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**
- Event source mapping
- Async destinations
- Environment variables
- **Concurrency**
- **Version and Alias**
- Stateless and stateful

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 based on the memory size of the function. There is **no charge** when your app is not used.

You just upload your code and set the memory size (by default 128 MB). Then Lambda takes care of everything required to run and scale it with high availability. Timeout is 15 minutes. I think you shouldn’t use Lambda if it runs more than 30 secs. Use EC2 or ECS instead.

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*.
- **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%.
- **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.
- **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.

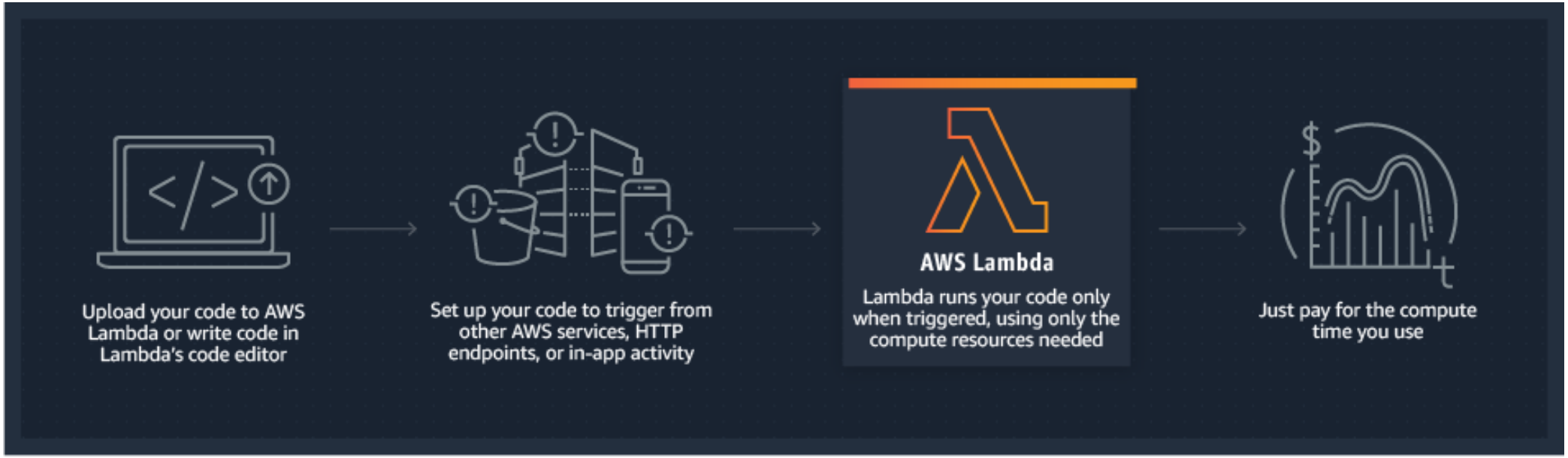
Other features

- **Orchestrate multiple functions** - You can coordinate multiple AWS Lambda functions (other services) for long-running and express tasks by building workflows with **AWS Step Functions**. Step Functions lets you define workflows that trigger a collection of Lambda functions using sequential, parallel, branching, and error-handling steps. With Step Functions and Lambda, you can build stateful, long-running processes for applications and backends.
- **Flexible resource model** - You choose the amount of memory you want to allocate to your functions and AWS Lambda allocates proportional CPU power, network bandwidth, and disk I/O.

In serverless world, problems like multi-threading and collisions are implicitly solved. Lambda instances are isolated. In S3, there could be unlimited number of connections. To learn more about serverless, check out serverless land which has many best practices, design patterns, technology stacks.

How Lambda works

The AWS Serverless Application Model (**SAM**) is an open-source framework for building serverless applications. In real-life, you will use SAM to build your lambda functions. With SAM, you write an YAML template for the infrastructure (API Gateway, Lambda, DynamoDB) that you can integrate with CICD pipeline.



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 (user is created in Cognito).
- 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**. Hence, the event payload is vary depending on what triggers the lambda.

Example event from an Application Load Balancer


```
{
    "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
}
```

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.

