AWS re:Invent

SVS401-R

Optimizing Your Serverless Applications

Chris Munns

Senior Manager/Principal Developer Advocate - Serverless

Amazon Web Services

AWS re:Invent

aws

About me

Chris Munns - munns@amazon.com, @chrismunns

- Sr Manager/Principal Developer Advocate – Serverless
- New Yorker (ehhh…ish.. kids/burbs/ya know?)

Previously:

- AWS Business Development Manager – DevOps, July ‘15 - Feb ‘17
- AWS Solutions Architect Nov ‘11- Dec ‘14
- 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

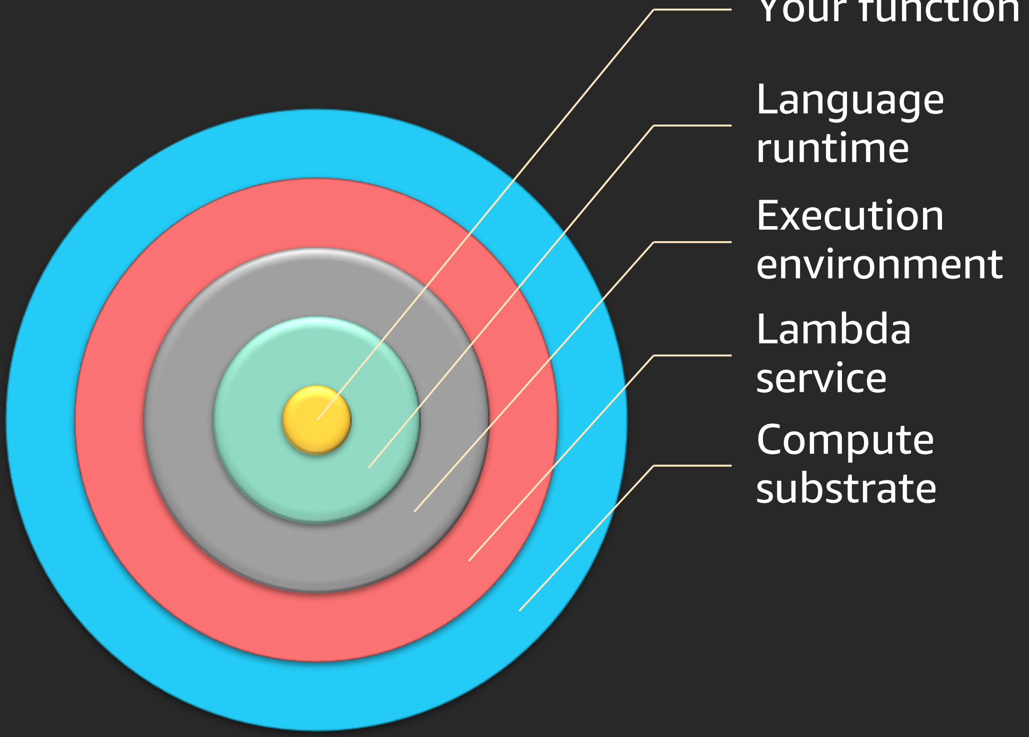
Why are we here today?

Today's focus:

Serverless applications

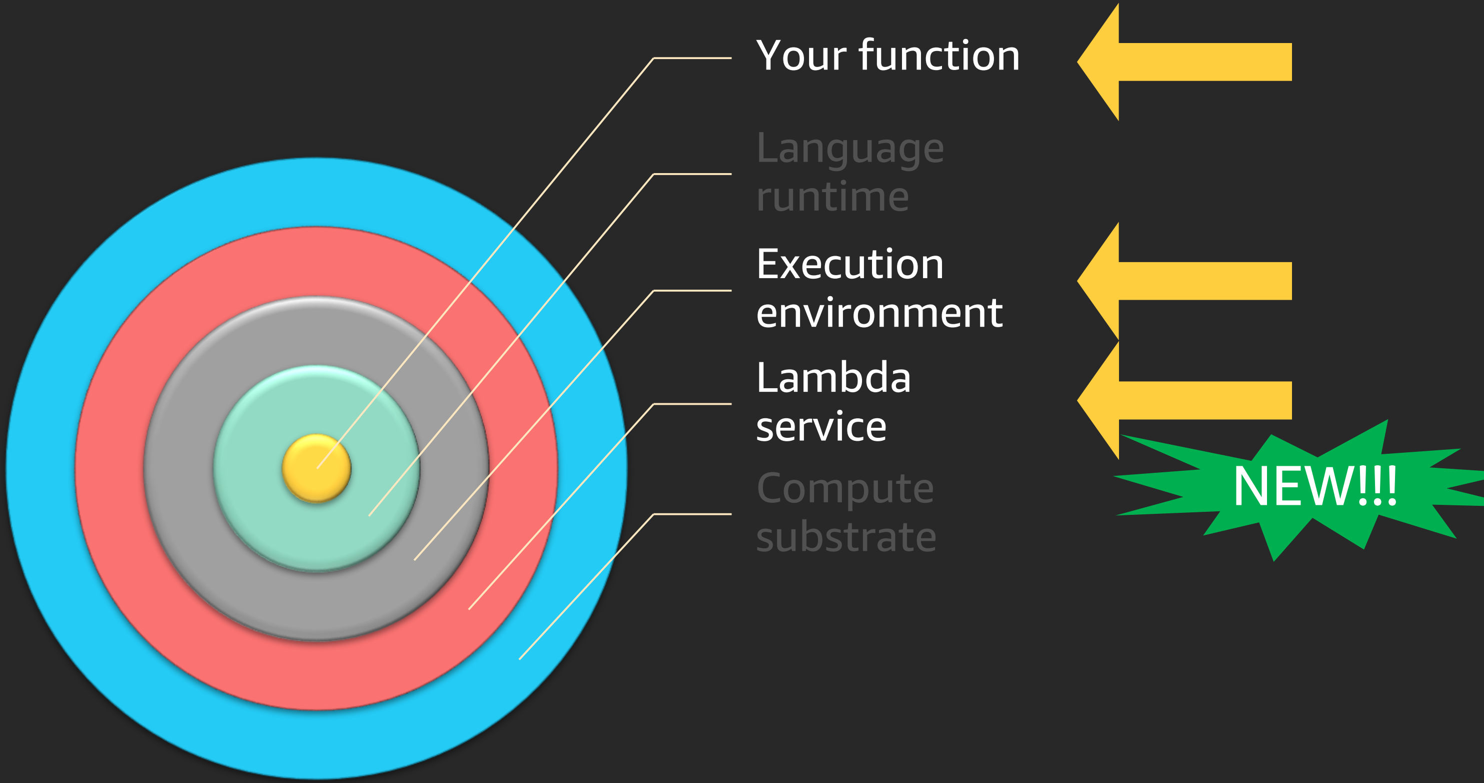
AWS Lambda

Anatomy of a Lambda function



Anatomy of a Lambda function

Places where you can impact performance



Your function

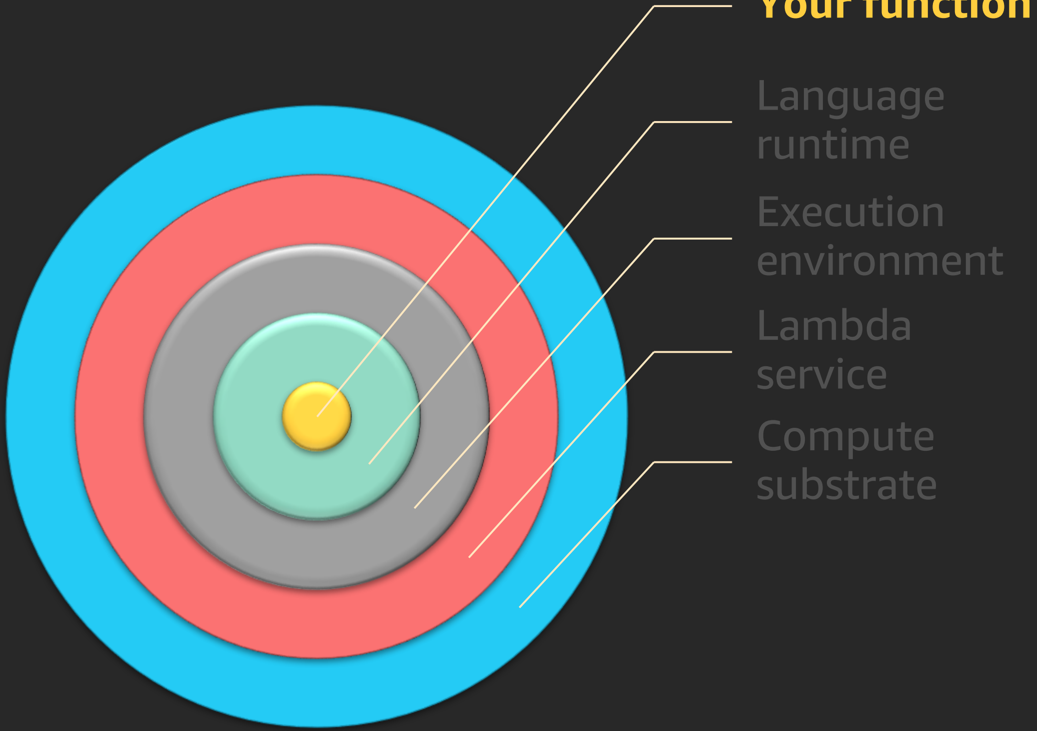
Language runtime

Execution environment

Lambda service

Compute substrate

Anatomy of a Lambda function



Serverless applications

AWS Lambda

Serverless applications

Function

Node.js
Python
Java
C#
Go
Ruby
Runtime API

Anatomy of a Lambda function

Handler() function

Function to be executed upon invocation

Event object

Data sent during Lambda function Invocation

Context object

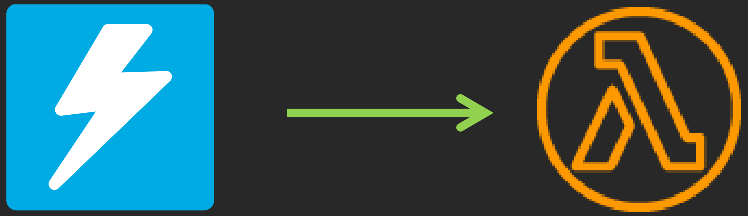
Methods available to interact with runtime information (request ID, log group, more)

```
import json

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

Serverless applications

Event source

Function



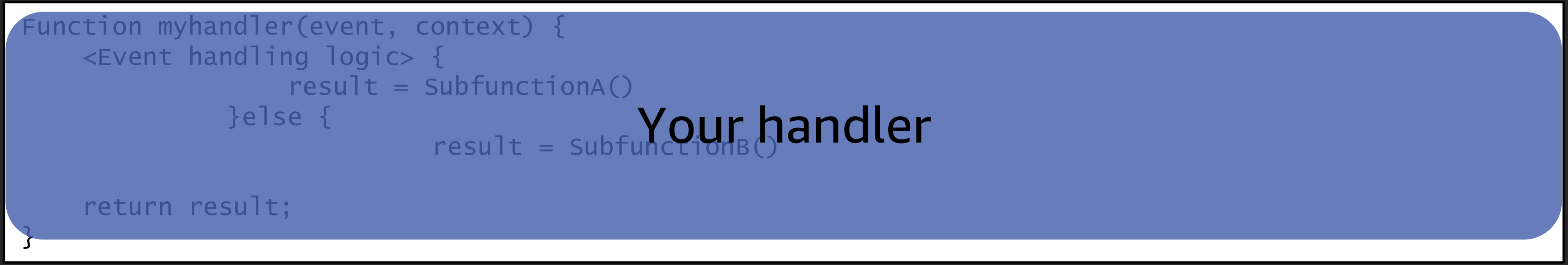
Changes in
data state

Requests to
endpoints

Changes in
Resource state

Node.js
Python
Java
C#
Go
Ruby
Runtime API

Anatomy of a Lambda function



```
Function myhandler(event, context) {
    <Event handling logic> {
                  result = SubfunctionA()
              }else {
                           result = SubfunctionB()

    return result;
}
```

Your handler

```
Import sdk
Import http-lib
Import ham-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()

Function myhandler(event, context) {
    <Event handling logic> {
                  result = SubfunctionA()
              }else {
                           result = SubfunctionB()

    return result;
}
```

Your handler

```
Import sdk
Import http-lib
Import ham-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()
```

Dependencies, configuration information, common helper functions

```
Function myhandler(event, context) {
    <Event handling logic> {
                  result = SubfunctionA()
              }else {
                           result = SubfunctionB()

    return result;
}
```

Your handler

Pre-handler code, dependencies, variables

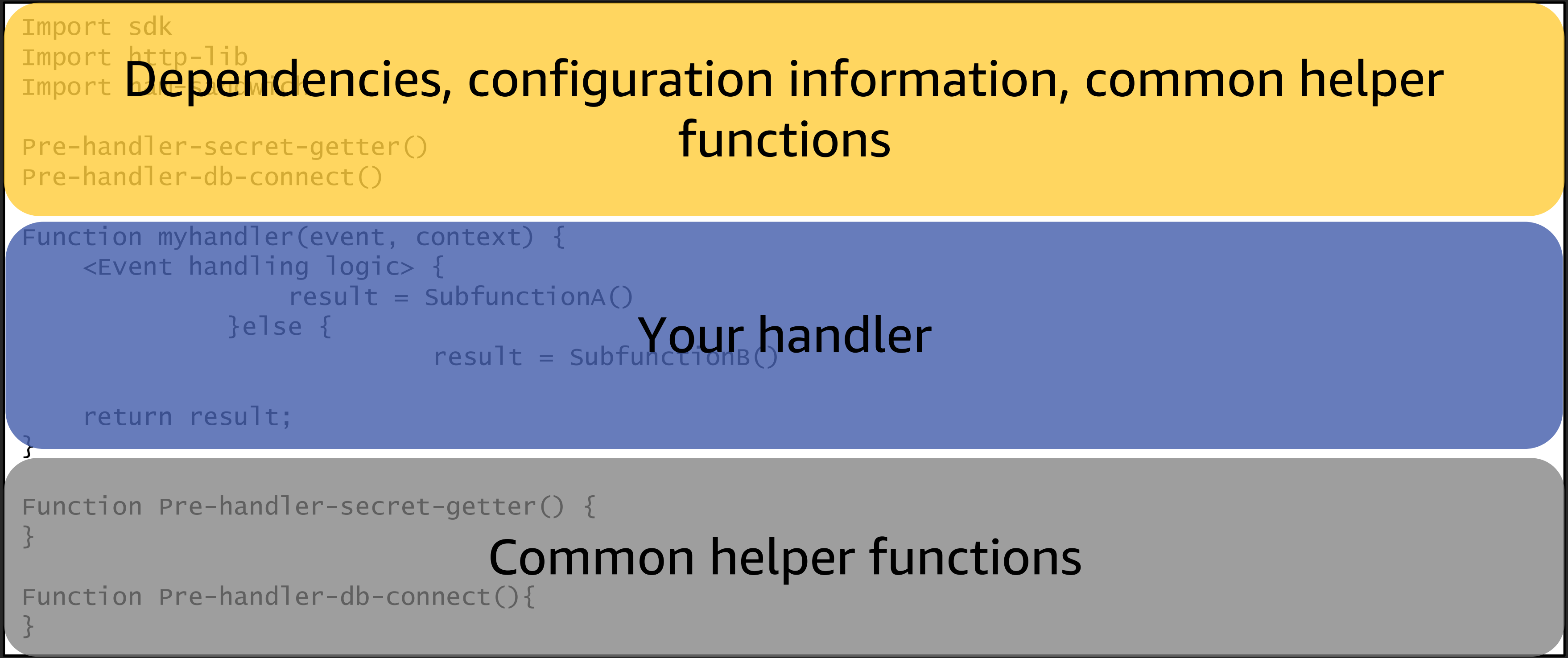
- Import only what you need
  - Where possible trim down SDKs and other libraries to the specific bits required
- Pre-handler code is great for establishing connections, but be prepared to then handle reconnections in further executions
- REMEMBER – execution environments are reused
  - Lazily load variables in the global scope
  - Don’t load it if you don’t need it – cold starts are affected
  - Clear out used variables so you don’t run into left-over state

```
Import sdk
Import http-lib
Import ham-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()

Function myhandler(event,
context) {
....
```

Dependencies, configuration information, common helper functions

Your handler

```
Import sdk
Import http-lib
Import ham-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()

Function myhandler(event, context) {
    <Event handling logic> {
                  result = SubfunctionA()
              }else {
                           result = SubfunctionB()

    return result;
}

Function Pre-handler-secret-getter() {
}

Function Pre-handler-db-connect(){
}
```



```
Import sdk
Import http-lib
Import ham-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()

