Quintor

WELKOM

SERVERLESS LAMBDA WORKSHOP

EVEN VOORSTELLEN

PIM OTTE

- Consultant @ Quintor

- Email: p.otte@quintor.nl
- LinkedIn: pim-otte-95b4ba124

- Specialisaties:
- Fullstack Java Development
- Cloud Native Software Development
- Blockchain Development

Quintor

- Java
- .Net
- Frontend
- Platform Engineering
- Security
- Agile Analyse

- Den Haag
- Den Bosch
- Amersfoort
- Deventer
- Groningen

https://quintor.nl/

Quintor

- Minor Cloud-Native Software Development
  - https://quintor.nl/minors/
- Afstuderen bij Quintor
  - https://quintor.nl/student/
- Young Professional Programma
  - https://quintor.nl/young-professional/

PROGRAMMA VANDAAG

- Theorie
- Hands-on opdracht

PROGRAMMA

- Serverless Architecture
- Function-as-a-Service
- AWS Lambda
- Serverless Application Model
- API Gateway
- DynamoDB

SERVERLESS ARCHITECTURE

When we run our own servers:

- We are charged for keeping the server up even when we are not serving out any requests.
- We are responsible for uptime and maintenance of the server and all its resources.
- We are also responsible for applying the appropriate security updates to the server.
- As our usage scales we need to manage scaling up our server as well. And as a result manage scaling it down when we don't have as much usage.

SERVERLESS ARCHITECTURE

What if we don't need to bother about servers?

What if we only create the system parts we care about?

CaaS, Paas, FaaS, DBaaS, TaaS, QaaS, ...

Serverless != FaaS, but are best friends!

SERVERLESS ARCHITECTURE

ADVANTAGES

- Focus: Less infrastructure management, more application development
- Turnaround: Can significantly cut time to market
- Scaling: Simplified scalability provided by serverless providers
- Costs: Don't pay for unused space, memory or idle CPU time

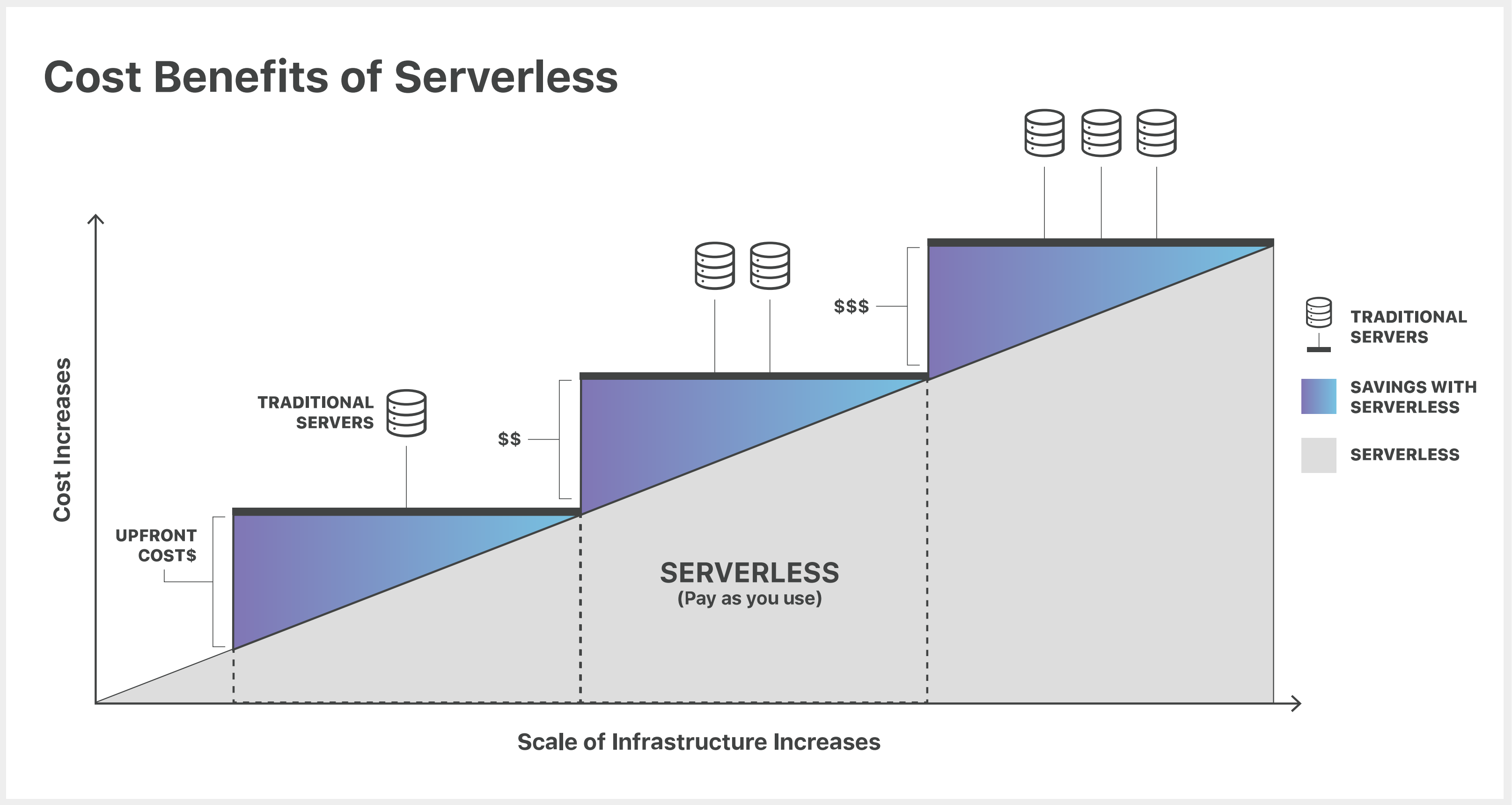
SERVERLESS ARCHITECTURE

DISADVANTAGES

- Vendor lock-in is a risk
- Testing and debugging become more challenging
- Serverless computing introduces new security concerns
- Serverless architectures are often not built for long-running processes
- Performance may be affected

