Micro Service

Architecture

Agenda

- Session 1: Evolution of Software Architecture
- Session 2: Basics of Microservice Architecture
- Session 3:
    - Hands on experiences on Microservice Architecture
    - Case Studies
        - WiNiT
        - TEQ

Raisul Islam

Lead Engineer & Solution Architect

aws certified Solutions Architect Associate

aws certified Developer Associate

aws certified Cloud Practitioner

Scrum Alliance CSPO CERTIFIED

Scrum Alliance CSM CERTIFIED

Scrum Alliance CSD CERTIFIED

PSM I

BRAIN STATION 23

Skype: raisulislam.bs23
Mobile: +88 01911 810019
Email: raisulislam@brainstation-23.com

My Experience

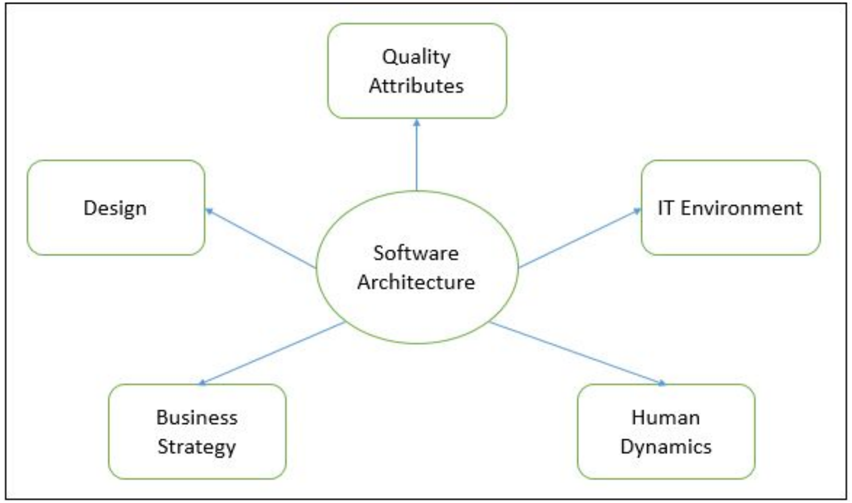
- 10+ years development experience in .NET and Python
- 5+ years working experience as a Solution Architect and Development Manager
- Designed 25+ Solution Architectures
- Experienced in Micro Service and Distributed System Architecture
- Experienced with AWS Serverless Architecture
- Worked on 45+ Projects in following business domains:
    - E-Commerce
    - ERP
    - Educational Institution
    - Defence Organization
    - Transportation and Fleet Management
    - HRM
    - Supply Chain
    - CRM and Customer Support
    - Mobile App and Game Development
    - Public facing Web Application with Big Data Handling

Evolution of Software Architecture

- What is Software Architecture
- Importance of Software Architecture
- Steps of Software Architecture
- Well-Architected Framework
- Evolution of Software Architecture
- Monolithic Architecture
- Monolithic to Microservice transformation
- Strategies to migrate Monolithic to Microservice

What is Software Architecture

- Software Architecture represents the blueprint of a system
- It defines a structured solution to meet all the technical and operational requirements
- Software Architecture is always an Evolving process
- Its must frequently evolve to cope with changing requirements



Importance of Software Architecture

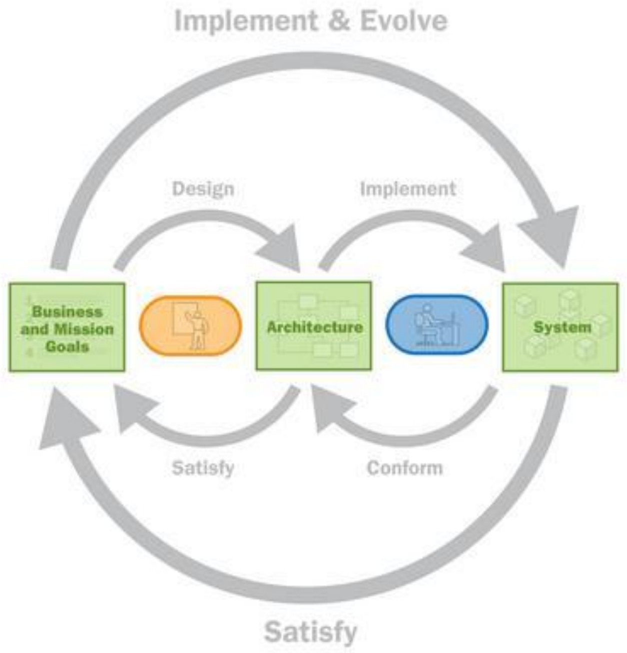
There is a saying, “Failing to plan is planning to fail”

- A software architecture is the foundation of a software system
- Developing a software is similar to constructing a building
- Scalability, Performance, Security and Usability is highly dependent on the Software Architecture

Steps of Software Architecture

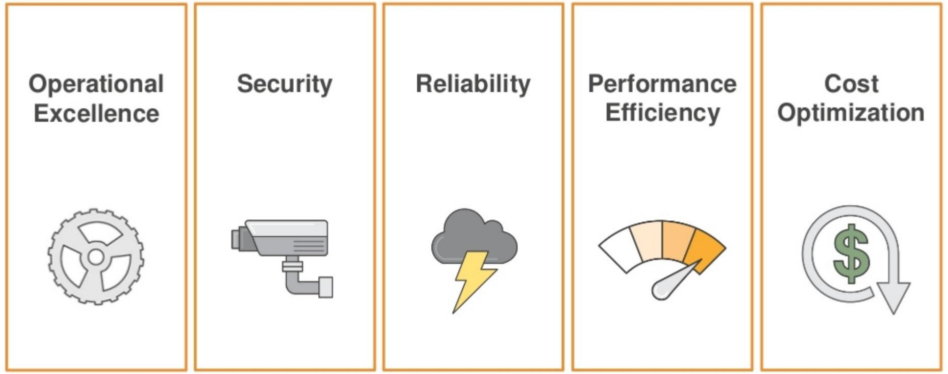
- Capture architecturally significant requirements
- Design an architecture
- Evaluate the architecture
- Document the architecture

Software Architecture is an iterative and evolving process. If the architecture can not be evolved, the software is not sustainable.

