Communication Styles and Risk Analysis

Mojtaba Shahin

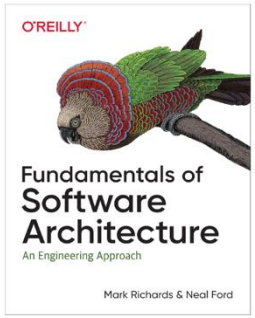
Week 9: Lectorial – Part 2

Content

- Part 1
  - Communication Styles
- Part 2
  - Architecture Risk Analysis

Acknowledgements

- Most of the **texts** and **images** in the slides come from the following sources:
  - Pautasso, C., Software Architecture- Visual Lecture Notes, LeanPub, 2023 (https://leanpub.com/software-architecture/)
  - Bass, L., Clements, P., Kazman, R., Software Architecture in Practice, Addison-Wesley, 2021.
  - Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020 (First Edition).
  - Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2025 (Second Edition).
  - Gandhi, R., Richards, M., Ford, N., Head First Software Architecture, O'Reilly Media, Inc. 2024
  - Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.
  - Introduction to gRPC (https://grpc.io/docs/what-is-grpc/introduction/)
  - Building a GraphQL service (https://spring.io/guides/gs/graphql-server)
  - What Is a REST API? Examples, Uses, and Challenges (https://blog.postman.com/rest-api-examples/)
  - Richard N. Taylor, Nenad Medvidovic, Eric M. Dashofy, Software Architecture: Foundations, Theory and Practice, John-Wiley, January 2009, ISBN 978047016774
    - https://www.softwarearchitecturebook.com/resources/



Analyzing Architecture Risk (Chapter 20)

Architecture Risk

- Every architecture has risks associated with it, whether it be risks involving availability, scalability, or data integrity.
- Analyzing architecture risk is one of the key activities of architecture.
- By continually analyzing risk, the architect can address deficiencies within the architecture and take corrective action to mitigate the risk.

Risk Matrix

- The architecture risk matrix uses two dimensions to qualify risk:
  - the overall impact of the risk
  - the likelihood of that risk occurring.
  - Numbers 1 and 2 are considered low risk (green), numbers 3 and 4 are considered medium risk (yellow), and numbers 6 through 9 are considered high risk (red).

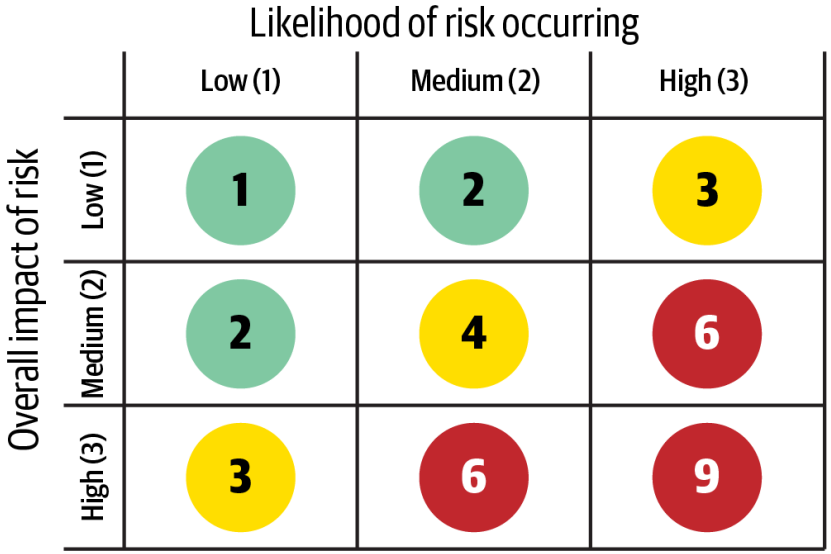


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Example

- **Concern**: Availability of a primary central database used in the application.
- The **Impact** dimension—what is the overall impact if the database goes down or becomes unavailable?
  - Here, an architect might deem that **high** risk
- The **likelihood of risk occurring** dimension –
  - The architect realizes that the database is on highly available servers in a clustered configuration, so the likelihood is **low** that the database would become unavailable.

The intersection between the high impact and low likelihood gives an overall risk rating of 3 (medium risk).

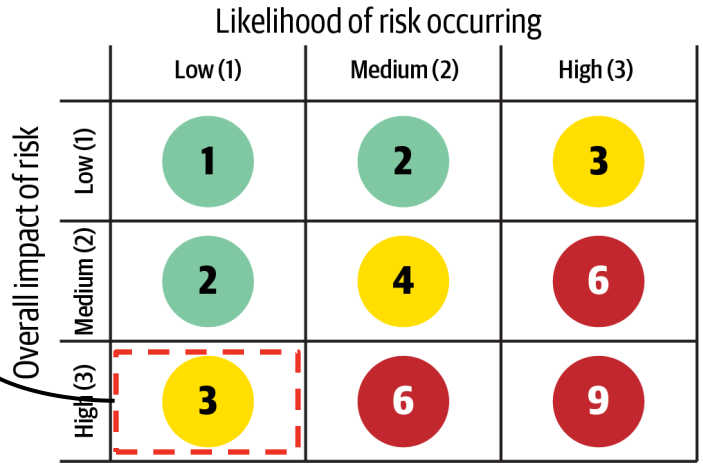


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Risk Assessment

- The **risk matrix** can be used to build what is called a **risk assessment**.
- A risk assessment is a summarized report of the **overall risk of an architecture** with respect to some sort of contextual and meaningful assessment criteria.
- Risk assessments can vary greatly, but in general they contain the risk (qualified from the risk matrix) of some assessment criteria based on services or domain areas of an application.

RISK CRITERIA	Customer registration	Catalog checkout	Order fulfillment	Order shipment	TOTAL RISK
Scalability	2	6	1	2	11
Availability	3	4	2	1	10
Performance	4	2	3	6	15
Security	6	3	1	1	11
Data integrity	9	6	1	1	17
TOTAL RISK	24	21	8	11	

Service or Domain Area

Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Risk Assessment

(1-2) is low risk, medium gray
(3-4) is medium risk, and dark gray (6-9) is high risk.

RISK CRITERIA	Customer registration	Catalog checkout	Order fulfillment	Order shipment	TOTAL RISK
Scalability	2	6	1	2	11
Availability	3	4	2	1	10
Performance	4	2	3	6	15
Security	6	3	1	1	11
Data integrity	9	6	1	1	17
TOTAL RISK	24	21	8	11	

The accumulated risk for **data integrity**

The accumulated risk for **Customer Registration**

Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Risk Storming

- The risk storming effort involves both an individual part and a collaborative part.
- Risk storming is broken down into three primary activities:

1. Identification (Individual)

- All participants individually (without collaboration) assign risk to areas of the architecture using the risk matrix.

2. Consensus (Collaborative)

- all participants work together to gain consensus on risk areas.

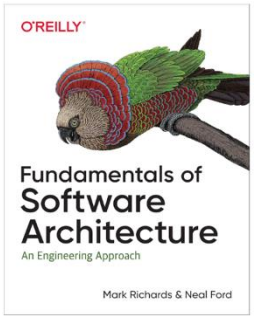
3. Mitigation (Collaborative)

- Once all participants reach a consensus, the final and most important activity occurs—risk mitigation. Mitigating risk within an architecture usually involves changes or enhancements to certain areas of the architecture.

Risk Storming

- Risk storming is a collaborative exercise used to determine architectural risk within a specific dimension.
- While most risk storming efforts involve multiple architects, it is wise to include senior developers and tech leads as well.
  - Not only will they provide an implementation perspective on the architectural risk, but involving developers will help them gain a better understanding of the architecture.

Initial identification of risk areas



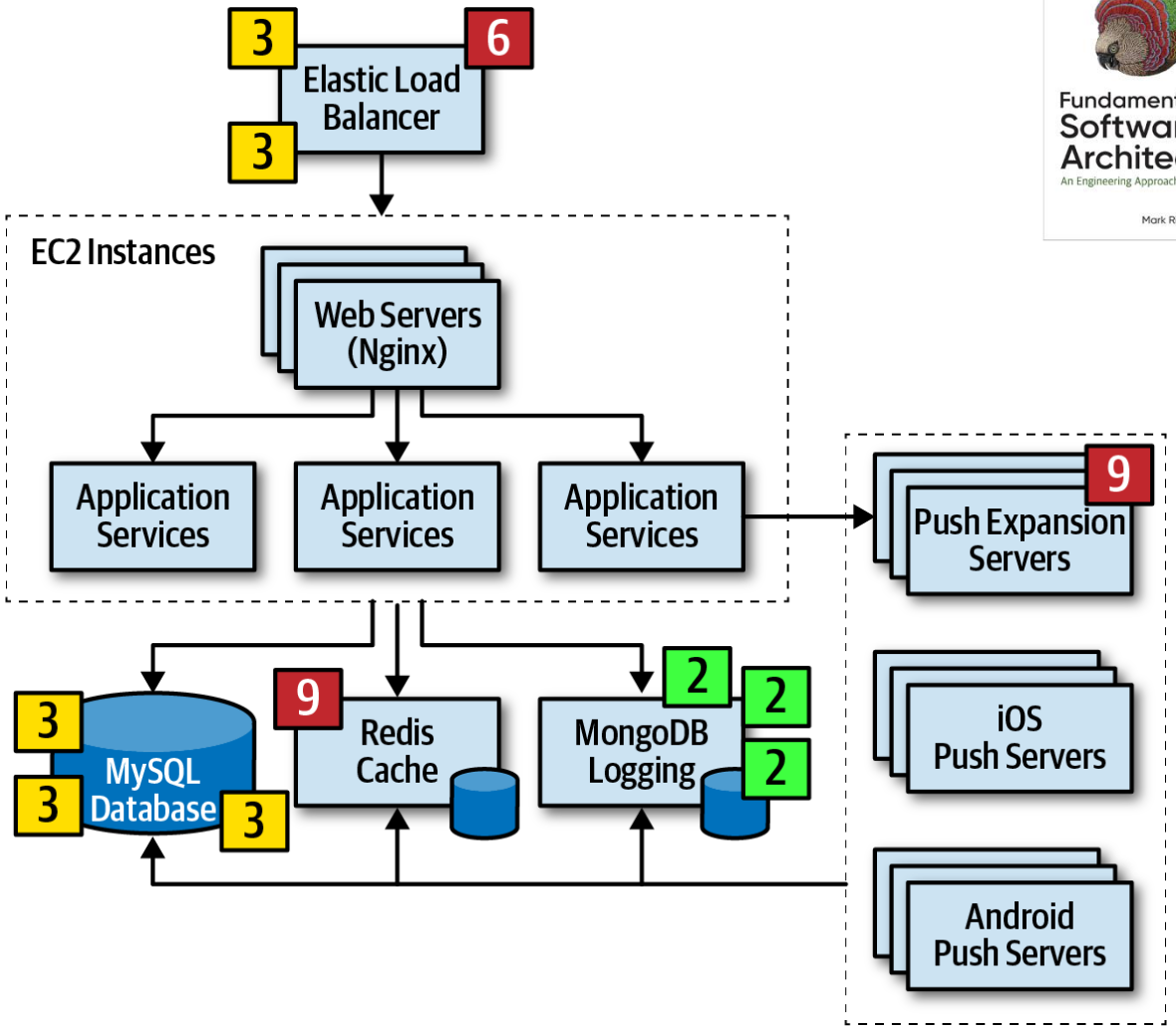


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Initial identification of risk areas

Two participants individually identified the Elastic Load Balancer as medium risk (3), whereas one participant identified it as high risk (6).

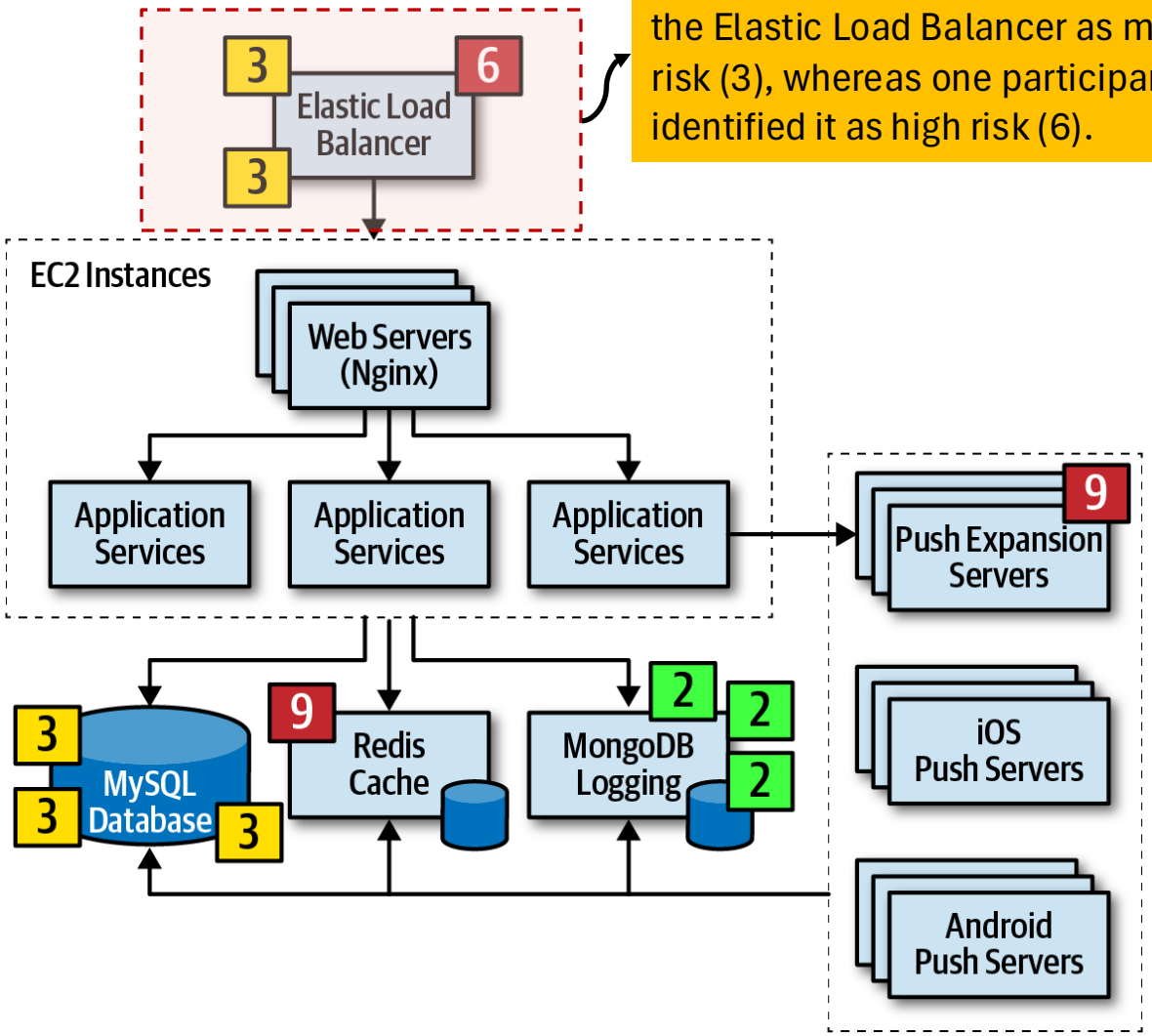


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Initial identification of risk areas

