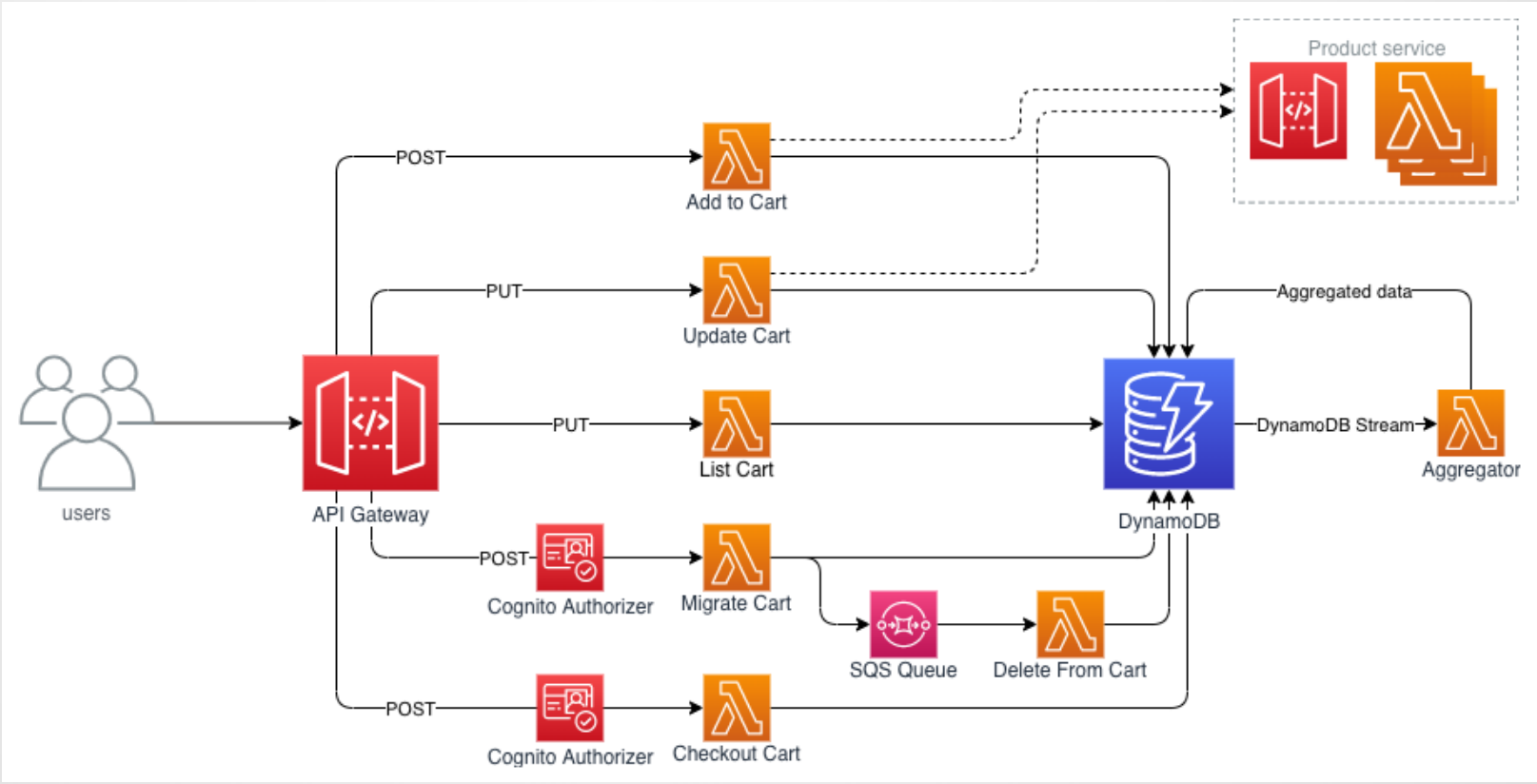
Serverless


Product service
POST
Add to Cart
PUT
Update Cart
Aggregated data
users
API Gateway
PUT
List Cart
DynamoDB
DynamoDB Stream
Aggregator
POST
Cognito Authorizer
Migrate Cart
SQS Queue
Delete From Cart
POST
Cognito Authorizer
Checkout Cart

Serverless

Ist ein Zusammenspiel zwischen Backend as a Service (**Baas**) und Function as a Service (**Faas**), das es erlaubt eine komplette Applikation zu erstellen, die nur auf der Hardware der jeweiligen Provider läuft.

