Track 4| Session 1

如何活用事件驅動架構快速擴展應用

Kim Kao (高翊凱)
Solutions Architect
Amazon Web Services

aws SUMMIT ONLINE

Agenda

From APIs to events

Event-driven choreography with Amazon EventBridge

Customer case study

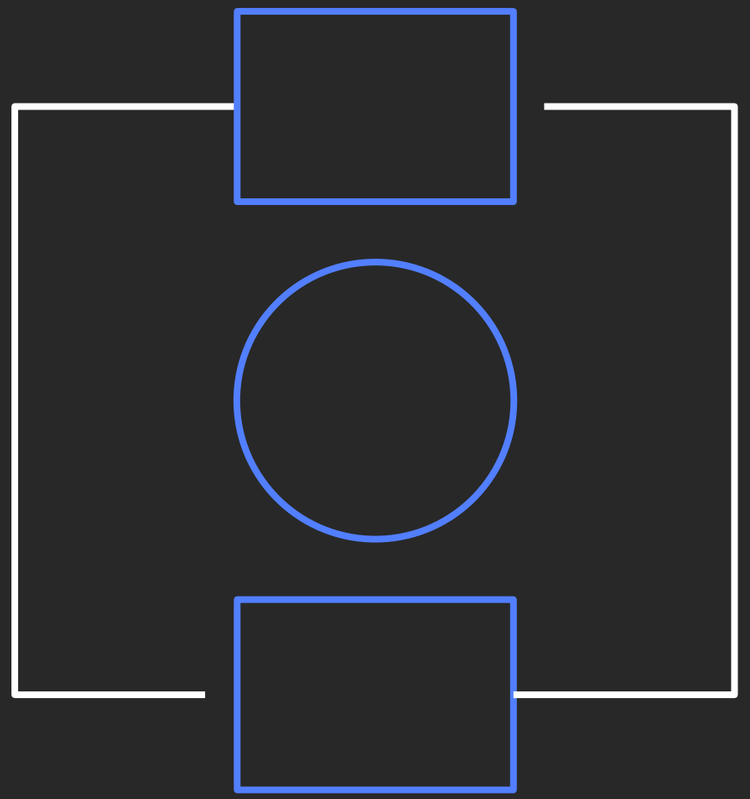
Orchestration with AWS Step Functions

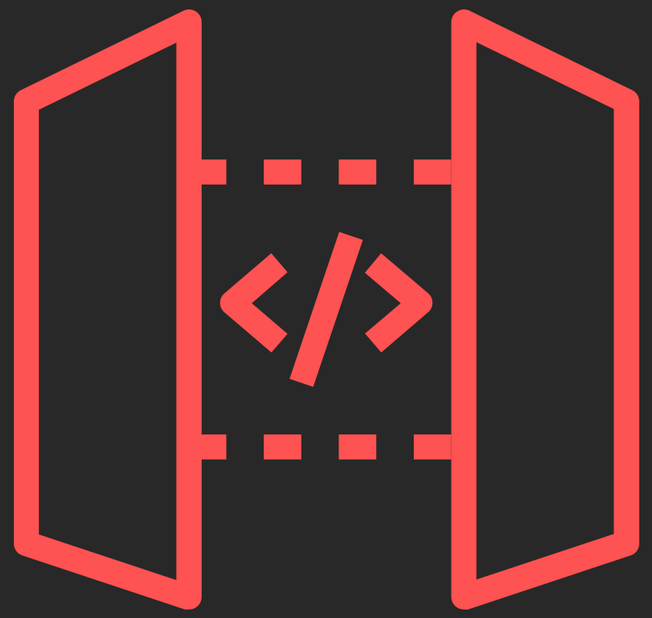
Tracing and observability

From APIs to events

Small pieces loosely joined

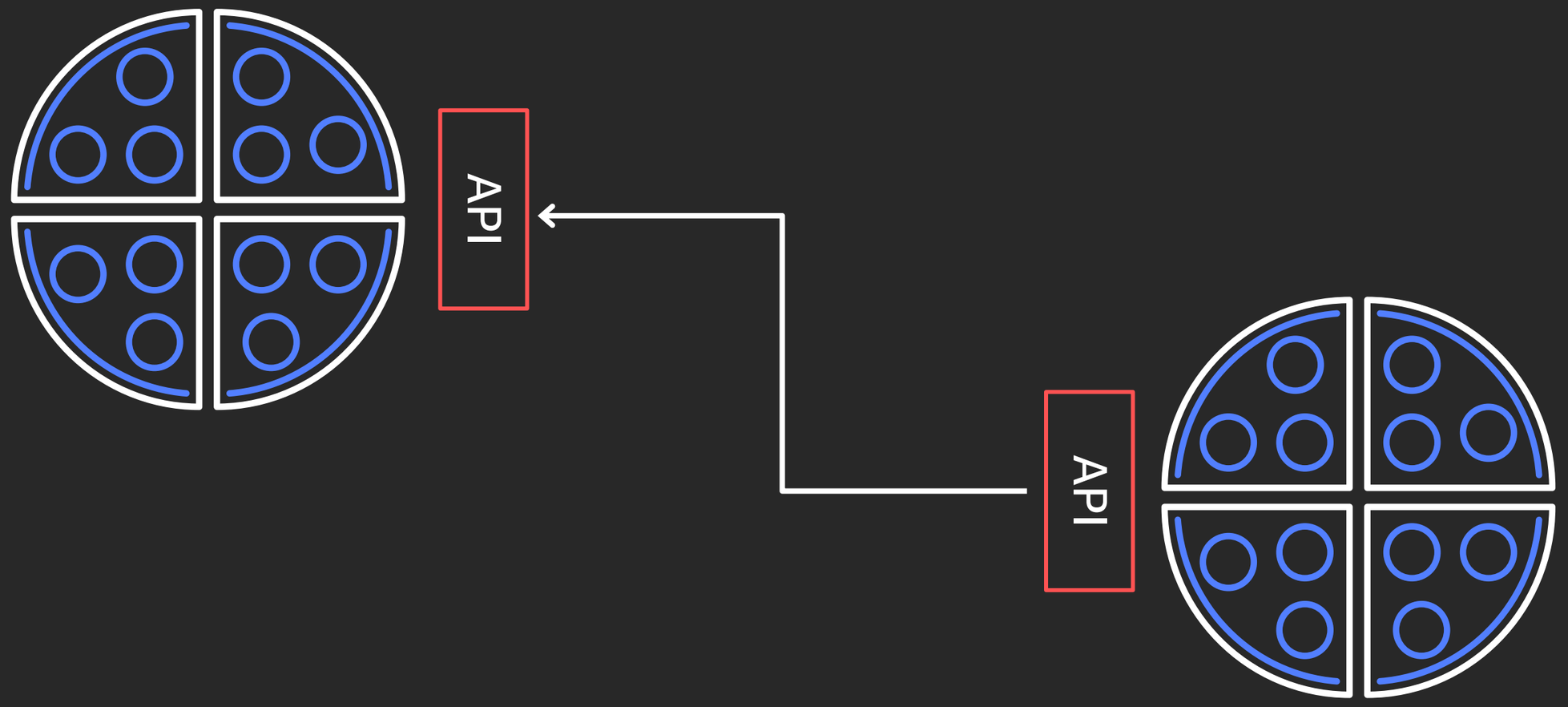
Cloud-native architectures are small pieces, loosely joined



