

# **AWS Solution Architect Training**

## **Module 06**

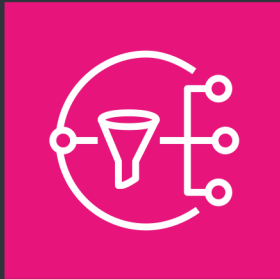
### **Messaging Services & AWS Lambda**

Instructor: Tim Platt, Cloud Solution Architect

# Simple Notification Service (SNS)

## Notifications

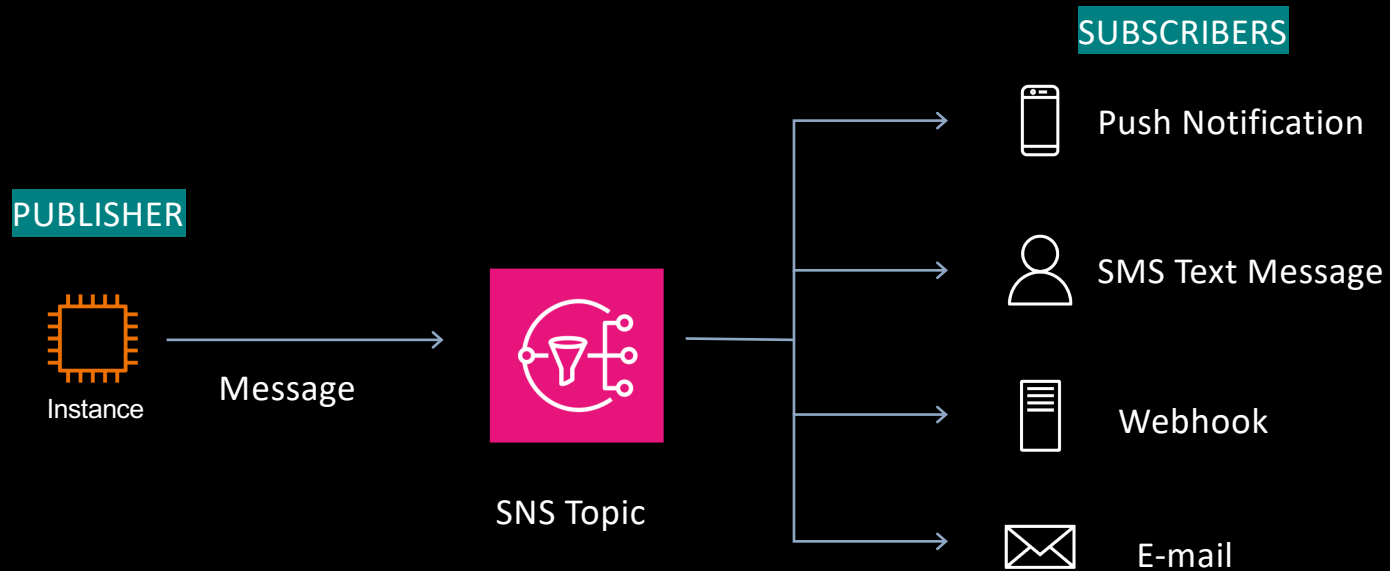
Send notifications using Publish and Subscribe (Pub-Sub) paradigm