BaaS

Backend as a Service

Baas

Über eine **im Browser** laufende Entwicklungsumgebung werden **Backend-Dienste** konfiguriert und zusammengeschaltet mit dem Ziel ein Backend bereitzustellen.

Als Möglichkeit der Anbindung an den Code des Entwicklers werden **SDKs** angeboten.

Baas - Beispiele

- ApiOmat
- Baasbox
- Baqend
- Built.io
- cloudbase
- Firebase
- Stackmob
- ...

- Benutzerverwaltungen
- Datenbanken
- Chats, Chatbots
- Push Notification Frameworks
- Social Media Adapters
- Payment Providers
- Loggers
- Analytics
- Add-Integrators
- ...

FaaS

Function as a Service

Faas

Bietet nicht gesamte Softwarelösungen, wie zum Beispiel Baas, sondern ein **Framework** für das Erstellen von **einzelnen Funktionen**, die zusammengehängt werden können.

Können auf **Events** reagieren.

Unterhalb werden Server, Netzwerk, Storage, Betriebssystem oder Laufzeitumgebungen als **gegeben betrachtet** (vom jeweiligen Provider transparent zur Verfügung gestellt).

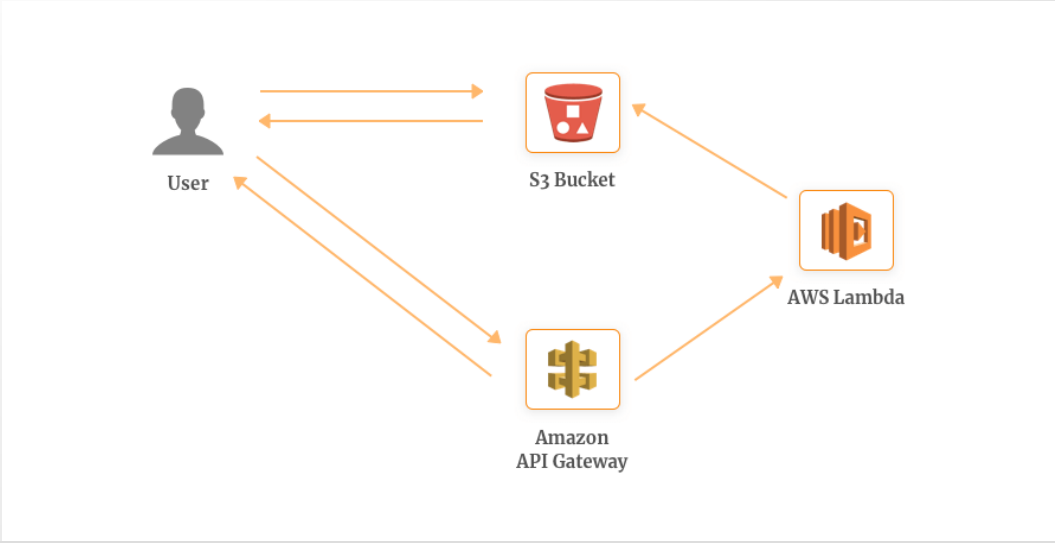
Faas - Beispiele

- AWS Lambda
- Google Cloud Functions
- Azure Functions
- Open Whisk
- Cloudflare Worker
- Oracle Cloud Fn
- ...