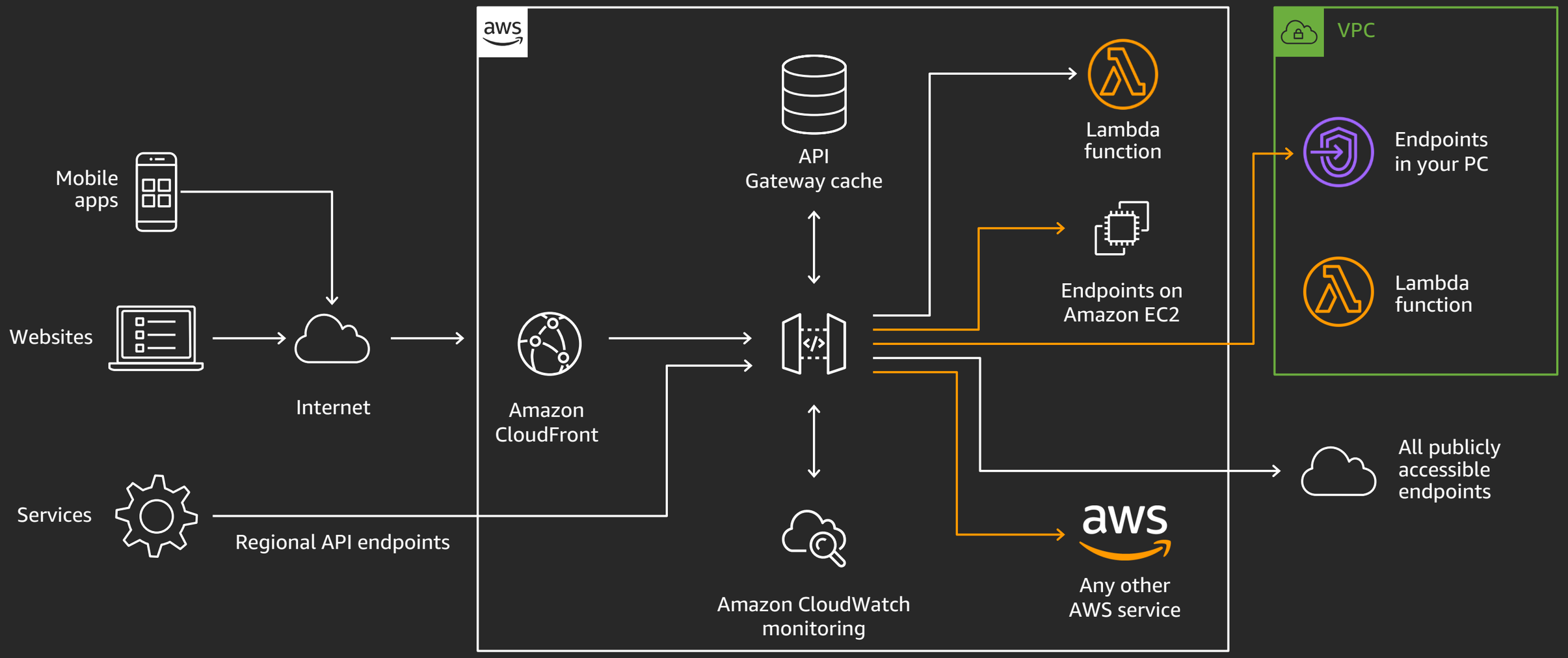


APIs are the front door
of microservices

APIs are hardened contracts



Manage APIs with Amazon API Gateway

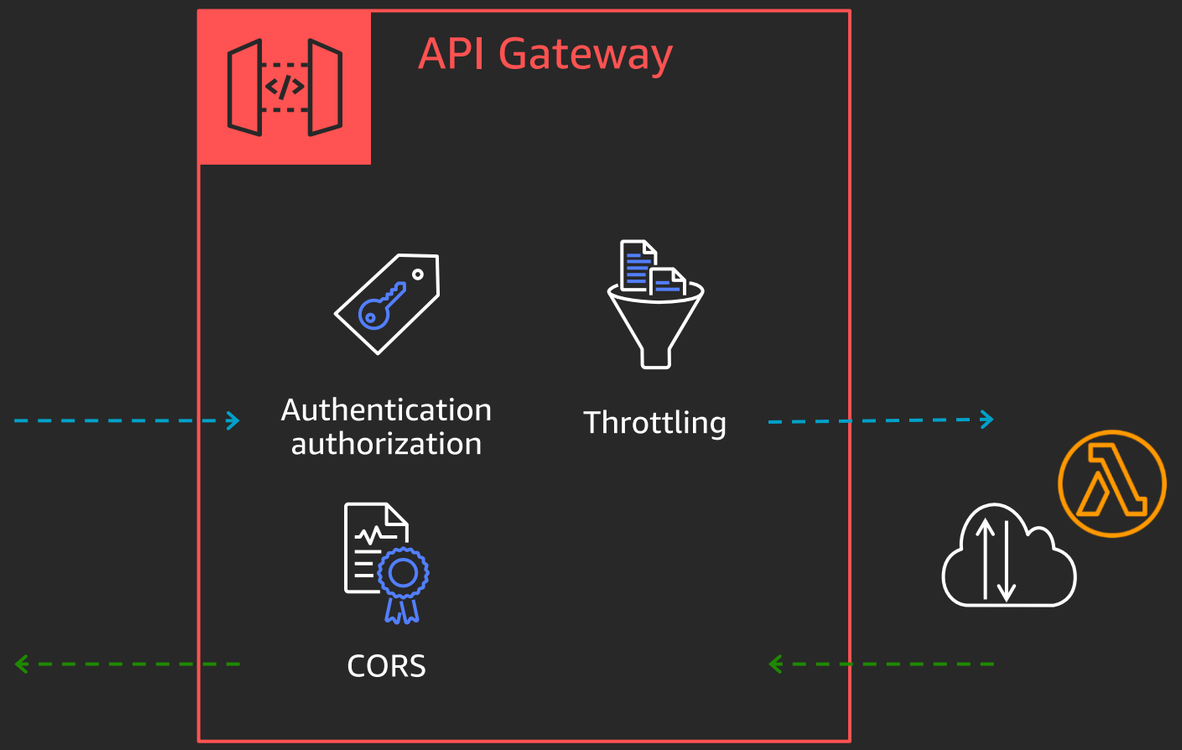


HTTP APIs

Up to 70% lower cost

50% lower latency

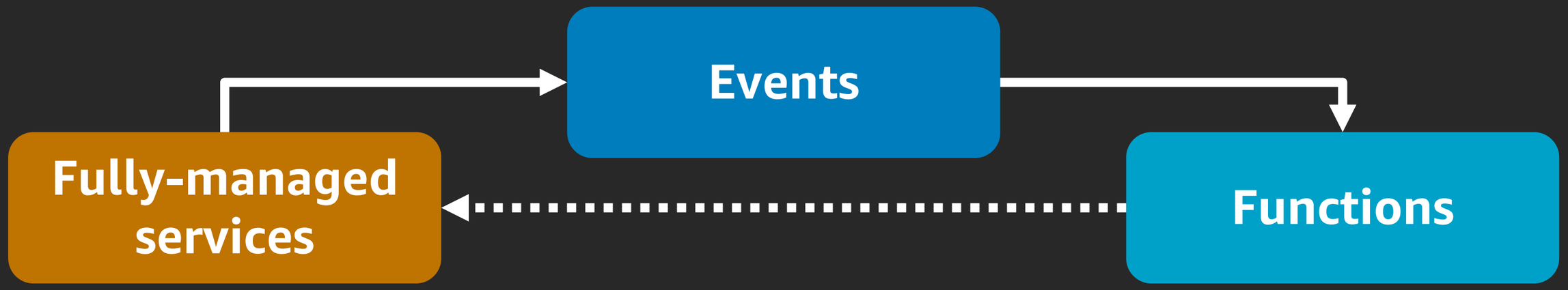
Standard authentication



Serverless application

… comprised of decoupled services that scale independently and trigger execution using events

How do Serverless applications work?



API Gateway
Storage
Databases
Analytics
. . .

API call
User uploads a picture
Customer data updated
Anomaly detected
. . .

Your unique business logic

Concise function logic

- Use functions to **transform**, not **transport**
  - Use purposefully built services for communication fan-out, message handling, data replication, and 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 & Amazon Aurora
  - Use Amazon S3 Select
  - Properly indexed databases
- Leverage Synchronous Vs Asynchronous invocations

Event-driven choreography with Amazon EventBridge

aws SUMMIT ONLINE

What is an “event” ?

“something that happens”

Events tell us a fact

Immutable time-series

Time	What
2019 06 21 08 07 06	CustomerCreated
2019 06 21 08 07 09	OrderCreated
2019 06 21 08 07 13	PaymentSuccessful
2019 06 21 08 07 17	CustomerUpdated
. . .	. . .

Commands Vs Events

Command	Event
Has an intent Directed to a target Personal communication	It's a fact For others to observe Broadcast one to many
"CreateAccount" "AddProduct"	"AccountCreated" "ProductAdded"

Introducing AWS Lambda Event Destinations

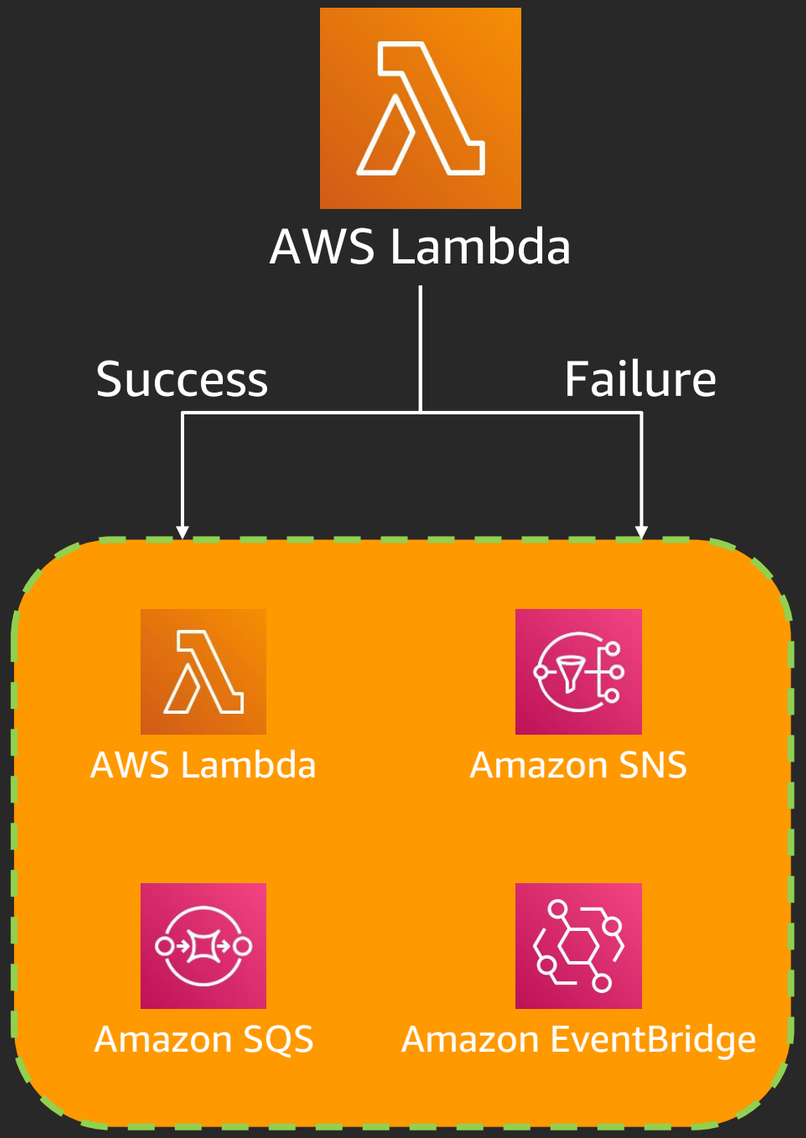
NEW!!!

For asynchronous invocations, capture success or failure

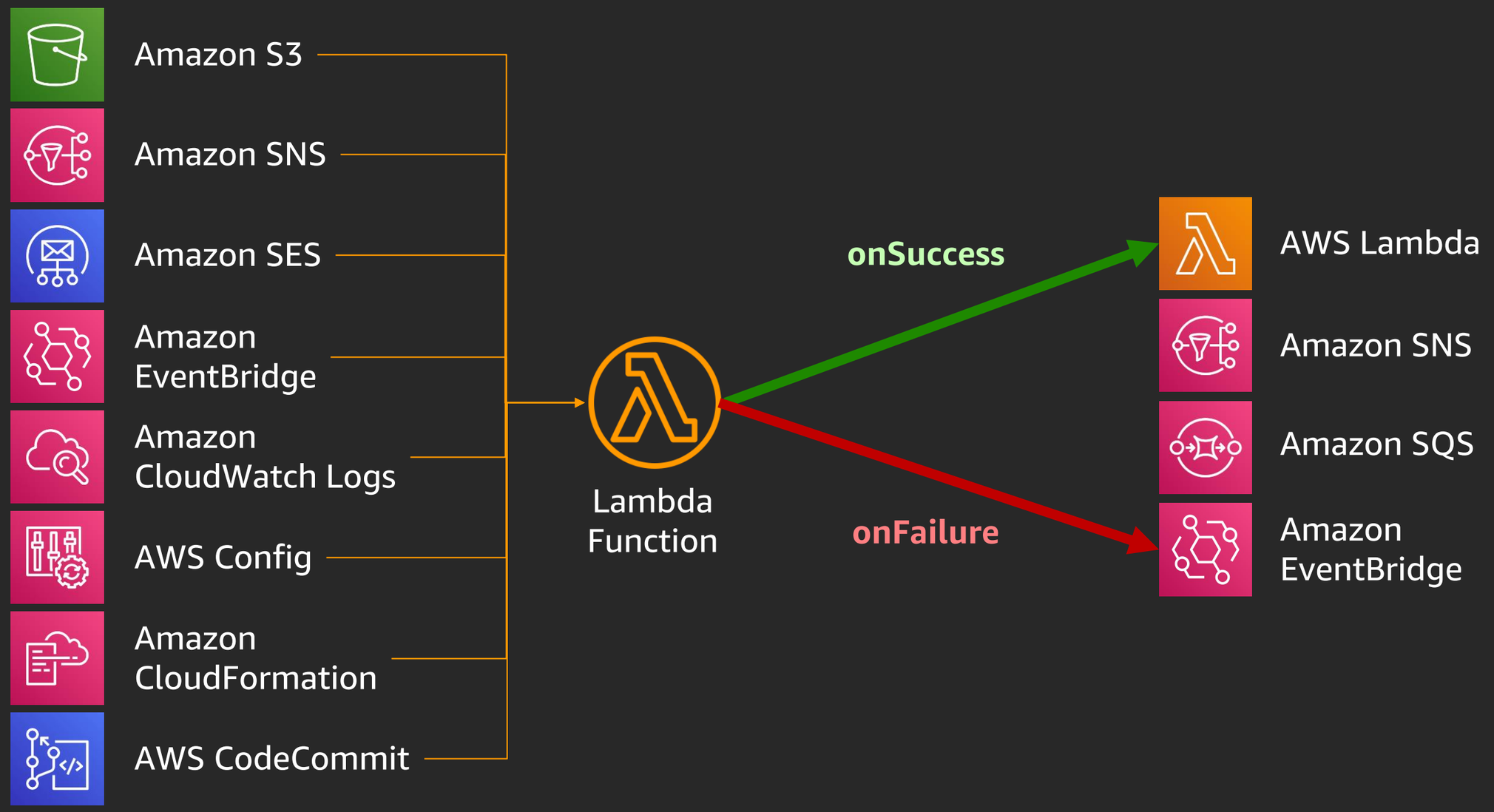
- Record contains details about the request and response in JSON format
- Contains more information than data sent to a DLQ
- Can send both outcomes to the same destination

  or

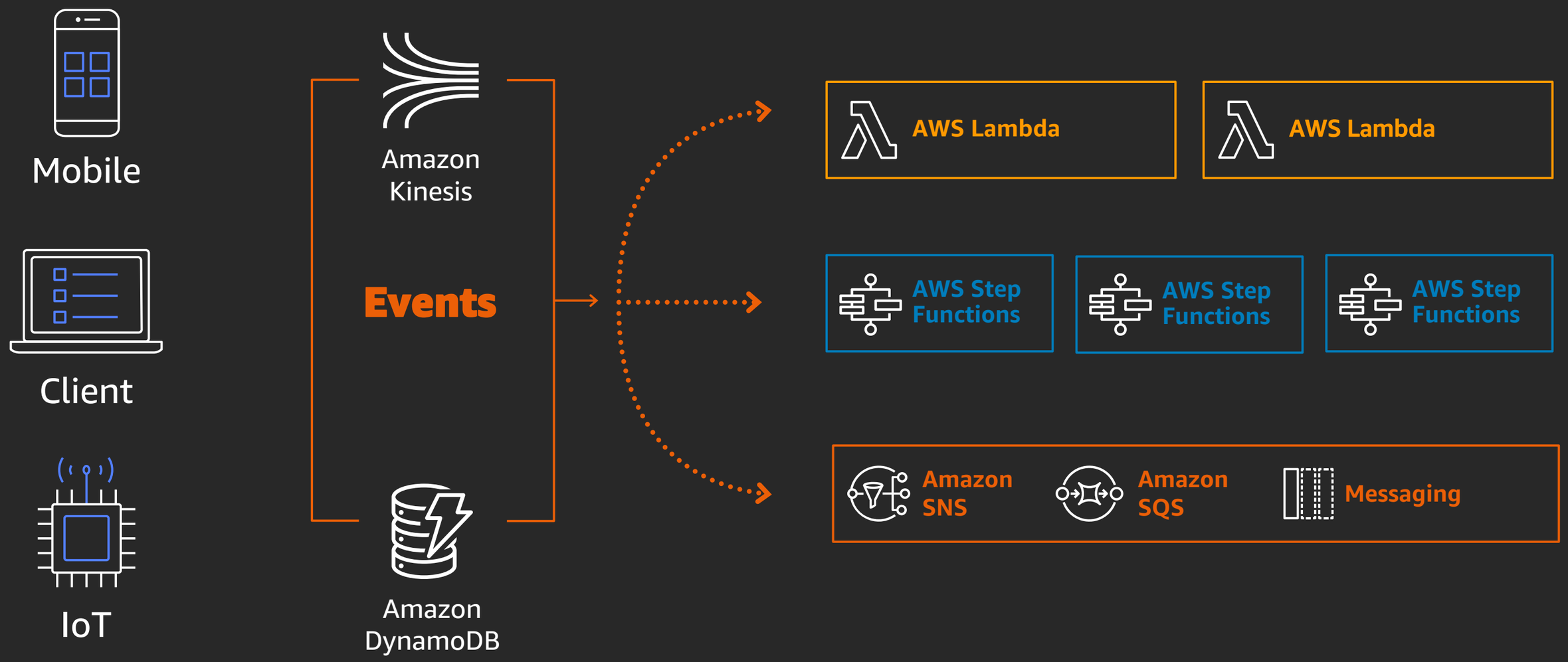
- Can send success to one destination, failure to another



