AWS Serverless

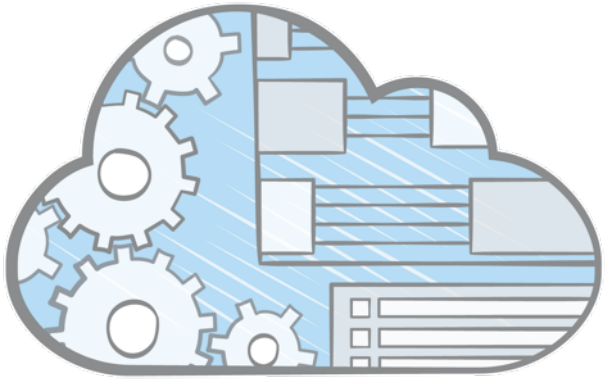
TV2 Workshop

Tobias Börjeson, borjeson@amazon.com

6 March, 2017

amazon web services

Agenda



Technical Workshop

- Intro
- AWS Technical Overview
- AWS Account Strategies
- Serverless Architectures
- CI/CD Pipelines
- Example Project

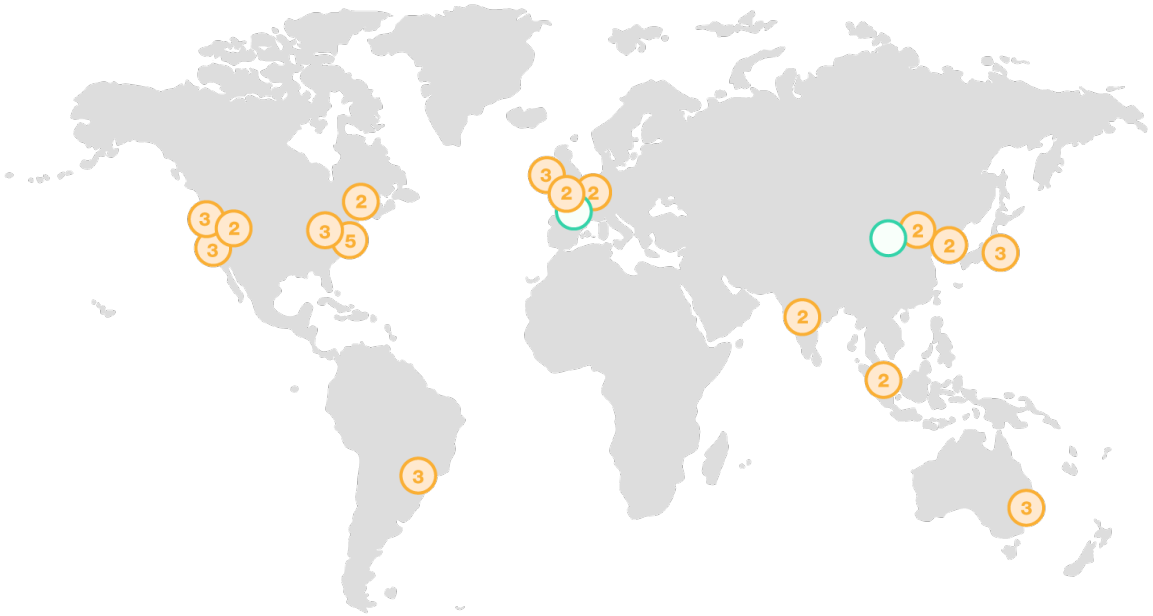
Objectives: Provide understanding of Serverless architectures on AWS, with focus on best practices and examples from a Java perspective

AWS Technical Overview

amazon web services

AWS Global Infrastructure

16 Regions – **42** Availability Zones – **68** Edge Locations



Region & Number of Availability Zones

AWS GovCloud (2)

US West

Oregon (3)

Northern California (3)

US East

N. Virginia (5), Ohio (3)

Canada

Central (2)

South America

São Paulo (3)

EU

Ireland (3)

Frankfurt (2)

London (2)

Asia Pacific

Singapore (2)

Sydney (2), Tokyo (3),

Seoul (2), Mumbai (2)

China

Beijing (2)

Announced Regions

Paris, Ningxia

amazon web services

Europe / Middle East / Africa

EU (Ireland) Region

EC2 Availability Zones: 3

EU (Frankfurt) Region

EC2 Availability Zones: 2

EU (London) Region

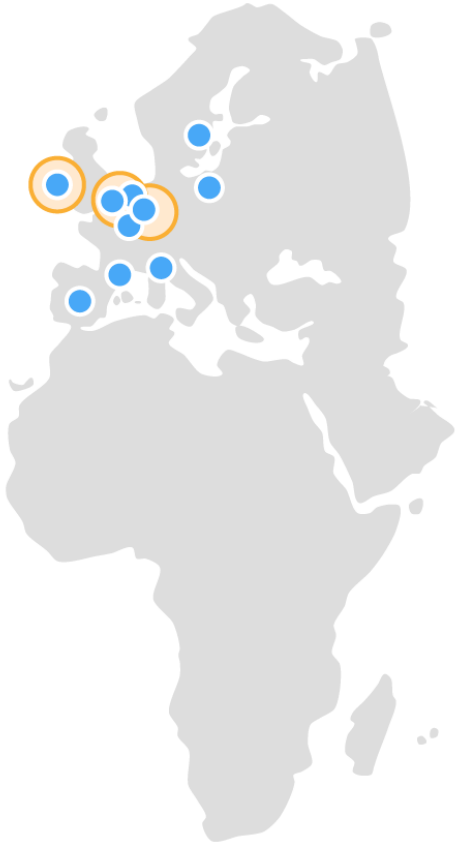
EC2 Availability Zones: 2

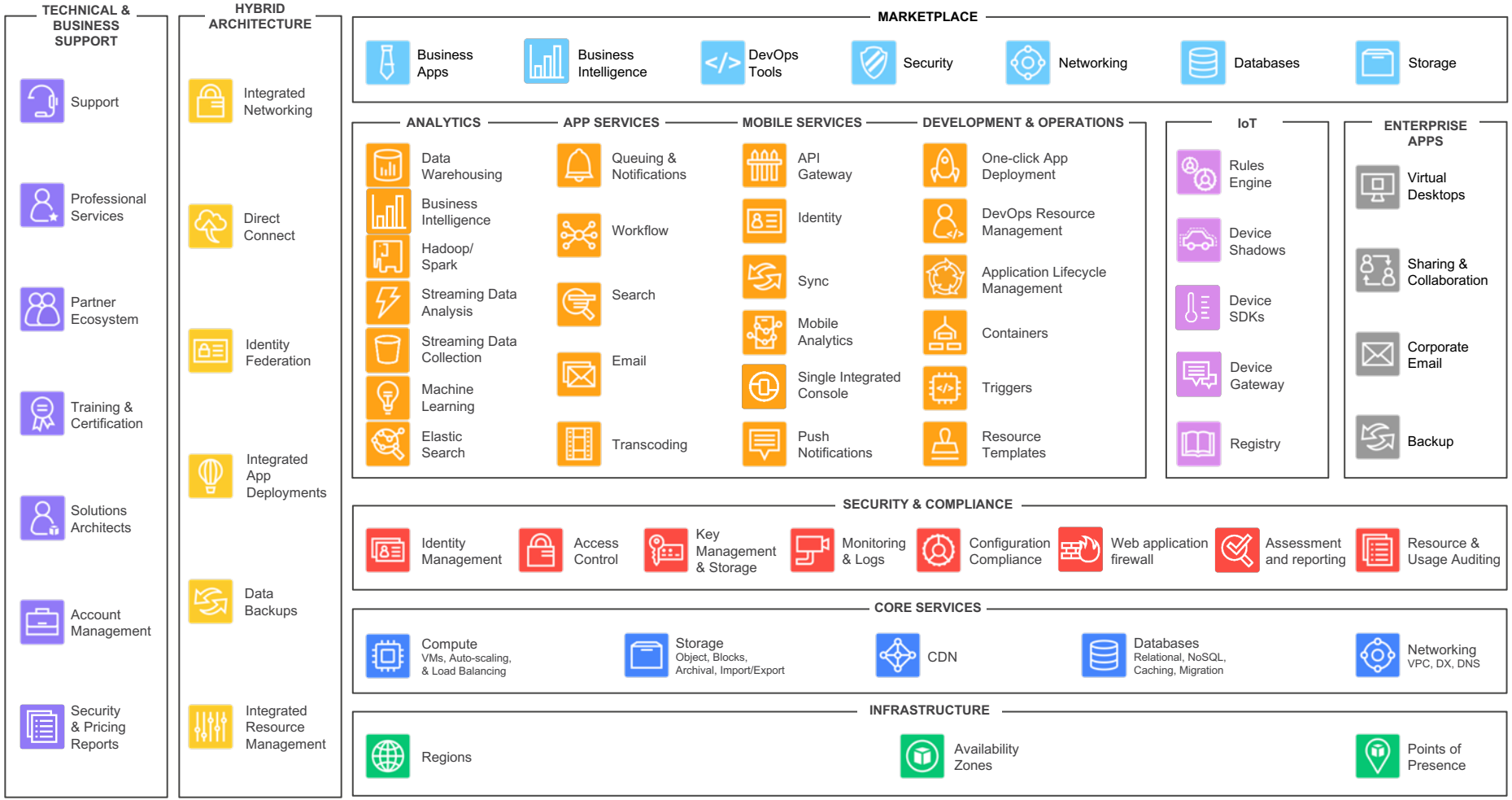
EU (Paris) Region

Announced

AWS Edge Locations

Amsterdam, The Netherlands (2), Berlin, Germany, Dublin, Ireland, Frankfurt, Germany (5), London, England (4), Madrid, Spain, Marseille, France, Milan, Italy, Paris, France (2), Stockholm, Sweden, and Warsaw, Poland




TECHNICAL & BUSINESS SUPPORT
Support
Professional Services
Partner Ecosystem
Training & Certification
Solutions Architects
Account Management
Security & Pricing Reports
HYBRID ARCHITECTURE
Integrated Networking
Direct Connect
Identity Federation
Integrated App Deployments
Data Backups
Integrated Resource Management
MARKETPLACE
Business Apps
Business Intelligence
DevOps Tools
Security
Networking
Databases
Storage
ANALYTICS
Data Warehousing
Business Intelligence
Hadoop/ Spark
Streaming Data Analysis
Streaming Data Collection
Machine Learning
Elastic Search
APP SERVICES
Queuing & Notifications
Workflow
Search
Email
Transcoding
MOBILE SERVICES
API Gateway
Identity
Sync
Mobile Analytics
Single Integrated Console
Push Notifications
DEVELOPMENT & OPERATIONS
One-click App Deployment
DevOps Resource Management
Application Lifecycle Management
Containers
Triggers
Resource Templates
IoT
Rules Engine
Device Shadows
Device SDKs
Device Gateway
Registry
ENTERPRISE APPS
Virtual Desktops
Sharing & Collaboration
Corporate Email
Backup
SECURITY & COMPLIANCE
Identity Management
Access Control
Key Management & Storage
Monitoring & Logs
Configuration Compliance
Web application firewall
Assessment and reporting
Resource & Usage Auditing
CORE SERVICES
Compute
VMs, Auto-scaling, & Load Balancing
Storage
Object, Blocks, Archival, Import/Export
CDN
Databases
Relational, NoSQL, Caching, Migration
Networking
VPC, DX, DNS
INFRASTRUCTURE
Regions
Availability Zones
Points of Presence

Spectrum of AWS offerings

"On EC2"

Amazon EC2

Microsoft SQL Server

Managed

Amazon EMR

Amazon Elasticsearch Service

Amazon ElastiCache

Amazon Redshift

Amazon RDS

Serverless

AWS Lambda

Amazon Cognito

Amazon Kinesis

Amazon S3

Amazon DynamoDB

Amazon SQS

Amazon API Gateway

Amazon CloudWatch

AWS IoT

amazon web services

AWS building blocks

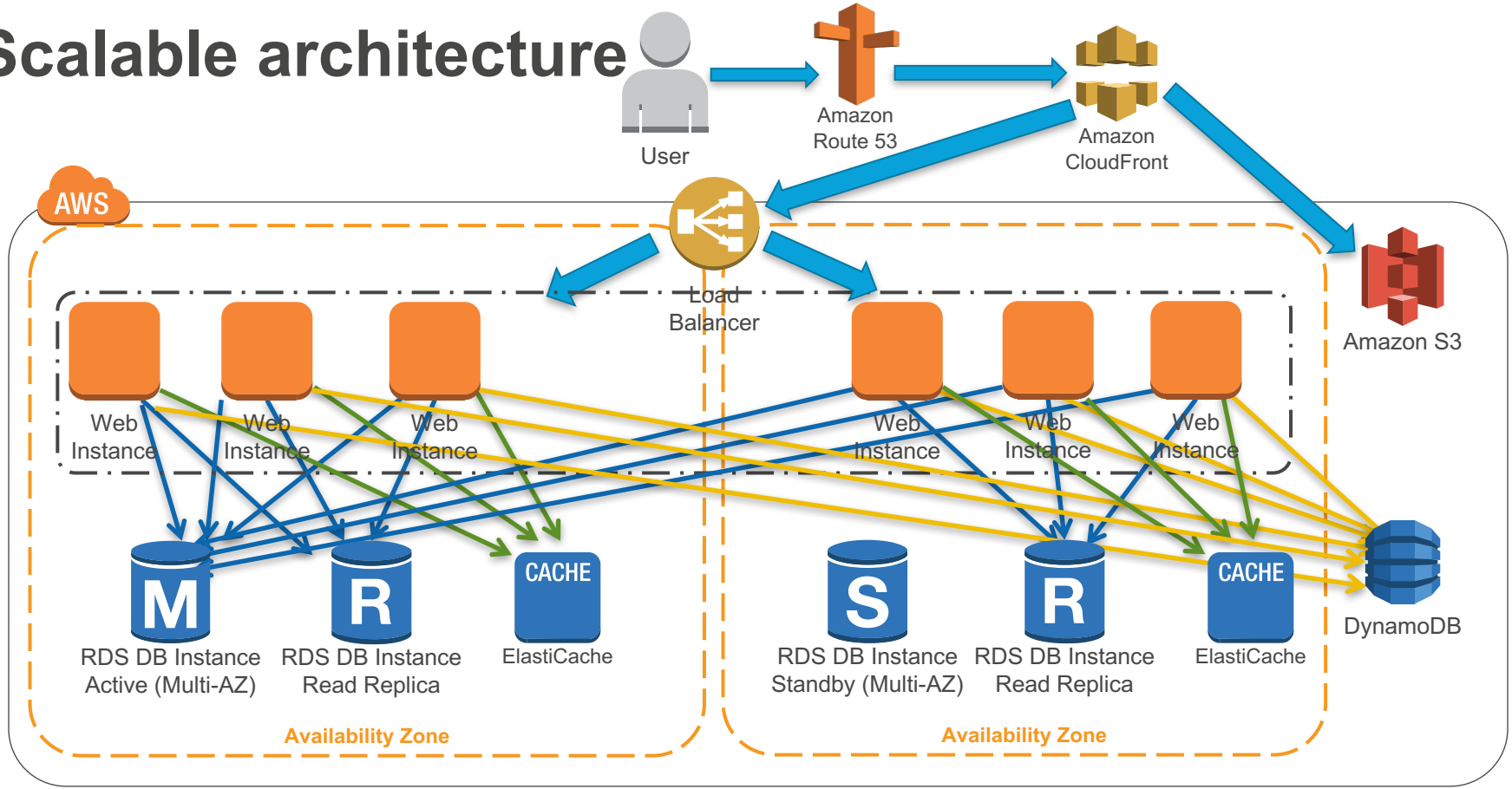
Inherently highly available and fault-tolerant services

- ✔ Amazon CloudFront
- ✔ Amazon Route 53
- ✔ Amazon S3
- ✔ Amazon DynamoDB
- ✔ Elastic Load Balancing
- ✔ Amazon EFS
- ✔ AWS Lambda
- ✔ Amazon SQS
- ✔ Amazon SNS
- ✔ Amazon SES
- ✔ Amazon SWF
- ✔ …

