Serverless Streams, Topics, Queues, & APIs!

How to Pick the Right Serverless Application Pattern

Chris Munns – Senior Developer Advocate – AWS Serverless

August 2018

About me:

Chris Munns - munns@amazon.com, @chrismunns

- Senior Developer Advocate - Serverless
- New Yorker
- Previously:
  - AWS Business Development Manager – DevOps, July ’15 - Feb ‘17
  - AWS Solutions Architect Nov, 2011- Dec 2014
  - Formerly on operations teams @Etsy and @Meetup
  - Little time at a hedge fund, Xerox and a few other startups
- Rochester Institute of Technology: Applied Networking and Systems Administration ’05
- Internet infrastructure geek

Why are we here today?

https://secure.flickr.com/photos/mgifford/4525333972

Serverless means…

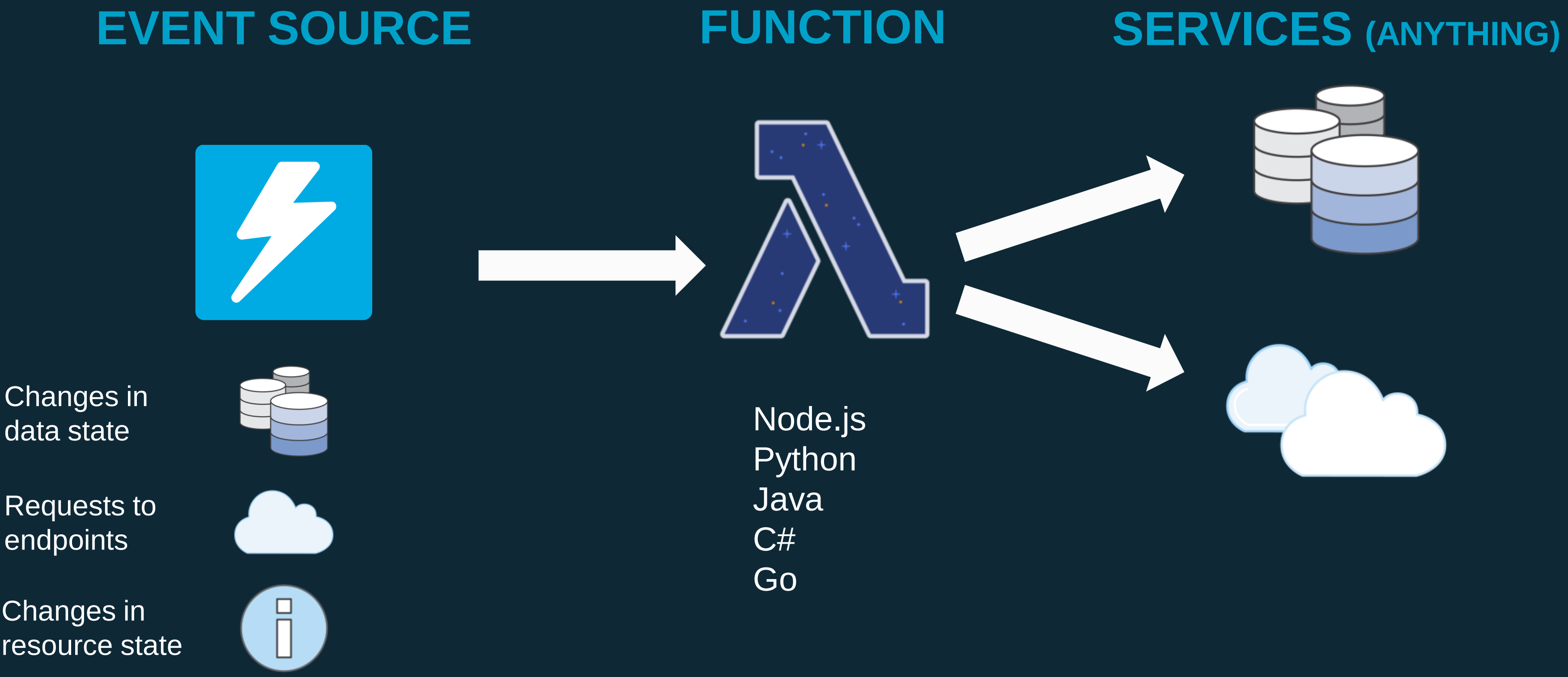
No servers to provision or manage

