AWS Lambda Overview

Run code without provisioning or managing servers

aws

Introduction to AWS Lambda

- Function-as-a-Service
- Run code without provisioning or managing servers
- Pay only for the compute time you consume
- Automatically runs your code with high availability
- Scale with usage

aws

Lambda handles

- Load balancing
- Auto scaling
- Handling failures
- Security isolation
- OS management
- Managing utilization

(and many other things) for you

aws

What is Serverless?

No infrastructure provisioning, no management

Automatic scaling

Pay for value

Highly available and secure

aws

AWS operational responsibility models

Compute	Virtual Machine	EC2	Elastic Beanstalk	ECS/EKS	Fargate	AWS Lambda
Databases	MySQL	MySQL on EC2	RDS MySQL	RDS Aurora	Aurora Serverless	DynamoDB
Storage	Storage					S3
Messaging	ESBs		Amazon MQ	Kinesis		SQS / SNS
Analytics	Hadoop	Hadoop on EC2	EMR	Elasticsearch Service		Athena

Comparison of operational responsibility

	AWS manages	Customer manages
AWS Lambda Serverless functions	• Data source integrations • Physical hardware, software, networking, and facilities • Provisioning	• Application code
AWS Fargate Serverless containers	• Container orchestration, provisioning • Cluster scaling • Physical hardware, host OS/kernel, networking, and facilities	• Application code • Data source integrations • Security config and updates, network config, management tasks
ECS/EKS Container-management as a service	• Container orchestration control plane • Physical hardware software, networking, and facilities	• Application code • Data source integrations • Work clusters • Security config and updates, network config, firewall, management tasks
EC2 Infrastructure-as-a-Service	• Physical hardware software, networking, and facilities	• Application code • Data source integrations • Scaling • Security config and updates, network config, management tasks • Provisioning, managing scaling and patching of servers