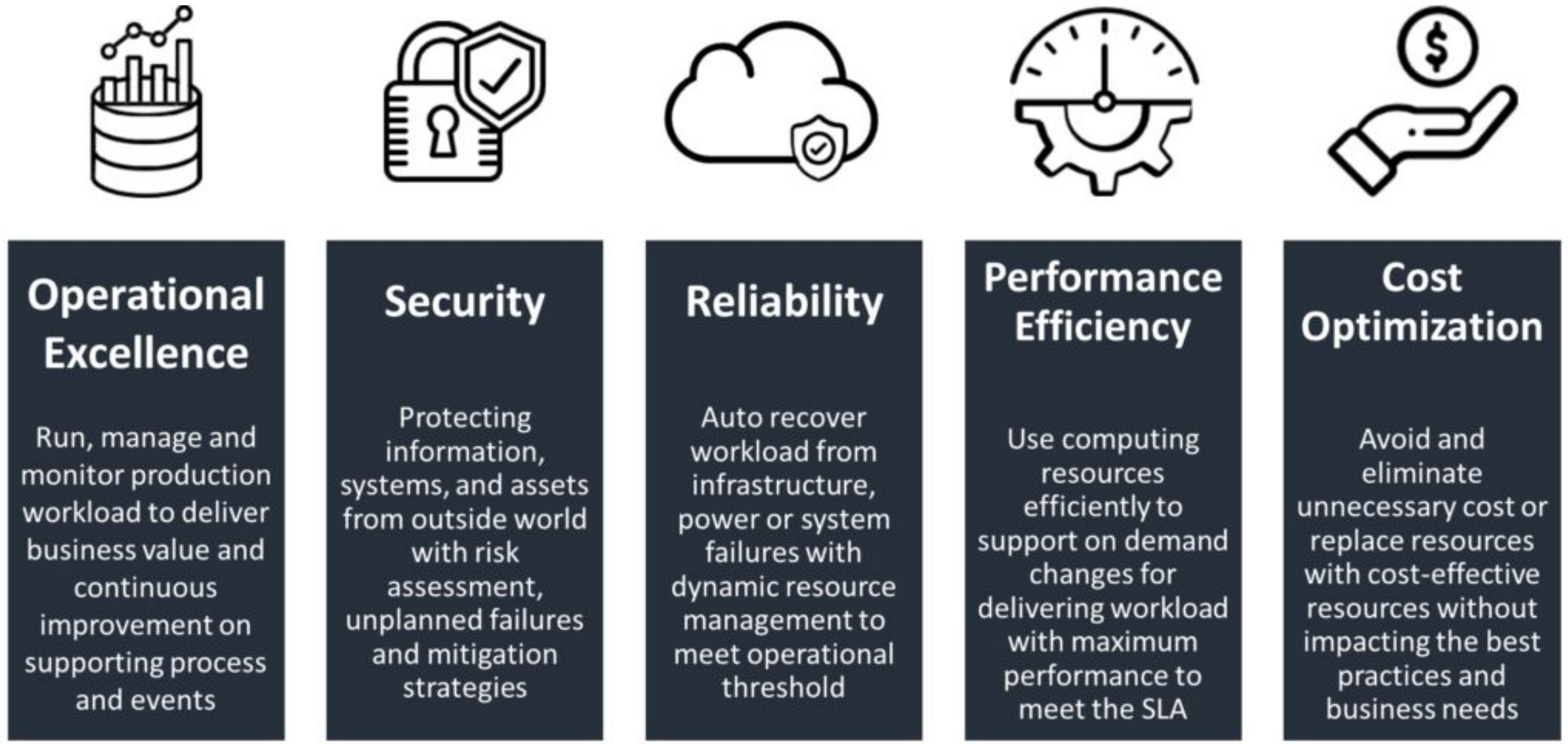


Well-Architected Framework

- The **Well-Architected** framework has been developed to build the most secure, high-performing, resilient, and efficient infrastructure for the applications
- This framework provides a consistent approach for customers and partners to evaluate architectures, and provides guidance to help implement designs that will scale with your application needs over time



Well-Architected Framework (Cont.)

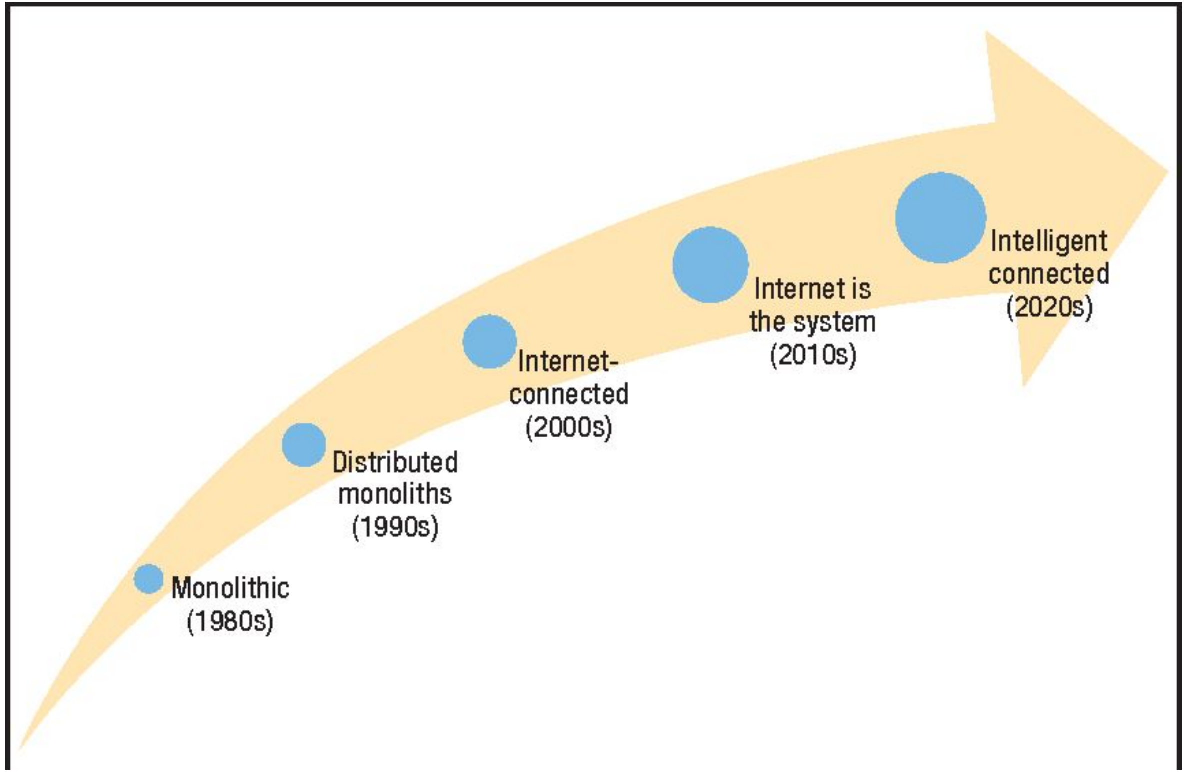


Operational Excellence	Security	Reliability	Performance Efficiency	Cost Optimization
Run, manage and monitor production workload to deliver business value and continuous improvement on supporting process and events	Protecting information, systems, and assets from outside world with risk assessment, unplanned failures and mitigation strategies	Auto recover workload from infrastructure, power or system failures with dynamic resource management to meet operational threshold	Use computing resources efficiently to support on demand changes for delivering workload with maximum performance to meet the SLA	Avoid and eliminate unnecessary cost or replace resources with cost-effective resources without impacting the best practices and business needs

