Hieu Nguyen

Revolutionizing digital services delivery with Cloud Computing and Serverless

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

10 September 2019

Author(s) Title Number of Pages Date	Hieu Nguyen Revolutionizing digital services delivery with Cloud Computing and Serverless 32 pages + 7 appendices 10 September 2019
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructor(s)	Janne Salonen, Supervisor

The object of this thesis is to introduce Amazon Web Services and Serverlesss Computing, all are relatively new development terms that have been proved to reduce cost, maintenance upkeep, help producing high scalability and increased service delivery speed
The paper first looks into the history of cloud computing, the reason why cloud technology is in high demand in the present day. Then the present day development and competition between many popular cloud providers, and the prediction in market shift for the future.
The goal is archived in two parts: The introduction of the most popular cloud service provider to date Amazon Web Services and the accompanying Serverless framework, And the implementation of a web service that tracks the movement of local public transportation utilizing the many cloud services.
In conclusion, this paper emphasizes the benefits of cloud services and serverless computing in the modern era software development.

Keywords	Amazon Web Services, Serverless framework, Serverless computing, cloud services, cloud computing, Lambda, Cloud Formation, Infrastructure as Code, Software as a Service

Contents

Abbreviations

API	Application programming interface.
AWS	Amazon Web Services.
CF	AWS Cloud Formation.
EC2	Elastic Compute Cloud.
ECS	Elastic Container Service.
FaaS	Function as a Service.
IaaS	Infrastructure as a Service.
IaC	Infrastructure as Code.
IDE	Integrated Development Environment.
IT	Information Technology.
JS	JavaScript.
JSON	Javascript Object Notation.
nvm	Node Version Manager.
PaaS	Platform As A Service.
RAM	Random access memory.
REST	REpresentational State Transfer.
S3	Amazon Simple Storage Service.
SaaS	Software as a Service.
SF	Step Functions.
SLS	Serverless Framework.
VM	Virtual Machine.
VTS	Vehicle Tracking System.

Glossary

Bare metal	Self-hosted on-premise server.
Container	A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.
MQTT	Message Queuing Telemetry Transport is an ISO standard (ISO/IEC PRF 20922) publish-subscribe-based messaging protocol. It works on top of the TCP/IP protocol.
NoSQL	NoSQL stands for Non relational query language and is different from SQL. a NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.

Serverless	Serverless or serverless computing is a relative new model of execution where a cloud provider is responsible for allocating the resources to execute the code and charge only for the resource consumed to fulfill the execution.
Software framework	Software framework is an abstraction boilerplate software that can be selectively modified with additional user-written code, creating application-specific software.
SQL	SQL stands for Structured Query Language, is a domain-specific language used in programming and designed for managing data held in a relational database management system.
YAML	A human-readable data-serialization language commonly used for configuration files.

1 Introduction

Nowadays in the ever-growing modern technological world, internet services and applications are continuously being developed and improved to reach and serve a tremendous amount of users. As the competitions rise between different service providers, whether e-commerce, entertainment or education, a software development solution that compete for quality, delivery time and service cost against competitors is business-critical. With those requirements being put on the table, the nature of using Bare metal are in fact need to be replaced. Provisioning, managing, and upkeep are all time consuming tasks and often require operations people. Cloud based solutions are then introduced to solve the aforementioned disadvantages.

Figure 1: Cloud Computing - internet

Cloud hosting has been available for long, and in the last decade, have evolved into complete cloud solutions [1]. Technologies such as Platform As A Service (PaaS) and micro-containers have been seen as one of the potential solutions to the inconsistent infrastructure environment and server management overhead [2, 1.1]. PaaS is a form of cloud solution that provides a platform to users to run their software without worrying about the underlying infrastructure, while micro-containers are bundled isolated application with its own environment. Together the combination of these two proves as a solid solution that increases availability and reliability, in addition, cut down operating cost and reduce upkeep.

Infrastructure as a Service (IaaS), Software as a Service (SaaS) and Serverless computing are given birth to cater to more specific tasks and practices that have become the de-facto of modern age software development. Improved upon the already advantageous PaaS and containerization, Infrastructure as Code (IaC) compacts the infrastructure configurations to an even simpler, more manageable and versionable file. SaaS provides managed software that can be patched up to create a full solution. Serverless computing is the newest term and is a form of SaaS, which help users focus solely on programming business logic and implementation details of their services without worrying about anything else.

To appeal to the need of the mass for a tool that patches everything together, Serverless framework was born. The framework organizes coded infrastructure configurations and business logic into a solid manageable project. Serverless framework normalizes the differences between many competing cloud providers to provide users with provider agnostic cloud services. In addition, Serverless framework conceptualizes the combinations of many cloud services as a big ”Service”, this in effect speeds up and minimize the production release effort, allowing fluid continuous delivery.

This thesis will look at the biggest Cloud provider of the present-day - AWS, how it has transformed and been the leader of the Cloud solutions industry. Following that, this paper will introduce some of the most important products that make up AWS, and how an open-sourced companion framework have accompanied AWS to the peak of in the software development industry. Lastly, this paper study how a vehicle tracking system utilizes both

AWS and Serverless framework to create a complete cloud service.

2 The history and characteristics of Cloud Computing

2.1 A brief history of Cloud Computing

The term ”Cloud Computing” was popularized by Amazon with the release of Elastic Compute Cloud (EC2) product in 2006, which allow users to rent virtual computers to run their applications The cloud symbol is used to represent the network and the ”cloud” word is a metaphor for the Internet [3]. The metaphor implies cloud that floats around the world and is widely visible to everybody as how cloud services are accessible everywhere and by everyone.

Wikipedia (2012) defines cloud computing concept as follow:

> Cloud computing refers to the delivery of computing and storage capacity as a service to a heterogeneous community of end-recipients. The name comes from the use of clouds as an abstraction for the complex infrastructure it contains in system diagrams. Cloud computing entrusts services with a user’s data, software and computation over a network.

Cloud Computing has been existing since the 2000s, with the introduction of Amazon Web Services subsidiary and its first service EC2. Following the boom of demands for cloud services, big players in the Information Technology (IT) sector start investing and introducing their own cloud, with Google introducing Google App Engine in 2008, Microsoft offering Microsoft Azure in 2010 and IBM with IBM Smart Cloud in 2011.

Amazon Web Services and their competitors have since then been actively introducing new offering and improvements to their existing services.

2.2 The characteristics that popularize Cloud Computing

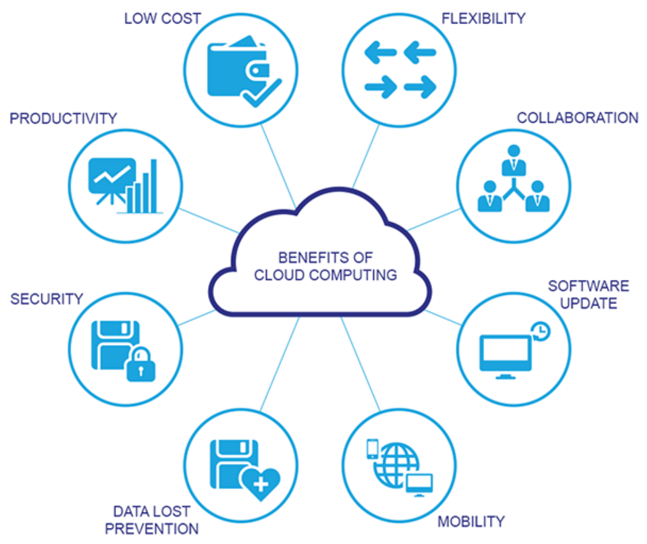


Figure 2: Benefits of Cloud computing - internet

As the key factor and number one advantage of Cloud Computing is cost. Cost is the number one factor that is the number one consideration of a profitable business and is the key selling point of cloud service providers. Cloud computing revolutionized how businesses build infrastructure and software by reducing the need for capital investment and allow companies to access pay for only services that they need on-demand ’pay as you go’ basis [4].

Arguing to be even more important to cost is delivery speed, in which Cloud Computing excels in. In the modern world where there are countless services on the market and the competition is at an all-time high, businesses strive to be the leader in their area by being the most innovative, with the highest quality and the most abundant in users offering before their competitors have a chance to catch up. Not only does delivery speed means having the ability to go to the market earlier, but also have to worry less about infrastructure and maintenance, allowing the software development to be solely directed to serving business priority.

Security is also improved upon traditional on-premise hosting due to centralization of data, increased security-focused resources, due to the devotion of service providers to solving security issue that many customers cannot afford to tackle or which they lack the technical skills to address. However, the complexity of security is greatly increased due to the distribution of data over a large number of data centers. In addition, user access to security audit is difficult and almost impossible. [3].

Contributing to the cost efficiency and high availability is multitenancy, which enables the sharing of resources between users thus allowing for infrastructure centralization in same locations with lower cost - which are mega data centers, and to recycle idle processing power to another user’s needs, improving the efficiency of systems that are often only 10-20% utilized [3] [4]. By combining demand patterns across many business units, the peaks and troughs of computing requirements flatten out. Combined with automation, the gap between peak and average loads are reduced, resulted in massive efficiency and economies of scale in energy use and infrastructure resources. [5]

As a side effect to multitenancy, Cloud Computing is also more environmentally friendly than traditional Bare metal due to minimal e-waste footprint resulted from requiring less physical equipment. [5]

3 The competition and the future

3.1 The modern cloud computing era

Gartner positiones current cloud market providers on a four quadrants figure separated by 2 axis . Each quadrant holds a description of a competitor in the market.

Challengers are well-positioned to serve current market needs. They target a more concentrated set of use cases and have a good track record of successful delivery, however, they are not adapting to market challenges sufficiently quickly or do not have a broad scope of ambition. [6].