aws

Anatomy of a Lambda Function

aws

Serverless applications

AWS
Lambda

aws

Serverless applications

Function

Node.js
Python
Java
C#
Go
Ruby
Runtime API

aws

Serverless Applications

Event source

Function

Changes in data state

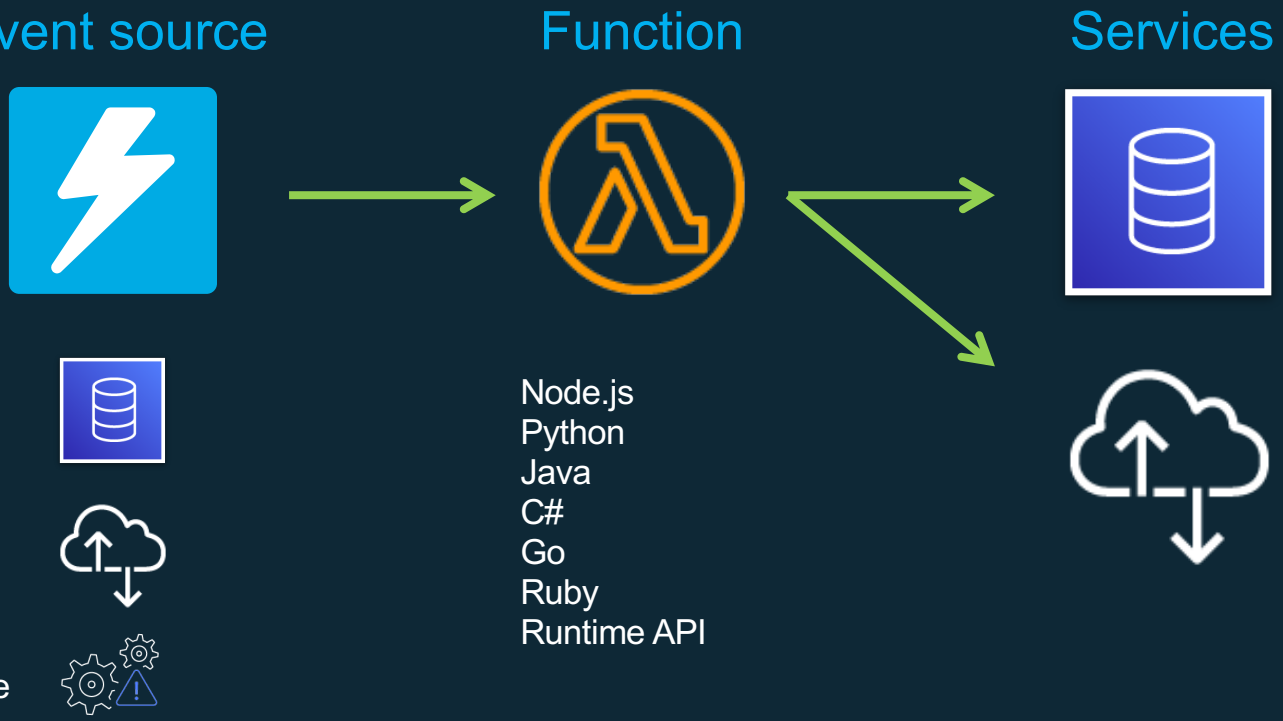
Requests to endpoints

Changes in Resource state

Node.js
Python
Java
C#
Go
Ruby
Runtime API

aws

Serverless Applications



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)

```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello World!')
    }
```


aws

Monitoring and debugging Lambda functions

- AWS Lambda console includes a dashboard for functions
  - Lists all Lambda functions
  - Easy editing of resources, event sources and other settings
  - At-a-glance metrics
- Metrics automatically reported to Amazon CloudWatch for each Lambda function
  - Requests
  - Errors
  - Latency
  - Throttles

Lambda Layers

Lets functions easily share code: Upload layer once, reference within any function

Promote separation of responsibilities, lets developers iterate faster on writing business logic

Built in support for secure sharing by ecosystem

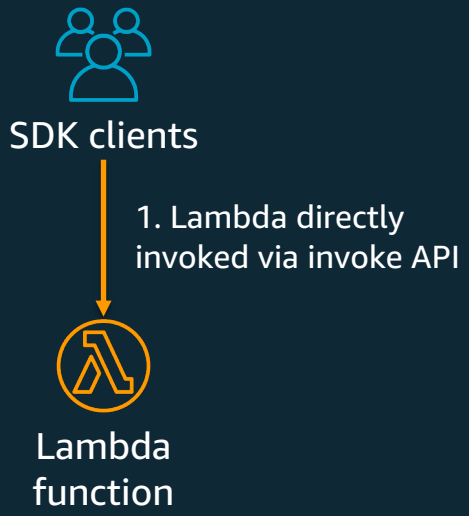
aws

Lambda Layers: Uses cases

- Custom code, that is used by more than one function
- Libraries, modules, frameworks to simplify the implementation of your business logic
    - Security/monitoring service
- Shared code that does not change frequently
- Bring your own Runtime
    - C++
    - Rust
    - PHP

aws

Lambda API



API provided by the Lambda service

Used by all other services that invoke Lambda across all models

Supports sync and async calls

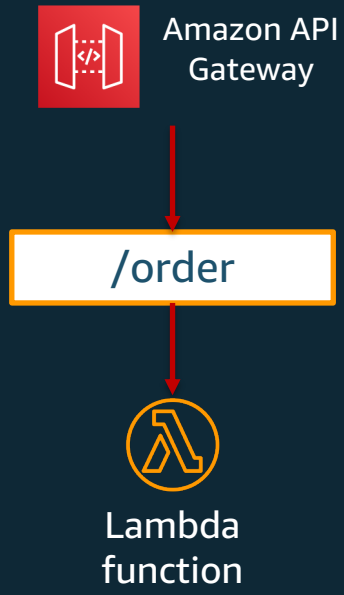
Can pass any event payload structure you want

