Serverless Workshop

Let‘s have a cup of coffee …

Lars Röwekamp | @**mobile**Larson | #**WISSEN**TEILEN

Lars Röwekamp

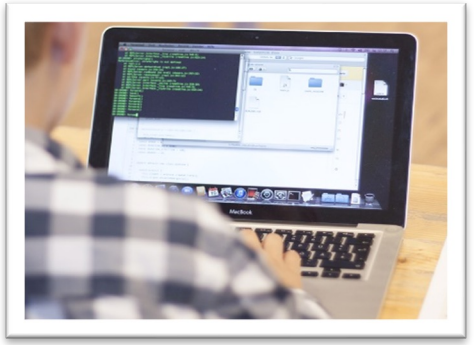
@mobile Larson

CIO New Technologies
OPEN KNOWLEDGE

(Architecture, Microservices, Cloud, AI & ML)

Und **da** arbeite ich >

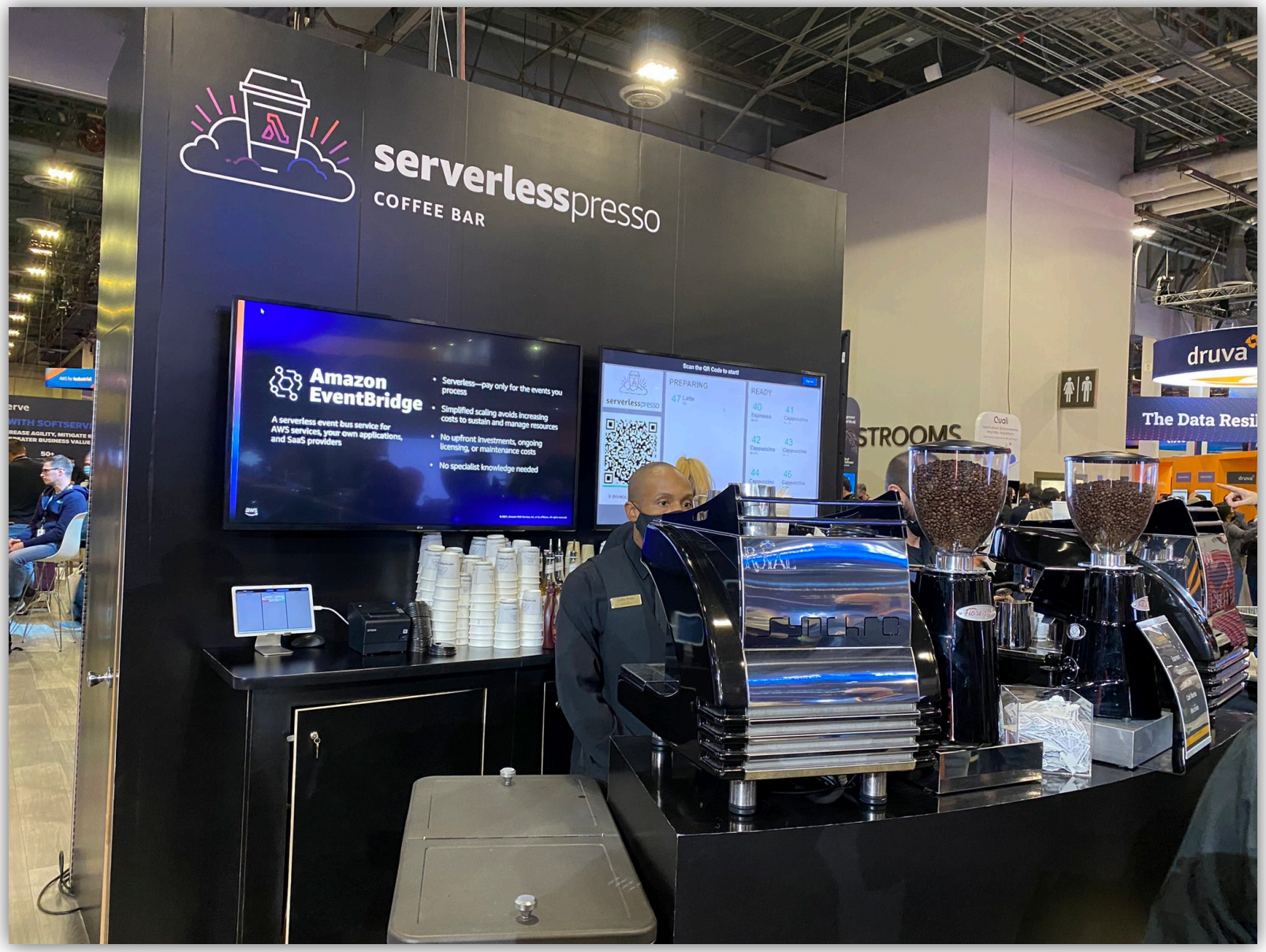
OPEN KNOWLEDGE
@_**open**knowledge




Let‘s start

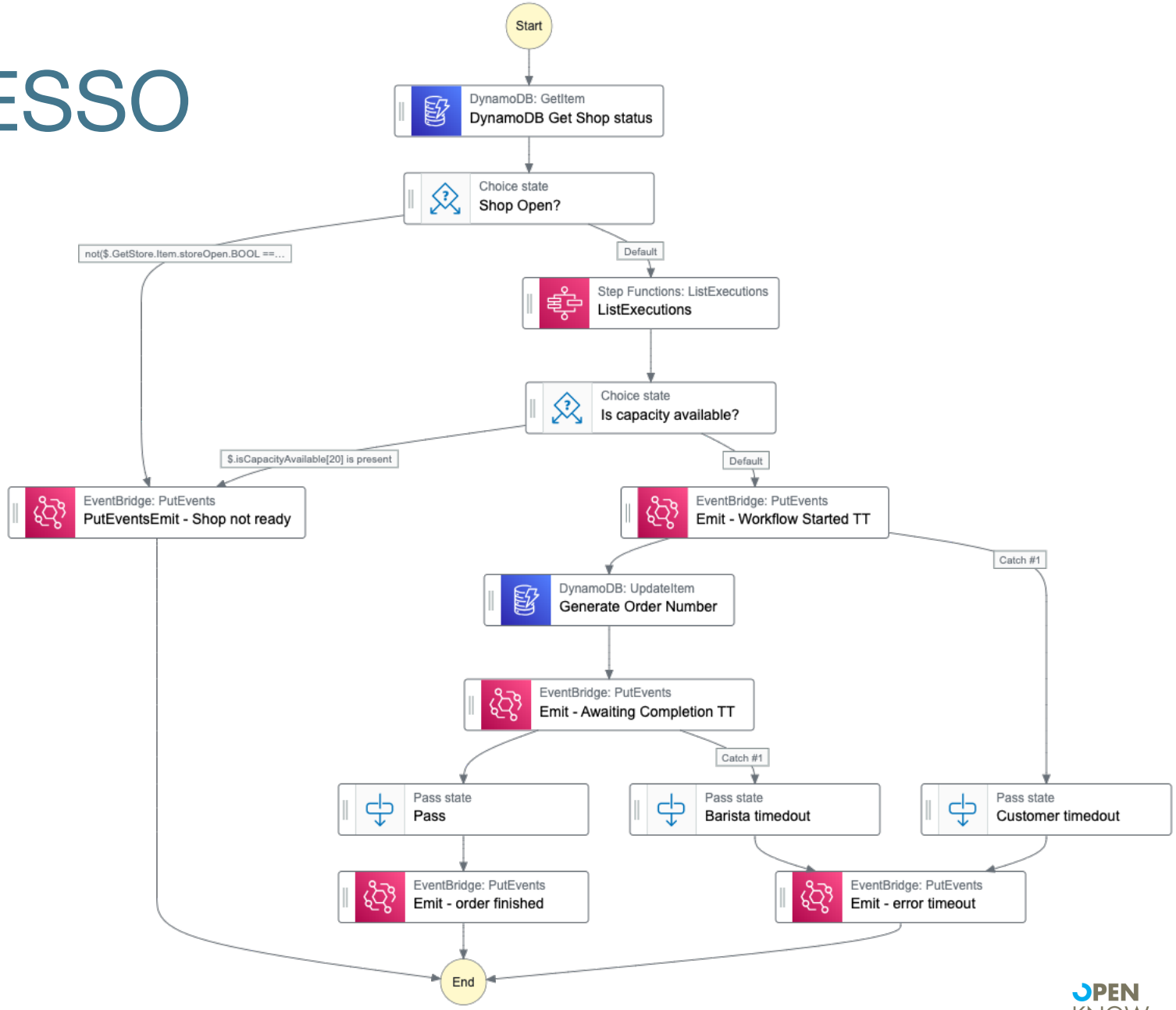
Serverless Workshop
Show-Case

OPEN KNOW LEDGE


serverlesspresso
COFFEE BAR
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
Scan the QR Code to start
PREPARING
READY
druva
The Data Resi

SERVERLESSPRESSO

Order Workflow



SERVERLESSPRESSO

Order Workflow

- **Kunde** startet Order via Code Scan.

SERVERLESSPRESSO

Order Workflow

Kunde startet Order via Code Scan.

Coffee Shop geöffnet? Freie Kapazitäten?

SERVERLESSPRESSO

Order Workflow

Kunde startet Order via Code Scan.

Coffee Shop geöffnet? Freie Kapazitäten?

Kunde hat 5 min seine Order zu platzieren.

SERVERLESSPRESSO

Order Workflow

Kunde startet Order via Code Scan.

Coffee Shop geöffnet? Freie Kapazitäten?

Kunde hat 5 min seine Order zu platzieren.

Barista hat 15 min die Order zu bearbeiten.

SERVERLESSPRESSO

Order Workflow

Kunde startet Order via Code Scan.

Coffee Shop geöffnet? Freie Kapazitäten?

Kunde hat 5 min seine Order zu platzieren.

Barista hat 15 min die Order zu bearbeiten.

Bestellung **zubereitet**

SERVERLESSPRESSO

Order Workflow

Kunde startet Order via Code Scan.

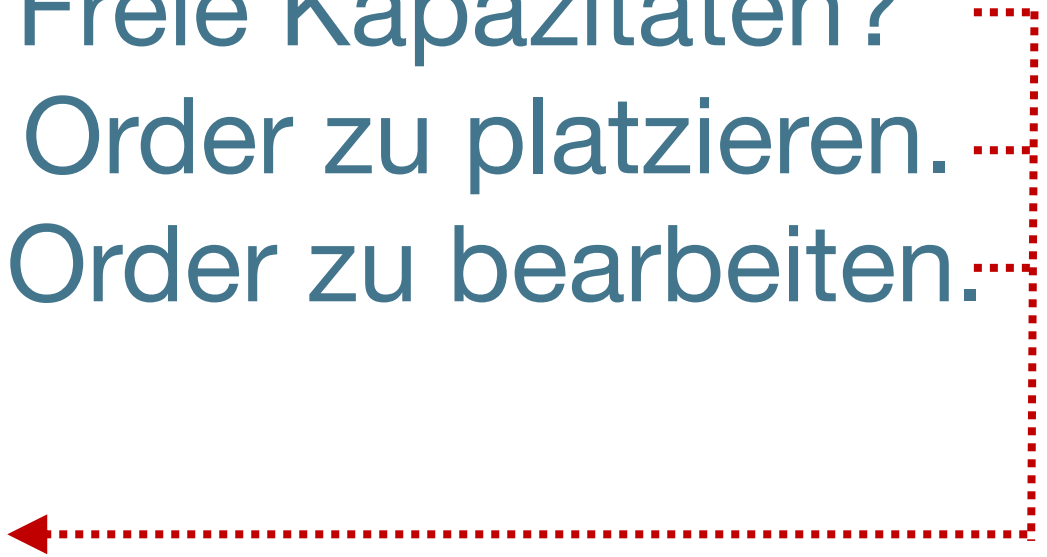
Coffee Shop geöffnet? Freie Kapazitäten?

Kunde hat 5 min seine Order zu platzieren.

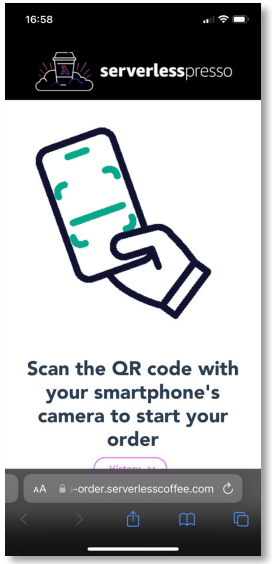
Barista hat 15 min die Order zu bearbeiten.

Bestellung **zubereitet**

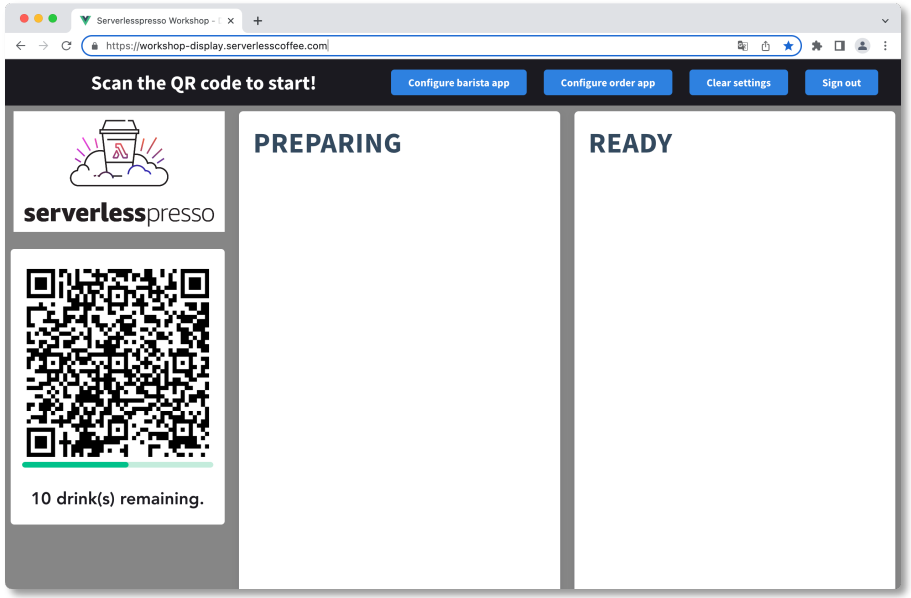
Bestellung **canceled**



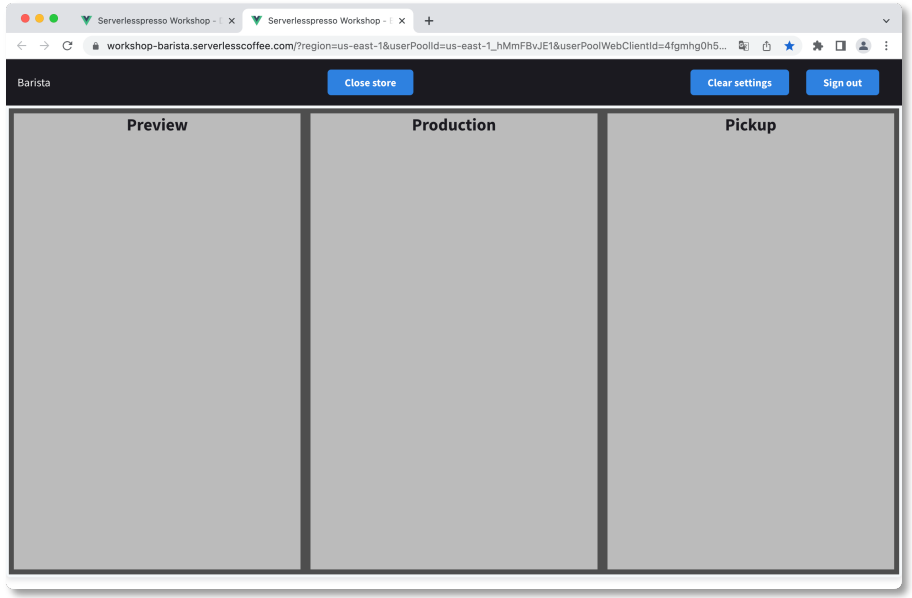
Order App



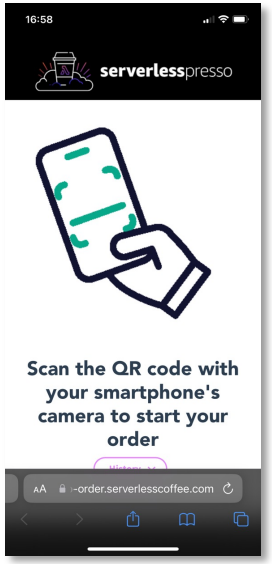
Coffeeshop Display App



Barista App

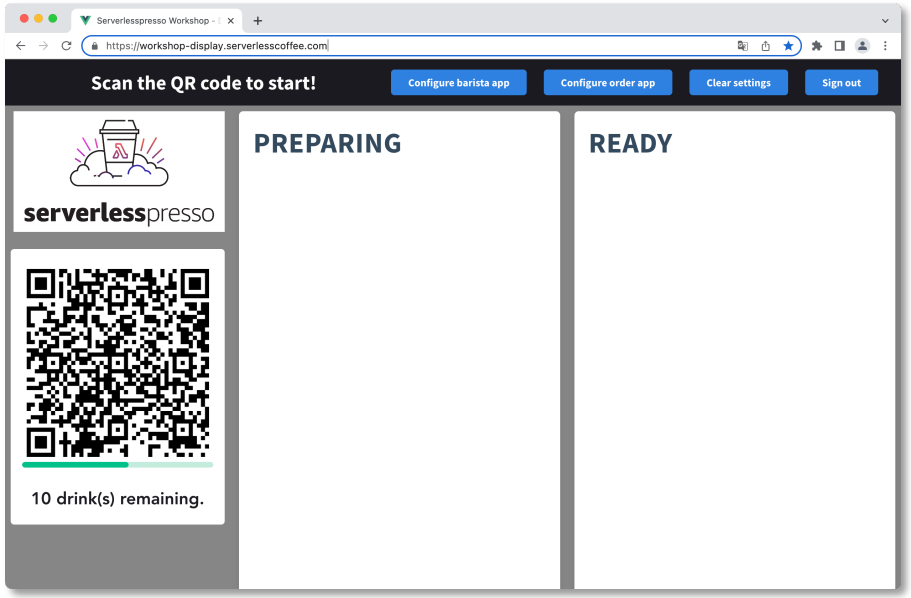


Order App



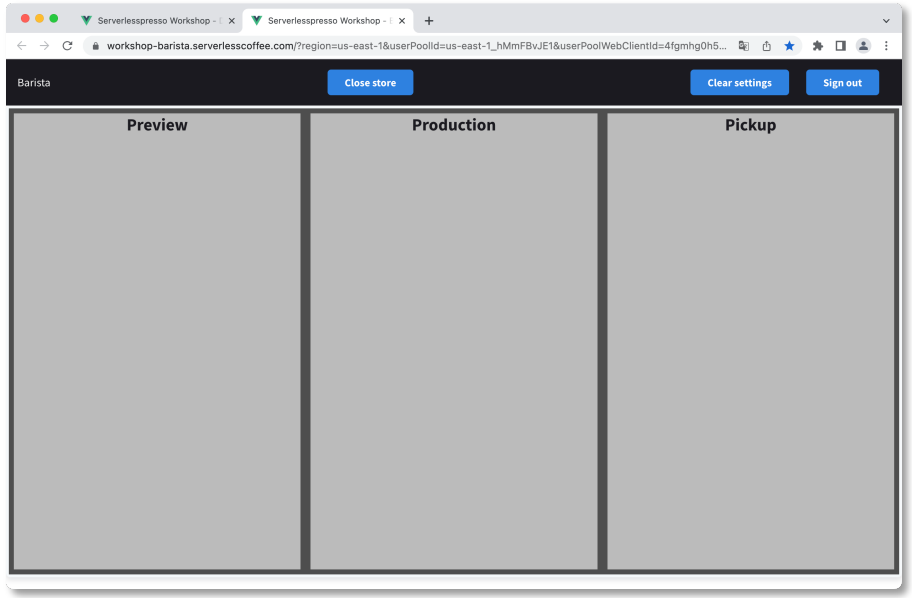
App is ready to scan code.

Coffeeshop Display App



No orders in preparation.
Remaining capacity is 10.

Barista App

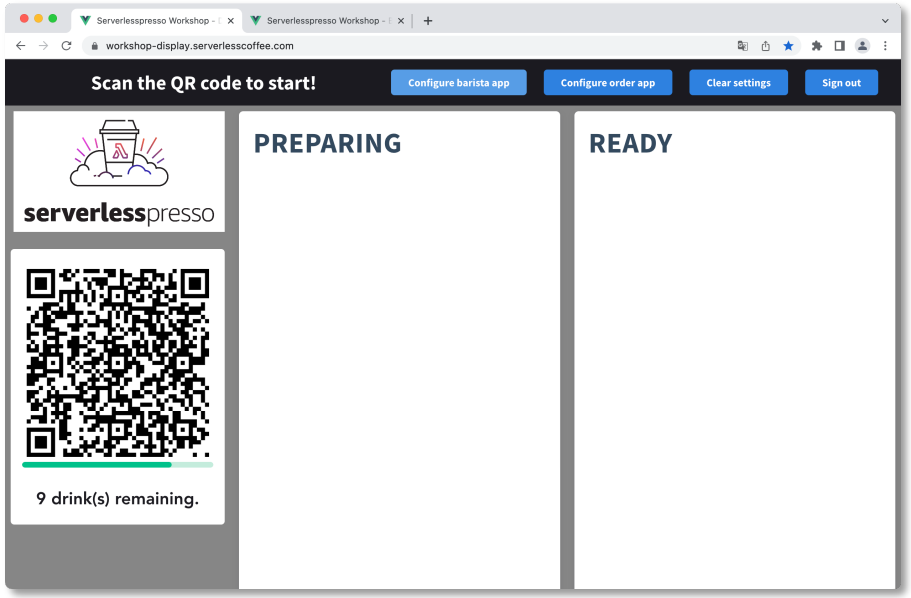


No orders in queue
or in preparation.

Order App

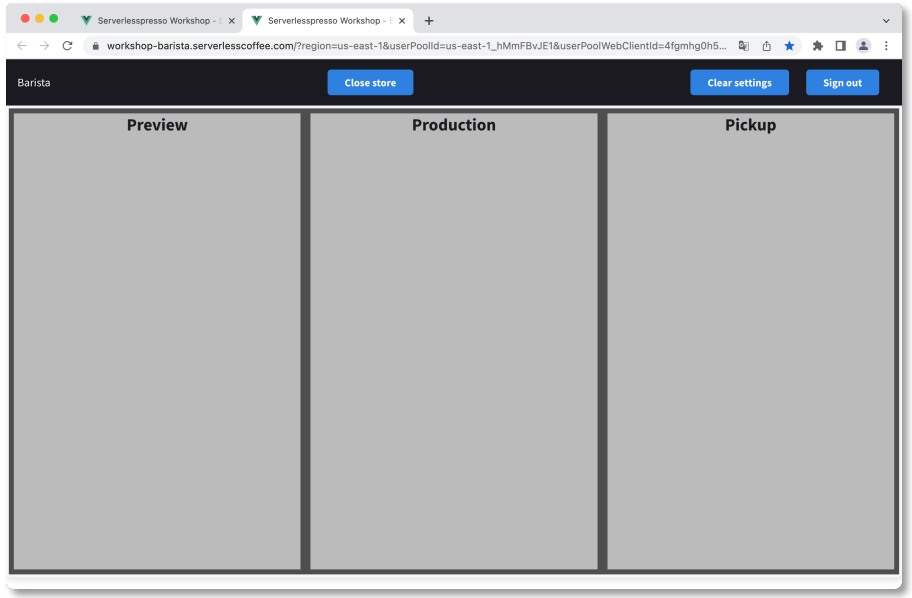
Code scanned.

Coffeeshop Display App



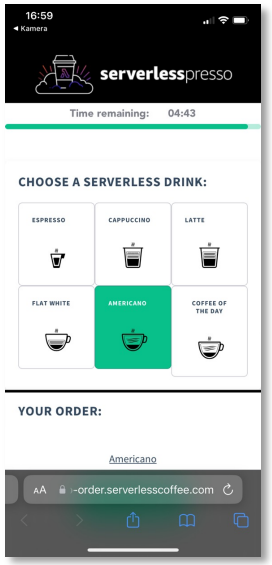
5 minute counter started.
Remaining capacity is 9.

Barista App



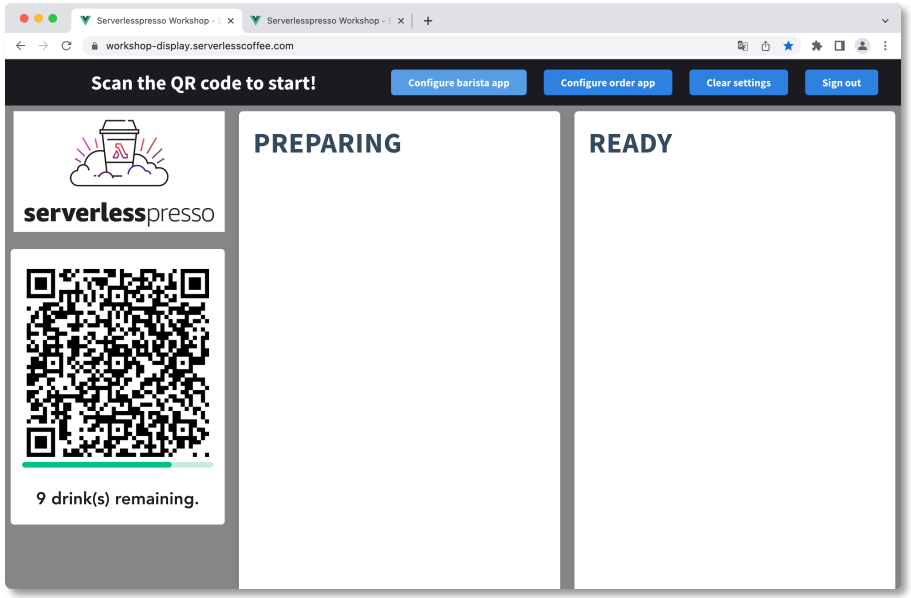
No orders in queue
or in preparation.

Order App



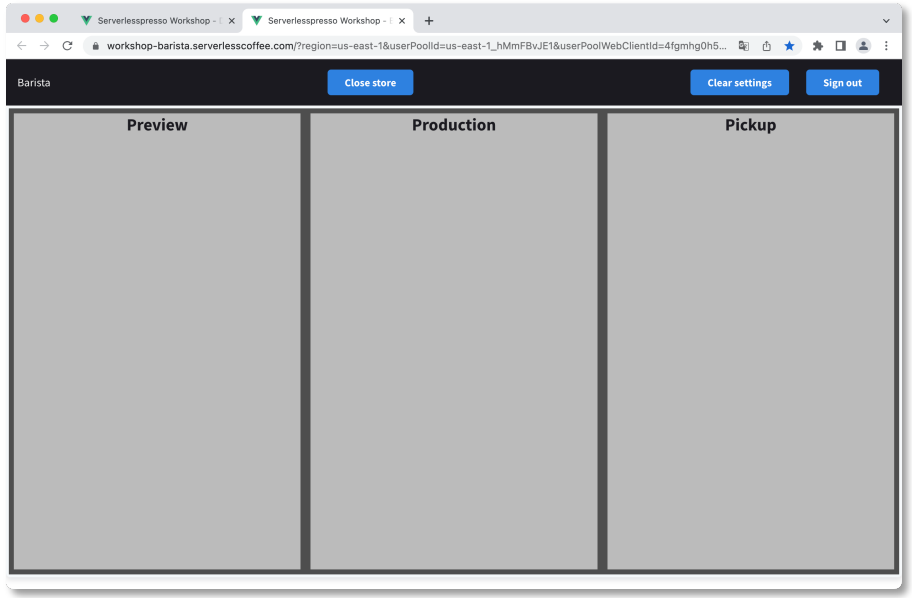
Drink to order selected.

Coffeeshop Display App



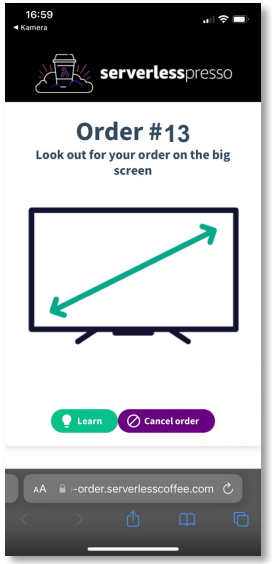
Waiting for oustanding order and for other customers.

Barista App



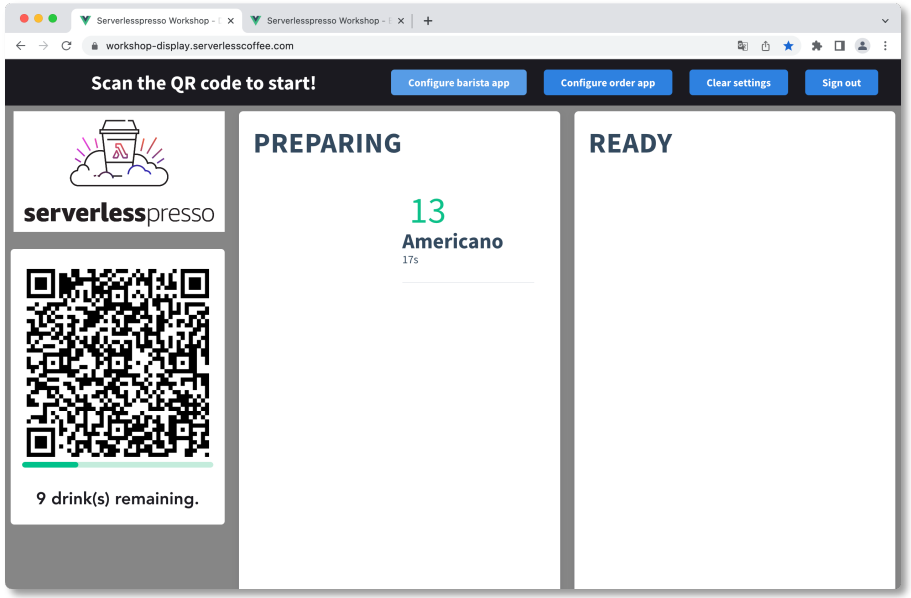
No orders in queue or in preparation.

Order App



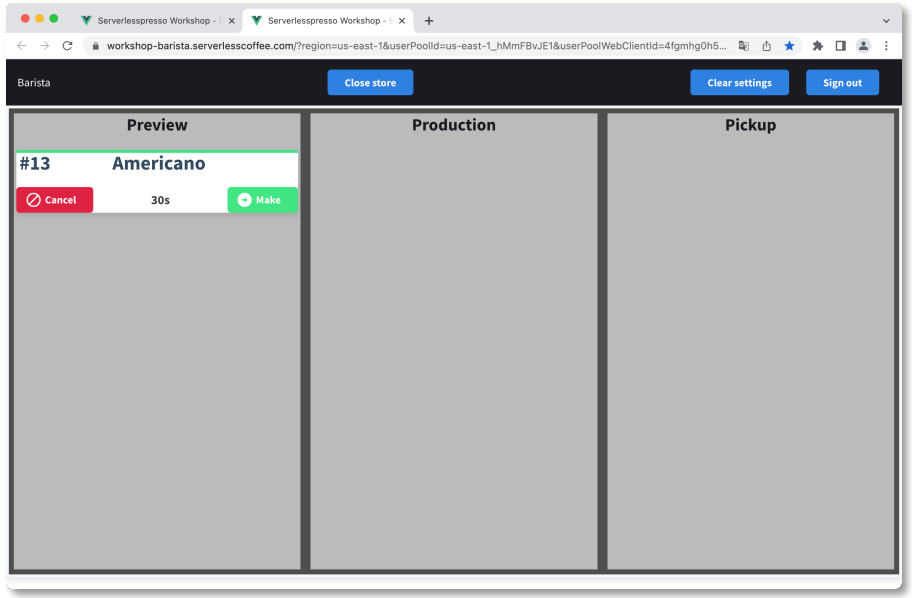
Selected drink ordered.

Coffeeshop Display App



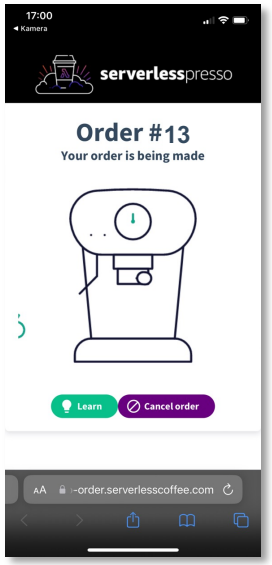
Order is in preparation (or at least queued).

Barista App



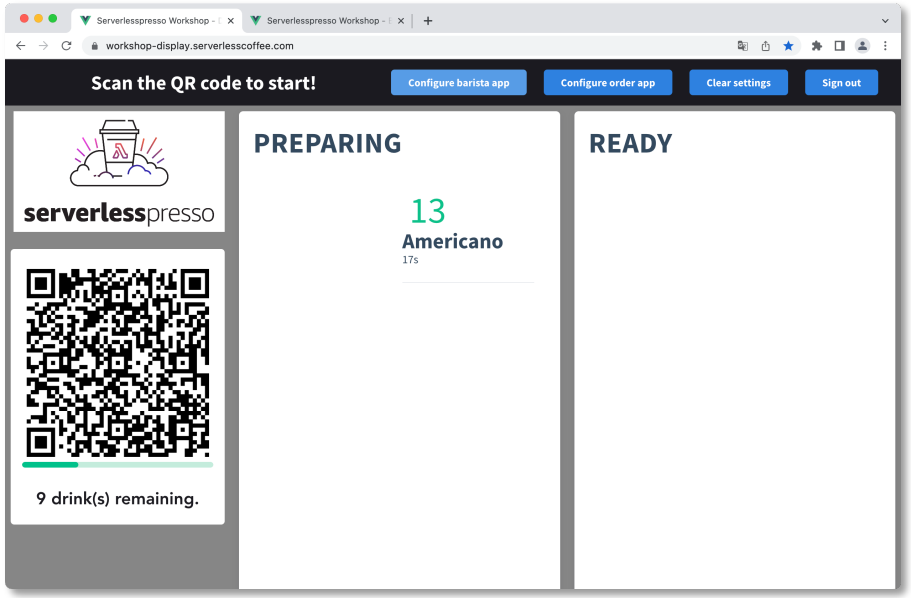
Some work to do. Lets hurry, only 15 minutes left.

Order App



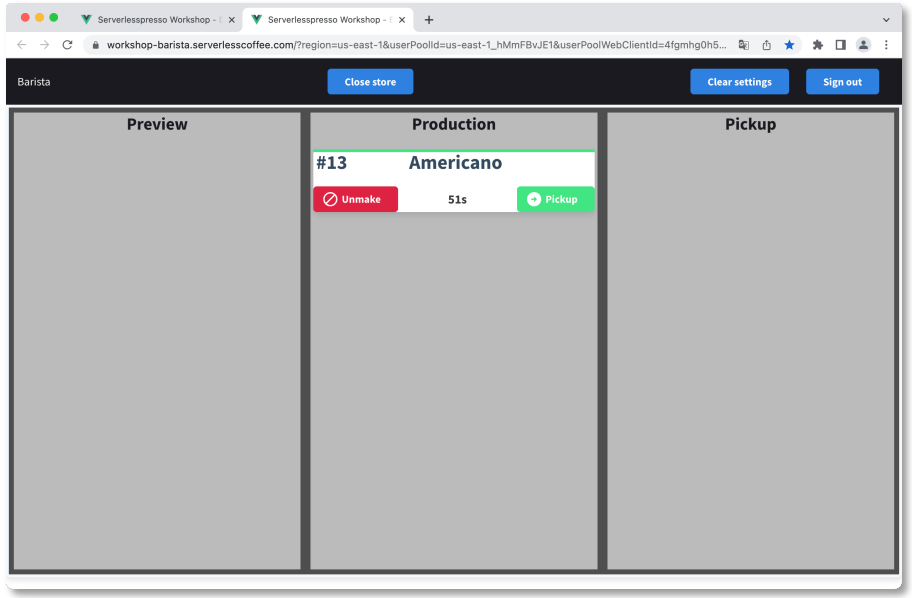
Waiting for order

Coffeeshop Display App



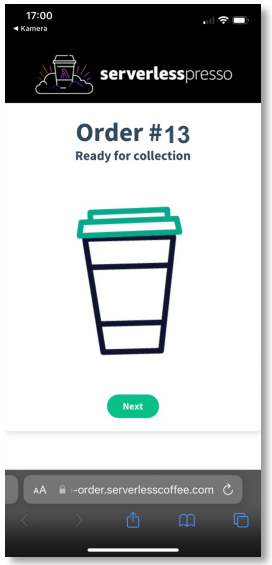
Order is in preparation.
Waiting for other orders.

Barista App



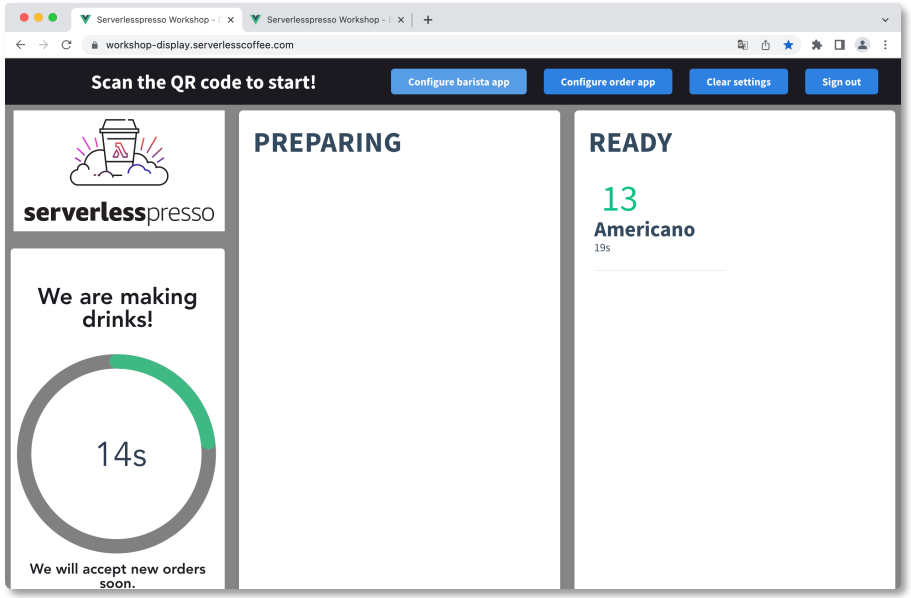
Order is in production.
Waiting for other orders.

Order App



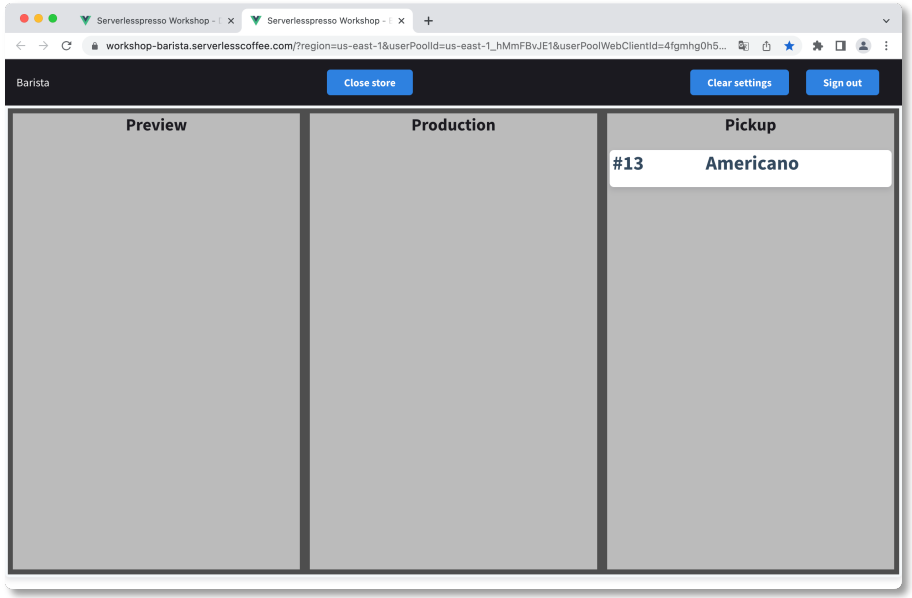
Order is ready
for pick up

Coffeeshop Display App



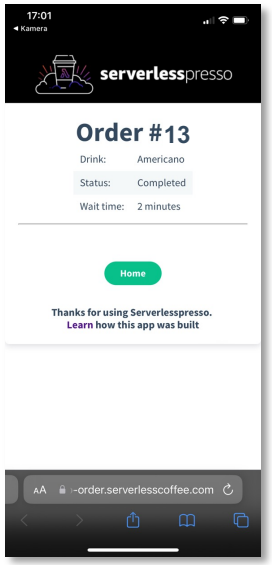
Order is ready.
Waiting for other orders.

Barista App



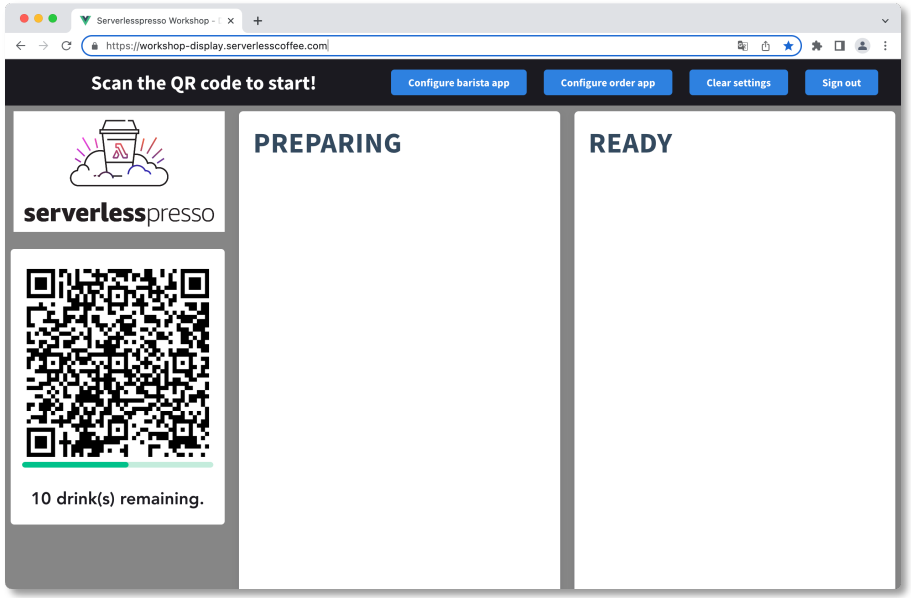
Order production is done.
Waiting for other orders.

Order App



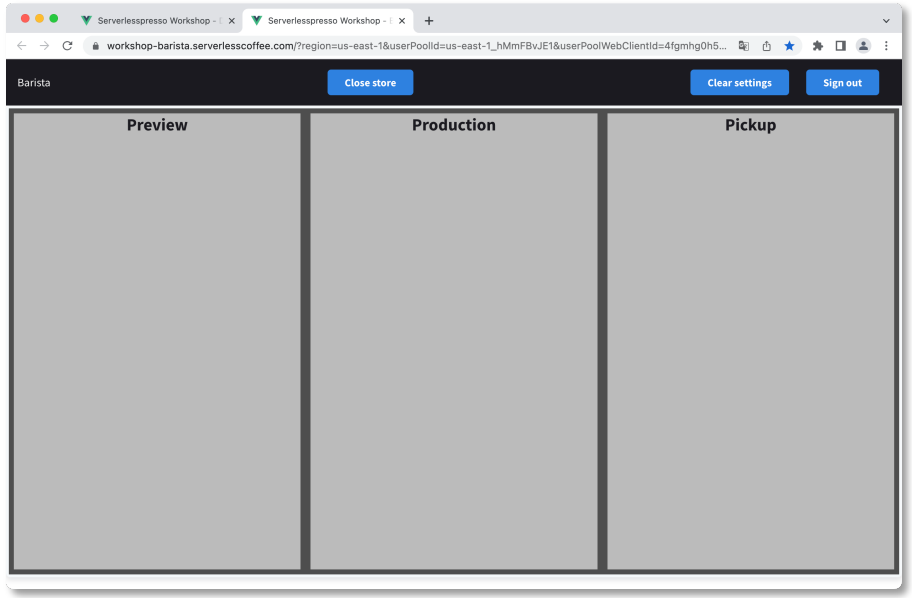
Order picked up

Coffeeshop Display App



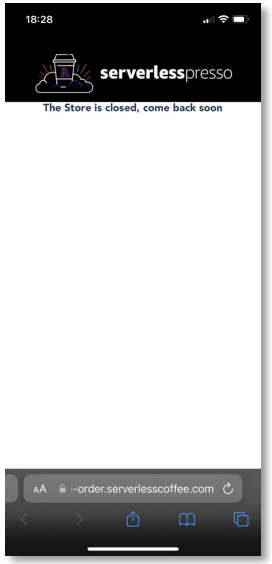
No orders in preparation.
Remaining capacity is 10.

Barista App



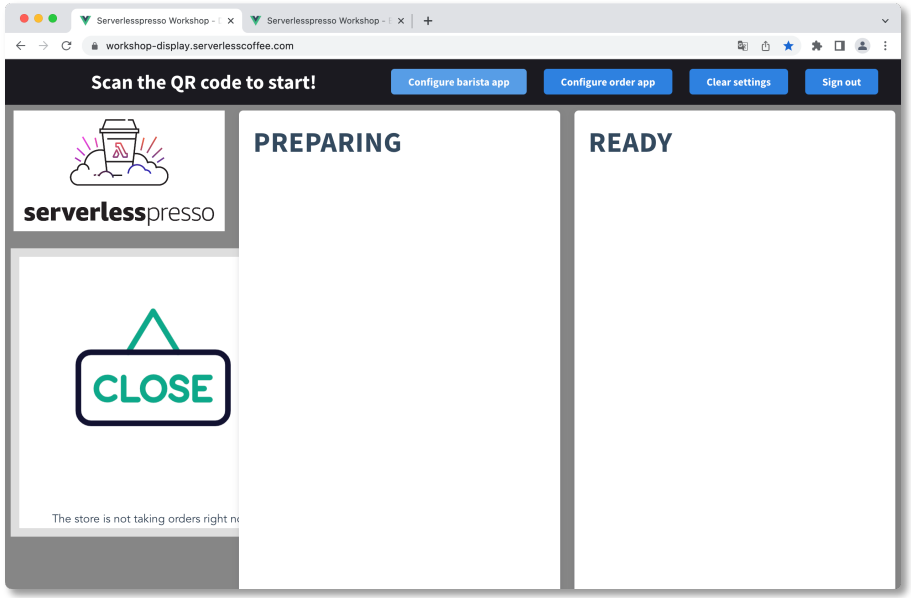
No orders in queue
or in preparation.

Order App



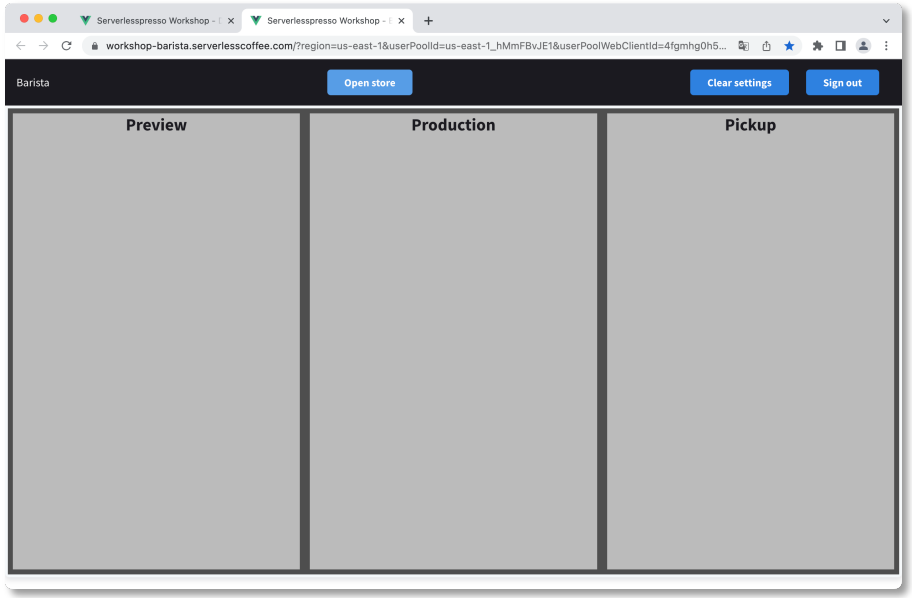
Store **closed**!

Coffeeshop Display App

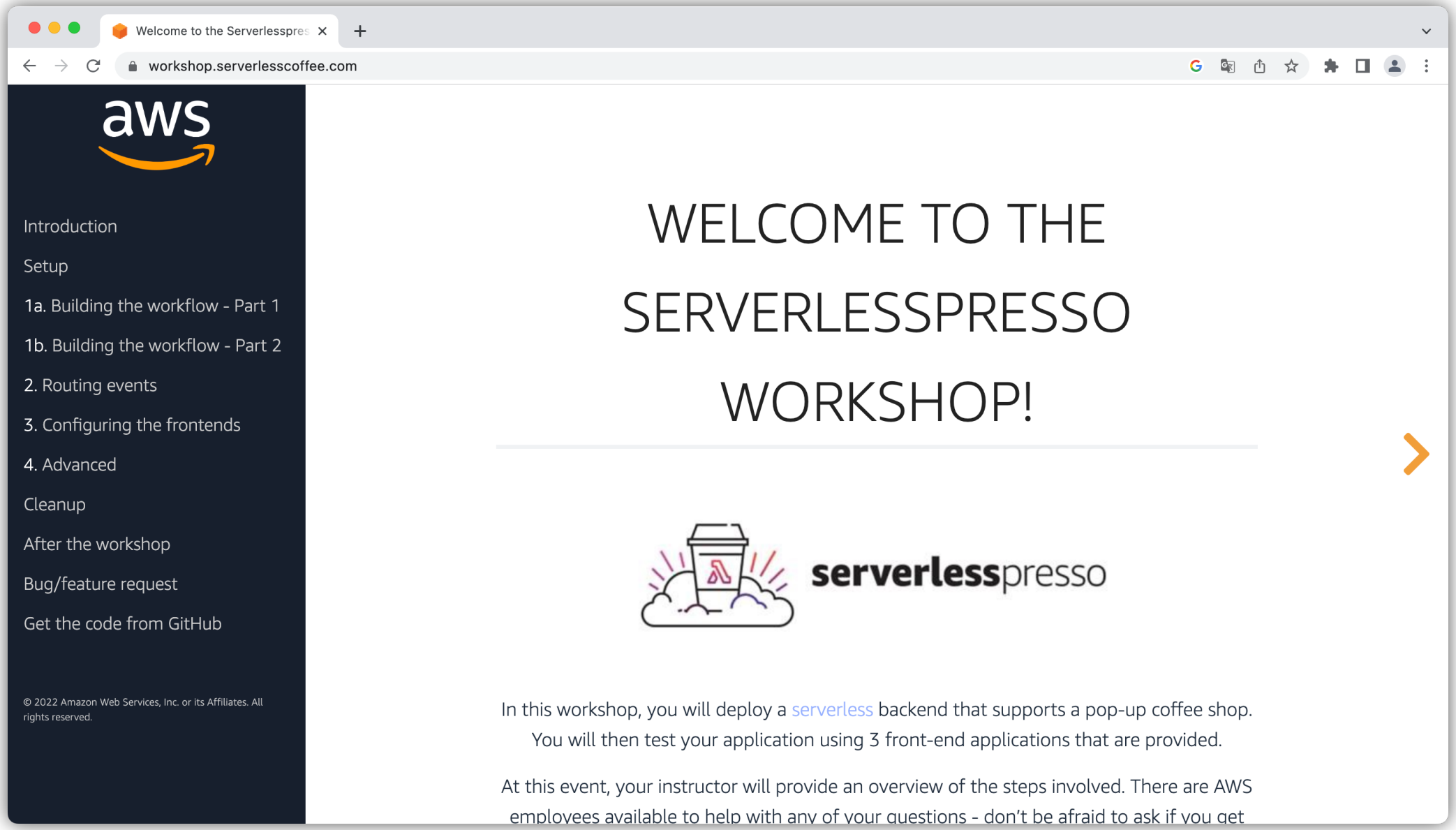


Store is **closed**!

Barista App

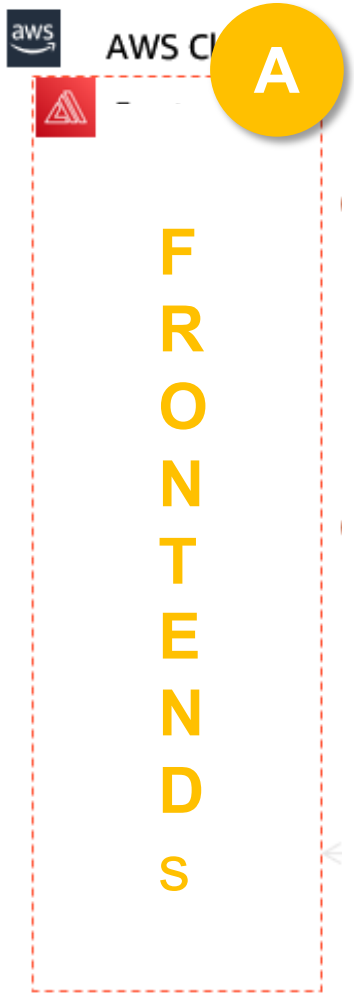


Store is **closed**!



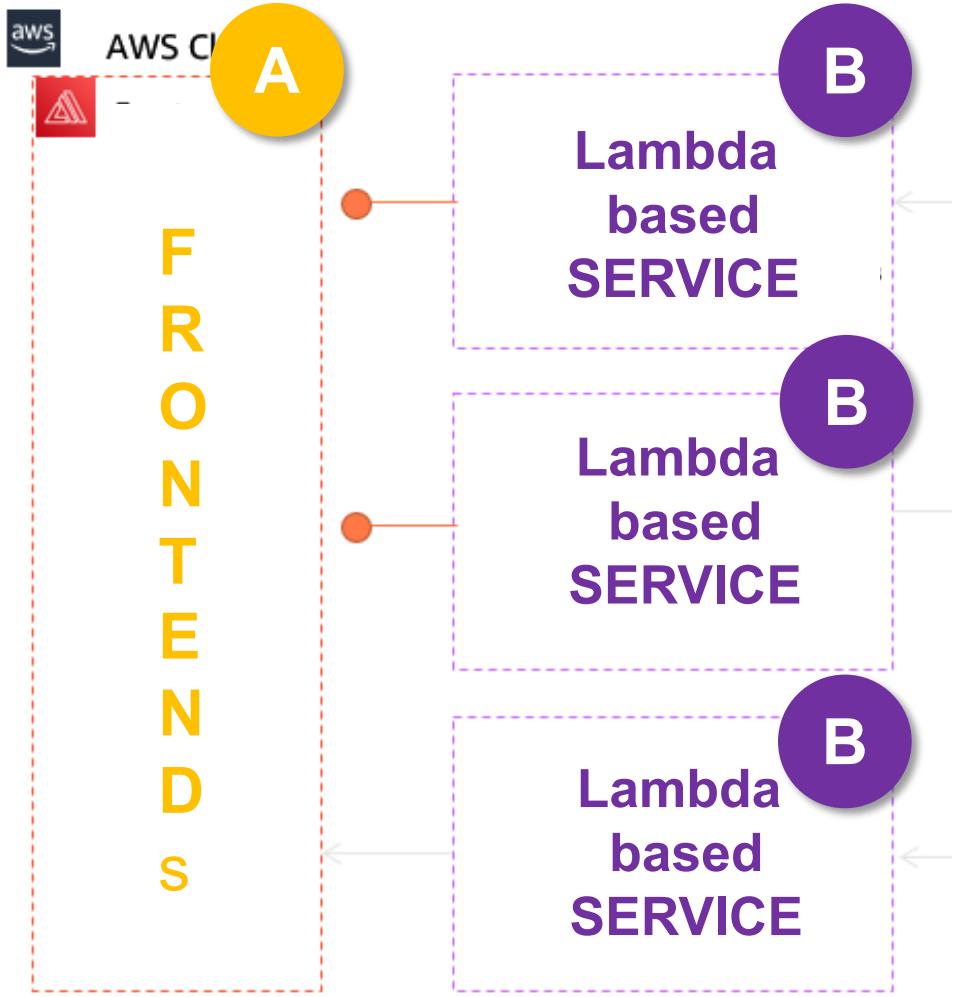
AWS Serverlesspresso Tutorial: https://workshop.serverlesscoffee.com/

Serverless Coffee **Architecture**



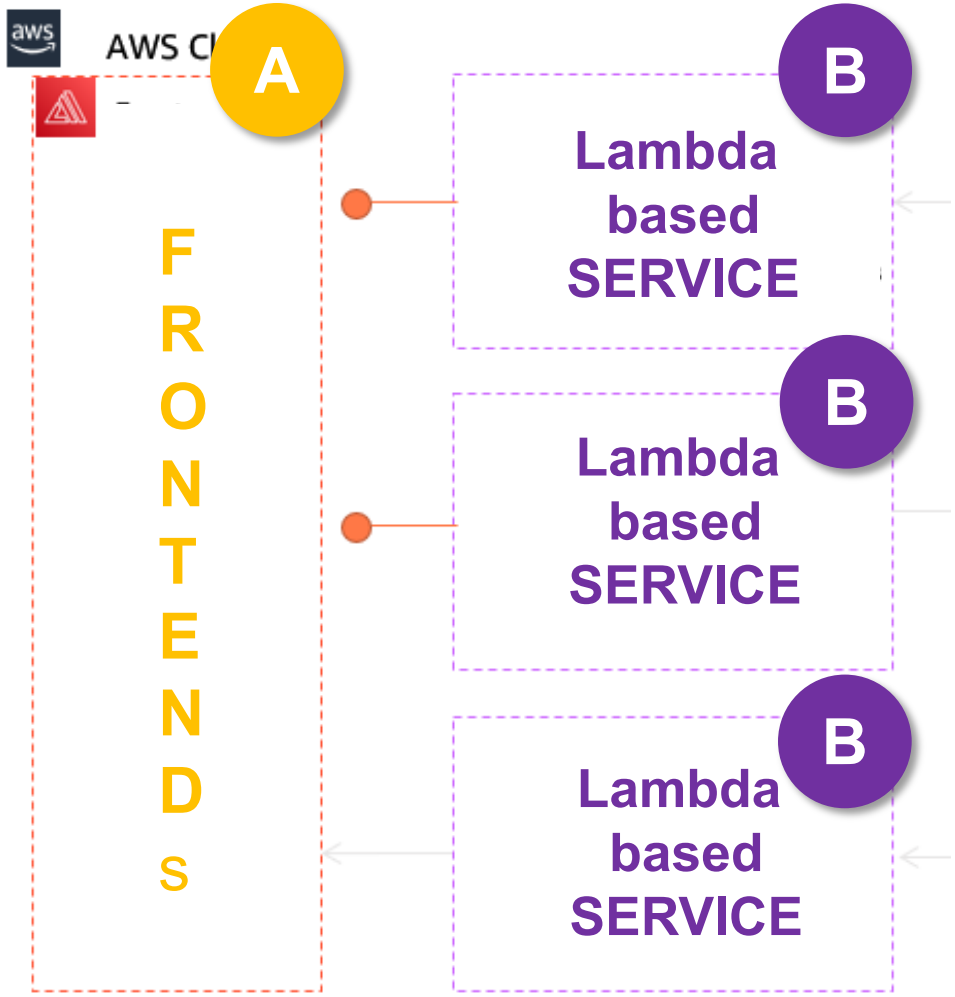
AWS Serverlesspresso Architecture: https://workshop.serverlesscoffee.com/images/se-0-architecture.png

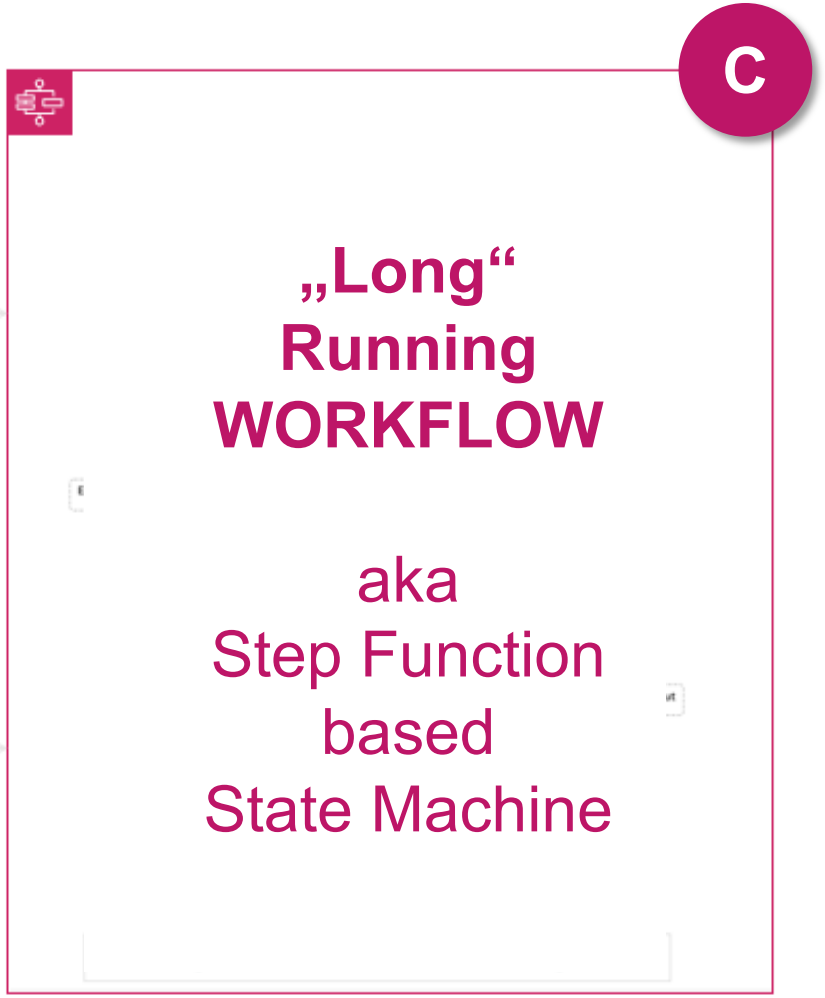
Serverless Coffee **Architecture**



AWS Serverlesspresso Architecture: https://workshop.serverlesscoffee.com/images/se-0-architecture.png

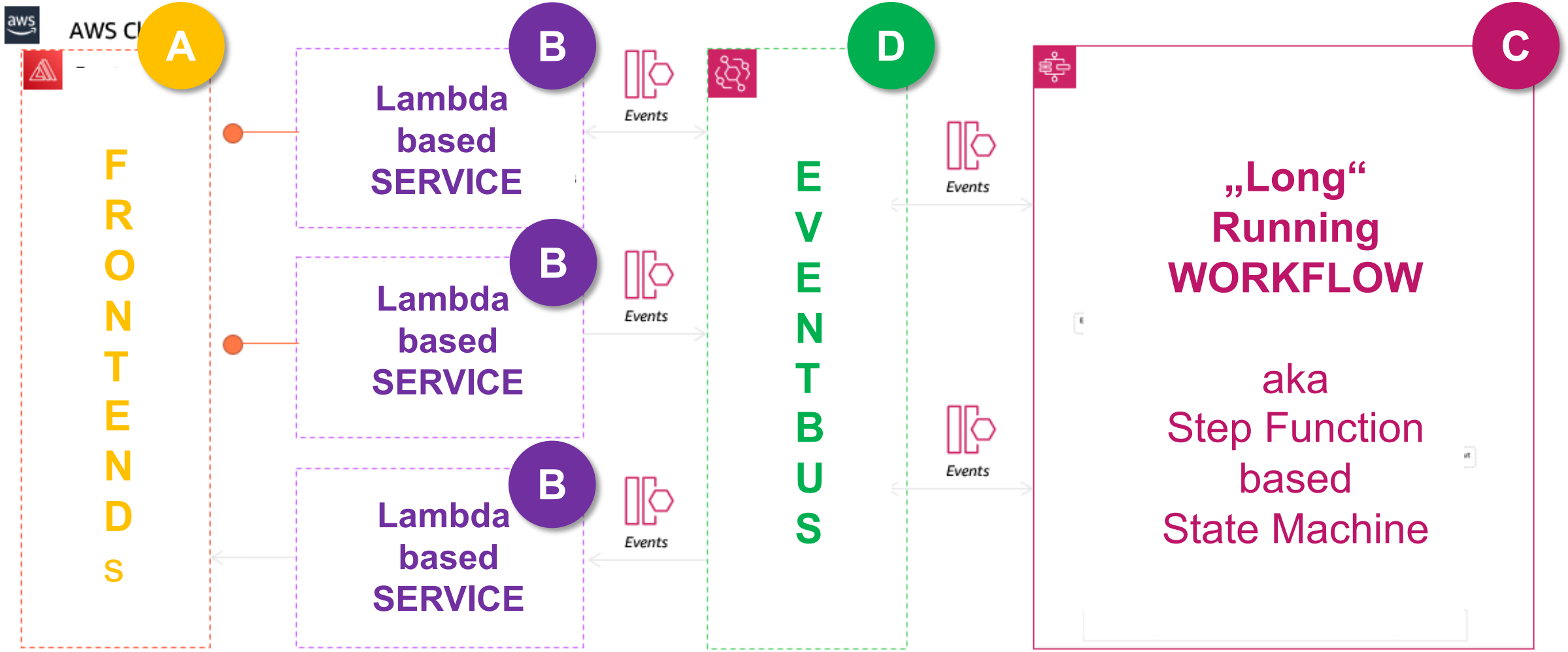
Serverless Coffee **Architecture**





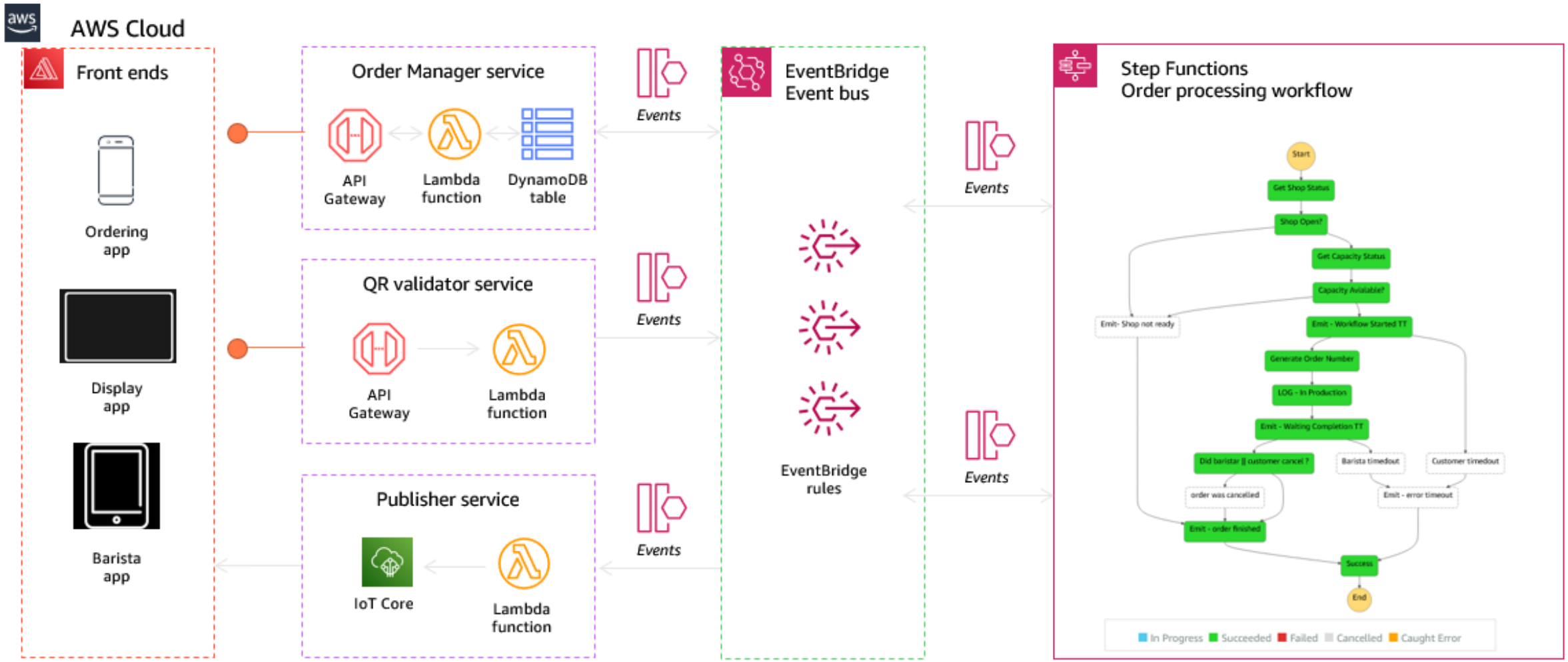
AWS Serverlesspresso Architecture: https://workshop.serverlesscoffee.com/images/se-0-architecture.png