Services that invoke Lambda functions asynchronously

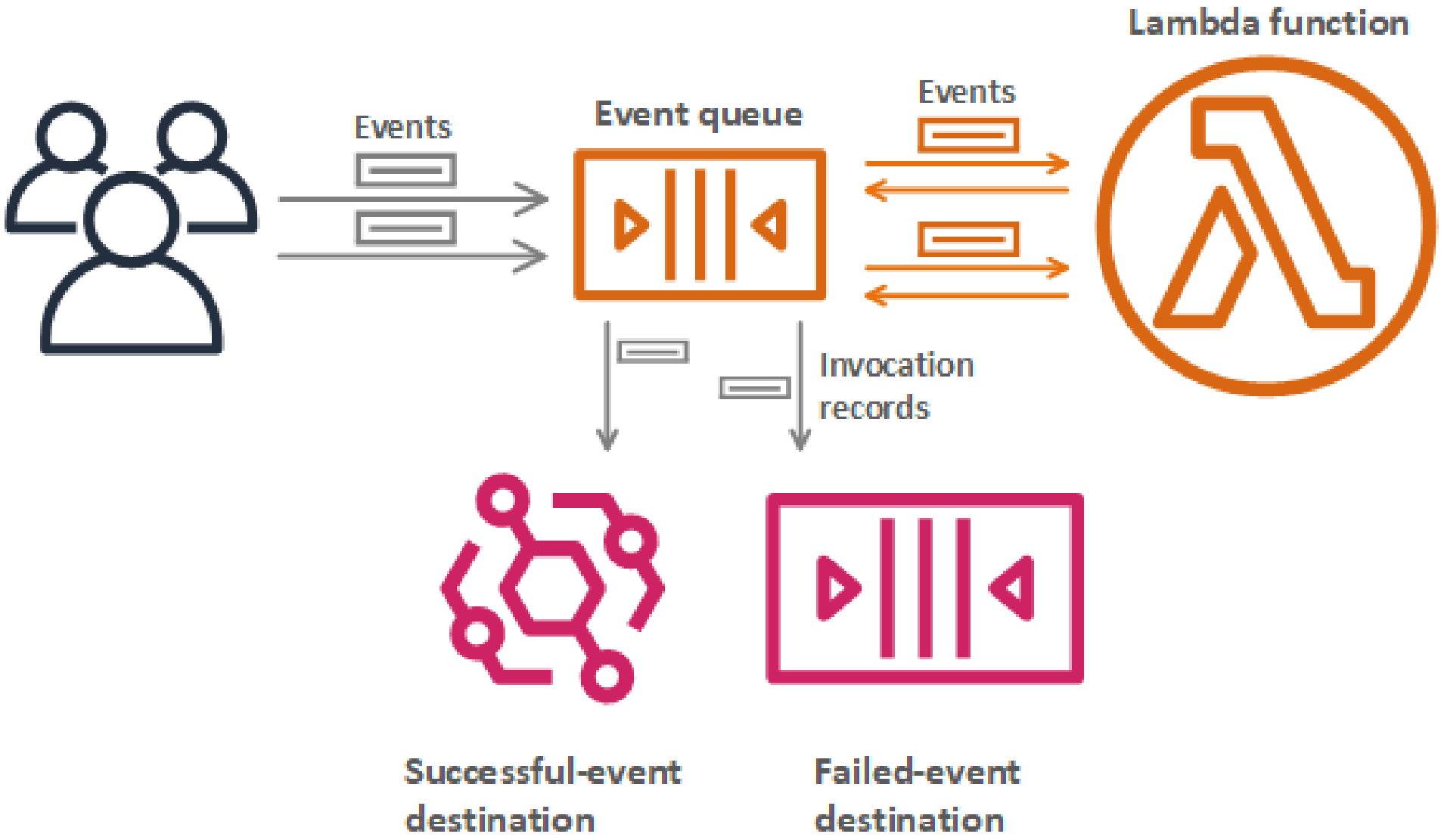
For asynchronous invocation, Lambda **queues** the event before passing it to your function.

