Building event-driven architectures on AWS

Peterson Larentis

Sr. Specialist Solutions Architect, Serverless - LATAM
AWS

06-08 DEZEMBRO
TDC FUTURE
PORTO ALEGRE EXPERIENCE
O PAPEL DA TECNOLOGIA NA CONSTRUÇÃO DO AMANHÃ

Who AMI?

/peterson-larentis

Agenda

Challenges with distributed systems

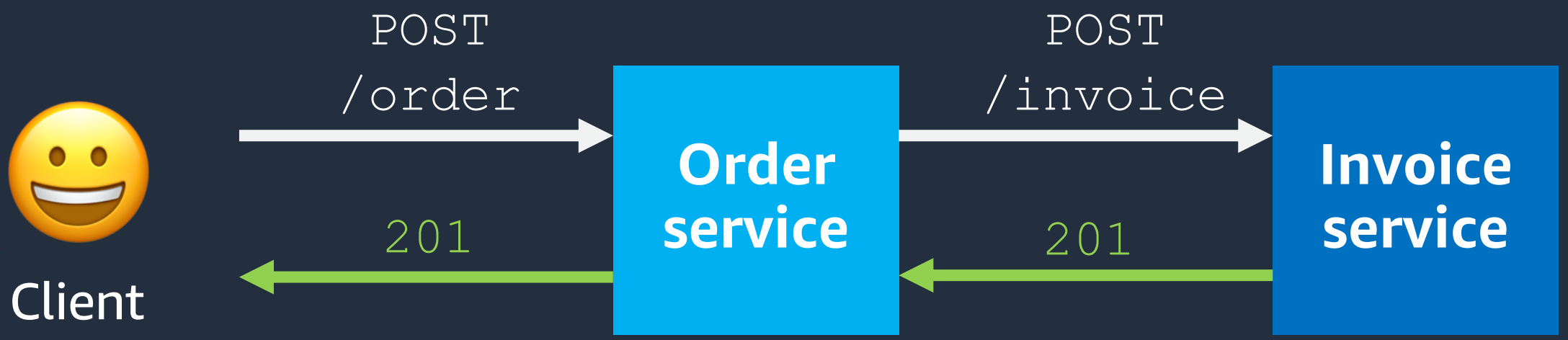
Journey to event-driven architectures

Design considerations

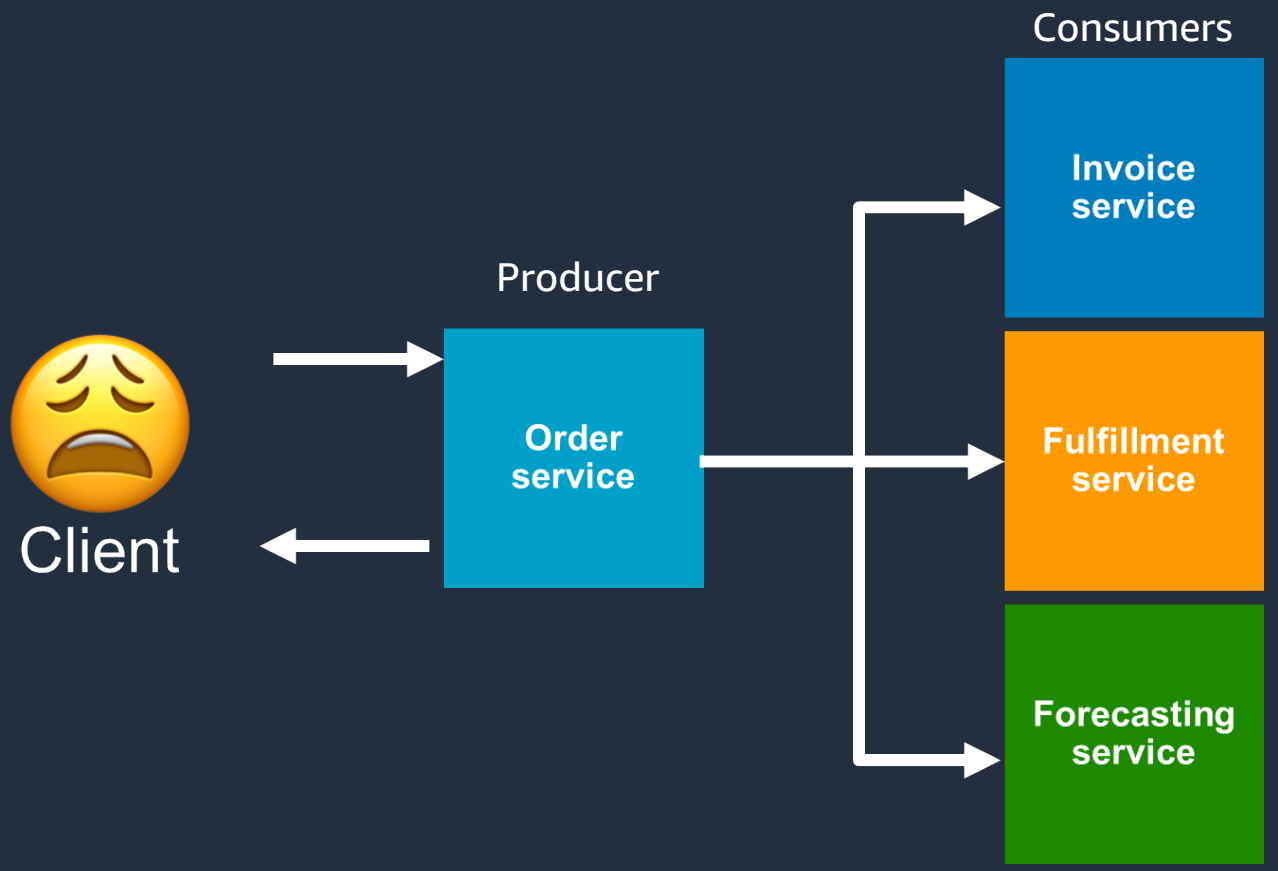
Next steps

Challenges with distributed systems

Microservices start simple



Synchronous API challenges over time



Coupling between producers and consumers

Multiple points of failure

Varying degree of quality of service

External dependencies

Coupling – Integration's magic word

Coupling is a measure of independent variability between connected systems.

Decoupling has a cost, both at design and run-time.

Coupling isn't binary.

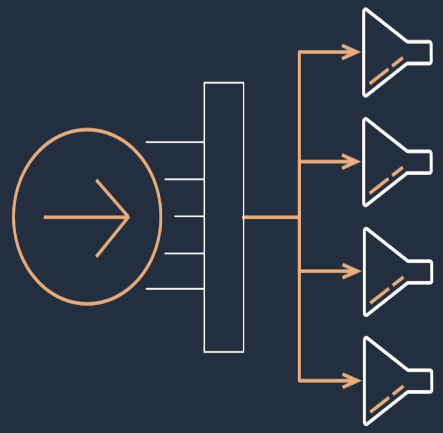
Coupling isn't one-dimensional.

Source: EnterpriseIntegrationPatterns.com

The many facets of coupling

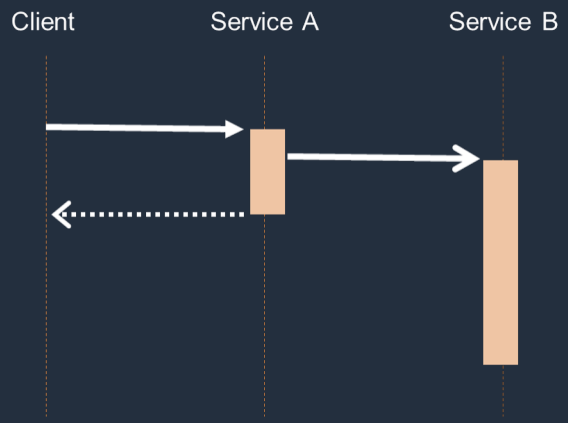
Temporal dependency:	sync, async
Location dependency:	IP addresses, DNS
Data format dependency:	Binary, XML, JSON, ProtoBuf, Avro
Semantic dependency:	Name, Middlename, ZIP
Data type dependency:	int16, int32, string, UTF-8, null, empty
Interaction style dependency:	messaging, RPC, query-style (GraphQL)
Conversation dependency:	pagination, caching, retries
Technology dependency:	Java vs. C++

