
AWS
re:Invent

SVS343-R1

Building microservices with AWS Lambda

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 2015 – Feb 2017
- 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

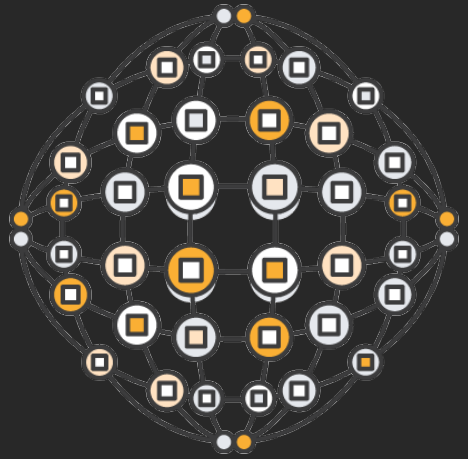
Why are we here today?

containers != microservices

microservices != containers

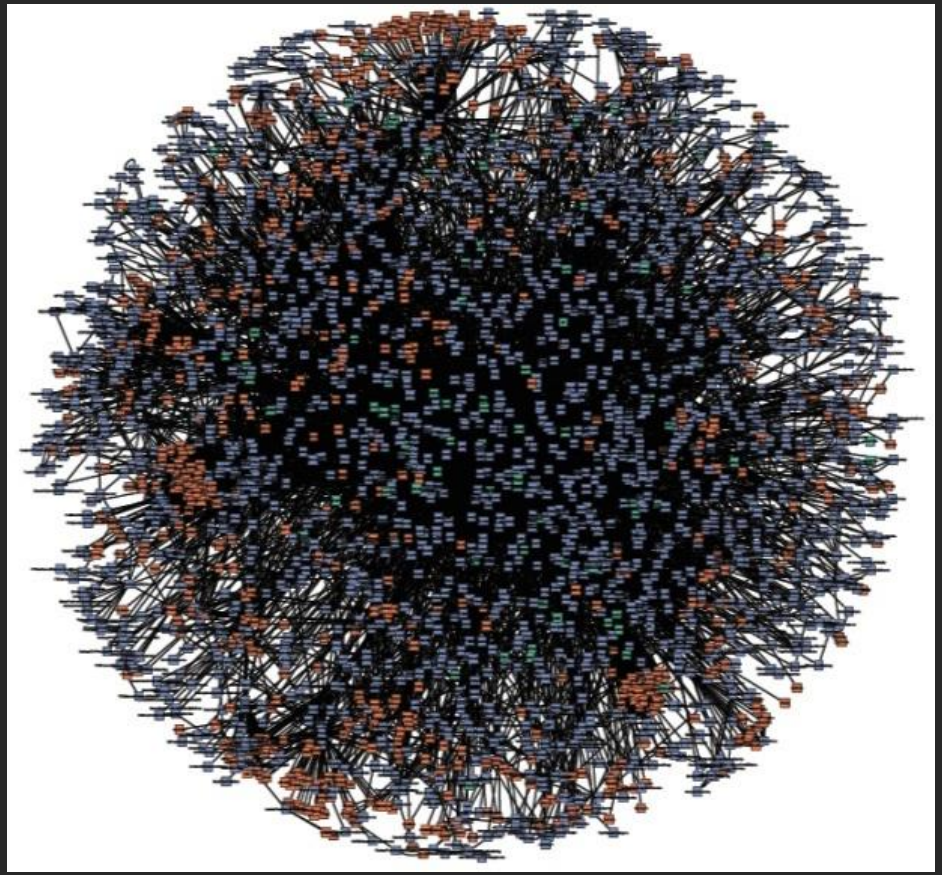
Today's focus:

AWS Lambda

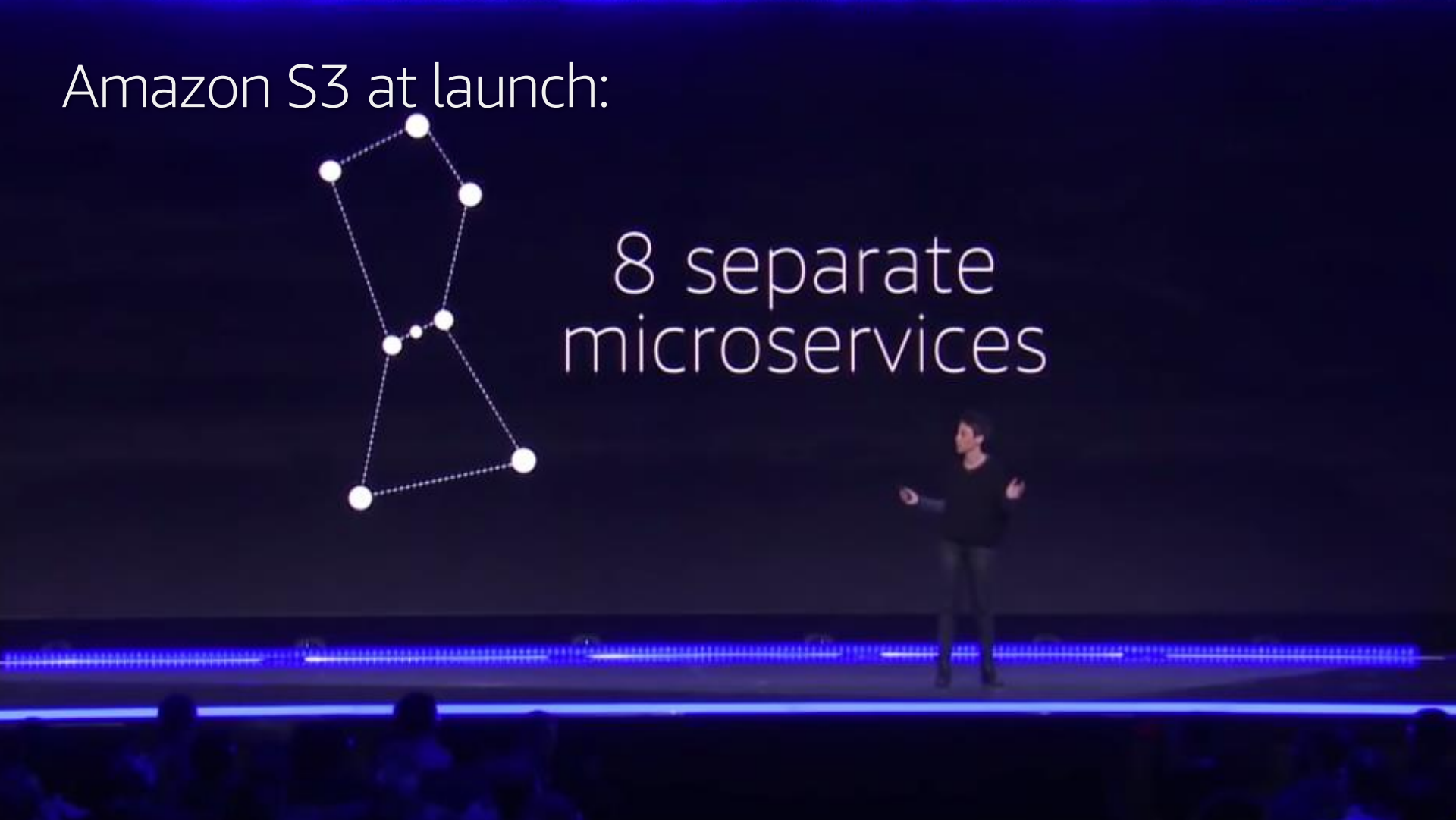


microservices

Microservices @ Amazon/AWS



Amazon S3 at launch:

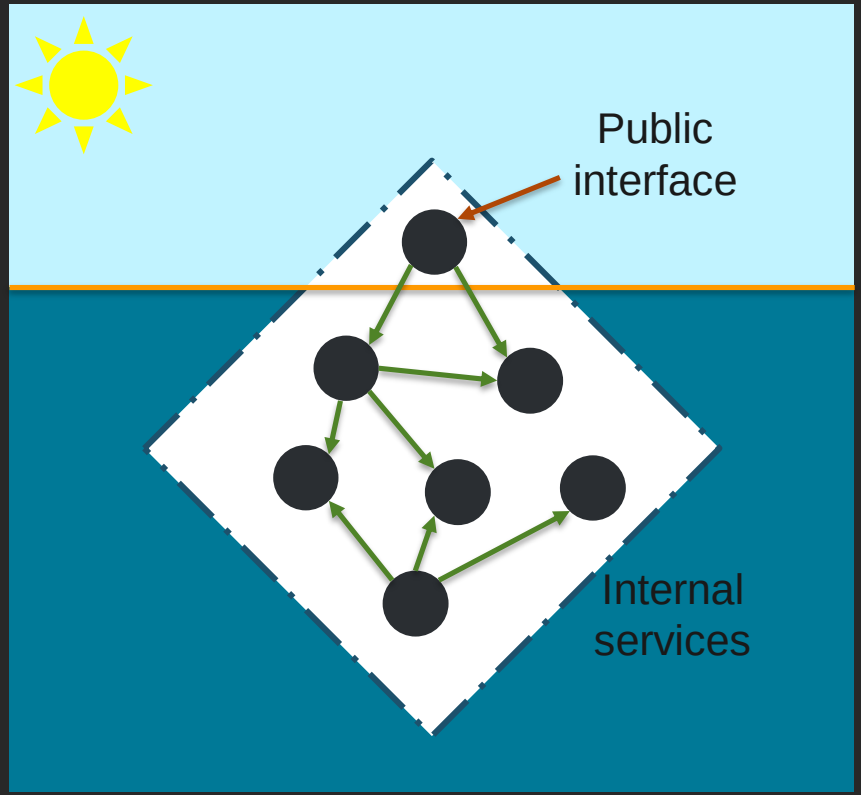



Amazon S3 @ re:Invent 2018:
More than 235
distributed
microservices

View of a single service

APIs in the front

Async in the back



Common microservices patterns at AWS

API in the front

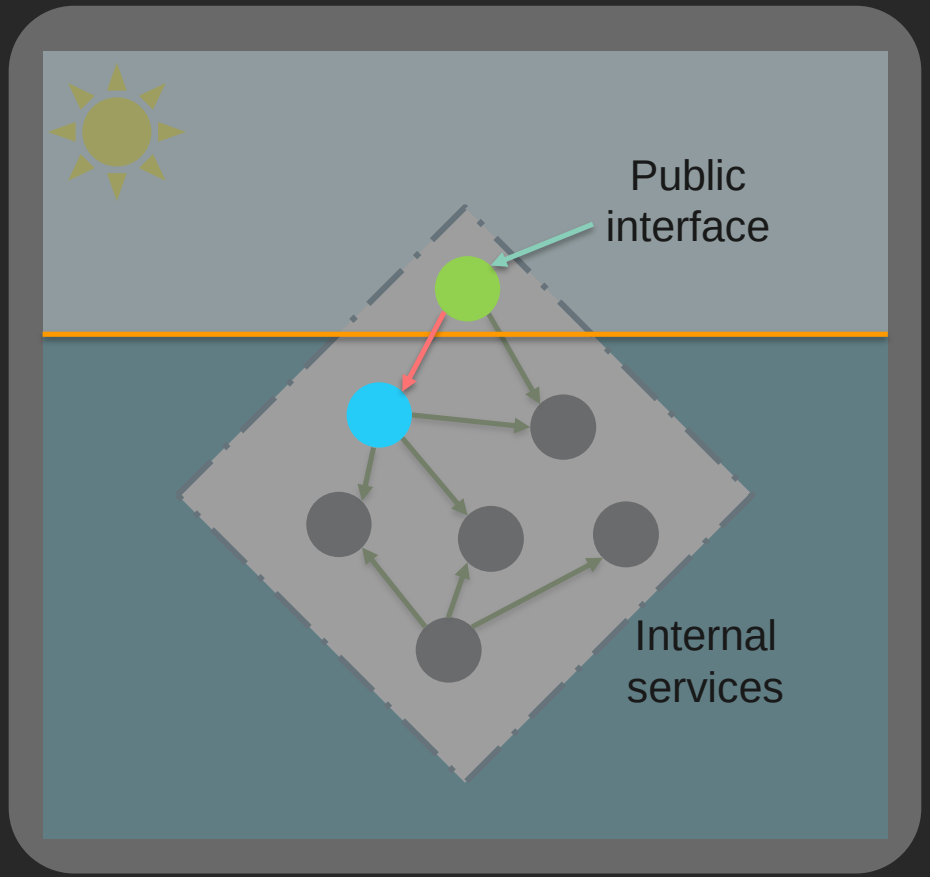
- Synchronous communication
- HTTPS clients
- Relying on API gateways or load balancers
- Flexible client interface
- Securing against client