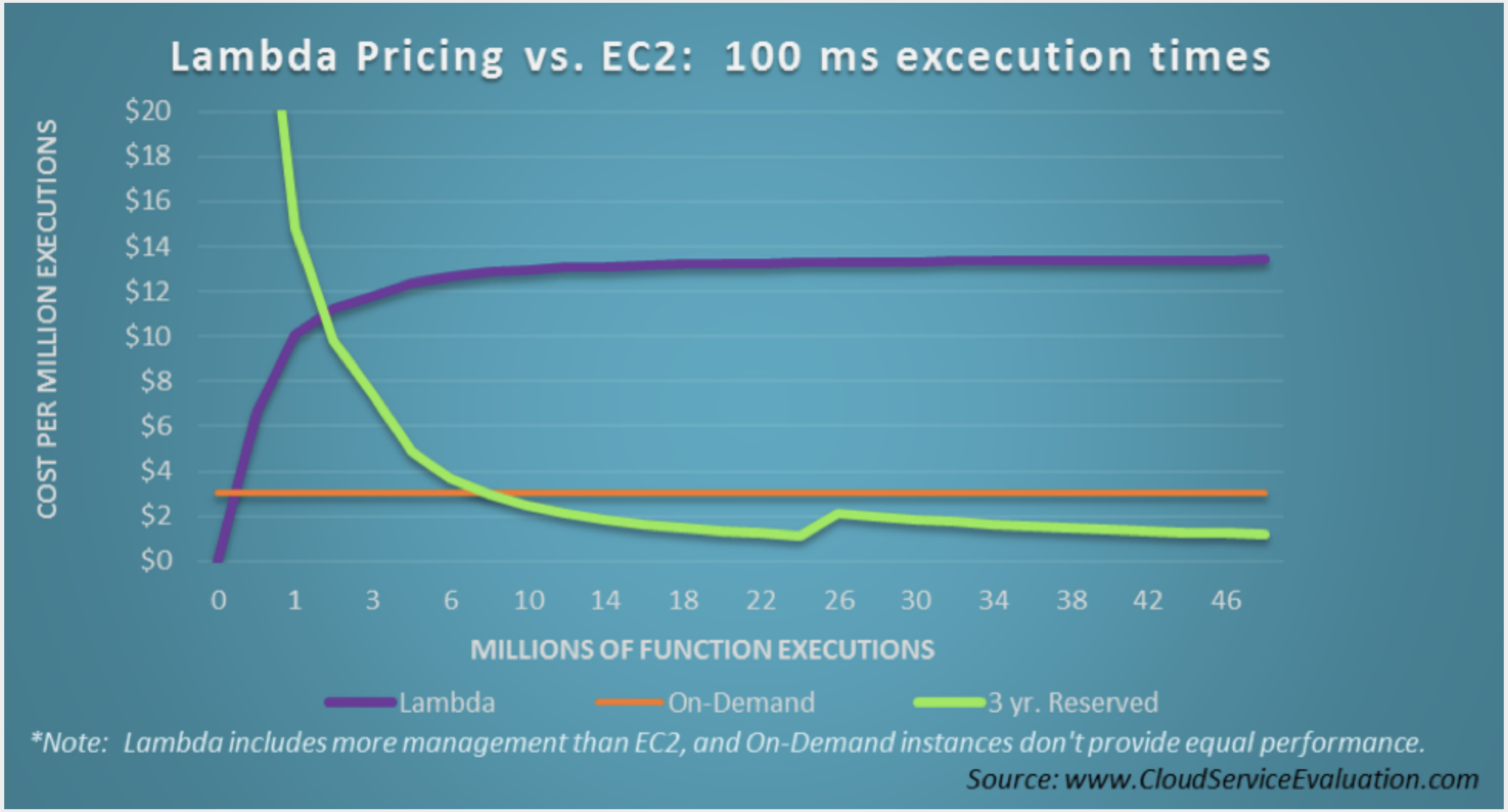
SERVERLESS ARCHITECTURE

COSTS



SERVERLESS ARCHITECTURE

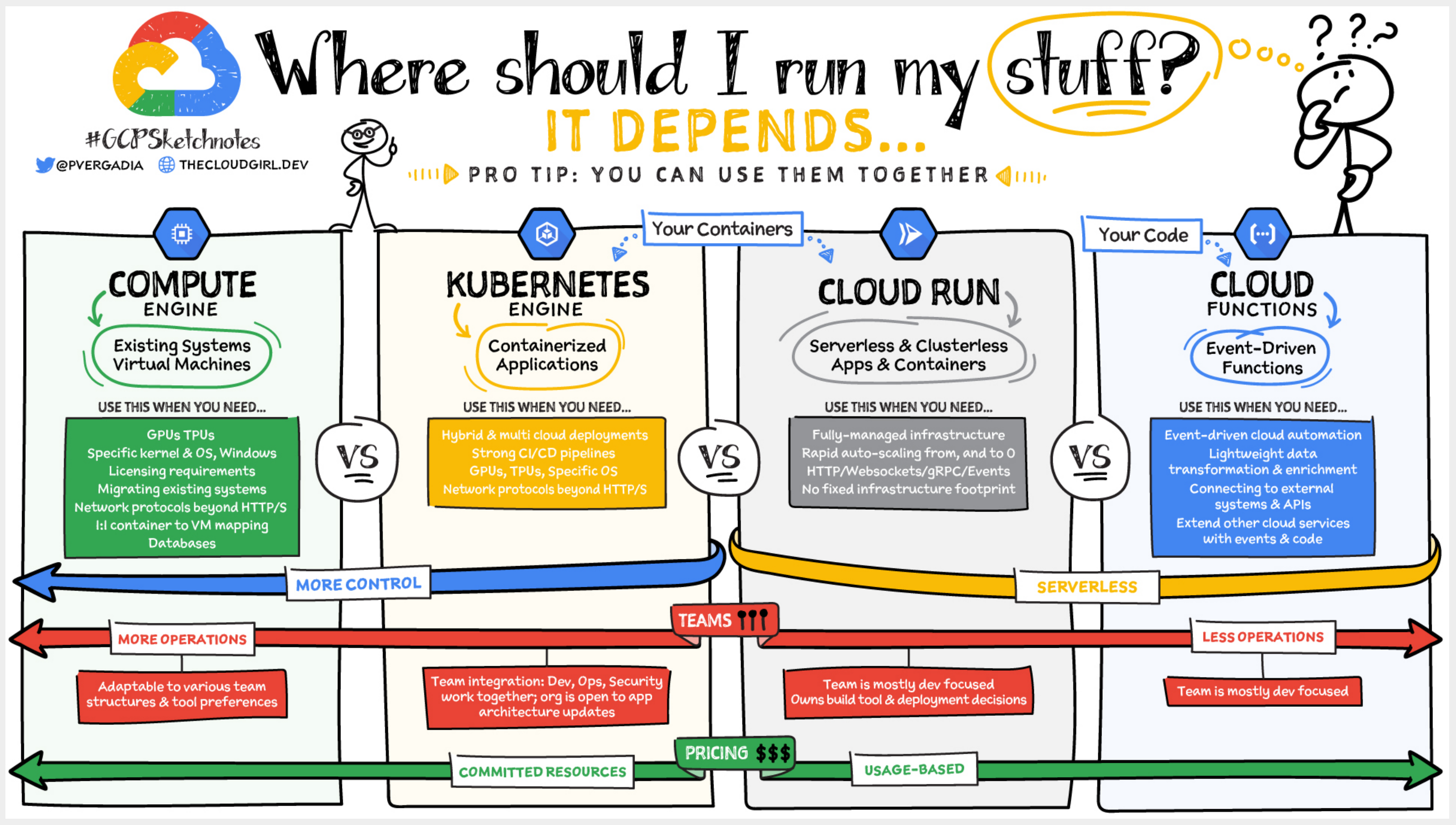
COSTS



**Note: Lambda includes more management than EC2, and On-Demand instances don't provide equal performance.*

Source: www.CloudServiceEvaluation.com

SERVERLESS ARCHITECTURE



Where should I run my stuff?

IT DEPENDS...

PRO TIP: YOU CAN USE THEM TOGETHER

#GCPSketchnotes

@PVERGADIA THECLOUDGIRL.DEV

COMPUTE ENGINE

Existing Systems Virtual Machines

USE THIS WHEN YOU NEED...

GPUs TPUs
Specific kernel & OS, Windows
Licensing requirements
Migrating existing systems
Network protocols beyond HTTP/S
1:1 container to VM mapping
Databases

KUBERNETES ENGINE

Containerized Applications

USE THIS WHEN YOU NEED...

Hybrid & multi cloud deployments
Strong CI/CD pipelines
GPUs, TPUs, Specific OS
Network protocols beyond HTTP/S

Your Containers

CLOUD RUN

Serverless & Clusterless Apps & Containers

