Service-Based Architecture

Software Architecture

Richard Thomas

March 16, 2026

> **_Definition 0._** 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 a 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]
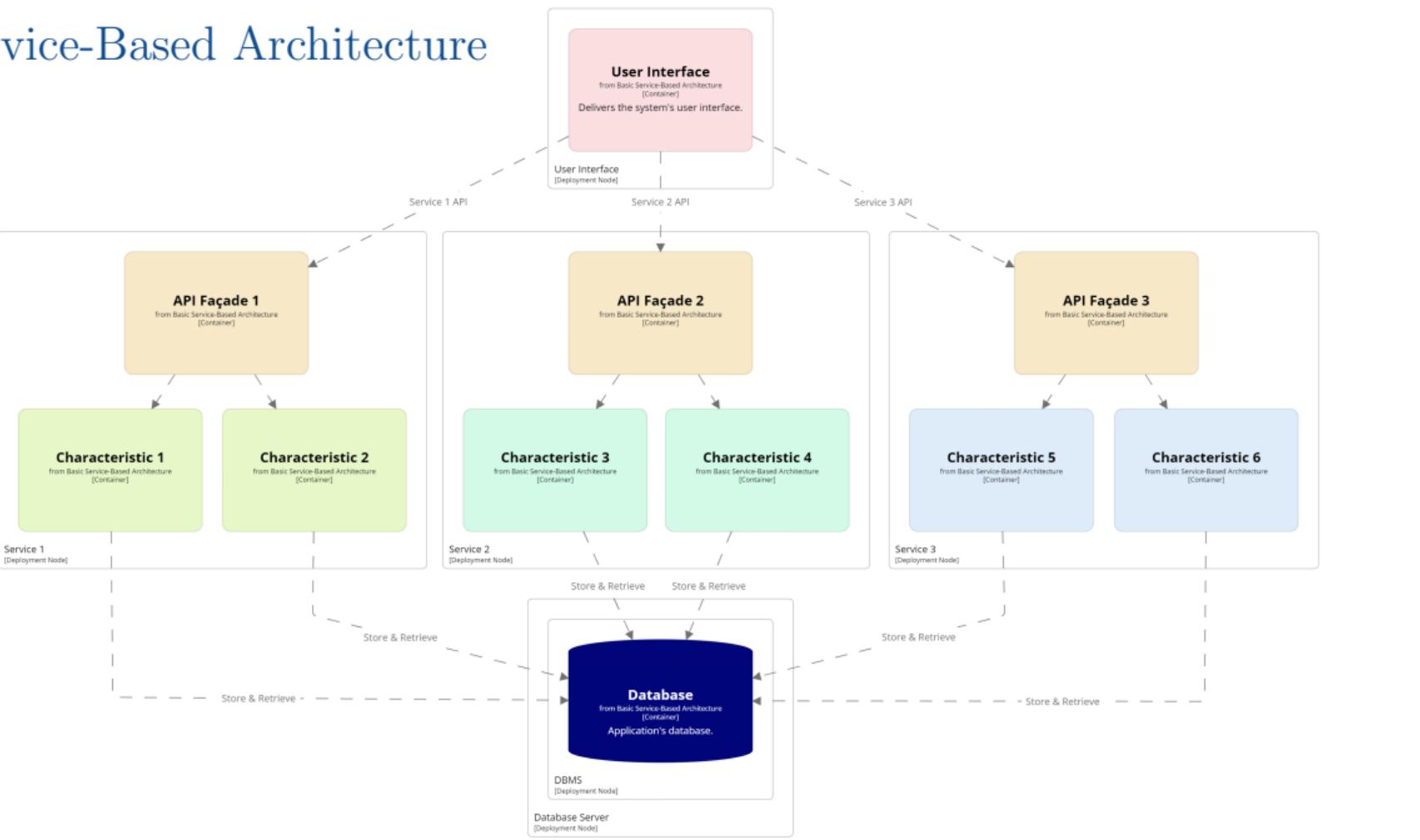
> **Definition 0.** Service-Based Architecture
>
> System is partitioned into business domains that are deployed as distributed services. Functionality is delivered through a user interface that interacts with the domain services.

Explain why this leads to a fairly simple distributed architecture.

Service-Based Architecture

User Interface
from Basic Service-Based Architecture
[Container]
Delivers the system's user interface.

User Interface
[Deployment Node]

Service 1 API

Service 2 API

Service 3 API

API Façade 1
from Basic Service-Based Architecture
[Container]

API Façade 2
from Basic Service-Based Architecture
[Container]

API Façade 3
from Basic Service-Based Architecture
[Container]

Characteristic 1
from Basic Service-Based Architecture
[Container]

Characteristic 2
from Basic Service-Based Architecture
[Container]

Characteristic 3
from Basic Service-Based Architecture
[Container]

Characteristic 4
from Basic Service-Based Architecture
[Container]

Characteristic 5
from Basic Service-Based Architecture
[Container]

Characteristic 6
from Basic Service-Based Architecture
[Container]

Service 1
[Deployment Node]

Service 2
[Deployment Node]

Service 3
[Deployment Node]

Store & Retrieve

Store & Retrieve

Store & Retrieve

Store & Retrieve

Store & Retrieve

Store & Retrieve

Database
from Basic Service-Based Architecture
[Container]
Application's database.

DBMS
[Deployment Node]

Database Server
[Deployment Node]

Terminology

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 0._ 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 0.** 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 0.** 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.