Highly available with the right architecture

- Amazon EC2
- Amazon EBS
- Amazon RDS
- Amazon VPC

Scalable architecture



amazon web services

AWS application management solutions

Higher-level services

Do it yourself

AWS
Elastic Beanstalk

AWS
OpsWorks

AWS
CloudFormation

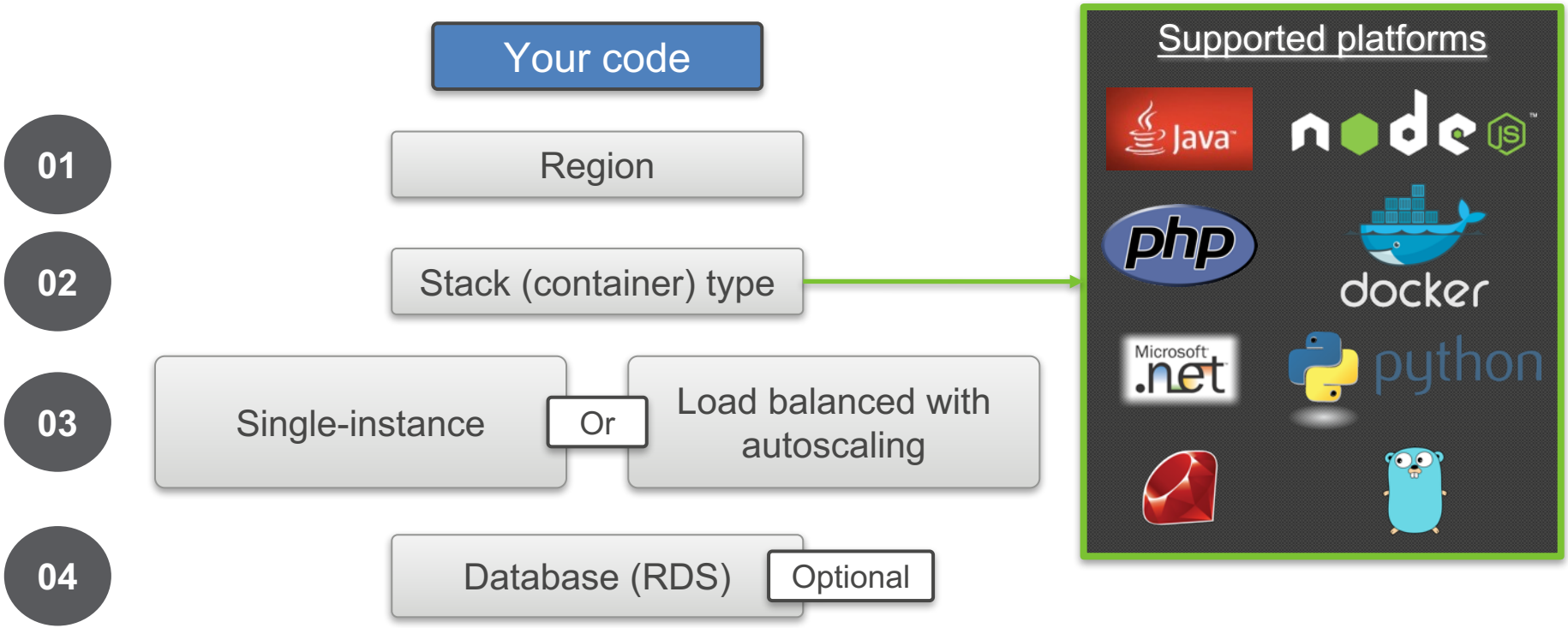
Amazon EC2

Convenience

Control

amazon web services

AWS Elastic Beanstalk



Loose coupling = winning

DON’T REINVENT THE WHEEL

- Email
- Queuing
- Transcoding
- Search
- Databases
- Monitoring
- Metrics
- Logging
- Compute


AWS Lambda


Amazon SNS


Amazon SES


Amazon ElasticSearch


Amazon SWF


Amazon SQS

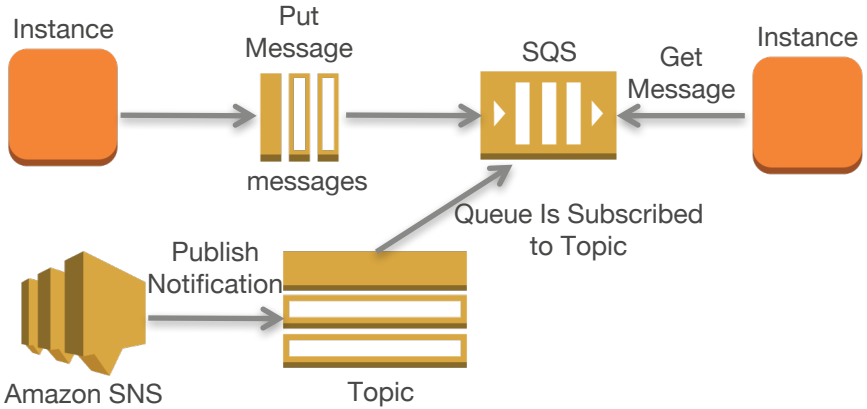

Amazon Elastic Transcoder

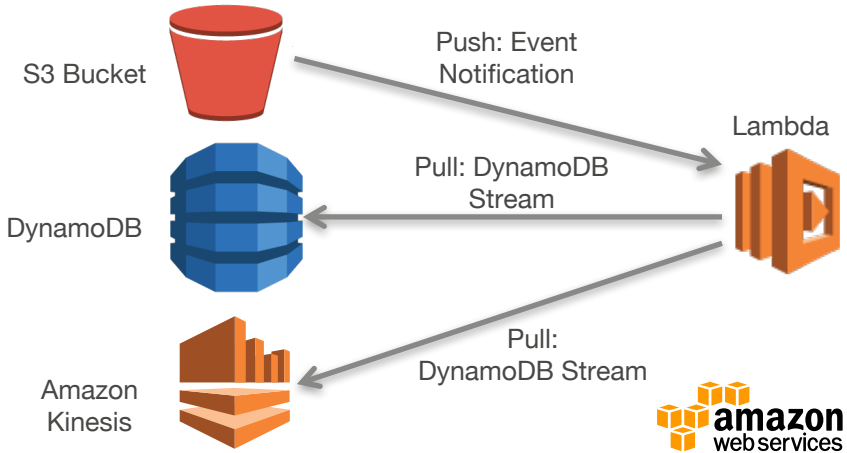
amazon web services

Loose coupling sets you free!

The looser they're coupled, the bigger they scale

- Independent components
- Design everything as a black box
- Decouple interactions
- Favor services with built-in redundancy and scalability rather than building your own





Serverless patterns built with functions

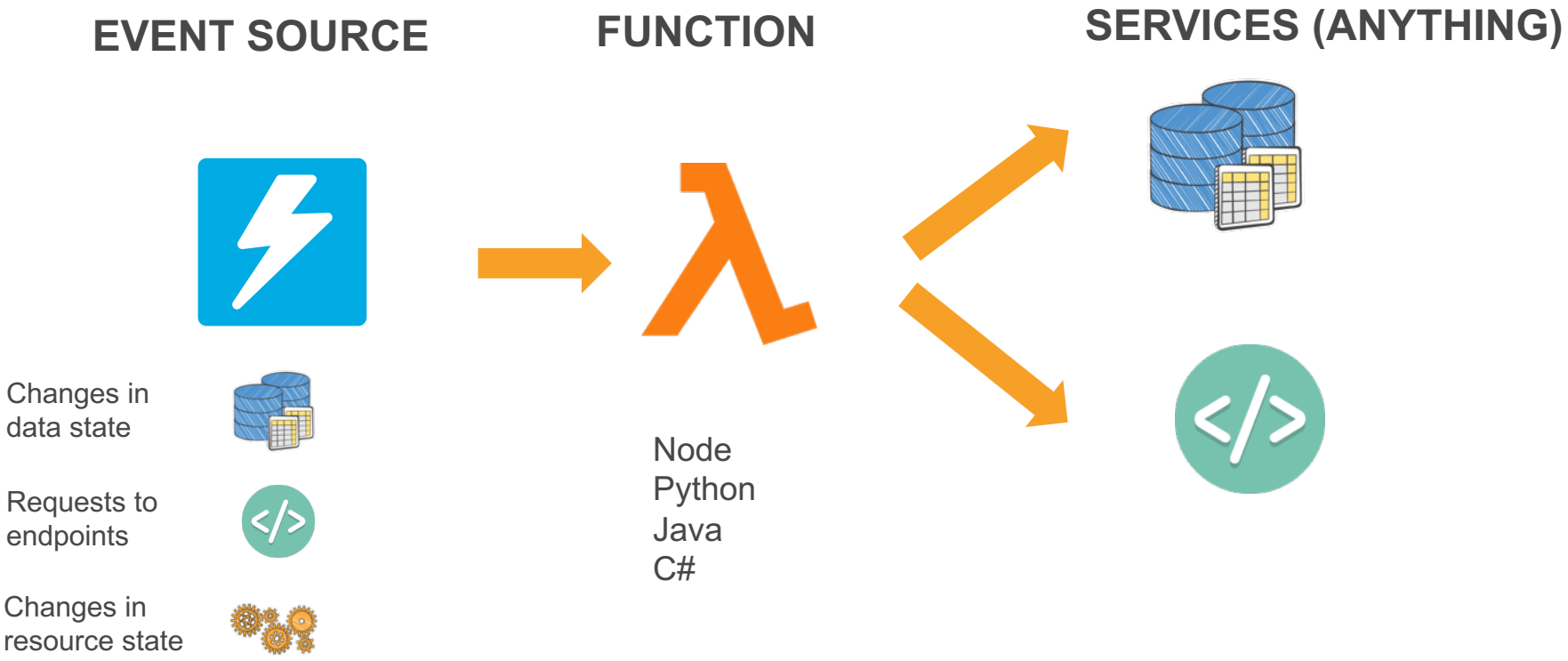
Functions are the unit of deployment and scale