USE THIS WHEN YOU NEED...

Fully-managed infrastructure
Rapid auto-scaling from, and to 0
HTTP/Websockets/gRPC/Events
No fixed infrastructure footprint

Your Code

CLOUD FUNCTIONS

Event-Driven Functions

USE THIS WHEN YOU NEED...

Event-driven cloud automation
Lightweight data transformation & enrichment
Connecting to external systems & APIs
Extend other cloud services with events & code

Adaptable to various team structures & tool preferences

Team integration: Dev, Ops, Security work together; org is open to app architecture updates

Team is mostly dev focused Owns build tool & deployment decisions

Team is mostly dev focused

SERVERLESS ARCHITECTURE

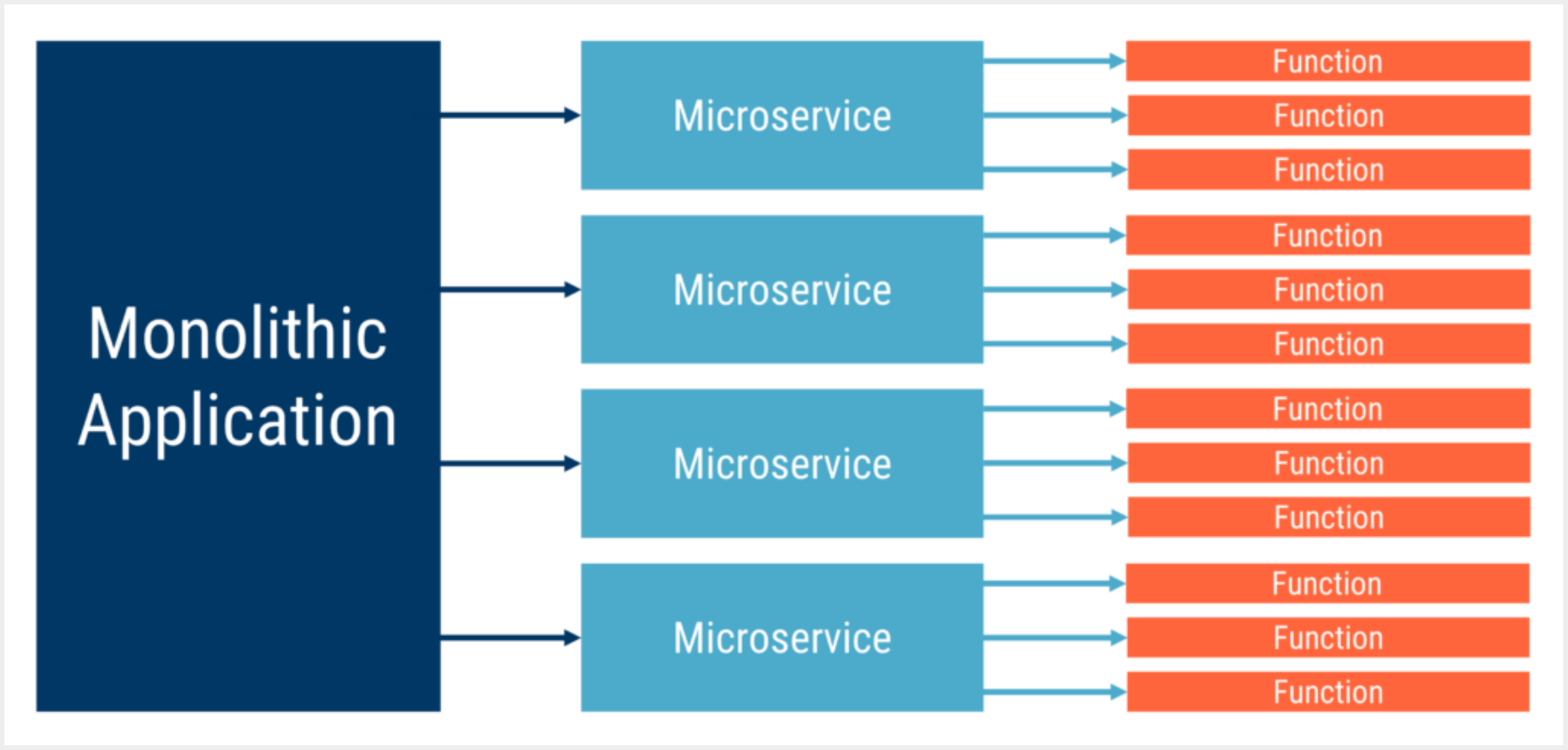
FUNCTION-AS-A-SERVICE

- Run modular chunks of functionality -> Single purpose
- Executed and scaled independently
- Not bothered by complex infrastructure -> Serverless

Serverless != FaaS, but are best friends!