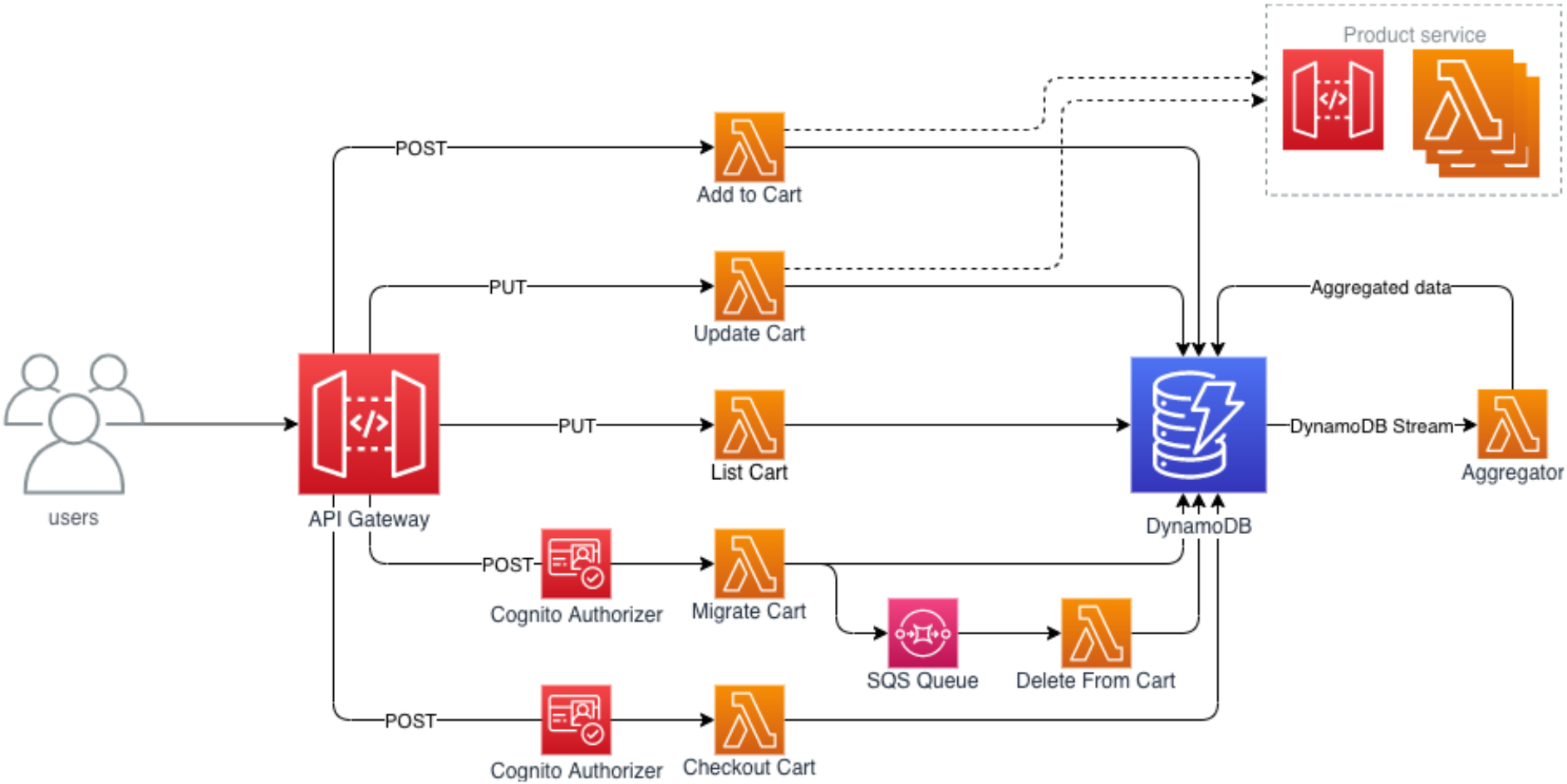
- Media-Conversion (mpg -> avi)
- Alexa Skills
- Image Recognition
- Any Business-Logic in General
  Reagiert auf Events von anderen Bausteinen und triggert andere Funktionen oder Bausteine (Storage, etc...)