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
- AWS CloudFormation
- Amazon CloudWatch
- SQS standard
- and more

Destinations for Asynchronous Invocation



Destinations for asynchronous invocation

You can configure Lambda to send an **invocation record** to **another service**. Lambda supports the following destinations for asynchronous invocation:

- Amazon SQS – A **standard** SQS queue.
- Amazon SNS – An SNS topic.
- AWS Lambda – A Lambda function.
- Amazon EventBridge – An EventBridge event bus.

The invocation record contains details about the request and response in JSON format. You can configure separate destinations for events that are processed **successfully**, and events that **fail** all processing attempts.

Alternatively, you can configure an SQS queue or SNS topic as a dead-letter queue for discarded events. For dead-letter queues, Lambda only sends the content of the event, without details about the response.

Services that Lambda reads events from

- Amazon DynamoDB
- Amazon Kinesis
- Amazon MQ
- Amazon Managed Streaming for Apache Kafka
- Amazon FIFO Simple Queue Service

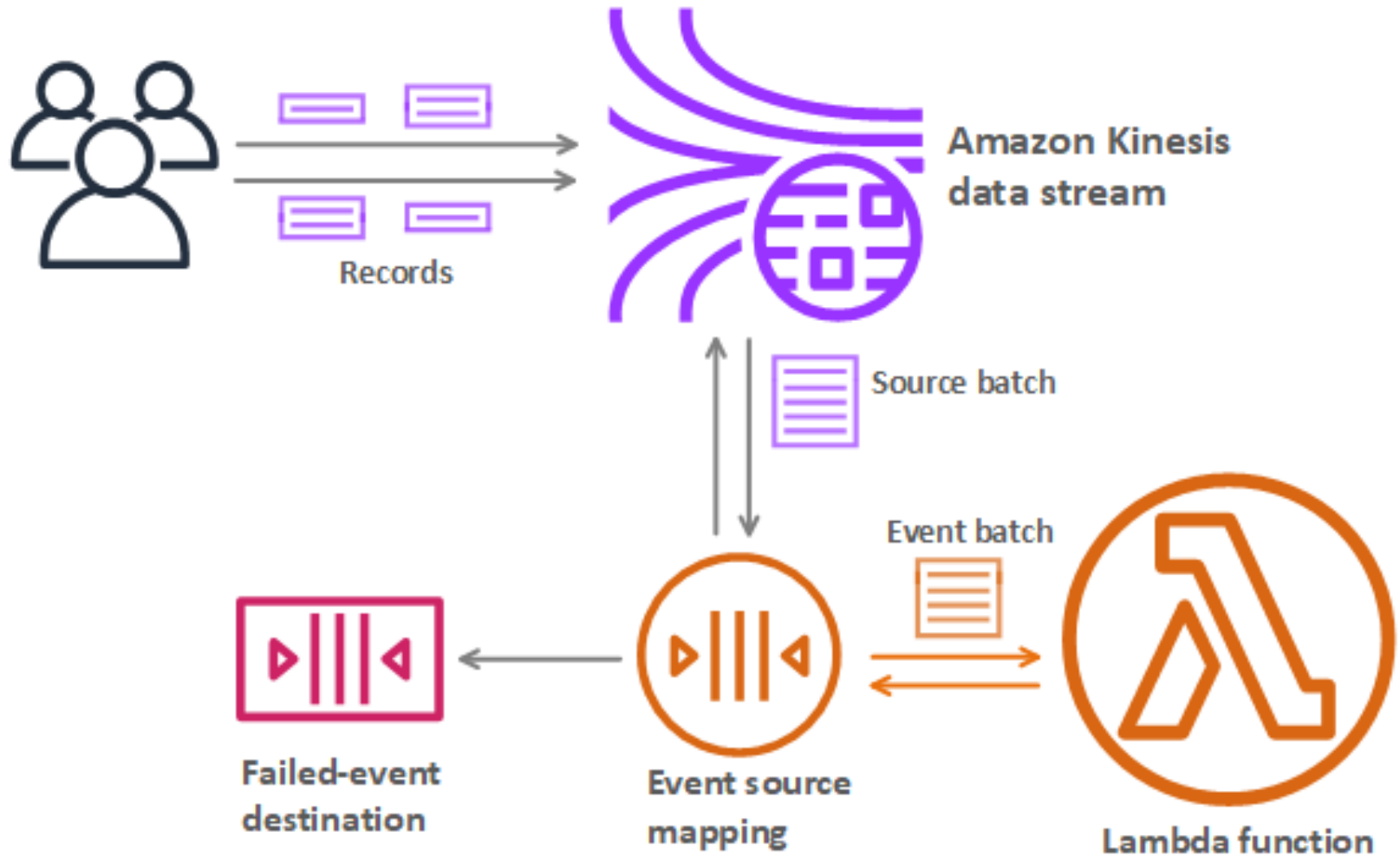
Event source mappings

An event source mapping is an AWS Lambda resource that reads from an event source and invokes a Lambda function. You can use event source mappings to process items from a stream or queue in **services that don't invoke Lambda functions directly**.

The example below is creating an event source mapping (under the hood, it is lambda) that reads events (json data) from DynamoDB that doesn’t directly invoke lambda. Now your my-function lambda can read data from DynamoDB with the help of event source mapping.

```
aws lambda create-event-source-mapping --function-name my-function --batch-size 500 --starting-position LATEST \
--event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table/stream/2019-06-10T19:26:16.525
```

Event Source Mapping with Kinesis Stream



Lambda environment variables

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

- Dynamic value that you can change at runtime. Lambda -> Third party API (api.example.com/v1 -> api.example.com/v2)
- Reading different values for the same environment variable key. For example, reading URLs for each env.

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

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.

Lambda stay warm for about 30 – 45 mins.

