Going serverless

A short, high-level but maybe useful introduction to AWS Lambda

Time for introductions

I’m Bruno! I’ve been a software developer for more than 10 years, with quite a bit of experience in different enterprises, big and small.

Currently I am the Domain Architect for the AML & CTF business unit @ Klarna.

Email: bruno.felix@klarna.com

Linkedin: https://www.linkedin.com/in/felixbruno/

Disclaimer: This presentation and its contents represent my personal experience and views and not those of my employer.

The history of computing is the history of ever increasing levels of abstraction.

The history of computing is the history of ever increasing levels of abstraction.

Serverless is no different.

The history of computing is the history of ever increasing levels of abstraction.

Serverless is no different.

But it’s still someone else’s computer in the end.

Sure, but what’s in it for me?

Key takeaways:

- I hope I can convince you this is a useful tool;
- The basics of AWS Lambdas;
- Use cases where AWS Lambda may be interesting;
- Limitations and things to look out for when using this technology.

Structure of this talk

- Intro and motivation;
- AWS Lambda;
  - Intro;
  - Lifecycle;
  - Security and permissions;
  - Dependencies;
  - Layers;
  - Deployment;
  - Monitoring;
  - Limits;
  - Costs;
  - Downsides;
  - Use cases.
- Parting words & QA.

AWS Lambda - intro

“AWS Lambda is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers” (source).

AWS Lambda - intro

“AWS Lambda is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers” (source).

Event-driven: something happens and that will trigger the execution of an arbitrary piece of code.

AWS Lambda - intro

“AWS Lambda is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers” (source)

Event-driven: something happens and that will trigger the execution of an arbitrary piece of code.

Without provisioning or managing servers: just focus on your application code (or not).

AWS Lambda - intro

What programming languages can I use with AWS Lambda?

Lambda supports multiple languages through the use of runtimes. A lot of common languages are supported out-of-the-box (Java, Python, Javascript, .Net, Ruby, Go: source). You can add your own custom runtimes for additional flexibility.

For now we will keep things simple and use Python.

DEMO-01

Your first contact with AWS Lambda (source)

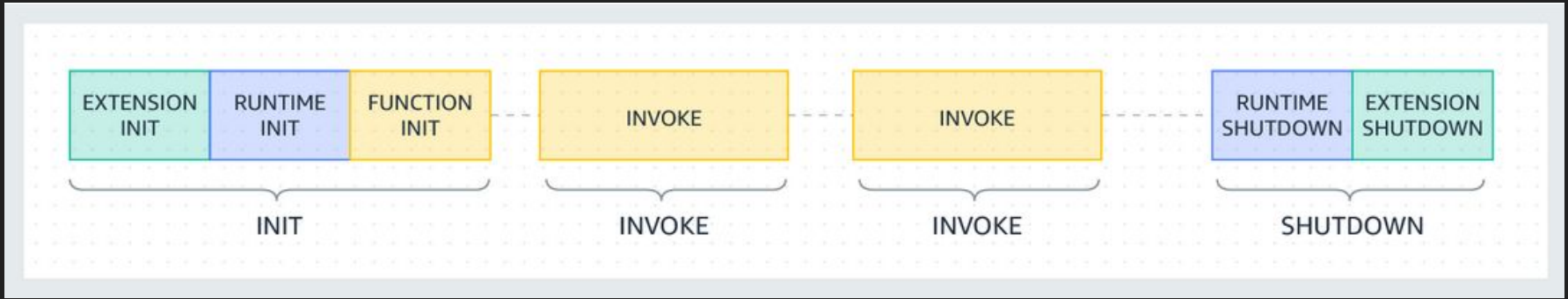
AWS Lambda - What have we learned so far?

Created a simple Python AWS Lambda function from the AWS console.

Used a test event to validate it works as expected.