Leaders distinguish themselves by offering a variety of services suitable for strategic adoption. They serve a broad range of use cases, although they do not excel in all areas. Leaders have a grant amount of market shares and many referenceable customers. [6].

The niche players may be excellent for a particular use case or in regions in which they operate but ultimately be viewed as specialist cloud providers. They often do not serve a wide range of use cases nor have an ambitious roadmap, and have only developed limited capabilities in cloud IaaS. [6]

Visionaries are ambitious and are making tremendous investments in the development of unique technology. Their services are emerging, and have many capabilities in de-

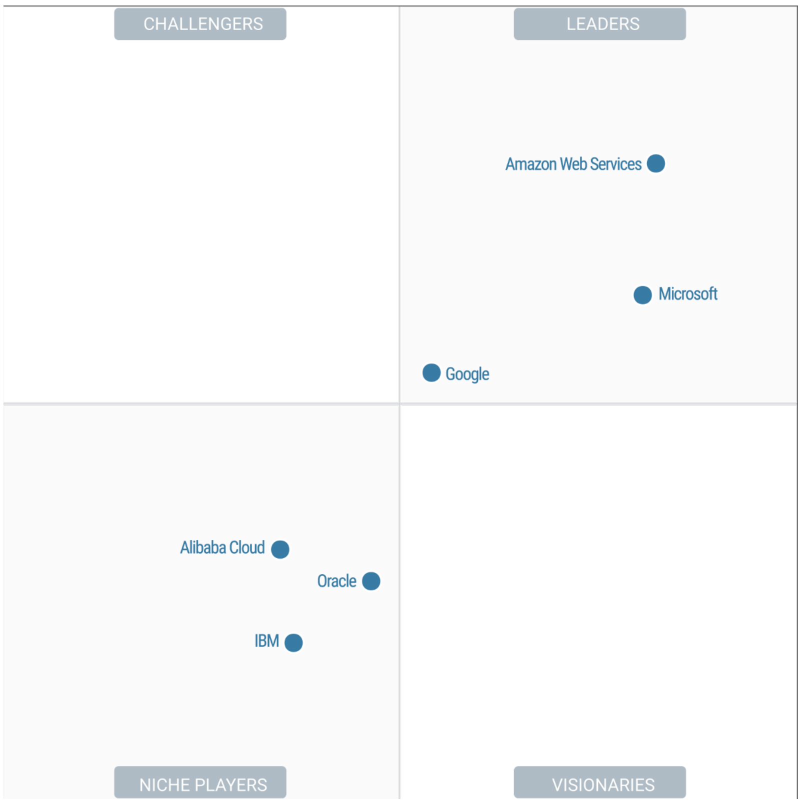
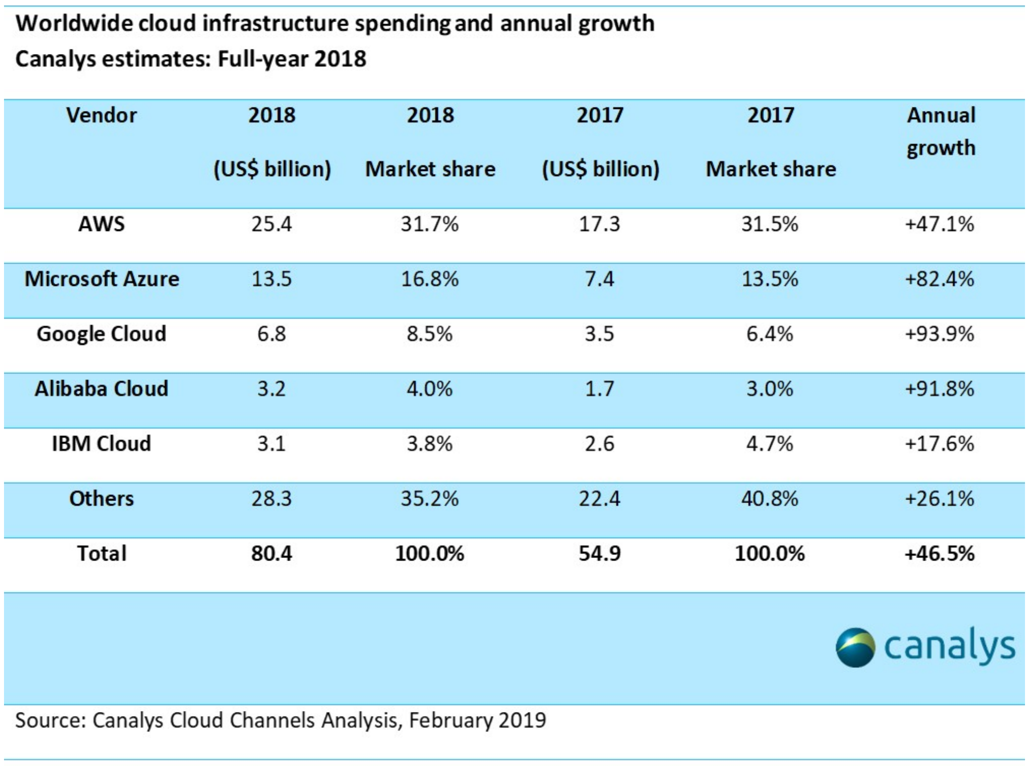


Figure 3: Magic Quadrant for Cloud Infrastructure as a Service, Worldwide - Gartner (July 2019)

velopment that are not yet generally available. While they may have a good amount of customers, they may not serve a broad range of use cases. [6]



Worldwide cloud infrastructure spending and annual growth
Canalys estimates: Full-year 2018

Vendor	2018 (US$ billion)	2018 Market share	2017 (US$ billion)	2017 Market share	Annual growth
AWS	25.4	31.7%	17.3	31.5%	+47.1%
Microsoft Azure	13.5	16.8%	7.4	13.5%	+82.4%
Google Cloud	6.8	8.5%	3.5	6.4%	+93.9%
Alibaba Cloud	3.2	4.0%	1.7	3.0%	+91.8%
IBM Cloud	3.1	3.8%	2.6	4.7%	+17.6%
Others	28.3	35.2%	22.4	40.8%	+26.1%
Total	**80.4**	**100.0%**	**54.9**	**100.0%**	**+46.5%**

canalys

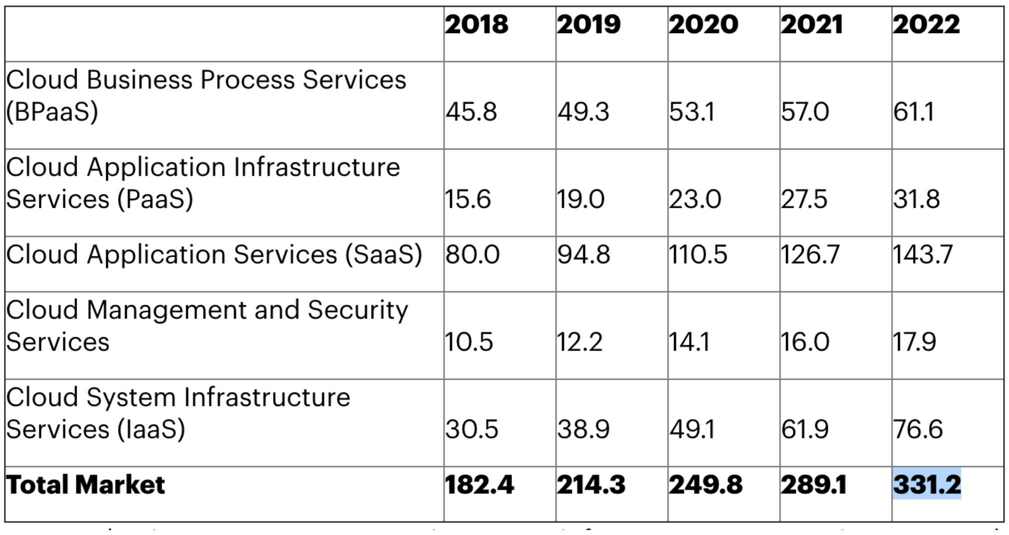
Source: Canalys Cloud Channels Analysis, February 2019

Figure 4: Worldwide Cloud infrastructure spending and annual growth full-year 2018 - Canalys cloud channels analysis, Ferbuary 2019

According to Canalys: ”Cloud Infrastructure spend grow 46% in Q4 2018 to exceed US$80 billion for full-year”. This makes Cloud Computing the most important sector in IT industry, not only by the growing rate but also the expanding size. [7]. To the present date, there are currently five biggest cloud provider in lead of the cloud computing market sector. From the highest market share down, the names are AWS, Microsoft Azure, Google Cloud, Alibaba Cloud, and IBM Cloud.

AWS is unsurprisingly is in the highest demand, due to its proven track record of successful services such as Lambda or EC2, remaining the dominant cloud service provider with 31.7% market share. The followup trio Microsoft Azure, Google Cloud, and Alibaba Cloud, despite having marginally smaller share in the market, are taking lead in annual growth with close to double in market share comparing to 2017. This market shift is the direct result of the introduction of channel partners in cloud services, in particular, understanding customer requirements, simplifying integration, deployment, and billing, as well as multi-cloud management [7]. In addition, channel partners building services on top of existing cloud services also extends the integration funnels to cloud service provider.