What are the consequences of having a shared database?
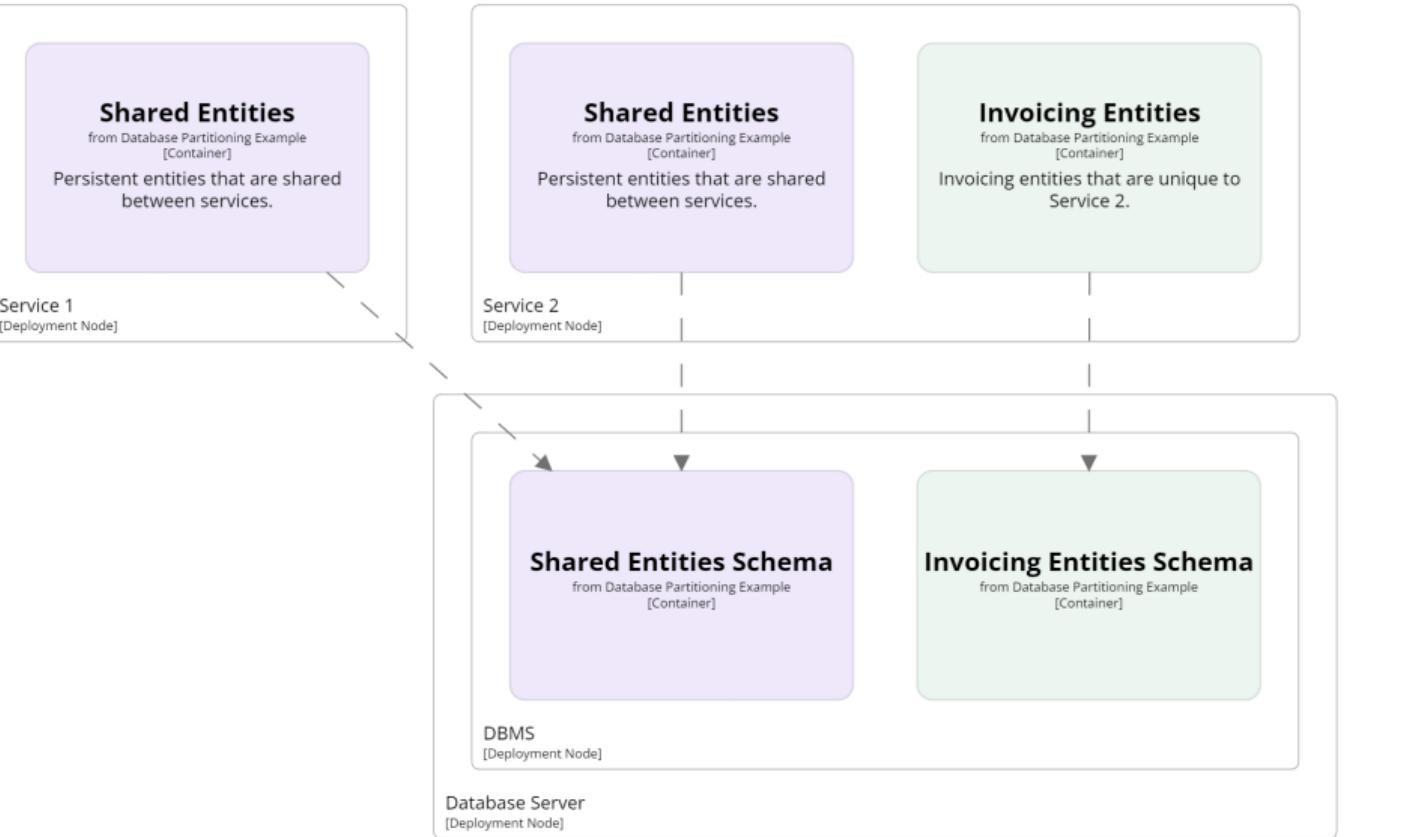
What are the consequences of having a shared database?

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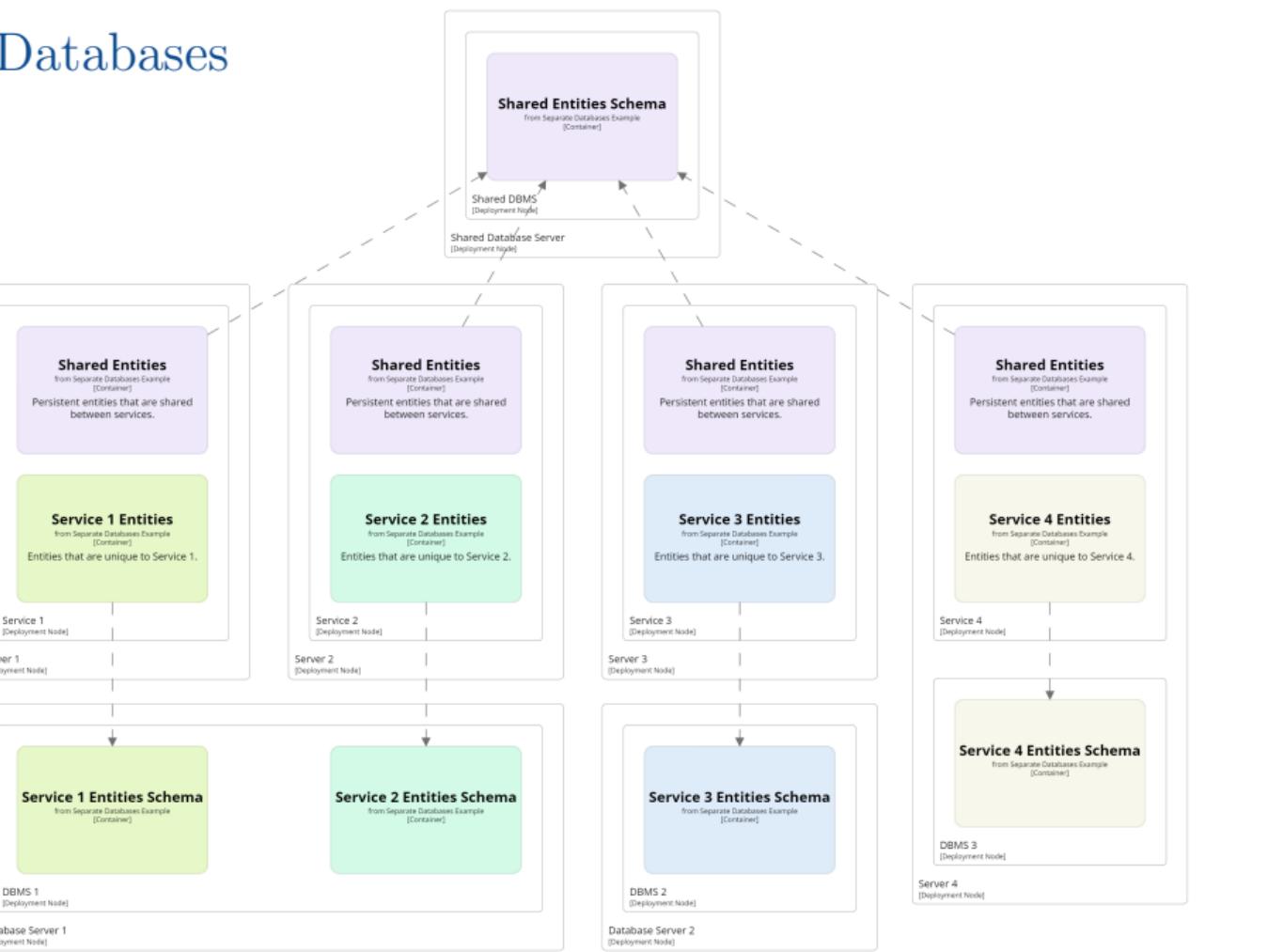
- If a row of a database is locked and another service wants to use it, it is blocked. Losing efficiency benefits of a distributed system.
- If one service changes the structure of its persistent data, all services using that data need to be updated and tested.
- If one service changes how it uses persistent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

