

Serverless Architecture

Software Architecture

Richard Thomas

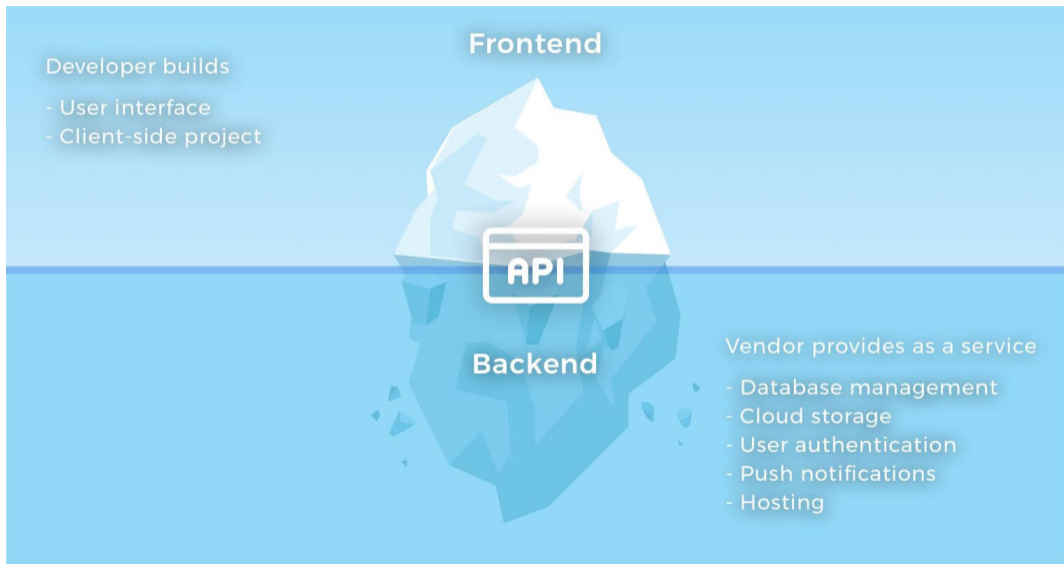
May 6, 2024

Oxymoron 1. Serverless

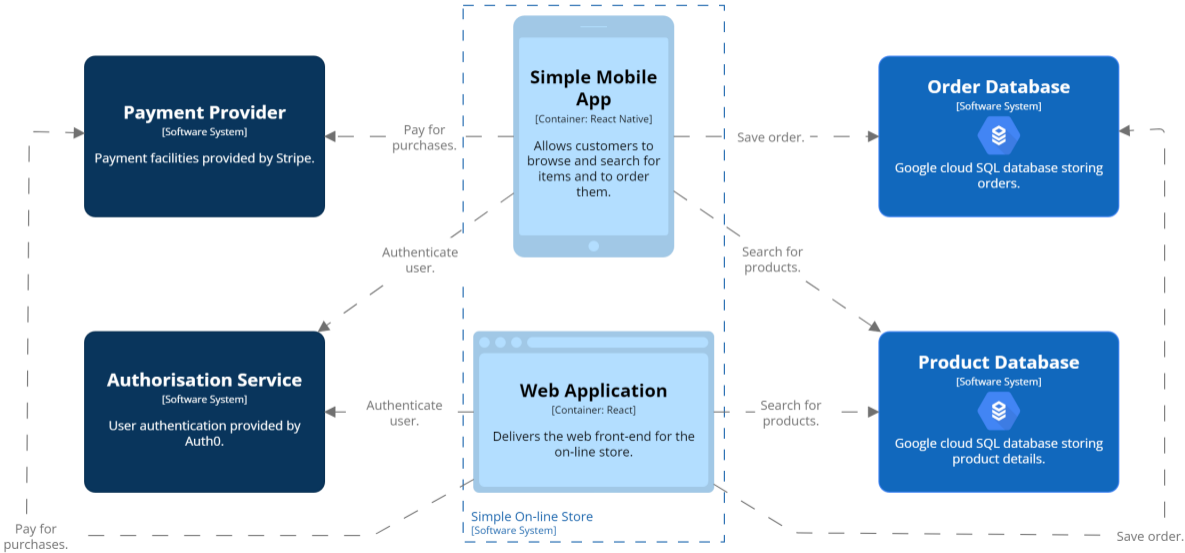
Logic running on someone else's server.

Definition 1. Backend as a Service (BaaS)

Cloud-hosted applications or services that deliver functionality used by an application front-end.



