

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

# Agenda

From APIs to events

Event-driven choreography with Amazon EventBridge

Customer case study

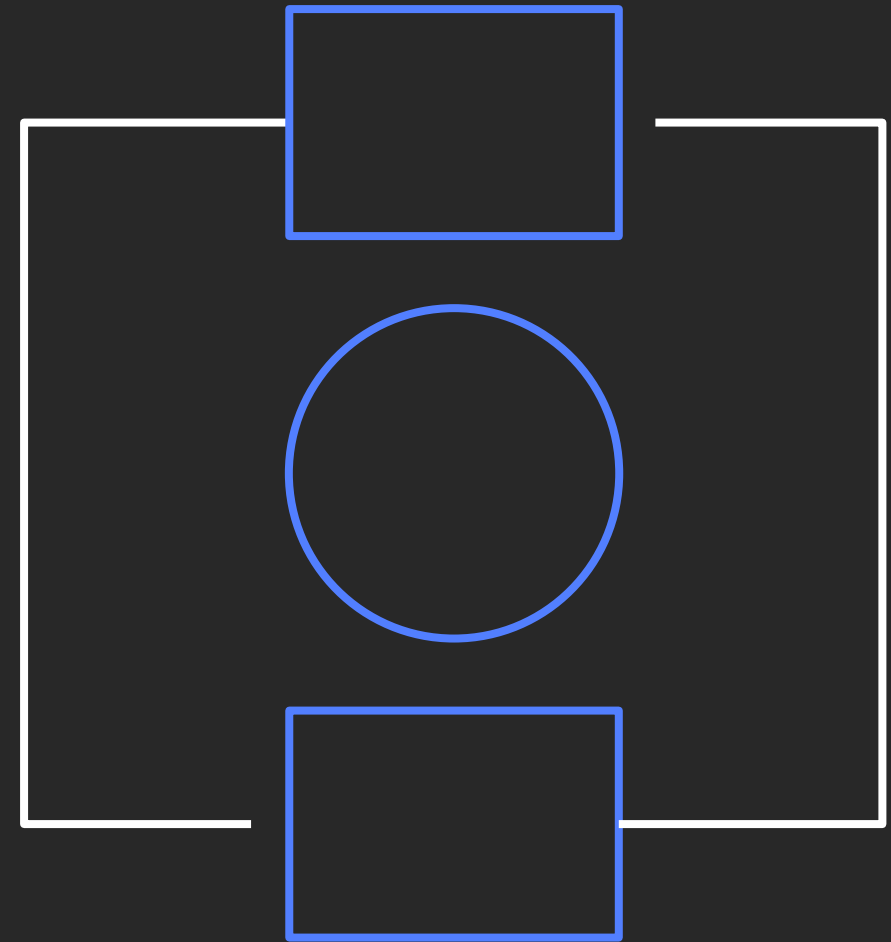
Orchestration with AWS Step Functions

Tracing and observability

# From APIs to events

Small pieces loosely joined

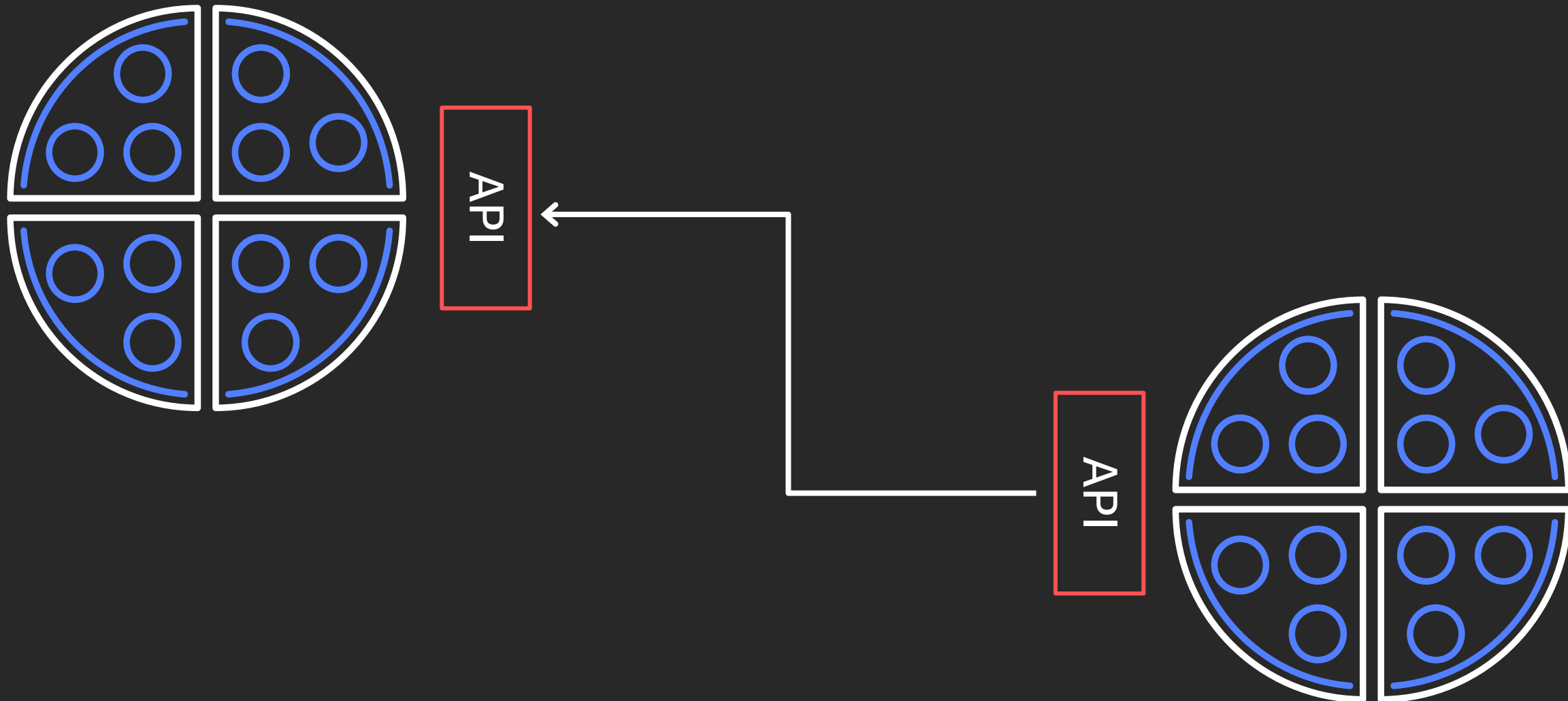
Cloud-native architectures are  
small pieces, loosely joined



