AWS Lambda Serverless Applications

Rohit Bhardwaj

Hands-on Director, Architecture, Salesforce

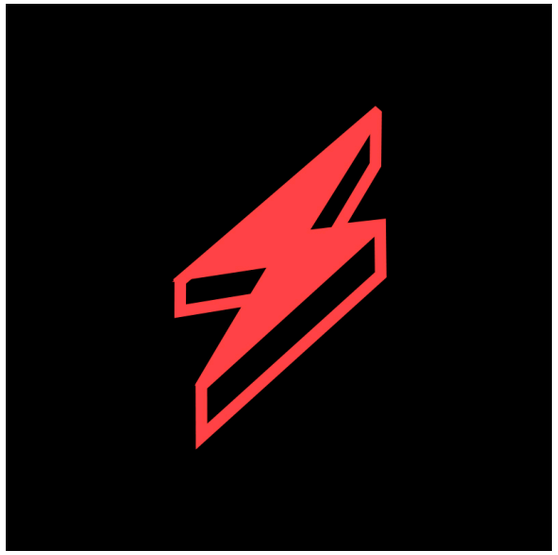
Founder: ProductiveCloudInnovation.com

Twitter: rbhardwaj1

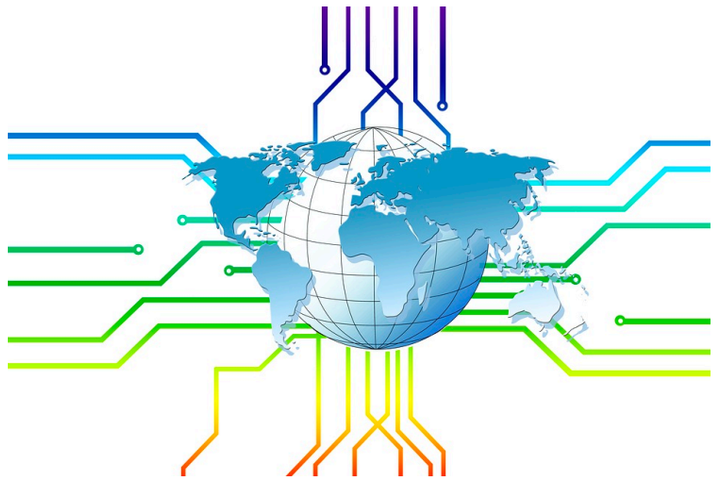
LinkedIn: www.linkedin.com/in/rohit-bhardwaj-cloud

https://tinyurl.com/NFJSAWSLambda

https://www.productivecloudinnovation.com/lessons



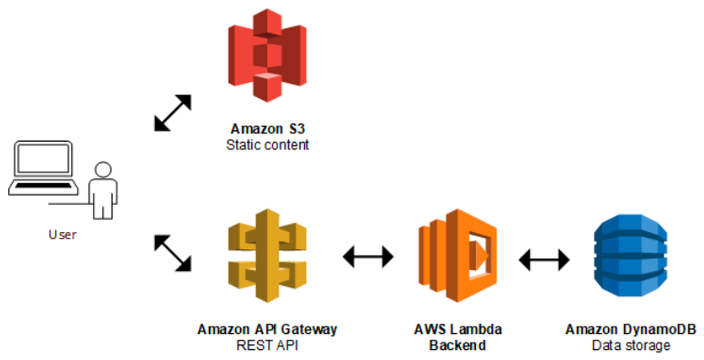
Serverless



Serverless Architecture

AWS Lambda

Amazon API Gateway

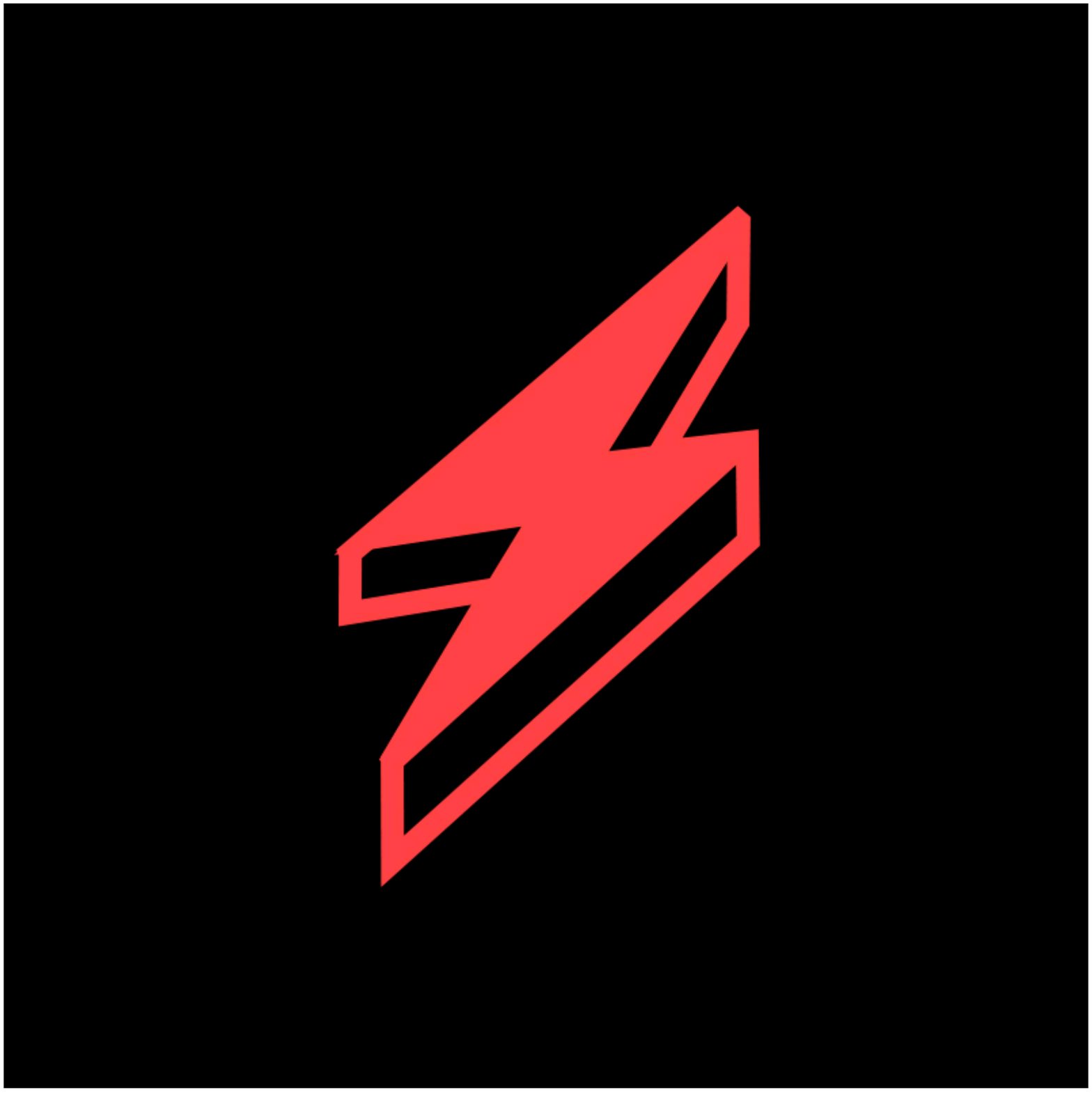


AWS Applications

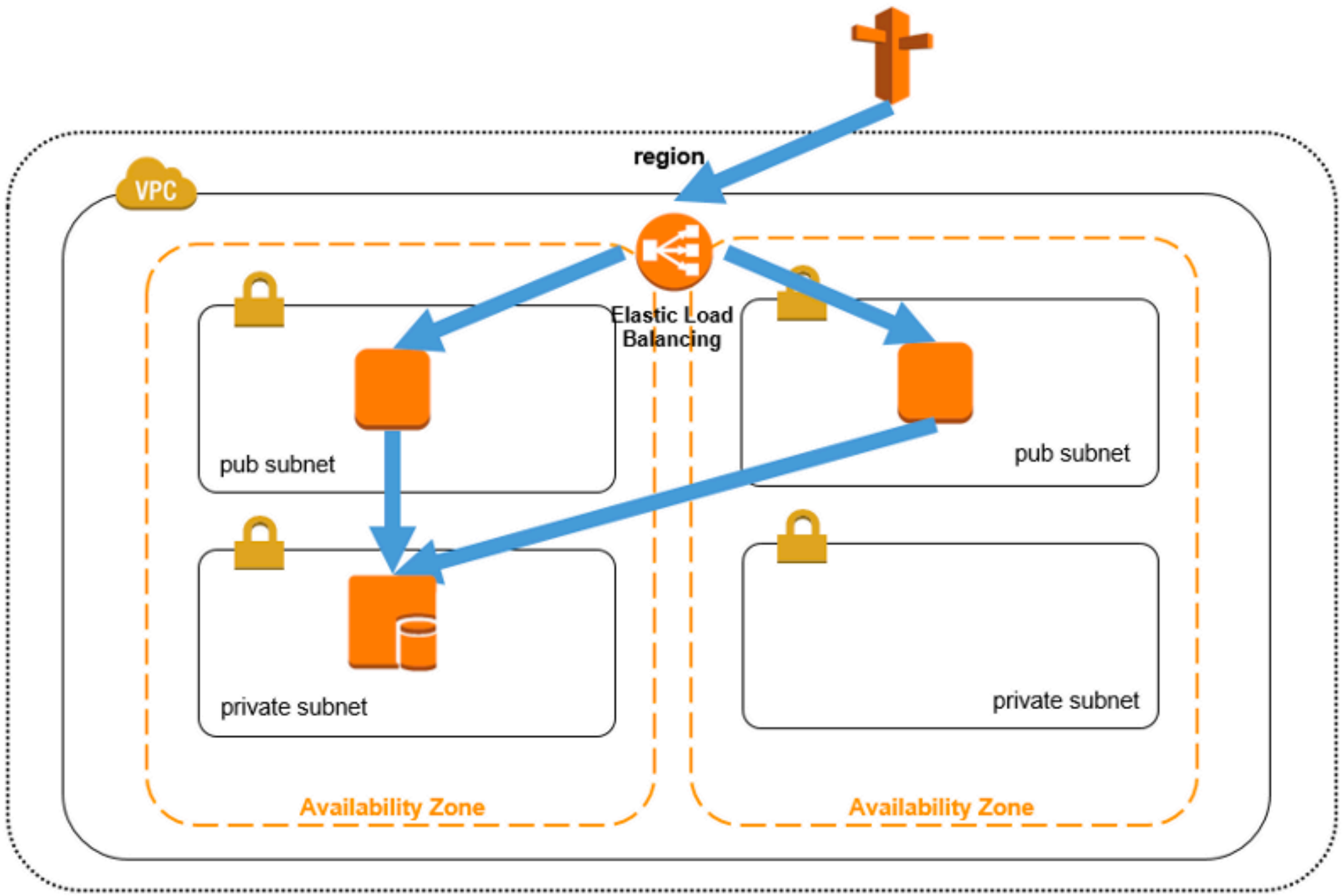
AWS Step Functions

Serverless Applications

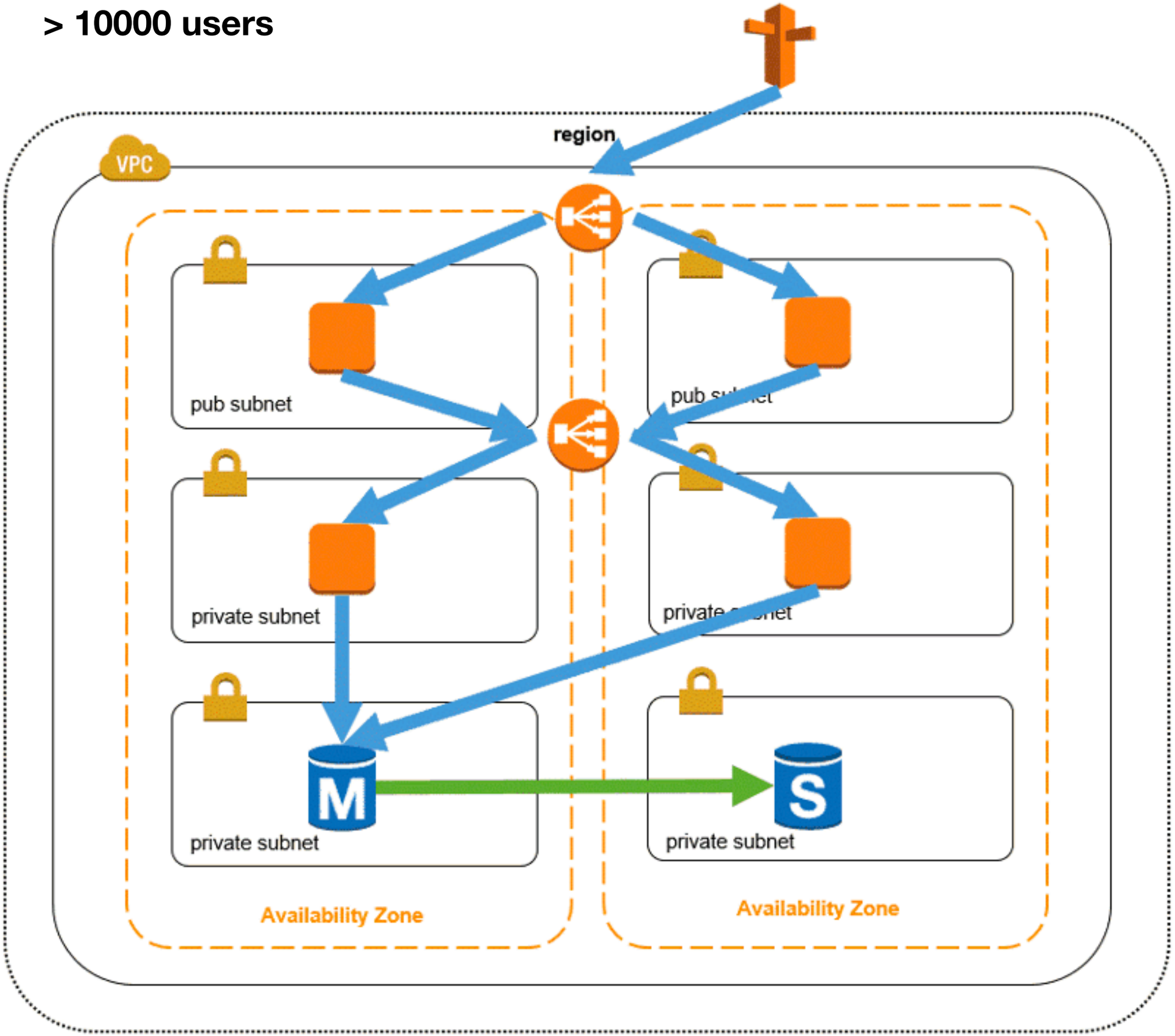
Serverless Lens



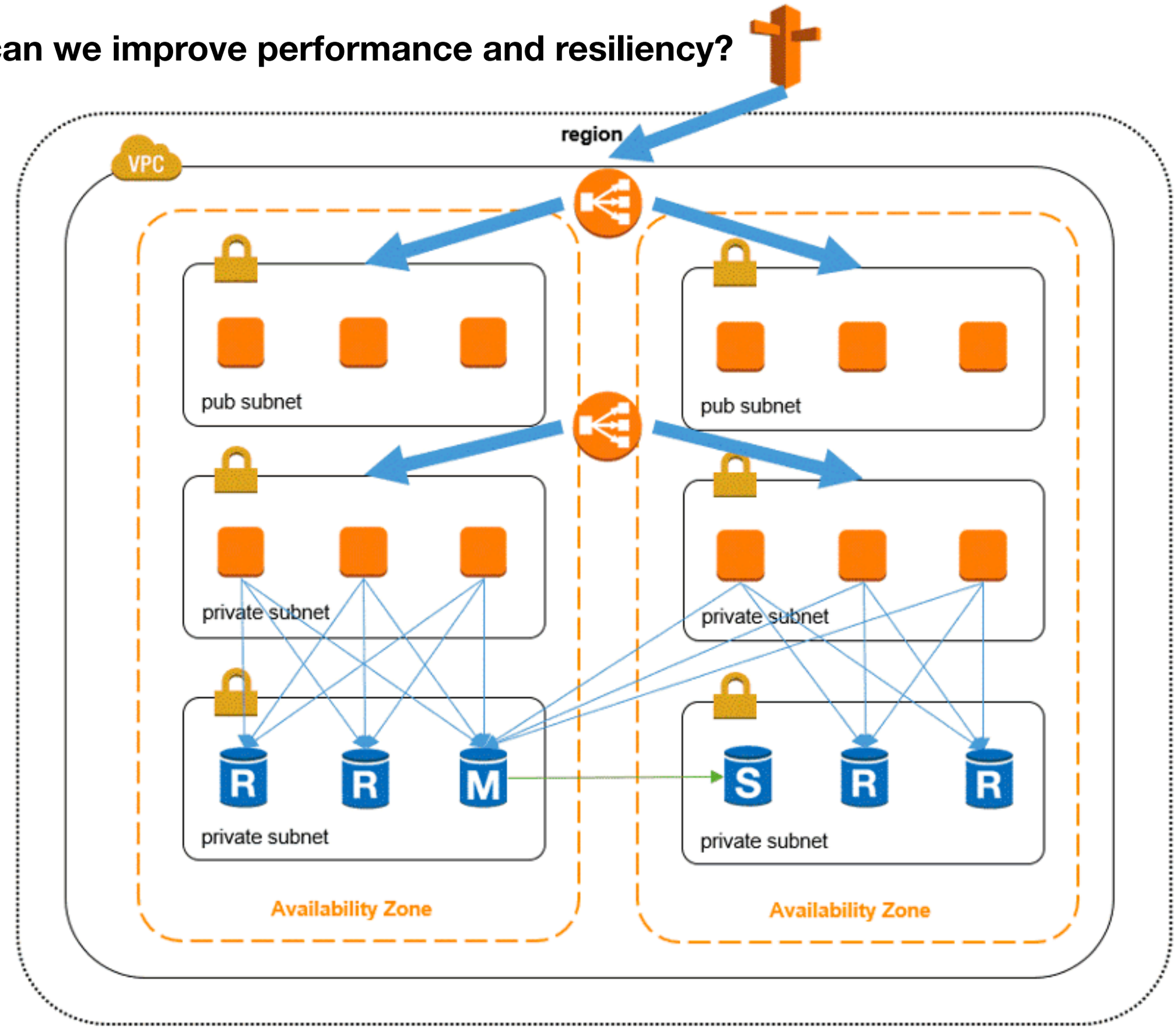
Serverless


region
VPC
Elastic Load Balancing
pub subnet
pub subnet
private subnet
private subnet
Availability Zone
Availability Zone

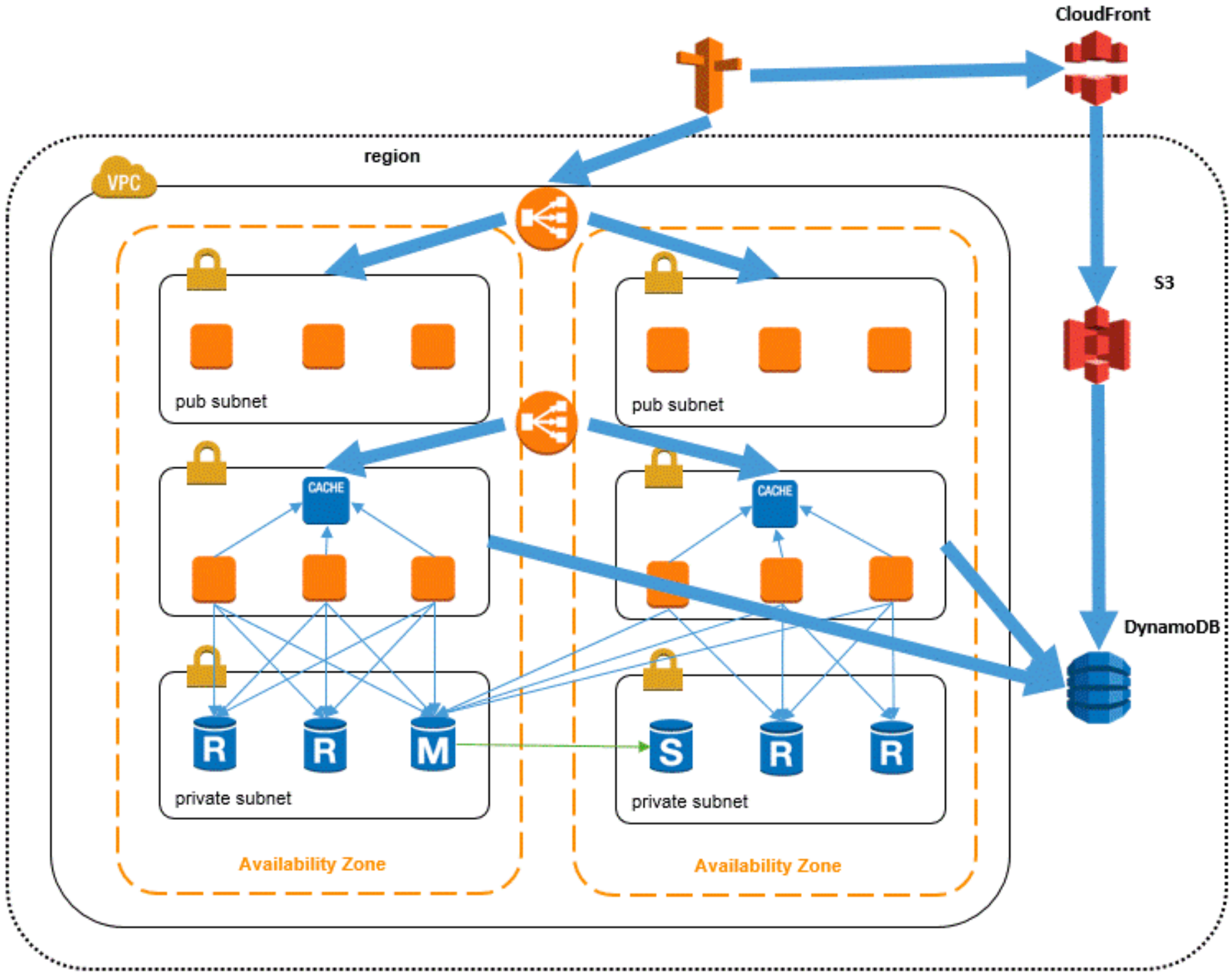
> 10000 users


region
VPC
pub subnet
pub subnet
private subnet
private subnet
private subnet
private subnet
M
S
Availability Zone
Availability Zone

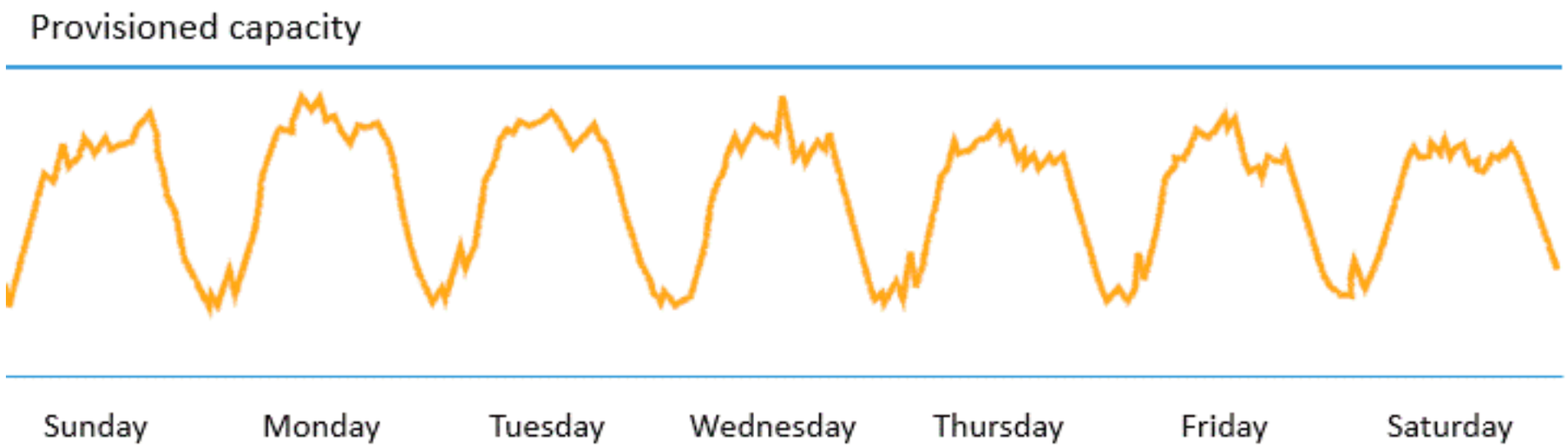
How can we improve performance and resiliency?



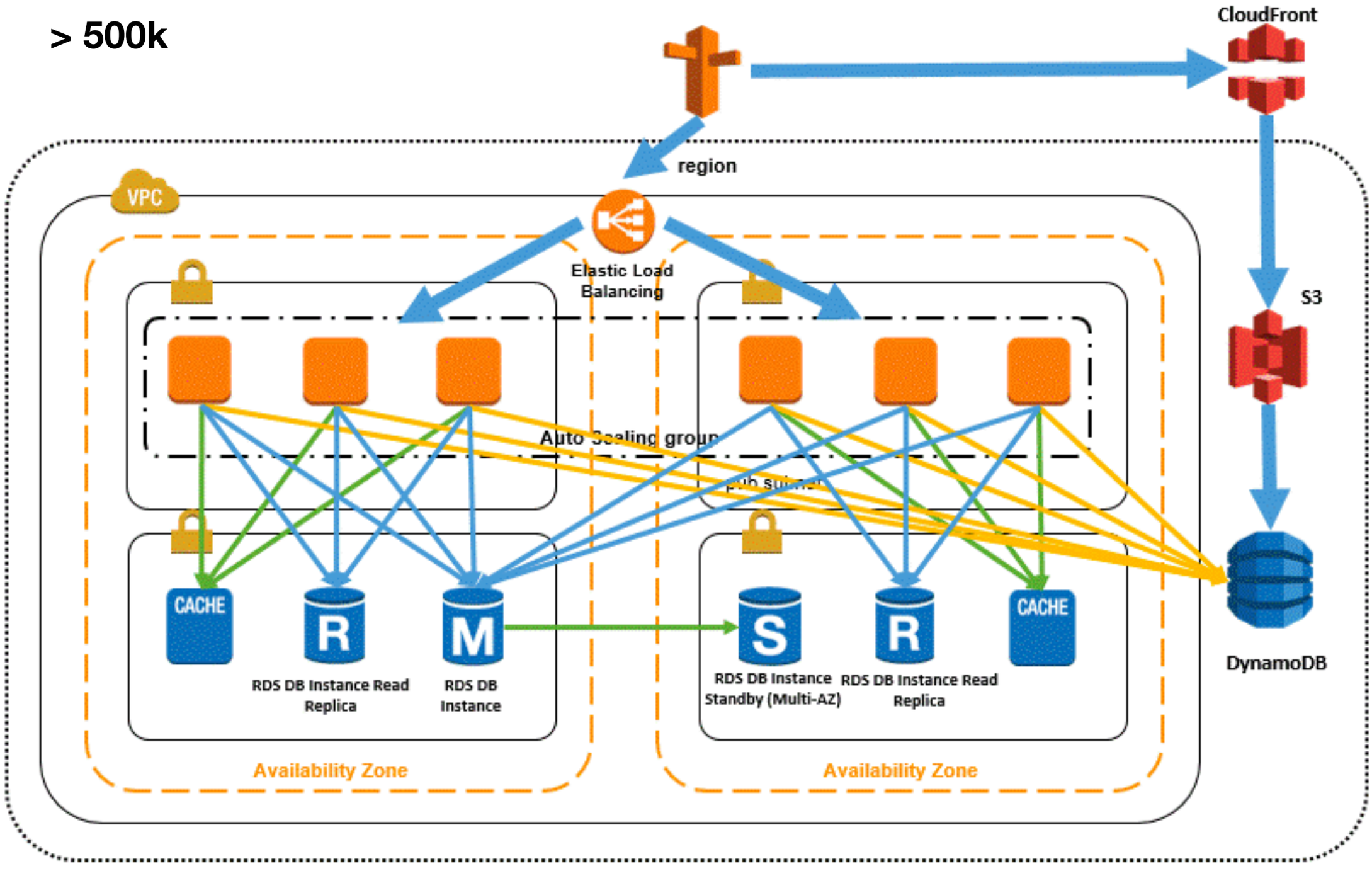
How to take care of variable load?



> 500k

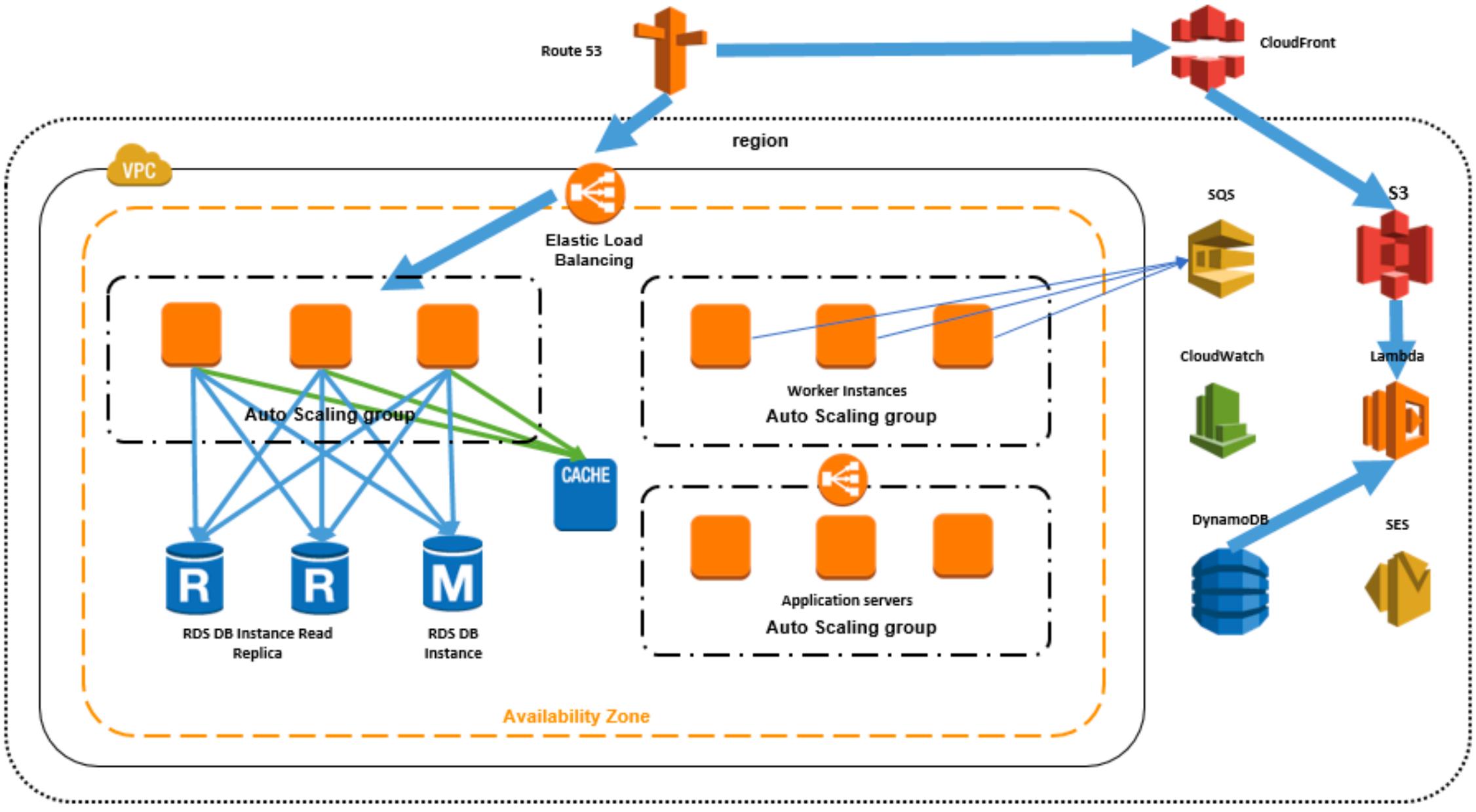

Provisioned capacity
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday

> 500k

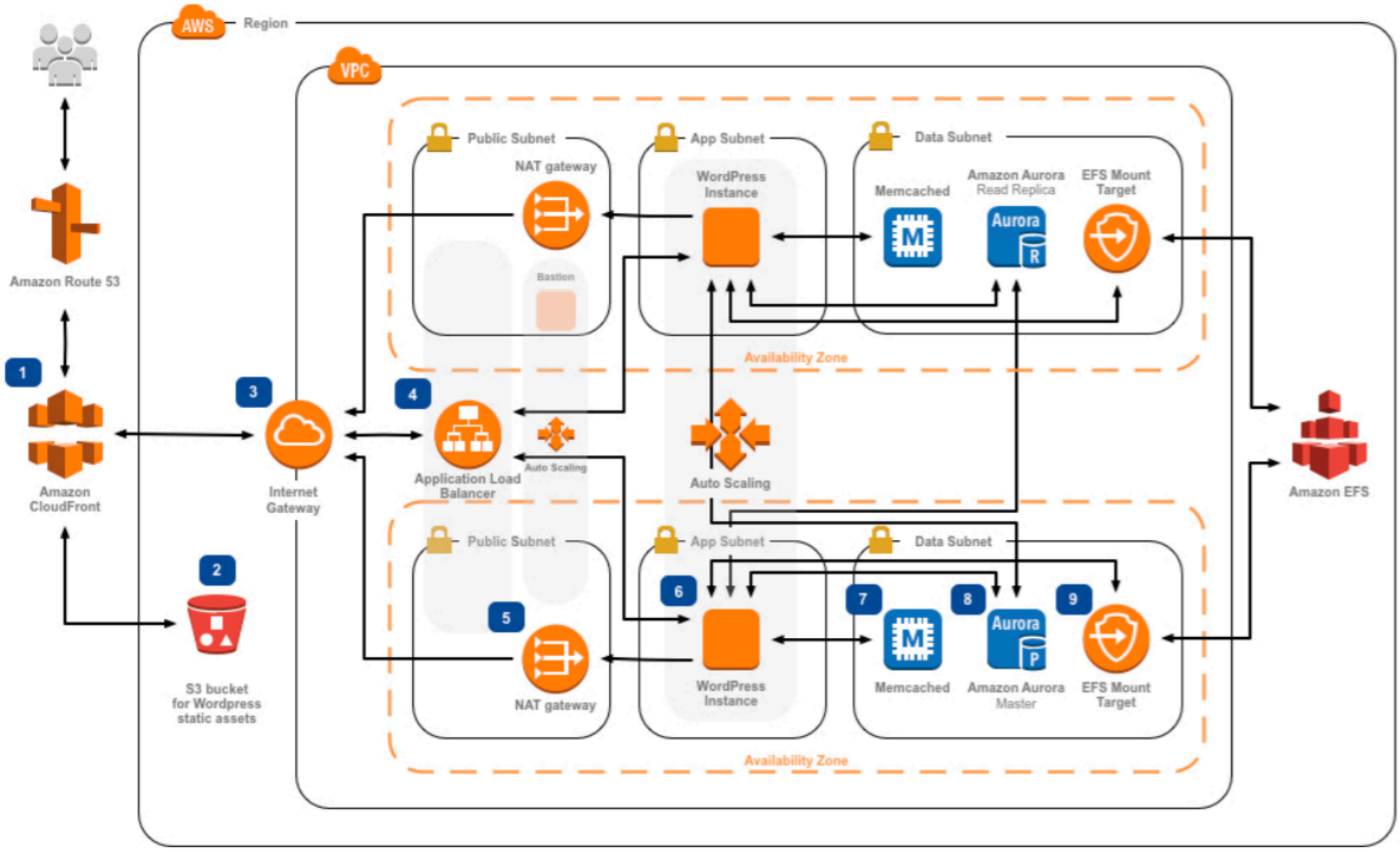

CloudFront
region
VPC
Elastic Load Balancing
S3
Auto Scaling group
CACHE
R
M
S
R
CACHE
RDS DB Instance Read Replica
RDS DB Instance
RDS DB Instance Standby (Multi-AZ)
RDS DB Instance Read Replica
DynamoDB
Availability Zone
Availability Zone

What else can we do to improve and scale?

