INTRODUCTION
TO
MICROSERVICE ARCHITECTURE

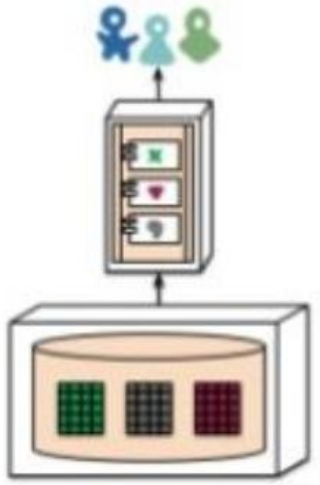
Outline

- Architectural Styles Overview
- Monolith Architecture
- Distributed Architecture
- Service Oriented Architecture (SOA)
- Microservice Architecture (MSA)
- Microservice Characteristics
- Principles of Microservices
- Benefits of Microservice Architecture
- Limitations of Microservice Architecture
- Microservice and API Ecosystem
- Microservice Reference Architecture

MONOLITHIC ARCHITECTURE

Monolithic Architecture

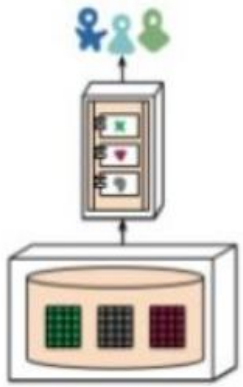
- Monolithic architecture is very similar to joint families. All components are present together in one box and all data stored in one database.



Monolithic Pros

1. Simplicity
2. Performance
3. Horizontal Scaling
4. Maintainability

Families have a well-defined structure, rules and shared responsibility which makes completing tasks easier.

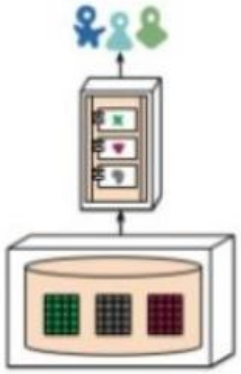


As it is normally owned by a single team, authority is centralized and technology is compatible it is easy to develop and build.

Monolithic Pros

1. Simplicity
2. **Performance**
3. Horizontal Scaling
4. Maintainability

Every member of a family knows his role and responsibility. Being together makes them strong to give better performance.

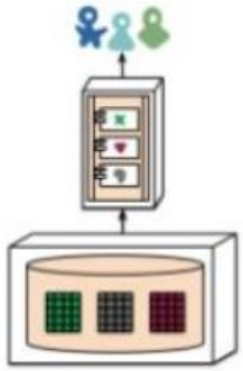


Close proximity and optimized design to the application needs gives best performance in the context.

Monolithic Pros

1. Simplicity
2. Performance
3. **Horizontal Scalable**
4. Maintainability

As a family grows, more rooms are created in the same space and new members are accommodated.

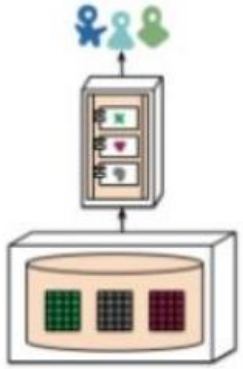


Applications can be easily scaled horizontally by adding new hardware and running new instances

Monolithic Pros

1. Simplicity
2. Performance
3. Horizontal Scaling
4. **Maintainability**

Every one contributes to the same pool and consumes what family produces.

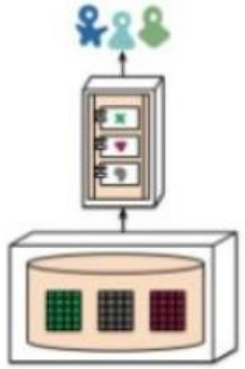


Maintaining a single application is simpler than multiple ones due to knowledge of it and control over it.

Monolithic Cons

1. Flexibility
2. Responsiveness
3. Availability
4. Heterogeneity

You need to get a lot of buy in to make changes in family with various stake holders and structure.

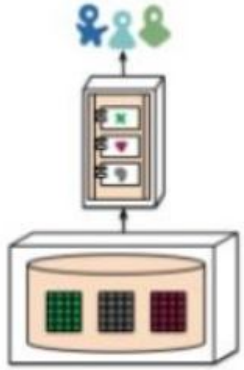


Due to the presence of tight coupling between components, making changes to one component may impact other components and thus would unintentionally change either its behaviour or break the code of other components.

Monolithic Cons

1. Flexibility
2. **Responsiveness**
3. Availability
4. Heterogeneity

When there are changing environments it takes time for family to adapt as members are used to living in a defined style

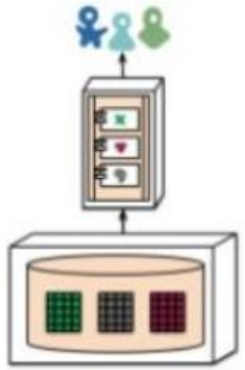


Due to less flexible architecture it is hard to respond to changing business needs and the time to market is longer.

Monolithic Cons

1. Flexibility
2. Responsiveness
3. **Availability**
4. Heterogeneity

If someone gets sick the entire family is impacted.

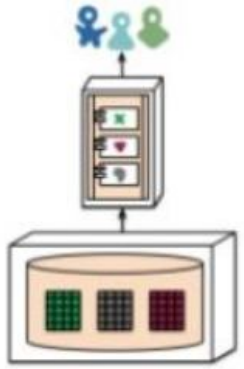


If one component goes down the whole solution is down. Any change can be made to the system only after shutting down all the components. Modular deployment is not possible.

Monolithic Cons

1. Flexibility
2. Responsiveness
3. Availability
4. **Heterogeneity**

Family follows same traditions, culture and clothing. If one diverts from it then the whole family is either stopped or member can't adjust with the rest of the family.



