

Introduction to Serverless



Speaker



Sanchit Jain

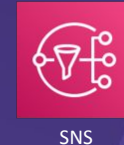
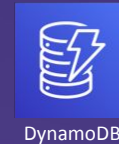
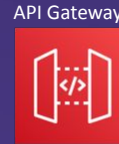
Lead Architect - AWS at Quantiphi
AWS APN Ambassador

FOLLOW ME



Agenda

AWS Serverless Introduction	
S.NO.	AGENDA
1	Introduction
2	Demo
3	Lambda Code Walkthrough
4	SQS & SNS
5	Demo
6	Step Function
7	Demo



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



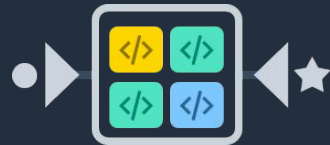
Faster Time To Market

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



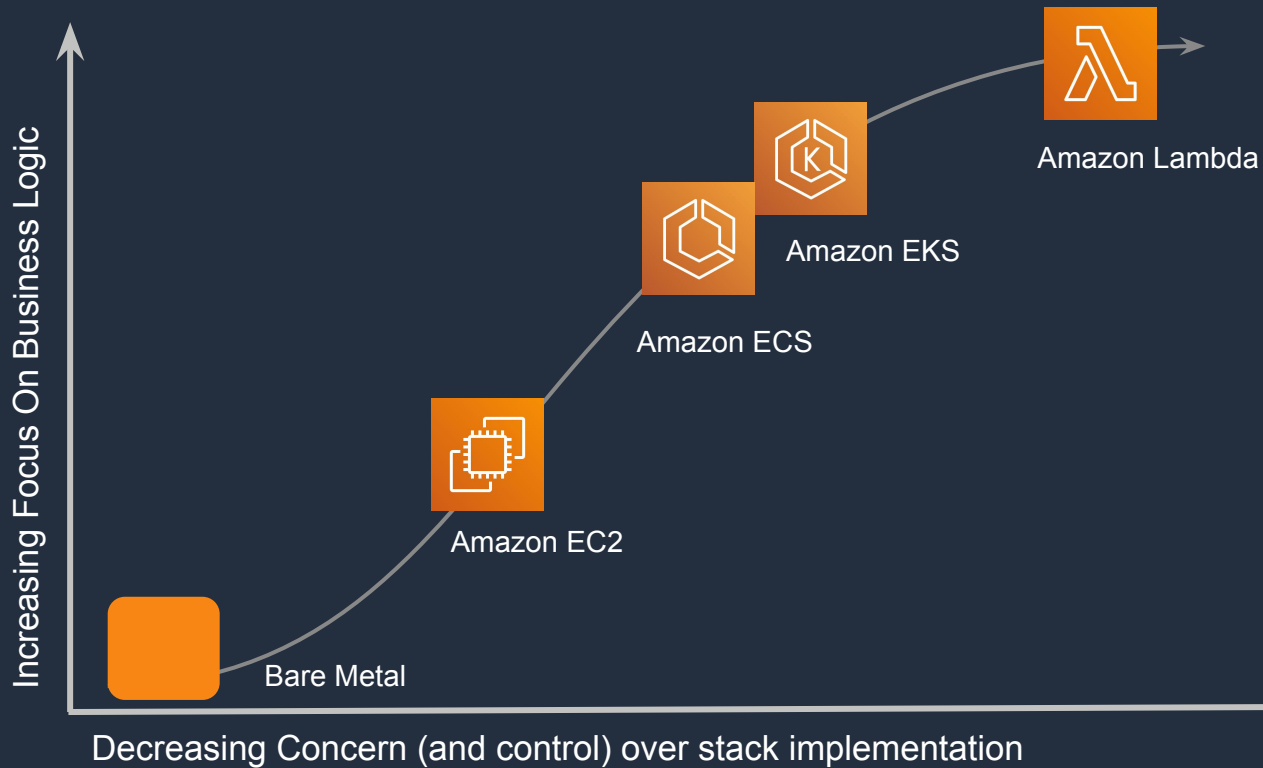
No servers



Just code

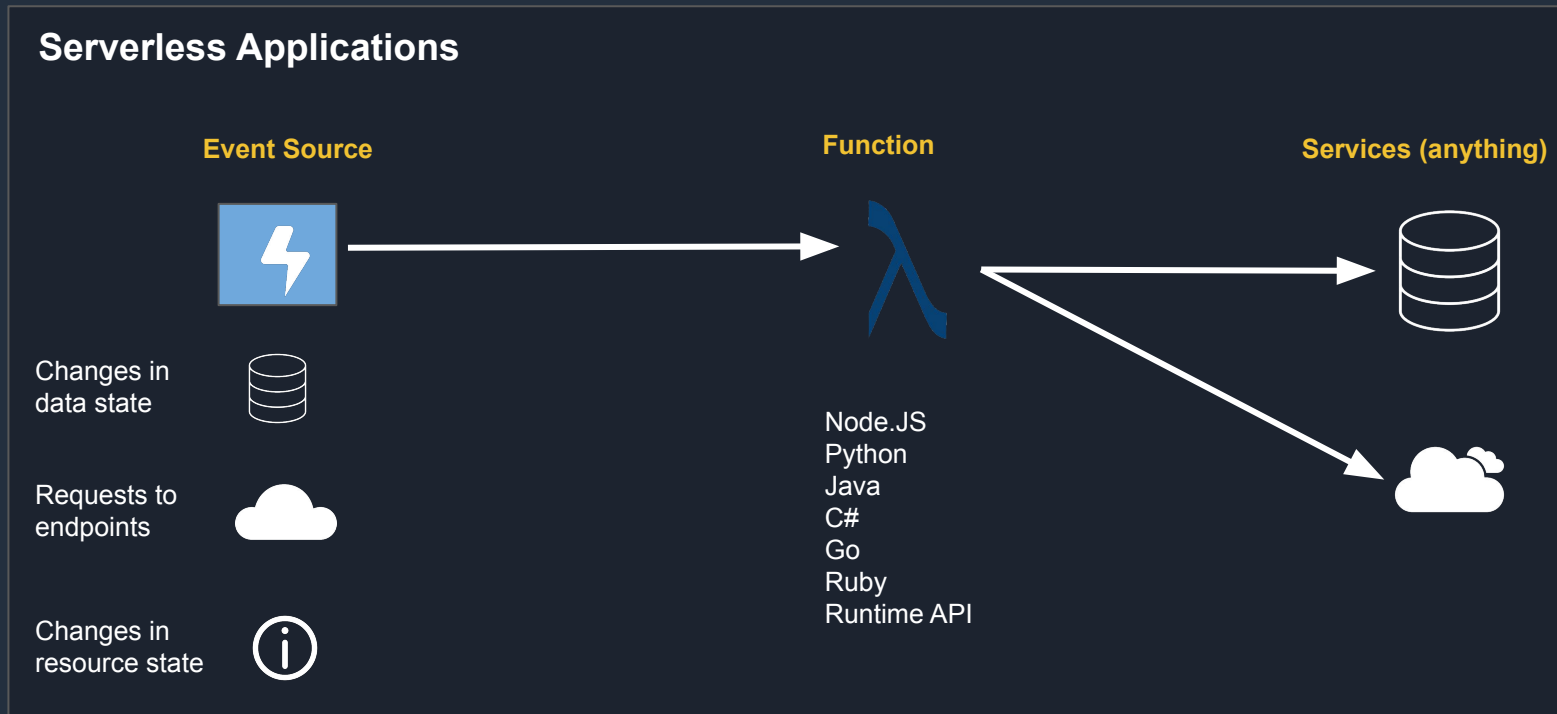
	On-prem	VMs	Containers	Serverless
Time to provision	Weeks-months	Minutes	Seconds-Minutes	Milliseconds
Utilization	Low	High	Higher	Highest
Charging granularity	CapEx	Hours	Minutes	Blocks of milliseconds

Where Serverless Stands?



What triggers code execution?