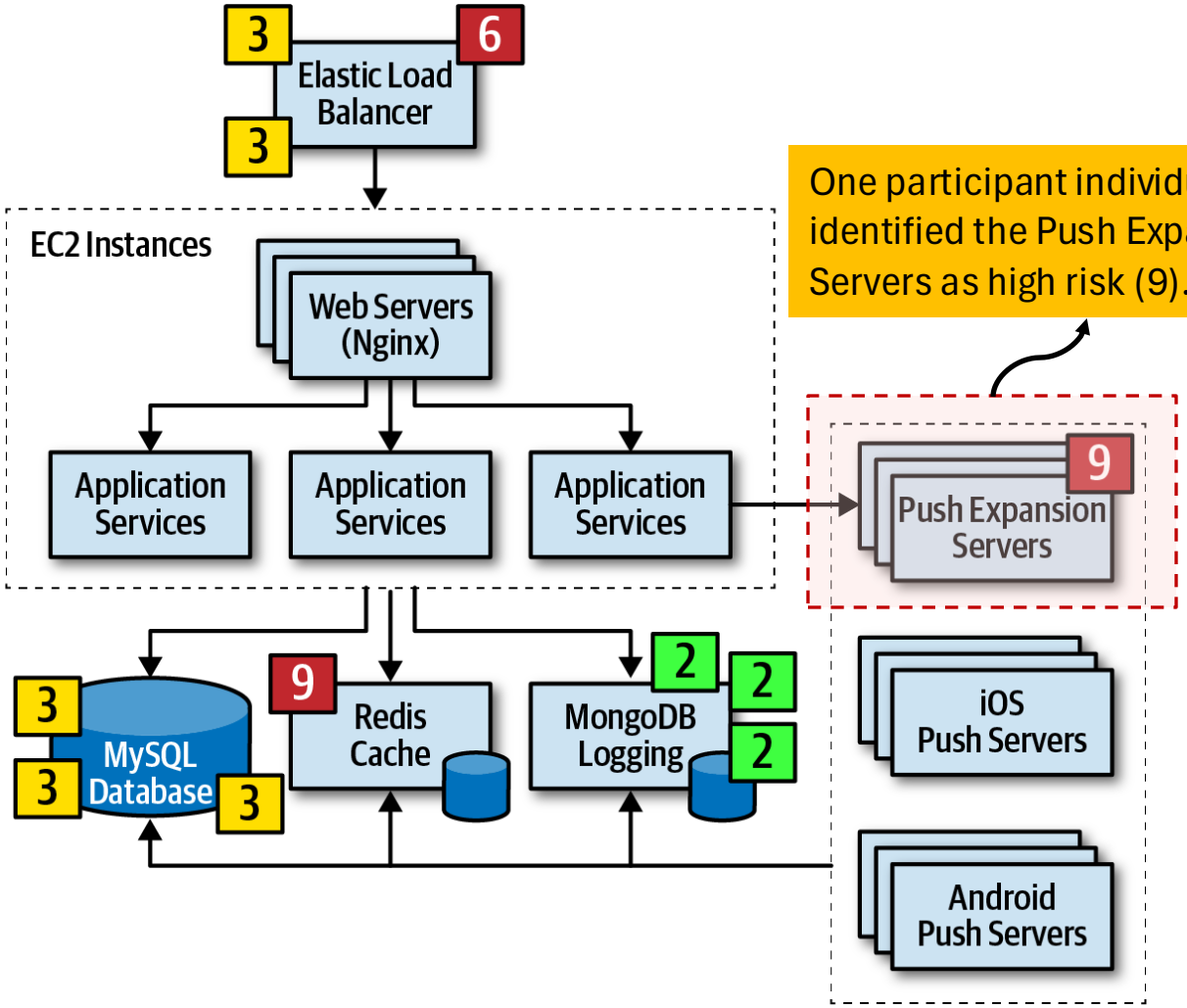


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Initial identification of risk areas

Three participants individually identified the MySQL database as medium risk (3).

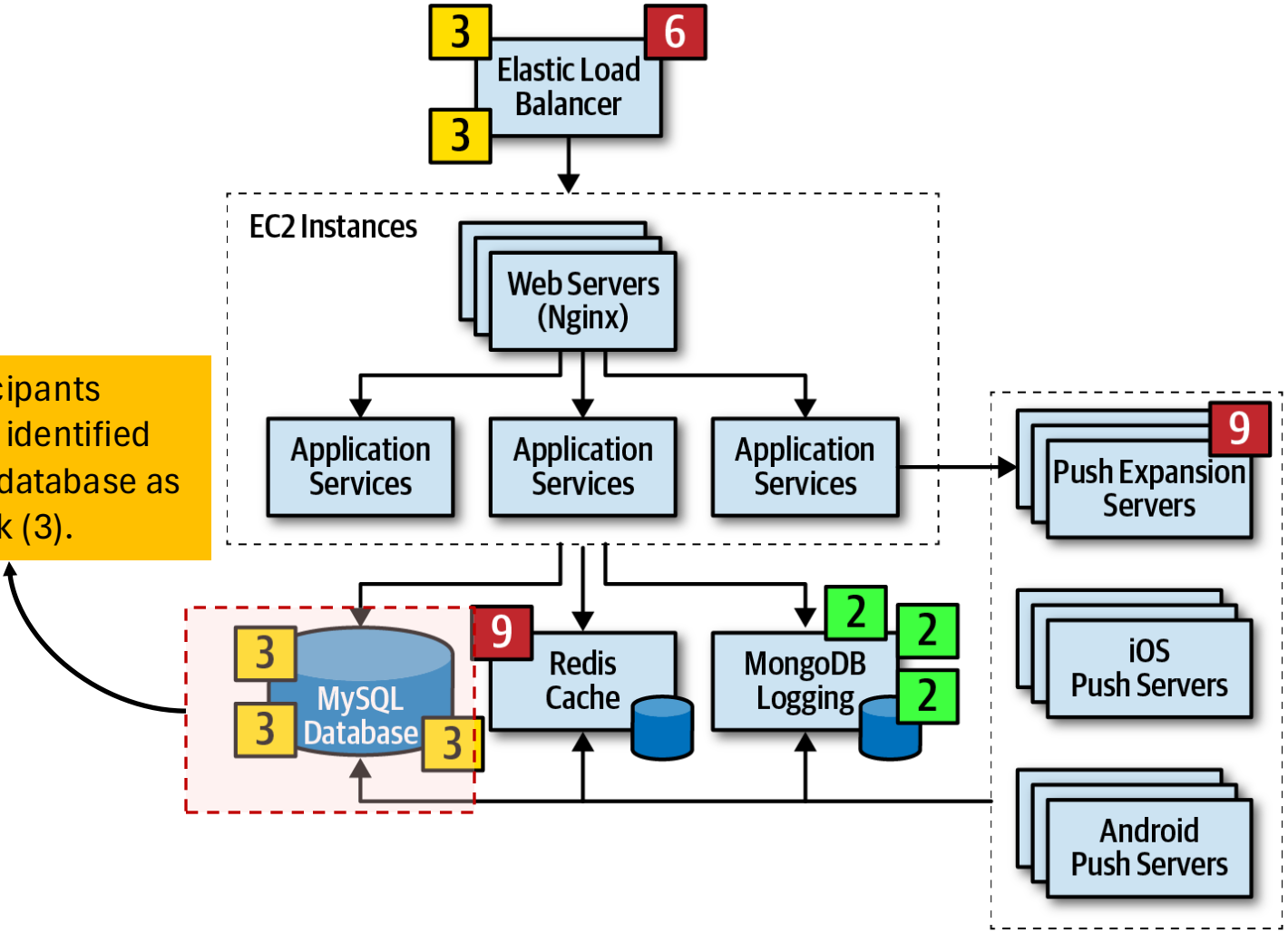
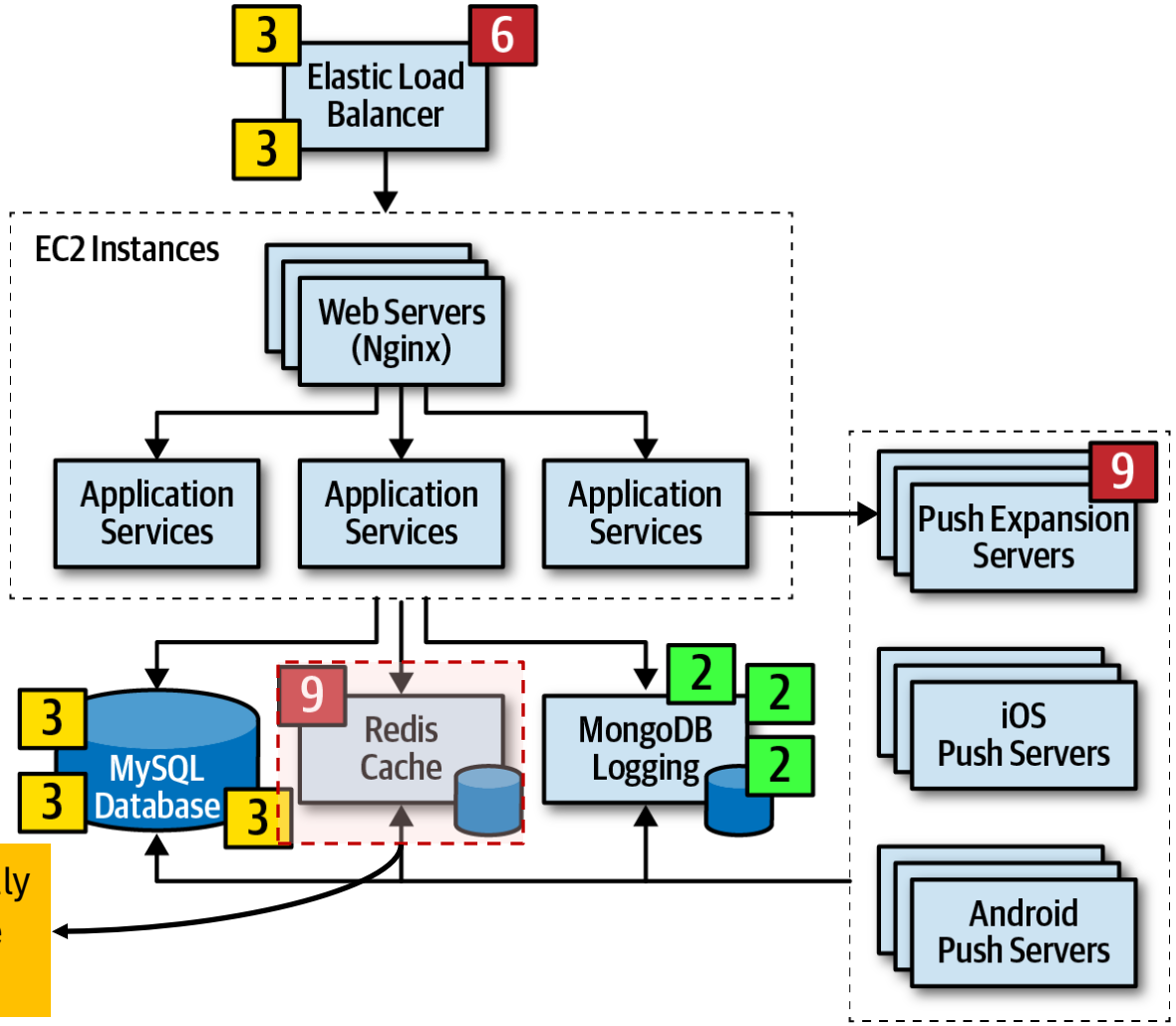


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Initial identification of risk areas



One participant individually identified the Redis cache as high risk (9).

Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Initial identification of risk areas

