Serverless Architecture

Software Architecture

Richard Thomas

April 20, 2026
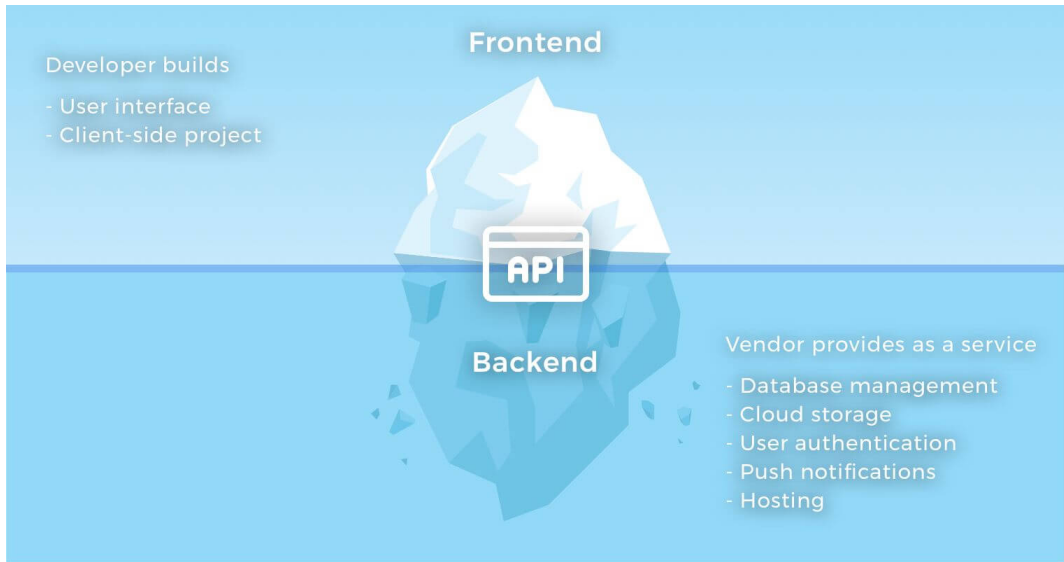
Oxymoron 1. Serverless

Logic running on someone else's server.

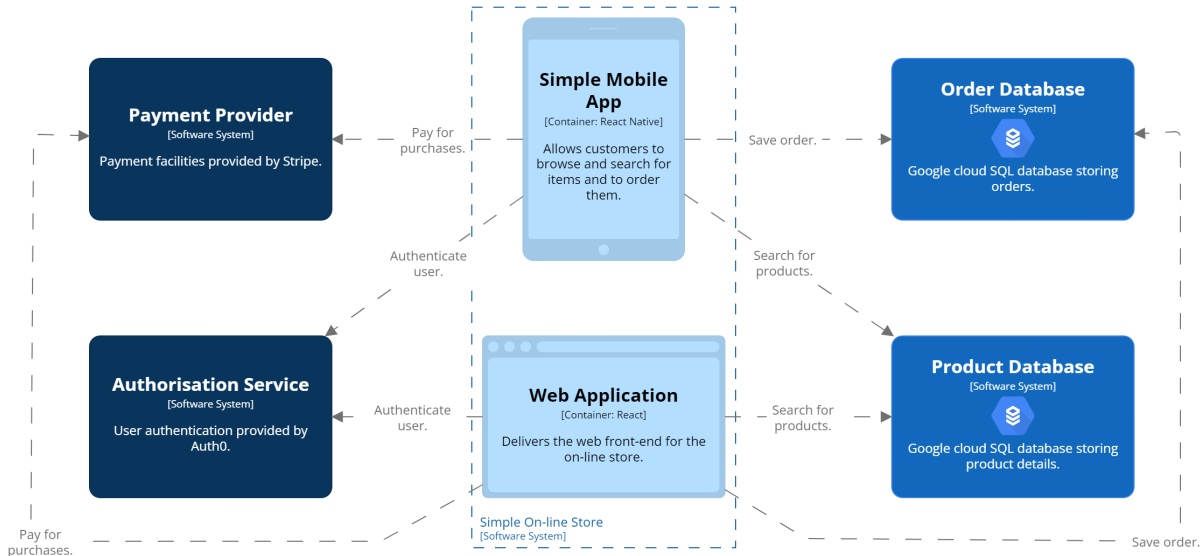> **Definition 0.** Backend as a Service (BaaS)
>
> Cloud-hosted applications or services that deliver functionality used by an application front-end.

