Designing, Evaluating, and Documenting Software Architectures

Mojtaba Shahin

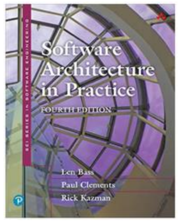
Week 8: Lectorial – Part 1

Content

Acknowledgements

- Most of the **texts** and **images** in the slides come from the following sources:
  - Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.
  - Bass, L., Clements, P., Kazman, R., Software Architecture in Practice, Addison-Wesley, 2021.
  - Brown, Simon. "The C4 model for visualising software architecture" (c4model.com)
  - https://docs.structurizr.com/
  - https://github.com/structurizr
  - https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs
  - https://martinfowler.com/bliki/CQRS.html
  - The University of Queensland's Software Architecture Course Materials, @ Richard Thomas, CC BY-SA 4.0 (https://github.com/CSSE6400)

Attribute-Driven Design (ADD)



- Architecture design is a difficult skill to master
  - Design can (and should) be performed in a systematic way.
  - Design decisions should be justified.
- The architect is accountable for design decisions
- A systematic method provides guidance in performing this complex activity so that it can be learned.

Attribute-Driven Design (ADD)

- Architectural design involves making decisions, and working with the available materials and skills, to satisfy requirements and constraints.
- In architectural design, we turn drivers into structures.

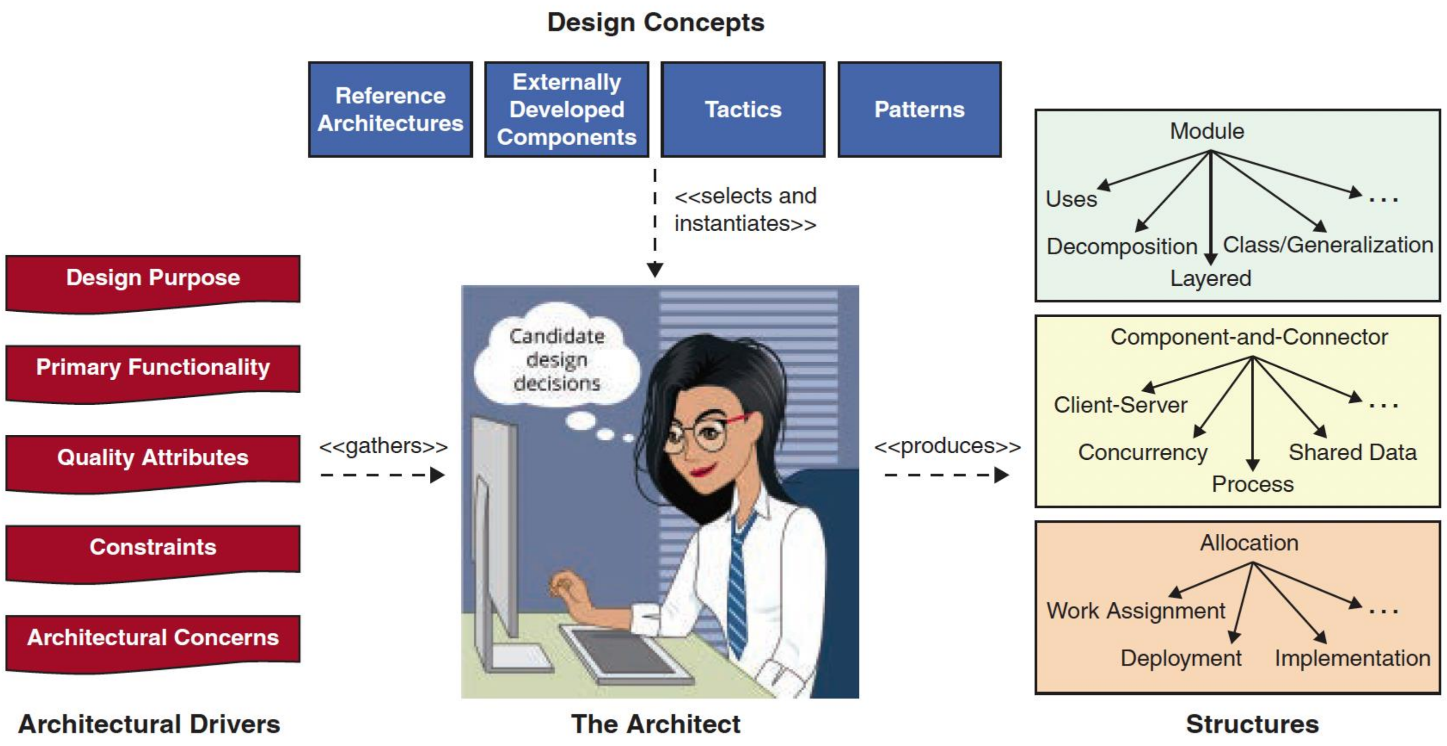


Image Source: Bass, L., Clements, P., Kazman, R., Software Architecture in Practice, Addison-Wesley, 2021.

The Steps of ADD (iterative process)

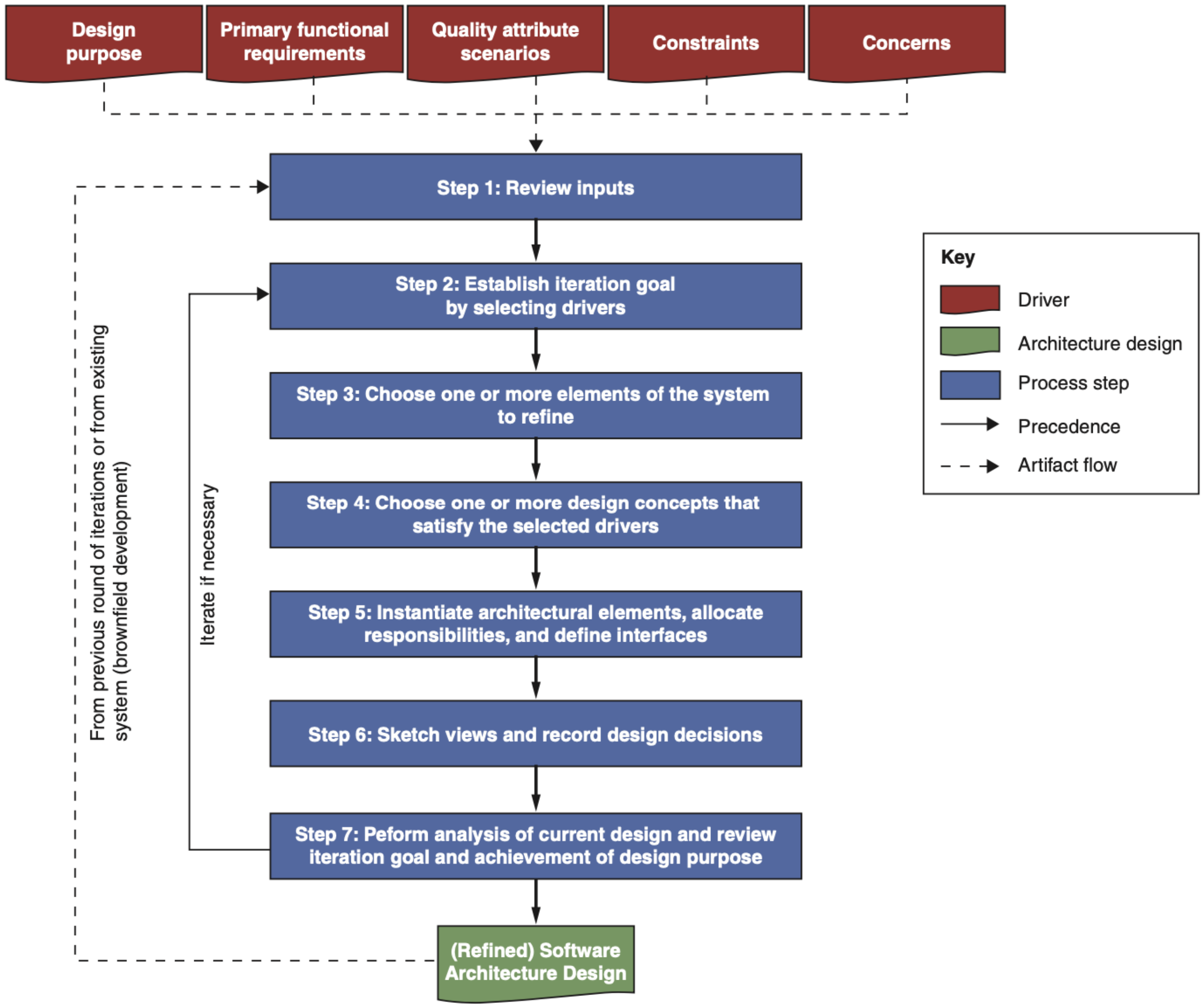
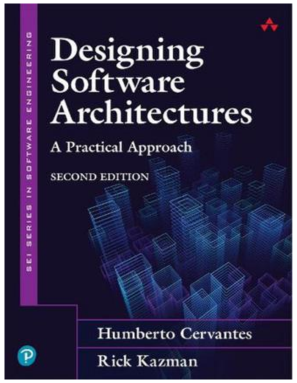
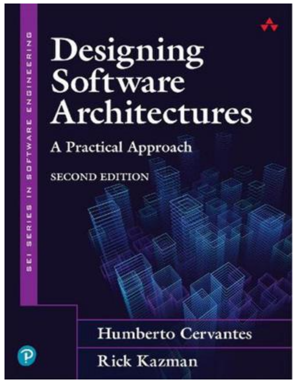

Design purpose
Primary functional requirements
Quality attribute scenarios
Constraints
Concerns
Step 1: Review inputs
Step 2: Establish iteration goal by selecting drivers
Step 3: Choose one or more elements of the system to refine
Step 4: Choose one or more design concepts that satisfy the selected drivers
Step 5: Instantiate architectural elements, allocate responsibilities, and define interfaces
Step 6: Sketch views and record design decisions
Step 7: Peform analysis of current design and review iteration goal and achievement of design purpose
(Refined) Software Architecture Design
From previous round of iterations or from existing system (brownfield development)
Iterate if necessary
Key
Driver
Architecture design
Process step
Precedence
Artifact flow

Image Source: Bass, L., Clements, P., Kazman, R., Software Architecture in Practice, Addison-Wesley, 2021.


