The C4 model

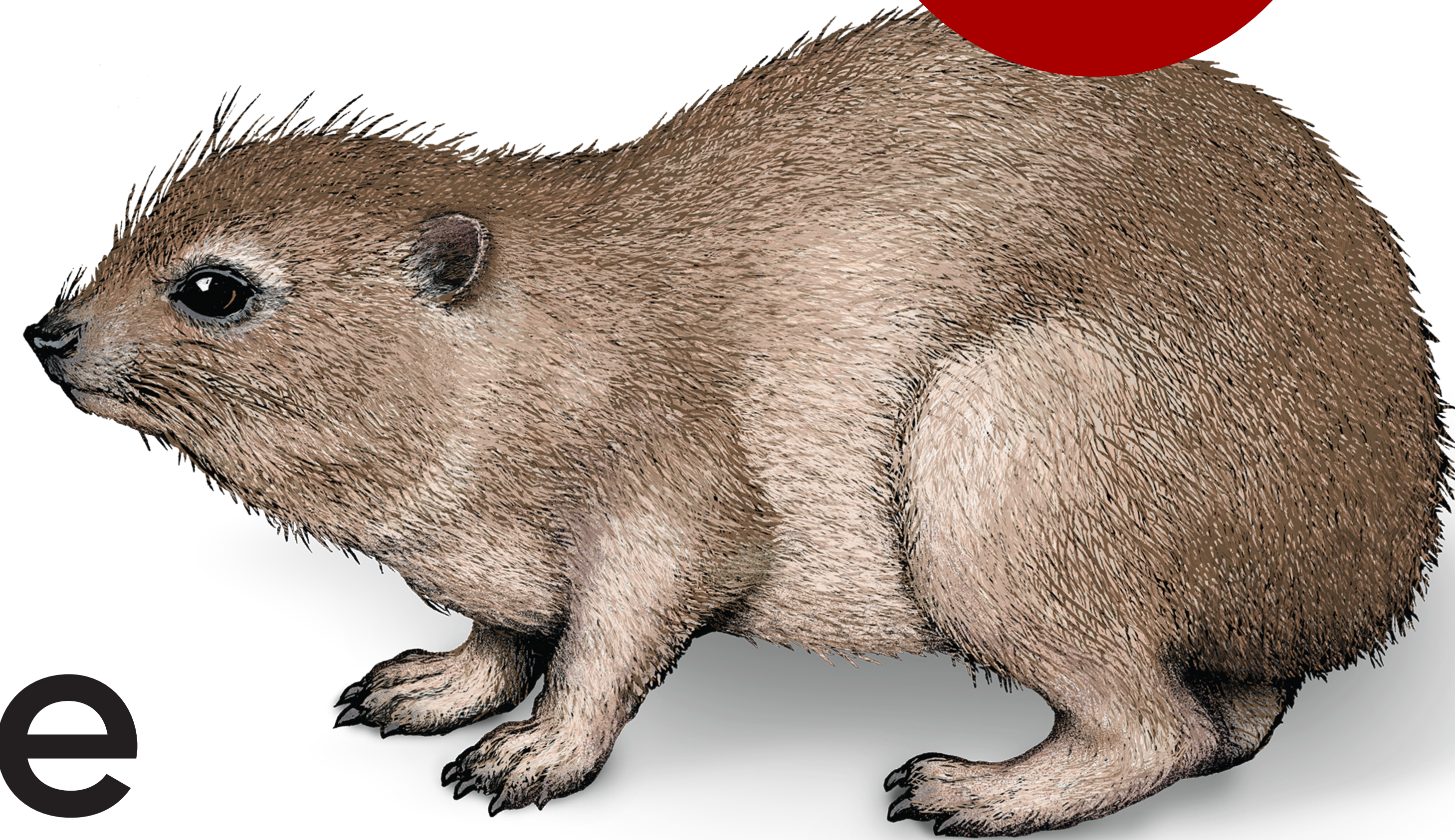
Beyond the basics

Simon Brown

O'REILLY

Early Release

RAW & UNEDITED



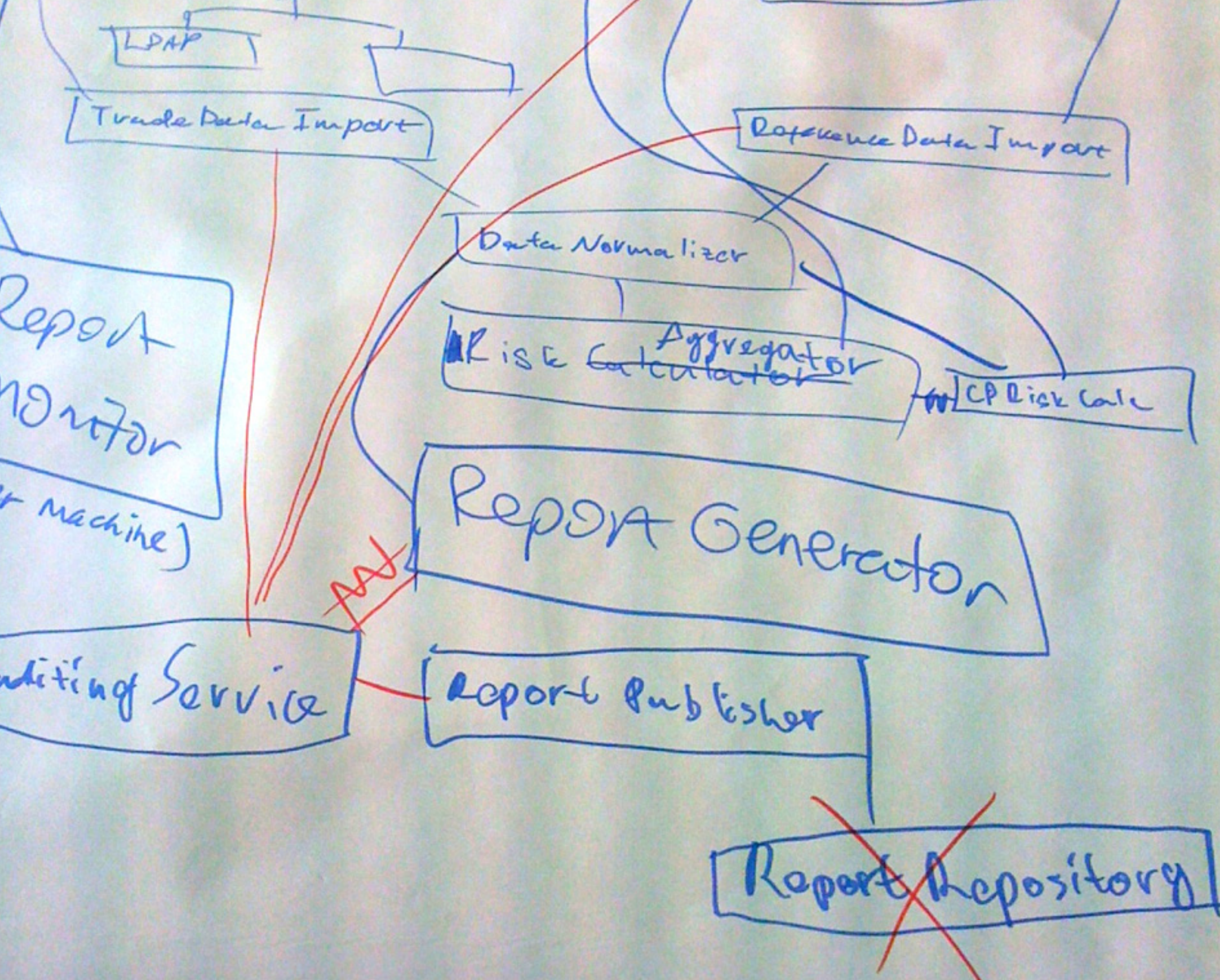
The C4 Model

Visualizing Software Architecture

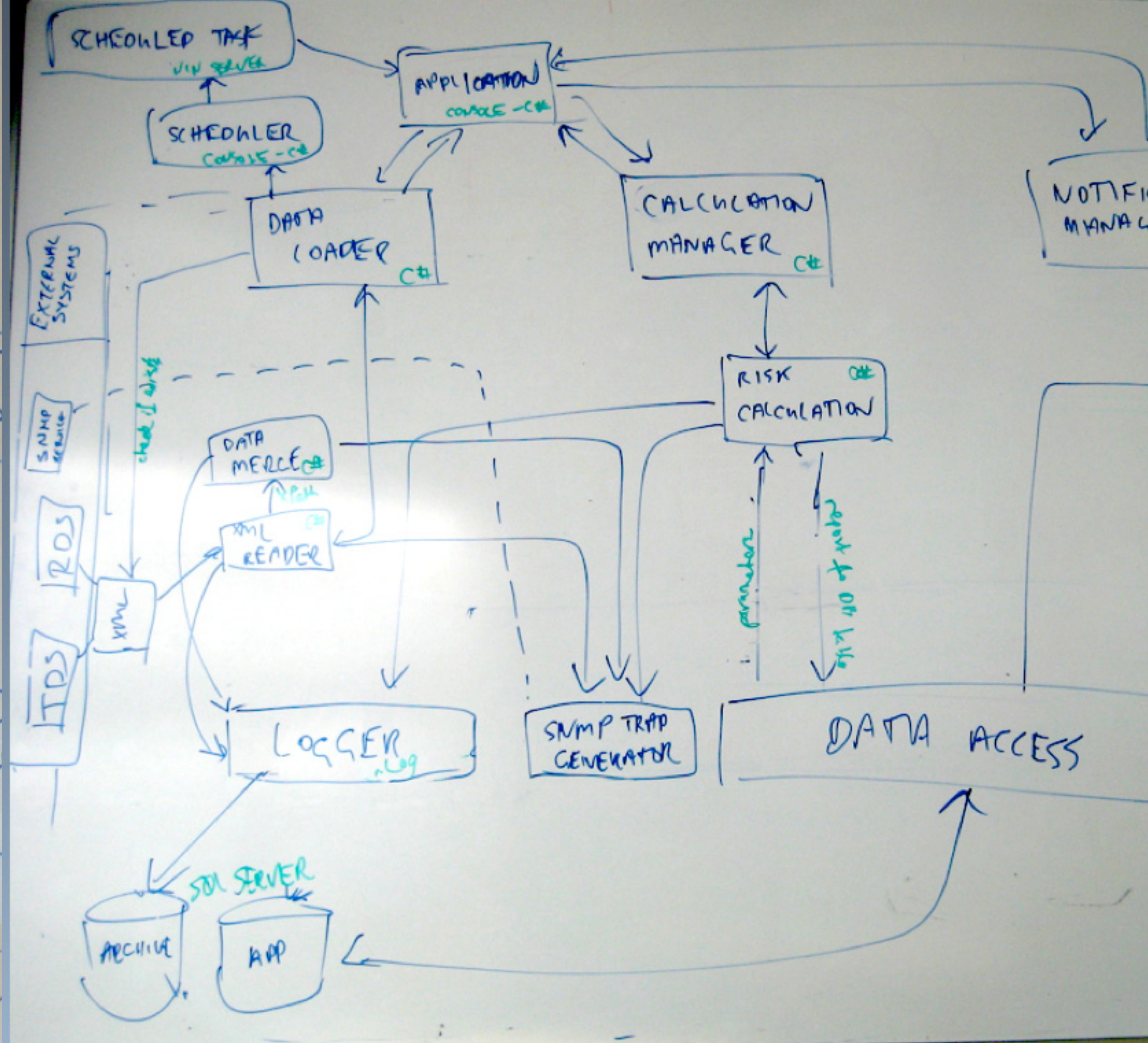
Simon Brown

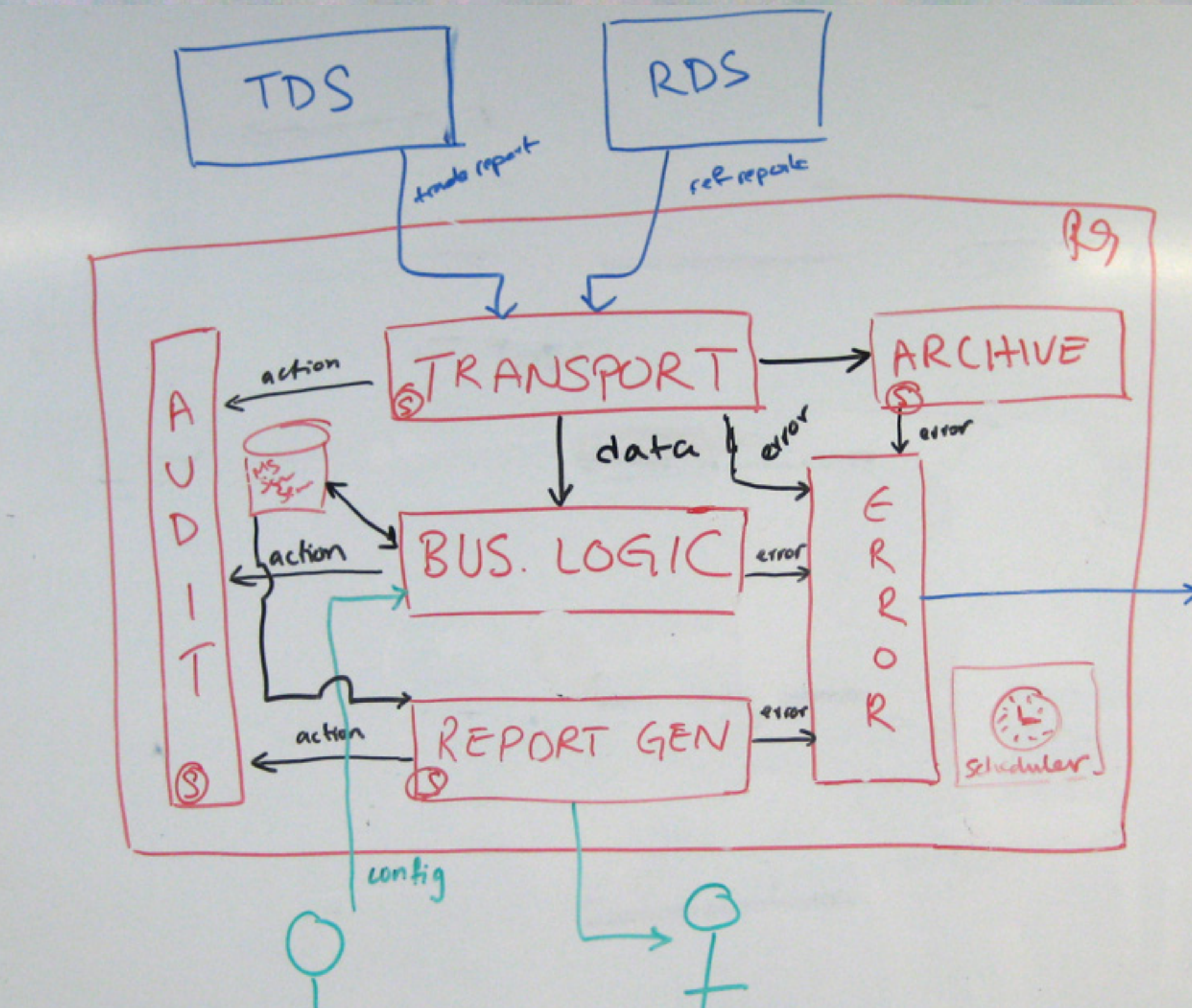
Scheduled for release July 2026,
early access available now
via the O'Reilly platform

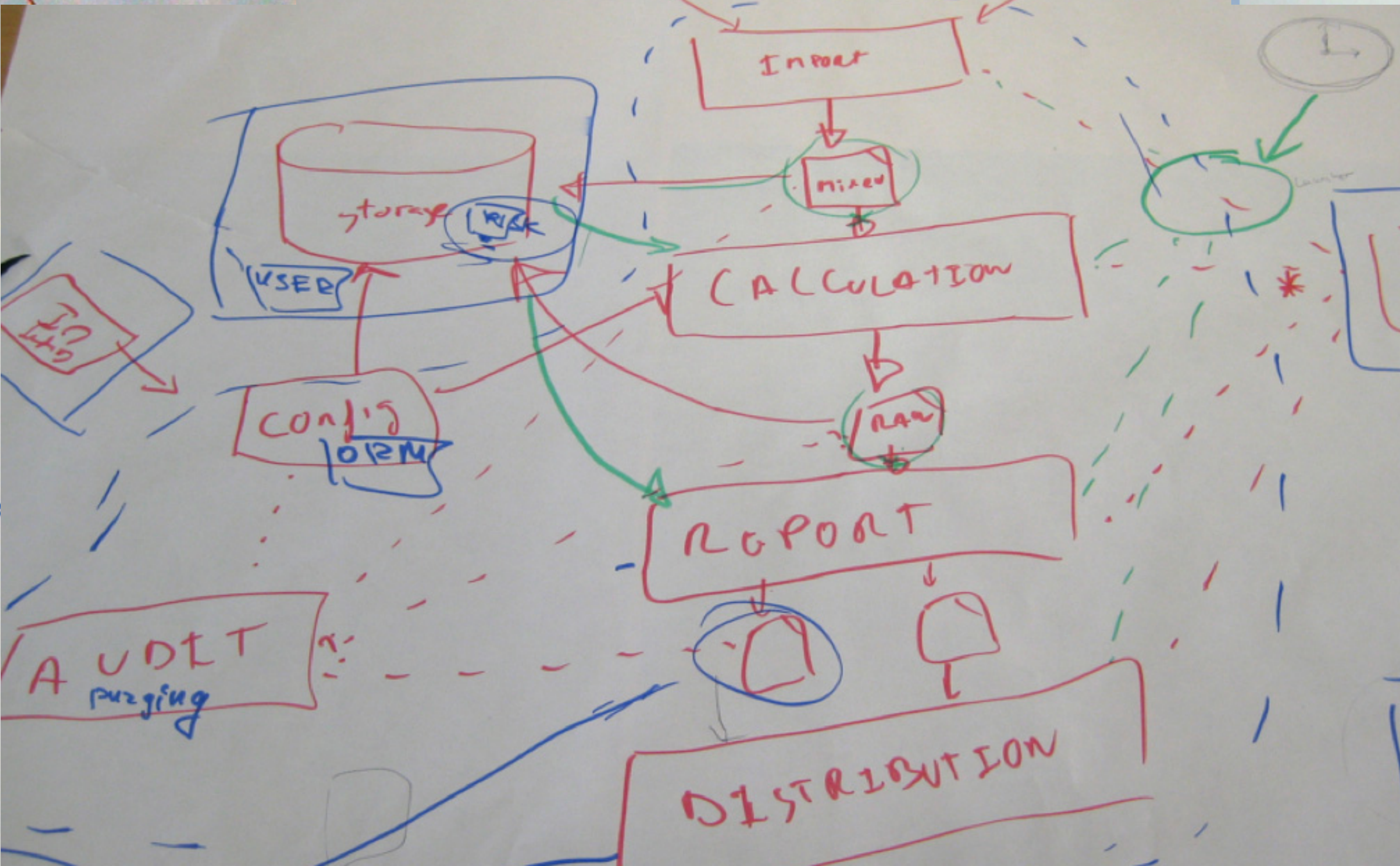
The C4 model?

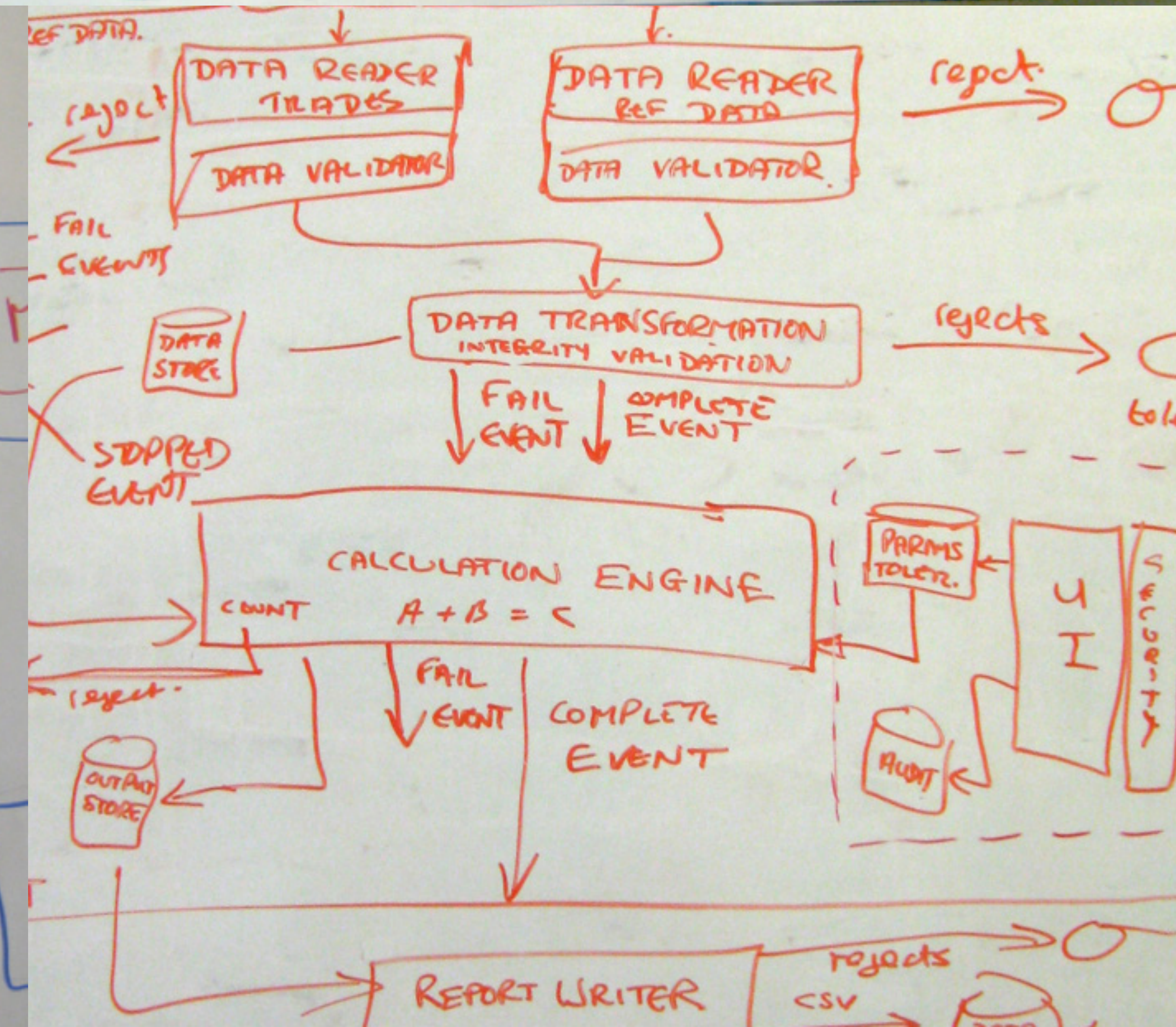

LDAP
Trade Data Import
Reference Data Import
Data Normalizer
Risk Calculator
Aggregator
CP Risk Calc
Report Generator
Report Publisher
Report Repository

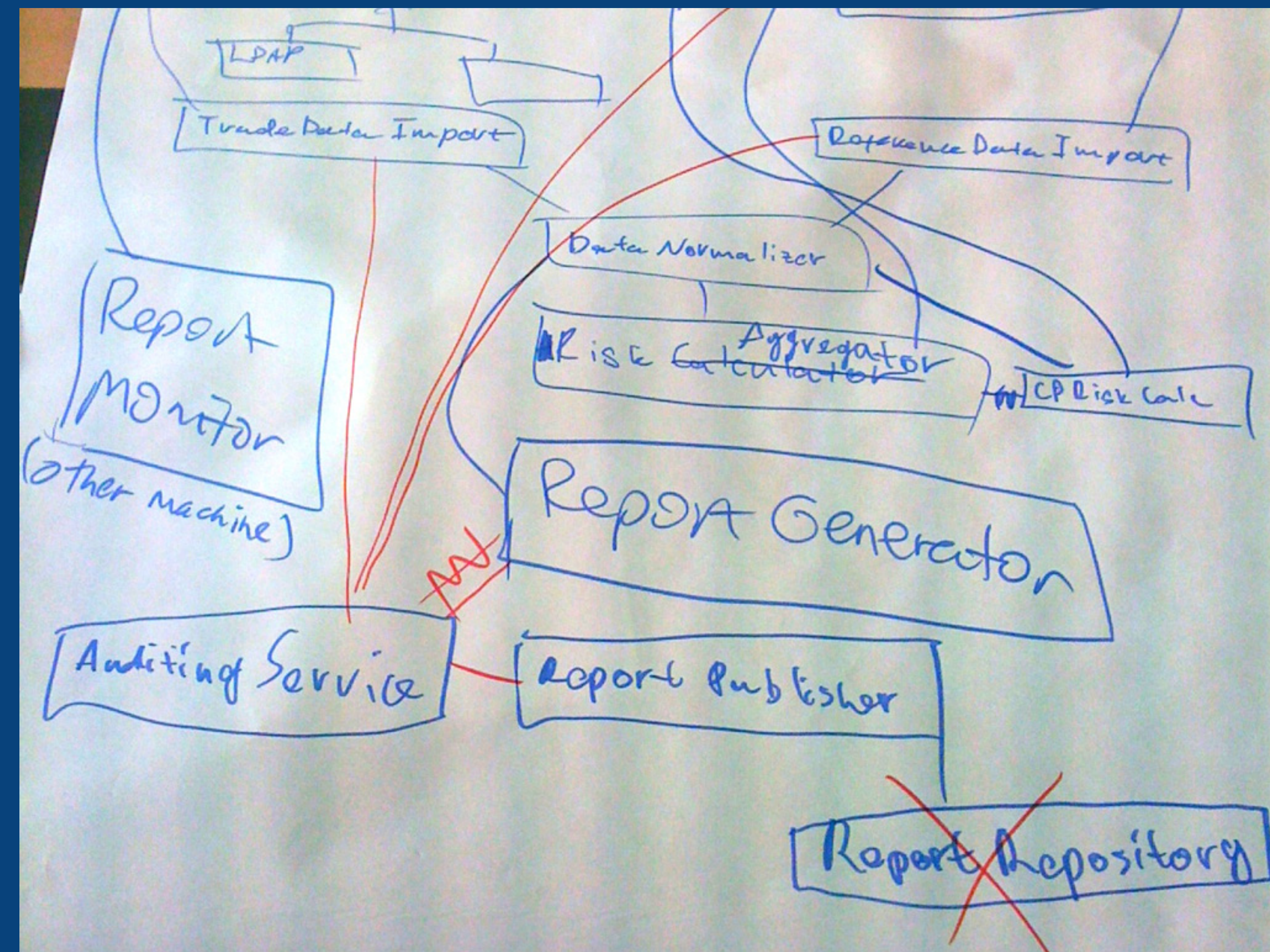
ASP NET
PARAMETER MANAGER
RISK CALCULATION
AUDITING
RDS
RISK DATA
PARAMS

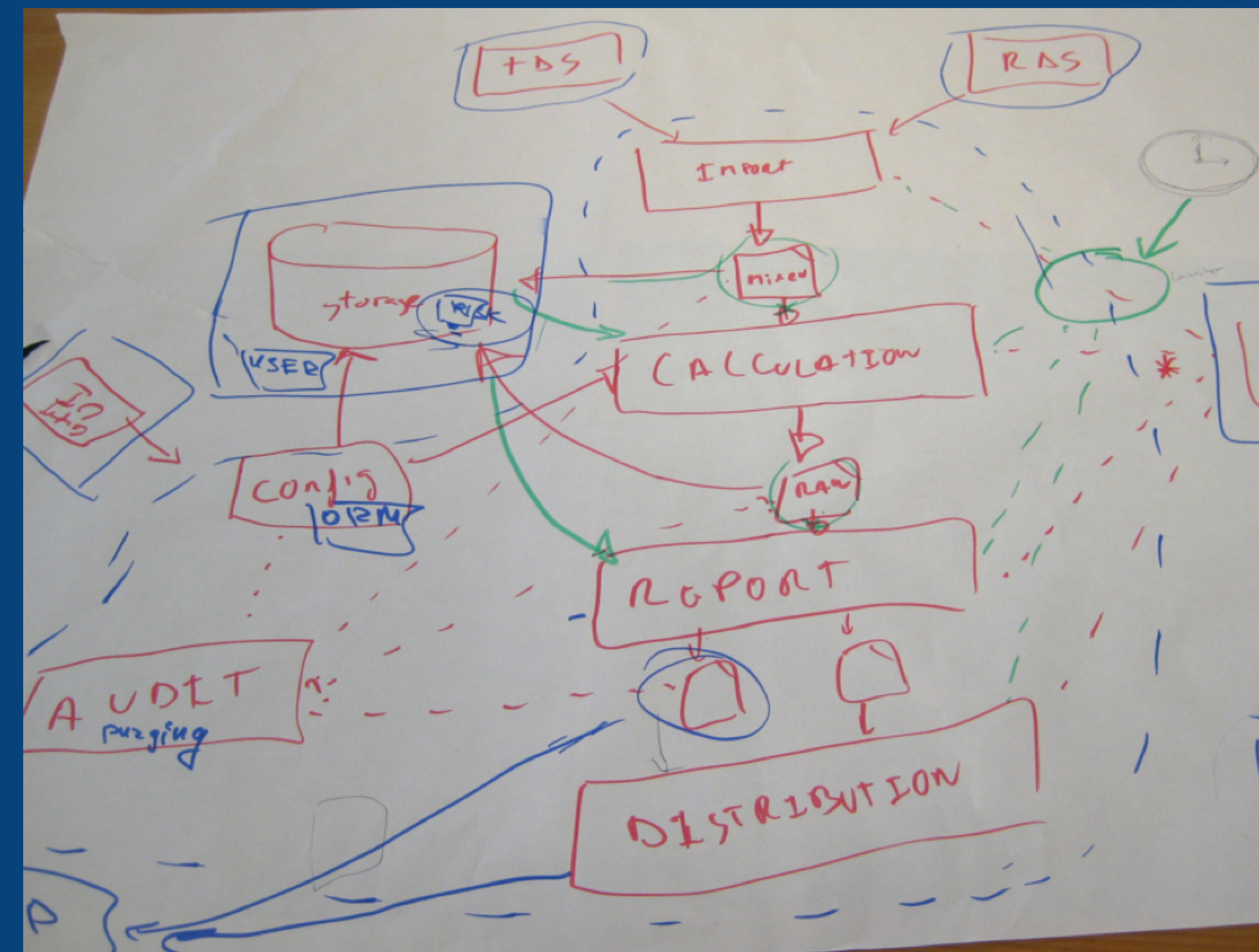

SCHEDULED TASK
APPLICATION
SCHEDULER
DATA LOADER
CALCULATION MANAGER
EXTERNAL SYSTEMS
RISK CALCULATION
DATA MERGE
XML READER
LOGGER
SNMP TRAP GENERATOR
DATA ACCESS
SQL SERVER
ARCHIVE
APP

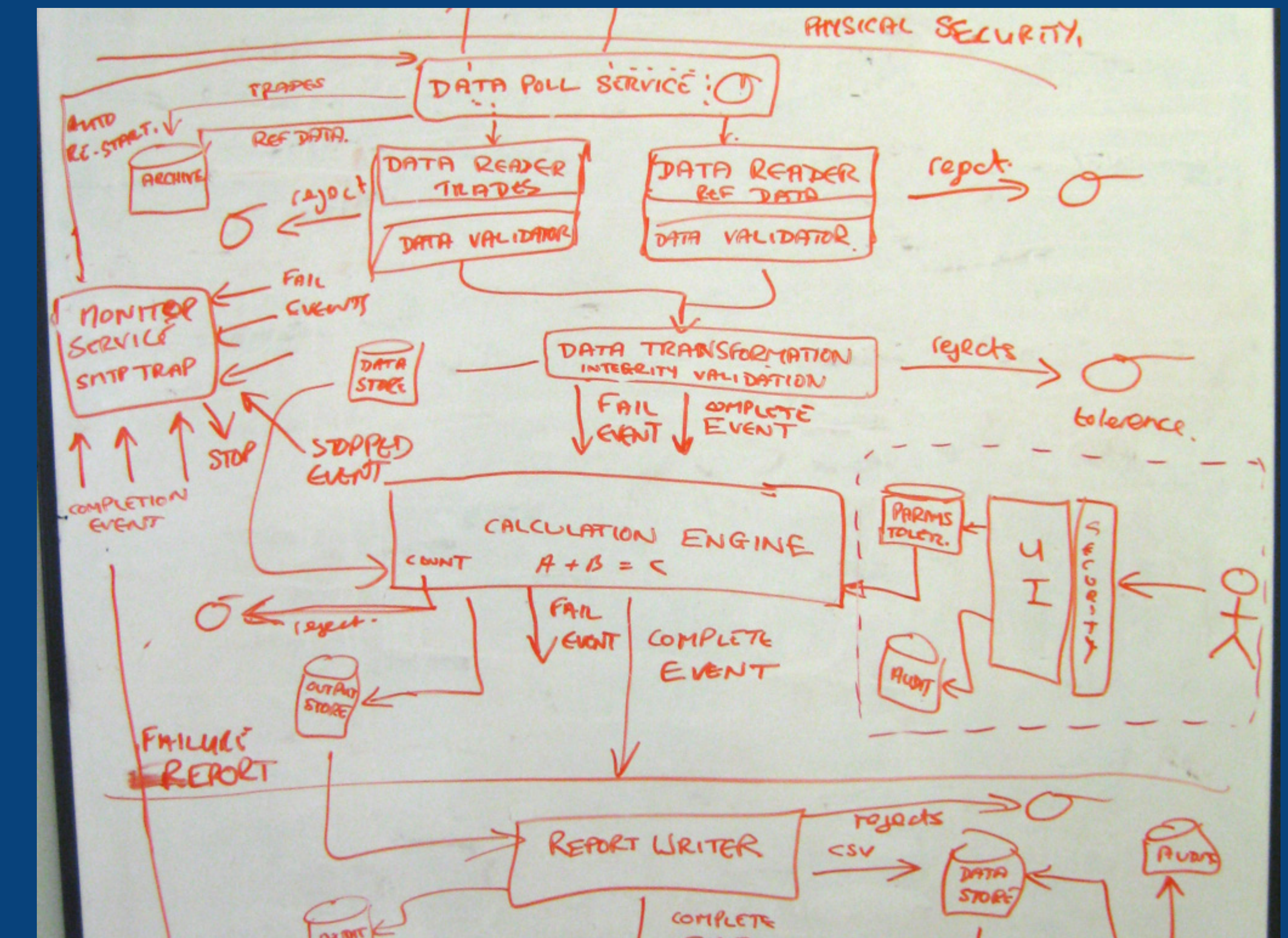

TDS
RDS
TRANSPORT
ARCHIVE
AUDIT
action
data
error
BUS. LOGIC
REPORT GEN
ERROR
Scheduler
config


TDS
RDS
Import
CALCULATION
USER
config
ORM
REPORT
AUDIT
purging
DISTRIBUTION


DATA READER TRADES
DATA VALIDATOR
DATA READER REF DATA
DATA VALIDATOR
reject
FAIL EVENTS
DATA STORE
DATA TRANSFORMATION
INTEGRITY VALIDATION
rejects
FAIL EVENT
COMPLETE EVENT
STOPPED EVENT
CALCULATION ENGINE
A + B = C
FAIL EVENT
COMPLETE EVENT
UI
REPORT WRITER
rejects
csv





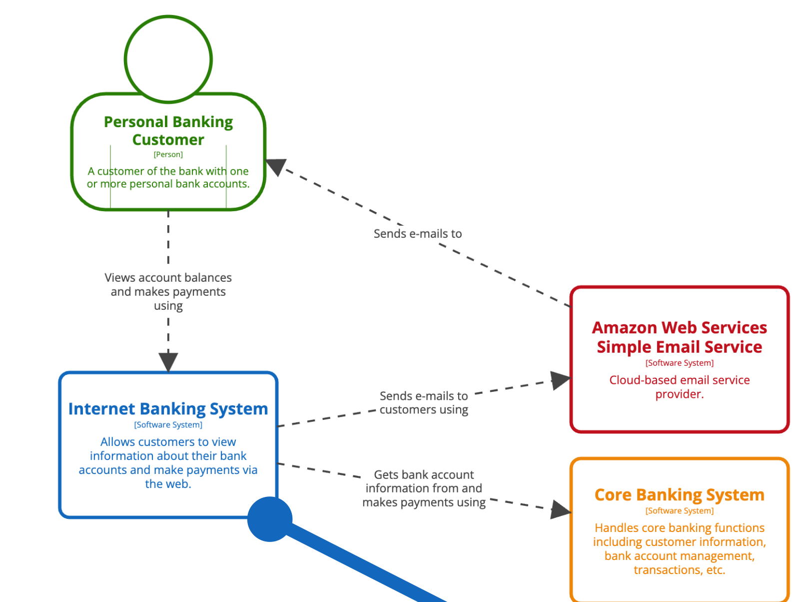


C4 is a way to introduce **structure** to “boxes & arrows” diagrams

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).