FUNCTION-AS-A-SERVICE

APPLICATION GRANULARITY



FUNCTION-AS-A-SERVICE

ADVANTAGES

- Serverless -> More focus on application development
- Stateless functions are inherently scalable
- Scale down to 0 -> Don't pay for idle resources
- Built in availability and fault tolerance
- Business logic in minimal shippable unit sizes

FUNCTION-AS-A-SERVICE

DISADVANTAGES

- Decreased transparency
- Potentially tough to debug
- Potentially tough to manage costs
- Decreased oversight of your system
- Increased chances of failures

AWS LAMBDA

Run code without provisioning or managing servers.

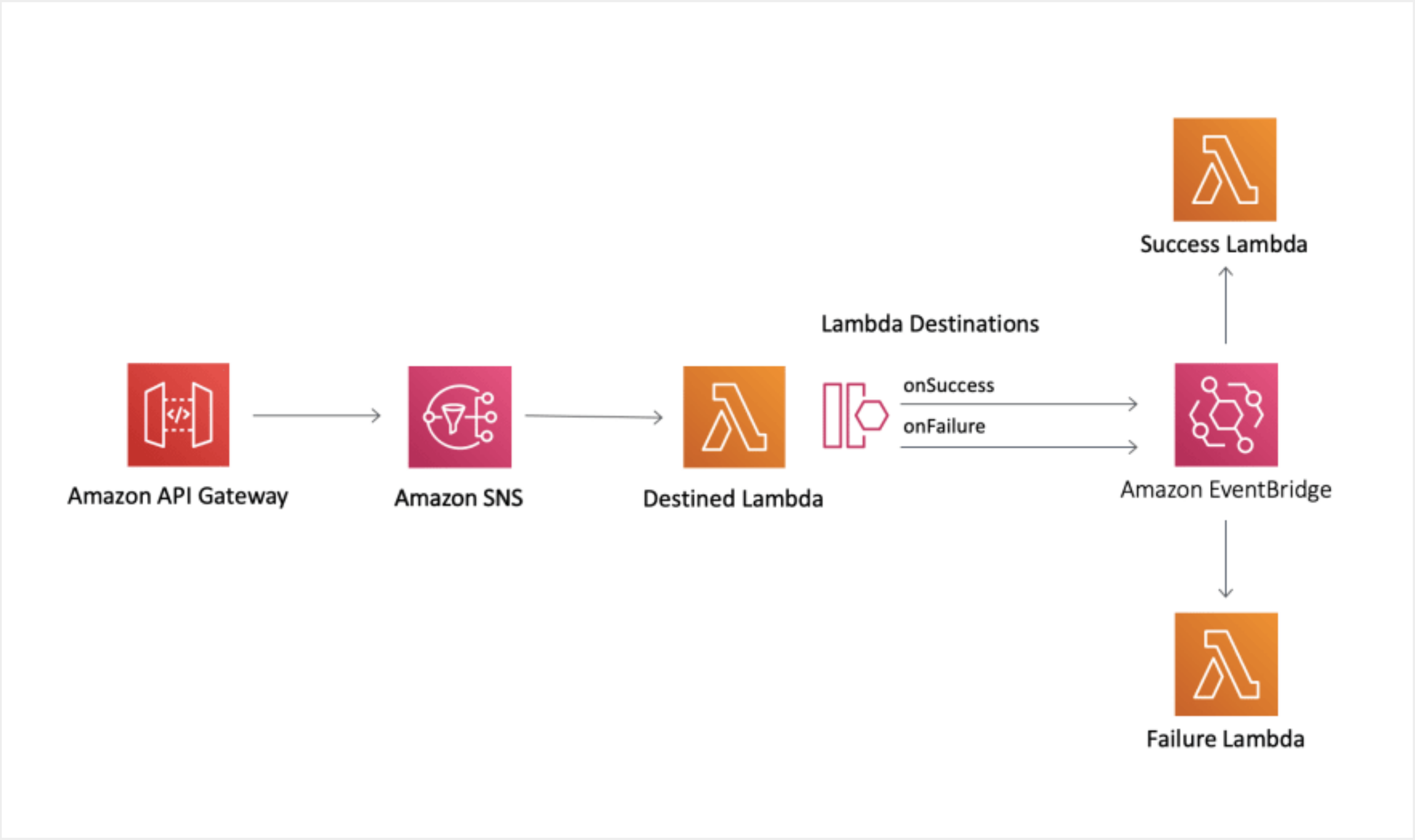
Pay only for the compute time you consume.

AWS LAMBDA

- Runs your code in response to events
- Manages the underlying compute resources
- Create your own back-end services
- Used to extend other AWS services with custom logic
- Runs your code on high-availability compute infrastructure
- Automatic scaling
- Only pay for what you use

