



1 : Microservices

EN6106 – Emerging Topics in Information
Technology

Level III - Semester 6

'Monolithic' Architecture

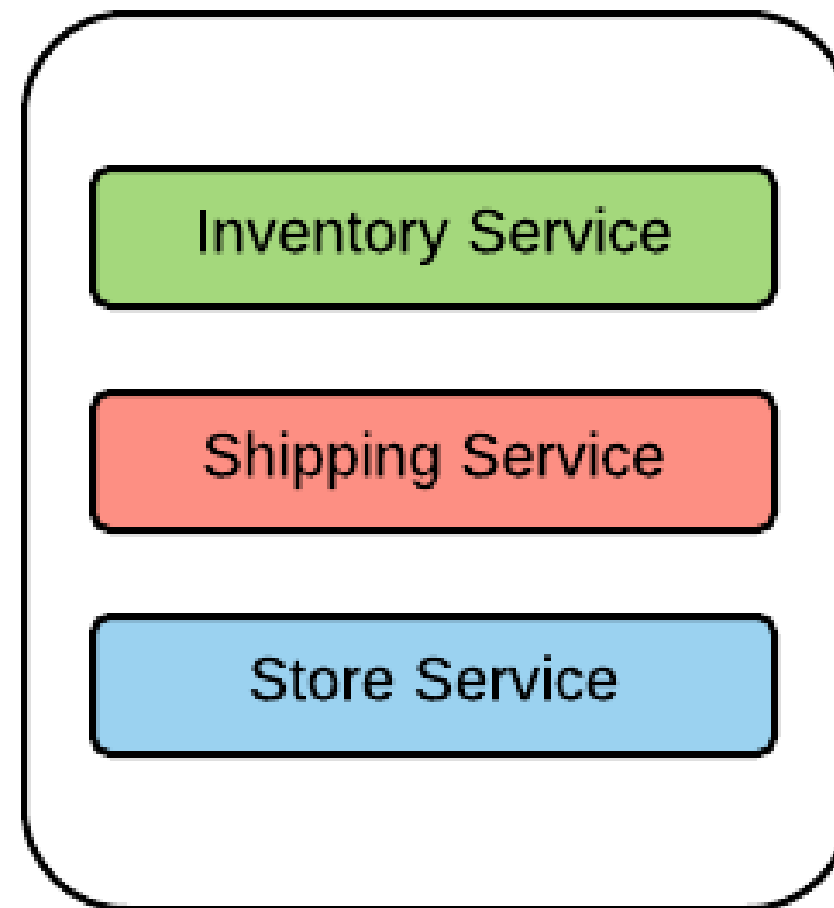


Monolithic Architecture

- All functionalities are implemented/deployed into a single software application.
- Enterprise software applications - ERPs, CRMs etc.
- SOA/web services: 'coarse-grained' services, broad scope, mammoth services with several dozens of operations and complex message formats

Monolithic Architecture

- **Use case** : Online *Retail software application with which comprises of multiple business functionalities.*



Monolithic Architecture

- **Developed** and **deployed** as a **single unit**.
- Overwhelmingly **complex**; which leads to nightmares in maintaining, upgrading and adding new features.
- **Redeploy the entire application**, in order to update a part of it.
- Scaling : **scaled as a single application** and difficult to scale with conflicting resource requirements
- **Reliability** - One unstable service can bring the whole application down.
- **Hard to innovate**, practice agile development and delivery methodologies

'Microservices' Architecture

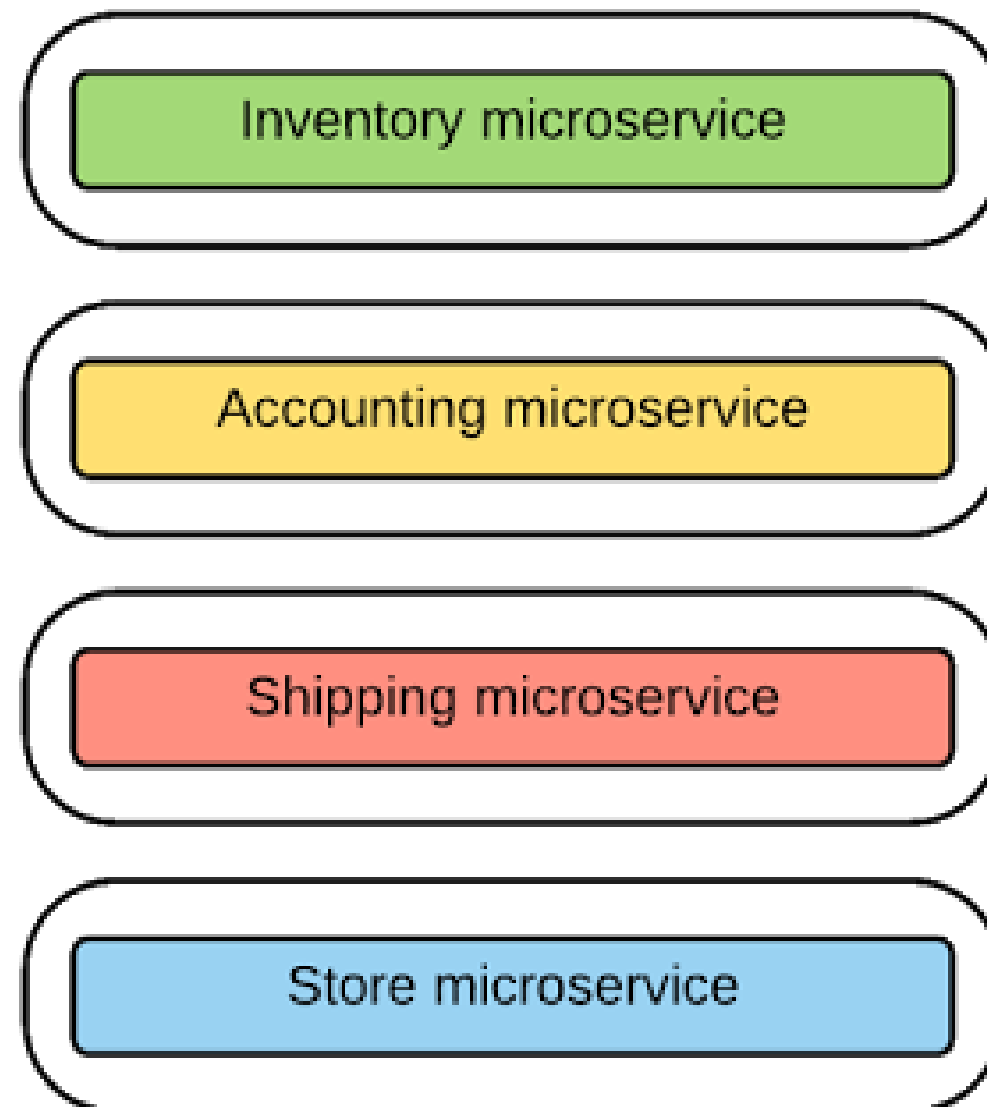


Microservices Architecture

- *The foundation of microservices architecture(MSA) is about developing a single application as a **suite of fine-grained and independent services** that are running in its own process, **developed and deployed independently***
- Its just more than segregating the services in a monolith.

Microservices Architecture

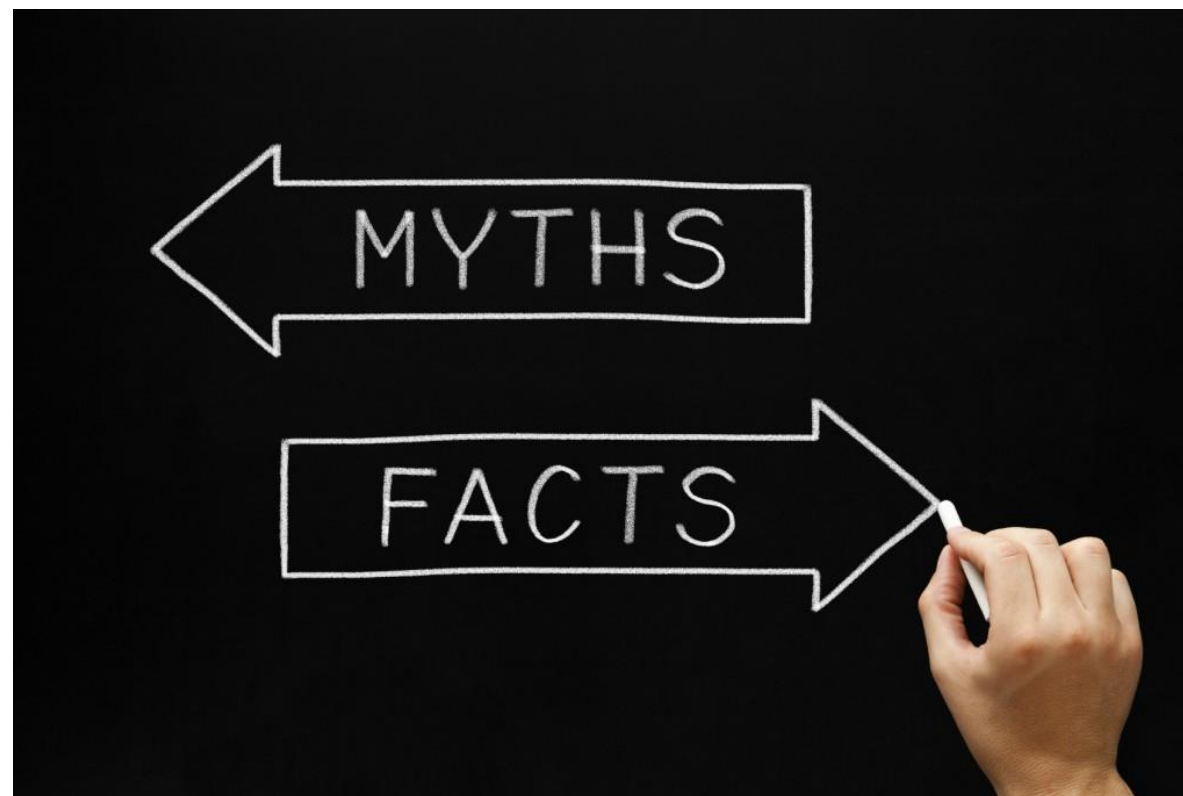
- **Use case** : Online retail application can be implemented with a suite of microservices



Designing Microservices : Size, scope and capabilities

- **Common Misconceptions**

- Lines of Code
- Team size
- 'Micro' is a bit misleading term
- Use web services and rebranding them as microservices



Designing Microservices : Size, scope and capabilities

- **Single Responsibility Principle**(SRP): Having a limited and a focused business scope.
- Find the **service boundaries** and align them with the **business capabilities** .
- Make sure the microservices design ensures the **agile/independent development and deployment** of the service.
- Focus on **scope** of the microservice, but not about making the the service smaller- **righted sized services**.
- Unlike service in web services, a given microservice should have a **very few operations/functionalities and simple message format**.
- Start with relatively **broad service boundaries to begin with**, refactoring to smaller ones (based on business requirements) as time goes on.

