AWS Lambda

CS516 – Cloud Computing

Computer Science Department

Maharishi International University

Maharishi International University - Fairfield, Iowa

Content

- Lambda and its features
- Lambda permissions
- Lambda sync and async triggers
- Environment variables
- Concurrency
- Version and Alias

Lambda

Lambda lets you run code without having to provision or manage servers (**serverless**).

You're only charged for requests and the compute time requests consume. There is **no charge** when your app is not used.

With Lambda, you can run code for virtually any type of application or backend service, all with **zero administration**. You just upload your code, and Lambda takes care of everything required to run and scale it with high availability.

Serverless computing, allowing code to be run without setup and maintenance of actual servers.

AWS Lambda stores code in Amazon S3 and encrypts it at rest.

Lambda benefits

No servers to manage (zero administration) - Just write the code and upload it to Lambda either as a ZIP file or *container image*.

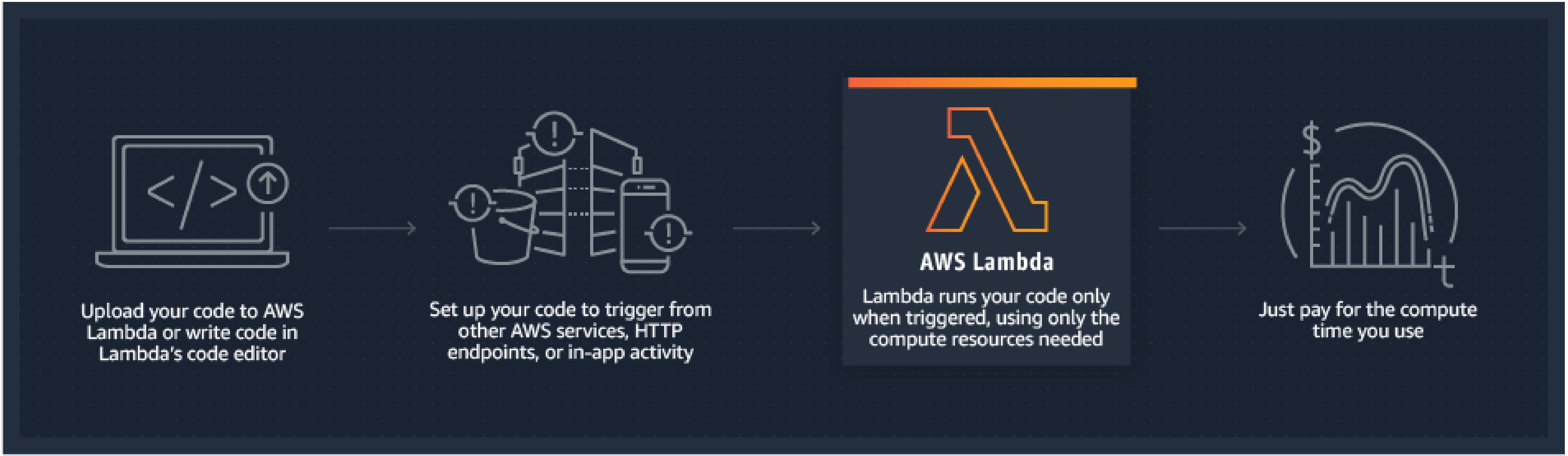
Continuous scaling - Your code runs in parallel and processes each trigger individually, scaling precisely with the size of the workload, from a few requests per day, to hundreds of thousands per second.

Cost optimized - You are charged for every millisecond your code executes and the number of times your code is triggered. You won't pay if it is not used. With Compute Savings Plan, you can additionally save up to 17%.

Consistent performance at any scale - With AWS Lambda, you can optimize your code execution time by choosing the right memory size for your function. You can also keep your functions initialized and hyper-ready to respond within double digit milliseconds by enabling Provisioned Concurrency.

How Lambda works

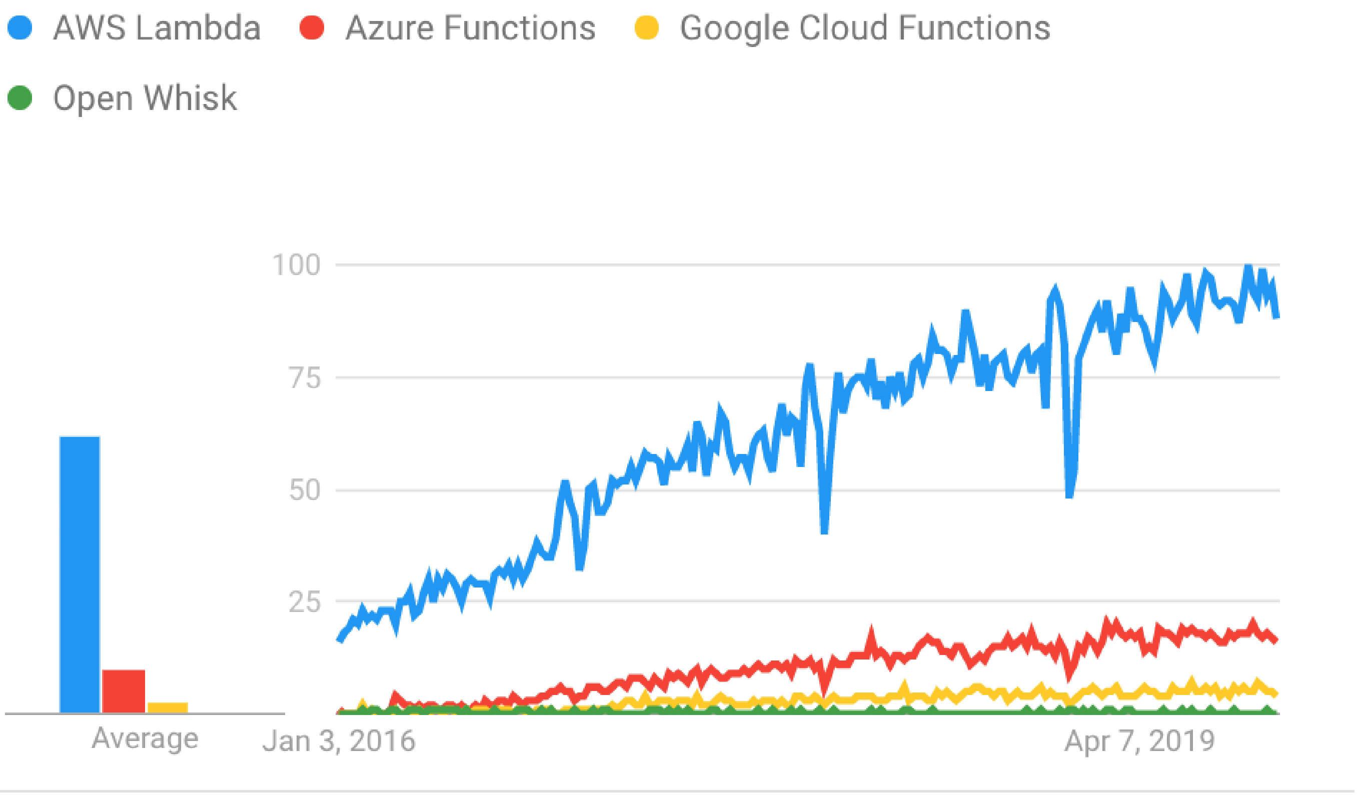
The AWS Serverless Application Model (**SAM**) is an open-source framework for building serverless applications.



