ApiConf

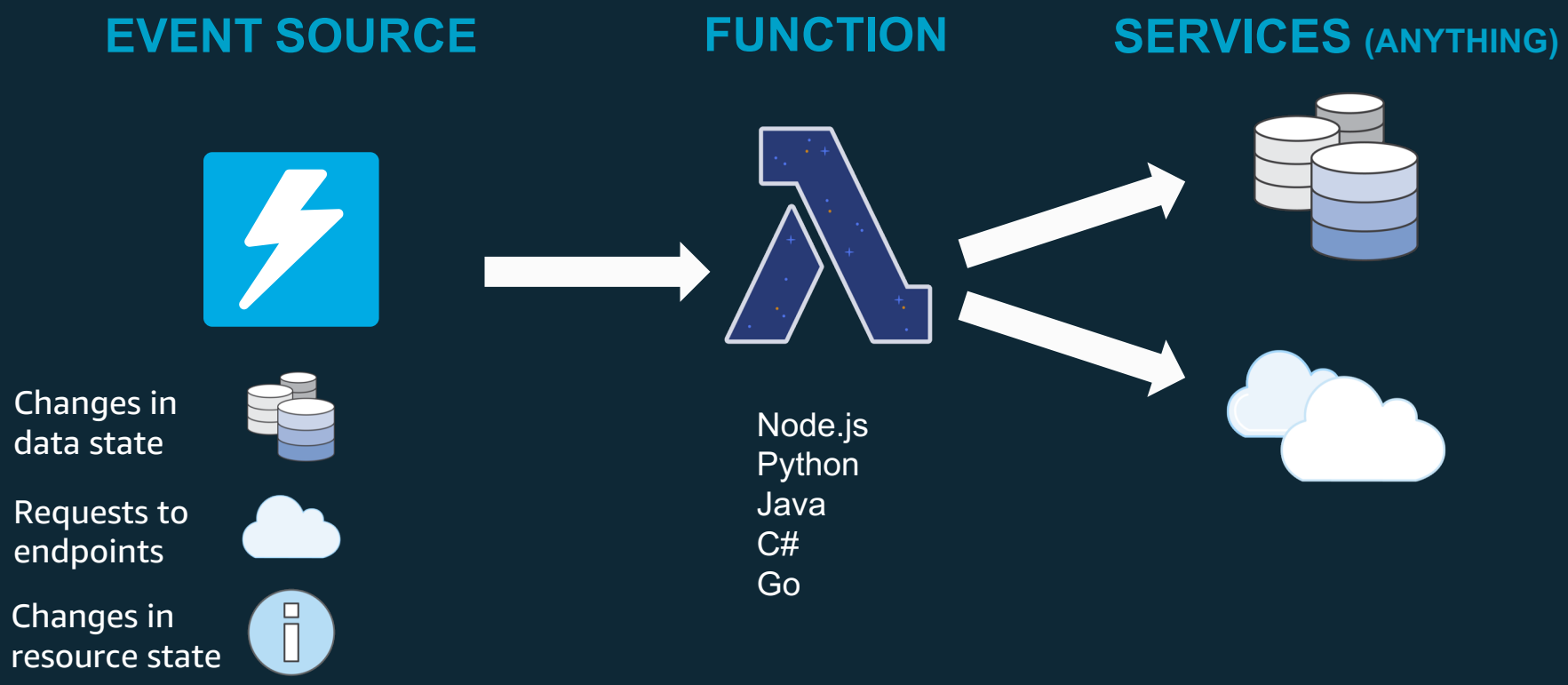
Serverless APIs on AWS

Key services and best practices

Alessandro Gambirasio
Technical Account Manager
Amazon Web Services

aws

Serverless applications



Using AWS Lambda

Bring your own code

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

Simple resource model

- Select power rating from 128 MB to 3 GB
- CPU and network allocated proportionately