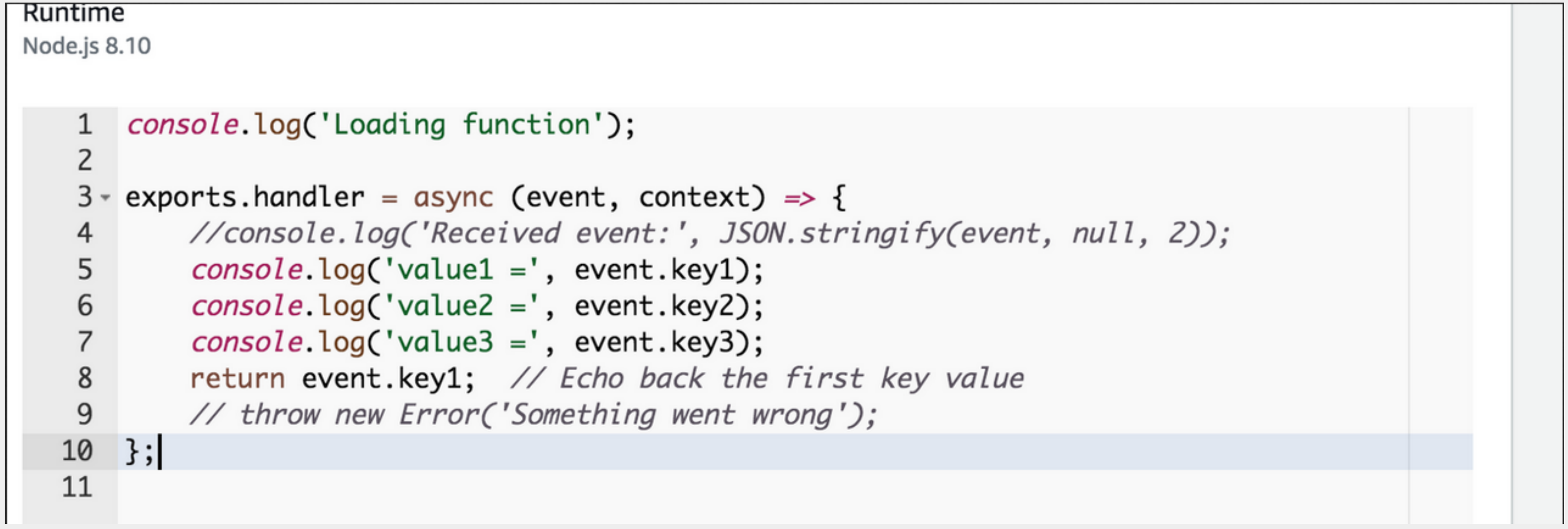
AWS LAMBDA

TRIGGERS


Success Lambda
Lambda Destinations
onSuccess
onFailure
Amazon API Gateway
Amazon SNS
Destined Lambda
Amazon EventBridge
Failure Lambda

AWS LAMBDA

SIMPLE EXAMPLE



Runtime
Node.js 8.10

```
console.log('Loading function');

exports.handler = async (event, context) => {
    //console.log('Received event:', JSON.stringify(event, null, 2));
    console.log('value1 =', event.key1);
    console.log('value2 =', event.key2);
    console.log('value3 =', event.key3);
    return event.key1;  // Echo back the first key value
    // throw new Error('Something went wrong');
};

```

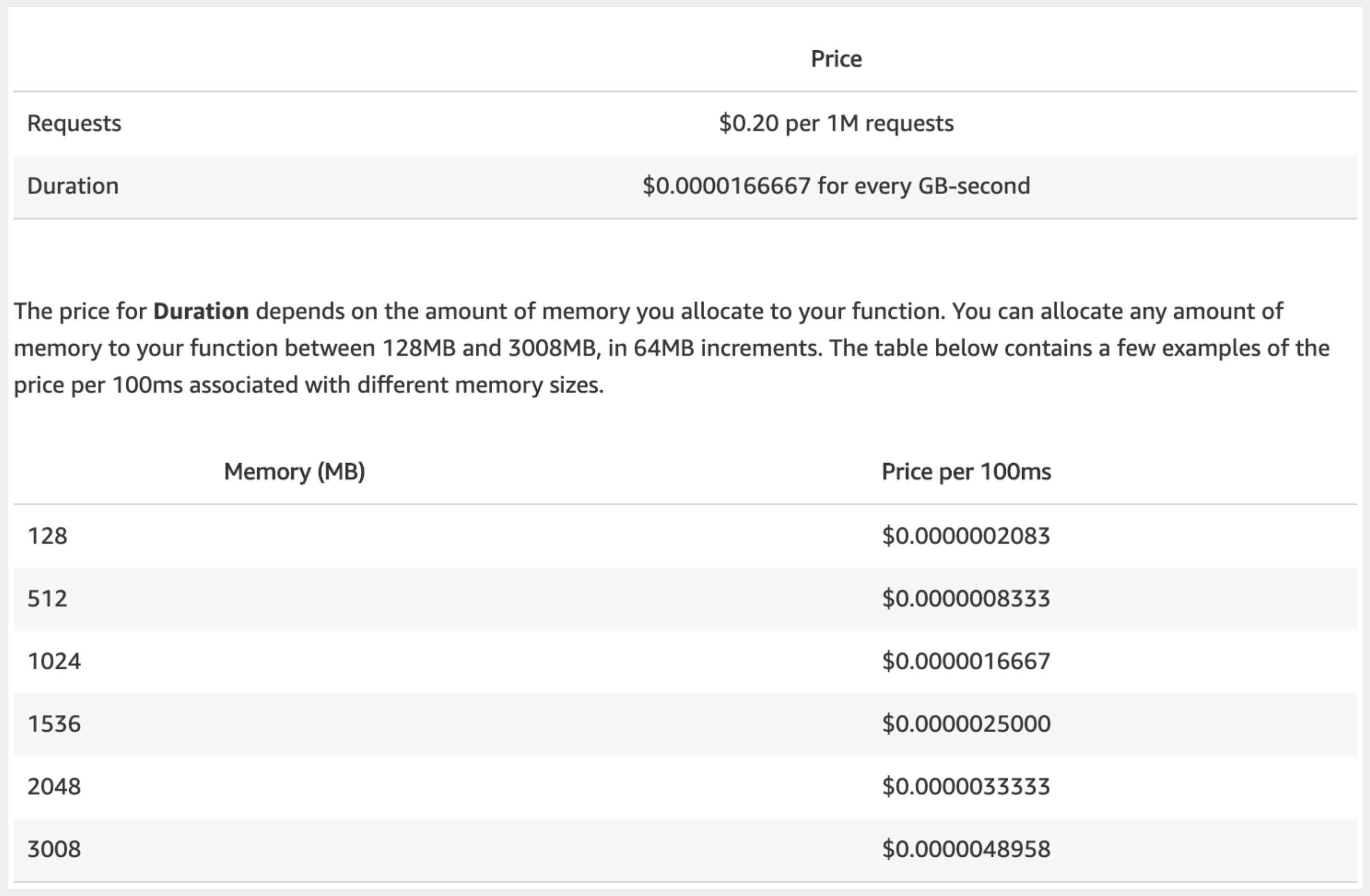
AWS LAMBDA

DYNAMODB EXAMPLE

```
import AWS from 'aws-sdk';
import { ok } from './utils/response';

const dynamoDB = new AWS.DynamoDB.DocumentClient();

export async function list() {
  // Note: Using scan is inefficient and should normally be av
  const accountsResult = await dynamoDB.scan({
    TableName: process.env.ACCOUNT_TABLE,
  }).promise();

  const accounts = accountsResult.Items;
  console.log('List Accounts:', accounts);

  return ok({ accounts });
}
```

AWS LAMBDA

PRICING



	Price
Requests	$0.20 per 1M requests
Duration	$0.0000166667 for every GB-second

The price for **Duration** depends on the amount of memory you allocate to your function. You can allocate any amount of memory to your function between 128MB and 3008MB, in 64MB increments. The table below contains a few examples of the price per 100ms associated with different memory sizes.