Function myhandler(event, context) {
    <Event handling logic> {
                  result = SubfunctionA()
              }else {
                           result = SubfunctionB()

    return result;
}

Function Pre-handler-secret-getter() {
}

Function Pre-handler-db-connect(){
}
```

Dependencies, configuration information, common helper functions

Your handler

Common helper functions

AWS Lambda Environment Variables

- Key-value pairs that you can dynamically pass to your function
- Available via standard environment variable APIs such as process.env for Node.js or os.environ for Python
- Can optionally be encrypted via AWS Key Management Service (AWS KMS)
    - Allows you to specify in IAM what roles have access to the keys to decrypt the information
- Useful for creating environments per stage (i.e., dev, testing, production)

AWS Systems Manager – Parameter Store

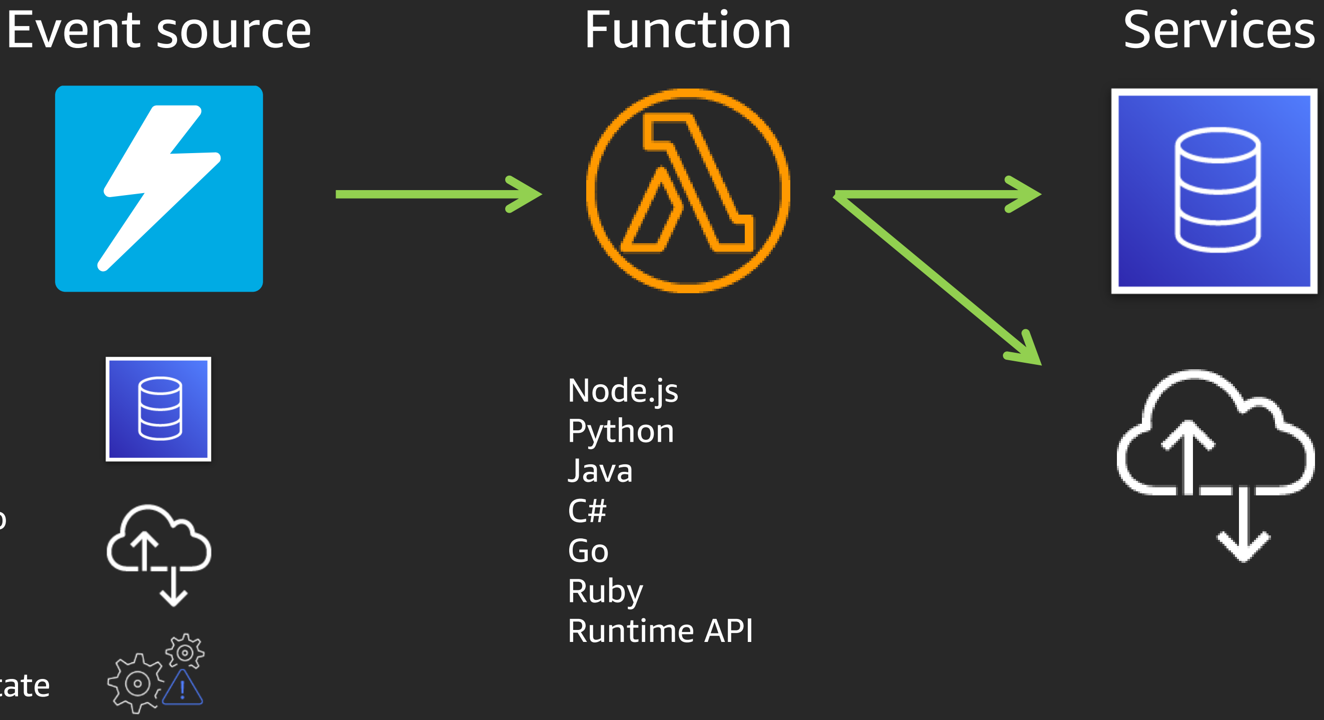
Centralized store to manage your configuration data

- Supports hierarchies
- Plaintext or encrypted with AWS KMS
- Can send notifications of changes to Amazon SNS/AWS Lambda
- Can be secured with IAM
- Calls recorded in AWS CloudTrail
- Can be tagged
- Works with AWS Secrets Manager
- Available via API/SDK

Useful for centralized environment variables, secrets control, feature flags

```
from __future__ import print_function
import json
import boto3
ssm = boto3.client('ssm', 'us-east-1')

def get_parameters():
    response = ssm.get_parameters(
        Names=['LambdaSecureString'],WithD
ecryption=True
    )
    for parameter in
response['Parameters']:
        return parameter['Value']

def lambda_handler(event, context):
    value = get_parameters()
    print("value1 = " + value)
    return value  # Echo back the first
key value
```

Serverless applications

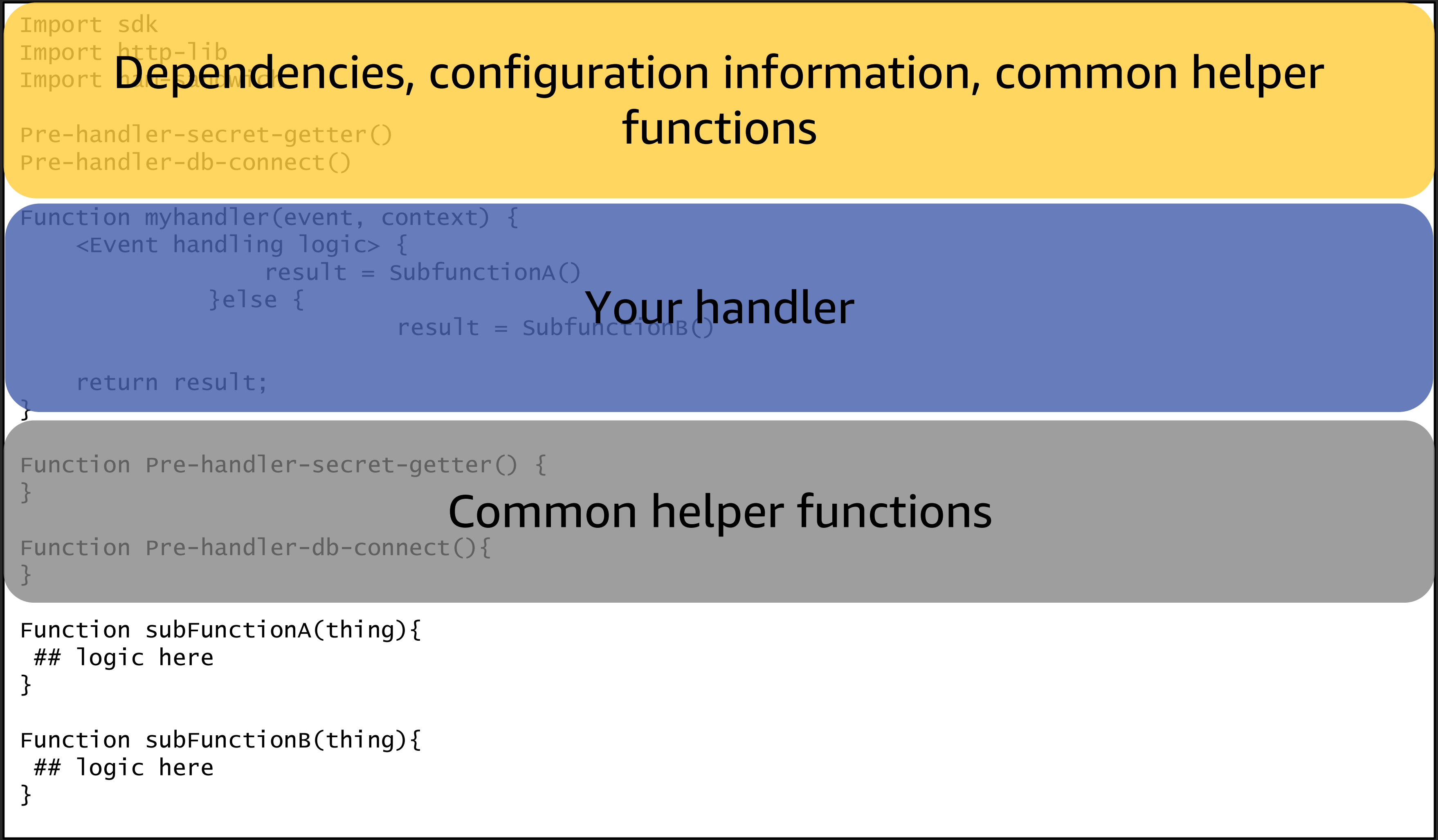


Changes in data state

Requests to endpoints

Changes in Resource state

Node.js
Python
Java
C#
Go
Ruby
Runtime API

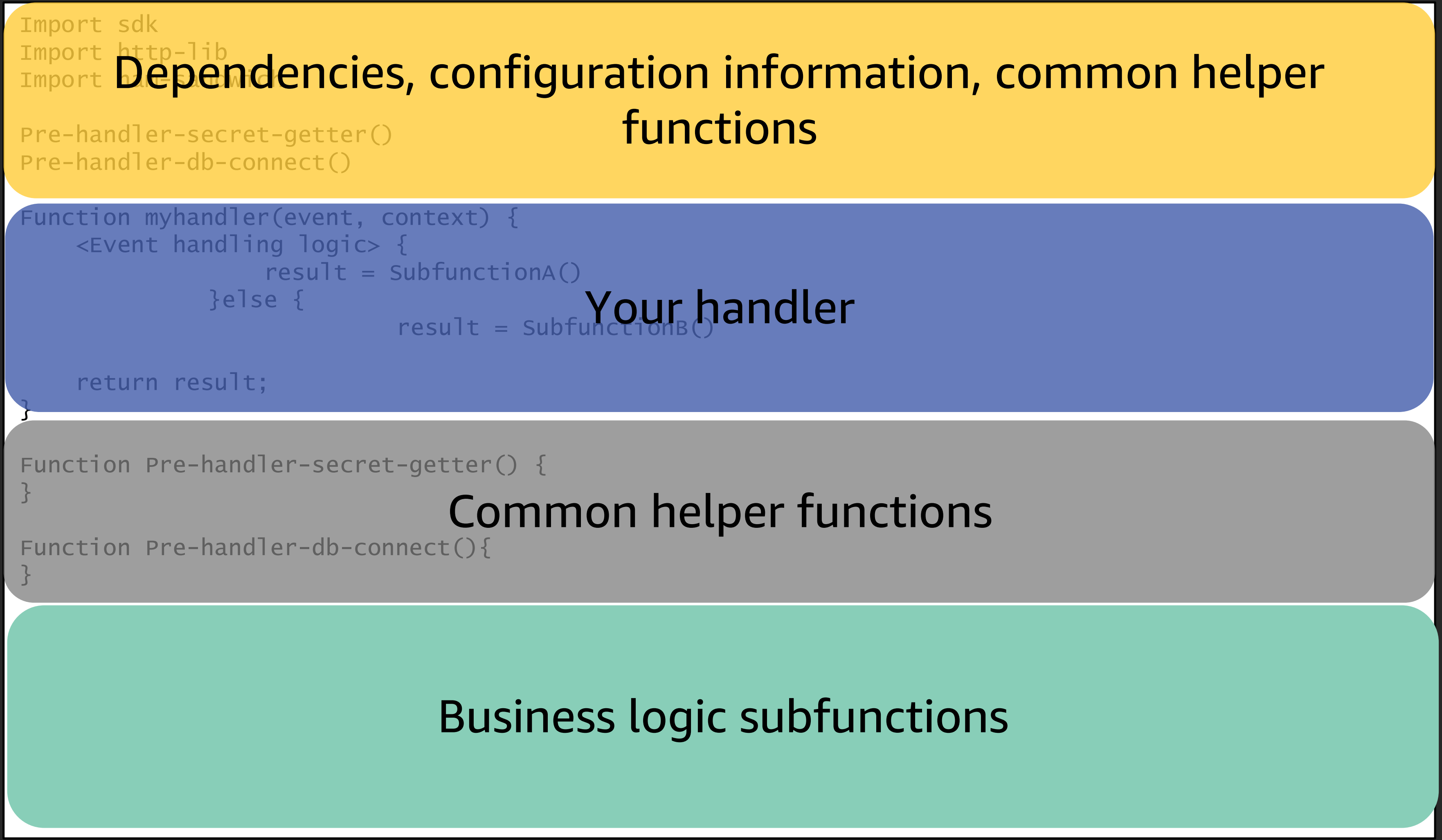


```
Import sdk
Import http-lib
Import ham-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()

Function myhandler(event, context) {
    <Event handling logic> {
                 result = SubfunctionA()
             }else {
                          result = SubfunctionB()

    return result;
}

Function Pre-handler-secret-getter() {
}

Function Pre-handler-db-connect(){
}

Function subFunctionA(thing){
 ## logic here
}

Function subFunctionB(thing){
 ## logic here
}
```

Dependencies, configuration information, common helper functions

Your handler

Common helper functions



```
Import sdk
Import http-lib
Import ham-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()
```

Dependencies, configuration information, common helper functions

```
Function myhandler(event, context) {
    <Event handling logic> {
                 result = SubfunctionA()
             }else {
                          result = SubfunctionB()