Serverless Coffee **Architecture**



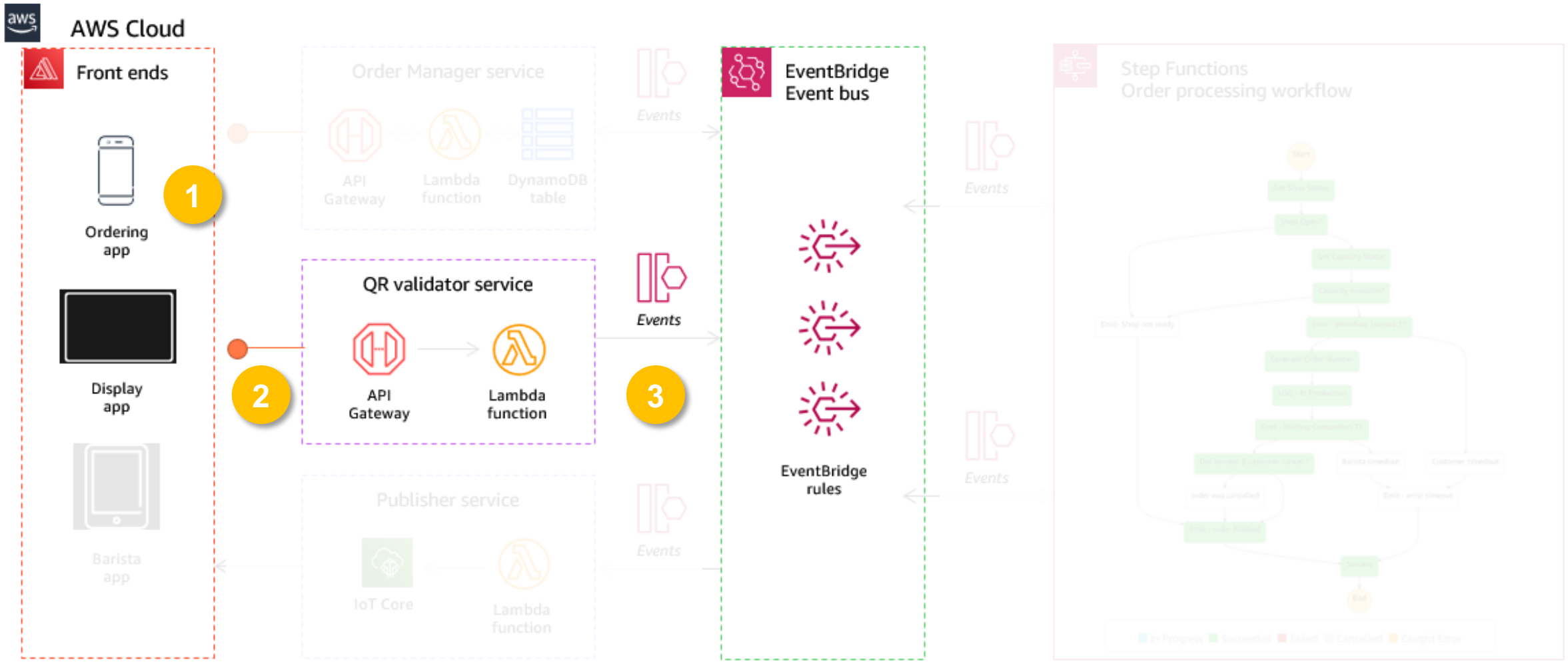
AWS Serverlesspresso Architecture: https://workshop.serverlesscoffee.com/images/se-0-architecture.png

Serverless Coffee **Architecture**

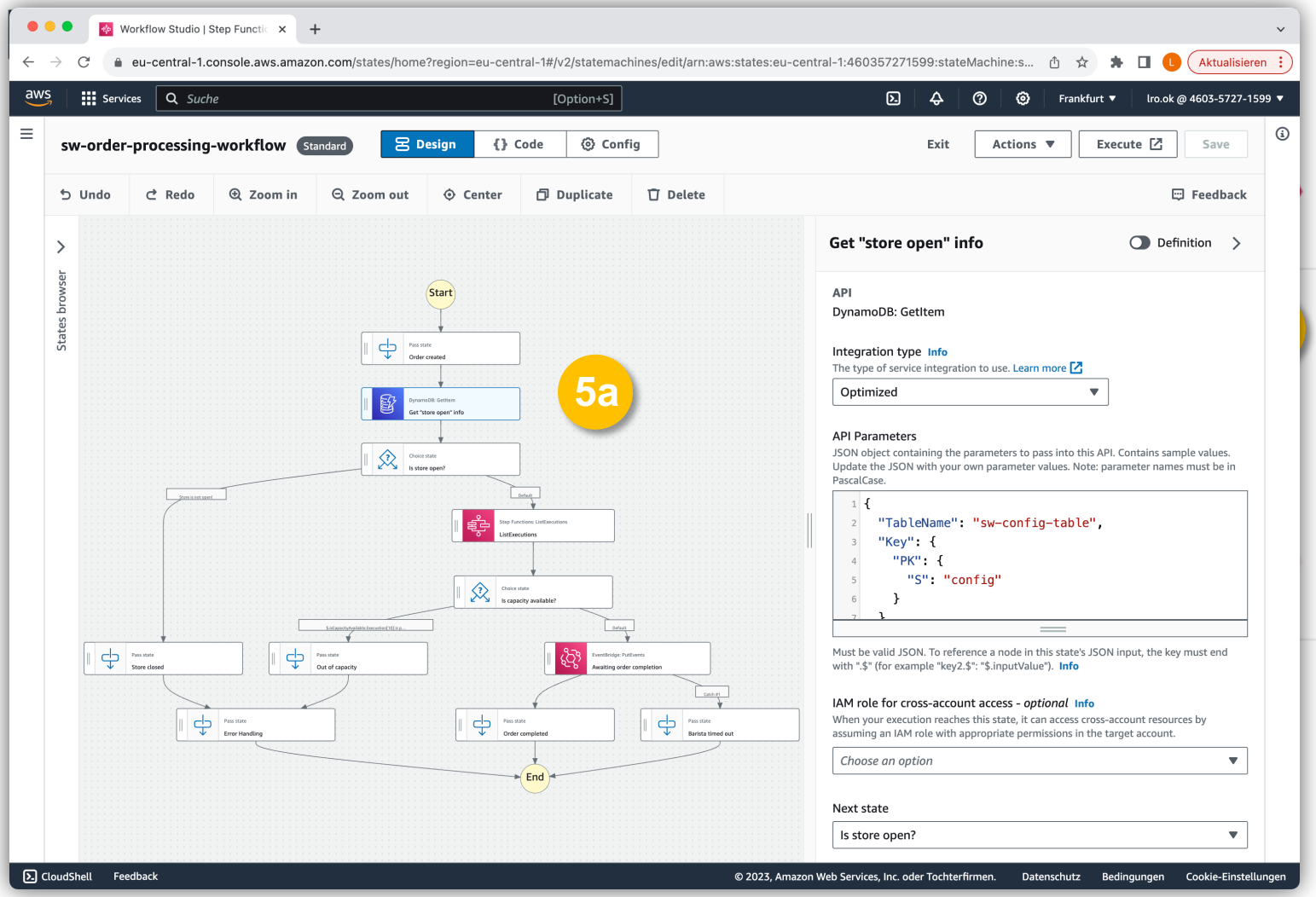


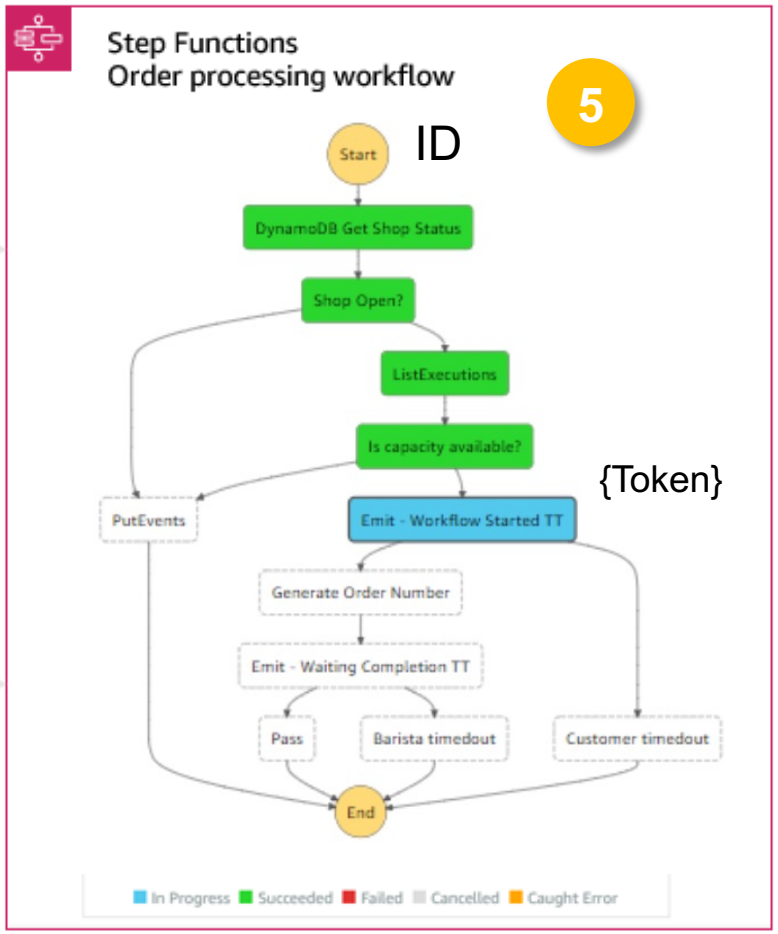
AWS Serverlesspresso Architecture: https://workshop.serverlesscoffee.com/images/se-0-architecture.png

Serverless Coffee „**Code scanned**“

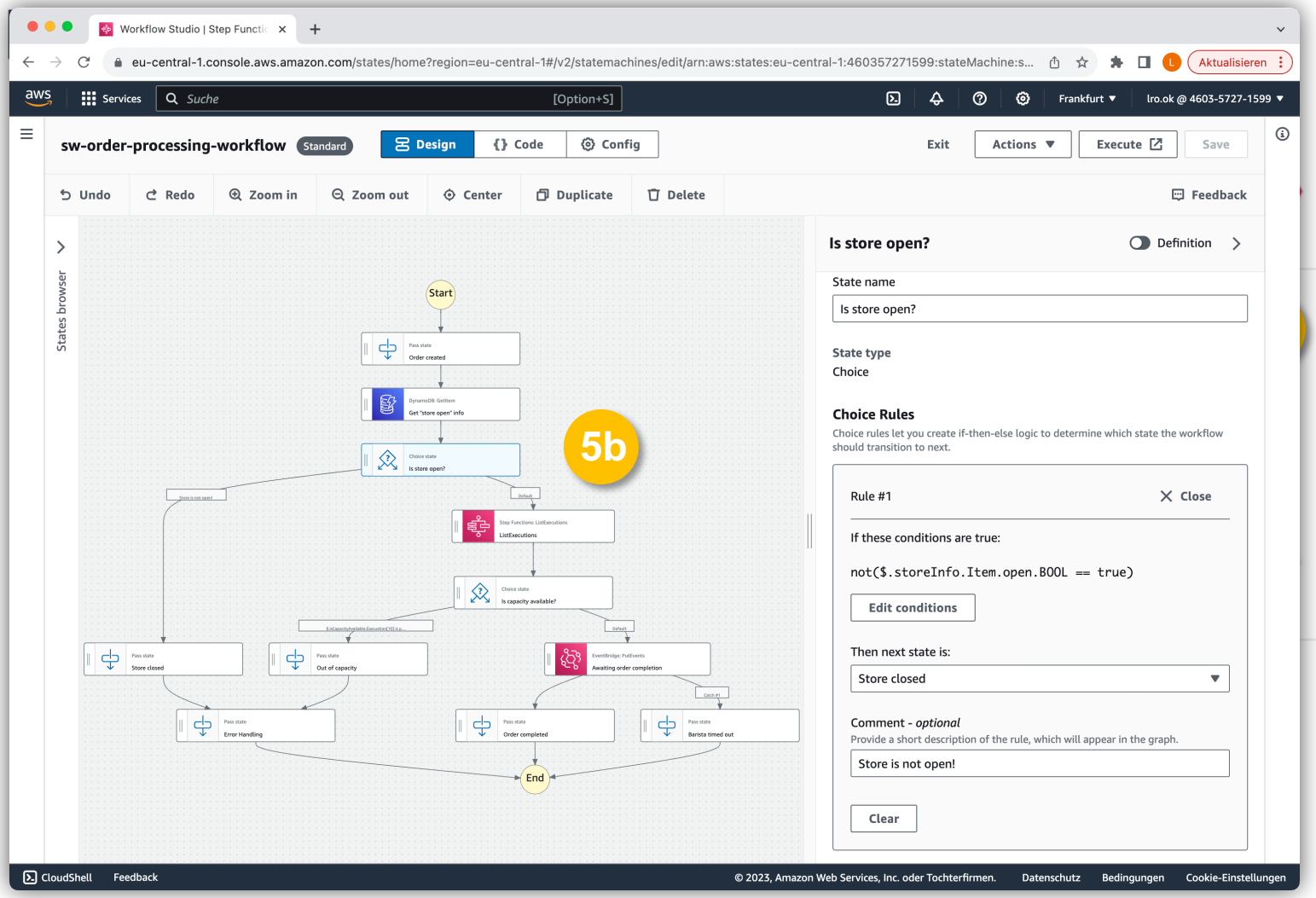


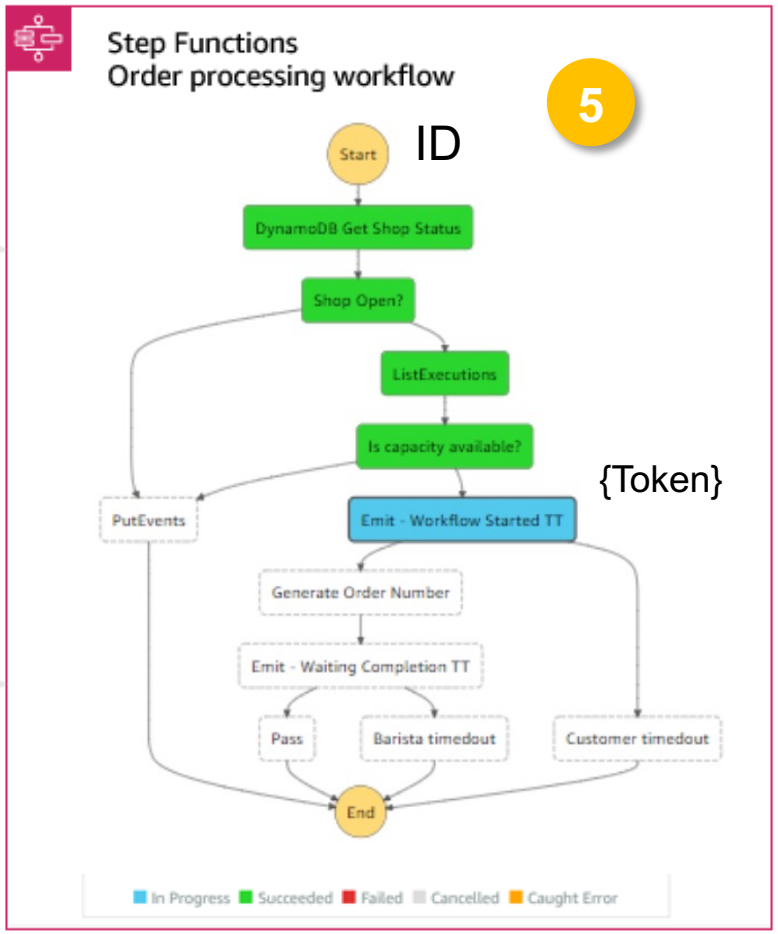
Serverless Coffee „**Code scanned**“



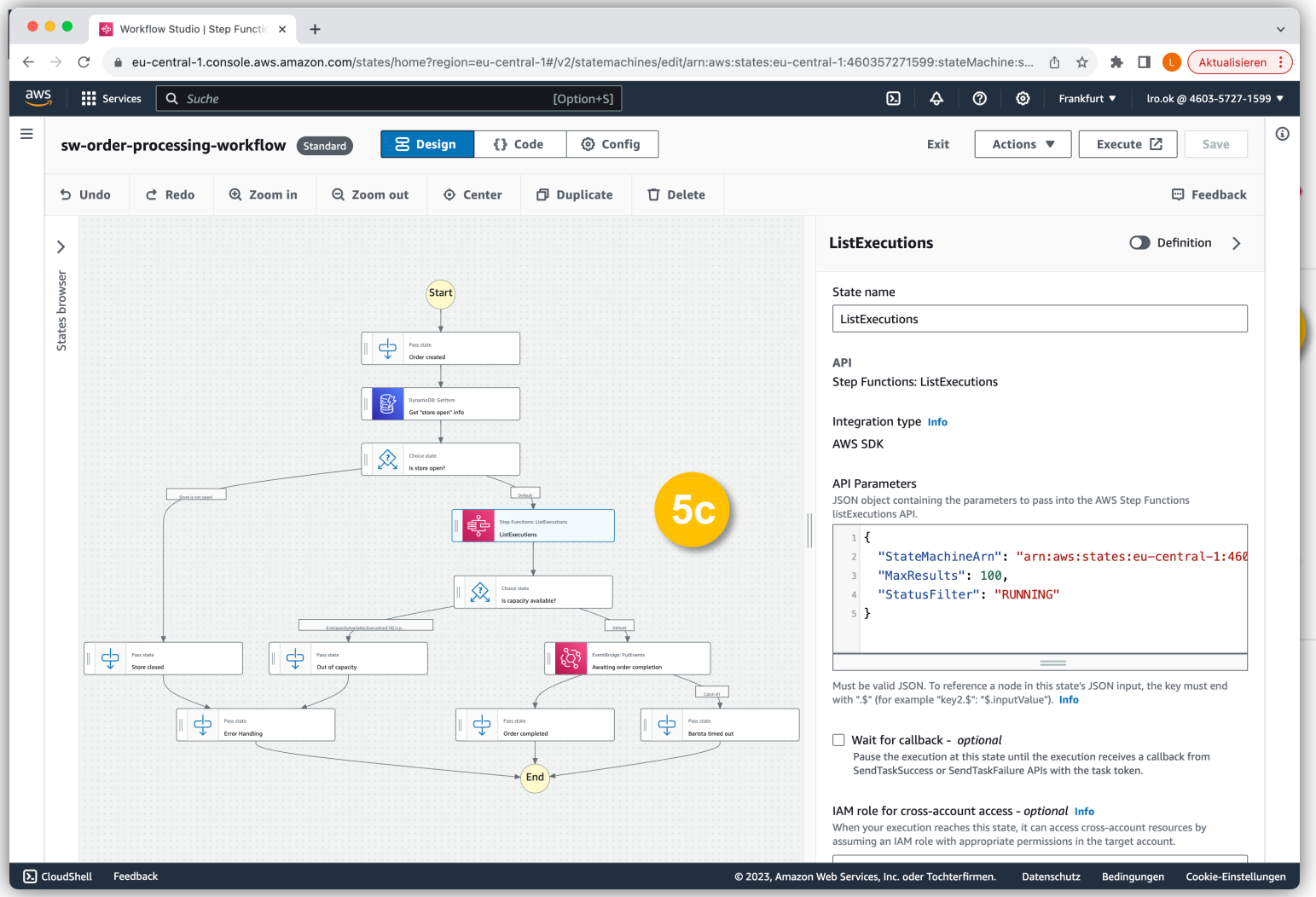


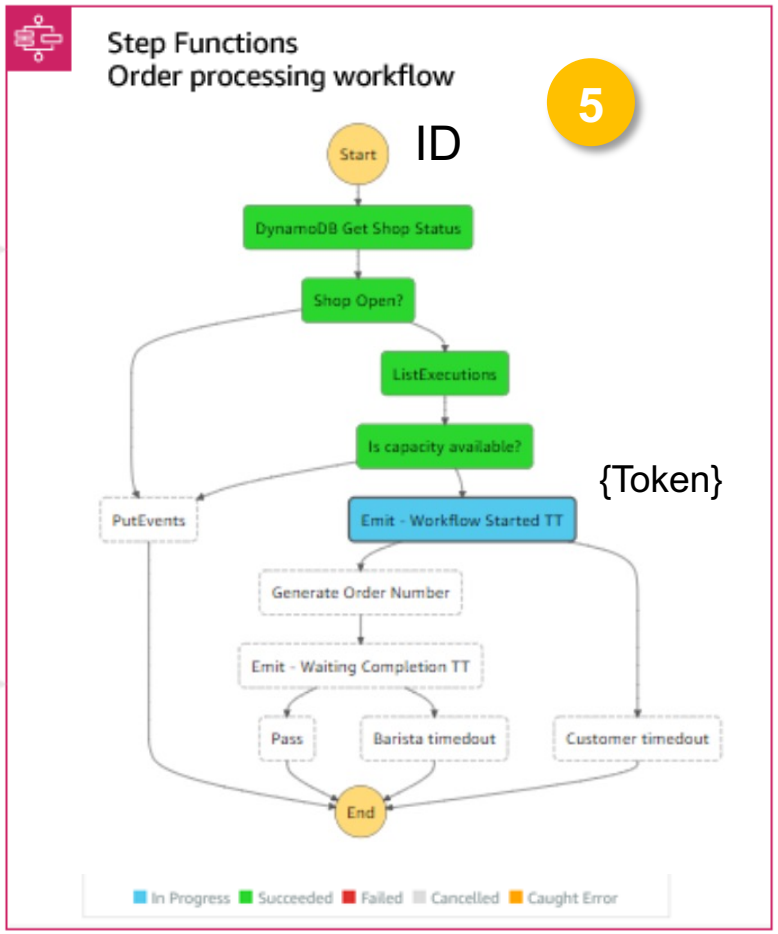
Serverless Coffee „**Code scanned**“



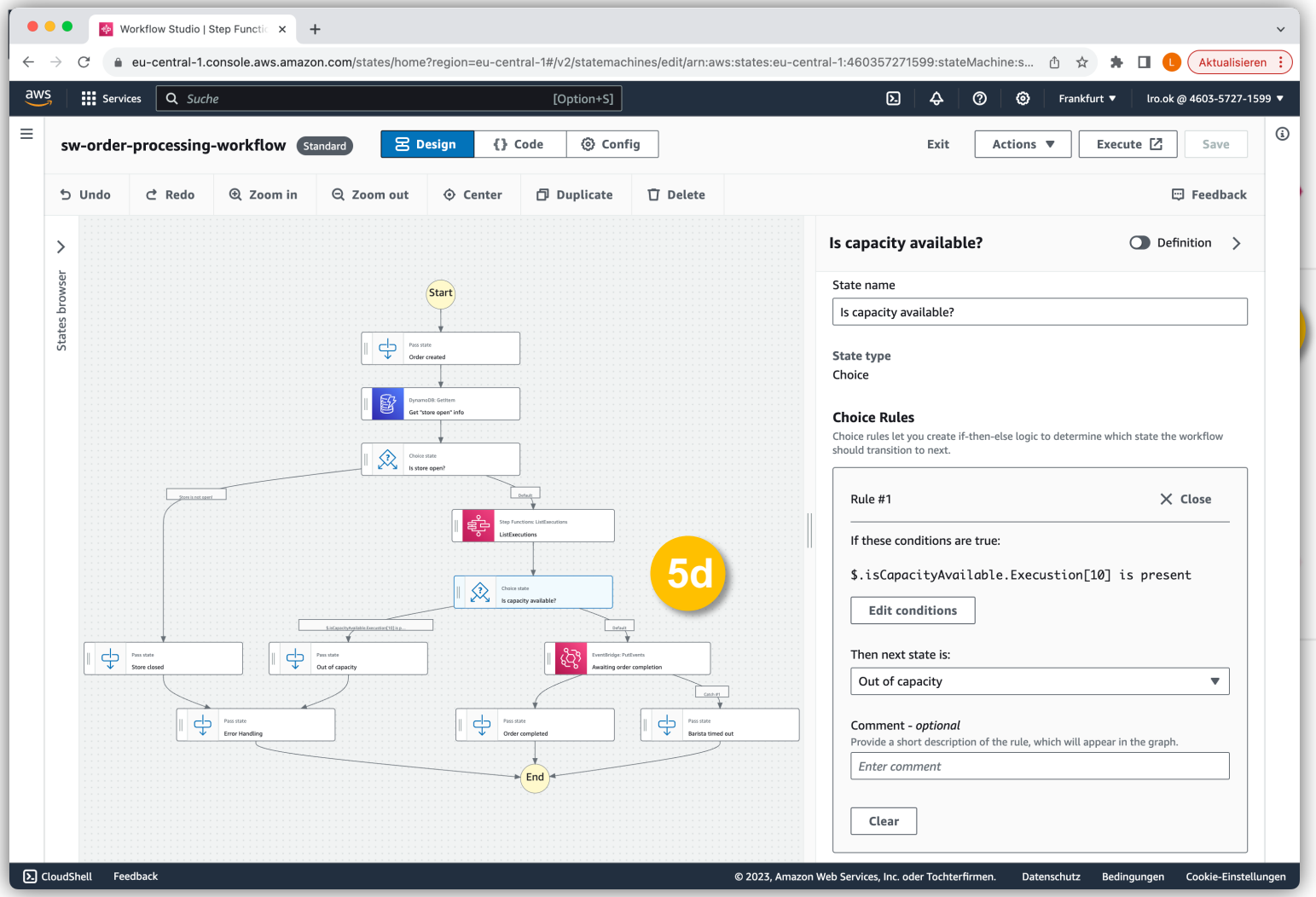


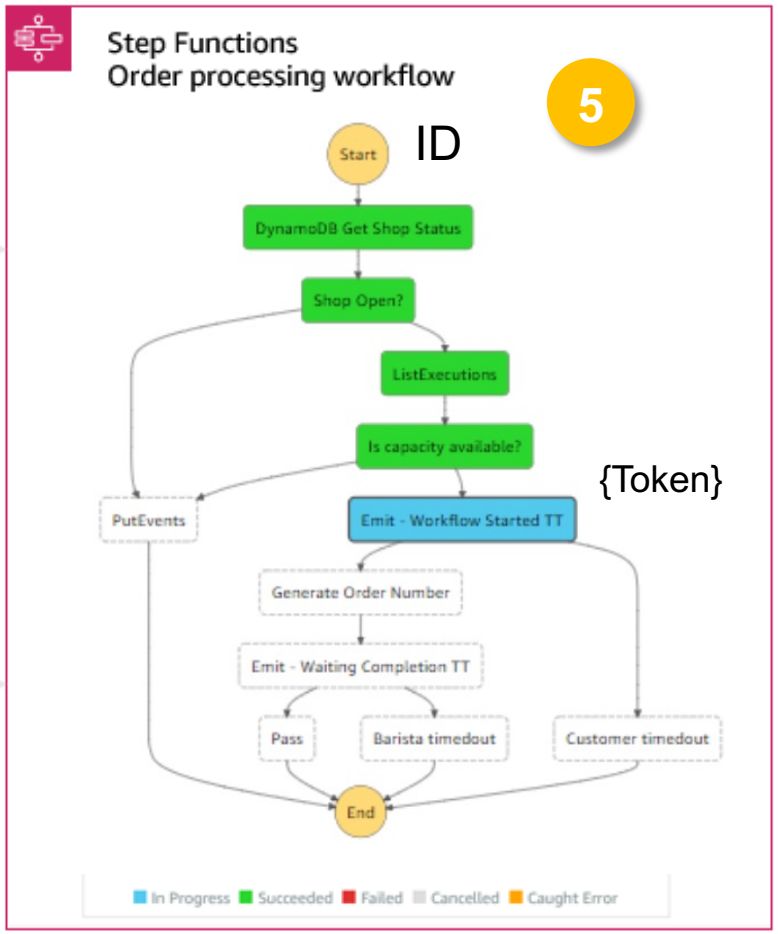
Serverless Coffee „**Code scanned**“



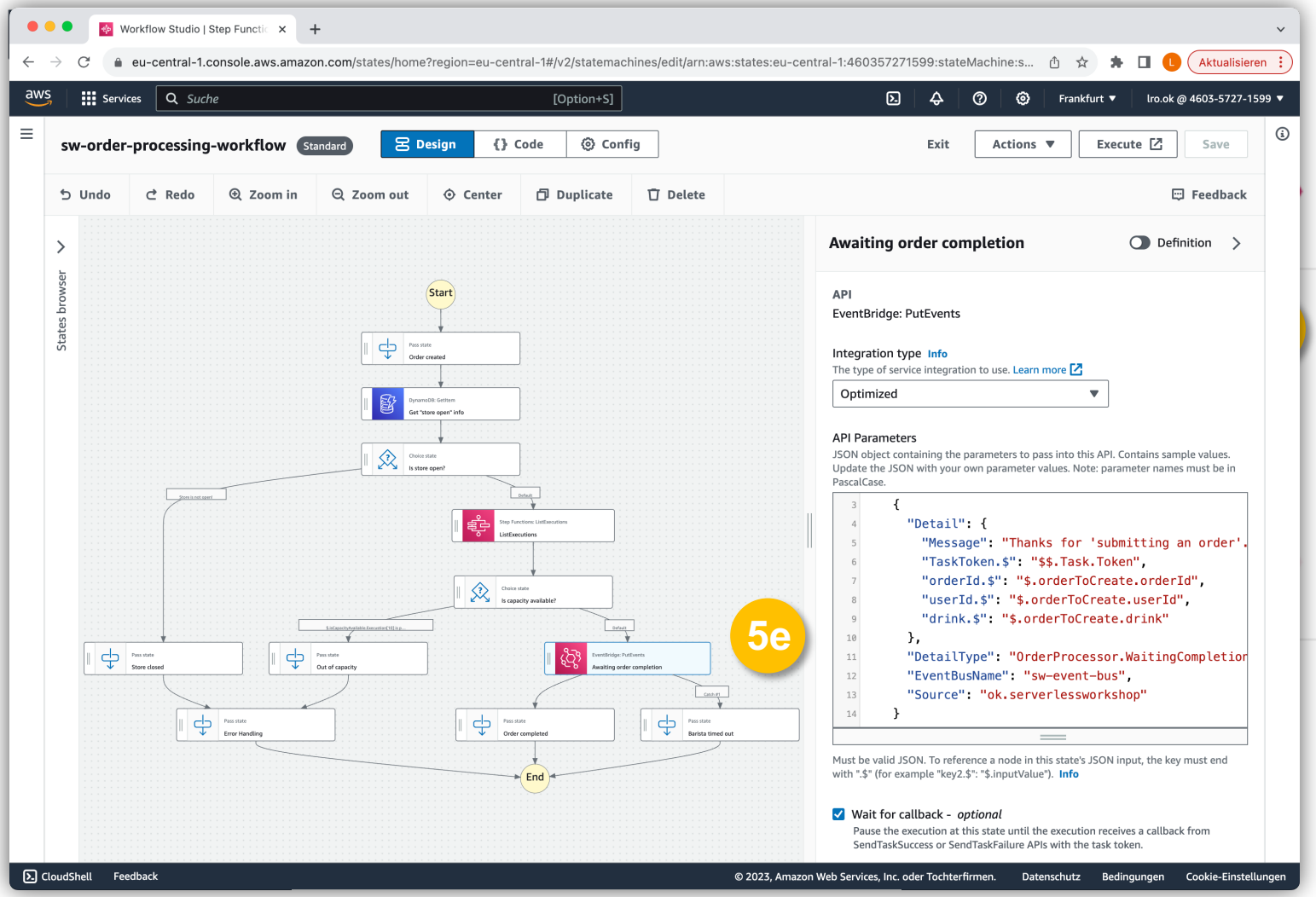


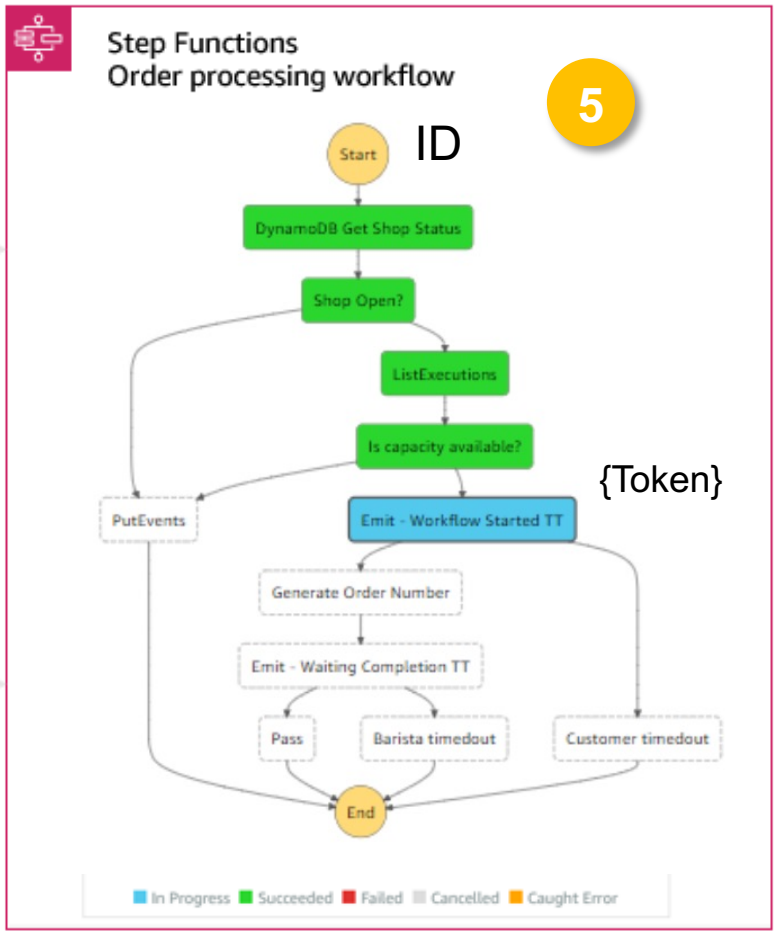
Serverless Coffee „**Code scanned**“



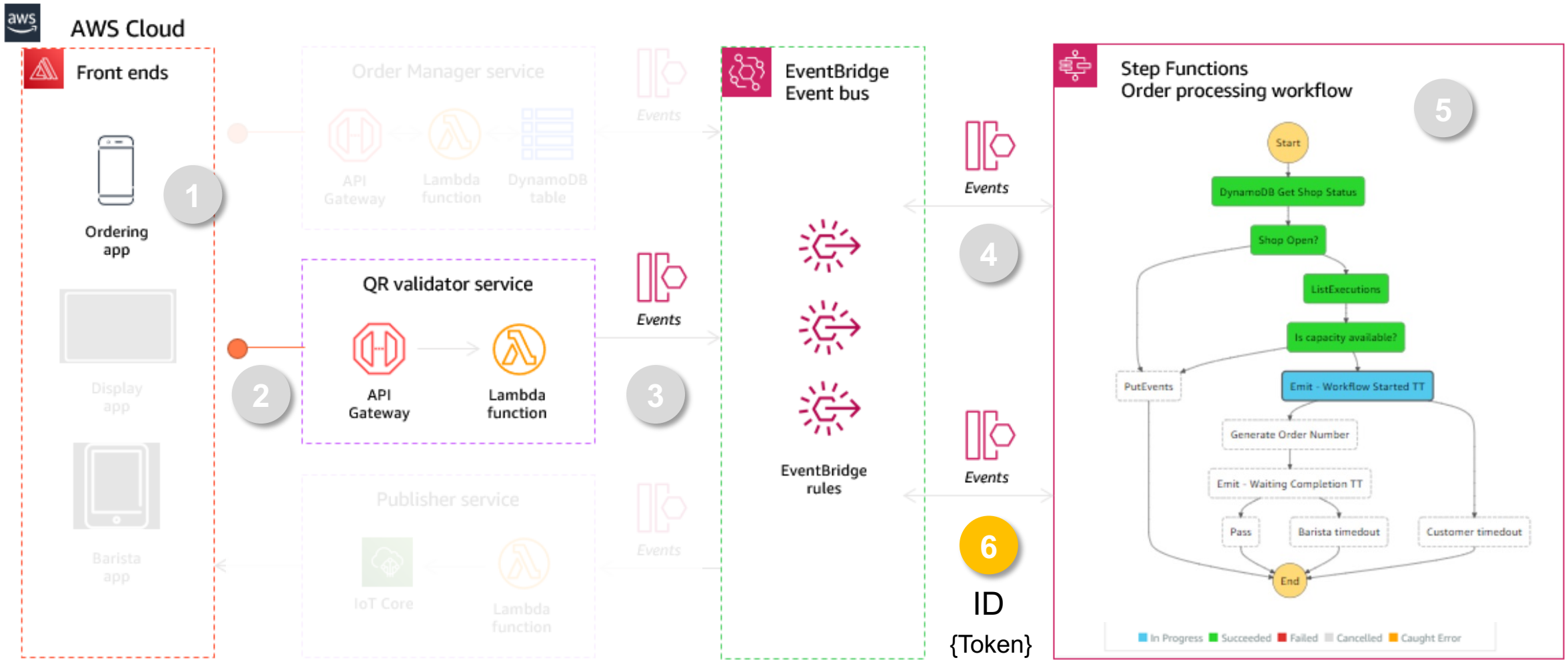


Serverless Coffee „**Code scanned**“

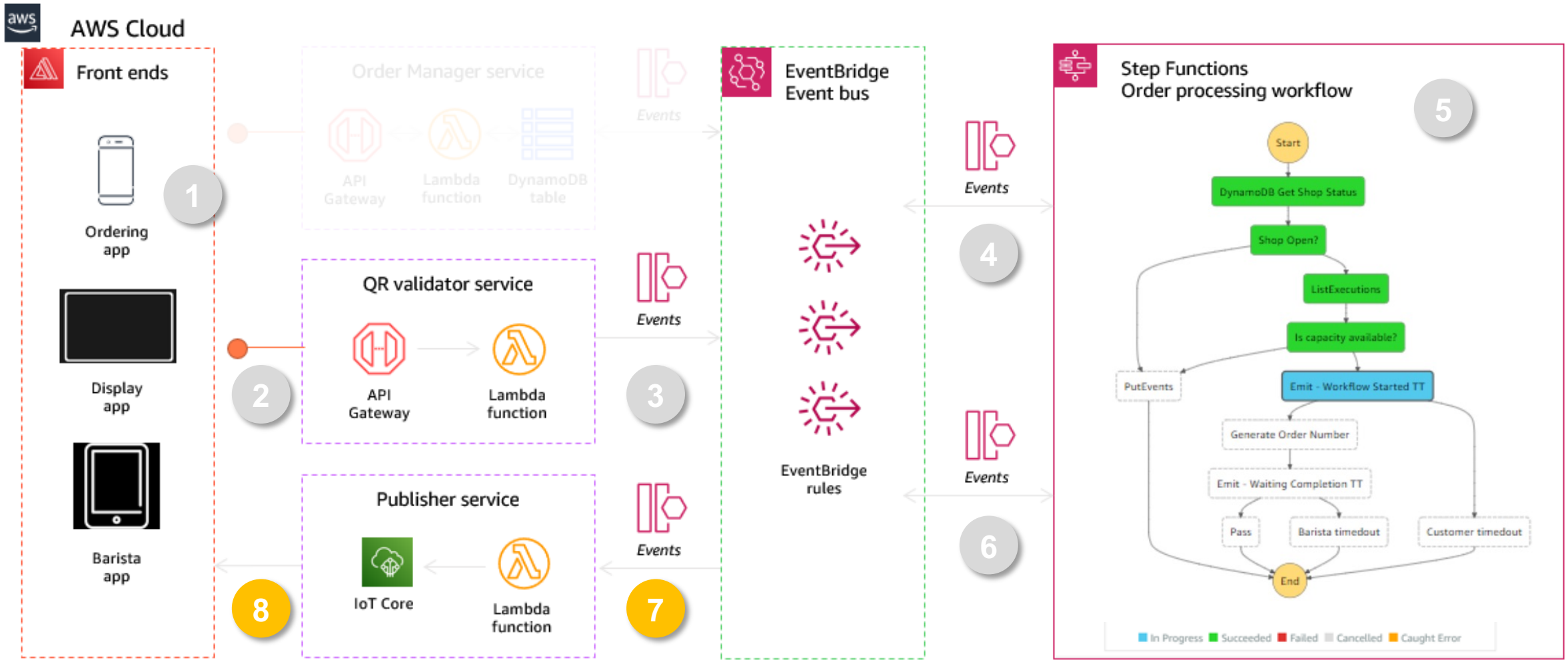




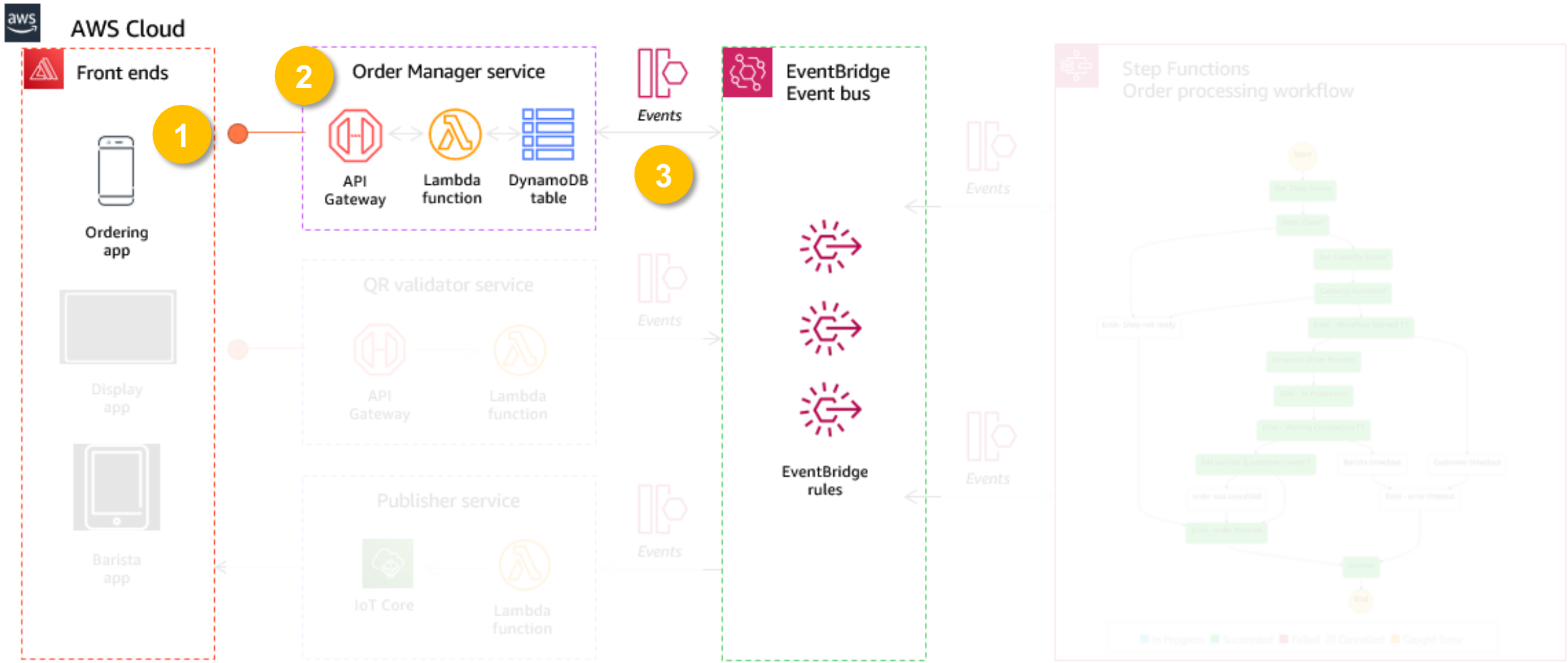
Serverless Coffee „**Code scanned**“



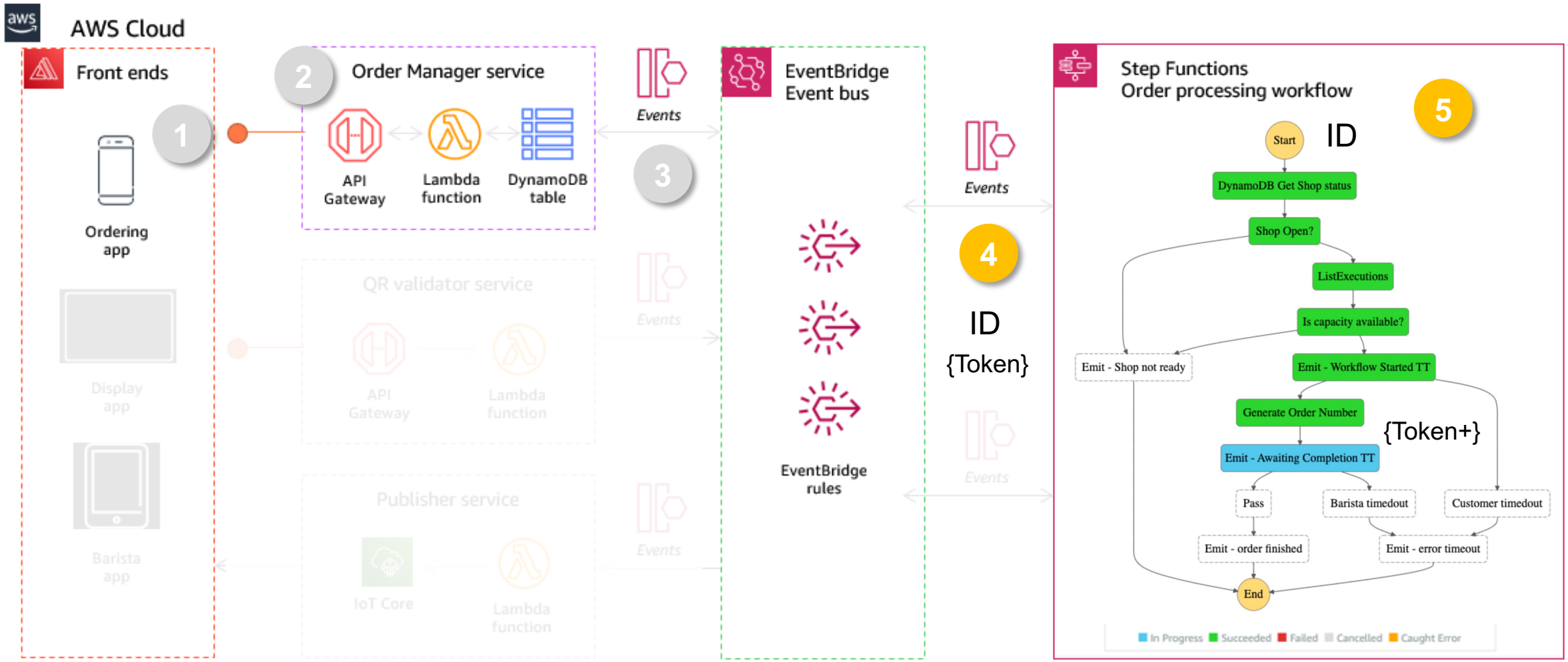
Serverless Coffee „**Code scanned**“



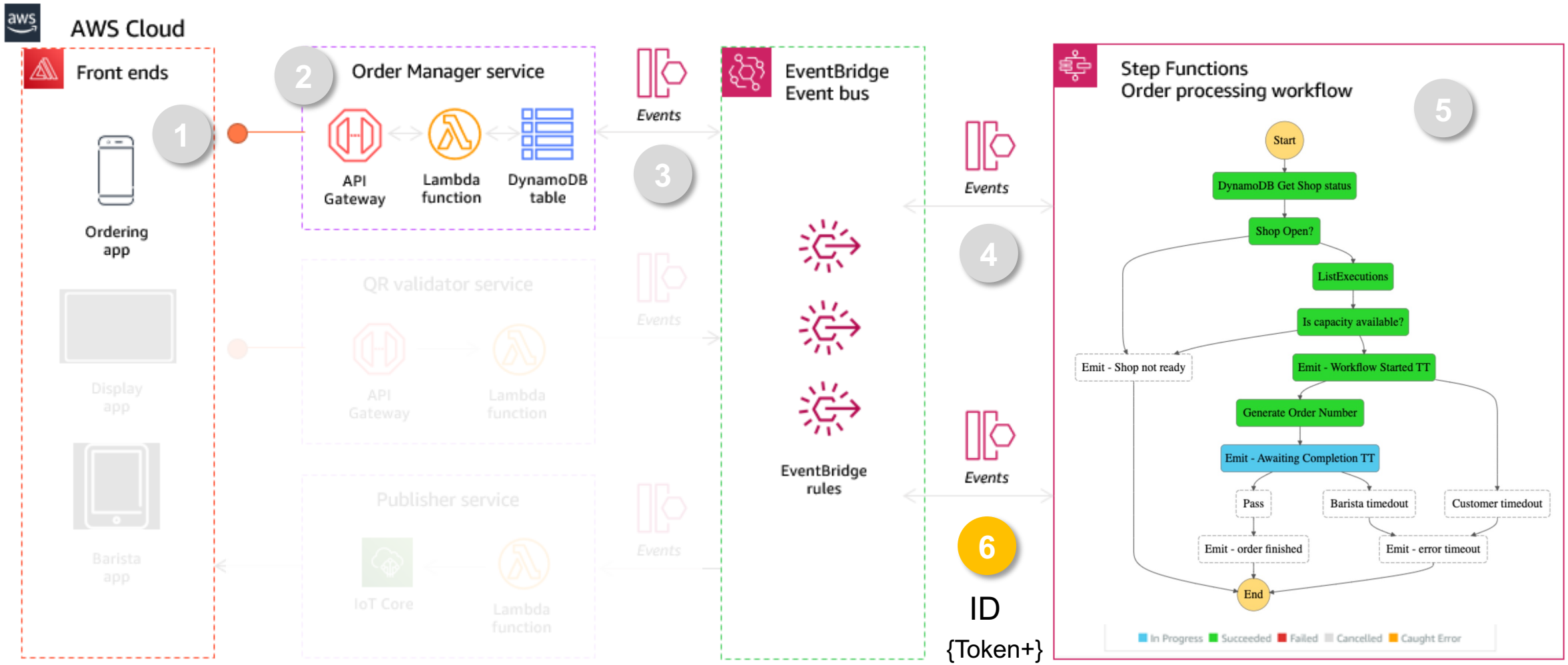
Serverless Coffee „**Place Order**“



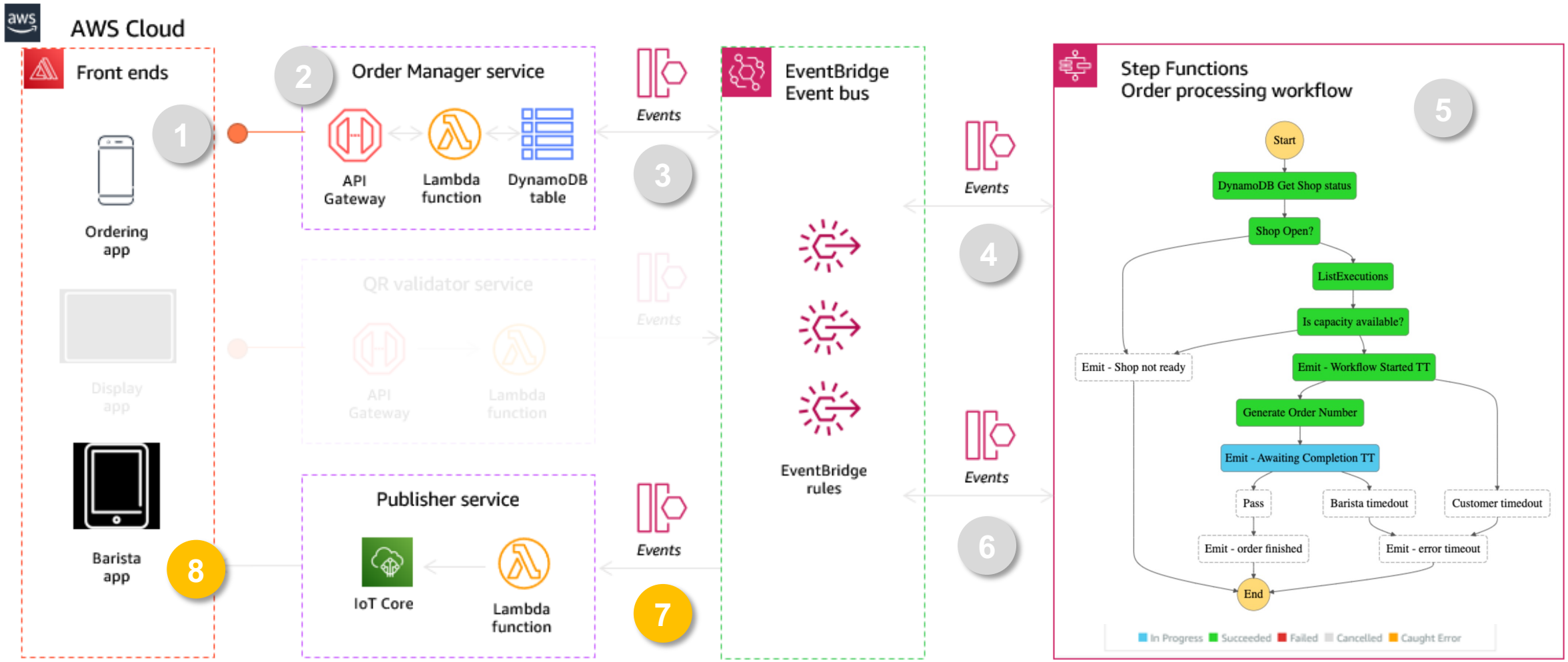
Serverless Coffee „**Place Order**“



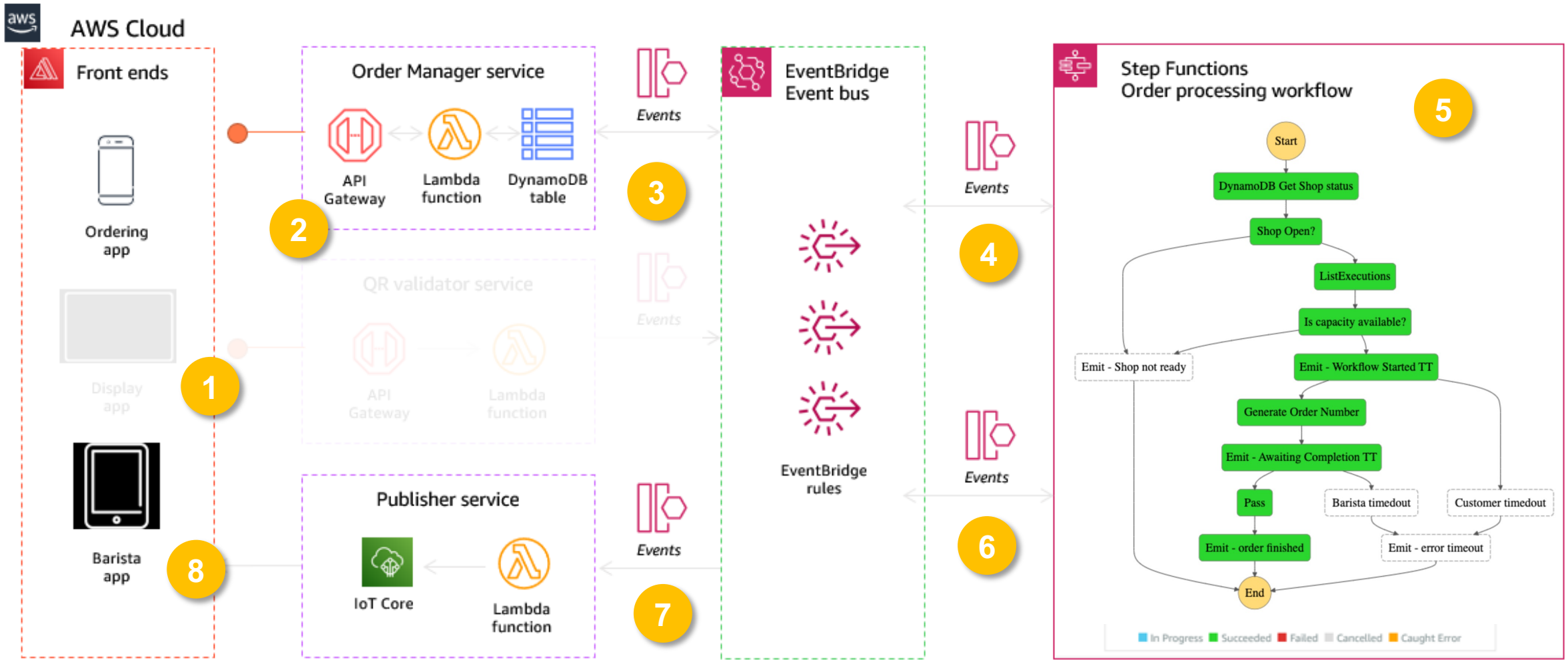
Serverless Coffee „**Place Order**“

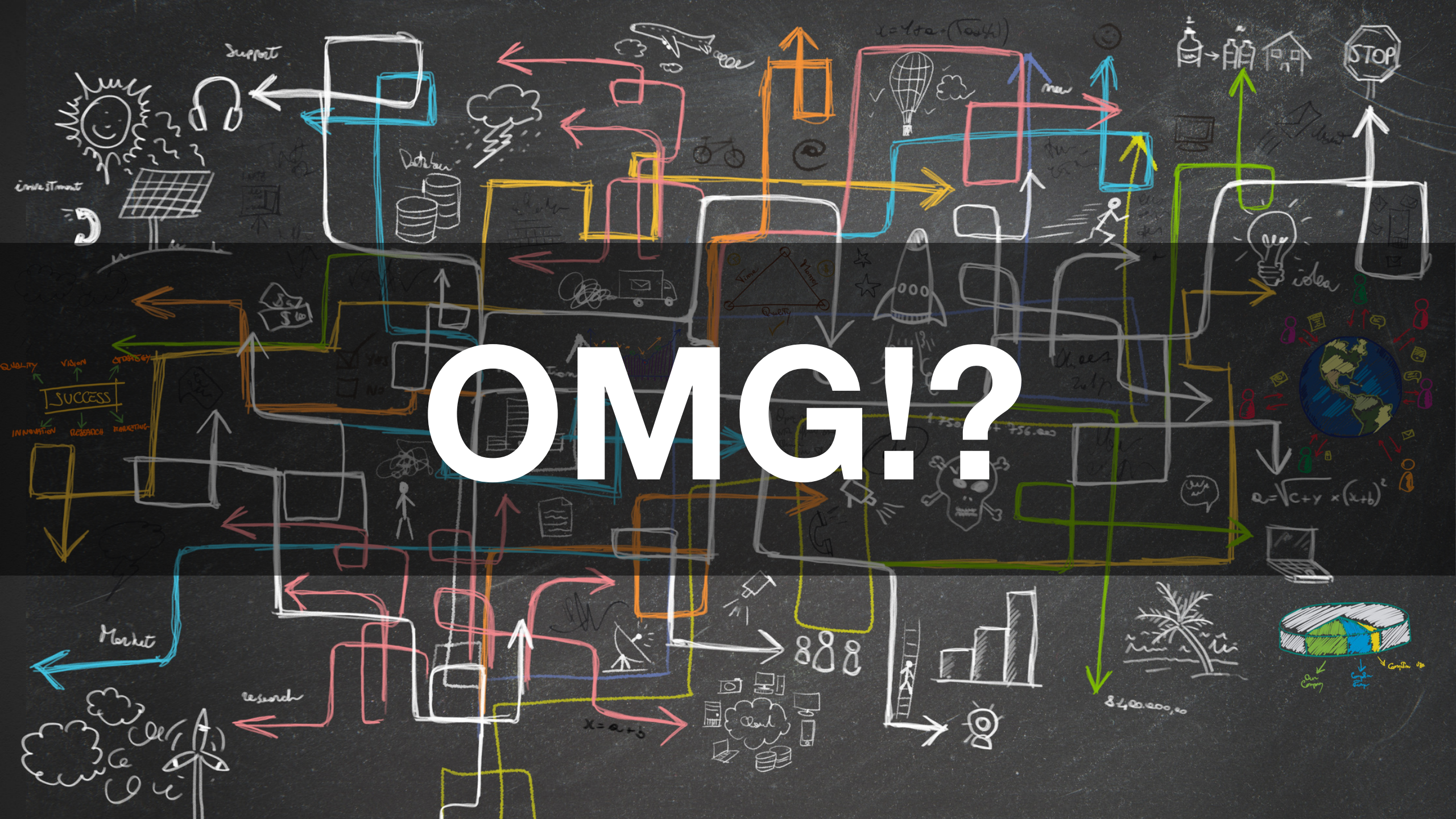


Serverless Coffee „**Place Order**“



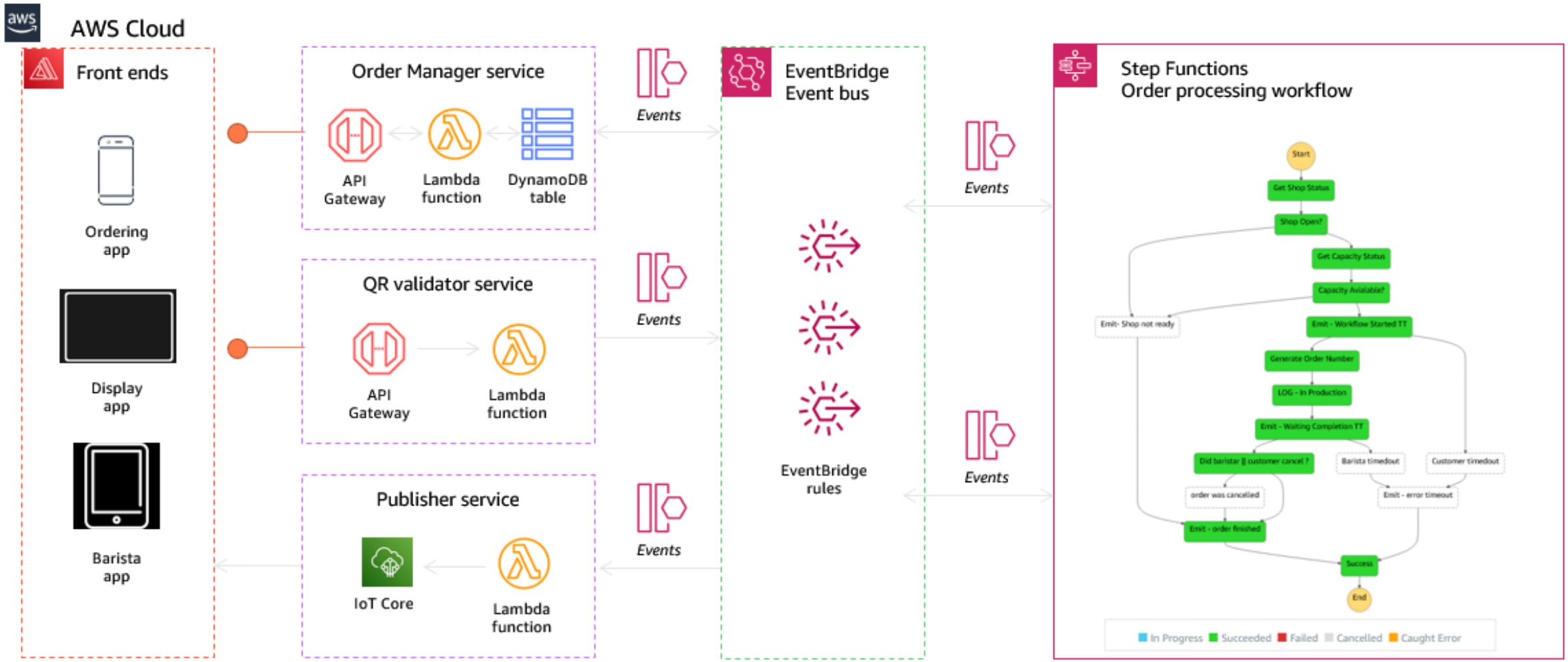
Serverless Coffee „**Coffee in Making**“



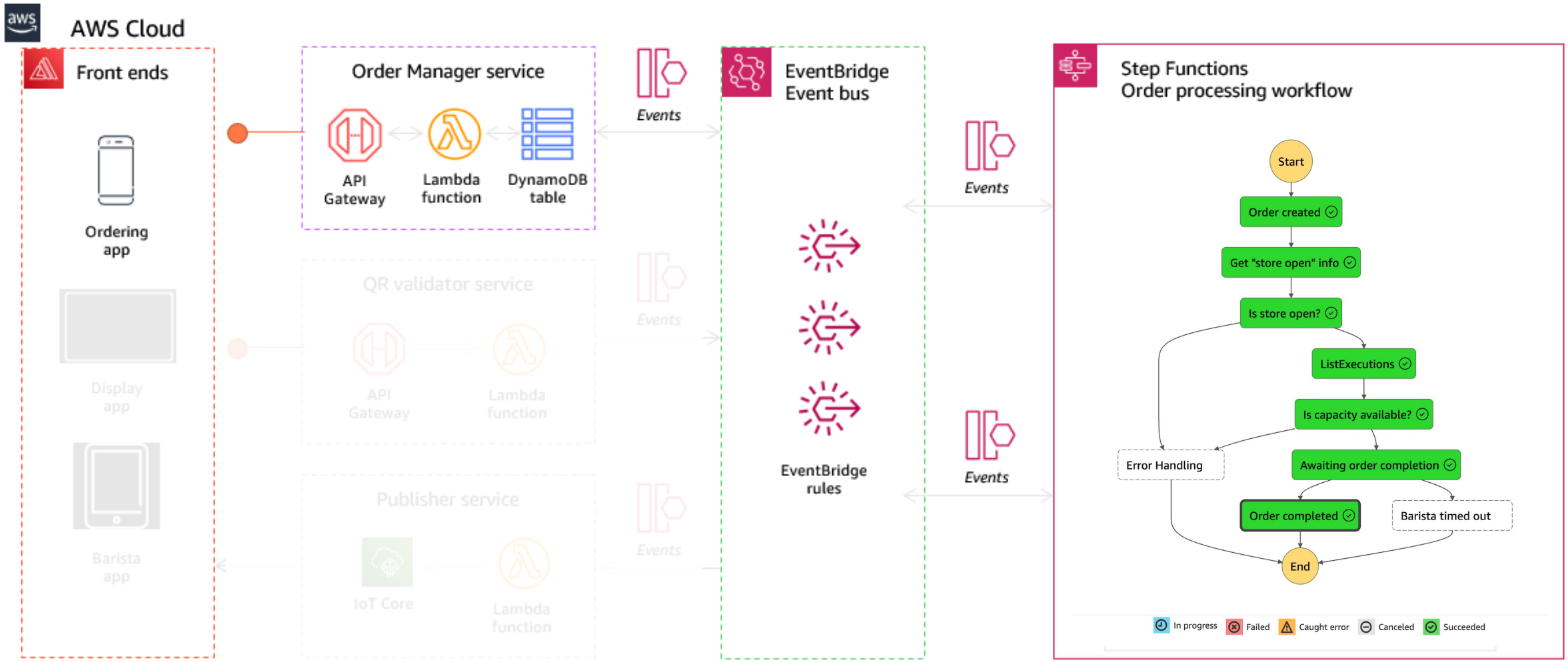

OMG!?