Client included in every SDK

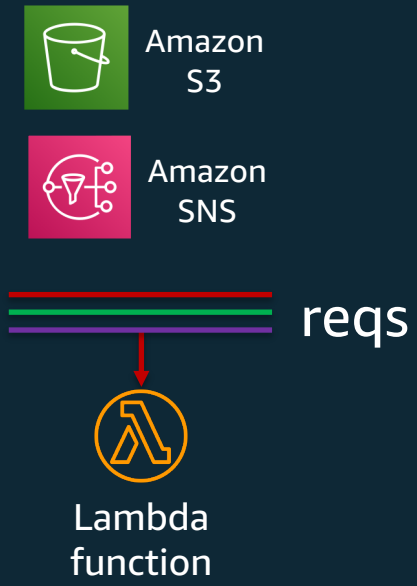
aws

Lambda Execution Models

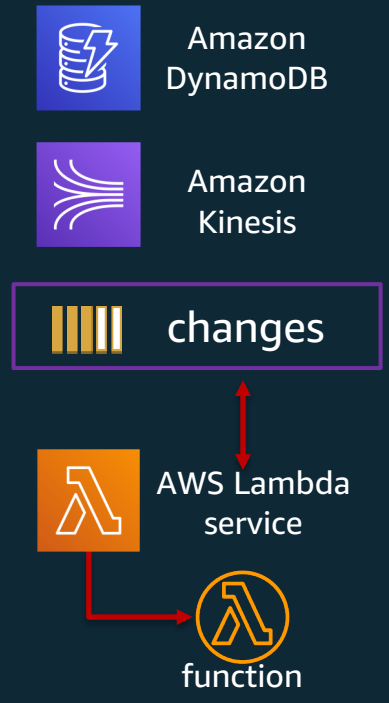
Synchronous (push)



Asynchronous (event)



Stream (Poll-based)



aws

Dead-Letter Queue

- Asynchronous Lambda invocations are retried two more times (3 times total)
- Lambda can forward payloads that were not processed to a dead-letter queue (IF configured!)
- A mechanism to handle exceptions and failures gracefully

aws

Lambda Permissions Model

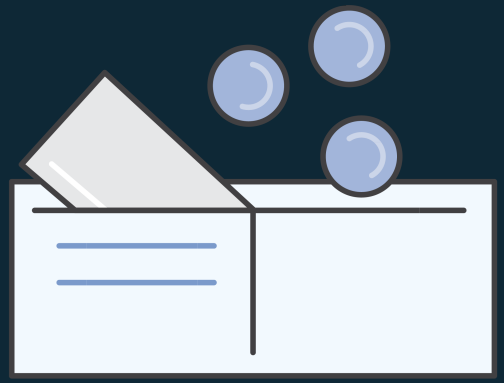
Function policies:

- “Actions on bucket X can invoke Lambda function Z"
- Resource policies allow for cross account access
- Used for sync and async invocations

Execution role:

- “Lambda function A can read from DynamoDB table users”
- Define what AWS resources/API calls can this function access via IAM
- Used in streaming invocations

Fine-grained Pricing



Free Tier
1M requests and 400,000 GBs of compute.
Every month, every customer.

Buy compute time in 100ms increments

Low request charge

No hourly, daily, or monthly minimums

No per-device fees

Never pay for idle

aws

Example event sources that trigger AWS Lambda

DATA STORES


Amazon S3


Amazon DynamoDB


Amazon Kinesis


Amazon Cognito

ENDPOINTS


Amazon API Gateway


AWS IoT


AWS Step Functions


Amazon Alexa

CONFIGURATION REPOSITORIES


AWS CloudFormation


AWS CloudTrail


AWS CodeCommit


Amazon CloudWatch

EVENT/MESSAGE SERVICES


Amazon SES


Amazon SNS


Cron events

… and more on the way!