Scales with usage

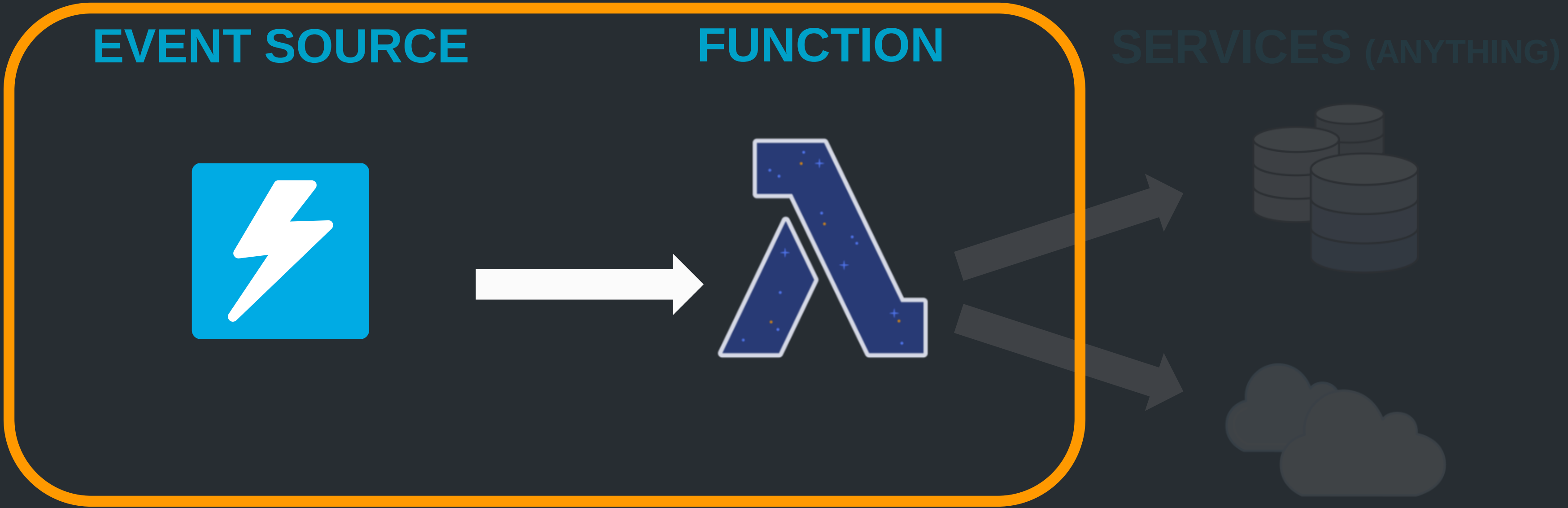
Never pay for idle

Availability and fault tolerance built in

Serverless applications



Serverless applications



Common Lambda use cases

Web Applications

- Static websites
- Complex web apps
- Packages for Flask and Express

Backends

- Apps & services
- Mobile
- IoT

Data Processing

- Real time
- MapReduce
- Batch

Chatbots

- Powering chatbot logic

Amazon Alexa

- Powering voice-enabled apps
- Alexa Skills Kit

IT Automation

- Policy engines
- Extending AWS services
- Infrastructure management

Common Lambda use cases

Web Applications

- Static websites
- Complex web apps
- Packages for Flask and Express

Backends

- Apps & services
- Mobile
- IoT

Data Processing

- Real time
- MapReduce
- Batch

