

goto;

GOTO AARHUS 2022

#GOTOaar





serverlesspresso

Who am I?

- Principal Developer Advocate – AWS
- Father of 5
- Brand new grandpa
- Drummer
- Foodie
- Twitter -> @edjgeek



Intro



serverlesspresso

An event driven coffee ordering app
built with serverless architecture

The frontend applications

VUE.JS APPS HOSTED WITH AWS AMPLIFY CONSOLE

1. Ordering app:

- Loads menu
- Scans barcode to get order token
- Sends order
- Displays updates

2. TV display app:

- Shows current barcode
- Receives order status updates
- Listens for store open/close events

3. Barista app:

- Shows incoming orders
- Enables order completion or cancellation
- Enables store open/close events



serverlesspresso

What is Serverlesspresso?

Serverlesspresso is a coffee ordering app built with serverless architecture:

1. Place an order on your mobile device.
2. The order appears on the monitor and the barista's tablet app.
3. You get a notification when the drink is ready.



serverlesspresso

Try it out! Go to:
<https://s12d.com/coffee>

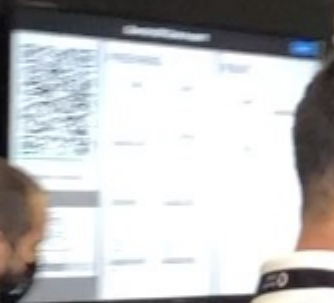




serverlesspresso
COFFEE BAR

Managing each coffee journey

- Multi-tenant architecture
- Distributed system
- Scalable architecture
- Distributed system
- Scalable architecture
- Distributed system



SERVERLESSLIVE.COM

driva

The Data Resiliency Co.

Where ?



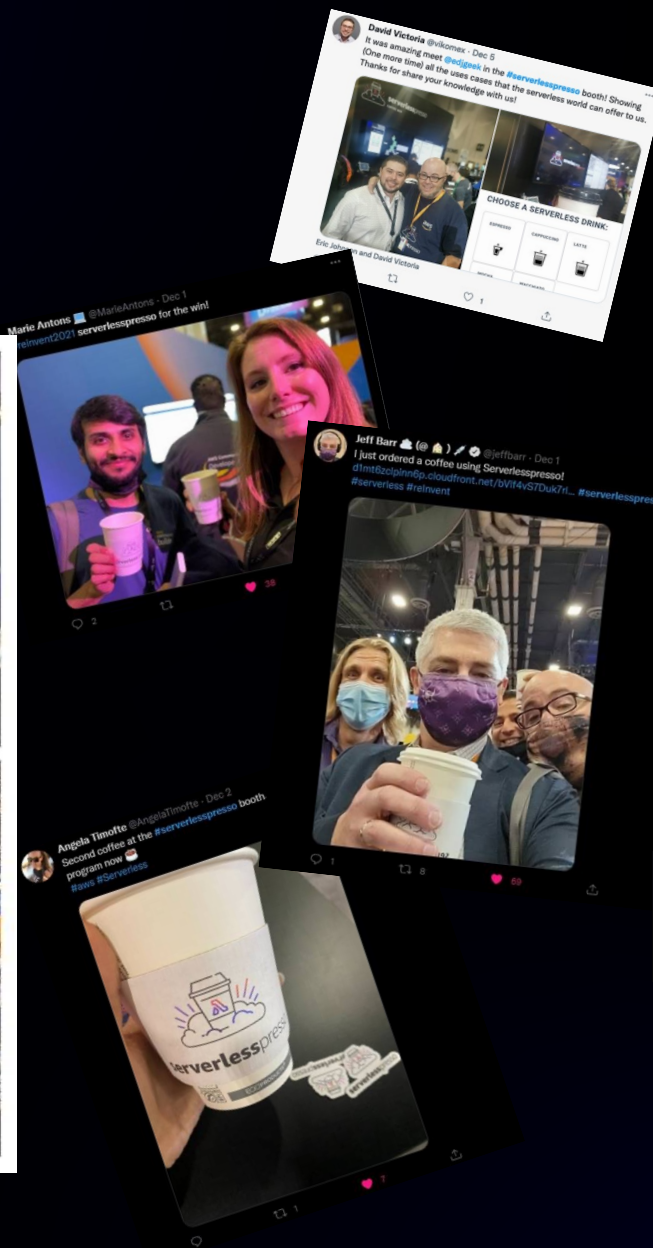
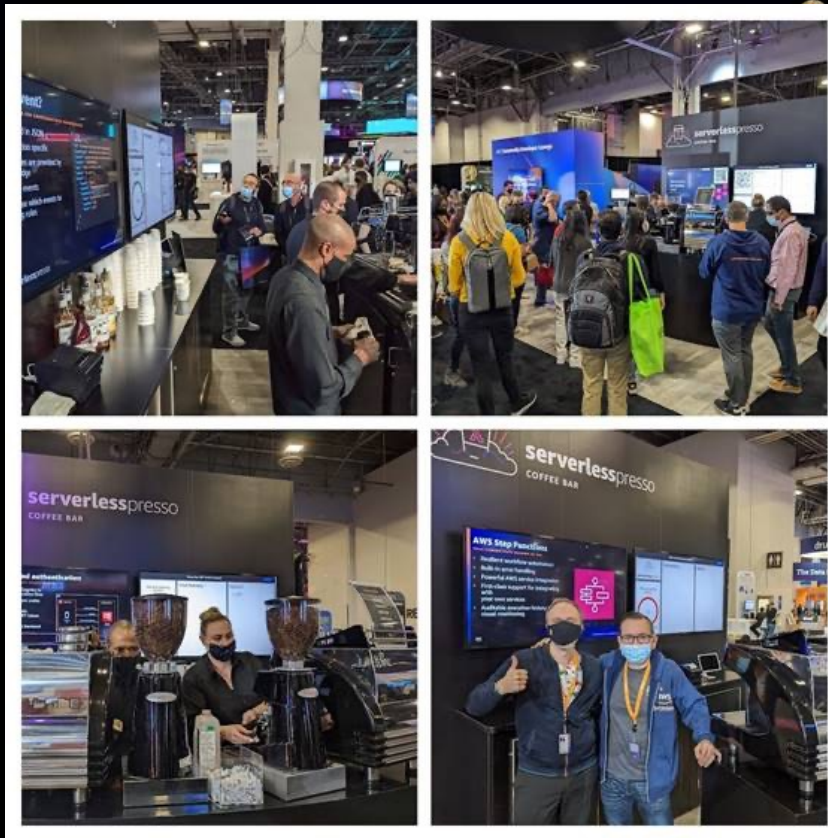
serverlesspresso

AWS re:Invent

1,920 orders

71 drinks per hour

3m 34s further reading



LEAP Saudi Arabia

1,310 orders

63 drinks per hour

2m further reading

AWS Summit Tel Aviv

1,378 coffees!

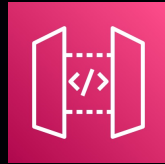
Current record holder, beating
The Paris summit by over 400!



