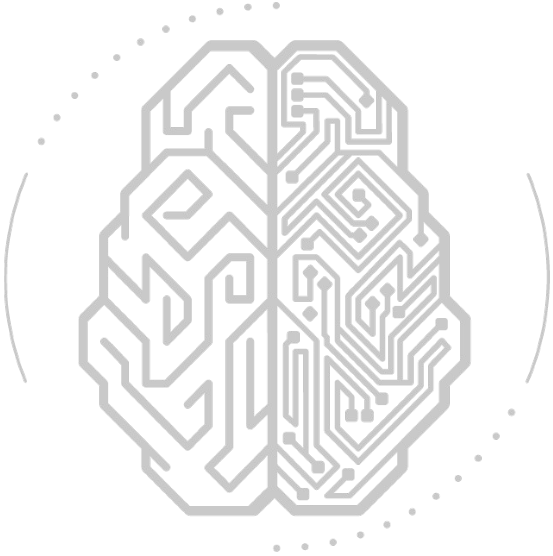
Introduction to Serverless



What's in it for you

AWS Serverless Introduction

S.NO.	AGENDA
1	Introduction
2	Demo
3	Lambda Code Walkthrough
4	API Gateway
5	Step Function
6	Design Patterns
7	Demo

AWS Lambda

Speaker

Sanchit Jain

Lead Architect - AWS at Quantiphi
AWS APN Ambassador

What is serverless?

What is Serverless?

a cloud-native platform

for

short-running, stateless computation

and

event-driven applications

which

scales up and down instantly and automatically

and

charges for actual usage at a millisecond granularity

Greater Agility

Less Overhead

Better Focus

Increased Scale

More Flexibility

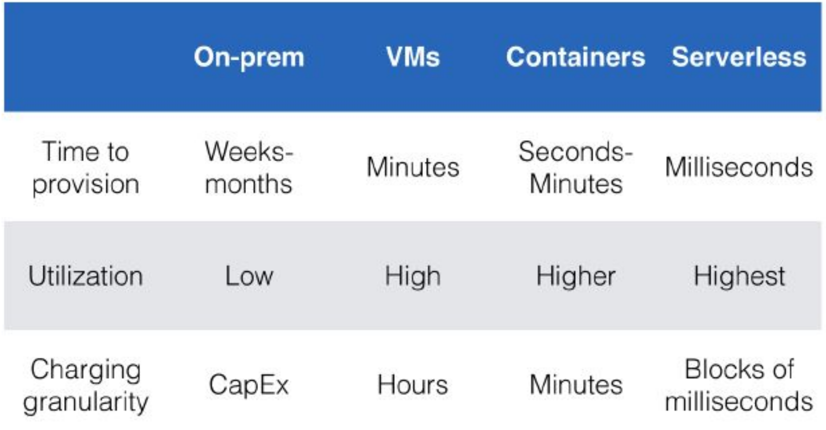
Faster Time To Market

Why is Serverless attractive?

- Server-less means no servers? Or worry-less about servers?
- Runs code **only** on-demand on a per-request basis
- Making app development & ops dramatically faster, cheaper, easier
- Drives infrastructure cost savings