Chatbots

- Powering chatbot logic

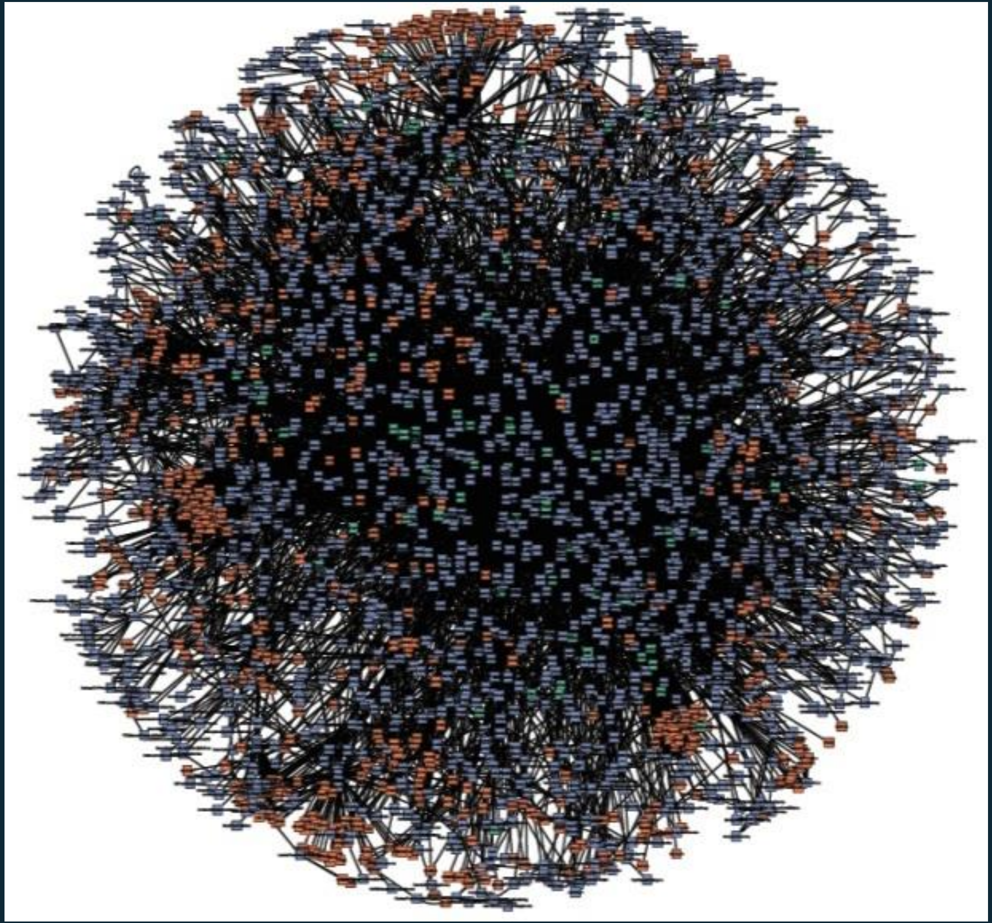
Amazon Alexa

- Powering voice-enabled apps
- Alexa Skills Kit

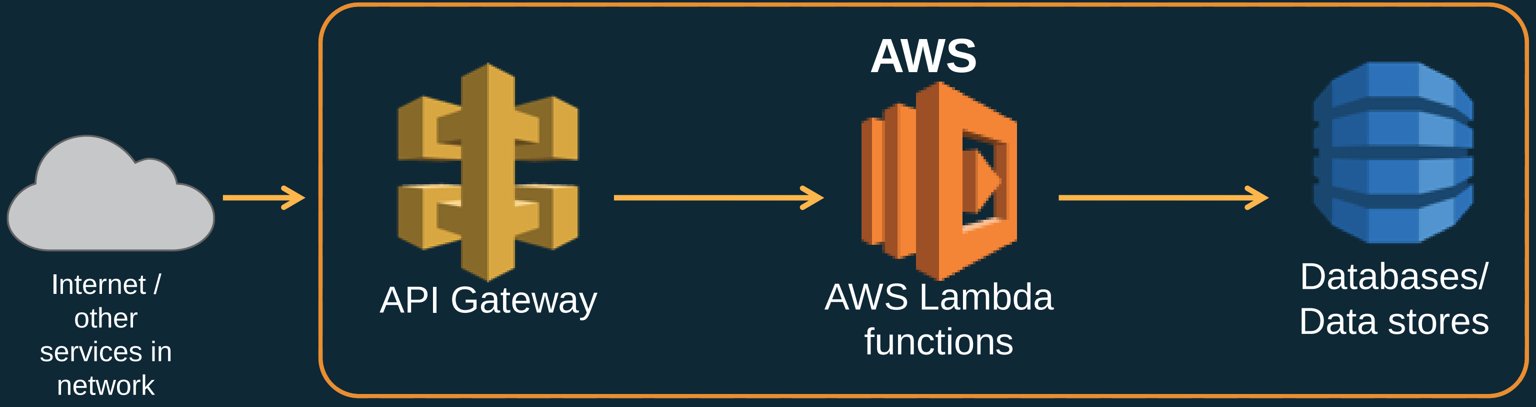
IT Automation

- Policy engines
- Extending AWS services
- Infrastructure management

Microservices at Amazon



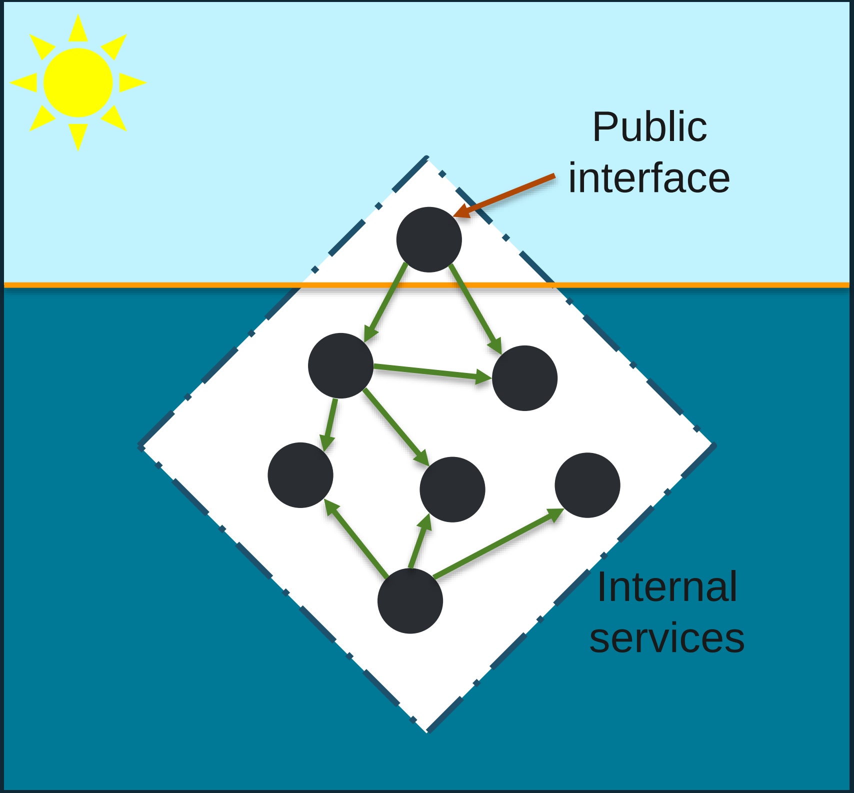
Basic Serverless API based Microservice



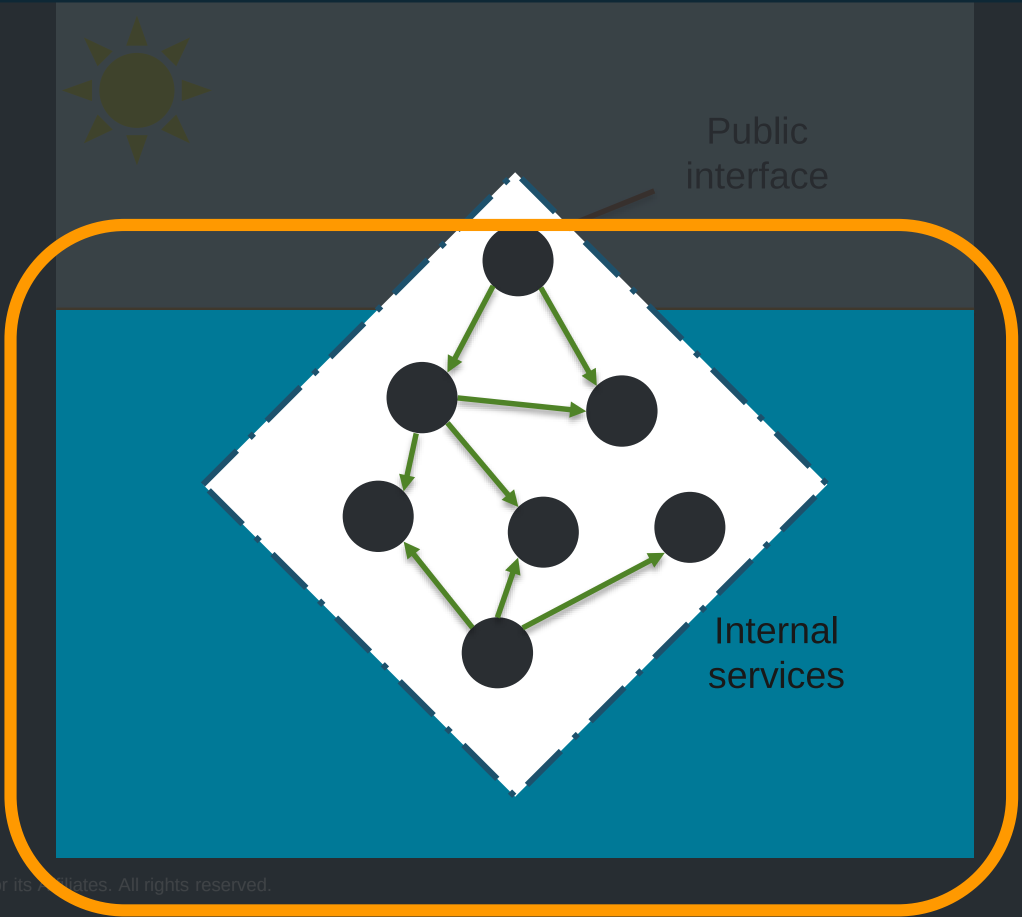
The microservices "iceberg"

Common question: "Should every service of mine talk to another using an API?"

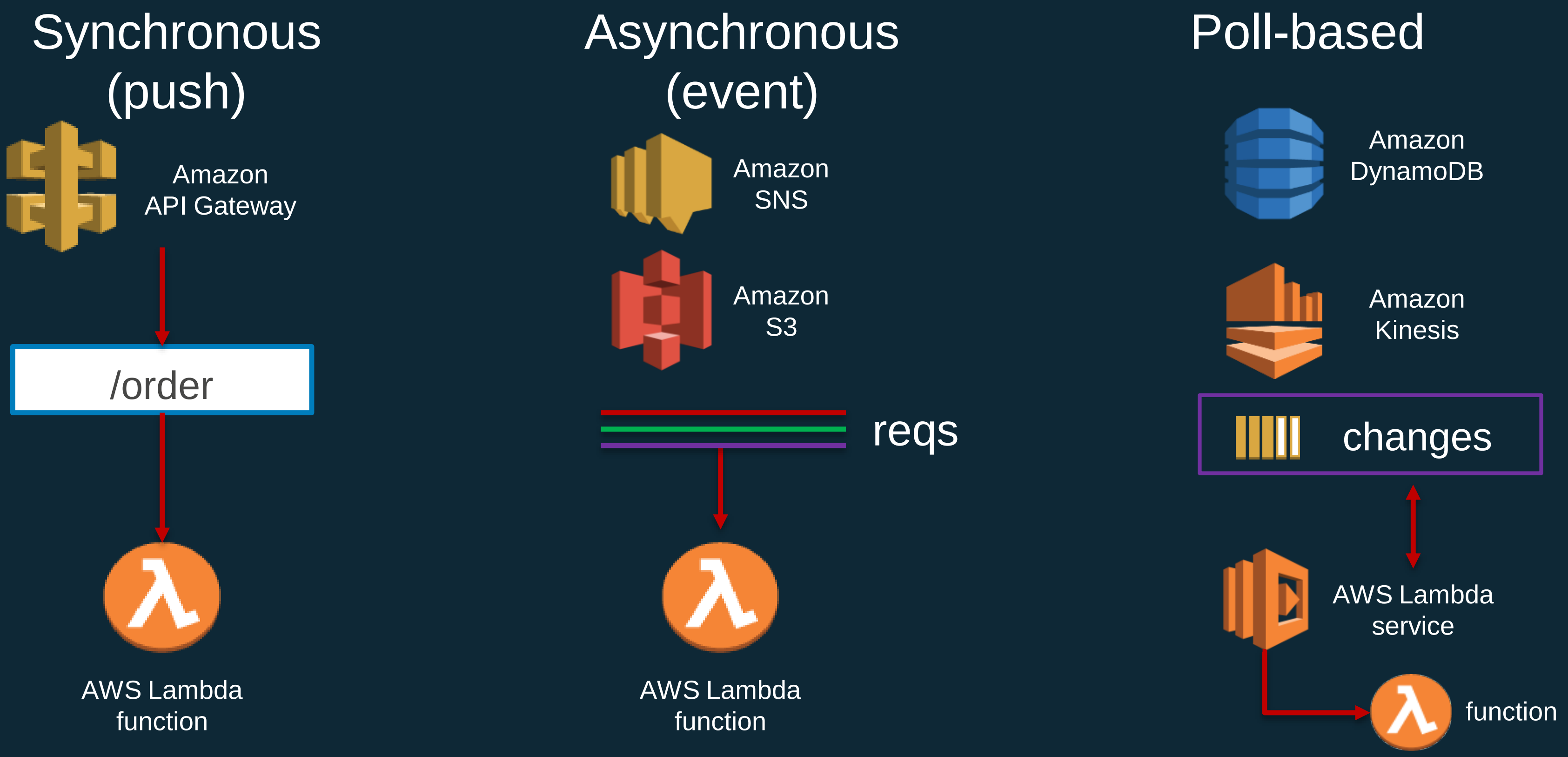
Maybe not!: Most microservices are internal only for a given product supporting their customer facing features. They may only need to pass messages to each other that are simple events and not need a full fledged interactive API.