BaaS Iceberg [Brunko, 2019]

Frontend

Developer builds

- User interface
- Client-side project

API

Backend

Vendor provides as a service

- Database management
- Cloud storage
- User authentication
- Push notifications
- Hosting

BaaS Example

Payment Provider
[Software System]

Payment facilities provided by Stripe.

Simple Mobile App
[Container: React Native]

Allows customers to browse and search for items and to order them.

Order Database
[Software System]

Google cloud SQL database storing orders.

Authorisation Service
[Software System]

User authentication provided by Auth0.

Web Application
[Container: React]

Delivers the web front-end for the on-line store.

Product Database
[Software System]

Google cloud SQL database storing product details.

Pay for purchases.

Authenticate user.

Save order.

Search for products.

Authenticate user.

Search for products.

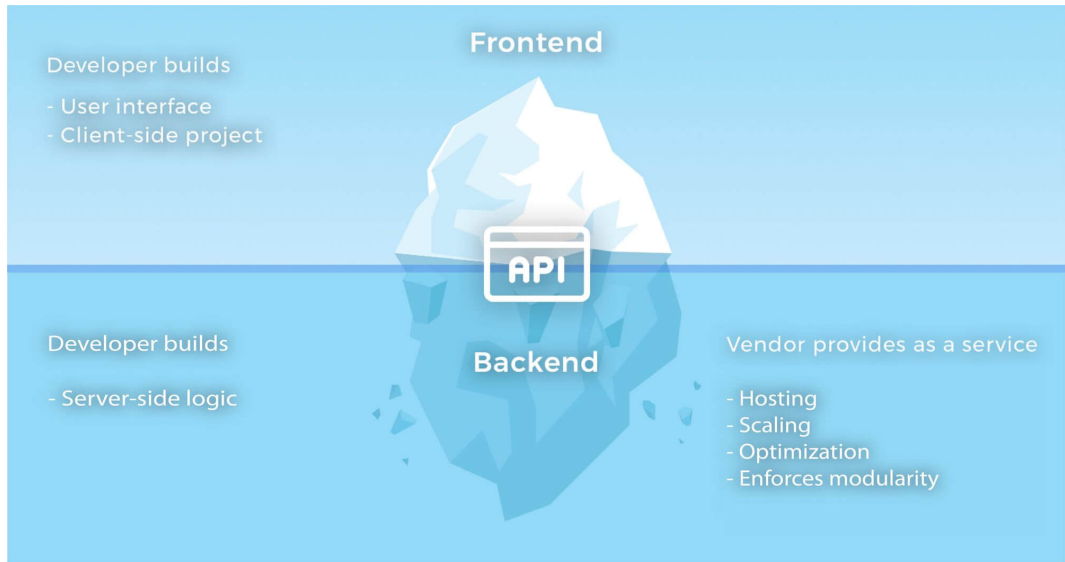Pay for purchases.

Save order.