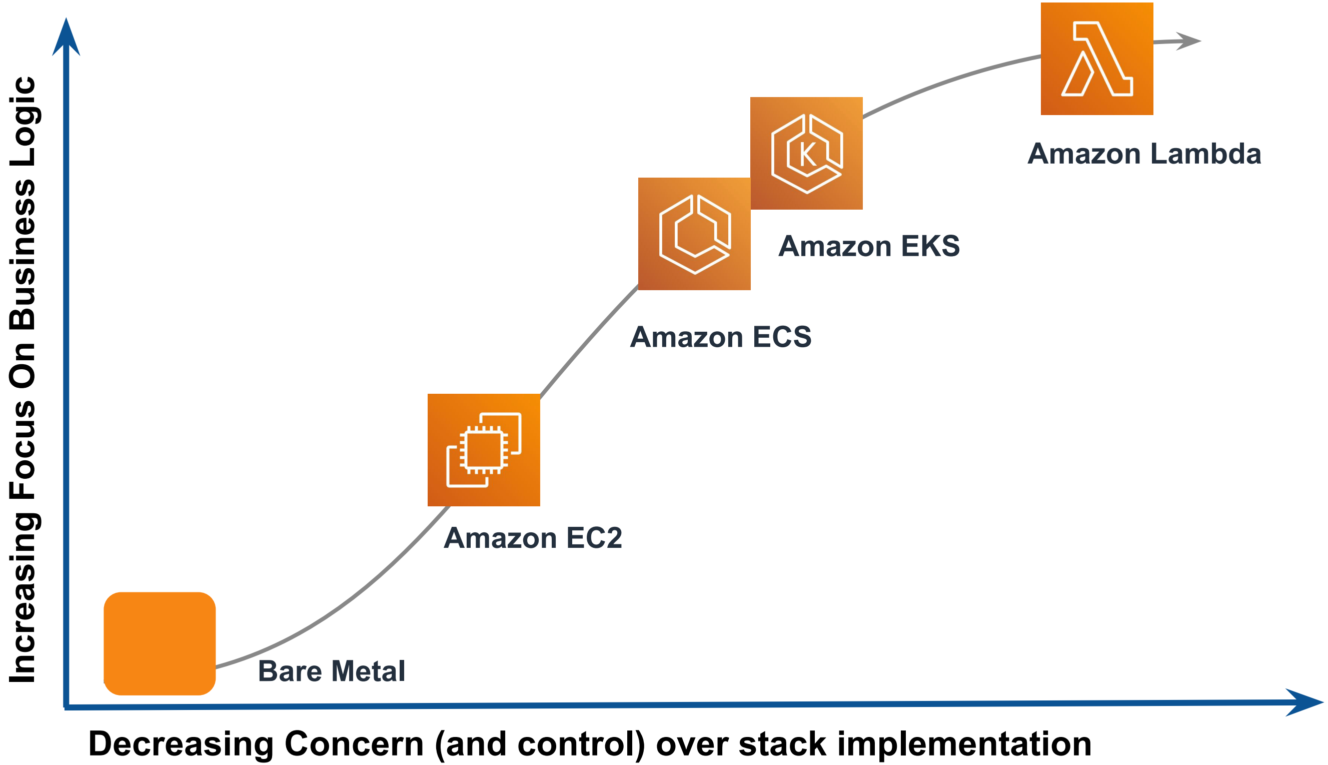
No servers

Just code



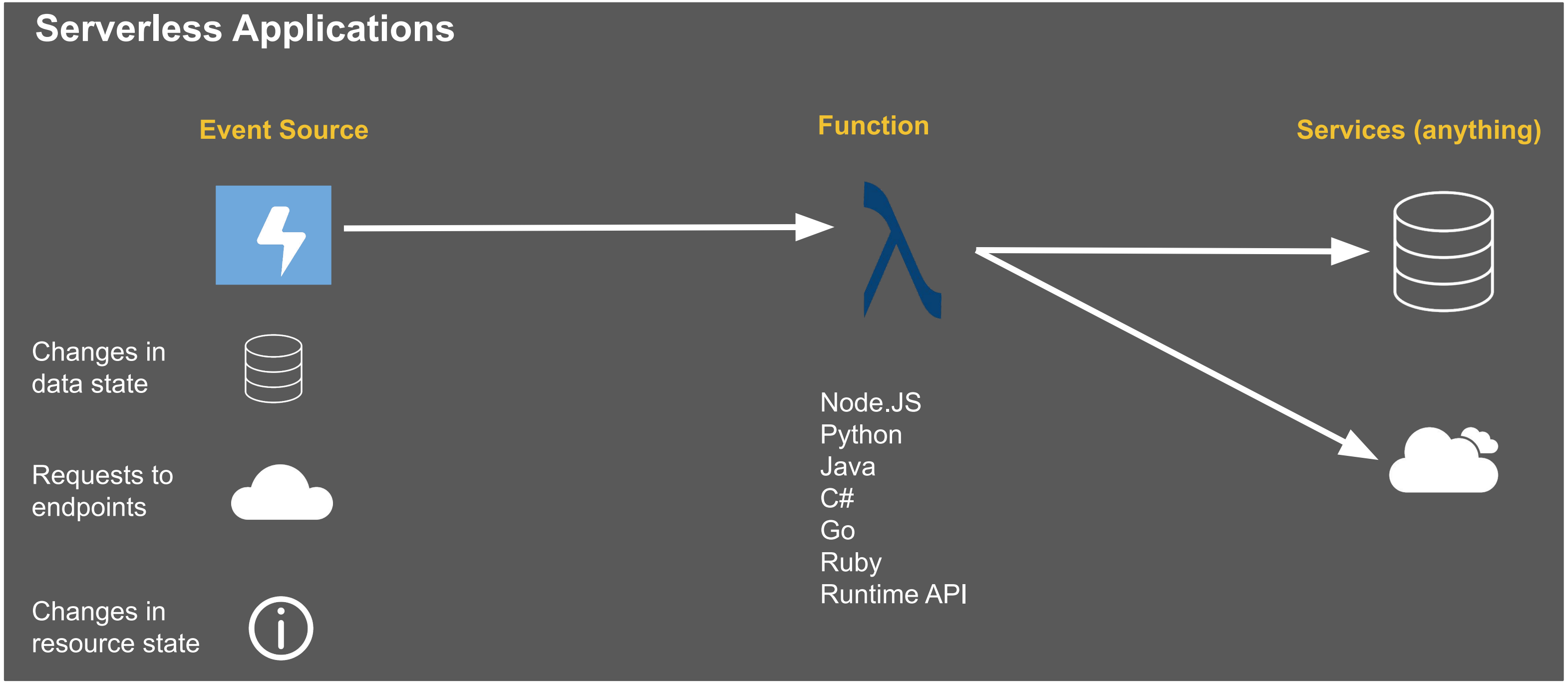
	On-prem	VMs	Containers	Serverless
Time to provision	Weeks-months	Minutes	Seconds-Minutes	Milliseconds
Utilization	Low	High	Higher	Highest
Charging granularity	CapEx	Hours	Minutes	Blocks of milliseconds