Fig 1: Pillars of AWS Architecture

Evolution of Software Architecture

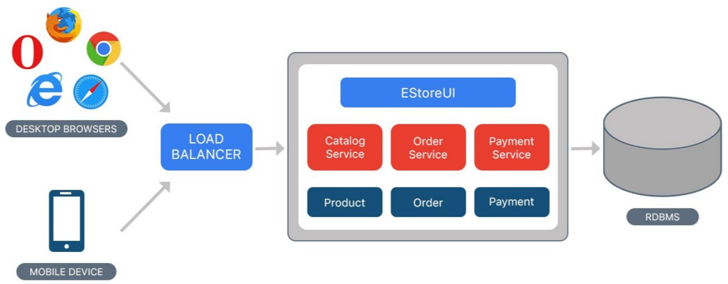
- 1980s: Most of the applications were console based application build on Monolithic applications
- 1990s: Windows came to the limelight, Desktop applications were developed
- 2000s: Internet changed everything, web applications dominated
- 2010s: Cloud Technologies came in front, Cloud overtaken on-premises
- 2020s:


Monolithic (1980s)
Distributed monoliths (1990s)
Internet-connected (2000s)
Internet is the system (2010s)
Intelligent connected (2020s)

Monolithic Architecture

- The **Monolithic** application describes a single-tiered **software** application in which different components combined into a single program from a single platform
- It's designed to be self-contained
- Single point to serve all request.

Example for Monolithic Approach



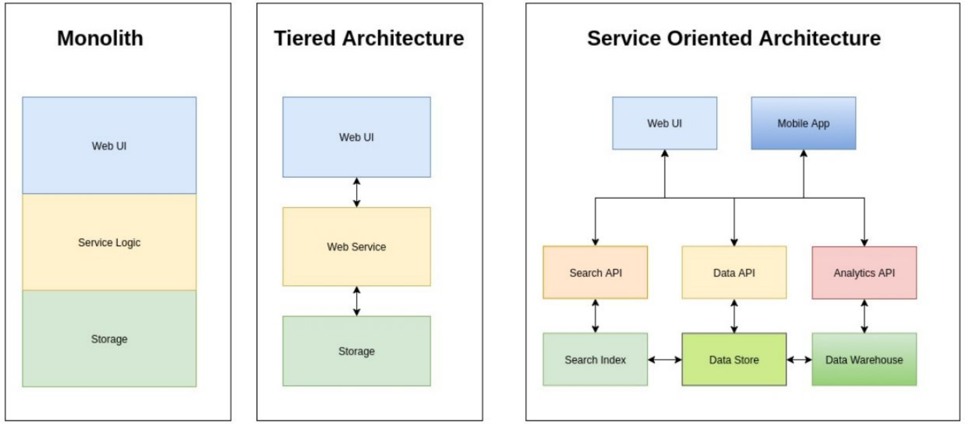
- Simple to develop — At the beginning of a project it is much easier to go with Monolithic Architecture.
- Simple to test. For example, you can implement end-to-end testing by simply launching the application and testing the UI with Selenium.

Characteristics:

- Simple to develop
- Simple to deploy
- Simple to scale horizontally
- Easy to test
- It's easy to start but hard to expand
- Hard to maintain — If Application is too large and complex

Monolithic -> Microservice Transformation

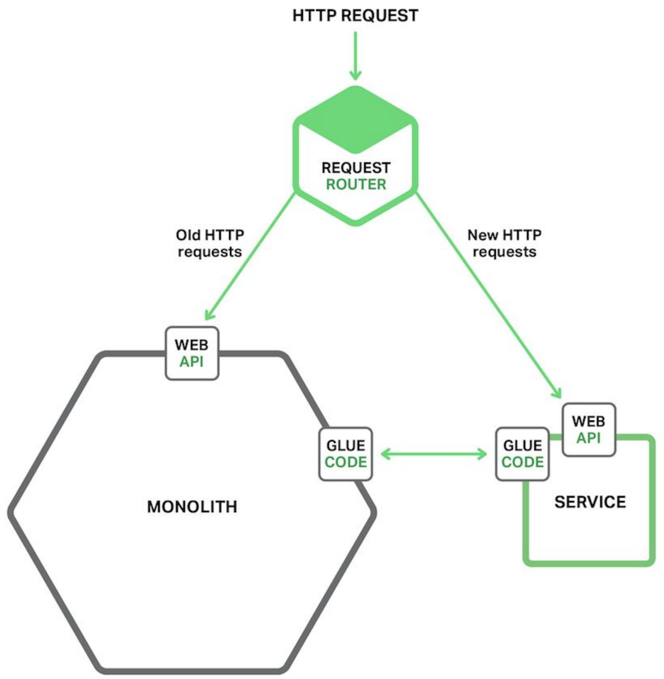
- Scalability is one of the primary motivations for moving to a microservice architecture.
- First breakdown the Monolithic applications in multiple segments
- Never jump directly to the Service Oriented Architecture, rather try to split the application part by part
- Multi-tier Architecture can be a good approach to break down the Monolithic applications



Strategies to migrate Monolith to Microservice

Strategy 1 - Stop Digging:

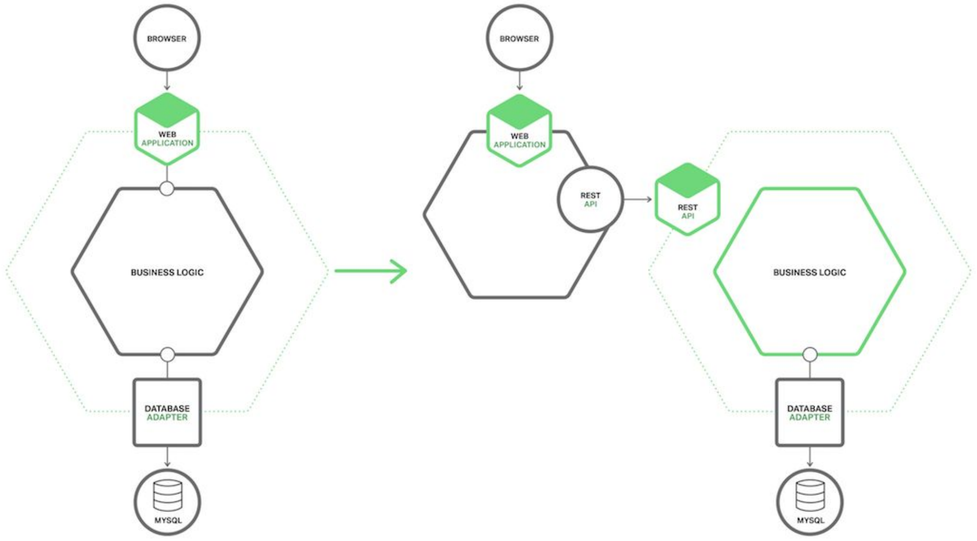
- The Law of Holes says that whenever you are in a hole you should stop digging. This is great advice to follow when the monolithic application has become unmanageable
- Stop making the monolith bigger
- Put that new code in a standalone microservice.



