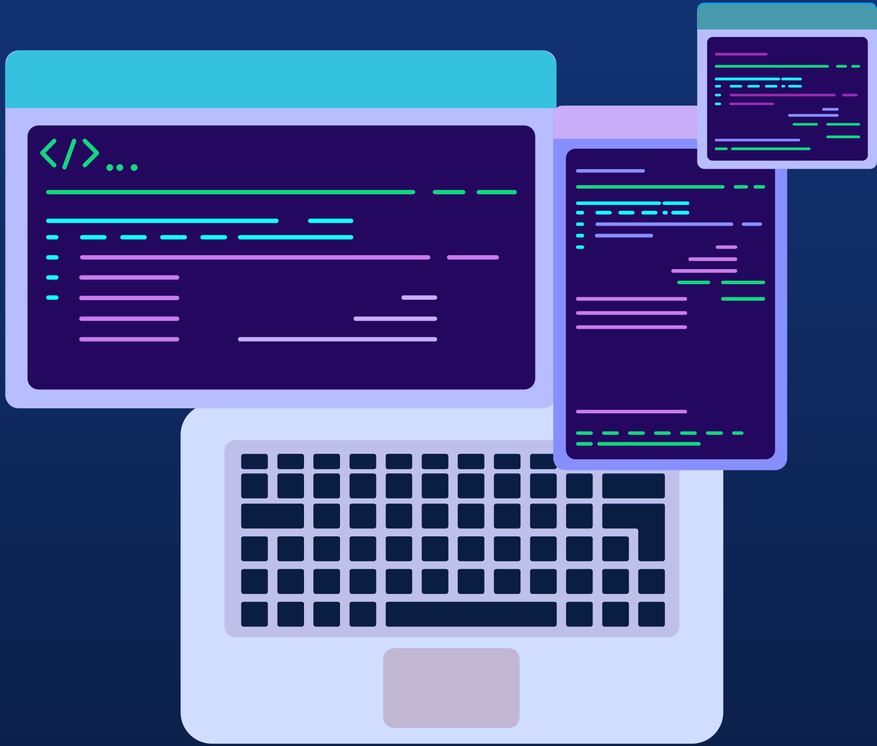
Serverless computing



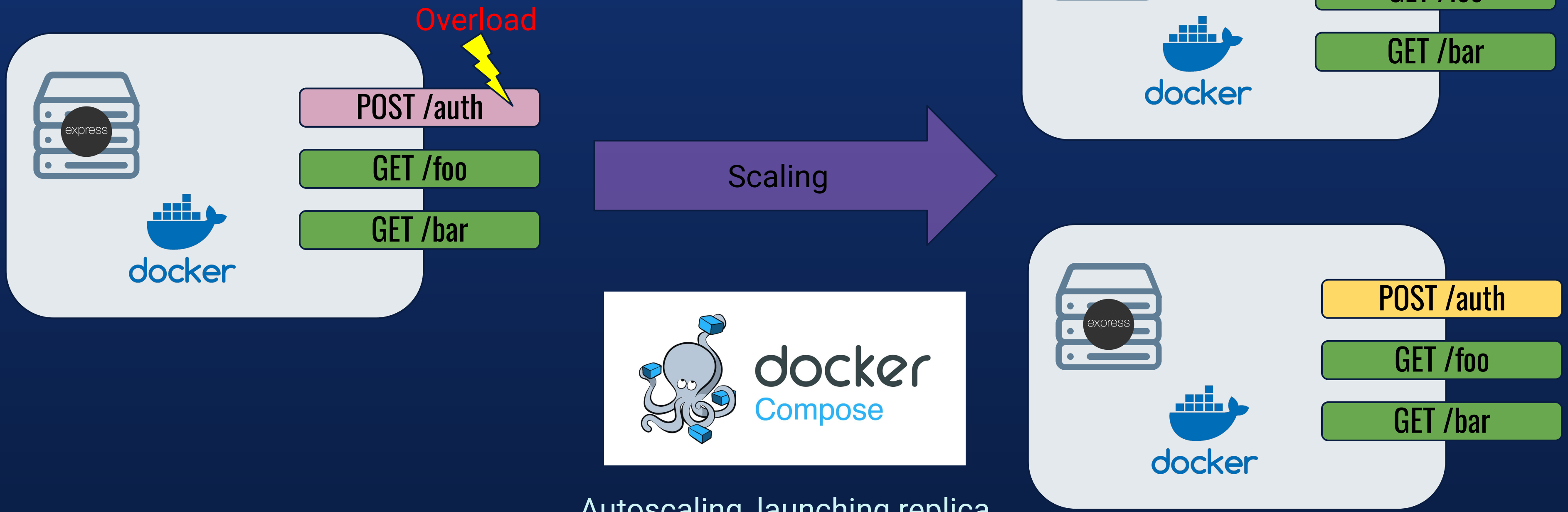
- Pay-as-you-use infrastructure
- Event-driven, automatic scaling
- Key component: functions as a service (FaaS)