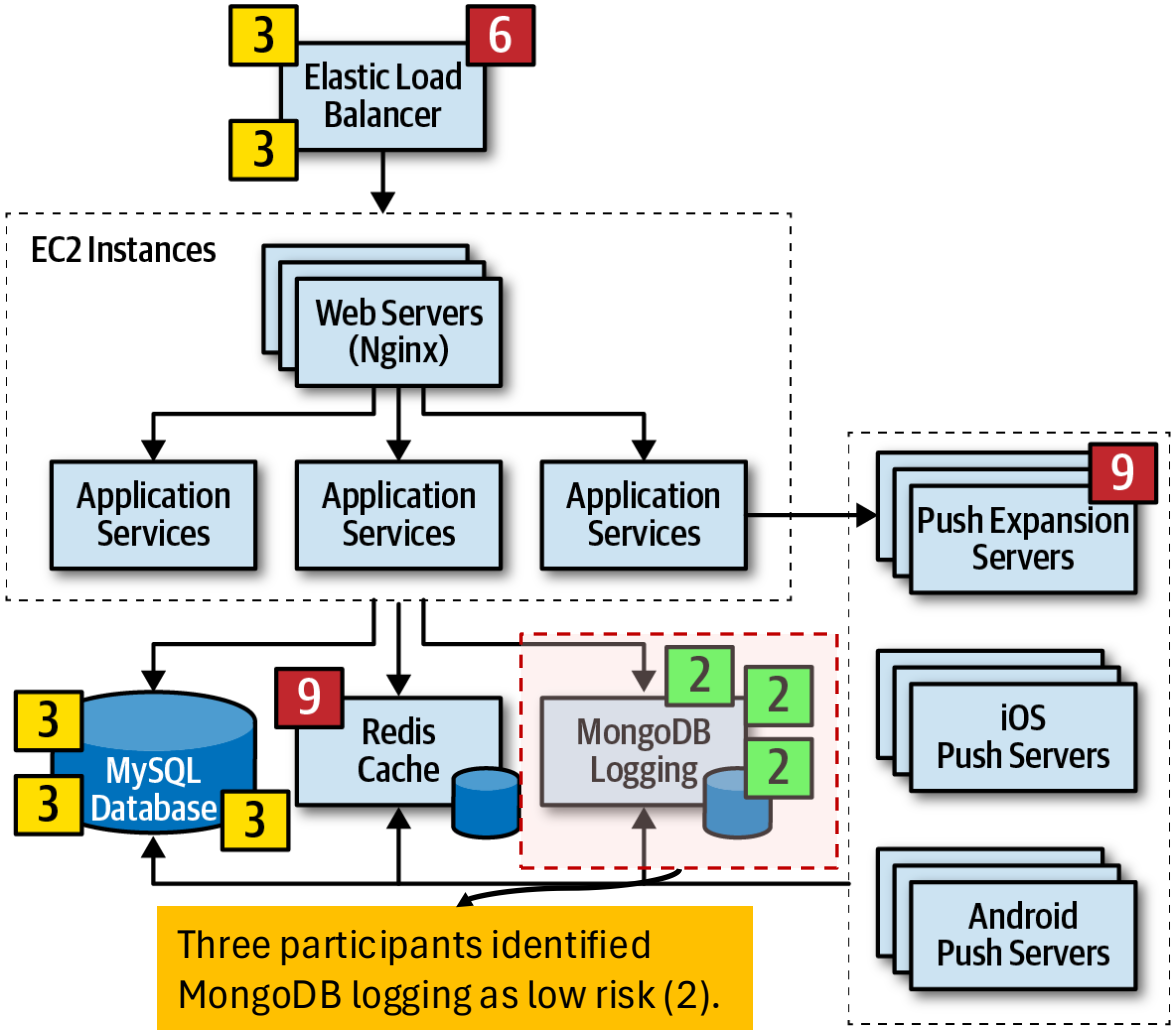


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Consensus of risk areas

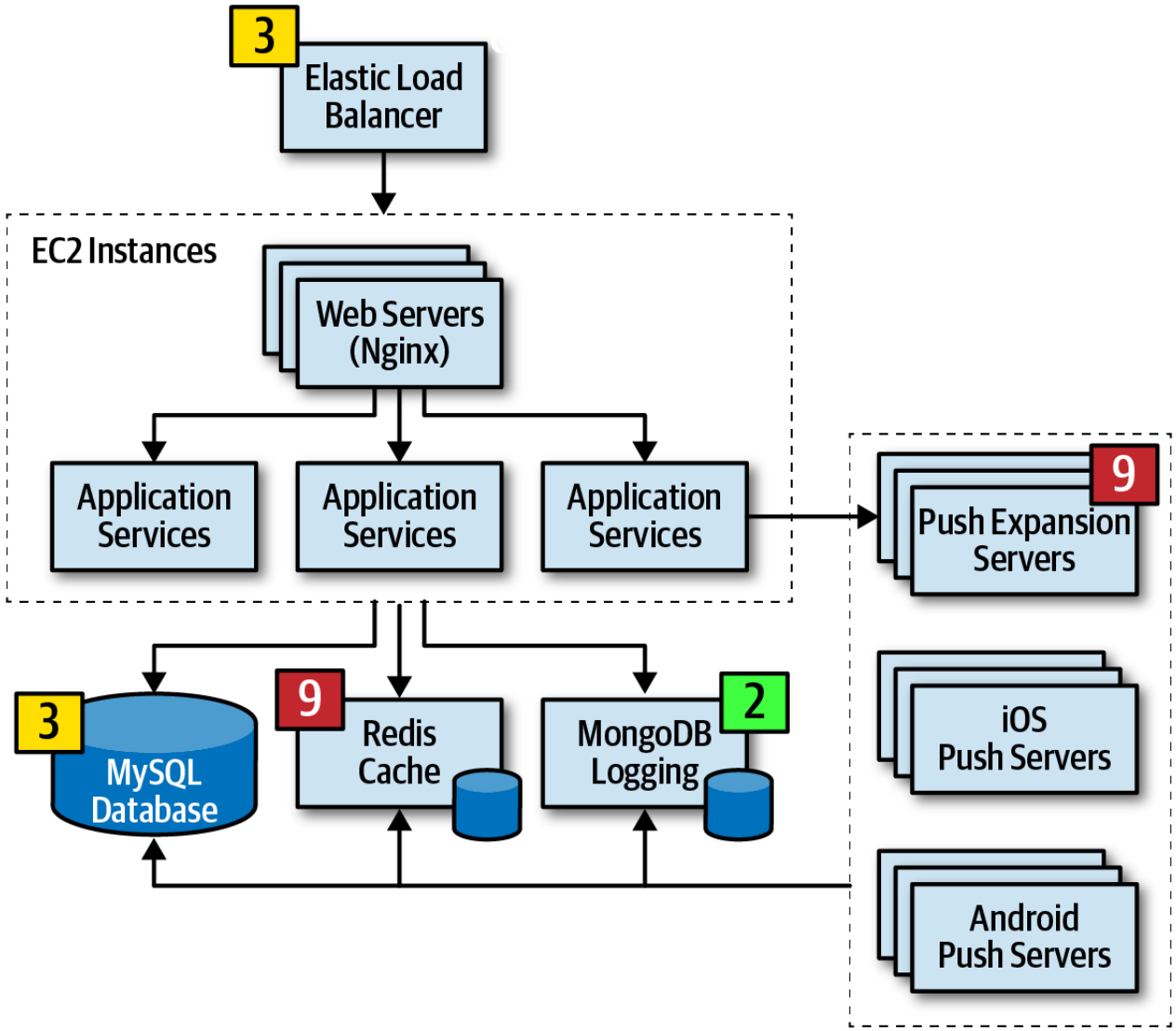
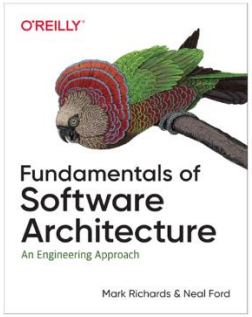


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.



Case Study

How Risk Storming Can Improve the Architecture of a System?

Case Study: Call Center System

- System: A **call center system** to support nurses advising patients on various health conditions.

Requirements

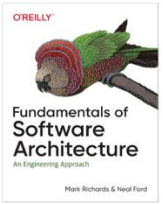
- **RQ1**. The system will use a third-party diagnostics engine that serves up questions and guides the nurses or patients regarding their medical issues.
- **RQ2**. Patients can either call in using the call center to speak to a nurse or choose to use a self-service website that accesses the diagnostic engine directly, bypassing the nurses.
- **RQ3**. The system must support 250 concurrent nurses nationwide and up to hundreds of thousands of concurrent self-service patients nationwide.
- **RQ4.** Nurses can access patients' medical records through a medical records exchange, but patients cannot access their own medical records.

Case Study: Call Center System

Requirements

- **RQ5.** The system must be HIPAA (Health Insurance Portability and Accountability Act) compliant with regard to the medical records. This means that it is essential that no one but nurses have access to medical records.
- **RQ6.** Outbreaks and high volume during cold and flu season need to be addressed in the system.
- **RQ7.** Call routing to nurses is based on the nurse's profile (such as languages spoken or medical specializations).
- **RQ8.** The third-party diagnostic engine can handle about 500 requests a second.