SEI SERIES IN SOFTWARE ENGINEERING
Designing Software Architectures
A Practical Approach
SECOND EDITION
Humberto Cervantes
Rick Kazman

A Case Study

Design the Architecture of the Hotel Pricing System with ADD


Designing Software Architectures
A Practical Approach
SECOND EDITION
SEI SERIES IN SOFTWARE ENGINEERING
Humberto Cervantes
Rick Kazman

Business Concerns and Requirements

Business Case

- AD&D Hotels is a mid-sized business hotel chain (currently around 300 hotels) that has been experiencing robust growth in recent years.
- The IT infrastructure of the company is composed of many different applications, such as a **Property Management System**, a **Commercial Analysis System**, an **Enterprise Reservation System**, and a **Channel Management System**.
- At the centre of this system of systems is the **Hotel Pricing System.**

Business Case

- The Hotel Pricing System (HPS) is used by **sales managers** and **commercial representatives** to establish prices for rooms on specific dates for the different hotels in the company.
- Prices are associated with different rates (e.g., a public rate, a discount rate), and most of the prices for the different rates are calculated by taking a base rate and applying business rules to it (although some of the rates can also be fixed and not depend on the base rate).
- Prices that are calculated by the HPS are used by other systems in the company to make reservations and are also sent to different online travel agencies through the Channel Management System (CMS).

Context diagram for the Hotel Pricing System

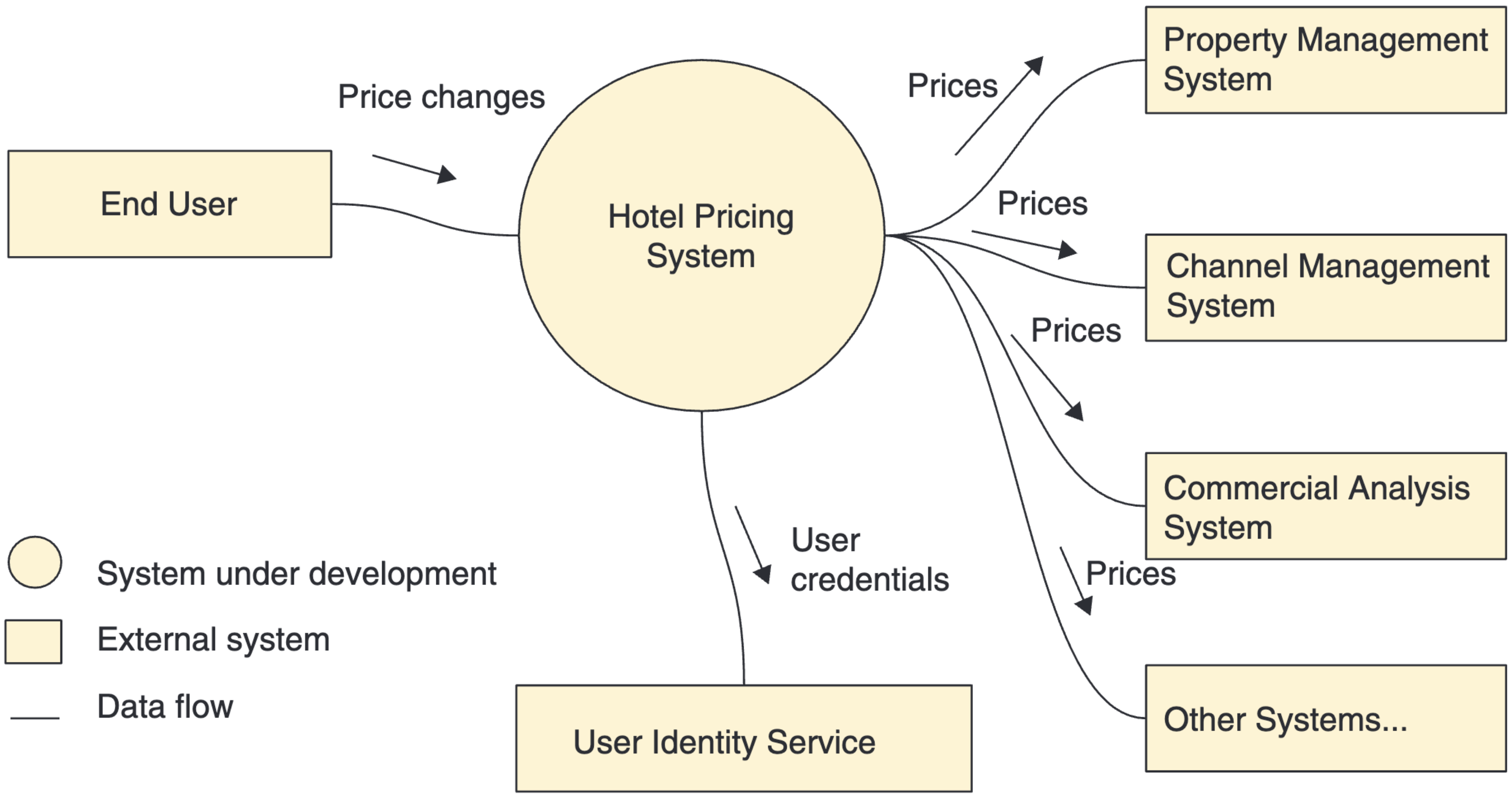
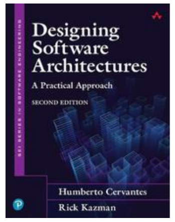


Image Source: Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.

Goal and Existing Issues

- The goal is the **complete replacement** of the existing **pricing system**
  - The existing pricing system was developed several years ago and is suffering from **reliability**, **performance**, **availability**, and **maintainability** issues.
  - The company has experienced difficulties because many of its systems are connected using **traditional SOAP** and **REST request–response endpoints**:
    - Changes to one application frequently impact other applications and complicate the deployment of individual updates to specific applications.
    - The failure of an application can **propagate** through the entire system.
  - Some of the applications interact using what are now well-known anti-patterns, such as integration through a shared database.
- So, the goal is to migrate **HPS** toward a **more decoupled model**.

Initial use case diagram for the Hotel Pricing System



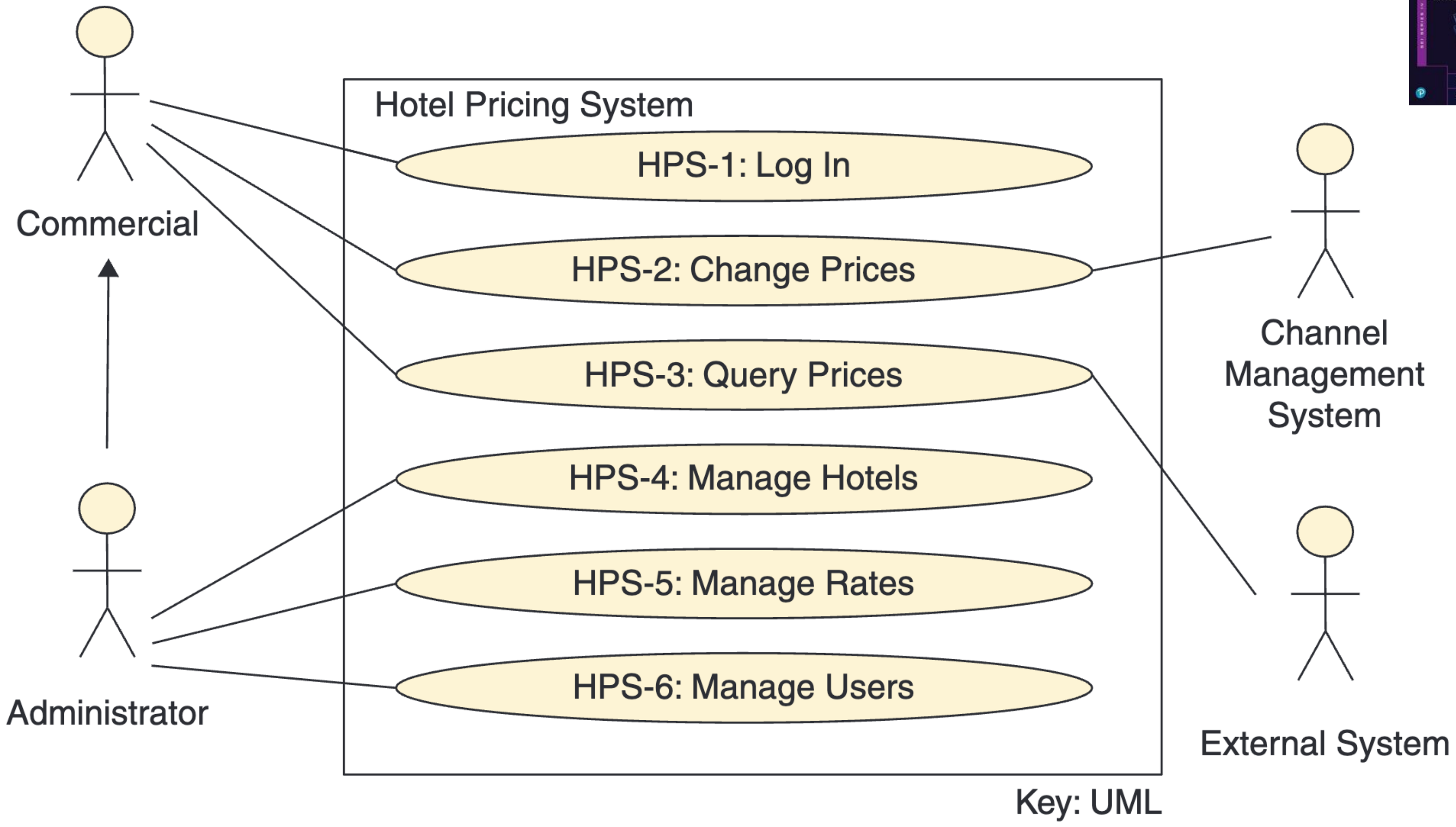
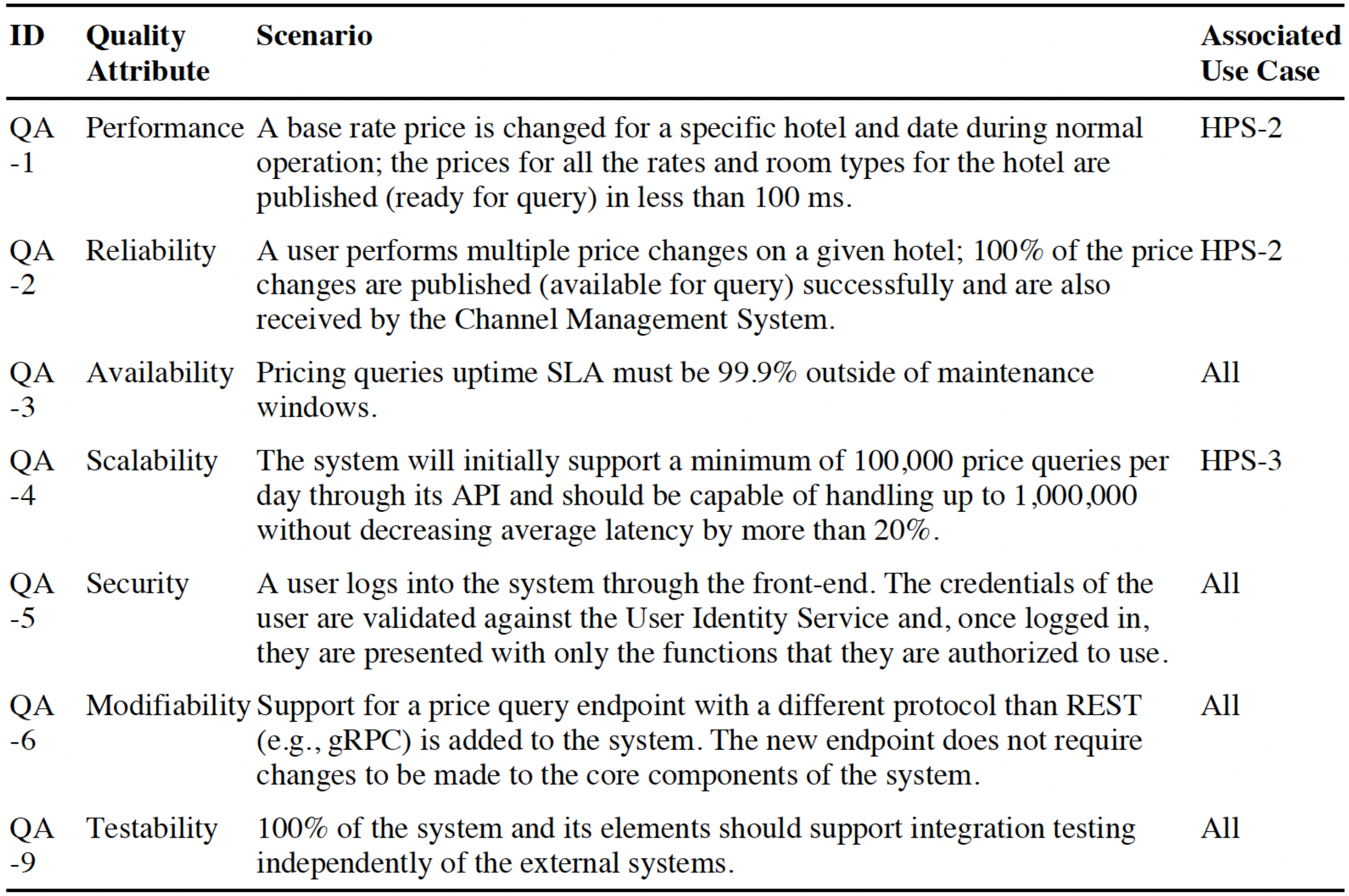