Strategies to migrate Monolith to Microservice (Cont.)

Strategy 2 – Split Frontend and Backend:

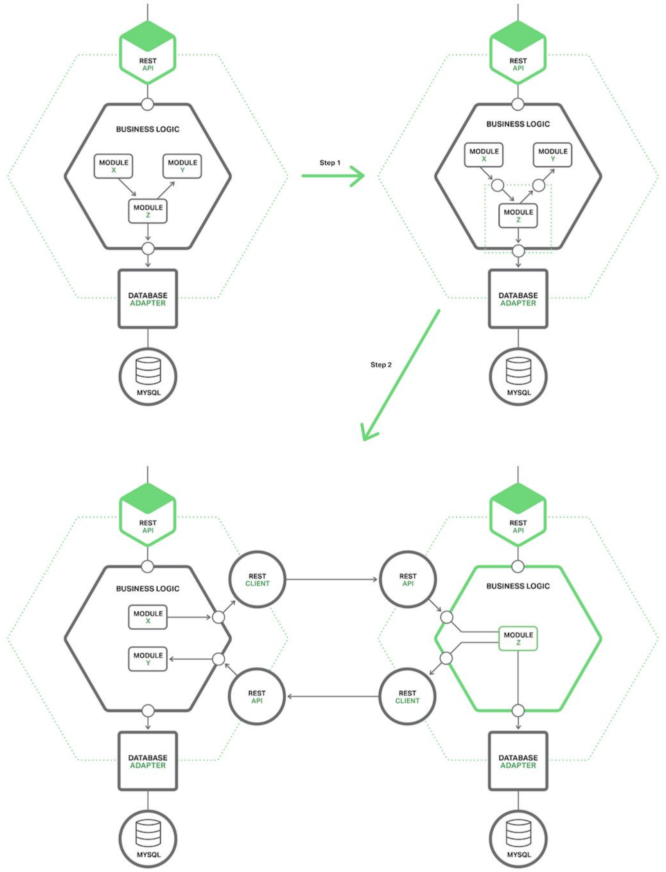
- A strategy that shrinks the monolithic application is to split the presentation layer from the business logic and data access layers
- A typical enterprise application consists of at least three different types of components:
  - Presentation layer – Components that handle HTTP requests and implement either a (REST) API or an HTML-based web UI. In an application that has a sophisticated user interface, the presentation tier is often a substantial body of code.
  - Business logic layer – Components that are the core of the application and implement the business rules.
  - Data-access layer – Components that access infrastructure components such as databases and message brokers.
- There is usually a clean separation between the presentation logic on one side and the business and data-access logic on the other



Strategies to migrate Monolith to Microservice (Cont.)

Strategy 3 – Extract Services:

- Prioritizing Which Modules to Convert into Services
- Figuring out which modules to convert first is often challenging. A good approach is to start with a few modules that are easy to extract
- It is also beneficial to extract modules that have resource requirements significantly different from those of the rest of the monolith
- After extracting a module, turns the module into a standalone service



Example

Traditional Ecommerce Website

Questions

Microservice Architecture

- Basic Understanding
- Characteristics
- How it works!
- Advantages
- Challenges
- When to use!
- When not to use!
- Docker

Microservice - Basic Understanding

Microservice architecture is an architectural style that structures an application as a collection of services that are -

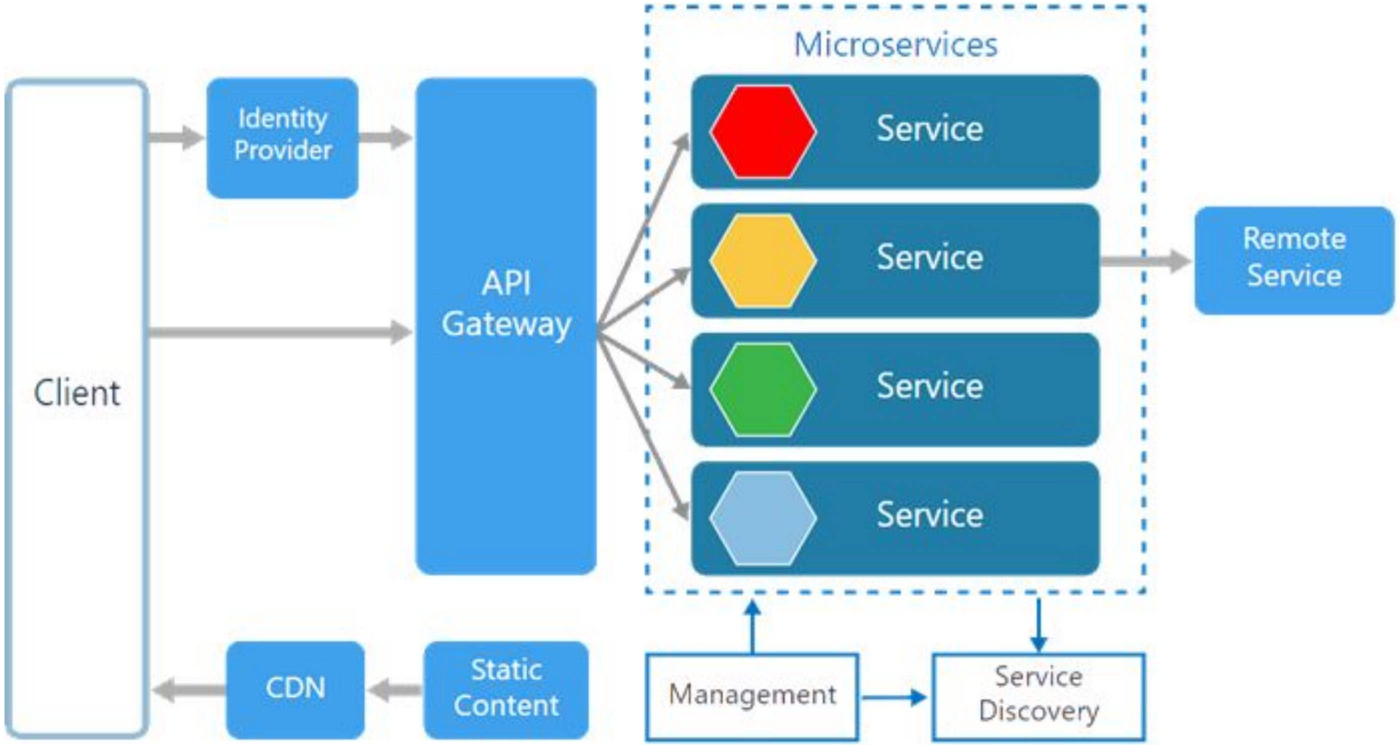
- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Each service is owned by a small team

The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organization to evolve its technology stack.

Microservice - Characteristics

- Every service should function as an independent component and consider as a product
- Every service must be loosely coupled
- Decentralized Technology
- Decentralized Data Model
- Scale where necessary
- Design with fault tolerance
- Services are reusable
- Supports Agile development
- Deployable independently

Microservice - How it works!