AWS Serverless services used



AWS Amplify
Console



Amazon
API Gateway



Amazon
DynamoDB



Amazon
EventBridge



AWS Step
Functions



AWS
IoT Core



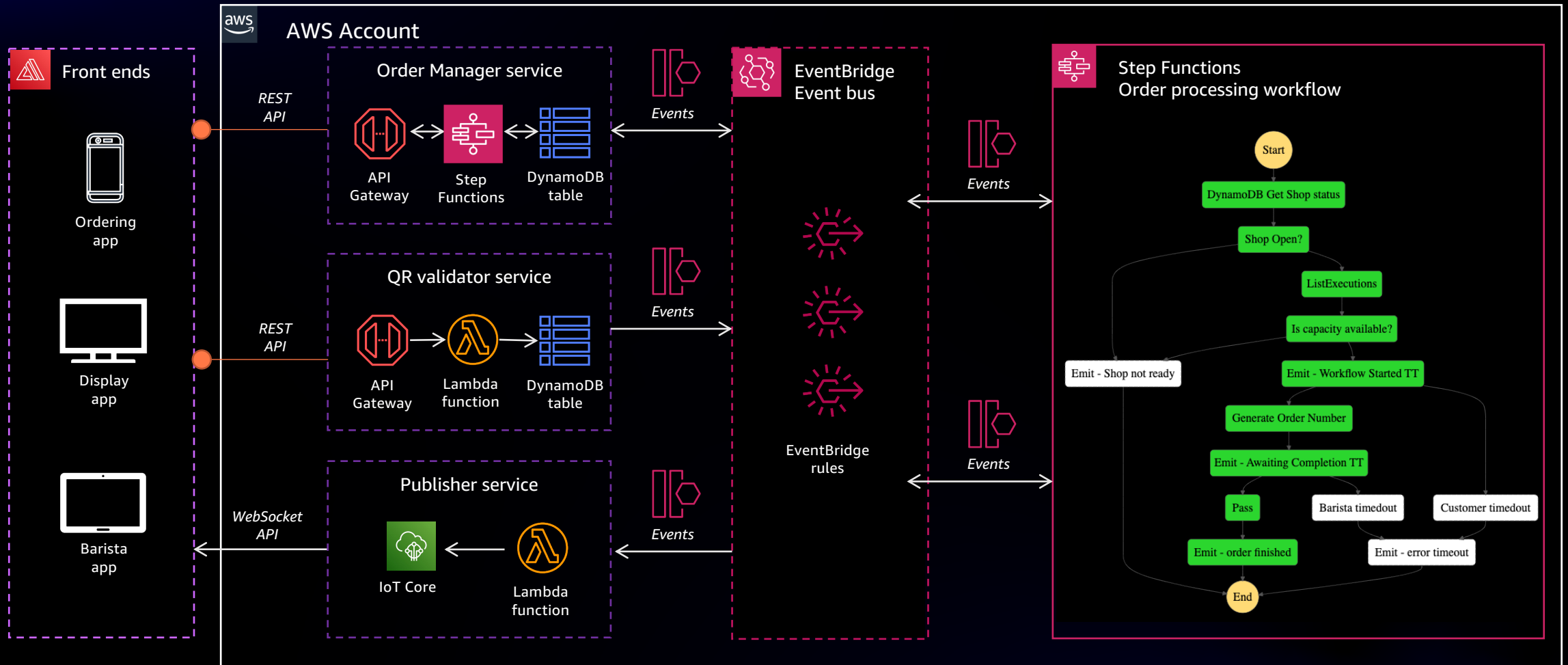
AWS
Lambda



serverlesspresso

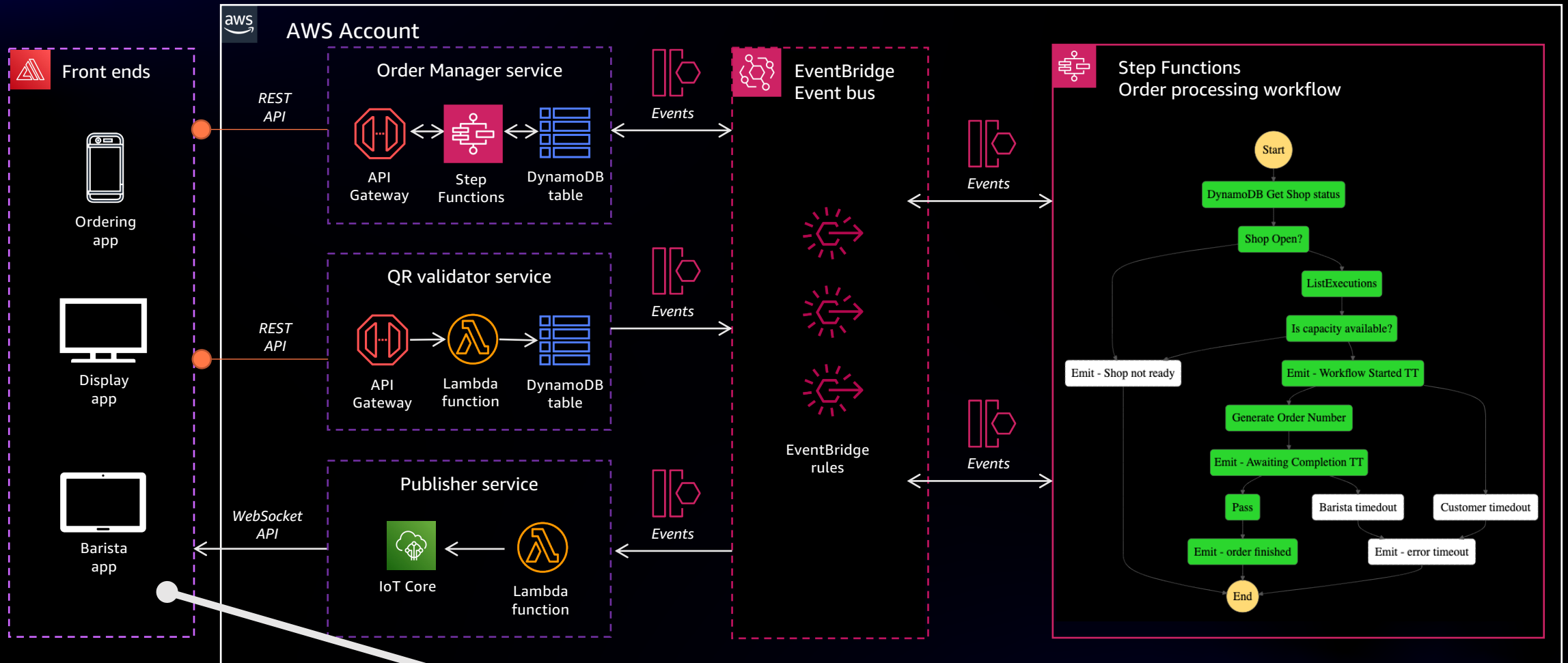
High level architecture

How it works



serverlesspresso

How it works

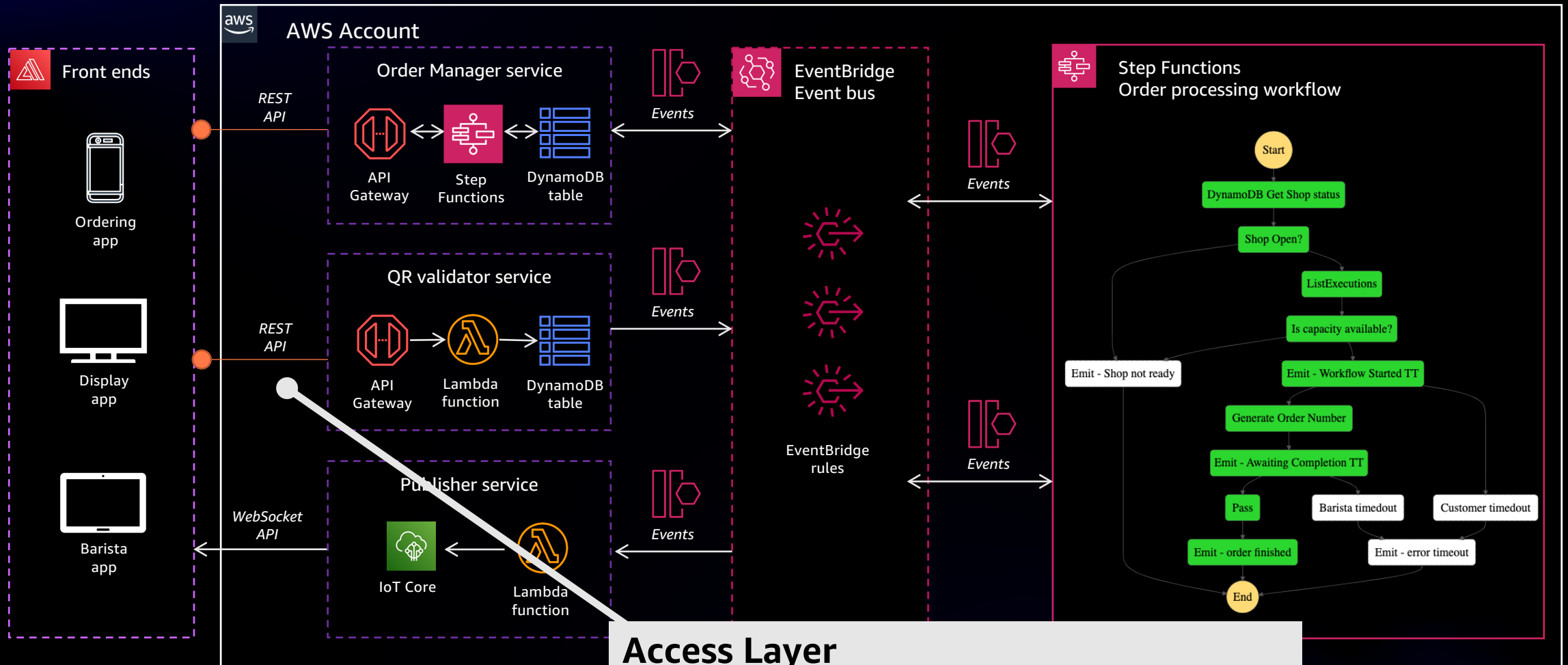


serverlesspresso

Front Ends

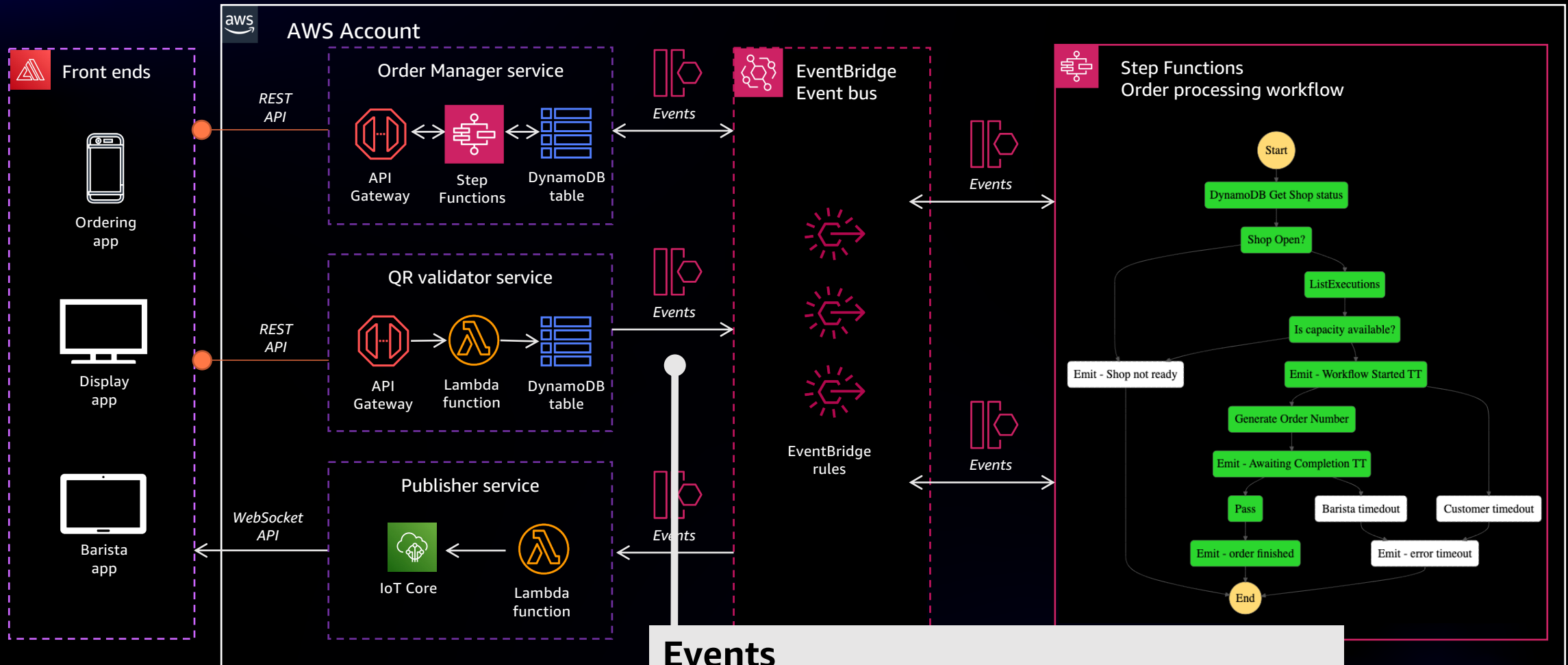
3 web apps hosted on Amazon S3

How it works



serverlesspresso

How it works



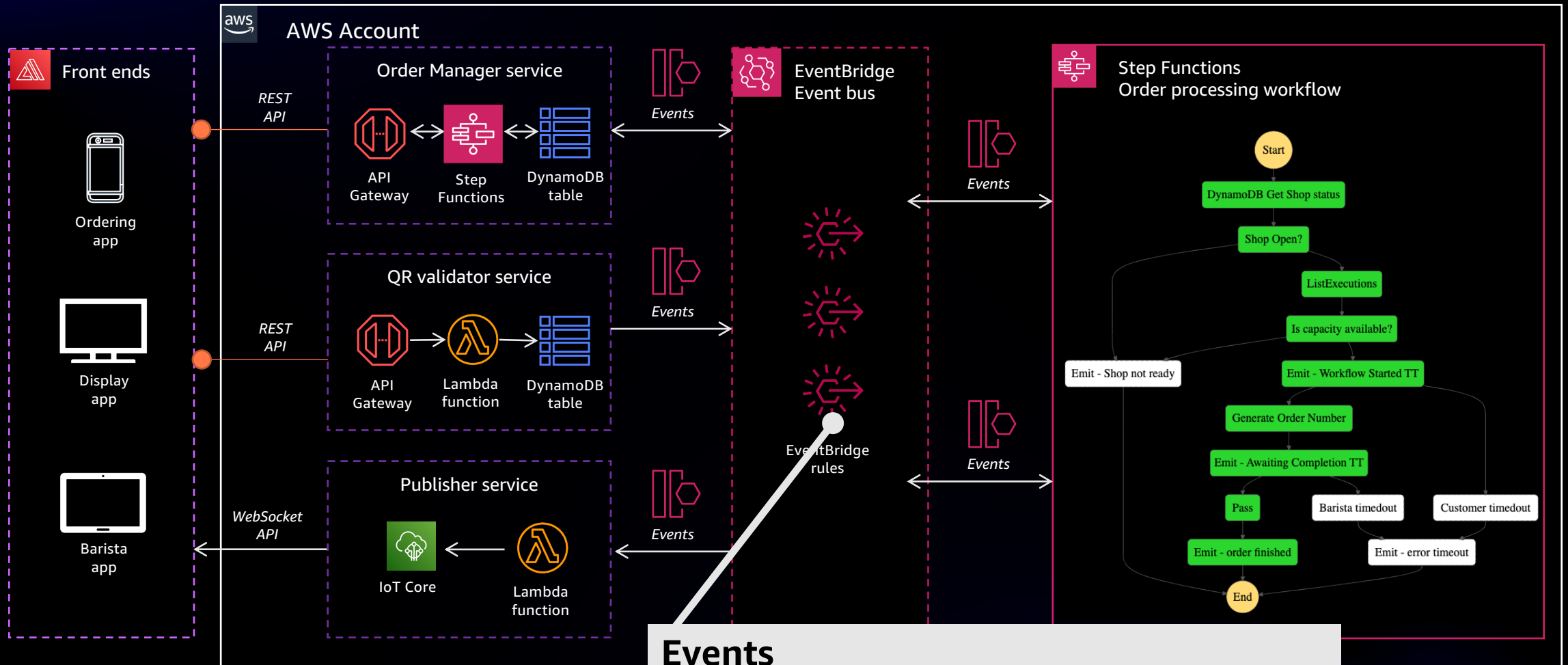
Events

Serverless event bus delivers multiple messages about the state of each order



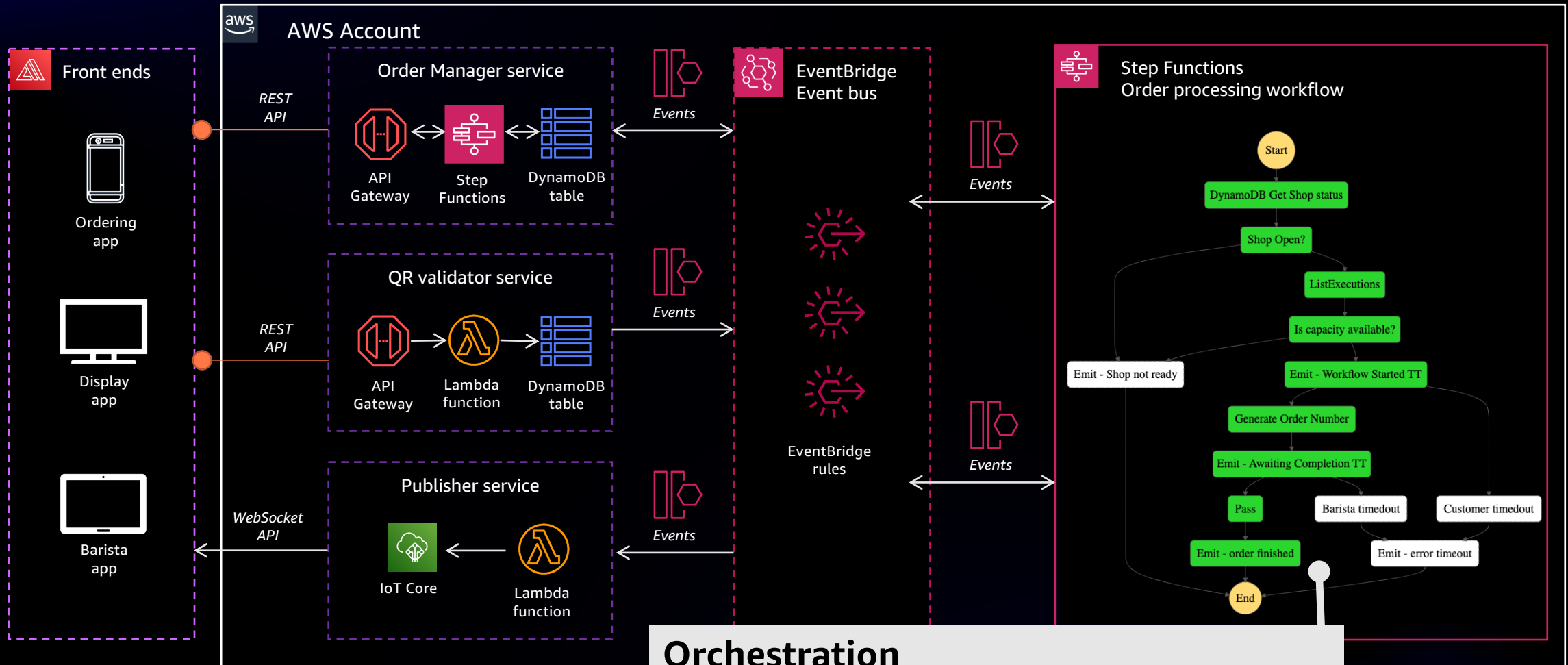
serverlesspresso

How it works



serverlesspresso

How it works



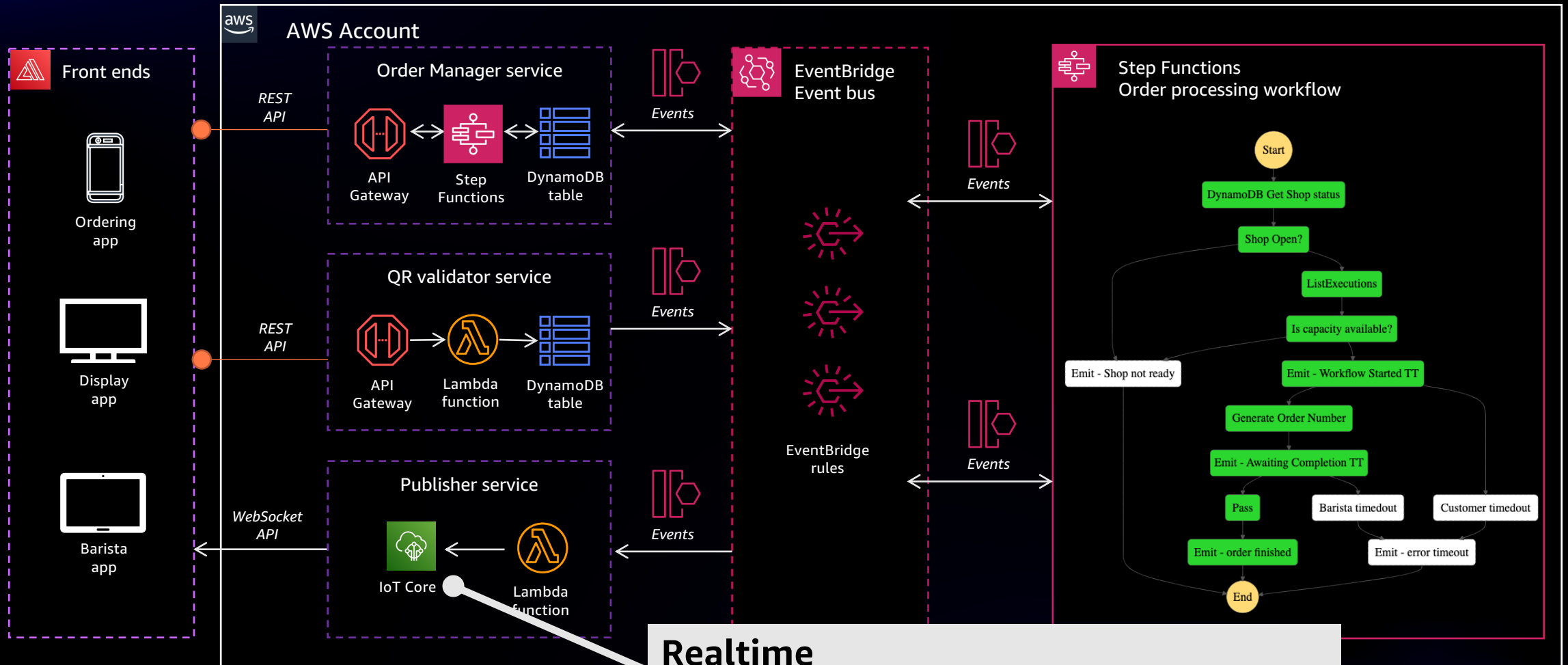
Orchestration

AWS Step Functions orchestrates each order from start to completion



serverlesspresso

How it works



Realtime

IoT topic updates web front ends



serverlesspresso

QR validator service

Throttles the queue to 10 drinks every 5 minutes
and prevents unauthorized orders

QR code generation

- A new QR code every 5 minutes.
- Valid for 10 scans.
- Hidden once all scans are “used”.

CODE: 41g_-KJHGT



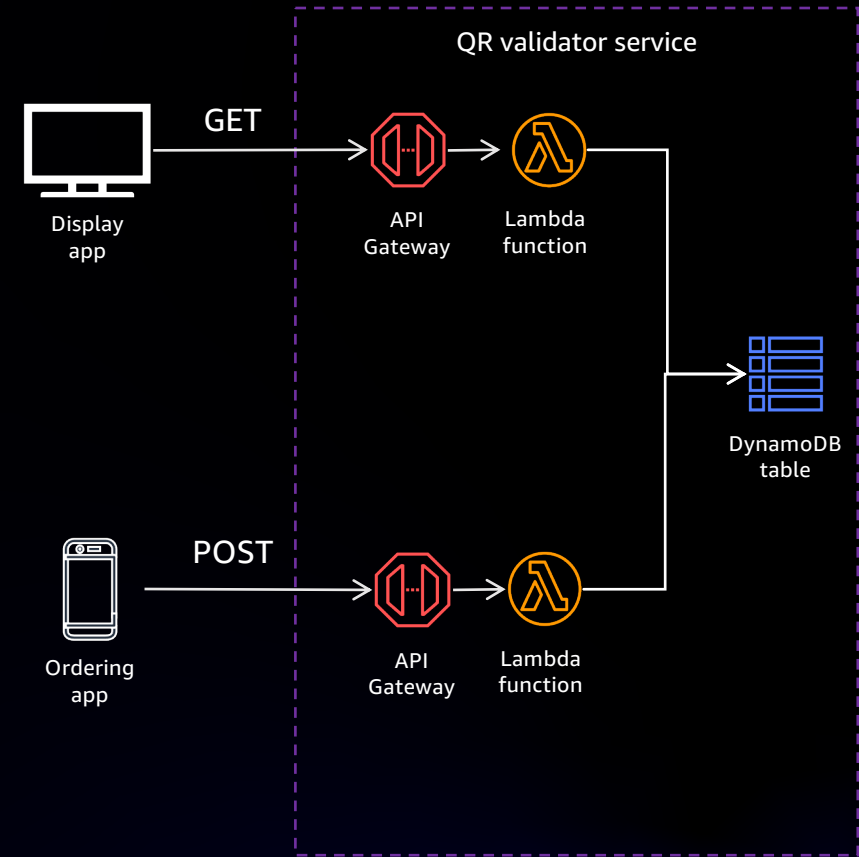
QR code generation

Generate

- The display app makes a request to generate the QR code and store the ID in DynamoDB.

Validate

- The Ordering app, makes a POST request to validate the QR code and ID



Token bucket system

When a valid QR is scanned, it decrements the *Available tokens* number and emits an event that triggers the order processor workflow.