The (initial) architecture of the Call Center System



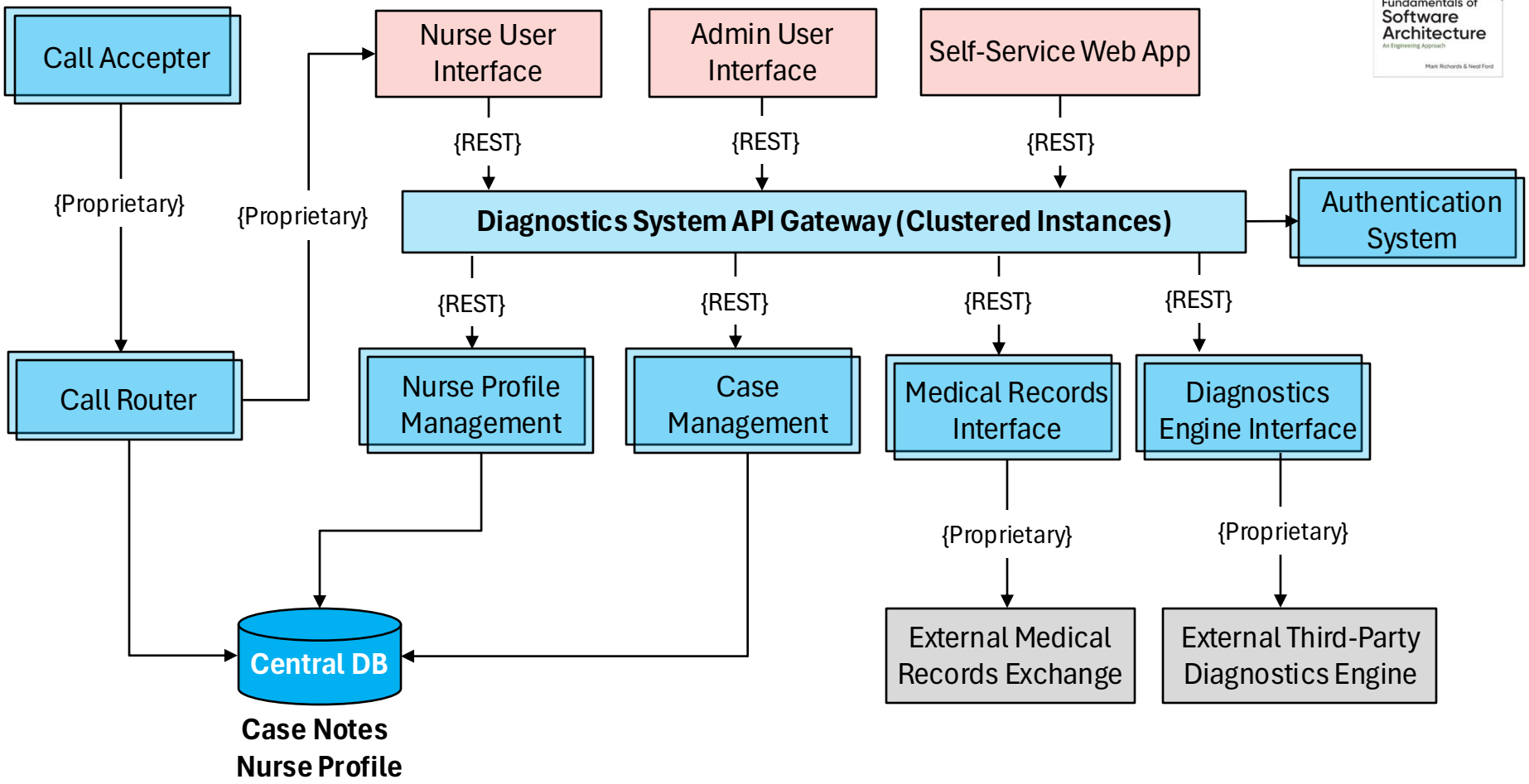


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

The (initial) architecture of the Call Center System

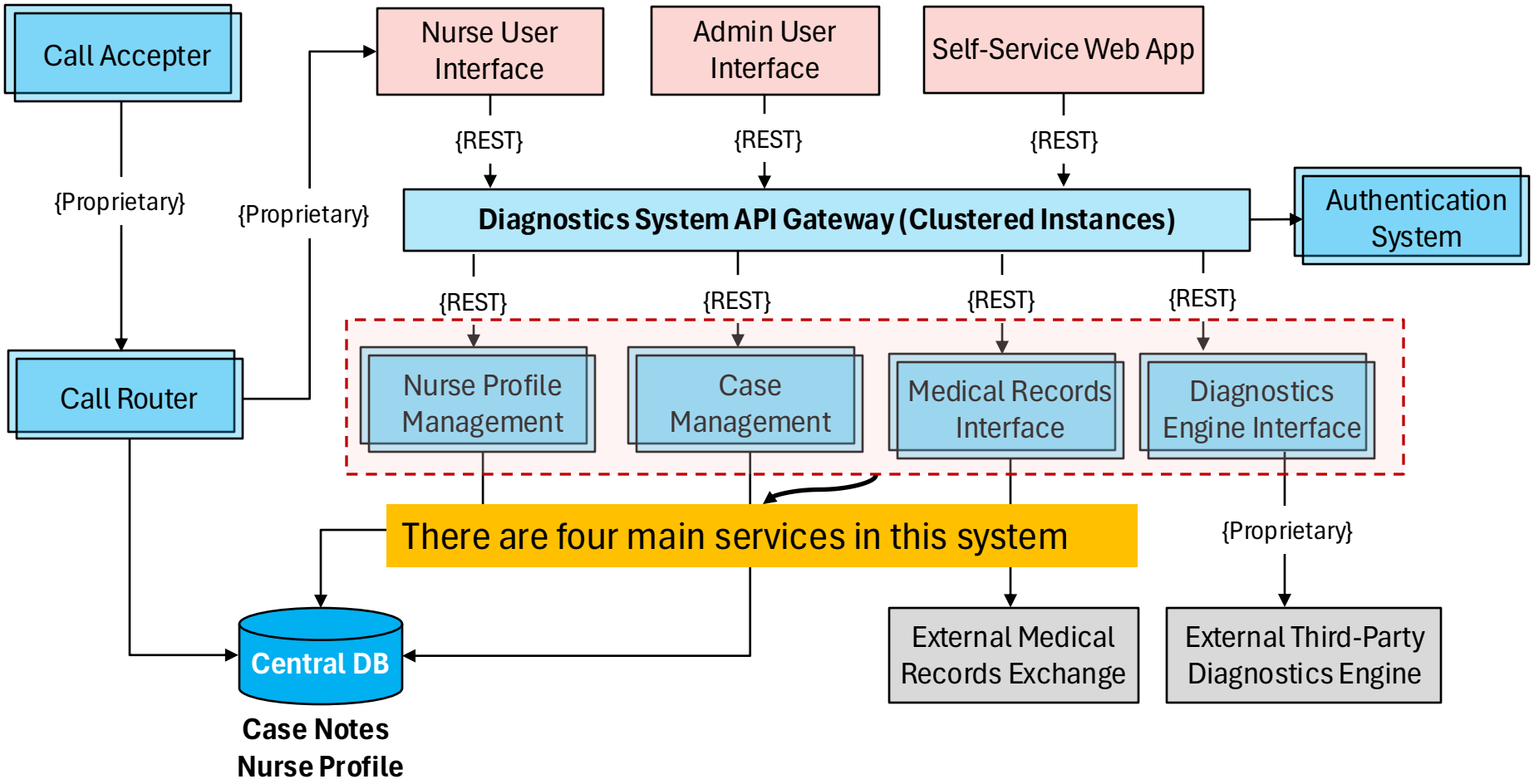


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

The (initial) architecture of the Call Center System

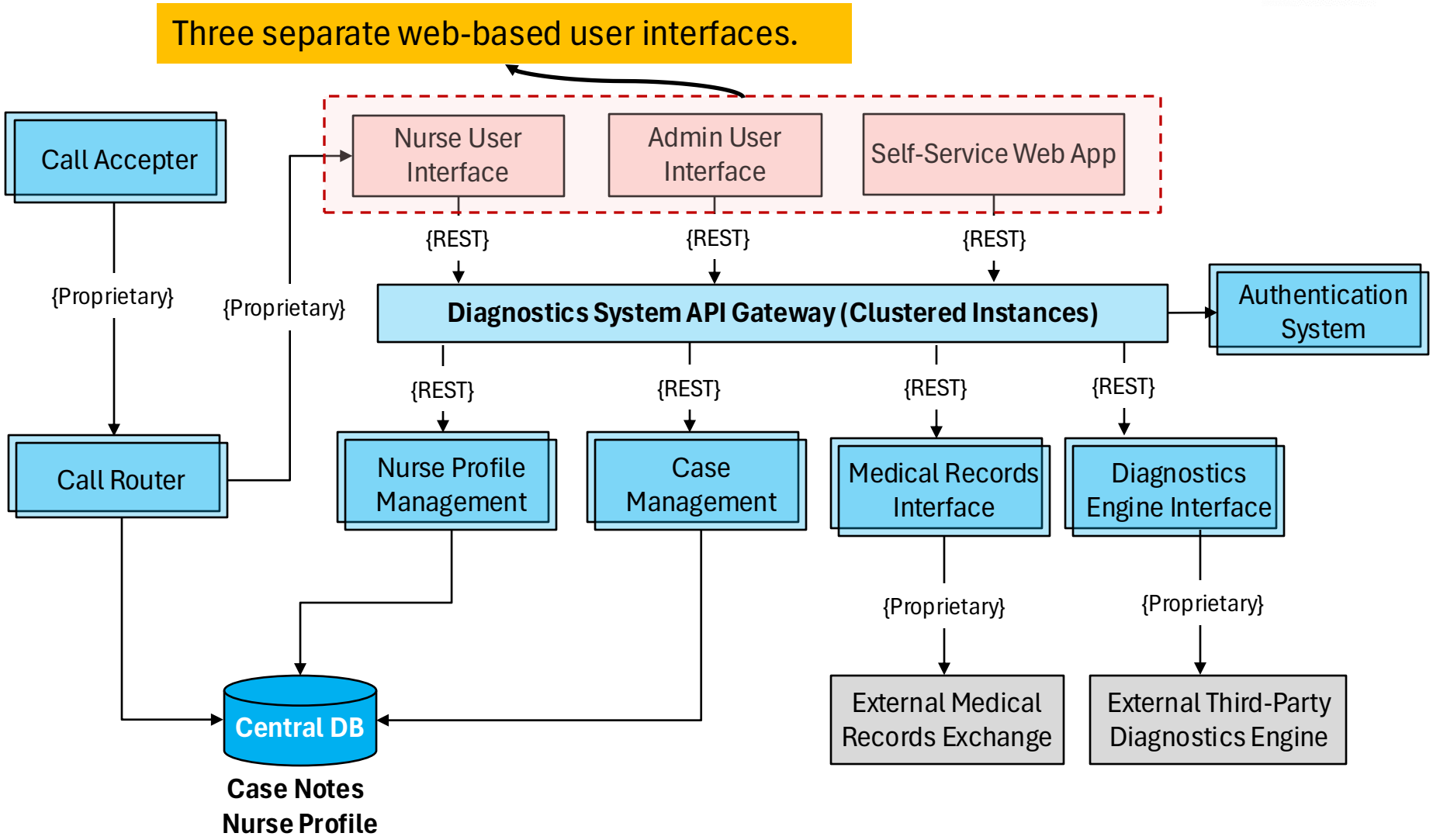
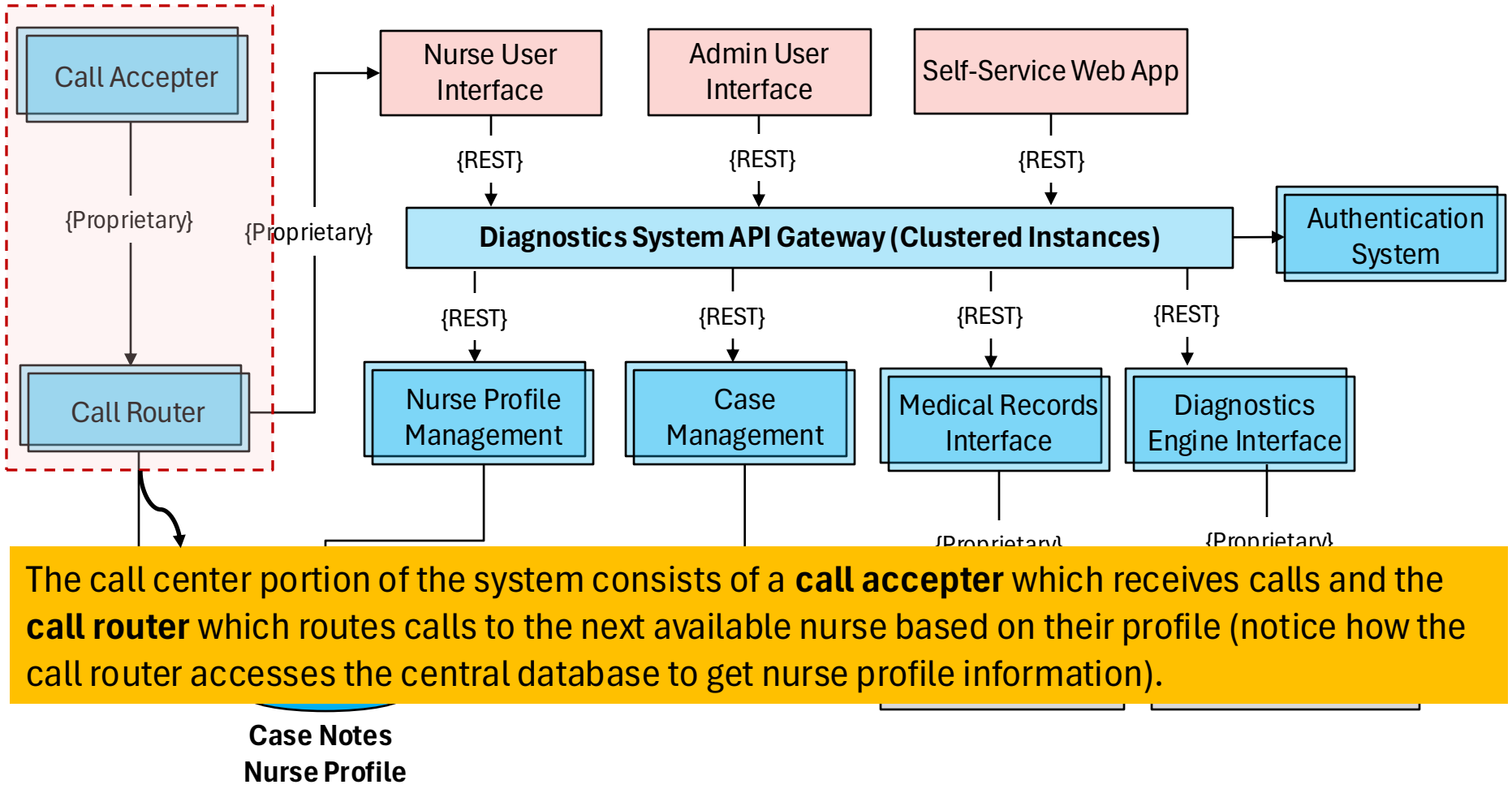


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

The (initial) architecture of the Call Center System



The call center portion of the system consists of a **call accepter** which receives calls and the **call router** which routes calls to the next available nurse based on their profile (notice how the call router accesses the central database to get nurse profile information).

Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

The (initial) architecture of the Call Center System

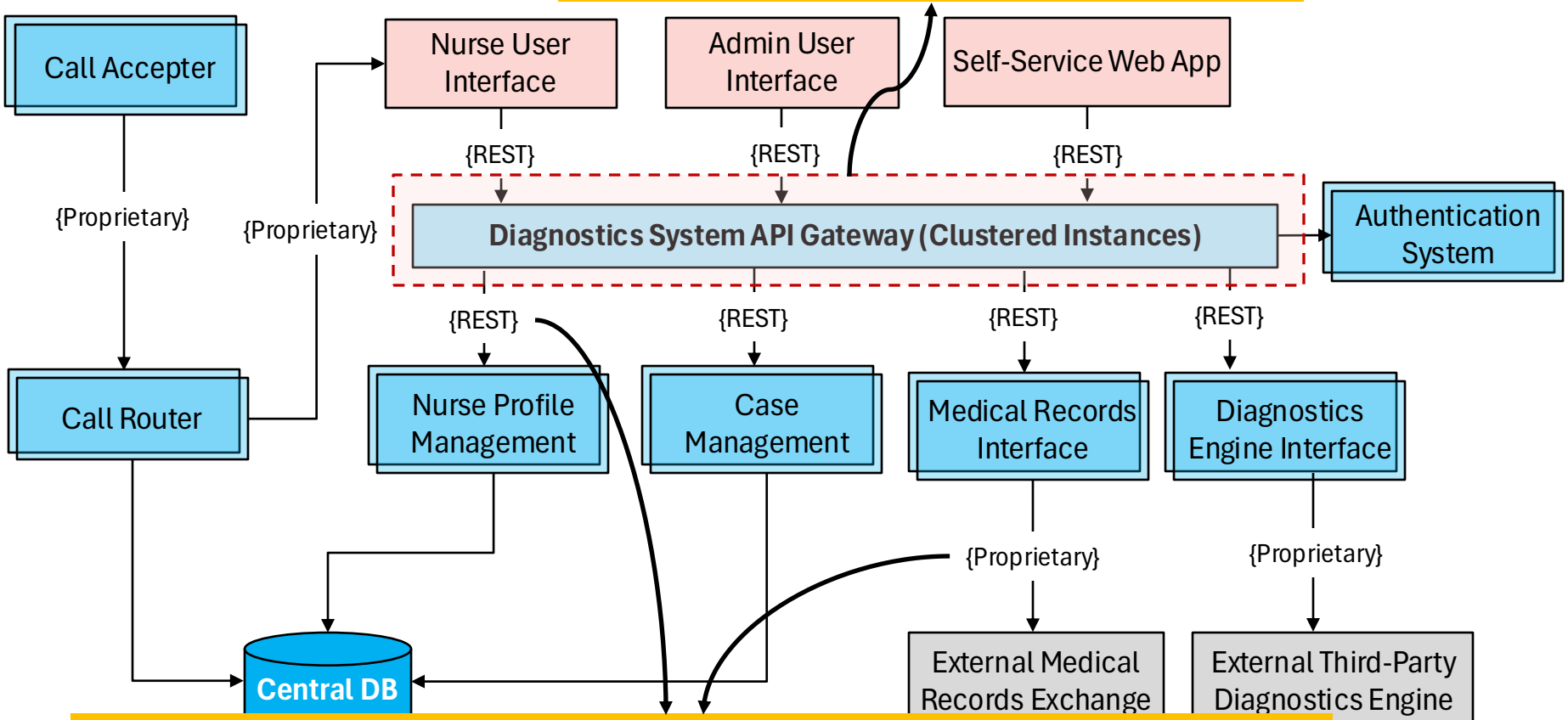


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Availability risk

- During the first risk storming exercise, the architect chose to focus on **availability** first since **system availability is critical** for the success of this system.
- After the risk storming identification and collaboration activities, the participants came up with the following risk areas using the risk matrix.
  - The **use of a central database** was identified as high risk **(6)** due to high impact **(3)** and medium likelihood **(2)**.
  - Other parts of the system were not deemed as risk for availability due to multiple instances of each service and clustering of the API gateway.

Availability risk

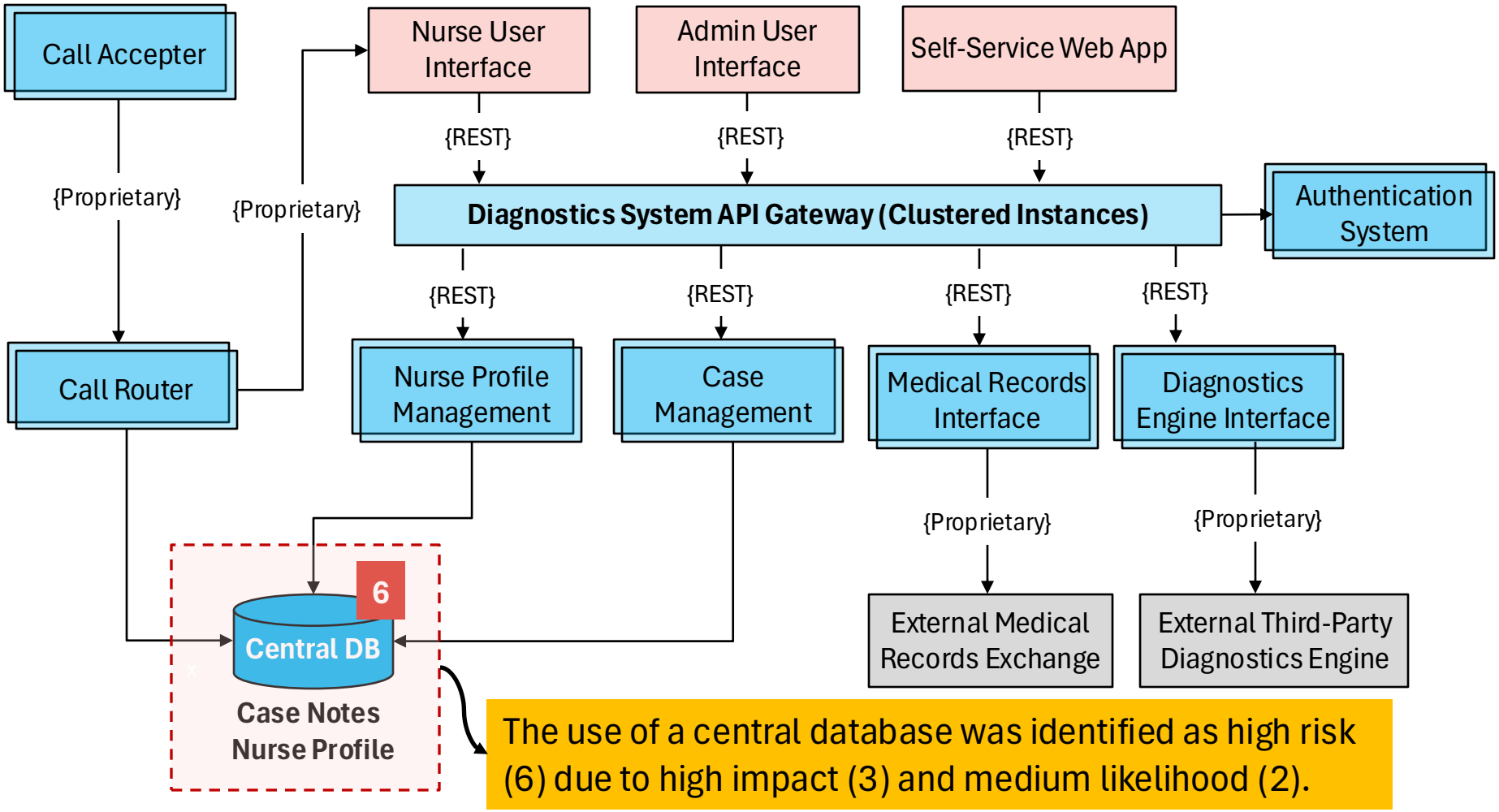


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Availability

Other parts of the system were not deemed as risk for availability due to multiple instances of each service and clustering of the API gateway.