Faas - Beispiel

- S3 liefert statische Seite aus
- Benutzer will Medien-Datei in bestimmten Format von S3
- Kriegt Redirect auf API-Gateway
- Lambda wird getriggert, holt Originalfile aus S3, konvertiert es und speichert es im neuen Format auf S3
- Benutzer bekommt wieder Redirect auf S3 und die Medien-Datei wird jetzt im richtigen Format gefunden.



Sample Shopping Cart Microservice



Vorteile

Vorteile

- No Managing Infrastructure
- No Initial Hardware Costs
- Decoupling Modules Trough Events
- Application Scaling
- Short Time-To-Market
- Infrastructure Security

Vorteile 1

- **No Managing Infrastructure**
  Nachdem die Funktionen in der Cloud laufen, brauchen wir uns nicht mehr um die Infrastruktur kümmern.

- **No Initial Hardware Costs**
  Die Kosten für die Basis-Server die ihr braucht, damit ihr überhaupt beginnen könnt, fallen weg.

Vorteile 2

- **Decoupling Modules Trough Events**
  Nachdem die Module der Applikation durch das Event-System getrennt sind, ist die Applikation als Gesamtes robuster gegenüber Ausfällen.

- **Application Scaling**
  Man kann die Applikation beliebig skalieren und auch automatisch skalieren lassen.

Vorteile 3

- **Short Time-To-Market**
  Time-To-Market ist wesentlich kürzer, da vorgefertigte Bausteine verwendet werden.

- **Infrastructure Security**
  Man muss sich nicht darum kümmern, dass die Infrastruktur abgesichert ist. Das sollte einem der Provider abnehmen.

Nachteile

Nachteile

- Price of Long-Running Workflows
- Cold-Start-Delay
- Provider-Dependency
- Higher Complexity
- Other Problems
- Less Flexibility

Nachteile 1

- **Price of Long-Running Workflows**
  Rechenintensive Projekte können auf Serverless-Architecture teurer sein, als auf eigenen Servern.

- **Cold-Start-Delay**
  Wenn das Projekt 'schlafen' geschickt wurde, weil länger kein Request passiert ist, kann es länger dauern, wenn es durch einen aktuellen Request wieder 'aufgeweckt' wird.
  (Azure ~5s max.)
  (AWS Lambda ~3s max.)

Nachteile 2

- **Provider-Dependency**
  Die Dienste sind in der Bedienung natürlich Provider-spezifisch. Dadurch entsteht ein Provider-Lock-In.

  Man ist im übrigen auch abhängig von den Debugging- und Monitoring-Tools des Providers.

  Sollte was passieren mit der Erreichbarkeit, dann kann man eigentlich nicht viel machen.

Nachteile 3

- **Higher Complexity**
  Die Anzahl der verfügbaren Funktionen ist enorm.
  Oft kann ein und das selbe Problem auf mehrere Arten gelöst werden (wie beim Programmieren so üblich), was die Übersichtlichkeit nicht erhöht.

- **Other Problems**
  Die Module in Serverless-Architecture sind extrem klein, was andere Probleme mit sich bringt, wie zum Beispiel Deployment, Versioning oder Integration Testing.

Nachteile 4

- **Less Flexibility**
  Nachdem die Provider ja sowohl die Art der angebotenen Komponenten, als auch deren Zusammenspiel direkt kontrollieren, kann das die Flexibilität des Entwicklers stark einschränken.

References

- https://www.simform.com/blog/serverless-architecture-guide/
- https://de.wikipedia.org/wiki/Backend_as_a_Service
- https://de.wikipedia.org/wiki/Function_as_a_Service
- https://www.simform.com/blog/serverless-aws-lambda-examples/
- https://aws.amazon.com/blogs/opensource/simplifying-serverless-best-practices-with-lambda-powertools/