

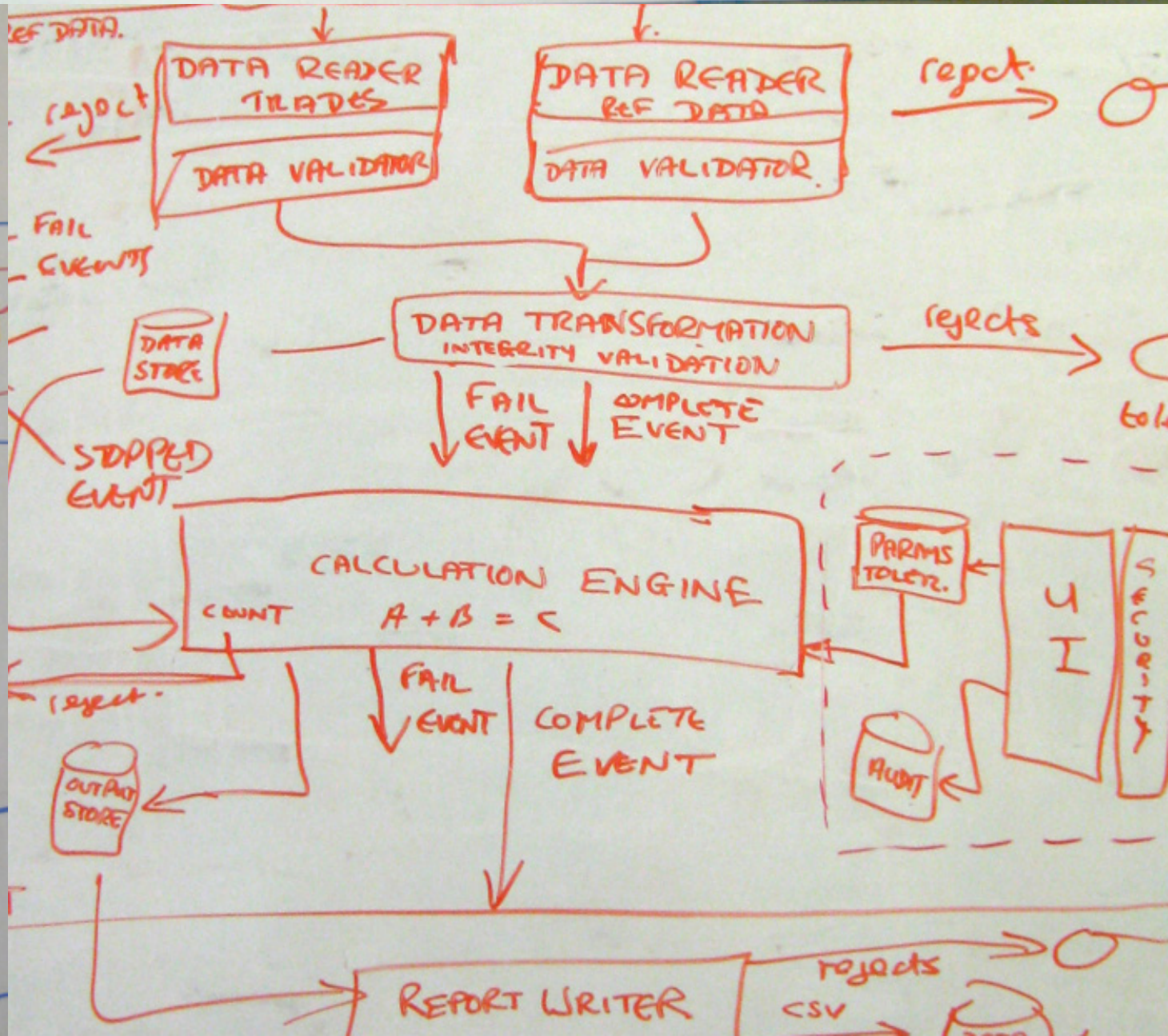
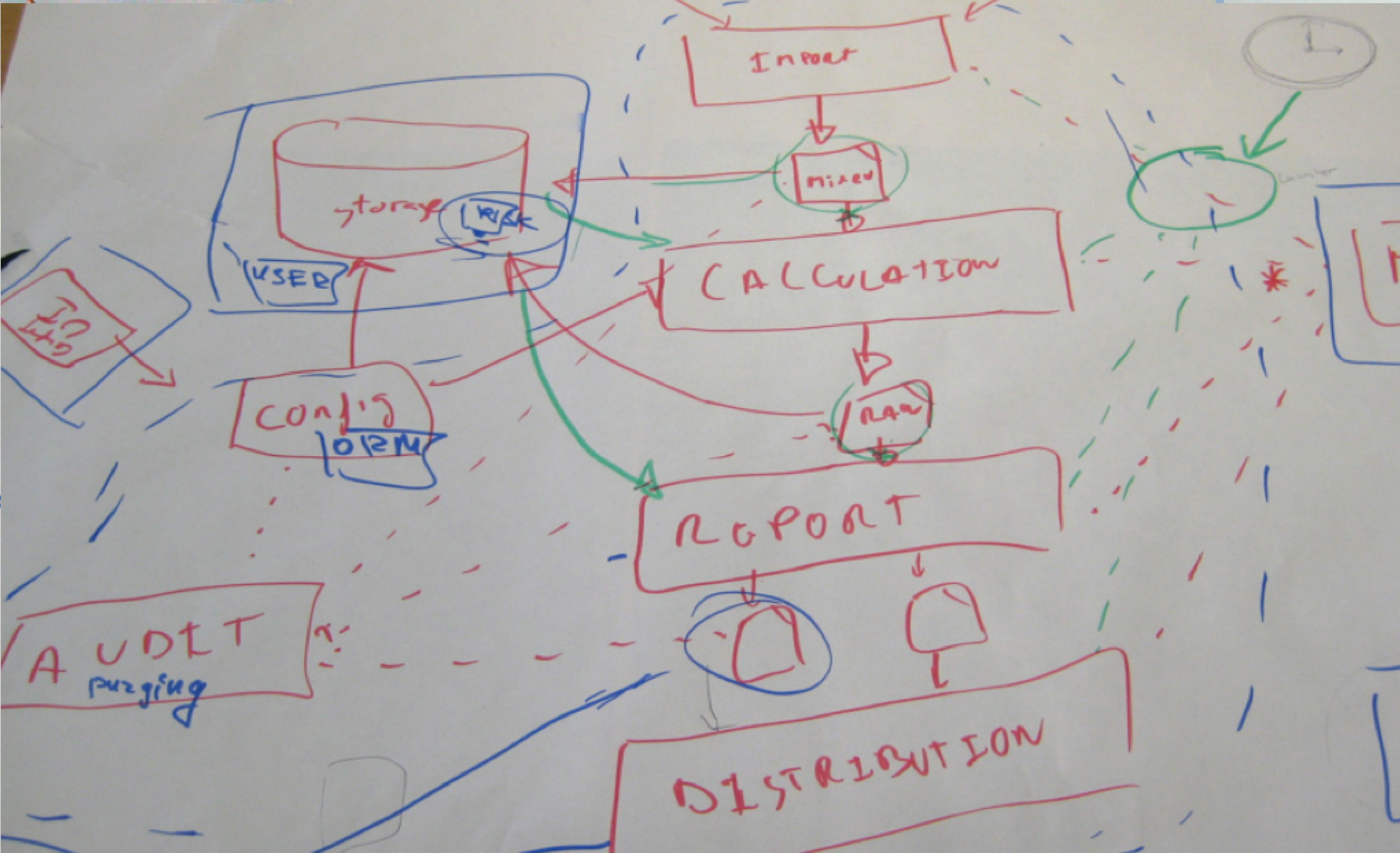
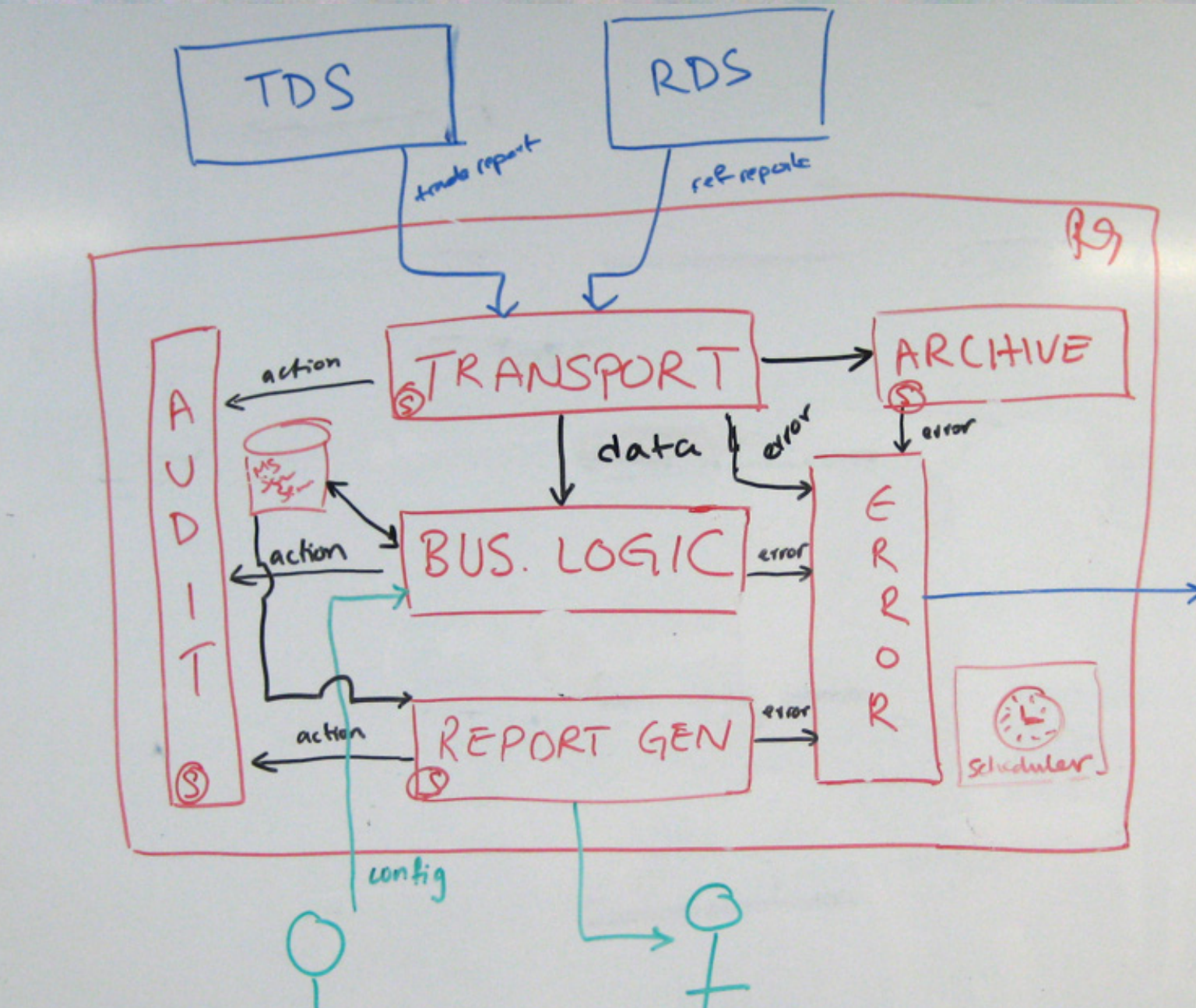
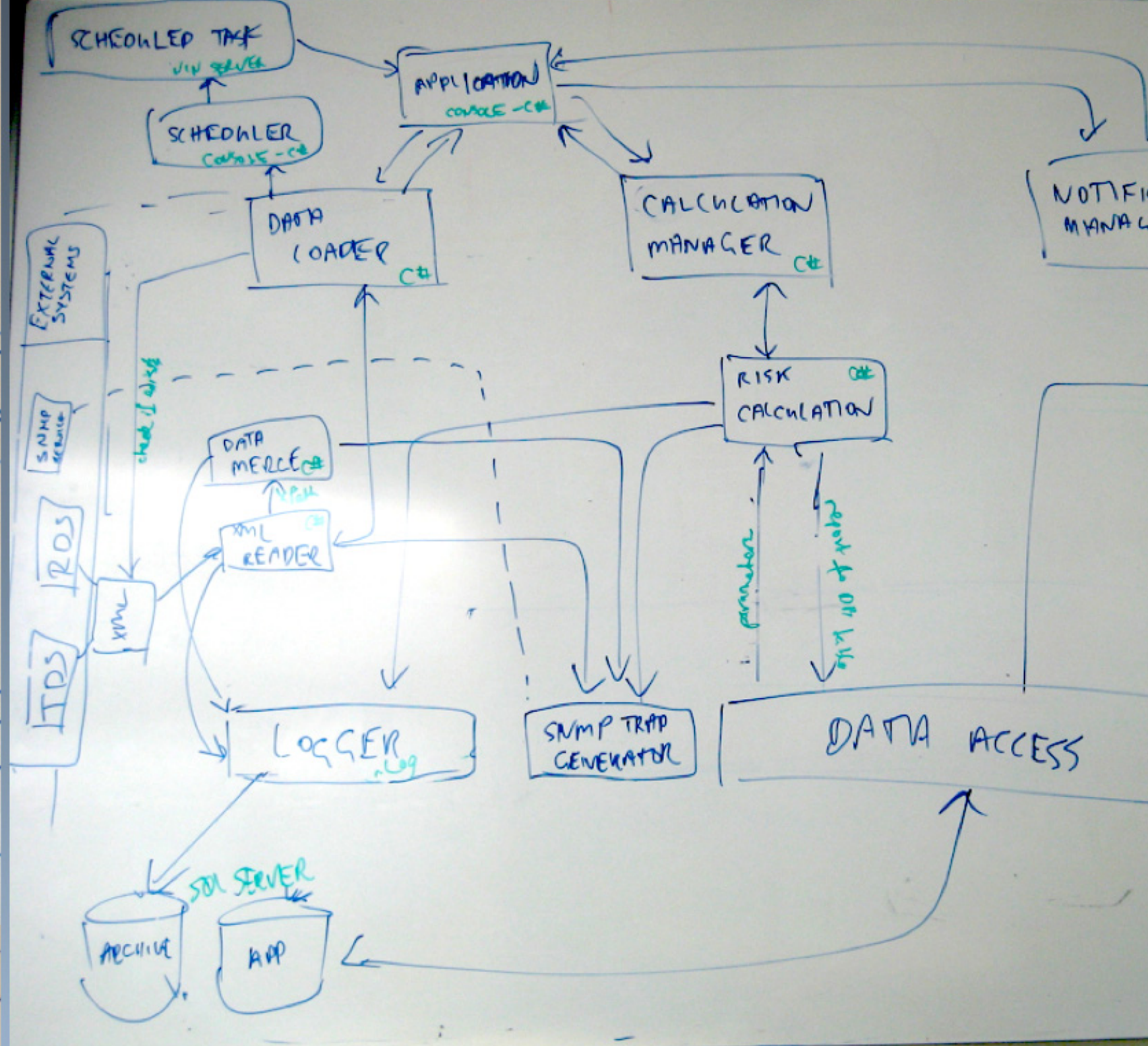
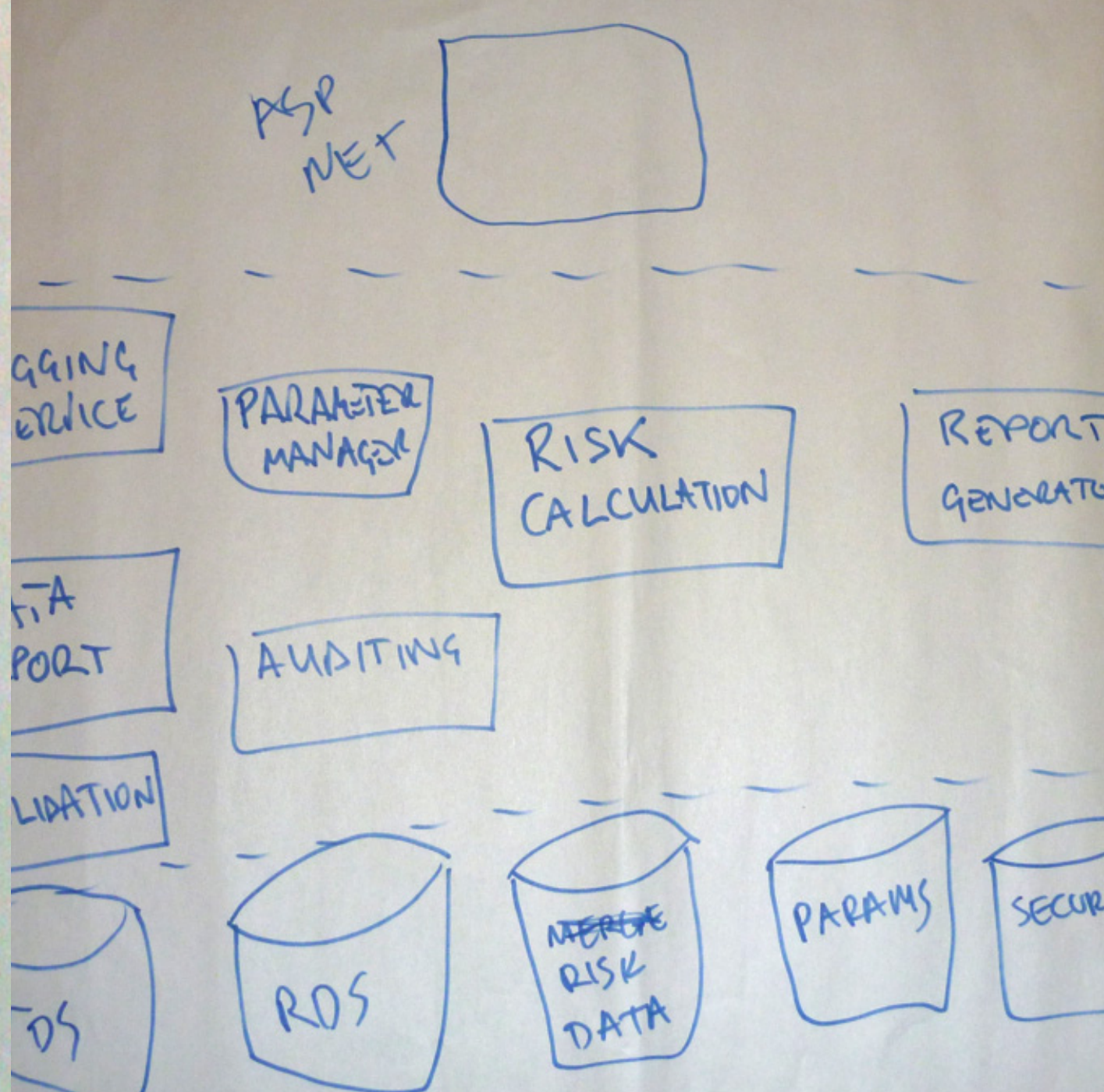
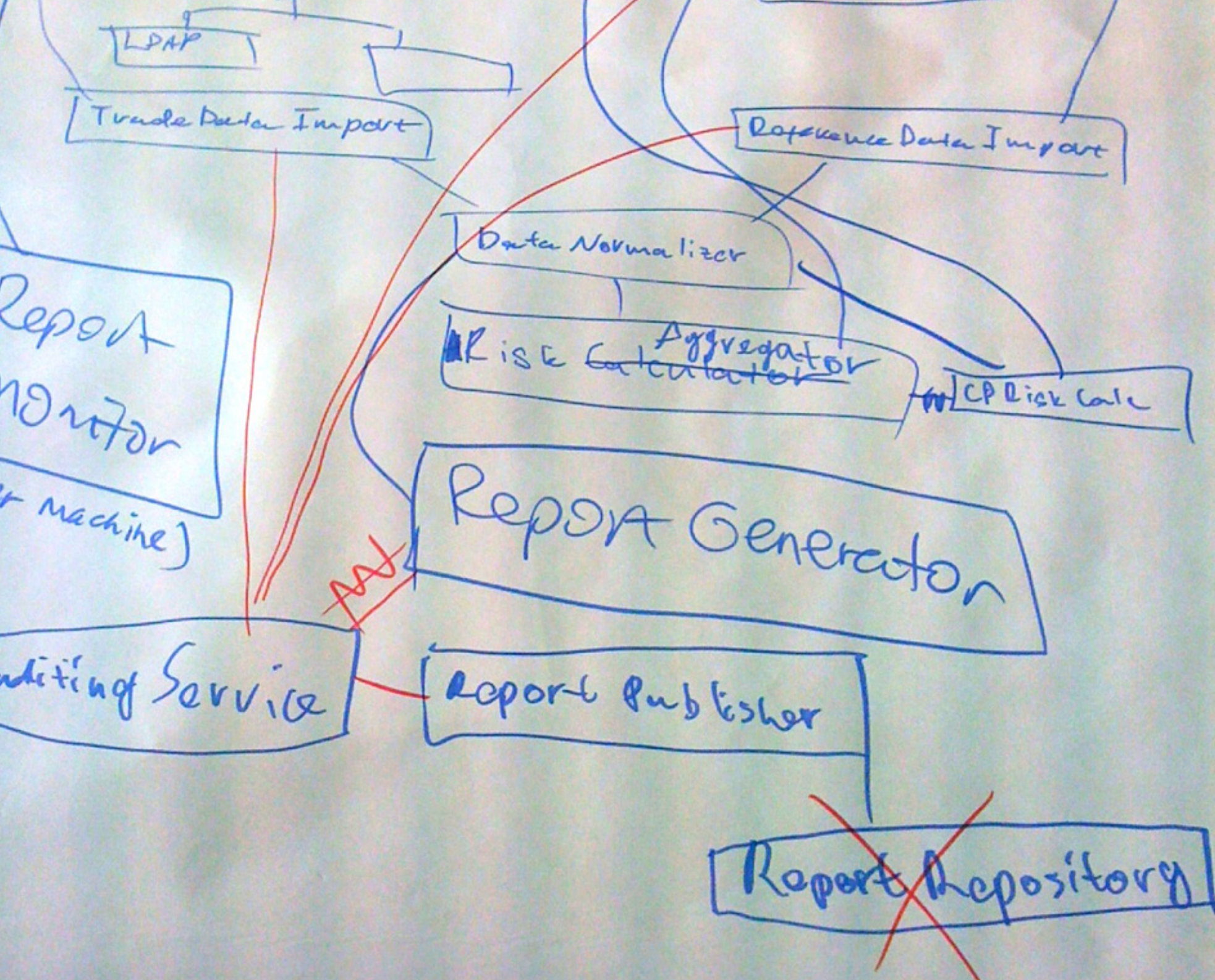
# The C4 model

Misconceptions, misuses, and mistakes

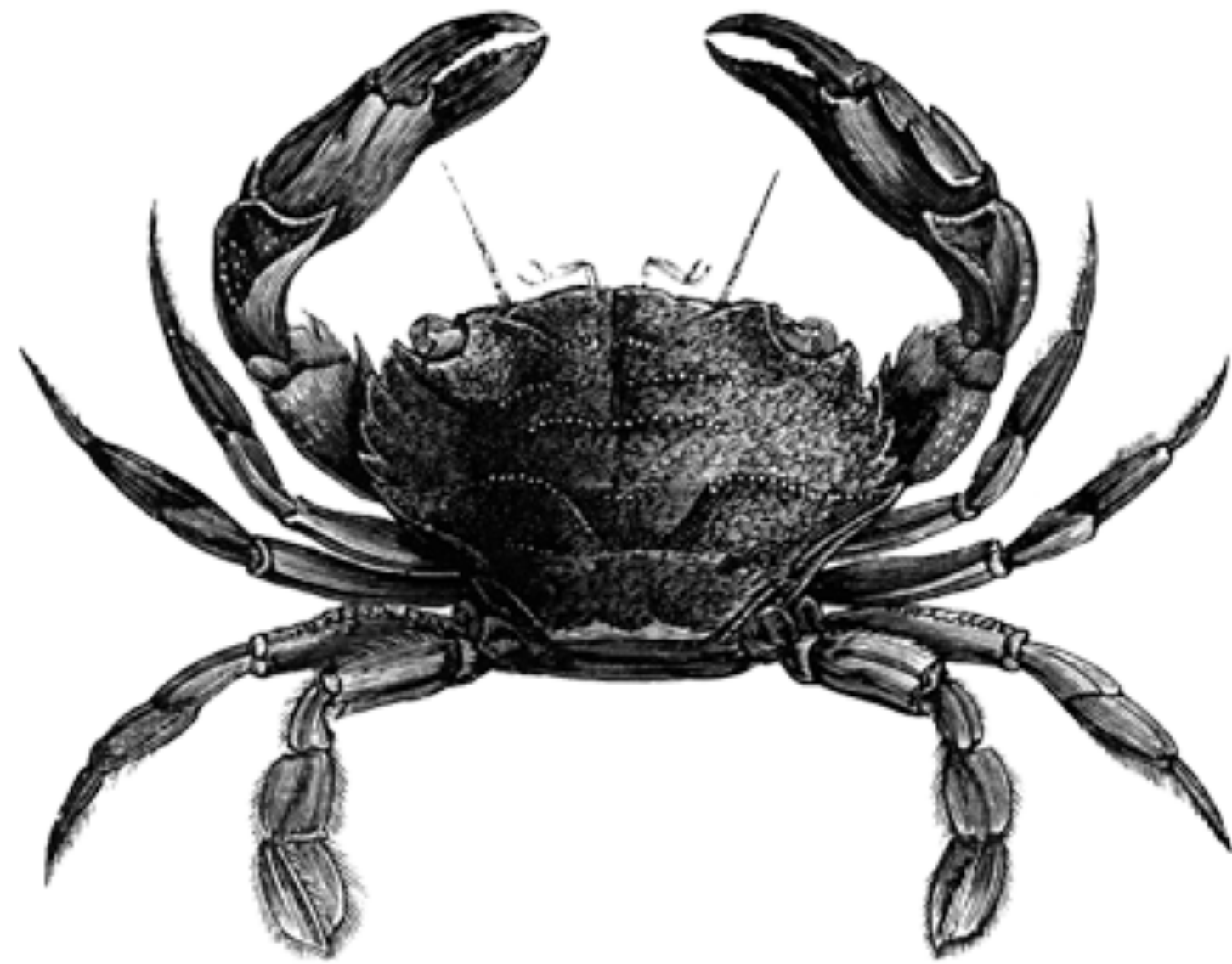
Simon Brown

What is the C4 model?









# 97 Ways to Sidestep UML

#2 "Not everybody else on the team knows it."

#3 "I'm the only person on the team who knows it."

#36 "You'll be seen as old."

#37 "You'll be seen as old-fashioned."

#66 "The tooling sucks."

#80 "It's too detailed."

#81 "It's a very elaborate waste of time."

#92 "It's not expected in agile."

#97 "The value is in the conversation."



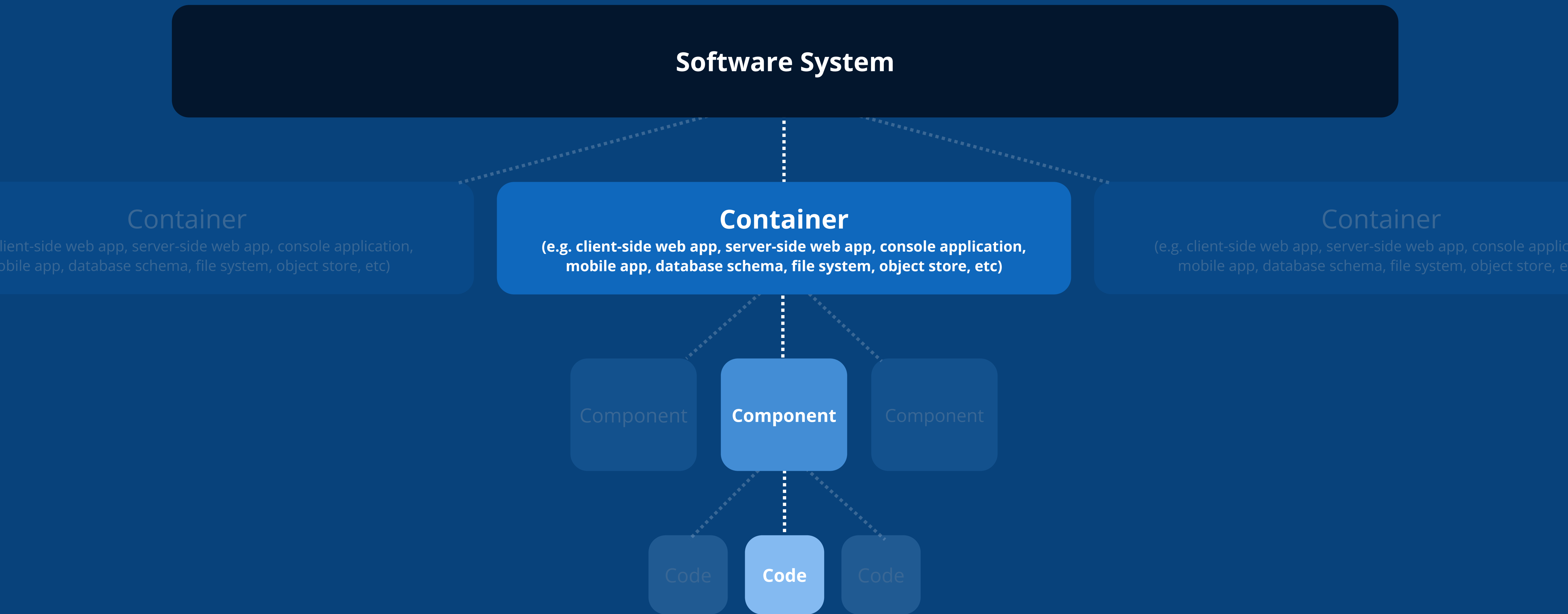
# C4

[c4model.com](http://c4model.com)



A way to introduce some **structure**  
to “boxes & lines” diagrams,  
using a **self-describing notation**



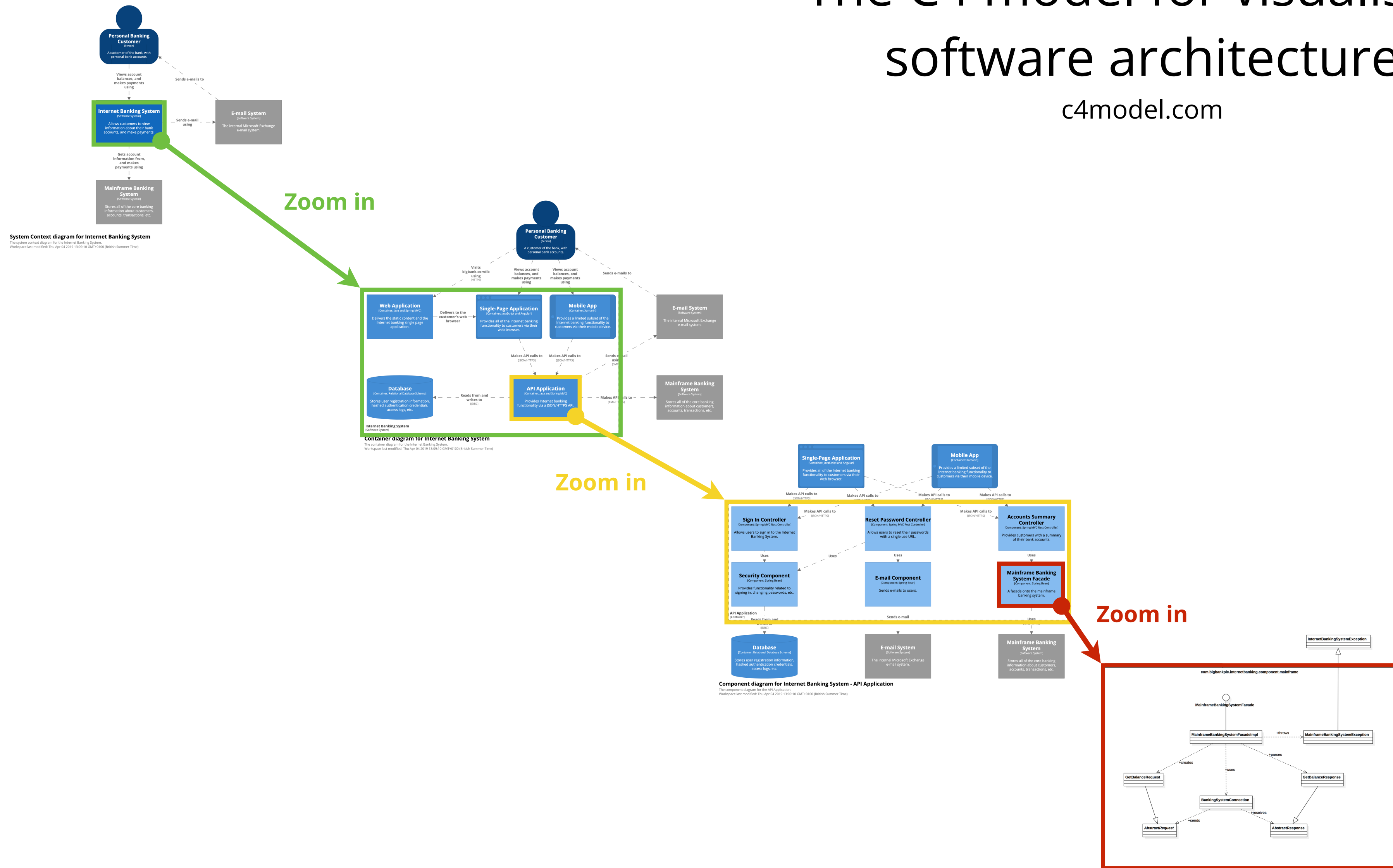


A **software system** is made up of one or more **containers** (applications and data stores), each of which contains one or more **components**, which in turn are implemented by one or more **code** elements (classes, interfaces, objects, functions, etc).