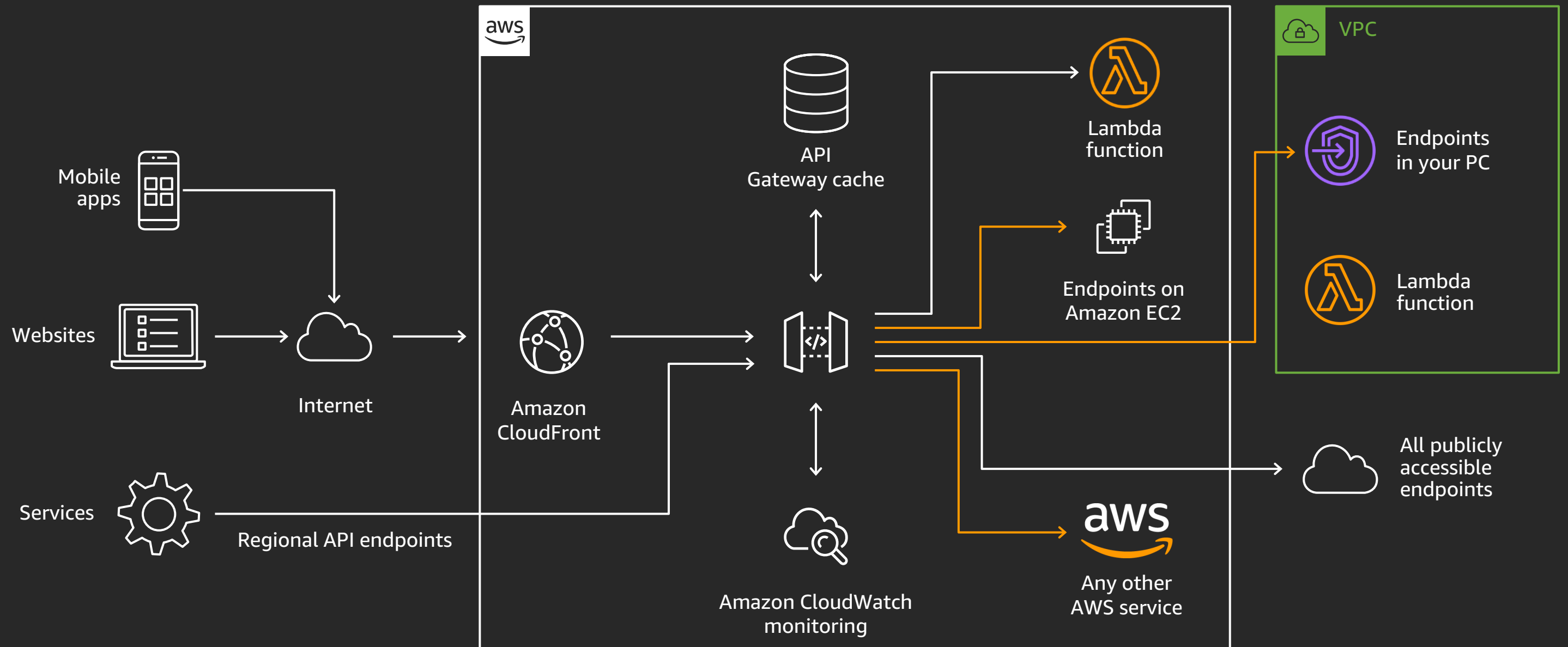


APIs are the front door  
of microservices

# APIs are hardened contracts



# Manage APIs with Amazon API Gateway

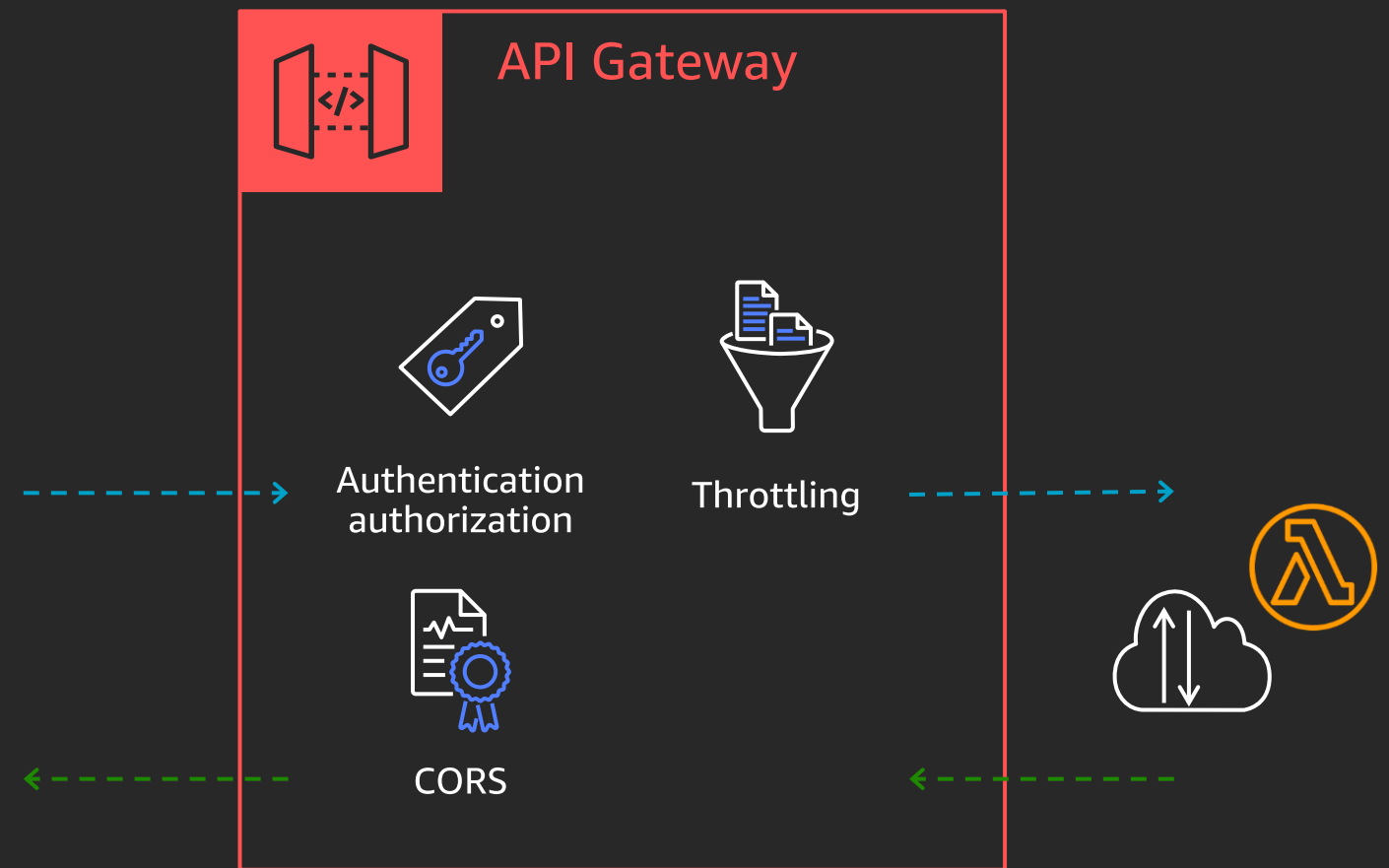


# HTTP APIs

**Up to 70% lower cost**

**50% lower latency**

**Standard authentication**

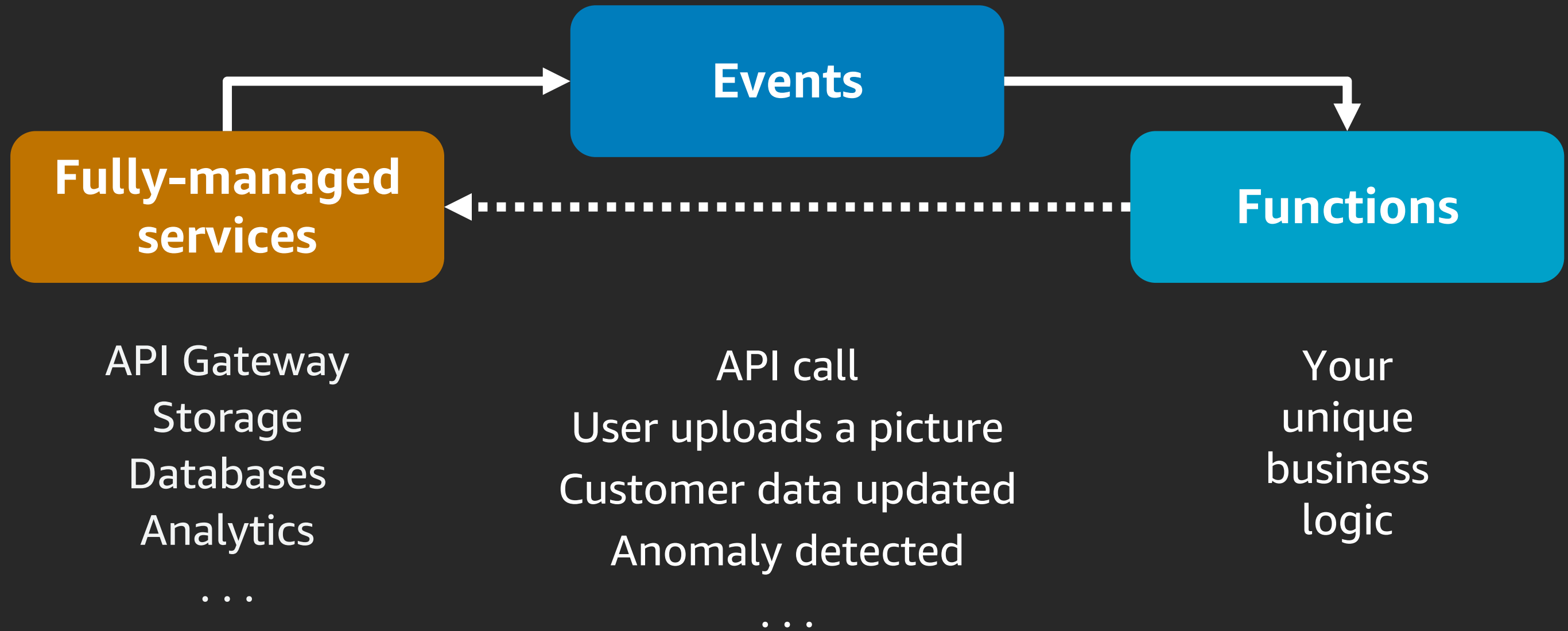




Serverless application

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

# How do Serverless applications work?



# 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

# 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

Has an intent  
Directed to a target  
Personal communication

"CreateAccount"  
"AddProduct"

## Event

It's a fact  
For others to observe  
Broadcast one to many

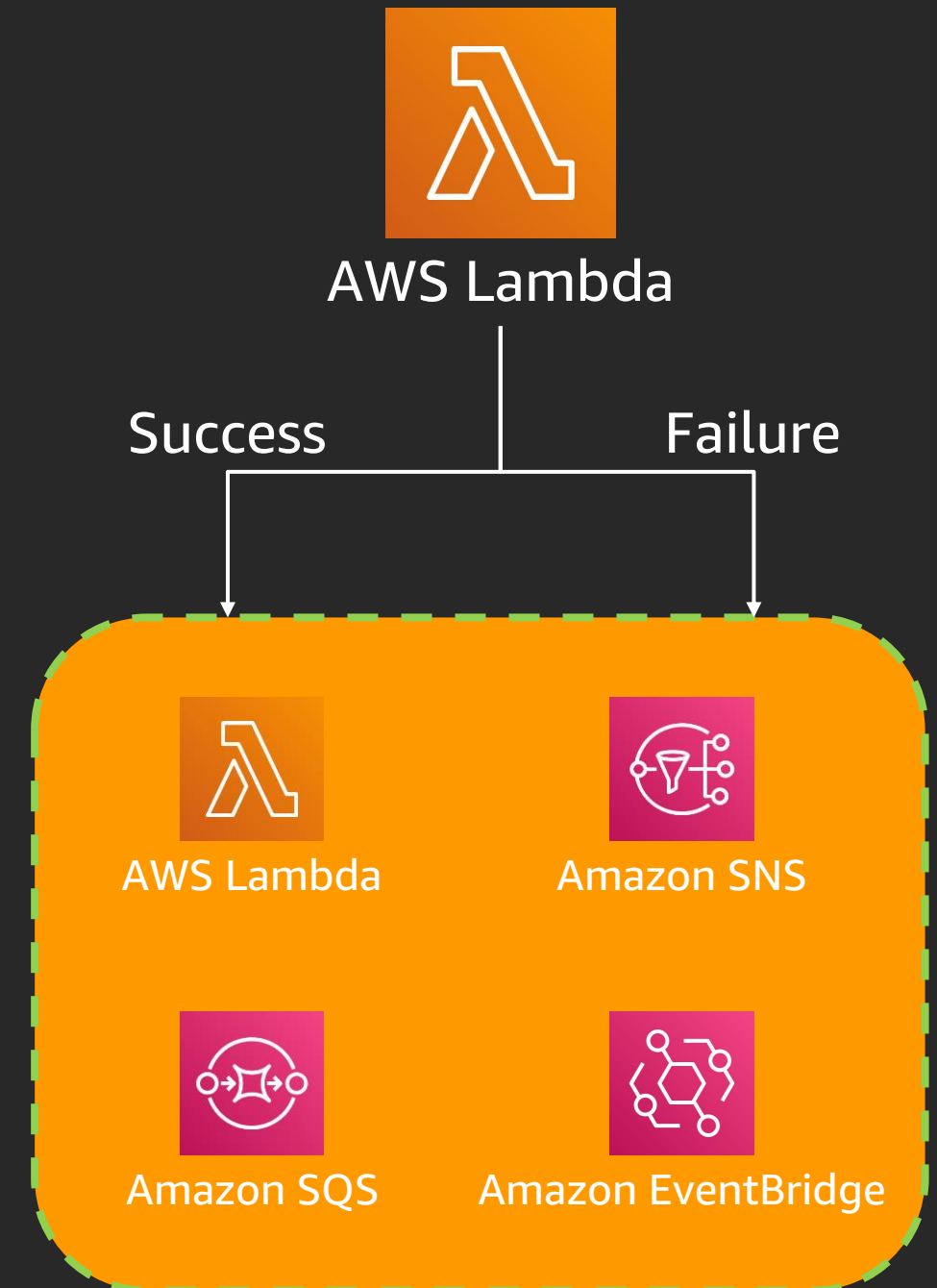
"AccountCreated"  
"ProductAdded"

# Introducing AWS Lambda Event Destinations

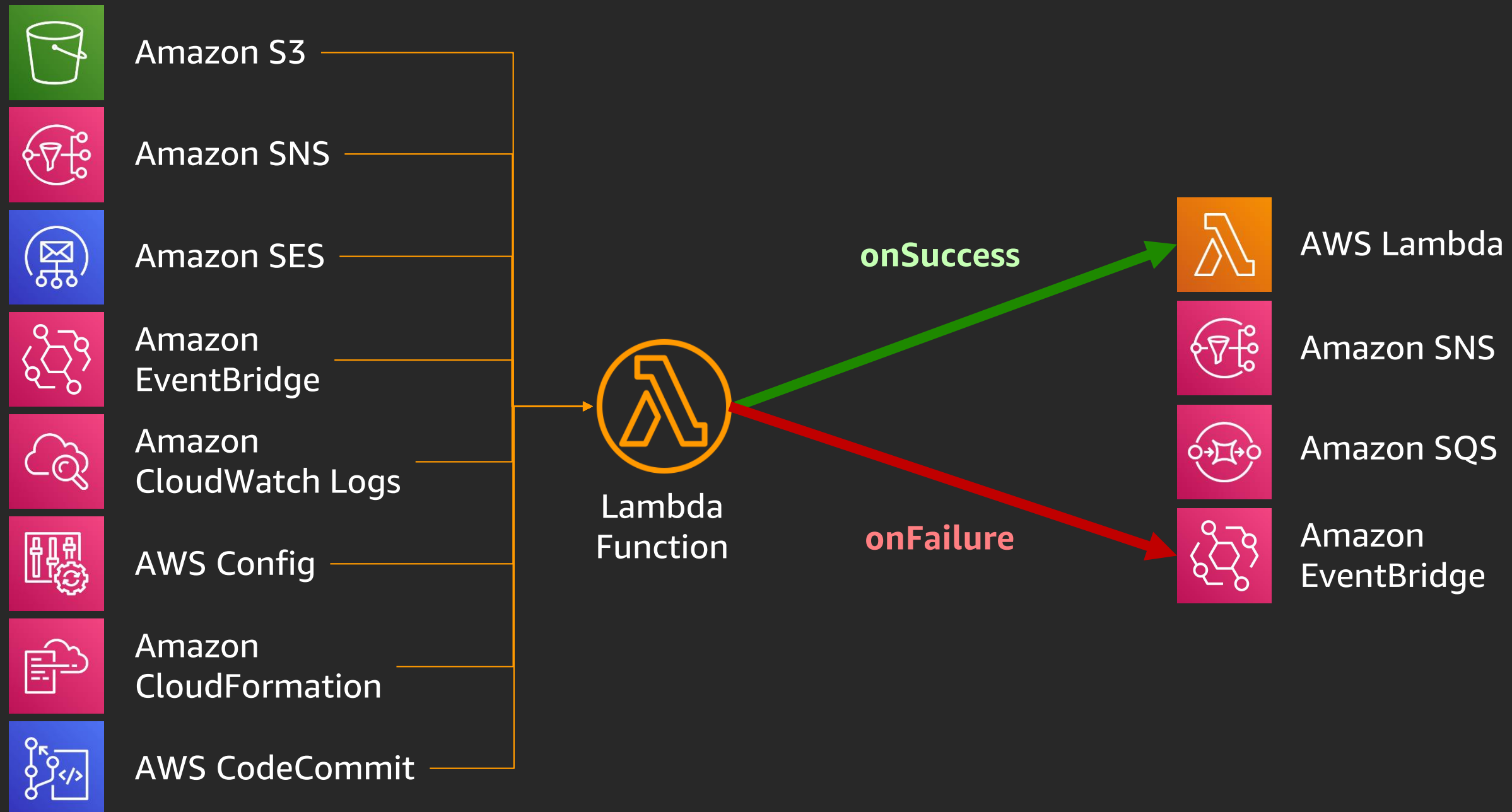
NEW!!!