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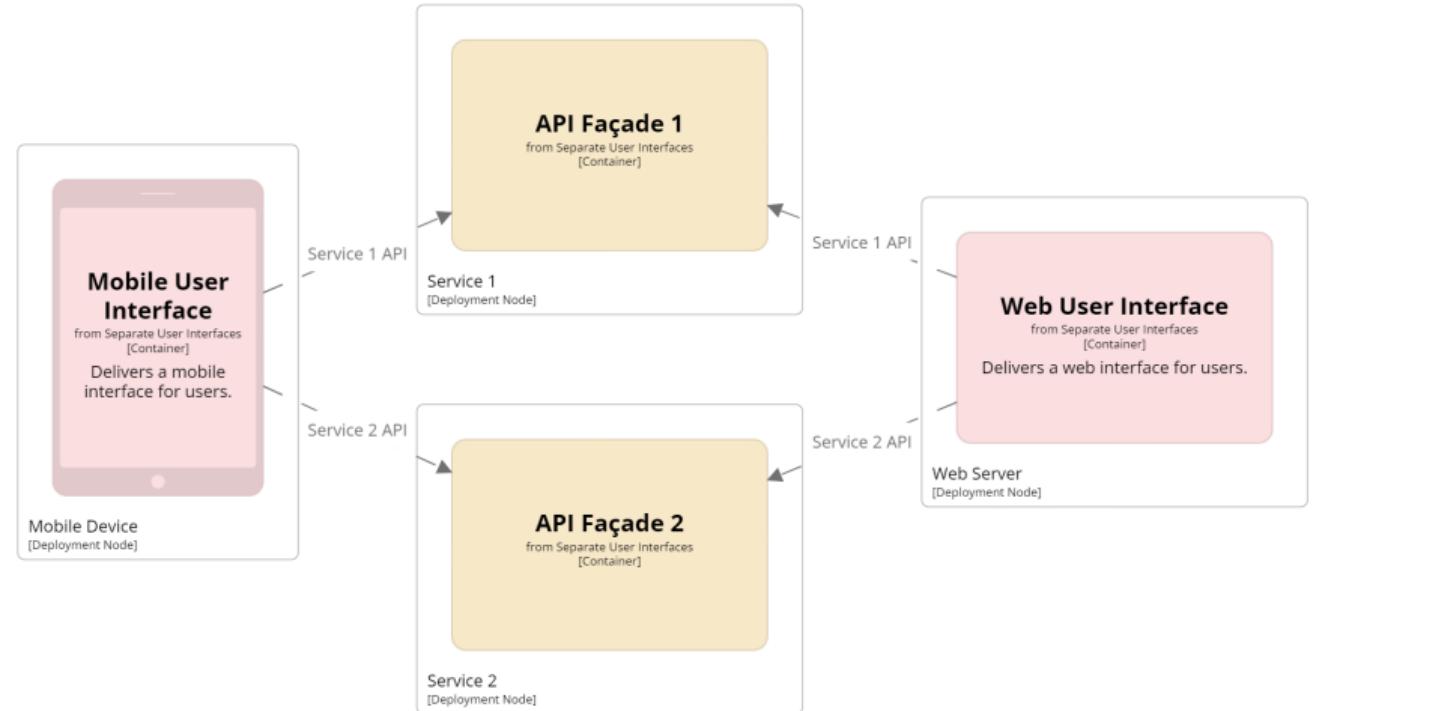
- Define a minimal set of shared persistent objects.
- Create a shared library to access these objects.
- Changes to shared persistent objects are restricted as they require changes to other services.
- Each service may have its own persistent objects stored in tables that are not shared with other services.