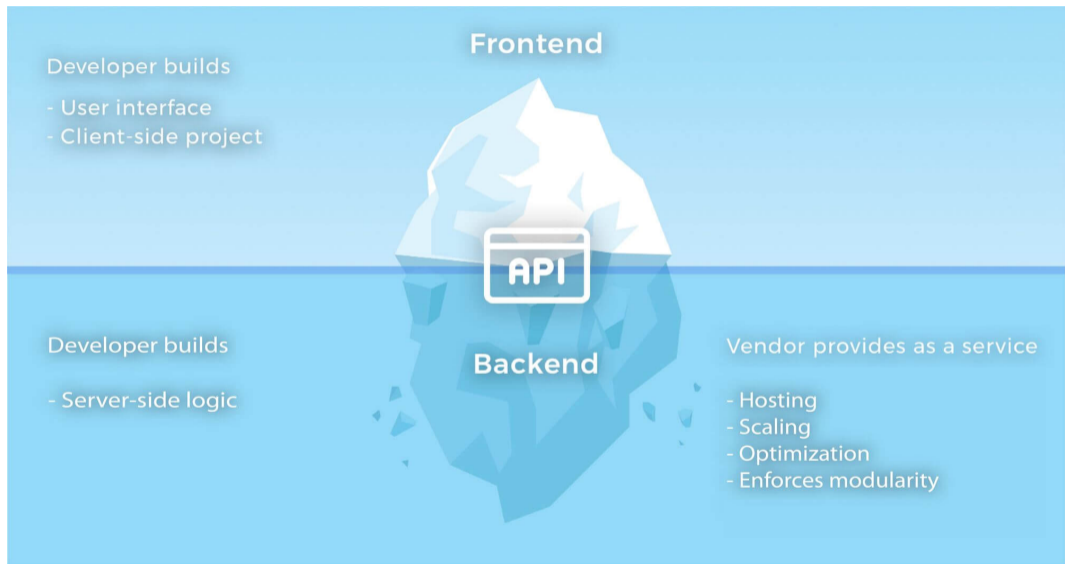
BaaS Example



Definition 2. 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



Definition 3. 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]

 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]

 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]

API Gateway
 from API Gateway
 [Container]

 AWS API Gateway.

API Gateway
 [Deployment Node]

Payment Provider
 from Payment Provider
 [Container]
 Payment facilities provided by Stripe.

Payment Provider
 [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]

AWS Services
 [Deployment Node]

Authenticate Customer
 (REST API | JSON/HTTPS)

Find & Retrieve Product Details
 (REST API | JSON/HTTPS)

Send Function Requests
 (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)

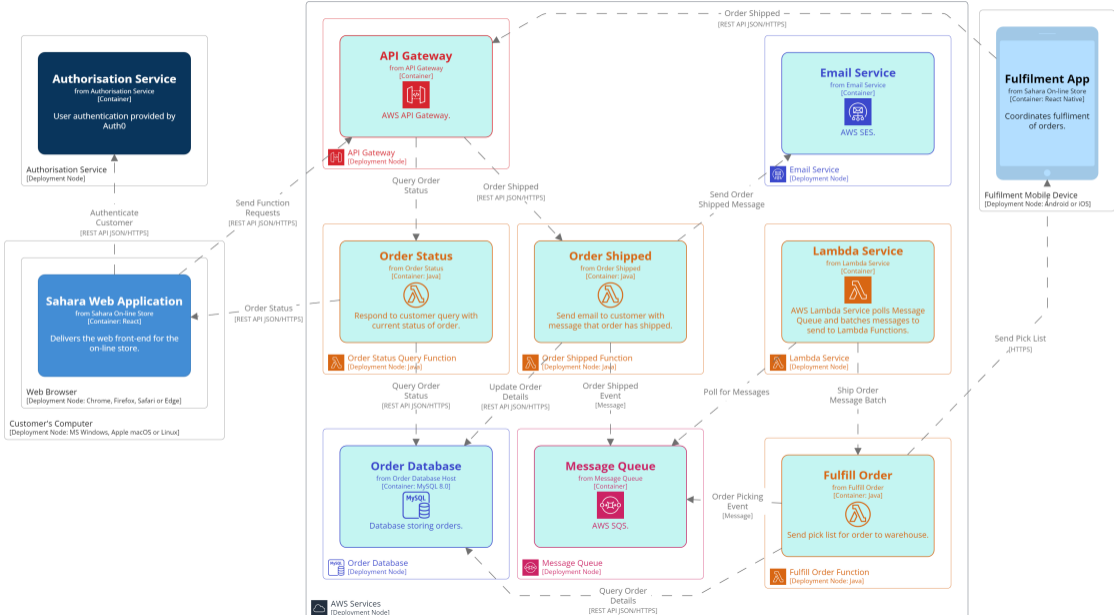
Payment Confirmed
 (REST API | JSON/HTTPS)

Send Purchase Request
 (REST API | JSON/HTTPS)

Process Payment
 (REST API | JSON/HTTPS)

Payment Confirmed
 (REST API | JSON/HTTPS)

Sahara Fulfilment — Serverless



Serverless Benefits

- Automatic scaling
 - Multiple instances of function

Serverless Benefits

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

Serverless Benefits

- 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

BaaS Tradeoffs

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

BaaS Tradeoffs

- 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, ...

BaaS Tradeoffs

- 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

FaaS Tradeoffs

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

FaaS Tradeoffs

- 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

FaaS Tradeoffs

- 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)

FaaS Tradeoffs

- 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

Question

When is serverless appropriate?

Question

When is serverless appropriate?

Answer

- Rich client apps with common backend
 - BaaS

Question

When is serverless appropriate?

Answer

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

Question

When is serverless appropriate?

Answer

- Rich client apps with common backend
 - BaaS
- High latency processing
 - Within function duration constraints
- Apps with variable load
 - Take advantage of auto-scaling

Question

When is serverless *not* appropriate?

Question

When is serverless *not* appropriate?

Answer

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

Question

When is serverless *not* appropriate?

Answer

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

Question

When is serverless *not* appropriate?

Answer

- Quick response required
 - Can't wait for FaaS to start
- Compute intensive processing
- Apps with steady load
 - Server-based approaches are cheaper

Self-Study Exercise

- 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/>.