AWS Cloud Practitioner Training
Module 05
Serverless Solutions with Lambda

Instructor: Tim Platt, Cloud Solution Architect

DynamoDB

DynamoDB

A Non-relational (NoSQL) database – Internet scale and accessed as a WEB SERVICE.

Key Points

- Based on Key/Value access pattern (Does that sound familiar?)
- No upper limit on table size or number of rows (called “Items” in DynamoDB)
- Massive, internet scale read/write throughput (if you need it)
- Significantly different than relational database. You don’t use SQL. DynamoDB has its own Application Programming Interface (API) with GETs, PUTs, QUERIES, SCANS. Accessed over the Internet as a WEB SERVICE.
- Originally invented for the Amazon.com Shopping Cart
- SUPERPOWER: Schema-less database – great for varied, diverse datasets and JSON
- SUPERPOWER: Single digit millisecond latency for GETs and PUTs (single Item access using a key) at any scale
- SUPERPOWER: Managed service using multiple AZs automatically. No upgrades or maintenance to deal with.

NOTE: Due to the extreme differences – migrating from Relational database to DynamoDB is a lot of work

Let’s create a DynamoDB Table to see how it is different than SQL databases…

AWS Lambda

Serverless Compute

Run code on demand without managing servers

Key Points

- Upload function code (C#, Java, Python, JavaScript, etc.) and it can be triggered (run) on demand by the Lambda service.
- Scales automatically with no ASG or Load Balancer needed
- SUPERPOWER: Only pay while the code is running! Very efficient for sporadic or highly variable loads
- SUPERPOWER: Make it a REST API by using API Gateway or trigger it when an object is created in an S3 bucket, or when a message hits a SQS queue, or an SNS topic – there’s many, many ways to trigger the Lambda to run.

BUT …

- Function code can only run for up to 15 minutes
- The execution environment is ephemeral (temporary) – your code must be stateless (store state in a database or S3)
- COLD STARTs happen when the Lambda service doesn’t have a previously used “WARM” environment and there’s a little bit of delay (not much) before your code starts running

Let’s create a simple Lambda function to understand how it works

Simple Notification Service (SNS)