Event-driven architectures drive reliability and scalability



Event Routers

Abstract producers and consumers from each other



Asynchronous Events

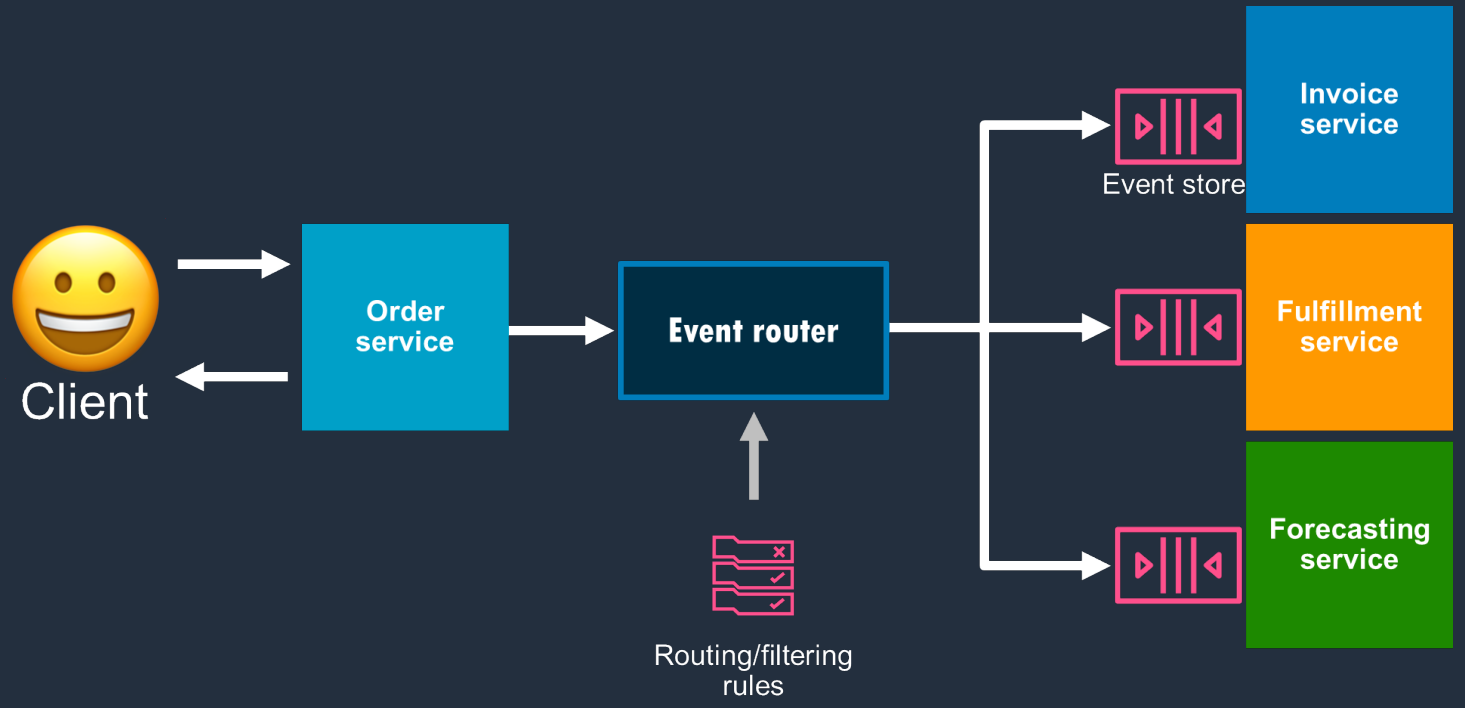
Improve responsiveness and reduce dependencies

Event Stores

Buffer messages until services are available to process

aws modern apps

Reliable, resilient, and independently scalable



"If your application is cloud-native, or large-scale, or distributed, and doesn't include a messaging or event component, that's probably a bug."

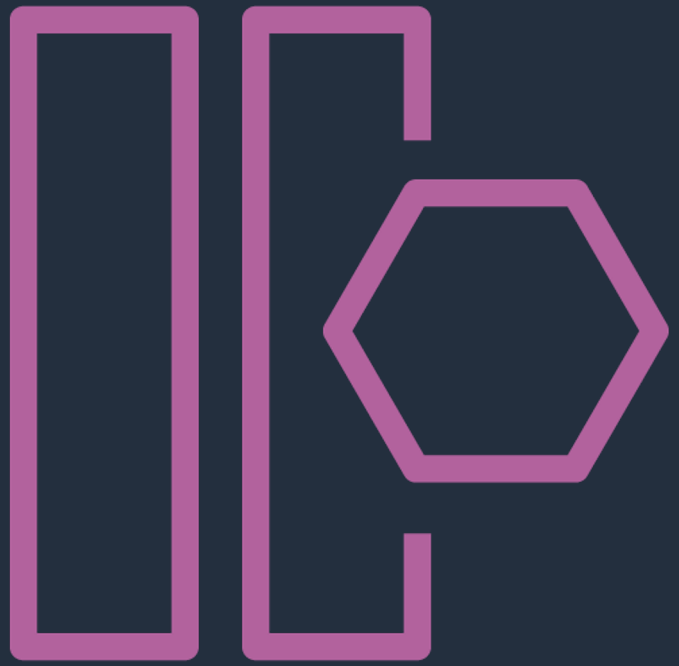
Tim Bray
General-purpose Internet-software geek

Journey to event-driven architectures

Journey to event-driven architectures

STEP 1

Start with the domain



event

[i-’vent] noun

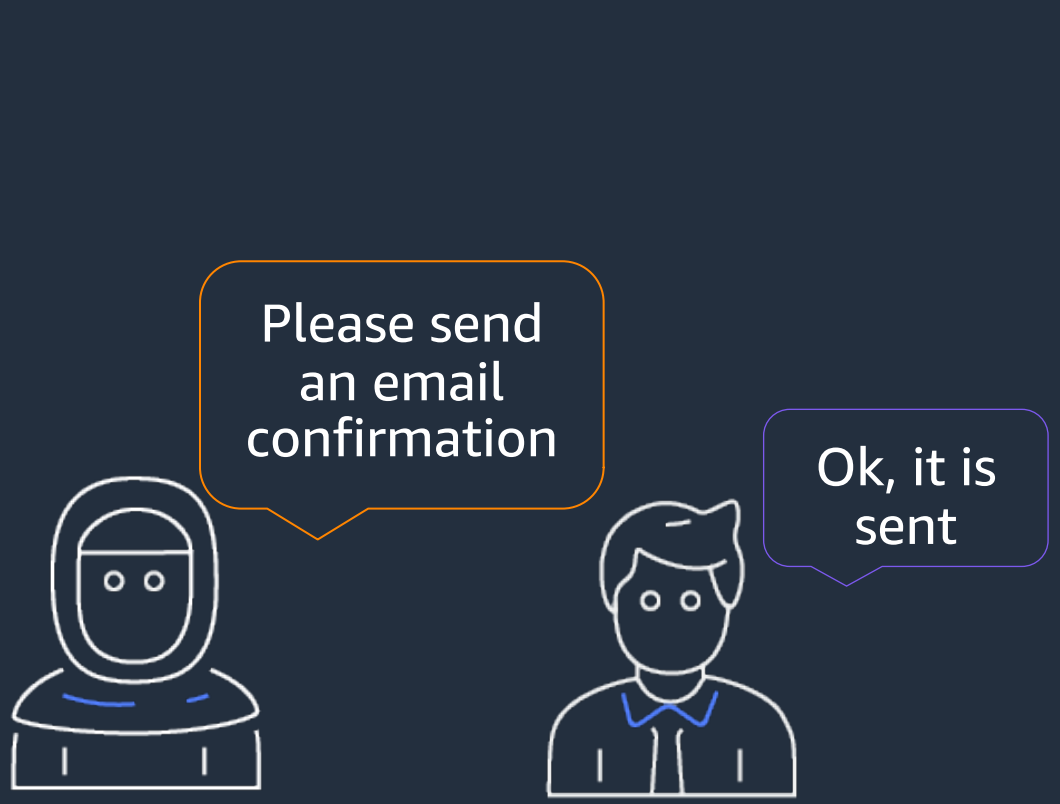
A signal that a system’s state has changed.

Properties of events

```
{
  "source": "com.orders",
  "detail-type": "OrderCreated",
  "detail": {
    "metadata": {
      "idempotency-key": "c1b95b88"
    },
    "data": {
      "order-id": "1073459984"
    }
}
```

- Events are signals that a system's state has changed
- Events occur in the past (e.g. OrderCreated)
- Events cannot be changed (immutable)
- Decrease semantic coupling by restricting information to key data