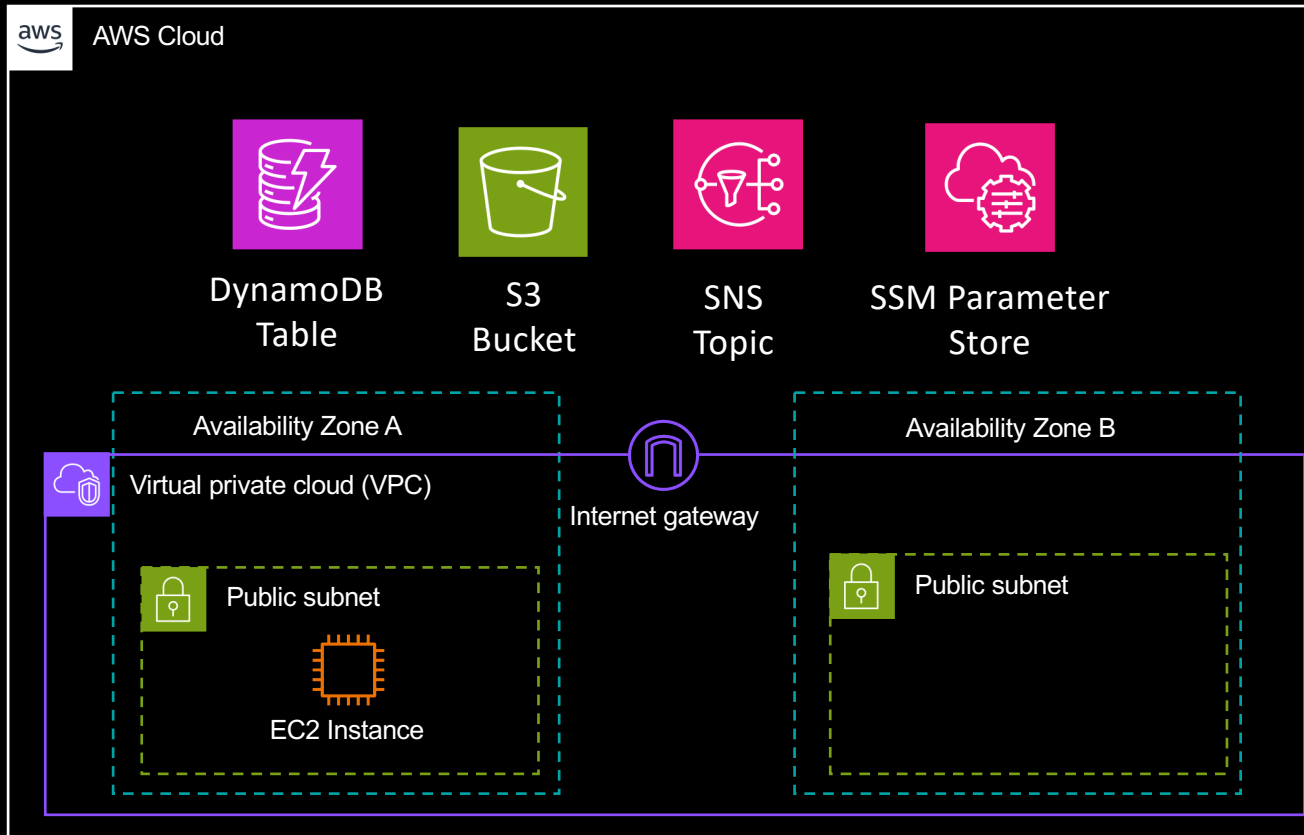
## Key Points

- 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)



# Let's add a NOTIFICATION system to our TODO App!



## Why?

- SNS is a simple, scalable way for us to add notifications to our application
- The app doesn't need to know (or care) how many subscribers there are – we'll just PUSH a notification every time a task is closed.
- We're going to use Systems Manager (SSM) to set "Feature Flags"

**REMEMBER:** Our EC2 instance will need IAM Permissions to be able to interact with these new services (SNS, SSM, etc.)

# AWS Systems Manager (SSM)

## Notifications

Centrally view, manage, and operate nodes (EC2 instances mostly) at scale



## Key Points

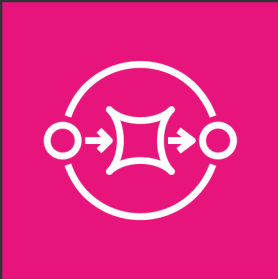
- All the functions you would need to manage virtual machines including:
  - Easy remote access via “Session Manager”
  - Patching and upgrades via “Patch Manager”
  - Remote Administration and Automated updates via “Automation Documents” and “Run Command”
  - Inventory of software versions installed
  - Centralized collection of configuration settings (Parameter Store)
  - And much more...

We’ve already used “Session Manager” to SSH in, but we’ll also use Parameter Store

# Simple Queue Service (SQS)

## Queueing

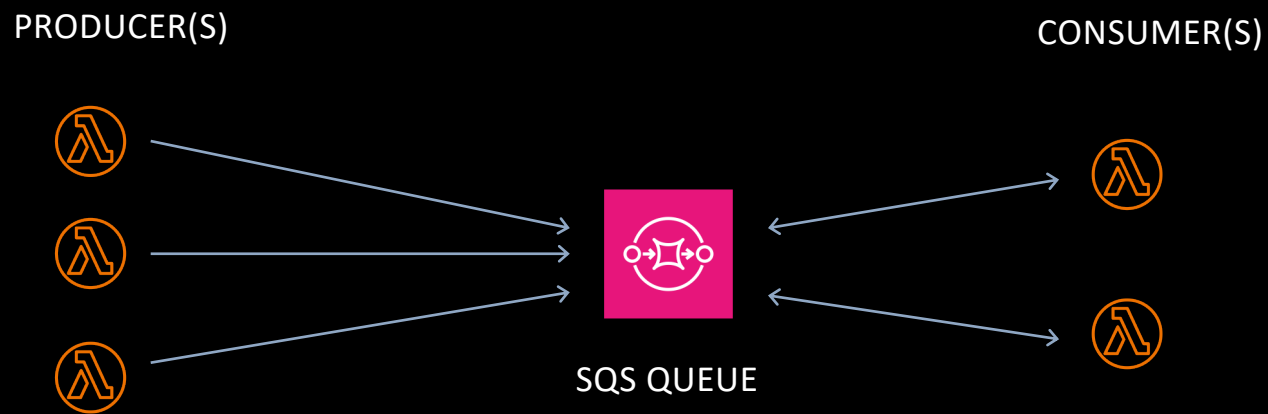
Use a highly scalable QUEUE to buffer or store messages until you can process them