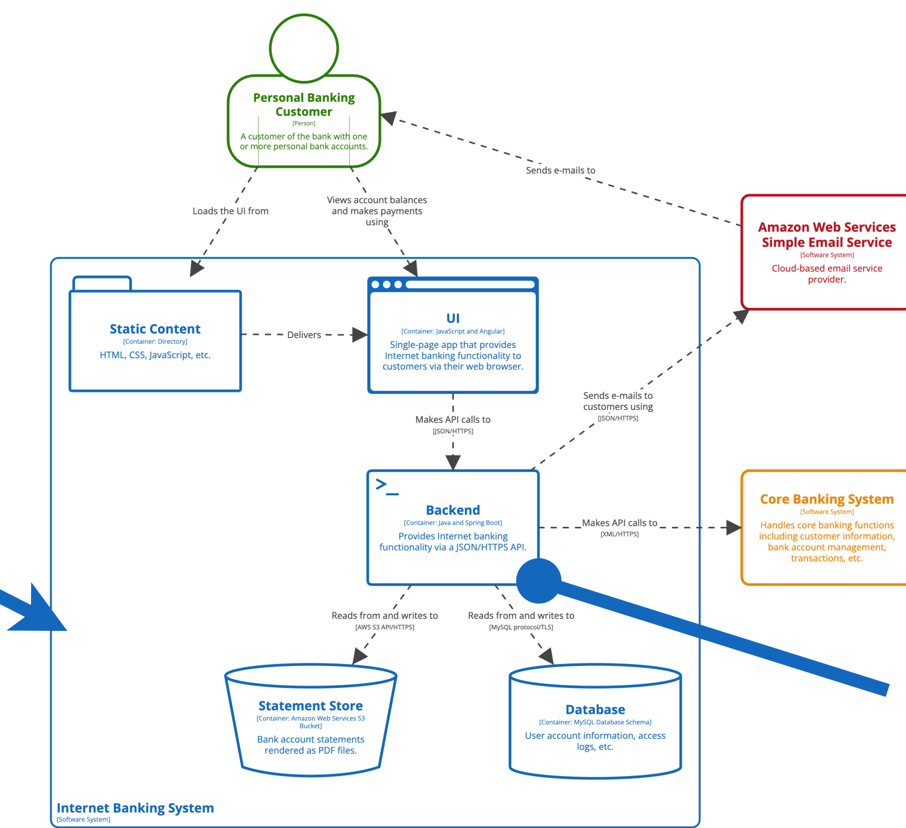
C4

c4model.com



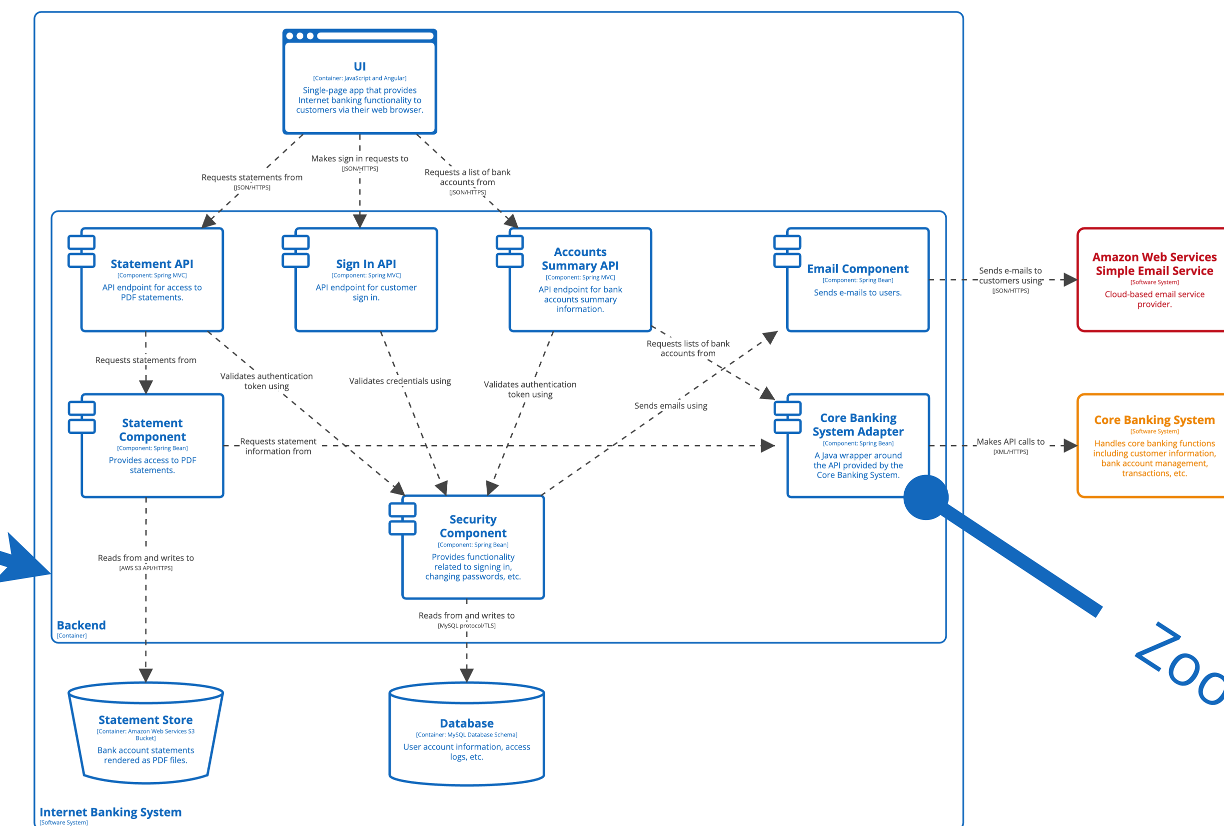
System Context View: Internet Banking System
The system context diagram for a fictional Internet Banking System

Zoom in



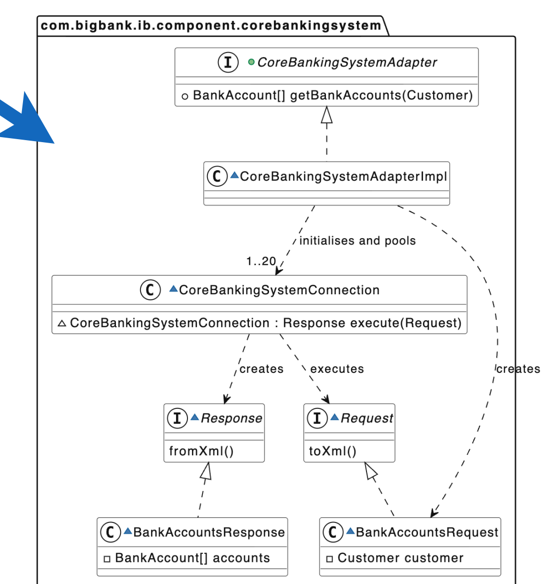
Container View: Internet Banking System

Zoom in



Component View: Internet Banking System - Backend

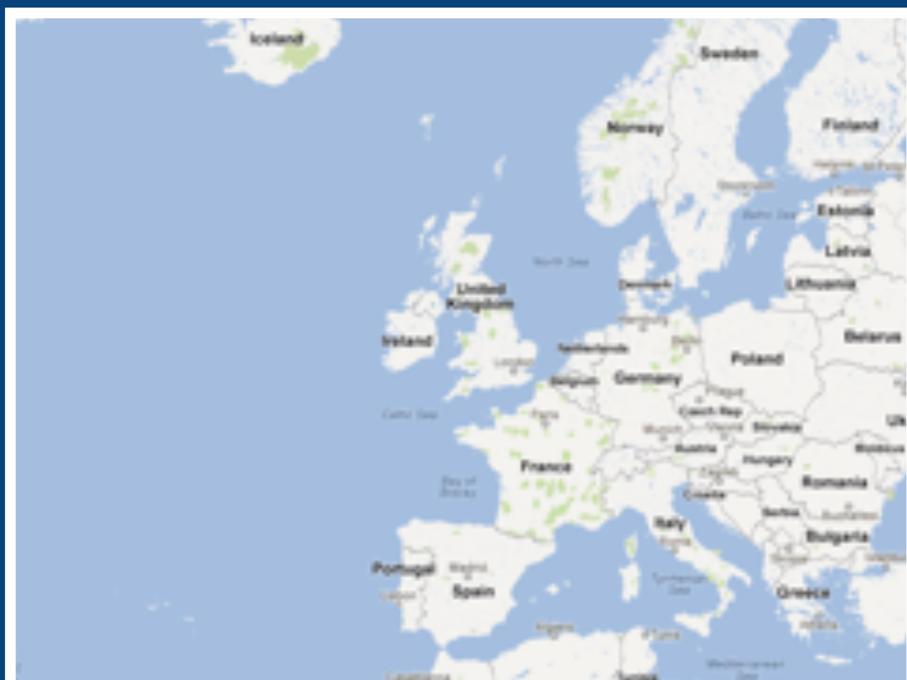
Zoom in

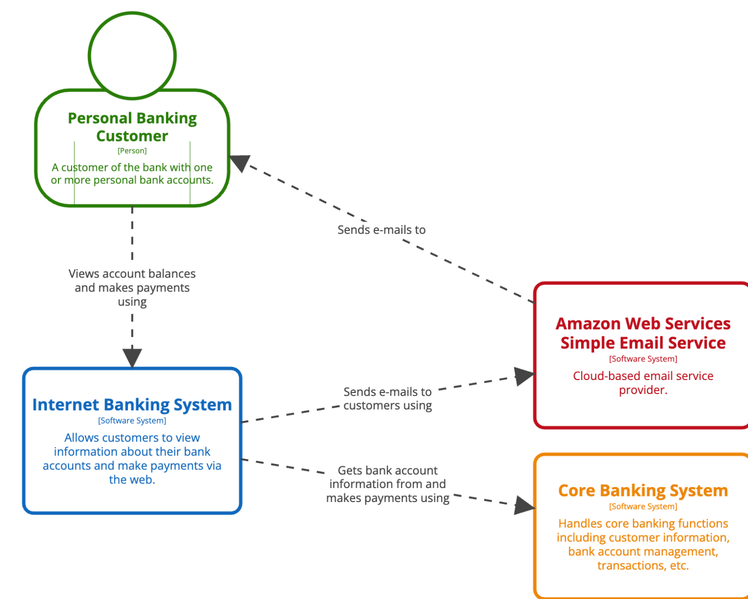


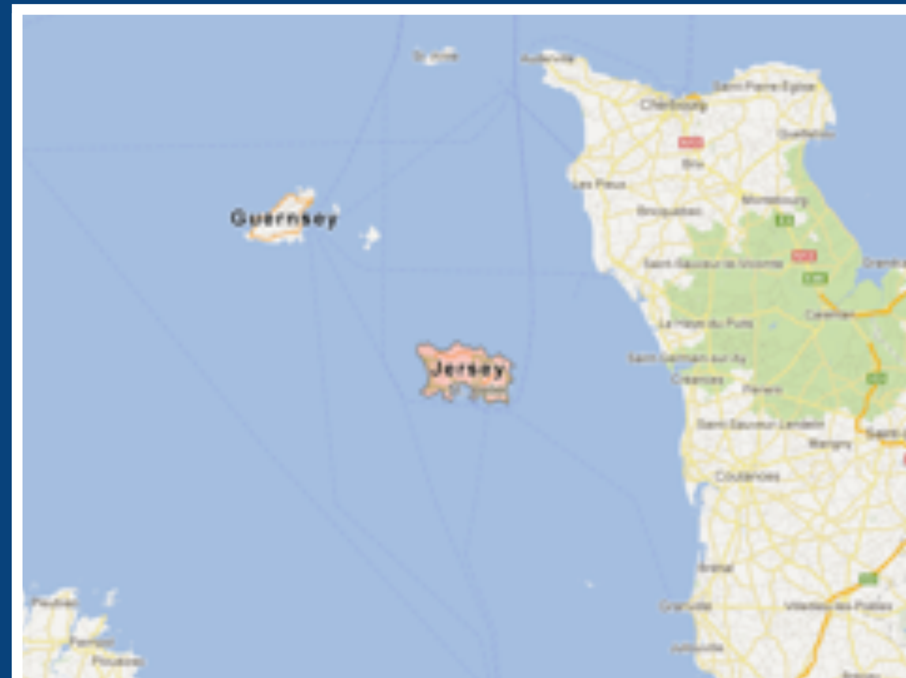
Code View: Internet Banking System - Backend - Core Banking System Adapter
A summary of the implementation details for the Core Banking System Adapter component

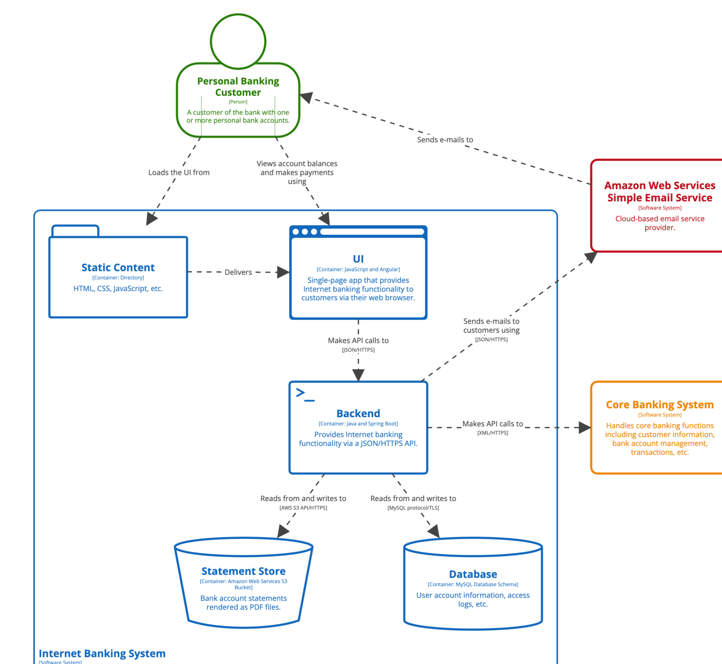
Static structure diagrams

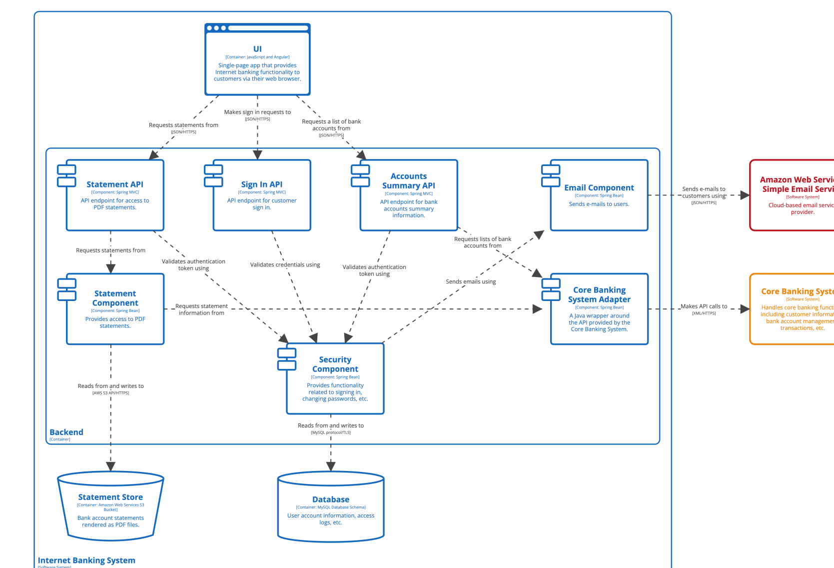
System Context | Containers | Components | Code

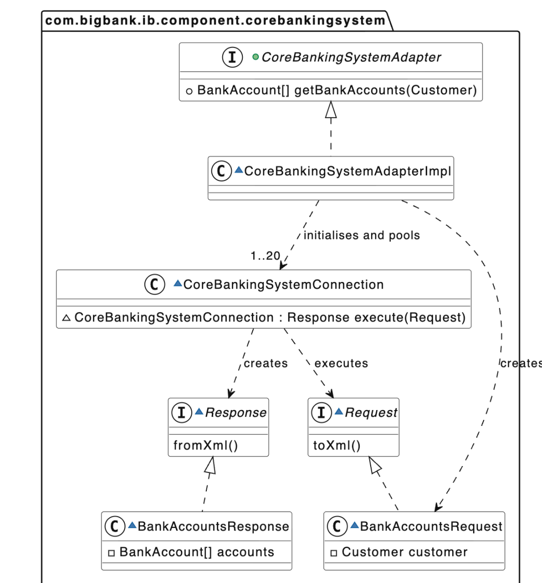








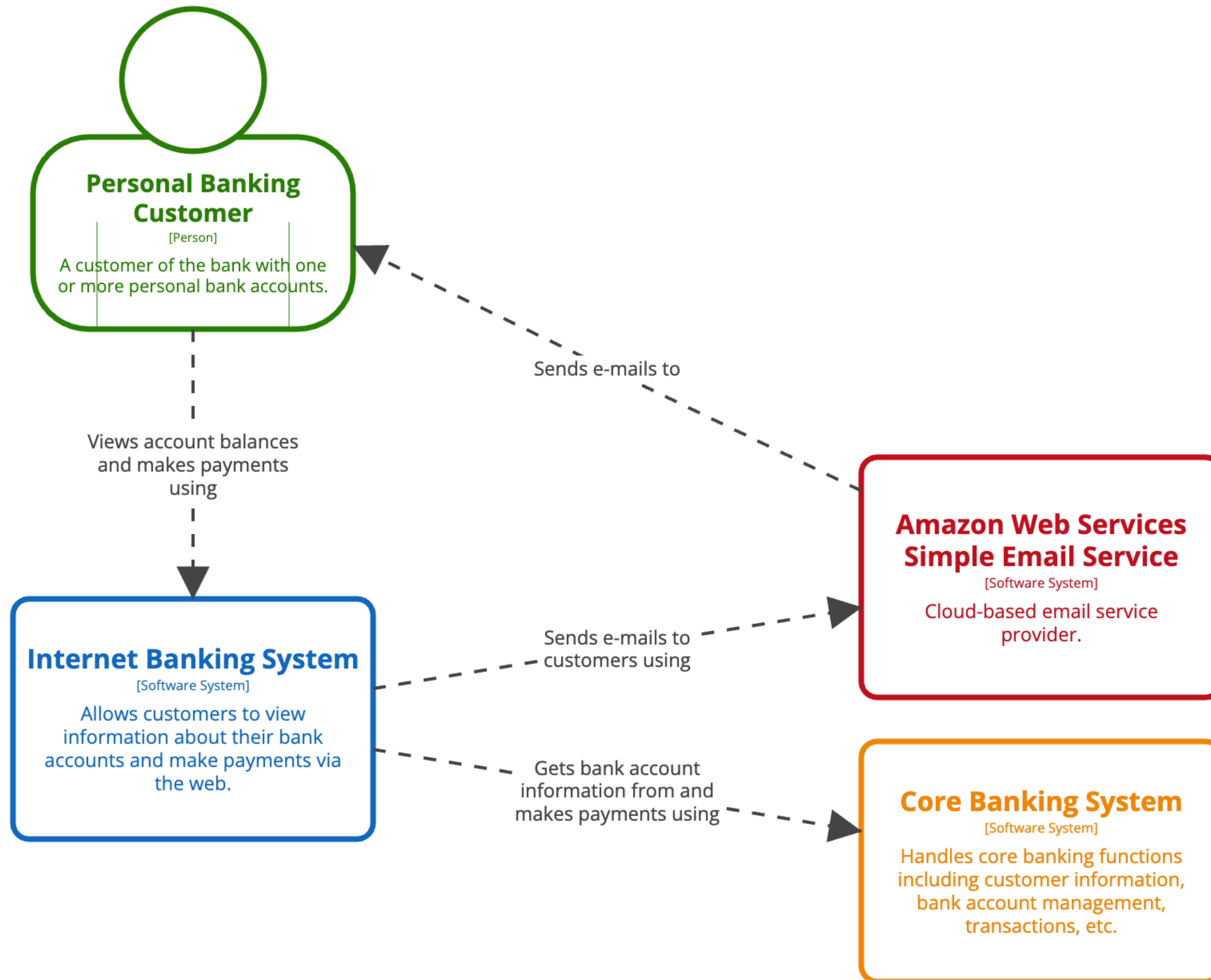




Diagrams are maps

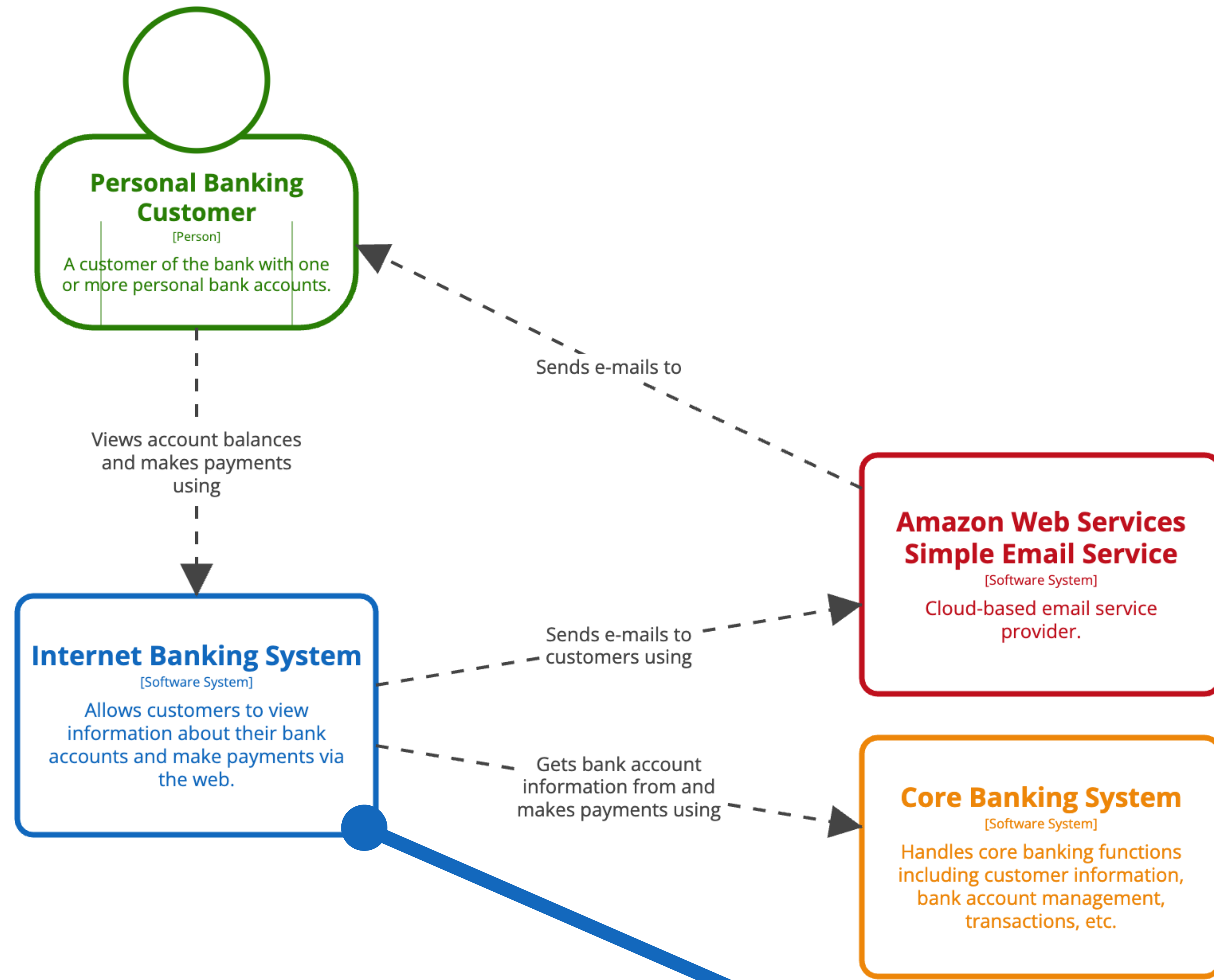
that help software developers navigate a large and/or complex codebase

The C4 model is notation independent



System Context View: Internet Banking System

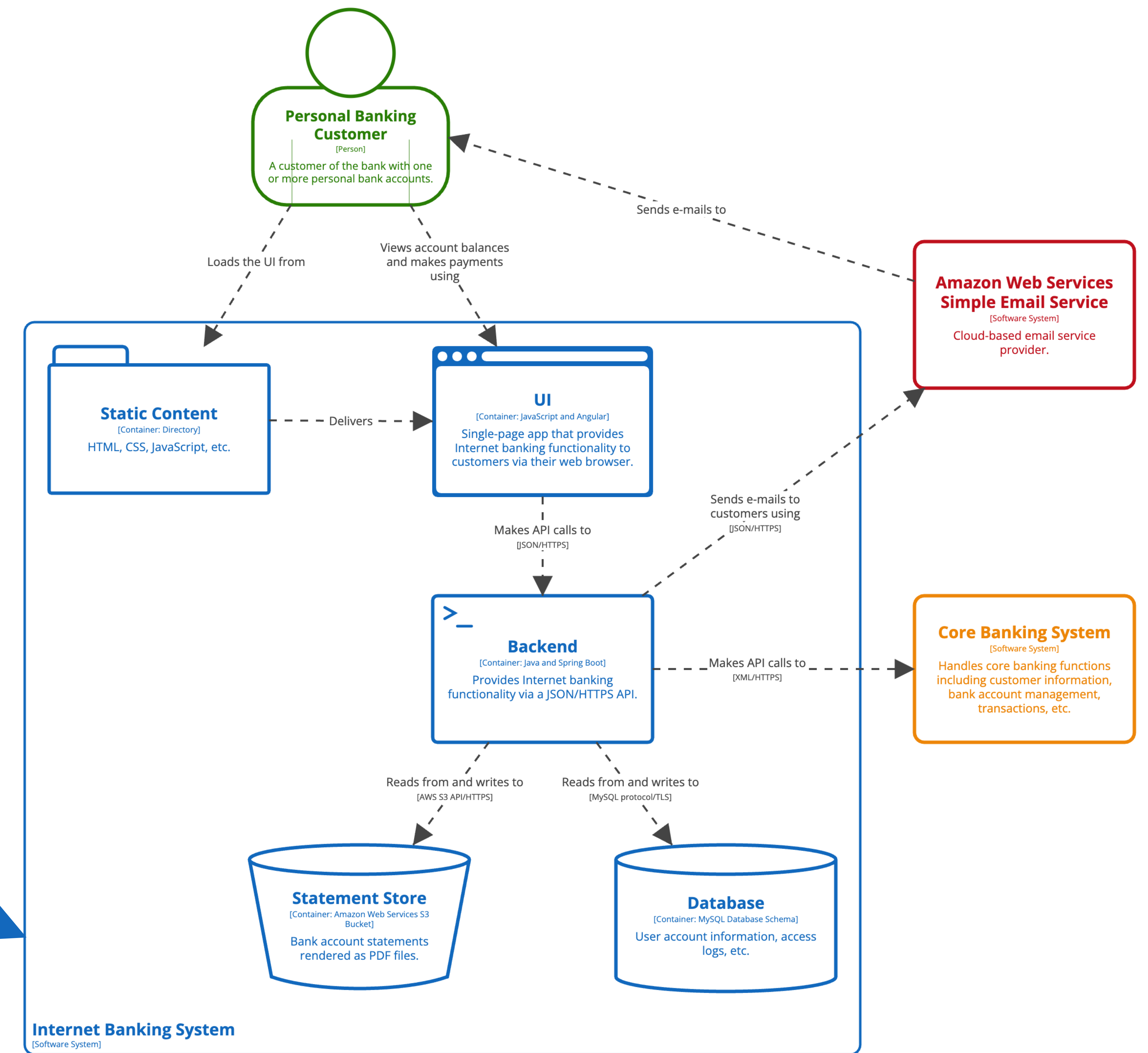
The system context diagram for a fictional Internet Banking System



System Context View: Internet Banking System

The system context diagram for a fictional Internet Banking System

Zoom in to a software system

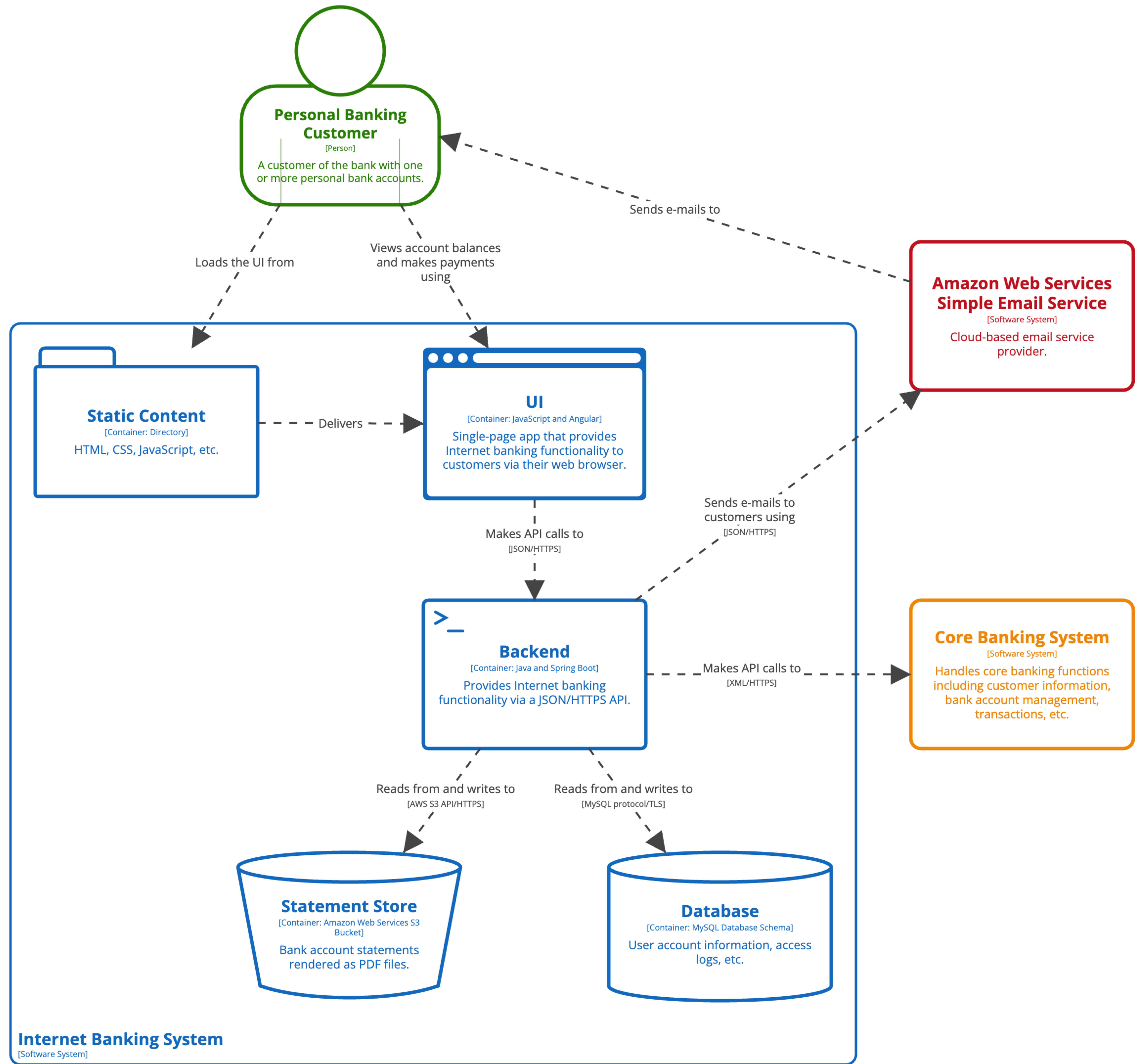


Container View: Internet Banking System

The container diagram for the Internet Banking System

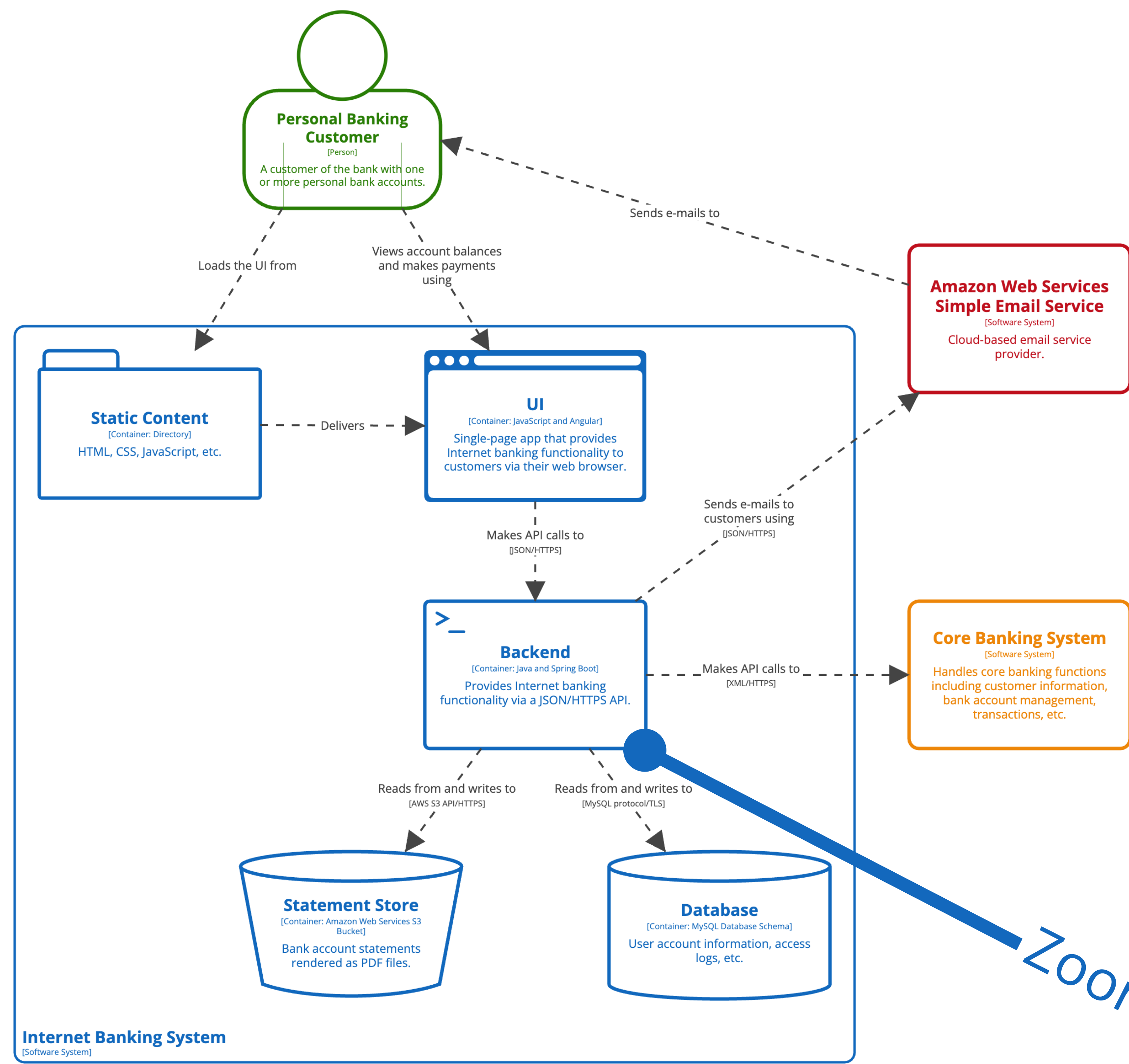
System context diagram

Container diagram



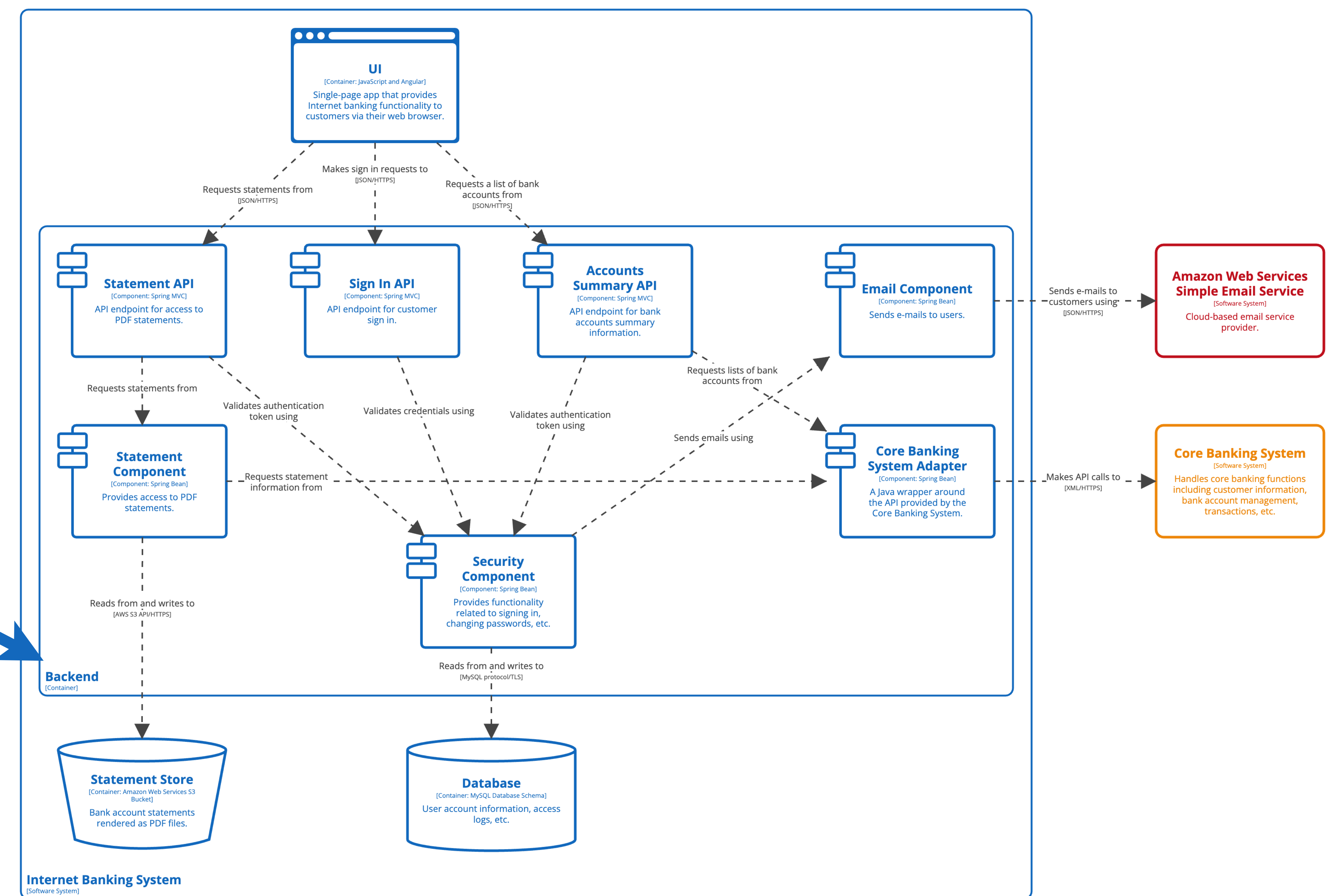
Container View: Internet Banking System

The container diagram for the Internet Banking System



Container View: Internet Banking System
The container diagram for the Internet Banking System

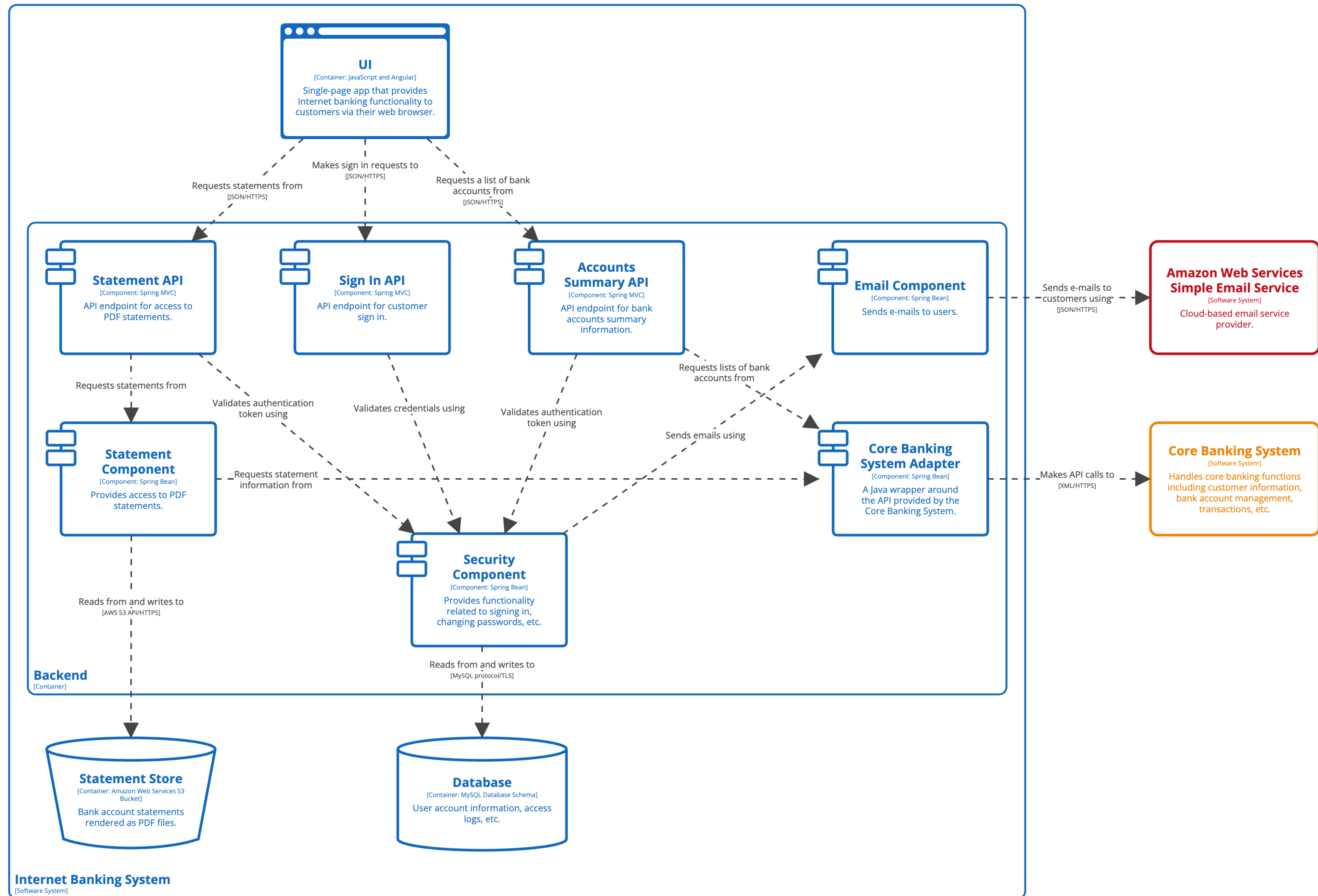
Zoom in to a container



Component View: Internet Banking System - Backend
The component diagram for the Internet Banking System Backend