„Aber eigentlich möchte ich doch nur einen Kaffee trinken.“

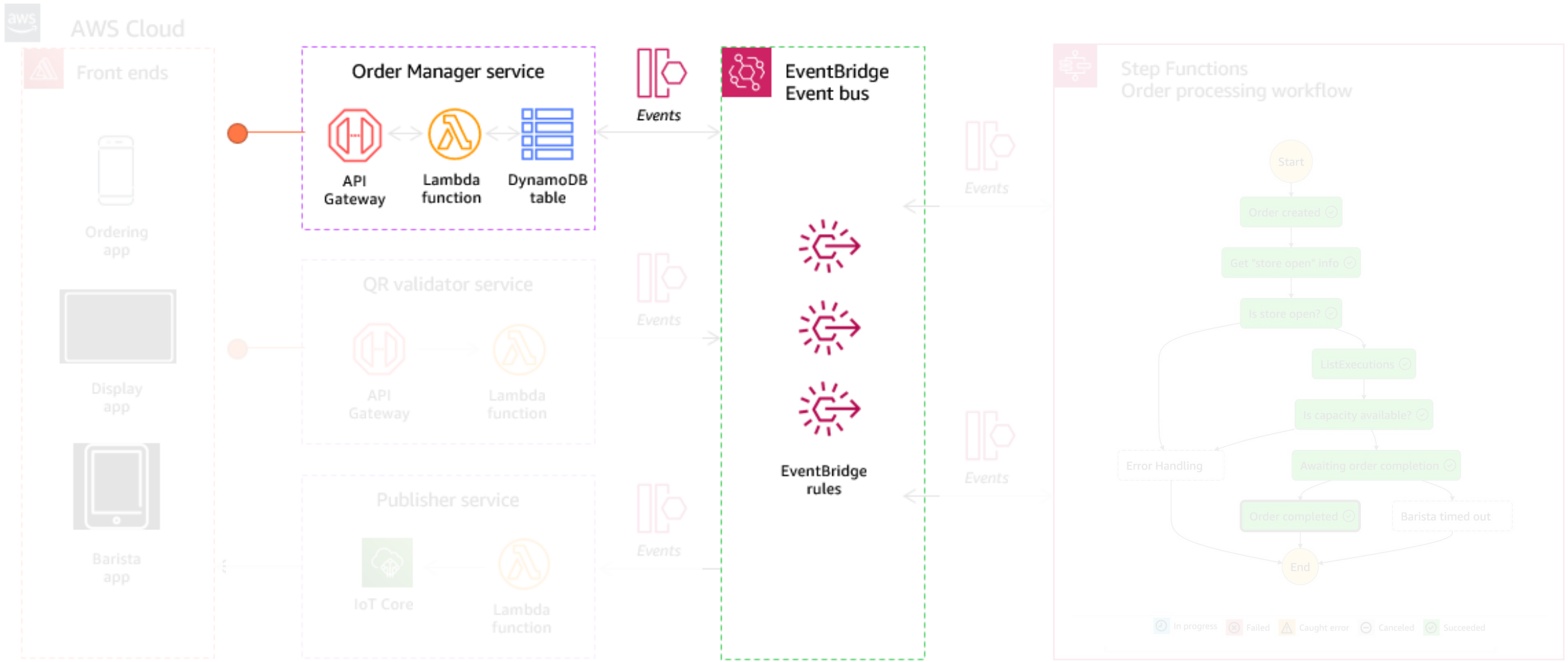
Serverless Coffee „**Full Edition**“



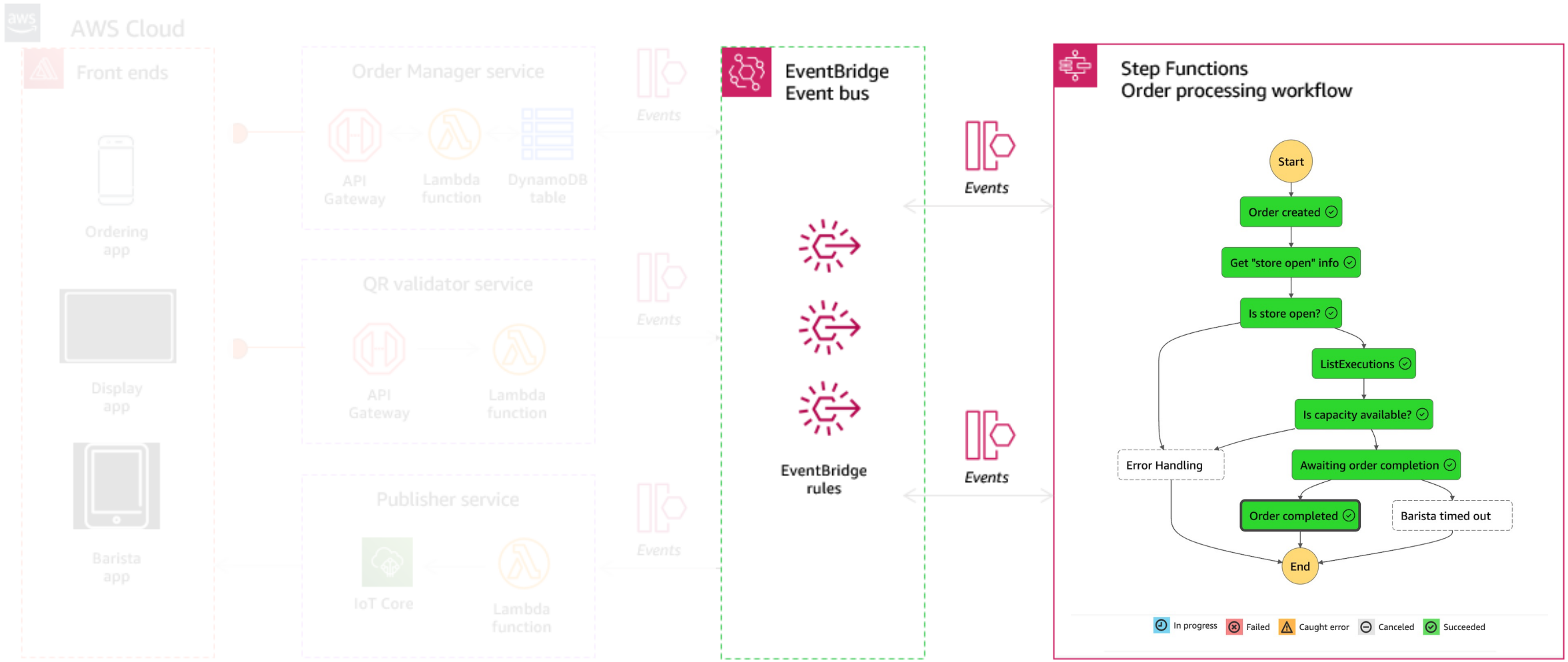
Serverless Coffee „**Light Edition**“



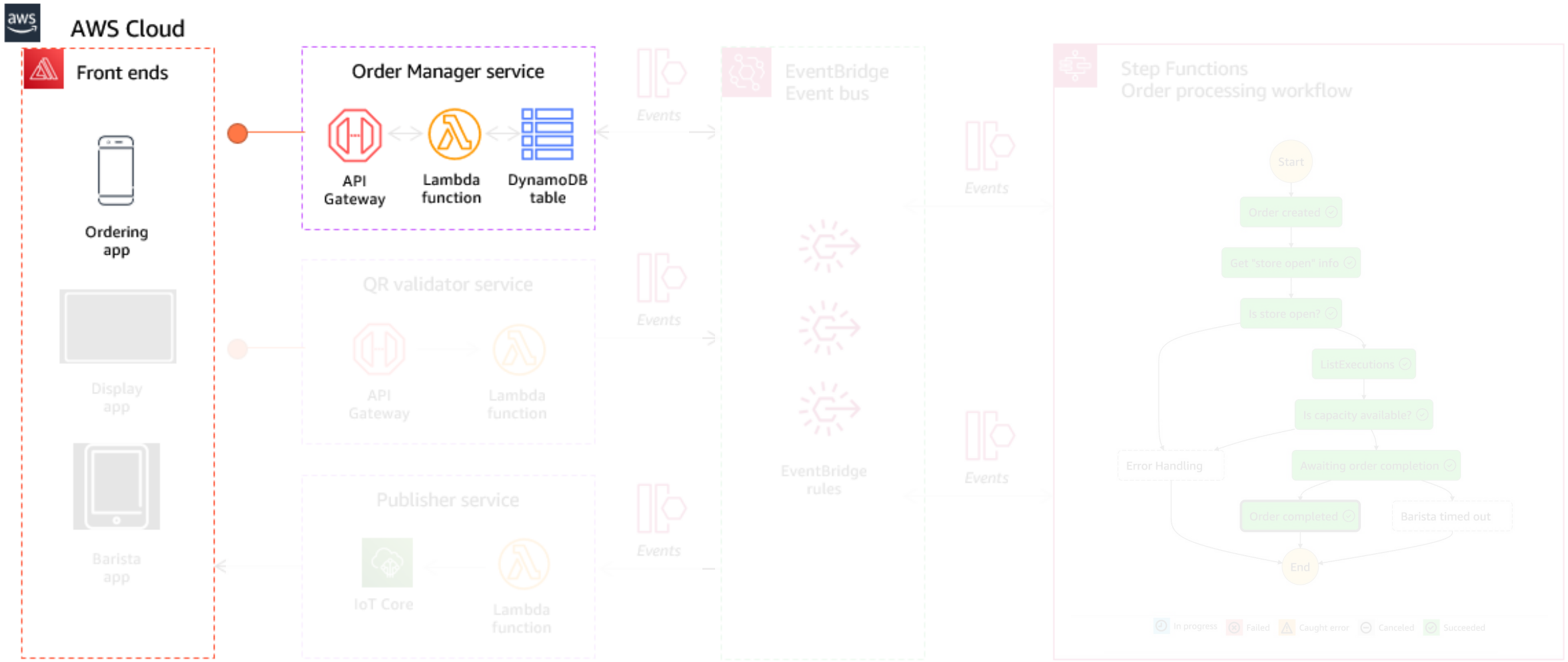
Serverless Coffee „**Order Management**“



Serverless Coffee „**Order Processing**“



Serverless Coffee „**API Gateway** “



Agenda

Hello World & Set-up

Order Management

Order Processing

API Gateway

Testing

OPEN KNOWLEDGE

Serverless Coffee

Hello Coffee & Set-up

OPEN KNOW LEDGE


Hello
Coffee

“Run **your business** code
highly-available
in the **cloud** in response
to **events** and **scale without**
any servers to manage.“*

* AWS Lambda Advertising

“ *Function as self-contained application*

Serverless Function: Entwickler schreibt eine Business-Funktion in einer der unterstützen Programmiersprachen, „bundled“ diese mit den entsprechenden Abhängigkeiten (LIBs) und lädt sie in die Cloud.

Serverless Environment: Führt die Funktion bei „Aufruf“ in der passenden Runtime effizient, flexibel und hoch skalierbar aus.

“ *Serverless no need to maintain*

Entwickler: Fokussiert sich ausschließlich auf die Umsetzung der Business-Logik und das Erstellen des Function-Bundle.

Cloud Provider: liefert und maintained rundum-sorglos Umgebung für die Serverless Functions, inklusive etwaiger Cloud Services (z.B. Storage, DB, Streaming, AI).

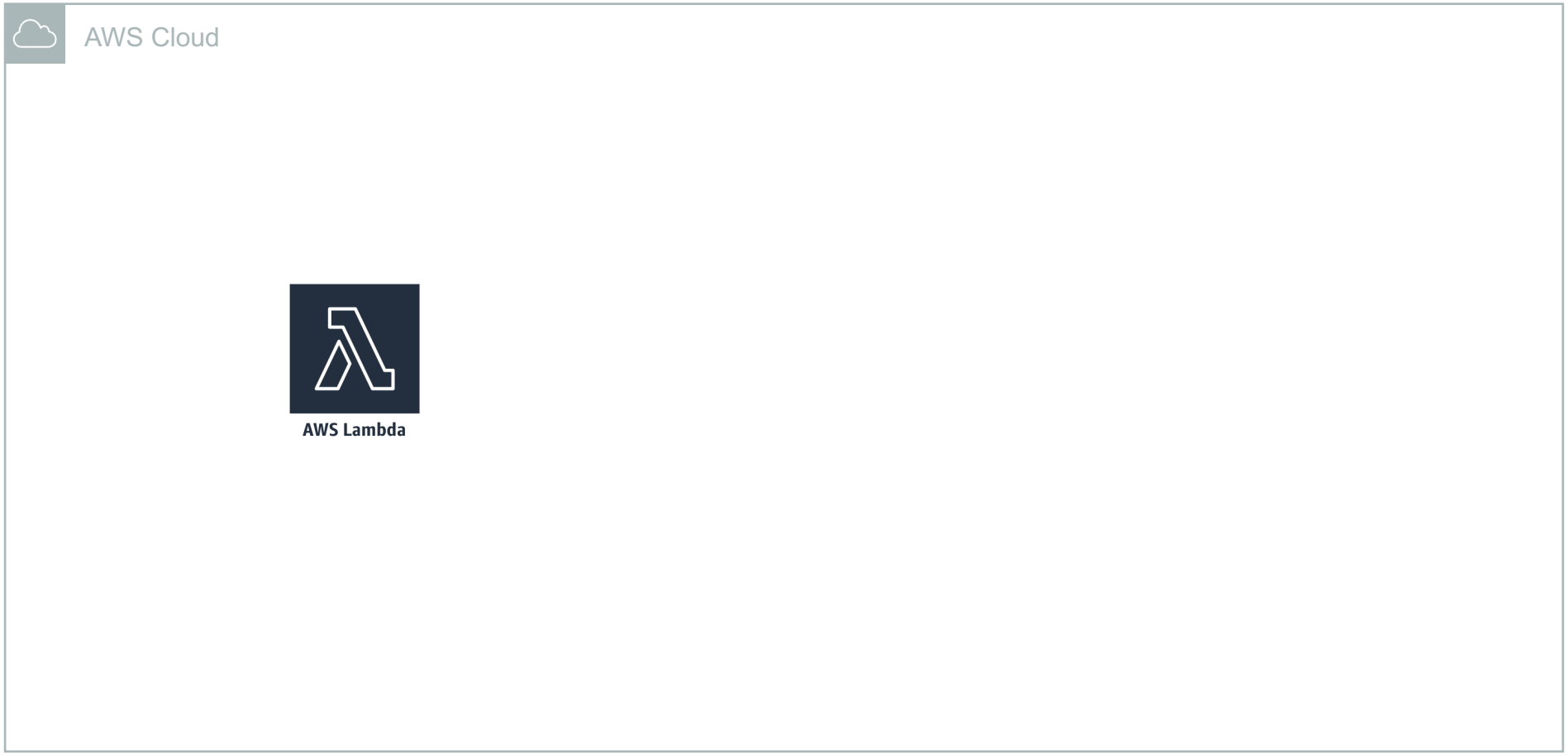

Telekom.de LTE
12:45
Twittern
Kate Pearce
@secvalve
The next time you try and use the word "serverless" just remember it's like calling takeout "kitchenless".
Original (Englisch) übersetzen
30.05.17, 23:32

HANDS-ON

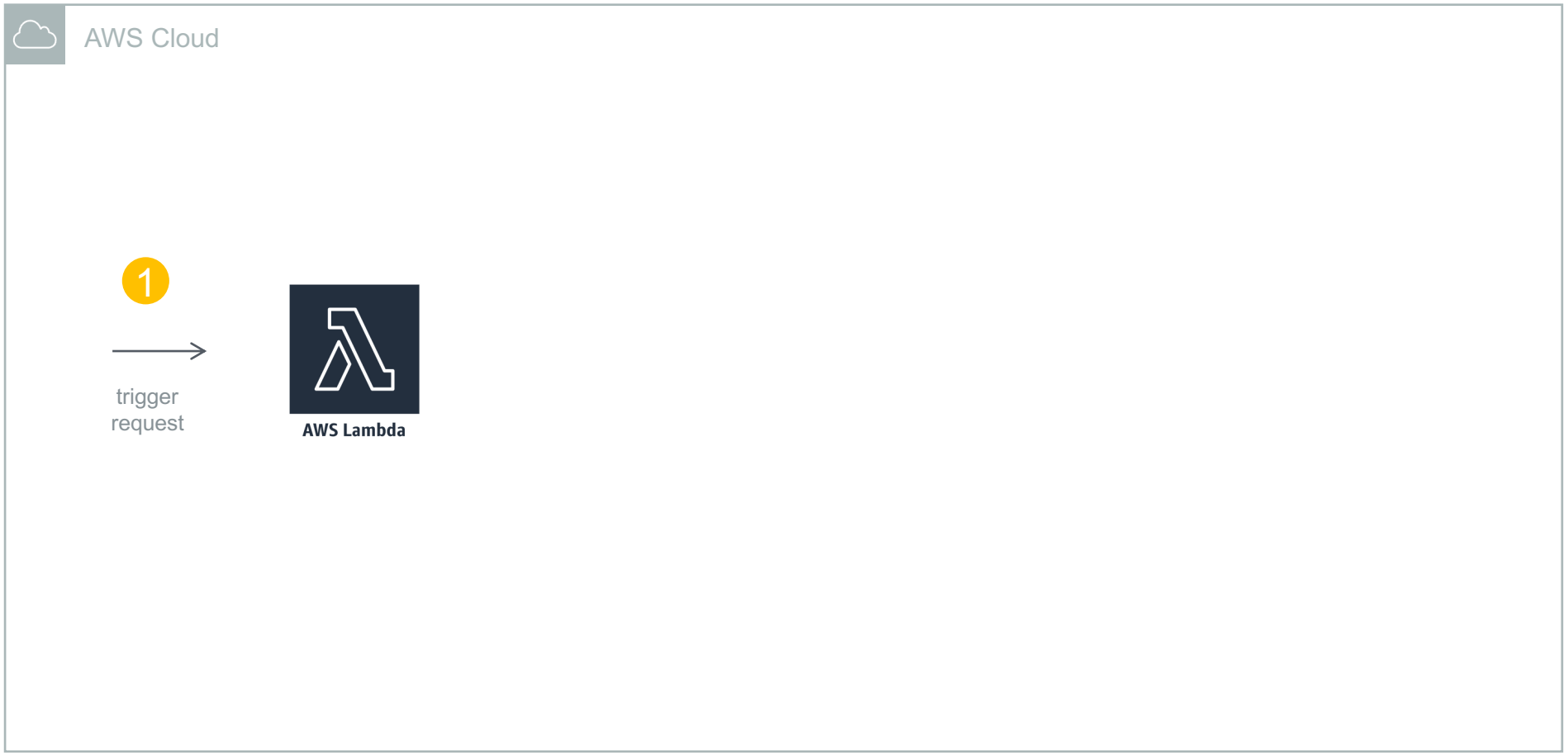
Hello Serverless Coffee

- Implementation
- Upload
- Testing

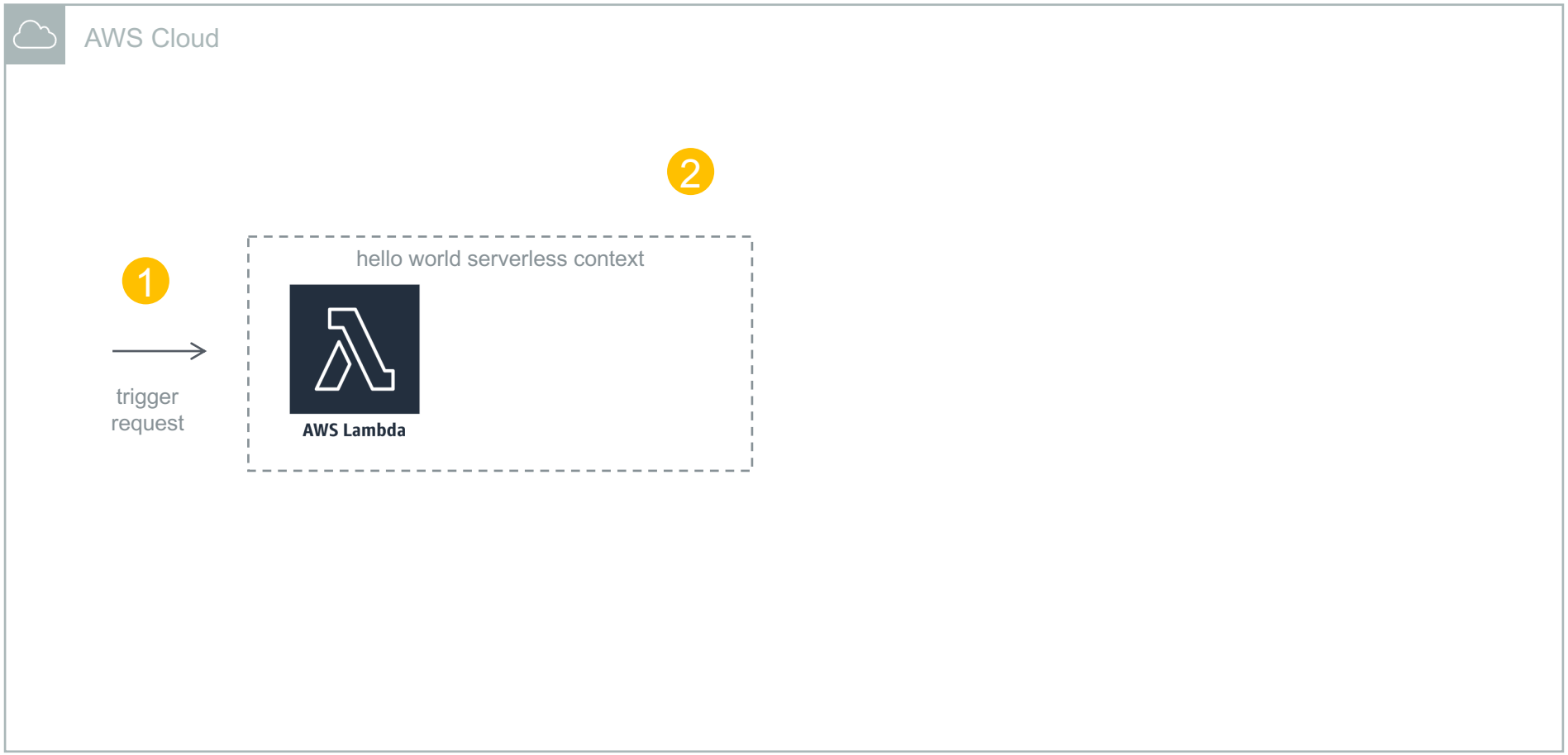
Hands-on: **Hello Serverless Coffee**



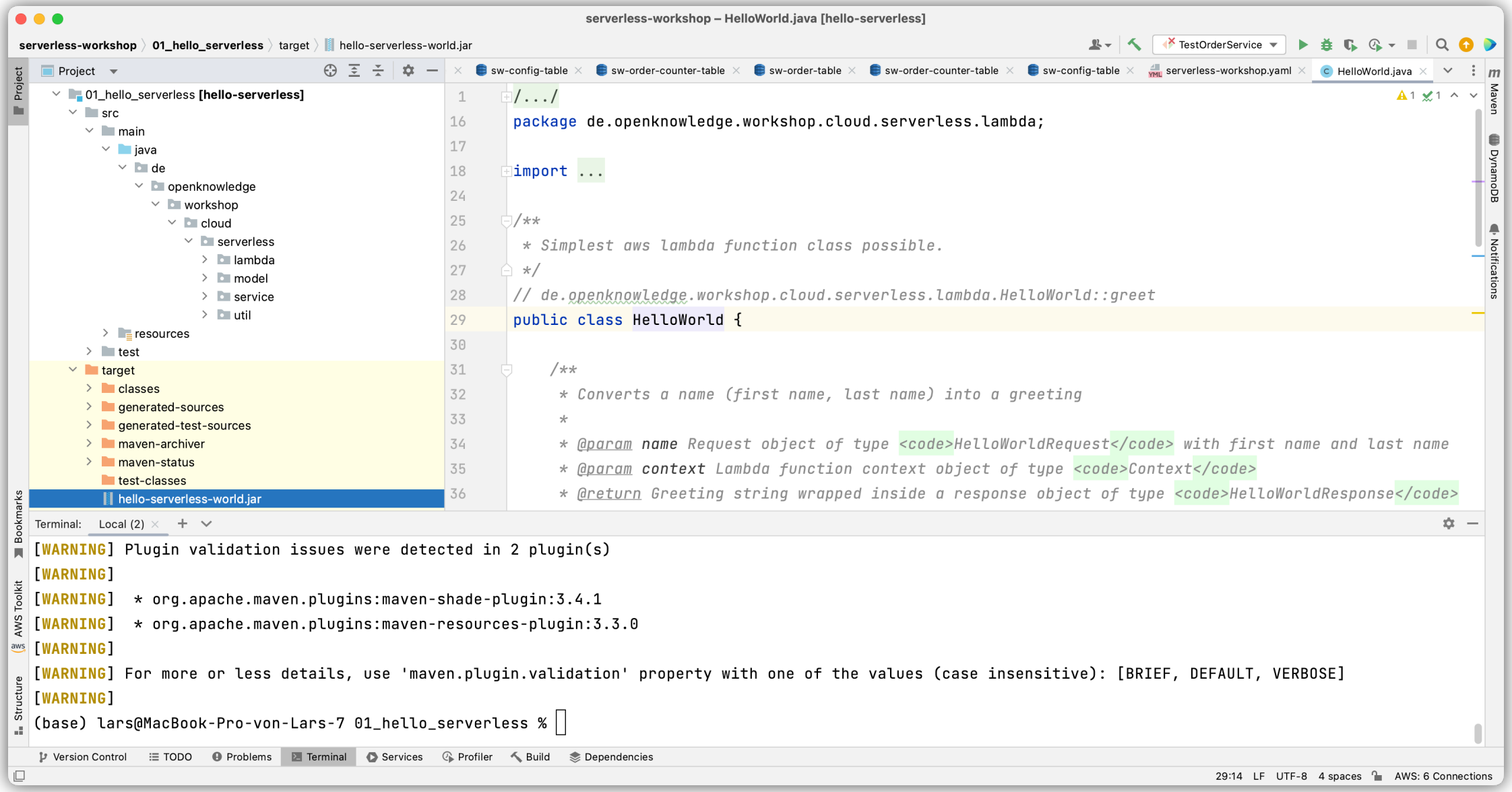
Hands-on: **Hello Serverless Coffee**



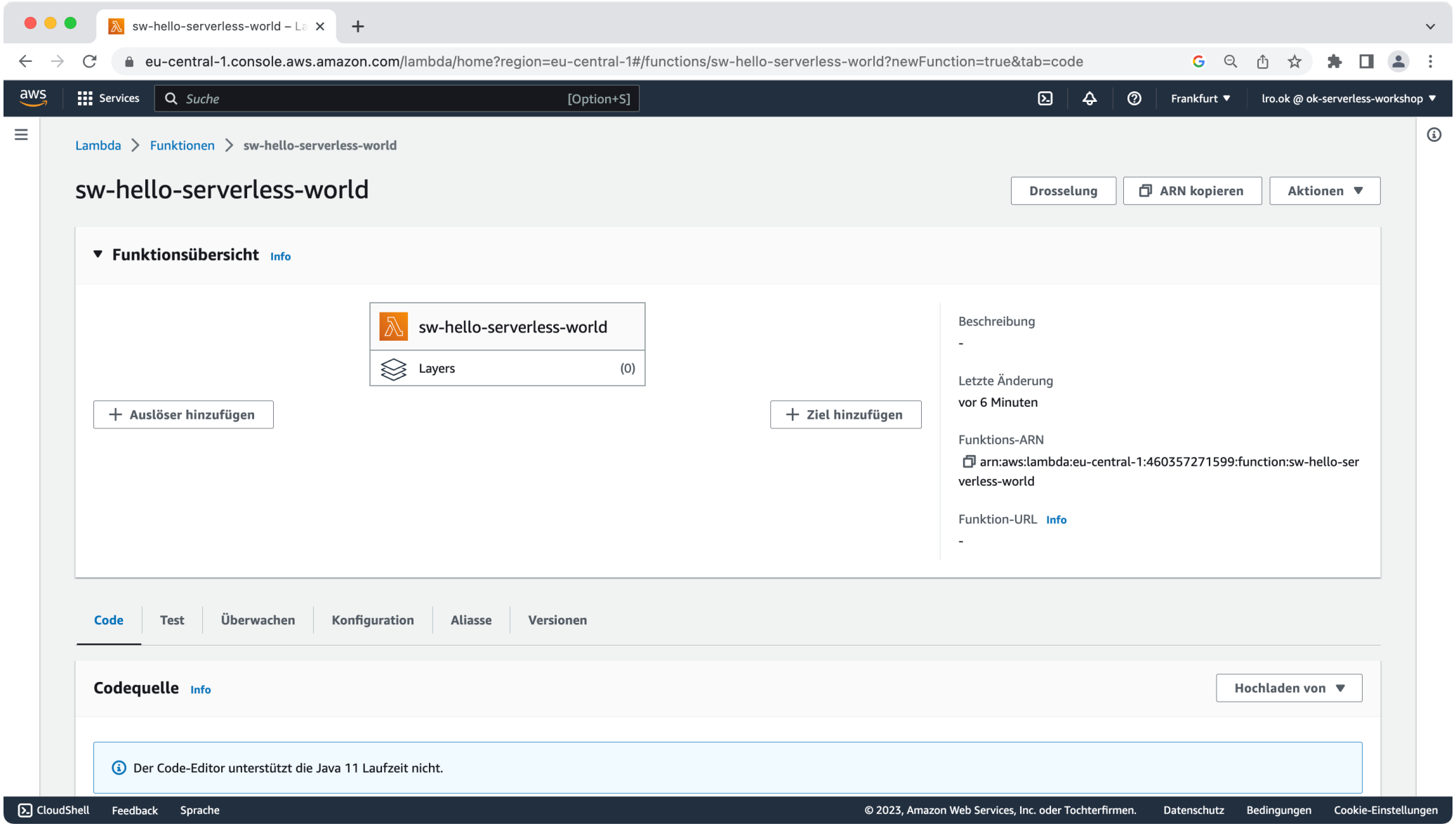
Hands-on: **Hello Serverless Coffee**



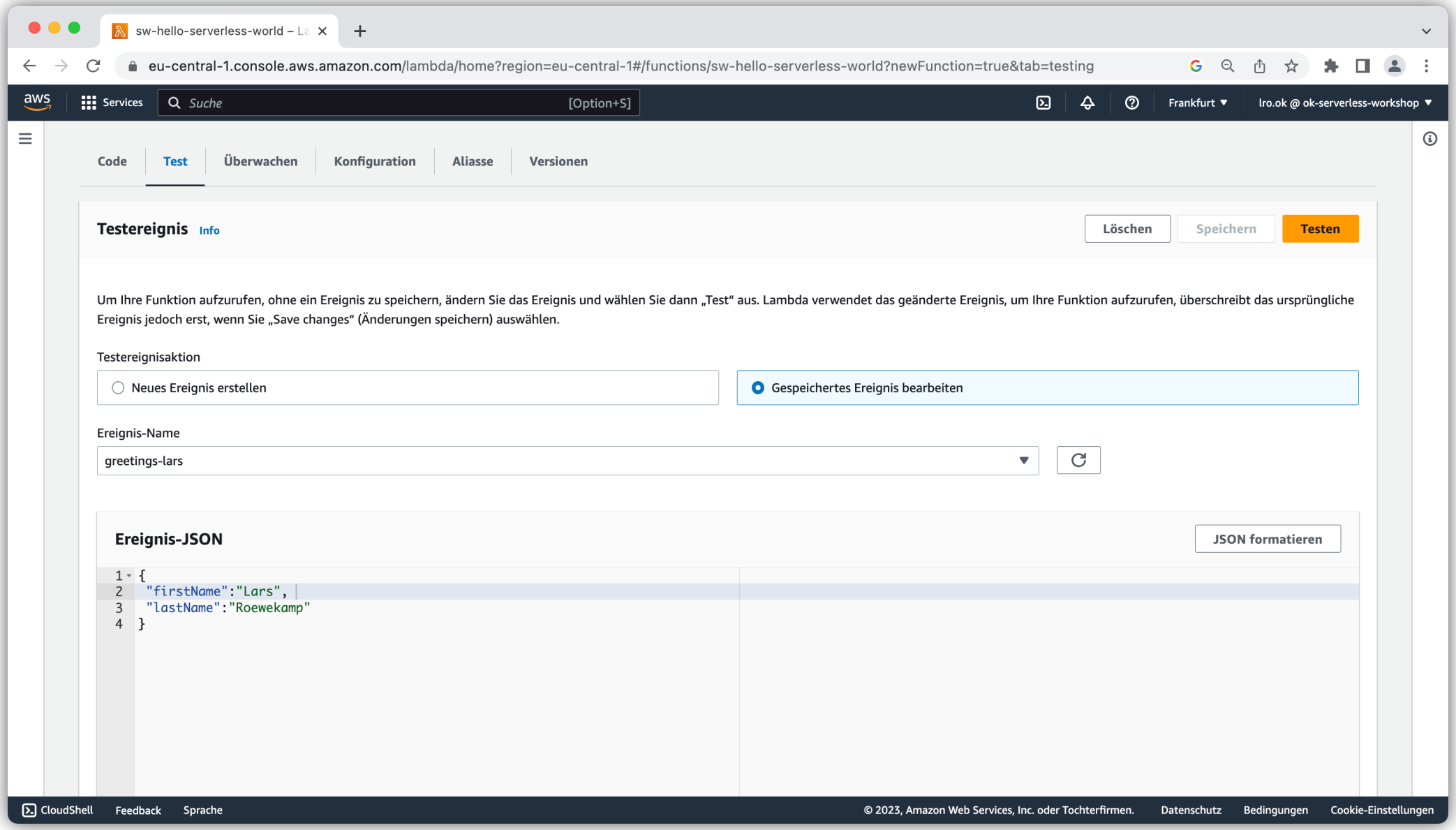
Hands-on: **Hello Serverless Coffee**



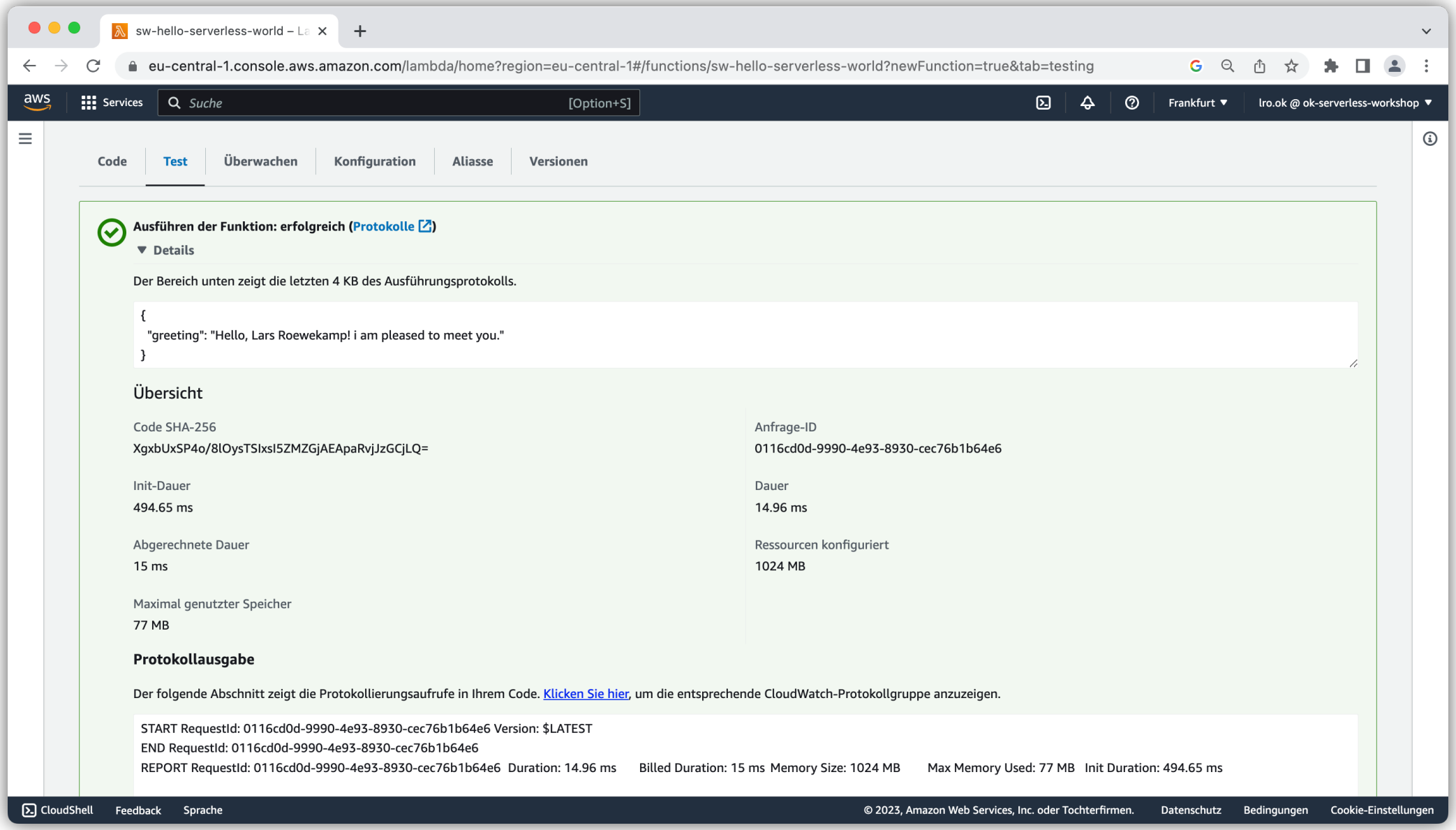
Hello Serverless Coffee: Source Code



Hello Serverless Coffee: Lambda Function Setup



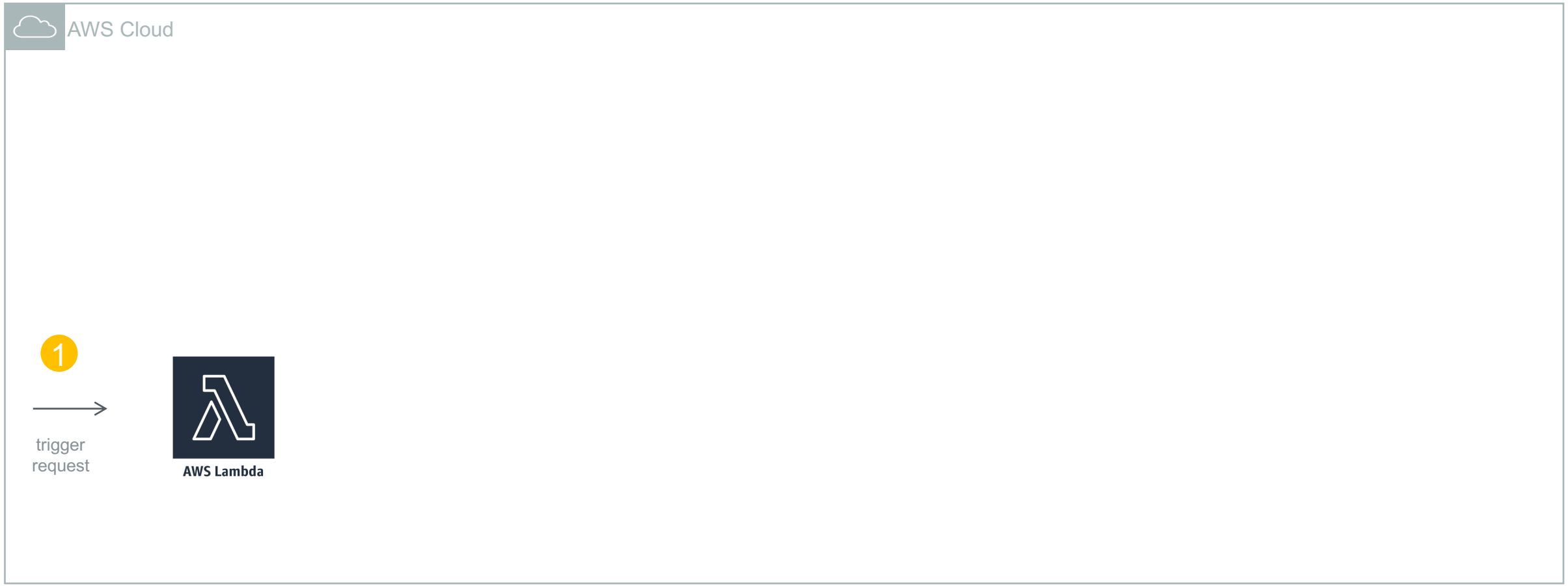
Hello Serverless Coffee: Test Request



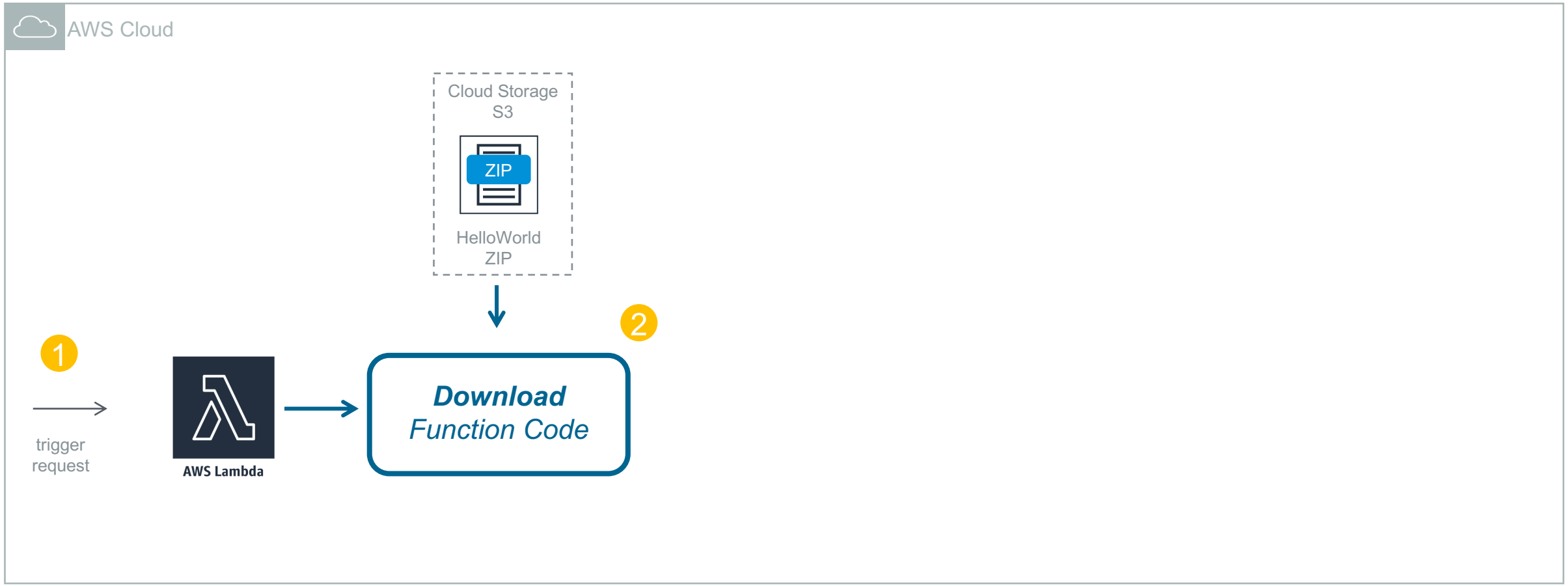
Hello Serverless Coffee: Successful Test


Serverless
unter der Haube

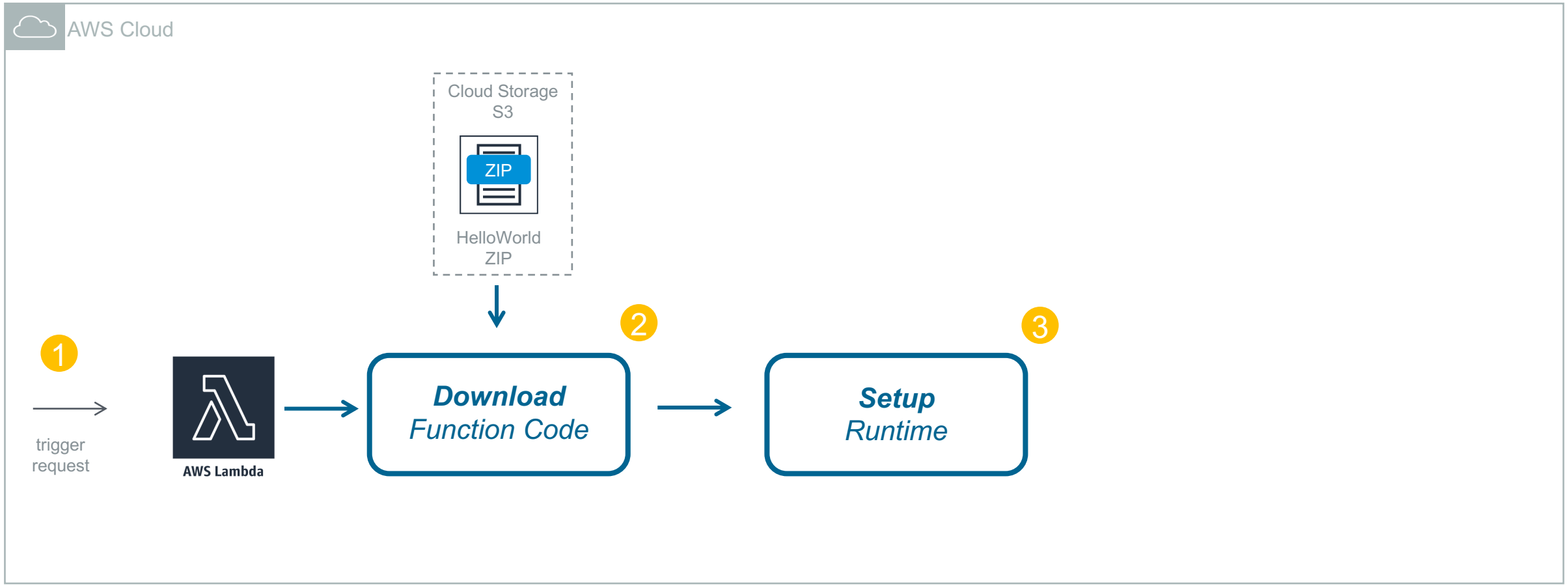
Hello Serverless „under the Hood“



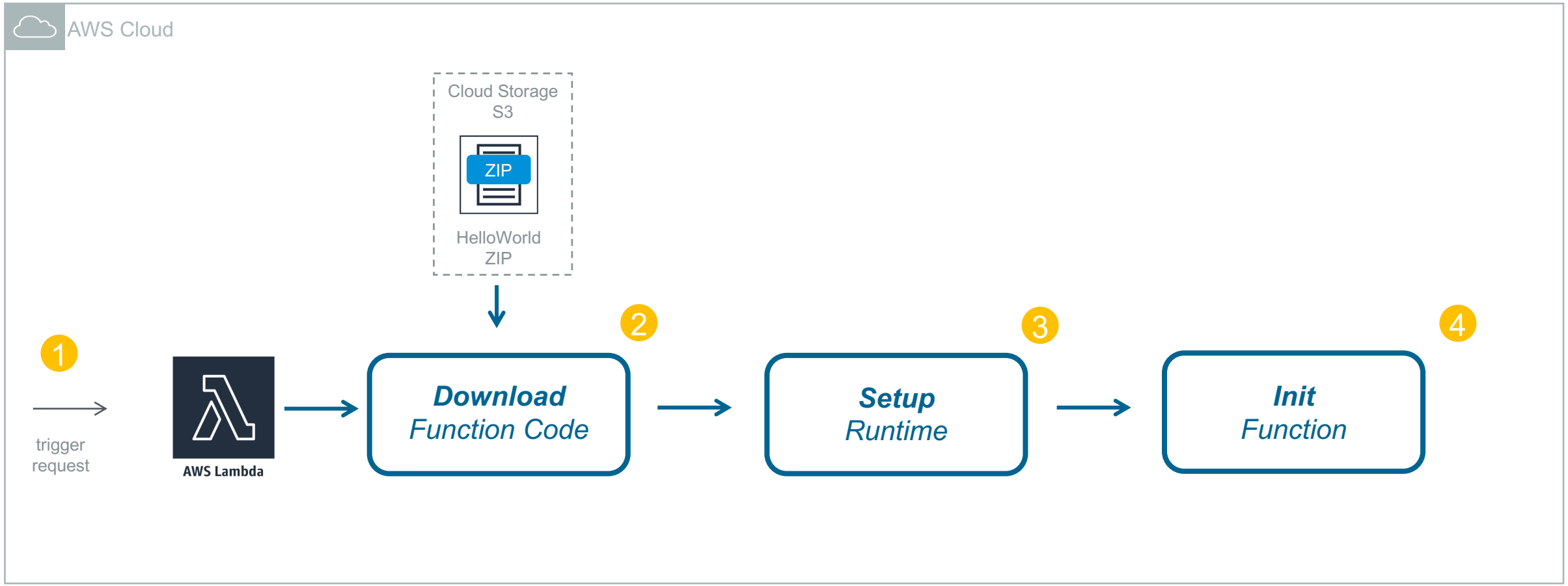
Hello Serverless „under the Hood“



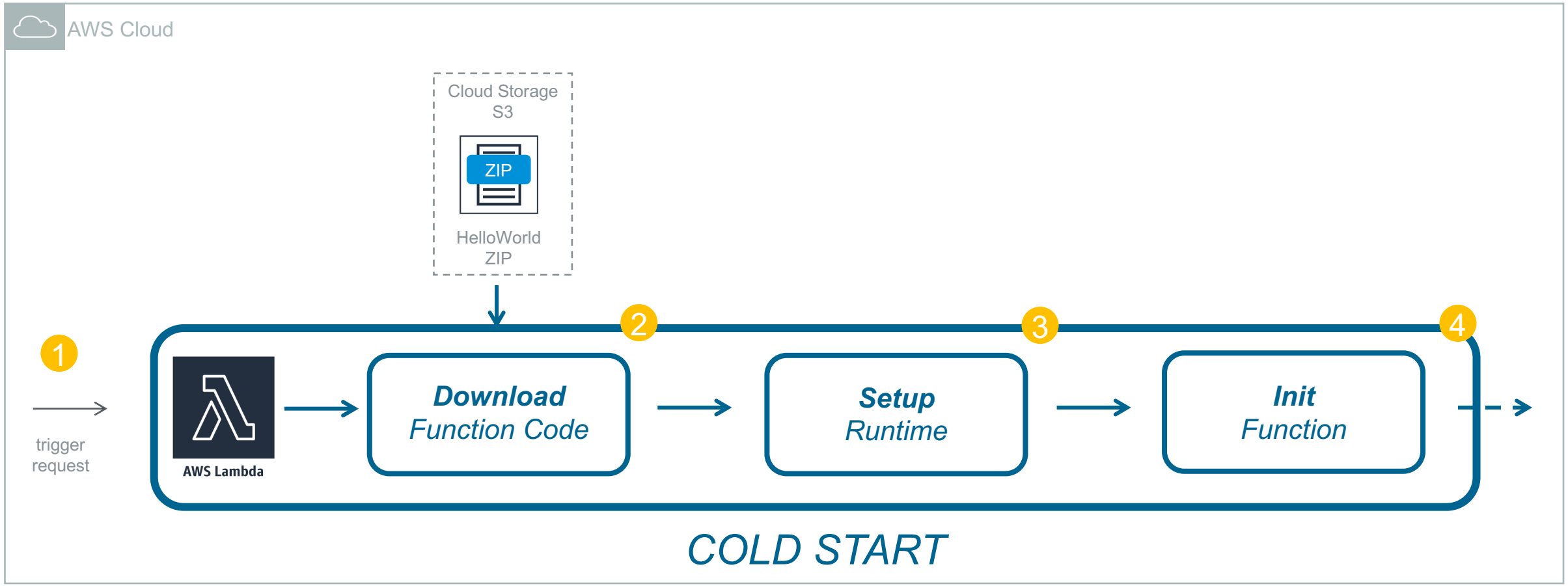
Hello Serverless „under the Hood“



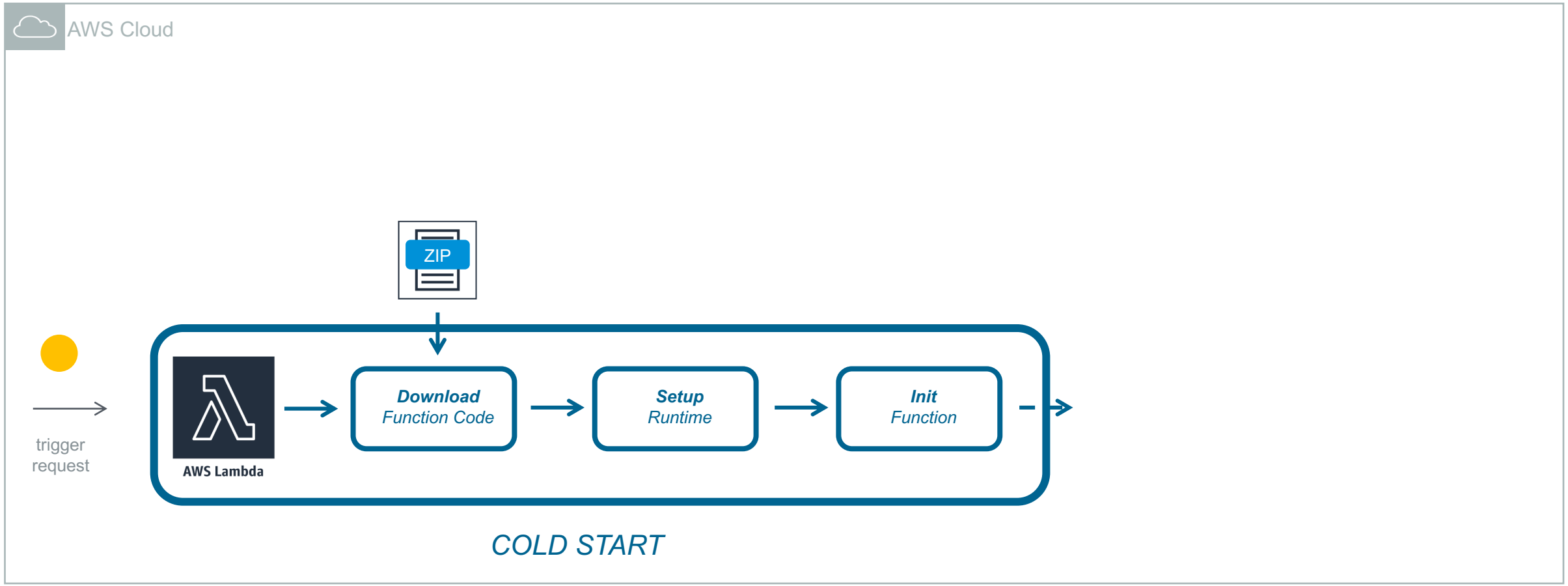
Hello Serverless „under the Hood“



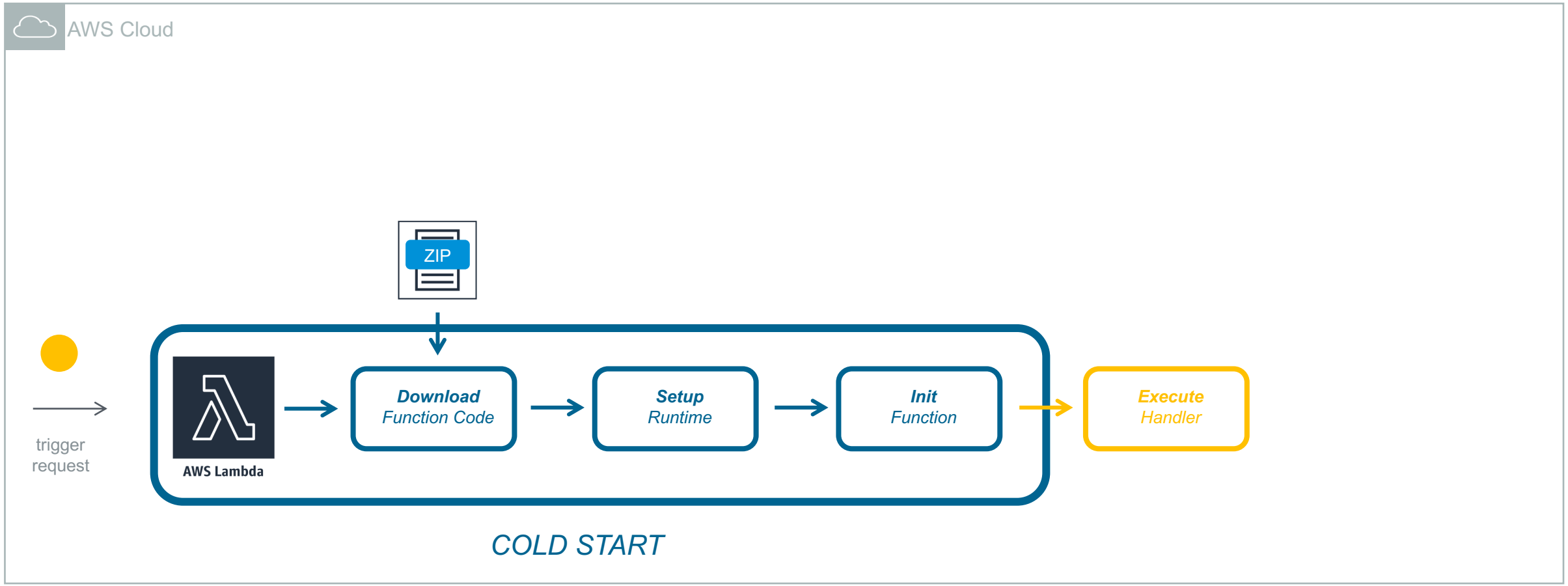
Hello Serverless „under the Hood“



Hello Serverless „under the Hood“



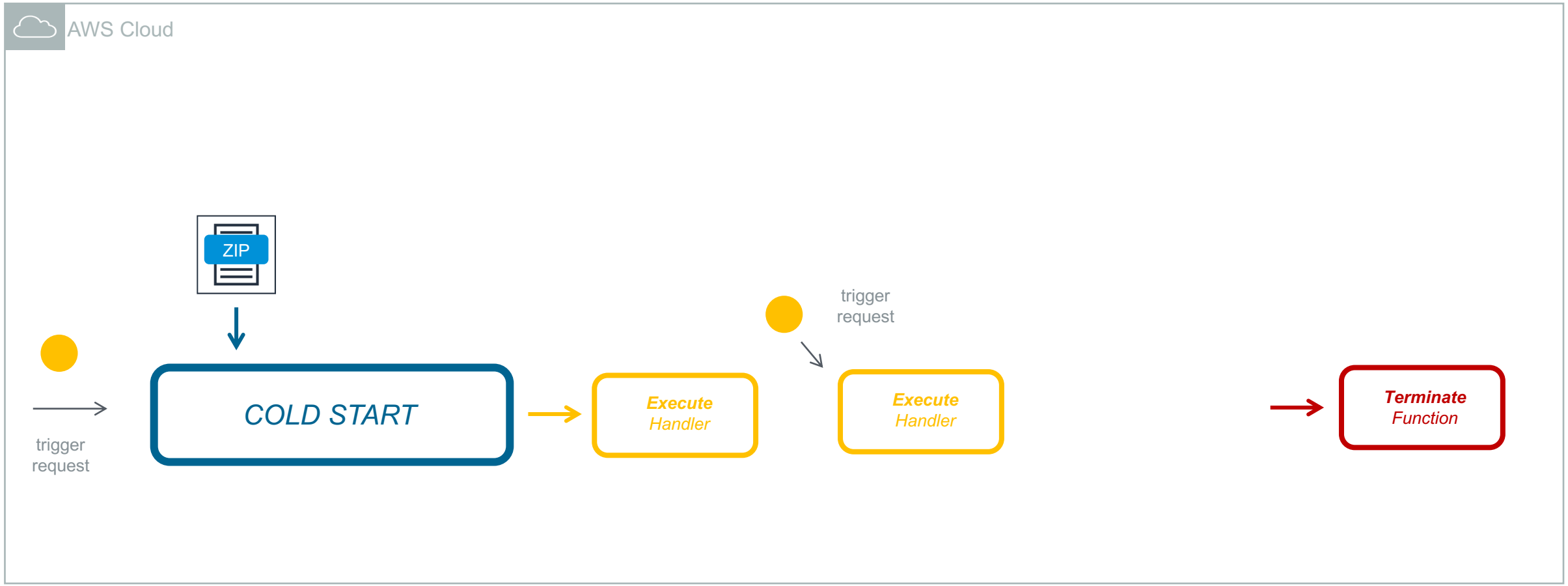
Hello Serverless „under the Hood“



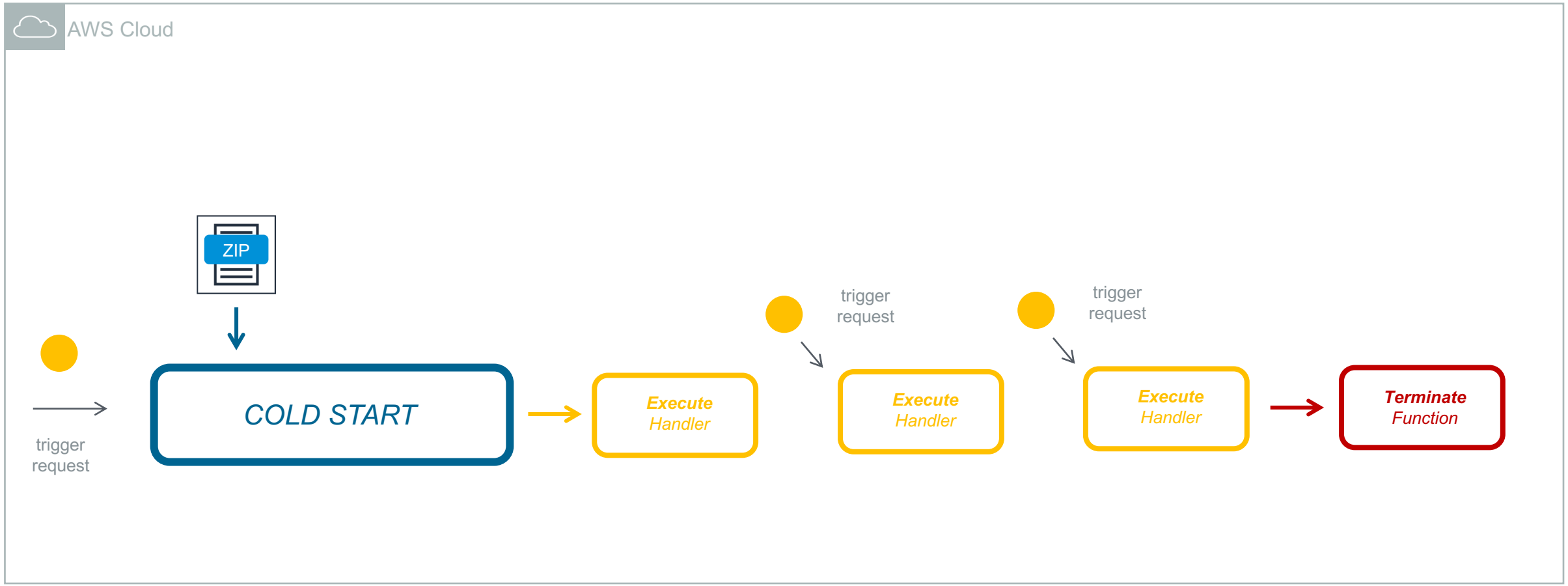
Hello Serverless „under the Hood“

Hello Serverless „under the Hood“

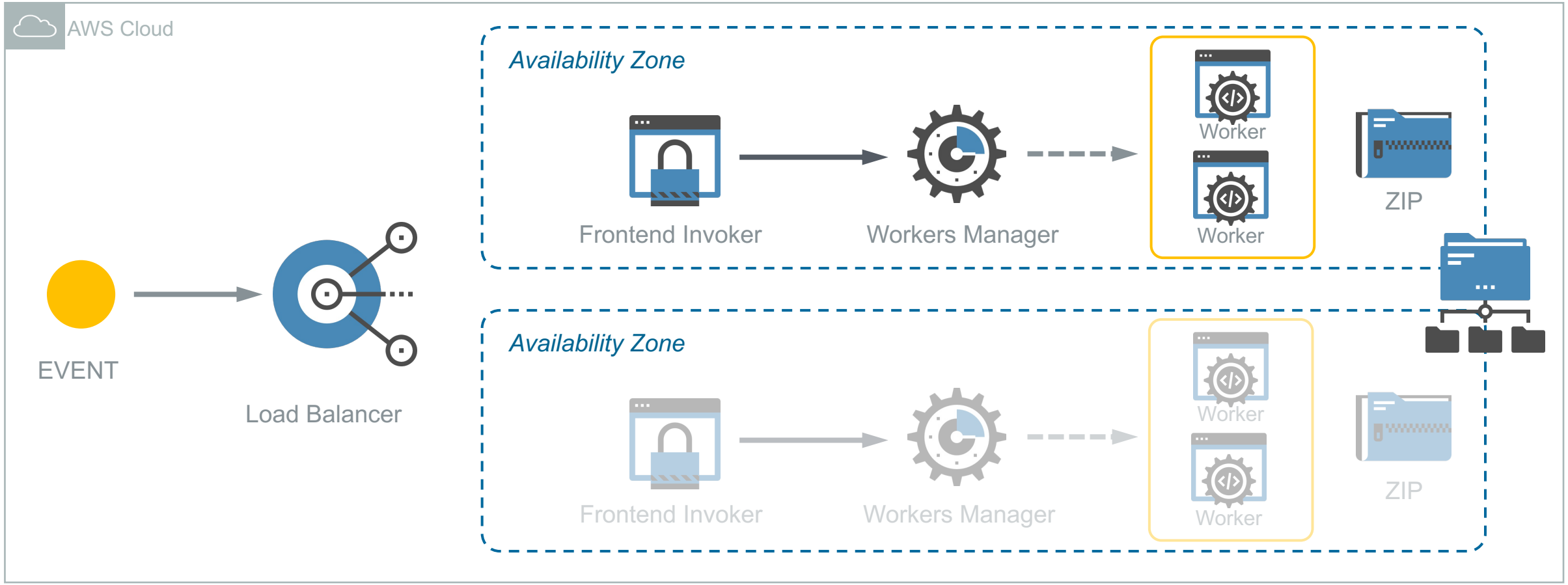
Hello Serverless „under the Hood“



Hello Serverless „under the Hood“

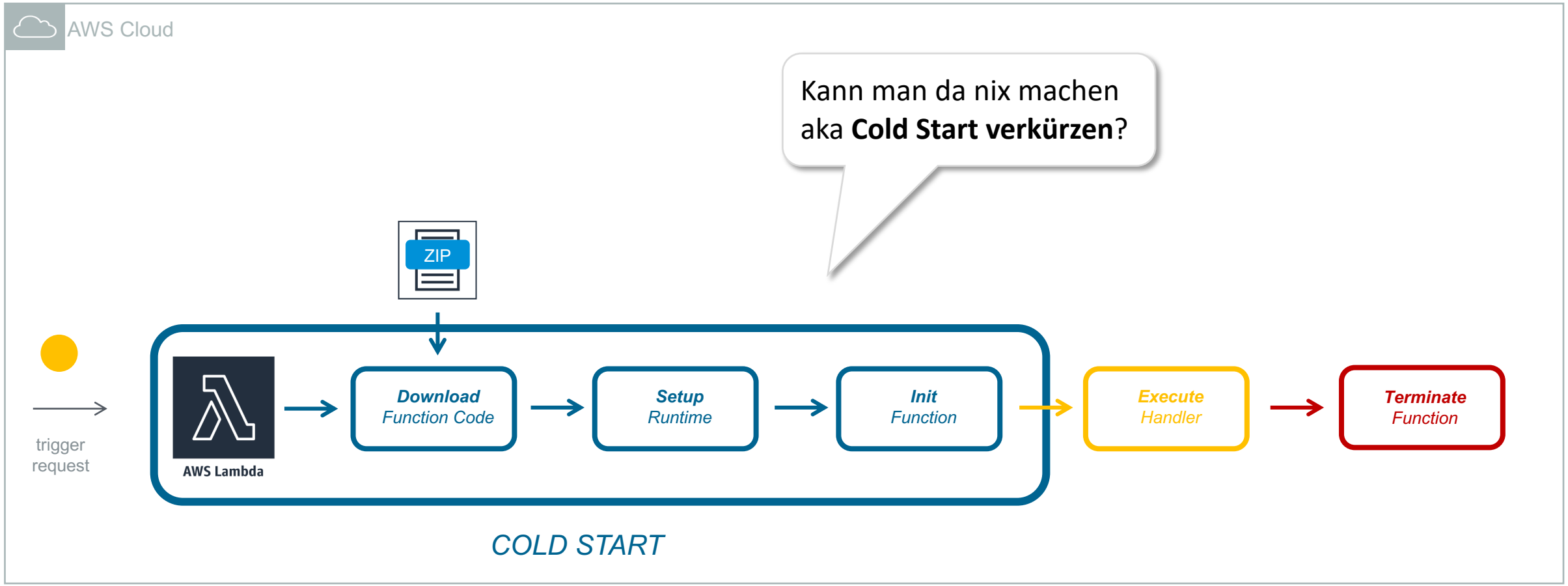


Hello Serverless **REALLY** „under the Hood“

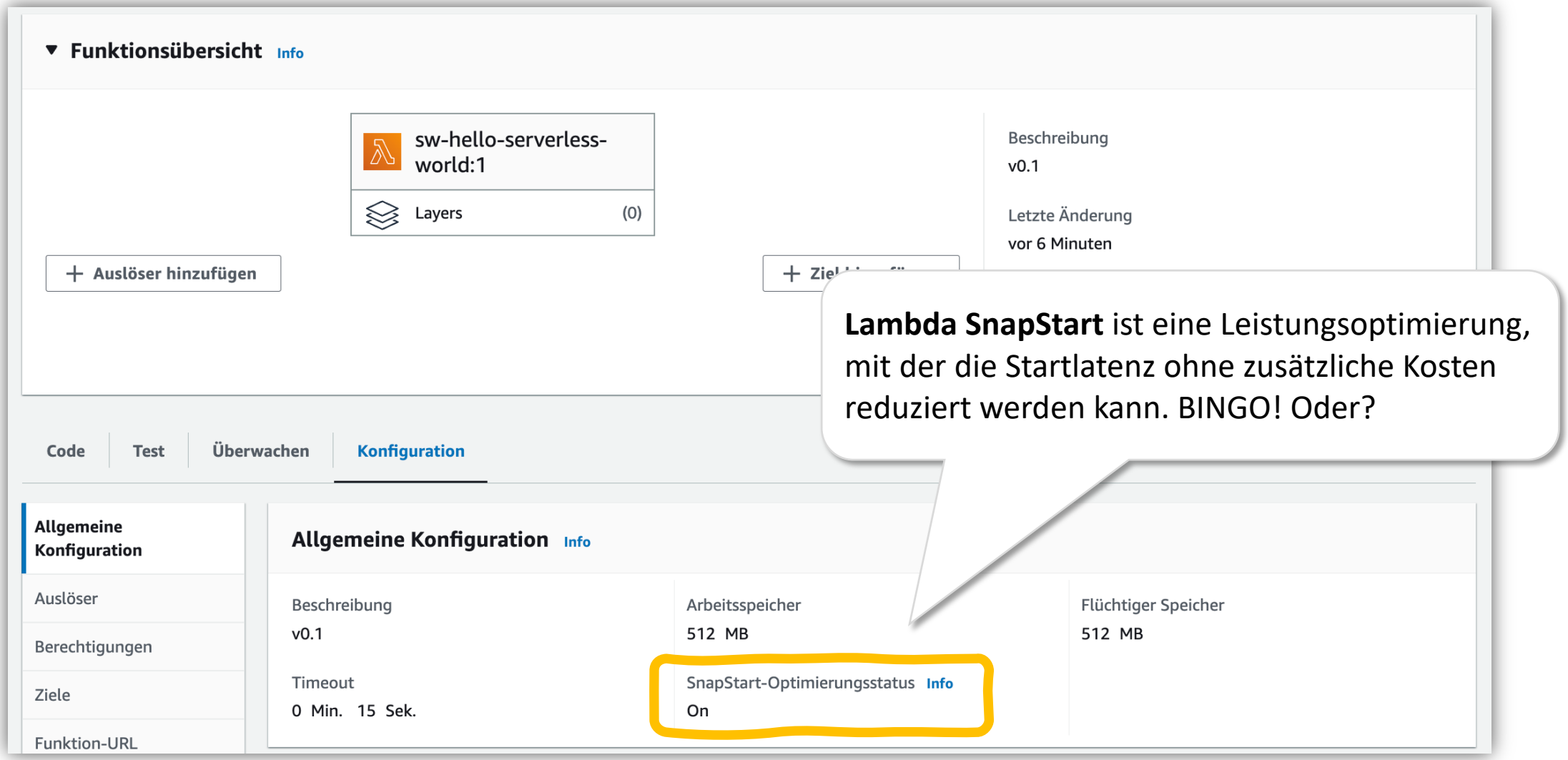



Spaß mit
Cold Start

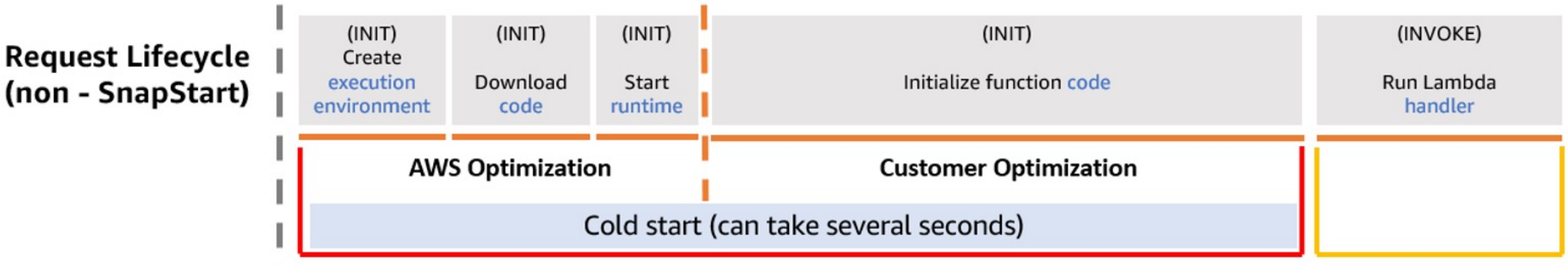
Spaß mit **Cold Start**



Spaß mit **Cold Start: SnapStart**



Spaß mit **Cold Start: SnapStart**



https://aws.amazon.com/de/blogs/compute/starting-up-faster-with-aws-lambda-snapstart/