Separate Databases
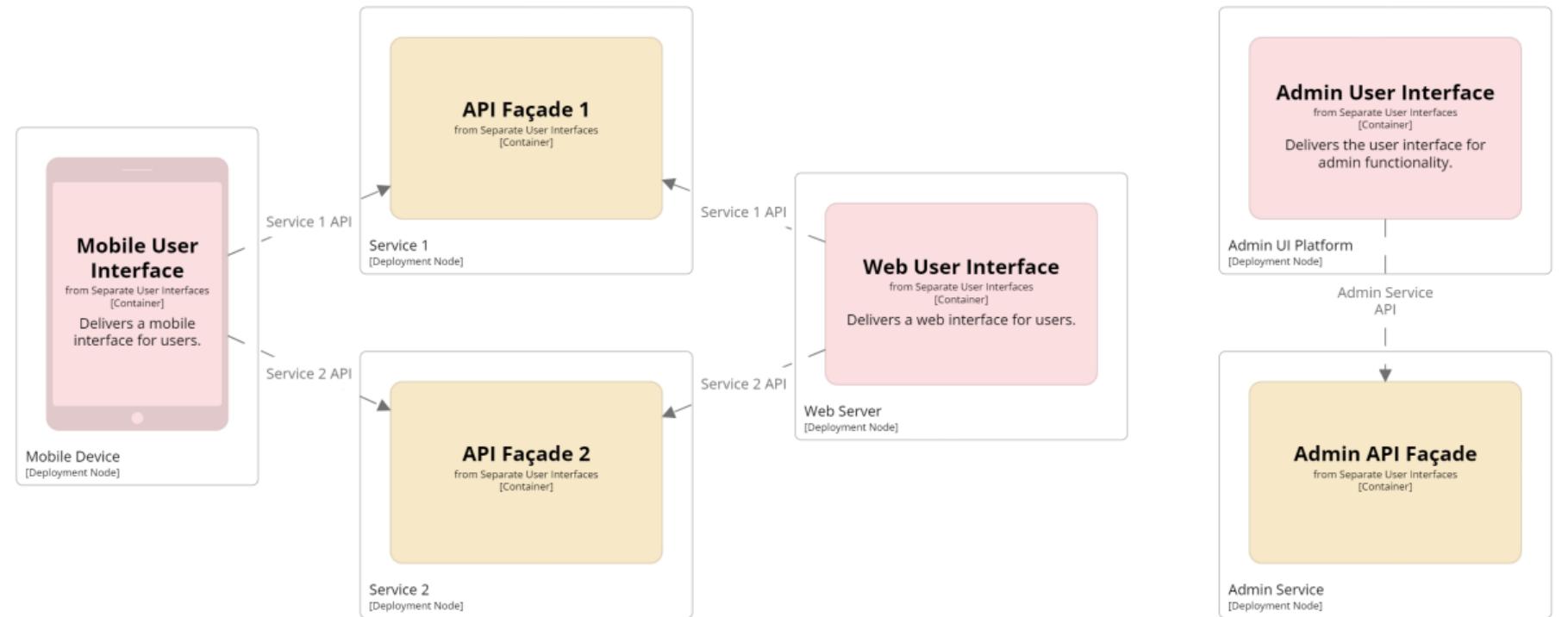
- Discuss options of
-     separate DB servers,
-     multiple DBs on one server,
-     DBs embedded in application.

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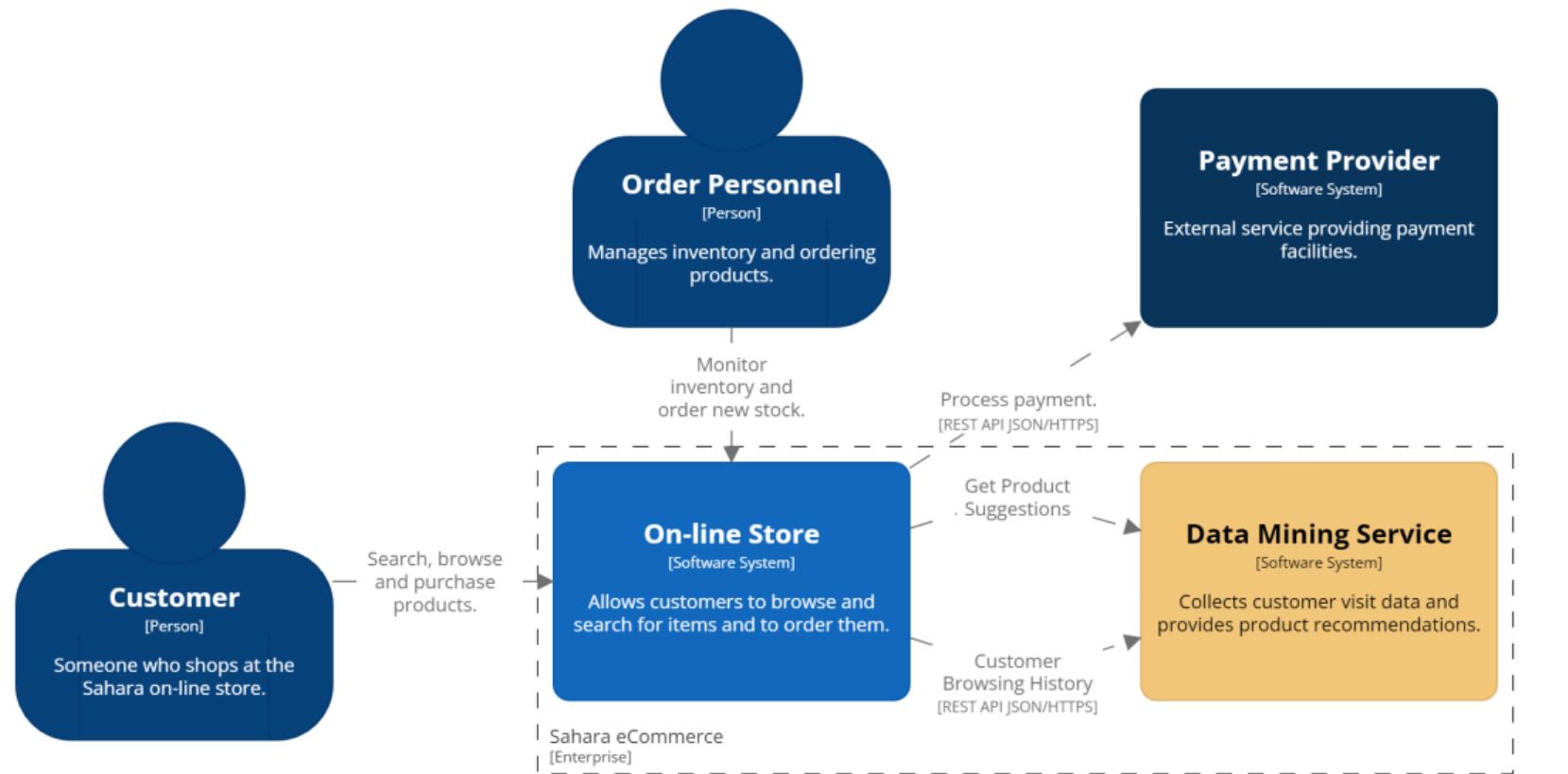
- UI Platform could be desktop, web or mobile app.
- Allows multiple concurrent users, even through one user interface.