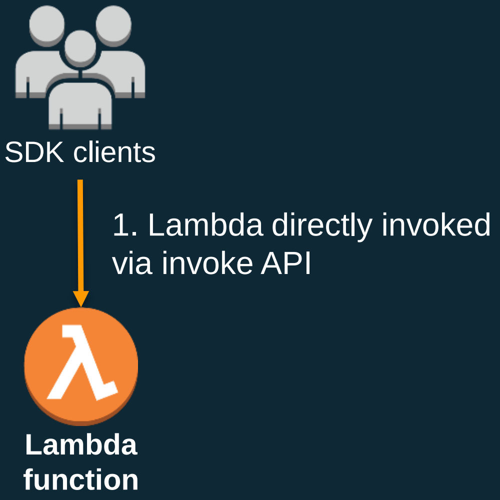
Focusing below the water line



Lambda execution model



Lambda API



API provided by the Lambda service

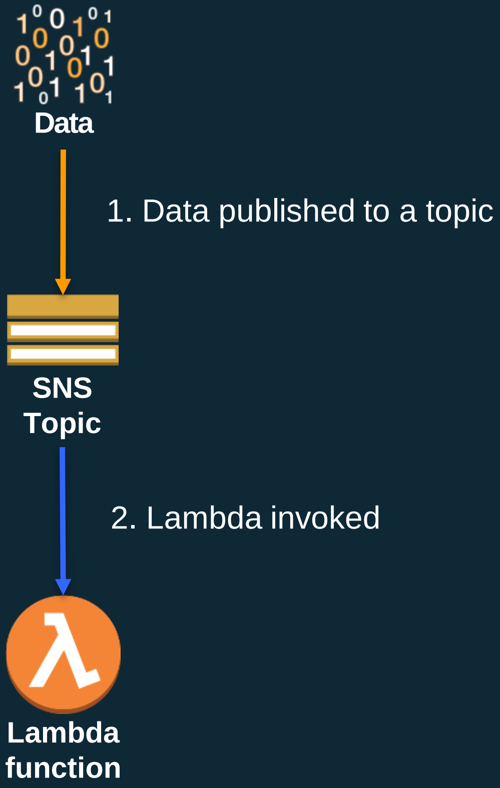
Used by all other services that invoke Lambda across all models

Supports sync and async

Can pass any event payload structure you want

Client included in every SDK

Amazon SNS + Lambda



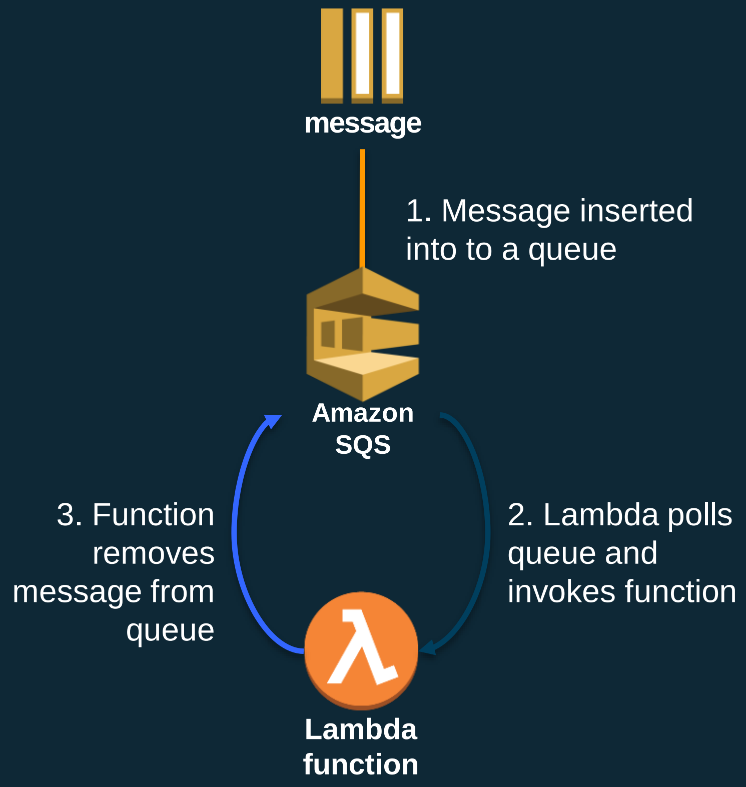
Simple, flexible, fully managed publish/subscribe messaging and mobile push notification service for high throughput, highly reliable message delivery

Messages are published to a Topic

Topics can have multiple subscribers (fanout)

Messages can be filtered and only sent to certain subscribers

Amazon SQS + Lambda


message
1. Message inserted into to a queue
Amazon SQS
2. Lambda polls queue and invokes function
3. Function removes message from queue
Lambda function