Events are observable, not directed

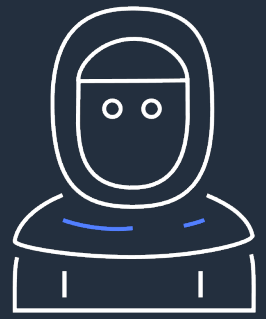


Directed commands

Observable events

Event-driven architectures start with business events

OrderCreated



OrderPicked

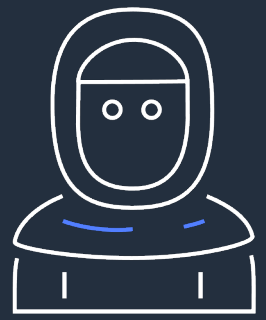
OrderShipped

ReturnRequested

ReturnReceived

Map events to business domains using attributes, not entities

OrderCreated



Retail

OrderPicked

OrderShipped

Fulfillment

ReturnRequested

ReturnReceived

Support

Event-driven modeling

1. Identify business events, processes, actors, etc.

2. Cluster-related concepts

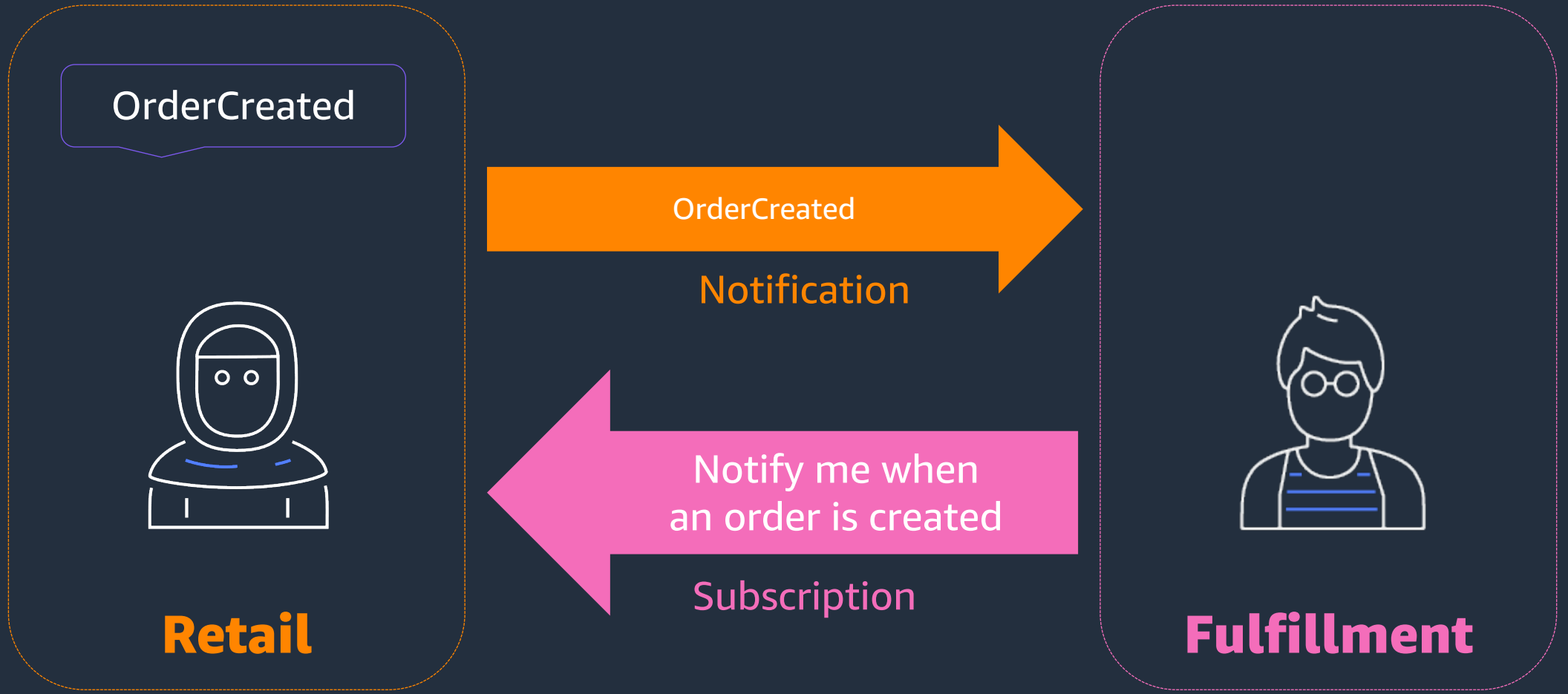
3. Define bounded contexts and sub-domains

Journey to event-driven architectures

STEP 1
Start with the domain

STEP 2
Coordinate events

Choreograph events *between domains* using subscriptions

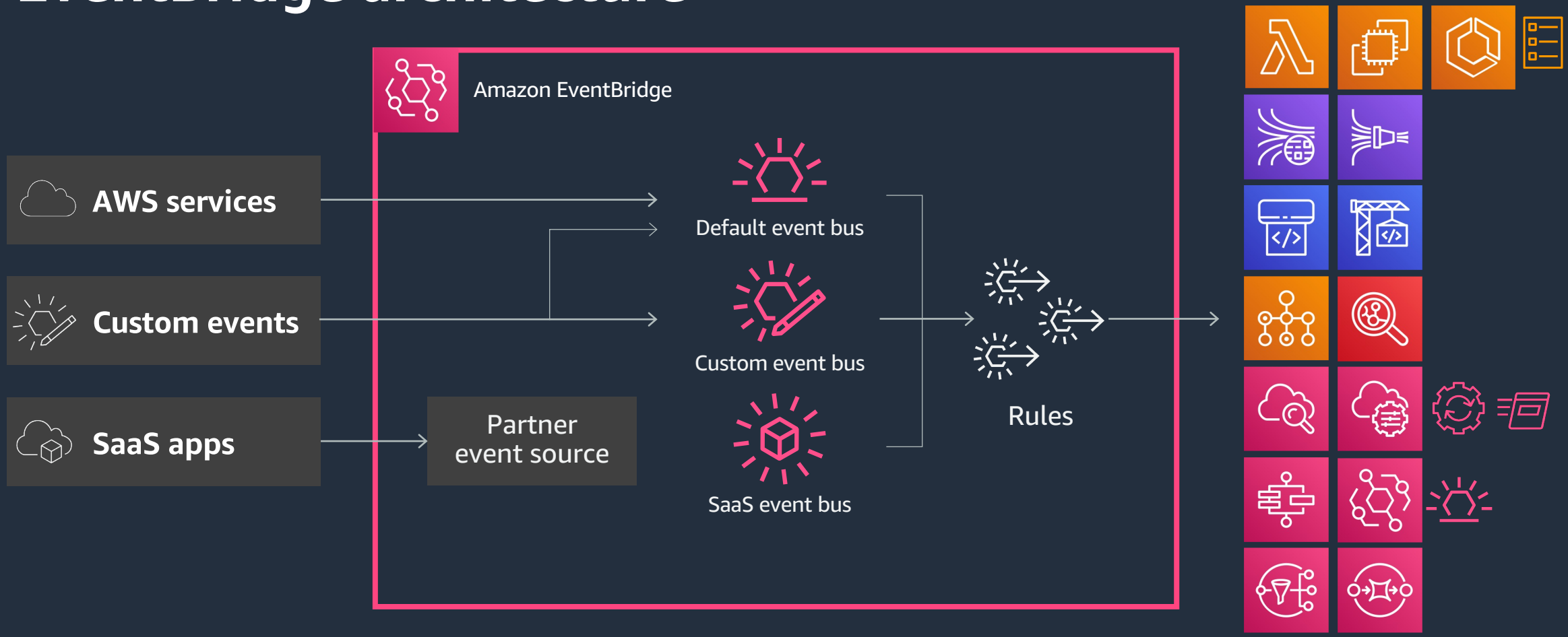


Amazon EventBridge

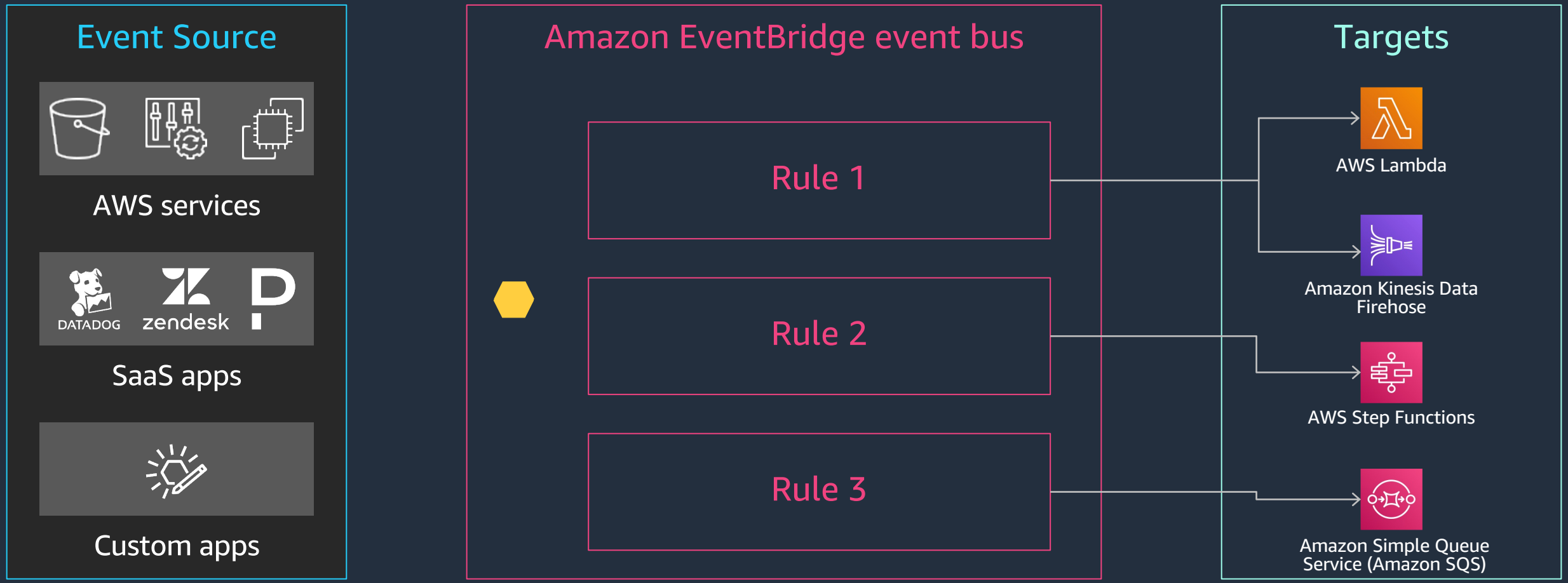
A serverless event bus service for AWS services, your own applications, and SaaS providers.

