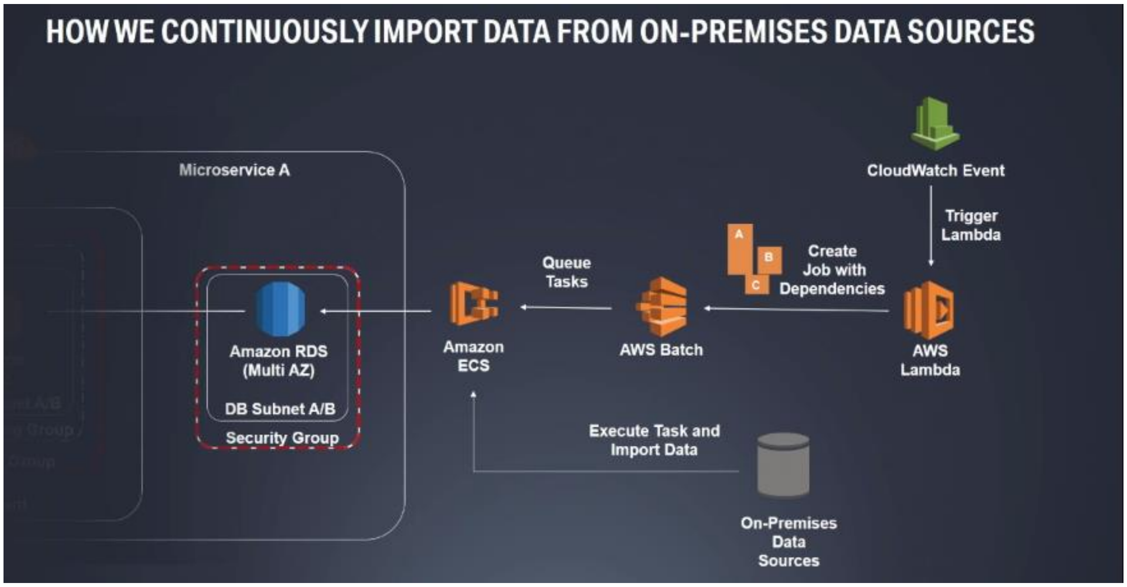


DATA IMPORTS NEED TO BE COORDINATED WORLDWIDE

We also do data processing in different time zones, we need the right job executed at the right time.

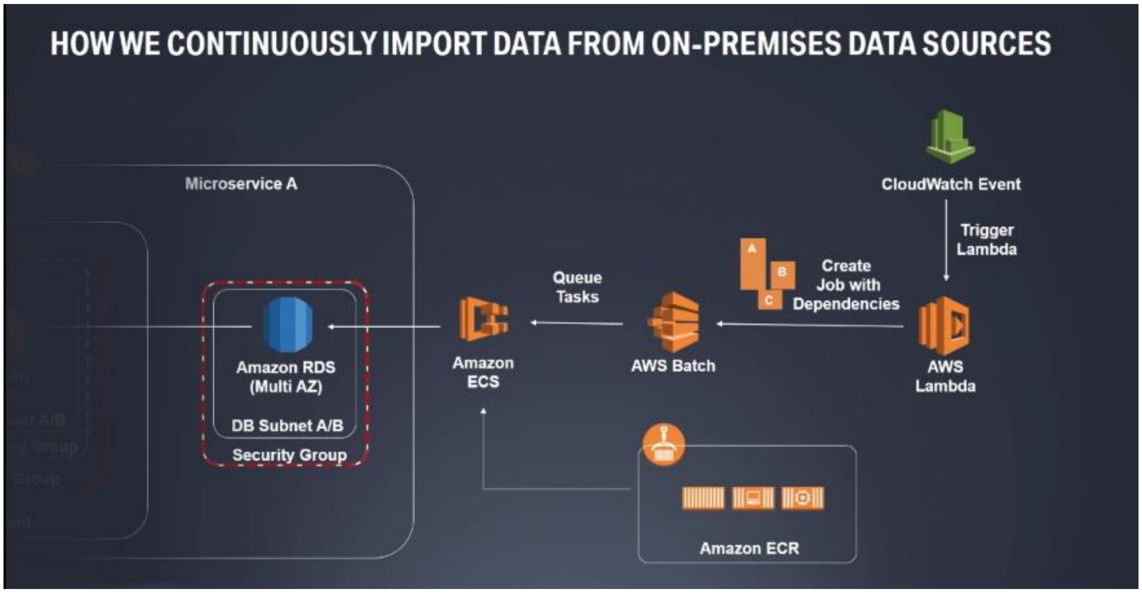


HOW WE CONTINUOUSLY IMPORT DATA FROM ON-PREMISES DATA SOURCES



HOW WE CONTINUOUSLY IMPORT DATA FROM ON-PREMISES DATA SOURCES

We use CloudWatch and Lambda services as a scheduler for triggering the right jobs at the right time in AWS Batch jobs.



HOW WE CONTINUOUSLY IMPORT DATA FROM ON-PREMISES DATA SOURCES

We also use this setup for some offline data processing that runs in Docker containers

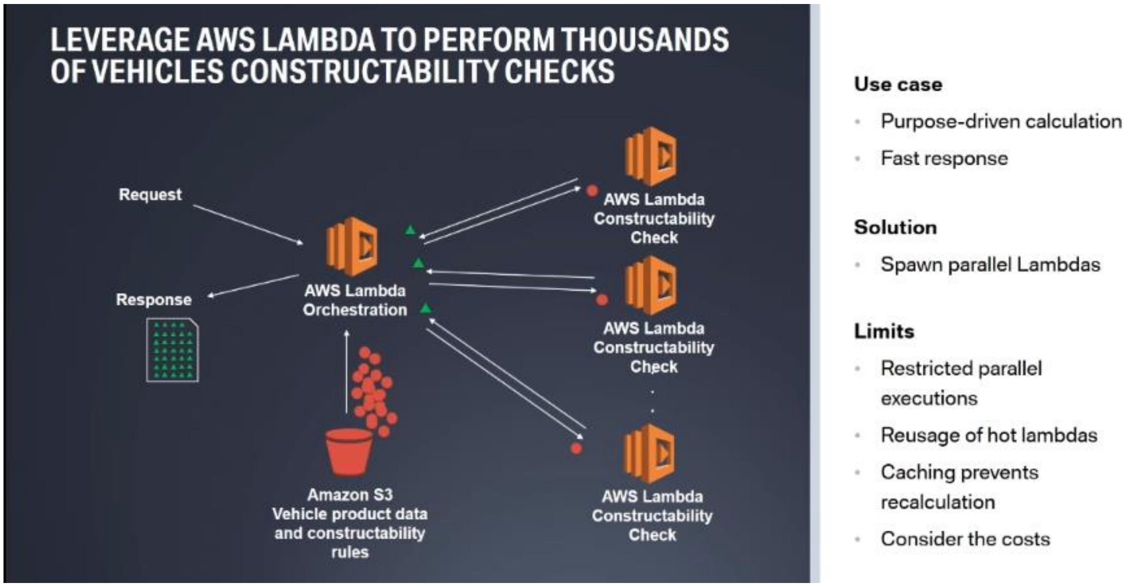
1 Introduction

2 Architecture overview

3 Architecture deep dives

- Service discovery
- AWS Lambda CI/CD
- Offline data processing
- **Advanced Lambda use case**
- Environments on demand