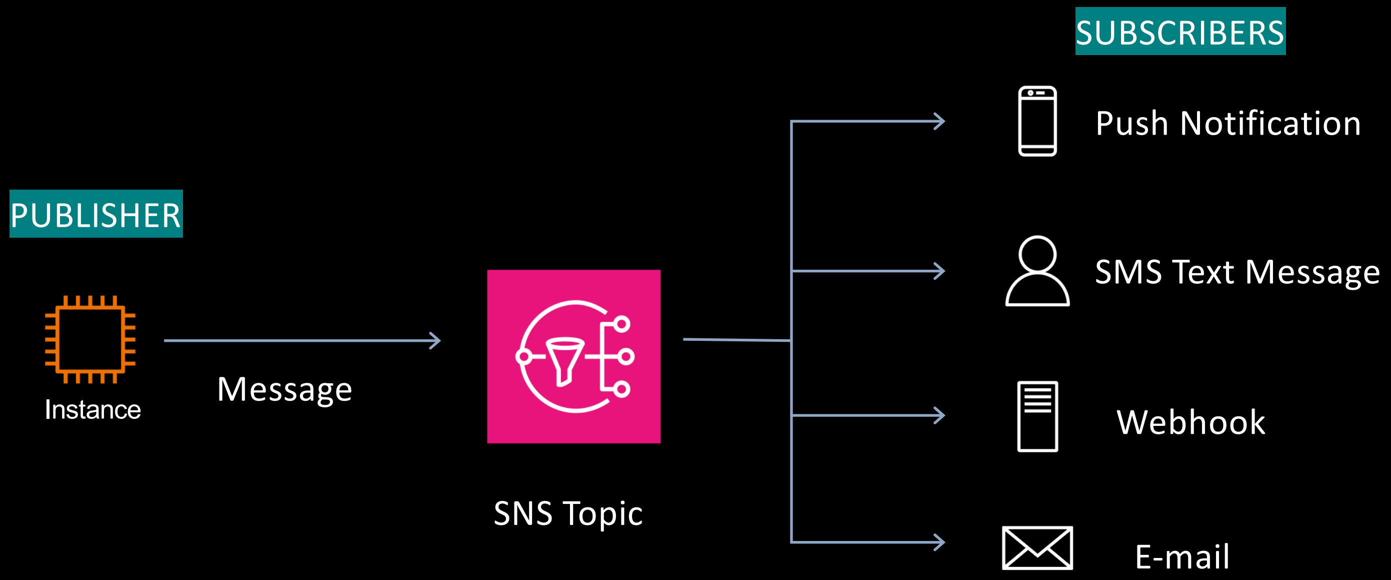
Notifications

Send notifications using Publish and Subscribe (Pub-Sub) paradigm

Key Points

- Your application or workflow can PUBLISH a message to a TOPIC and any and all SUBSCRIBERS will receive a notification
- Sometimes also known as a “fan-out” because ONE publisher can notify MANY subscribers.
- The messages are “pushed” to the subscribers – they don’t need to poll constantly.
- Publisher is “decoupled” from the Subscribers and needs no knowledge of the subscribers
- Very simple API to use
- SUPERPOWER: Very scalable – great for sending many, many, many messages to many, many, many subscribers
- SUPERPOWER: Send emails, text/SMS messages, push notifications – or invoke Lambda and other things
- SUPERPOWER: Use SNS topics to build simple event driven architectures. For example, an SNS Topic might push a notification to a Lambda function

SNS Architecture – Fan Out (One to Many)



Simple Queue Service (SQS)

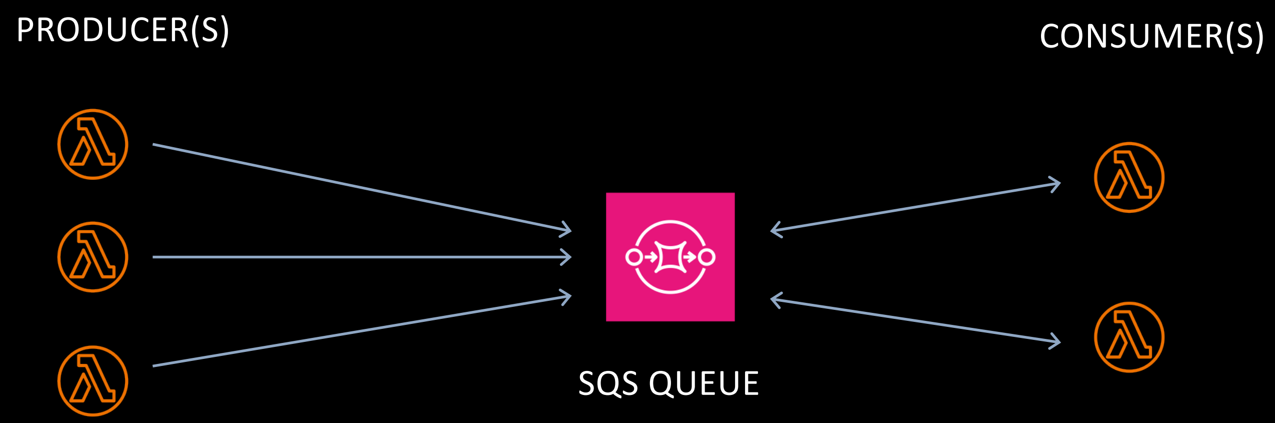
Queueing

Use a highly scalable QUEUE to buffer or store messages until you can process them

Key Points

- A durable queue that you can use to temporarily store messages (like incoming customer orders) until you can process them.
- One or more PRODUCERS will put MESSAGES into the QUEUE. One or more CONSUMERS will process the messages – and delete them from the queue when they are done.
- The consumers will POLL to find any unprocessed messages in the queue.
- There are two basic types of Queue : Standard (AT LEAST ONCE DELIVERY) and FIFO – First In First Out – which guarantees delivery ONCE and in ORDER that the messages arrived.
- Standard queue scales much higher – but you must deal with potential message duplication
- SUPERPOWER: Very scalable and highly reliable
- SUPERPOWER: SQS queues can hold messages for up to 14 days.

SQS Architecture



PRODUCER(S) are constantly polling, checking for messages in the queue.
They process each one, then delete it from the queue.

SNS Versus SQS

They are two very different things – even though both deal with ”Messages”

Persistence

SNS is not a buffer – there's no storage – the messages are sent immediately and not retained.