- Serverless. Pay only for the events you process
- Simplified scaling - avoids increasing costs to sustain and manage resources
- No upfront investments, ongoing licensing, or maintenance costs
- No specialist knowledge needed

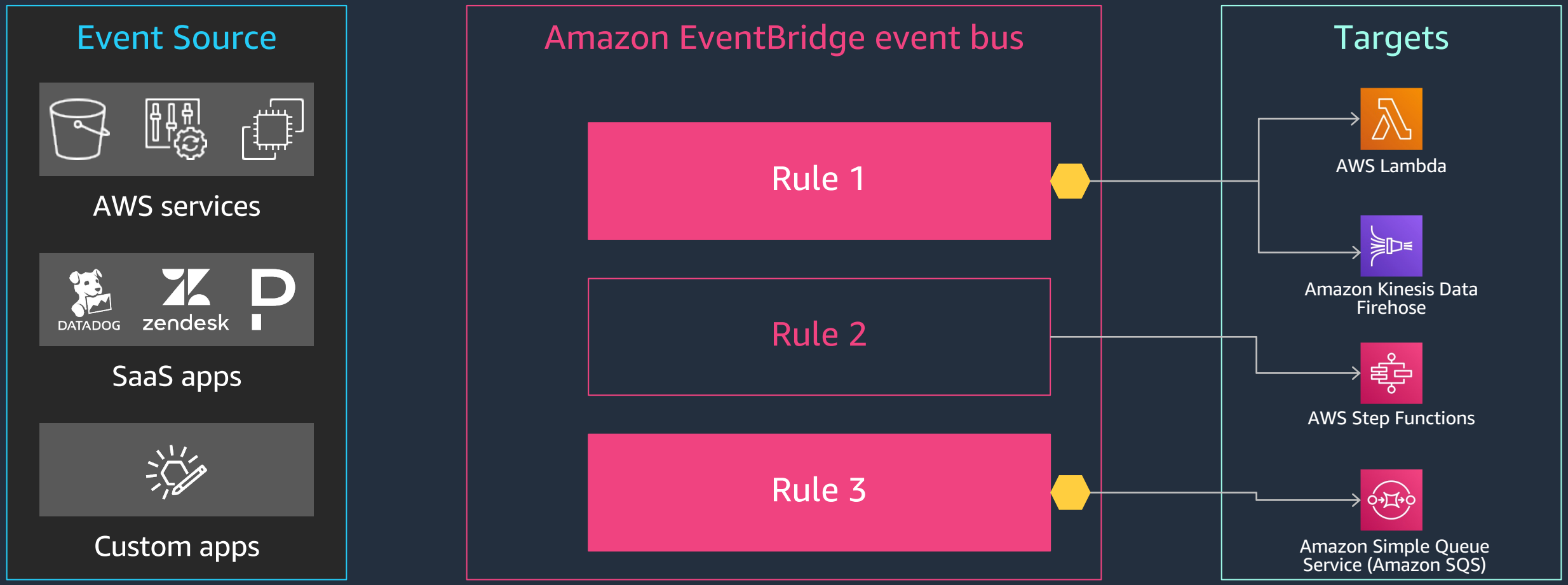
EventBridge architecture



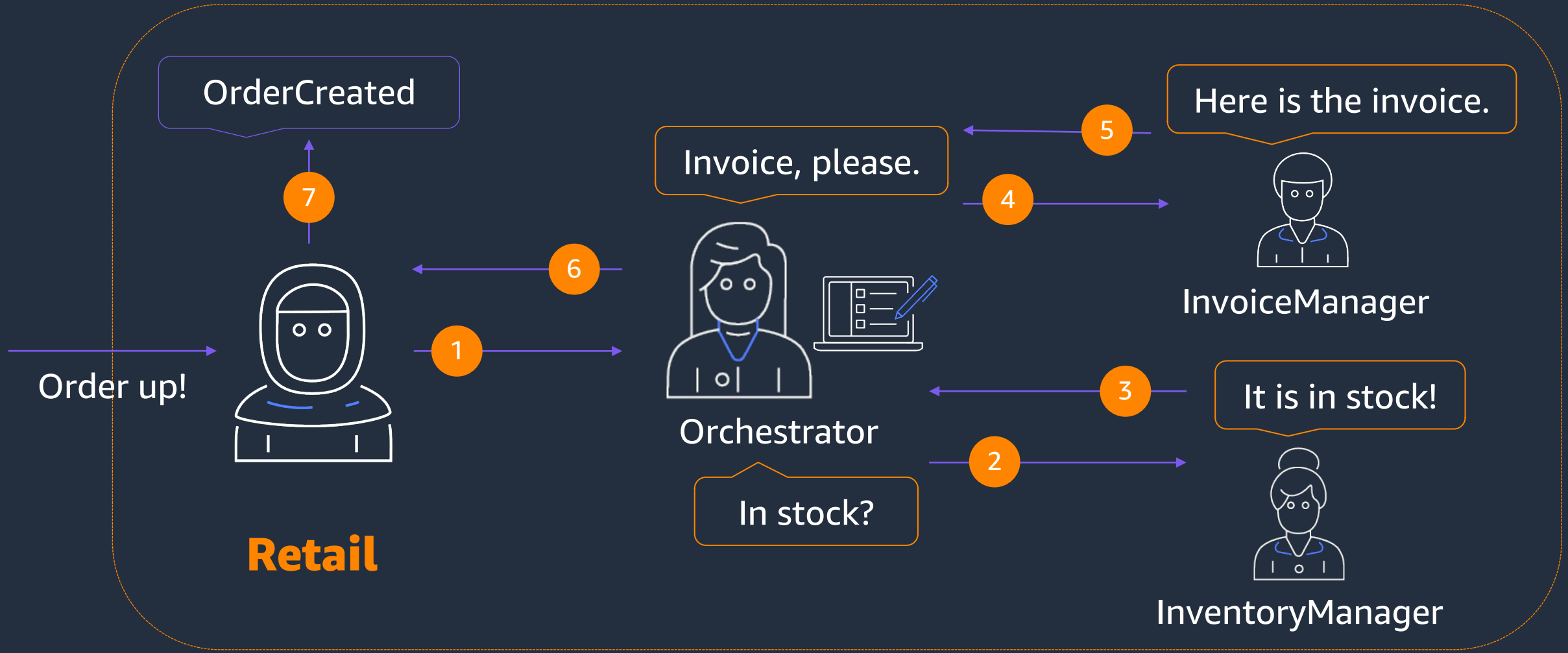
Architecture summary



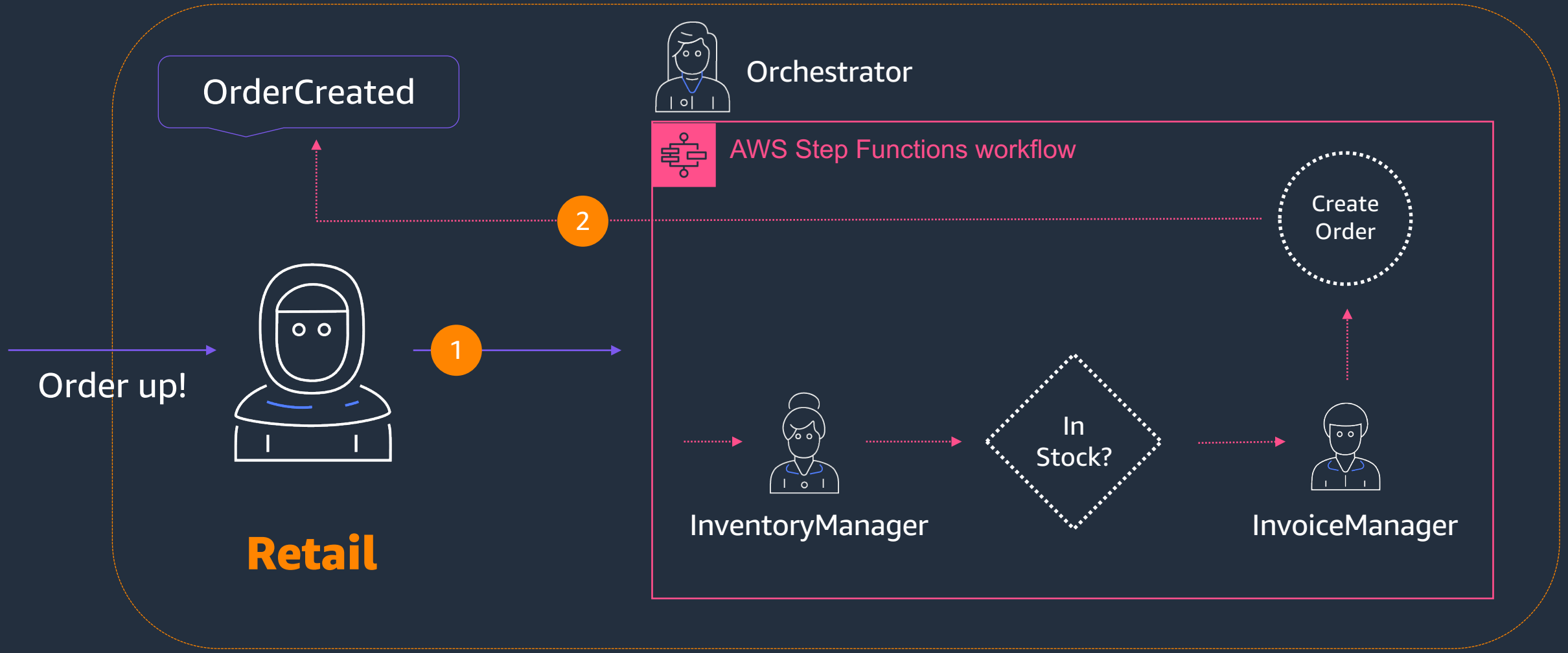
Architecture summary



Orchestrate a business process *within a domain*, resulting in a published event



Orchestrate a business process *within a domain*, resulting in a published event

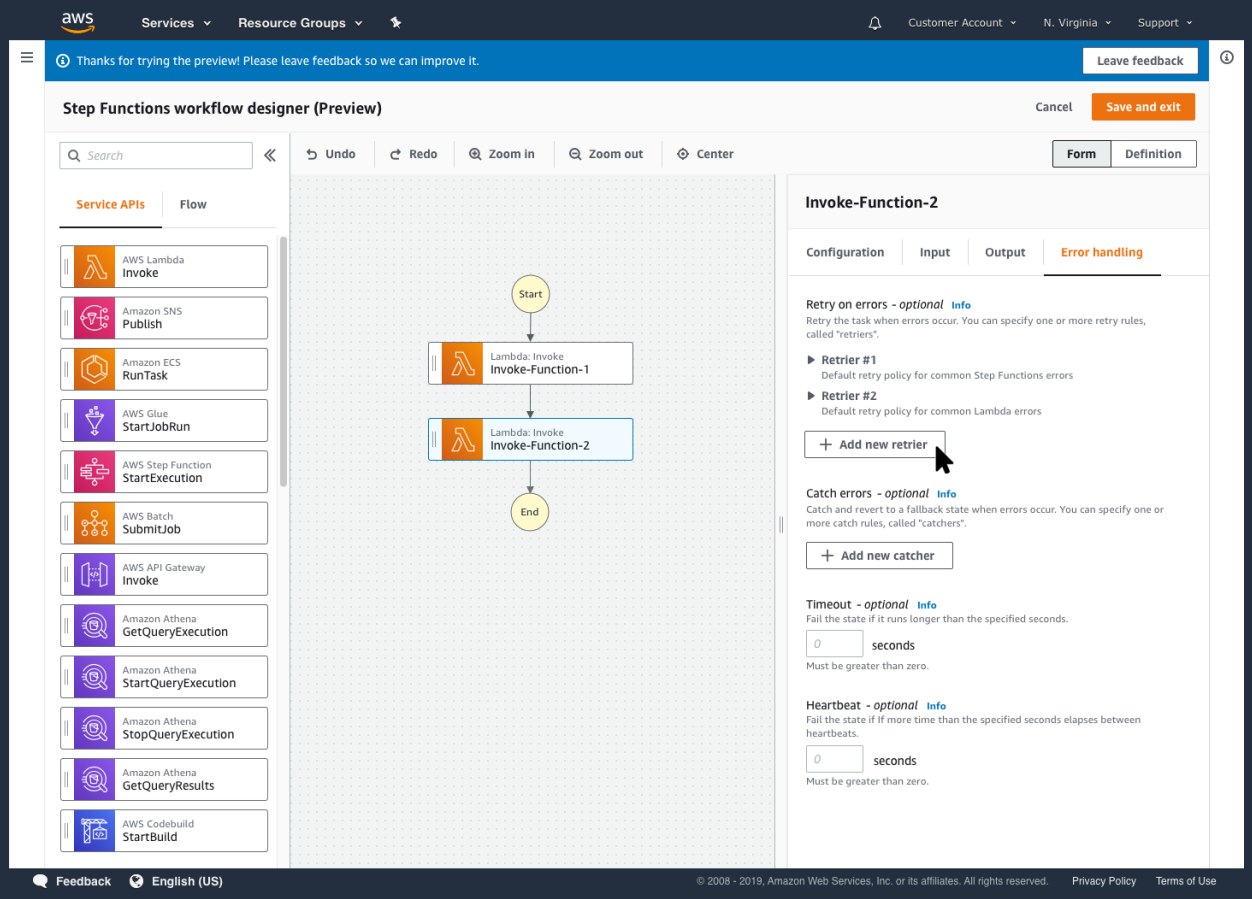


AWS Step Functions

The workflows you build with Step Functions are called state machines, and each step of your workflow is called a state.

When you execute your state machine, each move from one state to the next is called a state transition.