Spaß mit **Cold Start: SnapStart**

Request Lifecycle (non - SnapStart)

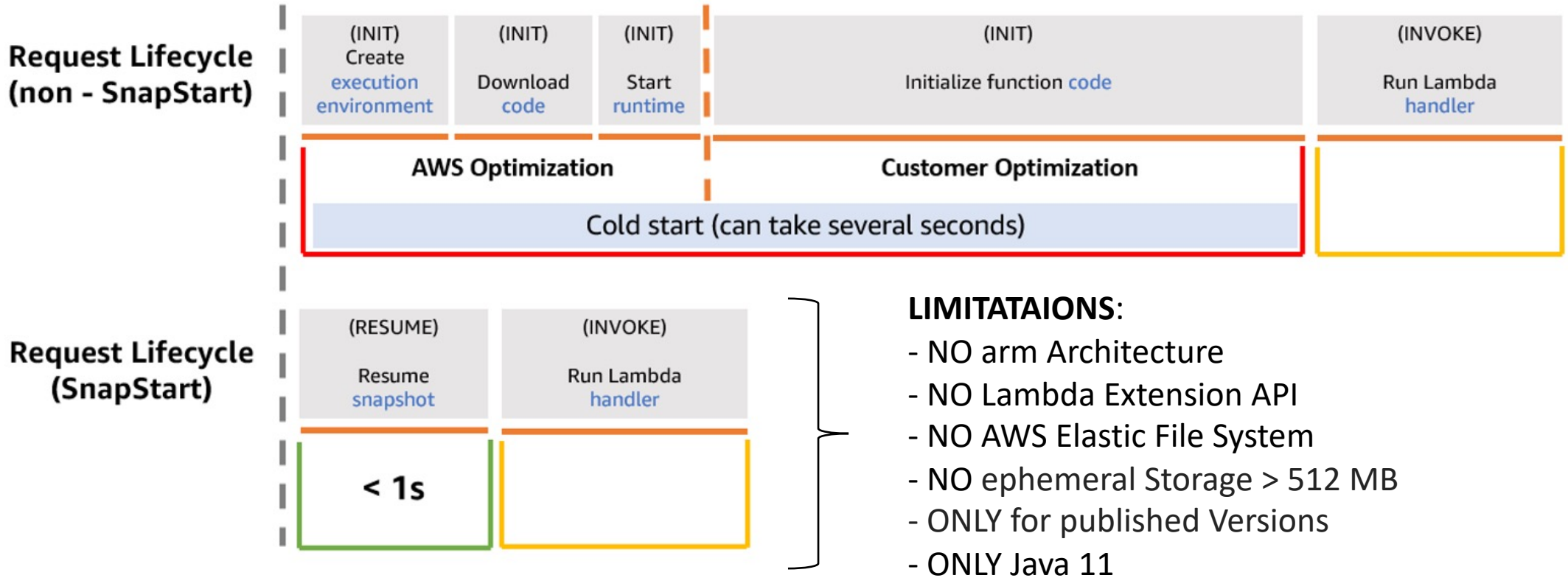
```
REPORT RequestId: 4fae385d-1083-4d0c-a0e1-3c749169b600 Duration: 493.25
ms Billed Duration: 494 ms Memory Size: 512 MB Max Memory Used: 160 MB
Init Duration: 3111.32 ms XRAY TraceId: 1-638fa4d3-
52b791875abf058d12ca200f SegmentId: 5c4362c041371b98 Sampled: true
```

Request Lifecycle (SnapStart)

```
REPORT RequestId: d1c283c7-cf6c-4ff6-a141-484ac460ae90 Duration: 470.25
ms Billed Duration: 803 ms Memory Size: 512 MB Max Memory Used: 124 MB
Restore Duration: 469.23 ms
```

https://aws.amazon.com/de/blogs/compute/starting-up-faster-with-aws-lambda-snapstart/

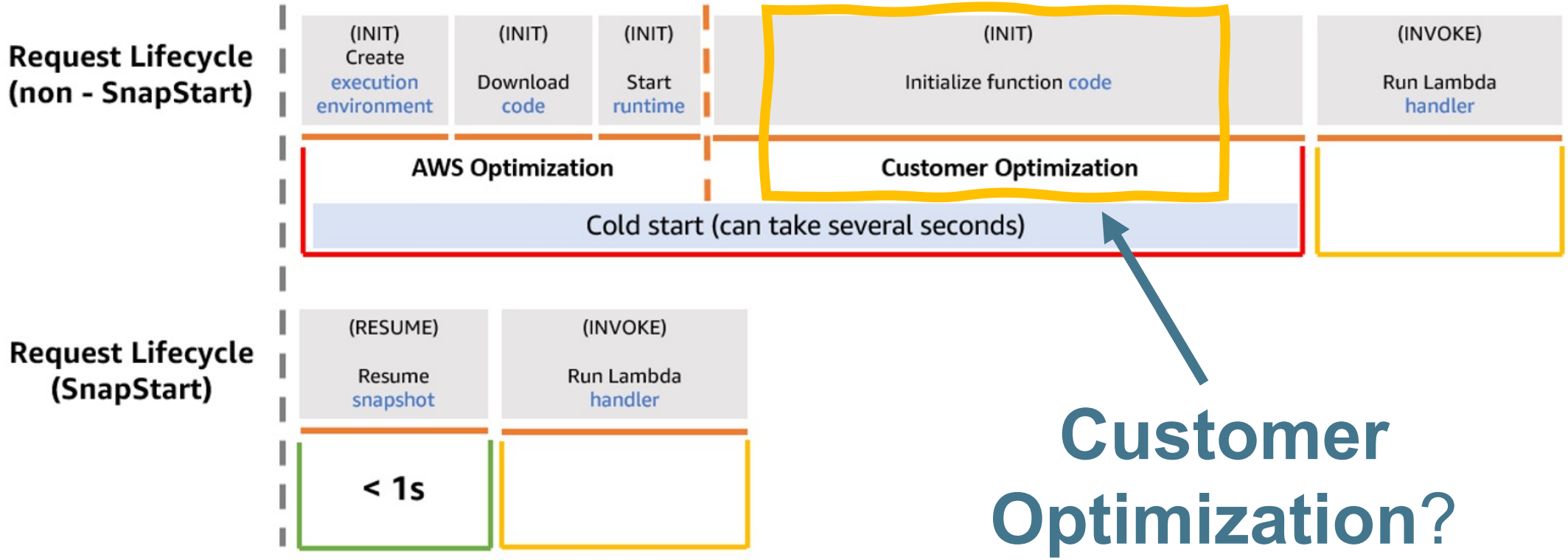
Spaß mit **Cold Start: SnapStart**



LIMITATAIONS:
- NO arm Architecture
- NO Lambda Extension API
- NO AWS Elastic File System
- NO ephemeral Storage > 512 MB
- ONLY for published Versions
- ONLY Java 11

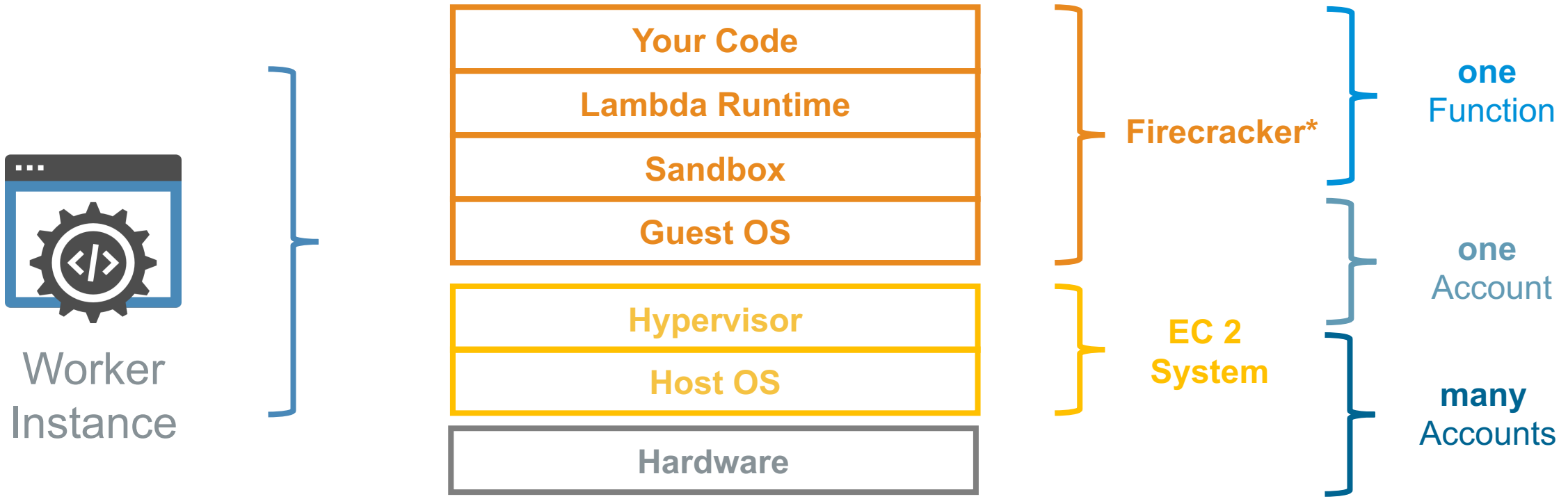
https://aws.amazon.com/de/blogs/compute/starting-up-faster-with-aws-lambda-snapstart/

Spaß mit **Cold Start: SnapStart**



https://aws.amazon.com/de/blogs/compute/starting-up-faster-with-aws-lambda-snapstart/

Spaß mit **Cold Start: Initialization**



*https://firecracker-microvm.github.io

Spaß mit **Cold Start**

Initialization vs. Handler

```
1  public class ReusableHandler implements RequestHandler<Input, Output> {
2
3      // dynamic code running each time
4      public Output handleRequest(Input input, Context context) {
5
6          Connection expensiveConnection = ConnectionFactory.getConnection(EXPENSIVE);
7
8          DataFetcher dataFetcher = expensiveConnection.initDataFetcher();
9          Data data = dataFetcher.getData();
10
11         return new Output(data);
12     }
13
14 }
```

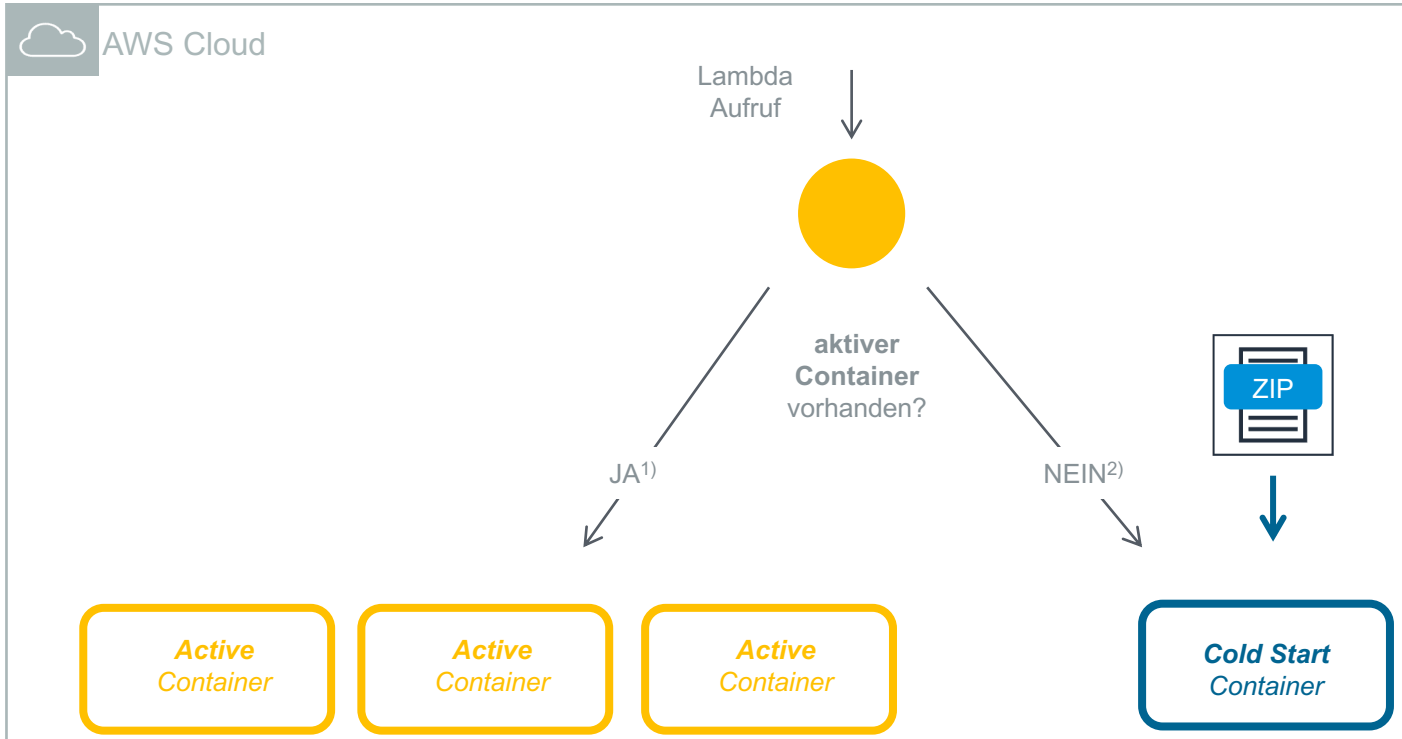
Spaß mit **Cold Start**

Initialization vs. Handler

```
1  public class ReusableHandler implements RequestHandler<Input, Output> {
2
3      // static code running only once (per lambda container instance)
4      private static Connection EXPENSIVE_CONNECTION
5                                          = ConnectionFactory.getConnection(EXPENSIVE);
6
7      // dynamic code running each time
8      public Output handleRequest(Input input, Context context) {
9
10         DataFetcher dataFetcher = EXPENSIVE_CONNECTION.initDataFetcher();
11         Data data = dataFetcher.getData();
12
13         return new Output(data);
14     }
15
16 }
```


Spaß mit
Concurrency

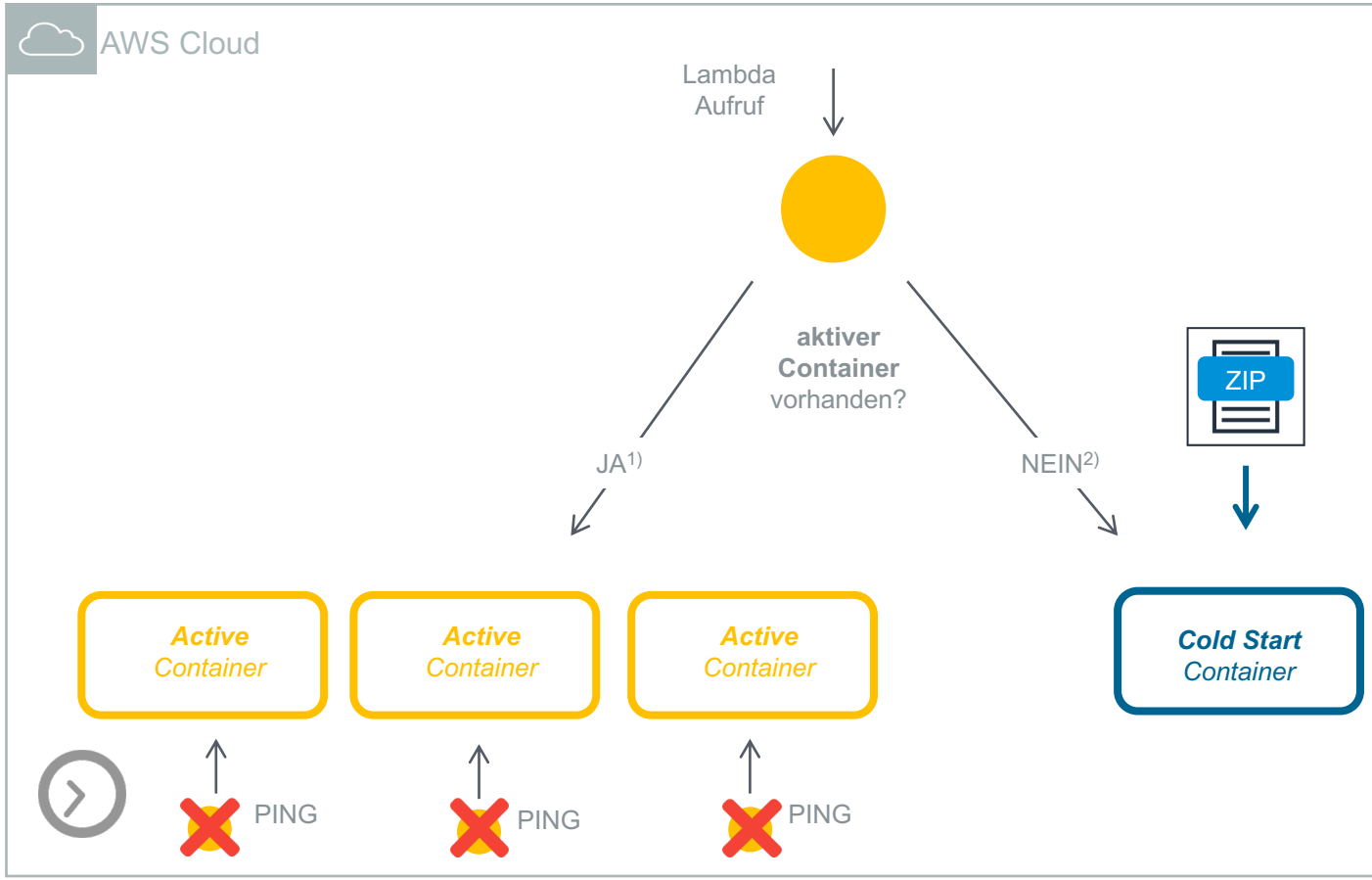
Spaß mit **Concurrency**



1) **Aktiver Container**, der nicht gerade einen Lambda Aufruf abarbeitet, kann wieder / weiter verwendet werden.

2) **Neuer Container** muss erzeugt, Function Code herunter geladen und die Runtime neu initialisiert werden.

Spaß mit **Concurrency**

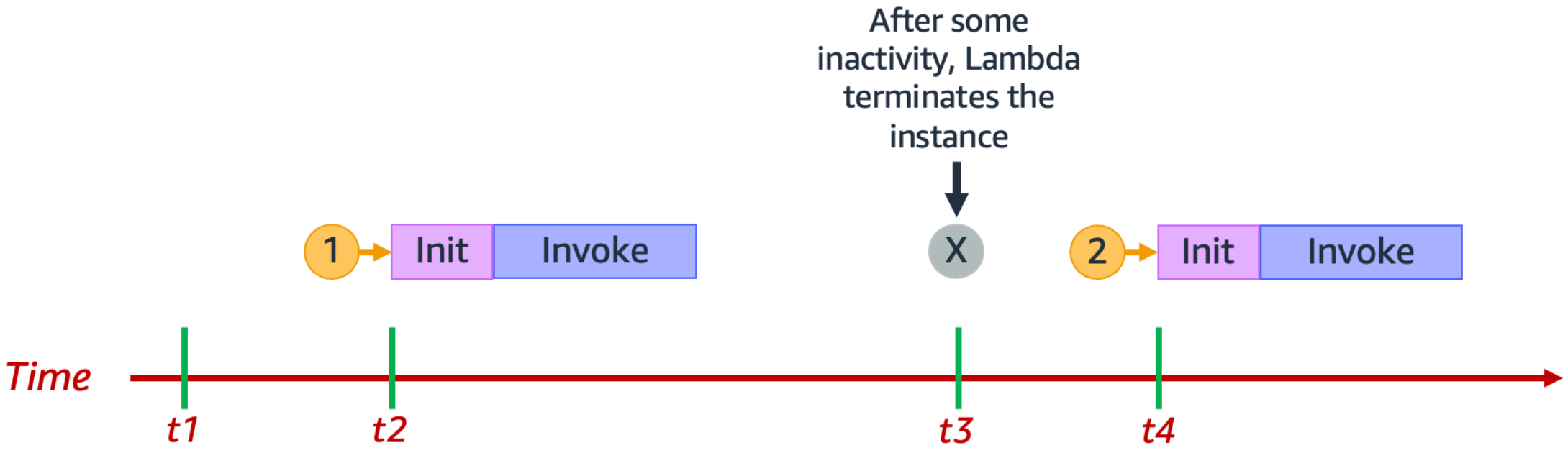


1) **Aktiver Container**, der nicht gerade einen Lambda Aufruf abarbeitet, kann wieder / weiter verwendet werden.

2) **Neuer Container** muss erzeugt, Function Code herunter geladen und die Runtime neu initialisiert werden.

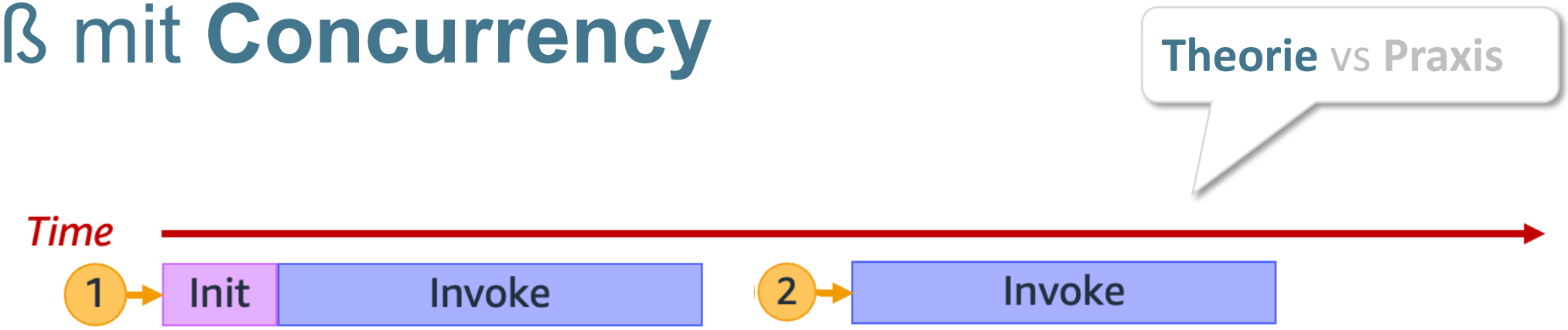
„Provisioned Concurrency“ aka $$$, d.h. so viel wie nötig, so wenig wie möglich!

Spaß mit **Concurrency**



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

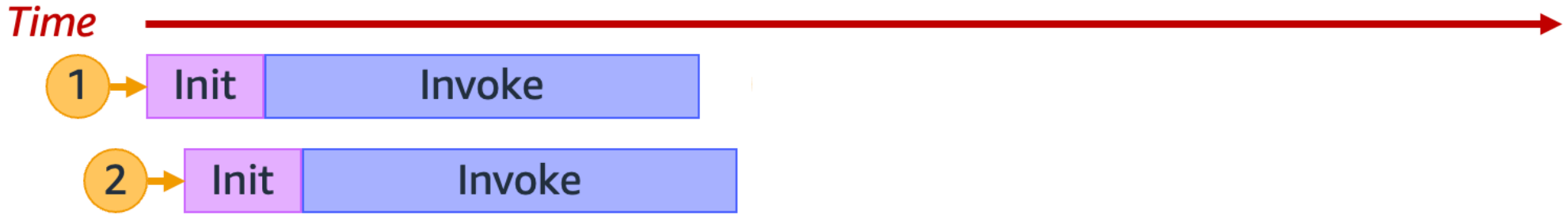
Spaß mit **Concurrency**



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

Spaß mit **Concurrency**

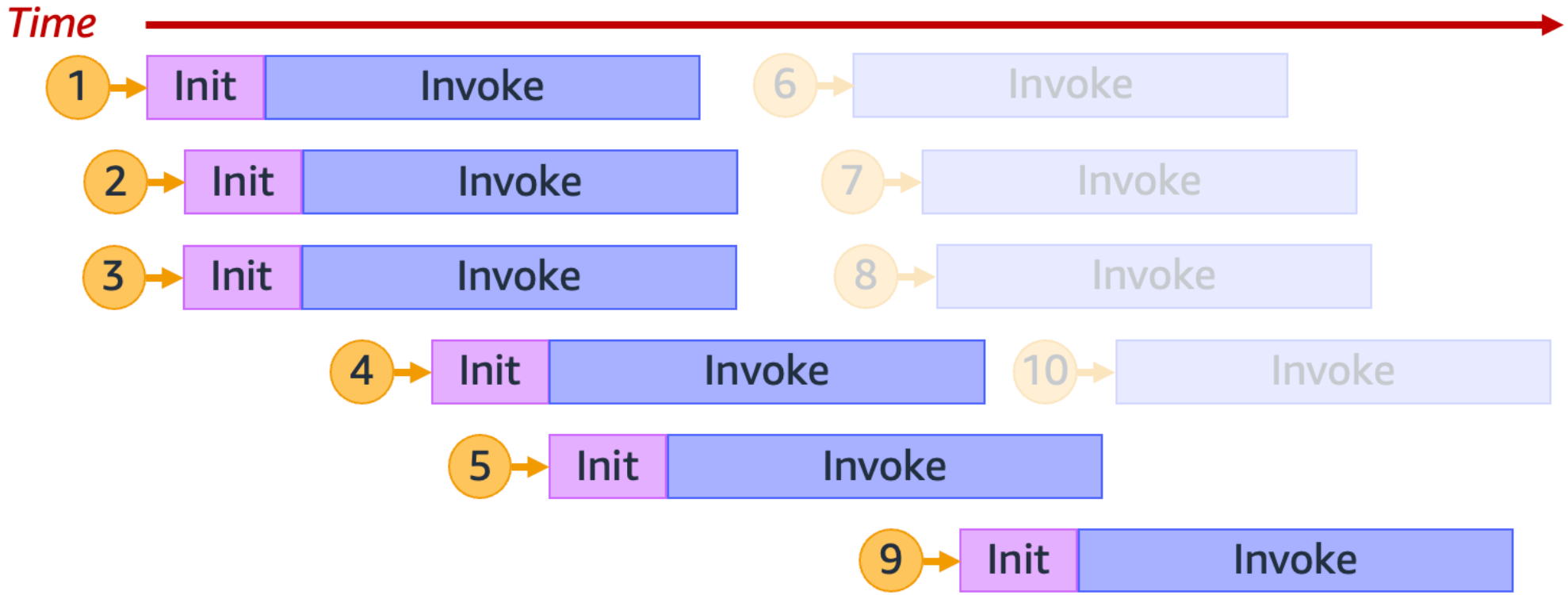
Theorie vs **Praxis**



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

Spaß mit **Concurrency**

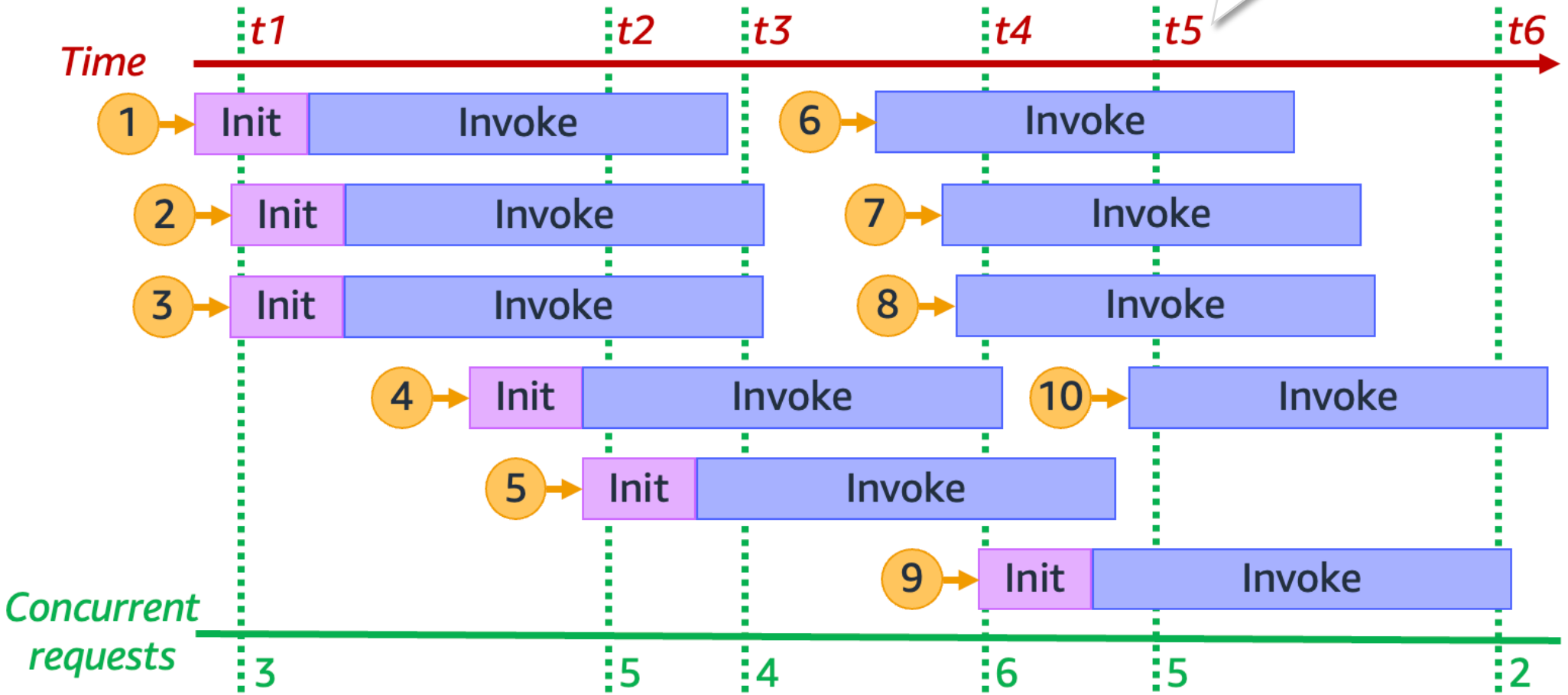
Theorie vs **Praxis**



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

Spaß mit **Concurrency**

Theorie vs **Praxis**



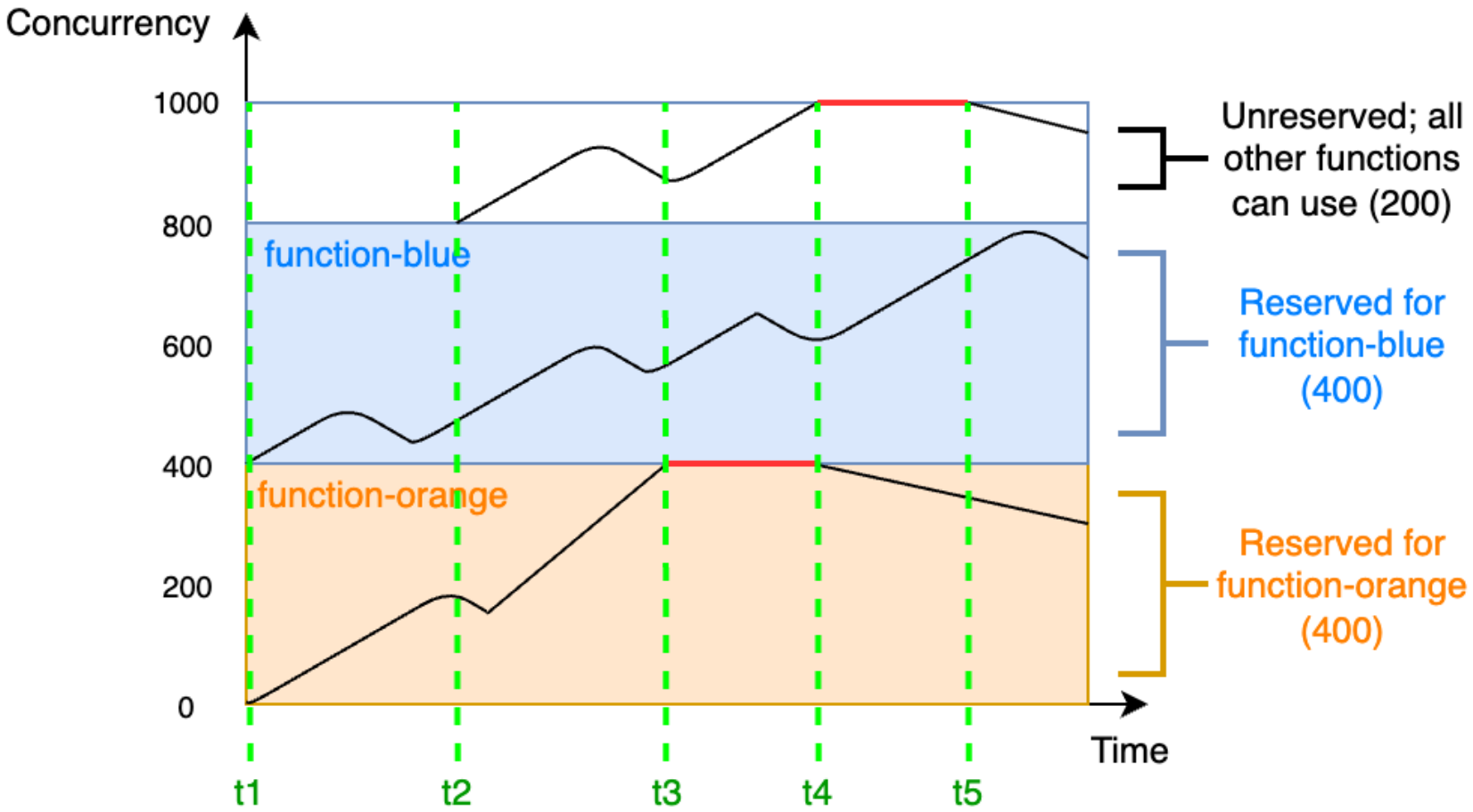
https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html

Spaß mit **Concurrency**

```
Concurrency =    (average req per sec)
               *  (average req duration in sec)
               +   some buffer
```

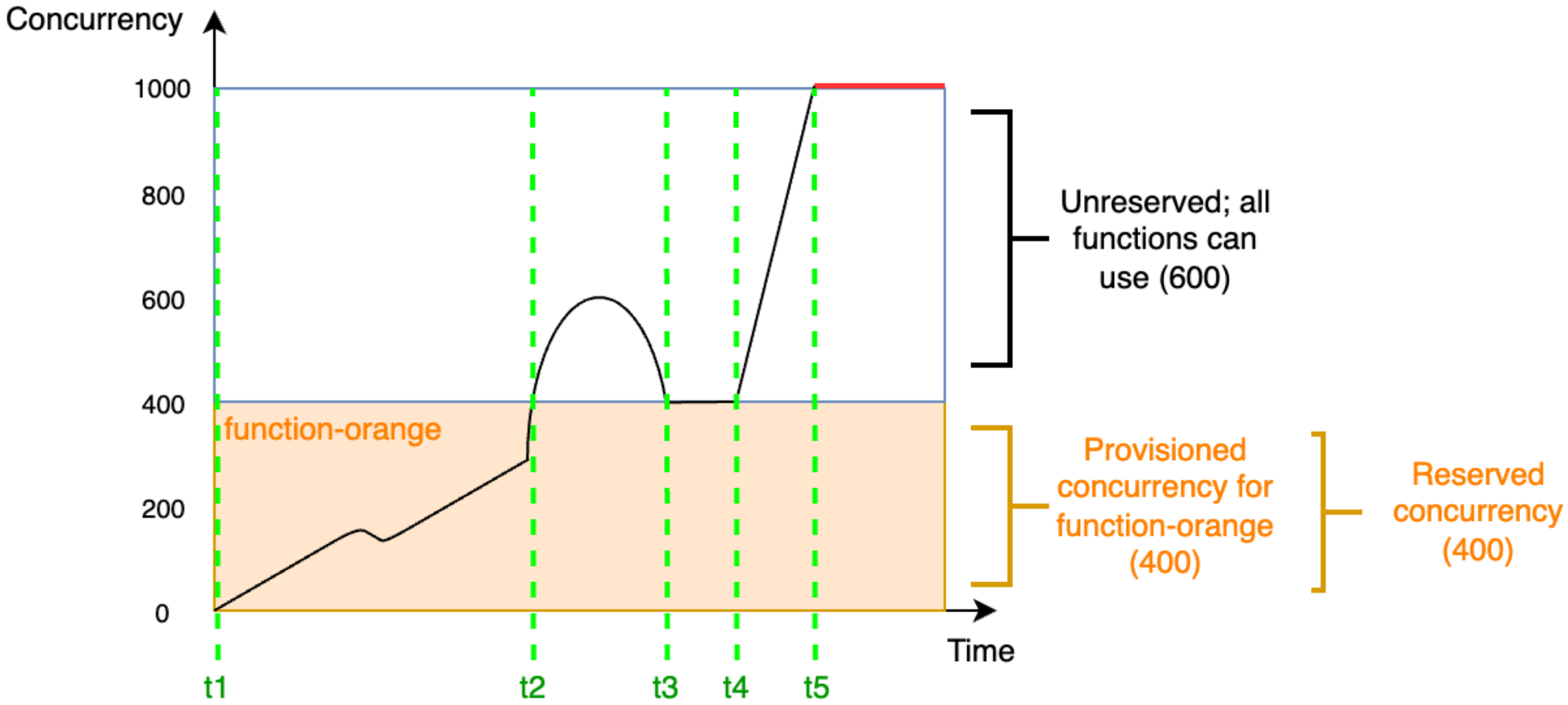
(max 1000 concurrent invocations: https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html)

Spaß mit **Concurrency**



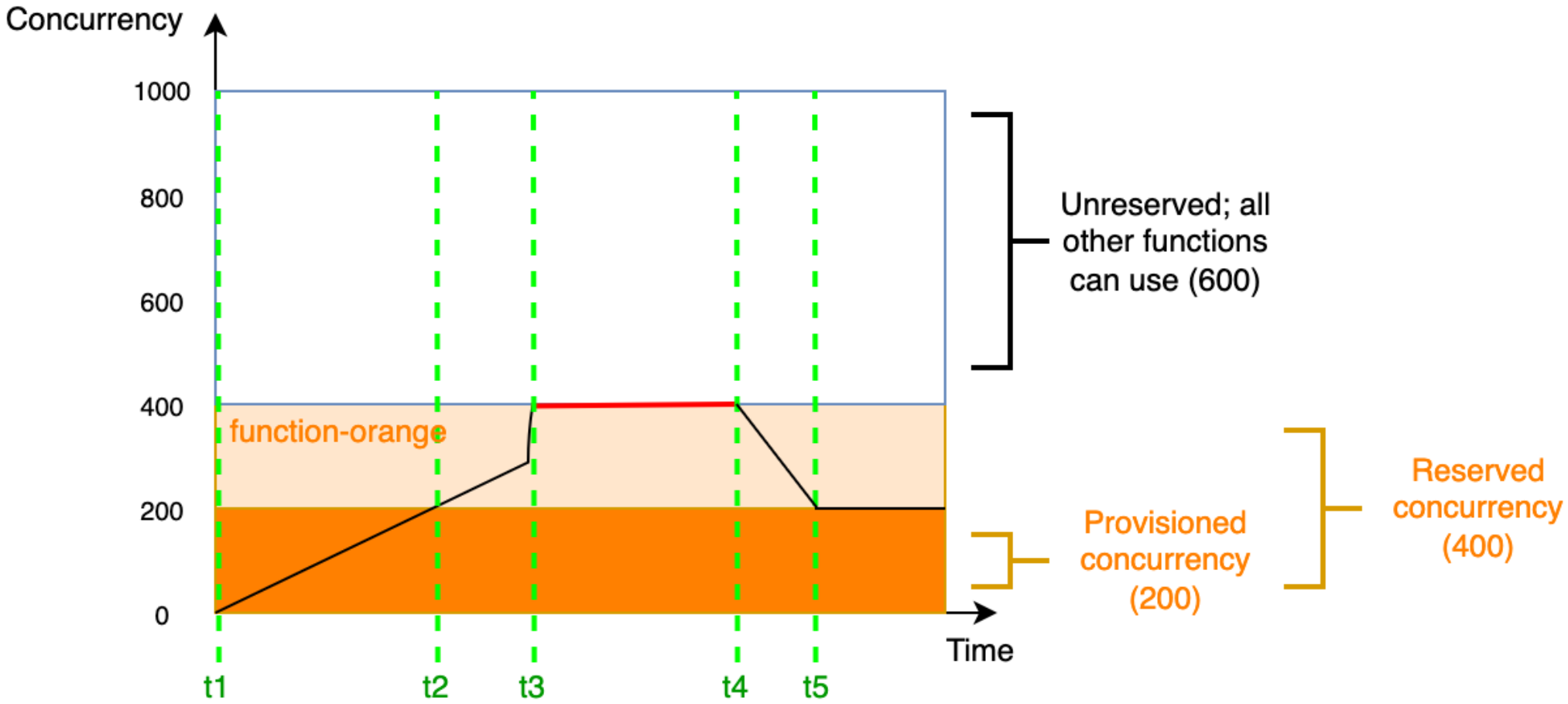
Provisioned vs. Reserved Concurrency

Spaß mit **Concurrency**



Provisioned vs. Reserved Concurrency

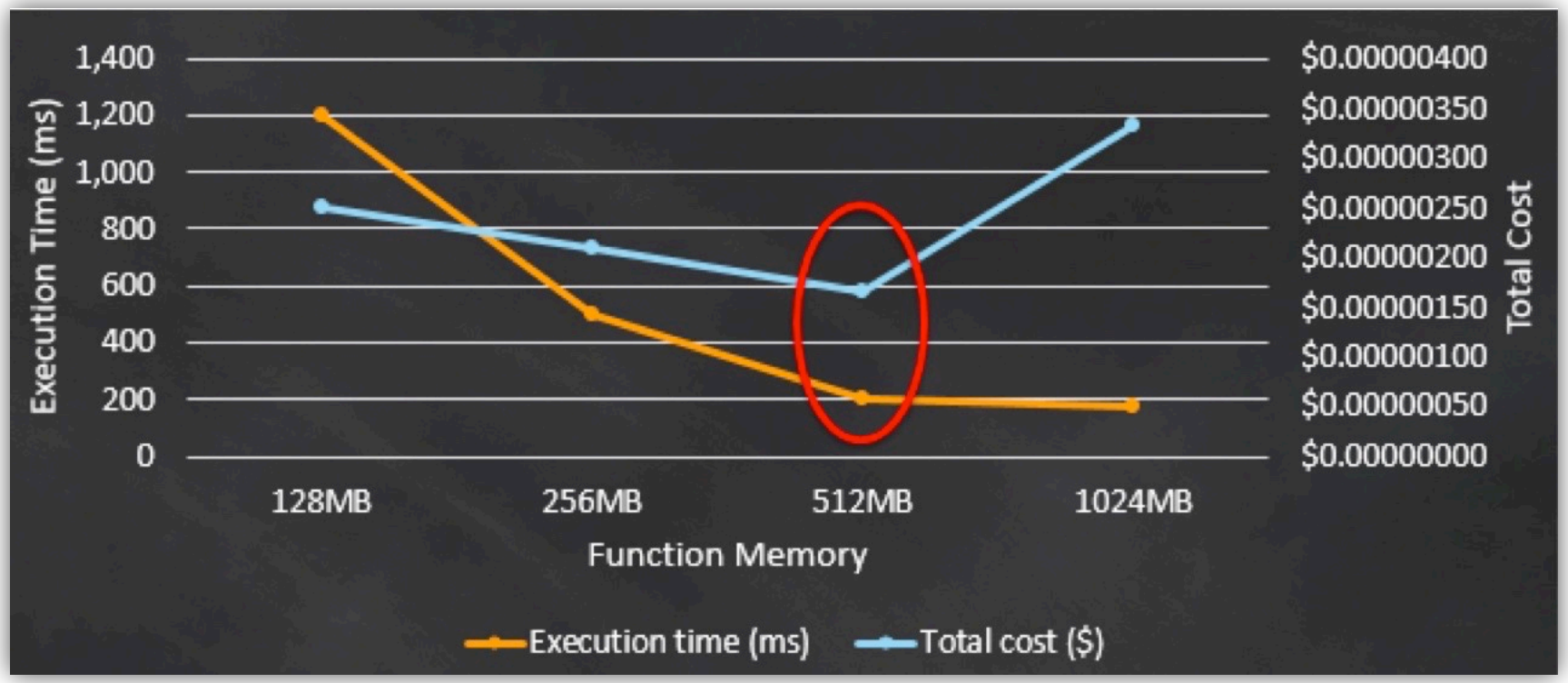
Spaß mit **Concurrency**



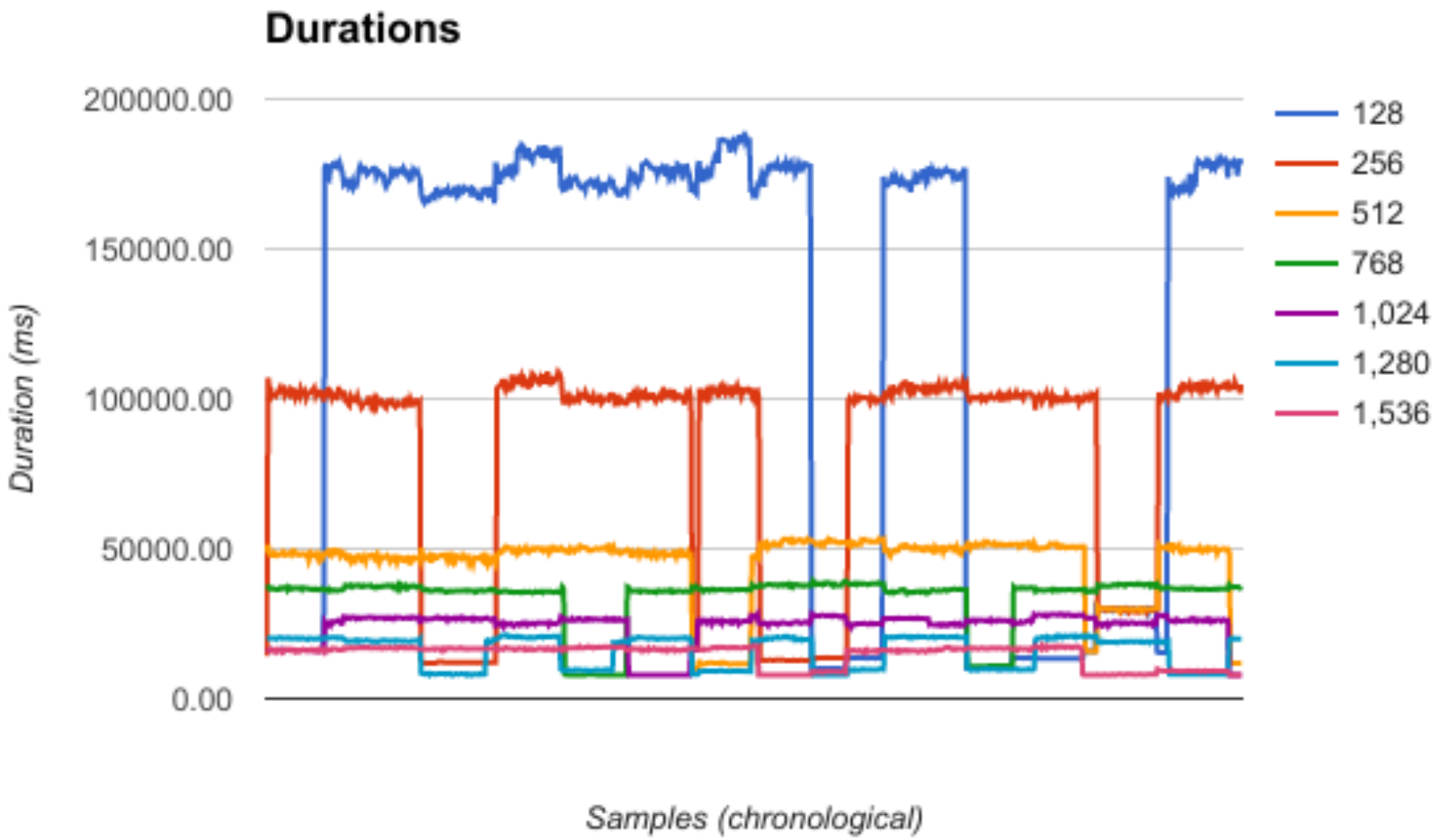
Provisioned vs. Reserved Concurrency


Spaß mit
Konfiguration

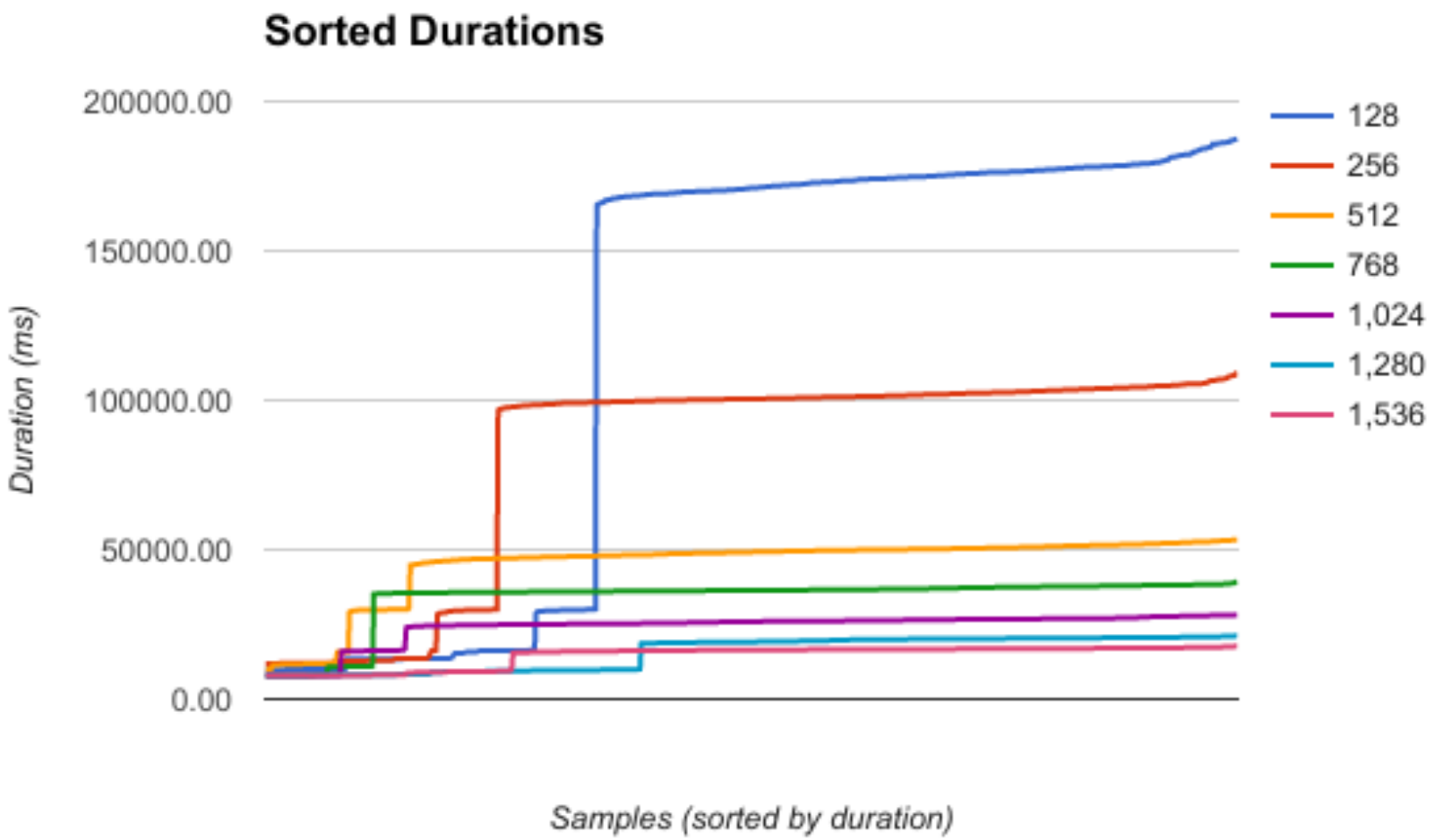
Spaß mit **Konfiguration**



Spaß mit **Konfiguration**



Spaß mit **Konfiguration**

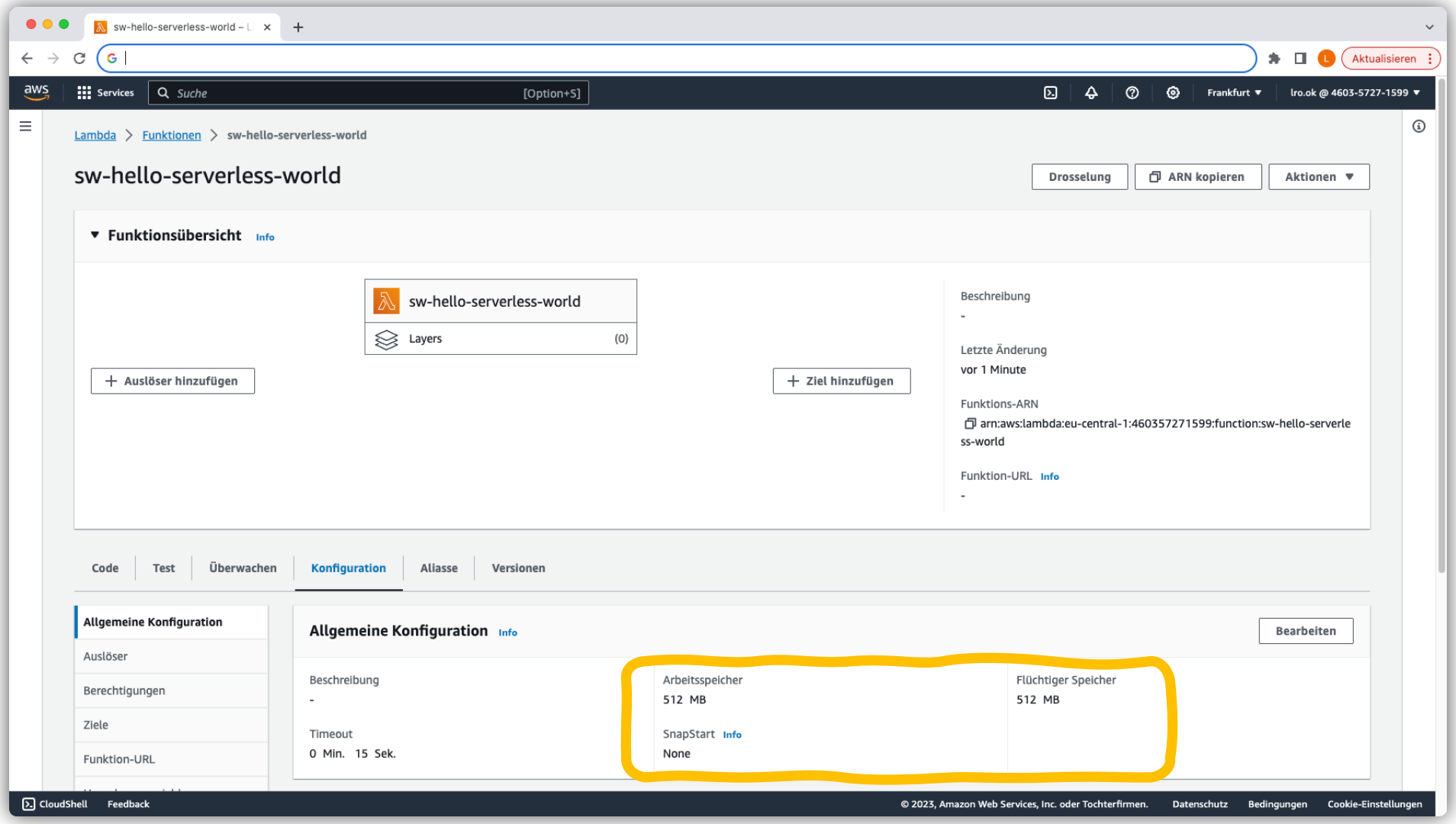


HANDS-ON

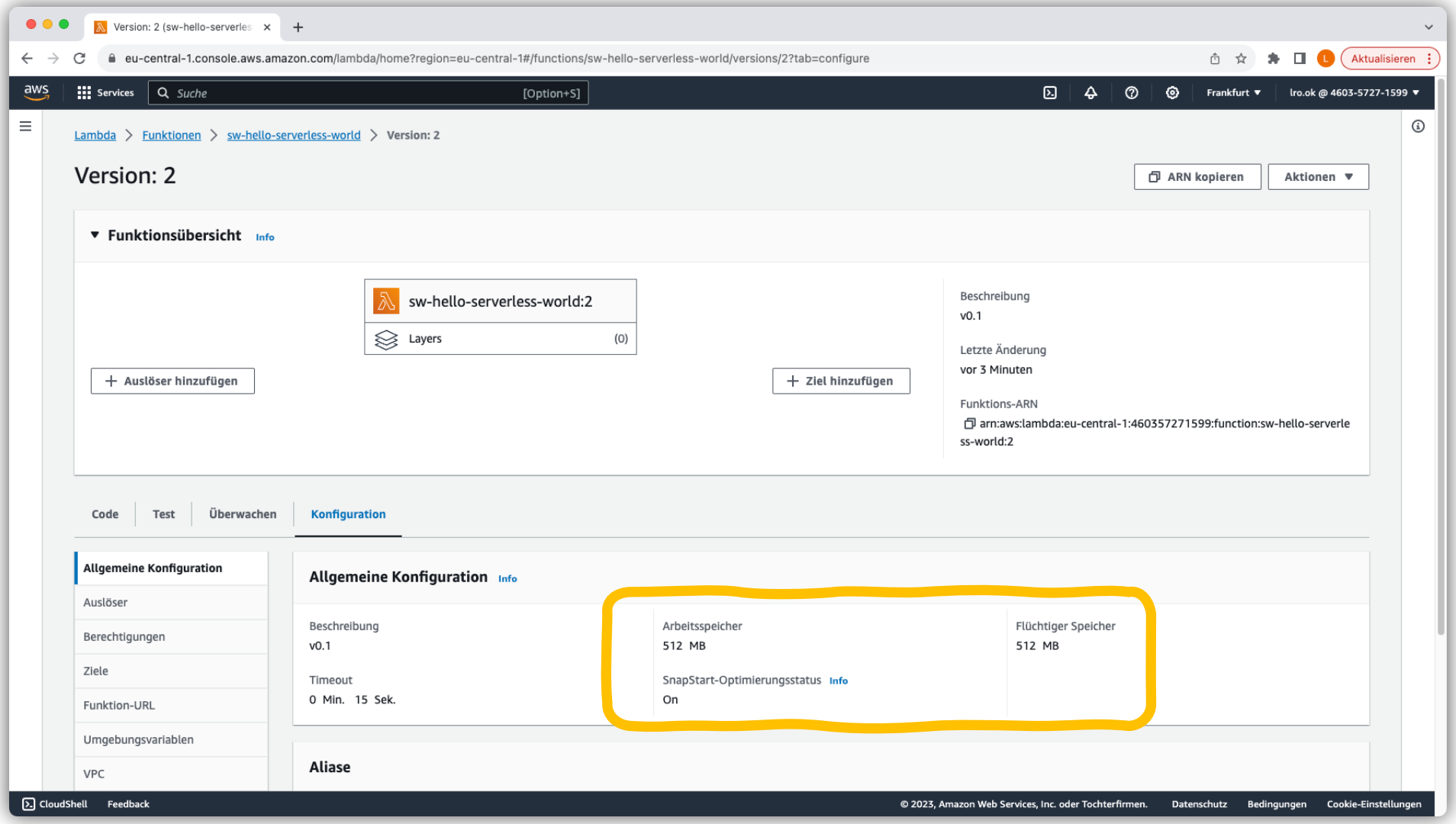
Hello Serverless Coffee V2

- Implementation
- Upload
- Testing

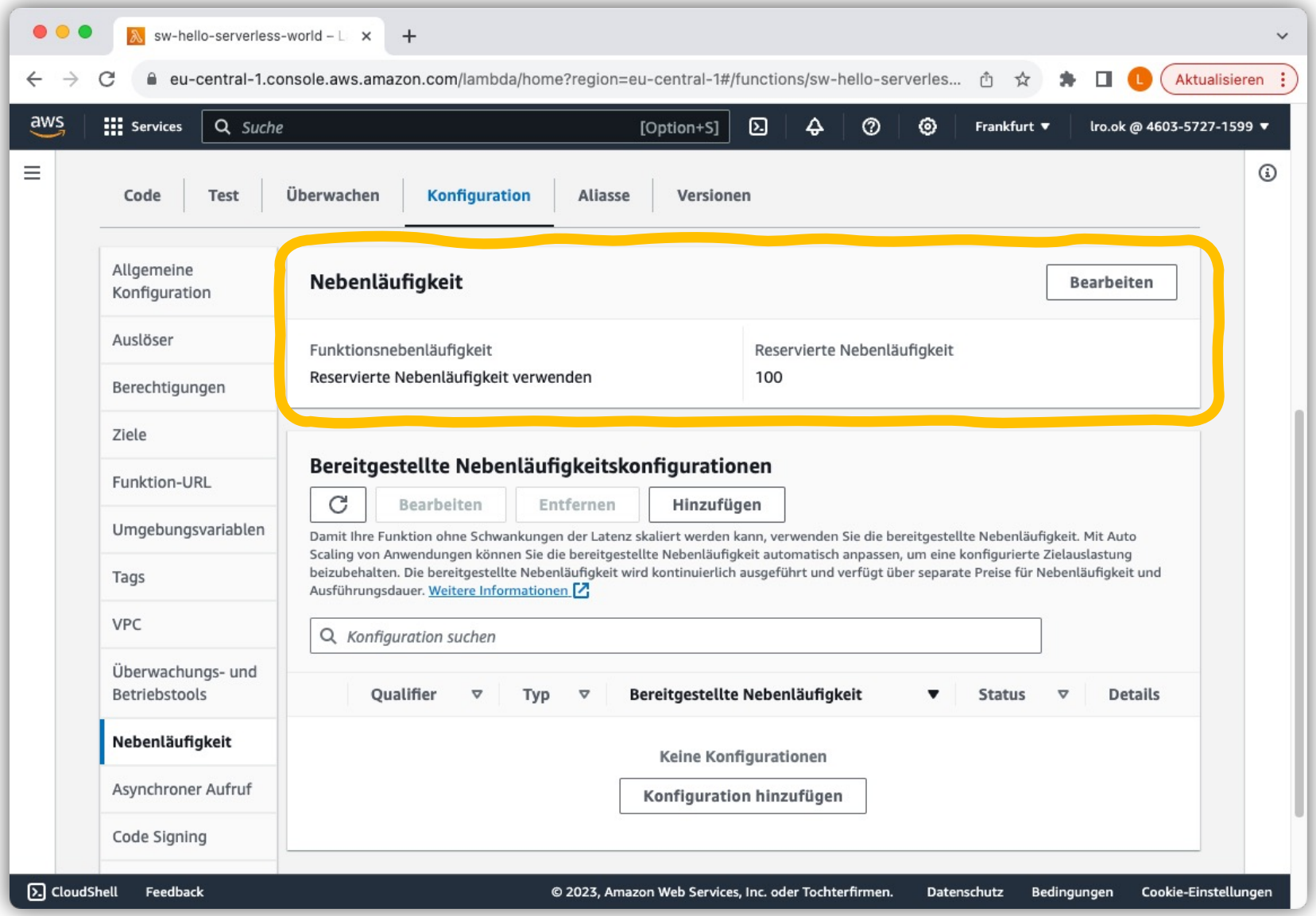
SnapStart in Action



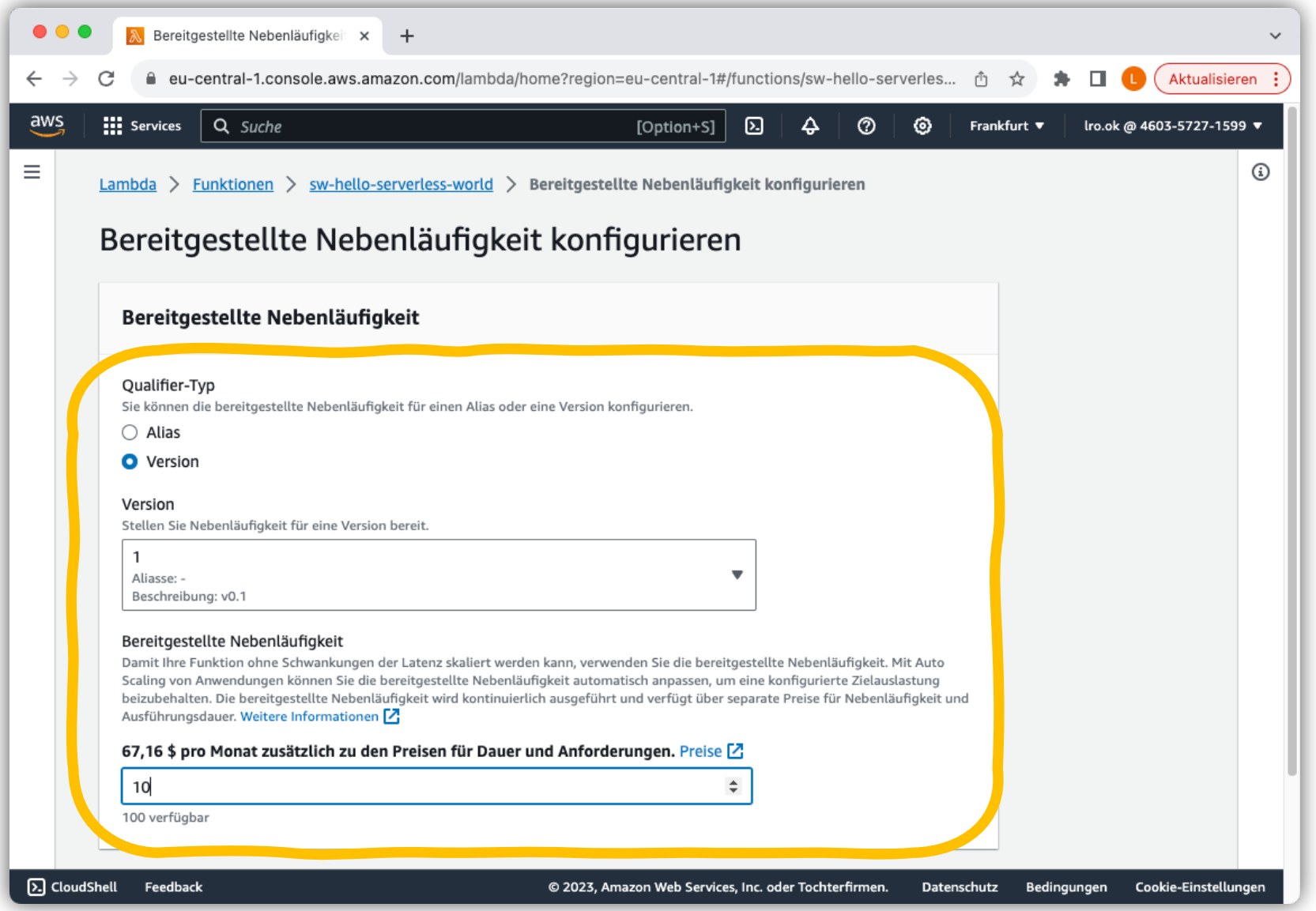
SnapStart in Action



Provisioned Concurrency in Action



Provisioned Concurrency in Action



Lessons Learned:

Hello Serverless

HELLO SERVERLESS

Lessons Learned (so far)

- **Business-Code** vs Infrastruktur-Code
- **Least Privilege** Execution Role
- **Cold Start** beim ersten Call (langsam/teuer)*
- **CloudWatch** ist dein Freund
- **Konfiguration** ist kontextsensitiv

(*SnapStart ODER Provisioned Concurrency als mögliche Lösung)


Initial
Set-up

SETUP – das **WAS**

Was gilt es vorab einzurichten?

- Tabellen
- Event-Bus
- Rollen & Rechte

SETUP – das **WIE**

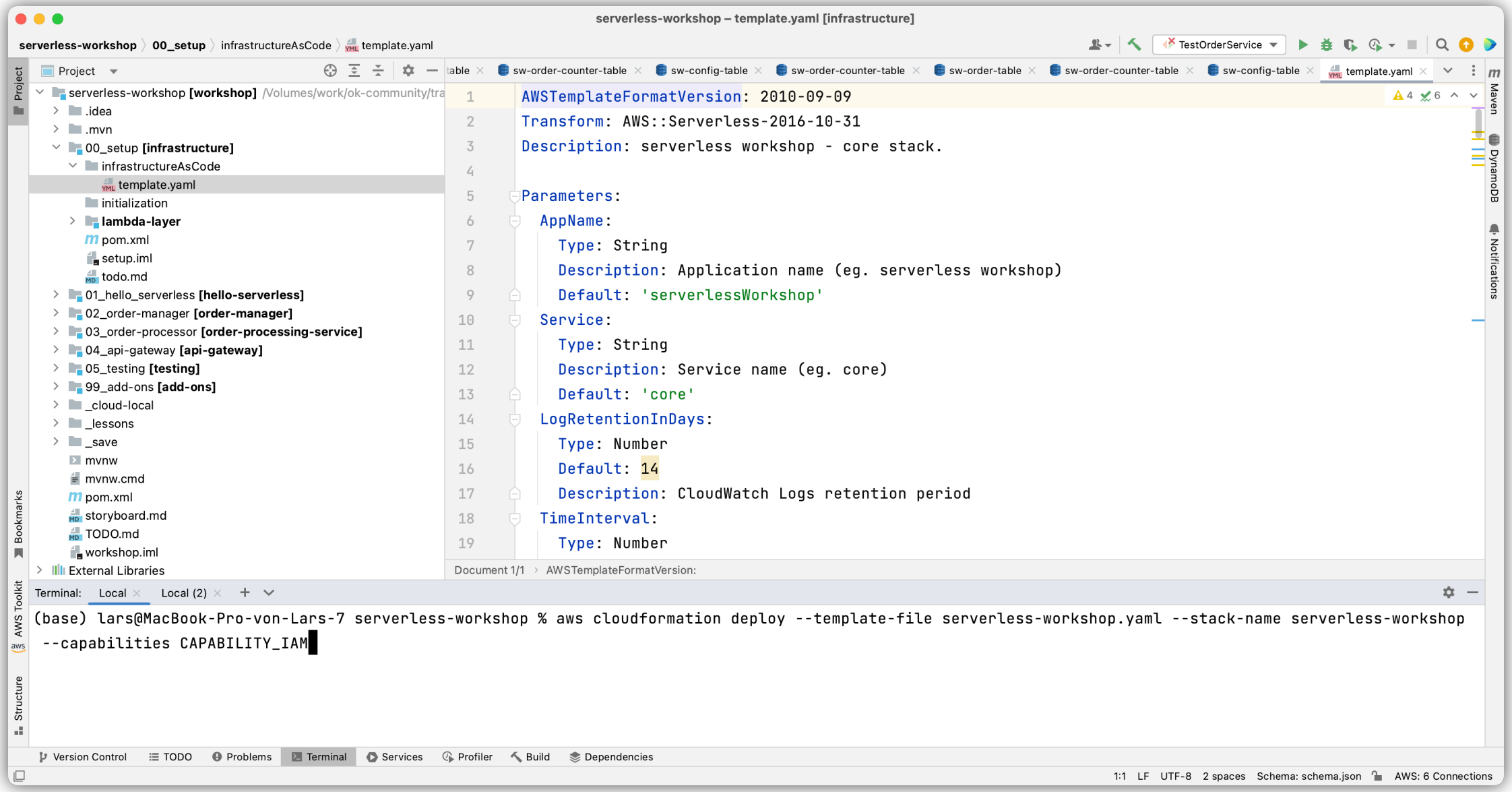
Wie gilt es vorab einzurichten?