    return result;
}
```

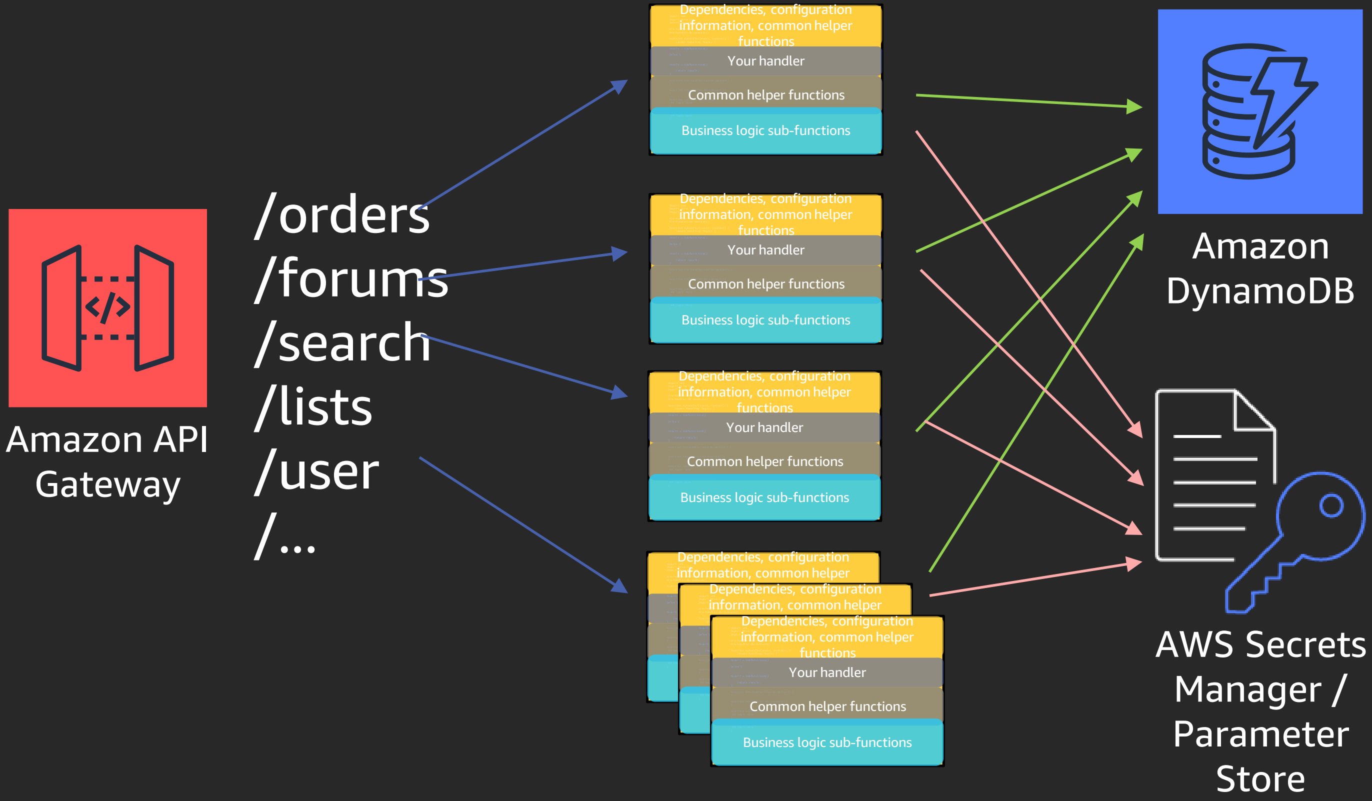
Your handler

```
Function Pre-handler-secret-getter() {
}

Function Pre-handler-db-connect(){
}
```

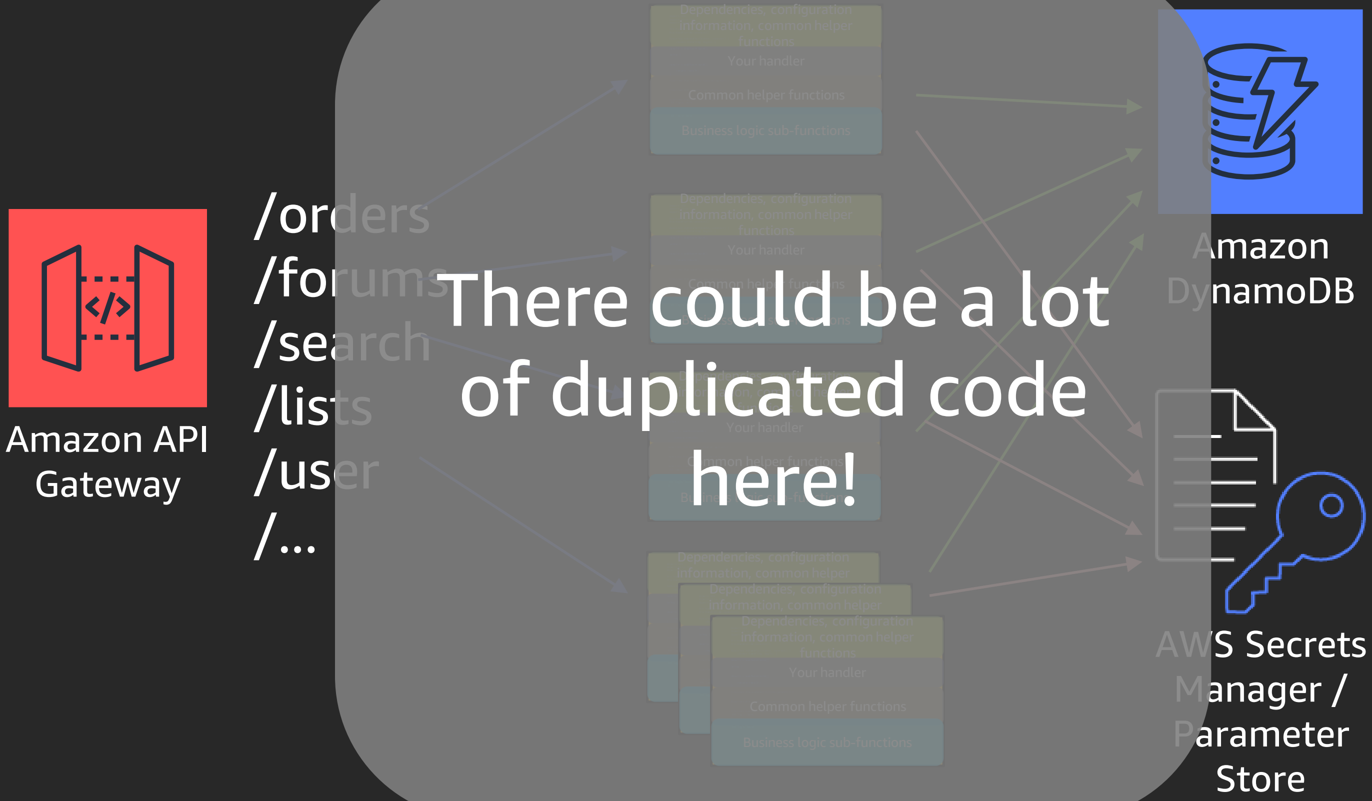
Common helper functions

Business logic subfunctions

Anatomy of a serverless application


Amazon API Gateway
/orders
/forums
/search
/lists
/user
/...
Dependencies, configuration information, common helper functions
Your handler
Common helper functions
Business logic sub-functions
Dependencies, configuration information, common helper functions
Your handler
Common helper functions
Business logic sub-functions
Dependencies, configuration information, common helper functions
Your handler
Common helper functions
Business logic sub-functions
Dependencies, configuration information, common helper
Dependencies, configuration information, common helper
Dependencies, configuration information, common helper functions
Your handler
Common helper functions
Business logic sub-functions
Amazon DynamoDB
AWS Secrets Manager / Parameter Store

Anatomy of a serverless application


Amazon API Gateway
/orders
/forums
/search
/lists
/user
/…
Dependencies, configuration information, common helper functions
Your handler
Common helper functions
Business logic sub-functions
Amazon DynamoDB
AWS Secrets Manager / Parameter Store

There could be a lot of duplicated code here!

Lambda Layers

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

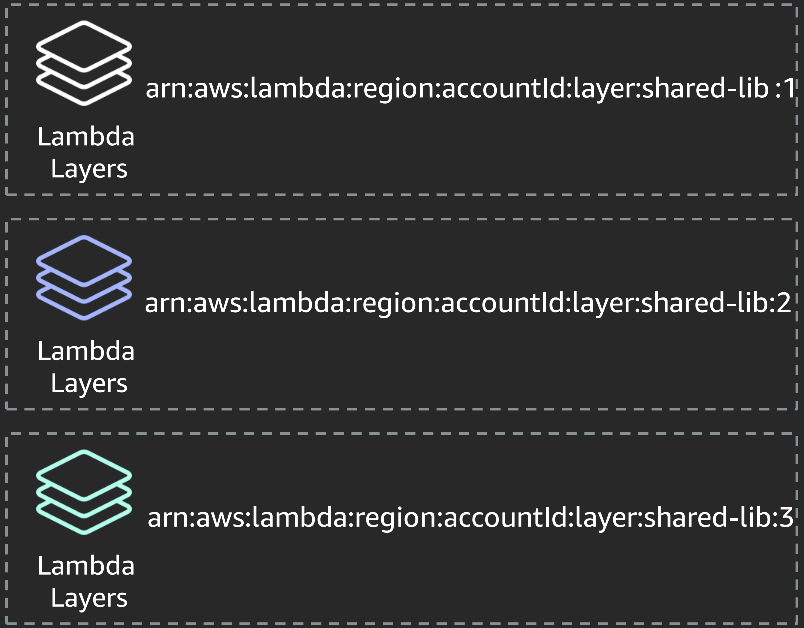
Layer can be anything: dependencies, training data, configuration files, etc

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

Built-in support for secure sharing by ecosystem

Using Lambda Layers

- Put common components in a ZIP file and upload it as a Lambda layer
- Layers are immutable and can be versioned to manage updates
- When a version is deleted or permissions to use it are revoked, functions that used it previously will continue to work, but you won’t be able to create new ones
- You can reference up to five layers, one of which can optionally be a custom runtime

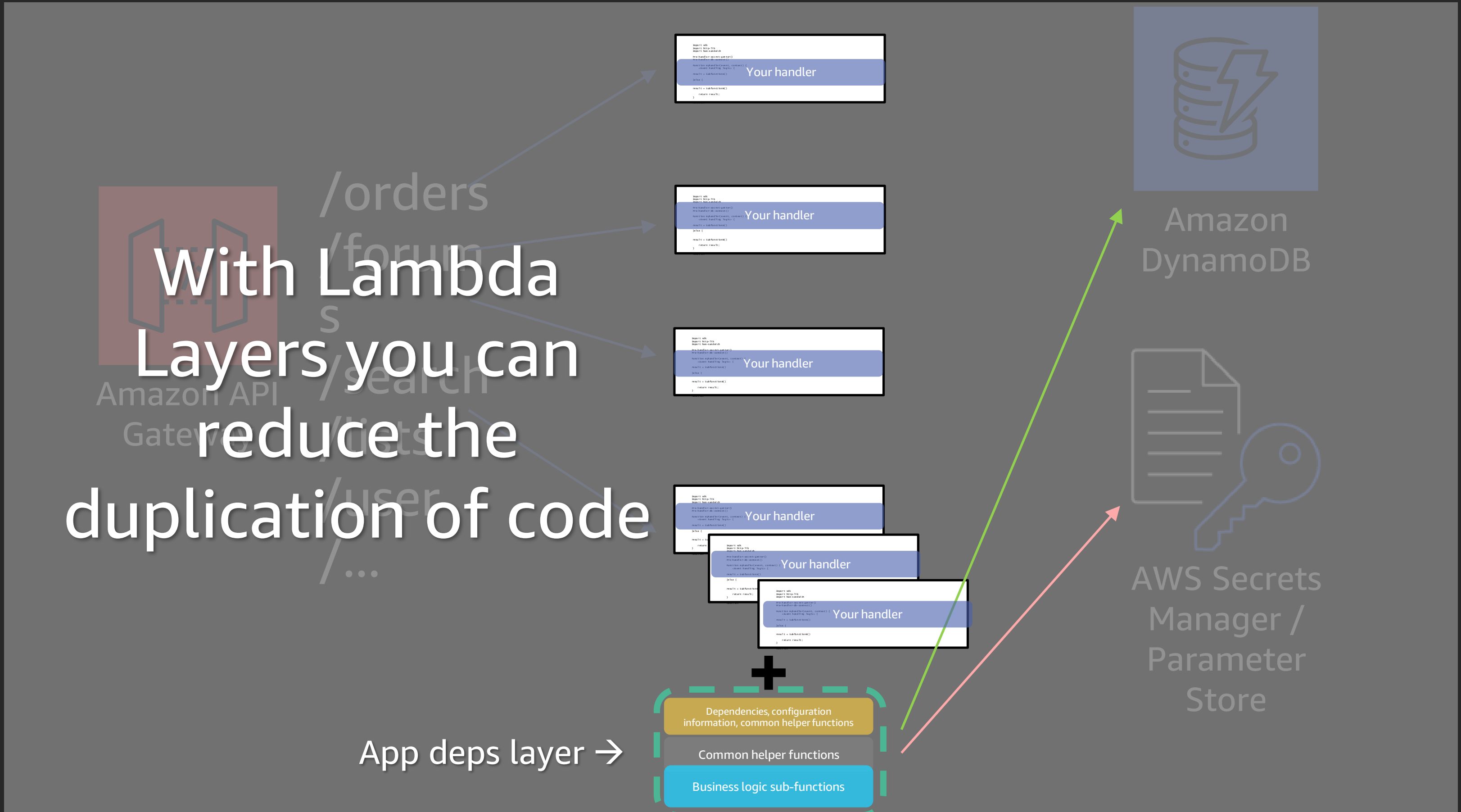


Lambda Layers
arn:aws:lambda:region:accountId:layer:shared-lib :1

Lambda Layers
arn:aws:lambda:region:accountId:layer:shared-lib:2

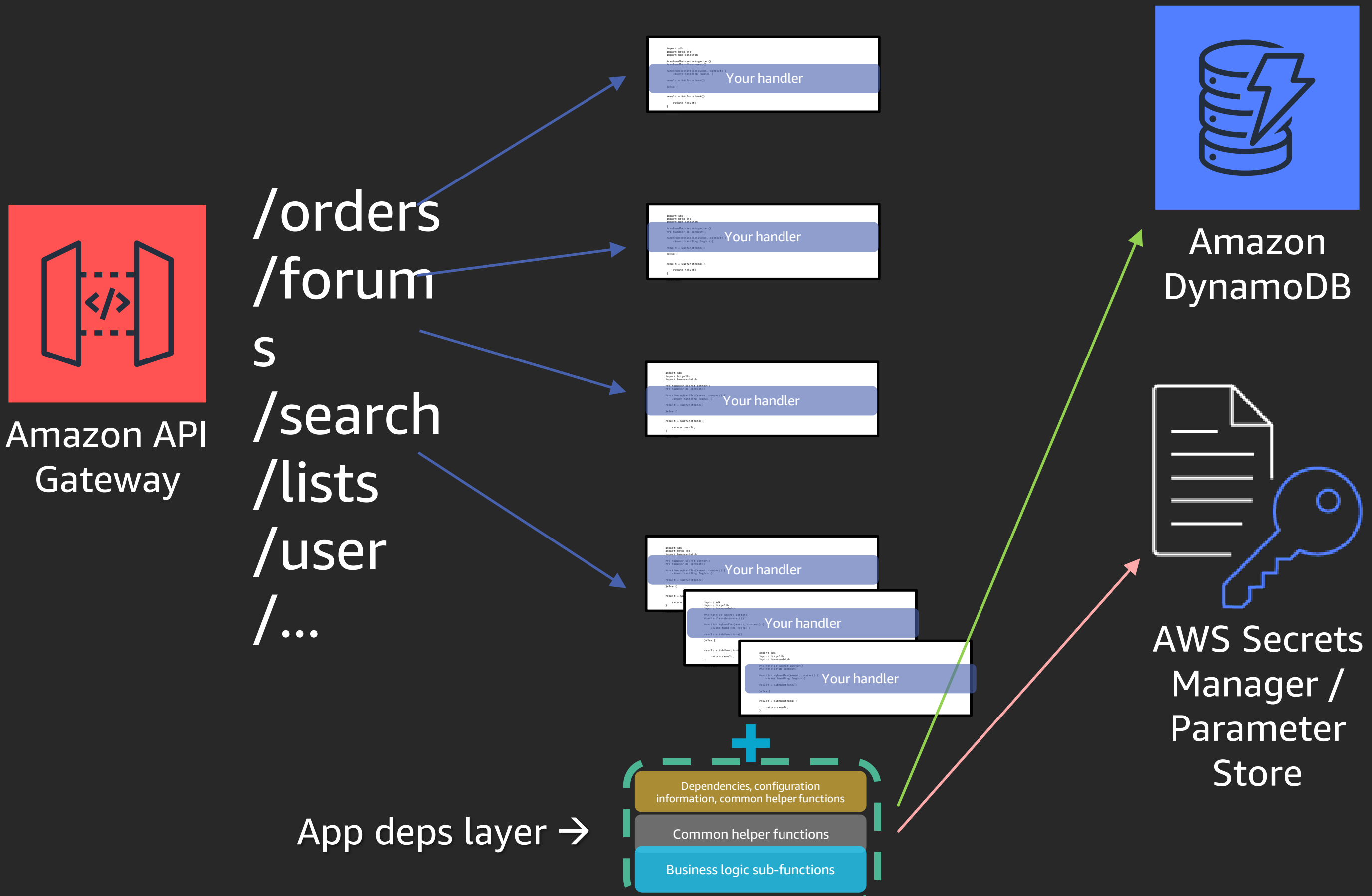
Lambda Layers
arn:aws:lambda:region:accountId:layer:shared-lib:3

Anatomy of a serverless application



With Lambda Layers you can reduce the duplication of code

Anatomy of a serverless application



Amazon API Gateway

/orders
/forum
s
/search
/lists
/user
/…

Your handler

Your handler

Your handler

Your handler

Your handler

Your handler

App deps layer →

Dependencies, configuration information, common helper functions

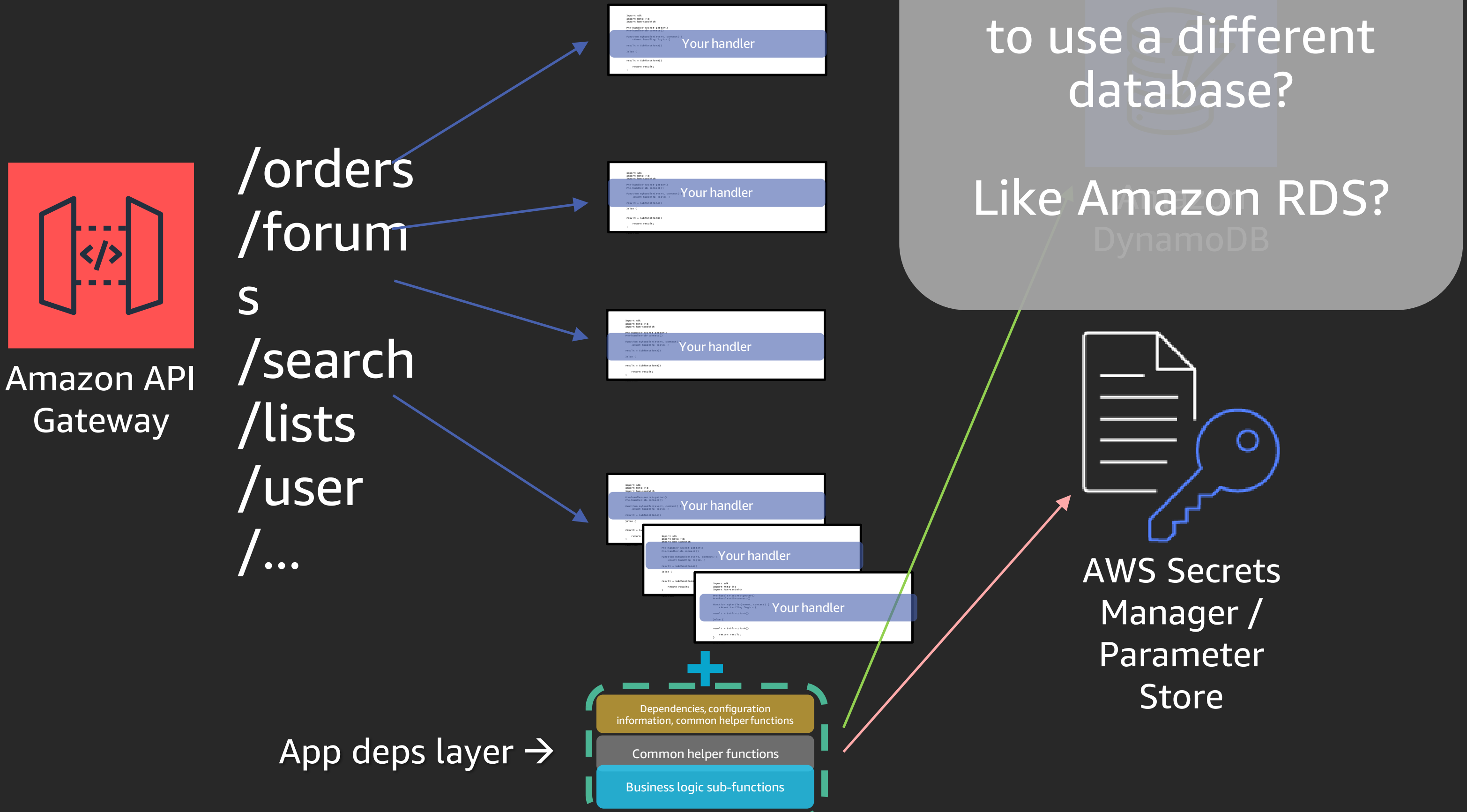
Common helper functions

Business logic sub-functions

Amazon DynamoDB

AWS Secrets Manager / Parameter Store

Anatomy of a serverless application



Amazon API Gateway

/orders
/forums
/search
/lists
/user
/...

Your handler

Your handler

Your handler

Your handler

Your handler

Your handler

App deps layer →

Dependencies, configuration information, common helper functions

Common helper functions

Business logic sub-functions