4 Conclusion



LEVERAGE AWS LAMBDA TO PERFORM THOUSANDS OF VEHICLES CONSTRUCTABILITY CHECKS

Request
Response
AWS Lambda Orchestration
Amazon S3 Vehicle product data and constructability rules
AWS Lambda Constructability Check
AWS Lambda Constructability Check
AWS Lambda Constructability Check

Use case

- Purpose-driven calculation
- Fast response

Solution

- Spawn parallel Lambdas

Limits

- Restricted parallel executions
- Reusage of hot lambdas
- Caching prevents recalculation
- Consider the costs

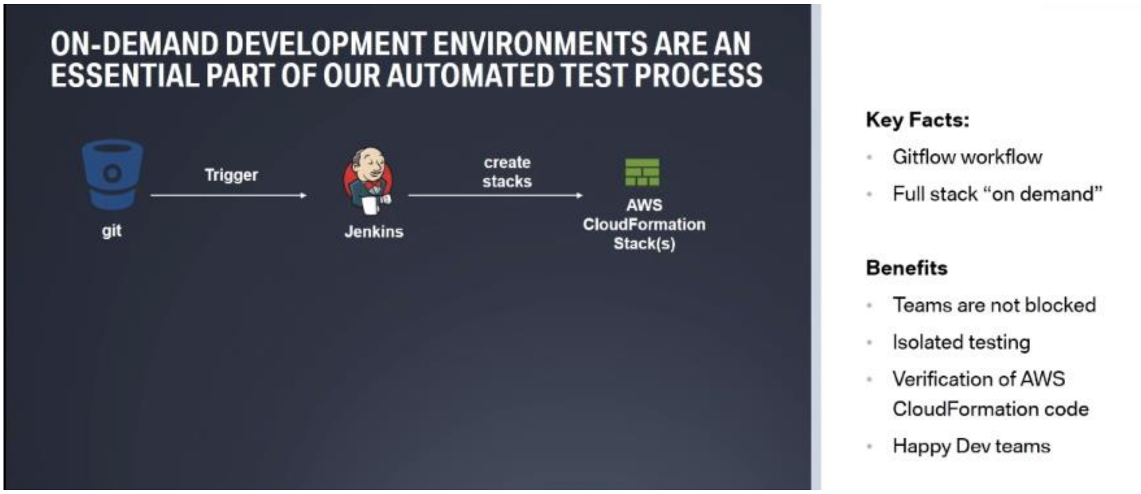
1 Introduction

2 Architecture overview

3 Architecture deep dives

- Service discovery
- AWS Lambda CI/CD
- Offline data processing
- Advanced Lambda use case
- **Environments on demand**

4 Conclusion



ON-DEMAND DEVELOPMENT ENVIRONMENTS ARE AN ESSENTIAL PART OF OUR AUTOMATED TEST PROCESS

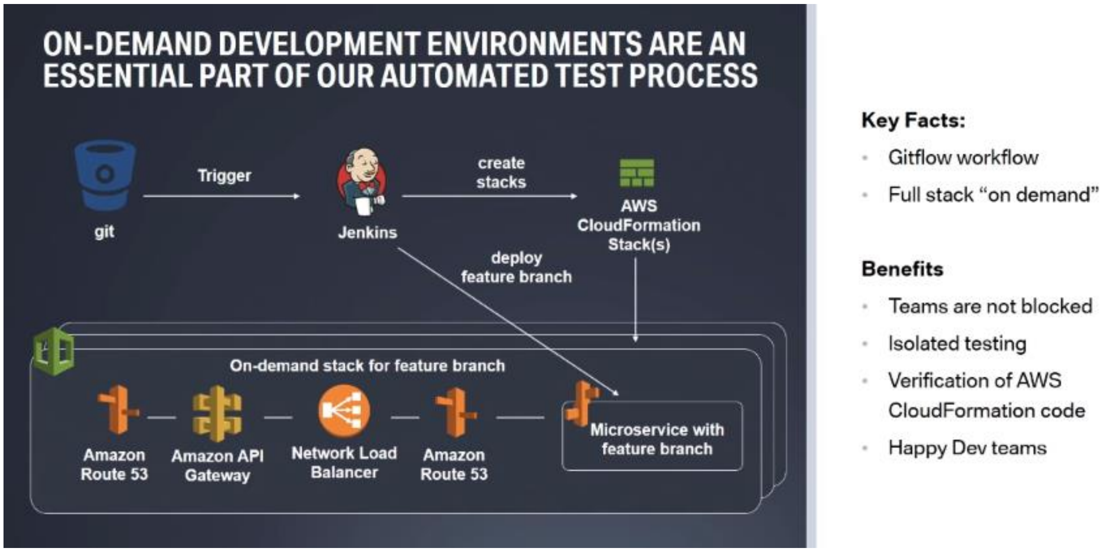
Key Facts:

- Gitflow workflow
- Full stack "on demand"

Benefits

- Teams are not blocked
- Isolated testing
- Verification of AWS CloudFormation code
- Happy Dev teams

We define our infrastructure using CloudFormation as infrastructure as code.



ON-DEMAND DEVELOPMENT ENVIRONMENTS ARE AN ESSENTIAL PART OF OUR AUTOMATED TEST PROCESS

Key Facts:

- Gitflow workflow
- Full stack "on demand"

Benefits

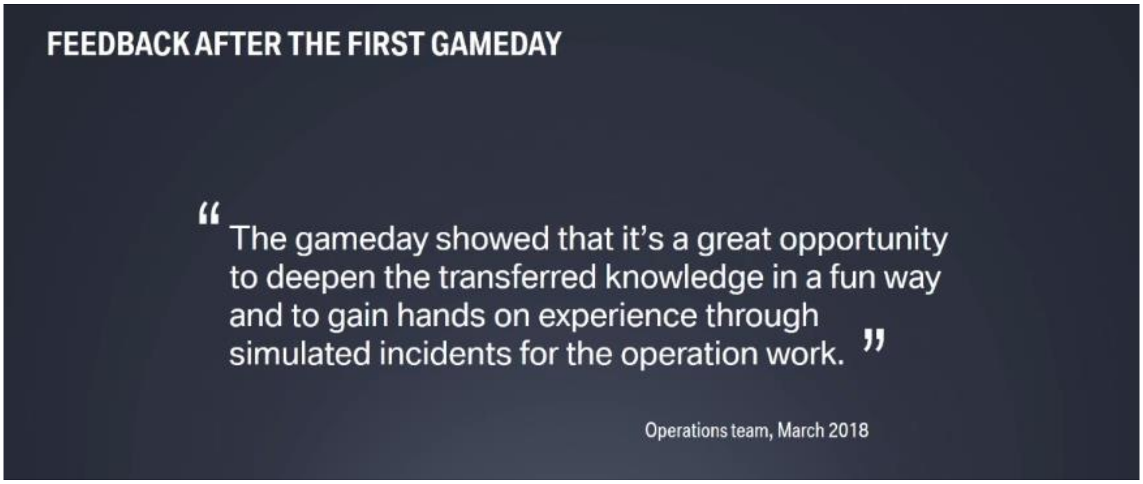
- Teams are not blocked
- Isolated testing
- Verification of AWS CloudFormation code
- Happy Dev teams

