

---

# Microservices

— An Introduction —

---

# Background

- In early days of computer science, barriers to entry in programming language were high
- Only **PhD in science** and **computer** can use these programming languages
- As at that time, nearly all use of computers required **writing custom software**
- In **1964**, **Basic** was developed, which was a **general-purpose** programming language



# Background

- It lowers the barriers, as now non-PhD students from other departments can also write programs
- As there was a **rapid growth** of **computing applications** in the 1960s, software became **large and complex**
- Computer Scientists tried to tackle the **complexity of Software Systems** with the ancient and proven technique: **Divide and Conquer**



# Background

- In 1972, **David Parnas** introduced concept of **modularity** and **information hiding** in softwares in his paper
- **Edsger W. Dijkstra** introduced concept **Separation of Concern** in his paper in 1974
- Also works of others lead to the **Modular Software Development** in 1970's
- Modularization on the principle of **decomposing a large, complex software system** into "Loosely coupled, highly cohesive"



# Background

- And these modules **communicate via internal interfaces**
- In simple means:
  - **Loosely coupled** - means the dependency between modules should be very low
  - **highly cohesive** - means that one module should focus on single or similar functionality
- With the **rise of internet and web** in 90's, softwares became widespread in business applications and became even more complex and large.

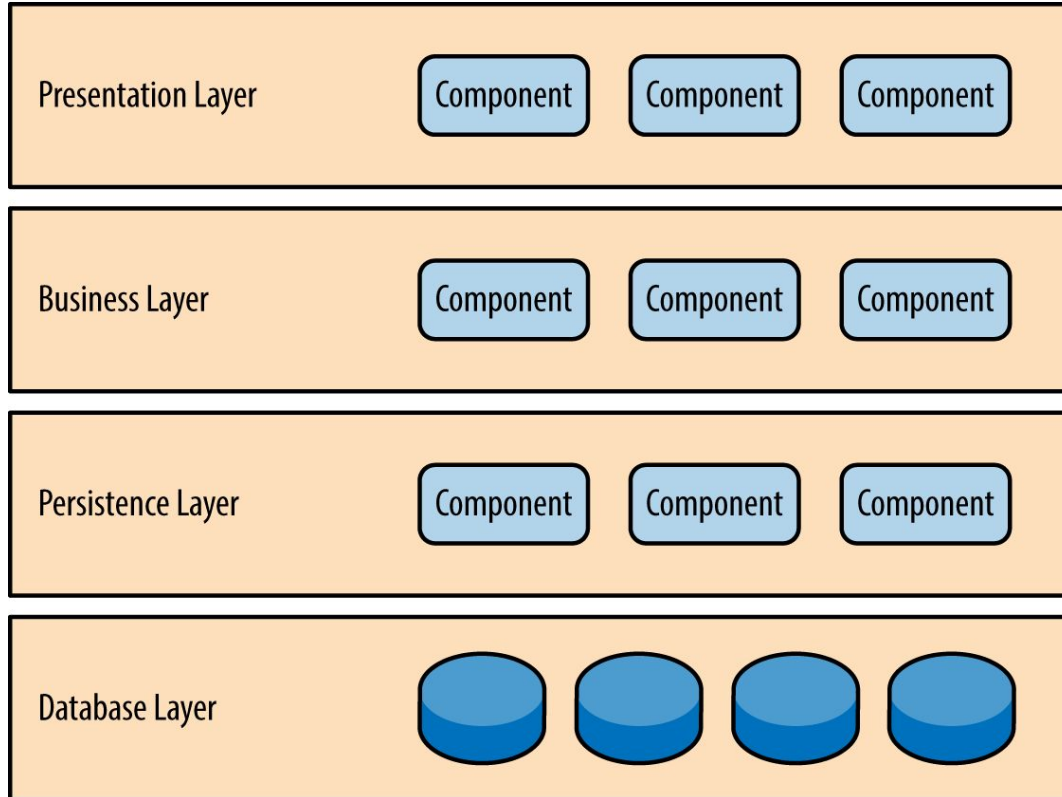


# Background

- Although **Modularity** is used to reduce the complexities of software application
- But often, it did not help as the soft Modular boundaries of software sub-systems are **easy to cross** and **misuse**
- Another **Software Architecture pattern** became very popular during the 1990s to **develop business applications**: **Layered Architecture**