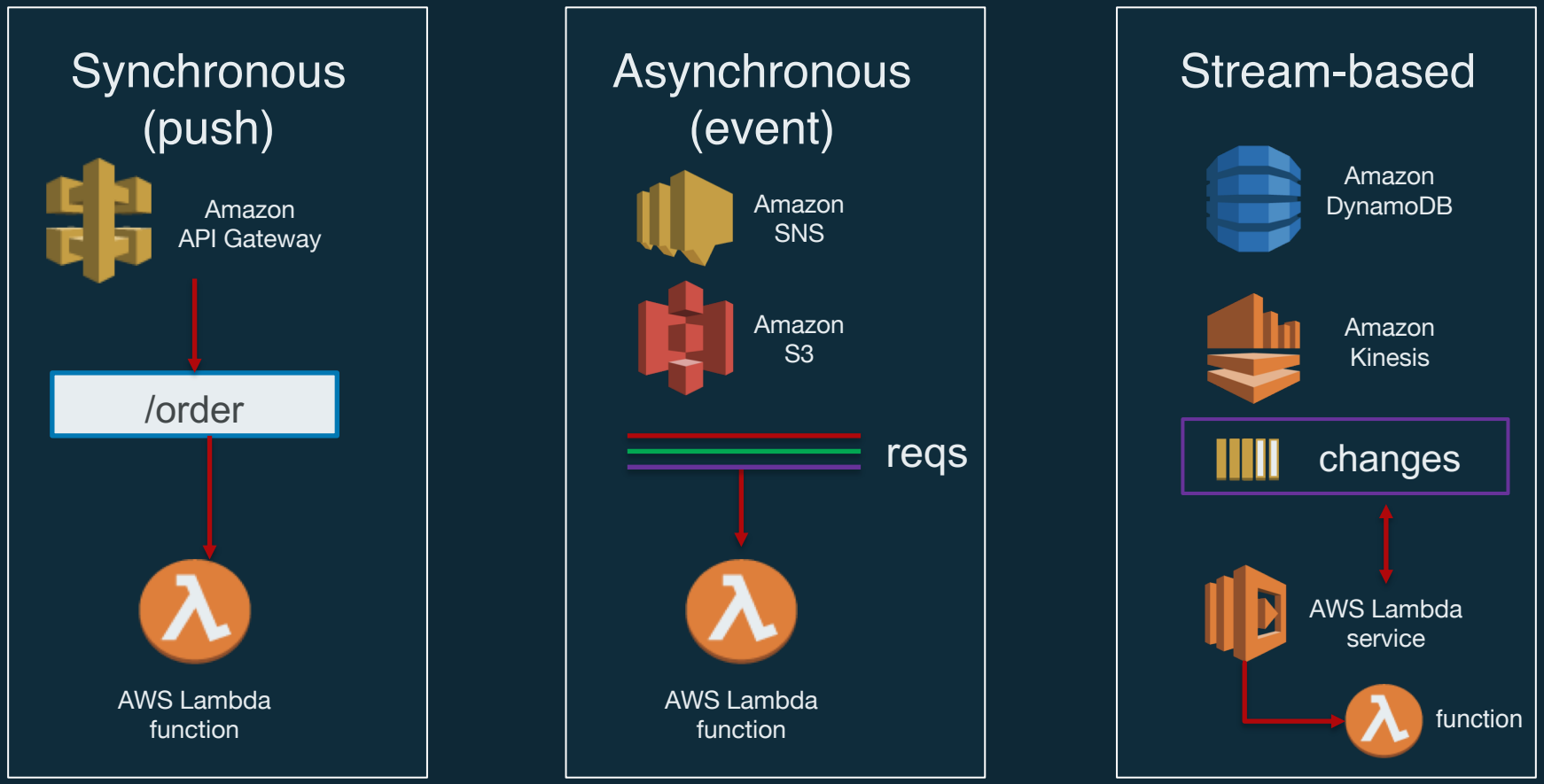
Flexible use

- Synchronous or asynchronous
- Integrated with other AWS services

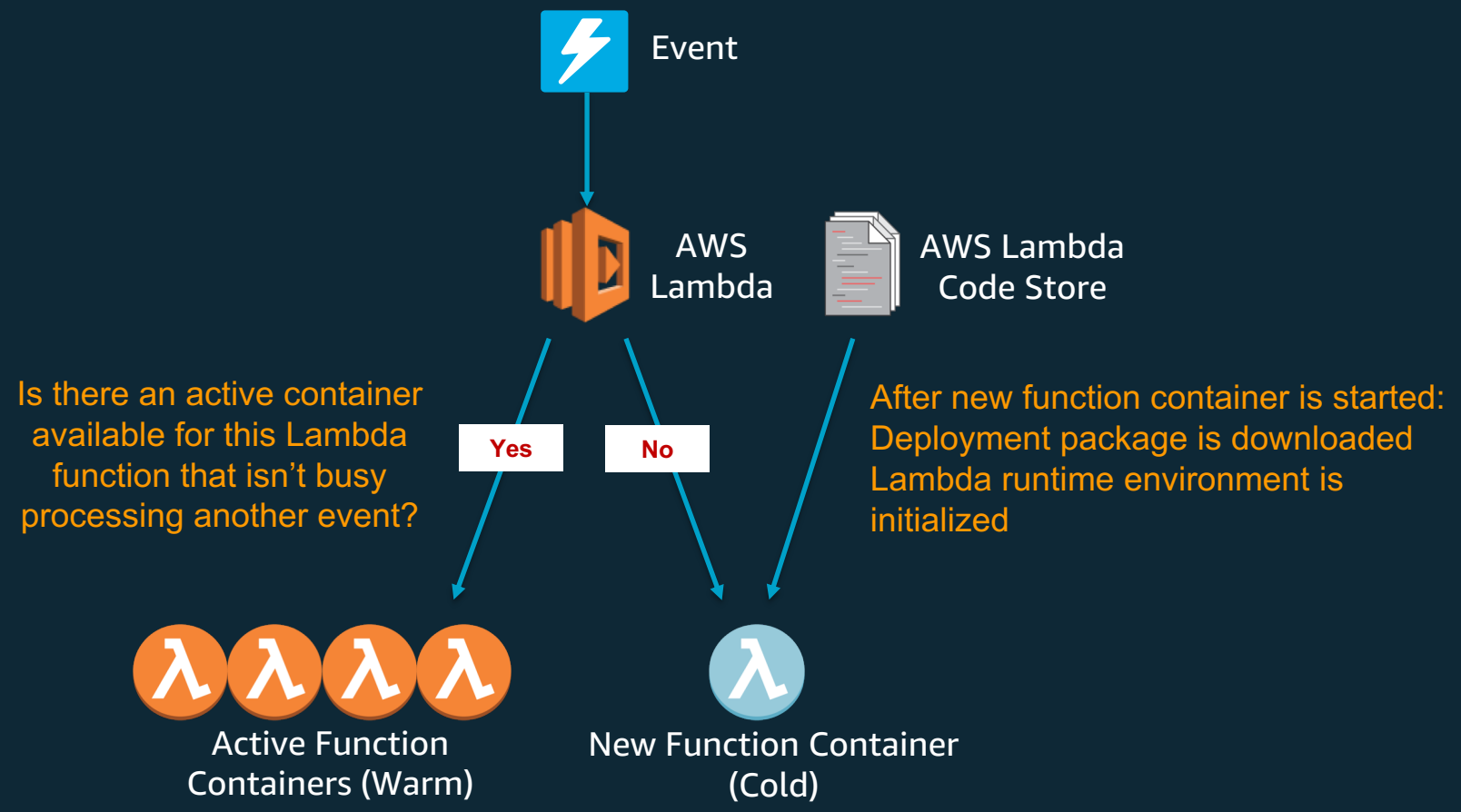
Flexible authorization

- Securely grant access to resources and VPCs
- Fine-grained control for invoking your functions

