Service-Based Architecture

Software Architecture

Richard Thomas

March 20, 2023

Definition 1. Distributed System

A system with multiple components located on different machines that communicate and coordinate actions in order to appear as a single coherent system to the end-user.

Introduce idea of distributed systems and then move on to service-based being an simple approach.

Quote

A distributed system is one in which the failure
of a computer you didn't even know existed
can render your own computer unusable.

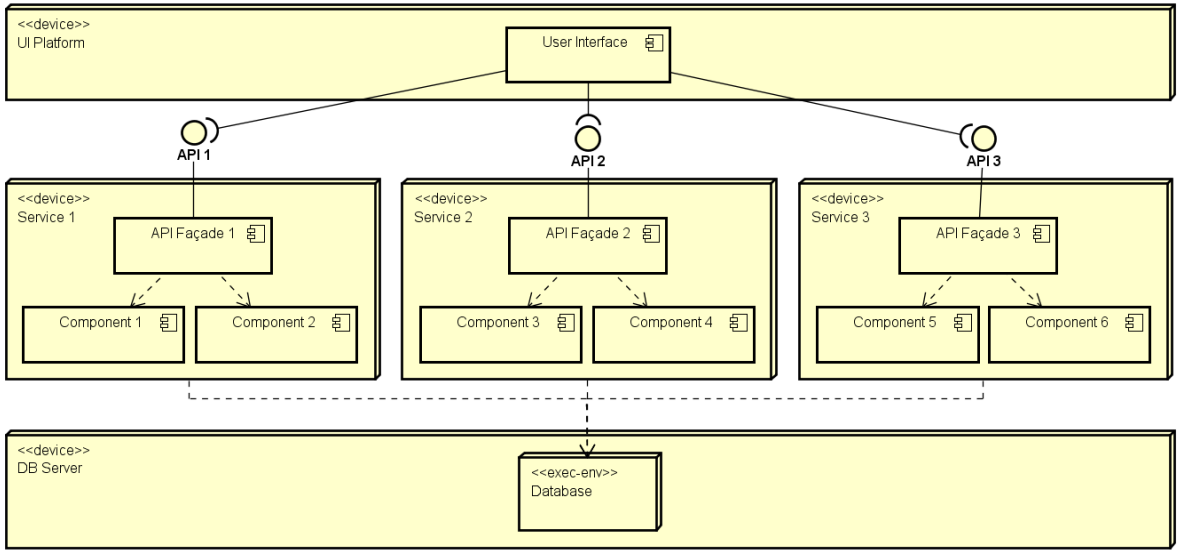
– Leslie Lamport [Turing Award, 2013]

Definition 2. Service-Based Architecture

System is partitioned into business domains that are deployed as distributed services. Functionality is delivered through a user interface that interacts with the domain services.

Explain why this leads to a fairly simple distributed architecture.

Service-Based Architecture



User Interface Provides access to system functionality

Services Implement functionality for a single, independent business process

Service APIs Communication mechanism between UI and each service

Database Stores persistent data for the system

- Explain that the Service APIs are communication protocols and data formats, not just a Java-style interface.
- Usually all Service APIs use the same communication protocol (e.g. REST).
- Also point out that messages between the UI and services will typically be asynchronous.

Definition 3. API Abstraction Principle

Services should provide an API that hides implementation details.

- Each service publishes its own API.
- Hides service implementation details, reducing coupling between UI and service.
- Makes it easier to reuse service across systems or by supporting service (e.g. auditing).

Definition 4. Façade Design Pattern

Provide a simple, abstract interface to use a service domain's functionality. A component within the service coordinates how to deliver the requested functionality with the service's internal components.

Summarise Façade Design Pattern and how it is used in a service-based architecture. Mention its from the GoF book.

Definition 5. Independent Service Principle

Services should be independent, with no dependencies on other services.

- Explain consequences of dependencies between services.
- Services can't easily be deployed separately if they depend on other services.
- They would require interfaces between services, increasing coupling.

Question

What are the consequences of having a shared database?

