

AWS re:Invent

NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

API202-R

Decoupled microservices

Dirk Fröhner (he/him)

Principal Solutions Architect
Amazon Web Services

Mithun Mallick (he/him)

Principal Solutions Architect
Amazon Web Services



Agenda

Motivation

Application integration patterns

Use cases: Our labs for today

Your turn: Work on the labs

Resources and call to action

Related sessions

API002	Advanced patterns with Amazon EventBridge
API303	Application integration patterns for microservices
API306	Building event-driven architectures
API307	Designing event-driven integrations using Amazon EventBridge
API308	Are you integrating or building distributed applications?
API312	How to select the right application integration service
SVS306	Serverlesspresso: Building an event-driven application from the ground up
SVS308	AWS serverless developer experience: A day in the life of a developer
SVS312	Building Serverlesspresso: Creating event-driven architectures

Motivation





Companies deal with integration scenarios

in many areas, on many layers

Photo: Adobe Stock #299286544



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

“In modern cloud applications, **integration isn’t an afterthought.
It’s an **integral part** of the **application architecture** and the software delivery lifecycle.”**

Gregor Hohpe

*Author of **Enterprise Integration Patterns**, **Cloud Strategy**, and **The Software Architect Elevator***

Potential drawbacks of synchronous systems

- Synchronous systems are inherently **tightly coupled**
- **Problems** in **downstream** systems can have immediate **impact** on **upstream** callers
- **Retries** from upstream callers can easily **fan out** and **amplify** problems



Potential drawbacks of synchronous systems

- Synchronous systems are inherently **tightly coupled**
- **Problems** in **downstream** systems can have immediate **impact** on **upstream** callers
- **Retries** from upstream callers can easily **fan out** and **amplify** problems
- ... and some things simply take **too much time to wait** or are **asynchronous by nature**



Application integration patterns



Applica

Gregor Hohpe and Bobby Woolf:

Enterprise Integration Patterns

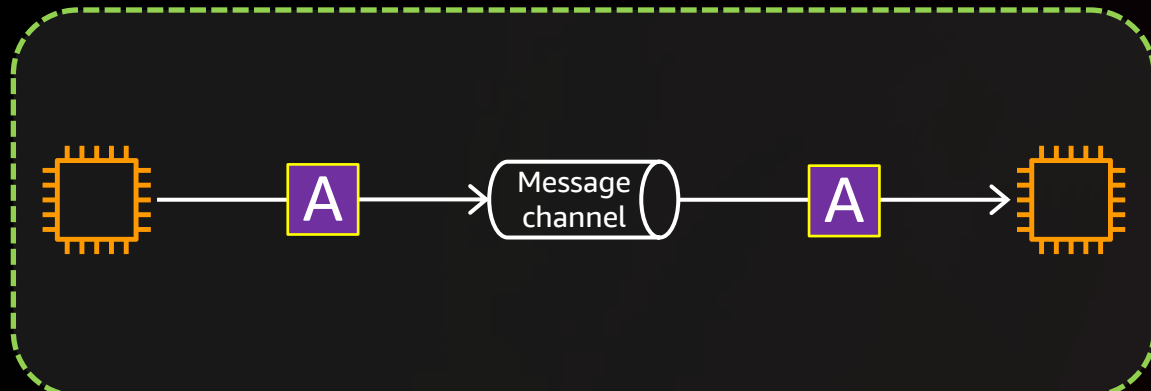
Designing, Building, and Deploying Messaging Solutions



Message exchange

Integration pattern

One-way



Sender

Receiver

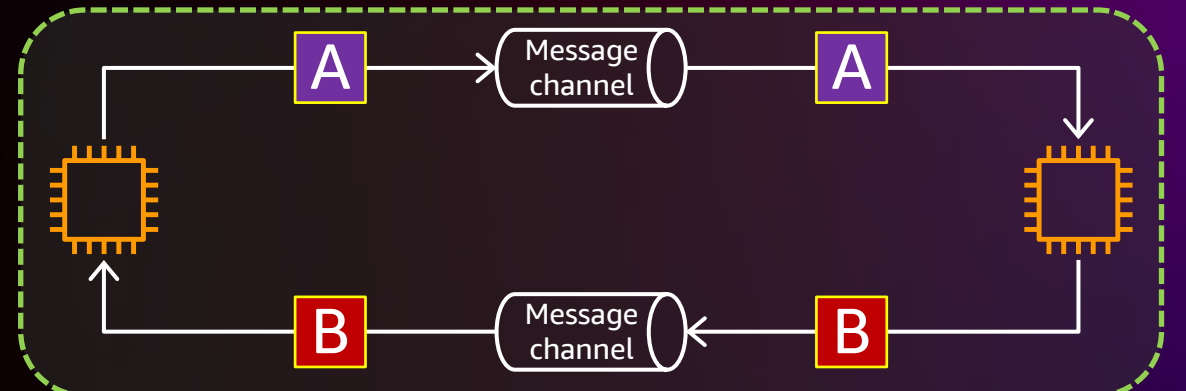
No response expected

Message channel decouples parties

**Message intent:
Command-, document-, event-message**

Conversation pattern

Asynchronous request-response



Requester

Responder

Response expected

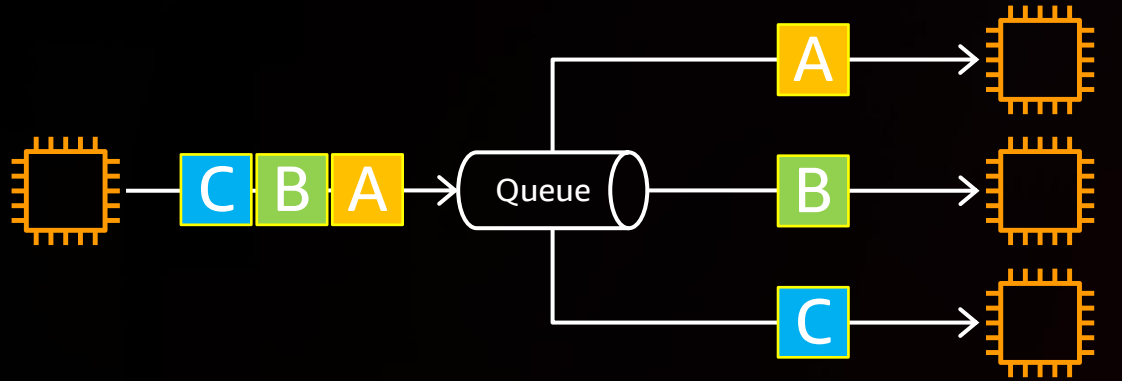
Message channels decouple parties

Return address

Correlation ID

Message channels

Point-to-point (**queue**)



Sender(s)

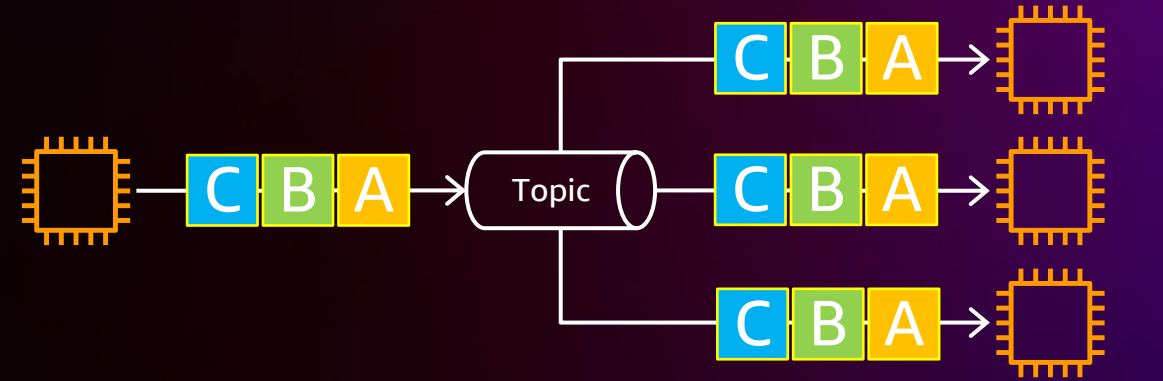
Receivers

Each message consumed by one receiver

Competing
consumers

Buffering
load balancer

Publish-subscribe / fan-out (**topic**)



Publisher(s)

Subscribers

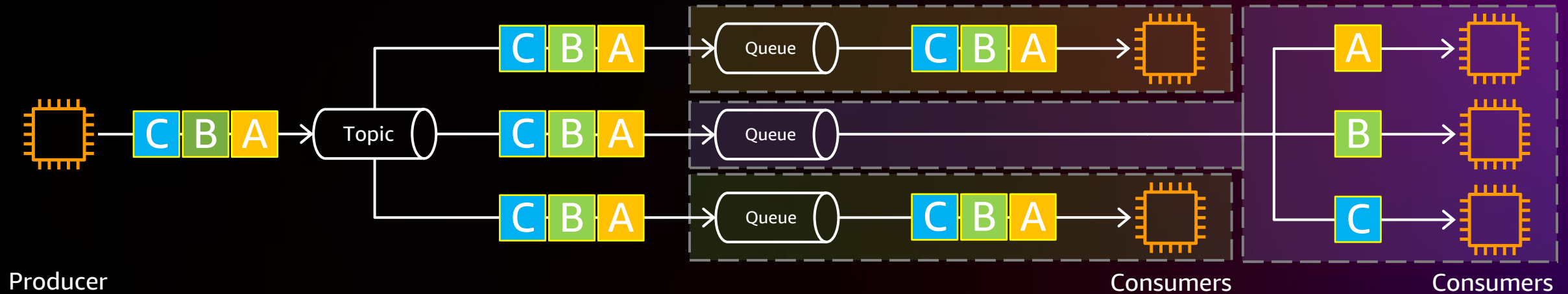
Each message consumed by each subscriber

How to scale
consumers

How to not miss a
message when down

Message channels

Composite pattern: Topic-queue-chaining



Durable subscriber pattern: How to avoid missing messages while not listening?

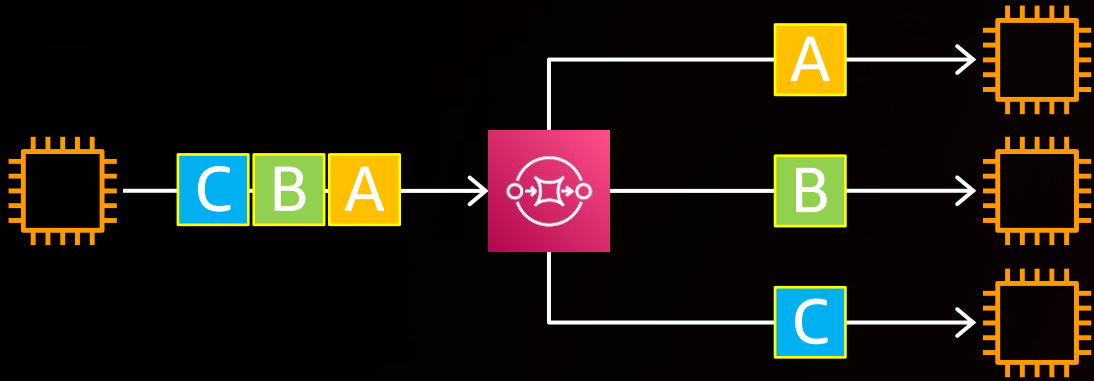
Concurrent consumer pattern: How to scale consumers of publish-subscribe channels?