Functions vs applications

	Functions	Traditional API	
✓	Native auto-scaling	Manual or scripted scaling, not so efficient	
✓	Pay as you use	Upfront infrastructure payment, but fixed price	✓
✓	Simplified deployment	User has to manage access, network…	
✓	Very lightweight	Cannot be as light as functions	
	Inefficient for heavy monothreaded processes	Bigger machine = better monothreaded performance	✓

Auto-scaling use case

Traditional API : Dockerized ExpressJS API



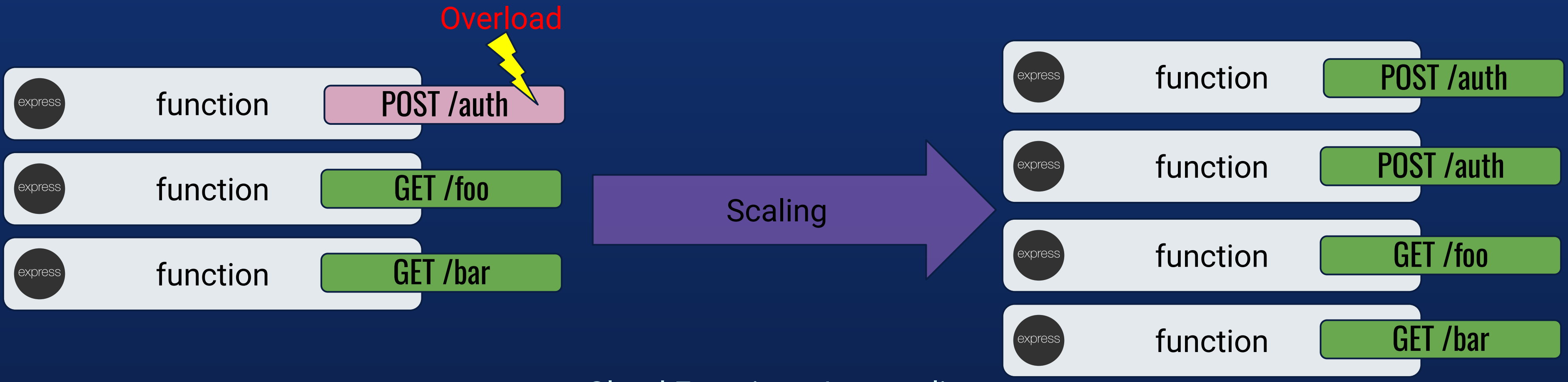
Autoscaling, launching replica container

Pros : Cost control (auto-scaling scripted, we exactly know what it will cost)

Cons : Inefficient, a whole API is redeployed because of a single endpoint, what about downscaling ?, not so fast