Scales per request - users cannot over or under-provision

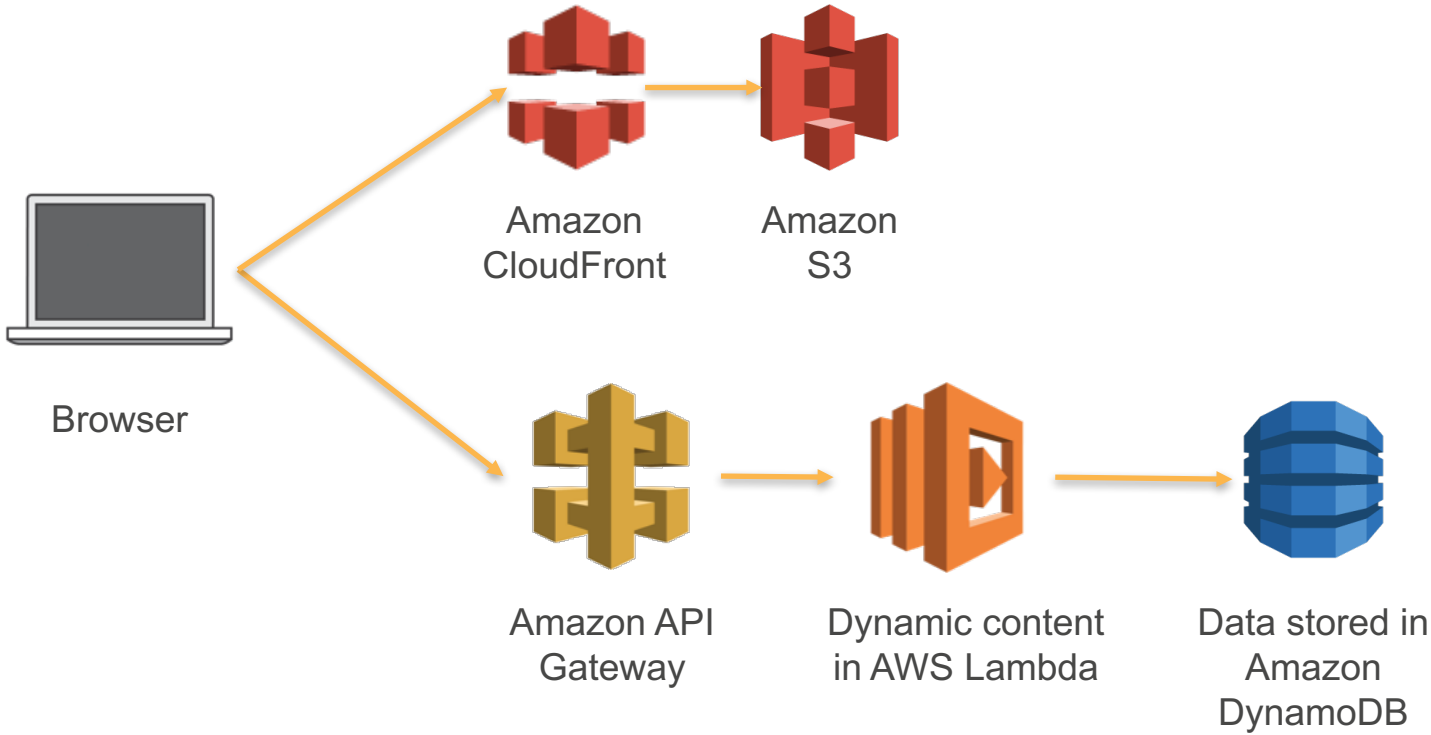
Never pay for idle

Skip the boring parts; skip the hard parts

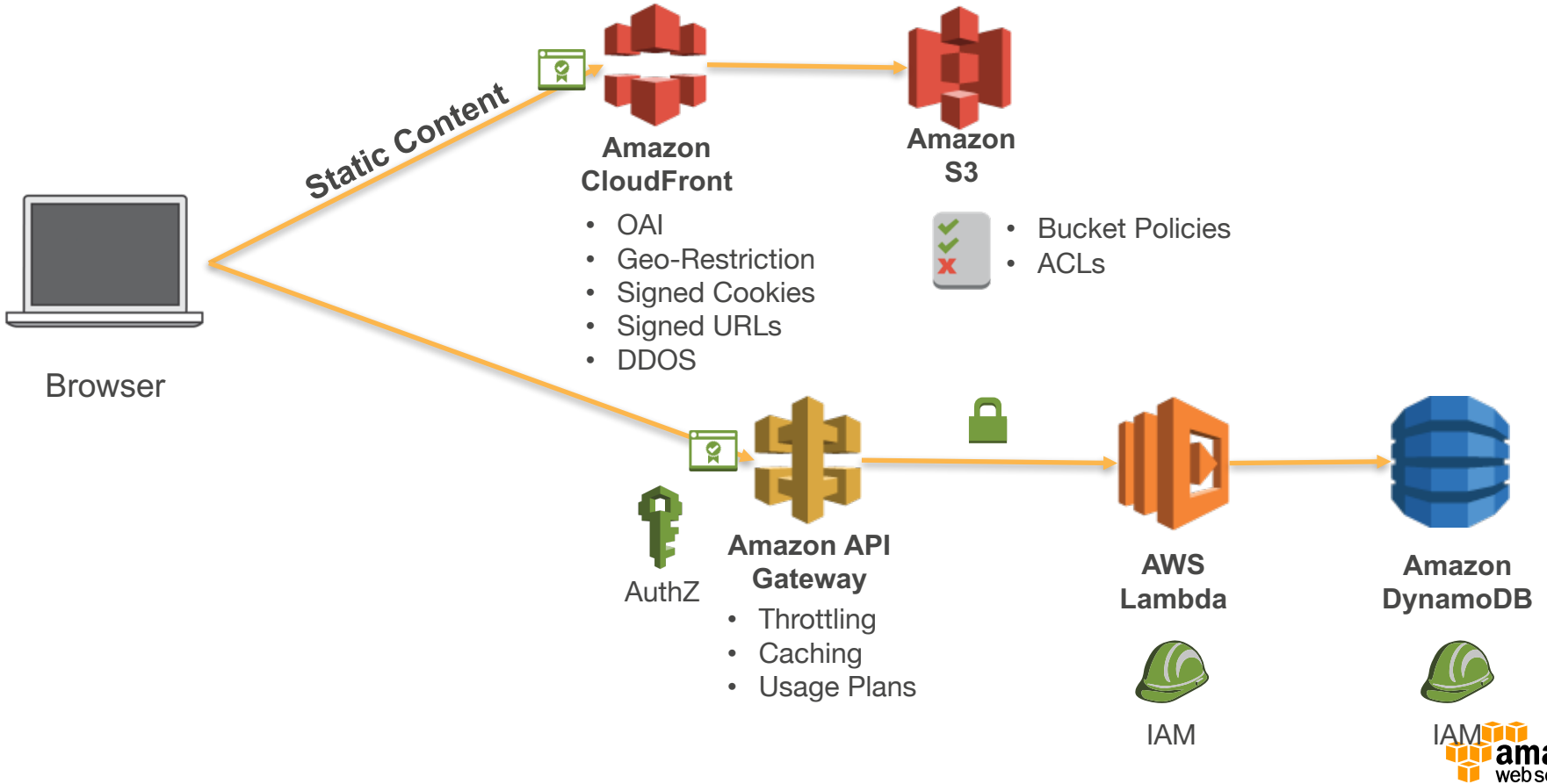
Working with AWS Lambda



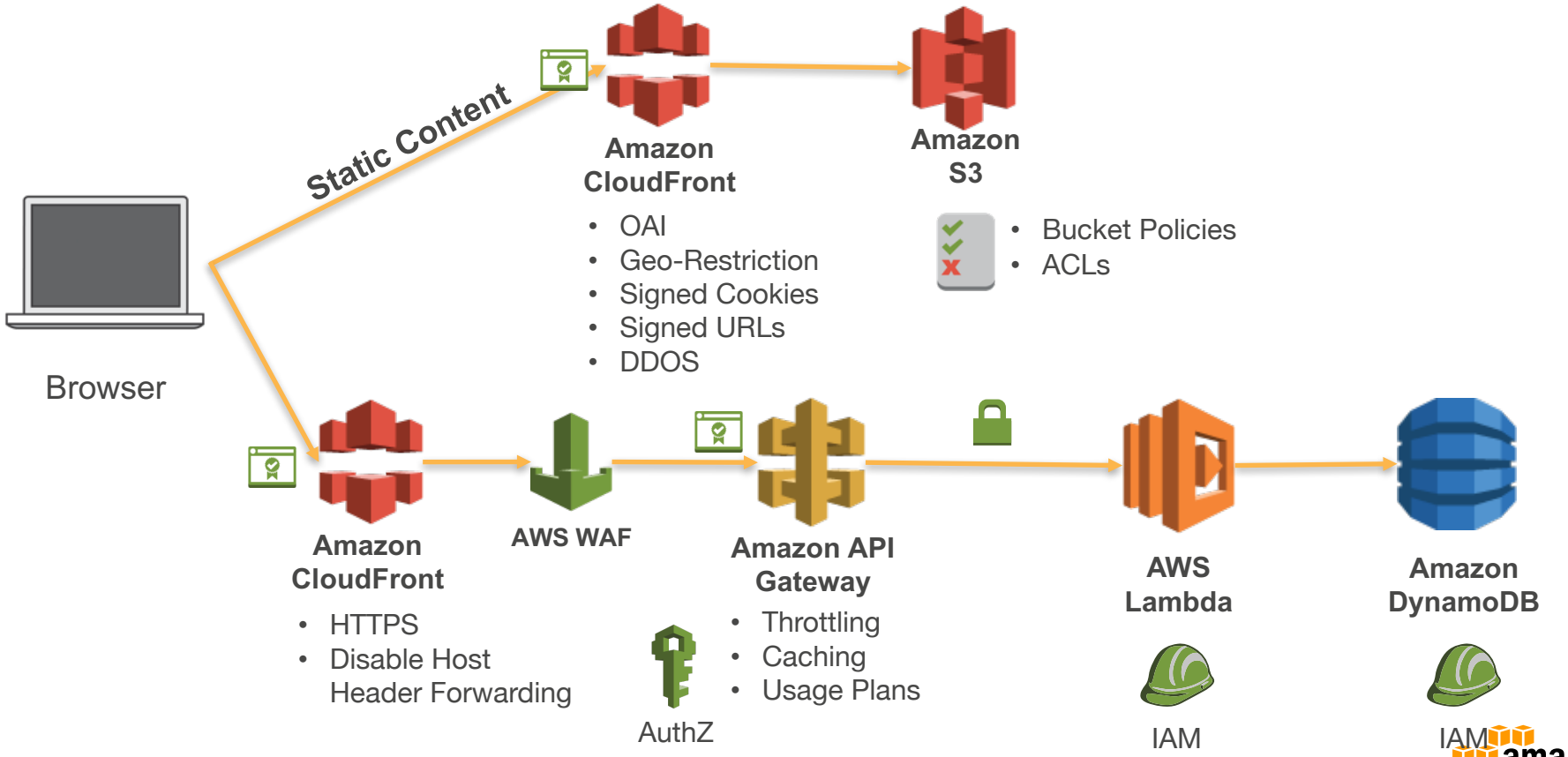
Serverless Web application



Serverless web app security

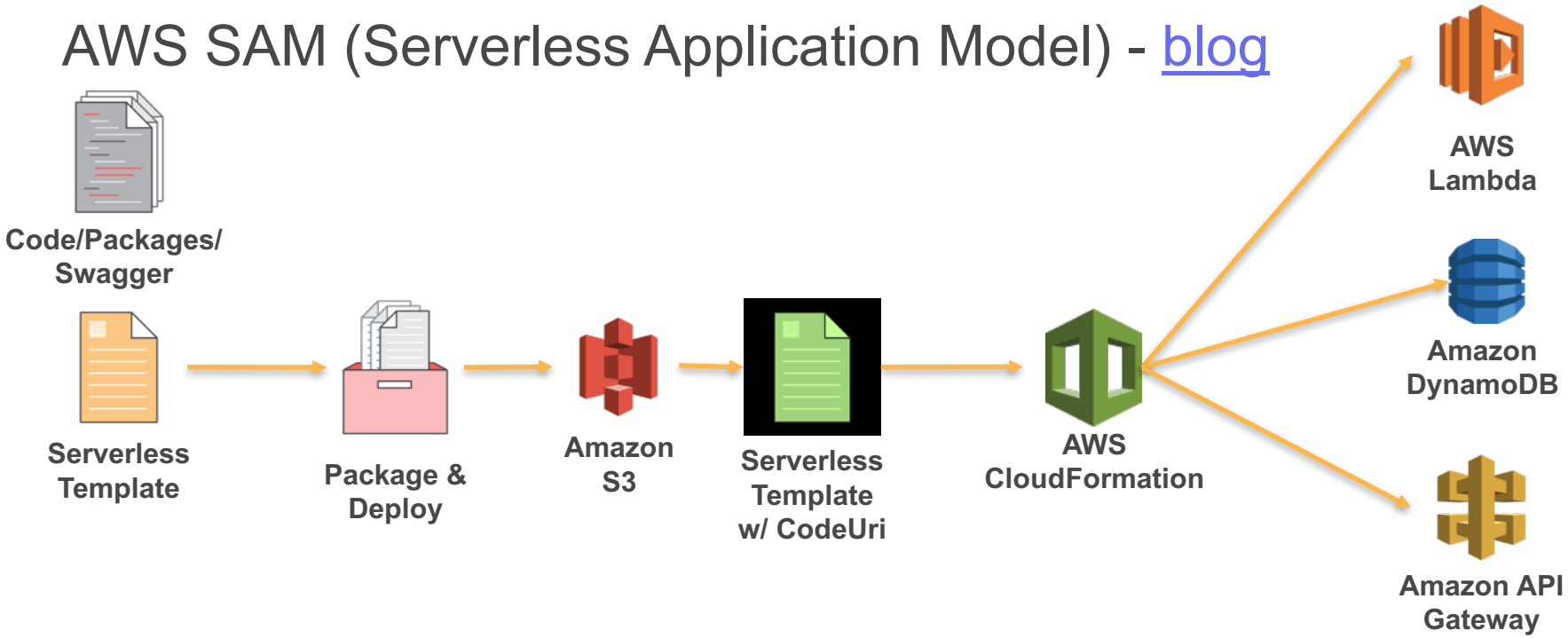


Serverless web app security



Serverless web app lifecycle management

AWS SAM (Serverless Application Model) - blog

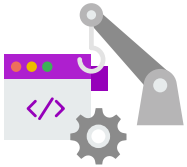


AWS Code* services

Software release steps:

AWS CodePipeline

AWS CodeCommit

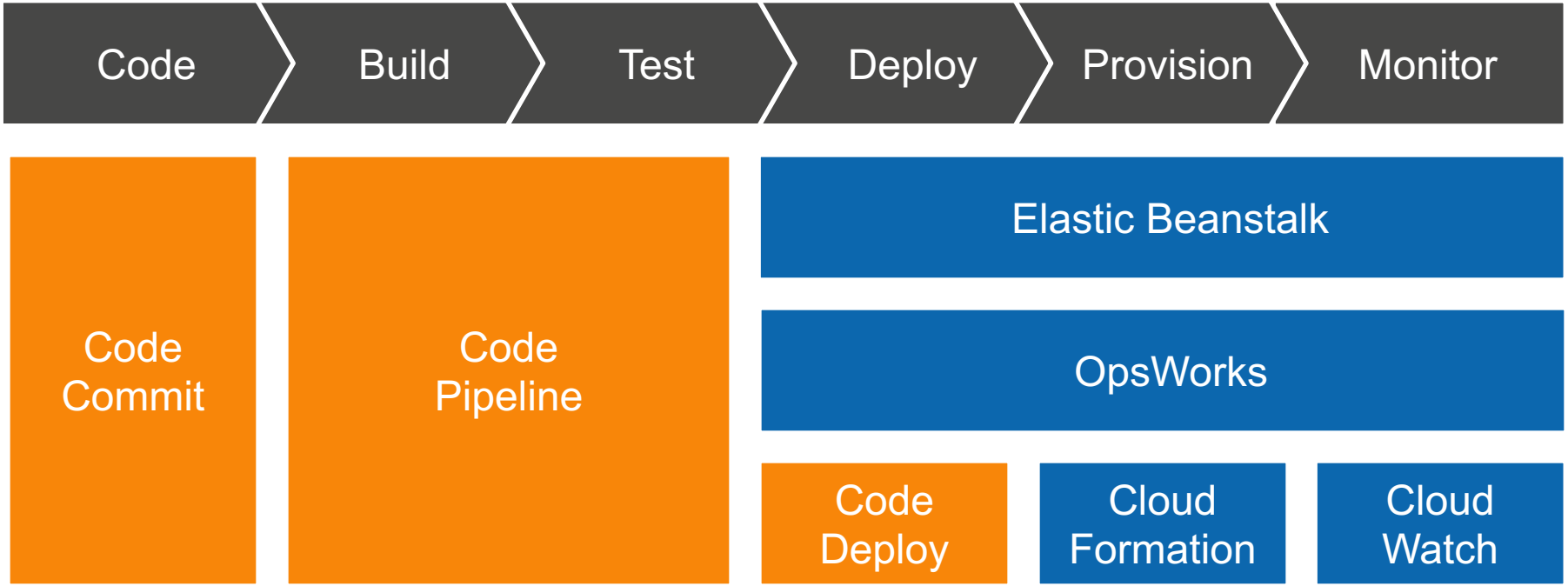


AWS CodeBuild

Third Party Tooling

AWS CodeDeploy

AWS Code, Deployment & Management Services



AWS Account Setup

Free Tier

- Includes most of the AWS services
- Available for all new accounts
- Good for one year from the day the account is created
- **Everything we show today can be done within the Free Tier!**
- More details at http://aws.amazon.com/free

Step 1: Sign up and configure basic security

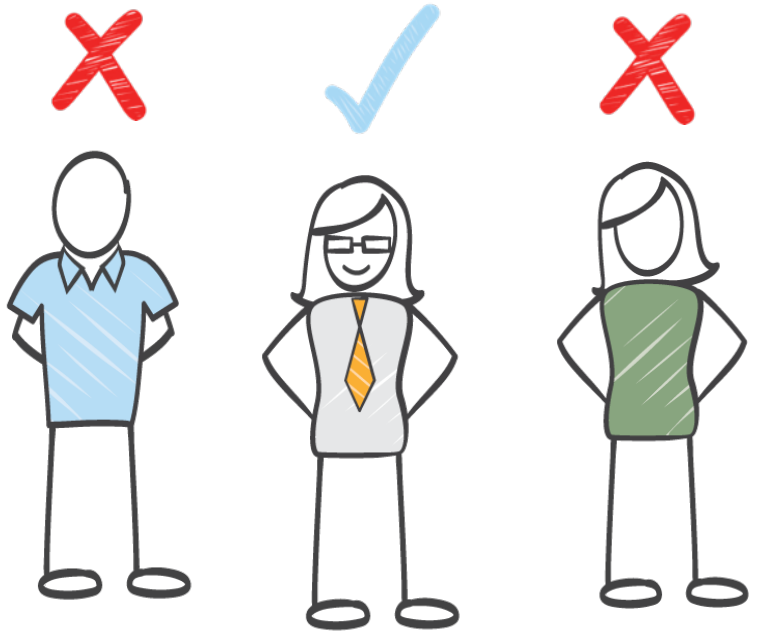
- Sign up though https://aws.amazon.com
- You need a credit card
- There will be a phone verification

amazon web services

Basic security: Creating IAM users

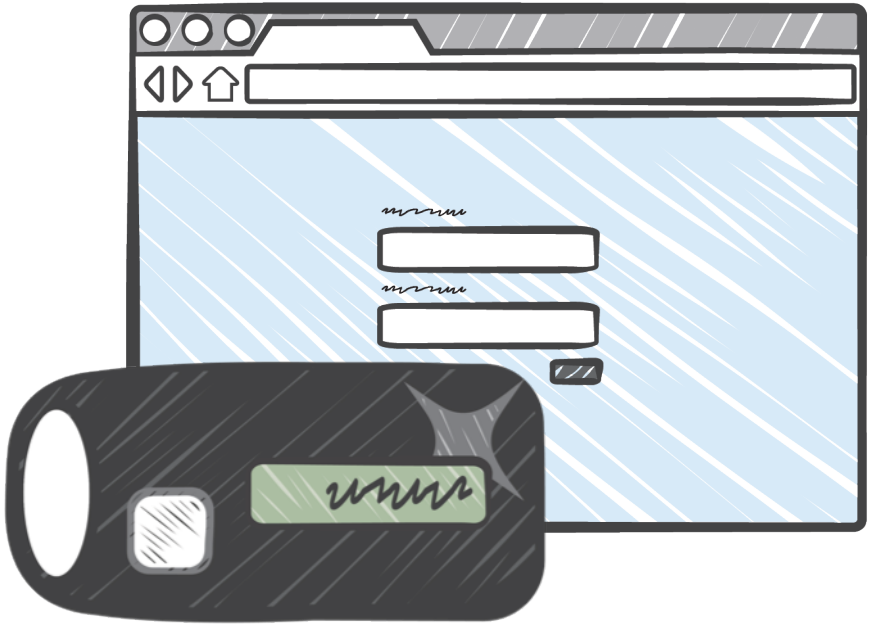
Using AWS Identity and Access Management (IAM), you can create and manage AWS users and groups.

You can control what resources each user has access to so you can avoid overly permissive accounts.



Enabling MFA

AWS allows you to require **Multi-Factor Authentication** (MFA) for your users through physical-based or software-based single use login tokens to thwart stolen passwords and key loggers as an attack vector.



AWS Account Strategies

Characteristics of an AWS account

Security boundary

Resource isolation

Billing separation

Account Structure Considerations

IAM users

- Across single or multiple AWS accounts?

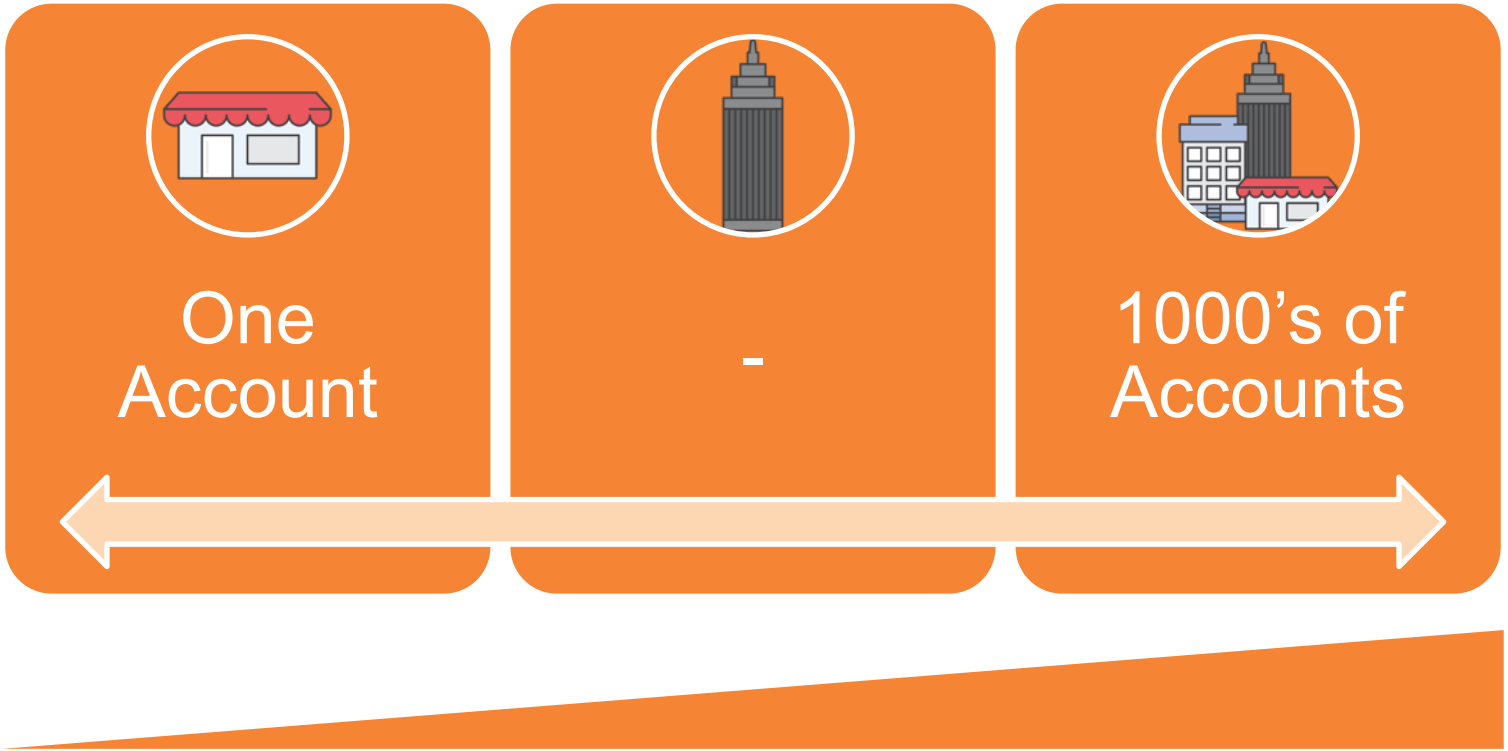
AD Federation

- To federate or not to federate?

Tool Chain

- Shared across all environments?

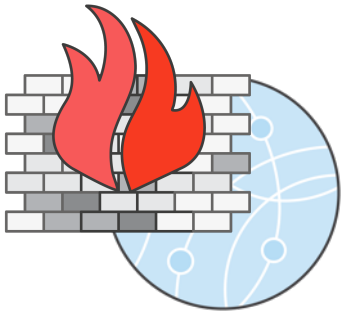
Approaches to account structure



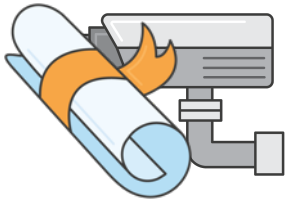
Why one isn't enough

Many Teams

Billing



Isolation



Security Controls

Business Process

Multiple accounts

Pros

+ Complete security and resources isolation
+ Smaller blast radius
+ Simplified billing per account

Cons

- Setup and operation overhead
- More complex security policies across accounts

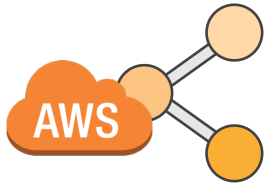
Goals of a multi-account architecture

- Automated setup of accounts
- Scalable
- Self-service
- Guardrails while enabling innovation
- Auditable for security and compliance
- Flexible

What accounts should I create?