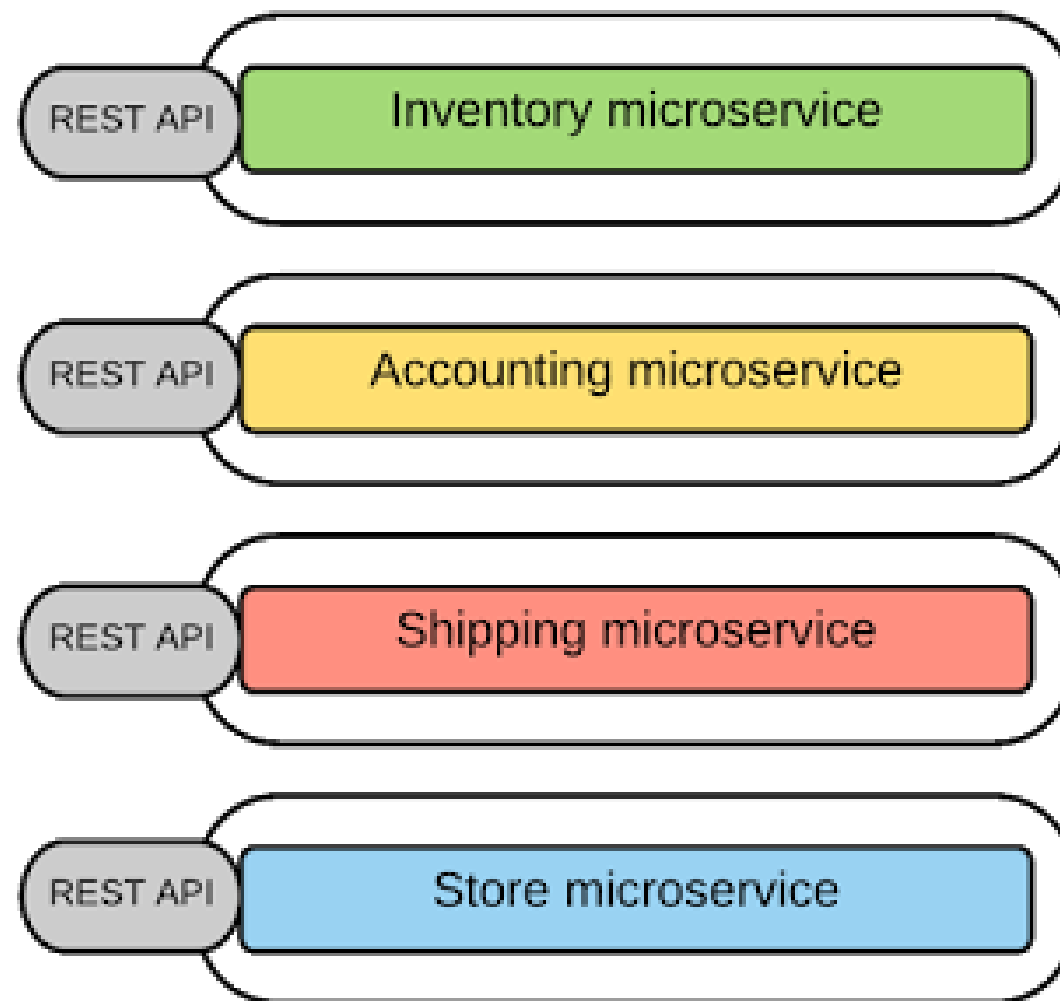
Messaging in Microservices

- In **Monolithic architecture**:
 - Function calls or language-level method calls
 - SOA/web services : SOAP and WS* with HTTP, JMS etc.
 - Webservices with several dozens of operations and complex message schemas
- In **Microservice architecture**:
 - Simple and lightweight messaging mechanism.

Messaging in Microservices

- **Synchronous Messaging**

- *Client expects a timely response from the service and waits till it get it.*
- Communication protocols: REST, Thrift



Messaging in Microservices

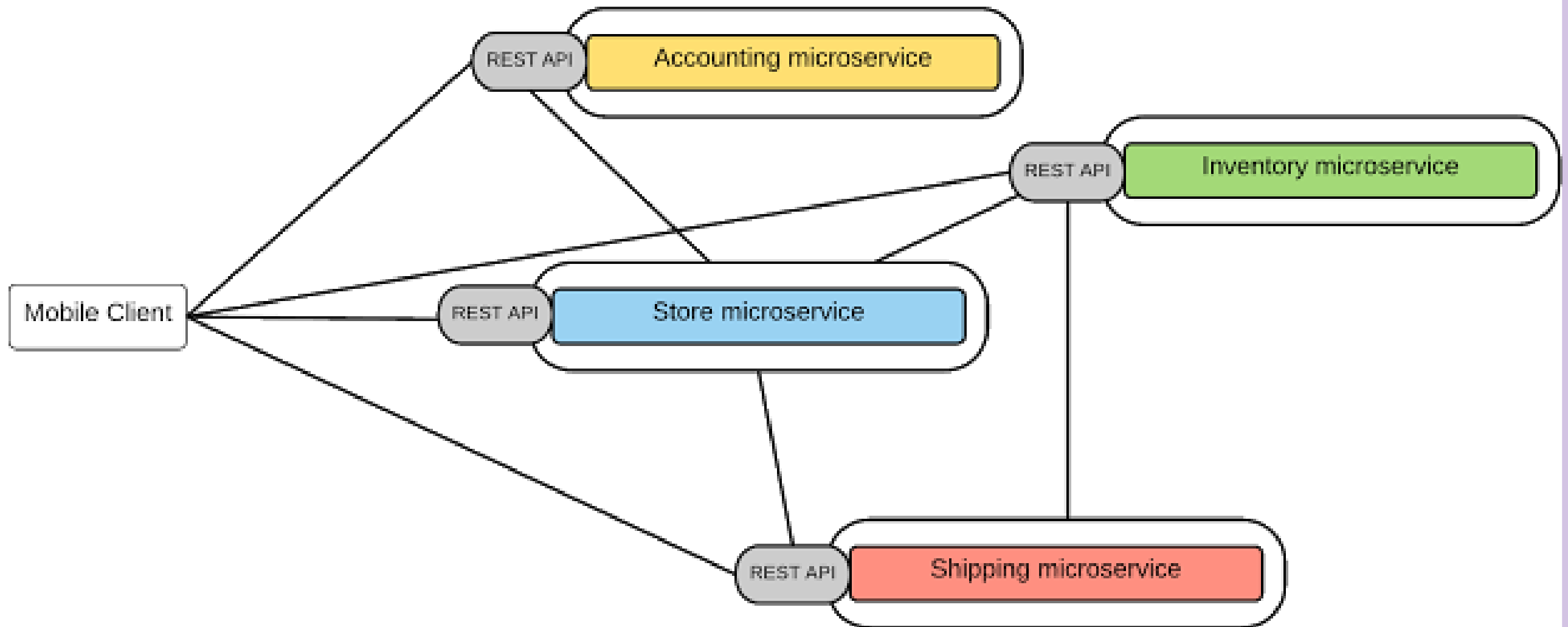
- **Asynchronous Messaging**
 - Client doesn't expect a response immediately, or not accepts a response at all
 - AMQP, STOMP, MQTT
- **Message Formats**
 - JSON, XML, Thrift, ProtoBuf, Avro
- **Service Contracts**
 - Defining the service interfaces - Swagger, RAML, Thrift IDL

Integrating Microservices (Inter-service Communication)

- Required to have the communication structures between different microservices.
- SOA/web services used ESB.
- Microservices promotes to eliminate the central message bus/ESB and move the 'smart-ness' or business logic to the services and client(known as 'Smart Endpoints').
- Connect services through 'dumb' pipes.