Jenkins pulls of CF out if the Git repo and creates the microservice stack that contains all that is needed. Jenkins then deploys the microservice into the feature-branch, then the team can run the CI pipeline to do manual testing. The on-demand environment then gets terminated after some fixed time period.

1. Introduction
2. Architecture overview
3. Architecture deep dives
4. **Conclusion**

WE INTRODUCED GAMEDAYS TO TRAIN OUR OPERATION TEAM TO BE WELL PREPARED FOR POTENTIAL PRODUCTION PROBLEMS

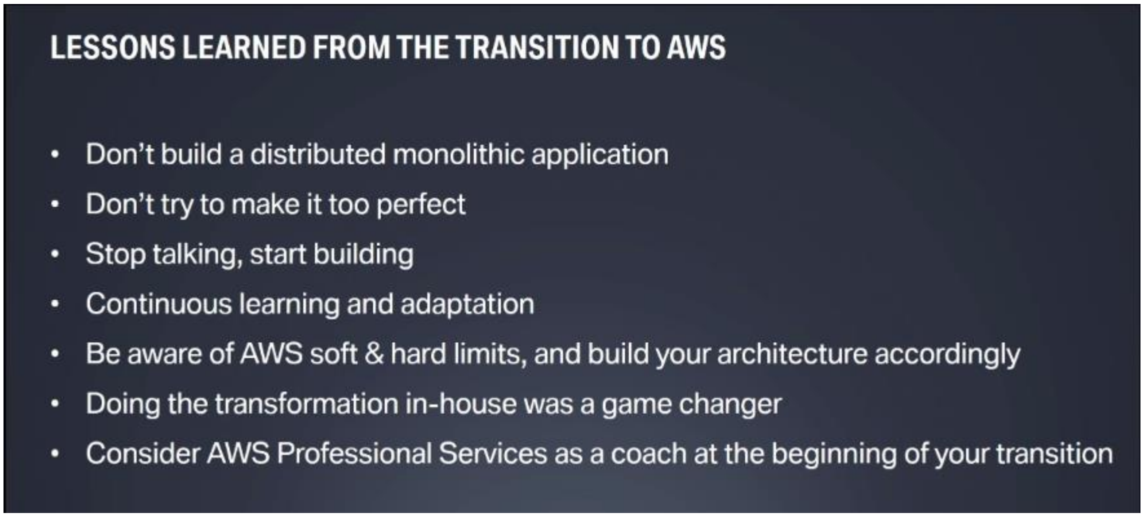
- Solve real production scenarios
- Gamification is used through scoring and leaderboards
- Chaos Monkey Team brings fun into the situation
- Each team gains points for
  - Availability of the system
  - Proposed improvements for DevOps-Cycle



FEEDBACK AFTER THE FIRST GAMEDAY

> “The gameday showed that it’s a great opportunity to deepen the transferred knowledge in a fun way and to gain hands on experience through simulated incidents for the operation work.”

Operations team, March 2018

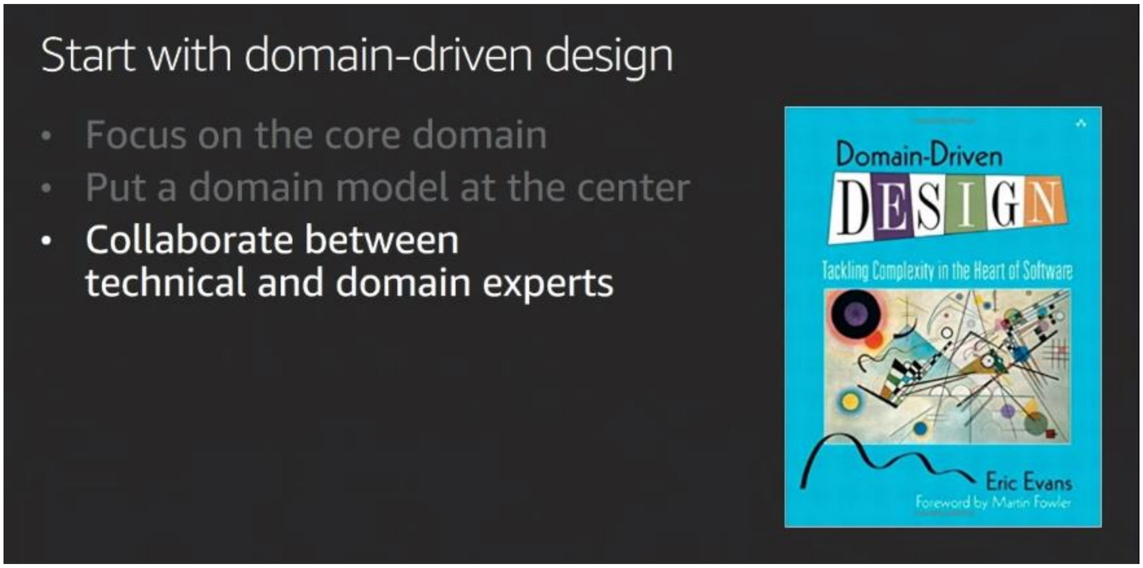


LESSONS LEARNED FROM THE TRANSITION TO AWS

- Don’t build a distributed monolithic application
- Don’t try to make it too perfect
- Stop talking, start building
- Continuous learning and adaptation
- Be aware of AWS soft & hard limits, and build your architecture accordingly
- Doing the transformation in-house was a game changer
- Consider AWS Professional Services as a coach at the beginning of your transition