Best of both worlds: fan-out and consumer scale-out and buffering load balancing all at the same time

Message channels

AWS services implementing
various message channel patterns

Point-to-point (queue)



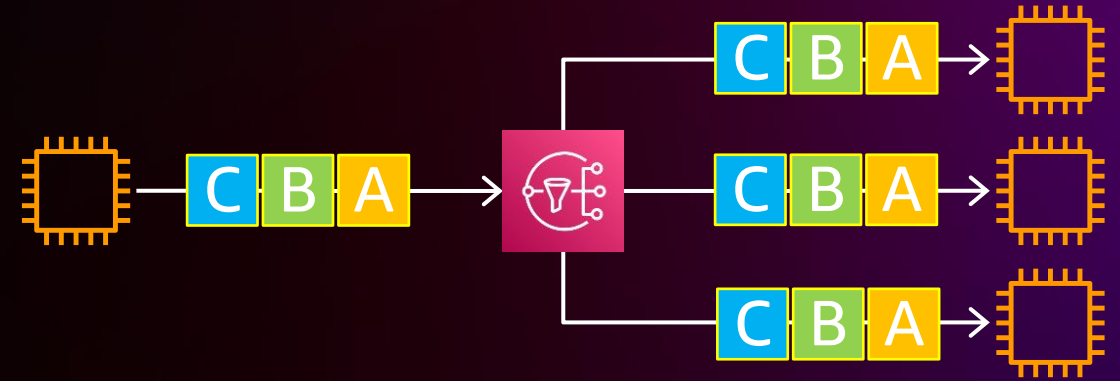
Sender(s)

Receivers

**Amazon Simple Queue Service
(Amazon SQS)**

Cloud-native and serverless

Publish-subscribe / fan-out (topic)



Publisher(s)

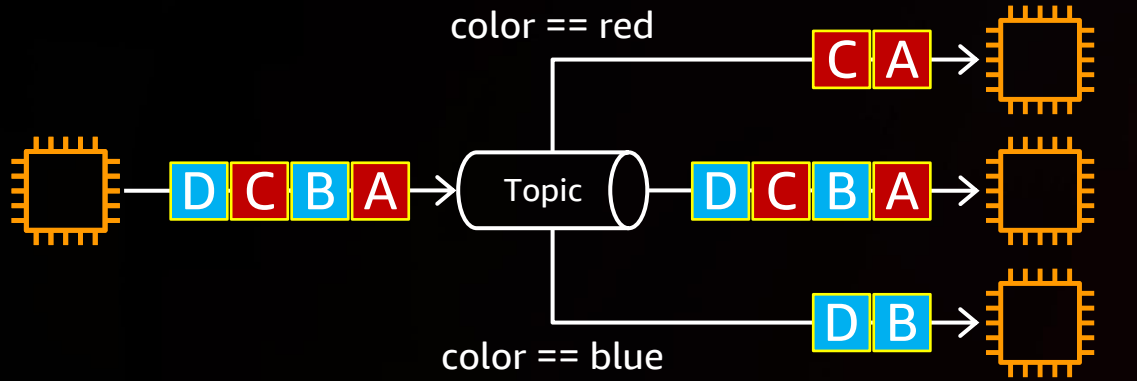
Subscribers

**Amazon Simple Notification Service
(Amazon SNS)**

Cloud-native and serverless

Message routing

Message filter



Publisher(s)

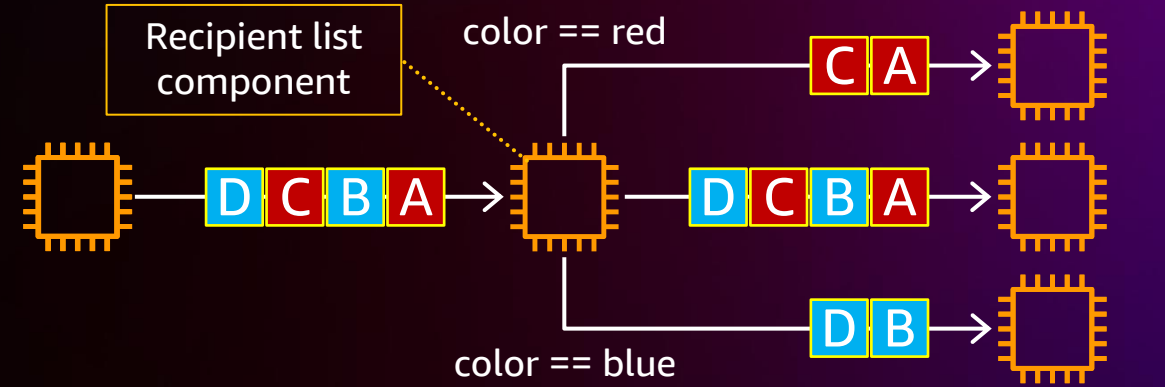
Subscribers

Receive only relevant subset of messages

Controlled by
subscriber

Publisher remains
completely unaware

Recipient list



Publisher(s)

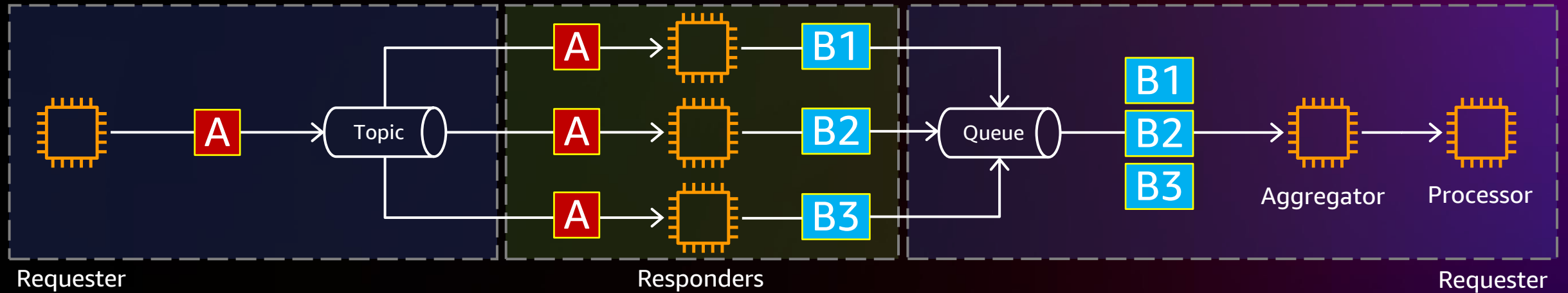
Subscribers

Send only relevant subset of messages
to each subscriber

Controlled directly by publisher
or a separate component

Message routing

Scatter-gather



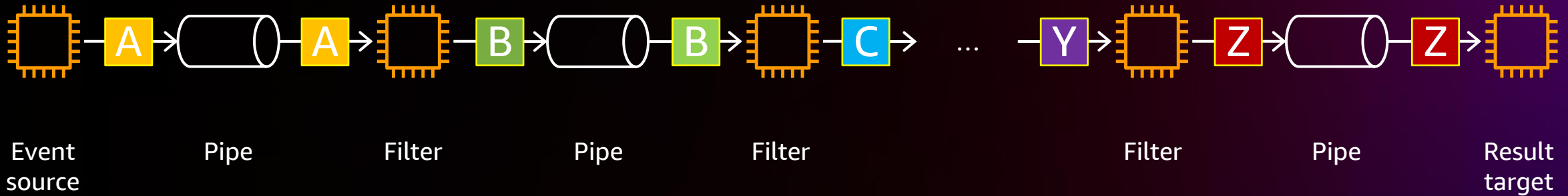
How to distribute a request to relevant/interested parties and capture their individual responses?

For election or parallel processing scenarios (i.e., search for **best** response or **accumulate** responses)

Message routing

Pipes and filters

Flow doesn't have to be linear!



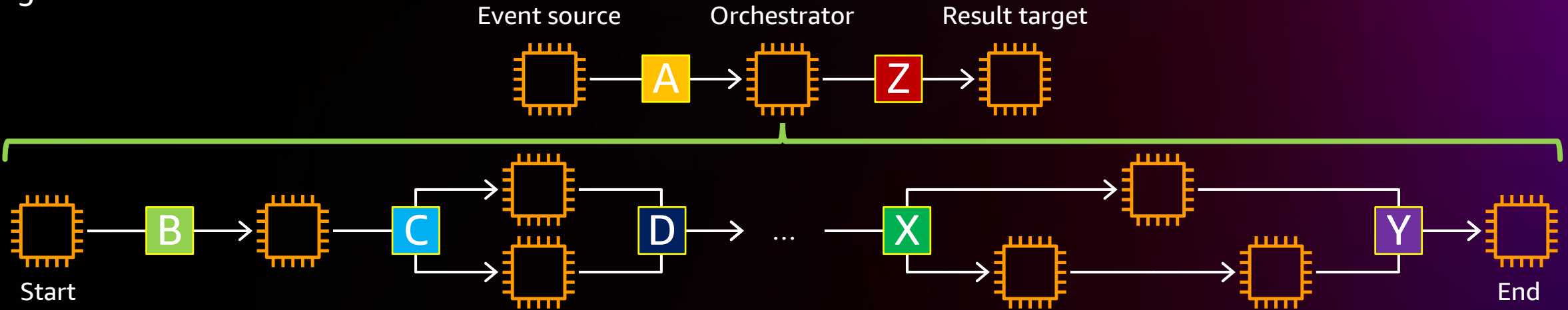
Event triggers chain of processing steps
("filters"), connected by "pipes"

Knowledge of destination or context for next
step is wired into each filter

Similar patterns: chain of responsibility, processing pipeline, saga choreography

Message routing

Saga orchestration



Event triggers orchestrated workflow

Workflow participants remain as loosely coupled as possible

Knowledge of workflow is externalized into orchestrator component

Orchestrator manages branches, retries, and rollbacks into a consistent state

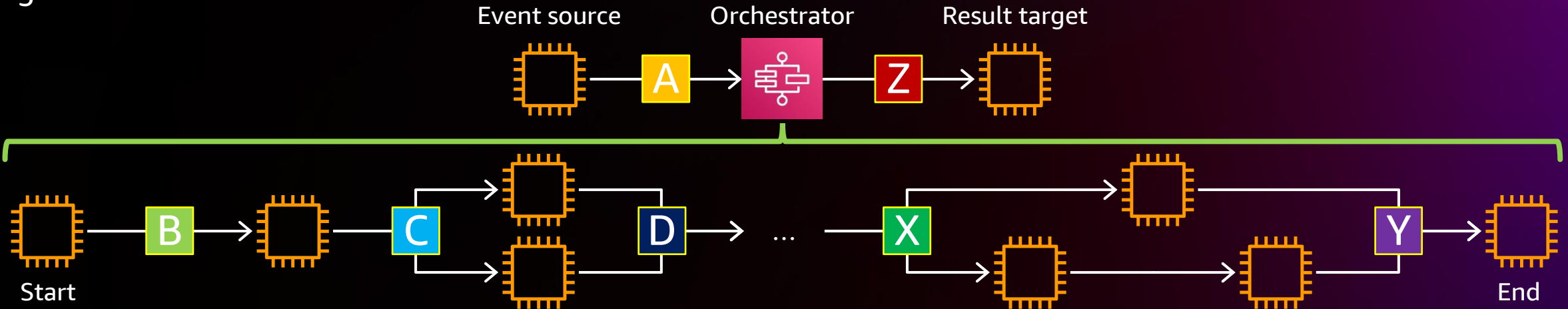
Message routing

AWS service that makes it easy to implement this pattern:



AWS
Step Functions

Saga orchestration



Event triggers orchestrated workflow

Knowledge of workflow is externalized into orchestrator component

Workflow participants remain as loosely coupled as possible

Orchestrator manages branches, retries, and rollbacks into a consistent state

Use cases: Our labs for today



Context: Wild Rydes, Inc.



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Choose your path

We have four labs for you today, plus a common foundation lab.

After introduction of use cases, context, and patterns, you can pick the most relevant labs for you or run through all of them if time permits.

The workshop is available on a public website and you can run the labs individually with your own AWS account anytime later.