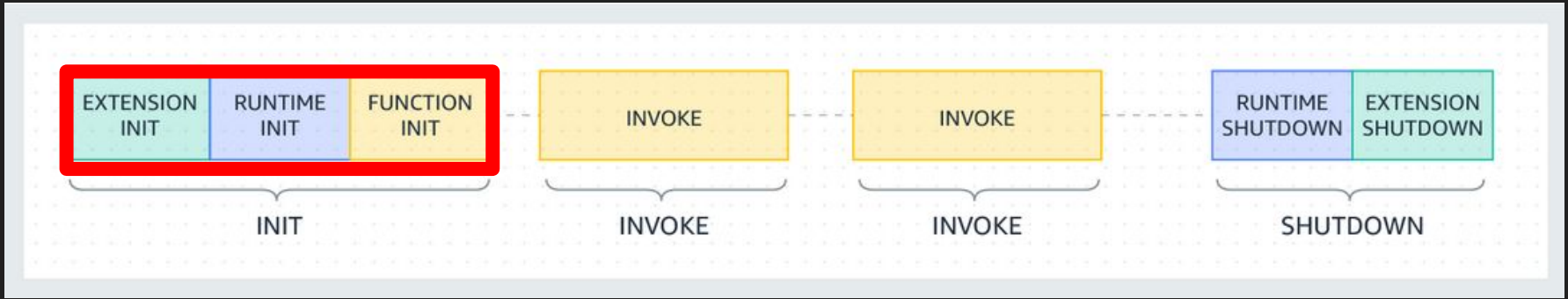
AWS Lambda - lifecycle

AWS Lambda executes your code in an isolated execution environment and provides the lifecycle support to init, invoke and shutdown your code and any resources. (source)



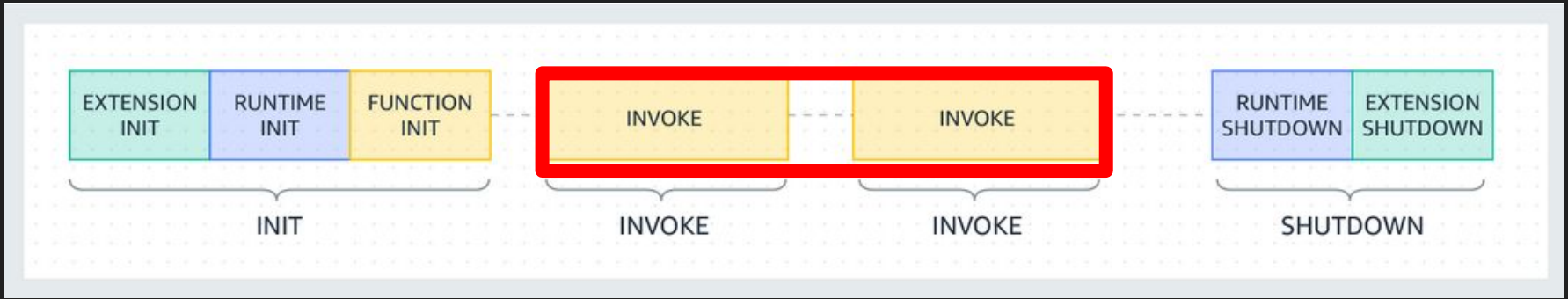
AWS Lambda - lifecycle

Init phase: start all extensions, bootstrap the runtime, run the function's static code



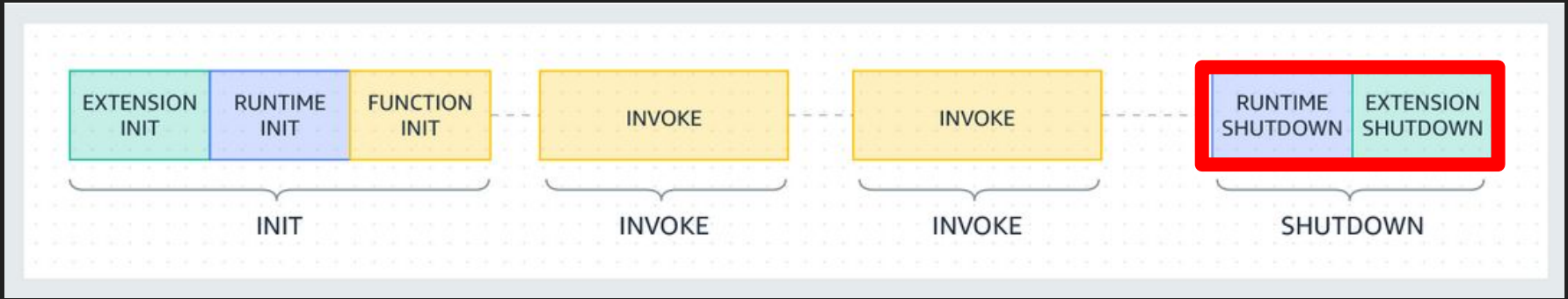
AWS Lambda - lifecycle

Invoke phase: executes the lambda handler for any pending events. If the Lambda function crashes or times out during the Invoke phase, Lambda resets the execution environment.