# The C4 model for visualising software architecture

c4model.com



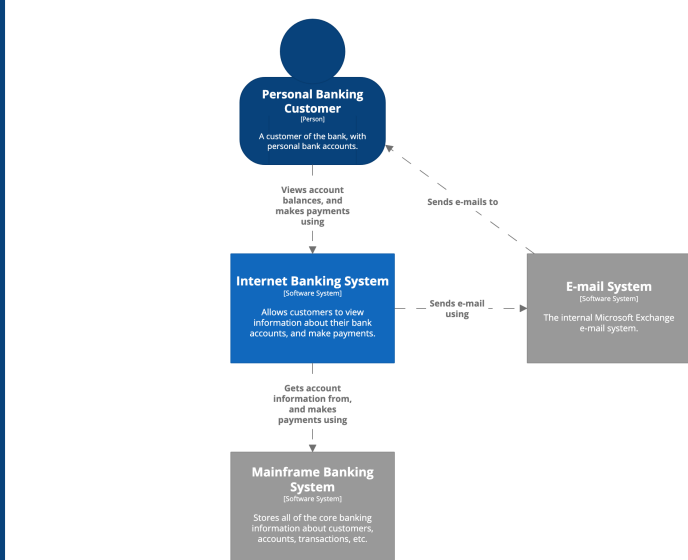
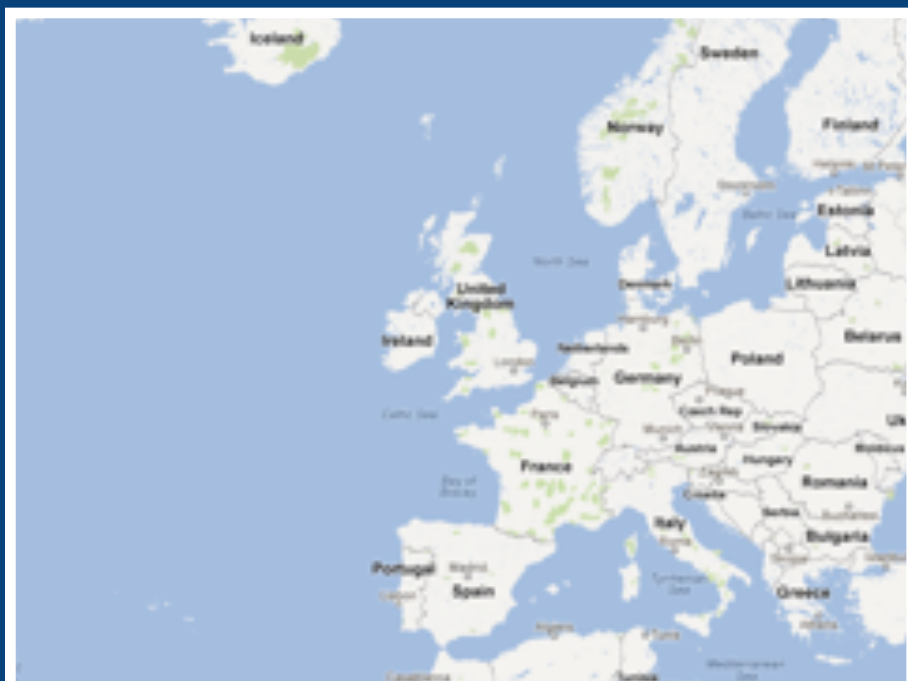
Level 1  
Context

Level 2  
Containers

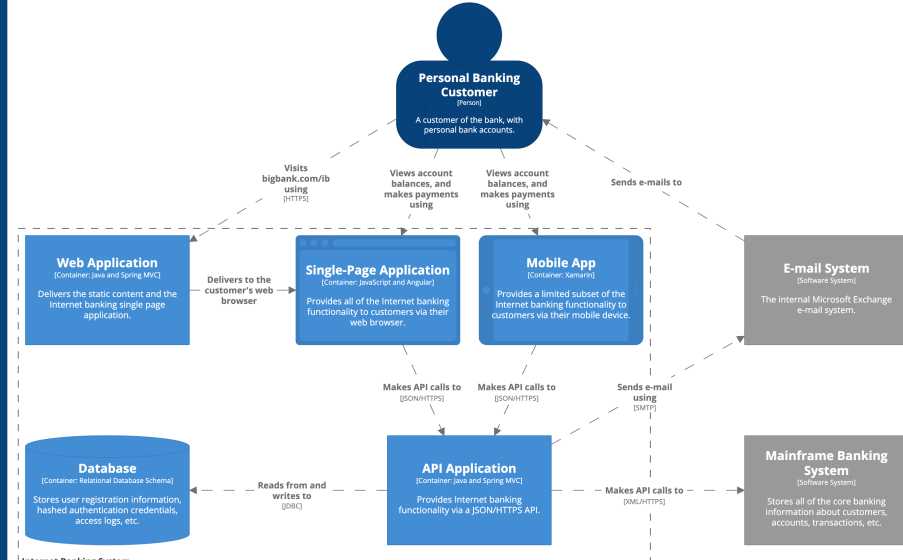
Level 3  
Components

Level 4  
Code

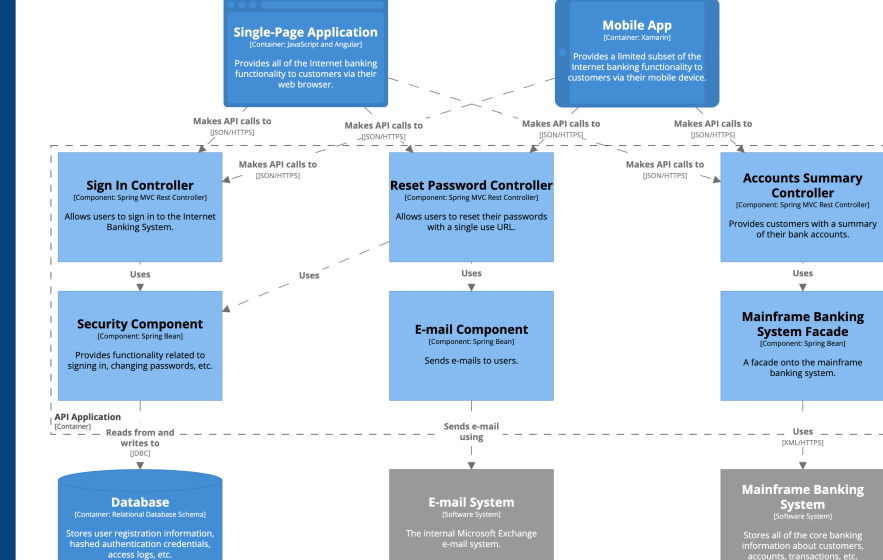




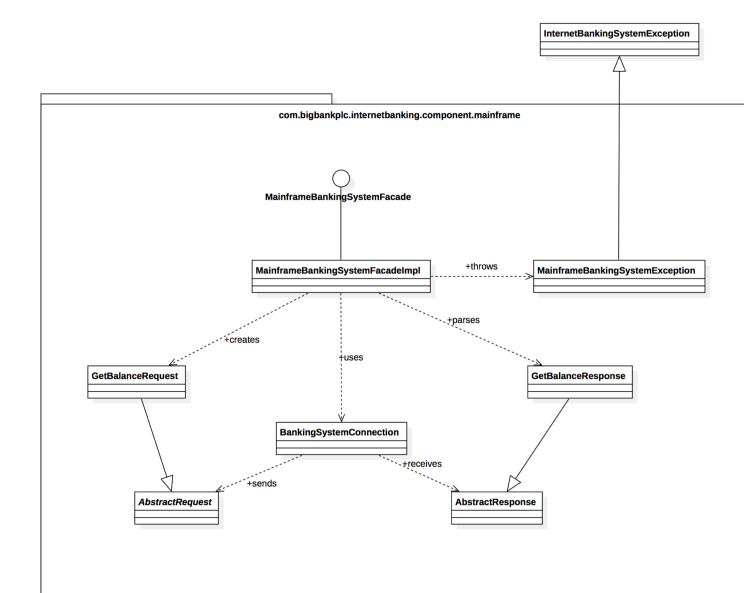
System Context diagram for Internet Banking System  
The system context diagram for the Internet Banking System.  
Workspace last modified: Thu Apr 04 2019 13:05:10 GMT+0100 (British Summer Time)



Container diagram for Internet Banking System  
The container diagram for the Internet Banking System.  
Workspace last modified: Thu Apr 04 2019 13:05:10 GMT+0100 (British Summer Time)



Component diagram for Internet Banking System - API Application  
The component diagram for the API Application.  
Workspace last modified: Thu Apr 04 2019 13:05:10 GMT+0100 (British Summer Time)





# System Context diagram

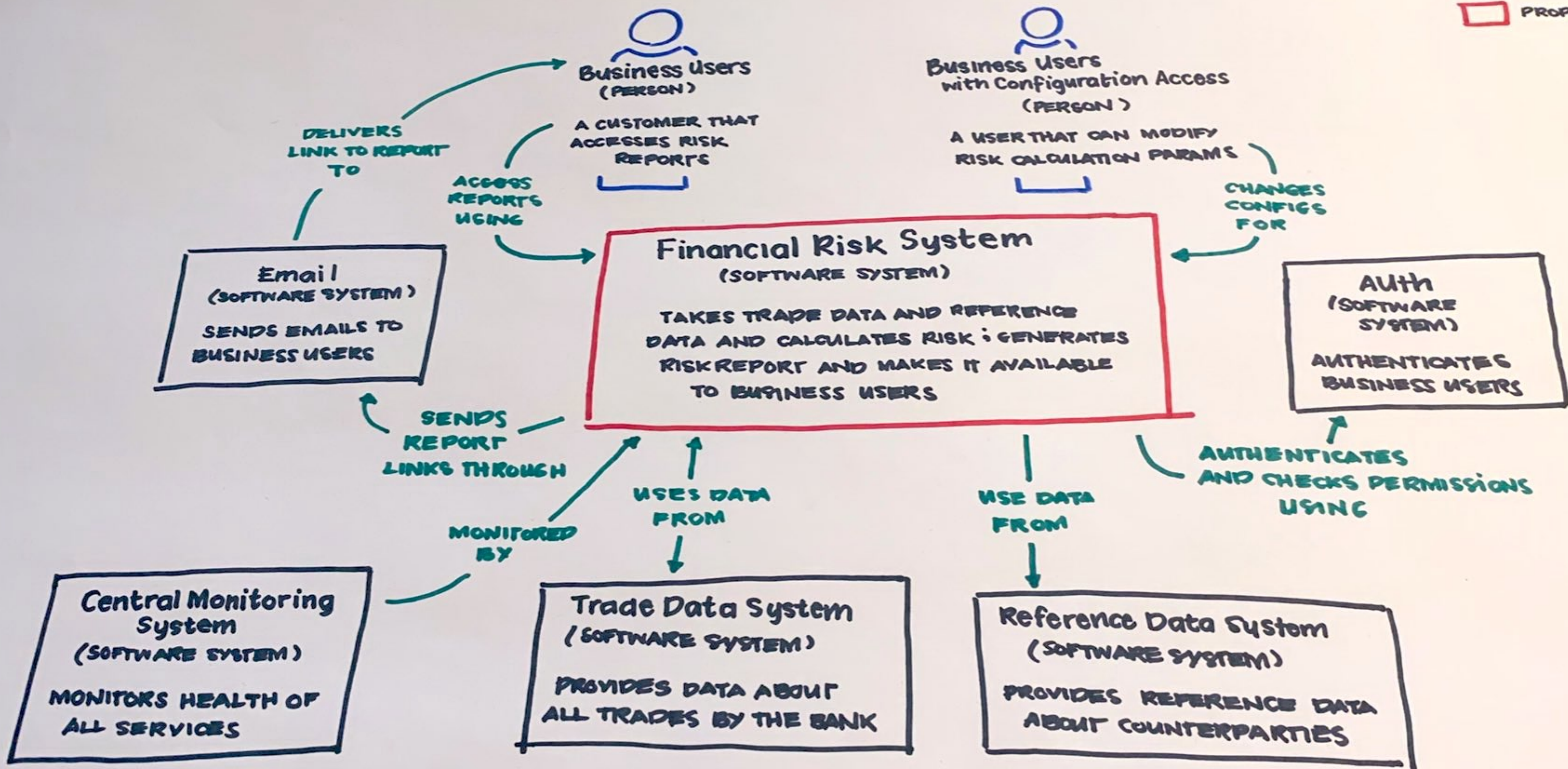
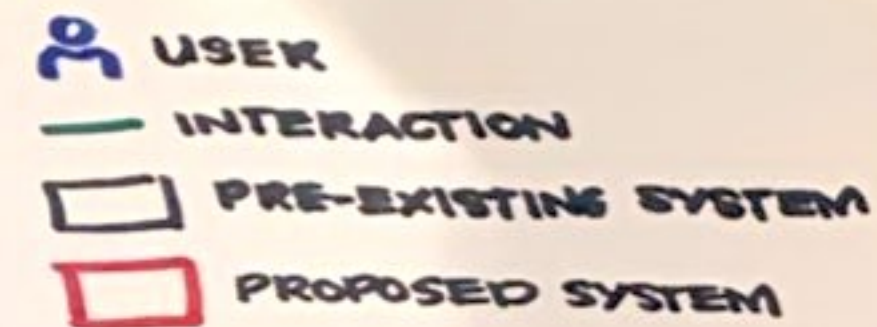
What is the scope of the software system we're building?

Who is using it? What are they doing?

What system integrations does it need to support?



# Financial Risk System: Context Diagram





# Container diagram

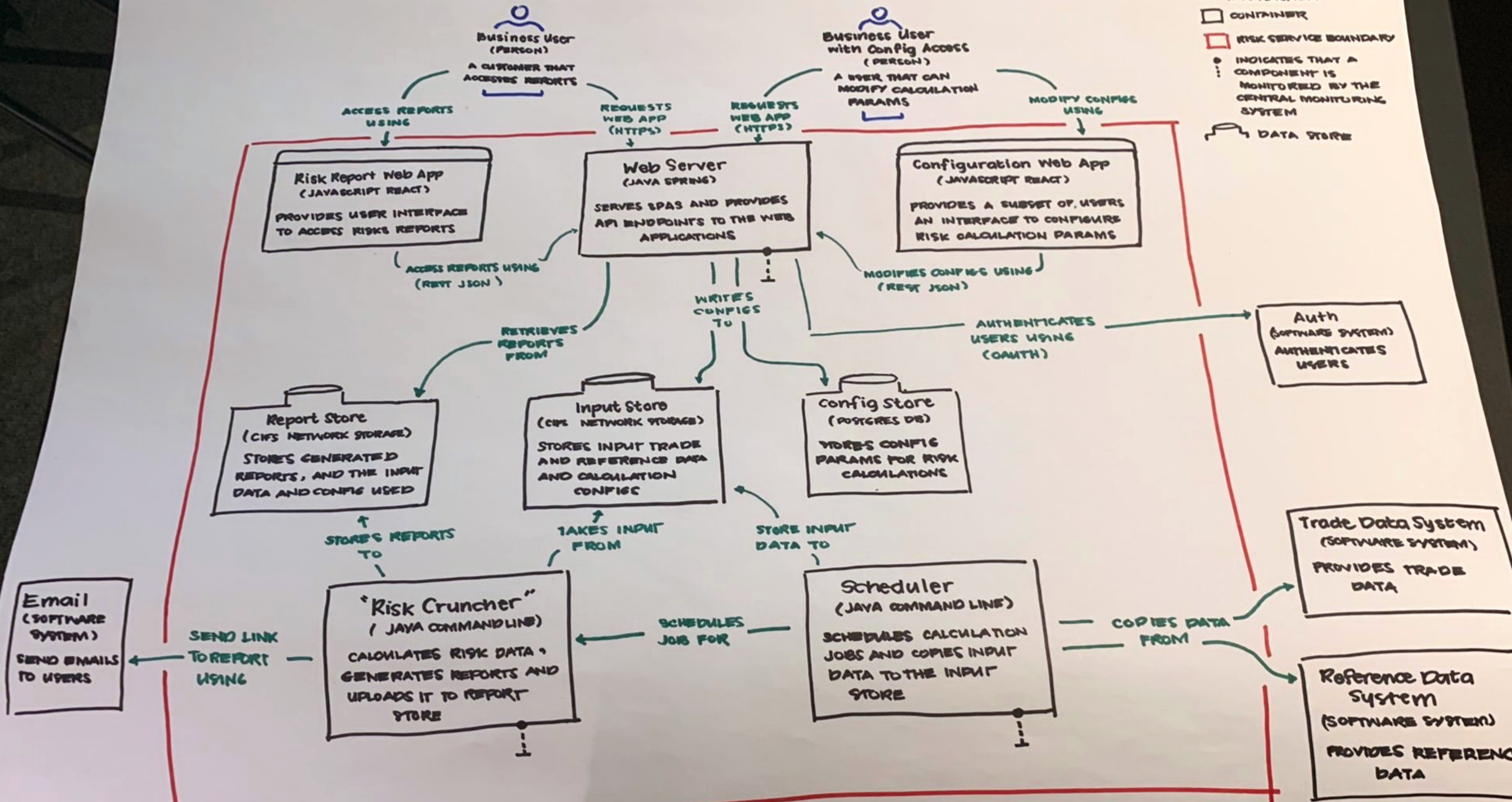
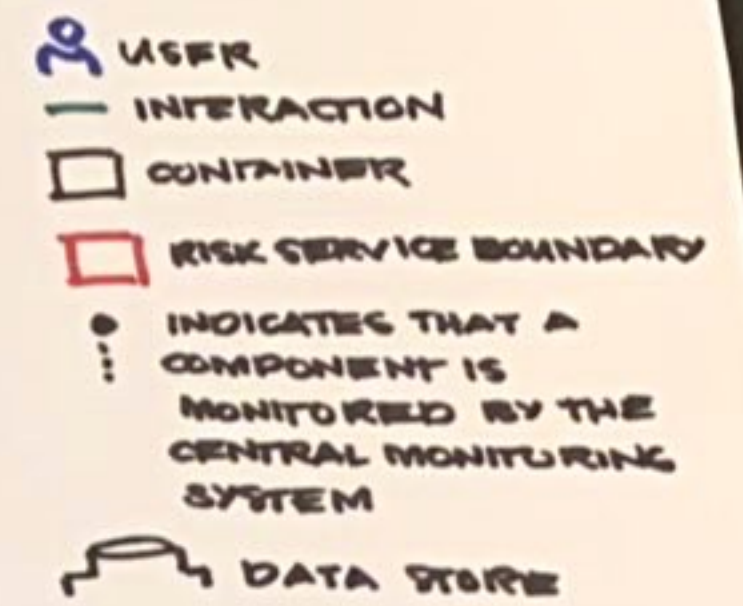
What are the major technology building blocks?

What are their responsibilities?

How do they communicate?



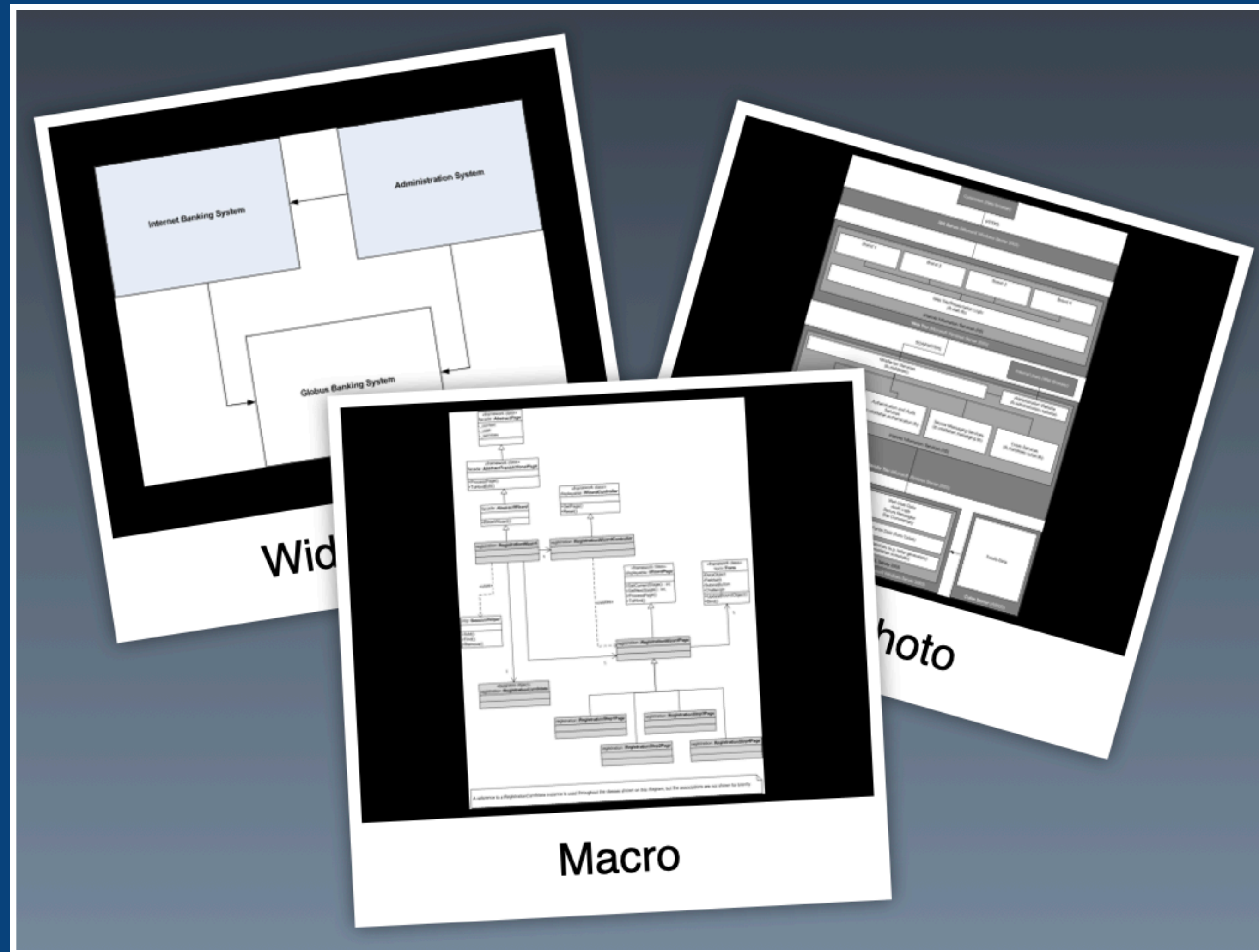
# Financial Risk System: Container Diagram





I've only just heard of the  
C4 model - I guess it's new?





“Software architecture for developers”

QCon London 2010



Start with the  
**big** picture

Telephoto (components)

“Where do you start?”

IASA London, March 2010





“Software architecture for developers”

QCon London 2011

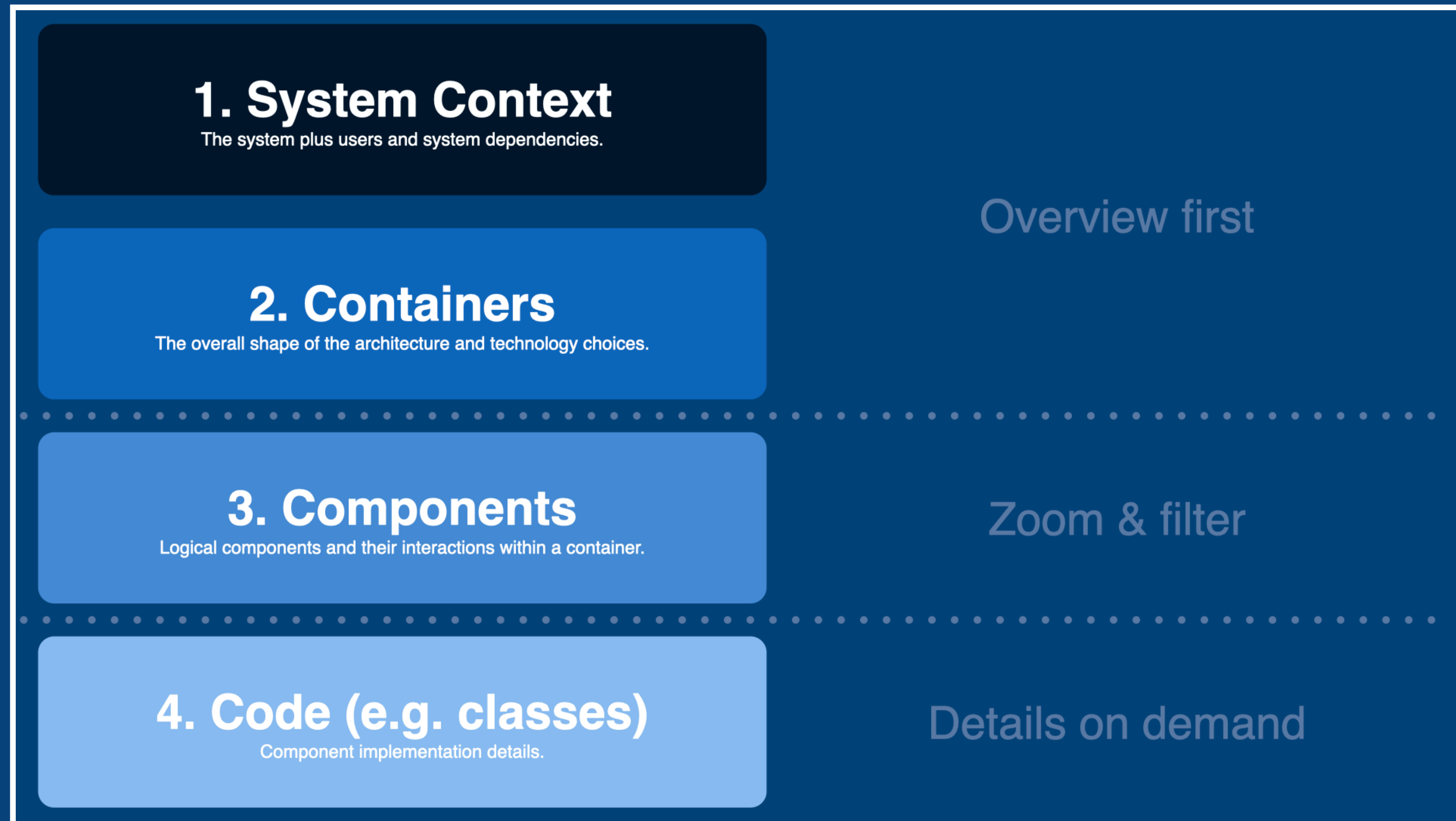


Diagrams are maps  
that help a team navigate a complex codebase

“Software architecture as code”

Devoxx UK 2015



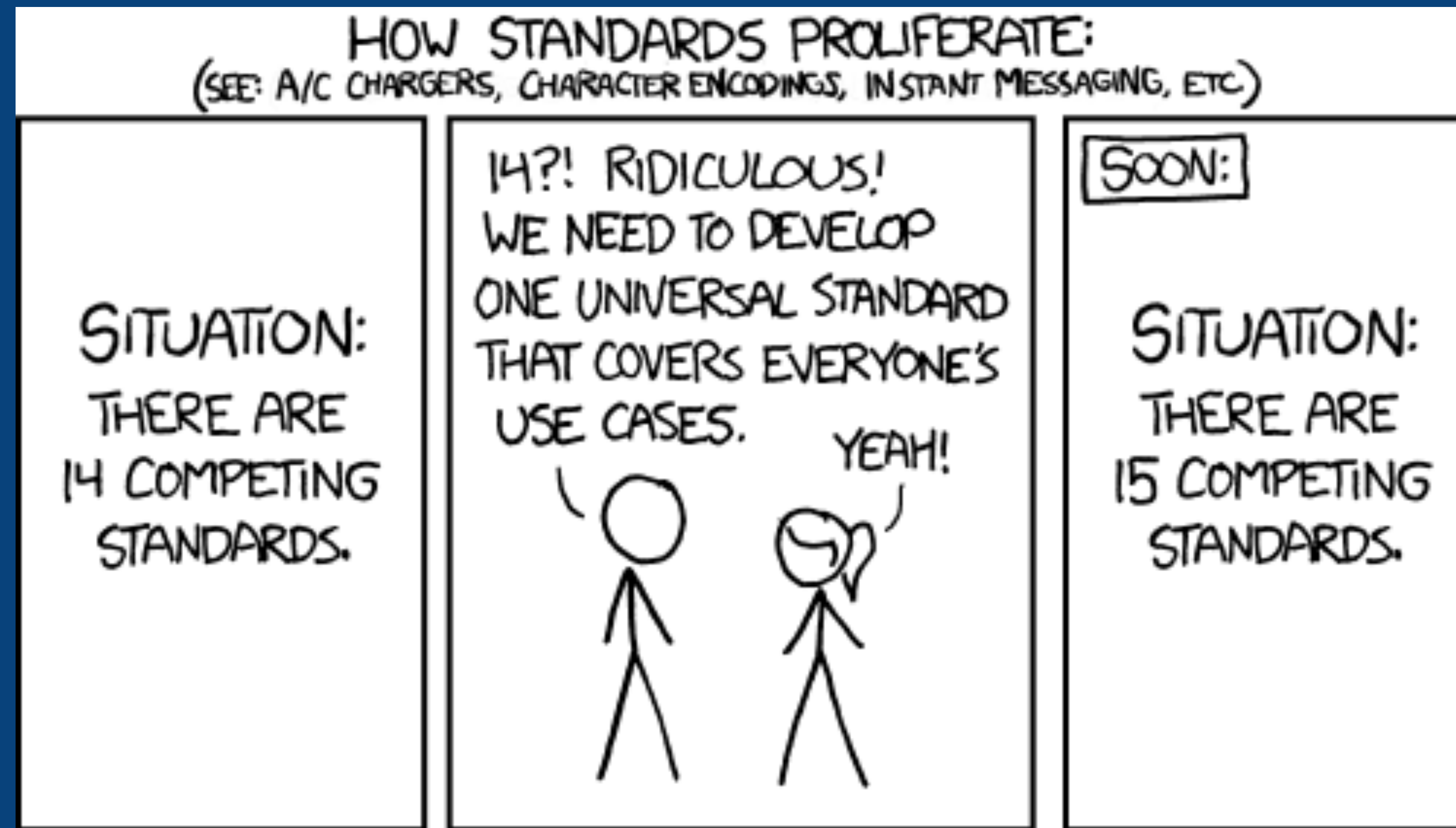


“Visualise, document, and explore your software architecture”

Build Stuff Spain 2018

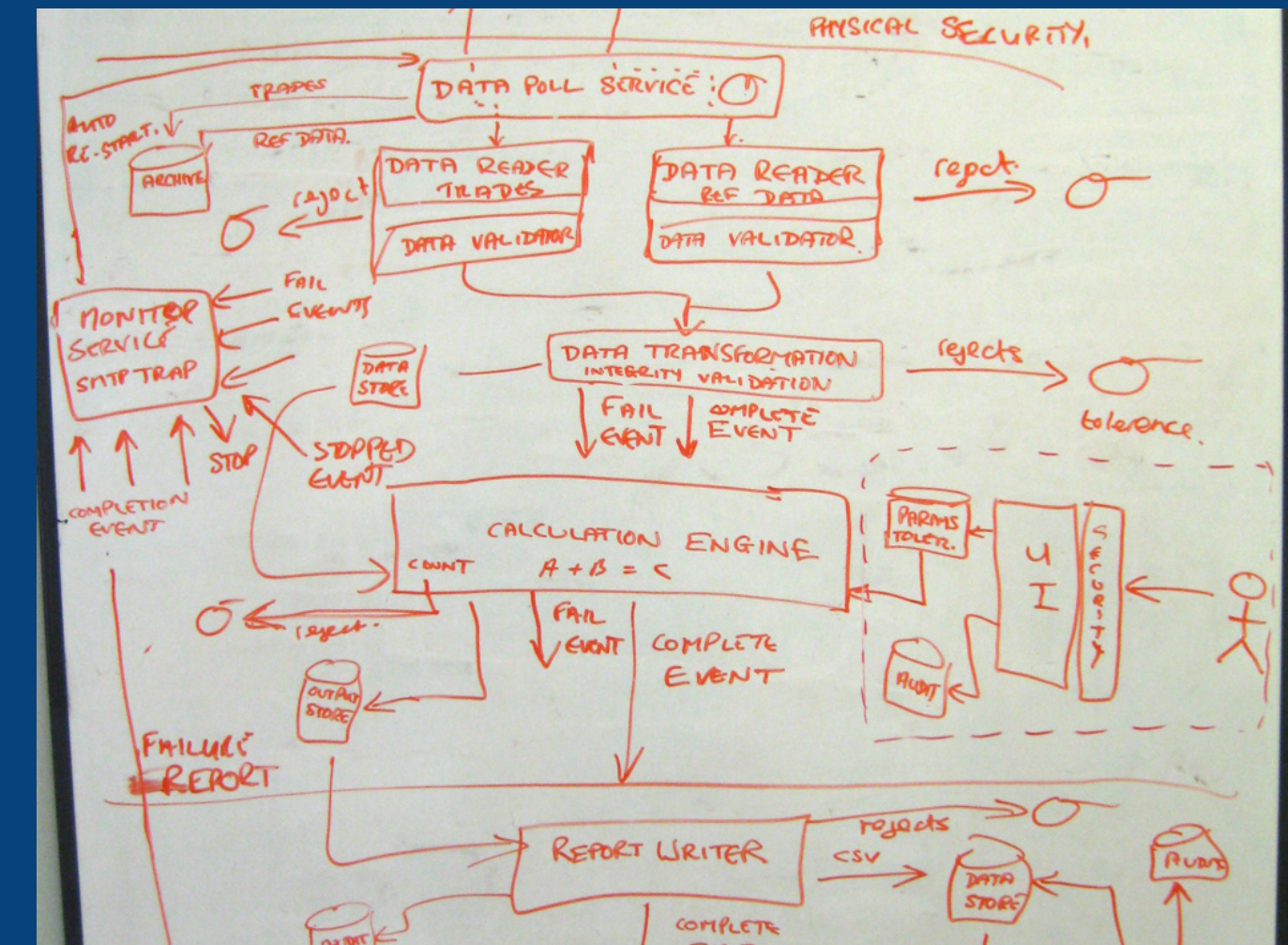
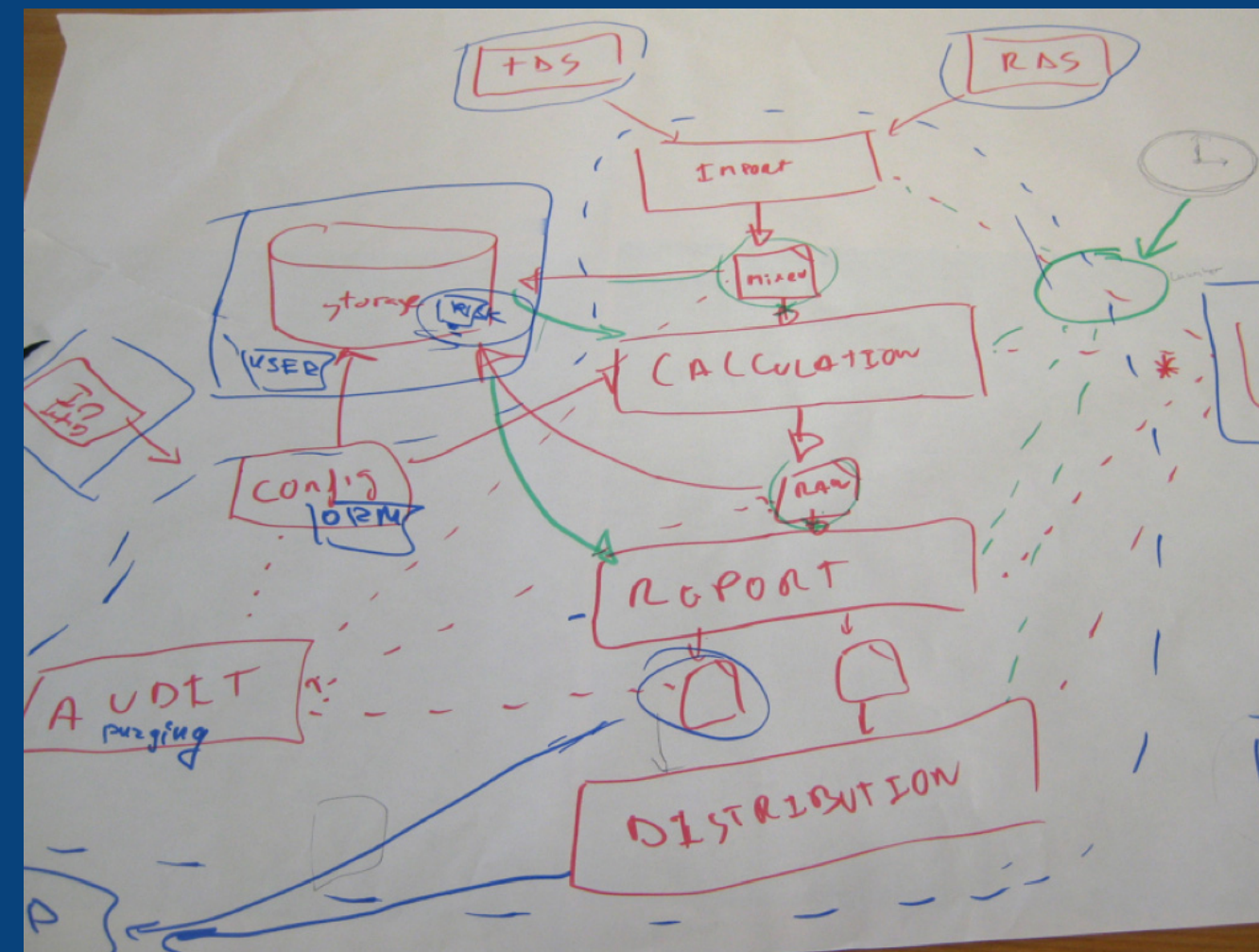
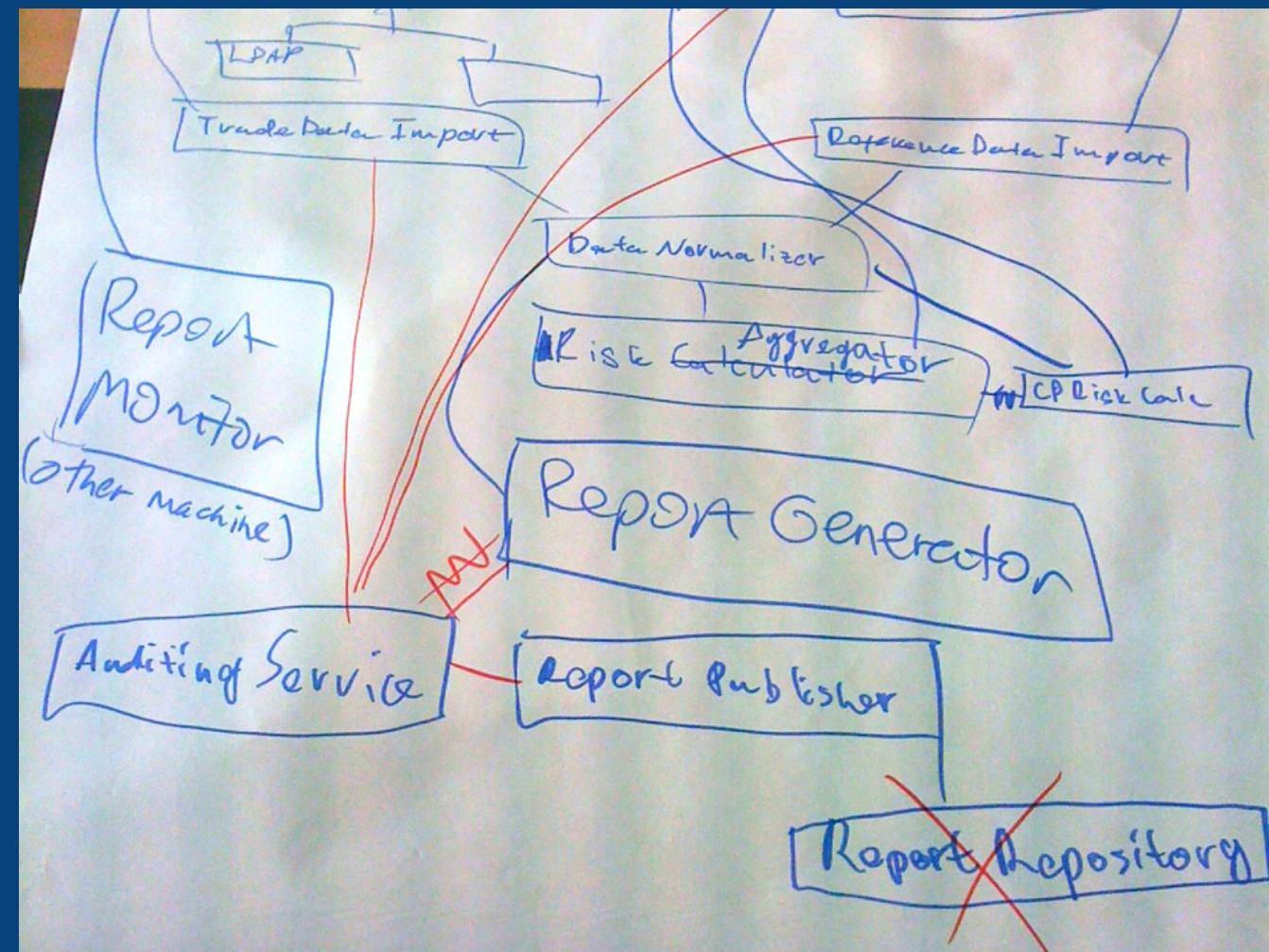


# Why did you reinvent UML?



C4 wasn't designed  
to replace UML





C4 was designed to bring structure to the typical ad hoc "boxes and arrows" diagrams teams typically create because they are no longer using UML



# The C4 model is...

A set of hierarchical  
abstractions

(software systems, containers,  
components, and code)

