AWS Lambda

in **RUST**

Rust Slovakia Meetup, 7.11.2023

František Horváth

Serverless Computing

- A way of running applications in the cloud
- There are servers, but You don't have to provision and manage them
- Only pay for what you use
- Small units of compute (functions), triggered by events

Not a universal solution, but it can work well in many situations!

AWS Lambda

Lambda function - serverless **Function as a Service** in AWS

Can be triggered by different kinds of events

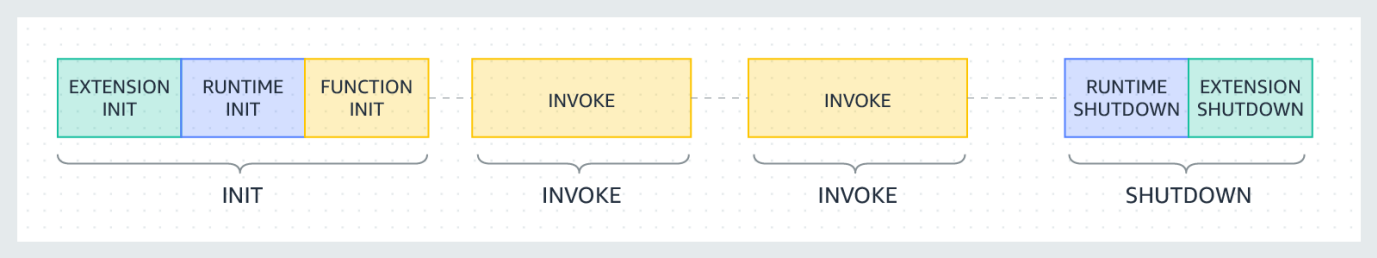
- HTTP Requests
- New files in S3
- Jobs in a Queue
- Orchestrated by Step Functions
- On a schedule
- Manually invoked

AWS Lambda - Limitations

- Memory size 128MB – 10240MB
- CPU scales by memory (2-6 vCPU cores)
- Function execution timeout is 15 minutes
- Invocation payload size (request/response)
- Environment variables (4kB)
- Deployment package size (50MB zipped)
- Function layers (max 5 layers)

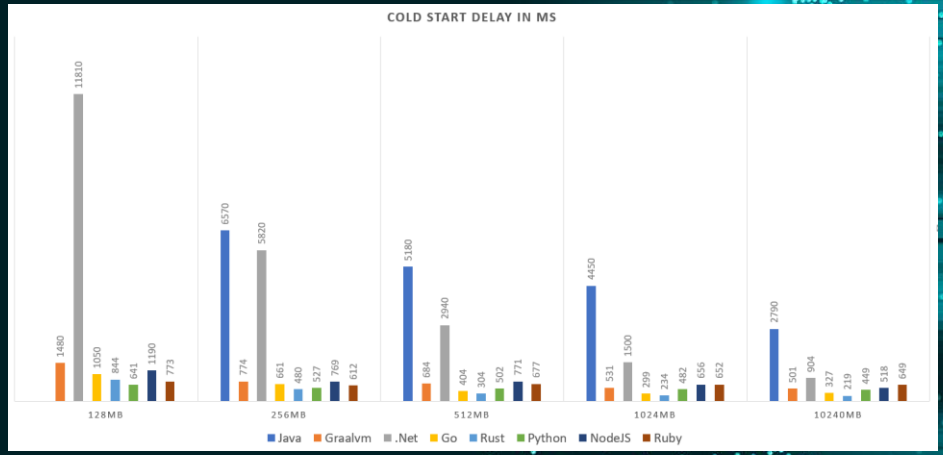
AWS Lambda - Lifecycle

- Lambda is serverless, it should run only when needed
- Lambda code is stored in S3 and it's executed only when triggered
- If no instance is available, one is created on the fly (cold-start)
- If an instance is available and ready, use that one
- If an instance is inactive for a while, it gets destroyed



AWS Lambda – Why Rust?

- Fast cold start
- Cost saving
- Well known benefits of Rust
  - Performance
  - Memory safety
  - Multi-thread safety
  - Error handling
  - Small footprint



Cargo-lambda

A Cargo subcommands that provides tools and workflows to help you develop AWS Lambda functions in Rust.

Cargo Lambda provides several subcommands for different tasks:

- New/Init
- Build
- Watch/Invoke
- Deploy

Terraform

Terraform is an Infrastructure as a Code tool that lets you build, change, and version infrastructure safely and efficiently.

Manage AWS Services

- Use the AWS provider to manage AWS services with Terraform
- Configure IAM policy documents, deploy Lambda functions

Terragrunt

Terragrunt is a thin wrapper for Terraform that provides extra tools for keeping your Terraform configurations DRY.

- Provides extra tools to manage complex infrastructure stacks with multiple dependencies and environments
- Define your Infrastructure as Code in a concise, maintainable, and reusable way
- Define your infrastructure code in a hierarchy of folders and files
- Simplifies Terraform remote state management

Observability

LOGS

(un)structured text records of discreet events that occurred at a specific time

TRACES

activity of a transaction or request as it flows through applications

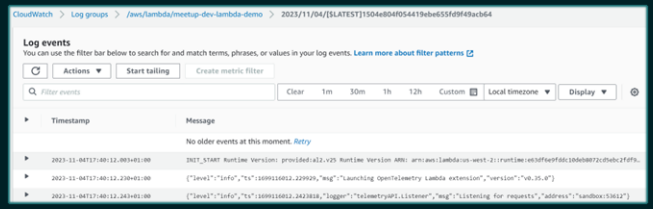
METRICS

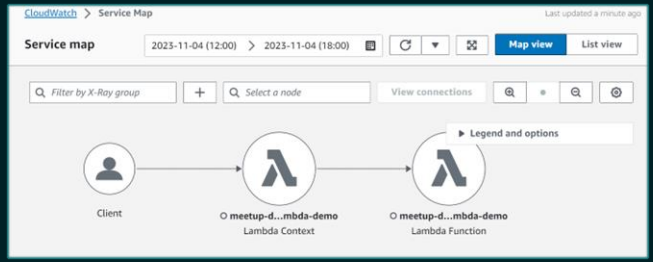
values as counts or measures that are often calculated or aggregated over a period of time

Observability – AWS Services

AWS Cloudwatch

- Log events per application
- Metrics
- X-Ray Traces
  - Service Map
  - Traces





Observability – AWS Services + Rust

Rust has no reflection - no automated code instrumentation

Rust has good support for OpenTelemetry (OTLP)

Instrument your functions manually and export OTLP records

```
#[tracing::instrument(ret, err)]
async fn function_handler(lambda_event: LambdaEvent<Request>) -> Result<Response, Error> {
```

Deploy AWS Distro for OpenTelemetry

- provides OTLP Collector
- converts the OTLP records to AWS Cloudwatch

THANKS!

Do you have any questions?

https://github.com/Rust-Slovakia/Bratislava-Rust-Meetup

fhorvath80@gmail.com

www.linkedin.com/in/horvathfrank

DEMO!