# Background



# Background

- Normally, a business Web Application is **divided into several layers: Presentation, Business, Database** layers
- In 1997, **Brian Foote** and **Joseph Yoder** has analyzed many Business applications and published the “**Big Ball of Mud**” papers





# Background

- The paper states that most of the Business applications suffer from the following problems
  - Unregulated growth
  - Too many responsibilities
  - Lacks proper Architecture
  - Spaghetti Code
  - Make it working aka. sweeping problems under the Carpet

# Background

- In the late 2000s, a **Cambrian Explosion** happens in the software industries due to the rise of Mobile Internet (Wifi, Smartphone) and faster network
- It was the time when softwares started to eat the world
- All types of companies like Banking, Insurance, Restaurants, Hotels, Music, Driving, etc



# Background

- Companies like **Facebook, Twitter, Uber, Netflix, Spotify** came with innovative ideas, aggressive strategy, move fast approach leads to the exponential growth of their applications
- Suddenly, engineers found that Monolithic Architecture cannot handle the challenges of Modern, Fast-Paced or Web-Scale Software development

# Limitations of Monolithic Architecture

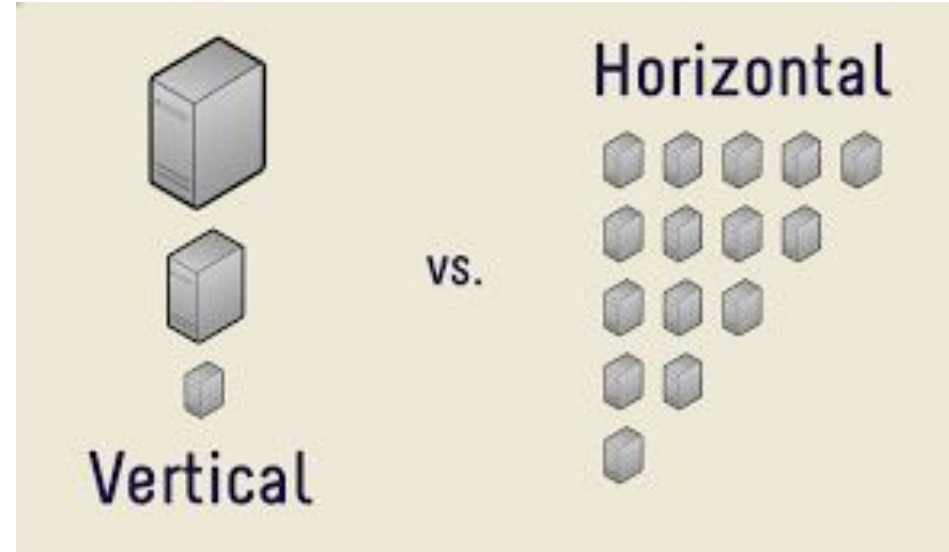
# Application Scaling

- As the successful Web Scale companies enjoy **exponential growth**
- Their softwares also need to **support high horizontal scalability**
- Sometimes, only a part of the software which is e.g. **CPU intensive** or **I/O intensive** needs to be scaled and handled separately (implemented with polyglot programming)
- **Monolithic software works as a single unit** and **developed in a single programming language** using a single Tech Stack



# Application Scaling

- To achieve horizontal scaling, the **whole application needs to be scaled**
- **Monolithic software** only supports one programming language, it is not possible to implement one single module of it in other programming language



# Development Velocity



- To shorten **time to market**, every company nowadays wants to have **fast feature development**
- In a large Monolithic Application, **adding new feature** is **very slow** because such a Monolithic Application gives **huge Cognitive Load** to the Developer
- **Modules of giant Monolithic applications** are **tightly coupled** and provide an additional challenge to add new features
- As a result, **adding new features** in a Monolithic application **become very expensive**