Lambda execution model



Understanding the function lifecycle



Anatomy of a Lambda function

Handler() function

Function to be executed upon invocation

Event object

Data sent during Lambda Function

Context object

Methods available to interact with runtime

```
14:31:12  Loading function
14:31:12  START RequestId: f0e306d3-723a-11e8-8a5a-4d0f248e9616 Version: $LATEST
14:31:12  Log stream name: 2018/06/17/[$LATEST]3414edd245314721990b98c7faeda37a
14:31:12  Log group name: /aws/lambda/s3sampleapiconf2018
14:31:12  Request ID: f0e306d3-723a-11e8-8a5a-4d0f248e9616
14:31:12  Mem. limits(MB): 128
14:31:13  Time remaining (MS): 1998
14:31:13  Received event: {
14:31:13  "Records": [
14:31:13  {
14:31:13  "eventVersion": "2.0",
14:31:13  "eventTime": "2018-06-17T14:30:10.966Z",
14:31:13  "requestParameters": {
14:31:13  "sourceIPAddress": "54.240.197.225"
14:31:13  },
14:31:13  "s3": {
14:31:13  "configurationId": "46091496-4dfd-49af-af9d-c8772685050a",
14:31:13  "object": {
14:31:13  "eTag": "07c0431842313cc81b68b1c0ba3ab467",
14:31:13  "sequencer": "005B267072E9A10757",
14:31:13  "key": "DummyFile.txt",
14:31:13  "size": 12
14:31:13  },
14:31:13  "bucket": {
14:31:13  "arn": "arn:aws:s3:::gambiraa-apiconf2018",
14:31:13  "name": "gambiraa-apiconf2018",
```

Understanding Lambda concurrency

Stream-based event sources for Lambda functions that process Kinesis or DynamoDB streams the number of shards is the unit of concurrency.

Event sources that aren't stream-based – each published event is a unit of work, in parallel, up to your account limits. Therefore, the number of events (or requests) these event sources publish influences the concurrency.

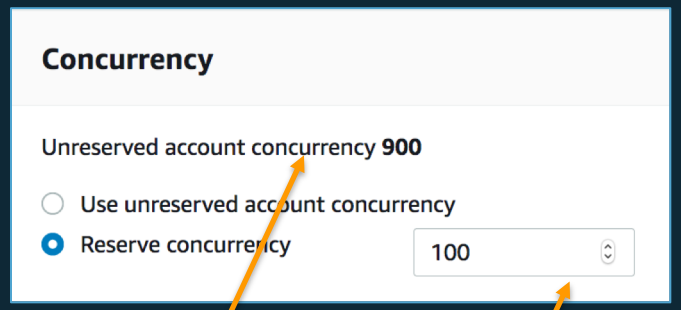
events (or requests) per second * function duration

https://docs.aws.amazon.com/lambda/latest/dg/scaling.html

Understanding Lambda concurrency

```
gambiraa$aws lambda get-account-settings

{
    "AccountLimit": {
        "CodeSizeUnzipped": 262144000,
        "UnreservedConcurrentExecutions": 900,
        "ConcurrentExecutions": 1000,
        "CodeSizeZipped": 52428800,
        "TotalCodeSize": 80530636800
    },
    "AccountUsage": {
        "FunctionCount": 2,
        "TotalCodeSize": 11694
    }
}
```



Account Level

Lambda Function Level

A wrong concurrency configuration may impact the proper execution of other functions in the same account and cause throttling.

Lambda permissions model

Security controls for execution and invocation:

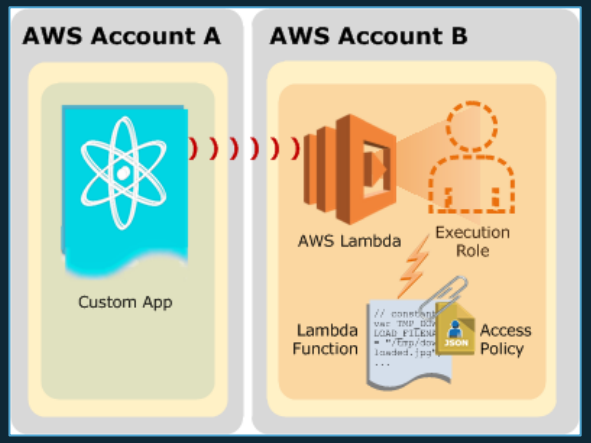
Execution policies:

- Define what AWS resources/API calls can this function access via IAM
- Used in streaming invocations
- E.g. “Lambda function A can read from DynamoDB table users”

Function policies:

- Used for sync and async invocations
- E.g. “Actions on bucket X can invoke Lambda function Z"
- Resource policies allow for cross account access

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```



Lambda functions must be idempotent

Invocations occur at least once in response to an event and functions must be idempotent to handle this.

If your function is given the same input (event) multiple times, the function MUST produce the same result.

Use a unique ID in the event like:

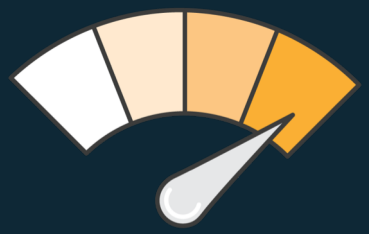
Kinesis: Records[].eventID
SNS: Records[].Sns.MessageId
API Gateway: requestContext.requestId
Scheduled CloudWatch Event: id

https://docs.aws.amazon.com/lambda/latest/dg/API_Invoke.html

aws

Fine-Grained Pricing

> Memory > Cores



Buy compute time in 100ms increments

Lambda exposes only a memory control, with the **% of CPU core and network capacity** allocated to a function proportionally.