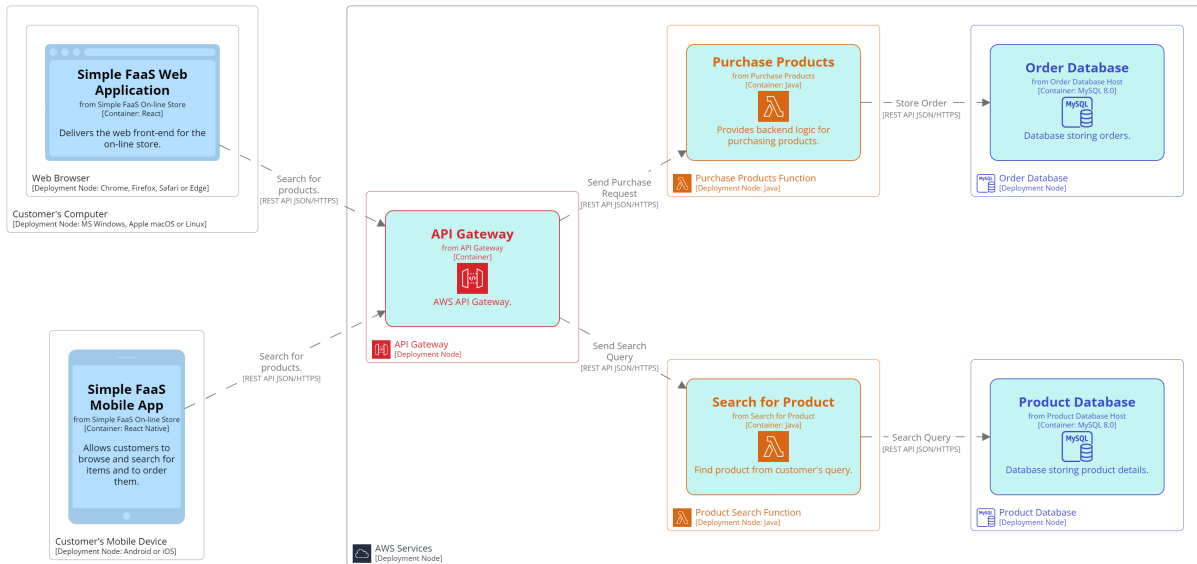Simple On-line Store
[Software System]

Definition 0. Functions as a Service (FaaS)

Application logic that is triggered by an event and runs in a *transient*, *stateless* compute node.

FaaS Iceberg [Brunko, 2019]

FaaS Example

Simple FaaS Web Application
from Simple FaaS On-line Store
[Container: React]

Delivers the web front-end for the on-line store.

Web Browser
[Deployment Node: Chrome, Firefox, Safari or Edge]

Customer's Computer
[Deployment Node: MS Windows, Apple macOS or Linux]

Simple FaaS Mobile App
from Simple FaaS On-line Store
[Container: React Native]

Allows customers to browse and search for items and to order them.

Customer's Mobile Device
[Deployment Node: Android or iOS]

API Gateway
from API Gateway
[Container]

AWS API Gateway.

API Gateway
[Deployment Node]

Search for products.
[REST API JSON/HTTPS]

Search for products.
[REST API JSON/HTTPS]

Purchase Products
from Purchase Products
[Container: Java]

Provides backend logic for purchasing products.

Purchase Products Function
[Deployment Node: Java]

Send Purchase Request
[REST API JSON/HTTPS]

Store Order
[REST API JSON/HTTPS]

Order Database
from Order Database Host
[Container: MySQL 8.0]

Database storing orders.

Order Database
[Deployment Node]

Send Search Query
[REST API JSON/HTTPS]

Search for Product
from Search for Product
[Container: Java]

Find product from customer's query.

Product Search Function
[Deployment Node: Java]

Search Query
[REST API JSON/HTTPS]

Product Database
from Product Database Host
[Container: MySQL 8.0]

Database storing product details.