# Development Scaling

- Companies want **parallelizing development** by hiring more developer for fast pace development
- However, developers **cannot work independently** on a Monolithic, tightly coupled code base which **needs extra synchronization**
- Therefore, adding more **developers doesn't produce more feature**
- Similarly due to cognitive load, new hires or fresh graduates take **long time to write first piece of productive code**





# Release Cycle

- Release cycle of large monoliths is even large; usually 6 months to 2 or 3 years
- In today's market, large release cycles can put the company under competitive disadvantages
- As during these gaps a new company can come and take away its market

# Modularization

- In Monolithic Architecture, the boundary between modules are **internal Interfaces**
- As soon as the application grows in size, the **boundary between modules starts to fall apart**
- As a result, often modules in Monolithic Architecture are tightly coupled instead of being **“Loosely coupled, highly cohesive”**



# Modernization

- Existing successful applications **needed to be modernized** due to many factors (e.g. **taking advantage of modern Hardware, Browser, Network Bandwidth or Attract good developers**)
- Modernization of Monolithic application is **expensive** and **time-consuming**
- It needs a **Big Bang modernization** of the whole application **without disrupting the Service**





# Microservice Architecture



# Microservice Architecture

- In the 2010s, other **disruptive technologies** arise which impact the Software Development landscape in a significant way
- **Cloud Computing, Containerization** (Docker, Kubernetes), **DevOps**
- Likewise some **highly productive, lightweight** new programming languages e.g. **Golang, Rust, Swift** comes to scenario
- Some **highly productive, easy to use, lightweight** programming language like **JavaScript, Python** become mainstream

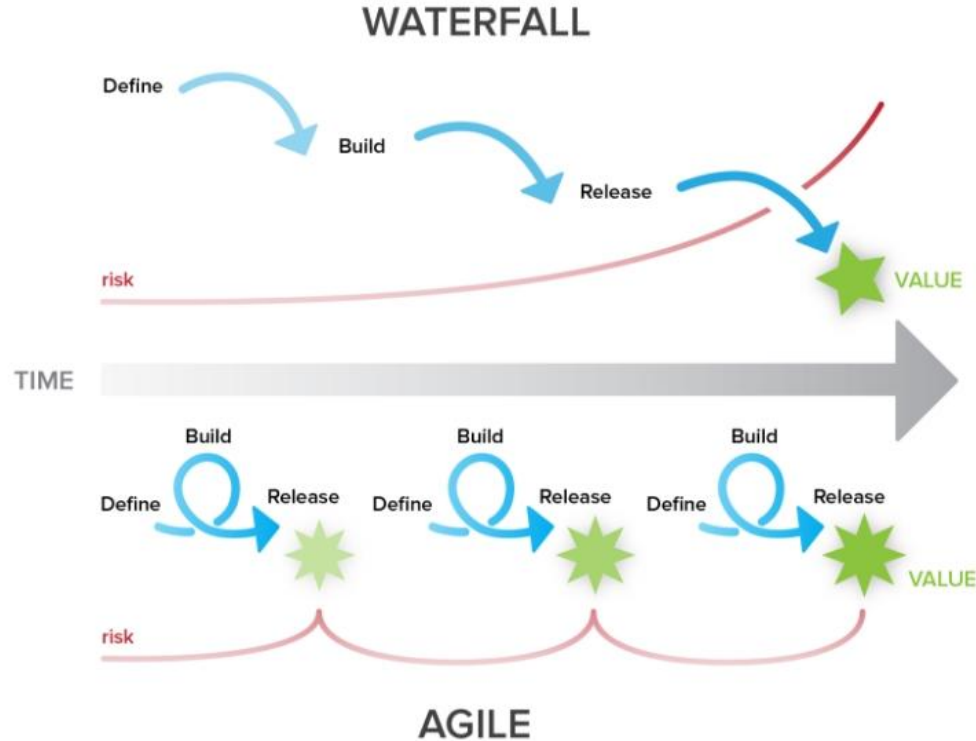


# Microservice Architecture

- There is a **change in Software Development model** also
- **Waterfall** software development model is almost discarded
- Replaced by fast, iterative, incremental Software development methodology: **Agile Software development**
- **Computer Hardware** also changed massively with **cheaper, faster main memory** and rise of **Multi-Core CPU, GPU**
- New **Database technologies** like NoSQL, **NewSQL** emerges and become mainstream