AWS Secrets Manager / Parameter Store

DynamoDB

What if we wanted to use a different database?

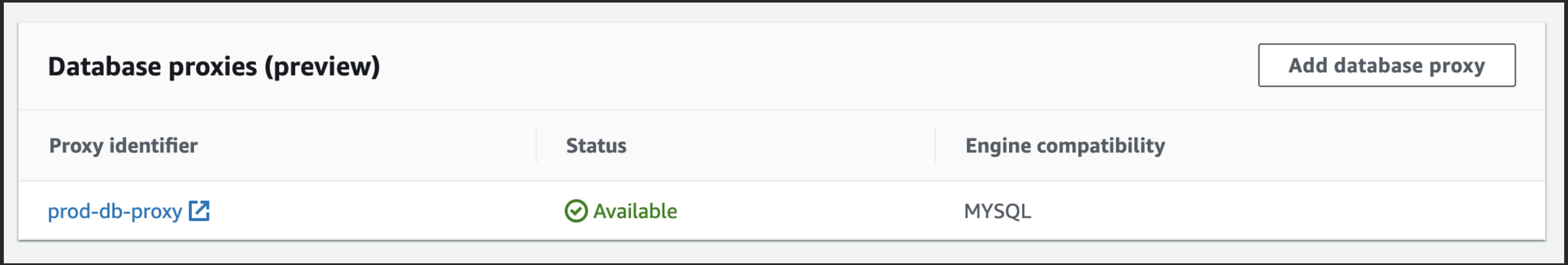
Like Amazon RDS?

Introducing Amazon RDS Proxy

NEW!!!

Simplifies connecting to Amazon RDS databases from Lambda

- Reduces connections via a shared connection pool to your database
- Integrates with Secrets Manager for simple authentication
- Handles failover of database instances transparently for you
- In preview today: limited regions, supports just MySQL



Database proxies (preview)

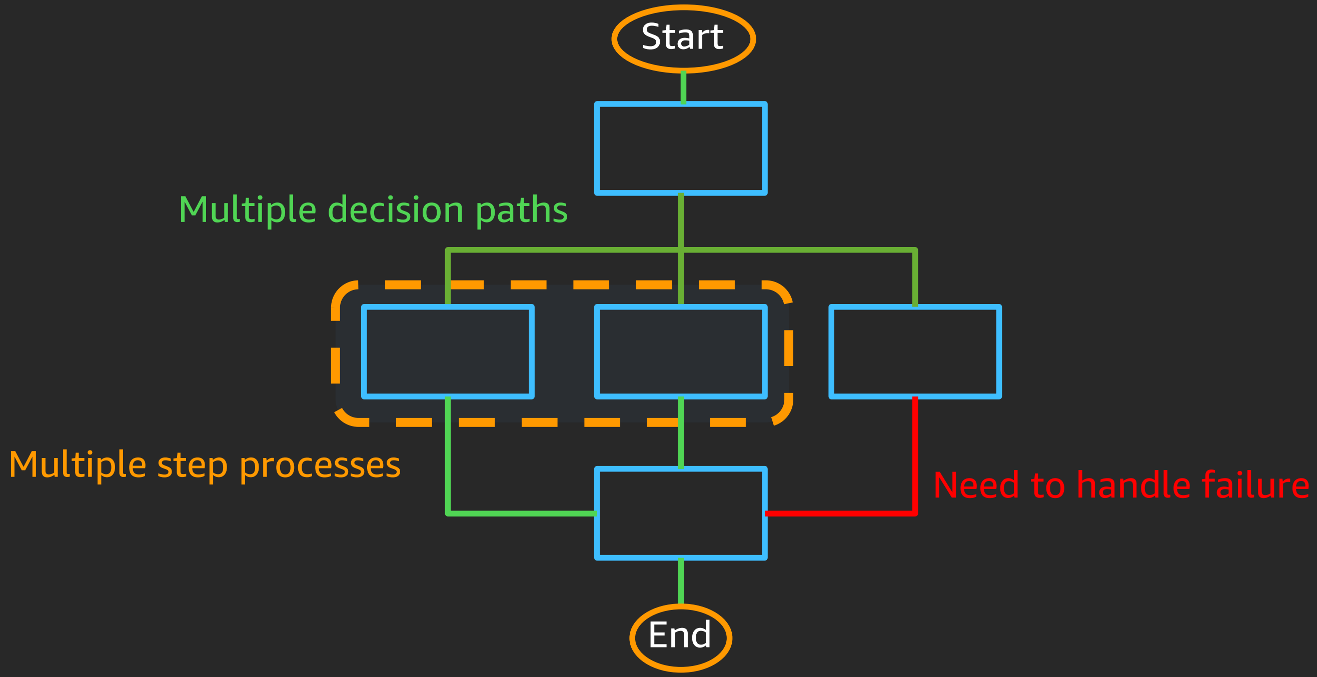
Proxy identifier	Status	Engine compatibility
prod-db-proxy	✓ Available	MYSQL

Less code > more code

Concise function logic

- Use functions to TRANSFORM, not TRANSPORT
  - Use purposeful built services for communication fan-out, message handling, data replication, writing to data stores/databases
- Leave retry and error handling to the services themselves
- Read only what you need. For example:
  - Message filters in Amazon SNS
  - Fine grained rules in Amazon EventBridge
  - Query filters in Amazon RDS Aurora
  - Use Amazon S3 Select
  - Properly indexed databases

Business workflow is rarely sequential start to finish



Multiple decision paths

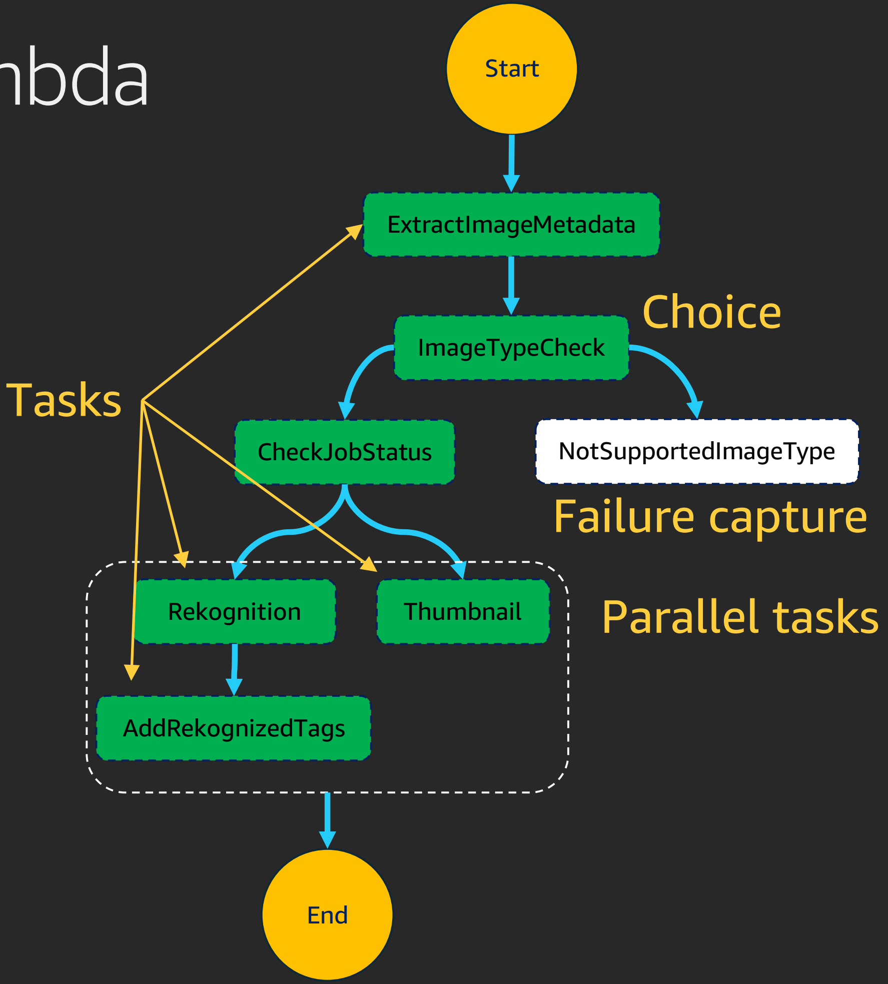
Multiple step processes

Need to handle failure

AWS Step Functions + Lambda

“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
- Logs the state of each step, so when things do go wrong, you can diagnose and debug problems quickly



Step Functions: Integrations

AWS Step Functions

Simplify building workloads such as order processing, report generation, and data analysis

Write and maintain less code; add services in minutes

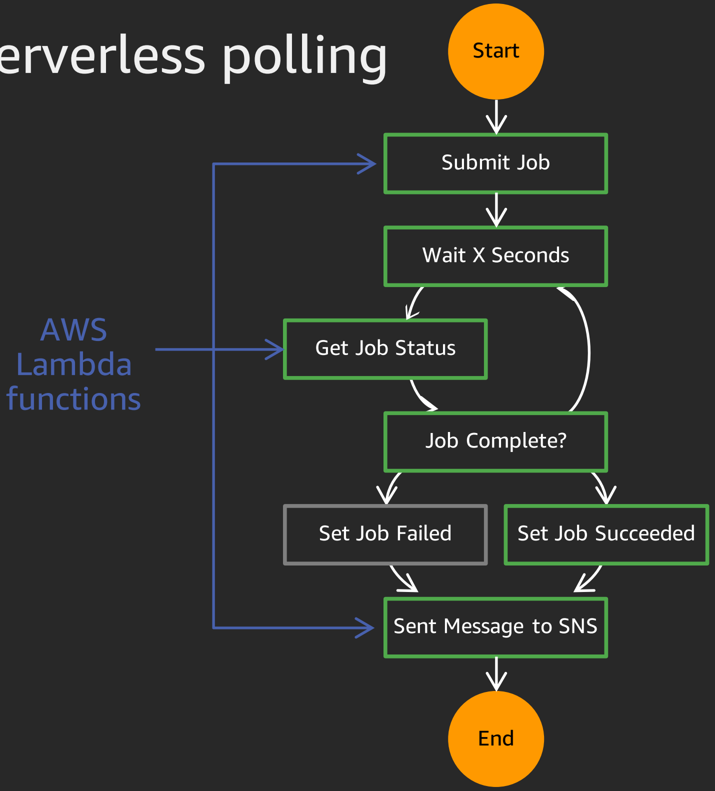
More service integrations:

- Amazon Simple Notification Service (Amazon SNS)
- Amazon Simple Queue Service (Amazon SQS)
- Amazon SageMaker
- AWS Glue
- AWS Batch
- Amazon Elastic Container Service
- AWS Fargate
- New! Amazon EMR

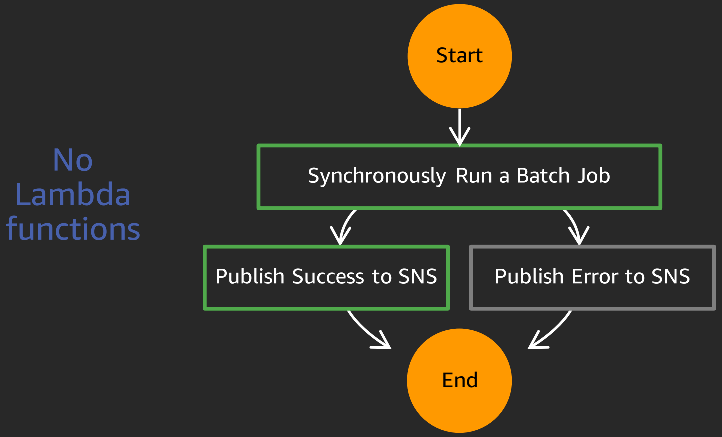
NEW!!!

Simpler integration, less code

With serverless polling



With direct service integration



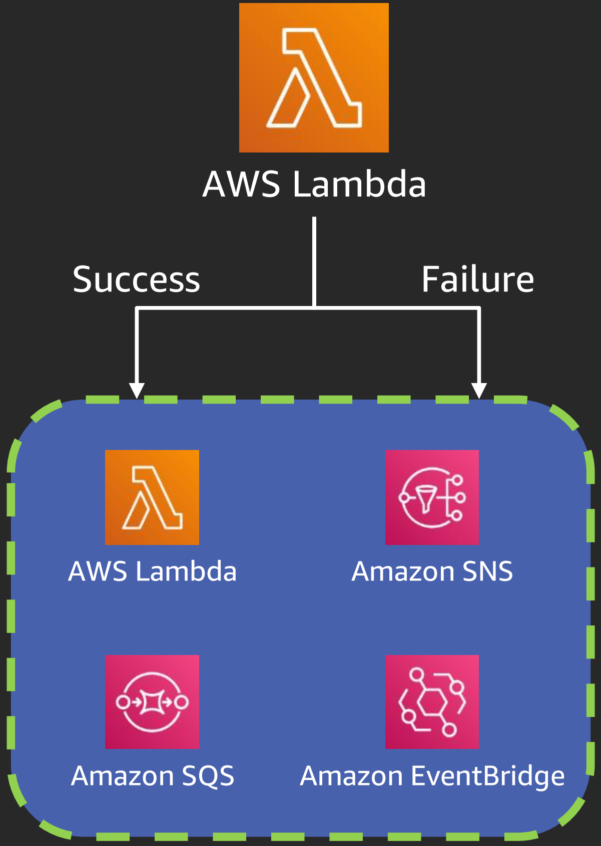
Introducing Lambda Event Destinations

NEW!!!

For asynchronous invocations, capture success or failure

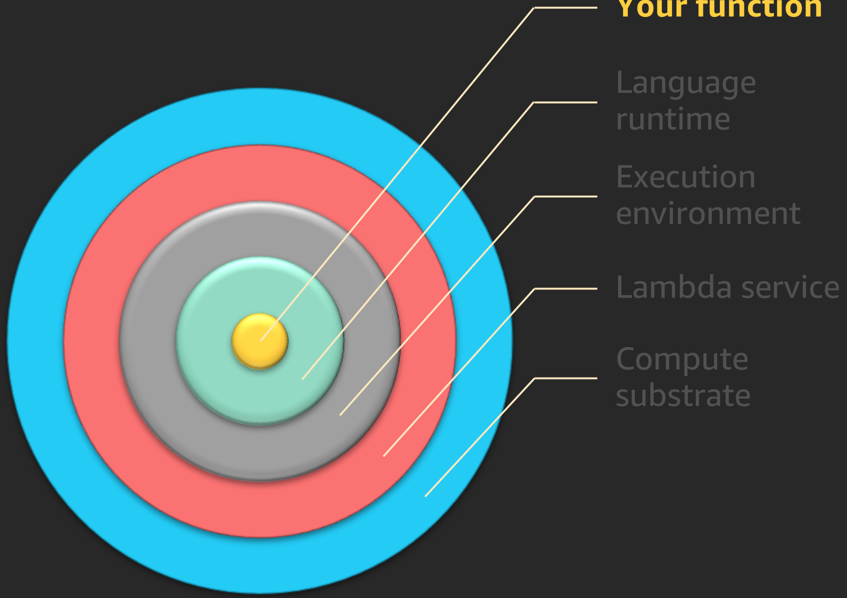
- Record contains details about the request and response in JSON format
- Contains more information than data sent to a DLQ
- Can send both outcomes to same destination

  or
- Can send success to one destination, failure to another



The best performing Lambda function is the one you rip out and replace with a built in integration

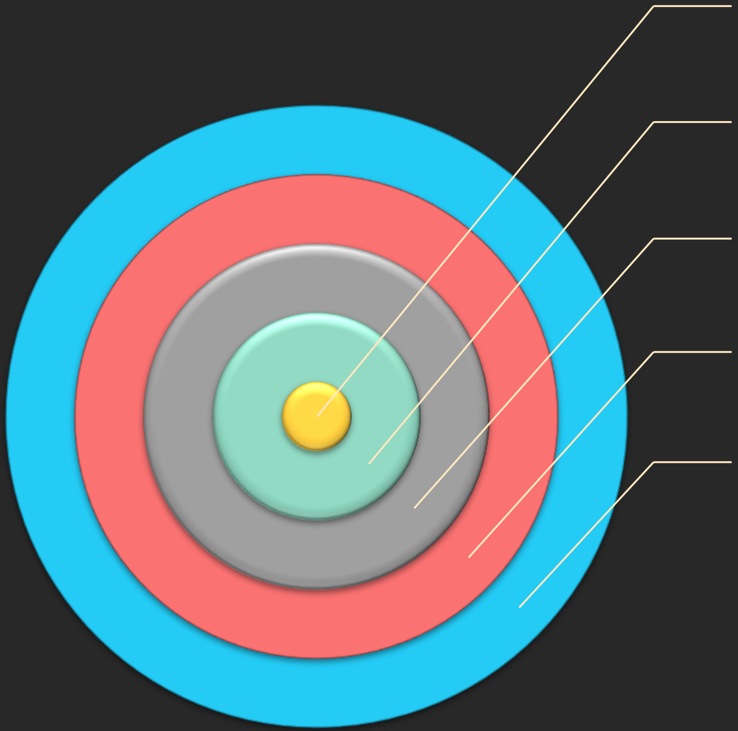
Anatomy of a Lambda function



Recap:

- Minimize dependencies
- Use pre-handler logic sparingly but strategically
- Share secrets based on application scope:
  - Single function: Env-Vars
  - Multi Function/shared environment: Parameter Store
- Think about how re-use impacts variables, connections, and dependency usage
- Layers save on code duplication and help enable standardization across functions
- Amazon RDS Proxy will simplify relational database usage with Lambda
- Concise logic