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

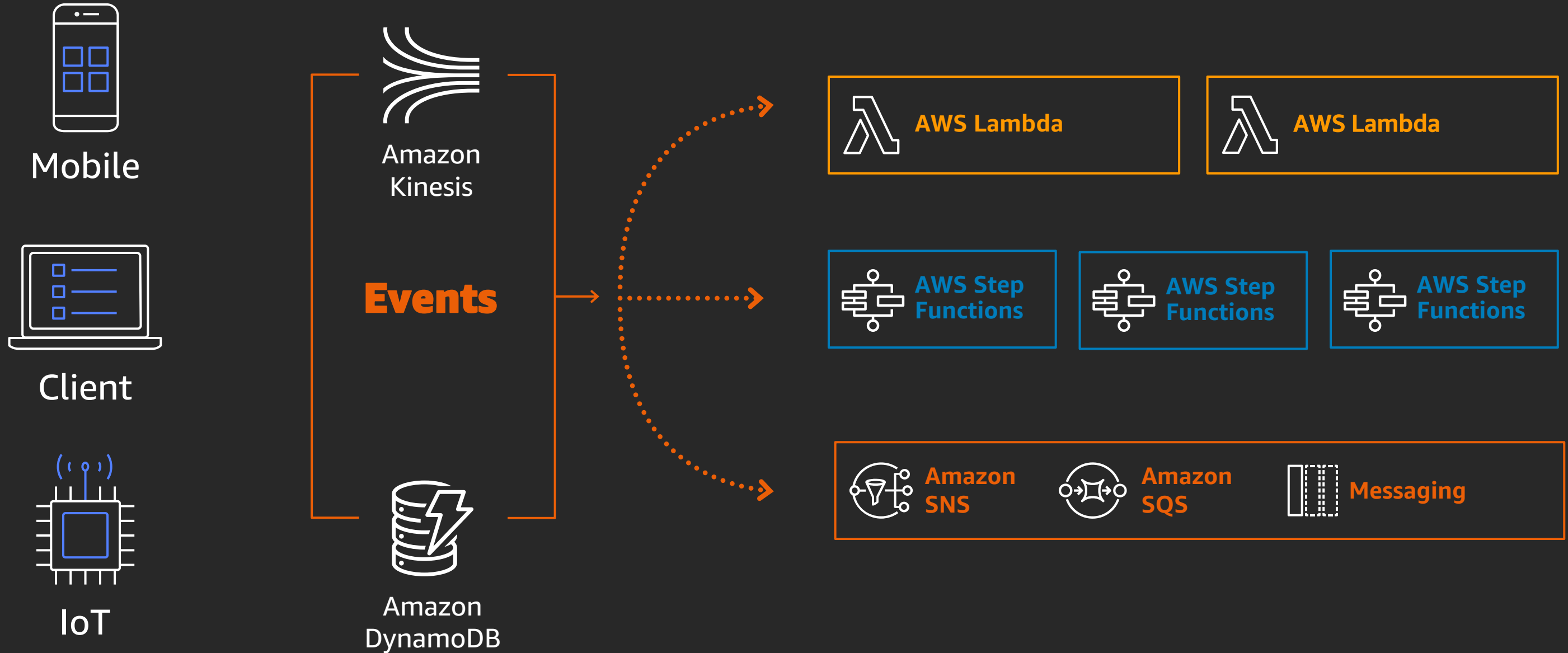


# AWS Lambda – Destinations for Async Invocations



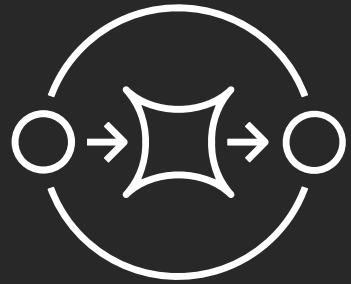


# Event-driven architectures



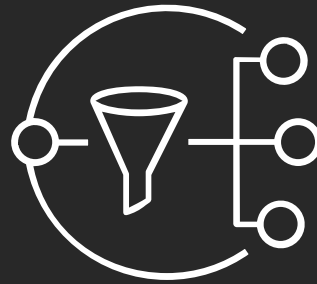
# Connecting AWS event sources

## Messaging



**Amazon SQS**

**Queues**



**Amazon SNS**

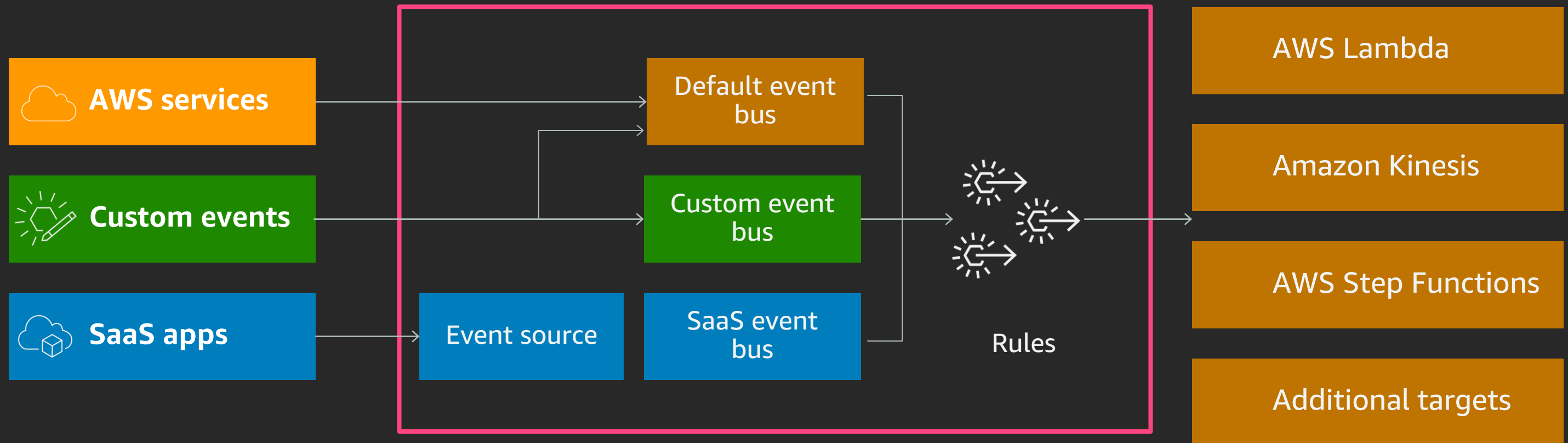
**Pub/Sub**



**Amazon  
EventBridge**

**Event Bus**

# Amazon EventBridge



# Amazon EventBridge

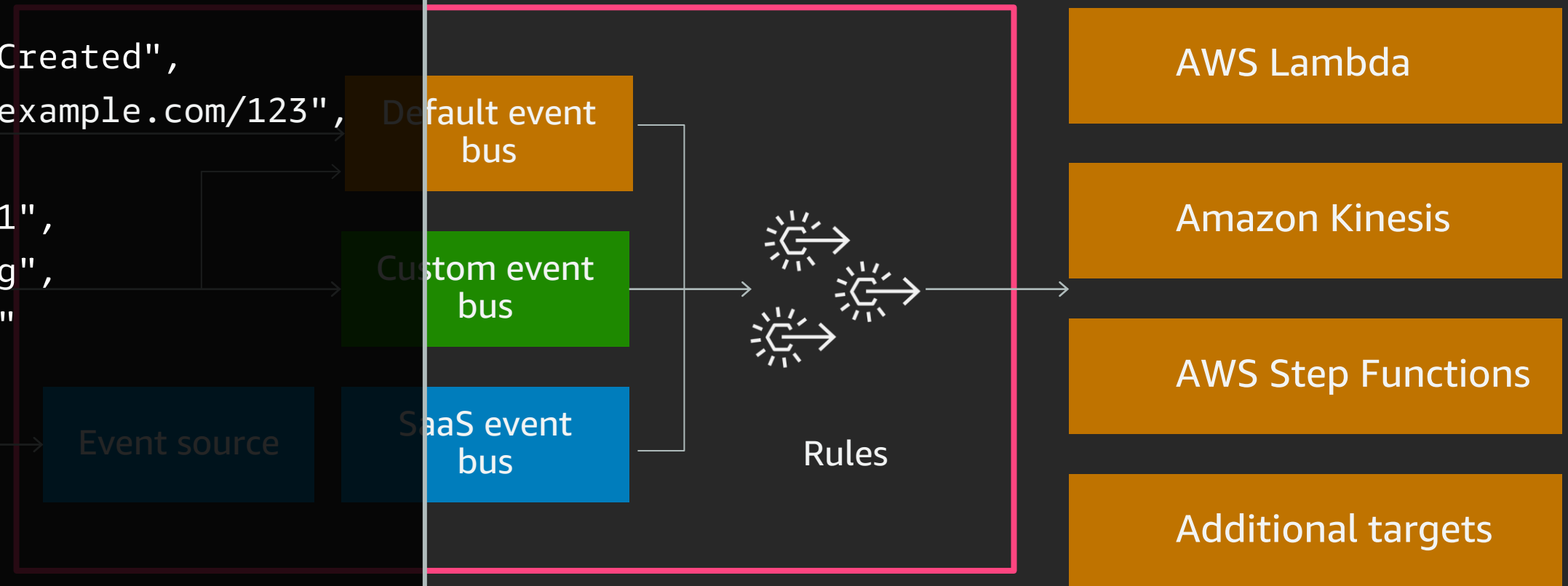
## Example event:

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

**AWS services**

**Custom events**

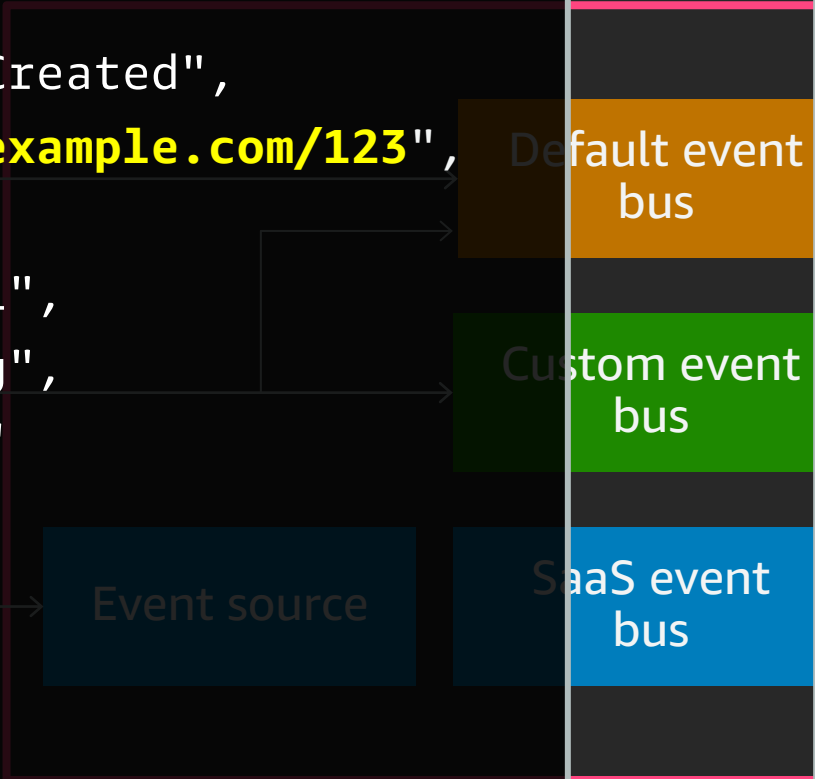
**SaaS apps**



# Amazon EventBridge

## Example event:

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



The diagram illustrates the flow of events from sources to buses. On the left, 'AWS services' and 'Custom events' are grouped. 'SaaS apps' is shown as an 'Event source'. Arrows point from 'AWS services', 'Custom events', and 'SaaS apps' to three event buses: 'Default event bus' (orange), 'Custom event bus' (green), and 'SaaS event bus' (blue). A red box highlights the 'source' field in the JSON event and the 'Default event bus'.