# Microservice Architecture



# Microservice Architecture

- To handle the complexity of modern software applications
- To take the advantages of Cloud Computing, Containerization, DevOps
- To get benefit from modern Programming languages
- To fulfill the need of **modern software development** (fast development, horizontal scaling)
- In 2012, **Microservices Architecture**; a new software architecture style arose





# Microservice Architecture

Definition :

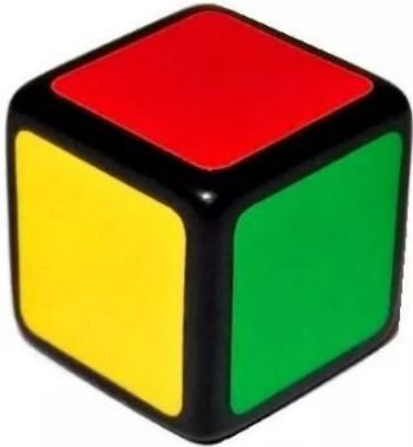
*" **Microservice Architecture** is about decomposing a Software System into autonomous Units which are independently deployable and which communicates via lightweight, language agnostic way and together they fulfill the business goal. "*



# Microservice Architecture

- Microservice Architecture also uses the same technique of divide and conquer
- However, the difference between two is; a microservice can be deployed independently whereas all modules of monolith must be deployed as whole

# Microservice Architecture



Monolith



Modular Monolith



Microservices

# Advantages of Microservices

# Application Scaling

- Microservices are often **Stateless**
- If carefully deployed then microservices can offer **horizontal scaling within seconds**
- It is the **high horizontal scaling** which leads the tech giants to move to microservices
- **Supports polyglotting**; if a microservice is e.g. CPU intensive, it can be implemented in CPU optimized programming language and other microservices can be implemented in other languages



# Development Velocity

- Microservices are often quite small in size
- Due to the size, adding new features in Microservices are usually faster



# Development Scaling

- Microservices are autonomous and can be developed independently
- Developers/teams can work on different microservices autonomously
- Companies can hire more developers to scale development
- Due to sizes, Microservices puts small Cognitive load on new hires
- Developers take lesser time to write first line of productive code



# Release Cycle

- Every microservice is **independently deployable**
- Resulting in the much **smaller release cycle**
- Using **CI/CD pipelines**, it is possible to give **several releases per day**



# Modularization

- Boundary between the microservices are **external Interfaces** aka Physical (Network) which is **hard to cross**
- Correctly crafted microservices often offers the “**Loosely coupled, highly cohesive**” modularization



# Modernization

- Microservices are **loosely coupled** and only **communicate via language-agnostic way** with each other
- A **microservice can easily be replaced** by a new one which can be developed using a new programming language
- Modernization in microservice architecture is **incremental** and not Big Bang

# Disadvantages of Microservices

# Disadvantages of Microservices

- As like anything in life, microservice architecture has also its price and a fair share of disadvantages
- It is by no means a Golden Hammer which can solve all sort of Problems in a Software Application
- There are scenarios in which moving to  $\mu$ service architecture from monolithic architecture without proper consideration will leads to nightmarish condition



# Design Complexity

- **Monolithic Architecture** often gives “One size fits for all” solution for Business applications
- But in **μservice architecture**, there are many solutions possible depending on the applications and use cases
- If the wrong solution is taken for wrong application size/type (e.g. put a kid's clothes on a full-grown man or vice versa), then **μservice architecture** is bound to fail



# Design Complexity

- Also, designing  $\mu$ services is challenging as there are far more moving parts compared to monoliths
- Usually, a badly designed  $\mu$ service is worse than a monolith