Choose your path

Foundation

Lab 0

Lab 1
Fan-out,
Message-filtering

Lab 2
Topic-queue-chaining,
Queues as buffering LBs

Lab 3
Scatter-gather

Lab 4
Saga orchestration



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

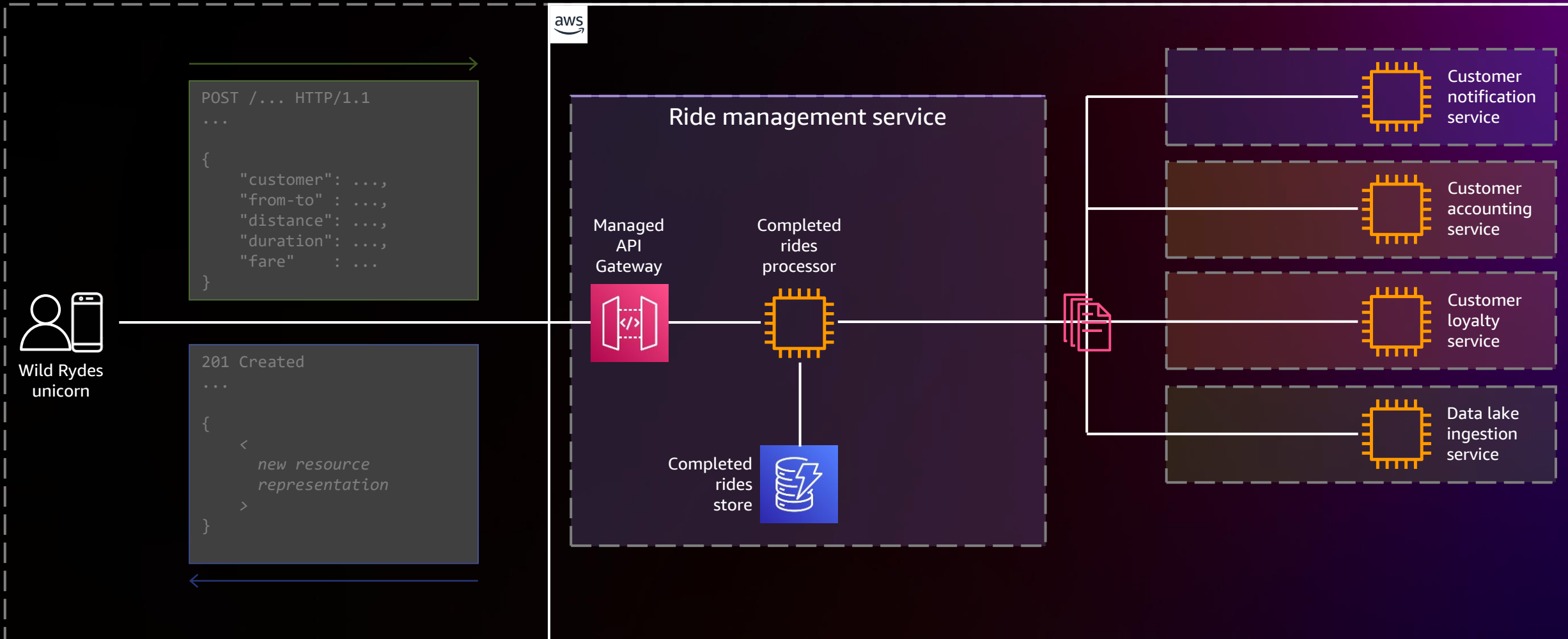


Photo: Dirk Fröhner

Use case: Submit a ride completion Context for labs 1 and 2

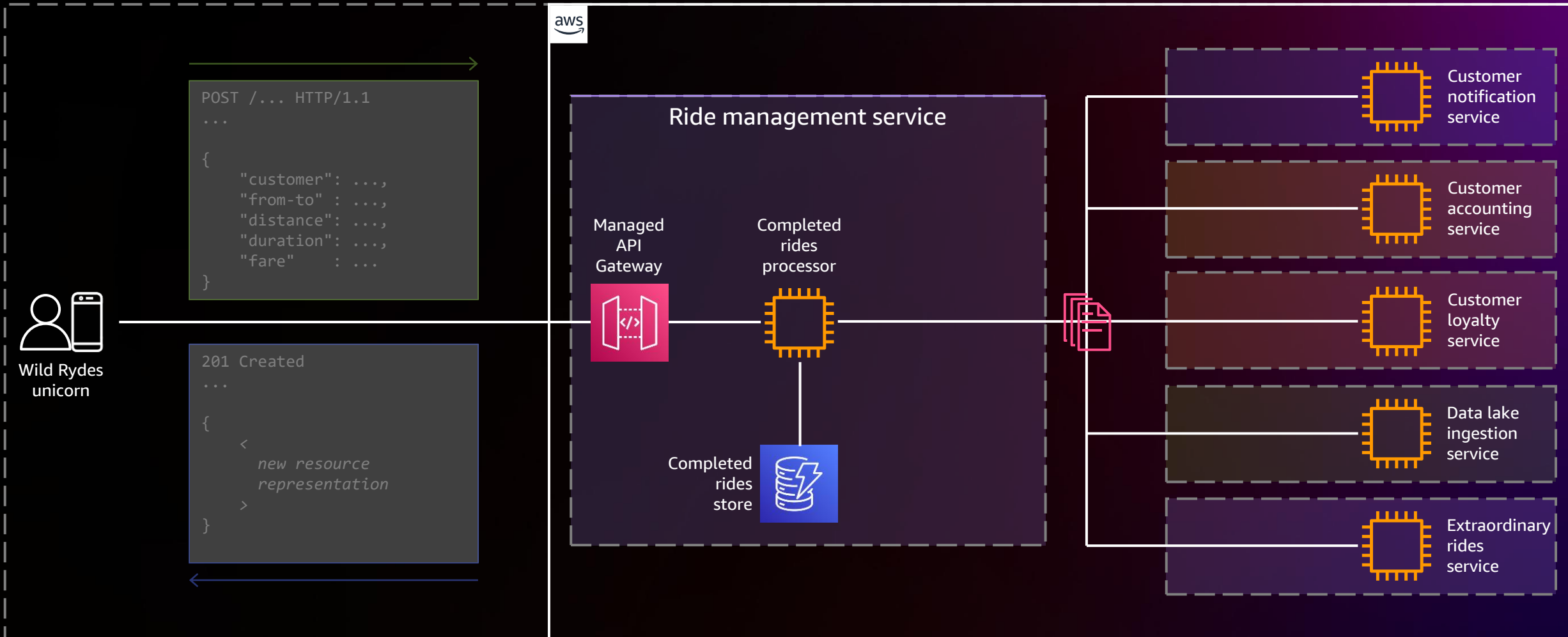
Submit ride completion

USE CASE



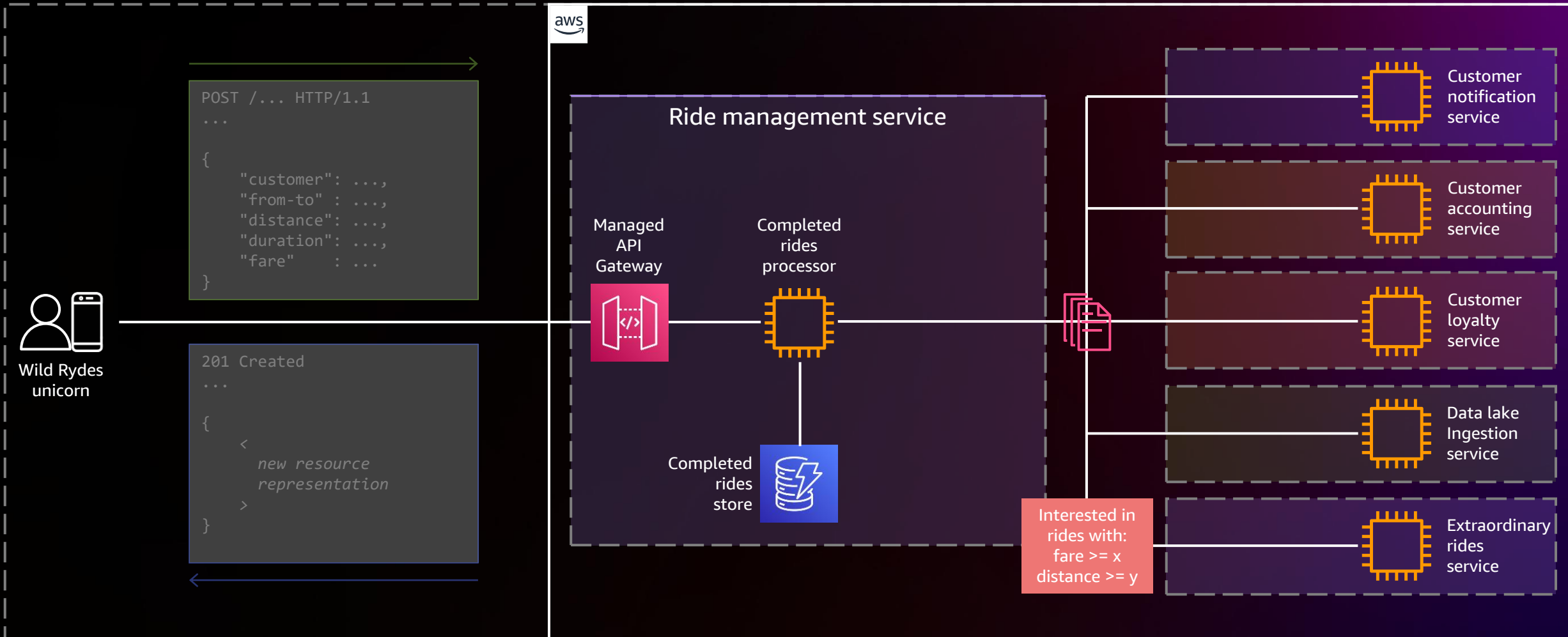
Submit ride completion

USE CASE



Submit ride completion

USE CASE



Submit ride completion

USE CASE

Recipient-list

```
POST /... HTTP/1.1
...
{
  "customer": ...,
  "from-to": ...,
  "distance": ...,
  "duration": ...,
  "fare" : ...
}
```



Wild Rydes unicorn

```
201 Created
...
{
  <
    new resource
    representation
  >
}
```

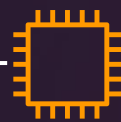


Ride management service

Amazon
API
Gateway

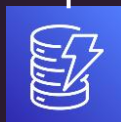


Completed
rides
processor

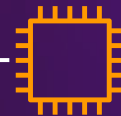


https://...

Completed
rides
store

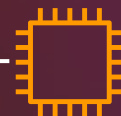


https://...



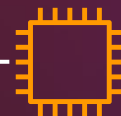
Customer
notification
service

https://...



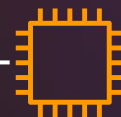
Customer
accounting
service

https://...



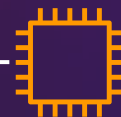
Customer
loyalty
service

https://...



Data lake
ingestion
service

https://...



Extraordinary
rides
service



Submit ride completion

USE CASE

Recipient-list service

```
POST /... HTTP/1.1
...
{
  "customer": ...,
  "from-to": ...,
  "distance": ...,
  "duration": ...,
  "fare": ...
}
```



Wild Rydes unicorn

```
201 Created
...
{
  <
    new resource
    representation
  >
}
```

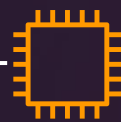


Ride management service

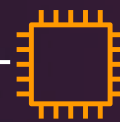
Amazon API Gateway



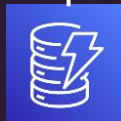
Completed rides processor



Request distribution service

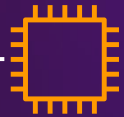


Completed rides store



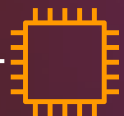
https://...

https://...



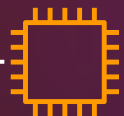
Customer notification service

https://...



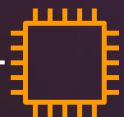
Customer accounting service

https://...



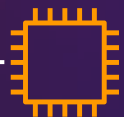
Customer loyalty service

https://...



Data lake ingestion service

https://...