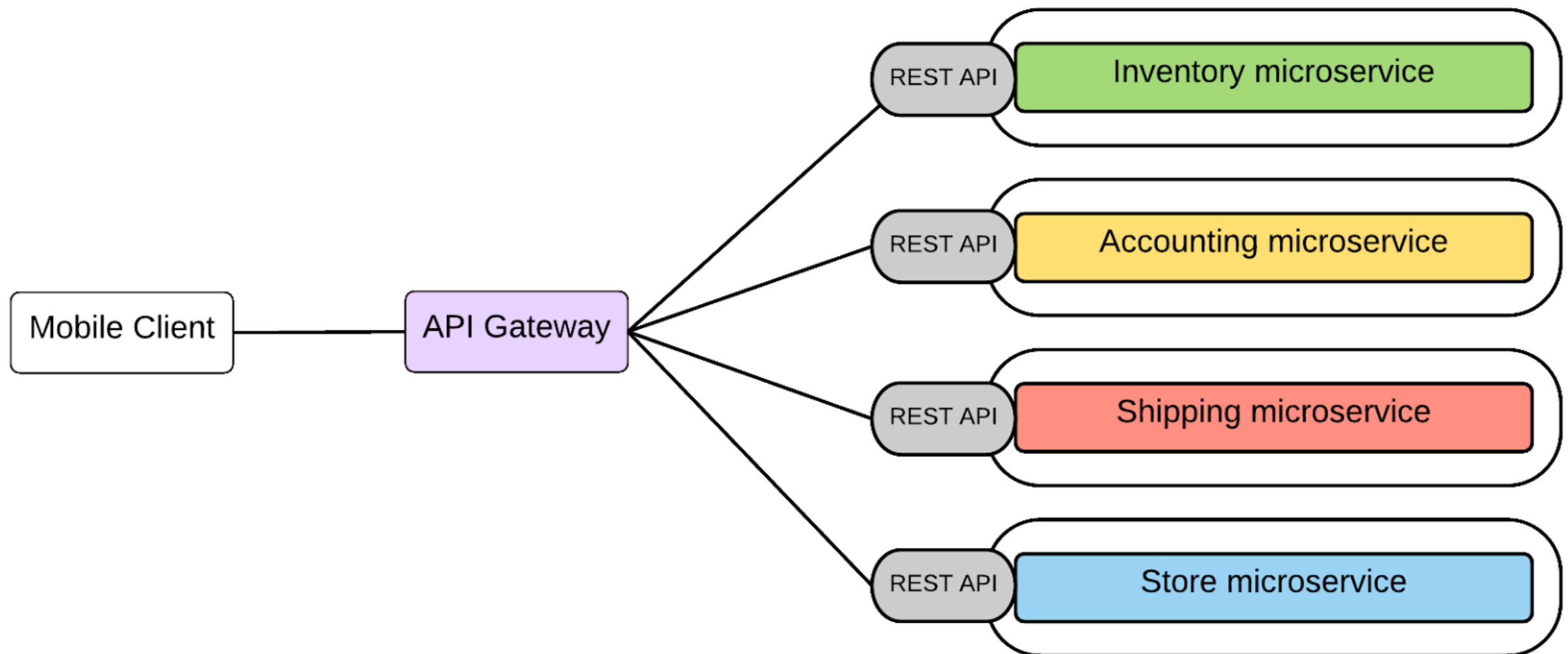
Integrating Microservices (Inter-service Communication)

- **Point-to-point style - Invoking services directly**



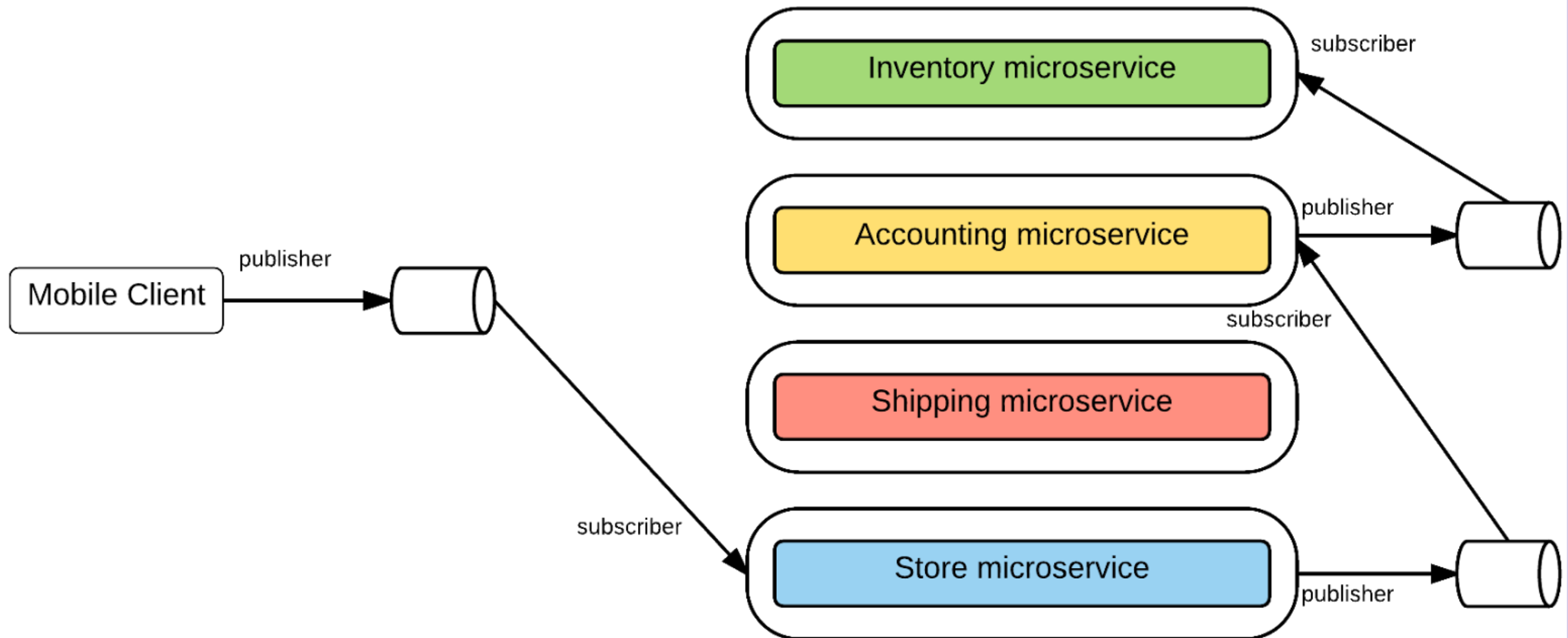
Integrating Microservices (Inter-service Communication)

- **API-Gateway style**



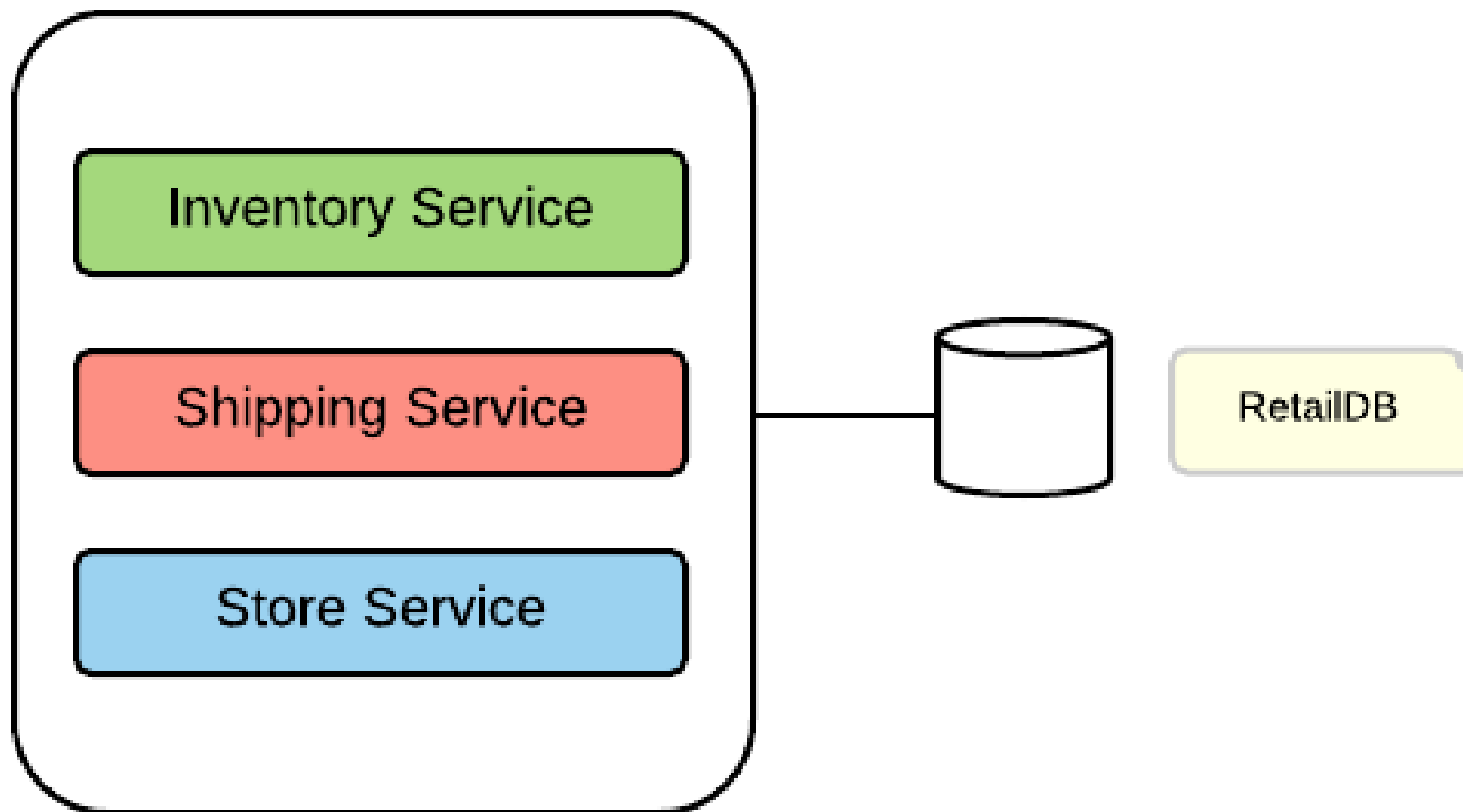
Integrating Microservices (Inter-service Communication)