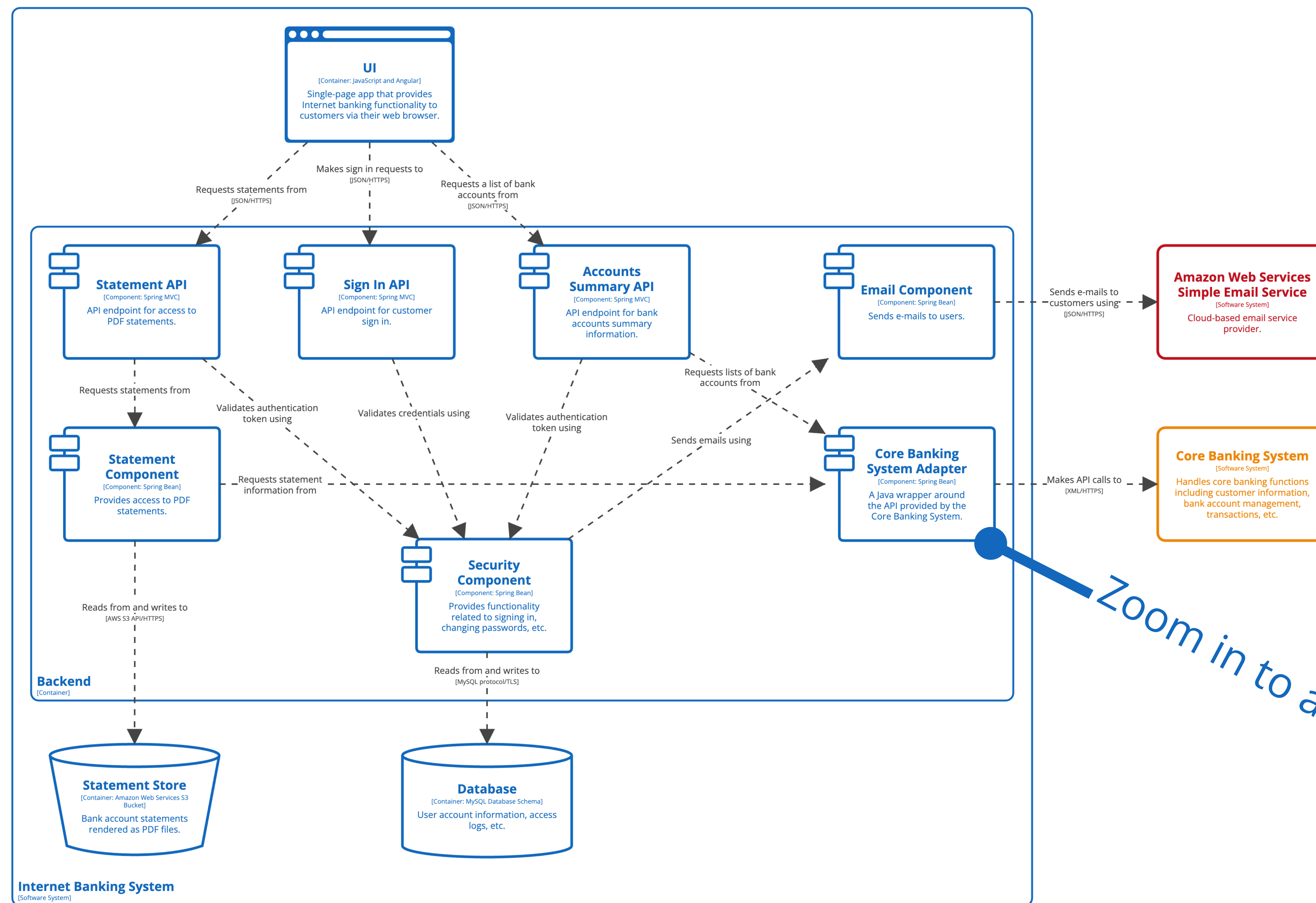
Container diagram

Component diagram



Component View: Internet Banking System - Backend

The component diagram for the Internet Banking System Backend



Component View: Internet Banking System - Backend
The component diagram for the Internet Banking System Backend

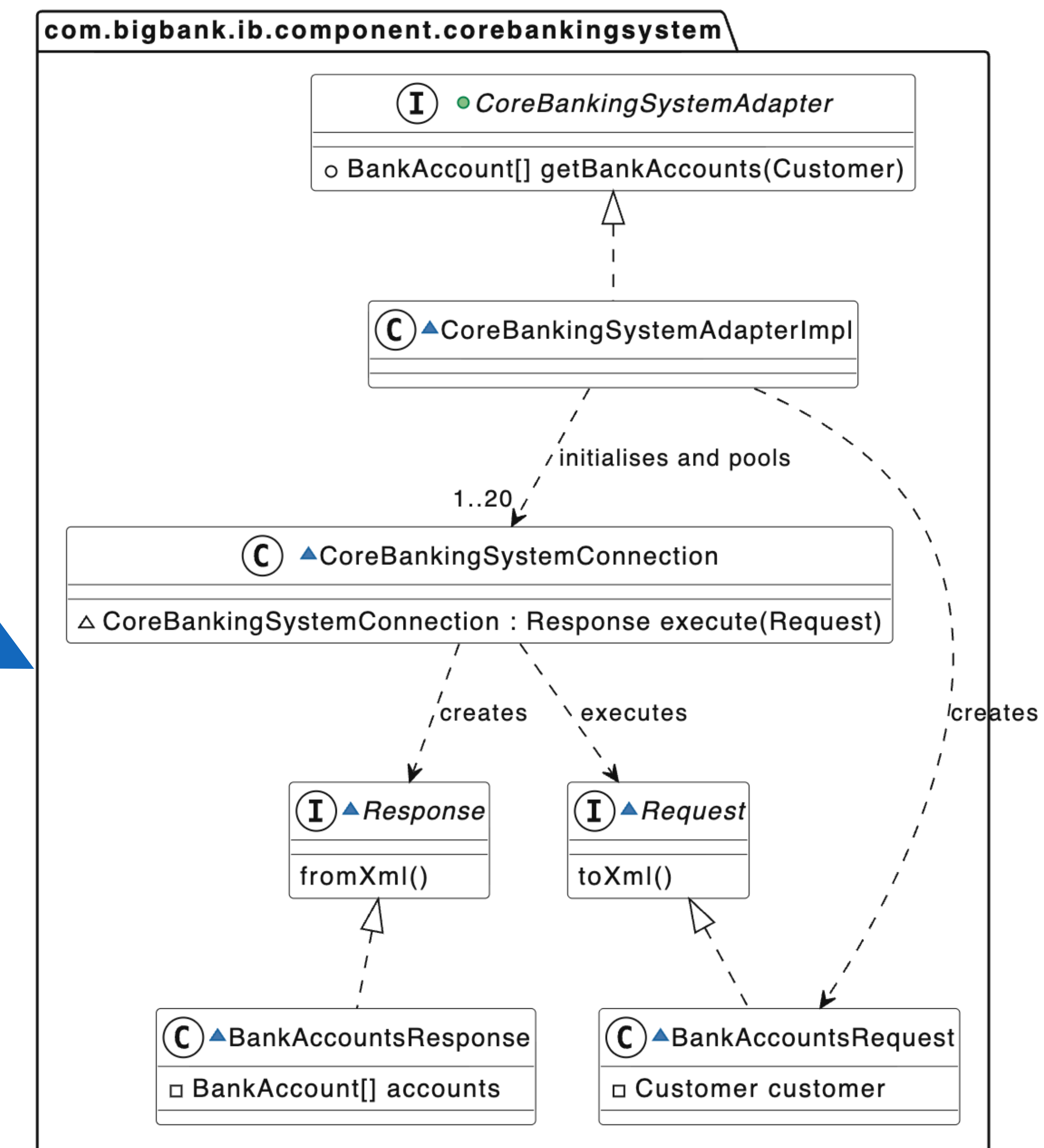
Sends e-mails to customers using

Amazon Web Services Simple Email Service
Cloud-based email service provider.

Makes API calls to

Core Banking System
Handles core banking functions including customer information, bank account management, transactions, etc.

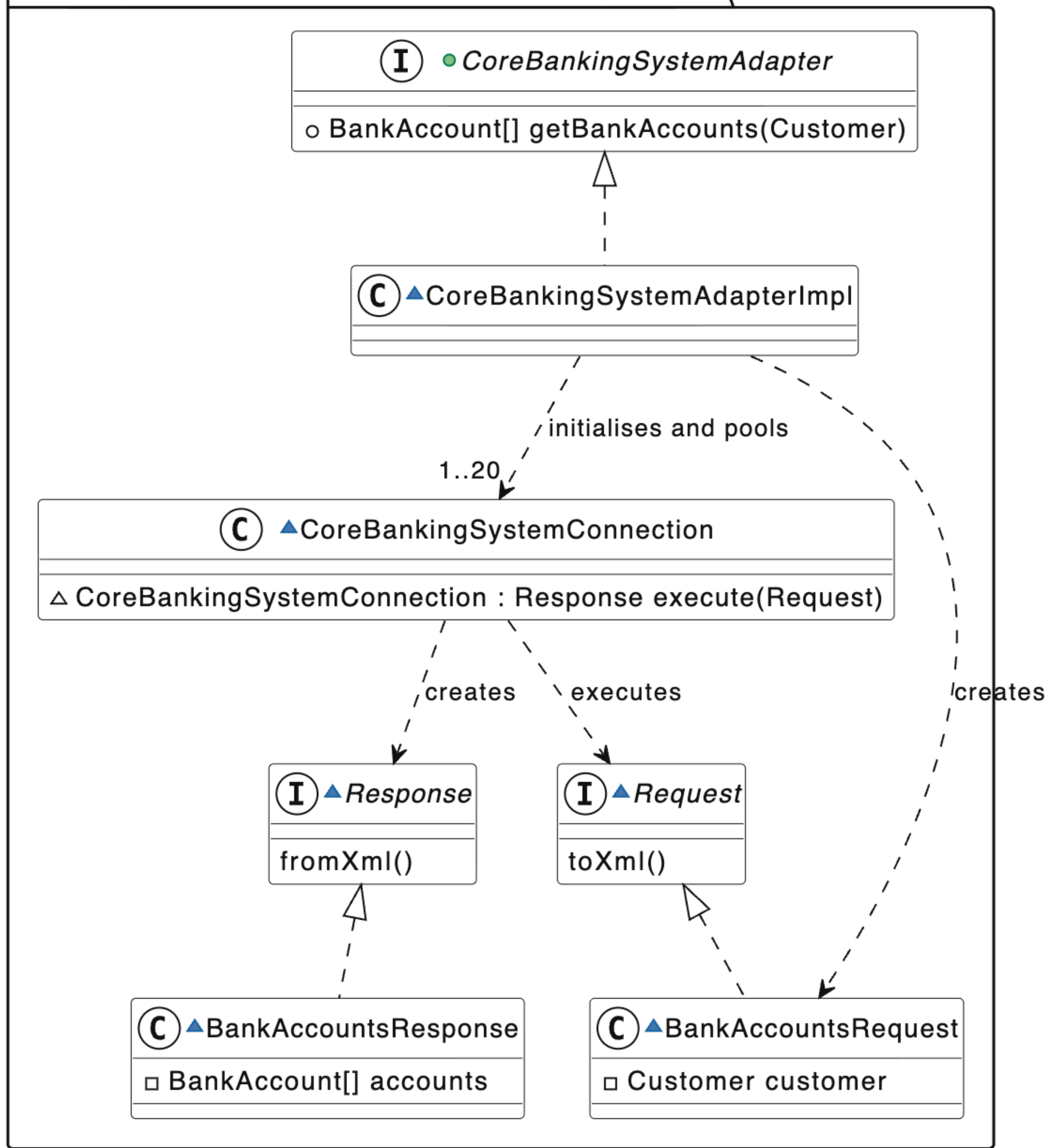
Zoom in to a component



Code View: Internet Banking System - Backend - Core Banking System Adapter
A summary of the implementation details for the Core Banking System Adapter component

Component diagram

Code diagram



Code View: Internet Banking System - Backend - Core Banking System Adapter

A summary of the implementation details for the Core Banking System Adapter component

1. System Context

The system plus users and system dependencies.

2. Containers

The overall shape of the architecture and technology choices.

Recommended for all engineering teams

3. Components

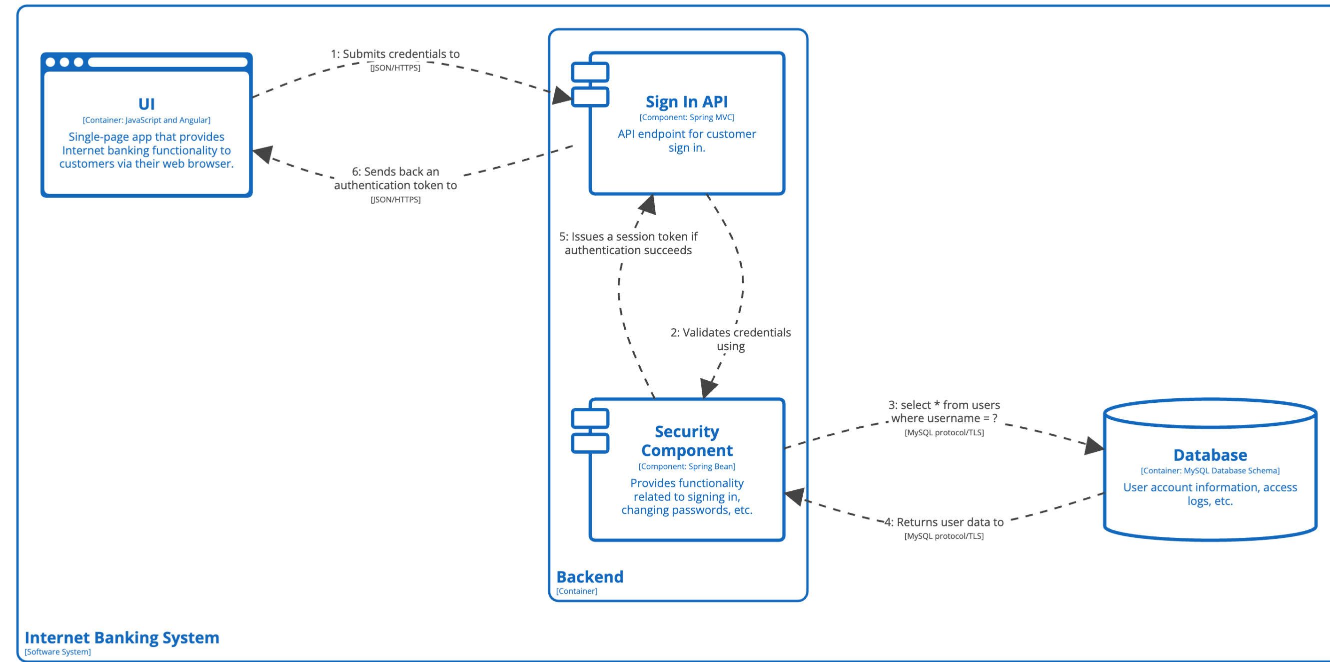
Logical components and their interactions within a container.

Optional

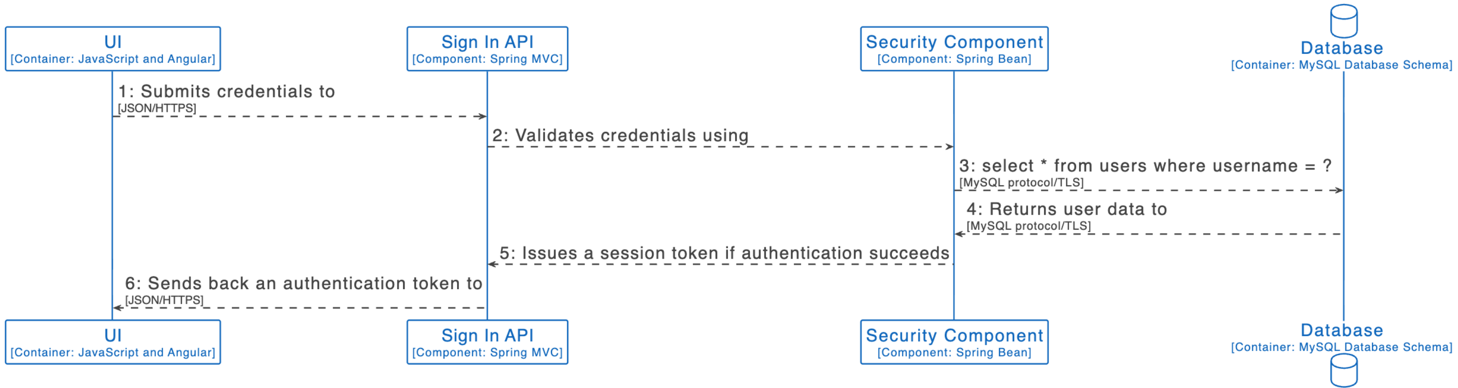
4. Code (e.g. classes)

Component implementation details.

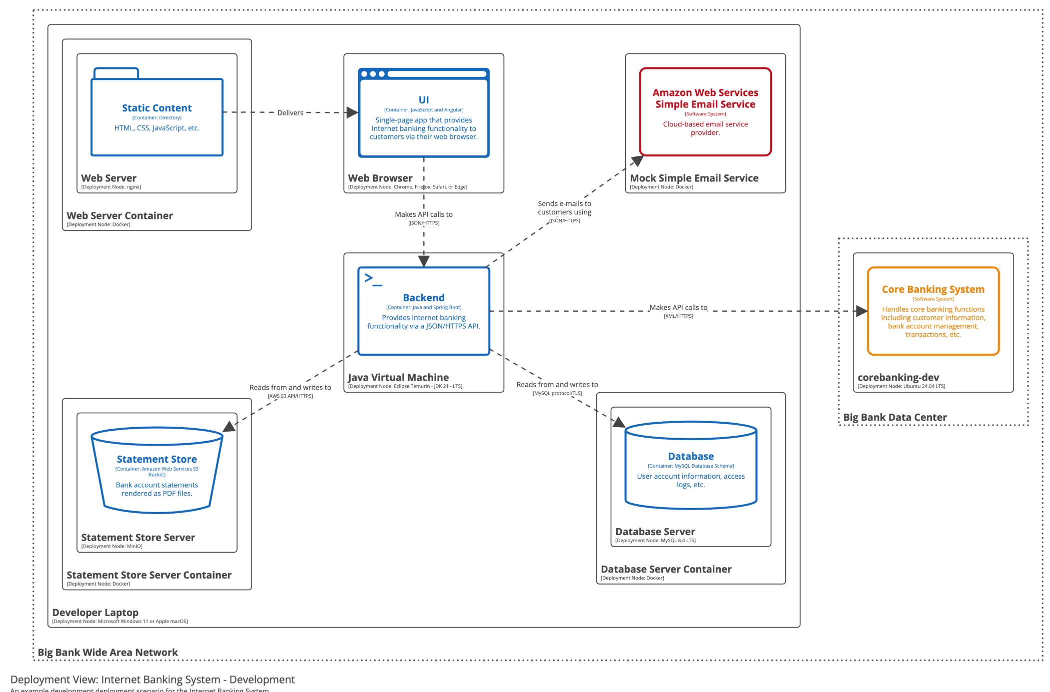
Not recommended



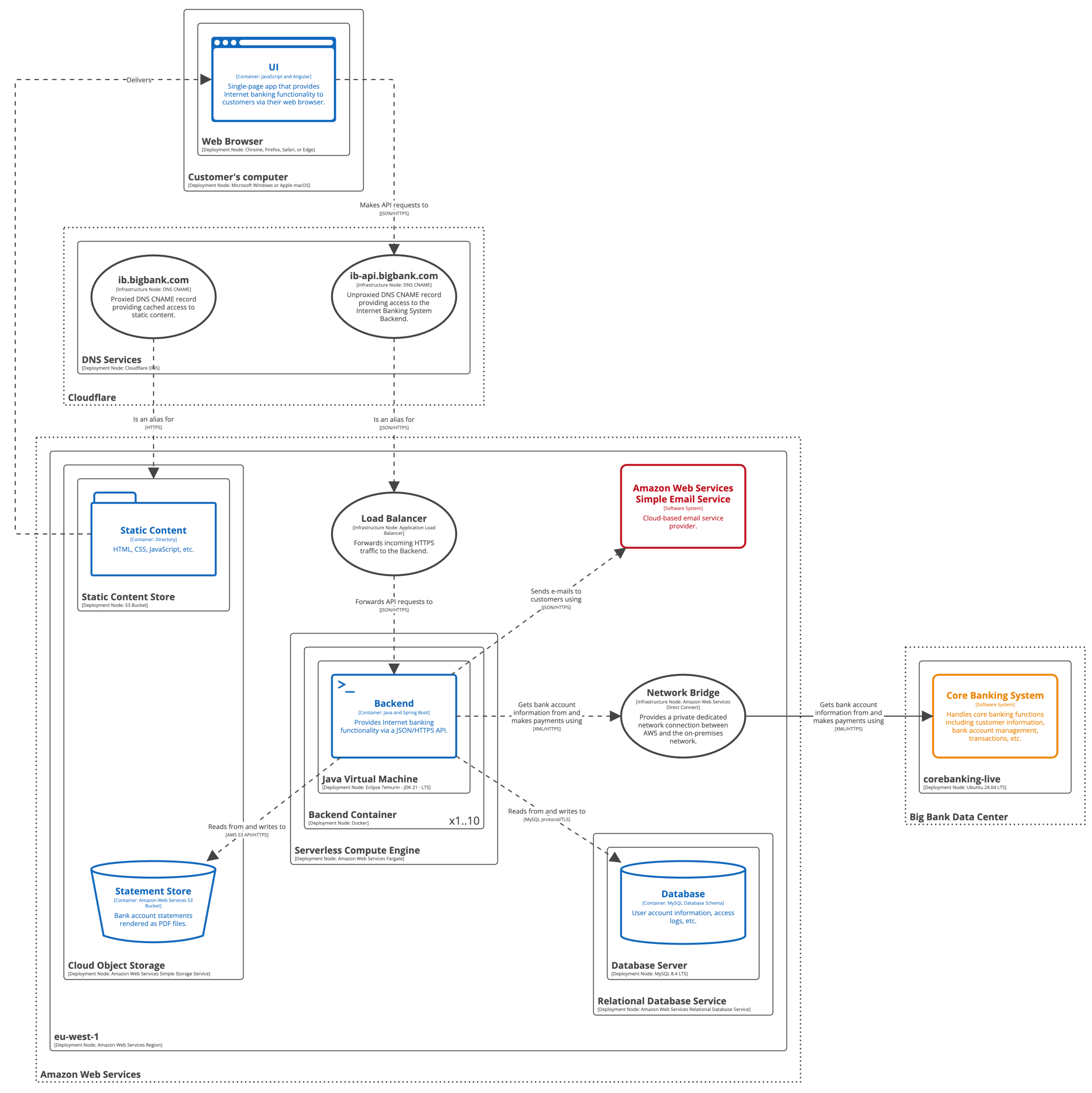
Dynamic View: Internet Banking System - Backend
Summarises how the sign in feature works in the single-page application



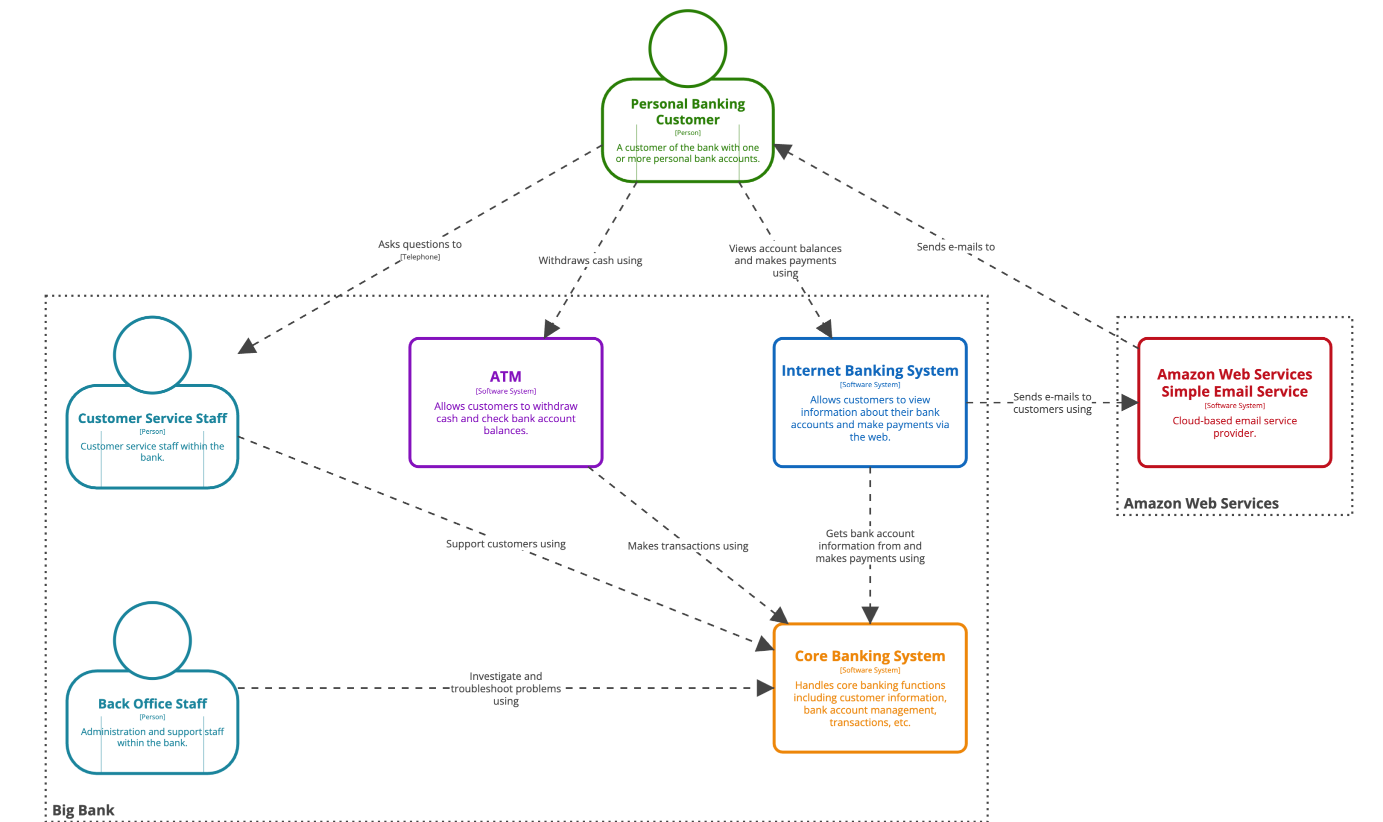
Dynamic View: Internet Banking System - Backend
Summarises how the sign in feature works in the single-page application



Deployment View: Internet Banking System - Development
An example development deployment scenario for the Internet Banking System



Deployment View: Internet Banking System - Live
An example live deployment scenario for the Internet Banking System



System Landscape View
A partial system landscape diagram for a fictional bank

Supporting diagrams

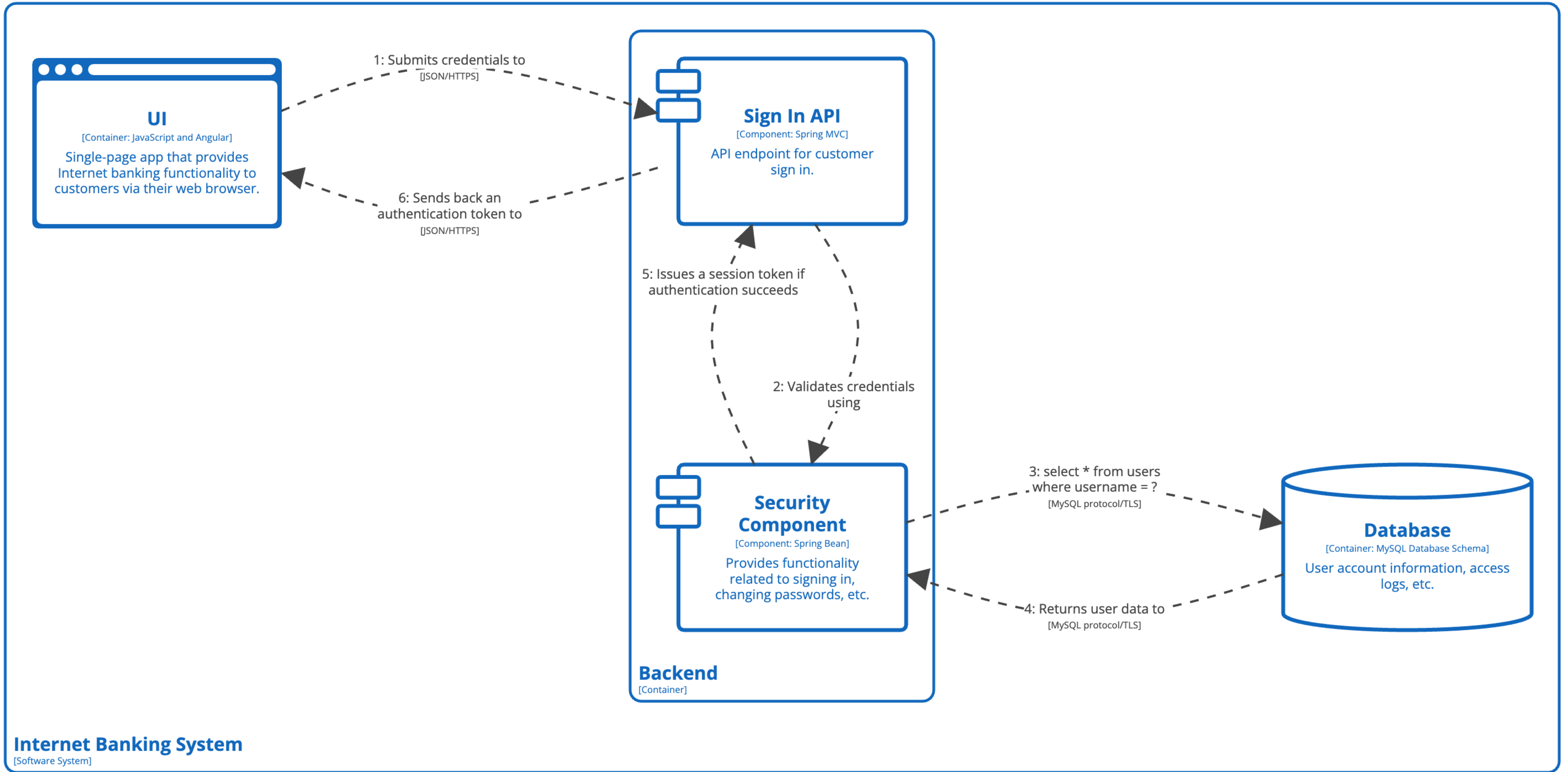
Dynamic

Deployment

System Landscape

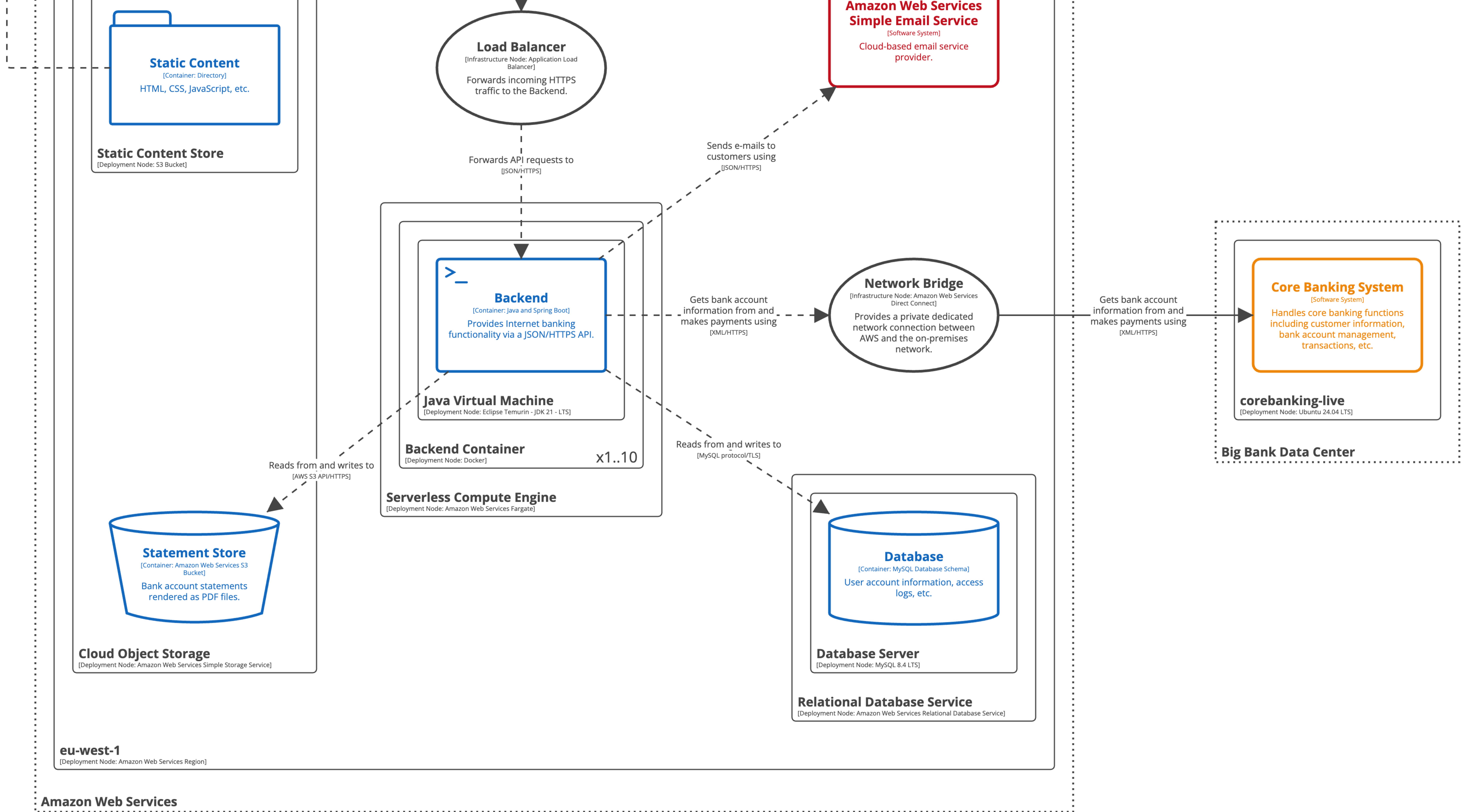
Dynamic View: Internet Banking System - Backend

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



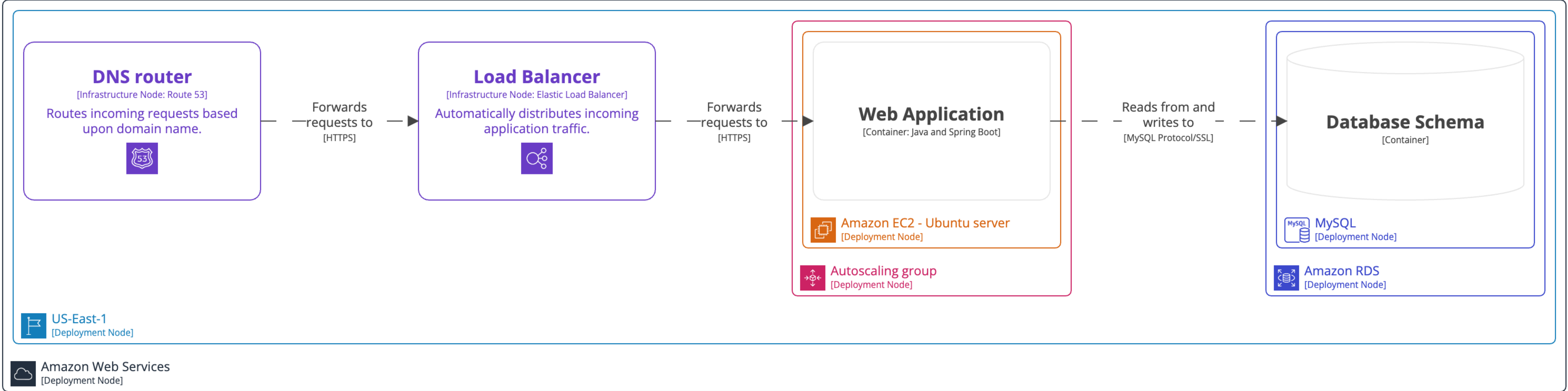
Dynamic View: Internet Banking System - Backend

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



Deployment View: Internet Banking System - Live

An example live deployment scenario for the Internet Banking System


DNS router
[Infrastructure Node: Route 53]
Routes incoming requests based upon domain name.
Forwards requests to
[HTTPS]
Load Balancer
[Infrastructure Node: Elastic Load Balancer]
Automatically distributes incoming application traffic.
Forwards requests to
[HTTPS]
Web Application
[Container: Java and Spring Boot]
Amazon EC2 - Ubuntu server
[Deployment Node]
Autoscaling group
[Deployment Node]
Reads from and writes to
[MySQL Protocol/SSL]
Database Schema
[Container]
MySQL
[Deployment Node]
Amazon RDS
[Deployment Node]
US-East-1
[Deployment Node]
Amazon Web Services
[Deployment Node]

1. Containers

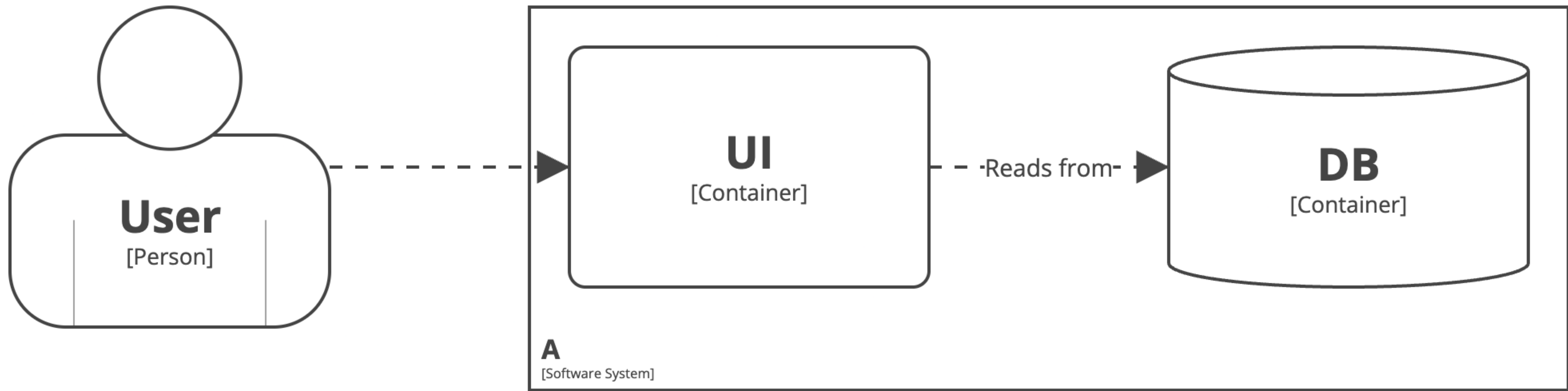
Home
Introduction
History
Abstractions
Diagrams
Tooling
FAQ
Interactive example
Book
Video
Training & workshops
Patreon & Discord

Perhaps just a week later I presented a session at IASA London titled "Where do you start?" - a talk about how you take a vague idea and turn it into a technical solution, by starting with the "big picture" and working down into the detail. I presented the same camera lens analogy (now wide angle, standard, telephoto, macro), but the four defined levels of abstractions (systems, containers, components, and classes) made an appearance:

My workshop at QCon London 2011 (this time titled, "Designing software, drawing pictures") builds upon all of this, being more explicit about the hierarchy of diagrams and elements.