PK	Available tokens	End_ts	Last_code	Last_id	Start_ts
1234423	10	1645200599999	41g_-KJHGT	1234423	1645200300000
1234123	8	1645628399999	6KJH_-FJ5Lh	1234123	1645628100000
5412322	1	1645449599999	91HHFFJHF	5412322	1645449300000
3435657	2	1645435199999	OCZomT756	3435657	1645434900000

DynamoDB validator-table

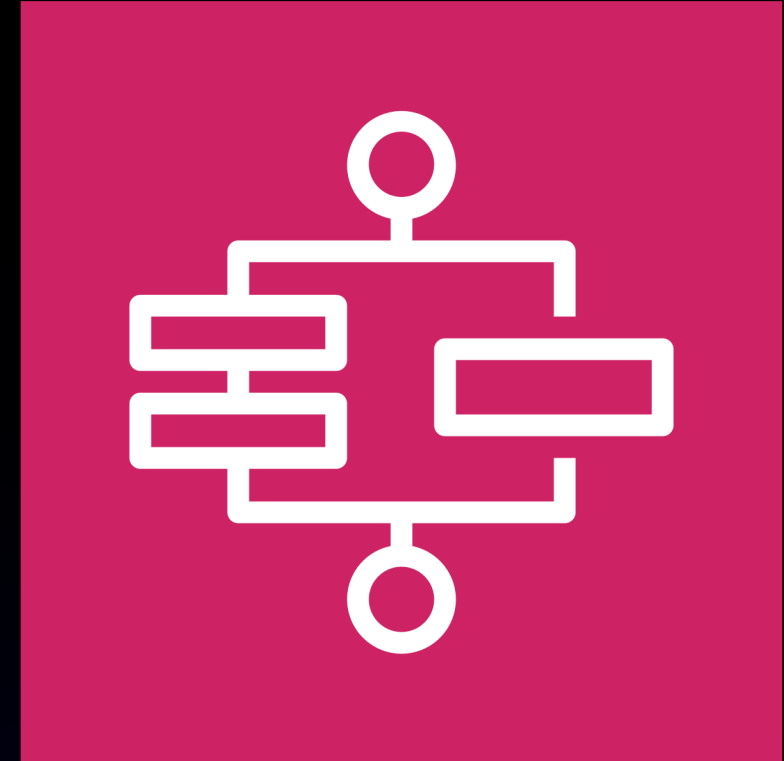
Order Processor service

Orchestrates each order from start to completion

AWS Step Functions

FULLY MANAGED STATE MACHINES ON AWS

- Resilient workflow automation
- Built-in error handling
- Powerful AWS service integration
- First-class support for integrating with your own services
- Auditable execution history and visual monitoring



Define the workflow...

DEFINE ALL THE STEPS IN MAKING A DRINK

Process for making a drink

- 1) Check the store is open
- 2) Get barista capacity
- 3) Wait for the customer order - cancel if > 5 mins
- 4) Generate an order number
- 5) Wait for barista to make drink - cancel if > 15 mins
- 6) Also handle cancelation by customer or barista



... then design visually with Workflow Studio

REPLACE SPAGHETTI CODE WITH STATE MACHINES

Step Functions Workflow Studio [Info](#)

Search

Actions

Flow

MOST POPULAR

 AWS Lambda Invoke

 Amazon SNS Publish

 Amazon ECS RunTask

 AWS Step Functions StartExecution

 AWS Glue StartJobRun

COMPUTE

 Amazon Data Lifecycle ...

 Amazon EBS

 Amazon EC2

 AWS EC2 Instance Conn...

 Elastic Inference

Undo

Redo

Zoom in

Zoom out

Center

Start

Lambda: Invoke
Get Shop Status

Choice state
Shop Open?

Rule #1

Default

Lambda: Invoke
Get Capacity Status

Choice state
Capacity Available?