aws

Common AWS Lambda use cases

Web Apps

- 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

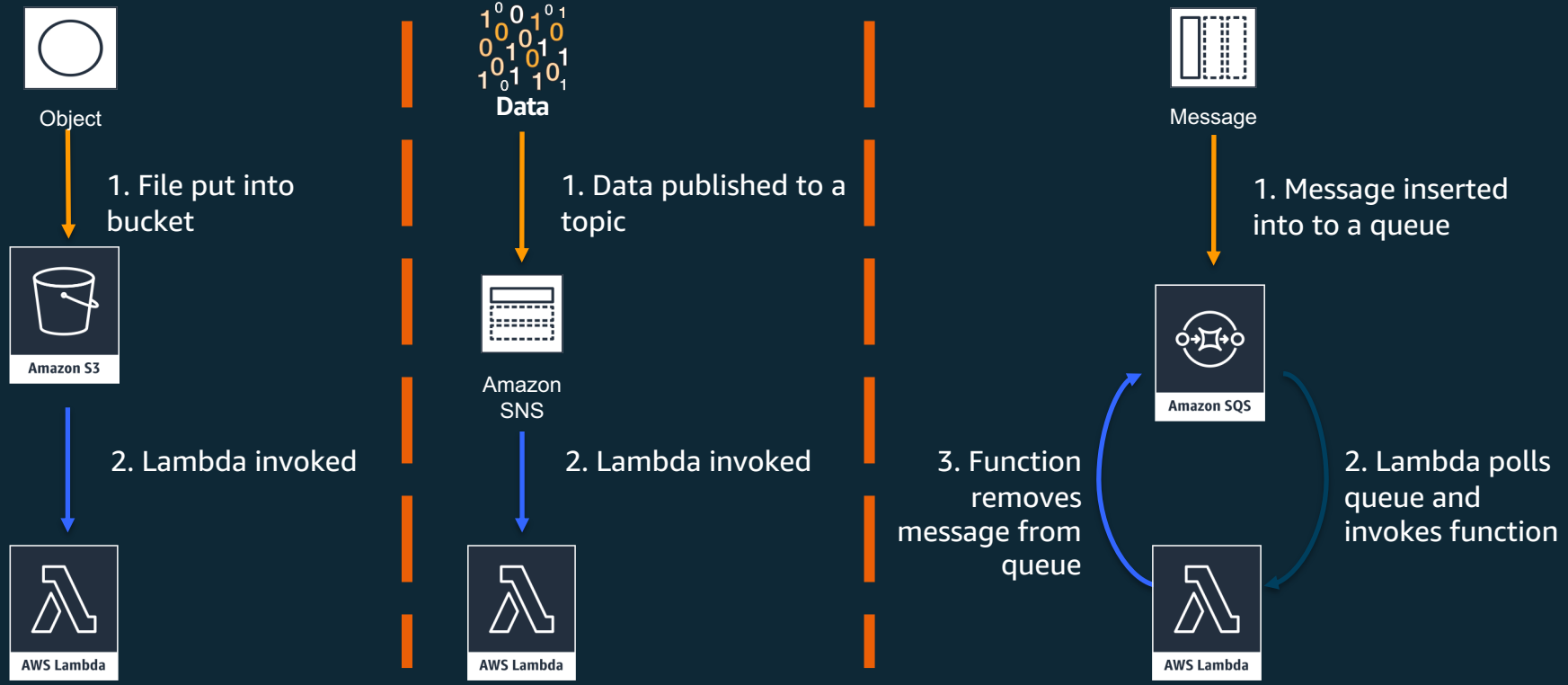
- Powering voice-enabled apps
- Alexa Skills Kit

IT Automation

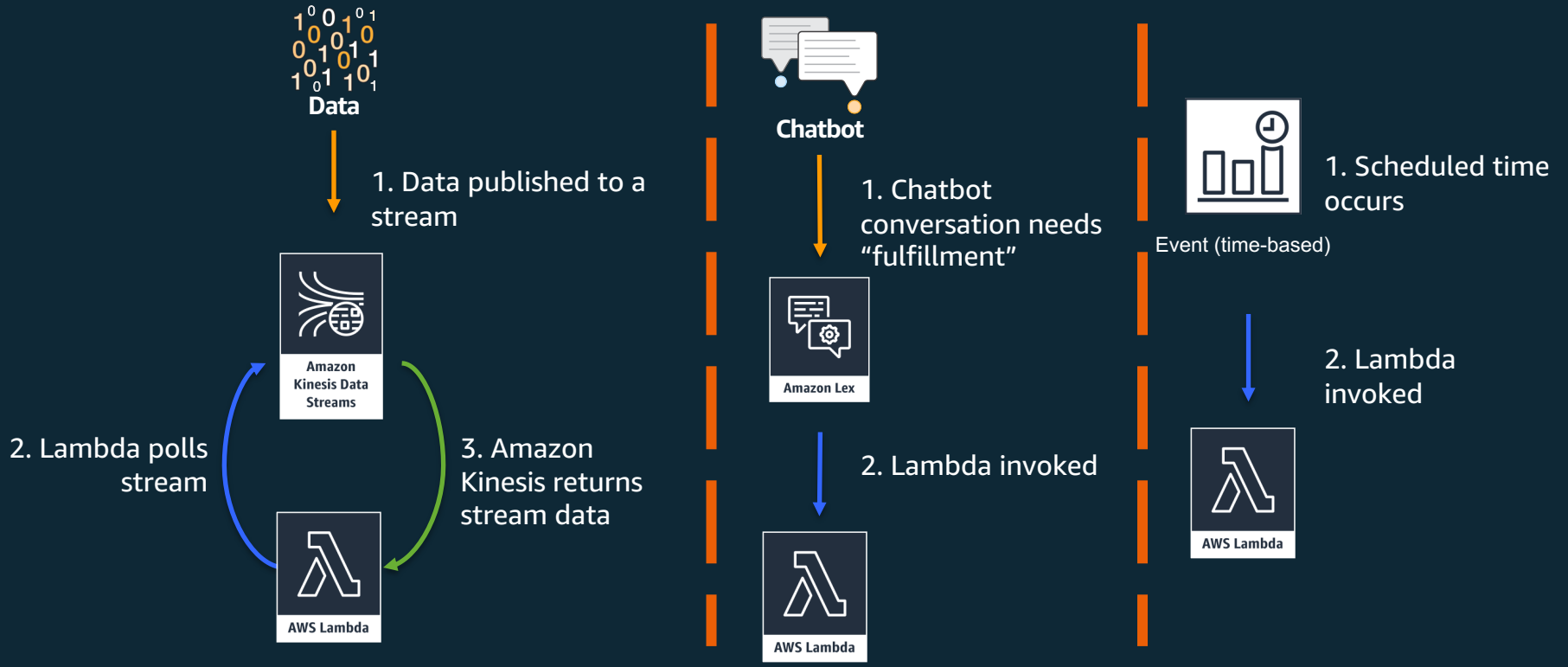
- Policy engines
- Extending AWS services
- Infrastructure management