SQS can store messages for up to 14 days – allowing time for our consumers to get caught up

Push vs Poll

SNS – Messages are PUSHed to the subscribers when they arrive

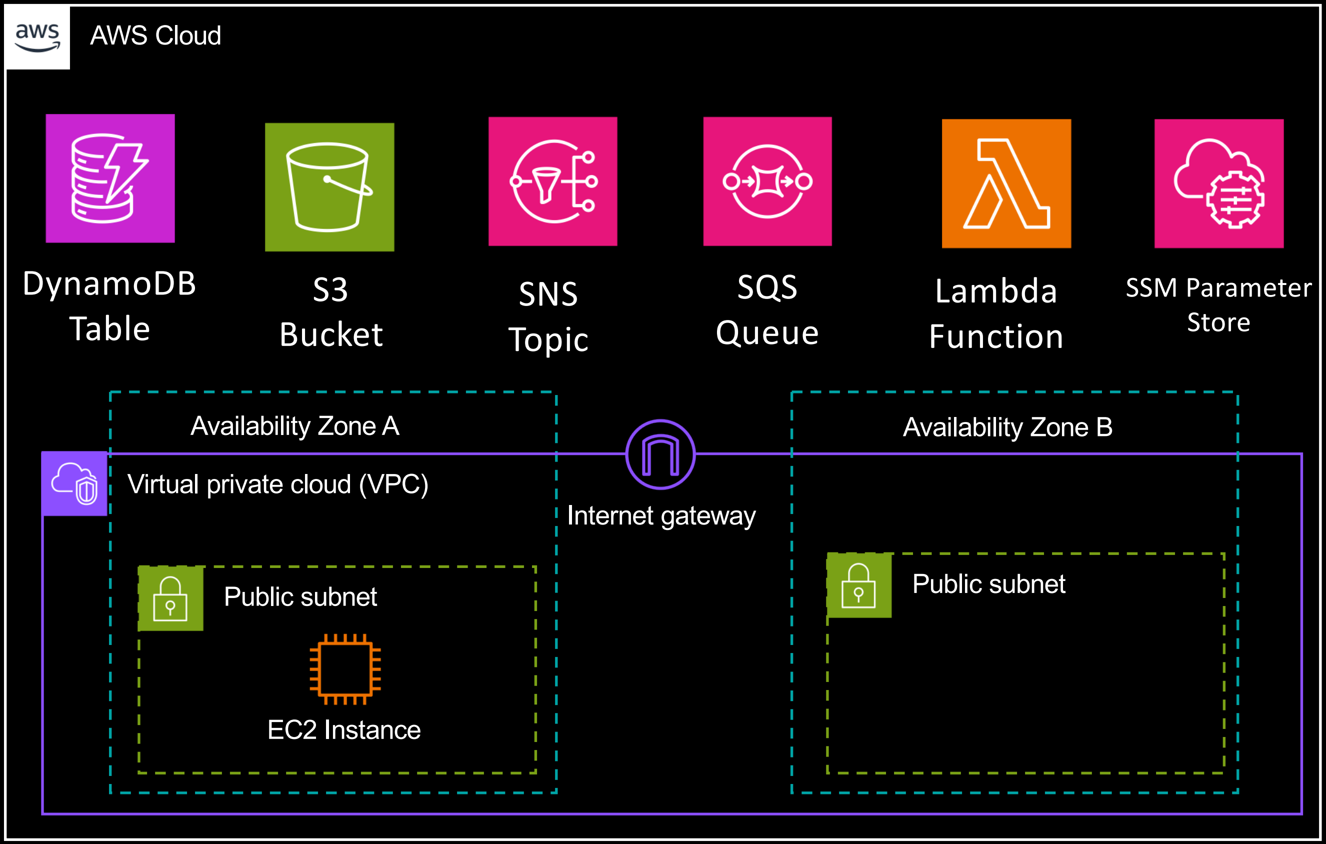
SQS – The consumers must poll repeatedly (in a loop) to constantly look for new messages