You can reuse components, easily edit the sequence of steps or swap out the code called by task states as your needs change.



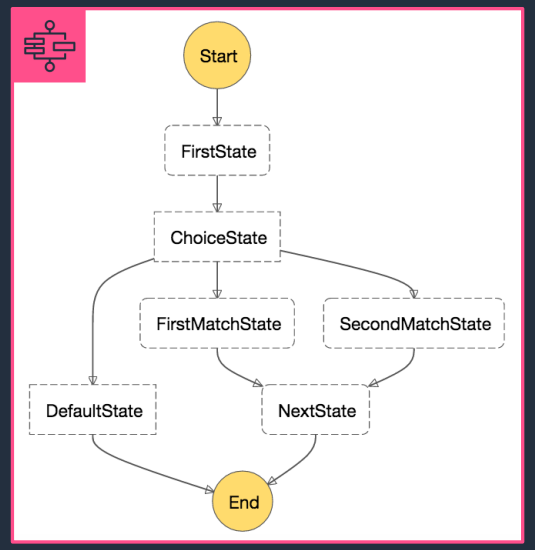
AWS STEP FUNCTIONS WORKFLOW STUDIO

Visual workflows

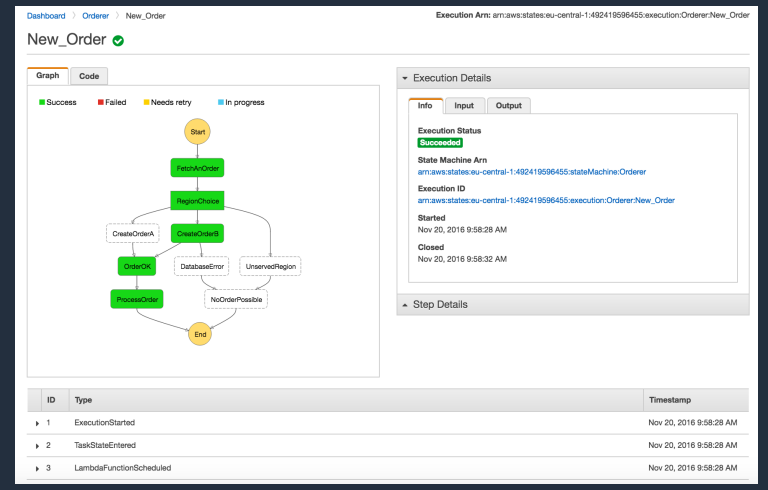
Define

- Drag-and-drop with Workflow Studio
- JSON (Amazon States Language)
- CDK **(**TypeScript, JavaScript, Python, Java, C#)
- Data Science SDK (Python)

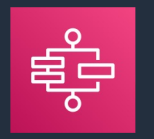
Visualize



Execute and monitor



Step Functions integration types



Optimized integrations

Customized to simplify the usage of 17 AWS services

Supported Integration patterns:

- Request Response
- Wait for a Callback (.waitForTaskToken)
- Run a Job (.sync)

AWS SDK integrations

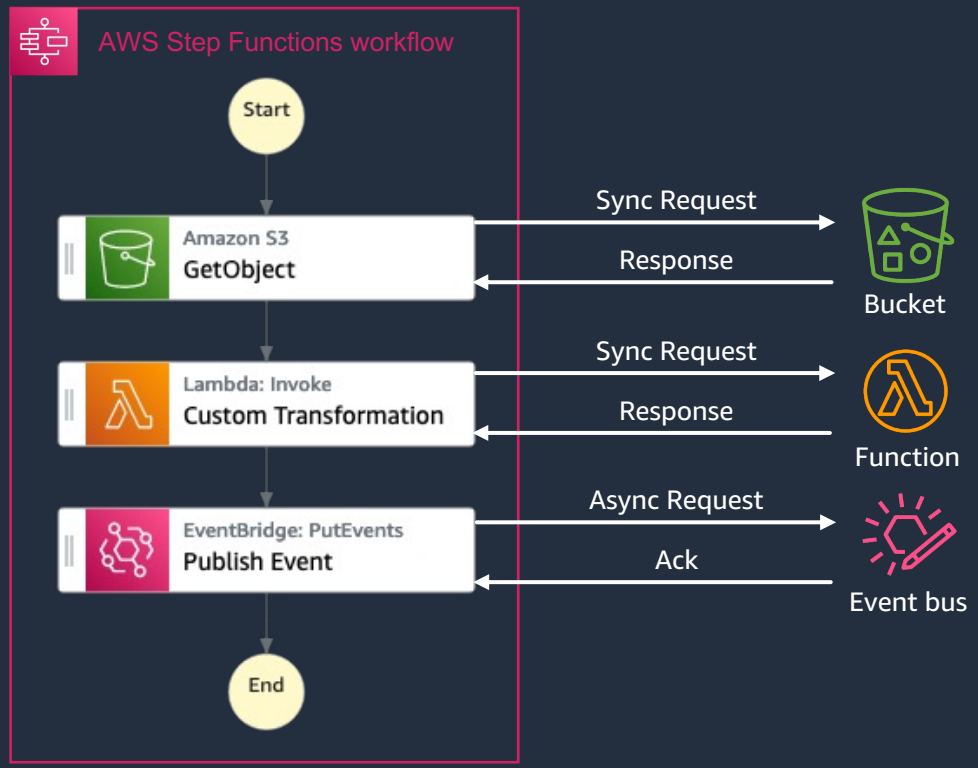
Call 200 AWS services directly (9000+ API actions)