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
$
t

Serverless Functions

- Event driven (File uploads, Scheduled times, API requests)
- Code focused
- Managed machines
- Cost effective
- Service integration
- Scaling



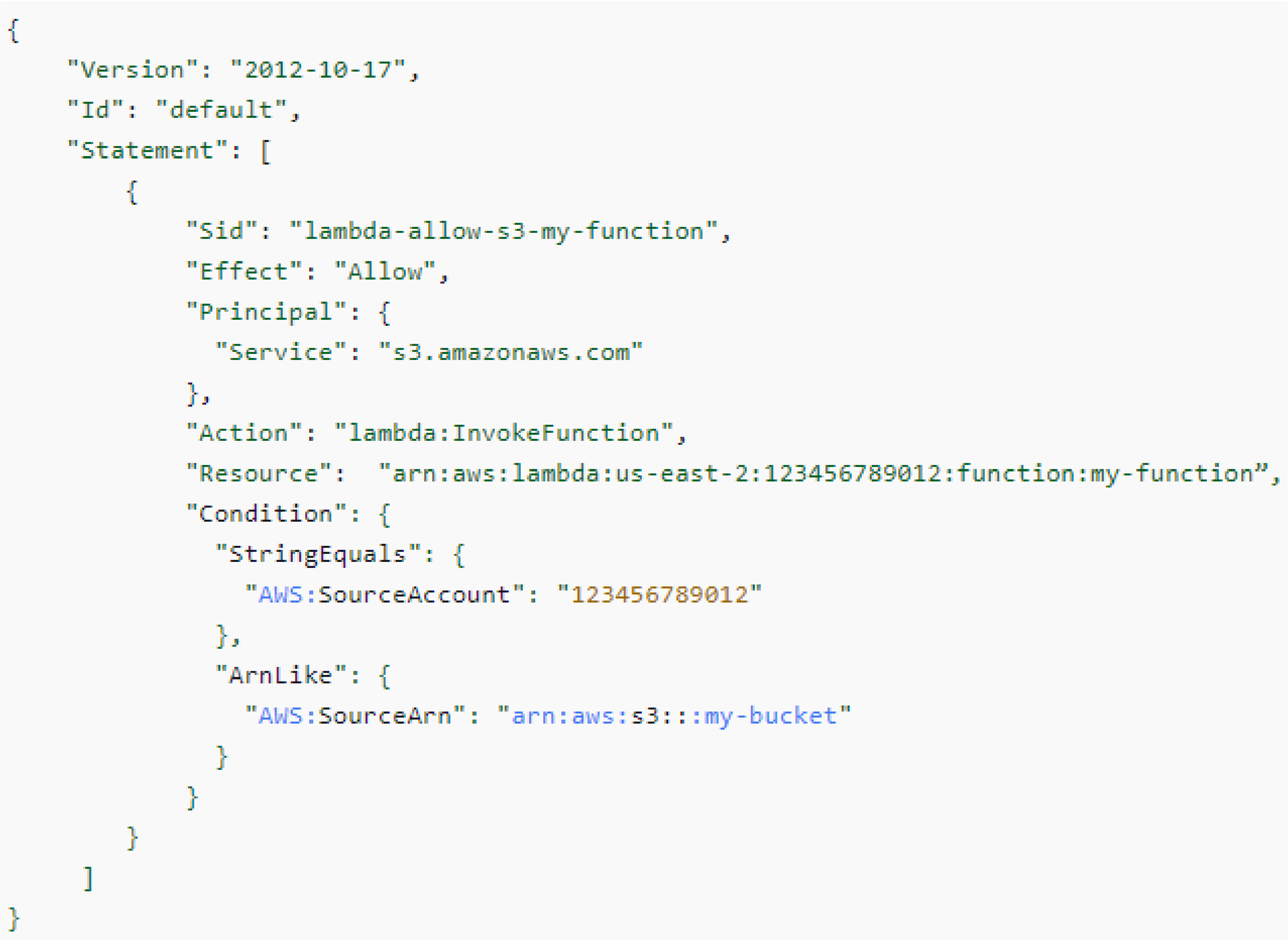
Lambda permissions

A Lambda function's **execution role** is an IAM role that grants the function permission to access AWS services and resources.