Microservice - Advantages

- The microservices can be independently tested and deployed. The smaller the unit of deployment, the easier the deployment.
- They can be implemented in different languages and frameworks. For each microservice, best technology can be chosen for its particular use case.
- They can be managed by different teams. The boundary between microservices makes it easier to dedicate a team to one or several microservices.
- Team works only with specific services so don’t need to have the complete business knowledge of the overall system and dependencies
- Microservice is designed for failure. By having clear boundaries between services, it's easier to determine what to do if a service is down.

Microservice - Challenges

- **Distribution**: Distributed systems are harder to program, since remote calls are slow and are always at risk of failure.
- **Eventual Consistency**: Maintaining strong consistency is extremely difficult for a distributed system, which means everyone has to manage eventual consistency.
- **Operational Complexity**: Need a mature operations team to manage lots of services, which are being redeployed regularly
- **Security:** Microservices communicate over a network and In some circumstances, this can be seen as a security concern

Microservice - When to use!

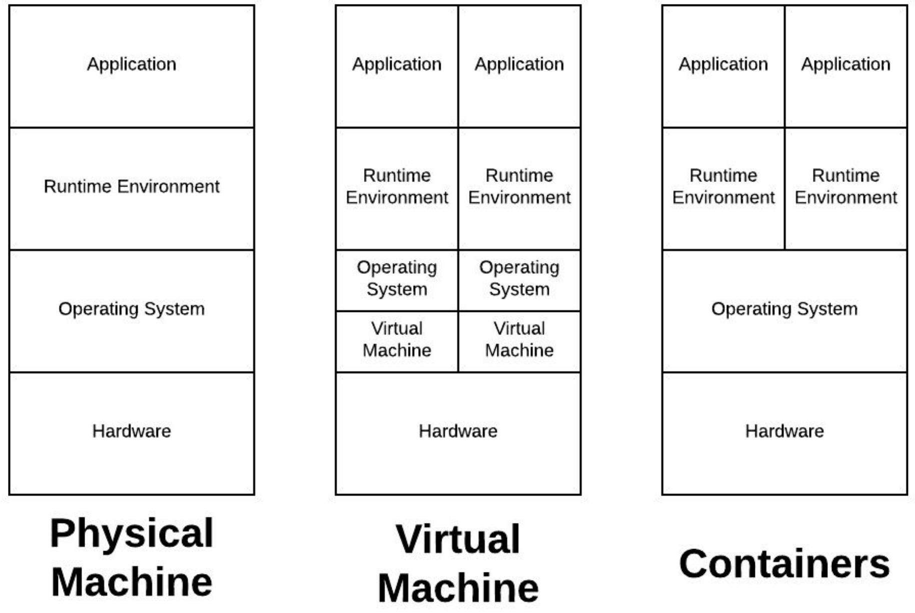
- Business model is proven and is mature enough to adopt microservices
- Support for different variety of clients that includes desktop browsers, mobile browsers and native mobile applications.
- Monolithic application migration due to improvements needed in scalability, manageability, agility or speed of delivery.
- Independent business applications or services reused across multiple channels. For example, payment services, login services, flight search services, customer profile services, notification services, etc.

Microservice - When not to use!

- Microservices are solutions to complex concerns and if the business doesn’t have complex issues, consider that there is no system in place to handle the complexities of microservices.
- Using microservices can prove to offer contrary consequences if the team size is not good enough to handle the tasks involved. This will only result in the delay of delivery.
- Implementing microservices for the sake of it can be hampering as well. If the application does not require to be broken down into microservices, does not require to apply microservice. There is no absolute necessity that all applications should be broken down to microservices. There are those that are simple by their nature and functionality.

Docker & The Rise of Microservices

- Docker is the world's leading software containerization platform.
- It encapsulates the microservice into what we call as Docker container which can then be independently maintained and deployed.
- Each of these containers will be responsible for one specific business functionality



Docker & The Rise of Microservices

Docker architecture

The Docker architecture is broken in three different components and are all needed to set up the Docker environment:

- **Docker Client:** Communicates with the Docker host by sending it a CLI command that Docker can understand.
- **Docker Host:** This platform executes the request from the CLI Docker Client; the platform can be on your computer or in the cloud.
- **Docker Registry:** Stores Docker images.

Microservices orchestration

Docker has three tools for microservices orchestration:

- **Docker Machine:** You can ask any cloud vendor for a Docker Machine instance. You can provide them a file and say I need this container, they will help you set up a Docker daemon, and you'll be running the command line as if you're somewhere on the cloud.
- **Docker Composer:** You would need it where you have one or more containers that support one use case. It allows you to define all your images and containers. It defines various ports or allies where containers can talk to each other. And using docker machine you can run this on the cloud.
- **Docker Swarm:** Allows you to orchestrate on a much larger scale.

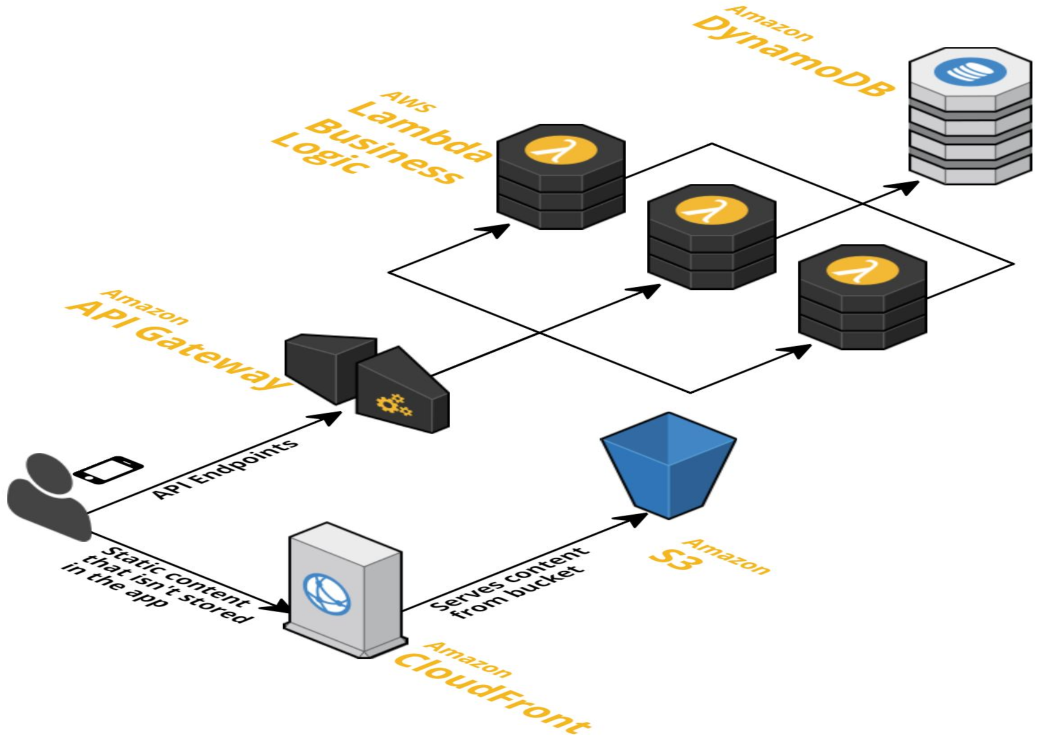
Serverless Architecture

- Microservice - Serverless
- How it works
- Why use serverless?
- When to Choose Serverless?
- Must have in Serverless Architecture
- Challenges