A set of hierarchical  
diagrams

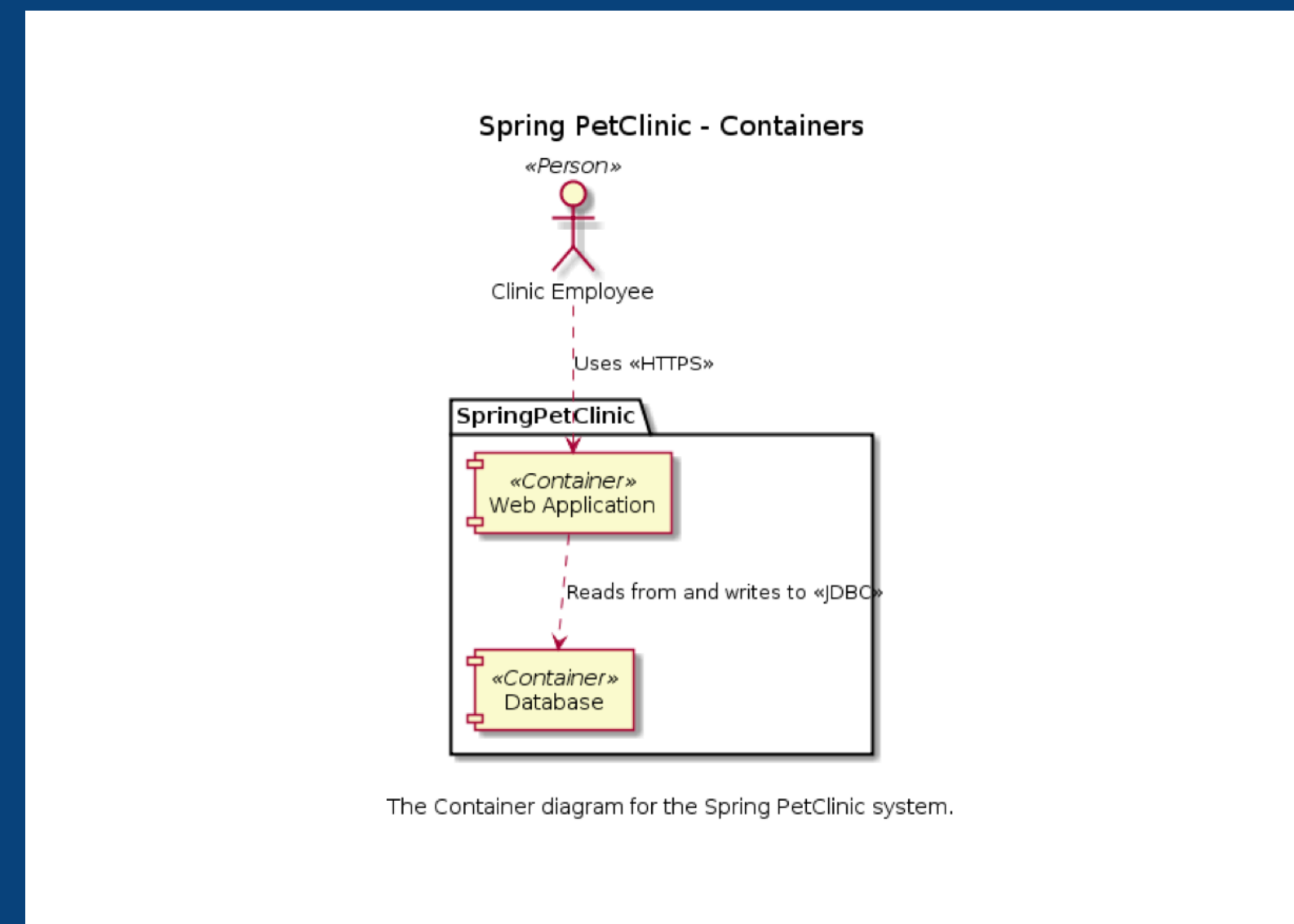
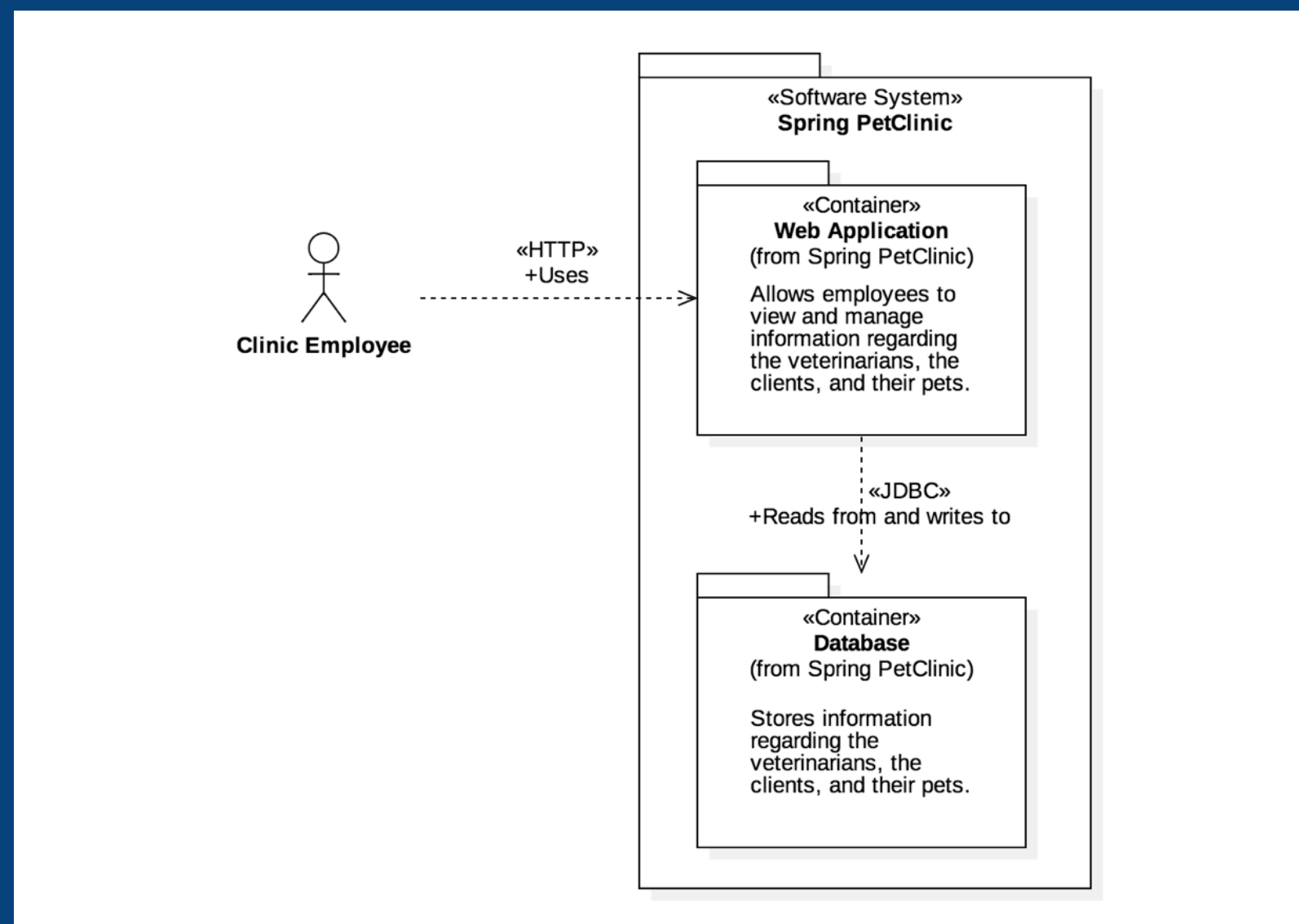
(system context, containers, components,  
and code)

Notation independent

Tooling independent



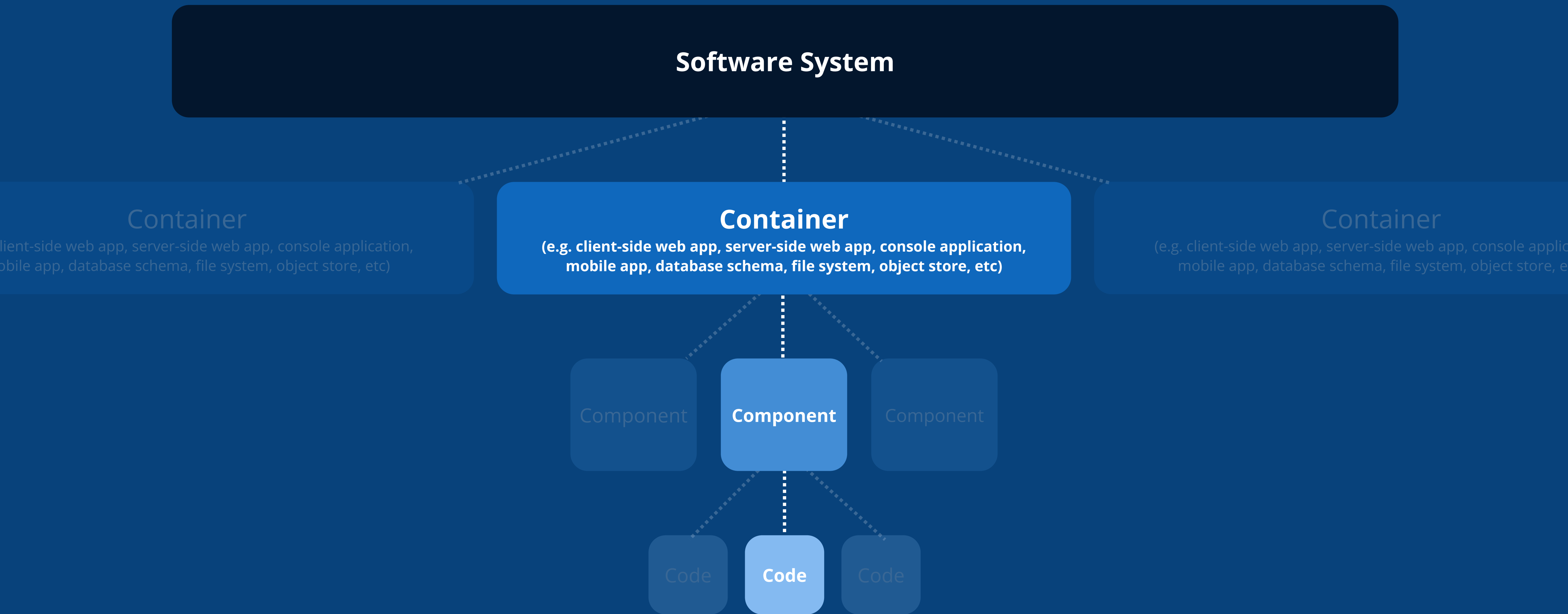
# UML + C4 model abstractions and diagram types





The background features four light blue parallelograms arranged in two pairs, one pair on the left and one pair on the right, framing the central text.


Few people use level 4.  
Why not just call it C3?



A **software system** is made up of one or more **containers** (applications and data stores), each of which contains one or more **components**, which in turn are implemented by one or more **code** elements (classes, interfaces, objects, functions, etc).



# Notation



The blue and grey  
notation is boring



# The C4 model is...

A set of hierarchical  
abstractions

(software systems, containers,  
components, and code)

A set of hierarchical  
diagrams

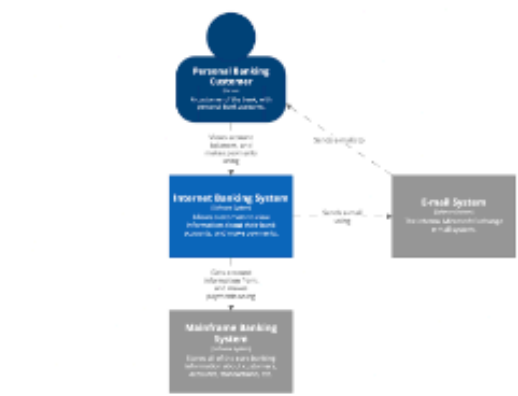
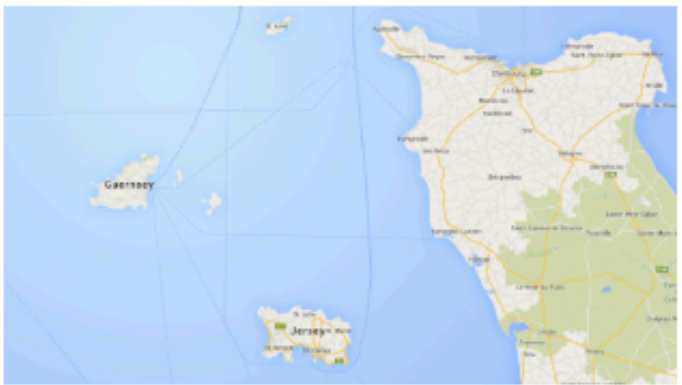
(system context, containers, components,  
and code)

Notation independent

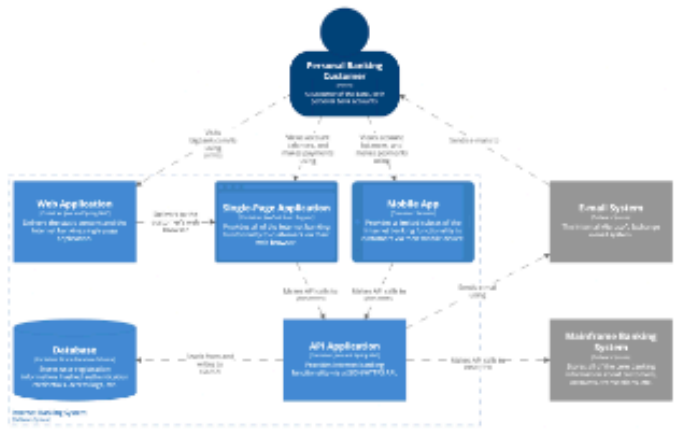
Tooling independent

Maps of your code

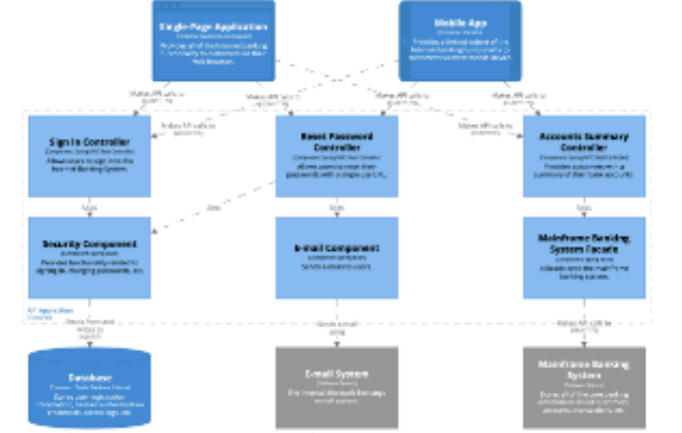
The C4 model was created as a way to help software development teams describe and communicate software architecture, both during up-front design sessions and when retrospectively documenting an existing codebase. It's a way to create "maps of your code", at various levels of detail, in the same way you would use something like Google Maps to zoom in and out of an area you are interested in.



[System Context] Internet Banking System



[Container] Internet Banking System



[Component] Internet Banking System - Web Application



Level 1: A [system context diagram](#) provides a starting point, showing how the software system in scope fits into the world around it.

Level 2: A [container diagram](#) zooms into the software system in scope, showing the applications and data stores inside it.

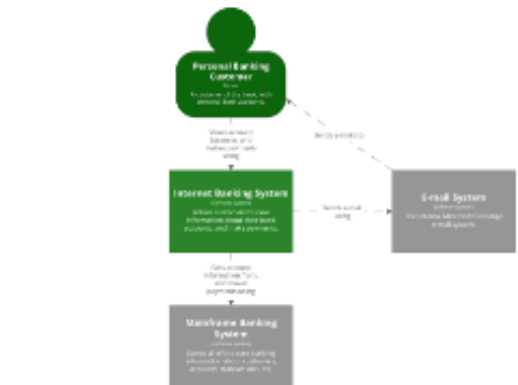
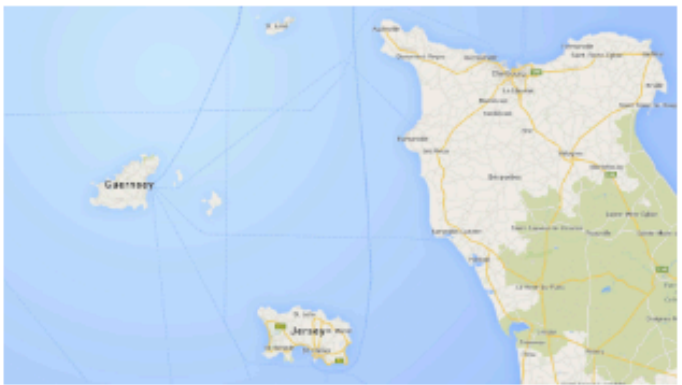
Level 3: A [component diagram](#) zooms into an individual container, showing the components inside it.

Level 4: A [code diagram](#) (e.g. UML class) can be used to zoom into an individual component, showing how that component is implemented at the code level.

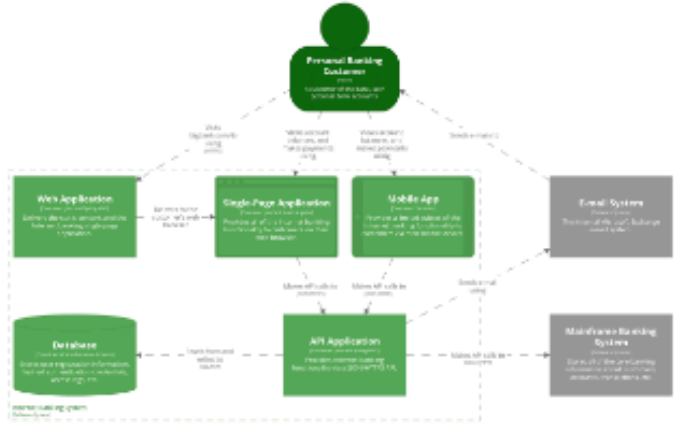


Maps of your code

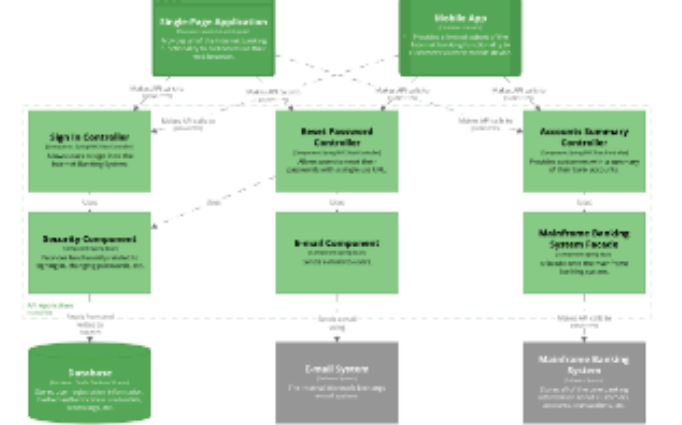
The C4 model was created as a way to help software development teams describe and communicate software architecture, both during up-front design sessions and when retrospectively documenting an existing codebase. It's a way to create "maps of your code", at various levels of detail, in the same way you would use something like Google Maps to zoom in and out of an area you are interested in.



[System Context] Internet Banking System



[Container] Internet Banking System



[Component] Internet Banking System - Web Application



Level 1: A [system context diagram](#) provides a starting point, showing how the software system in scope fits into the world around it.

Level 2: A [container diagram](#) zooms into the software system in scope, showing the applications and data stores inside it.

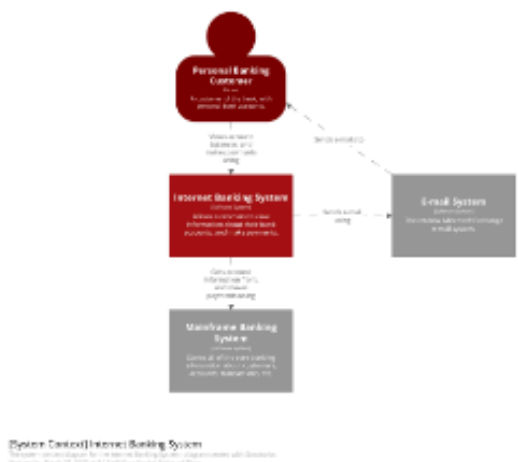
Level 3: A [component diagram](#) zooms into an individual container, showing the components inside it.

Level 4: A [code diagram](#) (e.g. UML class) can be used to zoom into an individual component, showing how that component is implemented at the code level.

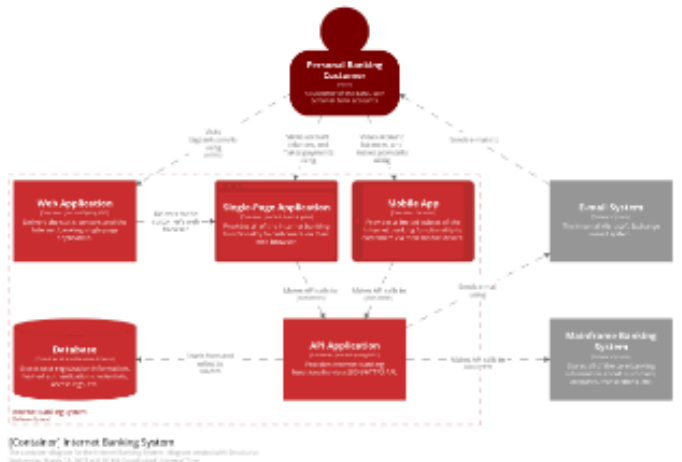


Maps of your code

The C4 model was created as a way to help software development teams describe and communicate software architecture, both during up-front design sessions and when retrospectively documenting an existing codebase. It's a way to create "maps of your code", at various levels of detail, in the same way you would use something like Google Maps to zoom in and out of an area you are interested in.



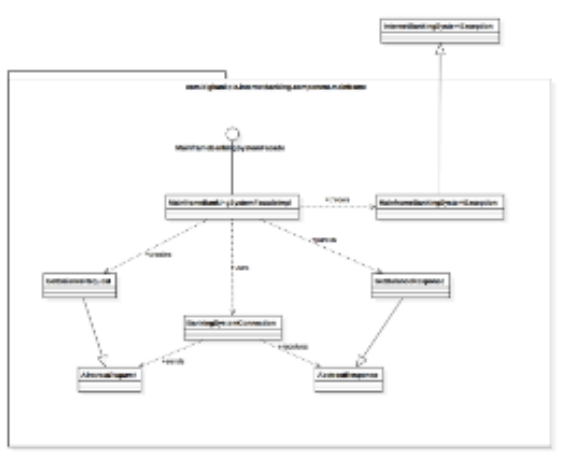
Level 1: A [system context diagram](#) provides a starting point, showing how the software system in scope fits into the world around it.



Level 2: A [container diagram](#) zooms into the software system in scope, showing the applications and data stores inside it.



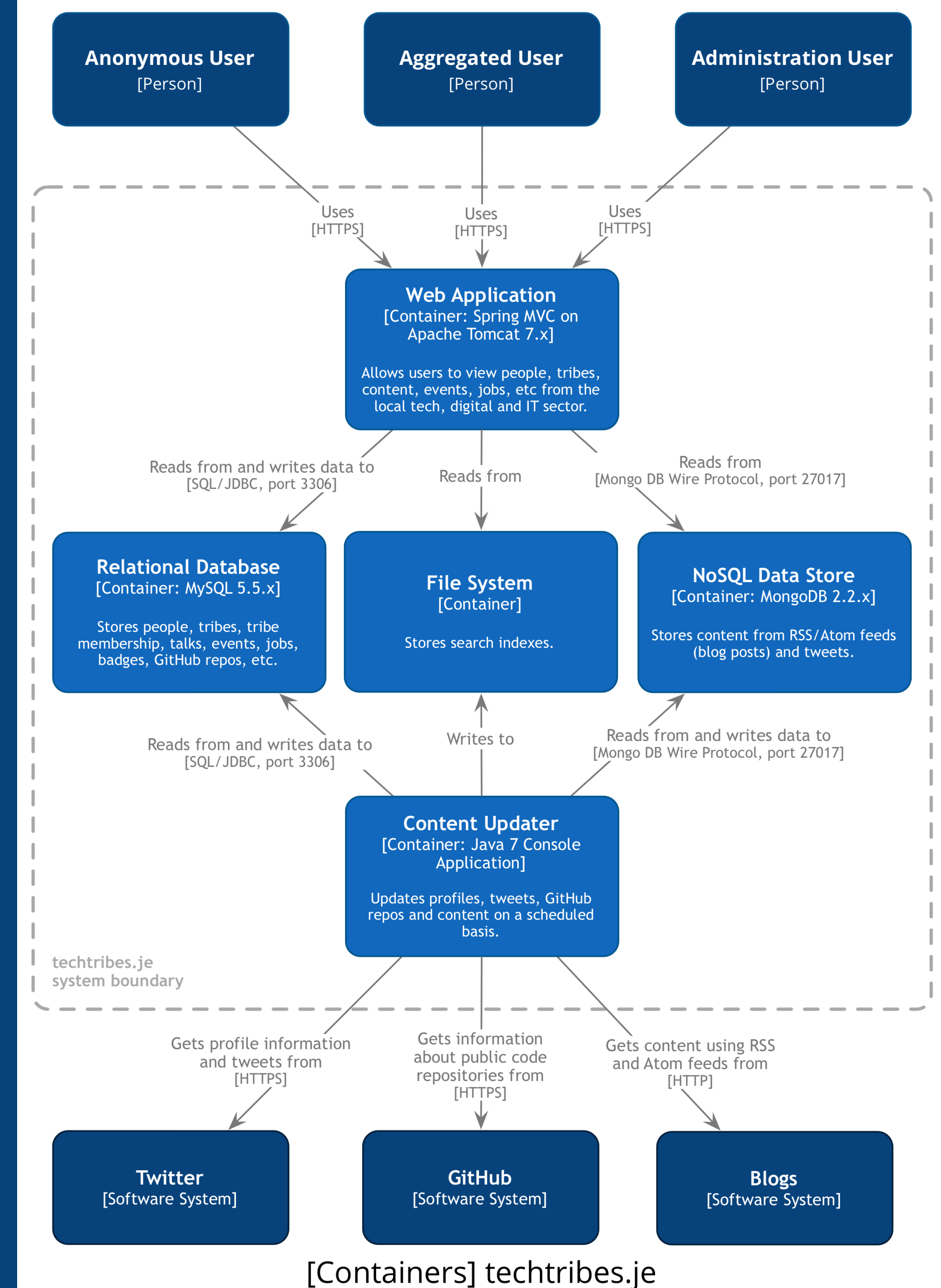
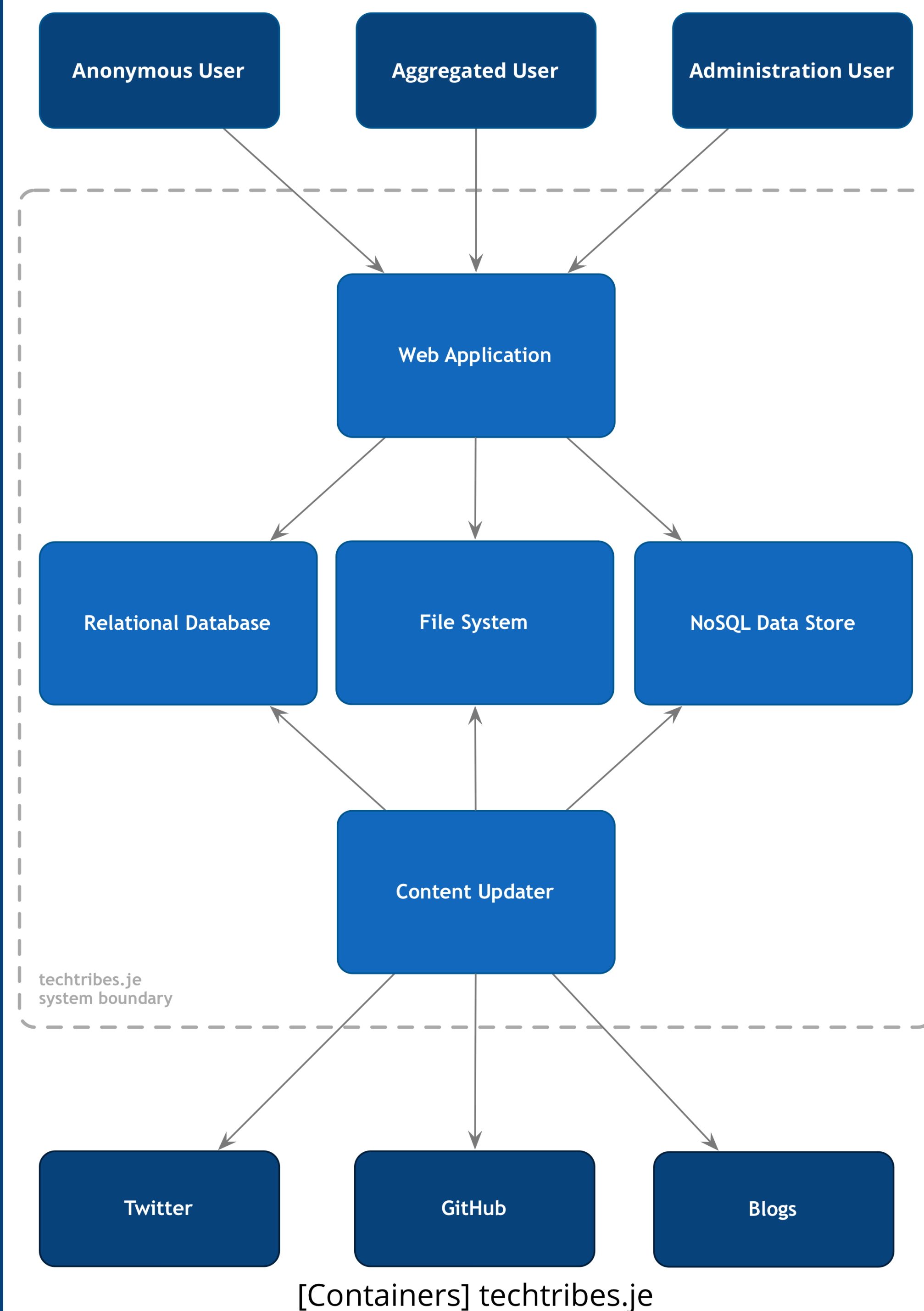
Level 3: A [component diagram](#) zooms into an individual container, showing the components inside it.



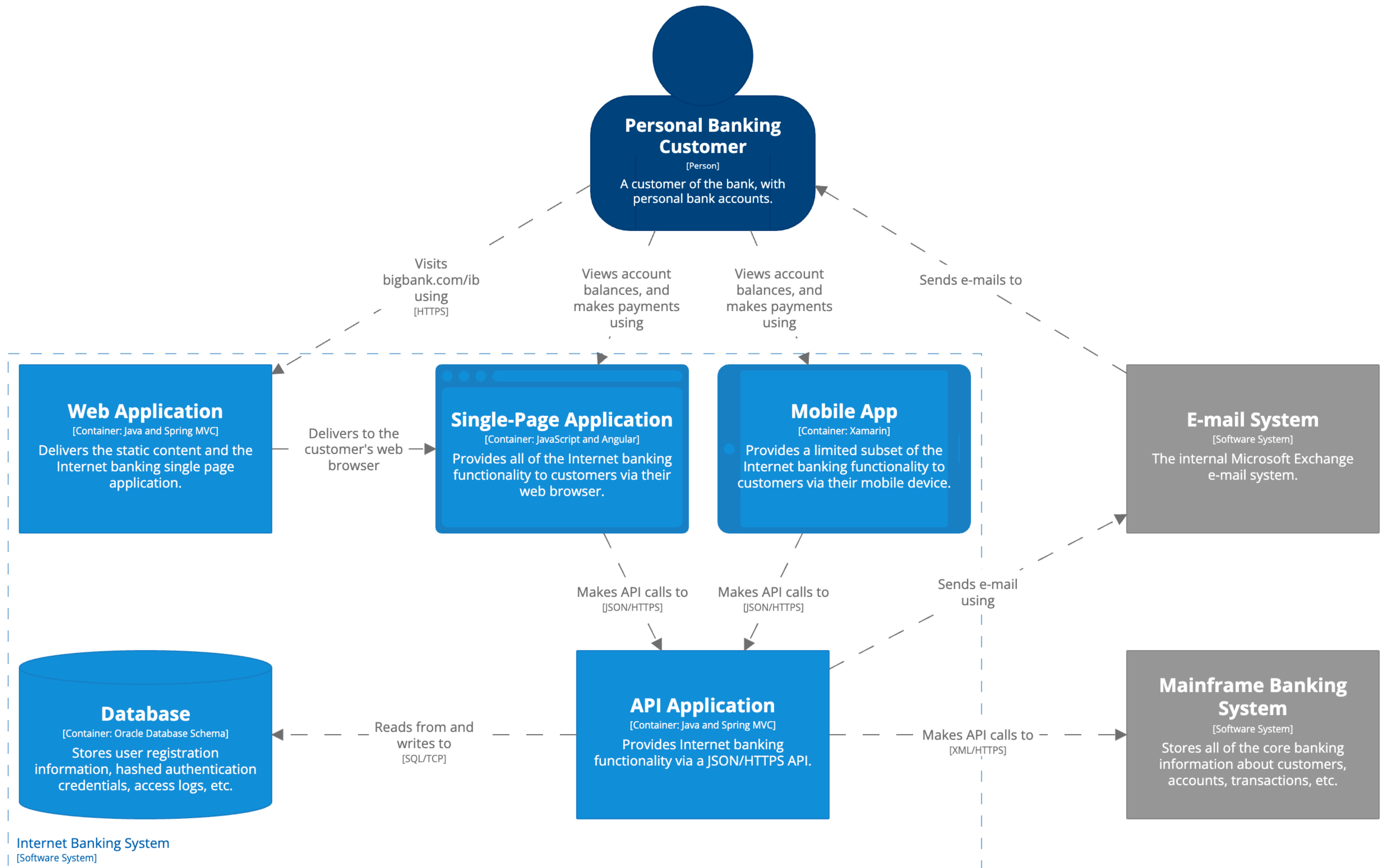
Level 4: A [code diagram](#) (e.g. UML class) can be used to zoom into an individual component, showing how that component is implemented at the code level.



[C4 is] unclear because you forced  
to place a lot of text in each box



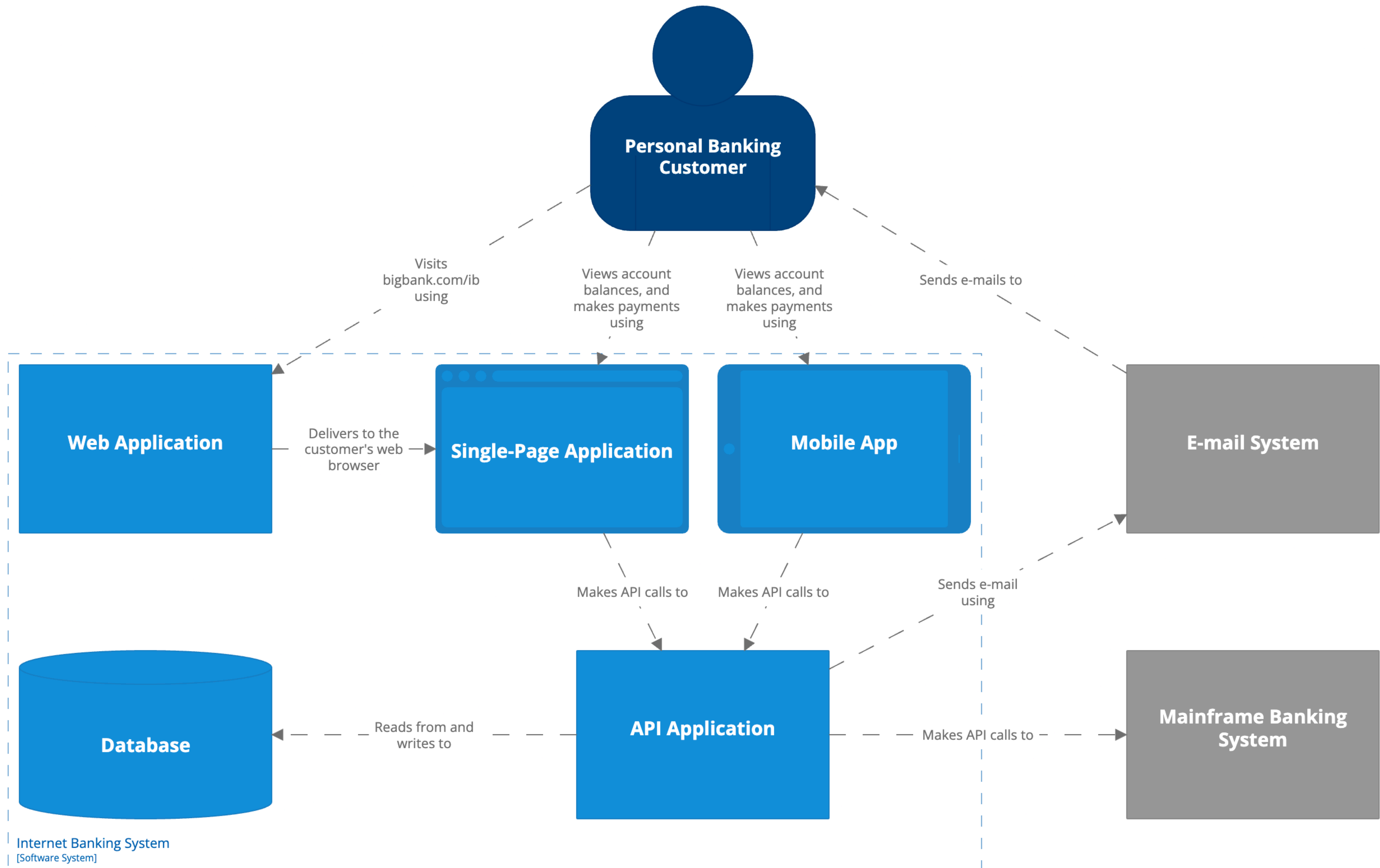




## [Container] Internet Banking System

The container diagram for the Internet Banking System - diagram created with Structurizr.

Wednesday, 22 March 2023 at 08:16 Greenwich Mean Time



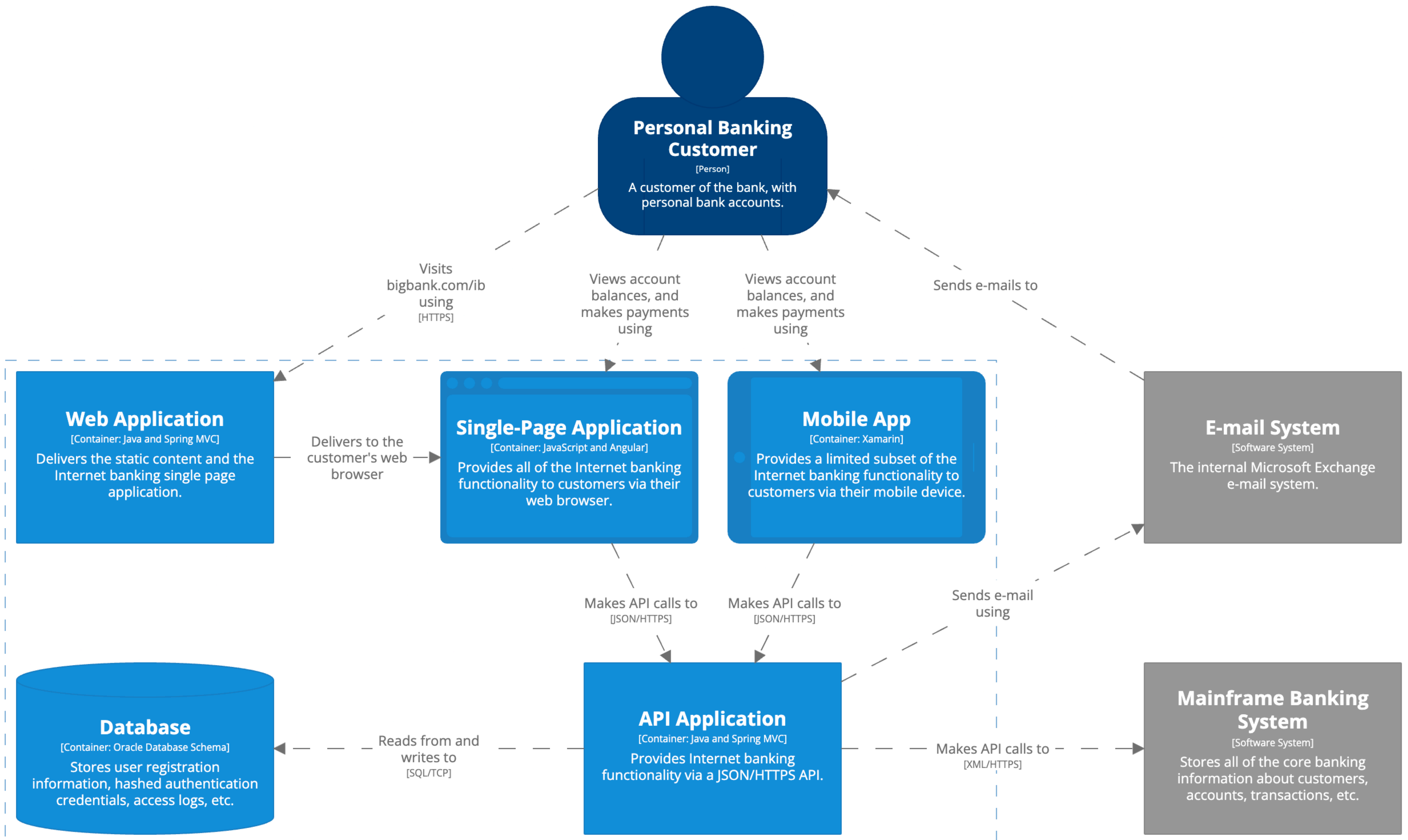
## [Container] Internet Banking System

The container diagram for the Internet Banking System - diagram created with Structurizr.