Separate UIs

- UI Platform could be desktop, web or mobile app.
- Allows multiple concurrent users, even through one user interface.
- Allows *separate UIs* for different clients or roles.

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.
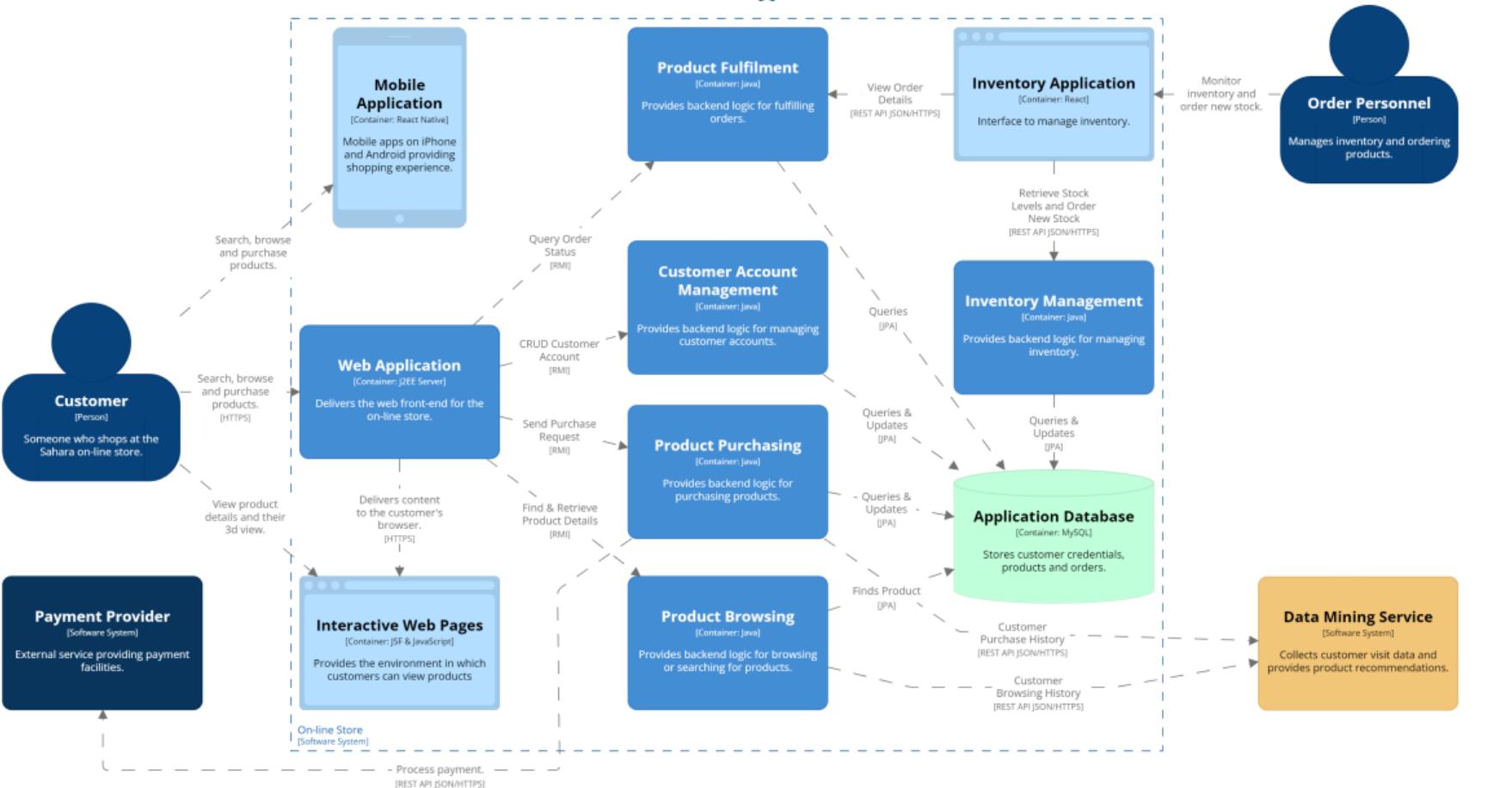
Partitioning

