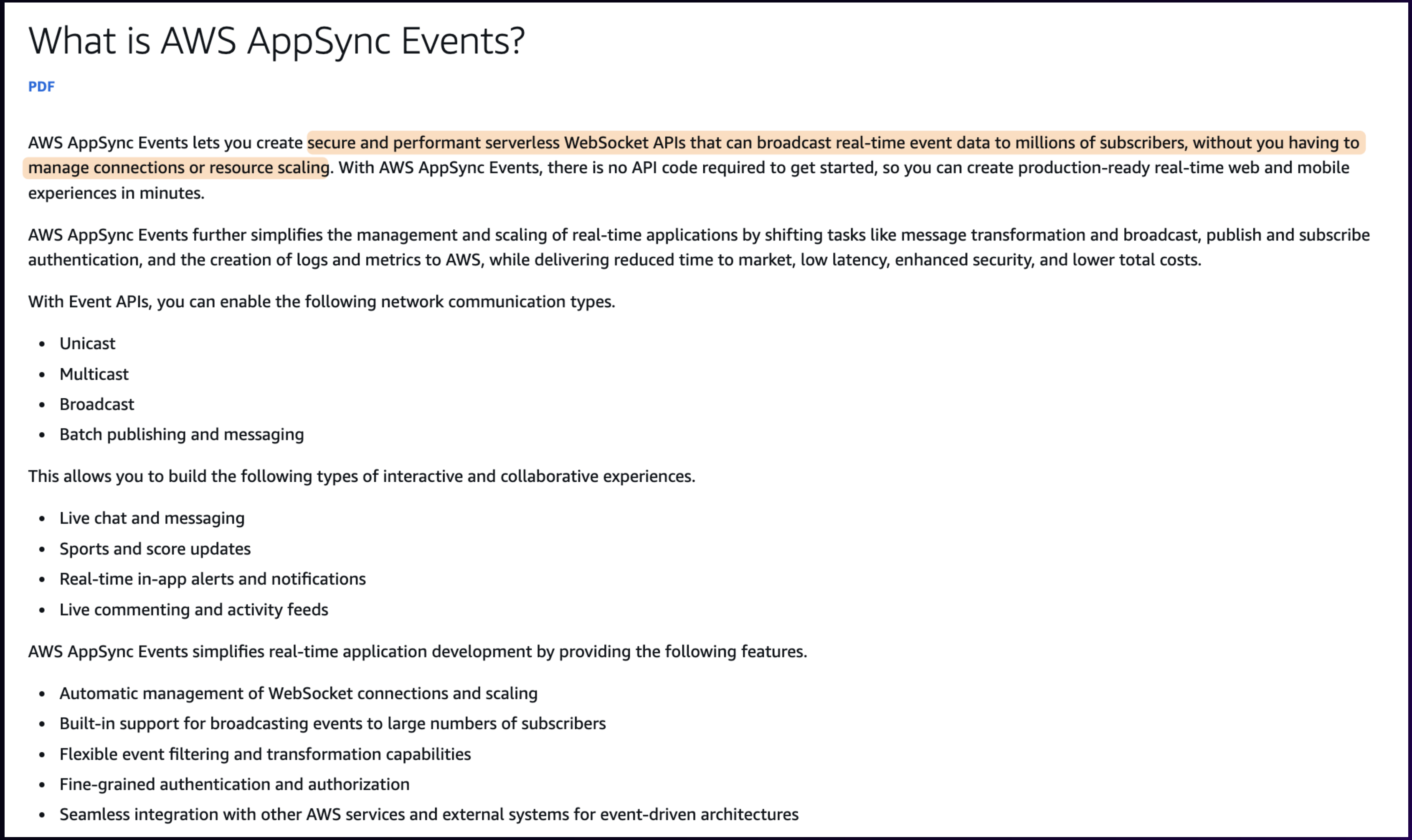
App Integration and Serverless

12 announcements (12 *pre-re:Invent*)

Real-time experiences with AppSync Events

AWS AppSync Events

SERVERLESS WEBSOCKET API TO POWER REAL-TIME WEB AND MOBILE EXPERIENCES AT SCALE



What is AWS AppSync Events?

PDF

AWS AppSync Events lets you create secure and performant serverless WebSocket APIs that can broadcast real-time event data to millions of subscribers, without you having to manage connections or resource scaling. With AWS AppSync Events, there is no API code required to get started, so you can create production-ready real-time web and mobile experiences in minutes.

AWS AppSync Events further simplifies the management and scaling of real-time applications by shifting tasks like message transformation and broadcast, publish and subscribe authentication, and the creation of logs and metrics to AWS, while delivering reduced time to market, low latency, enhanced security, and lower total costs.

With Event APIs, you can enable the following network communication types.

- Unicast
- Multicast
- Broadcast
- Batch publishing and messaging

This allows you to build the following types of interactive and collaborative experiences.

- Live chat and messaging
- Sports and score updates
- Real-time in-app alerts and notifications
- Live commenting and activity feeds

AWS AppSync Events simplifies real-time application development by providing the following features.

- Automatic management of WebSocket connections and scaling
- Built-in support for broadcasting events to large numbers of subscribers
- Flexible event filtering and transformation capabilities
- Fine-grained authentication and authorization
- Seamless integration with other AWS services and external systems for event-driven architectures

GA

Oct, 30th

pre-re:Invent

All Regions

AWS FEWAM Blog Post

Real-time examples

Event broadcast

Sport scores and stats

Price changes

Betting odd updates

Inventory availability

Feature flags

Group chat messages

Online voting windows

Stock and portfolio values

Presence

Game server state management

Real-time examples

Collaboration

- Gate check in tracking
- Location updates
- Luggage tracking
- Support ticket changes and assignees
- Truck scheduling
- Shared music queues
- Multi-turn game play
- LLM responses
- Document editing

Enabling real-time experiences can be difficult

Polling is inefficient

- Increased latency
- Unnecessary resource usage
- Scalability challenges

WebSockets are complex

- Connection management
- Message fanout
- Auto-scaling for peaks

Lots of backend code

- Pub/sub auth rules
- Message handling
- Logging and metrics

Traditional real-time solution architectures require heavy lifting and non-differentiating code

Options for building real-time experiences on AWS

Amazon API Gateway WebSockets

- Managed WebSocket API
- Plus self-hosted connection management and fanout

AWS IoT Core

- Managed MQTT broker
- Optimal for IoT device communications

AWS AppSync GraphQL

- Managed GraphQL API
- Configured to enable GraphQL subscriptions

Customers asked us for a purpose-built “pub/sub” solution for creating real-time experiences

AWS AppSync Events

The easiest way to build and scale real-time experiences

Create real-time experiences in minutes

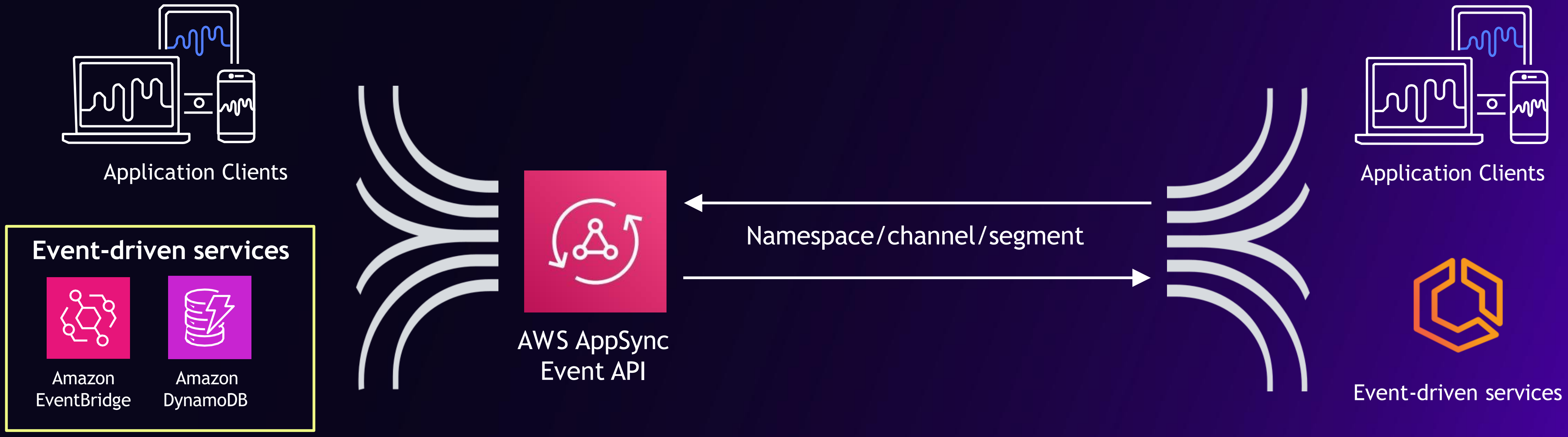
Reduce operational overhead

Boost application performance

Lower total costs

AppSync Events

SIMPLE PUB/SUB MODEL TO EASILY PUBLISH EVENTS TO 1000S OF CONNECTED SUBSCRIBERS



Publishers → WebSockets ← Subscribers

Built-in connection management and message fanout

DEMO

AppSync Events - Key benefits

Start fast

Real-time experiences

- Configure an Event API backend in minutes
- Publish events to channels via HTTP
- Subscribe to channels via standard WebSockets

Any scale

Low latency and massive scale

- Default quota of 1M messages/second
- Close to your customer with 30 global regions
- Custom domains, AWS WAF, logs and metrics

Reduced ops

Focus on your app business logic

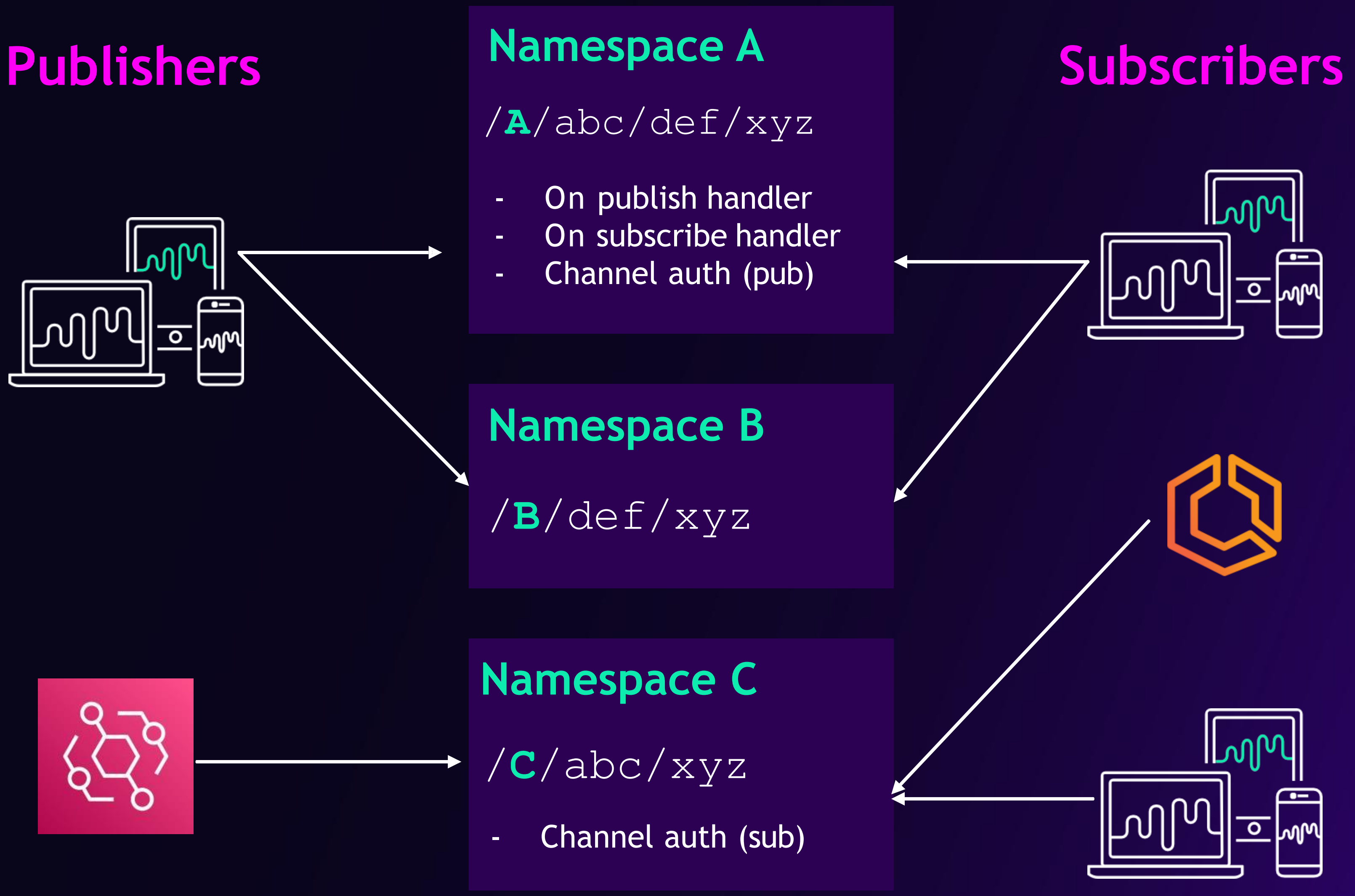
- Built-in connection management and fanout
- API key, Amazon Cognito, OIDC, AWS Lambda auth modes
- Built-in message filters and transformations

Lower costs

Serverless scale and pay-for-use billing

- Fully managed and serverless
- Generous free tier (250k ops/month)
- $1/million Event API operations

How AppSync Event APIs work



Publishers

Namespace A

/A/abc/def/xyz

- On publish handler
- On subscribe handler
- Channel auth (pub)

Namespace B

/B/def/xyz

Namespace C

/C/abc/xyz

- Channel auth (sub)

Subscribers

Publish

- Via HTTP
- Create channel
- JSON payload
- Optional handler

Subscribe

- Via WebSockets
- Optional handler

Auth

- API level
- Optional namespace auth override

AWS AppSync Events features

AUTHORIZATION TYPES

Amazon Cognito User Pools

API keys

AWS Lambda

OpenID Connect

AWS Identity and Access Management

Mix multiple auth modes at the API and Namespace level

AWS AppSync Events

DEFAULT QUOTAS AND PRICING

OUTBOUND MESSAGES

1,000,000

per second
(adjustable)

INBOUND MESSAGES

10,000

per second
(adjustable)

NEW CONNECTIONS

2,000

per second
(adjustable)

REGIONS

30

globally

API OPERATION COST

$1

per million operations
(in/out messages +
WebSocket operations)

CONNECTION COST

$0.08

per million minutes
(+standard data transfer
charges)

FREE TIER

250,000

operations and minutes per
month, for 12 months

Source: AWS Approved Public Stats

Amazon EventBridge Latency Reduction

94% END-TO-END LATENCY IMPROVEMENT FOR EVENT BUSES AT NO ADDITIONAL COST

GA

Nov, 12th

pre-re:Invent

All Regions

Amazon EventBridge Event Buses announces up to **94% improvement** in end-to-end latency for Event Buses, enabling you to handle highly latency sensitive applications.

You can now detect and respond to critical events more quickly, enabling rapid innovation, faster decision-making, and improved operational efficiency.

AWS Announcement

NEW

AWS Lambda SnapStart for Python and .NET

- Delivers faster startup performance as low as sub-second

AWS Lambda SnapStart for .NET and Python

REDUCE COLD-STARTS FOR .NET CORE AND PYTHON SERVERLESS APPLICATIONS

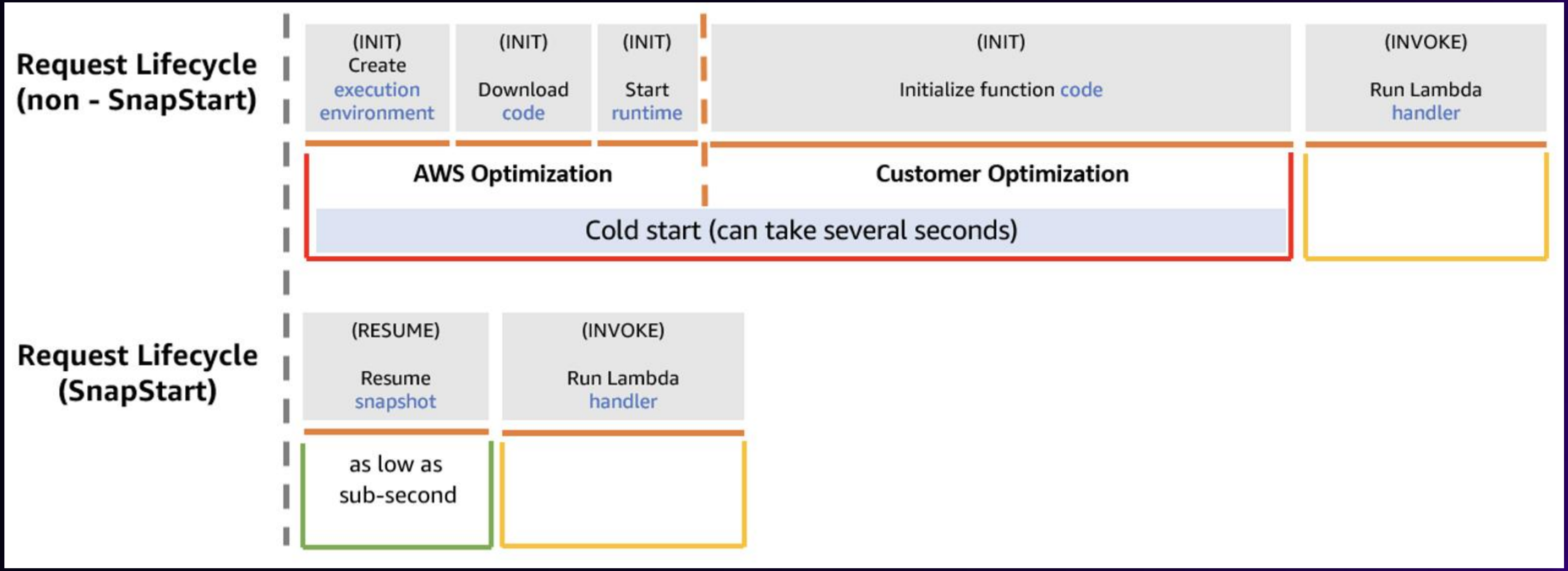
In addition to Java, from now you can use AWS Lambda SnapStart with your functions that use the **Python** and **.NET** managed runtimes, to deliver as low as sub-second startup performance.