AWS Lambda – Destinations for Async Invocations

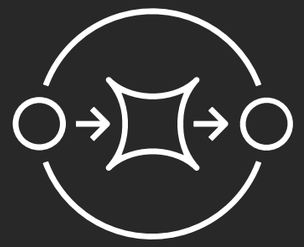


Event-driven architectures



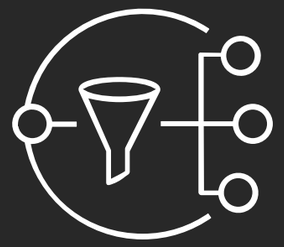
Connecting AWS event sources

Messaging



Amazon SQS

Queues



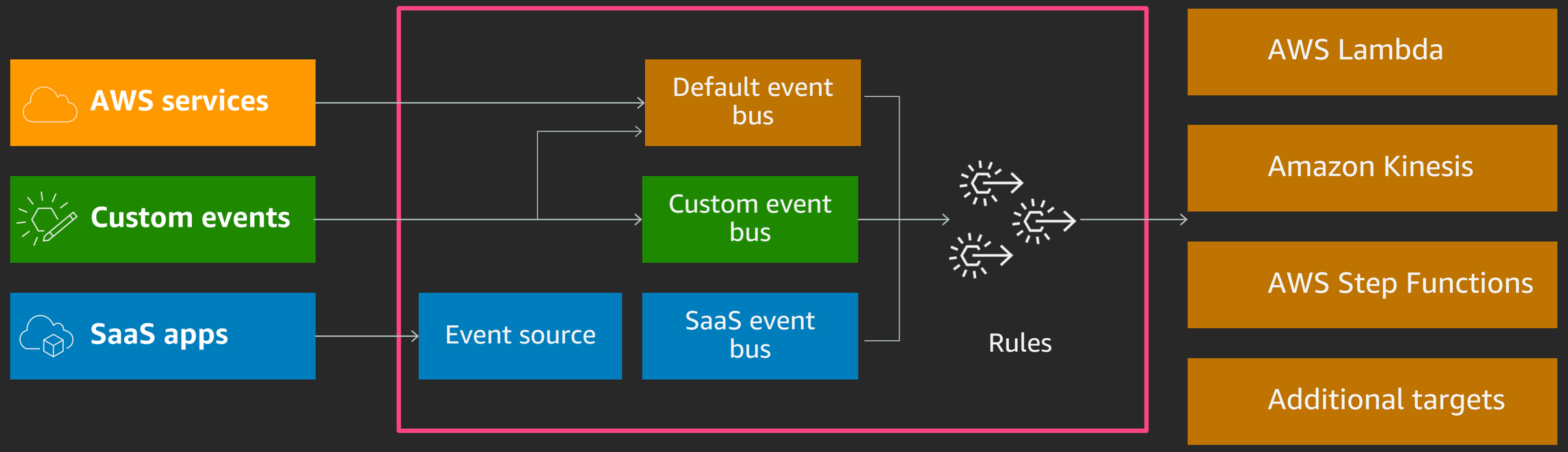
Amazon SNS

Pub/Sub

Amazon EventBridge

Event Bus

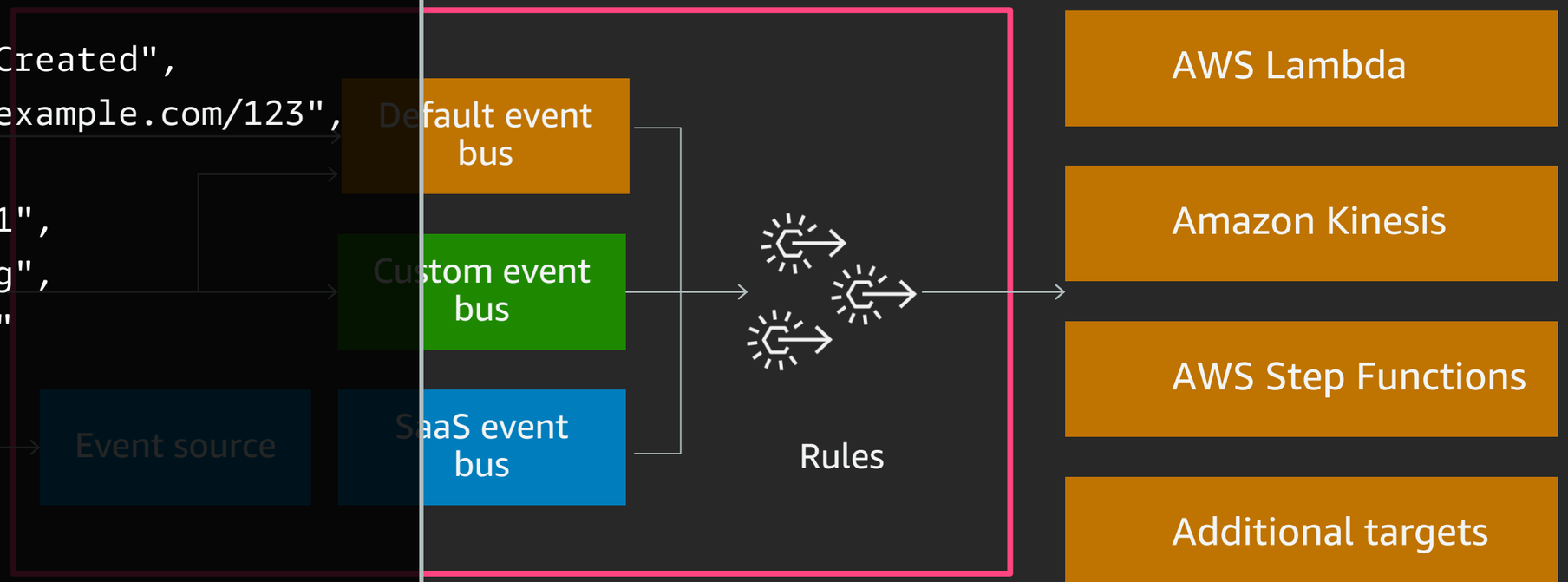
Amazon EventBridge



Amazon EventBridge

Example event:

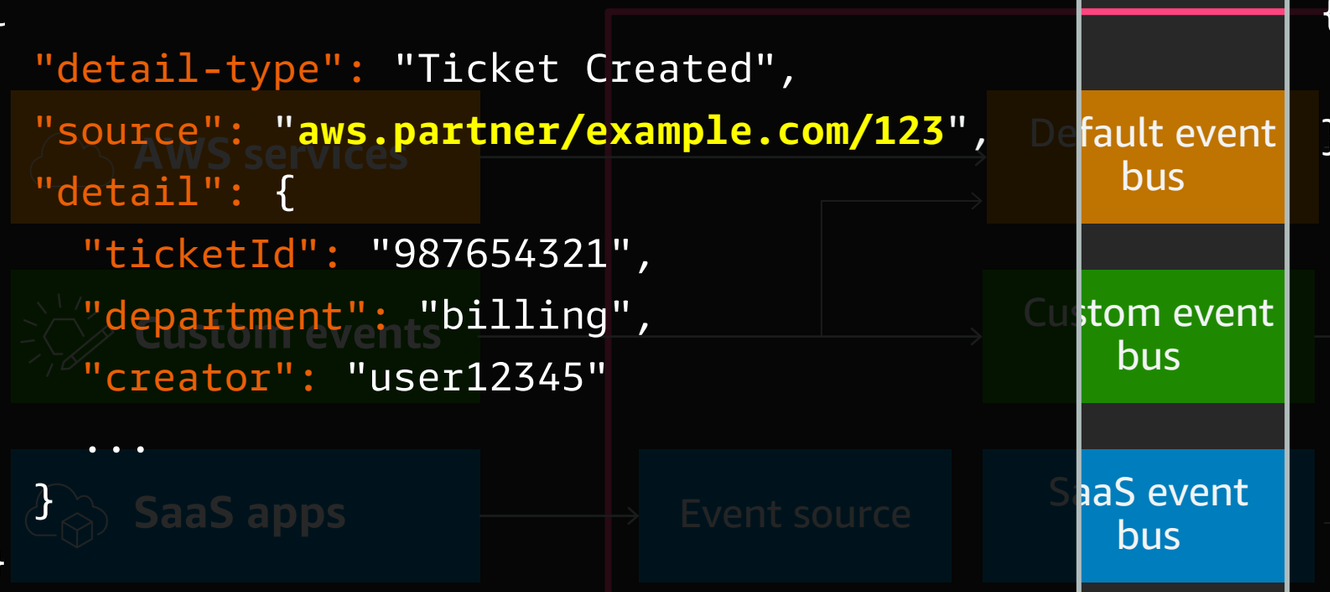
```
{
  "detail-type": "Ticket Created",
  "source": "aws.partner/example.com/123",
  "detail": {
    "ticketId": "987654321",
    "department": "billing",
    "creator": "user12345"
    ...
  }
}
```



Amazon EventBridge

Example event:

```
{
  "detail-type": "Ticket Created",
  "source": "aws.partner/example.com/123",
  "detail": {
    "ticketId": "987654321",
    "department": "billing",
    "creator": "user12345"
    ...
  }
}
```



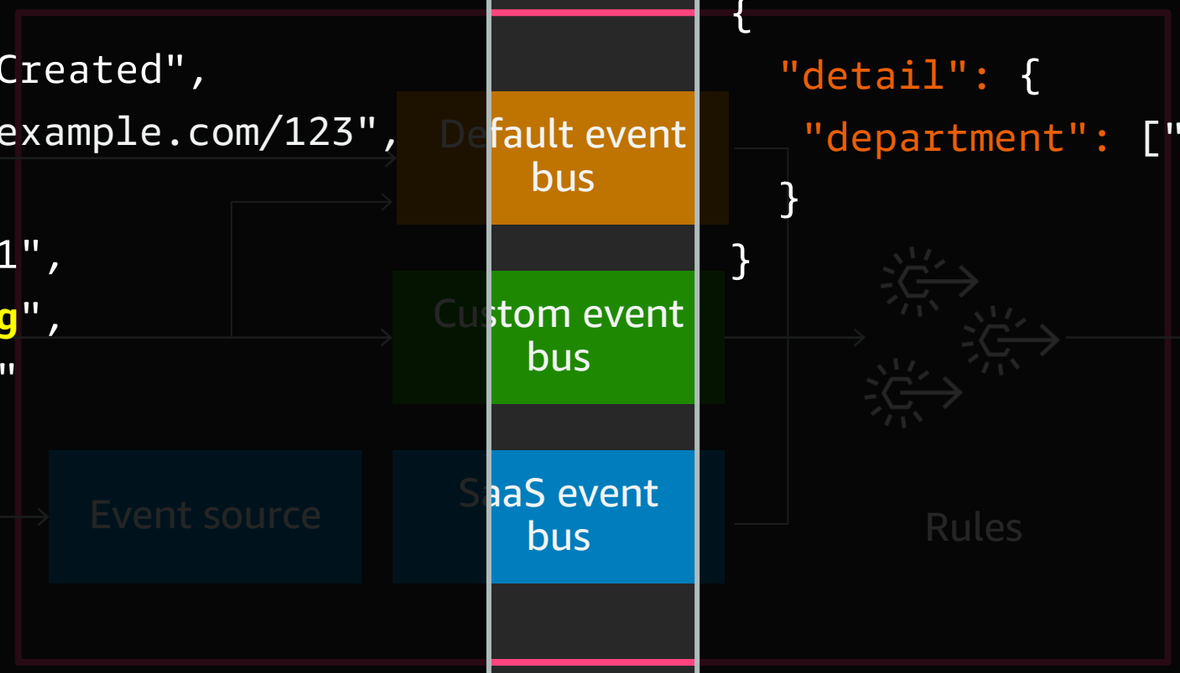
Example rule:

```
{
  "source": ["aws.partner/example.com/123"]
}
```