Auto-scaling use case

Cloud Function : ExpressJS API



Cloud Functions Autoscaling

Pros : Efficient scaling, automatized downscaling, very fast

Cons : Cost control

A bit of vocabulary, for a better serverless configuration

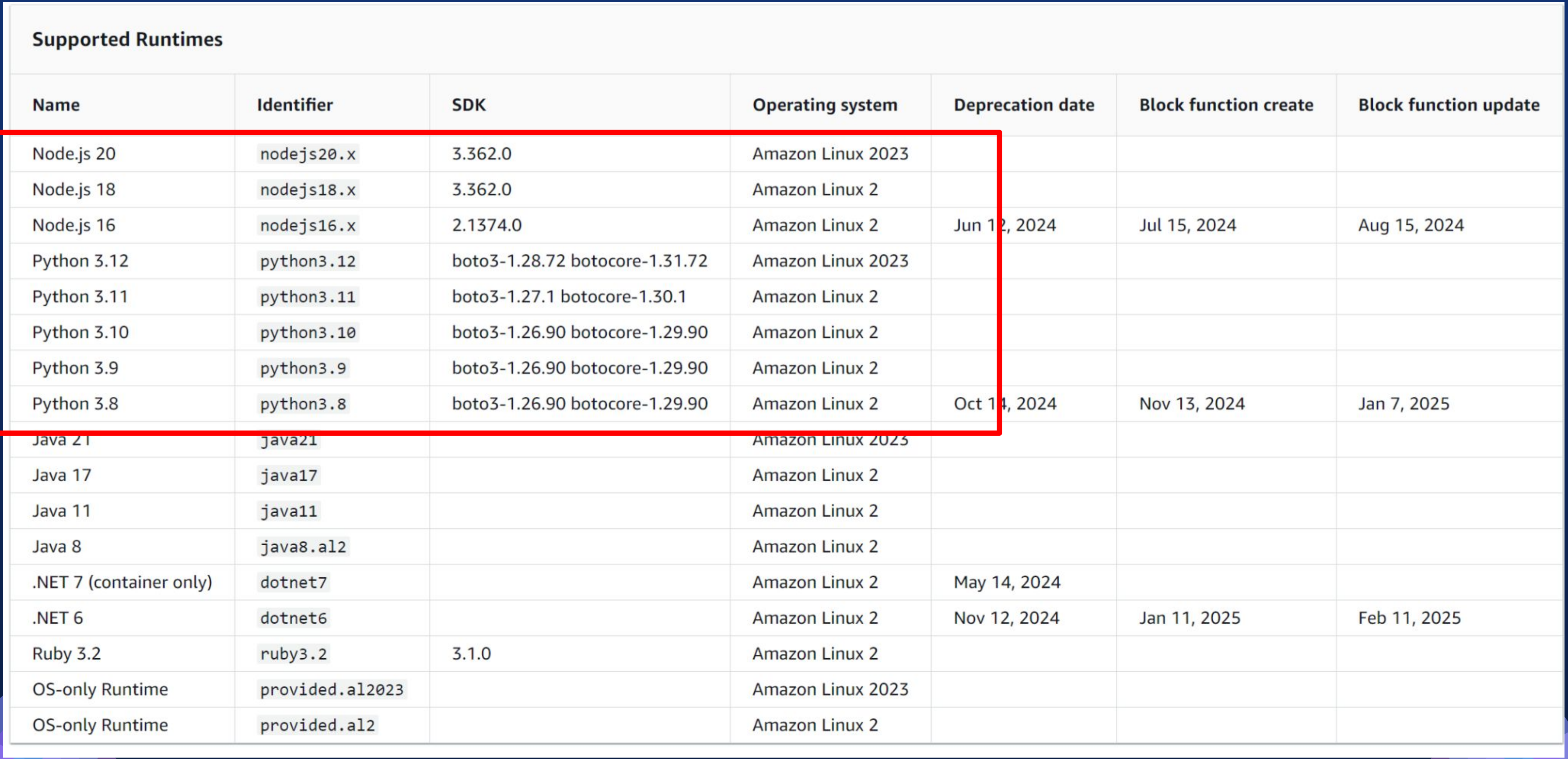
- **Cold start** : Initial invocation delay when no function instances are available, implying a longer response time
- **Uptime** : Duration of uninterrupted function availability
- **Timeout** : Maximum Uptime
- **Concurrency** : Maximum simultaneous executions of a function (cost management).
- **Warmup** : Instanciates recurrently the functions, so they are never cold (costs more, but no cold start)