Is your code CPU, Network or memory-bound? If so, it could be **cheaper** to choose more memory.

Fine-Grained Pricing

Stats for Lambda function that calculates **1000 times** all prime numbers **<= 1000000**

128mb	11.722965sec	$0.024628
256mb	6.678945sec	$0.028035
512mb	3.194954sec	$0.026830
1024mb	1.465984sec	$0.024638

Metrics and logging are a universal right!

- 6 Built in **metrics** for Lambda
  - Invocation Count, Invocation duration, Invocation errors, Throttled Invocation, Iterator Age, DLQ Errors

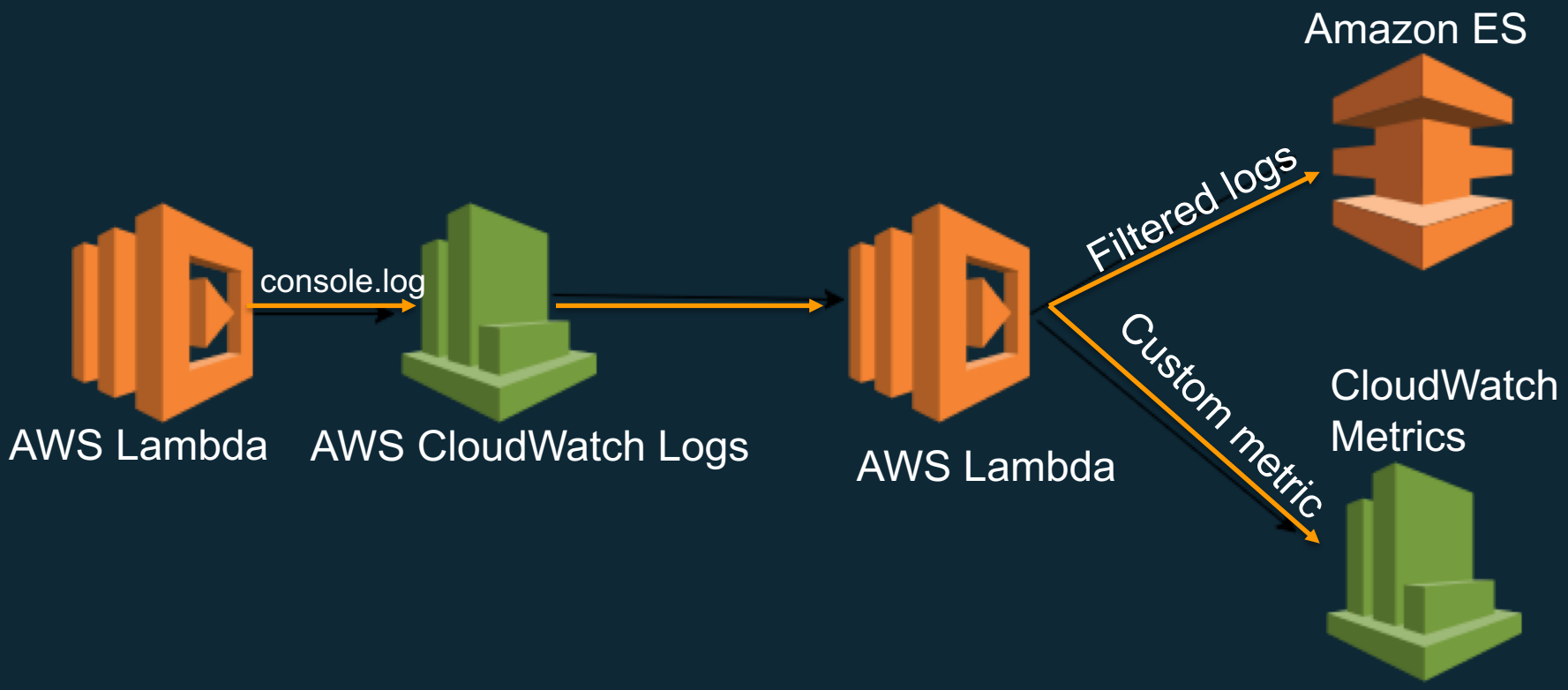
- 7 Built in **metrics** for API-Gateway
  - API Calls Count, Latency, 4XXs, 5XXs, Integration Latency, Cache Hit Count, Cache Miss Count
  - Error and Cache metrics support averages and percentiles

aws

Metrics and logging are a universal right!

- API Gateway **Logging**
  - 2 Levels of logging, ERROR and INFO
  - Optionally log method request/body content
  - Set globally in stage, or override per method
- Lambda **Logging**
  - Logging directly from your code with your language's equivalent of console.log()
  - Basic request information included
  - No Latency impact
- Log Pivots
  - Build metrics based on log filters
  - Jump to logs that generated metrics

Metrics and logging are a universal right!



Emit your own logs in custom formats – with console.log(specific format)

Process with a Lambda function - parseFormat

Emit Metric to CloudWatch Metrics & send logs to Elasticsearch

Visualize Per Customer Logs or Filter only Error logs

Lambda Environment Variables

- Key-value pairs that you can dynamically pass to your function.
- Available via standard environment variable APIs such as process.env for Node.js or os.environ for Python.
- Useful for creating environments per stage (i.e. dev, testing, production).