Product Database
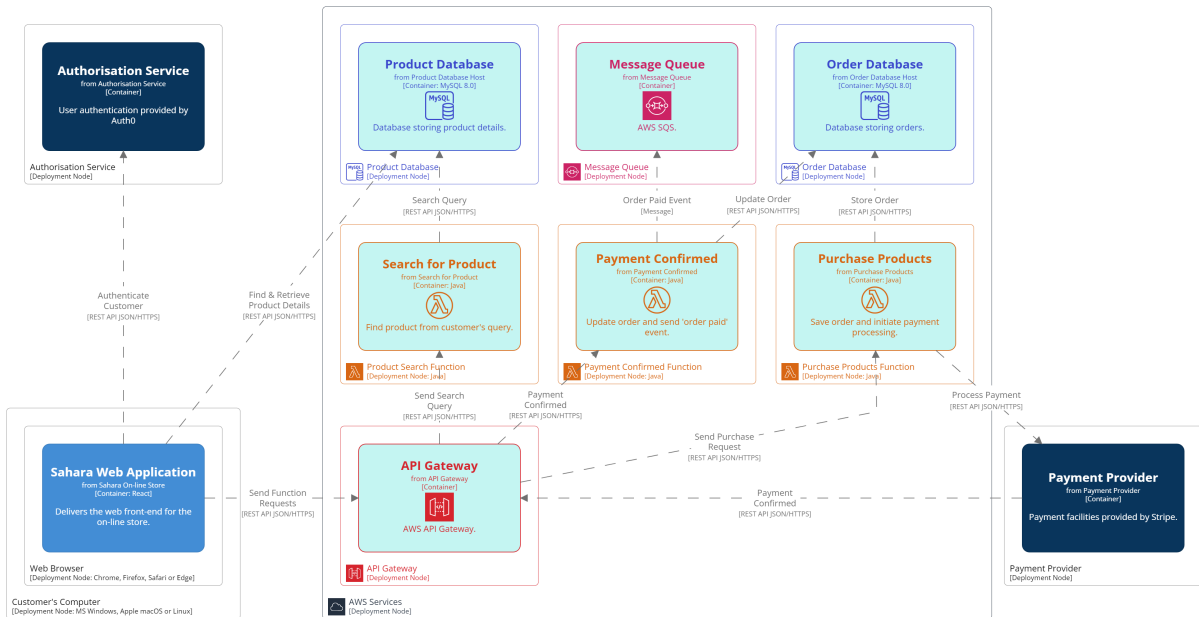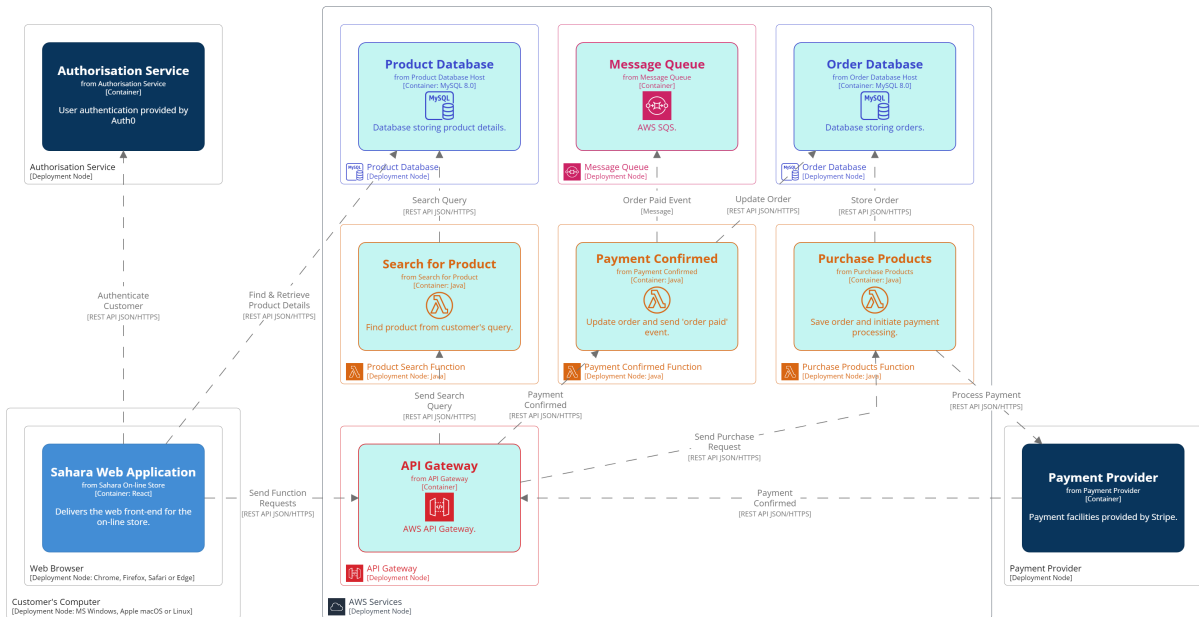[Deployment Node]