Billing

Security



Shared Services

Sandbox

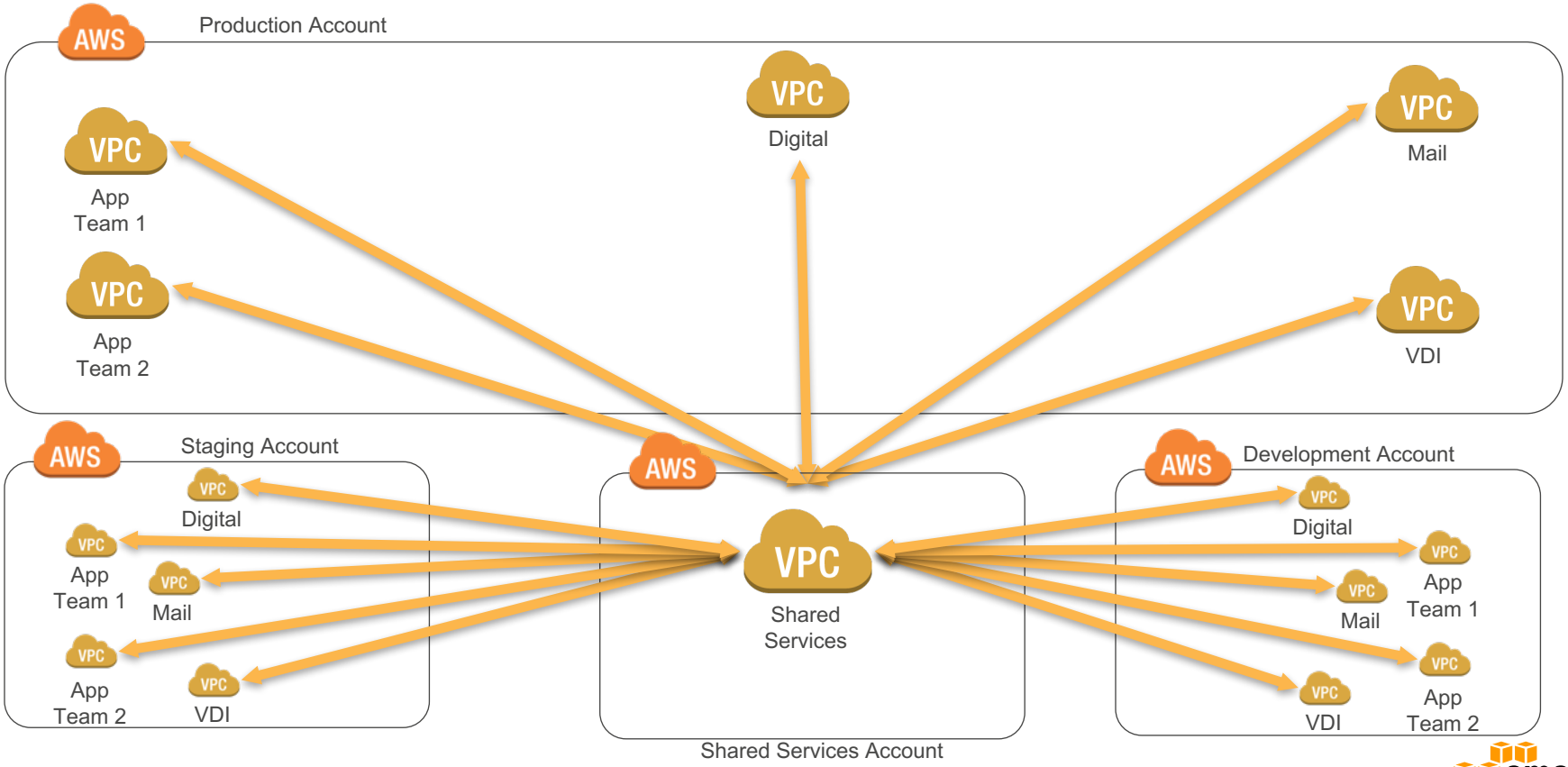
Non-prod

Prod

Other

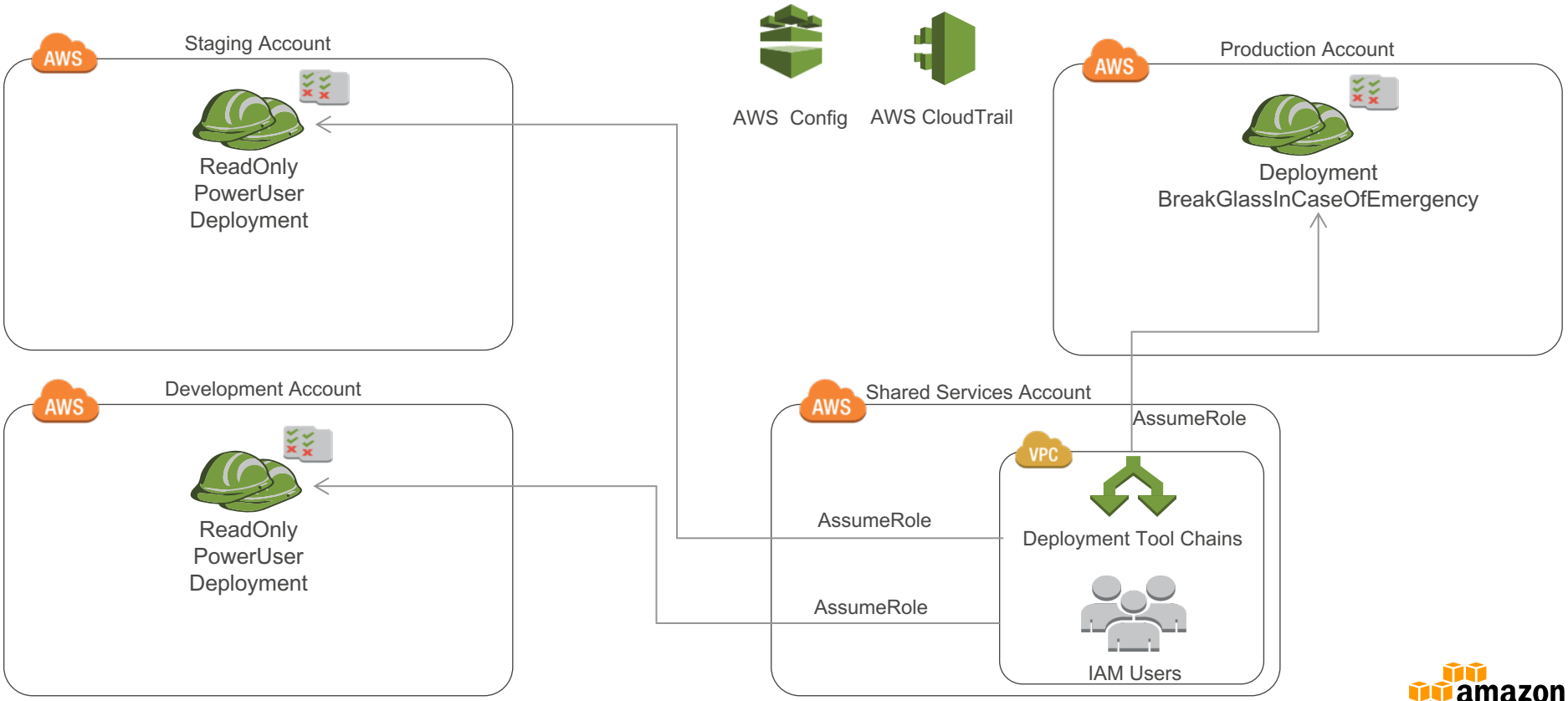
amazon web services

Central Account Structure



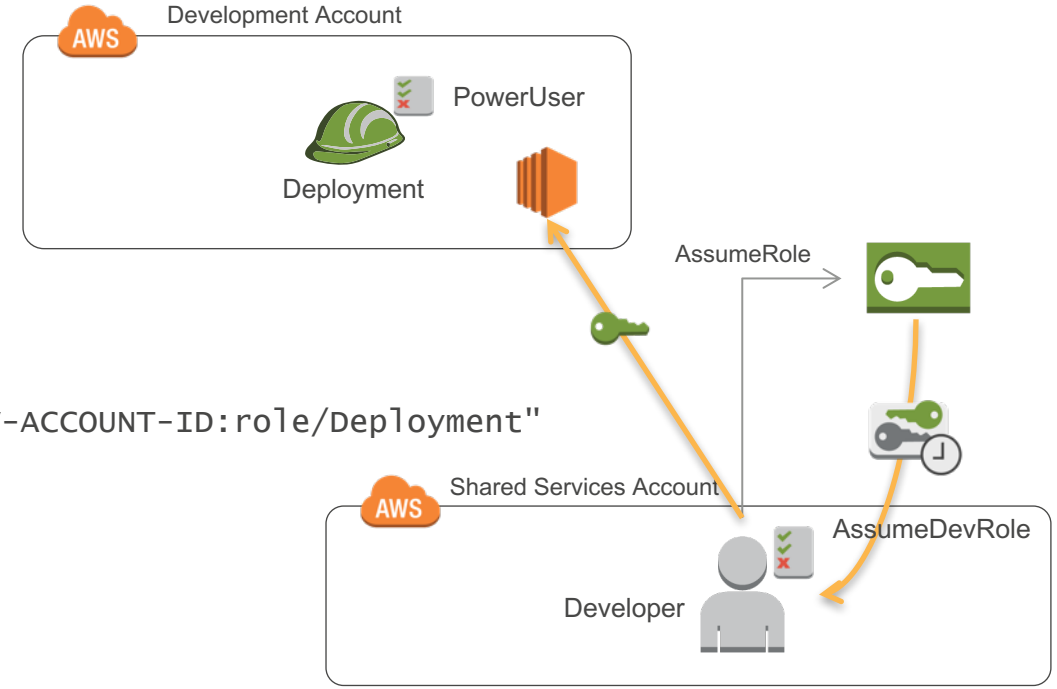
amazon web services

Central User Management



Grant User Access

```
"AssumeDeveloperRolePolicy": {
  "Properties": {
    "PolicyName": "AssumeDevRole",
    ”Users": [ "Developer" ],
    "PolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": "sts:AssumeRole”
          "Resource": "arn:aws:iam::DEV-ACCOUNT-ID:role/Deployment"
        }
      ]
    }
  },
  "Type": "AWS::IAM::Policy"
}
```



Demo AWS Console and CLI

Additional Resources

- AWS Getting Started
- AWS Multiple Account Billing Strategy
- Set up AWS CLI

Building Serverless APIs

Building blocks for serverless applications

Compute	Storage	Database
 AWS Lambda	 Amazon S3	 Amazon DynamoDB

API Proxy	Messaging and Queues	Analytics
 Amazon API Gateway	 Amazon SQS  Amazon SNS	 Amazon Kinesis

Orchestration and State Management	Monitoring and Debugging
 AWS Step Functions	 AWS X-Ray

Serverless APIs Fundamentals

Amazon API Gateway

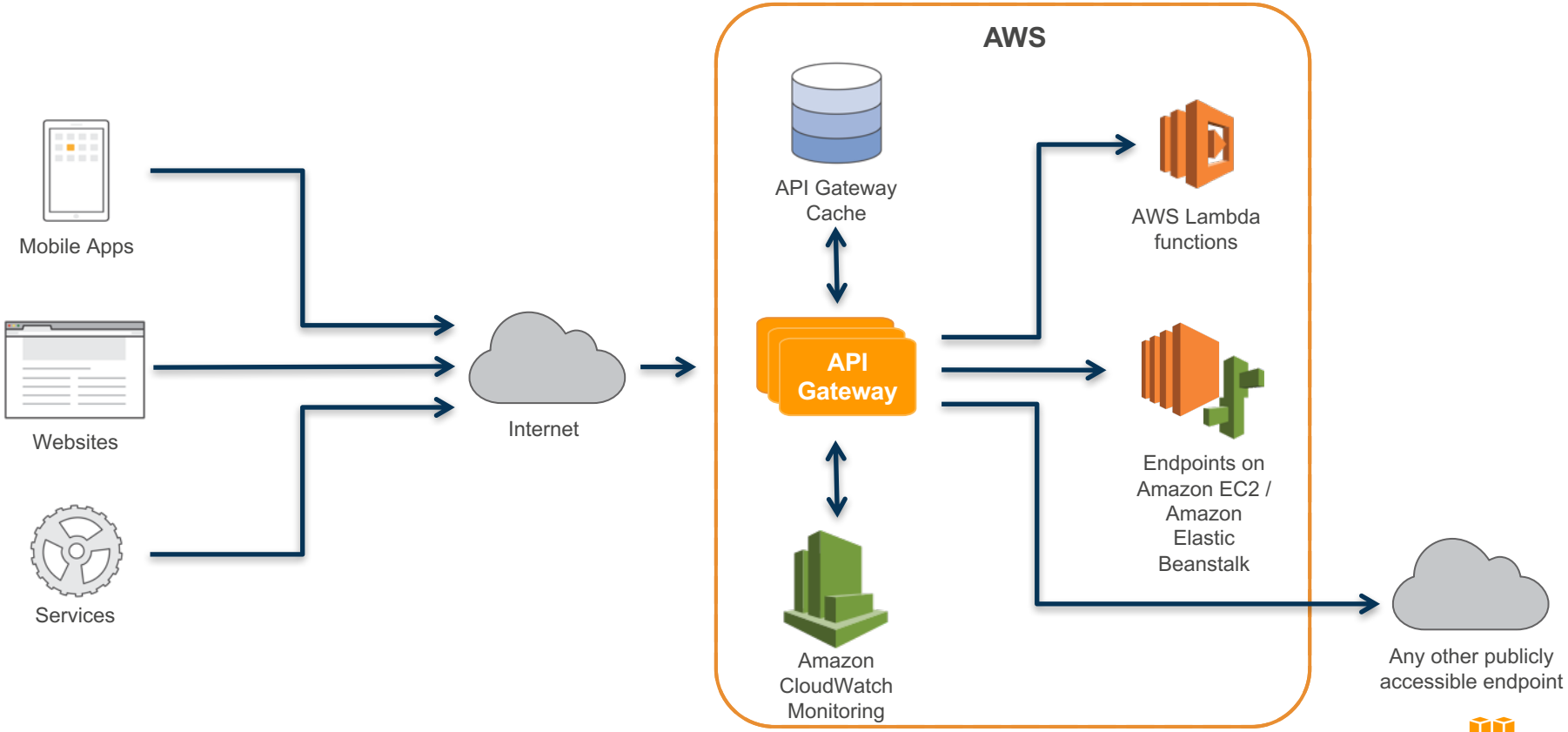
Dynamic content in AWS Lambda

Data stored in Amazon DynamoDB

Reference architectures: http://www.allthingsdistributed.com/2016/06/aws-lambda-serverless-reference-architectures.html

API Gateway

API Gateway: Serverless APIs



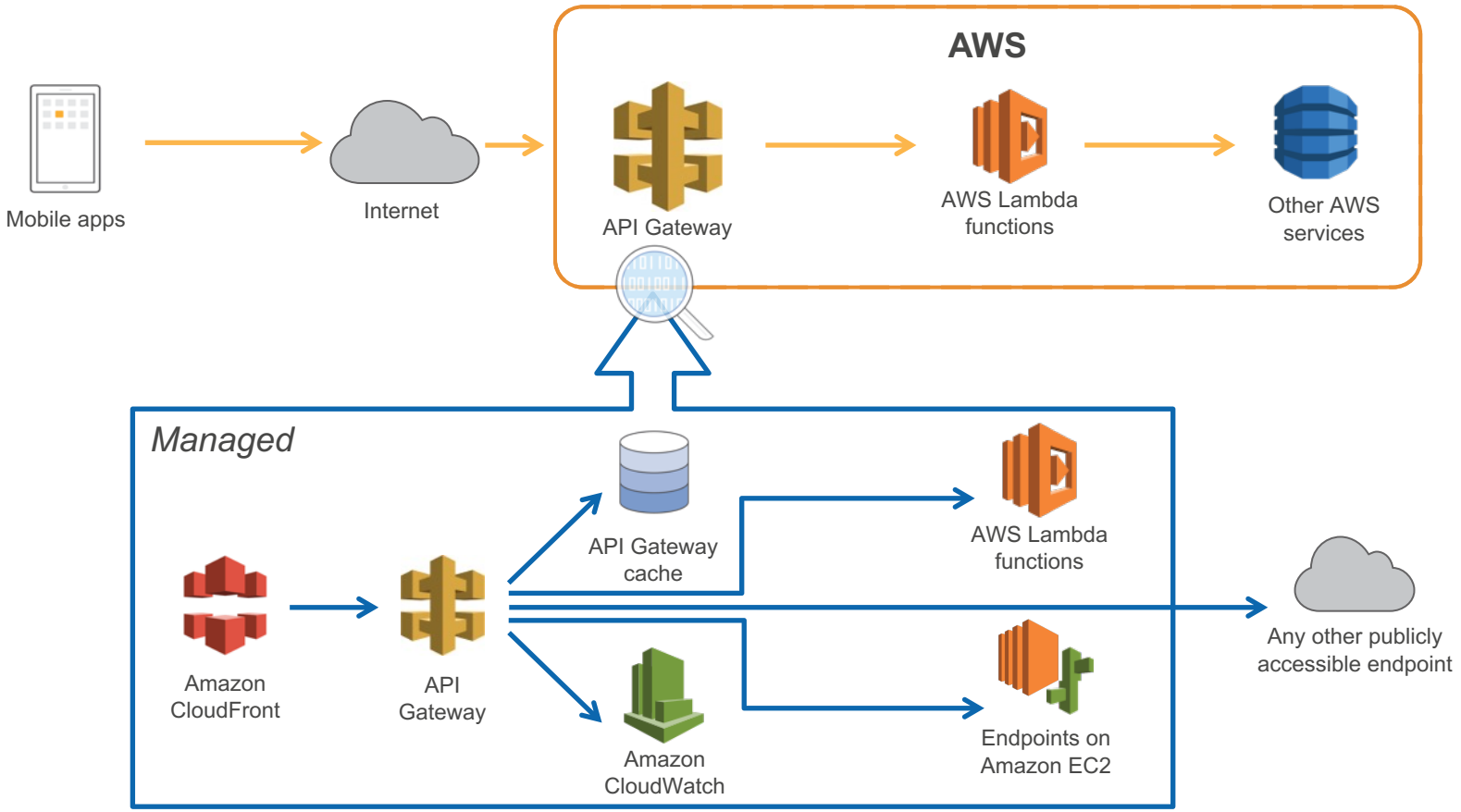
Introducing Amazon API Gateway

- Host multiple versions and stages of your APIs
- Create and distribute API Keys to developers
- Leverage AWS Sigv4 to authorize access to APIs
- Throttle and monitor requests to protect your backend
- Utilizes AWS Lambda 