You provide this role when you create a function, and Lambda **assumes** the role when your function is invoked. For example, Amazon CloudWatch for logs, and DynamoDB for storing data.

A **resource-based** permission policy to allow an AWS service to invoke your function on your behalf.

Example of Resource-based policy



```
{
    "Version": "2012-10-17",
    "Id": "default",
    "Statement": [
        {
            "Sid": "lambda-allow-s3-my-function",
            "Effect": "Allow",
            "Principal": {
              "Service": "s3.amazonaws.com"
            },
            "Action": "lambda:InvokeFunction",
            "Resource":  "arn:aws:lambda:us-east-2:123456789012:function:my-function”,
            "Condition": {
              "StringEquals": {
                "AWS:SourceAccount": "123456789012"
              },
              "ArnLike": {
                "AWS:SourceArn": "arn:aws:s3:::my-bucket"
              }
            }
        }
     ]
}
```

Events can trigger lambda

AWS Lambda integrates with other AWS services to invoke functions.

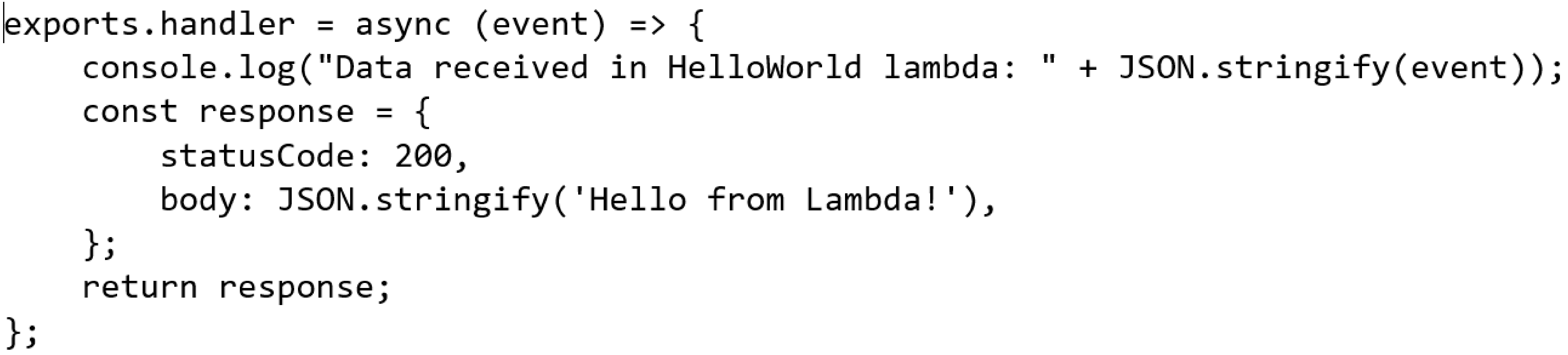
You can configure:

- triggers to invoke a function for other resources’ lifecycle events
- respond to incoming HTTP requests
- consume events from other resources such a queue
- run on a schedule.

Each service that integrates with Lambda sends data to your function in **JSON** as an **event**.

Sample Lambda code in NodeJS

The **HelloLambda** NodeJS code below on the index.js file:



```
exports.handler = async (event) => {
    console.log("Data received in HelloWorld lambda: " + JSON.stringify(event));
    const response = {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
    };
    return response;
};
```

Services that invoke Lambda functions synchronously

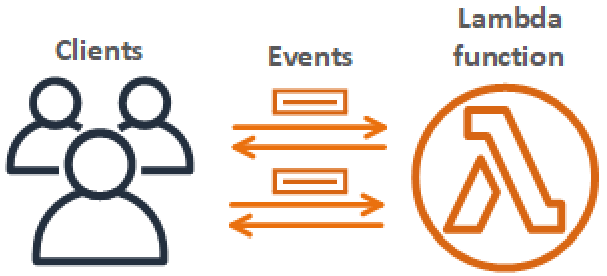
Other services invoke your function **directly**.

- You grant the other service permission in the function's resource-based policy
- Configure the other service to generate events and invoke your function.

