Cloud Applications Architecture

Course 12 - Serverless

What is Serverless?

There still are servers, just entirely abstracted by the providers.

A service can be considered serverless **if and only if** (iff):

- you don't have to setup any infrastructure
- you don't have to provision/scale capacity
- you pay only for what you use
- the service can scale down to 0

Pros/Cons

- **you don't have to setup any infrastructure**
  - Less to manage ✔️
  - Getting started quicker ✔️
  - Less control ▯
  - Vendor lock-in ▯

- **you don't have to provision/scale capacity**
  - Can scale incredibly high ✔️
  - You might want to limit the scale (native ways to limit scale were introduced)
  - Might overload dependencies (e.g. databases) ▯

Pros/Cons

- **you pay only for what you use**
  - Scales with your business ✔▯
  - You have to pay to keep it running
    - even if it ends up costing more than expected ▯
- **the service can scale down to 0**
  - Great for multiple environments and prototypes ✔▯
    - Dev resources are not needed outside of the business hours
  - **Cold starts ▯**

Classification

Usually refers to functions as a service (FaaS), but there is more

- **Compute**
- **Workflows**
- **Databases**
- **Integration**
- **Storage**
- **Analytics**
- **Monitoring**
- **Development/build tools**

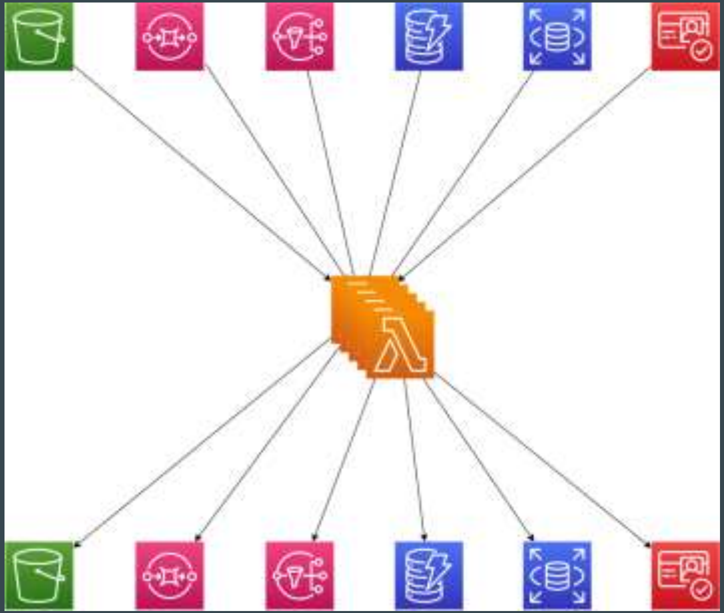
Serverless Compute

- FaaS (functions as a service)
  - E.g. AWS Lambda, Azure Functions, Google Functions, Cloudflare Workers
- Certain container services
  - E.g. Google Cloud Run, AWS Fargate
- Certain PaaS compute services
  - E.g. Google App Engine, Azure App Service

FaaS

Excellent for handling events (especially background events)

- Messages from integration services
- Authentication/user related actions
  - E.g. send custom email on sign-up, store user info in DB
- Changes in the database
  - Especially useful for NoSQL databases in which data consistency must be maintained programmatically due to its denormalized nature.
- File uploads