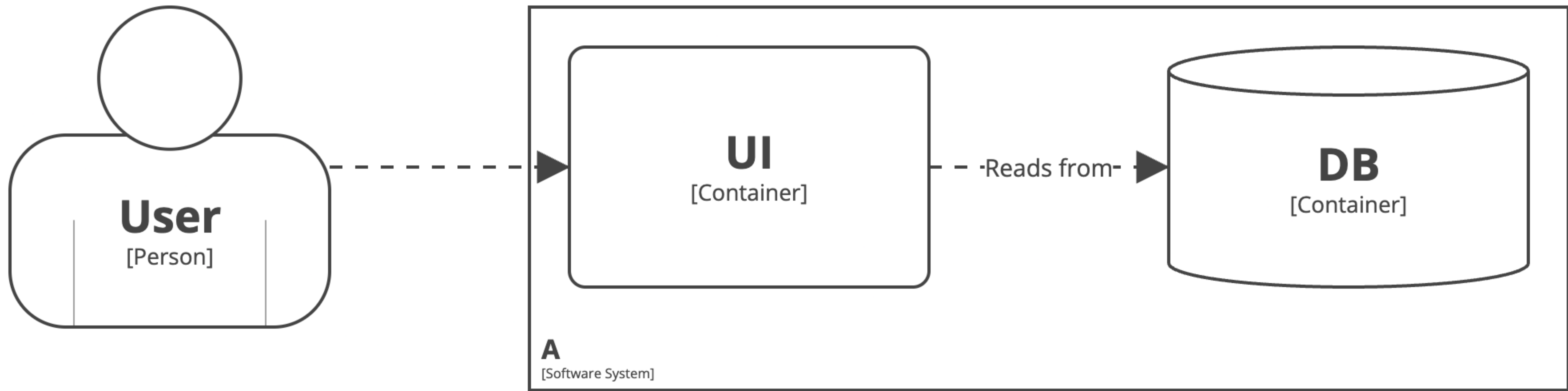
C4 containers are

applications or **data stores**

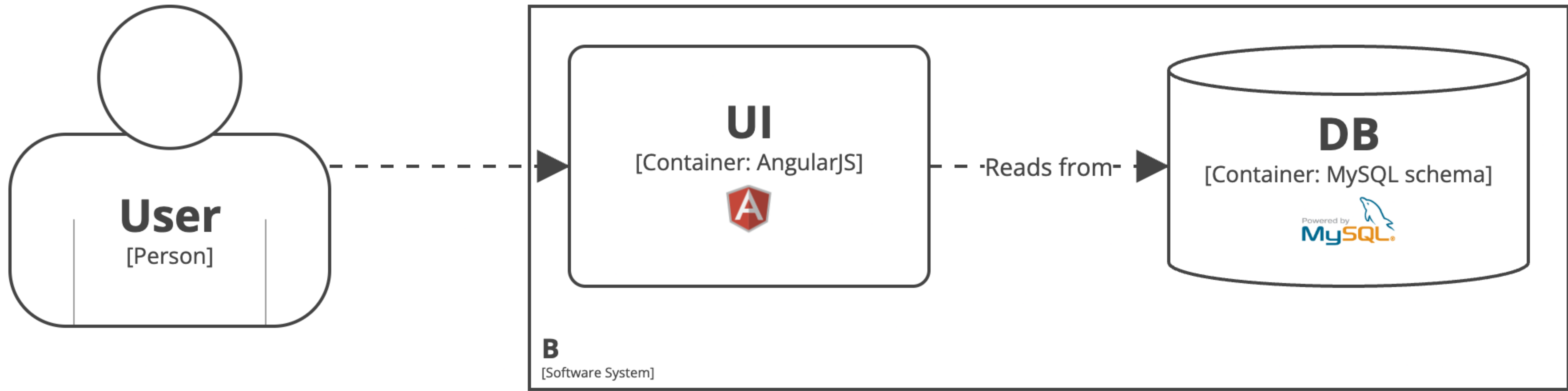


Container View: A

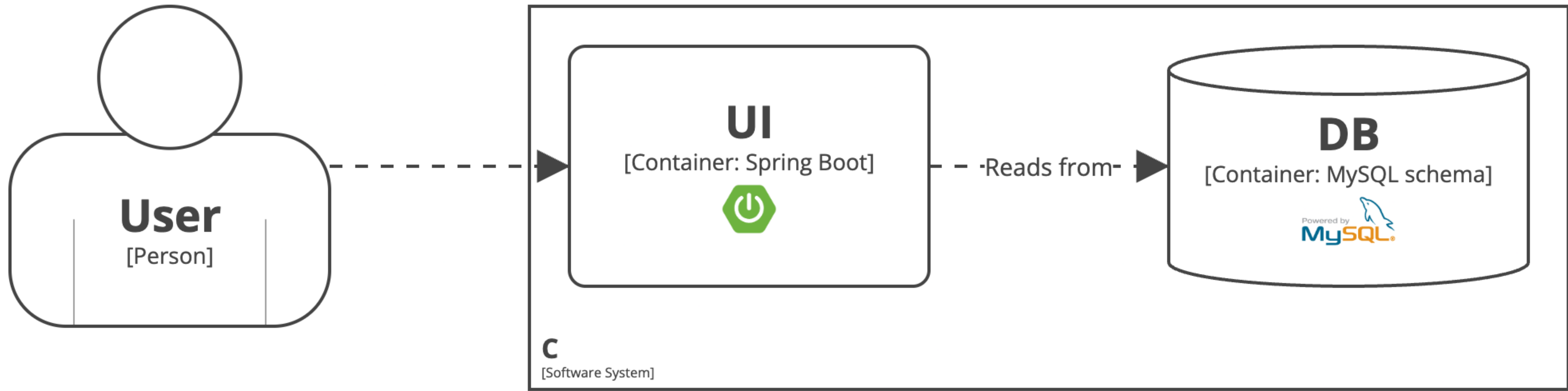
Technology choices are important



Container View: A



Container View: B



Container View: C

Struggling with the container concept?

Think of it as an operating system process boundary

Container to container interactions are typically out-of-process

(e.g. JSON/HTTPS, gRPC)

Component to component interactions within the same container are typically in-process

(e.g. method/function calls)

Application 1
[Container: Java and Spring Boot]

Java Virtual Machine

Application 2
[Container: Java and Spring Boot]

Java Virtual Machine

Out-of-process call (e.g. JSON/HTTPS, gRPC, etc)

Application 1
[Container: Java and Spring Boot]

Java Virtual Machine

Application 2
[Container: Java and Spring Boot]

Java Virtual Machine

Application 1
[Container: Java EE]

Application 2
[Container: Java EE]

Apache Tomcat

Java Virtual Machine

Application 1
[Container: Java EE]

ClassLoader

Application 2
[Container: Java EE]

ClassLoader

Apache Tomcat

Java Virtual Machine

Out-of-process cal
(e.g. JSON/HTTPS, gRPC, etc)

Application 1
[Container: Java EE]

ClassLoader

Application 2
[Container: Java EE]

ClassLoader

Apache Tomcat

Java Virtual Machine

Struggling with the container concept?

Think of it as an ~~operating system~~ ~~process boundary~~ isolation boundary

Service A

[Container: Java and MySQL]

Is this correct?

Service A
[Container: Java and H2 Database]

Is this correct?

C4 containers are separately deployable things, but the C4 container diagram should not show deployment concepts

Service A
[Container: Docker]

Is this correct?

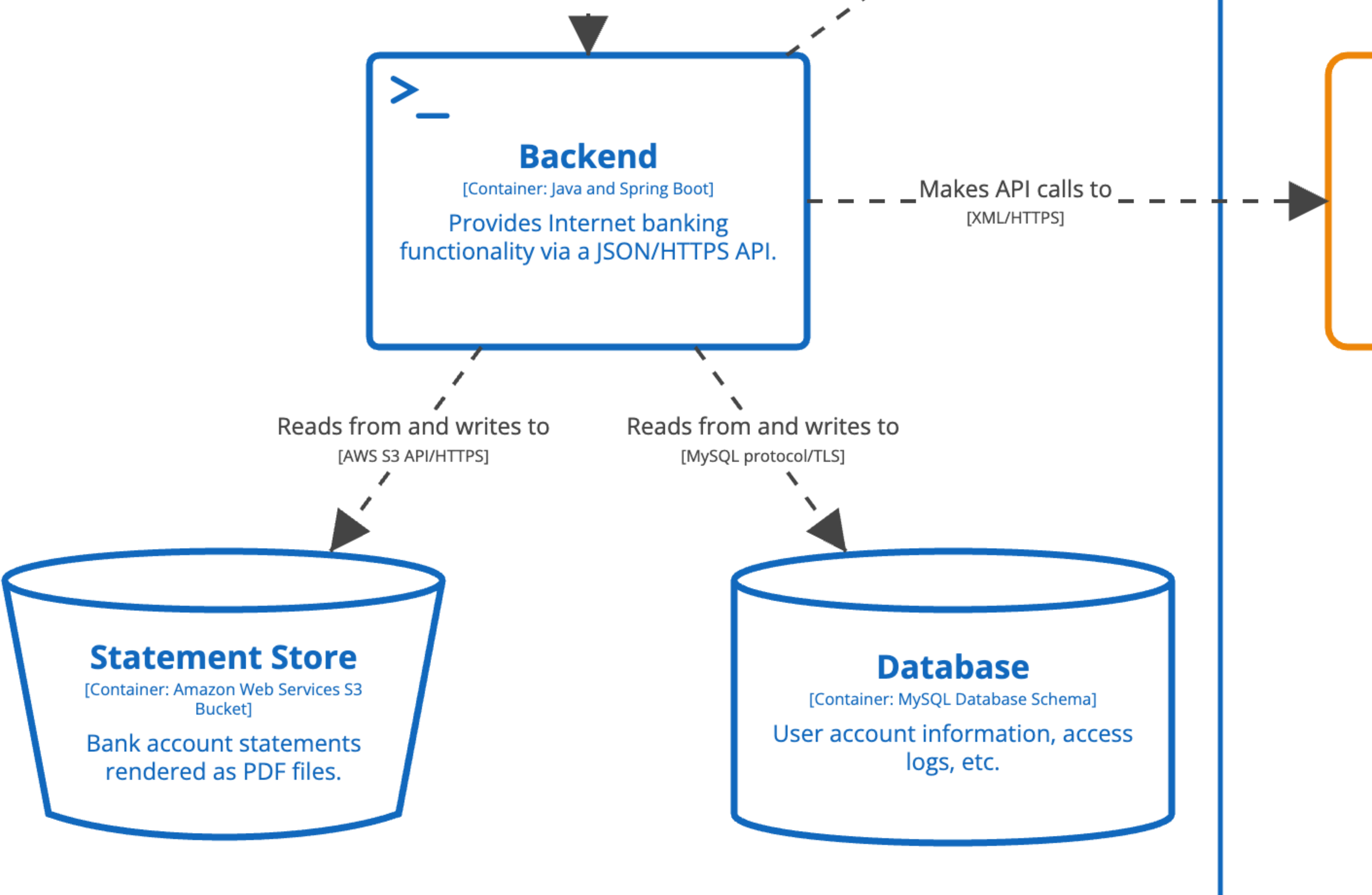
Service A
[Container: Amazon Web Services EC2]

Is this correct?

Service A

[Container: Amazon Web Services Fargate]

Is this correct?


Backend
[Container: Java and Spring Boot]
Provides Internet banking functionality via a JSON/HTTPS API.
Makes API calls to
[XML/HTTPS]
Reads from and writes to
[AWS S3 API/HTTPS]
Reads from and writes to
[MySQL protocol/TLS]
Statement Store
[Container: Amazon Web Services S3 Bucket]
Bank account statements rendered as PDF files.
Database
[Container: MySQL Database Schema]
User account information, access logs, etc.

Backend
[Container: Java and Spring Boot]
Provides Internet banking functionality via a JSON/HTTPS API.
Java Virtual Machine
[Deployment Node: Eclipse Temurin - JDK 21 - LTS]
Backend Container
[Deployment Node: Docker]
x1..10
Serverless Compute Engine
[Deployment Node: Amazon Web Services Fargate]
Gets bank account information from and makes payments using
[XML/HTTPS]
Network Bridge
[Infrastructure Node: Amazon Web Services Direct Connect]
Provides a private dedicated network connection between AWS and the on-premises network.
Reads from and writes to
[AWS S3 API/HTTPS]
Reads from and writes to
[MySQL protocol/TLS]
Statement Store
[Container: Amazon Web Services S3 Bucket]
Bank account statements rendered as PDF files.
Cloud Object Storage
[Deployment Node: Amazon Web Services Simple Storage Service]
Database
[Container: MySQL Database Schema]
User account information, access logs, etc.
Database Server
[Deployment Node: MySQL 8.4 LTS]
Relational Database Service
[Deployment Node: Amazon Web Services Relational Database Service]

UI
[Container: Javascript]

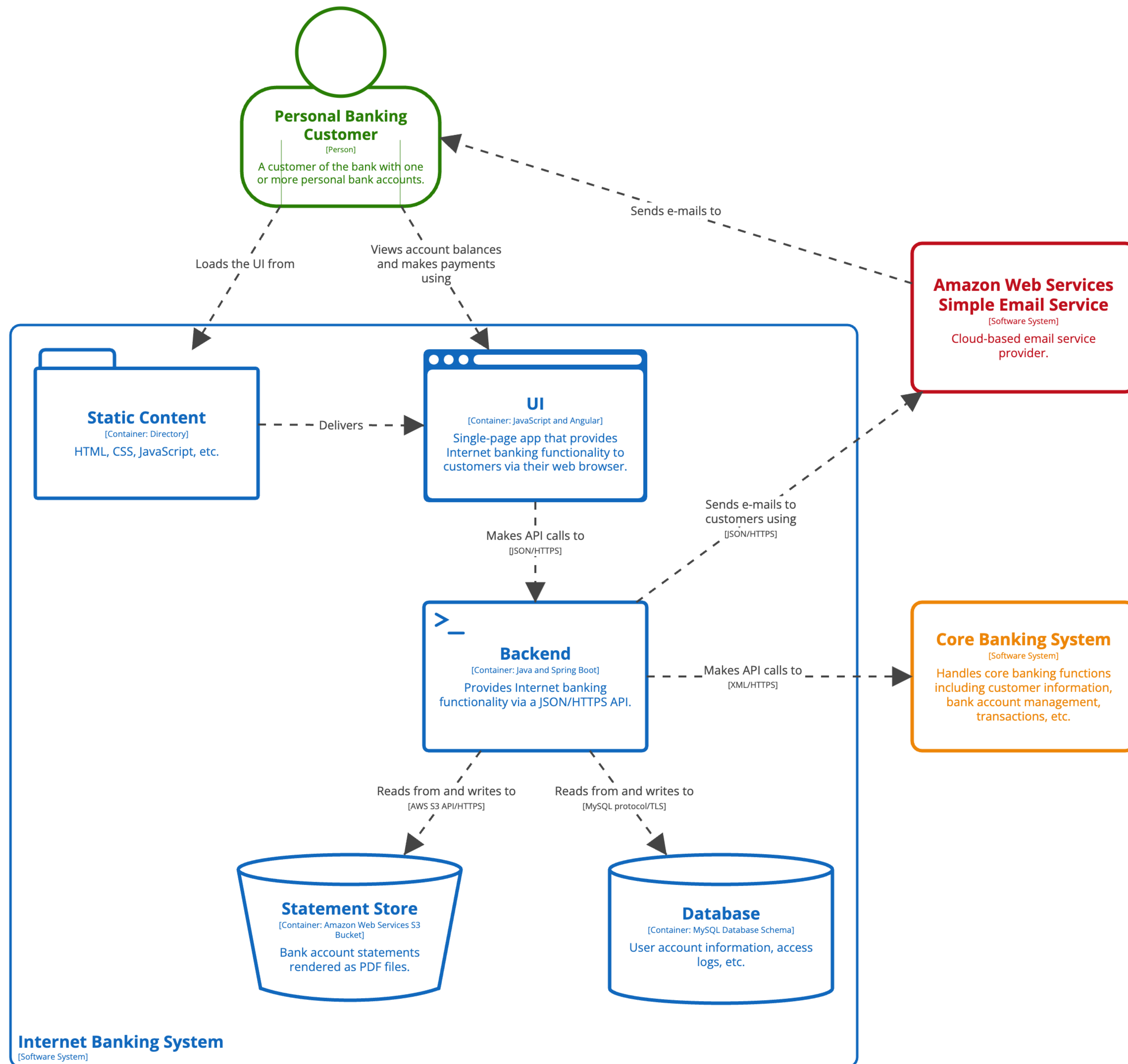
API Gateway
[Container: ...]

Backend
[Container: Java]

Is this correct?

Think of the container diagram
as showing the subset of elements
that will run across
all deployment environments

Why is AWS SES a software system, whereas the S3 bucket is a container?



Container View: Internet Banking System

The container diagram for the Internet Banking System

“You use a library,
a framework uses you”

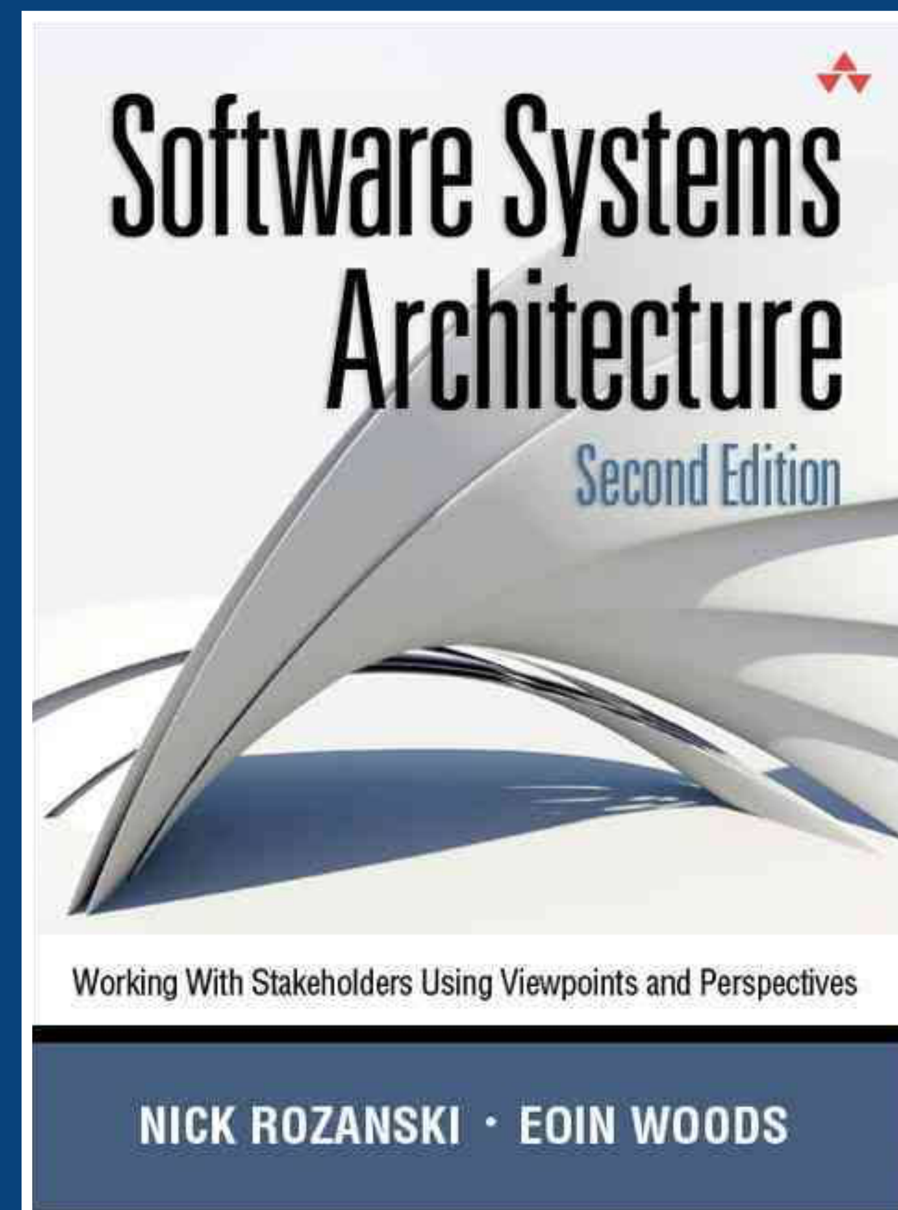
What is the type and strength of the dependency?

SES is a service that the Internet Banking System uses

The S3 bucket is an integral part of the Internet Banking System

2. Perspectives

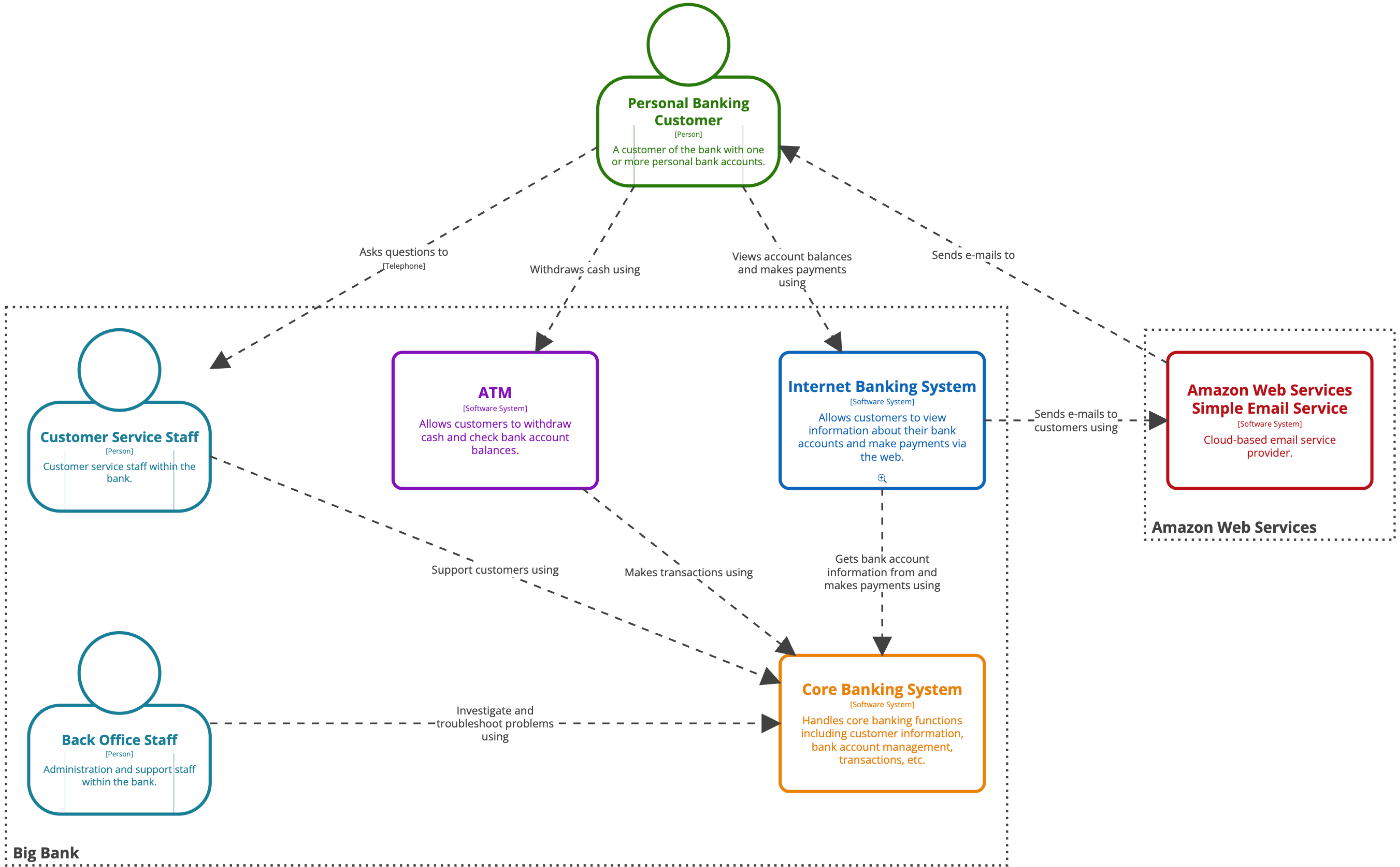
What about ownership, security, technical debt, etc?



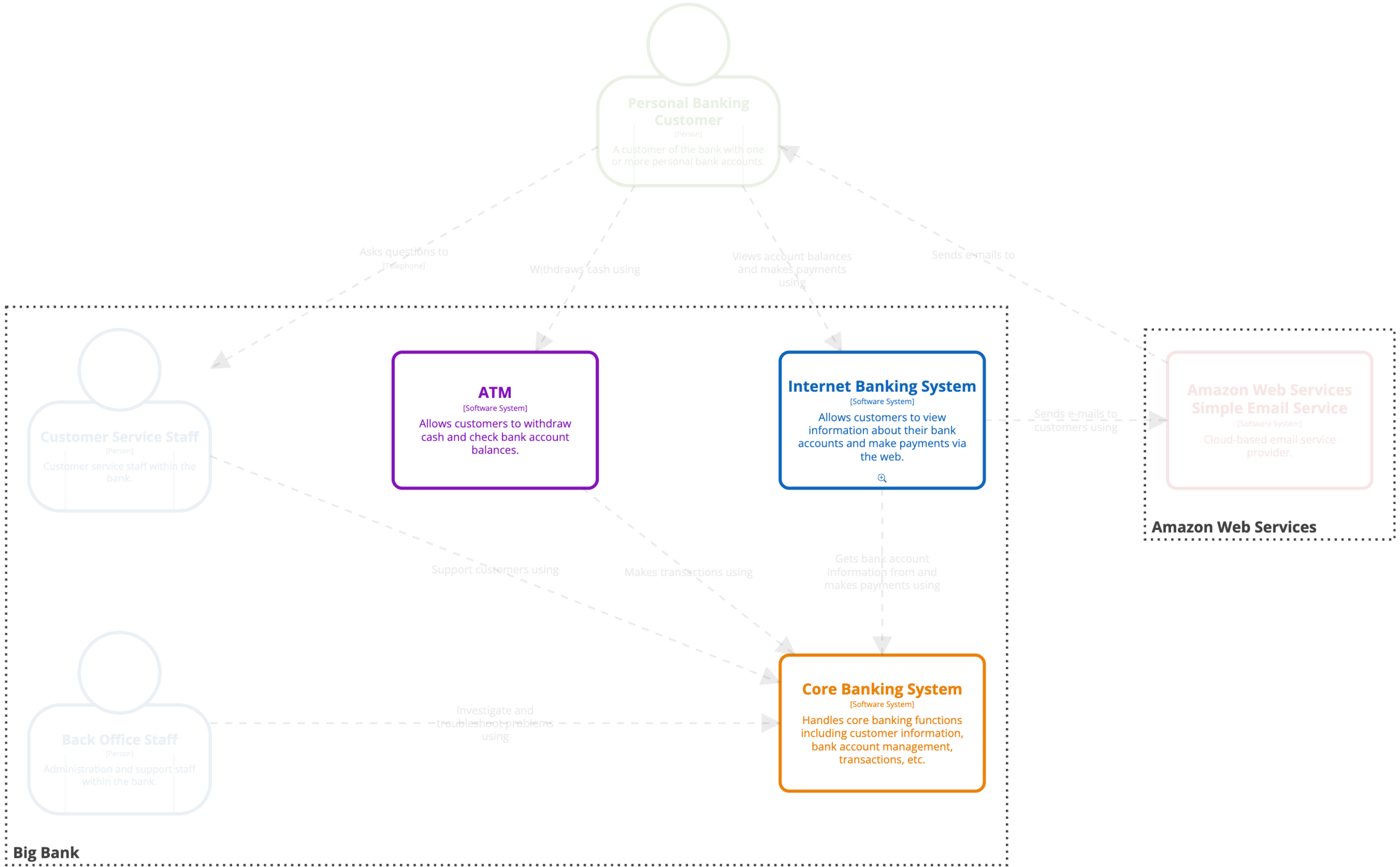
Perspectives

Rather than defining another viewpoint and creating another view, we need some way to modify and enhance our existing views to ensure that our architecture exhibits the desired quality properties. We therefore need something in our conceptual model that can be considered "orthogonal" to viewpoints, and we have coined the term architectural perspective (which we shorten to perspective) to refer to it.

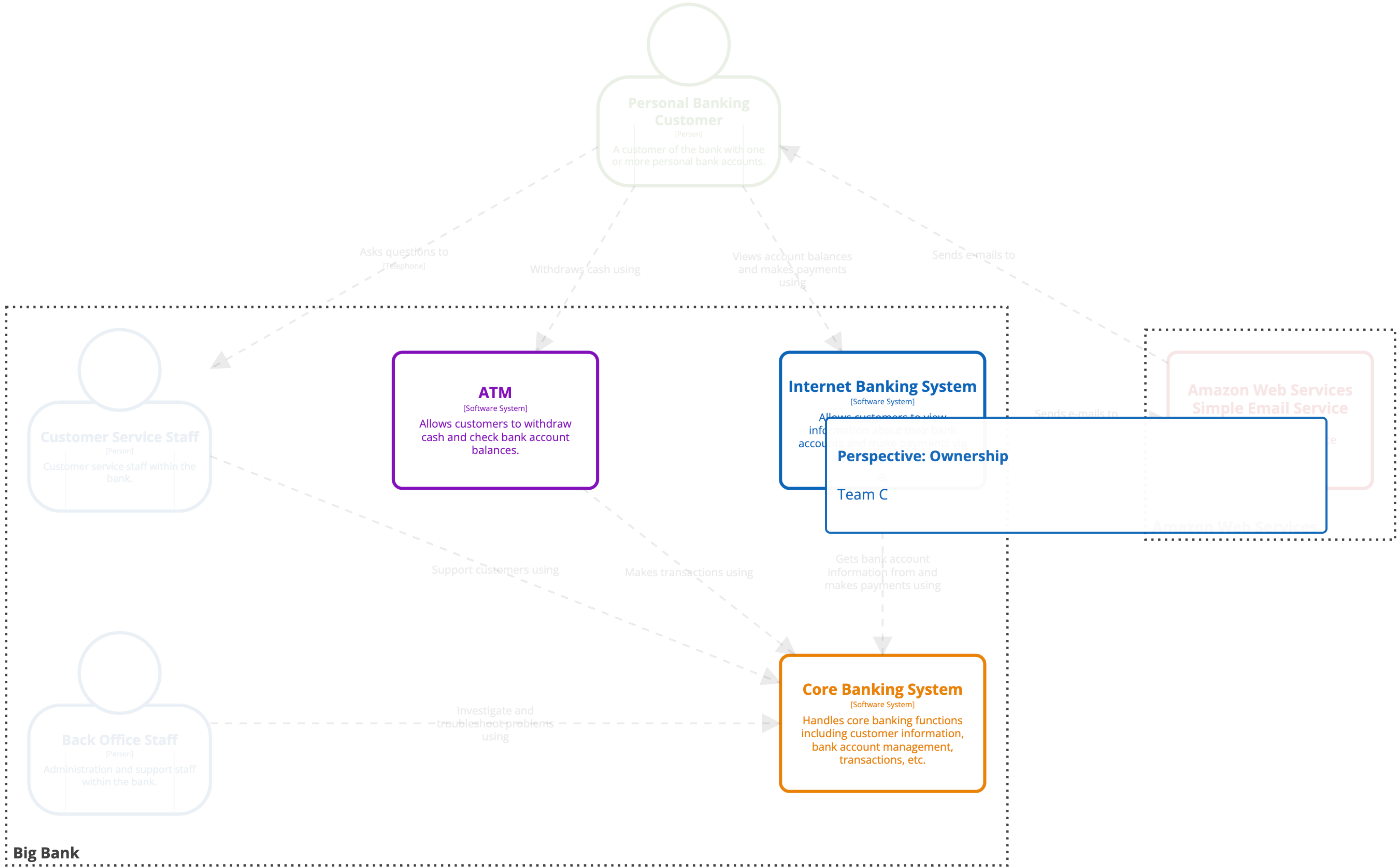
https://www.viewpoints-and-perspectives.info/home/perspectives/

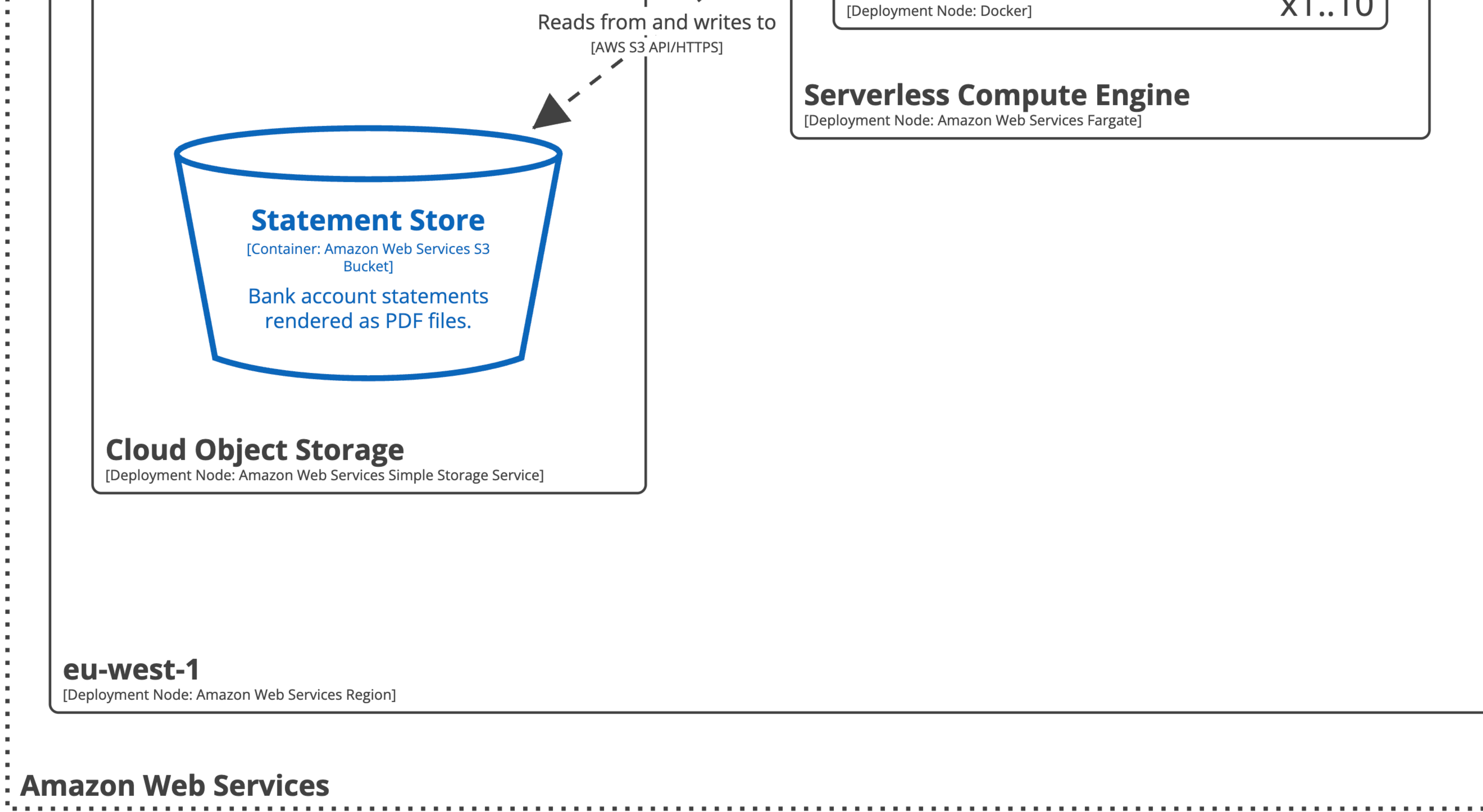


Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.
Asks questions to
[Telephone]
Withdraws cash using
Views account balances and makes payments using
Sends e-mails to
Customer Service Staff
[Person]
Customer service staff within the bank.
ATM
[Software System]
Allows customers to withdraw cash and check bank account balances.
Internet Banking System
[Software System]
Allows customers to view information about their bank accounts and make payments via the web.
Sends e-mails to customers using
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Amazon Web Services
Support customers using
Makes transactions using
Gets bank account information from and makes payments using
Back Office Staff
[Person]
Administration and support staff within the bank.
Investigate and troubleshoot problems using
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
Big Bank