GA

Nov, 18th

pre-re:Invent

9 Regions



Request Lifecycle (non - SnapStart)
(INIT) Create execution environment
(INIT) Download code
(INIT) Start runtime
(INIT) Initialize function code
(INVOKE) Run Lambda handler
AWS Optimization
Customer Optimization
Cold start (can take several seconds)
Request Lifecycle (SnapStart)
(RESUME) Resume snapshot
(INVOKE) Run Lambda handler
as low as sub-second

AWS News Blog Post

AWS Lambda SnapStart

Benefit

Delivers faster startup performance, from several seconds to as low as sub-second, with minimal or no changes to your function code

AWS Lambda SnapStart

Benefit

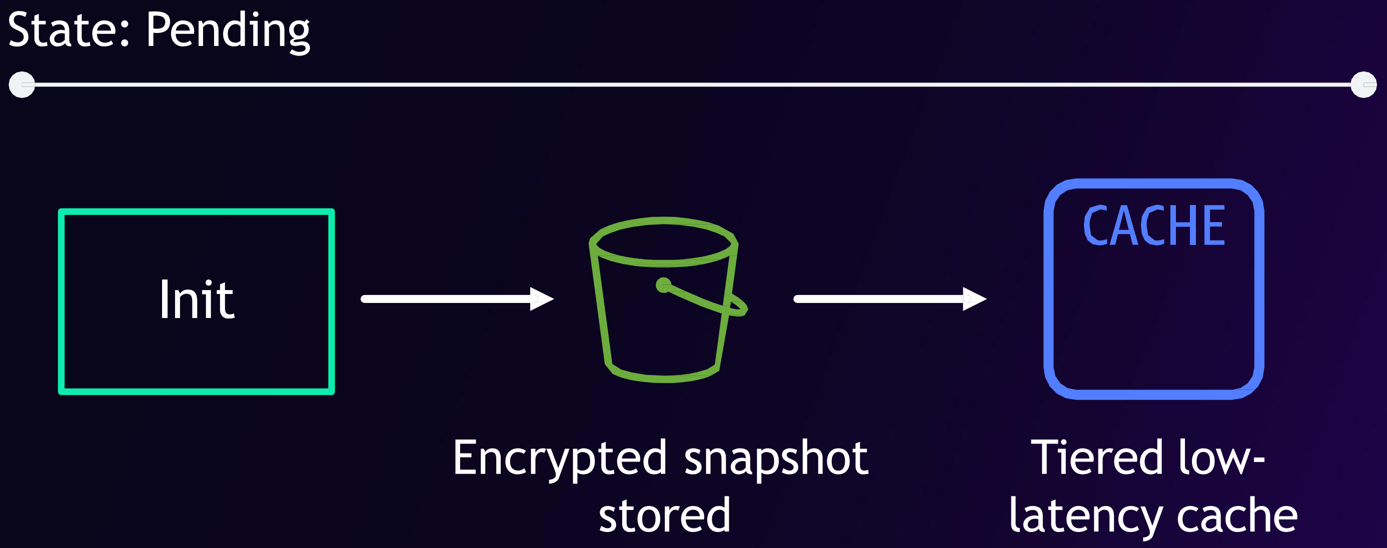
Delivers faster startup performance, from several seconds to as low as sub-second, with minimal or no changes to your function code