Most of the components are built using same or compatible technologies. Bringing a new technology or change is hard and mostly needs a rewrite.

DISTRIBUTED ARCHITECTURE

Distributed Architecture

Some of the challenges of joint family were addressed as people started building their own spaces, while living under the same roof. This allowed them freedom with in their own space.

However, members are still tightly coupled, driven by family rules and structure."

Distributed Architecture

- By identifying the components and their responsibilities, it becomes easier to recognize the changes to be made and localize changes to components. Components can be designed and developed independently.

- Distributed architecture brings these components with different responsibilities together to interact and achieve a common objective.

- However since they depend on other components for their functioning, tight coupling remains and the solution can be optimized to meet specific needs.

SOA

Shift to Service Oriented Architecture

From

- Build to last
- Function Oriented
- Prolonged development cycles

From

- Application silos
- Tightly coupled
- Structuring applications using components and objects
- Known implementation

To

- Process oriented
- Build to change
- Incrementally built & deployed

To

- Orchestrated solutions
- Loosely coupled
- Structure applications using services
- Implementation abstraction

Cohesion

Cohesion is a measure of the strength of the dependencies within a component

- Responsibilities have no meaningful relationship to each other
- Responsibilities which are related in time only
- Responsibilities of the same general category which are selected externally

- Responsibilities contribute to a few related business activities only
- Responsibilities are likely to use the same input or output data
- Responsibilities are likely to be used together

Coupling

Coupling is a measure of the strength of the dependencies between components

- Data is passed that is intended to control the internal component logic
- Components refer to the same global data area
- One component refers to the inside of another
- Different meanings are assigned to different parts of a range of data

- No unnecessary inter-component communication ("Don't Talk to Strangers")
- Communication is via fundamental data types
- If communication is via composite data then it is self-describing data (e.g. XML)

Isolation

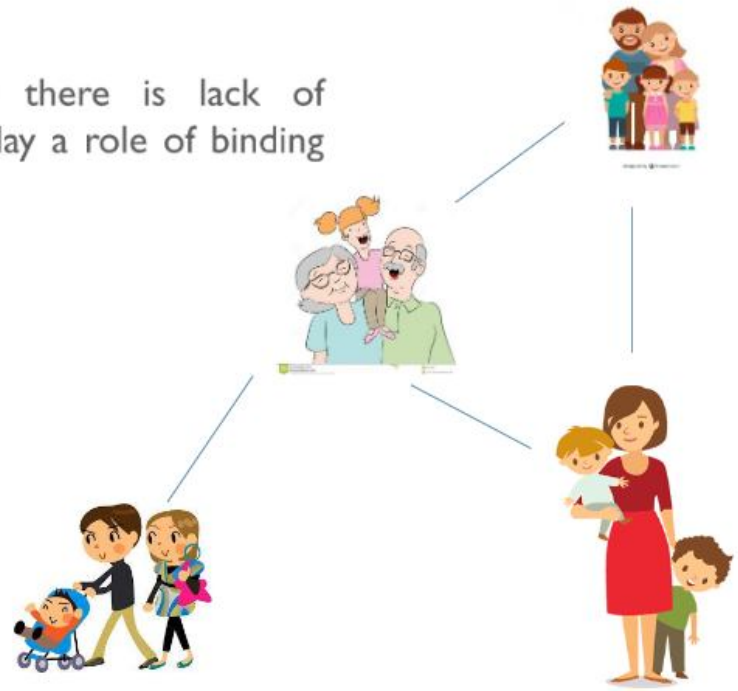
- Isolation is a measure of the degree to which product/technology dependencies are isolated
- Every component has a dependency on the product/technology specific aspects of other components
- No use of patterns

- Product/technology dependencies distributed between a relatively small, manageable number of components
- Some use of patterns

- Product/technology dependencies decoupled
- Use of patterns such as the GoF "Proxy", "Bridge", and "Mediator" or GRASP "Indirection"