1. CloudFormation Template einspielen
2. Initiale Tabellenwerte setzen

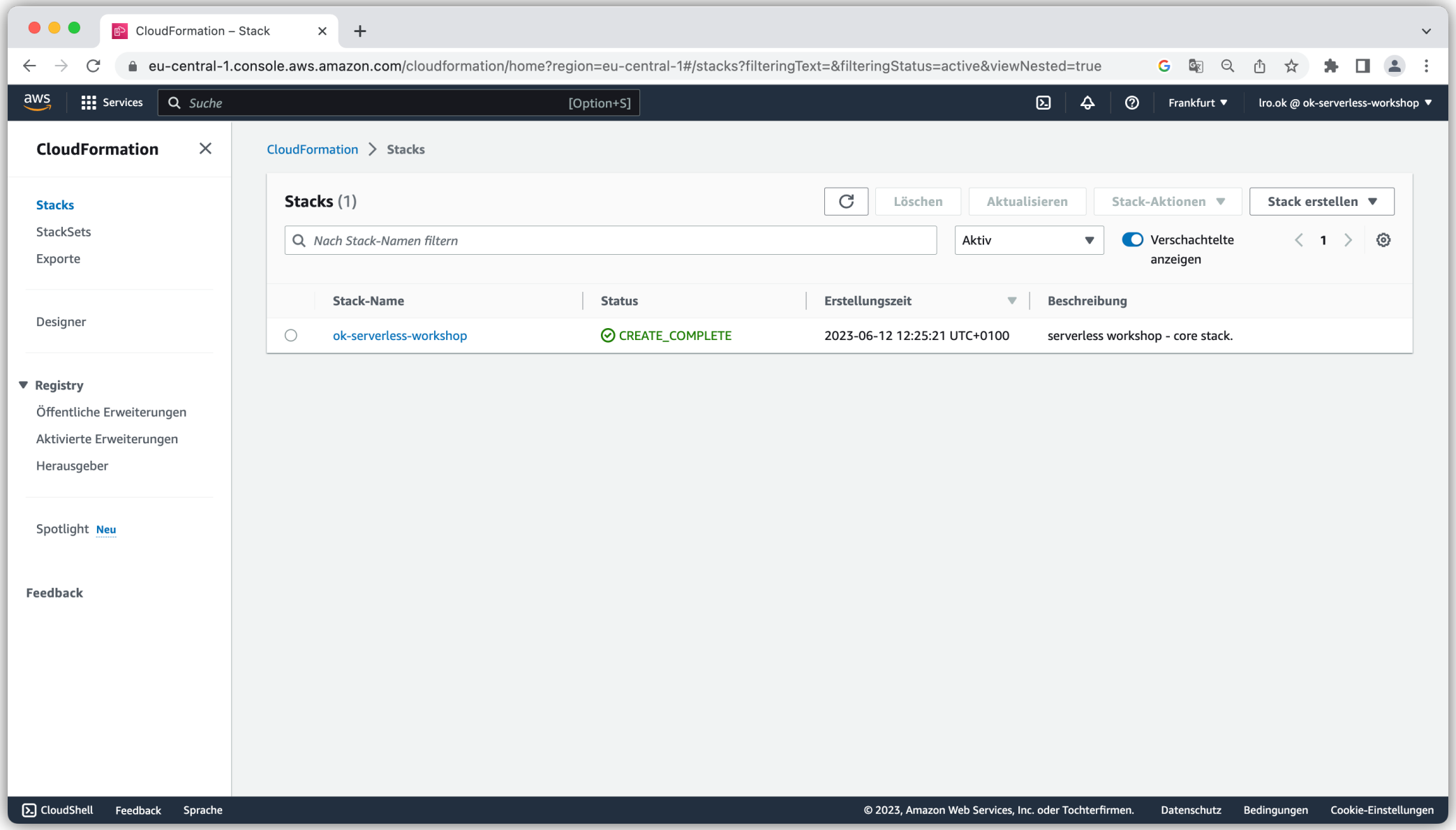
HANDS-ON

Workshop Set-up

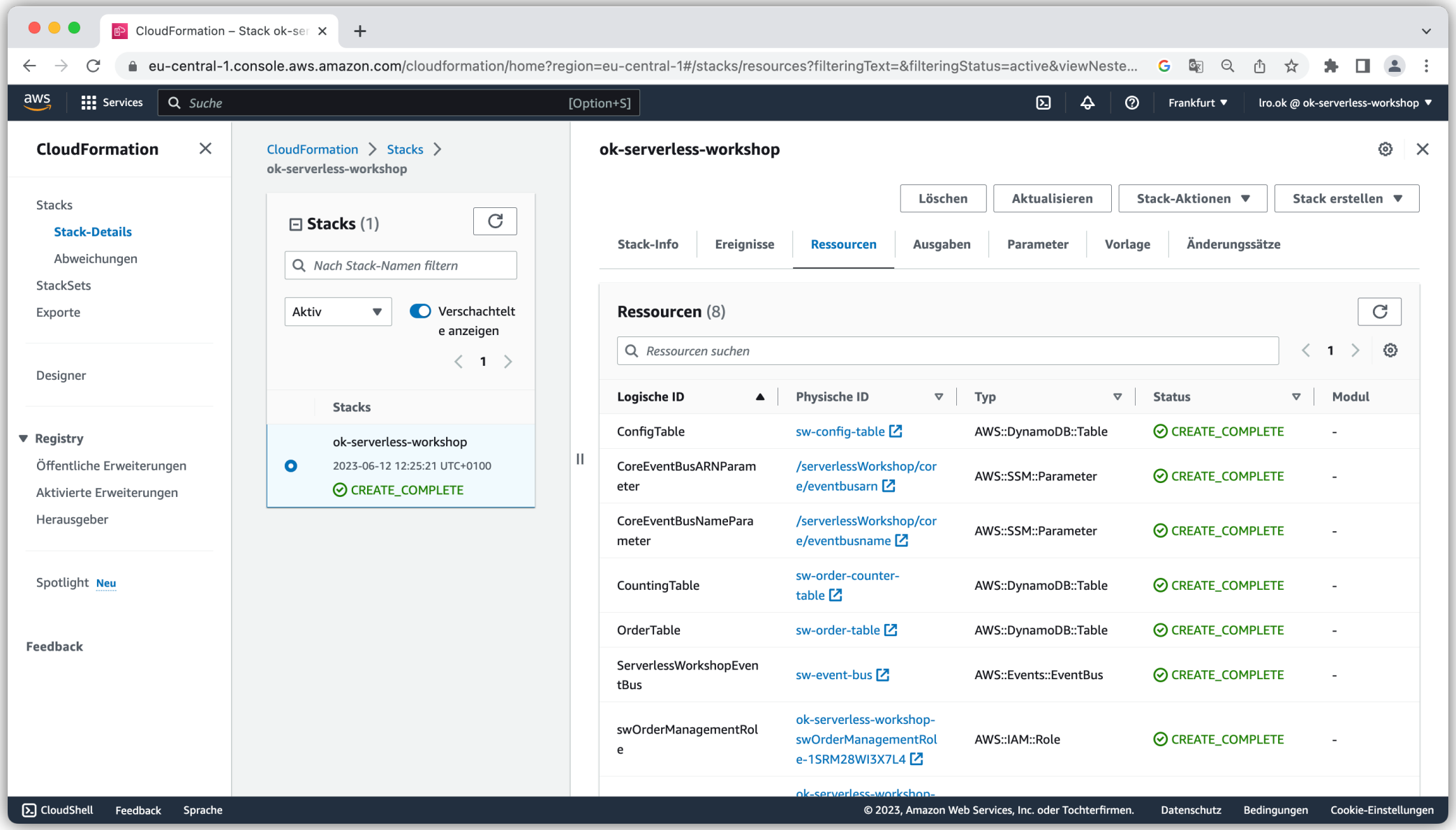
- CF Template einspielen
- Tabellen initialisieren
- Set-Up checken



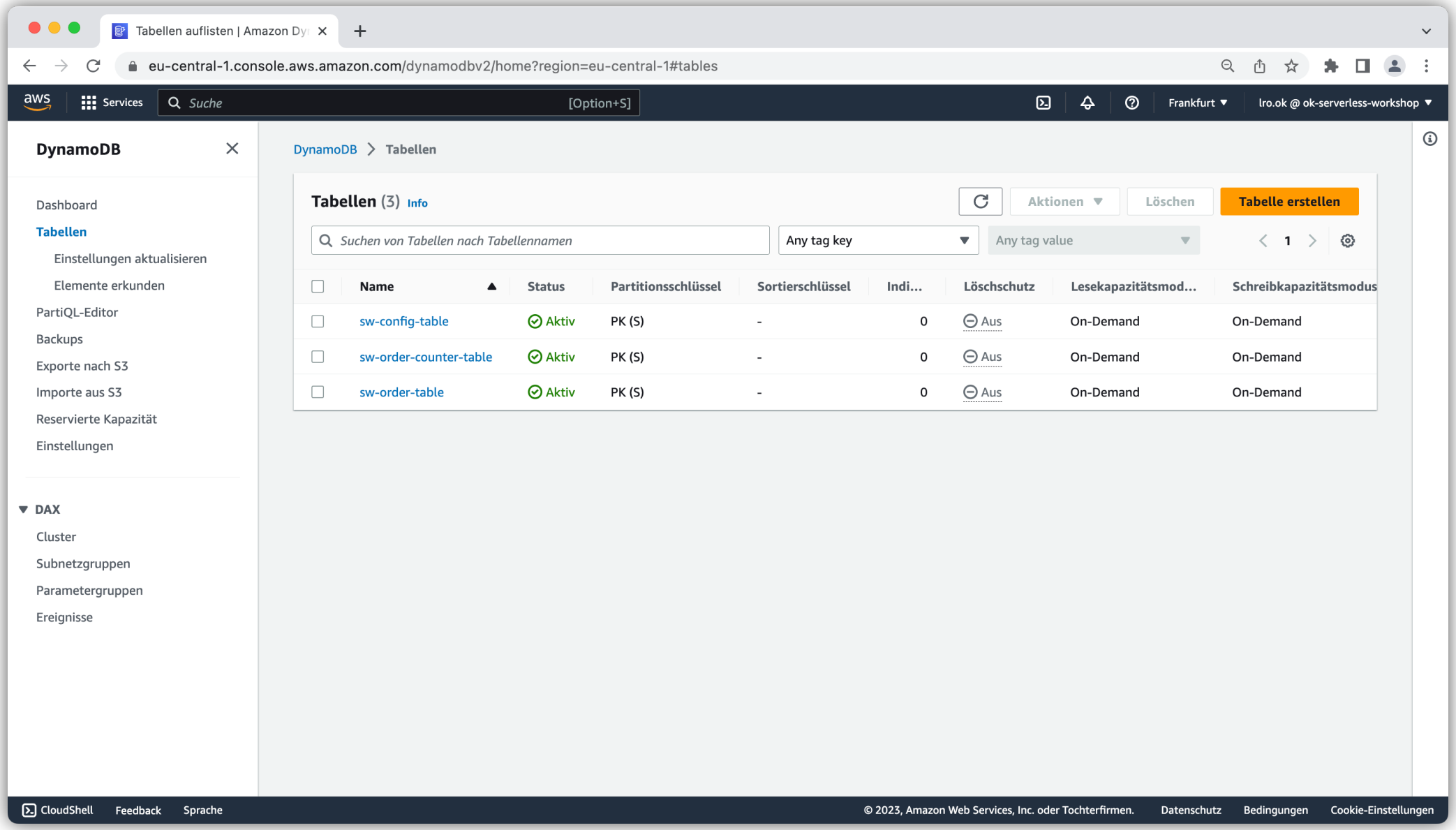
CloudFormation Template: serverless-workshop.yaml



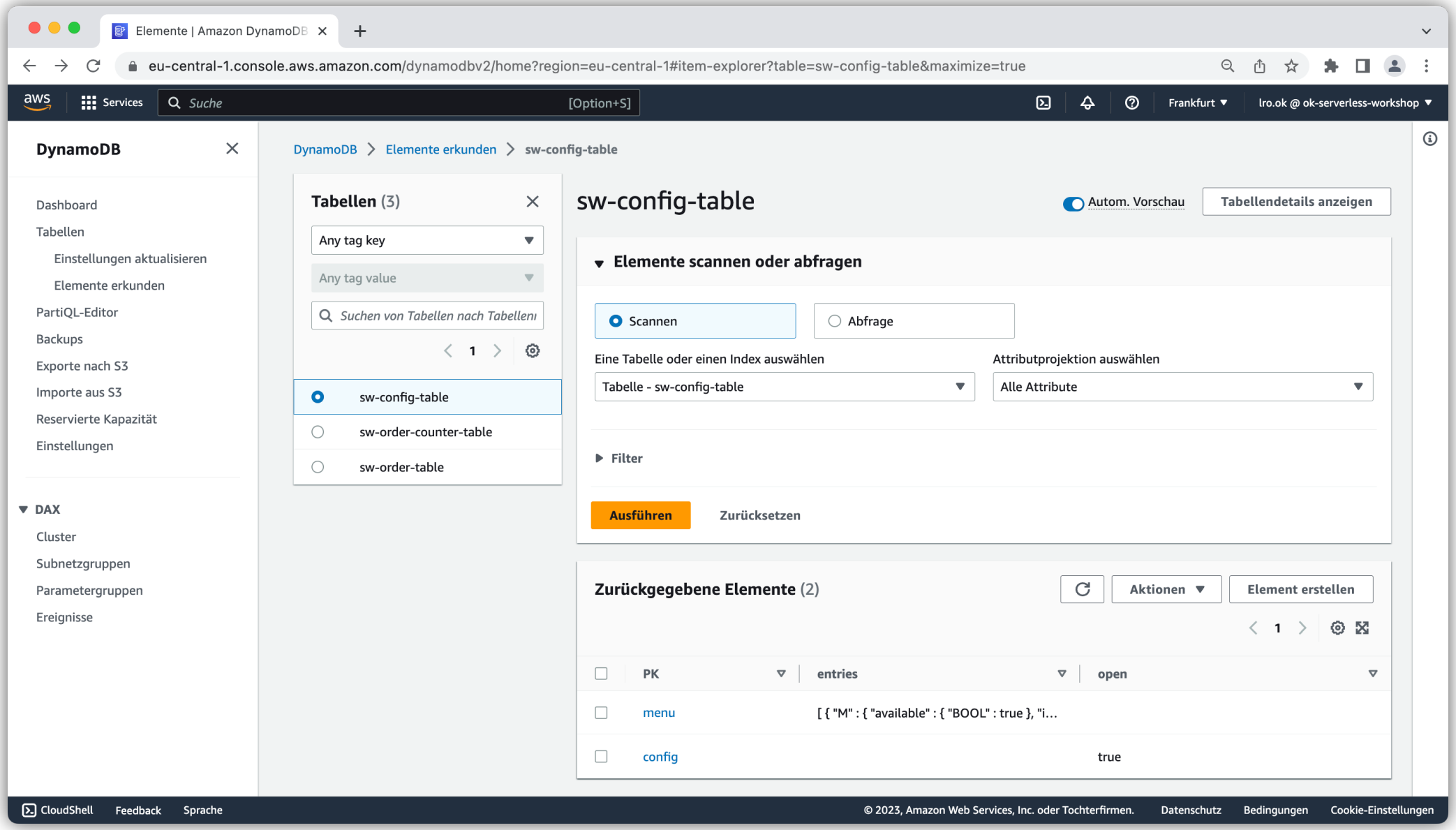
CloudFormation Service: serverless-workshop.yaml



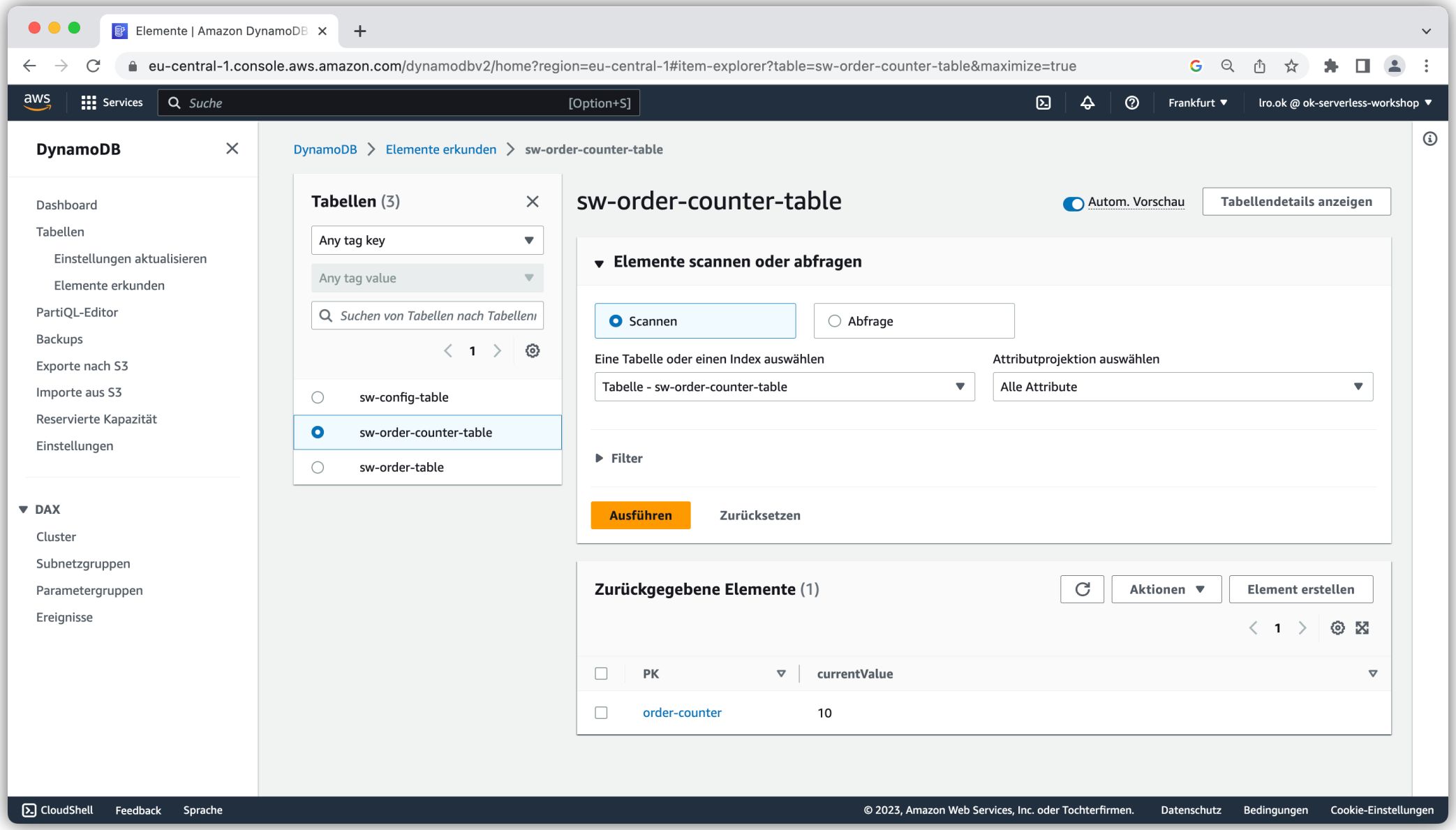
CloudFormation Service: serverless-workshop.yaml



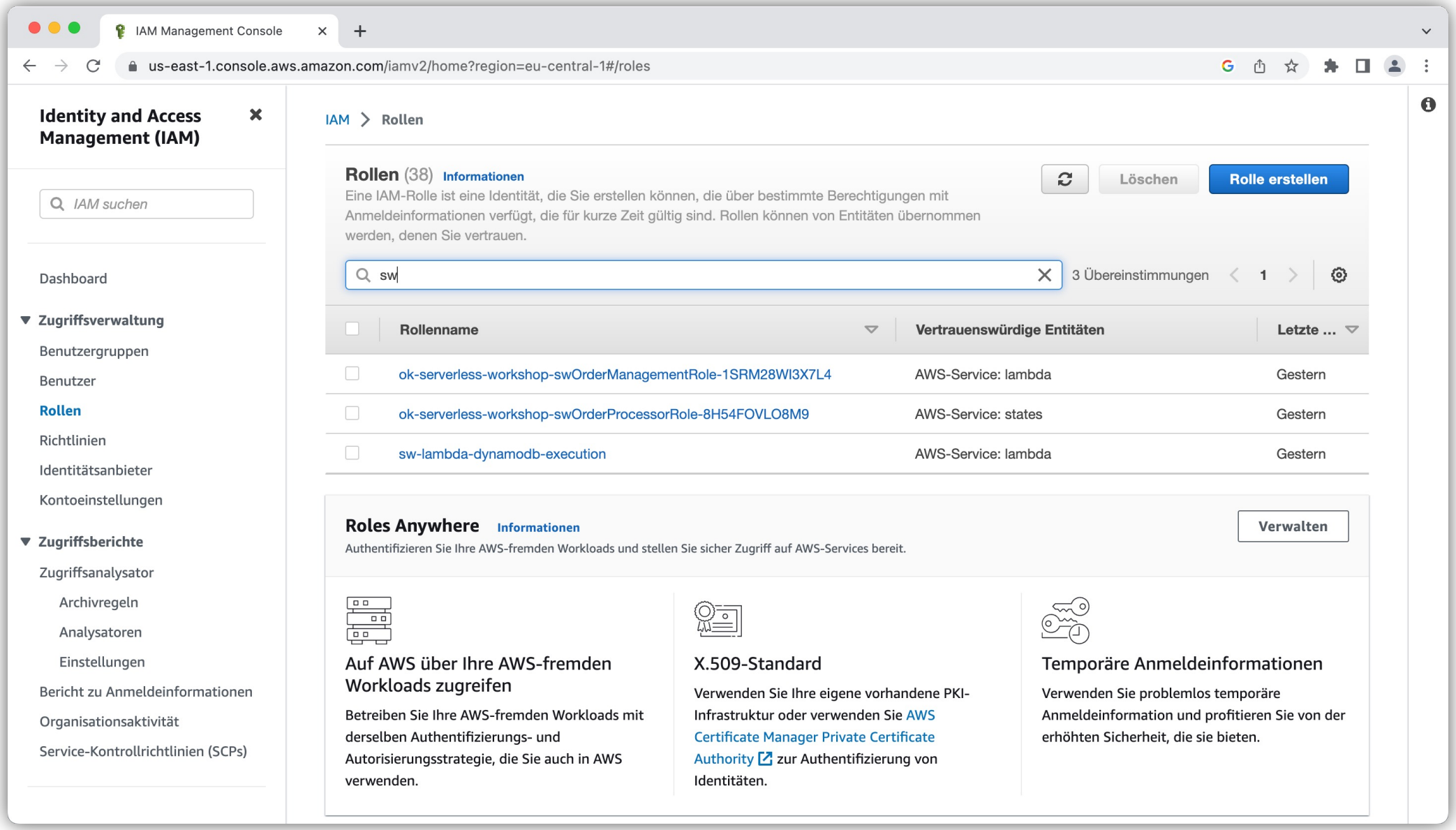
DynamoDB: Tabellenübersicht



DynamoDB: sw-config-table



DynamoDB: sw-order-counter-table



Identity & Access Management: Serverless Workshop Rollen

Agenda

Hello World & Set-up ✅

Order Management

Order Processing

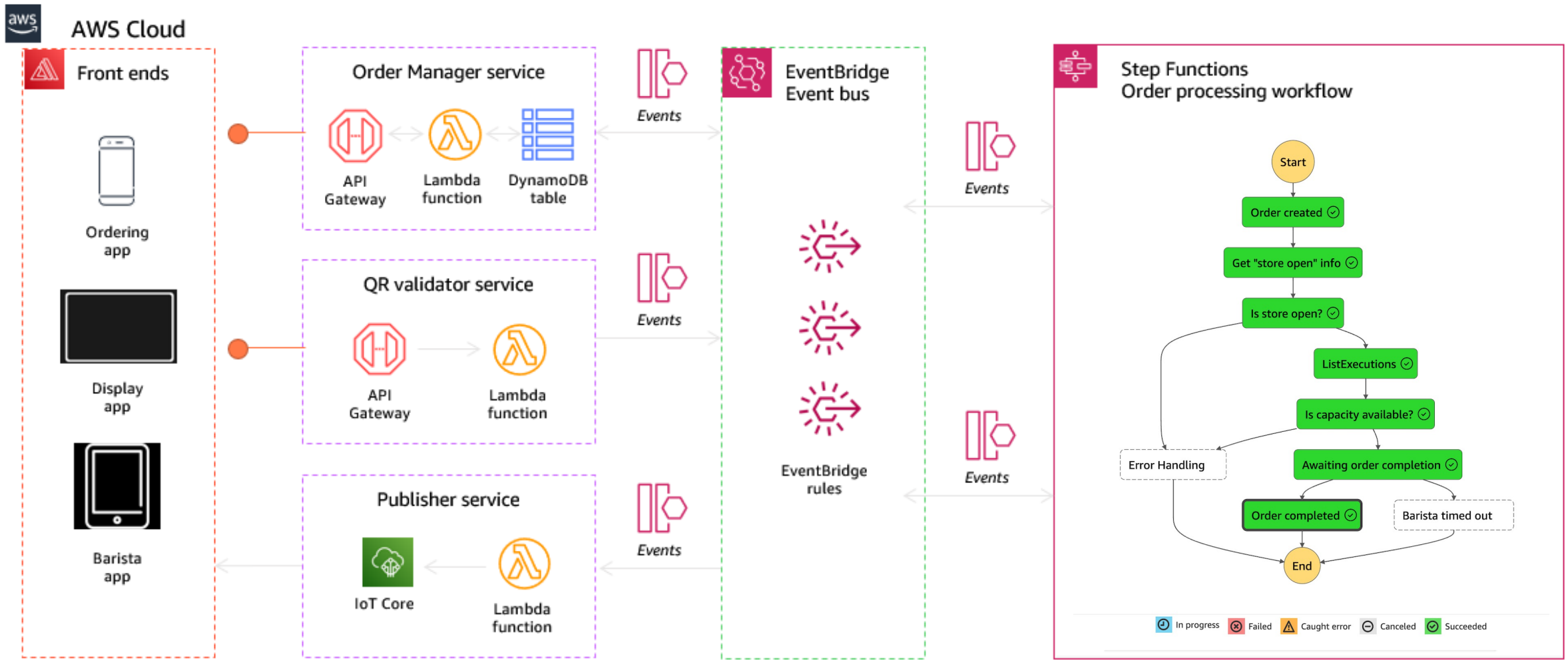
API Gateway

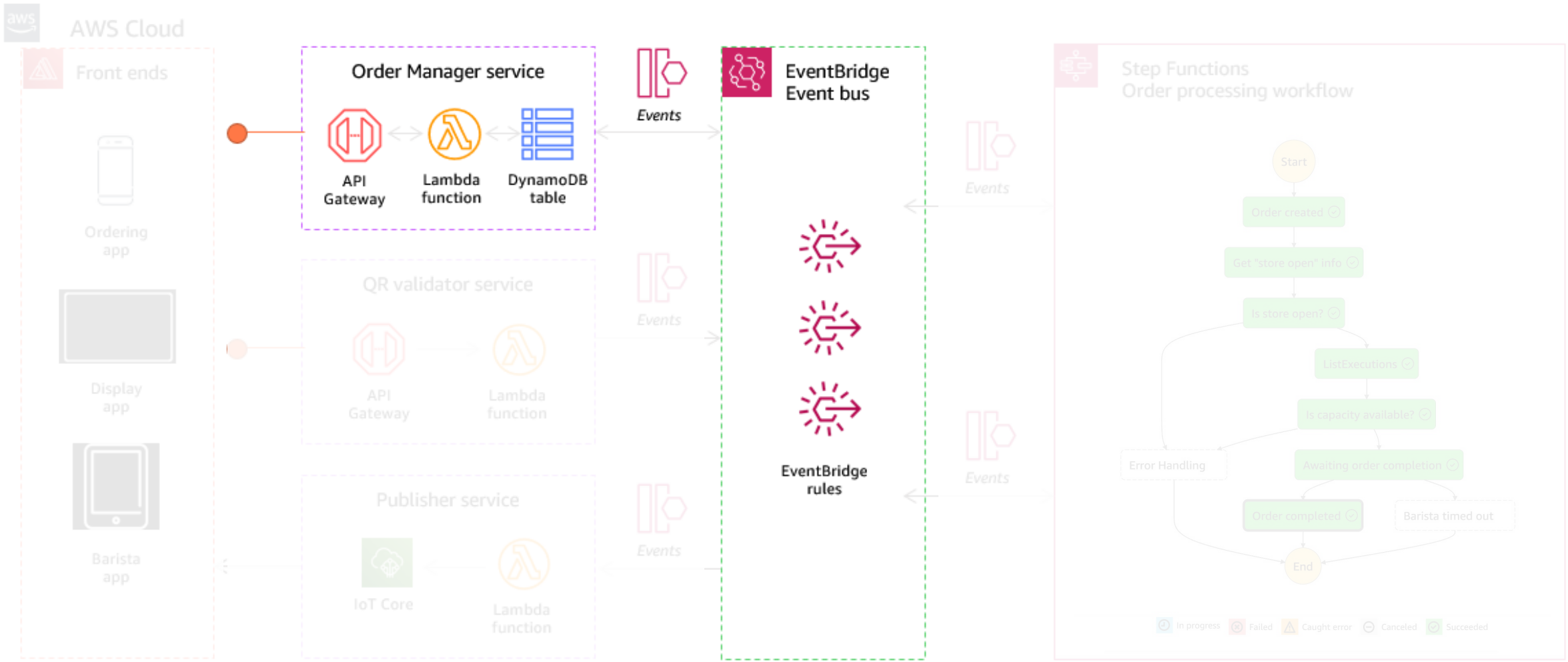
Testing

OPEN KNOWLEDGE

Serverless Coffee
Order Management

OPEN KNOWLEDGE

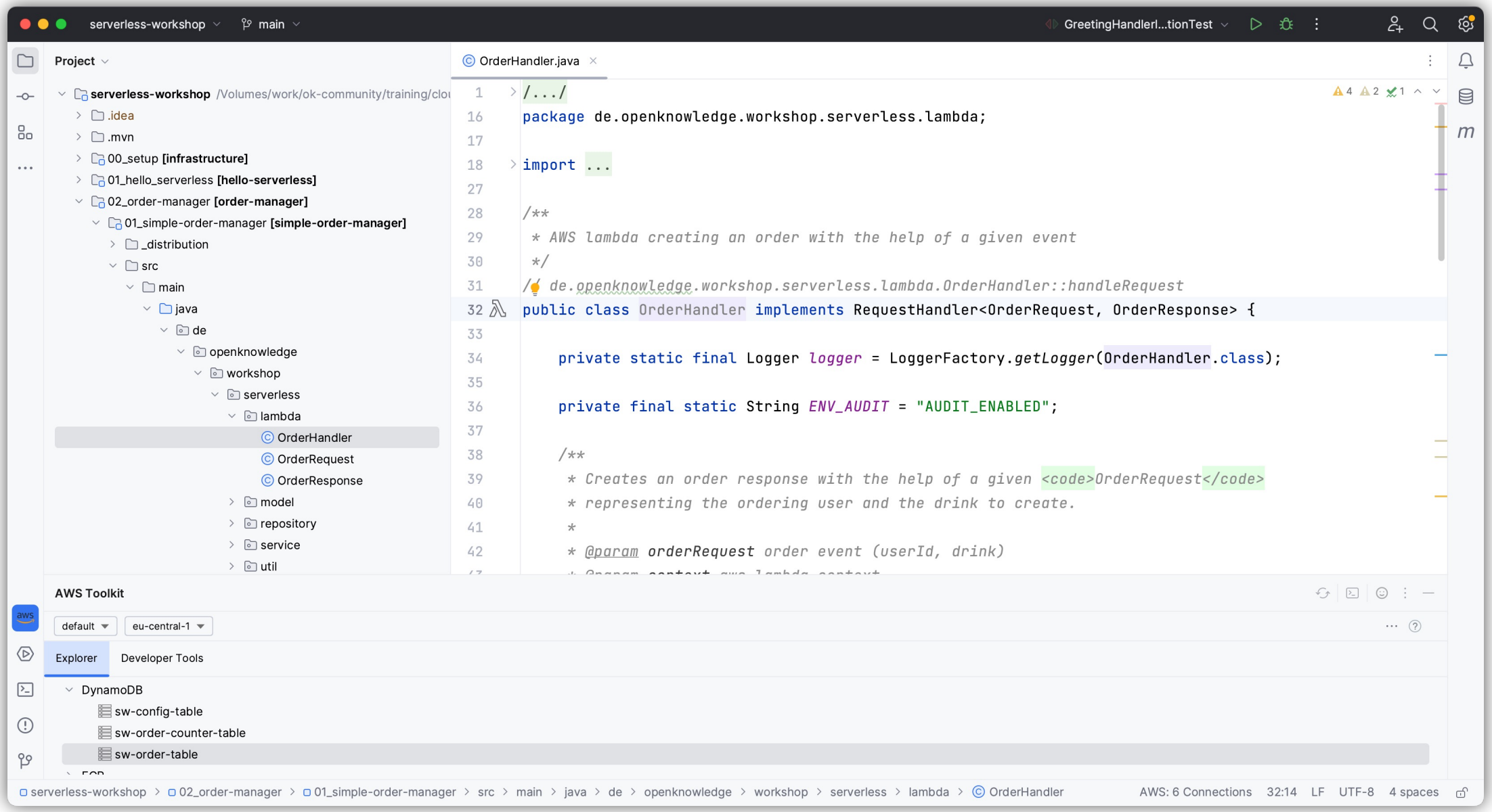

AWS Cloud
Front ends
Ordering app
Display app
Barista app
Order Manager service
API Gateway
Lambda function
DynamoDB table
QR validator service
API Gateway
Lambda function
Publisher service
IoT Core
Lambda function
Events
Events
Events
EventBridge Event bus
EventBridge rules
Events
Events
Step Functions
Order processing workflow
Start
Order created
Get "store open" info
Is store open?
ListExecutions
Is capacity available?
Error Handling
Awaiting order completion
Order completed
Barista timed out
End
In progress
Failed
Caught error
Canceled
Succeeded


AWS Cloud
Front ends
Ordering app
Display app
Barista app
Order Manager service
API Gateway
Lambda function
DynamoDB table
QR validator service
API Gateway
Lambda function
Publisher service
IoT Core
Lambda function
Events
Events
Events
Events
EventBridge Event bus
EventBridge rules
Step Functions
Order processing workflow
Start
Order created
Get "store open" info
Is store open?
ListExecutions
Is capacity available?
Error Handling
Awaiting order completion
Order completed
Barista timed out
End
In progress
Failed
Caught error
Canceled
Succeeded

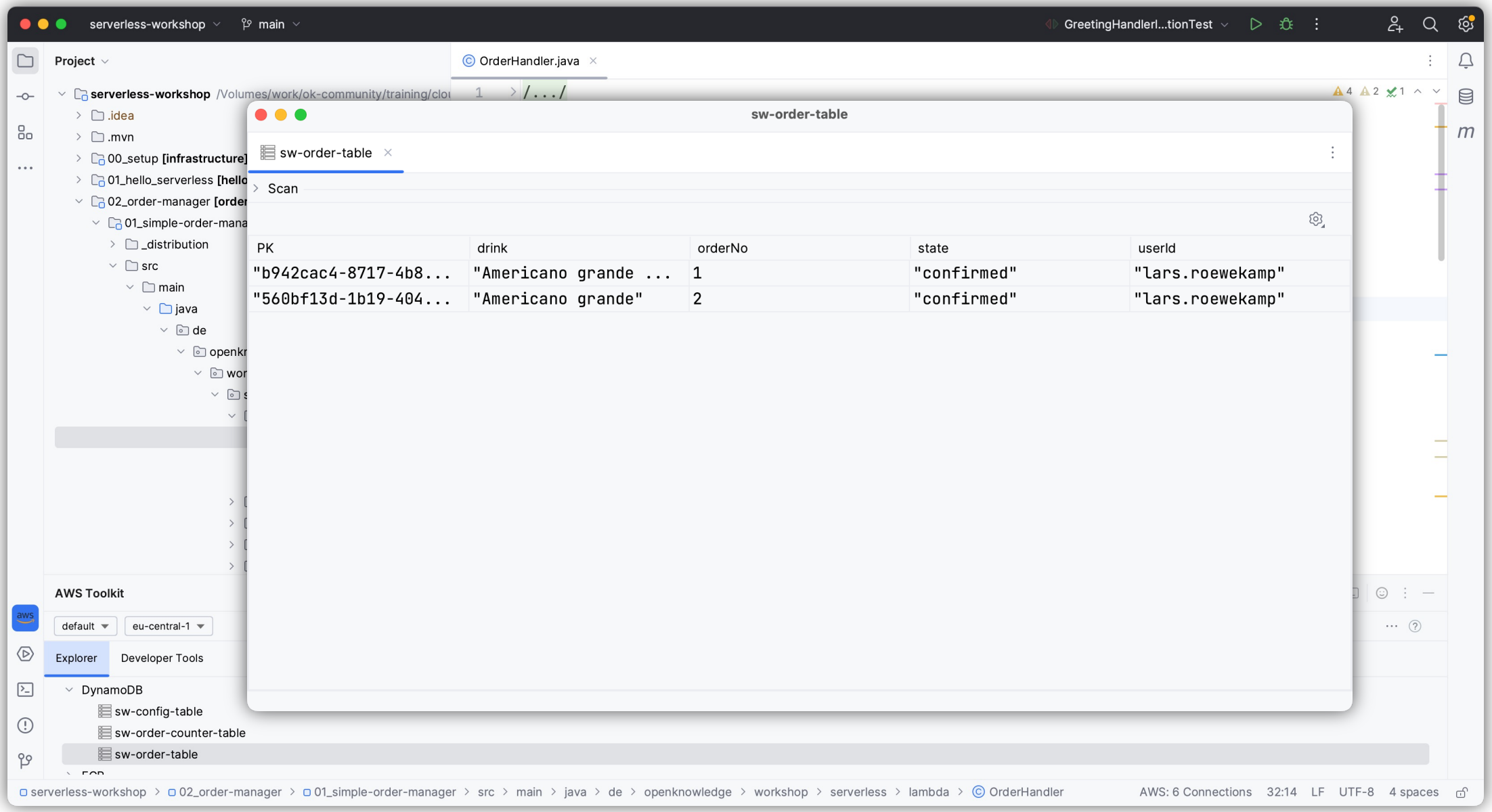
HANDS-ON

Order Managament V1

- Simple Order Handler
- dynamoDB Access



Order Management Service V1

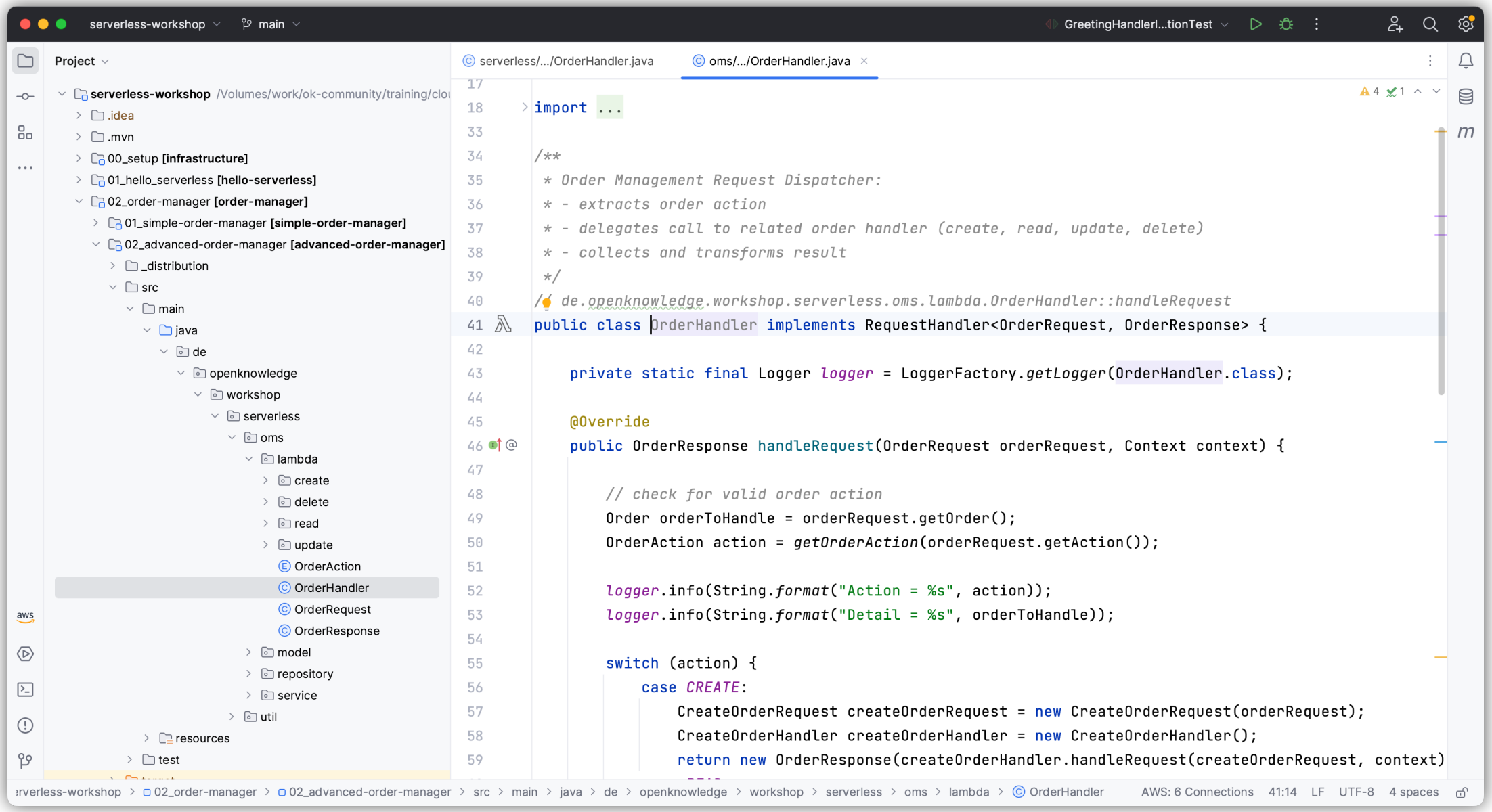


Order Management Service V1

HANDS-ON

Order Managament V2

- Advanved Order Handler
- CRUD Operations

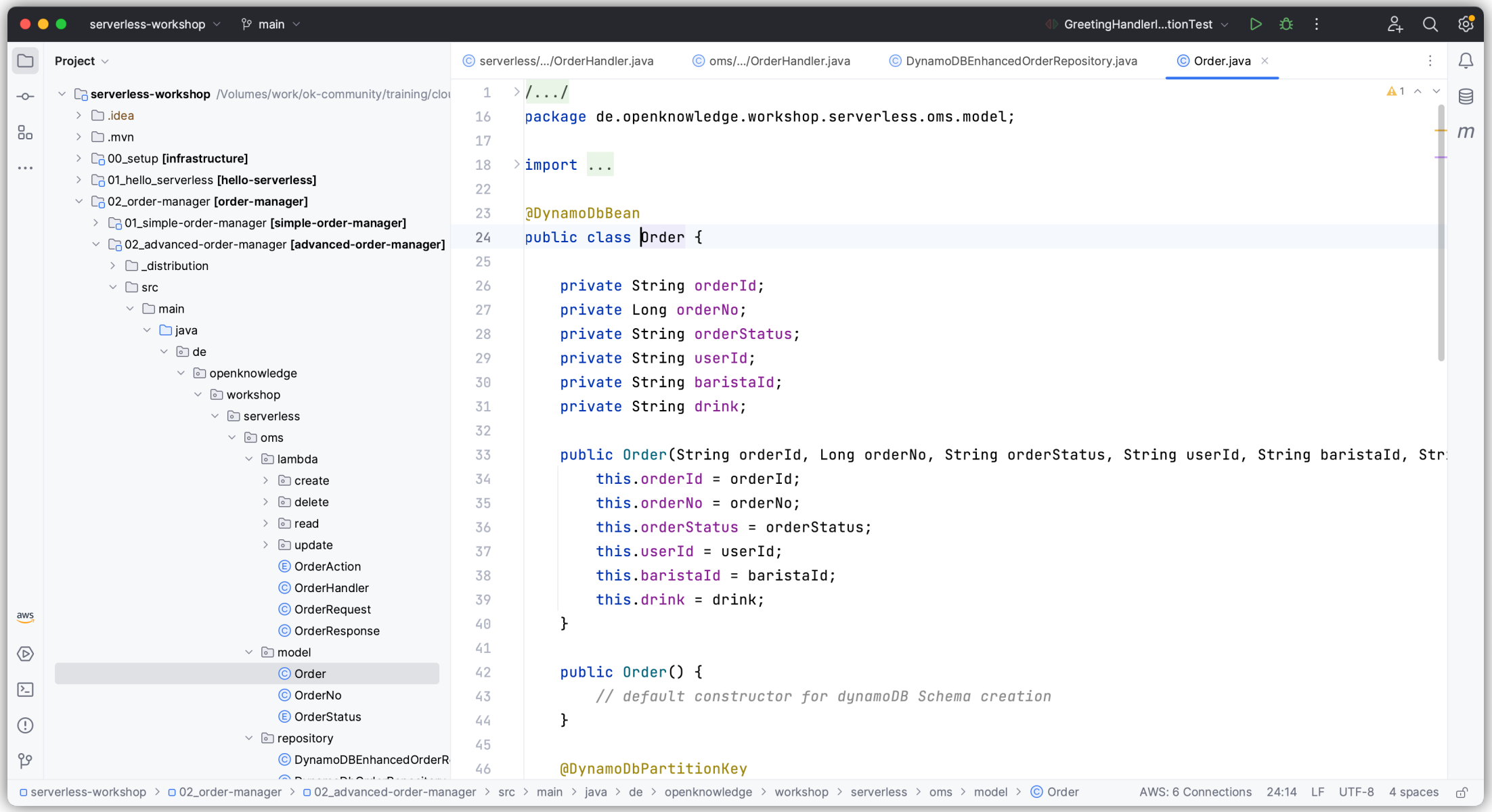


Order Management Service V2

HANDS-ON

Order Managament V3

- Enhanced Order Manager
- OR-Mapper Repository

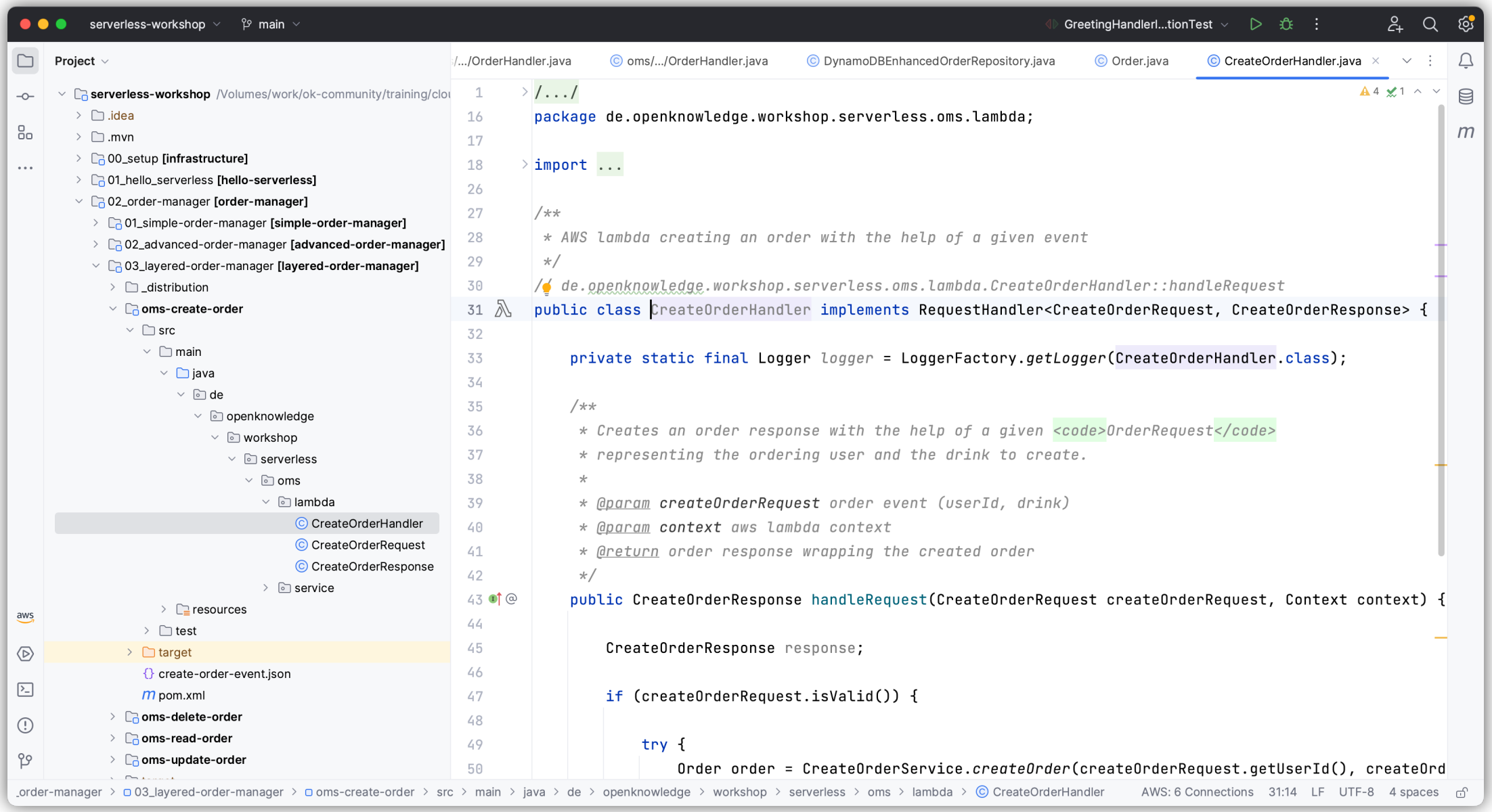


Order Management Service V3

HANDS-ON

Order Managament V4

- Single Responsibility
- Lambda Layer
- Domain Events



Order Management Service V4

Lessons Learned:

Lambdas im Zusammenspiel

LAMBDAS IM ZUSAMMENSPIEL

Lessons Learned

- **Separation** of Concerns
- **OR-Mapping** via DynamoDB Annotations
- **Lambda Layer** für die Wiederverwendung
- **Domain Events** via EventBus zur Kommunikation

- **Hobby-Testing** via AWS CLI

Agenda

Hello World & Set-up ✅

Order Management ✅

Order Processing

API Gateway

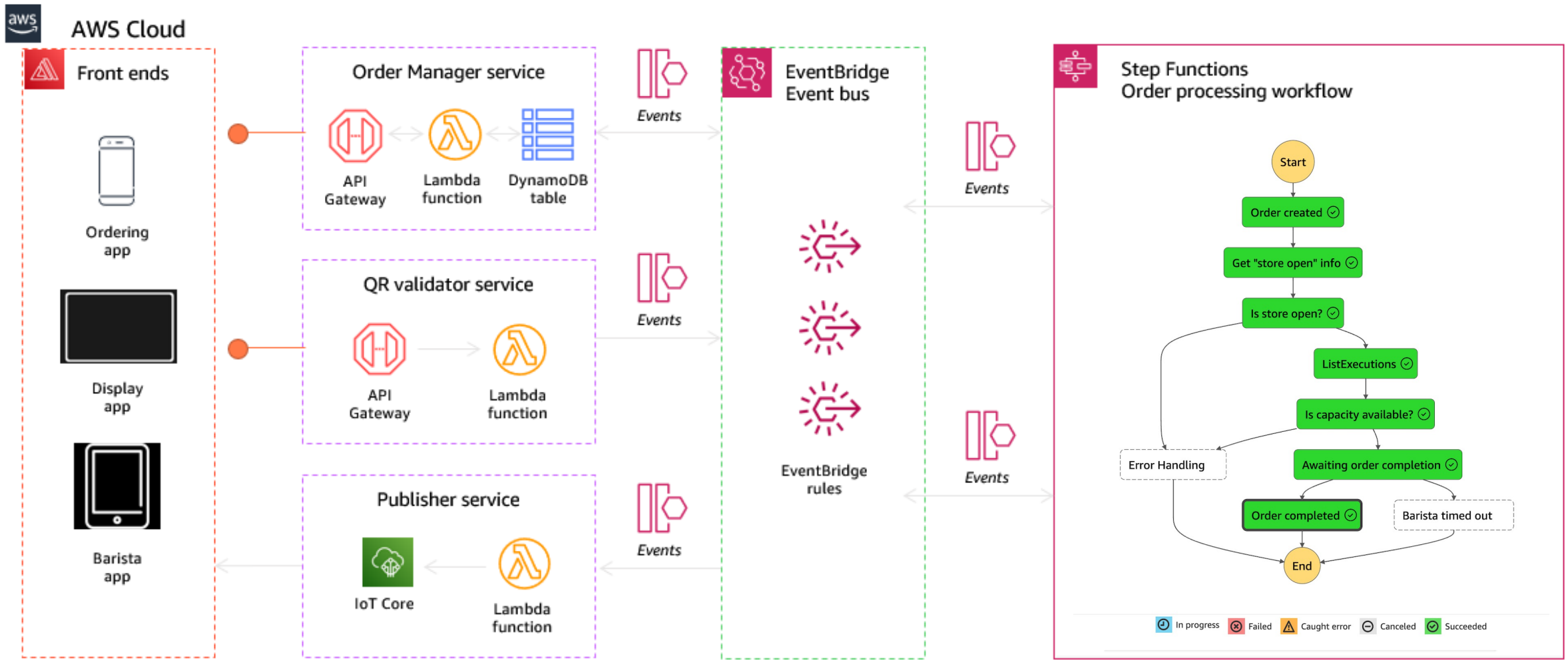
Testing

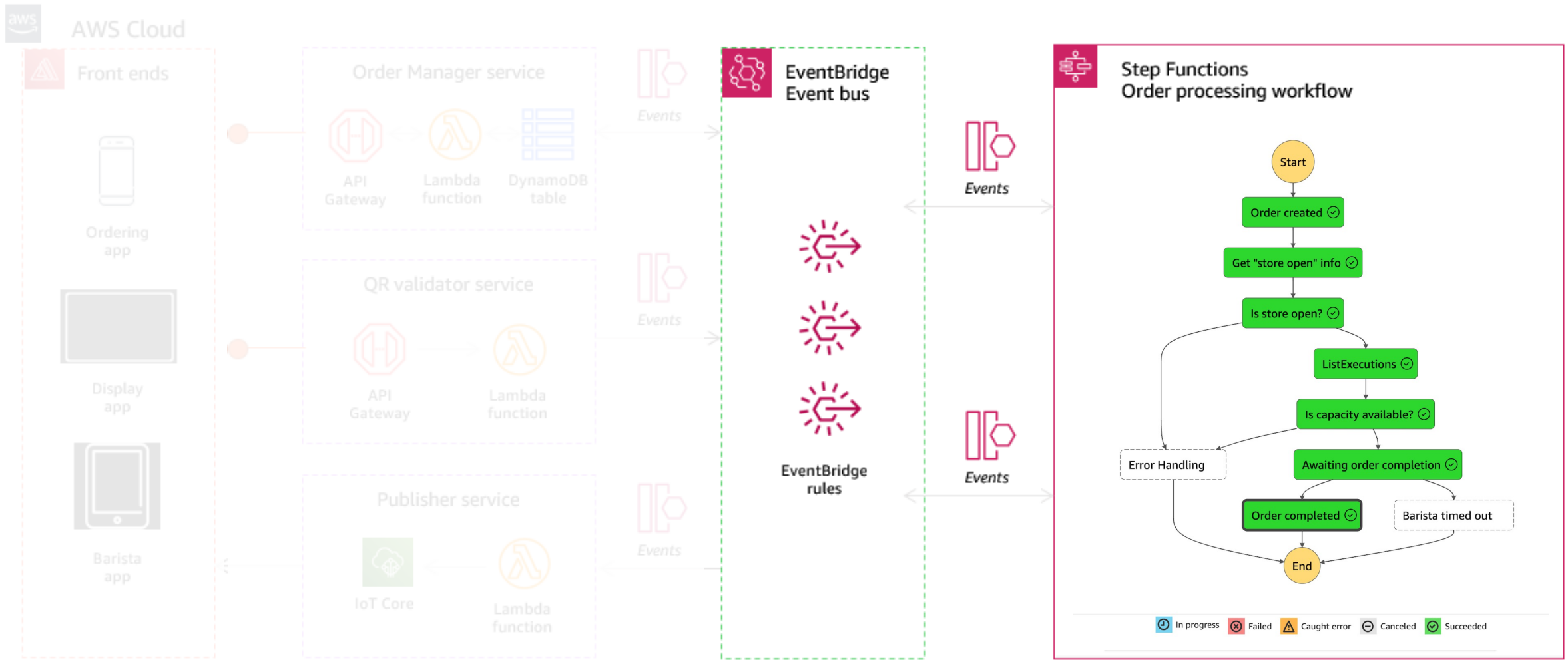
OPEN KNOWLEDGE

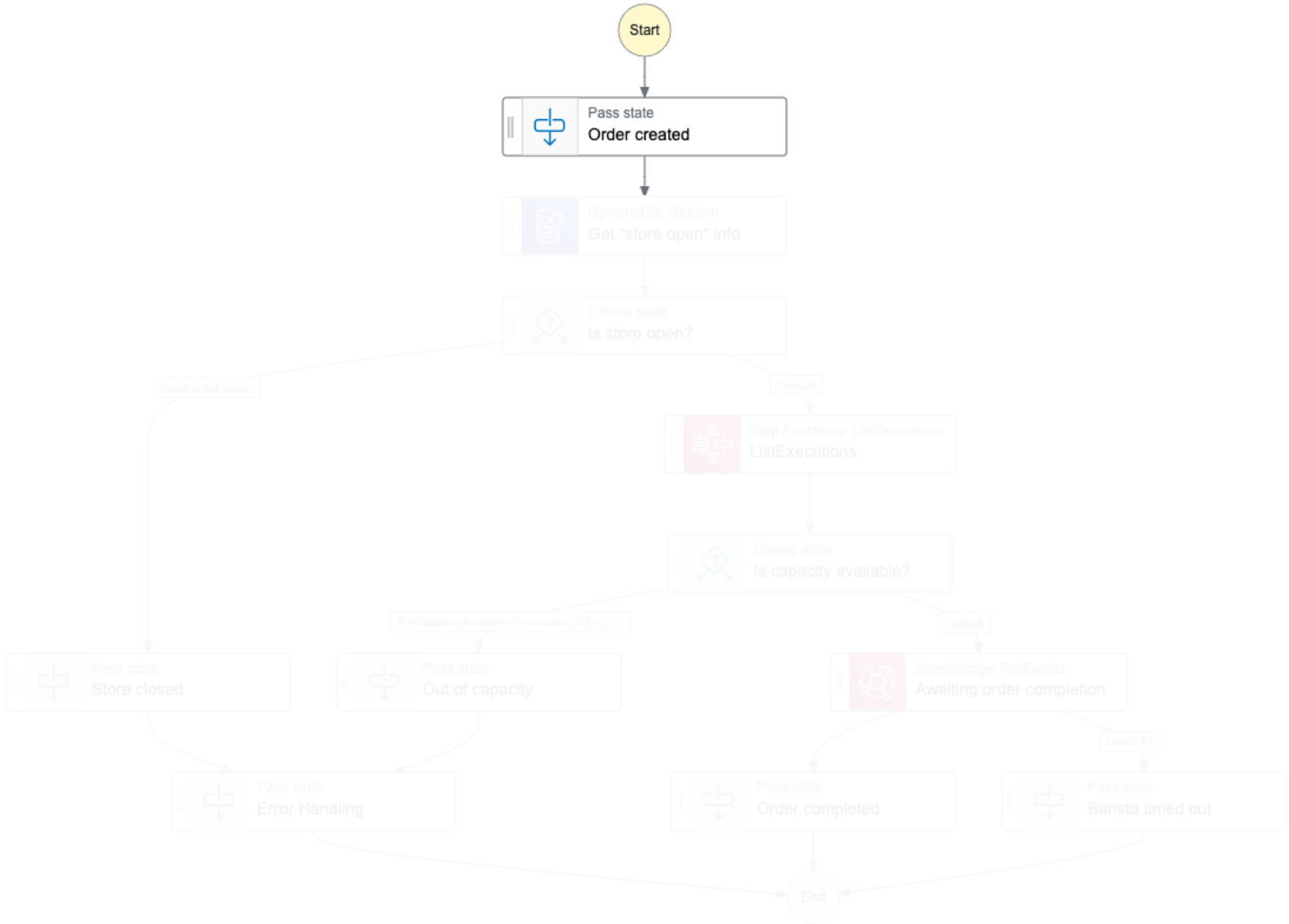
Serverless Coffee

Order Processing

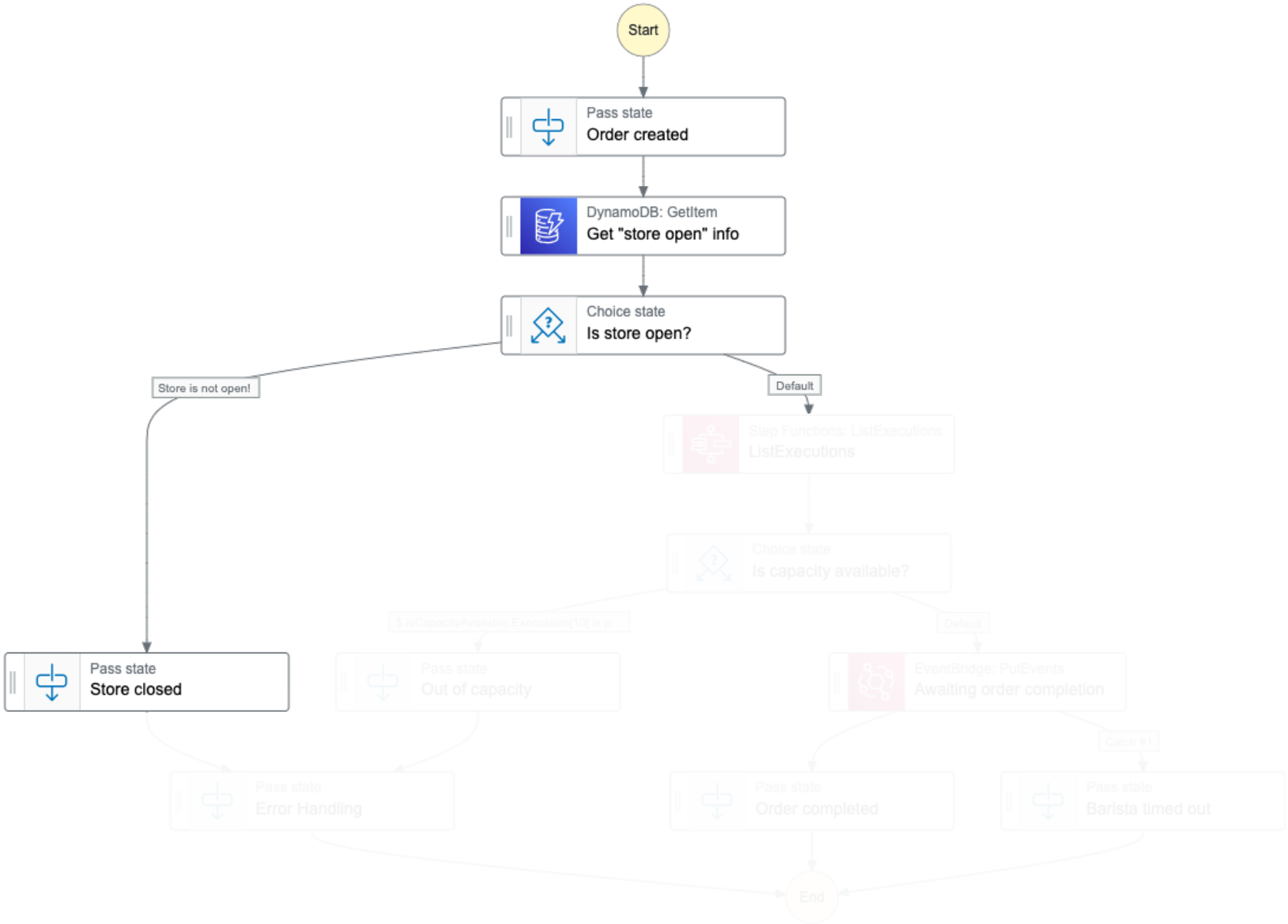
OPEN KNOWLEDGE


AWS Cloud
Front ends
Ordering app
Display app
Barista app
Order Manager service
API Gateway
Lambda function
DynamoDB table
QR validator service
API Gateway
Lambda function
Publisher service
IoT Core
Lambda function
Events
Events
Events
EventBridge Event bus
EventBridge rules
Events
Events
Step Functions
Order processing workflow
Start
Order created
Get "store open" info
Is store open?
ListExecutions
Is capacity available?
Error Handling
Awaiting order completion
Order completed
Barista timed out
End
In progress
Failed
Caught error
Canceled
Succeeded

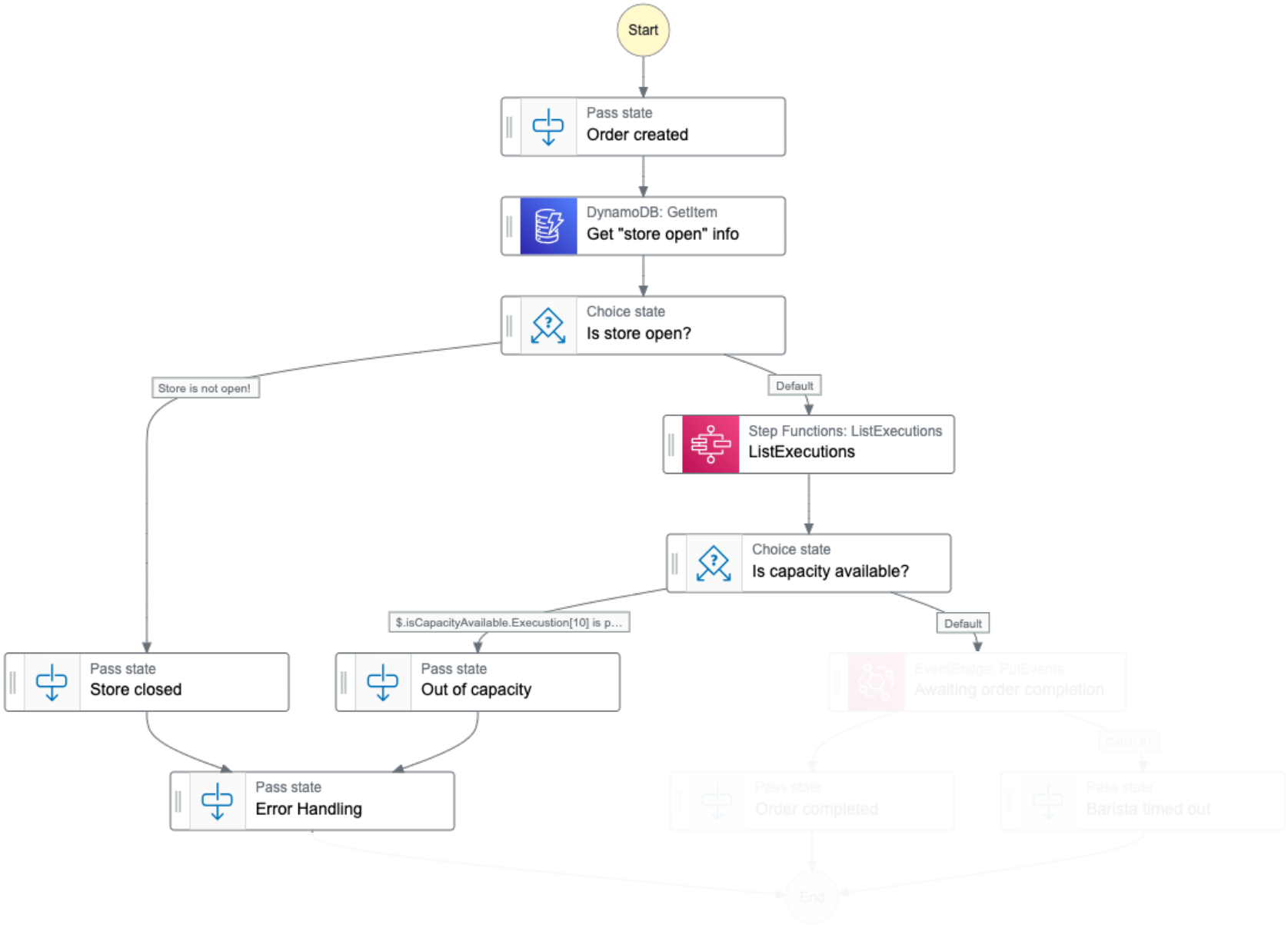

AWS Cloud
Front ends
Ordering app
Display app
Barista app
Order Manager service
API Gateway
Lambda function
DynamoDB table
QR validator service
API Gateway
Lambda function
Publisher service
IoT Core
Lambda function
Events
Events
Events
Events
EventBridge Event bus
EventBridge rules
Events
Events
Step Functions
Order processing workflow
Start
Order created
Get "store open" info
Is store open?
ListExecutions
Is capacity available?
Error Handling
Awaiting order completion
Order completed
Barista timed out
End
In progress
Failed
Caught error
Canceled
Succeeded



