

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

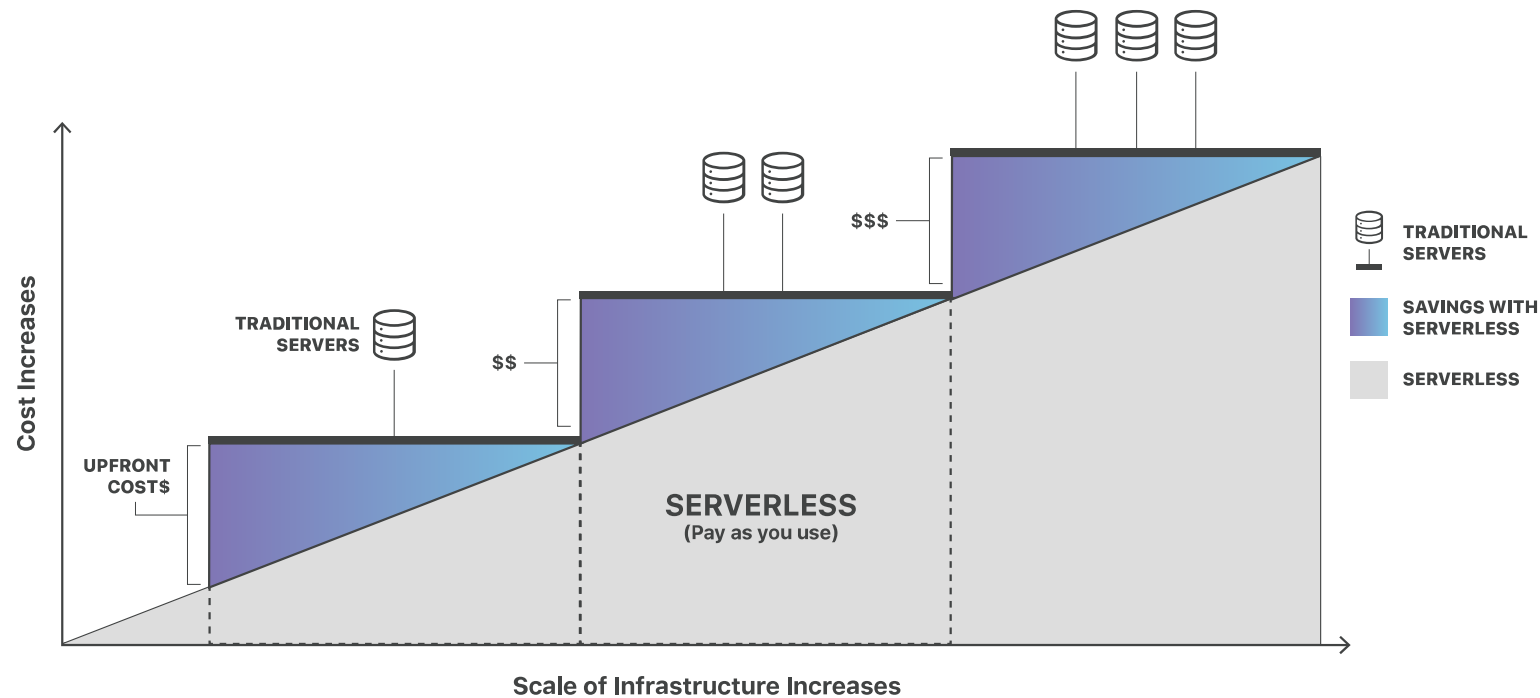
- Vendor lock-in is a risk
- Testing and debugging become more challenging
- Serverless computing introduces new security concerns
- Serverless architectures are often not built for long-running processes
- Performance may be affected