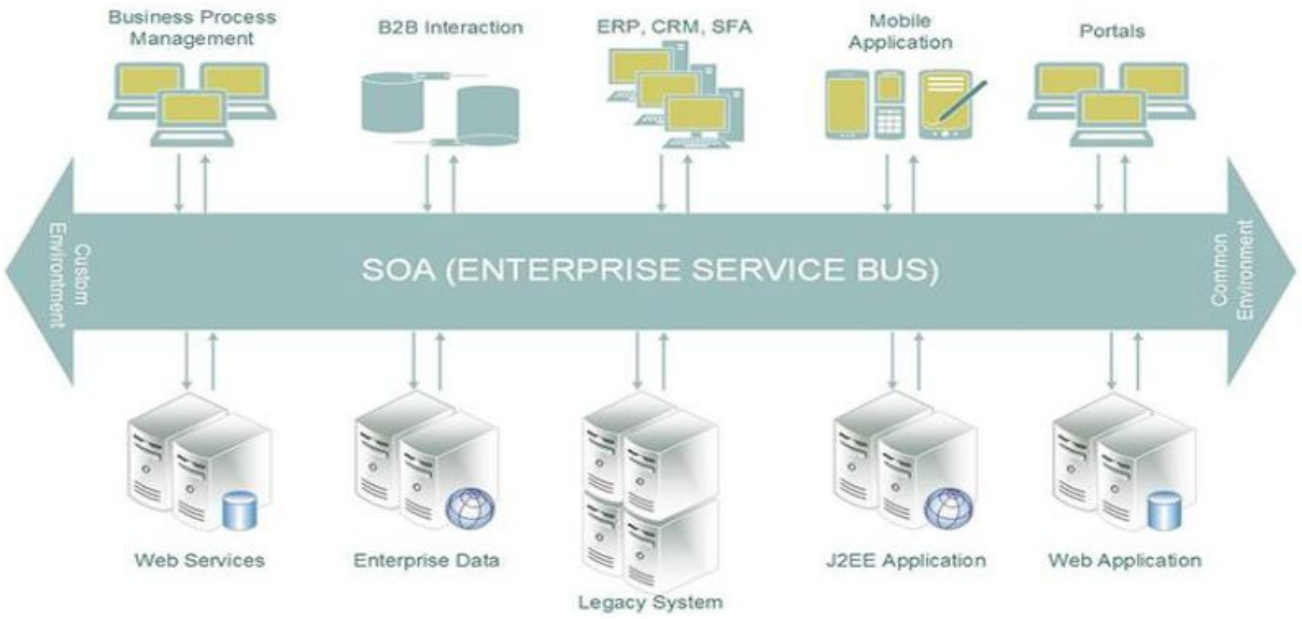
Smaller Families - SOA

- Small families give their members the freedom and flexibility of doing things their way and connecting with whom they like.
- Challenges arise when members are disconnected or there is lack of communication within the family. The elders in the family play a role of binding everybody together.



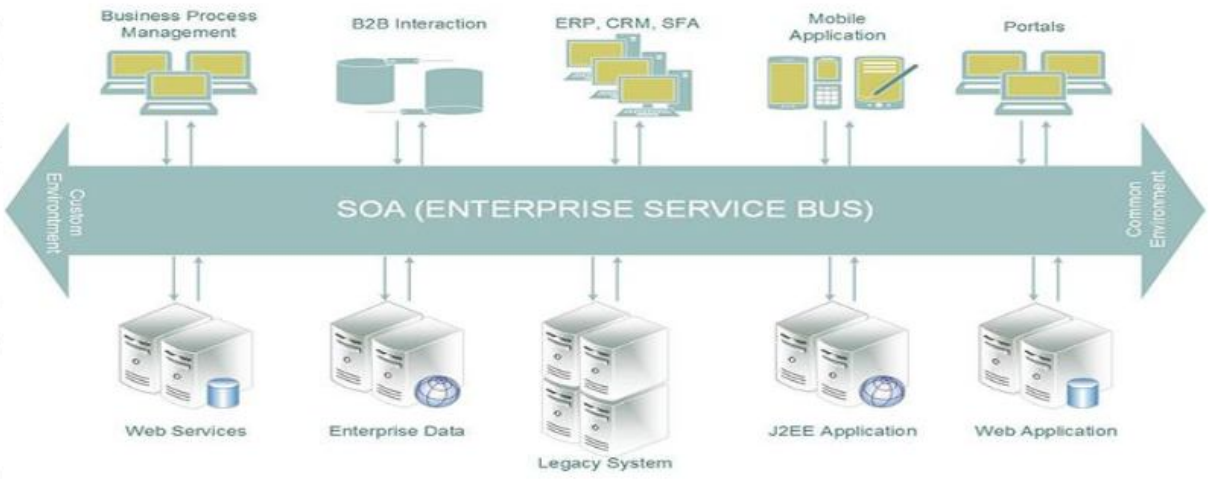
SOA - Advantages

- Provide Loose Coupling between components
- Heterogeneity of the technology is possible to integrate and adopt
- Changes are localized in services
- Reuse of services across organization is promoted



SOA - Challenges

- SOA started to focus too much on technology and less on business.
- ESB started to get more complex, though it was intended to have routing and transformation logic, business logic too was soon being embedded.
- Governance of the services across the organization was challenging due to different ownership and grouping.
- Even though services are separate underlying infrastructure and data source failure could bring down multiple services.



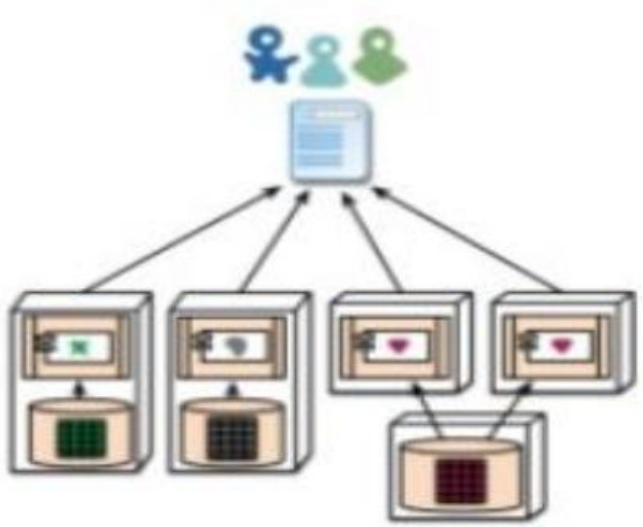
MICROSERVICES ARCHITECTURE

What are Microservices?

- Microservices have emerged from concepts like domain-driven design, Continuous delivery, On-demand virtualization, Infrastructure automation and Small autonomous teams.

"Microservices are small, autonomous services that work together."

- Embracing fine-grained, microservice architectures, they can deliver software faster and embrace newer & heterogeneous technologies.
- Microservices make organization agile and allowing to respond faster to the inevitable changes that are constant.



What are Microservices?

In short, the microservice architectural style is an approach to developing a single application as a **suite of small services**, each **running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery. There is a **bare minimum of centralized management** of these services, which may be written in different programming languages and use different data storage technologies.

-- James Lewis and Martin Fowler

Microservices - also known as the 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
- 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.

-- Chris Richardson

Microservices

- Microservices are driven from business domains and needs. It breaks the domain functions in smaller independent components. Technology helps in achieving these components to be developed , deployed and function independently.

- Since they are small and independent this makes them easy to manage and changes can be done quickly.

- The Learning curve for such smaller components is very short, new developers can be on boarded and start to contribute.

Microservices - Characteristics

1. Componentization via Services
2. Organized around Business Capabilities
3. Products not Projects
4. Smart Endpoints and Dumb Pipes
5. Decentralized Governance
6. Decentralized Data Management
7. Infrastructure Automation
8. Design for failure
9. Evolutionary Design

Organized around Business Capabilities

- The microservices approach to division is different. Services are split up and organized around business capability.
- Such services take a broad-stack implementation of software for that business area which includes:
  - User-interface
  - Persistent storage, and
  - Any external collaborations.