Amazon EventBridge

Example event:

```
{
  "detail-type": "Ticket Created",
  "source": "aws.partner/example.com/123",
  "detail": {
    "ticketId": "987654321",
    "department": "billing",
    "creator": "user12345"
    ...
  }
}
```



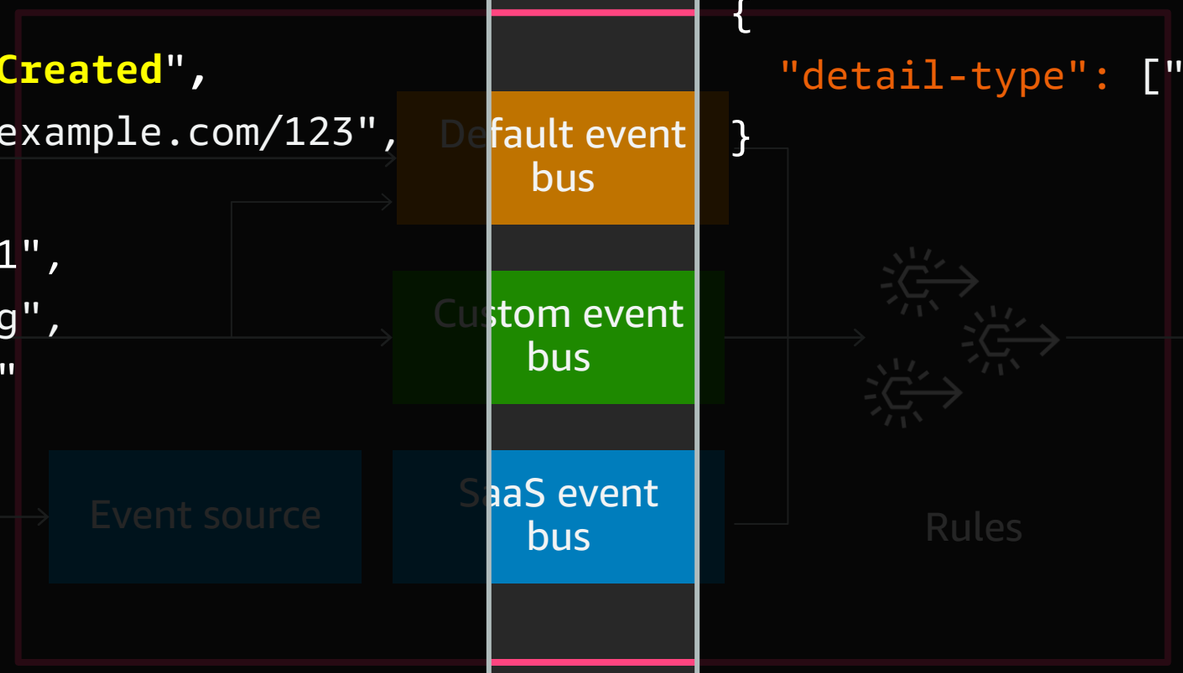
Example rule:

```
{
  "detail": {
    "department": ["billing", "fulfillment"]
  }
}
```

Amazon EventBridge

Example event:

```
{
  "detail-type": "Ticket Created",
  "source": "aws.partner/example.com/123",
  "detail": {
    "ticketId": ”987654321",
    "department": "billing",
    "creator": "user12345"
    ...
  }
}
```

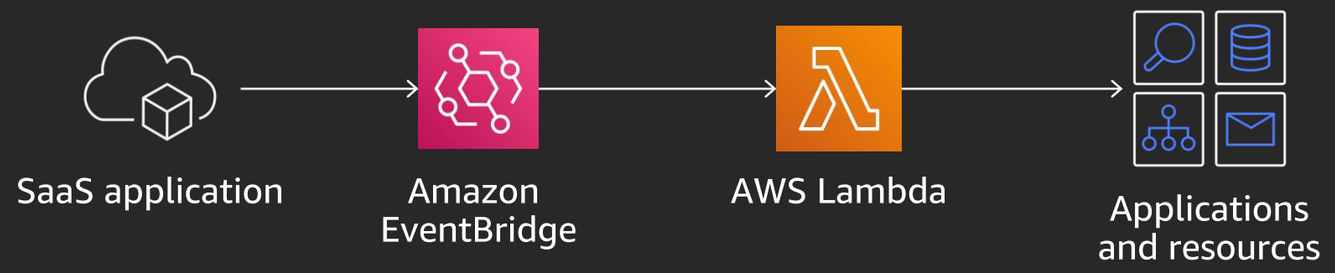


Example rule:

```
{
  "detail-type": ["Ticket Resolved"]
}
```

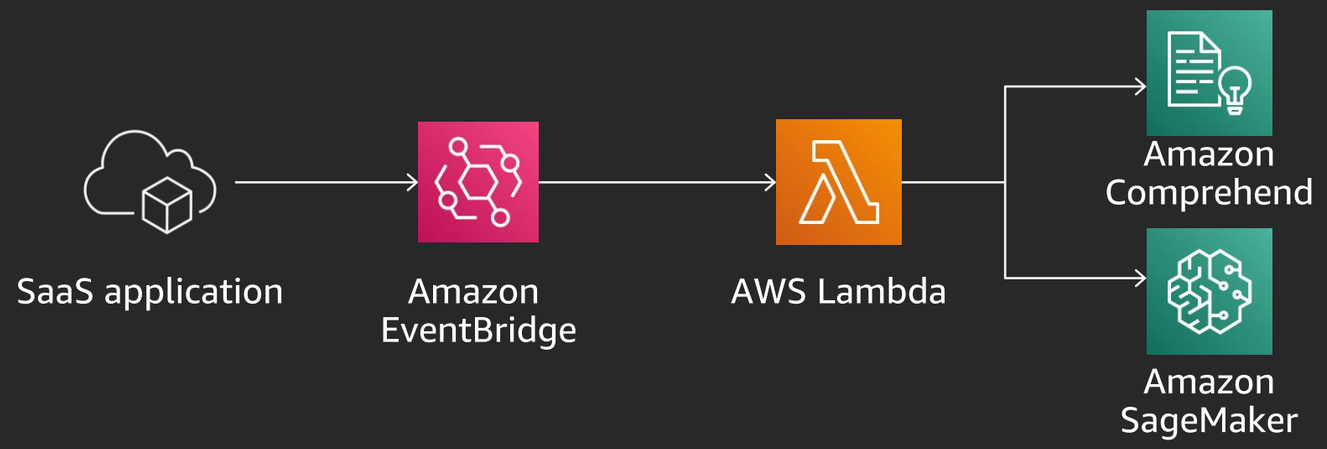
Common use cases

Take action



Run workflows

Apply intelligence



Expanding event sources

SaaS integrations

zendesk, kloudless, epsagon, mongoDB., pagerduty, Buildkite, whispir, DATADOG, Auth0, SAVIYNT, Symantec., Segment, Opsgenie, onelogin, SUGARCRM, New Relic, SignalFx

→

Amazon EventBridge

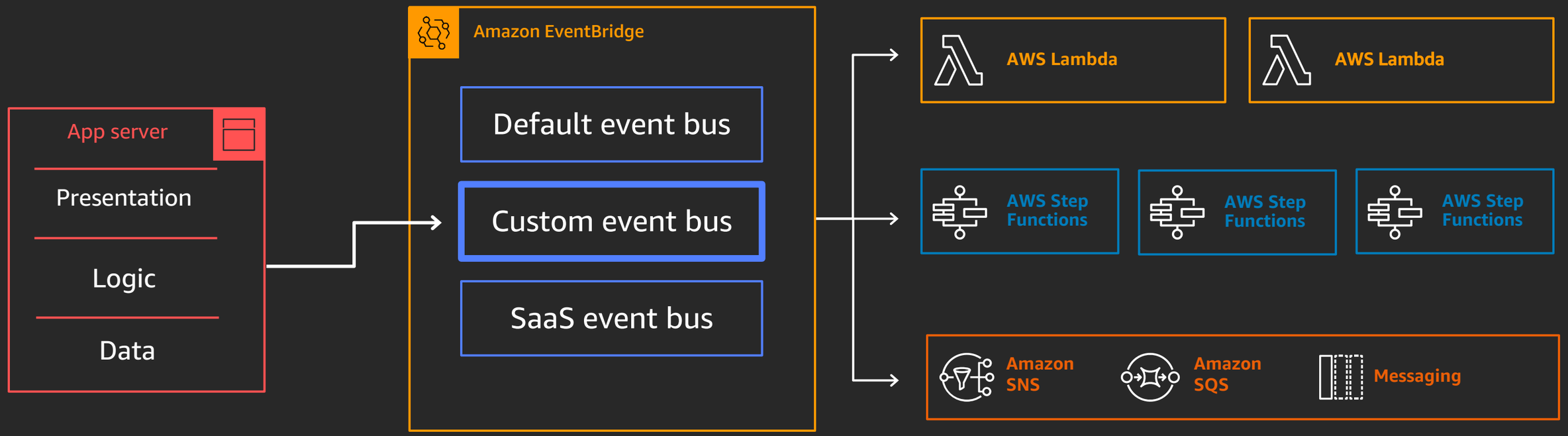
- Default event bus
- Custom event bus
- SaaS event bus