# Distributed Systems Complexity

- **Microservices are distributed system**; which are complex and has a unique set of challenges compared to single Machine systems
- Following problems can arise in Distributed Microservices:
  - Overall System latency is higher
  - Network failure or Individual Node failure can bring the whole system down
  - Operational complexities are higher

# Operational Complexity

- Once the Monolithic application is decomposed into **μservices**, the complexity moves from source code to operations
- Simple operations like **Logging**, **Monitoring** became more complex because instead of one Systems, many more need to be handled
- Sometimes existing Logging/Monitoring tools don't fit with **μservices** and new ones are needed
- Tracing is also very important in **μservices** to **measure the performance/latency** of individual **μservices** for a Service Request

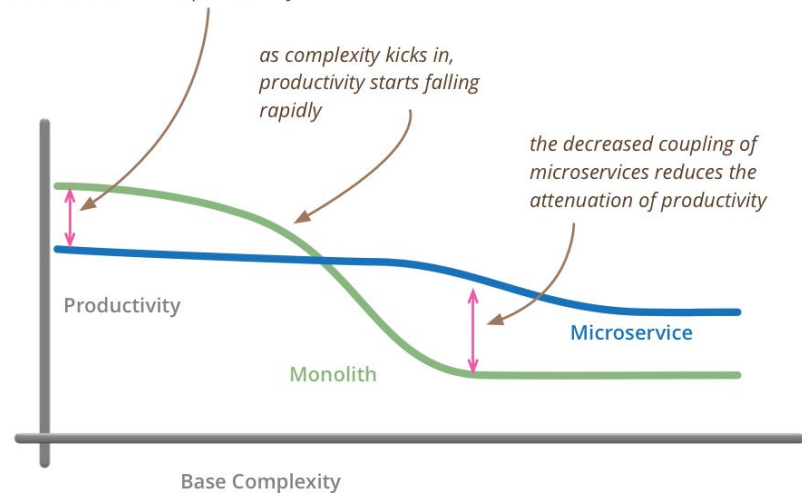




# Operational Complexity

- The **complete System test** is likewise quite complex in **μservices** compared to monolithic applications
- A renowned computer scientist **Martin Flower** states :  
" the initial Development Velocity of Microservice Architecture is lower compared to Monolithic Architecture due to the Operational Complexities "

for less-complex systems, the extra baggage required to manage microservices reduces productivity



but remember the skill of the team will outweigh any monolith/microservice choice



# Security

- Security in software systems is that elephant in the room what everybody can see but nobody wants to talk about
- Securing **one software application is hard**
- Securing hundreds of **μservices** which are often distributed systems is quite challenging

# Data Sharing and Data Consistency



- Ideally, every **μservices** should have its own data store
- Downside is that the **μservices** need to share data between themselves to fulfill the business goal
- **Data consistency** is another challenge
- To **support consistency** in the distributed databases is not recommended for two reasons:
  - It does not Scale and many Modern Data Store does not support it
  - Most of the modern NoSQL Databases only offers Eventual Consistency which needs careful design



# Communication Complexities

- Microservices achieves strict modularity and development autonomy via process/network boundaries
- Downside is that the services can only communicate via the physical network which eventually leads to higher network latency

# Conclusion

# Conclusion

- Designing and implementing **μservices** architecture is challenging compared to **monolithic software architecture**
- **Microservice architecture** is by no means a **silver bullet** which can solve the complexity issues of all sorts of applications
- Even after different arguments, it is believed that **μservices** architecture is a **very useful** and **handy** tool for modern software development

# Conclusion

- Specially for large Enterprises which normally develop complex softwares,  $\mu$ services architecture is the only way to tackle complexity and to be competitive in the market



# Resources

Link to article :

<https://towardsdatascience.com/microservice-architecture-a-brief-overview-and-why-you-should-use-it-in-your-next-project-a17b6e19adfd>

Feedback:

<https://forms.gle/PYnNbFfssdgWfCfX6>





# Summary