Introducing Amazon API Gateway

- Managed cache to store API responses
- Reduced latency and DDoS protection through CloudFront
- SDK Generation for iOS, Android and JavaScript
- Swagger support
- Request / Response data transformation and API mocking

A new, fully managed model



Build, Deploy, Clone & Rollback

Build APIs with their resources, methods, and settings

Deploy APIs to a Stage

- Users can create as many Stages as they want, each with its own Throttling, Caching, Metering, and Logging configuration

Clone an existing API to create a new version

- Users can continue working on multiple versions of their APIs

Rollback to previous deployments

- We keep a history of customers' deployments so they can revert to a previous deployment

API Configuration

You can create APIs

Define resources within an API

Define methods for a resource

- Methods are Resource + HTTP verb

API Deployments

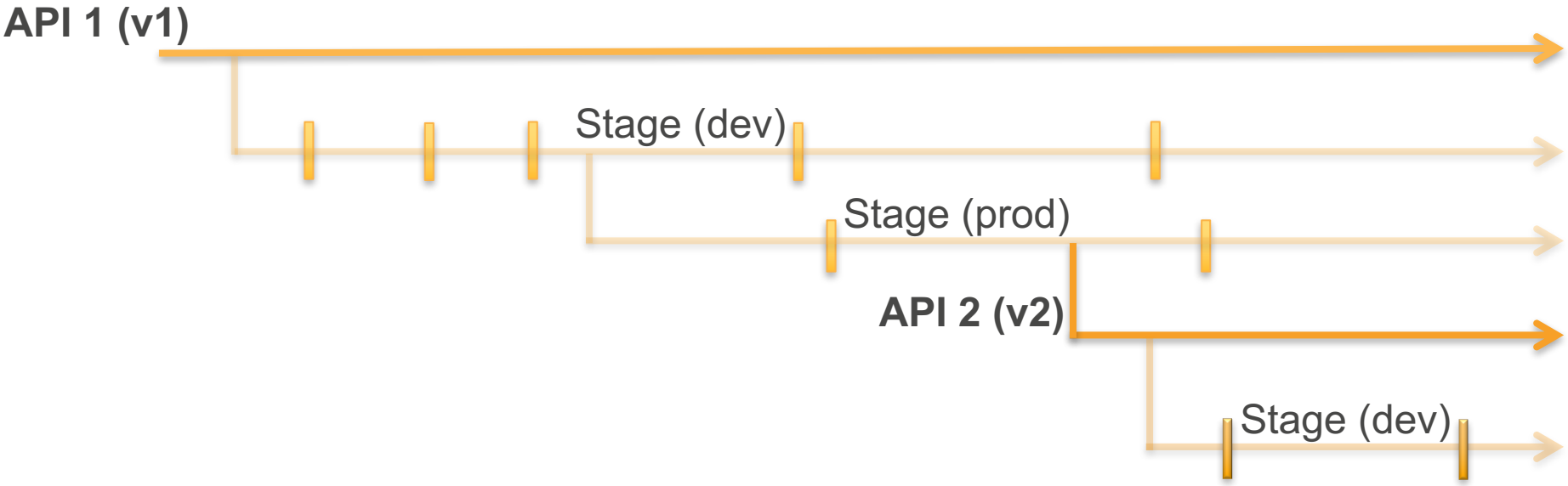
API Configuration can be deployed to a stage

Stages are different environments

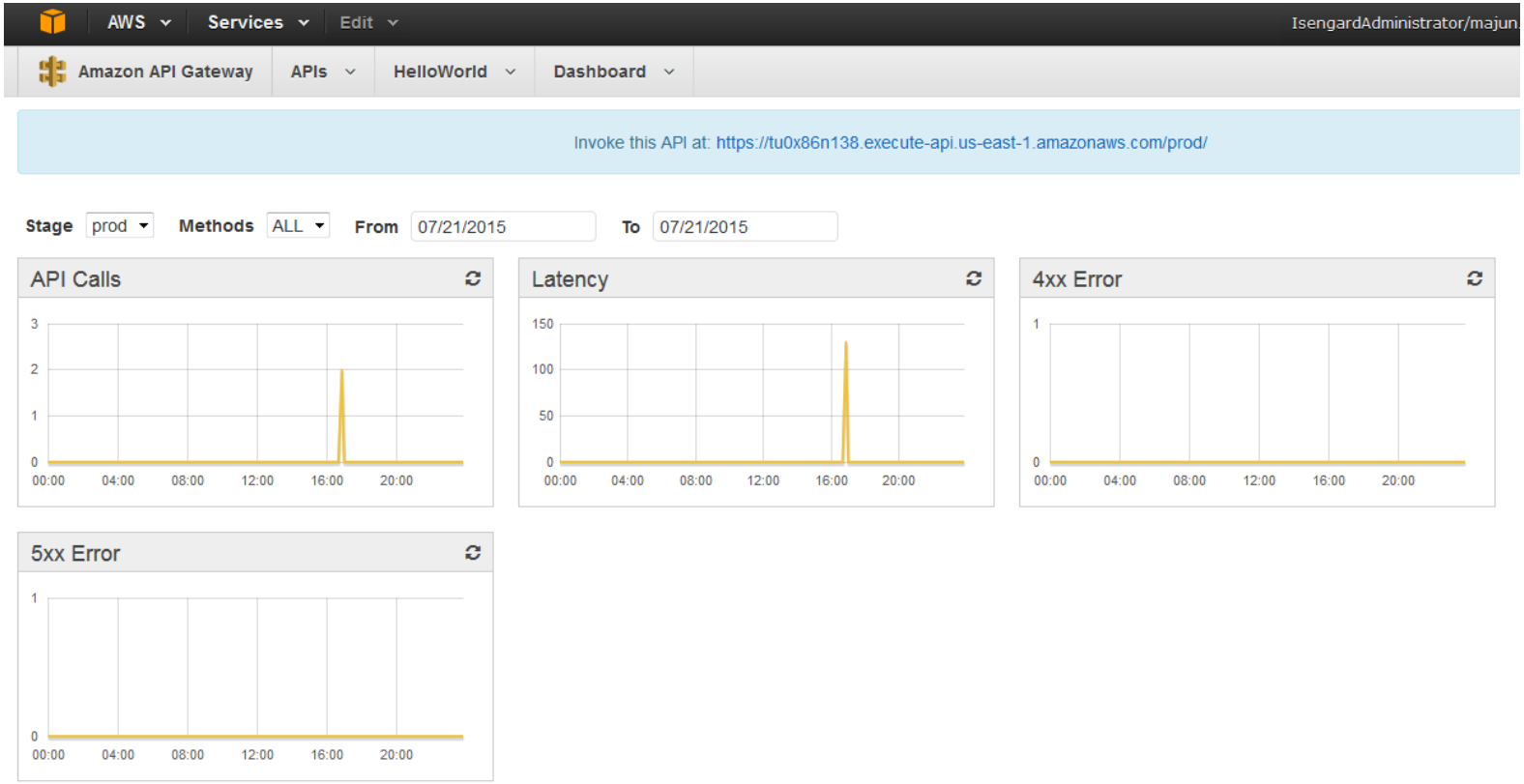
For example:

- Dev (e.g. awsapigateway.com/dev)
- Beta (e.g. awsapigateway.com/beta)
- Prod (e.g. awsapigateway.com/prod)
- As many stages as you need

Manage Multiple Versions and Stages of your APIs



API Monitoring



API Throttling

Throttling helps you manage traffic to your backend

Throttle by developer-defined Requests/Sec limits

Requests over the limit are throttled

- HTTP 429 response

The generated SDKs retry throttled requests

Caching of API Responses

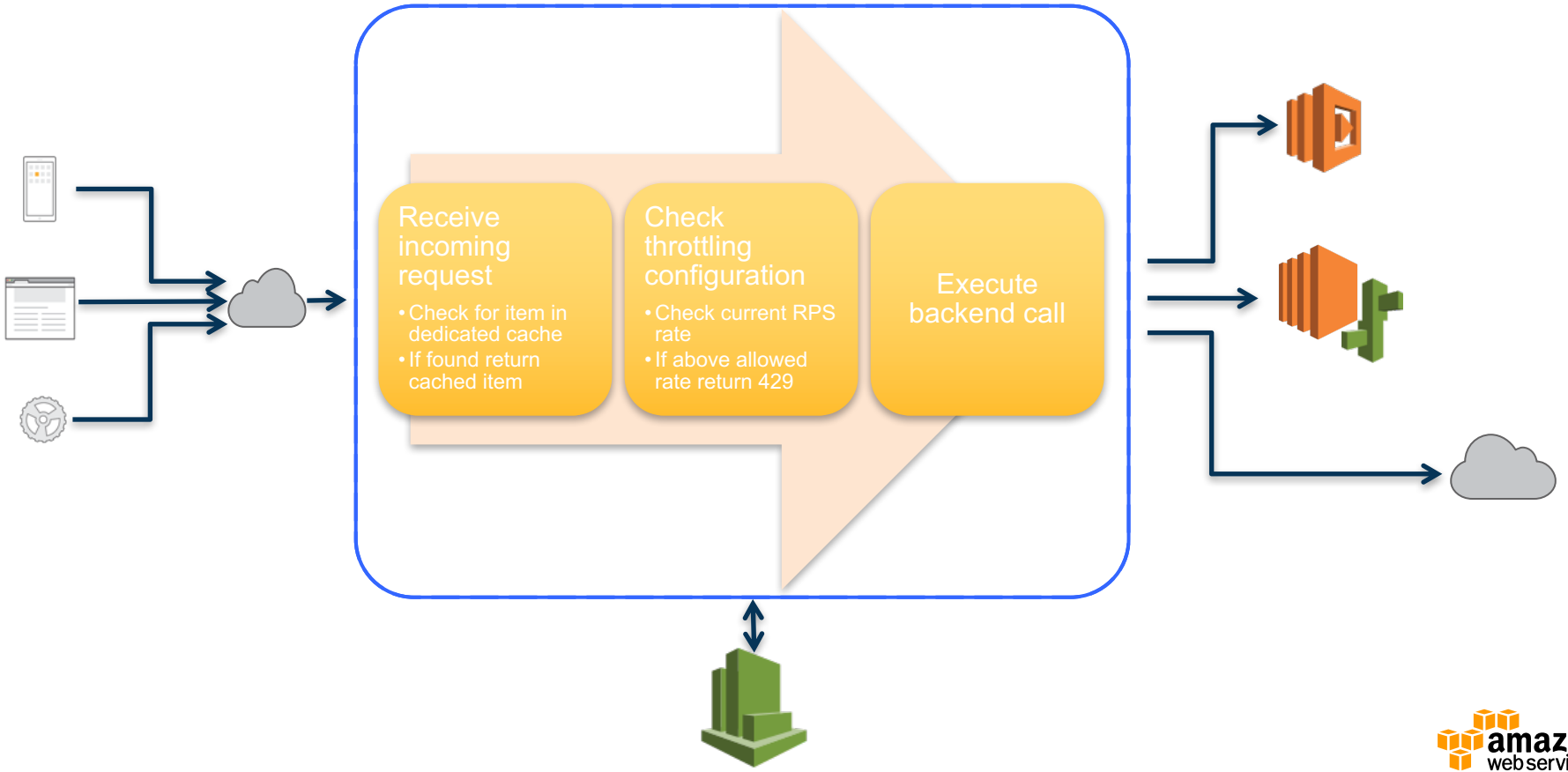
You can configure a cache key and the Time to Live (TTL) of the API response

Cached items are returned without calling the backend

A cache is dedicated to you, by stage

You can provision between 0.5GB to 237GB of cache

Request processing workflow



Identity and Access Management

The IAM role defines access permissions

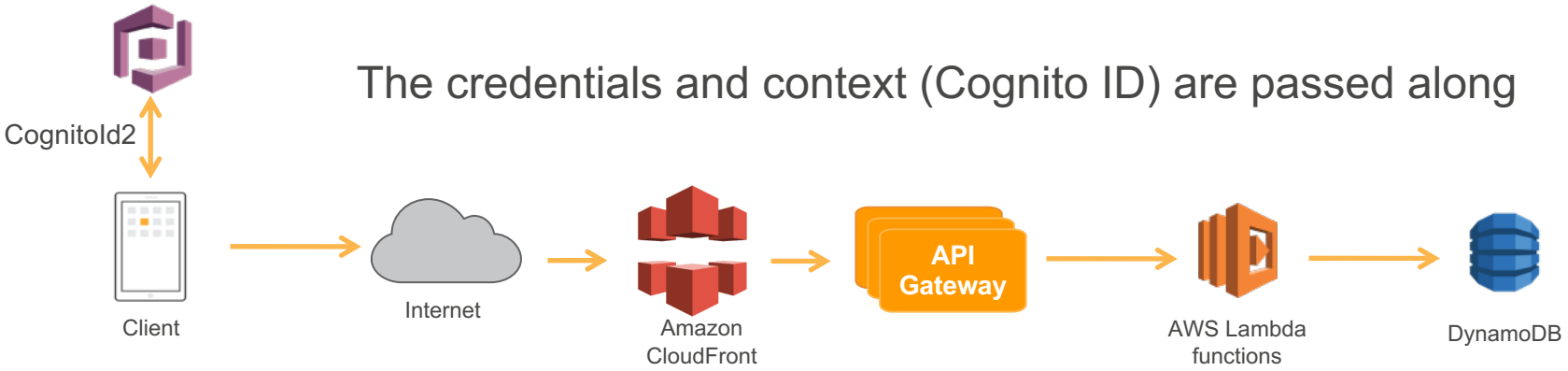
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
        "Action": [
          "dynamodb:GetItem",
          "dynamodb:PutItem",
          "dynamodb:Scan",
          "lambda:InvokeFunction",
          "execute-api:invoke"
        ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:xxxxxx:table/test_pets",
        "arn:aws:lambda:us-east-1:xxxxx:function:PetStore”,
        "arn:aws:execute-api:us-east-1:xxxx:API_ID/*/POST/pets"
      ]
    }
  ]
}
```

The role allows calls to:

- DynamoDB
- API Gateway
- Lambda

The role can access specific resources in these services

One step further: Fine-grained access permissions

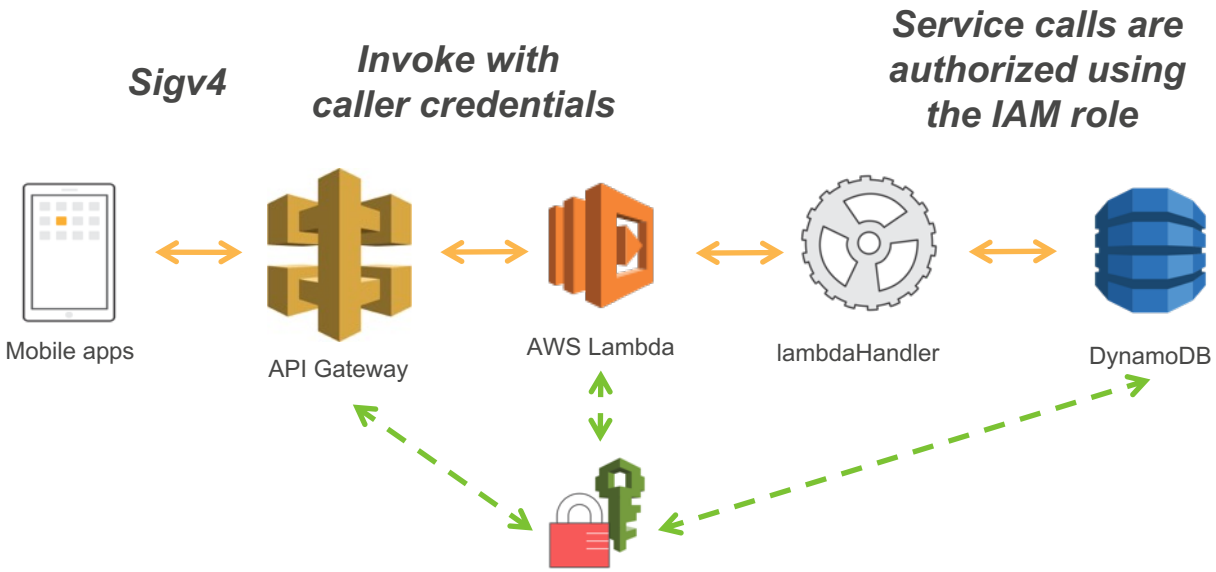


Both AWS Lambda & DynamoDB will follow the access policy

