# AWS SERVERLESS ARCHITECTURE BUILDING BLOCKS

OBJEKTspektrum



## SERVERLESS

Im Serverless-Modell werden Anwendungskomponenten, wie Datenbanken oder Komponenten zur Datenverarbeitung, automatisch und bedarfsabhängig vom Cloud-Provider zur Verfügung gestellt und betrieben. Die Verantwortung des Cloud-Nutzers liegt darin, diese Ressourcen zu konfigurieren – etwa durch eigenen Code oder anwendungsspezifische Parameter – und sie zu kombinieren.
Kosten fallen nach verbrauchten Kapazitäten an und die Skalierung erfolgt automatisch auf Basis der Last. Bereitstellung, Skalierung, Wartung, Hochverfügbarkeit und Verwaltung von Ressourcen liegen in der Verantwortung des Cloud-Providers.

**Serverless eignet sich besonders für schwer vorhersehbare oder kurzlebige Arbeitslasten, für Automatisierungsaufgaben oder Prototypen. Serverless ist weniger ideal für ressourcenintensive, langlebige und planbare Aufgaben, da in diesem Fall die Kosten signifikant höher sein können als bei selbstverwalteten Ausführungsumgebungen.**

## COMPUTE

### SERVERLESS FUNCTION
AWS Lambda

- Vom Cloud-Provider verwaltete **Ausführungsumgebung für Code**
- Automatische Provisionierung und Skalierung
- Ereignisgesteuerte Ausführung
- Kosten nur für Ausführungszeit

### SERVERLESS CONTAINERS
AWS Fargate

- Vom Cloud-Provider verwaltete **Ausführungsumgebung für Container**
- Container-Cluster ohne Management und Betrieb von virtuellen Maschinen
- Automatische Provisionierung und Skalierung
- Kosten nur für Laufzeit des Containers

## DATA

### RELATIONALE DATENBANK
Aurora Serverless

- Vollständig vom **Provider verwaltete,** mit MySQL/PostgreSQL kompatible **Datenbank**
- Automatische Skalierung und Redundanz
- Kosten für genutzten Speicherplatz, E/A, bereitgehaltene automatisch skalierte Kapazität

### NOSQL-DATENBANK
DynamoDB

- **Dokumenten-/Key-Value-Datenbank**
- Manuelle oder automatische Skalierung
- Automatische Ausfallsicherheit
- Kosten für Speicher und bereitgestellte Lese-/Schreibkapazität (manuell) oder E/A (automatisch skaliert)

## NETWORK

### REVERSE PROXY
Application Load Balancer

- **Routing** von HTTP-Anfragen auf verschiedene Backends (z.B Lambda)
- Vom Cloud-Provider verwaltete automatische Skalierung und Redundanz
- Kosten für Bereitstellung und verarbeitete Anfragen

### CONTENT DELIVERY NETWORK
Cloud Front

- **Caching** von statischen Ressourcen an global verteilten Edge-Standorten
- Eintrittspunkt ins AWS Backbone-Netz zur beschleunigten Übertragung von dynamischen Inhalten
- Kosten für Anfragen und übertragene Daten

### DOMAIN NAME SERVICE
Route 53

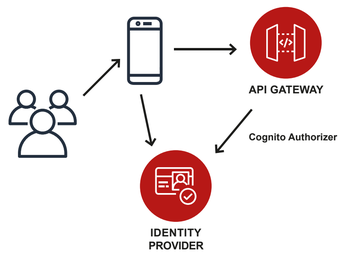
- **DNS-Dienst** mit fortgeschrittenen Funktionen (z.B.Routing-Strategien und DNS-Failover)
- Automatisiert konfigurierbar per API und Infrastructure-as-Code
- Kosten für verwaltete DNS-Zonen und Anfragen

## STORAGE

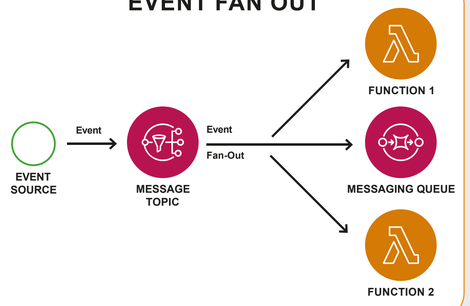
### OBJECT STORAGE
S3

- Unendlich skalierbarer **Objektspeicher**
- Upload und Download über HTTP-API
- Eignet sich für die Bereitstellung statischer Webseiten
- Kosten für genutzten Speicherplatz und abgehende Datenmenge

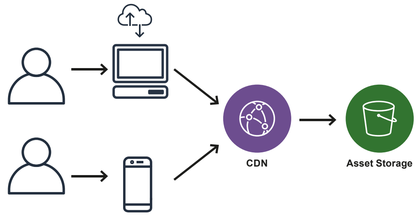
## COGNITO AUTHENTICATION



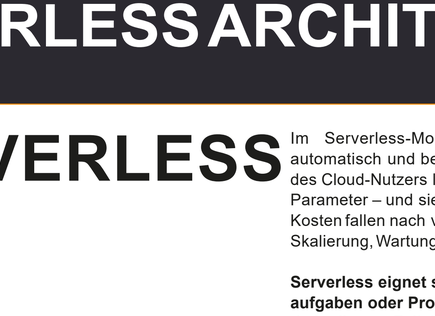

## EVENT FAN OUT




## STATIC WEBSITE HOSTING




## SIMPLE WEBSERVICE




## FILE PROCESSING


## DATA STREAM
Kinesis

- **Erfassung, Pufferung, Analyse** und **Verarbeitung** von **Streaming-Daten** in Echtzeit
- Integration mit Lambda, Redshift, Elasticsearch, Splunk, ...
- Kosten pro Stream und bereitgestellter Bandbreite