Microservice - Serverless

- Serverless architecture is a way to build and run applications and services without having to manage infrastructure.
- It is also known as Function as a Service(Faas).
- Fundamentally, its is about writing and running backend code without have to provision, scale, and maintain own server systems or own long-lived server applications.
- Focus on your **application**, not on the **infrastructure**

Serverless Architecture Sample for AWS



How it works

- Each Serverless function runs in its own container.
- When a function is created, packages it into a new container and then executes that container on a multi-tenant cluster of machines managed by vendor service.
- Before the functions start running, each function's container is allocated its necessary RAM and CPU capacity.
- When a function is started in response to an event, there may be a small amount of latency between the event and when the function run. It is called cold start time.

Why use Serverless?

Reduced Operational Responsibilities

Flexible scaling

Pay for value

Automated high availability

Increased Agility and Innovation

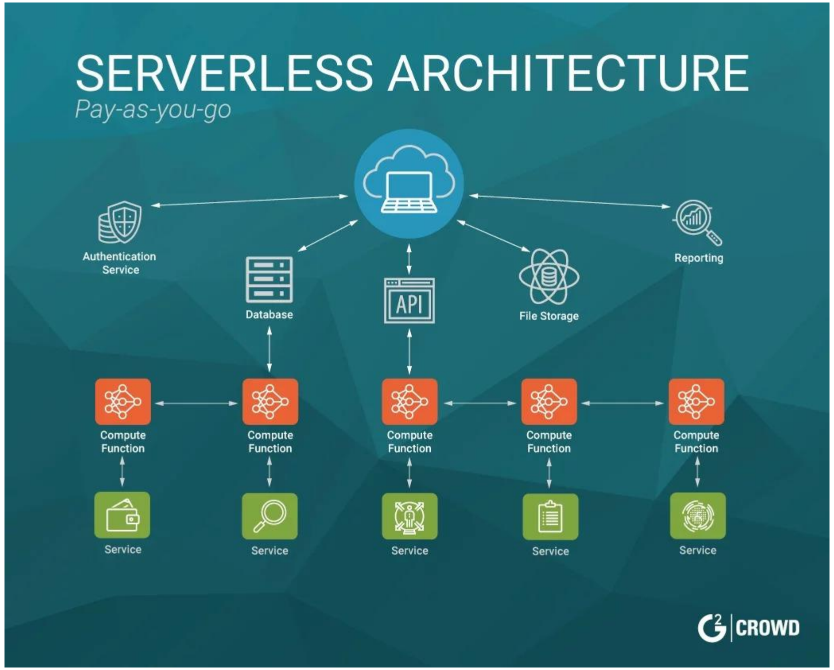
Built in Fault Tolerance

When to Choose Serverless?

- Unpredictable user base
- HIgher availability
- Automatic and Dynamic scaling
- Lower runtime costs
- Consistent performance
- Services that don't need to run all the time
- High latency background tasks like multimedia or data processing
- Architecture is service oriented

Must have in Serverless Architecture

- API and Microservices Design
- Event-driven Architectures and Asynchronous Messaging Patterns
- Lambda Computing Environment and Programming Model
- Serverless Identity Management, Authentication, and Authorization
- End-to-End Security Techniques
- Application Observability with Comprehensive Logging, Metrics, and Tracing



Challenges

- Less control over the infrastructures and system.
- Less opportunity of attaching other operational tools.
- Need to deploy every function separately.
- Limited runtime.

Example

Single Sign On (SSO)

Questions

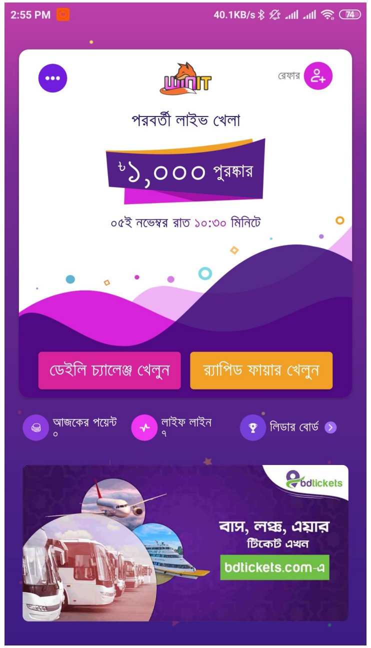
Hands on experiences with Docker

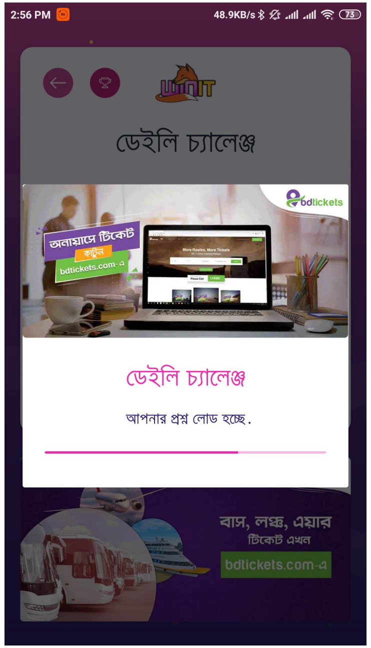
Microservice Prototype

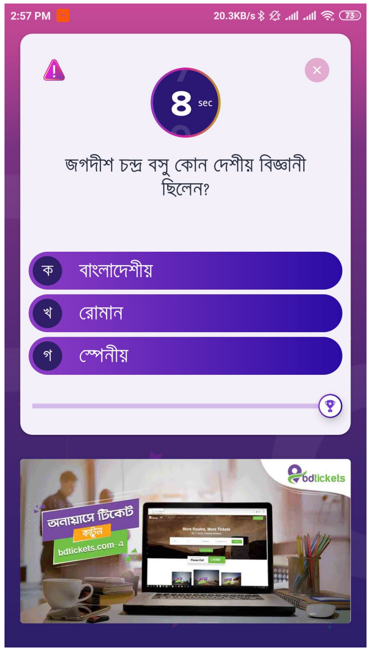
https://github.com/raisul-bs23/django-microservice-prototype.git

Real Life Applications

WiNiT - Microservice Based on Serverless Arch.