→

Targets

- AWS Lambda
- Amazon Kinesis Data Firehose
- Amazon SNS

Developing with events



Schema registry and discovery

Amazon EventBridge

Source of truth for sharing schema

Explicitly publish and discover

Integrations for JetBrains and VS Code

Language bindings for Java, Python, and TypeScript

Customer case study

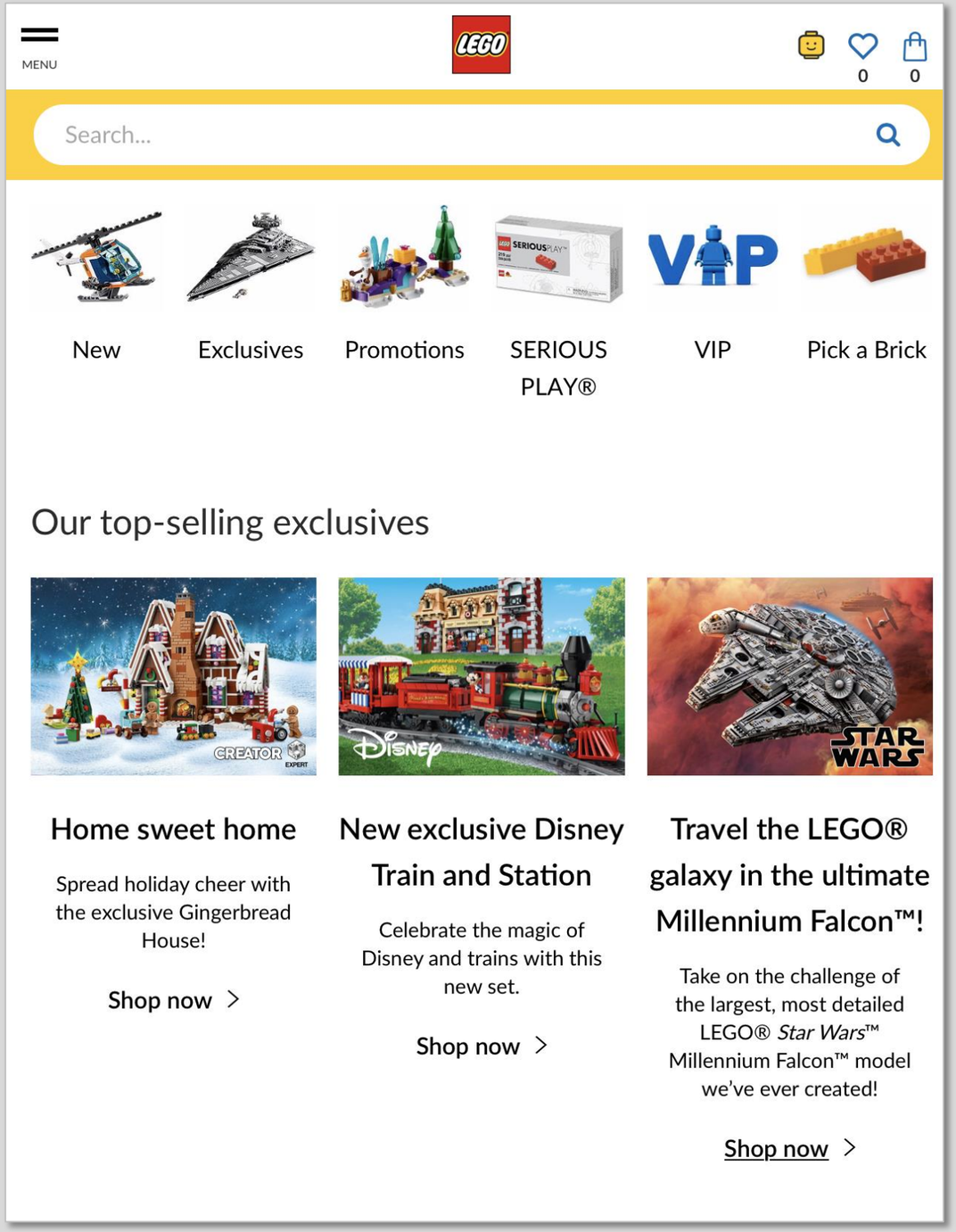
July 10, 2019

shop.LEGO.com

was switched to

serverless

on AWS



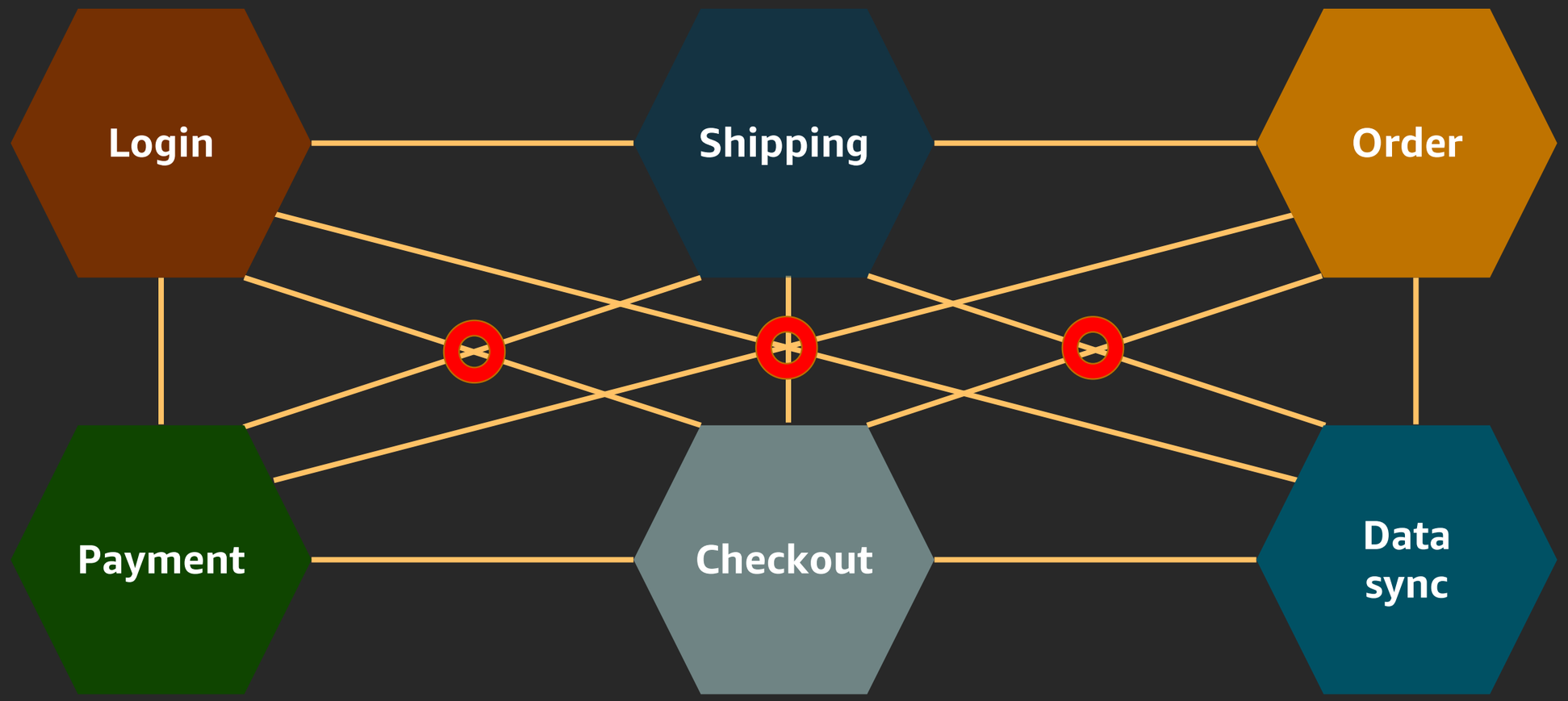
July 11, 2019

The day after,

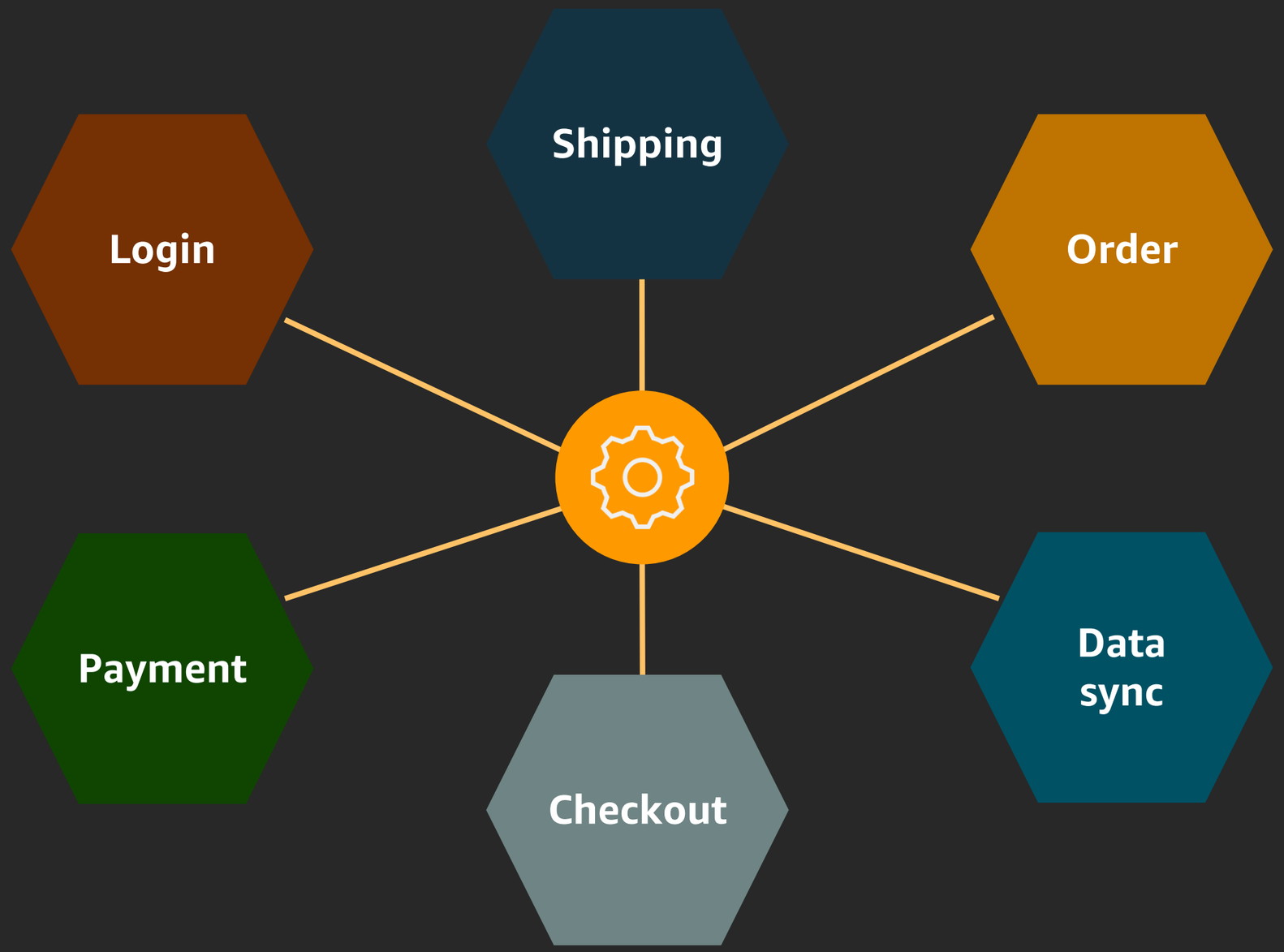
Amazon EventBridge

was launched

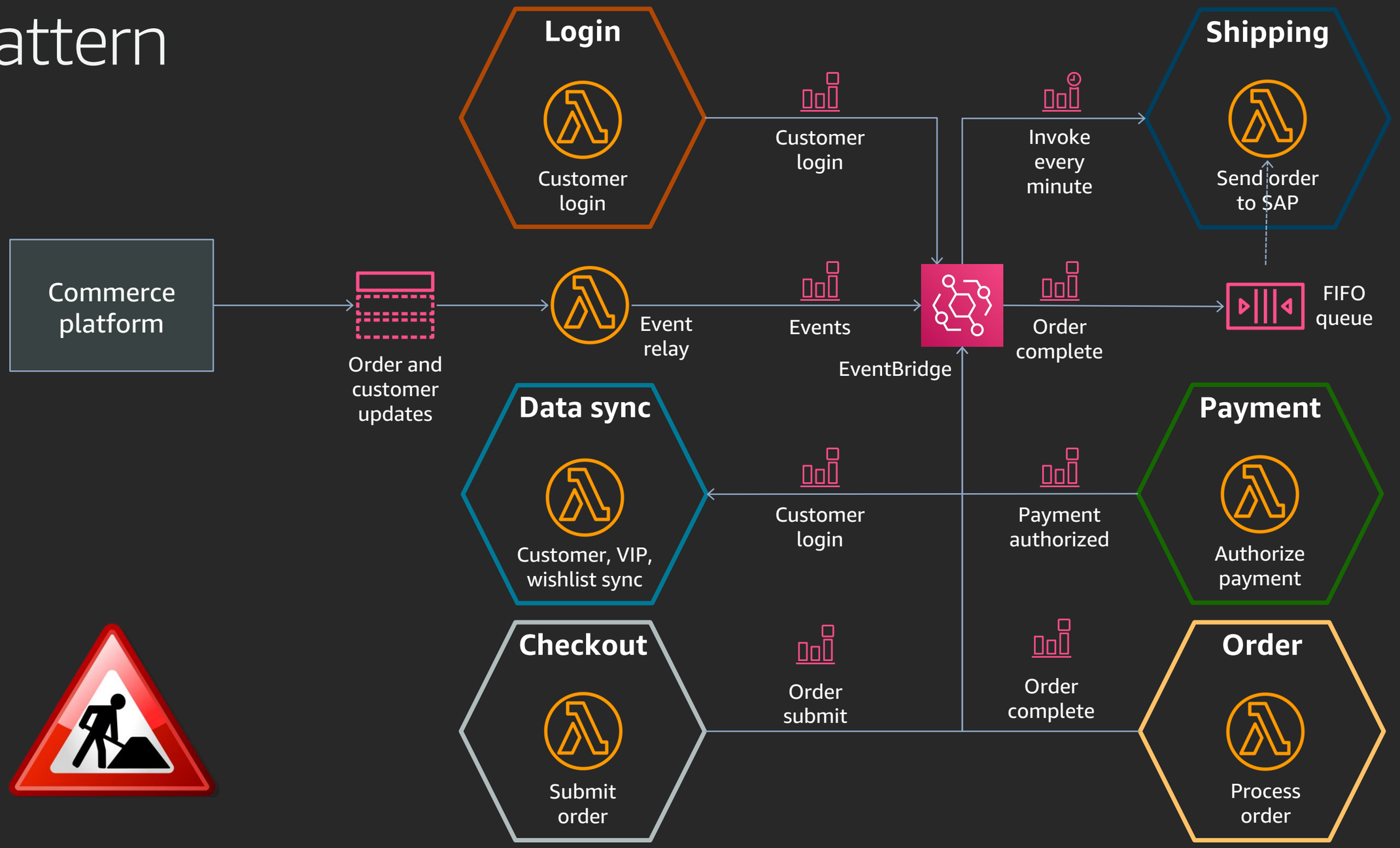
Checkout event processing



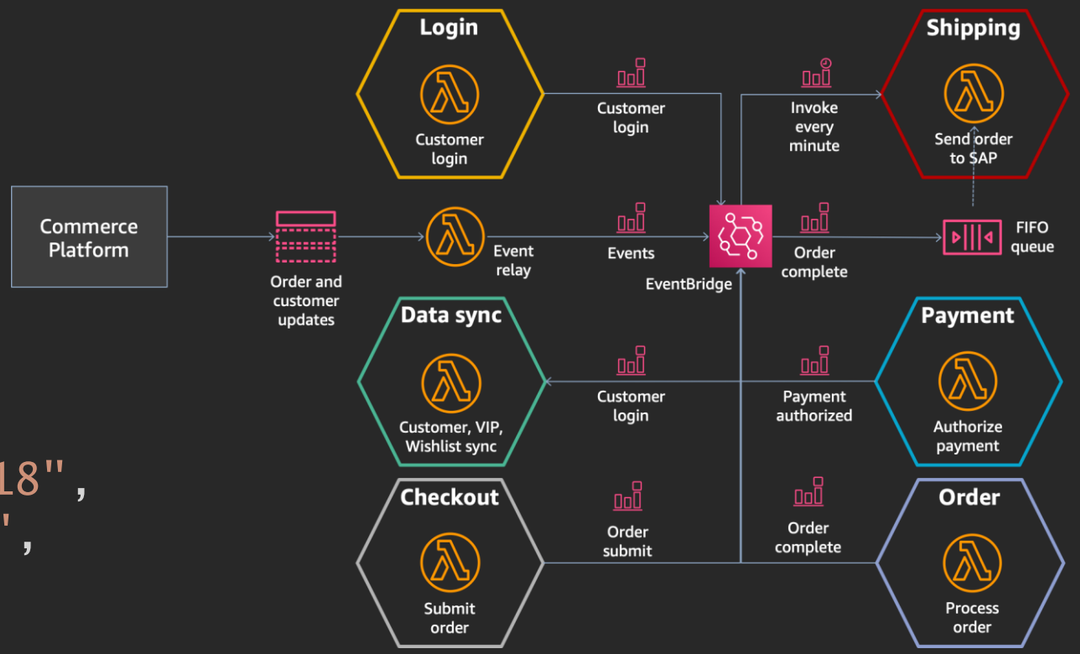
Hub-and-spoke event bus



Pattern


Login
Customer login
Customer login
Invoke every minute
Shipping
Send order to SAP
Commerce platform
Order and customer updates
Event relay
Events
EventBridge
Order complete
FIFO queue
Data sync
Customer, VIP, wishlist sync
Customer login
Payment authorized
Payment
Authorize payment
Checkout
Submit order
Order submit
Order complete
Order