Async in the back

- Asynchronous communication
- Simple clients
- Queues, topics, buses, streams
- Opinionated event/message production and consumption
- Back off and retry consumption from event source

Today let's focus on a simple application

One public interface →

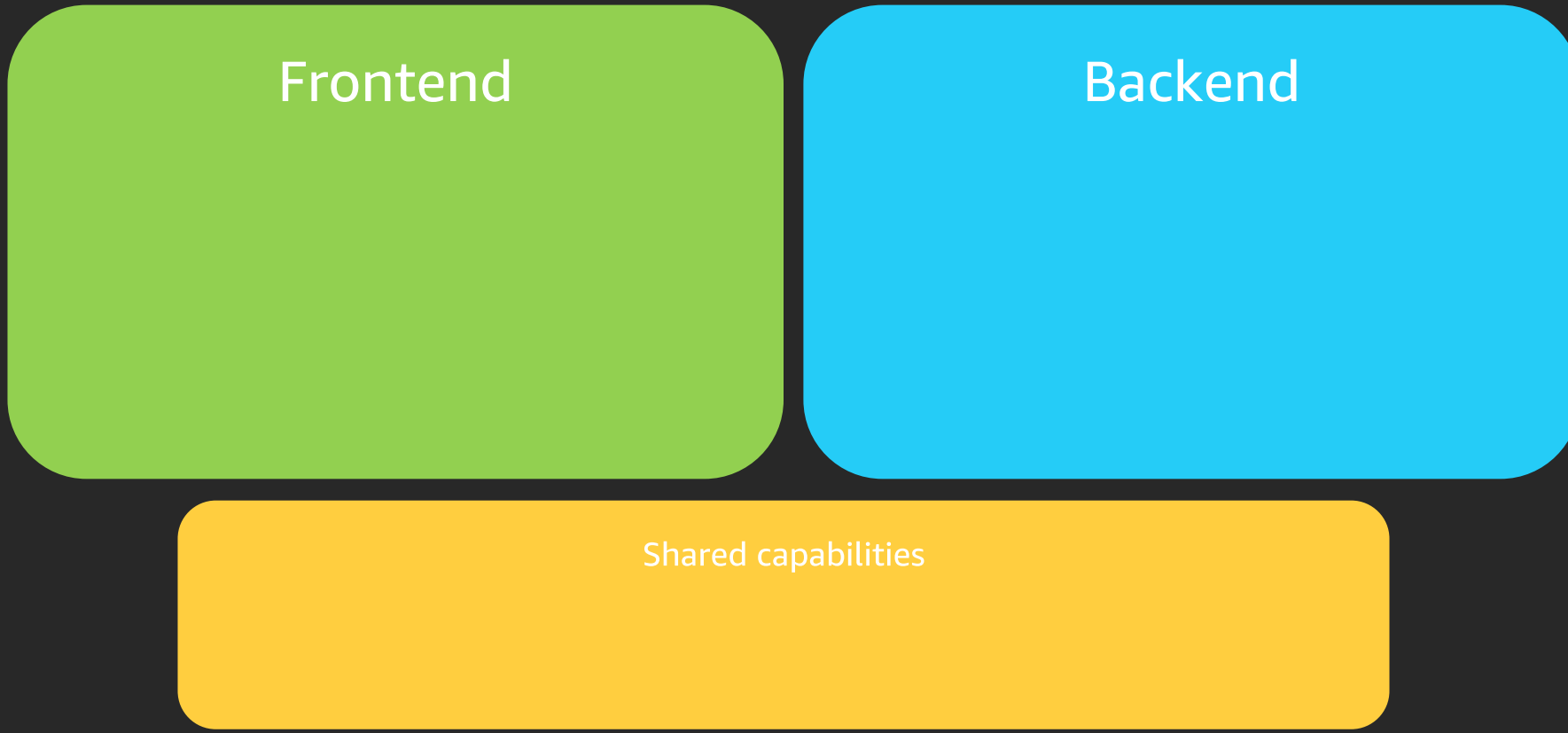


← One private internal backend service

MyService architecture

MyService architecture

MyService architecture



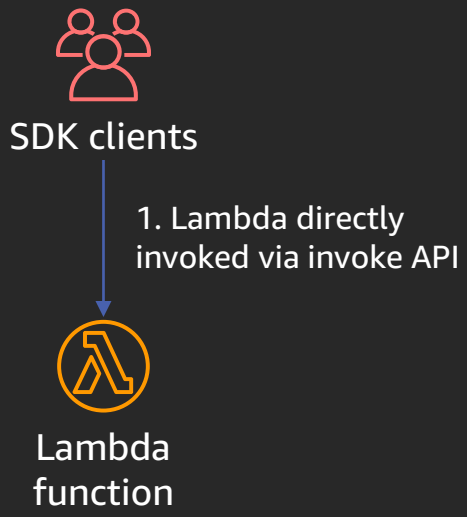
AWS Lambda

AWS Lambda

- Serverless compute service
- Code deployed as functions aligned to an event source
- Unique characteristics:
  - 3gb RAM max, 15 minute duration max, max 250mb application artifact, max 512mb /tmp
- Security, monitoring, logging built in
  - Scoped at the function level
- Ephemeral worker environments
  - Transient in nature, could be ”reaped” due to idle or other lifecycle events
  - No ability to “sticky session” customer data to a worker environment
  - Store all data that matters in a database or data store off function
- Pay per invocation and for execution duration

AWS SAM

Lambda API



API provided by the Lambda service

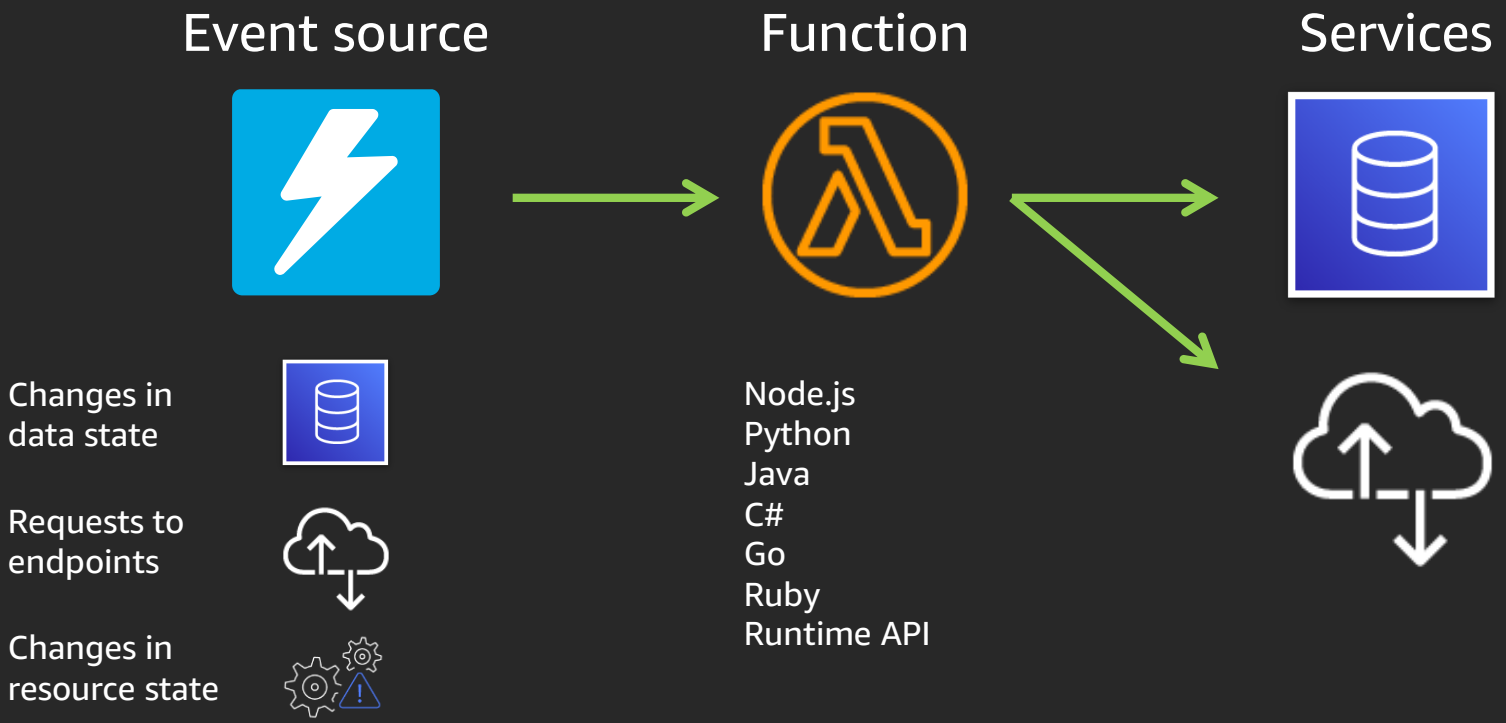
Used by all other services that invoke Lambda across all models

Supports sync and async

Can pass any event payload structure you want

Client included in every SDK

Serverless applications



MyService architecture

Frontend

AWS Lambda

Backend

AWS Lambda

Shared capabilities

MyService architecture

Frontend

AWS Lambda

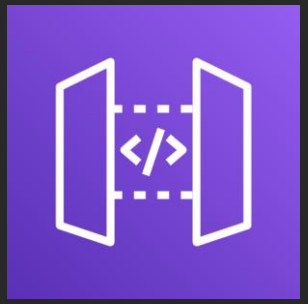
Backend

AWS Lambda

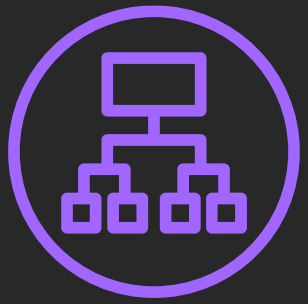
Shared capabilities

Accessing the API

Three options for exposing an API:



Amazon API Gateway

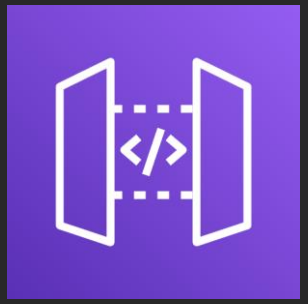


Application Load Balancer (ALB)

AWS AppSync

Accessing the API

Three options for exposing an API:



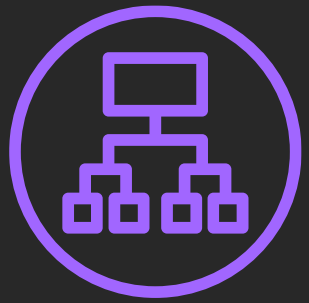
Amazon API Gateway

- REST & WebSocket support
- Flexible auth options
- Throttling/usage tiers
- Caching
- Client SDK generation
- Edge, regional, private endpoint types
- OpenAPI/Swagger support
- Pay per request & data transferred

See SVS212 & SVS402 for more!