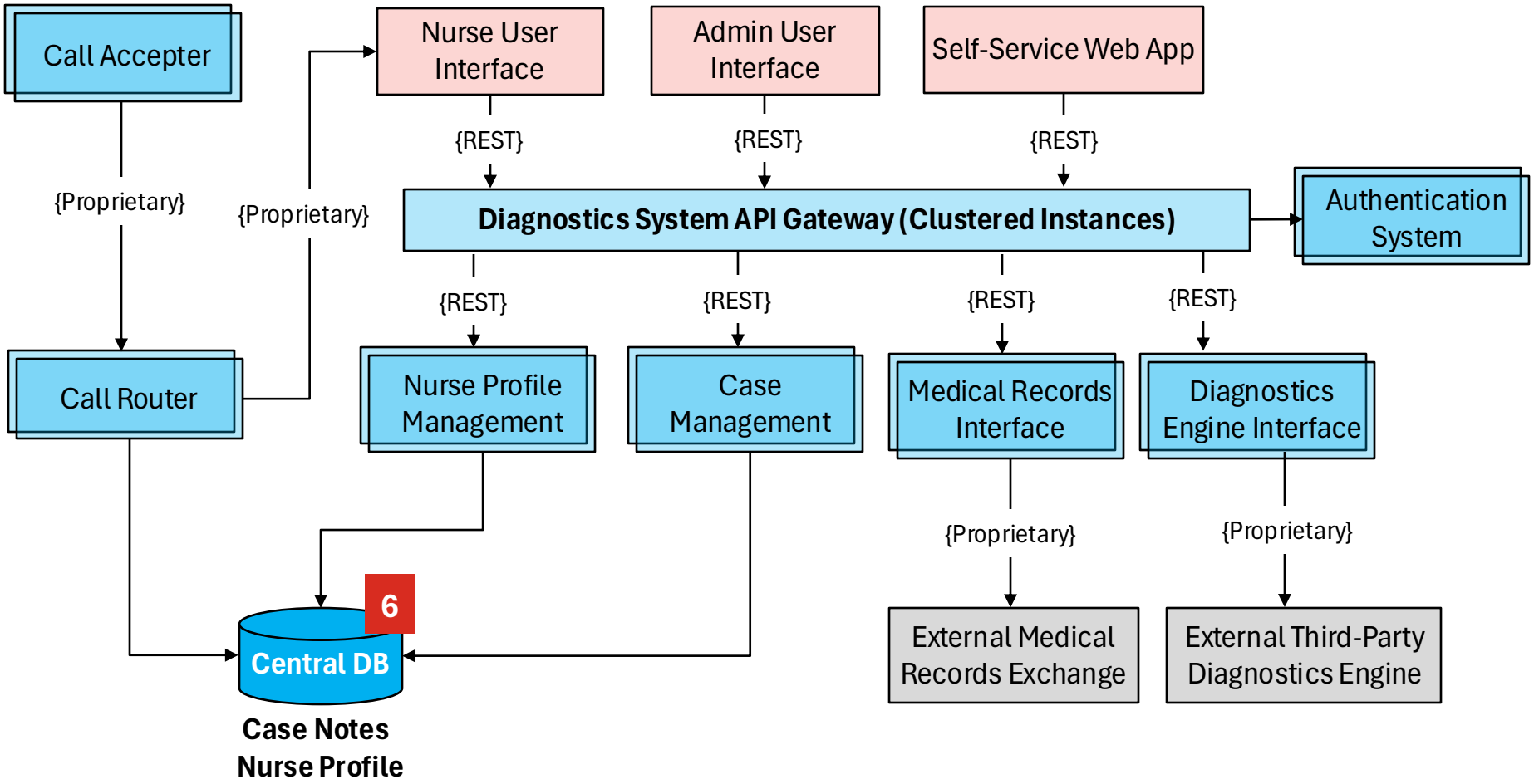


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Availability risk - Mitigation

- How to mitigate the risk of the availability of the database?
  - If the database goes down
    - Nurses can still manually write down case notes
    - The call router could not function

Mitigating Strategy

Break apart the single physical database into **two separate databases**:

- One clustered database containing the nurse profile information
- One single instance database for the case notes.

Architecture modifications to address availability risk

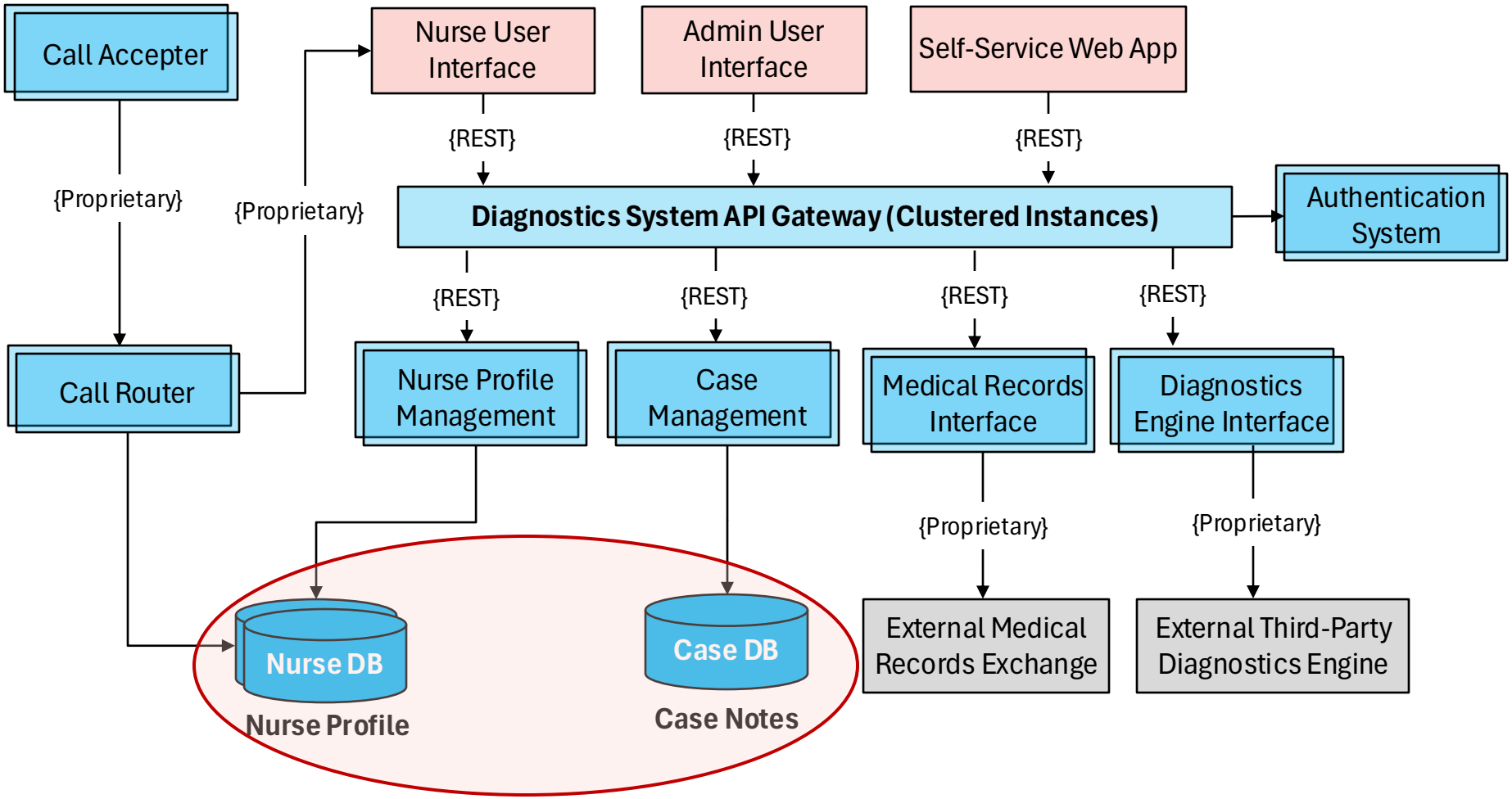


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Elasticity risk

- During the second risk storming exercise, the architect chose to focus on **elasticity** - spikes in user load
  - Although there are only 250 nurses (which means no more than 250 nurses will be accessing the Diagnostics Engine), the **self-service portion** of the system can access the **Diagnostics Engine** as well, which significantly increases the number of requests to the Diagnostics Engine interface. The participants are concerned about flu season and COVID outbreaks, which will significantly increase the anticipated load on the system.
- During the risk storming session, the participants all identified the **Diagnostics Engine Interface** as high risk **(9)**.
  - With only 500 requests per second, the participants calculated that there was no way the diagnostics engine interface could keep up with the anticipated throughput, particularly with the current architecture utilizing REST as the interface protocol.

Elasticity risk

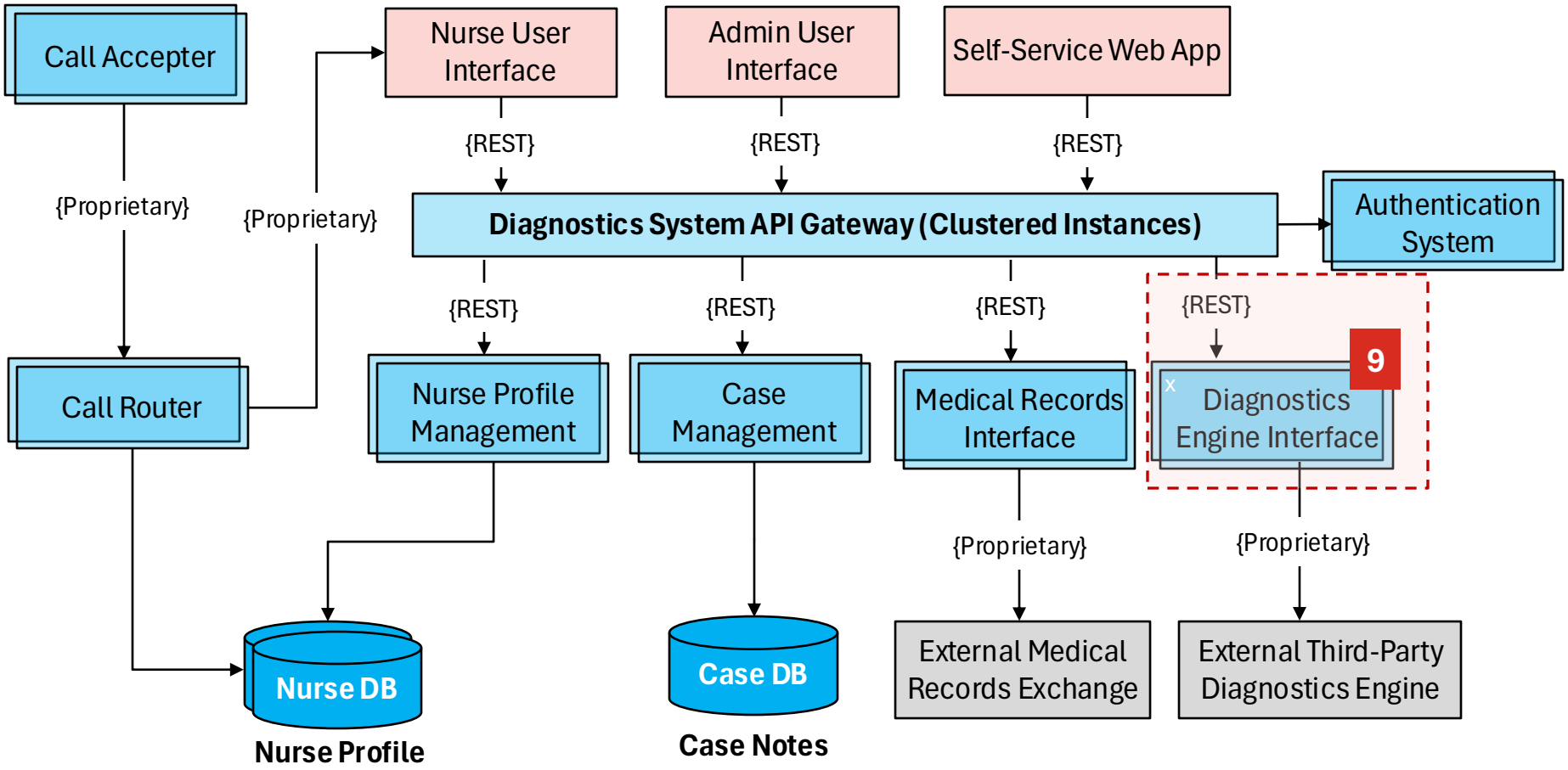


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Elasticity risk - Mitigation

- How to mitigate this elasticity risk?

Mitigating Strategies (Two)

(1) Leveraging **two asynchronous queues** (messaging) between the API gateway and the diagnostics engine interface:
   - One queue for the nurses
   - One queue for the self-service web app (self-service patients).

(2) Caching the particular diagnostics questions related to an outbreak would remove outbreak and flu calls from ever having to reach the diagnostics engine interface.

Architecture modifications to address scalability risk

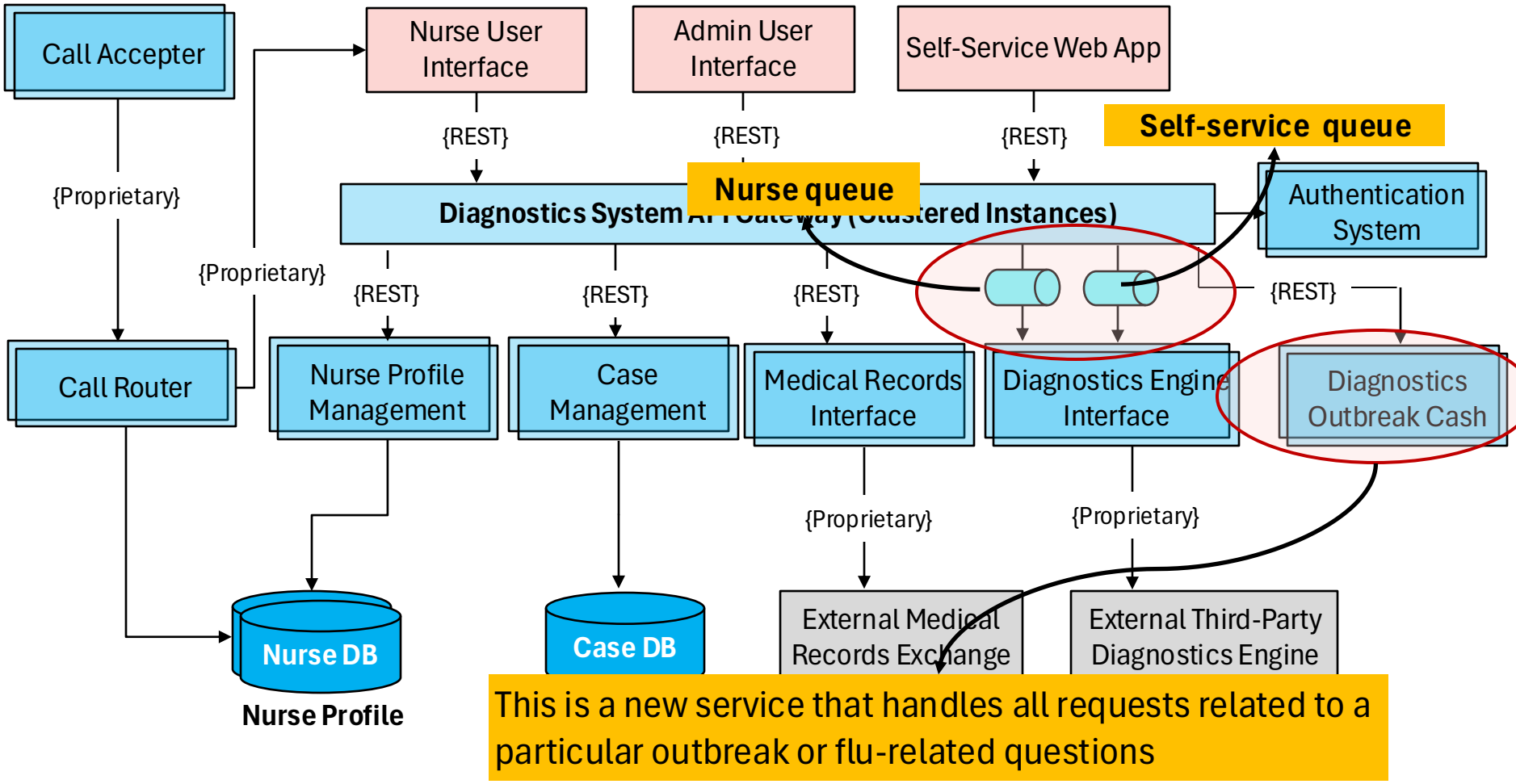


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Security risk

- During the third risk storming exercise, the architect chose to focus on **security**
  - Due to HIPAA regulatory requirements, access to medical records via the medical record exchange interface must be secure, allowing only nurses to access medical records if needed. However, all participants in risk storming exercise believe that **this is not a problem due to security checks in the API gateway** (authentication and authorization).
  - The participants all identified the **Diagnostics System API gateway** as a high security risk **(6)**. The rationale for this high rating was the high impact of admin staff or self-service patients accessing medical records **(3)** combined with medium likelihood **(2)**.