Where Serverless Stands?


Increasing Focus On Business Logic
Amazon Lambda
Amazon EKS
Amazon ECS
Amazon EC2
Bare Metal
Decreasing Concern (and control) over stack implementation

What triggers code execution?

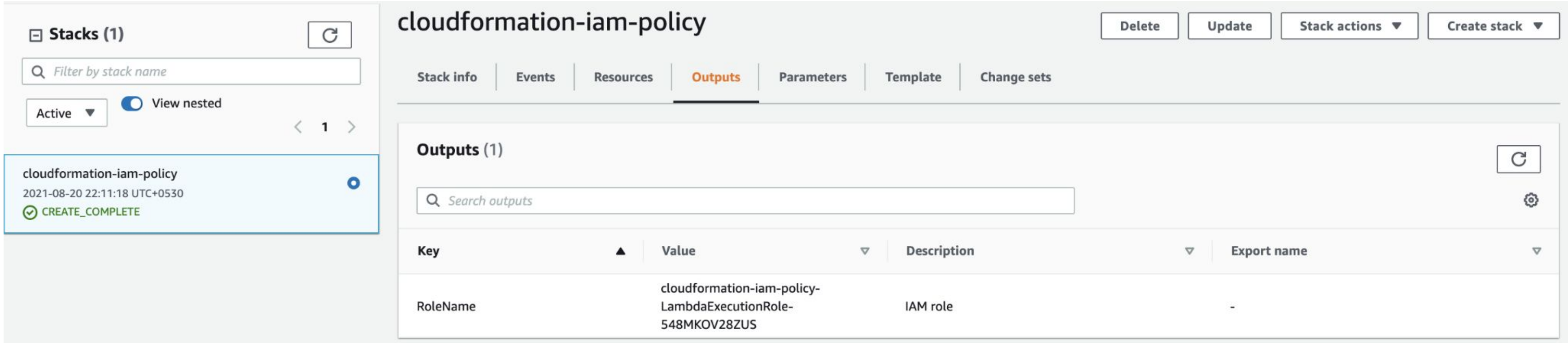
- Runs code in response to events
- Event-programming model