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

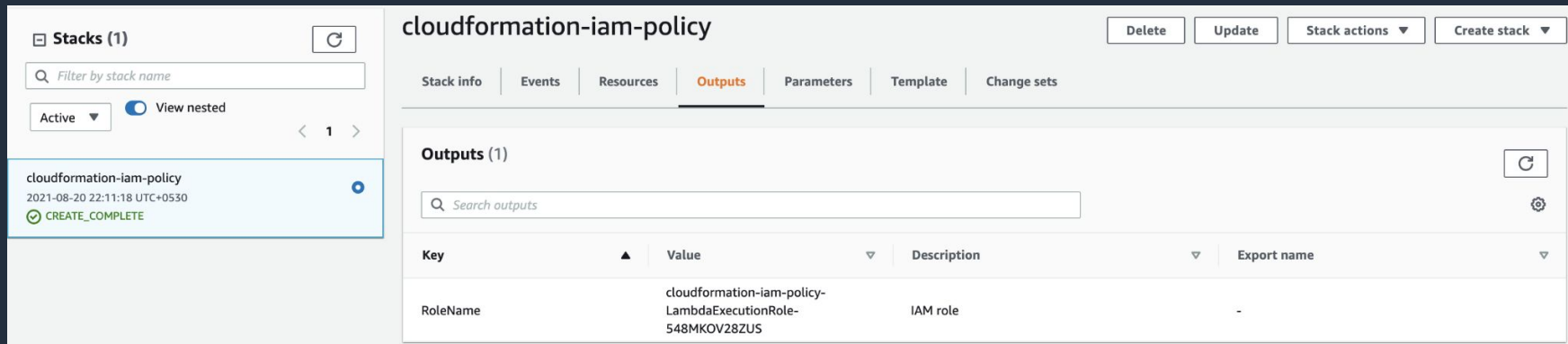


Demo



Prerequisites

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



The screenshot displays the AWS CloudFormation console interface. On the left, the 'Stacks (1)' sidebar shows a single stack named 'cloudformation-iam-policy' with a status of 'CREATE_COMPLETE'. The main panel is titled 'cloudformation-iam-policy' and features tabs for 'Stack info', 'Events', 'Resources', 'Outputs', 'Parameters', 'Template', and 'Change sets'. The 'Outputs' tab is selected, showing a table with one output named 'RoleName'. The output value is 'cloudformation-iam-policy-LambdaExecutionRole-548MKOV28ZUS', described as an 'IAM role'. The console also includes search bars for stacks and outputs, and buttons for stack management actions like 'Delete', 'Update', and 'Create stack'.

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.

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.