Rule #1

Default

EventBridge: PutEvents
Emit- Shop not ready

EventBridge: PutEvents
Emit - Workflow Started TT

Lambda: Invoke
Generate Order Number

Cancel

Apply and exit

Export

Form

Definition

Shop Open?

Configuration

Input

Output

State name

Shop Open?

State type

Choice

Choice Rules

Choice rules let you create if-then logic to determine which state the workflow should transition to next.

Rule #1

not(\$.StoreOpen.modified.storeOpen == true)

Default rule

Defines default state when no rule evaluates to true

+ Add new choice rule

Comment - optional

check if Capacity is available

```
graph TD; Start((Start)) --> GetShopStatus[Lambda: Invoke  
Get Shop Status]; GetShopStatus --> ShopOpen{Choice state  
Shop Open?}; ShopOpen -- Rule #1 --> EmitNotReady[EventBridge: PutEvents  
Emit- Shop not ready]; ShopOpen -- Default --> GetCapacityStatus[Lambda: Invoke  
Get Capacity Status]; GetCapacityStatus --> CapacityAvailable{Choice state  
Capacity Available?}; CapacityAvailable -- Rule #1 --> EmitNotReady; CapacityAvailable -- Default --> EmitWorkflowStarted[EventBridge: PutEvents  
Emit - Workflow Started TT]; EmitWorkflowStarted --> GenerateOrderNumber[Lambda: Invoke  
Generate Order Number];
```

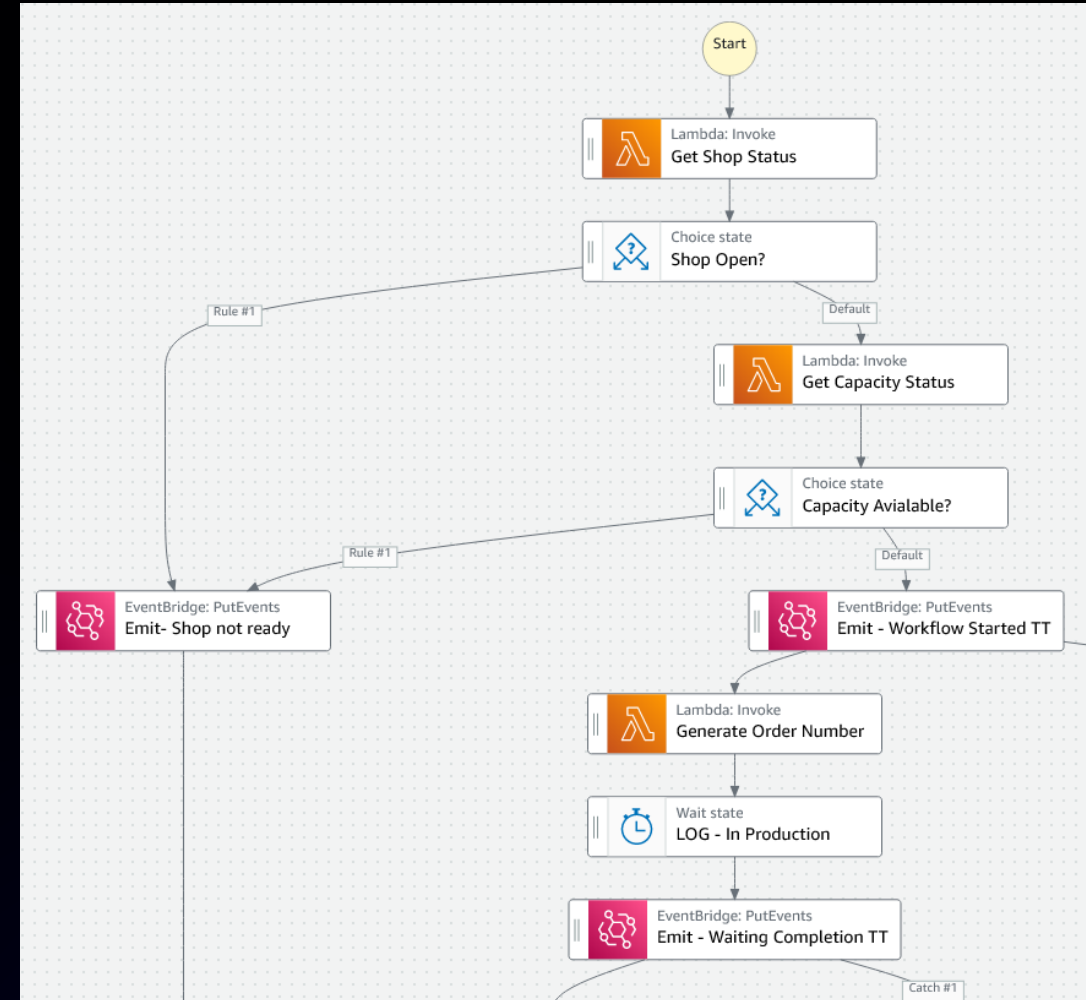
Managing each coffee's journey

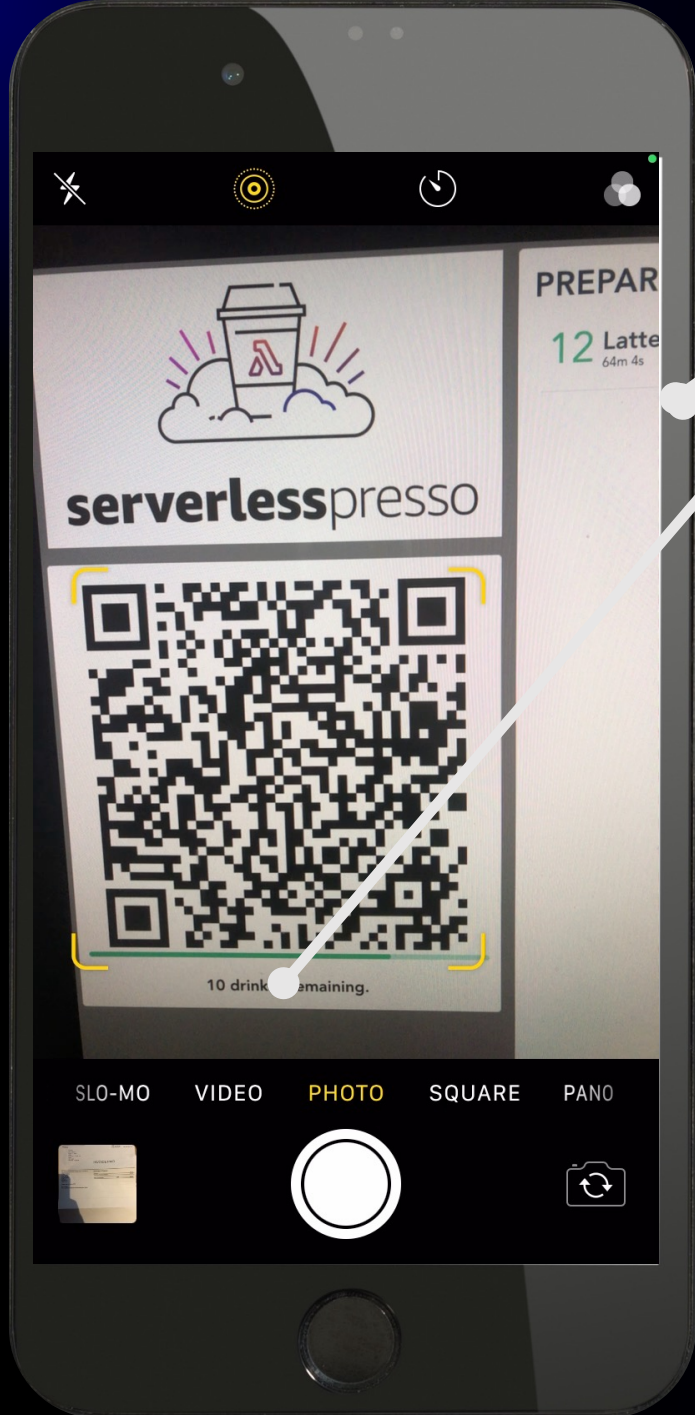
USING AWS STEP FUNCTIONS TO MANAGE EACH WORKFLOW EXECUTION

- Workflow ensures store is open and baristas have capacity
- Allows customer 5 minutes to order before timing out / cancelling
- Allows barista 15 minutes to make before timing out / cancelling
- Uses Lambda functions for custom logic; uses direct service integrations otherwise



serverlesspresso





Collect drink, learn about the order journey, and share.
for a one-time login code.



Check shop is ready, wait for customer to submit order.

Resumes workflow which generates new order number. Wait for barista to complete order.

Barista makes drink. Workflow resumes and emits order completion event.

Order Manager service

Handles order persistence to DynamoDB

Order table

Each order is persisted to a DynamoDB table.
The entry is updated at various stages of the order lifecycle.

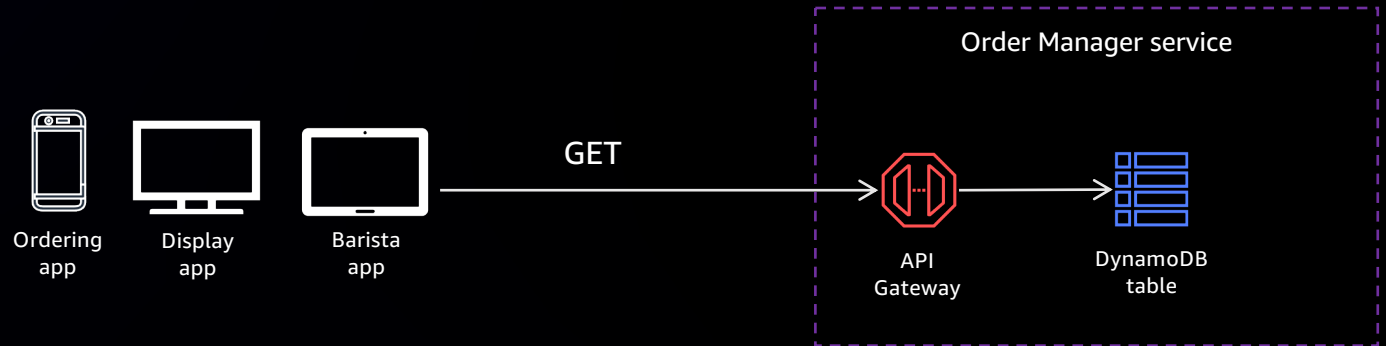
PK	SK	TS	UID	OrderNo	TaskToken	OrderState	DrinkOrder
Orders	2	1645726 346199	1	10	AAAAKgAAAAIAAAAAAAAAA Alp4um0gPw/FO3rqpCDvLE AL+l/h+	COMPLETED	{"userId":"1","drink":"Cappuccino","modifiers":[] ,"icon":"barista-icons_cappuccino-alternative"}

API Gateway to DynamoDB

/myOrders - GET

/orders - GET

/orders/{id} - GET



Front end applications query *Order* table directly from API Gateway.