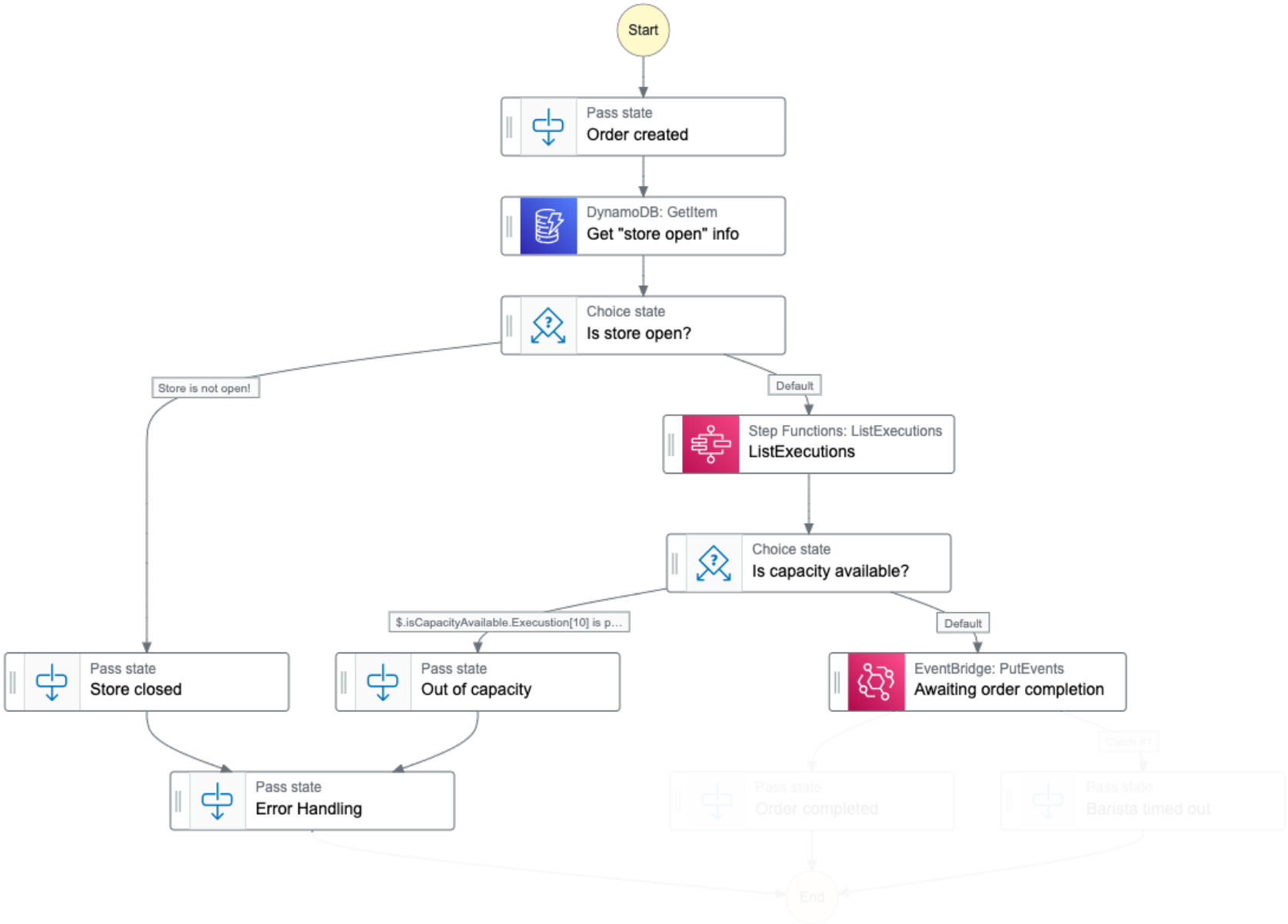
Order Processing: **Initialization**



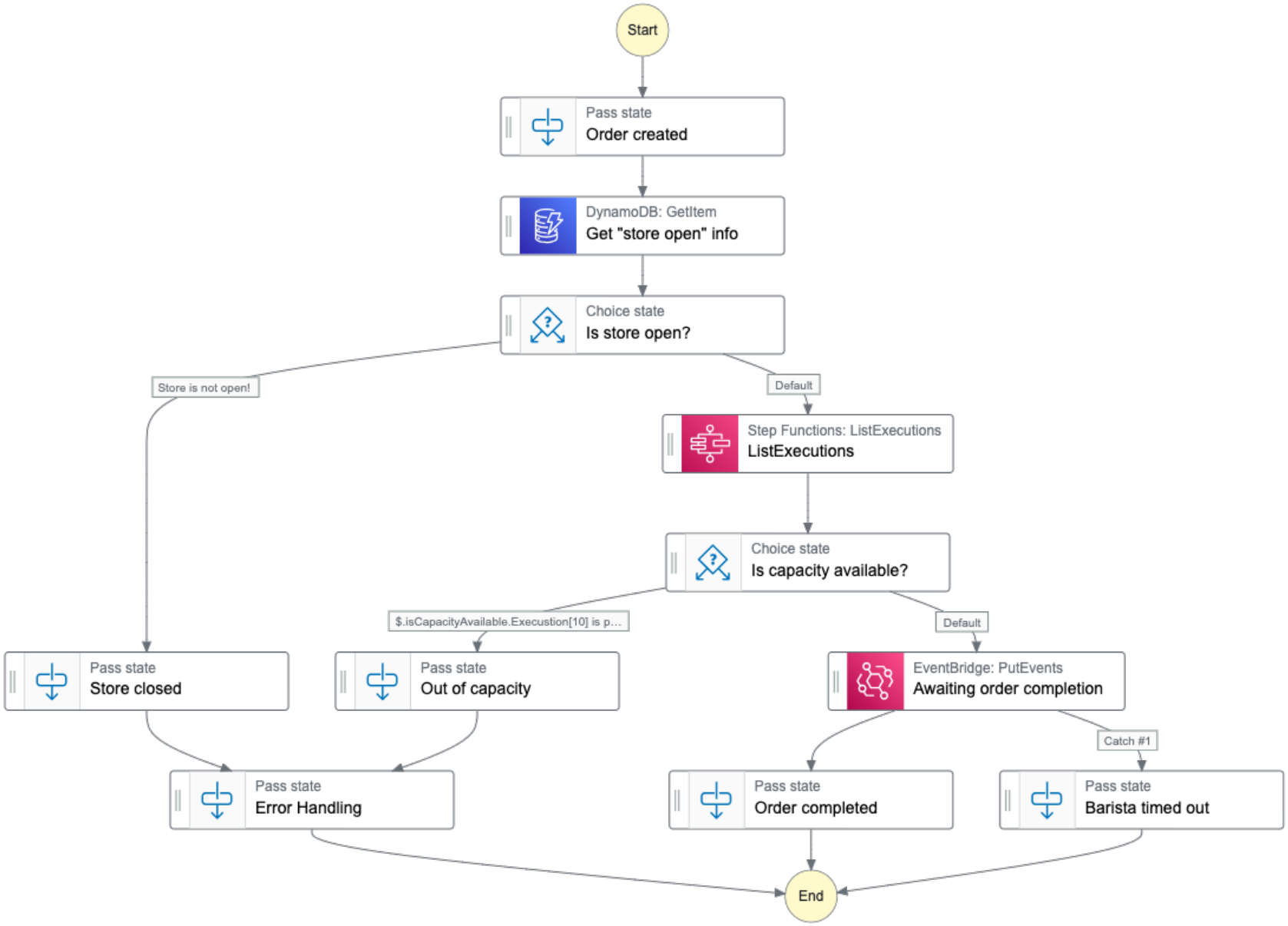
Order Processing: **Is store open?**



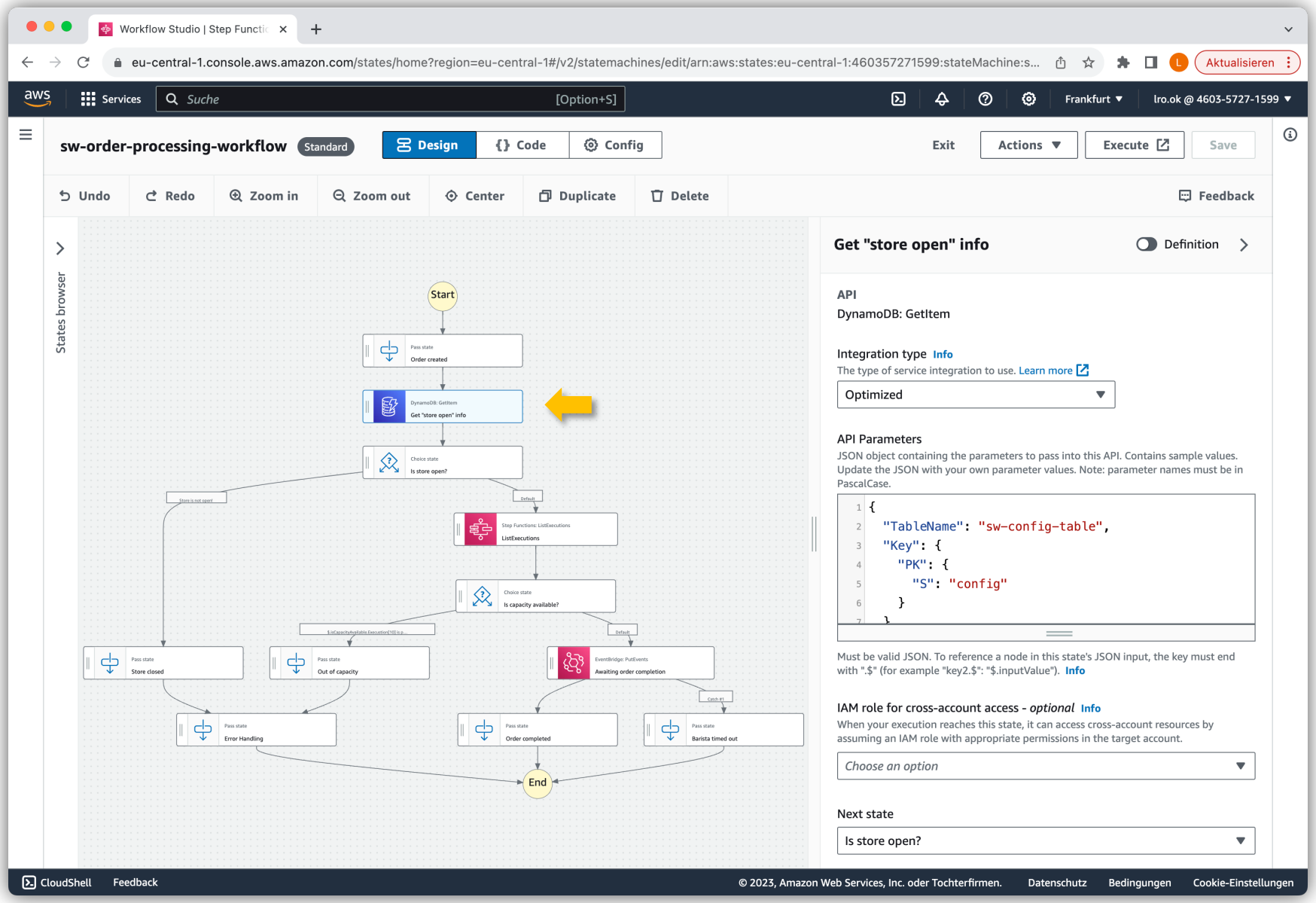
Order Processing: **Is capacity available?**



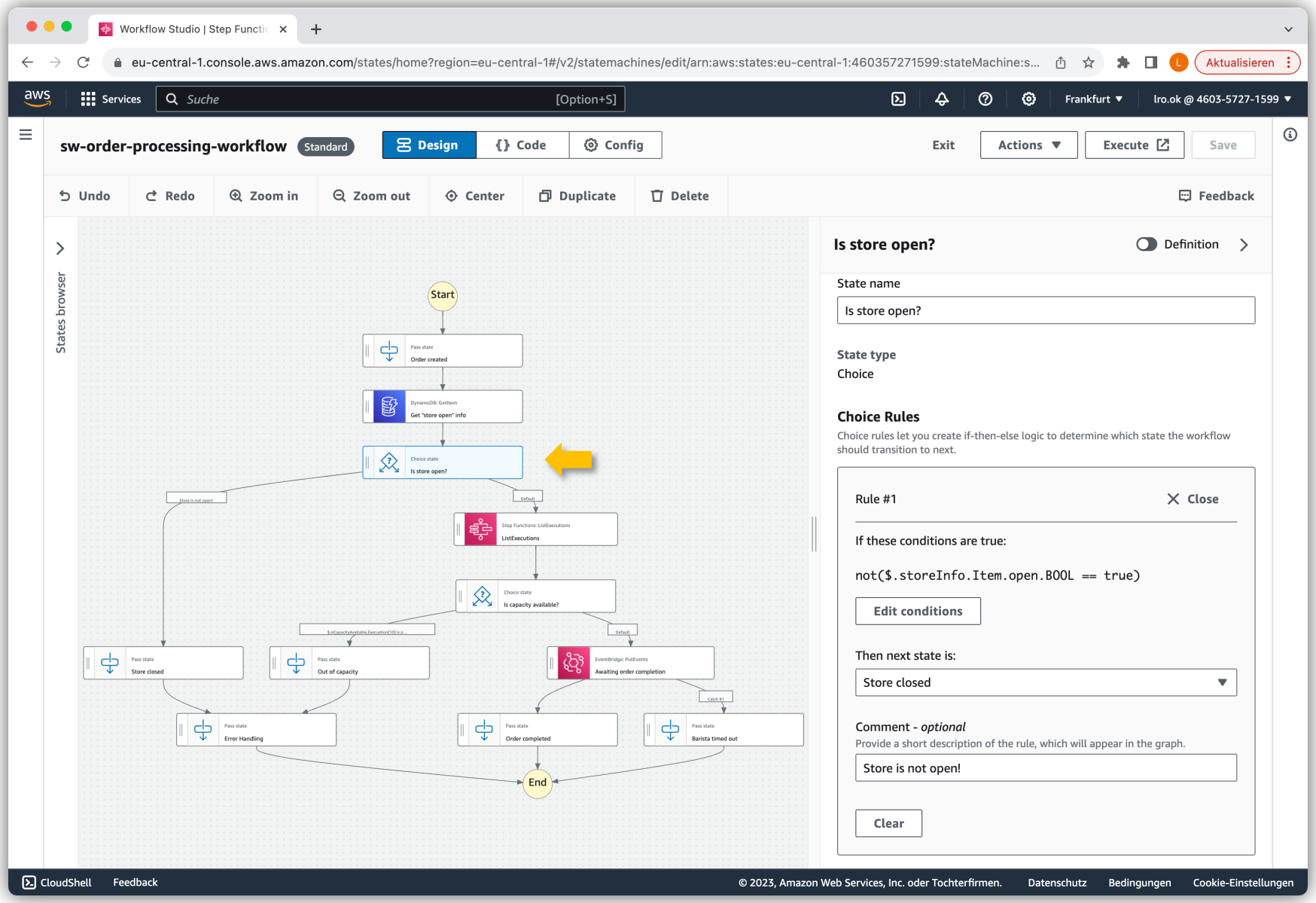
Order Processing: **Awaiting completion!**



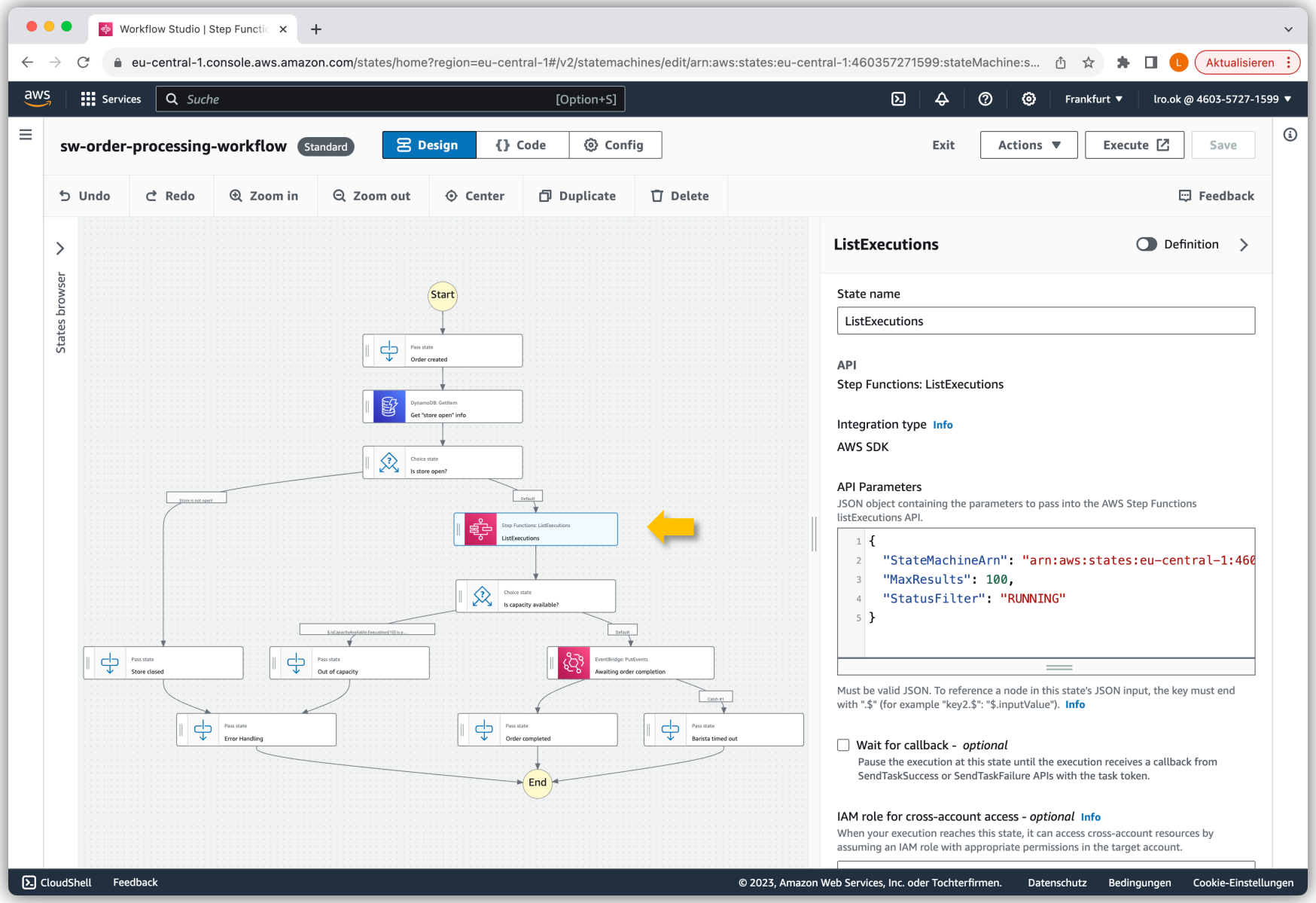
Order Processing: **Final Workflow**



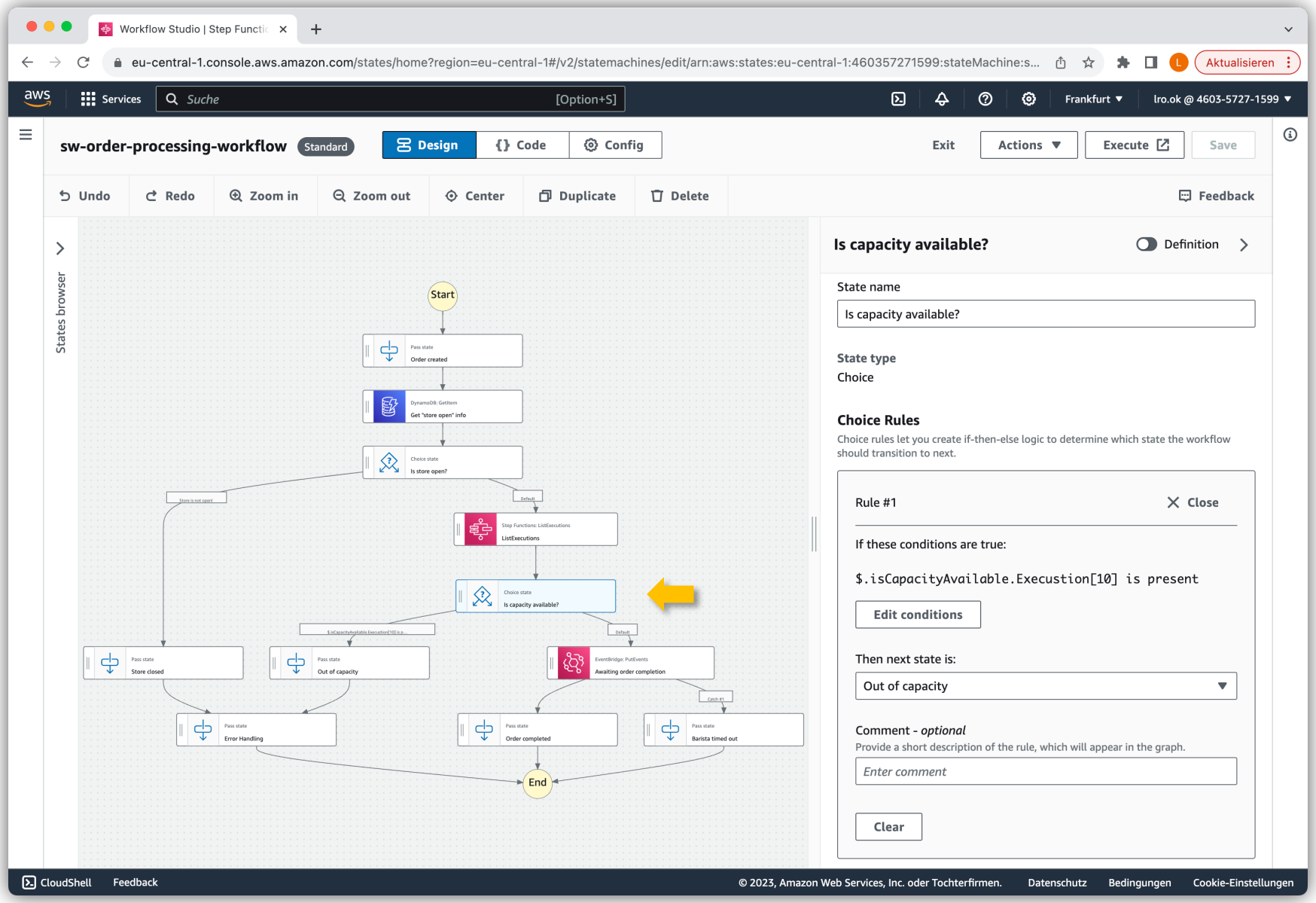
Order Processing: **Get Store Open Info from DynamoDB**



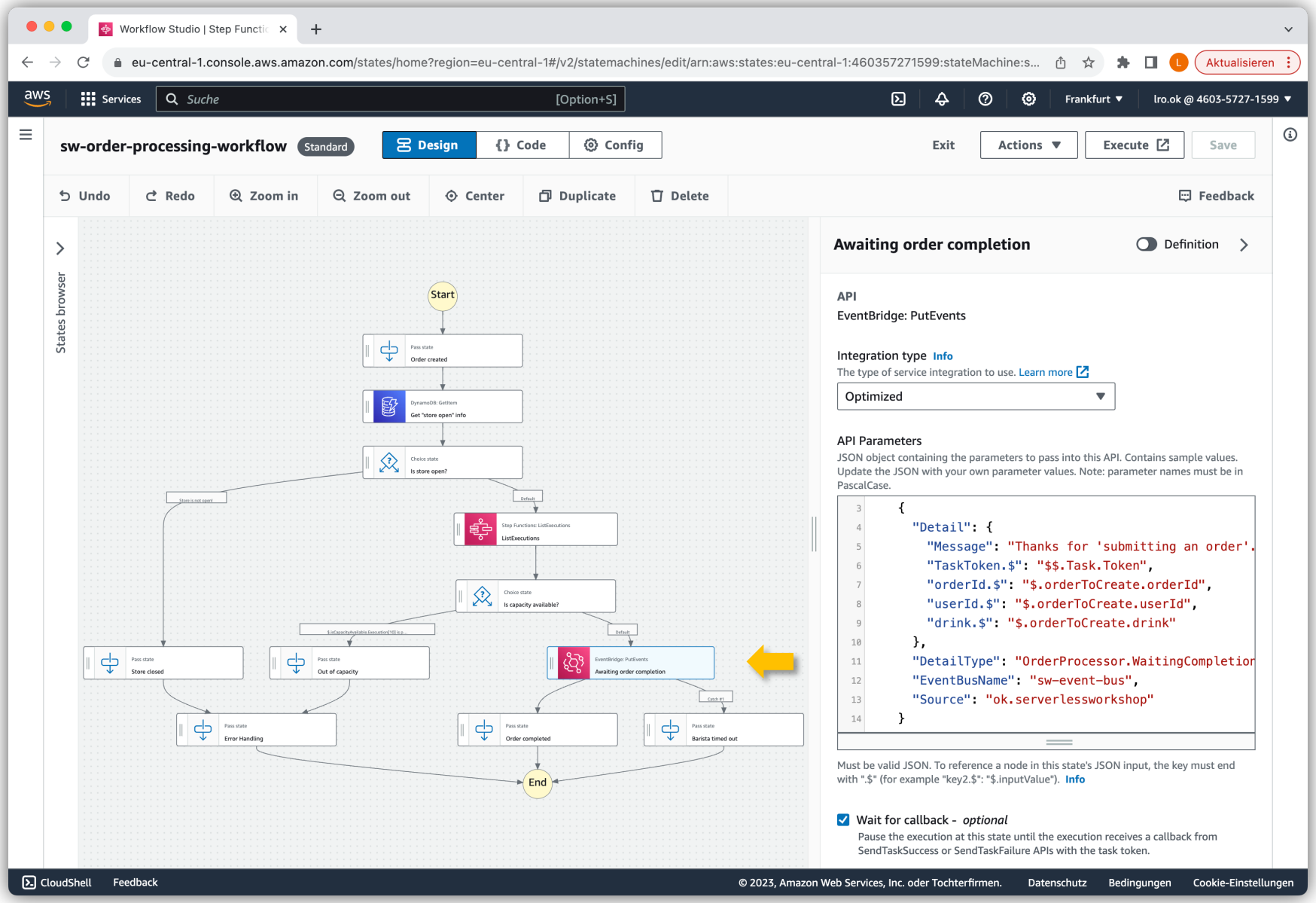
Order Processing: **Check if Store Open**



Order Processing: **Get # of concurrent Workflows**



Order Processing: **Check Capacity**



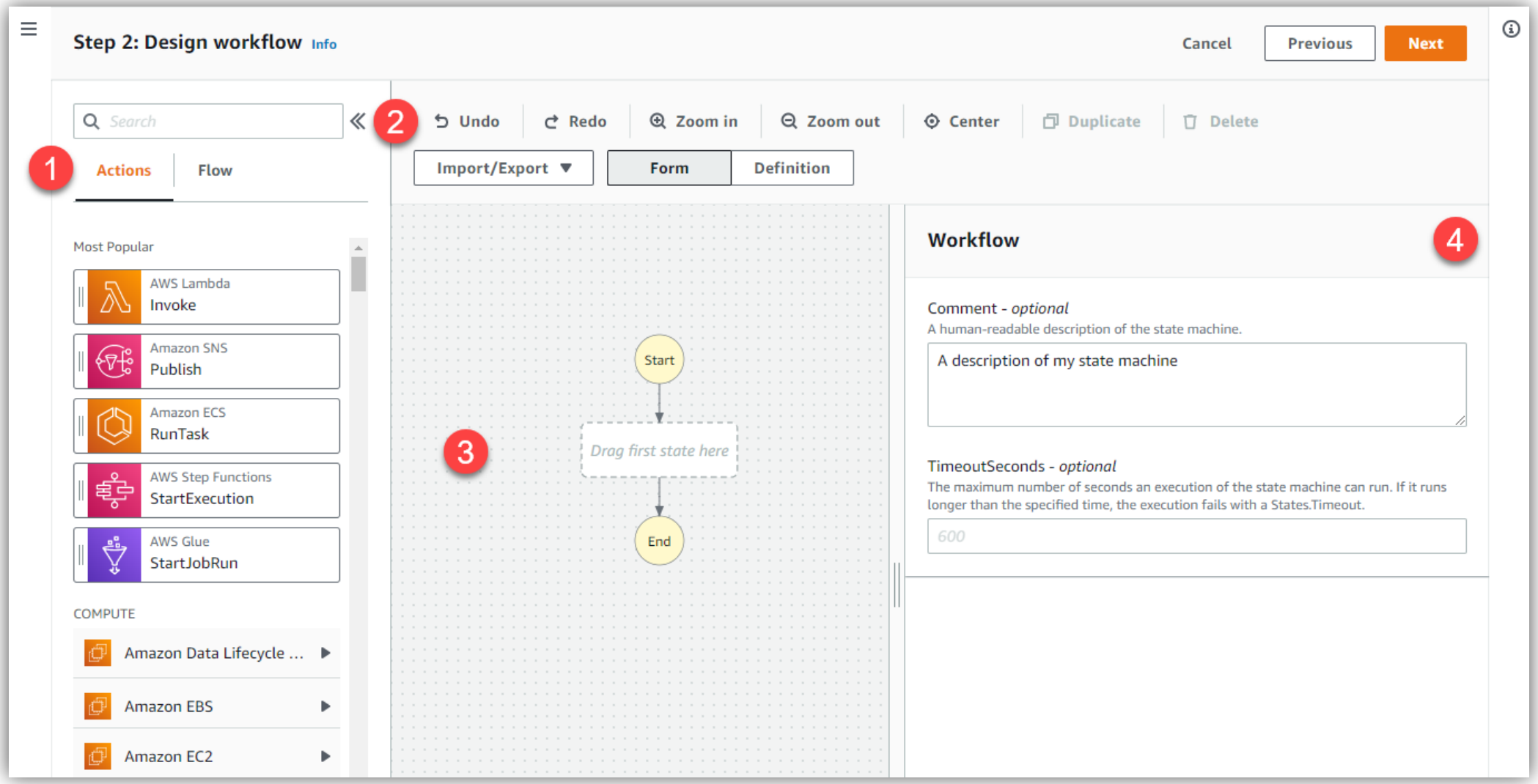
Order Processing: **Send Event to EventBridge and … WAIT**

HANDS-ON

Order Processing

- Store open?
- Capacity available?
- Coffee done?

- Success Handling
- Error Handling



AWS StepFunctions: **Viusal Workflow Desinger**

Lessons Learned:

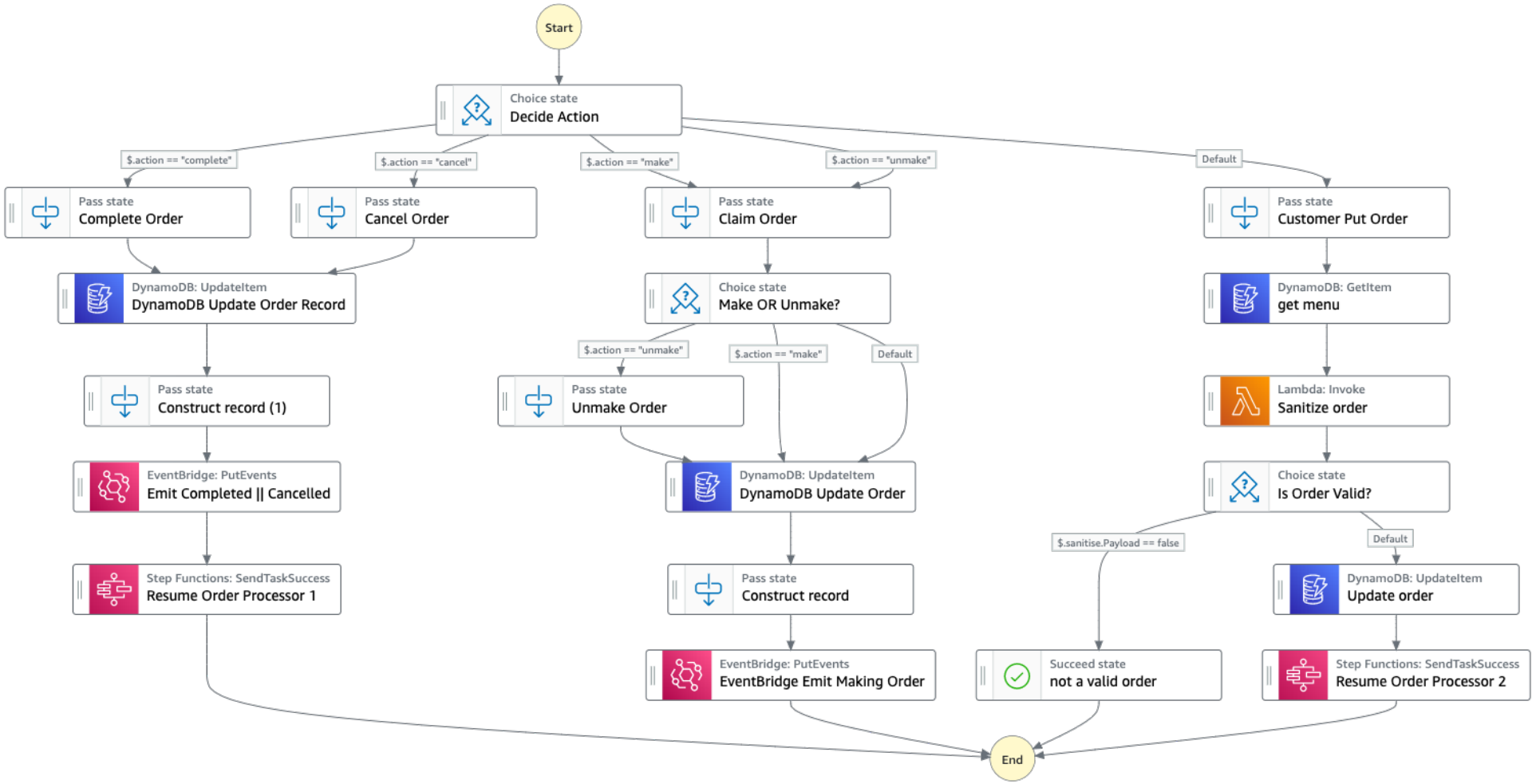
Workflows via StepFunctions

WORKFLOWS VIA STEP FUNCTIONS

Lessons Learned

- **Flow** Elements
- **Action** Elements
- **Callback** via EventBridge Events
- **Payload Manipulation** via JSONPath
- **Success** Handling
- **Error** Handling

WORKFLOWS VIA STEP FUNCTIONS



AWS StepFunctions: **Order Management Service (alternative Version)**

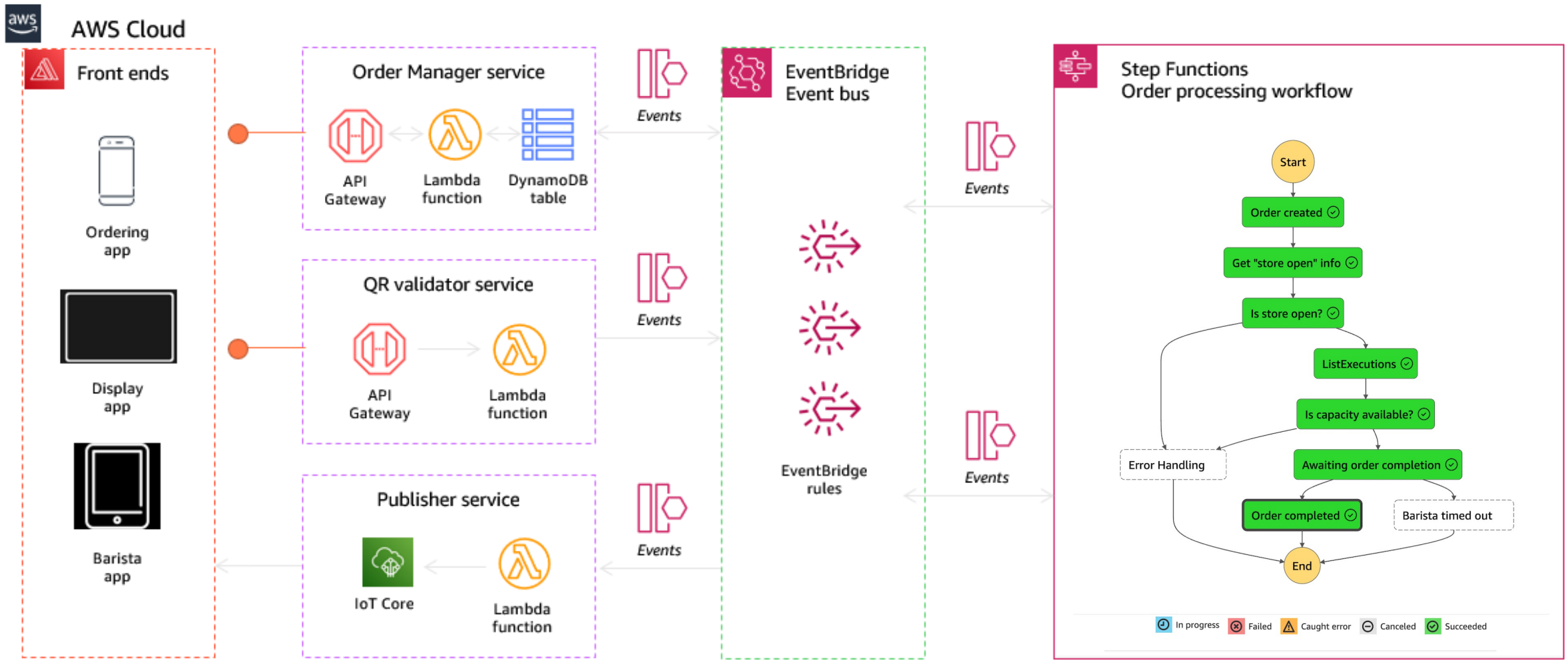
Agenda

- *Hello World & Set-up* ✅
- *Order Management* ✅
- *Order Processing* ✅
- *API Gateway*
- *Testing*

OPEN KNOW LEDGE

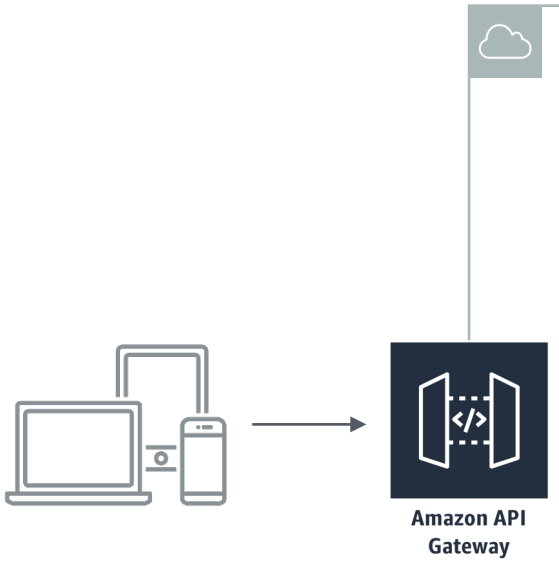
Serverless Coffee
API Gateway

OPEN KNOWLEDGE


AWS Cloud
Front ends
Ordering app
Display app
Barista app
Order Manager service
API Gateway
Lambda function
DynamoDB table
QR validator service
API Gateway
Lambda function
Publisher service
IoT Core
Lambda function
Events
Events
Events
EventBridge Event bus
EventBridge rules
Events
Events
Step Functions
Order processing workflow
Start
Order created
Get "store open" info
Is store open?
ListExecutions
Is capacity available?
Error Handling
Awaiting order completion
Order completed
Barista timed out
End
In progress
Failed
Caught error
Canceled
Succeeded

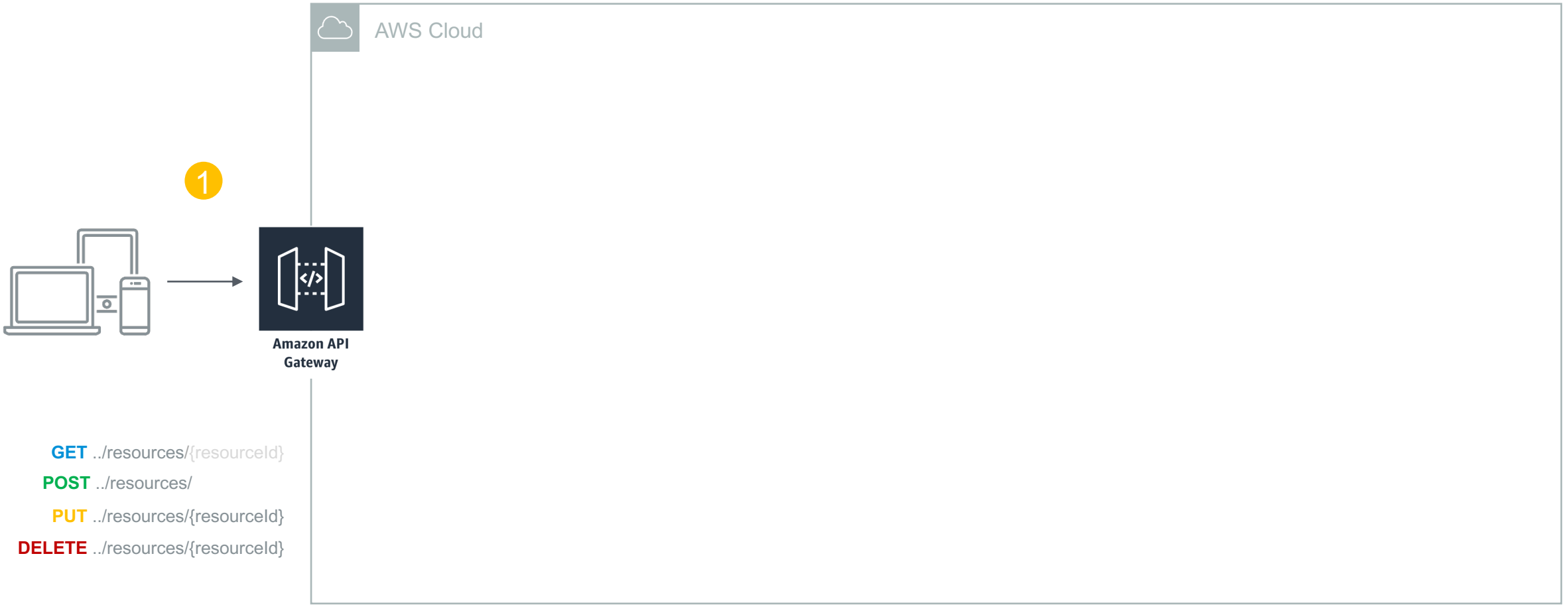

AWS Cloud
Front ends
Ordering app
Display app
Barista app
Order Manager service
API Gateway
Lambda function
DynamoDB table
QR validator service
API Gateway
Lambda function
Publisher service
IoT Core
Lambda function
Events
Events
Events
EventBridge Event bus
EventBridge rules
Step Functions
Order processing workflow
Start
Order created
Get "store open" info
Is store open?
ListExecutions
Is capacity available?
Error Handling
Awaiting order completion
Order completed
Barista timed out
End
In progress
Failed
Caught error
Canceled
Succeeded

Wofür ist ein **API Gateway gut?**

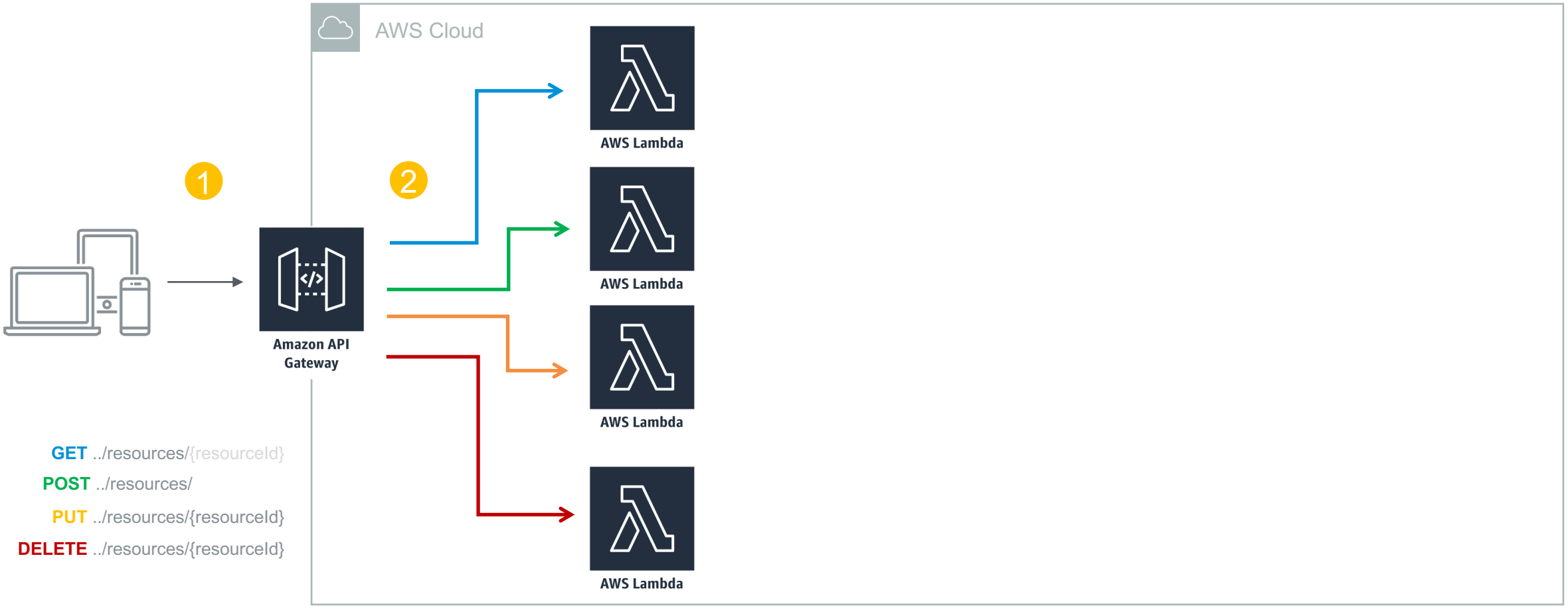


- Security
- API Key Handling
- Throttling
- Proxying
- Logging / Tracing
- Request / Respons Mapping
- Staging

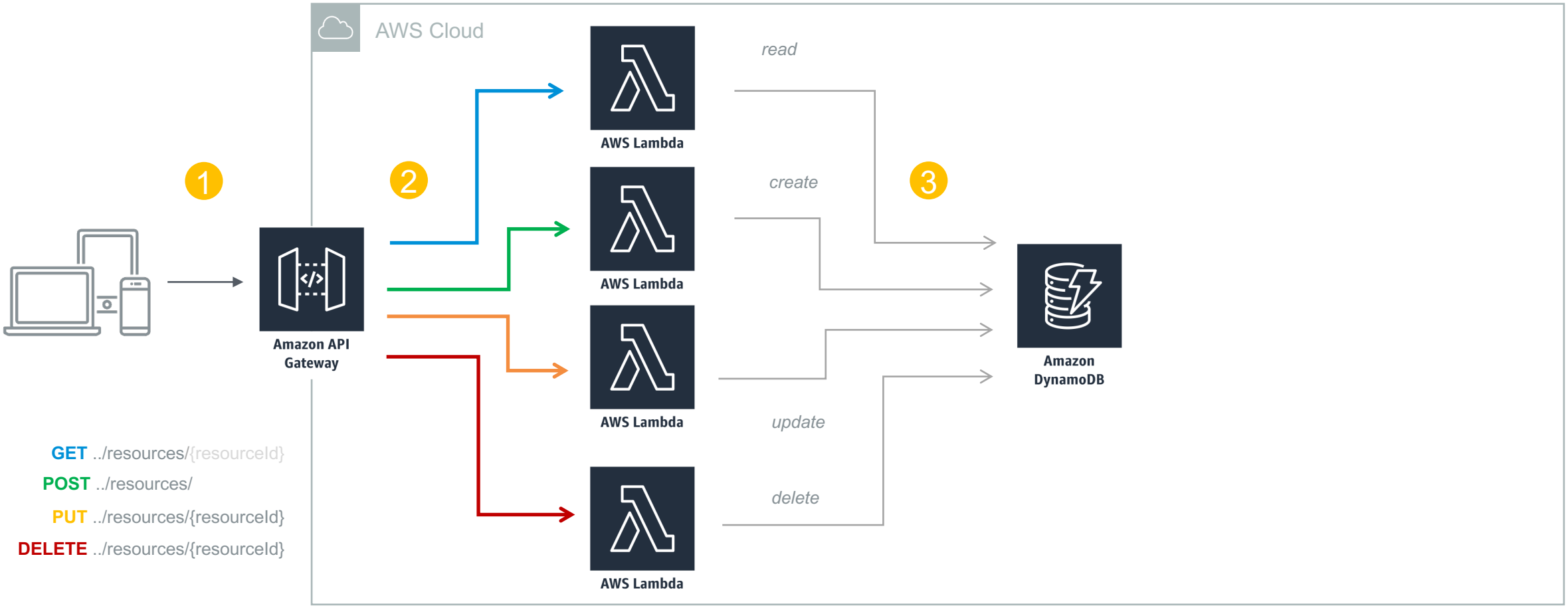
AWS API Gateway



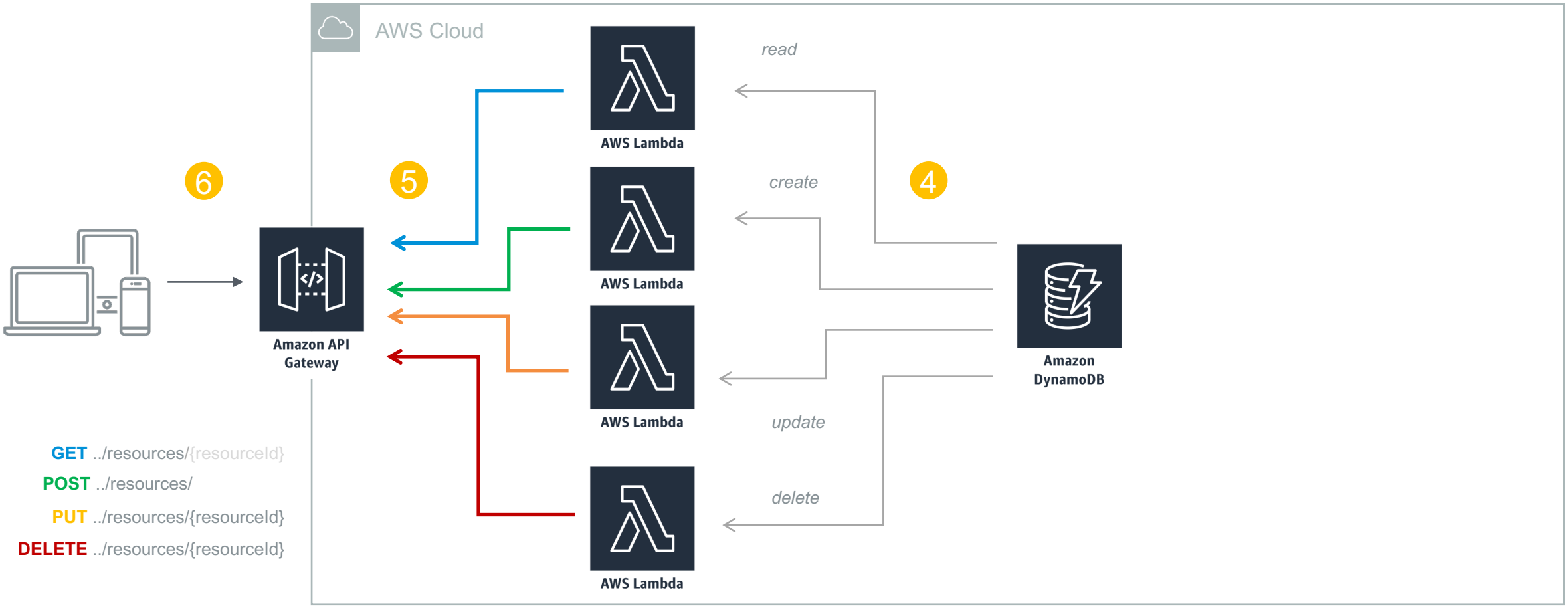
AWS API Gateway



AWS API Gateway



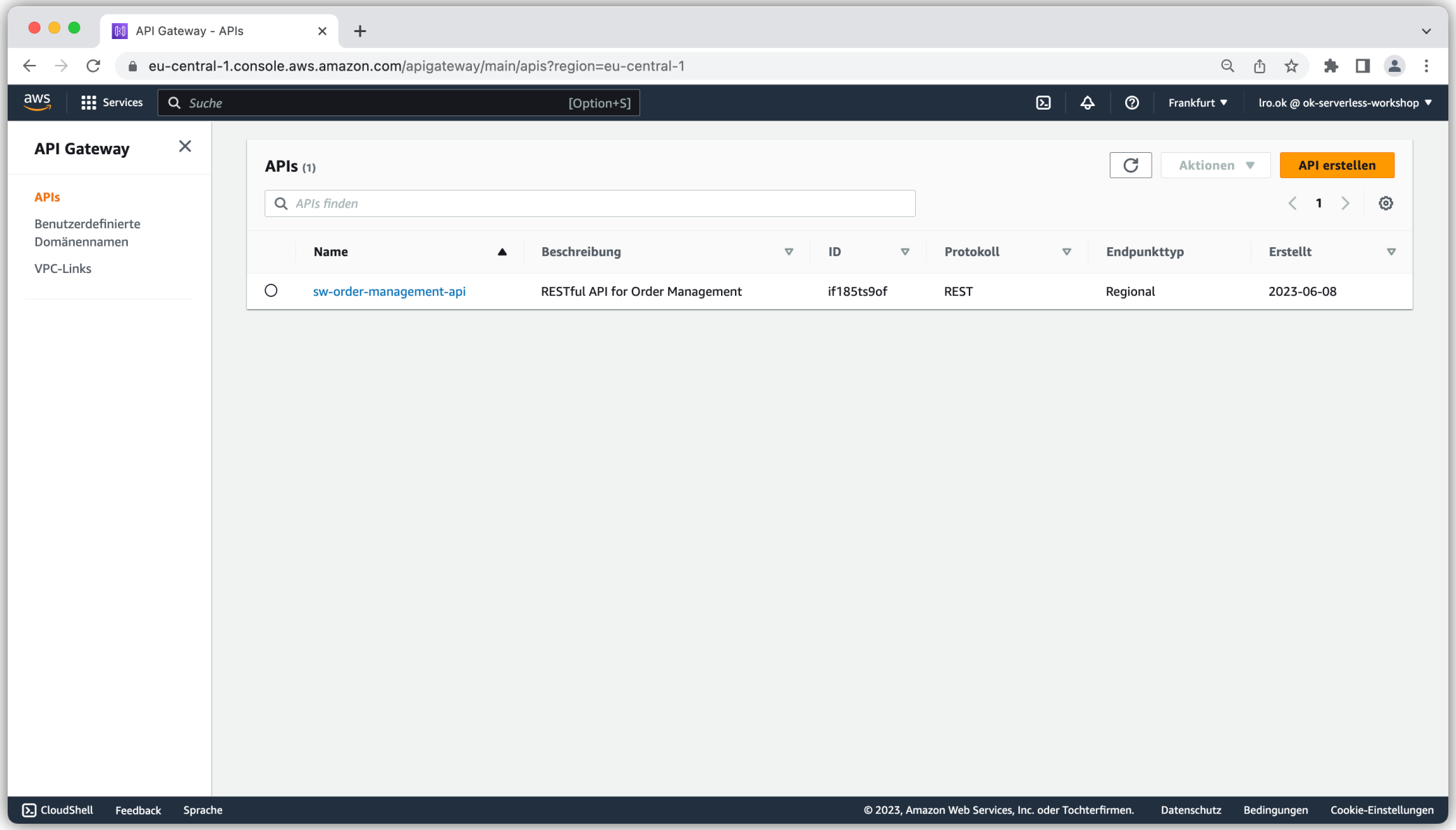
AWS API Gateway



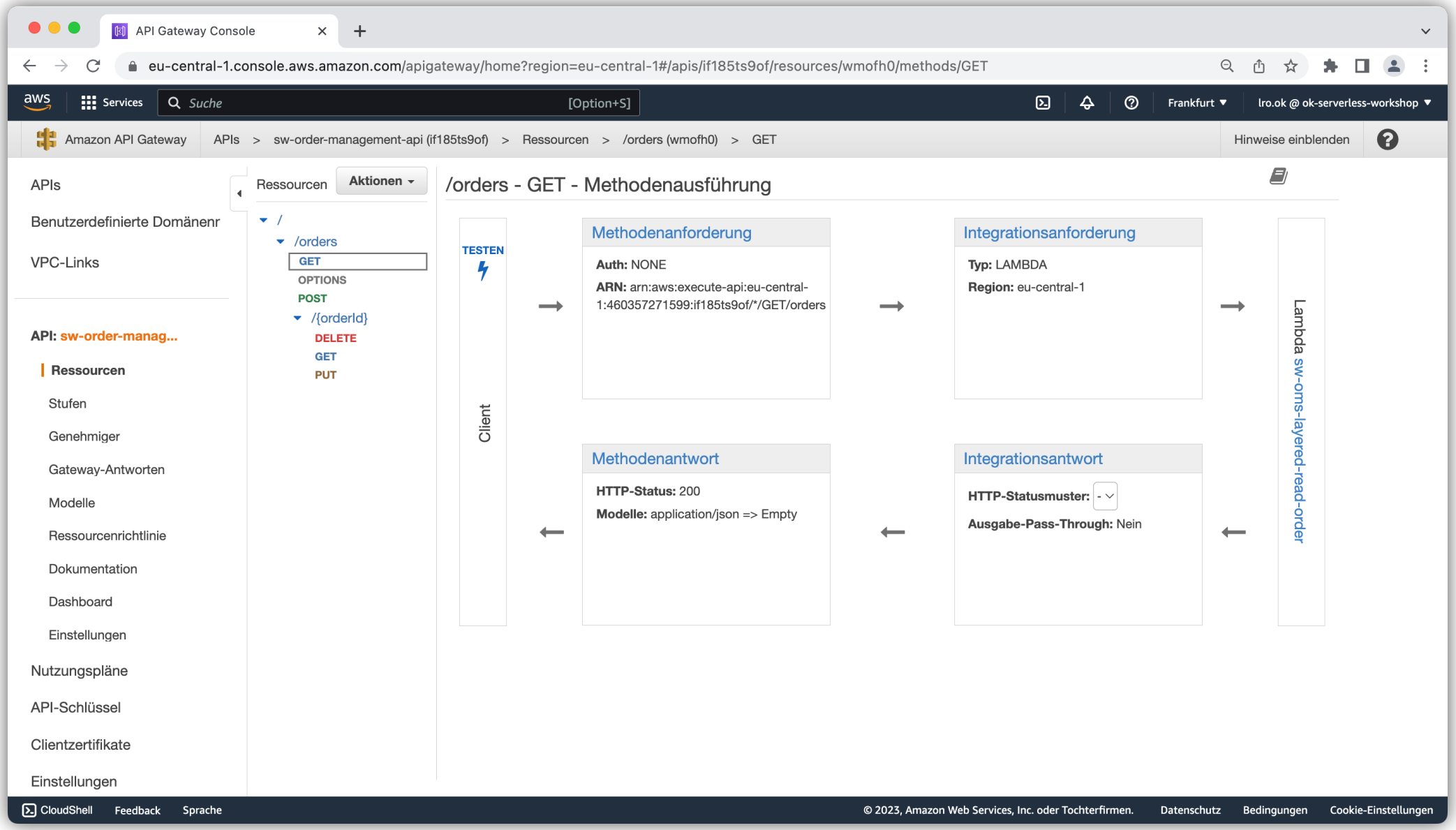
HANDS-ON

API Gateway

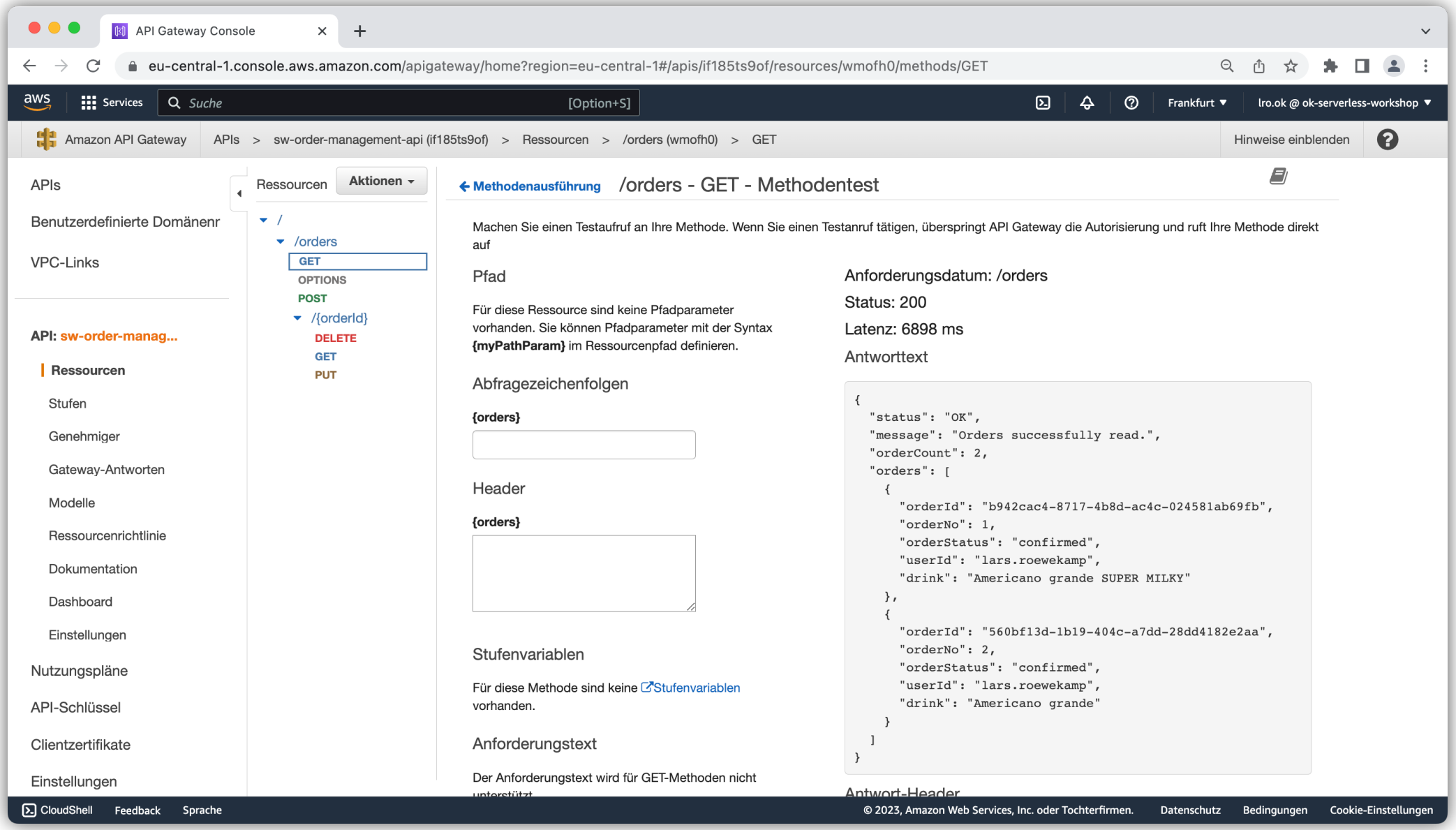
- Backend Integration
- Request Mapping
- Response Mapping
- Mocking
- Multiple Stages



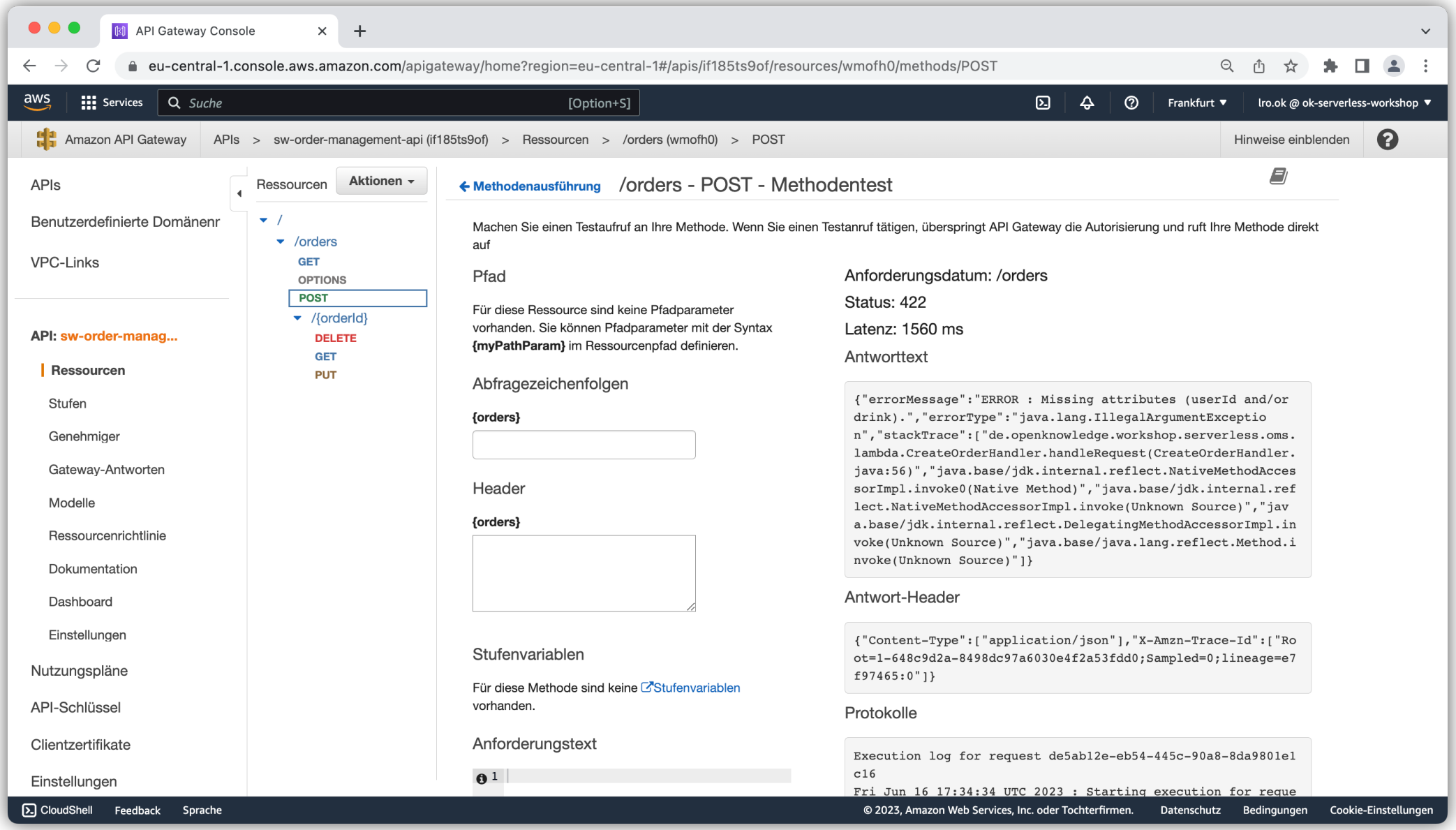
API Gateway: Order Management API



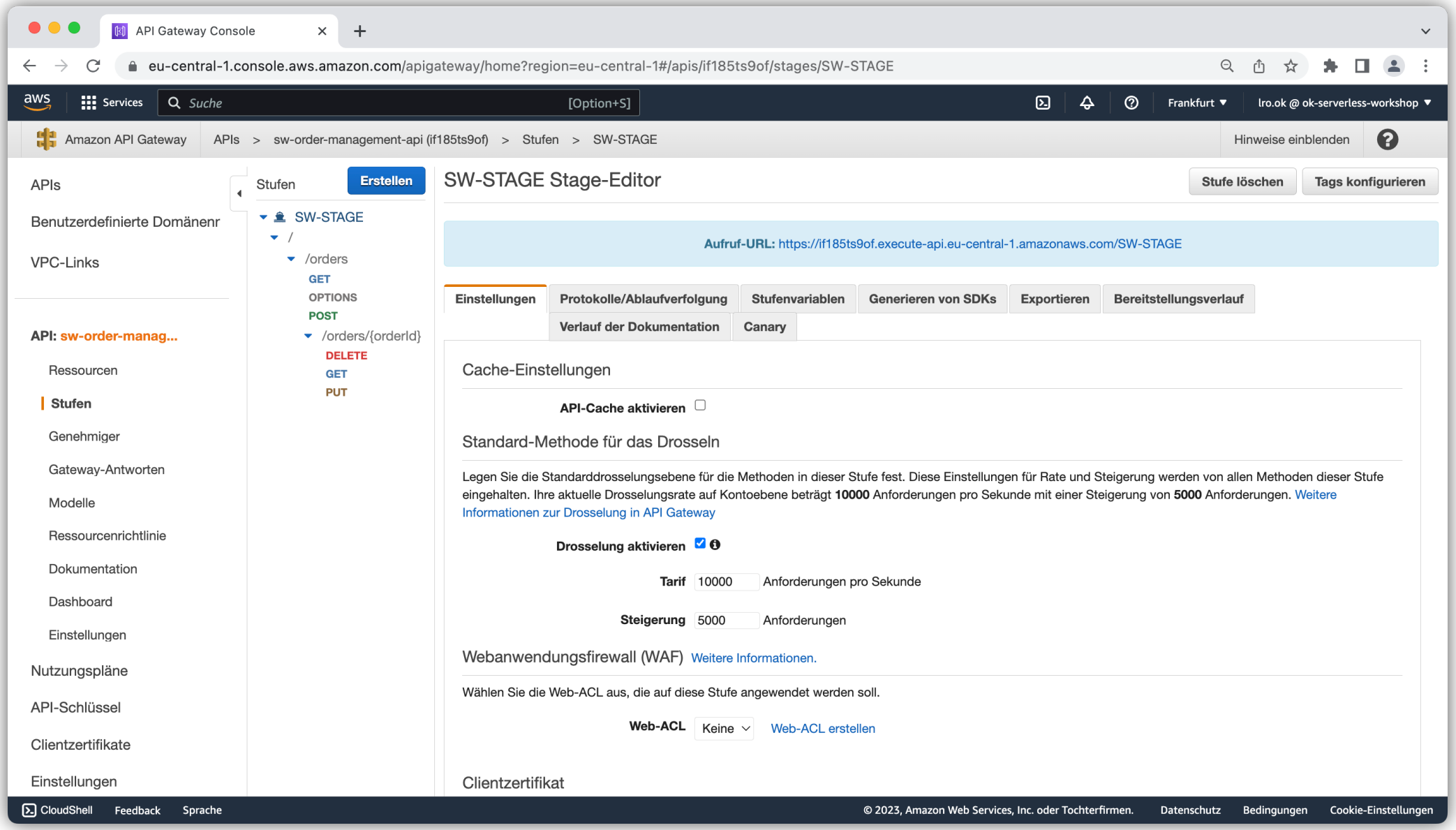
API Gateway: Ressourcen & Methoden



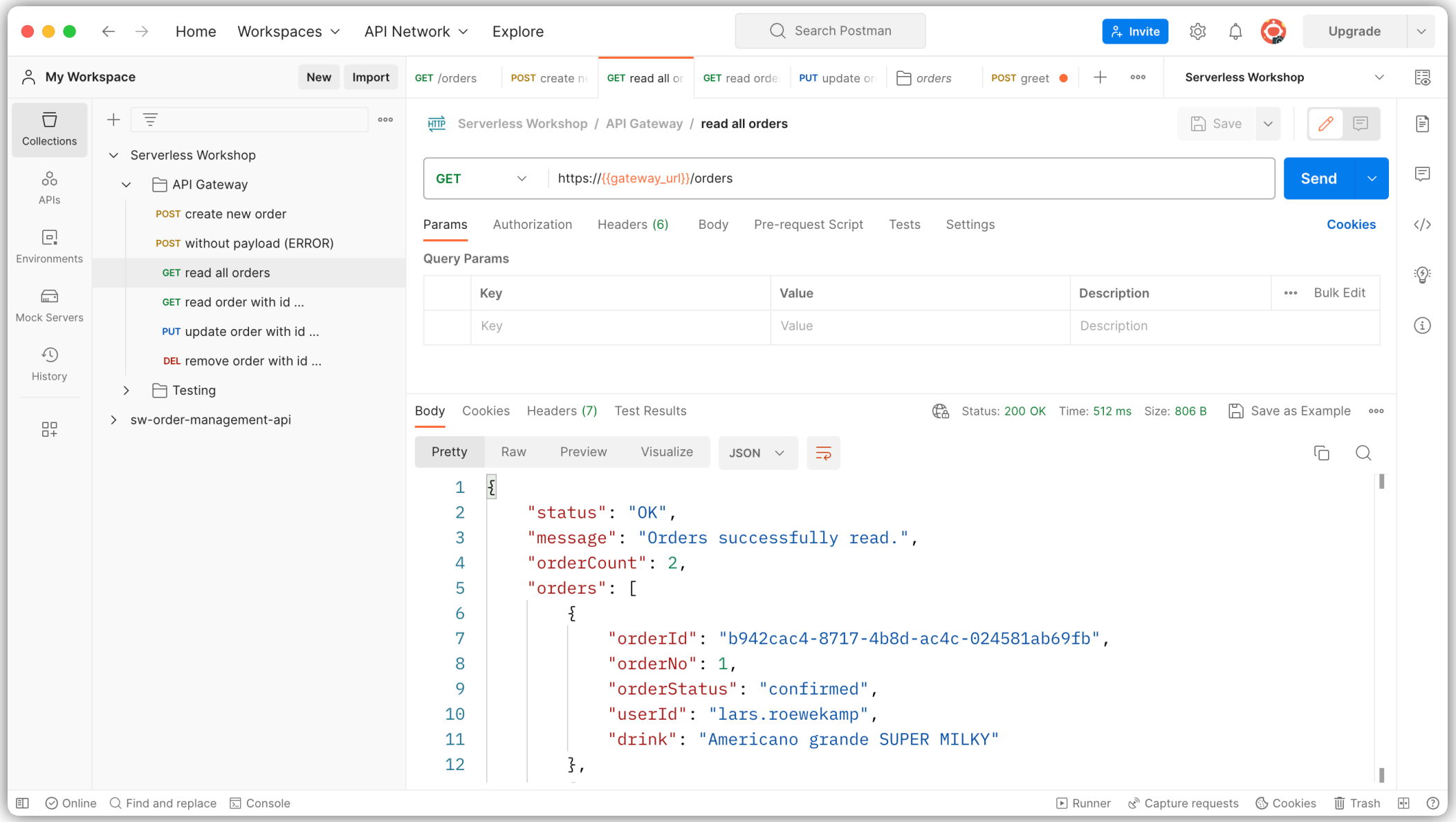
API Gateway: GET all orders



API Gateway: POST new order (ERROR)