Supported on

- Python runtime versions 3.12 and later
- .NET 8 and later
- Java 11 and later

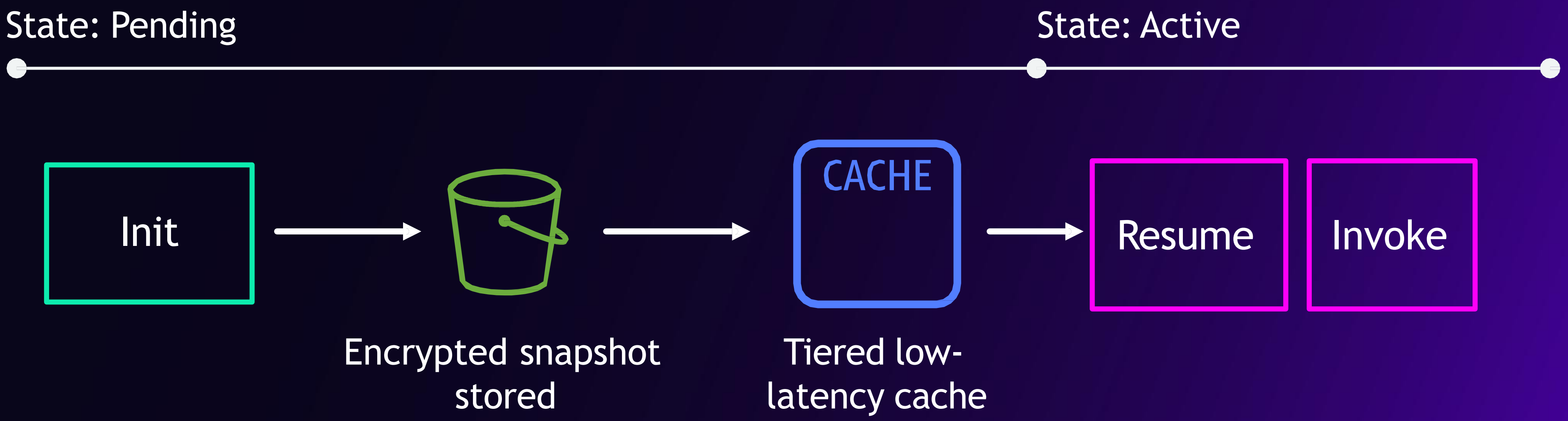
SnapStart overview

publish-version



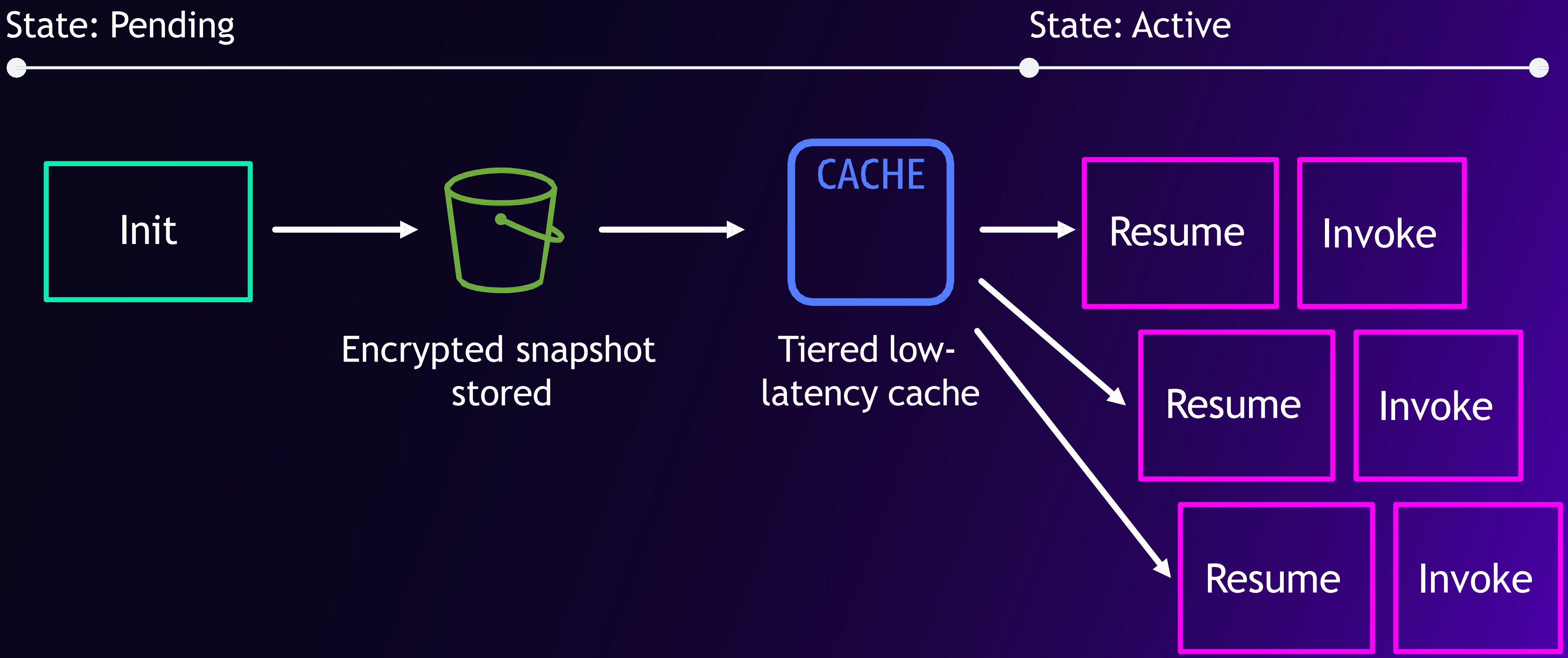
SnapStart overview

publish-version



SnapStart overview

publish-version



State: Pending

State: Active

Init

Encrypted snapshot stored

CACHE

Tiered low-latency cache

Resume

Invoke

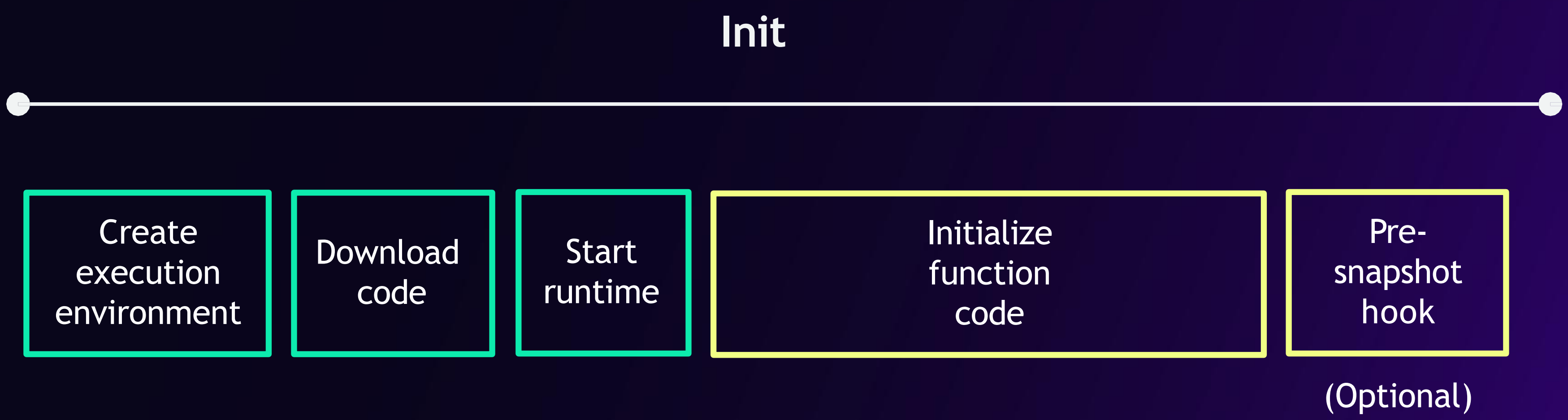
Resume

Invoke

Resume

Invoke

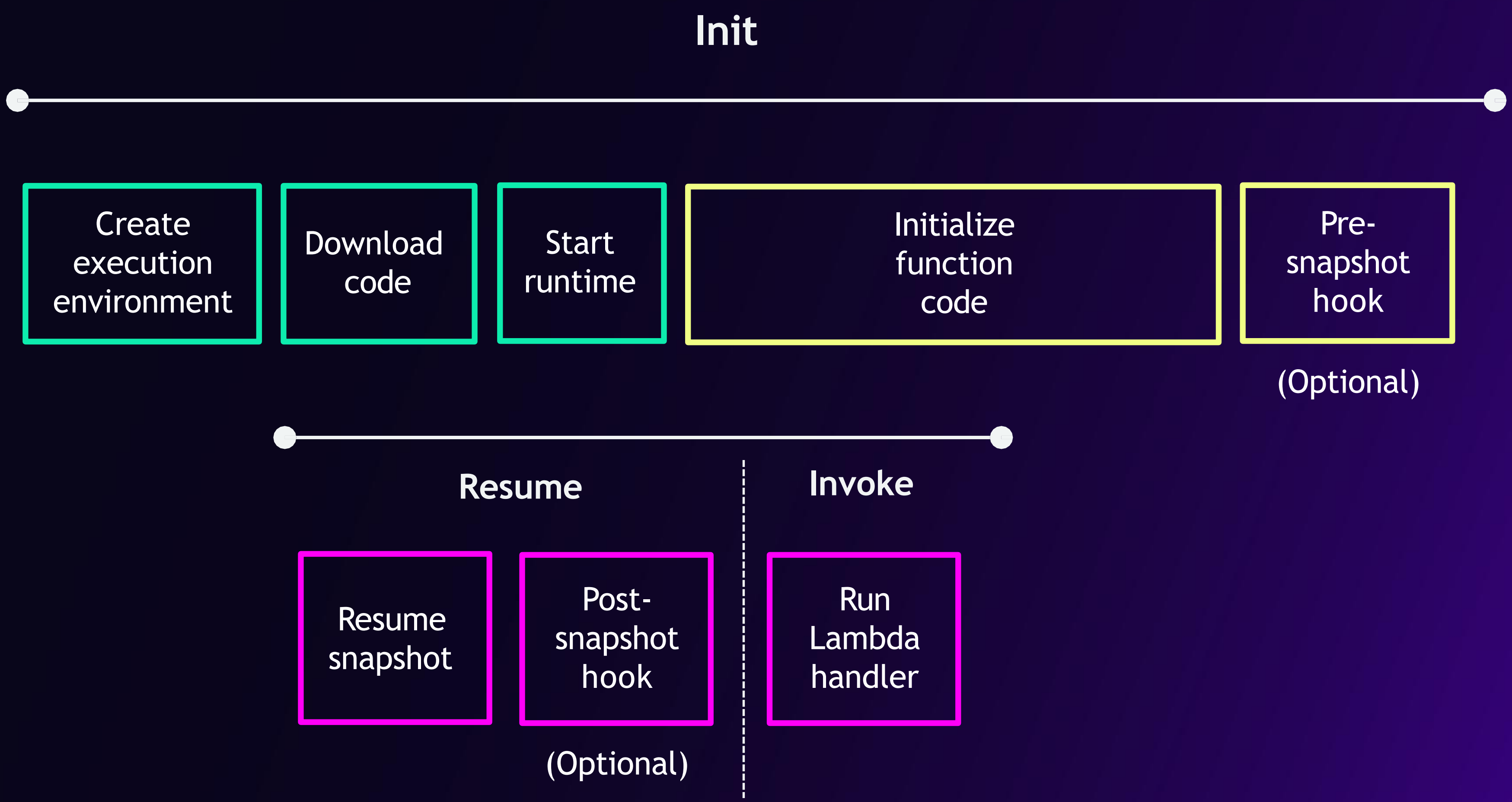
Invocation model



Firecracker

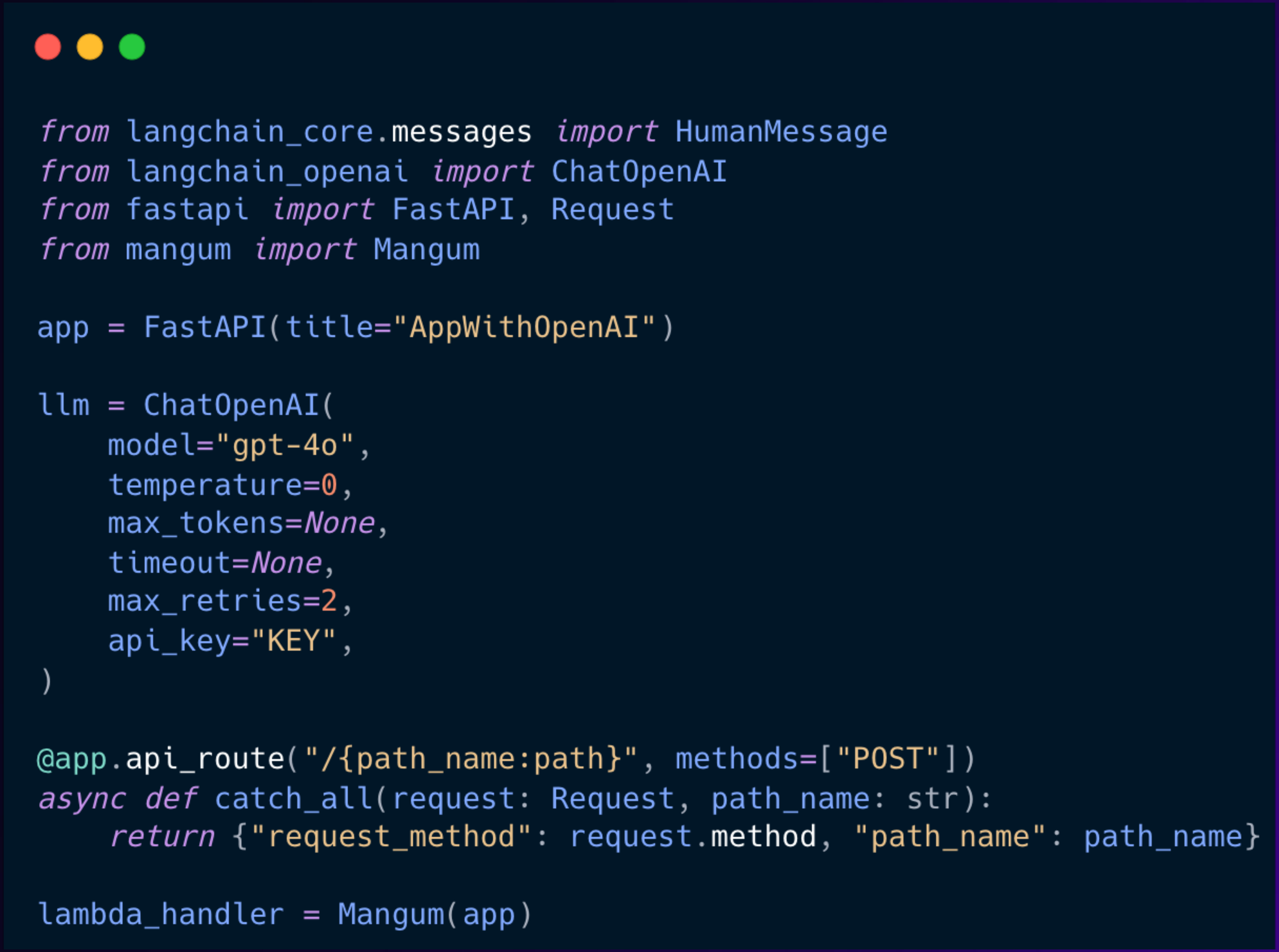
microVM snapshot technology

Invocation model



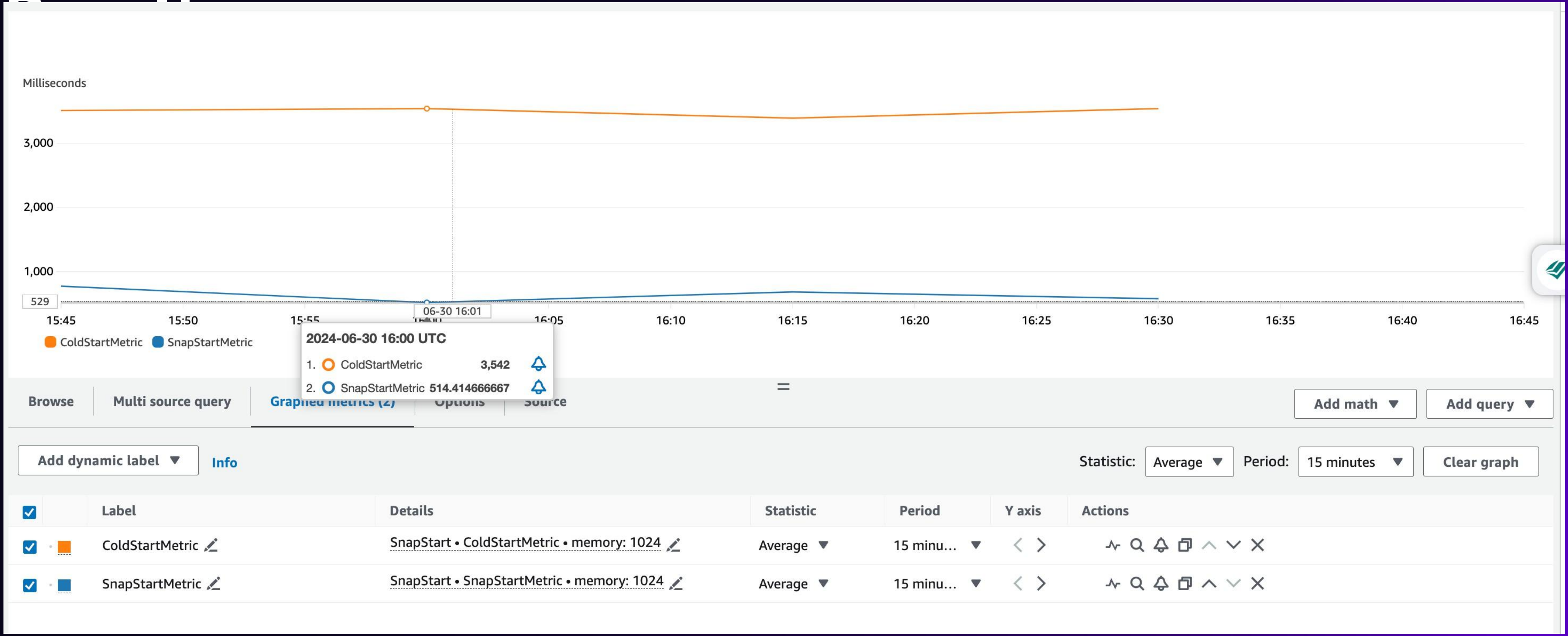
Use cases

Chatbot with gen AI



```python
from langchain_core.messages import HumanMessage
from langchain_openai import ChatOpenAI
from fastapi import FastAPI, Request
from mangum import Mangum

app = FastAPI(title="AppWithOpenAI")

llm = ChatOpenAI(
    model="gpt-4o",
    temperature=0,
    max_tokens=None,
    timeout=None,
    max_retries=2,
    api_key="KEY",
)

@app.api_route("/{path_name:path}", methods=["POST"])
async def catch_all(request: Request, path_name: str):
    return {"request_method": request.method, "path_name": path_name}

lambda_handler = Mangum(app)
```

Chatbot with gen AI –



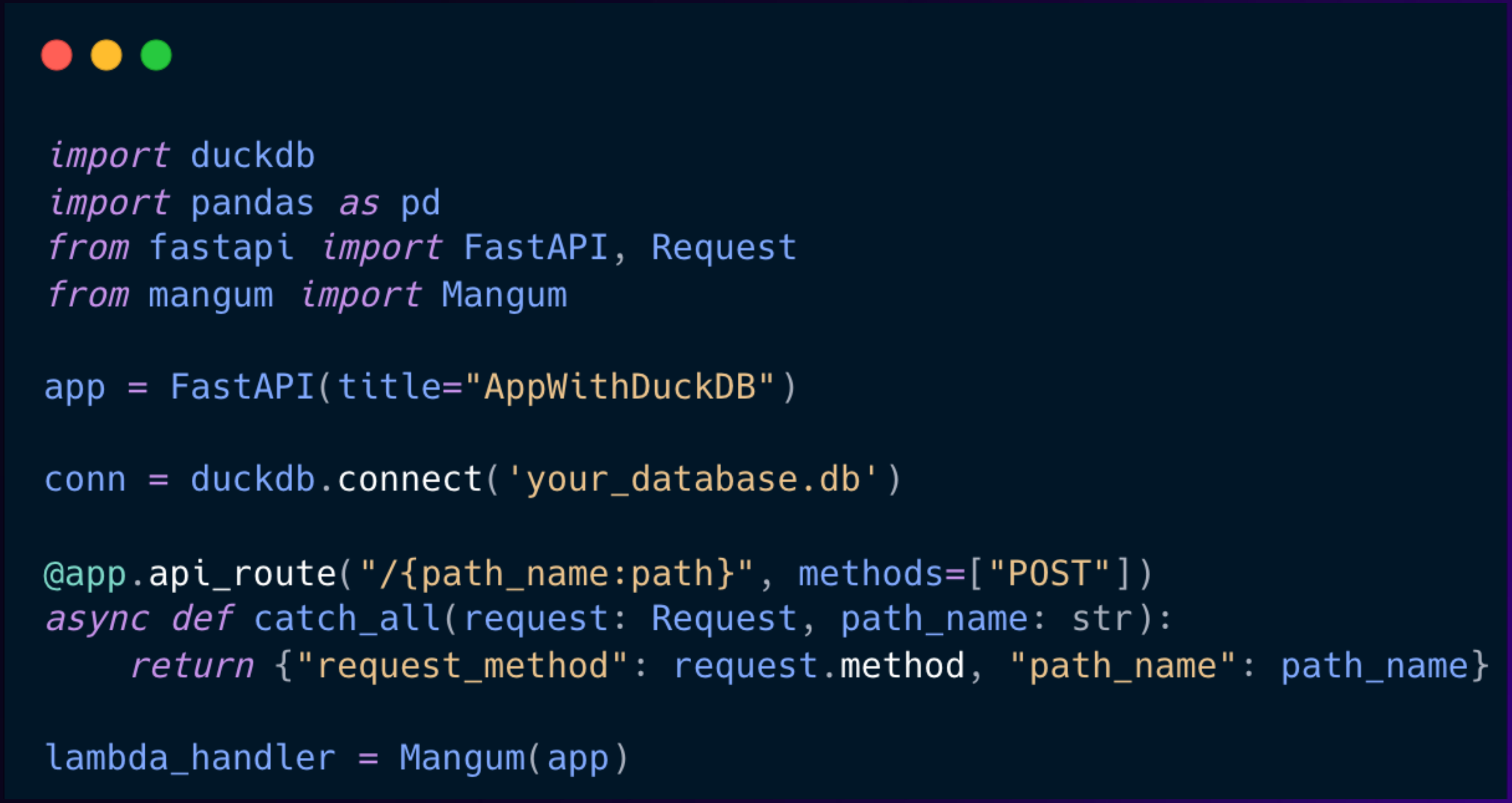
2024-06-30 16:00 UTC

1. ○ ColdStartMetric **3,542**
2. ○ SnapStartMetric **514.414666667**

Statistic: Average ▼ Period: 15 minutes ▼

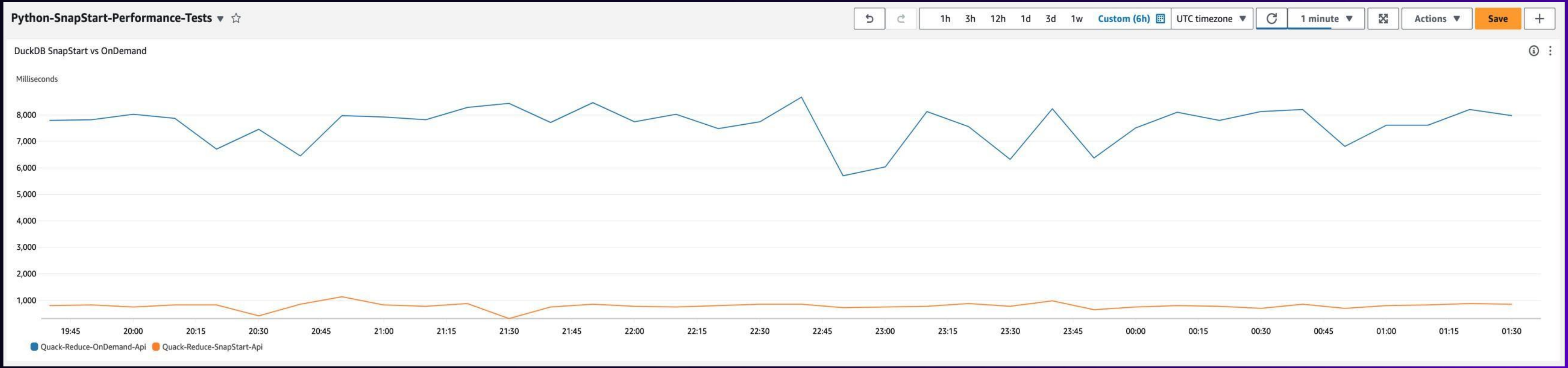
☑		Label	Details	Statistic	Period	Y axis	Actions
☑	🟧	ColdStartMetric	SnapStart • ColdStartMetric • memory: 1024	Average ▼	15 minu… ▼	< >	
☑	🟦	SnapStartMetric	SnapStart • SnapStartMetric • memory: 1024	Average ▼	15 minu… ▼	< >	

Data analysis



```python
import duckdb
import pandas as pd
from fastapi import FastAPI, Request
from mangum import Mangum

app = FastAPI(title="AppWithDuckDB")

conn = duckdb.connect('your_database.db')

@app.api_route("/{path_name:path}", methods=["POST"])
async def catch_all(request: Request, path_name: str):
    return {"request_method": request.method, "path_name": path_name}

lambda_handler = Mangum(app)
```

Data analysis – Results



Pricing

SnapStart pricing

Usage priced along two dimensions – represents a nominal added charge for typical use cases

- Cache – $3.9 per GB-month
  - Charged over active duration of a function version ($0.0000015046 per GB-second)
  - Lower costs by deleting unused versions
- Restore – $1.4 per GB restored with 10K restores
  - Charged per GB restored ($0.0001397998 per GB restored)

Pricing example (monthly)

- Let’s assume a 1 GB function, 300 ms execution duration

- 100M invokes, **250K restores** (i.e., cold starts)
  - Total charges: $558.8
    - Compute charges: $500; request charges: $20 (***no change***)
    - SnapStart cache charges: $3.9 ($3.9 x 1 GB)
    - SnapStart restore charges: $34.9 ($0.0001397998 x 1 GB x 250K restores)

Event Source Mappings CloudWatch Metrics

INCREASED OBSERVABILITY INTO EVENT SOURCE MAPPINGS (ESM) IN AWS LAMBDA

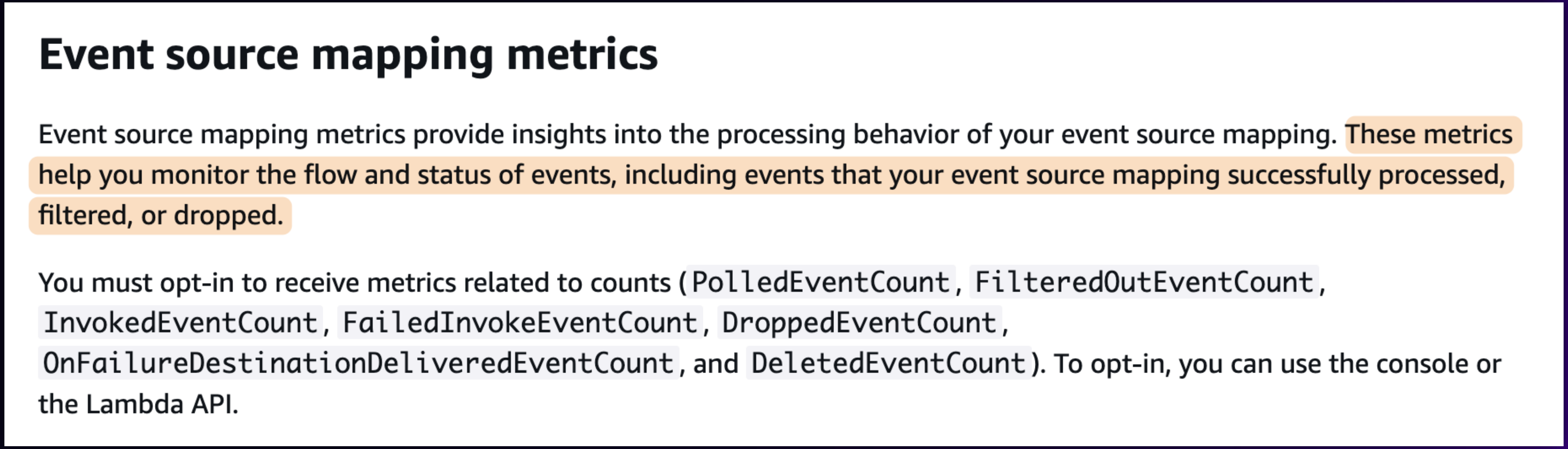
GA

Nov, 21st

pre-re:Invent

All Regions

AWS Lambda team announced new Amazon CloudWatch metrics for Event Source Mappings (ESMs), which provide customers visibility into the processing state of events with Amazon SQS, Amazon Kinesis, and Amazon DynamoDB event source integrations.



Event source mapping metrics

Event source mapping metrics provide insights into the processing behavior of your event source mapping. These metrics help you monitor the flow and status of events, including events that your event source mapping successfully processed, filtered, or dropped.