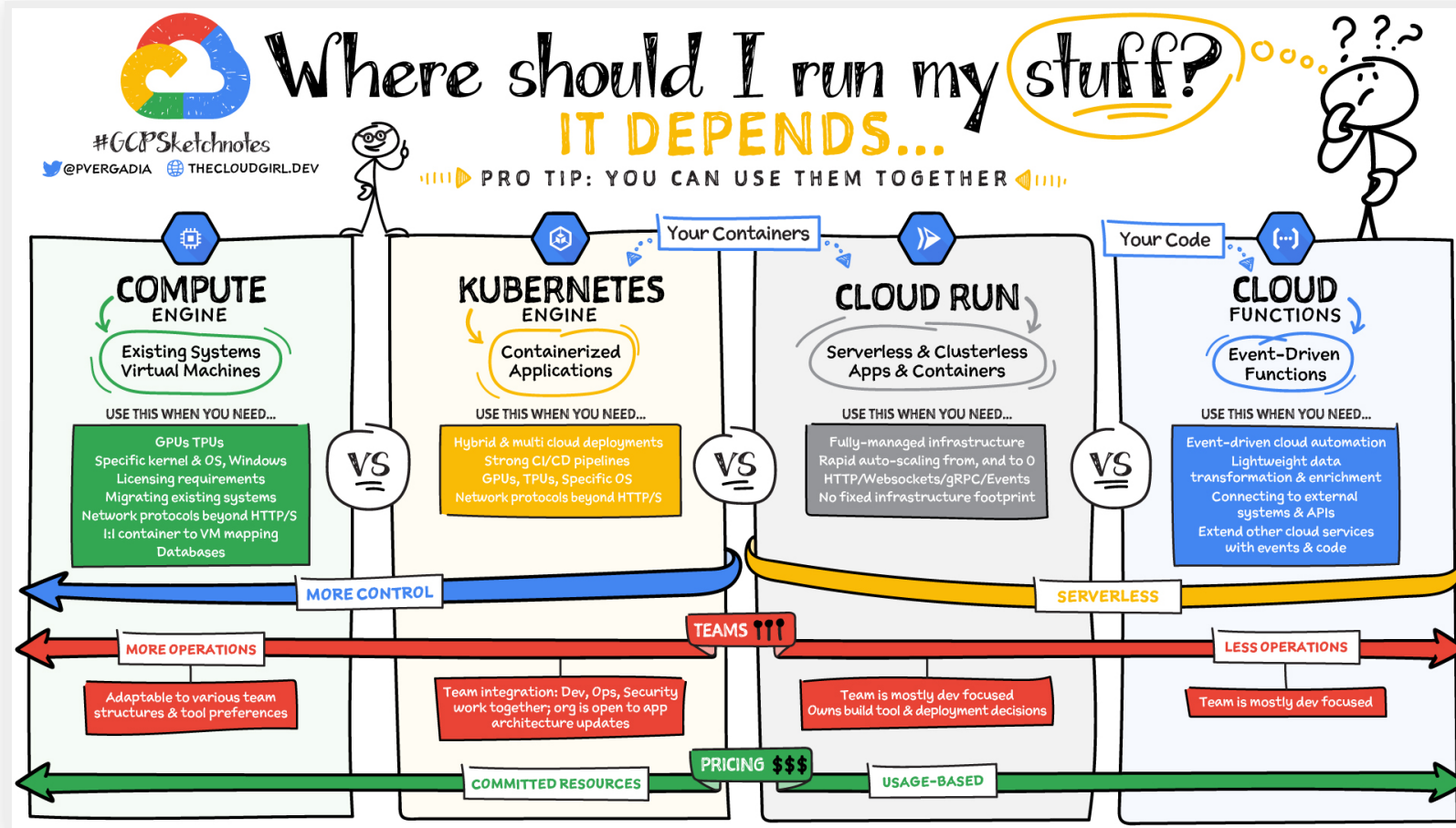
SERVERLESS ARCHITECTURE

COSTS

Cost Benefits of Serverless

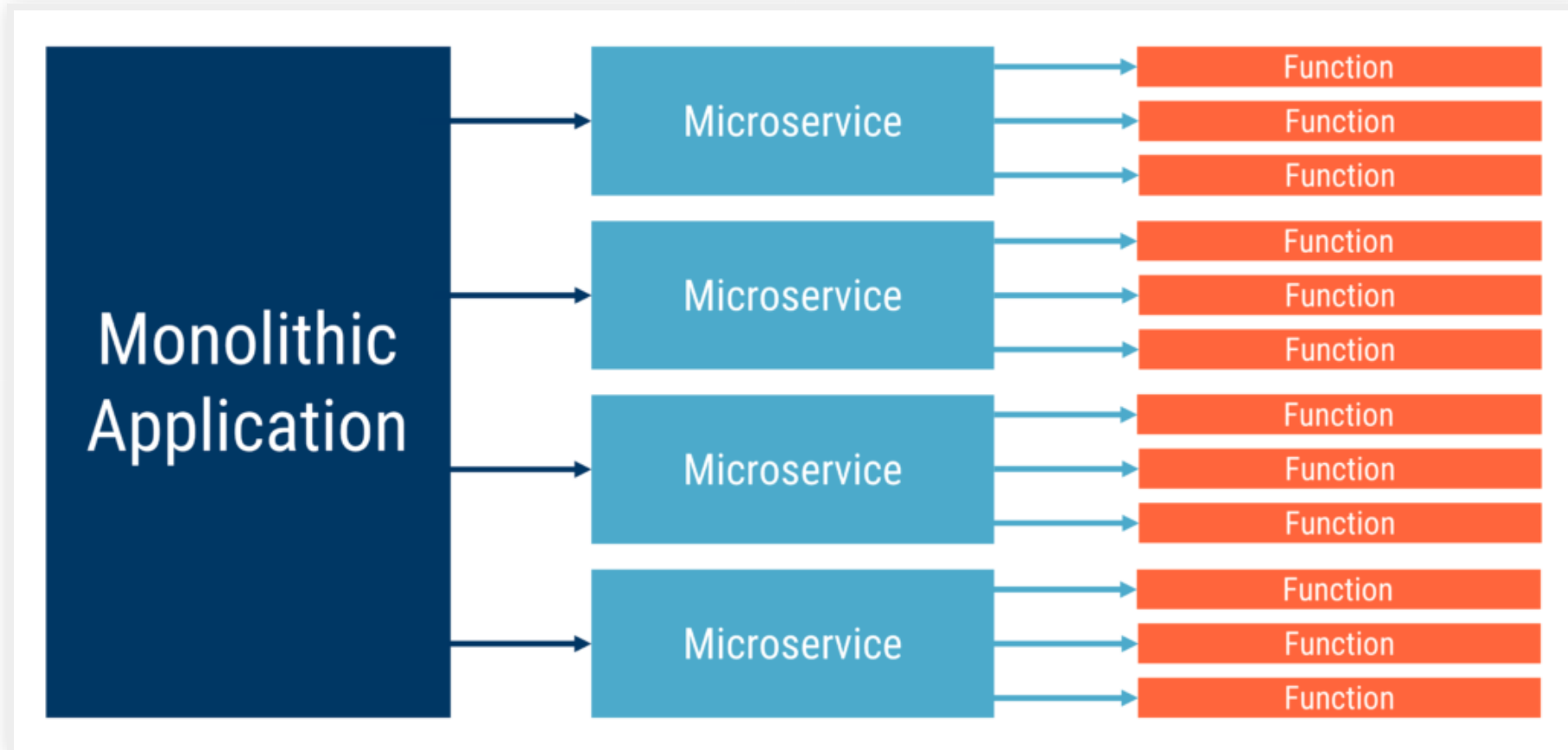


SERVERLESS ARCHITECTURE