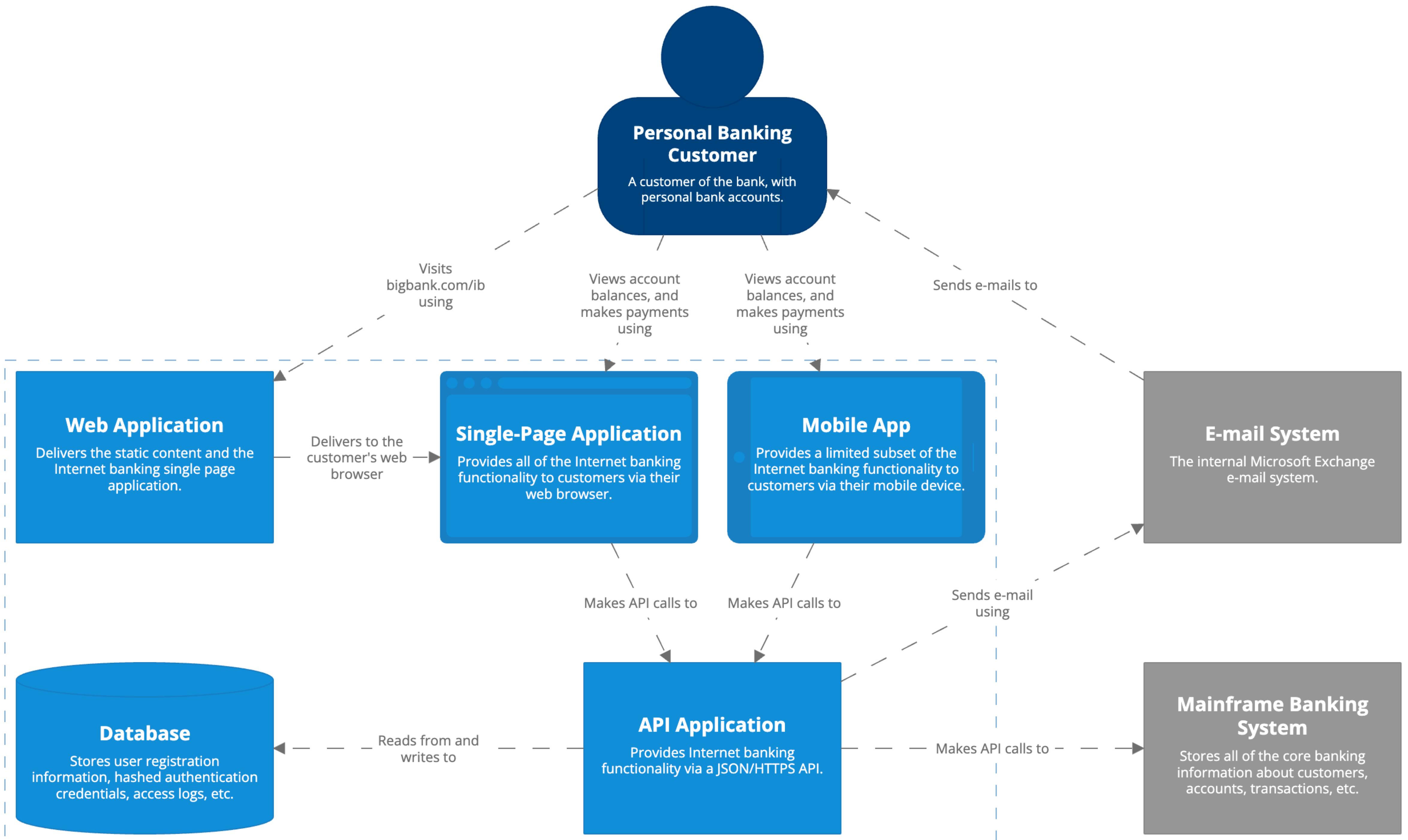
Wednesday, 22 March 2023 at 08:16 Greenwich Mean Time



We found the notation too cluttered,  
so removed the metadata







# Viewpoints



C4 isn't good at showing decisions

Architecture diagrams show  
the **outcome** of the  
decision making process



**Title** These documents have names that are short noun phrases. For example, "ADR 1: Deployment on Ruby on Rails 3.0.10" or "ADR 9: LDAP for Multitenant Integration"

**Context** This section describes the forces at play, including technological, political, social, and project local. These forces are probably in tension, and should be called out as such. The language in this section is value-neutral. It is simply describing facts.

**Decision** This section describes our response to these forces. It is stated in full sentences, with active voice. "We will ..."

**Status** A decision may be "proposed" if the project stakeholders haven't agreed with it yet, or "accepted" once it is agreed. If a later ADR changes or reverses a decision, it may be marked as "deprecated" or "superseded" with a reference to its replacement.

**Consequences** This section describes the resulting context, after applying the decision. All consequences should be listed here, not just the "positive" ones. A particular decision may have positive, negative, and neutral consequences, but all of them affect the team and project in the future.

# “Architecture Decision Record”

A short description of an  
architecturally significant decision

<https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions>

Michael Nygard



C4 model omits the  
deployment story



The C4 model was inspired by  
UML and the 4+1 model

The description of an architecture—the decisions made—can be organized around these four views, and then illustrated by a few selected *use cases*, or *scenarios* which become a fifth view. The architecture is in fact partially evolved from these scenarios as we will see later.

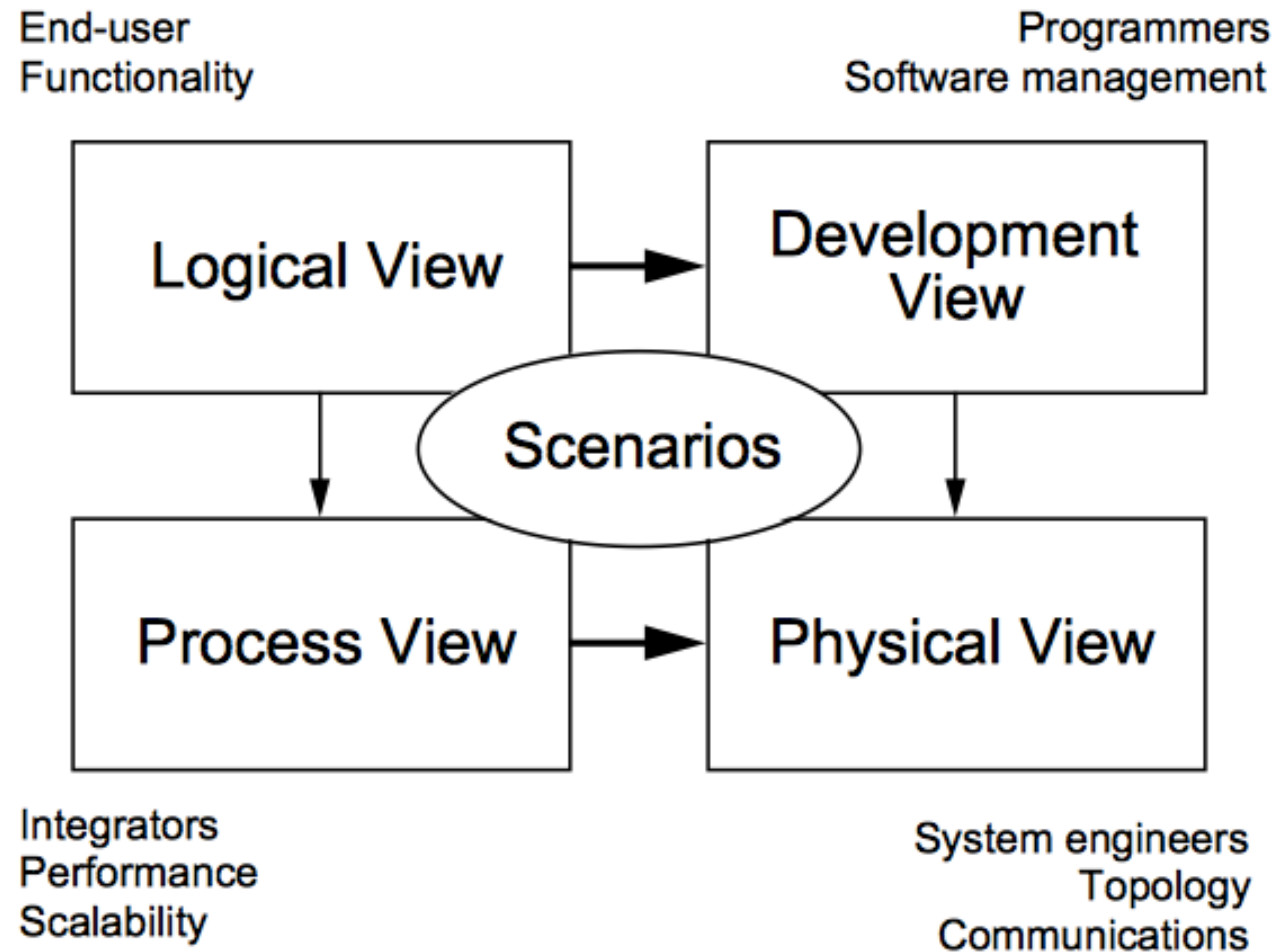
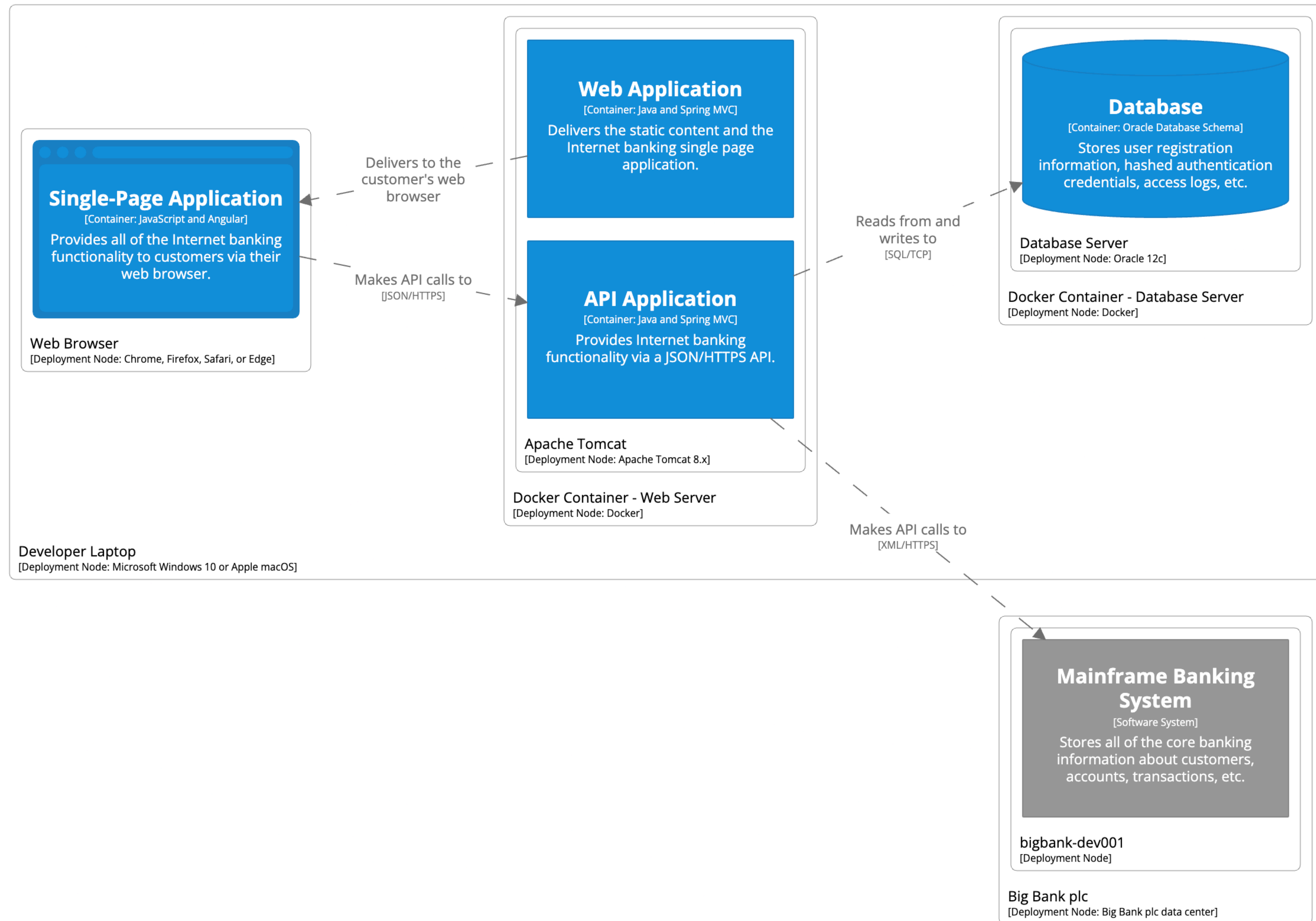


Figure 1 — The "4+1" view model

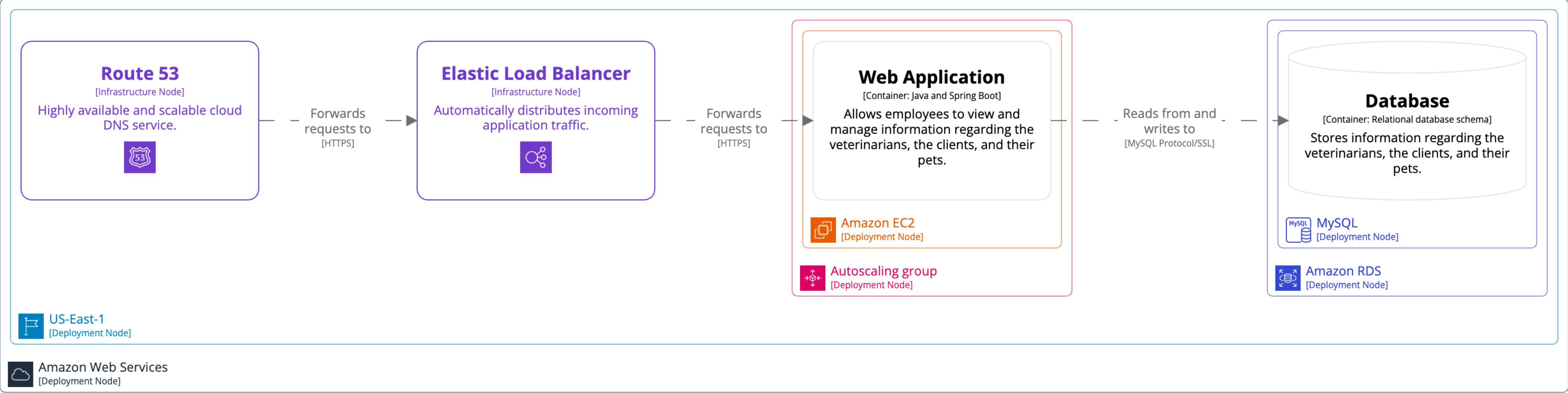




## [Deployment] Internet Banking System - Development

An example development deployment scenario for the Internet Banking System - diagram created with Structurizr.

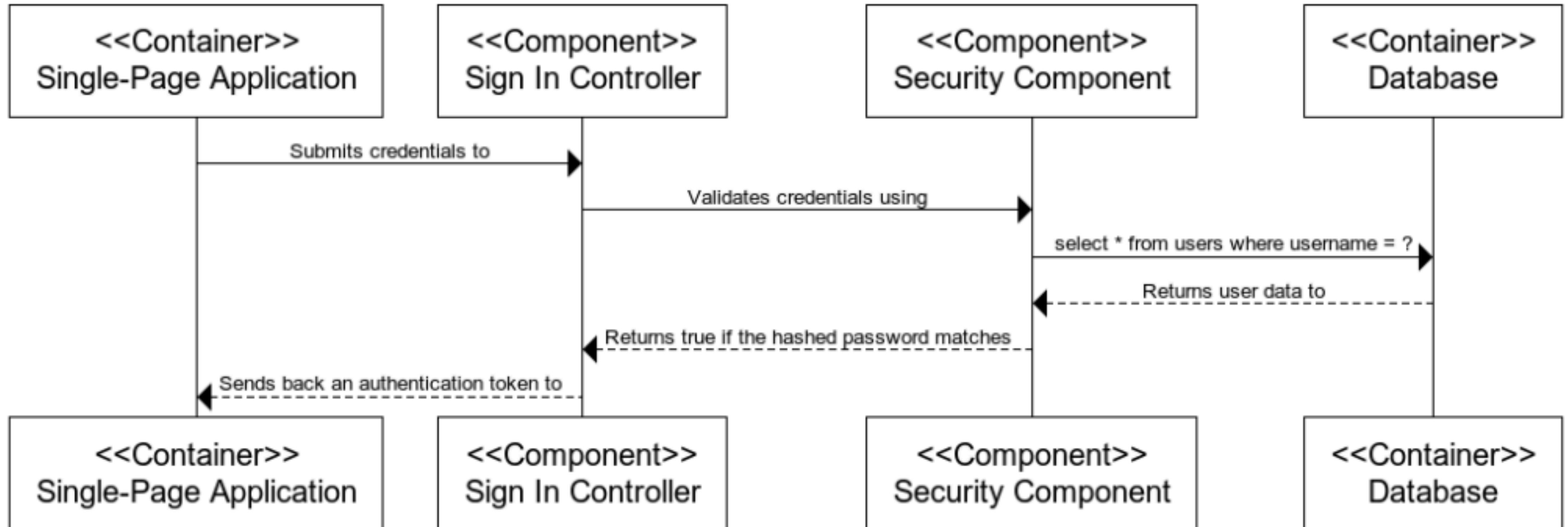
Wednesday, 22 March 2023 at 08:16 Greenwich Mean Time



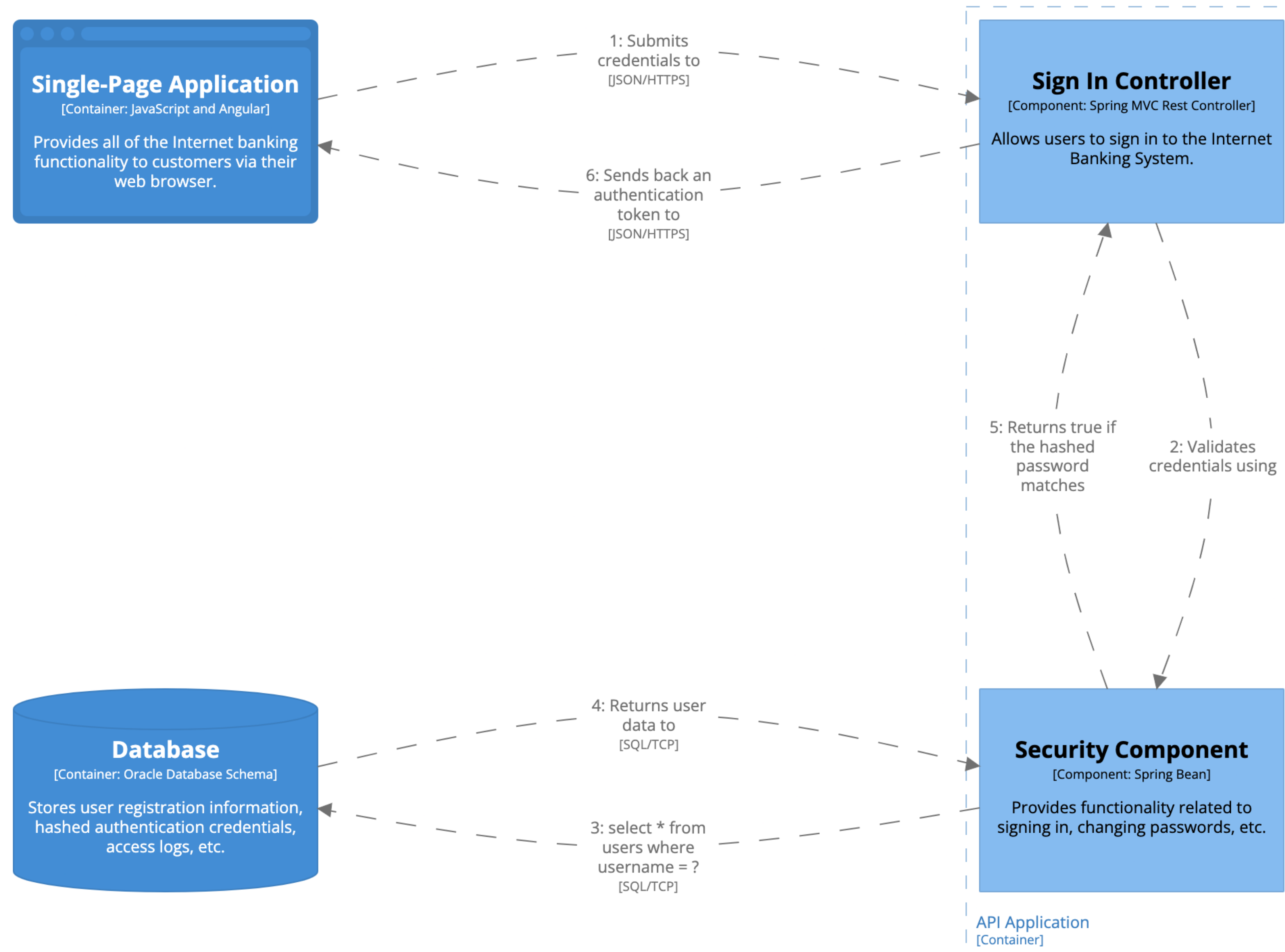


I personally prefer Event Modeling over C4. C4 has the major downside that it does not reflect the concept of time very well.

## API Application - Dynamic - SignIn







## [Dynamic] Internet Banking System - API Application

Summarises how the sign in feature works in the single-page application.

Monday, 27 February 2023 at 15:36 Greenwich Mean Time

“C4 is too limiting”



The problem is that systems today have many different kinds of things. Servers, databases, virtualized containers, APIs, pipelines, repositories, packages, libraries, and (many, many) cloud resources are all real, concrete things that provide real value. Forcing them into one of C4's four levels of abstraction doesn't really accomplish much.

"Concrete Diagramming Models, a Lightweight Alternative to C4"

[www.ilograph.com/blog/posts/concrete-diagramming-models](http://www.ilograph.com/blog/posts/concrete-diagramming-models)

A database is a database; debating whether it is also a Container or a Component just isn't worthwhile.

"Concrete Diagramming Models, a Lightweight Alternative to C4"

[www.ilograph.com/blog/posts/concrete-diagramming-models](http://www.ilograph.com/blog/posts/concrete-diagramming-models)

What is a “database”?





microservices shouldn't  
share a database

We need to be more precise  
with our terminology

("system, container, or component?" is helpful here)

The power of the C4 model is the small set of named hierarchical abstractions that help teams reason about their codebases in a structured and more precise way



Sep 26-27, 2024


**InfoQ Dev Summit Munich**  
Get clarity from senior software practitioners on today's critical dev priorities. Register Now.

Nov 18-22, 2024

**QCon San Francisco**  
Level up your software skills by uncovering the emerging trends you should focus on. Register now.

April 7-9, 2025

**QCon London**  
Discover emerging trends, insights, and real-world best practices in software development & tech leadership. Join now.



**The Software Architects' Newsletter**  
Your monthly guide to all the topics, technologies and techniques that every professional needs to know about. Subscribe for free.

ARCHITECTURE & DESIGN


InfoQ Dev Summit Munich (Sep 26-27): Save up to 60% with our Summer Sale until August 13.

# Navigating Software Architecture at Scale: Insights from Decathlon's Architecture Process

LIKE

JUL 24, 2024 • 6 MIN READ

by



Eran Stiller

FOLLOW

Principal Software Architect

## Write for InfoQ

### Feed your curiosity.

Help 550k+ global senior developers each month stay ahead.

[Get in touch](#)

[Raphaël Tahar](#), staff engineer at Decathlon, recently published his [insights from co-leading an architecture process at scale](#). In a 4-part blog post series, Tahar depicts how, by combining methodologies like architecture committees, the C4 model, and System Thinking and emphasizing the importance of ADRs and centralized documentation, Decathlon ensures its teams are well-equipped to make informed, strategic decisions.

He is part of a group supporting over 120 engineers across 23 feature teams comprising one domain out of 1500+ engineers globally at Decathlon. Supporting this scale of developers is no small feat and involves providing support with designing new systems, optimizing existing ones, and ensuring alignment with global guidelines. To tackle this, Decathlon established an architecture committee, which plays a crucial role in guiding teams through the intricate decision-making process.

Tahar explains the need for an architecture committee via the [Garbage Can Model](#). Developed in the 1970s, this model describes organizational decision-making as a chaotic process where problems, solutions, and decision-makers exist in separate streams. These streams interact in unpredictable ways, much like items in a garbage can, leading to decision opportunities that emerge from this interplay.


However, according to Tahar, this "garbage can flow" is missing three items: decision alternatives, consequences, and consequences vs objectives.

“Organizations must ensure that those 3 points are covered; if they aren't, the chances for late projects, late releases, and backfires (like unreliable services



## RELATED CONTENT

- Getting Technical Decision Buy-In Using the Analytic Hierarchy Process


MAR 13, 2024


- Accelerating Technical Decision-Making by Empowering ICs with Engineering Strategy


MAY 22, 2024


-  The Decision Buy-In Algorithm

MAR 19, 2024


- How to Deal with Complexity in Product Development by Using Solution-Focused Coaching

JAN 29, 2024


- AWS Discontinues Amazon Quantum Ledger Database (QLDB)

JUL 26, 2024
- How Team Health Checks Help Software Teams to Deliver

JUL 25, 2024
- AWS Launches Open-Source Agent for AWS Secrets Manager

JUL 25, 2024

**InfoQ: What have been the most significant impacts of implementing the C4 model at Decathlon, and how has it helped manage complexity and improve system understanding among engineers?**

“

**Tahar:** The C4 model is inherently declarative. It requires teams to address and synchronize their mental models of the code, components, containers, and even the context of their applications. This led to valuable discussions where knowledge was shared, and beliefs were reconsidered.

It also assisted leadership teams in understanding the interdependencies among teams and external systems. In other words, it helped identify risks and visualize the landscape for organizational optimizations.

Lastly, as the saying goes, "A picture is worth a thousand words." The C4 model makes cross-team and cross-domain discussions much more seamless by standardizing the capture and sharing of contexts.

Inspired by C4 Model and Structurizr DSL, but with some flexibility. You define your own notation, custom element types and **any number of nested levels in the architecture model**. Perfectly tailored to your needs.



```
model {  
  service service {  
    component backend1 {  
      component api  
    }  
    component backend2 {  
      component api  
      component graphql  
    }  
  }  
}
```

What is a  
“component”?

```
model{
  c1 = component "c1"{
    c2 = component "c2"{
      c4 = component "c4"
      c5 = component "c5"
      c3 -> c4
      c4 -> c5
    }
  }

  c3 = component "c3"
}

views{
  view of c1{
    include *
  }

  view of c2{
    include *
  }
}
```

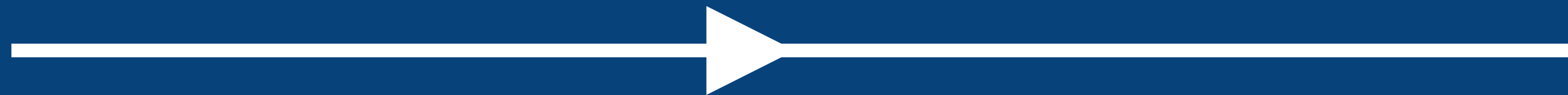


## Ad hoc abstractions

Most “boxes and arrows” diagrams

## Defined abstractions

C4 model



Forcing them into one of C4's four levels of abstraction doesn't really accomplish much.

Inspired by C4 Model and Structurizr DSL, but with some flexibility. You define your own notation, custom element types and any number of nested levels in the architecture model.



# Ad hoc abstractions

Most “boxes and arrows” diagrams

# Defined abstractions

C4 model

# Flexible abstractions

Illograph, LikeC4, etc

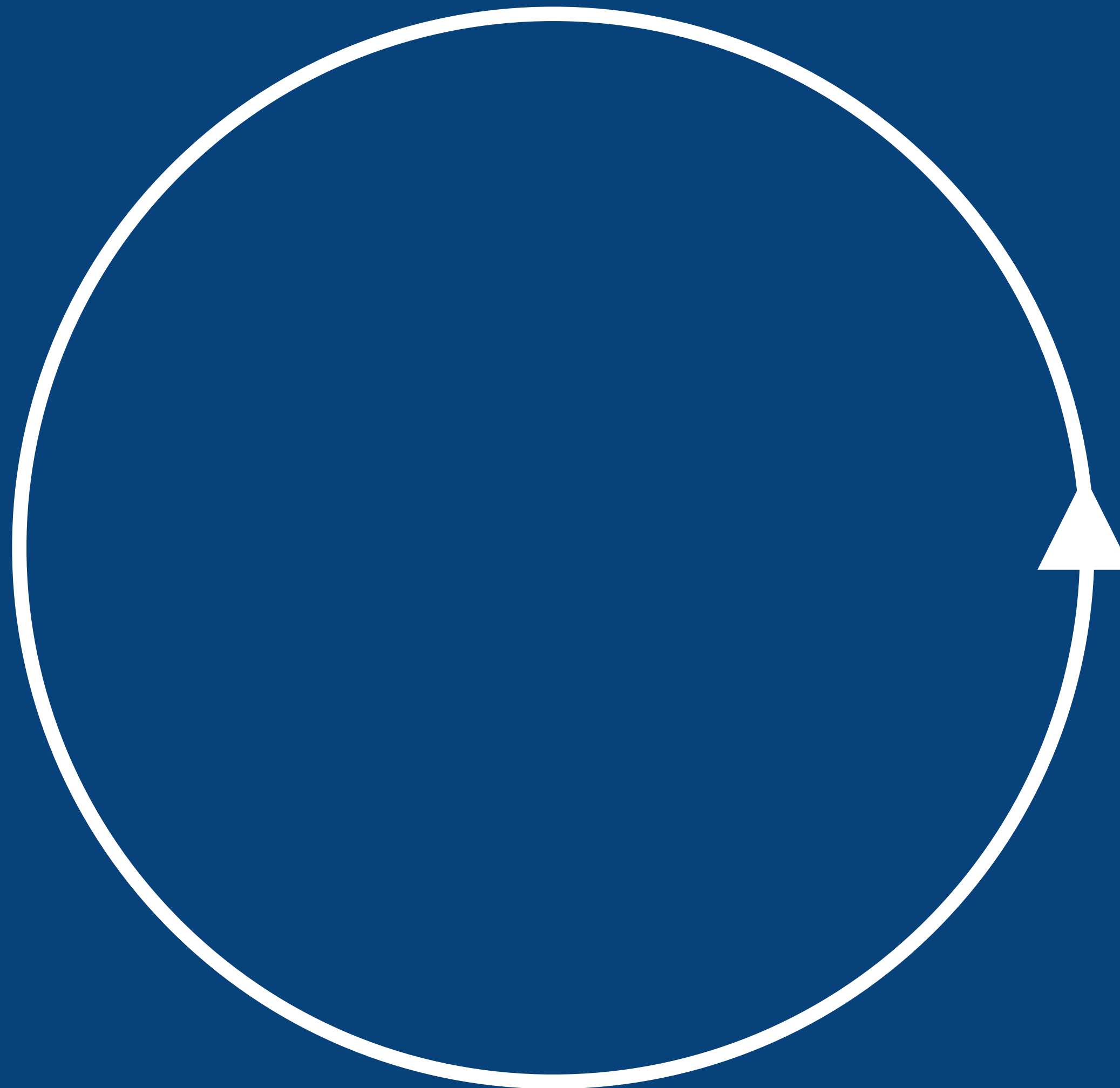


# Ad hoc abstractions

Most “boxes and arrows” diagrams

# Flexible abstractions

Ilograph, LikeC4, etc



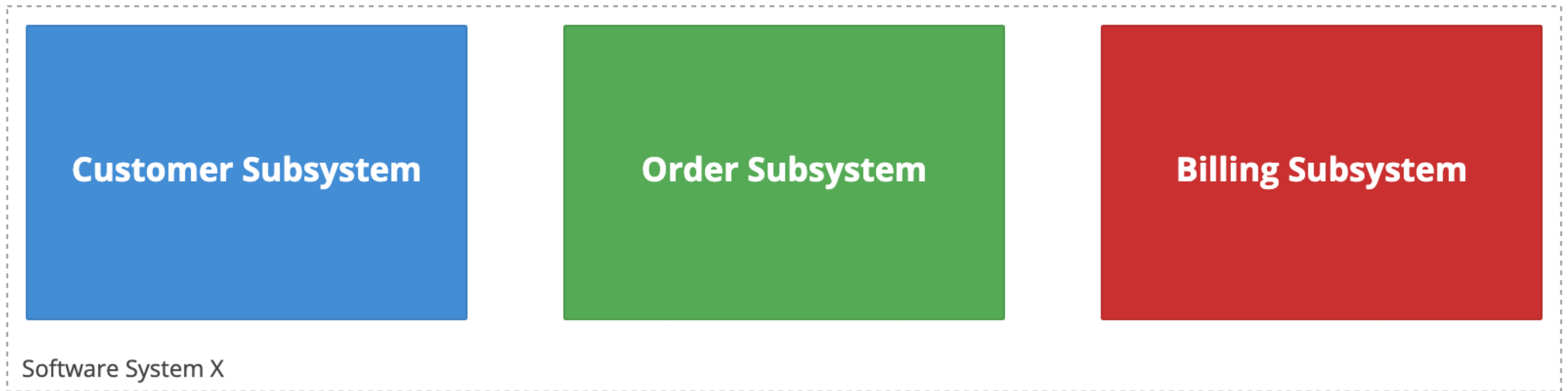
# Defined abstractions

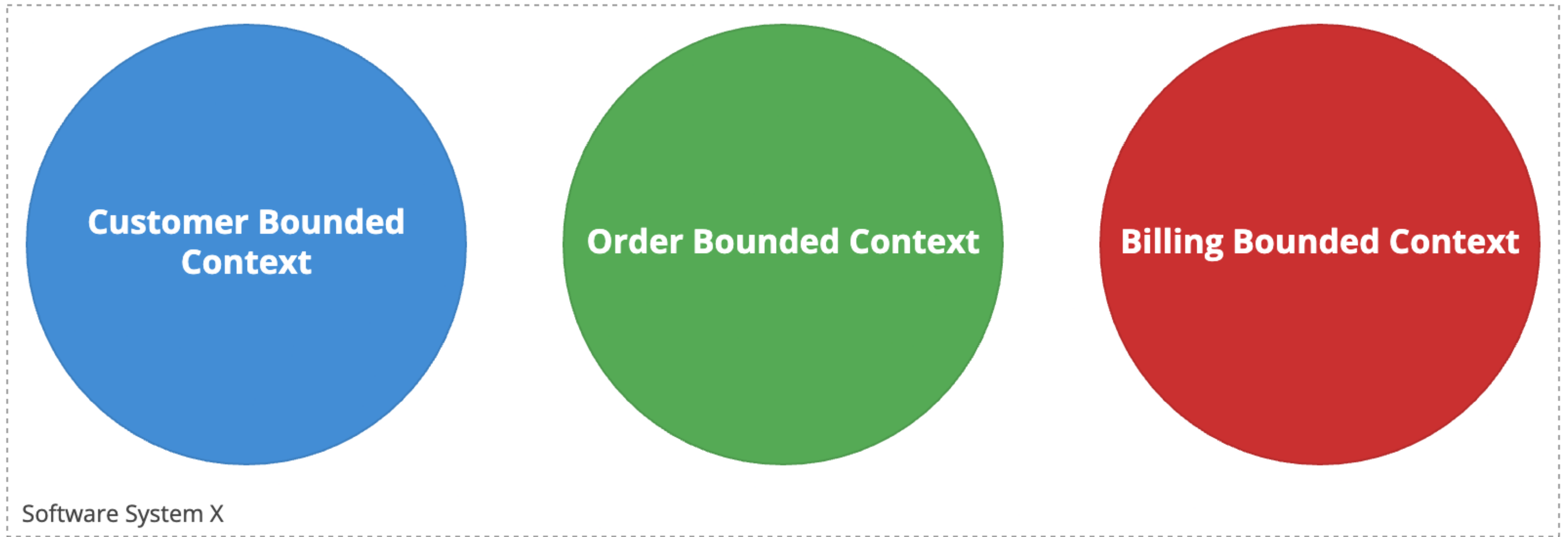
C4 model

# Abstraction vs organisation

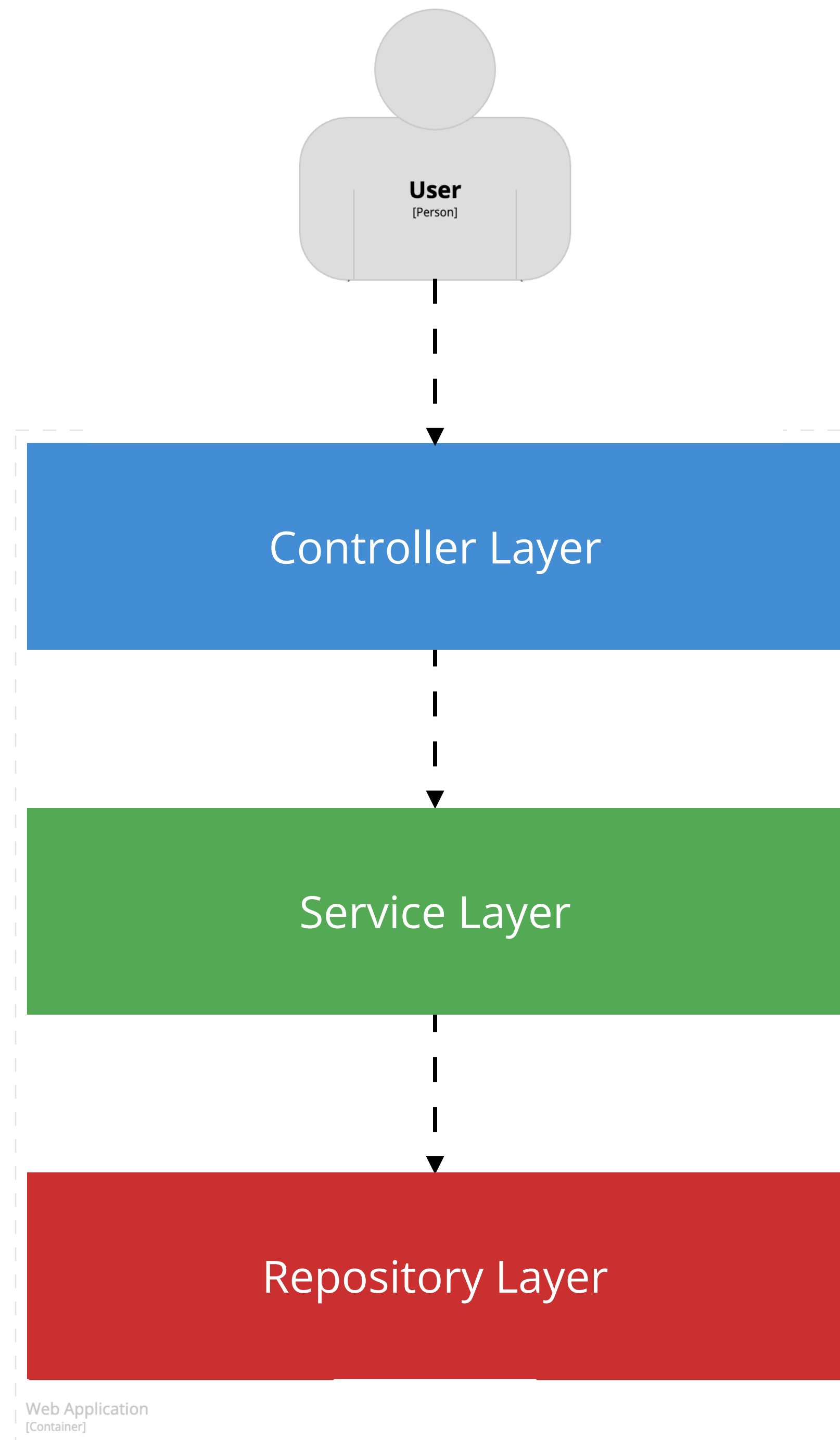


What are your thoughts on modelling  
additional abstractions?