Process order

Pattern – Hub-and-spoke event bus




```
{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "State change Notification",
  "source": "service-order-submit-dev",
  "account": "111122223333",
  "time": "2019-08-29T12:10:21Z",
  "region": "eu-central-1",
  "resources": [
    "arn:aws:events:event-bus/checkout-bus"
  ],
  "detail": {
  }
}
```

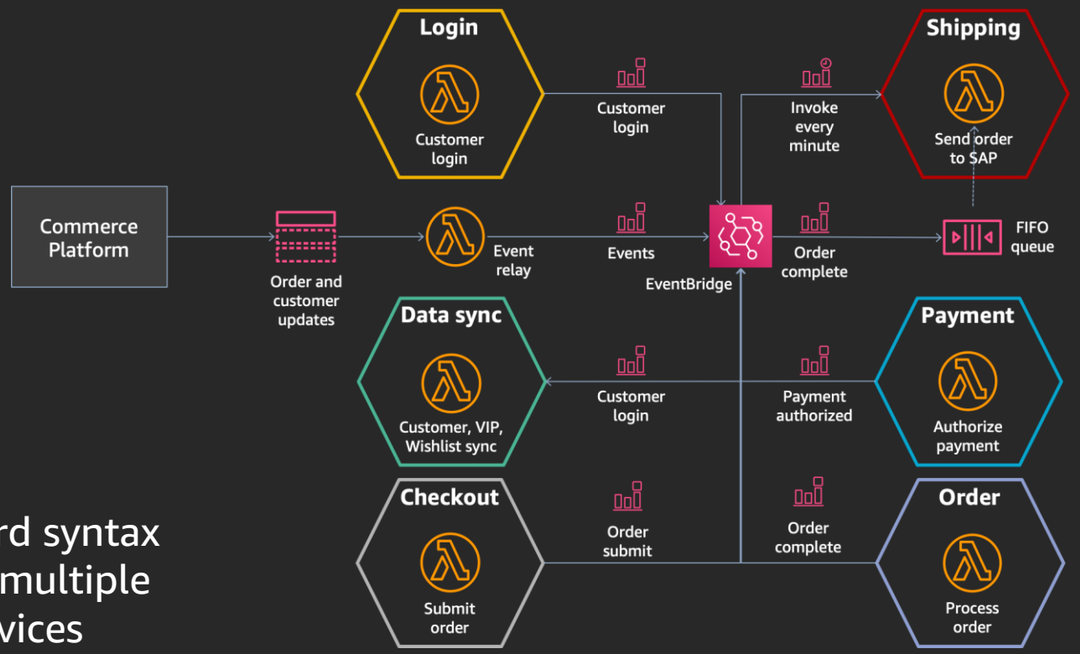

Customer-specific data goes in the “detail”

Pattern – Hub-and-spoke event bus


```
{
 "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "State change Notification",
  "source": "service-order-submit-dev",
  "account": "123456789012",
  "time": "2019-08-29T12:10:21Z",
  "region": "eu-central-1",
  "resources": ["arn:aws:events:event-bus/checkout-bus"],
   "detail": {
     "event": {
       "meta_data": {
         "site_id": "LEGO Shop",
         "type": "CHECKOUT",
         "subtype": "ORDER",
         "status": "COMPLETE"
       },
       "data": {
         "order_number": "T123456789",
         "customer_id": "bf3703467718-29T12-6a7e8feb"
       }
     }
   }
}
```