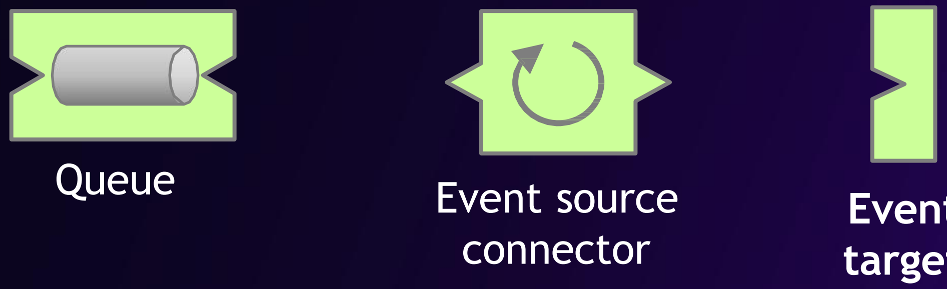
You must opt-in to receive metrics related to counts (`PolledEventCount`, `FilteredOutEventCount`, `InvokedEventCount`, `FailedInvokeEventCount`, `DroppedEventCount`, `OnFailureDestinationDeliveredEventCount`, and `DeletedEventCount`). To opt-in, you can use the console or the Lambda API.

AWS Announcement

What is an event source connector

An **event source connector** is a class of event processor services that transform and route events to consumers

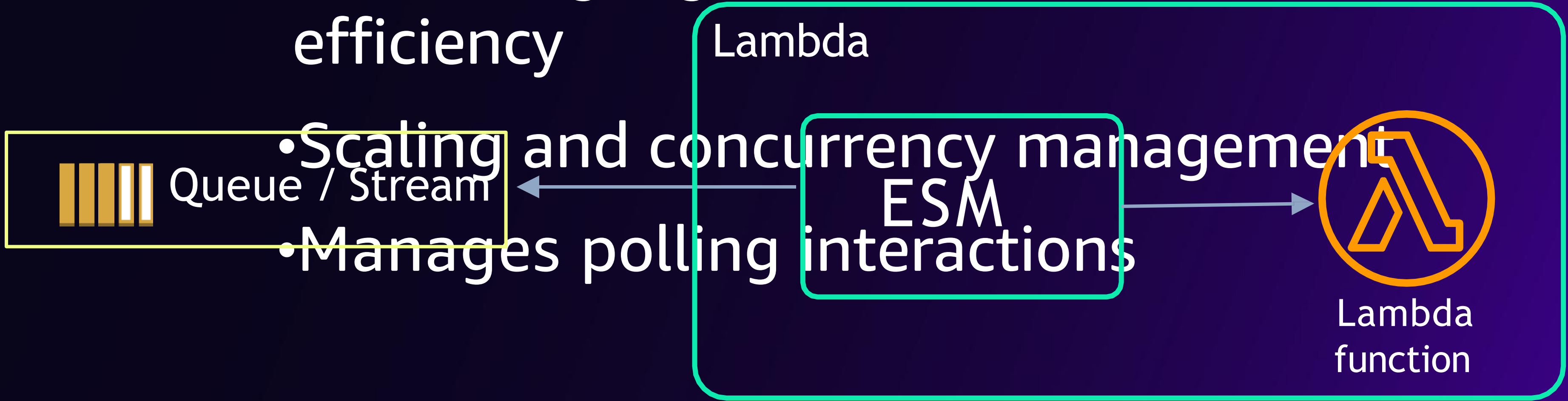
- Examples: Amazon EventBridge Pipes and AWS Lambda ESM
- Connects passive event sources with downstream targets



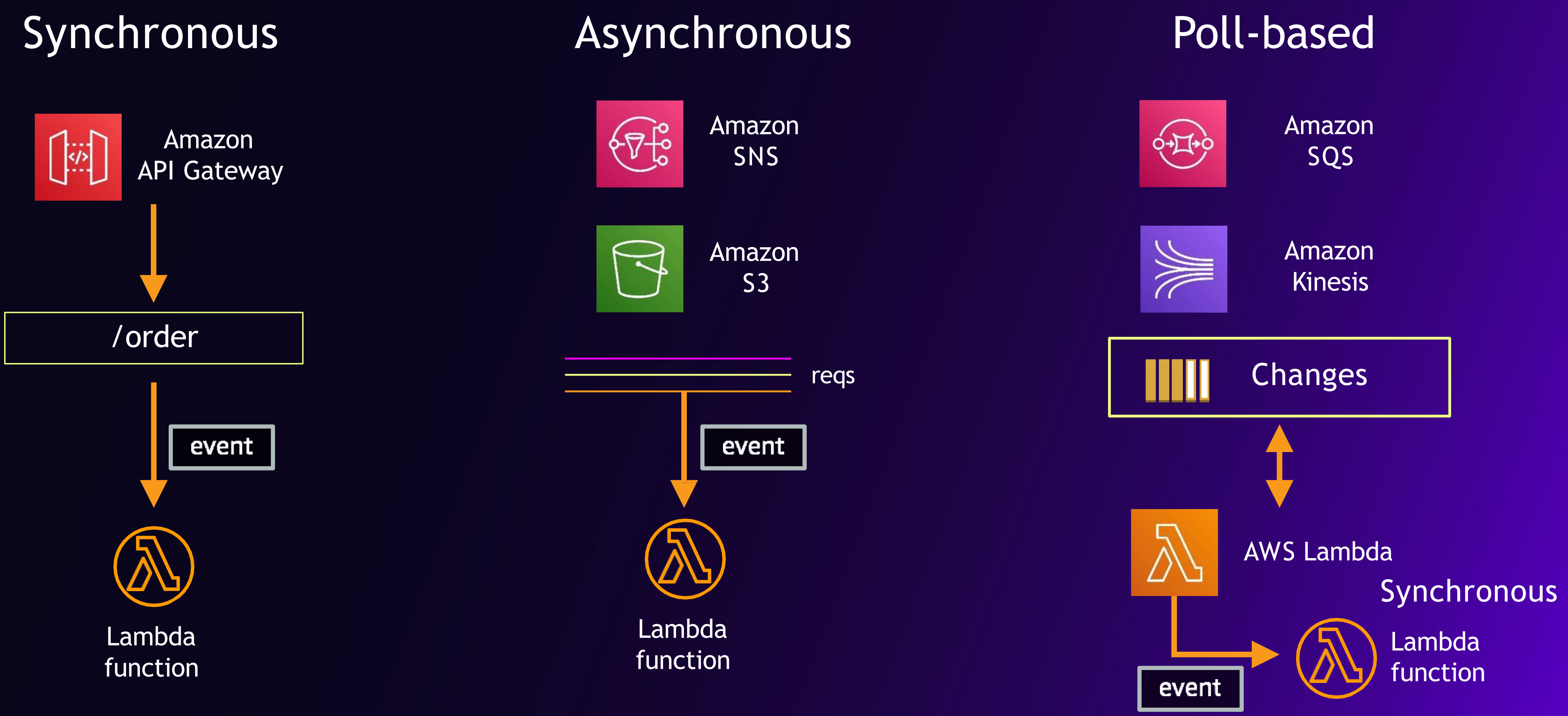
What is Lambda event source mapping (ESM)

An **event source mapping** is a Lambda resource that reads items from stream- and queue-based services and invokes a function with batches of records

- Processing high-volume data via batch for high efficiency
- Scaling and concurrency management
- Manages polling interactions



Lambda invocation models



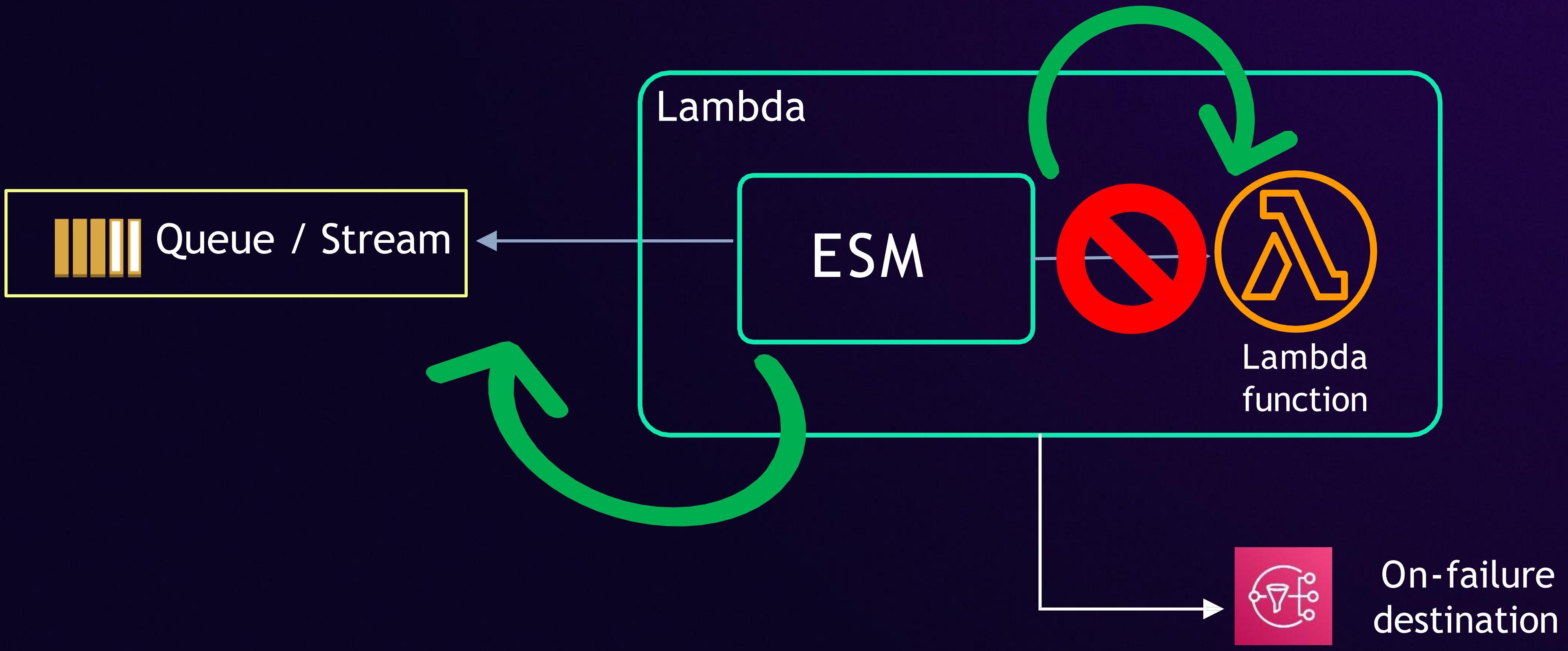
How Lambda ESM works

1. Lambda Service polls the queue / stream

2. The ESM synchronously invokes the Lambda function with the batch of records

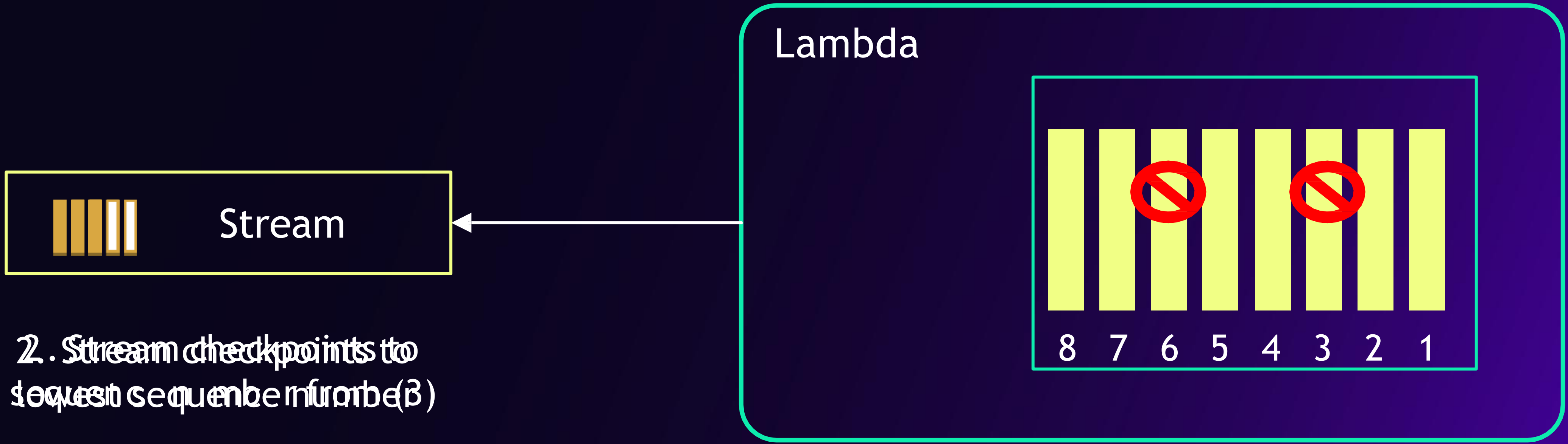
3. If the Lambda returns successfully then the Lambda service advances to the next set of records and repeats #1

4. If the Lambda errors, the behavior depends on the event source configuration



Error handling – ReportBatchItemFailures

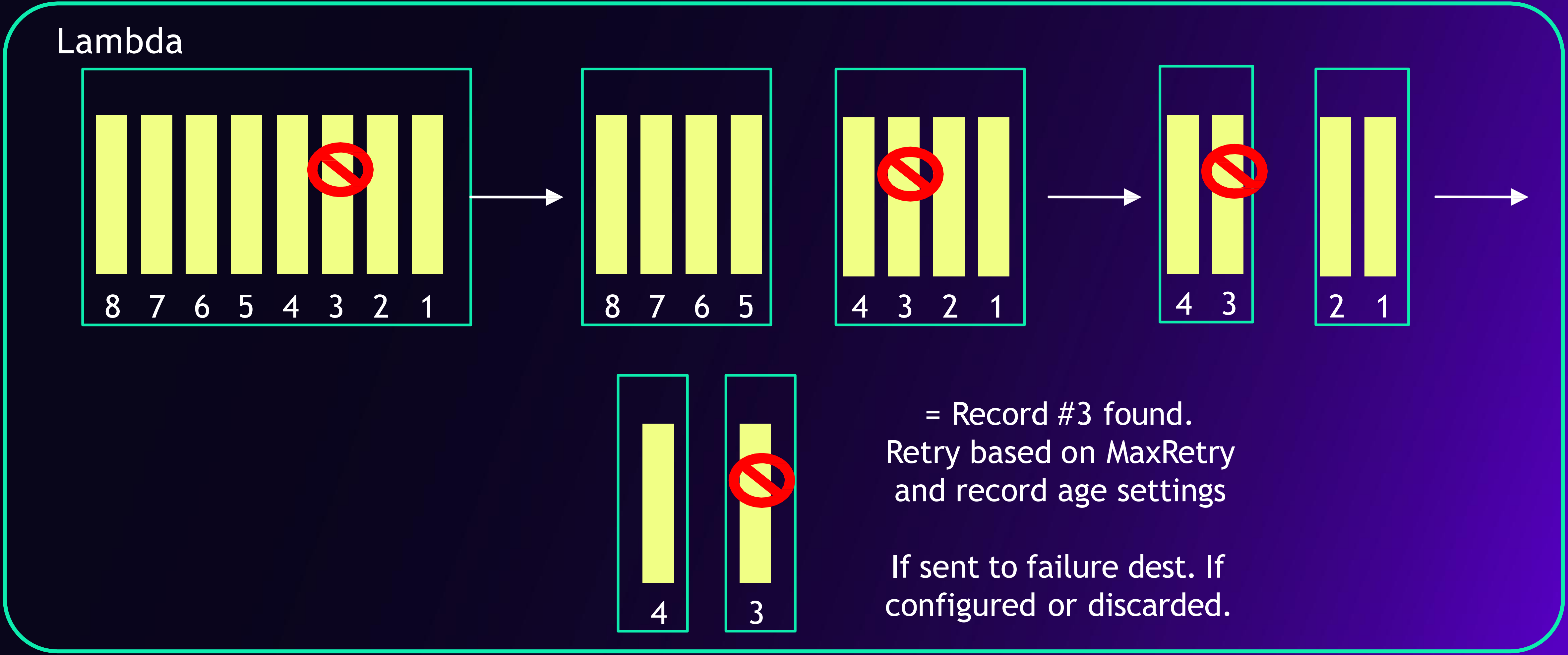
1. Report batch item failure (3,6)



2. Stream checkpoints to lowest sequence number (3)

3. Poll records from updated checkpoint

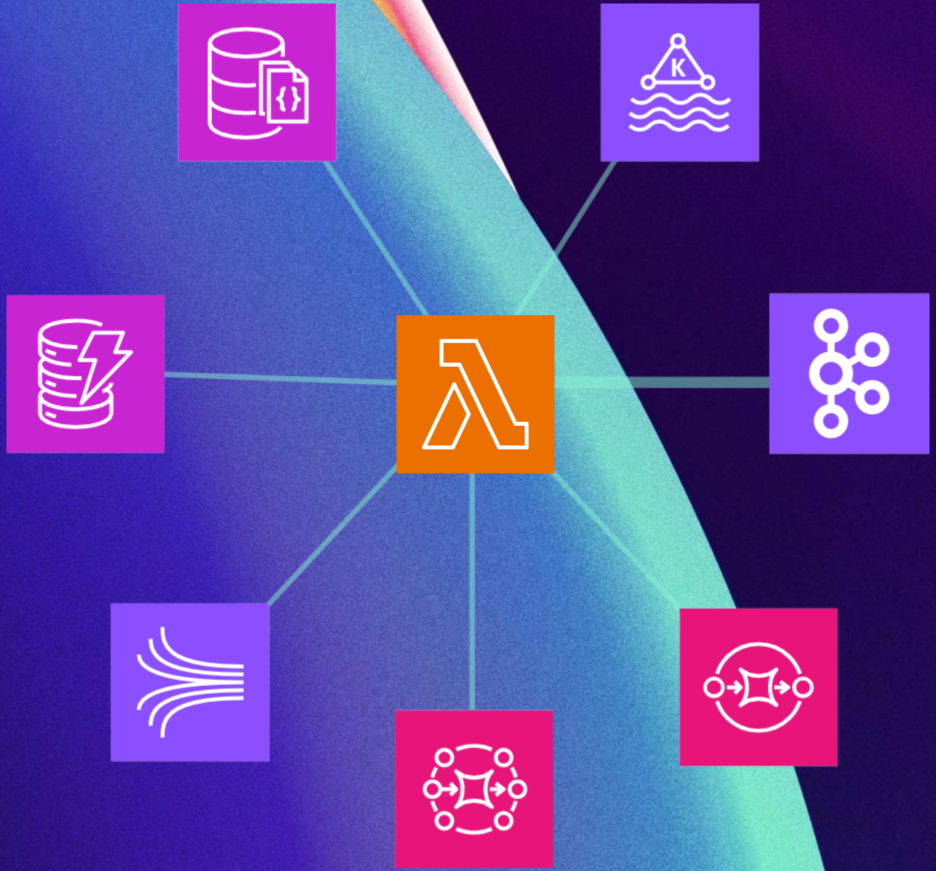
Error handling – BisectBatchOnFunctionError



= Record #3 found.
Retry based on MaxRetry
and record age settings

If sent to failure dest. If
configured or discarded.