Security risk

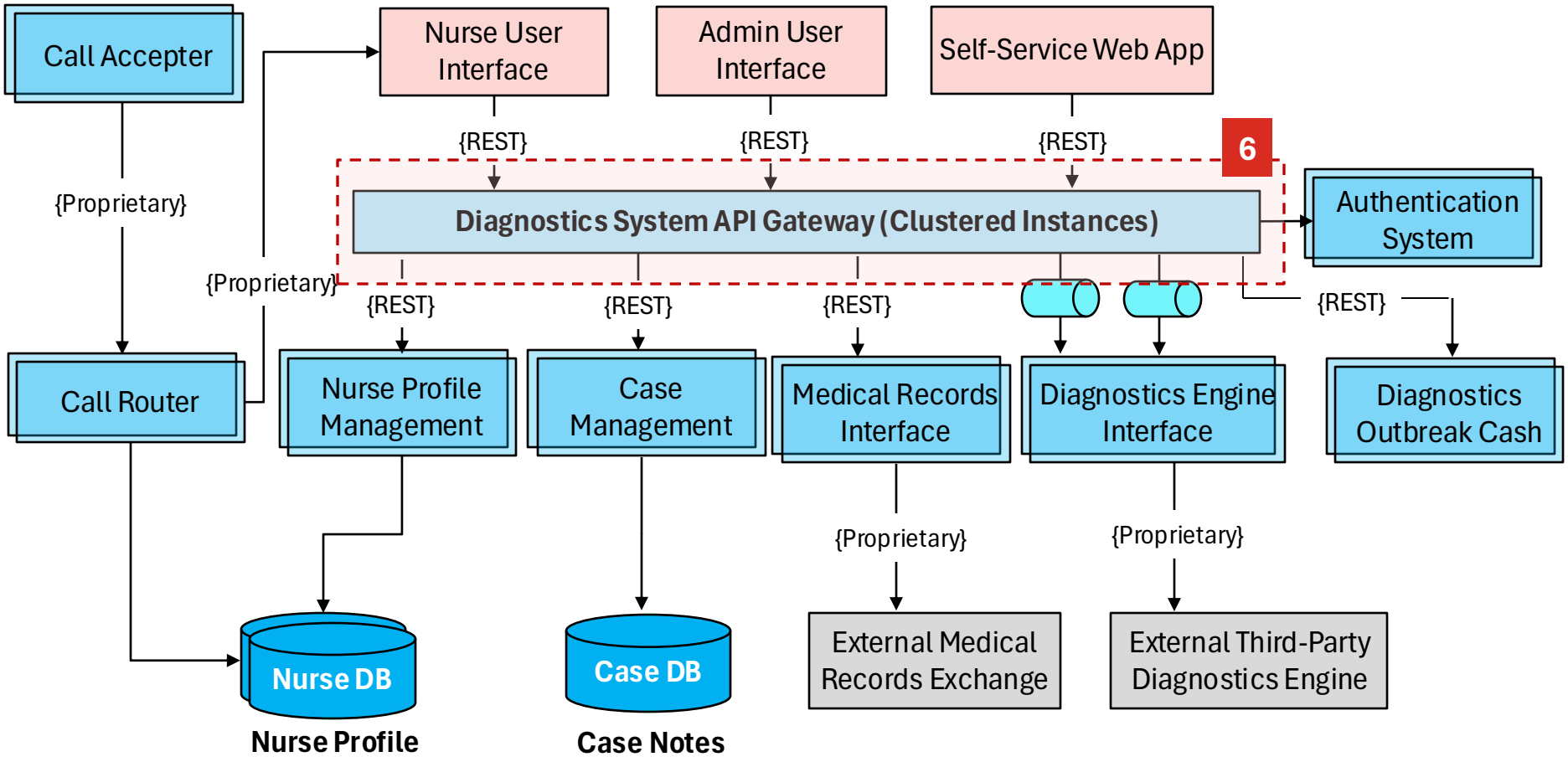


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Security risk - Mitigation

- How to mitigate this security risk?

Mitigating Strategy

Separating API gateways for each type of user (admin, self-service/diagnostics, and nurses) would prevent calls from either the admin web user interface or the self-service web user interface from ever reaching the medical records exchange interface.

Architecture modifications to address security risk

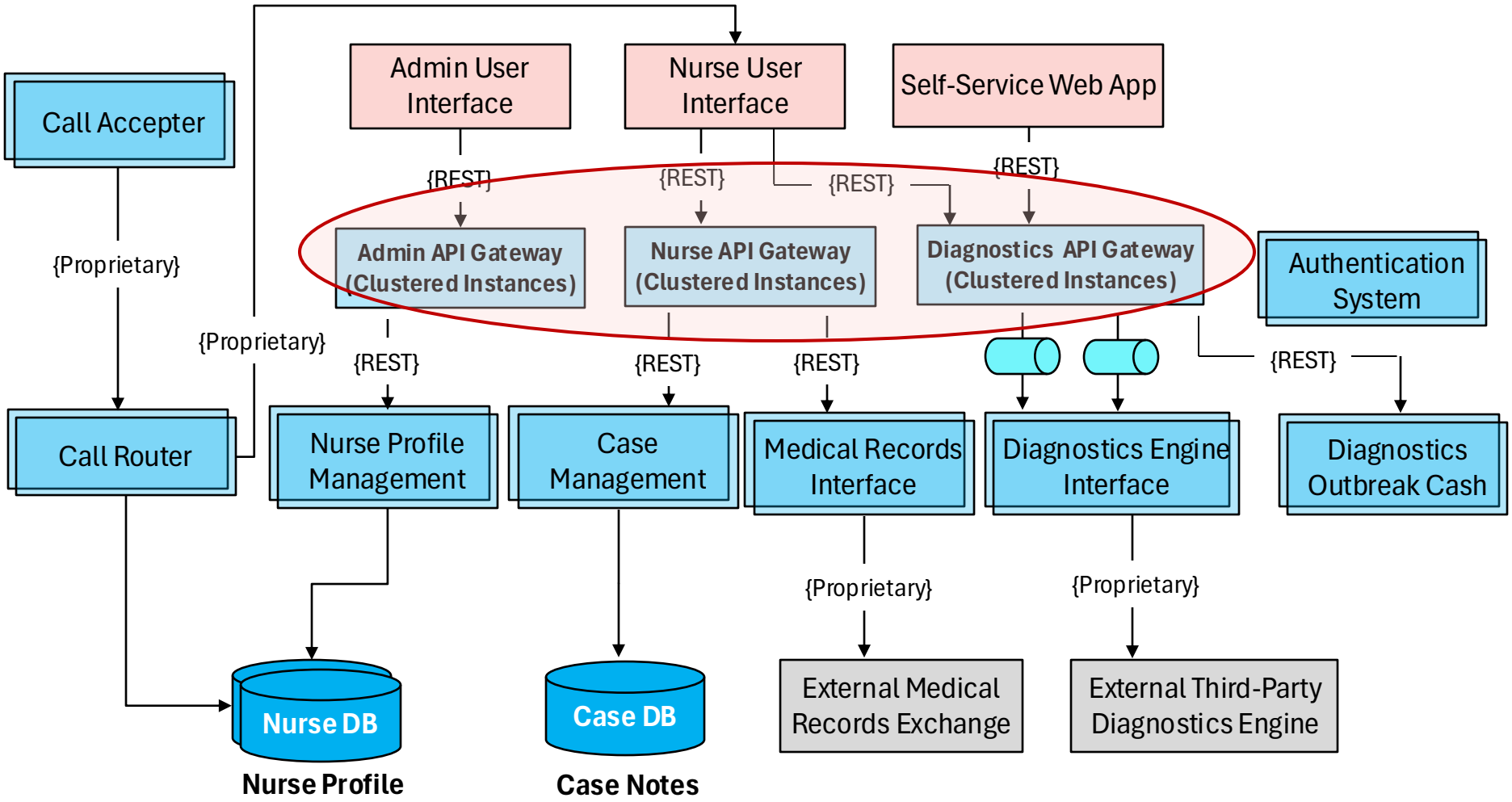


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

Modeling Distributed Architectures with C4 Model

Modeling Service-based Architecture with C4 Model

Service-based Architecture

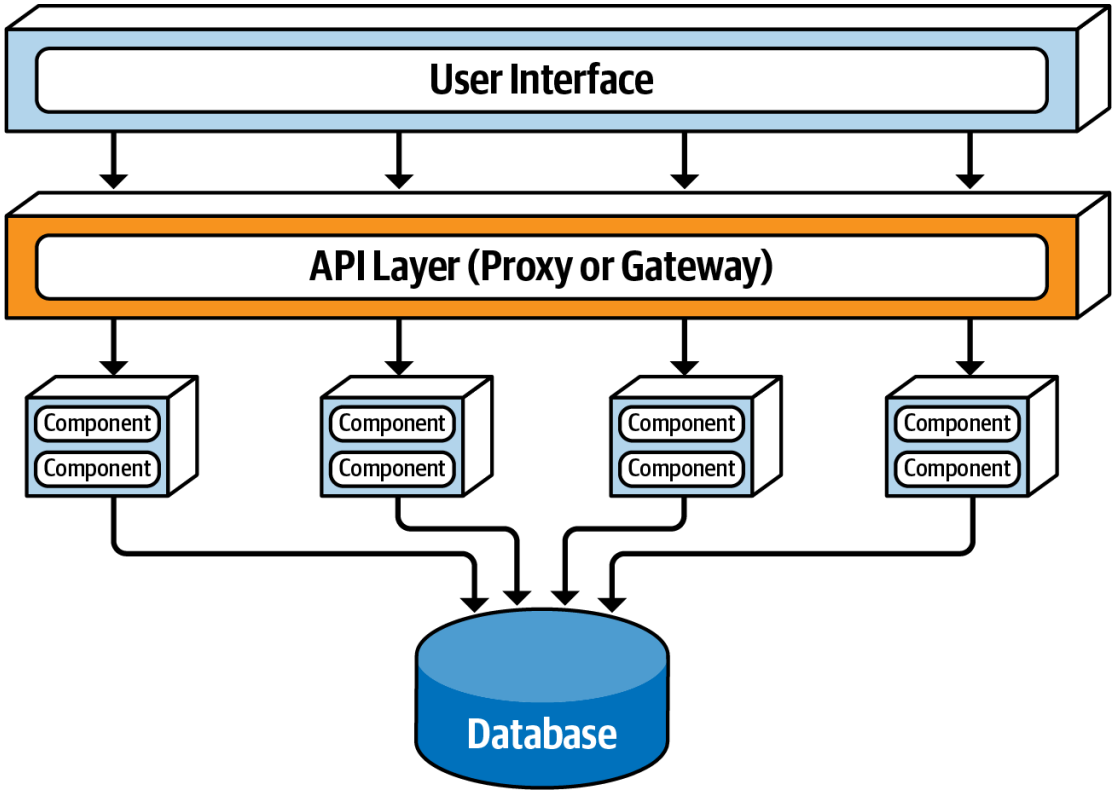
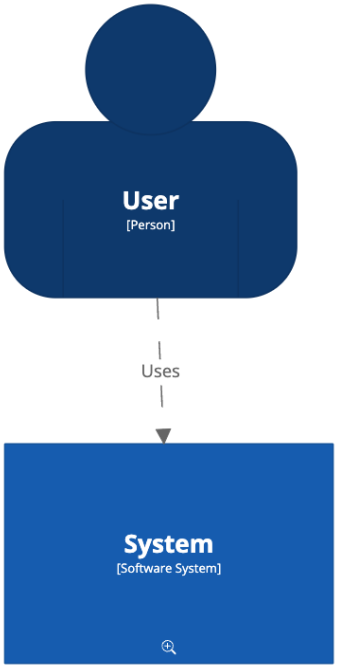


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

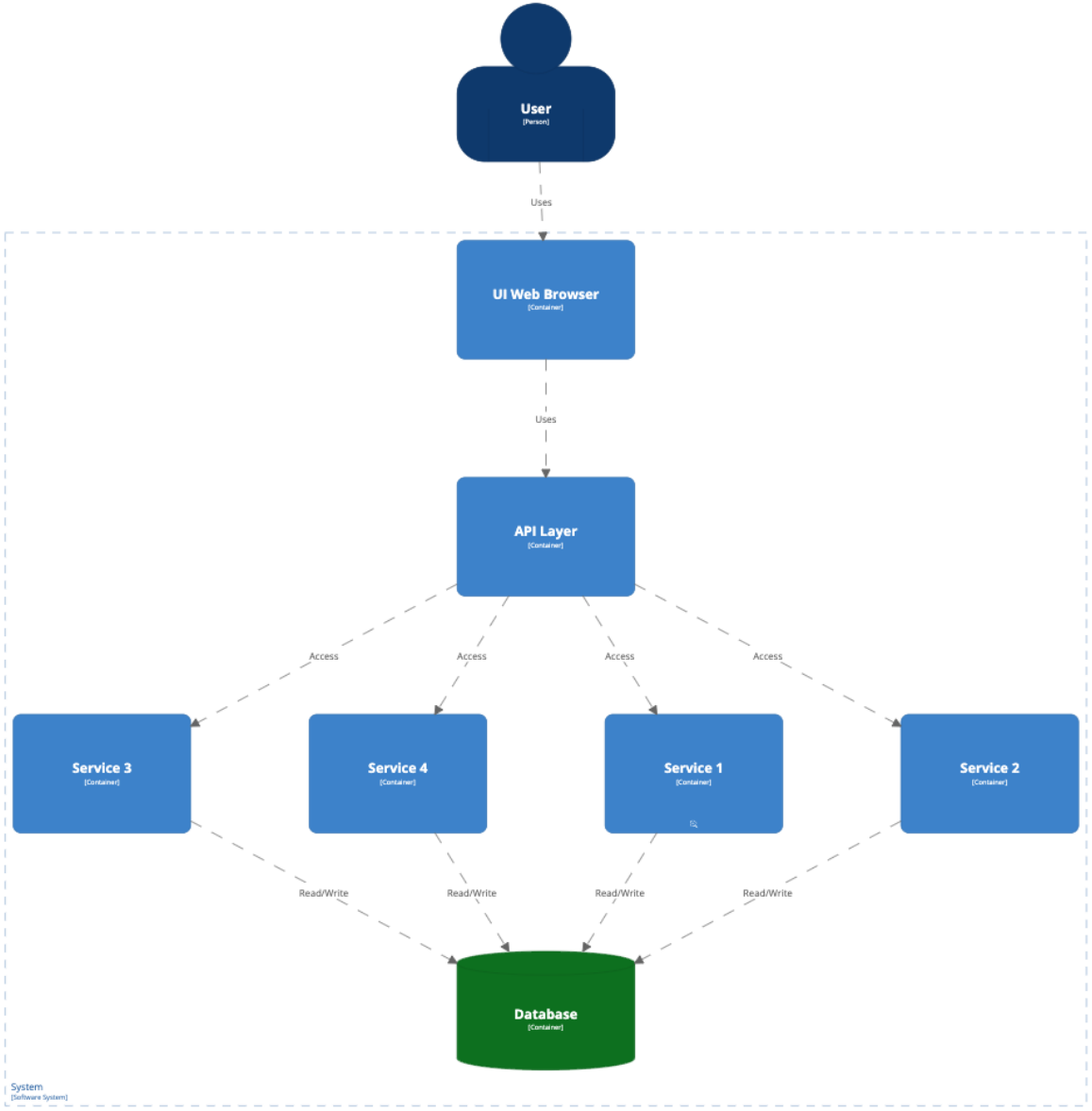
Service-based Architecture with the C4 Model