Your turn

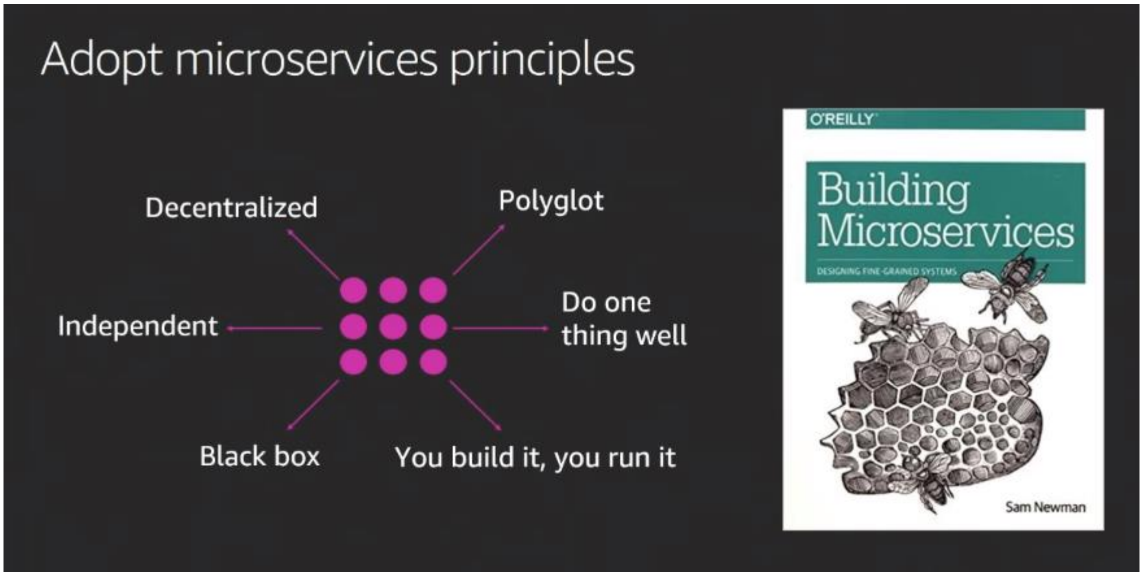
Best practices



Start with domain-driven design

- Focus on the core domain
- Put a domain model at the center
- **Collaborate between technical and domain experts**

Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler



Adopt microservices principles

Decentralized

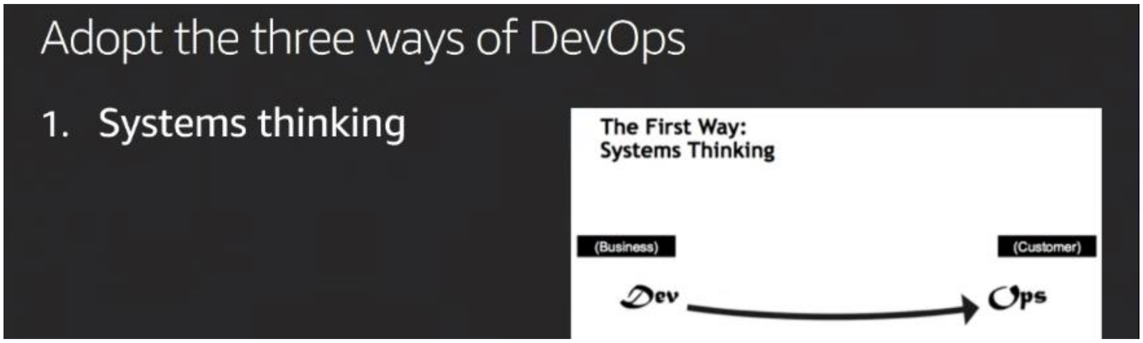
Polyglot

Independent

Do one thing well

Black box

You build it, you run it



Adopt the three ways of DevOps

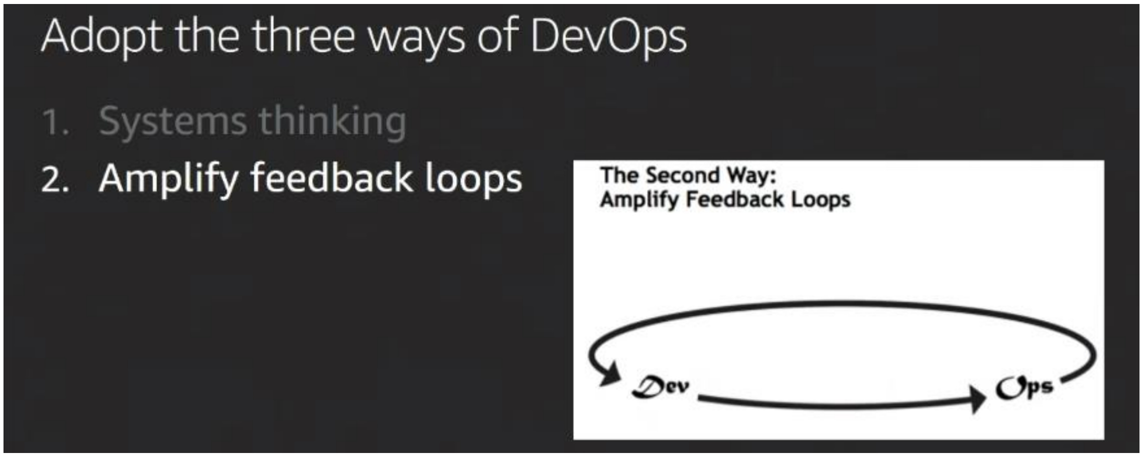
1. Systems thinking

The First Way:
Systems Thinking

(Business)

(Customer)