## Example rule:

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

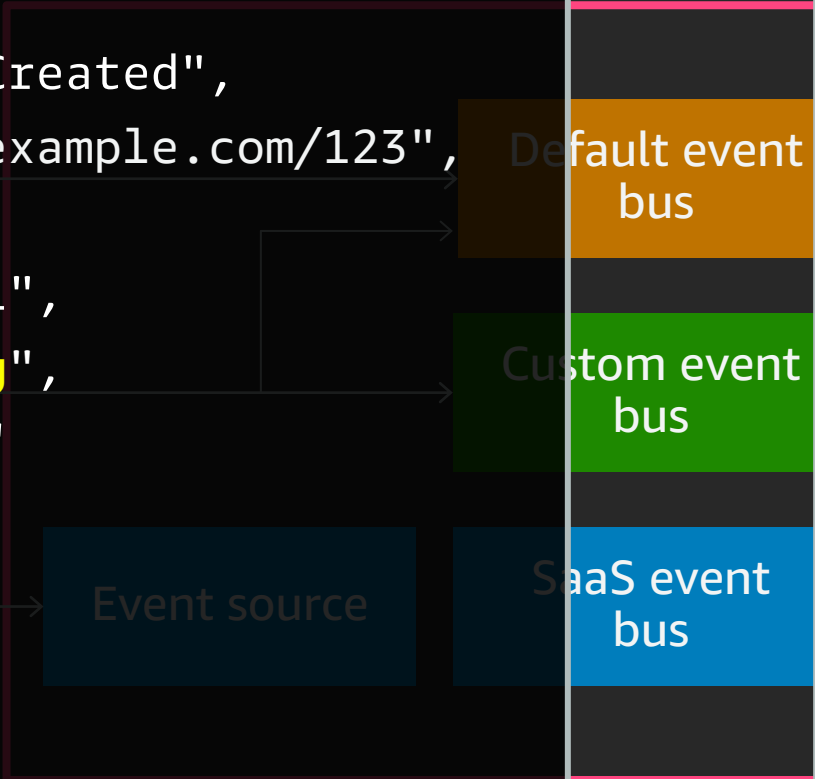


The diagram shows a rule being triggered by events from the 'Default event bus' and 'SaaS event bus'. The rule is represented by a gear icon with an arrow. It points to a 'Rules' box, which then points to a list of targets: 'Amazon Kinesis', 'AWS Step Functions', and 'Additional targets'. A red box highlights the 'source' field in the JSON rule and the 'Rules' box.

# Amazon EventBridge

## Example event:


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



The diagram illustrates the flow of events from sources to buses. On the left, 'AWS services' and 'Custom events' are grouped. Below them, 'SaaS apps' are shown with an 'Event source' box. Arrows indicate that events from AWS services, custom events, and SaaS apps flow into three event buses: 'Default event bus' (orange), 'Custom event bus' (green), and 'SaaS event bus' (blue). A red box highlights the 'Custom event bus' and the 'SaaS event bus'.

## Example rule:

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

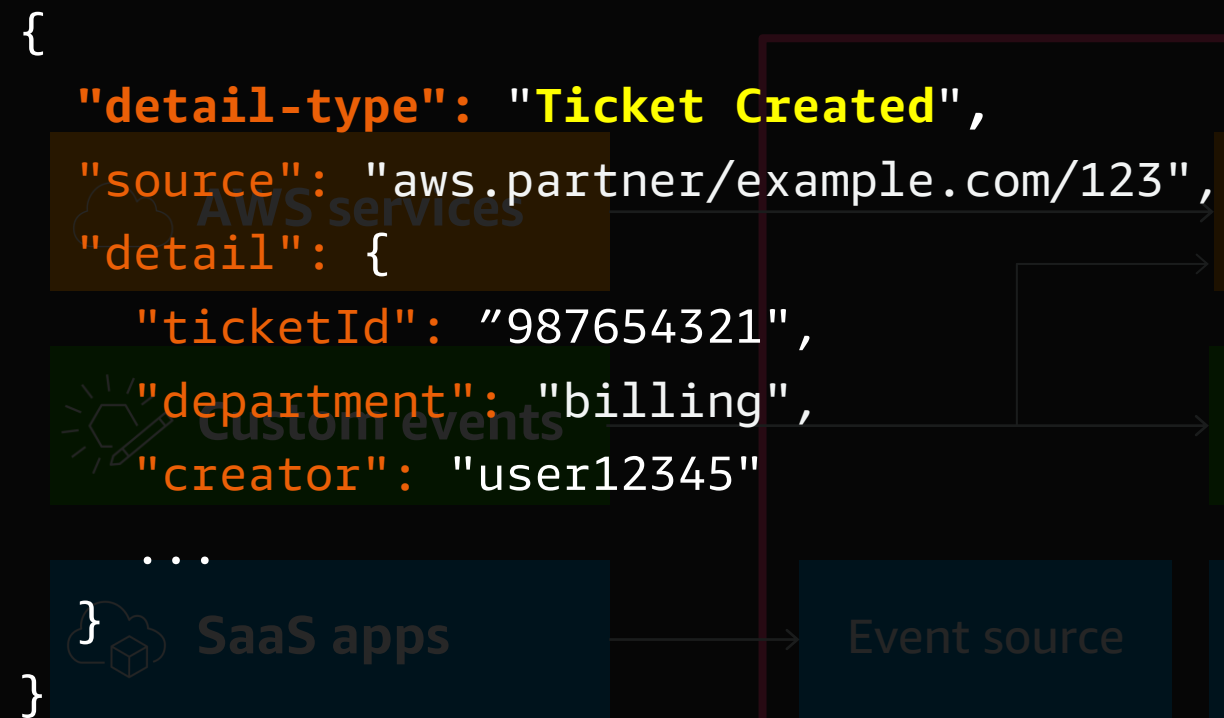


The diagram shows the flow from a rule to various targets. On the left, a 'Rules' box contains a gear icon and an arrow. An arrow points from the 'Rules' box to a list of targets: 'AWS Lambda', 'Amazon Kinesis', 'AWS Step Functions', and 'Additional targets'. A red box highlights the 'Rules' box and the 'Additional targets' box.

# Amazon EventBridge

## Example event:

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



The diagram illustrates the flow of an event from its source to various event buses. On the left, a box labeled 'SaaS apps' with a cloud and cube icon is connected by an arrow to a box labeled 'Event source'. From the 'Event source', three arrows point to three stacked boxes representing event buses: 'Default event bus' (orange), 'Custom event bus' (green), and 'SaaS event bus' (blue). A red rectangular box highlights the 'Event source' and the three event bus boxes.

## Example rule:

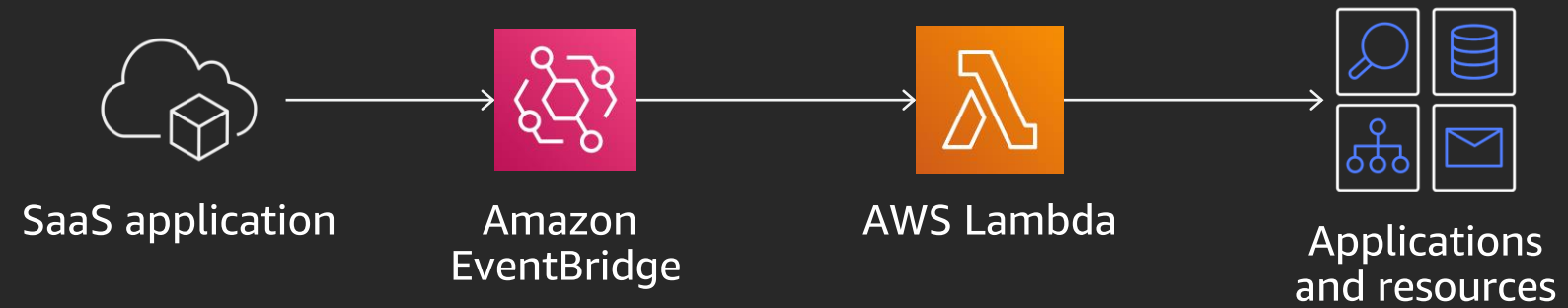
```
{  
  "detail-type": ["Ticket Resolved"]  
}
```



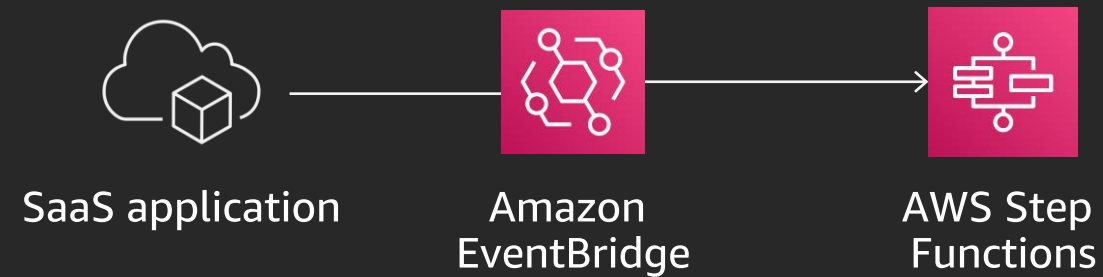
The diagram shows a rule configuration. A box on the left contains the JSON for a rule: `{ "detail-type": ["Ticket Resolved"] }`. An arrow points from this box to a central area labeled 'Rules' which contains three gear icons. From the 'Rules' area, an arrow points to a stack of four boxes representing targets: 'Amazon S3', 'Amazon Kinesis', 'AWS Step Functions', and 'Additional targets'.