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

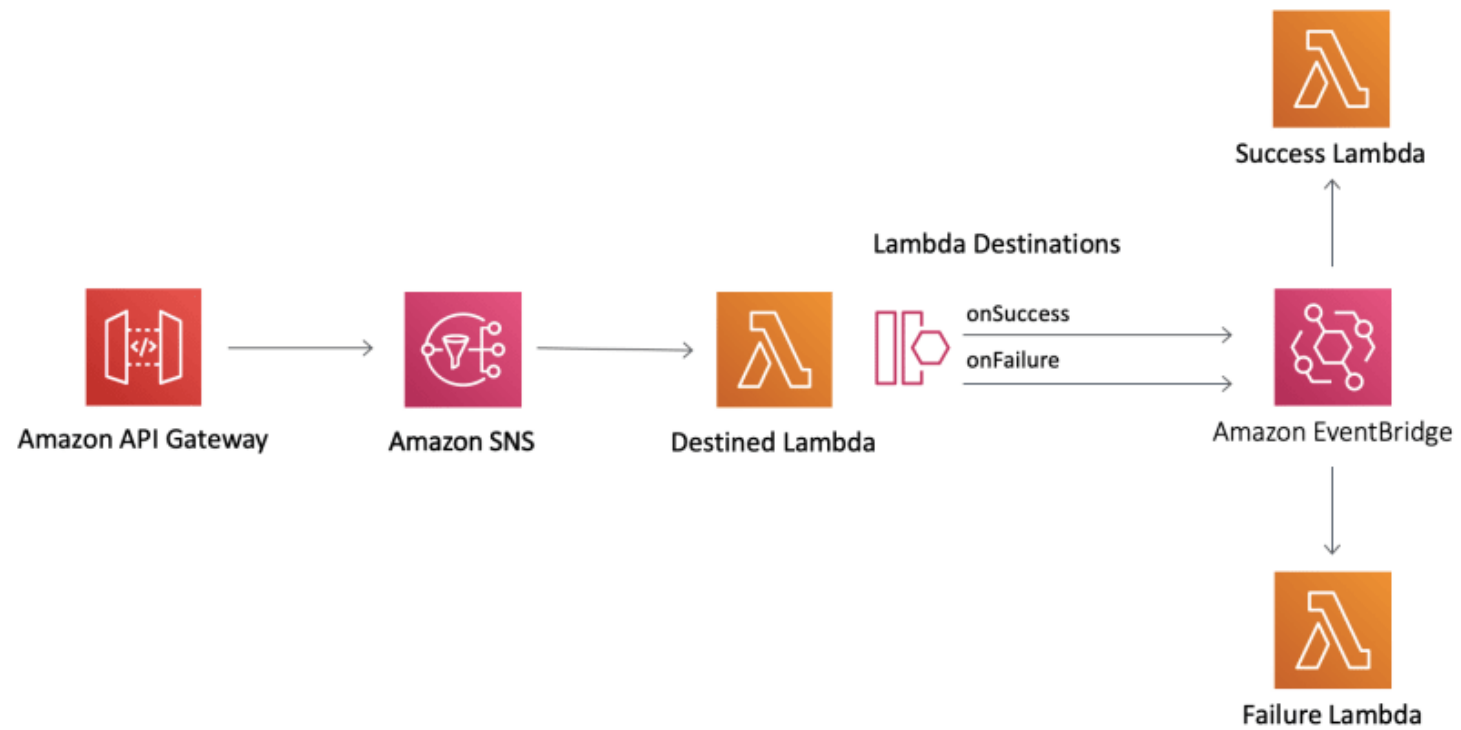
- Runs your code in response to events
- Manages the underlying compute resources
- Create your own back-end services
- Used to extend other AWS services with custom logic
- Runs your code on high-availability compute infrastructure
- Automatic scaling
- Only pay for what you use