Two main concurrents

AWS lambda

GCP functions

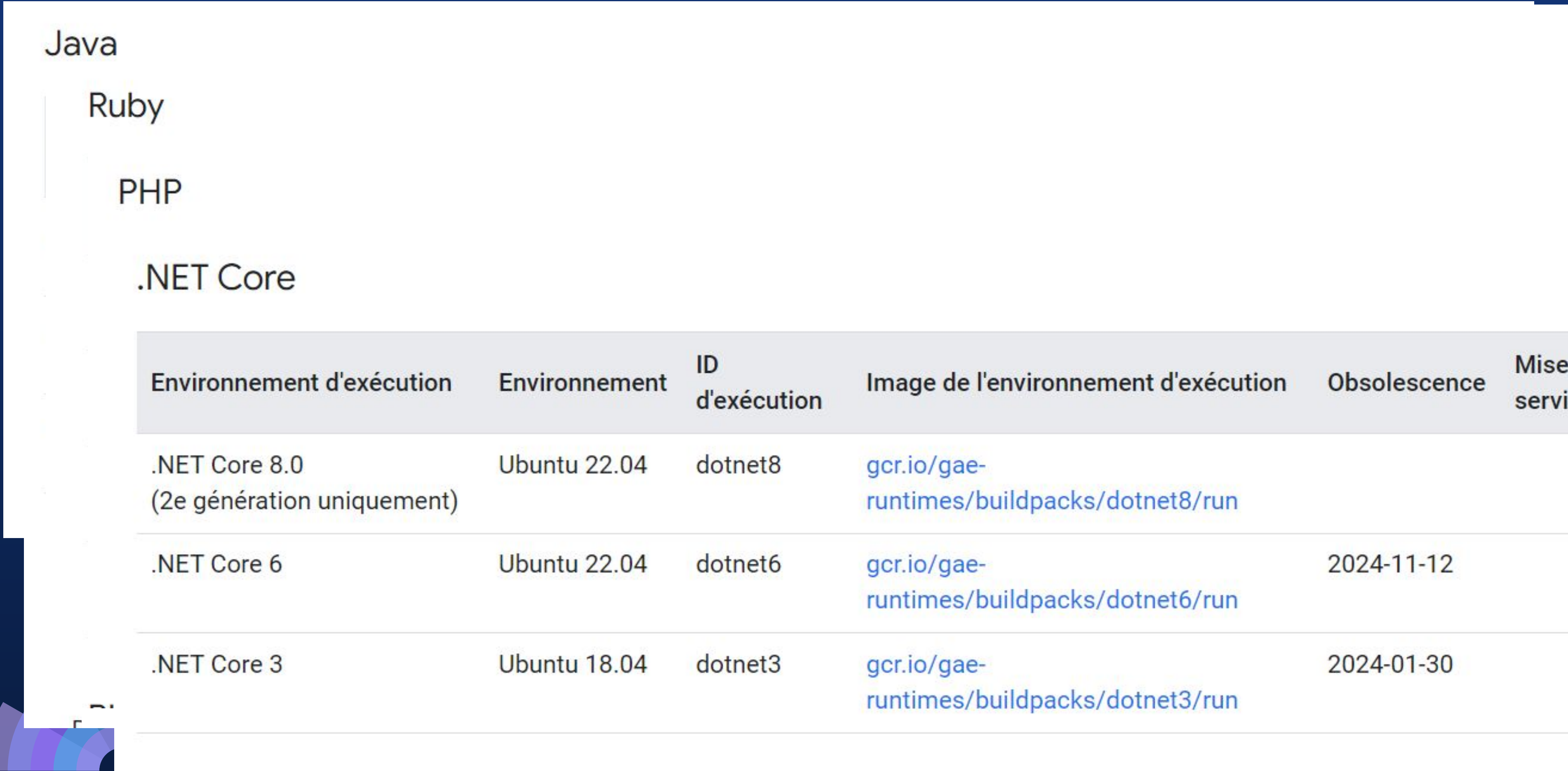
AWS Lambda : supported runtimes



Supported Runtimes

Name	Identifier	SDK	Operating system	Deprecation date	Block function create	Block function update
Node.js 20	`nodejs20.x`	3.362.0	Amazon Linux 2023			
Node.js 18	`nodejs18.x`	3.362.0	Amazon Linux 2			
Node.js 16	`nodejs16.x`	2.1374.0	Amazon Linux 2	Jun 12, 2024	Jul 15, 2024	Aug 15, 2024
Python 3.12	`python3.12`	boto3-1.28.72 botocore-1.31.72	Amazon Linux 2023			
Python 3.11	`python3.11`	boto3-1.27.1 botocore-1.30.1	Amazon Linux 2			
Python 3.10	`python3.10`	boto3-1.26.90 botocore-1.29.90	Amazon Linux 2			
Python 3.9	`python3.9`	boto3-1.26.90 botocore-1.29.90	Amazon Linux 2			
Python 3.8	`python3.8`	boto3-1.26.90 botocore-1.29.90	Amazon Linux 2	Oct 14, 2024	Nov 13, 2024	Jan 7, 2025
Java 21	`java21`		Amazon Linux 2023			
Java 17	`java17`		Amazon Linux 2			
Java 11	`java11`		Amazon Linux 2			
Java 8	`java8.al2`		Amazon Linux 2			
.NET 7 (container only)	`dotnet7`		Amazon Linux 2	May 14, 2024		
.NET 6	`dotnet6`		Amazon Linux 2	Nov 12, 2024	Jan 11, 2025	Feb 11, 2025
Ruby 3.2	`ruby3.2`	3.1.0	Amazon Linux 2			
OS-only Runtime	`provided.al2023`		Amazon Linux 2023			
OS-only Runtime	`provided.al2`		Amazon Linux 2			