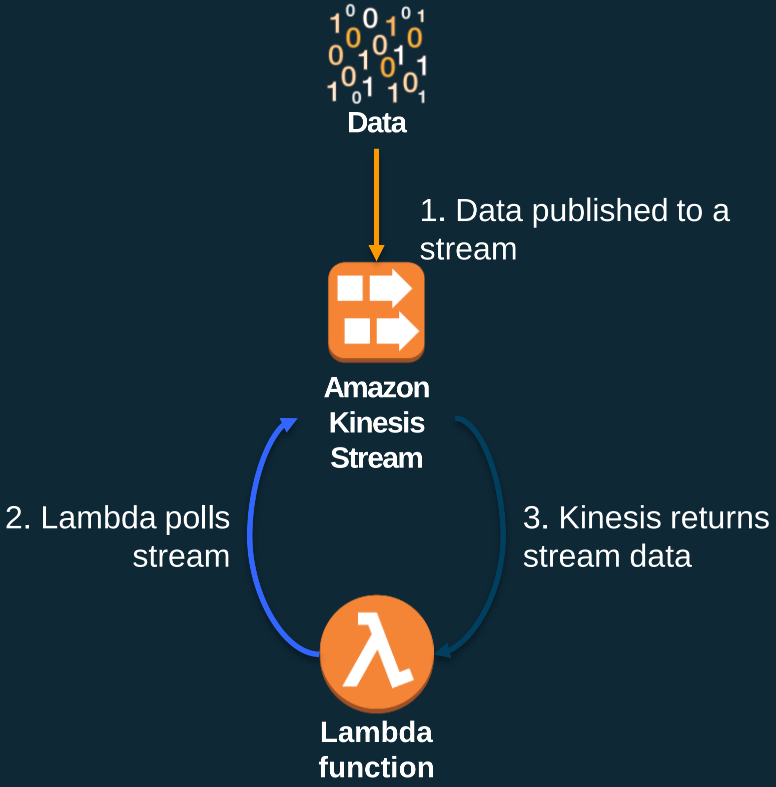
Simple, flexible, fully managed message queuing service for reliably and continuously exchanging any volume of messages from anywhere

Processed in batches

At least once delivery

Visibility timeout allows for handling of failures during processing

Amazon Kinesis Streams + Lambda



Fully managed, highly scalable service for collecting and processing real-time data streams for analytics and machine learning

Stream consists of shards with a fixed amount of capacity and throughput

Lambda receives batches and potentially batches of batches

Can have different applications consuming the same stream

Ways to compare

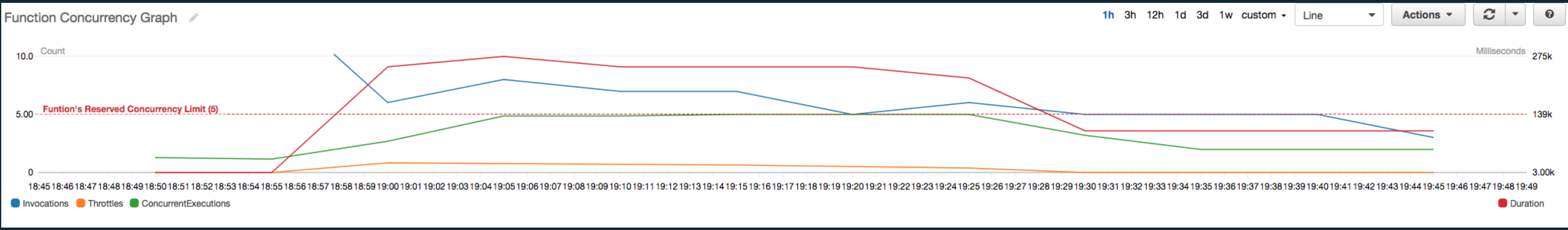
- Scale/Concurrency controls
- Durability
- Persistence
- Consumption models
- Retries
- Pricing

Scaling/Concurrency Controls

Service	Scaling controls
Lambda API	Concurrency is point in time, not TPS, can go to 0 up through maximum for account per region and is shared for all functions in a functions in a region. By default no per function concurrency throttle is set.
SNS	Service automatically scales, use Lambda Per Function Concurrency Concurrency setting to control downstream consumption.
SQS	Service automatically scales, use Lambda trigger Batch size setting setting and Per Function Concurrency setting to control downstream downstream consumption.
Kinesis Streams	Shards in a stream: One shard provides ingest capacity of 1MB/sec 1MB/sec or 1000 records/sec, up to 2MB/sec of data output.

Lambda Per Function Concurrency controls

- Concurrency a shared pool by default
- Separate using per function concurrency settings
  - Acts as reservation
- Also acts as max concurrency per function
  - Especially critical for data sources like RDS
- “Kill switch” – set per function concurrency to zero

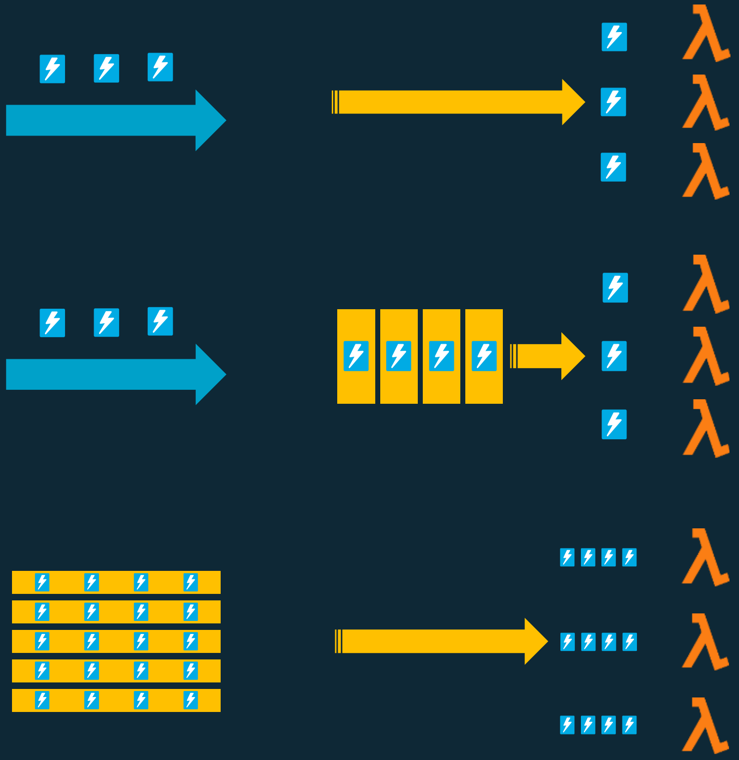


Concurrency across models

SNS/API
No event store

Queue based

Stream based



Concurrency vs. Latency

Streams

- Maximum theoretical throughput:
  # shards * 2 MB / (s)
- Effective theoretical throughput:

  (# shards * batch size (MB)) /

(function duration (s) * retries until expiry)

- If put / ingestion rate is greater than the theoretical throughput, consider increasing number of shards while optimizing function duration to increase throughput

Everything else

- Maximum Processing rate :

  Maximum concurrency / average duration (events per second)
- Effective Processing rate :

  Effective concurrency / average duration (events per second)
- Use concurrency metric and duration metric to estimate processing time

Durability

Service	Durability of requests “in flight”
Lambda API	Lambda API is built to be highly available but offers no durability of requests, client would need to handle failures/retries.
SNS	*SNS provides durable storage of all messages that it receives. Upon receiving a publish request, SNS stores multiple copies (to disk) of the message across multiple Availability Zones before acknowledging receipt of the request to the sender.
SQS	*Amazon SQS stores all message queues and messages within a single, highly-available AWS region with multiple redundant Availability Zones (AZs), so that no single computer, network, or AZ failure can make messages inaccessible.
Kinesis Streams	*Amazon Kinesis Data Streams synchronously replicates data across three availability zones, providing high availability and data durability

*Taken from relevant service FAQs

Durability

Service	Durability of requests "in flight"
Lambda API	Lambda API is built to be highly available but offers no durability of requests, client would need to handle failures/retries.
SNS	*SNS provides durable storage of all messages that it receives. Upon receiving a publish request, SNS stores multiple copies (to disk) of the message across multiple Availability Zones before acknowledging receipt of the request to the sender.
SQS	*Amazon SQS stores all message queues and messages within a single, highly-available AWS region with multiple redundant Availability Zones (AZs), so that no single computer, network, or AZ failure can make messages inaccessible.
Kinesis Streams	*Amazon Kinesis Data Streams synchronously replicates data across three availability zones, providing high availability and data durability

Short version: Data is replicated across multiple Availability Zones for all 3 of these services.

*Taken from relevant service FAQs

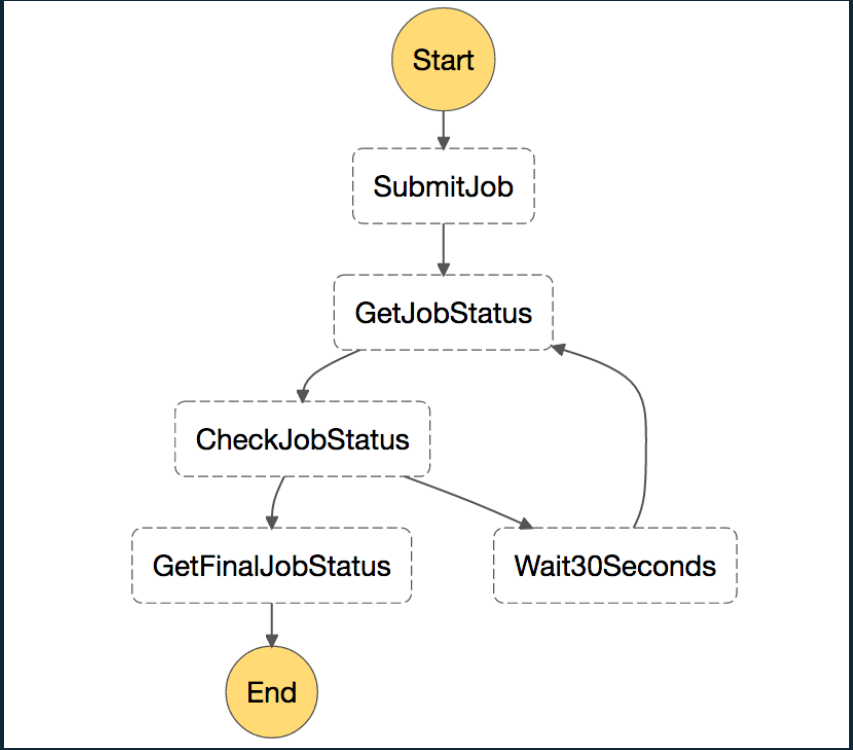
Persistence

Service	Persistence of requests "in flight"
Lambda API	No formal persistence model
SNS	No formal persistence model beyond delivery retry logic that extends up through potentially 13 hours
SQS	By default messages are stored for 4 days. This can be modified to as little as 60 seconds up to 14 days by configuring a queue's MessageRetentionPeriod attribute
Kinesis Streams	By default data is stored for 24 hours. You can increase this up to 168 hours (7 days). Extended data retention costs $0.02 per Shard Hour above 24 hours

Consumption

Service	Invocation model	Guidance
Lambda API	Can be sync or async from client to a single single invocation	For complicated Lambda to Lambda workflows use AWS Step Functions
SNS	Async to Lambda. SNS can "fanout" to multiple subscribing Lambda functions the same message	Use Message Filtering to control which messages go to which subscribers. Use Message delivery status to track failures
SQS	Lambda service polls messages from queue and invokes Lambda on your behalf. Scales polling based on inflight messages.	Can call message delete from within your code code or let the service handle it via successful successful Lambda function execution
Kinesis Streams	Lambda service polls messages from streams streams and invokes Lambda on your behalf. behalf. Can run multiple applications to consume the same stream for different needs needs	Use the AWS Kinesis Client Library. Configure Configure batch size so that your function has has enough time to complete processing of records (which might be batches on ingest as as well)

Keep orchestration out of code.



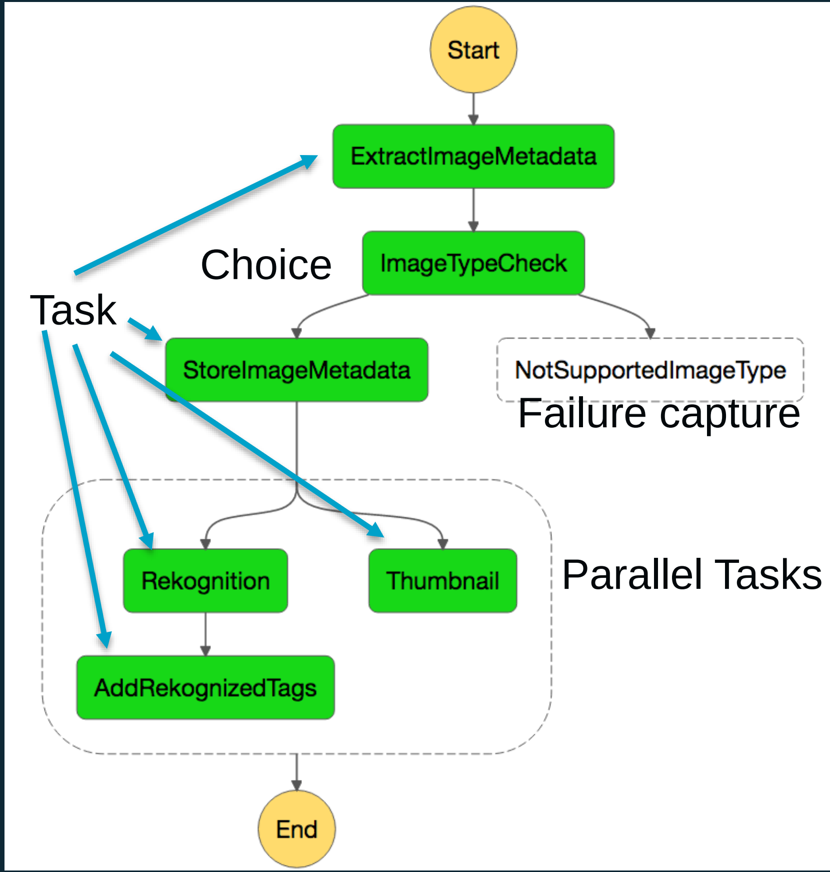
AWS Step Functions

“Serverless” workflow management with zero administration:

Makes it easy to coordinate the components of distributed applications and microservices using visual workflows

Automatically triggers and tracks each step, and retries when there are errors, so your application executes in order and as expected

Can handle custom failure messages from Lambda



Retry/failure handling

Service	Retry/failure capabilities
Lambda API	Retry/failure logic is client dependent for synchronous invocations. For asynchronous invocations are retried twice by the Lambda service.
SNS	If Lambda is not available, SNS will retry 2 times at 1 seconds apart, then 10 times times exponentially backing off from 1 seconds to 20 minutes and finally 38 times times every 20 minutes for a total 50 attempts over more than 13 hours before the before the message is discarded from SNS.
SQS	Messages remain in the queue until deleted. They are prevented by being accessed accessed by other consumers during a period of time known as the "visibility timeout". Successful Lambda invocations will cause deletions of messages automatically. If an invocation fails or doesn't delete a message during the visibility visibility timeout window it is made available again for other consumers.
Kinesis Streams	When using the Kinesis Client Library (KCL) it maintains a checkpoint/cursor of processed records and will retry records from the same shard in order until the cursor shows completion.

Lambda Dead Letter Queues

"By default, a failed Lambda function invoked asynchronously is retried twice, and then the event is discarded. Using Dead Letter Queues (DLQ), you can indicate to Lambda that unprocessed events should be sent to an Amazon SQS queue or Amazon SNS topic instead, where you can take further action." – https://docs.aws.amazon.com/lambda/latest/dg/dlq.html

- Turn this on! (for async use-cases)
- Monitor it via an SQS Queue length metric/alarm
- If you use SNS, send the messages to something durable and/or a trusted endpoint for processing
    - Can send to Lambda functions in other regions
- If and when things go "**boom**" DLQ can save your invocation event information

Pricing

Service	Model	Cost Per Mil	Factor	Other
Lambda API	Per request	$0.20*		
SNS	Per request	$0.50*	Each 64KB chunk of delivered data is billed as 1 request	No charge for deliveries to Lambda
SQS	Per request	$0.40*	Each 64 KB chunk of a payload is billed as 1 request	A single request can have from 1 to 10 messages
Kinesis Streams	Per Shard hour & per request PUT Payload Units	Shard per Hour = $0.015 PUT Payload Units $0.014	Each 25KB chunk of a payload (PUT Payload Units) are billed as 1 request	Enhanced Fanout and Extended Data Retention (beyond 24 hours) cost extra

* First 1 Million requests are free per month

Ways to compare

Scale/Concurrency controls

Durability

Persistence

Consumption models

Retries

Pricing

Tell me what to do already!

So what invocation resource is the right one for you?

How real time is your “real time” need?

- How synchronous is your synchronous workload? Would polling for updates after an async invocation work?

Does order matter?

Do multiple services need to feed off of the same data?

What does breaking your Lambda function due to a bad code deploy have impact on?

Think about the downstream!

- What happens when a downstream service fails?
- Is there the potentially to overwhelm a database or other service?

Tell me what to do already!

So what invocation resource is the right one for you?

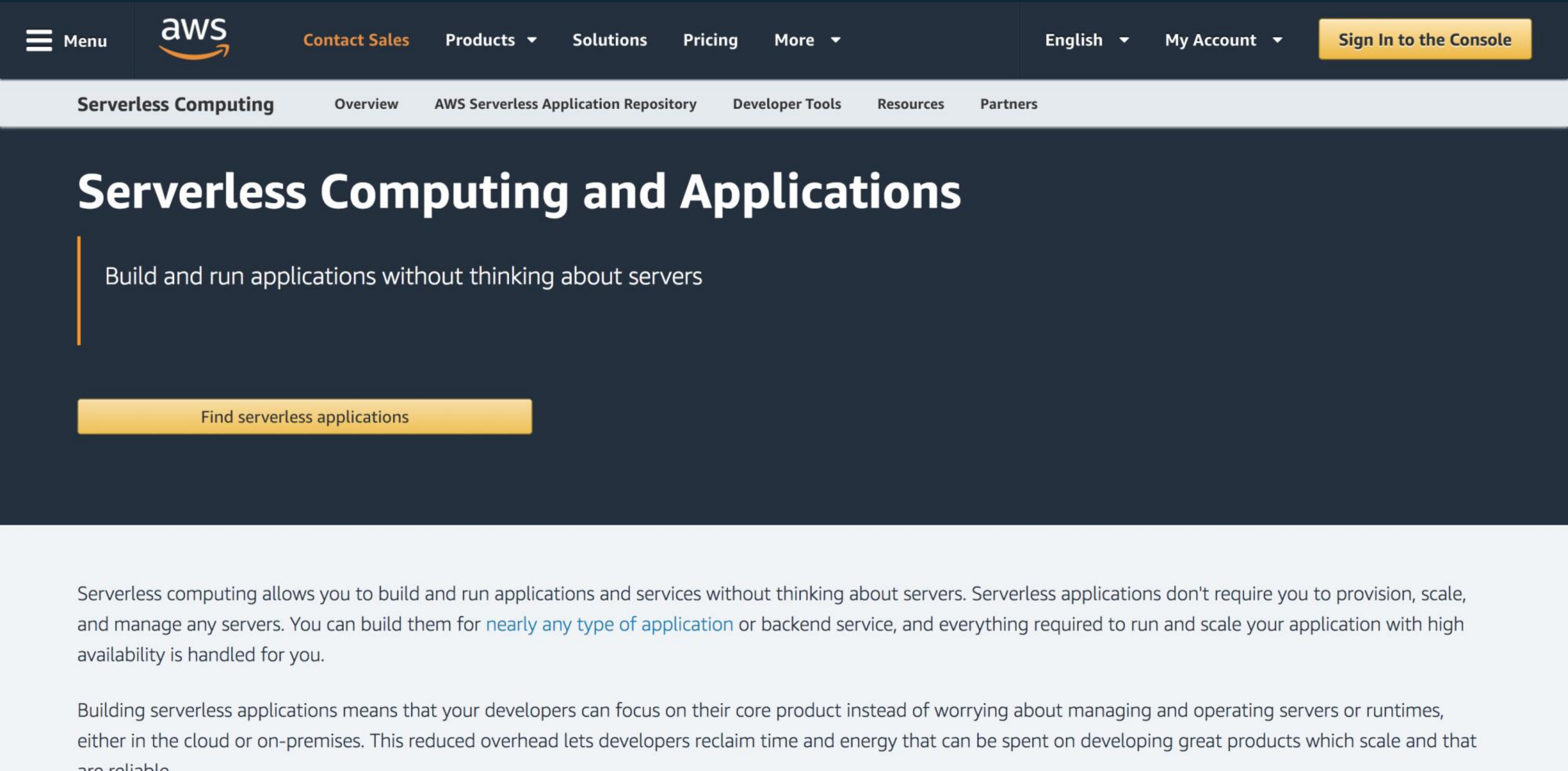
- All of these services require little care and feeding in terms of management
- All are HIPAA eligible and PCI compliant
- All support fine grained permissions via AWS IAM
- All have a pay as you go model without commitments