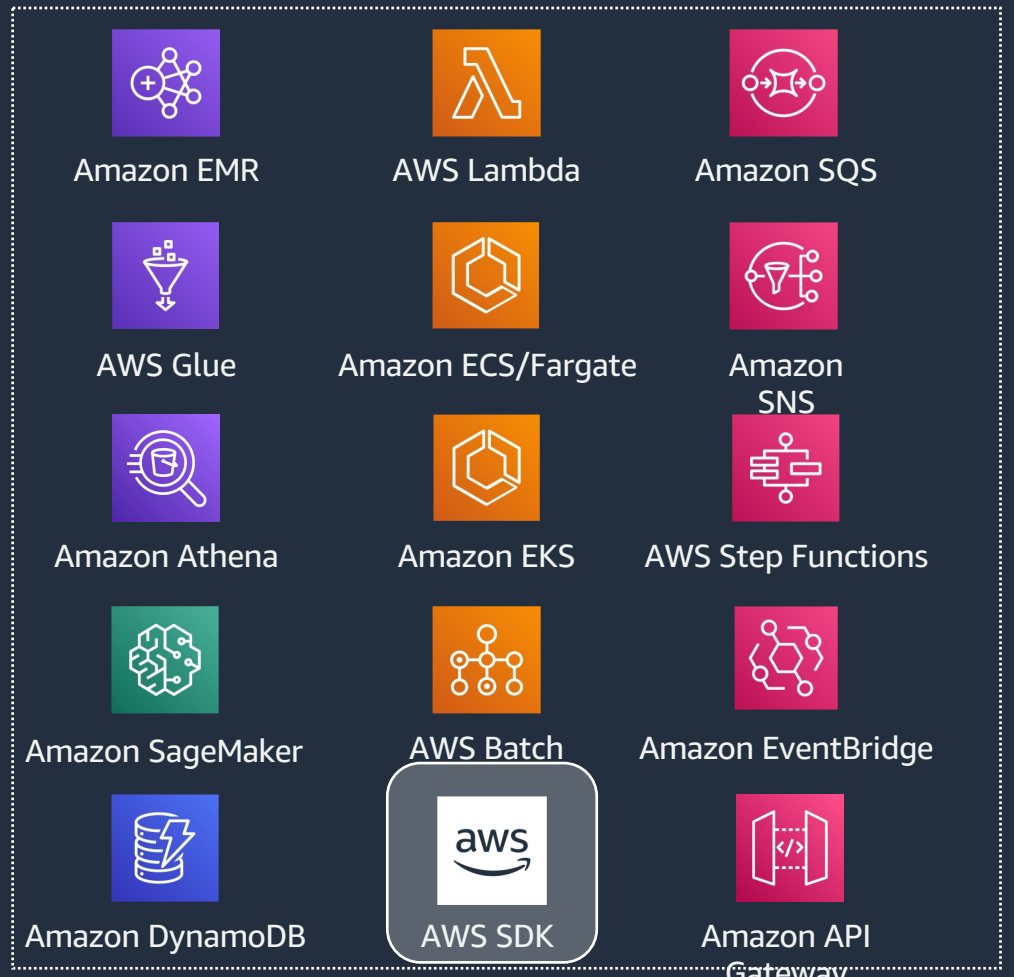
Supported Integration patterns:

- Request Response
- Wait for a Callback (.waitForTaskToken)

Step Functions: Request Response

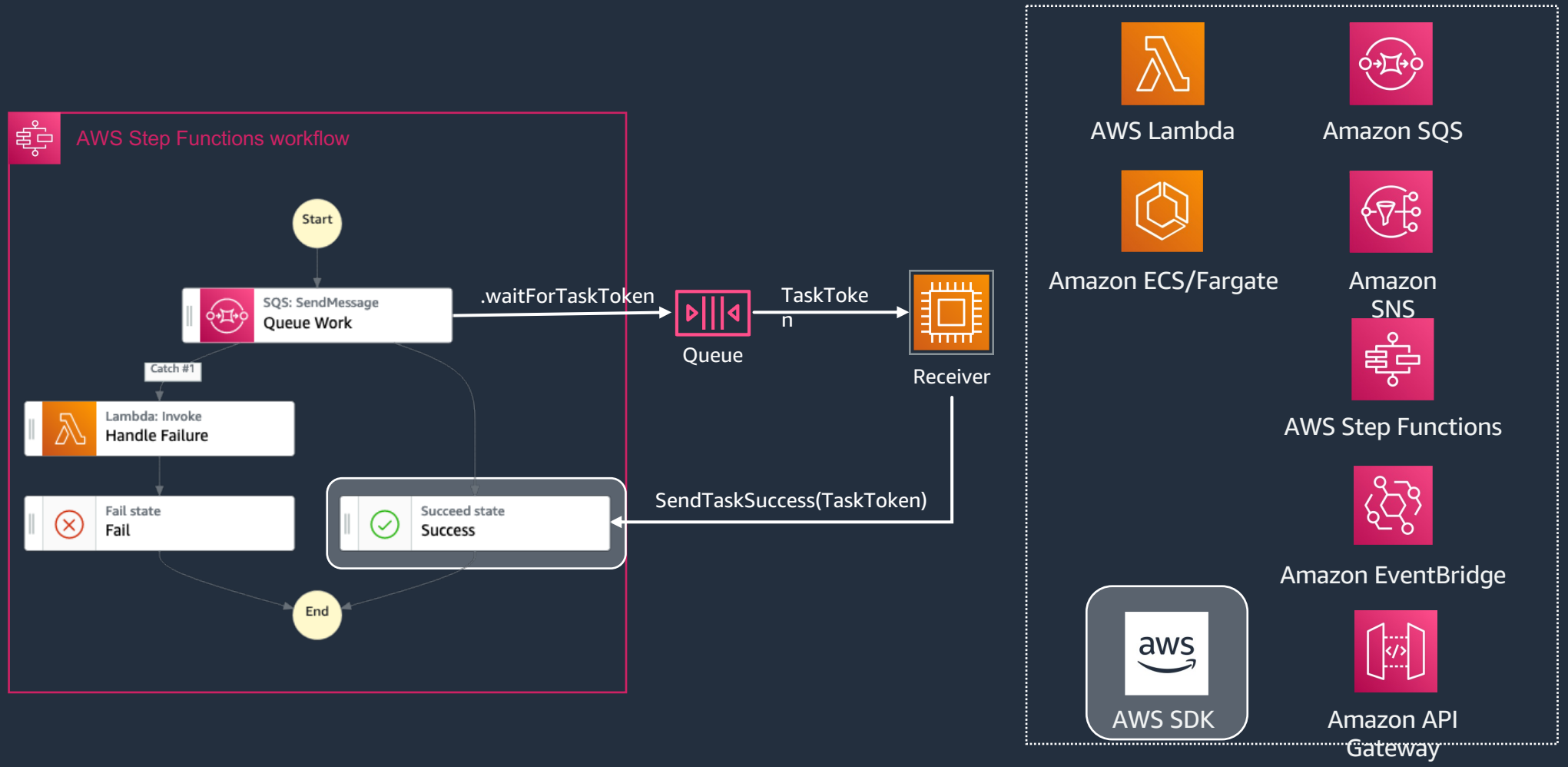
Service Integration Pattern





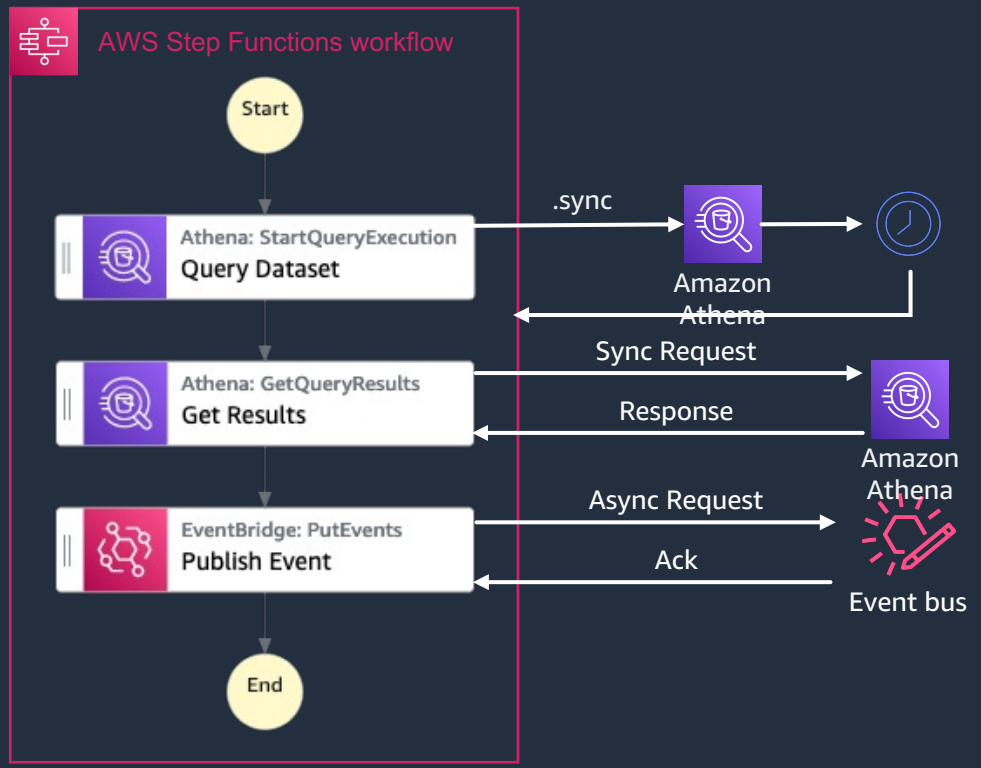
Step Functions: Wait for a Callback (.waitForTaskToken)