- **Message Broker style**



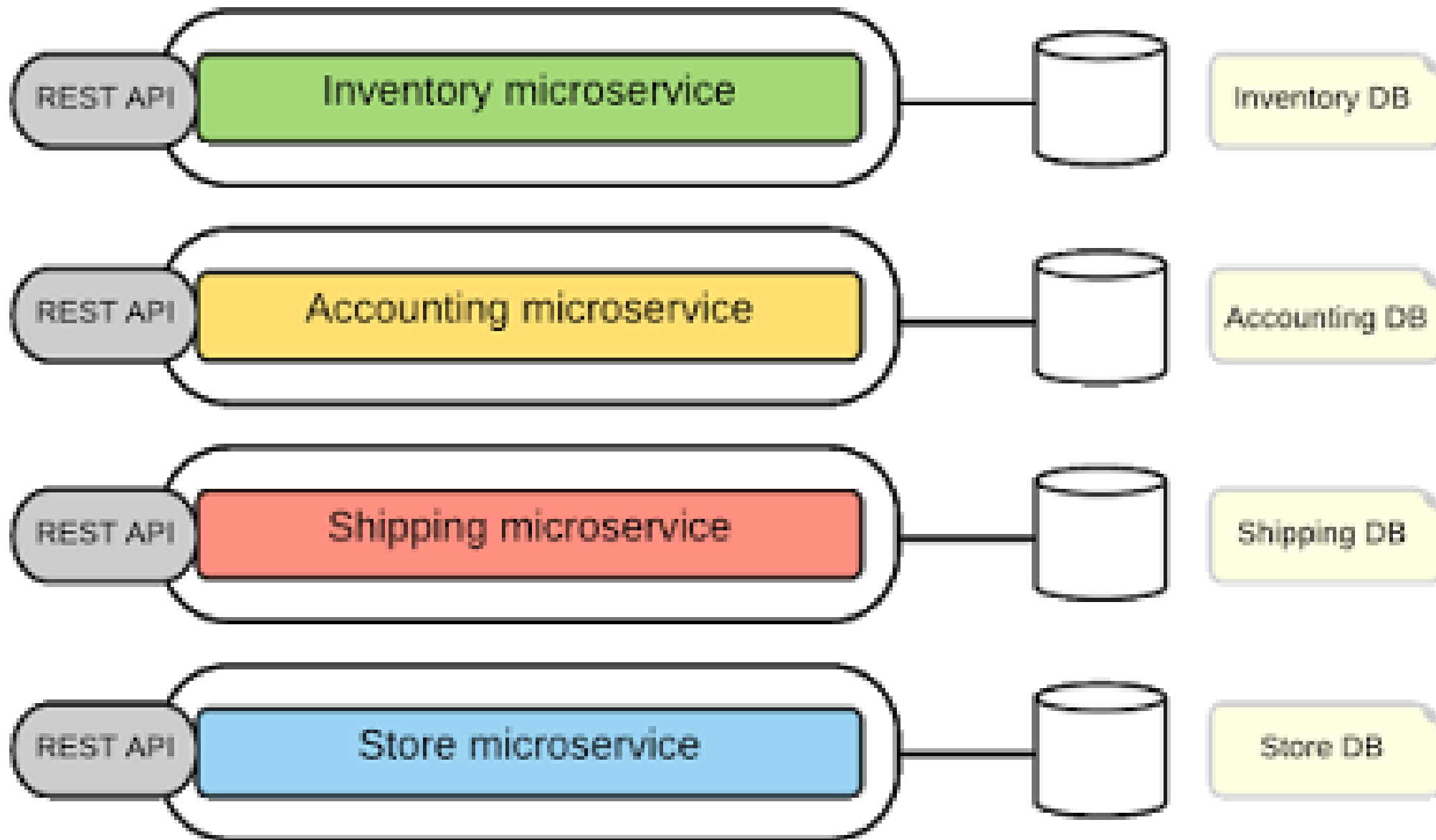
Data Management

- **Monolithic applications use a centralized database**



Data Management

- Decentralized Data management with Microservices



Data Management

- **Decentralized Data management with Microservices**
 - Each microservice can have a **private database** to persist the data that requires to implement the business functionality offered from it.
 - A given microservice can **only access the dedicated private database** but not the databases of other microservices.
 - In some business scenarios, you might have to update several database for a single transaction. In such scenarios, **the databases of other microservices** should be updated **through its service API only**.
 - May make it particularly challenging to **migrate from a monolithic system (with one database schema) to a microservices-based system**.

Decentralized Governance

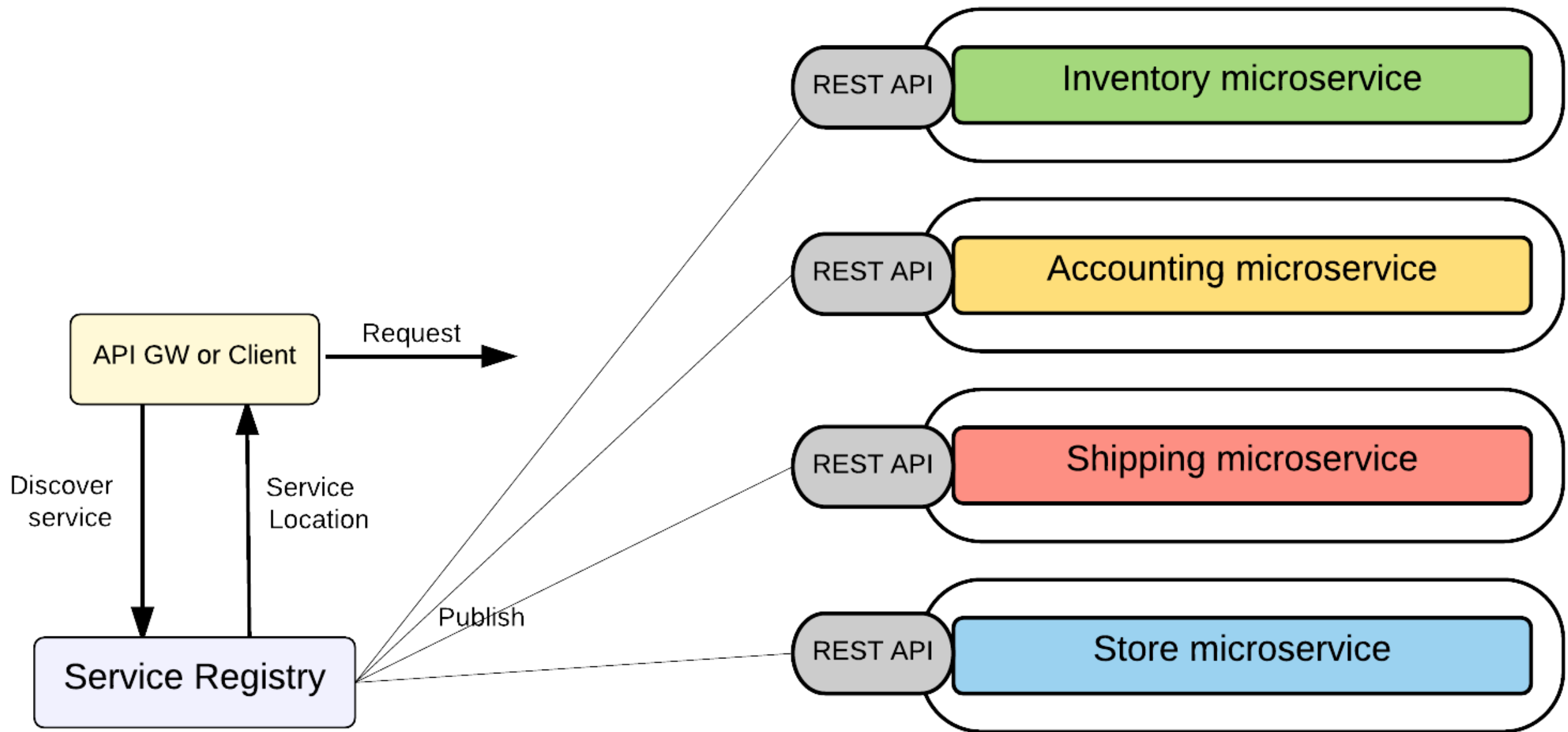
- **Governance** - establishing and enforcing how people and solutions work together to achieve organizational objectives
 - Design and runtime governance.
- In Microservices Architecture:
 - **No centralized design-time governance.**
 - Make their own decisions about its design and implementation.
 - Foster the sharing of common/reusable services.
 - **Run-time governance** aspects such as SLAs, throttling, monitoring, common security requirements and service discovery may be **implemented at API-GW** level.

Service Registry and Service Discovery

- ***Service Registry*** - Holds the microservices instances and their locations
- ***Service Discovery*** - find the available microservices and their location