API Gateway: Staging



API Gateway: External Call (via POSTMAN)

Lessons learned:

API Gateway

API GATEWAY

Lessons Learned

- **Schutztor** zur der Aussenwelt
- **HTTP Request** werden zu Lambda Triggern
- **HTTP Payload** wird zu Event Payload
- **Lambda Result** wird zu HTTP Status Code
- **Lambda Result** Object wird zu HTTP Payload
- **Staging-Konzept** sorgt für Flexibilität

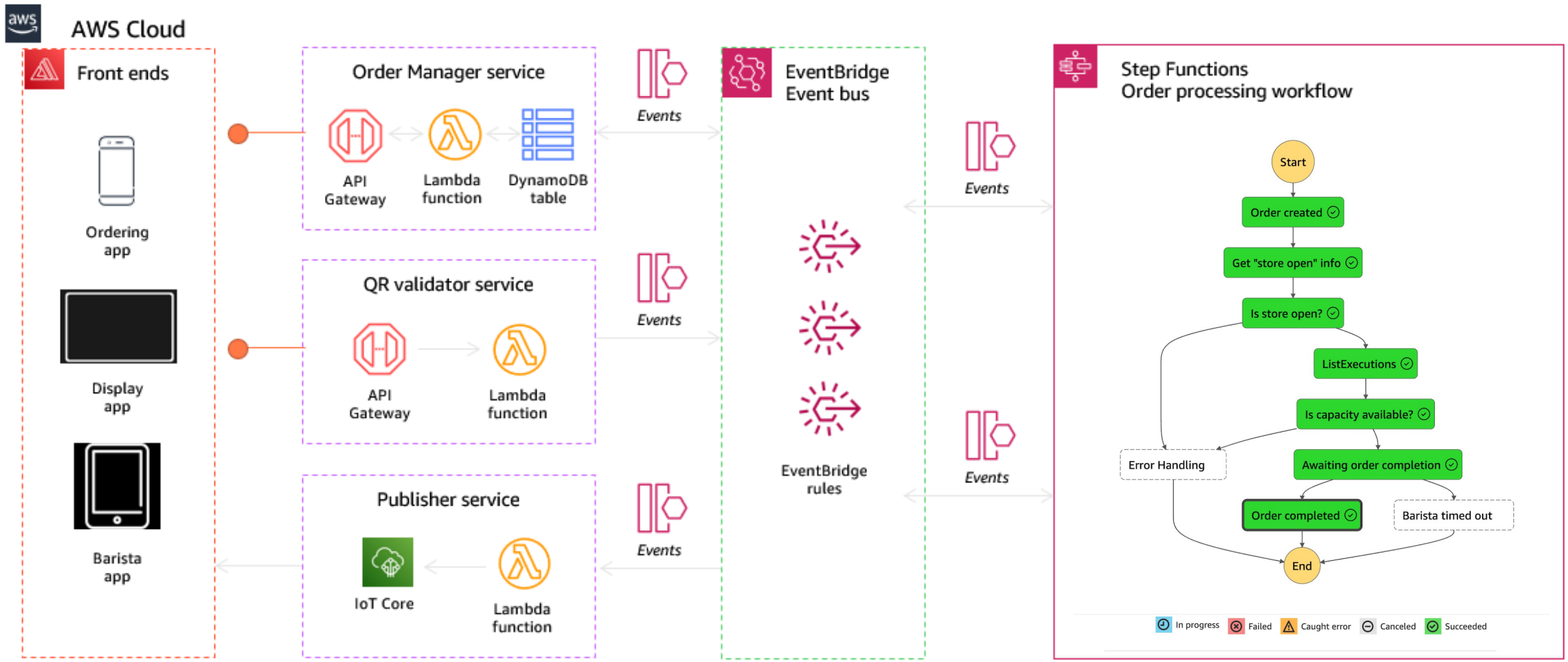
Agenda

- *Hello World & Set-up* ✅
- *Order Management* ✅
- *Order Processing* ✅
- *API Gateway* ✅
- *Testing*

OPEN KNOWLEDGE

Serverless Coffee
Testing U/I/A

OPEN KNOWLEDGE


AWS Cloud
Front ends
Ordering app
Display app
Barista app
Order Manager service
API Gateway
Lambda function
DynamoDB table
QR validator service
API Gateway
Lambda function
Publisher service
IoT Core
Lambda function
Events
Events
Events
EventBridge Event bus
EventBridge rules
Events
Events
Step Functions
Order processing workflow
Start
Order created
Get "store open" info
Is store open?
ListExecutions
Is capacity available?
Error Handling
Awaiting order completion
Order completed
Barista timed out
End
In progress
Failed
Caught error
Canceled
Succeeded

HANDS-ON

Testing

- Unit Testing
- Integration Testing
- Acceptantce Testing

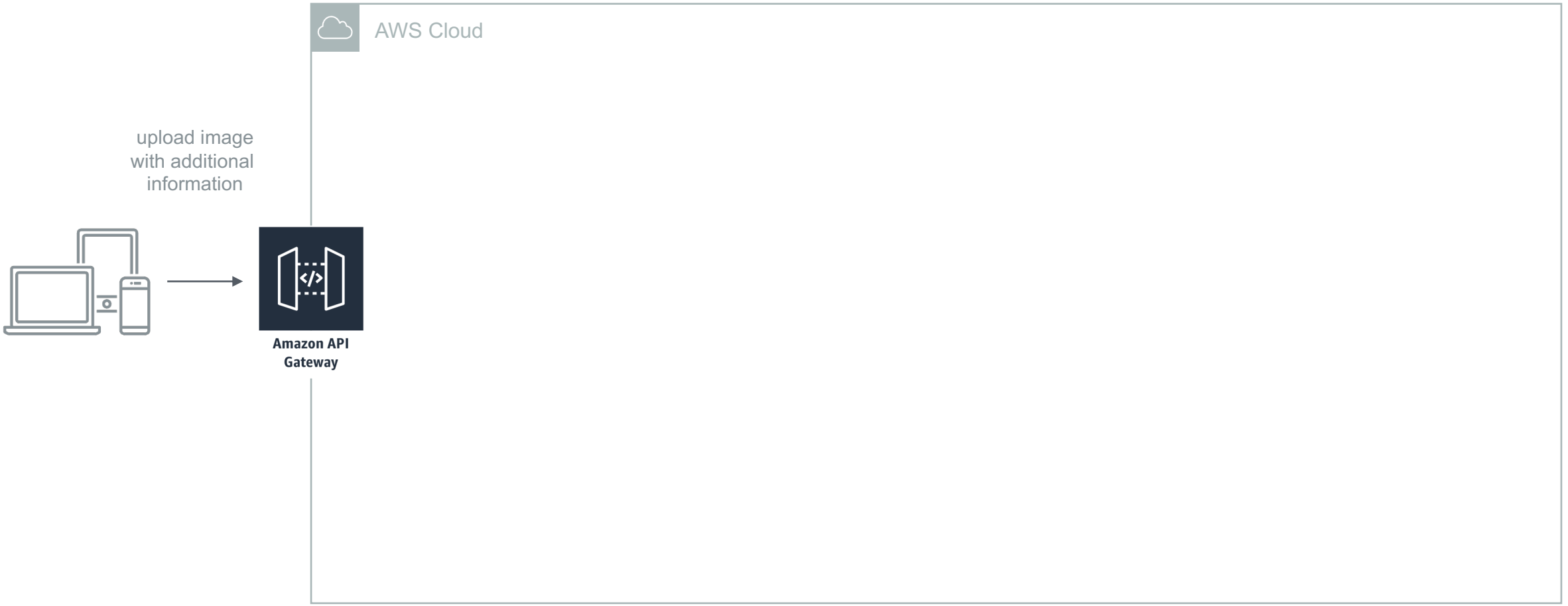
USE CASE

OrderApp
Admin erstellt
neuen Menu-Eintrag
via API Gateway

`POST /drinks …`

OPEN KNOW LEDGE

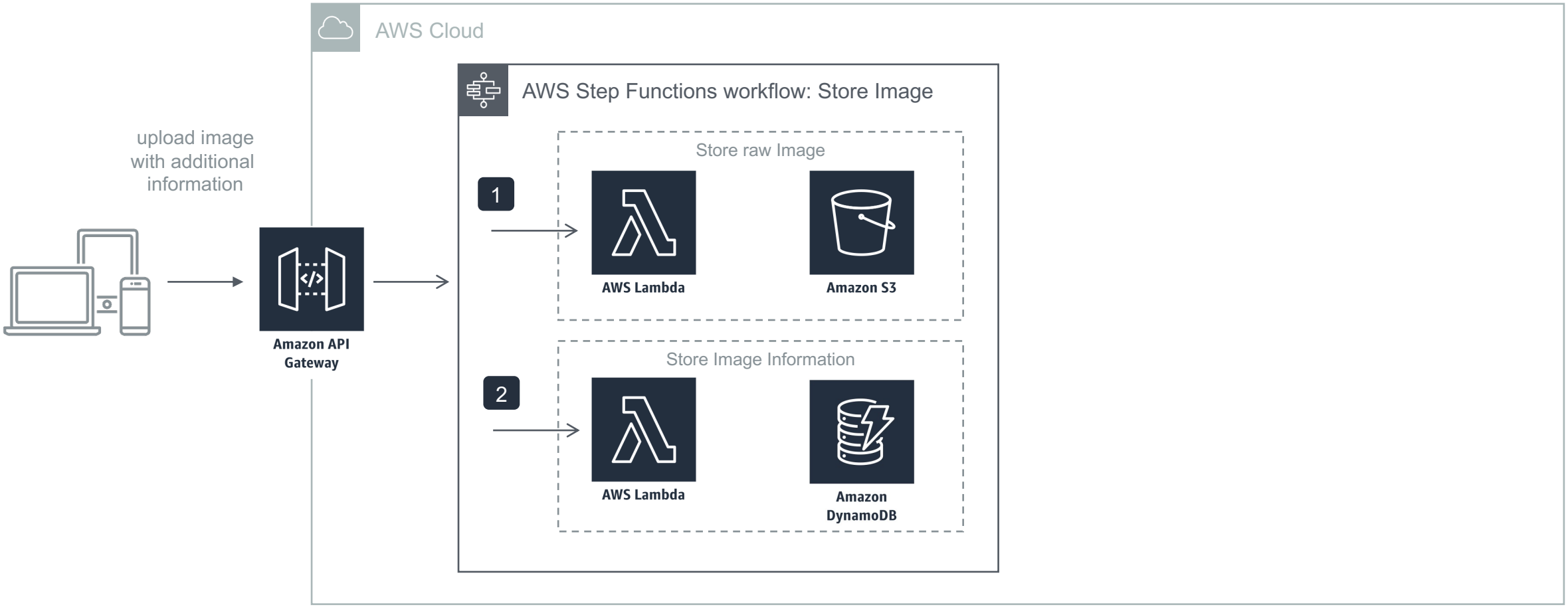
Upload Data & Image



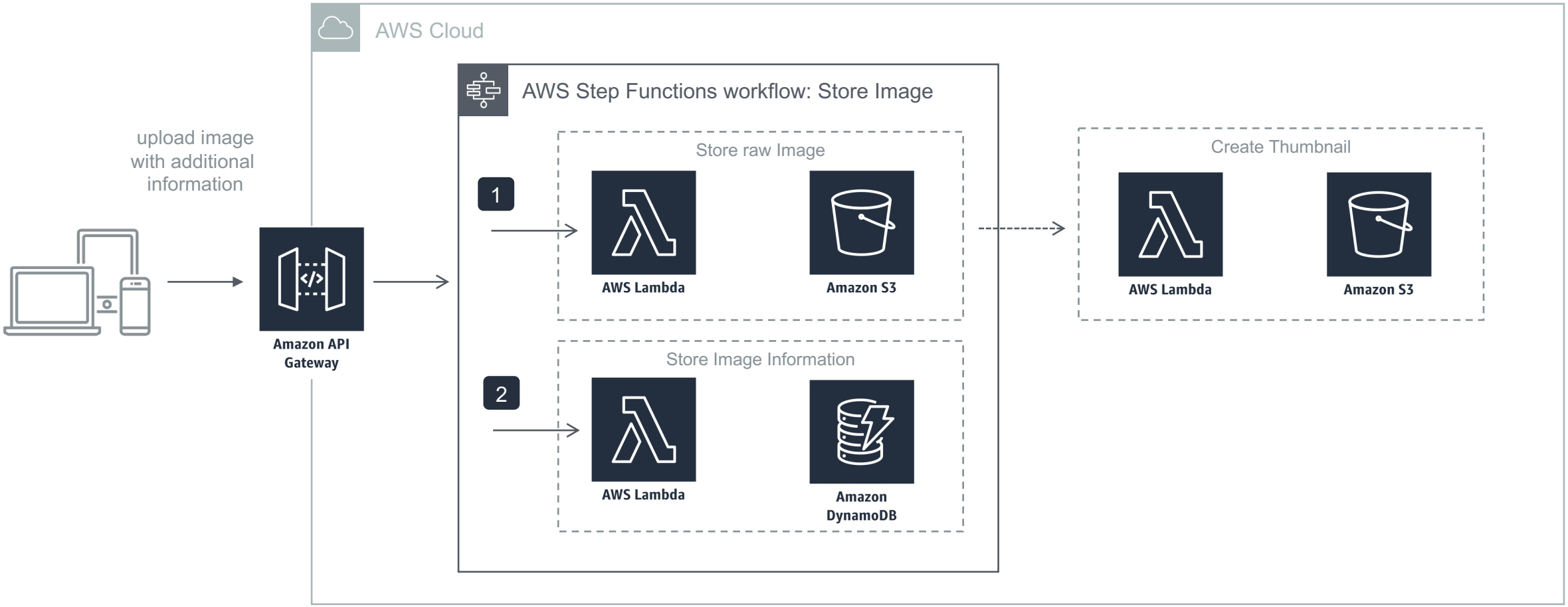
Upload Data & Image

Upload Data & Image

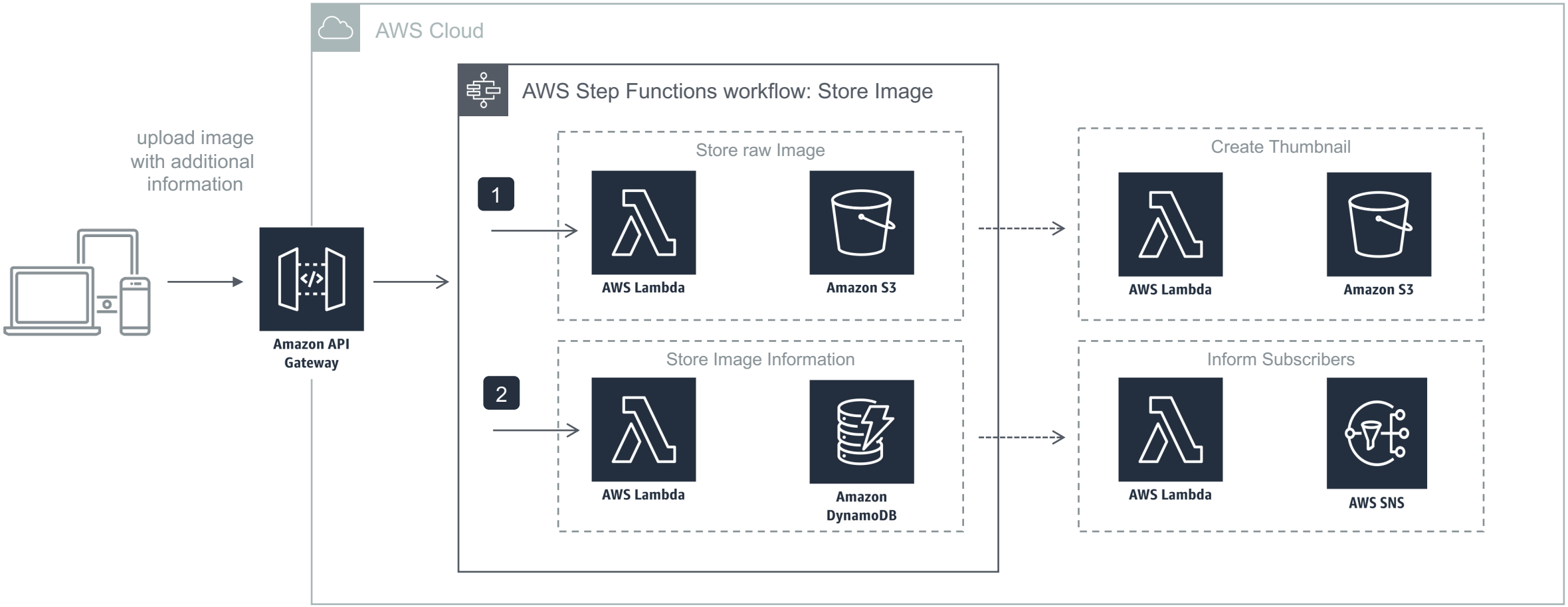
Upload Data & Image



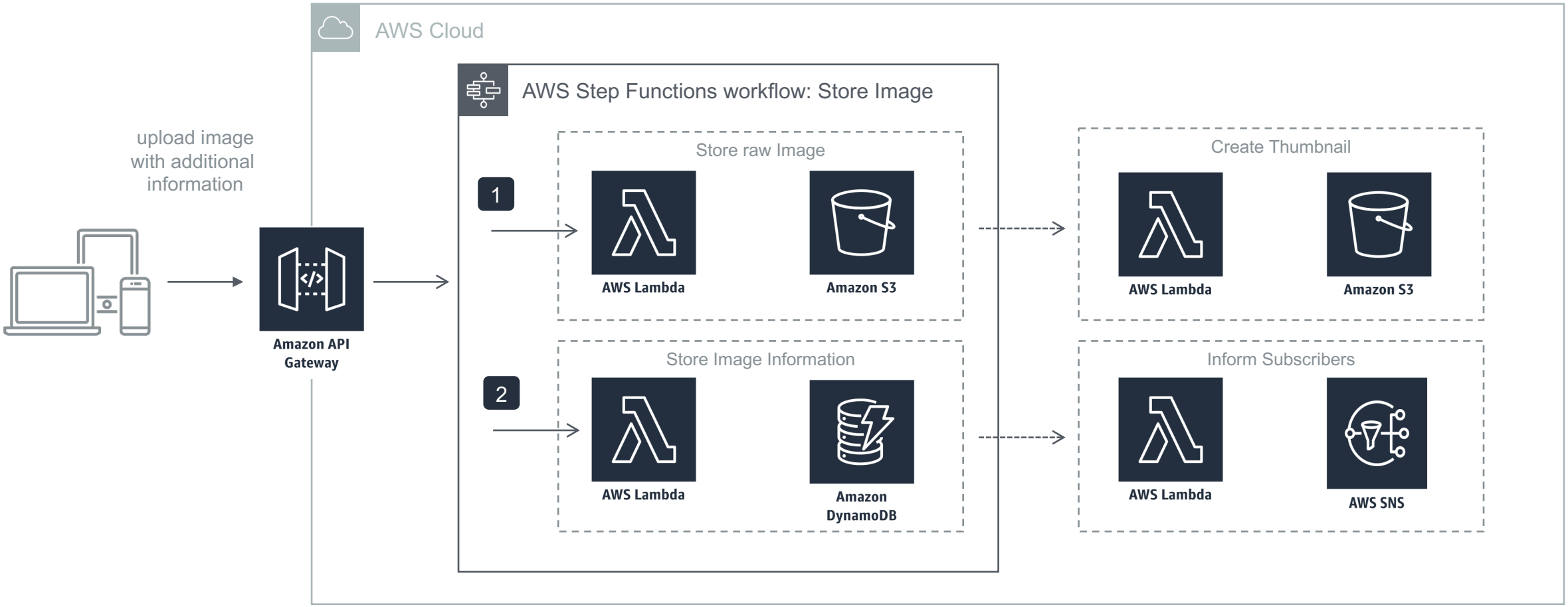
Upload Data & Image



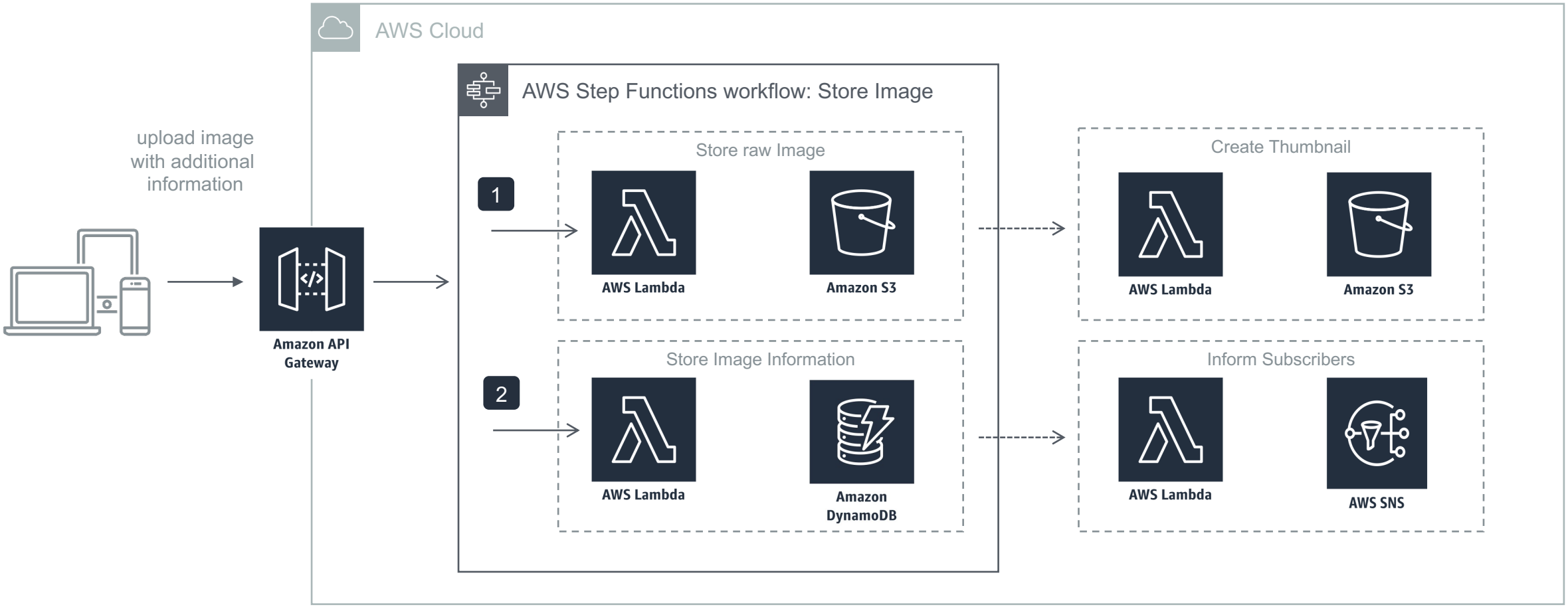
Upload Data & Image



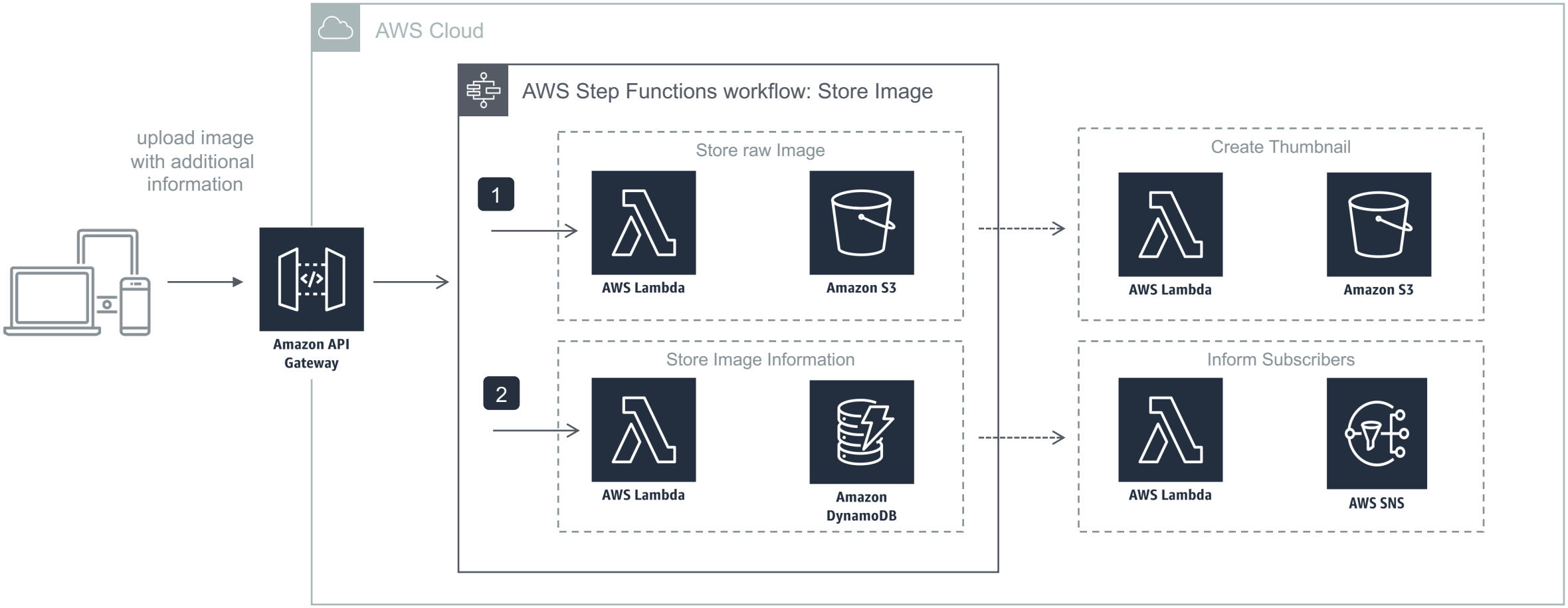
Upload Data & Image



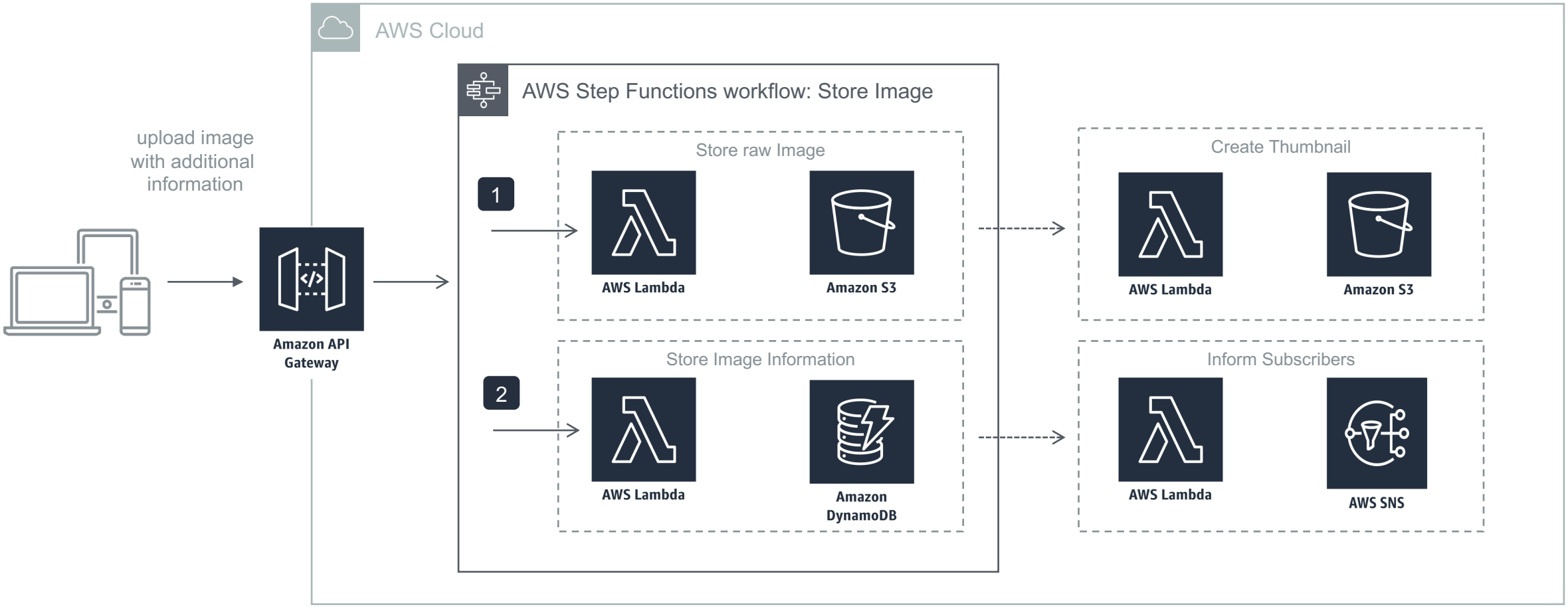
Upload Data & Image



Upload Data & Image

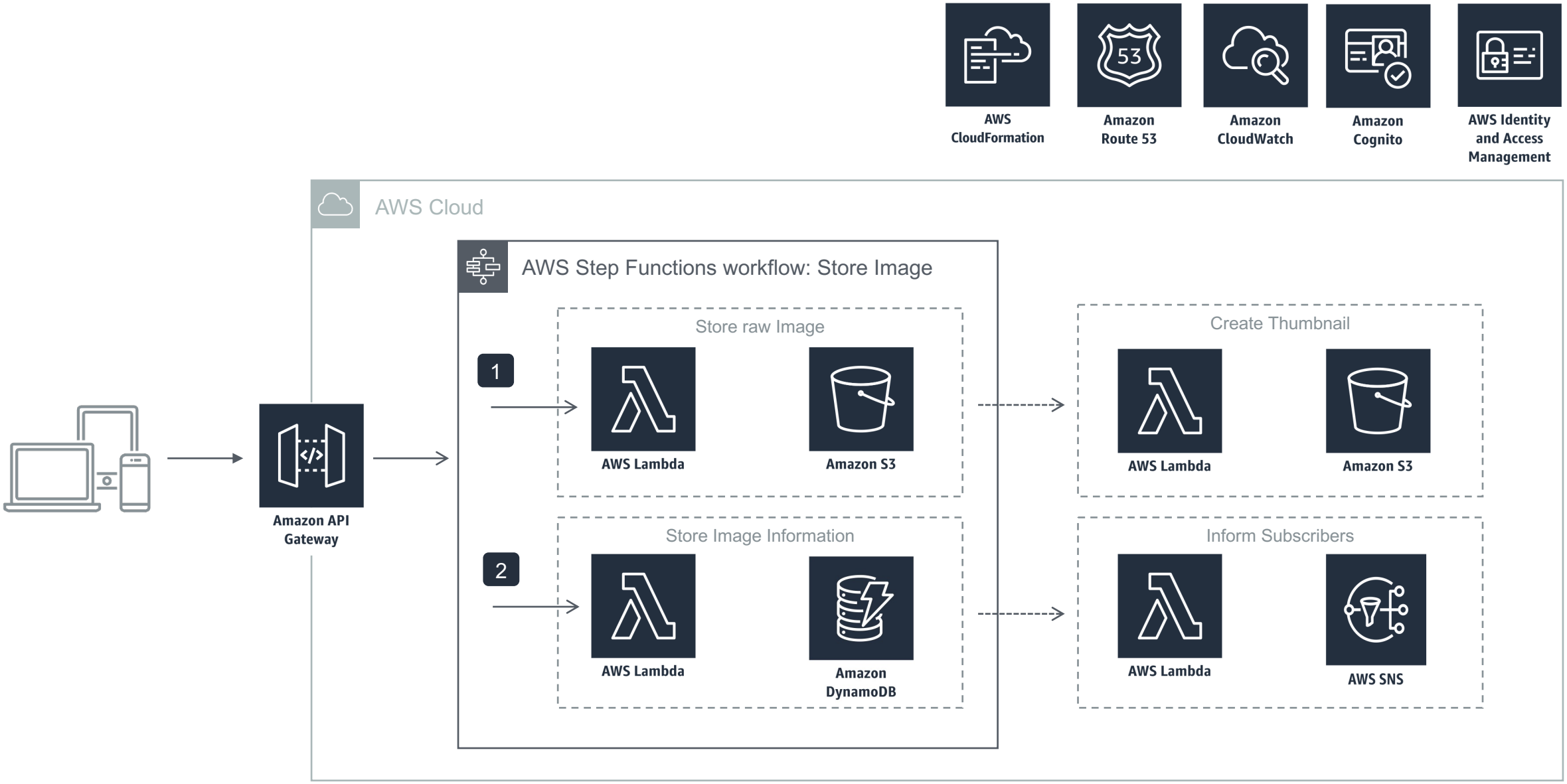


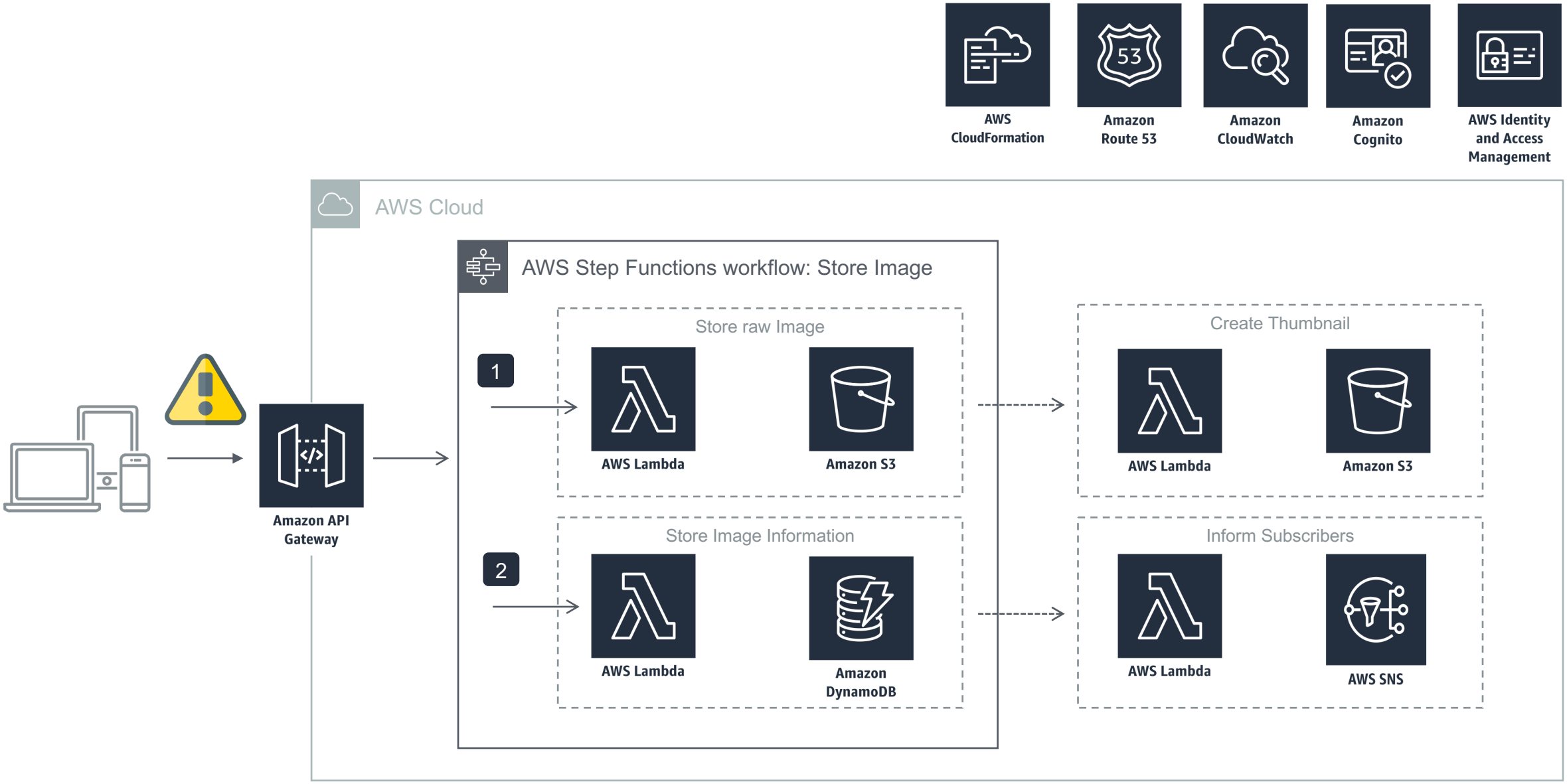
Upload Data & Image

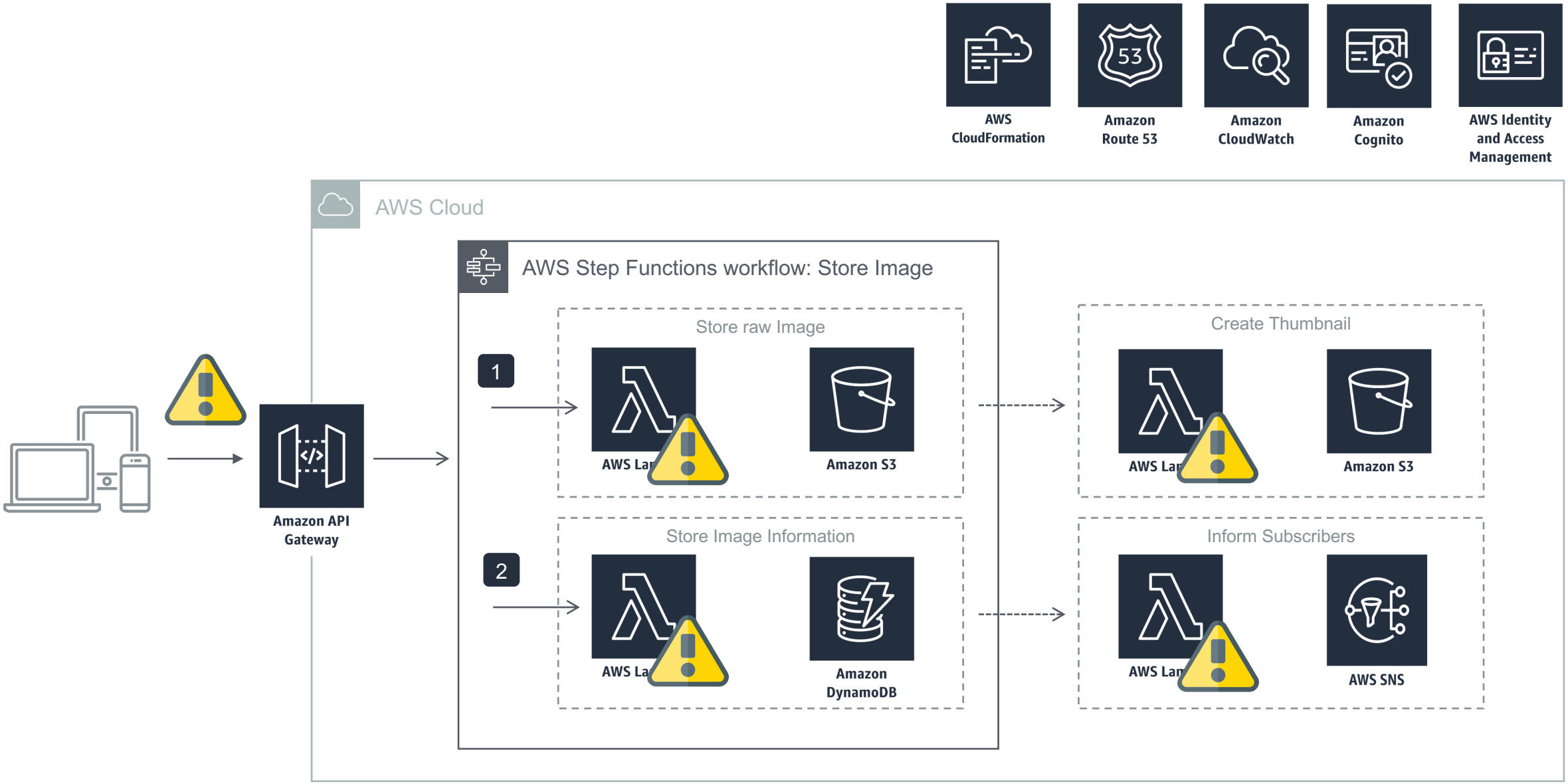


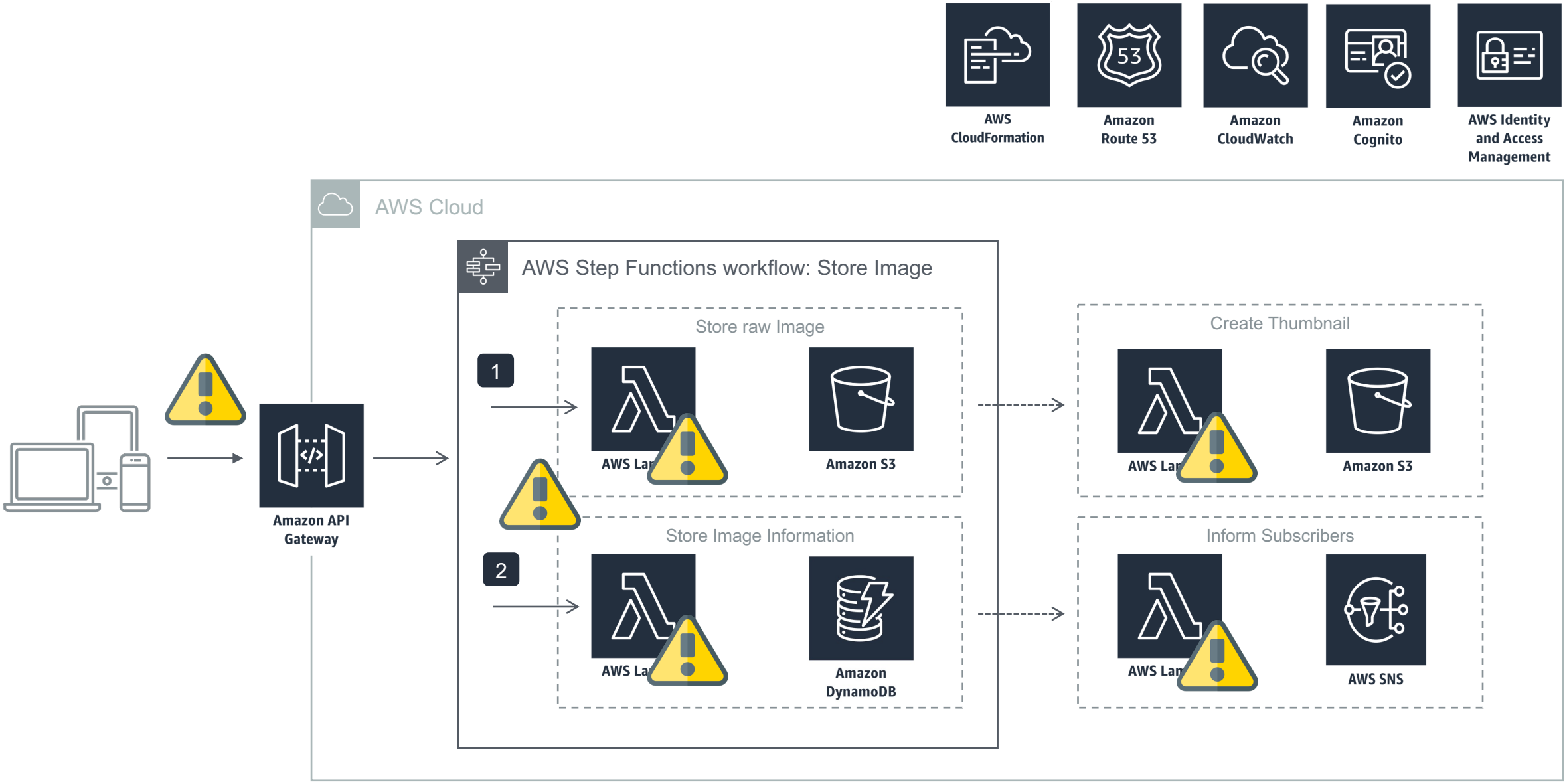
NO SERVERS
NO STRESS

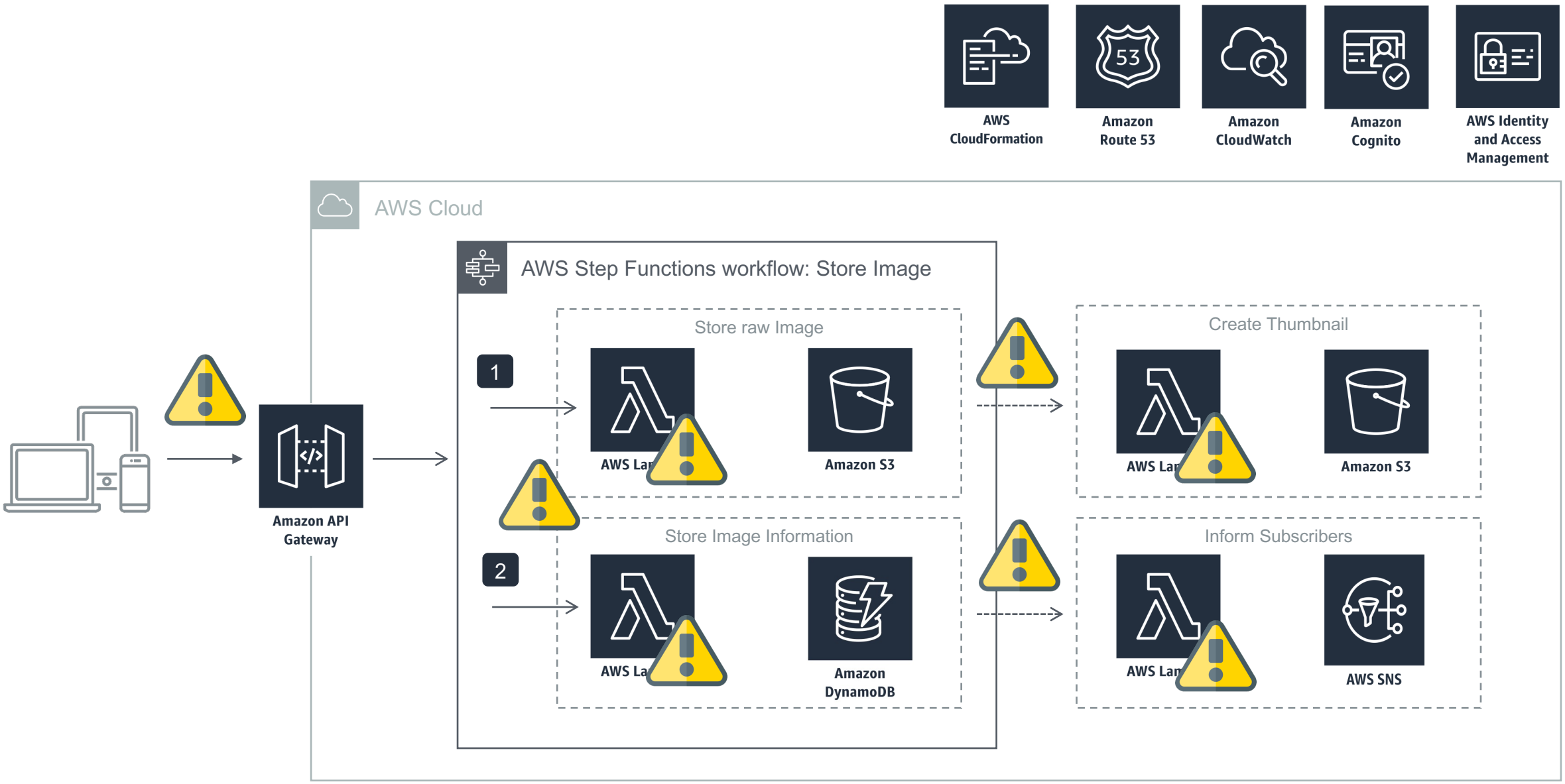
„Was soll da schon schiefgehen?“

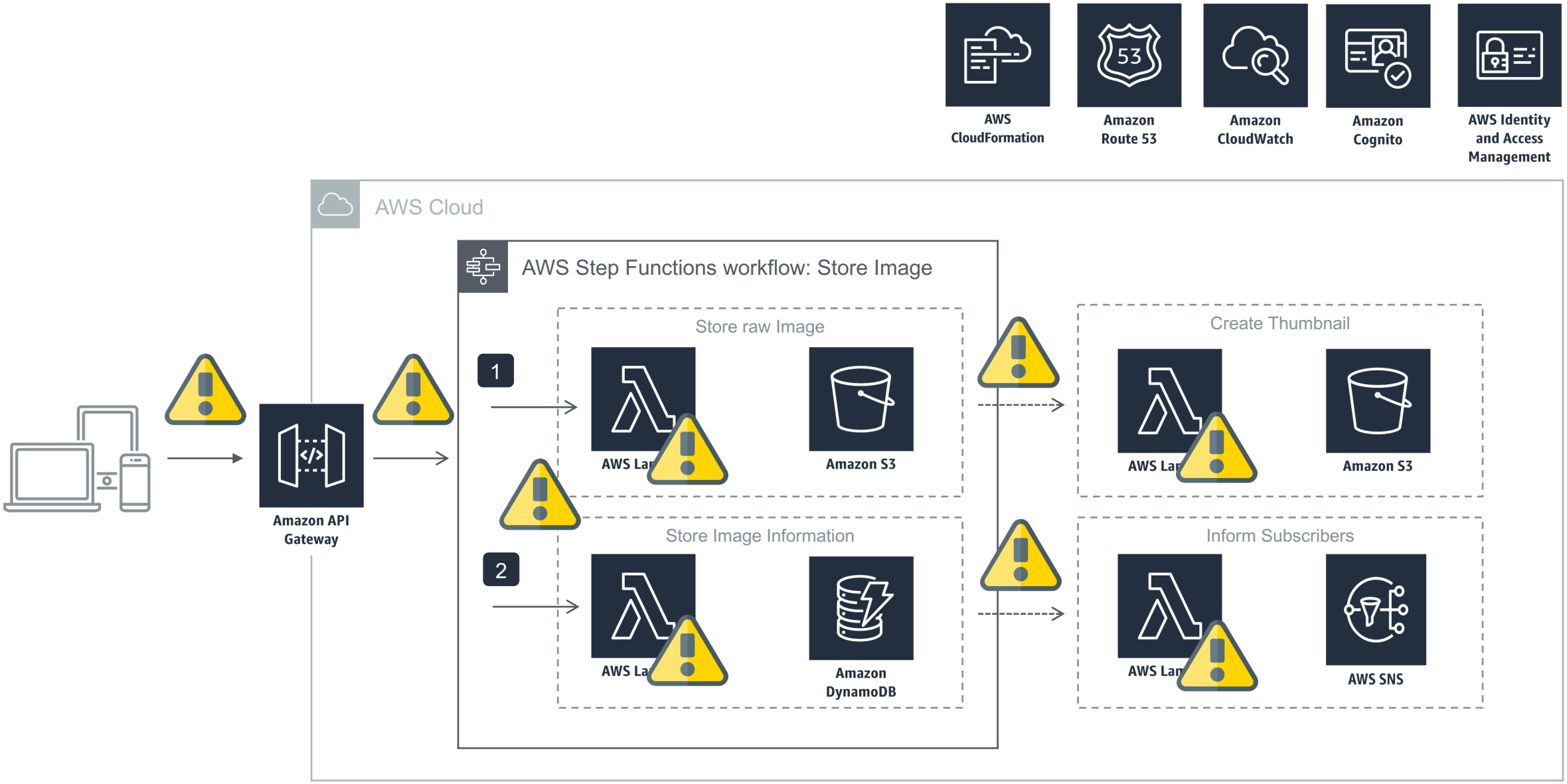

AWS CloudFormation
Amazon Route 53
Amazon CloudWatch
Amazon Cognito
AWS Identity and Access Management
AWS Cloud
AWS Step Functions workflow: Store Image
Store raw Image
1
AWS Lambda
Amazon S3
Store Image Information
2
AWS Lambda
Amazon DynamoDB
Amazon API Gateway
Create Thumbnail
AWS Lambda
Amazon S3
Inform Subscribers
AWS Lambda
AWS SNS

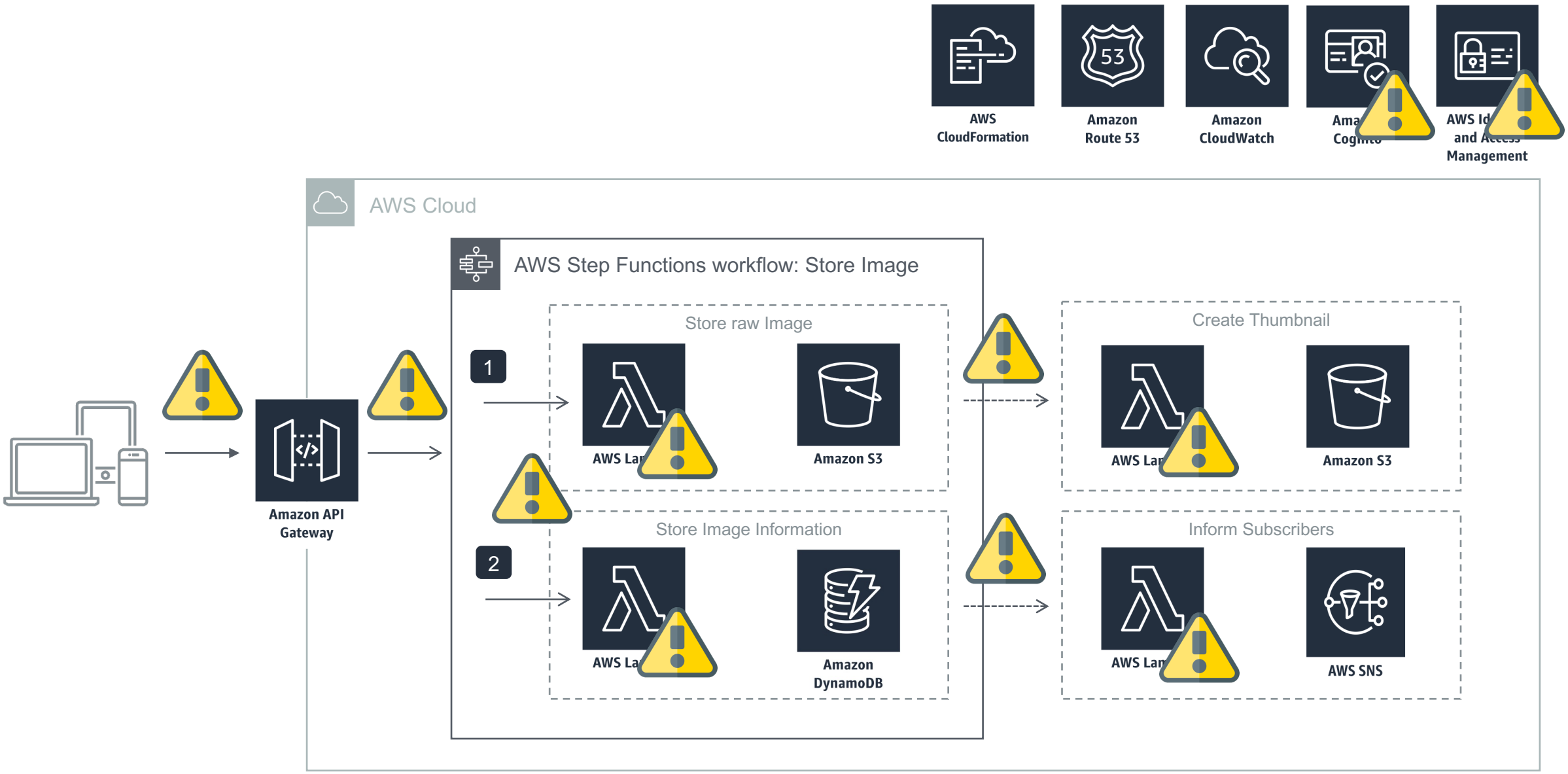

AWS CloudFormation
Amazon Route 53
Amazon CloudWatch
Amazon Cognito
AWS Identity and Access Management
AWS Cloud
AWS Step Functions workflow: Store Image
Store raw Image
1
AWS Lambda
Amazon S3
Store Image Information
2
AWS Lambda
Amazon DynamoDB
Amazon API Gateway
Create Thumbnail
AWS Lambda
Amazon S3
Inform Subscribers
AWS Lambda
AWS SNS

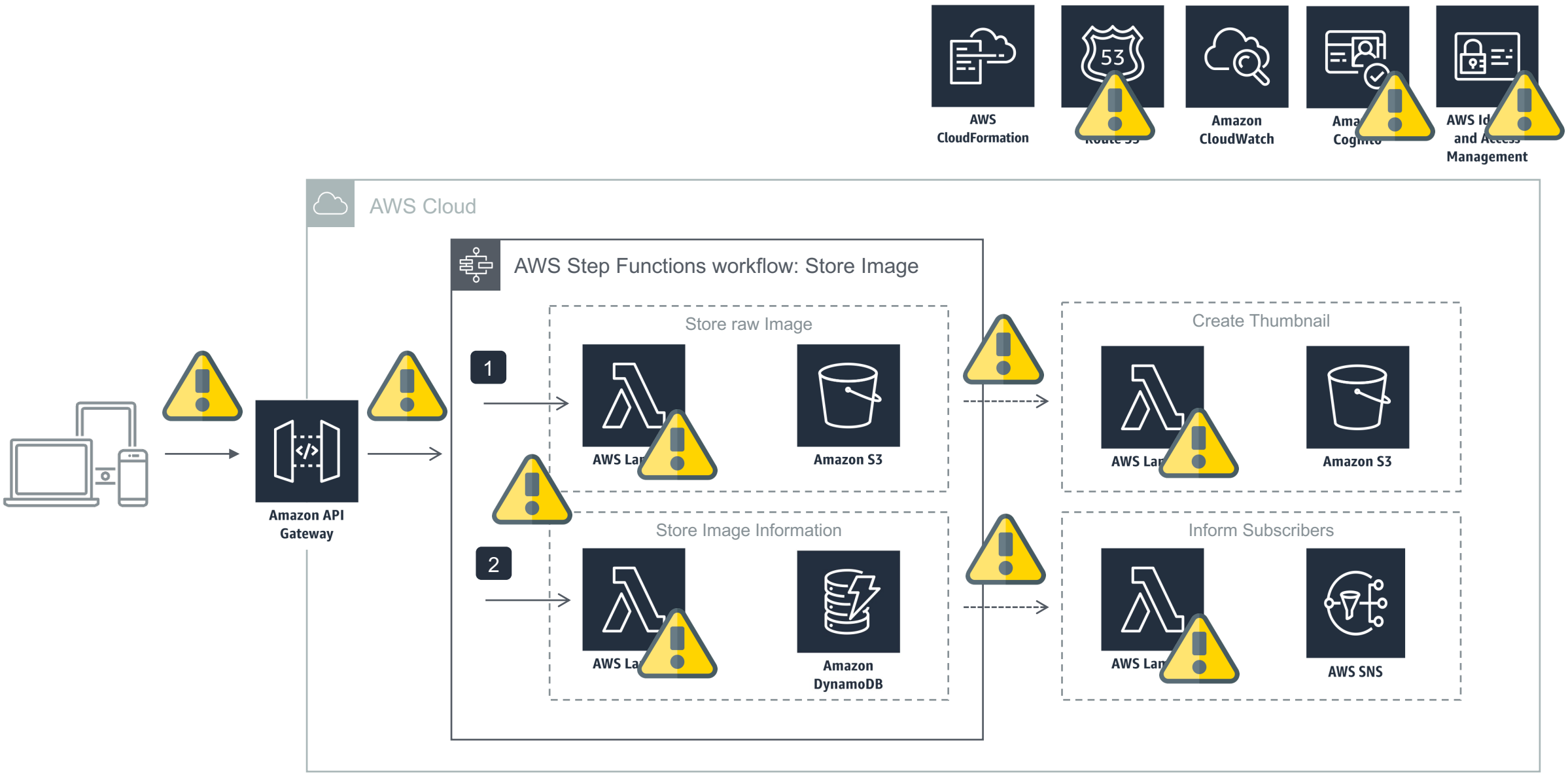

AWS CloudFormation
Amazon Route 53
Amazon CloudWatch
Amazon Cognito
AWS Identity and Access Management
AWS Cloud
AWS Step Functions workflow: Store Image
Store raw Image
Store Image Information
Create Thumbnail
Inform Subscribers
1
2
Amazon API Gateway
Amazon S3
Amazon DynamoDB
AWS SNS

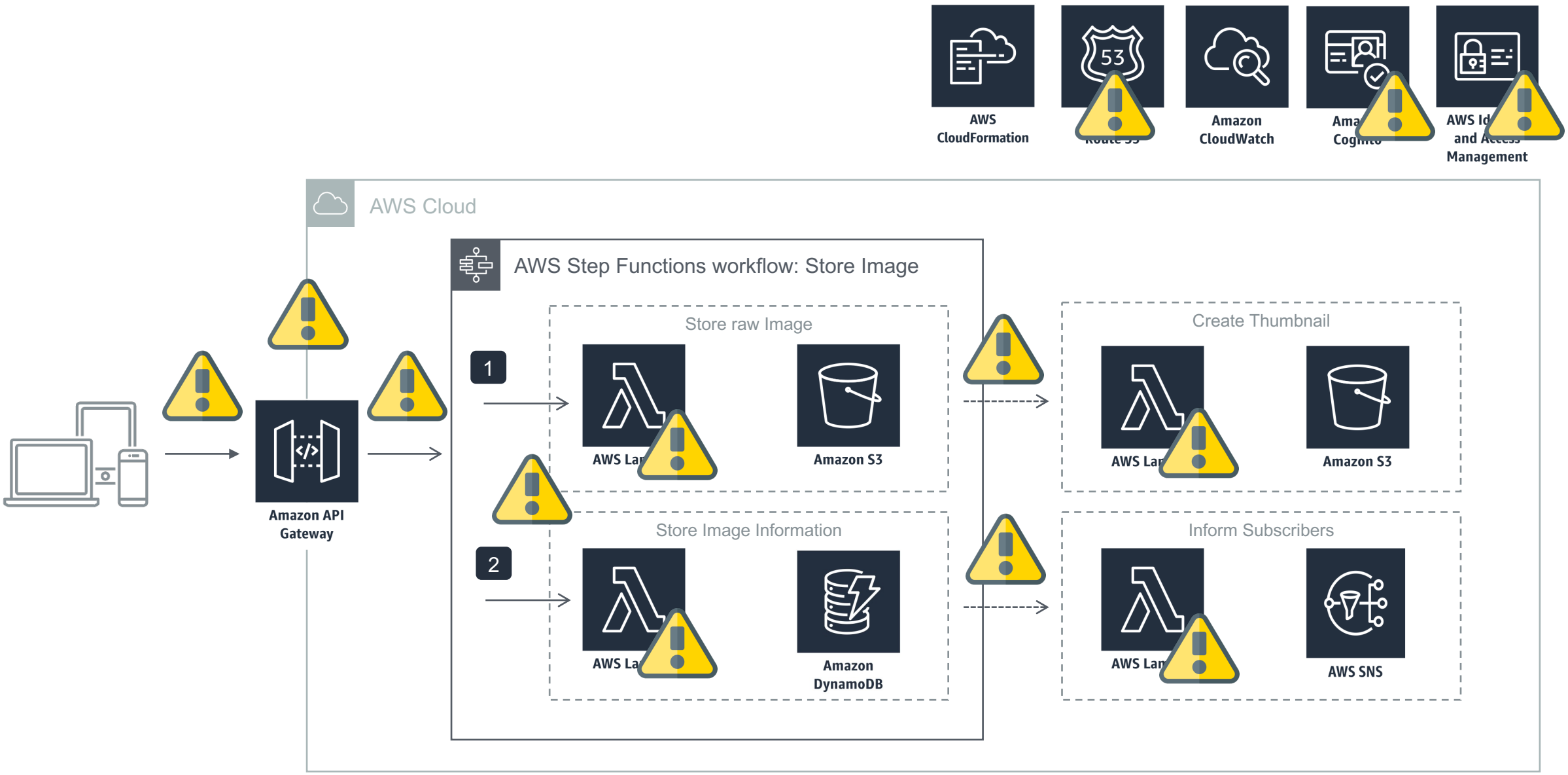

AWS CloudFormation
Amazon Route 53
Amazon CloudWatch
Amazon Cognito
AWS Identity and Access Management
AWS Cloud
AWS Step Functions workflow: Store Image
Store raw Image
Store Image Information
Create Thumbnail
Inform Subscribers
1
2
Amazon API Gateway
Amazon S3
Amazon DynamoDB
AWS SNS

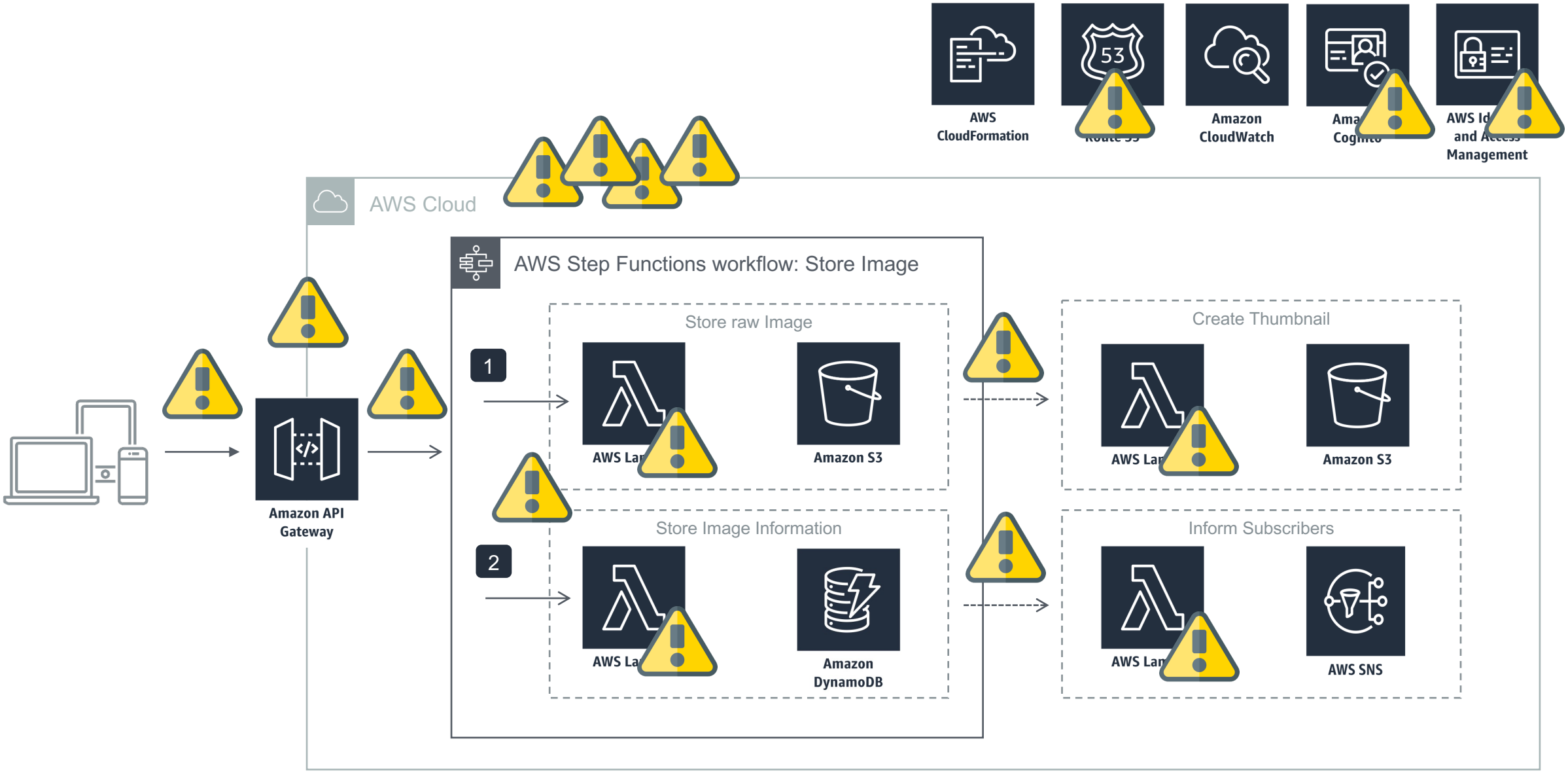

AWS CloudFormation
Amazon Route 53
Amazon CloudWatch
Amazon Cognito
AWS Identity and Access Management
AWS Cloud
AWS Step Functions workflow: Store Image
Store raw Image
Store Image Information
Create Thumbnail
Inform Subscribers
1
2
Amazon API Gateway
Amazon S3
Amazon DynamoDB
AWS SNS


AWS CloudFormation
Amazon Route 53
Amazon CloudWatch
Amazon Cognito
AWS Identity and Access Management
AWS Cloud
AWS Step Functions workflow: Store Image
Store raw Image
Store Image Information
Create Thumbnail
Inform Subscribers
1
2
Amazon API Gateway
Amazon S3
Amazon DynamoDB
AWS SNS


AWS CloudFormation
Amazon Route 53
Amazon CloudWatch
Amazon Cognito
AWS Identity and Access Management
AWS Cloud
AWS Step Functions workflow: Store Image
Store raw Image
Store Image Information
Create Thumbnail
Inform Subscribers
1
2
Amazon API Gateway
AWS Lambda
Amazon S3
Amazon DynamoDB
AWS SNS


AWS CloudFormation
Amazon CloudWatch
AWS Id
and Access
Management
AWS Cloud
AWS Step Functions workflow: Store Image
Store raw Image
Amazon S3
Create Thumbnail
Amazon S3
Store Image Information
Amazon DynamoDB
Inform Subscribers
AWS SNS
1
2
Amazon API Gateway


AWS CloudFormation
Amazon CloudWatch
AWS Id
and Access
Management
AWS Cloud
AWS Step Functions workflow: Store Image
Store raw Image
Amazon S3
Create Thumbnail
Amazon S3
Amazon API Gateway
1
2
Store Image Information
Amazon DynamoDB
Inform Subscribers
AWS SNS


AWS CloudFormation
Amazon CloudWatch
AWS Id and Access Management
AWS Cloud
AWS Step Functions workflow: Store Image
Store raw Image
Amazon S3
Store Image Information
Amazon DynamoDB
Create Thumbnail
Amazon S3
Inform Subscribers
AWS SNS
Amazon API Gateway
1
2

… schiefgehen?