3.2 The future of cloud computing



	2018	2019	2020	2021	2022
Cloud Business Process Services (BPaaS)	45.8	49.3	53.1	57.0	61.1
Cloud Application Infrastructure Services (PaaS)	15.6	19.0	23.0	27.5	31.8
Cloud Application Services (SaaS)	80.0	94.8	110.5	126.7	143.7
Cloud Management and Security Services	10.5	12.2	14.1	16.0	17.9
Cloud System Infrastructure Services (IaaS)	30.5	38.9	49.1	61.9	76.6
Total Market	**182.4**	**214.3**	**249.8**	**289.1**	**331.2**

Figure 5: Worldwide Public Cloud Service Revenue Forecast (Billions of U.S. Dollars) - Gartner, April 2019

The market for cloud computing is maturing at a tremendous growth rate. The fastest-growing market segment will be cloud system infrastructure service or IaaS, the second highest will be PaaS followed up by SaaS. [8] Gartner projects revenue in the cloud IaaS to increase to $331.2 billion by 2022. However, most of interests and revenues are currently directed in two providers: AWS and Microsoft Azure. The market views them as general-purpose providers capable of serving a broad range of use cases with high availability and efficiency. Google is making steady progress in term of adoptions, but it remains in a distant third place in terms of overall annual revenue and interest. All other vendors are forced to focus on regional dominance of niche workloads, given the momentum of AWS and Azure, and the scale at which they operate. [6]

Cloud computing will continue to dominate and become mainstream within most organizations due to its various benefits. According to recent Gartner surveys, one out of every three organizations sees cloud investments as their top 3 priority. Gartner forecasts by the end of 2019, more than 30% of service providers will shift from cloud-first to cloud-only solution, this drops the usage of licensed software while giving rise to SaaS and subscription-based cloud consumption model. [8]

Out of all current cloud market leaders, AWS will continue to reign for a long era of cloud computing to come. AWS's lead on Microsoft's Azure is bigger than it seems, thanks to the signing of deals in both size and quantities, with the fact that AWS has always been first and ahead in the competition from the early days. For AWS it's a paradigm of innovation, where everything is achievable. [9]. Overall, from the development perspective, AWS is in a good position, with 140 services available and counting in production across multiple regions in 2019, spanning a wide range including computing, storage, networking, database, analytics, application services, deployment, management, mobile, developer tools, and tools for the Internet of Things. Ultimately AWS will continue to stay to be the leader of all market leaders in terms of Cloud Solutions.

4 Amazon Web Services

Out of the many above mentioned competitors, this paper will look at some of the offerings from AWS. The reason why AWS is chosen is due to its sheer number of services and highly successful operating track records. The services being introduced are oriented around AWS's main service domains: Infrastructure as Code, Software as a Service with Serverless computing, and cloud storage. These crucial components are important in forming a basic digital web service that the thesis will discuss later on.

4.1 Infrastructure as Code with AWS Cloud Formation

Infrastructure as Code (IaC) has emerged as a best practice for automating the provisioning of infrastructure services. Infrastructure management is a process associated with software engineering, and is a crucial and difficult element in many system. Traditionally organizations rely on dedicated system administrators to manually provision technical resources as well as maintaining upkeep. The manual processes have many disadvantages: [10]

- Error prone due to inconsistency, leading to derivations from the configuration standard.
- Added operation cost because of human capital requirements that could otherwise be directed to other business needs
- Lack of agility in deployment and maintenance, difficult to upgrade, limiting the

speed to market of the products

IaC bring to the table the solutions to these problems by incorporating automation and consistency into the process. Rather than provisioning through manual procedures operated by only system administrators, IaC promotes the use of configuration files that can be instantiated by also developers with full confidence of the consistency between deployments. This practice also solves widespread scaling issues for many large organizational systems that were only affecting large enterprises. IaC treats these configuration files as software codes, to produce many components that comprise an operating system such as storage, compute, network configurations and application services.

Further on, Cloud computing capitalizes virtualization to enable the provisioning of on-demand cloud services that are combined into working infrastructures. AWS Cloud Formation (CF) is AWS's approach to IaC. CF gives system administrators and developers an easy and manageable method to create, manage, provision and upgrade a collections of related AWS resources in an orderly and predictable way. [10] CF takes in Javascript Object Notation (JSON) or YAML template to orchestrate the included AWS services, their dependencies and any associated runtime parameters. CF ensures the consistency of provisions between repeated deployment of the same template. CF also versions each variation of the templates deployed, allowing flexible upgrades and rollbacks in a predictable way. And as template files are essentially code, they can be versioned alongside application code. CF itself is a SaaS therefore the already deployed state file will be managed on AWS servers.