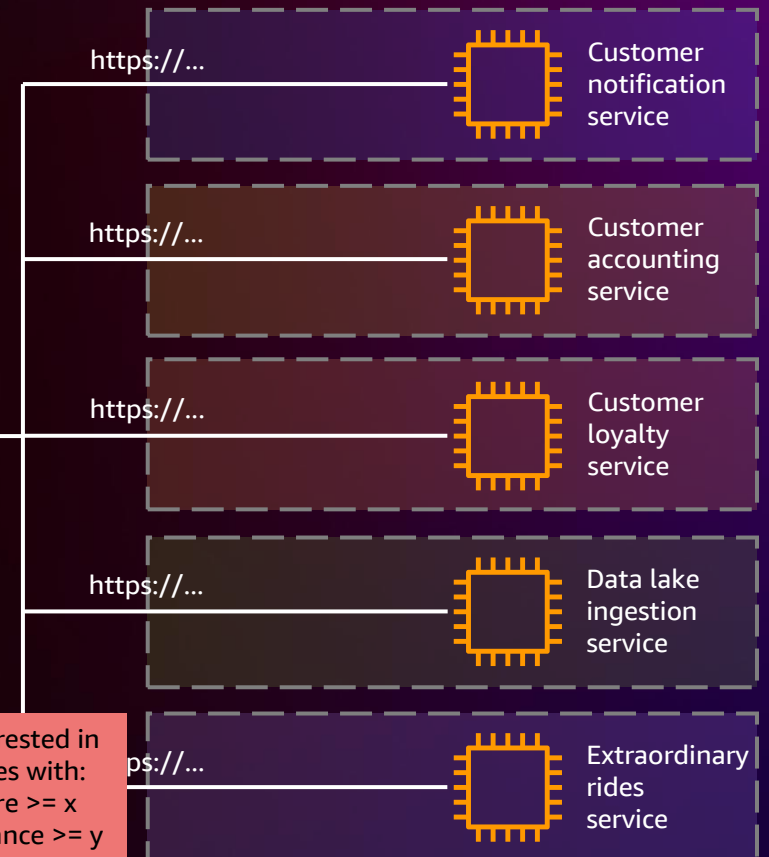
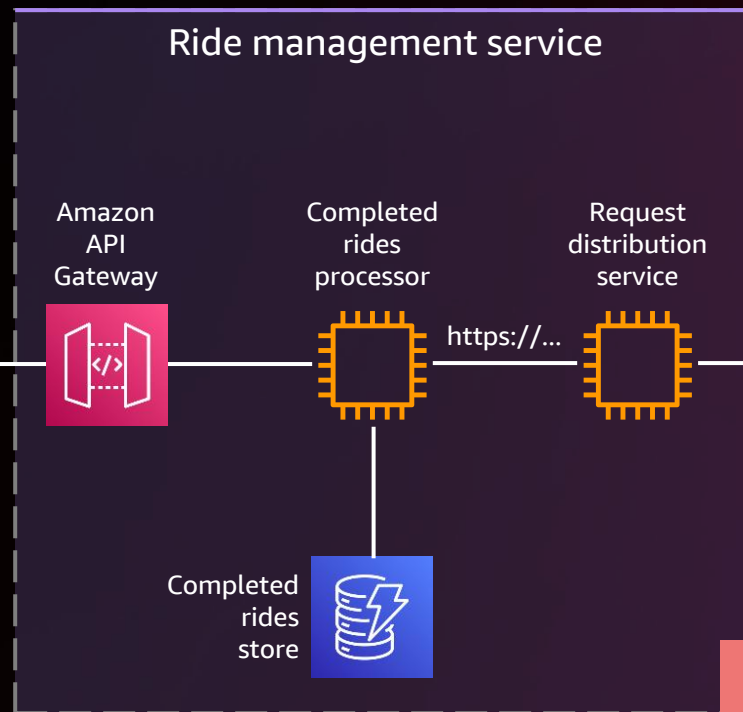
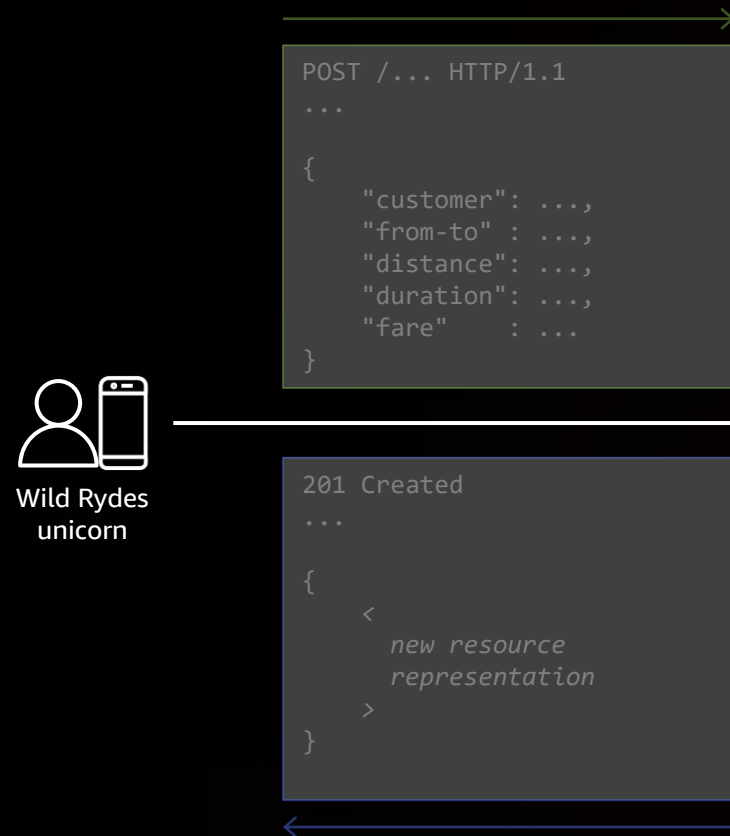
Extraordinary rides service



Submit ride completion

USE CASE

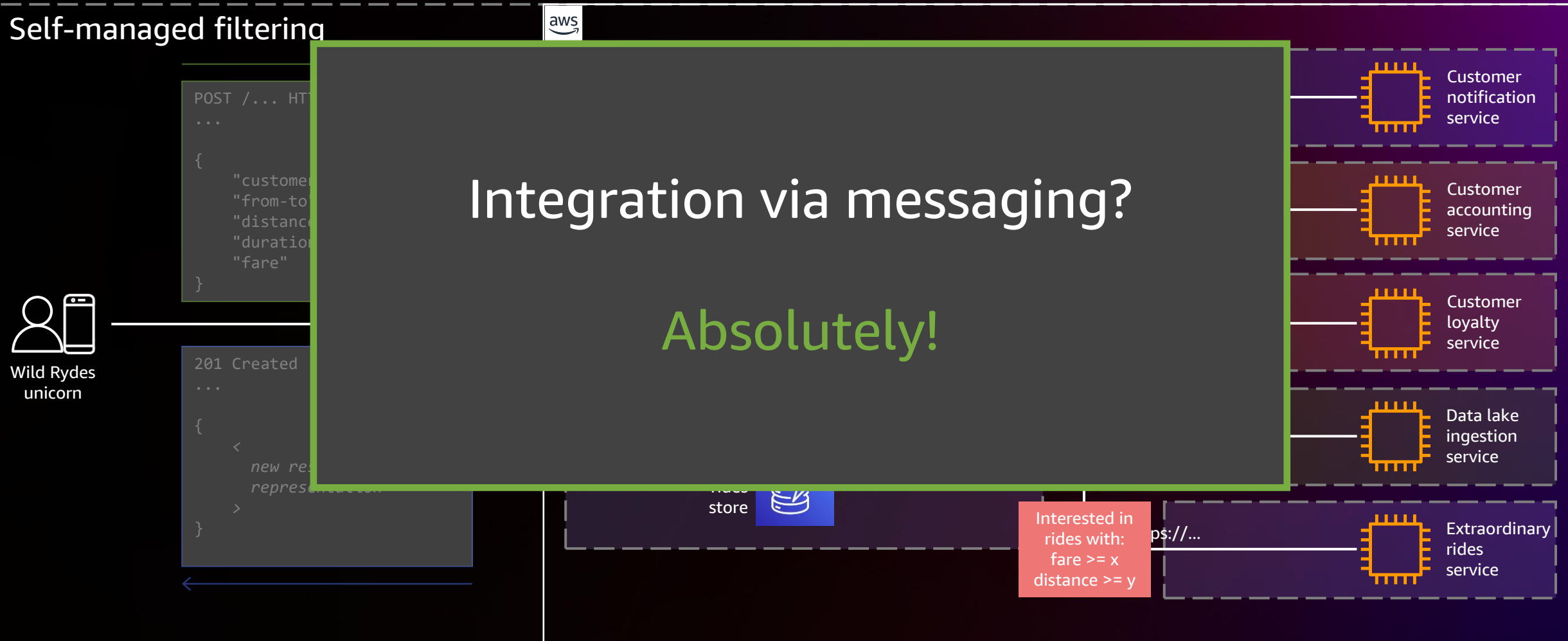
Self-managed filtering



Submit ride completion

USE CASE

Self-managed filtering



Submit ride completion

USE CASE

Publish-subscribe (topic)

```
POST /... HTTP/1.1
...
{
  "customer": ...,
  "from-to": ...,
  "distance": ...,
  "duration": ...,
  "fare" : ...
}
```



Wild Rydes unicorn

```
201 Created
...
{
  <
    new resource
    representation
  >
}
```