Accessing the API

Three options for exposing an API:



Application Load Balancer

- HTTP/S support
- Path-based routing
- URL redirects
- Custom HTTP responses
- Container support
- Pay per hour & LCU consumed

See NET413 for more!

Accessing the API

Three options for exposing an API:

AWS AppSync

- GraphQL support
  - Can combine data from several sources in a single response
- Polyglot to backend data sources
- Deep integration with Amplify Framework
- Subscriptions & offline sync
- Pay per query and data transfer

See MOB307 & MOB402 for more!

Picking an API fronting service cheat sheet

Complex API with multiple data sources or very unique queries against data?

- AWS AppSync

WebSockets?

- Amazon API Gateway

Need transforms, throttling, usage tiers, flexible auth?

- Amazon API Gateway

Normal API, potentially high requests per month, no need for added transform capabilities?

- Application Load Balancer

Else: Normal API, with < tens of millions of requests per month?

- Amazon API Gateway

Warning:
May not be accurate by end of week.

MyService architecture

Frontend

Amazon API Gateway

AWS Lambda

Backend

AWS Lambda

Shared capabilities

Authorization

- Open
  - No authentication or authorization
- AWS Identity and Access Management (IAM) permissions
  - Use IAM policies and AWS credentials to grant access
- Amazon Cognito authorizer
  - Amazon Cognito is a managed user directory for authentication
  - Connect to Amazon Cognito User Pool and together with OAuth scopes to enable authorization
- Lambda authorizers
  - Use Lambda to validate a bearer token (OAuth or SAML as examples) or request parameters and grant access

A quick bit on where to handle request routing

Who should handle the routing to backend logic?

api.myservice.aws/

/users

/products

/support

/search

/burgerandfries

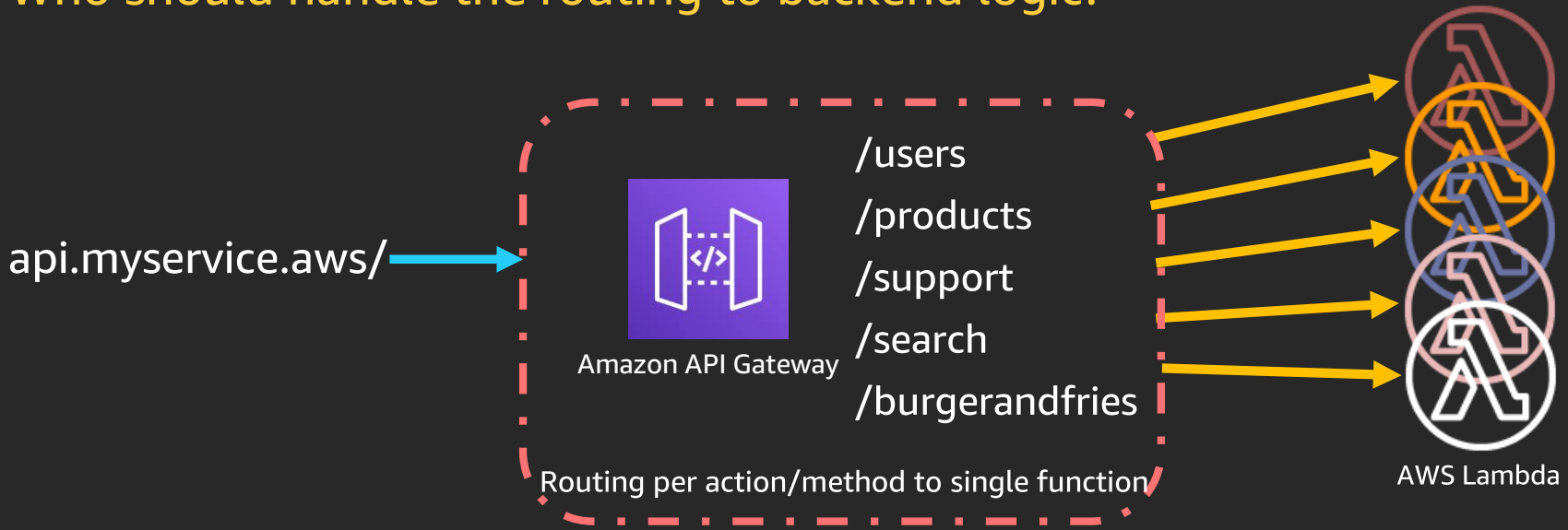
Amazon API Gateway

AWS Lambda

Certain frameworks will put that logic in either place. Does it matter?

A quick bit on where to handle request routing

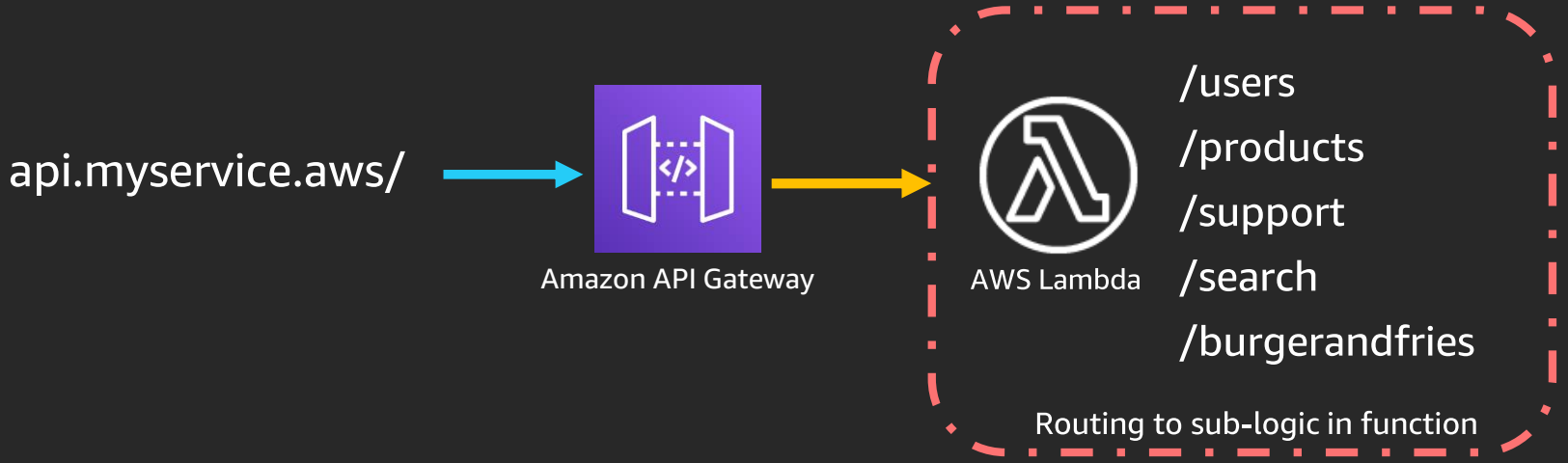
Who should handle the routing to backend logic?



Certain frameworks will put that logic in either place. Does it matter?

A quick bit on where to handle request routing

Who should handle the routing to backend logic?



Certain frameworks will put that logic in either place. Does it matter?

Lambda + in function routing model

The characteristics we mentioned earlier are what put customers into the “danger zone” with the function routing model:

- Security constructs applied to the whole
- Performance settings applied to the whole
- Limited amount of application size
- Limited duration
  - Might be better to chain logic out to other functions with AWS Step Functions or via the async model
- The ”Lambda-lith” can grow too complex, leaving you stuck rethinking the whole logic routing model

AWS Lambda

Where to handle request routing

Team API Gateway

- Using the benefits of API Gateway
- Better compatibility with AWS's tools
- Better security granularity
- Better performance granularity

Team in code

- Native code/framework capabilities
- Potentially better portability of code
- Fewer security constructs (IAM roles, policies, etc.)
- Fewer cold starts? (questionable based on Yan Cui's blog*)

Amazon API Gateway

AWS Lambda

*https://hackernoon.com/aws-lambda-should-you-have-few-monolithic-functions-or-many-single-purposed-functions-8c3872d4338f

Where to handle request routing

Team API Gateway

- Using the benefits of API Gateway
- Better compatibility with AWS's tools
- Better security granularity
- Better performance granularity

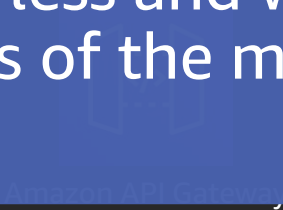
Team in code

- Native code/framework capabilities
- Potentially better portability of code
- Fewer security constructions (IAM Roles, Policies, etc.)
- Better performance (based on Yan Cui's blog*)

Honestly, it's a toss up. Customer feedback seems very split.

If you love your framework, stick with it and its capabilities.

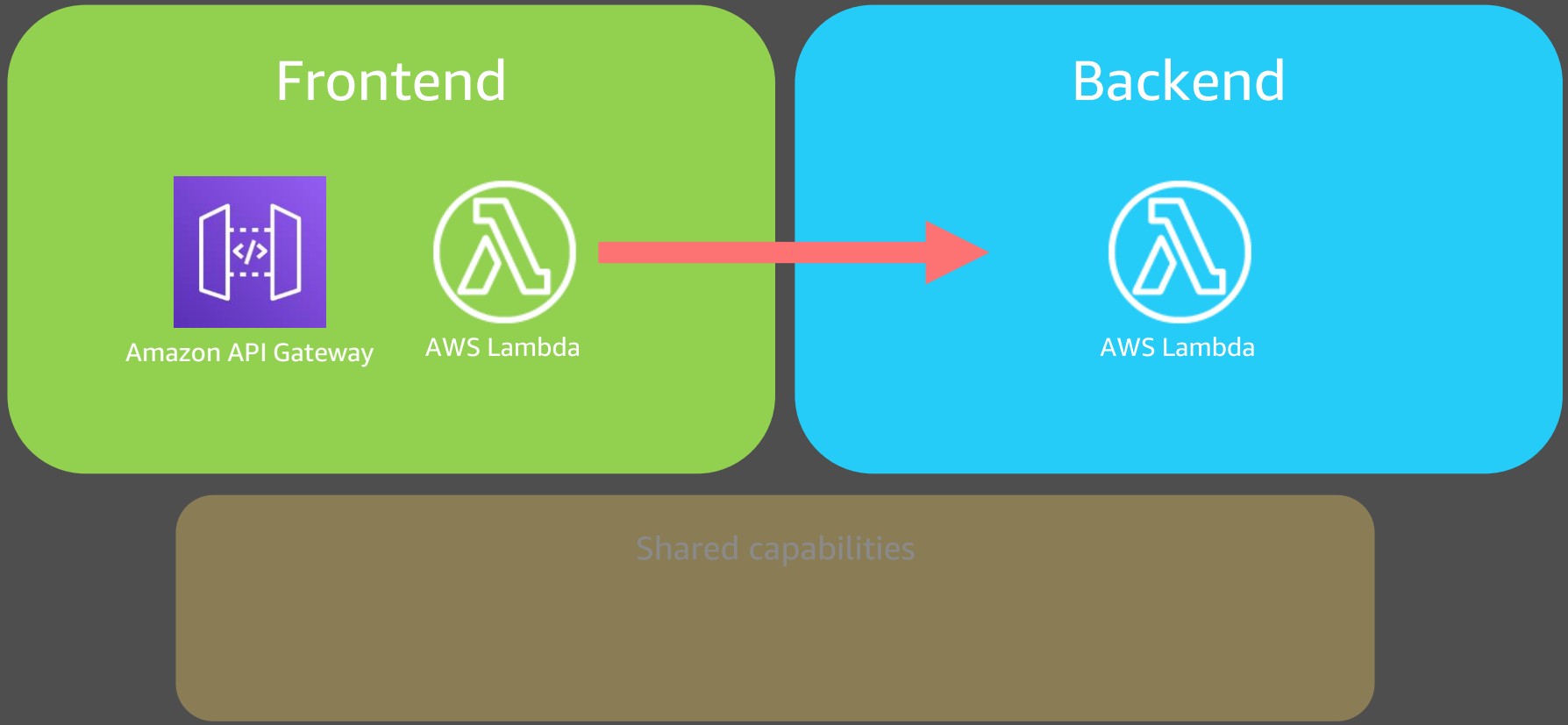
If you want to do less and write less code, go with the benefits of the managed services.