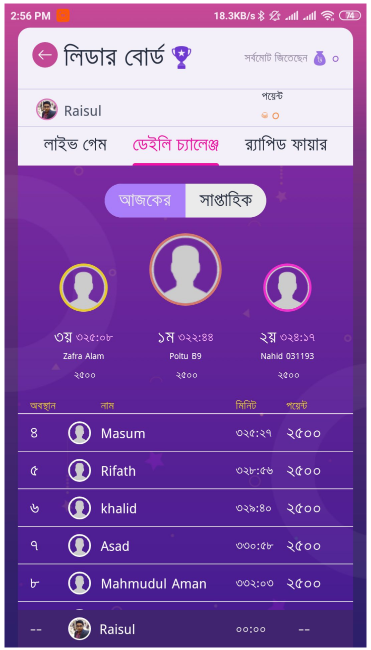






জগদীশ চন্দ্র বসু কোন দেশীয় বিজ্ঞানী ছিলেন?

ক বাংলাদেশীয়
খ রোমান
গ স্পেনীয়



অবস্থান	নাম	মিনিট	পয়েন্ট
৪	Masum	৩২৫:২৭	২৫০০
৫	Rifath	৩২৮:৫৬	২৫০০
৬	khalid	৩২৯:৪০	২৫০০
৭	Asad	৩৩০:৫৮	২৫০০
৮	Mahmudul Aman	৩৩২:০৩	২৫০০
--	Raisul	০০:০০	--