See more: Lambda quotes

One concurrent lambda can handle this

- Image source: AWS Lambda function and its valuable nuances

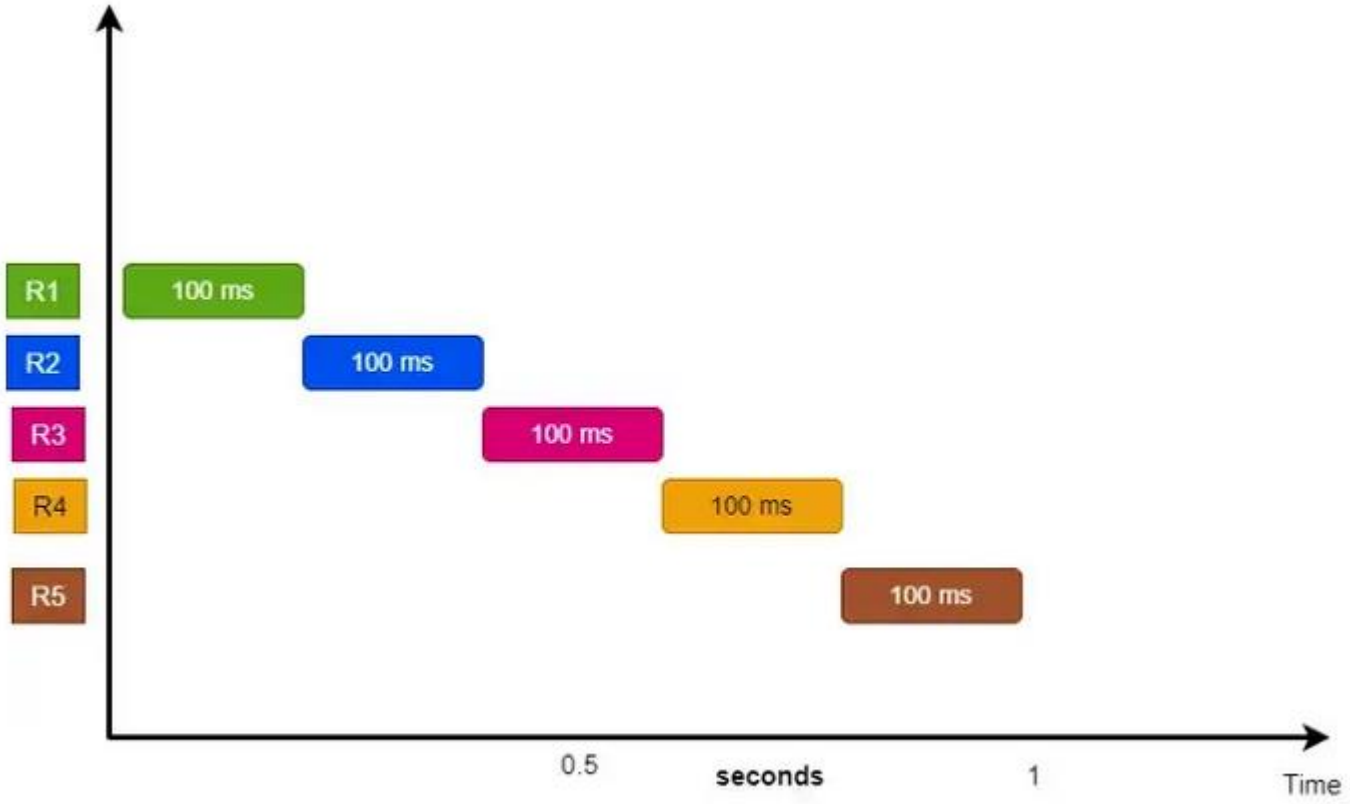


Fig 16: Example Lambda execution time

For this, you need more than one lambda instance for concurrency

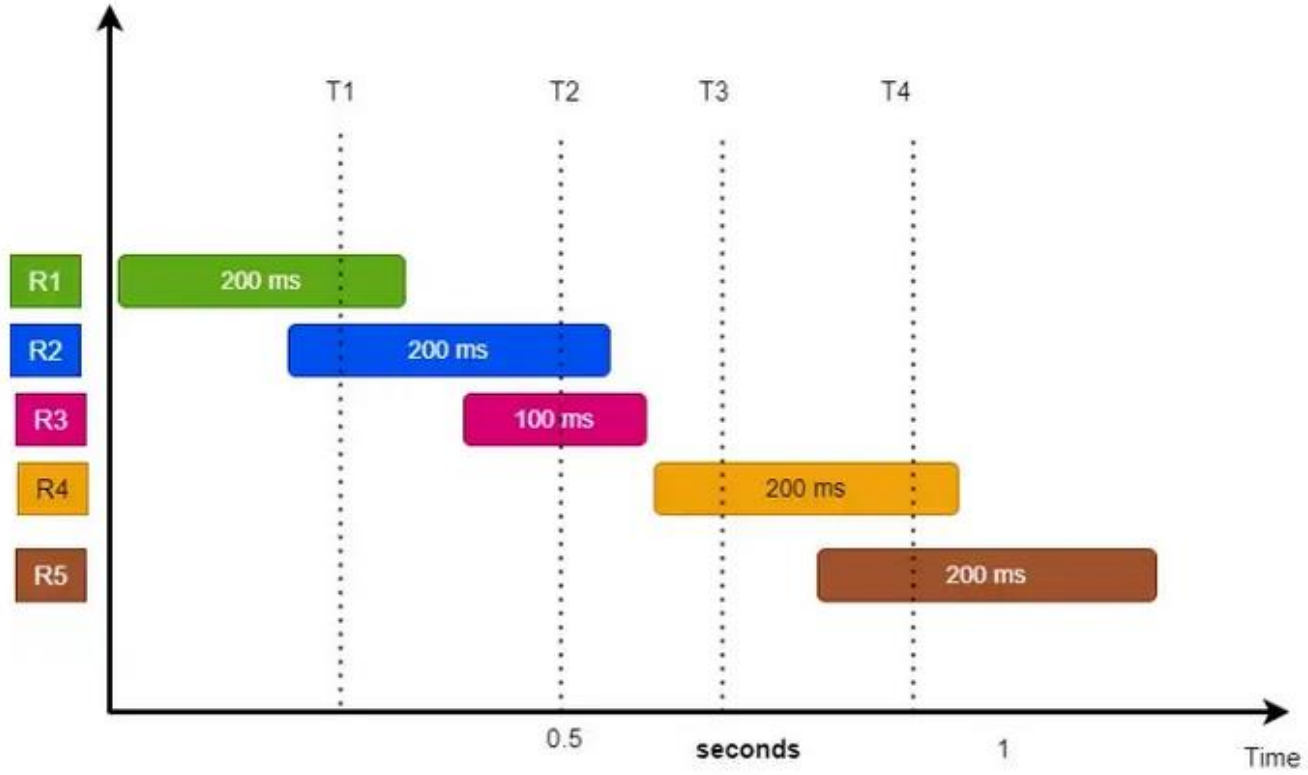


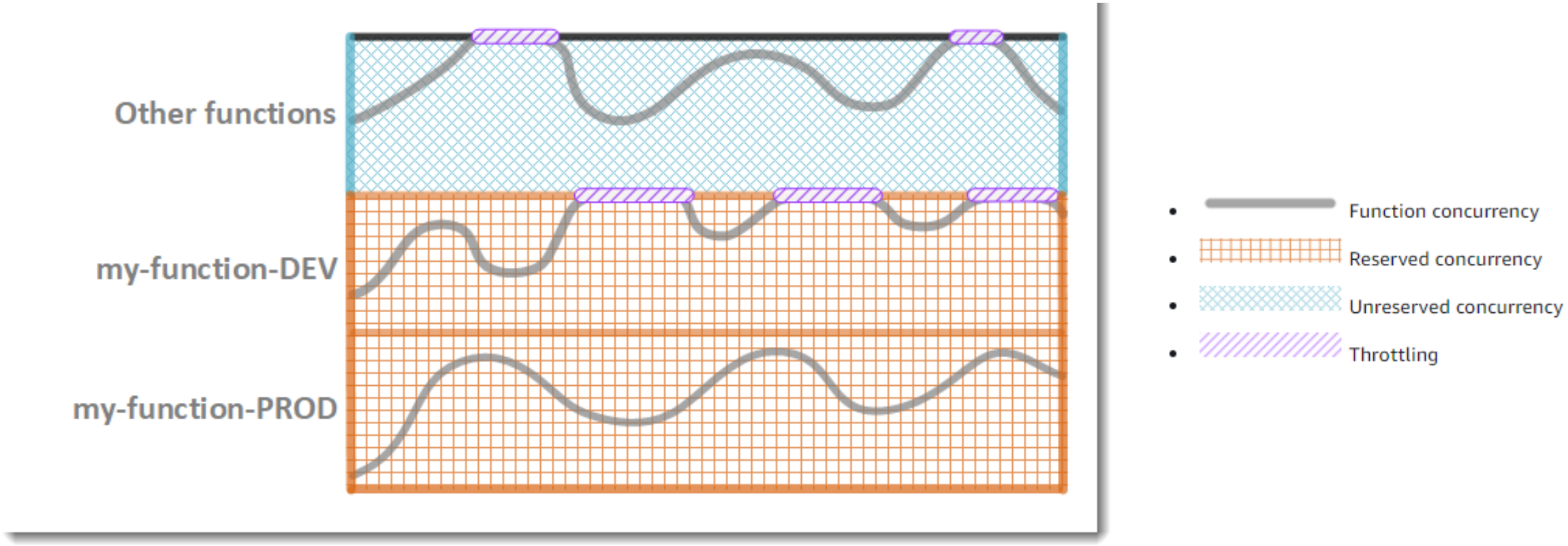
Fig 17: Example Lambda execution time

Concurrency controls

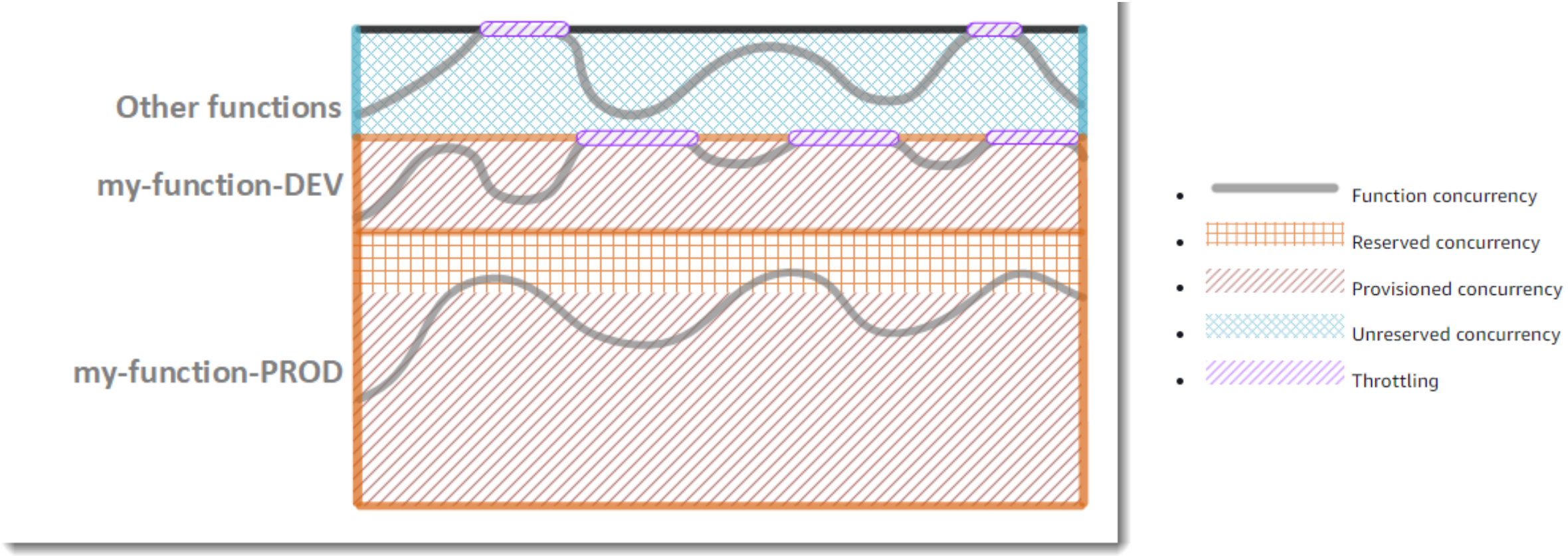
There are two types of concurrency controls available:

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 improves **cold start**. Note that configuring provisioned concurrency incurs charges to your AWS account.

Reserved concurrency



Provisioned Concurrency with Reserved Concurrency



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.

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.

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 %