AWS Lambda - lifecycle

Shutdown phase: Stops the runtime and shuts down any extension. The execution environment is kept “frozen” for a period of time in anticipation for further requests (link).



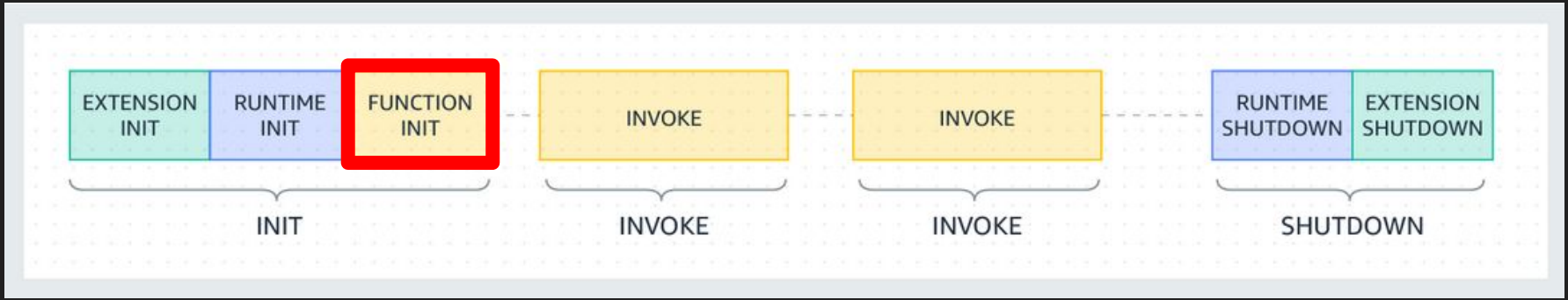
AWS Lambda - lifecycle

Cool! But remember: it’s still someone else’s computer!

AWS Lambda uses microVMs behind the scenes to execute Lambda functions (and containers) with the required security, performance and isolation.

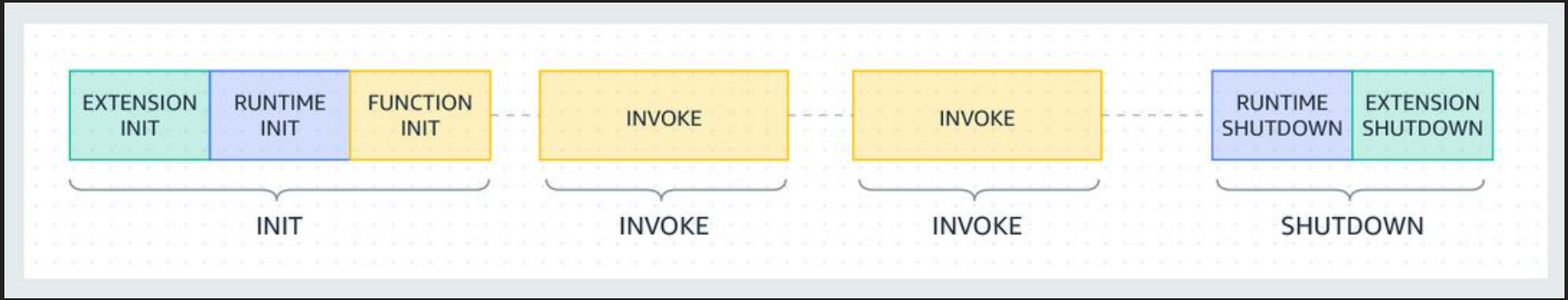
AWS Lambda - lifecycle

Pro-tip: if you have some expensive resource you need to set-up and and re-use across invocations, e.g. a database connection, you can use the function init stage.



AWS Lambda - lifecycle

Pro-tip: one note of caution if you plan to use Java, the JVM is extremely powerful and robust but it takes some seconds to get started. This can lead to erratic latencies up to several seconds as the Lambda is started.