Products not Projects

- A team should own a product over its full lifetime. Inspired by Amazon's notion of "you build, you run it" where a 2 pizza team takes full responsibility for the software in production.
- Developers get a taste of how their software behaves in production in day to day
- Developers also have to take on some of the support burden.
- This also enables independent and easy distribution of work among developers.

Smart Endpoints and Dumb Pipes

- The microservice principle of smart endpoints and dumb pipes is easy to understand when you embrace the concept of decentralization in architecture and logic.
- Despite using "dumb pipes", microservices can still implement essential messaging primitives eliminating the need for a centralized service bus.
- Microservices makes use of the broad ecosystem of frameworks that exist as dumb pipes for both request-response and observer communication.
- Applications built from microservices aim to be as decoupled and as cohesive as possible - they own their own domain logic. This is a big difference from tradition SOA where communication channel were doing more responsibilities.

Decentralized Governance

- Need of decentralized governance evident with Microservice architecture and organization.
- Biggest preposition of decentralized governance is the build it/run it concept. Teams are responsible for all aspects of the software they build including operating the software
- Who would like to wake up at 3am every night due to a client issue ?...This is certainly a powerful incentive to focus on quality when writing your code.
- This also helps in building governance around microservice ecosystem by introducing checks in automated pipelines.

Decentralized Data Management

- Monolithic applications prefer a single logical database.
- Enterprises often prefer a single database across a range of applications.
- Microservices prefer letting each service manage its own database.
- With data distribution comes challenges of distributed transactions.
- As each service lives in its own space, transaction boundaries span across multiple services and requires a compensating transactions in case of failures.

Infrastructure Automation

- Microservices are complete and independently deployable. However, the number of microservices and their instances have grown significantly and with cloud enabled infrastructure it is of even more importance to have the build and deploy process automated.

- To ensure that the software is functioning as intended, **automated tests are conducted**. Promotion of working software 'up' the pipeline means we **automate deployment** to each new environment.

- DevOps becomes the integral part of any microservice architecture as DevOps automated pipelines enables microservices changes to roll up and down as required

Design for Failure

- Applications should be designed so that they can tolerate the failure of services. Any service call could fail due to the unavailability of the service producer. The consumer has to respond to this as gracefully as possible.

- Microservice applications put a lot of emphasis on real-time monitoring of the application, checking both architectural elements and business relevant metrics, this enables the eco system to get a early warning signs and take the appropriate actions, where possible self healing pattern is put in place.

Principle of Microservices

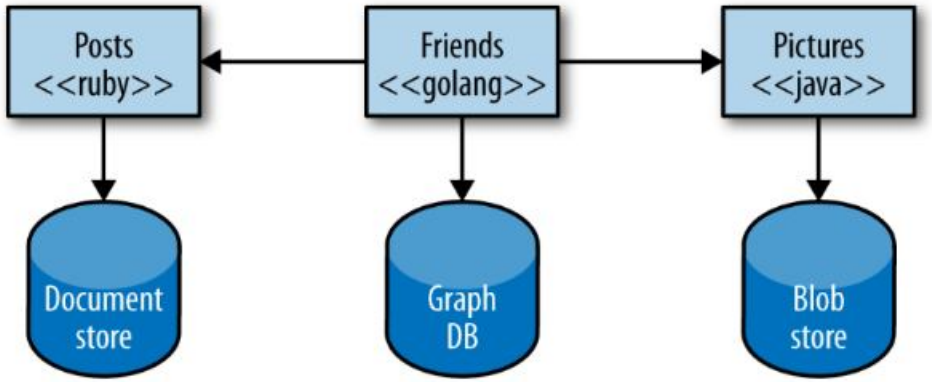
1. Model around Your Business Domain
2. Build a Culture of Automation
3. Hide Implementation details
4. Embrace Decentralization
5. Deploy Independently
6. Focus on Consumers First
7. Isolate Failure
8. Make them Highly Observable

Microservices - Benefits

- Ease of build and understand
- Heterogeneity of Technology
- Resilient
- Robust and Scalable
- Smaller and Faster Deployment

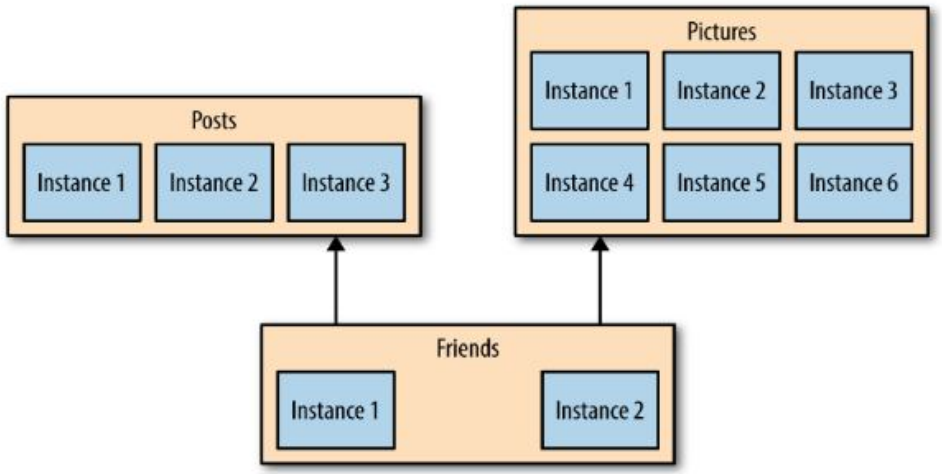
Heterogeneity of Technology

- With a system composed of multiple, collaborating services, different technologies can be chosen inside each one depending on the best fitment.
- This enables organizations to pick the right tool for each job instead of having to select a more standardized, one-size-fits-all approach.