Velocity mapping templates modify the incoming request.

```
#set($subFromJWT = $context.authorizer.claims.sub)

{
  "TableName": "serverlesspresso-order-table",
  "IndexName": "GSI-userId",
  "KeyConditionExpression": "#USERID = :USERID",
  "ExpressionAttributeNames": {
    "#USERID": "USERID"
  },
  "ExpressionAttributeValues": {
    ":USERID": {
      "S": "$subFromJWT"
    }
  },
  "ScanIndexForward": true,
  "ProjectionExpression": "PK, SK, orderNumber, robot, drinkOrder, ORDERSTATE, TS"
}
```

CRUD operations

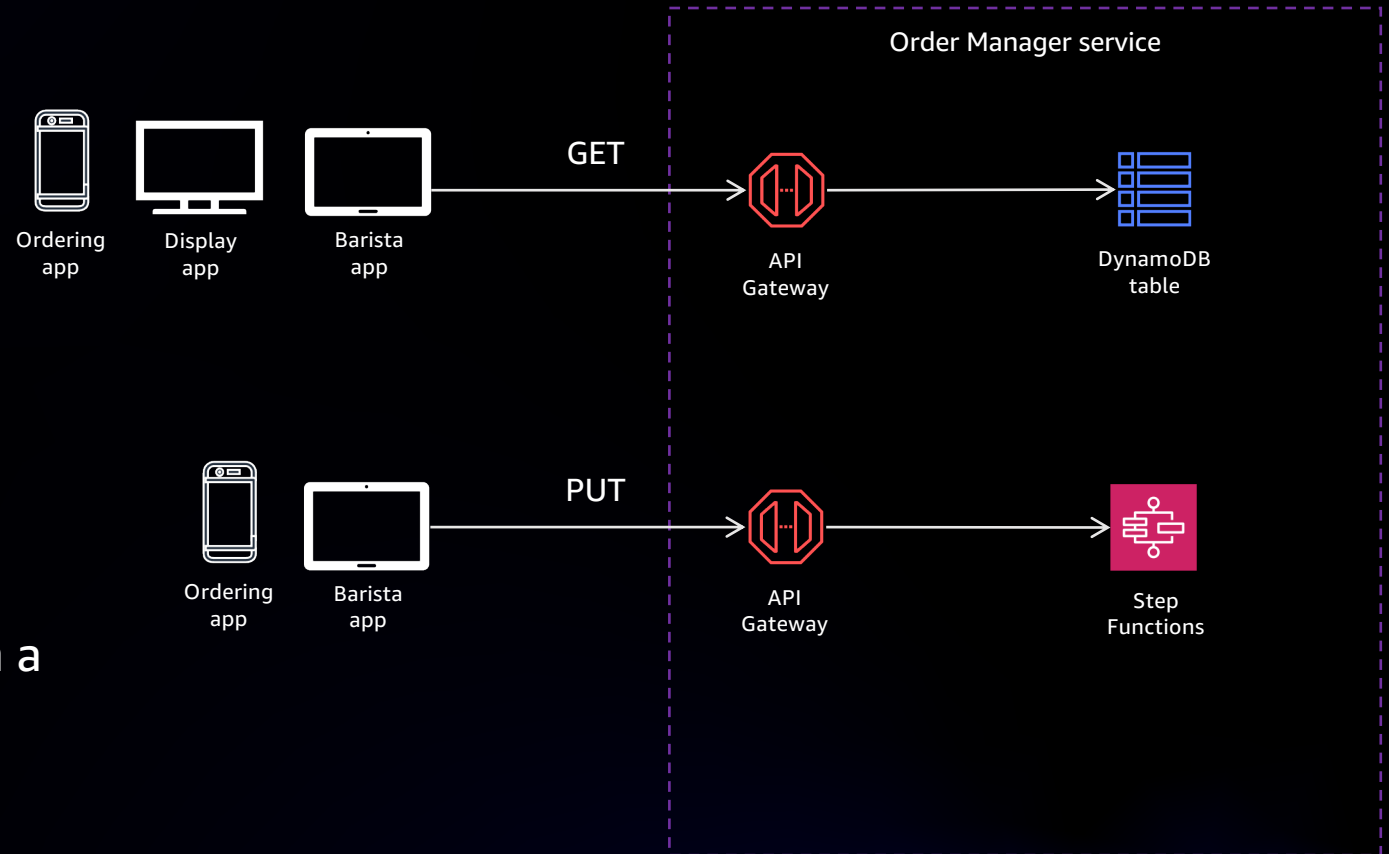
/myOrders - GET

/orders - GET

/orders/{id} - GET

/orders/{id} - PUT

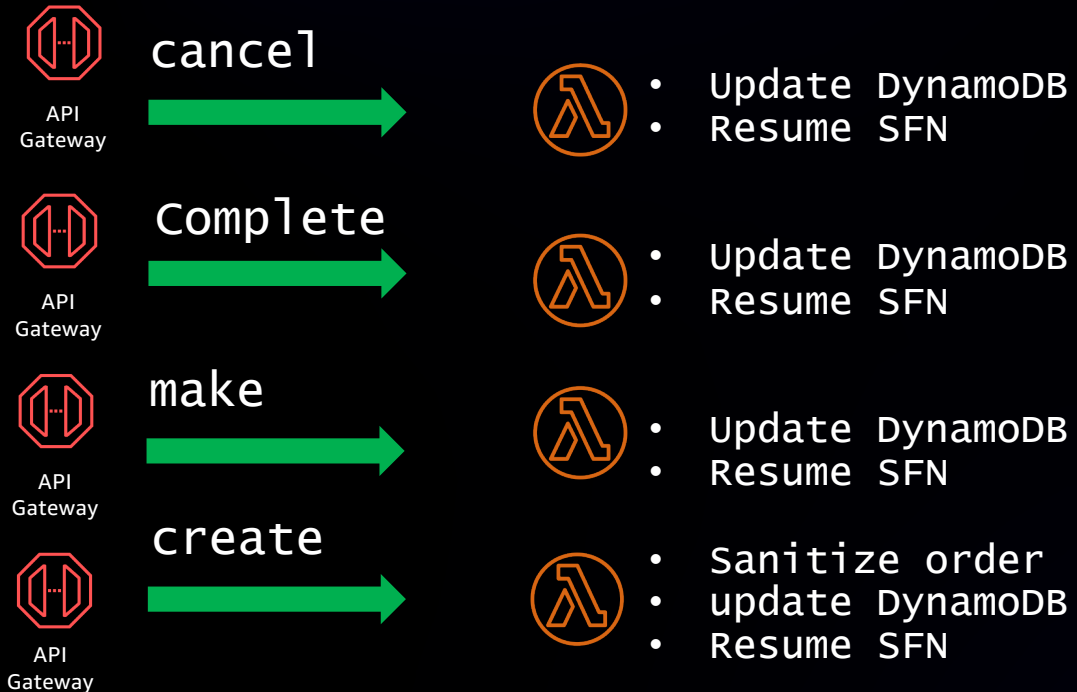
Updates are made to the *Order* table via a PUT request.



Order Manager service as a Lambda function

Version 1

Each operation invokes a Lambda Function



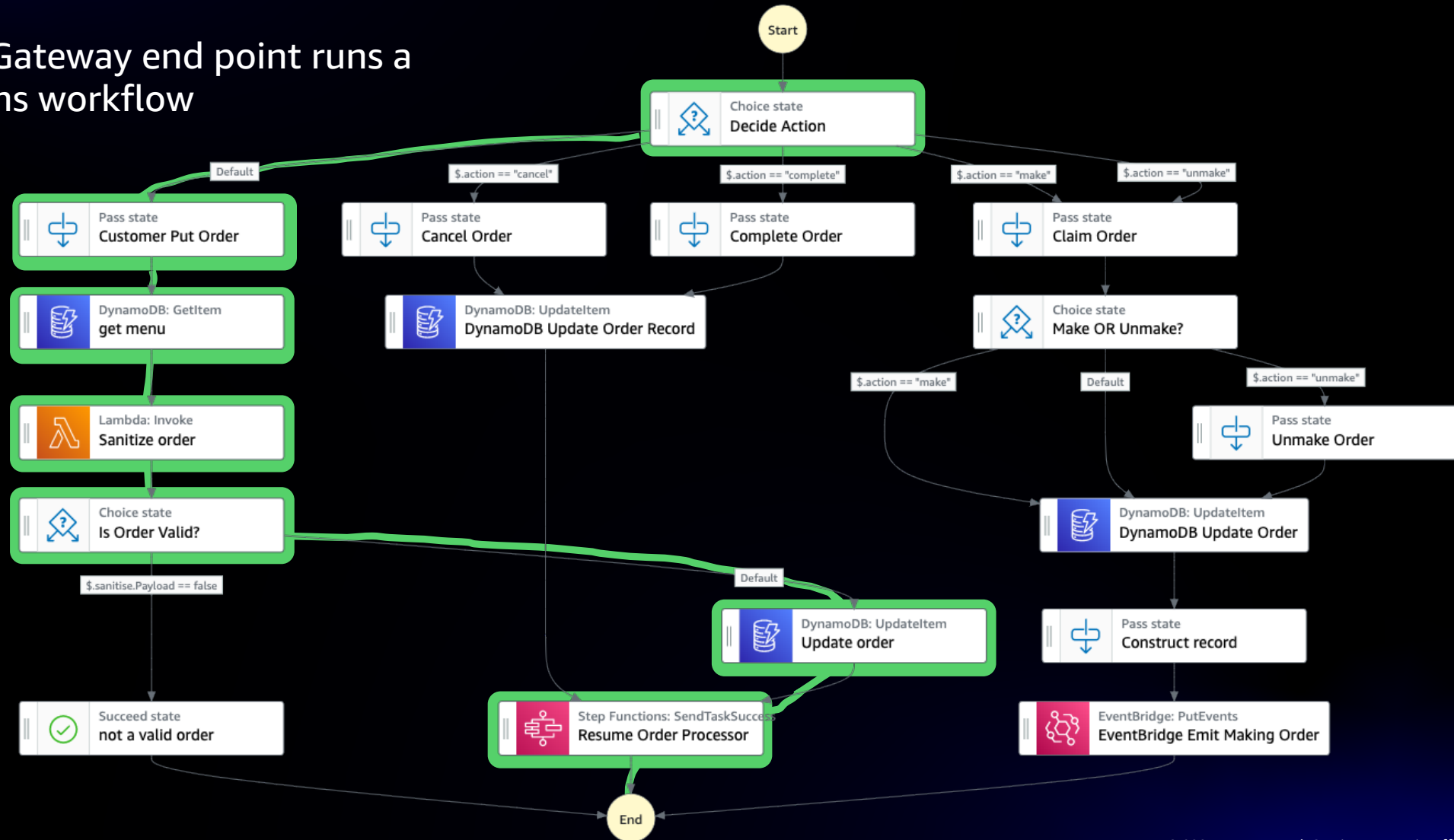
Application grew more complex over time, performing multiple tasks to handle increasingly complex business logic, leading to

- Tightly coupled code base
- Slower release cadence
- Poor discoverability
- Additional complexity

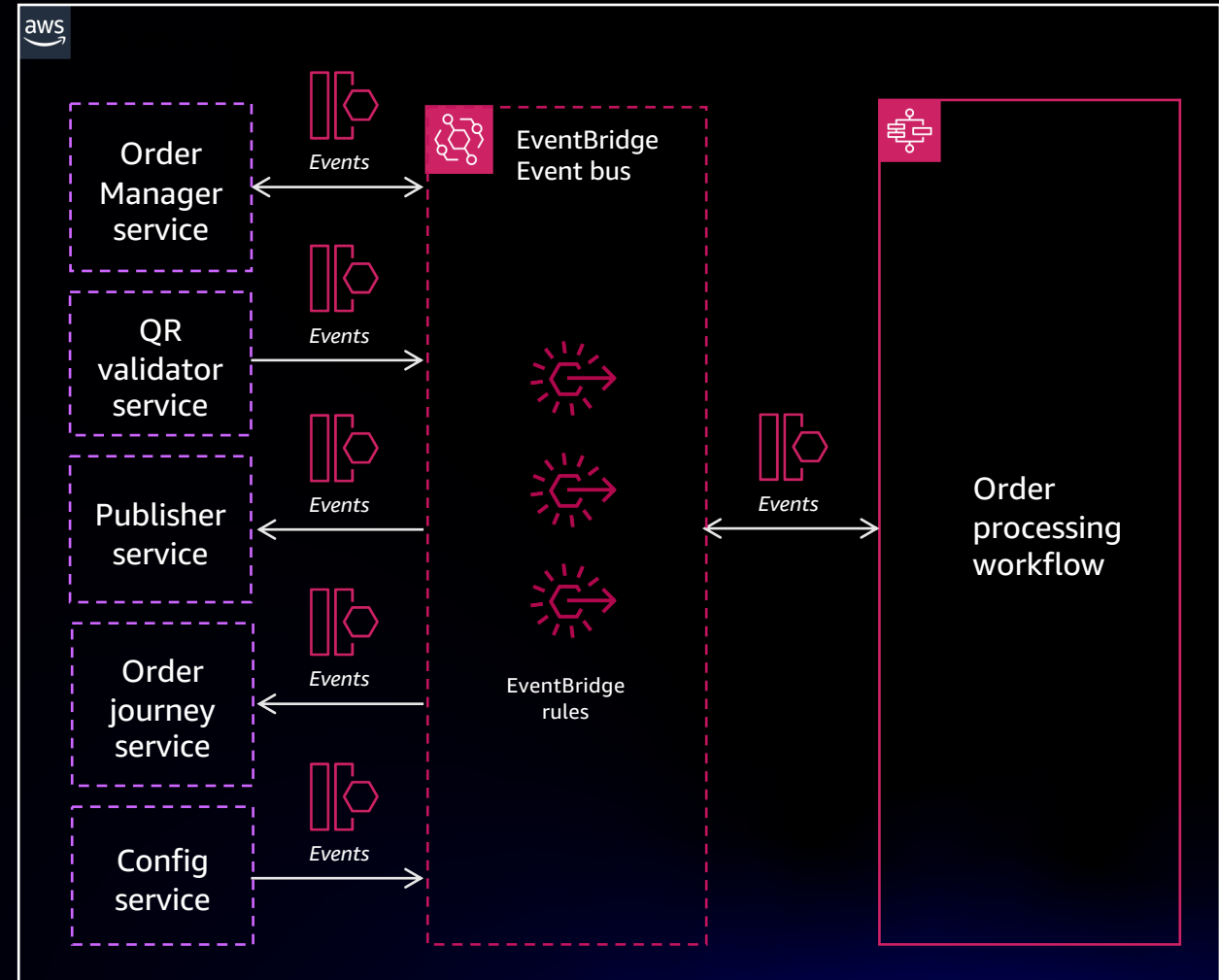
Order Manager service as a workflow

Version 2

A single API Gateway end point runs a Step Functions workflow



Communicate between micro-services using events



What is an event?