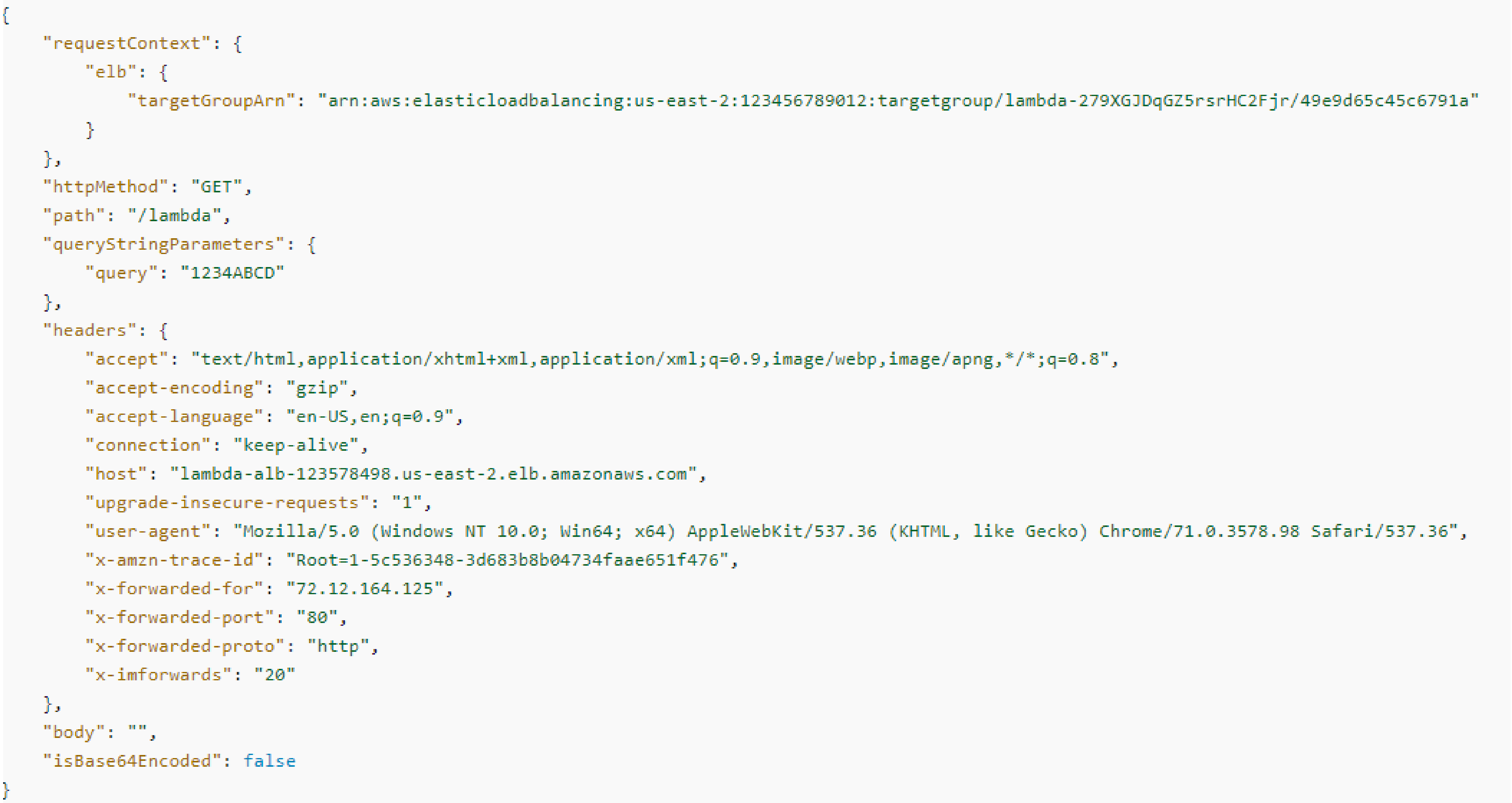
Depending on the service, the invocation can be synchronous or asynchronous. For synchronous invocation, the other service waits for the response from your function and **might retry** on errors.

- Elastic Load Balancing (Application Load Balancer)
- Amazon Cognito
- Amazon API Gateway
- Amazon CloudFront (Lambda@Edge)
- And more.