```
…
"Condition": {
    "ForAllValues:StringEquals": {
        "dynamodb:LeadingKeys":  ["${cognito-
identity.amazonaws.com:sub}"],
        "dynamodb:Attributes": [
            "UserId","GameTitle","Wins","Losses",
            "TopScore","TopScoreDateTime"
        ]
    },
    "StringEqualsIfExists": {
        "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
    }
}
…
```

	UserID	Wins	Losses
✗	cognitoId1	3	2
✓	cognitoId2	5	8
✗	cognitoId3	2	3

Authenticated flow in depth



Learn more about fine-grained access permissions

http://amzn.to/1YkxcjR

API Gateway Authentication

IAM credentials per API method

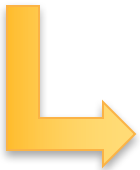
AWS Sigv4 to sign and authorize API calls

Temporary credentials

- Via Amazon Cognito or AWS Security Token Service (STS)

```
POST https://iam.amazonaws.com/ HTTP/1.1
host: iam.amazonaws.com
Content-type: application/x-www-form-urlencoded; charset=utf-8
x-amz-date: 20110909T233600Z

Action=ListUsers&Version=2010-05-08
```



```
POST https://iam.amazonaws.com/ HTTP/1.1
Authorization: AWS4-HMAC-SHA256 Credential=AKIDEXAMPLE/20110909/us-east-1/iam/aws4_requ
host: iam.amazonaws.com
Content-type: application/x-www-form-urlencoded; charset=utf-8
x-amz-date: 20110909T233600Z

Action=ListUsers&Version=2010-05-08
```

Authentication via Custom Header

OAuth or other authorization mechanisms through custom headers

Simply configure your API methods to forward the custom headers to you backend

Benefits of using AWS auth & IAM

- Separation of concerns – our **authorization strategy is delegated** to a dedicated service
- We have **centralized access management** to a single set of policies
- **Roles and credentials can be disabled** with a single API call

API Keys to Meter Developer Usage

Create API Keys

Set access permissions at the API/Stage level

Meter usage of the API Keys through CloudWatch Logs

API Keys to Authorize Access

The name “Key” implies security – there is no security in baking text in an App’s code

API Keys should be used purely to meter app/developer usage

API Keys should be used alongside a stronger authorization mechanism

Amazon API Gateway Best Practices

1. Use mock integrations
2. Combine with Amazon Cognito for managed end user-based access control.
3. Use stage variables (inject API config values into Lambda functions for logging, behavior).
4. Use request/response mapping templates everywhere within reason, not passthrough.
5. Take ownership of HTTP response codes.
6. Use Swagger import/export for cross-account sharing.

API Gateway Demo

AWS Lambda

AWS Lambda

Announced at re:Invent 2014

- Deploy pure functions in Java, Python, Node.js and C#
- Just code, without the infrastructure drama
- Built-in scalability and high availability
- Integrated with many AWS services
- Pay as you go
  - Combination of execution time (100ms slots) & memory
  - Starts at $0.000000208 per 100ms
  - Free tier available: first 1 million requests per month are free

http://aws.amazon.com/lambda

AWS re:Invent 2014 | (MBL202) NEW LAUNCH: Getting Started with AWS Lambda https://www.youtube.com/watch?v=UFj27laTWQA

amazon web services

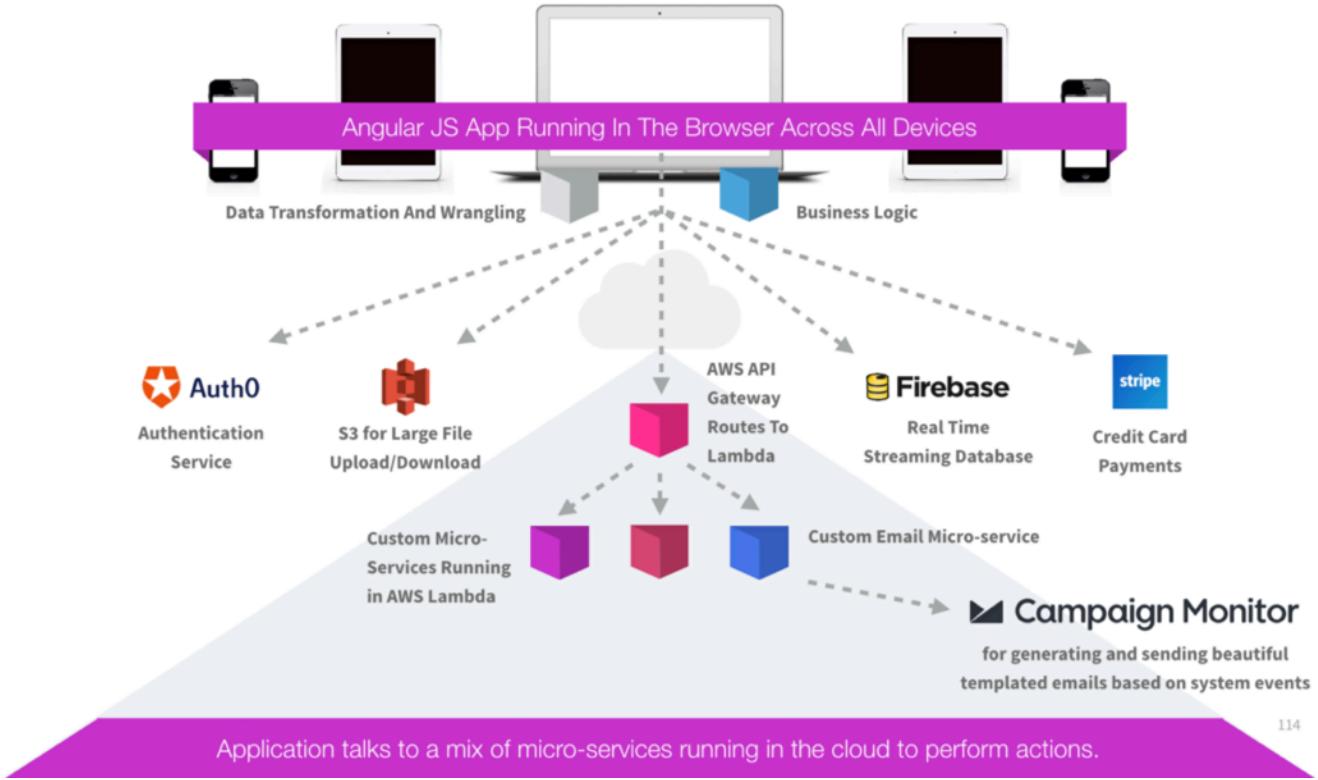
What can you do with AWS Lambda?

- Grow ‘connective tissue’ in your AWS infrastructure
  - Example: http://www.slideshare.net/JulienSIMON5/building-a-serverless-pipeline
- Build event-driven applications
- Build APIs together with Amazon API Gateway
  - RESTful APIs
  - Resources, methods
  - Stages

http://aws.amazon.com/apigateway

amazon web services

A Cloud Guru: 100% Serverless

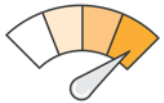


https://read.acloud.guru/serverless-the-future-of-software-architecture-d4473ffed864

AWS Lambda Programming Model

Bring your own code

- Node.js, Java, Python, C#
- Bring your own libraries (even native ones)

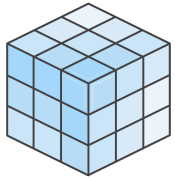


Simple resource model

- Select power rating from 128 MB to 1.5 GB
- CPU and network allocated proportionately
- Reports actual usage

Programming model

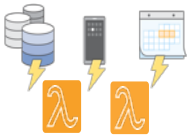
- AWS SDK built in (Python and Node.js)
- Lambda is the “webserver”
- Use processes, threads, /tmp, sockets normally



Stateless

- Persist data using Amazon DynamoDB, S3, or ElastiCache
- No affinity to infrastructure (can’t “log in to the box”)

Using AWS Lambda

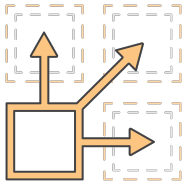


Flexible use

- Call or send events
- Integrated with other AWS services
- Build whole serverless ecosystems

Flexible authorization

- Securely grant access to resources, including VPCs
- Fine-grained control over who can call your functions



Authoring functions

- Author directly using the console WYSIWYG editor
- Package code as a .zip and upload to Lambda or S3
- Plugins for Eclipse and Visual Studio
- Command line tools

Monitoring and logging

- Built-in metrics for requests, errors, latency, and throttles
- Built-in logs in Amazon CloudWatch Logs

amazon web services

AWS Lambda Best Practices

1. Limit your function size – especially for Java (starting the JVM takes time).
2. Don’t assume function container reuse – but take advantage of it when it does occur.
3. Don’t forget about disk (500 MB /tmp directory provided to each function).
4. Use function aliases for release.
5. Use the included logger (include details from service-provided context).
6. Create custom metrics (operations-centric, and business-centric).

Amazon DynamoDB

Amazon DynamoDB

Run your business, not your database

DynamoDB Benefits

Fully managed

Fast, consistent performance

Highly scalable

Flexible

Event-driven programming

Fine-grained access control

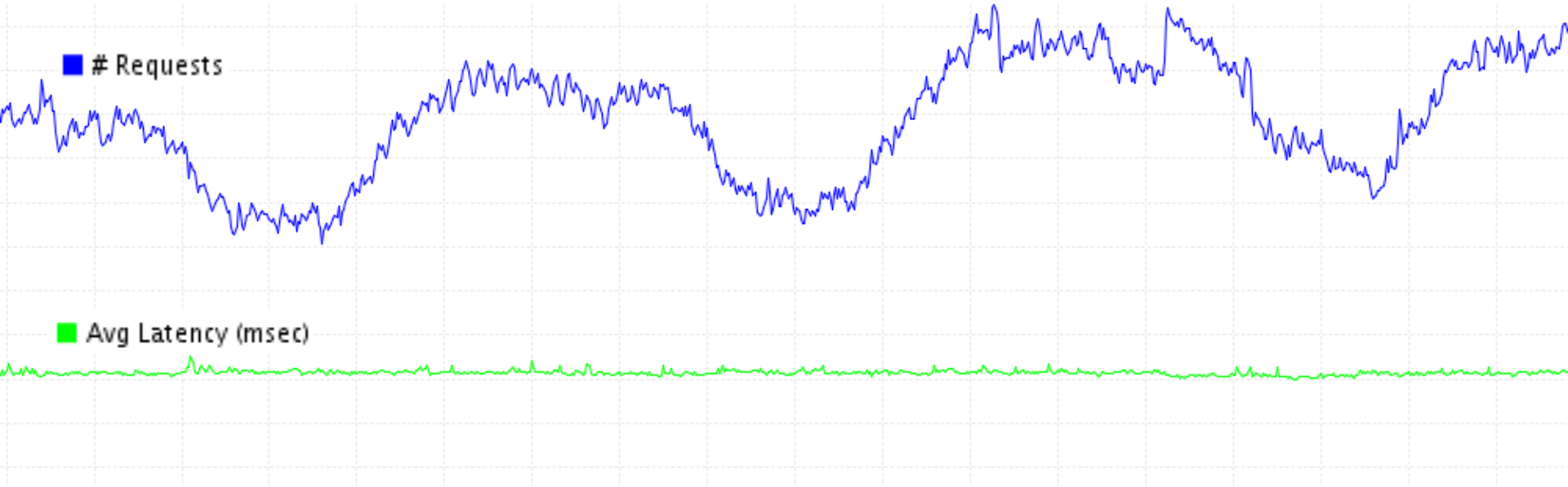
Fully managed service = automated operations

DB hosted on premise

DynamoDB

amazon
web services

Consistently low latency at scale

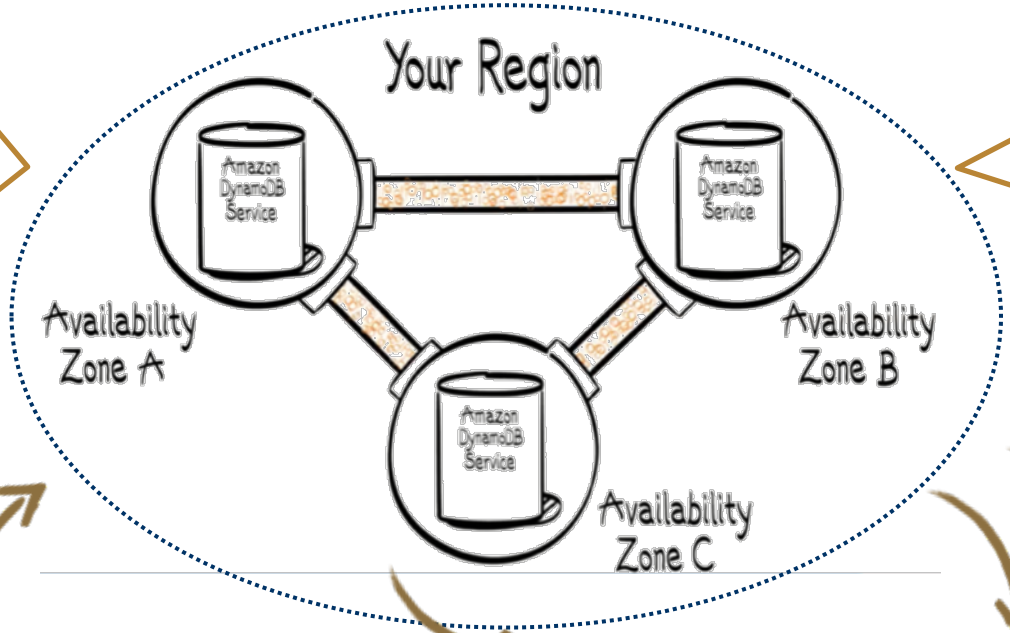


PREDICTABLE PERFORMANCE!

amazon web services

High availability and durability

Built for high durability

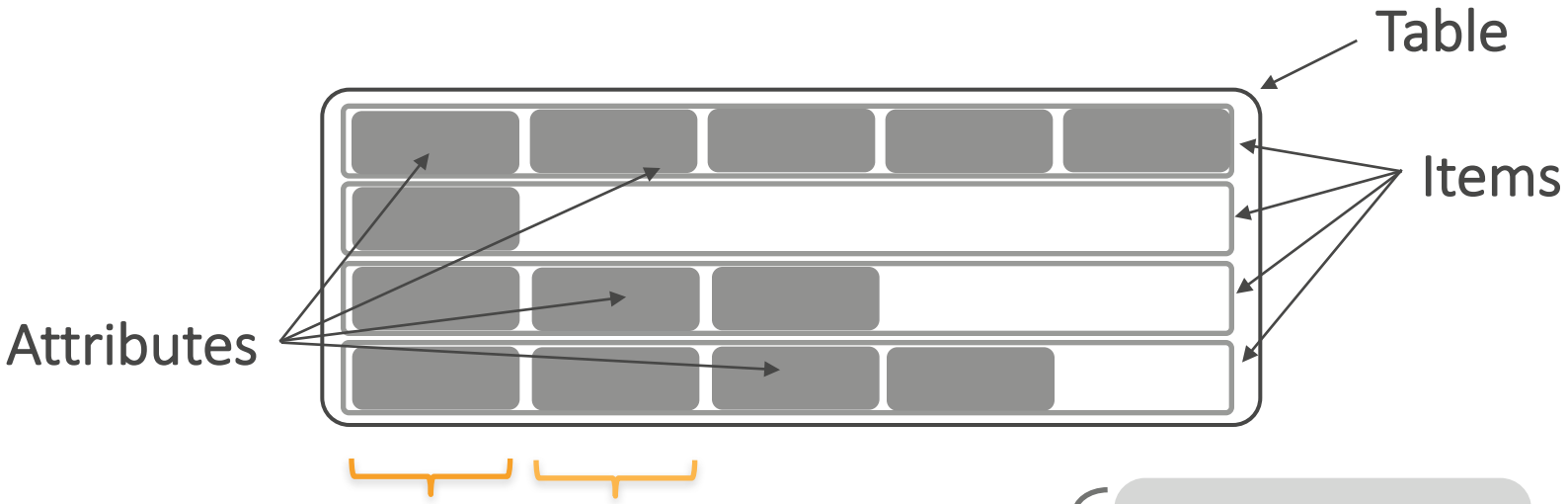


Designed to support 99.99% of availability

WRITES
Replicated continuously to 3 AZs
Persisted to disk (custom SSD)

READS
Strongly or eventually consistent
No latency trade-off

DynamoDB table structure



Mandatory
Key-value access pattern
Determines data distribution

Optional
Model 1:N relationships
Enables rich query capabilities

All items for key
==, <, >, >=, <=
“begins with”
“between”
“contains”
“in”
sorted results
counts
top/bottom N values

Advanced topics in DynamoDB

- Data modeling
- Indexes - query a table using alternate attributes
  - Local Secondary Indexes
  - Global Secondary Indexes
- Understanding Partitions
  - # of partitions depend on table throughput and size
- Design patterns and best practices

Scaling

Throughput

- Provision any amount of throughput to a table

Size

- Add any number of items to a table
  - Maximum item size is 400 KB
  - LSIs limit the number of range keys due to 10 GB limit

Scaling is achieved through partitioning

Throughput

Provisioned at the table level

- Write capacity units (WCUs) are measured in 1 KB per second
- Read capacity units (RCUs) are measured in 4 KB per second
  - RCUs measure strictly consistent reads
  - Eventually consistent reads cost 1/2 of consistent reads

Read and write throughput limits are independent

Simplifying Development

Typical development workflow

1. Write and deploy a Lambda function
2. Create and deploy a REST API with API Gateway
3. Connect the API to the Lambda function
4. Invoke the API
5. Test, debug and repeat ;)

The Serverless framework

formerly known as JAWS: Just AWS Without Servers

- Announced at re:Invent 2015 by Austen Collins and Ryan Pendergast
- Supports Node.js, as well as Python and Java (with restrictions)
- Auto-deploys and runs Lambda functions locally or remotely
- Auto-deploys your Lambda event sources: API Gateway, S3, DynamoDB
- Creates all required infrastructure with CloudFormation
- Simple configuration in YML

http://github.com/serverless/serverless
https://serverless.com
AWS re:Invent 2015 | (DVO209) https://www.youtube.com/watch?v=D_U6luQ6l90 https://vimeo.com/141132756

amazon web services

Serverless: “Hello World” API