Service integration breakdown



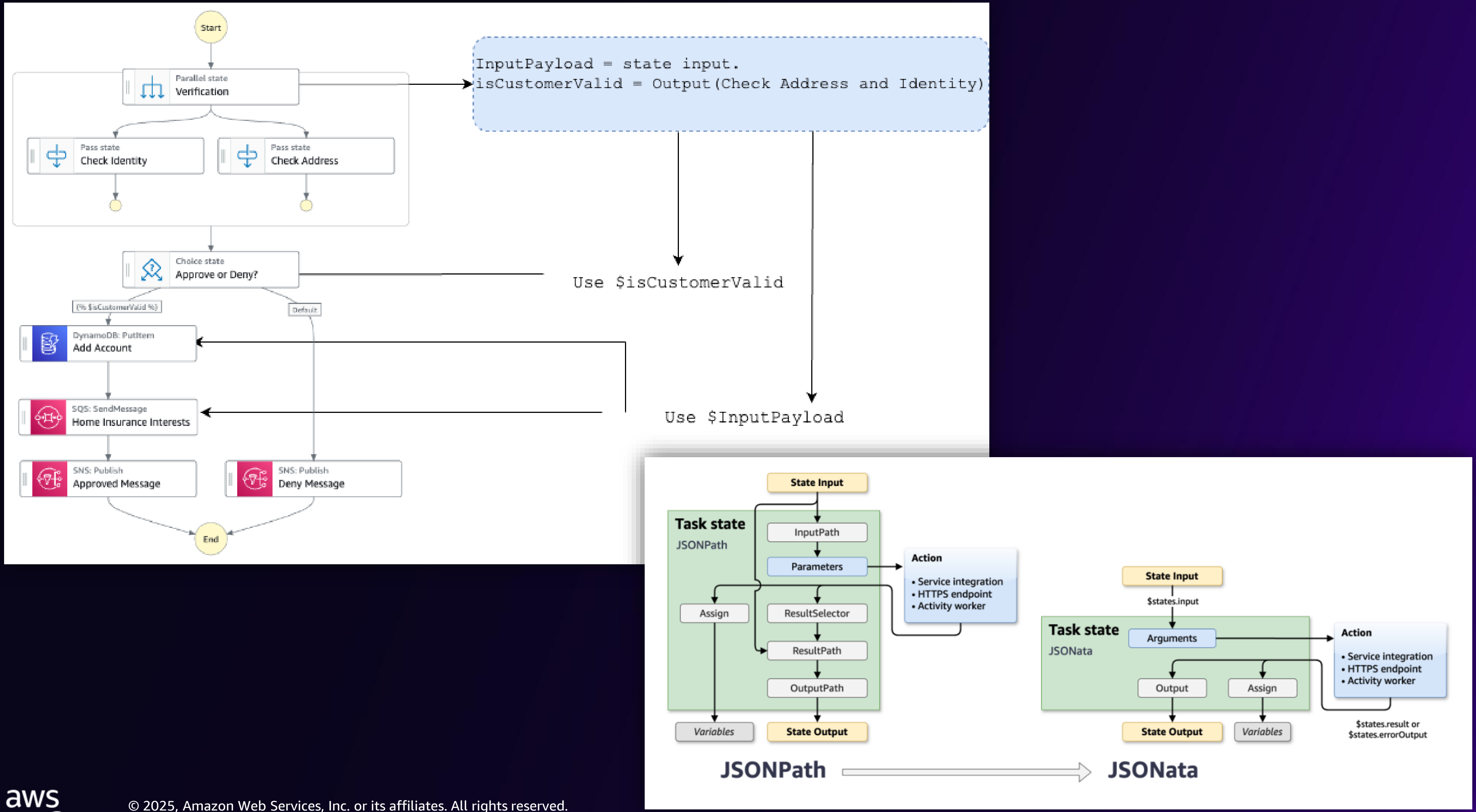
Services

- Amazon DocumentDB
- Amazon DynamoDB Amazon Kinesis Amazon MQ

- Amazon Managed Streaming for Apache Kafka (Amazon MSK)
- Self-managed Apache Kafka
- Amazon Simple Queue Service (Amazon SQS)

AWS Step Functions: JSONata and Variables

SIMPLIFY STATE PAYLOAD MANAGEMENT AND DATA TRANSFORMATION IN STATE MACHINES



GA

Nov, 22nd

pre-re:Invent

All Regions

AWS Compute Blog Post

Provisioned Mode for Event Source Mapping

PROVISIONING EVENT POLLING RESOURCES FOR EVENT SOURCE MAPPING (ESM) IN AWS LAMBDA

GA

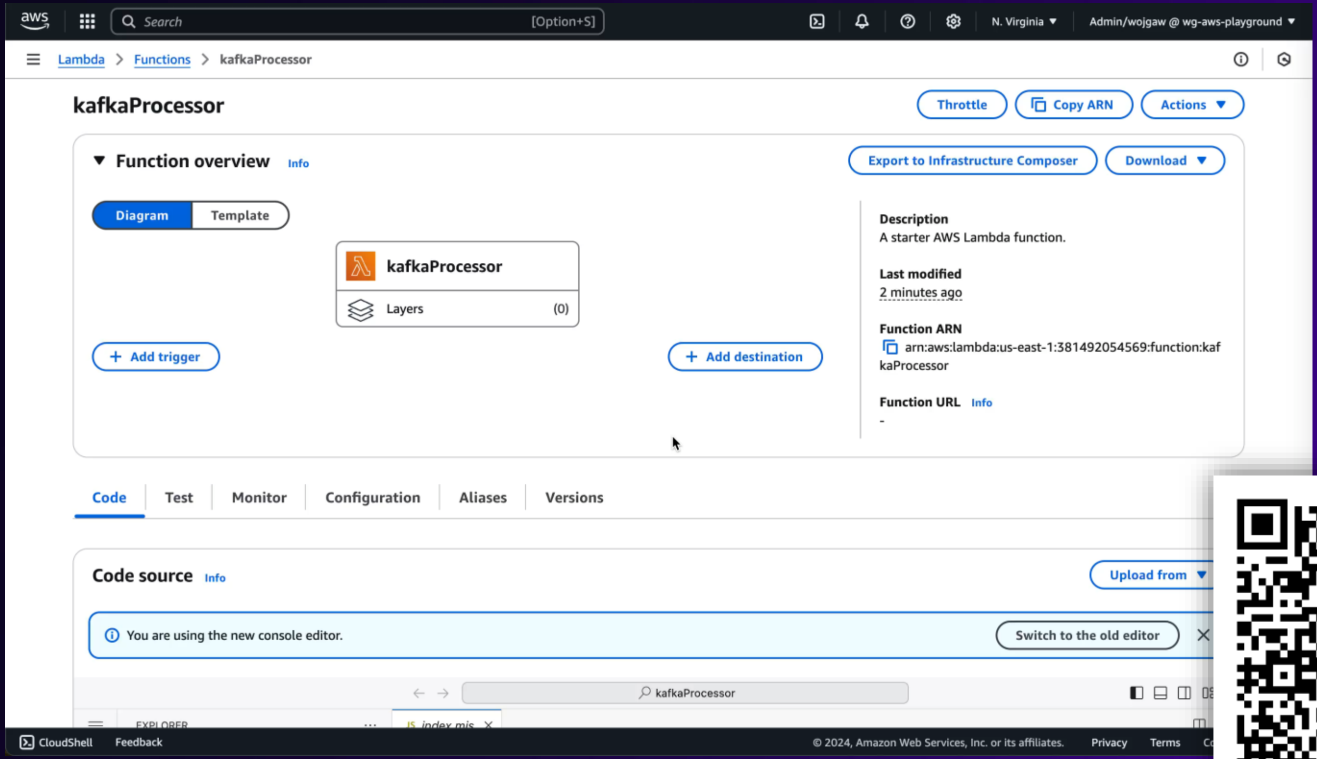
Nov, 22nd

pre-re:Invent

9 Regions

AWS Lambda announces Provisioned Mode for event source mappings (ESMs) that subscribe to Apache Kafka event sources.

This feature that allows you to optimize the throughput of your Kafka ESM by provisioning event polling resources that remain ready to handle sudden spikes in traffic.



kafkaProcessor

Description
A starter AWS Lambda function.

Last modified
2 minutes ago

Function ARN
arn:aws:lambda:us-east-1:381492054569:function:kafkaProcessor

Function URL Info
-

Code source Info

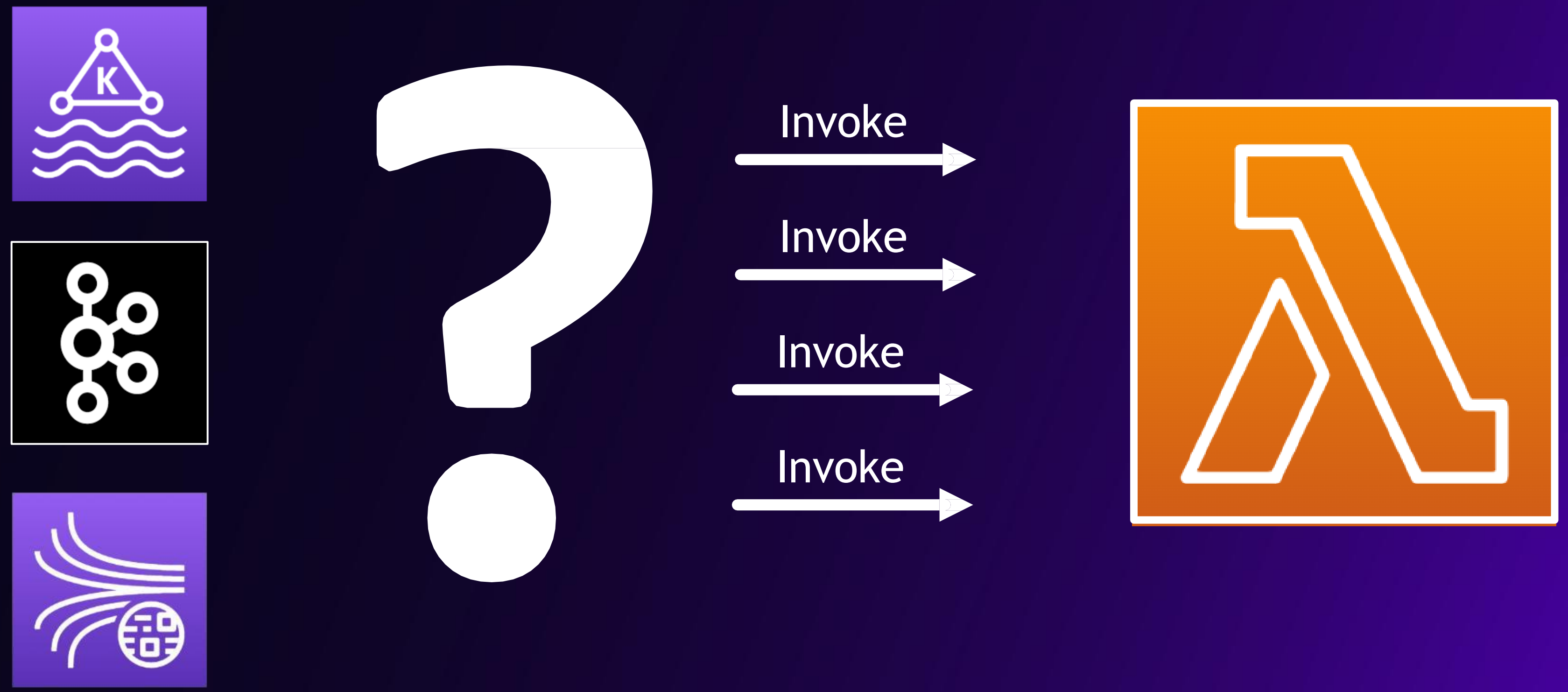
You are using the new console editor.

kafkaProcessor

© 2024, Amazon Web Services, Inc. or its affiliates.