USING AMAZON EVENTBRIDGE FOR CHOREOGRAPHING MICROSERVICES

- An event is defined in JSON
- "Detail" is application specific
- Envelope attributes are provided by Amazon EventBridge
- Producers create events
- Consumers choose which events to listen to by using rules

```
{
  "version": "0",
  "id": "6ac4e27b-1234-1234-1234-5fb02c880319",
  "detail-type": "OrderProcessor.OrderStarted",
  "source": "awsserverlessda.serverlesspresso",
  "account": "123456789012",
  "time": "2021-11-28T13:12:30Z",
  "region": "us-west-2",
  "detail": {
    "userId": "jbesw",
    "orderId": "eYmAfqlD67vlbdUViLe_D",
    "drinkOrder": {
      "icon": "barista-icons_cafe-latte",
      "modifiers": [],
      "drink": "Latte"
    }
  }
}
```



serverlesspresso

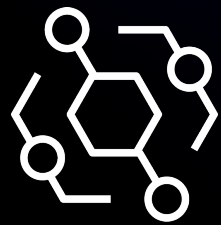
Serverlesspresso events

15 EVENTS

1. ConfigService.ConfigChanged
2. OrderJourney.AllEventsStored
3. OrderManager.MakeOrder
4. OrderManager.OrderCancelled
5. OrderManager.WaitingCompletion
6. OrderProcessor.OrderTimeOut
7. OrderProcessor.ShopNotready
8. OrderProcessor.WaitingCompletion
9. OrderProcessor.WaitingProduction
10. OrderProcessor.WorkflowStarted
11. QueueService.OrderCancelled
12. QueueService.OrderCompleted
13. Validator.NewOrder
14. QueueService.OrderStarted
15. OrderManager.OrderCompleted



serverlesspresso



Amazon EventBridge

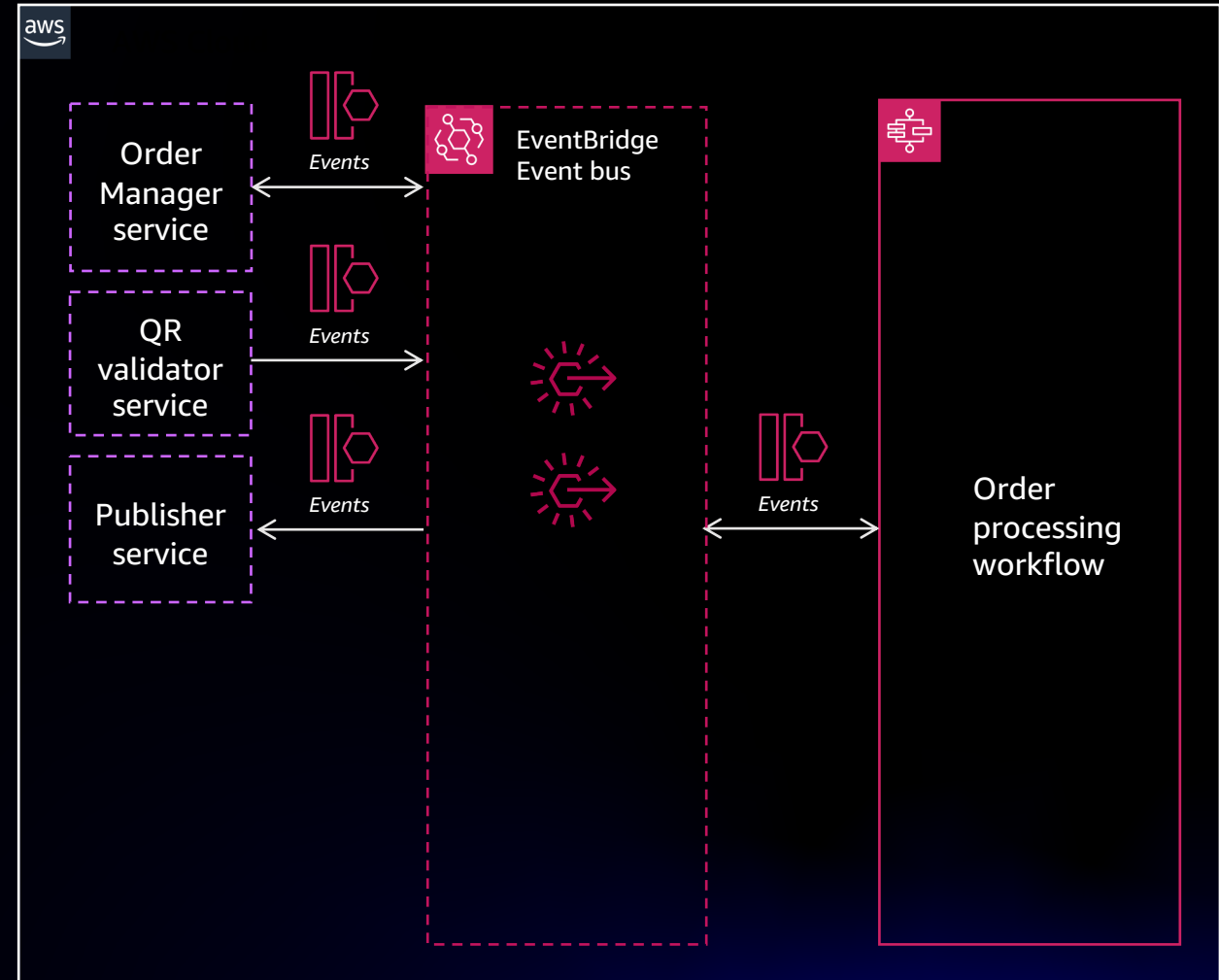
A serverless event bus service for AWS services, your own applications, and SaaS providers

- 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

Event-driven architectures

USING AMAZON EVENTBRIDGE FOR CHOREOGRAPHING MICROSERVICES

- Event flow drives the application
- Events choreograph the services, while Step Functions orchestrates the transactions

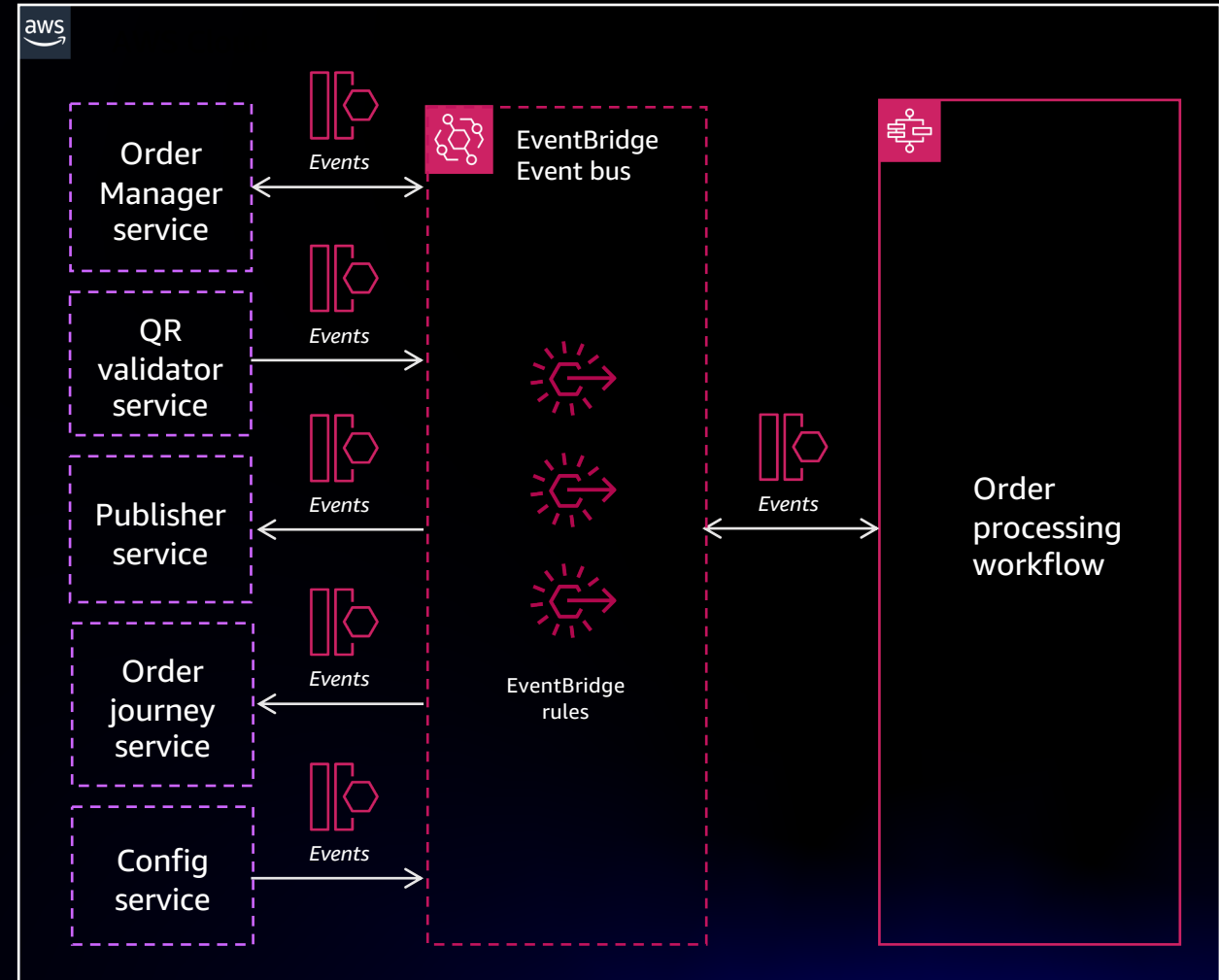


serverlesspresso

Event-driven architectures

USING AMAZON EVENTBRIDGE FOR CHOREOGRAPHING MICROSERVICES

- Event flow drives the application
- Events choreograph the services, while Step Functions orchestrates the transactions
- Add new microservices as event consumers without changing existing code
- Microservices emit events independently of consumers