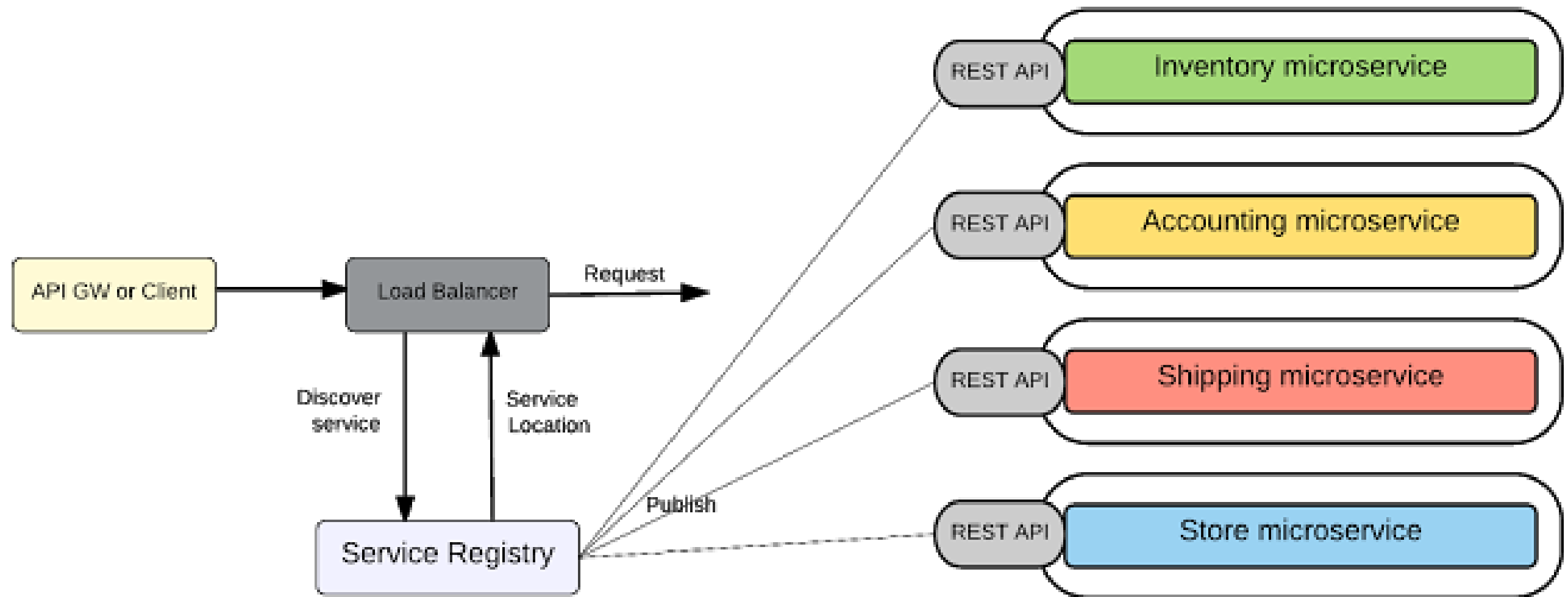
Service Discovery

- Client-side Discovery***



Service Discovery

- Server-side Discovery*

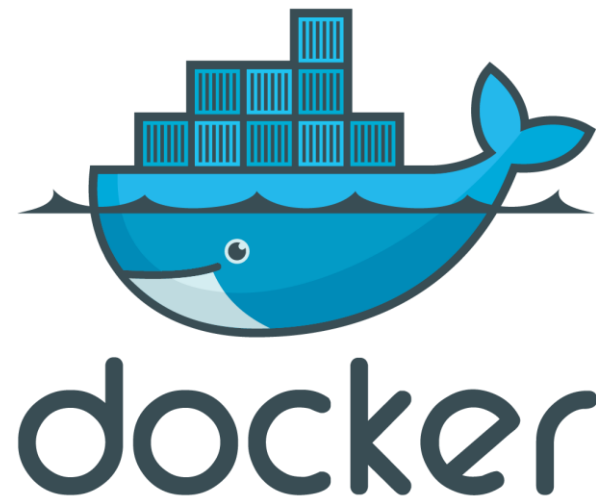


Microservice Deployment

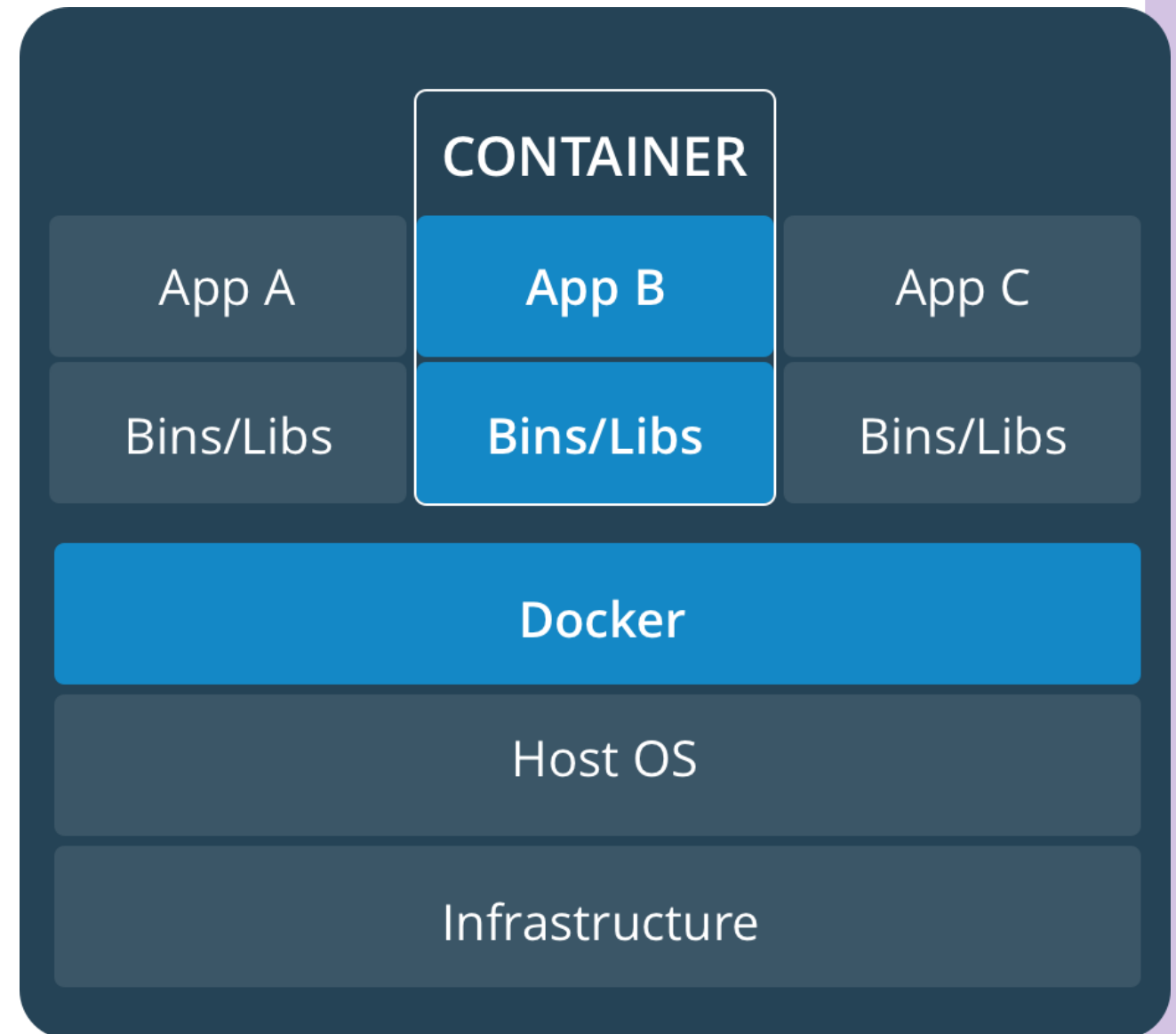
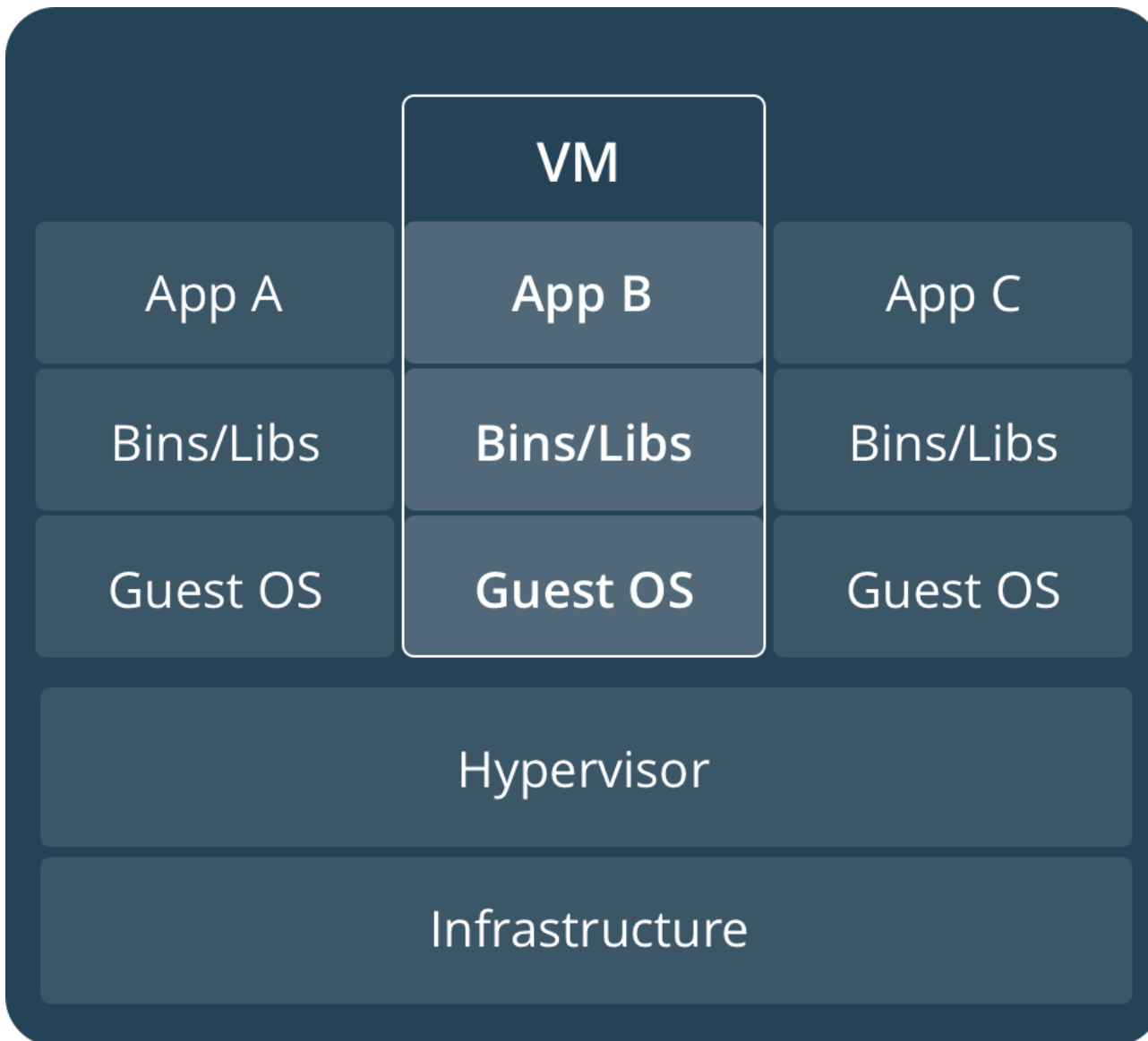
- Ability to deploy/un-deploy independently of other microservices.
- Must be able to scale at each microservices level.
- Building and deploying microservices quickly.
- Failure in one microservice must not affect any of the other services.