Image Source: Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.

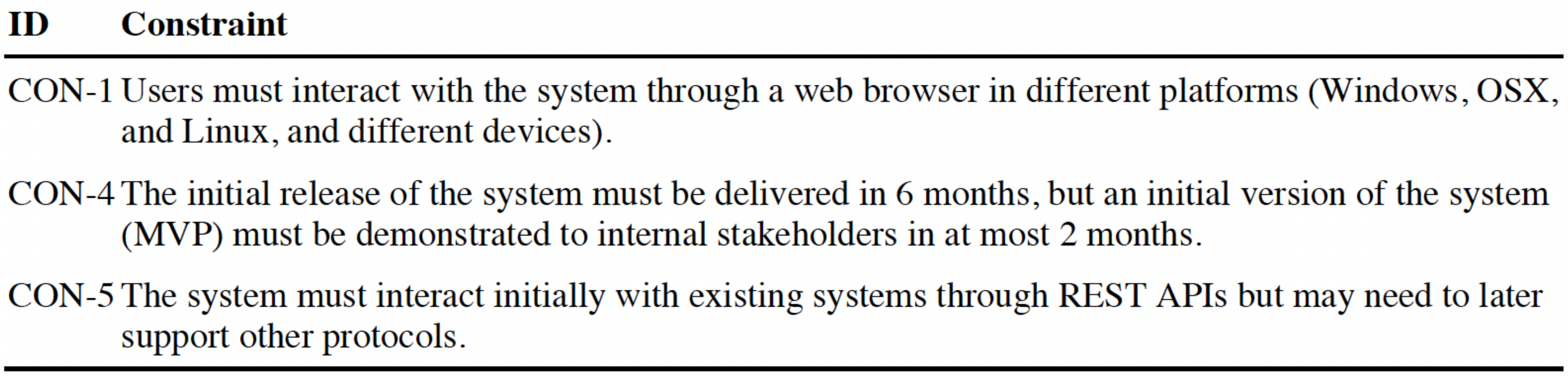
Quality Attribute Scenarios



ID	Quality Attribute	Scenario	Associated Use Case
QA -1	Performance	A base rate price is changed for a specific hotel and date during normal operation; the prices for all the rates and room types for the hotel are published (ready for query) in less than 100 ms.	HPS-2
QA -2	Reliability	A user performs multiple price changes on a given hotel; 100% of the price changes are published (available for query) successfully and are also received by the Channel Management System.	HPS-2
QA -3	Availability	Pricing queries uptime SLA must be 99.9% outside of maintenance windows.	All
QA -4	Scalability	The system will initially support a minimum of 100,000 price queries per day through its API and should be capable of handling up to 1,000,000 without decreasing average latency by more than 20%.	HPS-3
QA -5	Security	A user logs into the system through the front-end. The credentials of the user are validated against the User Identity Service and, once logged in, they are presented with only the functions that they are authorized to use.	All
QA -6	Modifiability	Support for a price query endpoint with a different protocol than REST (e.g., gRPC) is added to the system. The new endpoint does not require changes to be made to the core components of the system.	All
QA -9	Testability	100% of the system and its elements should support integration testing independently of the external systems.	All

Image Source: Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.

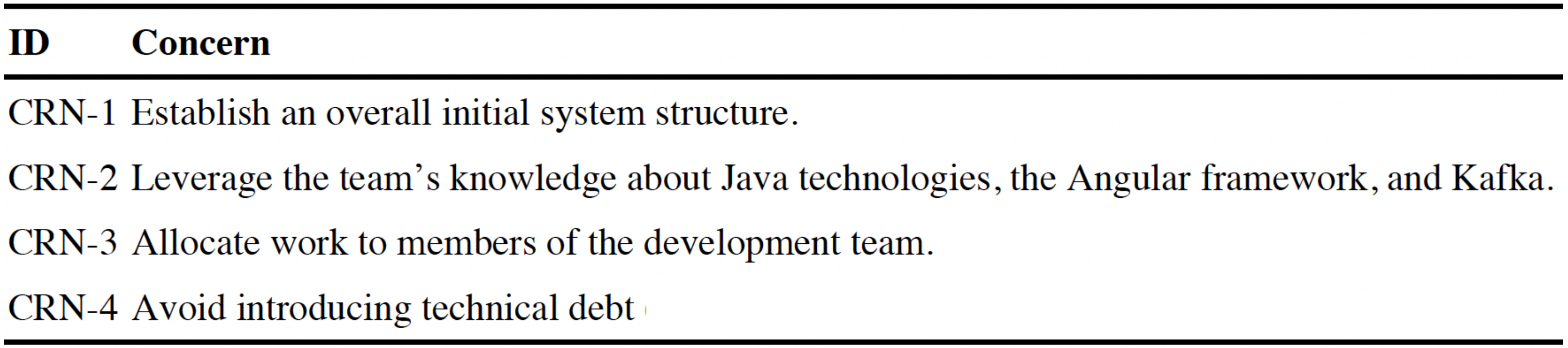
Constraints



ID	Constraint
CON-1	Users must interact with the system through a web browser in different platforms (Windows, OSX, and Linux, and different devices).
CON-4	The initial release of the system must be delivered in 6 months, but an initial version of the system (MVP) must be demonstrated to internal stakeholders in at most 2 months.
CON-5	The system must interact initially with existing systems through REST APIs but may need to later support other protocols.

Image Source: Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.

Architectural Concerns



ID	Concern
CRN-1	Establish an overall initial system structure.
CRN-2	Leverage the team's knowledge about Java technologies, the Angular framework, and Kafka.
CRN-3	Allocate work to members of the development team.
CRN-4	Avoid introducing technical debt

Image Source: Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.