Cancel

Create

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 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.

Pricing

- It charges for the number of invocations and how long it took to process the request. But the amount for the number of invocations is tiny. The main cost is how long it took and how big your lambda (memory) is.
- For example, provisioned lambda costs $0.000004167 for every GB-second. If the memory is 512 MB then, $0.000004167 * 0.5 , the memory is 2 GB then, $0.000004167 * 2 and, so on.
- The max memory as of Jan 2023 for the lambda is **10 GB**.
- How much will it cost for a lambda that has 512 MB memory and 10 million requests with a duration of 2 seconds each?
  - Invocation is **$2** = $0.20 * 10M and the compute cost is **$166**.
  - When **50** lambda instances are **provisioned**, the provisioning cost is **$279**, and the total cost is **$380.**

Lambda best practices

- Instantiate DB connection and AWS SDK outside of the function. So, it is shared across all functions.
- Import only required libraries to prevent cold start! For example, AWS SDK initialization will take about 300 ms, and the other library takes 200 ms. So, the total lambda instantiation takes about 500 ms. If you remove the other library that is not necessary, then the cold start time is decreased to 300 ms or 40% faster.
- Use it for transforming data, not transporting. When you make an API call, the function needs to wait that charges more. Use step functions when you need to wait.

What is stateful and stateless app?

Stateful applications and processes are those that can be returned to again and again, like online banking or email. They're performed with the **context of previous transactions** and the current transaction may be affected by what happened during previous transactions. For these reasons, stateful apps use **the same servers** each time they process a request from a user.

A **stateless** process or application can be understood in isolation. There is **no stored knowledge** of or reference to past transactions. Each transaction is made as if from scratch for the first time.

Keeping functions stateless enables AWS Lambda to rapidly launch as many copies of the function as needed to scale to the rate of incoming events. While AWS Lambda's programming model is stateless, your code can access stateful data by calling other web services, such as Amazon S3 or Amazon DynamoDB.

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 at a second
- Use EC2 when
  - you need to manage the underlying resources
  - number of transactions are predictable, relatively consistent over time.
  - large number of transactions at a time

Cost comparison

- More details on a Medium blog