- Push orchestration up to Step Functions
- Lambda destinations will simplify and improve asynchronous workflows

Anatomy of a Lambda function



Your function

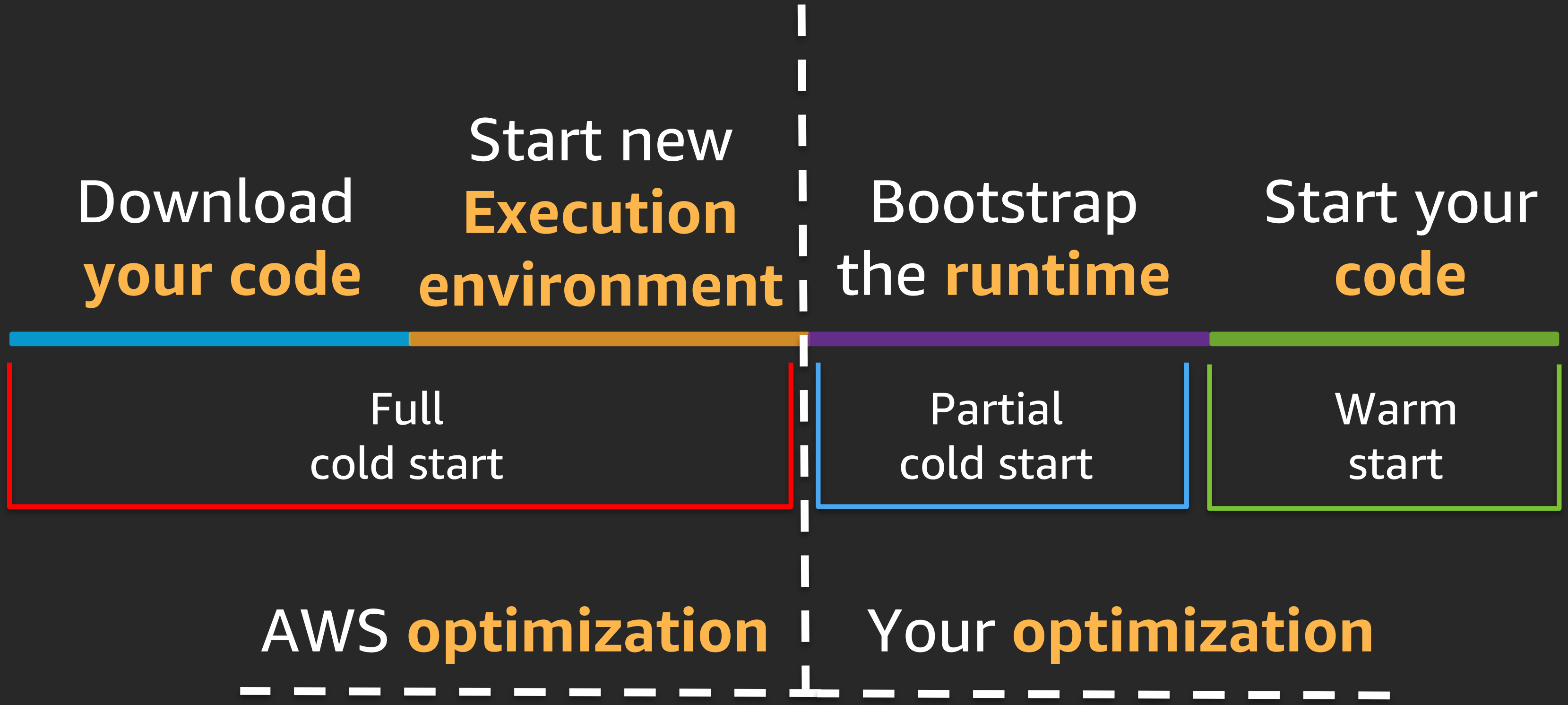
Language runtime

Execution environment

Lambda service

Compute substrate

The function lifecycle



Download **your code**

Start new **Execution environment**

Bootstrap the **runtime**

Start your **code**

Full cold start

Partial cold start

Warm start

AWS **optimization**

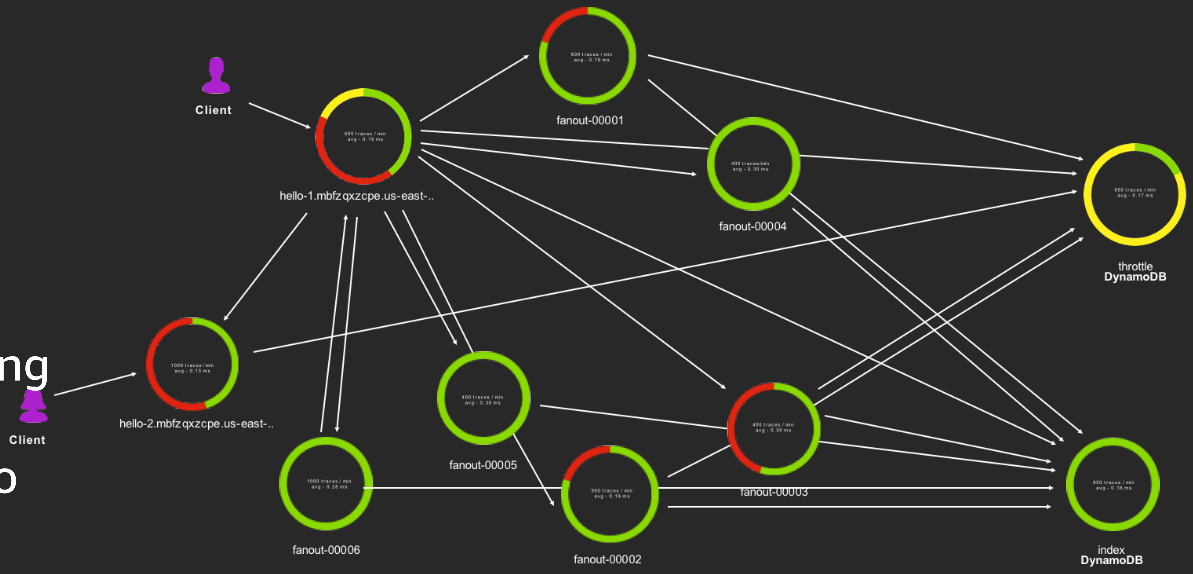
Your **optimization**

AWS X-Ray

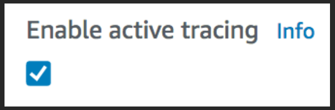
```
var AWSXRay = require(‘aws-xray-sdk-core‘);
var AWS = AWSXRay.captureAWS(require(‘aws-sdk’));
s3Client = AWS.S3();
```

Profile and troubleshoot serverless applications:

- Lambda instruments incoming requests for all supported languages and can capture calls made in code
- API Gateway inserts a tracing header into HTTP calls as well as reports data back to X-Ray itself



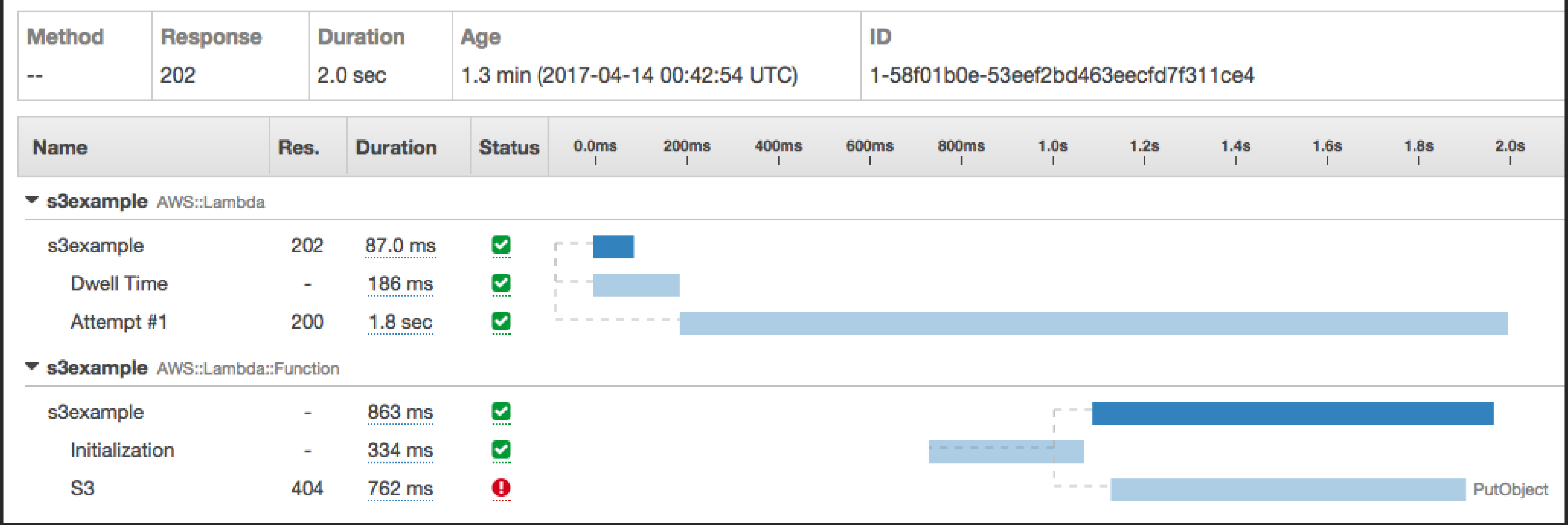
Enable X-Ray Tracing ☑



Enable active tracing Info

☑

X-Ray Trace Example



Method	Response	Duration	Age	ID
--	202	2.0 sec	1.3 min (2017-04-14 00:42:54 UTC)	1-58f01b0e-53eef2bd463eecfd7f311ce4

Name	Res.	Duration	Status
▼ **s3example** AWS::Lambda			
s3example	202	87.0 ms	✓
Dwell Time	-	186 ms	✓
Attempt #1	200	1.8 sec	✓
▼ **s3example** AWS::Lambda::Function			
s3example	-	863 ms	✓
Initialization	-	334 ms	✓
S3	404	762 ms	!

Tweak your function’s computer power

Lambda exposes only a memory control, with the **% of CPU core and network capacity** allocated to a function proportionally

Is your code CPU, Network, or memory-bound? If so, it could be **cheaper** to choose more memory.

Smart resource allocation

Match resource allocation (up to 3 GB!) to logic

Stats for Lambda function that calculates **1000 times** all prime numbers **<= 1000000**

128 MB	11.722965sec	$0.024628
256 MB	6.678945sec	$0.028035
512 MB	3.194954sec	$0.026830
1024 MB	1.465984sec	$0.024638

Green==Best   **Red**==Worst

Smart resource allocation

Match resource allocation (up to 3 GB!) to logic

Stats for Lambda function that calculates **1000 times** all prime numbers **<= 1000000**

128 MB	11.722965sec	$0.024628
256 MB	6.678945sec	$0.028035
512 MB	3.194954sec	$0.026830
1024 MB	1.465984sec	$0.024638

-10.256981sec

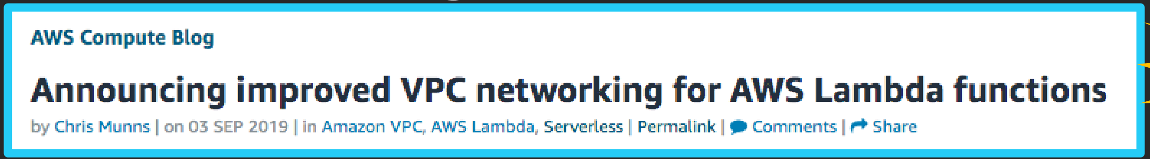
+$0.00001

Green==Best **Red**==Worst

Multithreading? Maybe!

- <1.8 GB is still single core
    - CPU bound workloads won’t see gains – processes share same resources
- >1.8 GB is multicore
    - CPU bound workloads will gains, but need to multithread
- I/O bound workloads WILL likely see gains
    - e.g. parallel calculations to return

Lambda + VPC, no longer a cold-start pain point!

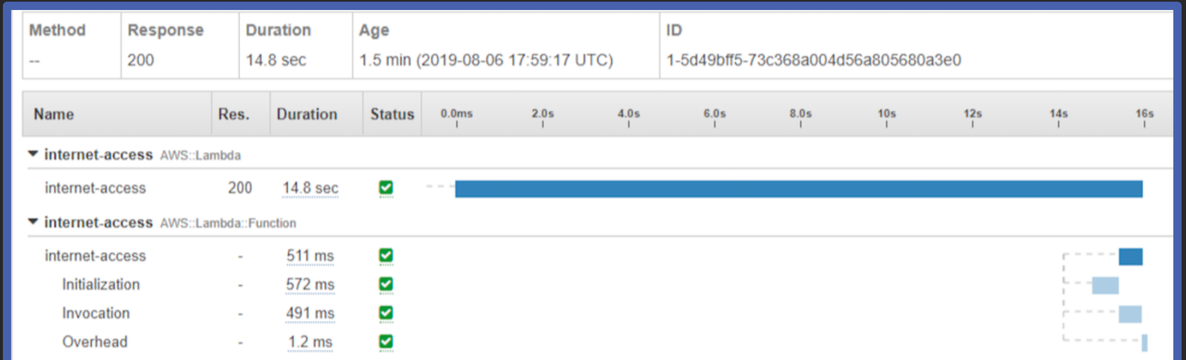


AWS Compute Blog

Announcing improved VPC networking for AWS Lambda functions

by Chris Munns | on 03 SEP 2019 | in Amazon VPC, AWS Lambda, Serverless | Permalink | Comments | Share

NEW!!!

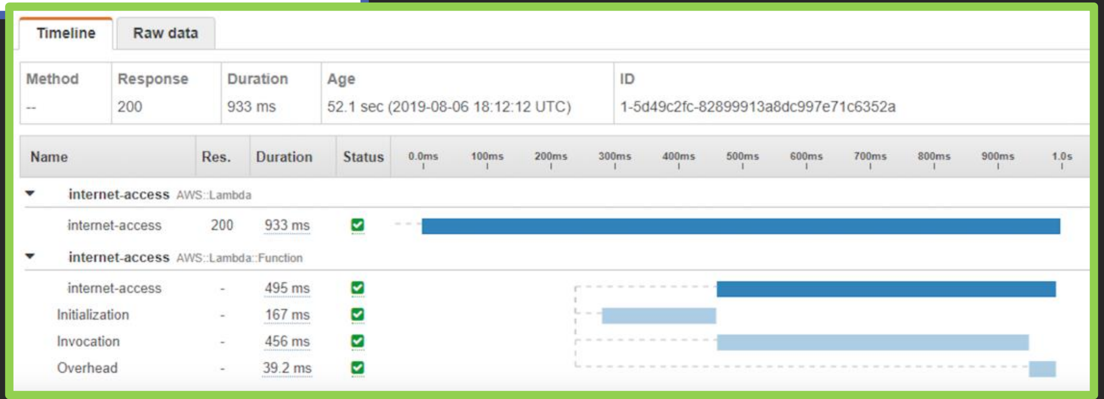


Method	Response	Duration	Age	ID
--	200	14.8 sec	1.5 min (2019-08-06 17:59:17 UTC)	1-5d49bff5-73c368a004d56a805680a3e0

Name	Res.	Duration	Status
▾ internet-access AWS::Lambda			