Development and Operations Requirements

- As part of the modernization effort, AD&D Hotels wants to integrate agile (specifically Scrum) and DevOps practices into the development of the HPS.
- Besides the previously discussed system requirements, other development and operational requirements need to be taken into account.
- Artifacts in the development process move through four different environments

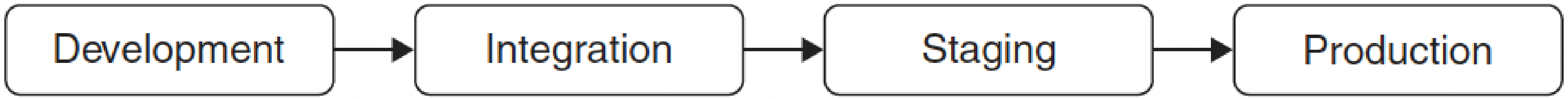
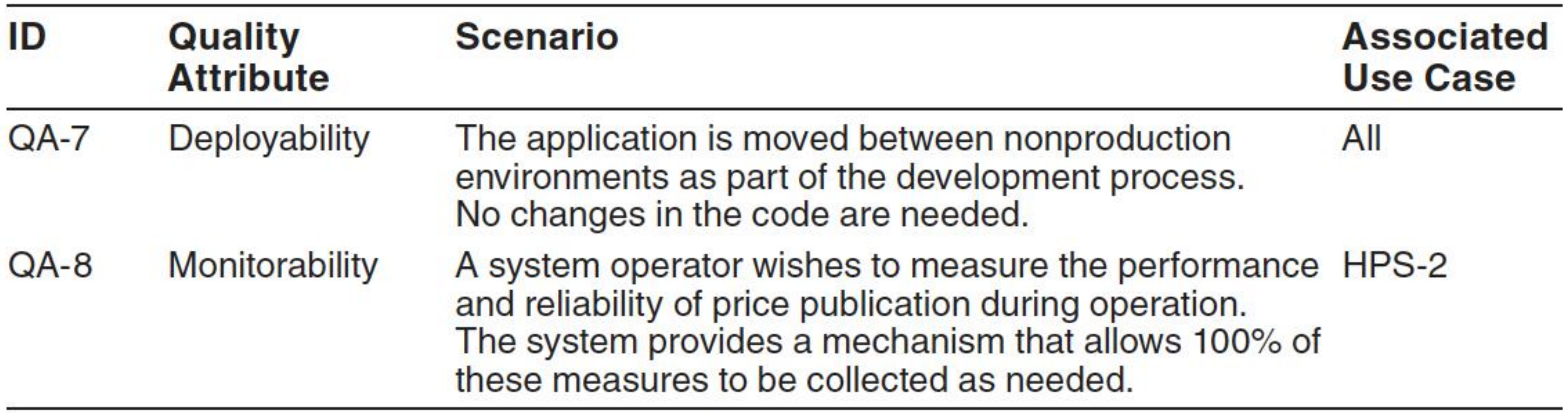


Image Source: Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.

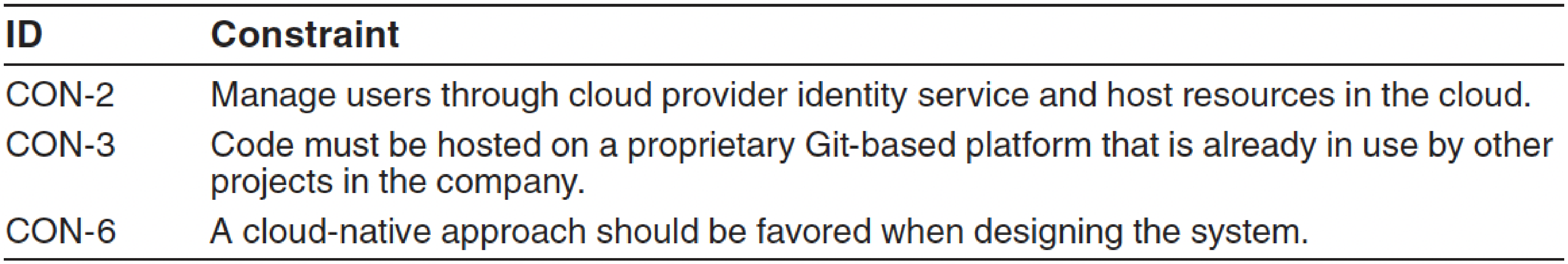
Development and Operations QA Scenarios for HPS



ID	Quality Attribute	Scenario	Associated Use Case
QA-7	Deployability	The application is moved between nonproduction environments as part of the development process. No changes in the code are needed.	All
QA-8	Monitorability	A system operator wishes to measure the performance and reliability of price publication during operation. The system provides a mechanism that allows 100% of these measures to be collected as needed.	HPS-2

Image Source: Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.

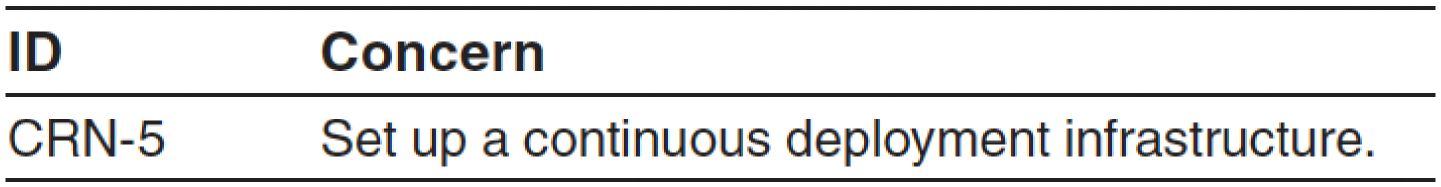
Development and Operations Constraints for HPS



ID	Constraint
CON-2	Manage users through cloud provider identity service and host resources in the cloud.
CON-3	Code must be hosted on a proprietary Git-based platform that is already in use by other projects in the company.
CON-6	A cloud-native approach should be favored when designing the system.

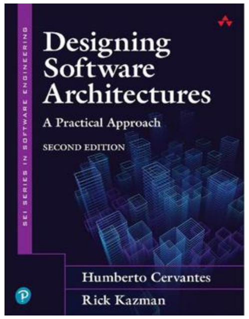
Image Source: Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.

Development and Operations Architectural Concerns for HPS



ID	Concern
CRN-5	Set up a continuous deployment infrastructure.

Image Source: Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.


Designing Software Architectures
A Practical Approach
SECOND EDITION
Humberto Cervantes
Rick Kazman
SEI SERIES IN SOFTWARE ENGINEERING

The Software Design Process using ADD

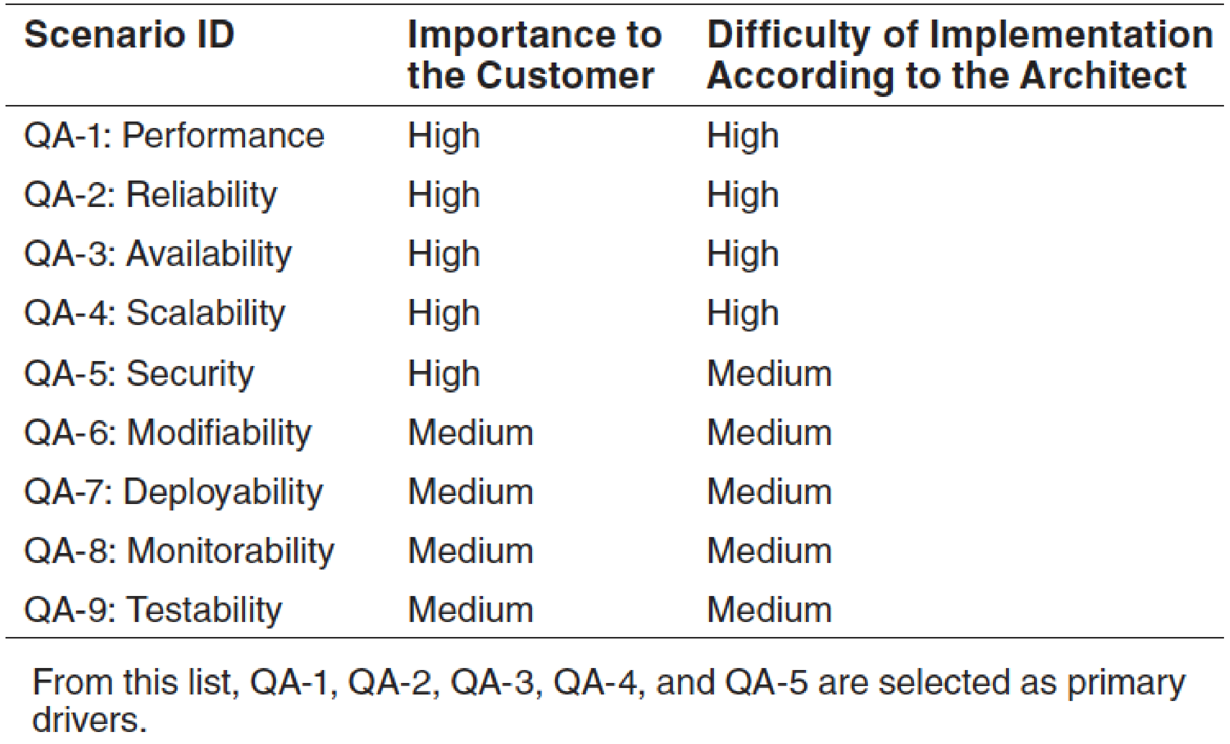
From the world of requirements and business concerns to the world of design.

ADD Step 1. Review Inputs

Category	Details
Design purpose	This project can be considered greenfield development, as it involves the complete replacement of an existing system. The purpose of the design activity is to make initial decisions to support the construction of the system from scratch.
Primary functional requirements	From the listed user stories, the primary ones were determined to be: ■ **HPS-2**: Change Prices—Because it directly supports the core business ■ **HPS-3**: Query Prices—Because it directly supports the core business ■ **HPS-4**: Manage Hotels—Because it establishes a basis for many other user stories
Constraints	All of the previously discussed constraints are included as drivers.
Architectural concerns	All of the previously discussed architectural concerns associated with the system are included as drivers.

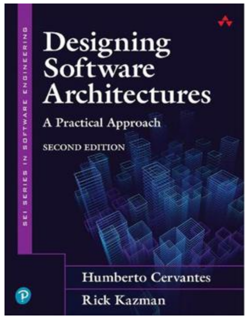
ADD Step 1. Review Inputs

Category	Details
Quality attribute scenarios	The scenarios for the HPS have now been prioritized.



Scenario ID	Importance to the Customer	Difficulty of Implementation According to the Architect
QA-1: Performance	High	High
QA-2: Reliability	High	High
QA-3: Availability	High	High
QA-4: Scalability	High	High
QA-5: Security	High	Medium
QA-6: Modifiability	Medium	Medium
QA-7: Deployability	Medium	Medium
QA-8: Monitorability	Medium	Medium
QA-9: Testability	Medium	Medium

From this list, QA-1, QA-2, QA-3, QA-4, and QA-5 are selected as primary drivers.