Dev → Ops

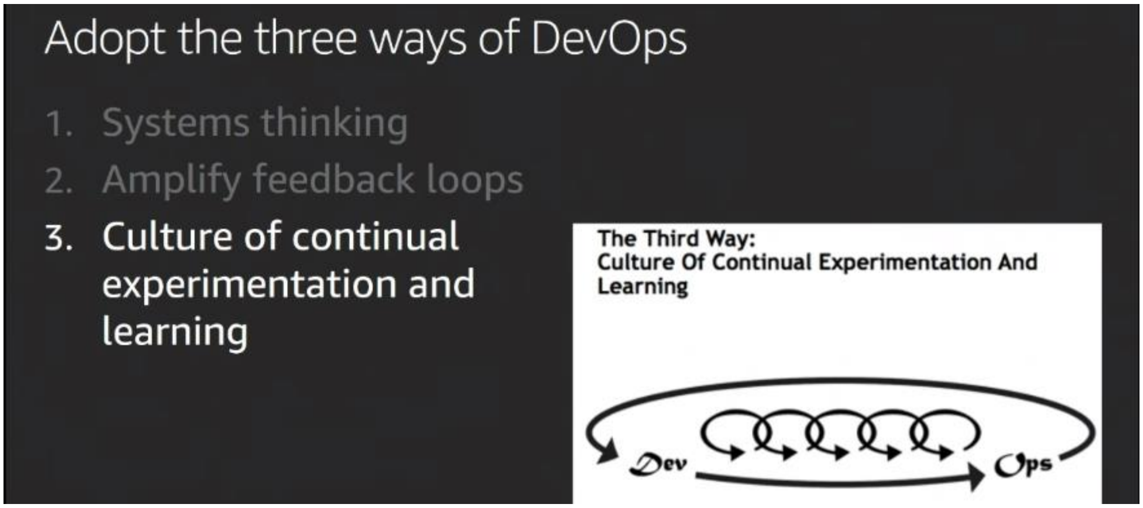


Adopt the three ways of DevOps

1. Systems thinking
2. Amplify feedback loops

The Second Way:
Amplify Feedback Loops

Dev → Ops

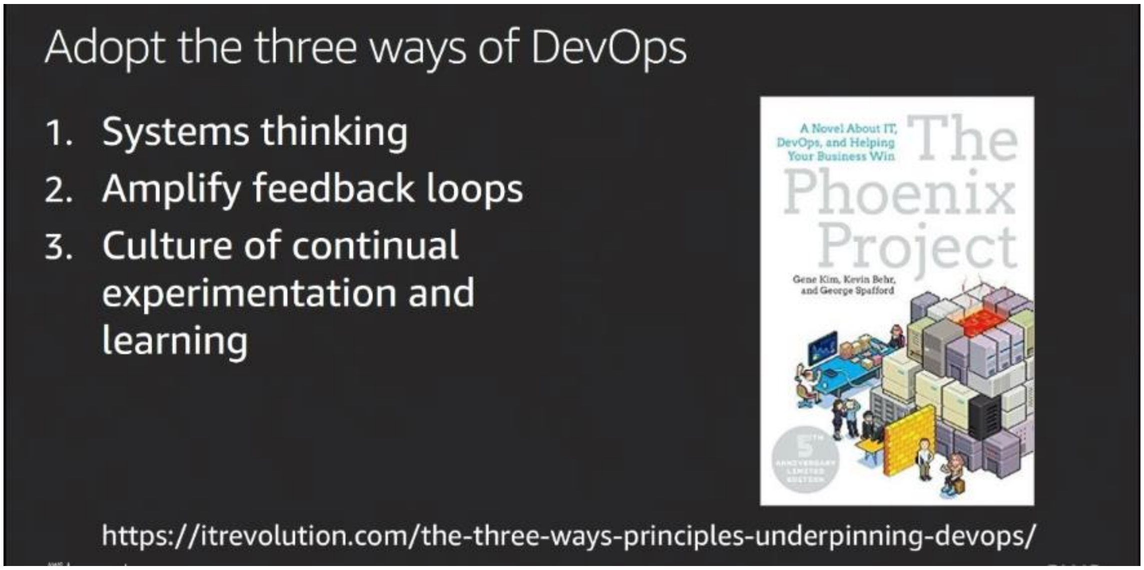


Adopt the three ways of DevOps

1. Systems thinking
2. Amplify feedback loops
3. Culture of continual experimentation and learning

The Third Way:
Culture Of Continual Experimentation And Learning

Dev → Ops

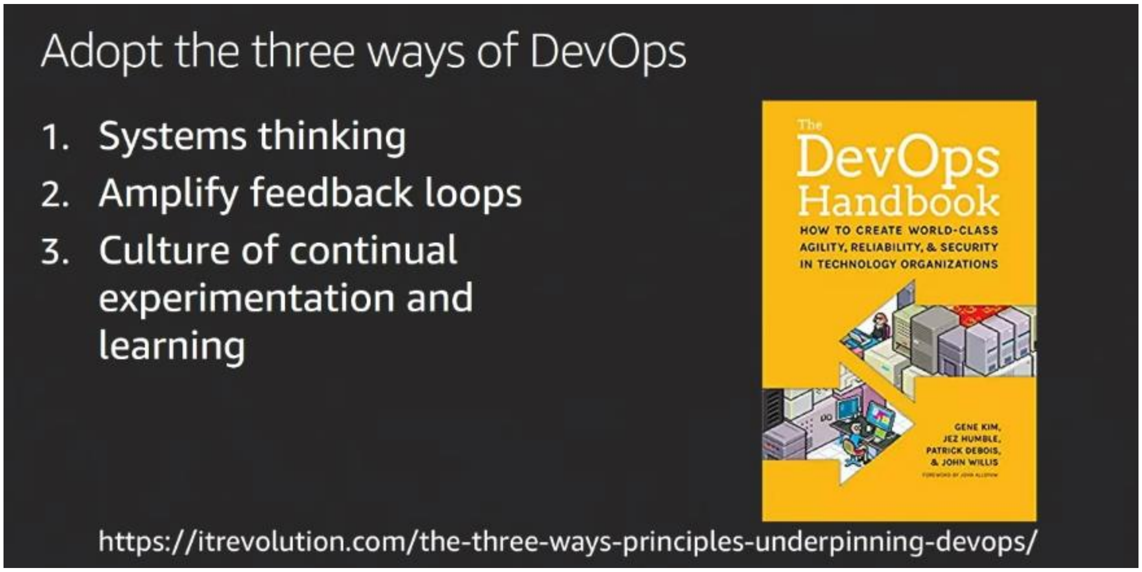


Adopt the three ways of DevOps

1. Systems thinking
2. Amplify feedback loops
3. Culture of continual experimentation and learning

A Novel About IT, DevOps, and Helping Your Business Win
The Phoenix Project
Gene Kim, Kevin Behr, and George Spafford
5th Anniversary Limited Edition