> 1 Million

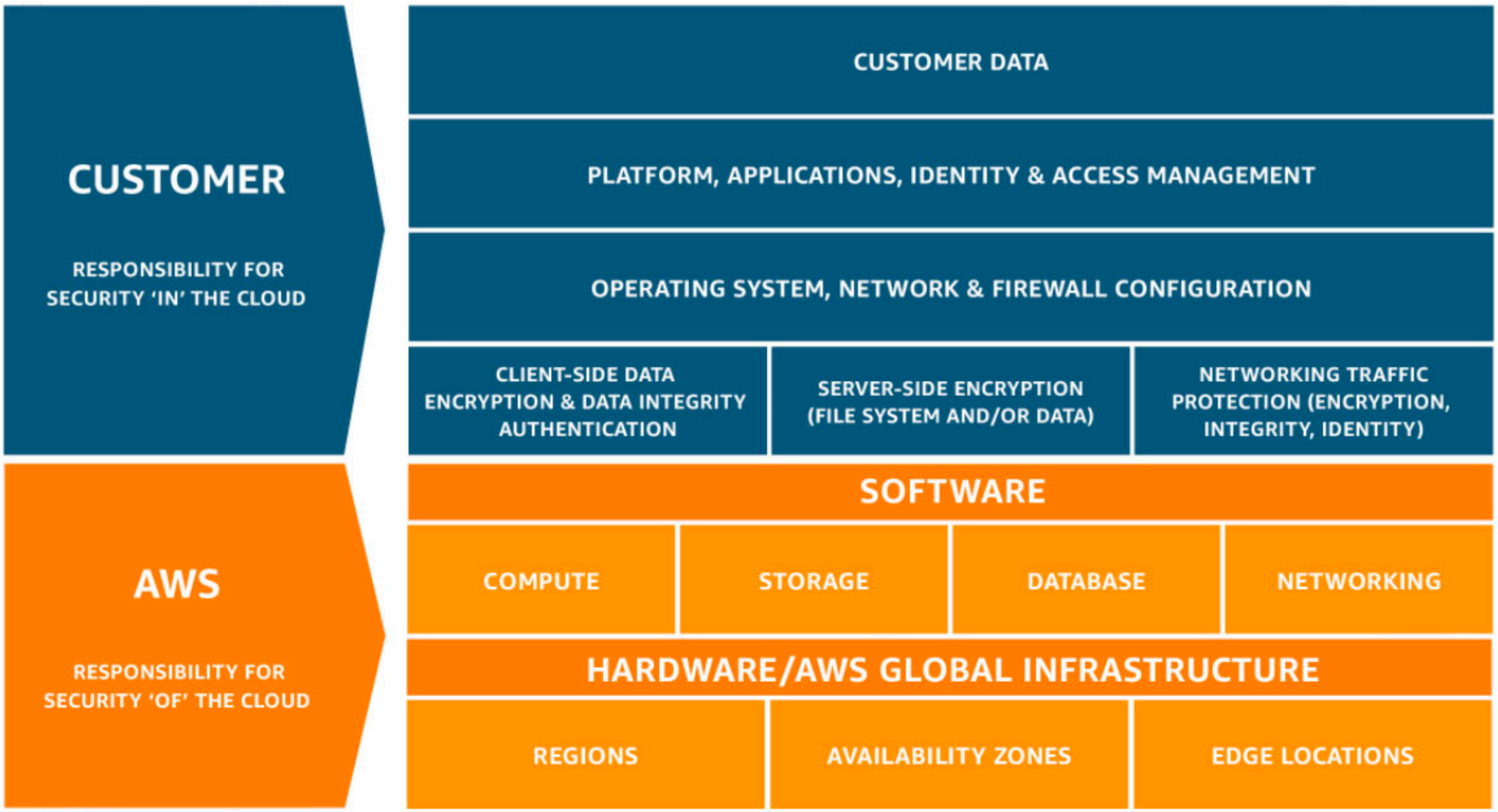


AWS Best Practices



https://aws.amazon.com/blogs/architecture/wordpress-best-practices-on-aws/

Shared responsible model



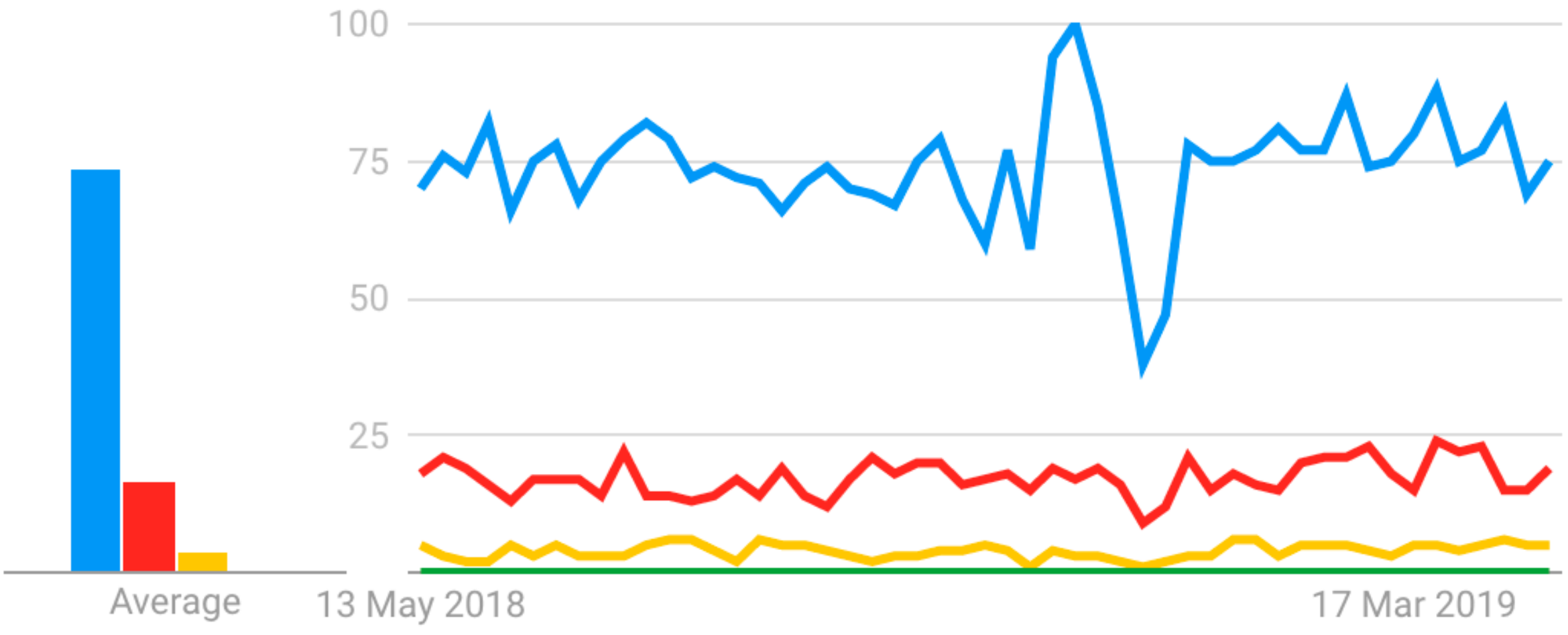
https://aws.amazon.com/compliance/shared-responsibility-model/

Interest over time

Google Trends

"AWS Lambda" "Azure Functions" "Google Cloud Functions"

Alibaba function compute



Serverless martket offerings

	amazon web services	Microsoft Azure	Google Cloud	Alibaba Cloud
Functions Event-driven serverless compute platform	AWS Lambda	Azure Functions	Cloud Functions	Function compute
API Management Create, publish, and secure APIs	API Gateway	API Management	Cloud Endpoints	Alibaba API Gateway
Storage Storage of objects	Amazon S3	Azure Storage	Cloud Storage	Alibaba Object Storage Service
Database services	Amazon DynamoDB	Azure Cosmos DB	Cloud BigTable	ApsaraDB
Security and Access Control	Amazon Cognito	Azure Active Directory	Google Security Model	Resource Access Management
Orchestration	AWS Step Functions	Logic Apps	App Engine	

https://medium.com/@pablo.iorio/serverless-architectures-iii-market-offerings-aws-azure-google-and-alibaba-67c49d1a691d