internet-access	200	14.8 sec	☑
▾ internet-access AWS::Lambda::Function			
internet-access	-	511 ms	☑
Initialization	-	572 ms	☑
Invocation	-	491 ms	☑
Overhead	-	1.2 ms	☑

← Before: 14.8-sec duration

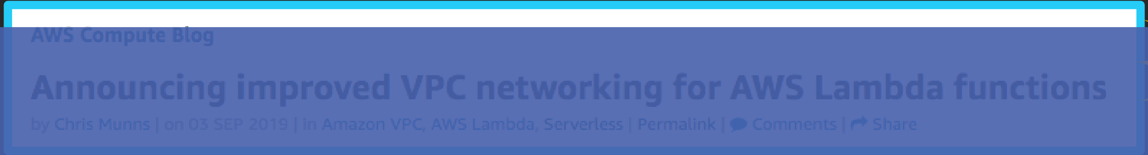
After: 933 ms duration →



Method	Response	Duration	Age	ID
--	200	933 ms	52.1 sec (2019-08-06 18:12:12 UTC)	1-5d49c2fc-82899913a8dc997e71c6352a

Name	Res.	Duration	Status
▾ internet-access AWS::Lambda			
internet-access	200	933 ms	☑
▾ internet-access AWS::Lambda::Function			
internet-access	-	495 ms	☑
Initialization	-	167 ms	☑
Invocation	-	456 ms	☑
Overhead	-	39.2 ms	☑

Lambda + VPC, no longer a cold-start pain point!

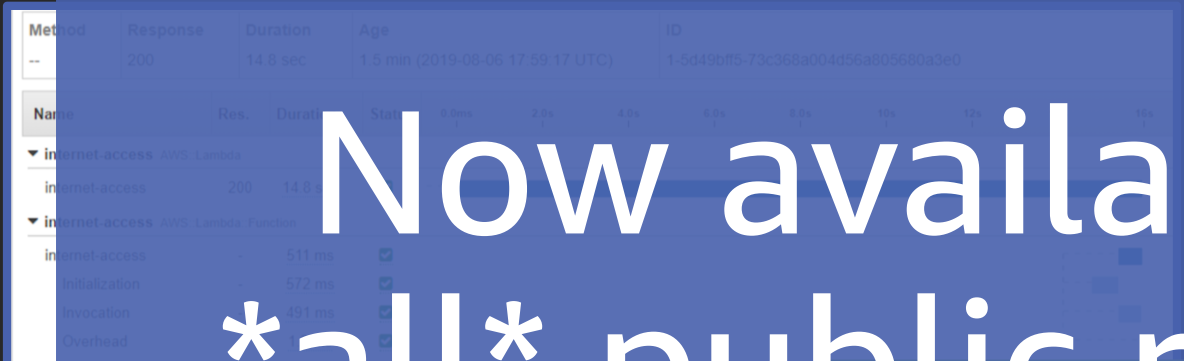


AWS Compute Blog

Announcing improved VPC networking for AWS Lambda functions

by Chris Munns | on 03 SEP 2019 | in Amazon VPC, AWS Lambda, Serverless | Permalink | Comments | Share

NEW!!!



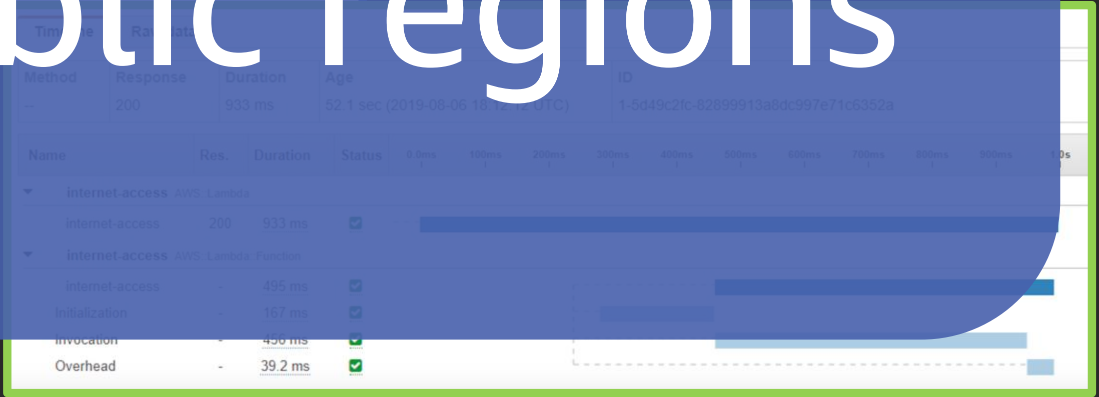
Method	Response	Duration	Age	ID
--	200	14.8 sec	1.5 min (2019-08-06 17:59:17 UTC)	1-5d49bff5-73c368a004d56a805680a3e0

Name	Res.	Duration	Status
internet-access AWS::Lambda			
internet-access	200	14.8 s	
internet-access AWS::Lambda::Function			
internet-access	-	511 ms	☑
Initialization	-	572 ms	☑
Invocation	-	491 ms	☑
Overhead		1	☑

← Before: 14.8 sec duration

Now available in *all* public regions

After: 933ms duration →



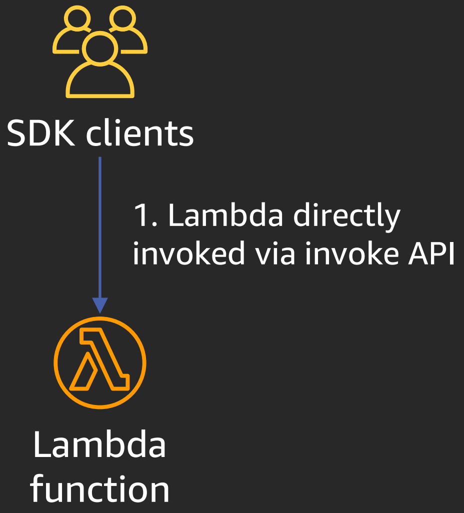
Timeline Raw data

Method	Response	Duration	Age	ID
--	200	933 ms	52.1 sec (2019-08-06 18:12:12 UTC)	1-5d49c2fc-82899913a8dc997e71c6352a

Name	Res.	Duration	Status
internet-access AWS::Lambda			
internet-access	200	933 ms	☑
internet-access AWS::Lambda::Function			
internet-access	-	495 ms	☑
Initialization	-	167 ms	☑
Invocation	-	456 ms	☑
Overhead	-	39.2 ms	☑

Events and you

Lambda API



API provided by the Lambda service

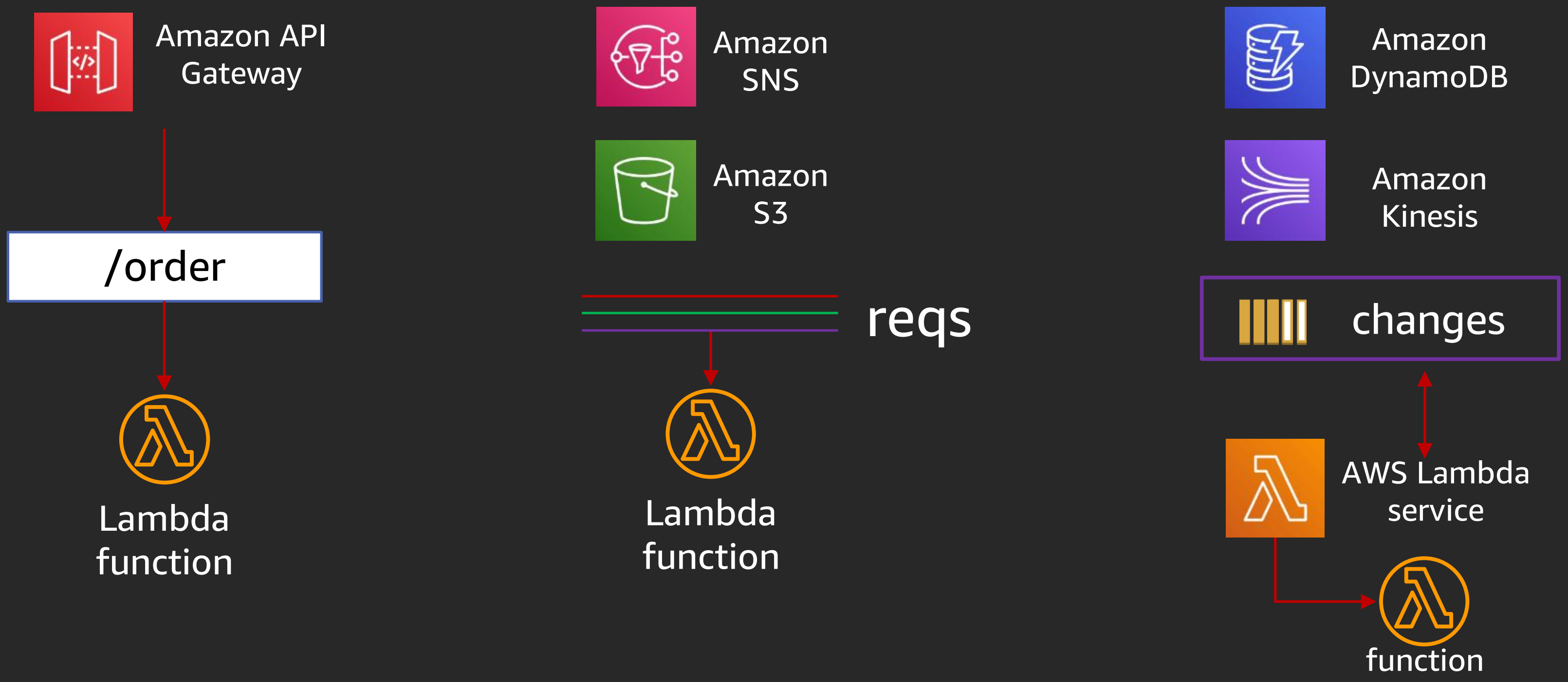
Used by all other services that invoke Lambda across all models

Supports sync and async

Can pass any event payload structure you want

Client included in every SDK

Lambda execution model



Synchronous (push)

Amazon API Gateway

/order

Lambda function

Asynchronous (event)

Amazon SNS

Amazon S3

reqs

Lambda function

Stream (Poll-based)

Amazon DynamoDB

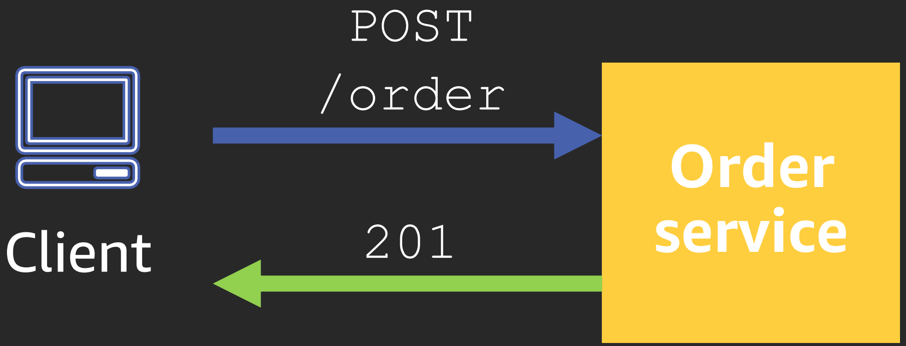
Amazon Kinesis

changes

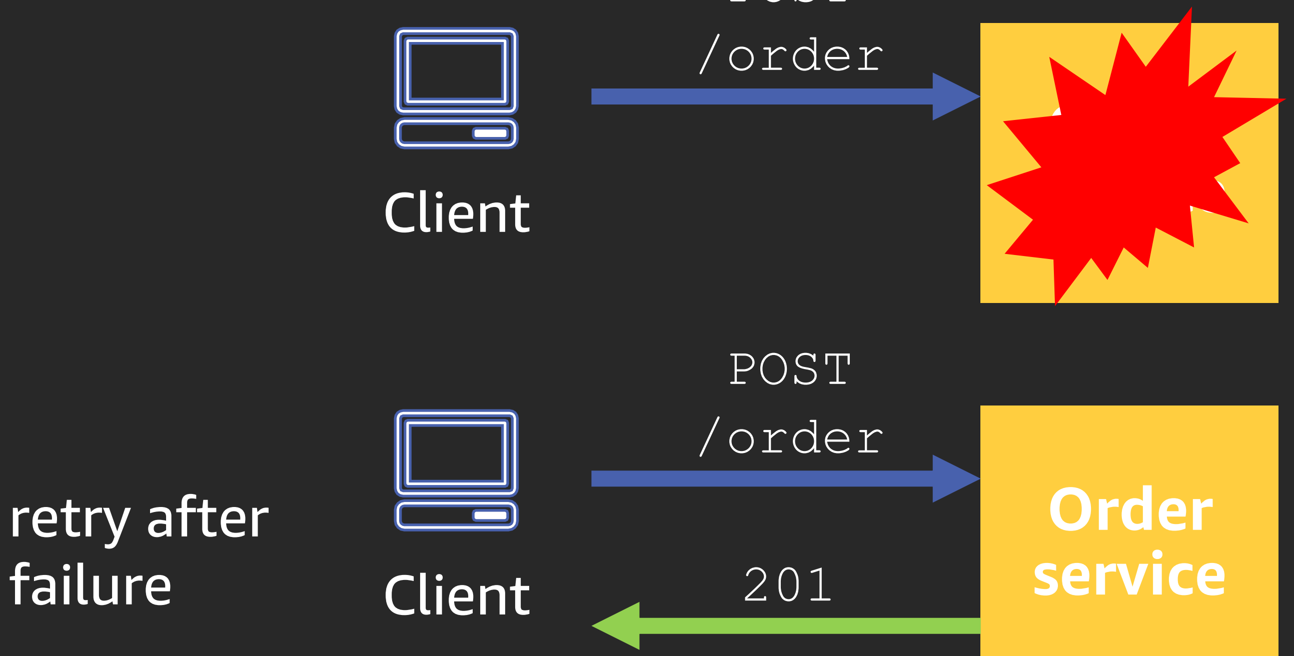
AWS Lambda service

function

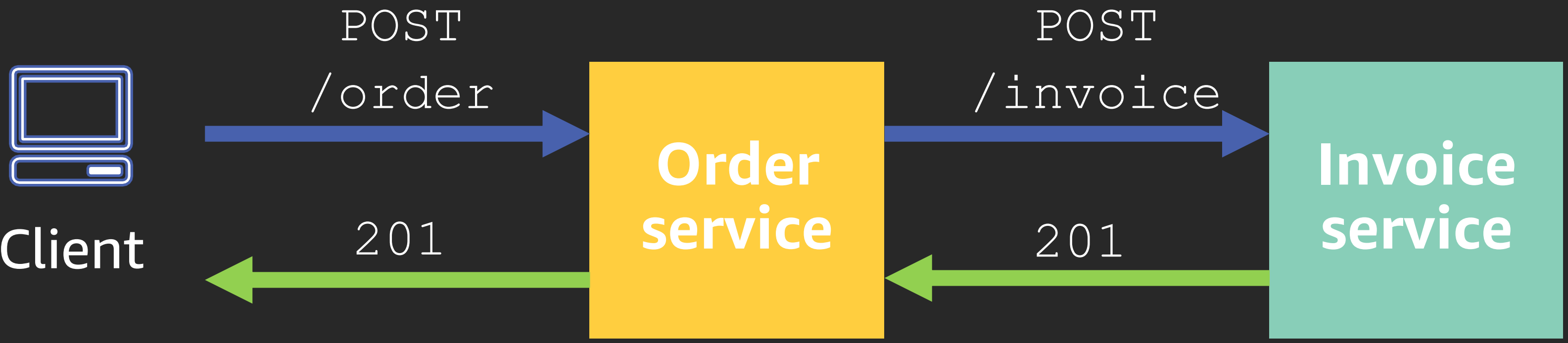
Synchronous APIs



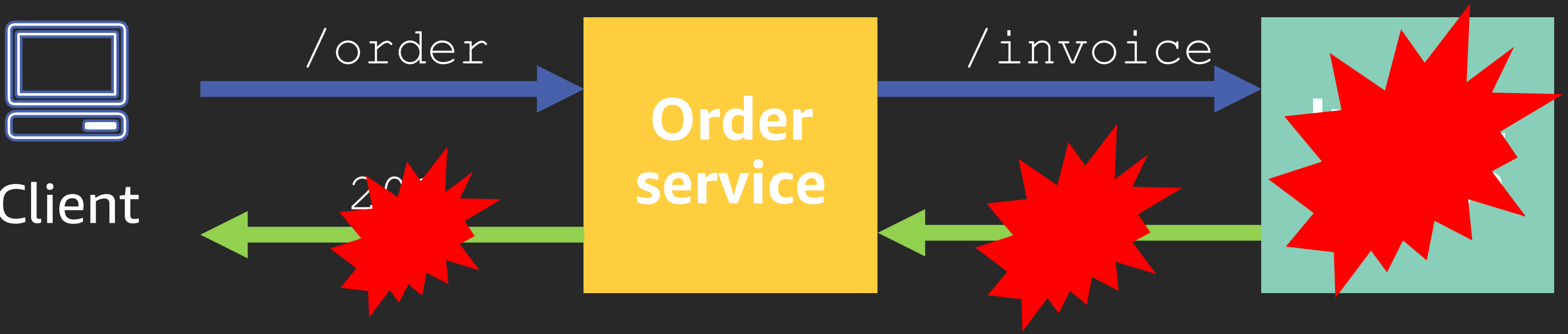
Synchronous APIs



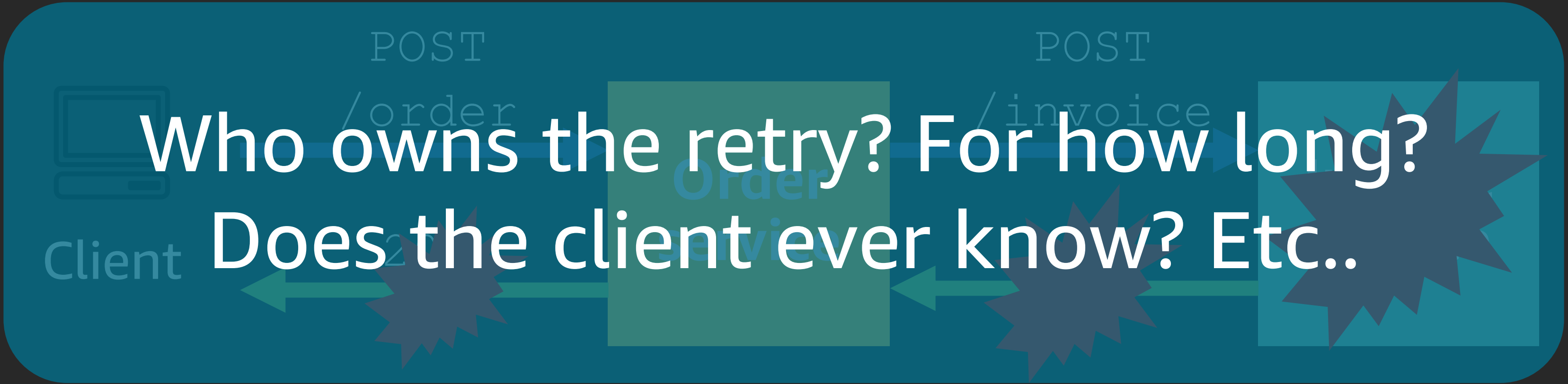
Synchronous APIs



Synchronous APIs



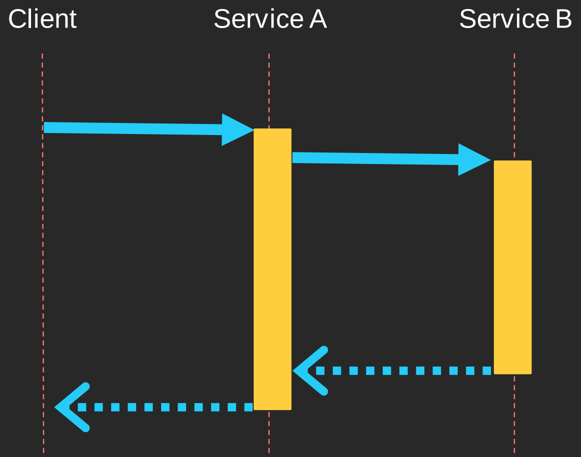
Synchronous APIs



Who owns the retry? For how long?
Does the client ever know? Etc..

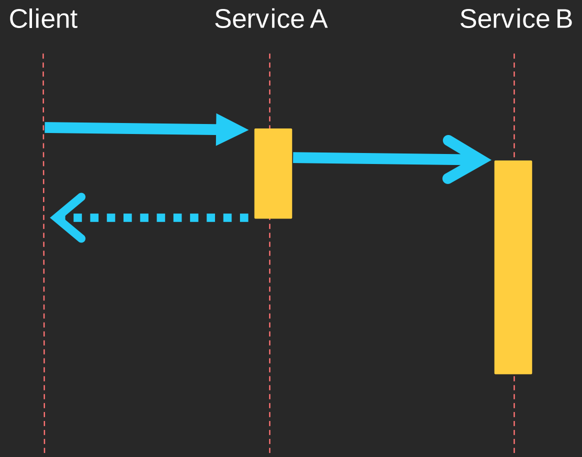
This effectively creates a “tight coupling”
where failures become harder to recover from

Thinking asynchronously



Synchronous

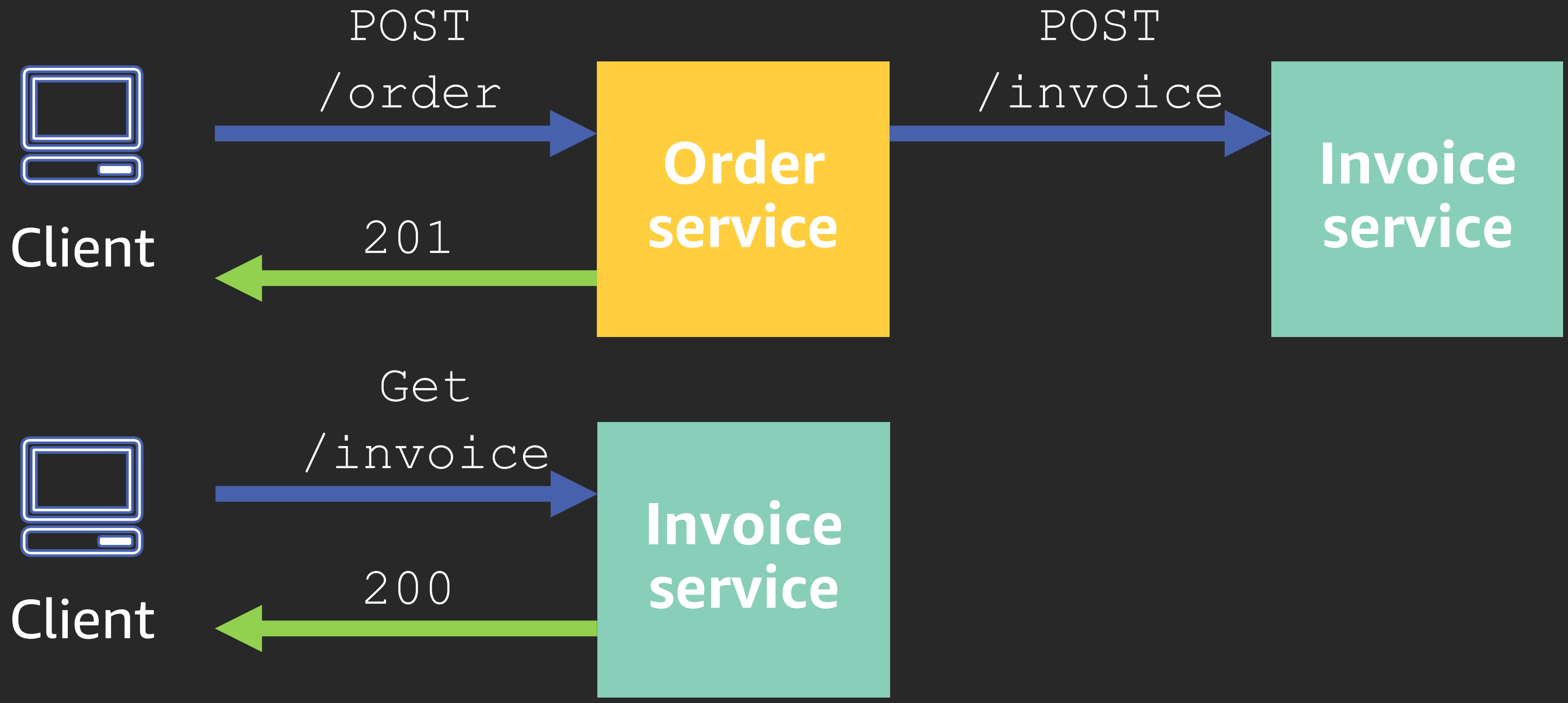
commands



Asynchronous

events

Asynchronous APIs



If you don’t need a response, execute async

Use the Lambda APIs to start an asynchronous execution

- Built-in queue (Amazon SQS behind the scenes)
- Automatic retries
- Dead letter queue for failed events