AWS Services
[Deployment Node]

> *Definition 0.* Serverless Architecture
>
> Software system delivering functionality through BaaS or FaaS.
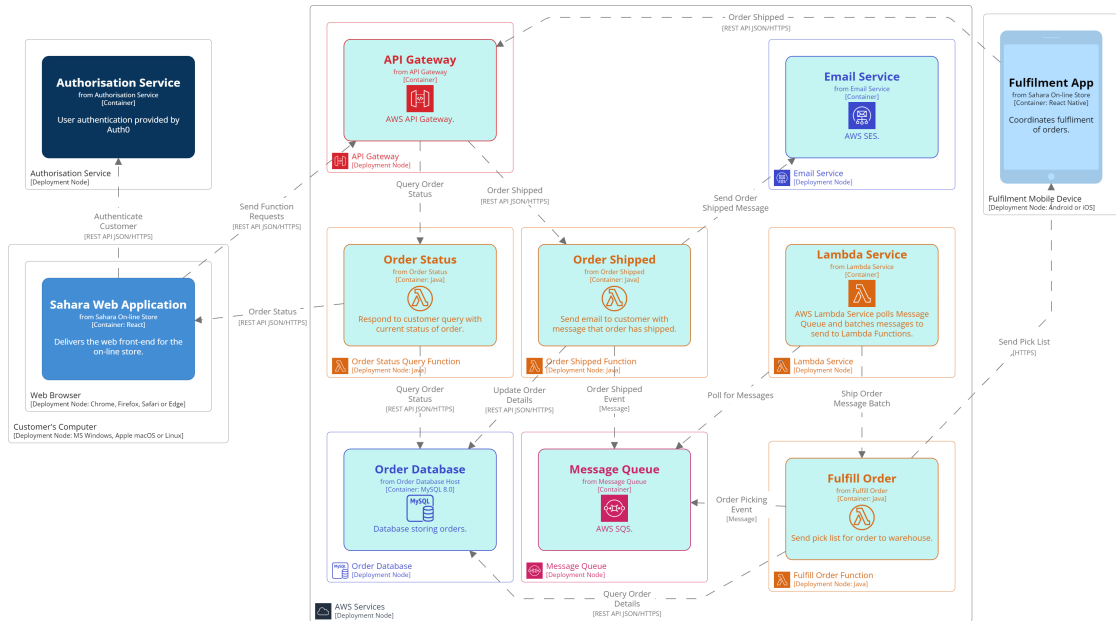
Sahara Browse & Order — Serverless

Authorisation Service
from Authorisation Service
[Container]

User authentication provided by Auth0

Authorisation Service
[Deployment Node]

Product Database
from Product Database Host
[Container: MySQL 8.0]

MySQL

Database storing product details.

Product Database
[Deployment Node]

Message Queue
from Message Queue
[Container]

AWS SQS.

Message Queue
[Deployment Node]

Order Database
from Order Database Host
[Container: MySQL 8.0]

MySQL

Database storing orders.

Order Database
[Deployment Node]

Search Query
[REST API JSON/HTTPS]

Order Paid Event
[Message]

Update Order
[REST API JSON/HTTPS]

Store Order
[REST API JSON/HTTPS]

Authenticate
Customer
[REST API JSON/HTTPS]

Find & Retrieve
Product Details
[REST API JSON/HTTPS]

Search for Product
from Search for Product
[Container: Java]

Find product from customer's query.