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "Dynamo DB Table that will store user information",
    "Resources": {
        "UserTable": {
            "Type": "AWS::DynamoDB::Table",
            "DeletionPolicy": "Delete",
            "Properties": {
                "BillingMode": "PAY_PER_REQUEST",
                "KeySchema": [
                    {
                        "AttributeName": "userId",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "email",
                        "KeyType": "RANGE"
                    }
                ],
                "AttributeDefinitions": [
                    {
                        "AttributeName": "userId",
                        "AttributeType": "S"
                    },
                    {
                        "AttributeName": "email",
                        "AttributeType": "S"
                    },
                    {
                        "AttributeName": "name",
                        "AttributeType": "S"
                    }
                ]
            }
        }
    }
}
```

Listing 1: A sample CF template file written in JSON to create a Amazon Simple Storage Service (S3) storage bucket

Cloud Formation utilizes the terms ”stack” and ”resources” to make responsibility division, resource organization and applying constraints easier. ”Stack” is a group of ”resources”, and ”resources” can be a service configuration, runtime parameter, or a nested ”stack”.

Despite the advantages, CF also has down sides. Scaling CF requires good partitioning between have more stacks and having more resources on the same stack because of the service limits setup by AWS [11]. At the time of writing, the commonly run-into limits are:

- Stack limit is at default 200. This is a soft limit and can be lifted through contacting AWS customer support
- Resources limit in a single stack is at 200. This is a hard limit and cannot be raised through any mean. This constraint is designed to improve deployment performance, and to promote good resource organizations

In practice, there have been user reports of common problems with templates containing numerous resources being in a dead-lock state where an upgrade to the resource fails and the rollback of the failed upgrade also fails. However AWS support team acknowledges this problem and have been solving them diligently on special support request tickets.

4.2 Software as a Service and Serverless Computing

4.2.1 Lambda

Introduced in November 2014, AWS Lambda is one of the most crucial Function as a Service (FaaS) offerings from AWS, and is arguably the service that defines and popularizes the term Serverless. Lambda is a computing service that allows users to run application code without provisioning and managing servers, executing only when needed and support infinite horizontal scaling. Service cost is based on runtime duration with the minimum as small as 100 milliseconds. The service was designed to be able to handle a tremendous amount of concurrent request from different sources up to 1000 concurrent execution. This is possible because AWS Lambda runs code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring and logging. [12] When using Lambda, users will be responsible for only application code. AWS Lambda manages a compute fleet that offers a balance of memory, CPU, network, and other resources. System administrators will now only need to focus on managing the core services such as database performance and load balancers while developers can focus solely on application code running on the platform. Deployment to Lambda is as simple as uploading a zip file containing application code through AWS Lambda web console.

Each concurrent request to Lambda results in the initiation of a new Lambda container. This results in cold starts. A cold start occurs when a Lambda function is invoked after not being used in an extended period of time resulting in increased application startup latency. Cold start happens to the first execution of a Lambda invocation because all the dependencies are loaded. Each concurrent request to invoke Lambda will result in Lambda spinning up a new Lambda container leading to the invocation going through cold start again. Subsequent requests to the same Lambda container will not suffer from cold start latency.

AWS lambda supports securely running native Linux executables via calling out from a supported runtime such as Node.js, Python, Java and many more. Each language has an impact on Lambda runtime and especially cold start. Go and Python are observed to have lower cold start due to smaller dependencies size comparing to Node.js or Java. [13]

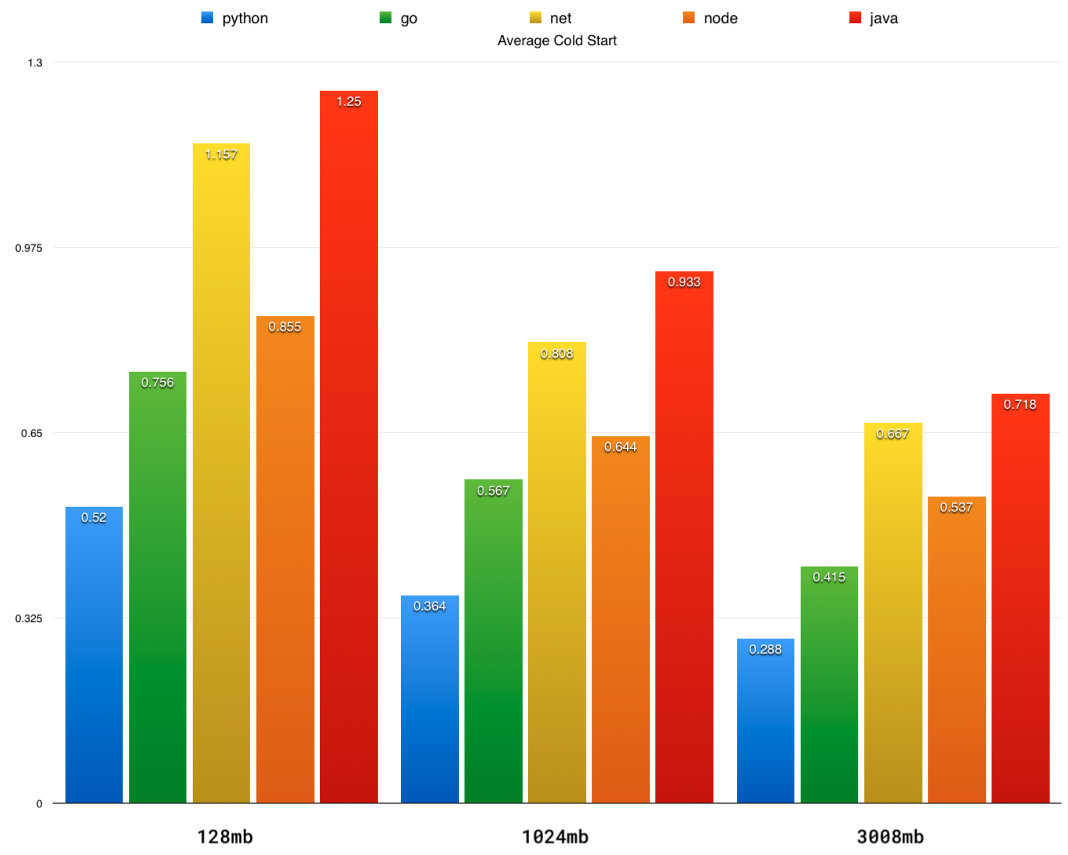


Figure 6: AWS Lambda cold start benchmark with different language runtime - Nathan Malishev

When a Lambda container finish executing a request, it goes into idle state where it keeps on running and wait for further request. If not receiving any new request, the container will be recycle for another Lambda function. The exact algorithm behind container recycle is not officially documented. Any further request will result in either a new container or an existing one.

Lambda is the glue that binds many AWS services together due to its event driven nature. It can be invoked from an Application programming interface (API) request, a trigger to database or a cron job executor to automate processes. This proves Lambda to be the most important building block of AWS that extends functionality of sibling services.

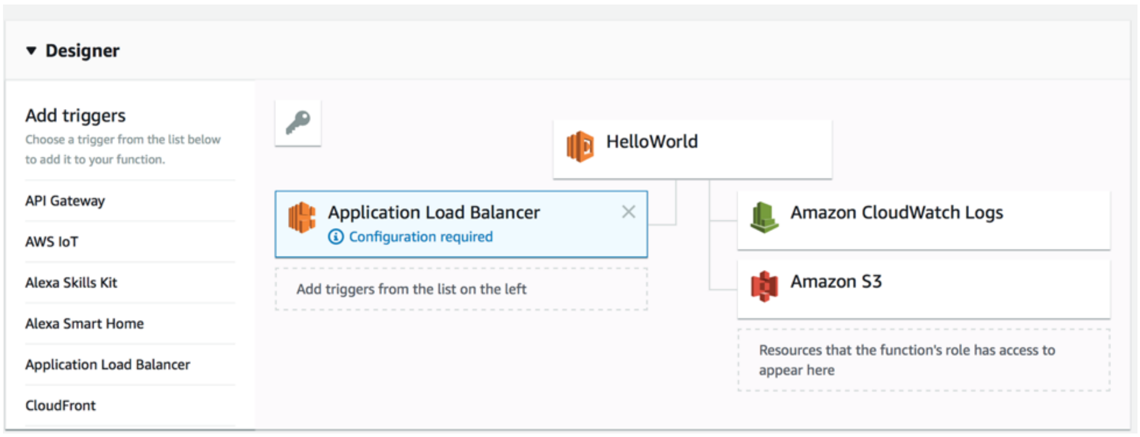

Designer
Add triggers
Choose a trigger from the list below to add it to your function.
API Gateway
AWS IoT
Alexa Skills Kit
Alexa Smart Home
Application Load Balancer
CloudFront
HelloWorld
Application Load Balancer
Configuration required
Add triggers from the list on the left
Amazon CloudWatch Logs
Amazon S3
Resources that the function's role has access to appear here

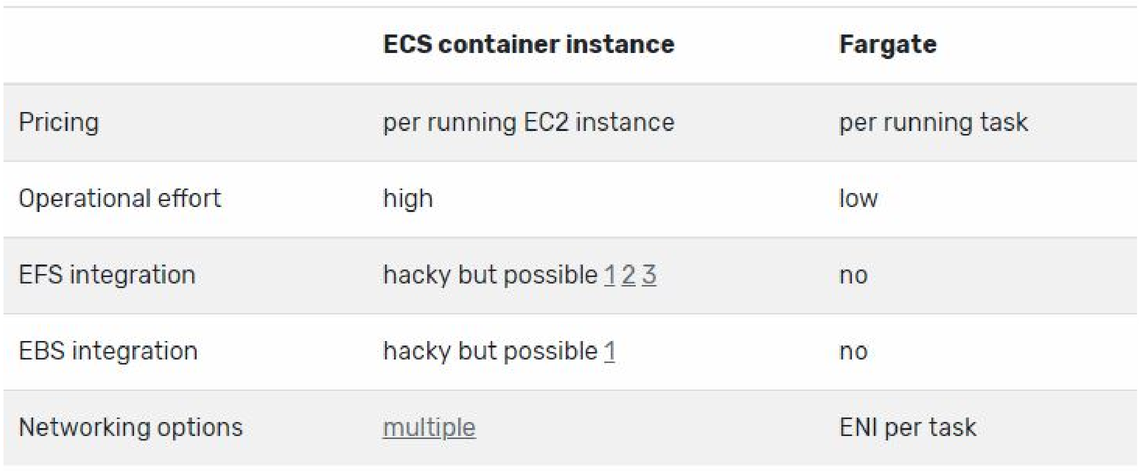
Figure 7: AWS Lambda console and many possible service triggers that can be plugged into the function

In present day, there are many competitors of AWS Lambda, such as Google Functions, Azure Functions or Apache OpenWhisk that all offer similar features as AWS Lambda. Despite competition, Lambda remains the highest in demand and will continue to be the most popular Function as a Service offering.

4.2.2 ECS and AWS Fargate

Since the open-source of Docker Container in 2013, Container development has become the new standard for software development, revolutionize software delivery. A container is a unit of software bundle that shares an operating system installed on the server and contain all configuration for an application and their dependencies. Containers run as resource-isolated processes ensuring quick, reliable, and consistent deployments, regardless of environment.

AWS ECS is a highly scalable container orchestration service for containerized applications built with Docker. ECS discard the need to install and operate own orchestration software, manage and scale a cluster of Virtual Machine (VM)s, or schedule containers on those VMs. ECS features AWS Fargate, which is a compute engine service introduced in late 2017 that enable an user to run containers on AWS inside managed VM without provisioning infrastructure. This results in seamless application scaling. Because the underlying infrastructure is managed by AWS, infrastructure security and upkeep overhead is no longer a concern. Users are recommended to embed security reinforcement in the containers. AWS Fargate charges only during the runtime of the containers inside the underlying VM, and not the total time that the VM is running. In combination, the most economic method is to have a cluster of running Fargate instances managed by ECS.



	ECS container instance	Fargate
Pricing	per running EC2 instance	per running task
Operational effort	high	low
EFS integration	hacky but possible 1 2 3	no
EBS integration	hacky but possible 1	no
Networking options	multiple	ENI per task

Figure 8: AWS ECS and Fargate pricing scheme as of 2019

4.2.3 AWS Step Functions

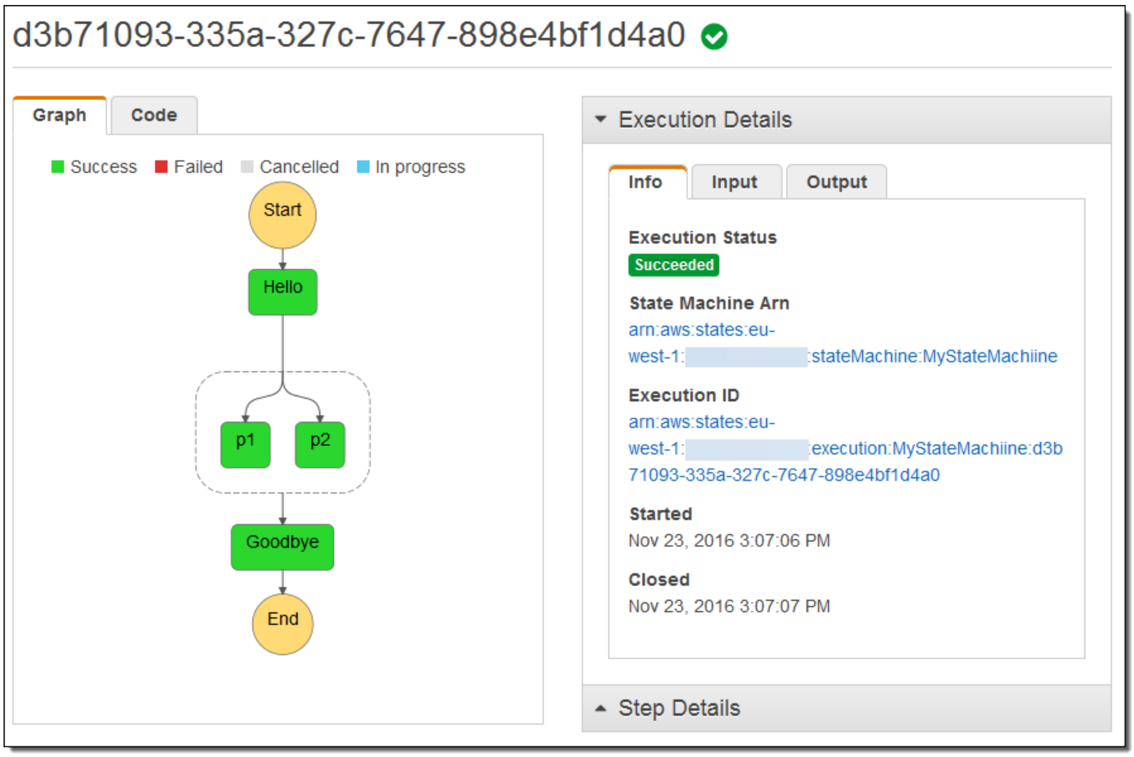
Introduced in December 2016, AWS Step Functions (SF) is a coordination service for components of distributed applications and microservices using visual workflows. Using SF you can design and run workflows that stitch together multiple services such as AWS Lambda or ECS. Workflows in SF are made up of serially connected steps, where each step result is input to the next step, enabling continuation and fluid execution.

In SF term, workflows are called state machine, and every step of the workflow is a state. Users use State Language, which is a JSON-based, structured language to define the state machines in a collection of states with transitions between states, error handling, conditional retries and more. After defining the state machine, users can start execution with different input. These executions are isolated and use a copy of the currently active state machine, allowing effortless updates to the state machine definition. All executions started after the update to the state machine will be using a new definition, while all existing executions will continue on the older version of the definition.