Pablo Iorio

serverless

framework

the easy & open way to build serverless applications

aws Azure Tencent Cloud Google Cloud CLOUDFLARE Alibaba Cloud twilio

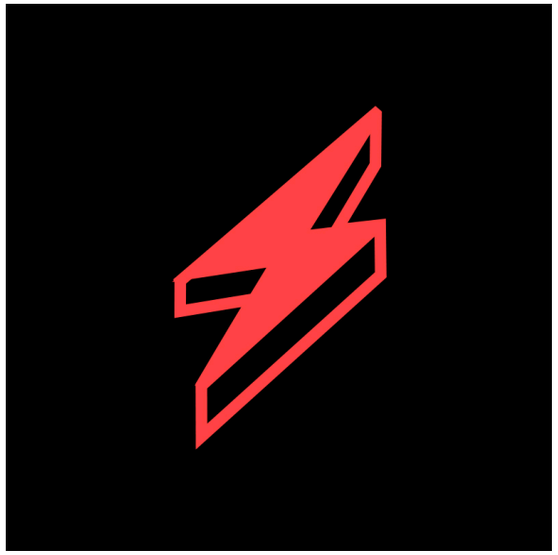
serverless


serverless framework
FULLSTACK
Express.js
React
DynamoDB

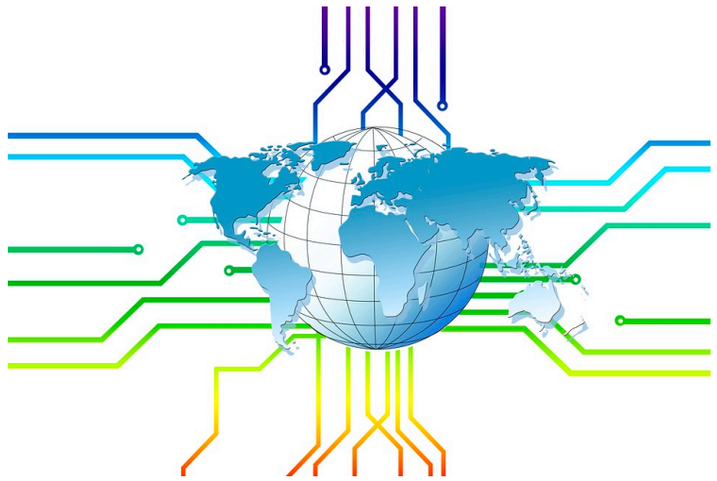

serverless framework
Express.js


serverless framework
WEBSITE

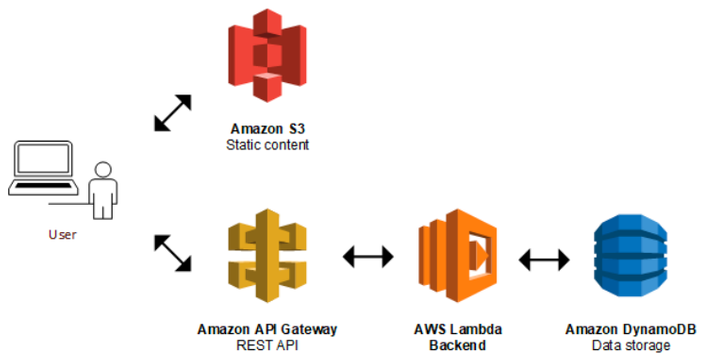

serverless framework
DYNAMODB



Serverless Framework



Serverless Architecture



AWS Applications

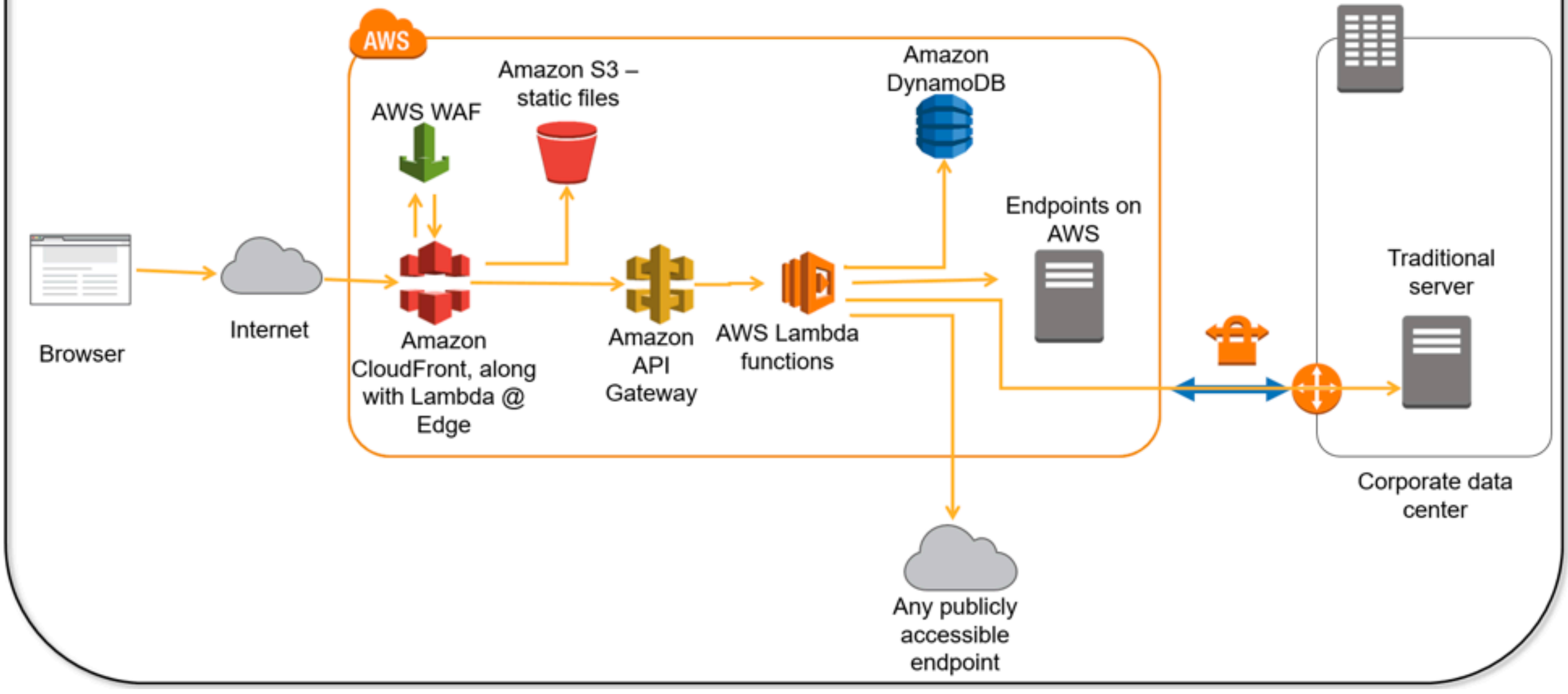
AWS Step Functions

Serverless Applications

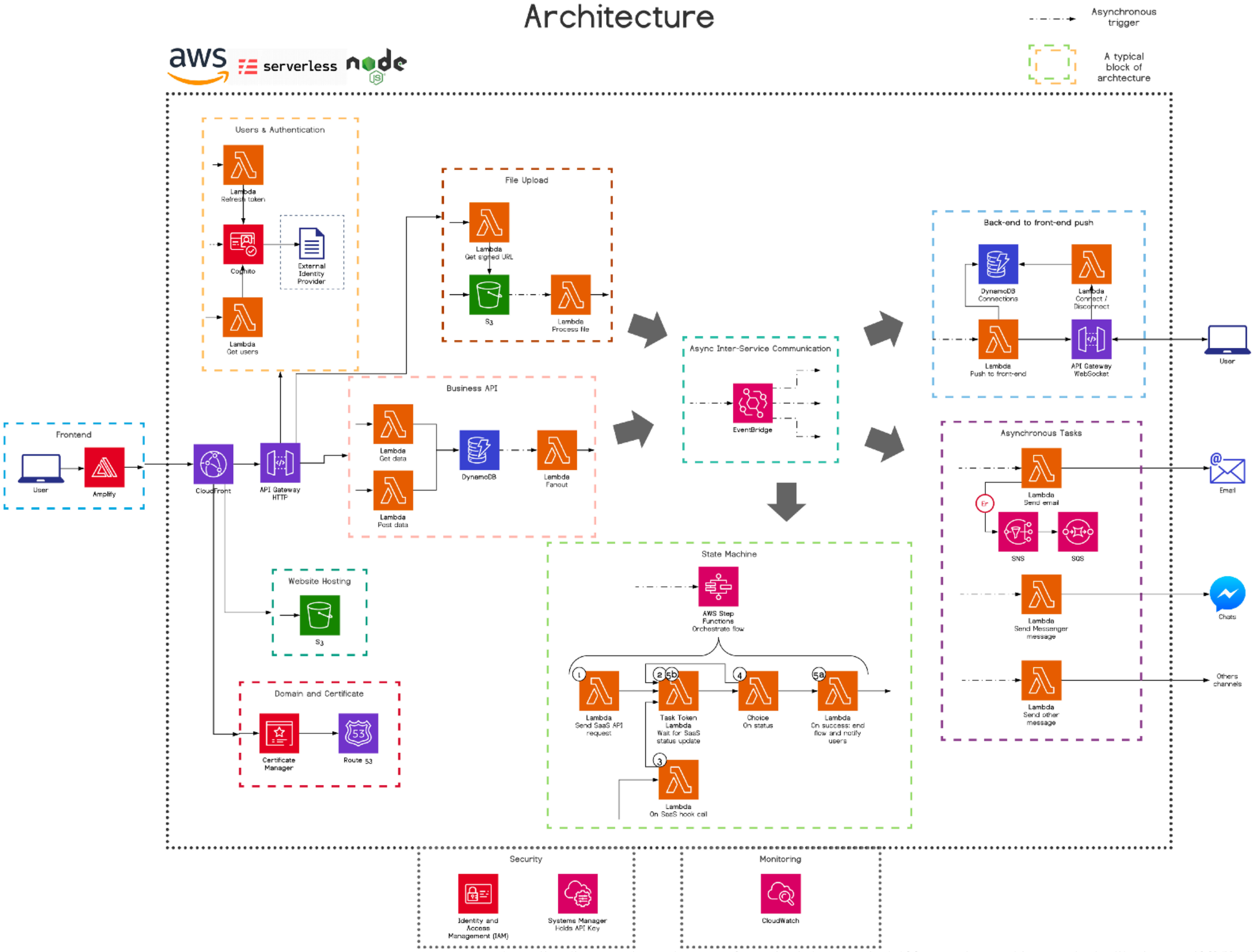
Serverless Lens

AWS Serverless Architecture

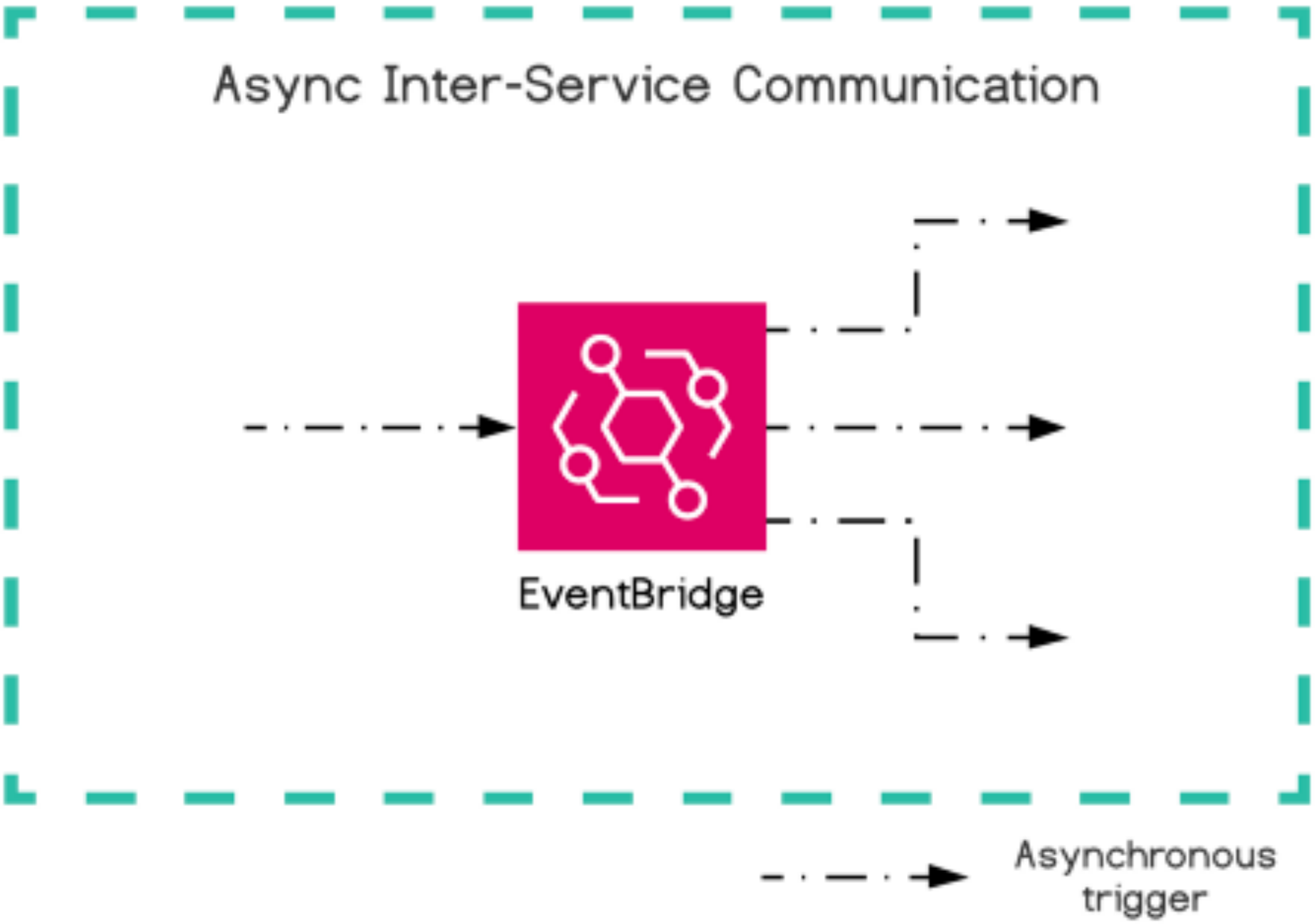
Serverless Web Application

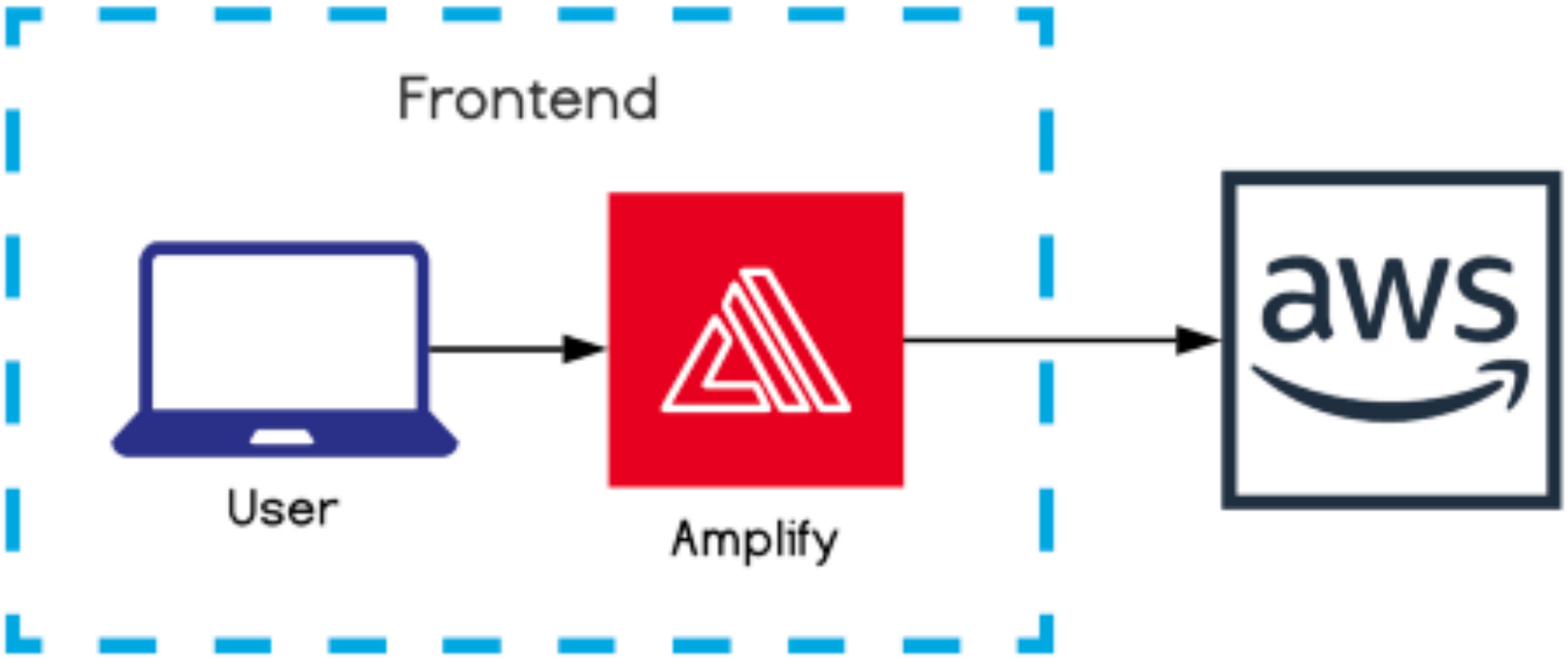


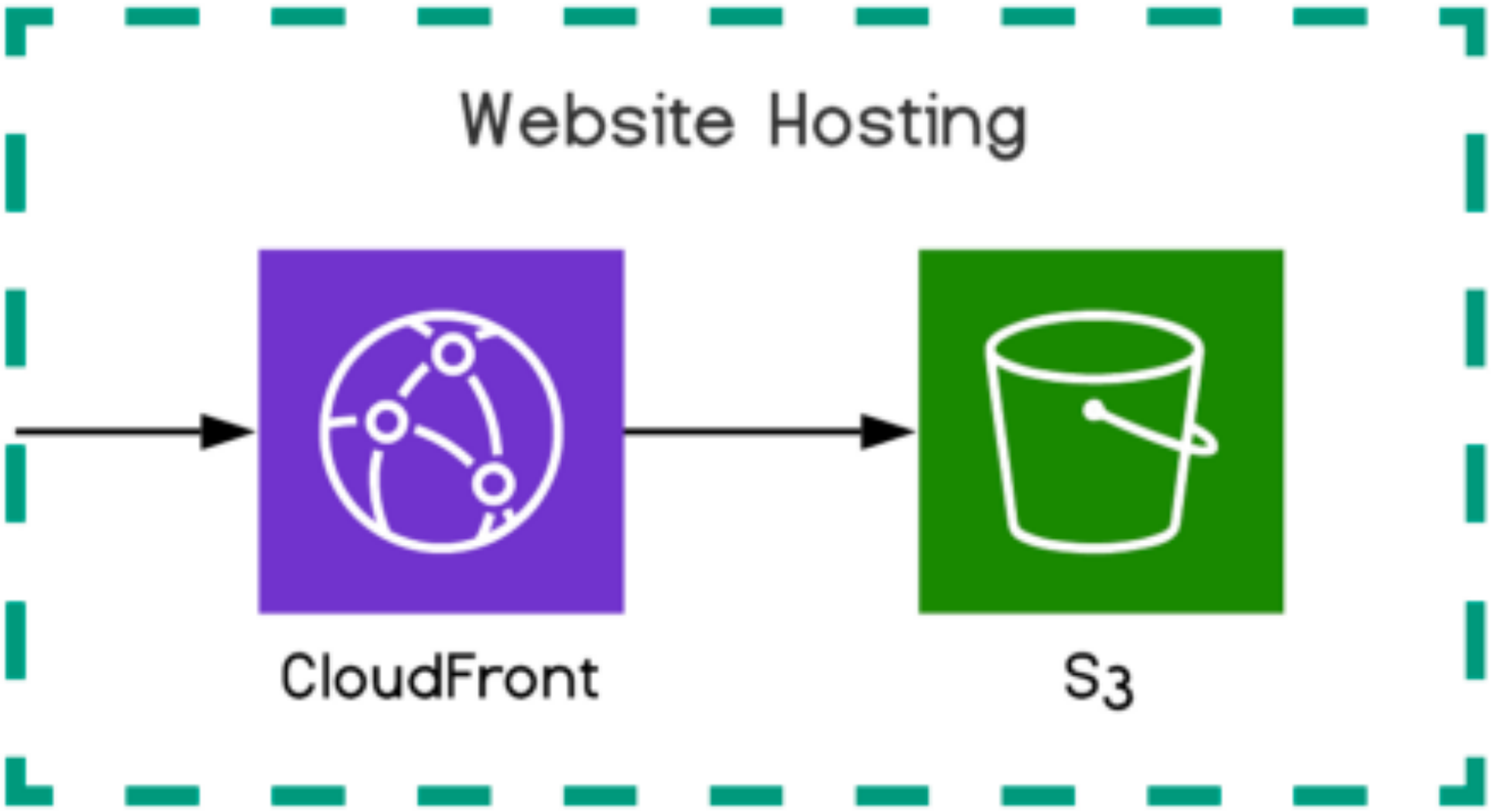
Typical Serverless Architecture

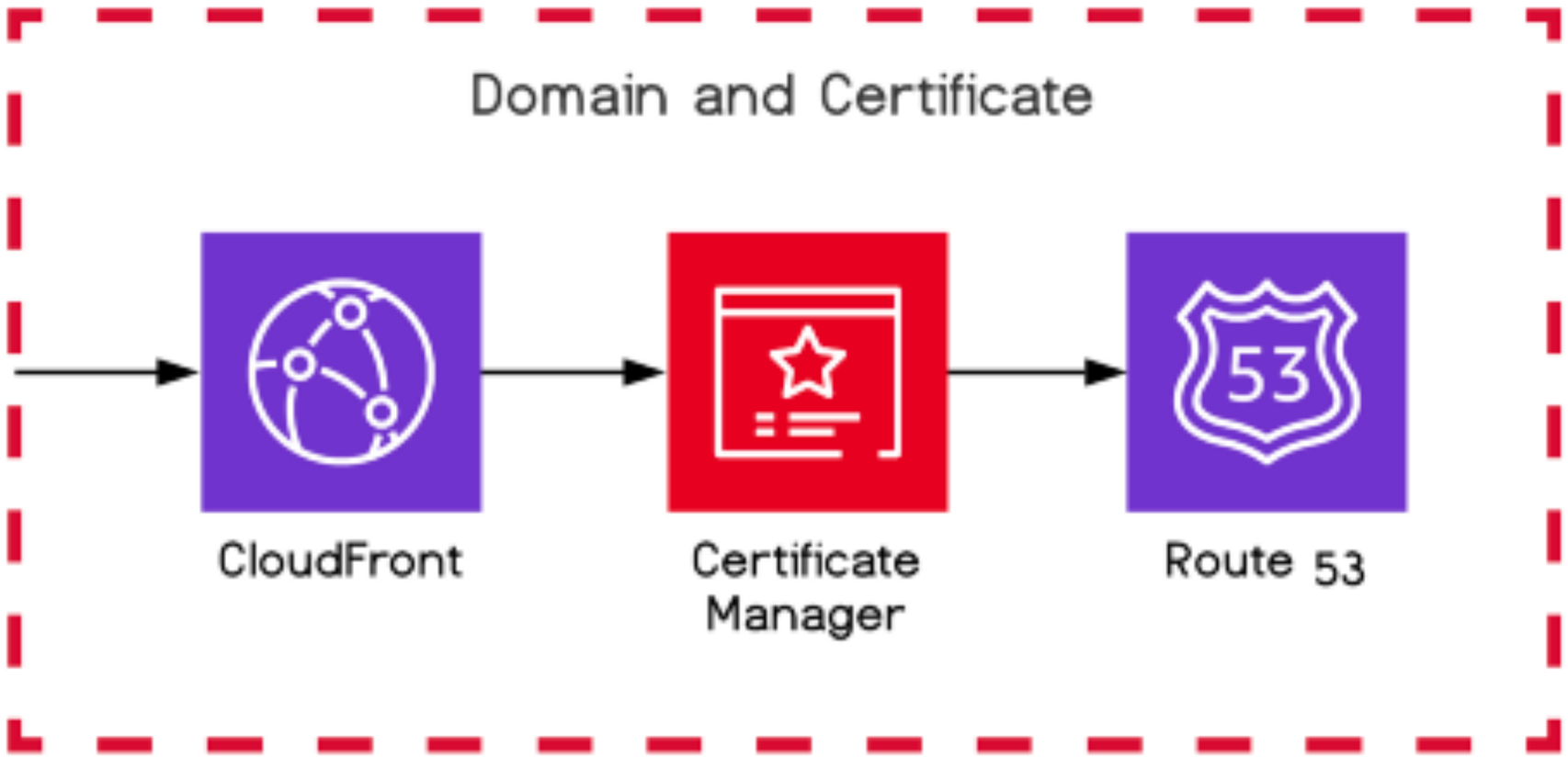


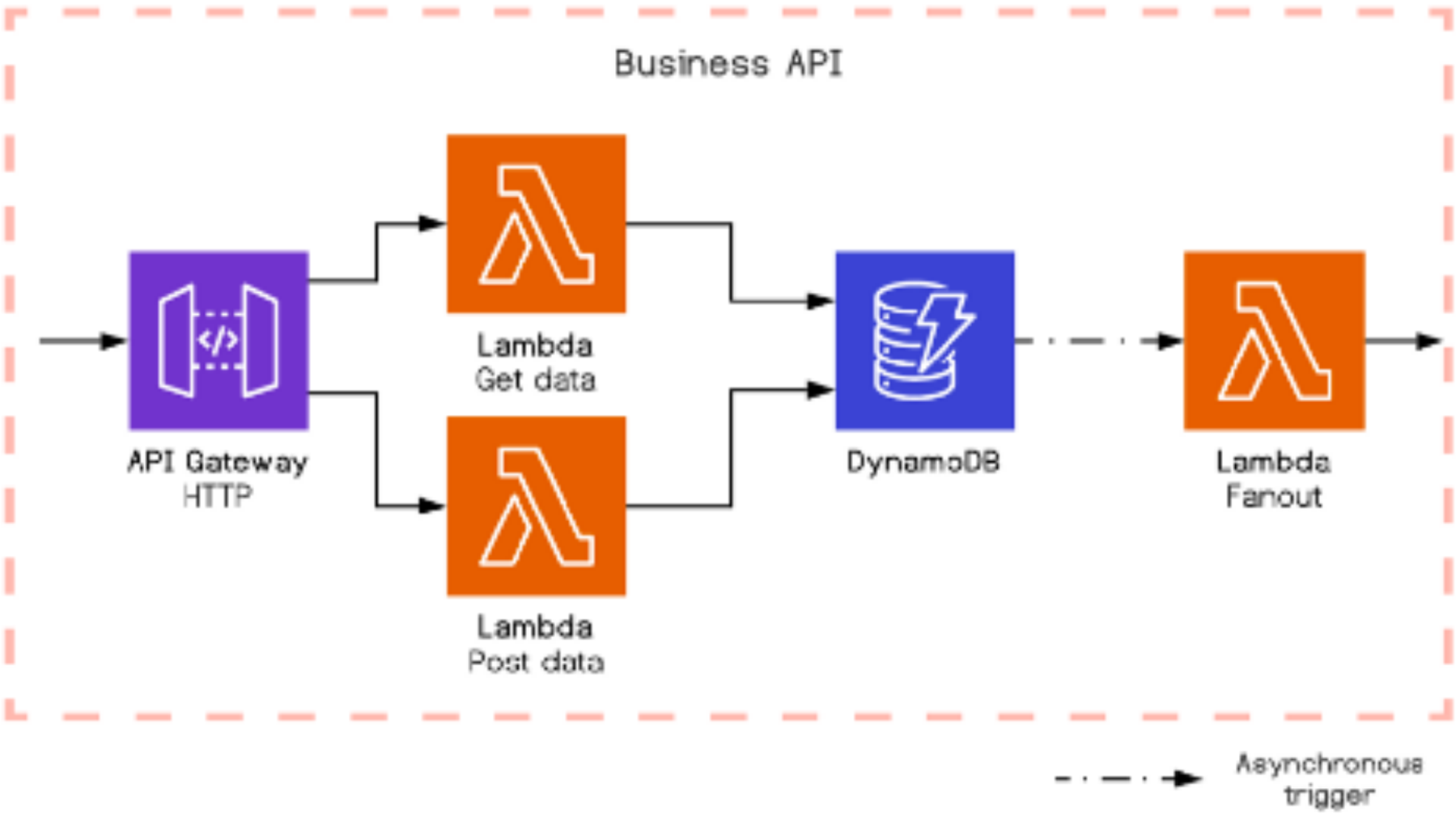
https://medium.com/serverless-transformation/what-a-typical-100-serverless-architecture-looks-like-in-aws-40f252cd0ecb

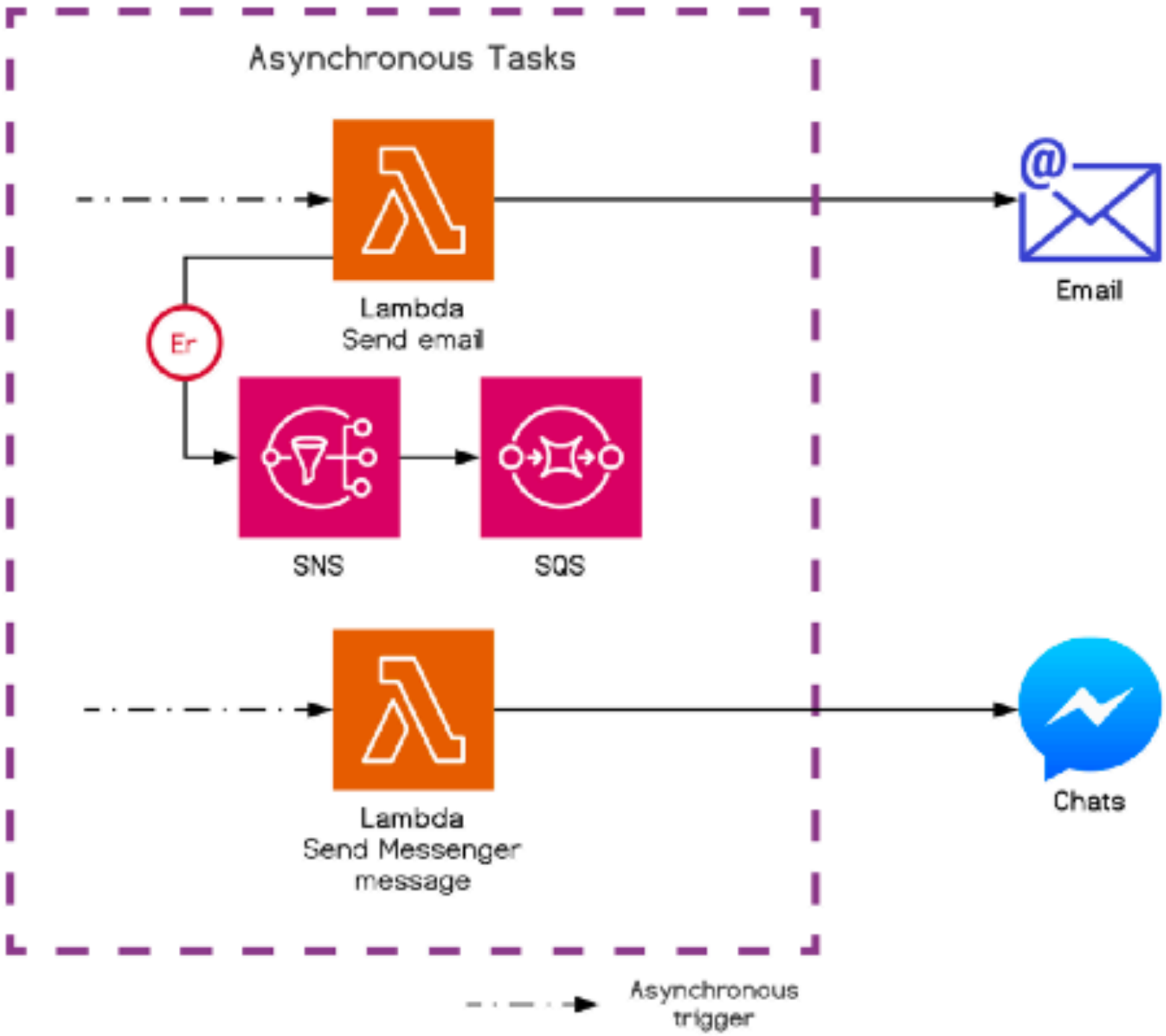

Async Inter-Service Communication
EventBridge
Asynchronous
trigger

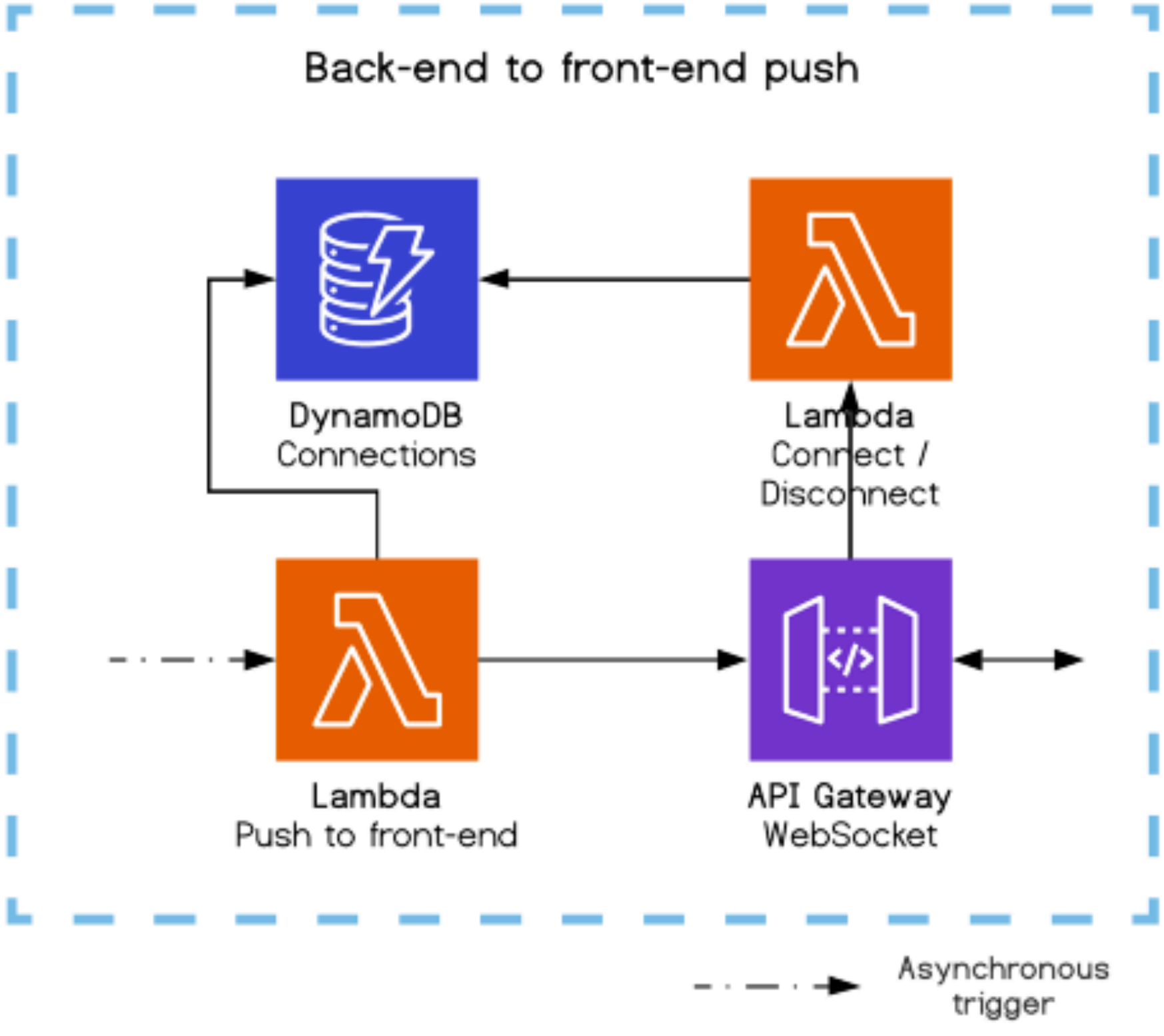

Frontend
User
Amplify
aws

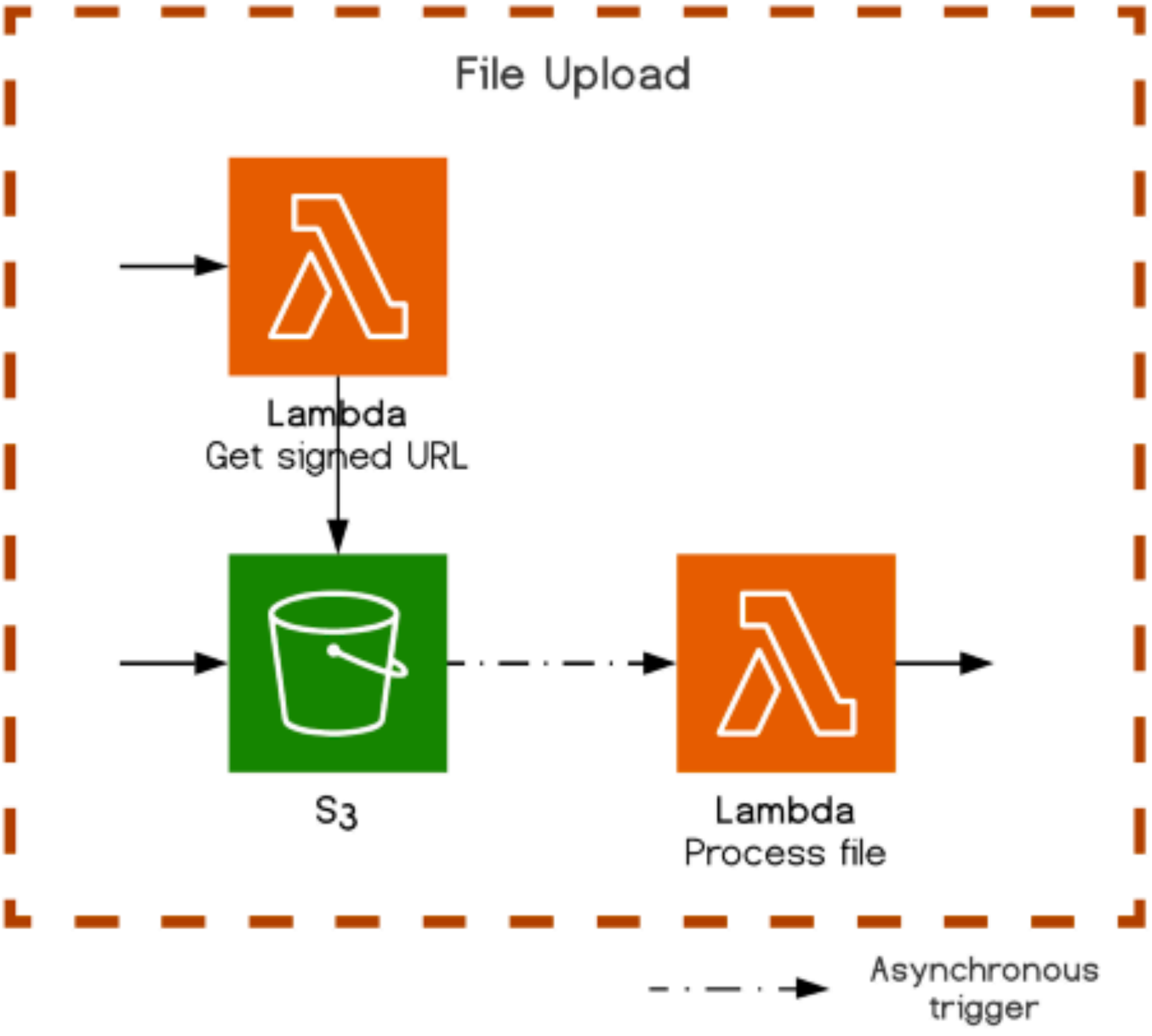

Website Hosting
CloudFront
S3

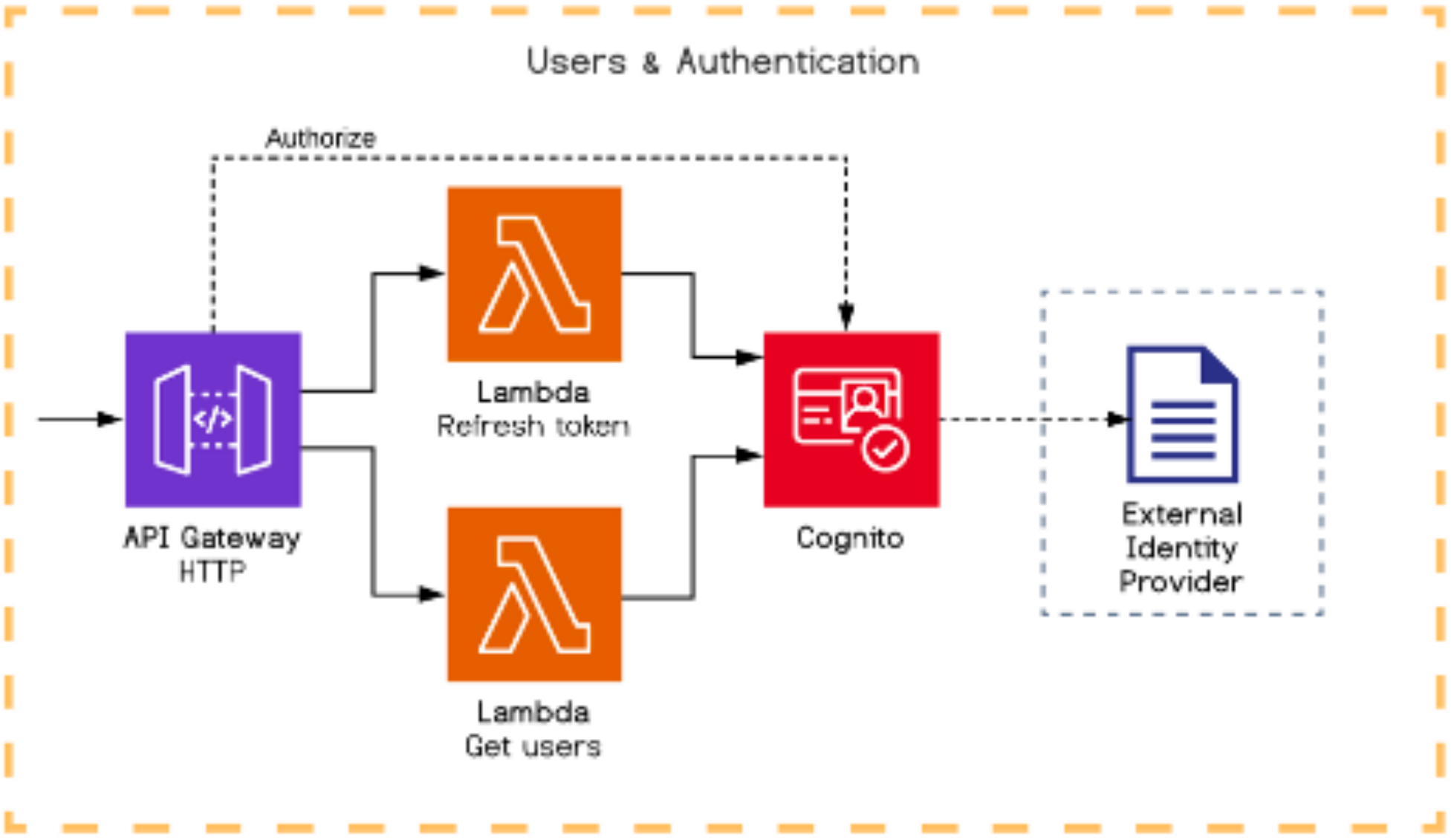

Domain and Certificate
CloudFront
Certificate Manager
Route 53

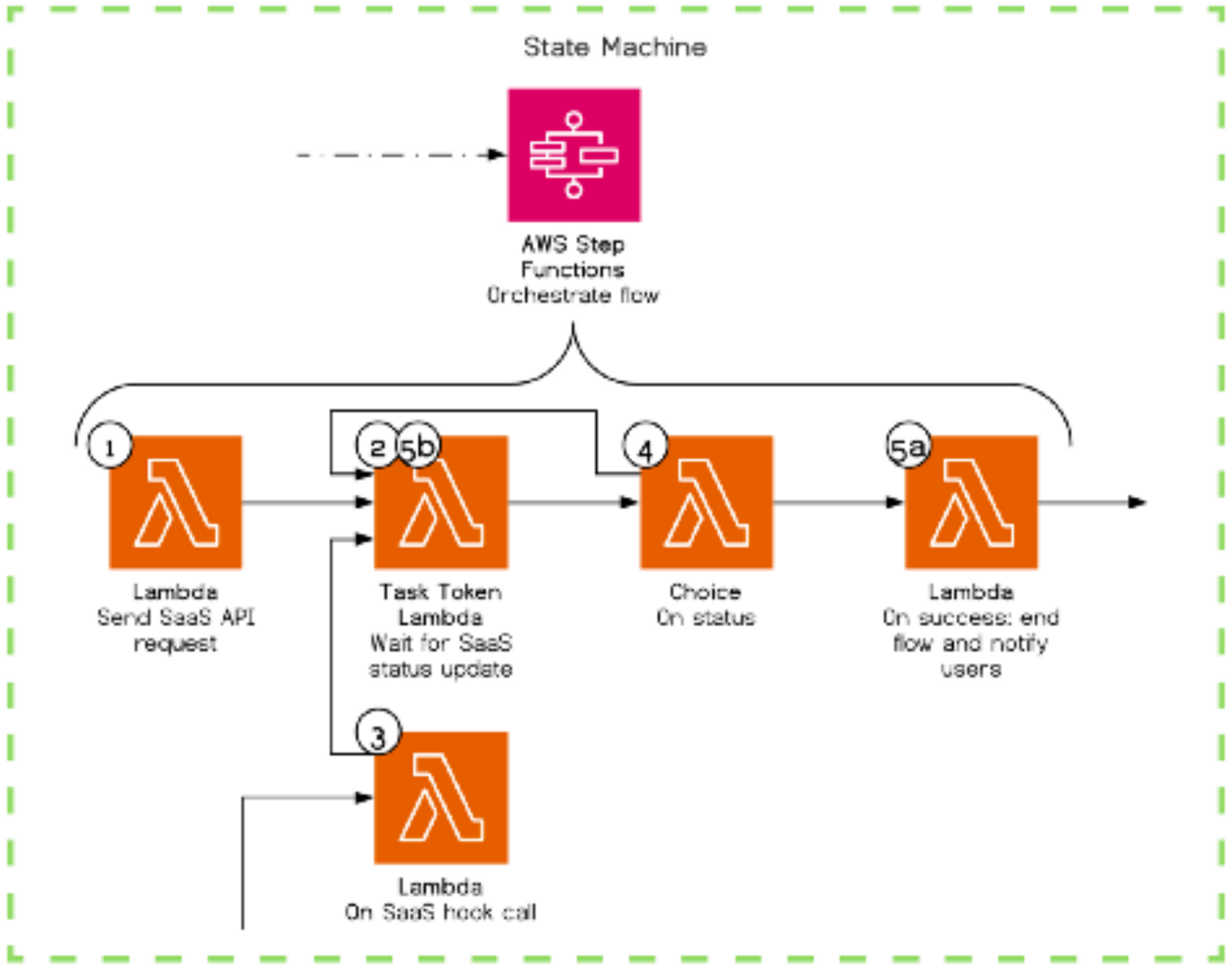

Business API
API Gateway
HTTP
Lambda
Get data
Lambda
Post data
DynamoDB
Lambda
Fanout
Asynchronous
trigger

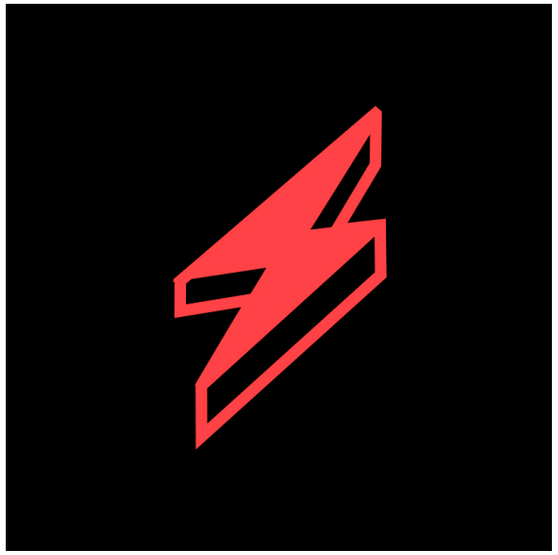

Asynchronous Tasks
Lambda
Send email
Email
Er
SNS
SQS
Lambda
Send Messenger
message
Chats
Asynchronous
trigger


Back-end to front-end push
DynamoDB
Connections
Lambda
Connect /
Disconnect
Lambda
Push to front-end
API Gateway
WebSocket
Asynchronous
trigger

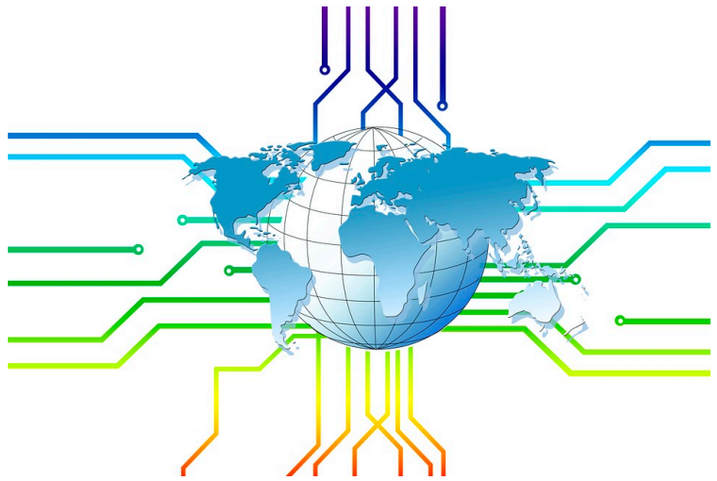

File Upload
Lambda
Get signed URL
S3
Lambda
Process file
Asynchronous
trigger


Users & Authentication
Authorize
Lambda
Refresh token
API Gateway
HTTP
Cognito
External
Identity
Provider
Lambda
Get users

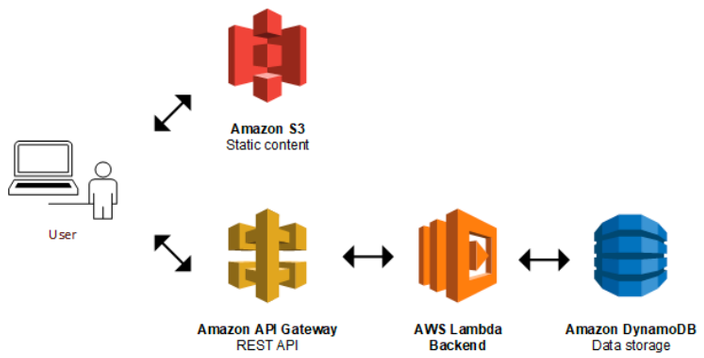

State Machine
AWS Step Functions Orchestrate flow
1
Lambda Send SaaS API request
2 5b
Task Token Lambda Wait for SaaS status update
3
Lambda On SaaS hook call
4
Choice On status
5a
Lambda On success: end flow and notify users