Question

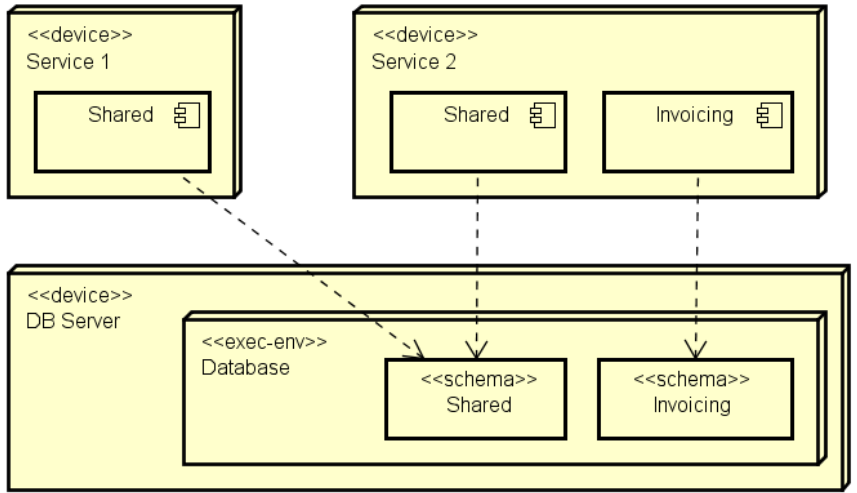
What are the consequences of having a shared database?

Answer

Increased *data coupling*.

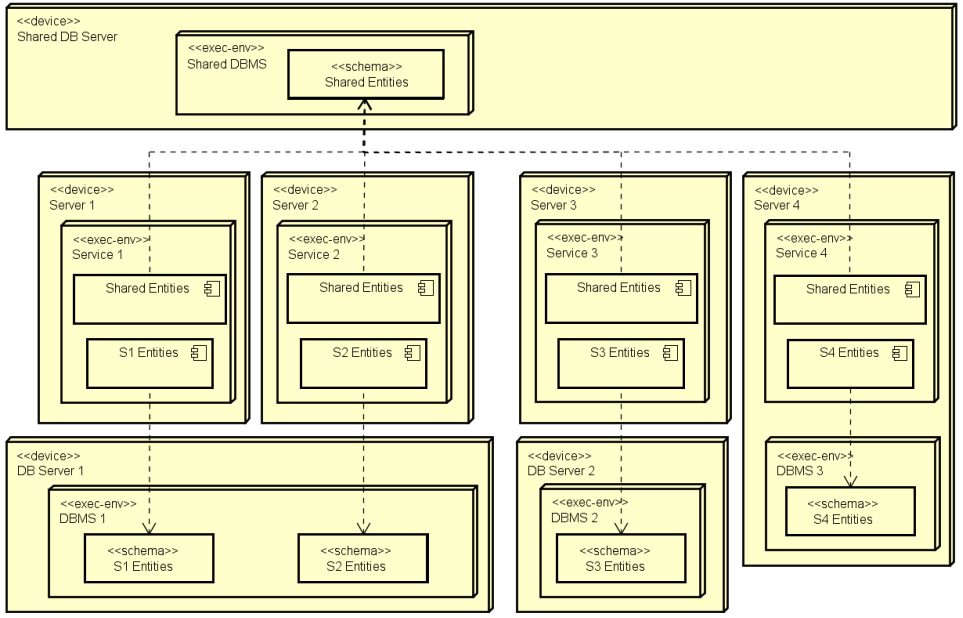
- If a row of a database is locked and another service wants to use it, it is blocked. Losing efficiency benefits of a distributed system.
- If one service changes the structure of its persistent data, all services using that data need to be updated and tested.
- If one service changes how it uses persistent data, all other services using the same data need to be retested.