Synchronous Invocation

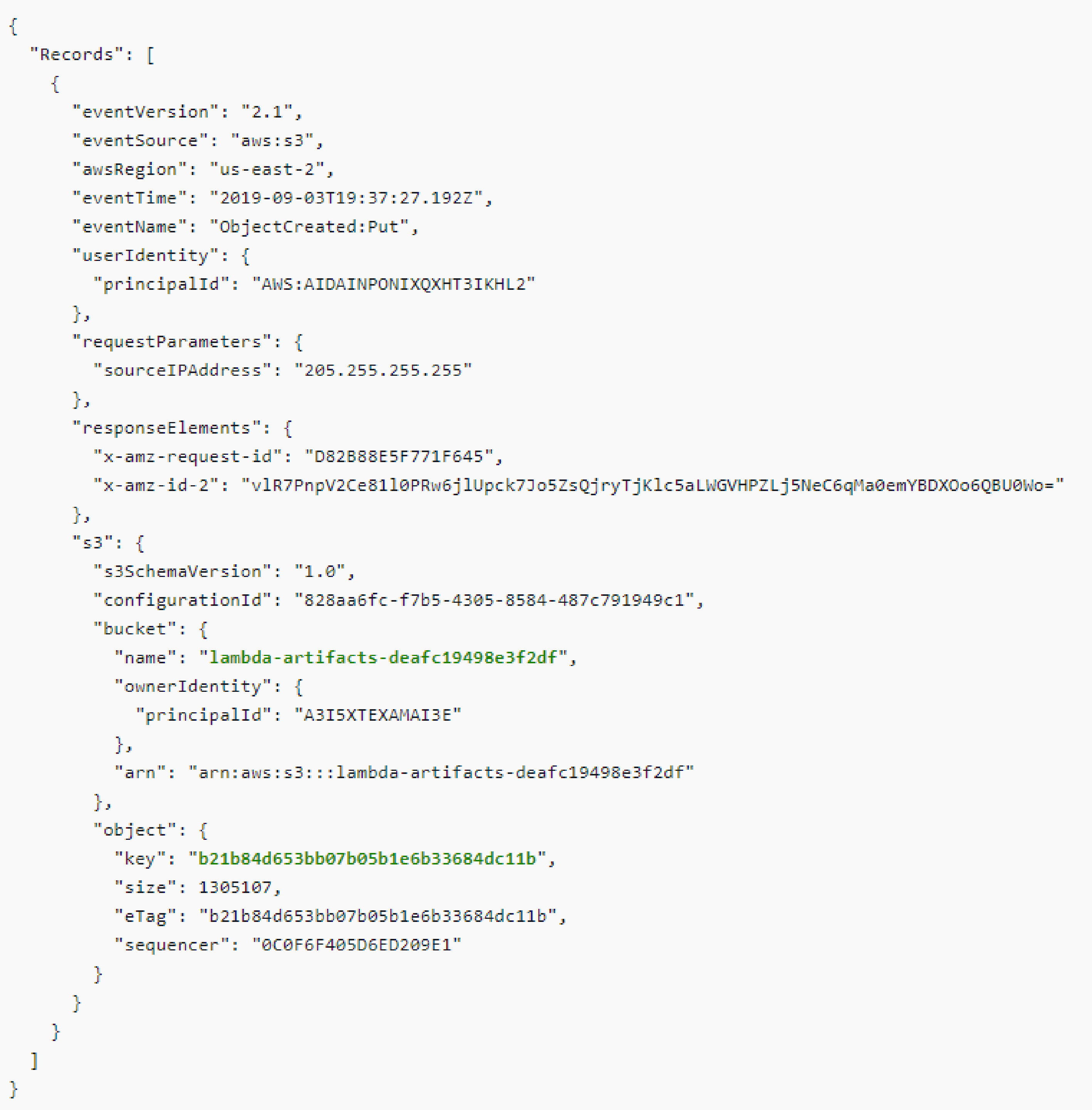


Example event from an ALB



```
{
    "requestContext": {
        "elb": {
            "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-2:123456789012:targetgroup/lambda-279XGJDqGZ5rsrHC2Fjr/49e9d65c45c6791a"
        }
    },
    "httpMethod": "GET",
    "path": "/lambda",
    "queryStringParameters": {
        "query": "1234ABCD"
    },
    "headers": {
        "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8",
        "accept-encoding": "gzip",
        "accept-language": "en-US,en;q=0.9",
        "connection": "keep-alive",
        "host": "lambda-alb-123578498.us-east-2.elb.amazonaws.com",
        "upgrade-insecure-requests": "1",
        "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36",
        "x-amzn-trace-id": "Root=1-5c536348-3d683b8b04734faae651f476",
        "x-forwarded-for": "72.12.164.125",
        "x-forwarded-port": "80",
        "x-forwarded-proto": "http",
        "x-imforwards": "20"
    },
    "body": "",
    "isBase64Encoded": false
}
```

Example event from S3



```
{
  "Records": [
    {
      "eventVersion": "2.1",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-2",