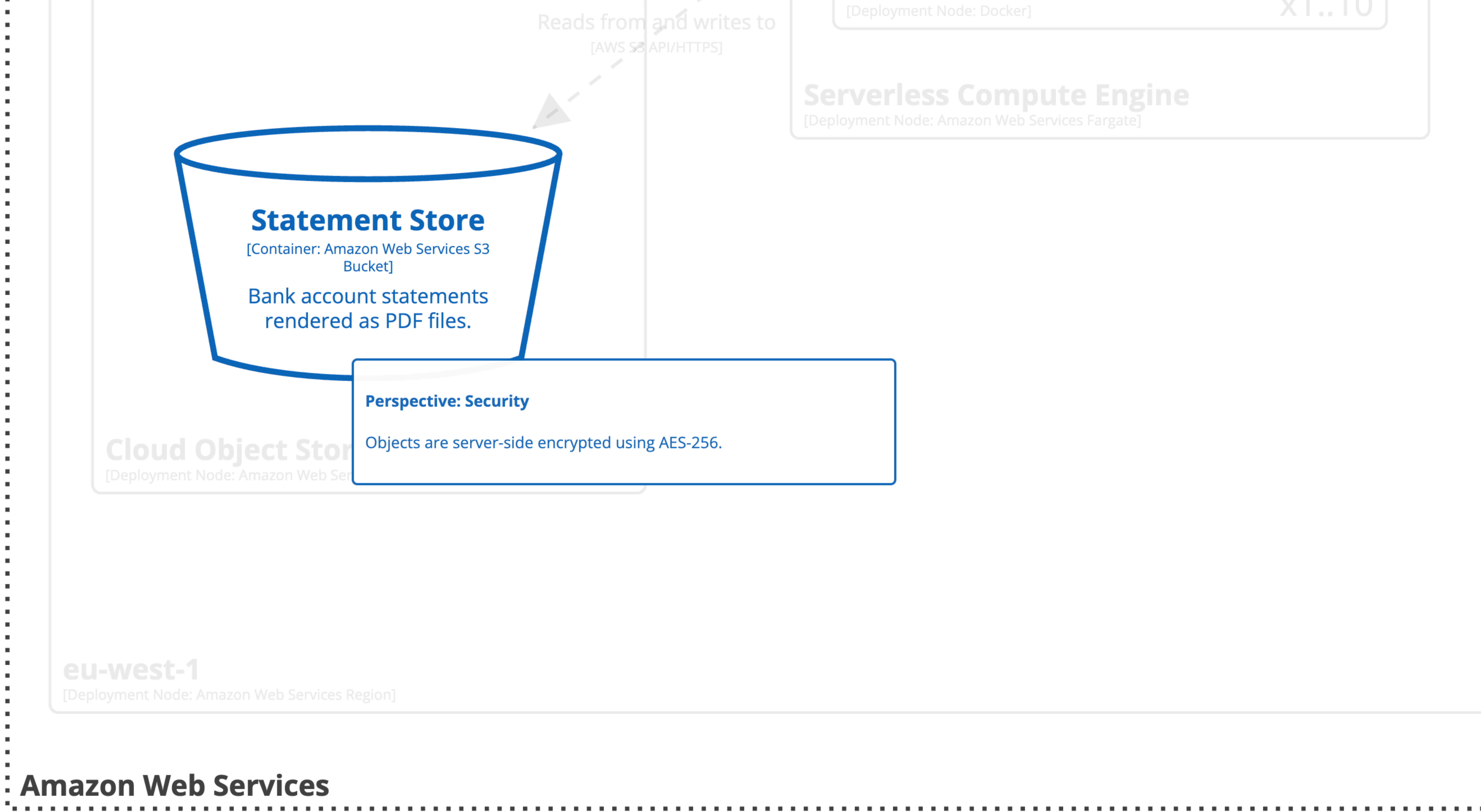
Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.
Asks questions to
[Telephone]
Withdraws cash using
Views account balances and makes payments using
Sends e-mails to
Customer Service Staff
[Person]
Customer service staff within the bank.
ATM
[Software System]
Allows customers to withdraw cash and check bank account balances.
Internet Banking System
[Software System]
Allows customers to view information about their bank accounts and make payments via the web.
Sends e-mails to customers using
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Amazon Web Services
Support customers using
Makes transactions using
Gets bank account information from and makes payments using
Back Office Staff
[Person]
Administration and support staff within the bank.
Investigate and troubleshoot problems using
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
Big Bank


Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.
Asks questions to
[Telephone]
Withdraws cash using
Views account balances and makes payments using
Sends e-mails to
Customer Service Staff
[Person]
Customer service staff within the bank.
ATM
[Software System]
Allows customers to withdraw cash and check bank account balances.
Internet Banking System
[Software System]
Perspective: Ownership
Team C
Amazon Web Services Simple Email Service
Sends e-mails to
Amazon Web Services
Support customers using
Makes transactions using
Gets bank account information from and makes payments using
Back Office Staff
[Person]
Administration and support staff within the bank.
Investigate and troubleshoot problems using
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
Big Bank



Deployment View: Internet Banking System - Live

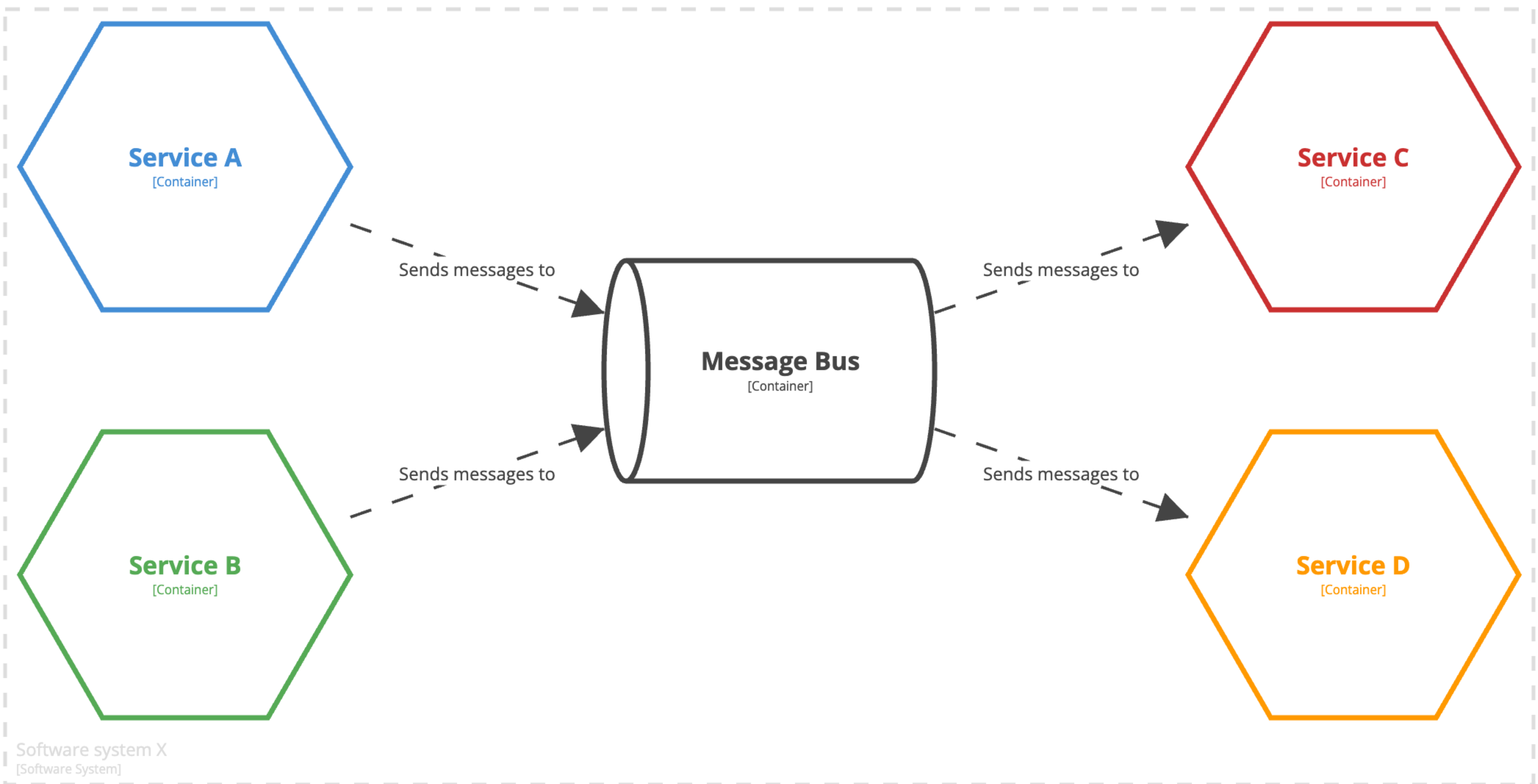
An example live deployment scenario for the Internet Banking System | Simon Brown | c4model.com | License: CC BY 4.0

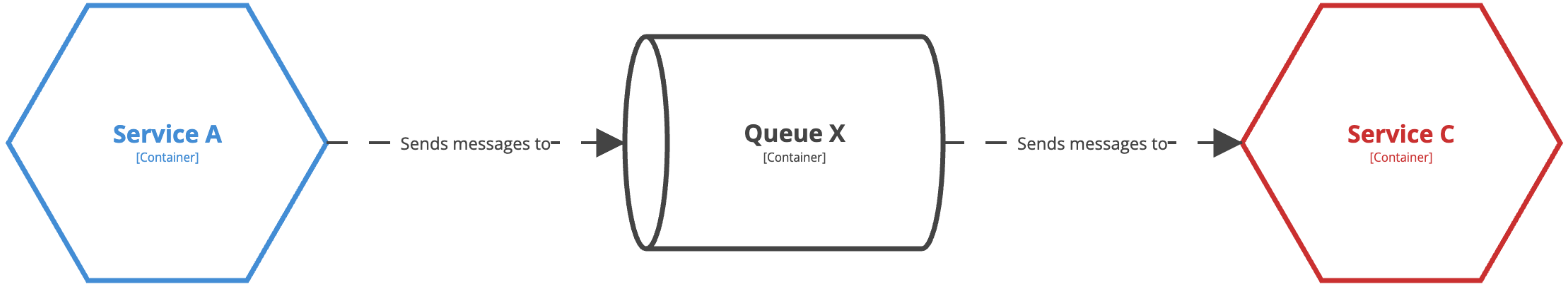


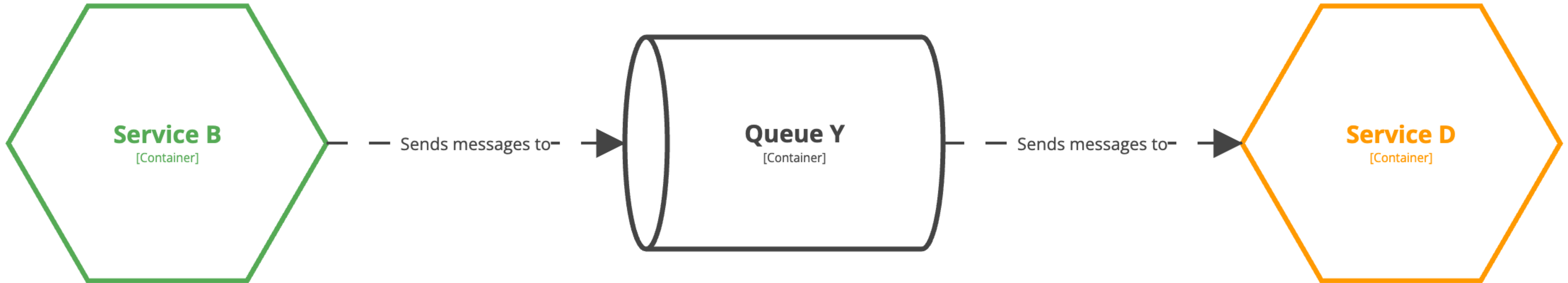
Deployment View: Internet Banking System - Live

An example live deployment scenario for the Internet Banking System | Simon Brown | c4model.com | License: CC BY 4.0

3. Message-driven architectures

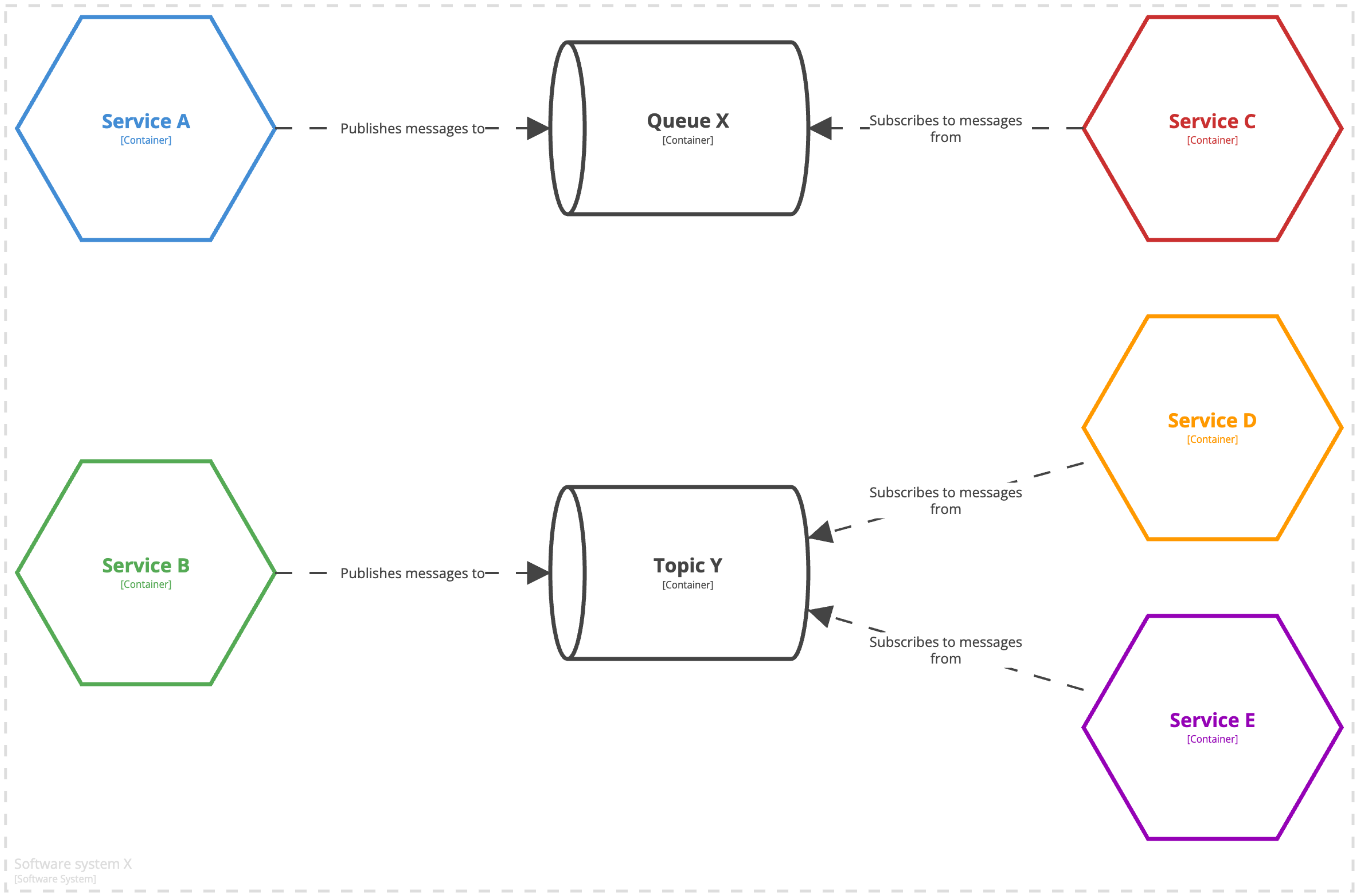

Service A
[Container]
Service C
[Container]
Sends messages to
Sends messages to
Message Bus
[Container]
Sends messages to
Sends messages to
Service B
[Container]
Service D
[Container]
Software system X
[Software System]





Software system X
[Software System]

Service A
[Container]
Sends messages to
[via Queue X]
Service C
[Container]
Service B
[Container]
Sends messages to
[via Queue Y]
Service D
[Container]
Software system X
[Software System]

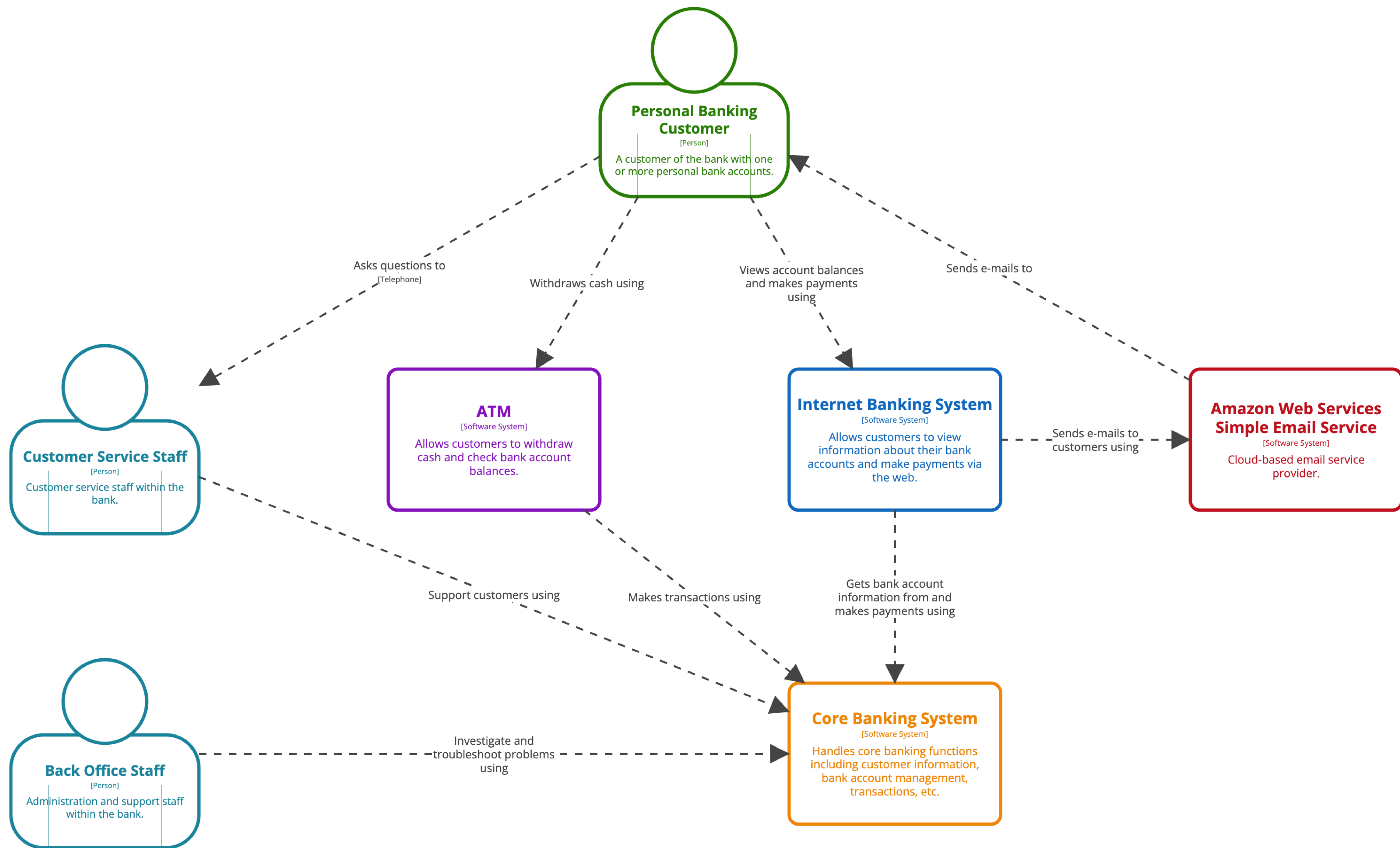

Service A
[Container]
Publishes messages to
Queue X
[Container]
Subscribes to messages from
Service C
[Container]
Service D
[Container]
Service B
[Container]
Publishes messages to
Topic Y
[Container]
Subscribes to messages from
Subscribes to messages from
Service E
[Container]
Software system X
[Software System]

4. Abstraction vs organisation

What are your thoughts on modelling additional abstractions?

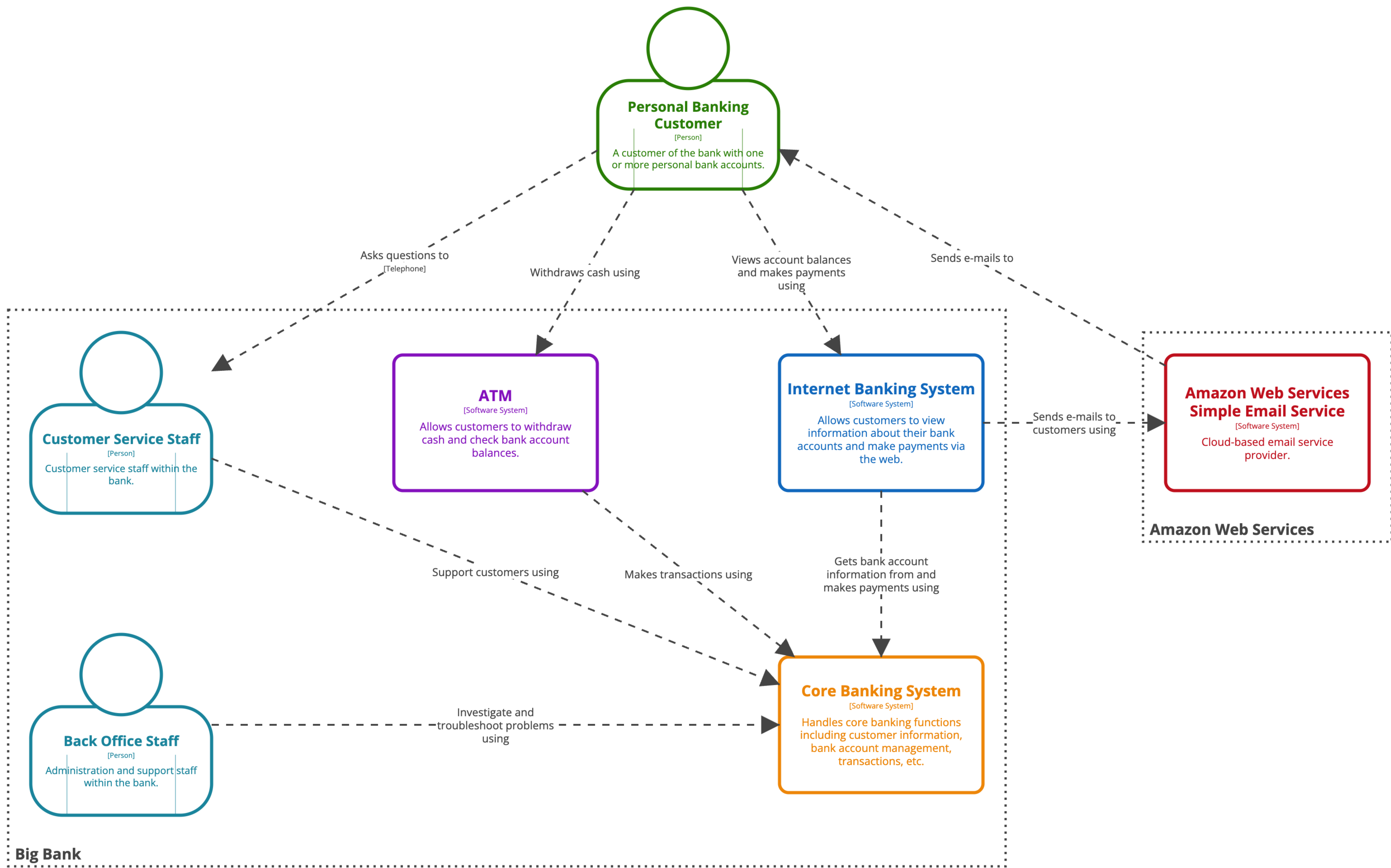
Some of these concepts
might be better thought of as
organisational constructs
rather than abstractions

Organisations, groups, departments, etc



System Landscape View

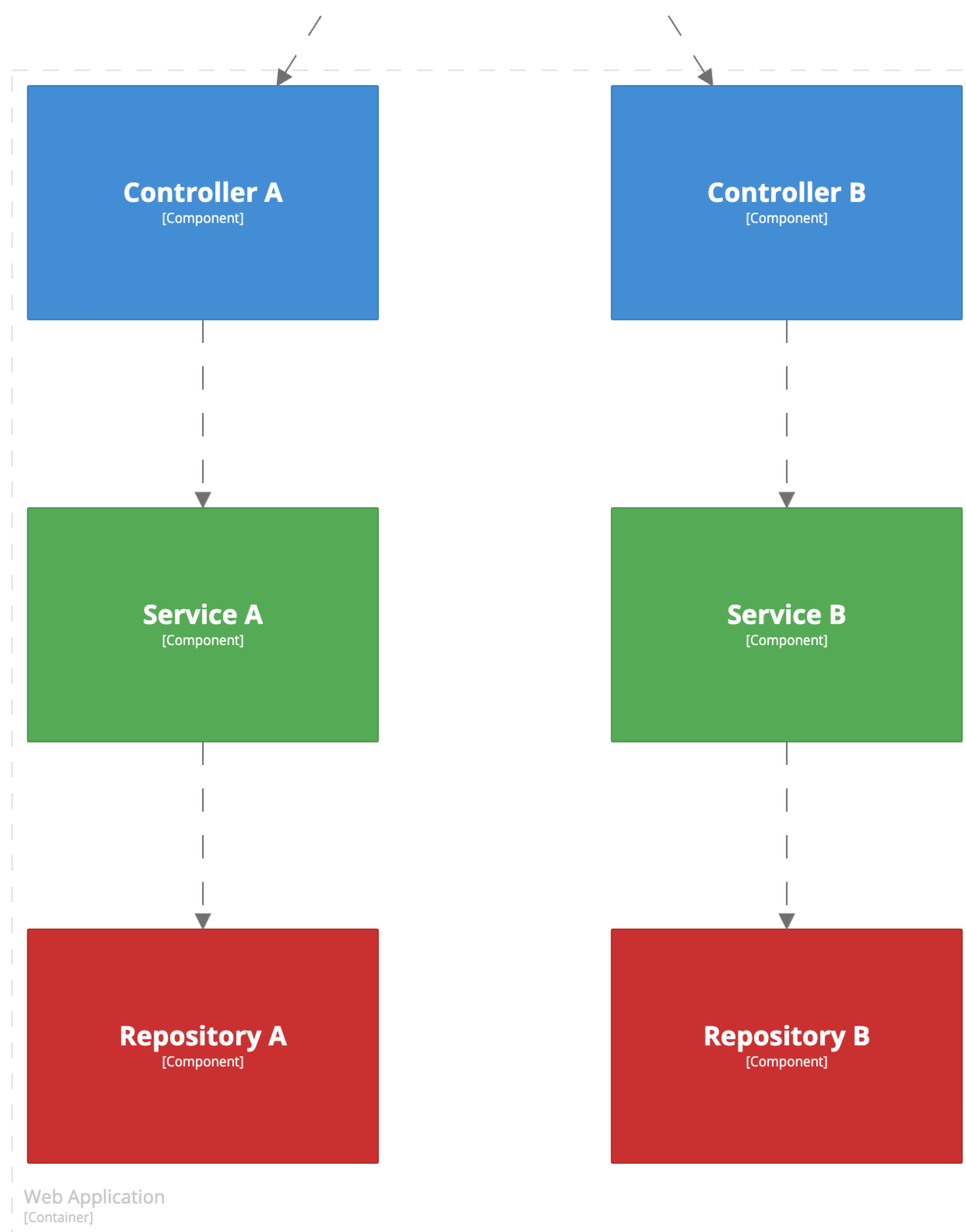
A partial system landscape diagram for a fictional bank

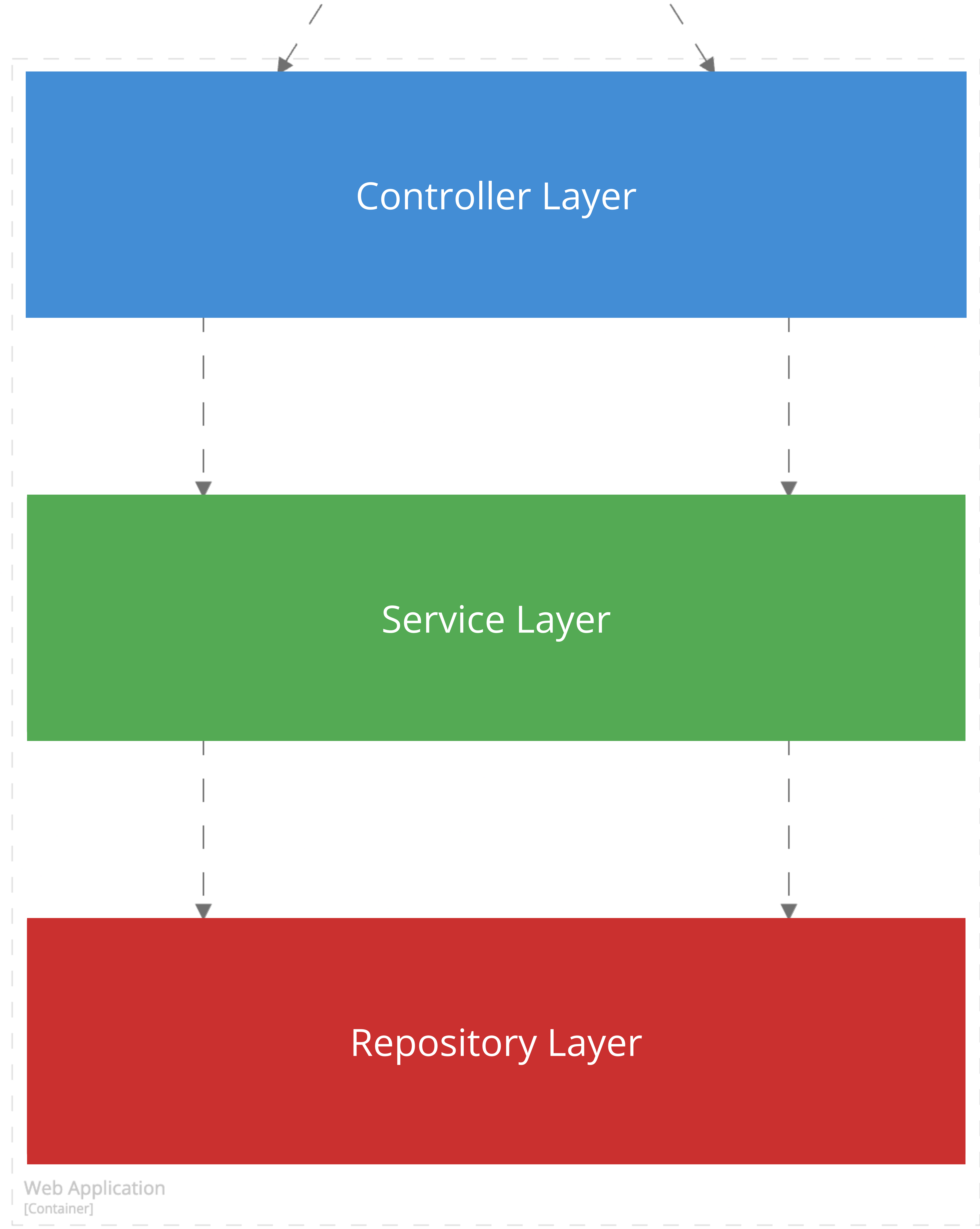


System Landscape View

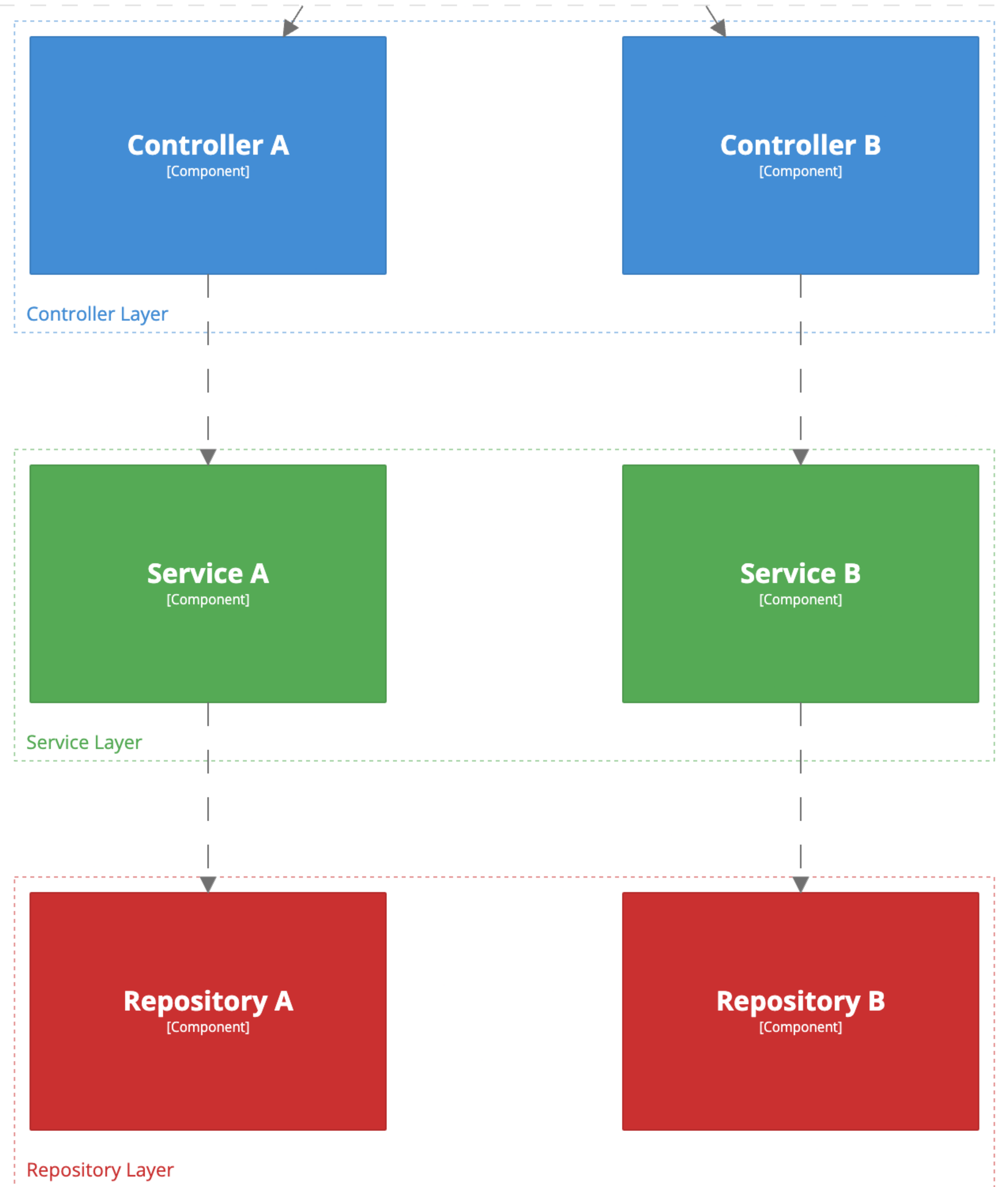
A partial system landscape diagram for a fictional bank

Architectural layers

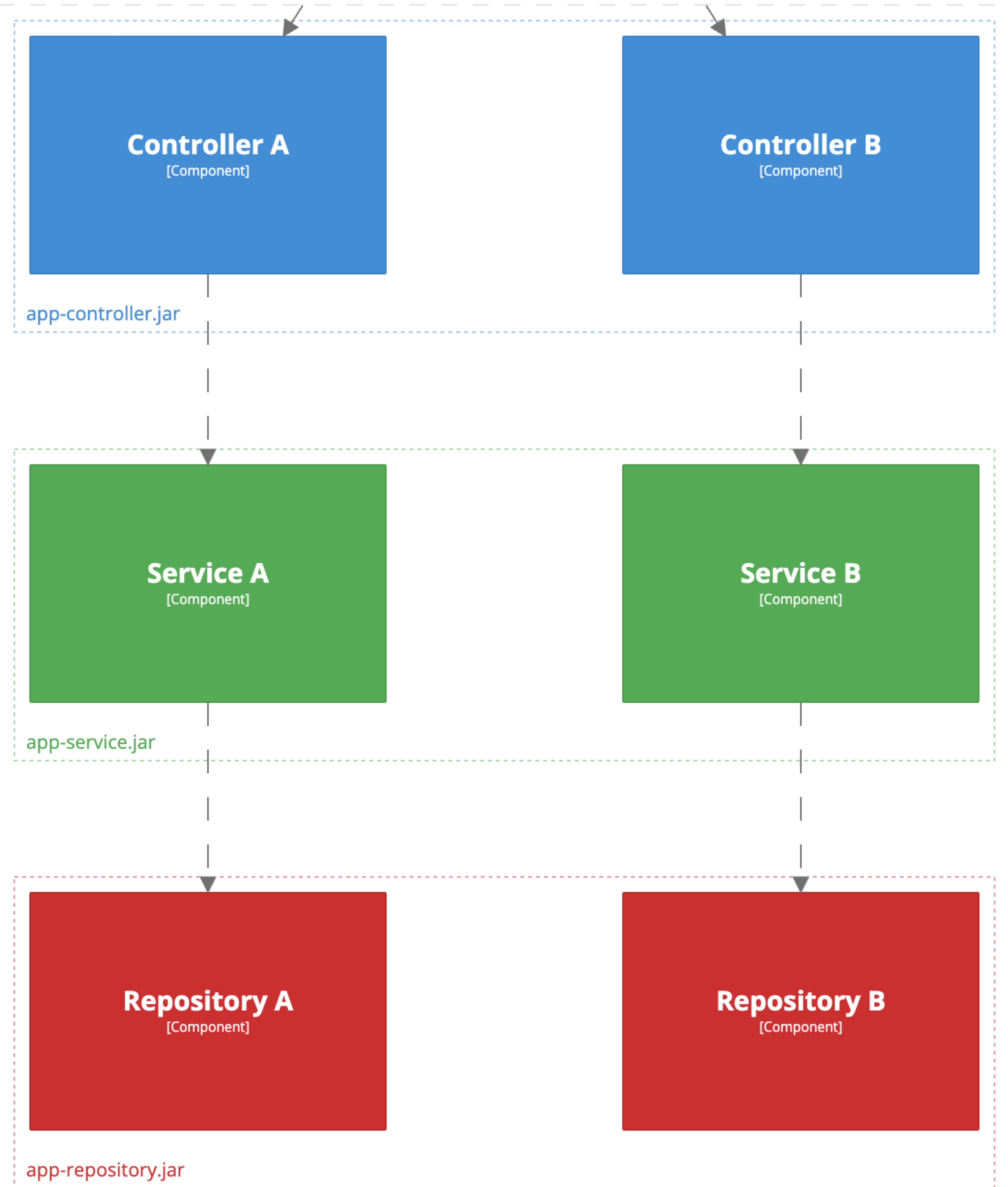

Controller A
[Component]
Controller B
[Component]
Service A
[Component]
Service B
[Component]
Repository A
[Component]
Repository B
[Component]
Web Application
[Container]



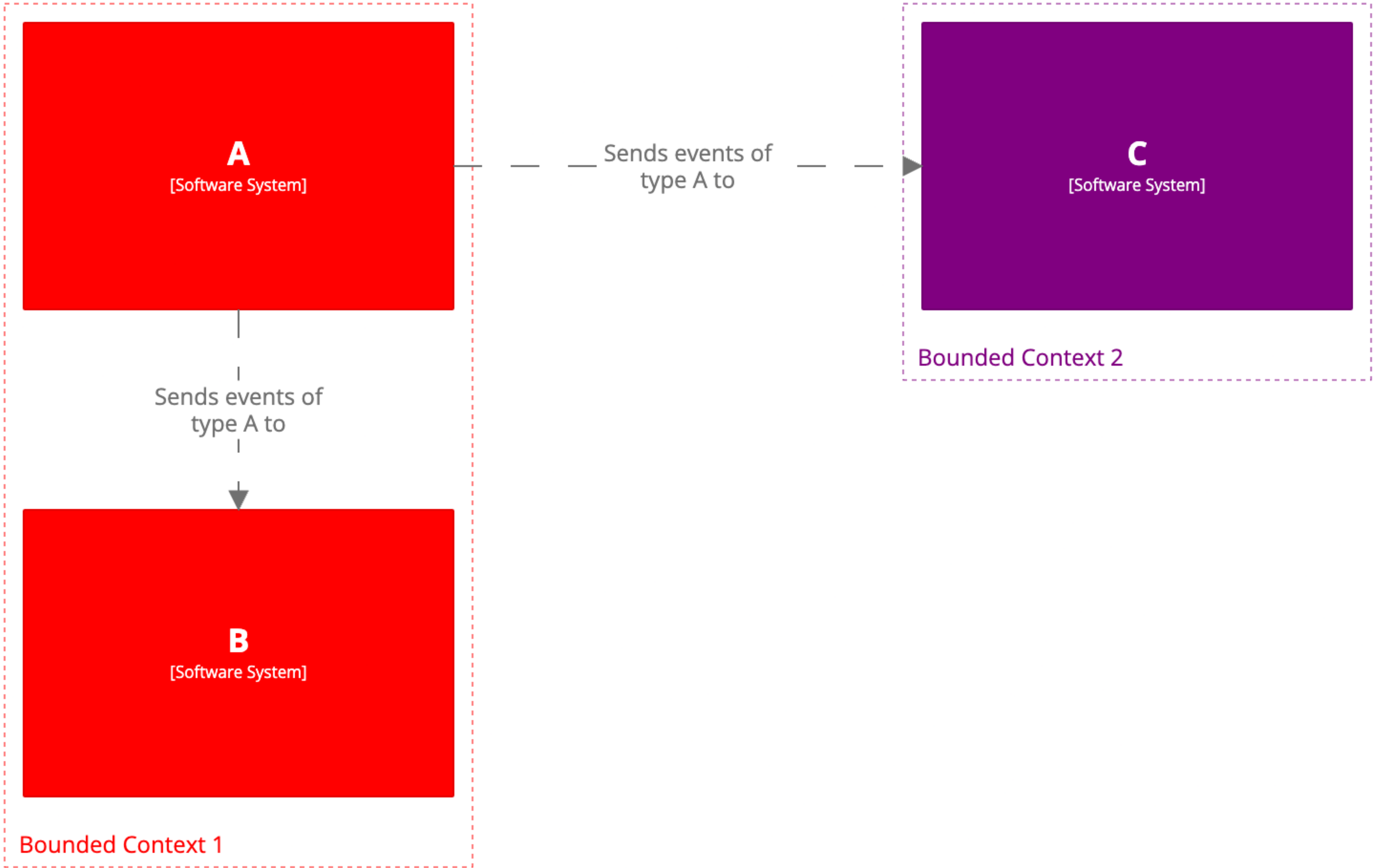
Controller Layer
Service Layer
Repository Layer
Web Application
[Container]