FaaS for HTTP

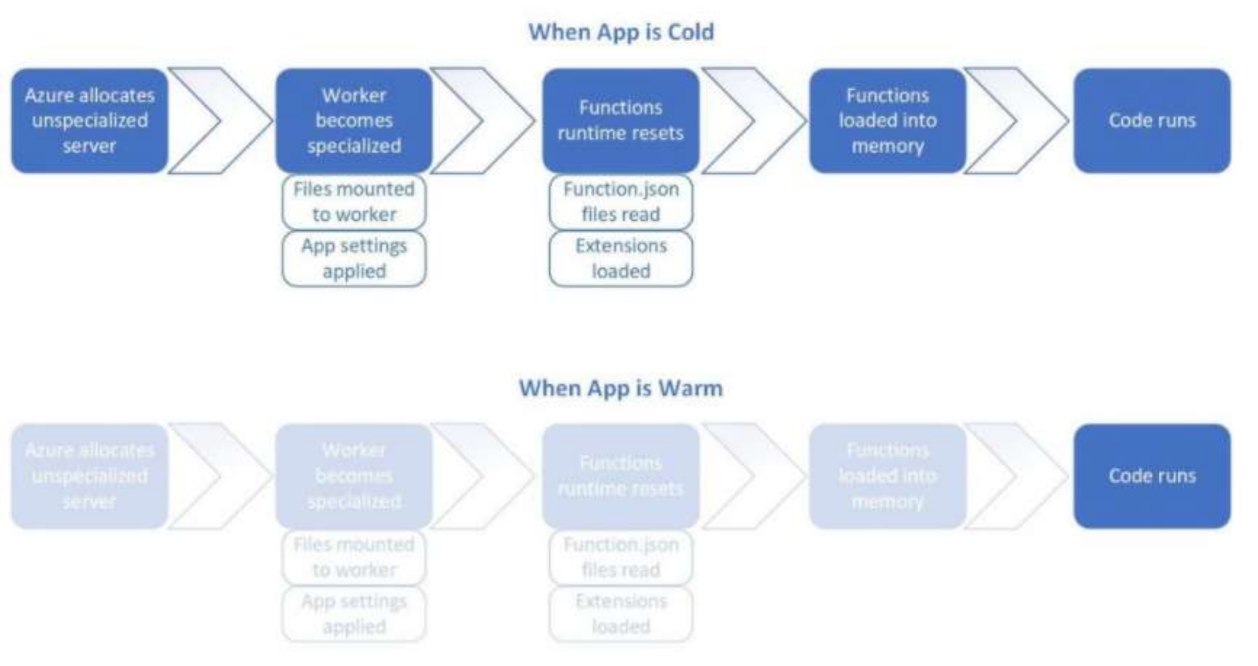
HTTP requests (e.g. through a REST API) are still events, but the user waits for a response.

Some FaaS requires additional integration services (e.g. Lambda requires API Gateway to be exposed as HTTP endpoints), while others provide URL endpoints automatically (Google and Azure).

- Both Google and Azure leveraged their PaaS services for FaaS (Google App Engine, Azure App Service). AWS built it from the ground up on Firecracker.

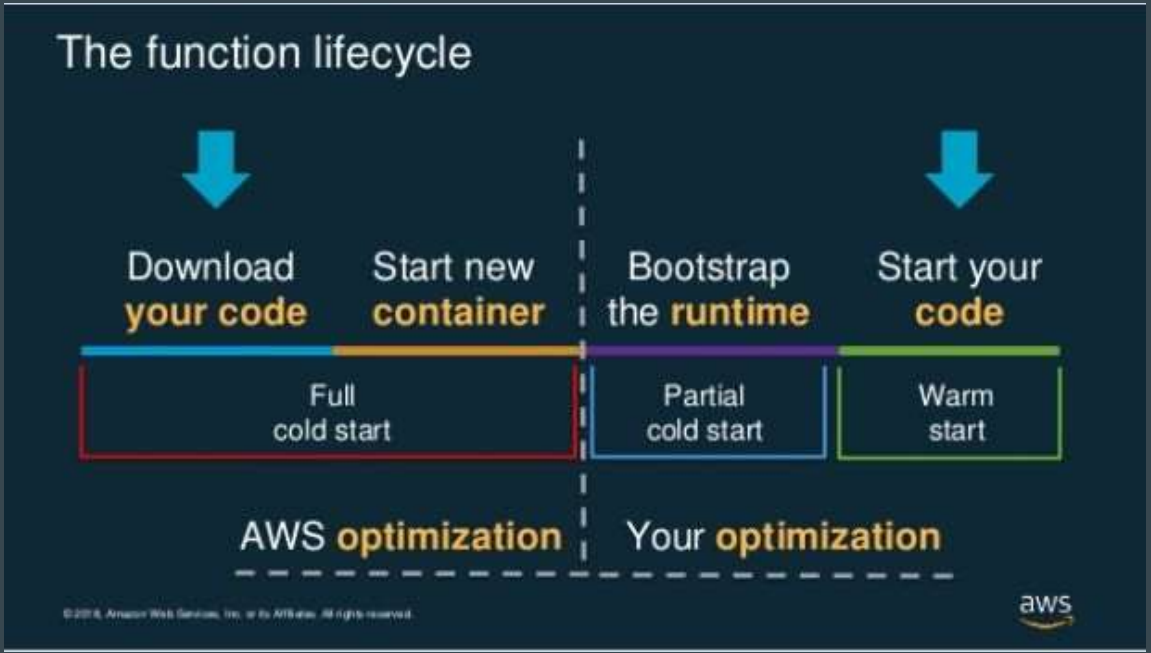
Cold-start is a common issue for HTTP.