Product Search Function
[Deployment Node: Java]

Payment Confirmed
from Payment Confirmed
[Container: Java]

Update order and send 'order paid' event.

Payment Confirmed Function
[Deployment Node: Java]

Purchase Products
from Purchase Products
[Container: Java]

Save order and initiate payment processing.

Purchase Products Function
[Deployment Node: Java]

Process Payment
[REST API JSON/HTTPS]

Send Search
Query
[REST API JSON/HTTPS]

Payment
Confirmed
[REST API JSON/HTTPS]

Sahara Web Application
from Sahara On-line Store
[Container: React]

Delivers the web front-end for the on-line store.

Web Browser
[Deployment Node: Chrome, Firefox, Safari or Edge]

Send Function
Requests
[REST API JSON/HTTPS]

API Gateway
from API Gateway
[Container]

AWS API Gateway.

API Gateway
[Deployment Node]

Send Purchase
Request
[REST API JSON/HTTPS]

Payment
Confirmed
[REST API JSON/HTTPS]

Payment Provider
from Payment Provider
[Container]

Payment facilities provided by Stripe.

Payment Provider
[Deployment Node]

Customer's Computer
[Deployment Node: MS Windows, Apple macOS or Linux]

AWS Services
[Deployment Node]

Sahara Browse & Order — Serverless

Authorisation Service
from Authorisation Service
[Container]

User authentication provided by Auth0

Authorisation Service
[Deployment Node]

Product Database
from Product Database Host
[Container: MySQL 8.0]

MySQL

Database storing product details.

Product Database
[Deployment Node]

Message Queue
from Message Queue
[Container]

AWS SQS.

Message Queue
[Deployment Node]

Order Database
from Order Database Host
[Container: MySQL 8.0]

MySQL

Database storing orders.

Order Database
[Deployment Node]

Search for Product
from Search for Product
[Container: Java]

Find product from customer's query.

Product Search Function
[Deployment Node: Java]

Payment Confirmed
from Payment Confirmed
[Container: Java]

Update order and send 'order paid' event.

Payment Confirmed Function
[Deployment Node: Java]

Purchase Products
from Purchase Products
[Container: Java]

Save order and initiate payment processing.

Purchase Products Function
[Deployment Node: Java]

Sahara Web Application
from Sahara On-line Store
[Container: React]

Delivers the web front-end for the on-line store.

Web Browser
[Deployment Node: Chrome, Firefox, Safari or Edge]

Customer's Computer
[Deployment Node: MS Windows, Apple macOS or Linux]

API Gateway
from API Gateway
[Container]

AWS API Gateway.

API Gateway
[Deployment Node]

AWS Services
[Deployment Node]

Payment Provider
from Payment Provider
[Container]

Payment facilities provided by Stripe.

Payment Provider
[Deployment Node]

Authenticate
Customer
[REST API JSON/HTTPS]

Find & Retrieve
Product Details
[REST API JSON/HTTPS]

Search Query
[REST API JSON/HTTPS]

Order Paid Event
[Message]

Update Order
[REST API JSON/HTTPS]

Store Order
[REST API JSON/HTTPS]

Send Search
Query
[REST API JSON/HTTPS]

Payment
Confirmed
[REST API JSON/HTTPS]

Process Payment
[REST API JSON/HTTPS]

Send Function
Requests
[REST API JSON/HTTPS]

Send Purchase
Request
[REST API JSON/HTTPS]

Payment
Confirmed
[REST API JSON/HTTPS]

Sahara Fulfilment — Serverless

Authorisation Service
from Authorisation Service
[Container]

User authentication provided by
Auth0

Authorisation Service
[Deployment Node]

Sahara Web Application
from Sahara On-line Store
[Container: React]

Delivers the web front-end for the
on-line store.

Web Browser
[Deployment Node: Chrome, Firefox, Safari or Edge]

Customer's Computer
[Deployment Node: MS Windows, Apple macOS or Linux]

API Gateway
from API Gateway
[Container]

AWS API Gateway.