## 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)



# AWS Lambda

## Serverless Compute

Run code on demand without managing servers



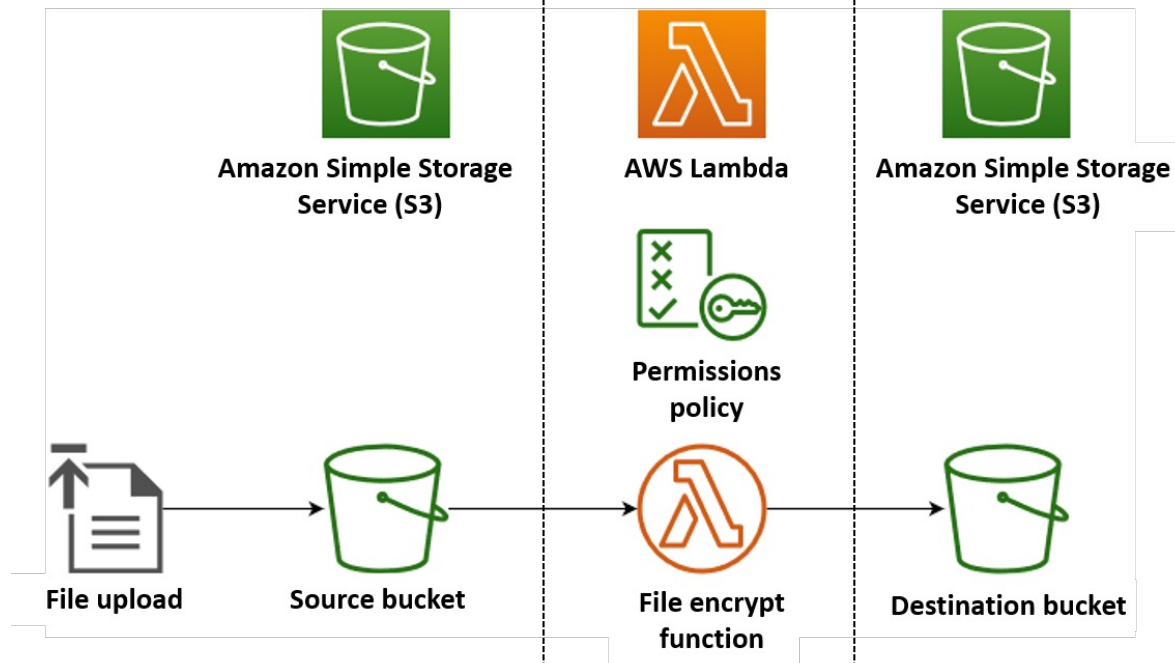
## Key Points

- 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.