```
client =
boto3.client("lambda")

client.invoke_async(
    FunctionName="test"
    InvokeArgs=json_payload
)
```

Topics, streams, queues, and buses

Amazon SNS

Amazon SQS

Amazon EventBridge

Amazon Kinesis
Data Streams

Ways to compare

Scale/concurrency controls

Durability

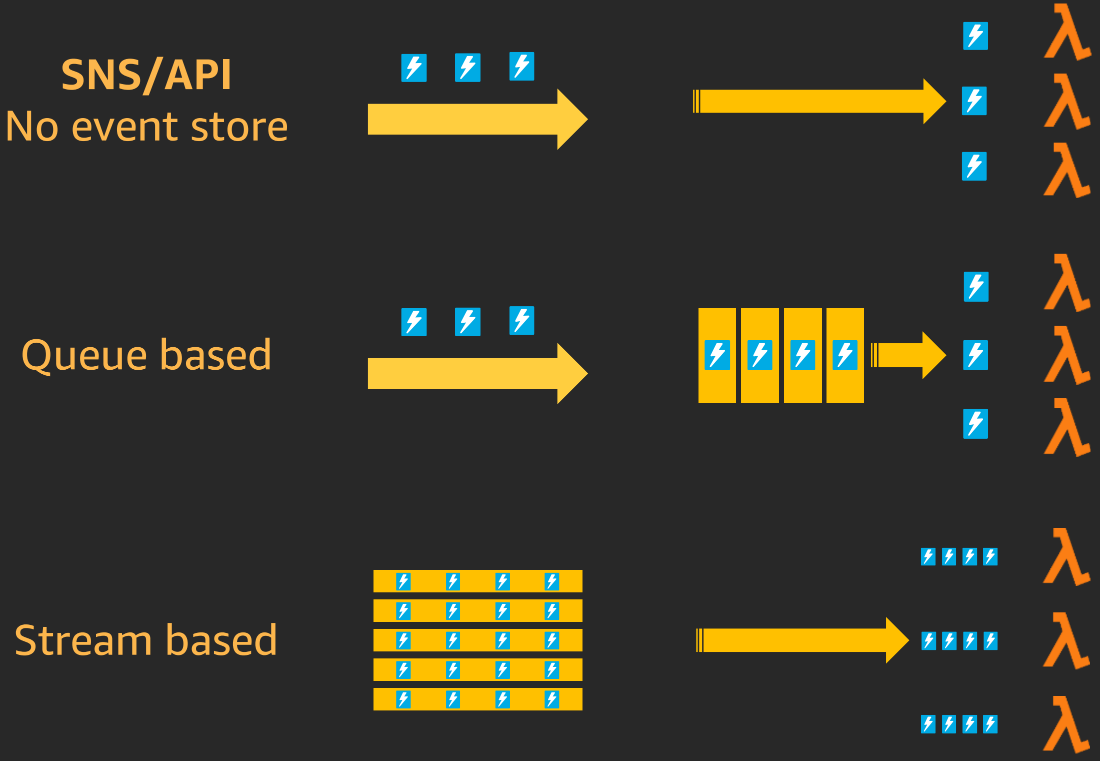
Persistence

Consumption models

Retries

Pricing

Concurrency across models

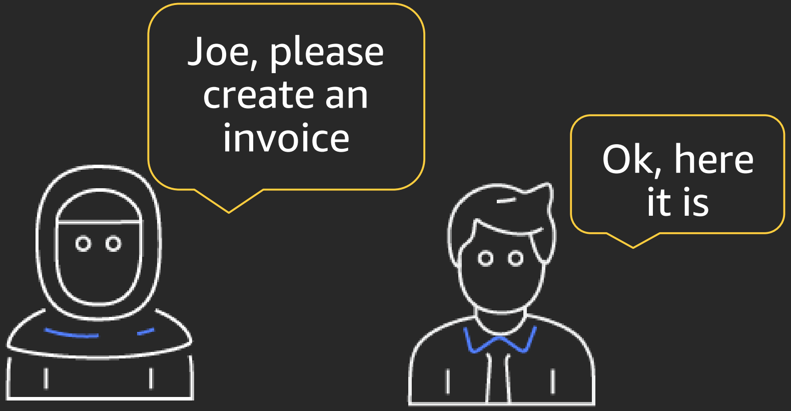


Recent announcements for async event sources

- Amazon SQS FIFO as invoke source for Lambda
- Amazon SNS Dead Letter Queues (DLQs)
- For streamed events:
  - MaximumRetryAttempts, MaximumRecordAgeInSeconds, BisectBatchOnFunctionError, On-failure destination
  - Batch Window
  - Parallelization Factor
- For async events:
  - MaximumRetryAttempts
  - MaximumEventAgeInSeconds

ICYMI: Serverless pre:Invent 2019

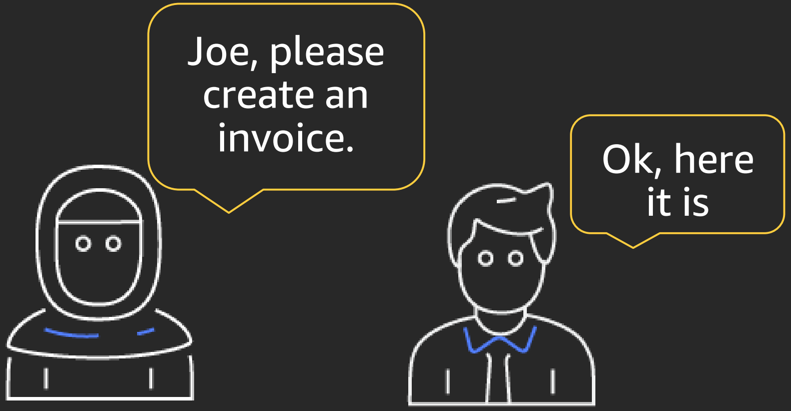
Directed vs. Observable events



Directed

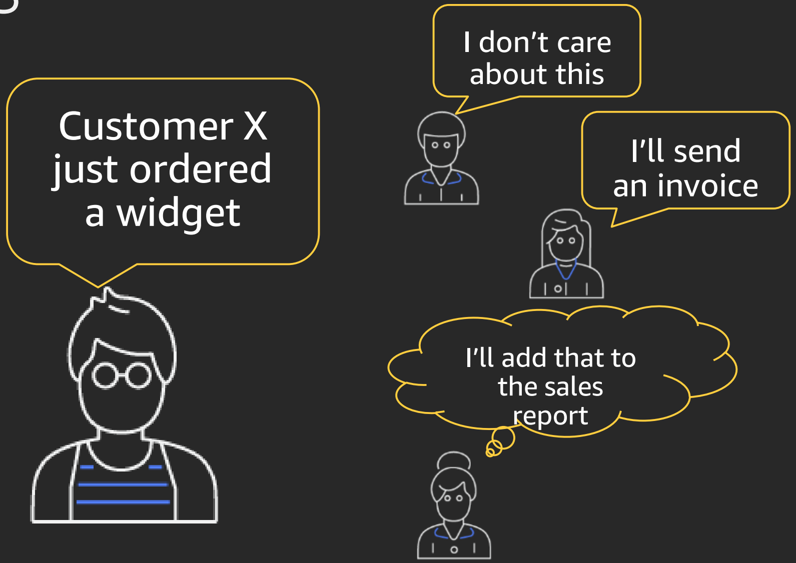
Commands

Directed vs. Observable events



Directed

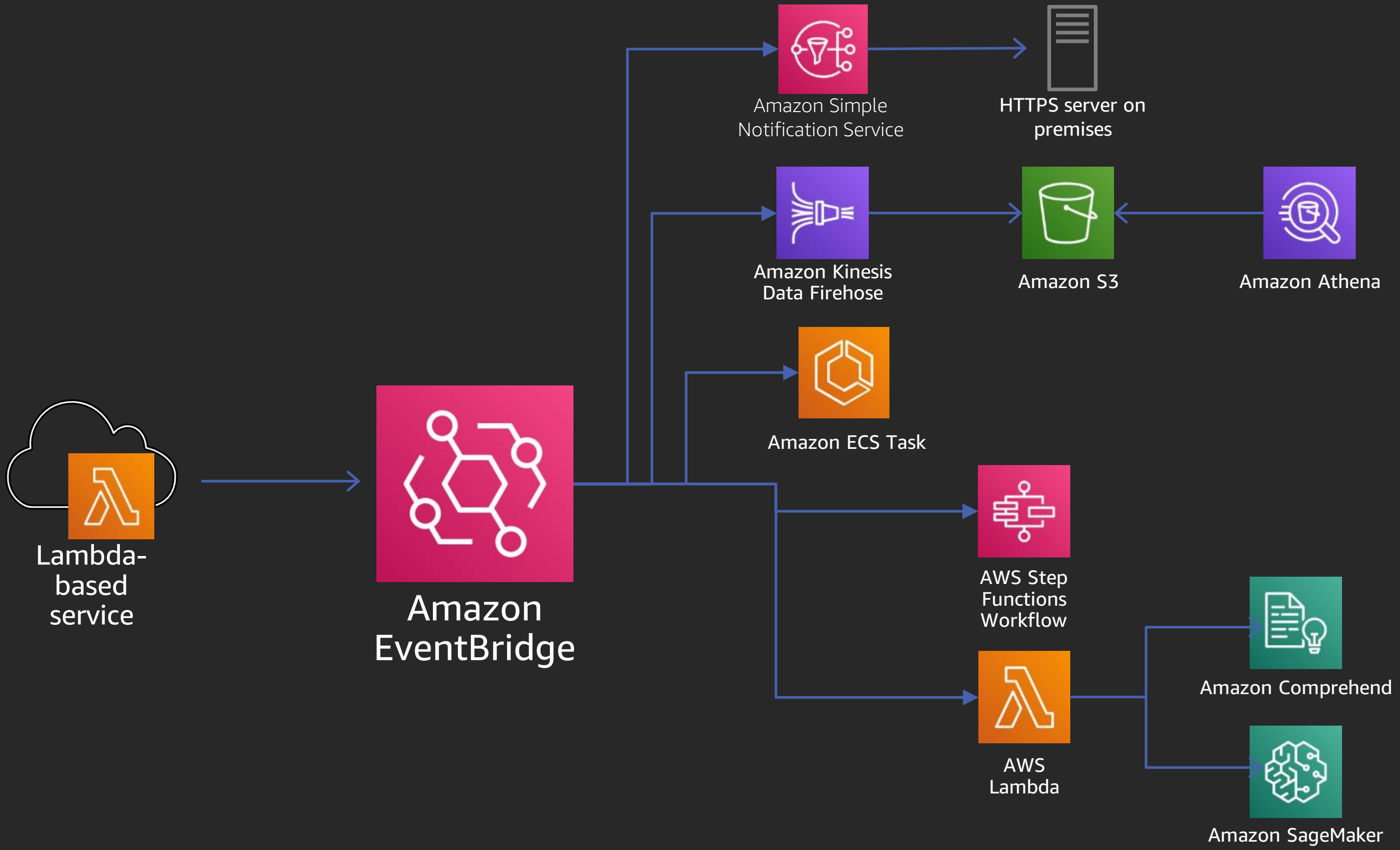
Commands


Customer X just ordered a widget
I don't care about this
I'll send an invoice
I'll add that to the sales report

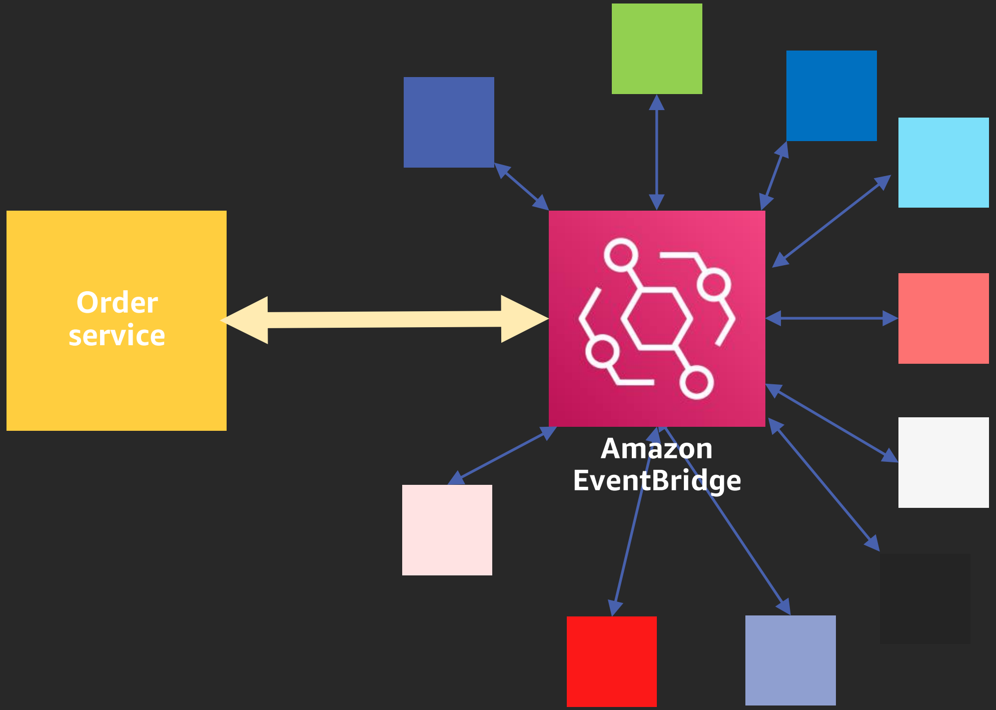
Observable

Events

Event passing with EventBridge



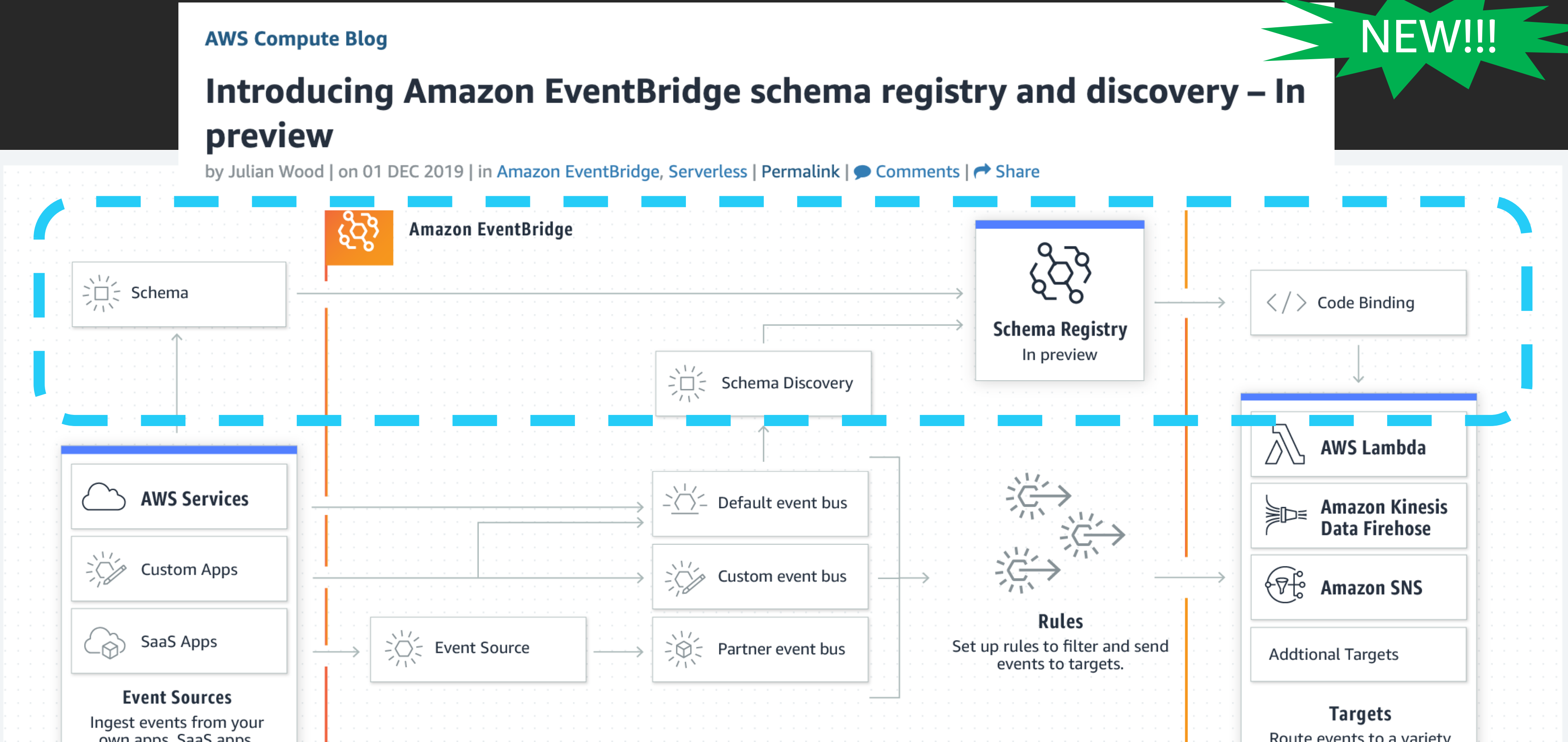
Events with EventBridge



- Your services can both produce messages onto the bus and consume just the messages they need from the bus
- Services don’t need to know about each other, just about the bus.

Recent announcements for async event sources

NEW!!!



AWS Compute Blog

Introducing Amazon EventBridge schema registry and discovery – In preview

by Julian Wood | on 01 DEC 2019 | in Amazon EventBridge, Serverless | Permalink | Comments | Share

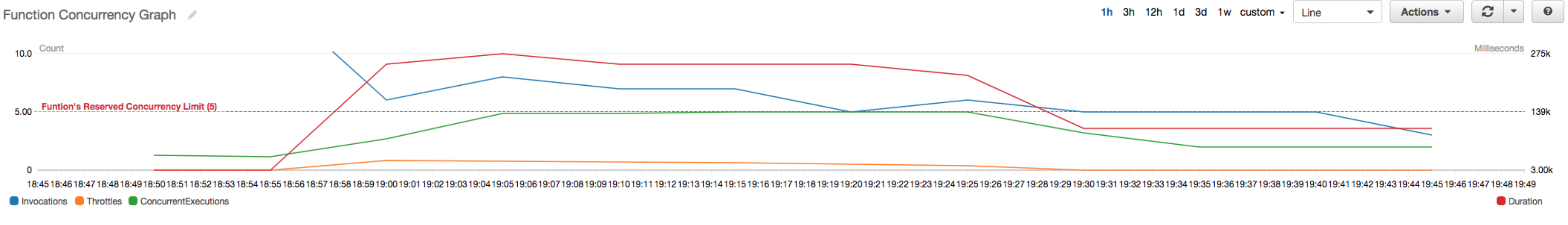
Rules
Set up rules to filter and send events to targets.

Event Sources
Ingest events from your own apps, SaaS apps

Targets
Route events to a variety

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 downstream resources like databases
- “Kill switch” – set per function concurrency to zero



Lambda Dead Letter Queues

"By default, a failed Lambda function invoked asynchronously is retried twice, and then the event is discarded." – 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

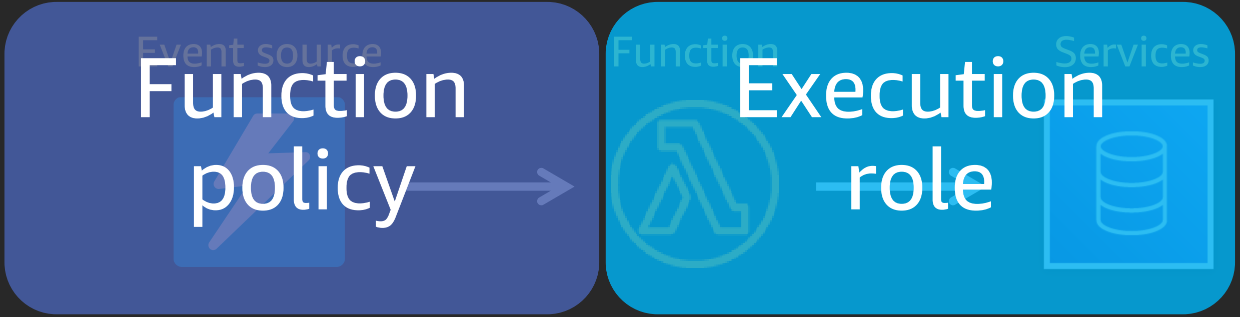
In many ways, Lambda Destinations supersedes this

Friends don’t let friends "Action": "s3:*"

Lambda permissions model

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

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



Function policy

Execution role

AWS Serverless Application Model (AWS SAM)

AWS CloudFormation extension optimized for serverless

Special serverless resource types: functions, APIs, tables, layers, and applications

Supports anything AWS CloudFormation supports

Open specification (Apache 2.0)

https://aws.amazon.com/serverless/sam

AWS SAM template

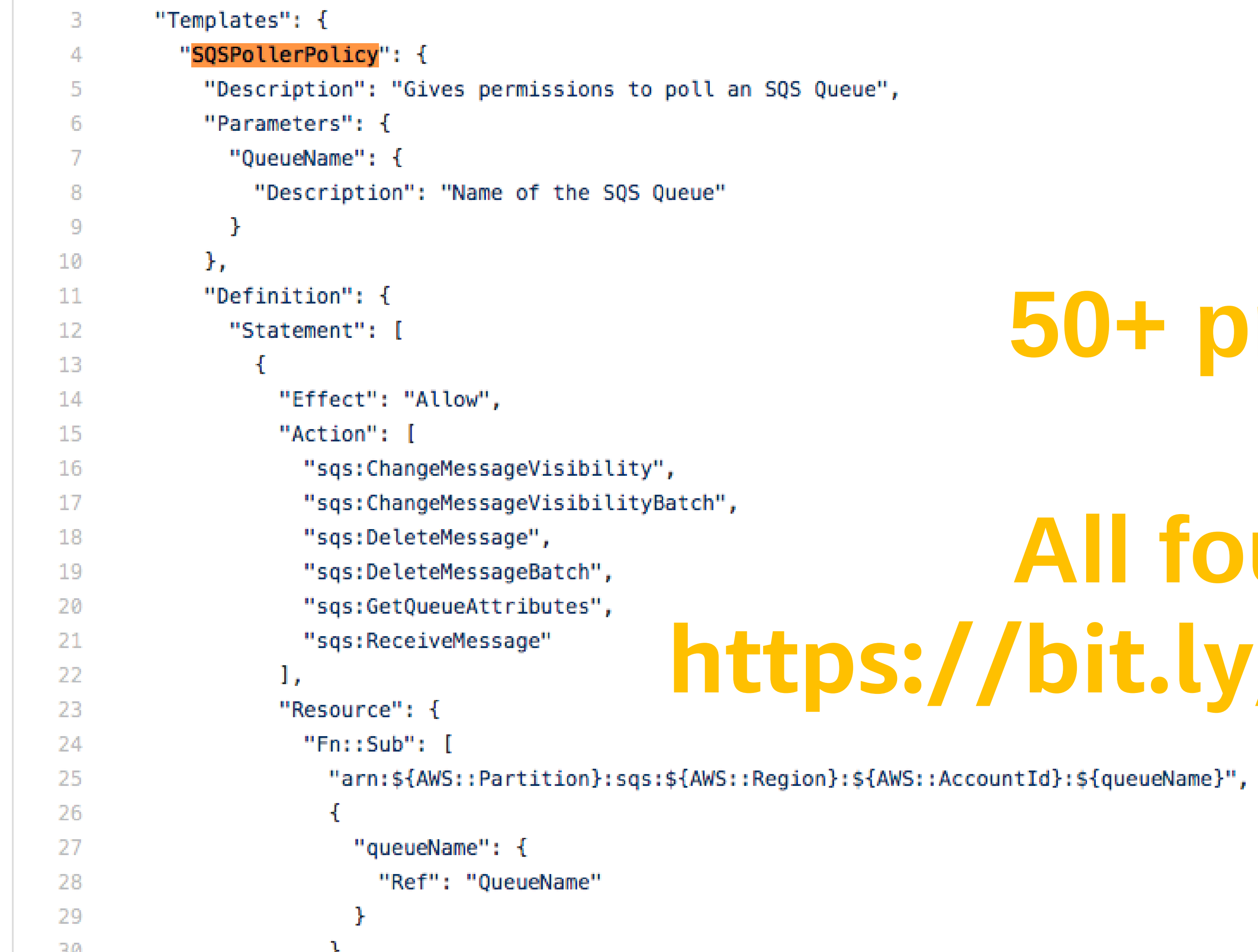
```
AWSTemplateFormatVersion: '2010-09-09’
Transform: AWS::Serverless-2016-10-31
Resources:
 GetProductsFunction:
  Type: AWS::Serverless::Function
  Properties:
   Handler: index.getProducts
   Runtime: nodejs8.10
   CodeUri: src/
   Policies:
    - DynamoDBReadPolicy:
      TableName: !Ref ProductTable
   Events:
    GetResource:
     Type: Api
     Properties:
      Path: /products/{productId}
      Method: get
 ProductTable:
  Type: AWS::Serverless::SimpleTable
```

Just 20 lines to create:

- Lambda function
- IAM role
- API Gateway
- DynamoDB table

AWS SAM policy templates