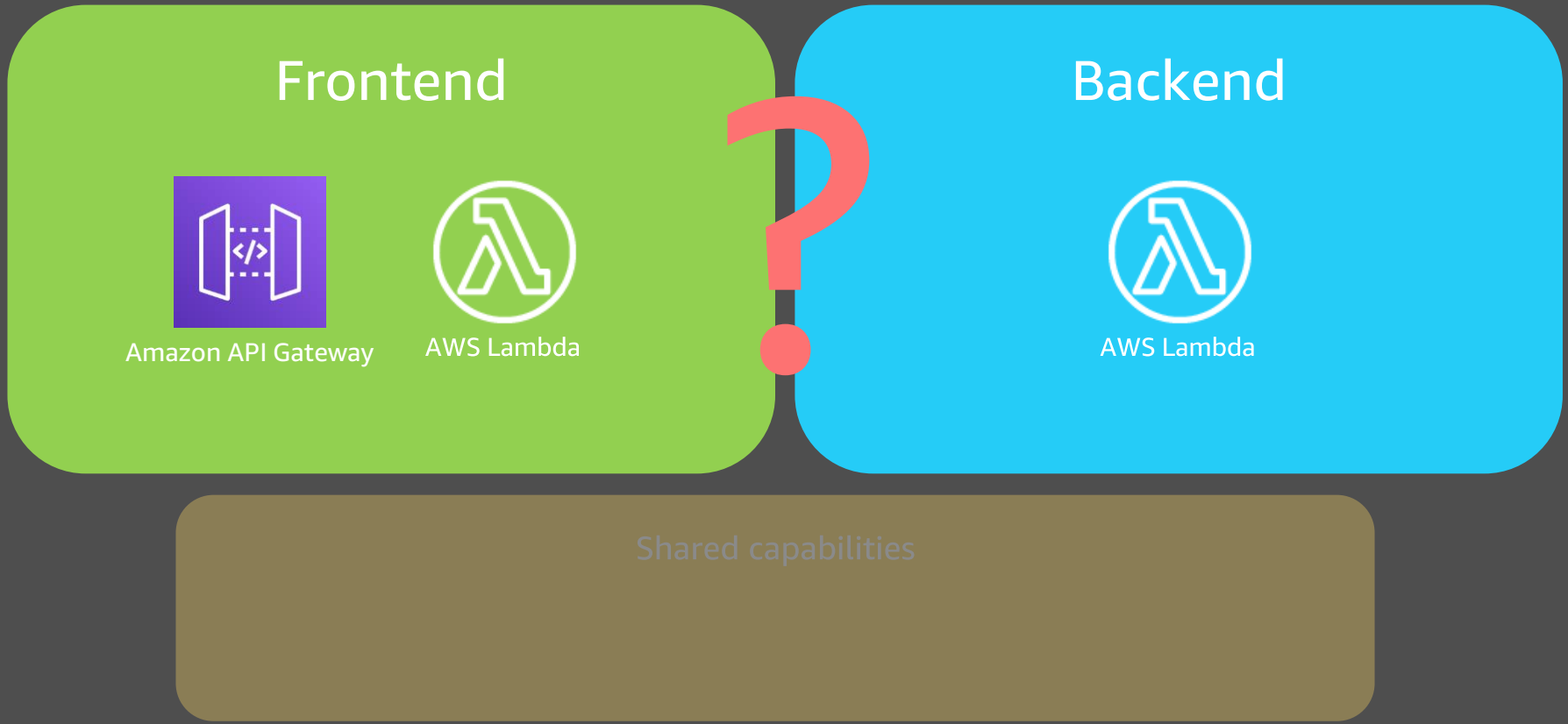
https://hackernoon.com/aws-lambda-should-you-have-few-monolithic-functions-or-many-single-purposed-functions-8c3872d4338f

Topics, streams, queues, and buses! Oh my!