Services are defined by domain partitioning
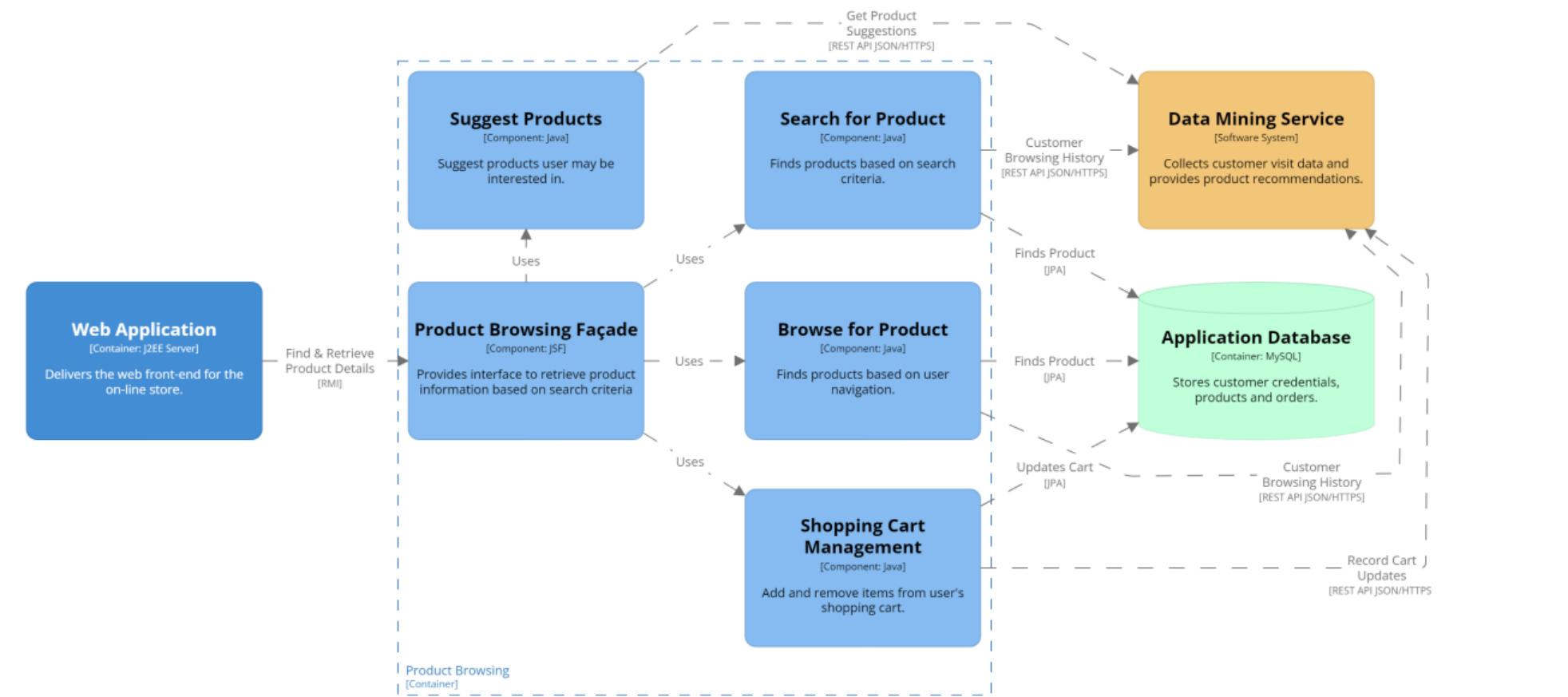
- Domains are large
  - *Coarse-grained* services

- 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 shared with Browsing & Purchasing.
- Order shared with Purchasing & Fulfilment.
- Product shared with everything except Account Mgt.
- User shared with almost everything.
- Most of these will be query only, or only locking a single row.
- Repeat idea that Service APIs mean you may have multiple UIs (Web, Mobile, Inventory Apps).
- Inventory app is example of completely separate UI delivery a different app.

Sahara: Product Browsing Component Diagram

Web Application
[Container: J2EE Server]
Delivers the web front-end for the on-line store.

Find & Retrieve Product Details [RMI]

Product Browsing Façade
[Component: JSF]
Provides interface to retrieve product information based on search criteria

Suggest Products
[Component: Java]
Suggest products user may be interested in.

Search for Product
[Component: Java]
Finds products based on search criteria.

Browse for Product
[Component: Java]
Finds products based on user navigation.

Shopping Cart Management
[Component: Java]
Add and remove items from user's shopping cart.

Data Mining Service
[Software System]
Collects customer visit data and provides product recommendations.

Application Database
[Container: MySQL]
Stores customer credentials, products and orders.

Uses

Get Product Suggestions [REST API JSON/HTTPS]

Customer Browsing History [REST API JSON/HTTPS]

Finds Product [JPA]

Finds Product [JPA]

Updates Cart [JPA]

Customer Browsing History [REST API JSON/HTTPS]

Record Cart Updates [REST API JSON/HTTPS]

Product Browsing
[Container]

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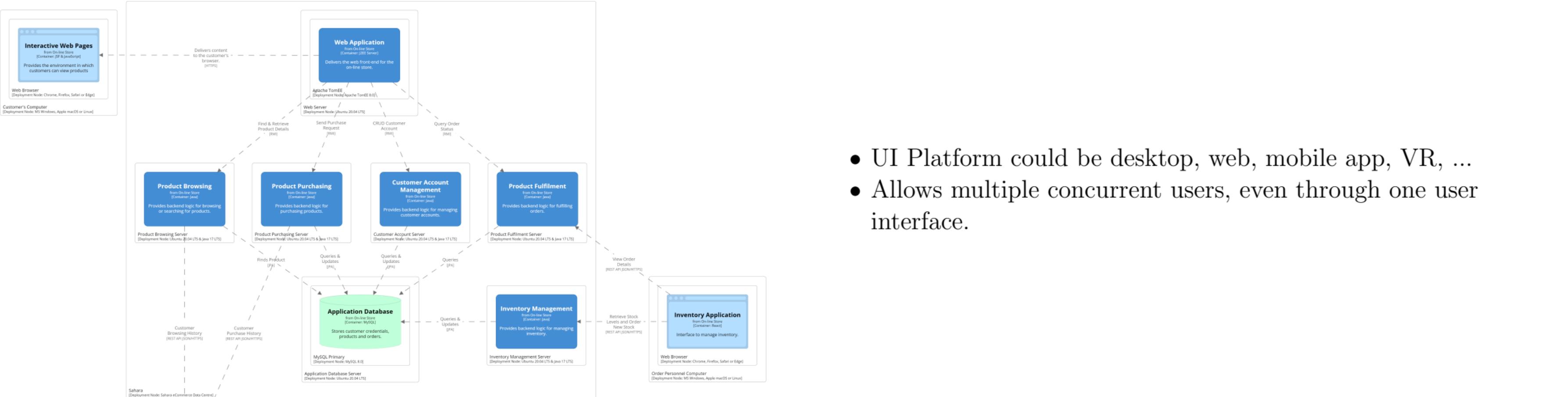
Add to Cart https://api.sahara.com/v1/cart