https://itrevolution.com/the-three-ways-principles-underpinning-devops/

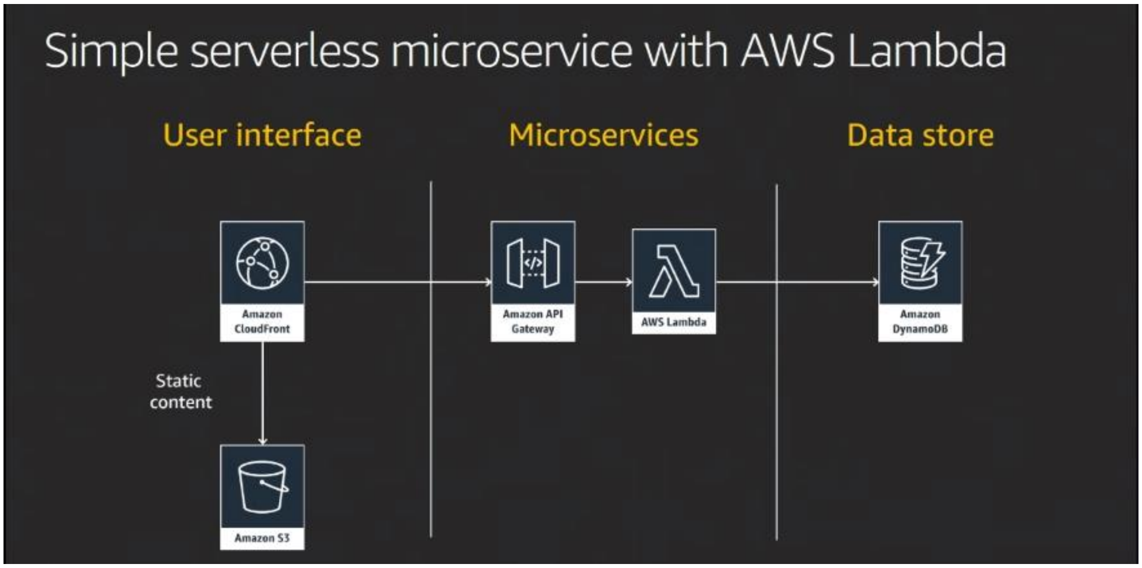


Adopt the three ways of DevOps

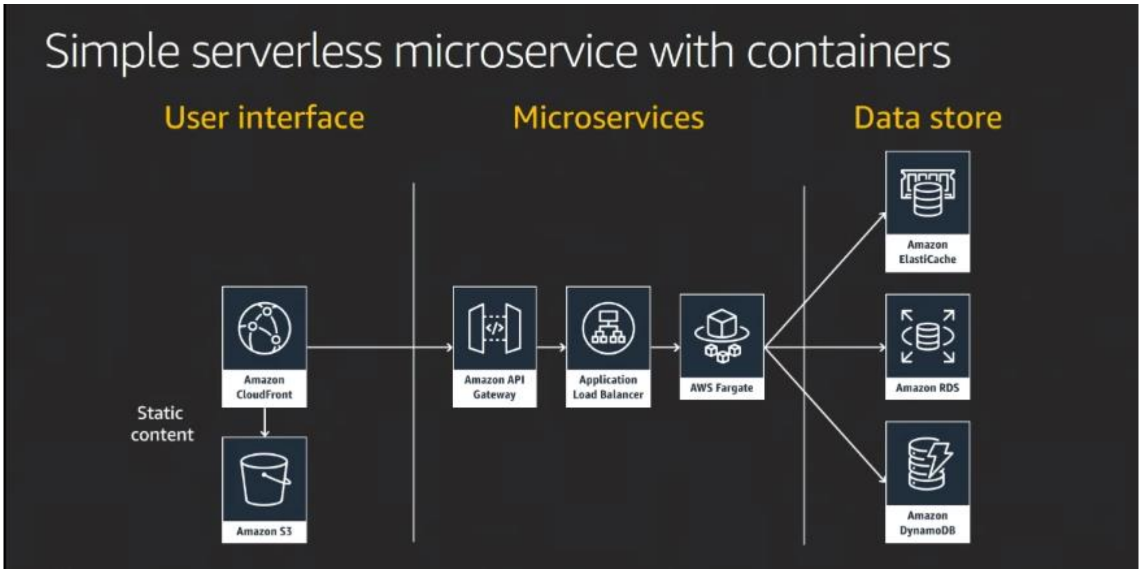
1. Systems thinking
2. Amplify feedback loops
3. Culture of continual experimentation and learning

The
DevOps
Handbook
HOW TO CREATE WORLD-CLASS
AGILITY, RELIABILITY, & SECURITY
IN TECHNOLOGY ORGANIZATIONS
GENE KIM,
JEZ HUMBLE,
PATRICK DEBOIS,
& JOHN WILLIS

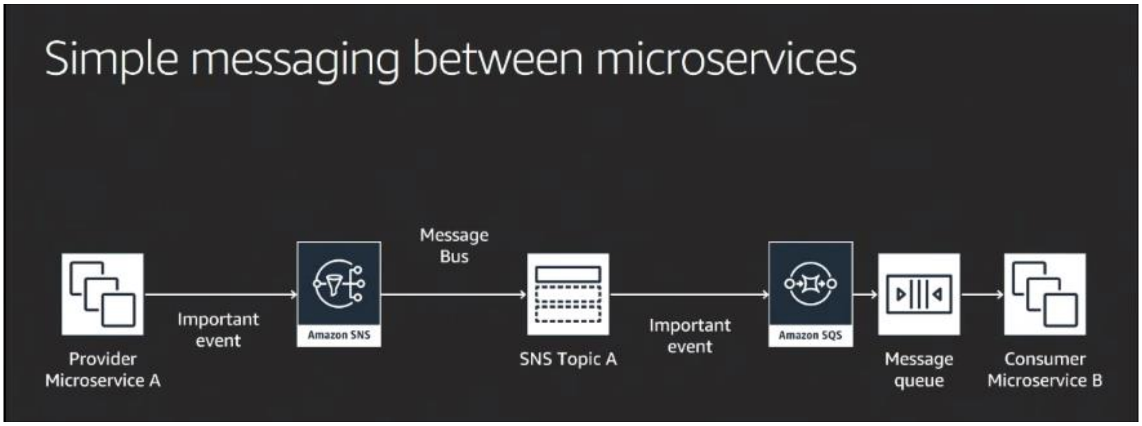
https://itrevolution.com/the-three-ways-principles-underpinning-devops/



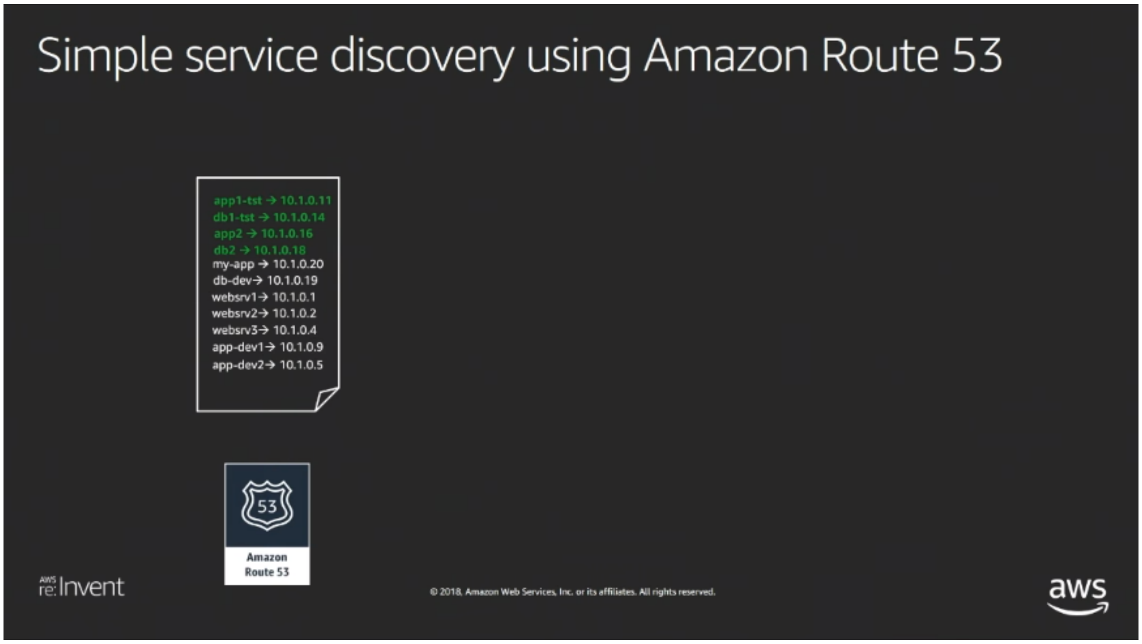
Simple serverless microservice with AWS Lambda