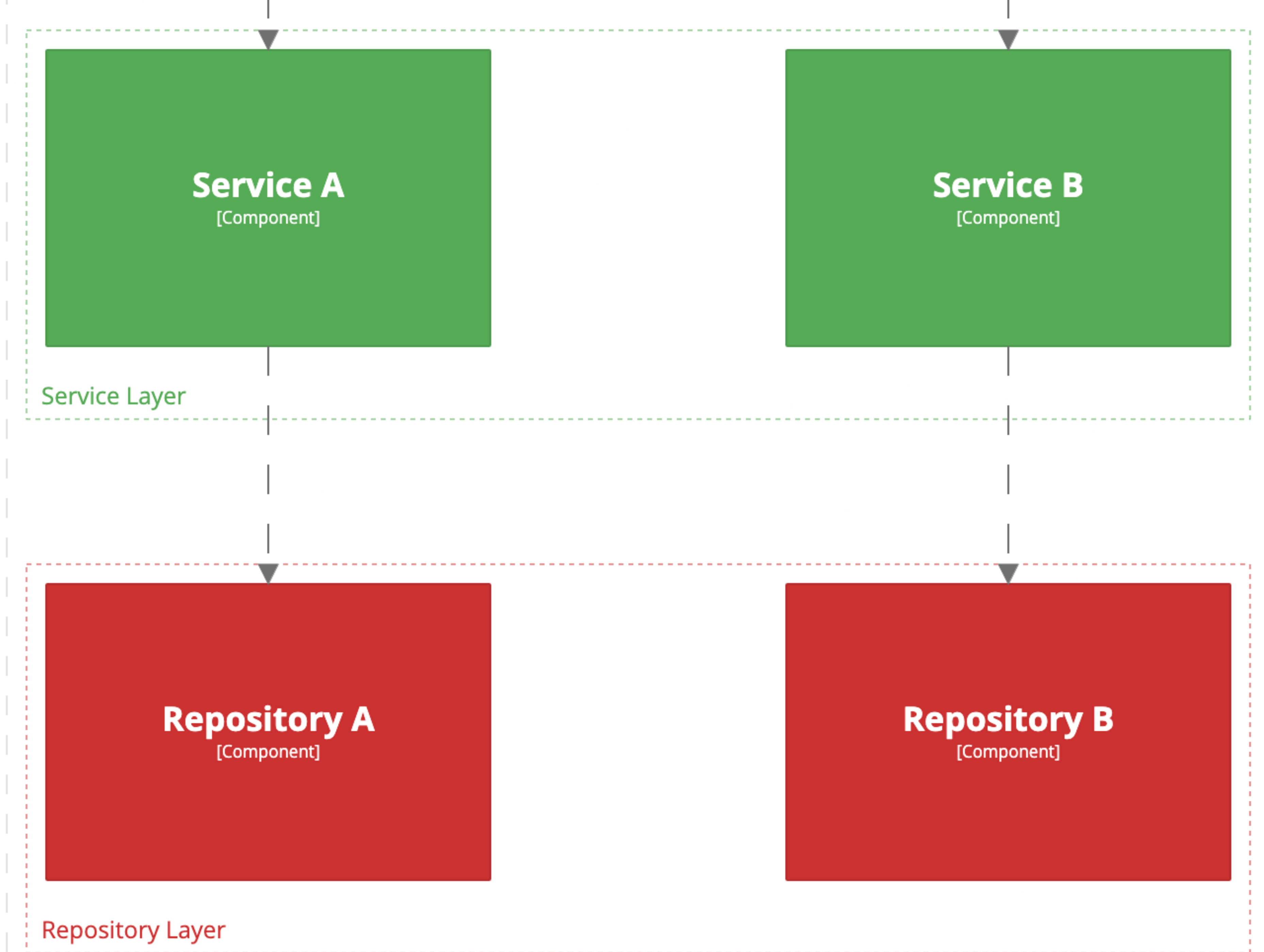




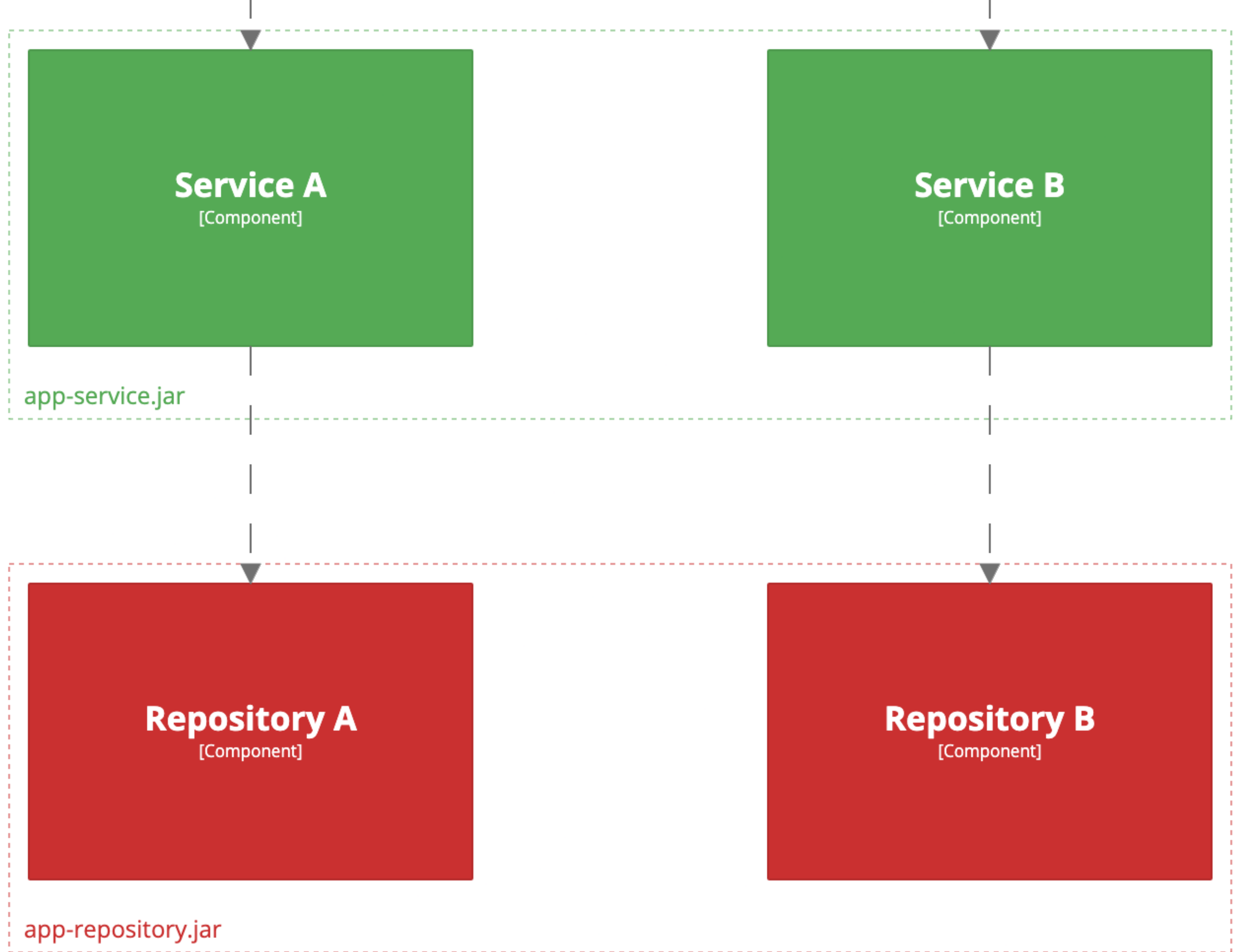




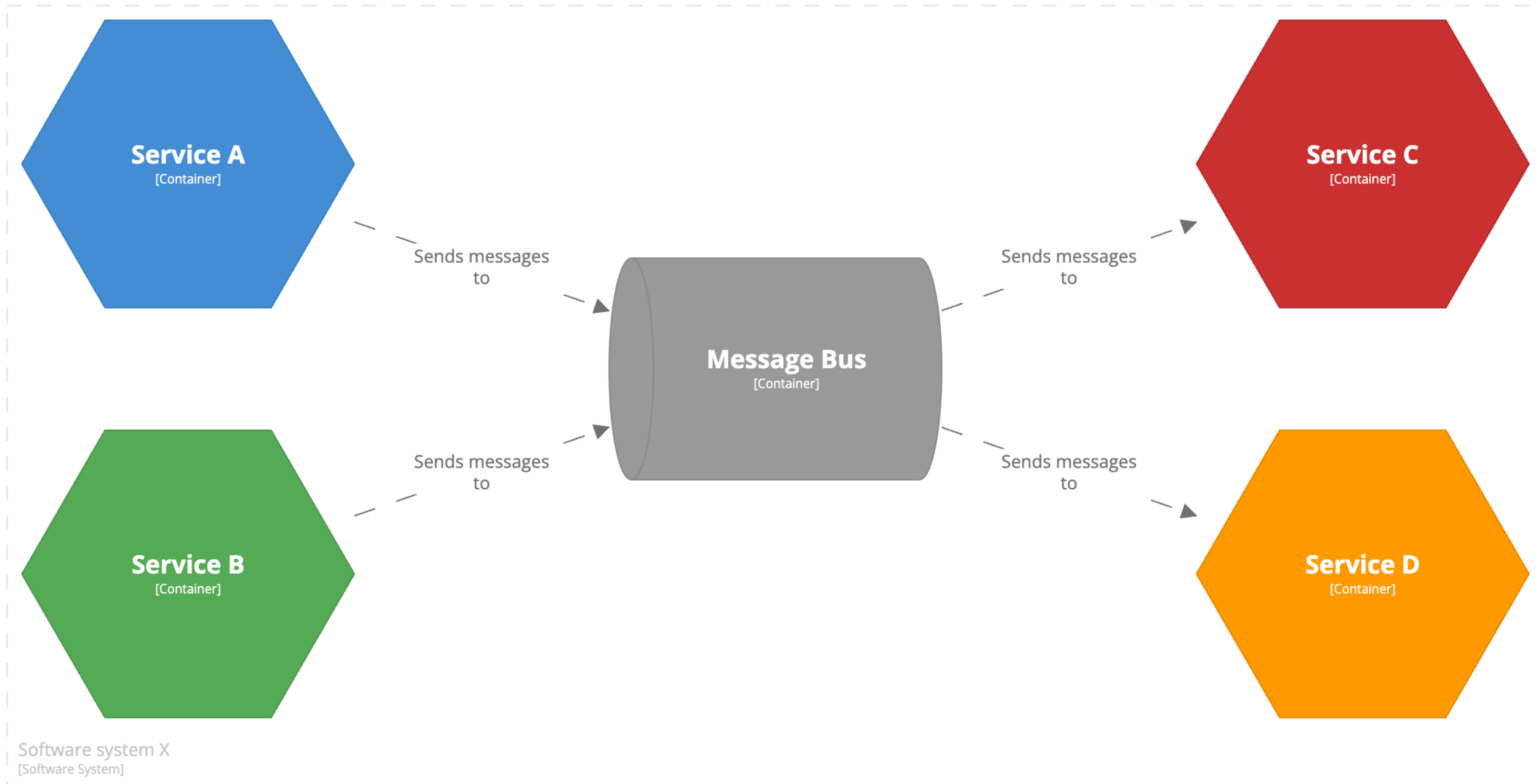
Some of these concepts  
might be better thought of as  
**organisational constructs**  
rather than abstractions



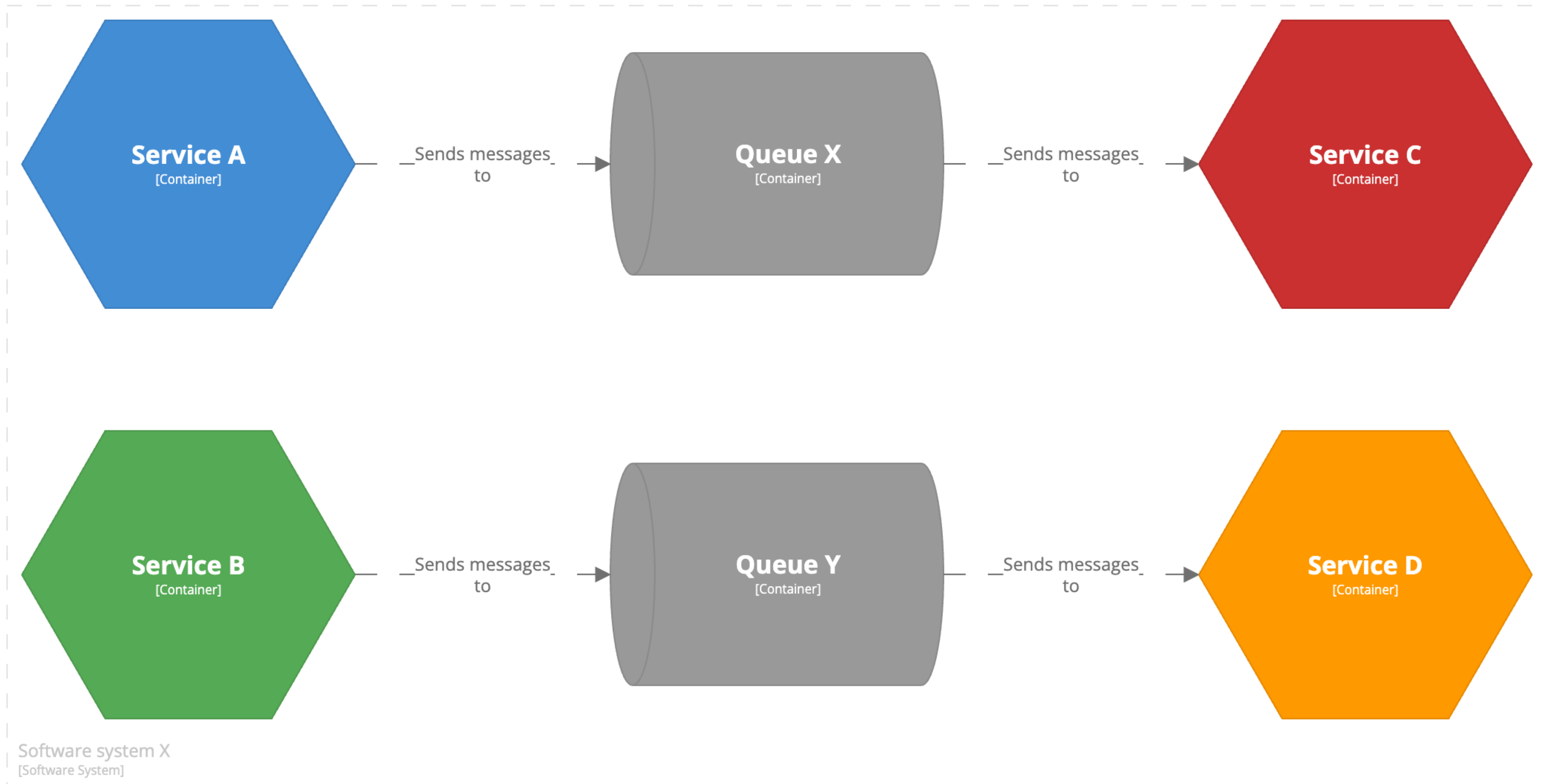


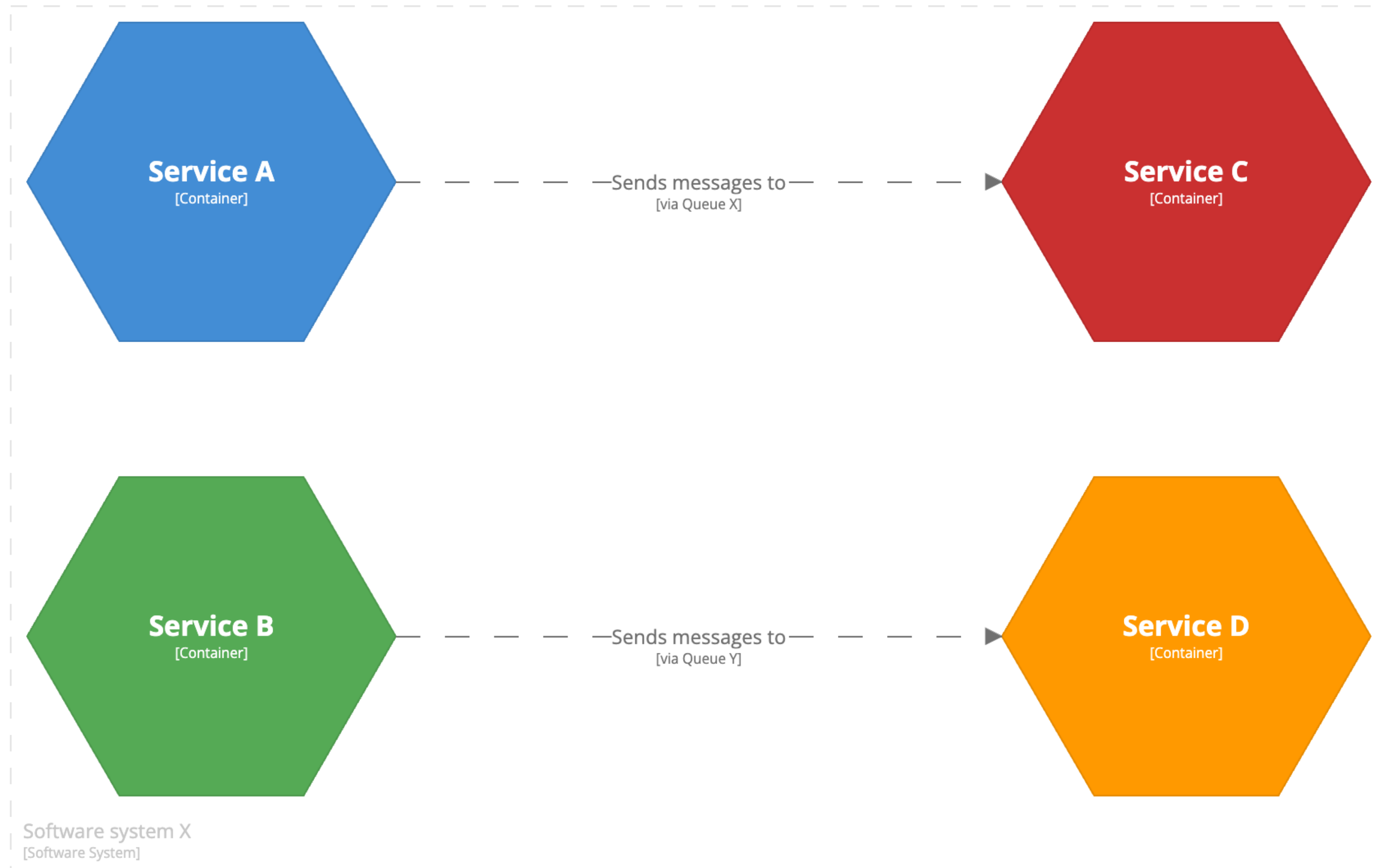


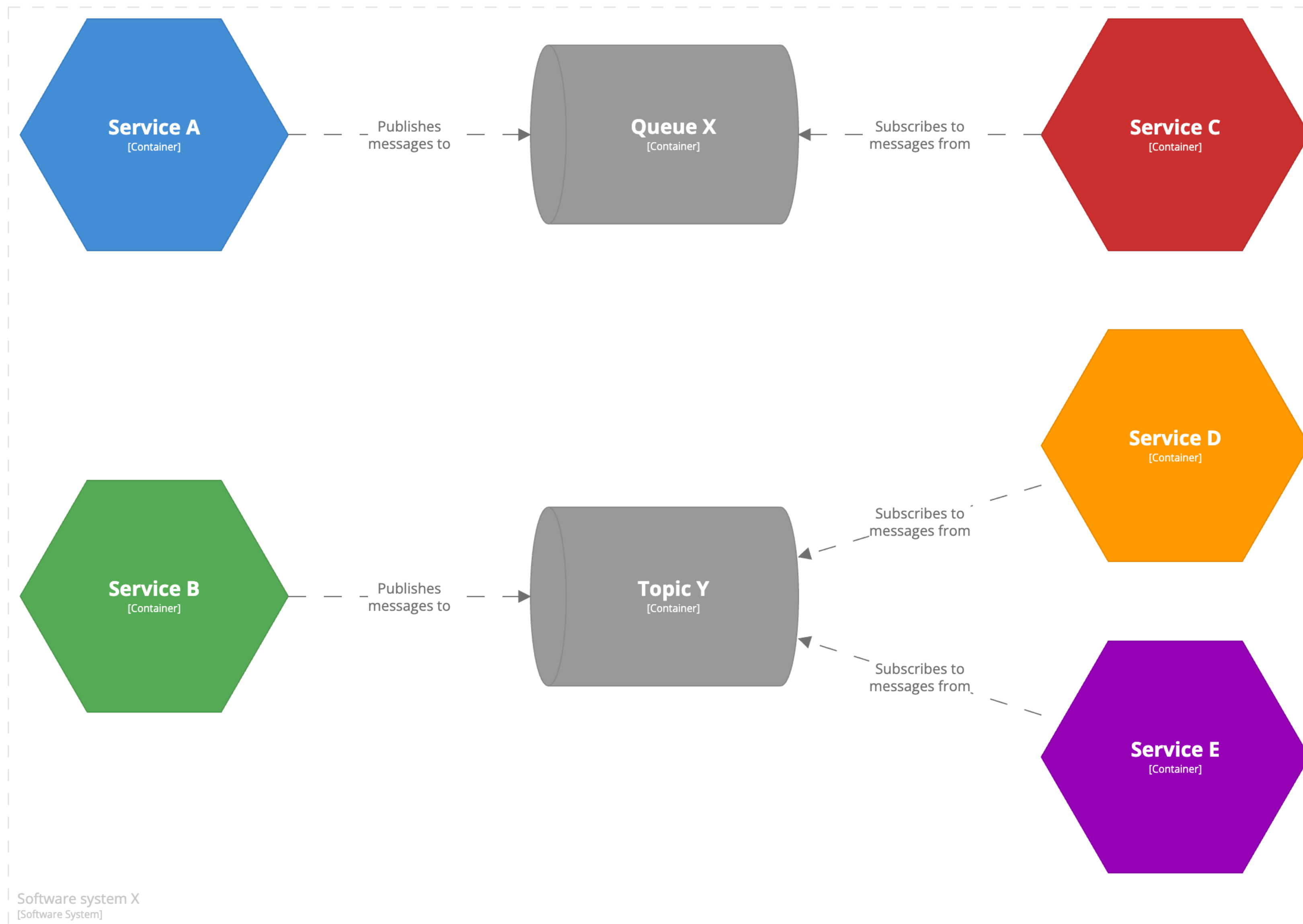
# Message-driven architectures





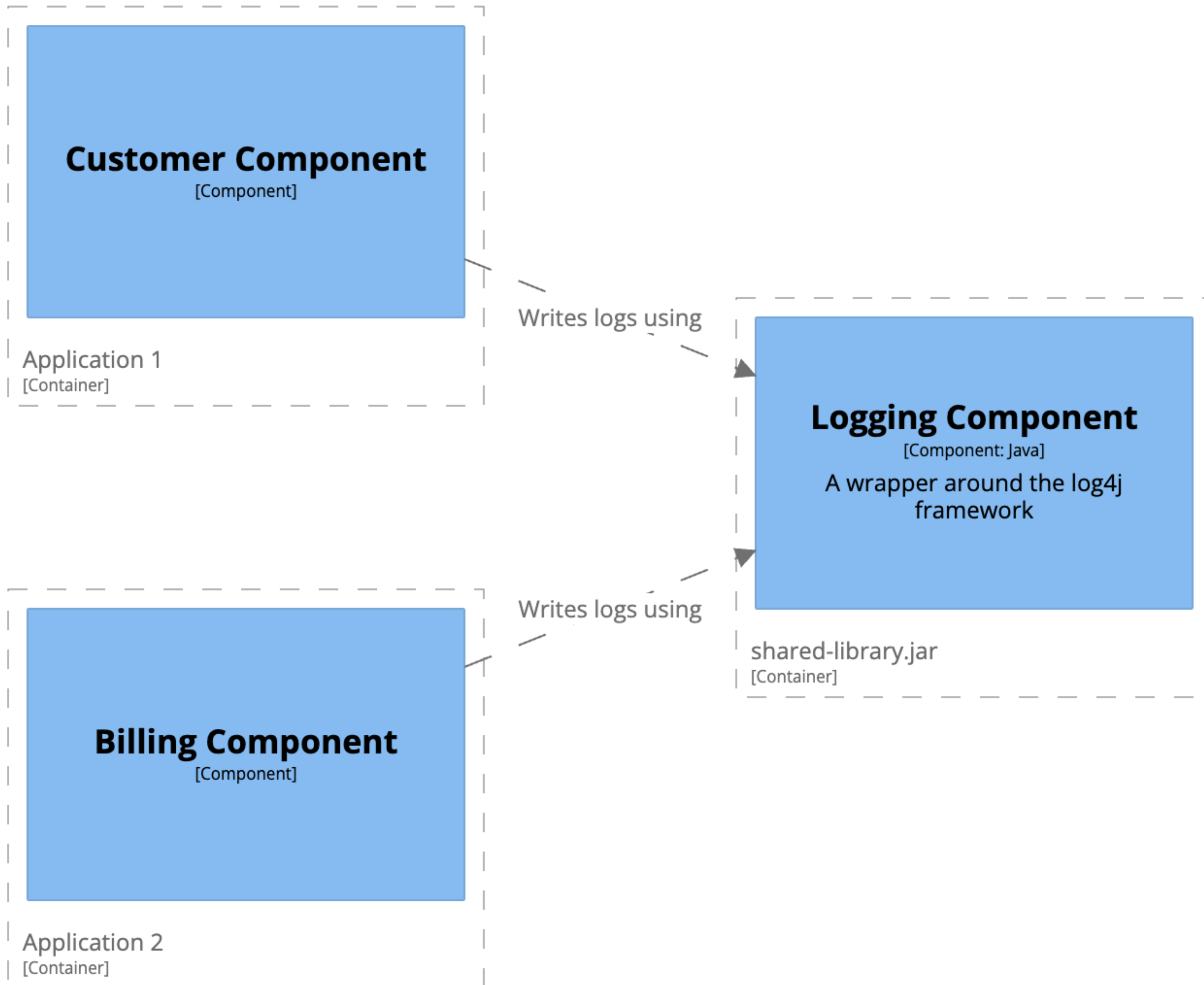


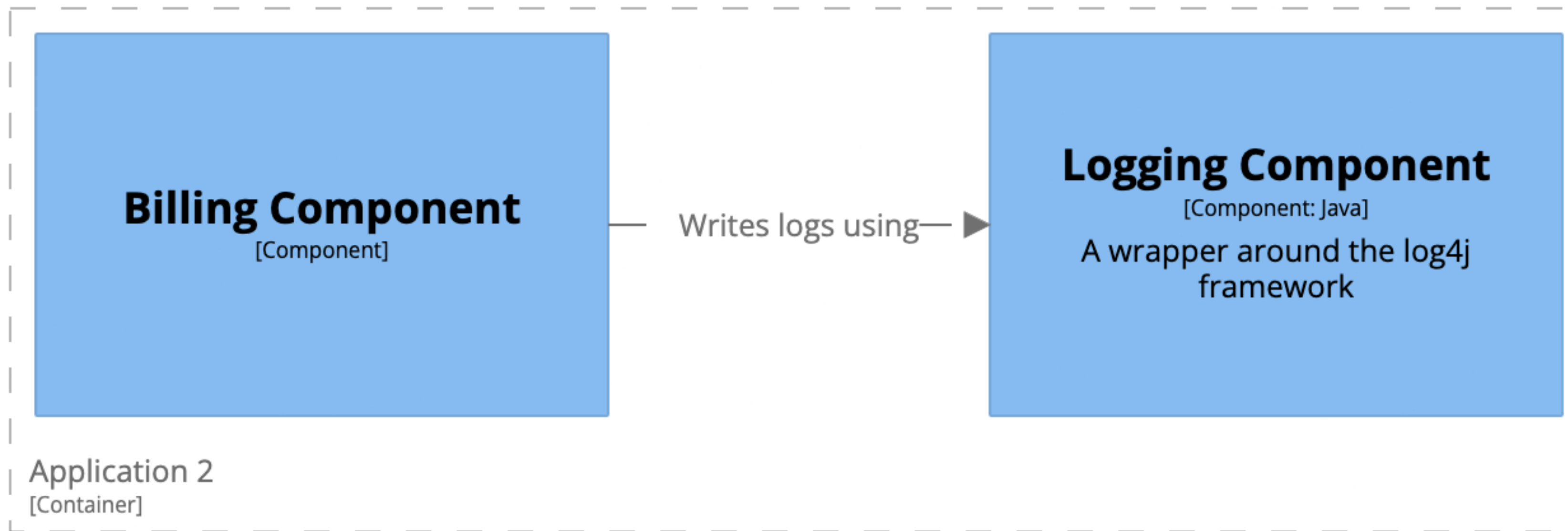
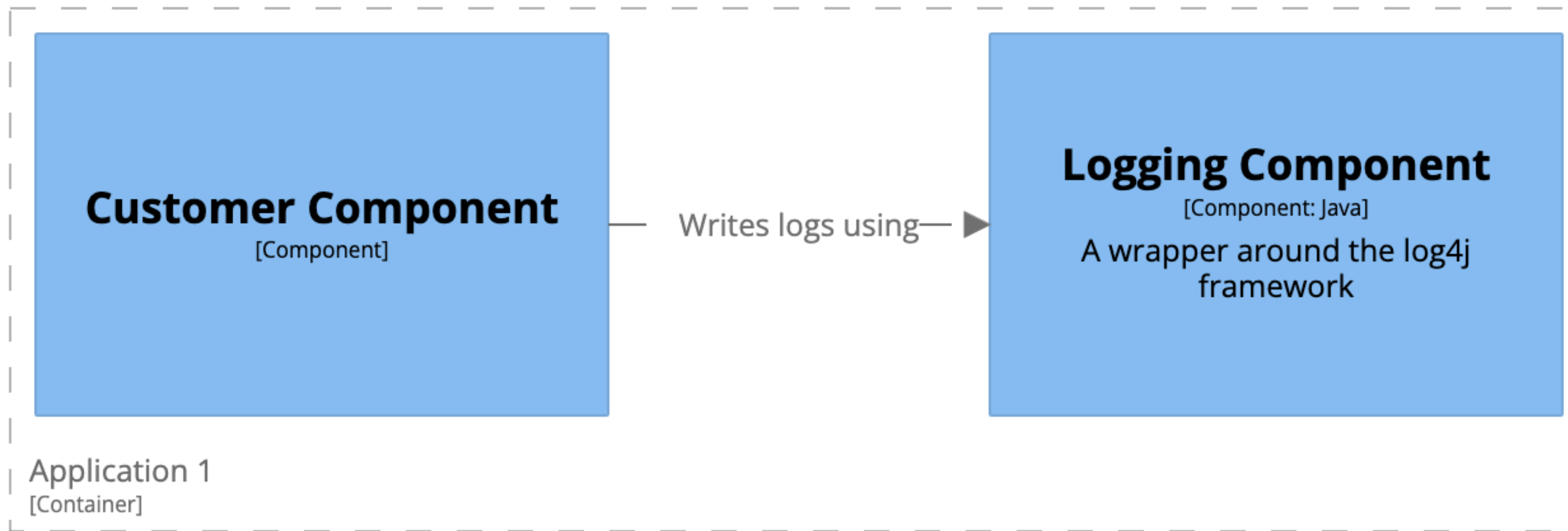




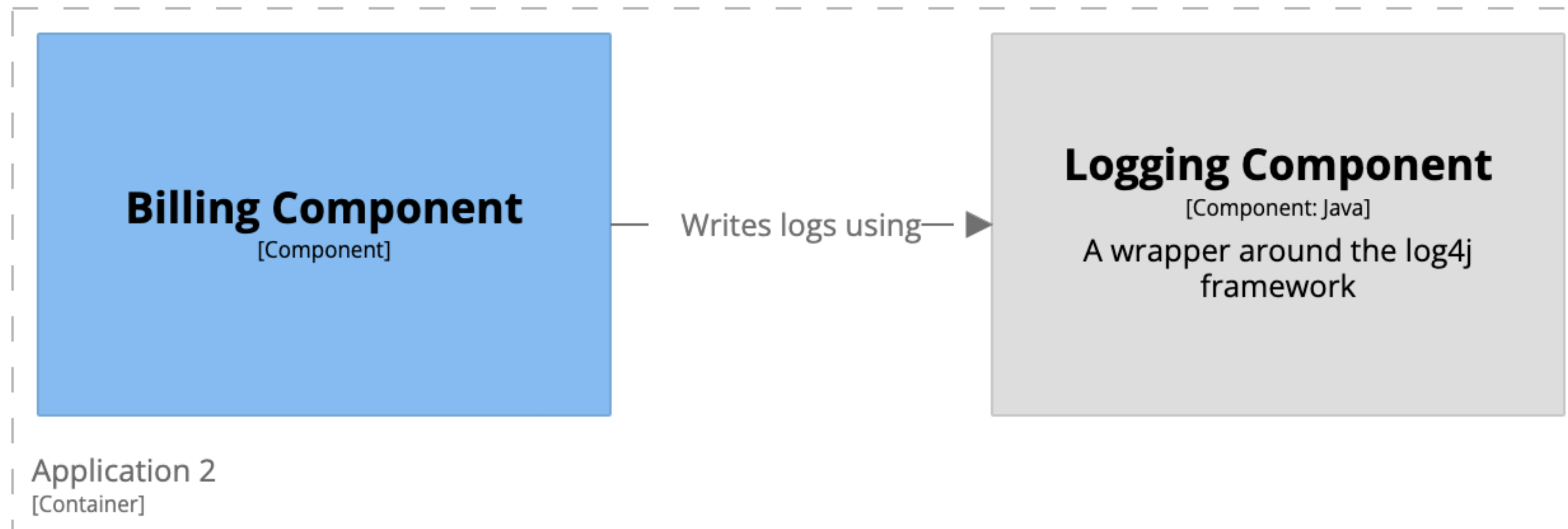
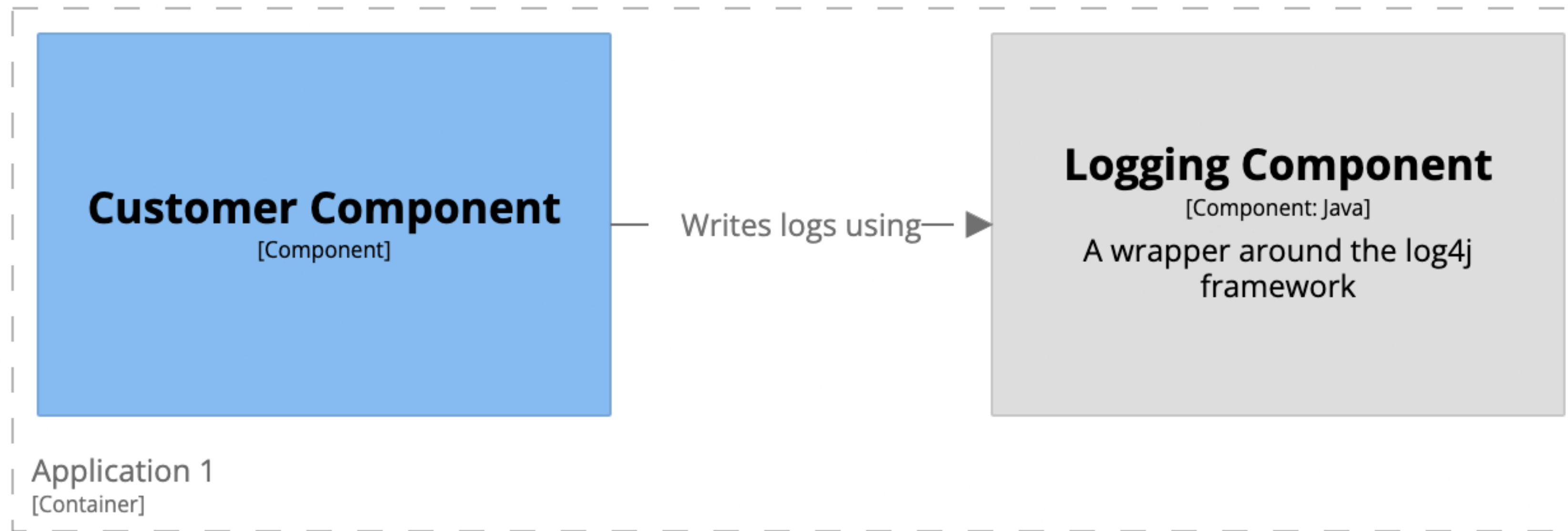