FIN, ACK

There are many ways to get data between microservices!

- Kinesis, SNS, SQS, and the Lambda API are just a few of the ways
- You *might* need an API that you create yourself
- Think through the factor comparisons on scale, durability, persistence, consumption models, retries, and pricing.
- You will probably end up needing more than one and potentially end up using each of these in some part of your infrastructure
- Evaluate and test using SAM CLI
- Serverless pricing models make testing new ideas low cost and easy to get started with!

aws.amazon.com/serverless



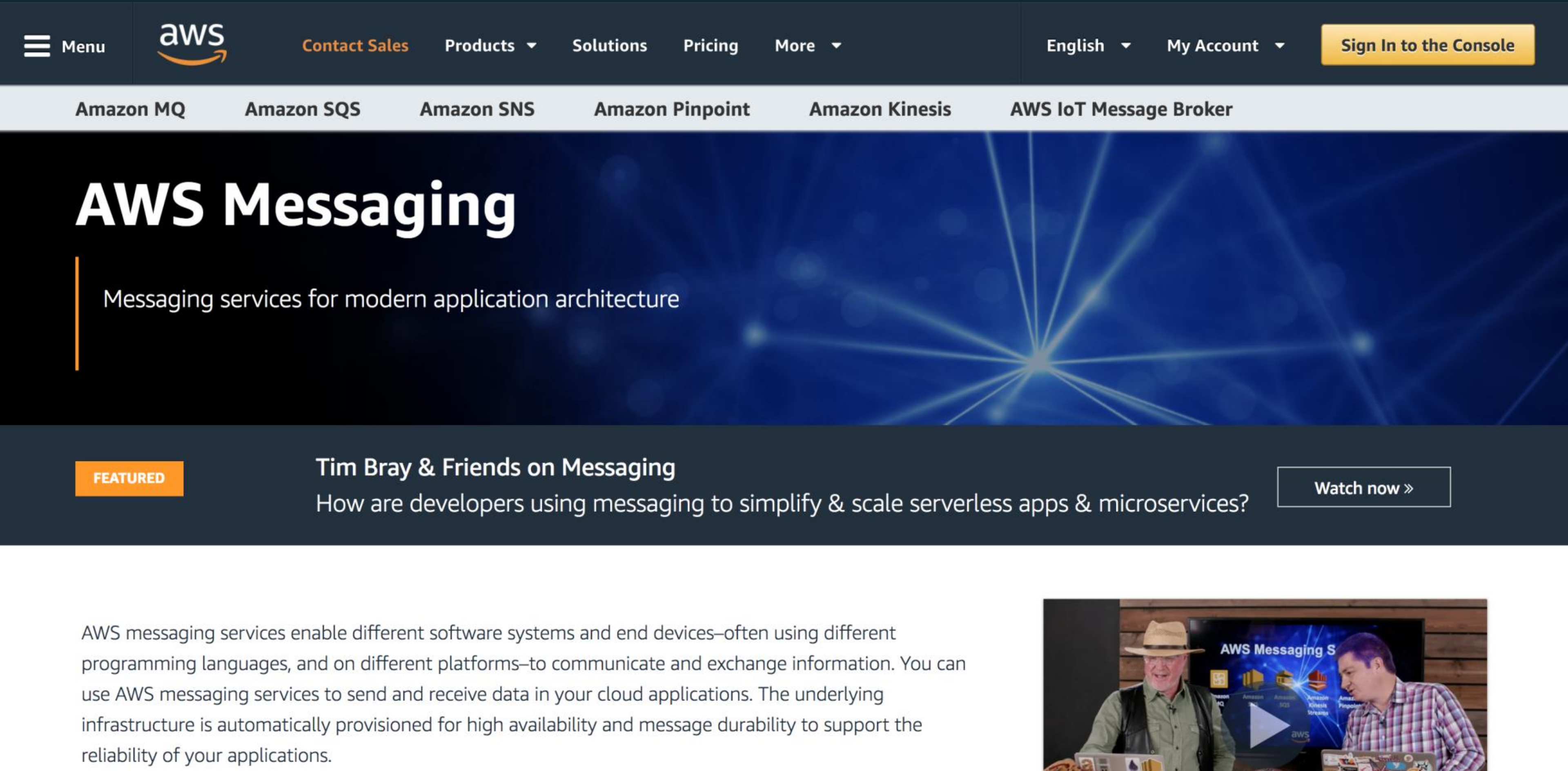
Serverless Computing and Applications

Build and run applications without thinking about servers

Serverless computing allows you to build and run applications and services without thinking about servers. Serverless applications don't require you to provision, scale, and manage any servers. You can build them for nearly any type of application or backend service, and everything required to run and scale your application with high availability is handled for you.

Building serverless applications means that your developers can focus on their core product instead of worrying about managing and operating servers or runtimes, either in the cloud or on-premises. This reduced overhead lets developers reclaim time and energy that can be spent on developing great products which scale and that

aws.amazon.com/messaging



AWS Messaging

Messaging services for modern application architecture

FEATURED

Tim Bray & Friends on Messaging

How are developers using messaging to simplify & scale serverless apps & microservices?

Watch now »

AWS messaging services enable different software systems and end devices–often using different programming languages, and on different platforms–to communicate and exchange information. You can use AWS messaging services to send and receive data in your cloud applications. The underlying infrastructure is automatically provisioned for high availability and message durability to support the reliability of your applications.

Chris Munns
munns@amazon.com
@chrismunns

https://www.flickr.com/photos/theredproject/3302110152/

https://secure.flickr.com/photos/dullhunk/202872717/