AWS Systems Manager – Parameter Store

```
from __future__ import print_function
import json
import boto3
ssm = boto3.client('ssm', 'us-east-1')

def get_parameters():
    response = ssm.get_parameters(
        Names=['LambdaSecureString'],WithDecryption=True
    )
    for parameter in response['Parameters']:
        return parameter['Value']

def lambda_handler(event, context):
    value = get_parameters()
    print("value1 = " + value)
    return value  # Echo back the first key value
```

Useful for: centralized environment variables, secrets control, feature flags

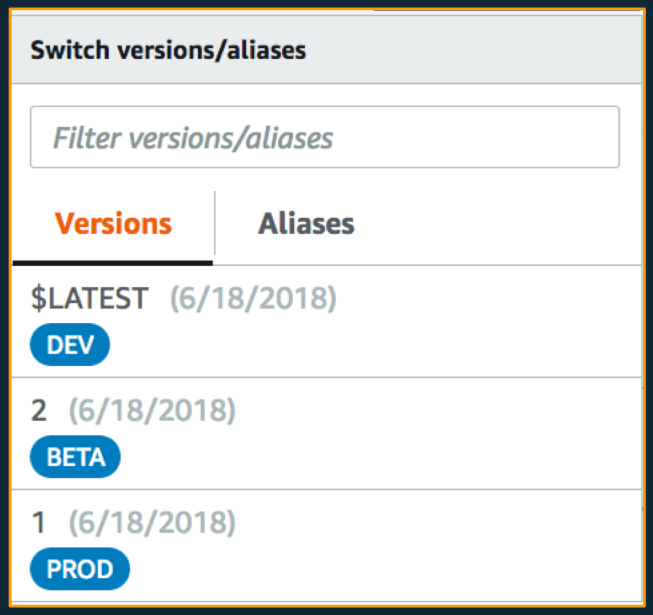
- plain-text or encrypted with KMS
- Can send notifications of changes to Amazon SNS/ AWS Lambda
- Secured with IAM and calls recorded in CloudTrail
- Available via API/SDK

Lambda Versions and Aliases

Versions = immutable copies of code + properties

Aliases = mutable pointers to versions

- ✓ Rollbacks
- ✓ Staged promotions



Lambda Alias Traffic Shifting & Safe Deployments

By default, an alias points to a single Lambda function version.

When the alias is updated to point to a different function version, incoming traffic in instantly points to the updated version.

To minimize this impact, you can implement the routing-config parameter of the Lambda alias that allows you to point to two different versions of the Lambda function and dictate what percentage of incoming traffic is sent to each version."

```
aws lambda create-alias --name alias name --function-name function-name --
function-version 1 --routing-config AdditionalVersionWeights={"2"=0.02}
```

```
aws lambda update-alias --name alias name --function-name function-name --
routing-config AdditionalVersionWeights={"2"=0.05}
```

AWS Serverless Application Model (SAM)

CloudFormation extension optimized for serverless

New serverless resource types: functions, APIs, and tables

Supports anything CloudFormation supports

Open specification (Apache 2.0)

https://github.com/awslabs/serverless-application-model

AWS Serverless Application Model (SAM)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  GetHtmlFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://sam-demo-bucket/todo_list.zip
      Handler: index.gethtml
      Runtime: nodejs4.3
      Policies: AmazonDynamoDBReadOnlyAccess
      Events:
        GetHtml:
          Type: Api
          Properties:
            Path: /{proxy+}
            Method: ANY