DEMO

Prerequisites

- Download the cloudformation template from this link and Deploy it
- Once the Cloudformation stack is deployed successfully please capture the values for RoleName



cloudformation-iam-policy
2021-08-20 22:11:18 UTC+0530
CREATE_COMPLETE

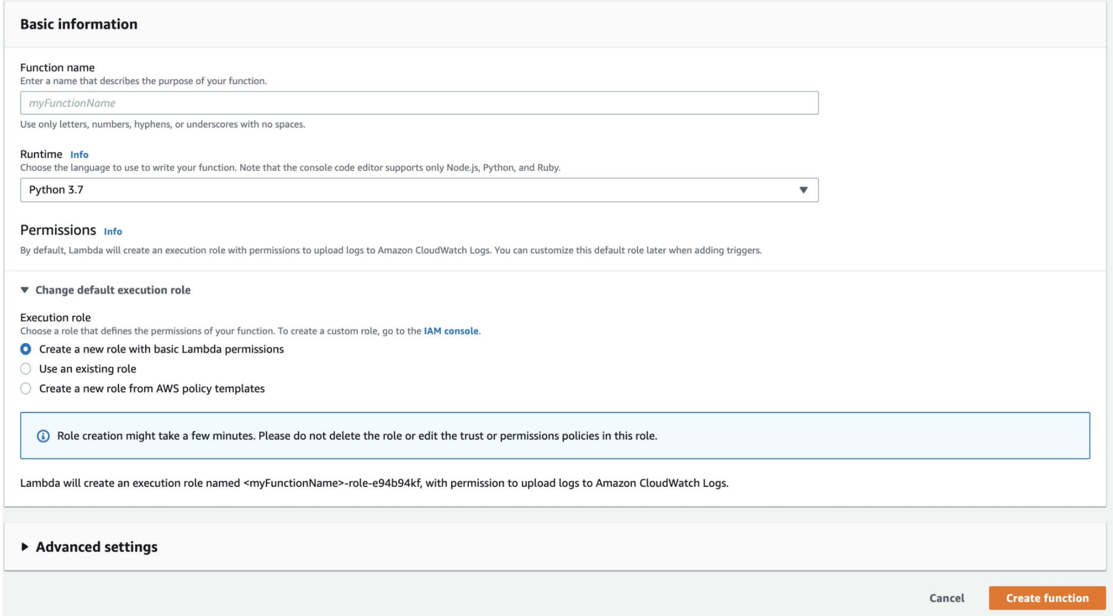
cloudformation-iam-policy

Outputs (1)

Key	Value	Description	Export name
RoleName	cloudformation-iam-policy-LambdaExecutionRole-548MKOV28ZUS	IAM role	-

Create function

- Navigate to the AWS Lambda service
- On the Lambda console, Click Create function
- In the Create function, enter demo-lambda as the function name, runtime as Python 3.7
- In the execution role section, select the role cloudformation-iam-policy--xxxxx from the drop-down list
- Click Create function



Basic information

Function name
Enter a name that describes the purpose of your function.
myFunctionName
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.7 ▼

Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ **Change default execution role**

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the **IAM console**.

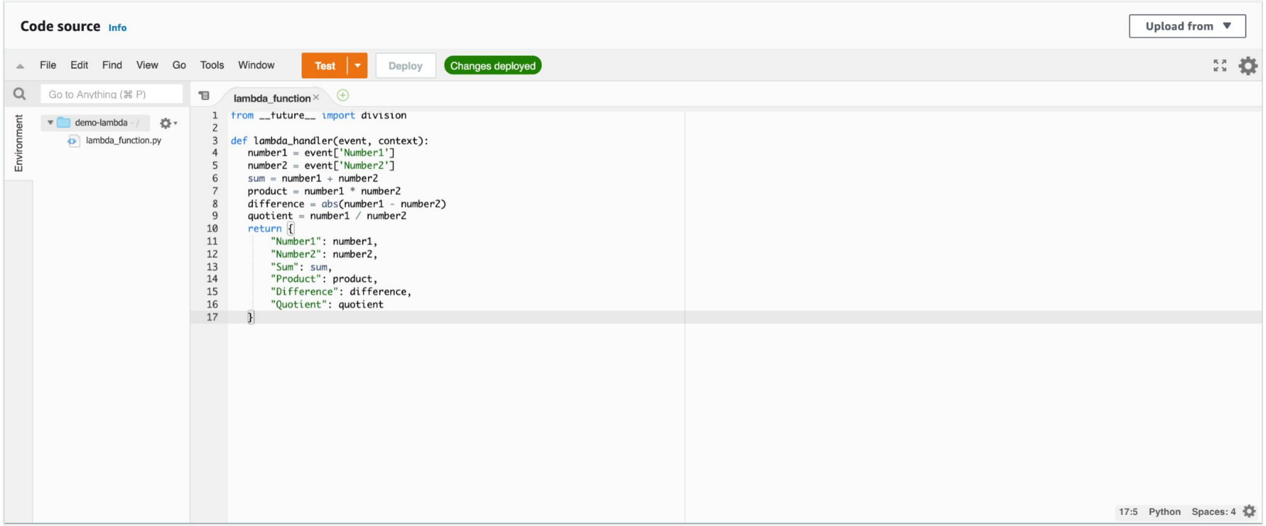
● Create a new role with basic Lambda permissions
○ Use an existing role
○ Create a new role from AWS policy templates

ⓘ Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.