AWS LAMBDA

TRIGGERS





AWS LAMBDA

SIMPLE EXAMPLE

Runtime

Node.js 8.10

```
1 console.log('Loading function');
2
3 exports.handler = async (event, context) => {
4     //console.log('Received event:', JSON.stringify(event, null, 2));
5     console.log('value1 =', event.key1);
6     console.log('value2 =', event.key2);
7     console.log('value3 =', event.key3);
8     return event.key1; // Echo back the first key value
9     // throw new Error('Something went wrong');
10 };|
11
```



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
<p>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.</p>	
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

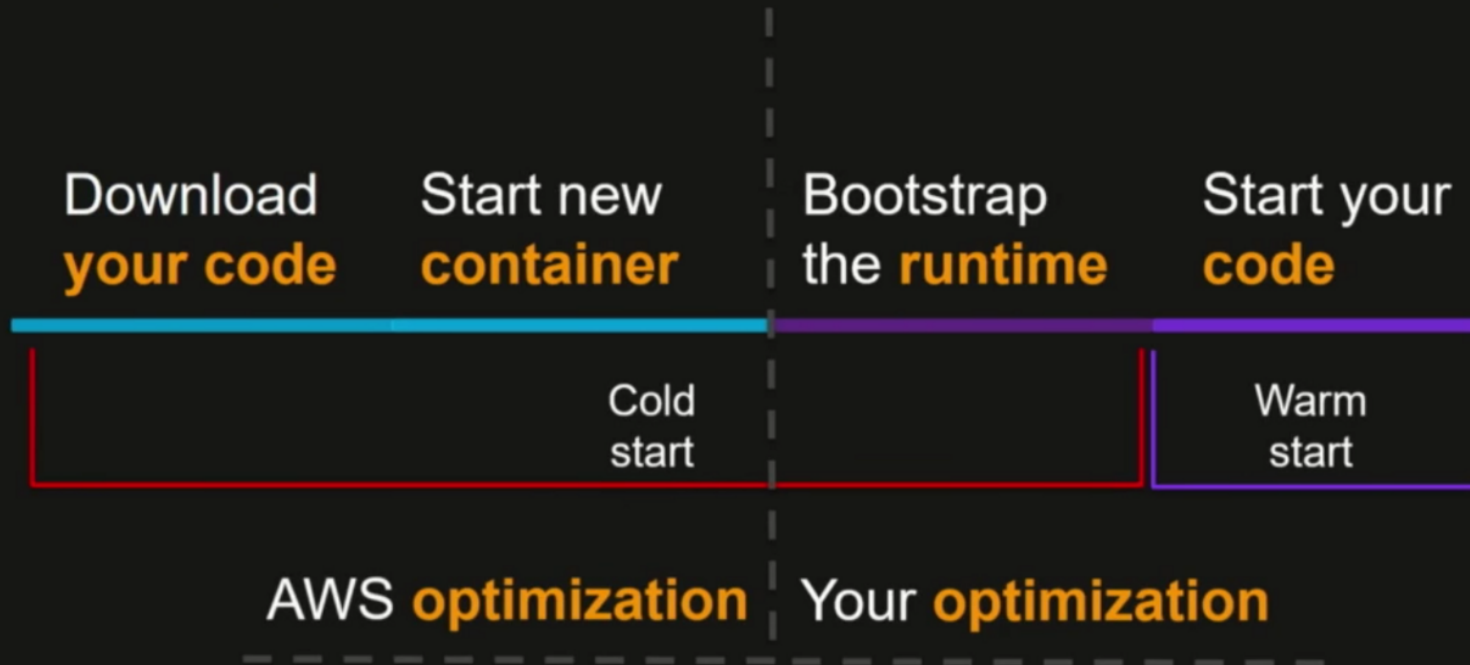


AWS LAMBDA

LIFECYCLE



THE REQUEST LIFECYCLE



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

```
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

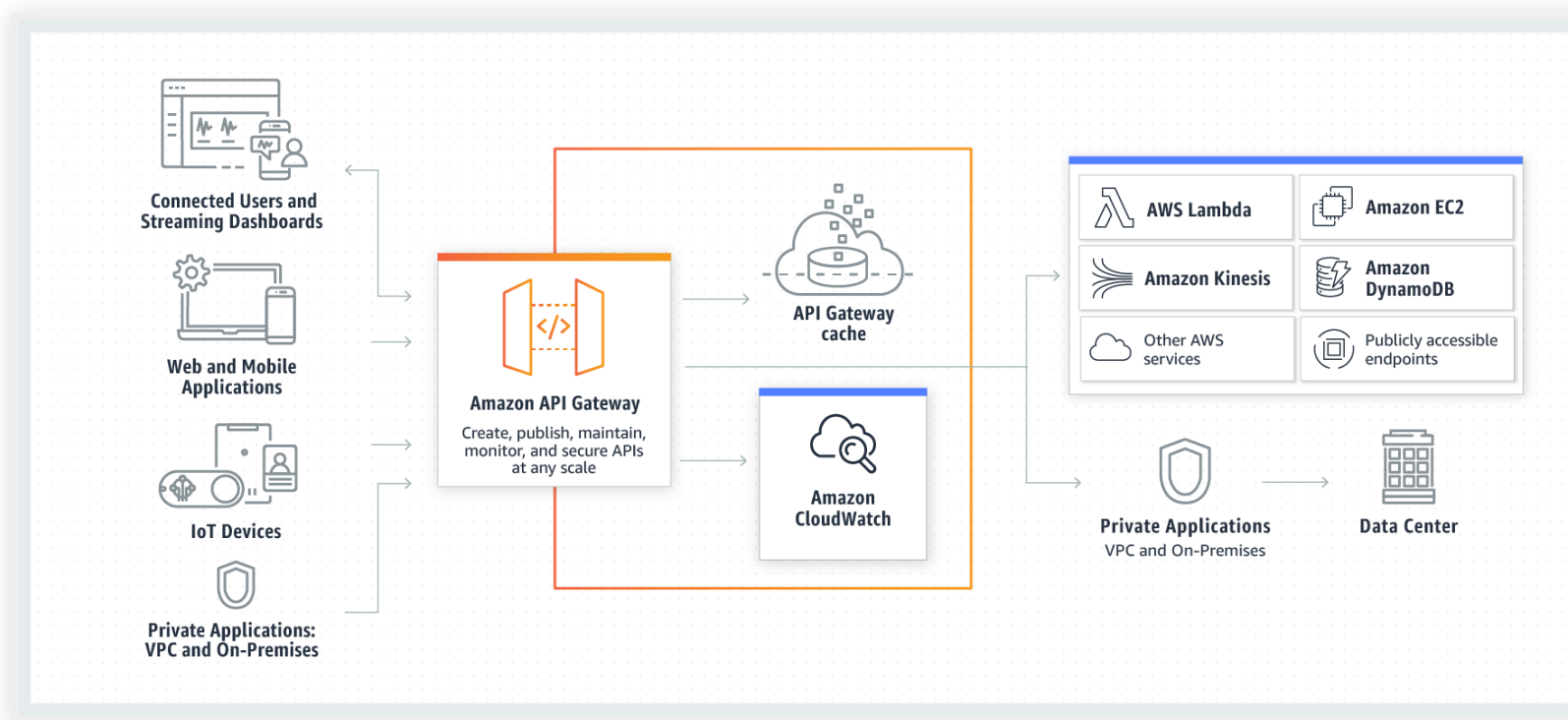
CLI

- `sam init` - Initializes a serverless project