# Common use cases

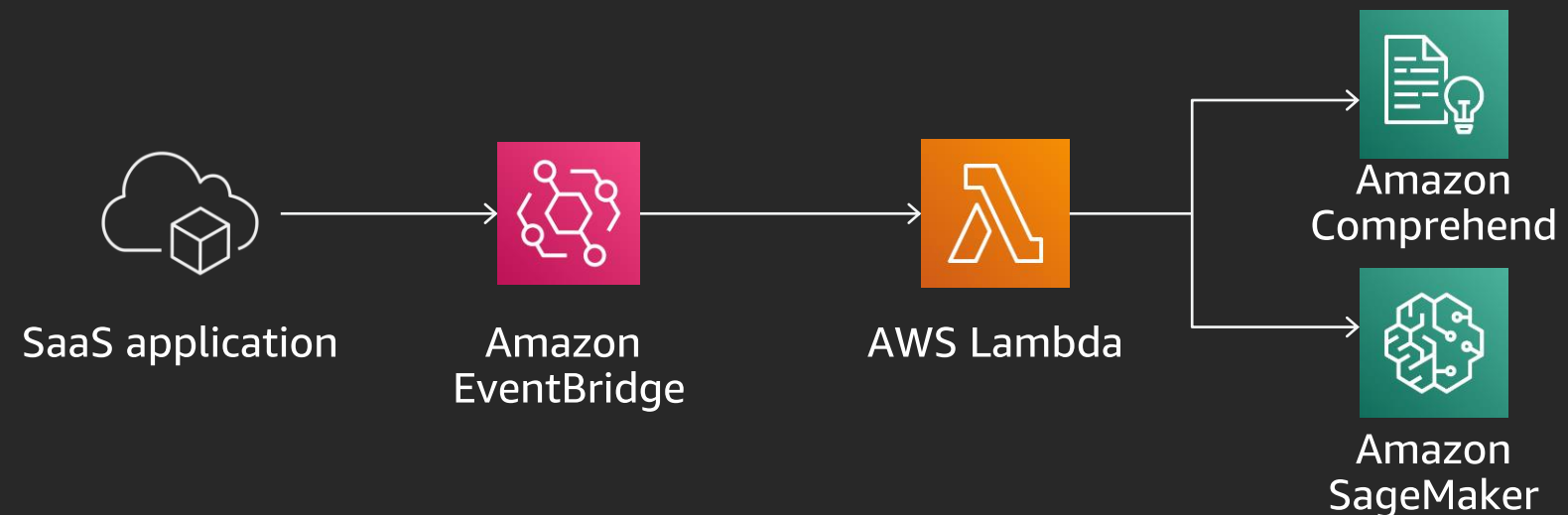
## Take action



## Run workflows

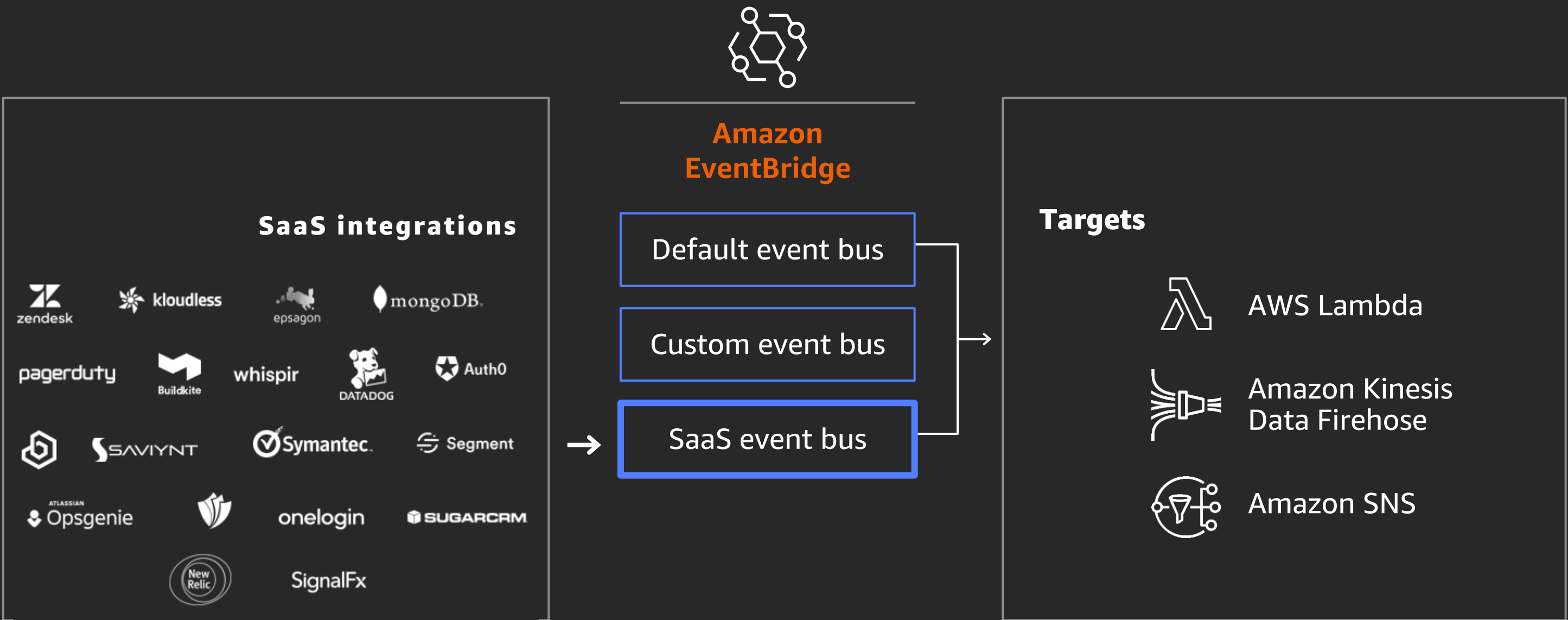


## Apply intelligence

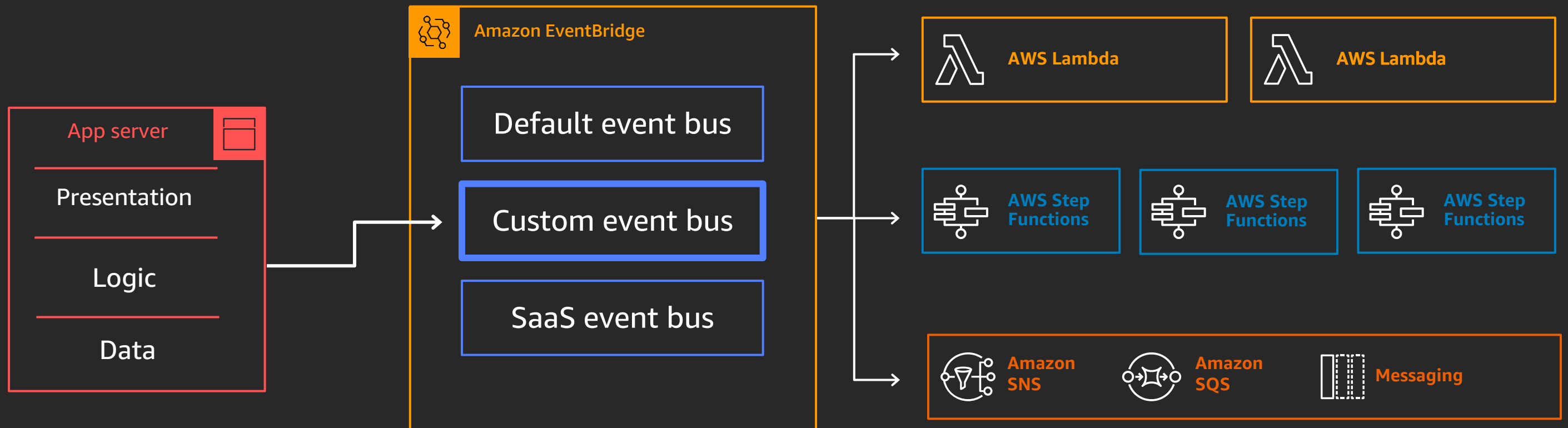




# Expanding event sources



# 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

The screenshot shows the LEGO.com website with a yellow header. The top navigation bar includes a menu icon, the LEGO logo, and user account icons (a person, a heart, and a shopping bag) with a count of 0. Below the header is a search bar with the placeholder text "Search...". The main content area features a row of six category tiles: "New" (with a helicopter image), "Exclusives" (with a ship image), "Promotions" (with a Christmas tree image), "SERIOUS PLAY®" (with a box image), "VIP" (with a blue minifigure image), and "Pick a Brick" (with two bricks). Below this is a section titled "Our top-selling exclusives" which displays three featured products: "Home sweet home" (a gingerbread house), "New exclusive Disney Train and Station" (a Disney train), and "Travel the LEGO® galaxy in the ultimate Millennium Falcon™!" (the Millennium Falcon). Each product tile includes an image, a title, a short description, and a "Shop now >" link.

MENU

LEGO

0 0

Search...

New Exclusives Promotions SERIOUS PLAY® VIP Pick a Brick

Our top-selling exclusives

**Home sweet home**  
Spread holiday cheer with the exclusive Gingerbread House!  
[Shop now >](#)

**New exclusive Disney Train and Station**  
Celebrate the magic of Disney and trains with this new set.  
[Shop now >](#)

**Travel the LEGO® galaxy in the ultimate Millennium Falcon™!**  
Take on the challenge of the largest, most detailed LEGO® Star Wars™ Millennium Falcon™ model we've ever created!  
[Shop now >](#)

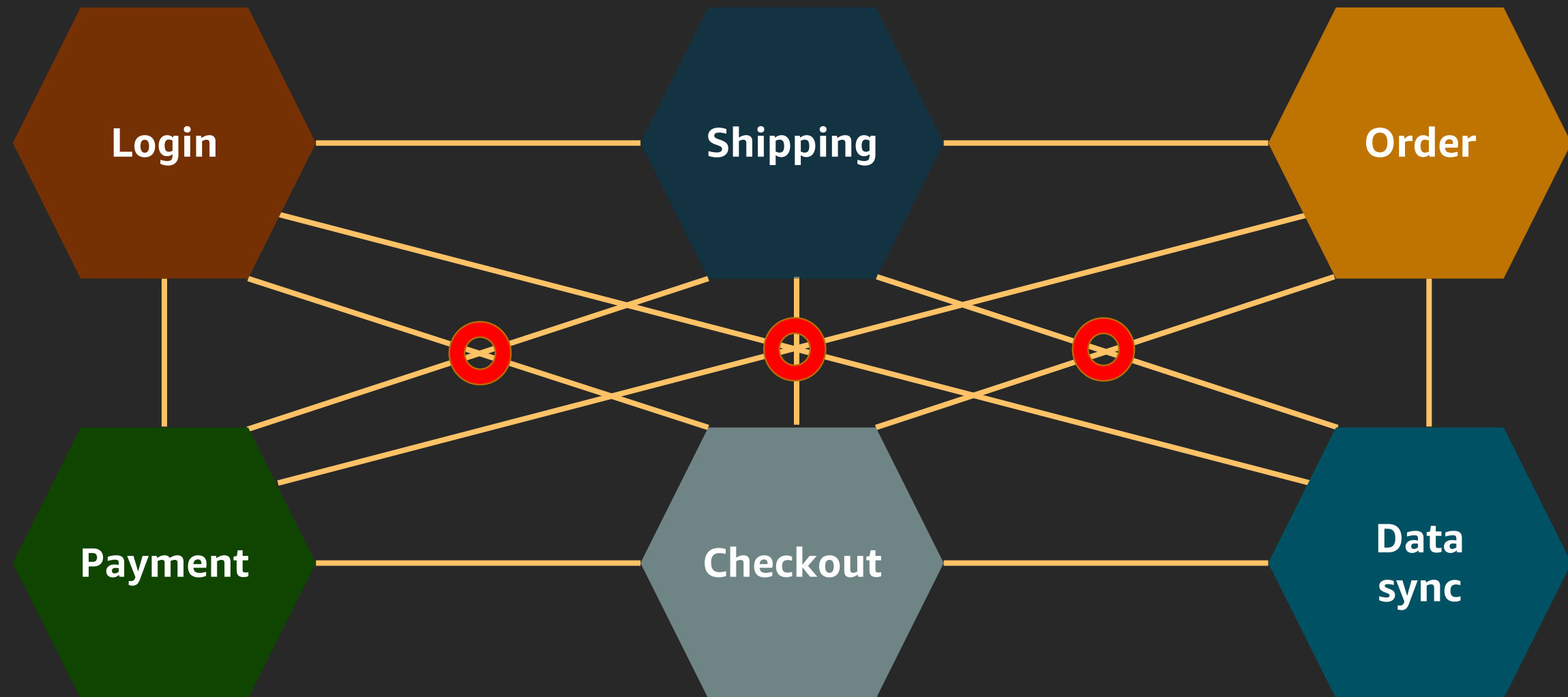
July 11, 2019



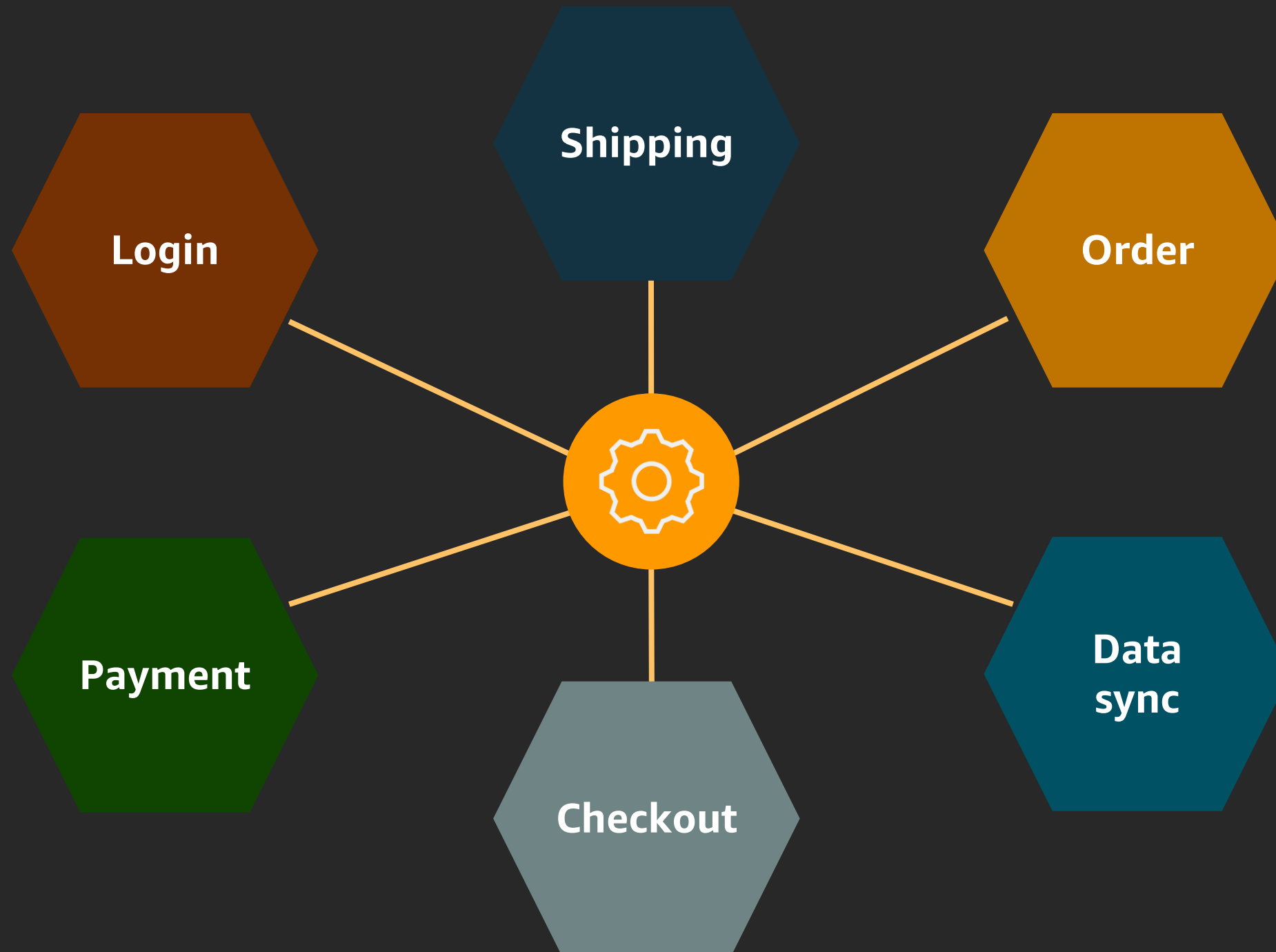
The day after,  
**Amazon EventBridge**  
was launched