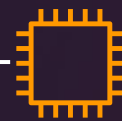


Ride management service

Managed
API
Gateway



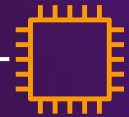
Completed
rides
processor



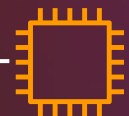
Ride
completion
topic



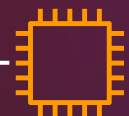
Completed
rides
store



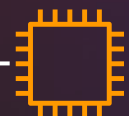
Customer
notification
service



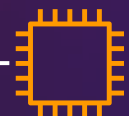
Customer
accounting
service



Customer
loyalty
service



Data lake
ingestion
service



Extraordinary
rides
service



Submit ride completion

USE CASE

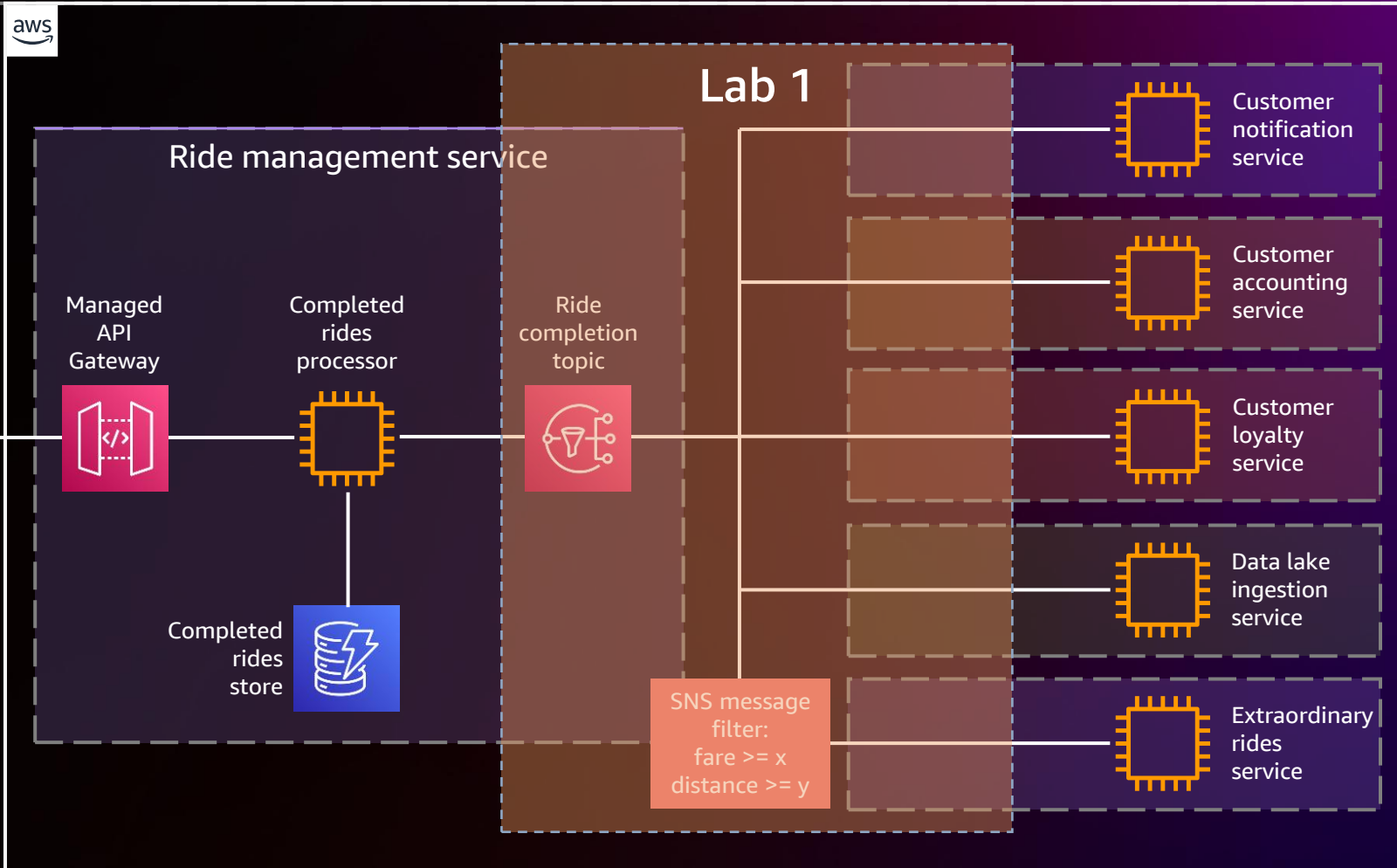
Message filter

```
POST /... HTTP/1.1
...
{
  "customer": ...,
  "from-to": ...,
  "distance": ...,
  "duration": ...,
  "fare" : ...
}
```



Wild Rydes unicorn

```
201 Created
...
{
  <
    new resource
    representation
  >
}
```



Submit ride completion

USE CASE

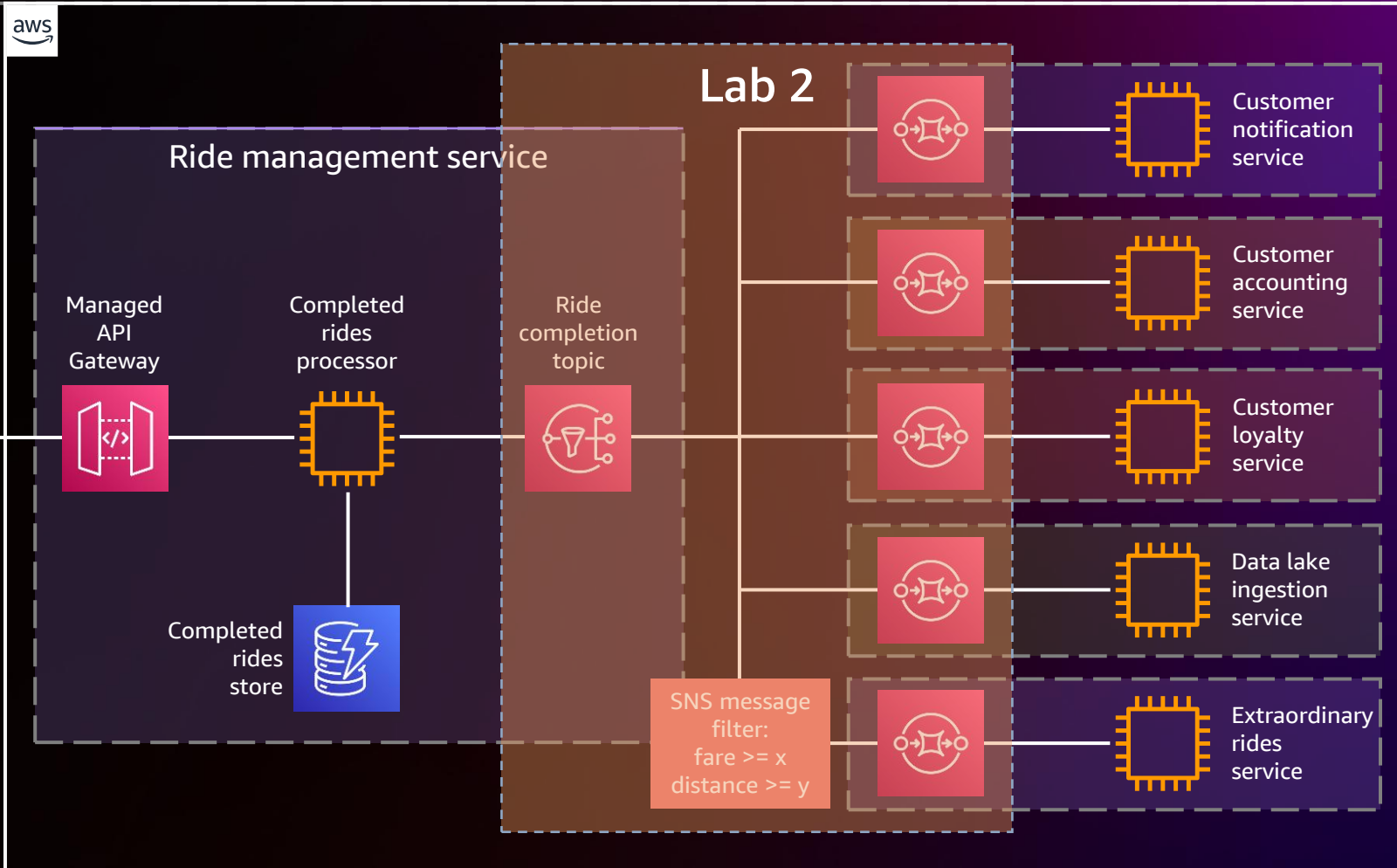
Topic-queue-chaining

```
POST /... HTTP/1.1
...
{
  "customer": ...,
  "from-to": ...,
  "distance": ...,
  "duration": ...,
  "fare" : ...
}
```



Wild Rydes unicorn

```
201 Created
...
{
  <
    new resource
    representation
  >
}
```

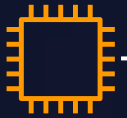


Use case: Instant ride RFQ

Context for lab 3

Message routing

Scatter-gather

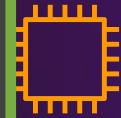


A

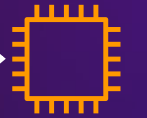
Requester

Remember the

scatter-gather pattern?



Aggregator



Processor

Requester

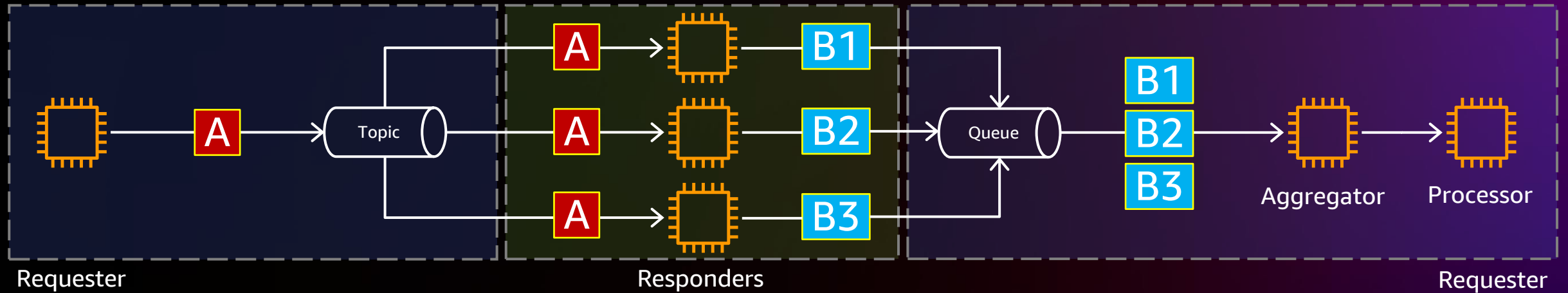
How to distribute

Responses?

For election or parallel processing scenarios, i.e., search for **best** response or **accumulate** responses

Message routing

Scatter-gather



How to distribute a request to relevant/interested parties and capture their individual responses?

For election or parallel processing scenarios, i.e., search for **best** response or **accumulate** responses

Instant ride RFQ

USE CASE

UMR: Unicorn management resource

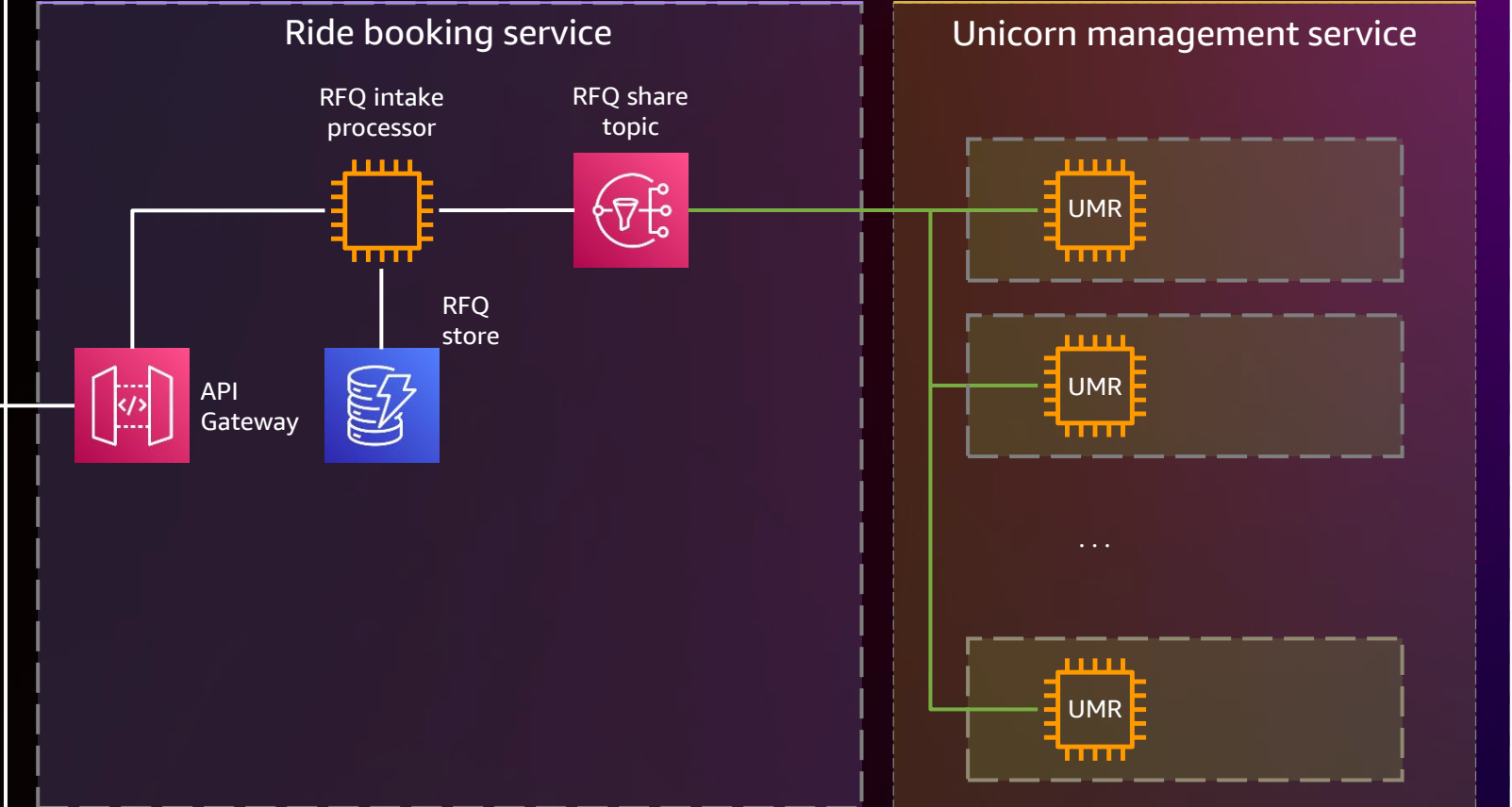
Scatter-gather



Wild Rydes customer

```
POST /<submit> HTTP/1.1
...
{
  "customer": ...,
  "from" : ...,
  "to" : ...
}
```

```
HTTP/1.1 202 Accepted
...
{
  "rfq-id" : ...
}
```