Serverless Framework



Serverless Architecture



AWS Applications

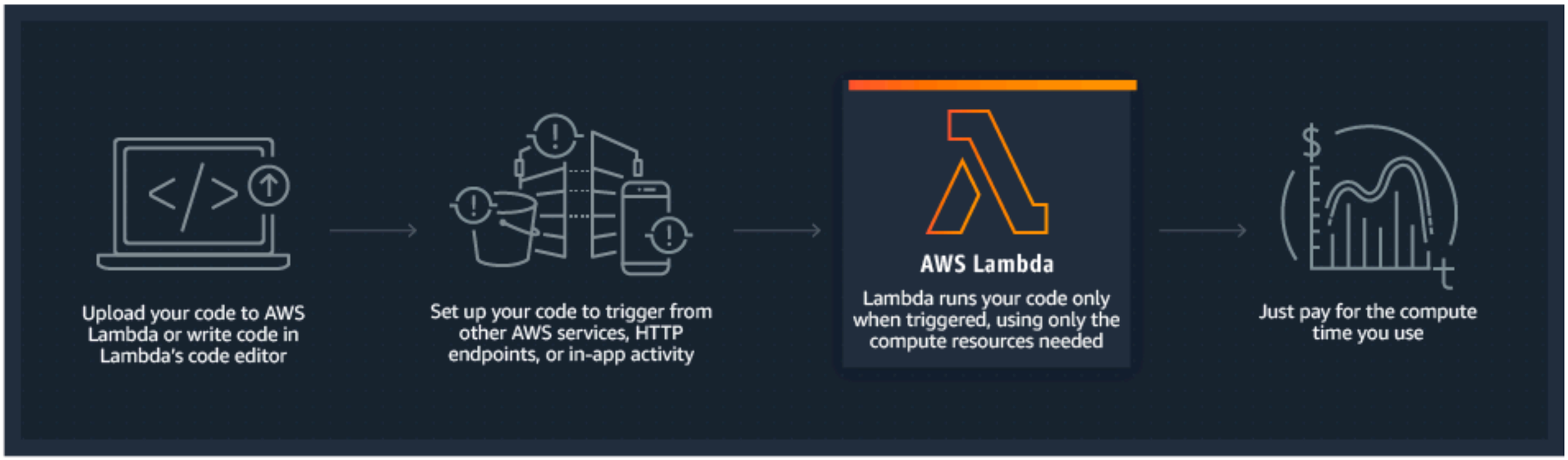
AWS Step Functions

Serverless Applications

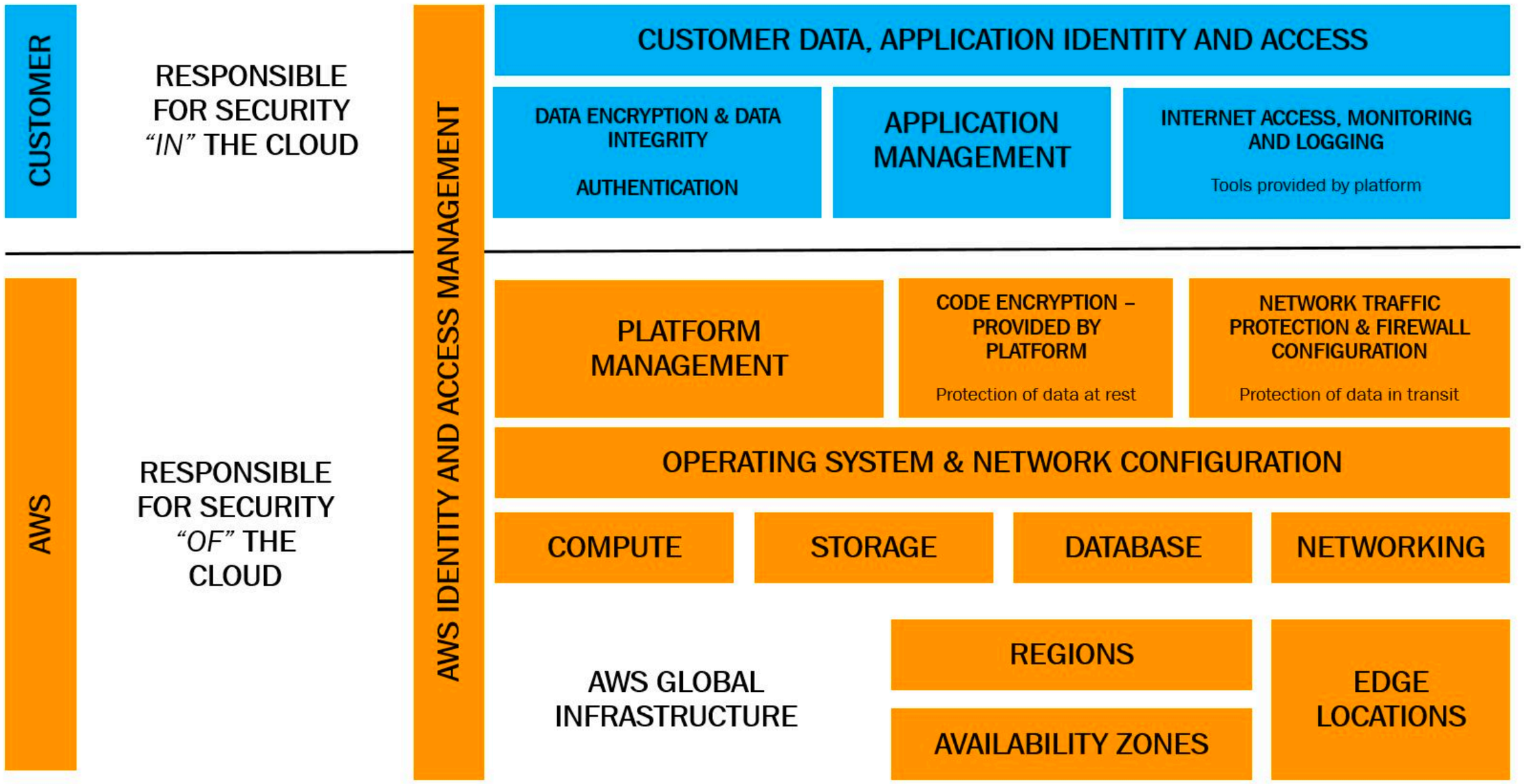
Serverless Lens

AWS Lambda

AWS Lambda

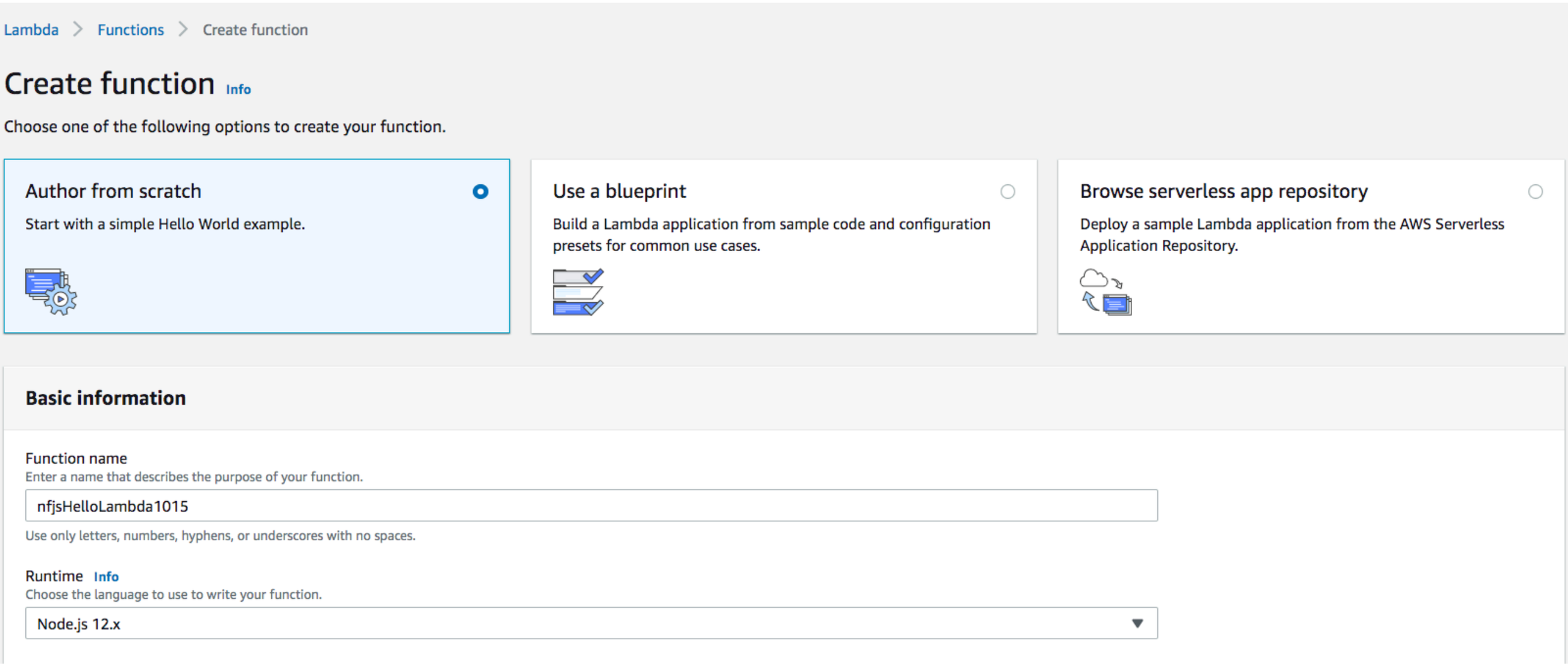

Upload your code to AWS Lambda or write code in Lambda's code editor
Set up your code to trigger from other AWS services, HTTP endpoints, or in-app activity
AWS Lambda
Lambda runs your code only when triggered, using only the compute resources needed
Just pay for the compute time you use

SHARED RESPONSIBILITY MODEL - LAMBDA



https://docs.aws.amazon.com/whitepapers/latest/security-overview-aws-lambda/the-shared-responsibility-model.html

Hello NFJS with Lambda



https://docs.google.com/document/d/1ad1moO9cFBzug7OIXZR95PO6YczfxqepC-1FMJTV5rQ/edit?usp=sharing

aws Services gabriel

Successfully created the function **nfjsHelloLambda1015**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

Lambda > Functions > nfjsHelloLambda1015

ARN - arn:aws:lambda:us-east-1:066177708087:functio

nfjsHelloLambda1015

Throttle | Qualifiers | Actions | test1

Execution result: succeeded (logs)

▶ Details

Configuration | Permissions | Monitoring

Designer

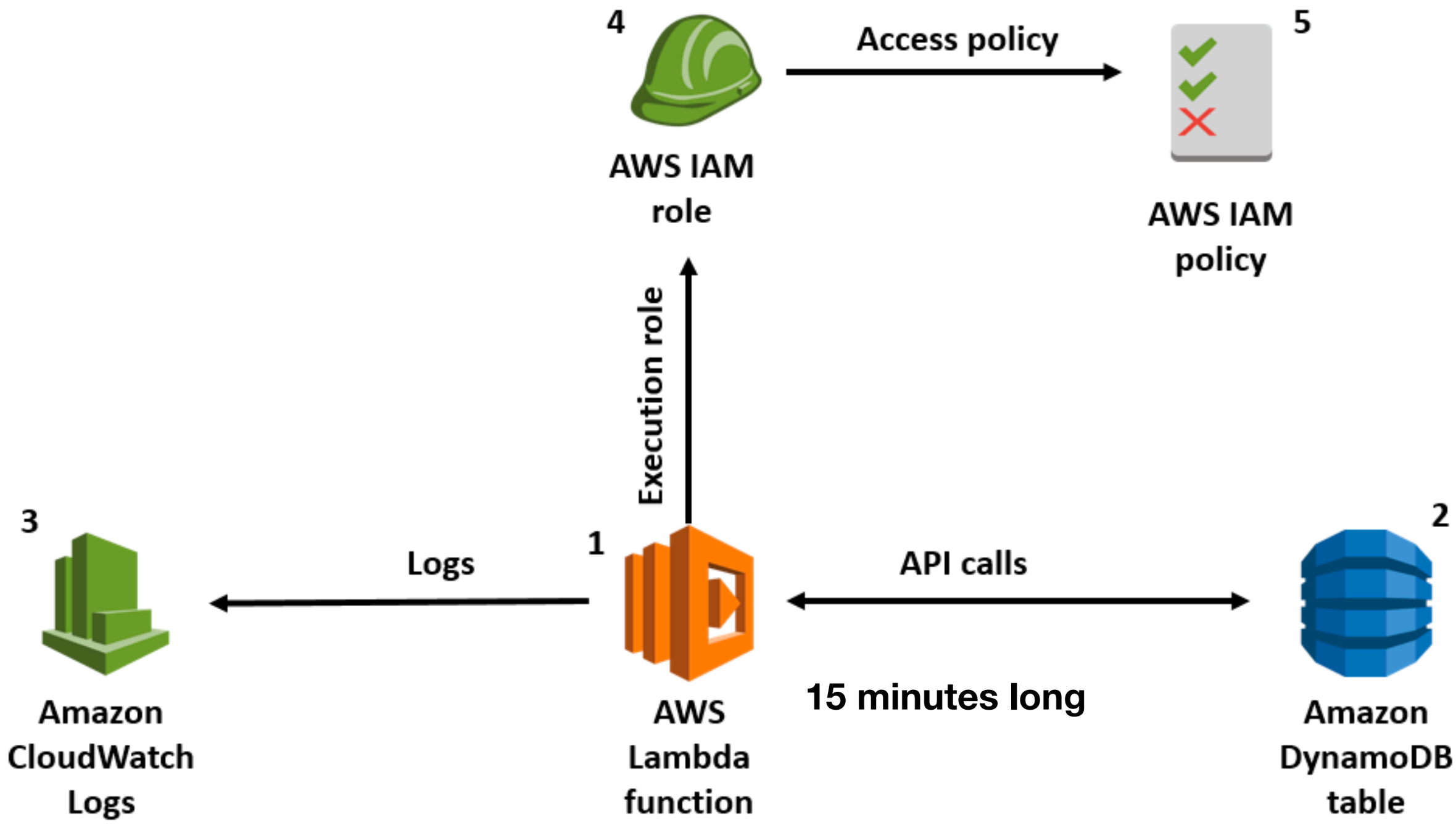
nfjsHelloLambda1015

Layers (0)

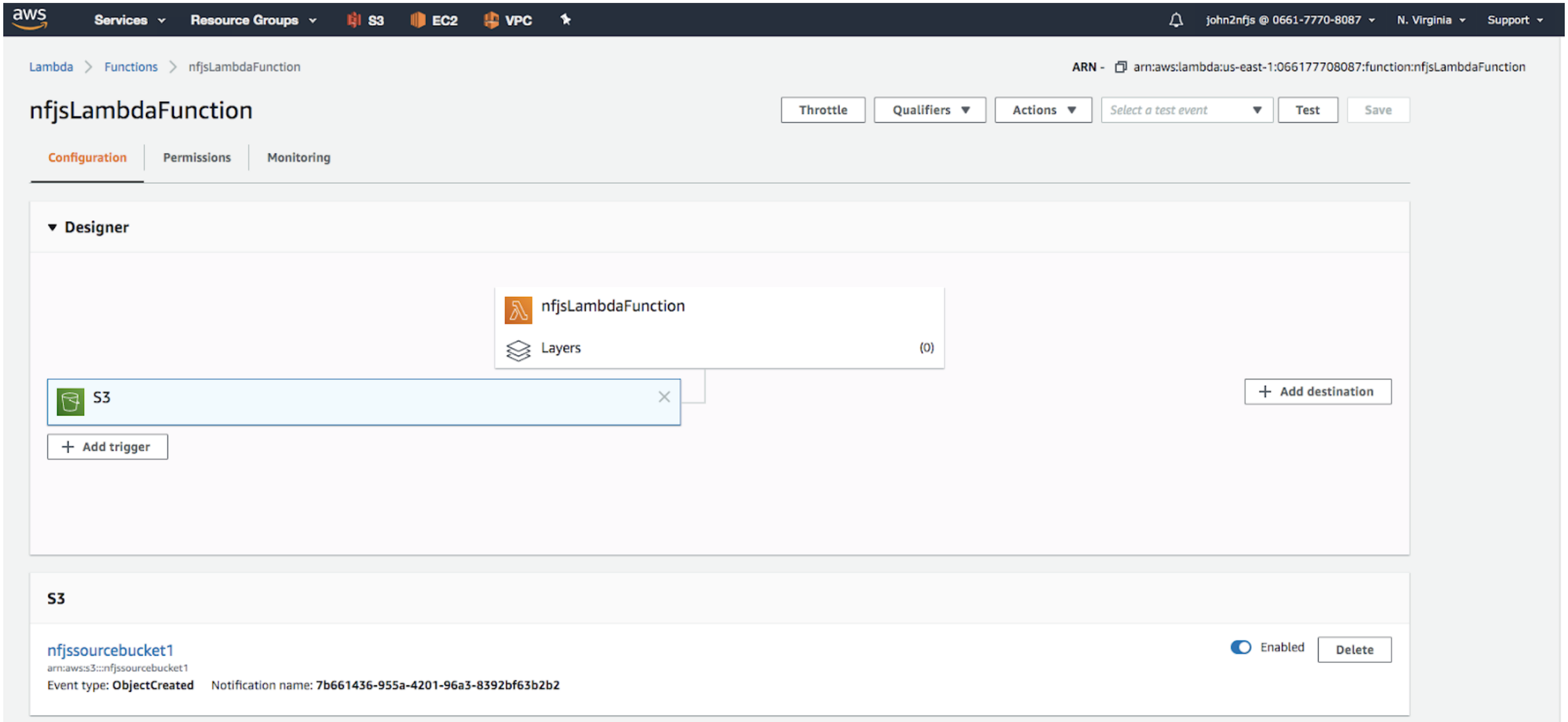
+ Add trigger

+

Lambda Permissions Model



Amazon Lambda with input from S3



https://drive.google.com/file/d/1SIkckGWKOa7JN45PwdFicE9Hw_UcGp3Z/view?usp=sharing

Photo processing

https://github.com/aws-samples/lambda-refarch-fileprocessing

Name of the Lambda

Event Object

Context Object

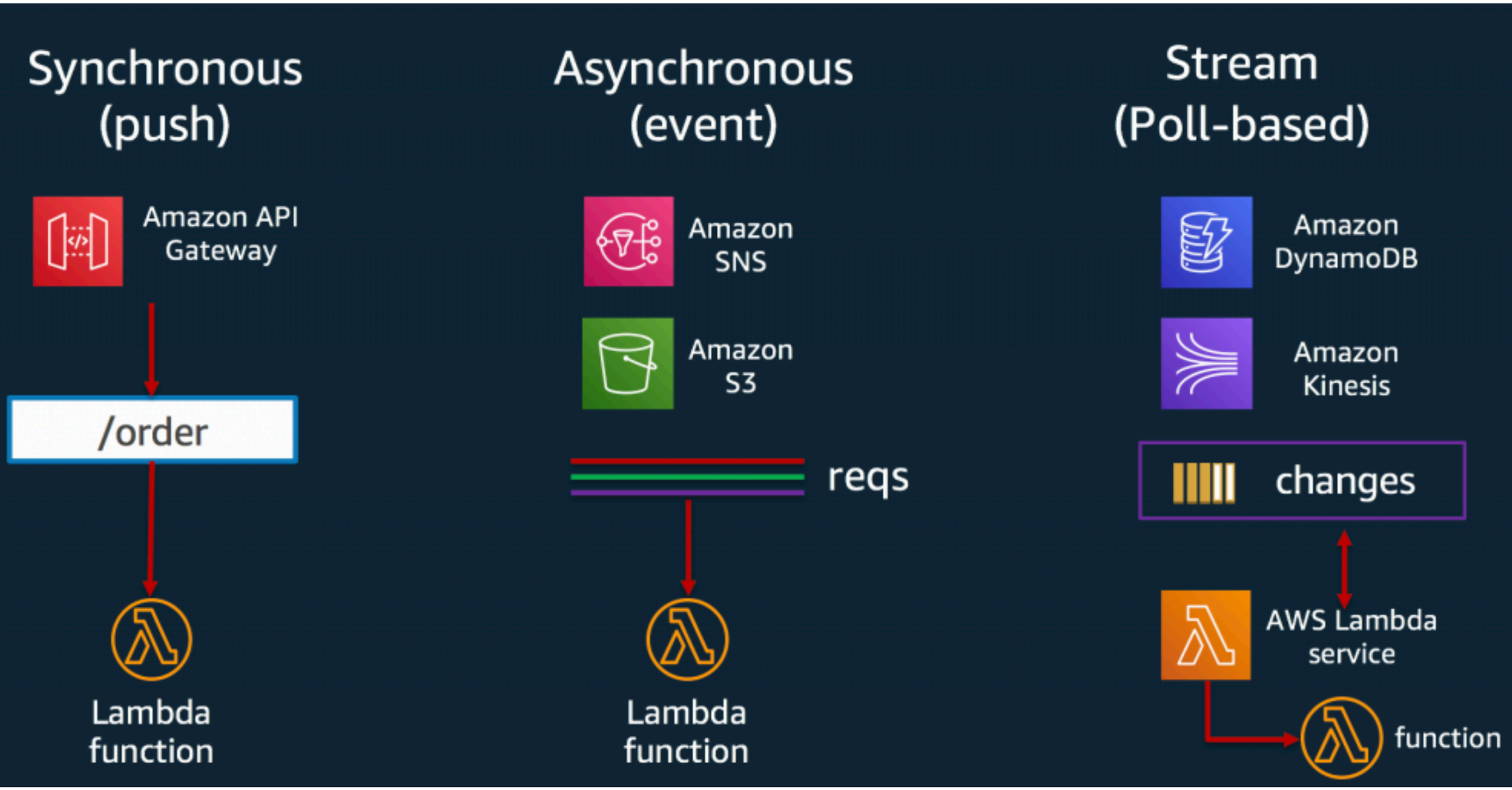
Callback Function

```
exports.myHandler = function(event, context, callback) {
  // Do stuff

  callback(Error error, Object result);

}
```

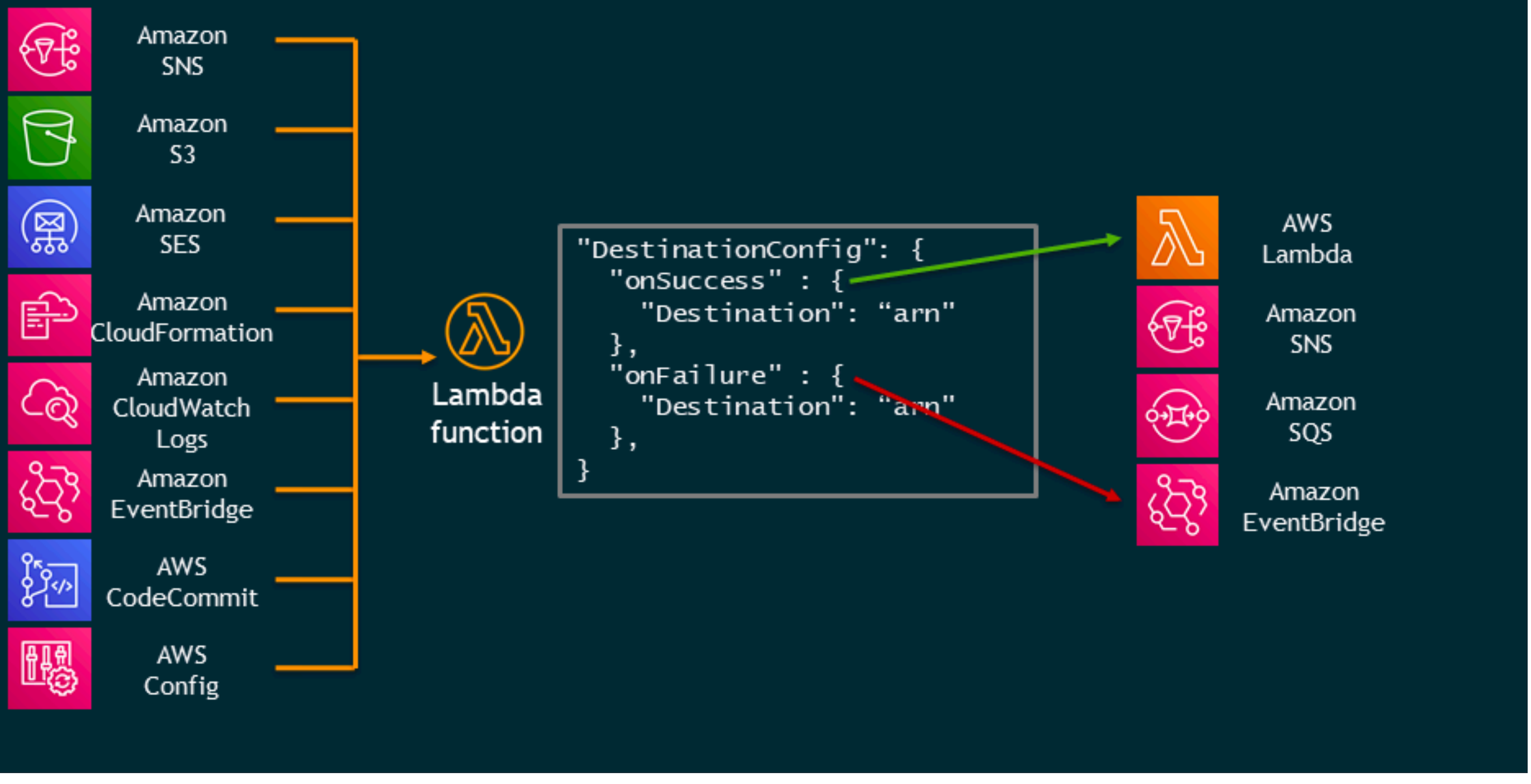
Error Object

Result Object

Event object



Asynchronous Function Execution Result



https://aws.amazon.com/blogs/compute/introducing-aws-lambda-destinations/

EC2 Instance

Container

JVM

LambdaHandler (Java instance)

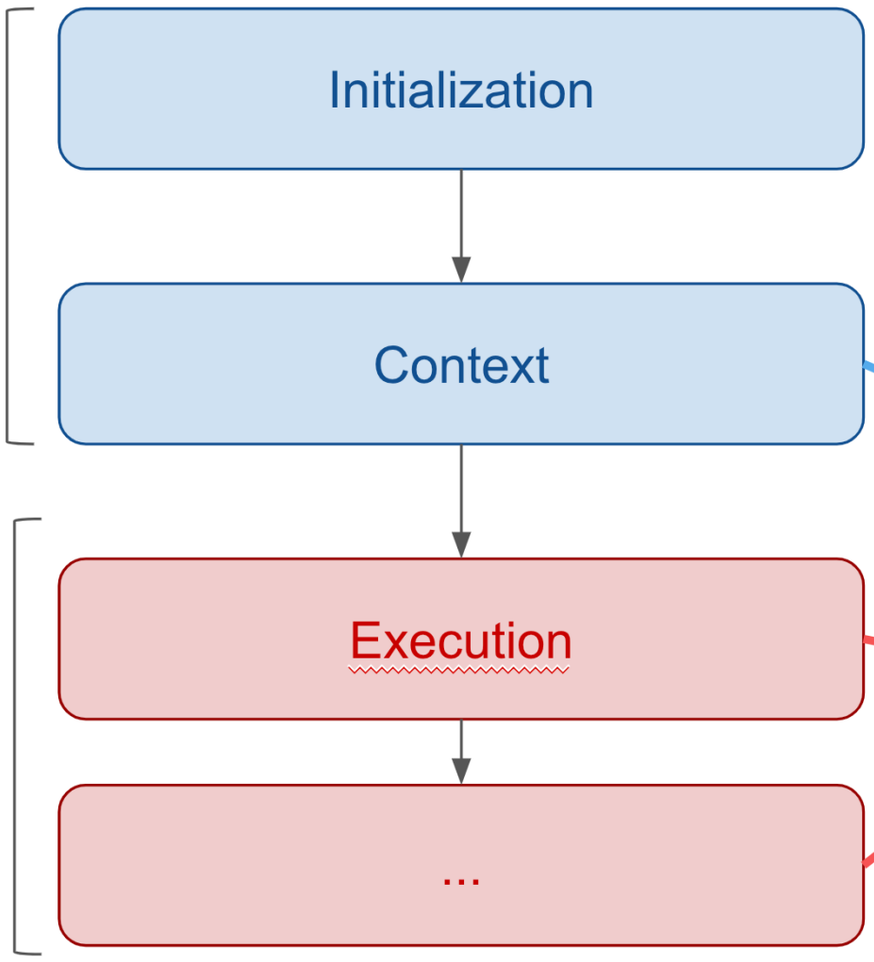
```java
public class LambdaHandler implements RequestHandler<String, String> {

  public LambdaHandler() { }

  public String handleRequest(final String input, final Context context) {
    return input.toUpperCase();
  }

}
```

function invoked

https://blog.ippon.tech/lambda-execution-context-demystified/

one Cold Start

multiple invocations




```java
public class LambdaHandler implements RequestHandler<String, String> {

    DatabaseConnection databaseConnection;

    public LambdaHandler() {
        databaseConnection = new DatabaseConnection();
    }


    @Override
    public String handleRequest(String input, Context context) {
        return input.toUpperCase();
    }
}
```

Context Object

- When Lambda runs your function, it passes a context object to the handler. This object provides methods and properties that provide information about the invocation, function, and execution environment.

Greets Lambda function

```
var greets = [
   "Hello NFJS!",
   "Hello Awesome Serverless!",
   "Today is a great day!",
   "Go Lambda function!",
   "I am expecting miracles",
   "Go get it!",
   "Twinkle Twinkle Little star!",
   "How I wonder what you are",
   "Love the serverless!",
   "Cool apps using serverless!"
];

exports.handler = (event, context, callback) => {
   let greet = greets[Math.floor(Math.random()*10)];
      // Context object
      console.log('Remaining time: ', context.getRemainingTimeInMillis())
      console.log('Function name: ', context.functionName)
      console.log('Memory Limit In MB : ', context.memoryLimitInMB)

   callback(null, greet);
};
```

Error Handling

```
exports.handler = (event, context, callback) => {
   let greet = greets[Math.floor(Math.random()*10)];
   // Context object
   console.log('Remaining time: ', context.getRemainingTimeInMillis())
   console.log('Function name: ', context.functionName)
   console.log('Memory Limit In MB : ', context.memoryLimitInMB)
   console.error('Error in the function ')
   console.info('Memory Limit In MB : ', context.memoryLimitInMB)
   console.warn('Warning in Lambda ')
```

•

myNFJSServerless

Throttle | Qualifiers ▼ | Actions ▼ | mytest ▼ | Test | Save

File Edit Find View Go Tools Window Save Test

Environment

myNFJSServerless

index.js

index.js

```
1  var greets = [
2     "Hello NFJS!",
3     "Hello Awesome Serverless!",
4     "Today is a great day!",
5     "Go Lambda function!",
6     "I am expecting miracles",
7     "Go get it!",
8     "Twinkle Twinkle Little star!",
9     "How I wonder what you are",
10    "Love the serverless!",
11    "Cool apps using serverless!"
12 ];
13
14 exports.handler = (event, context, callback) => {
15    let greet = greets[Math.floor(Math.random()*10)];
16    // Context object
```

(799 Bytes) 1:1 JavaScript Spaces: 3

Execution Result

Execution results

Status: Succeeded | Max Memory Used: 65 MB | Time: 39.51 ms

```
Response:
"I am expecting miracles"

Request ID:
"8d0164f9-59b7-4f0b-a374-526f36f9832c"

Function logs:
START RequestId: 8d0164f9-59b7-4f0b-a374-526f36f9832c Version: $LATEST
2020-09-02T12:19:46.867Z    8d0164f9-59b7-4f0b-a374-526f36f9832c    INFO    Remaining time:  2999
2020-09-02T12:19:46.884Z    8d0164f9-59b7-4f0b-a374-526f36f9832c    INFO    Function name:  myNFJSServerless
2020-09-02T12:19:46.884Z    8d0164f9-59b7-4f0b-a374-526f36f9832c    INFO    Memory Limit In MB :  128
2020-09-02T12:19:46.884Z    8d0164f9-59b7-4f0b-a374-526f36f9832c    ERROR   Error in the function
2020-09-02T12:19:46.884Z    8d0164f9-59b7-4f0b-a374-526f36f9832c    INFO    Memory Limit In MB :  128
2020-09-02T12:19:46.884Z    8d0164f9-59b7-4f0b-a374-526f36f9832c    WARN    Warning in Lambda
END RequestId: 8d0164f9-59b7-4f0b-a374-526f36f9832c
```

Lambda Limits

Resource	Default quota	Can Be Increased Up To
Concurrent executions	1,000	Hundreds of thousands
Function and layer storage	75 GB	Terabytes
Elastic network interfaces per VPC	250	Hundreds

https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html

Resource	Quota
Function memory allocation	128 MB to 3,008 MB, in 64 MB increments.
Function timeout	900 seconds (15 minutes)
Function environment variables	4 KB
Function resource-based policy	20 KB
Function layers	5 layers
Function burst concurrency	500 - 3000 (varies per region)
Invocation payload (request and response)	6 MB (synchronous) 256 KB (asynchronous)
Deployment package size	50 MB (zipped, for direct upload) 250 MB (unzipped, including layers) 3 MB (console editor)
Test events (console editor)	10
`/tmp` directory storage	512 MB
File descriptors	1,024
Execution processes/threads	1,024

Pricing

- 1M free requests per month and 400,000 GB-seconds of compute time per month

AWS Lambda Pricing

Region: US East (Ohio)

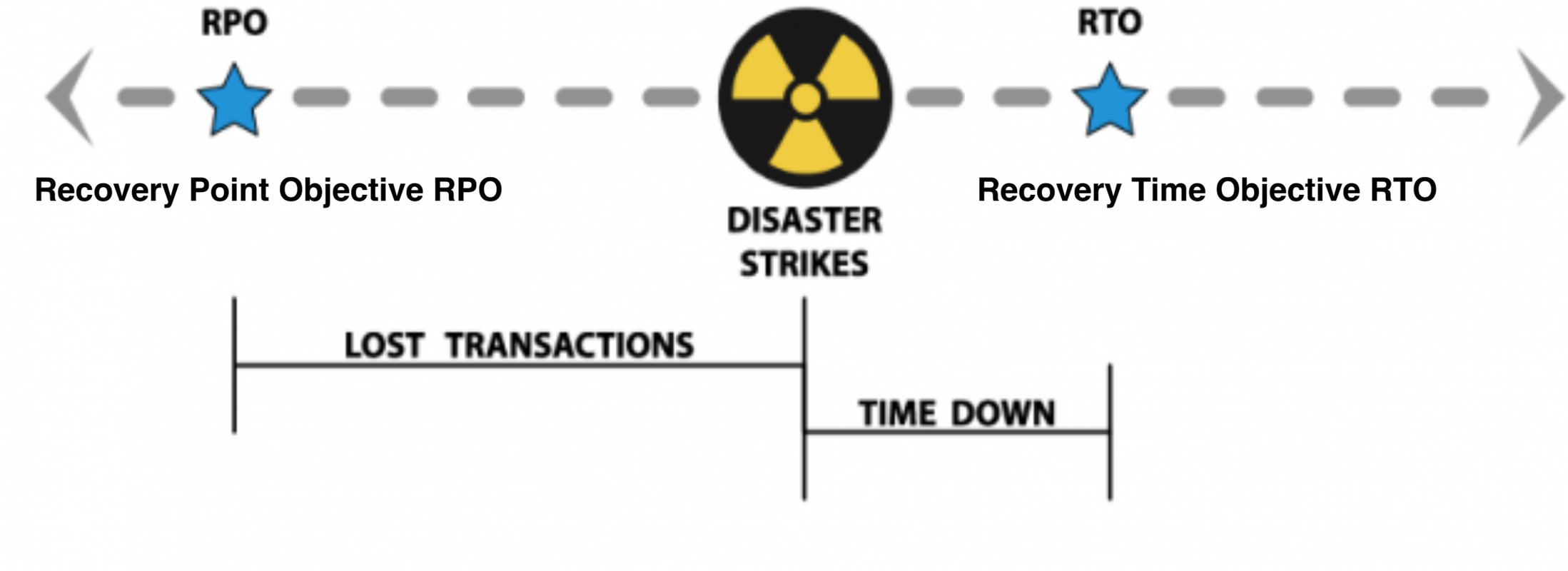
	Price
Requests	$0.20 per 1M requests
Duration	$0.0000166667 for every GB-second

How do you regulate inbound request rates?