Controller A
[Component]
Controller B
[Component]
Controller Layer
Service A
[Component]
Service B
[Component]
Service Layer
Repository A
[Component]
Repository B
[Component]
Repository Layer


Controller A
[Component]
Controller B
[Component]
app-controller.jar
Service A
[Component]
Service B
[Component]
app-service.jar
Repository A
[Component]
Repository B
[Component]
app-repository.jar

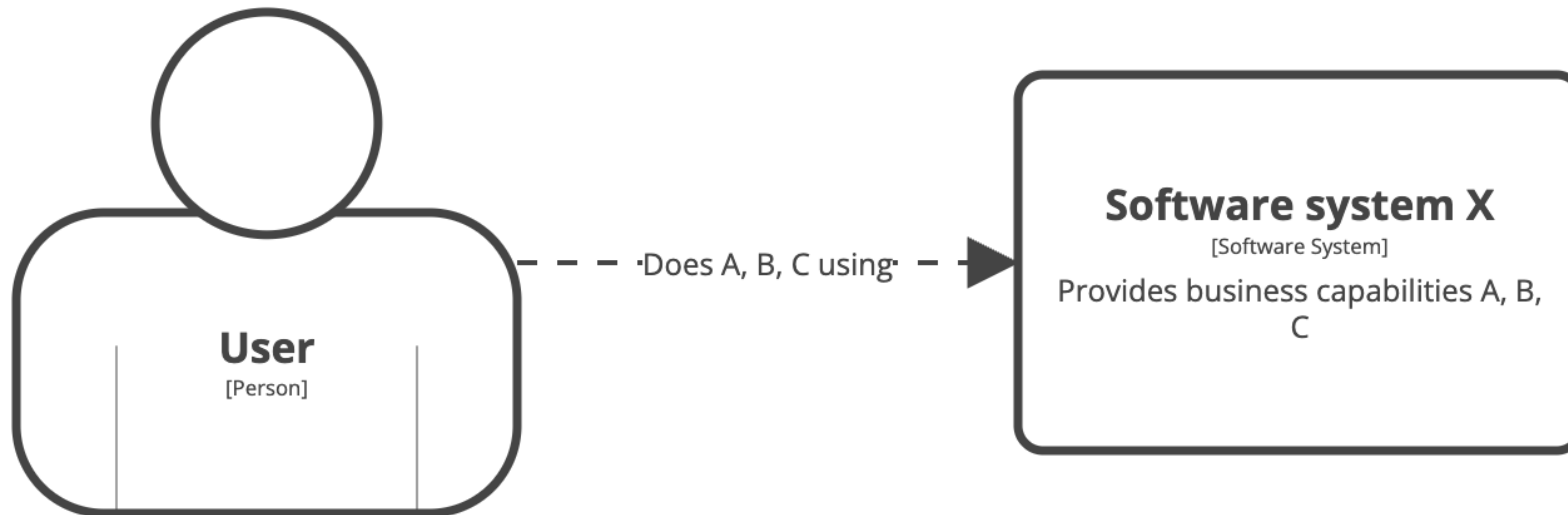
Bounded contexts


A
[Software System]
Sends events of type A to
C
[Software System]
Bounded Context 2
Sends events of type A to
B
[Software System]
Bounded Context 1

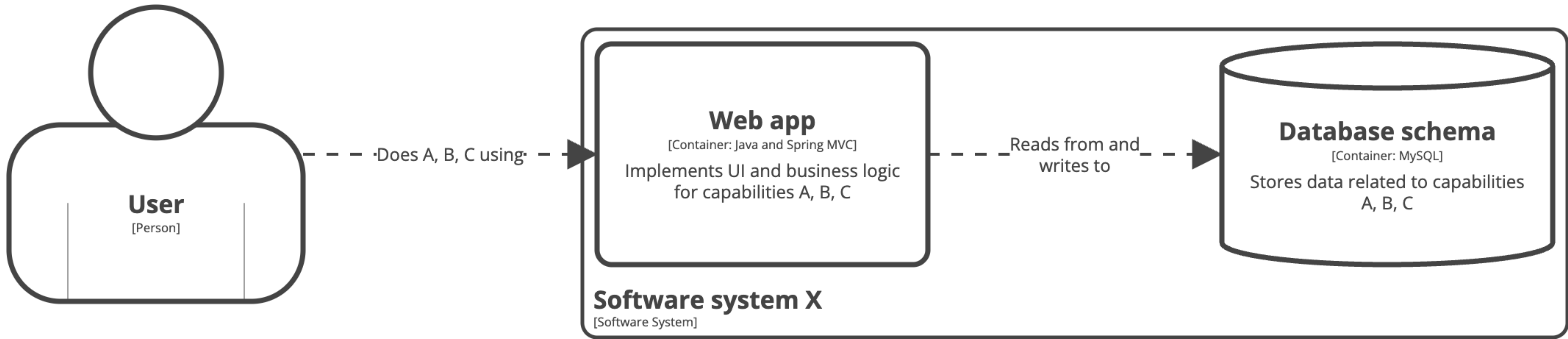
5. Microservices

A microservice should be modelled
as a **software system**
or a **group of containers**

(monolithic architecture)



System Context View: Software system X



Container View: Software system X

Stage 2: 💵 💵
(microservices)

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

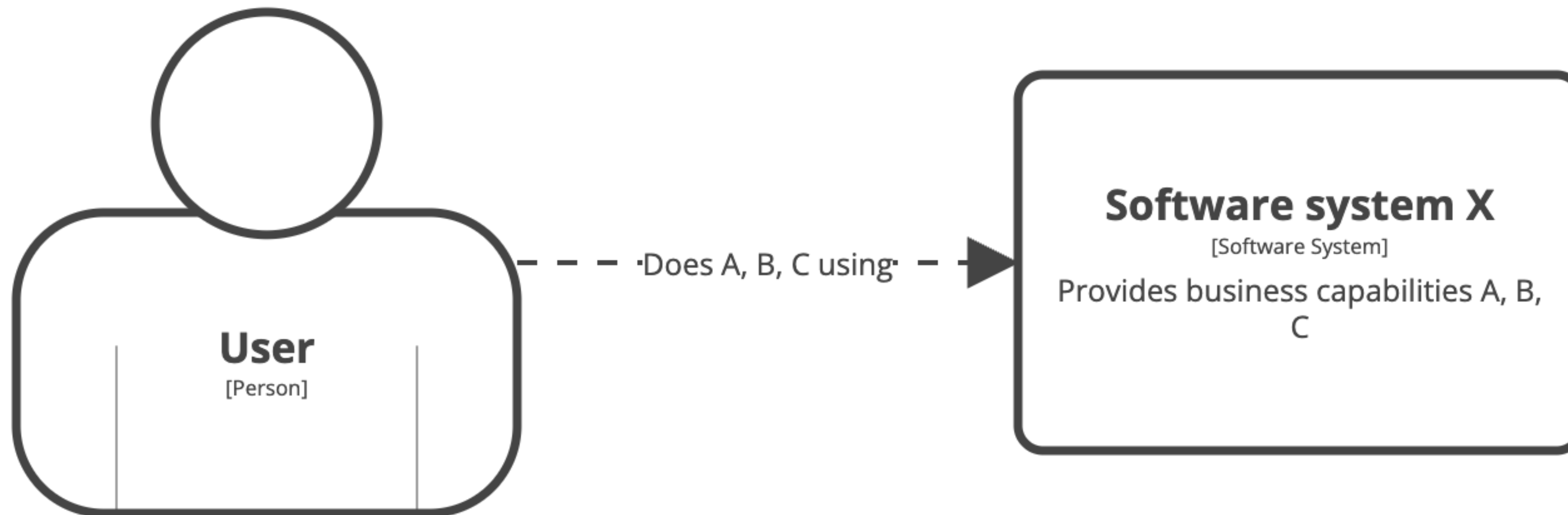
James Lewis

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

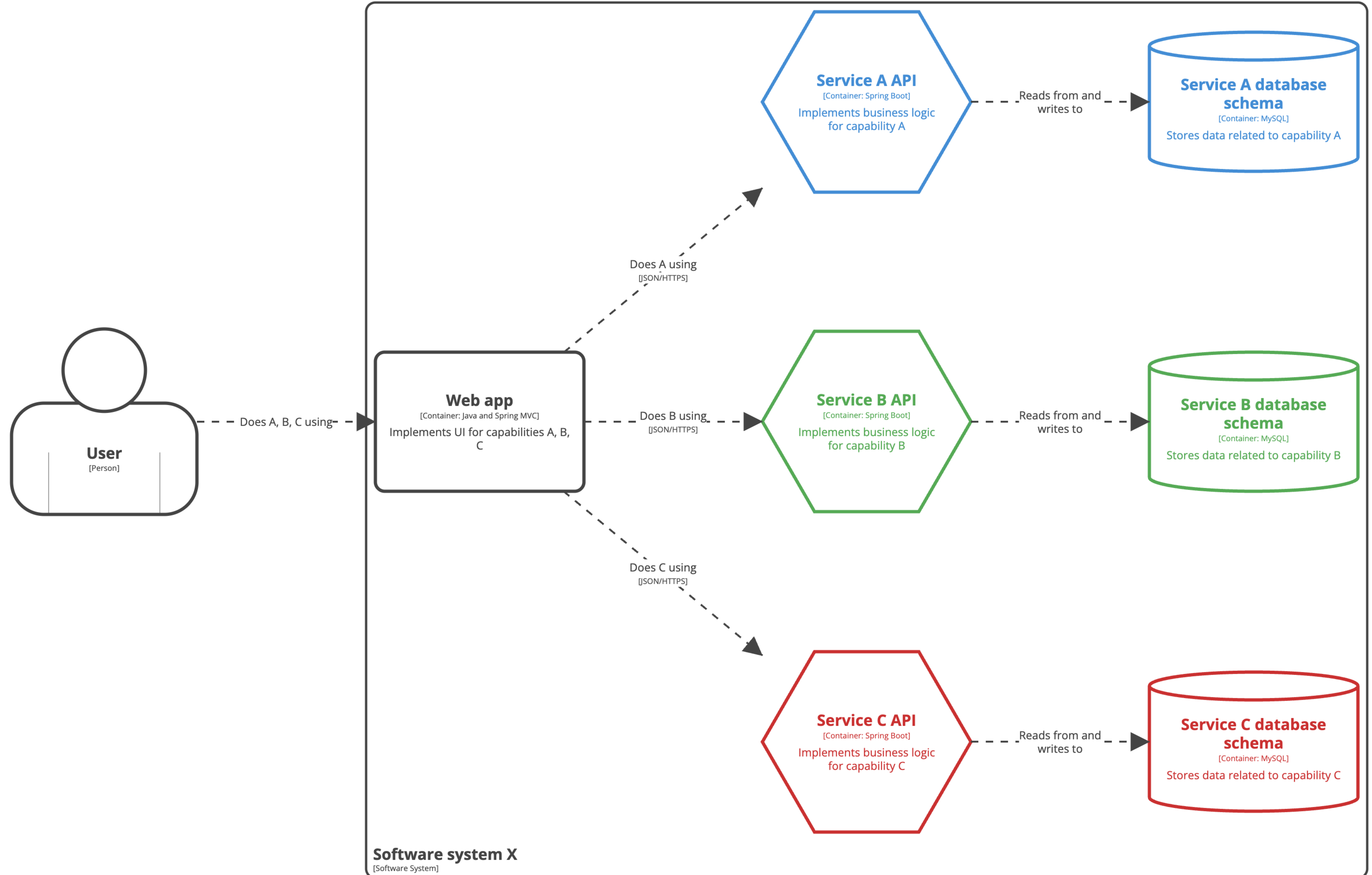
CONTENTS

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

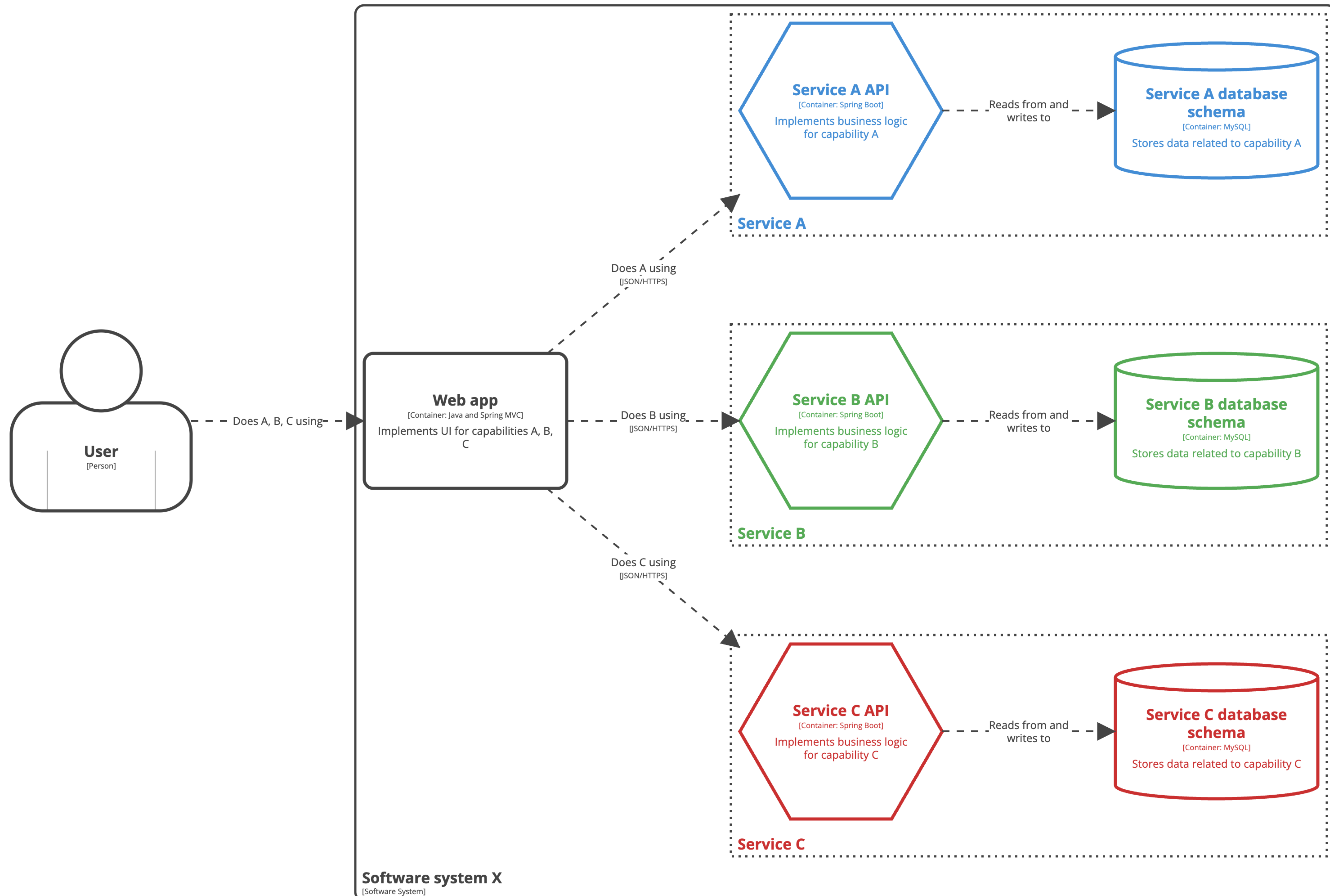
In short, the microservice architectural style [1] is an approach to developing a single software system as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.



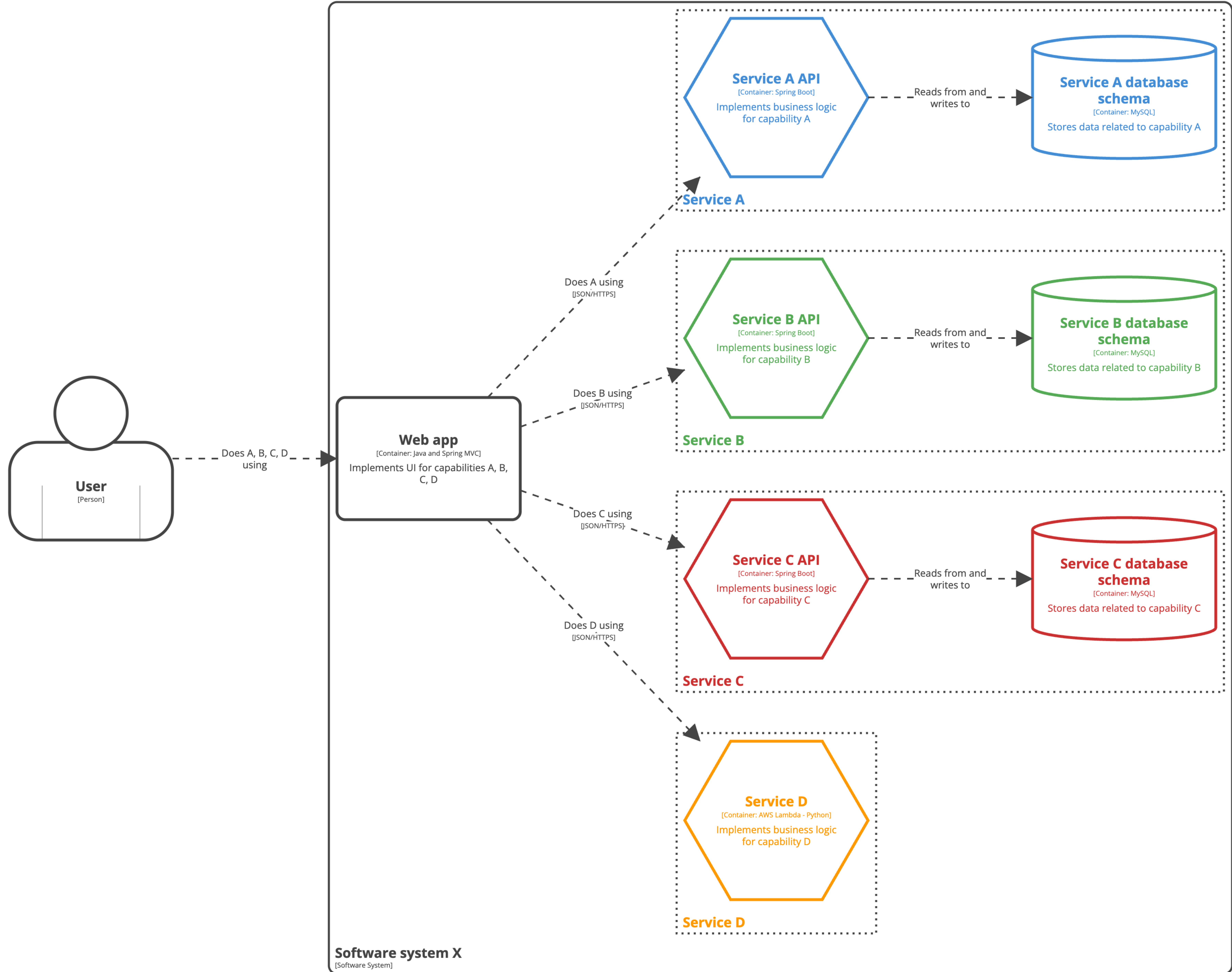
System Context View: Software system X



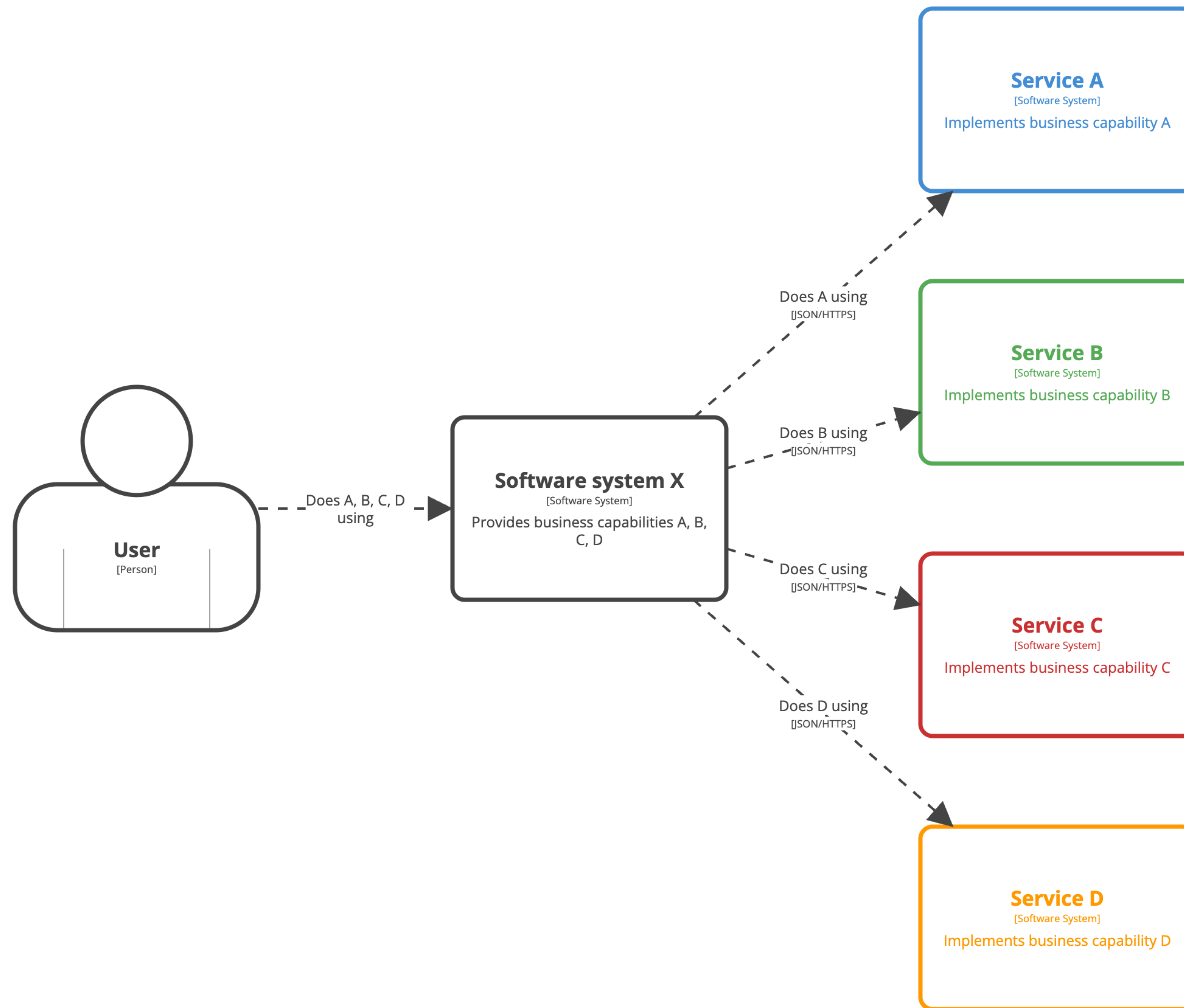
User
[Person]
Does A, B, C using
Web app
[Container: Java and Spring MVC]
Implements UI for capabilities A, B, C
Does A using
[JSON/HTTPS]
Does B using
[JSON/HTTPS]
Does C using
[JSON/HTTPS]
Service A API
[Container: Spring Boot]
Implements business logic for capability A
Service B API
[Container: Spring Boot]
Implements business logic for capability B
Service C API
[Container: Spring Boot]
Implements business logic for capability C
Reads from and writes to
Reads from and writes to
Reads from and writes to
Service A database schema
[Container: MySQL]
Stores data related to capability A
Service B database schema
[Container: MySQL]
Stores data related to capability B
Service C database schema
[Container: MySQL]
Stores data related to capability C
Software system X
[Software System]



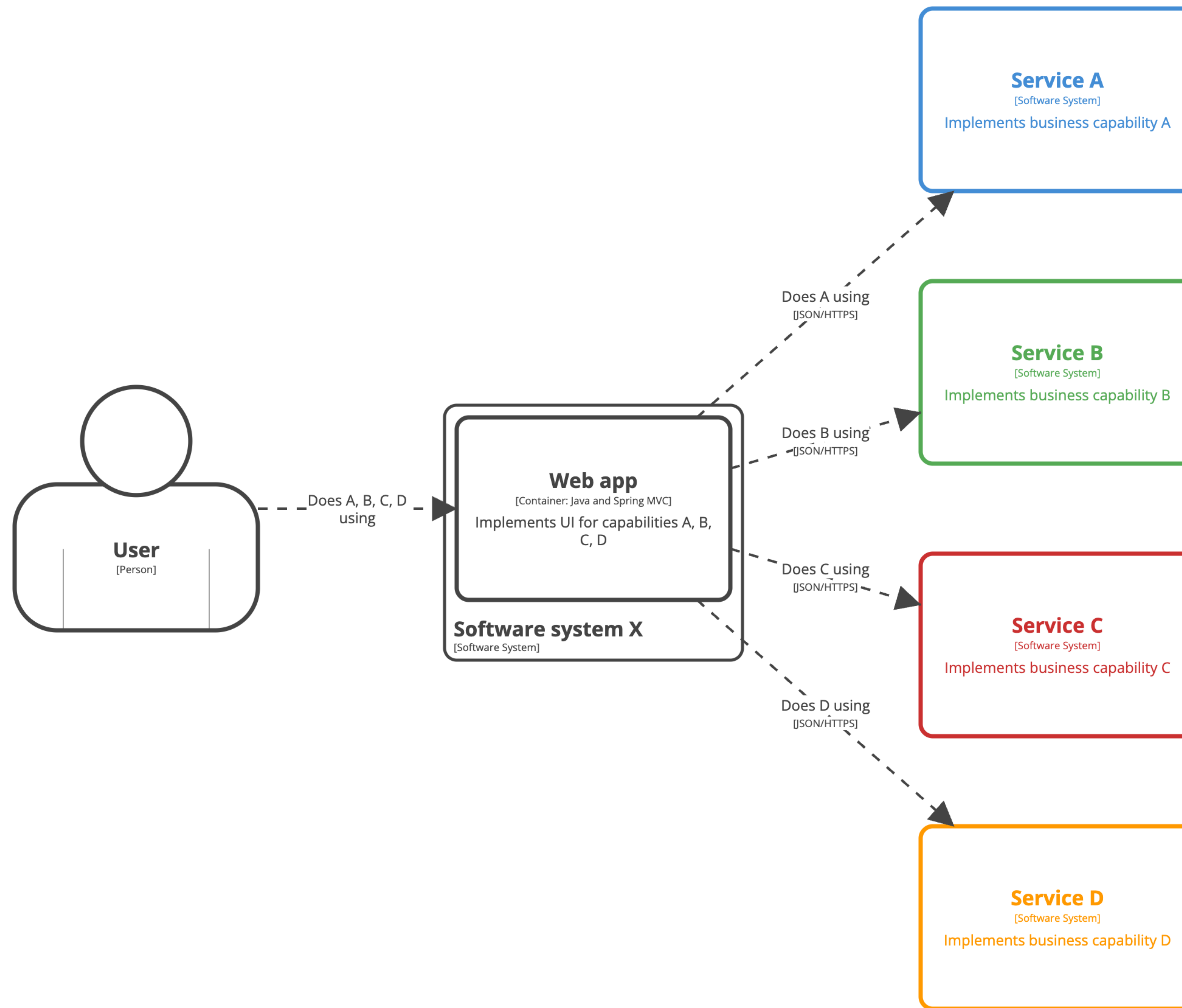
Service A API
[Container: Spring Boot]
Implements business logic for capability A
Reads from and writes to
Service A database schema
[Container: MySQL]
Stores data related to capability A
Service A
Does A using
[JSON/HTTPS]
User
[Person]
Does A, B, C using
Web app
[Container: Java and Spring MVC]
Implements UI for capabilities A, B, C
Does B using
[JSON/HTTPS]
Service B API
[Container: Spring Boot]
Implements business logic for capability B
Reads from and writes to
Service B database schema
[Container: MySQL]
Stores data related to capability B
Service B
Does C using
[JSON/HTTPS]
Service C API
[Container: Spring Boot]
Implements business logic for capability C
Reads from and writes to
Service C database schema
[Container: MySQL]
Stores data related to capability C
Service C
Software system X
[Software System]


User
[Person]
Does A, B, C, D using
Web app
[Container: Java and Spring MVC]
Implements UI for capabilities A, B, C, D
Does A using
[JSON/HTTPS]
Does B using
[JSON/HTTPS]
Does C using
[JSON/HTTPS]
Does D using
[JSON/HTTPS]
Service A API
[Container: Spring Boot]
Implements business logic for capability A
Reads from and writes to
Service A database schema
[Container: MySQL]
Stores data related to capability A
Service A
Service B API
[Container: Spring Boot]
Implements business logic for capability B
Reads from and writes to
Service B database schema
[Container: MySQL]
Stores data related to capability B
Service B
Service C API
[Container: Spring Boot]
Implements business logic for capability C
Reads from and writes to
Service C database schema
[Container: MySQL]
Stores data related to capability C
Service C
Service D
[Container: AWS Lambda - Python]
Implements business logic for capability D
Service D
Software system X
[Software System]

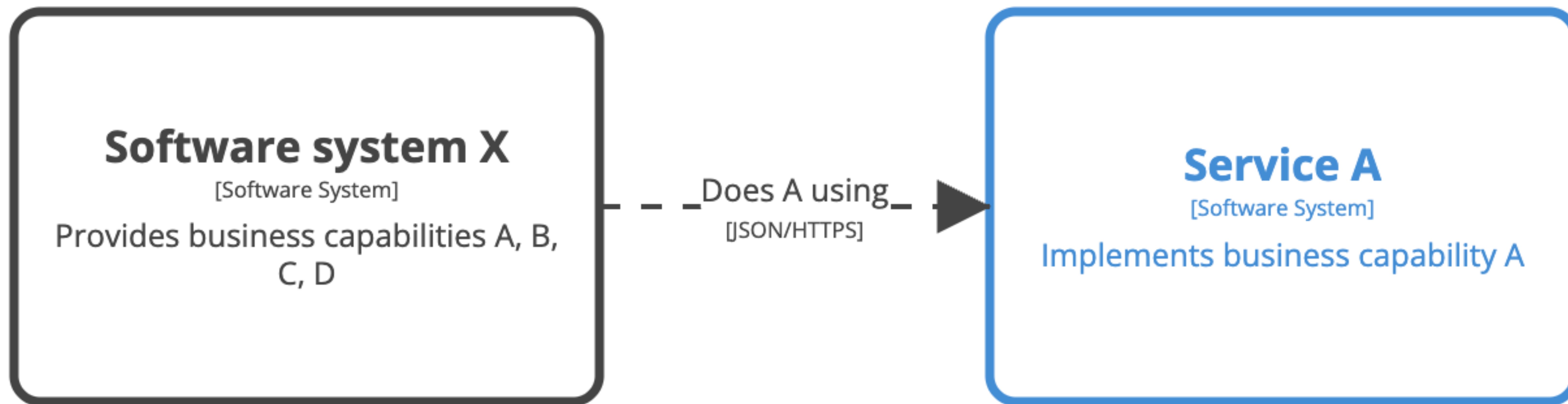
(Conway’s Law)



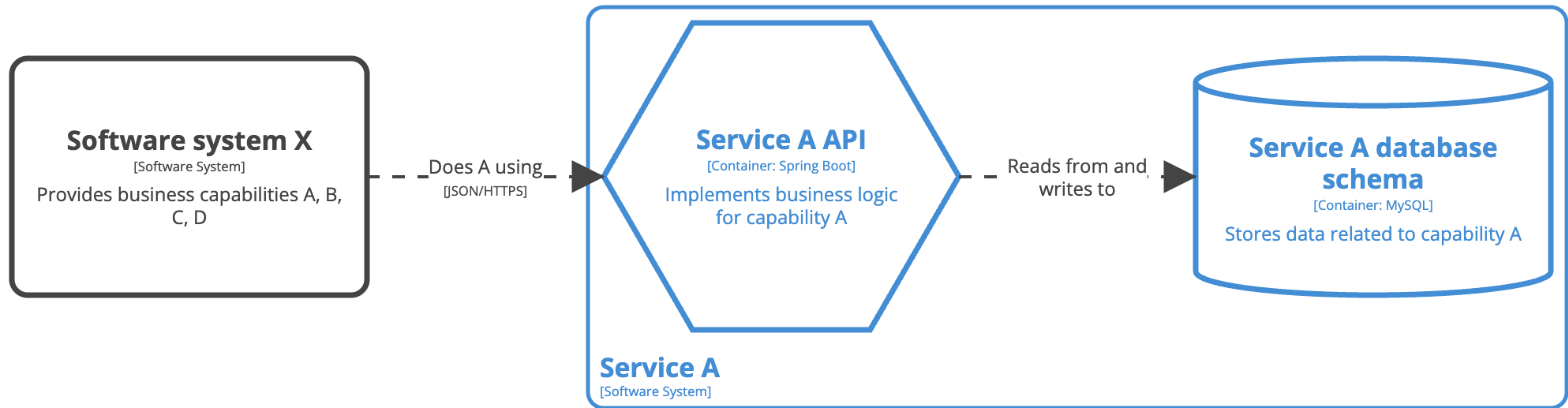
System Context View: Software system X



Container View: Software system X



System Context View: Service A

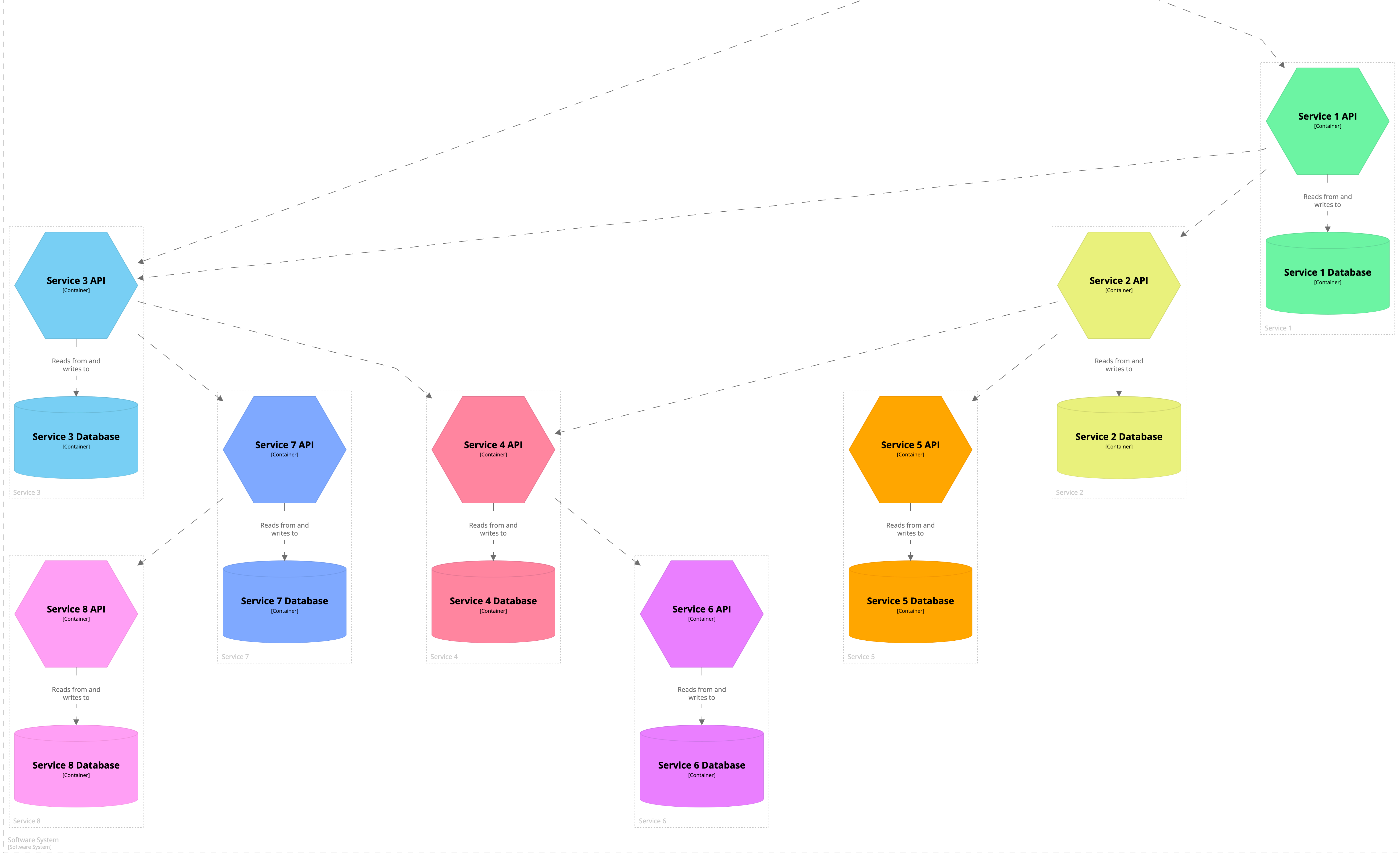


Container View: Service A

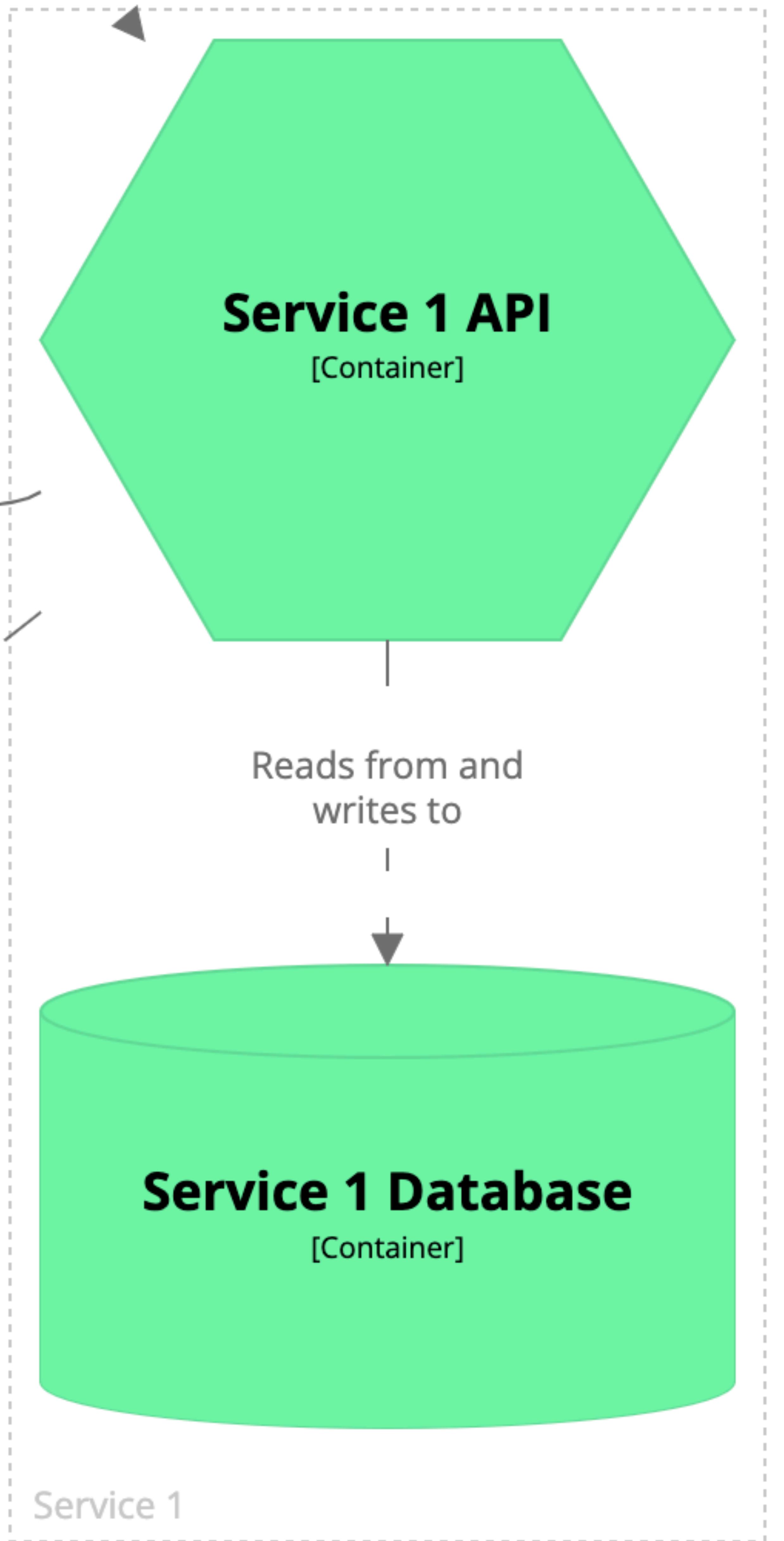
Service A API
[Container: Spring Boot]
Implements business logic for capability A
Reads from and writes to
Service A database schema
[Container: MySQL]
Stores data related to capability A
Service A
[Software System]
Does A using
[JSON/HTTPS]
Service B API
[Container: Spring Boot]
Implements business logic for capability B
Reads from and writes to
Service B database
[Container: MySQL]
Stores data related to capability B
Service B
[Software System]
Does B using
[JSON/HTTPS]
Does A, B, C, D using
Web app
[Container: Java and Spring MVC]
Implements UI for capabilities A, B, C, D
Software system X
[Software System]
Does C using
[JSON/HTTPS]
Service C API
[Container: Spring Boot]
Implements business logic
Reads from and writes to
Service C database
[Container: MySQL]