AWS Announcement

Event sources



Amazon Managed Streaming for Apache Kafka

Apache Kafka

Amazon Kinesis Data Streams

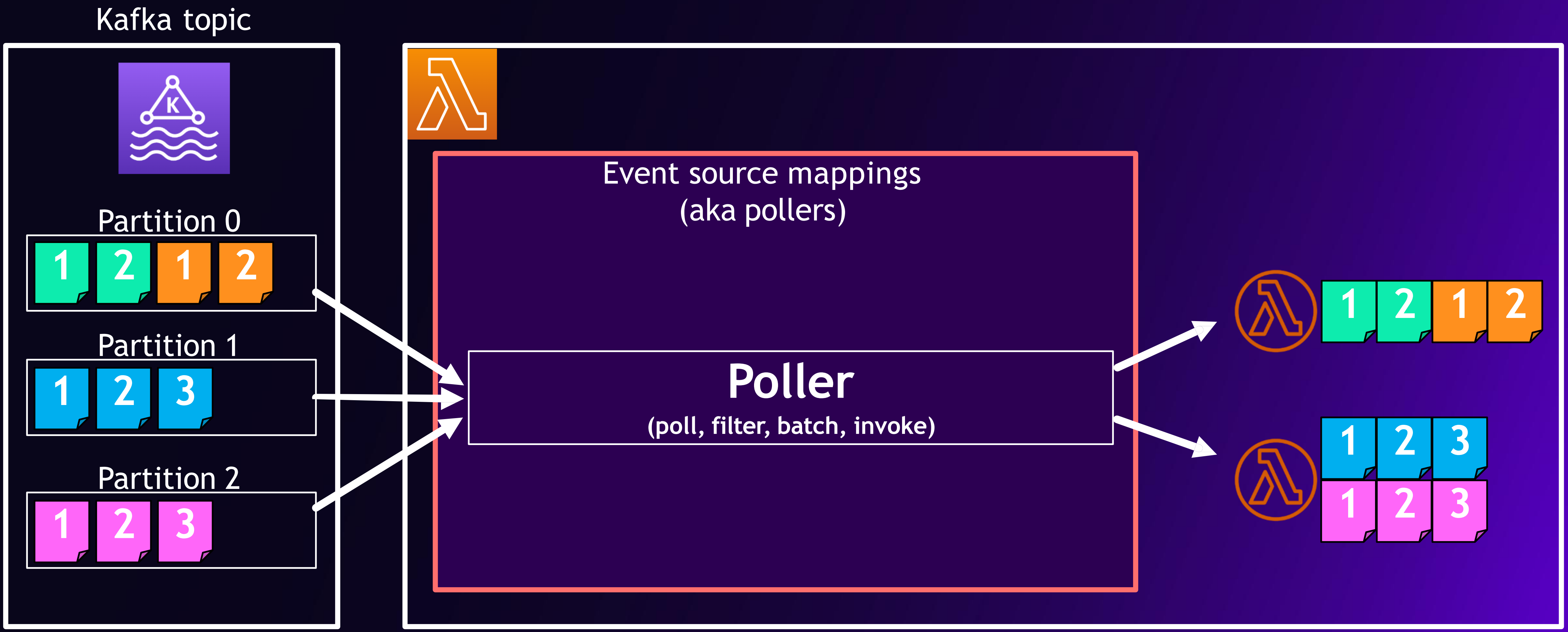
Invoke

Invoke

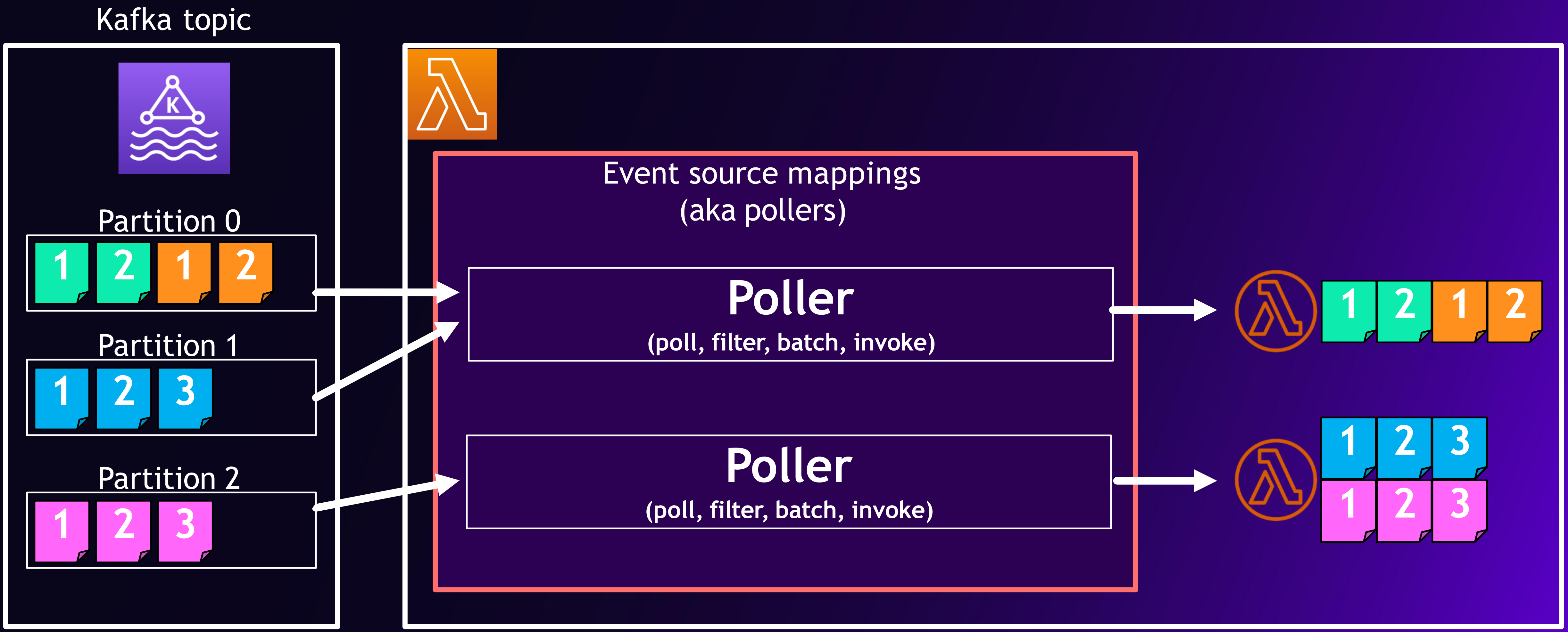
Invoke

Invoke

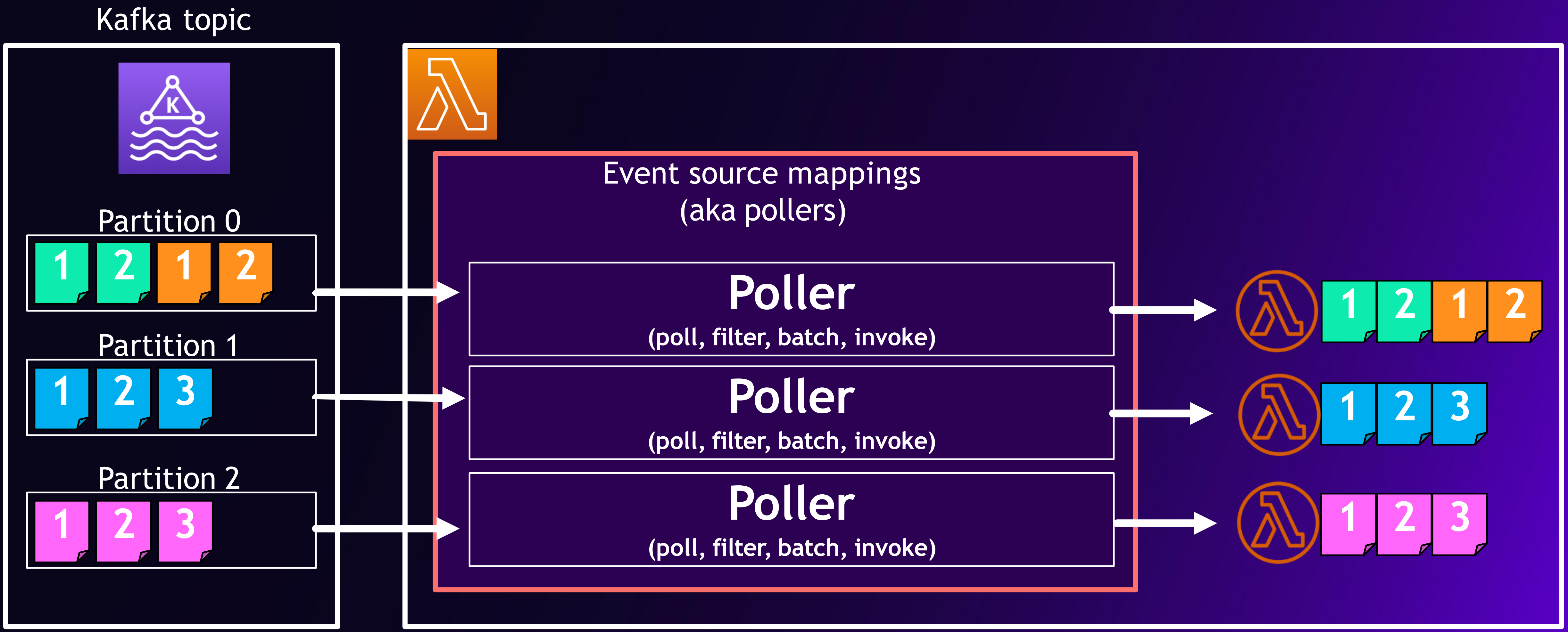
Consuming Kafka with Lambda - scaling



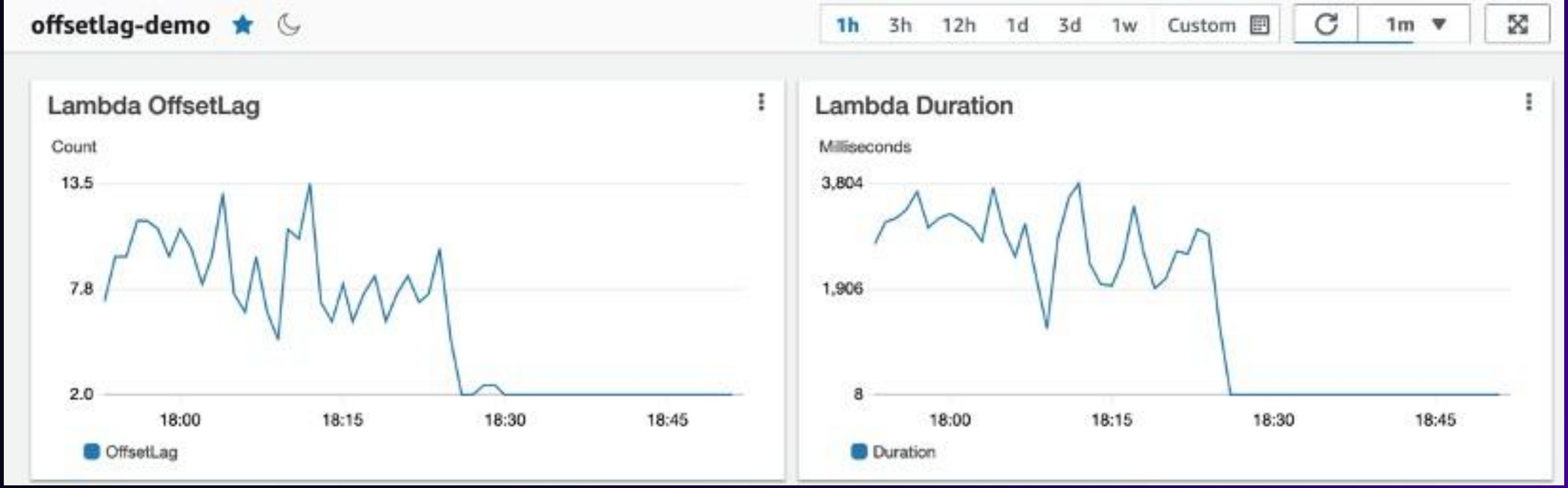
Consuming Kafka with Lambda - scaling



Consuming Kafka with Lambda - scaling



Kafka monitoring



PartitionCount
BytesInPerSec
BytesOutPerSec
MaxOffsetLag
OffsetLag

Throttles
Duration
ConcurrentExecutions
ClaimedAccountConcurrency
OffsetLag

https://docs.aws.amazon.com/lambda/latest/dg/monitoring-metrics.html

But what if…

“My Kafka workload is very spiky, latency sensitive, and requires faster, predictable performance”

Announcing Provisioned Mode for Kafka ESM

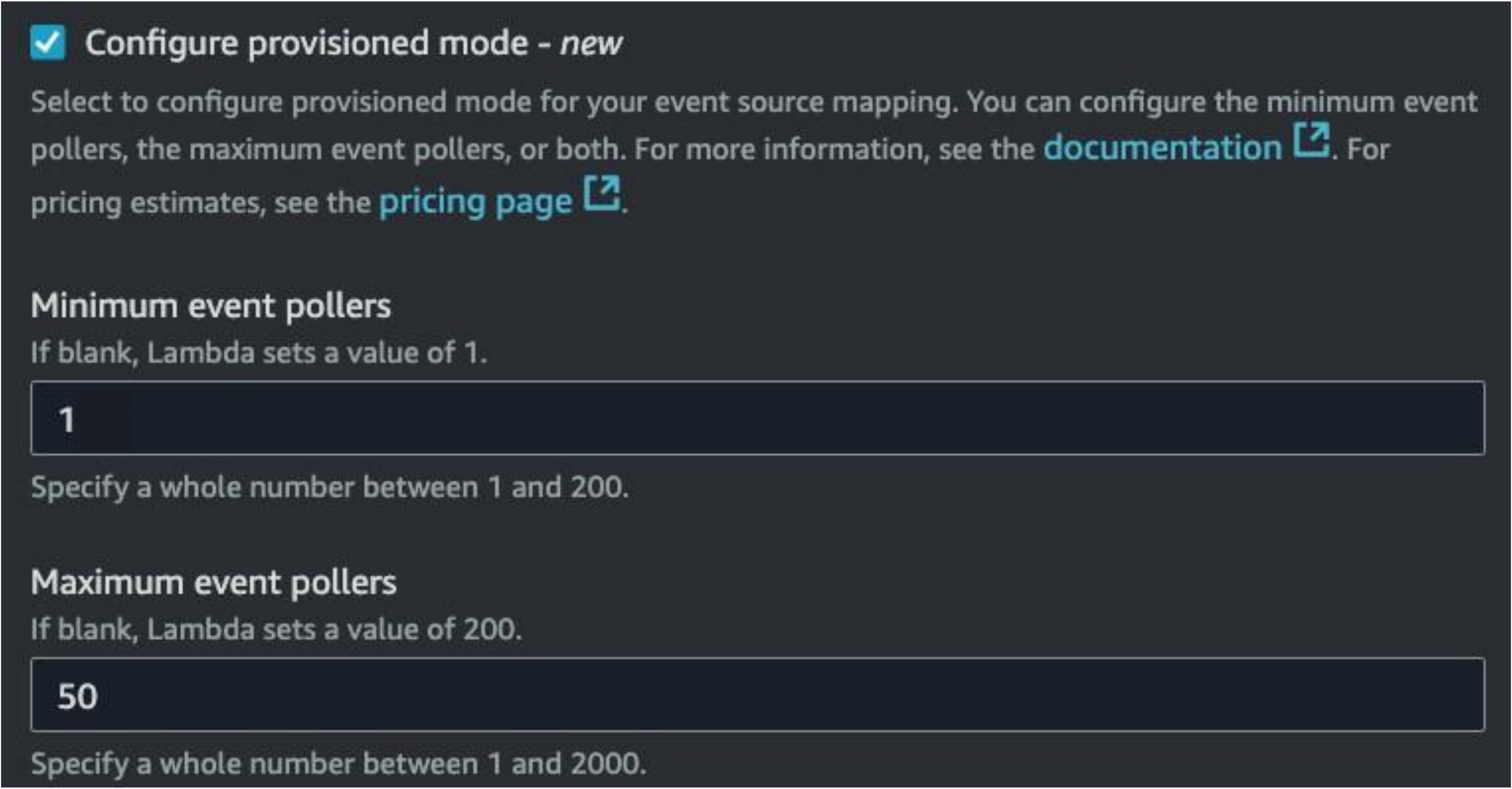
NEW

Configurable minimum and maximum number of always-on event pollers

Faster scaling, great for latency-sensitive workloads

Announcing Provisioned Mode for Kafka ESM

NEW



☑ Configure provisioned mode - *new*

Select to configure provisioned mode for your event source mapping. You can configure the minimum event pollers, the maximum event pollers, or both. For more information, see the documentation ↗. For pricing estimates, see the pricing page ↗.

Minimum event pollers

If blank, Lambda sets a value of 1.

1

Specify a whole number between 1 and 200.

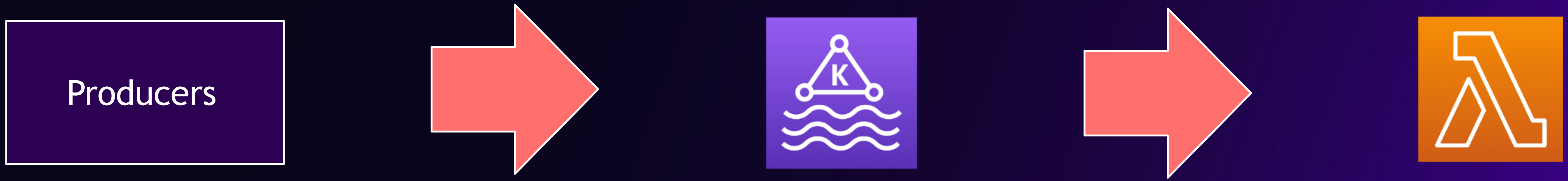
Maximum event pollers

If blank, Lambda sets a value of 200.

50

Specify a whole number between 1 and 2000.