- Use throttling to control inbound request rates
- Use, analyze, and enforce API quotas
- Use mechanisms to protect non-scalable resources

How do you build resiliency into your Serverless application?

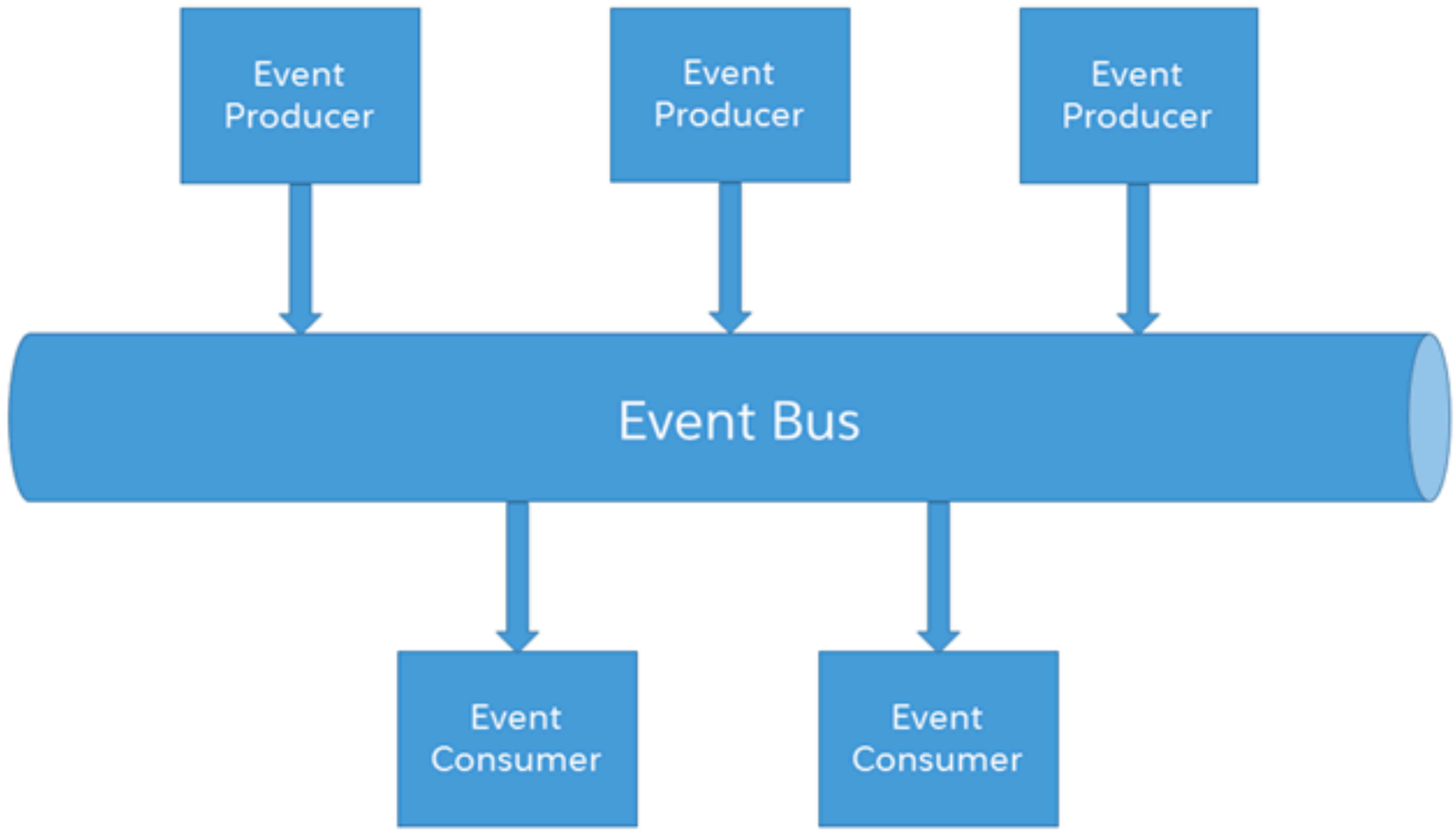
- Use exponential backoff with jitter.
- Use a dead-letter queue mechanism to retain, investigate, and retry failed transactions
- Circuit Breaker



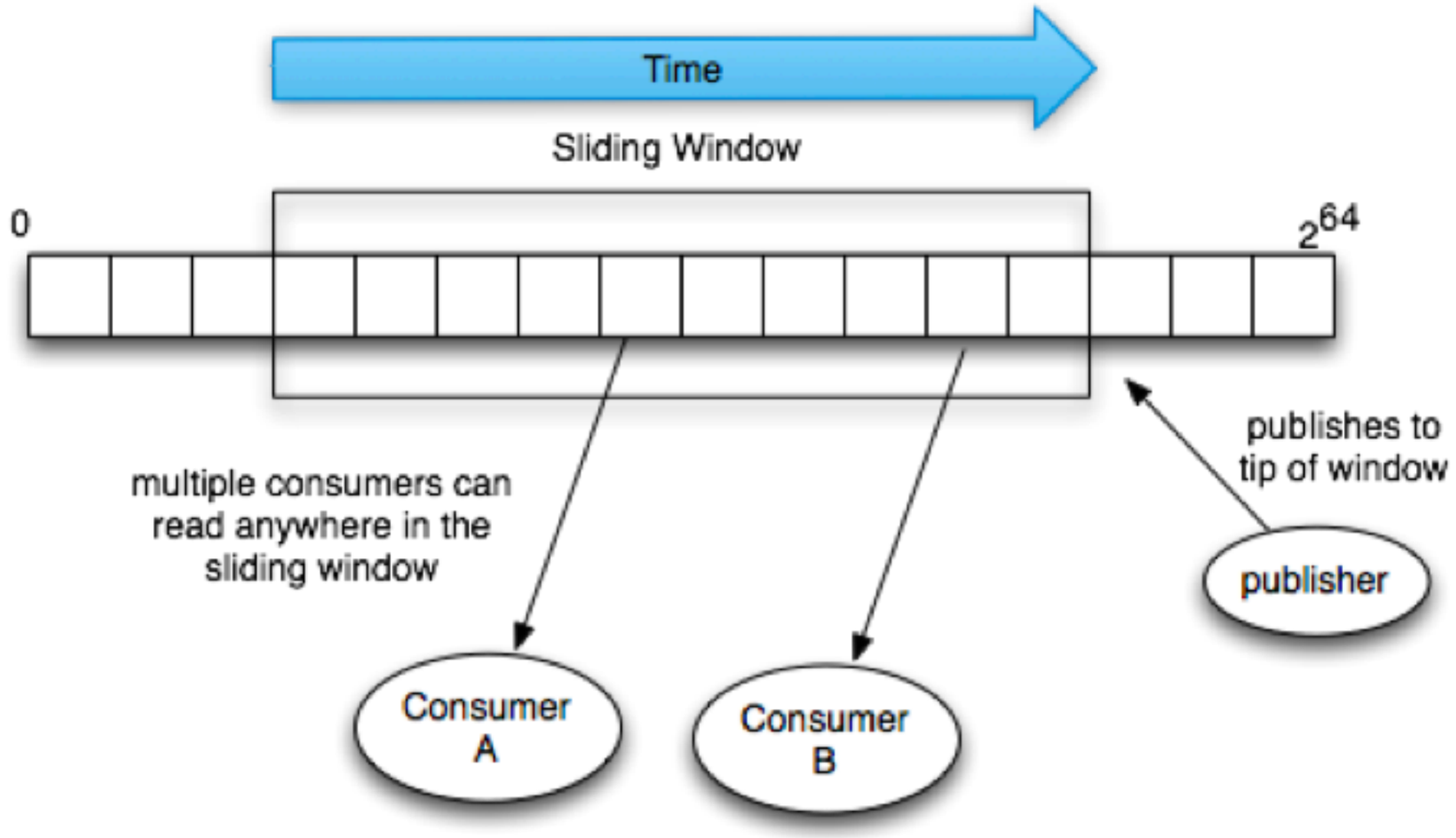
Recovery Point Objective RPO
Recovery Time Objective RTO
WRT: Work recovery time
MTDT: Maximum Tolerable Down Time = RTO + WRT

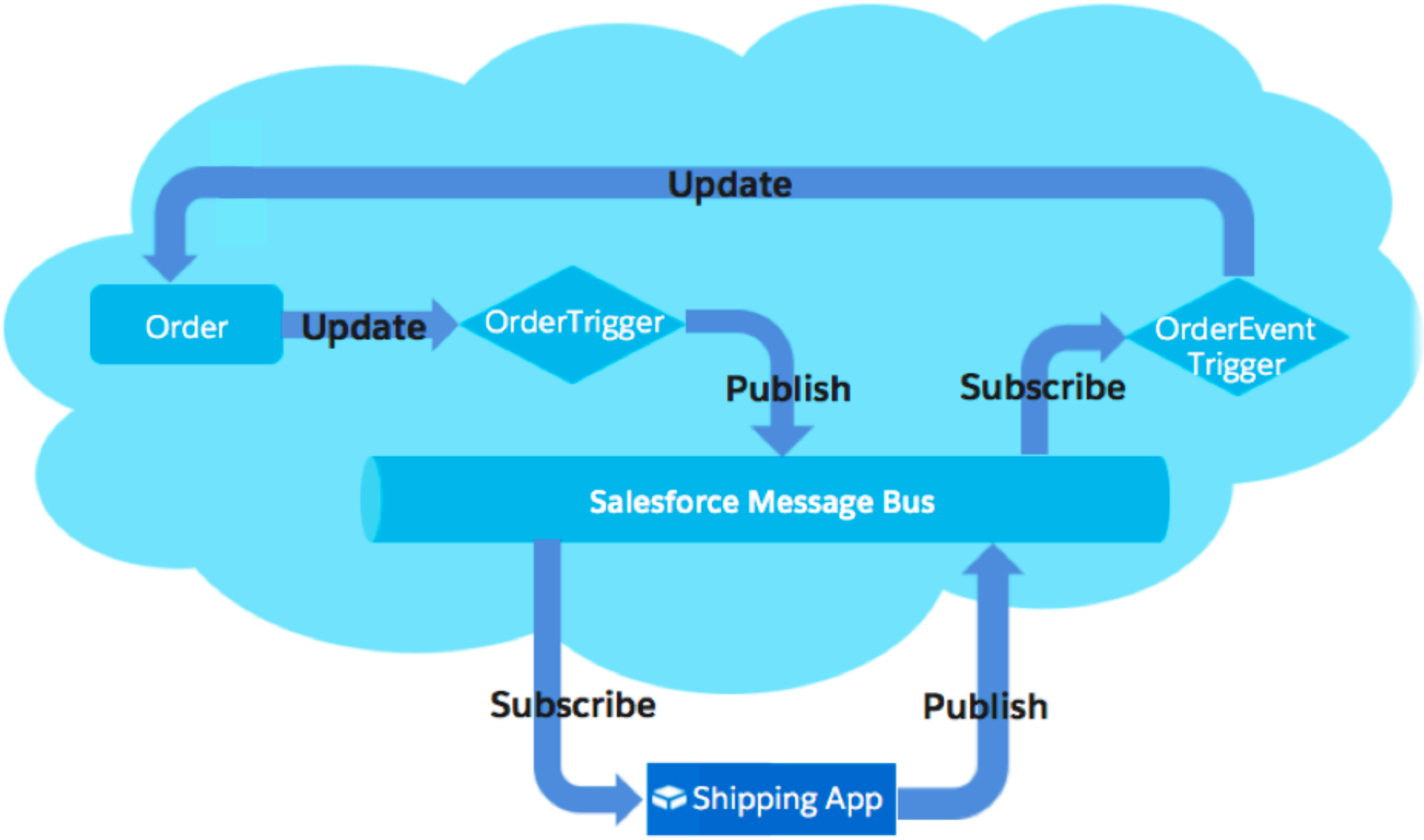
How do you build resiliency into your Serverless application?

- Manage duplicate and unwanted events
- transaction ID, payment ID, booking ID
- Review service account limits with combined utilization across resources.
- Evaluate key metrics to understand how your workload recovers from bursts.



https://developer.salesforce.com/docs/atlas.en-us.api_streaming.meta/api_streaming/using_streaming_api_durability.htm


Time
Sliding Window
0
2^{64}
multiple consumers can read anywhere in the sliding window
publishes to tip of window
publisher
Consumer A
Consumer B



https://developer.salesforce.com/blogs/developer-relations/2017/05/first-impressions-platform-events-salesforce-enterprise-messaging-platform.html

Lambda Best Practices

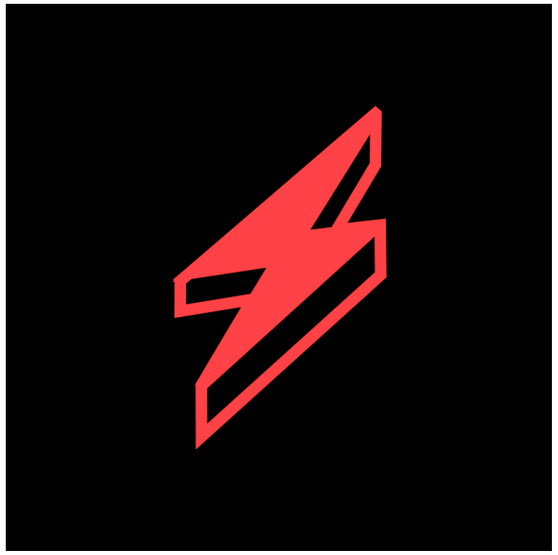
- Programming Best practices
- Keep declarations or installations outside Lambda handler
- Keep the Lambda Handler lean
- Avoid Hardcoding, usr environment variables
- Single responsibility principle- One function, one task

Lambda Best Practices

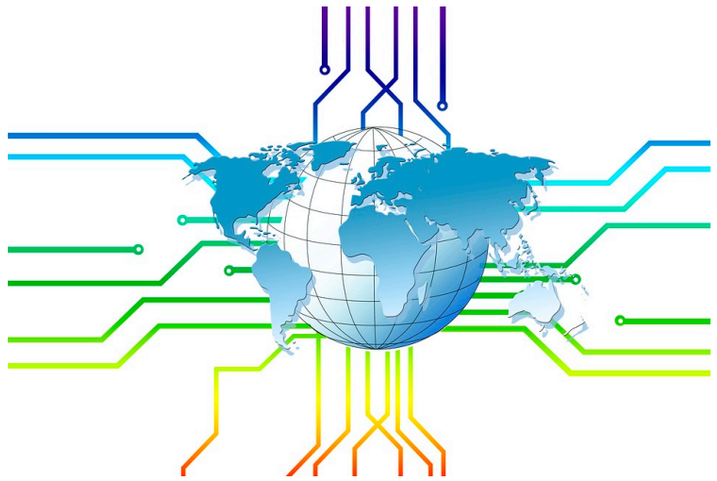
- Deployment package size
- Lambda logs in cloud watch
- Do not give all permissions to lambda, give only as needed
- Delete unused functions
- Error handling- exceptions, DLQs
- Use only if necessary - VPC based resources

Lambda Best Practices

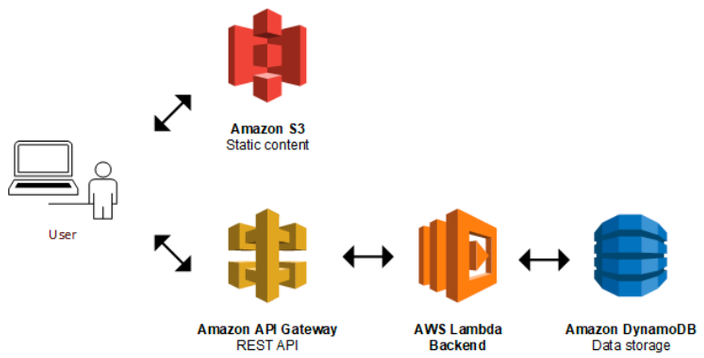
- Reserve concurrency - 1000 concurrency
- Keep container warm - Cold start - dummy invocations cloud watch events
- Use AWS SAM or server less framework
- CICD, DevOps tools



Serverless Framework



Serverless Architecture



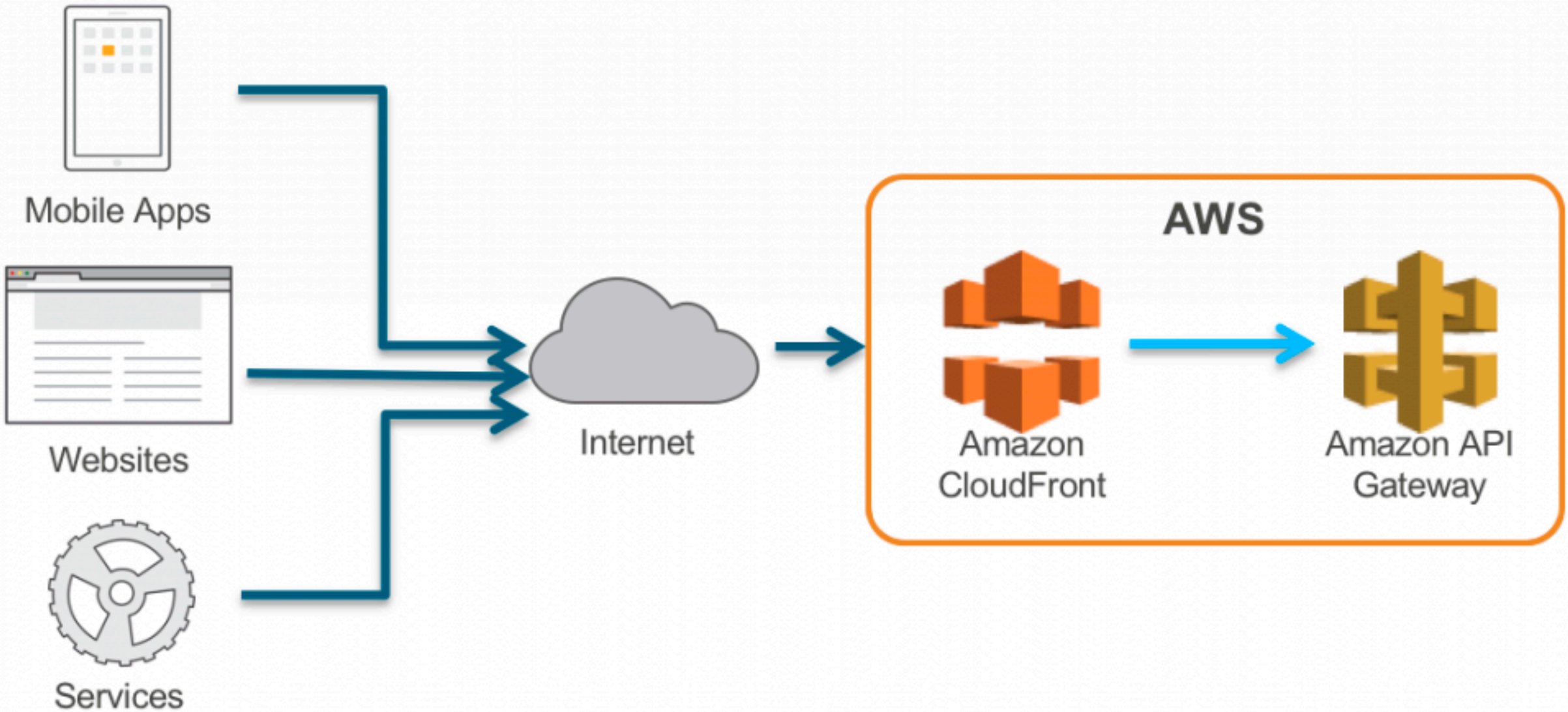
AWS Applications

AWS Step Functions

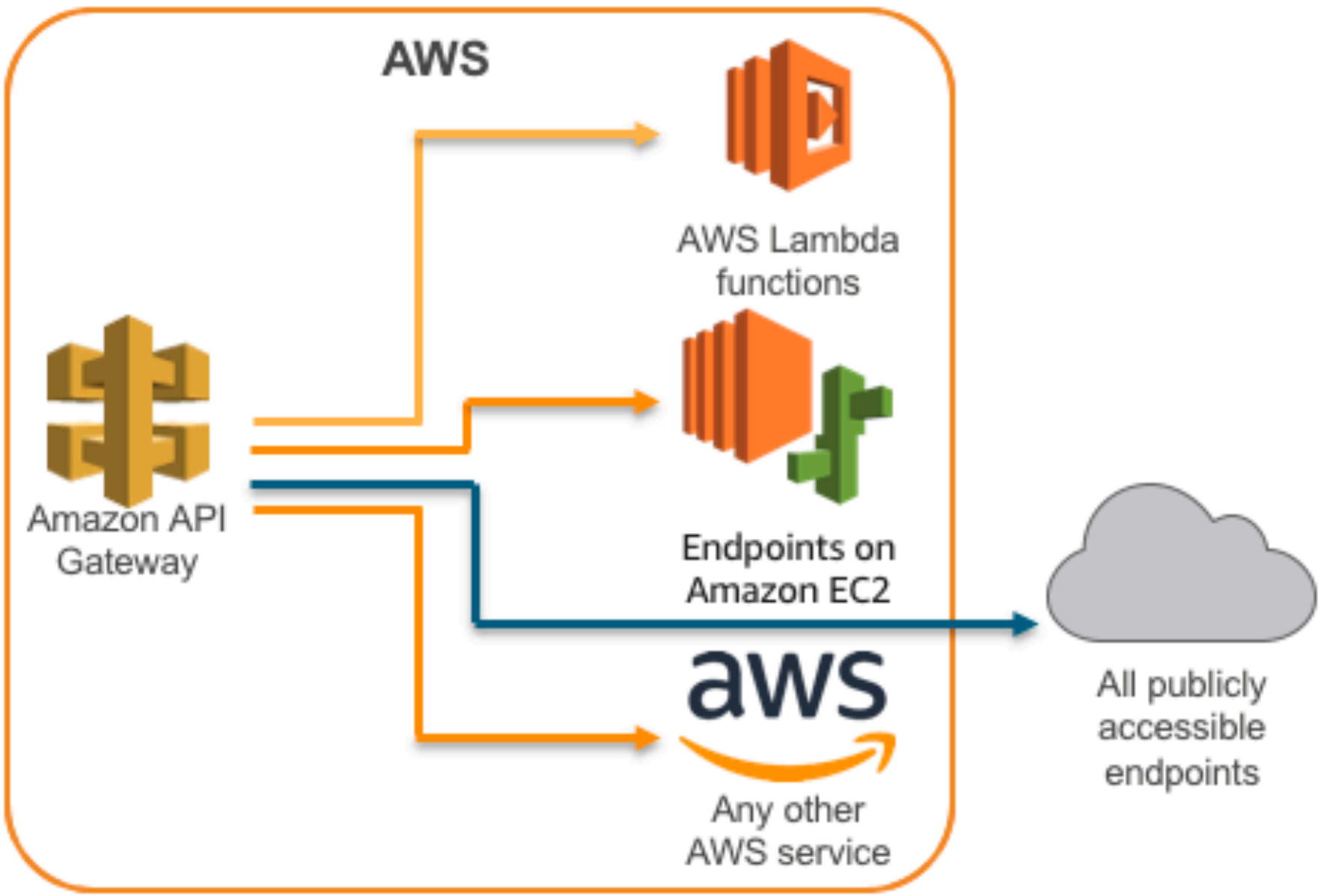
Serverless Applications

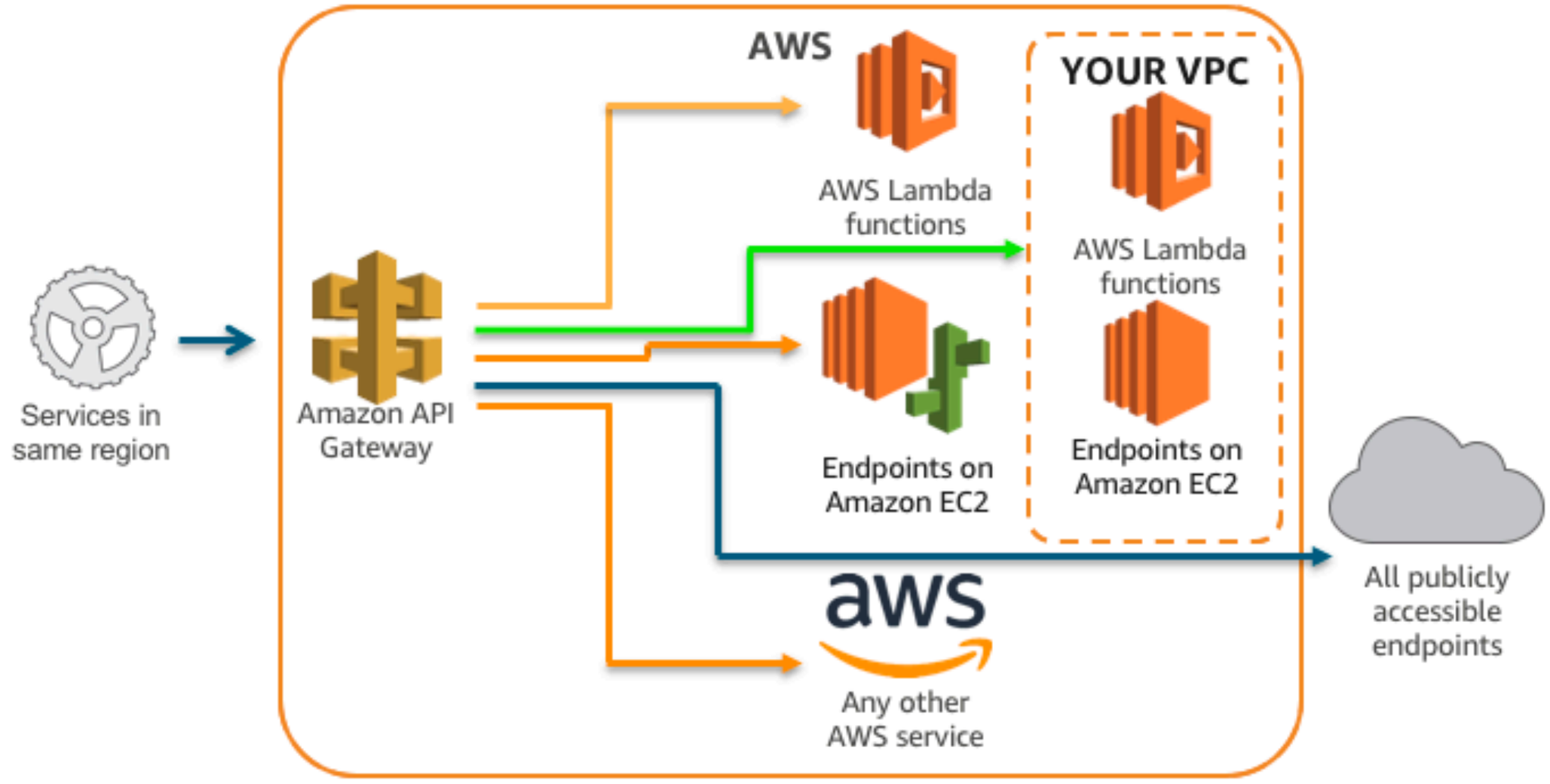
Serverless Lens

Amazon API Gateway

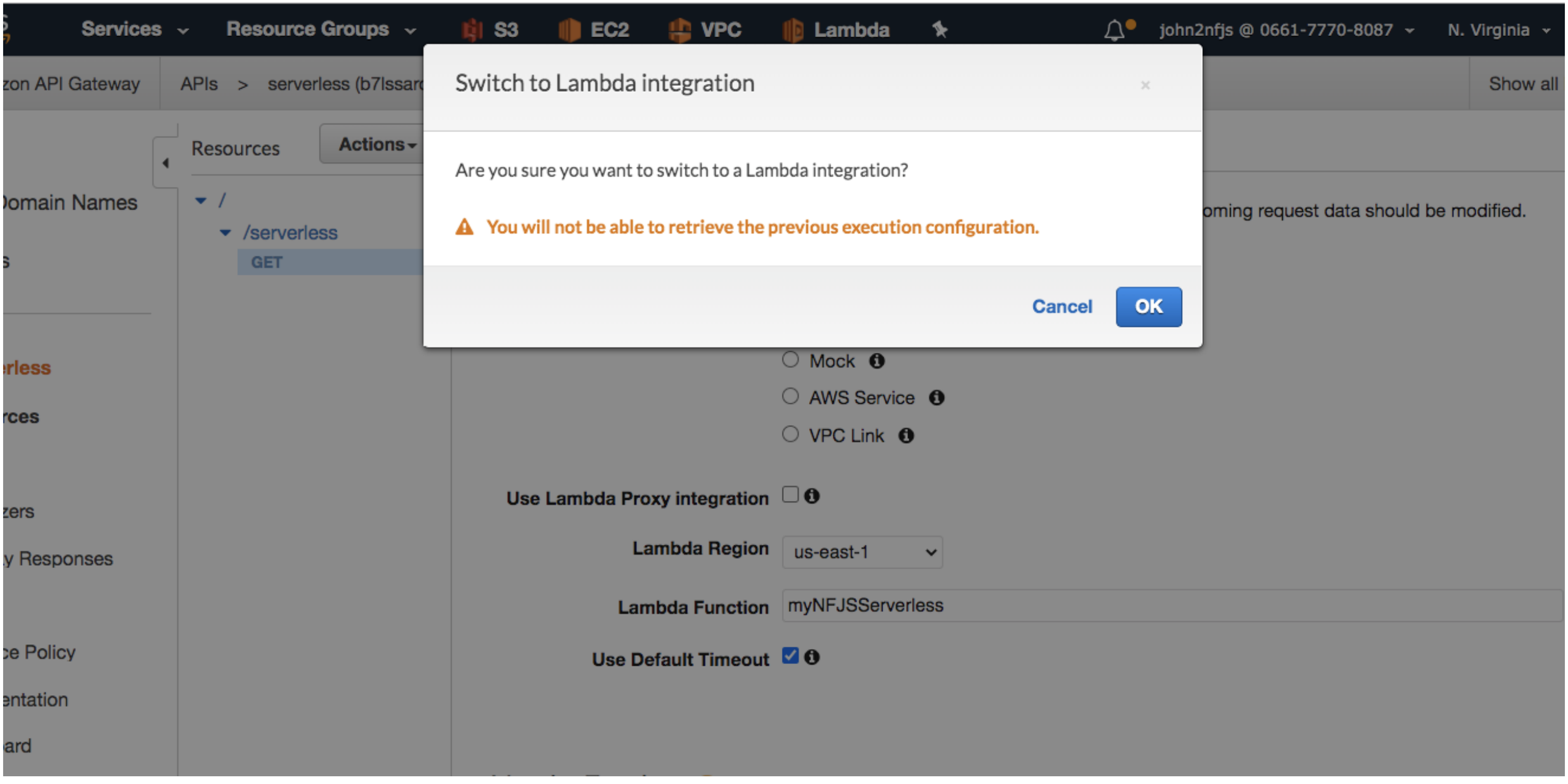


https://aws.amazon.com/blogs/compute/introducing-amazon-api-gateway-private-endpoints/


AWS
AWS Lambda functions
Amazon API Gateway
Endpoints on Amazon EC2
aws
Any other AWS service
All publicly accessible endpoints