Service Integration Pattern

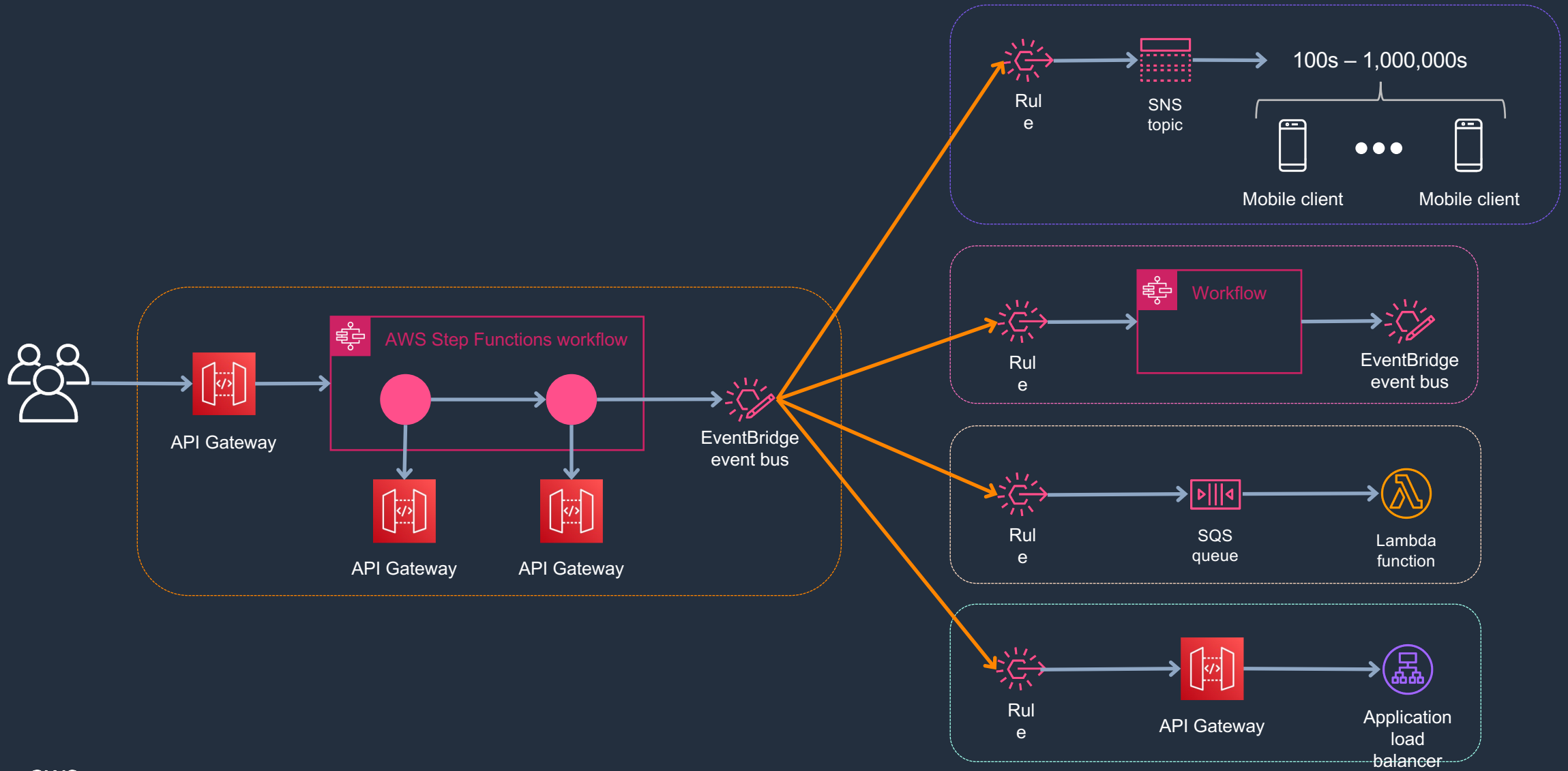


Step Functions: Run a Job (.sync)

Service Integration Pattern

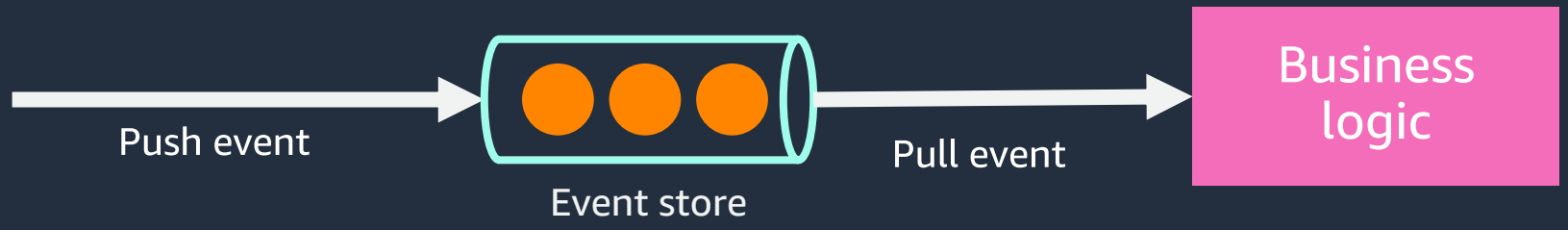


Better together: Orchestration + Choreography



Journey to event-driven architectures

Improve resiliency and scalability with event stores



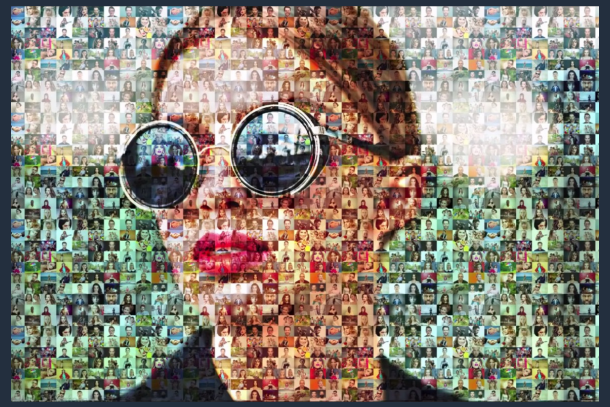
Buffer messages until service are available to process

Event stores handle messages and streams

Message Processing

- The individual message is the unit of work
- Computation/processing per message
- Message occurrence varies
- Message are deleted after consumption
- No need to track the position
- DLQ functionality built-in

Stream Processing



- The message stream is the unit of work
- Complex computation on many messages
- Constant stream of messages
- Messages are available after consumption until expiration
- Each client needs to track the current position in the stream
- No built-in DLQ functionality

Amazon Simple Queue Service (SQS)

WHAT IT IS

Simple, flexible, fully managed message queuing service for reliably and continuously exchanging any volume of messages from anywhere

(Standard and FIFO)

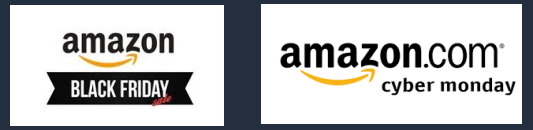
USE CASE

Build decoupled, highly scalable microservices, distributed systems, and serverless applications in the cloud

COOL CAPABILITIES

Nearly infinite scalability without pre-provisioning capacity

25B messages per hour



aws

Amazon Kinesis

WHAT IT IS

Enables you to ingest, buffer, and process streaming data in real-time, so you can derive insights in seconds or minutes instead of hours or days.

USE CASE

Ingest real-time data such as video, audio, application logs, website clickstreams, and IoT telemetry data for machine learning, analytics.

COOL CAPABILITIES

Can handle any amount of streaming data and process data from hundreds of thousands of sources with very low latencies.

aws

Journey to event-driven architectures

STEP 1
Start with the domain

STEP 3
Pick an event store

STEP 2
Coordinate events

STEP 4
Structure events

Sparse events vs. full state descriptions

Order 123 was created at 10:47 a.m. by customer 456

Events

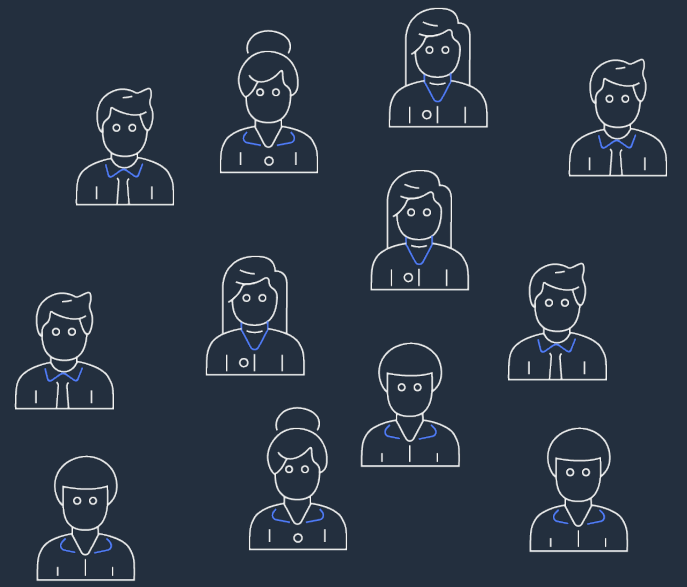
Order 123 was created at 10:47 a.m. by customer 456. The current status is Open, the total was $237.51, the items were …

Full state description

Considerations with sparse events

Order 123 was created by customer 456

What are the details for order 123?