aws

Serverless architectures



Serverless Architectures



When do we choose Lambda over other compute offerings?

AWS Compute Offerings

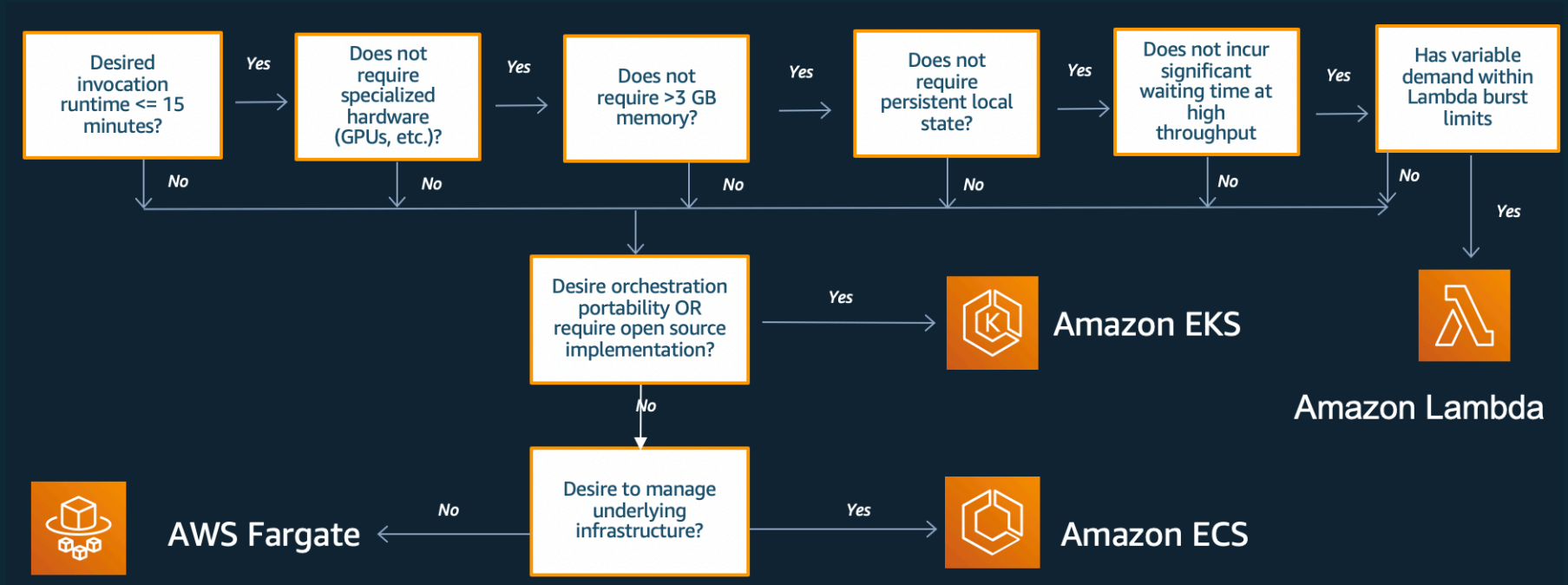
Service	**Amazon EC2**	**AWS Fargate**	**AWS Lambda**
Unit of scale	**VM**	**Task**	**Function**
Level of abstraction	**H/W**	**OS**	**Runtime**

aws

AWS Compute Offerings

			
Service	**Amazon EC2**	**AWS Fargate**	**AWS Lambda**
How do I choose?	I want to configure servers, storage, networking, and my OS	I want to run my containers	Run my code when it's needed