API Gateway
[Deployment Node]

Email Service
from Email Service
[Container]

AWS SES.

Email Service
[Deployment Node]

Fulfilment App
from Sahara On-line Store
[Container: React Native]

Coordinates fulfilment
of orders.

Fulfilment Mobile Device
[Deployment Node: Android or iOS]

Order Status
from Order Status
[Container: Java]

Respond to customer query with
current status of order.

Order Status Query Function
[Deployment Node: Java]

Order Shipped
from Order Shipped
[Container: Java]

Send email to customer with
message that order has shipped.

Order Shipped Function
[Deployment Node: Java]

Lambda Service
from Lambda Service
[Container: Java]

AWS Lambda Service polls Message
Queue and batches messages to
send to Lambda Functions.

Lambda Service
[Deployment Node]

Order Database
from Order Database Host
[Container: MySQL 8.0]

Database storing orders.

Order Database
[Deployment Node]

Message Queue
from Message Queue
[Container]

AWS SQS.

Message Queue
[Deployment Node]

Fulfill Order
from Fulfill Order
[Container: Java]

Send pick list for order to warehouse.

Fulfill Order Function
[Deployment Node: Java]

AWS Services
[Deployment Node]

Authenticate
Customer
[REST API JSON/HTTPS]

Send Function
Requests
[REST API JSON/HTTPS]

Order Status
[REST API JSON/HTTPS]

Order Shipped
[REST API JSON/HTTPS]

Query Order
Status

Order Shipped
[REST API JSON/HTTPS]

Send Order
Shipped Message

Send Pick List
[HTTPS]

Query Order
Status
[REST API JSON/HTTPS]

Update Order
Details
[REST API JSON/HTTPS]

Order Shipped
Event
[Message]

Poll for Messages

Ship Order
Message Batch

Order Picking
Event
[Message]

Query Order
Details
[REST API JSON/HTTPS]

Sahara Fulfilment — Serverless

Authorisation Service
from Authorisation Service
[Container]

User authentication provided by Auth0

Authorisation Service
[Deployment Node]

Sahara Web Application
from Sahara On-line Store
[Container: React]

Delivers the web front-end for the on-line store.

Web Browser
[Deployment Node: Chrome, Firefox, Safari or Edge]

Customer's Computer
[Deployment Node: MS Windows, Apple macOS or Linux]

API Gateway
from API Gateway
[Container]

AWS API Gateway.

API Gateway
[Deployment Node]

Order Status
from Order Status
[Container: Java]

Respond to customer query with current status of order.

Order Status Query Function
[Deployment Node: Java]

Order Shipped
from Order Shipped
[Container: Java]

Send email to customer with message that order has shipped.

Order Shipped Function
[Deployment Node: Java]

Order Database
from Order Database Host
[Container: MySQL 8.0]

Database storing orders.

Order Database
[Deployment Node]

Message Queue
from Message Queue
[Container]

AWS SQS.

Message Queue
[Deployment Node]

Email Service
from Email Service
[Container]

AWS SES.

Email Service
[Deployment Node]

Lambda Service
from Lambda Service
[Container]

AWS Lambda Service polls Message Queue and batches messages to send to Lambda Functions.

Lambda Service
[Deployment Node]

Fulfilment App
from Sahara On-line Store
[Container: React Native]

Coordinates fulfilment of orders.

Fulfilment Mobile Device
[Deployment Node: Android or iOS]

Fulfill Order
from Fulfill Order
[Container: Java]

Send pick list for order to warehouse.