Resilience

- The problem of one component failure which can be isolated and the rest of the system can carry on working. This is a key concept in resilience engineering.
- With smaller services, we can only scale services that need scaling.
  - This way other parts of the system can still run on less powerful hardware.
- When embracing on-demand provisioning systems like those provided by Amazon Web Services, Azure, we can apply this scaling on demand for the component that need it.



Ease of Deployment

- With microservices, we can:
  - Make a change to a single service and deploy it independently.
  - This allows to get code deployed faster.

- Fast rollback:
  - If a problem occurs, can be isolated quickly to an individual service.
  - This enables us to get new functionality out to customers faster.

- Ability to deploy/un-deploy independent of other microservices.
- Must be able to scale at each microservices level.
- Building and deploying microservices quickly.
- Failure in one microservice does not affect any of the other services.

Organizational Alignment

- Many of us have experienced the problems associated with large teams and large codebases. These problems can be exacerbated when the team is distributed. Also given the fact, smaller teams working on smaller codebases are generally tend to be more productive.

- Microservices allow us to better align enterprise architecture to the organization business, helping it to minimize the number of people working on any one codebase.

- We can also shift ownership of services between teams to try to keep people working on one service collocated.

Microservices - Limitations

- Distributed System Complexity
- More services equals More resources
- Significant Operations Overhead
- End to End Testing is challenging
- Debugging and Troubleshooting is challenging
- More tools to maintain Microservice Infrastructure

Popular Microservice Architecture Adopters

SOA VS MICROSERVICE

SOA vs Microservice

SOA is like an orchestra where each artist is performing with his/her instrument while the music director guides them all.

With Microservices each dancer is independent and know what they need to do. If they miss some steps they know how to get back on the sequence.

SOA vs Microservice

SOA	MSA
More importance on business functionality reuse	More importance on the concept of "bounded context"
Common governance and standards	Relaxed governance, with more focus on people collaboration and freedom of choice
Uses enterprise service bus (ESB) for communication	Uses less elaborate and simple messaging system
Supports multiple message protocols	Uses lightweight protocols such as HTTP/REST & AMQP
Multi-threaded with more overheads to handle I/O	Single-threaded usually with use of Event Loop features for non-locking I/O handling
Not designed to scale on cloud and container based environments	Containers work very well in MSA
Maximizes application service reusability	More focused on decoupling
Uses existing data structures and does not guide on data distributions	Emphasis on data being together with services .
A systematic change requires modifying the monolith	A systematic change is to create a new service
DevOps / Continuous Delivery is becoming popular, but not yet mainstream	Strong focus on DevOps / Continuous Delivery

MICROSERVICE AND API ECOSYSTEM

The Role of APIs in Microservices

- A microservice architecture function is based on concept that the infrastructure's components must be able to interact. Microservices composition builds a the business functionality and in order to achieve that they should be able to work with any of the other services and websites are required in order to achieve it.

- Each microservice must have an interface, that's why the Web API is a vital enabler of microservices. Being based on the open networking principles of the Web, RESTful APIs provide the most logical model for building interfaces between the various components of a microservice architecture.

Benefits of APIs Ecosystem

Centralized Visibility :

- Have a centralized view of all your APIs and makes discovery by their consumers easier promoting reuse.

Streamlined API Connections:

- Easy to publish and maintain end point of services.

Documentation Management :

- Your published APIs need an edge to attract developer attention and one way to accomplish this goal comes from well-written documentation.

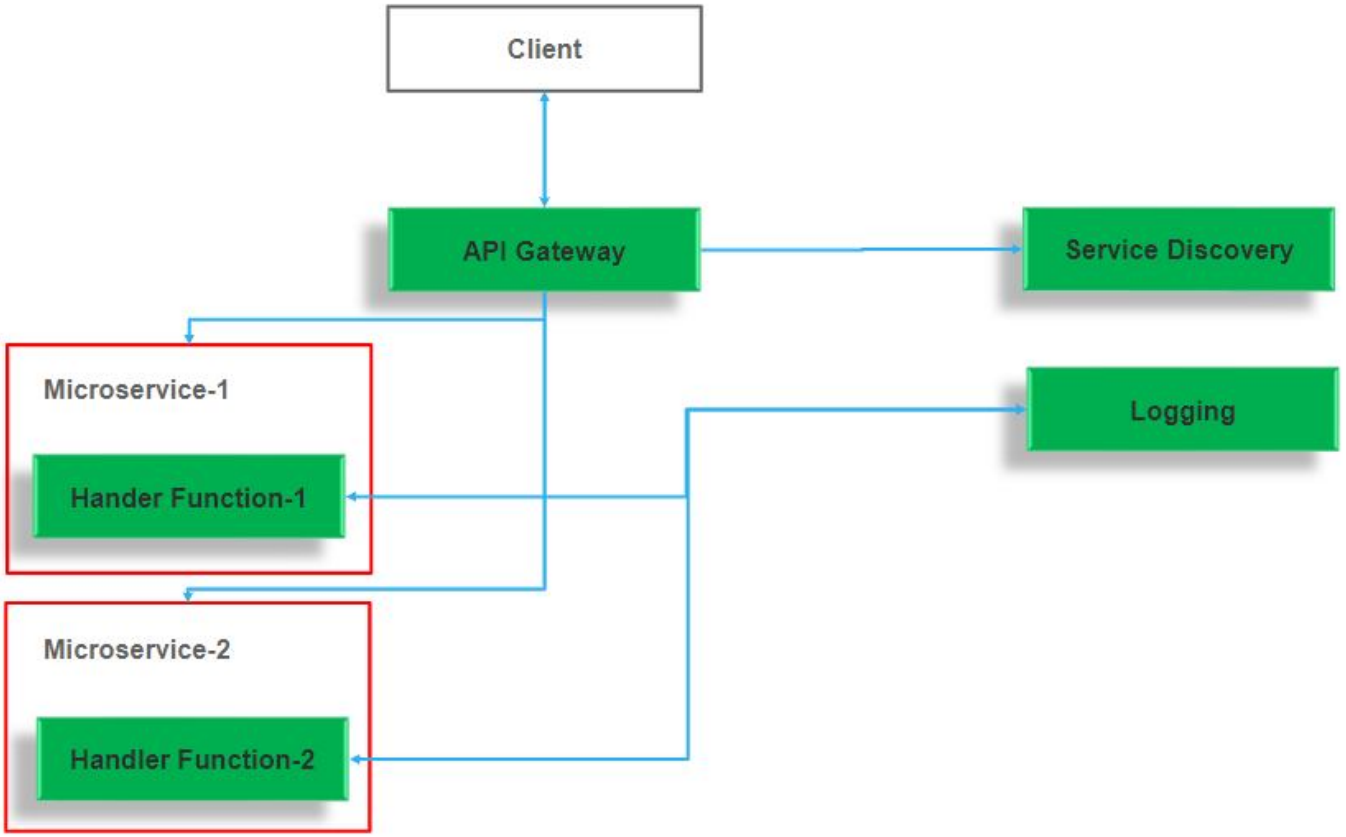
Performance Tweaks :