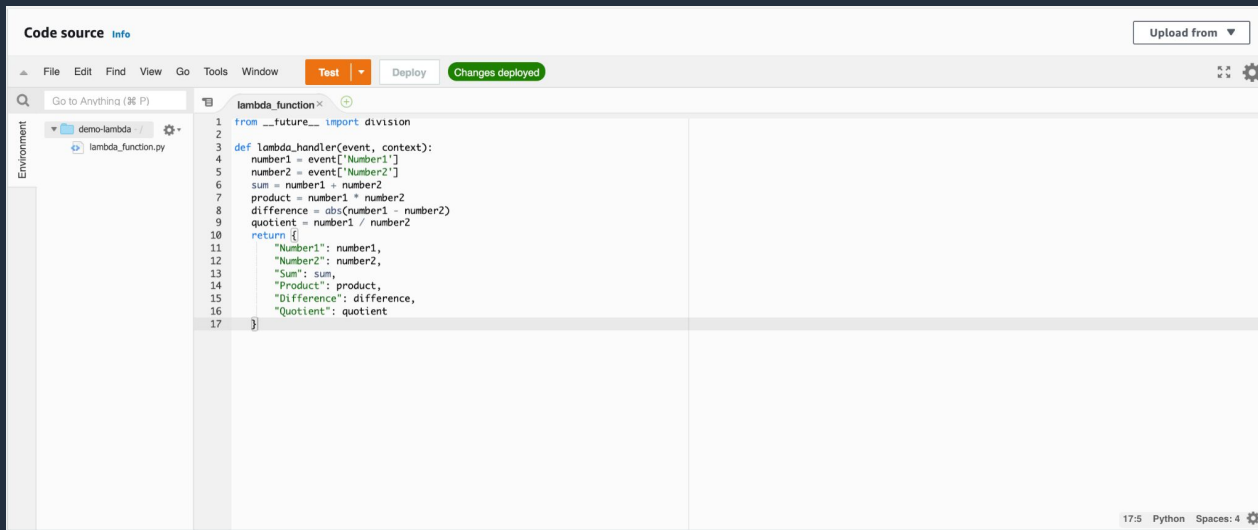
► Advanced settings

Cancel

Create function

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



Lambda Code Walkthrough

Anatomy Of A Lambda Function

Handler() Function

Function to be executed upon invocation

Event Object

Data sent during Lambda Function invocation

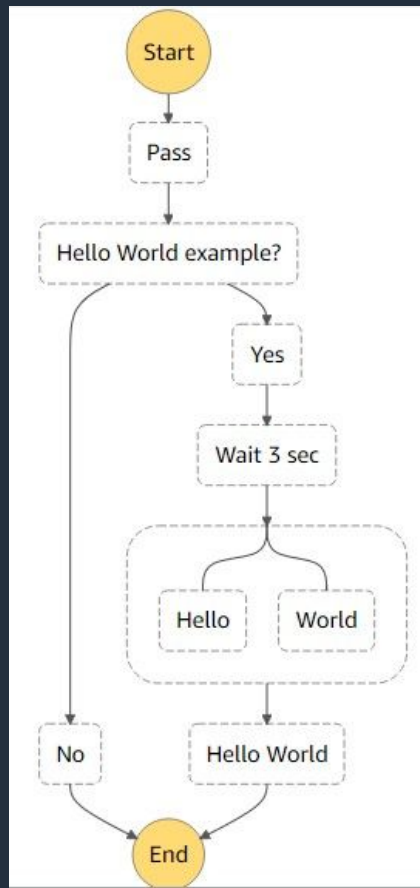
Context Object

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

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

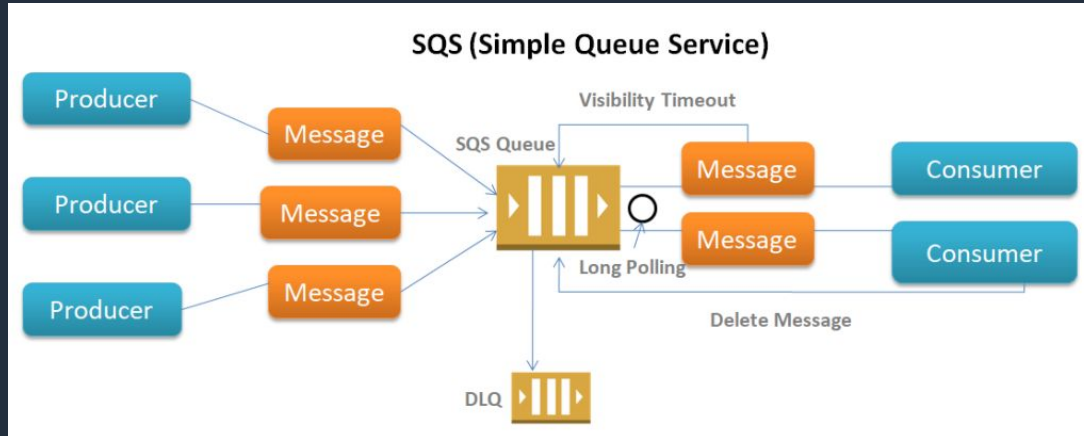
What is AWS Step Functions?

- 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** - Ininsics 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



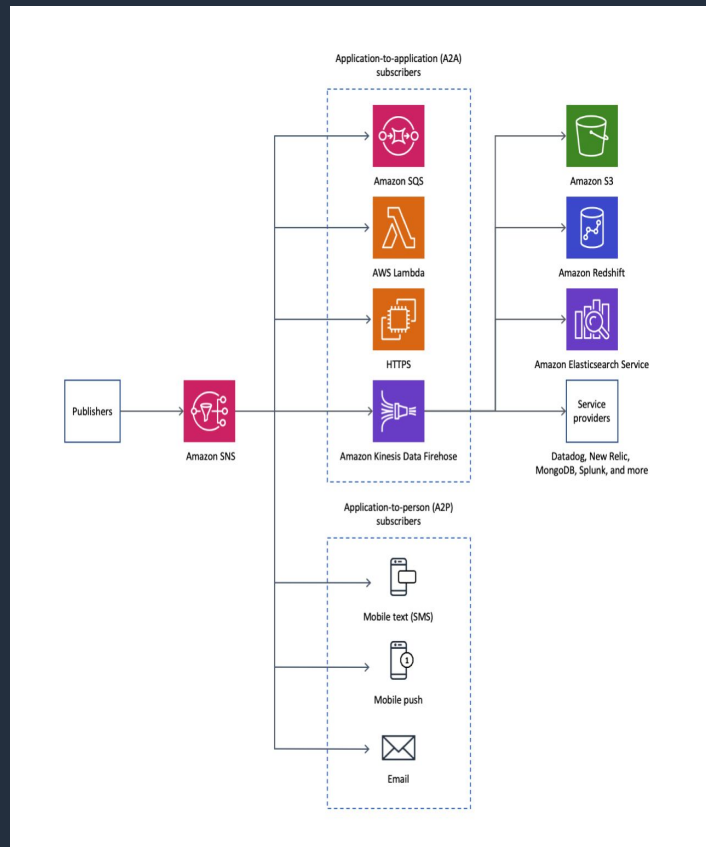
What is AWS SQS?