Standard syntax across multiple services

Custom for each service

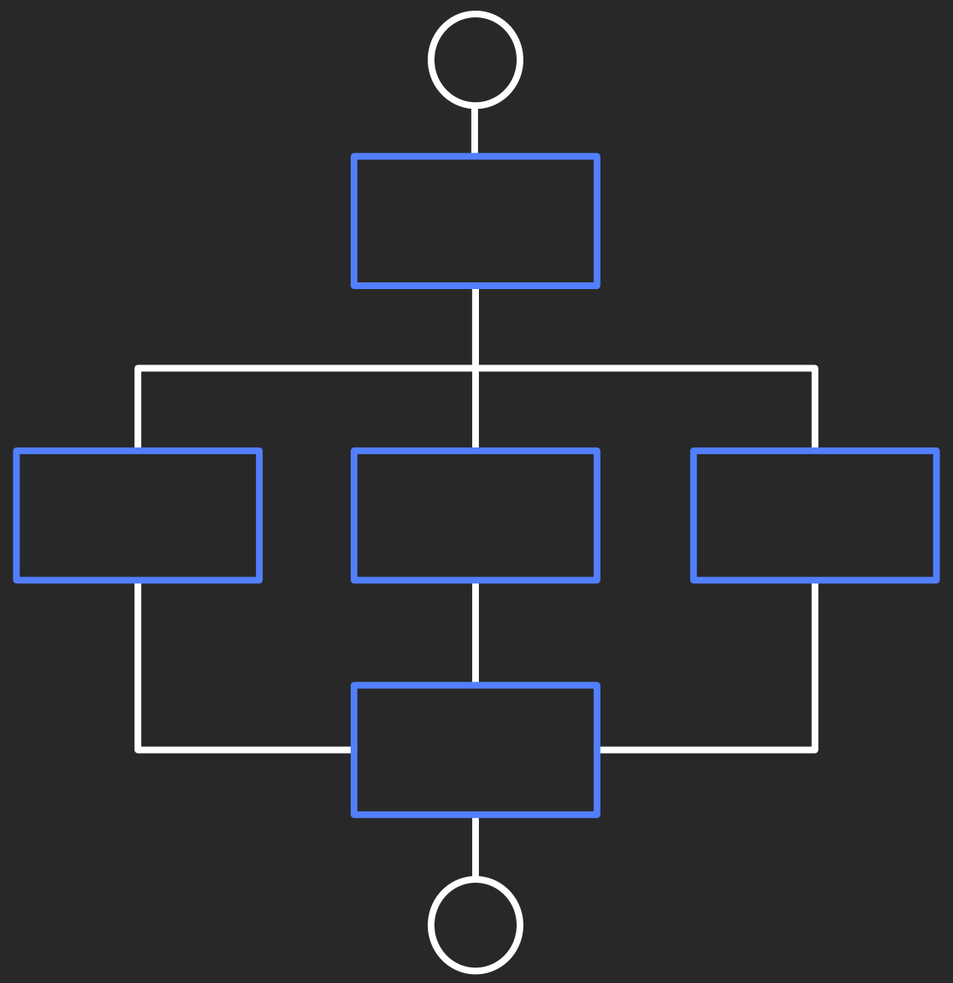


Orchestration with AWS Step Functions

aws SUMMIT ONLINE

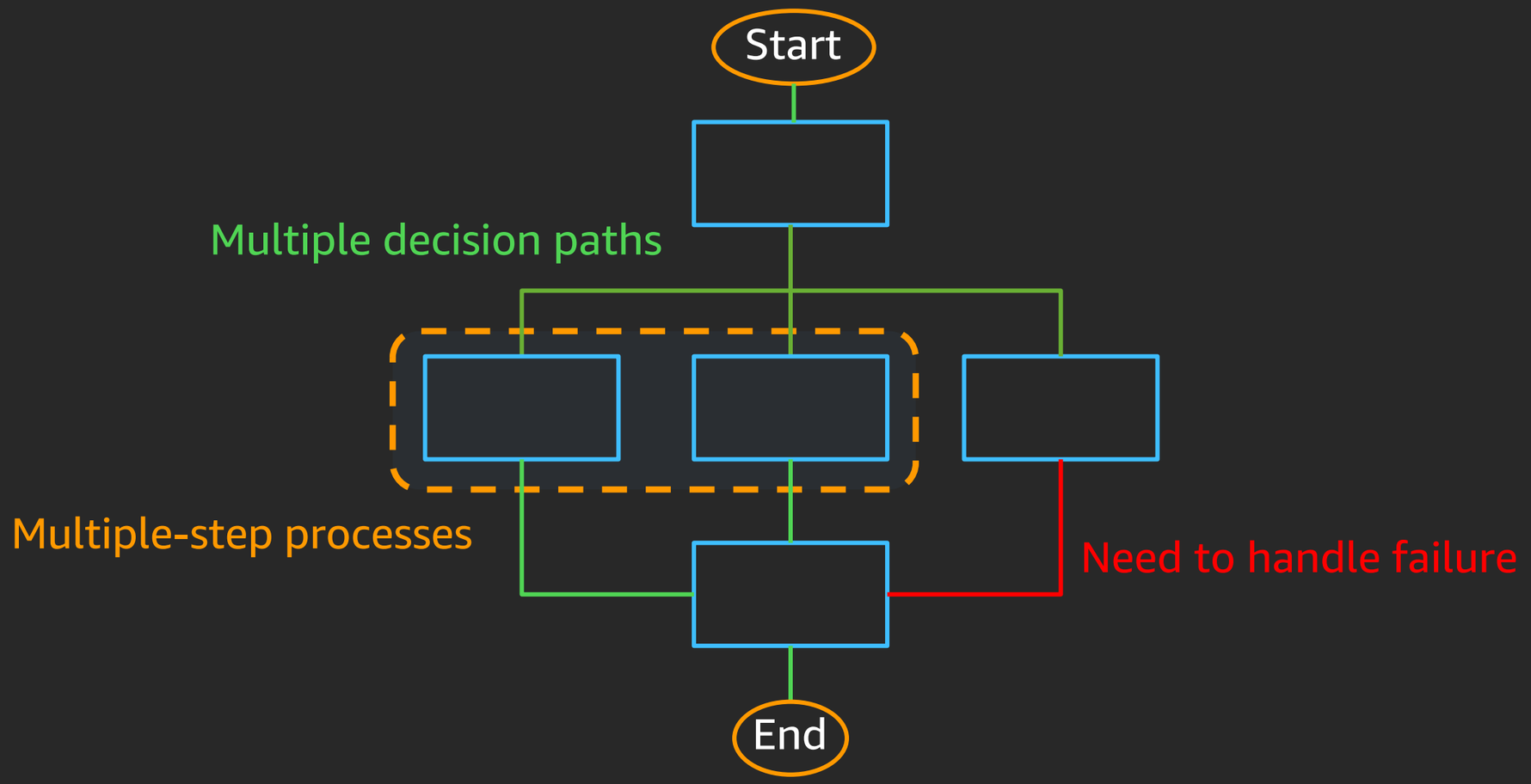
Coordinate function execution

Track status of data and execution



Remove redundant code

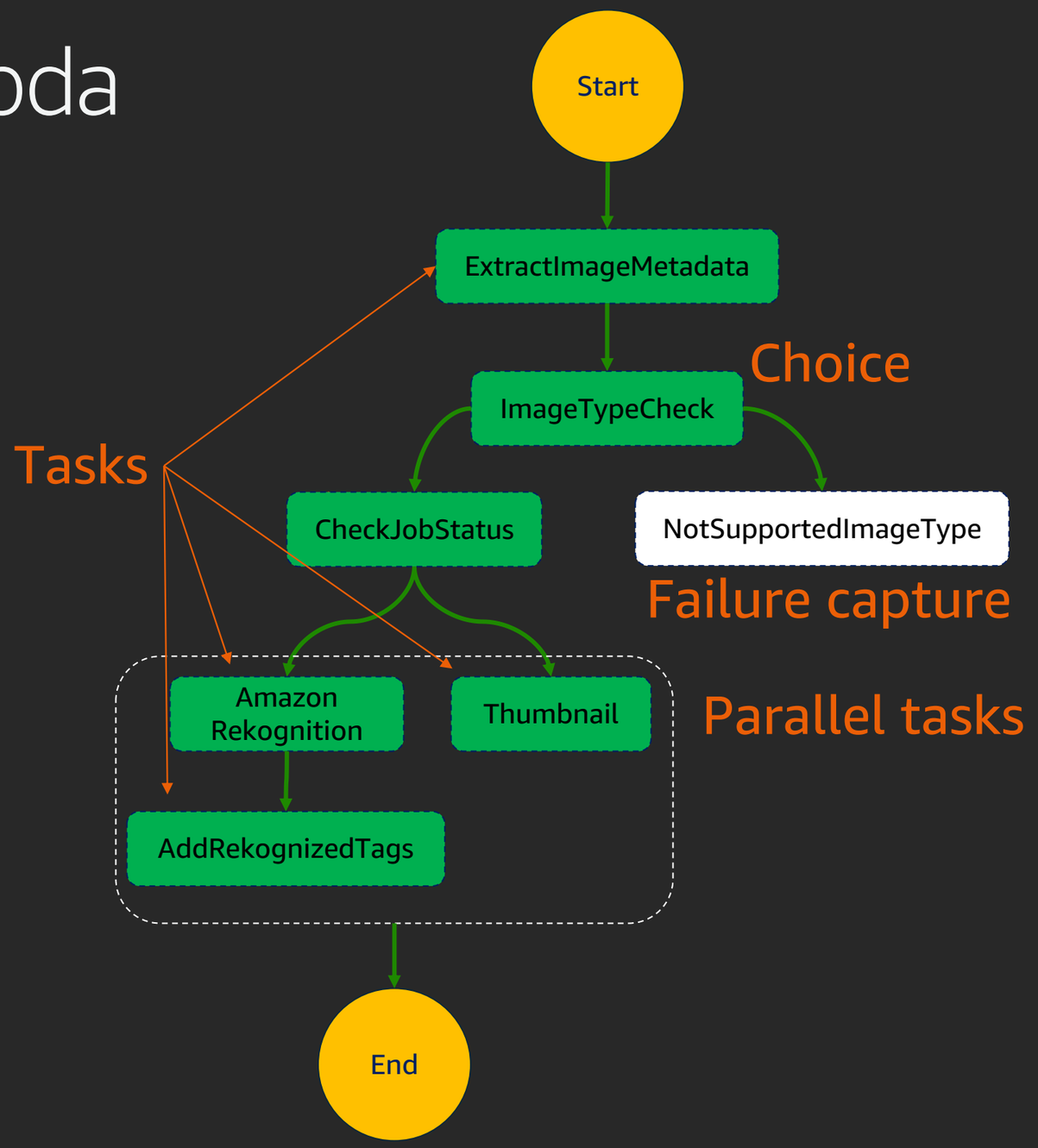
Business workflow is rarely sequential start to finish



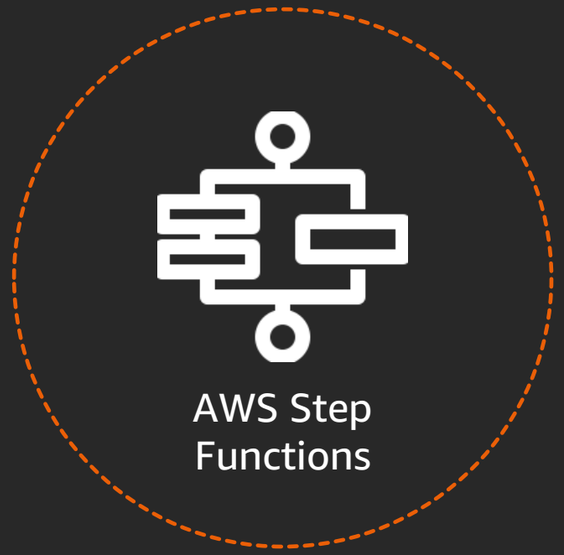
AWS Step Functions + Lambda

"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 quickly diagnose and debug problems



AWS Step Functions: Integrations



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 SNS


Amazon SQS


Amazon SageMaker


AWS Glue


AWS Batch


Amazon ECS


AWS Fargate

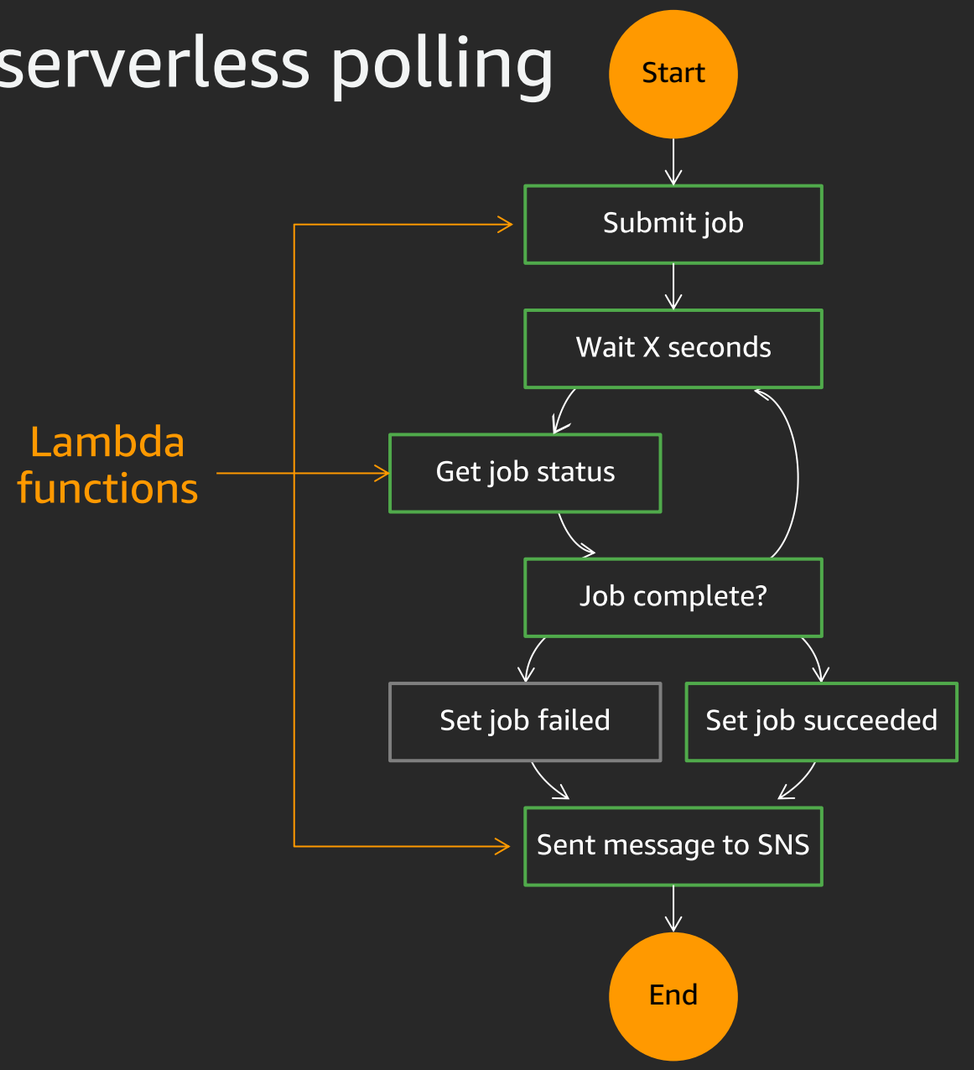
New!


Amazon EMR

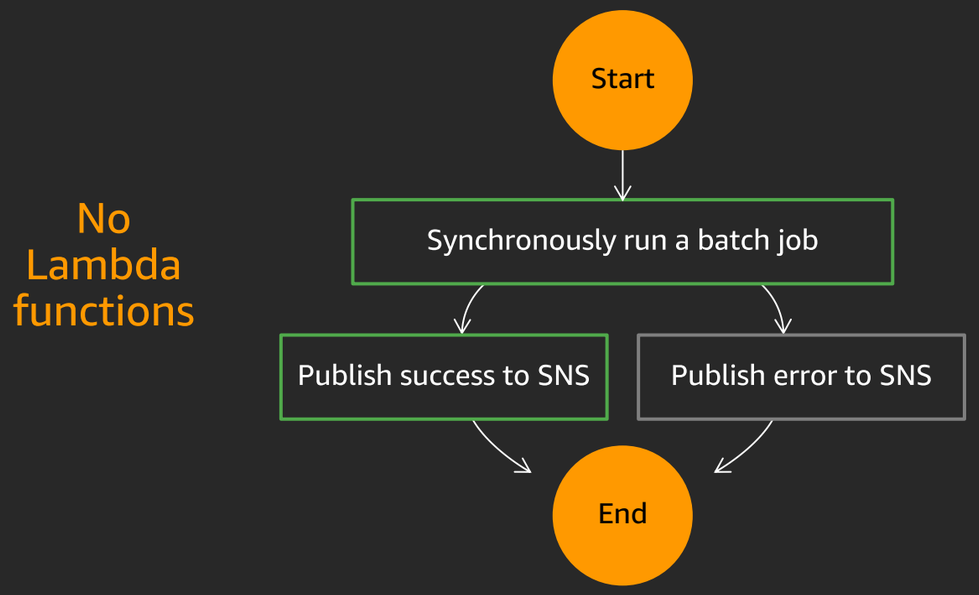
NEW!!!