Microservice Deployment

- Docker
 - *Docker is becoming an extremely popular way of packaging and deploying services.*
 - Package the microservice as a (Docker) container image.
 - Deploy each service instance as a container.
 - Scaling is done based on changing the number of container instances.
 - Building, deploying and starting microservice will be much faster as we are using docker containers



Virtual Machines Vs. Docker

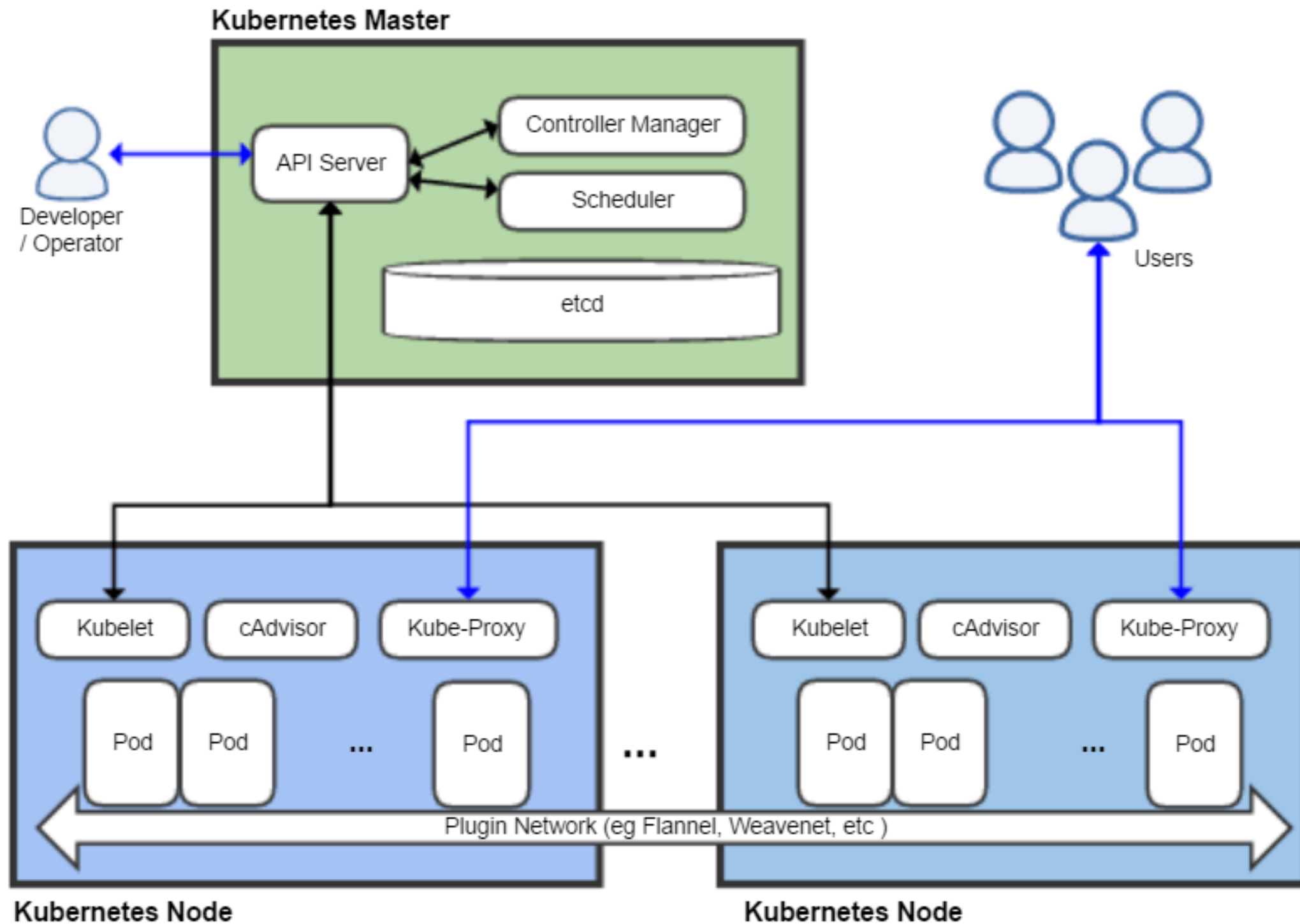


Microservice Deployment

- Kubernetes for container orchestration
 - Extending Docker's capabilities by allowing to manage a cluster of Linux containers as a single system, managing and running Docker containers across multiple hosts, offering co-location of containers, service discovery and replication control.

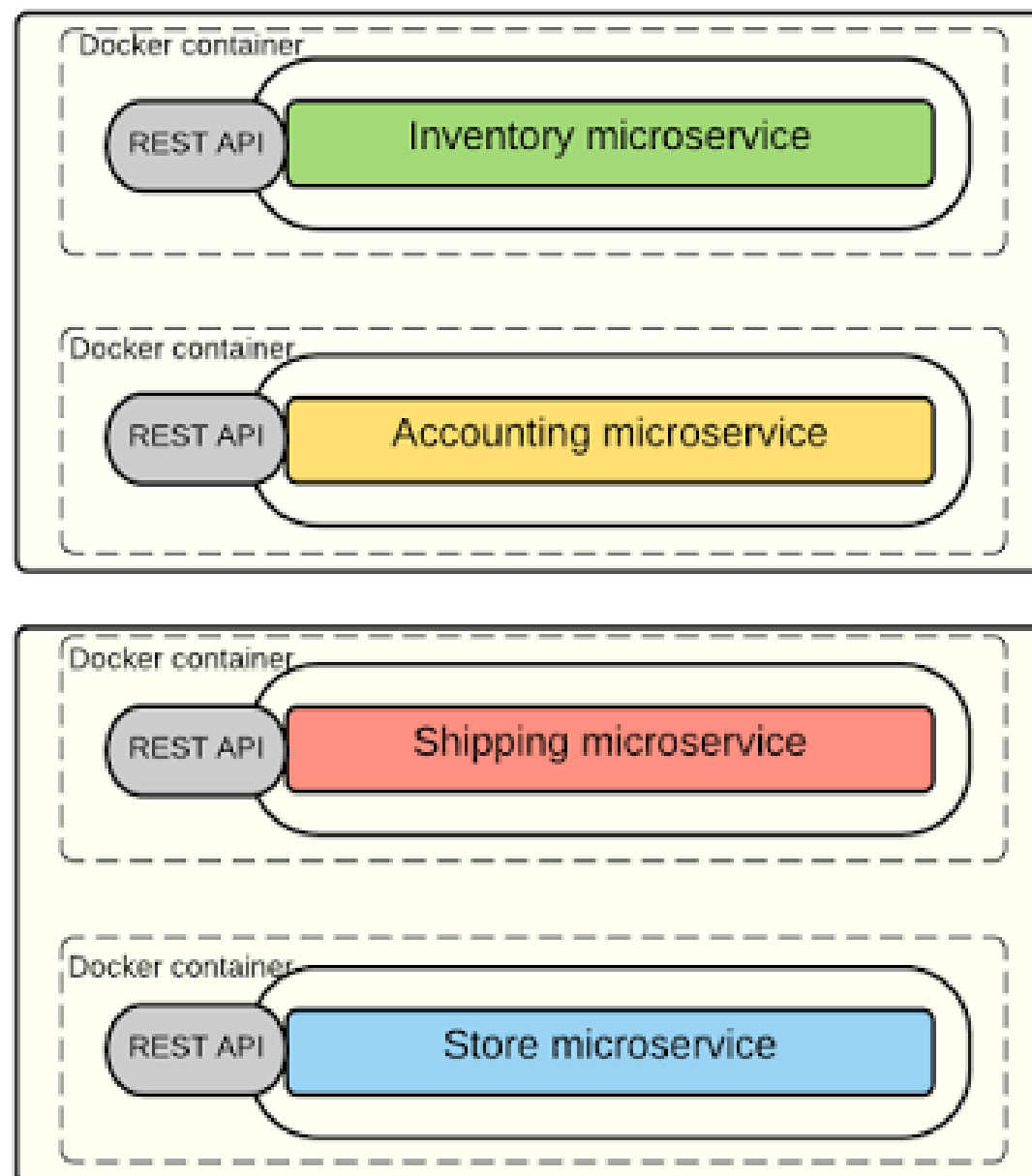


Kubernetes



Microservice Deployment

- Use case : The microservices of Online *Retail software application* with can be deployed and scaled with Docker and Kubernetes.



Security with Microservice

- Security in Monolithic applications
 - Its about '*who is the caller*', '*what can the caller do*' and '*how do we propagate that information*'.
 - Often implemented at a common security component which is at the beginning of the request handling chain and that component populates the required information with the use of an underlying user repository
- Security in Microservices
 - a security component implemented at each microservice's level that uses a central user repository/store.
 - Leverage the widely used API-Security standards such as OAuth2 and OpenID Connect

Security with Microservice

- OAuth 2.0
 - The client authenticates with authorization server and get an opaque token which is known as 'Access token'. Access token has zero information about the user/client.
 - It only has a reference to the user information that can only be retrieved by the Authorization server. Hence this is known as a '**by-reference token**' and it is safe to use this token even in the public network/internet.