- Oldest AWS offering (over 10 years old)
- Fully managed service, use to decouple applications
- Scales from 1 message per second to 10,000s per second, and Low latency (<10 ms on publish and receive)
- Default retention of messages: 4 days, maximum of 14 days
- No limit to how many messages can be in the queue, and messages are deleted after they're read by consumers
- Consumers share the work to read messages & scale horizontally

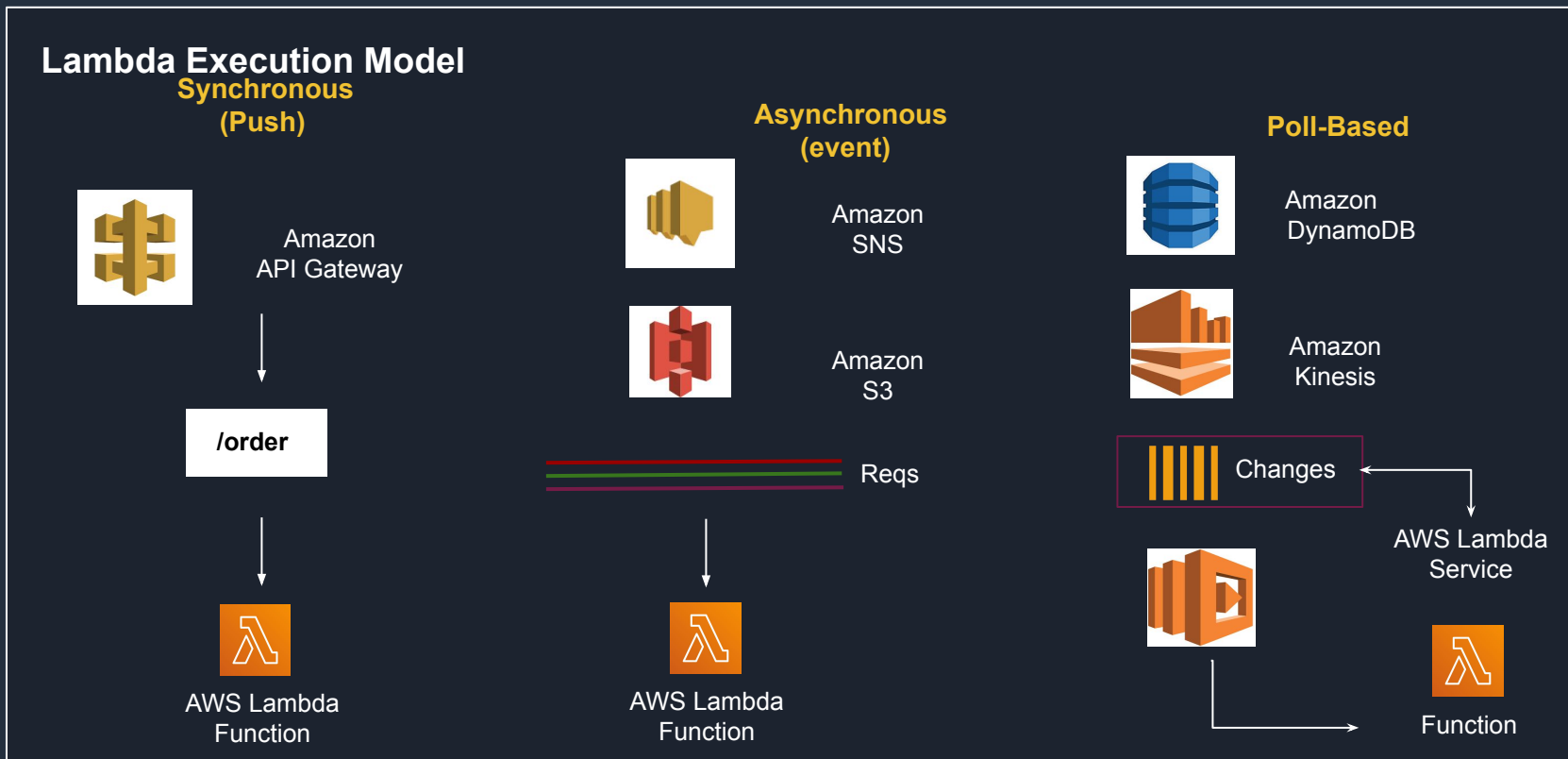


What is AWS SNS?

- Publishers send message to a SNS topic and multiple subscribers listen to the SNS topic notifications
- Each subscriber to the topic will get all the messages