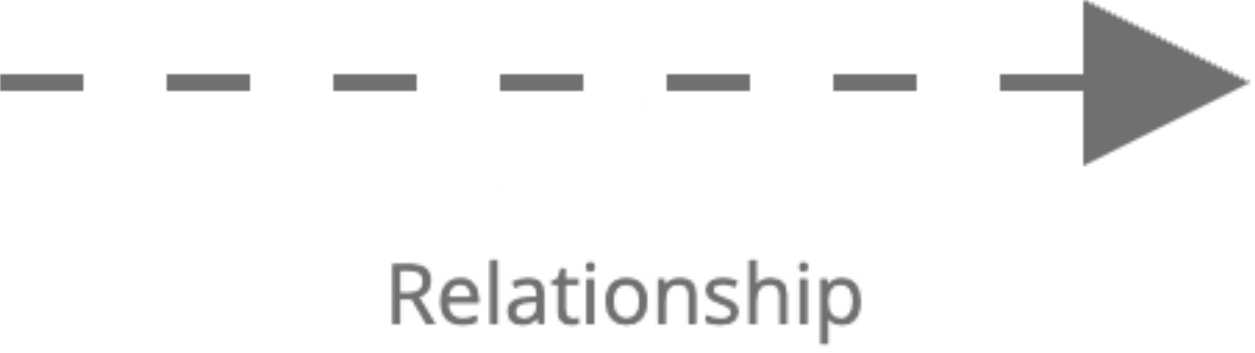
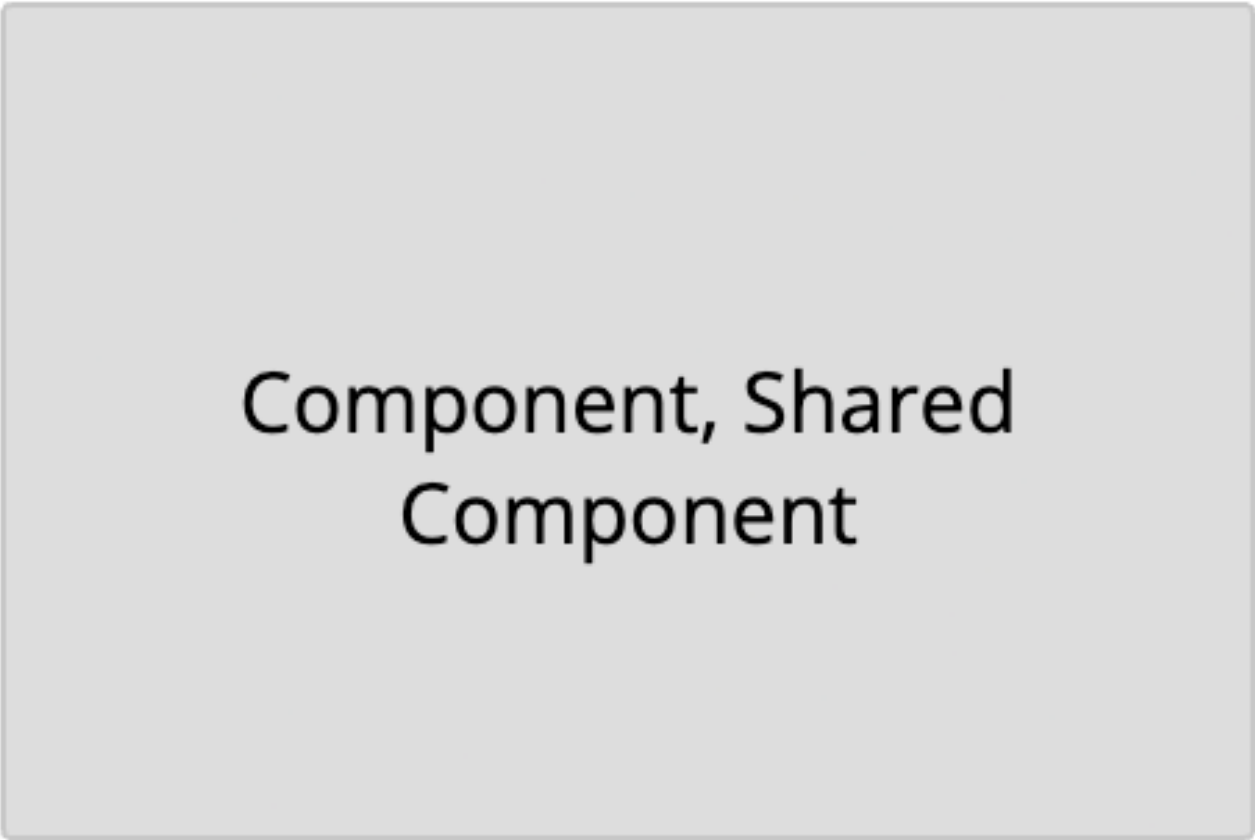
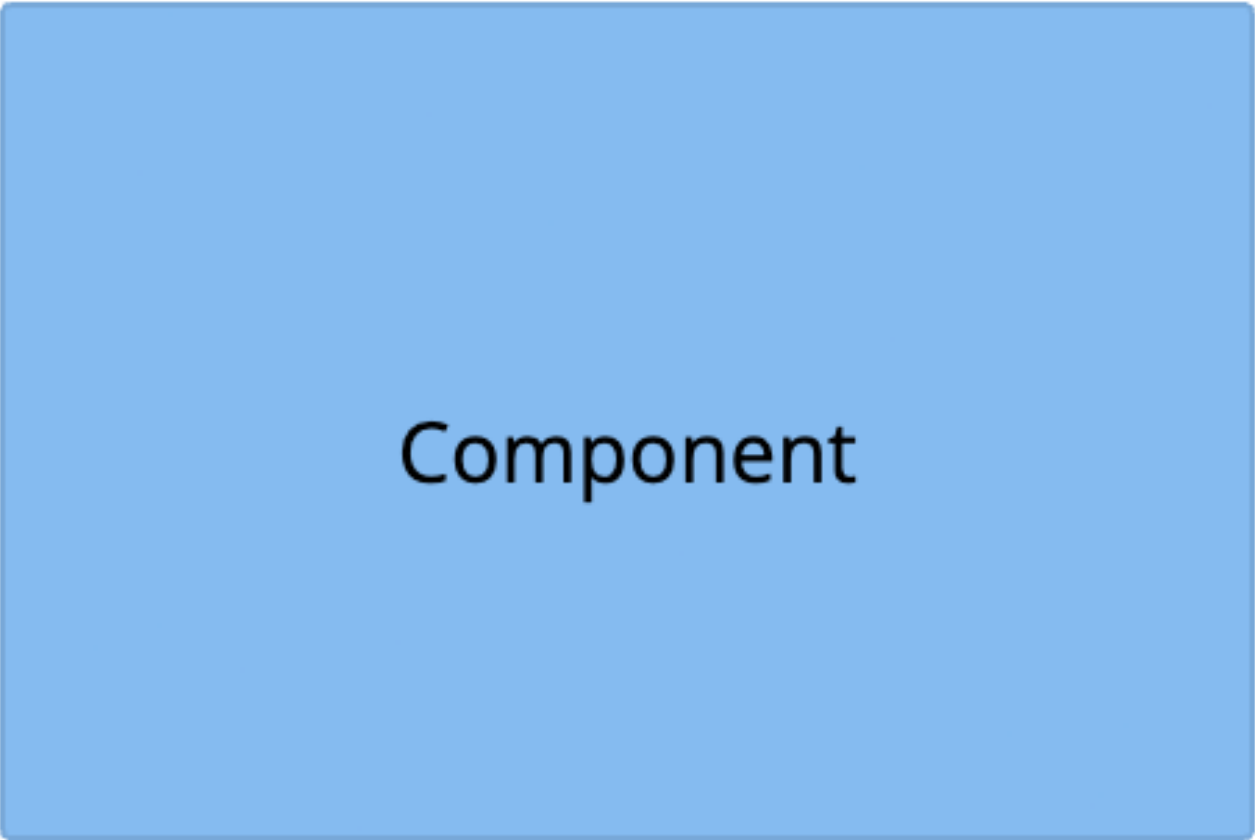
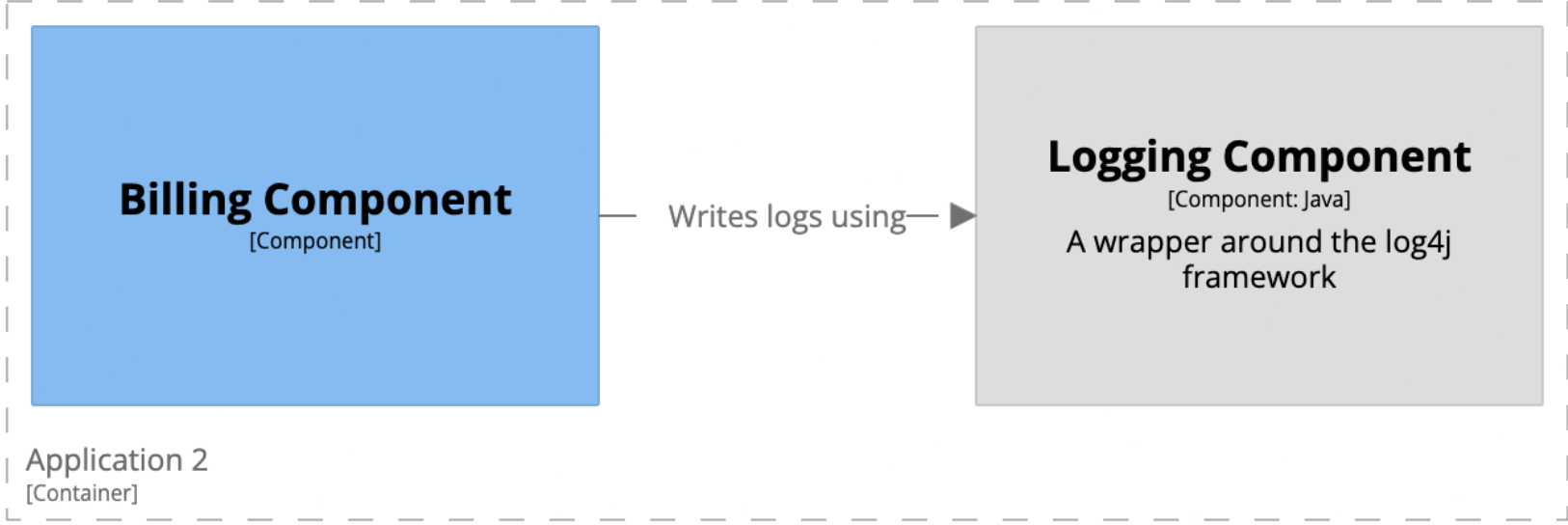
Shared libraries

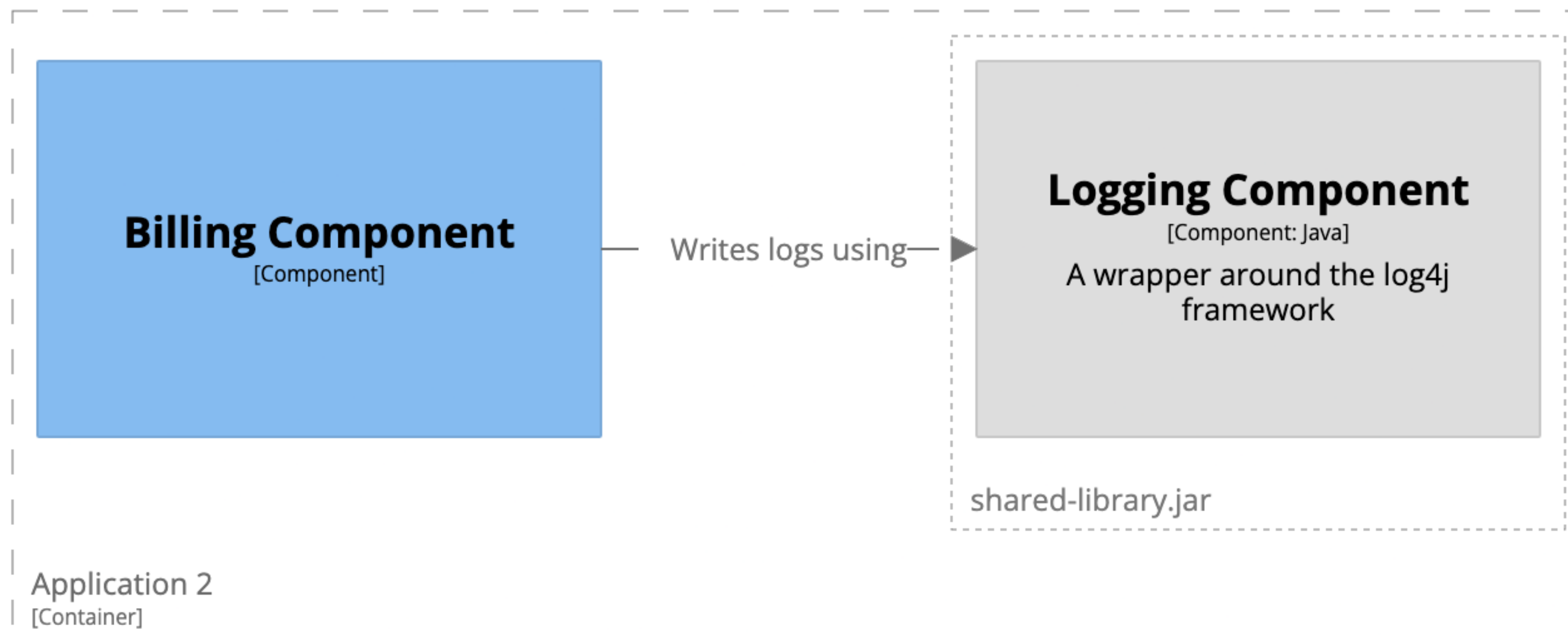
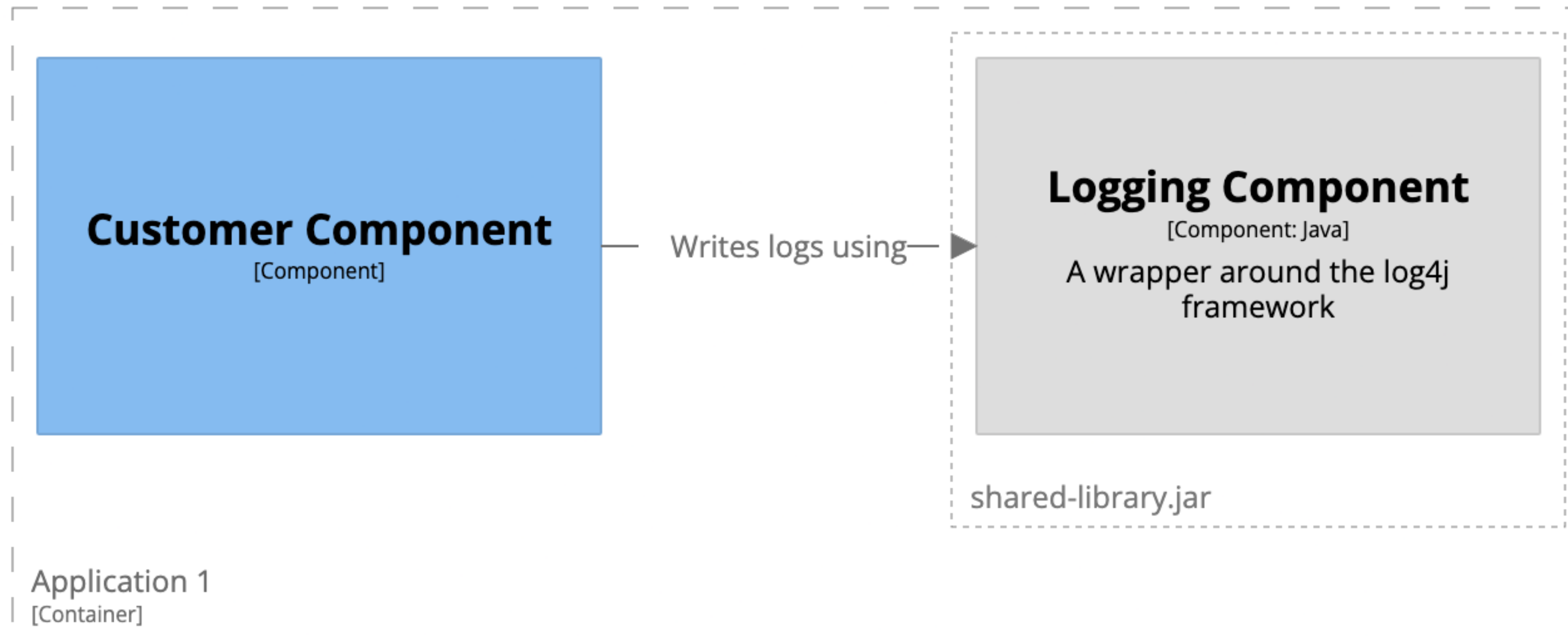














# Microservices

C4 is more suited to monolithic architectures, and doesn't support distributed architectures well

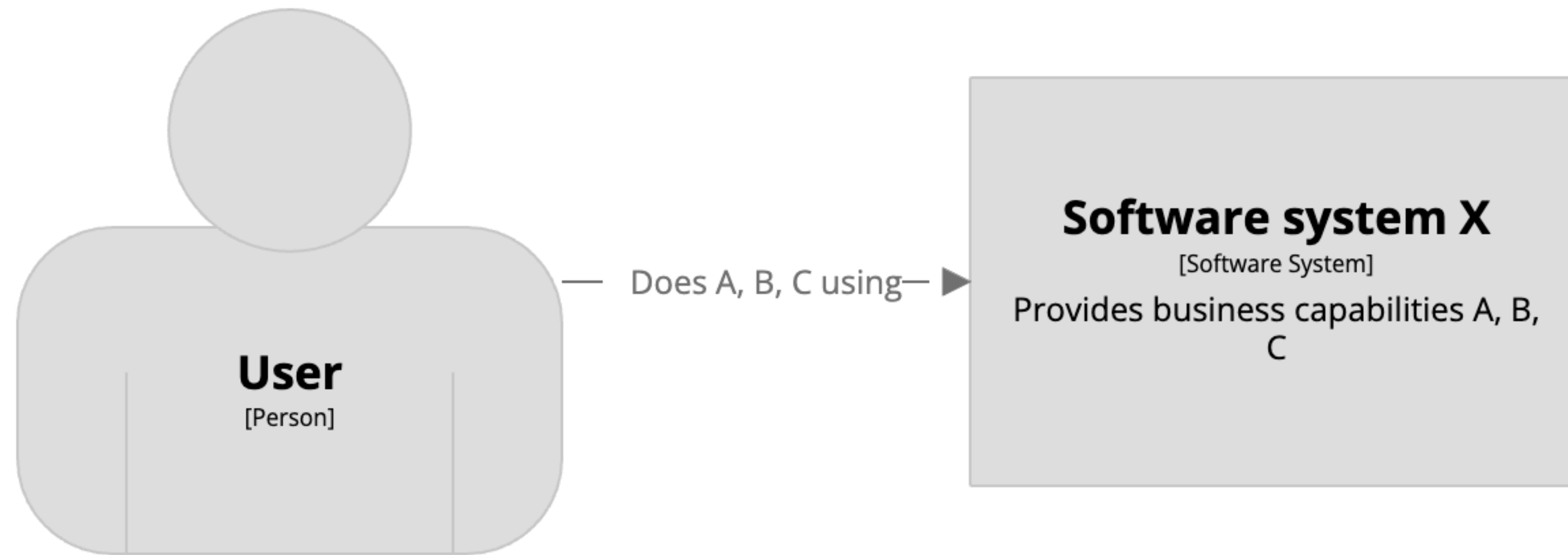
We're modelling microservices as  
containers, with APIs and database  
schemas as components

A microservice should be modelled  
as one of the following:

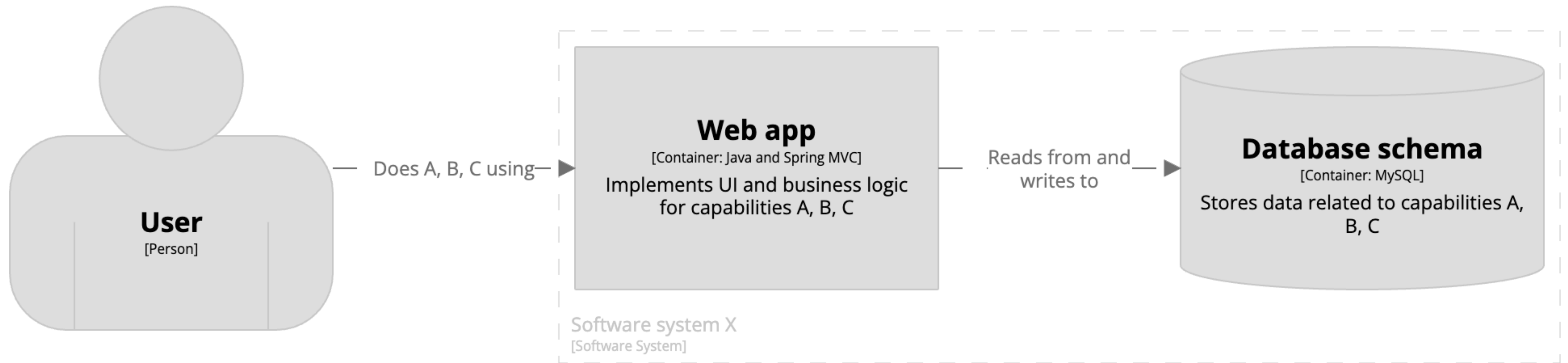
1. A software system
2. A container
3. A group of containers



Stage 1:   
(monolithic architecture)



[System Context] Software system X



[Container] Software system X

Stage 2:    
(microservices)



What is a  
“microservice”?

# Microservices

a definition of this new architectural term

*The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.*

25 March 2014



**James Lewis**

James Lewis is a Principal Consultant at Thoughtworks and member of the Technology Advisory Board. James' interest in building applications out of small collaborating services

## CONTENTS

### Characteristics of a Microservice Architecture

Componentization via Services

Organized around Business Capabilities

Products not Projects

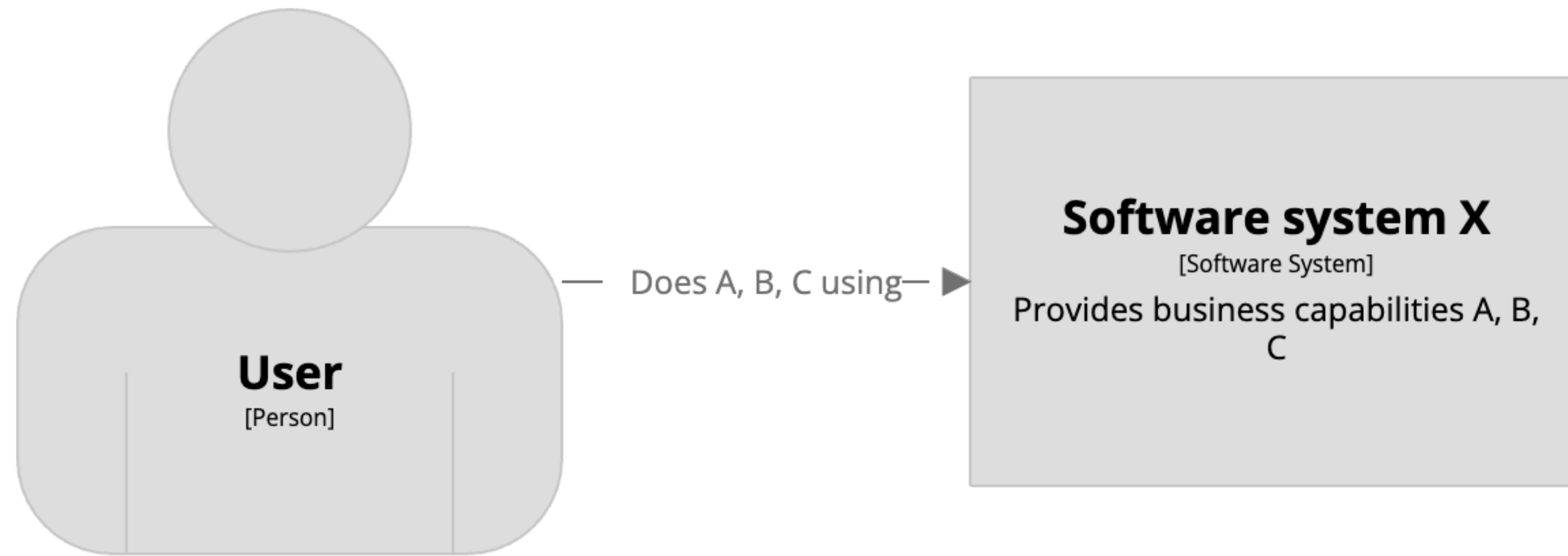
Smart endpoints and dumb pipes

Decentralized Governance

Decentralized Data Management

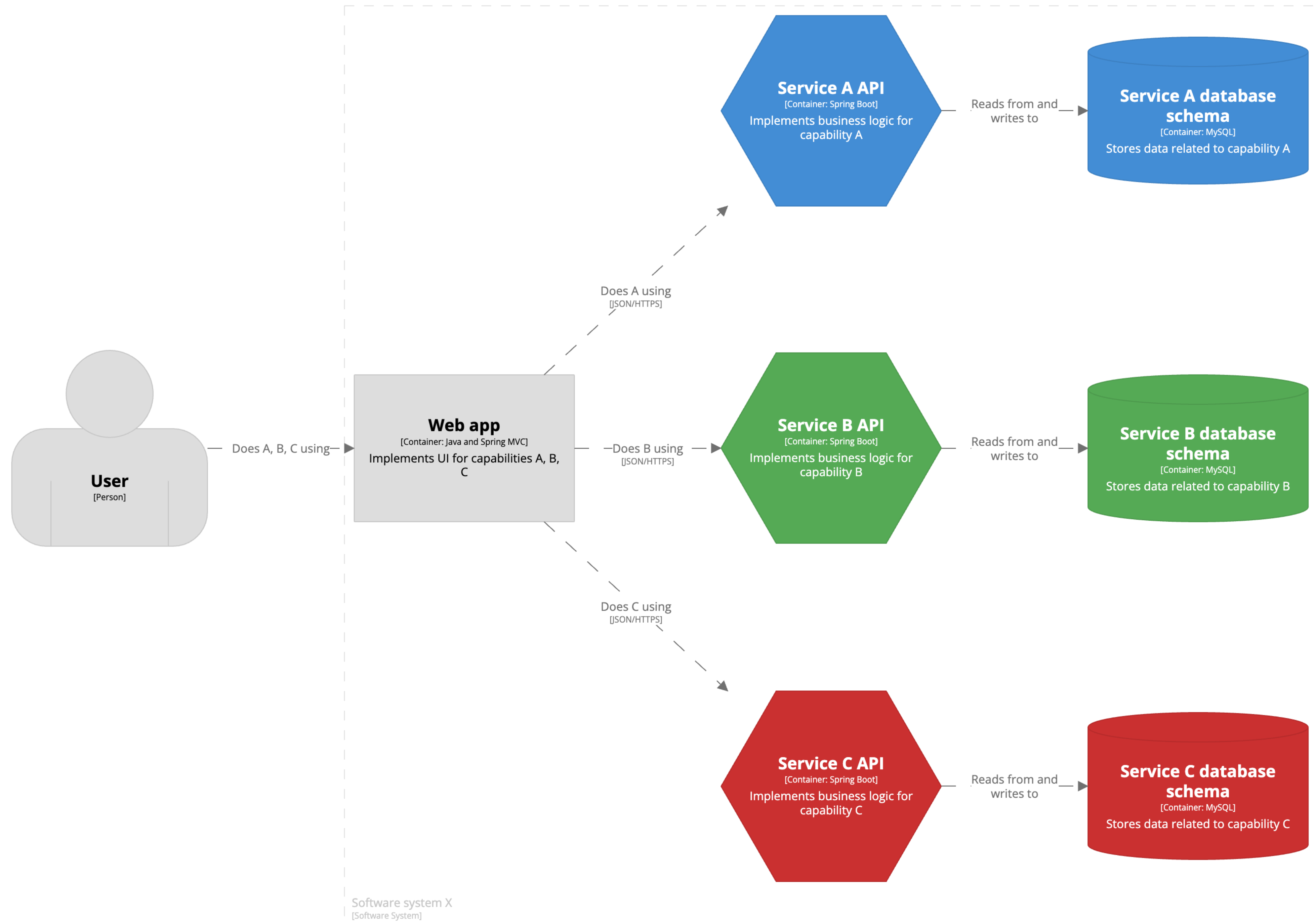


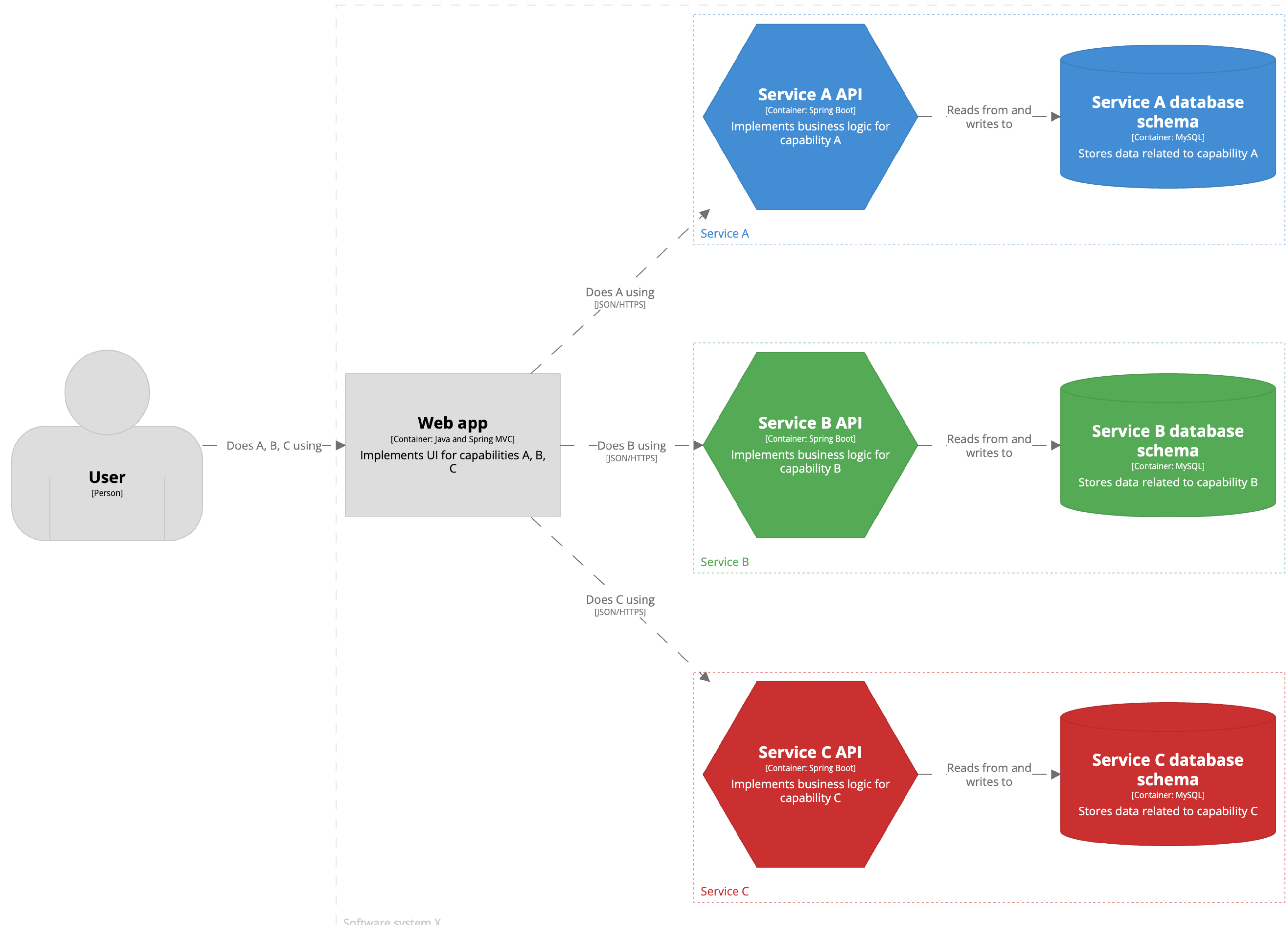
In short, the microservice architectural style [1] is an approach to developing a single software system as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.

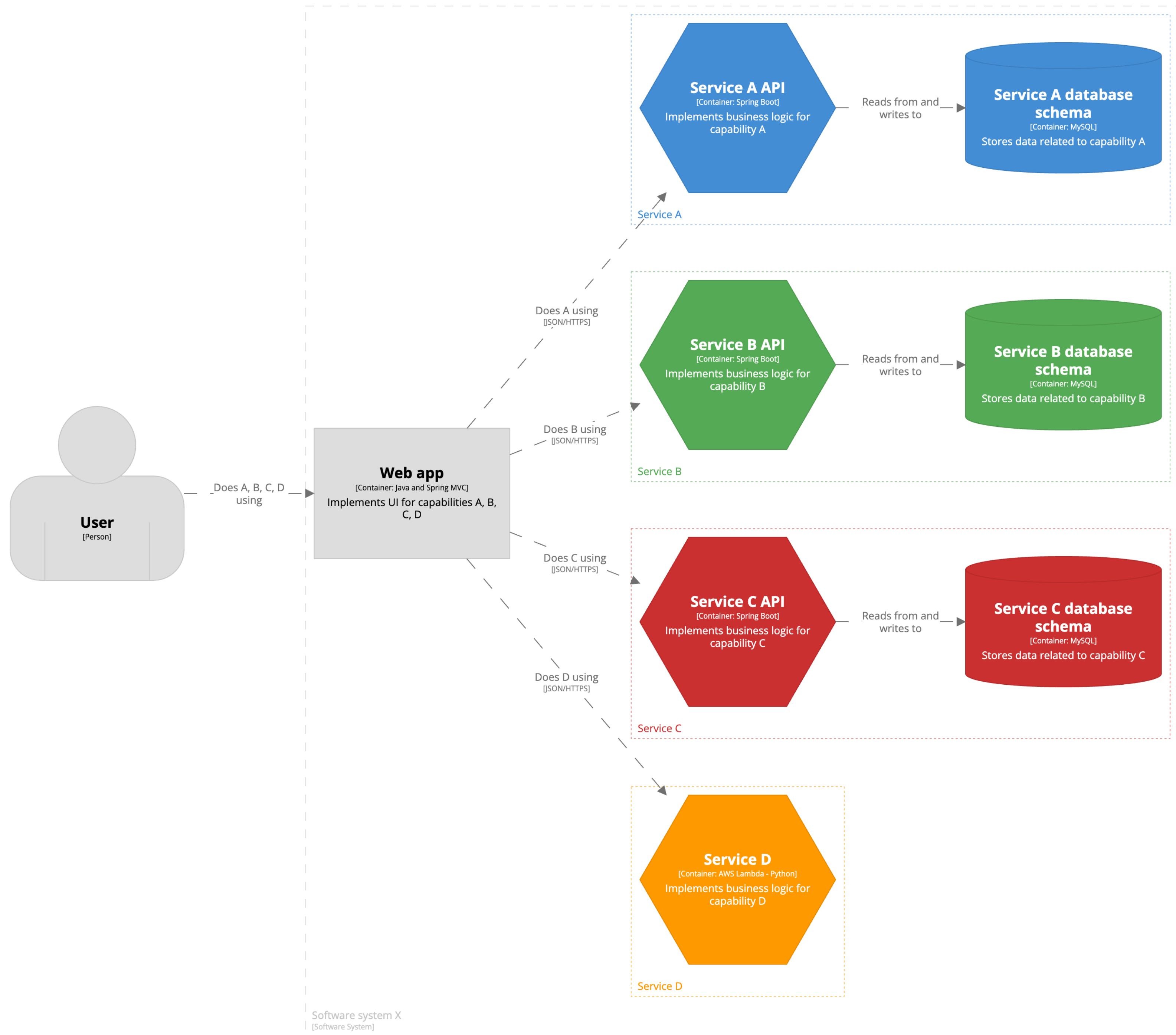


[System Context] Software system X



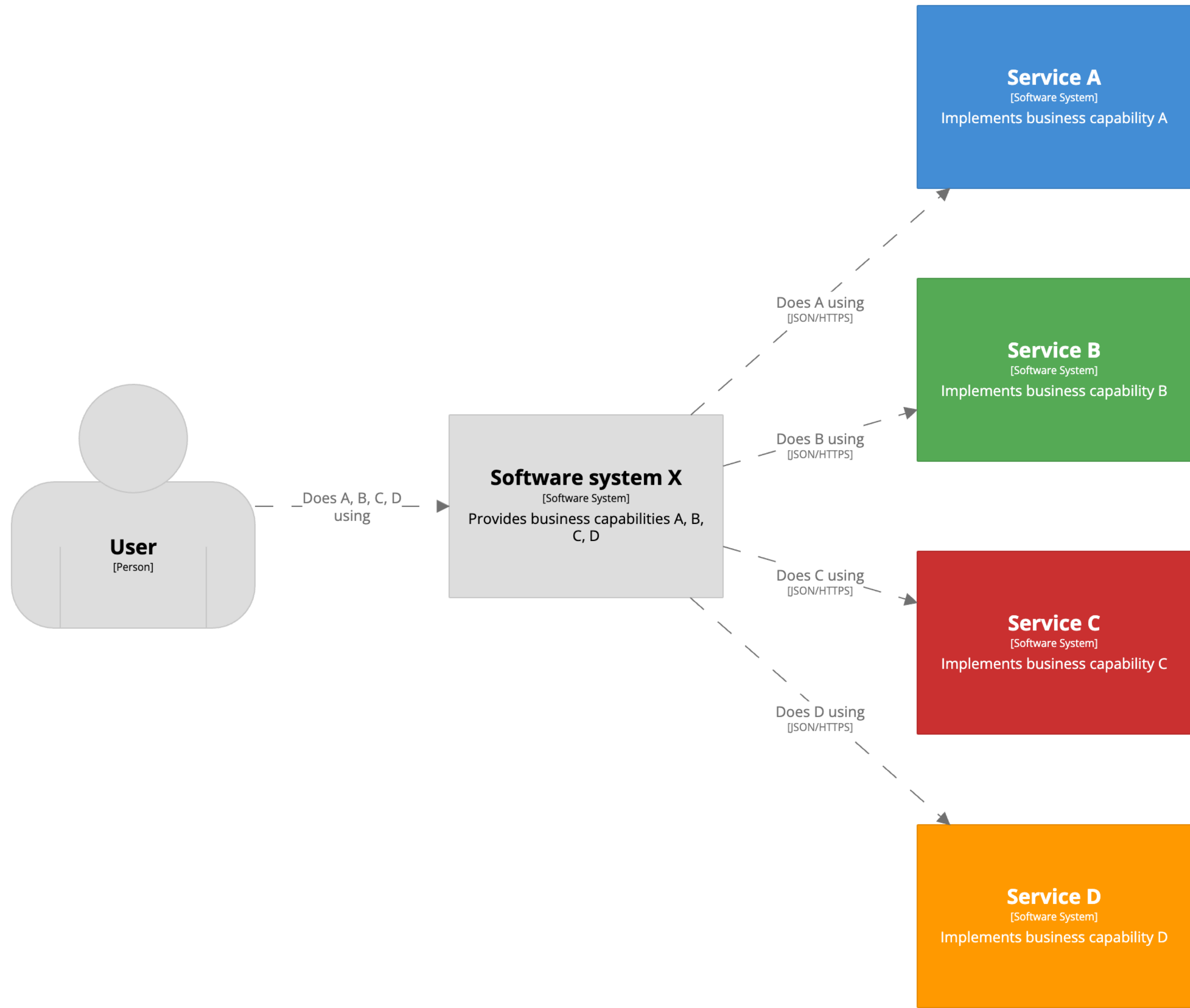


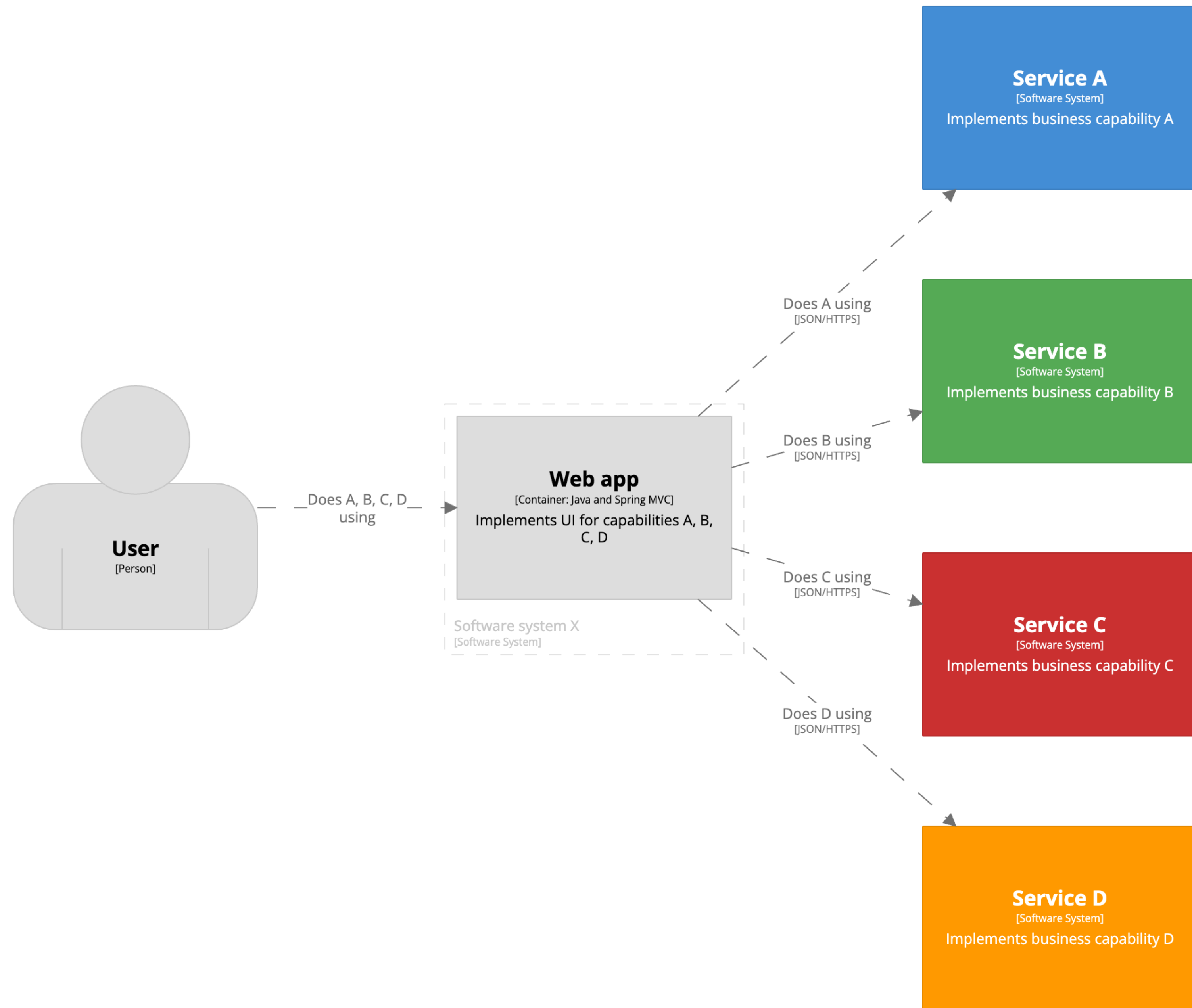




Stage 3:   
(Conway's Law)