  ListTable:
    Type: AWS::Serverless::SimpleTable
```

Tells CloudFormation this is a SAM template it needs to “transform”

Creates a Lambda function with the referenced managed IAM policy, runtime, code at the referenced zip location, and handler as defined. Also creates an API Gateway and takes care of all mapping/permissions necessary

Creates a DynamoDB table with 5 Read & Write units

SAM Local (SAM cli)

CLI tool for local testing of serverless apps

Works with Lambda functions and “proxy-style” APIs

Response object and function logs available on local machine

Uses open source docker-lambda images to mimic Lambda’s execution environment:

- Emulates timeout, memory limits, runtimes
- Does not emulate CPU limits
- Partial API Gateway emulation (proxy calls)

https://github.com/awslabs/aws-sam-cli

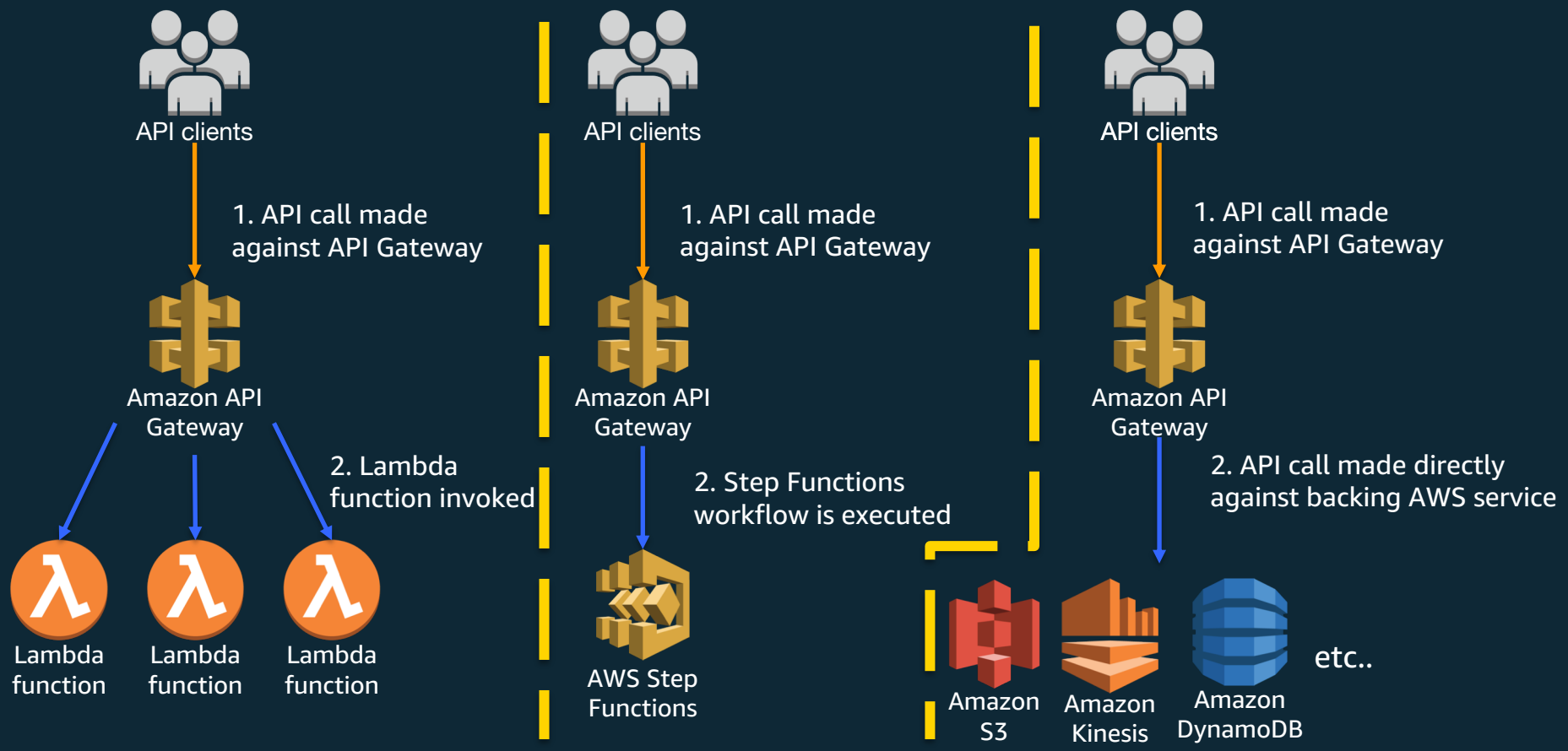
Introducing Amazon API Gateway

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale:

- Host multiple versions and stages of your APIs
- Create and distribute API Keys to developers
- Throttle and monitor requests to protect your backend
- Leverage signature version 4 to authorize access to APIs
- Request / Response data transformation and API mocking
- Reduced latency and DDoS protection through CloudFront
- Optional Managed cache to store API responses
- SDK Generation for Java, JavaScript, Java for Android, Objective-C or Swift for iOS, and Ruby
- Swagger support

aws

Amazon API Gateway patterns



Amazon API Gateway Security

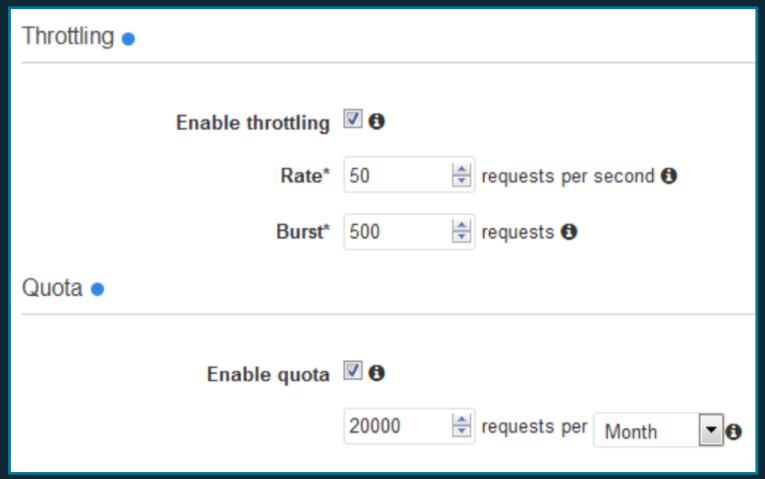
Several mechanisms for adding Authz/Authn to our API:

- IAM Permissions
  - Use IAM policies and AWS credentials to grant access
- Custom Authorizers
  - Use Lambda to validate a bearer token(Oauth or SAML as examples) or request parameters and grant access