Let’s see the performance difference



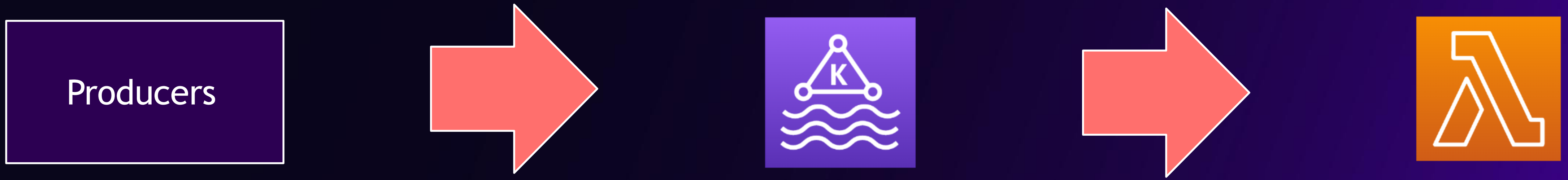
Producers

- Record size 1.5KB
- Random partition key
- Initial traffic – 3,000 records / second
- Traffic spike – 9,000 records / second

- MSK cluster
- 2 brokers
- 1 topic
- 100 partitions

- BatchSize = 50
- Batching window = 1 sec
- Mean duration = 200ms
- Min pollers = 5

Let’s see the performance difference

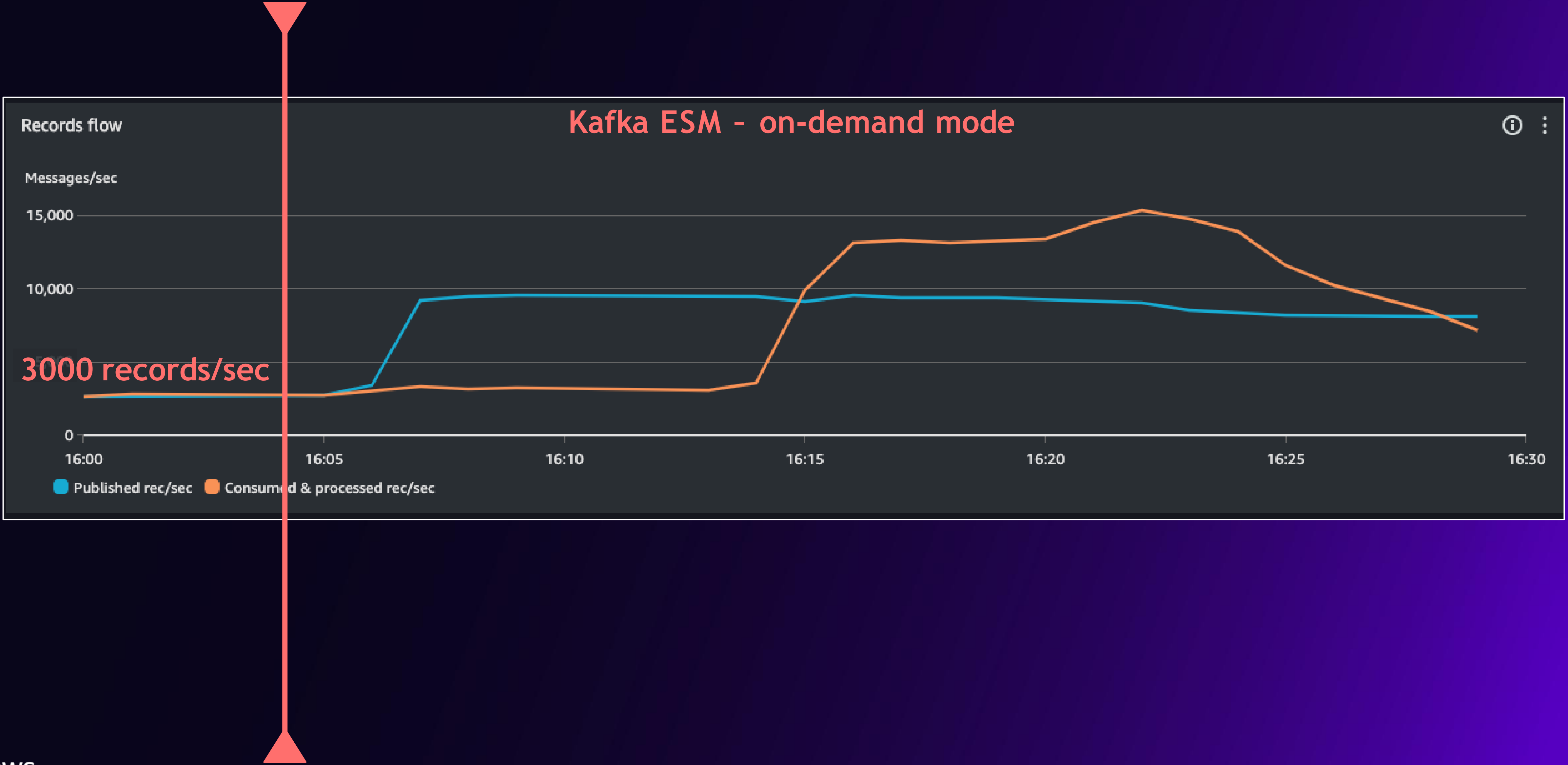


- Record size 1.5KB
- Random partition key
- **Initial traffic - 3,000 records / second**
- **Traffic spike - 9,000 records / second**

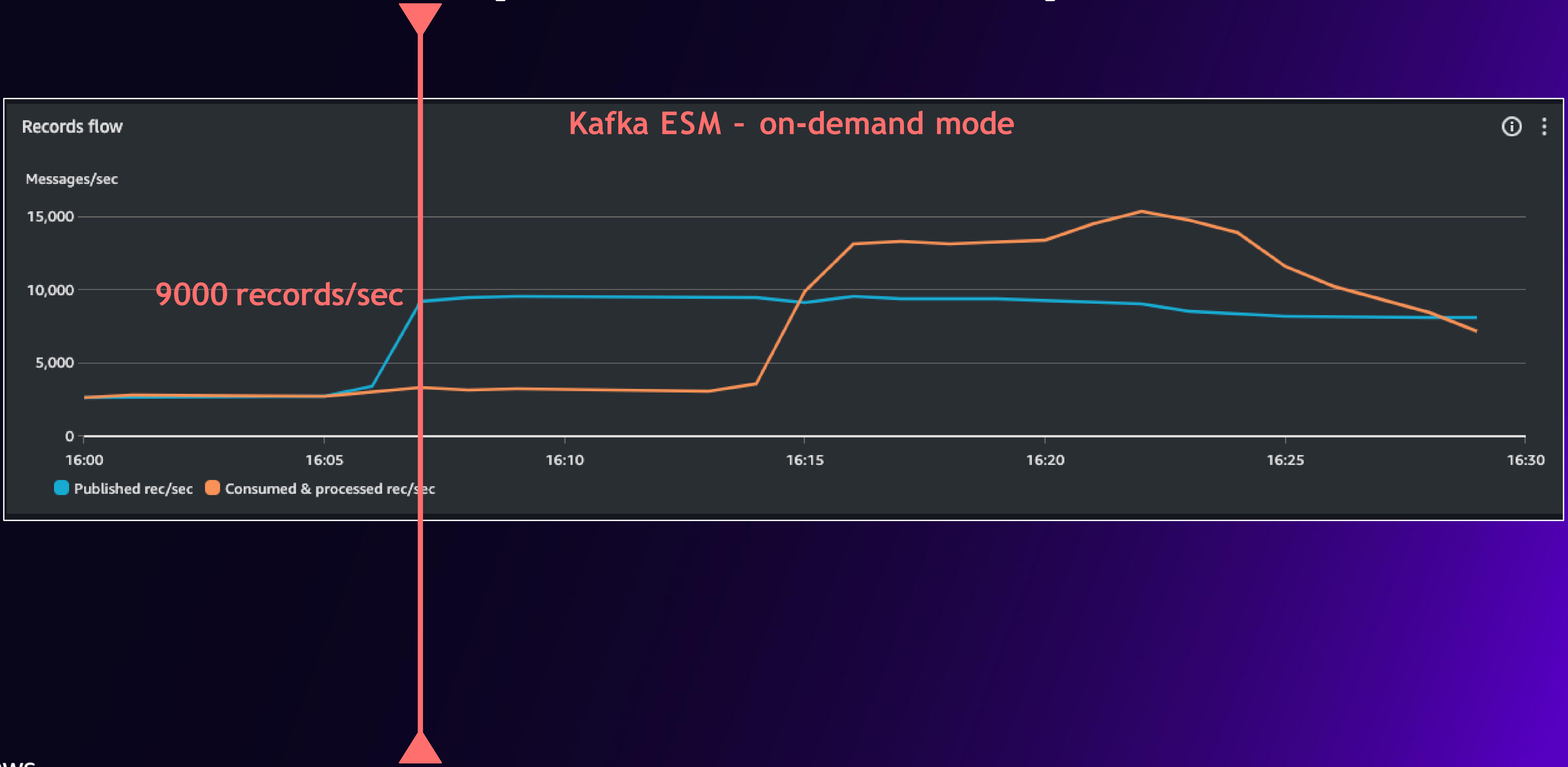
- MSK cluster
- 2 brokers
- 1 topic
- 100 partitions

- BatchSize = 50
- Batching window = 1 sec
- Mean duration = 200ms
- **Min pollers = 5**

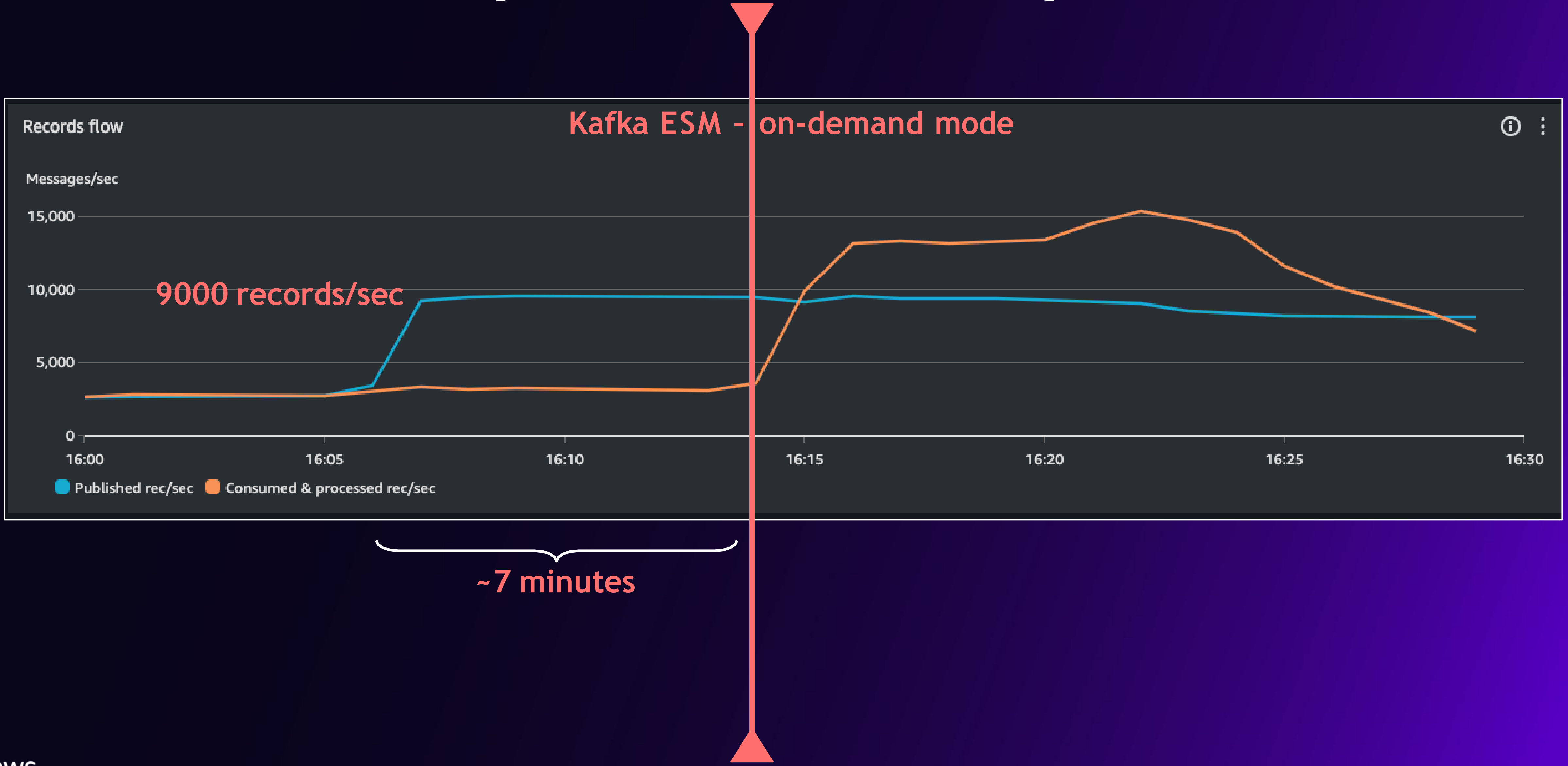
On-demand vs. provisioned ESM performance



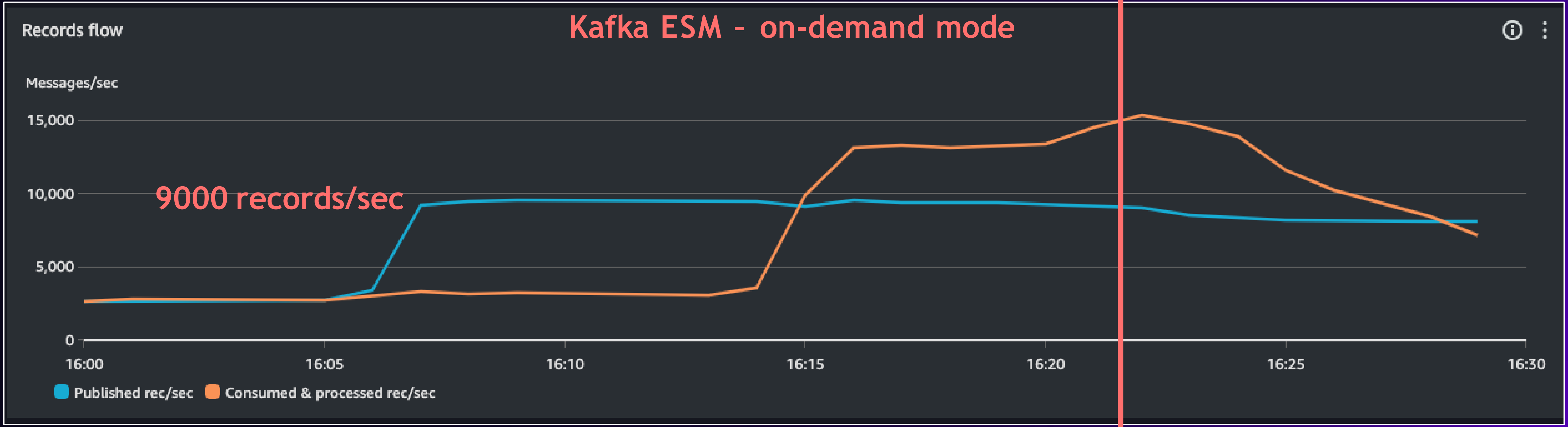
On-demand vs. provisioned ESM performance