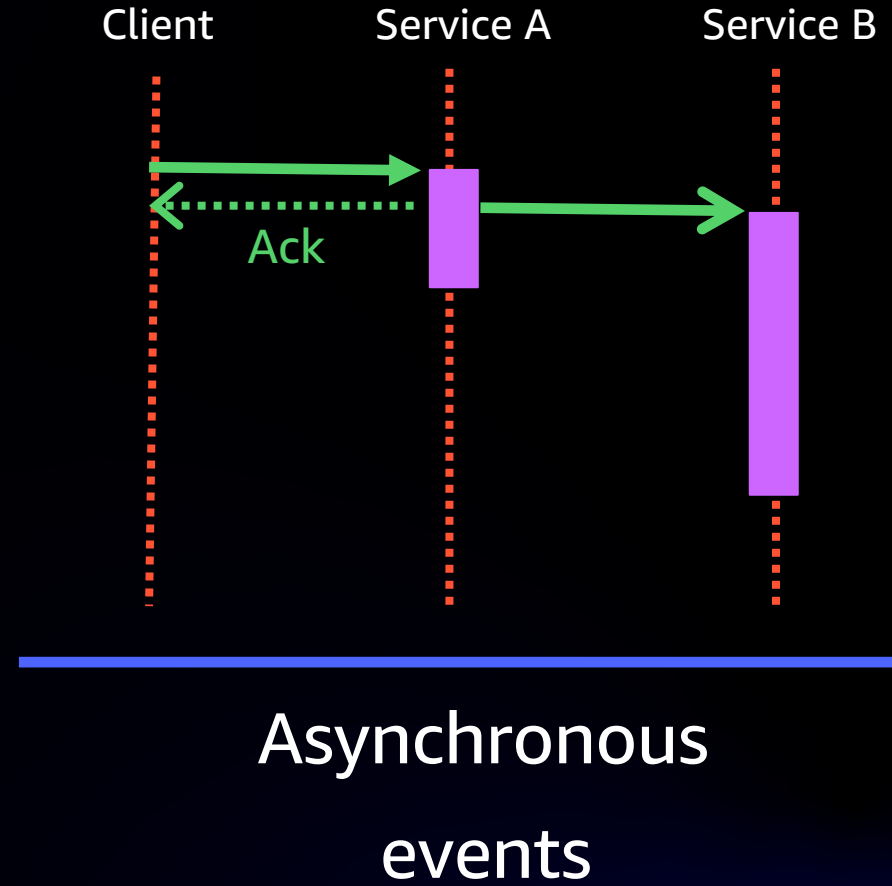


serverlesspresso

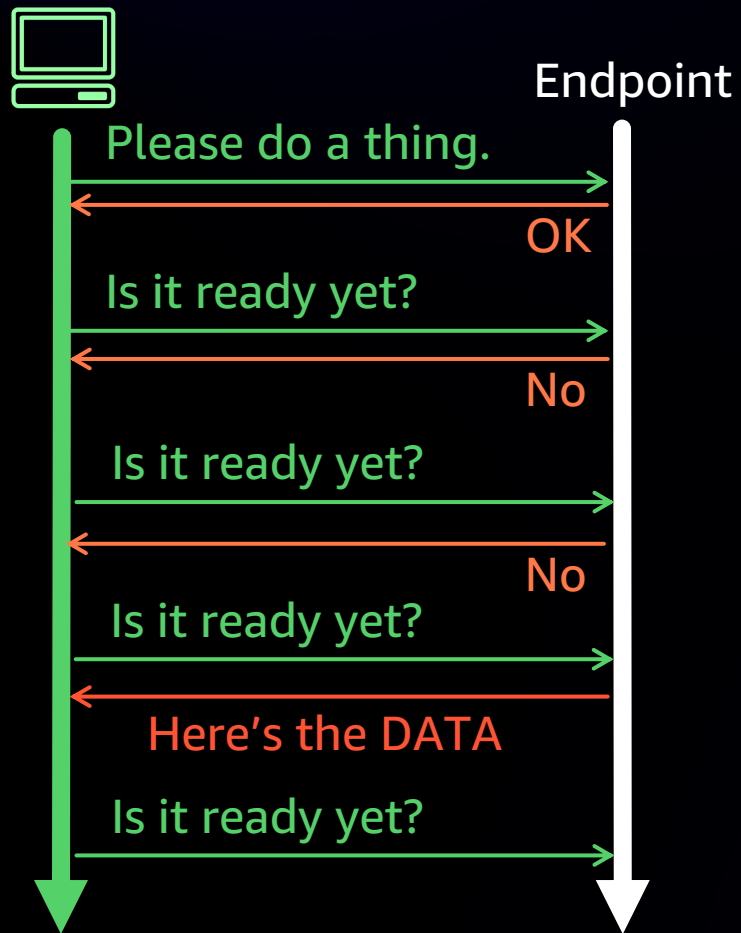
Handling async responses with real-time updates

Handling response values and state for asynchronous requests

No return path to provide further information beyond the initial acknowledgement

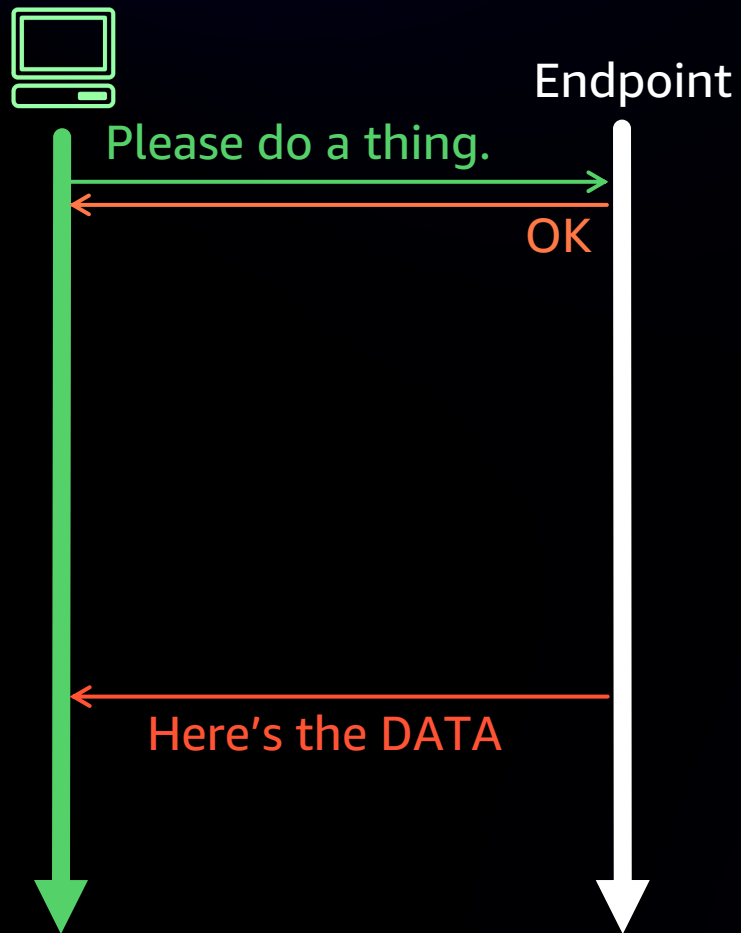


Tracking an inflight request: Polling



- Initial request returns a tracking identifier
- Create a second API endpoint for the front end to check the status of the request, referencing the tracking ID
- Use DynamoDB or another data store to track the state of the request
- Simple mechanism to implement
- Can create many empty calls
- Delay between availability and front-end notification

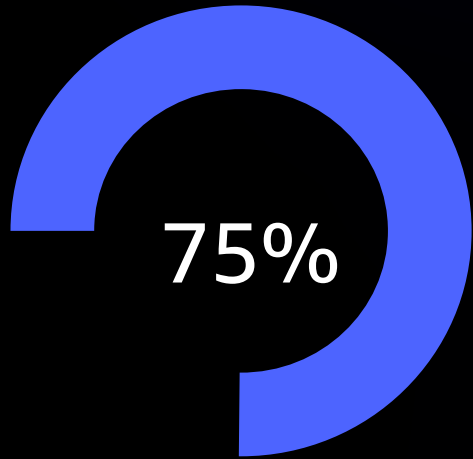
Tracking an inflight request: WebSocket



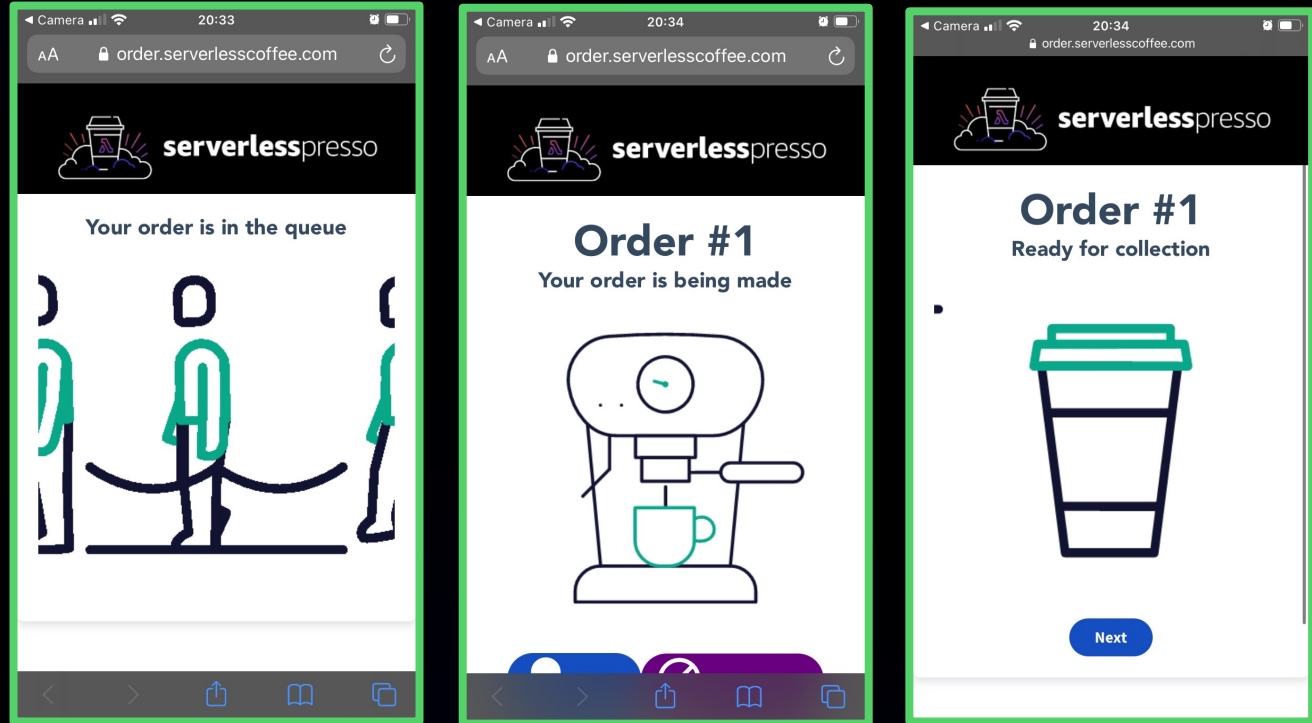
- A bidirectional connection between the front end client and the backend service
- Your backend services can continue to send data back to the client by using a WebSocket connection
- Closer to real time
- Reduces the number of messages between the client and backend system
- Often more complex to implement

Using AWS IoT Core for real-time messaging

Web applications often require partial information:



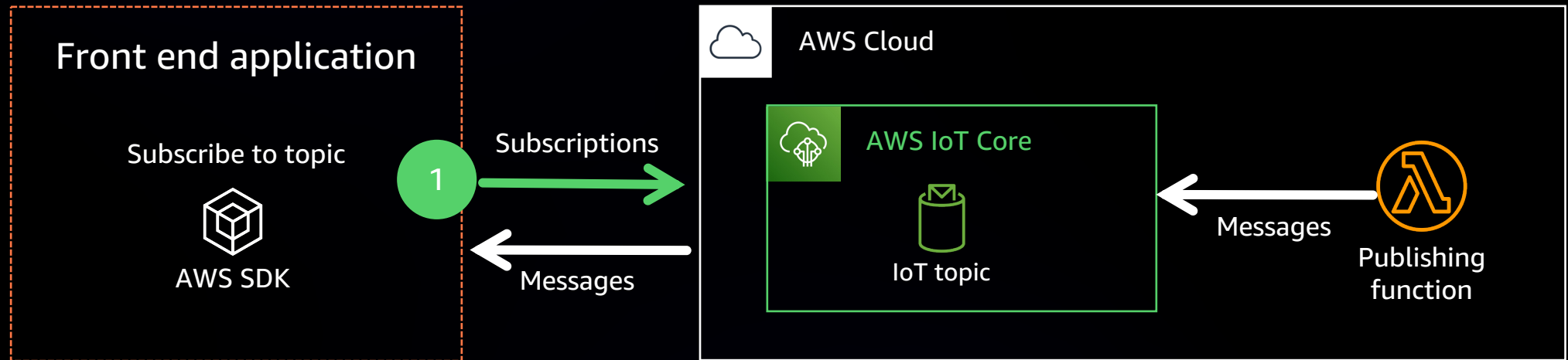
Completion percentages



Continuous data changes

Using AWS IoT Core for real-time messaging

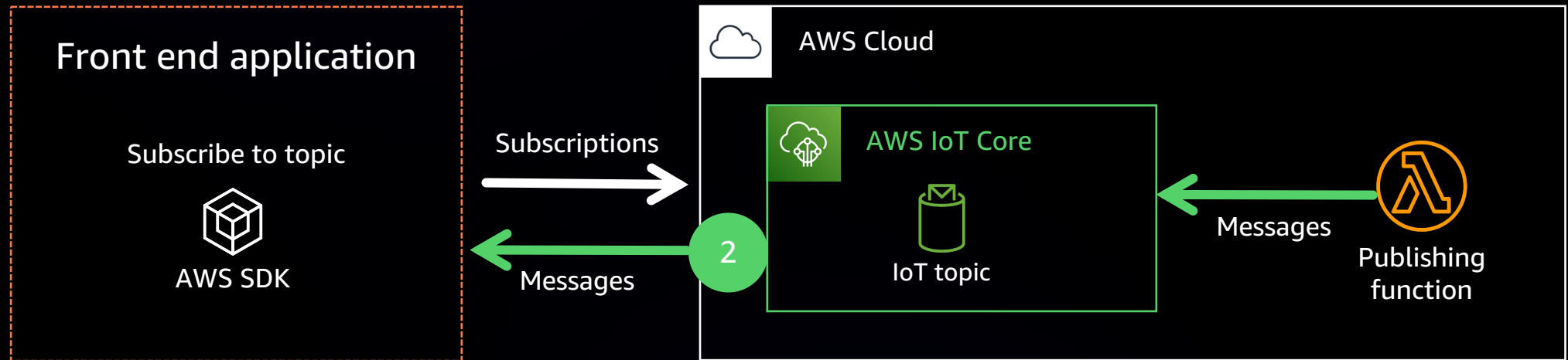
Front end application uses pub-sub to “listen” for event updates



Front end uses AWS SDK to subscribe to a topic based on user's unique user ID.