Logical Partitioning of Persistent Data



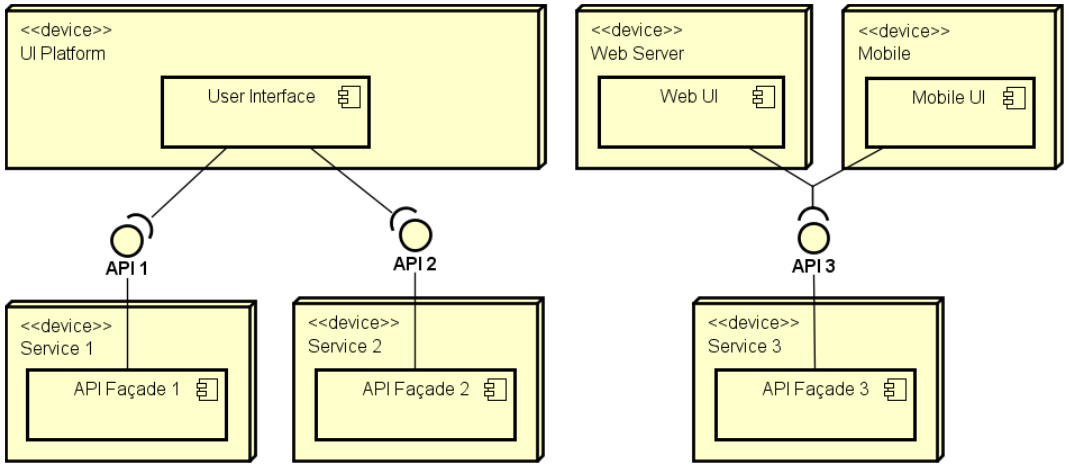
- Define a minimal set of shared persistent objects.
- Create a shared library to access these objects.
- Changes to shared persistent objects are restricted as they require changes to other services.
- Each service may have its own persistent objects stored in tables that are not shared with other services.

Separate Databases



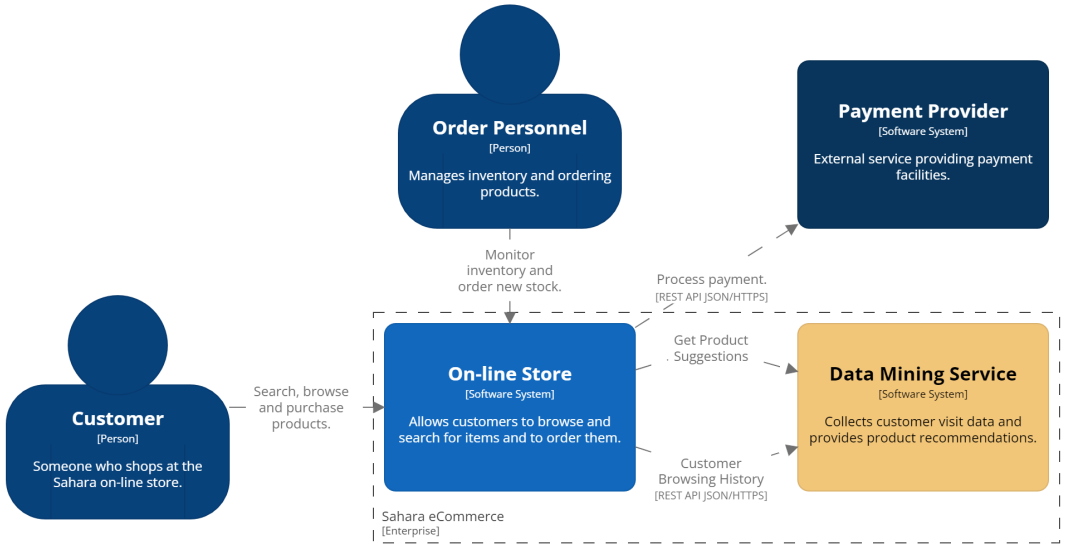
Discuss options of separate DB servers, share DBs on one server, DBs embedded in application.

Separate UIs



- UI Platform could be desktop, web or mobile app.
- This allows multiple concurrent users, even through one user interface.

Sahara: Context Diagram



- Summarise Sahara eCommerce example.
- Order Personnel & Payment Provider added to this example.

On-line Store Service Domains

Browsing Customers can find products & add to cart

Purchasing Customers can purchase products in cart

Fulfilment Customers & staff can track order fulfilment

Account Management Customers can manage their account details

Inventory Management Staff can view stock levels and order new stock

- Mention that service-based architectures are based on domain partitioning.
- Could provide examples of fulfilment, and inventory management activities.
- Customer's tracking order, staff generating pick lists and packaging details.
- Generating reports on stock levels and product popularity. Ordering new stock.