Instant ride RFQ

USE CASE

UMR: Unicorn management resource

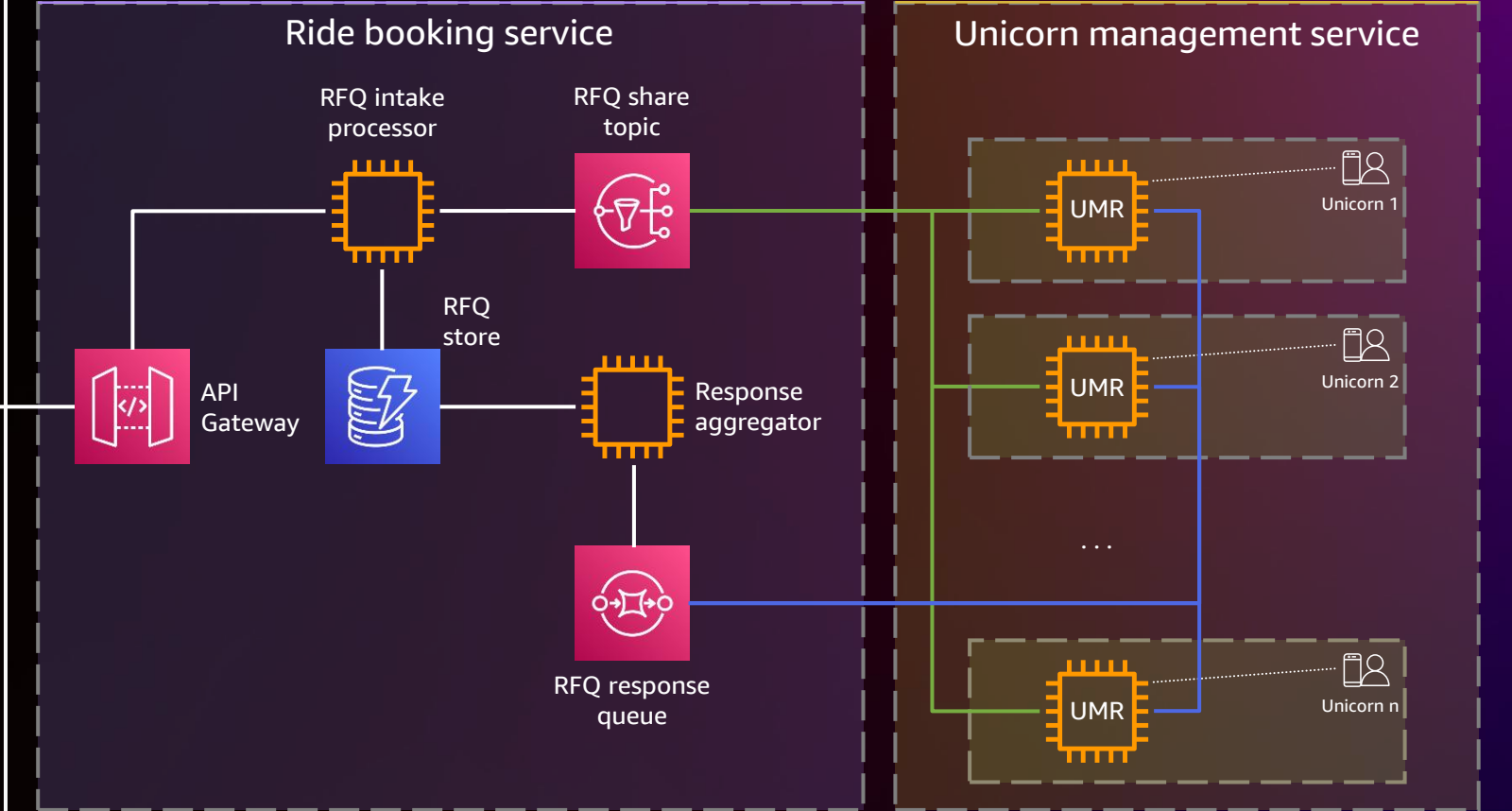
Scatter-gather



Wild Rydes customer

```
POST /<submit> HTTP/1.1
...
{
  "customer": ...,
  "from" : ...,
  "to" : ...
}
```

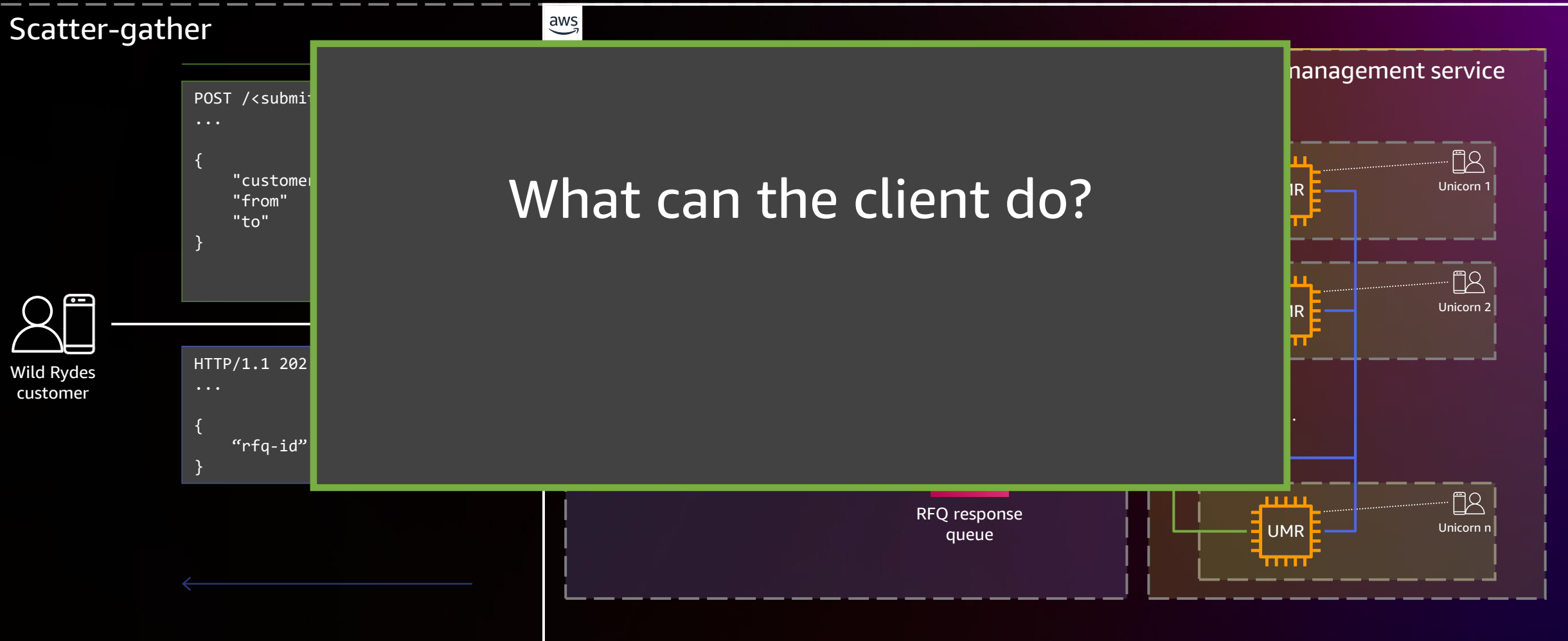
```
HTTP/1.1 202 Accepted
...
{
  "rfq-id" : ...
}
```



Instant ride RFQ

USE CASE

Scatter-gather



Instant ride RFQ

USE CASE

Scatter-gather



```
POST /<submit>
...
{
  "customer": "John",
  "from": "123 Main St",
  "to": "456 Main St"
}
```

```
HTTP/1.1 202 Accepted
...
{
  "links": {
    "self": "/ride/123456"
  },
  "status": "running",
  "eta": 10
}
```

What can the client do?