Lambda will create an execution role named <myFunctionName>-role-e94b94kf, with permission to upload logs to Amazon CloudWatch Logs.

Configure function

- Download the code from this link and paste it into code source
- Click Deploy, then click Test
- Copy the message { "Number1": 10, "Number2": 20 }, and paste it
- Click Test again, and capture the output



```python
from __future__ import division

def lambda_handler(event, context):
    number1 = event['Number1']
    number2 = event['Number2']
    sum = number1 + number2
    product = number1 * number2
    difference = abs(number1 - number2)
    quotient = number1 / number2
    return {
        "Number1": number1,
        "Number2": number2,
        "Sum": sum,
        "Product": product,
        "Difference": difference,
        "Quotient": quotient
    }
```

Lambda Code Walkthrough

Anatomy Of A Lambda Function

Handler() Function

Function to be executed upon invocation

Event Object

Data sent during Lambda Function invocation

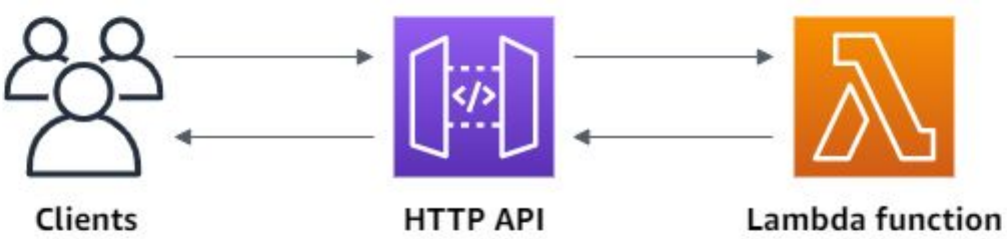
Context Object

Methods available to interact with runtime information (request ID, log group, more)

```
def lambda_handler(event, context):
    return {
        “Hello World!”
 }
```

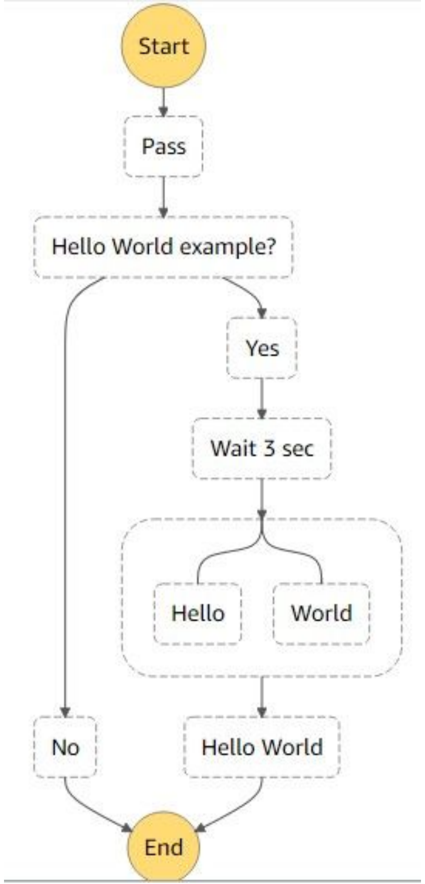
What is AWS API Gateway?

- Amazon API Gateway is an AWS service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale. API developers can create APIs that access AWS or other web services, as well as data stored in the AWS Cloud
- API Gateway creates RESTful APIs that:
  - Are HTTP-based.
  - Enable stateless client-server communication.
  - Implement standard HTTP methods such as GET, POST, PUT, PATCH, and DELETE.