Cold Starts



Azure Functions Lifecycle

Cold Starts



The function lifecycle

Download **your code**

Start new **container**

Bootstrap the **runtime**

Start your **code**

Full cold start

Partial cold start

Warm start

AWS **optimization**

Your **optimization**

AWS Lambda Lifecycle

Cold Starts - Improvements

We can try to keep the functions warm:

- Call the function on a schedule (e.g. every 15 minutes)
- Good enough solution, but might still run into cold-starts when scaling up the underlying servers
    - One server runs multiple functions
- Can be scheduled either using dedicated services (e.g. Google Cloud Scheduler) or using features of Load Balancers or Monitoring Software.

Some providers might offer a way to provision a minimum amount of functions that are always ready (deviates a bit from the serverless principles):

- AWS Lambda Provisioned Concurrency
- Azure Functions Premium Plan and App Service Plan

FaaS - Common Limits/Quotas

Resources

- Memory
  - We can usually control the amount of memory available to each function
  - CPU scales proportionally
  - **Maximum memory is limited** (e.g. 10GB for Lambda, 4GB for Google Functions)
- Deployment size
  - Usually limited to a few hundred MBs.
  - Dependencies might cause issues
- Concurrency
  - New functions will be instantiated if needed, but up to a point (commonly 1000).
- Input
  - The size of the input data is usually limited to a few MBs

FaaS - Common Limits/Quotas

Duration

- **Functions were not designed for long running processes**
  - E.g. you cannot run a Wordpress website on them
- Commonly limited to a few minutes (e.g. max 15 minutes for AWS Lambda)
- They also become really expensive when running for long (longer than a few seconds).

See the complete service quotas here:

- AWS Lambda
- Google Cloud Functions
- Azure Functions

Serverless Databases

- “Serverless-Native” - designed with the serverless paradigm in mind (NoSQL)
  - E.g. Google (Firebase) Firestore, FaunaDB
  - No provisioning, just pay for data storage and operations (reads, writes, deletes)
- Hybrid, NoSQL - have a serverless mode alongside a provisioned mode
  - E.g. AWS DynamoDB, Azure Cosmos DB (still in preview)
  - Usually based on some capacity units (e.g. Cosmos DB has request units) that are managed by other services (e.g. AWS Cloudwatch).
- Hybrid, SQL
  - E.g. AWS Aurora Serverless, Azure SQL Database (Serverless Tier)
  - Scaling still takes a bit of time, but services are getting better (e.g. Aurora Serverless v2)

Serverless databases tend to be more expensive than their provisioned counterparts (when running continuously)

Serverless Storage

Most object storage services are serverless

- E.g. AWS S3, Azure Blob Storage, Google Cloud Storage

Are usually leveraged by serverless compute services for storing the application files.

Can act as a trigger for compute - e.g. when an image is uploaded, a function is invoked automatically to resize the image (be careful to not run into infinite loops)

Closing Remarks

- Trying out ideas and building is more accessible than ever
  - Most projects can stay within free tiers
- Massive scale - can be good, can be bad
- Infinite loops are costlier than ever
- NoSQL and serverless is a great match
- There is room for both serverful and serverless paradigms

Resources

AWS My Architecture - Nordstrom

David Schmitz’ Presentation

Where Should I Run My Code? (‘19) (also see the ‘18 one)

Azure Functions Cold Starts

Serverless Horror Stories