Retrieve response quotes using response API



management service



RFQ response queue

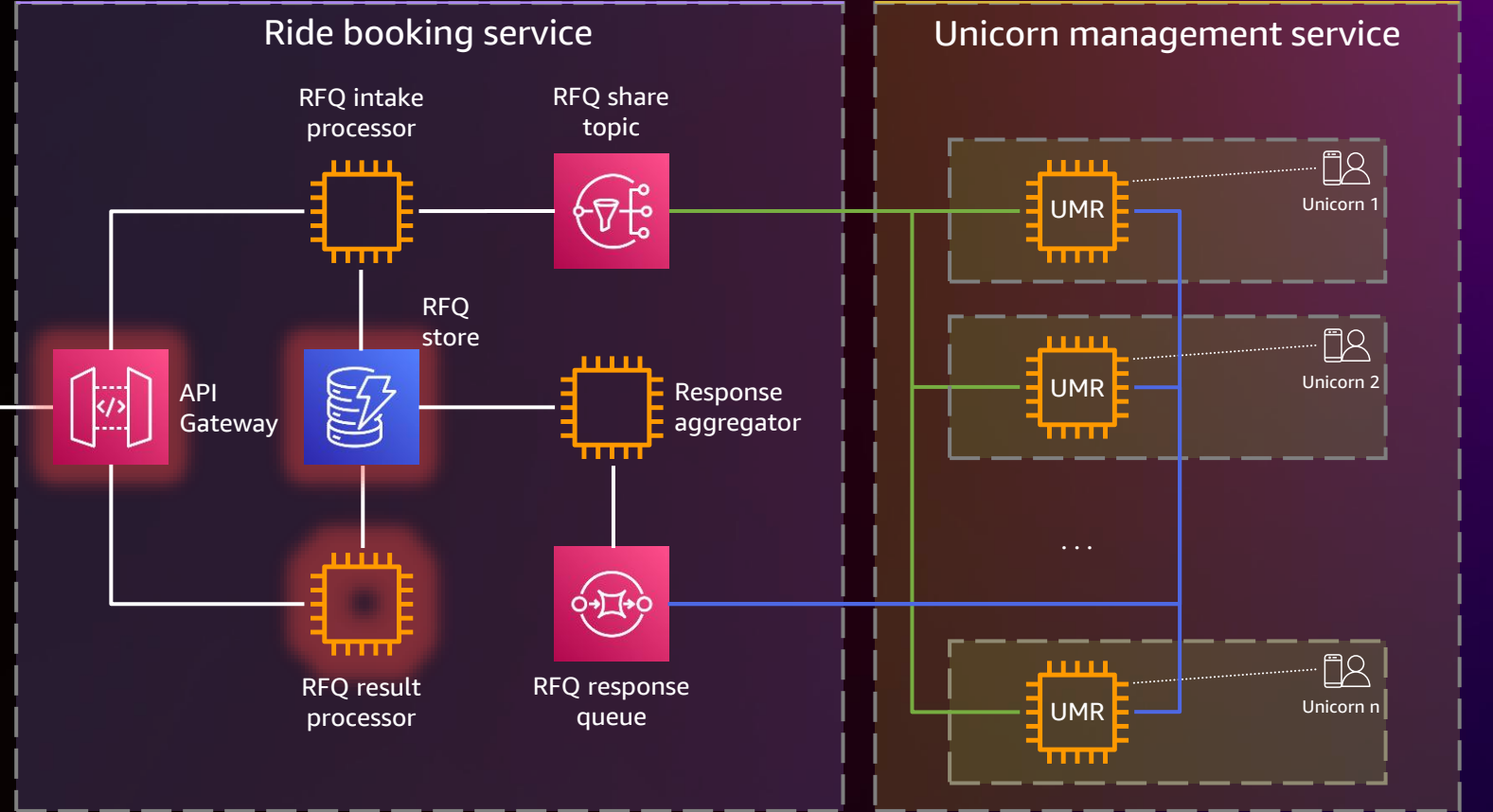


Instant ride RFQ

USE CASE

Scatter-gather

```
GET /resp/<rfq-id> HTTP/1.1  
...
```



Instant ride RFQ

USE CASE

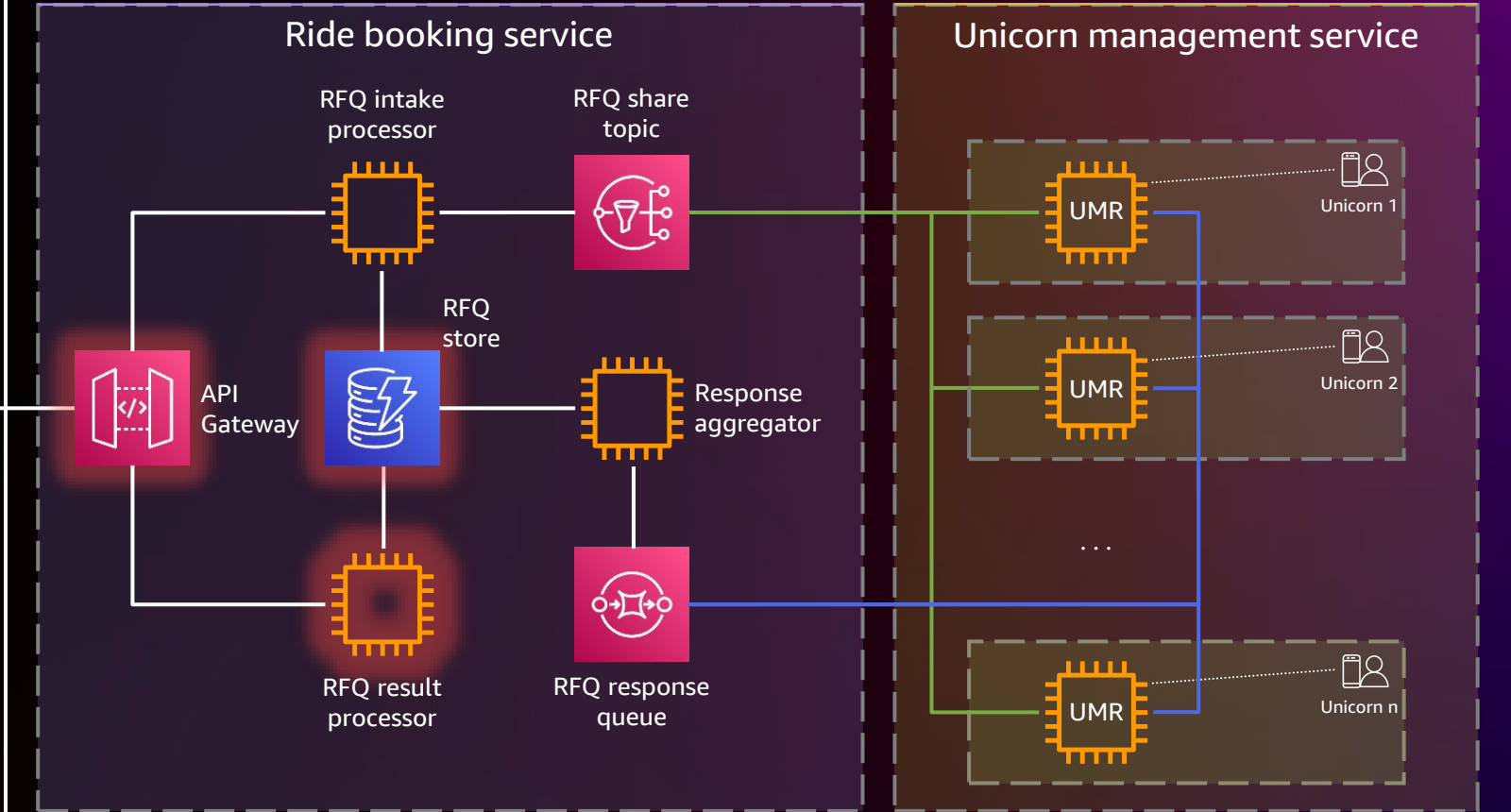
Scatter-gather



Wild Rydes customer

```
GET /resp/<rfq-id> HTTP/1.1  
...
```

```
HTTP/1.1 200 OK  
...  
{  
  "links": {  
    <links>  
  },  
  "ride-data": ...,  
  "quotes" : ...  
}
```