System Context Diagram



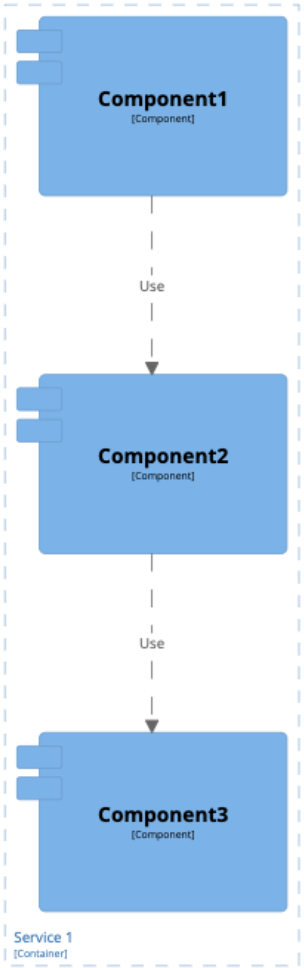
Service-based Architecture with the C4 Model

Container Diagram



Service-based Architecture with the C4 Model

Component Diagram for Service 1



Modelling Microservices Architecture with C4 Model

Microservices Architectures

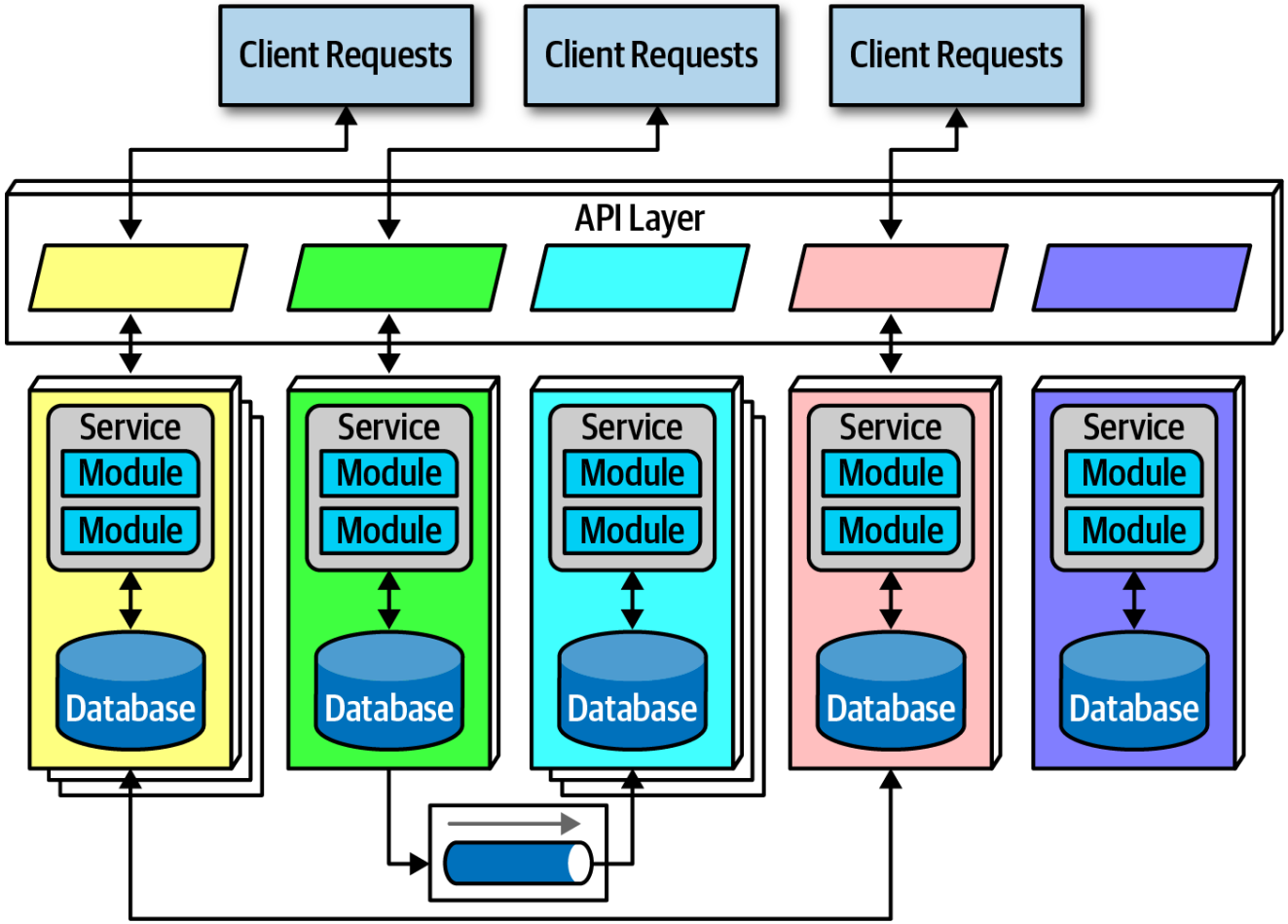
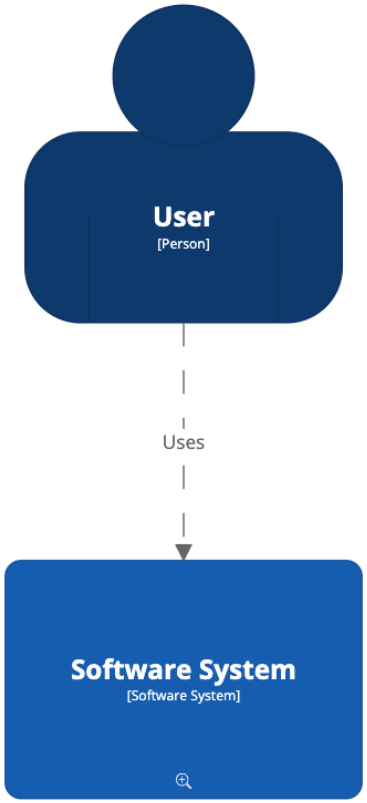


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020.

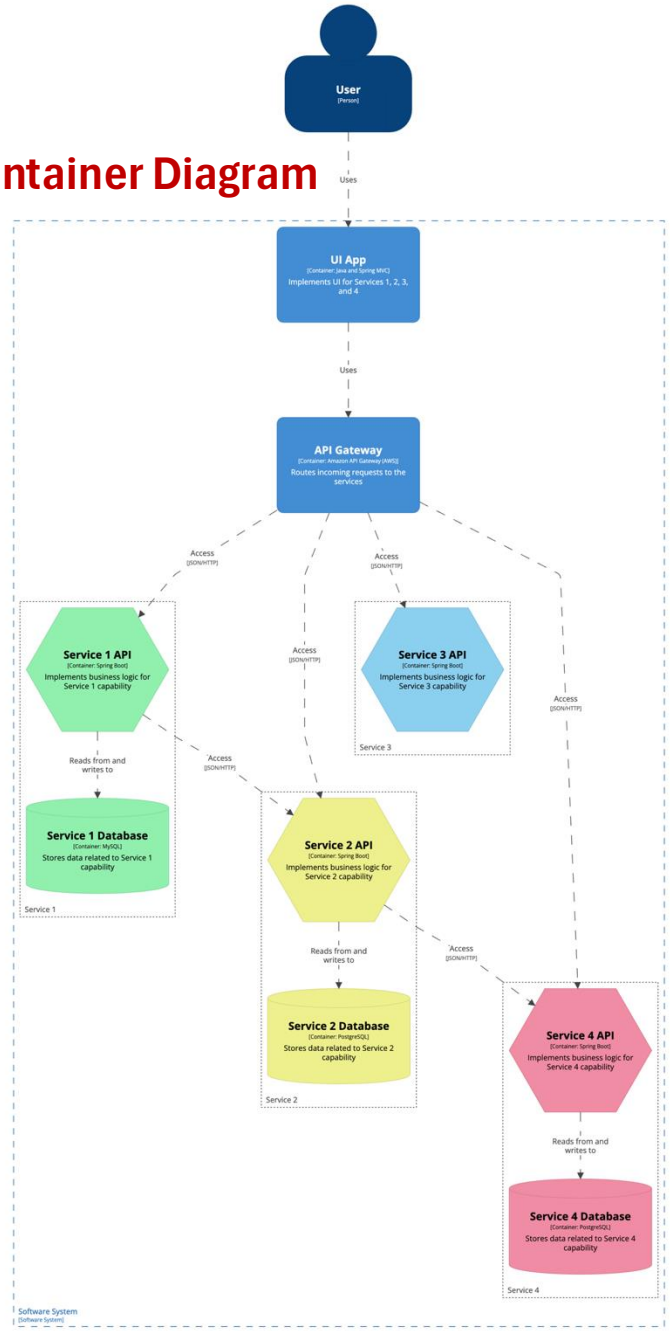
Microservices Architecture with C4 Model

System Context Diagram



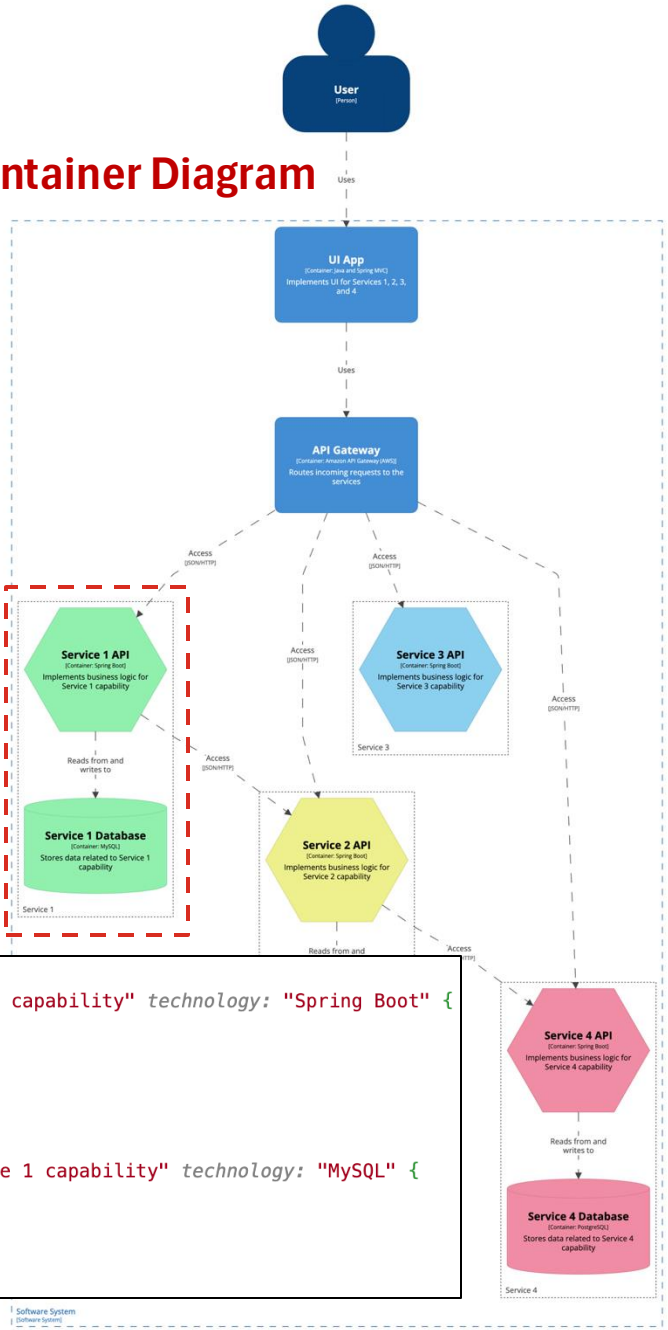
Microservices Architecture with C4 Model

Container Diagram



Microservices Architecture with C4 Model

Container Diagram



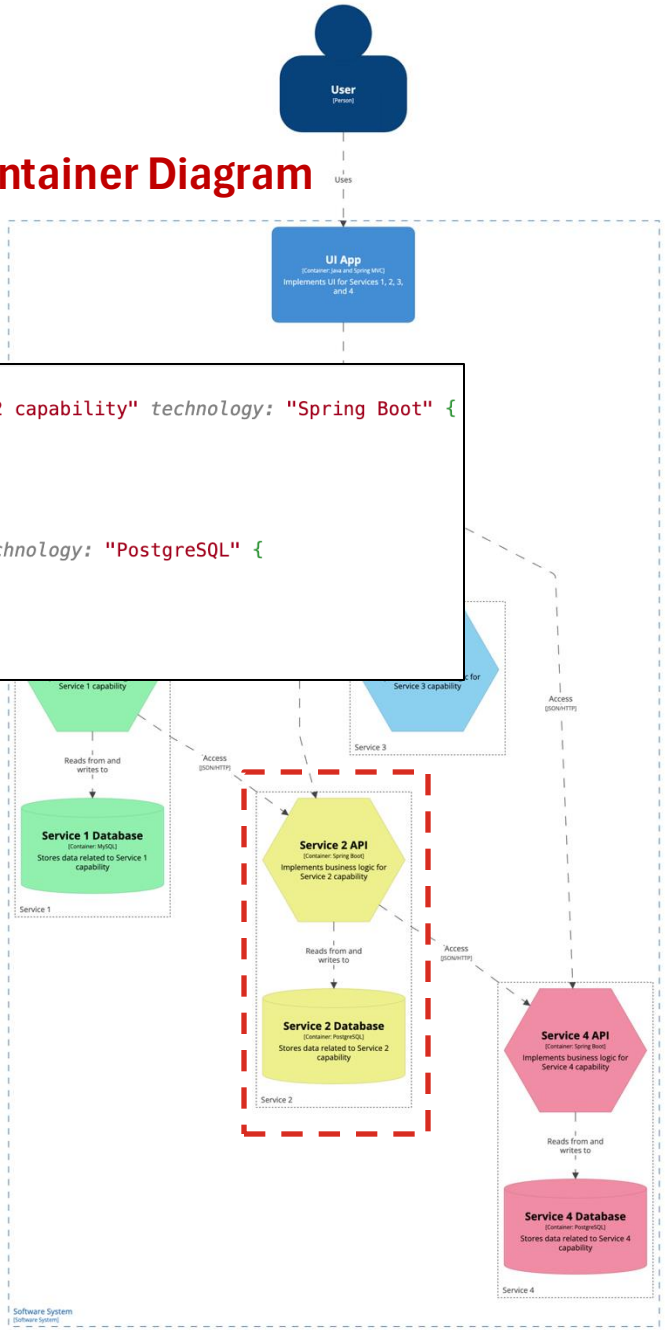

```
service1 = group "Service 1" {
    service1Api = container name: "Service 1 API" description: "Implements business logic for Service 1 capability" technology: "Spring Boot" {
        tags "Service 1" "Service"
        component1_1 = component name: "Component1"
        component1_2 = component name: "Component2"
        component1_3 = component name: "Component3"
    }
    service1database = container name: "Service 1 Database" description: "Stores data related to Service 1 capability" technology: "MySQL" {
        tags "Service 1" "Database"
        service1Api -> this description: "Reads from and writes to"
    }
}
```