- `sam build` - Builds a serverless application
- `sam local invoke` - Invokes a local Lambda function once
- `sam local start-api` - Runs your serverless application locally
- `sam deploy` - Deploy a serverless application
- `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



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

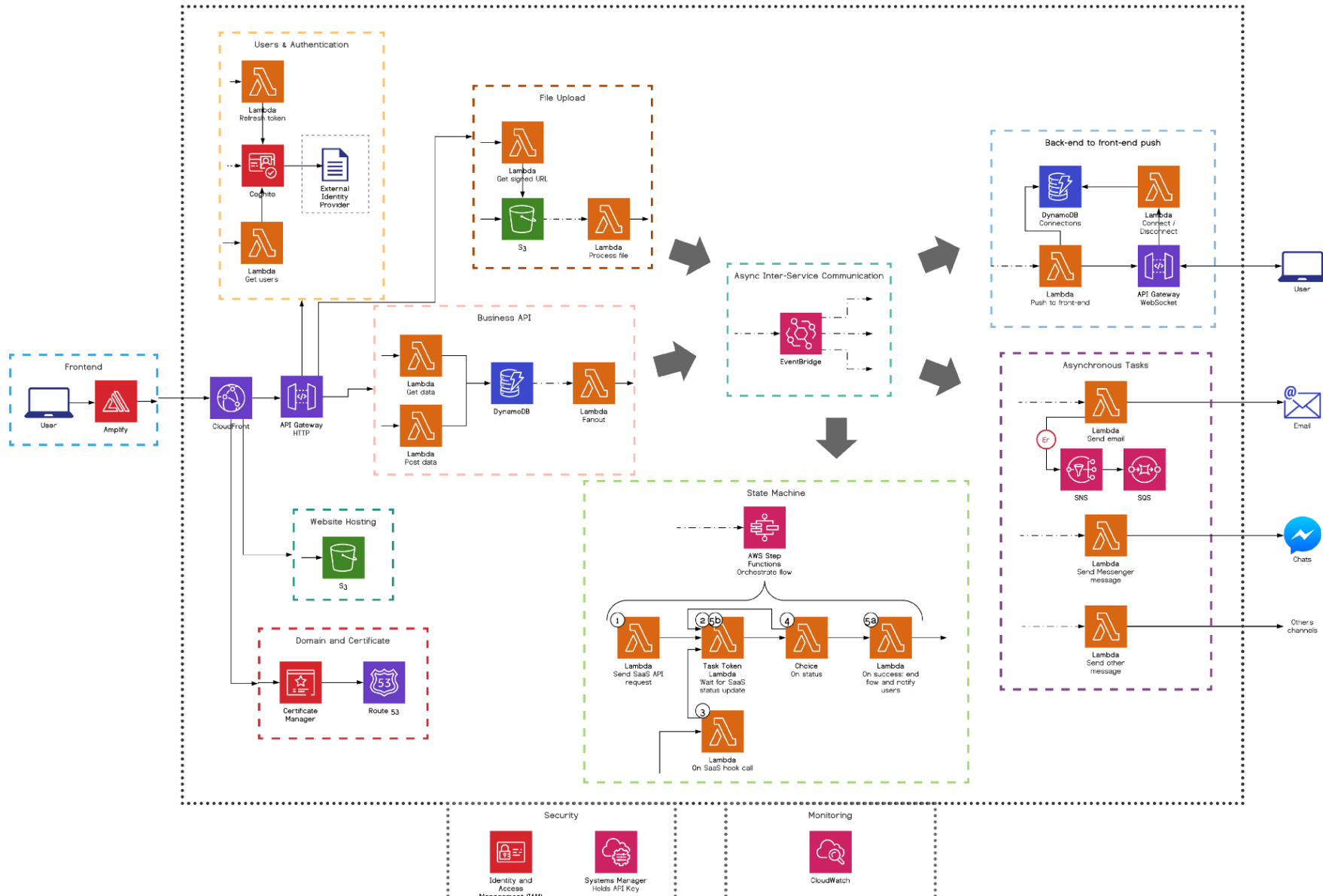
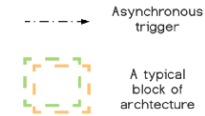


DYNAMODB

- 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>