Memory (MB)	Price per 100ms
128	$0.0000002083
512	$0.0000008333
1024	$0.0000016667
1536	$0.0000025000
2048	$0.0000033333
3008	$0.0000048958

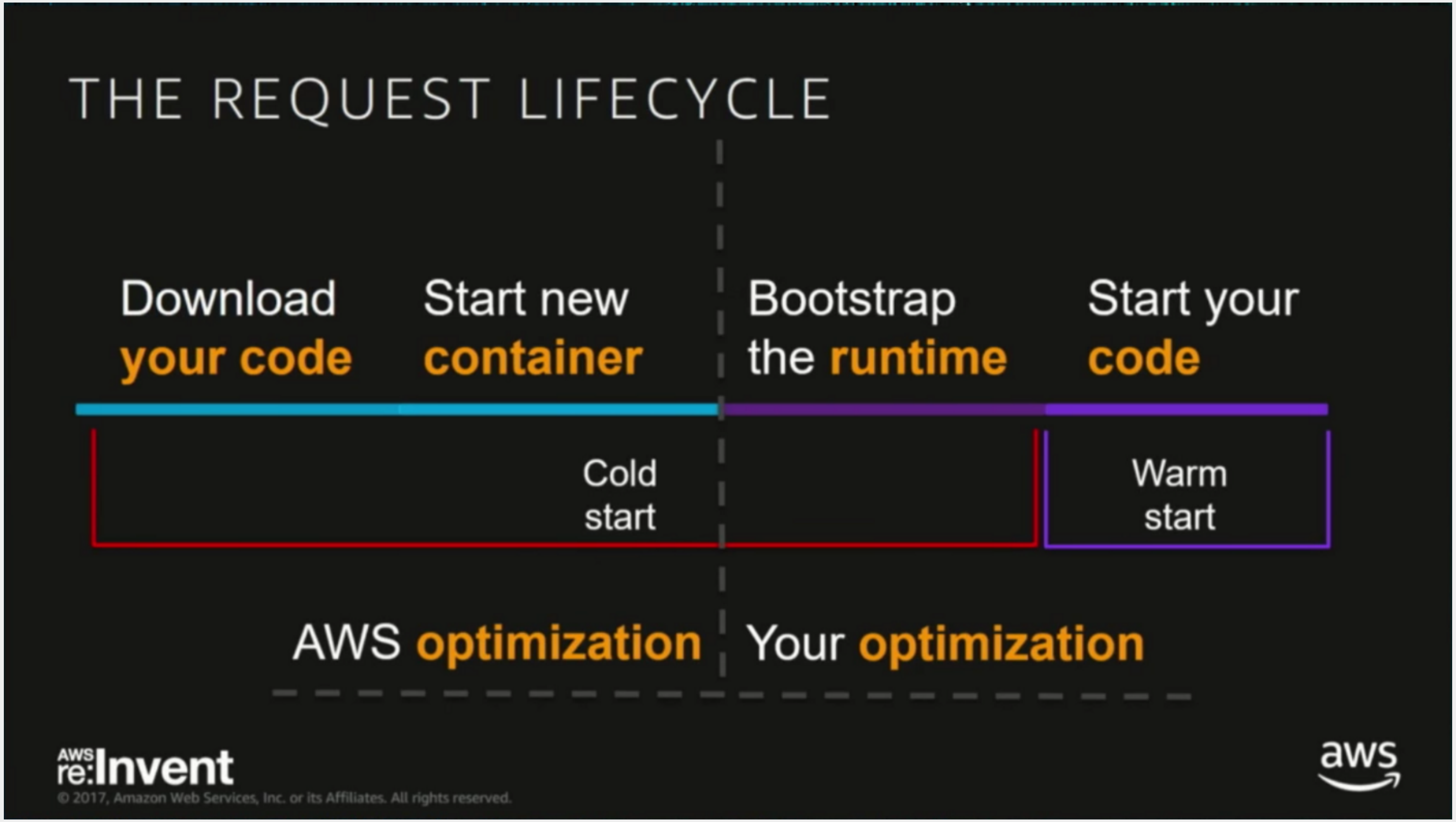
AWS LAMBDA

RUNTIMES

- JavaScript / TypeScript (Node.js)
- Python
- Ruby
- Java
- Go
- C# (.NET Core)
- Custom Runtime