Iteration 1: Establishing an Overall System Structure

The steps of ADD in the first iteration of the design process

Step 2. Establish Iteration Goal by Selecting Drivers

- This is the first iteration in the design of a greenfield system, so the iteration goal is to achieve concern **CRN-1**, "*establishing an overall system structure*".
- The architect must also be mindful of the following:
  - **QA-1**: Performance
  - **QA-2**: Reliability
  - **QA-3**: Availability
  - **QA-5**: Security
  - **CON-1**: Access to the system via web browser ...
  - **CON-2**: Cloud provider ...
  - **CON-6**: Develop cloud-native solution ...
  - **CRN-2**: Leverage team's knowledge ...

Step 3. Choose Elements of the System to Refine

- Since this project involves the complete replacement of an existing system, the first element to refine is the **whole HPS system**. In this case, refinement is performed through **decomposition**.

Step 4. Choose Design Concepts That Satisfy the Selected Drivers

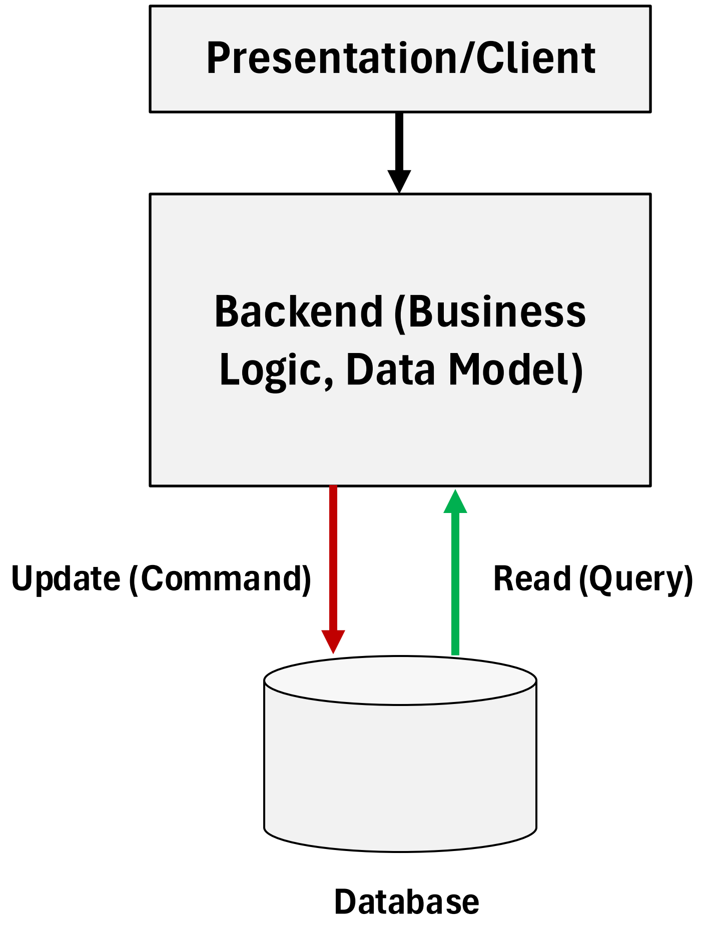
Decision. Structure the back-end part of the system following the **CQRS pattern** and separate the command and query components that communicate.

Rationale: CQRS (Command and Query Responsibility Segregation) is a pattern that separates changes to data from queries to this data.
This pattern is useful to support performance, scalability, availability, and security, albeit at the cost of added complexity. The command and query components communicate using **events**. This type of communication using events also supports the decision by the company to move away from connecting applications using traditional request-response invocations. Furthermore, this will **allow other applications to connect** to the event channel in the future to perform tasks such as analysis or event storage.

Alternatives: Discarded alternatives include developing the back-end as a monolithic application.

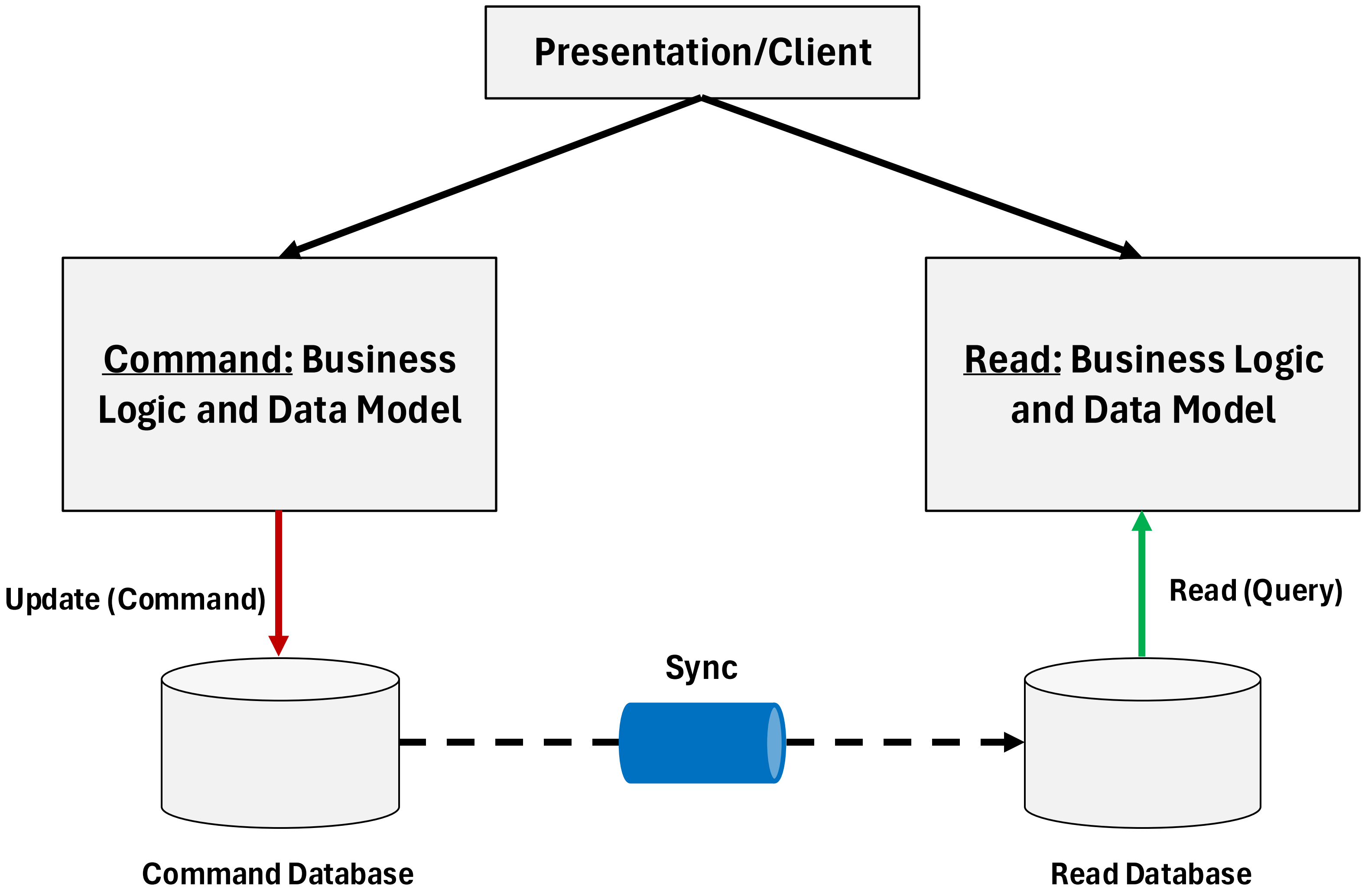
Traditional architecture

In a traditional architecture, a single data model/source is often used for both read and write operations.



Source: https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs

CQRS pattern



Source: https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs

Step 4. Choose Design Concepts That Satisfy the Selected Drivers

Decision. Implement the command and query components as microservices.

CRN-3 Allocate work to members of the development team.

Rationale: The command and query parts of the system are developed and deployed as independent microservices that expose service APIs. Multiple replicas of the query side can be set up to support larger query volumes and higher availability. Security is enhanced because clients accessing the query side cannot make changes to important aspects of the system such as business rule calculations. The use of microservices also allows **CRN-3** to be addressed, as each microservice can be implemented by a small team.

Alternatives: N/A.

Step 5. Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

Decision. Use **Apache Kafka** as the event bus.

Rationale: Kafka is a durable message broker that enables applications to process, persist, and reprocess streamed data. Events are published and consumed from "topics". **Kafka supports message ordering through the use of keys, which is necessary because changes to the hotel prices for a given day must be ordered**. Kafka also has the benefit of retaining all of the messages in a log, which acts as the "source of truth" for prices.

CRN-2 Leverage the team's knowledge about Java technologies, the Angular framework, and Kafka.

Alternatives: Discarded alternatives are RabbitMQ and other messaging alternatives that are less familiar to the team (**CRN-2**).

Step 5. Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

Decision. Create an additional **microservice** to handle price change events and deliver them to the Channel Management System.

Rationale: **The Channel Management System is a legacy application that cannot be changed to listen to the CQRS events**. Instead, an additional microservice is created so that when it receives events from the command side, these events are then pushed to the CMS. This microservice will not require a database, because messages are pulled from the log (Kafka) and sent to the CMS. In case the CMS is not available, messages are put back in the log and delivery is retried.

Alternatives: N/A.

Step 5. Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

Decision. Use **Docker containers** to promote portability across environments.

Rationale: Because the application has to be moved across environments (**QA-7**), deploying the application as a set of container images is preferred. Each microservice is deployed in a container. This ensures that the application will run in the same execution environment from development to production.

QA-7. Deployability: The application is moved between non - production environments as part of the development process. No changes in the code are needed.

Alternatives: A discarded alternative is the use of virtual machines, as the size of VM images is typically considerably larger than container images; also VMs require more resources (memory, CPU) than containers..

Step 5. Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

Decision. Use an **API gateway** to limit access to the microservices that expose the APIs.

Rationale: Access to the microservices is not performed directly but rather through an API gateway, which can perform various functions including securing and monitoring the APIs. The calls that are received by the API are routed to the microservices, which are protected inside a private network in the cloud.

Alternatives: N/A.