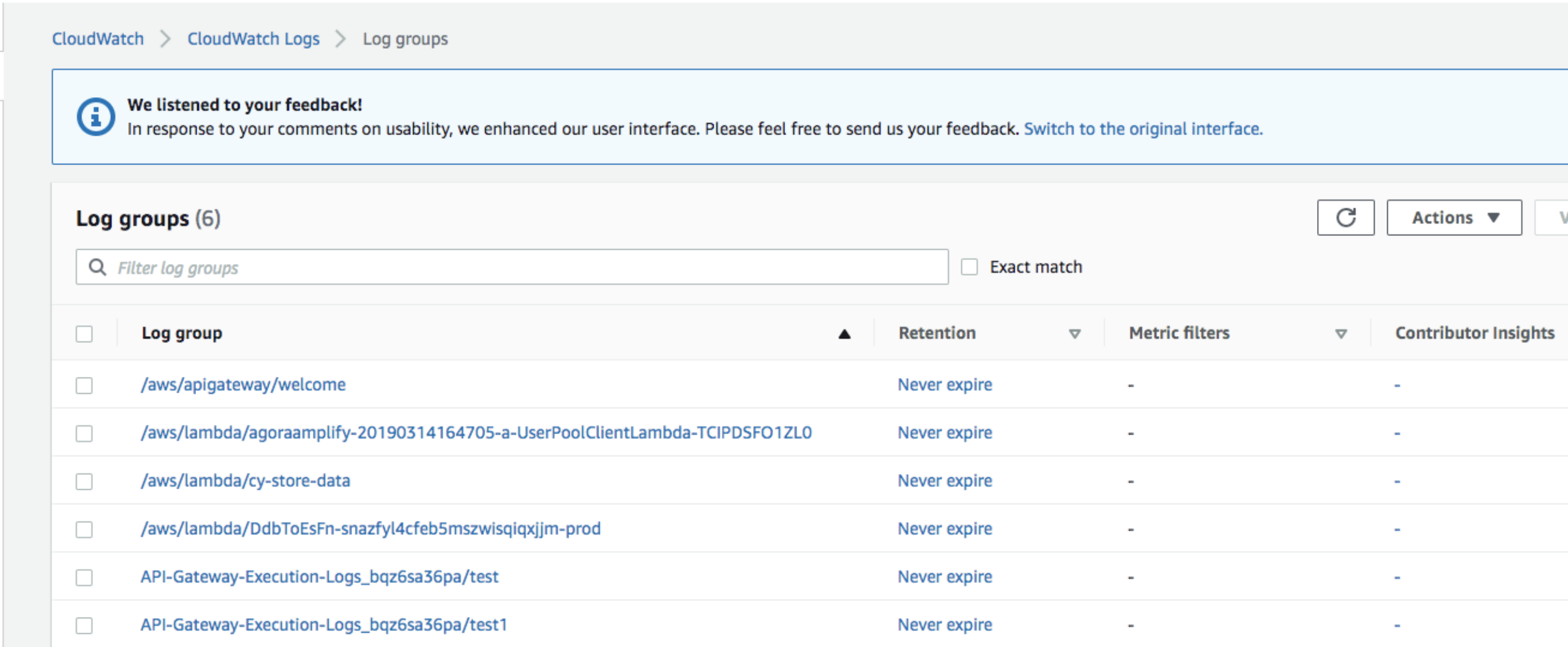

AWS
YOUR VPC
AWS Lambda functions
AWS Lambda functions
Services in same region
Amazon API Gateway
Endpoints on Amazon EC2
Endpoints on Amazon EC2
All publicly accessible endpoints
Any other AWS service

Hook API Gateway with Lambda function



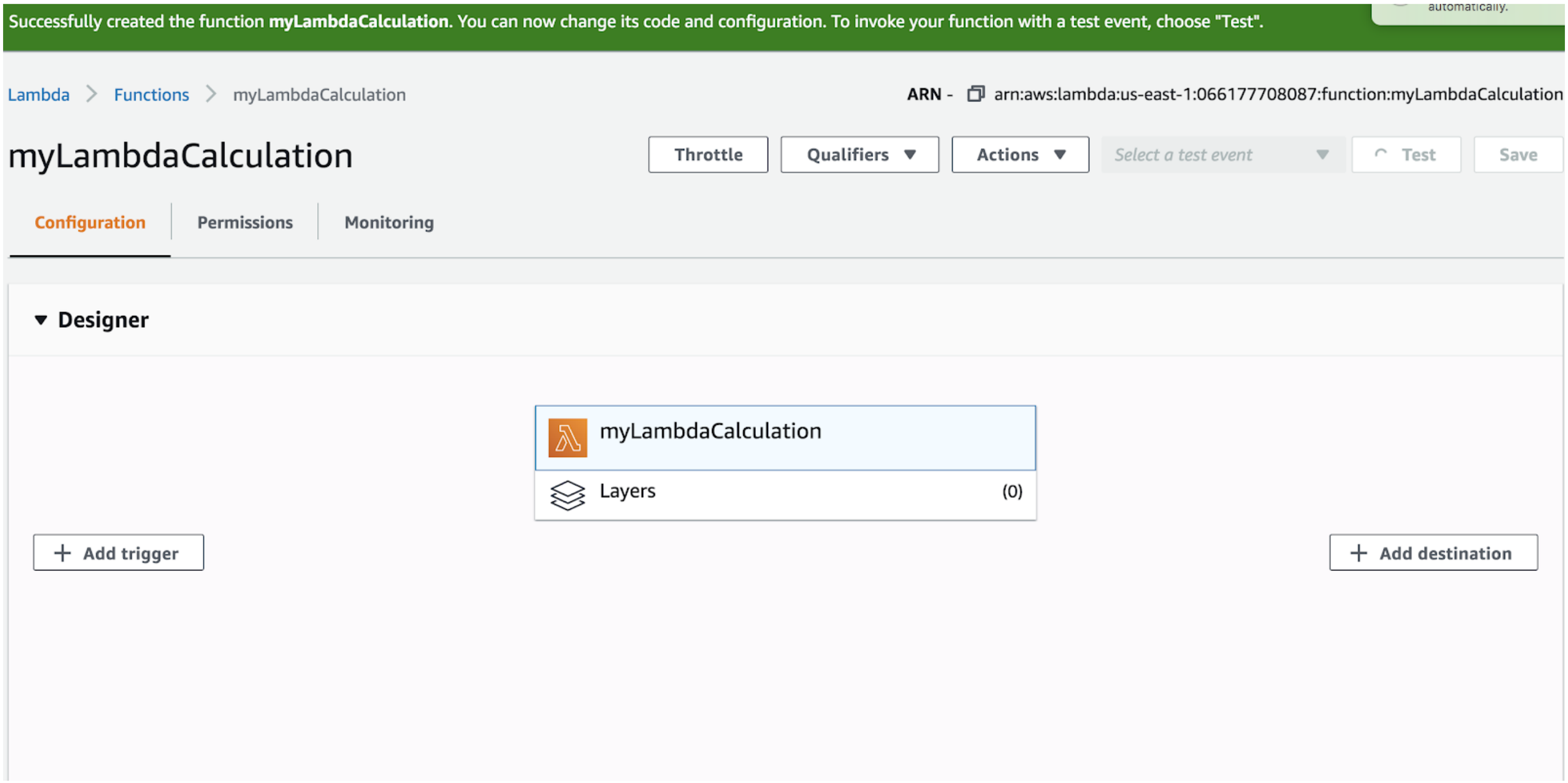
Hook API Gateway with Lambda function

CloudWatch Logs in API Gateway



CloudWatch Logs in API Gateway

Serverless Lambda POST method Calculator method



Serverless Lambda POST method Calculator method

CORS Configuration for Lambda Proxy

test-cors.org

Use this page to test CORS requests. You can either send the CORS request to a remote server (to test if CORS is supported), or send the CORS request to a test server (to explore certain features of CORS). Send feedback or browse the source here: https://github.com/monsur/test-cors.org.

Client

HTTP Method: GET

With credentials? ☐

Request Headers

Request Content

Send Request

Server

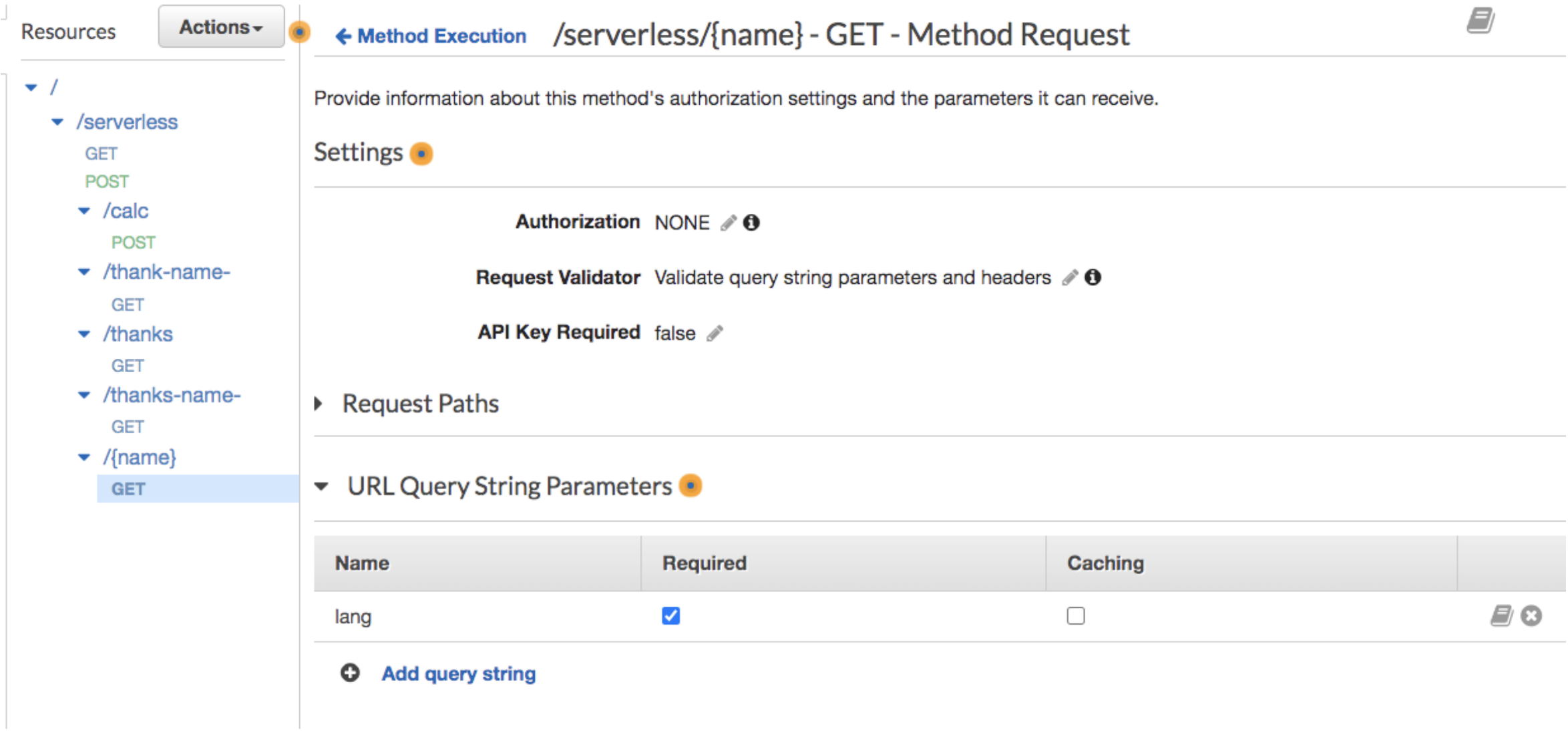
Remote | Local

Remote URL

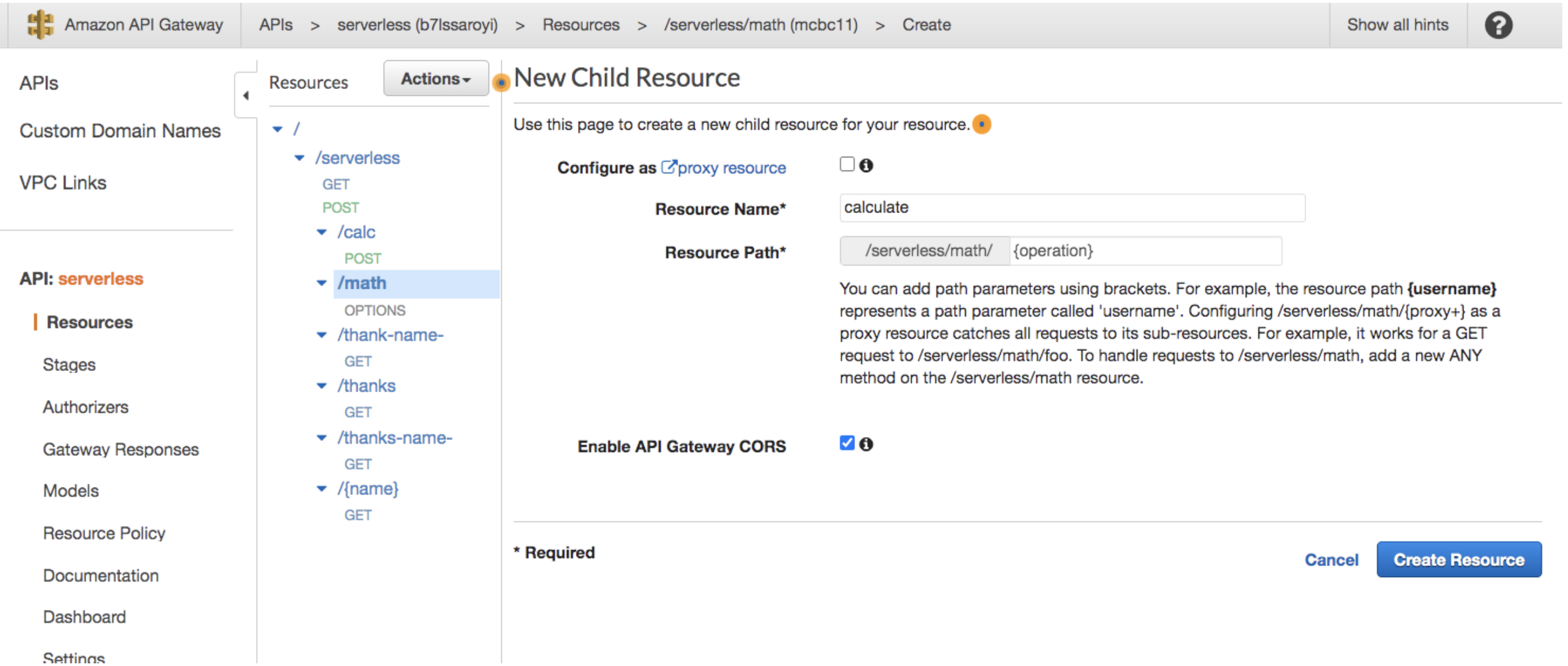
CORS Lambda

```
const moment = require('moment');
const thanks = {
    "en": "Thank you, Thanks",
    "fr": "Merci",
    "hi": "Dhanyavad",
    "es": "gracias",
    "jp": "arigato",
    "ru": "spasiba",
          "it": "Grazie",
    "gr": "danke"
}

// Async handler
exports.handler = async (event) => {
    // path parameter
    let name = event.pathParameters.name;

    // query parameter
    let {lang, ...info} = event.queryStringParameters || {};

    let message = `${thanks[lang] ? thanks[lang] : thanks['en'] } ${name}`;
    let response = {
        message: message,
        info: info,
        timestamp: moment().unix()
    }
    // return http response body for api gateway
      return {
          statusCode: 200
          ,
          headers:{
              "Access-Control-Allow-Origin": "*"
          },
          body: JSON.stringify(response)
      }
}
```

Request Validator



POST Request



Passing Parameters using Event Object

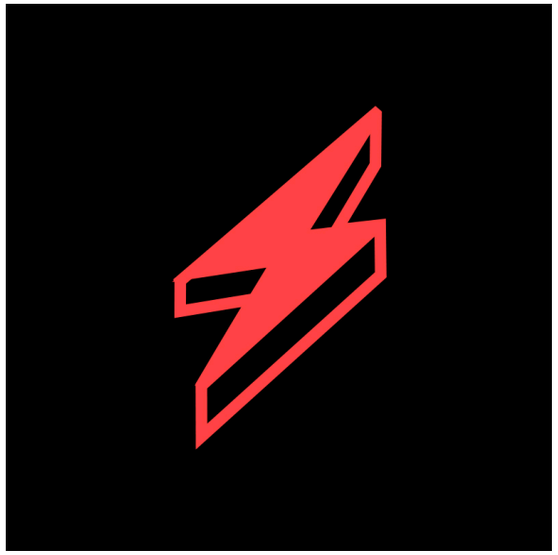

```
{
  "body": "eyJ0ZXN0IjoiYm9keSJ9",
  "resource": "/{proxy+}",
  "path": "/path/to/resource",
  "httpMethod": "POST",
  "isBase64Encoded": true,
  "queryStringParameters": {
    "lang": "fr"
  },
  "multiValueQueryStringParameters": {
    "foo": [
      "bar"
    ]
  },
  "pathParameters": {
    "name": "NFJSThanks"
  },
  "stageVariables": {
    "baz": "qux"
  },
```

API gateway Best Practices

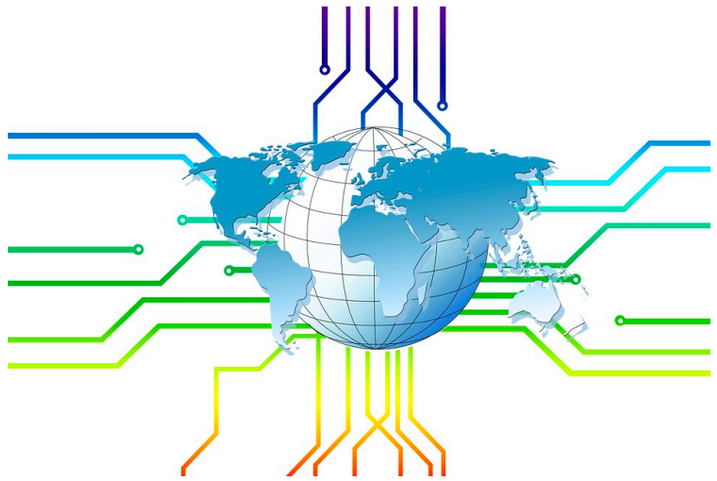
- Keep API definitions lean - Move all logic to Backend lambda functions
- Lambda proxy integration - Data manipulation in Lambda
- Schema validations in API gateway
- Return useful responses to caller

API gateway Best Practices

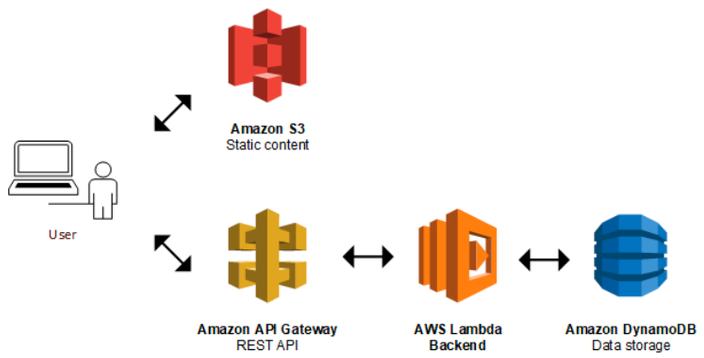
- Enable cloud watch logging
- Use custom domains in production
- Closer to customers deployment - Low Latency
- Edge optimized endpoints
- API caching



Serverless Framework



Serverless Architecture



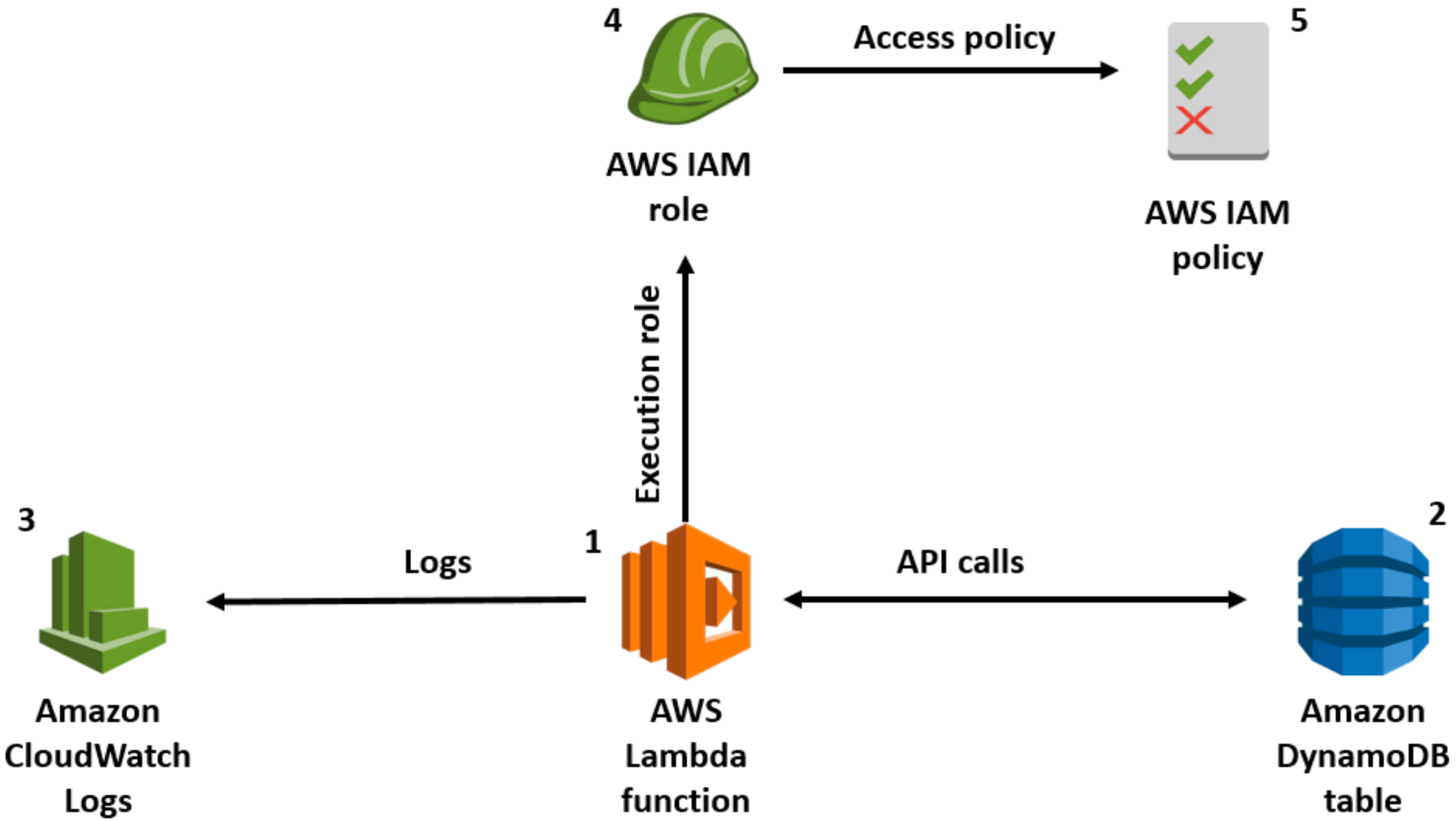
AWS Applications

AWS Step Functions

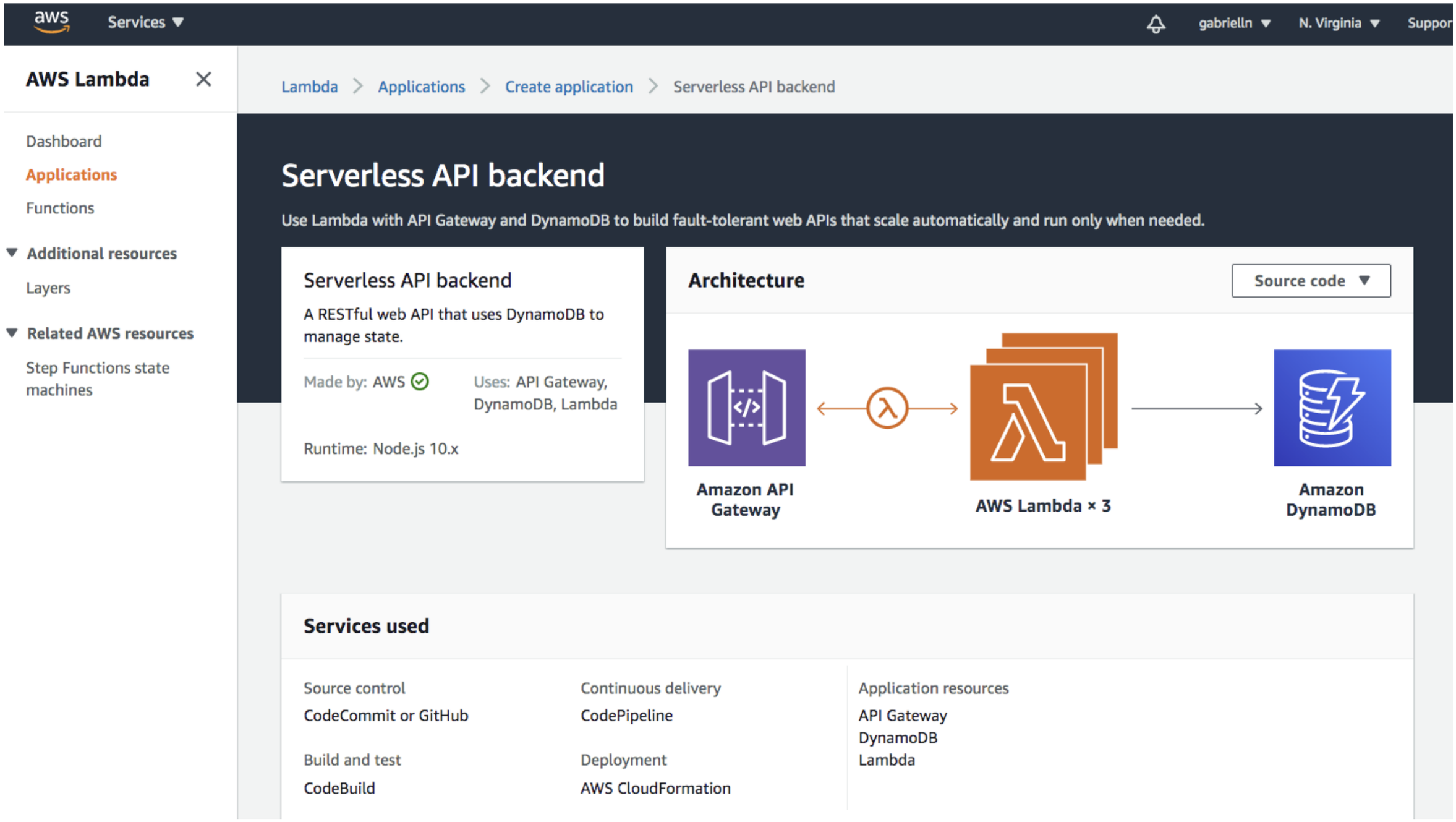
Serverless Applications

Serverless Lens

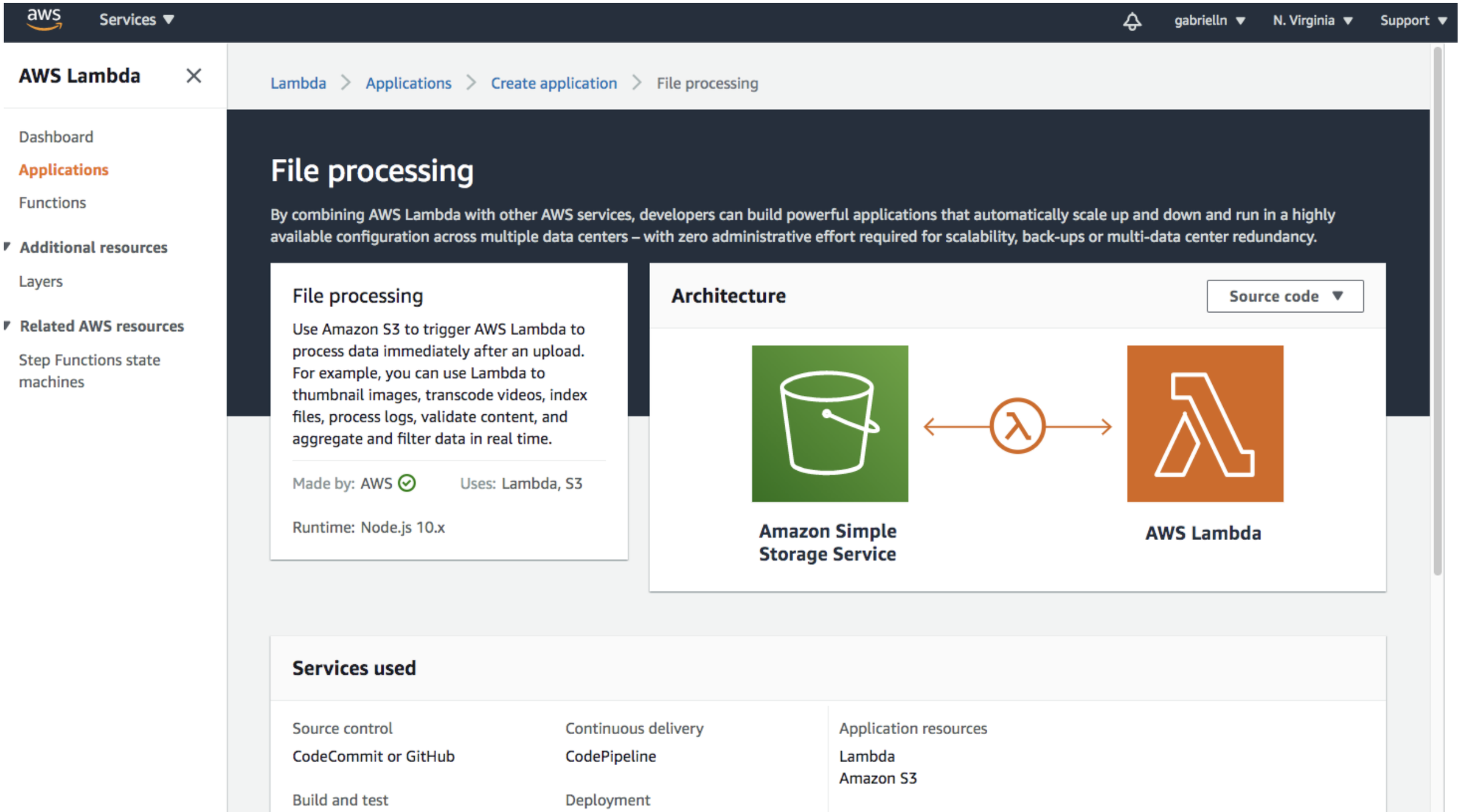
AWS Applications

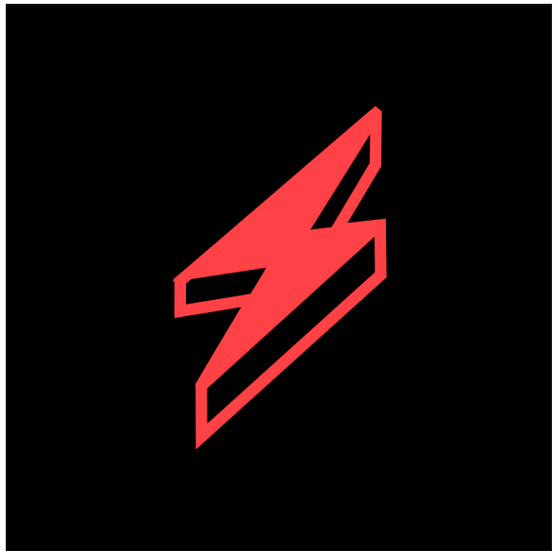

4
Access policy
5
AWS IAM
role
AWS IAM
policy
Execution role
3
Logs
1
API calls
2
Amazon
CloudWatch
Logs
AWS
Lambda
function
Amazon
DynamoDB
table