AWS LAMBDA

LIFECYCLE



THE REQUEST LIFECYCLE

Download **your code**

Start new **container**

Bootstrap the **runtime**

Start your **code**

Cold start

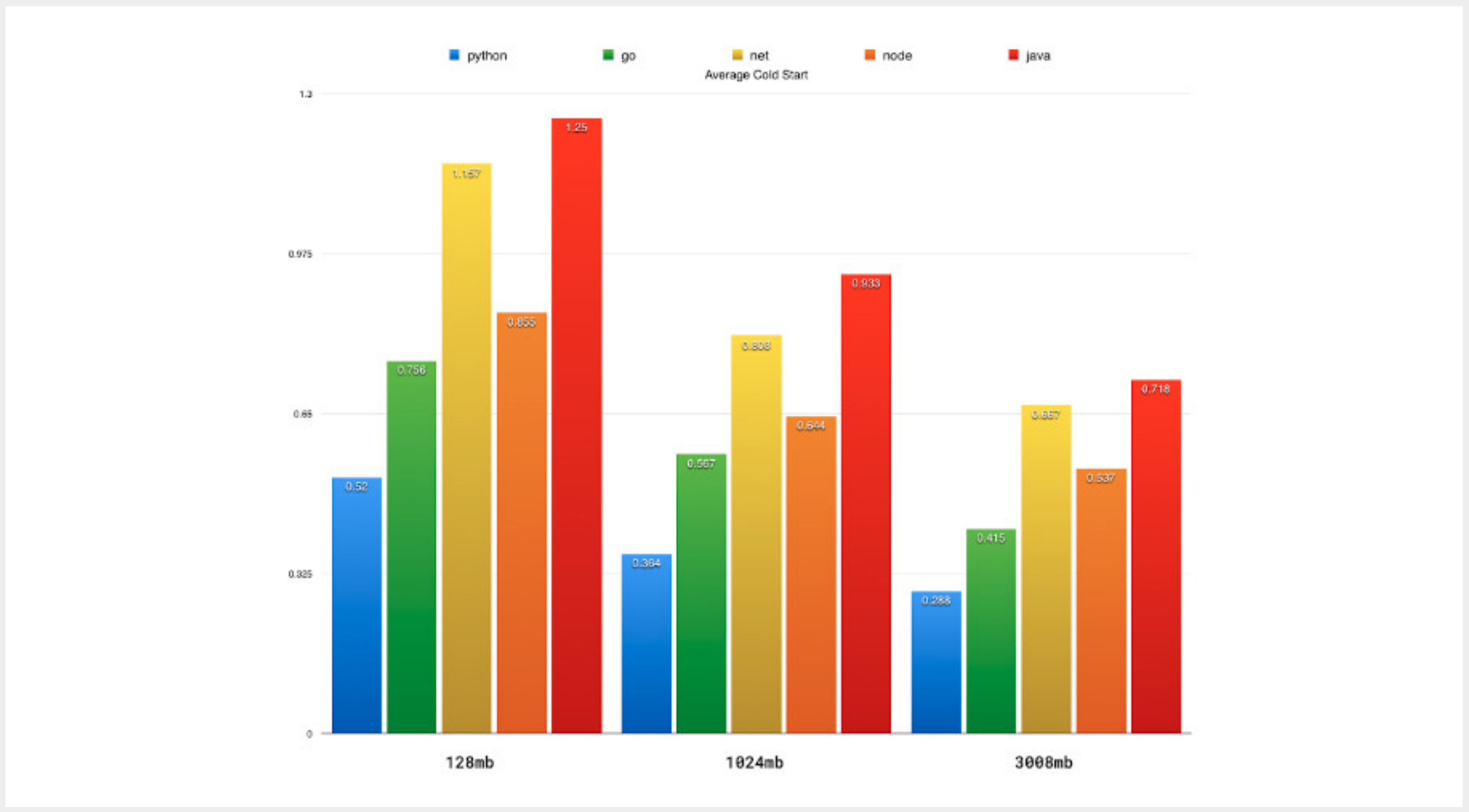
Warm start

AWS **optimization**

Your **optimization**

AWS LAMBDA

COLD START


🟦 python
🟩 go
🟨 net
🟧 node
🟥 java
Average Cold Start
1.3
0.975
0.65
0.325
0
0.52
0.756
1.157
0.855
1.25
0.364
0.567
0.808
0.644
0.933
0.288
0.415
0.657
0.537
0.718
128mb
1024mb
3008mb

SERVERLESS APPLICATION MODEL

- Open-source framework
- Build serverless applications on AWS
- AWS SAM template specification
- AWS SAM command line interface (AWS SAM CLI)

SERVERLESS APPLICATION MODEL

The AWS::Serverless transform, which is a macro hosted by AWS CloudFormation, takes an entire template written in the AWS Serverless Application Model (AWS SAM) syntax and transforms and expands it into a compliant AWS CloudFormation template

SERVERLESS APPLICATION MODEL

BENEFITS

- Single-deployment configuration
- Extension of AWS CloudFormation
- Built-in best practices
- Local debugging and testing

SERVERLESS APPLICATION MODEL

TEMPLATE

- Closely follows the format of an AWS CloudFormation template file
- Transform declaration - `Transform: AWS::Serverless-2016-10-31`
- Globals section - Defines properties that are common to all the resources
- Resources section - Also AWS SAM resources
- Parameters section - Causes `sam deploy --guided` to present additional prompts

SERVERLESS APPLICATION MODEL

SAM RESOURCES

- AWS::Serverless::Function
- AWS::Serverless::Api
- AWS::Serverless::HttpApi
- AWS::Serverless::StateMachine
- AWS::Serverless::SimpleTable

SERVERLESS APPLICATION MODEL

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: >-
  A simple backend (read/write to DynamoDB) with a RESTful A
Parameters:
  TableNameParameter:
    Type: String

Globals:
  #https://github.com/awslabs/serverless-application-model/b
  Function:
    Runtime: python3.6
    MemorySize: 512
    Environment:
      Variables:
        TABLE_NAME:
          Ref: Table

Resources:
  microservicehttpendpointpython3:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: lambda_function.lambda_handler
      CodeUri: .
      Description: >-
        A simple backend (read/write to DynamoDB) with a RES
```

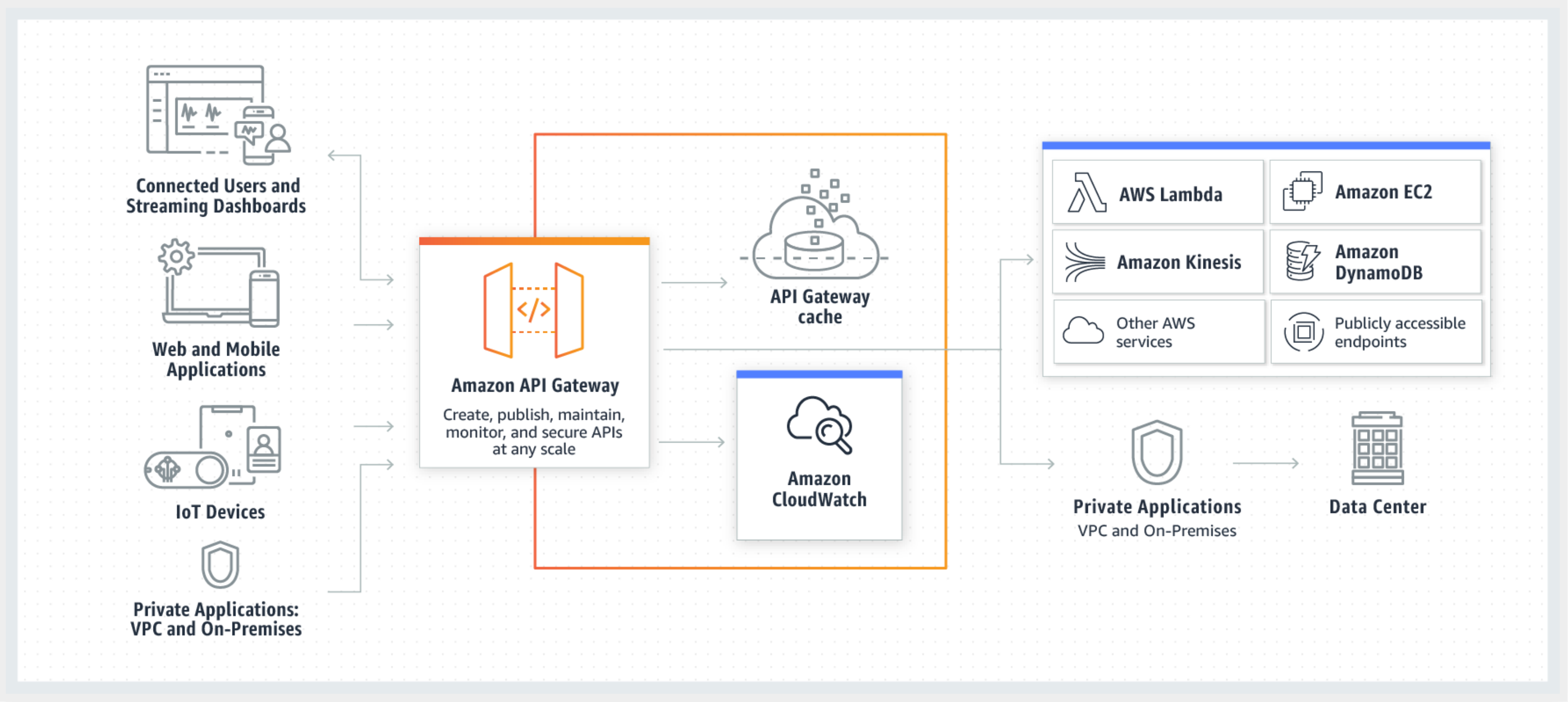
SERVERLESS APPLICATION MODEL

CLI

- sam build - Builds a serverless application
- sam deploy - Deploy a serverless application
- sam init - Initializes a serverless project
- sam local invoke - Invokes a local Lambda function once
- sam local start-api - Runs your serverless application locally
- sam logs - Fetches logs that are generated
- sam validate - Verifies whether an SAM template file is valid

API GATEWAY

- Fully managed service for serving and managing APIs
- HTTP, RESTful and WebSocket APIs
- Frontdoor to application
  - Request proxying too AWS Lambda targets


Connected Users and Streaming Dashboards
Web and Mobile Applications
IoT Devices
Private Applications: VPC and On-Premises
Amazon API Gateway
Create, publish, maintain, monitor, and secure APIs at any scale
API Gateway cache
Amazon CloudWatch
AWS Lambda
Amazon EC2
Amazon Kinesis
Amazon DynamoDB
Other AWS services
Publicly accessible endpoints
Private Applications
VPC and On-Premises
Data Center

API GATEWAY

SAM EXAMPLE