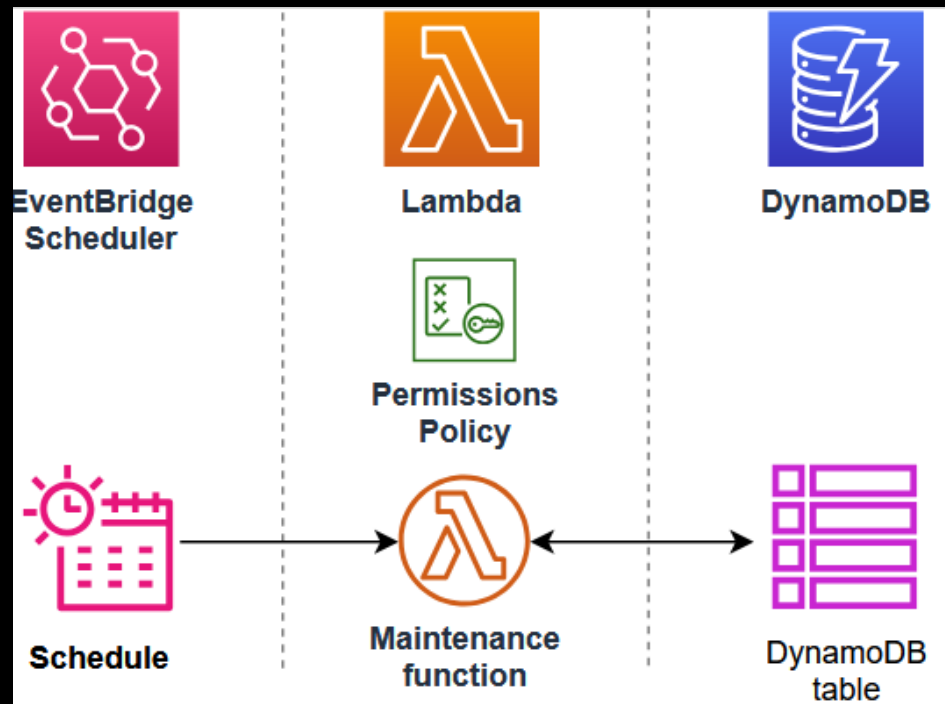
BUT ...

- 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

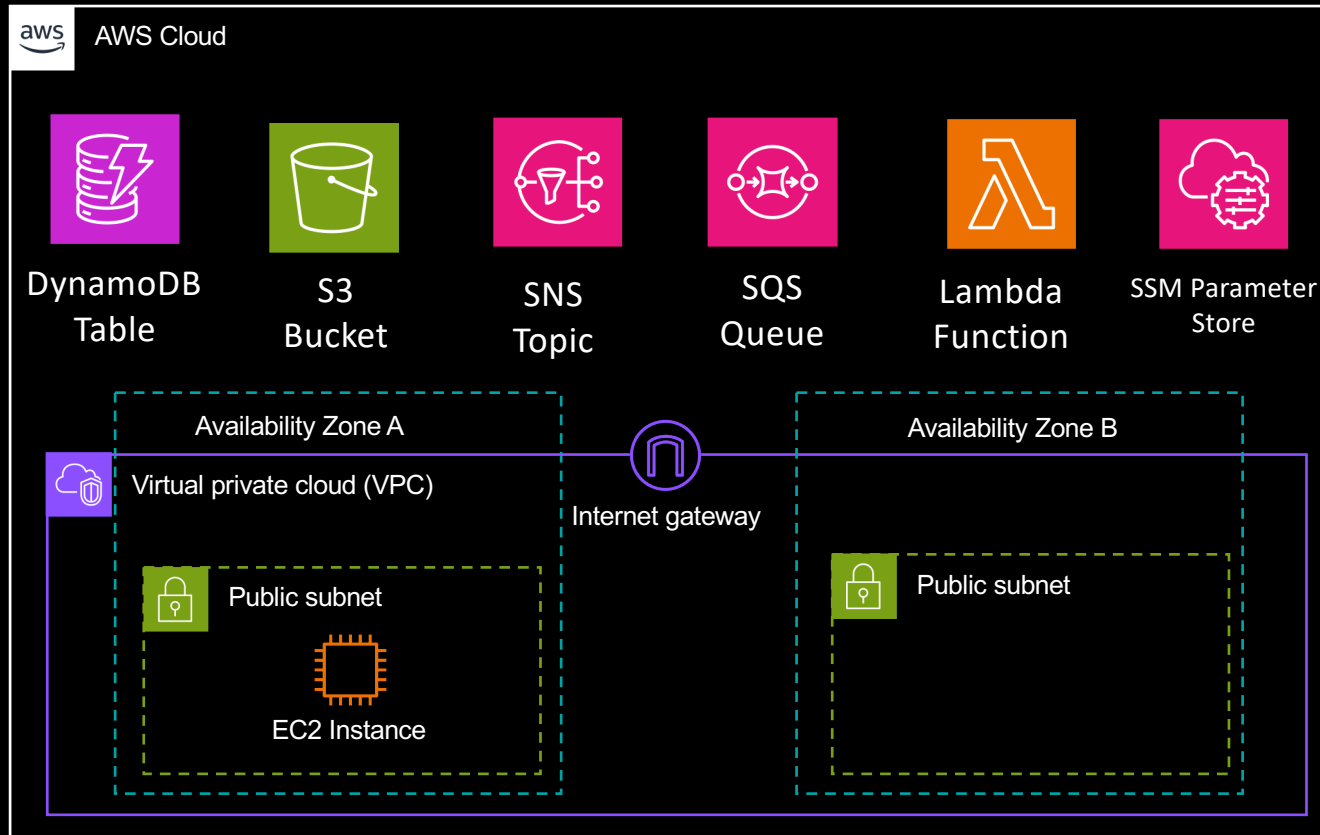
# Lambda example – encrypting files after upload