Distribution

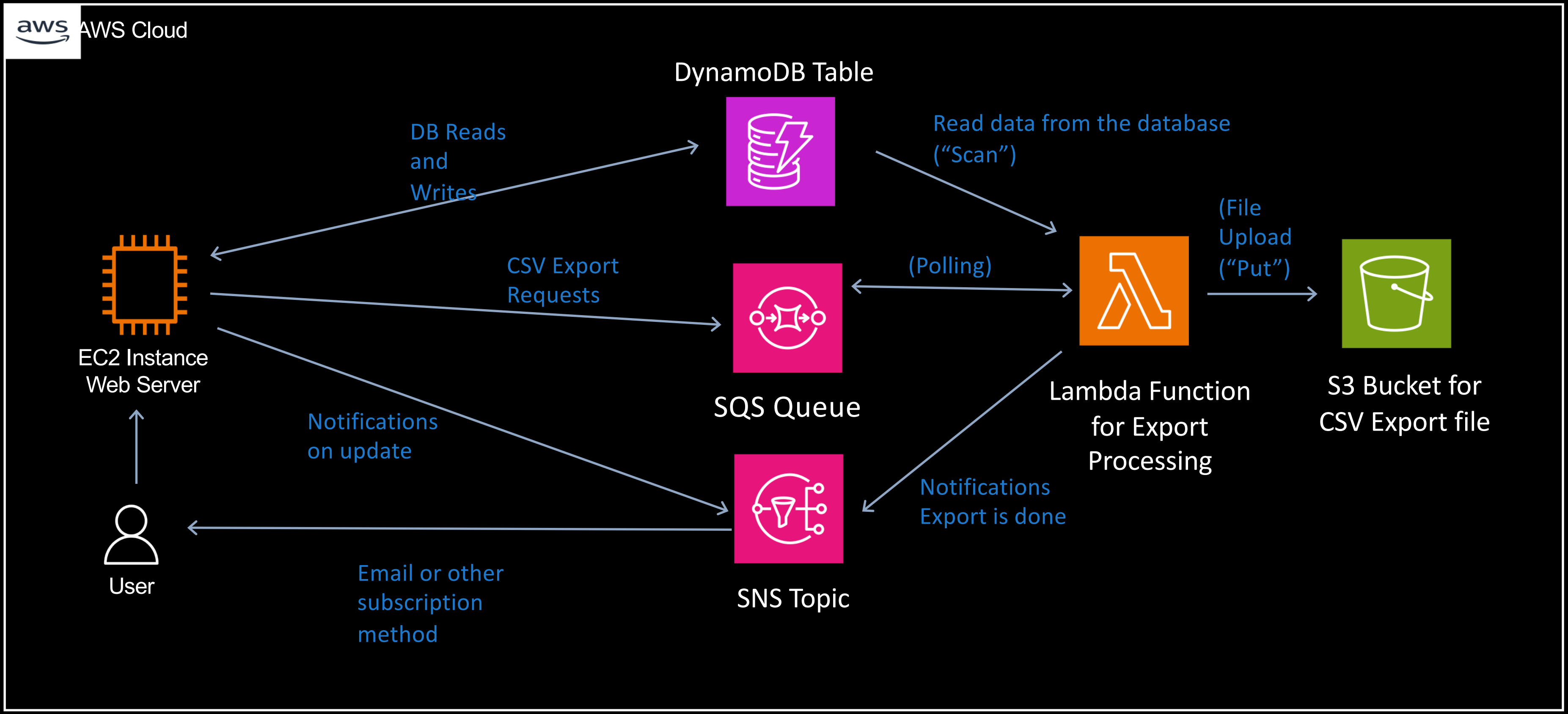
SNS Messages can go to many, many subscribers (One to Many)

SQS Messages are intended to be “consumed” by a **SINGLE** Consumer process (Imagine processing a customer order, for example)

Serverless Services – live in the REGION, not your VPC



Application Architecture



AWS Cloud
DynamoDB Table
DB Reads and Writes
Read data from the database ("Scan")
EC2 Instance Web Server
CSV Export Requests
SQS Queue
(Polling)
Lambda Function for Export Processing
(File Upload ("Put")
S3 Bucket for CSV Export file
Notifications on update
SNS Topic
Notifications Export is done
User
Email or other subscription method

Links

- Simple Notification Service (SNS) Developer Guide: https://docs.aws.amazon.com/sns/latest/dg/welcome.html
- Simple Queue Service (SQS) Developer Guide: https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html
- AWS Lambda Developer Guide: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
- Lambda Execution environment lifecycle: https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtime-environment.html
- Lambda Example – File Processing: https://docs.aws.amazon.com/lambda/latest/dg/file-processing-app.html
- Lambda Example – Cron Job: https://docs.aws.amazon.com/lambda/latest/dg/scheduled-task-app.html