## ETL SERVICE
AWS Glue

- **Automatische Erstellung** eines **Datenkatalogs** anhand ausgewählter Datenquellen
- Automatische Generierung des ETL-Codes passend zu Quell- und Ziel-Daten
- Kosten für ETL-Jobs, Crawler-Laufzeit, Katalog-Objekte und Anfragen

## ANALYTICS

### STORAGE QUERY SERVICE
Athena

- Interaktiver **Abfrageservice** zur Analyse von **Daten** in Amazon S3
- Schema-Definition per Hive-DDL
- Abfragen über Standard-SQL
- Kosten für gelesene Datenmenge
- Optimierung über Kompression, Partitionierung und Spaltenformate möglich

## IDENTITY PROVIDER
Cognito

- OAuth-kompatibler **Identitätsanbieter** für die Benutzerverwaltung
- Registrierung, Passwort-Rücksetzung, Kontoauflösung, Multifaktor-Anmeldung und weitere Prozesse verfügbar
- Integration mit Social-Logins und OAuth-kompatiblen Providern
- Kosten für monatlich aktive Nutzer

## KEY MANAGEMENT
KMS

- Zentraler Dienst für die **sichere Ablage** und **Verwaltung von kryptografischen Schlüsseln**
- Ausführung von Ver- und Entschlüsselungsvorgängen mit Integration in andere AWS-Dienste
- Kosten nach Anzahl und Nutzung der Schlüssel

## SECURITY

### WEB APPLICATION FIREWALL
WAF

- Individuelle, regelbasierte **Firewall**
- Anwendungsbasierte Regeln möglich
- Integration in Content Delivery Network, Application Load Balancer und API Gateway
- API zur automatisierten Entwicklung, Bereitstellung und Wartung der Sicherheitsregeln
- Kosten für bereitgestellte Regeln und verarbeitete Anfragen

## PUB-/SUB-MESSAGING
Simple Notification Service

- **Mehrpunkt-Kommunikation** zwischen verschiedenen Systemen (z.B.Lambda)
- Eignet sich zur Entkopplung in einer ereignisgesteuerten Architektur
- Kosten für Nachrichten und übertragene Datenmenge

## MESSAGE QUEUES
Simple Queue Service

- **Nachrichtenzustellung über Warteschlangen** zur Kommunikation zwischen verschiedenen Systemen (z.B.Lambda)
- Exakt einmalige Zustellung und FIFO-Reihenfolge möglich
- Kosten für Anfragen und übertragene Datenmenge

## INTEGRATION

### GRAPHQL API
App Sync

- **Vollständig verwaltetes GraphQL-Gateway** und API-Management
- GraphQL-Anfragen können von verschiedenen Systemen (z.B.Lambda) aufgelöst werden
- Kosten für durchgeführte Anfragen

### WORKFLOW ENGINE
AWS Step Functions

- **Orchestrierungsdienst** zur Kombination von Serverless-Diensten (z.B Lambda)
- Konfiguration komplexer Workflows (inkl. Fehlerhandling, Bedingungen etc.)
- Kosten für durchgeführte Arbeitsschritte

### API MANAGEMENT
API Gateway

- Vollständig verwaltetes **API-Management für REST- und Websocket-APIs**
- Verteilung von API-Anfragen auf verschiedene Dienste (z.B.Lambda)
- Integration von Authentifizierung und API-Mapping möglich
- Kosten für verarbeitete Anfragen

## BEST PRACTICES FÜR LAMBDA-FUNKTIONEN

- Jede Lambda-Funktion sollte nur eine Sache tun (Single Responsibility Principle). Dies verbessert Wartbarkeit und Wiederverwendbarkeit. Speicherkapazität, Zugriffsrechte und Timeout-Einstellung können gezielter konfiguriert werden.
- Mit Erhöhung des zugewiesenen Speichers einer Lambda-Funktion werden auch CPU- und Netzwerkkapazität erhöht. Ein optimales Verhältnis aus Ausführungszeit und Kosten sollte per Benchmarking gefunden werden.
- Eine Lambda-Funktion sollte keine weitere Funktion synchron aufrufen. Das Warten führt zu unnötigen Kosten und erhöhter Kopplung. Stattdessen sollte asynchrone Verarbeitung, z.B. über Message Queues, eingesetzt werden.
- Das Deployment-Paket einer Lambda-Funktion sollte so klein wie möglich sein. Auf große externe Bibliotheken sollte verzichtet werden. Das verbessert die Kaltstartzeit. Wiederkehrende Initialisierungen von Abhängigkeiten sollten außerhalb der Handler-Funktion stattfinden, damit sie nur einmalig beim Kaltstart ausgeführt werden müssen. Betriebliche Parameter sollten über Umgebungsvariablen einer Lambda-Funktion abgebildet werden. Das verbessert die Wiederverwendbarkeit.
- Jede Lambda-Funktion sollte eine individuelle IAM-Rolle erhalten, welche die notwendigen Zugriffsrechte so restriktiv wie möglich definiert. Zustandsbehaftete Datenbankverbindungen sind zu vermeiden. Stattdessen sollten Service-APIs wie bei DynamoDB verwendet werden.

**Inhaltliche Entwicklung:**
Michael Muck
Christian Heinemann
Leo Lindhorst