## Lambda Example – Scheduled Job (Like a Cron job)



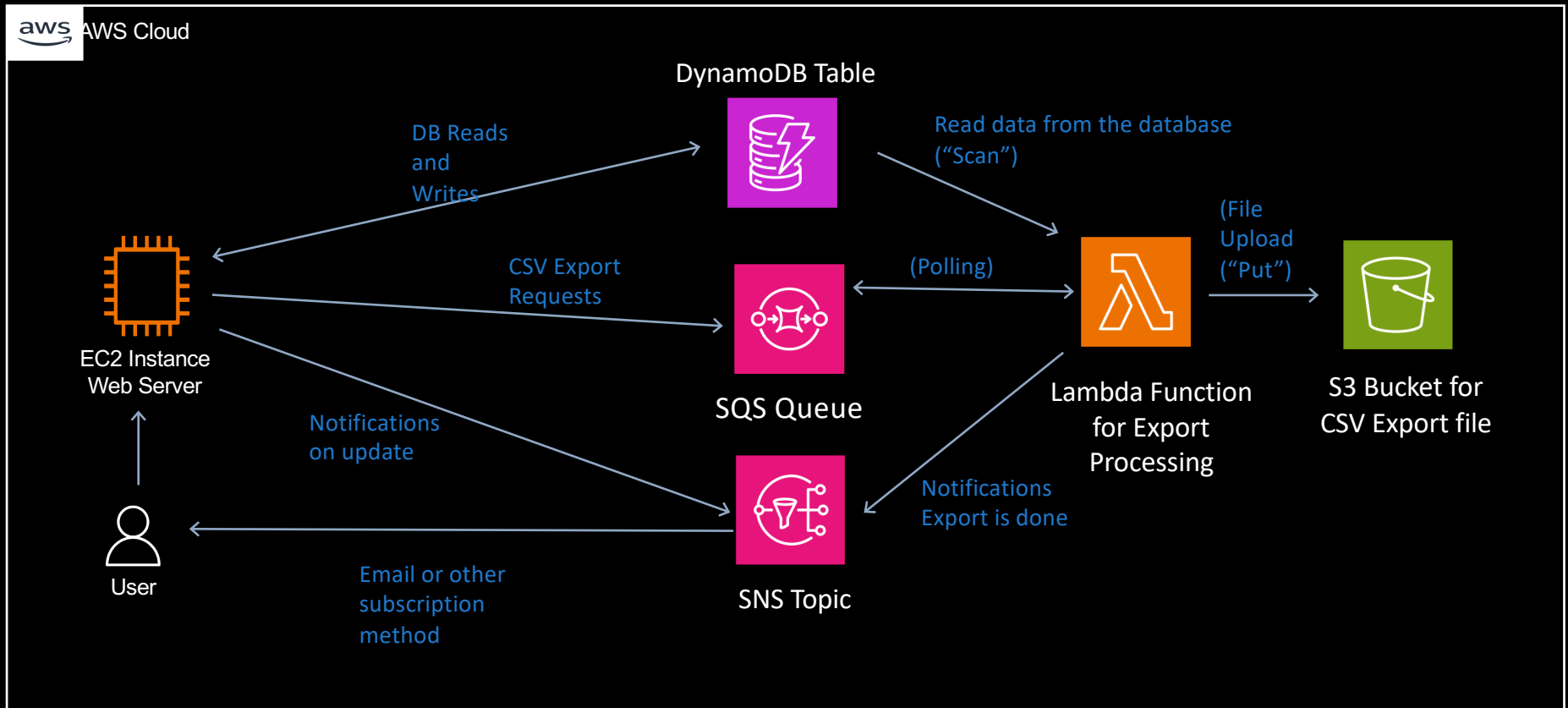
# Let's add a QUEUE for those long running exports



## Why?

- We'll have the app add a message to the queue
- The lambda service will be POLLING and will execute a Lambda Function to handle the new message.
- Once the message is processed, the Lambda will delete it – so it isn't accidentally processed twice!

# Application Architecture



# 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>