Picking a Compute Platform: Containers vs. Lambda



Step Functions

Orchestration for serverless apps

“I want to sequence functions”

“I want to select functions based on data”

“I want to run functions in parallel”

“I want to retry functions”

“I want to try/catch/finally”

“I want to run code for hours”

AWS Step Functions

aws

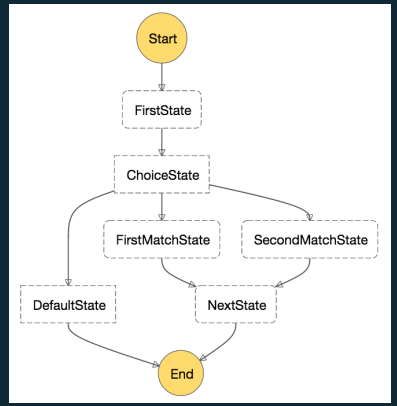
AWS Step Functions

Easily coordinate multiple Lambda functions using visual workflows

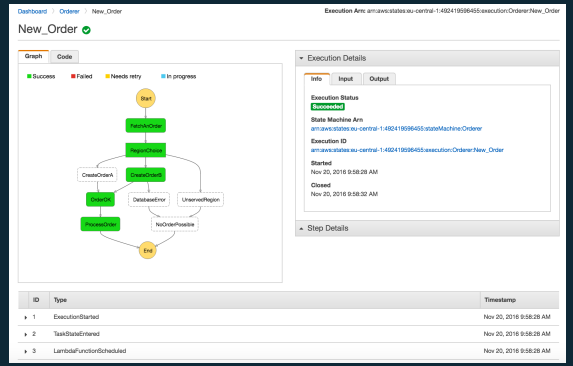
Define in JSON

```
Code
{
  "Comment": "An AWL example using a choice state.",
  "StartAt": "FirstState",
  "States": {
    "FirstState": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
      "Next": "ChoiceState"
    },
    "ChoiceState": {
      "Type" : "Choice",
      "Choices": [
        {
```

Visualize in the Console



Monitor Executions



aws

Benefits of Step Functions orchestration

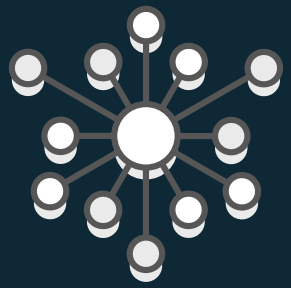
Productivity

Coordinate and visualize Lambda functions as a series of steps to quickly create serverless apps

Resilience

Automatically trigger and track each step at scale and handle errors with built-in retry and fallback

Agility



Change and add steps without writing code to evolve applications and innovate faster

aws

AWS Service integrations with Step Functions


AWS Lambda


AWS Batch


Amazon DynamoDB


AWS Fargate


Amazon Elastic Container Service


Amazon Simple Notification Service


AWS Step Functions


Amazon Simple Queue Service


Amazon SageMaker


AWS Glue

aws

Serverless Application Model

AWS Serverless Application Model (AWS SAM)

AWS CloudFormation brings:

- Infrastructure as code
- Easy to provision and manage a collection of related AWS resources
- Input .yaml file and output provisioned AWS resources
- Optimized for infrastructure

AWS SAM:

- CloudFormation extension optimized for serverless
- New serverless resources: functions, APIs, and tables
- Supports anything CloudFormation supports
- Open specification (Apache 2.0)

AWS Lambda best practices

- Limit your function/code size
- Node – remember execution is asynchronous
- 500 MB /tmp directory provided to each function
- Don’t assume function will reuse underlying infrastructure
  - But take advantage of it when it does occur
- You own the logs
  - Include details from service-provided context
- Create custom metrics
  - Operations-centric vs. business-centric

aws

Additional best practices

- Use environment variables
  - Parameterize code and change parameters independent of code updates
  - Use for securing credentials and keeping them out of code
- Externalize authorization to IAM roles whenever possible
  - Least privilege and separate IAM roles
- Externalize configuration
  - DynamoDB is great for this
- Take advantage of dead letter queues
  - Use to handle failed invocations

aws

Additional best practices

- Make sure your downstream setup “keeps up” with Lambda scaling
  - Limit concurrency when talking to relational databases
- Be aware of service throttling
  - Engage AWS Support to increase your limits
- Contact AWS Support before known large scaling event
  - Infrastructure Event Management (IEM) offers real-time support for large scaling events
  - IEM is available for Enterprise and Business support customers

aws

Thank you!