Simple serverless microservice with containers



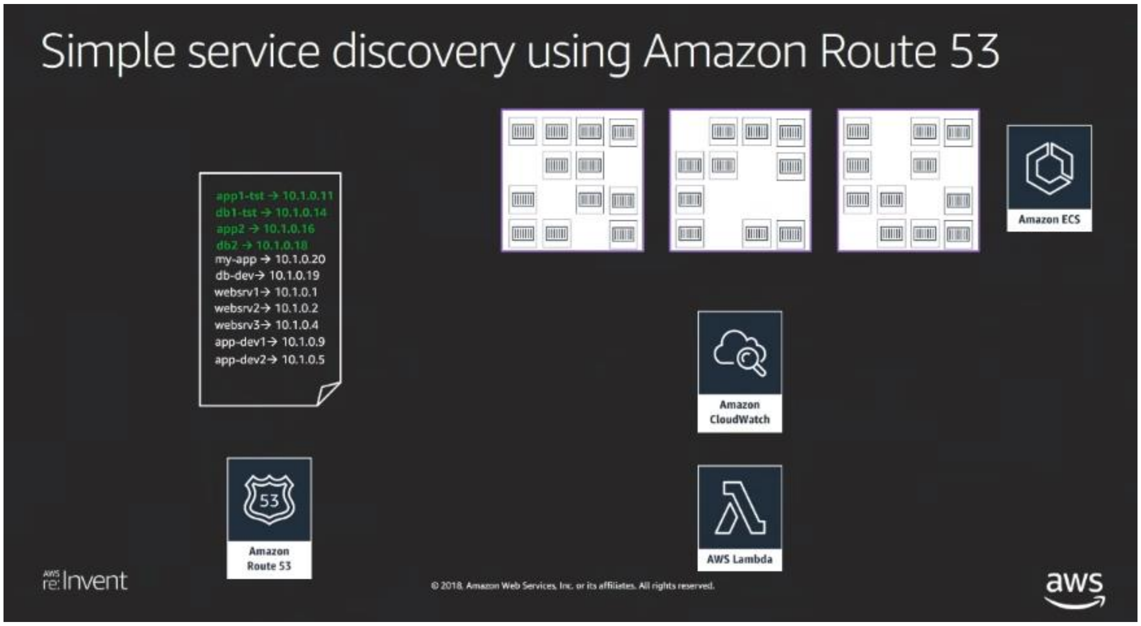
Simple messaging between microservices



Simple service discovery using Amazon Route 53

app1-tst → 10.1.0.11
db1-tst → 10.1.0.14
app2 → 10.1.0.16
db2 → 10.1.0.18
my-app → 10.1.0.20
db-dev→ 10.1.0.19
websrv1→ 10.1.0.1
websrv2→ 10.1.0.2
websrv3→ 10.1.0.4
app-dev1→ 10.1.0.9
app-dev2→ 10.1.0.5

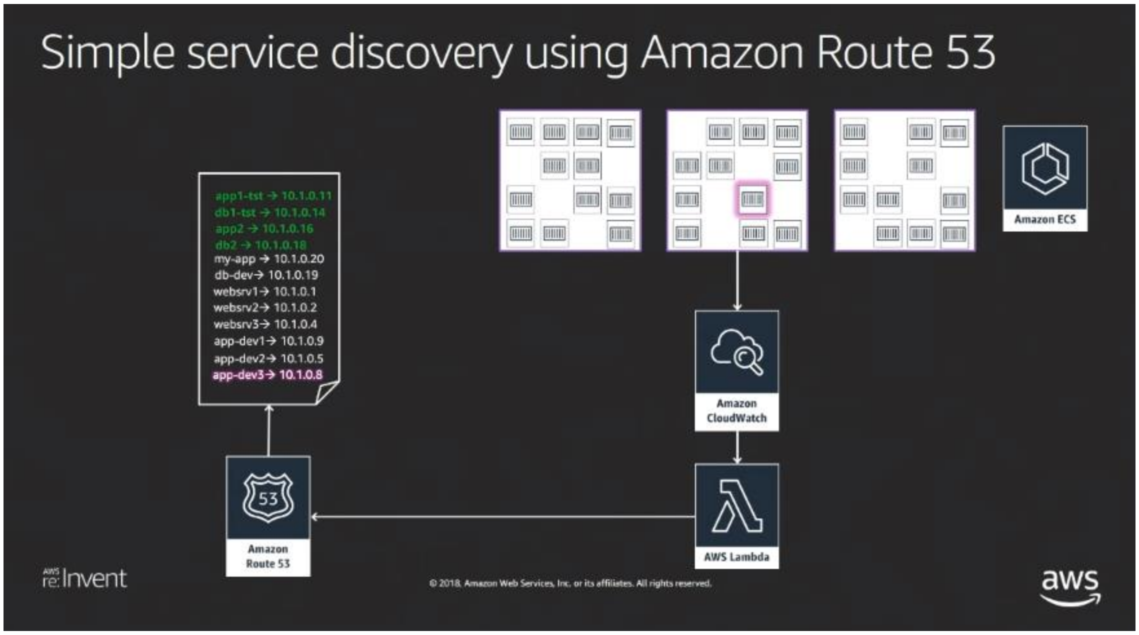
You can use Route53 DNS records to do the mapping between your service name and the IP address where your service is available.



Simple service discovery using Amazon Route 53

app1-tst → 10.1.0.11
db1-tst → 10.1.0.14
app2 → 10.1.0.16
db2 → 10.1.0.18
my-app → 10.1.0.20
db-dev→ 10.1.0.19
websrv1→ 10.1.0.1
websrv2→ 10.1.0.2
websrv3→ 10.1.0.4
app-dev1→ 10.1.0.9
app-dev2→ 10.1.0.5