- Up to 10,000,000 subscriptions per topic, 100,000 topics limit
- SNS Subscribers can be:
 - HTTP / HTTPS (with delivery retries – how many times)
 - Emails, SMS messages, Mobile Notifications
 - SQS queues (fan-out pattern), Lambda Functions (write-your-own integration)

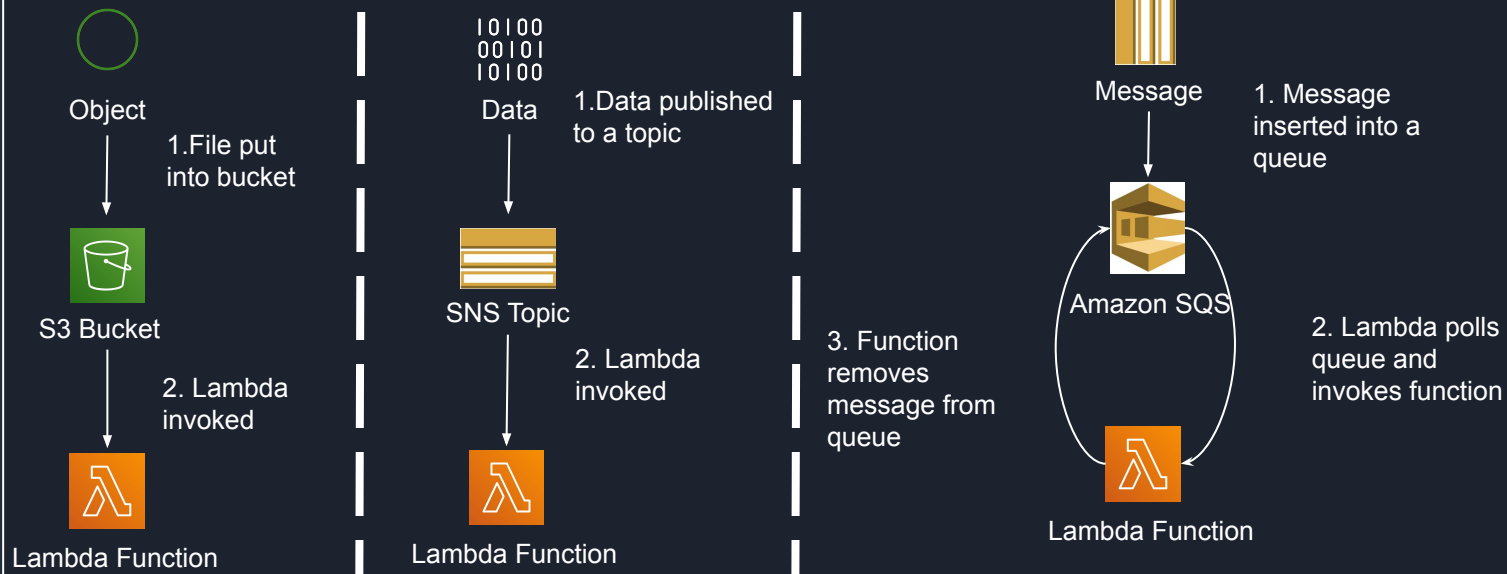


Lambda Invocation/Execution Model



Different Serverless Architecture

Serverless Architectures



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



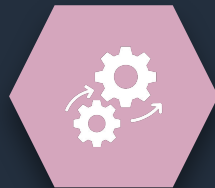
Chatbots

- 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 - 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 - Lambda with 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!