# Checkout event processing

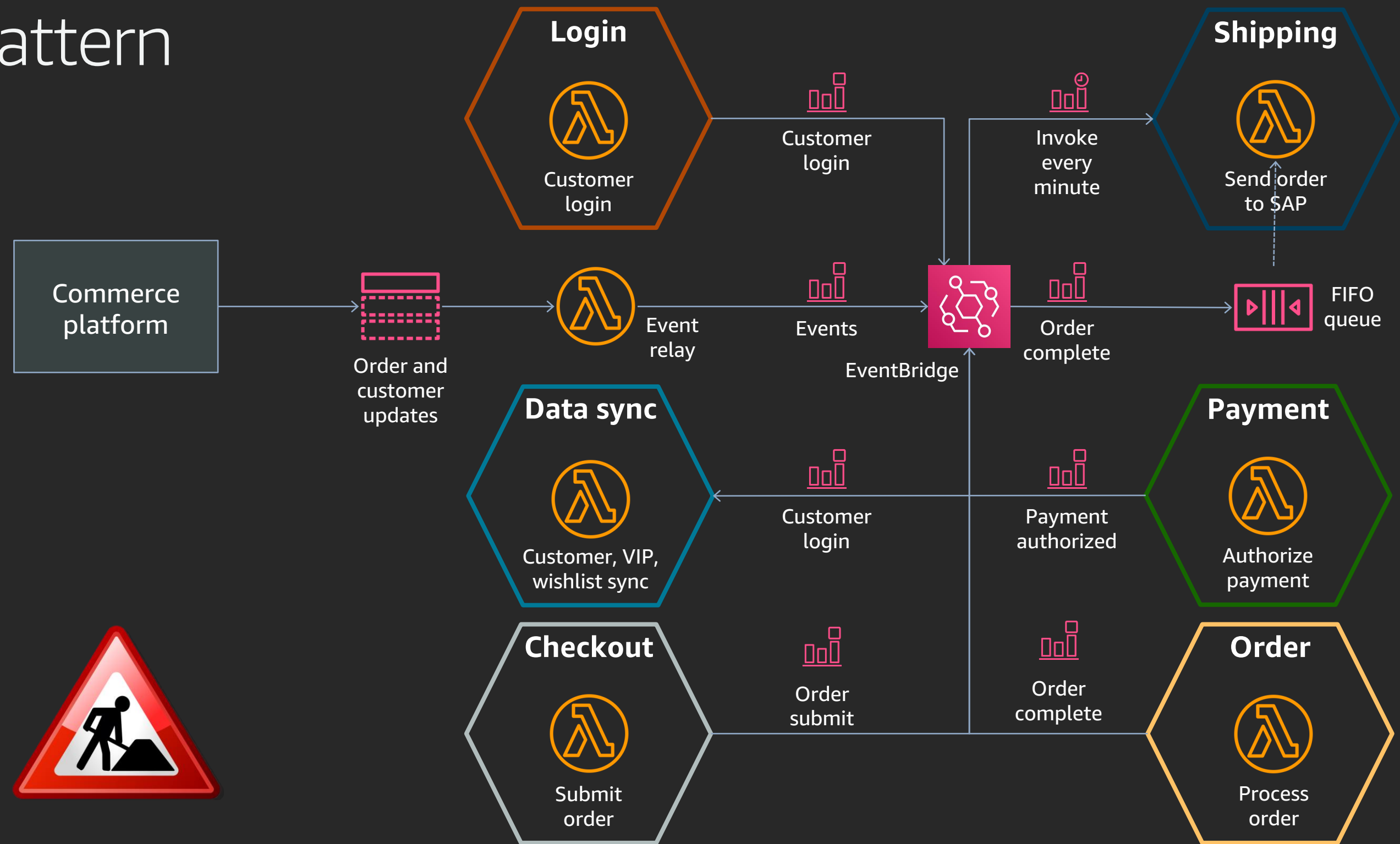


# Hub-and-spoke event bus

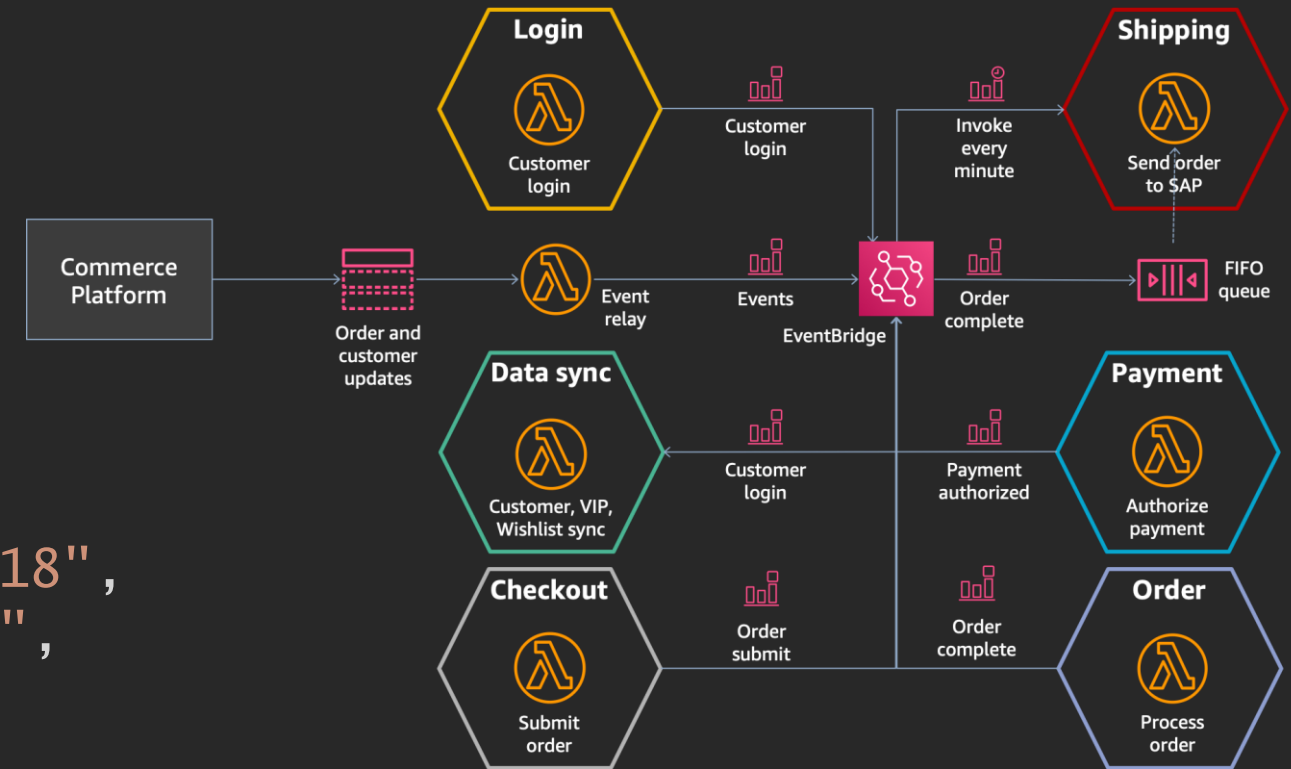




# Pattern



# Pattern – Hub-and-spoke event bus



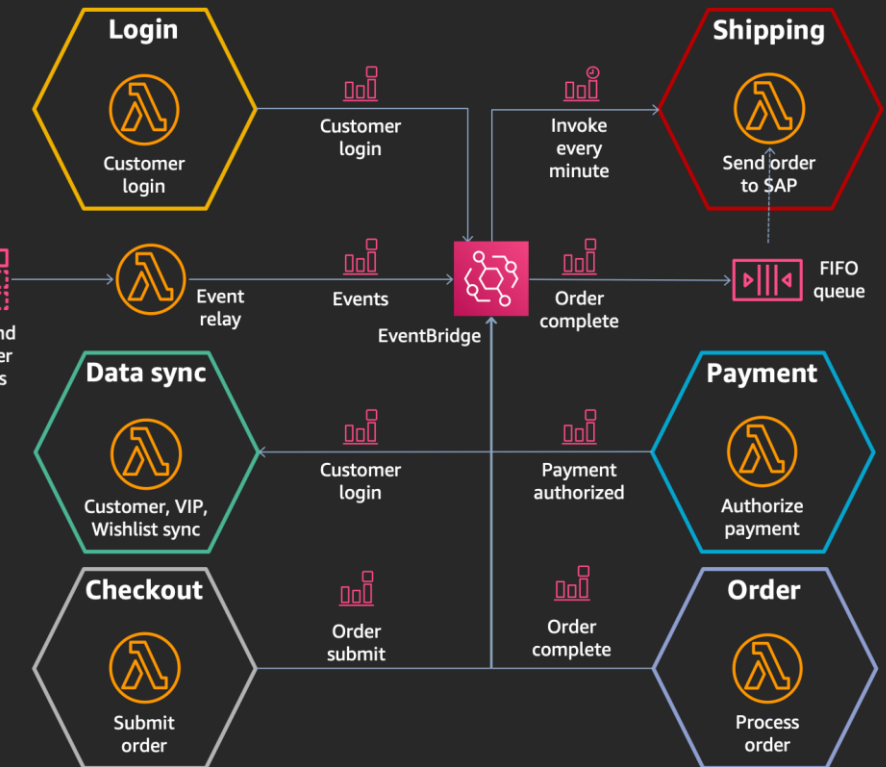
```
{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "State change Notification",
  "source": "service-order-submit-dev",
  "account": "111122223333",
  "time": "2019-08-29T12:10:21Z",
  "region": "eu-central-1",
  "resources": [
    "arn:aws:events:event-bus/checkout-bus"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

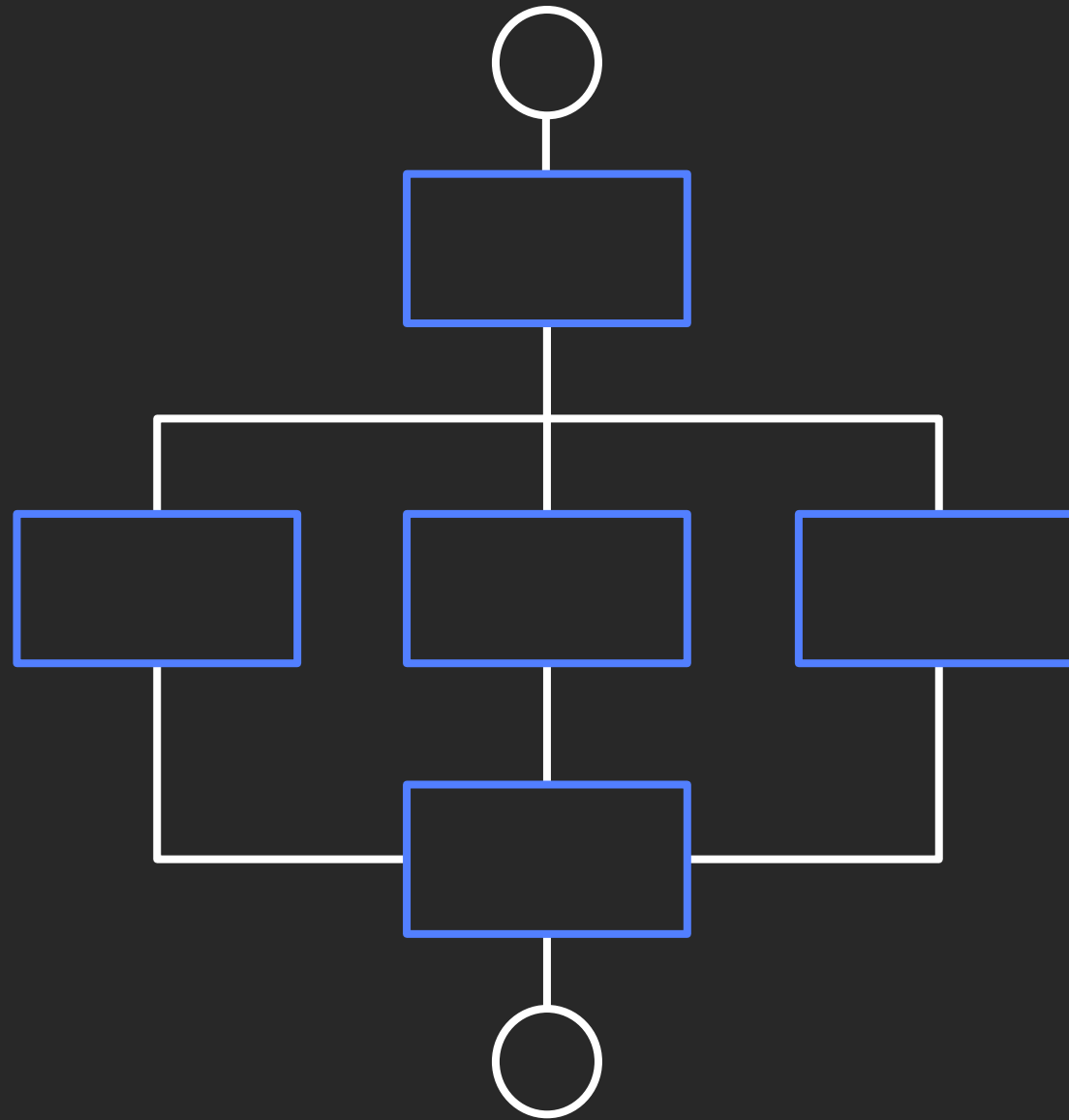


Custom for each  
service

# Orchestration with AWS Step Functions

# Coordinate function execution

Track status  
of data and  