Serverless API backend application

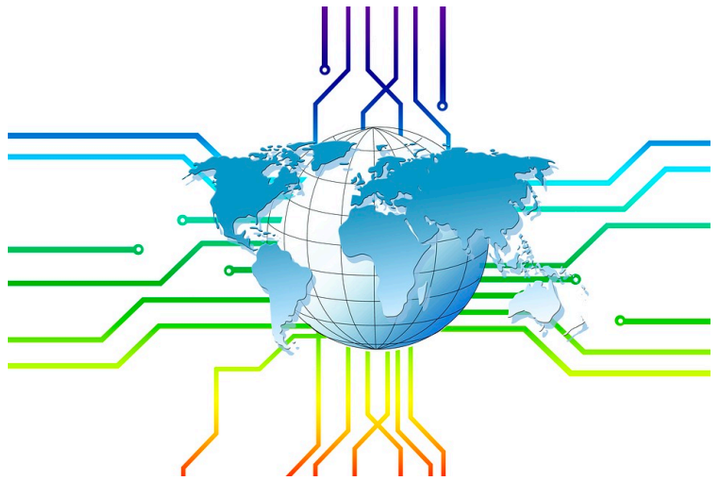


File Processing application

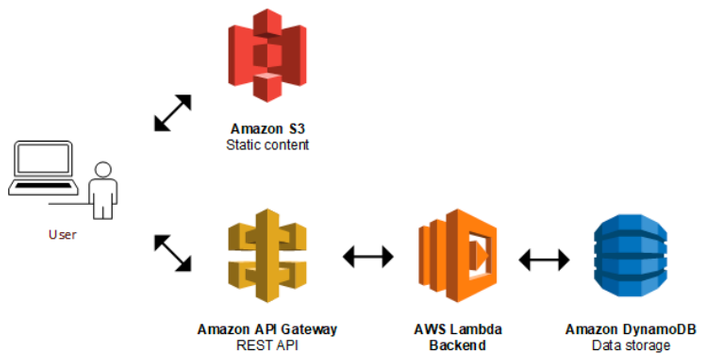




Serverless Framework



Serverless Architecture



AWS Applications

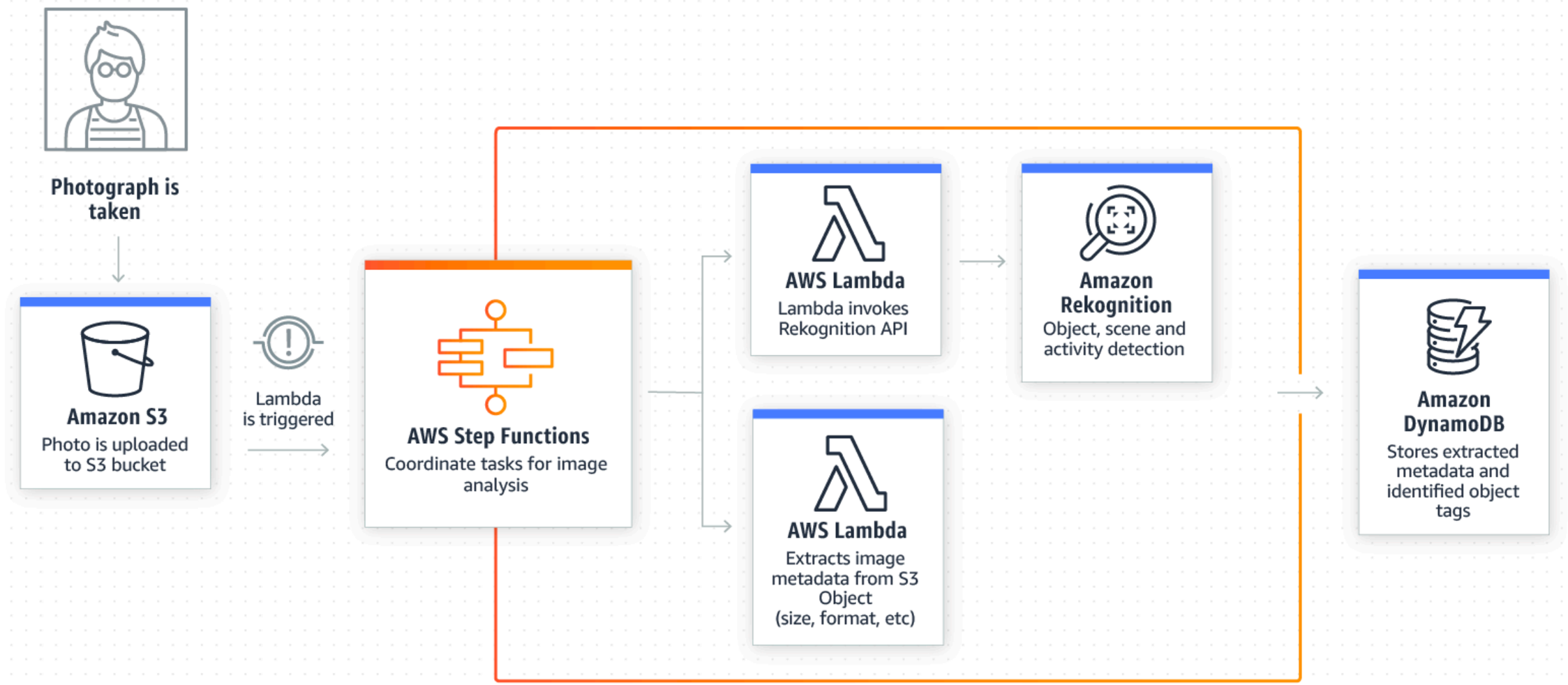
AWS Step Functions

Serverless Applications

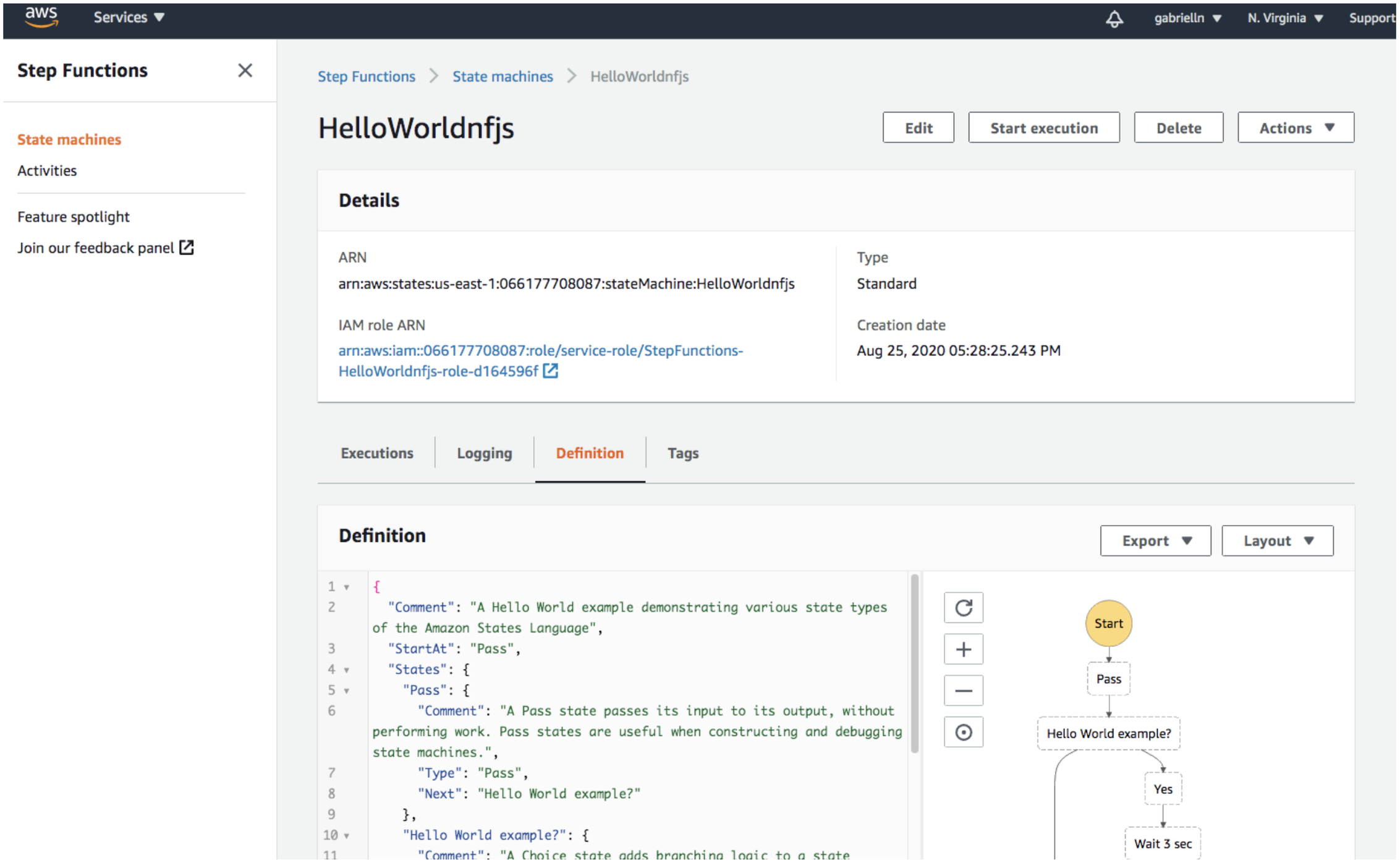
Serverless Lens

AWS Step Functions

Serverless Deep Dive - AWS Step functions



NFJS Hello World Step Function

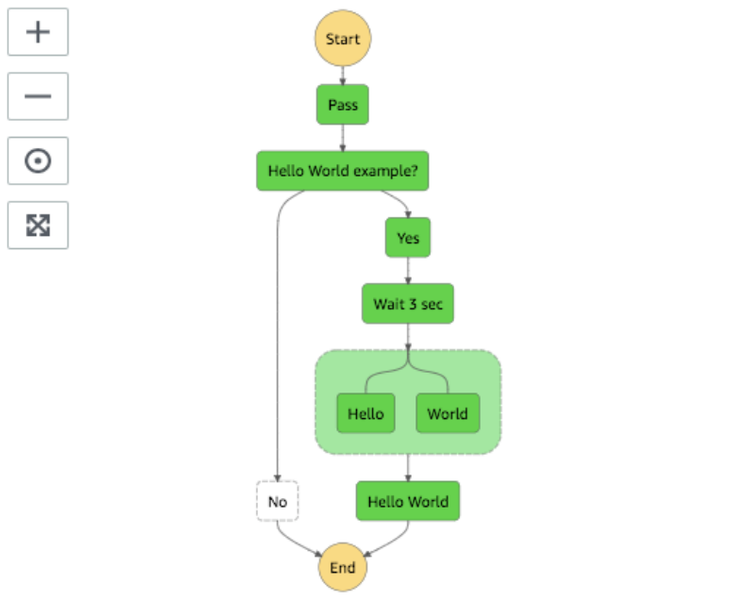


Details | Execution input | Execution output | Definition

```
1 ▾ {
2     "IsHelloWorldExample": true
3   }
```

Graph inspector

Export ▼ Layout ▼



Details | Step input | Step output

Select a step to view its details.

NFJS Calculator Lambda Step function

ARN

arn:aws:states:us-east-1:066177708087:stateMachine:nfjsCalculator1015

IAM role ARN

arn:aws:iam::066177708087:role/service-role/StepFunctions-nfjsCalculator1015-role-b0d421c6

Type

Standard

Creation date

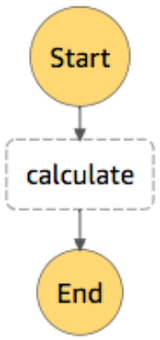
Oct 16, 2020 01:13:21.634 AM

Executions | Logging | Definition | Tags

Definition

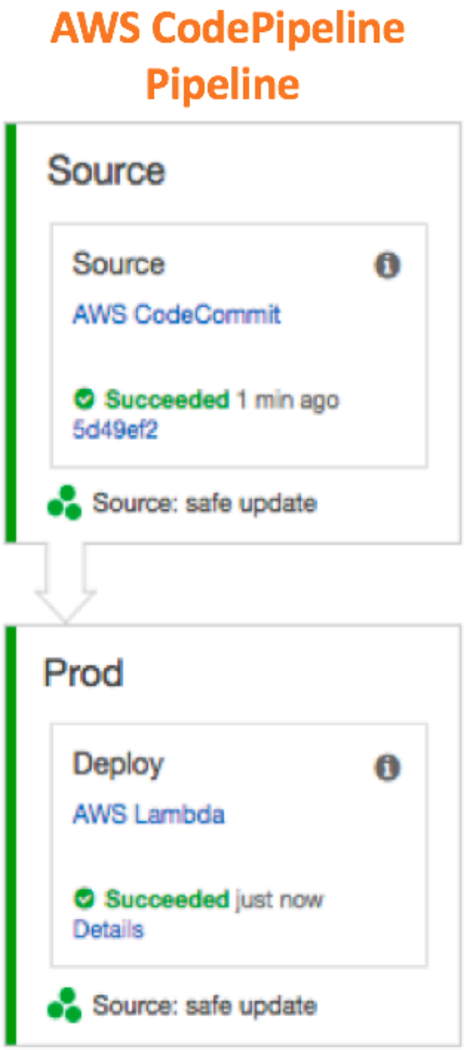
Export ▼ Layout ▼

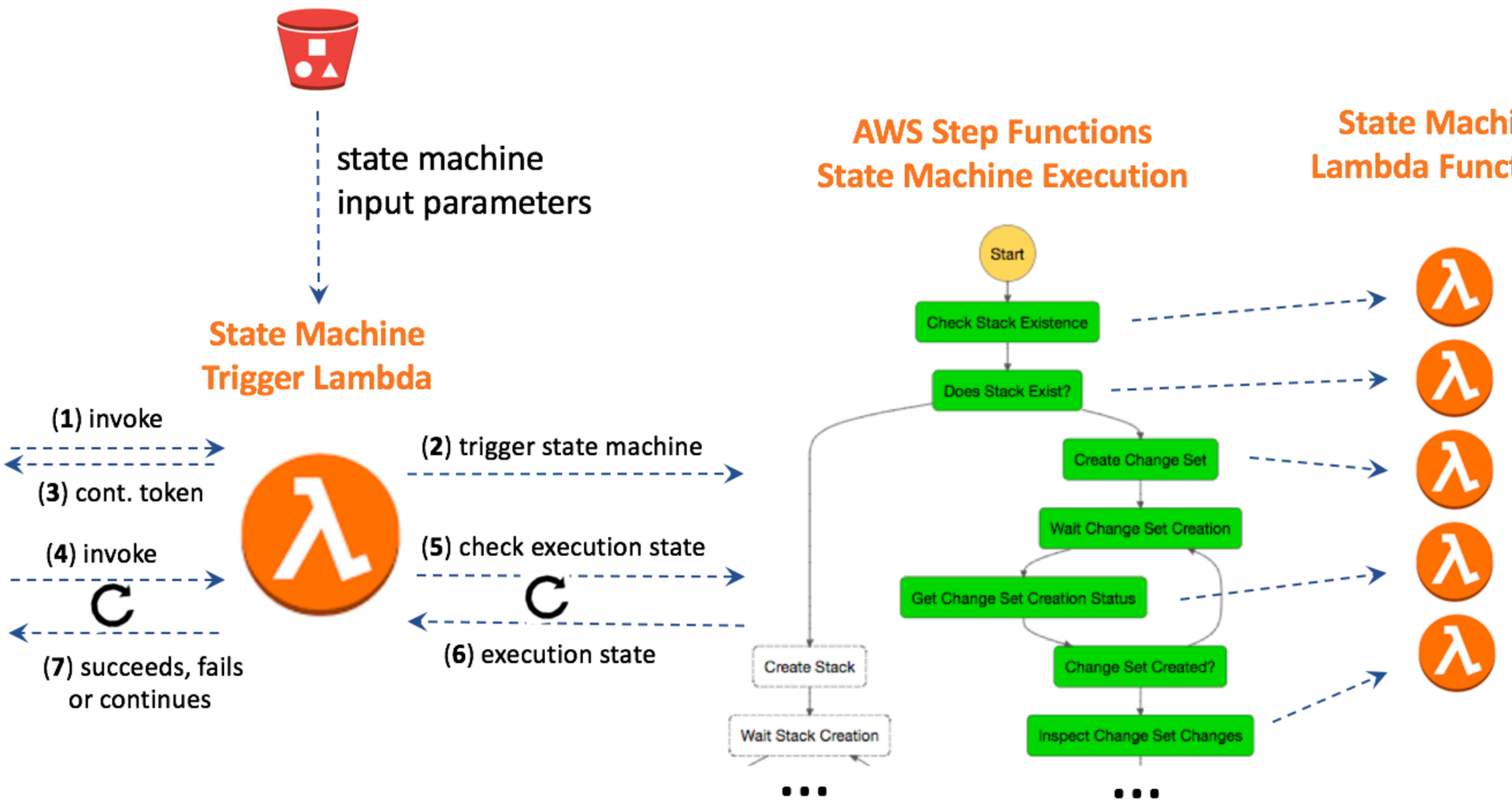
```
{
  "Comment": "A Hello World example of the Amazon States Language using Pass states",
  "StartAt": "calculate",
  "States": {

    "calculate": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:066177708087:function:myLambdaCalculation",
      "InputPath": "$.lambda",
      "End": true
    }
  }
}
```



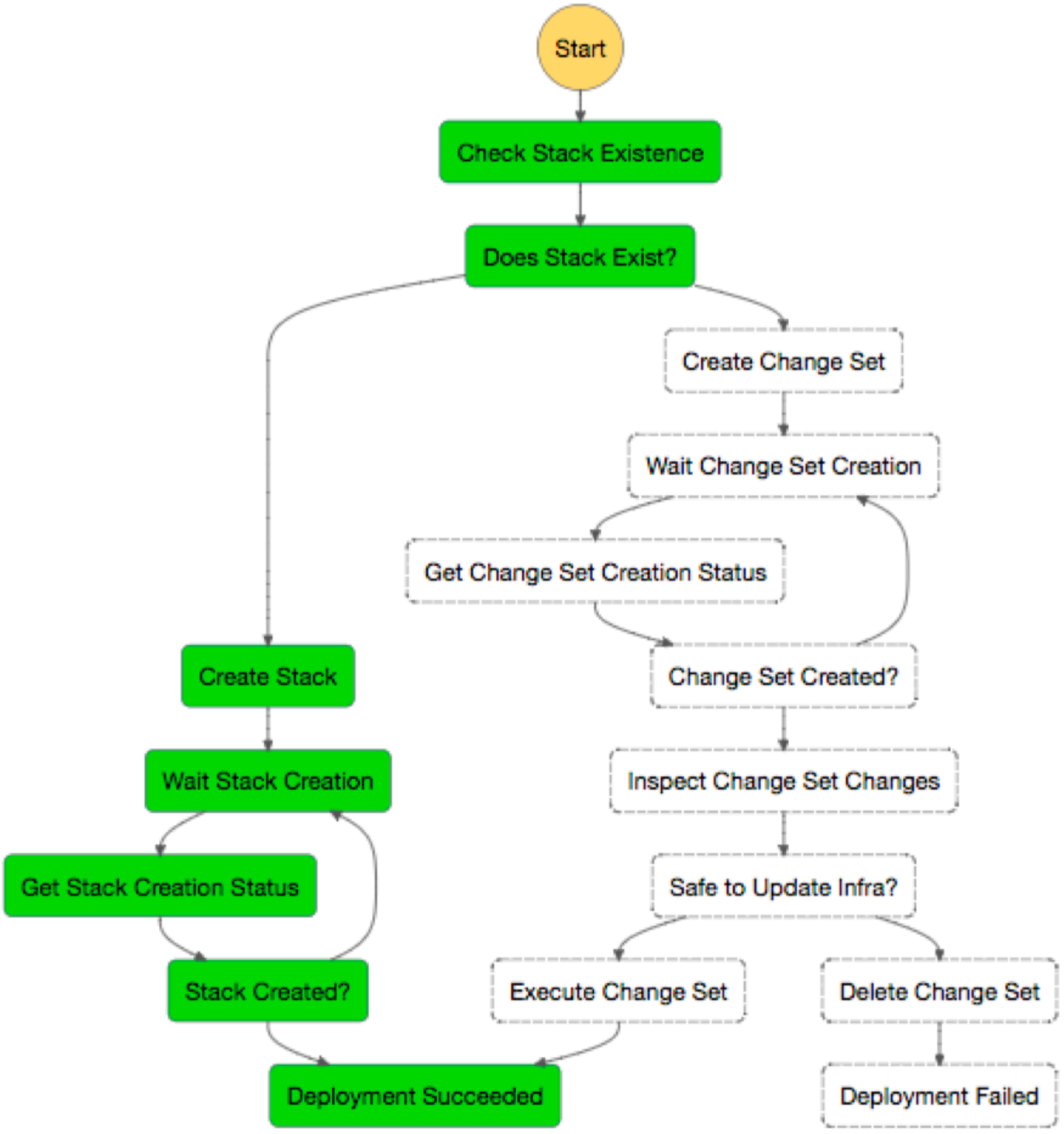
NFJS Calculator

Code pipeline



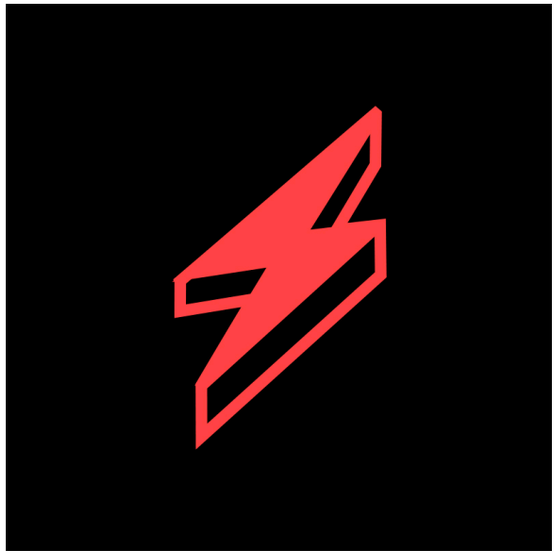


https://aws.amazon.com/blogs/devops/using-aws-step-functions-state-machines-to-handle-workflow-driven-aws-codepipeline-actions/

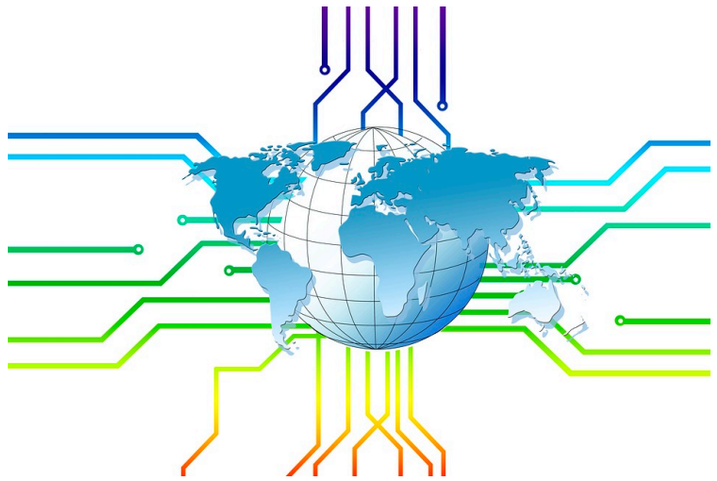

Start
Check Stack Existence
Does Stack Exist?
Create Change Set
Wait Change Set Creation
Get Change Set Creation Status
Change Set Created?
Inspect Change Set Changes
Safe to Update Infra?
Execute Change Set
Delete Change Set
Deployment Failed
Create Stack
Wait Stack Creation
Get Stack Creation Status
Stack Created?
Deployment Succeeded

Step Functions Best Practices

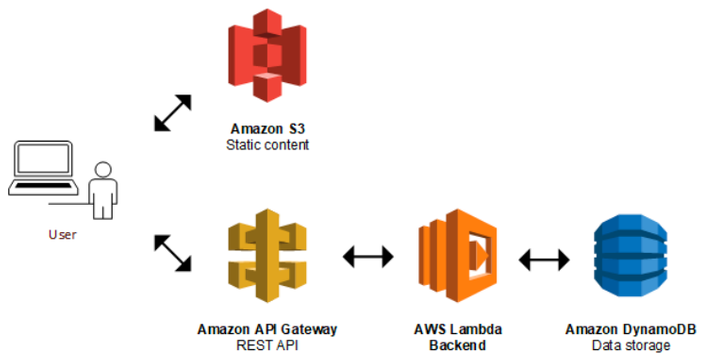
- Use Timeouts in tasks states
    - Something goes wrong, how to handle it?
- Lambda can run into errors - Retries or Cachers
- Large amount of data between states >32 k
    - Use S3 to keep data-> Pass ARN between states



Serverless Framework



Serverless Architecture

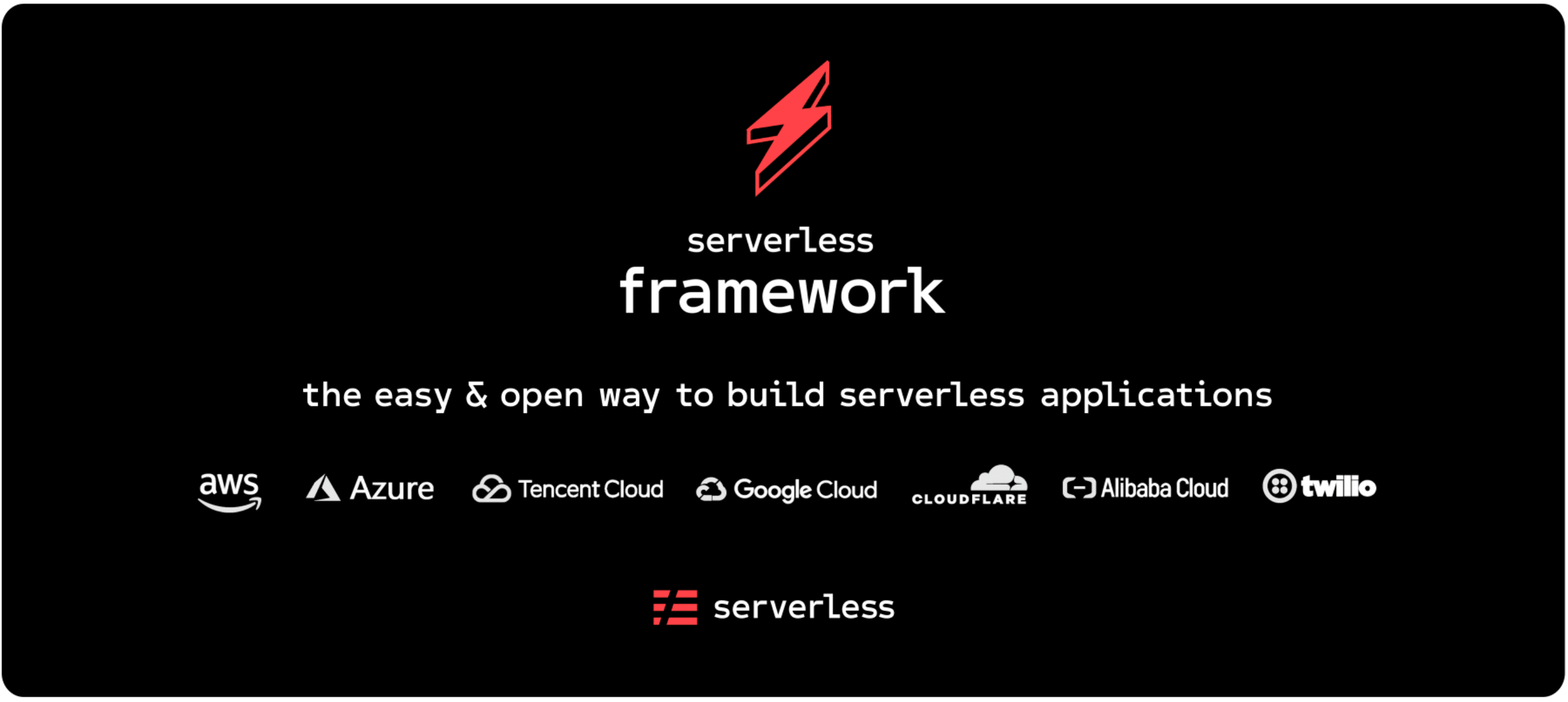


AWS Applications

AWS Step Functions

Serverless Applications

Serverless Lens



Serverless Framework Applications

https://app.serverless.com/

Serverless Framework Applications

app.serverless.com/rohitbh/apps/new

Paused

apps

create app

apps

ci/cd

org

support

fullstack

Deploy a complete, serverless, full-stack application built on AWS Lambda, AWS HTTP API, Express.js, React, DynamoDB and more.

express.js

Deploy a serverless Express.js API that is cheap and massively scalable on AWS Lambda and AWS HTTP API. Automatically set up a custom domain and free AWS ACM SSL certificate.

react.js

Deploy a serverless React.js application that is cheap and massively scalable on AWS S3, AWS CloudFront. Automatically set up a custom domain and free AWS ACM SSL certificate.

serverless framework

Add an existing Serverless Framework project from your local computer.

graphql

Deploy a serverless GraphQL API on AWS AppSync that is easy to configure and scale.

vue.js

Deploy a serverless Vue.js application on AWS S3, AWS Cloudfront. Automatically set up a custom domain and free AWS ACM SSL certificate.

deploy now

fullstack

this must be deployed from the command line interface

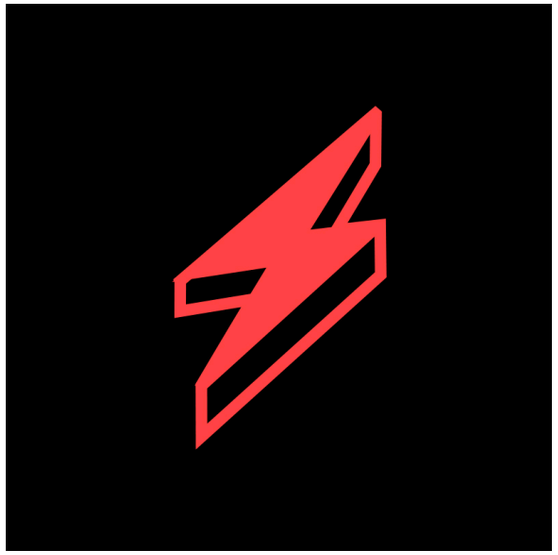
install

```
$ npm i -g serverless && serverless init sf_ykFxn3rX
```

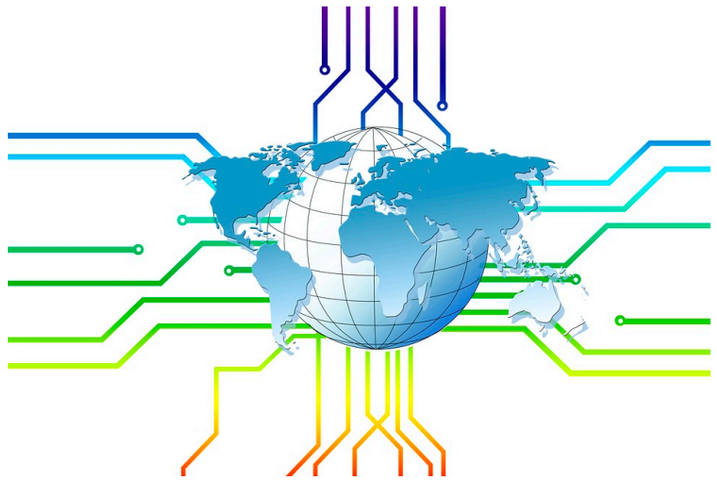
deploy

```
$ cd nfjs-fullstack-app && serverless deploy
```

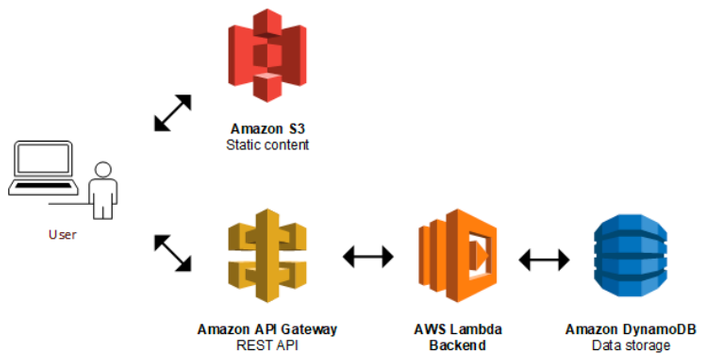
waiting for deploy...



Serverless Framework



Serverless Architecture



AWS Applications

AWS Step Functions

Serverless Applications

Serverless Lens


AWS
SAM

No server management

There is no need to provision or maintain any servers. There is no software or runtime to install, maintain, or administer.

Flexible scaling

Your application can be scaled automatically or by adjusting its capacity through toggling the units of consumption (e.g. throughput, memory) rather than units of individual servers.

High availability

Serverless applications have built-in availability and fault tolerance. You do not need to architect for these capabilities since the services running the application provide them by default.

No idle capacity

You do not have to pay for idle capacity. There is no need to pre- or over-provision capacity for things like compute and storage. For example, there is no charge when your code is not running.

Amazon S3 hosts static website content such as HTML, CSS, JavaScript, etc...

AMAZON S3
Object storage and website hosting

Amazon Cognito registers and authenticates new users to your application

AMAZON COGNITO USER POOL
User directory and authentication

Your app's serverless backend receives dynamic API calls and performs business logic

AMAZON API GATEWAY
RESTful API

AWS LAMBDA
Serverless compute

AMAZON DYNAMODB
NoSQL database

AWS Lambda

Dashboard
Applications
Functions

Additional resources
Code signing configurations
Layers
Replicas

Related AWS resources
Step Functions state machines

serverless-todo — version 1.0.3

Review, configure and deploy

Copy as SAM Resource

Application details

Author	Source code URL	Description	Report a vulnerability
evanchiu	https://github.com/evanchiu/serverless-todo/tree/v1.0.3	React TodoMVC with a Serverless backend	If you believe this application poses a security risk, please file a vulnerability report.