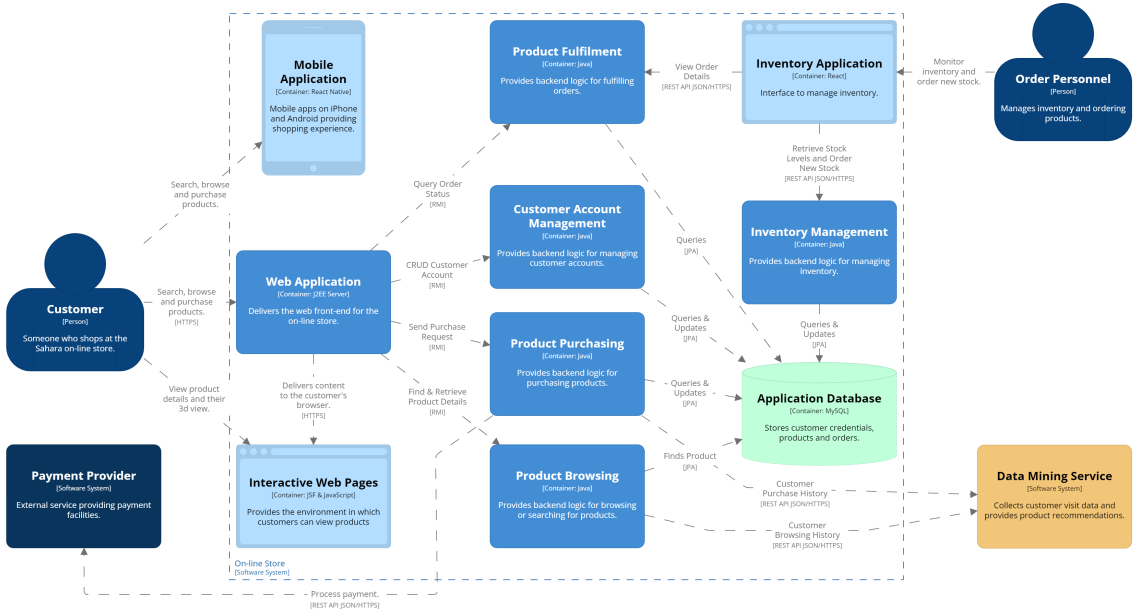
Partitioning

Services are defined by domain partitioning

Coarse Services

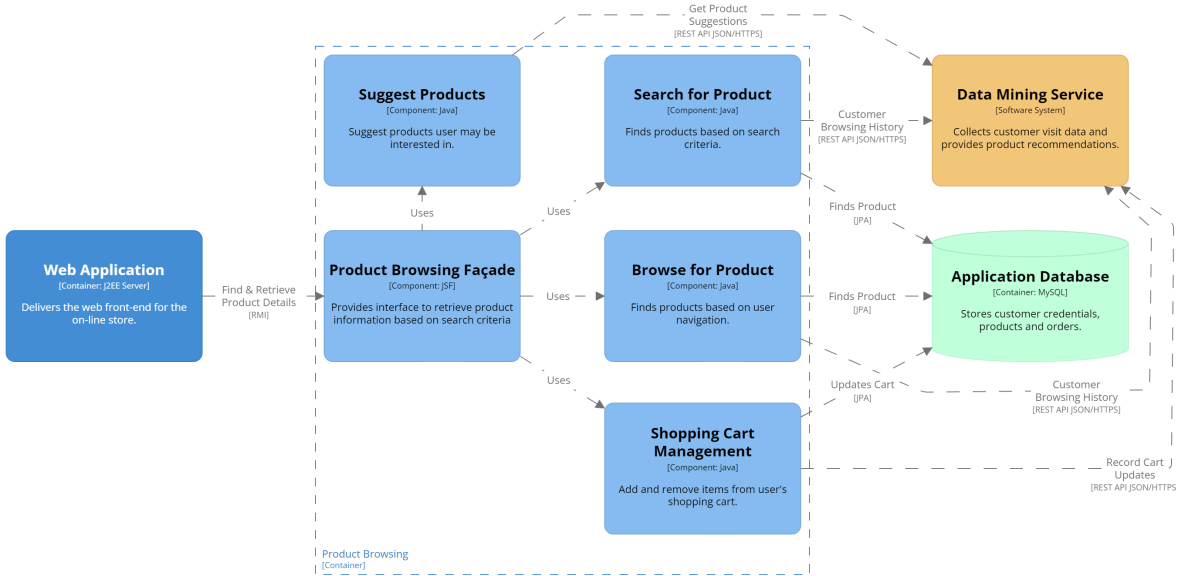
- Domains are large
  - *Coarse-grained* services
- Each service will have an internal architecture
  - Technical or domain partitioning

Sahara: On-line Store Container Diagram



- Review Domain Services in context of the overall system.
- Mobile App relationships not shown to reduce clutter.
- Single DB for simplicity of diagram, not good practice. Should be split.
- Cart needs to be shared with Browsing & Purchasing.
- Order needs to be shared with Purchasing & Fulfilment.
- Product needs to be shared with everything except Account Management.
- User needs to be shared with almost everything.
- Most of these will be query only, or only locking a single row of some tables.
- Repeat idea that Service APIs mean you may have multiple UIs (Web, Mobile, Inventory Apps).