      "eventTime": "2019-09-03T19:37:27.192Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "AWS:AIDAINPONIXQXHT3IKHL2"
      },
      "requestParameters": {
        "sourceIPAddress": "205.255.255.255"
      },
      "responseElements": {
        "x-amz-request-id": "D82B88E5F771F645",
        "x-amz-id-2": "vlR7PnpV2Ce81l0PRw6jlUpck7Jo5ZsQjryTjKlc5aLWGVHPZLj5NeC6qMa0emYBDXOo6QBU0Wo="
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "828aa6fc-f7b5-4305-8584-487c791949c1",
        "bucket": {
          "name": "lambda-artifacts-deafc19498e3f2df",
          "ownerIdentity": {
            "principalId": "A3I5XTEXAMAI3E"
          },
          "arn": "arn:aws:s3:::lambda-artifacts-deafc19498e3f2df"
        },
        "object": {
          "key": "b21b84d653bb07b05b1e6b33684dc11b",
          "size": 1305107,
          "eTag": "b21b84d653bb07b05b1e6b33684dc11b",
          "sequencer": "0C0F6F405D6ED209E1"
        }
      }
    }
  ]
}
```

Services that invoke Lambda functions asynchronously

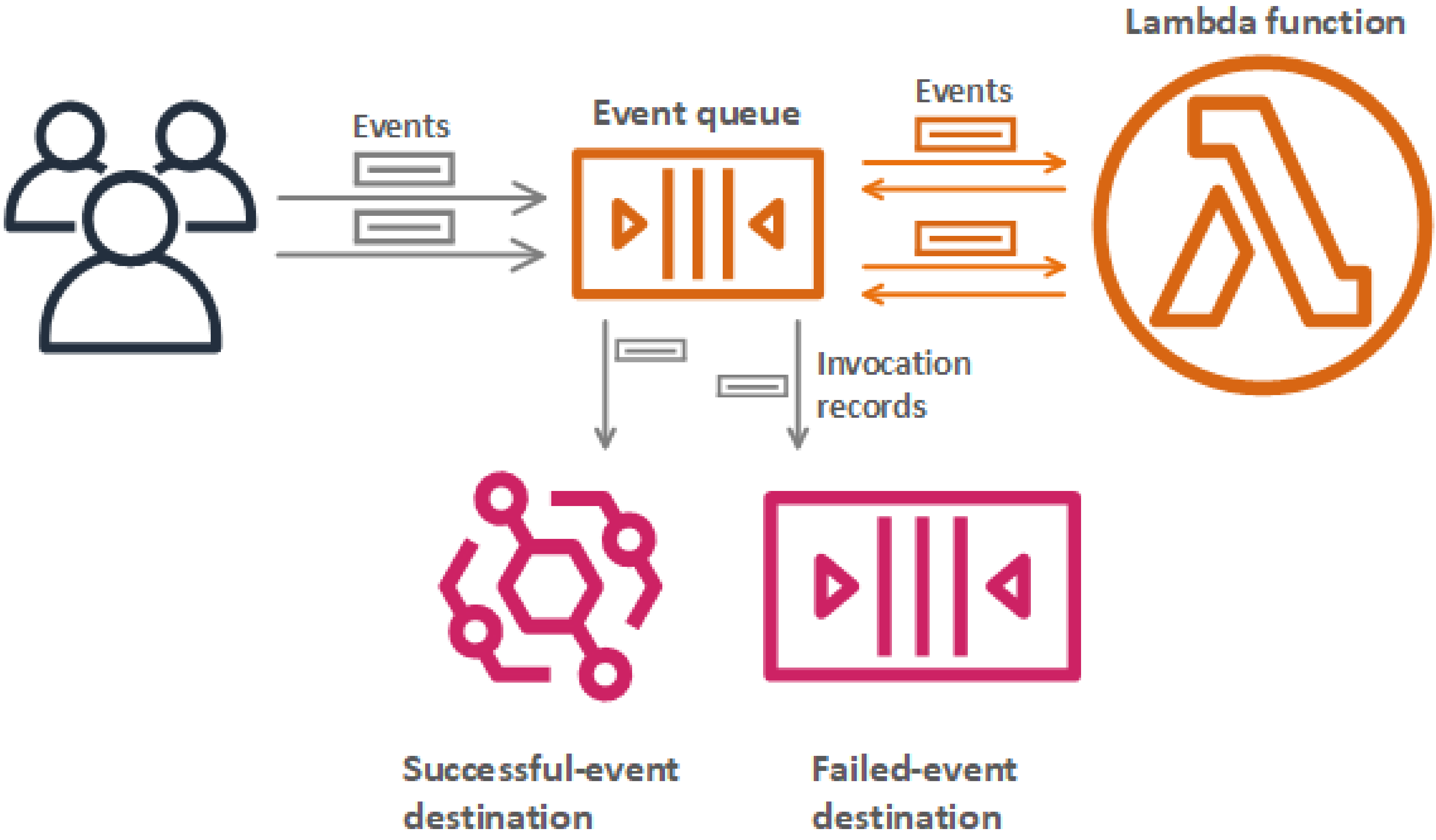
For asynchronous invocation, Lambda **queues** the event before passing it to your function.