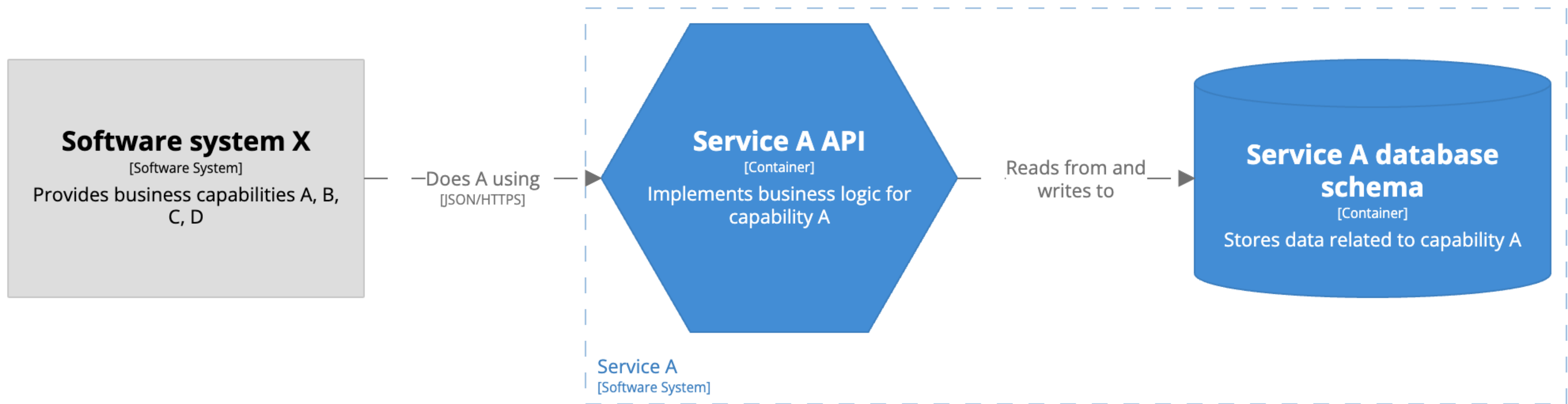








[System Context] Service A



[Container] Service A



# Micro frontends

# Micro Frontends

*Good frontend development is hard. Scaling frontend development so that many teams can work simultaneously on a large and complex product is even harder. In this article we'll describe a recent trend of breaking up frontend monoliths into many smaller, more manageable pieces, and how this architecture can increase the effectiveness and efficiency of teams working on frontend code. As well as talking about the various benefits and costs, we'll cover some of the implementation options that are available, and we'll dive deep into a full example application that demonstrates the technique.*

19 June 2019



**Cam Jackson**

Cam Jackson is a full-stack web developer and consultant at Thoughtworks, with a particular interest in how large organisations scale their frontend development process and practices. He has worked with clients across multiple

## CONTENTS

### Benefits

[Incremental upgrades](#)

[Simple, decoupled codebases](#)

[Independent deployment](#)

[Autonomous teams](#)

[In a nutshell](#)

### The example

[Integration approaches](#)

[Summary and next steps](#)

Dependencies to  
“external” containers

My recommendation is that container diagrams only show containers inside the software system that is the scope of the diagram

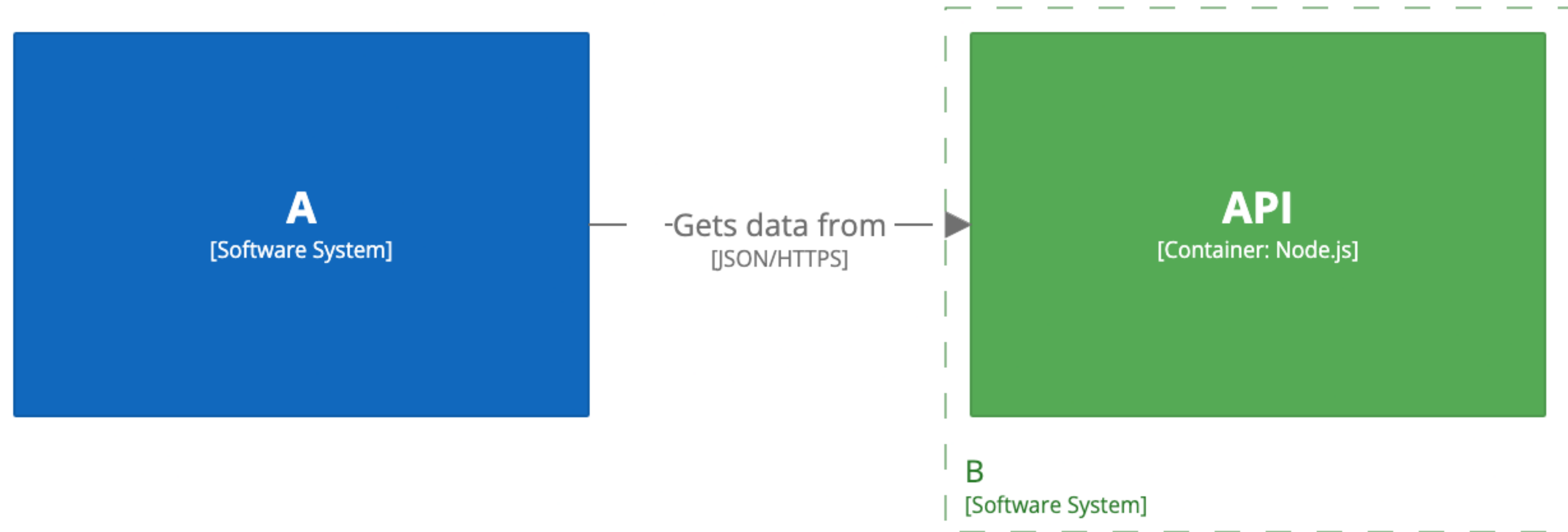




Container diagram for software system A

```
container a {  
    include *  
}
```



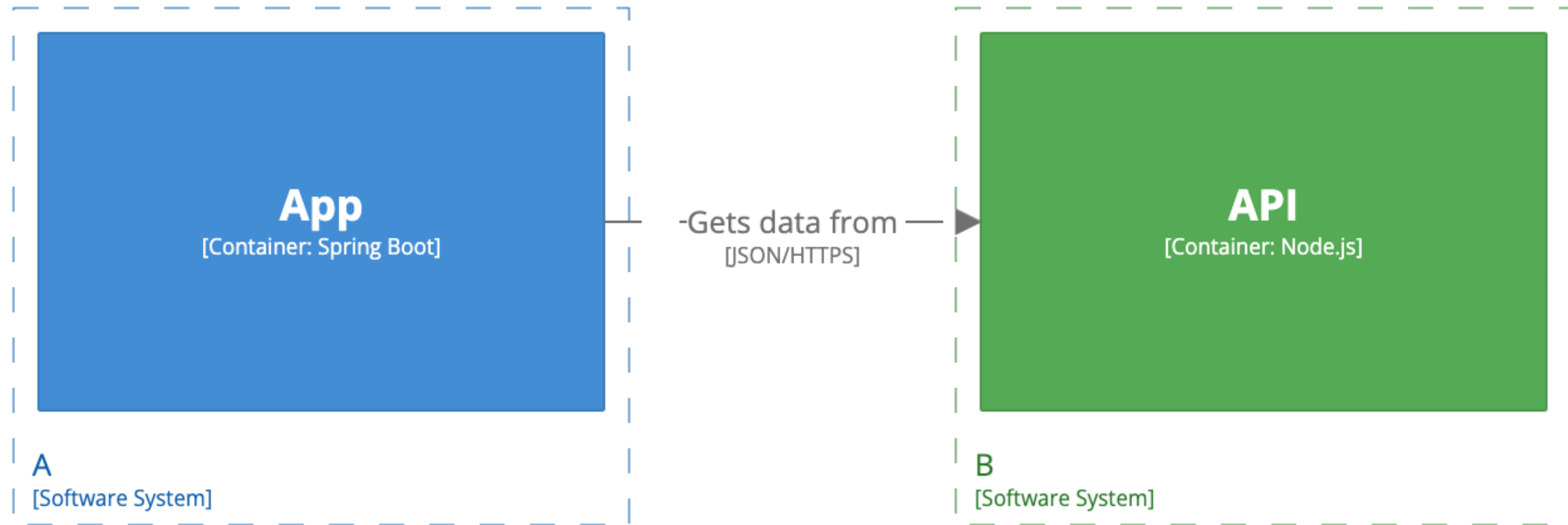


Container diagram for software system B

```
container b {  
    include *  
}
```



I don't recommend showing  
“external” containers



Container diagram for software systems A and B

```
container a {  
    include a.app b.api  
}
```

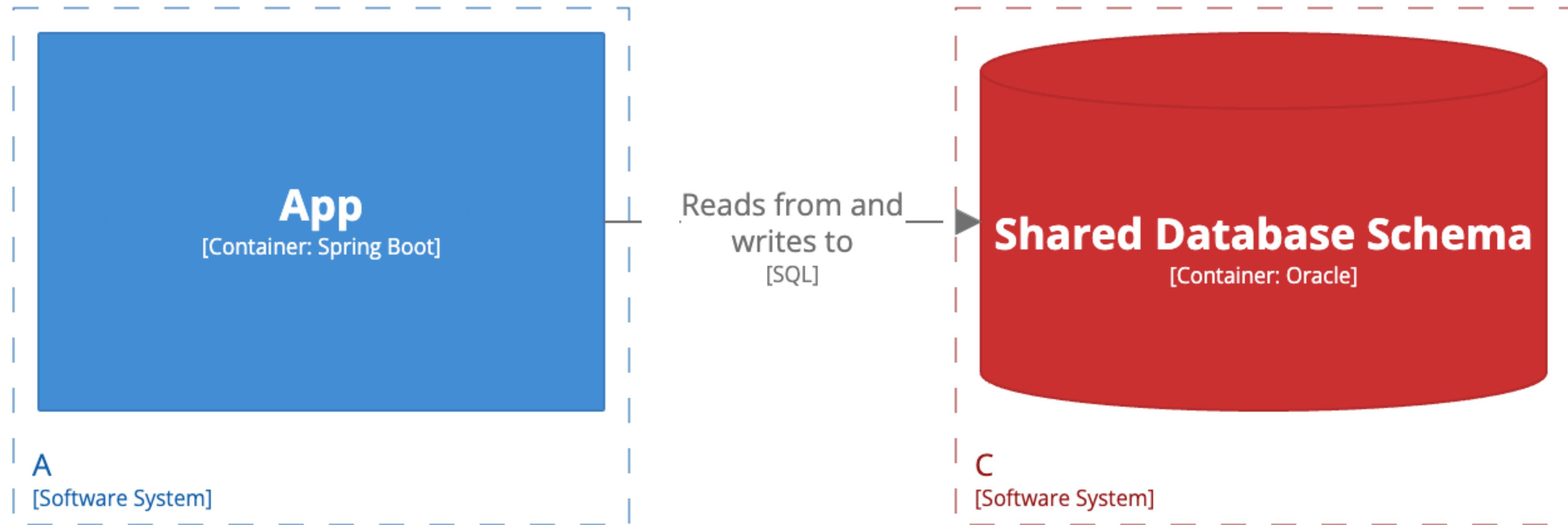




Showing “external” containers implies  
some understanding of  
implementation details, which makes  
the diagrams more volatile to change

This is a form of coupling

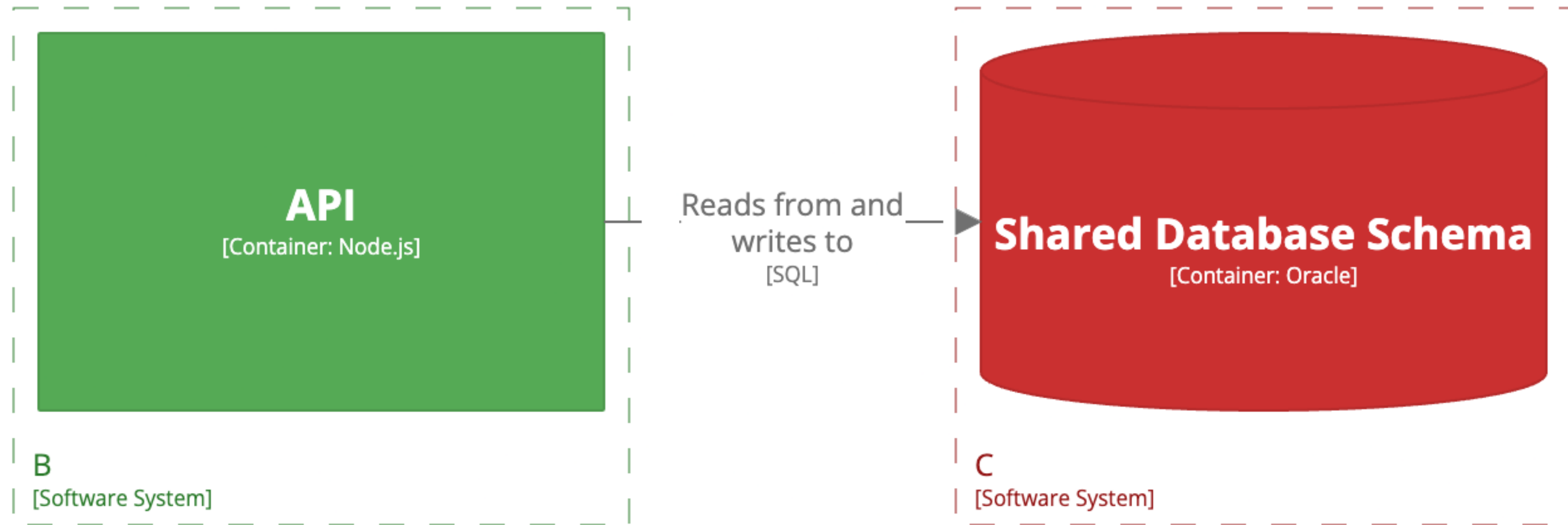
There may some useful exceptions  
to this guidance...



Container diagram for software system A, showing a shared DB

```
container a {  
    include a.app c.db  
}
```





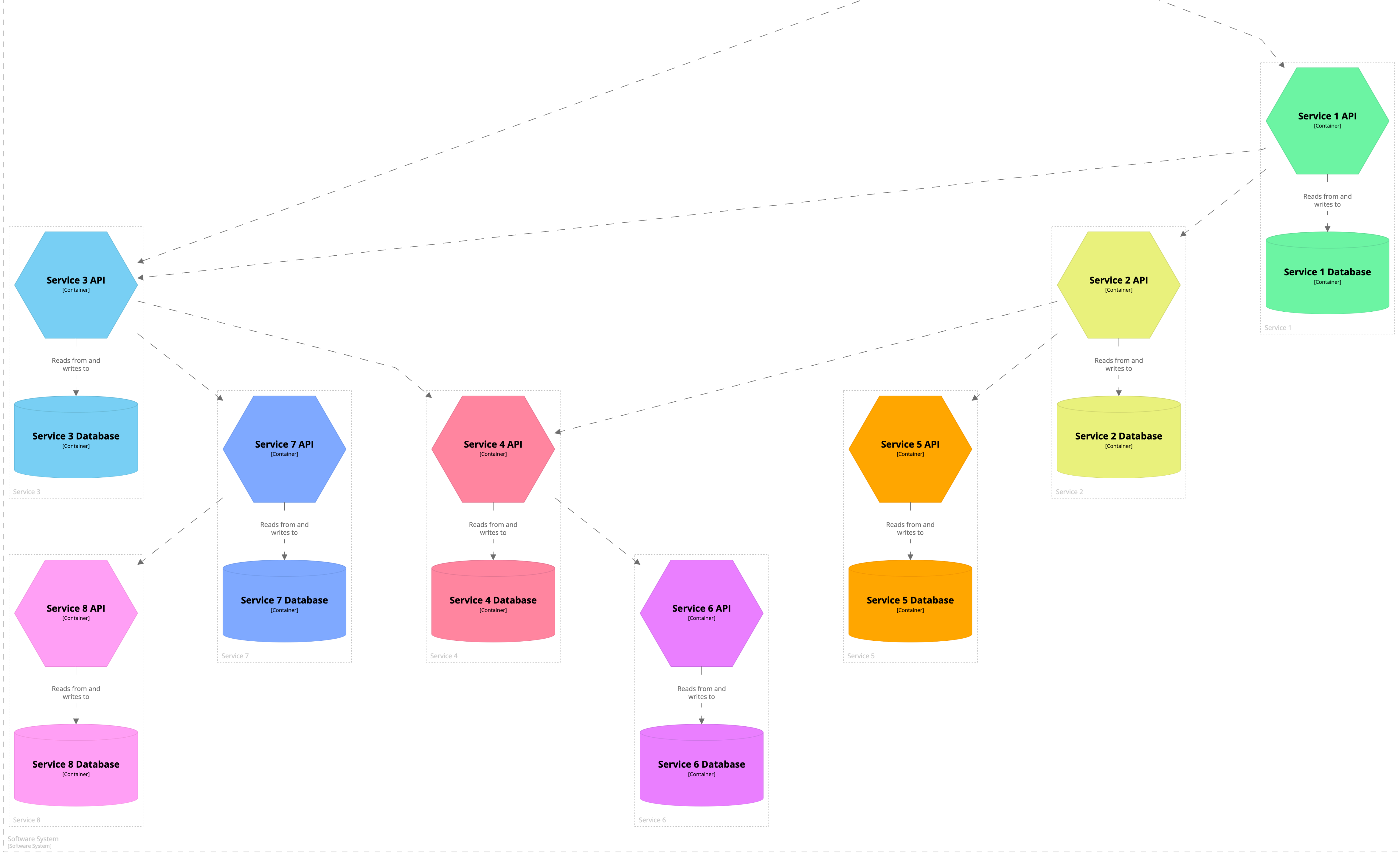
Container diagram for software system B, showing a shared DB

```
container b {  
    include b.api c.db  
}
```

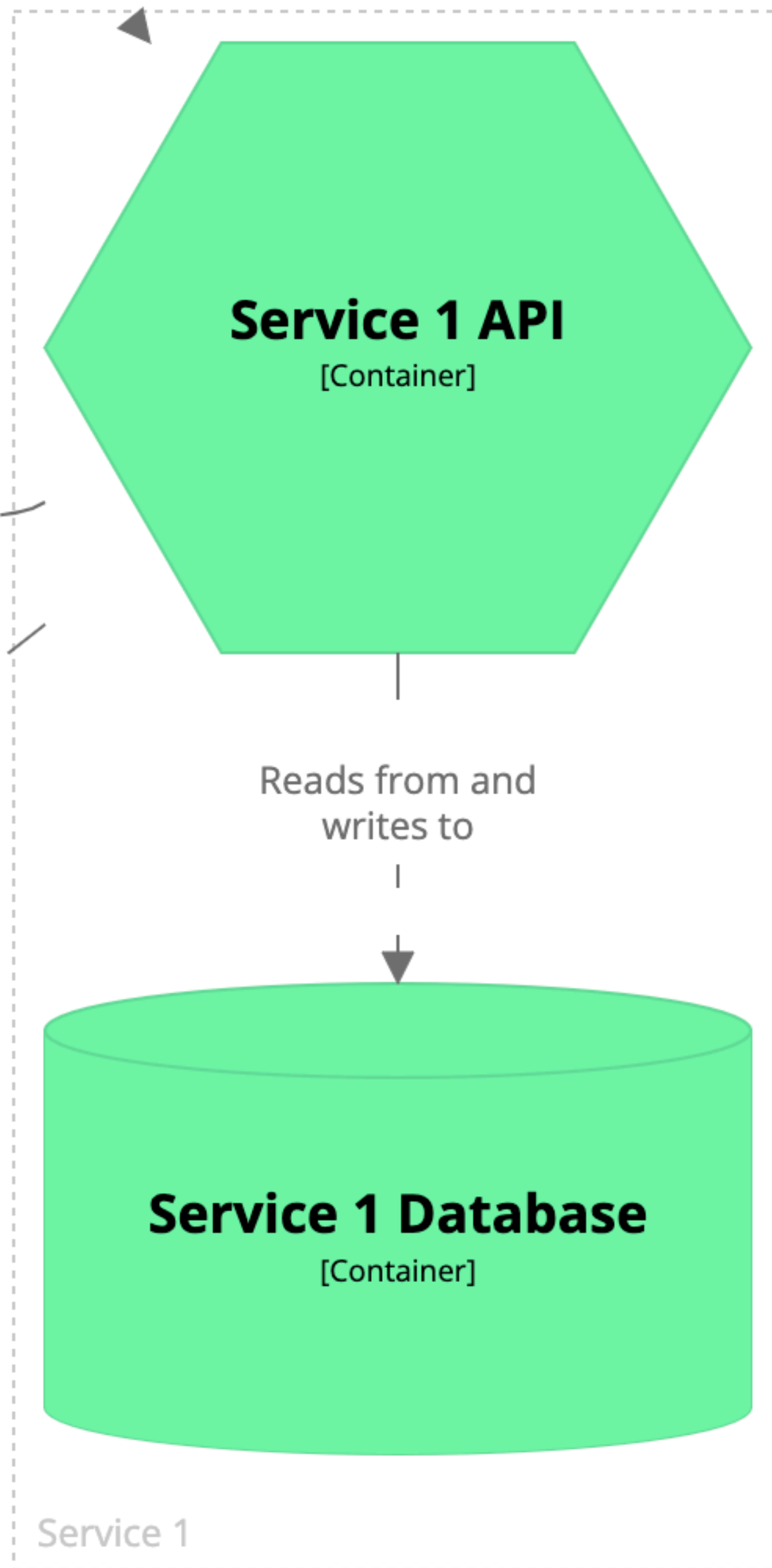
# Tooling



C4 doesn't scale

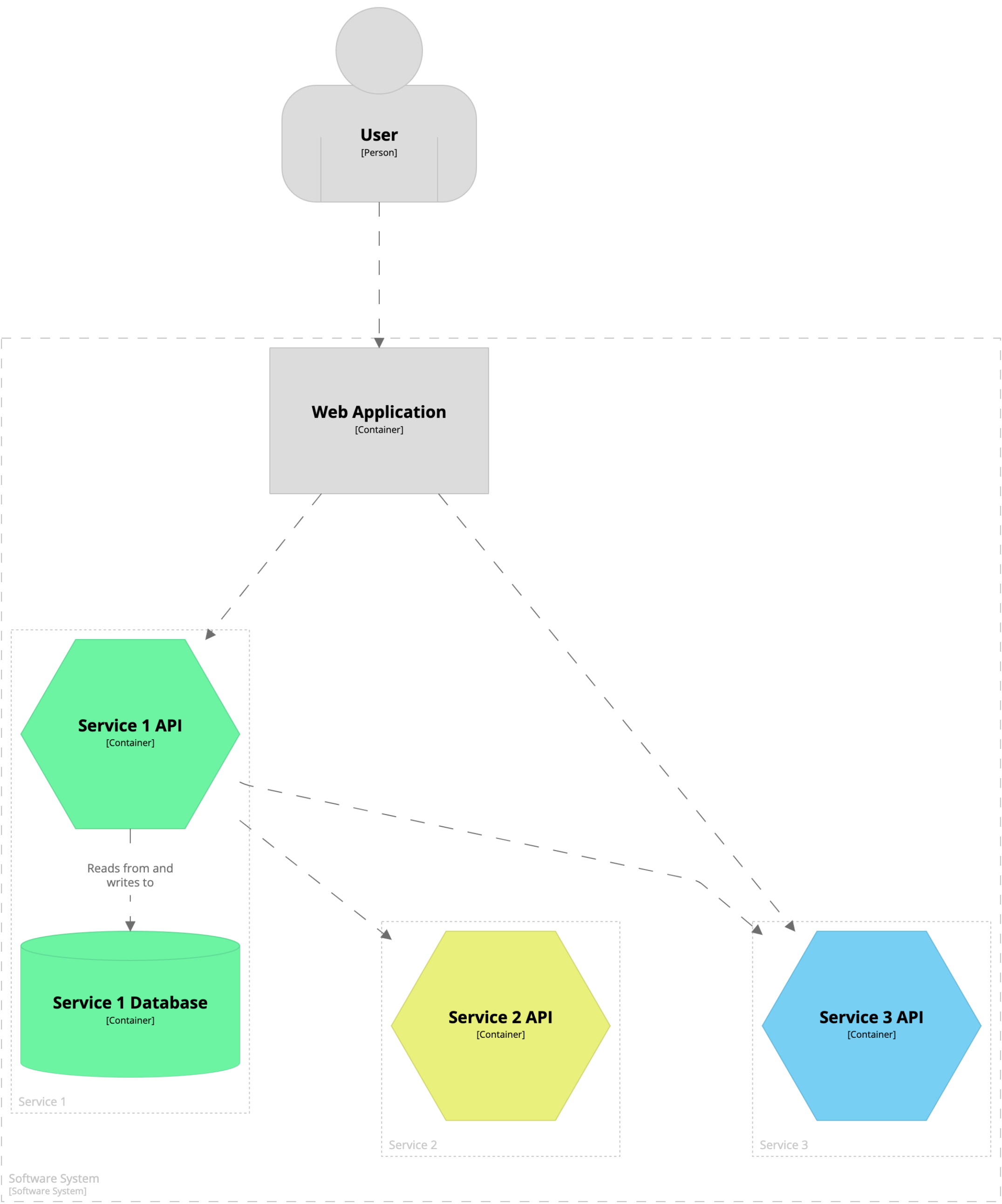




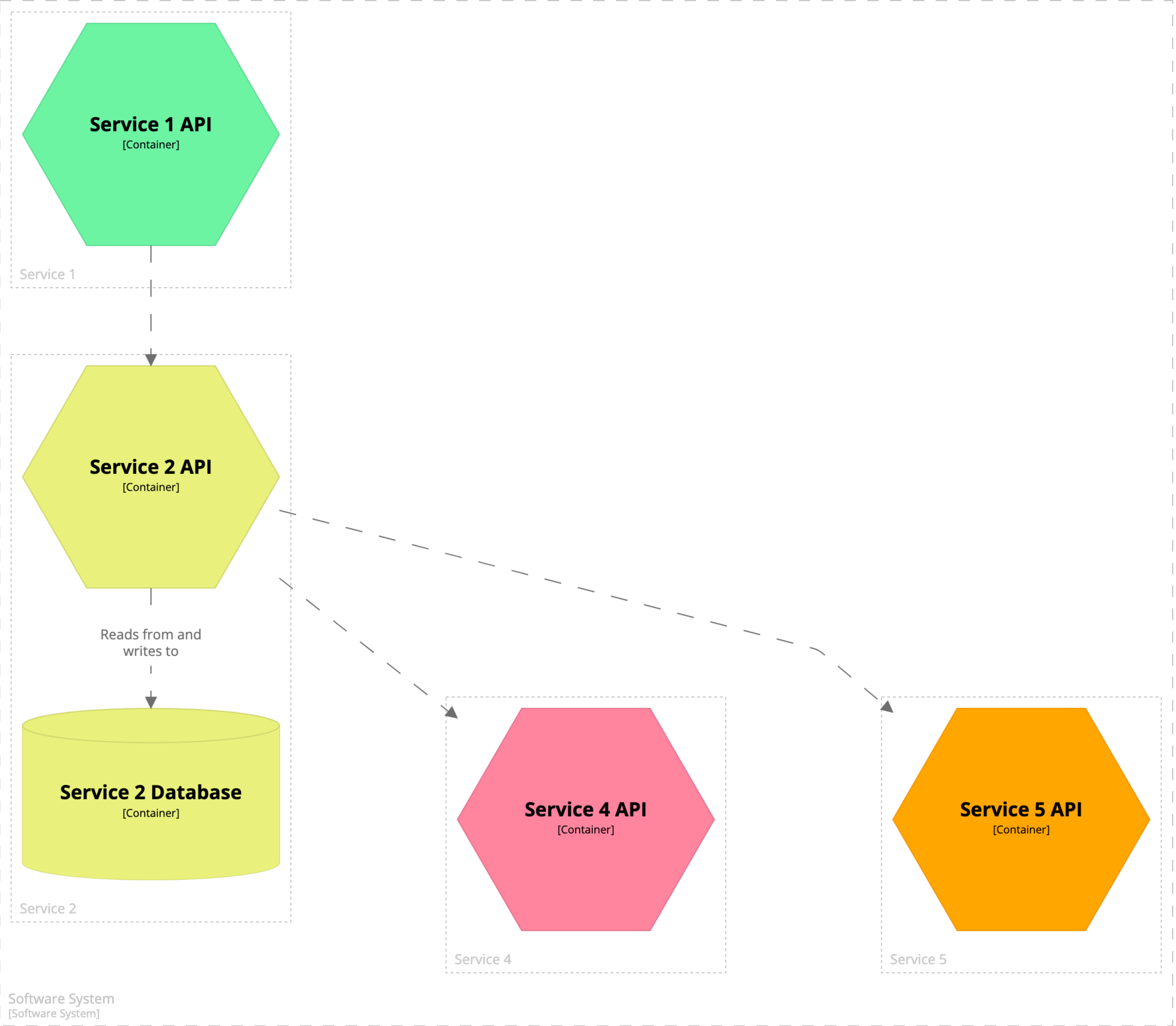


In this example,  
a microservice is  
a combination of  
an API and  
a database schema

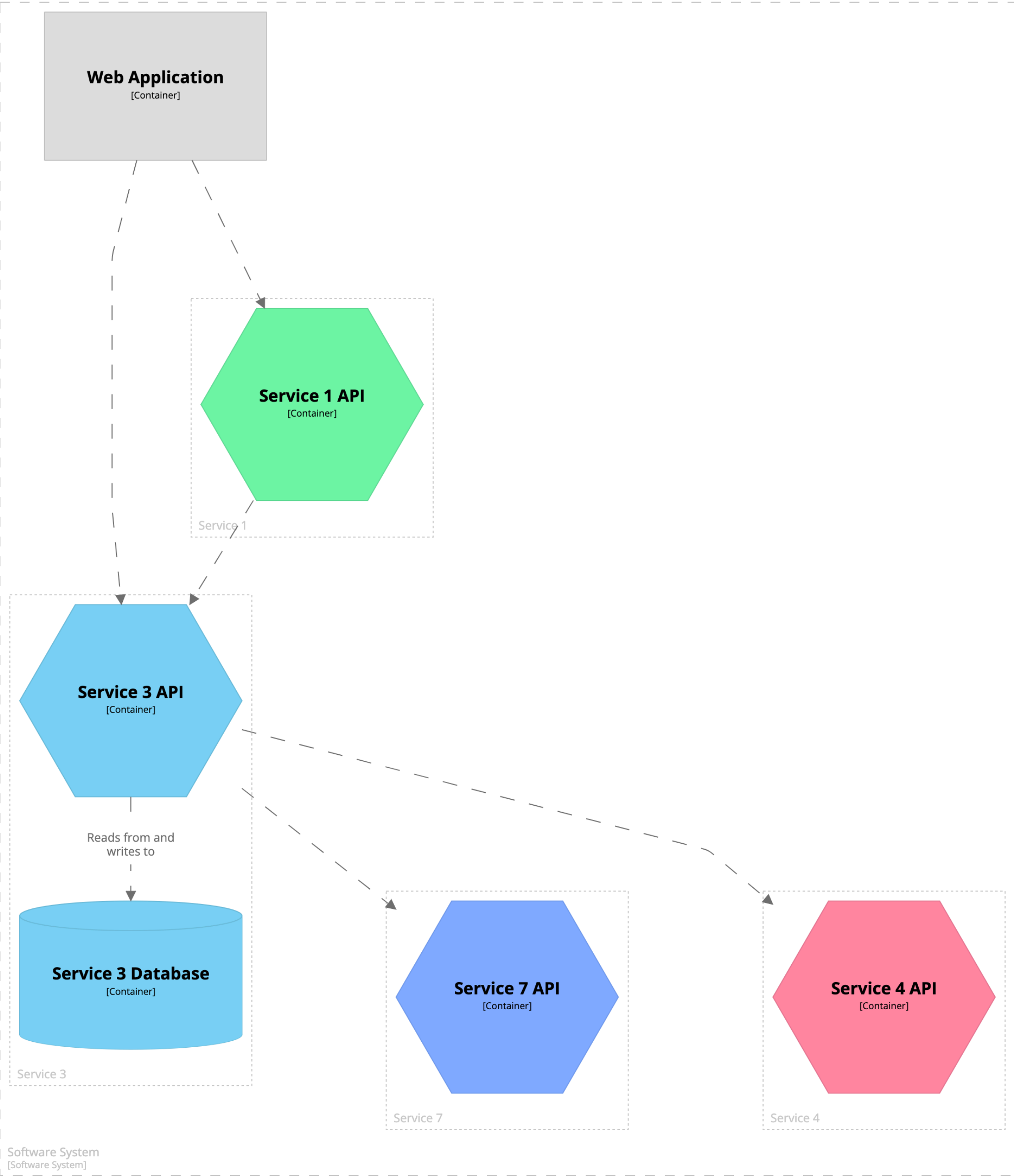
```
container softwareSystem {
  include user
  Include ->service1->
}
```