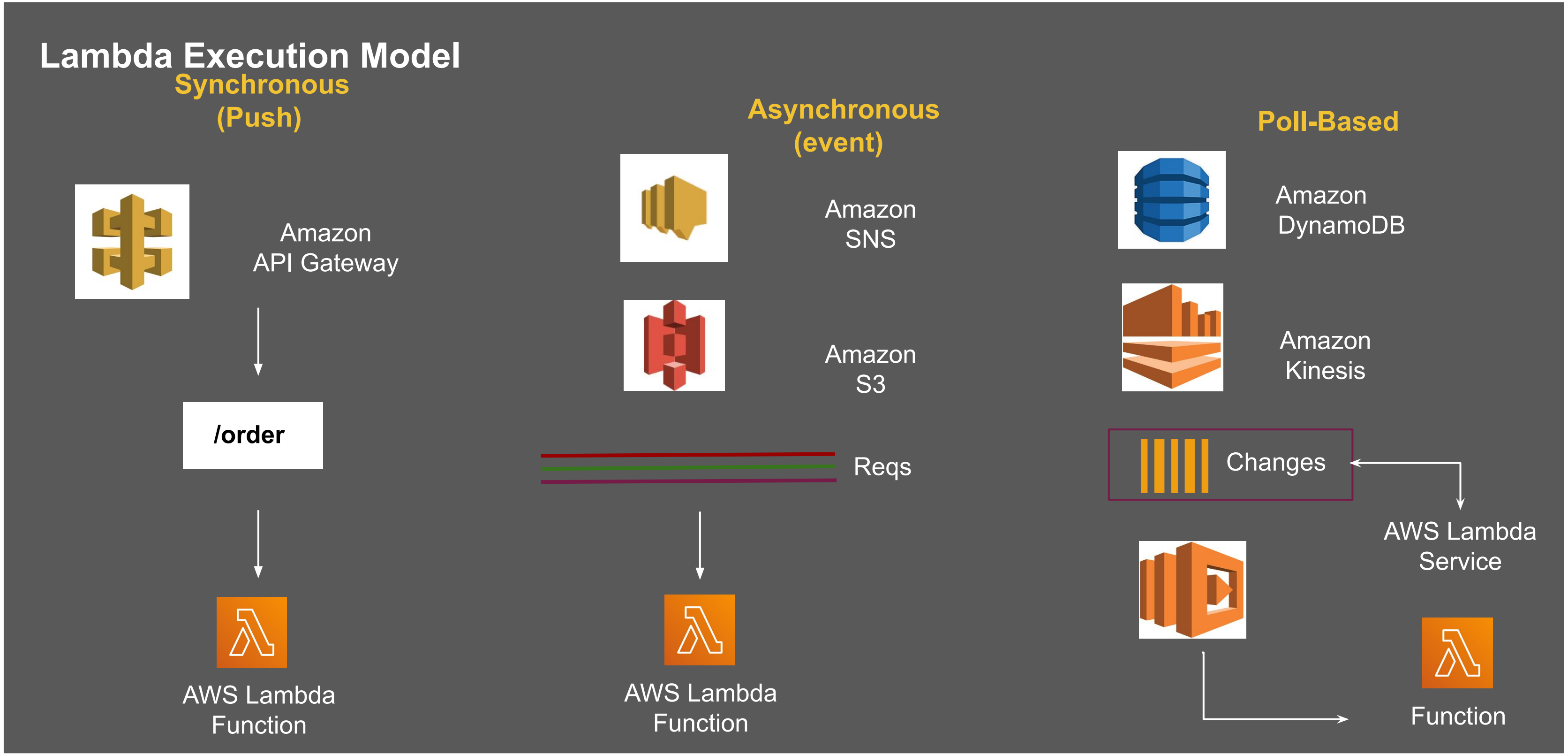


What is AWS Step Functions?

- AWS Step Function is a serverless orchestration service that allows integrating multiple AWS services to collate & design an enterprise-critical application or workflow with advance conditional branching and error handling
- ASL consist of three things
  - **State Machine Structure** - State machines are declared using JSON text and represents a structure consists of Comment, TimeoutSeconds, Version, StartAt, States
  - **Intrinsic functions** - Intrinsics are constructs like in programming languages, and can be leveraged to manipulate the data going to and from Task Resources
  - **Common State Fields** - Common State Fields consists of Comment, InputPath, OutputPath, Type, Next, End



Lambda Invocation/Execution Model



Lambda Execution Model

Synchronous (Push)

Amazon API Gateway

/order

AWS Lambda Function

Asynchronous (event)