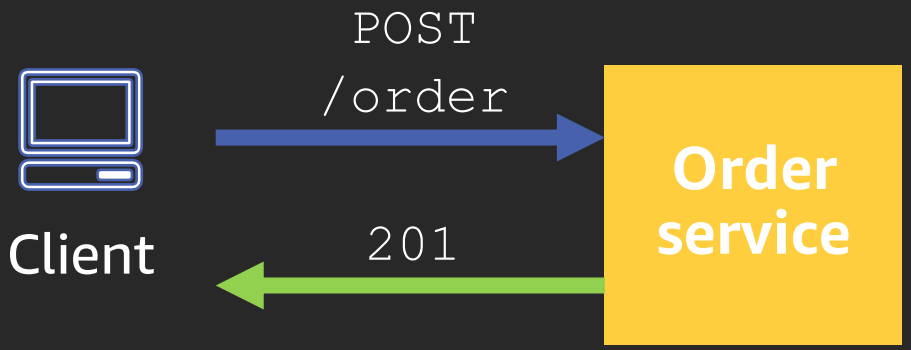
MyService architecture



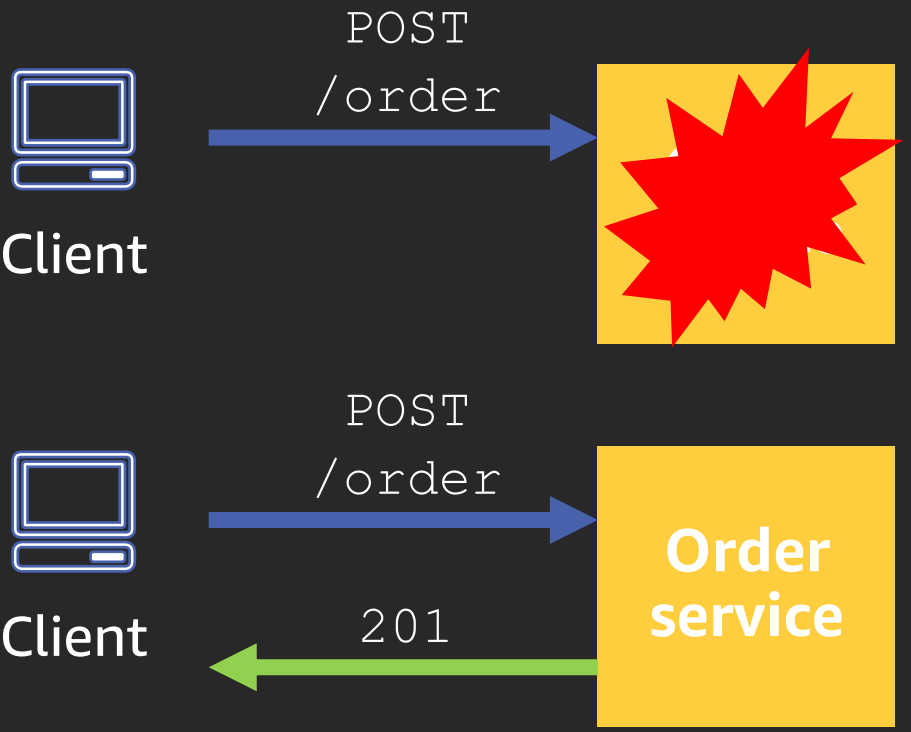
MyService architecture



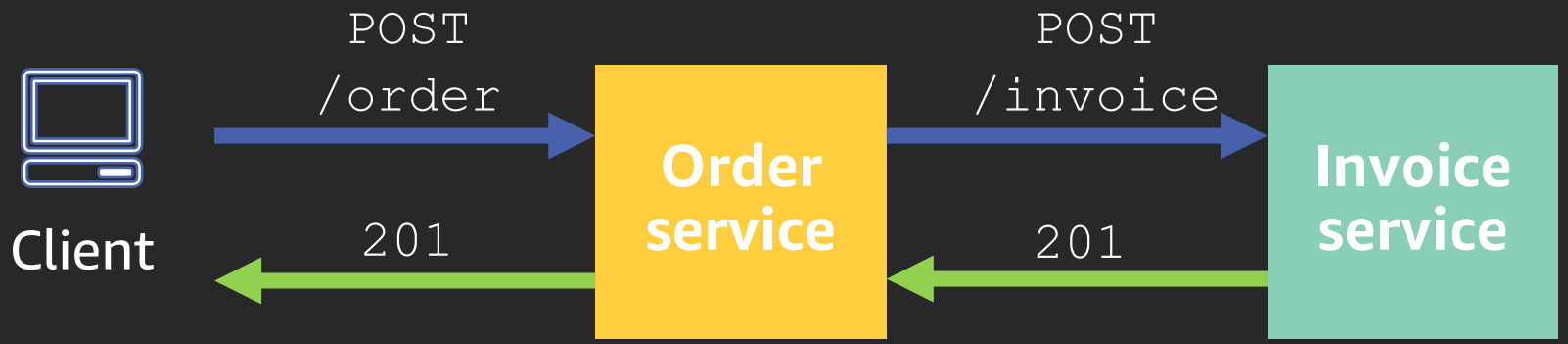
Synchronous APIs



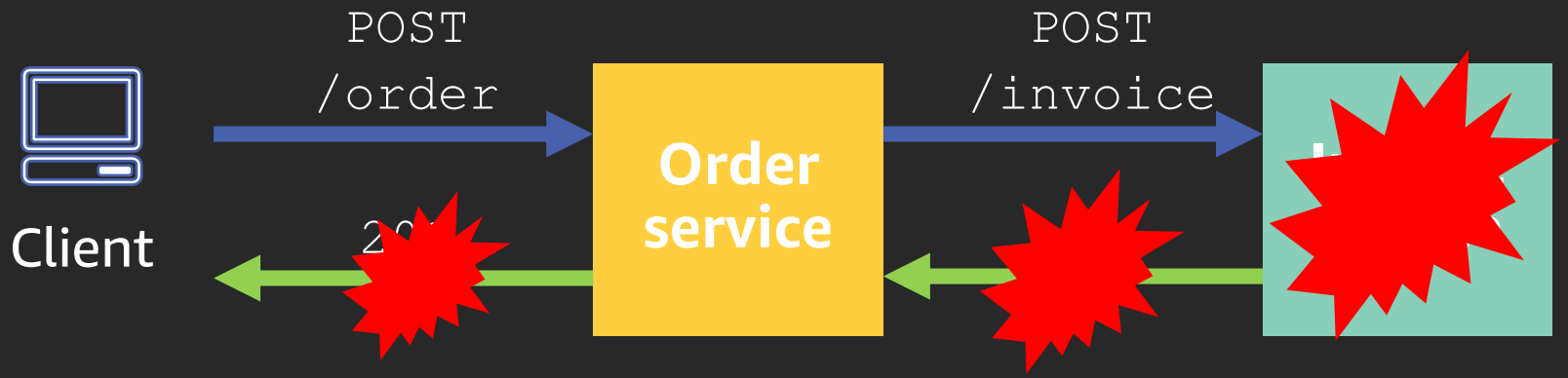
Synchronous APIs



Synchronous APIs



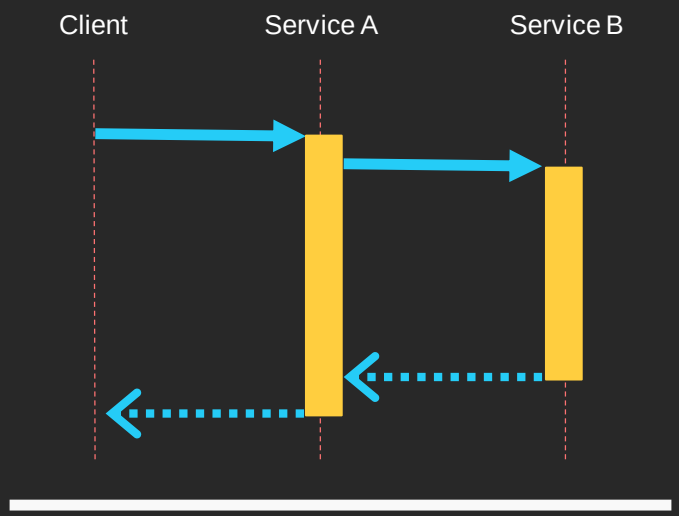
Synchronous APIs



Synchronous APIs

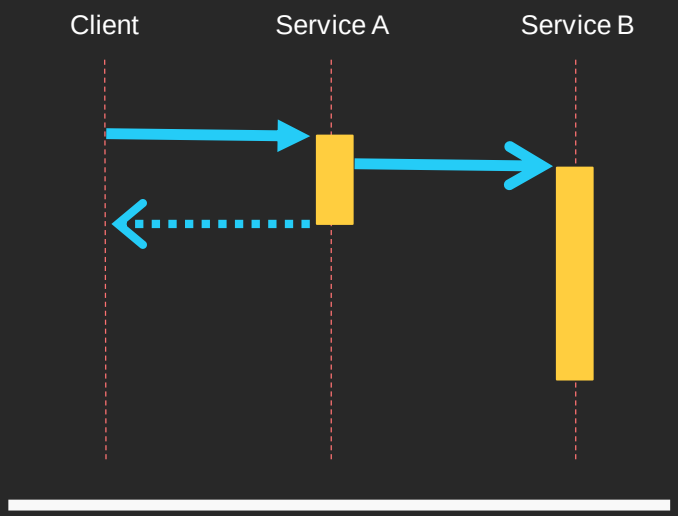
This effectively creates a "tight coupling"
where failures become harder to recover from

Thinking asynchronously



Synchronous

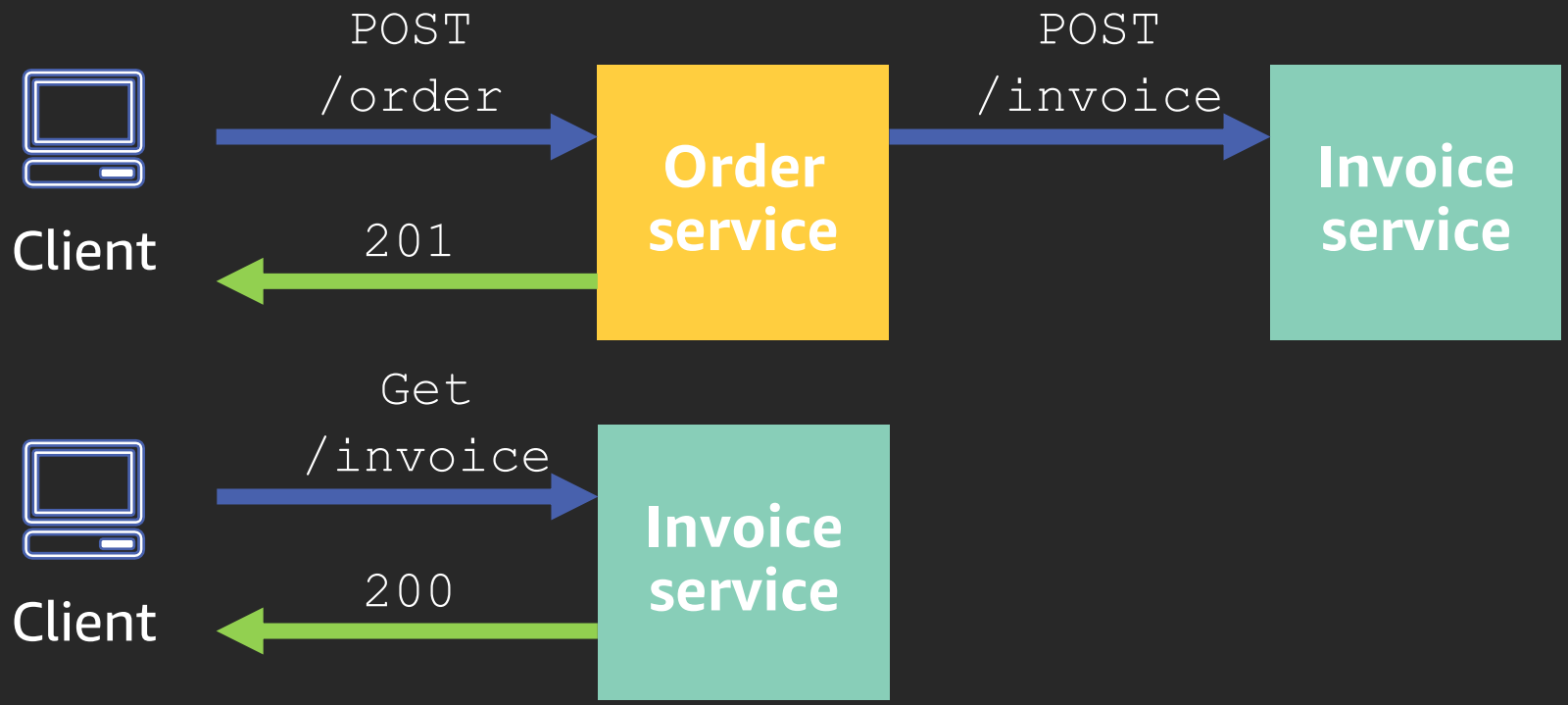
commands



Asynchronous

events

Asynchronous APIs



"The time spent to try making a process async will pay for itself in you gaining deeper understanding of what is really happening with your data."

-Me

Right Now, 2019

Topics, streams, queues, and buses! Oh my!

Amazon Simple Notification Service (Amazon SNS)

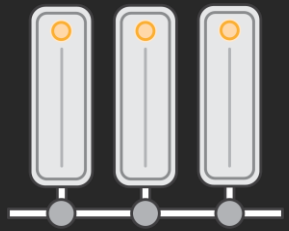
Amazon Simple Queue Service (Amazon SQS)

Amazon EventBridge

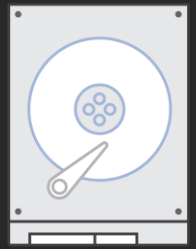
Amazon Kinesis Data Streams

Ways to compare

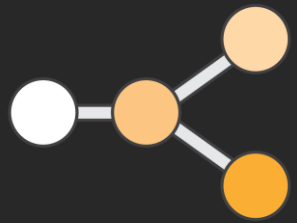
Scale/concurrency controls



Durability



Persistence



Consumption models

Retries

Pricing

Recent announcements for async event sources

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

ICYMI: Serverless pre:Invent 2019

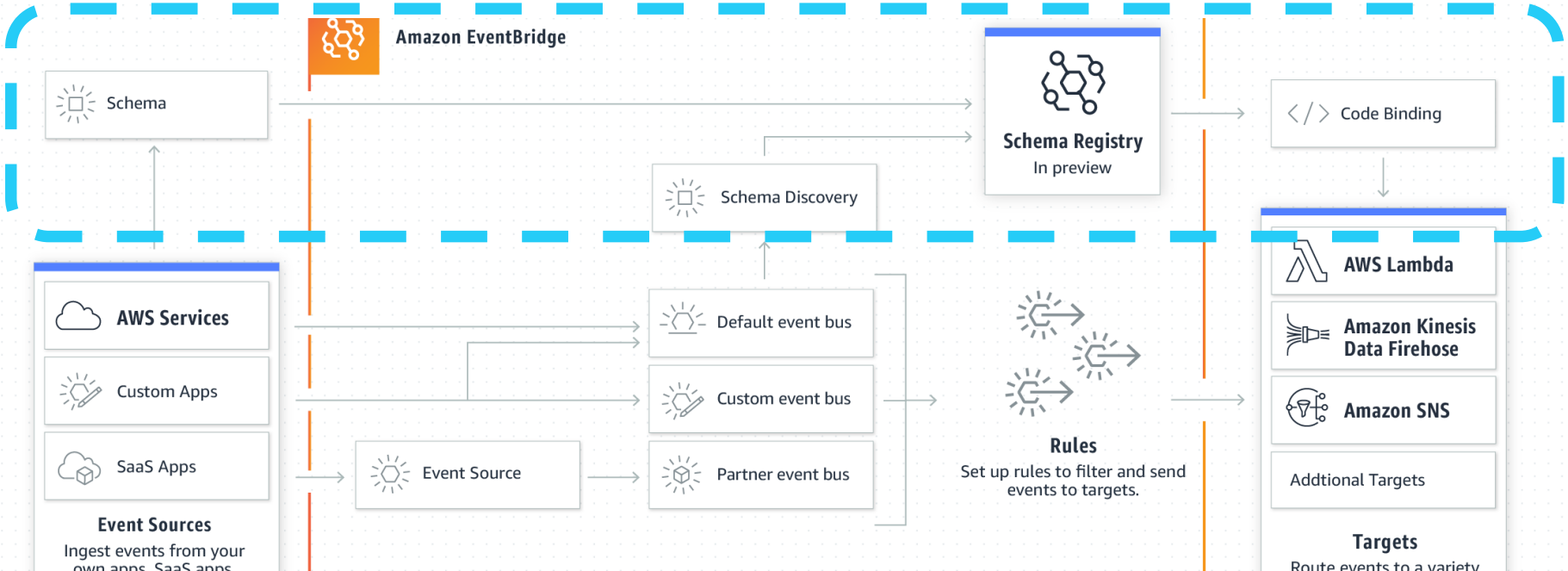
Last night!