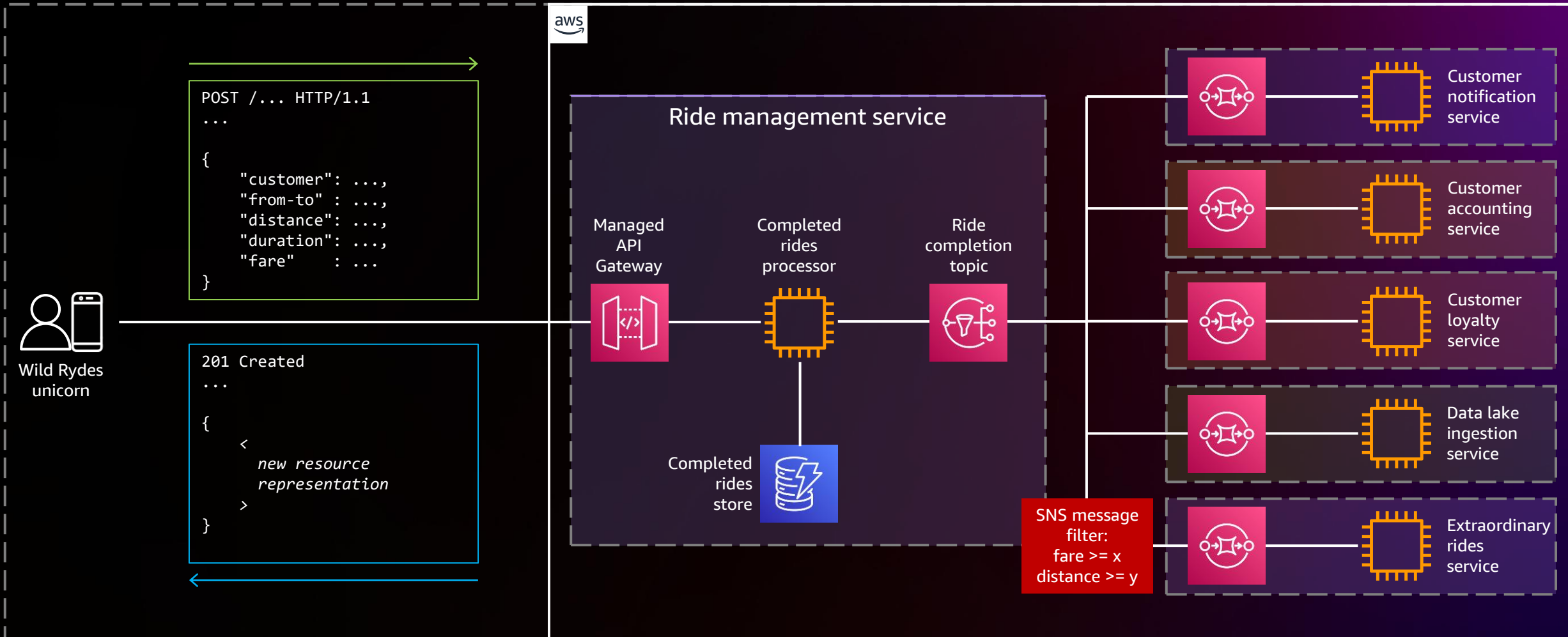


Use case: Fare collection

Context for lab 4

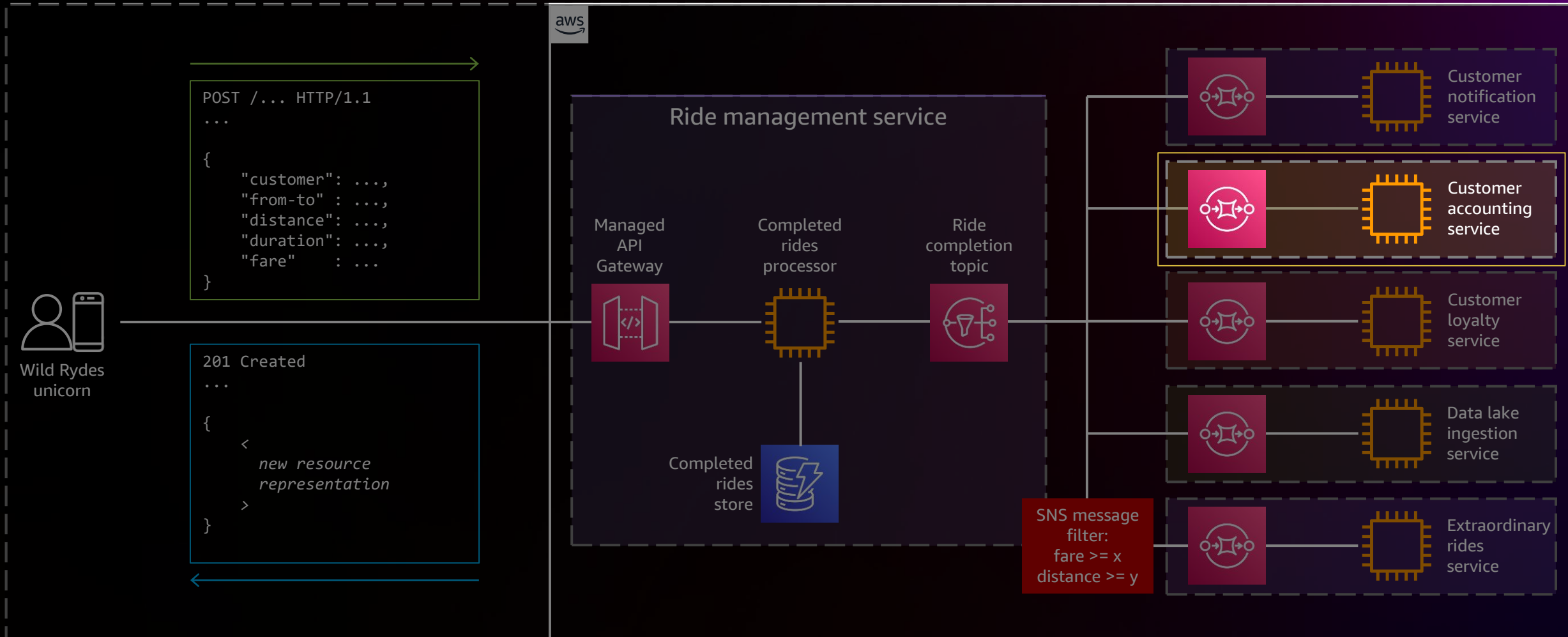
Submit ride completion

USE CASE



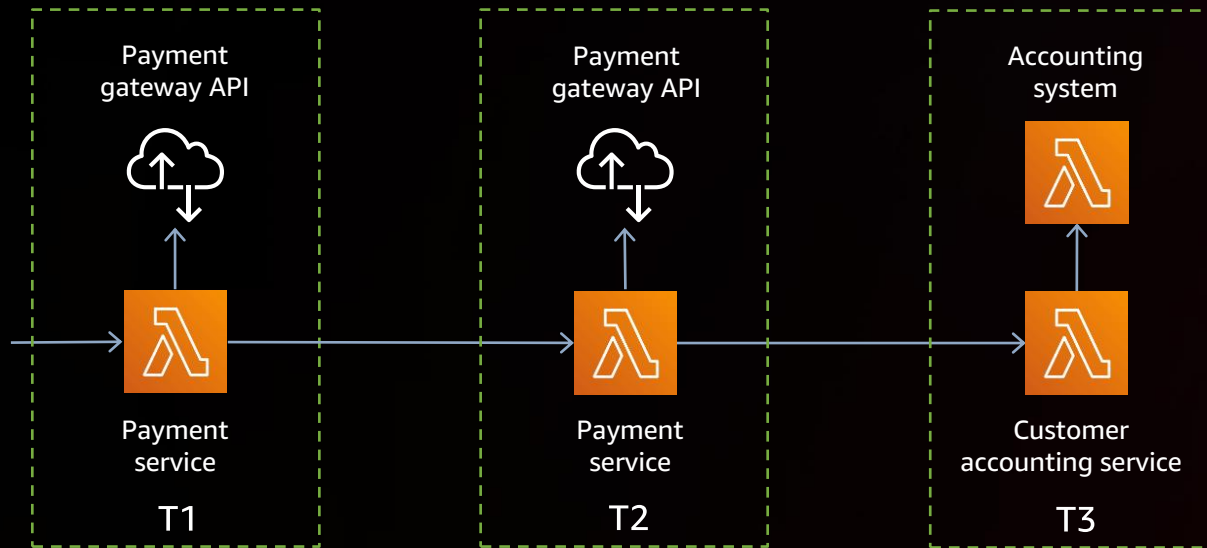
Submit ride completion

USE CASE



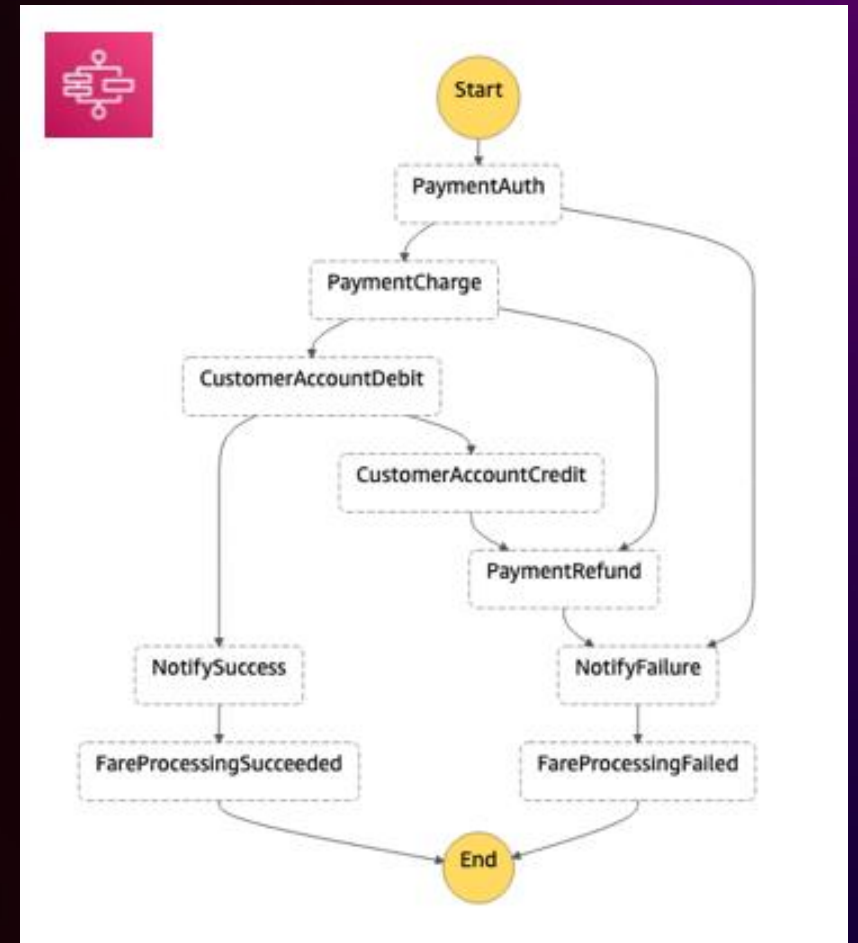
Use case: Fare collection

Saga orchestration



1. Credit card pre-authorization
2. Charge card using pre-authorization code
3. Update customer account

To be treated as one distributed TA, and leave the systems in a semantically consistent state



Getting started with this workshop

- As a participant, you will have access to an AWS account with any optional pre-provisioned infrastructure and IAM policies needed to complete this workshop
- The AWS account will only be available for the duration of this workshop; you will lose access to the account thereafter
- The optional pre-provisioned infrastructure will be deployed to a specific Region; check your workshop content to determine whether other regions will be used
- Be sure to review the terms and conditions of the event; do not upload any personal or confidential information in the account

Step 1: Sign-In via your preferred method

<https://catalog.workshops.aws/join>



aws workshop studio

Workshop Studio > Sign in

Sign in

Choose a preferred sign-in method

Email one-time password (OTP)

Enter your personal or corporate email to receive a one-time password

Login with Amazon

Login with your Amazon.com retail account

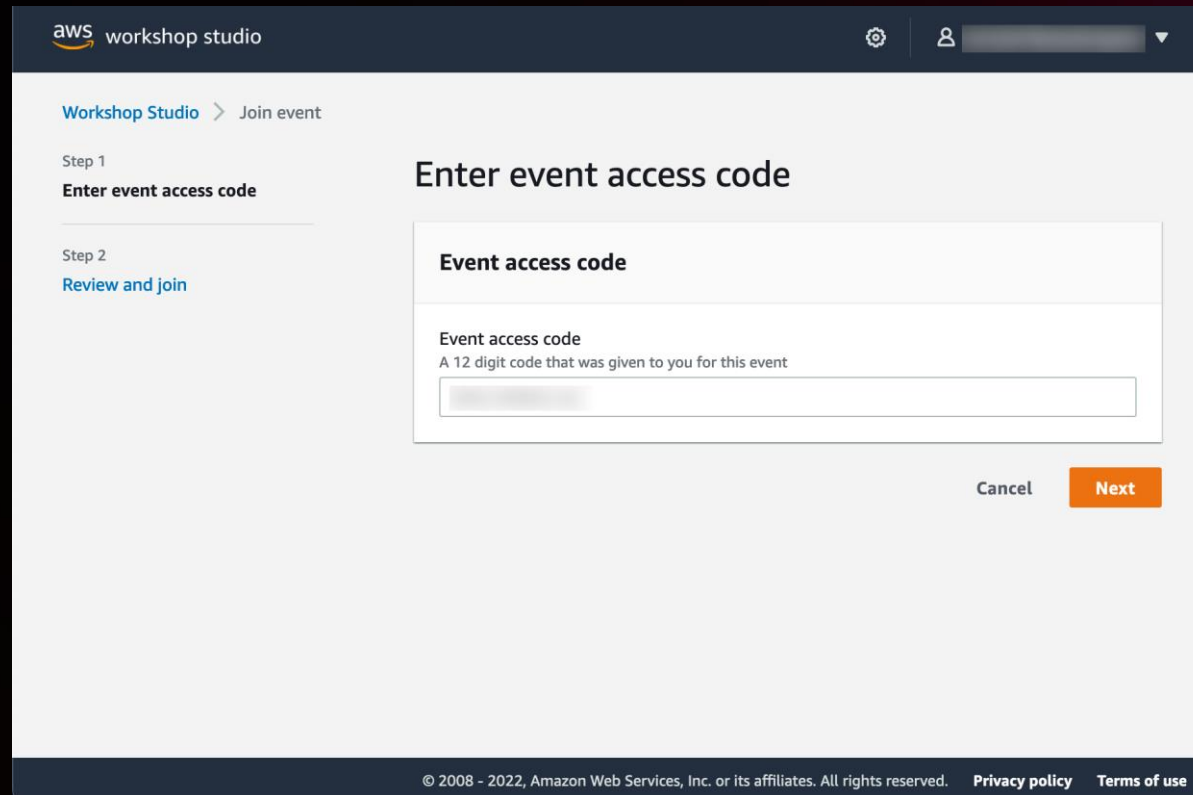
Amazon employee

Login with your Amazon Corporate account. Only for Amazon Employees.

© 2008 - 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy policy](#) [Terms of use](#)

Step 2: Enter event access code

Enter 12-digit event access code: 8c2f-0e0c57-62



The screenshot shows the AWS Workshop Studio interface. At the top, there's a header with the AWS logo and 'workshop studio' text. Below the header, a breadcrumb trail shows 'Workshop Studio > Join event'. The main content area is titled 'Enter event access code'. On the left, a sidebar indicates 'Step 1: Enter event access code' (current step) and 'Step 2: Review and join'. The main form area has a title 'Event access code' and a description: 'Event access code: A 12 digit code that was given to you for this event'. Below this is a text input field. At the bottom right of the form, there are 'Cancel' and 'Next' buttons. The footer contains copyright information: '© 2008 - 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.' and links to 'Privacy policy' and 'Terms of use'.

aws workshop studio

Workshop Studio > Join event

Step 1
Enter event access code

Step 2
Review and join

Enter event access code

Event access code




Event access code
A 12 digit code that was given to you for this event

Cancel Next

© 2008 - 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy policy](#) [Terms of use](#)

Step 3: Review terms and join event

aws workshop studio

Workshop Studio > Join event

Step 1
[Enter event access code](#)

Step 2
Review and join

Review and join

Event details

Name	Start time	Duration	Level
AWS General Immersion Day	9/23/2022 01:13 AM	12 hours	-

Description
AWS General Immersion Day

Terms and Conditions

Read and accept before joining the event

1. By using AWS Workshop Studio for the relevant event, you agree to the AWS Event Terms and Conditions and the AWS Acceptable Use Policy. You acknowledge and agree that are using an AWS-owned account that you can only access for the duration of the relevant event. If you find residual resources or materials in the AWS-owned account, you will make us aware and cease use of the account. AWS reserves the right to terminate the account and delete the contents at any time.
2. You will not: (a) process or run any operation on any data other than test data sets or lab-approved materials by AWS, and (b) copy, import, export or otherwise create derivate works of materials provided by AWS, including but not limited to, data sets.
3. AWS is under no obligation to enable the transmission of your materials through Event Engine and may, in its discretion, edit, block, refuse to post, or remove your materials at any time.
4. Your use of AWS Workshop Studio will comply with these terms and all applicable laws, and your access to AWS Workshop Studio will immediately and automatically terminate if you do not comply with any of these terms or conditions.

☒ I agree with the Terms and Conditions

Cancel

Previous

Join event

© 2008 - 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

[Privacy policy](#)

[Terms of use](#)

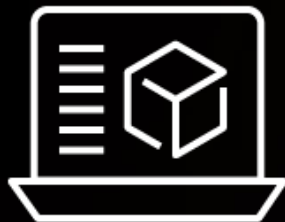


Resources and call to action



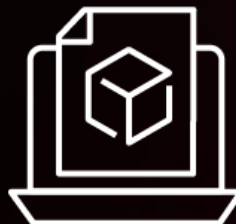
Continue your AWS Serverless learning

Learn at your
own pace



Expand your Serverless
skills with our learning plan
on **AWS Skill Builder**

Increase your
knowledge



Use our **Ramp-Up Guides**
to build your Serverless
knowledge

Earn AWS
Serverless badge



Demonstrate your
knowledge by achieving
digital badges



<https://s12d.com/serverless-learning>

Resources and call to action

AWS blog series

Read again about the use cases and patterns **from this talk** (and more):

AWS architecture blog

Various blog posts in the **application integration** category:



Reach out to your friendly
AWS solutions architect

and keep in mind

Loose coupling is **always**
better than **lousy** coupling



Thank you!

Mithun Mallick

LinkedIn:



Dirk Fröhner

Twitter: @dirk_f5r

LinkedIn:



Please complete the session survey in the **mobile app**



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.