GCP Functions : supported runtimes



Java

Ruby

PHP

.NET Core

Environnement d'exécution	Environnement	ID d'exécution	Image de l'environnement d'exécution	Obsolescence	Mise servi
.NET Core 8.0 (2e génération uniquement)	Ubuntu 22.04	dotnet8	gcr.io/gae-runtimes/buildpacks/dotnet8/run		
.NET Core 6	Ubuntu 22.04	dotnet6	gcr.io/gae-runtimes/buildpacks/dotnet6/run	2024-11-12	
.NET Core 3	Ubuntu 18.04	dotnet3	gcr.io/gae-runtimes/buildpacks/dotnet3/run	2024-01-30	

● Unlimited lambda by project	● 1000 functions max by project
● Less expensive for very big projects	● Less expensive for small projects
● Cold start <1s	● Cold start 1-2sec
● RAM: 128Mo - 10Go	● RAM: 128Mo - 4Go

One framework to unite them all

Invented by Austen Collin in 2015

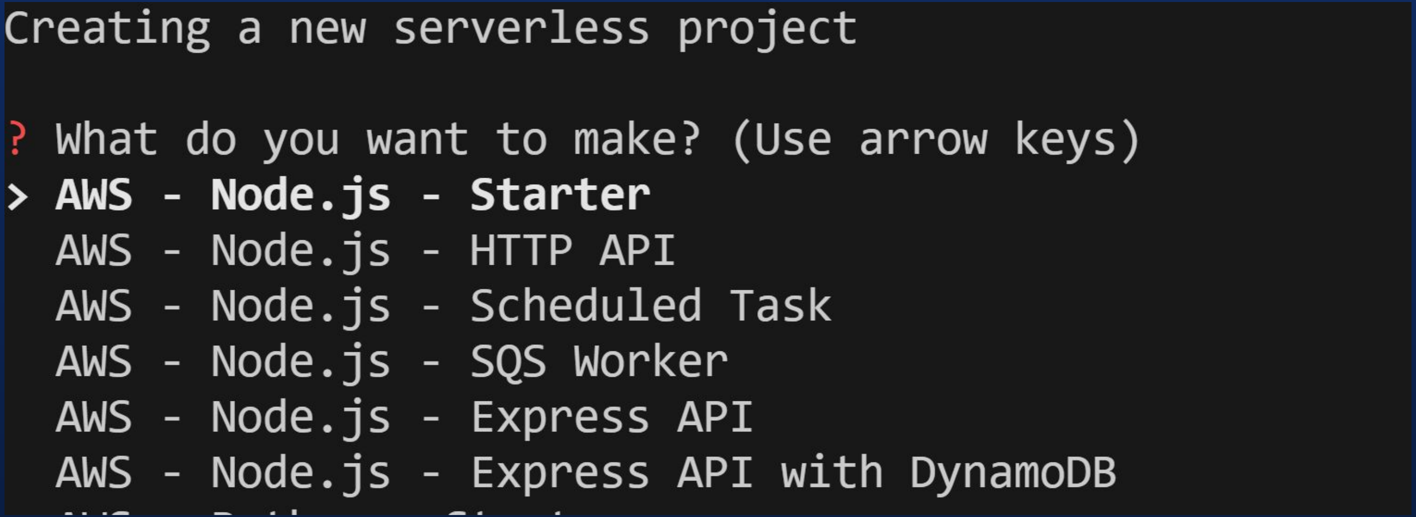
First name was JAWS, because initially purposed for AWS

Now compatible with all major cloud providers (AWS, GCP, Azure, Alibaba, IBM....)

Create a serverless project : the simple way explained with nodeJS

```
$ npm install -g serverless
```

```
$ serverless
```



```
Creating a new serverless project

? What do you want to make? (Use arrow keys)
> AWS - Node.js - Starter
  AWS - Node.js - HTTP API
  AWS - Node.js - Scheduled Task
  AWS - Node.js - SQS Worker
  AWS - Node.js - Express API
  AWS - Node.js - Express API with DynamoDB
```

serverless.yml

```yaml
service: tp05-poirrier
frameworkVersion: '3'

provider:
  name: aws
  runtime: nodejs18.x

functions:
  auth:
    handler: auth.handler
    events:
      - httpApi:
          method: POST
          path: /auth
  routeFoo:
    handler: routeFoo.handler
    events:
      - httpApi:
          method: GET
          path: /foo
```

Deploy to AWS Lambda

```
$ pip install aws-cli
```

```
$ aws configure
```

Enter your credential set when asked

```
$ serverless deploy
```

… that’s all folks !

Deploy to GCP Cloud functions

```
$ npm install --save serverless-google-cloudfunctions
```

```
$ gcloud auth <app-name> login
```

Enter your credential set when asked

```
$ serverless deploy
```

… that’s all folks !