Amazon SNS

Amazon S3

Reqs

AWS Lambda Function

Poll-Based

Amazon DynamoDB

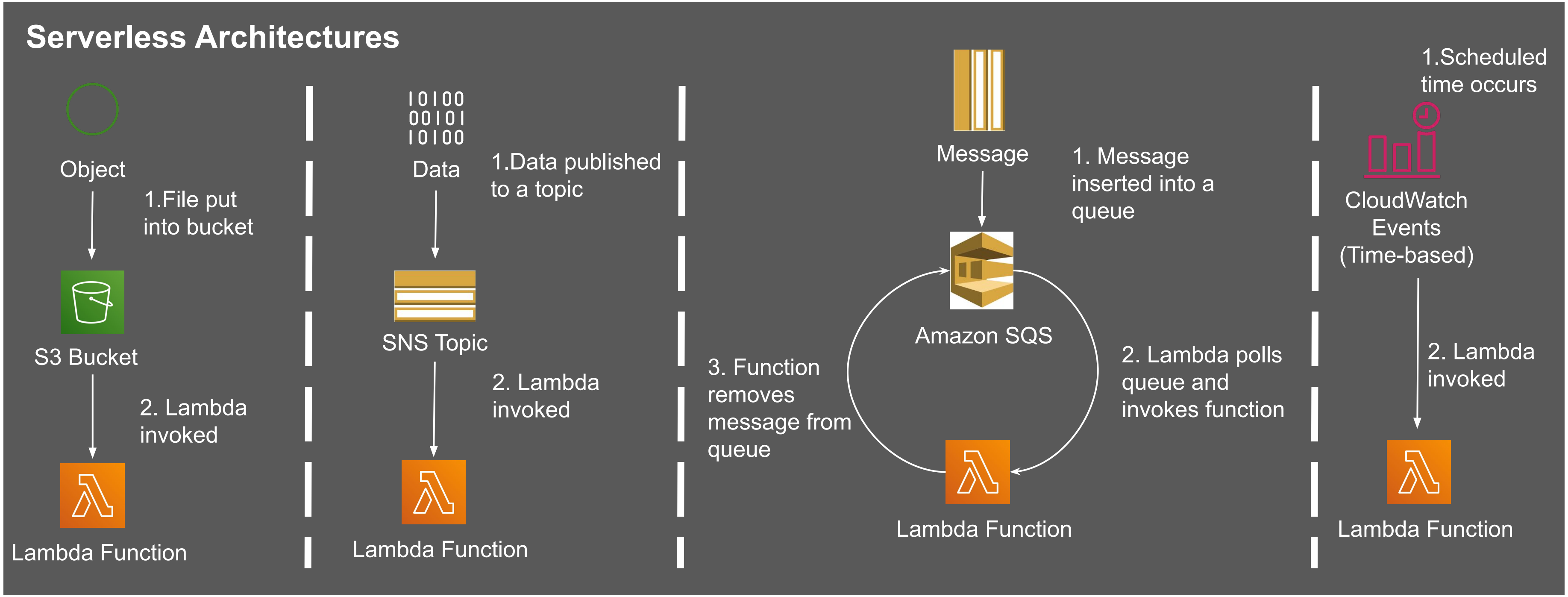
Amazon Kinesis

Changes

AWS Lambda Service

Function

Different Serverless Architecture



Different Use cases

Web Applications

- Static Websites
- Complex Web Apps
- Packages for Flask and Express

Backends

- Apps and Services
- Mobile
- IoT

Data Processing

- Real time
- Map Reduce
- Batch

Chatbots

- Powering Chatbot Logic

Amazon Alexa

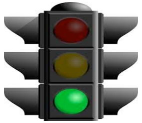
- Powering Voice Enabled Apps
- Alexa Skills Kit

IT Automation

- Policy Engines
- Extending AWS Services
- Infrastructure Management

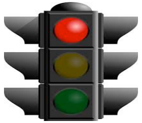
What is Serverless good for?