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

6. The C4 model at scale

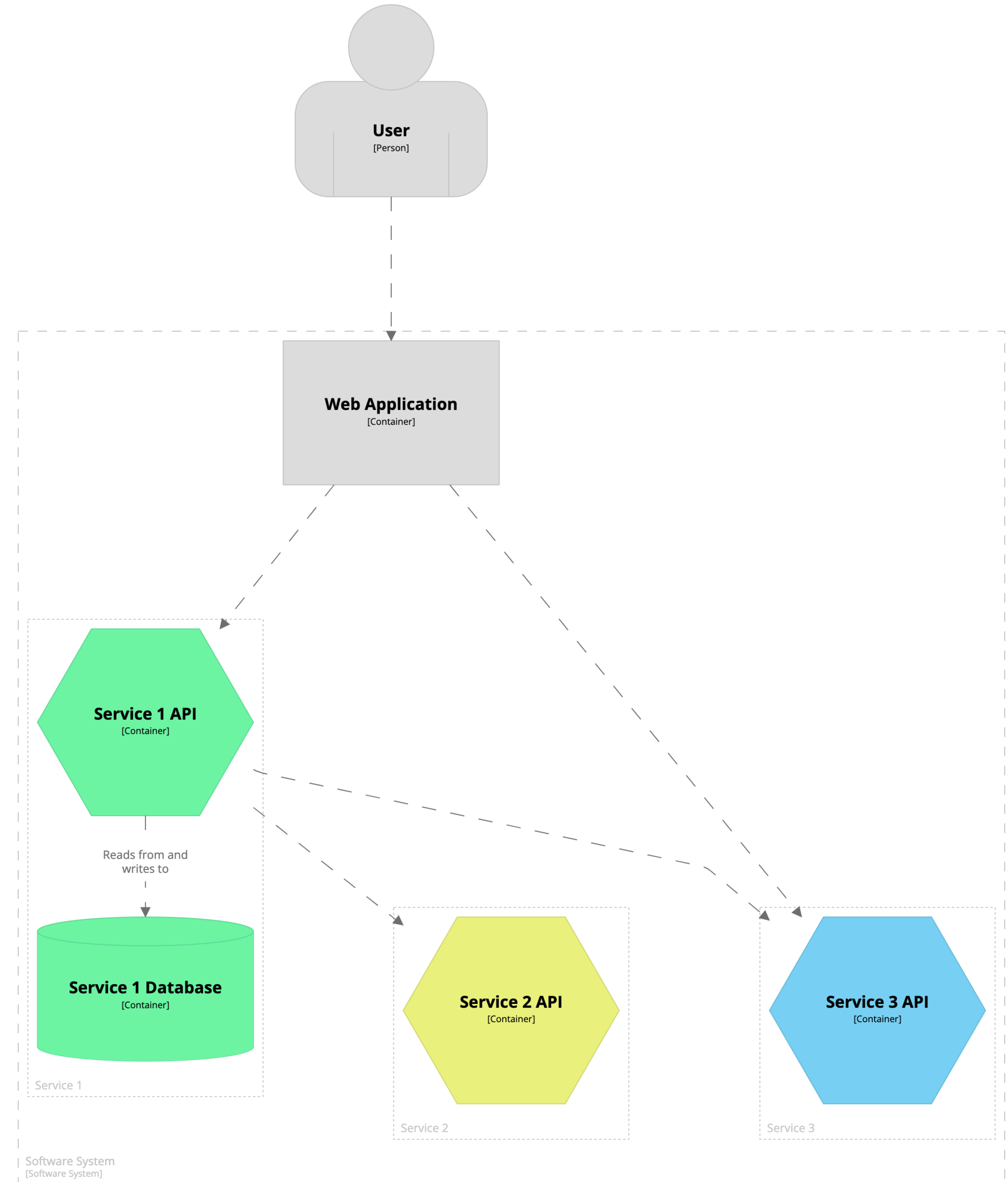


Service 1 API
[Container]
Reads from and writes to
Service 1 Database
[Container]
Service 1
Service 3 API
[Container]
Reads from and writes to
Service 3 Database
[Container]
Service 3
Service 2 API
[Container]
Reads from and writes to
Service 2 Database
[Container]
Service 2
Service 7 API
[Container]
Reads from and writes to
Service 7 Database
[Container]
Service 7
Service 4 API
[Container]
Reads from and writes to
Service 4 Database
[Container]
Service 4
Service 5 API
[Container]
Reads from and writes to
Service 5 Database
[Container]
Service 5
Service 8 API
[Container]
Reads from and writes to
Service 8 Database
[Container]
Service 8
Service 6 API
[Container]
Reads from and writes to
Service 6 Database
[Container]
Service 6
Software System
[Software System]

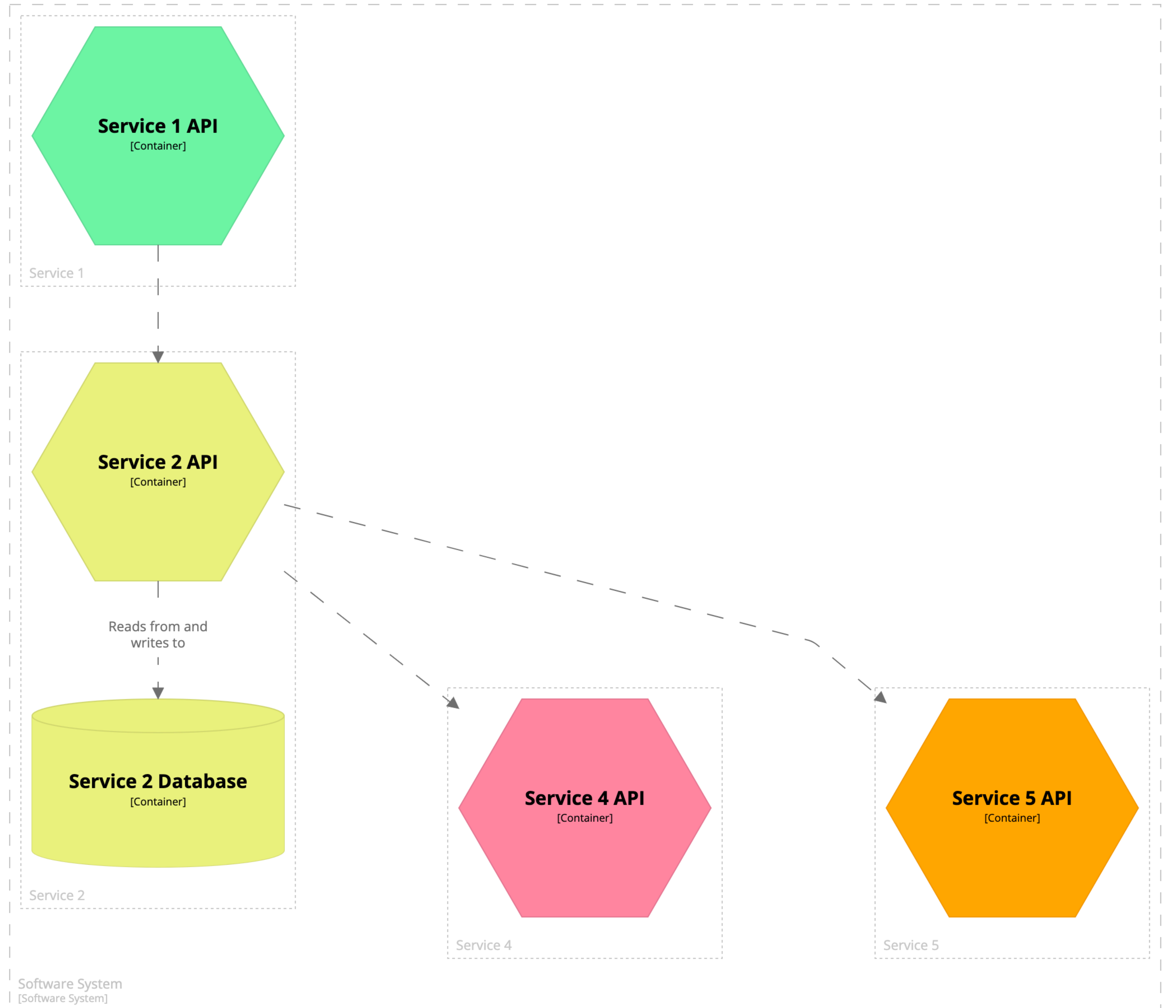


In this example,
a microservice is
a combination of
an API and
a database schema

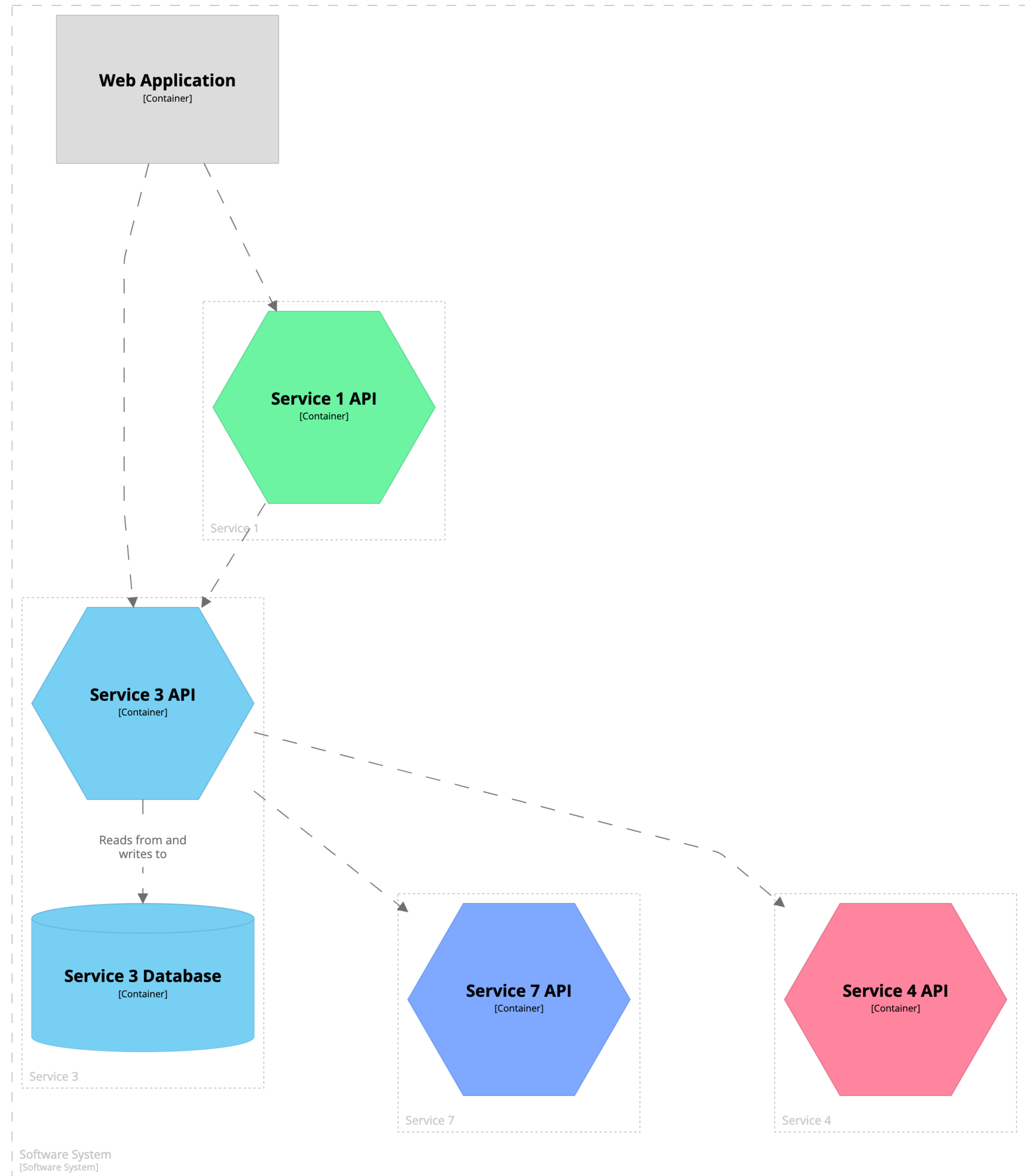
```
container softwareSystem {
    include user
    include ->service1->
}
```

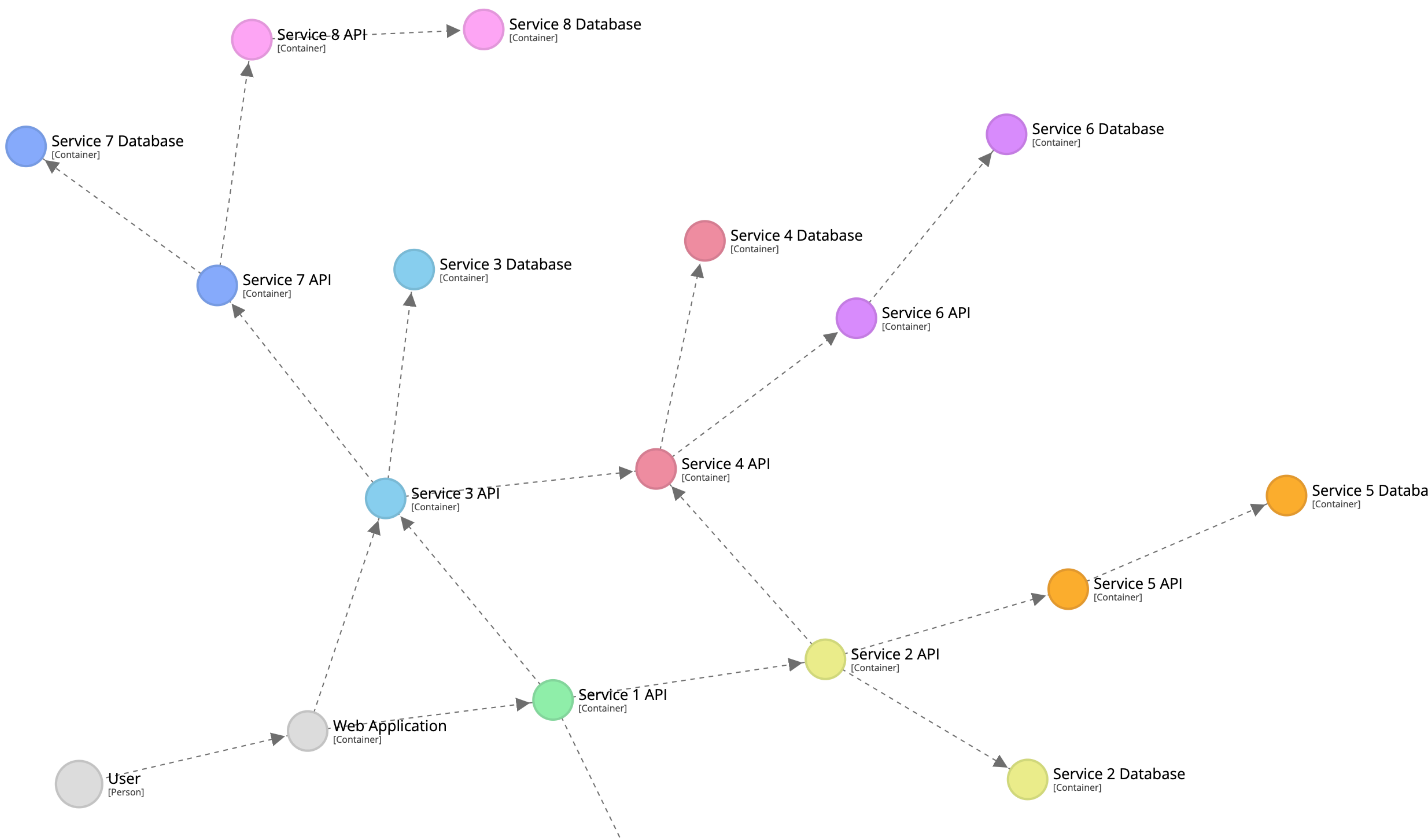


```
container softwareSystem {
    include ->service2->
}
```

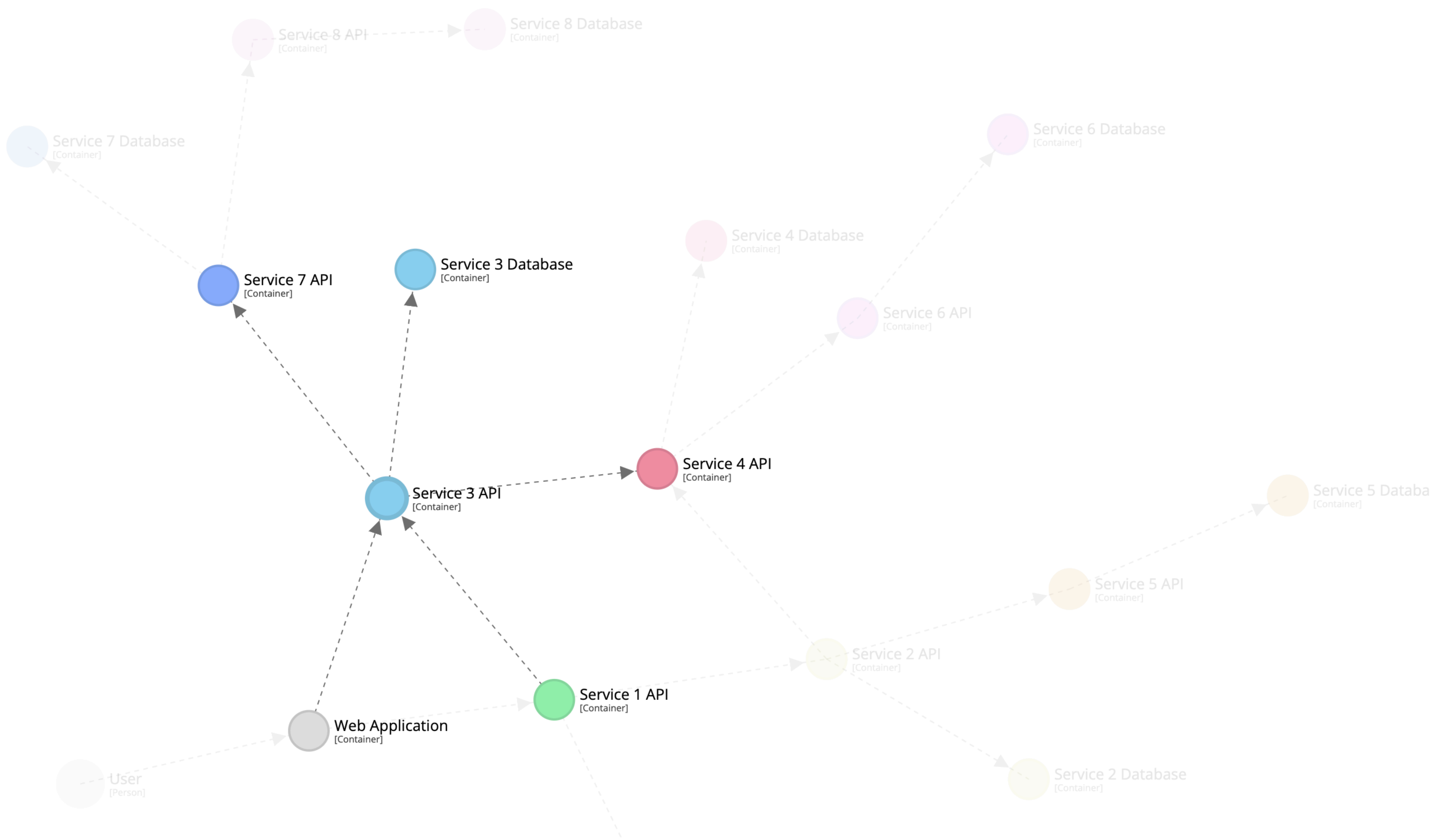


```
container softwareSystem {
    include ->service3->
}
```

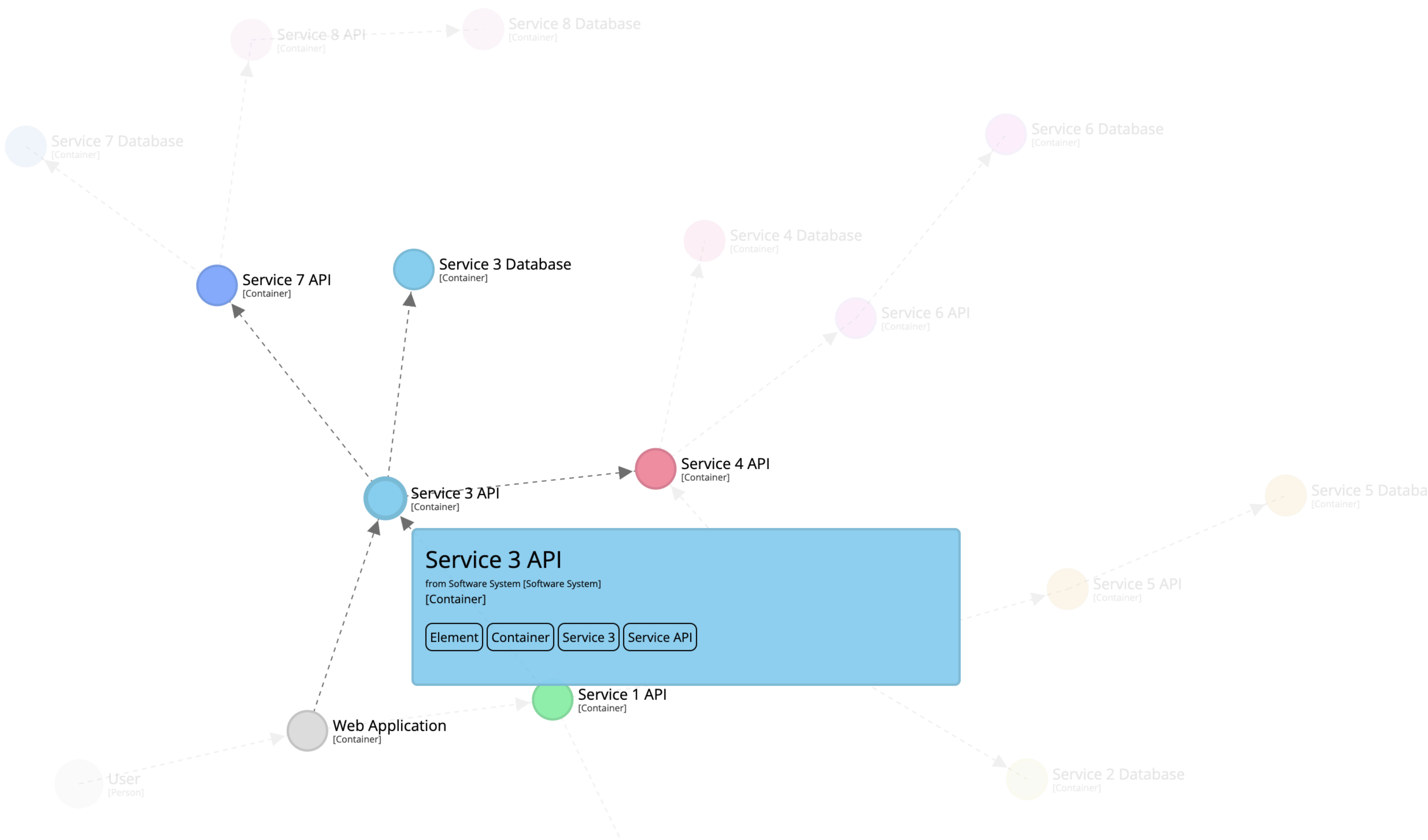




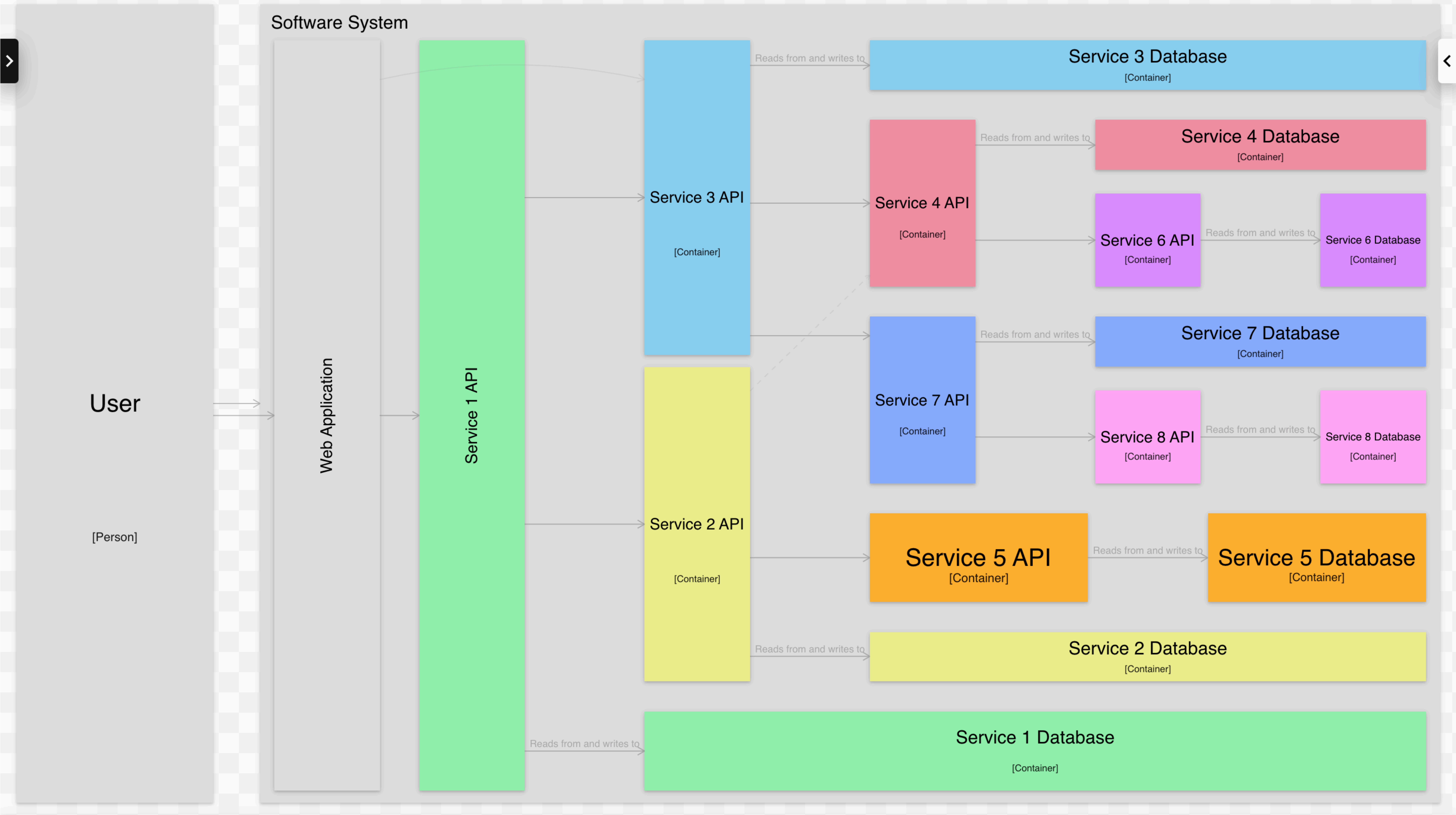
Service 8 API
[Container]
Service 8 Database
[Container]
Service 7 Database
[Container]
Service 6 Database
[Container]
Service 4 Database
[Container]
Service 3 Database
[Container]
Service 7 API
[Container]
Service 6 API
[Container]
Service 4 API
[Container]
Service 3 API
[Container]
Service 5 Databa
[Container]
Service 5 API
[Container]
Service 2 API
[Container]
Service 1 API
[Container]
Web Application
[Container]
User
[Person]
Service 2 Database
[Container]

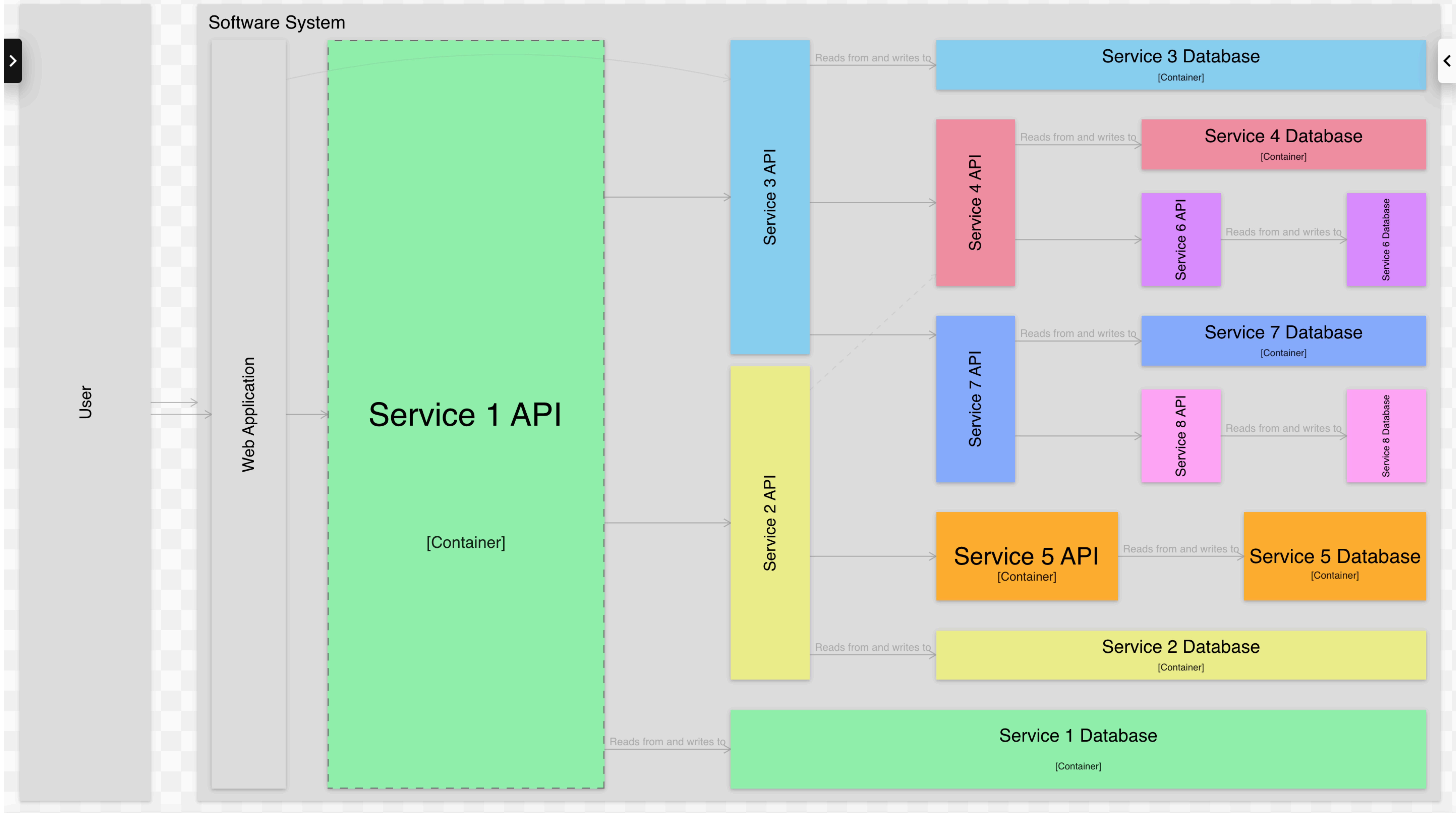


Service 8 API
[Container]
Service 8 Database
[Container]
Service 7 Database
[Container]
Service 6 Database
[Container]
Service 4 Database
[Container]
Service 7 API
[Container]
Service 3 Database
[Container]
Service 6 API
[Container]
Service 4 API
[Container]
Service 3 API
[Container]
Service 5 Databa
[Container]
Service 5 API
[Container]
Service 2 API
[Container]
Service 1 API
[Container]
Web Application
[Container]
User
[Person]
Service 2 Database
[Container]

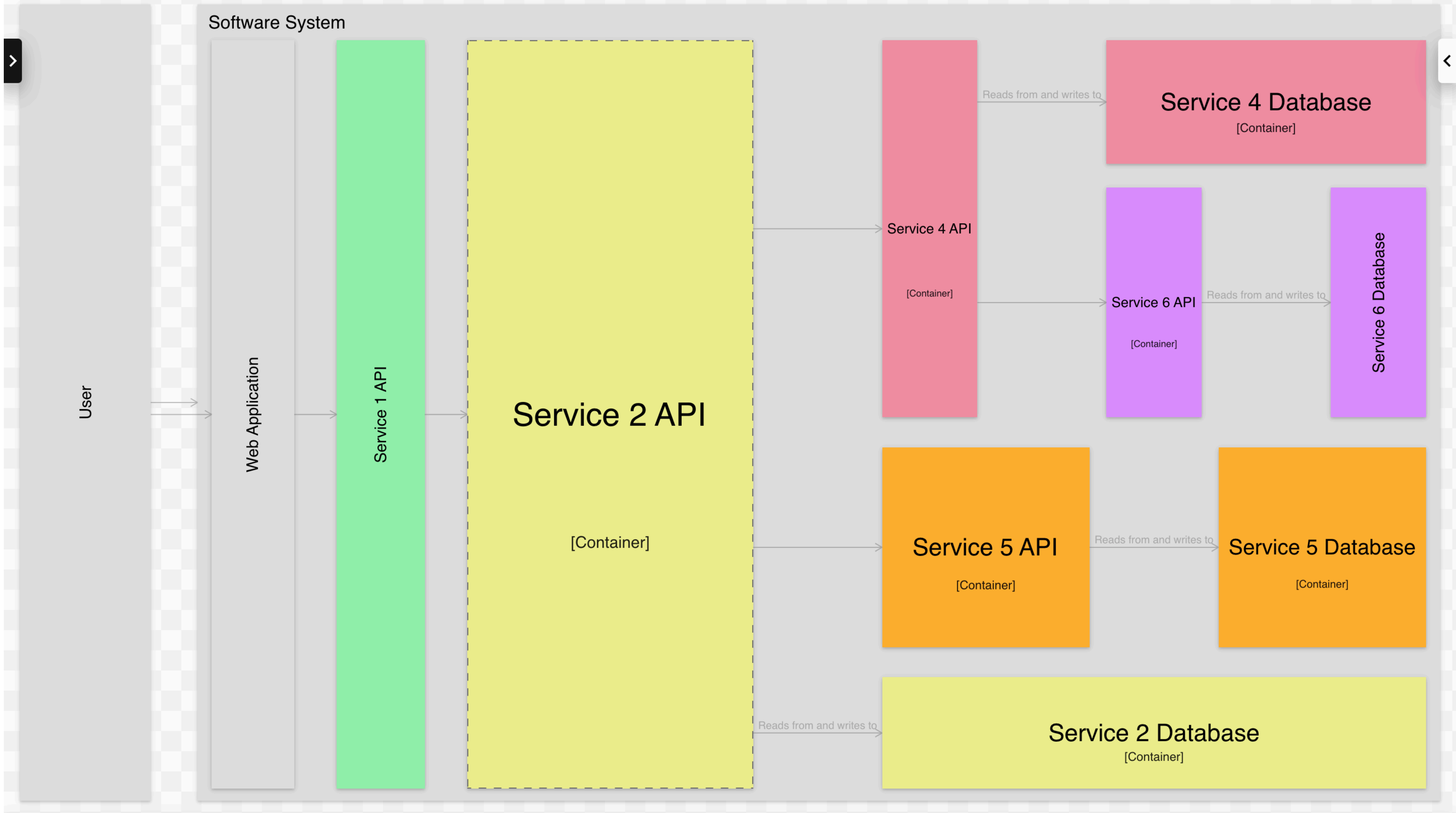


Service 8 API
[Container]
Service 8 Database
[Container]
Service 7 Database
[Container]
Service 6 Database
[Container]
Service 4 Database
[Container]
Service 6 API
[Container]
Service 7 API
[Container]
Service 3 Database
[Container]
Service 4 API
[Container]
Service 3 API
[Container]
Service 5 Databa
[Container]
Service 3 API
from Software System [Software System]
[Container]
Element
Container
Service 3
Service API
Service 5 API
[Container]
Service 1 API
[Container]
Web Application
[Container]
User
[Person]
Service 2 Database
[Container]