execution



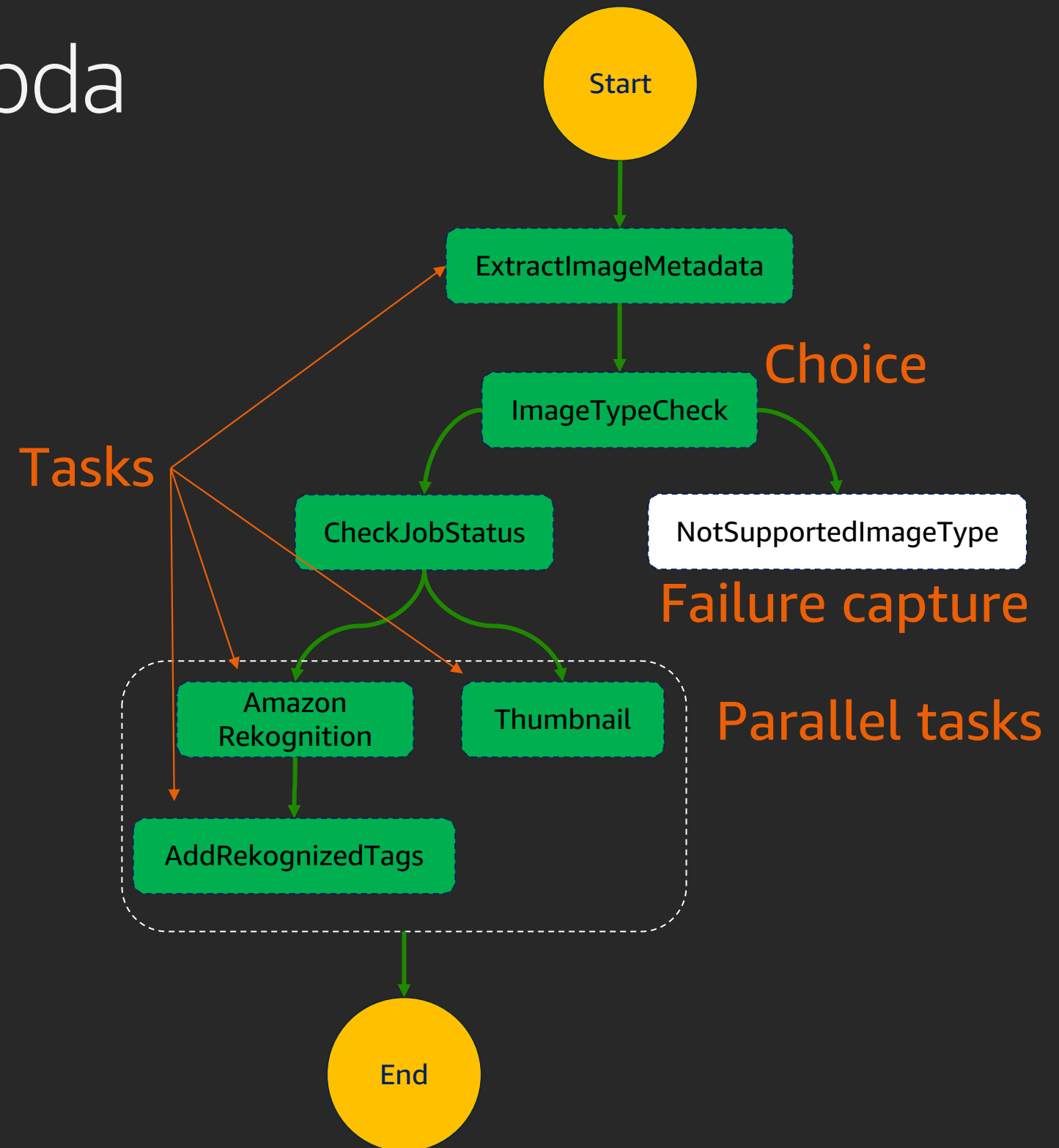
Remove  
redundant code

# 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!



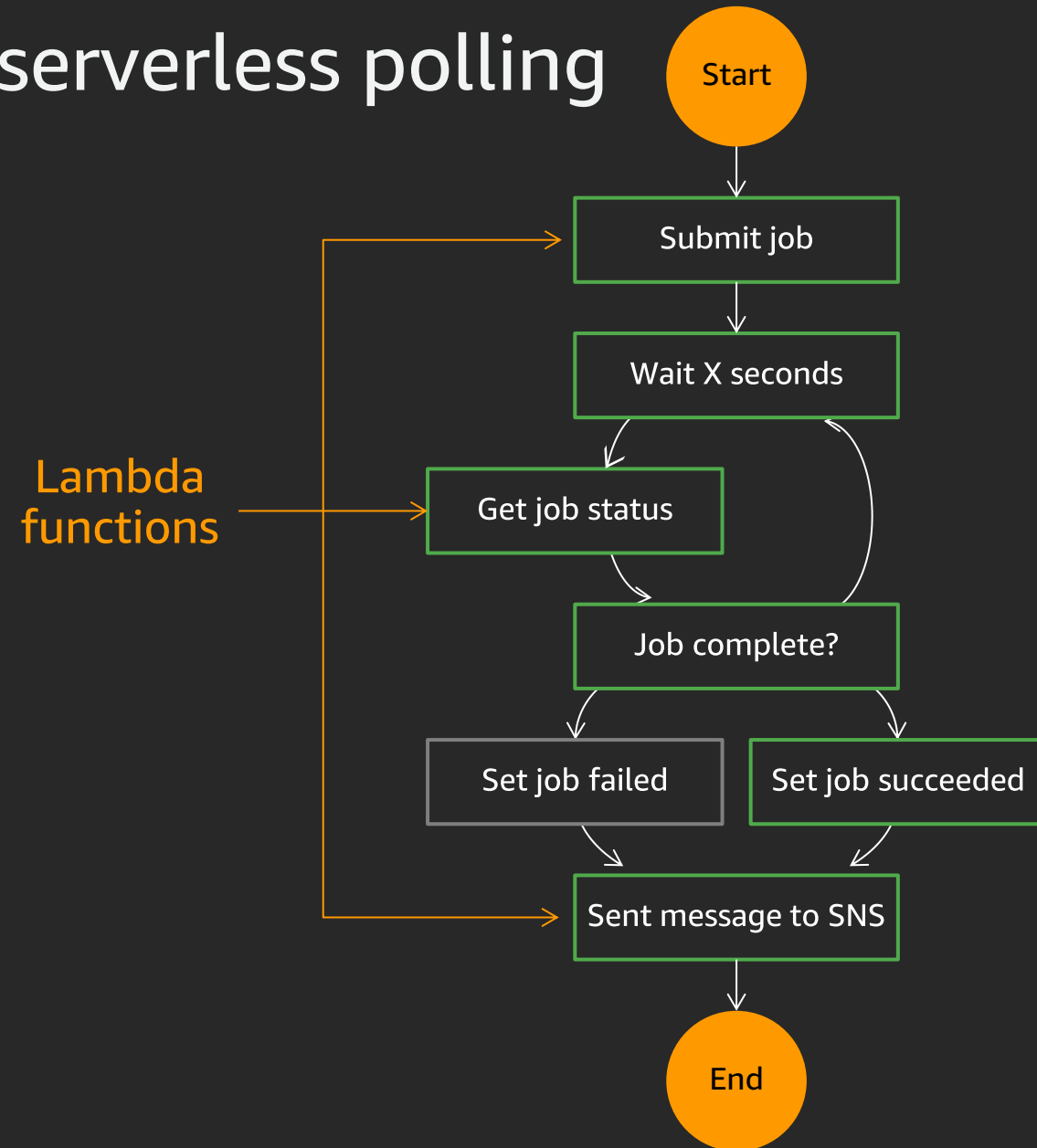
Amazon EMR

NEW!!!



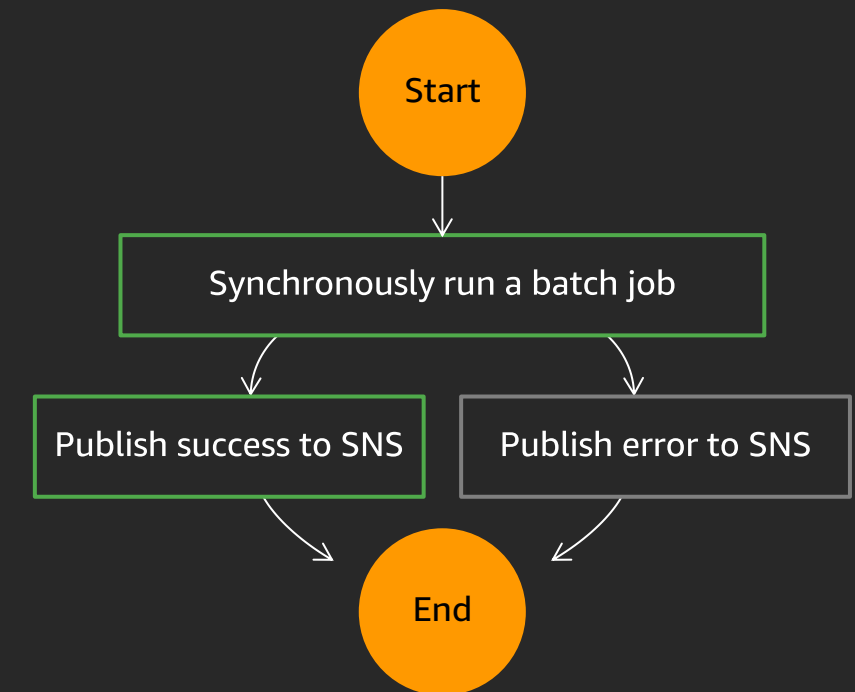
# Simpler integration, less code

With serverless polling

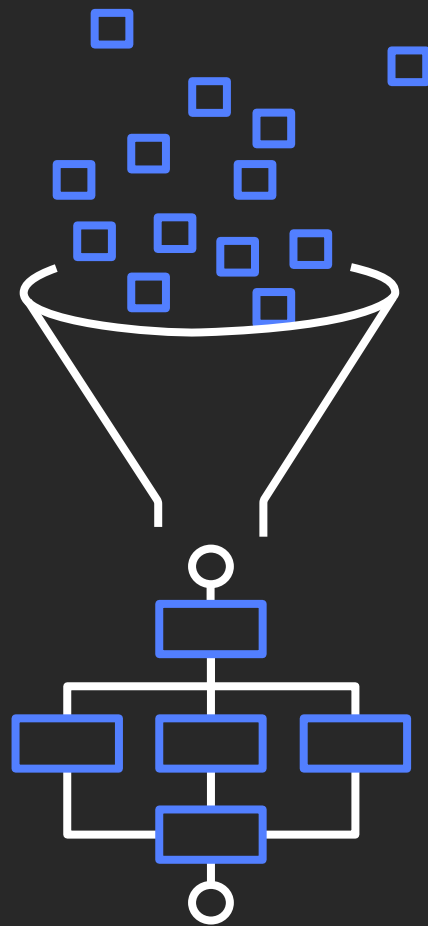


With direct service integration

No  
Lambda  
functions



# Express workflows



---

**Event streams over 100,000 per second**

---

**High volume, short duration**

---

**Cost effective at scale**

---

**Streaming data processing –  
IoT ingestion**

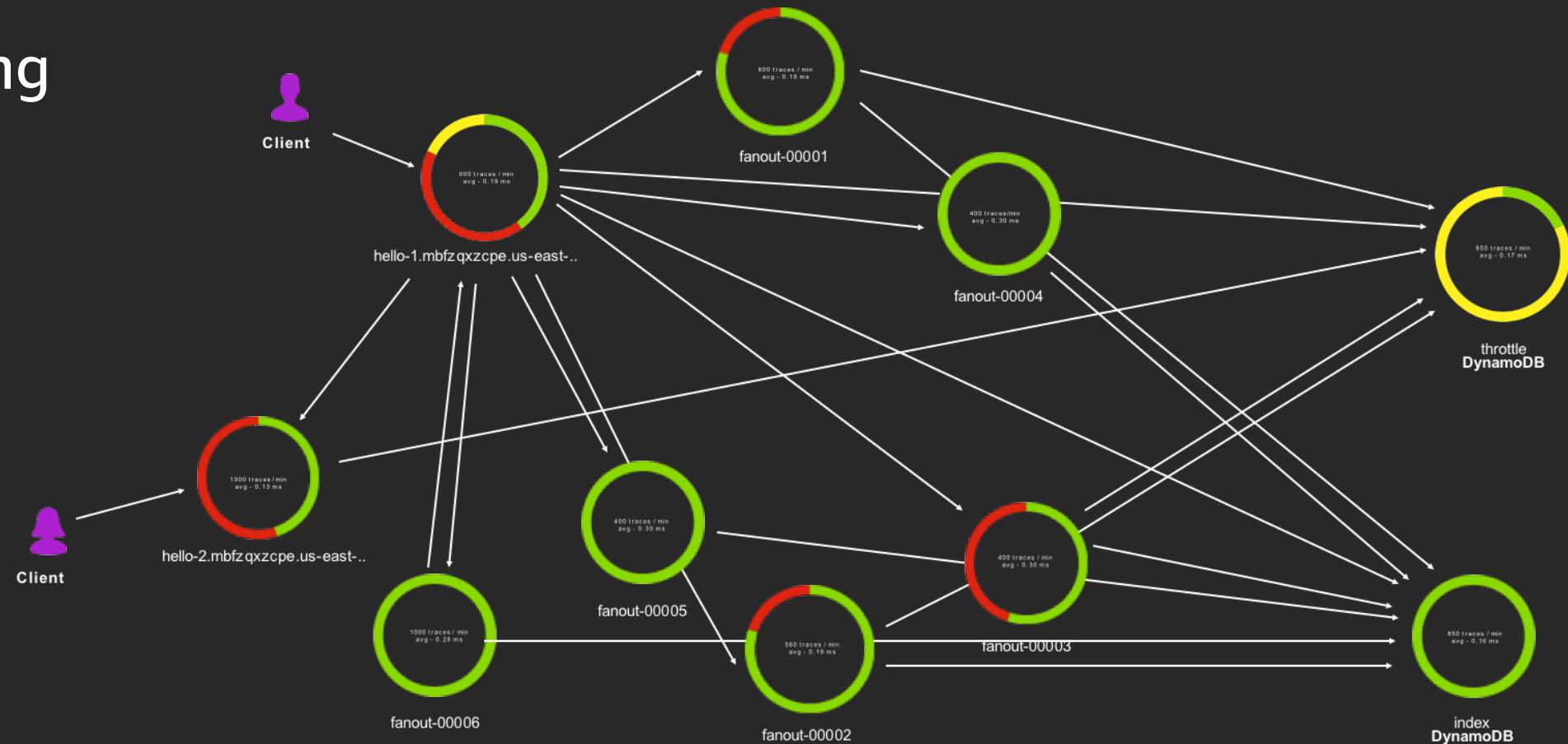
# Tracing and observability

# 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();
```