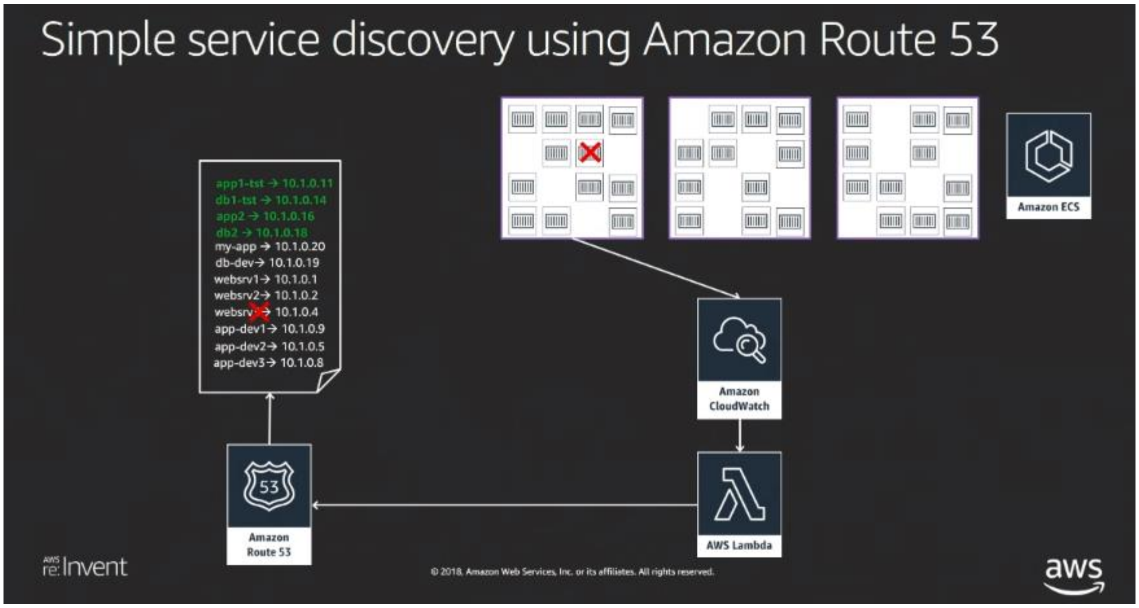
You can then set up CloudWatch to monitor the lifecycle of the individual containers running in your container fleet and you can use AWS Lambda as a dynamic way of updating the Route53 records.



Simple service discovery using Amazon Route 53

app1-tst → 10.1.0.11
db1-tst → 10.1.0.14
app2 → 10.1.0.16
db2 → 10.1.0.18
my-app → 10.1.0.20
db-dev→ 10.1.0.19
websrv1→ 10.1.0.1
websrv2→ 10.1.0.2
websrv3→ 10.1.0.4
app-dev1→ 10.1.0.9
app-dev2→ 10.1.0.5
app-dev3→ 10.1.0.8

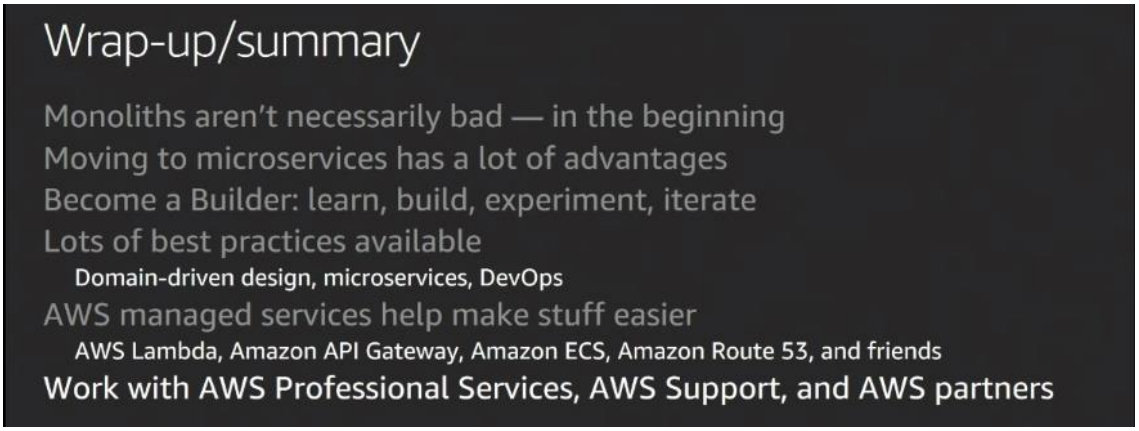
If you start a new microservice container in your container fleet, CloudWatch will automatically detect that a new container is running. It will notify AWS Lambda and then you can use code to update the record in Route53 so that the new entry shows up in Route53 and can serve requests.



Simple service discovery using Amazon Route 53

app1-tst → 10.1.0.11
db1-tst → 10.1.0.14
app2 → 10.1.0.16
db2 → 10.1.0.18
my-app → 10.1.0.20
db-dev→ 10.1.0.19
websrv1→ 10.1.0.1
websrv2→ 10.1.0.2
websrv→ 10.1.0.4
app-dev1→ 10.1.0.9
app-dev2→ 10.1.0.5
app-dev3→ 10.1.0.8

The same process takes place when a microservice container is down.

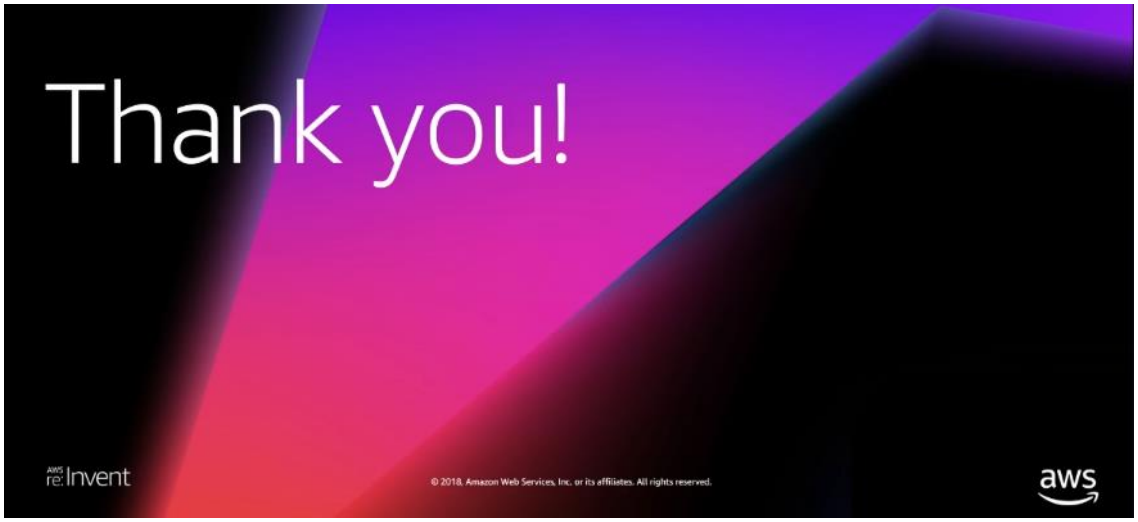


Wrap-up/summary

Monoliths aren't necessarily bad — in the beginning
Moving to microservices has a lot of advantages
Become a Builder: learn, build, experiment, iterate
Lots of best practices available
- Domain-driven design, microservices, DevOps

AWS managed services help make stuff easier
- AWS Lambda, Amazon API Gateway, Amazon ECS, Amazon Route 53, and friends

Work with AWS Professional Services, AWS Support, and AWS partners



Thank you!

AWS re:Invent

aws