```
...
Resources:
  microservicehttpendpointpython3:
    Type: 'AWS::Serverless::Function'
    Properties:
      ...
      Events:
        Api1:
          Type: Api
          Properties:
            Path: /MyResource
            Method: GET
```

DYNAMODB

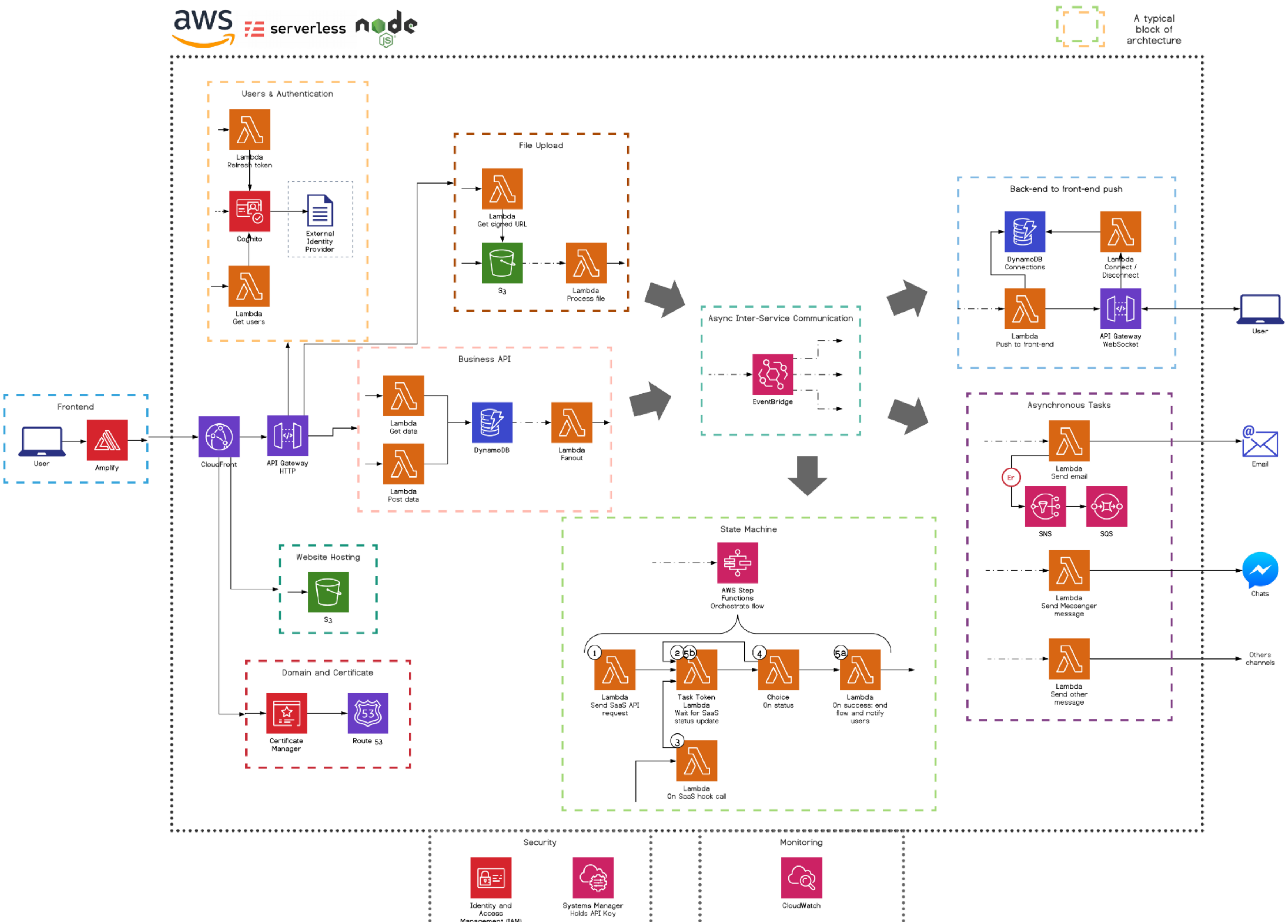
DynamoDB is een fully managed (Serverless) widecolumn non-relational database

- Geoptimaliseerd voor schaal en performance
- Flexibel schema
- JSON en key-value data structuren
- Availability, durability, en scalability built-in

DYNAMODB

- Maak tabellen zonder na te denken over hardware of zelfs DBMS
    - DynamoDB tabellen kunnen elke hoeveelheid data en verkeer verwerken
    - Voorzie tabellen van resources aan de hand van
        - Read Capacity Units (RCU)
        - Write Capacity Units (WCU)

Typical Serverless Architecture



VRAGEN?

HAND-ON OPDRACHT

https://github.com/Quintor/serverless-lambda-workshop/blob/main/assignment/serverless.md