The other service gets a success response as soon as the event is queued and isn't aware of what happens afterwards.

If an error occurs, Lambda handles retries, and can send failed events to a destination that you configure.

- S3
- SNS
- SQS
- and more

Destinations for Asynchronous Invocation



Lambda environment variables

You can use environment variables to adjust your function's behavior without updating code.

An environment variable is a pair of strings that are stored in a function's version-specific configuration.

Key	Value
THIRD_PARTY_API_URL	api.thirdparty.com/v1

```
const thirdPartyApiUrl = process.env.THIRD_PARTY_API_URL;
```

You don’t need to release the app again for changing a value if there is a change. Instead, just change to environment variable at run time.

The Lambda runtime makes environment variables available to your code.

Concurrency

Concurrency is the number of requests that your function is serving at any given time. When the function code finishes running, it can handle another request.

If the function is invoked again while a request is still being processed, another instance is allocated, which increases the function's concurrency.

Concurrency is subject to a Regional quota that is **shared by all functions in a Region**. Default quota for concurrent executions is **1000**. Can be increased up to hundreds of thousands.

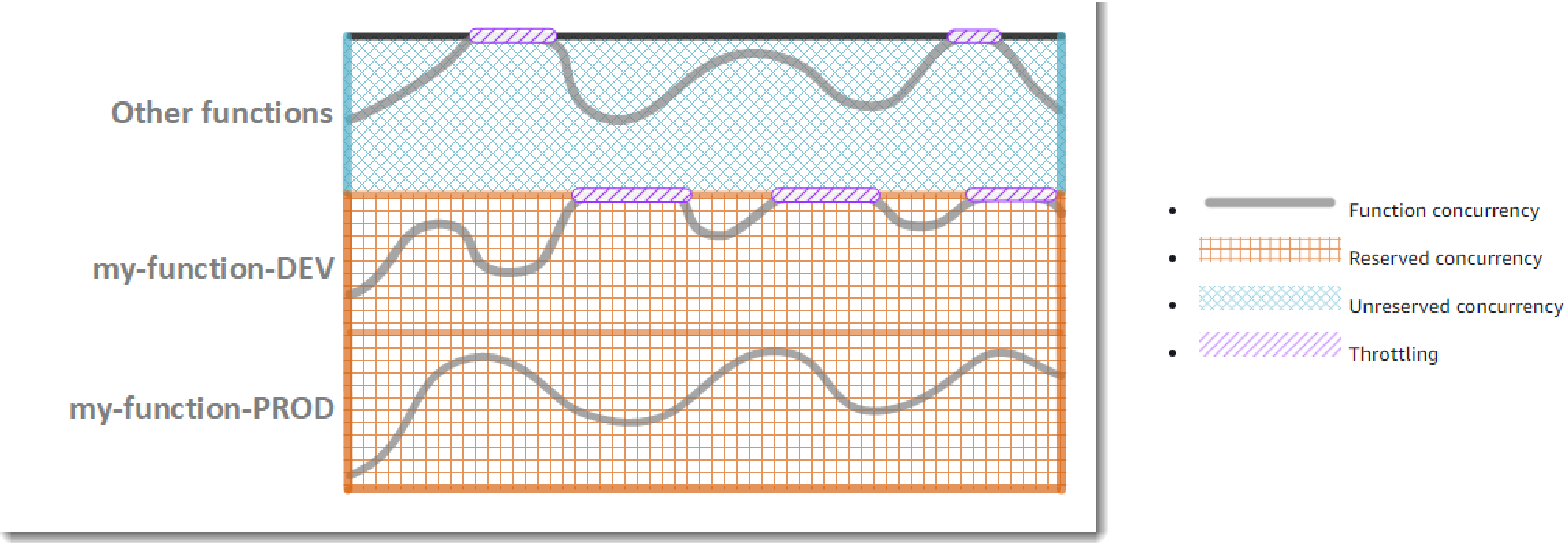
See more: Lambda quotes

Concurrency controls

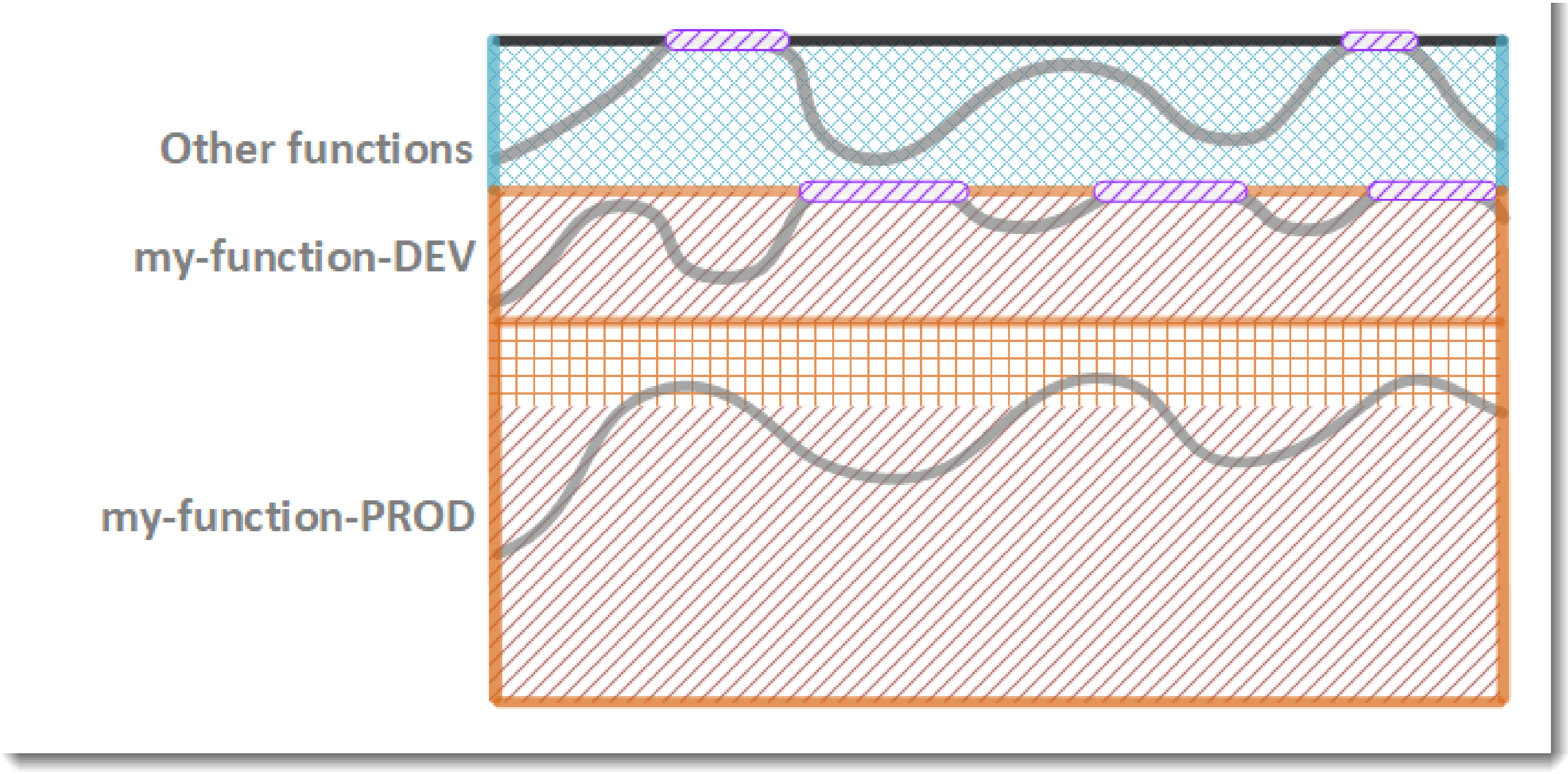
There are two types of concurrency controls available:

1. **Reserved concurrency** – Reserved concurrency guarantees the maximum number of concurrent instances for the function. When a function has reserved concurrency, no other function can use that concurrency.
2. **Provisioned concurrency** – Provisioned concurrency initializes a requested number of execution environments so that they are prepared to respond **immediately** to your function's invocations. That solves **cold-start** problem. Note that configuring provisioned concurrency incurs charges to your AWS account.

Reserved concurrency



Provisioned Concurrency with Reserved Concurrency



- Function concurrency
- Reserved concurrency
- Provisioned concurrency
- Unreserved concurrency
- Throttling

Versions

You can use versions to manage the deployment of your functions. For example, you can publish a new version of a function for beta testing without affecting users of the stable production version. Lambda creates a new version of your function each time that you publish the function.

There is a unique Amazon Resource Name (ARN) to identify the **specific version** of the function.

You **cannot edit** the code once you published the new version.

Version number is numeric increments such as 1, 2, 3, and so on.

Function – arn:aws:lambda:*us-west-2*:*123456789012*:function:*my-function*

Function alias – arn:aws:lambda:*us-west-2*:*123456789012*:function:*my-function*:*TEST*