Fulfill Order Function
[Deployment Node: Java]

AWS Services
[Deployment Node]

Authenticate Customer
[REST API JSON/HTTPS]

Send Function Requests
[REST API JSON/HTTPS]

Order Status
[REST API JSON/HTTPS]

Query Order Status
[REST API JSON/HTTPS]

Order Shipped
[REST API JSON/HTTPS]

Send Order Shipped Message

- Order Shipped -
[REST API JSON/HTTPS]

Send Pick List
[HTTPS]

Query Order Status
[REST API JSON/HTTPS]

Update Order Details
[REST API JSON/HTTPS]

Order Shipped Event
[Message]

Poll for Messages

Ship Order Message Batch

Order Picking Event
[Message]

Query Order Details
[REST API JSON/HTTPS]

- Automatic scaling
  - Multiple instances of function

- Automatic scaling
  - Multiple instances of function
- Reduced cost for dynamic loads
  - No server idle time

- Automatic scaling
  - Multiple instances of function
- Reduced cost for dynamic loads
  - No server idle time
- Reduced server management

Serverless Benefits

- Automatic scaling
  - Multiple instances of function
- Reduced cost for dynamic loads
  - No server idle time
- Reduced server management
- Easier to run closer to client
  - Launch in same zone as client

- Front-end accesses database directly
  - Front-end needs to sanitise inputs
  - Easy to spoof messages from front-end
    - Hope DB provider is secure

- Front-end accesses database directly
  - Front-end needs to sanitise inputs
  - Easy to spoof messages from front-end
    - Hope DB provider is secure
- Application logic is in front-end
  - Less modularisation
  - Duplication of logic with multiple front-ends
    - Web, mobile, . . .

- Front-end accesses database directly
  - Front-end needs to sanitise inputs
  - Easy to spoof messages from front-end
    - Hope DB provider is secure
- Application logic is in front-end
  - Less modularisation
  - Duplication of logic with multiple front-ends
    - Web, mobile, …
- No control over server optimisation

- No server state
  - All state needs to be saved (e.g. Redis, S3, . . .)
    - Not just persistent state

- No server state
  - All state needs to be saved (e.g. Redis, S3, …)
    - Not just persistent state
- Execution duration
  - Can't be long running process
    - AWS Lambda – up to 15 minutes

- No server state
  - All state needs to be saved (e.g. Redis, S3, …)
    - Not just persistent state
- Execution duration
  - Can't be long running process
    - AWS Lambda – up to 15 minutes
- Startup latency
  - Functions take time to start
    - Some languages worse than others (e.g. Java)

- No server state
  - All state needs to be saved (e.g. Redis, S3, . . .)
    - Not just persistent state
- Execution duration
  - Can't be long running process
    - AWS Lambda – up to 15 minutes
- Startup latency
  - Functions take time to start
    - Some languages worse than others (e.g. Java)
- Proliferation of functions
  - Loss of encapsulation

When is serverless appropriate?

When is serverless appropriate?

- Rich client apps with common backend
  - BaaS

When is serverless appropriate?

- Rich client apps with common backend
  - BaaS
- High latency processing
  - Within function duration constraints

When is serverless appropriate?

- Rich client apps with common backend
  - BaaS

- High latency processing
  - Within function duration constraints

- Apps with variable load
  - Take advantage of auto-scaling

When is serverless *not* appropriate?

When is serverless *not* appropriate?

- Quick response required
  - Can't wait for FaaS to start

When is serverless *not* appropriate?

- Quick response required
  - Can't wait for FaaS to start
- Compute intensive processing

When is serverless *not* appropriate?

- Quick response required
  - Can't wait for FaaS to start

- Compute intensive processing

- Apps with steady load
  - Server-based approaches are cheaper

- Redesign your scalability assignment to be serverless.
  - What parts of your design would benefit from being serverless?

- Implement your revised design.

Pros & Cons

Extensibility 👍

Reliability 👍

Interoperability 👍

Scalability 👍

Deployability 👍

Modularity 😐

Testability 😐

Maintainability 😐

Security 🙁

Simplicity 🙁

References

[Brunko, 2019] Brunko, P. (2019).
Serverless architecture: When to use this approach and what benefits it gives.
`https://apiko.com/blog/serverless-architecture-benefits/`.