- You can optimize APIs in several ways, for example by testing the best API for a particular application, by allocating additional resources to mission-critical connections, or by eliminating redundancies within organization.

Analytics:

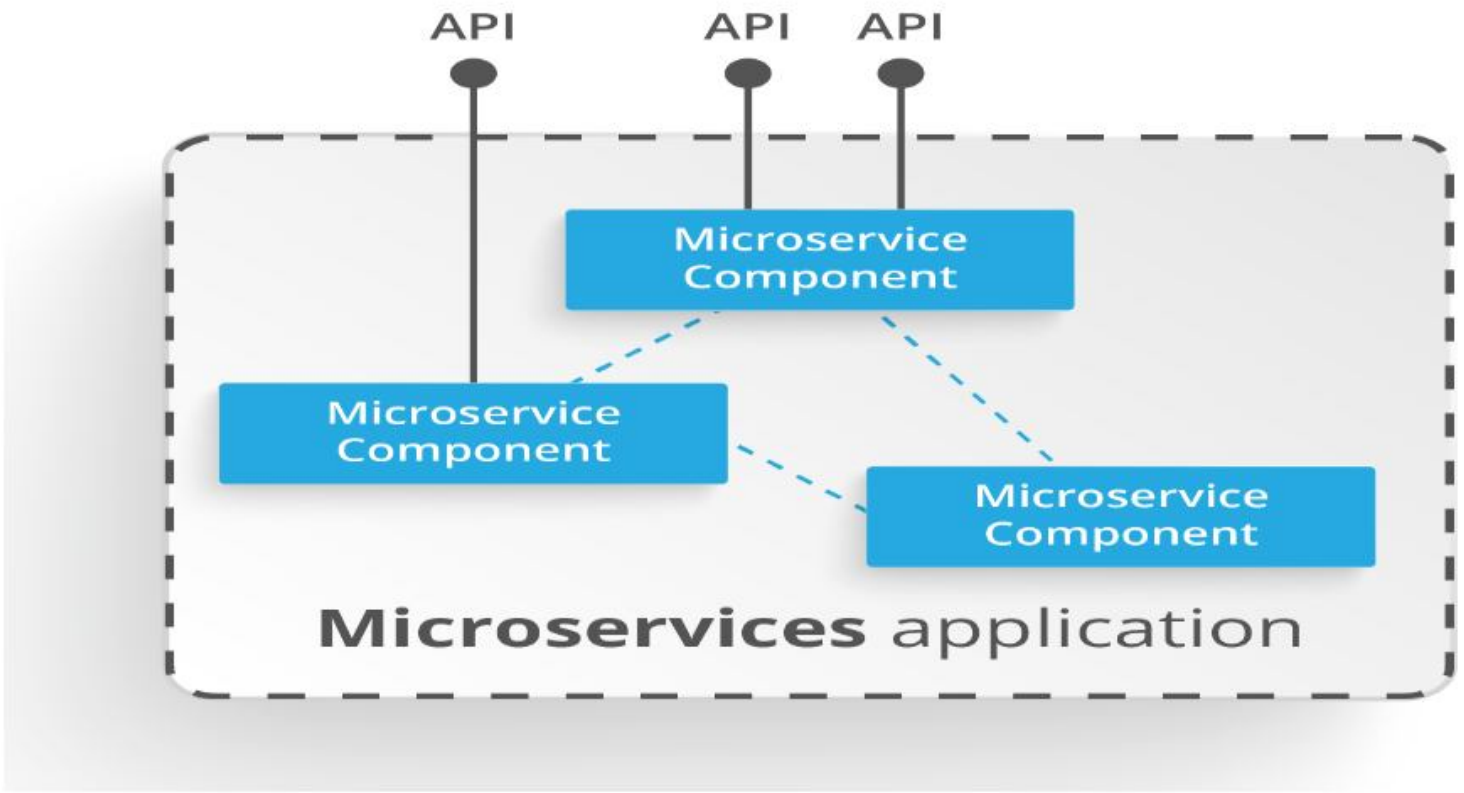
- APIs are tracked and monitored collecting the data about their uses and performance, generating right set of data to analyze their users and optimize the benefits.