Step 6. Sketch Views and Record Design Decisions

Main elements of the Hotel Pricing System

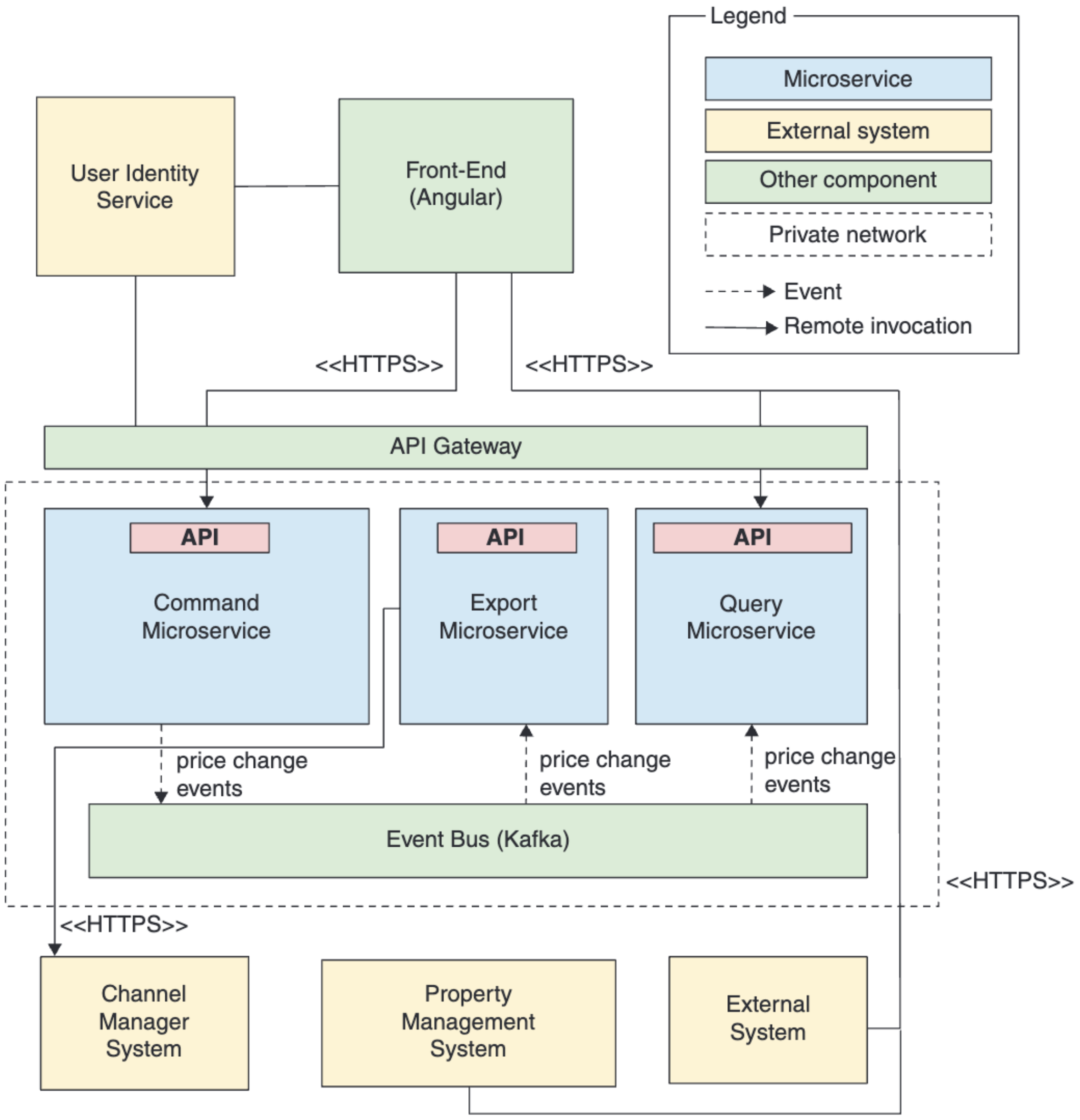


Image Source: Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.

Step 6. Sketch Views and Record Design Decisions

This microservice encapsulates the logic for the command part of the CQRS pattern. This includes managing the different hotels, rates, room types, and price calculation business rules.

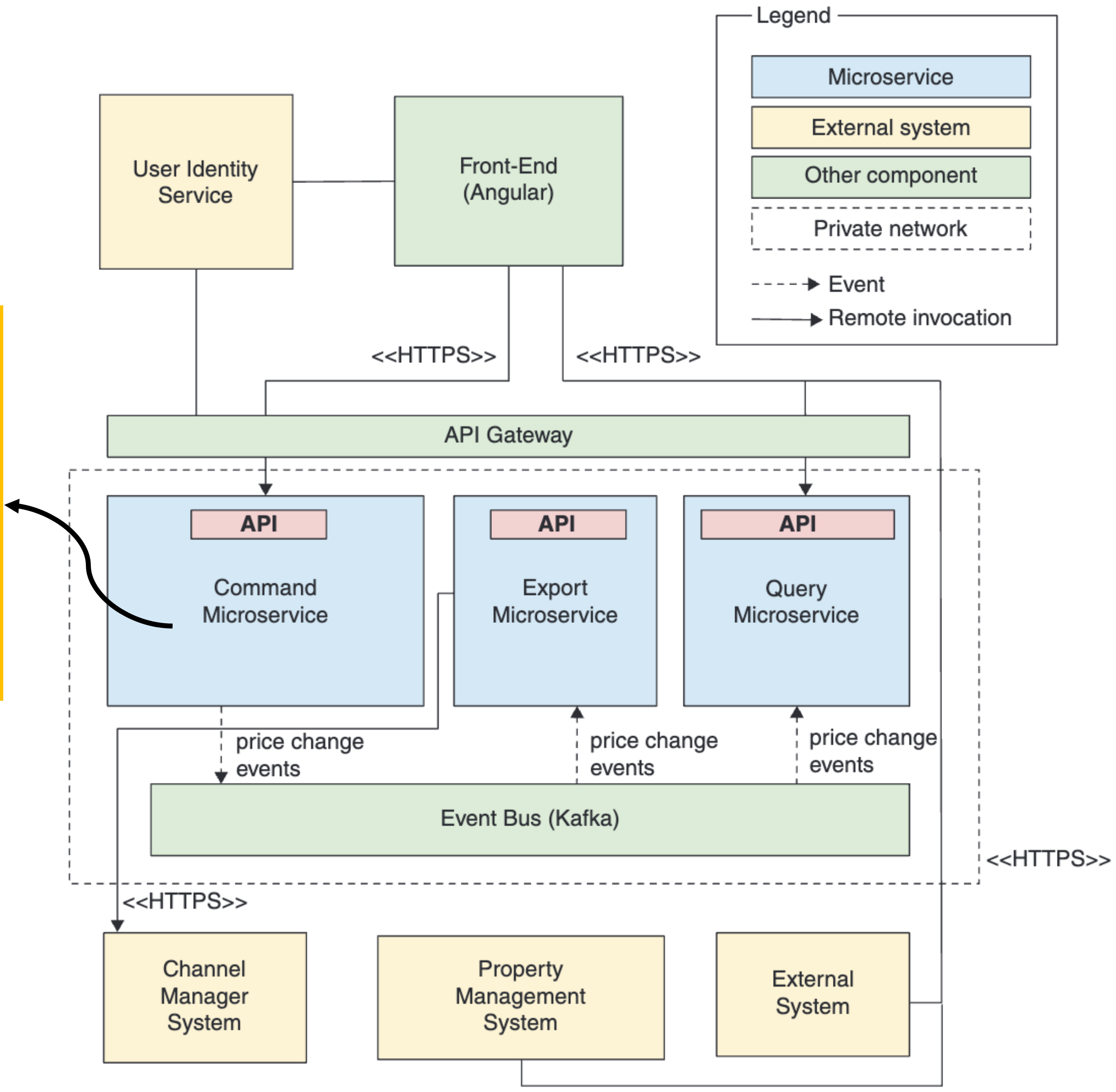


Image Source: Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.

Step 6. Sketch Views and Record Design Decisions

This microservice's primary responsibility is to serve price requests. It exposes two APIs: one used by the front-end and one used by legacy internal systems.

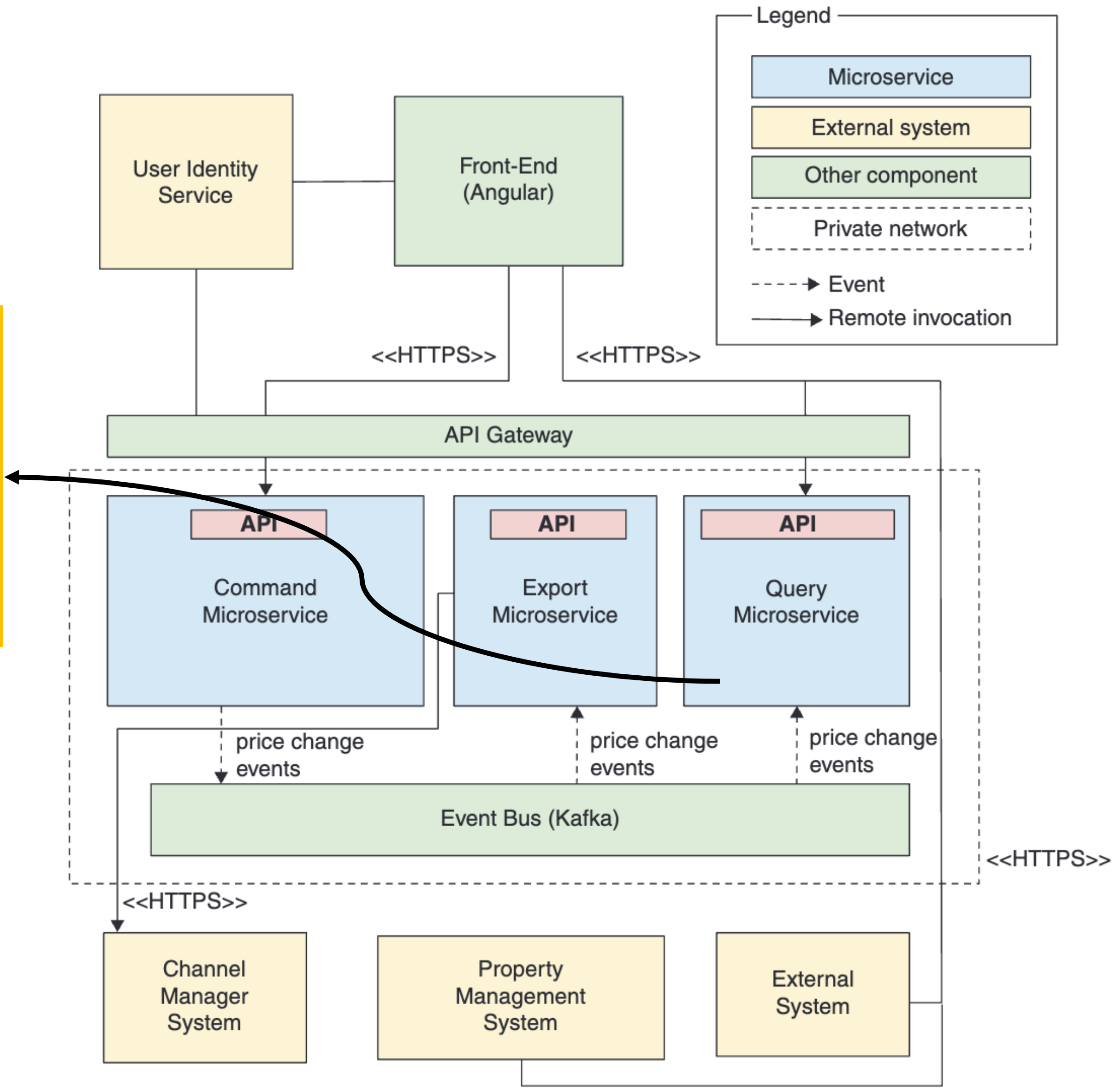


Image Source: Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.