Template

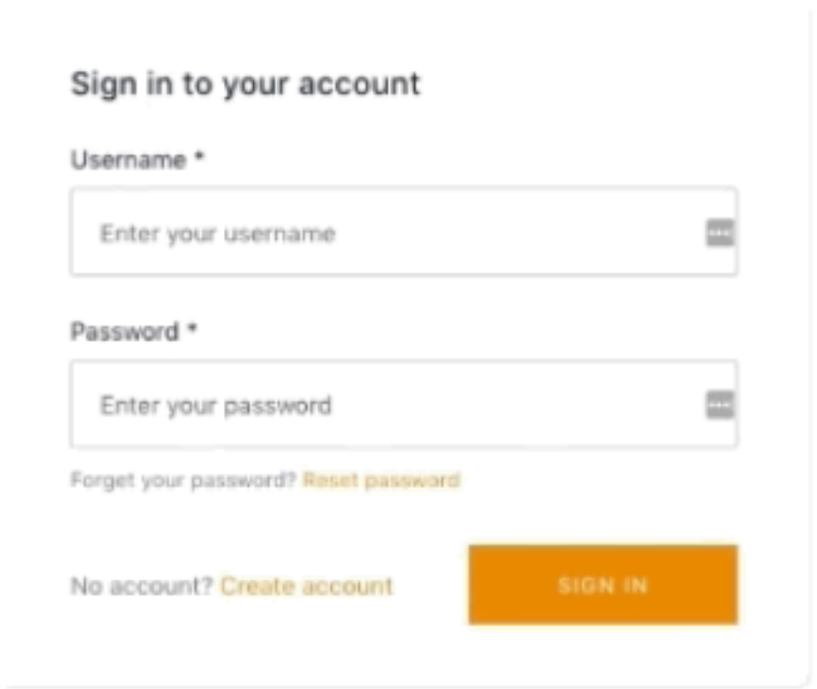
```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: React TodoMVC with a Serverless backend
Globals:
```

AWS Amplify

Create-react-app with AWS Amplify Auth

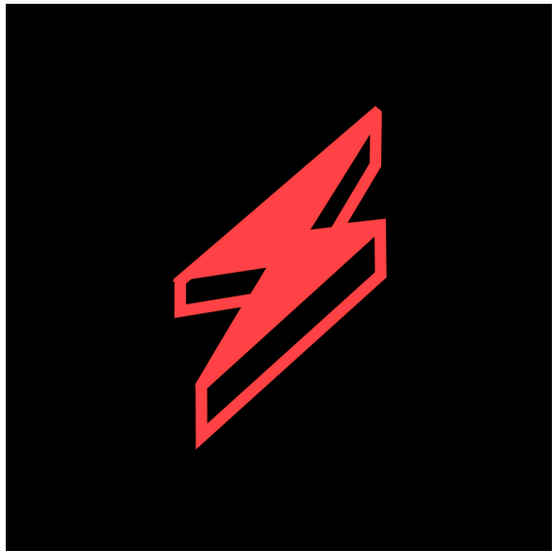
This auth starter implements withAuthenticator HOC to provide a basic authentication flow for signing up signing in users as well as protected client side routing using AWS Amplify. Auth features: User sign up, User sign in, Multi-factor Authentication, User sign-out.

View Demo

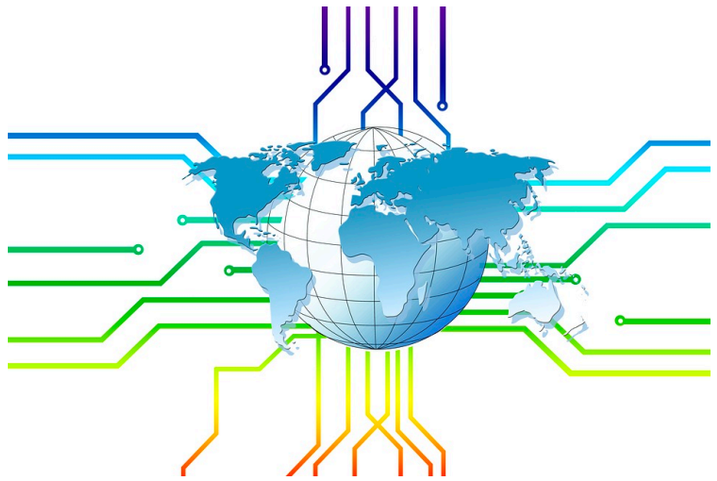


https://github.com/rohitbhardwaj/create-react-app-auth-amplify/tree/master

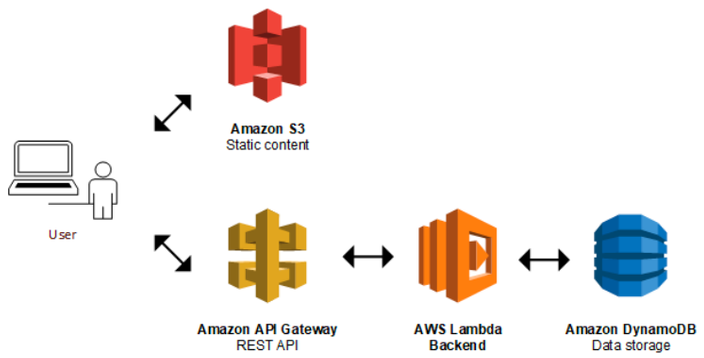
https://master.d1ogtsd1wqph9d.amplifyapp.com



Serverless Framework



Serverless Architecture



AWS Applications

AWS Step Functions

Serverless Applications

Serverless Lens

AWS Serverless Lens

event gateway

React to any event with serverless functions across clouds

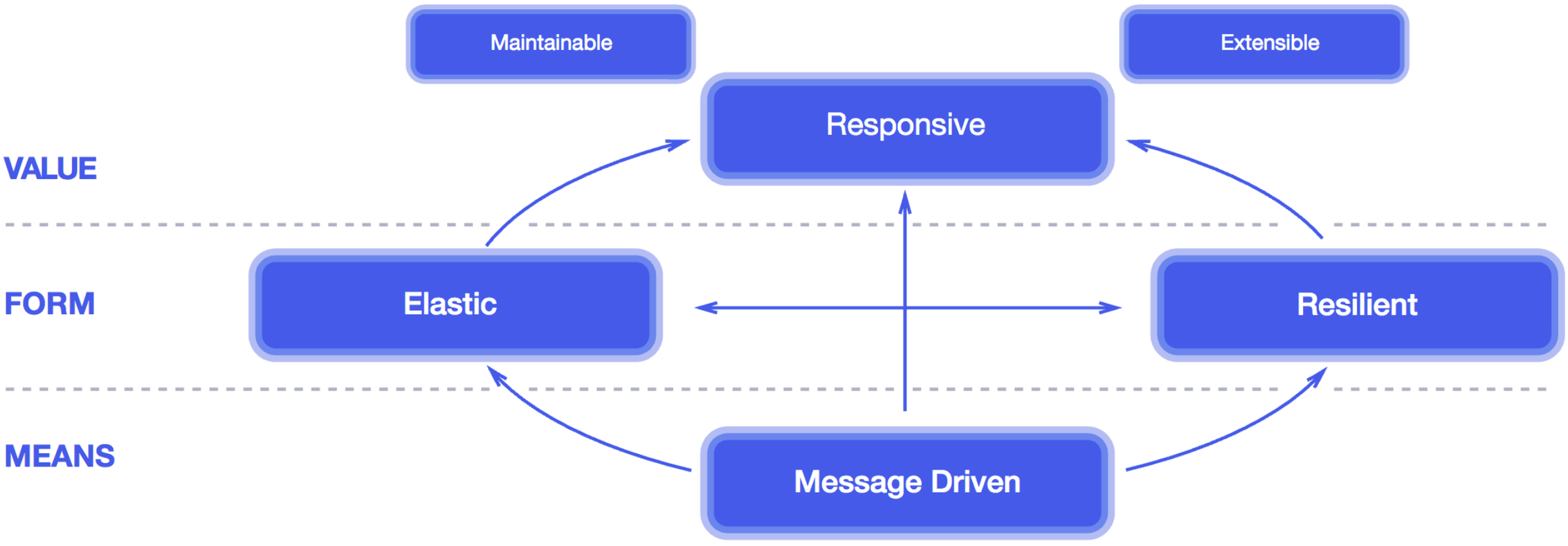
serverless

https://github.com/serverless/event-gateway


event
gateway
AWS Lambda
Azure Functions
Google Cloud Functions
Kubernetes
AWS Kinesis
Webhooks
Devices & IoT
twilio
GitHub
stripe

Design Resilient Architectures

Resilient Architecture



Performance Efficient Architectures

How do you optimize your Serverless application's performance?

- Measure, evaluate, and select optimum capacity units
- Measure and optimize function startup time
- Take advantage of concurrency via async and stream-based function invocations
- Optimize access patterns and apply caching where applicable
- Integrate with managed services directly over functions when possible

Security

Security Best Practices

- Execution roles
- Avoid wild card permissions
- Full Access to Lambda to avoid
- Only required actions
- Use environment variables
- Encrypt sensitive data using KMS
- Do never never log decrypted values
- Lambda VPC-> Security groups, Subnets

How do you control access to your Serverless API?

- Use appropriate endpoint type and mechanisms to secure access to your API - Public/Private

- Use authentication and authorization mechanisms - SAML, JWT

- Scope access based on identity's metadata

How do you manage your Serverless application's security boundaries?

- Evaluate and define resource policies
- Use temporary credentials between resources and components
- Control network traffic at all layers - ingress and egress traffic
- Design smaller, single purpose functions

How do you implement application Security in your workload?

- Review security awareness documents frequently

- Store secrets that are used in your code securely: Secrets Manager - rotation, secure and audited access.

- Store your secrets such as database passwords or API keys in a that allows for Implement runtime protection to help prevent against malicious code execution

- Automatically review workload’s code dependencies/libraries

- Validate inbound events: Sanitize inbound events and validate them against a predefined schema. Test your inputs by using fuzzing techniques (Malformed data).

API Access

- API Keys and usage
- Client Certificates
- CORS Headers
- API gateway resource policies
- IAM Policies
- Lambda Authorizers
- Cognito User Pool Authorizers
- Federated Identity access using Cognito

Operational Excellence

How do you evaluate your Serverless application's health?

- Understand, analyze, and alert on metrics provided out of the box. Examples of key metrics include function errors, queue depth, failed state machine executions, and response times.

- Use distributed tracing and code is instrumented with additional context

- Use structured and centralized logging

- Use application, business, and operations metrics: Identify key performance indicators (KPIs) based on desired business and customer outcomes.

How do you approach application lifecycle management?

- Use infrastructure as code and stages isolated in separate environments
- Prototype new features using temporary environments
- Use a rollout deployment mechanism
- Use configuration management
- Review the function runtime deprecation policy
- Use CI/CD including automated testing across separate accounts

Cost Optimization

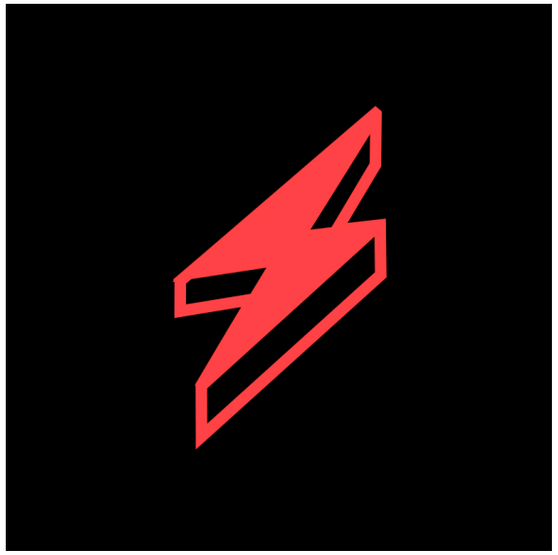
How do you optimize your Serverless application's costs?

- Minimize external calls and function code initialization: Functions may call other managed services and third-party APIs to operate as intended. Functions may also use application dependencies that may not be suitable for ephemeral environments.

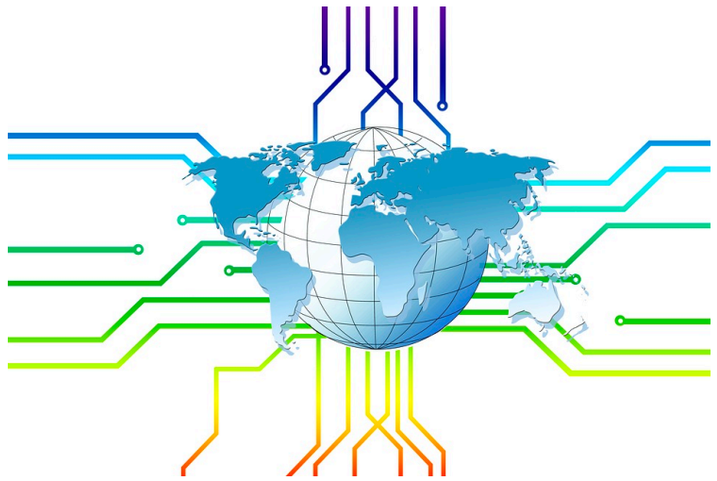
- Optimize logging output and its retention: Review logging level, logging output, and log retention to ensure that they meet your operational needs.

- Optimize function configuration to reduce cost: Functions unit of scale is memory where CPU, Network and I/O are proportionately allocated. Consider benchmarking

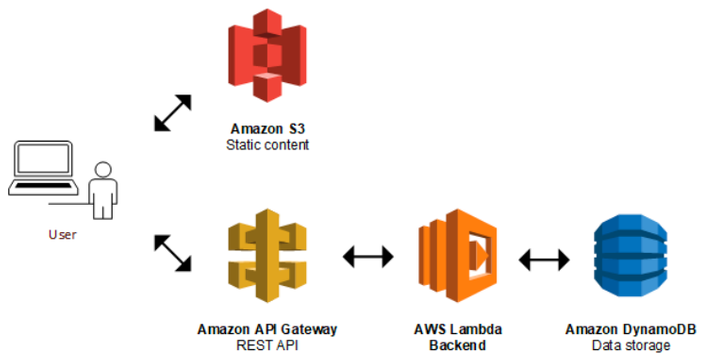
- Use cost-aware usage patterns in code: Reduce the time consumed by running functions by eliminating job-polling or task coordination.



Serverless Framework



Serverless Architecture



AWS Applications

AWS Step Functions

Serverless Applications

Serverless Lens

Thanks

Rohit Bhardwaj

Hands-on Director, Architecture, Salesforce

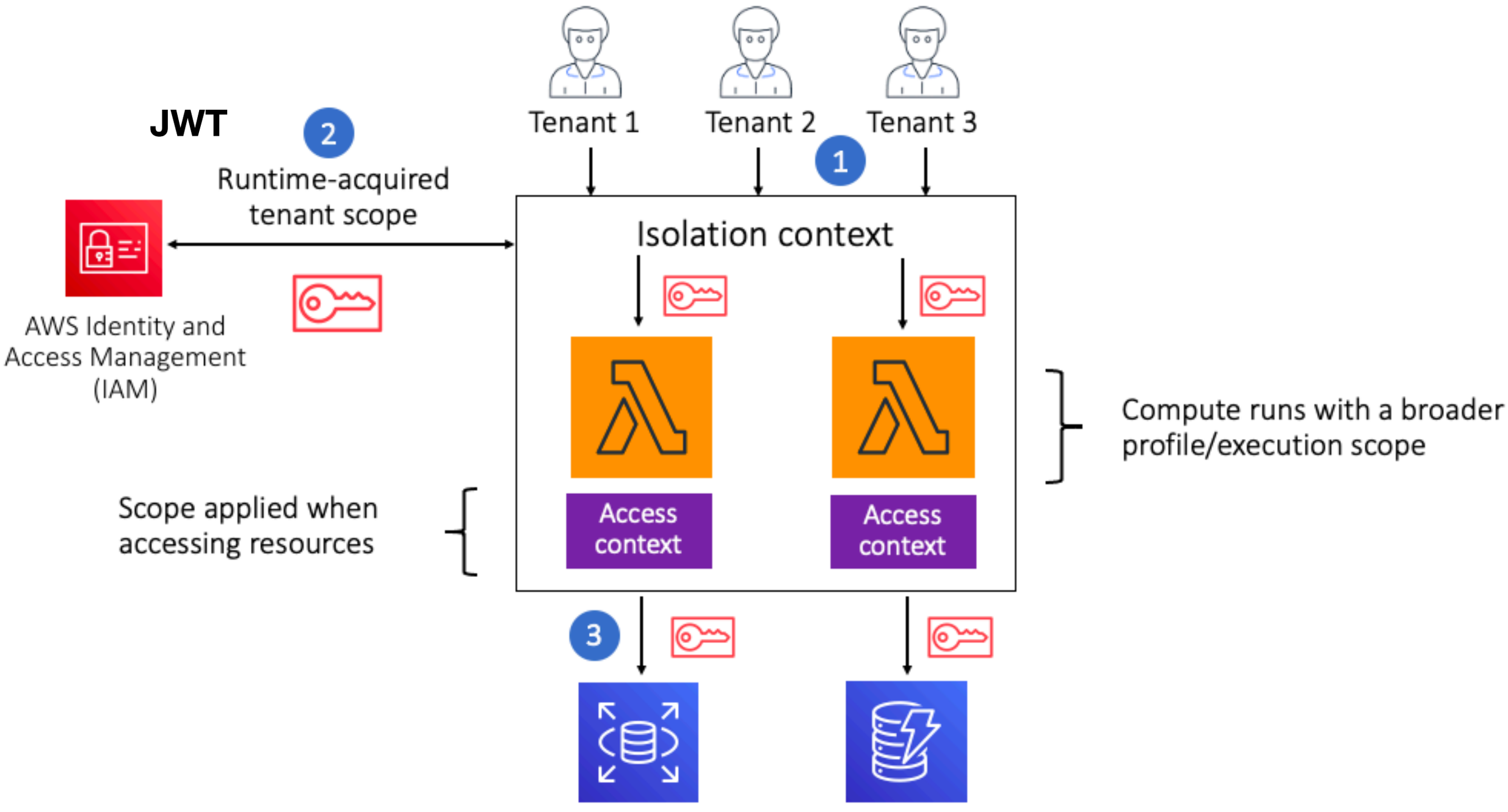
Founder: ProductiveCloudInnovation.com

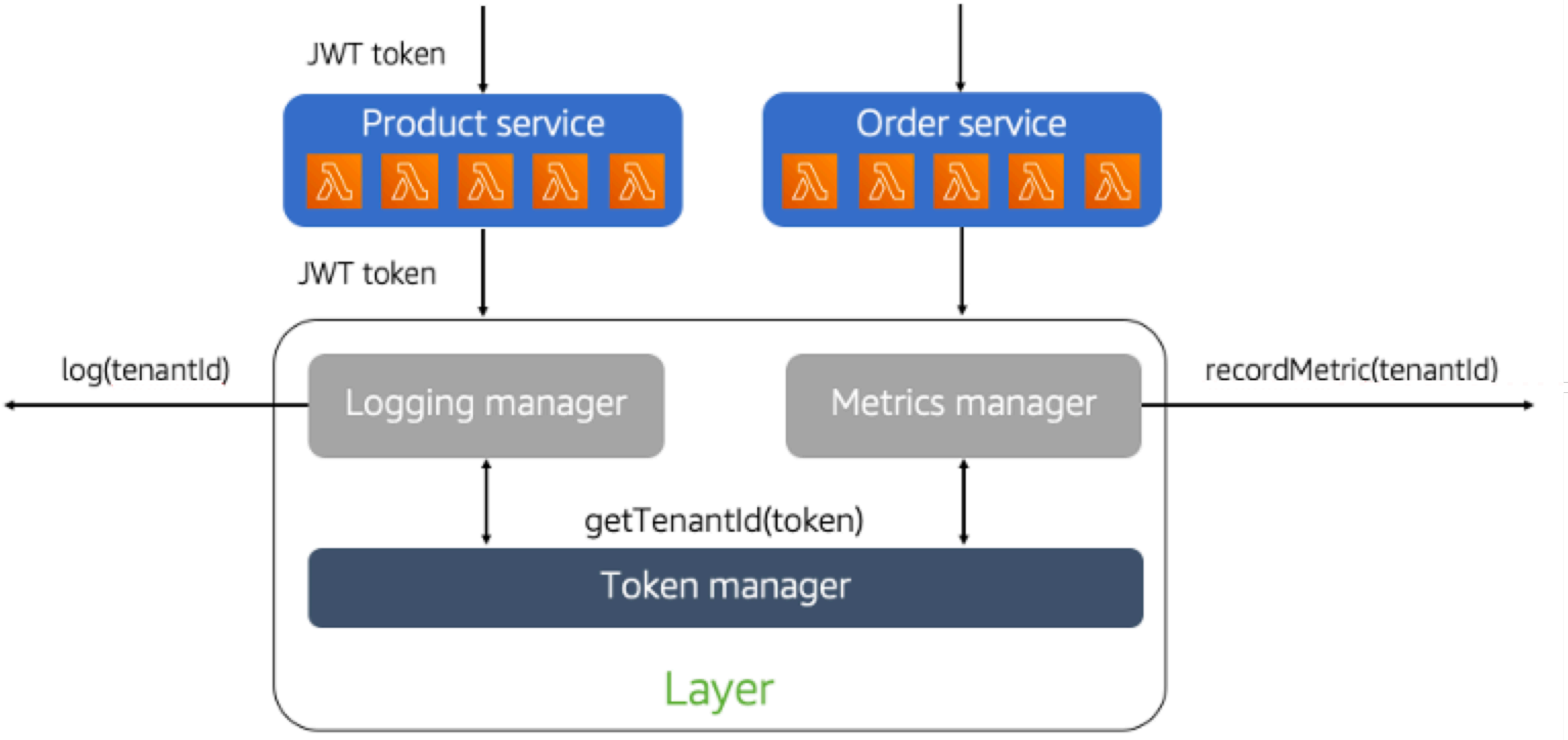
Twitter: rbhardwaj1

LinkedIn: www.linkedin.com/in/rohit-bhardwaj-cloud

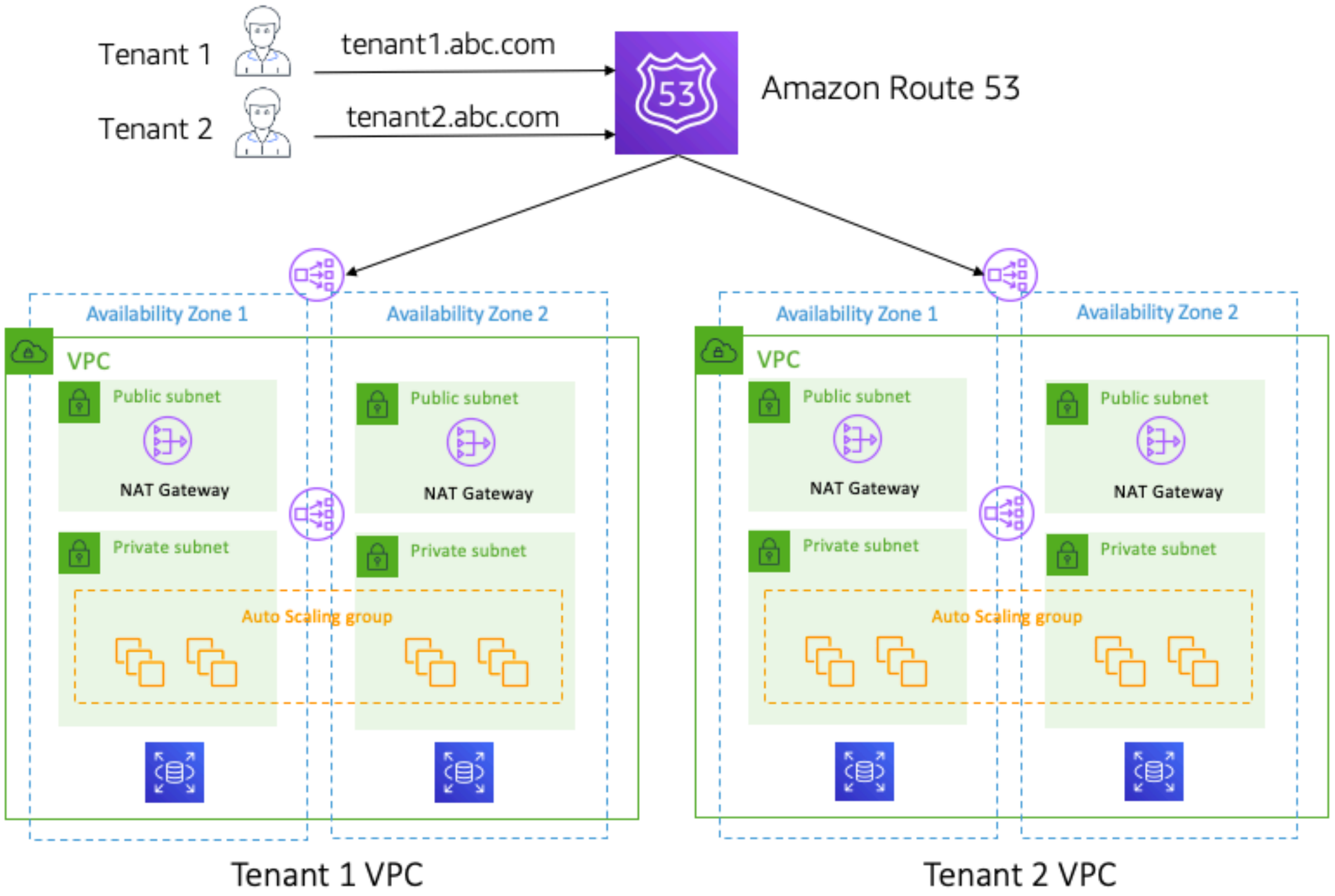
https://www.productivecloudinnovation.com/lessons

Access Management

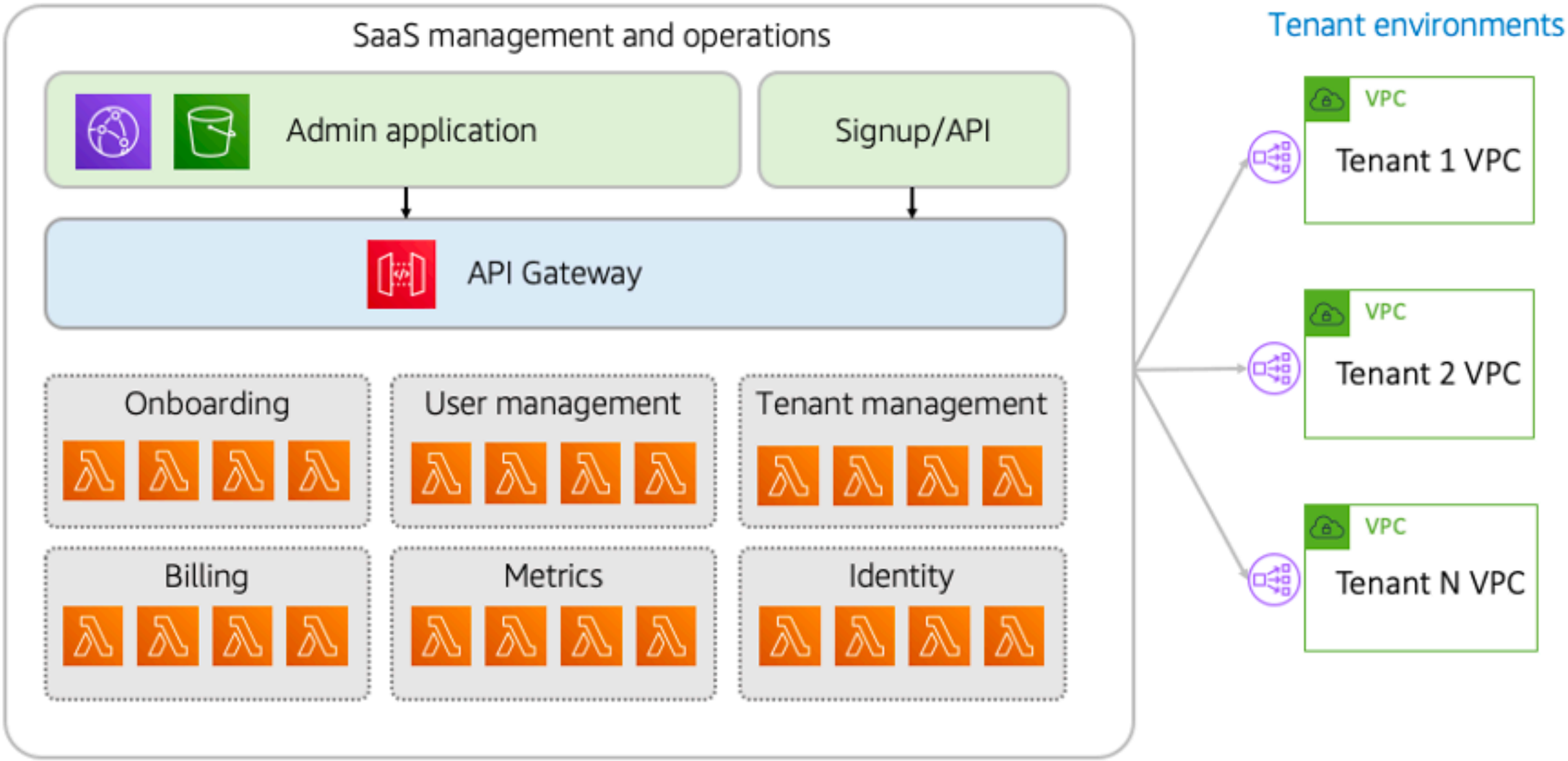



JWT token
Product service
Order service
JWT token
log(tenantId)
Logging manager
Metrics manager
recordMetric(tenantId)
getTenantId(token)
Token manager
Layer

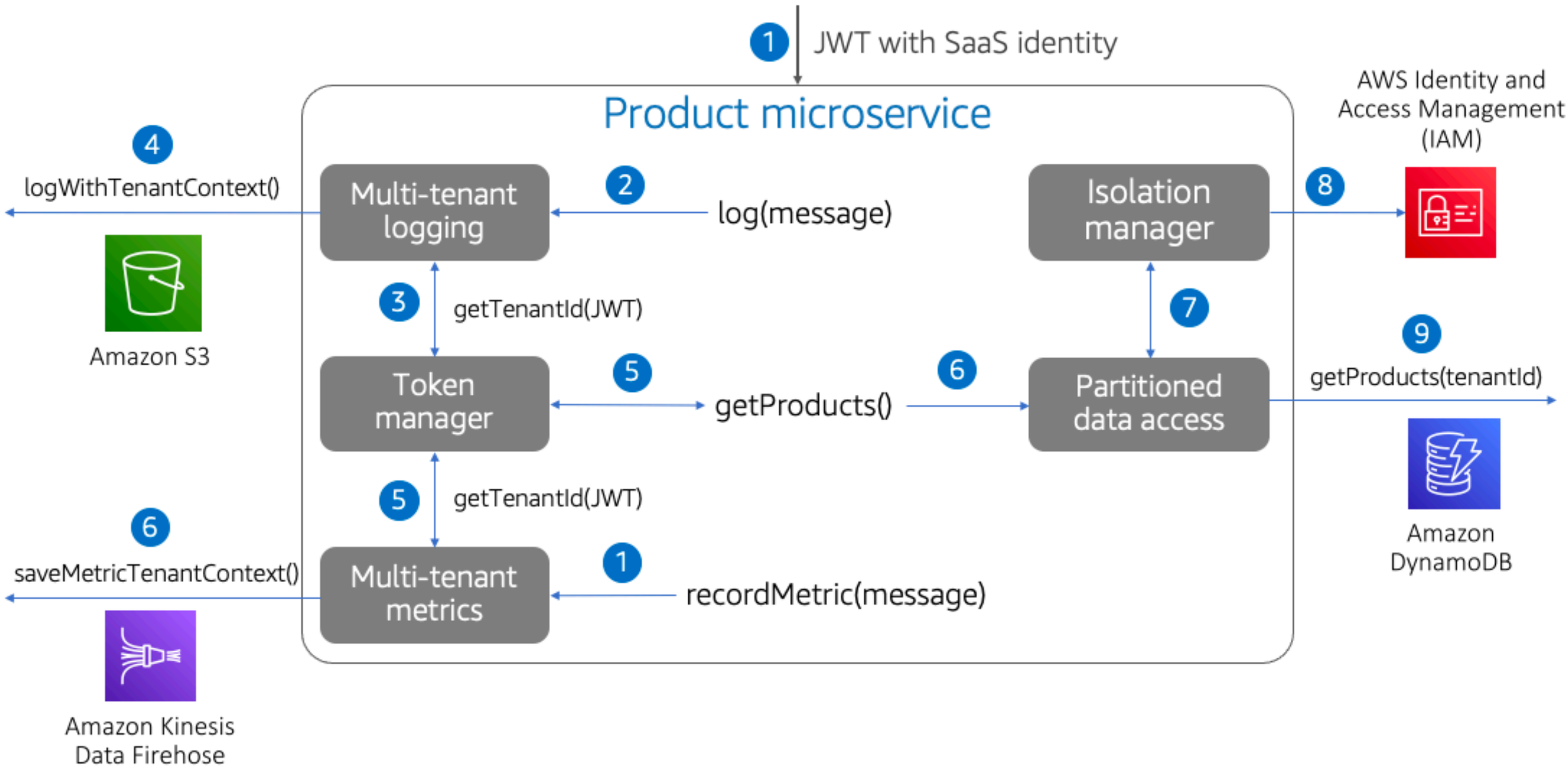
Multi tenant solution



Onboarding and Management



Multi-Tenant Microservices



https://docs.aws.amazon.com/wellarchitected/latest/saas-lens/tenant-insights.html

Dynamo DB Best Practices

- Uniform data access to prevent hot partitions - Read write loads
- DAX - Cache data
- Avoid capacity scaling up or down
- Attribute makes short
- Compress large non key attributes
- Use S3 and keep pointer in DB
- Avoid Scan operations
- Eventual consistency