Simpler integration, less code

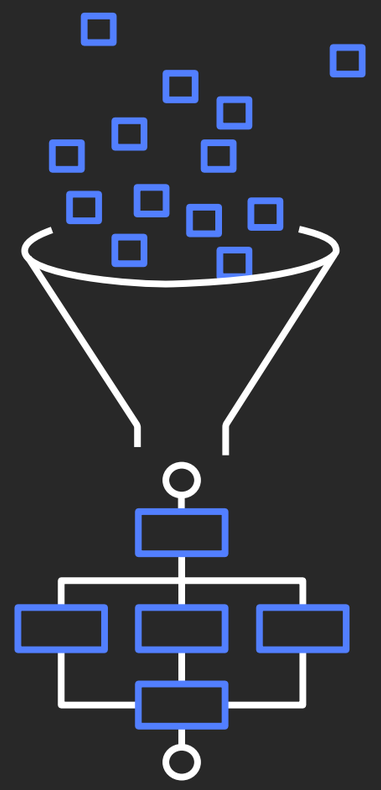
With serverless polling



With direct service integration



Express workflows



Event streams over 100,000 per second

High volume, short duration

Cost effective at scale

Streaming data processing – IoT ingestion

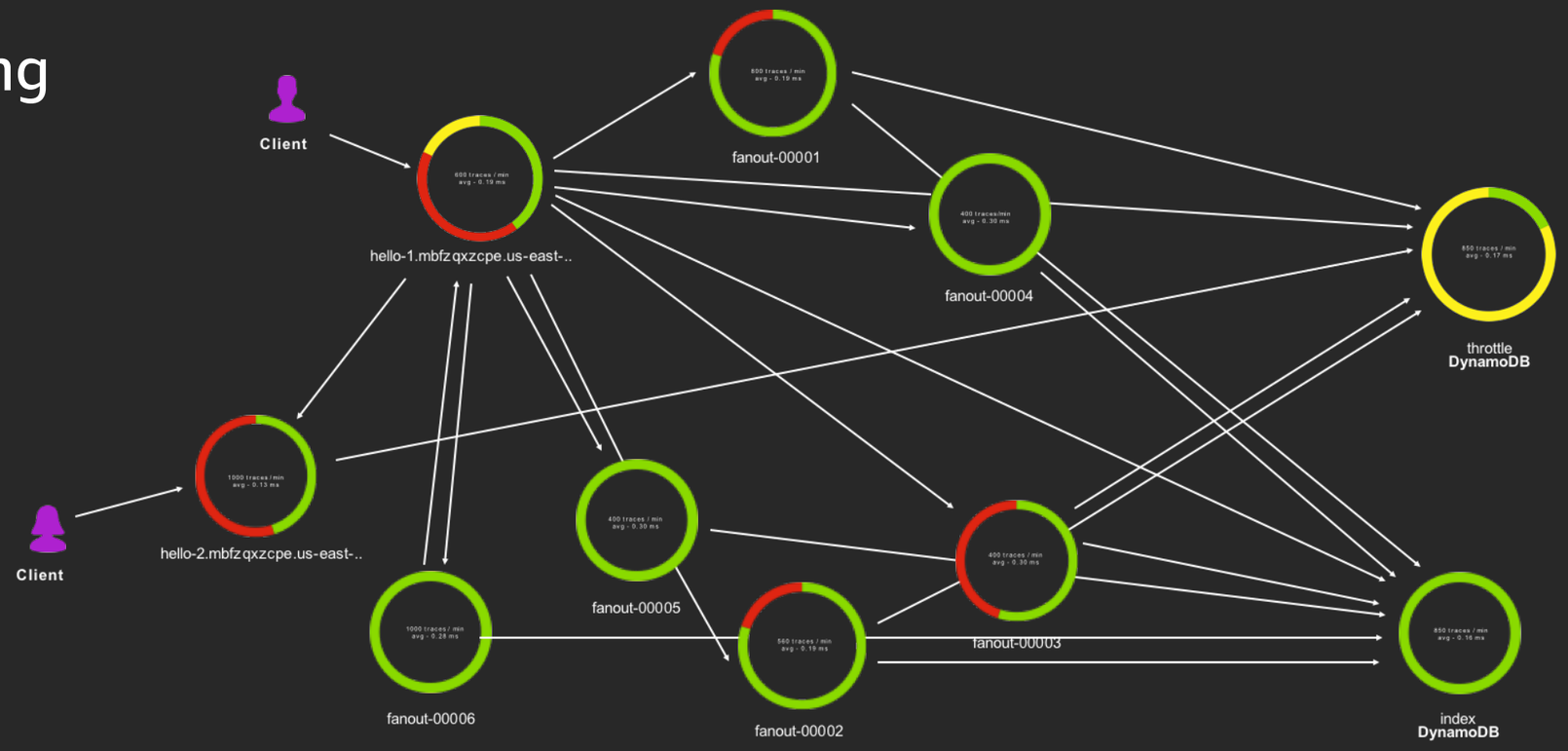
Tracing and observability

AWS X-Ray

Profile and troubleshoot distributed applications:

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

```
var AWSXRay = require('aws-xray-sdk-core');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));
S3client = AWS.S3();
```

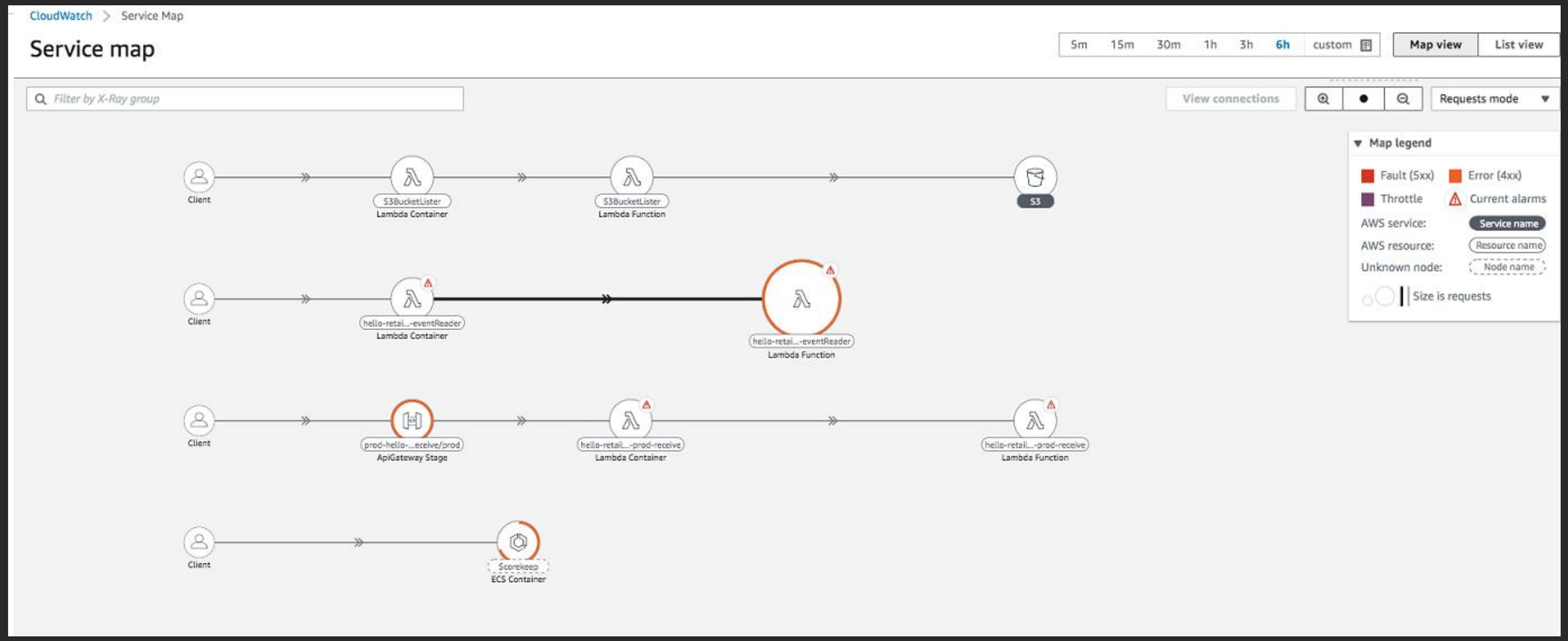


Enable X-Ray Tracing

Enable active tracing Info

CloudWatch ServiceLens – a single interface for traces, metrics, logs, and alarms

Preview



CloudWatch Embedded Metric Format – custom metrics from ephemeral resources (functions, containers)

Installation

```
npm install aws-embedded-metrics
```

Usage

To get a metric logger, you can either decorate your fu

Using the metricScope decorator without function para

```
const { metricScope, Unit } = require("aws-em

const myFunc = metricScope(metrics =>
  async () => {
    metrics.putDimensions({ Service: "Aggrega
    metrics.putMetric("ProcessingLatency", 10
    metrics.setProperty("RequestId", "422b156
    // ...
  });

await myFunc();
```

Installation

```
pip3 install aws-embedded-metrics
```

Usage

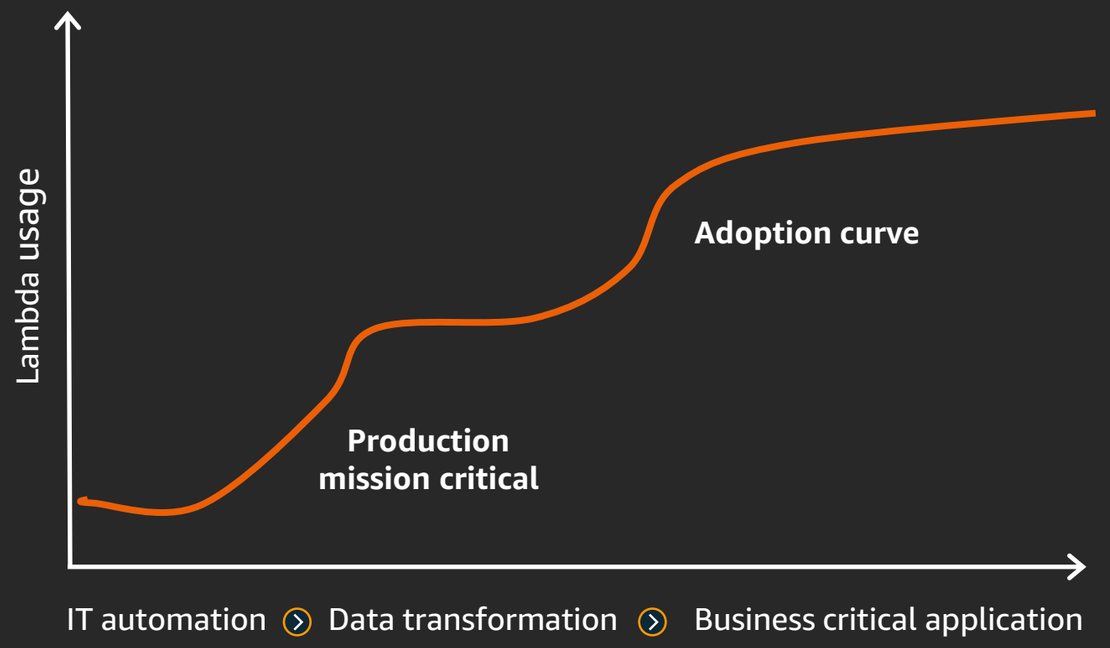
To get a metric logger, you can decorate your function with a `metric_scope`:

```
from aws_embedded_metrics import metric_scope

@metric_scope
def my_handler(metrics):
    metrics.put_dimensions({"Foo": "Bar"})
    metrics.put_metric("ProcessingLatency", 100, "Milliseconds")
    metrics.set_property("AccountId", "123456789012")
    metrics.set_property("RequestId", "422b1569-16f6-4a03")
    metrics.set_property("DeviceId", "61270781-c6ac-46f1")

    return {"message": "Hello!"}
```

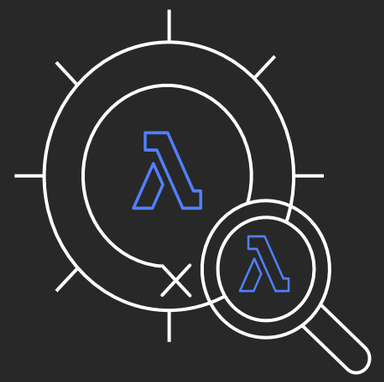
What now?

Finding success

Organic



Serverless first



Considerations

Rapid development

Time-to-market – agility

Changing business

More tooling available

serverless framework

HashiCorp Terraform

DATADOG

epsagon

lumigo

New Relic

THUNDRA™

SignalFx

a Splunk® company

sumo logic

STACKERY

Thank you!

Kim Kao

@Yikaikao Kimkao.solid

aws SUMMIT ONLINE