Step 6. Sketch Views and Record Design Decisions

This microservice's primary responsibility is to consume price change events and publish them on the Channel Management System.

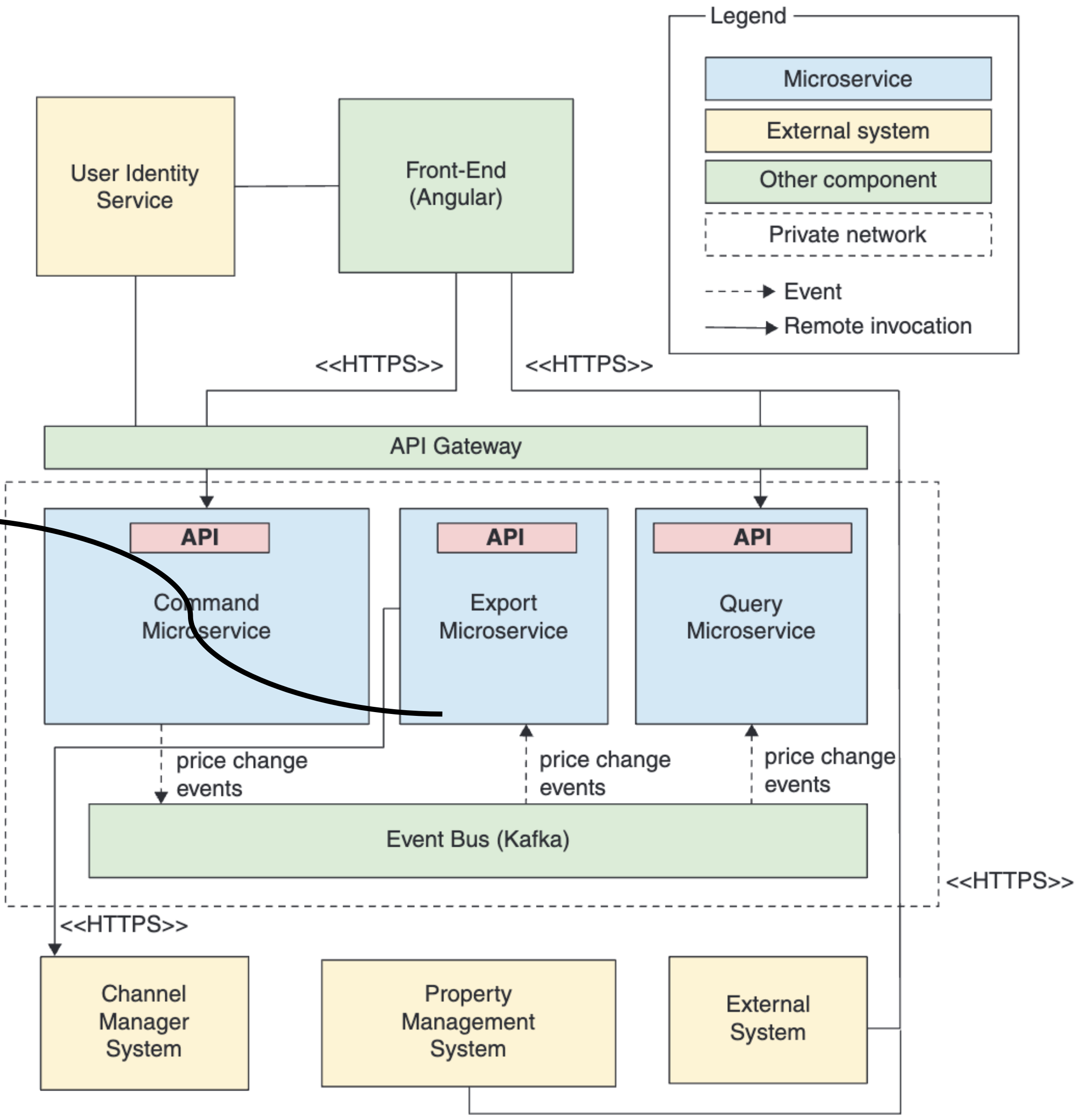


Image Source: Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.

Step 6. Sketch Views and Record Design Decisions

This event bus is used by the Command microservice to publish price change events, and by the Query and Export microservices to consume these events.

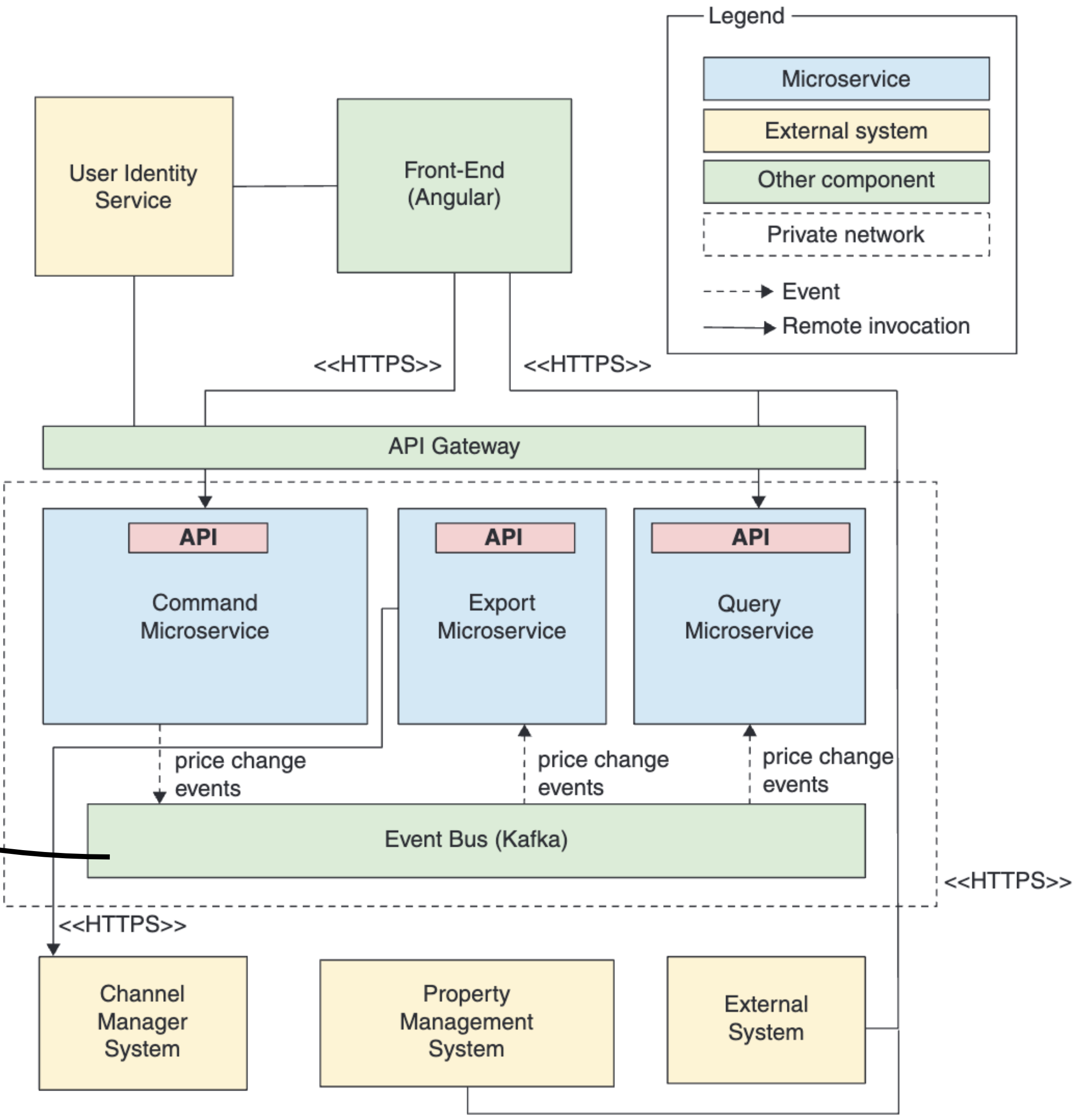
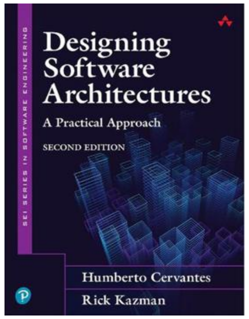


Image Source: Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.

Moving from the abstract view to more detailed decisions.



Iteration 2: Identifying Structures to Support Primary Functionality

The steps of ADD in the second iteration of the design process

\

Step 2. Establish Iteration Goal by Selecting Drivers

- The goal of this iteration is to "*identify structures to support primary functionality*".
- In this second iteration, the architect considers the system's primary user stories:
  - **HPS-2**: Change prices
  - **HPS-3**: Query prices
- Note that in this iteration, the design is focused on the back-end part of the application.

Step 3. Choose Elements of the System to Refine

- The elements that will be refined in this iteration are the Command, Query, and Export microservices derived from the CQRS pattern used in the first iteration.