- interne Programmierfehler
- Fehler in der Businesslogik
- unerwartete Calls / Payloads
- verlorene oder doppelte Events
- hohe Latenz & Timeouts
- Security Attacken (z.B. DoS/DDos)
- SLAs / Usage Plans
- Workload Peaks
- …

„Run **Code**,
not **Servers**!“

“Run **your business** code **highly-available** in the **cloud** in response to **events** and **scale** **without any servers** to manage.“*

*(AWS Lambda product description)

“Run **your business** code **highly distributed** and **event driven** in a **non transparent** environment with **no single point of control**.“*

*(my personal interpretation)

Wie teste ich?

meine Serverless Application

Was, wann, wie und wo sollte ich testen, um …

- **Vertrauen in meinen Code** zu gewinnen
- das **Risiko von Fehlern** zu minimieren*

* vor allem in Produtktion

Testen in der **Serverless** Welt

„The biggest complexity is not within the function itself, but in how it interacts with other functions and services (a.k.a. cloud components).“


Best
Practice

Testing Best Practices

*„Welche Art von ‚Benchmarks‘ wollen wir für unser **Testing**?“*

- **funktionale Änderungen** schnell/kosteneffizient testen
- **integrative Änderungen** schnell/kosteneffizient testen
- **integrative Änderungen** so „real“ wie möglich testen
- **Use-Cases** und **User-Stories** so „real“ wie möglich testen

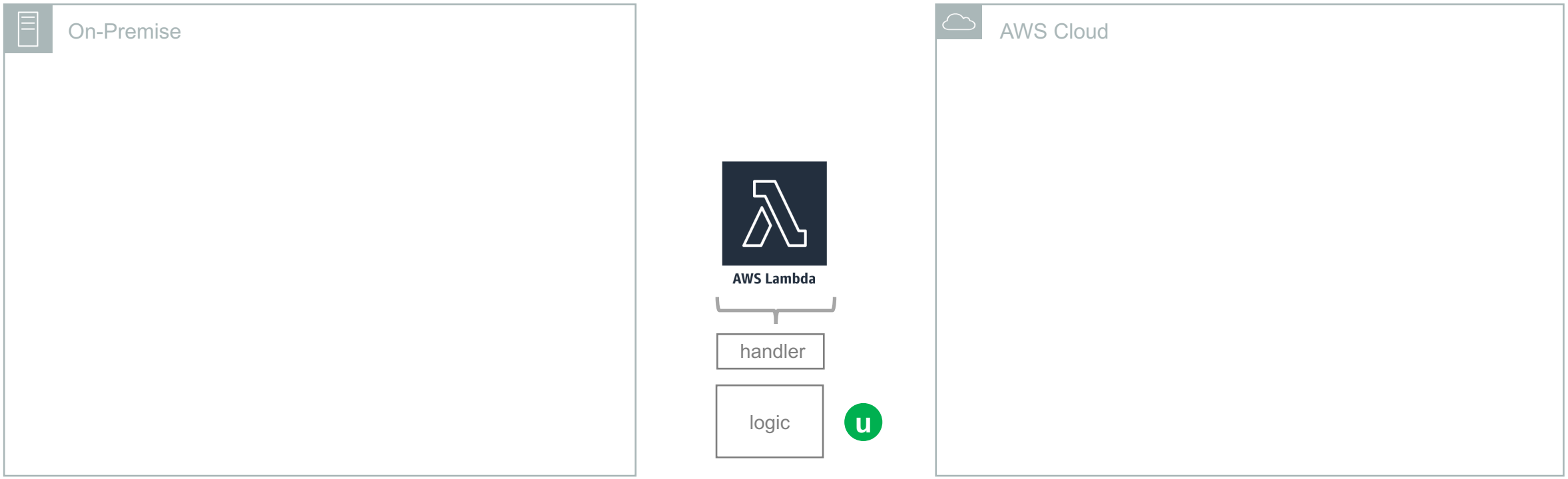
Testing Best Practices

u Kandidat **für Unit Tests**

i Kandidat **für Integration Tests**

e Kandidat **für End-to-Ende Tests**

#1 Trennen von Businesslogik und Infrastruktur



Infrastructure

Business Logic

```java
// AWS lambda handler generating a "greet" response
public GreetingResponse handleRequest(GreetingRequest name, Context context) {

    LambdaLogger logger = context.getLogger();

    // Print info from the context object
    logger.log("Function name: " + context.getFunctionName());
    logger.log(String.format("firstName: %s", name.getFirstName()));
    logger.log(String.format("lastName: %s", name.getLastName));

    if (firstName != null && lastName != null) {
        return new GreetingResponse(String.format("Hello, %s %s!
                                                  , firstName, lastName));
    } else {
        return new GreetingResponse(DEFAULT_GREET);
    }
}
```

Infrastructure

Business Logic

```java
// AWS lambda handler generating a "greet" response
public GreetingResponse handleRequest(GreetingRequest name, Context context) {

    logContextInfo(context);

    String greeting = GreetingService.greet(name.getFirstName(),
                                            name.getLastName());
    return new GreetingResponse(greeting);
}

// Print info from the context object
private void logContextInfo(Context context) {
    LambdaLogger logger = context.getLogger();
    logger.log("Function name: " + context.getFunctionName());
}
```

Business Logic Tests

```java
@Test
void whenGreetWithFirstNameAndLastName_thenCorrect() {

    // given
    String firstName = "Lars";
    String lastName = "Roewekamp";

    // when
    String greetResult = GreetingService.greet(firstName, lastName);

    // then
    assertEquals("Hello, Lars Roewekamp! i am pleased to meet you.", greetResult);
}

@Test
void whenGreetWithoutFirstName_thenInCorrect() { ... }

@Test
void whenGreetWithoutLastName_thenInCorrect() { ... }
```

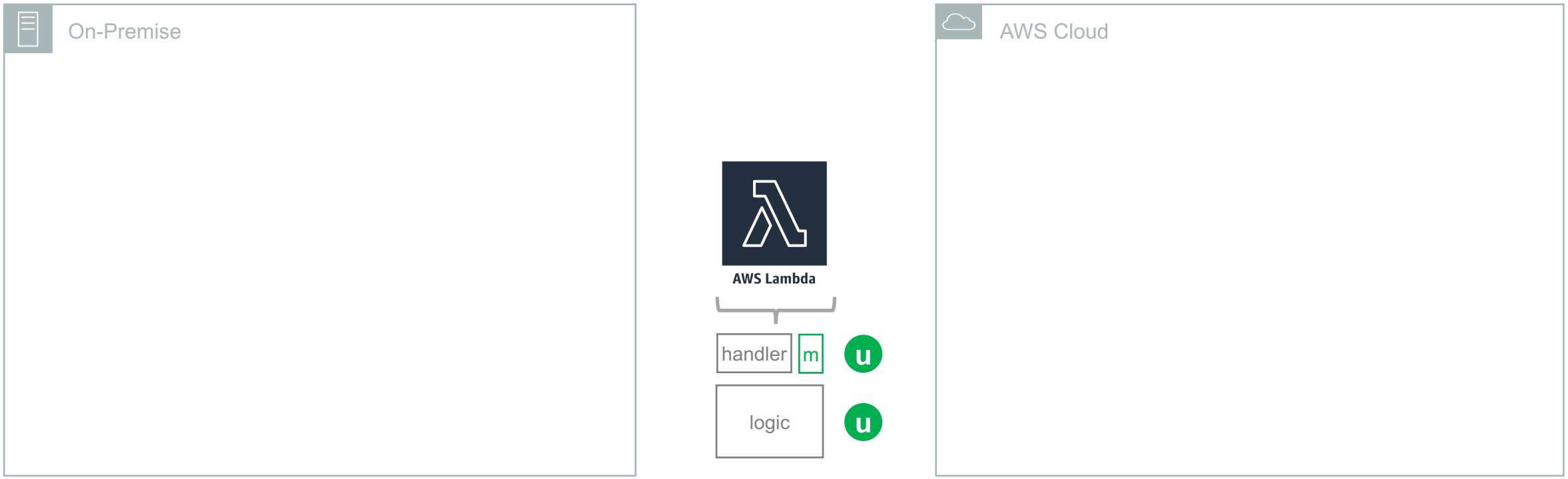
Testing Best Practices

u Kandidat **für Unit Tests**

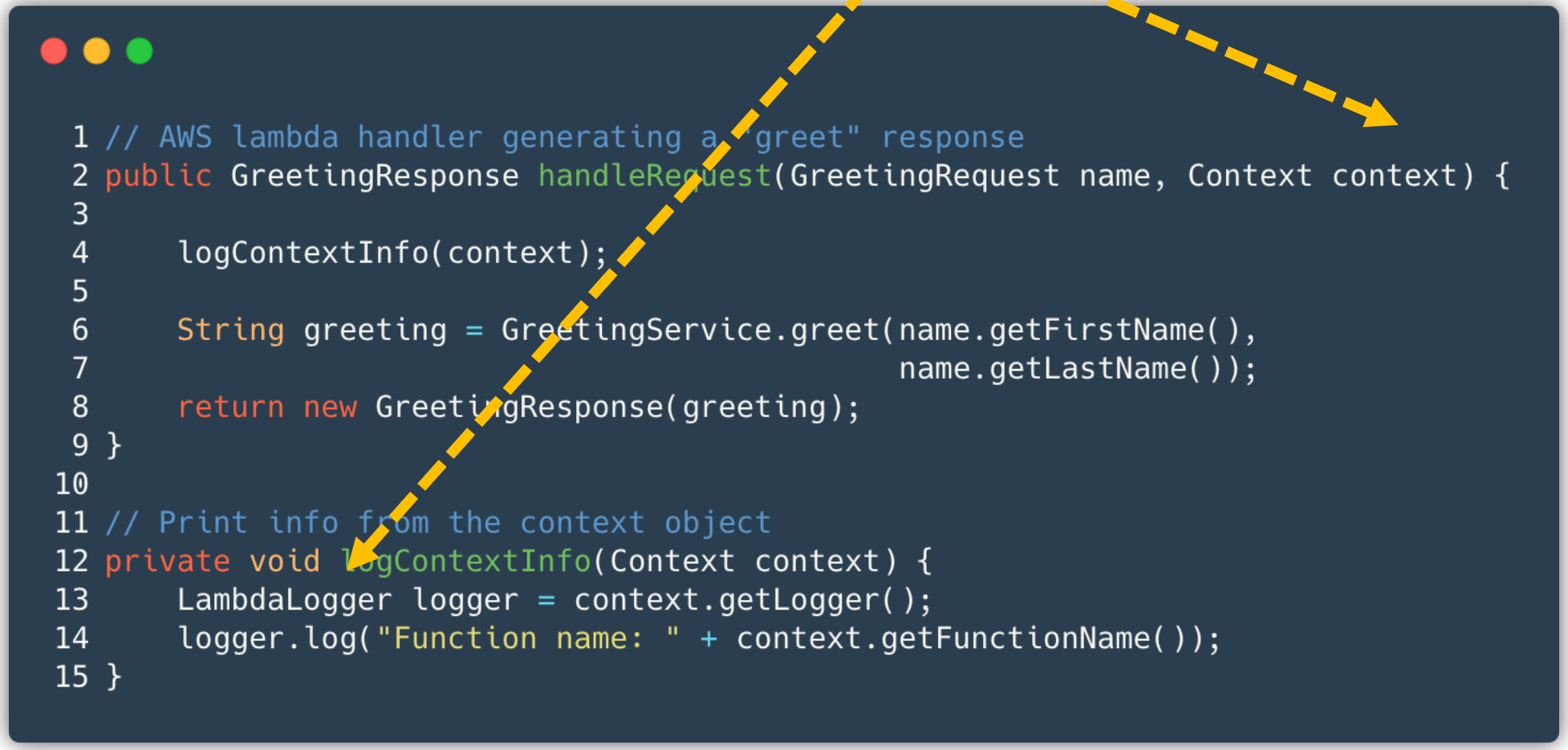
i Kandidat **für Integration Tests**

e Kandidat **für End-to-Ende Tests**

#2 Cloud-Infrastruktur Komponenten mocken

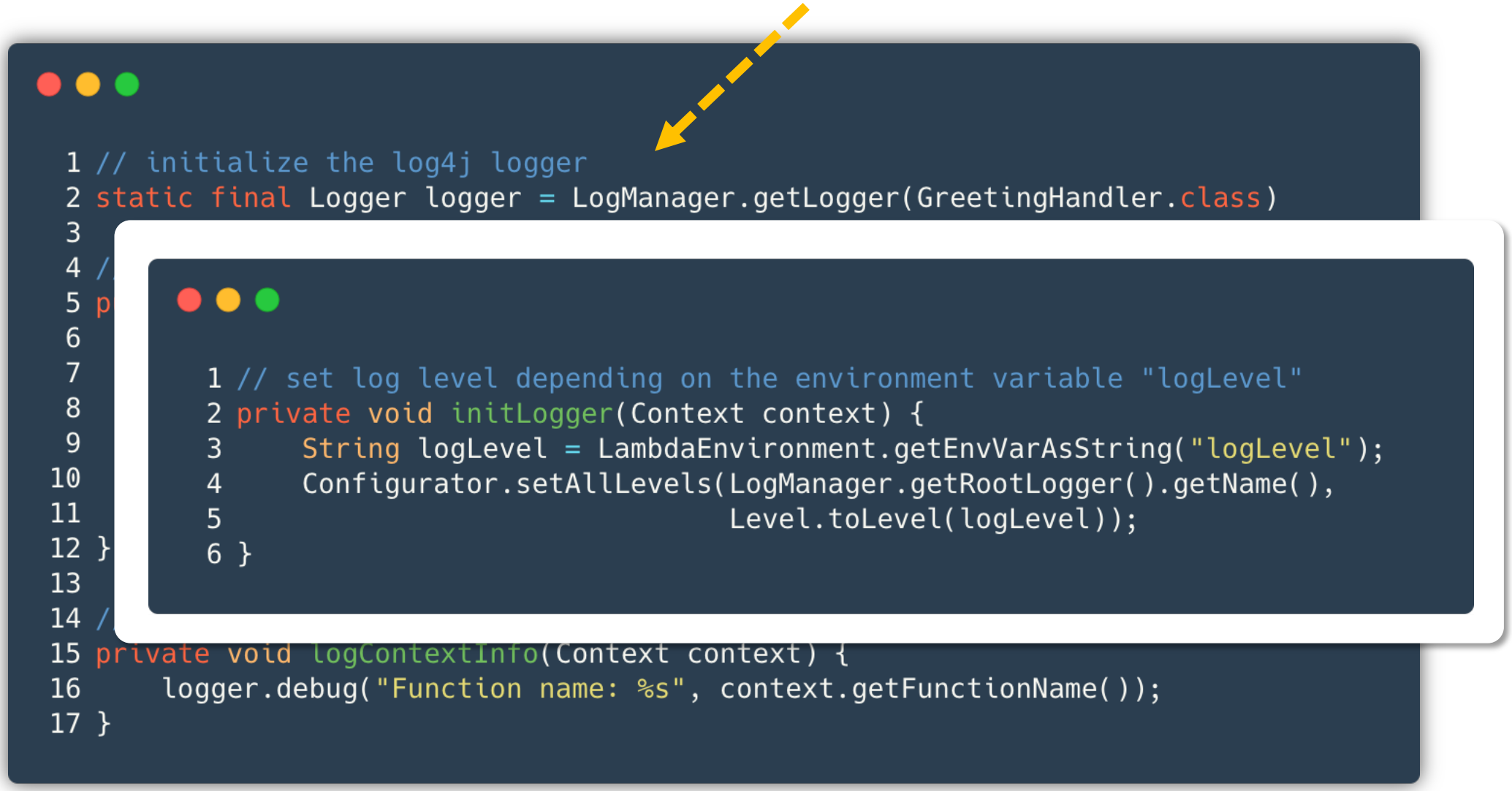


Infrastructure



```java
// AWS lambda handler generating a "greet" response
public GreetingResponse handleRequest(GreetingRequest name, Context context) {

    logContextInfo(context);

    String greeting = GreetingService.greet(name.getFirstName(),
                                            name.getLastName());
    return new GreetingResponse(greeting);
}

// Print info from the context object
private void logContextInfo(Context context) {
    LambdaLogger logger = context.getLogger();
    logger.log("Function name: " + context.getFunctionName());
}
```

AWS agnostic logger (log4j)



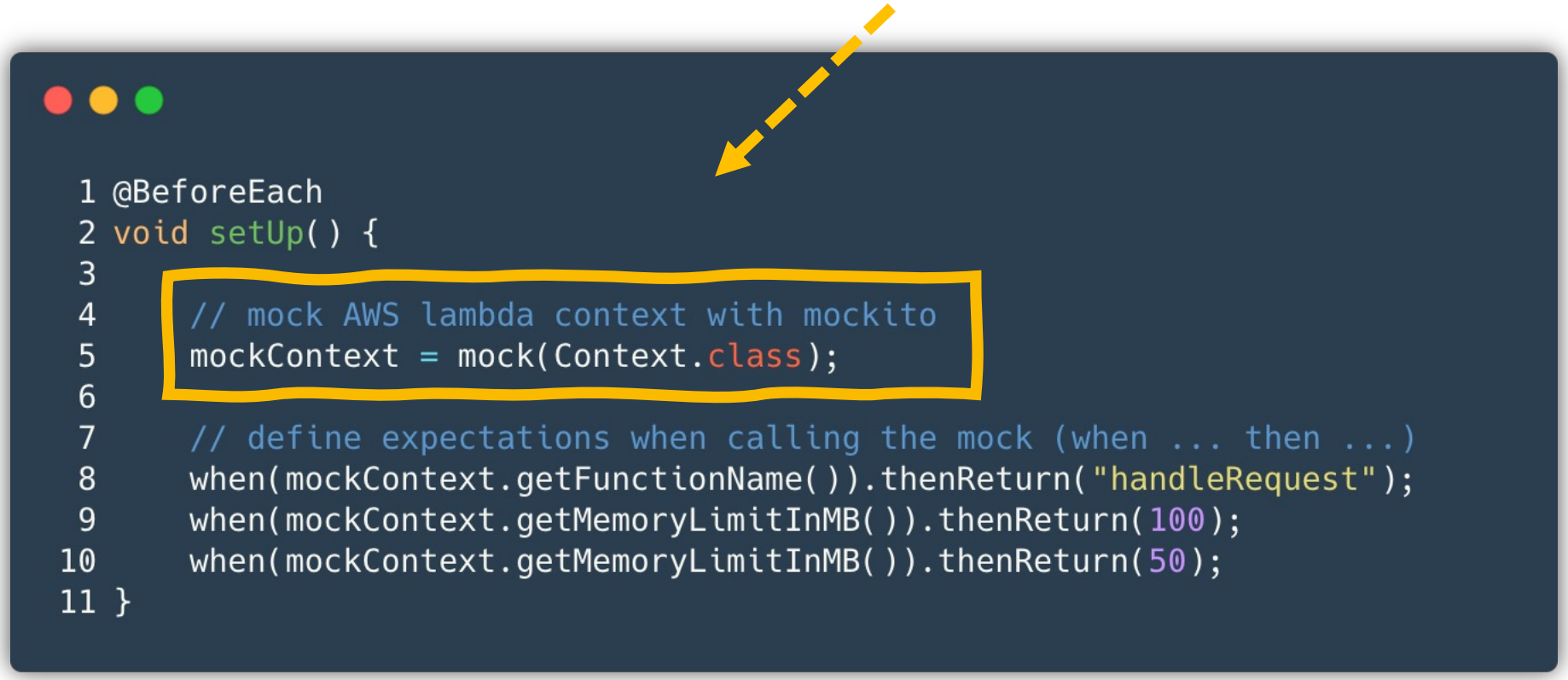
```java
// initialize the log4j logger
static final Logger logger = LogManager.getLogger(GreetingHandler.class)

/
p

}

/
private void logContextInfo(Context context) {
    logger.debug("Function name: %s", context.getFunctionName());
}
```

```java
// set log level depending on the environment variable "logLevel"
private void initLogger(Context context) {
    String logLevel = LambdaEnvironment.getEnvVarAsString("logLevel");
    Configurator.setAllLevels(LogManager.getRootLogger().getName(),
                              Level.toLevel(logLevel));
}
```

fake infrastructure component (Context)

```
1 @Test
2 void whenHandelRequestWithFirstNameAndLastName_thenCorrect() {
3
4     // given
5     String firstName = "Lars";
6     String lastName = "Roewekamp";
7     GreetingRequest request = new GreetingRequest(firstName, lastName);
8     GreetingRequestHandler clazzUnderTest = new GreetingRequestHandler();
9     Context context = new TestContext() { ... };
10
11    // when
12    GreetingResponse response = clazzUnderTest.handleRequest(request, context);
13
14    // then
15    assertEquals("Hello, Lars Roewekamp! i am pleased to meet you.",
16                 response.getGreeting());
17 }
```

mock infrastructure component (Context)



```
1 @BeforeEach
2 void setUp() {
3
4     // mock AWS lambda context with mockito
5     mockContext = mock(Context.class);
6
7     // define expectations when calling the mock (when ... then ...)
8     when(mockContext.getFunctionName()).thenReturn("handleRequest");
9     when(mockContext.getMemoryLimitInMB()).thenReturn(100);
10    when(mockContext.getMemoryLimitInMB()).thenReturn(50);
11 }
```

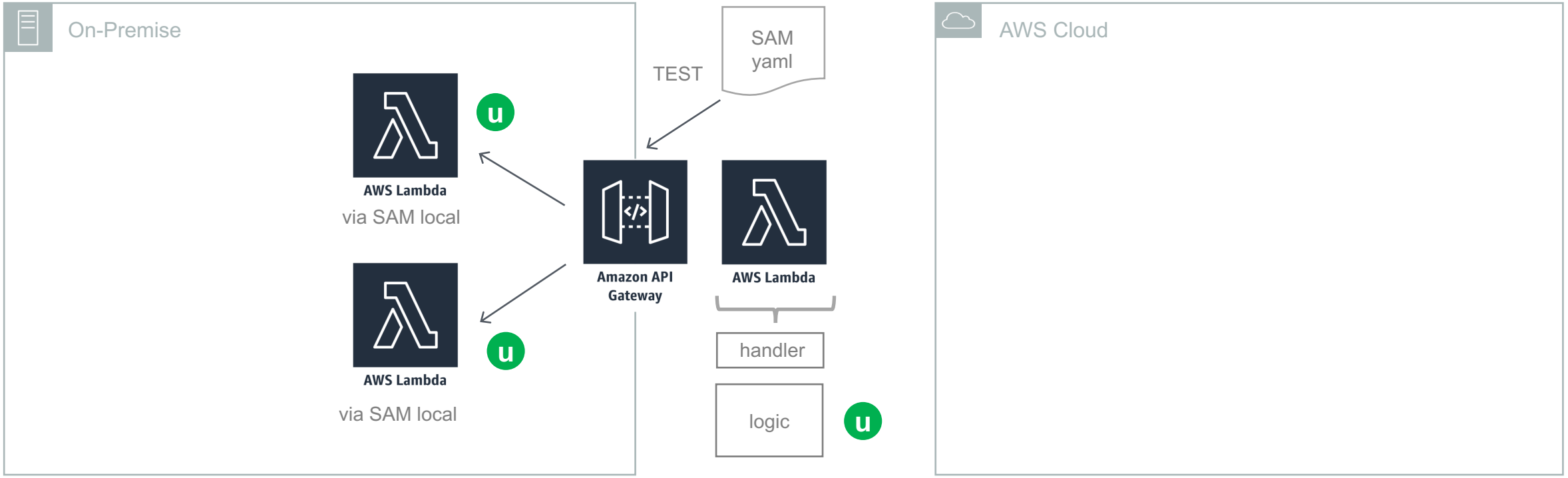
Testing Best Practices

u Kandidat **für Unit Tests**

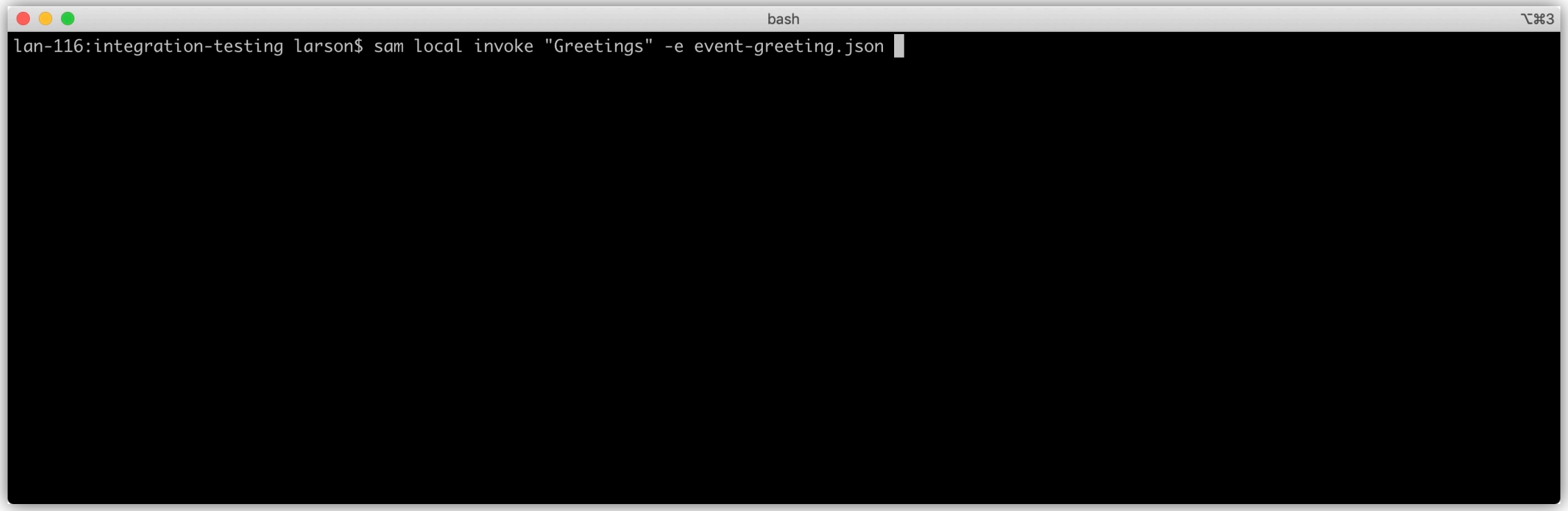
i Kandidat **für Integration Tests**

e Kandidat **für End-to-Ende Tests**

#3 Lokale Umgebung für funktionale Tests verwenden (z.B. SAM local)



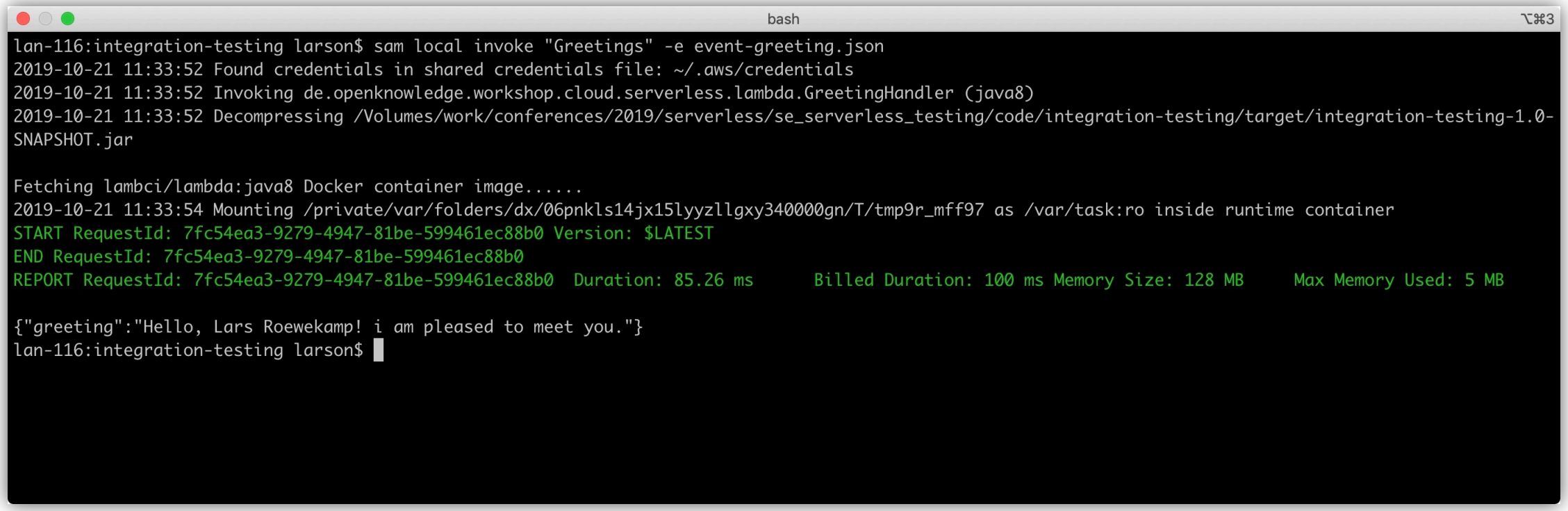
```yaml
Description: AWS Lambda Greeting Service
Resources:
  Greetings:
    Type: AWS::Serverless::Function
    Properties:
      Handler: de.openknowledge.workshop.cloud.serverless.lambda.GreetingHandler
      CodeUri: ./target/integration-testing-1.0-SNAPSHOT.jar
      Runtime: java8
      Timeout: 300
      Environment:
        Variables:
          ENVIRONMENT: "test"
      Events:
        CreateResource:
          Type: Api
          Properties:
            Path: /greetings/
            Method: post
```



$ sam local invoke "Greetings" -e event-greeting.json --env-vars env.json

function name

payload for function



$ sam local invoke "Greetings" -e event-greeting.json --env-vars env.json

function name

payload for function

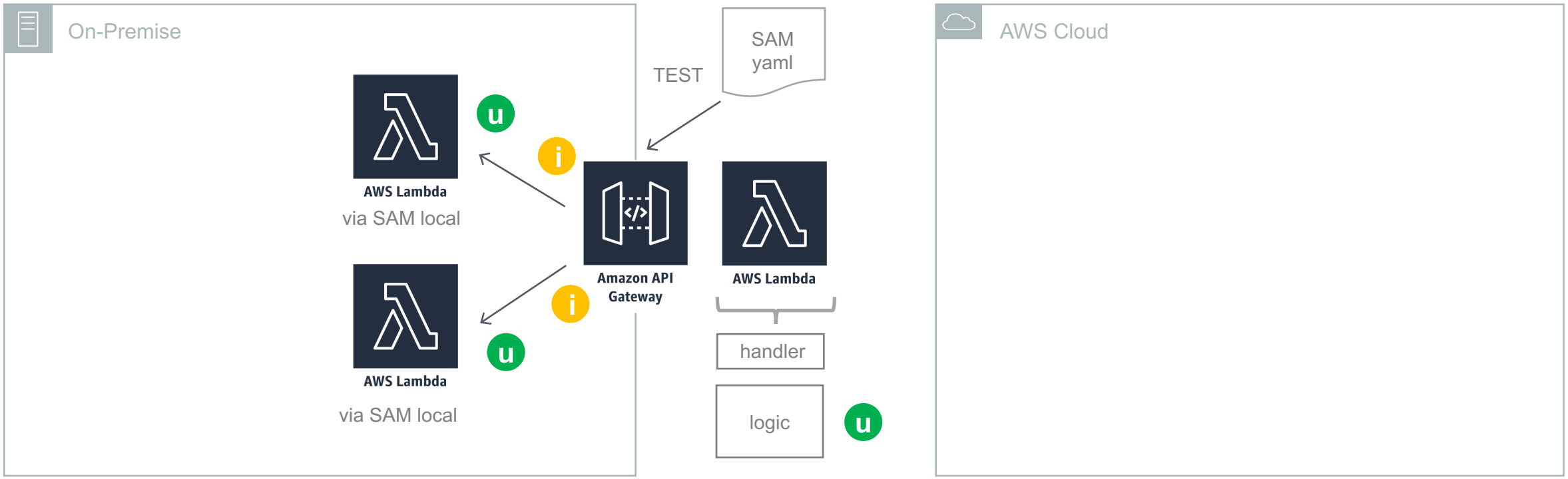
Testing Best Practices

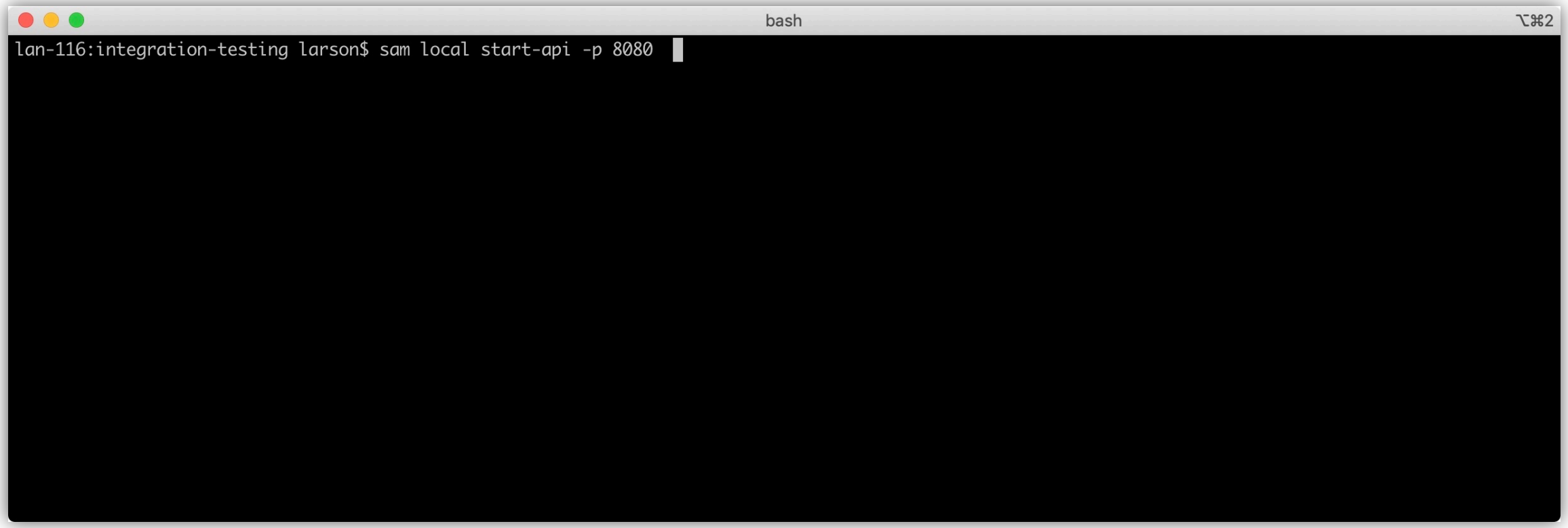
u Kandidat **für Unit Tests**

i Kandidat **für Integration Tests**

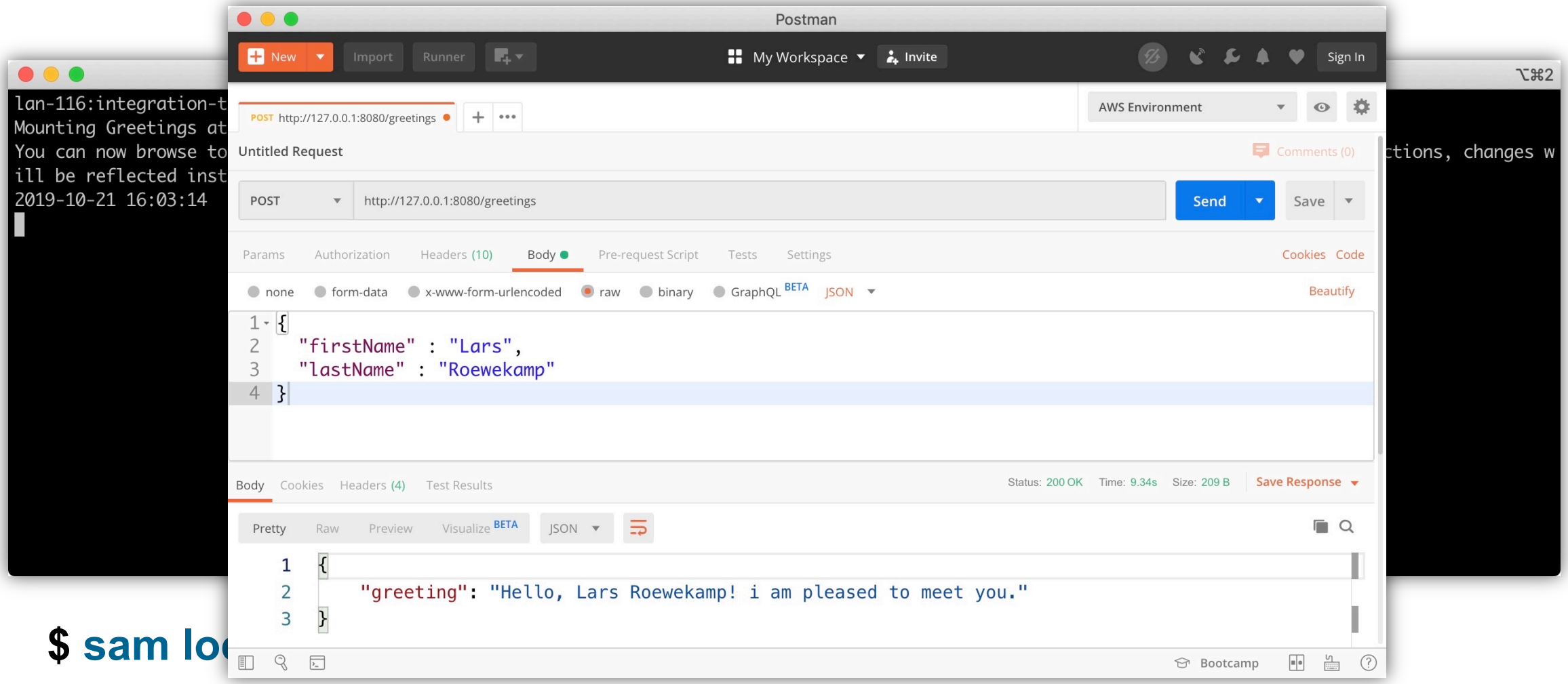
e Kandidat **für End-to-Ende Tests**

#4 Lokale Umgebung zum Triggern von Integration Tests verwenden

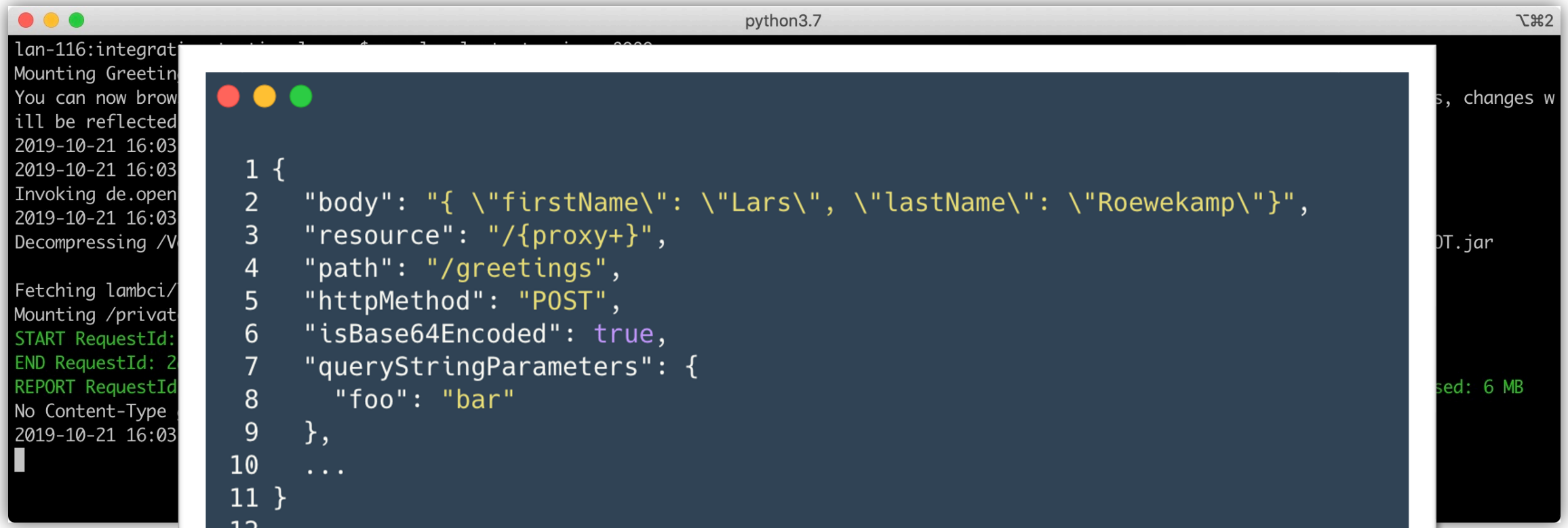




$ sam local start-api –p 8080


lan-116:integration-t
Mounting Greetings at
You can now browse to
ill be reflected inst
2019-10-21 16:03:14
⌥⌘2
ctions, changes w
Postman
New
Import
Runner
My Workspace
Invite
Sign In
POST http://127.0.0.1:8080/greetings
AWS Environment
Untitled Request
Comments (0)
POST
http://127.0.0.1:8080/greetings
Send
Save
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
Cookies
Code
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL BETA
JSON
Beautify
{
"firstName" : "Lars",
"lastName" : "Roewekamp"
}
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 9.34s
Size: 209 B
Save Response
Pretty
Raw
Preview
Visualize BETA
JSON
{
"greeting": "Hello, Lars Roewekamp! i am pleased to meet you."
}
Bootcamp

$ sam lo


python3.7
⌥⌘2
lan-116:integrat
Mounting Greetin
You can now brow
ill be reflected
2019-10-21 16:03
2019-10-21 16:03
Invoking de.open
2019-10-21 16:03
Decompressing /V
Fetching lambci/
Mounting /privat
START RequestId:
END RequestId: 2
REPORT RequestId
No Content-Type
2019-10-21 16:03
s, changes w
OT.jar
sed: 6 MB
{
"body": "{ \"firstName\": \"Lars\", \"lastName\": \"Roewekamp\"}",
"resource": "/{proxy+}",
"path": "/greetings",
"httpMethod": "POST",
"isBase64Encoded": true,
"queryStringParameters": {
"foo": "bar"
},
...
}

$ sam

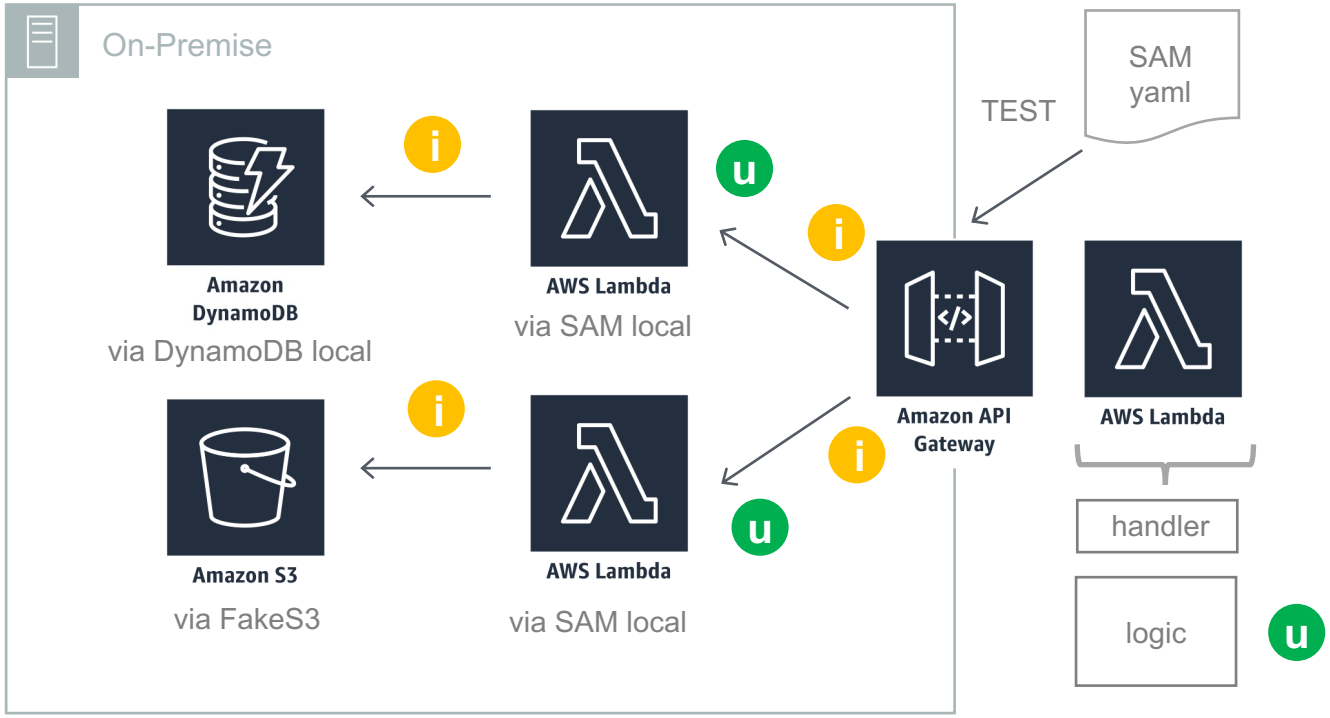
Testing Best Practices

u Kandidat **für Unit Tests**

i Kandidat **für Integration Tests**

e Kandidat **für End-to-Ende Tests**

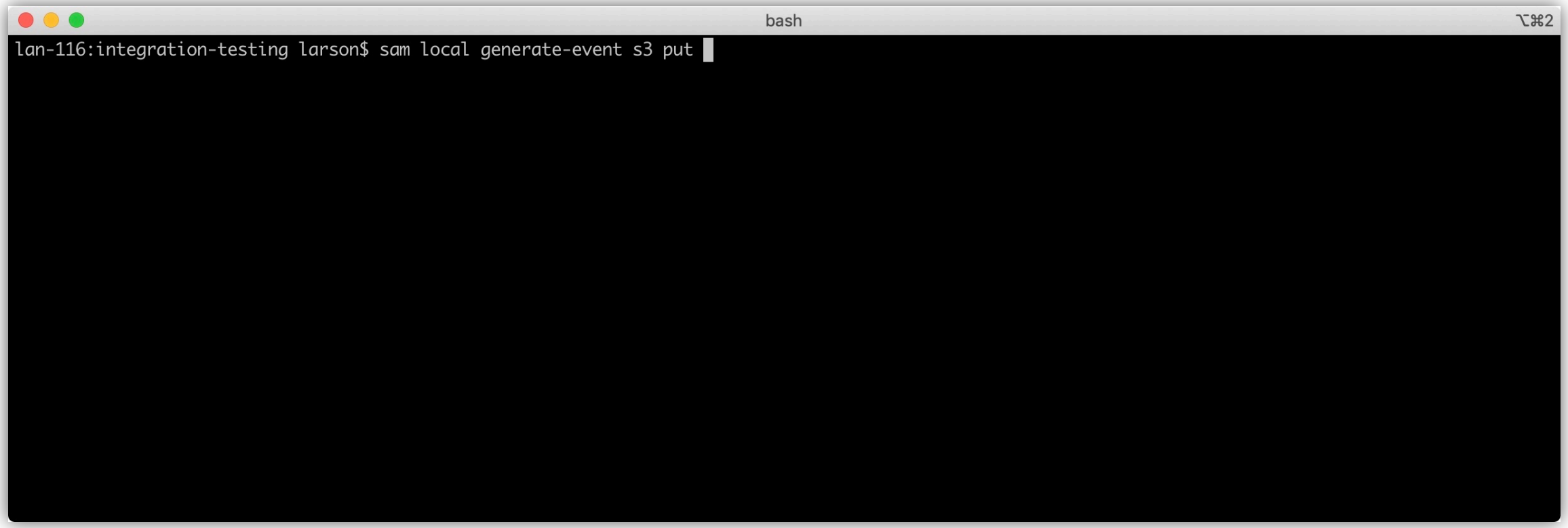
#5 Lokale Cloud-Komponenten für Integration Tests*



AWS Cloud

WARNUNG: lokale Cloud Komponenten können lediglich funktionale Korrektheit sicherstellen, nicht aber infrastrukturelle, wie z.B. DLQs, Timeouts, Throttling, SLAs, …

Simulate Component Event to trigger Lambda



$ sam local generate-event [SERVICE] [OPTION]

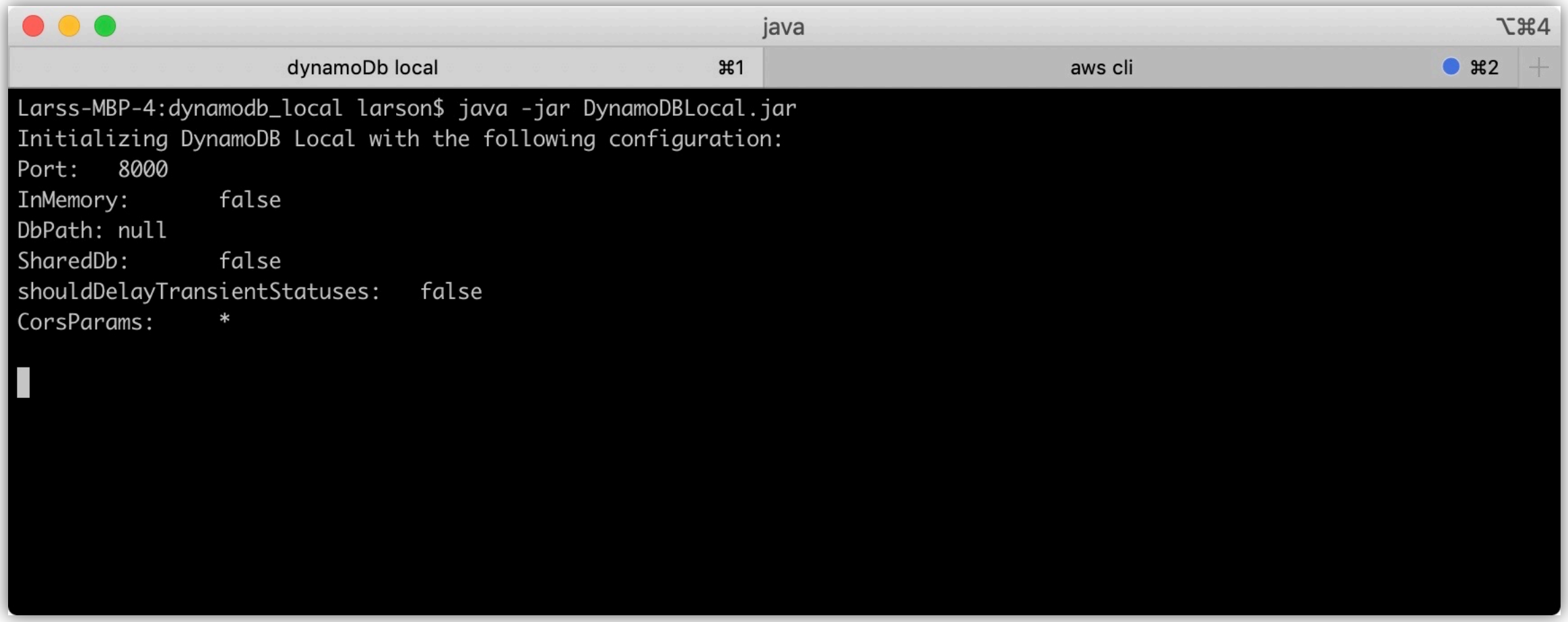
Simulate Component Event to trigger Lambda

bash ⌥⌘2

```
lan-116:integration-testing larson$ sam local generate-event s3 put
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "EXAMPLE123456789",
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
```

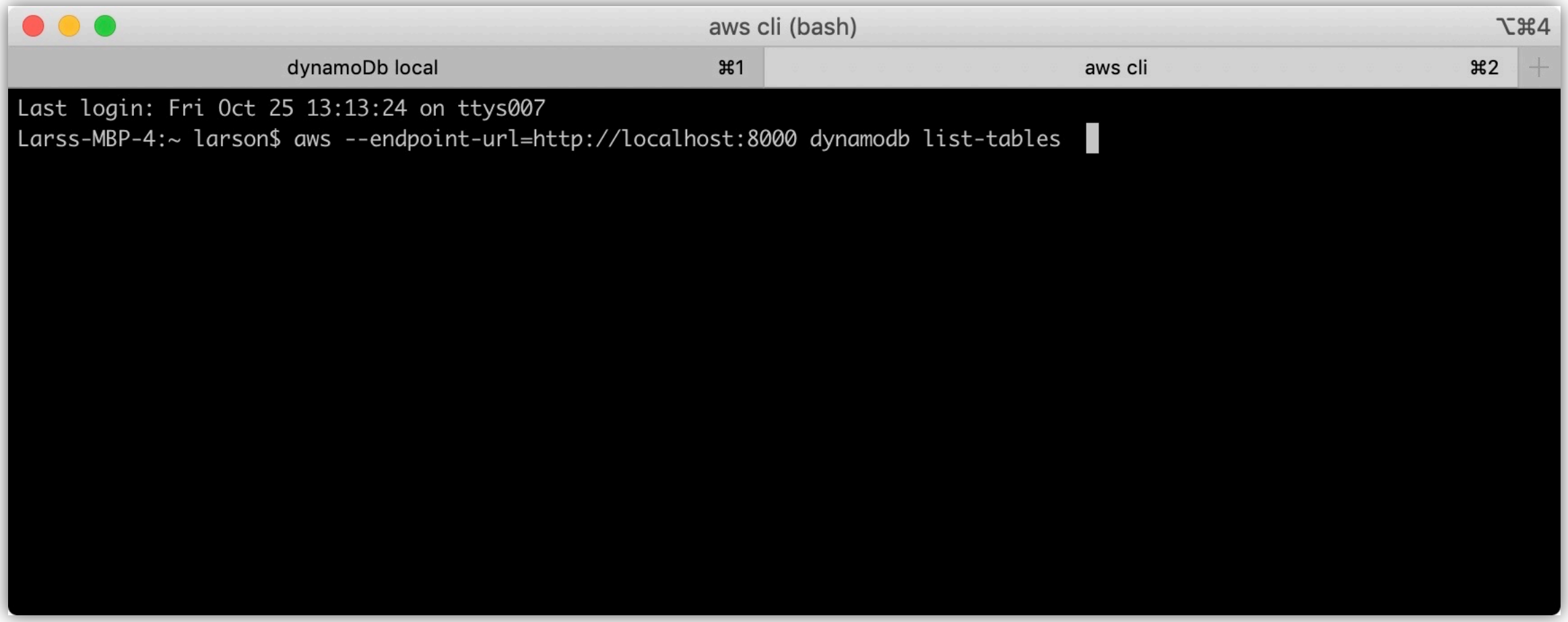
$ sam local generate-event [SERVICE] [OPTION]

Simulate Component triggered by Lambda



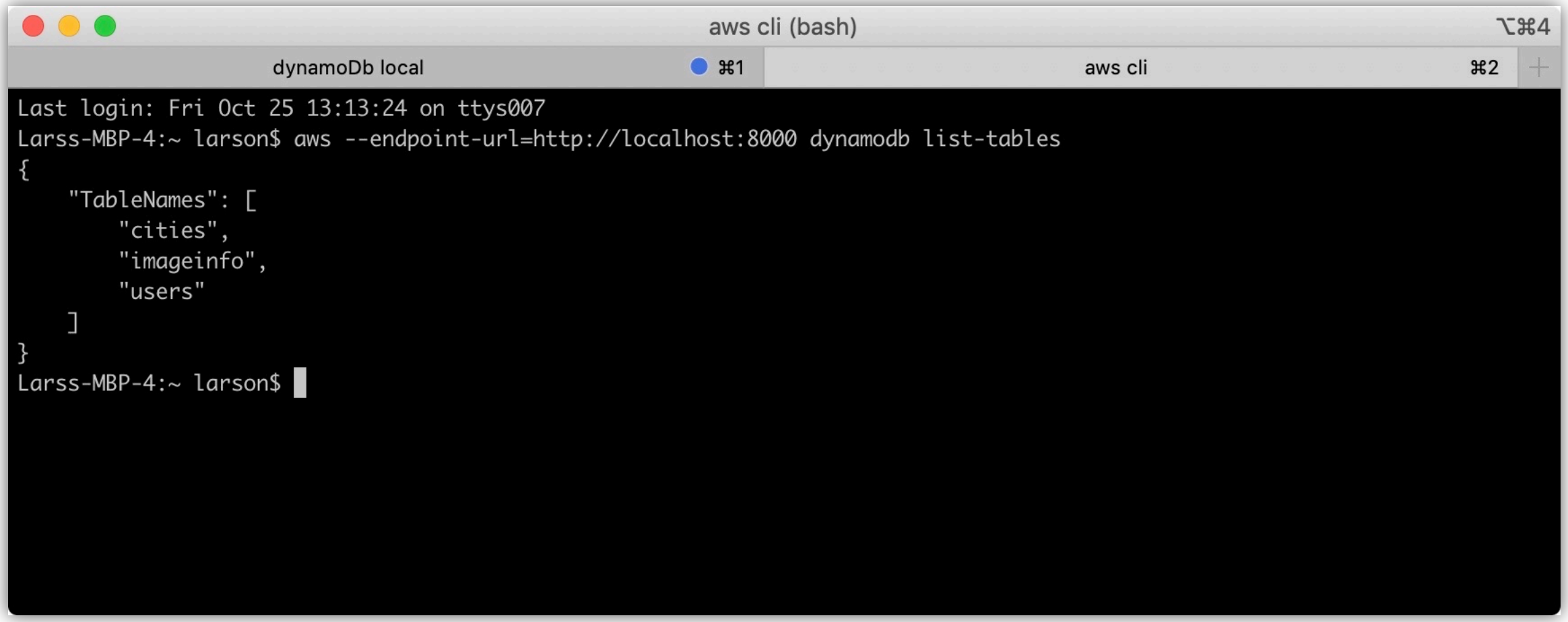
$ java –jar DynamoDBLocal.jar

Simulate Component triggered by Lambda



$ aws –endpoint-url=http://localhost:8000 dynamodb list-tables

Simulate Component triggered by Lambda



$ aws –endpoint-url=http://localhost:8000 dynamodb list-tables

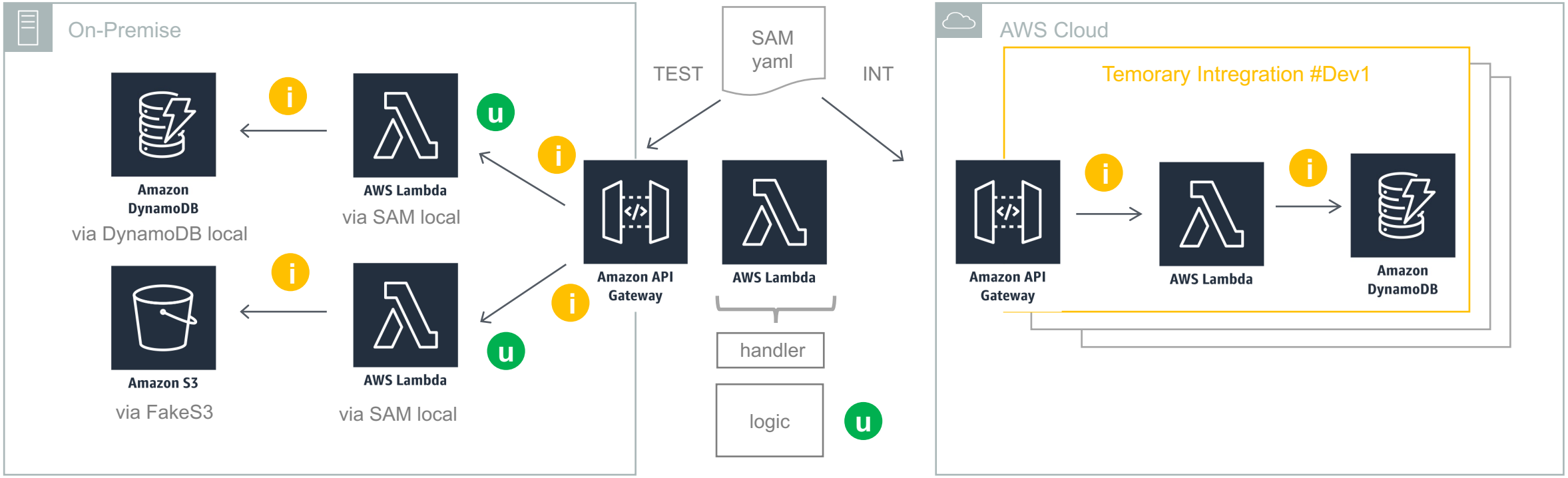
Testing Best Practices

u Kandidat **für Unit Tests**

i Kandidat **für Integration Tests**

e Kandidat **für End-to-Ende Tests**

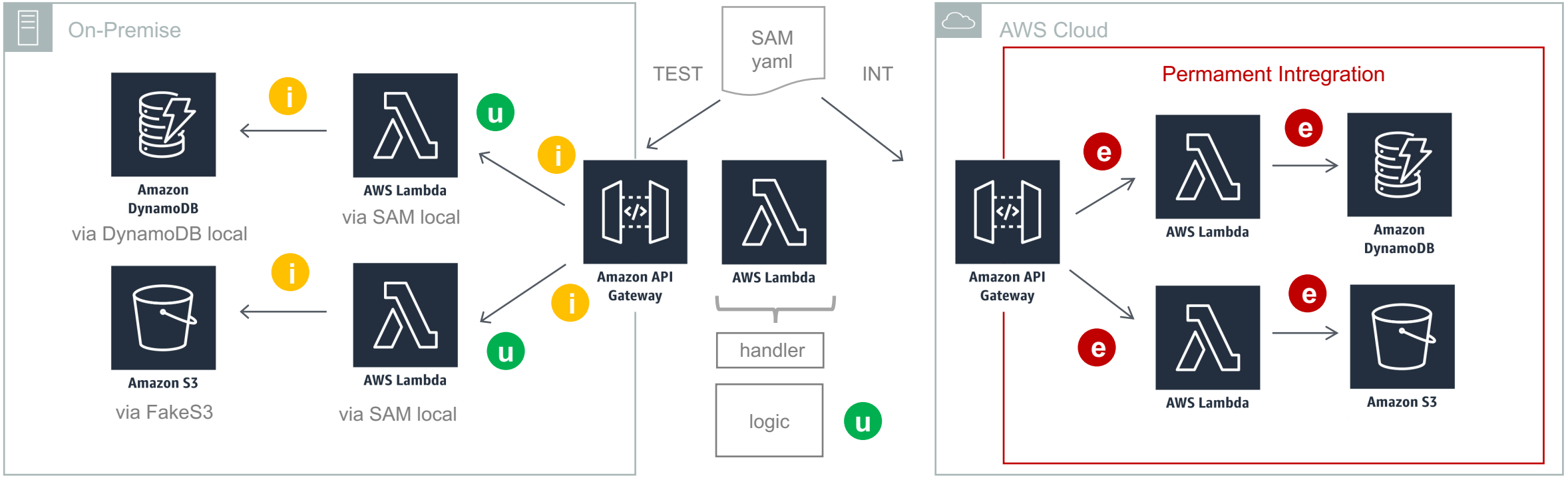
#6 temporäre Integration-Cloud für partielle Integration Tests



Testing Best Practices

- u Kandidat **für Unit Tests**
- i Kandidat **für Integration Tests**
- e Kandidat **für End-to-Ende Tests**

#7 permanente Integration-Cloud für End-to-End Tests



Lessons learned:

Serverless U/I/A-Testing

SERVERLESS U/I/A-TESTING

Lessons Learned

- **Separation** of Concerns
- **SAM** zum lokalen Starten von Lambdas
- **Locale Cloud** zur lokalen Simulation
- **CloudFormation** zum Aufsetzen von Cloud ENVs

- **Nur die Cloud** verhält sich wie die Cloud!

Agenda

- *Hello World & Set-up* ✓
- *Order Management* ✓
- *Order Processing* ✓
- *API Gateway* ✓
- *Testing* ✓

OPEN KNOWLEDGE

*Was ihr aus dem heutigen Workshop **mitnehmen** solltet:*

- **eine Lambda** kommt nie allein
- je **kleiner** desto besser
- je **kürzer** desto besser
- **Coldstart** als Herausforderung
- **Testen** auf allen Ebenen!
- **IaC** ist dein (Cloud-)Freund

OPEN KNOWLEDGE

*„**Going Serverless**“*
ist keine Frage der
Technologie, sondern eine
Architekturentscheidung!

OPEN KNOW LEDGE

Zeit für Fragen? Immer!

#WISSENTEILEN

by **open** knowledge **GmbH**

Vielen Dank!

Lars Röwekamp, CIO New Technologies

@_**open**Knowledge | @**mobile**Larson

BILDNACHWEIS

Folie 01: © Biletsky Egeniy, iStockphoto.com

Folie 05: © Ljubco, iStockphoto.com

Folie 06: © Orba Aljia, iStockphoto.com

Folie 44: © foxaon1987, Shuttertock.com

Folie 90: © Framework Wonderland, Shutterstock.com

All other pictures, drawings and icons originate from

- **pexels.com**, **pixabay.com**, **unsplash.com**,
- **flaticon.com**

or were made by my own.

OPEN KNOW LEDGE