Serverless Architectures

COMP6205: Web Development

Dr A. Rezazadeh
November 16
What is Serverless Architecture?

• **Serverless** architectures refer to applications that significantly depend on
  - third-party services (known as **Backend as a Service** or "BaaS") or
  - on custom code that's run in ephemeral containers (known as **Function as a Service** or "FaaS")

• By using these ideas, and by moving much behavior to the front end, such architectures remove the need for the traditional *always on* server system sitting behind an application.
  - Depending on the circumstances, such systems can significantly reduce operational cost and complexity at a cost of vendor dependencies.
Backend as a Service

- Web and mobile apps require a similar set of features on the backend, including push notifications, integration with social networks, and cloud storage.
  - Each of these services has its own API that must be individually incorporated into an app, a process that can be time-consuming and complicated for app developers.
- BaaS providers form a bridge between the frontend of an application and various cloud-based backends via a unified API and SDK.
  - Providing a consistent way to manage backend data means that developers potentially saving both time and money.
Functions as a Service

• Serverless can also mean applications where some amount of server-side logic is still written by the application developer but
  – unlike traditional architectures this is run in stateless compute containers that are event-triggered, ephemeral (may only last for one invocation), and fully managed by a 3rd party.

• One way to think of this is ‘Functions as a Service / FaaS’.
  – AWS Lambda and Azure Functions are examples of implementations of FaaS at present, but there are others.
Advantages of Serverless Architectures

1. Horizontal scaling is completely automatic, elastic, and managed by the provider.
   - For example, if your system needs to be processing 100 requests in parallel the provider will handle that without any extra configuration on your part.

2. Functions in FaaS are triggered by event types defined by the provider.
   - With Amazon AWS such stimuli include S3 (file) updates, time (scheduled tasks) and messages added to a message bus (e.g. Kinesis).
   - Most providers also allow functions to be triggered as a response to inbound http requests, typically in some kind of API gateway.
Advantages of Serverless Architectures

3. Serverless is about running backend code without managing your own server systems or your own server applications.
   - The second clause - server applications - is a key difference when comparing with other modern cloud related concepts like containers and PaaS (Platform as a Service.)

4. Since we have no server applications to run, deployment is very different to traditional systems. We upload the code to the FaaS provider and it does everything else.
   - that typically means uploading a new definition of the code (e.g. in a zip or JAR file), and then calling a proprietary API to initiate the update.
Microservices Architecture

Monolithic deployment approach

- A traditional application has most of its functionality within a few processes that are componentized with layers and libraries.
- Scales by cloning the app on multiple servers/VMs.

Coarse-grained density of apps/services

Need to deploy the full application

Microservices application approach

- A microservice application segregates functionality into separate smaller services.
- Scales out by deploying each service independently with multiple instances across servers/VMs.

Independent deployment of microservice

Fine-grained density of services

App 1

App 2
Data sovereignty per Microservice

Data in Traditional approach

- Single monolithic database
- Tiers of specific technologies

Data in Microservices approach

- Graph of interconnected microservices
- State typically scoped to the microservice
- Remote Storage for cold data

- Cache doesn’t help much for massive data ingress (Events, IoT, etc.)
- Database servers are usually the bottleneck
- Monolithic Databases are shared across services.

Each microservice owns its model/data!
An Example of Microservice-based App

**eShopOnContainers reference application**
(Development environment architecture)

**Client apps**
- **eShop mobile app**
  - Xamarin.Forms
  - C#
  - xPlat. OS: iOS, Android, Windows
- **eShop traditional Web app**
  - HTML
- **eShop SPA Web app**
  - TypeScript/Angular 2

**Docker Host**
- **eShop WebApp MVC**
  - ASP.NET Core MVC

**Microservices**
- **Identity microservice (STS+users)**
  - SQL Server database
- **Catalog microservice**
  - SQL Server database
- **Ordering microservice**
  - Ordering.API
  - GracePeriod worker svc.
  - SQL Server database
- **Basket microservice**
  - Redis cache
- **Marketing microservice**
  - MongoDB / CosmosDB
  - SQL Server
- **Locations microservice**
  - MongoDB / CosmosDB

**Event Bus**
- RabbitMQ
- Azure Service Bus

**COMP6205 - Serverless Architecture**
Challenges of Service-based Architecture

• **Challenge #1: How to define the boundaries of each microservice**
  
  – Defining microservice boundaries is probably the first challenge anyone encounters.
  
  – Each microservice has to be a piece of your application and each microservice should be autonomous with all the benefits and challenges that it conveys.

• **But how do you identify those boundaries?**
  
  – You must try to identify decoupled islands of data and different contexts within the same application.
  
  – You always attempt to minimize the coupling between those microservices.
Challenges of Service-based Architecture

• **Challenge #2: How to create queries that retrieve data from several microservices?**
  
  – An example could be a single screen from a mobile app that needs to show user information that is owned by the basket, catalog, and user identity microservices.

  – Another example would be a complex report involving many tables located in multiple microservices.

• **API Gateway.** For simple data aggregation from multiple microservices that own different databases, the recommended approach is an aggregation microservice referred to as an API Gateway.
Challenge #3: How to achieve consistency across multiple microservices?

- The data owned by each microservice is private to that microservice and can only be accessed using its microservice API.
- To make an update to the Basket microservice, the Product microservice should use eventual consistency probably based on asynchronous communication such as integration events (message and event-based communication).
Challenge #4: How to design communication across microservice boundaries

- **Event-Driven Architectures** use a publish-subscribe (pub-sub) model, where producers publish events, and consumers subscribe to them. The producers are independent from the consumers, and consumers are independent from each other.
Managing Microservices

- Microservices architecture is an efficient mechanisms of solving modern enterprise problems.
- However, this simplification comes at a cost. Manually managing hyperscale deployments of Microservices is nearly impossible.
- Automating Microservices lifecycle management becomes an inevitable requirement to achieve enterprise-grade environment stability.
- **Azure Service Fabric** is a distributed systems platform that makes it easy to package, deploy, and manage scalable and reliable Microservices.
Architecture of Azure Service Fabric

- Service Fabric is a collection of services grouped into different subsystems.