...nouncements for async event sources

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



“Woah”

-Me

Right Now, 2019

Async service decider cheat sheet

Massive throughput/ordering/multiple consumers/replay?

- Kinesis Data Streams

One to mostly one or minimal fanout, direct to Lambda/HTTP target?

- Amazon SNS

Buffer requests until they can be consumed, whether ordered or not?

- Amazon SQS

One to many fanout, lots of different consumer targets, schema matching, granular target rules?

- EventBridge

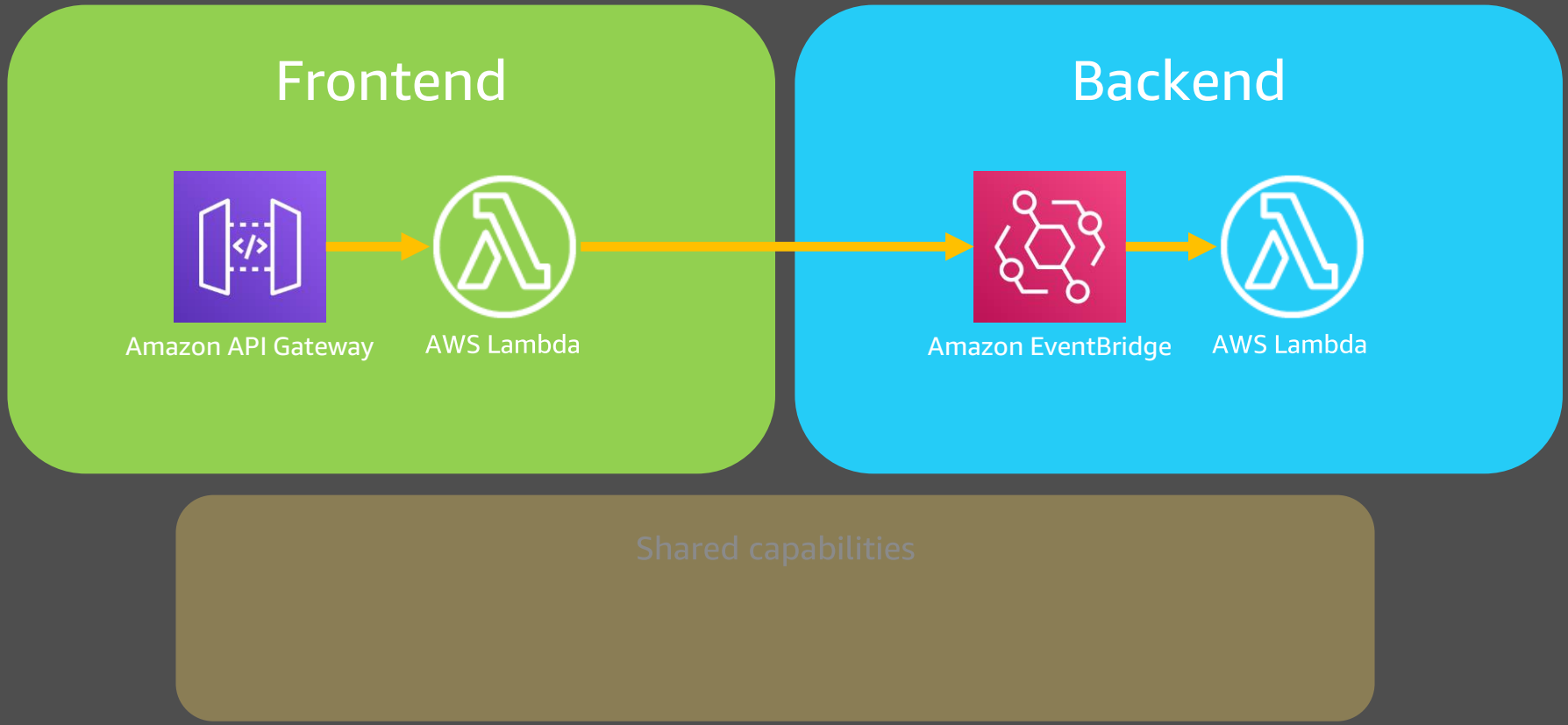
Async service decider cheat sheet

This is a big topic itself with a lot of things to consider. Evaluate along the 6 criteria mentioned previously.

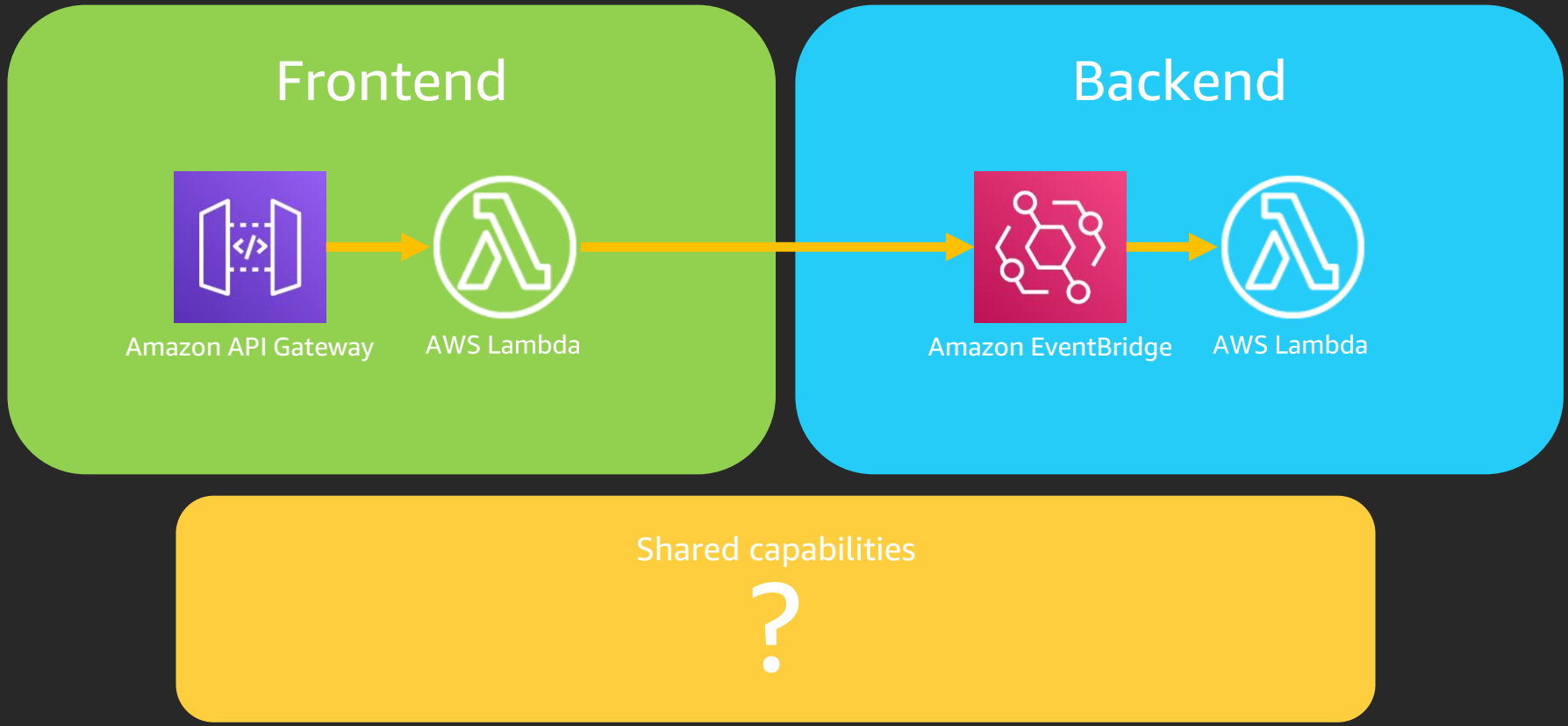
EventBridge changes so much of this because it can target to all of these as well. ::mind blown gif::

See API304, API315, API320, SVS308, SVS317 for more!

MyService architecture



MyService architecture



Shared capabilities

There are still a lot of things to think about

- Secrets/configuration management
- Simplifying code management
- Debugging/troubleshooting
- Performance controls
- Security

AWS Serverless Application Model (AWS SAM)

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
- Integrated 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
```

```
Import sdk
Import http-lib
Import [illegible]

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

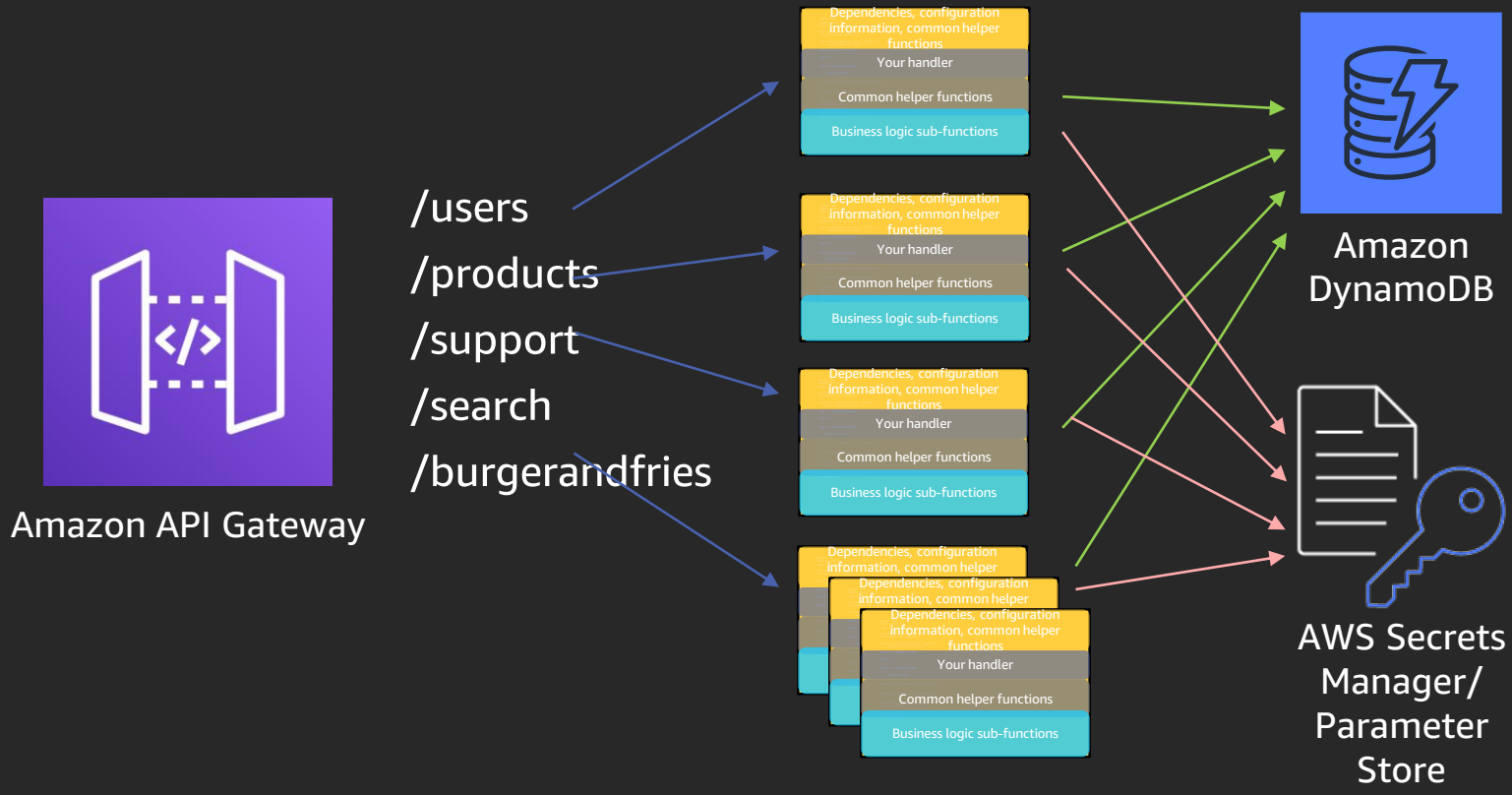
```
Function Pre-handler-secret-getter() {
}

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

Common helper functions

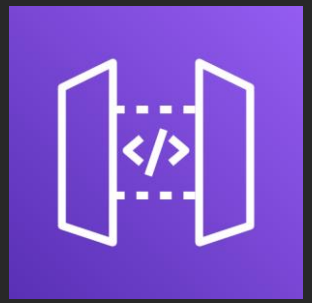
```
Function subFunctionA(thing){
 ## logic here
}