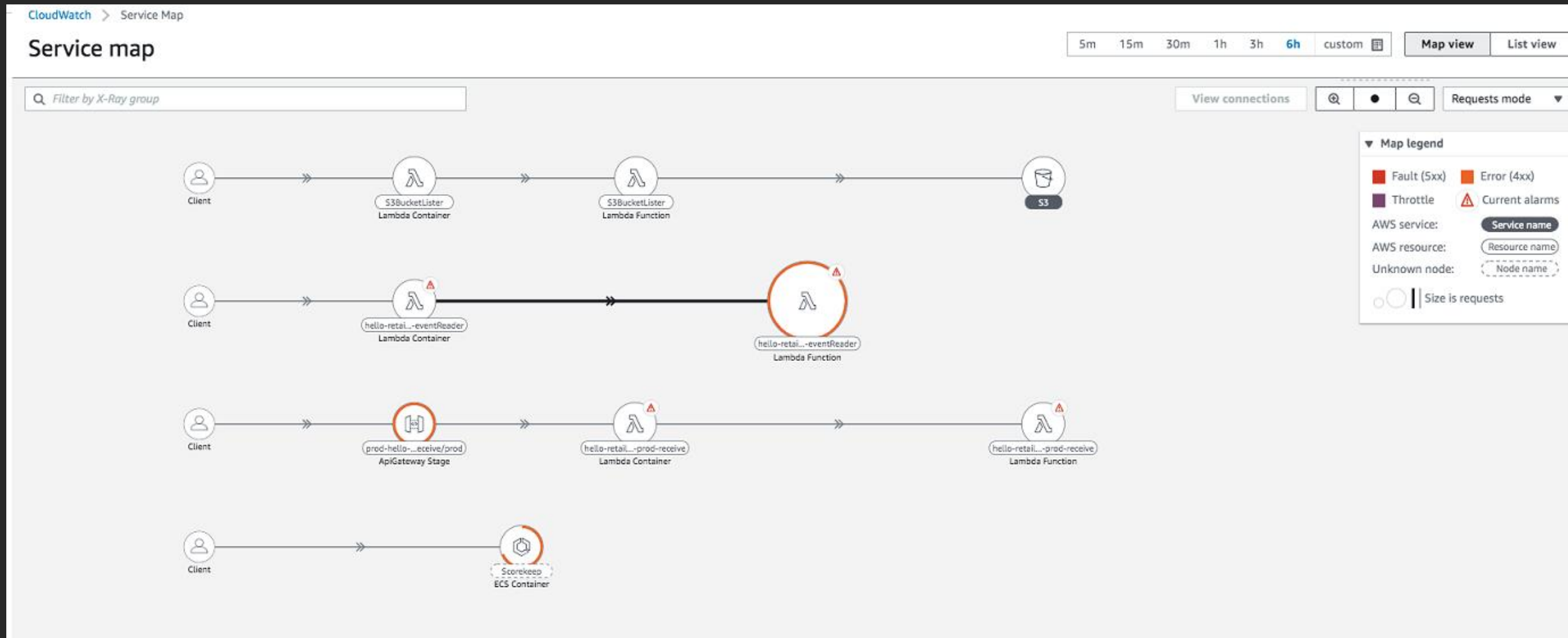
Enable X-Ray Tracing ☒ [Info](#)

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 function with

Using the `metricScope` decorator without function parameters

```
const { metricScope, Unit } = require("aws-embedded-metrics");

const myFunc = metricScope(metrics =>
  async () => {
    metrics.putDimensions({ Service: "Aggregator" });
    metrics.putMetric("ProcessingLatency", 100, Unit.MILLISECONDS);
    metrics.setProperty("RequestId", "422b1569-16f6-4a03");
    // ...
  });

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

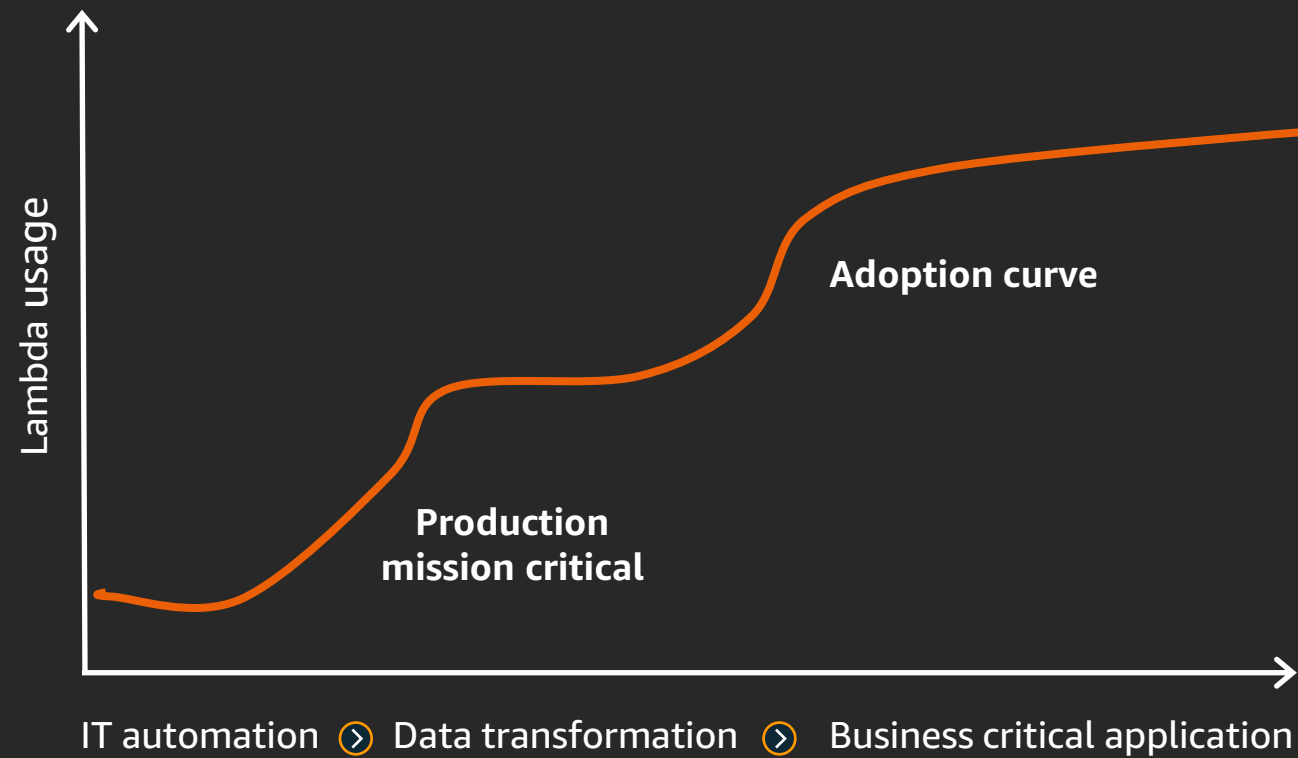
@metric_scope
def my_handler(metrics):
    metrics.put_dimensions({"Foo": "Bar"})
    metrics.put_metric("ProcessingLatency", 100, "Milliseconds")
    metrics.set_property("AccountId", "123456789012")
    metrics.set_property("RequestId", "422b1569-16f6-4a03")
    metrics.set_property("DeviceId", "61270781-c6ac-46f1")

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

# What now?

# Finding success

## Organic



## Serverless first



### Considerations

- Rapid development
- Time-to-market – agility
- Changing business



# More tooling available



# Thank you!

Kim Kao

 @Yikaikao  Kimkao.solid