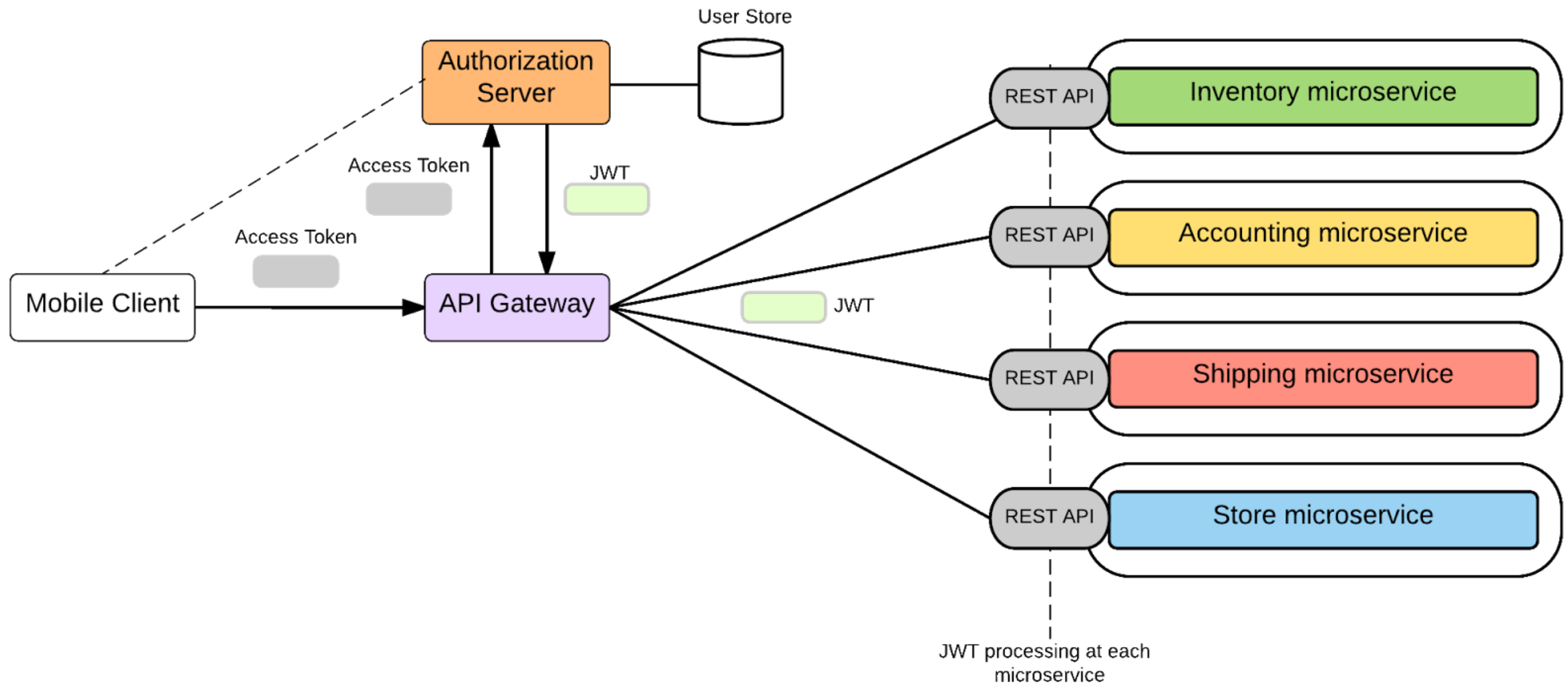
Security with Microservice

- OpenID Connect
 - OpenID Connect behaves similar to OAuth but in addition to the Access token, the authorization server issues an ID token which contains information about the user.
 - Implement with a JWT (JSON Web Token) and that is signed by authorization server. So, this ensures the trust between the authorization server and the client.
 - JWT token is therefore known as a '**By-value token**' as it contains the information of the user and obviously it is not safe to use it outside the internal network.



Security with Microservice

- Microservice security with OAuth2 and OpenID Connect



Transactions

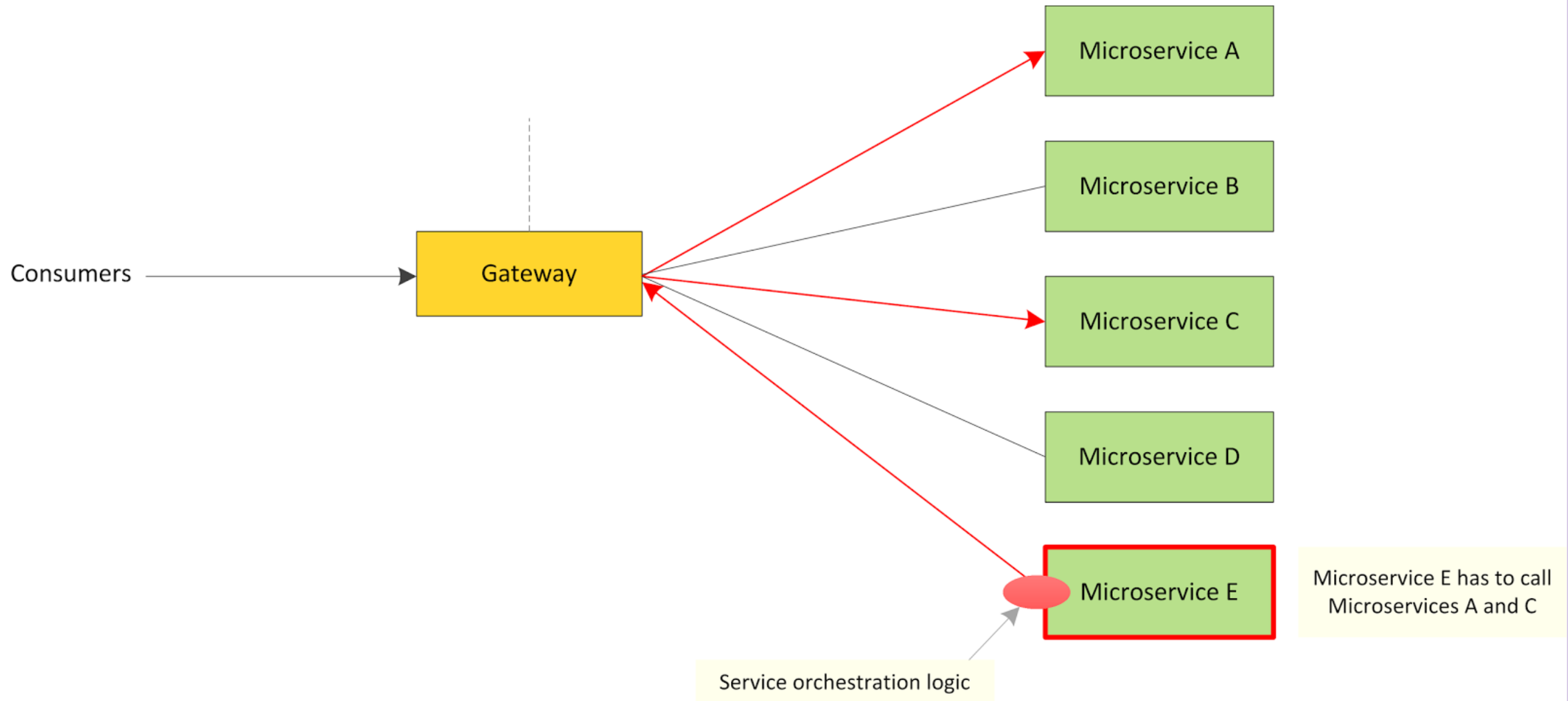
- Supporting distributed transactions across multiple microservices
 - Too complex.
- Microservice architecture itself encourages the transaction-less coordination between services.
- Mandatory transaction requirements can be fulfilled with 'compensating operations'

Design for Failures

- Increases the possibility of having failures at each service level
- Unavailable or unresponsive microservice should not bring the whole system down
- Microservices should be fault tolerant, be able to recover when that is possible and the client has to handle it gracefully.
- Error handling patterns
 - Circuit Breaker
 - Timeout
 - Bulkhead