```
$ serverless create

Edit handler.js, serverless.yml and event.json

$ serverless deploy [--stage stage_name]

$ serverless invoke [--local] --function function_name

$ serverless info

$ http $URL
```

Gordon

Released in Oct'15 by Jorge Batista

Supports Python, Javascript, Golang, Java, Scala, Kotlin (including in the same project)

Auto-deploys and runs Lambda functions, locally or remotely

Auto-deploys your Lambda event sources: API Gateway, CloudWatch Events, DynamoDB Streams, Kinesis Streams, S3

Creates all required infrastructure with CloudFormation

Simple configuration in YML

https://github.com/jorgebastida/gordon
https://news.ycombinator.com/item?id=11821295

Gordon: “Hello World” API

```
$ gordon startproject helloworld

$ gordon startapp helloapp

Write hellofunc() function

$ gordon build

$ echo '{"name":"Julien"}' | gordon run helloapp.hellofunc

$ gordon apply [--stage stage_name]

$ http post $URL name=Julien
```

AWS Chalice

Think of it as a serverless framework for Flask apps

- Released in Jul'16, still in beta
- Just add your Python code
  - Deploy with a single call and zero config
  - The API is created automatically, the IAM policy is auto-generated
- Run APIs locally on port 8000 (similar to Flask)
- Fast & lightweight framework
  - 100% *boto3* calls (AWS SDK for Python) → fast
  - No integration with CloudFormation → no creation of event sources

https://github.com/awslabs/chalice
https://aws.amazon.com/blogs/developer/preview-the-python-serverless-microframework-for-aws/

AWS Chalice: “Hello World” API

```
$ chalice new-project helloworld

Write your function in app.py

$ chalice local

$ chalice deploy

$ export URL=`chalice url`

$ http $URL

$ http put $URL/hello/tobias

$ chalice logs [ --include-lambda-messages ]
```

amazon web services

AWS Chalice: PUT/GET in S3 bucket

```
$ chalice new-project s3test

Write your function in app.py

$ chalice local

$ http put http://localhost:8000/objects/doc.json value1=5 value2=8

$ http get http://localhost:8000/objects/doc.json

$ chalice deploy [stage_name]

$ export URL=`chalice url`

$ http put $URL/objects/doc.json value1=5 value2=8

$ http get $URL/objects/doc.json
```

Demo Chalice

Summing things up

Serverless	Gordon	Chalice
The most popular serverless framework	Great challenger!	AWS project, in beta
Built with and for Node.js. Python and Java: YMMV	Node.js, Python, Java, Scala, Golang	Python only
Rich features, many event sources	Comparable to Serverless feature-wise	Does only one thing, but does it great
Not a web framework	Not a web framework	Dead simple, zero config
		Flask web framework

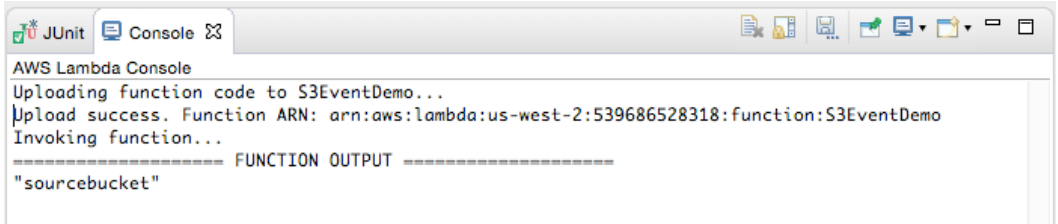
More Lambda frameworks

- Kappa https://github.com/garnaat/kappa
  - Released Dec'14 by Mitch Garnaat, author of boto and the AWS CLI (still maintained?)
  - Python only, multiple event sources
- Apex https://github.com/**apex/apex**
  - Released in Dec'15 by TJ Holowaychuk
  - Python, Javascript, Java, Golang
  - Terraform integration to manage infrastructure for event sources
- Zappa https://github.com/Miserlou/Zappa
  - Released in Feb'16 by Rich Jones
  - Python web applications on AWS Lambda + API Gateway
- Docker-lambda https://github.com/lambci/docker-lambda
  - Released in May'16 by Michael Hart
  - Run functions in Docker images that "replicate" the live Lambda environment

2 Java tools for AWS Lambda

Eclipse plug-in

- Code test and deploy Lambdas from Eclipse
- Run your functions locally and remotely
- Test with local events and Junit4
- Deploy standalone functions, or with the AWS Serverless Application Model (Dec'16)



Serverless Java Container

- Run Java RESTful APIs as-is
- Default implementation of the Java servlet `HttpServletRequest` `HttpServletResponse`
- Support for Java frameworks such as Jersey or Spark

https://java.awsblog.com/post/TxWZES6J1RSQ2Z/Testing-Lambda-functions-using-the-AWS-Toolkit-for-Eclipse
https://aws.amazon.com/blogs/developer/aws-toolkit-for-eclipse-serverless-application
https://github.com/awslabs/aws-serverless-java-container

amazon web services

Lambda Java Resources

- AWS Lambda Getting Started
- Lambda Java Programming Model
- AWS Java Developer Blog
- AWS Serverless Java Container
- POJO Modeling with DynamoDBMapper
- Lambada Framework

Best Practices

1. Use strategic, consumable naming conventions (Lambda function names, IAM roles, API names, API stage names, etc.).
2. Use naming conventions and versioning to create automation.
3. Externalize authorization to IAM roles whenever possible.
4. Least privilege and separate IAM roles.
5. Externalize configuration – DynamoDB is great for this.
6. Be aware of service throttling, engage AWS support if so.

Simplifying Deployment

AWS Serverless Application Model (SAM)

- CloudFormation extension released in Nov'16 to bundle Lambda functions, APIs & events
- 3 new CloudFormation resource types
  - *AWS::Serverless::Function*
  - *AWS::Serverless::Api*
  - *AWS::Serverless::SimpleTable* (DynamoDB)
- 2 new CloudFormation CLI commands
  - '*aws cloudformation package*'
  - '*aws cloudformation deploy*'
- Integration with CodeBuild and CodePipeline for CI/CD
- Expect SAM to be integrated in most / all frameworks

https://aws.amazon.com/fr/blogs/compute/introducing-simplified-serverless-application-deployment-and-management
https://github.com/awslabs/serverless-application-model/

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Get items from a DynamoDB table.
Resources:
  GetFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.get
      Runtime: nodejs4.3
      Policies: AmazonDynamoDBReadOnlyAccess
      Environment:
        Variables:
          TABLE_NAME: !Ref Table
      Events:
        GetResource:
          Type: Api
          Properties:
            Path: /resource/{resourceId}
            Method: get
  Table:
    Type: AWS::Serverless::SimpleTable
```

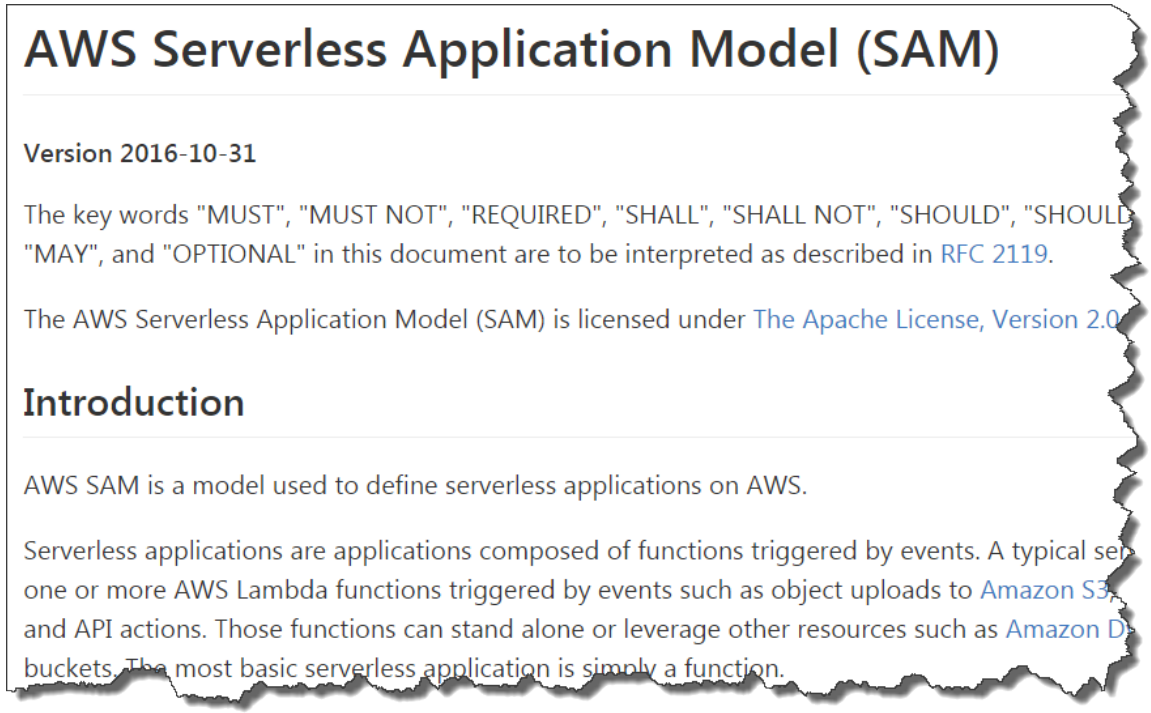
Sample SAM template for:

- Lambda function
- HTTP GET API
- DynamoDB table

SAM: Open Specification

A common language to describe the content of a serverless application *across the ecosystem*.

Apache 2.0 licensed GitHub project



AWS Serverless Application Model (SAM)

Version 2016-10-31

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD
"MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

The AWS Serverless Application Model (SAM) is licensed under The Apache License, Version 2.0

Introduction

AWS SAM is a model used to define serverless applications on AWS.

Serverless applications are applications composed of functions triggered by events. A typical se
one or more AWS Lambda functions triggered by events such as object uploads to Amazon S3,
and API actions. Those functions can stand alone or leverage other resources such as Amazon D
buckets. The most basic serverless application is simply a function.

amazon web services

Demo AWS Toolkit for Eclipse

New Lambda videos from re:Invent 2016

AWS re:Invent 2016: What's New with AWS Lambda (SVR202)
https://www.youtube.com/watch?v=CwxWhyGteNc

AWS re:Invent 2016: Serverless Apps with AWS Step Functions (SVR201)
https://www.youtube.com/watch?v=75MRve4nv8s

AWS re:Invent 2016: Real-time Data Processing Using AWS Lambda (SVR301)
https://www.youtube.com/watch?v=VFLKOy4GKXQ

AWS re:Invent 2016: Serverless Architectural Patterns and Best Practices (ARC402)
https://www.youtube.com/watch?v=b7UMoc1iUYw

AWS re:Invent 2016: Bringing AWS Lambda to the Edge (CTD206)
https://www.youtube.com/watch?v=j26novaqF6M

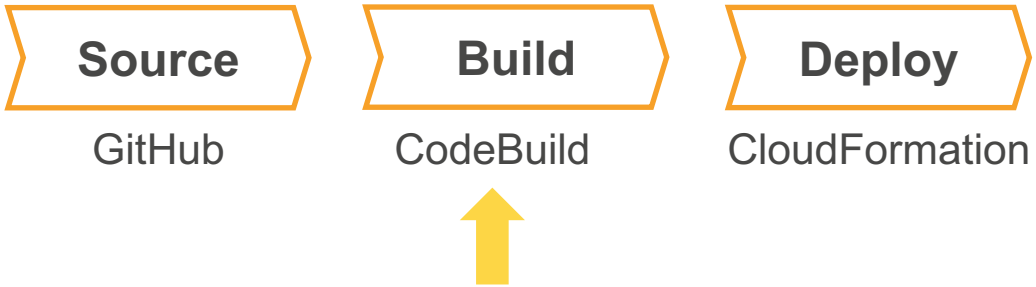
AWS re:Invent 2016: Ubiquitous Computing with Greengrass (IOT201)
https://www.youtube.com/watch?v=XQQjX8GTEko

Putting it all together

Serverless CI/CD pipeline

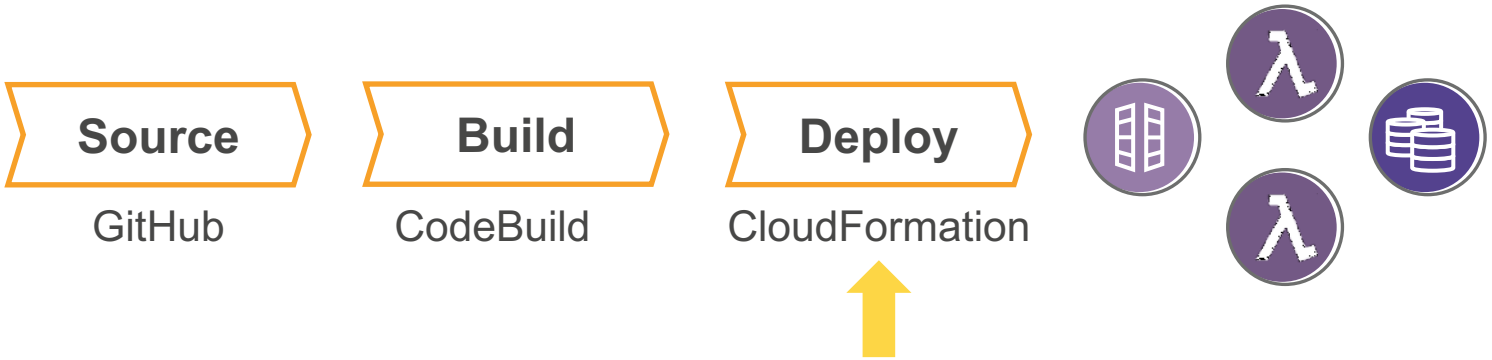
- Pull source directly from GitHub or CodeCommit using CodePipeline

Serverless CI/CD pipeline



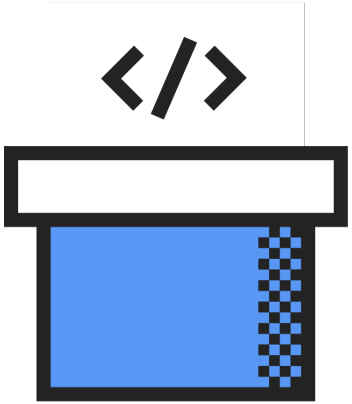
- Pull source directly from GitHub or AWS CodeCommit using AWS CodePipeline
- Build and package serverless apps with AWS CodeBuild
  - npm, pip, Java compilation, BYO Docker…

Serverless CI/CD pipeline



- Pull source directly from GitHub or AWS CodeCommit using AWS CodePipeline
- Build and package serverless apps with AWS CodeBuild
- Deploy your completed Lambda app with AWS CloudFormation