- Cognito User Pools
  - Create a completely managed user management system

Usage Plans in API gateway

Create usage plans to control:

- Throttling — overall request rate (average requests per second) and a burst capacity
- Quota — number of requests that can be made per day, week, or month
- API/stages — the API and API stages that can be accessed



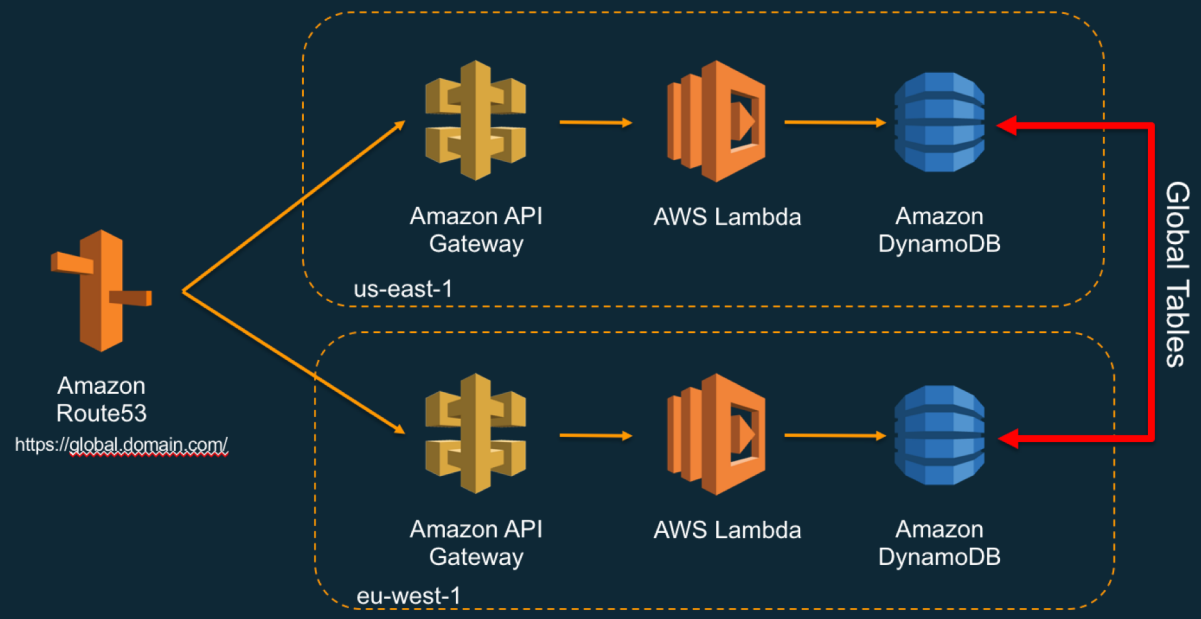
Custom domains

Run your APIs within your own DNS zone

Recommended for supporting multiple versions

- api.tampr.com/v1 -> restapi1
- api.tampr.com/v2 -> restapi2

Support for cross-region redundancy with regional API endpoints

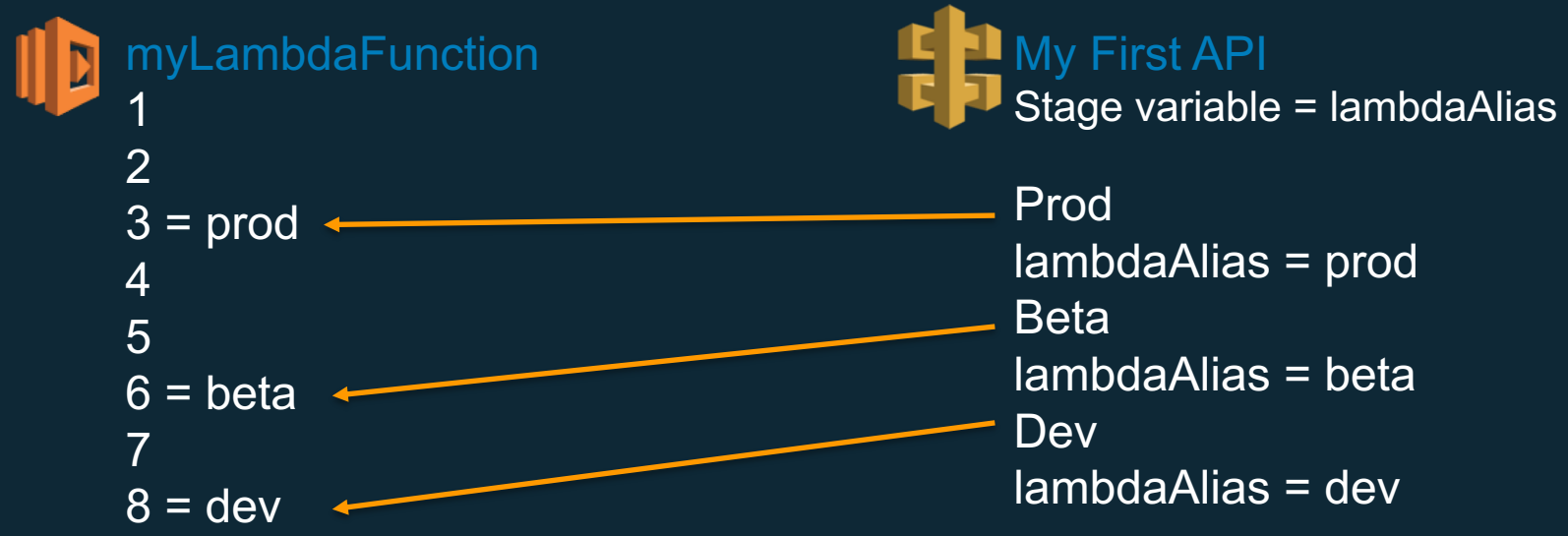


API Gateway Stage Variables

- Stage variables act like environment variables
- Use stage variables to store configuration values
- Stage variables are available in the $context object
- Values are accessible from most fields in API Gateway
  - Lambda function ARN
  - HTTP endpoint
  - Custom authorizer function name
  - Parameter mappings

Stage Variables and Lambda Aliases

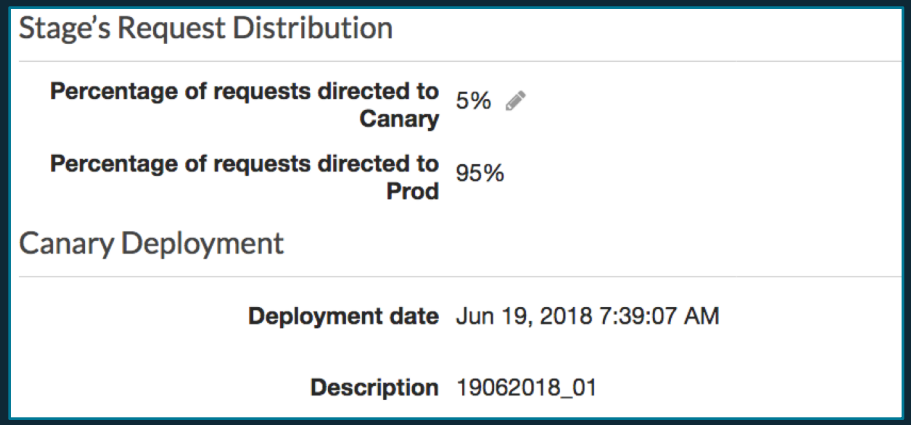
Using Stage Variables in API Gateway together with Lambda function Aliases you can manage a single API configuration and Lambda function for multiple environment stages



Amazon API Gateway Canary Support

Use canary release deployments to gradually roll out new APIs in Amazon API Gateway:

- configure percent of traffic to go to a new stage deployment
- can test stage settings and variables
- API gateway will create additional Amazon CloudWatch Logs group and CloudWatch metrics for the requests handled by the canary deployment API
- To rollback: delete the deployment or set percent of traffic to 0



Best practices recap

In event-based architecture with many downstream calls, timeouts and retries are important to get right. Be aware of service throttling.