Considerations with full state descriptions

```
{
  "source": "com.orders",
  "detail-type": "OrderCreated",
  "detail": {
    "metadata": {
      "idempotency-key": "c1b95b88"
    },
    "data": {
      "order-id": "1073459984",
      "status": "Open",
      "total": "237.51"
    }
}
```

- Event schemas should be backwards compatible

Considerations with full state descriptions

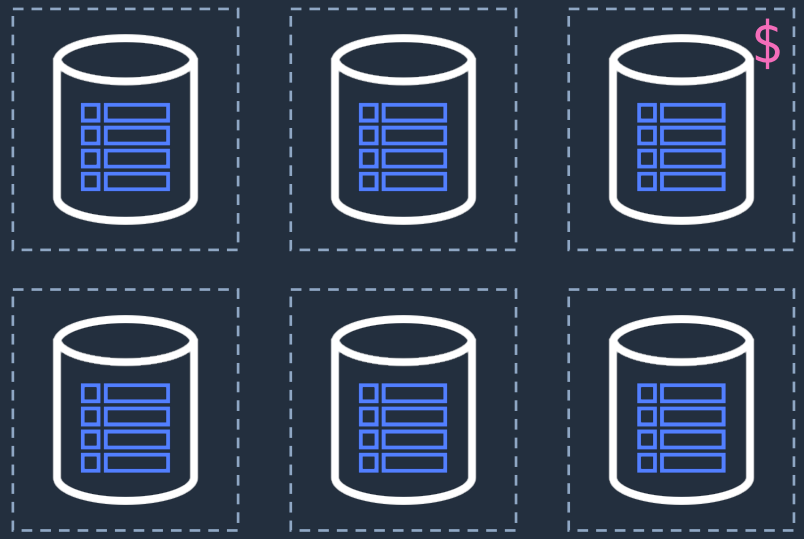
```
{
  "source": "com.orders",
  "detail-type": "OrderCreated",
  "detail": {
    "metadata": {
      "idempotency-key": "c1b95b88"
    },
    "data": {
      "order-id": "1073459984",
      "status": "Open",
      "total": "237.51"
    }
}
```

- Event schemas should be backwards compatible
- Cost to calculate values can increase over time

Considerations with full state descriptions

```
{
  "source": "com.orders",
  "detail-type": "OrderCreated",
  "detail": {
    "metadata": {
      "idempotency-key": "c1b95b88"
    },
    "data": {
      "order-id": "1073459984",
      "status": "Open",
      "total": "237.51"
    }
}
```

- Event schemas should be backwards compatible
- Cost to calculate values can increase over time



Design Considerations

Delivery semantics

At-least once delivery

Events can be delivered to a target more than once. Include logic to detect duplicate events by tracking the state of processed events (idempotent).

- Amazon EventBridge
- Amazon SNS Standard
- Amazon SQS Standard
- Amazon Kinesis

Exactly-once delivery*

Specify an identifier used by the AWS service for deduplication.

- Amazon MQ
- Amazon MSK
- Amazon SNS FIFO
- Amazon SQS FIFO

** outside of application error handling*

Event uniqueness

Idempotency

Operation will return the same results whether it is called once or multiple times (EIP).

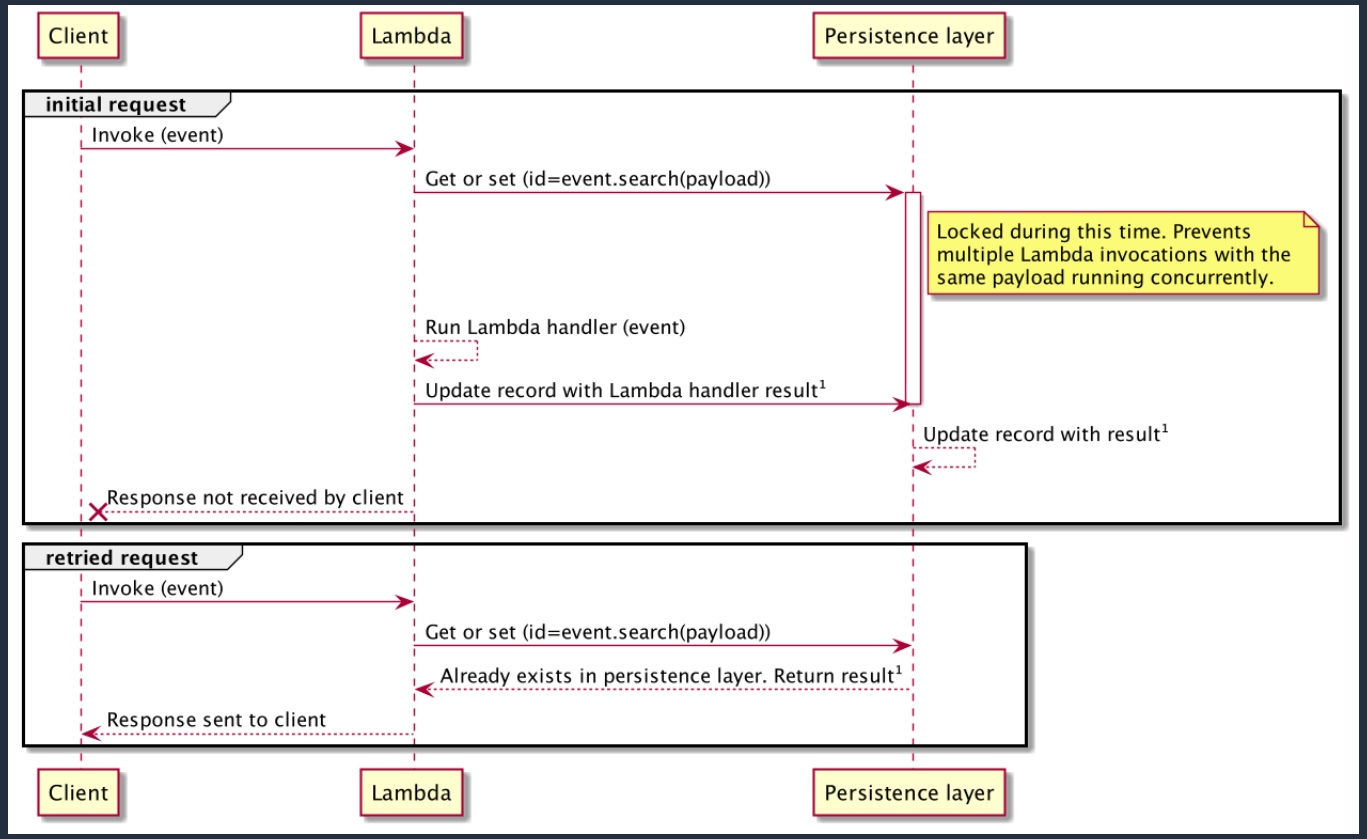
Idempotency Key

Assigned to the message by the sender to simplify deduplication by the receiver.

```
{
  "source": "com.orders",
  "detail-type": "OrderCreated",
  "detail": {
    "metadata": {
      "idempotency-key": "c1b95b88"
    },
    "data": {
      "order-id": "1073459984"
    }
}
```

Event deduplication strategies

- Manage idempotent state in an external data store
- Specify a time window for deduplication
- Lambda Powertools Python Idempotency library
- Making retries safe with idempotent APIs (*AWS Builders' Library*)



(*Lambda Powertools Python Idempotency*, https://bit.ly/3iWKu7z)

aws modern apps

Ordering semantics*

Unordered

Events can be delivered out of order.

- Amazon EventBridge
- Amazon SNS Standard
- Amazon SQS Standard

Ordered

Events are delivered in order within a partition, message group, etc (no global order).

- Amazon MQ
- Amazon MSK
- Amazon Kinesis
- Amazon SNS FIFO
- Amazon SQS FIFO

** Out-of-order event handling logic is highly application specific. If the application cannot be designed to handle out-of-order events, consider orchestration instead.*

Next Steps

- Event-driven architecture best practices
  https://aws.amazon.com/event-driven-architecture/

- Digital Course: Architecting Serverless Solutions
  https://www.aws.training/Details/eLearning?id=42594

- Event-driven reference architecture
  https://github.com/aws-samples/aws-serverless-ecommerce-platform

- Building event-driven architectures on AWS workshop
  http://event-driven-architecture.workshop.aws

Thank you!

Peterson Larentis

linkedin.com/in/peterson-larentis/

Repo Git with Deck + Additional Materials

https://github.com/plarentis/tdc-2022

🙏 **Survey** 🙏

https://eventbox.dev/survey/S8YLY3F