Microservice Architecture – API Ecosystem



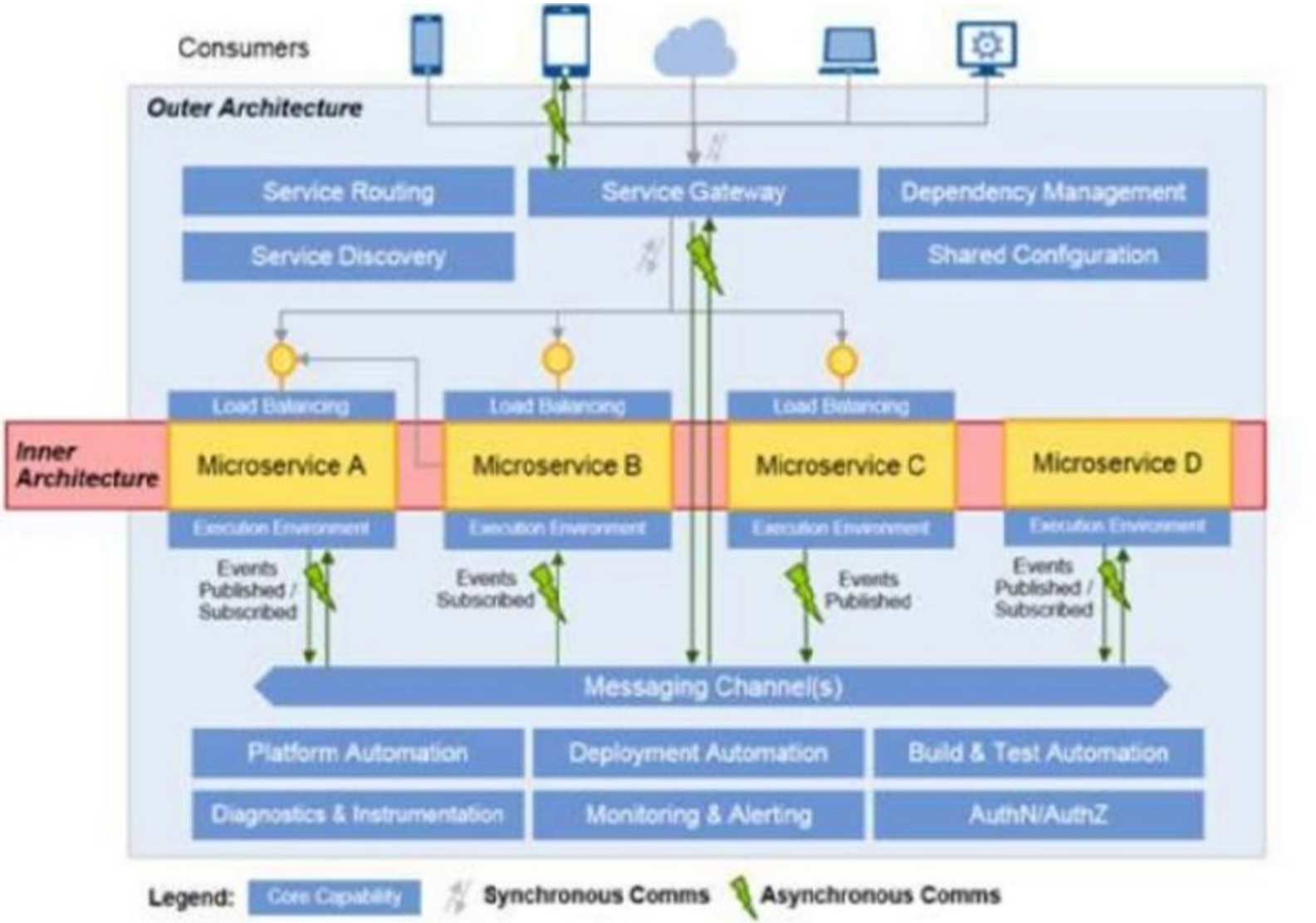
Microservice & API

Microservice and APIs are related and compliment each other , however they are not same .