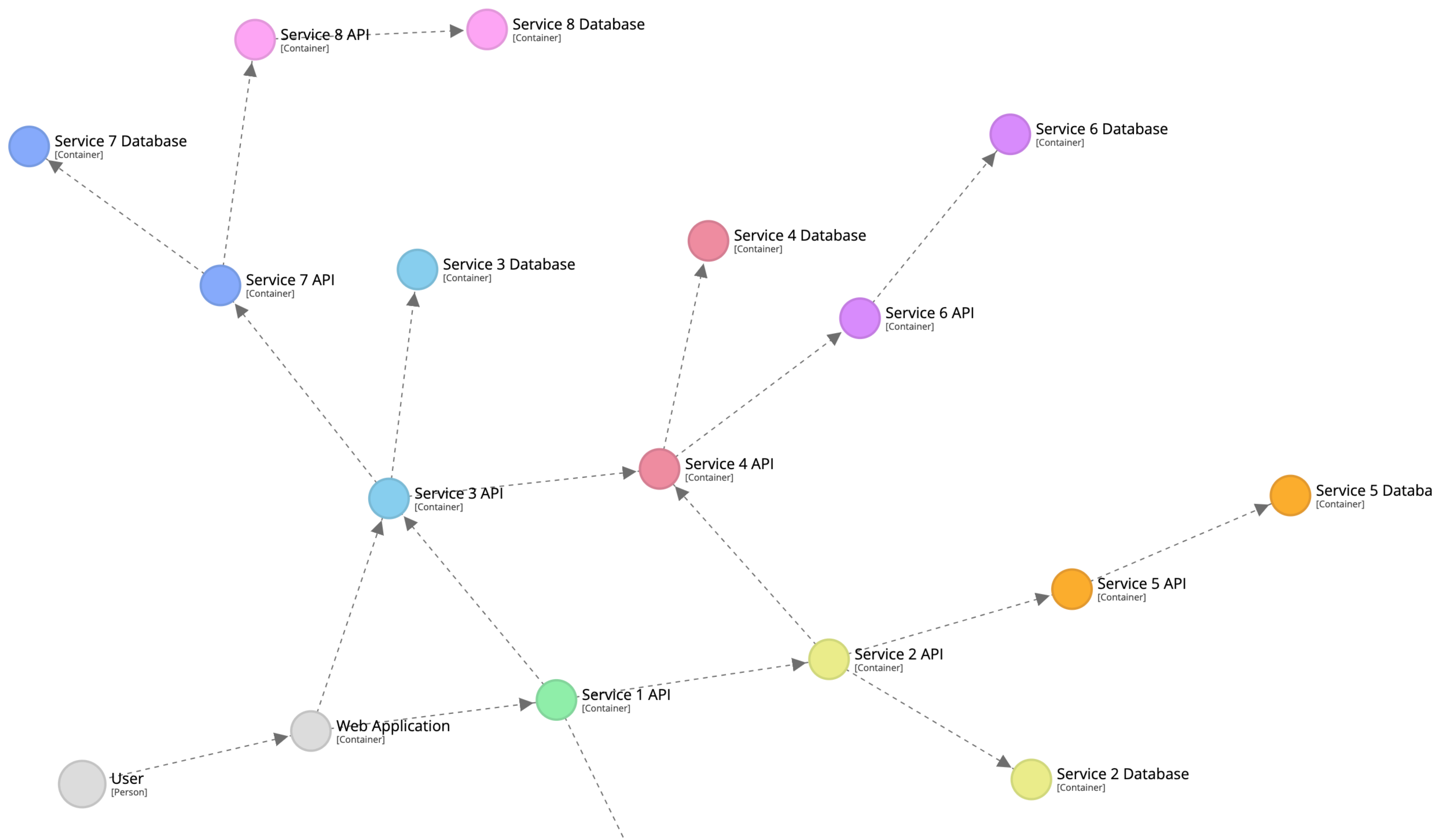
```
container softwareSystem {
  include ->service2->
}
```

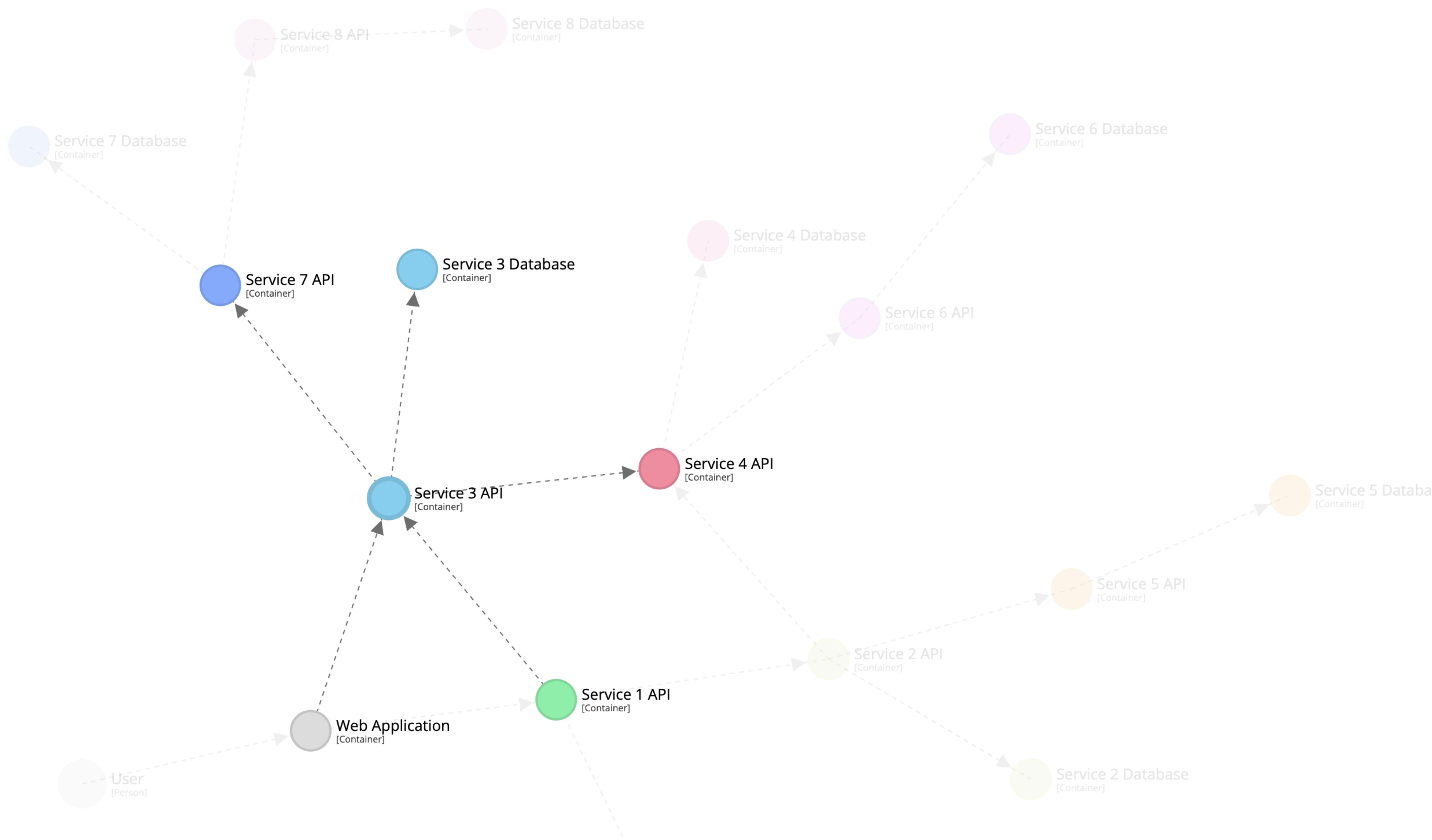


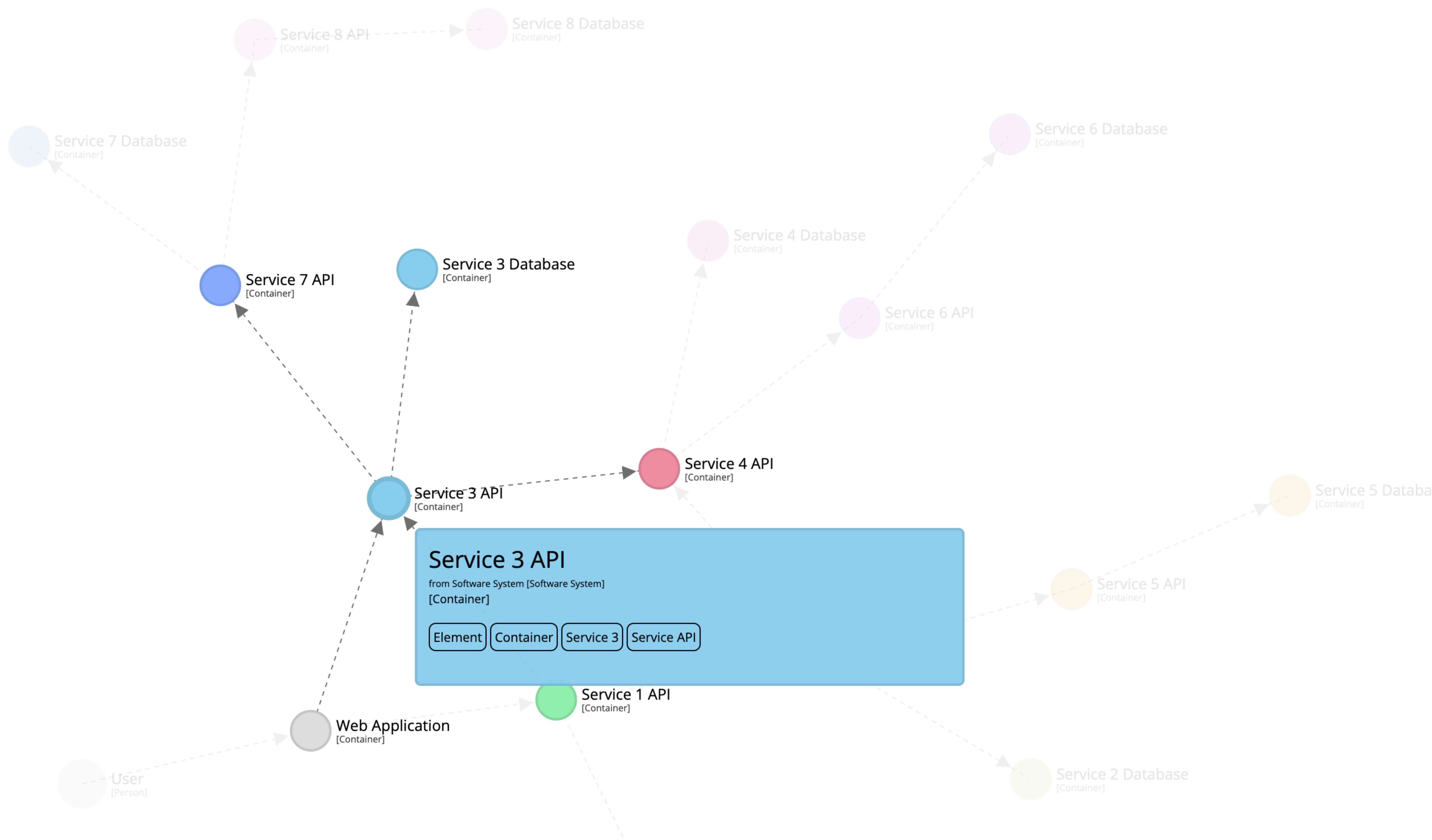
```
container softwareSystem {
  include ->service3->
}
```

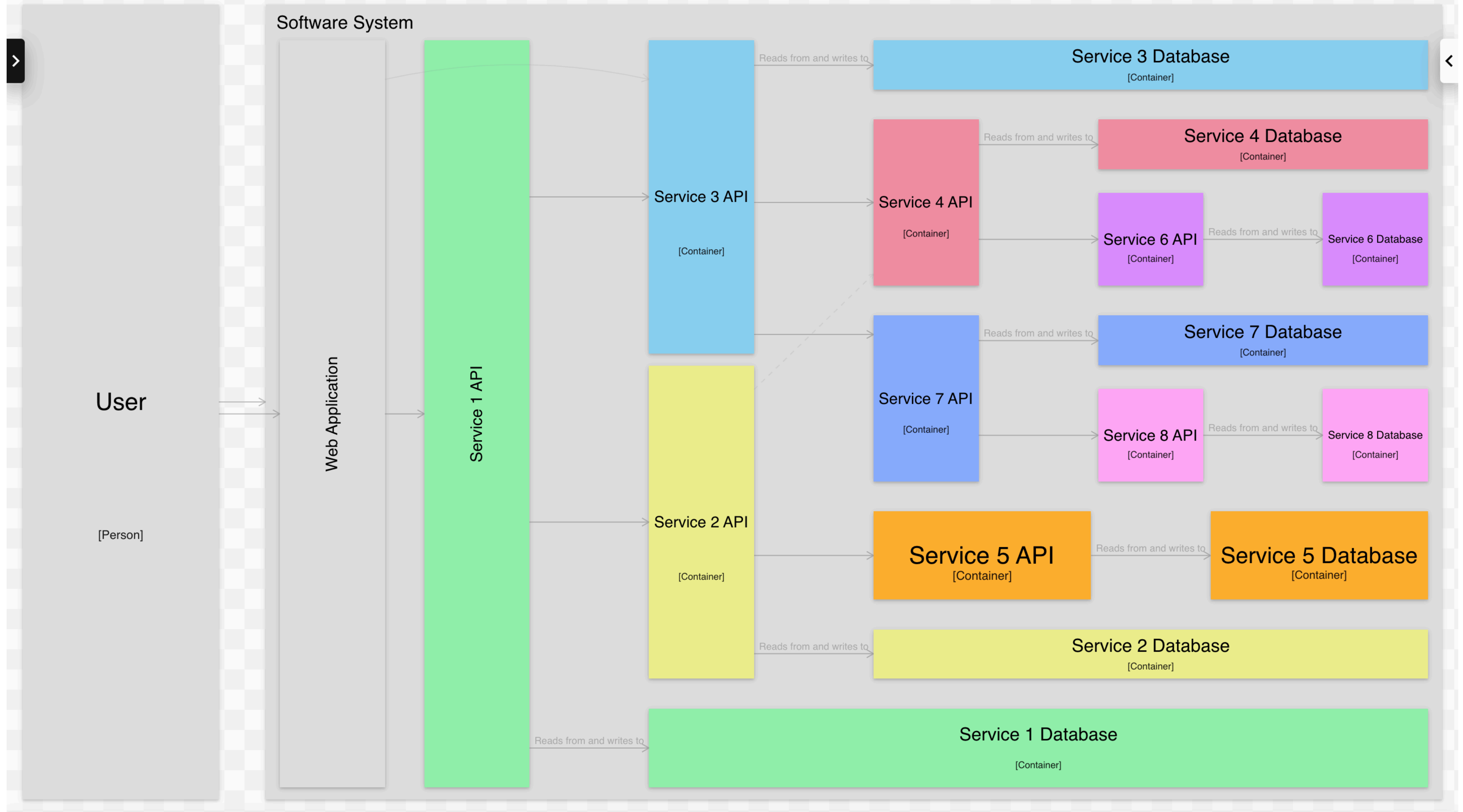




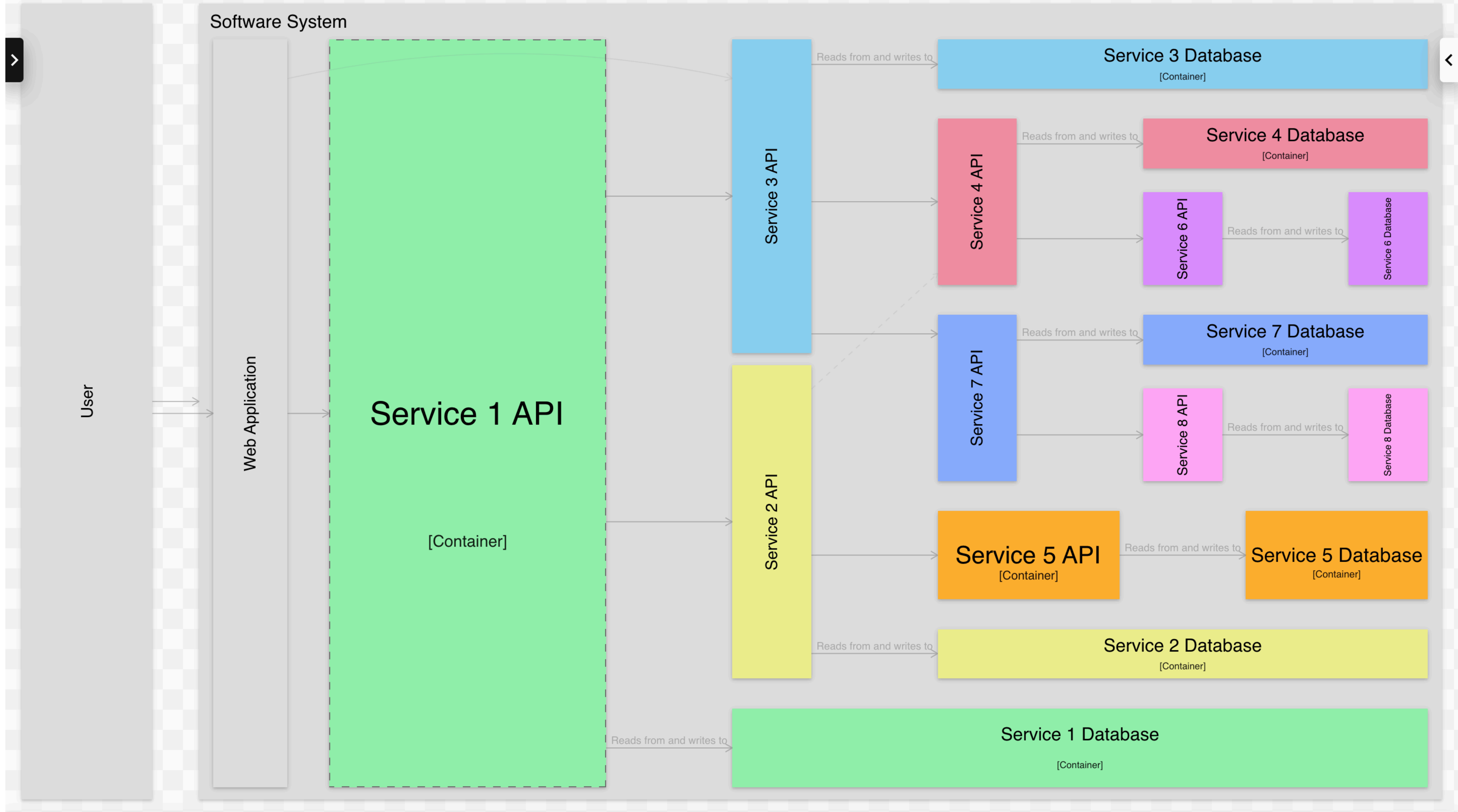


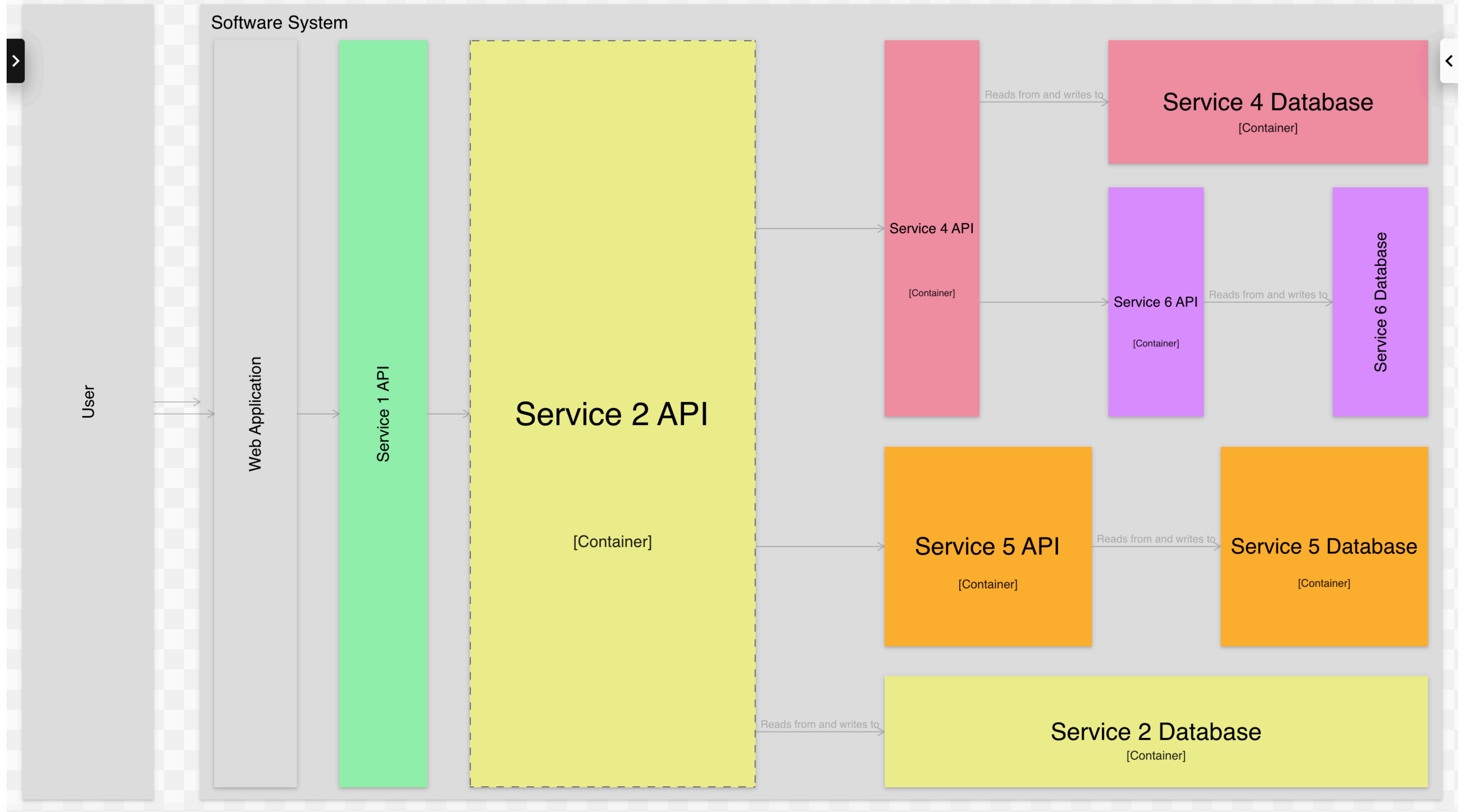


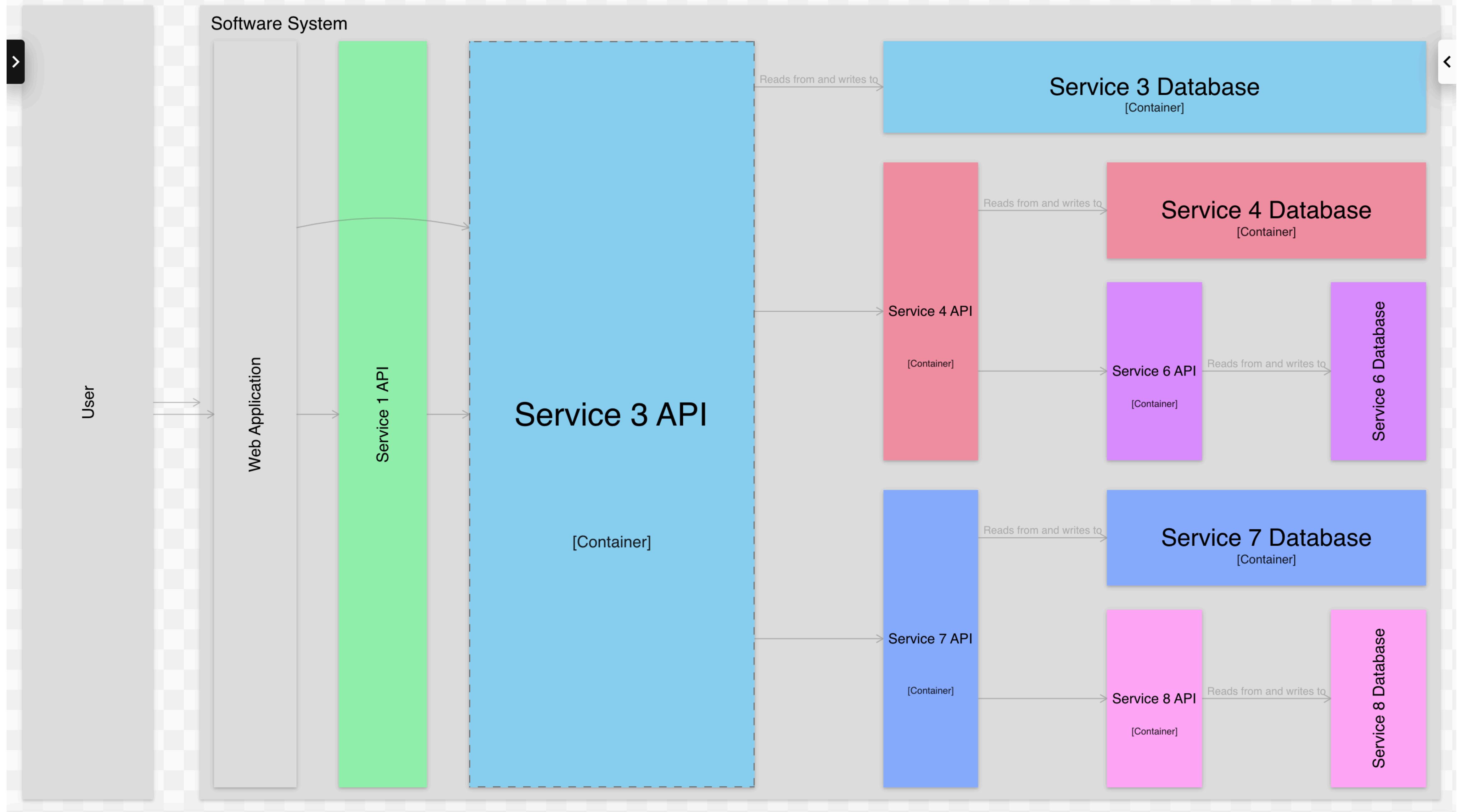












C4 still doesn't solve the problem  
of diagrams becoming out of date



# The C4 model is...

A set of hierarchical  
abstractions

(software systems, containers,  
components, and code)

A set of hierarchical  
diagrams

(system context, containers, components,  
and code)

Notation independent

Tooling independent

# System landscape and context diagrams

Use existing system/service catalogs  
(e.g. Backstage, ServiceNow, etc) as a source of data  
for identifying software systems and relationships

# Container diagrams

Parse log files or use OpenTelemetry data as a source of data for identifying applications/services and relationships

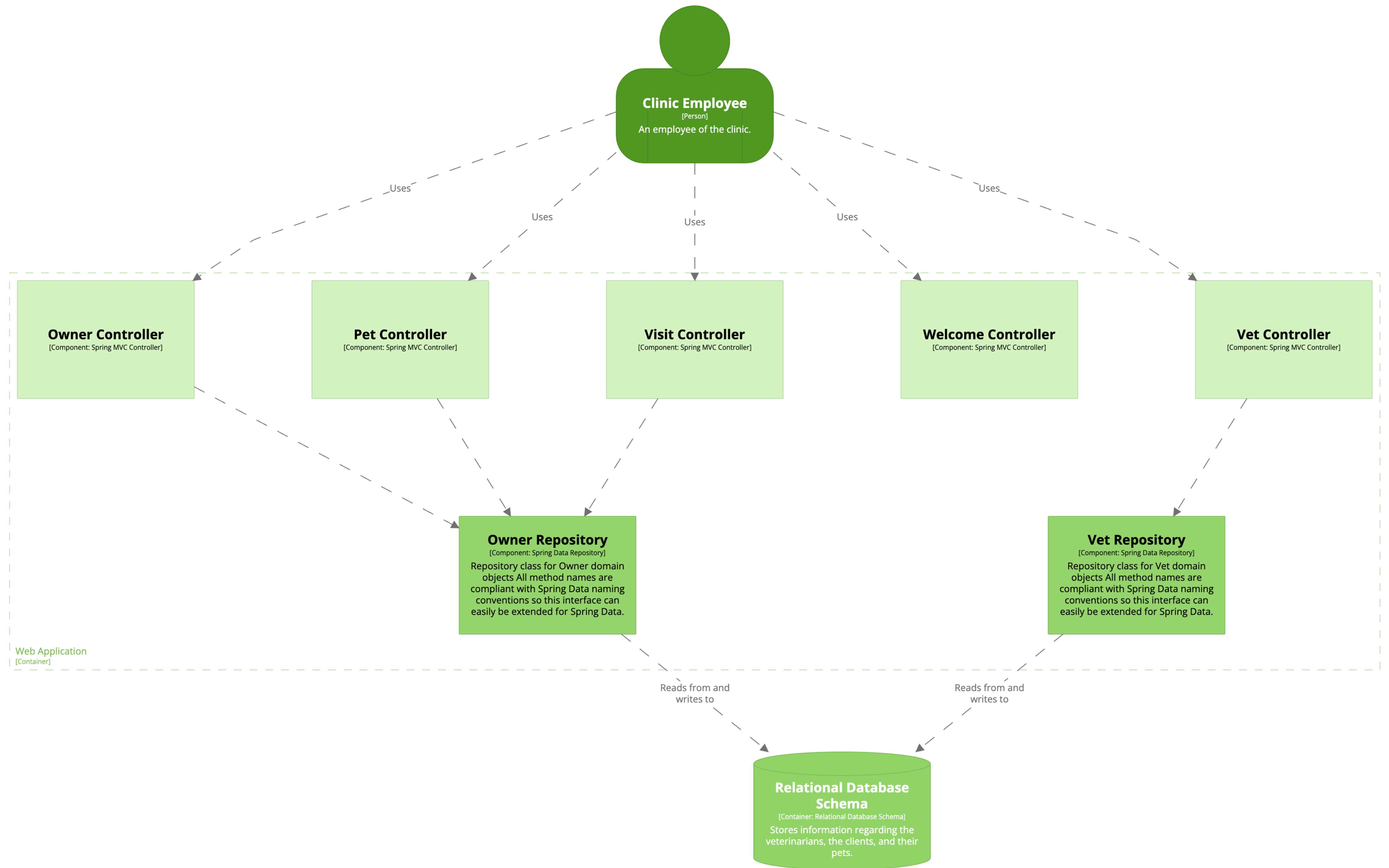
# Component diagrams

Static analysis/reverse-engineering of code as a source of data  
for identifying components and their relationships

```
webApplication = container "Web Application" {
  description "Allows employees to view and manage information regarding the veterinarians, the clients, and their pets."
  technology "Java and Spring"

  !components {
    classes "${SPRING_PETCLINIC_HOME}/target/spring-petclinic-3.3.0-SNAPSHOT.jar"
    source "${SPRING_PETCLINIC_HOME}/src/main/java"
    filter include fqcn-regex "org.springframework.samples.petclinic.*"
    strategy {
      technology "Spring MVC Controller"
      matcher annotation "org.springframework.stereotype.Controller"
      filter exclude fqcn-regex ".*.CrashController"
      url prefix-src "https://github.com/spring-projects/spring-petclinic/blob/main/src/main/java"
      forEach {
        clinicEmployee -> this "Uses"
        tag "Spring MVC Controller"
      }
    }
  }
  strategy {
    technology "Spring Data Repository"
    matcher implements "org.springframework.data.repository.Repository"
    description first-sentence
    url prefix-src "https://github.com/spring-projects/spring-petclinic/blob/main/src/main/java"
    forEach {
      -> relationalDatabaseSchema "Reads from and writes to"
      tag "Spring Data Repository"
    }
  }
}
}
```





# Deployment diagrams

Parse “infrastructure as code” definitions  
(e.g. Terraform, CloudFormation, etc)

or reverse-engineer cloud environment configuration  
as a source of data for identifying deployment elements

A final note...

Level 1	Level 2	Level 3	Level 4	Level 5
<b>Initial</b> No software architecture diagrams.	<b>Ad hoc</b> Software architecture diagrams with ad hoc abstractions and notation, in a general purpose diagramming tool.	<b>Defined</b> Software architecture diagrams with defined abstractions and notation, in a general purpose diagramming tool.	<b>Modelled</b> Software architecture diagrams with defined abstractions and notation, in a modelling tool, authored manually.	<b>Optimising</b> <ul style="list-style-type: none"><li>- Model elements are shared between teams.</li><li>- Centralised system landscape views are generated by aggregating decentralised team-based models.</li><li>- Model elements are reverse-engineered from source code, deployment environment, logs, etc.</li><li>- Alternative visualisations are used for different use cases (e.g. communication vs exploration).</li><li>- Models are used as queryable datasets.</li><li>...</li></ul>

# Software architecture diagramming maturity model

Level 1	Level 2	Level 3	Level 4	Level 5
<b>Initial</b> No software architecture diagrams.	<b>Ad hoc</b> Software architecture diagrams with ad hoc abstractions and notation, in a general purpose diagramming tool.	<b>Defined</b> Software architecture diagrams with defined abstractions and notation, in a general purpose diagramming tool.	<b>Modelled</b> Software architecture diagrams with defined abstractions and notation, in a modelling tool, authored manually.	<b>Optimising</b> <ul style="list-style-type: none"><li>- Model elements are shared between teams.</li><li>- Centralised system landscape views are generated by aggregating decentralised team-based models.</li><li>- Model elements are reverse-engineered from source code, deployment environment, logs, etc.</li><li>- Alternative visualisations are used for different use cases (e.g. communication vs exploration).</li><li>- Models are used as queryable datasets.</li><li>...</li></ul>

# Software architecture diagramming maturity model



Level 1	Level 2	Level 3	Level 4	Level 5
<b>Initial</b> No software architecture diagrams.	<b>Ad hoc</b> Software architecture diagrams with ad hoc abstractions and notation, in a general purpose diagramming tool.	<b>Defined</b> Software architecture diagrams with defined abstractions and notation, in a general purpose diagramming tool.	<b>Modelled</b> Software architecture diagrams with defined abstractions and notation, in a modelling tool, authored manually.	<b>Optimising</b> <ul style="list-style-type: none"><li>- Model elements are shared between teams.</li><li>- Centralised system landscape views are generated by aggregating decentralised team-based models.</li><li>- Model elements are reverse-engineered from source code, deployment environment, logs, etc.</li><li>- Alternative visualisations are used for different use cases (e.g. communication vs exploration).</li><li>- Models are used as queryable datasets.</li><li>...</li></ul>

# Software architecture diagramming maturity model

Level 1	Level 2	Level 3	Level 4	Level 5
<b>Initial</b> No software architecture diagrams.	<b>Ad hoc</b> Software architecture diagrams with ad hoc abstractions and notation, in a general purpose diagramming tool.	<b>Defined</b> Software architecture diagrams with defined abstractions and notation, in a general purpose diagramming tool.	<b>Modelled</b> Software architecture diagrams with defined abstractions and notation, in a modelling tool, authored manually.	<b>Optimising</b> <ul style="list-style-type: none"><li>- Model elements are shared between teams.</li><li>- Centralised system landscape views are generated by aggregating decentralised team-based models.</li><li>- Model elements are reverse-engineered from source code, deployment environment, logs, etc.</li><li>- Alternative visualisations are used for different use cases (e.g. communication vs exploration).</li><li>- Models are used as queryable datasets.</li><li>...</li></ul>

# Software architecture diagramming maturity model

Level 1	Level 2	Level 3	Level 4	Level 5
<b>Initial</b> No software architecture diagrams.	<b>Ad hoc</b> Software architecture diagrams with ad hoc abstractions and notation, in a general purpose diagramming tool.	<b>Defined</b> Software architecture diagrams with defined abstractions and notation, in a general purpose diagramming tool.	<b>Modelled</b> Software architecture diagrams with defined abstractions and notation, in a modelling tool, authored manually.	<b>Optimising</b> <ul style="list-style-type: none"><li>- Model elements are shared between teams.</li><li>- Centralised system landscape views are generated by aggregating decentralised team-based models.</li><li>- Model elements are reverse-engineered from source code, deployment environment, logs, etc.</li><li>- Alternative visualisations are used for different use cases (e.g. communication vs exploration).</li><li>- Models are used as queryable datasets.</li><li>...</li></ul>

# Software architecture diagramming maturity model

# The C4 model is...

A set of hierarchical  
abstractions

(software systems, containers,  
components, and code)

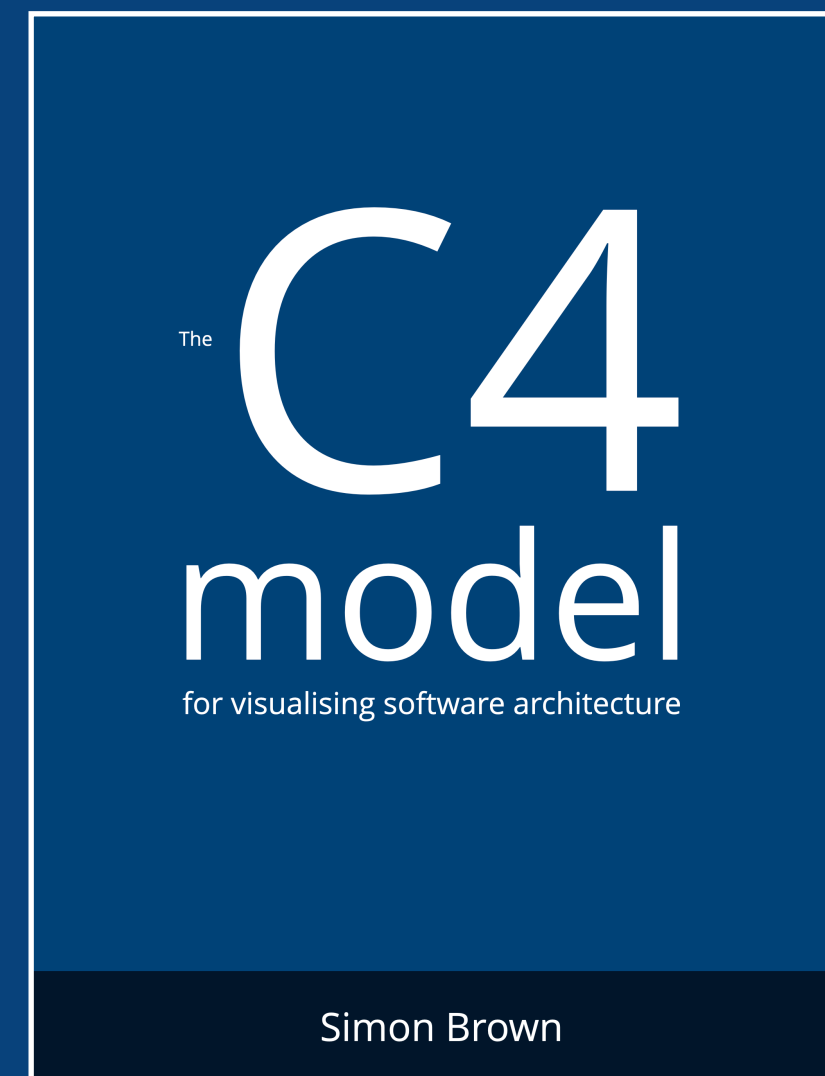
A set of hierarchical  
diagrams

(system context, containers, components,  
and code)

Notation independent

Tooling independent

# Thank you!



<https://leanpub.com/b/software-architecture>

Simon Brown