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

Business logic sub-functions

Anatomy of a serverless application



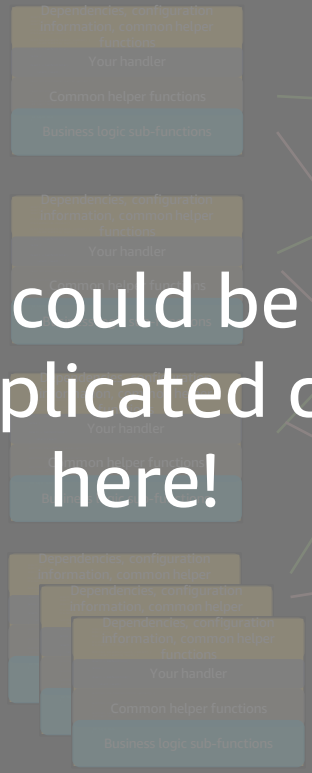
Anatomy of a serverless application



Amazon API Gateway

/users
/products
/support
/search
/burgerandfries

There could be a lot of duplicated code here!



Amazon DynamoDB

AWS Secrets Manager/ Parameter Store

Lambda Layers

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

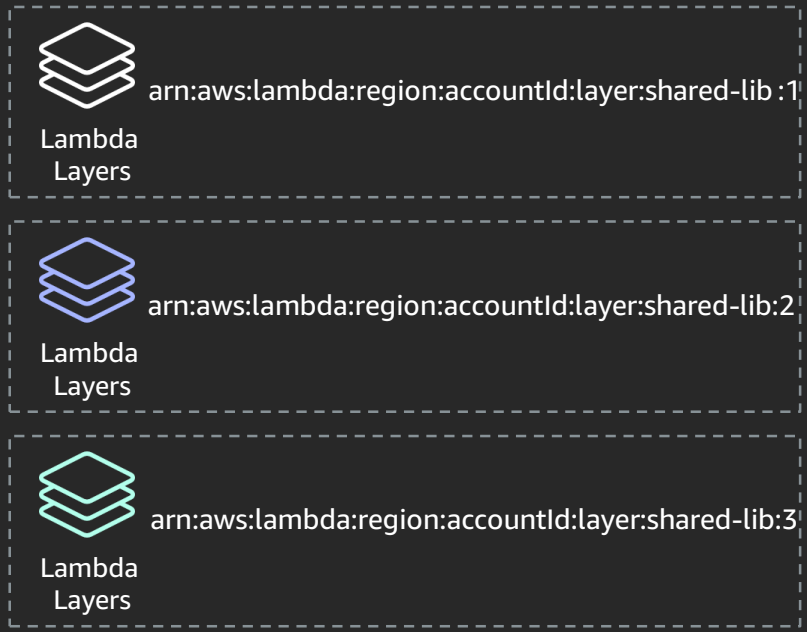
Layer can be anything: dependencies, training data, configuration files, etc

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

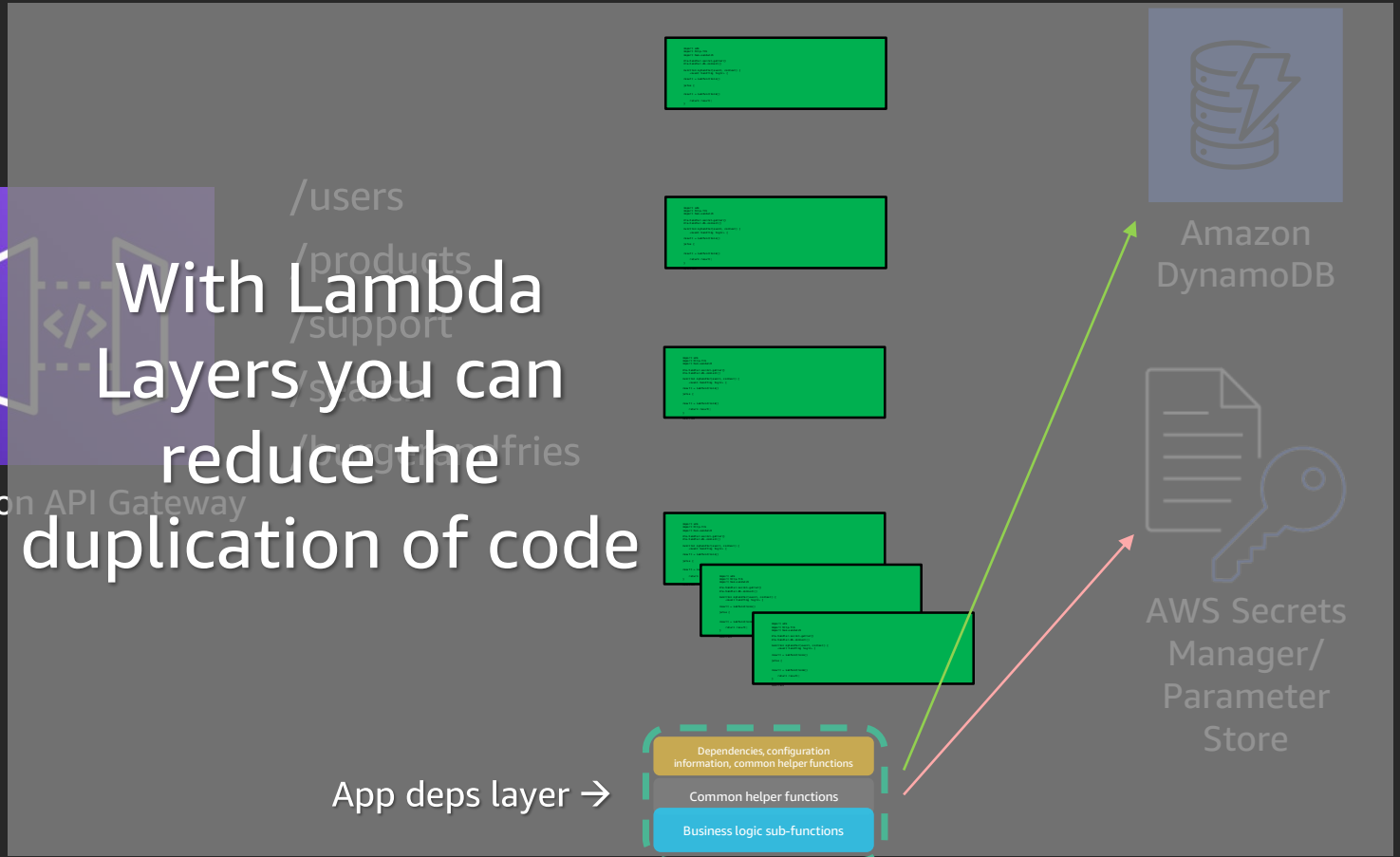
Built-in support for secure sharing by ecosystem

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



Anatomy of a serverless application

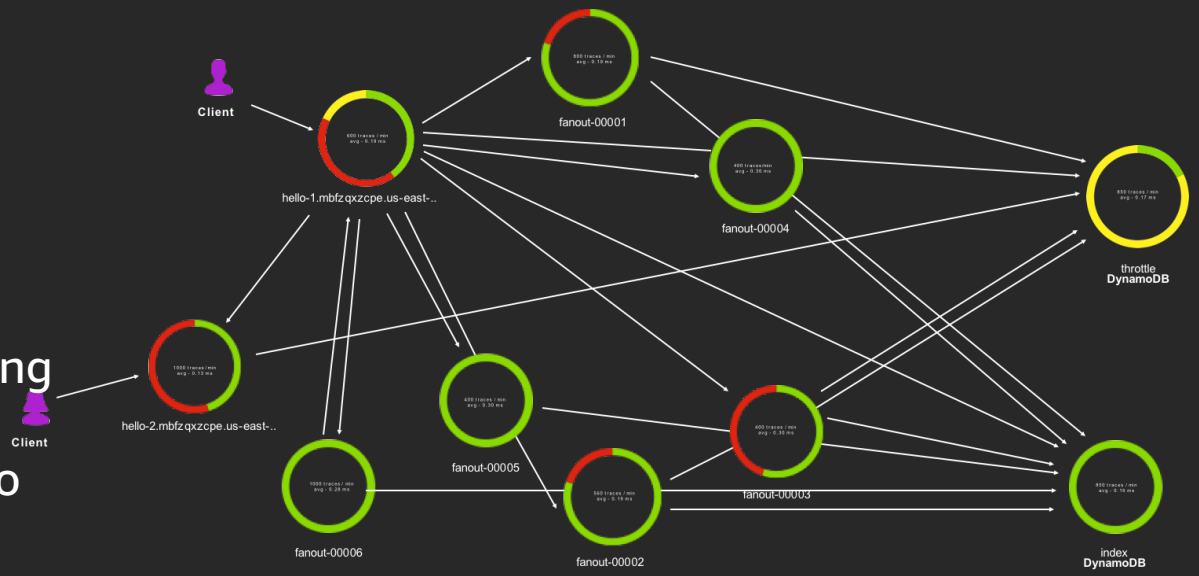


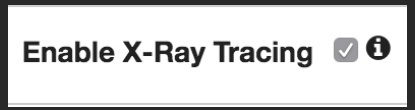
AWS X-Ray

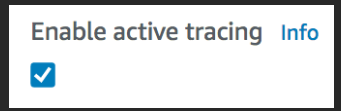
Profile and troubleshoot serverless applications:

- 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

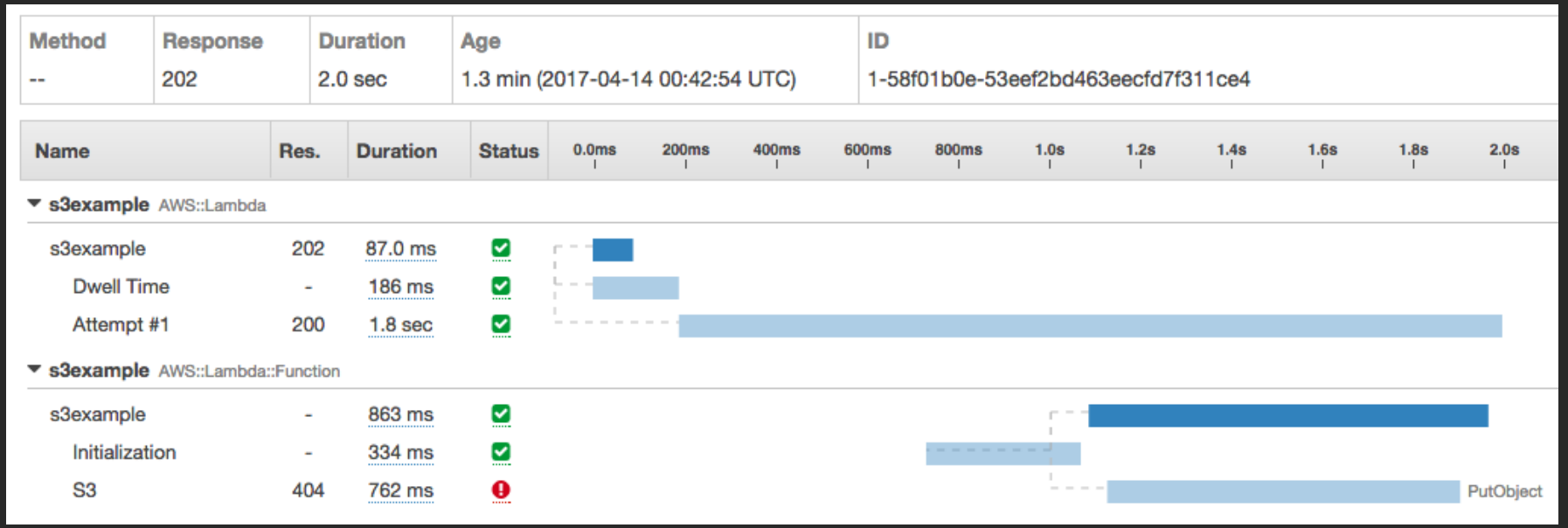
```
var AWSXRay = require('aws-xray-sdk-core');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));
S3Client = AWS.S3();
```







X-Ray trace example



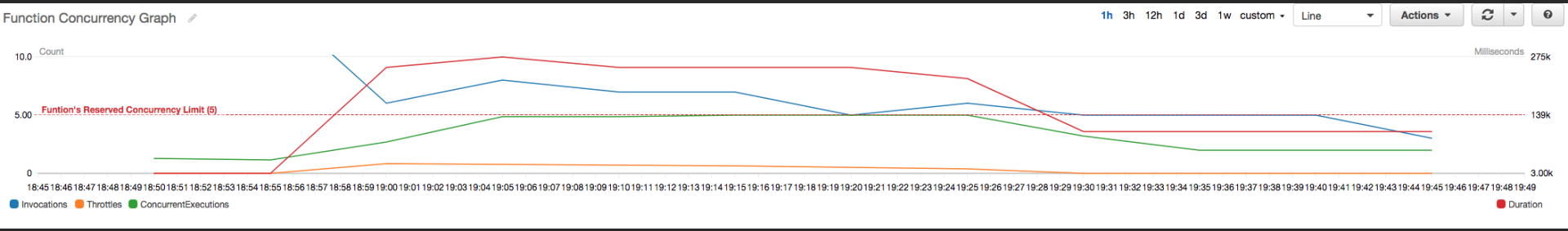
Recent announcements for X-Ray & Amazon CloudWatch

- CloudWatch ServiceLens
  - A “single pane of glass” monitoring tool
- CloudWatch Synthetics
  - Generate “canary” tests for your APIs and other services
- Embedded Metric Format for CloudWatch Logs
- CloudWatch Contributor Insights
  - Identify top talkers, common patterns in logs
- X-Ray Trace Maps
  - Map a single event though your distributed application
- X-Ray integration with CloudWatch Synthetics

ICYMI: Serverless pre:Invent 2019

Lambda per function concurrency controls

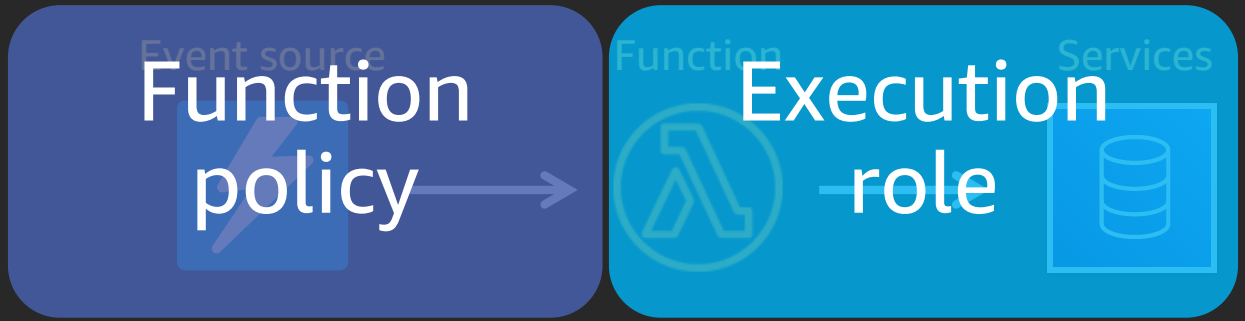
- Concurrency is a shared pool by default
- Separate using per function concurrency settings
    - Acts as reservation
- Also acts as max concurrency per function
    - Especially critical for downstream resources like databases
- “Kill switch” – set per function concurrency to zero



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



AWS Serverless Application Model (AWS SAM)

AWS CloudFormation extension optimized for serverless

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

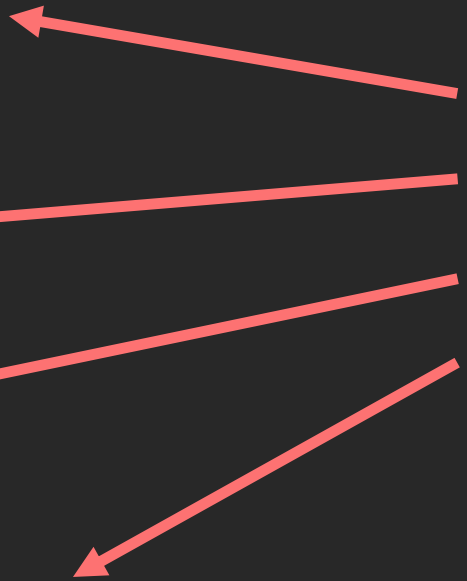
Supports anything AWS CloudFormation supports

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

AWS SAM Command Line Interface (AWS SAM CLI)

CLI tool for local development, debugging, testing, deploying, and monitoring of serverless applications

Supports API Gateway “proxy-style” and Lambda service API testing

Response object and function logs available on your local machine

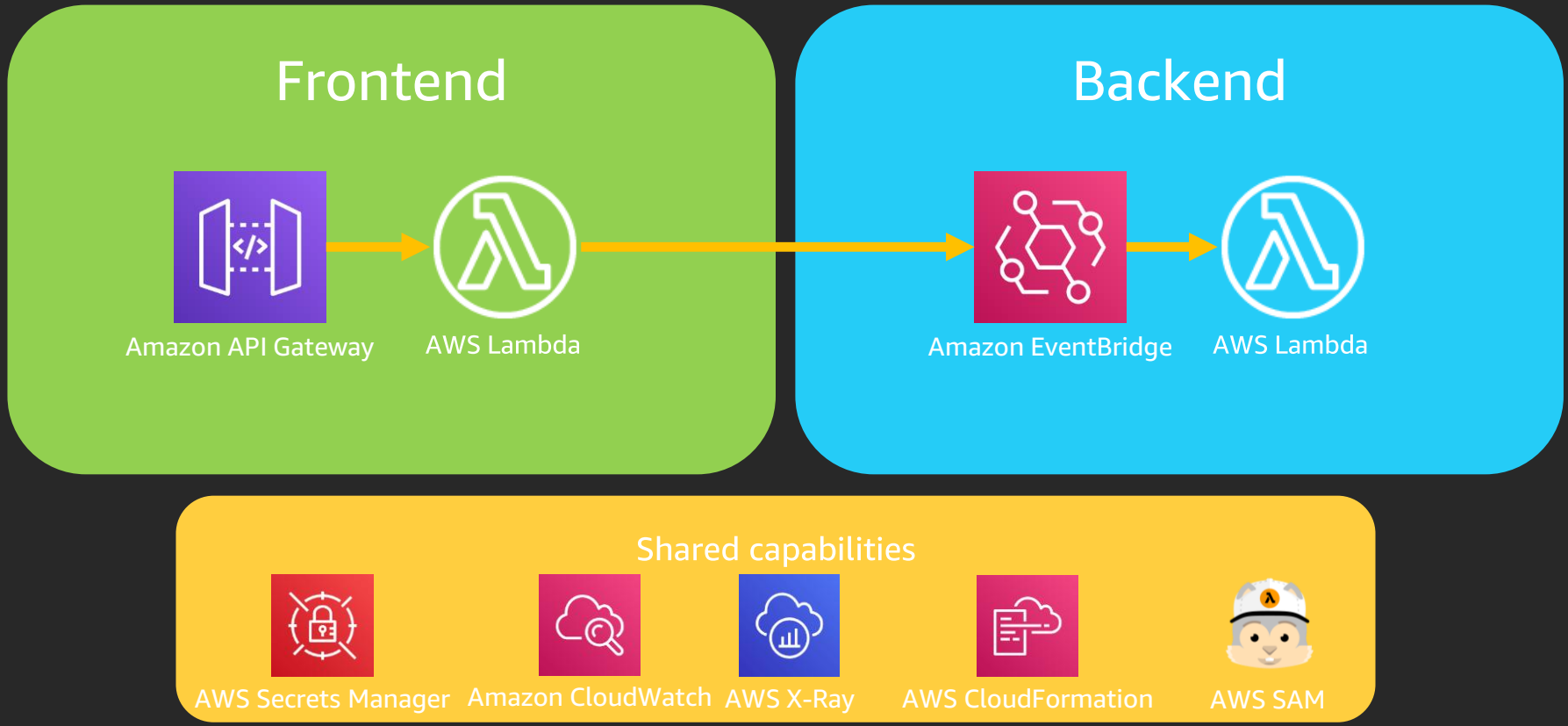
Uses open source docker-lambda images to mimic Lambda’s execution environment such as timeout, memory limits, runtimes

Can tail production logs from CloudWatch Logs

Can help you build in native dependencies

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

MyService architecture



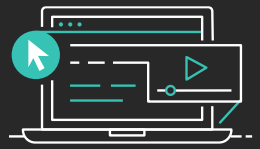
FIN/ACK - Building microservices with AWS Lambda

The characteristics of Lambda make you think differently about how to build microservices:

- Lambda's service limits impact your architectural decisions
  - Store state elsewhere
- 3 ways to do APIs: Amazon API Gateway, ALB, AWS AppSync
- 4 stream/async methods: Amazon SNS, Amazon SQS, EventBridge, Kinesis Data Streams
- Ecosystem of tools simplify a lot of things:
  - X-Ray
  - CloudWatch metrics & logs
  - AWS SAM + AWS CloudFormation
  - Secrets Manager + Parameter Store

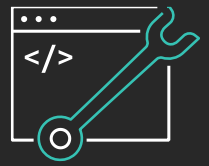
Learn Serverless with AWS Training and Certification

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

aws training and certification

Thank you!

Chris Munns

munns@amazon.com
@chrismunns

Please complete the session survey in the mobile app.