AWS Lambda - security and permissions

Any non-trivial example will require wiring together several different AWS services. In the next example we will build a simple REST-like API, and for that we will need to set up permissions to use the API Gateway.

This is a fairly large and complex topic, and in this session we will only scratch the surface.

AWS Lambda - security and permissions

We will use AWS Identity and Access Management (IAM) to manage access to the required resources. There two key elements to keep in mind:

- Execution role: where the permissions for the resources your lambda needs are expressed;
- Resource based policy: to allow other AWS services to use your AWS Lambda function.

AWS Lambda - security and permissions

Pro-tip: if your Lambda needs to use other AWS services, you don’t need to pass any AWS credentials! The AWS lambda execution environment already has a set of temporary credentials for the execution role in the following environment variables: AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, and AWS_SESSION_TOKEN

DEMO-02

Your first AWS Lambda HTTP service (source)

AWS Lambda - What have we learned so far?

- AWS Lambda is cool! Seems straightforward (foreshadowing…);
- The first request has some overhead (cold start);
- To do anything useful with this, we need to wire AWS Lambda to other services;
- AWS Lambda comes with a few dependencies out-of-the-box.

A question…

- Bruno, but what if we need to add dependencies to our project?

AWS Lambda - dependencies

The AWS Lambda runtime environment already provides a lot of dependencies out-of-the-box, in particular the AWS SDK (link). So you may not need to package any dependencies.

However you may need to add your own dependencies. So far we have manually deployed from the AWS console. Let’s learn how to deal with dependencies by using better deployment tools!

AWS Lambda - deployment

So far we have been doing things manually. It’s time to talk about deployment options:

- Cloud formation - link;
- Serverless Application Model (SAM) - link;
- Cloud Development Kit (CDK) - link.

DEMO-03

Your first SAM deployment of a Lambda function with dependencies (source)

AWS Lambda - What have we learned so far?

- In practice you typically don’t deploy directly using the AWS console, in this example we saw how to use the SAM CLI to deploy a Lambda function;

AWS Lambda - monitoring

By default you can monitor your AWS lambda function via Cloudwatch (for logging and metrics) and X-Ray (for distributed tracing).

A good practice is to use the Lambda power tools. It offers some nice improvements over what you get out-of-the-box in terms of logs, metrics and traces (and many more things!).

DEMO-04

Your first Lambda layer + Lambda powertools usage (source)

AWS Lambda - What have we learned so far?

- Fully fleshed out example of a Lambda function that can be deployed via the SAM CLI tool;
- Usage of layers for keeping your dependencies separated from your code;
- Usage of the Lambda Powertools, especially to enrich monitoring.

AWS Lambda - limits

Guess what, at the end of the day serverless is someone else's computer, which means that there are limits/quotas!

AWS Lambda quotas - link

AWS Lambda - limits

Some noteworthy limits:

- Lambda function execution time: 15 min. (900 sec.);
- Deployment package size: 50MB (zipped) / 250MB (unzipped), shared with layers;
- Container image size: 10 GB (maximum uncompressed image size, including all layers);
- Concurrent executions per account: 1000.

AWS Lambda - cost

It’s not trivial (link)!

- You are charged per request. More requests = more money;
- You can choose the memory you want to allocate your function (more memory comes with more CPU). More memory = more money;
- You pay for the duration of the execution of your function rounded to the millisecond. More execution time = more money;
- Don’t forget storage and network costs;
- At scale these costs add up quite quickly.

AWS Lambda - downsides

As with every other technology, it’s tradeoffs all the way. Let’s look at some downsides:

- Reasoning about large code bases that are a tangle of Lambda functions can be hard (believe me, I’ve been there…);
- Testing in general is not straightforward, especially if you are using several AWS services;
- Java is not a good fit if your use case requires consistent and “fast” responses;
- Cost estimation is not trivial.

AWS Lambda - use cases

Use cases where I’ve seen AWS Lambda succeed:

- Rapid prototyping;
- Periodic workloads (e.g. processing data once every 24h, APIs that need to exist but are not called very often);
- Event driven workloads (especially if those events come from other AWS Services).

Parting words

AWS Lambda is a nice tool to have in your toolbelt.

Engineering is all about tradeoffs - it is exceedingly rare to have “perfect” solutions. For certain use cases, especially if you are working with the AWS ecosystem this technology can be used to great effect.

Thanks!
Any questions?