Sahara: Product Browsing Component Diagram



- Summarise the components making up the key parts of the Product Browsing Service (container).
- Product Browsing Façade provides the Service API.

Product Browsing Service API

Search

https://api.sahara.com/v1/search?keywords=...

Browse

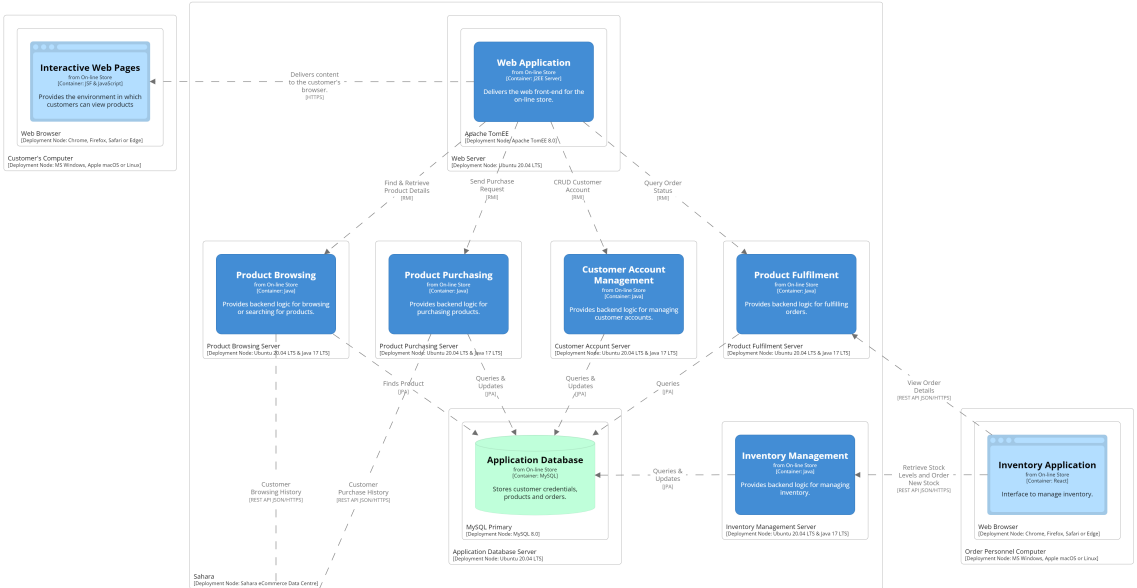
https://api.sahara.com/v1/browse?category=...

Add to Cart https://api.sahara.com/v1/cart

- JSON to pass data
- JSF action controller handles request

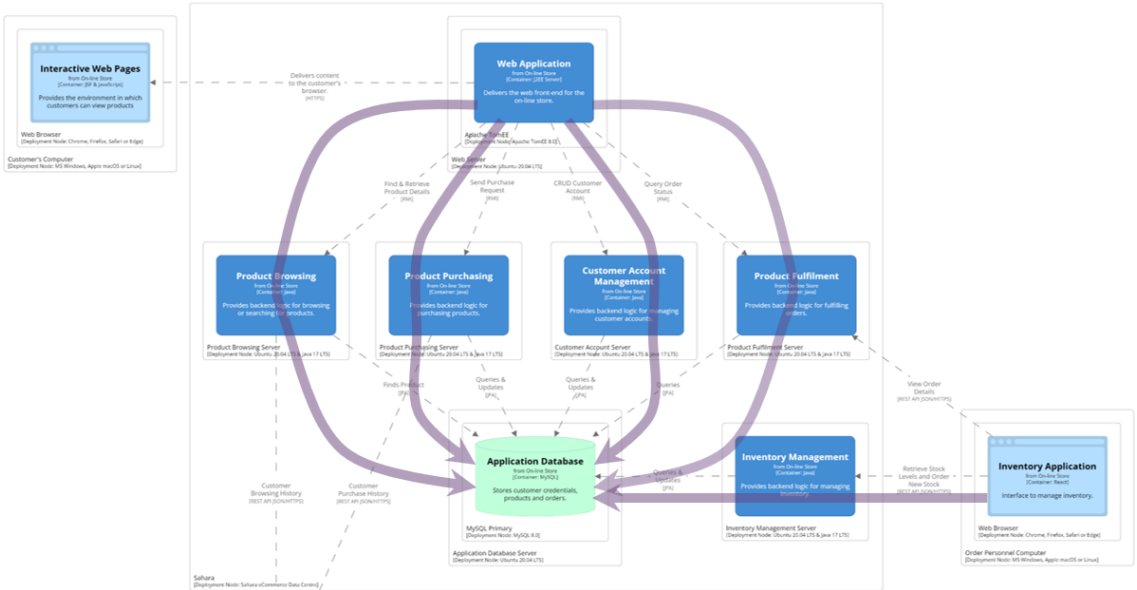
- Search & Browse are GET requests passing parameters.
- Add to Cart is a POST request passing product id and quantity to be added to cart.
- Authentication needs to be part of requests.
- API Versioning shown in URIs.