WiNiT - AWS Services

AWS Lambda

Amazon API Gateway

Amazon Cognito

Amazon DynamoDB

Amazon Simple Storage Service

Amazon Simple Notification Service

Amazon Simple Queue Service

AWS Elemental MediaLive

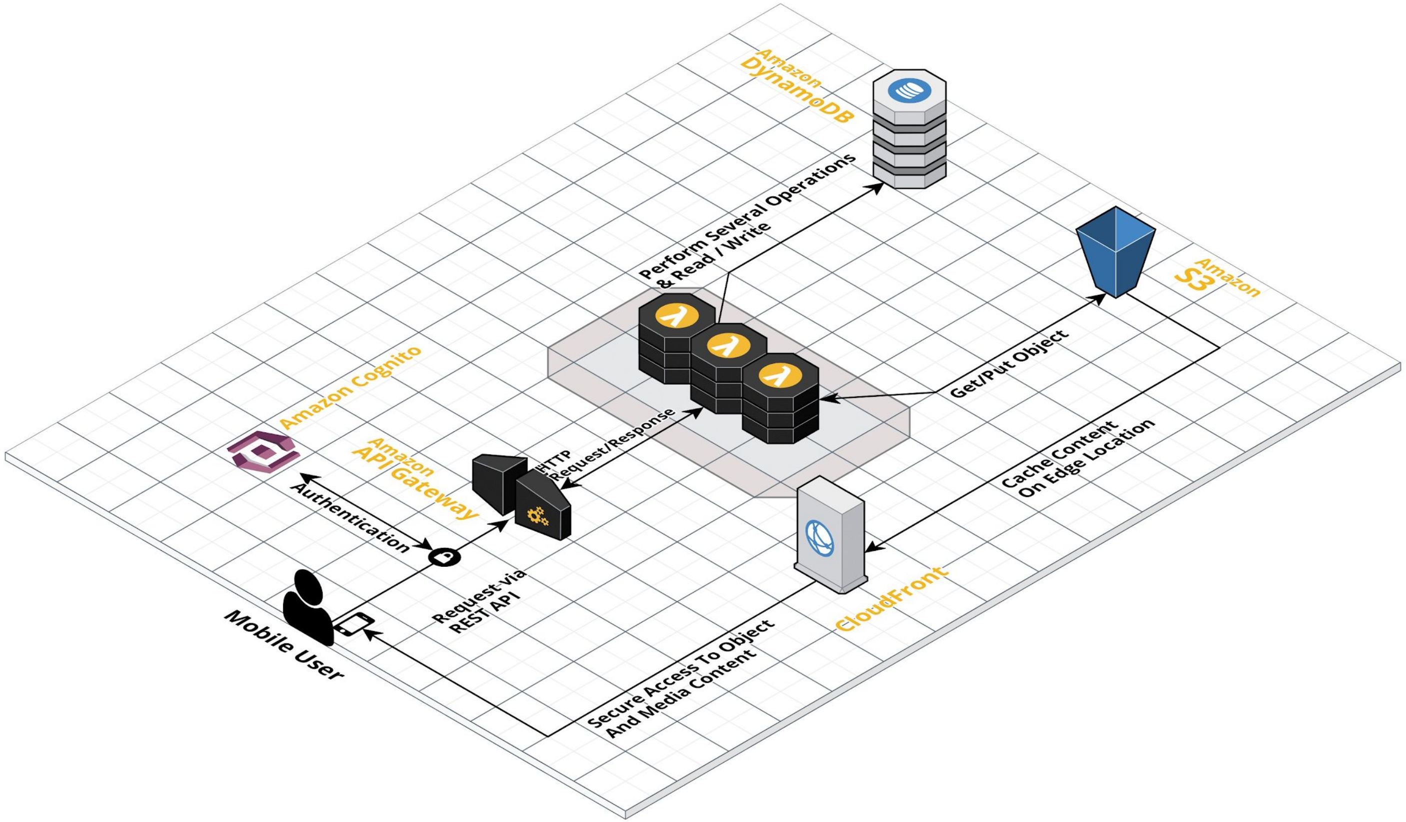
Amazon CloudWatch

AWS CloudTrail

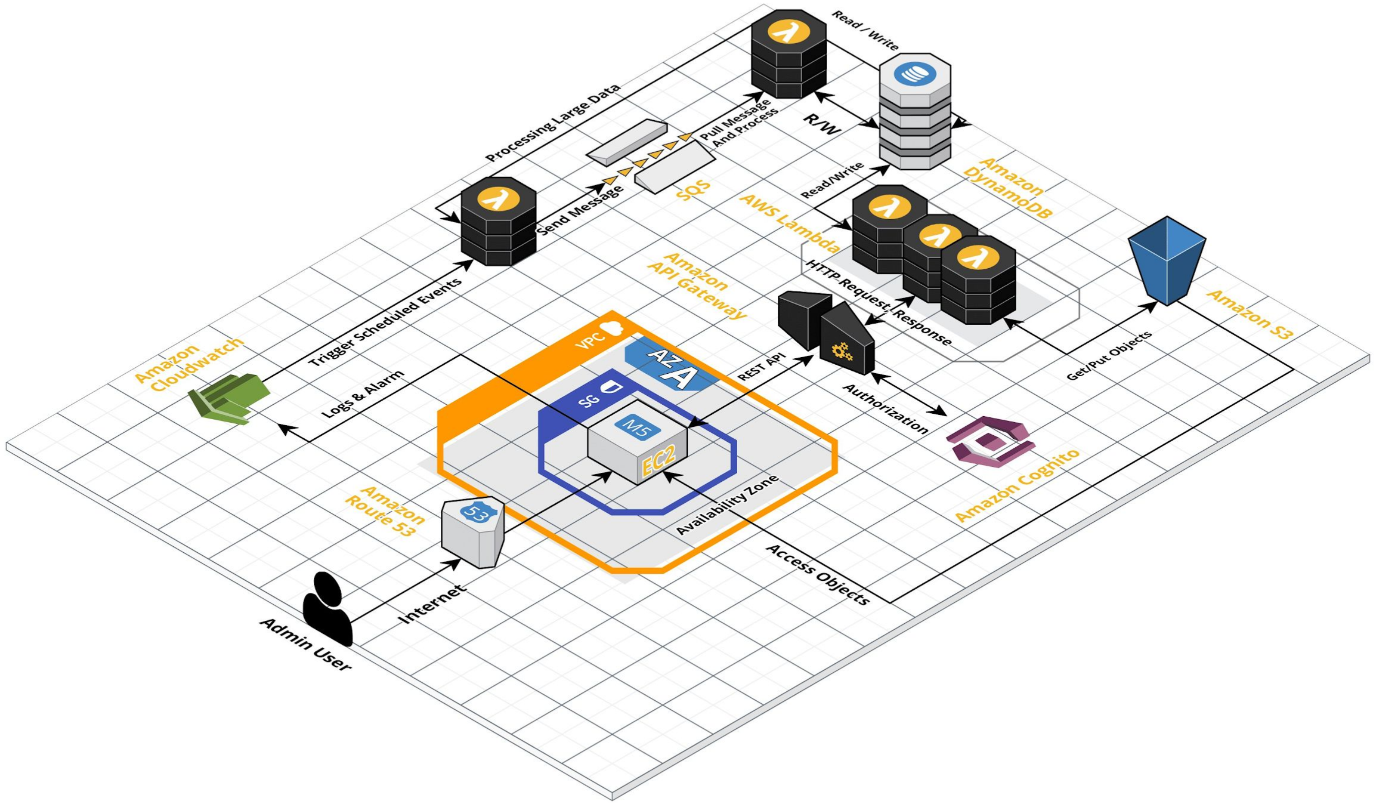
Amazon CloudFront

Amazon Route 53

WiNiT - Mobile Backend Architecture



WiNiT - Web Portal Backend Architecture

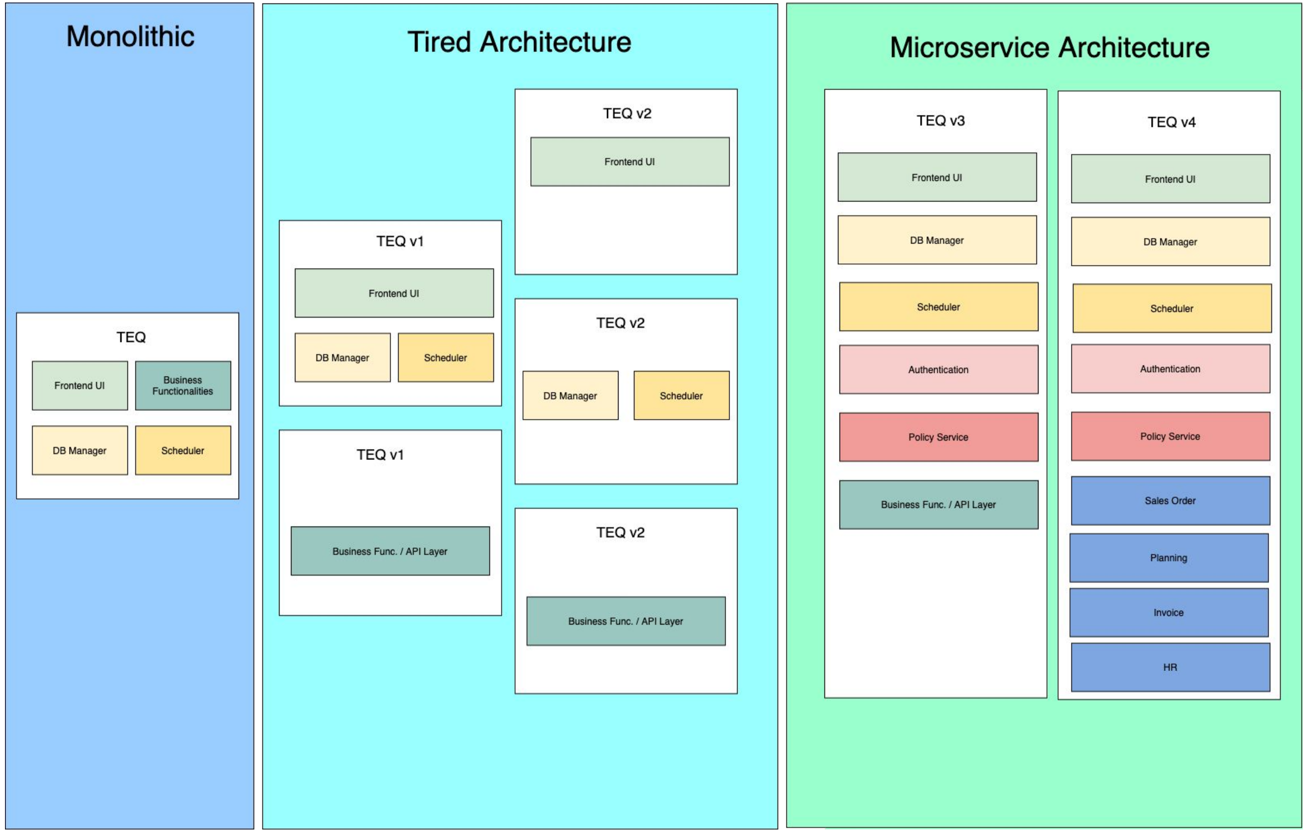


WiNiT - Some Statistics (January, 2021)

API Gateway Requests	298 M Requests
CloudFront Data Transfer Out	343 GB
CloudWatch Logs	104 GB
S3 Data Requests	150 M Requests
DynamoDB	3 B ReadRequestUnits 435 M WriteRequestUnits
Lambda	300 M Requests 50 M Lambda-GB-Second
Simple Queue Service	4 M Requests

TEQ - Transformation

- Started with the Monolithic Architecture in 2016
- Started to split components in Multiple Modules and Layers in mid 2017
- Started the migration in Service Oriented Architectures from Q4 2020.
- Plan to start dividing services into more particles from Q2 2021



References

- https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
- https://www.serverless.com/aws-lambda
- https://martinfowler.com/microservices/
- https://microservices.io/
- https://subscription.packtpub.com/book/application_development/9781788624060
- https://www.nginx.com/blog/refactoring-a-monolith-into-microservices/
- https://en.wikipedia.org/wiki/Law_of_holes
- https://developer.ibm.com/articles/breaking-down-docker-and-microservices/
- https://aws.amazon.com/architecture/well-architected/?wa-lens-whitepapers.sort-by=item.additionalFields.sortDate&wa-lens-whitepapers.sort-order=desc

Questions

Thank you