```json
{
  "Comment": "An example of the Amazon States Language.",
  "StartAt": "FirstState",
  "States": {
    "FirstState": {
      "Type": "Task",
      "Resource":
          "arn:aws:lambda:us-east-1:123456789012:function:FUNCTION_NAME",
      "Next": "ChoiceState"
    },
    "ChoiceState": {
      "Type" : "Choice",
      "Choices": [
        {
          "Variable": "$.foo",
          "NumericEquals": 1,
          "Next": "FirstMatchState"
        }
      ],
      "Default": "DefaultState"
    },
    "FirstMatchState": {
      "Type" : "Task",
      "Resource":
          "arn:aws:lambda:us-east-1:123456789012:function:OnFirstMatch",
      "Next": "NextState"
    },
    "DefaultState": {
      "Type": "Fail",
      "Error": "DefaultStateError",
      "Cause": "No Matches!"
    },
    "NextState": {
      "Type": "Task",
      "Resource":
          "arn:aws:lambda:us-east-1:123456789012:function:FUNCTION_NAME",
      "End": true
    }
  }
}
```

Listing 2: A sample SF definition file

SF provides a rich web console interface to visualize executions of the state machine. They are represented as a flowchart with color-coded steps to indicate the execution status of each step. Green for success, yellow for on-going and red for failures.


d3b71093-335a-327c-7647-898e4bf1d4a0
Graph
Code
🟩 Success 🟥 Failed ▨ Cancelled 🟦 In progress
Start
Hello
p1
p2
Goodbye
End
Execution Details
Info
Input
Output
Step Details

Execution Status
Succeeded

State Machine Arn
arn:aws:states:eu-west-1:[redacted]:stateMachine:MyStateMachiine

Execution ID
arn:aws:states:eu-west-1:[redacted]:execution:MyStateMachiine:d3b71093-335a-327c-7647-898e4bf1d4a0

Started
Nov 23, 2016 3:07:06 PM

Closed
Nov 23, 2016 3:07:07 PM

Figure 9: A sample SF execution visualization

4.3 Cloud Storages

4.3.1 DynamoDB

AWS DynamoDB is a key-value storage and NoSQL database with high performance and scalability. Amazon features DynamoDB with consistent single-digit millisecond query duration making data available almost instantaneously. DynamoDB is a fully managed, multi-region database with built-in security and automated backup capabilities. Provisioning is no longer a concern thanks to scaling automation that increase or decrease capacity to maintain performance.

DynamoDB supports three basic data model units: Tables, Items and Attributes. Tables are collections of Items, and Items are collections of Attributes. Each attribute is of one of the supported data types:

- Primitive: Number, String, Binary, Boolean and Null
- Multi-valued: String Set, Number Set, and Binary Set

- Document: List and Map

Hash key is the designated property that uniquely identifies the item, and is used for data partitioning. The number of partitions is the bigger number between the number of partitions for throughput and number of partitions for size, which is calculated using the following formulas: [14]

$$Numberofpartitionsforthroughput = \frac{RCUreads}{3000RCU} + \frac{WCUwrites}{1000WCU} \quad (1)$$

$$Numberofpartitionsforsize = \frac{TablesizeinGigabytes}{10Gigabytes} \quad (2)$$

In which RCU stands for Read Capacity Units measured in 4Kb/sec, WCU stands for Write Capacity Units measured in 1Kb/sec.

4.3.2 Amazon Elasticache and Redis

Caching is the process of storing frequently accessed data in a temporary data store, in contrary to a permanent data store. This allows faster data queries, reducing load and cost to permanent storage layer, resulting in enhanced service performance and availability. Release in August 2011, AWS Elasticache is a Caching-as-a-Service that enhance caching solutions with scalability and easy of management. It offers cost-effective caching solutions while reducing the complexity associated with deployment and distribution management. In Elasticache there are two important concepts: Node and Cluster. Elasticache nodes are the smallest service building block and generally are network-attached Random access memory (RAM)s. Clusters are logical collections of Nodes.

Elasticache offers features that boost reliability for critical production deployments that include:

- Automatic detection and recovery from cache node failures
- Flexible Availability Zone placement of nodes and clusters

- Integration with other Amazon Web Services

Users can setup Elasticache to use either Redis or Memcached data store as engines. Each engine has its own perks that serve different use cases.

Memcached is a simple volatile cache server. It allows storing key/value pairs where the value is limited to being a string up to 1MB. Users can retrieve values by their key at extremely high speed, often saturating available network or even memory bandwidth.

Highly regarded as the most popular caching solution, Redis is a fast, open-sourced in-memory data store. Comparing to Memcached, Redis is better equipped with advanced features that serve a wider range of applications such as complex data types and publish/subscribe model. According to Redis own official definition:

> ”Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes with radius queries and streams.”

Typically clusters of Memcached nodes are used when users need to run large nodes with multiple cores or threads but with only a basic set of functions. Whereas Redis cluster consist of a single node with a larger feature set to serve more complex use cases.

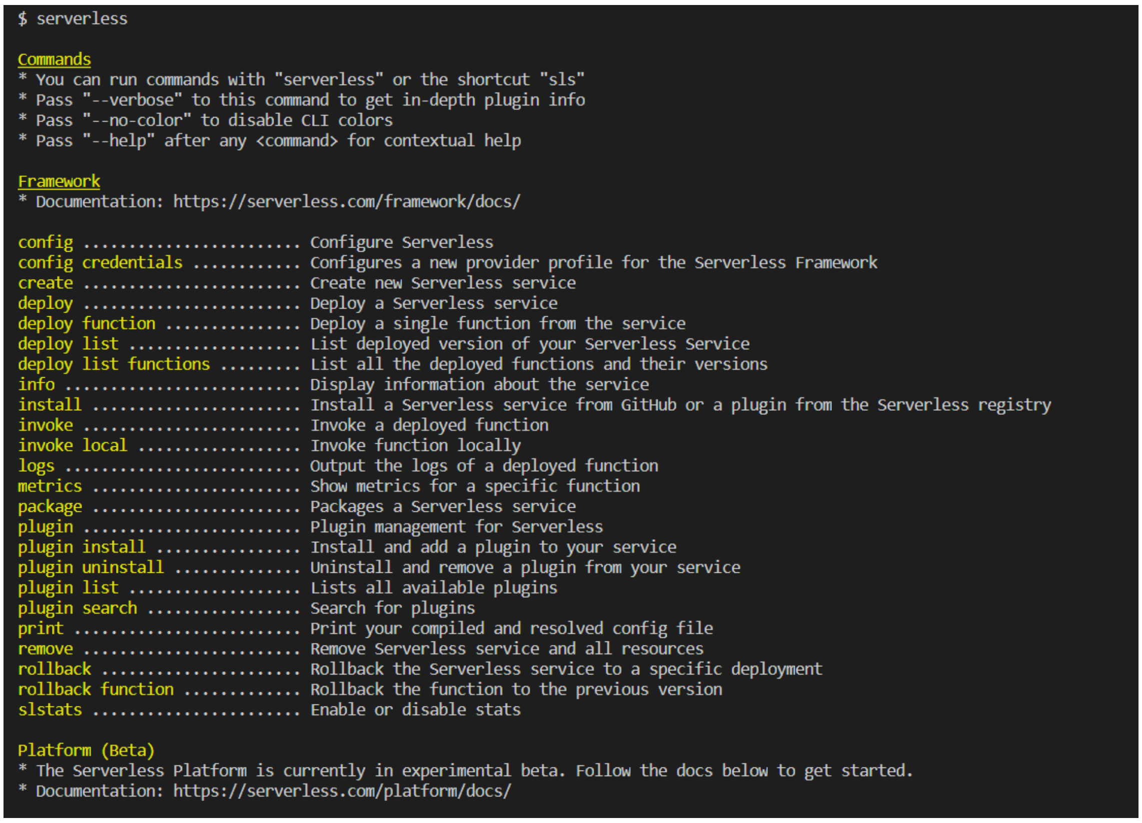
5 Serverless framework

Figure 10: Serverless Framework Logo

Serverless Framework (SLS) is a free and open-source Software framework written in JavaScript (JS) that simplify the process of building, managing and deploying of a serverless application. Earlier version of SLS before version 0.5.6 target exclusively AWS platform, and after the major release of version 1.0.0, SLS framework received a complete revamp, developing toward the abstraction between many service providers, allowing users to create seamless multi-provider cloud applications. Users are able to stitch services between AWS, Microsoft Azure and Google Cloud using SLS service description files.

FaaS functions are the main building blocks of SLS. These functions are services such as AWS Lambda or Google Function. ”Resource” is the term used to abstract cloud provider infrastructure components such as AWS DynamoDB Table, AWS Step Functions or AWS Simple Notification Service, that will conditionally invoke defined functions. These invocation conditions are termed as triggering ”events”. Events can practically be from any supporting provider services. All functions, public events, private events are defined in SLS own IaaS implementation, ultimately defining higher-level concept ”service” - which is a SLS unit of organization, or a project. This offers structure, automation, and built-in best practices, allowing users to focus on building sophisticated, event-driven, serverless architectures.

Supporting and extending the features of SLS is the comprehensive supporting plugin system. SLS is entirely made up of plugins. The plugins themselves can also be extended with plugins, resulting in high horizontal scalability. SLS plugins are extending a common interface, ensuring the compatibility between many plugins. Out of the box, there are many plugins for orchestrating standard deployment process, single function/event deployment, and local function invocation.