Sahara: Deployment Diagram



- UI Platform could be desktop, web or mobile app.
- This allows multiple concurrent users, even through one user interface.

Sahara: Concurrent Access



- Emphasise many users from different UIs accessing distributed services concurrently.
- Point out that this & REST require stateless services.

Question

What happens if a service goes down?

Question

What happens if a service goes down?

Answer

Need to manage timeouts, retries, graceful failure, ...

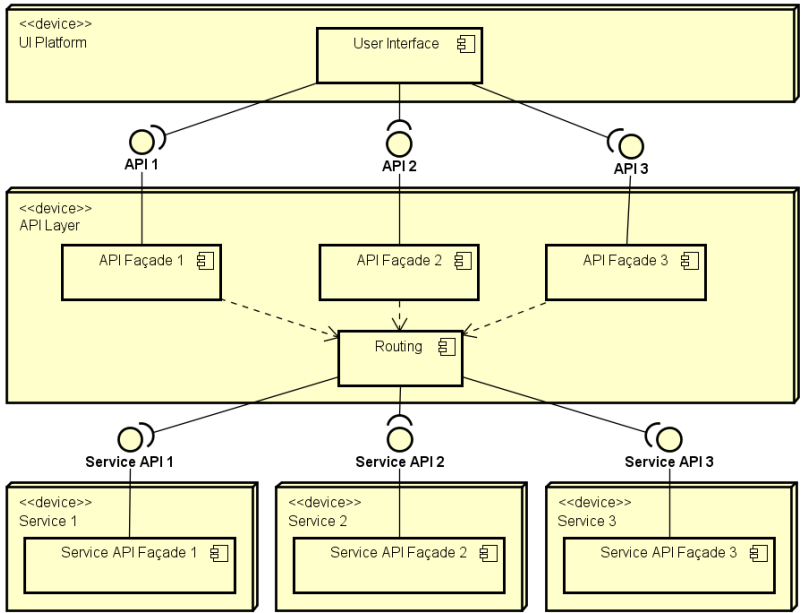
- Some of this can be managed by infrastructure, which requires monitoring systems.
- Some issues are harder to deal with due to coarse service domains.
- Some of this needs to be managed within the application.

Consider Network Failure

If customer tried to add product to cart:

- What happens if Product Browsing didn't receive it?
- What happens if UI didn't get a response?
- What happens if Database wasn't updated?

API Layer


<<device>>
UI Platform
User Interface
API 1
API 2
API 3
<<device>>
API Layer
API Façade 1
API Façade 2
API Façade 3
Routing
Service API 1
Service API 2
Service API 3
<<device>>
Service 1
Service API Façade 1
<<device>>
Service 2
Service API Façade 2
<<device>>
Service 3
Service API Façade 3

API Layer Advantages

- Acts as a reverse proxy or gateway to services
- Hides internal network structure
- Easier to implement *cross-cutting* concerns
  - e.g. security policies
- Allows service discovery
  - Interface to register service
  - Clients can find out what services are available

Pros & Cons

- Simplicity For a distributed system 
- Modularity Services 
- Extensibility New services 
- Deployability Independent services 
- Testability Independent services 
- Security API layer 
- Reliability Independent services 
- Interoperability Service APIs 
- Scalability Coarse-grained services