Have no timeouts at all, and a downstream API being down could hang your whole microservice. Set your maximum invocation time for Lambda functions and integration timeout (1–29 s) per method on API gateway.

Limit number of retries and employ exponential backoff to avoid resource exhaustion and backlog.

Duplicates may happen; code must be idempotent.

aws

Best practices recap

Use synchronous execution when response is needed. The invoking application is responsible for retries.

Use asynchronous execution when no response is needed.

Create and enable one DLQ per function—SQS or SNS.

Externalize authorization to IAM roles whenever possible.

Externalize configuration. DynamoDB is great for this.

Make sure your downstream setup "keeps up" with Lambda scaling. Limit concurrency when talking to relational databases.

aws

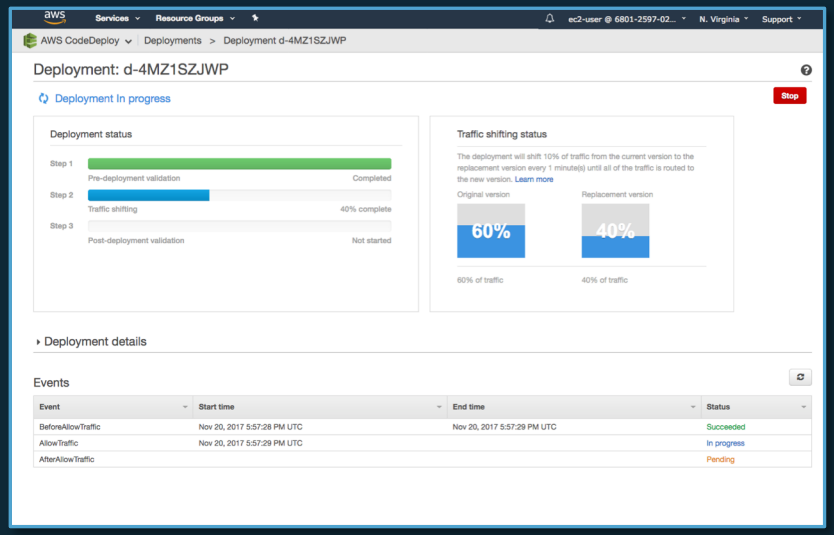
AWS CodeDeploy + Lambda

Uses AWS SAM to deploy serverless applications

Supports Lambda Alias Traffic Shifting enabling canaries and blue|green deployments

Can rollback based on CloudWatch Metrics/Alarms

Pre/Post-Traffic Triggers can integrate with other services (or even call Lambda functions)



AWS CodeDeploy + Lambda

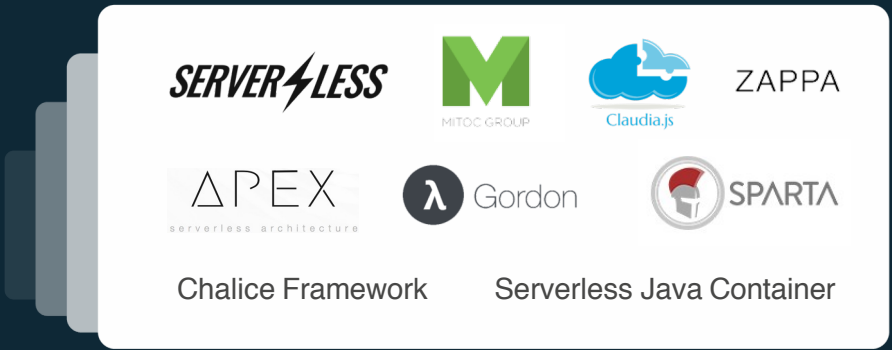
CodeDeploy comes with a number of added capabilities:

- Custom deployment configurations. Examples:
  - “Canary 5% for 1 hour”
  - “Linear 20% every 1 hour”
- Events via SNS on success/failure/rollback
- Console with visibility on deploy status, history, and rollbacks.

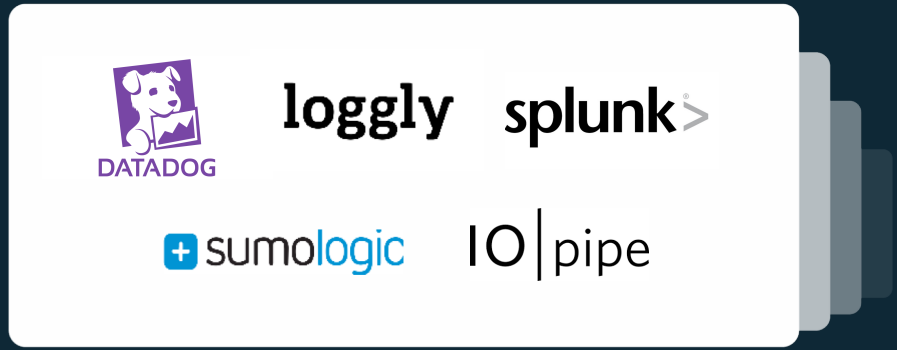
Serverless Ecosystem

Build and CI/CD

Applications and Deployment



Logging and Monitoring



aws

See you at the AWS desk.

Alessandro Gambirasio
Technical Account Manager
Amazon Web Services