```
$ serverless

Commands
* You can run commands with "serverless" or the shortcut "sls"
* Pass "--verbose" to this command to get in-depth plugin info
* Pass "--no-color" to disable CLI colors
* Pass "--help" after any <command> for contextual help

Framework
* Documentation: https://serverless.com/framework/docs/

config ........................ Configure Serverless
config credentials ............ Configures a new provider profile for the Serverless Framework
create ........................ Create new Serverless service
deploy ........................ Deploy a Serverless service
deploy function ............... Deploy a single function from the service
deploy list ................... List deployed version of your Serverless Service
deploy list functions ......... List all the deployed functions and their versions
info .......................... Display information about the service
install ....................... Install a Serverless service from GitHub or a plugin from the Serverless registry
invoke ........................ Invoke a deployed function
invoke local .................. Invoke function locally
logs .......................... Output the logs of a deployed function
metrics ....................... Show metrics for a specific function
package ....................... Packages a Serverless service
plugin ........................ Plugin management for Serverless
plugin install ................ Install and add a plugin to your service
plugin uninstall .............. Uninstall and remove a plugin from your service
plugin list ................... Lists all available plugins
plugin search ................. Search for plugins
print ......................... Print your compiled and resolved config file
remove ........................ Remove Serverless service and all resources
rollback ...................... Rollback the Serverless service to a specific deployment
rollback function ............. Rollback the function to the previous version
slstats ....................... Enable or disable stats

Platform (Beta)
* The Serverless Platform is currently in experimental beta. Follow the docs below to get started.
* Documentation: https://serverless.com/platform/docs/
```

Figure 11: Serverless command line interface screenshot. Each command is a separate plugin

6 Implementation - Vehicle Tracking System

6.1 Project overview

As bought up in the introduction of this thesis, a Vehicle Tracking System (VTS) is implemented to demonstrate the structure of an application built upon note-worthy service offering from AWS with the help of Serverless Framework to simplify development and deployment process.

VTS is an on-demand tracking system that allows both Finnish organization and users to track the movement of public transit vehicles operated by Helsinki Regional Transport Authority - ”Helsingin seudun liikenne” (HSL) in Finnish. Making use of the numerous tracking chips and sensors planted on each vehicle, HSL collected data such as geographic position, velocity, directions, and time tables from each vehicle in real-time and expose their data for public consumption. This work is under the Finnish Open Data and

Smart City movement to develop Helsinki to the city of tech and future.

HSL opens API with web sockets that stream from documented data channel to subscribed feeds. VTS will improve upon this API to introduce how a serverless application is implemented.

6.2 Project objective

The objective of the project is to use Lambda as the main computation service while using API Gateway as the interface to the service. and DynamoDB as the main storage solutions. To streamline the development process, all IaaS code will be managed, verified and deployed with Serverless Framework.

In details, all below points will be achieved:

- Implement a Lambda that user can request tracking of certain vehicle
- Implement a Lambda that retrieves the status of a tracked vehicle such as current location, velocity, heading direction...
- Setup a DynamoDB storage to store the currently active tracked vehicles' statuses
- Implement a long-running Lambda that listens to HSL vehicle status data feeds and updates DynamoDB records
- A Representational state transfer (REST) API as the main communication interface between users and the application

6.3 Implementation methods

6.3.1 Prerequisite

Before the project work is started, there are requirements to fulfill before being able to develop the VTS system. Requirements and chosen tools are as follow:

- A working Integrated Development Environment (IDE) - Visual Studio Code is chosen by writer

- NodeJS v10 installed on console terminal - instructions below
- A AWS account on Free Tier - instructions below

Installing NodeJS on MacOS or Linux

To make NodeJS installation easier, Node Version Manager (nvm) is used. To install nvm, use the install script included in the module using wget (internet required):

```
        wget -qO-
            https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh
            | bash
```

After the script runs successfully, close and re-open the terminal for nvm to work. On the new terminal, run the following commands:

```
        nvm install 10 # this install Node version 10 on nvm
        nvm alias default 10 # this defaults NodeJS to version 10
```

After running the above commands, run `node -v` should output the current node version (v10.12.0 at the time of writing). This indicates successful installation

Setting up AWS account

To create an AWS account:

1. Open https://aws.amazon.com/, and then choose Create an AWS Account.
2. Follow the online instructions.

To setup programmatic AWS Access Keys for console terminal usage:

1. Login to your AWS account and go to the Identity & Access Management (IAM) page.
2. Click on Users and then Add user. Enter vehicle-tracker as Name. Enable Programmatic access by clicking the checkbox. Click Next to go through to the Permissions page. Click on Attach existing policies directly. Search for and select AdministratorAccess then click Next: Review. Check to make sure everything looks good and

click Create user. After this step the credential keys are shown. These keys are required in the next step

3. Export the credentials as environment variables as follow. Repeat this steps every time after the terminal is re-opened

```
        export AWS_ACCESS_KEY_ID=<key-from-previous-step>
        export AWS_SECRET_ACCESS_KEY=<your-secret-key-here>
```

Now the console terminal is ready for development and application deployment to AWS.

6.3.2 Project components

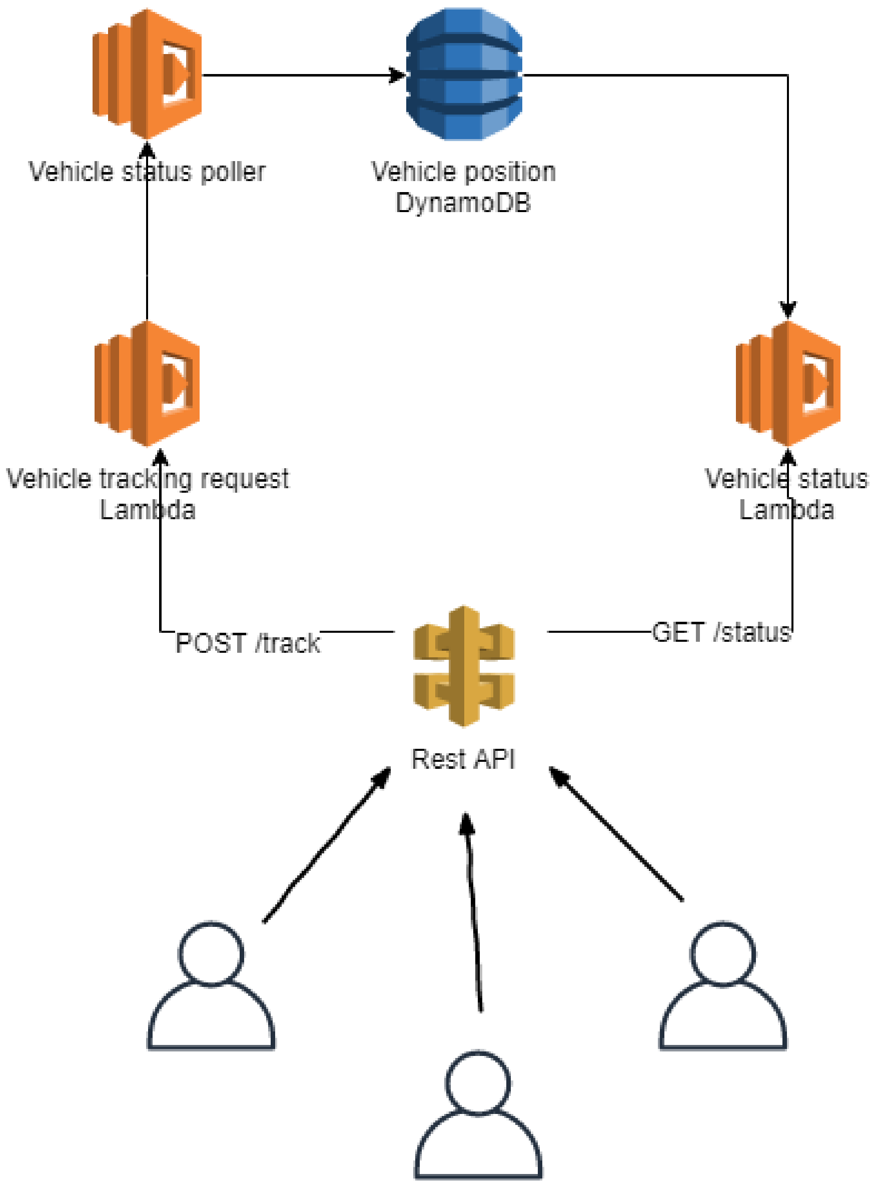


Figure 12: Project architecutre overview

Serverless project configuration file

Serverless is powered with a provider agnostic configuration file. If deployed to AWS this file will be translated into CloudFormation IaC input file.

Firstly the project needs to be initialized with global configuration. This configuration file instructs SLS on which provider to deploy and how to deploy the Lambda functions. AWS is the provider of choice, and Lamda will be deployed running on NodeJS version 10.x to AWS Western European region 1 - which is Ireland. `iamRoleStatements` defines the permissions for the Lambdas: allowing access to all other AWS services with unrestricted actions. In more sophisticated projects, it is recommended to apply more fine-tune access control for Lambda.