On-demand vs. provisioned ESM performance



On-demand vs. provisioned ESM performance



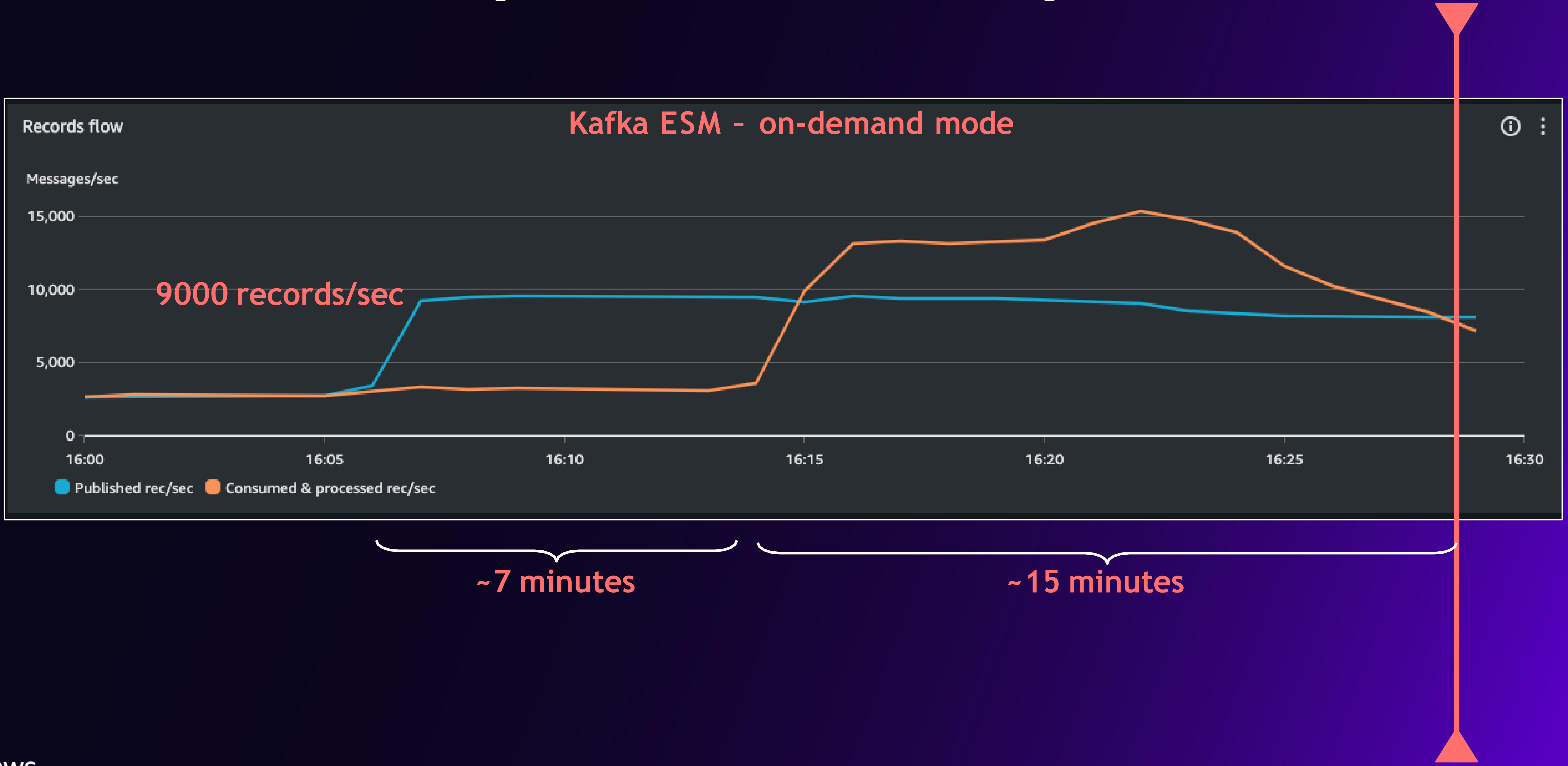
Records flow

Kafka ESM - on-demand mode

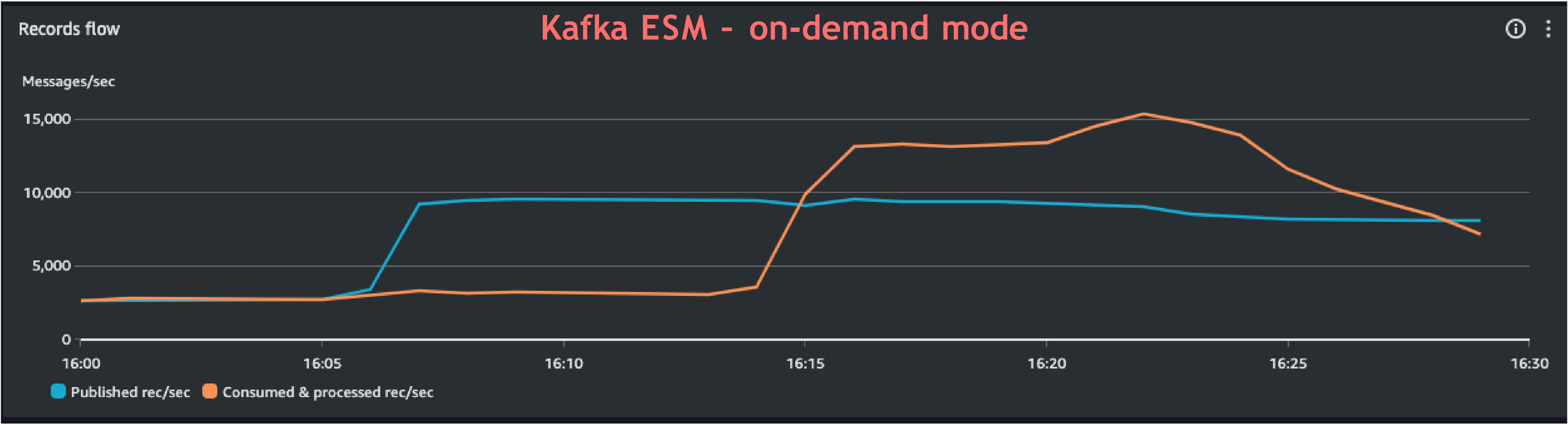
9000 records/sec

~7 minutes

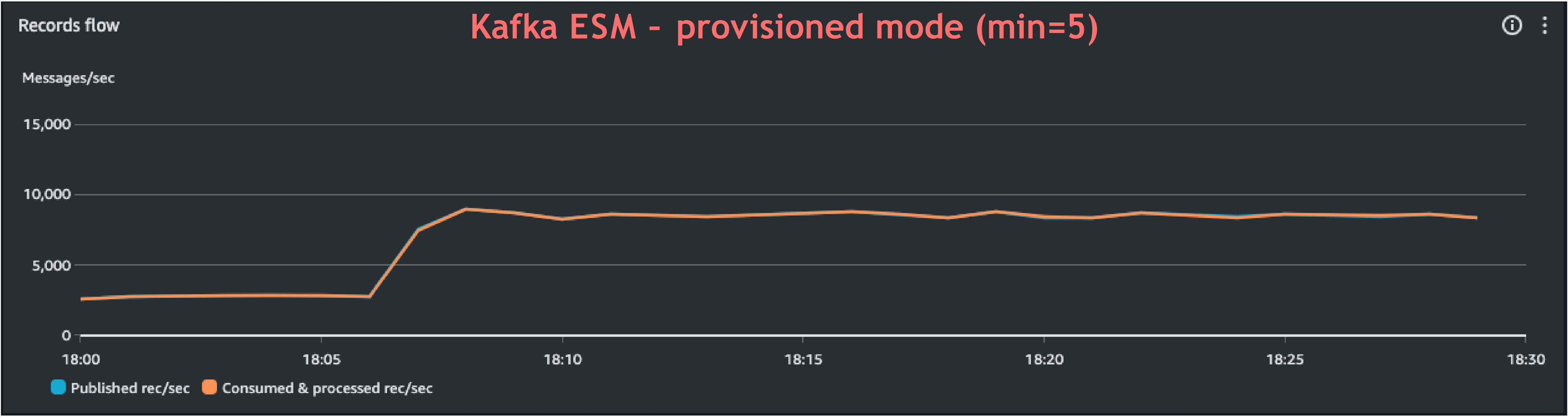
On-demand vs. provisioned ESM performance



On-demand vs. provisioned ESM performance

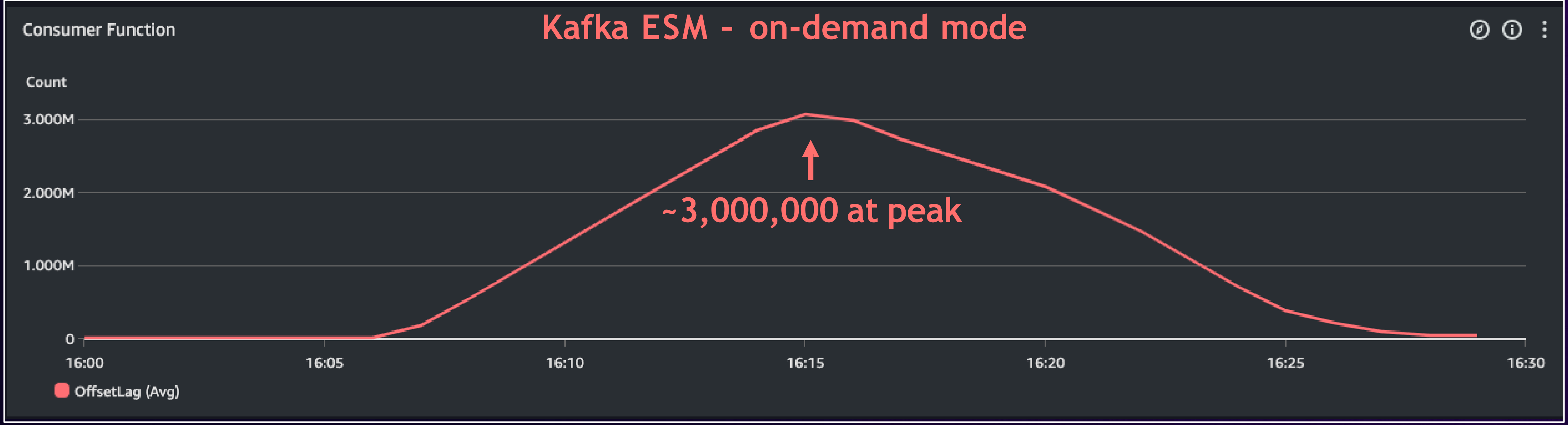


Kafka ESM - on-demand mode



Kafka ESM - provisioned mode (min=5)

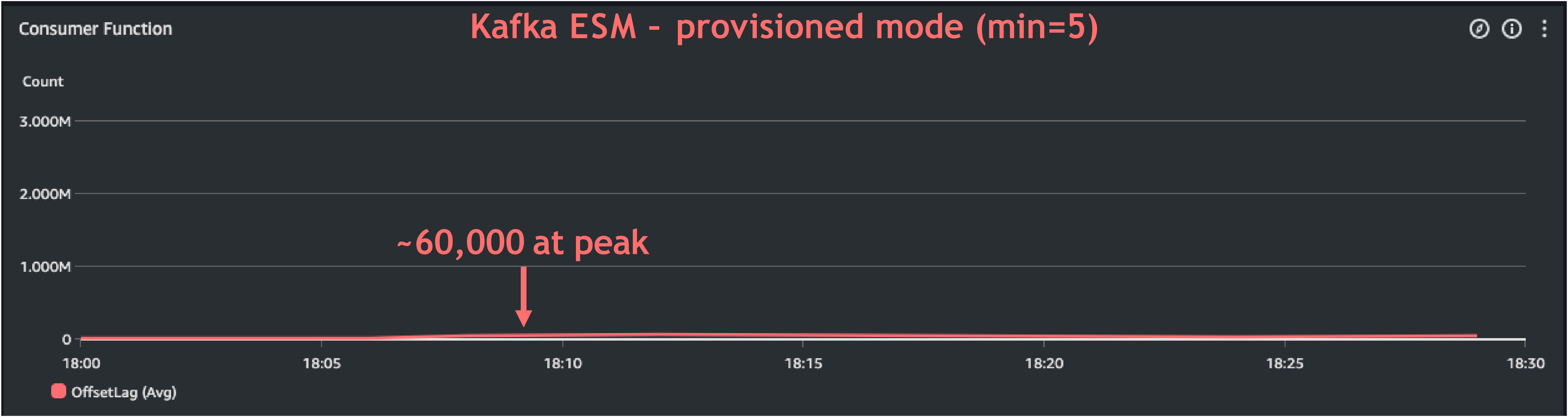
On-demand vs. provisioned ESM performance



Consumer Function

Kafka ESM - on-demand mode

~3,000,000 at peak

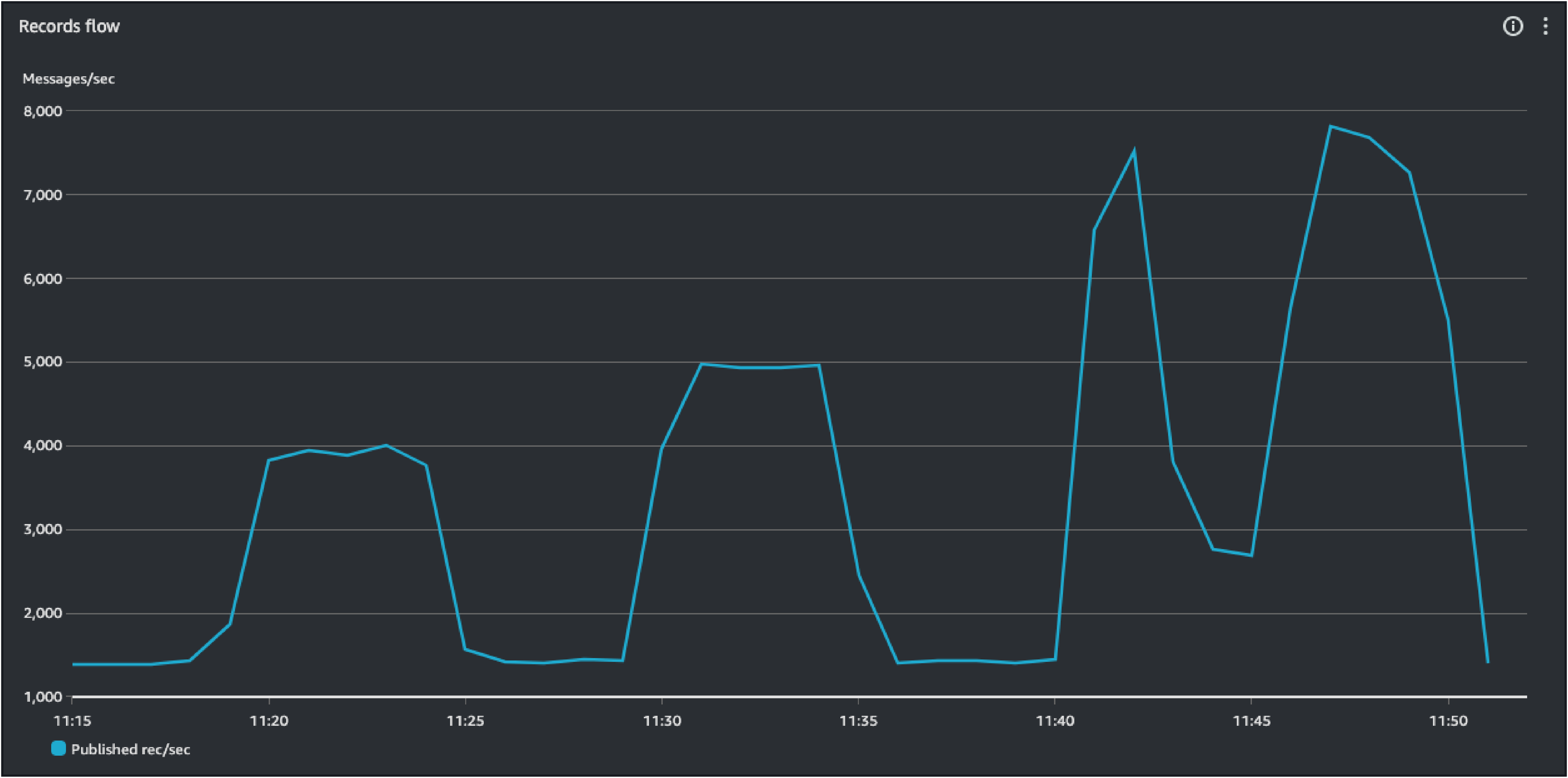


Consumer Function

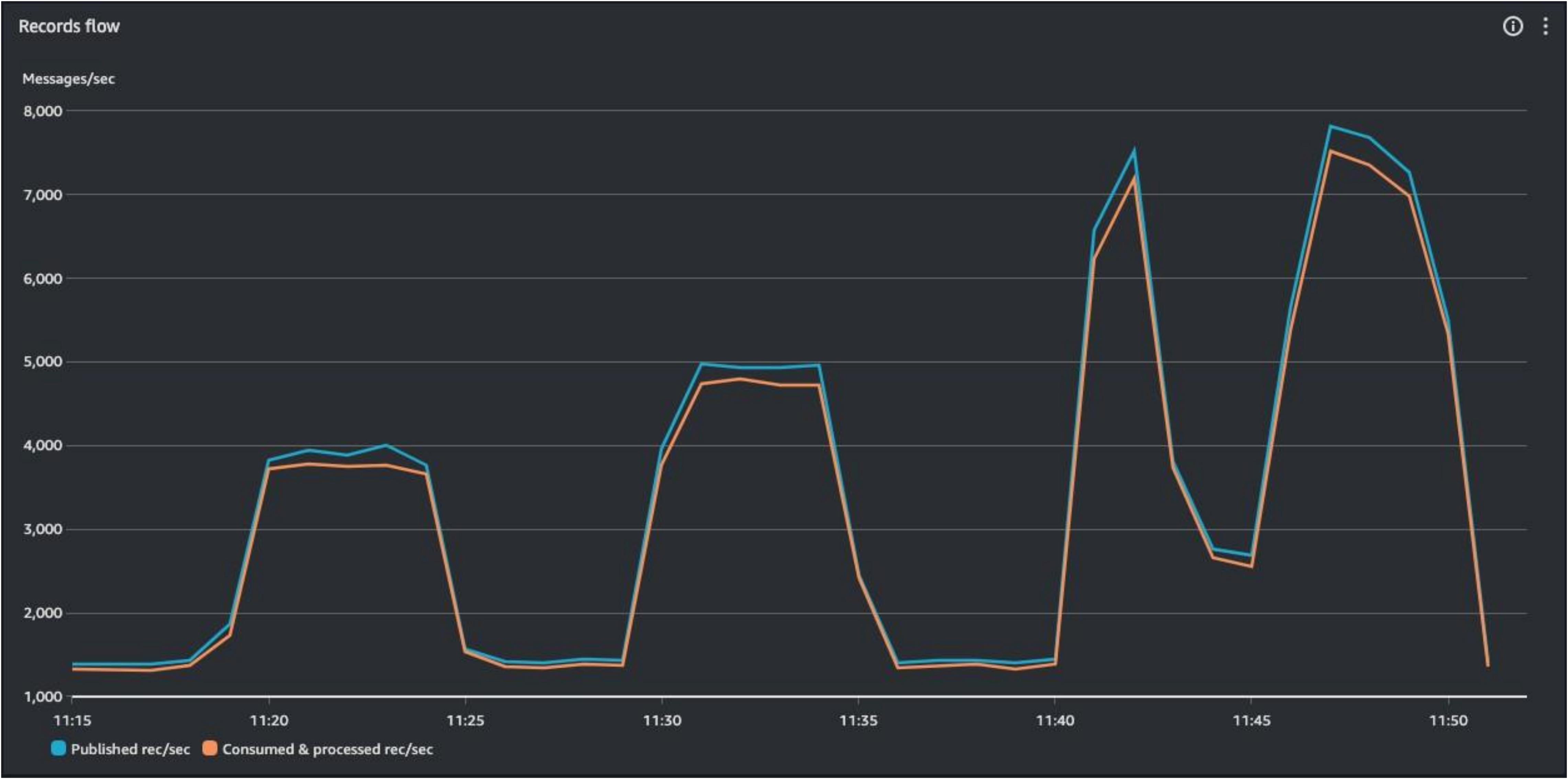
Kafka ESM - provisioned mode (min=5)

~60,000 at peak

Remember the spiky workload?



Remember the spiky workload?



Other Selected Announcements

1. **AWS Lambda runtime updates:**
   a) **Support for Python 3.13 (pre-re:Invent)**
   b) **Support for Node.js 22 (pre-re:Invent)**
2. **AWS Lambda now supports AWS Fault Injection Service (FIS) actions (pre-re:Invent)**
3. **S3 support as failed-event destination for all integrations in AWS Lambda (pre-re:Invent)**
4. **AWS AppSync now supports cross account sharing of GraphQL APIs (pre-re:Invent)**
5. **Announcing new APIs for Amazon Location Service Routes, Places, and Maps (pre-re:Invent)**