Serverless is **good** for
short-running stateless
event-driven



- Microservices
- Mobile Backends
- Bots, ML Inferencing
- IoT
- Modest Stream Processing
- Service integration

Serverless is **not good** for
Long-running stateful
number crunching

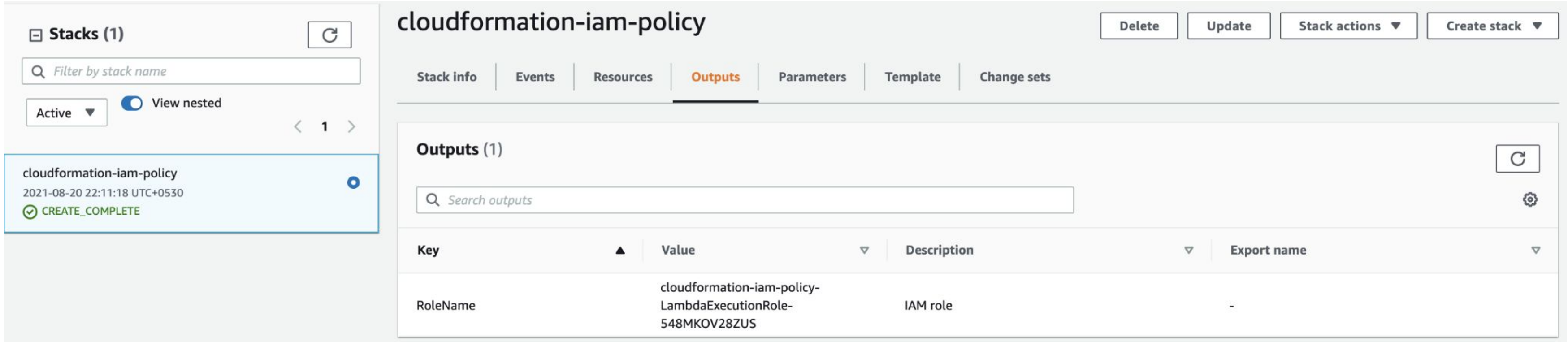


- Databases
- Deep Learning Training
- Heavy-Duty Stream Analytics
- Numerical Simulation
- Video Streaming

DEMO

Demo - Prerequisites

- Download the cloudformation template from this link and Deploy it
- Once the Cloudformation stack is deployed successfully please capture the values for RoleName



cloudformation-iam-policy
2021-08-20 22:11:18 UTC+0530
CREATE_COMPLETE

cloudformation-iam-policy

Outputs (1)

Key	Value	Description	Export name
RoleName	cloudformation-iam-policy-LambdaExecutionRole-548MKOV28ZUS	IAM role	-

Note: We will leverage the Lambda function deployed in the previous demonstration

Demo - Lambda invocation via SQS

- Create an Amazon SQS queue
  - Navigate to the SQS console
  - Choose Create a SQS, and then provide queue name
  - Capture the ARN (Amazon Resource Name) of the SQS
- Configure the event source
  - Under SQS console switch to details section, and select Lambda triggers
  - Click configure Lambda triggers, and select or provide Lambda ARN
- Now we will test the integration
  - Copy the message { "Number1": 10, "Number2": 20 }, and send it as a SQS message
  - Verify the lambda Cloudwatch logs and output for the same

Demo 2 - Lambda invocation via API Gateway

- Create an HTTP API using the API Gateway console. Then, you invoke your API
  - Navigate to the API Gateway console
  - Choose Create API, and then choose Build for REST API.
  - For Integrations, choose Add integration as Lambda and provide Lambda function name
  - Then provide API name, Choose Next.
  - Review the route that API Gateway creates for you, and then choose Next.
  - Review the stage that API Gateway creates for you, and then choose Next.
  - Choose Create
- Test your API
  - Capture your API's invoke URL, the full URL should look like https://abcdef123.execute-api.us-east-2.amazonaws.com/dev.
  - Command: curl -X "POST" -H "Content-Type: application/json" -d '{"Number1": 10, "Number2": 20}' {{ url }}
  - Verify your API's response in your command line.

Demo - Lambda invocation via Step Function

- Create an Amazon Step Function
  - Navigate to the Step Function console and choose Create a state machine.
  - On the Define state machine page, choose Write your workflow in code. For Type, choose Standard.
  - Download the state machine json from this link, and add the following state machine definition using the ARN of the Lambda function that we created earlier. Choose Next.
  - Enter a Name for your state machine, and provide execution role. Click Create state machine
- Now we will test the step function
  - Under step function detail page, click start execution
  - Copy the message { "Number1": 10, "Number2": 20 }, and click start execution
  - Verify the lambda Cloudwatch logs and output for the same

THANK YOU