```
provider:
  name: aws
  runtime: nodejs10.x
  stage: dev
  region: eu-west-1
  iamRoleStatements:
    - Effect: "Allow"
      Action:
        - "*"
      Resource: "*"
```

The function configurations follow. Following the specification in Project Objectives, three Lambdas will be made: vehicle-tracker-request, vehicle-status-poller, vehicle-status-retrieve. vehicle-tracker-request Lambda will be responsible for verifying API request, and to start the execution of vehicle-status-poller, which is a background running Lambda that will listen to vehicle status data feed from HSL, and save to Database. Lastly, vehicle-status-retrieve will retrieve the status of currently tracked vehicle. As visualized in the project specification overview, vehicle-tracker-request and vehicle-status-retrieve will be connected with API Gateway as handlers for two REpresentational State Transfer (REST) endpoint requests: `POST /track` and `GET /status` respectively. Serverless simplify these API Gateway endpoint configuration as ”event”. vehicle-status-poller serve as a background task processor, thus has no API front. With timeout configuration at 600 seconds, it is ensured that the vehicle will be tracked for sufficient duration of time.

```
functions:
  vehicle-tracker-request:
      events:
        - http:
            path: track
            method: post

    vehicle-status-poller:
      timeout: 600

    vehicle-status-retrieve:
      events:
        - http:
            path: status
            method: get
            request:
              parameters:
                paths:
                  route: true
```

DynamoDB storage and SSM secret management

DynamoDB is chosen as storage for vehicle status due to its simplicity and lighting fast operations.

```
VehicleStatusTable:
  Type: AWS::DynamoDB::Table
  Properties:
    BillingMode: PAY_PER_REQUEST
    KeySchema:
      - AttributeName: route
        KeyType: HASH
      - AttributeName: vehicleNumber
        KeyType: RANGE
    AttributeDefinitions:
      - AttributeName: route
        AttributeType: S
      - AttributeName: vehicleNumber
        AttributeType: S
```

There are two BillingMode options: PROVISIONED and PAY_PER_REQUEST. PROVISIONED billing is the default option and is recommended if the number of database requests is predictable. PAY_PER_REQUEST is used should the workloads are unpredictable. To increase query performance, auto-indexing will apply to two specified properties: route and vehicleNumber. In combination, HASH and RANGE key will become an unique identifier for the database record. Both properties are strings, defined in AttributeTypes: S - which is short for String.

VehicleStatusTable name and VehicleStatusTable Amazon Resource Name (ARN) is stored securely on AWS System Manager service encrypted key-value Parameter Store. These parameters will, later on, be used to prevent private credential and configuration leakage in the source code.

```
VehicleStatusTableParameter:
  Type: AWS::SSM::Parameter
  Properties:
    Name: /applications/vehicle-tracker/vehicle-status-table-name
    Type: String
    Value: !Ref VehicleStatusTable
    Description: Vehicle status table name
VehicleStatusTableArnParameter:
  Type: AWS::SSM::Parameter
  Properties:
    Name: /applications/vehicle-tracker/vehicle-status-table-arn
    Type: String
    Value: !GetAtt VehicleStatusTable.Arn
    Description: Vehicle status table ARN
```

These secret can then be retrieved simply with:

```
ssm
    .getParameters({
      Names:
          ['/applications/vehicle-tracker/vehicle-status-table-name']
    })
```

Lambda to request a tracking of certain HSL vehicle route

vehicle-tracker-request serves as the reception to the service. It will verify requests made by users to prevent malformed or supported service request parameters, in addition, safeguard the service from Distributed Denial-of-Service Attack by limiting tracking request for the same vehicle.

```
if (!event.route) {
    throw new Error("Unspecified route ID");
} else if (event.mode && !["train", "bus",
    "tram"].includes(event.mode)) {
    throw new Error(`Invalid mode ${event.mode}, type --help`);
}
```

```

    const existingRecords = await getAllItemsForRoute(event.route);

    if (existingRecords.length > 0) {
        return {
            statusCode: 400,
            body: {
                ok: false,
                reason: `Another polled is ran for ${event.mode}
                    ${event.route}`
            }
        };
    }
```

If all safety conditions are fulfilled, vehicle-tracker-request will invoke vehicle-status-poller to start polling for request vehicle status in the background.

```
    await Lambda.invoke({
      FunctionName: POLLER_FUNCTION_ARN,
      Payload: JSON.stringify({
        route: event.route,
        mode: event.mode
      }),
      InvocationType: "Event"
    }).promise();
```

`InvocationType: "Event"` instructs vehicle-status-poller to run asynchronously independent with vehicle-tracker-request execution

Lambda to poll for requested vehicle location

After being invoked by vehicle-tracker-request, vehicle-status-poller will open a MQTT socket connection to HSL data feed, and leave the connection open for five minutes.

```
      const client = mqtt.connect(HSL_ENDPOINT);
      const startDate = Date.now();

      client.on("connect", async () => {
        const topic = constructTopic(mode, route);
        await client.subscribe(topic);
      });

      client.on("message", async (topic, messageBuffer) => {
        try {
          await saveVehicleStatus(parseMessage(route, messageBuffer));
        } catch (error) {
          console.log("Could not save vehicle status, ignoring", error);
        }

        if (duration - (Date.now() - startDate) <= 0) {
          await client.end(true);
        }
      });

      client.on("end", async () => {
        await removeVehicleStatus(route);
      });
```

On connection to HSL data feed, vehicle-status-poller will subscribe to vehicle status channel, which contains various information about the vehicle such as the public transit line, vehicle movement speed, acceleration, door status ... These feed data will be parsed to a more readable format, and save to DynamoDB, indexed by the transit line number and vehicle number. After reaching a connection timeout of five minutes, the connection will be automatically closed. On closing, the tracked vehicle status will be removed from the database, opening slot for another tracking request.

Lambda to retrieve currently tracked vehicle locations

During any tracking of any active vehicle, user can make a GET request to /status endpoint to query for vehicle status with vehicle-status-retrieve Lambda. This lambda will respond with data fetched from DynamoDB which are populated by vehicle-status-poller.

```
      const params = {
          TableName: tableName,
          KeyConditionExpression: "#route = :route",
          ExpressionAttributeNames: {
              "#route": "route"
          },
          ExpressionAttributeValues: {
              ":route": route
          }
      };

      const response = await documentClient.query(params).promise();
```

7 Conclusion

The thesis advocates the trend of software development with Cloud and Serverless computing by discussing the history - the present - the future of cloud computing and serverless, and how they shape of trend of future development with many advantages over traditional software development and delivery, such as cost reduction, maintenance upkeep delegation and increased service delivery speed. Cloud technology will vigorously continue to develop thanks to the climbing competition between many cloud service providers. Development will shift from hardware-focus to software-focus, indirectly improving product delivery efficiency while optimizing cost to the fullest.

The paper also introduces a number of services from the most popular cloud provider Amazon Web Services. After that, some of the introduced services are stitched together with the help of Serverless Framework to create a functional cloud-based Serverless application - Vehicle Tracking. The project put into example the easiness of creating, deployment and managing an application entirely with only Infrastructure as Code and application logic, thus simple implementation, but functional and extensible.

While cloud technology improves, the technical jobs sector will see changes in the paradigm. There will be less need for server administrators, and there will be more cloud solutionist and architects jobs being available in the market. Developers will be more than ever in need of due to the rising number of internet services to improve quality of life.

The combination of all these elements will ensure the future of cloud technology, open up a bright path to a technologically advanced world. The possibility of the internet is endless, and it only scales up the need for Cloud technology. It can be foreseen that there will be many new services born to enrich the diverse ecosystem that is already existing, on much higher level of abstraction to cover specific requirements of the industry. In any pace, Cloud technology is here to stay.

Bibliography

1 Keith D Foote. A Brief History of Cloud Computing. dataversity.net; 2017. Available from: https://www.dataversity.net/brief-history-cloud-computing [cited July 08, 2019].

2 Peter, Sbarski. Serverless Architecture on AWS. Shelter Island, NY 11964: Manning Publications Co.; 2017.

3 Cloud Computing. wikipedia.org;. Available from: https://en.wikipedia.org/wiki/Cloud_computing [cited July 10, 2019].

4 Hansen S. 10 Benefits and Advantages of Cloud Computing. hackernoon.com; 2017. Available from: https://hackernoon.com/10-benefits-and-advantages-of-cloud-computing-3c20c7433814 [cited July 10, 2019].

5 Christopher Mines. 4 Reasons Why Cloud Computing is Also a Green Solution. www.greenbiz.com; 2011. Available from: https://www.greenbiz.com/blog/2011/07/27/4-reasons-why-cloud-computing-also-green-solution [cited July 08, 2019].

6 Raj Bala DSDW Bob Gill. Magic Quadrant for Cloud Infrastructure as a Service, Worldwide. www.gartner.com; 2019. Available from: https://www.gartner.com/doc/reprints?id=1-1CMAPXNO&ct=190709&st=sb [cited July-20 2019].

7 Canalys. Cloud Market Share Q4 2018 and full year 2018. www.canalys.com; 2019. Available from: https://www.canalys.com/newsroom/cloud-market-share-q4-2018-and-full-year-2018 [cited July-17, 2019].

8 Costello K. Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.5 Percent in 2019. www.gartner.com; 2019. Available from: https://www.gartner.com/en/newsroom/press-releases/2019-04-02-gartner-forecasts-worldwide-public-cloud-revenue-to-g [cited July-20 2019].

9 Spencer MK. AWS isn't just the Future of the Cloud. Utopiapress; 2018. Available from: https://medium.com/utopiapress/aws-isnt-just-the-future-of-the-cloud-13cbaa0ff45 [cited July-20 2019].

10 Infrastructure as Code. Amazon Web Services; 2017. Available from: https://d1.awsstatic.com/whitepapers/DevOps/infrastructure-as-code.pdf [cited July-24 2019].

11 AWS CloudFormation Limits. Amazon Web Services;. Available from: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cloudformation-limits.html [cited July-24 2019].

12 What is AWS Lambda? Amazon Web Services;. Available from: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html [cited July-24 2019].

13 Malishev N. Lambda Cold Starts, A Language Comparison. Nathan Malishev;. Available from: https://medium.com/@nathan.malishev/lambda-cold-starts-language-comparison-%EF%B8%8F-a4f4b5f16a62 [cited July-25 2019].

14 Patra C. Amazon DynamoDB: 10 Things You Should Know. Cloudacademy;. Available from: https://cloudacademy.com/blog/amazon-dynamodb-ten-things [cited July-31 2019].

1 serverless.yml