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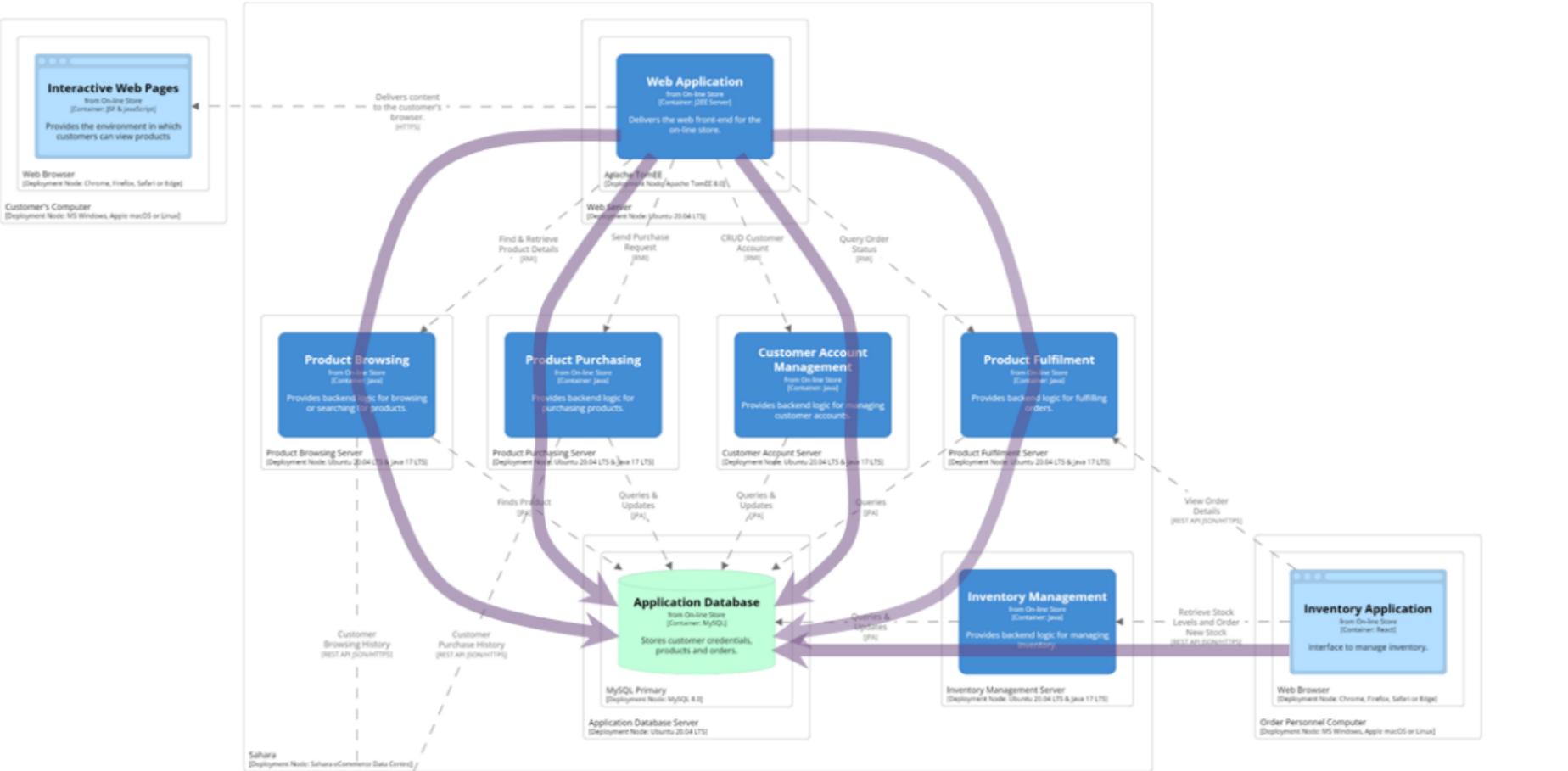
- Search & Browse are GET requests passing parameters.
- Add to Cart is a POST request passing product id and quantity to be added to cart.
- Authentication needs to be part of requests.
- API Versioning shown in URIs.

Sahara: Deployment Diagram

Interactive Web Pages
from On-line Store
[Container: JSP & JavaScript]

Provides the environment in which customers can view products

Web Browser
[Deployment Node: Chrome, Firefox, Safari or Edge]

Customer's Computer
[Deployment Node: MS Windows, Apple macOS or Linux]

Web Application
from On-line Store
[Container: J2EE Server]

Delivers the web front-end for the on-line store.

Apache TomEE
[Deployment Node: Apache TomEE 8.0]

Web Server
[Deployment Node: Ubuntu 20.04 LTS]

Delivers content to the customer's browser.
[HTTPS]

Find & Retrieve Product Details
[RMI]

Send Purchase Request
[RMI]

CRUD Customer Account
[RMI]

Query Order Status
[RMI]

Product Browsing
from On-line Store
[Container: Java]

Provides backend logic for browsing or searching for products.

Product Browsing Server
[Deployment Node: Ubuntu 20.04 LTS & Java 17 LTS]

Product Purchasing
from On-line Store
[Container: Java]

Provides backend logic for purchasing products.

Product Purchasing Server
[Deployment Node: Ubuntu 20.04 LTS & Java 17 LTS]

Customer Account Management
from On-line Store
[Container: Java]

Provides backend logic for managing customer accounts.

Customer Account Server
[Deployment Node: Ubuntu 20.04 LTS & Java 17 LTS]

Product Fulfilment
from On-line Store
[Container: Java]

Provides backend logic for fulfilling orders.