```
GetProductsFunction:
  Type: AWS::Serverless::Function
  Properties:
      ...
    Policies:
          - DynamoDBReadPolicy:
          TableName: !Ref ProductTable
      ...
ProductTable:
  Type: AWS::Serverless::SimpleTable
```



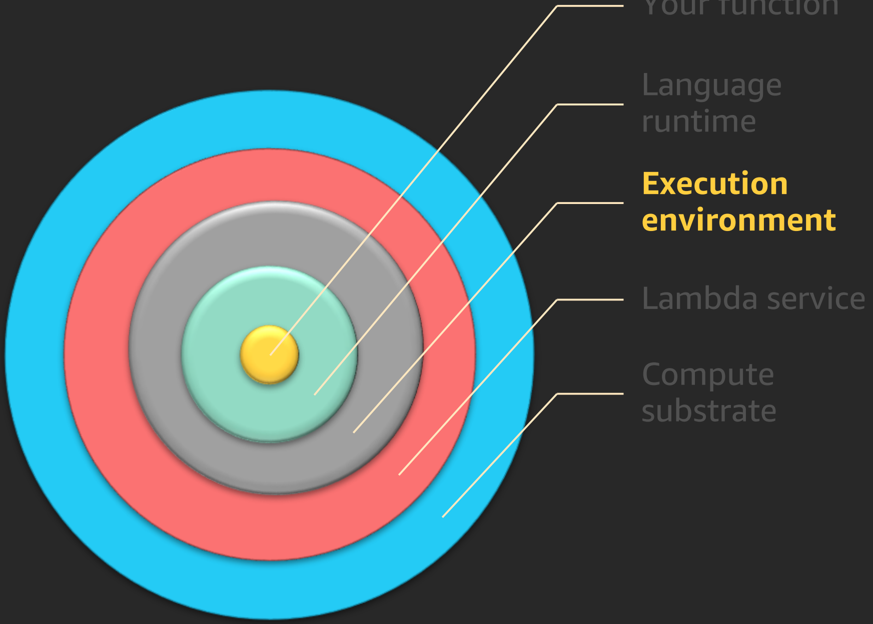
```
    "Templates": {
      "SQSPollerPolicy": {
        "Description": "Gives permissions to poll an SQS Queue",
        "Parameters": {
          "QueueName": {
            "Description": "Name of the SQS Queue"
          }
        },
        "Definition": {
          "Statement": [
            {
              "Effect": "Allow",
              "Action": [
                "sqs:ChangeMessageVisibility",
                "sqs:ChangeMessageVisibilityBatch",
                "sqs:DeleteMessage",
                "sqs:DeleteMessageBatch",
                "sqs:GetQueueAttributes",
                "sqs:ReceiveMessage"
              ],
              "Resource": {
                "Fn::Sub": [
                  "arn:${AWS::Partition}:sqs:${AWS::Region}:${AWS::AccountId}:${queueName}",
                  {
                    "queueName": {
                      "Ref": "QueueName"
                    }
                  }
```

50+ predefined policies

All found here:

https://bit.ly/2xWycnj

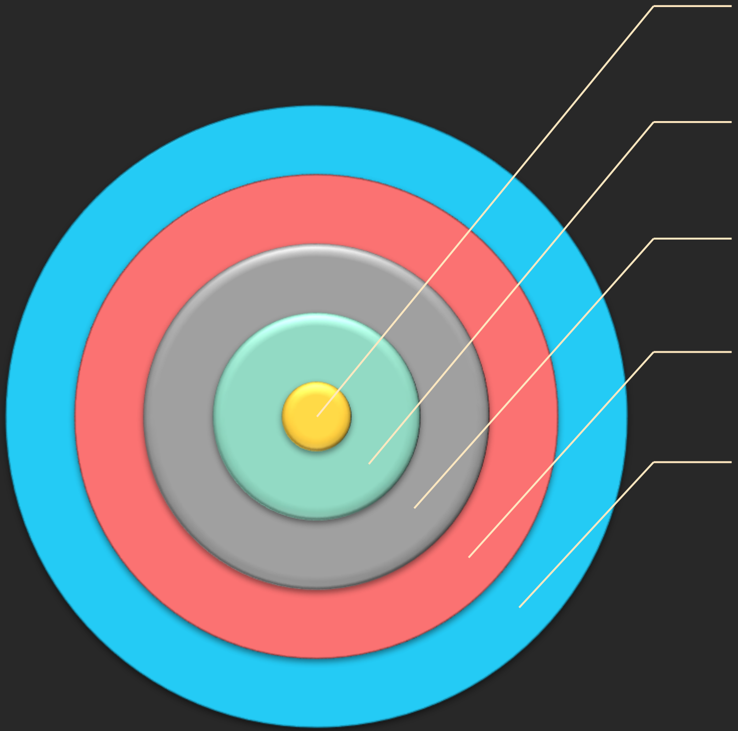
Anatomy of a Lambda function



Recap:

- More memory == More CPU and I/O (proportionally)
  - Can also be lower cost
- Use AWS X-Ray to profile your workload
- >1.8GB memory get's you 2 cores, but you might not use/need it
- Think deeply about your execution model and invocation source needs
  - Not everything needs to be an API
- Thinking async will get you over some of the biggest scaling challenges
- Understand the various aspects to queues, topics, streams, and event buses when using them
- Minimize the scope of IAM permissions
  - Leverage tooling like AWS SAM

Anatomy of a Lambda function



Your function

Language runtime

Execution environment

Lambda service

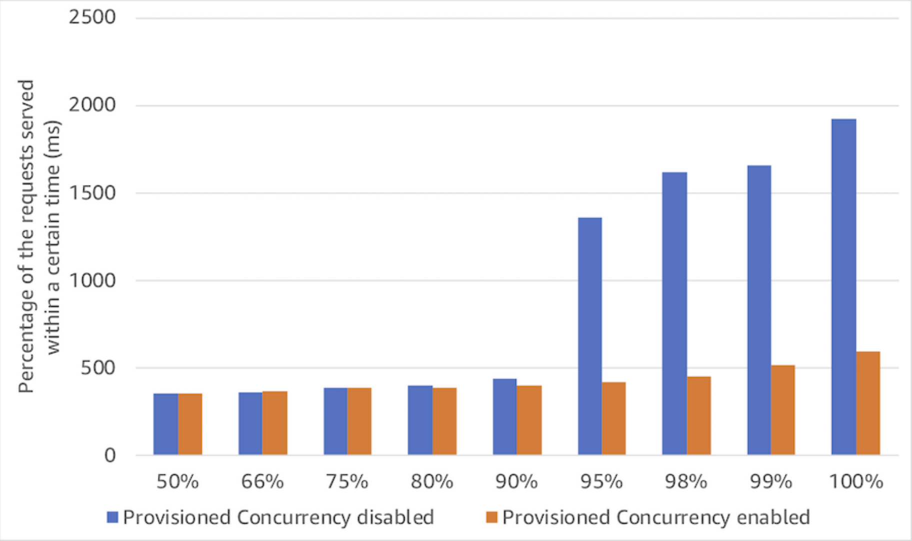
Compute substrate

Introducing Provisioned Concurrency for Lambda

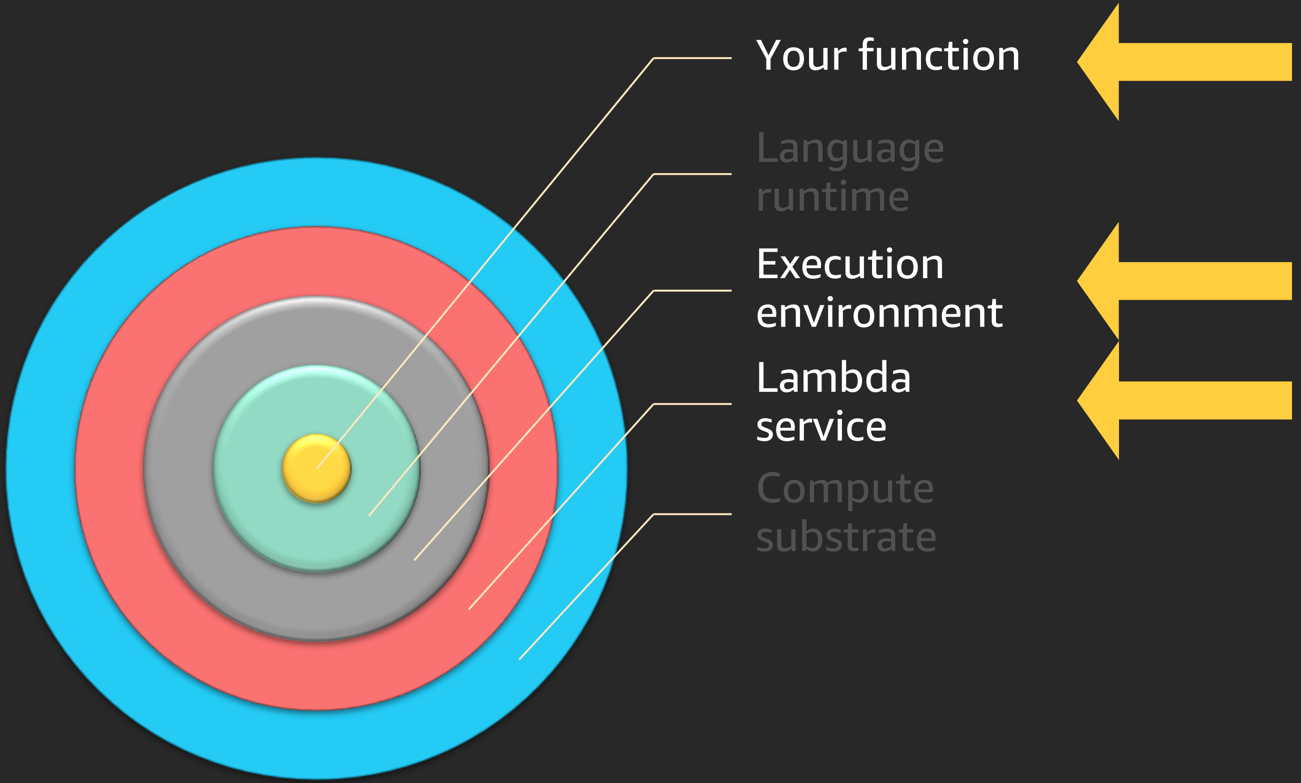
NEW!!!

Pre-creates execution environments all the way up through the INIT phase.

- Mostly for interactive workloads that are heavily latency sensitive
- Greatly improved consistency across the full long tail of performance
- Little to no changes to your code or way you use Lambda
- Works with AWS Auto Scaling
- Adds a cost factor for per concurrency provisioned but a lower duration cost for execution
    - This could end up saving you money when heavily utilized



Anatomy of a Lambda function



FIN/ACK

Your Function Recap:

- Minimize dependencies
- Use pre-handler logic sparingly but strategically
- Share secrets based on application scope:
  - Single function: Env-Vars
  - Multi Function/shared environment: Parameter Store
- Think about how re-use impacts variables, connections, and dependency usage
- Layers save on code duplication and help enable standardization across functions
- Amazon RDS Proxy will simplify relational database usage with Lambda
- Concise logic
- Push orchestration up to AWS Step Functions
- Lambda destinations will simplify and improve asynchronous workflows

Execution Environment Recap:

- More memory == More CPU and I/O (proportionally)
  - Can also be lower cost
- Use AWS X-Ray to profile your workload
- >1.8 GB memory gets you 2 cores, but you might not use/need it
- Think deeply about your execution model and invocation source needs
  - Not everything needs to be an API
- Thinking async will get you over some of the biggest scaling challenges
- Understand the various aspects to queues, topics, streams and event buses when using them
- Minimize the scope of IAM permissions
  - Leverage tooling like AWS SAM

Lambda Service Recap:

- Provisioned Concurrency will improve consistency and overall latency of function execution

Learn Serverless with AWS Training and Certification

Resources created by the experts at AWS to help you learn modern application development

Free, on-demand courses on serverless, including:

- Introduction to Serverless Development
- Getting in the Serverless Mindset
- AWS Lambda Foundations
- Amazon API Gateway for Serverless Applications
- Amazon DynamoDB for Serverless Architectures

Additional digital and classroom trainings cover modern application development and computing

Visit the Learning Library at https://aws.training

Thank you!

Chris Munns

munns@amazon.com
@chrismunns

!

Please complete the session survey in the mobile app.

AWS re:Invent

aws