AWS CodePipeline

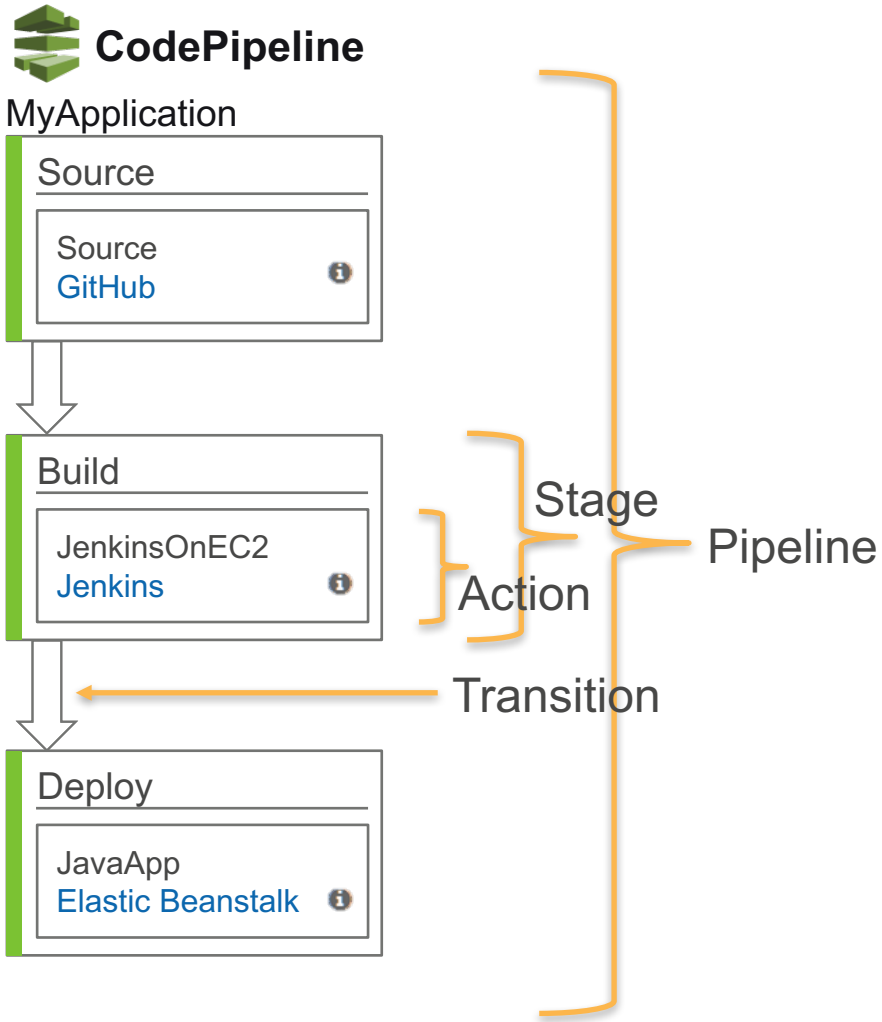


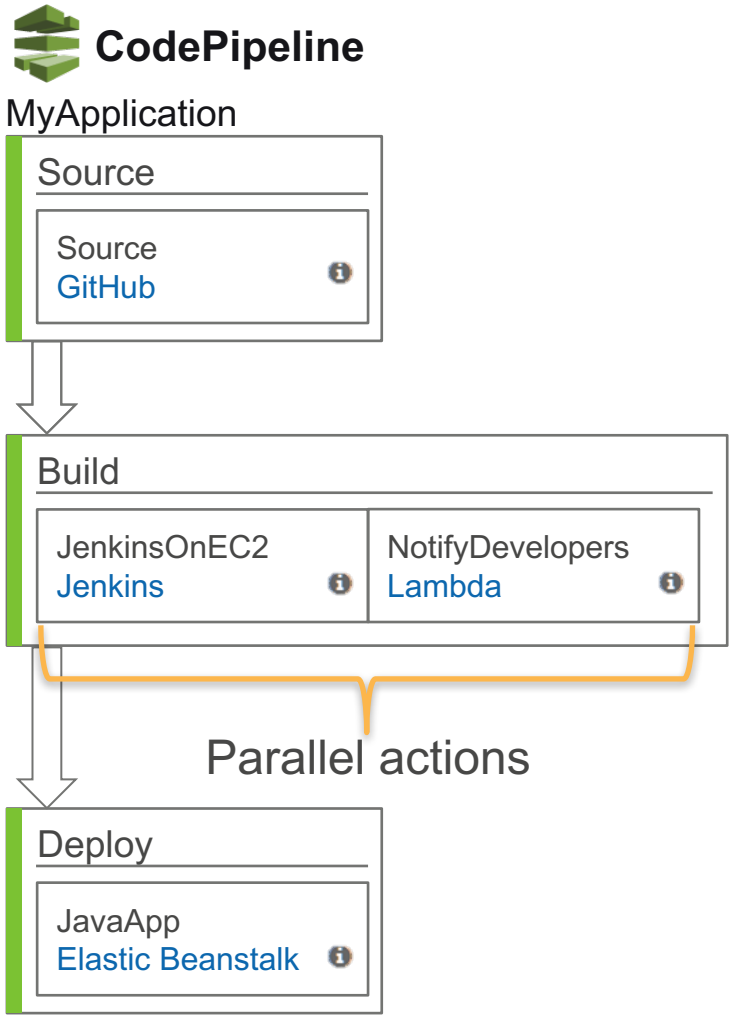
Continuous delivery service for fast and reliable application updates

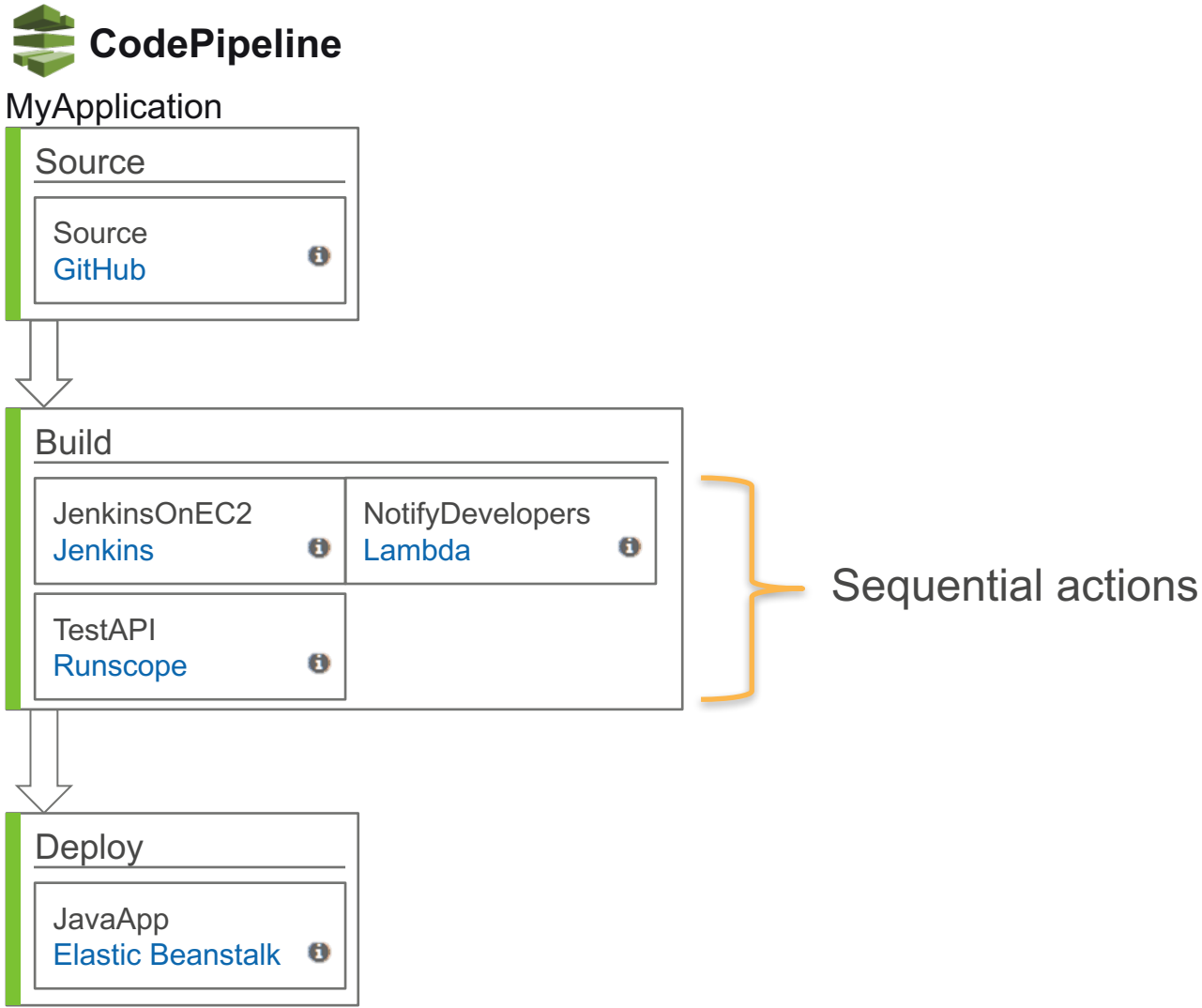
Model and visualize your software release process

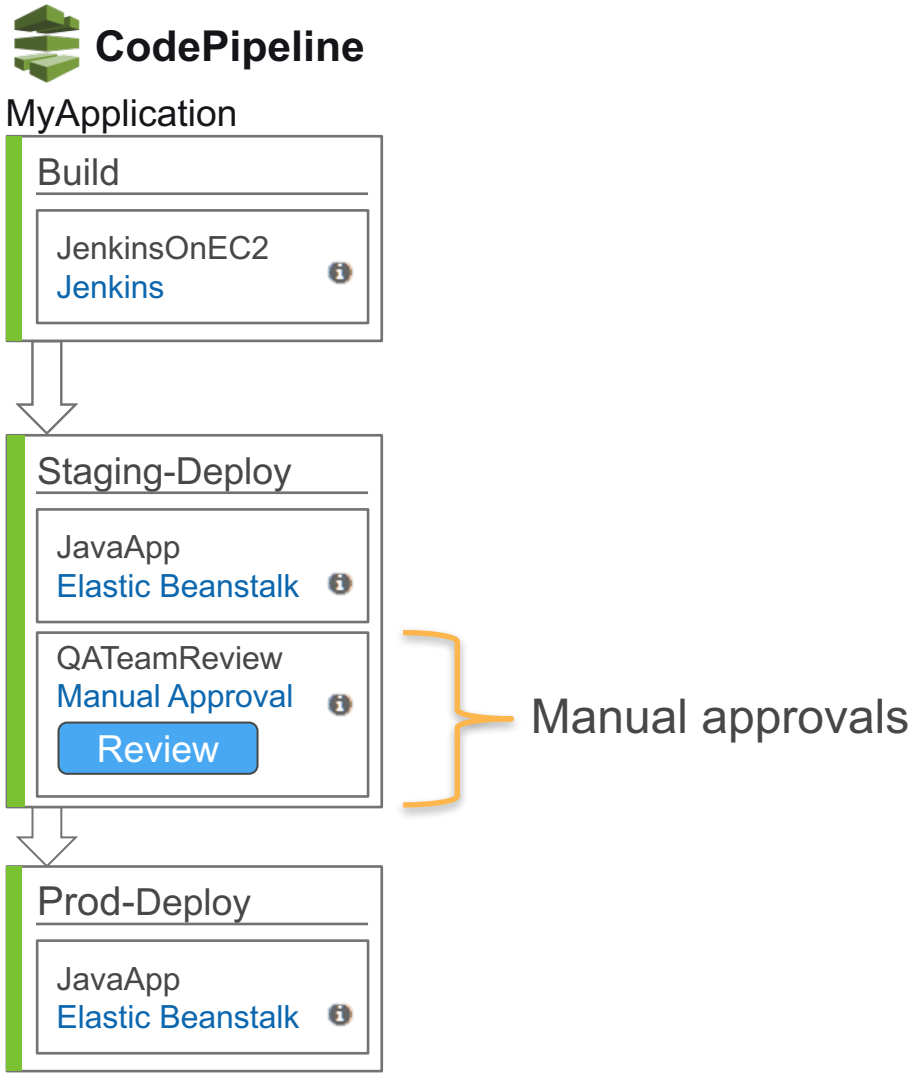
Builds, tests, and deploys your code every time there is a code change

Integrates with third-party tools and AWS

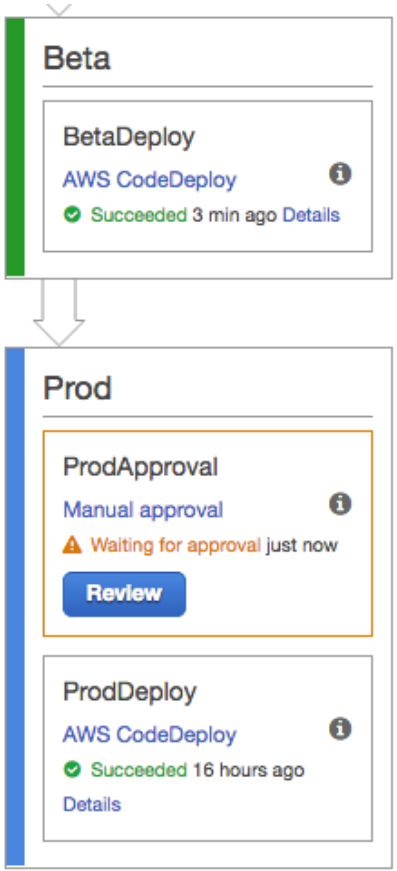





CodePipeline
MyApplication
Source
Source
GitHub
Build
JenkinsOnEC2
Jenkins
NotifyDevelopers
Lambda
TestAPI
Runscope
Sequential actions
Deploy
JavaApp
Elastic Beanstalk



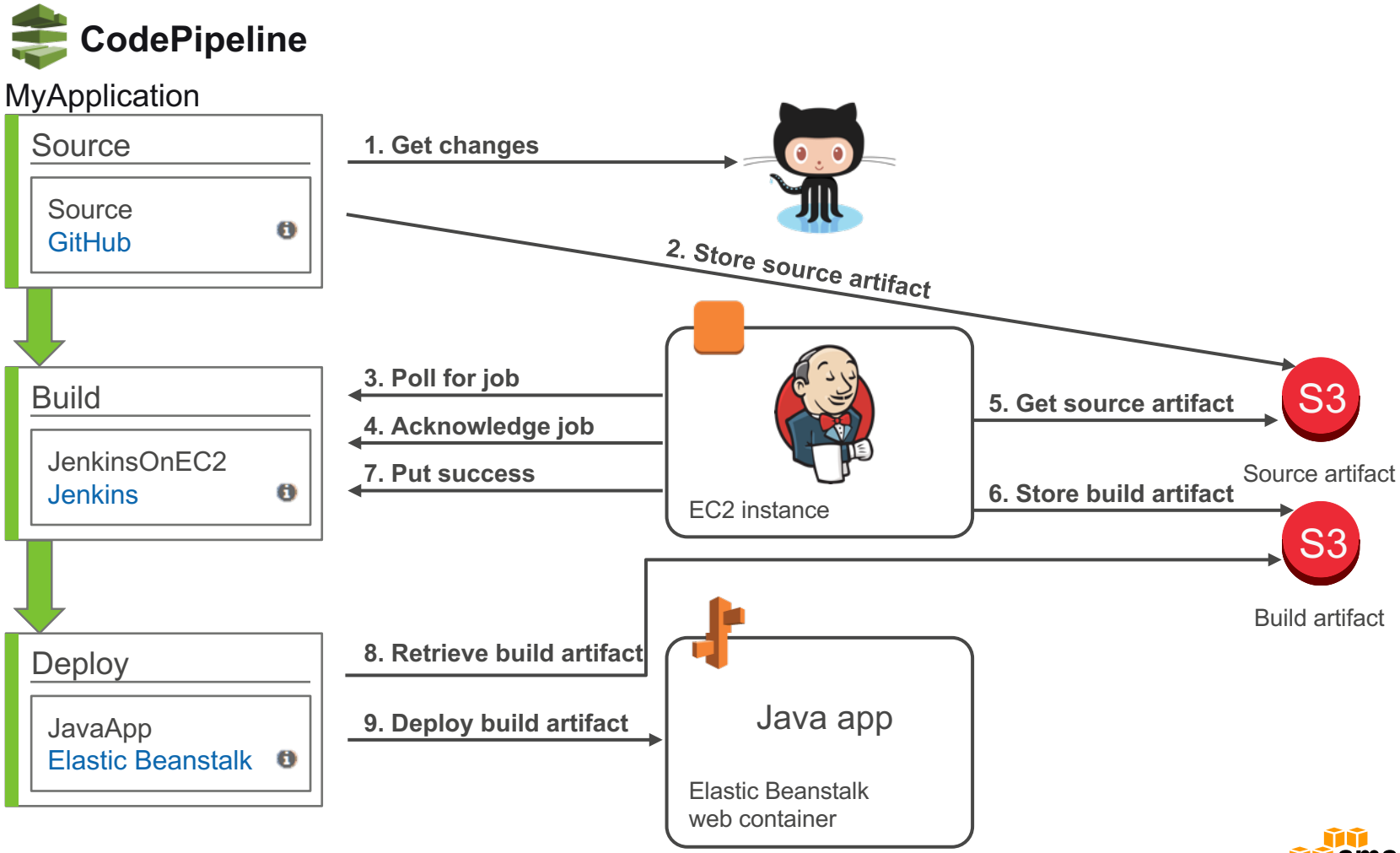
Manual approvals



You can add a manual approval at the point where you want the pipeline to stop running until someone approves or rejects the revision in progress.

- Pipeline will stop executing when it has reached the point at which you set the approval action
- Pipeline execution resumes only when the action has been approved
- Approval action managed with AWS Identity and Access Management (IAM) permissions
- Notify approvers in several ways including email, SMS, webhooks, and more
- Useful for manual QA actions or as part of “Canary” deploy models

amazon web services


CodePipeline
MyApplication
Source
Source
GitHub
Build
JenkinsOnEC2
Jenkins
Deploy
JavaApp
Elastic Beanstalk
1. Get changes
2. Store source artifact
3. Poll for job
4. Acknowledge job
7. Put success
EC2 instance
5. Get source artifact
6. Store build artifact
S3
Source artifact
S3
Build artifact
8. Retrieve build artifact
9. Deploy build artifact
Java app
Elastic Beanstalk
web container

We have a strong partner list, and it's growing

GitHub

CloudBees

Apica

XebiaLabs
Deliver Faster

BlazeMeter

Jenkins

Ghost Inspector

Solano Labs

HPE StormRunner

TeamCity
*beta

Runscope

amazon web services

AWS service integrations

Amazon S3

AWS CodeCommit

AWS Lambda

AWS CodeDeploy

AWS Elastic Beanstalk

AWS OpsWorks

AWS CloudFormation

Demo Jersey Serverless Application and CI/CD Pipeline

https://github.com/sctobbor/aws-serverless-jersey-example

amazon web services

Additional Resources

- SSO to AWS with AD FS and SAML
- AWS WAF Security Automations
- Setting up DynamoDB Local

amazon
web services