Step 4. Choose Design Concepts That Satisfy the Selected Drivers

Decision. Logically structure the microservices using the **Service Application** reference architecture. Microservices expose APIs.

Rationale: Service applications do not provide a user interface but rather expose services that are consumed by other applications. This reference architecture is suitable to logically structure each microservice that exposes an API. These APIs will be consumed by the client application and by other legacy systems (**CON-5**).

Alternatives: N/A.

Step 5. Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

Decision. Use a **relational database** for the Command microservice.

Rationale: Given that the Command microservice manages the entities of the domain model (e.g., hotel prices, rates, room types), and given that these entities are related to each other, the most natural and appropriate mechanism for storing them is a relational database.

Alternatives: Discarded alternatives are nonrelational databases and other types of storage.

Step 5. Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

Decision. Use a **non-relational database** for the Query microservice.

Rationale: Because the database on the query side does not need to handle transactions and relationships between entities, and because the prices that are stored have a variable structure depending on the hotel, it was decided that the use of a nonrelational database is more appropriate. PriceChangeEvents can be appropriately stored in a document database.

Alternatives: Discarded alternatives include other types of NoSQL, relational databases, and other types of storage.

Step 5. Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

Decision. Use the **Spring Framework technology stack** to implement the microservices and map the system user stories to the Service Application reference architecture modules.

Rationale: The Spring Framework technology stack is selected because the **team is familiar with it** (**CRN-2**). Because of this, no alternative frameworks for languages other than Java are considered. The Service Application reference architecture and Spring Framework provide categories of modules for the different layers: **controllers to expose APIs**, **services to manage business logic**, **entities to manage business entities**, and **repositories to persist them**, among others.

Alternatives: N/A.

Step 6. Sketch Views and Record Design Decisions

Module view of the query-side microservice

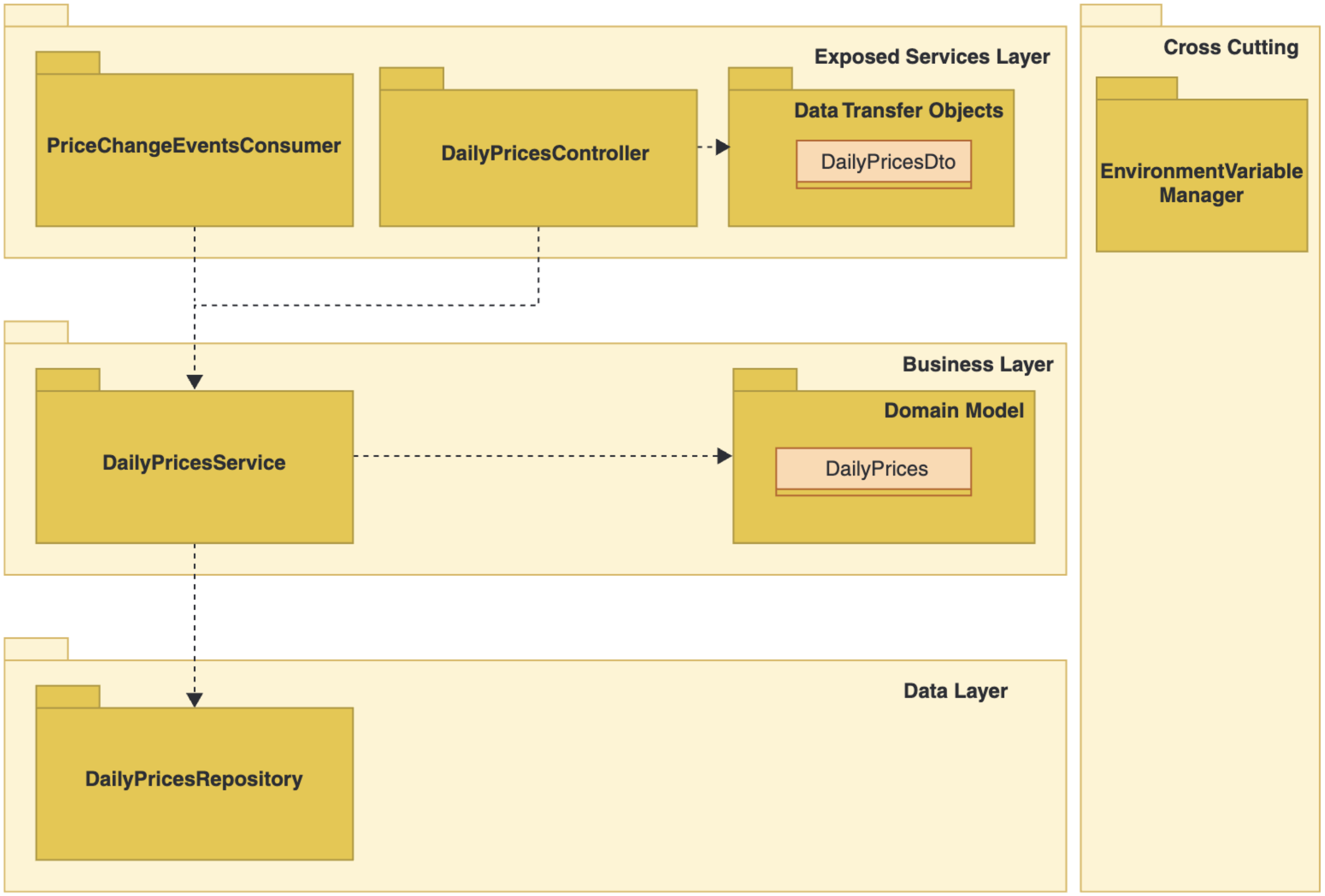


Image Source: Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.

Step 6. Sketch Views and Record Design Decisions

Initial deployment diagram of the system (Key: UML)

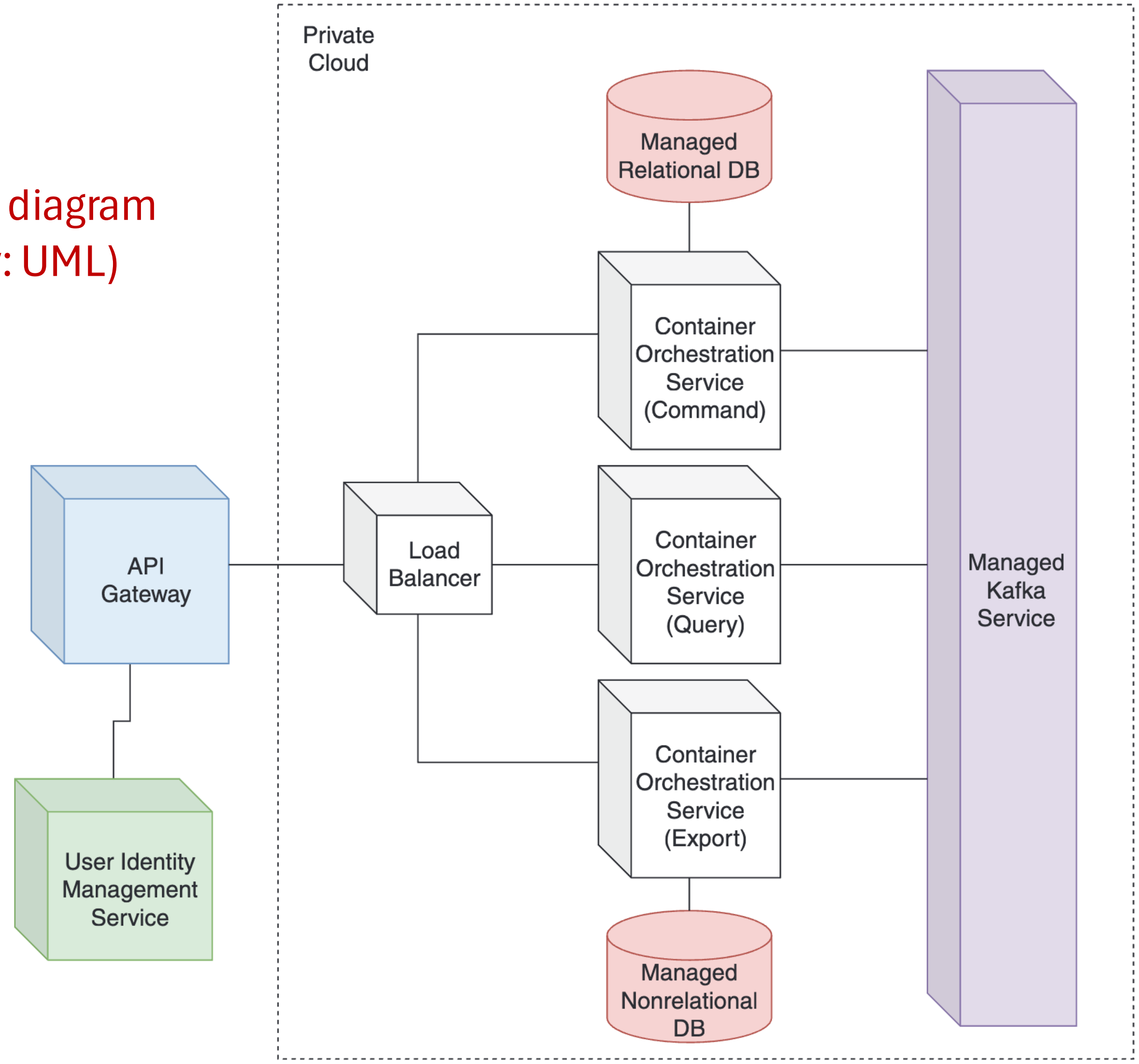

Private Cloud
Managed Relational DB
Container Orchestration Service (Command)
API Gateway
Load Balancer
Container Orchestration Service (Query)
Managed Kafka Service
Container Orchestration Service (Export)
User Identity Management Service
Managed Nonrelational DB

Image Source: Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.

References

- Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.
- Bass, L., Clements, P., Kazman, R., Software Architecture in Practice, Addison-Wesley, 2021.
- Brown, Simon. "The C4 model for visualising software architecture" (c4model.com)
- https://docs.structurizr.com/
- https://github.com/structurizr
- https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs
- https://martinfowler.com/bliki/CQRS.html
- The University of Queensland's Software Architecture Course Materials, @ Richard Thomas, CC BY-SA 4.0 (https://github.com/CSSE6400)