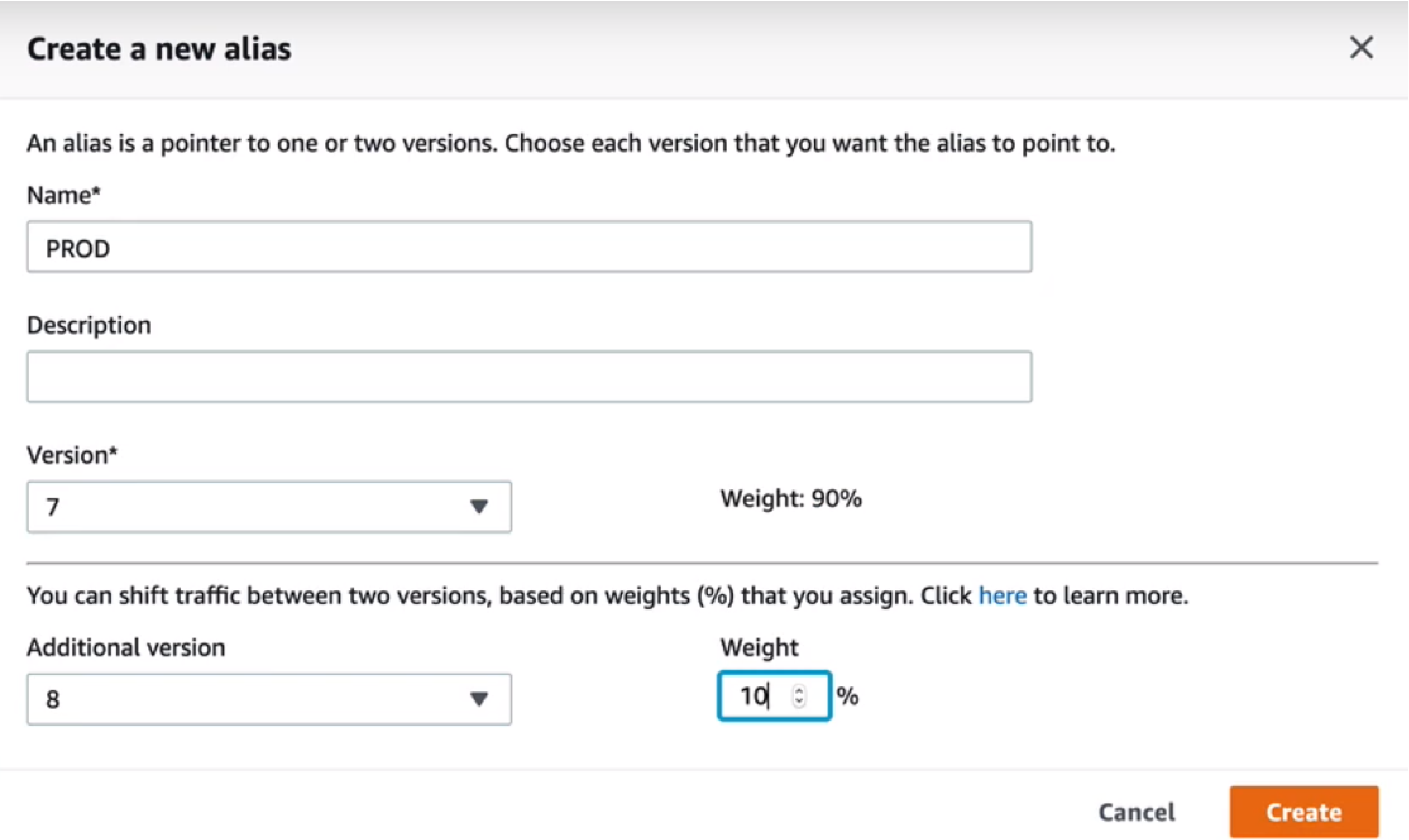
Function version – arn:aws:lambda:*us-west-2*:*123456789012*:function:*my-function*:*1*

Aliases

You can create one or more aliases for your Lambda function. A Lambda alias is like a pointer to a specific function version. You can give any name to the alias and refer to it such as prod, test, dev.

There are 2 benefits of using aliases instead of version in production:

1. You can switch versions back and forth.
2. You can implement canary deployment with it. A **canary deployment** is a deployment strategy that releases an app’s new version incrementally to a subset of users. All infrastructure in a target environment is updated in small phases (e.g: 2%, 25%, 75%, 100%).



Create a new alias

An alias is a pointer to one or two versions. Choose each version that you want the alias to point to.

Name*

PROD

Description

Version*

7

Weight: 90%

You can shift traffic between two versions, based on weights (%) that you assign. Click here to learn more.

Additional version

8

Weight

10 %

File storage for AWS Lambda

AWS Lambda includes a 512-MB temporary file system for your code, this is not intended for durable storage.

Amazon EFS is a fully managed, elastic, shared file system designed to be consumed by other AWS services, such as Lambda. You can easily share data across function invocations.

You can also read large reference data files, and write function output to a persistent and shared store.

Read more: Using Amazon EFS for AWS Lambda

Database proxies

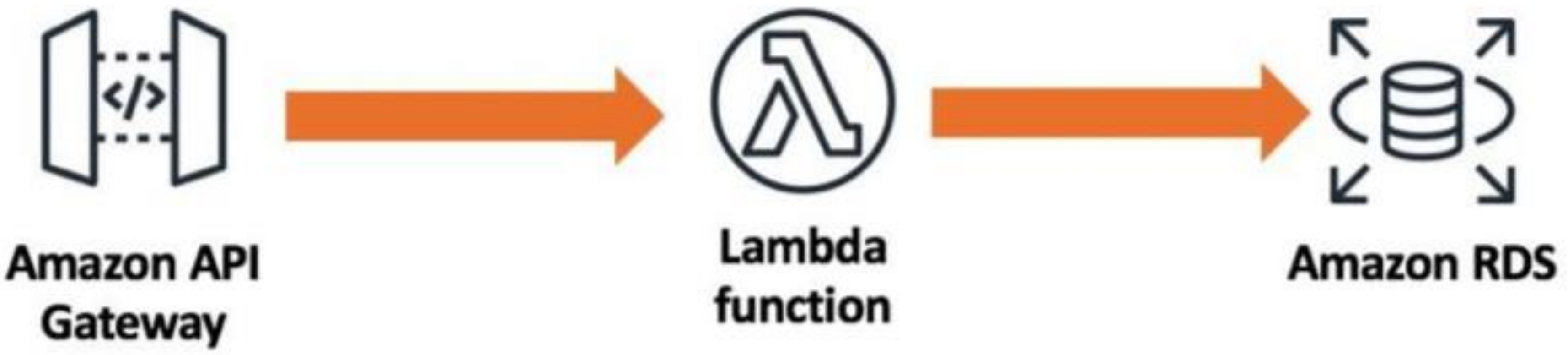
Often developers must access data stored in relational databases. You can connect to relational databases (AWS RDS) from Lambda functions.

But it can be challenging to ensure that your Lambda invocations do not overload your database with too many connections. This is because each connection consumes memory and CPU resources on the database server.

Read more: [Using Amazon RDS Proxy with AWS Lambda]()

Database proxies

The number of maximum concurrent connections for a relational database depends on how it is sized. Lambda functions can scale to tens of thousands of concurrent connections, meaning your database needs more resources to maintain connections instead of executing queries.



When should I use Lambda vs EC2

Use lambda when

- you don’t have many developers to manage servers
- you want faster development
- number of transactions are **unpredictable**
- the max number of transactions are hundreds or thousands at a second

Use EC2 when

- you need to manage the underlying
- number of transactions are predictable, relatively consistent over time.
- large number of transactions at a time