Using AWS IoT Core for real-time messaging

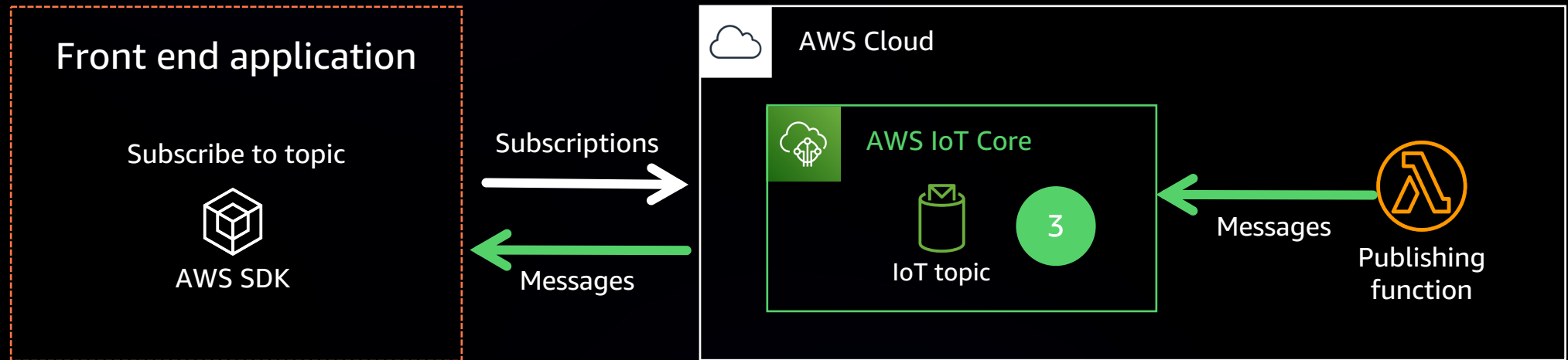
Front end application uses pub-sub to “listen” for event updates



Receives messages published by the backend to this topic.

Using AWS IoT Core for real-time messaging

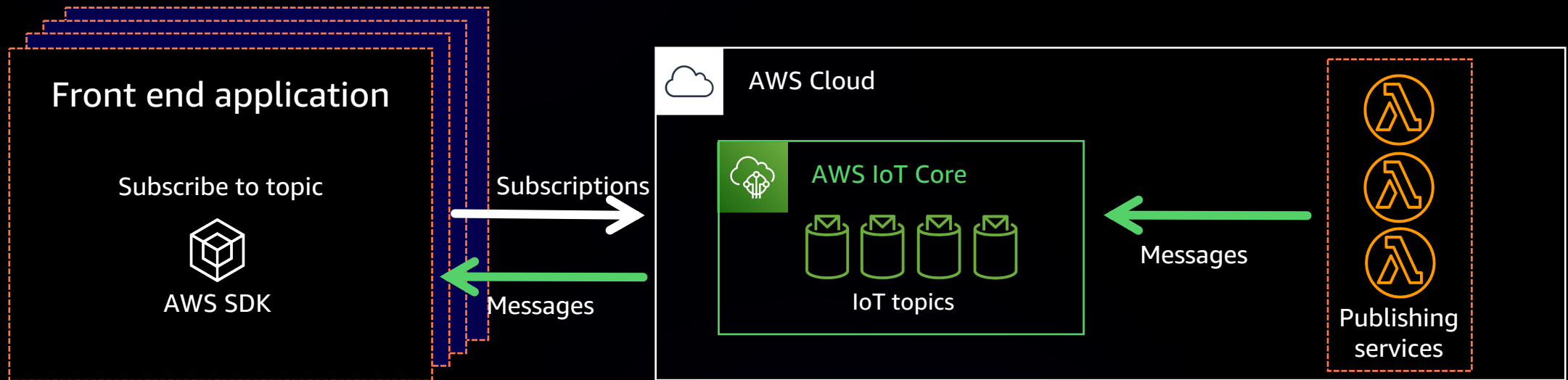
Front end application uses pub-sub to “listen” for event updates



Messages are categorized using topics. Topic names are UTF 8 encoded strings.

Using AWS IoT Core for real-time messaging

Front end application uses pub-sub to “listen” for event updates



The IoT core service manages the WebSocket connection between backend publishers and front-end subscribers.

This enables **fanout** functionality to thousands front-end devices.

Using AWS IoT Core for real-time messaging

The *IoTData* class in the AWS SDK returns a client that uses the MQTT protocol

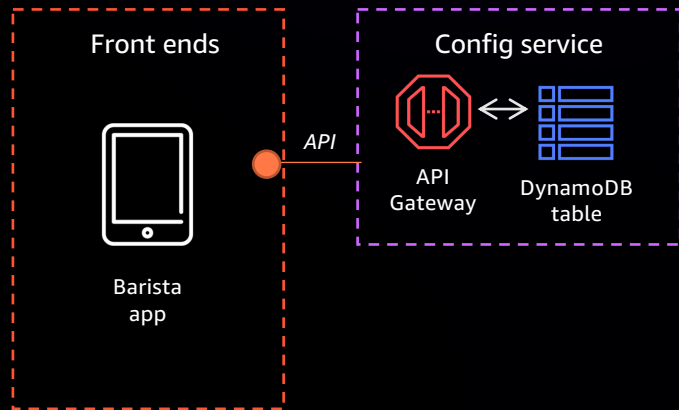
Front End app

```
mqttClient.on('connect', function () {  
    console.log('mqttClient connected')  
})  
  
mqttClient.on('error', function (err) {  
    console.log('mqttClient error: ', err)  
})  
  
mqttClient.on('message', function (topic, payload) {  
    const msg = JSON.parse(payload.toString())  
    console.log('IoT msg: ', topic, msg)  
})
```

Once the frontend application establishes the connection, it returns messages, errors, and connection status via callbacks.

Combining multiple approaches for your front-end application

Many front-end applications can combine synchronous and asynchronous response models.



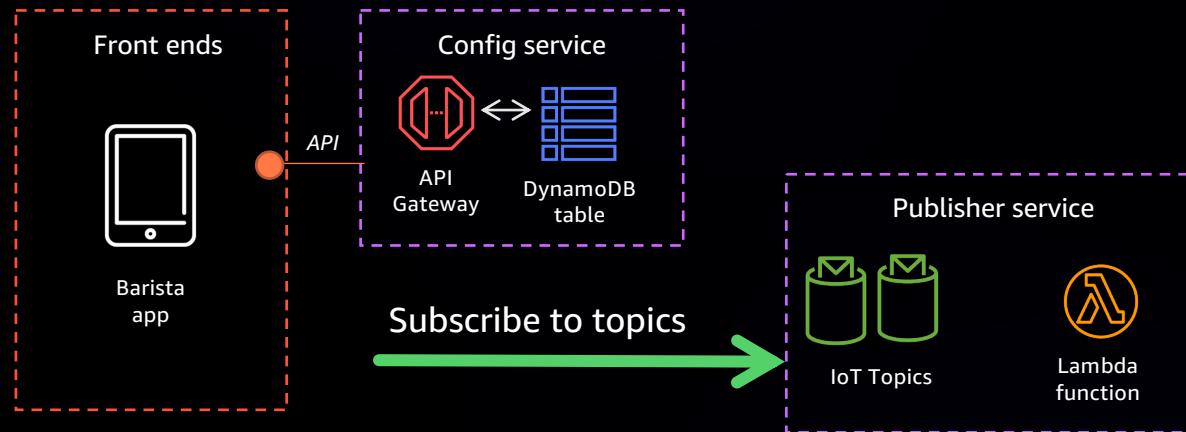
Serverlesspresso sends an initial synchronous request to retrieve the current “state of things.”



serverlesspresso

Combining multiple approaches for your front-end application

Many front-end applications can combine synchronous and asynchronous response models.

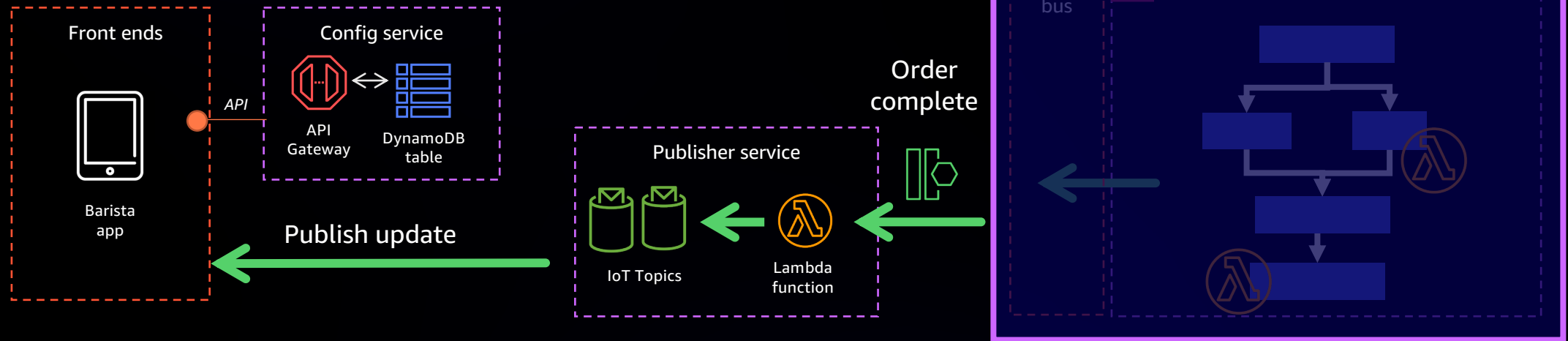


Simultaneously the front end subscribes to global and user-based IoT topics.



serverlesspresso

Combining multiple approaches for your front-end application



New orders are posted to the application, and processed asynchronously, with updates published to the front end via the IoT topic.



serverlesspresso

What does it cost to run this workload?

We can serve up to 960 drinks to 960 customers every day

Service	Daily cost	With Free Tier
AWS Amplify Console	\$0.28	Free
Amazon API Gateway	\$0.01	Free
Amazon Cognito	\$0.00	Free
Amazon DynamoDB	\$0.01	Free
Amazon EventBridge	\$0.01	Free
AWS IoT Core	\$0.01	Free
AWS Lambda	\$0.01	Free
Amazon SNS	\$7.98	\$7.98
AWS Step Functions	\$0.29	Free



serverlesspresso

Learn about the AWS Free Tier:
<https://aws.amazon.com/free>

Summary



serverlesspresso

An event driven coffee ordering app
built with serverless architecture

Use Step Functions to orchestrate resources within a microservice

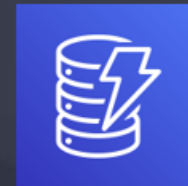
Use events to communicate between microservices

Use IoT Core to maintain open connection for asynchronous responses

[New](#)[Blogs](#)[Videos](#)[Learn](#)[Events ▾](#)[Patterns](#)[About](#)

Welcome to Serverless Land

This site brings together all the latest blogs, videos, and training for AWS Serverless. Learn to use and build apps that scale automatically on low-cost, fully-managed serverless architecture.

[Learn More](#)

For more serverless learning resources, visit:

<https://serverlessland.com>

Thank you
@edjgeek

DON'T FORGET TO **RATE THE SESSIONS**

#GOTOaar

Rate a minimum of **5 sessions** and
claim your **reward** at the
Registration Desk at the Trifork Hall