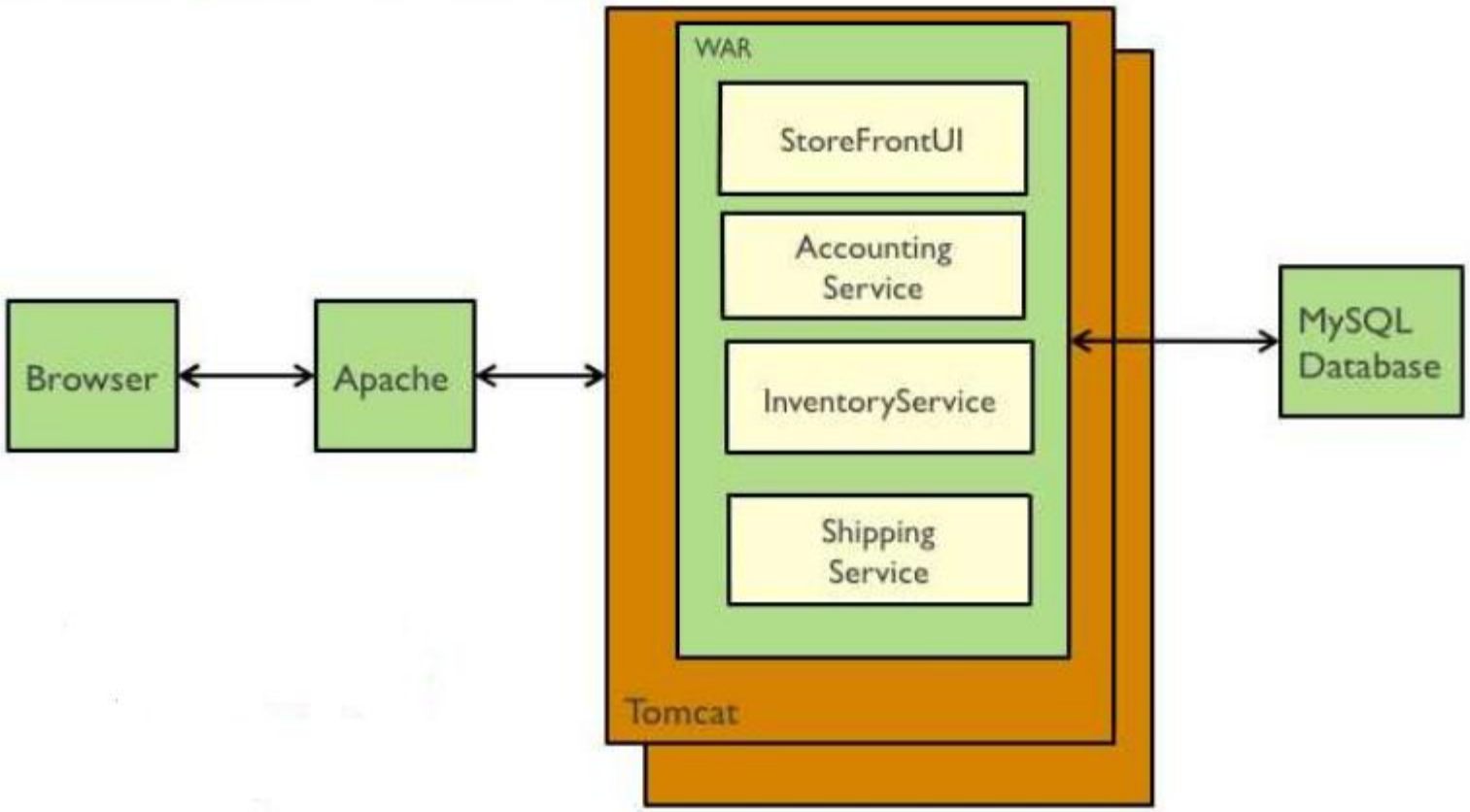
MICROSERVICE REFERENCE ARCHITECTURE

Microservice – Reference Architecture

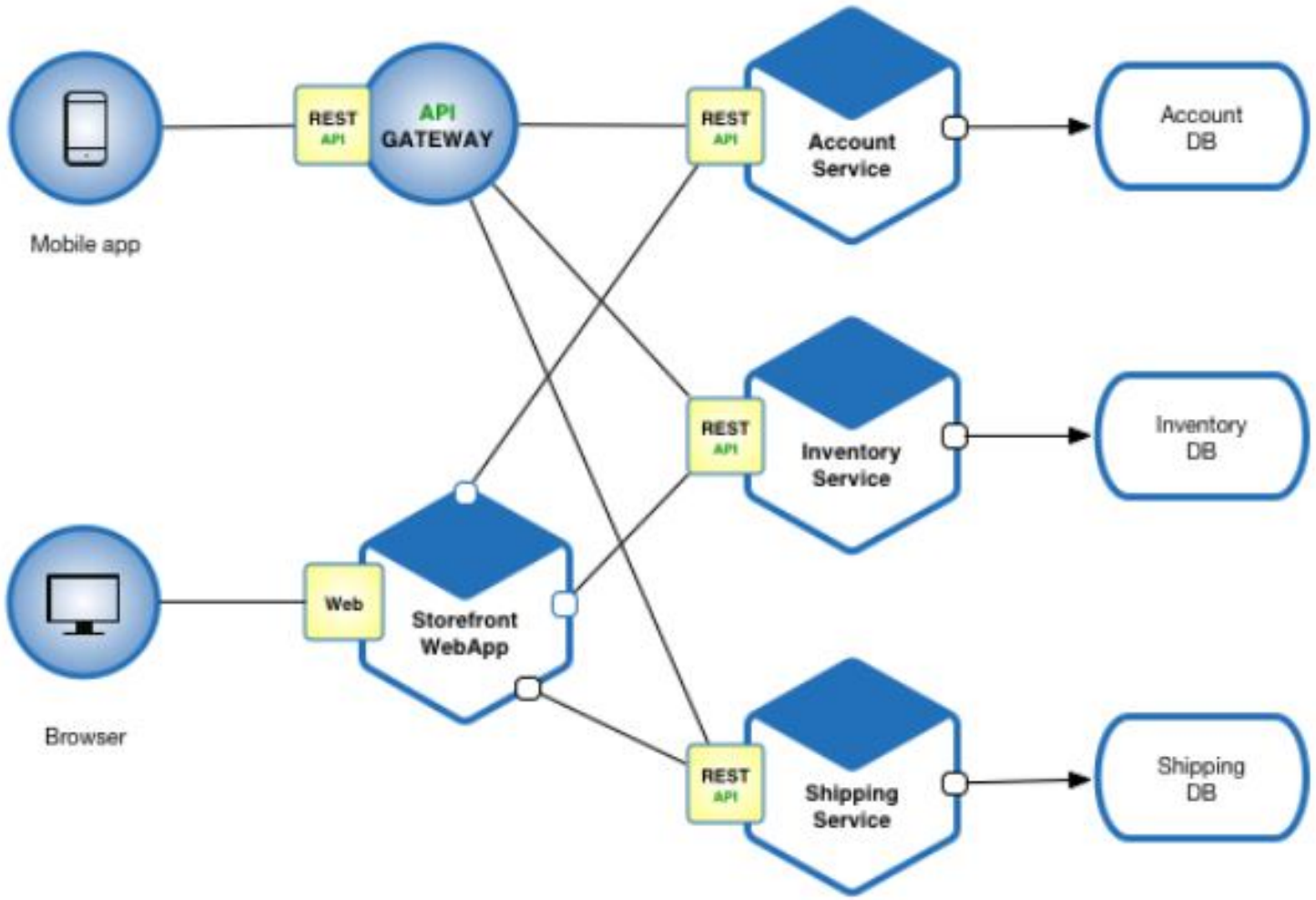


AN EXAMPLE

Monolithic Architecture - Example



Microservice Architecture - Example



MIRCOSERVICE PATTERNS

Microservice Patterns

- http://microservices.io/patterns

Thank You!