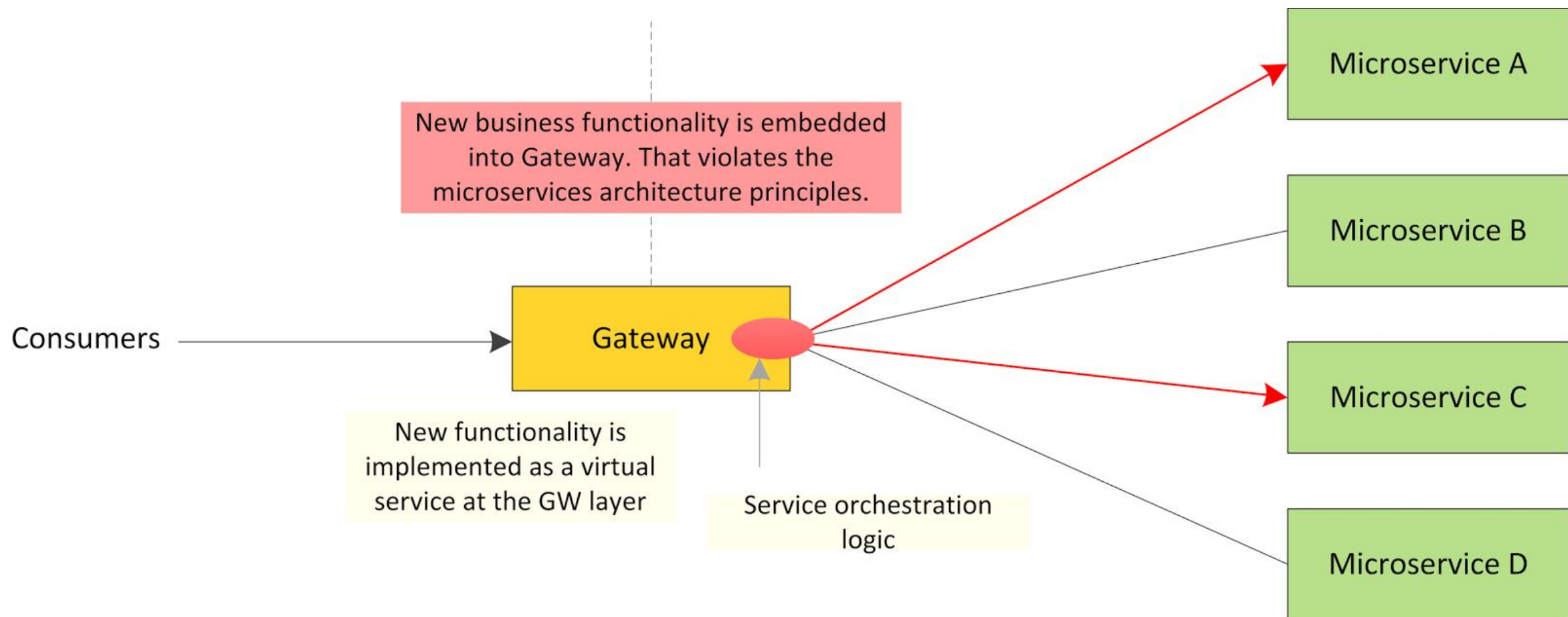
Orchestrating Microservices

- Orchestration at Microservices Layer



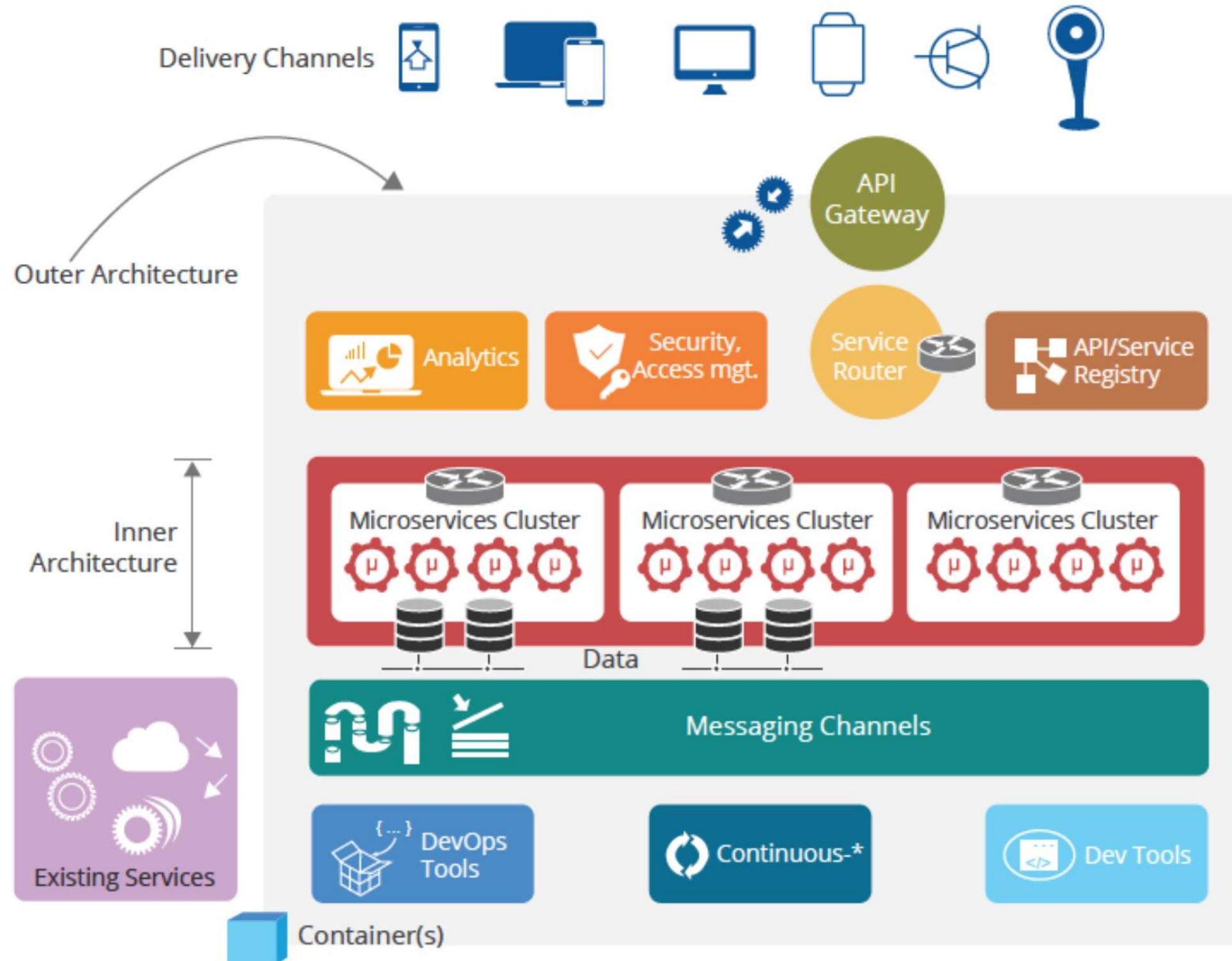
Orchestrating Microservices

- Orchestration at the Gateway Layer – Not recommended



Microservices in Modern Enterprises

- Inner and Outer Architecture

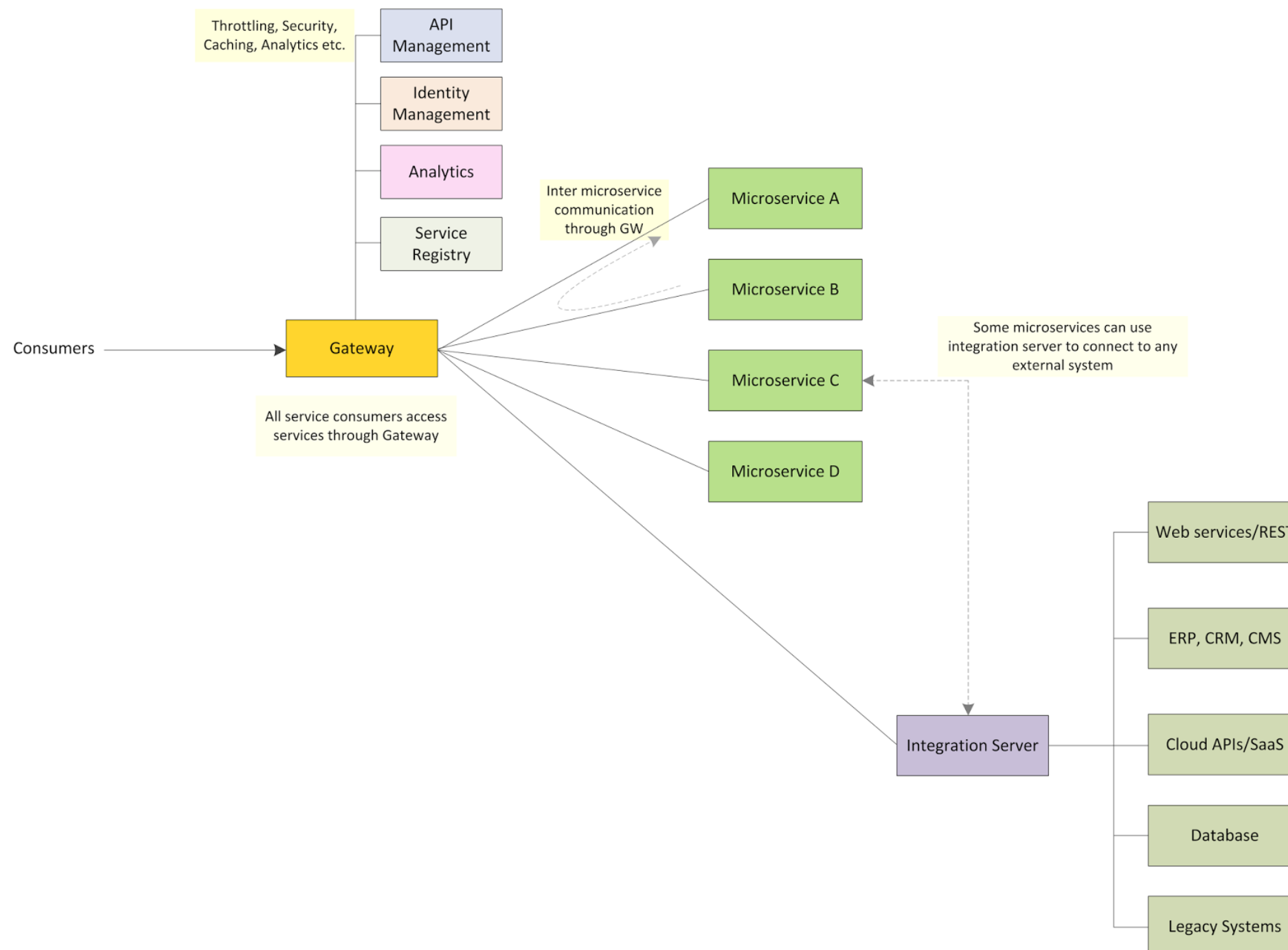


Microservices in Modern Enterprises

- **Inner and Outer Architecture**
 - ***Inner Architecture*** : The pure microservices components which is less complex are categorized under 'Inner Architecture'
 - ***Outer Architecture*** : This delivers the platform capabilities that are required to build a solution around the microservices that we build.

Microservices in Modern Enterprises

- Modern Enterprise Architecture with Microservices, Enterprise Integration and API Management



Migrating to Microservices

- Challenging to migrate from a monolithic system to a microservices-based system
- May have to do it gradually
- Most systems may eventually have a hybrid architecture, which is some combination of monolithic and microservices architectures

Microservices – case study

- Mastering Chaos - A Netflix Guide to Microservices

<https://www.youtube.com/watch?v=CZ3wluvmHeM>

Here Josh Evans talks about the chaotic and vibrant world of microservices at Netflix. He starts with the basics- the anatomy of a microservice, the challenges around distributed systems, and the benefits. Then he builds on that foundation exploring the cultural, architectural, and operational methods that lead to microservice mastery.

Microservices – Conclusion

- Microservices is not a panacea : It won't solve all your enterprise IT needs
- 'SOA done right'?
- Most enterprises won't be able to convert their entire enterprise IT systems to microservices.
- Enterprise Integration never goes away.
- Microservices are exposed as APIs.
- Interaction between microservices should be support via a lightweight orchestration engine/Gateway or inside another microservice.

References

- Microservices in Practice, <http://kasunpanorama.blogspot.com/2015/11/microservices-in-practice.html>
- Microservices in Modern Enterprise Architecture, <http://kasunpanorama.blogspot.com/2016/02/microservices-enterprise-integration.html>
- A pattern language for microservices, <http://microservices.io/patterns/>
- Microservices Guide, <http://martinfowler.com/microservices/>
- Docker, <https://www.docker.com/>
- Kubernetes, <https://kubernetes.io/>