[Container] Software System
Tuesday 23 September 2025 at 15:49 Australian Eastern Standard Time

Microservices Architecture with C4 Model

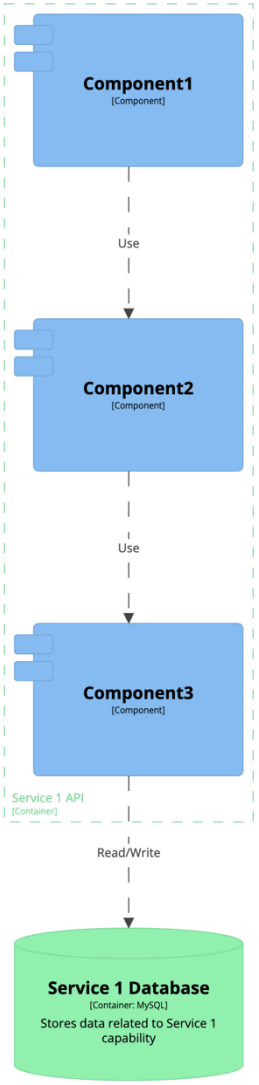
```
service2 = group "Service 2" {
    service2Api = container name: "Service 2 API" description: "Implements business logic for Service 2 capability" technology: "Spring Boot" {
        tags "Service 2" "Service"
        component2_1 = component name: "Component1"
        component2_2 = component name: "Component2"
    }
    container name: "Service 2 Database" description: "Stores data related to Service 2 capability" technology: "PostgreSQL" {
        tags "Service 2" "Database"
        service2Api -> this description: "Reads from and writes to"
    }
}
```

Container Diagram



Microservices Architecture with C4 Model

Component Diagram for Service 1



[Component] Software System - Service 1 API
Tuesday 23 September 2025 at 15:51 Australian Eastern Standard Time

Modelling Event-Driven Architecture with C4 Model

Event Driven Architecture

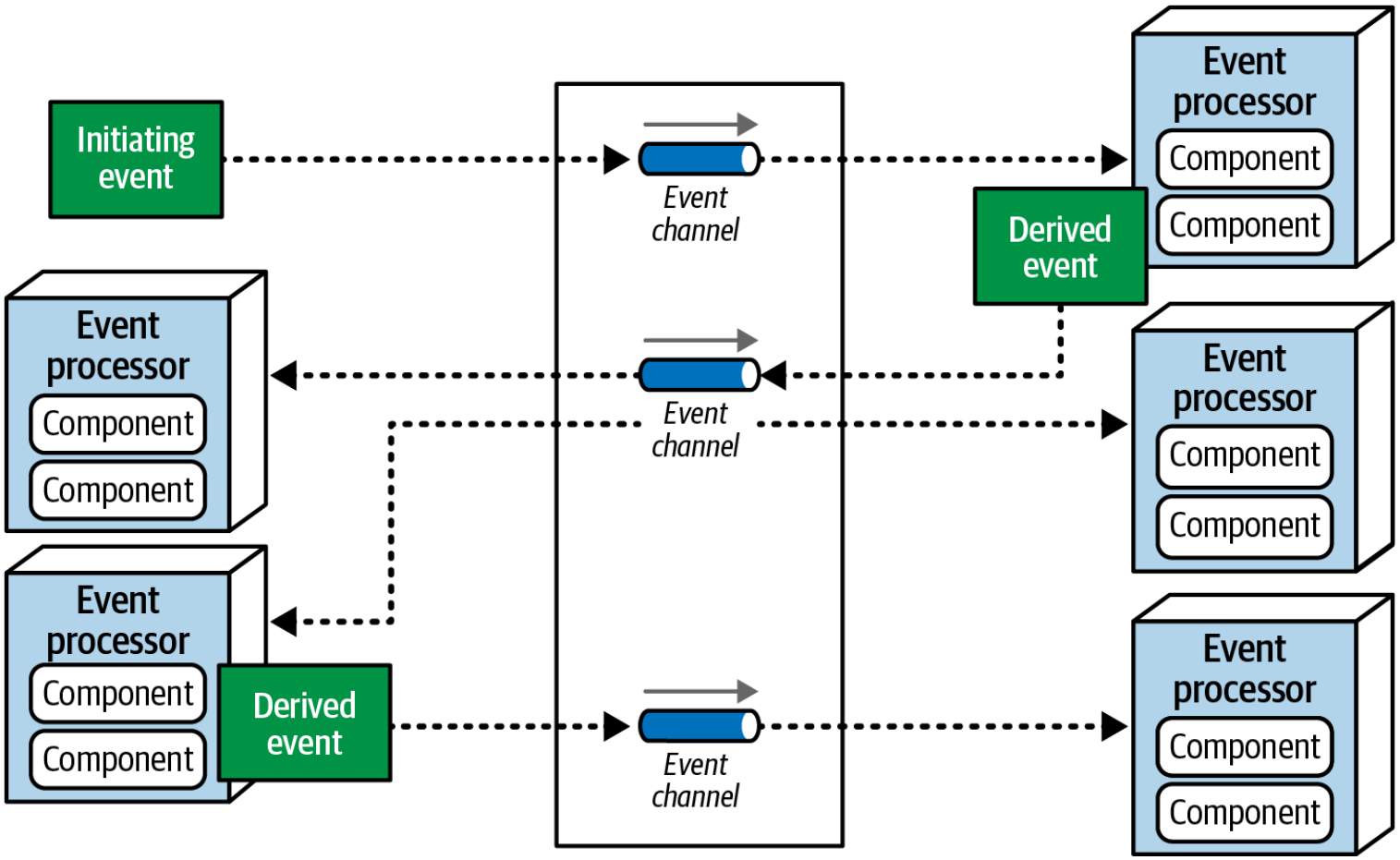
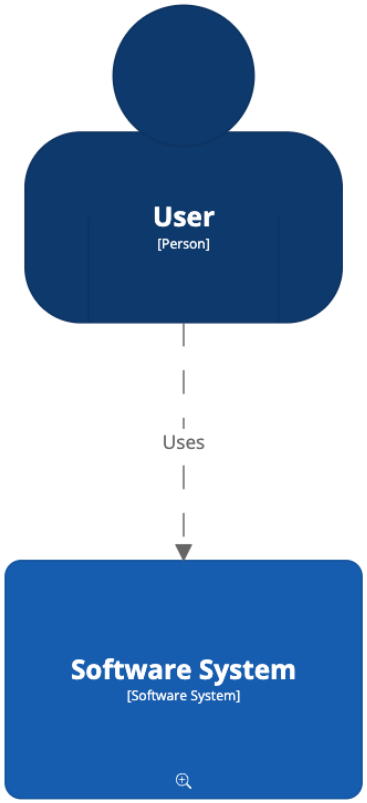


Image Source: Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2025.

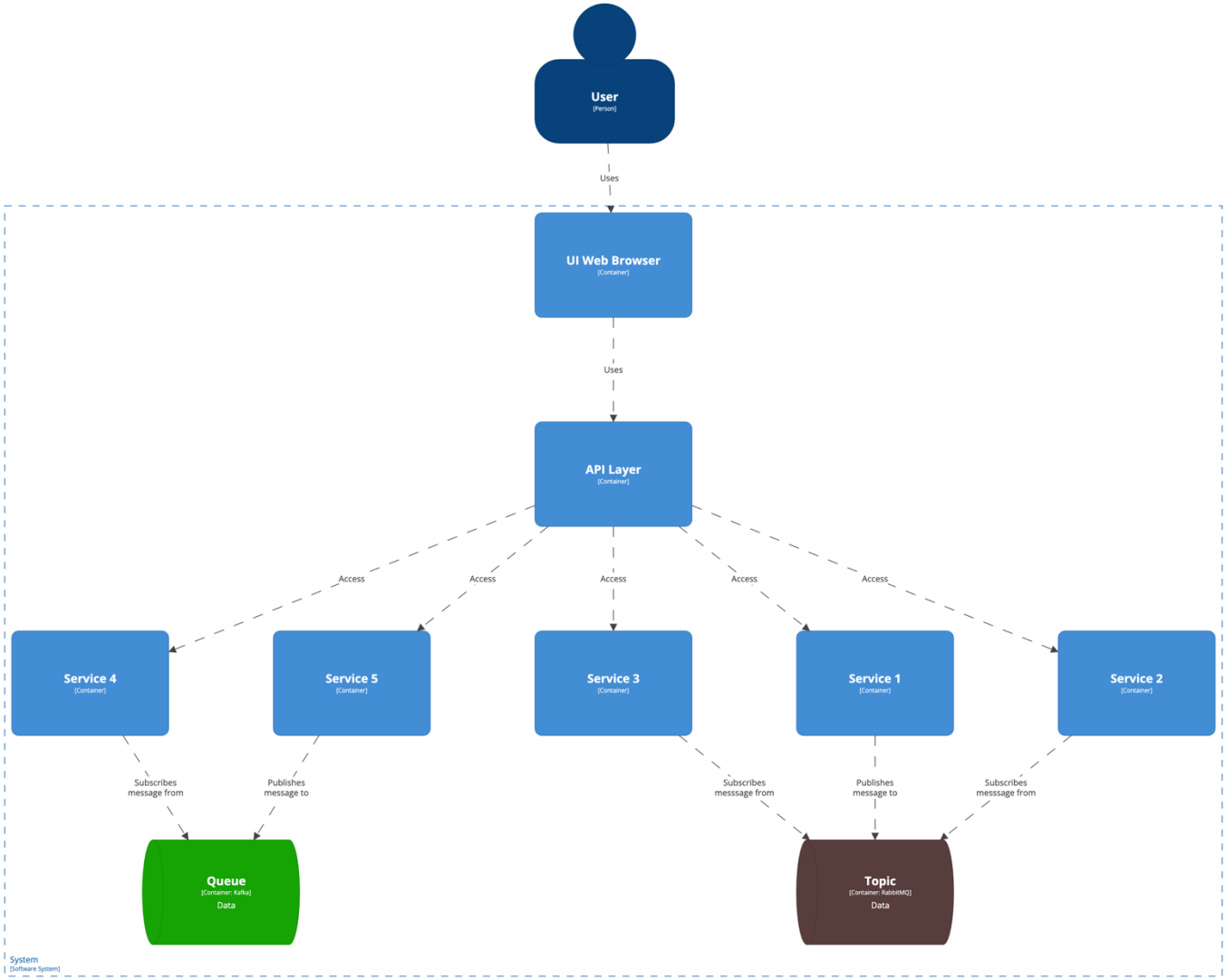
Event-driven Architecture with C4 Model

System Context Diagram



Event-driven Architecture with C4 Model

Container Diagram



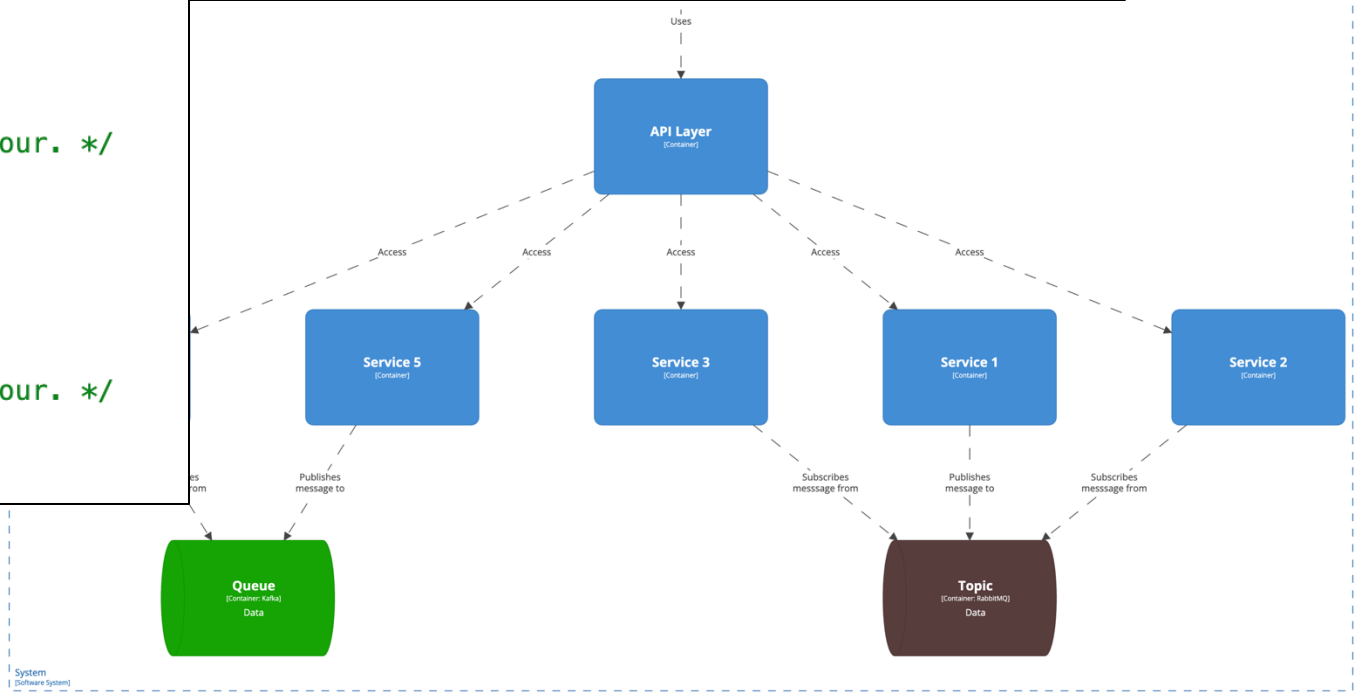
[Container] System
Tuesday 23 September 2025 at 16:02 Australian Eastern Standard Time

Event-driven Architecture with C4 Model

Container Diagram

```
rabbitMQ = container name: "Topic" description: "Data" technology: "RabbitMQ" {
    tags "Topic"
}
kafka = container name: "Queue" description: "Data" technology: "Kafka" {
    tags "Queue"
}
```

```
element "Topic" {
    shape Pipe
    background #583D3D
    /* colour is text colour. */
    colour #FFFFFF
}
 element "Queue" {
    shape Pipe
    background #16A304
    /* colour is text colour. */
    colour #FFFFFF
}
```



[Container] System
Tuesday 23 September 2025 at 16:02 Australian Eastern Standard Time

References

- Pautasso, C., Software Architecture- Visual Lecture Notes, LeanPub, 2023 (https://leanpub.com/software-architecture/)
- Bass, L., Clements, P., Kazman, R., Software Architecture in Practice, Addison-Wesley, 2021.
- Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2020 (First Edition).
- Richards, M., Ford, N., Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, 2025 (Second Edition).
- Gandhi, R., Richards, M., Ford, N., Head First Software Architecture, O'Reilly Media, Inc. 2024
- Humberto, C., and Kazman R., Designing Software Architectures: A Practical Approach. Second Edition, Addison-Wesley Professional, 2024.
- Building a GraphQL service (https://spring.io/guides/gs/graphql-server)
- Introduction to gRPC (https://grpc.io/docs/what-is-grpc/introduction/)
- What Is a REST API? Examples, Uses, and Challenges (https://blog.postman.com/rest-api-examples/)
- Richard N. Taylor, Nenad Medvidovic, Eric M. Dashofy, Software Architecture: Foundations, Theory and Practice, John-Wiley, January 2009, ISBN 978047016774
  - https://www.softwarearchitecturebook.com/resources/