Product Fulfilment Server
[Deployment Node: Ubuntu 20.04 LTS & Java 17 LTS]

Finds Product
[JPA]

Queries & Updates
[JPA]

Queries & Updates
[JPA]

Queries
[JPA]

View Order Details
[REST API JSON/HTTPS]

Customer Browsing History
[REST API JSON/HTTPS]

Customer Purchase History
[REST API JSON/HTTPS]

Application Database
from On-line Store
[Container: MySQL]

Stores customer credentials, products and orders.

MySQL Primary
[Deployment Node: MySQL 8.0]

Application Database Server
[Deployment Node: Ubuntu 20.04 LTS]

Queries & Updates
[JPA]

Inventory Management
from On-line Store
[Container: Java]

Provides backend logic for managing inventory.

Inventory Management Server
[Deployment Node: Ubuntu 20.04 LTS & Java 17 LTS]

Retrieve Stock Levels and Order New Stock
[REST API JSON/HTTPS]

Inventory Application
from On-line Store
[Container: React]

Interface to manage inventory.

Web Browser
[Deployment Node: Chrome, Firefox, Safari or Edge]

Order Personnel Computer
[Deployment Node: MS Windows, Apple macOS or Linux]

Sahara
[Deployment Node: Sahara eCommerce Data Centre]

- UI Platform could be desktop, web, mobile app, VR, ...
- 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.

What happens if a service goes down?

What happens if a service goes down?

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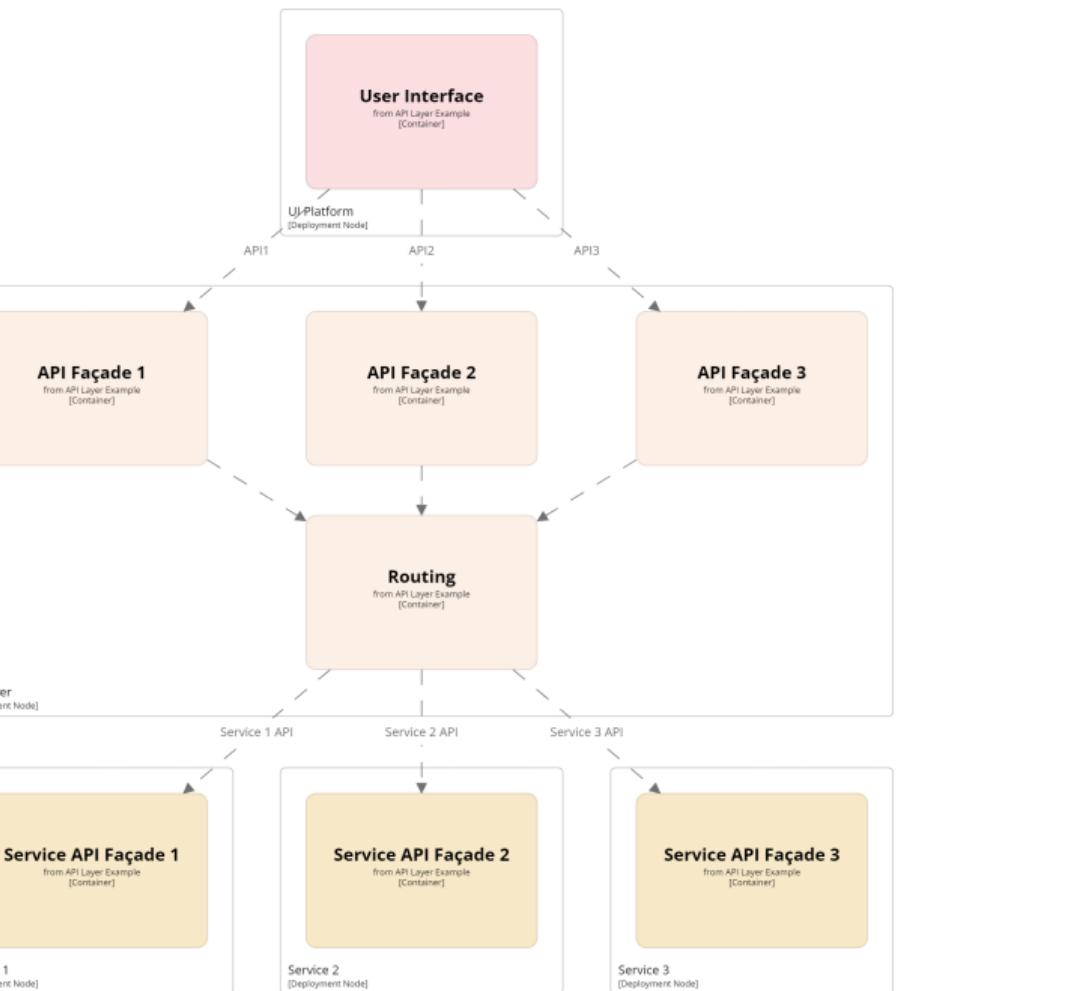
- Some of this can be managed by infrastructure, which requires monitoring systems.
- Some issues are harder to deal with due to coarse service domains.
- Some of this needs to be managed within the application.

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

User Interface
from API Layer Example
[Container]

UI-Platform
[Deployment Node]

API1 · API2 · API3

API Façade 1
from API Layer Example
[Container]

API Façade 2
from API Layer Example
[Container]

API Façade 3
from API Layer Example
[Container]

Routing
from API Layer Example
[Container]

API Layer
[Deployment Node]

Service 1 API · Service 2 API · Service 3 API

Service API Façade 1
from API Layer Example
[Container]

Service API Façade 2
from API Layer Example
[Container]

Service API Façade 3
from API Layer Example
[Container]

Service 1
[Deployment Node]

Service 2
[Deployment Node]

Service 3
[Deployment Node]

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

- **Reverse proxy** hides internal network structure of architecture.
- Can expose different interfaces to external & internal systems.
- Facilitates delivering the security principle of least privilege.
- **Gateway** adds "intelligence" to the reverse proxy.
- Can process requests & responses.
- Orchestrate or aggregate requests / responses to improve performance.
- Translate protocols.

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 🙁