```
service: vehicle-tracking

provider:
  name: aws
  runtime: nodejs10.x
  stage: dev
  region: eu-west-1
  iamRoleStatements:
    - Effect: "Allow"
      Action:
        - "*"
      Resource: "*"

functions:
  vehicle-tracker-request:
    handler: vehicle-tracker-request/index.requestTracker
    events:
      - http:
          path: track
          method: post

  vehicle-status-poller:
    handler: vehicle-status-poller/index.startPolling
    timeout: 600

  vehicle-status-retrieve:
    handler: vehicle-status-retrieve/index.getVehicleStatus
    events:
      - http:
          path: status
          method: get
          request:
            parameters:
              paths:
                route: true

package:
  exclude:
    - "node_modules/aws-sdk/**"

resources:
  Resources:
    # Vehicle DynamoDB table
    VehicleStatusTable:
      Type: AWS::DynamoDB::Table
      DeletionPolicy: Delete
      Properties:
        BillingMode: PAY_PER_REQUEST
        KeySchema:
          - AttributeName: route
            KeyType: HASH
          - AttributeName: vehicleNumber
            KeyType: RANGE
        AttributeDefinitions:
          - AttributeName: route
            AttributeType: S
          - AttributeName: vehicleNumber
            AttributeType: S
    VehicleStatusTableParameter:
      Type: AWS::SSM::Parameter
      Properties:
        Name: /applications/vehicle-tracker/vehicle-status-table-name
        Type: String
        Value: !Ref VehicleStatusTable
        Description: Vehicle status table name
    VehicleStatusTableArnParameter:
      Type: AWS::SSM::Parameter
      Properties:
        Name: /applications/vehicle-tracker/vehicle-status-table-arn
        Type: String
        Value: !GetAtt VehicleStatusTable.Arn
        Description: Vehicle status table ARN
```

2 package.json

```json
{
  "name": "serverless-vehicle-tracking-service",
  "version": "1.0.0",
  "description": "A plugable serverless service to track vehicle status",
  "scripts": {
    "deploy": "npx sls deploy"
  },
  "author": "Hieu Nguyen",
  "devDependencies": {
    "serverless": "^1.49.0"
  },
  "dependencies": {
    "async-mqtt": "^2.3.0",
    "aws-sdk": "^2.504.0"
  }
}
```

3 dynamo.js

```
"use strict";

const AWS = require("aws-sdk");
AWS.config.update({ region: "eu-west-1" });
const { getVehicleStatusTableName } = require("./ssm");
const documentClient = new AWS.DynamoDB.DocumentClient();

const HASH_KEY = "route";
const RANGE_KEY = "vehicleNumber";

/**
 * Save vehicle status to Dynamo vehicle status table
 *
 * @param {object} status
 * @returns {null}
 */
async function saveVehicleStatus(status) {
  if (!status.route) throw new Error("Cannot save status without route");
  if (!status.vehicleNumber)
    throw new Error("Cannot save status without vehicleNumber");
  if (!status.lat) throw new Error("Cannot save status without lat");
  if (!status.lon) throw new Error("Cannot save status without lon");

  const tableName = await getVehicleStatusTableName();

  const params = {
    Item: status,
    ReturnConsumedCapacity: "TOTAL",
    TableName: tableName,
    ReturnValues: "NONE"
  };

  return documentClient
    .put(params)
    .promise()
    .then(response => {
      return null;
    })
    .catch(error => {
      console.log(error);
      throw error;
    });
}

/**
```

```
 * Get all items on Dynamo with given route
 * @param {string} route
 *
 * @return {Array<Object>}
 */
async function getAllItemsForRoute(route) {
  const tableName = await getVehicleStatusTableName();

  const params = {
    TableName: tableName,
    KeyConditionExpression: "#route = :route",
    ExpressionAttributeNames: {
      "#route": "route"
    },
    ExpressionAttributeValues: {
      ":route": route
    }
  };

  const response = await documentClient.query(params).promise();
  return response.Items;
}

/**
 * Delete all vehicle status related on a route
 *
 * @param {string} route
 * @returns {null}
 */
async function removeVehicleStatus(route) {
  const tableName = await getVehicleStatusTableName();

  const deletingItems = await getAllItemsForRoute(route);

  const params = {
    RequestItems: {
      [tableName]: deletingItems.map(item => {
        return {
          DeleteRequest: {
            Key: {
              [HASH_KEY]: route,
              [RANGE_KEY]: item.vehicleNumber
            }
          }
        };
      })
    }
  };

  console.log(`Removing vehicle status for route ${route}`);
  await documentClient.batchWrite(params).promise();

  return null;
}

module.exports = {
  saveVehicleStatus,
  getAllItemsForRoute,
  removeVehicleStatus
};
```

4 ssm.js

```
"use strict";

const AWS = require("aws-sdk");
AWS.config.update({ region: "eu-west-1" });
const ssm = new AWS.SSM();

const VEHICLE_STATUS_TABLE_PATH =
  "/applications/vehicle-tracker/vehicle-status-table-name";

/**
 * From given SSM paramter store name, get the value
 *
 * @param {string} name
 * @returns {string} value
 */
function getSsmParameter(name) {
  return ssm
    .getParameters({
      Names: [name]
    })
    .promise()
    .then(({ Parameters }) => {
      return Parameters[0].Value;
    });
}

function getVehicleStatusTableName() {
  return getSsmParameter(VEHICLE_STATUS_TABLE_PATH);
}

module.exports = {
  getVehicleStatusTableName
};
```

5 vehicle-tracker-request.js

```
"use strict";

const AWS = require("aws-sdk");
AWS.config.update({ region: "eu-west-1" });
const Lambda = new AWS.Lambda();
const { getAllItemsForRoute } = require("../lib/dynamo");

const POLLER_FUNCTION_ARN =
    `arn:aws:lambda:eu-west-1:${283241335956}:function:vehicle-tracking-dev-vehicle-

module.exports.requestTracker = async event => {
  if (!event.route) {
    throw new Error("Unspecified route ID");
  } else if (event.mode && ![“train", "bus", "tram"].includes(event.mode))
      {
    throw new Error(`Invalid mode ${event.mode}, type --help`);
  }

  const existingRecords = await getAllItemsForRoute(event.route);

  if (existingRecords.length > 0) {
    return {
      statusCode: 400,
      body: {
        ok: false,
        reason: `Another polled is ran for ${event.mode} ${event.route}`
      }
    };
  }

  await Lambda.invoke({
    FunctionName: POLLER_FUNCTION_ARN,
    Payload: JSON.stringify({
      route: event.route,
      mode: event.mode
    }),
    InvocationType: "Event"
  }).promise();

  return {
    statusCode: 200,
    body: {
      ok: true
    }
  };
};
```

6 vehicle-status-poller.js

```
"use strict";

const mqtt = require("async-mqtt");

const { saveVehicleStatus, removeVehicleStatus } =
    require("../lib/dynamo");

const HSL_ENDPOINT = "mqtts://mqtt.hsl.fi:8883";

const DEFAULT_DURATION = 5 * 60 * 1000;

function constructTopic(mode, routeId) {
  return `/hfp/v2/journey/ongoing/vp/${mode}/+/+/${routeId}/1/+/+/+/+/#`;
}

/**
 * Parse Reittiopas message buffer to readable js object format
 *
 * @param {Buffer} messageBuffer
 * @return {Object}
 */
function parseMessage(route, messageBuffer) {
  const messageString =
      JSON.parse(Buffer.from(messageBuffer).toString()).VP;

  return {
    lat: messageString.lat,
    lon: messageString.long,
    route,
    routeName: messageString.desi,
    vehicleNumber: `${messageString.oper}/${messageString.veh}`, //
        operatorId/vehicleNumber format
    vehicleSpeed: messageString.spd // meters per second
  };
}

/**
 * Pull vehicle status update from Reittiopas and update to Dynamo
 *
 * @param {string} mode
 * @param {string} route
 */
async function pullUpdate(mode, route, duration = DEFAULT_DURATION) {
  return new Promise(resolve => {
    const client = mqtt.connect(HSL_ENDPOINT);
    const startDate = Date.now();

    client.on("connect", async () => {
      const topic = constructTopic(mode, route);
      await client.subscribe(topic);
      console.log("Subscribed to ", topic);
    });

    client.on("message", async (topic, messageBuffer) => {
      try {
        console.log(parseMessage(route, messageBuffer));
        await saveVehicleStatus(parseMessage(route, messageBuffer));
      } catch (error) {
        console.log("Could not save vehicle status, ignoring", error);
      }

      if (duration - (Date.now() - startDate) <= 0) {
        await client.end(true);
      }
    });

    client.on("end", async () => {
      await removeVehicleStatus(route);
      resolve();
    });
  });
}

module.exports.startPolling = async event => {
  const mode = event.mode || process.env.mode;
  const route = event.route || process.env.route;

  await pullUpdate(mode, route);

  return {
    statusCode: 200,
    ok: true
  };
};
```

7 vehicle-status-retrieve.js

```javascript
"use strict";

const { getAllItemsForRoute } = require("../lib/dynamo");

module.exports.getVehicleStatus = async event => {
  return getAllItemsForRoute(event.route);
};
```