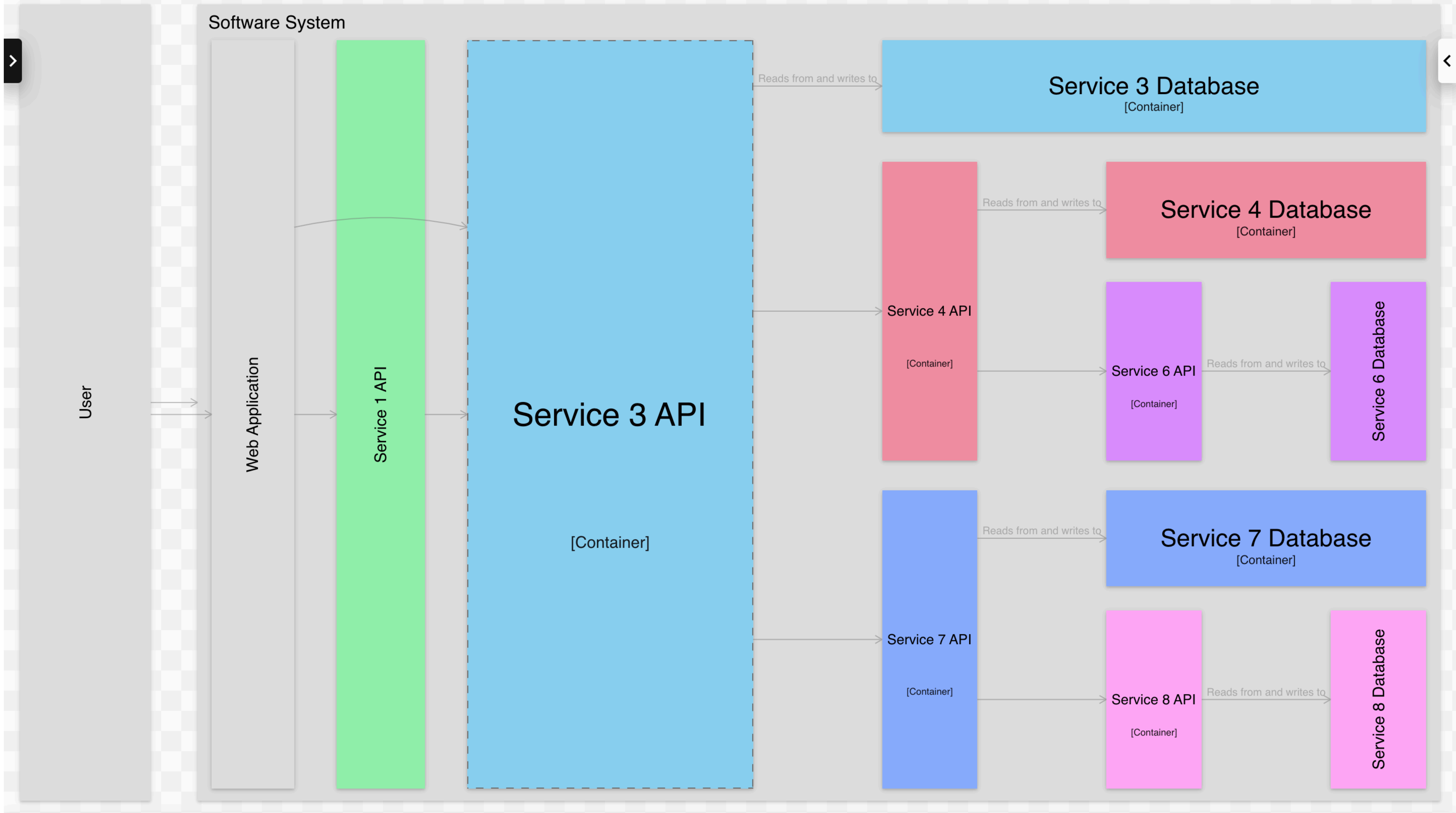

New Diagram > Static Structure Perspective
Ilograph for Desktop
Software System
User
[Person]
Web Application
Service 1 API
Service 3 API
[Container]
Reads from and writes to
Service 3 Database
[Container]
Service 4 API
[Container]
Reads from and writes to
Service 4 Database
[Container]
Service 6 API
[Container]
Reads from and writes to
Service 6 Database
[Container]
Service 7 API
[Container]
Reads from and writes to
Service 7 Database
[Container]
Service 8 API
[Container]
Reads from and writes to
Service 8 Database
[Container]
Service 2 API
[Container]
Service 5 API
[Container]
Reads from and writes to
Service 5 Database
[Container]
Reads from and writes to
Service 2 Database
[Container]
Reads from and writes to
Service 1 Database
[Container]
Static Structure



New Diagram > Static Structure Perspective > Service 1 API
Ilograph for Desktop
Software System
User
Web Application
Service 1 API
[Container]
Service 3 API
Service 2 API
Reads from and writes to
Service 3 Database
[Container]
Service 4 API
Service 4 Database
[Container]
Service 6 API
Service 6 Database
Service 7 API
Service 7 Database
[Container]
Service 8 API
Service 8 Database
Service 5 API
[Container]
Service 5 Database
[Container]
Service 2 Database
[Container]
Service 1 Database
[Container]
Static Structure

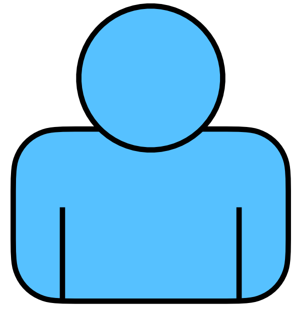
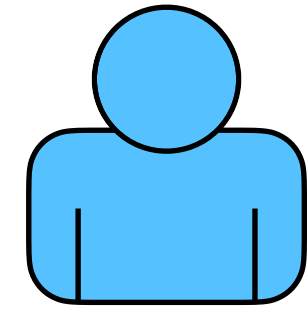


New Diagram > Static Structure Perspective > Service 2 API
Ilograph for Desktop
Software System
User
Web Application
Service 1 API
Service 2 API
[Container]
Service 4 API
[Container]
Reads from and writes to
Service 4 Database
[Container]
Service 6 API
[Container]
Reads from and writes to
Service 6 Database
Service 5 API
[Container]
Reads from and writes to
Service 5 Database
[Container]
Reads from and writes to
Service 2 Database
[Container]
Static Structure

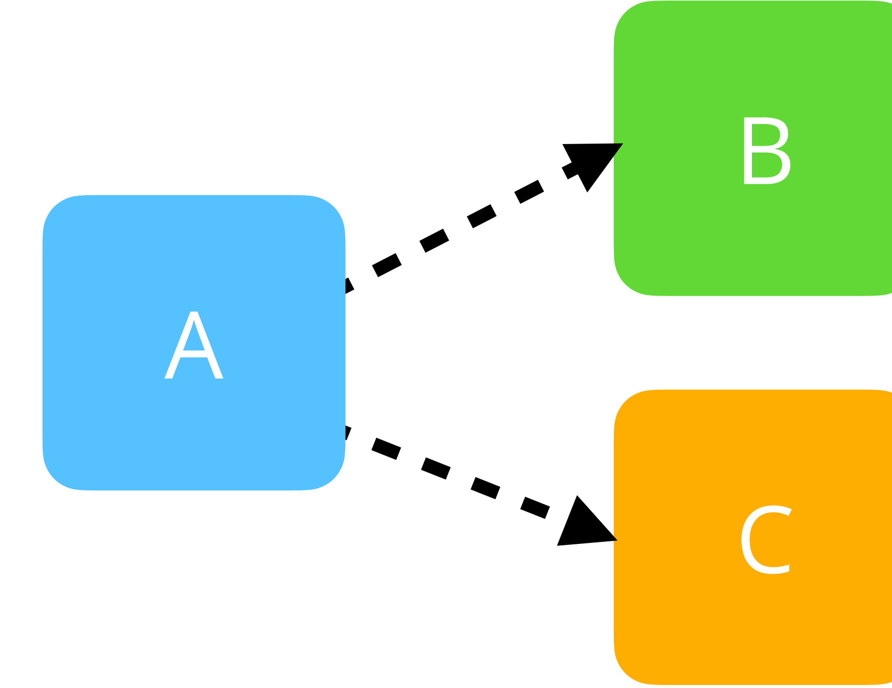


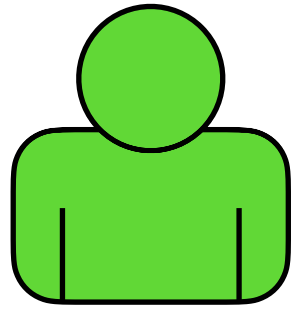
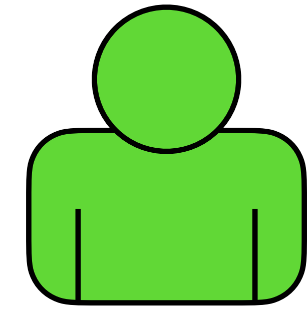
New Diagram > Static Structure Perspective > Service 3 API
Ilograph for Desktop
Software System
User
Web Application
Service 1 API
Service 3 API
[Container]
Reads from and writes to
Service 3 Database
[Container]
Service 4 API
[Container]
Reads from and writes to
Service 4 Database
[Container]
Service 6 API
[Container]
Reads from and writes to
Service 6 Database
Service 7 API
[Container]
Reads from and writes to
Service 7 Database
[Container]
Service 8 API
[Container]
Reads from and writes to
Service 8 Database
Static Structure

7. The C4 model in the enterprise

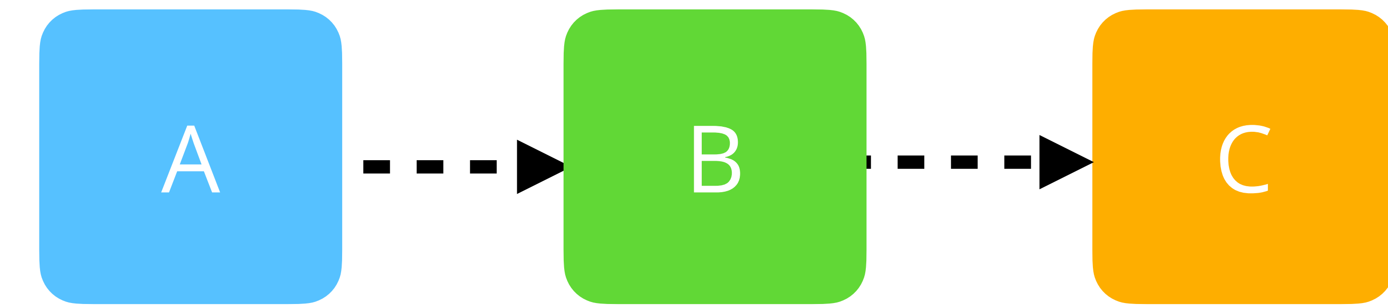


Team A

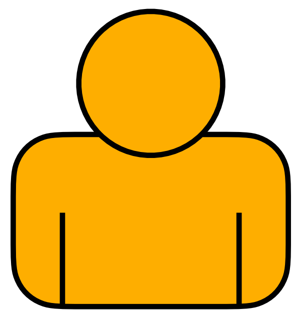
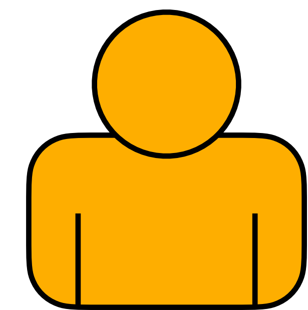
```
model {
  a = softwareSystem "A" {
    app = container "App"
  }
  b = softwareSystem "B"
  c = softwareSystem "C"
  a.app -> b
  a.app -> c
}
```



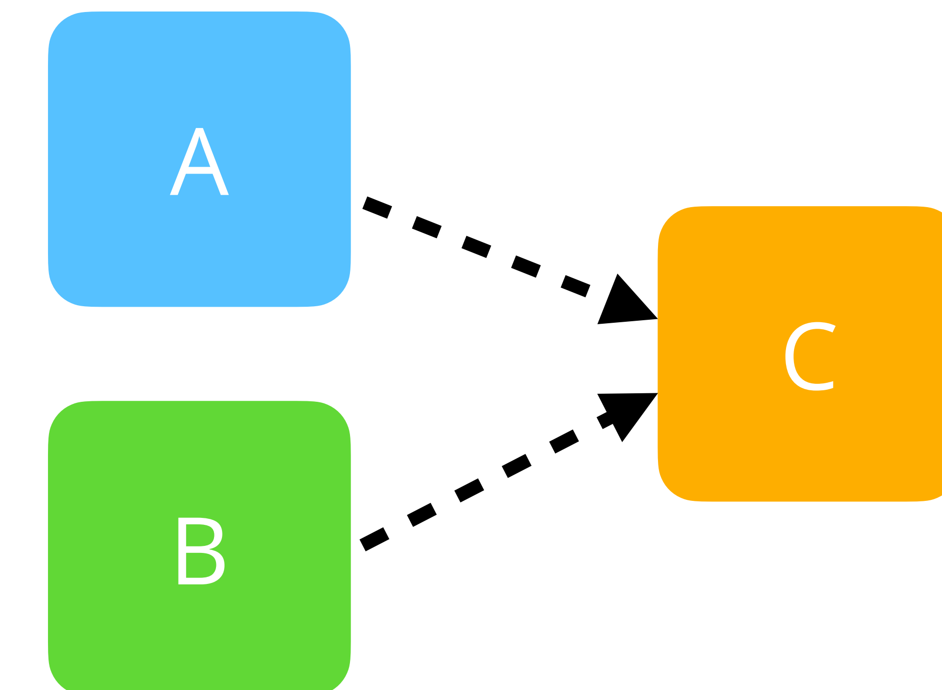


Team B

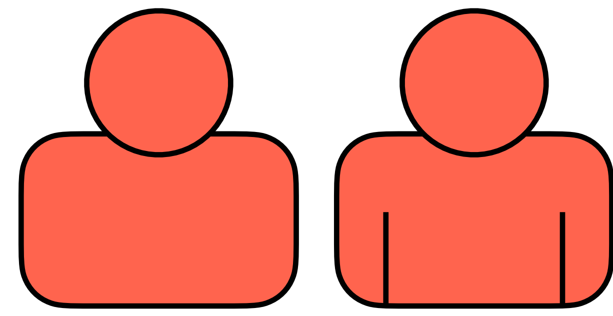
```
model {
  a = softwareSystem "A"
  b = softwareSystem "B" {
    api = container "API"
  }
  c = softwareSystem "C"
  a -> b.api
  b.api -> c
}
```





Team C

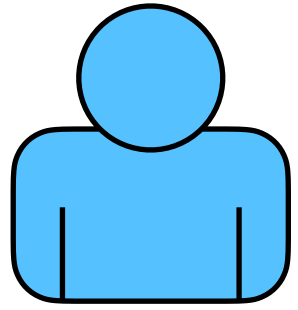
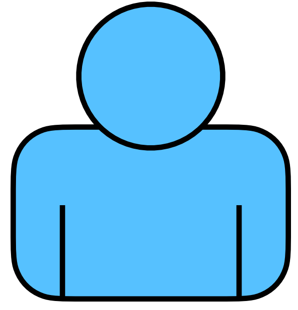
```
model {
  a = softwareSystem "A"
  b = softwareSystem "B"
  c = softwareSystem "C" {
    api = container "API"
  }
  a -> c.api
  b -> c.api
}
```

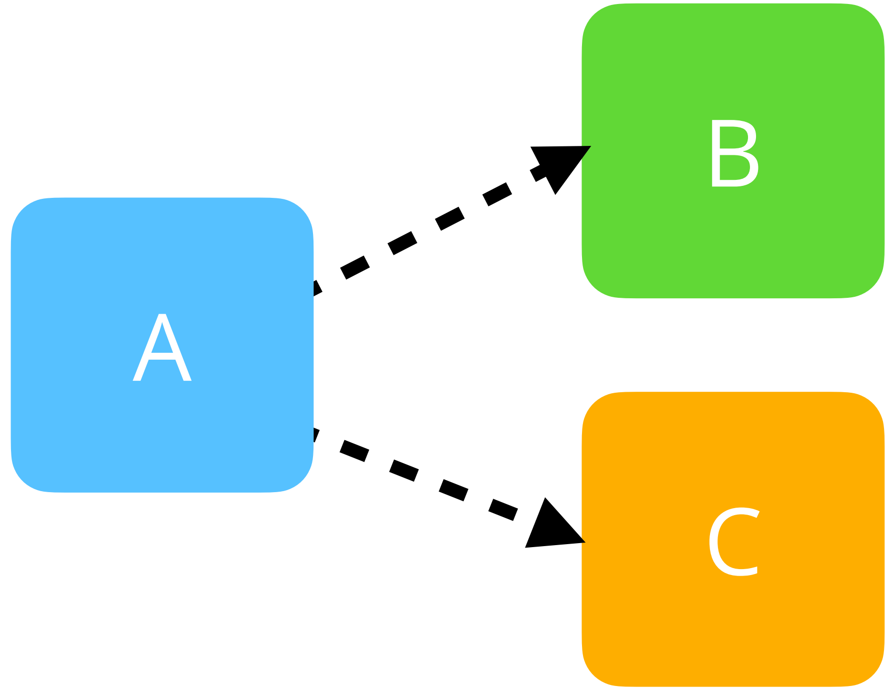


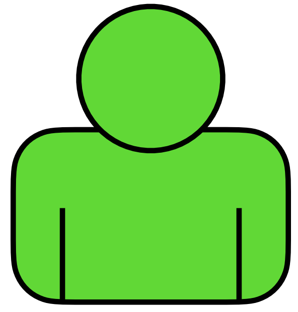
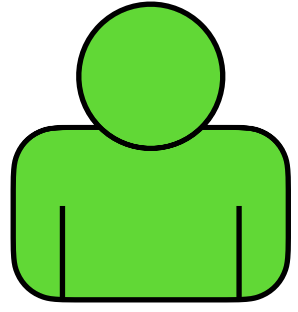


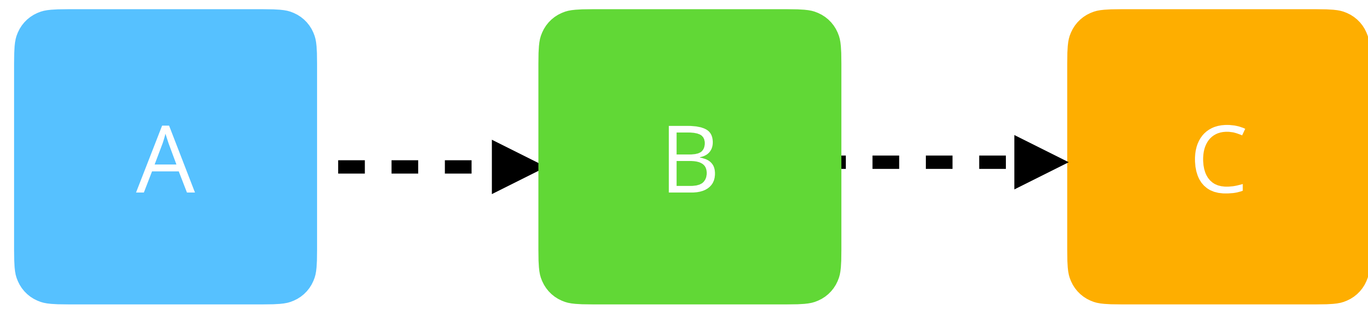
Enterprise architecture team

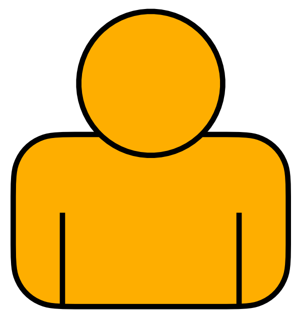
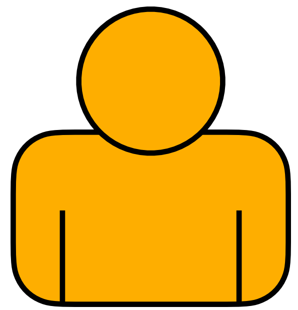
How can we generate a diagram showing all software systems in our landscape/portfolio and the relationships between them?

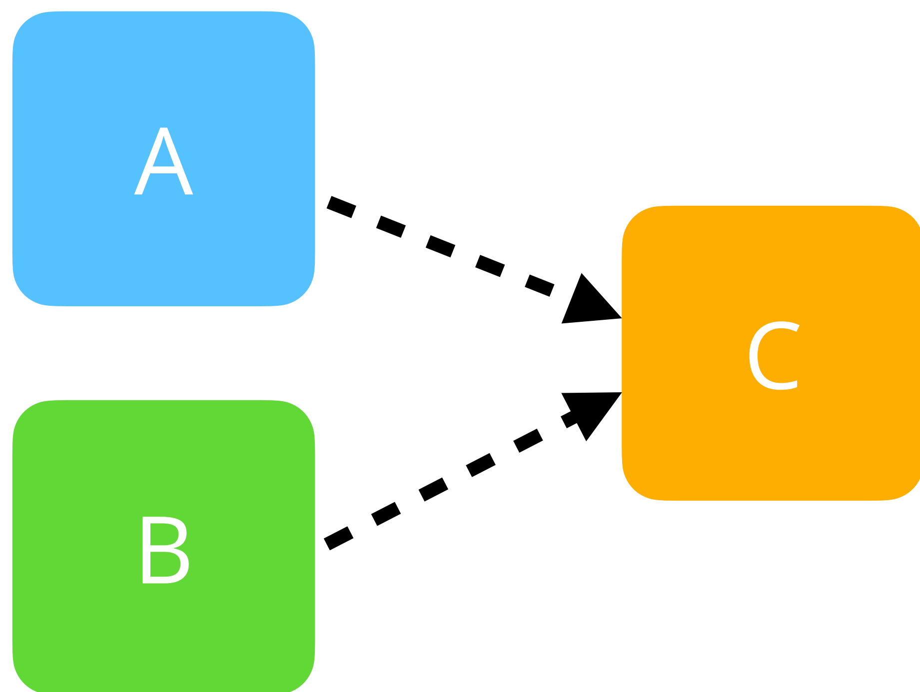


Team A

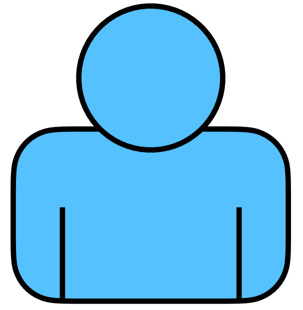
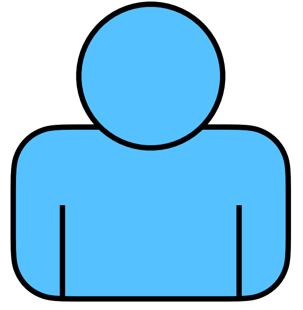




Team B

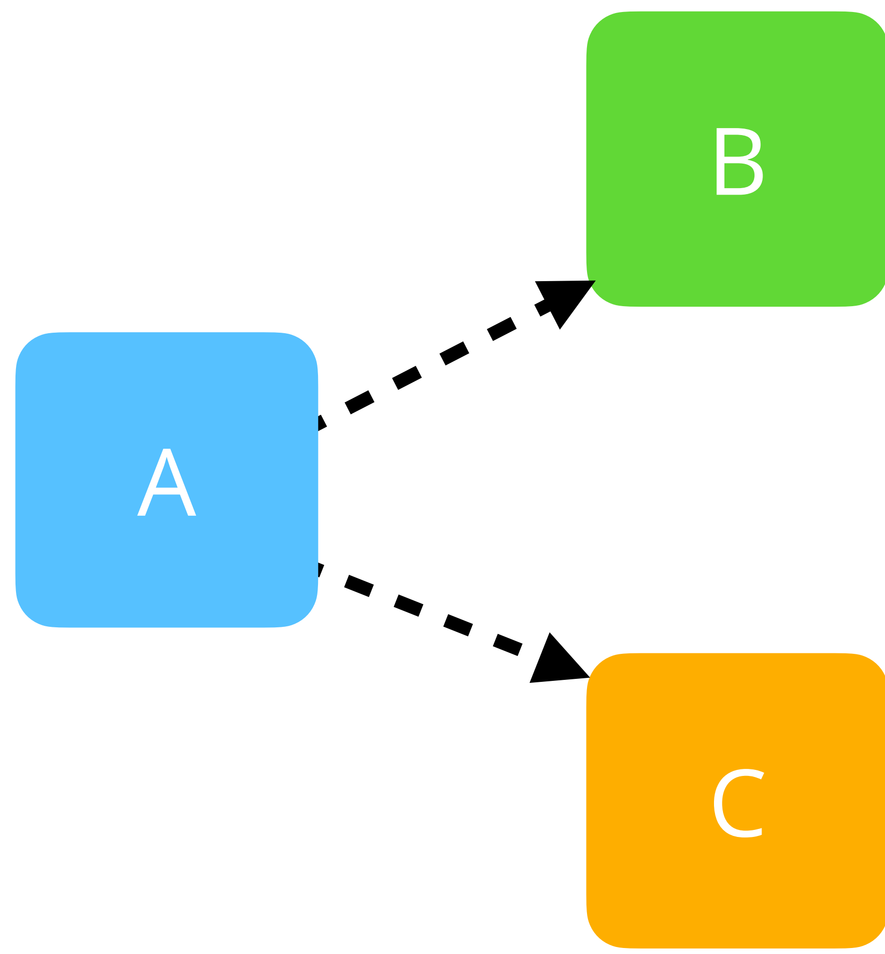


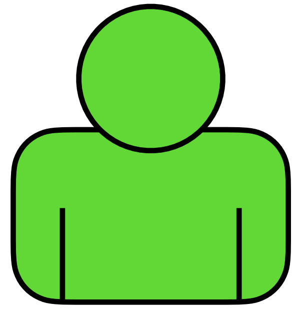
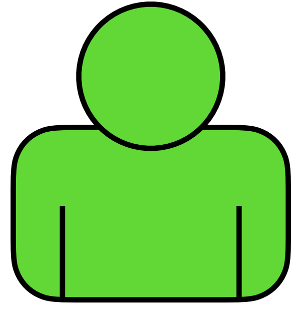


Team C



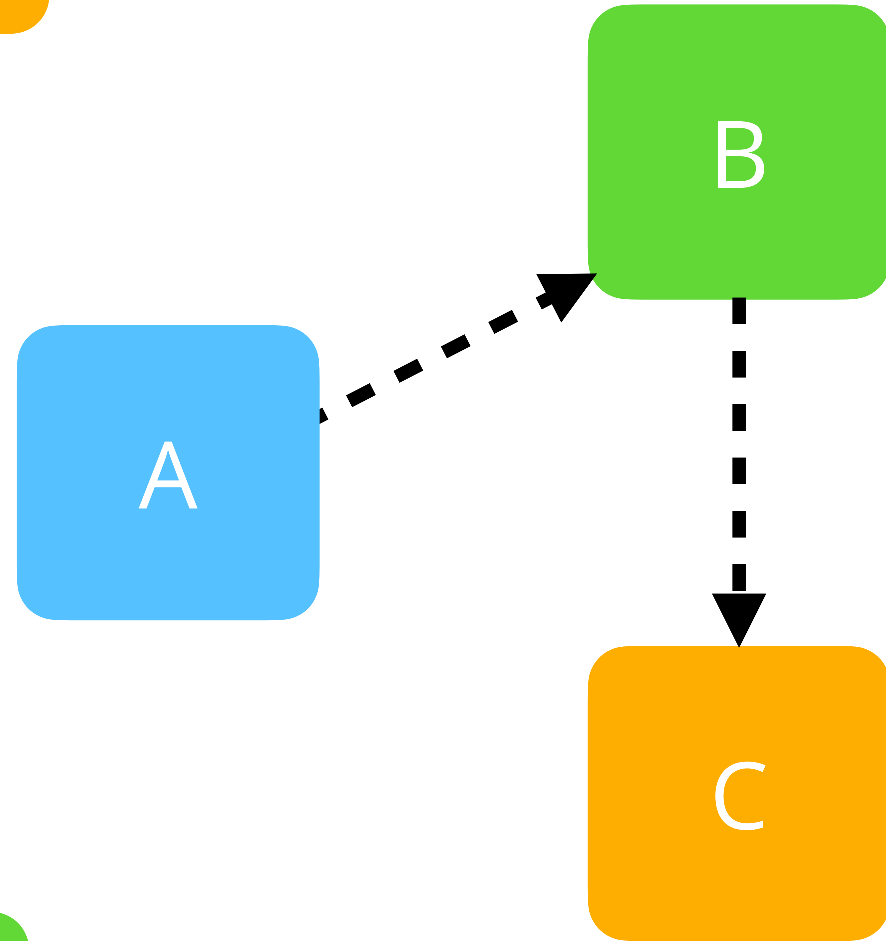



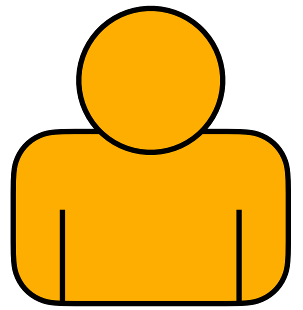
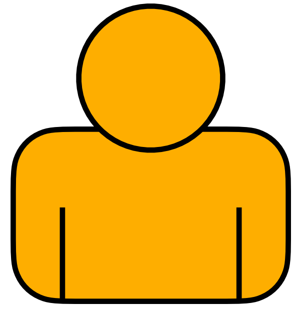
Team A



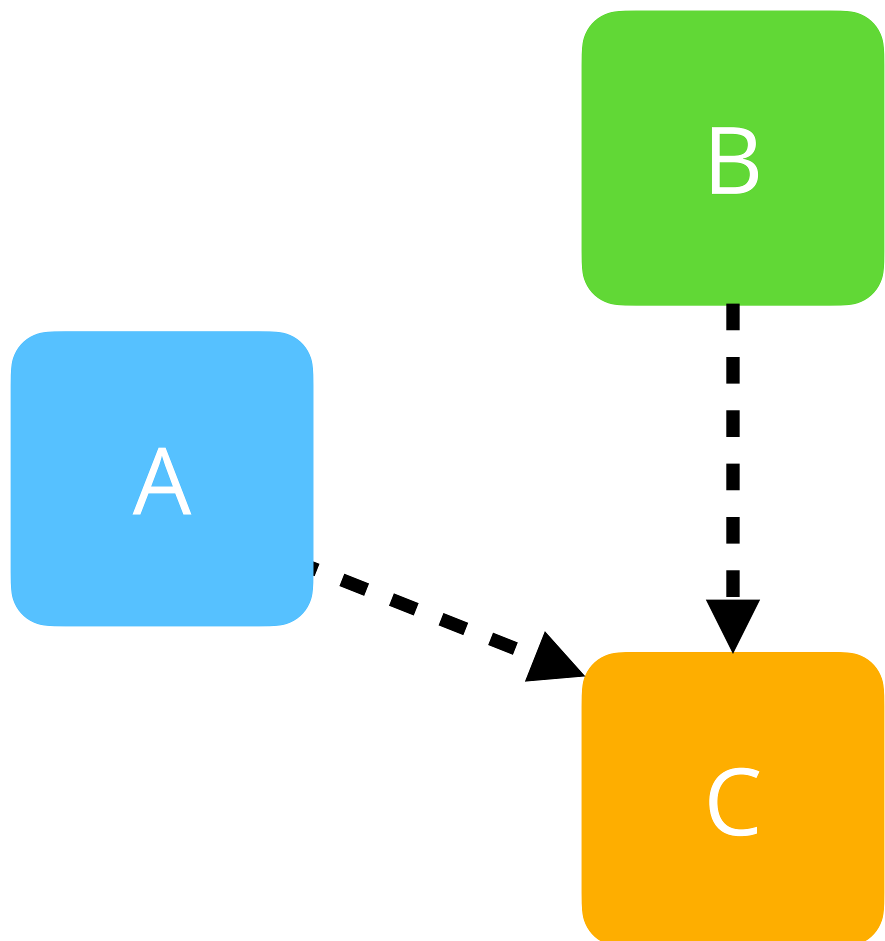



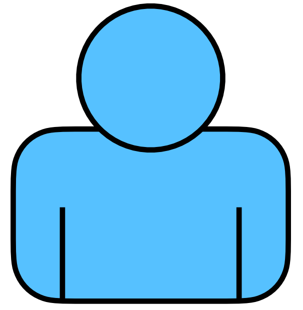
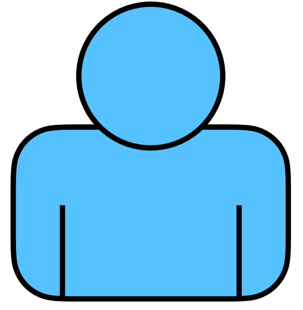
Team B



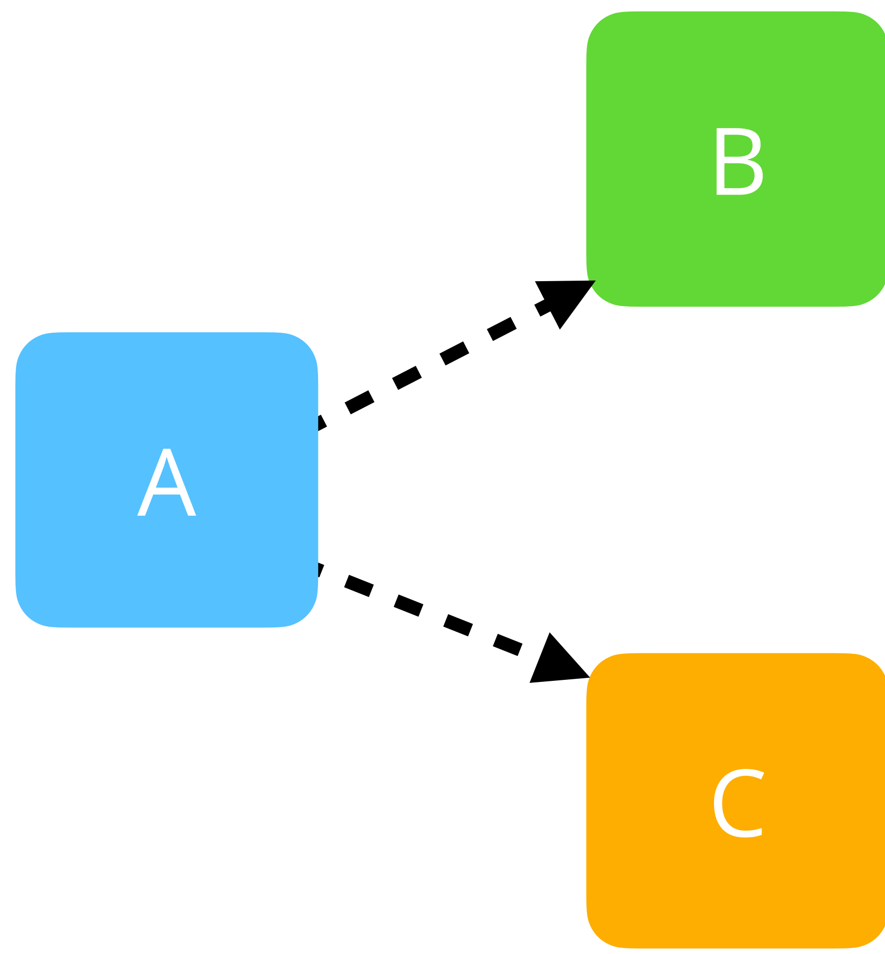



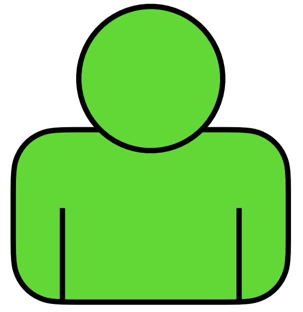
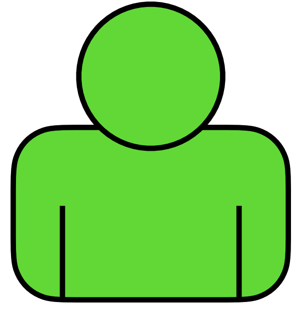
Team C



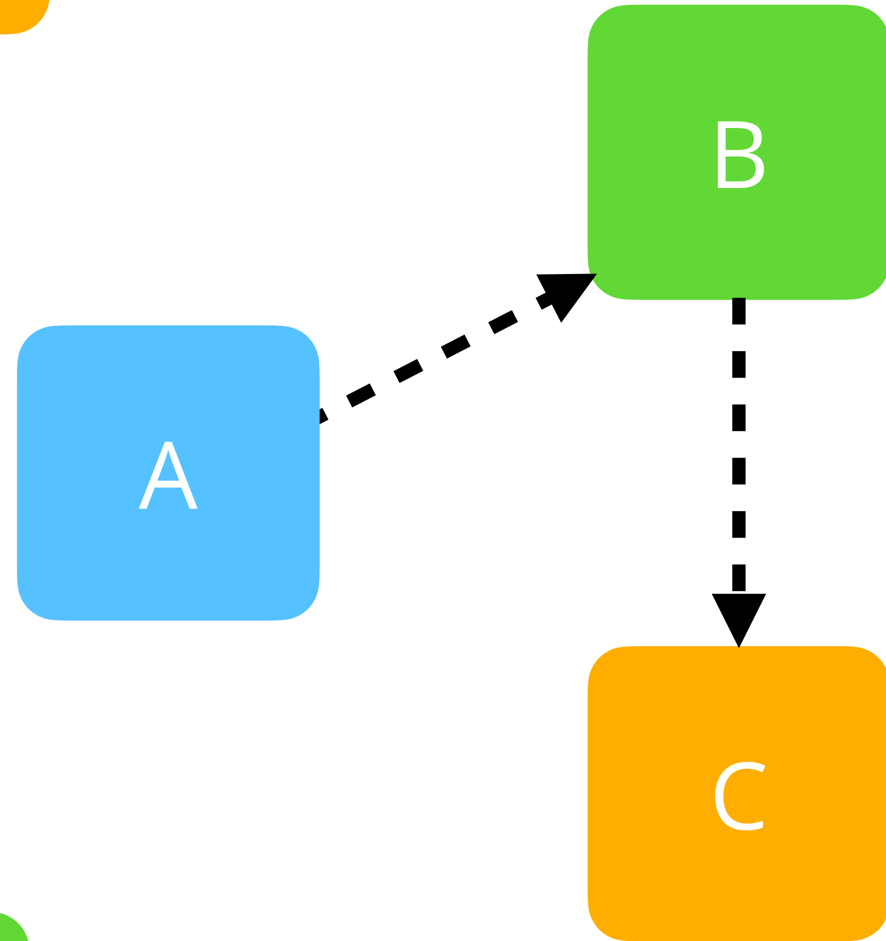



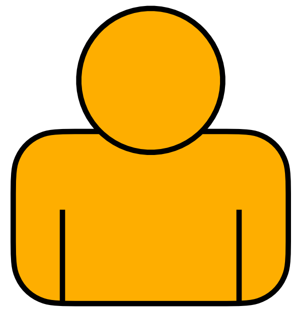
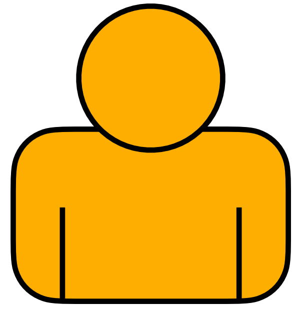
Team A



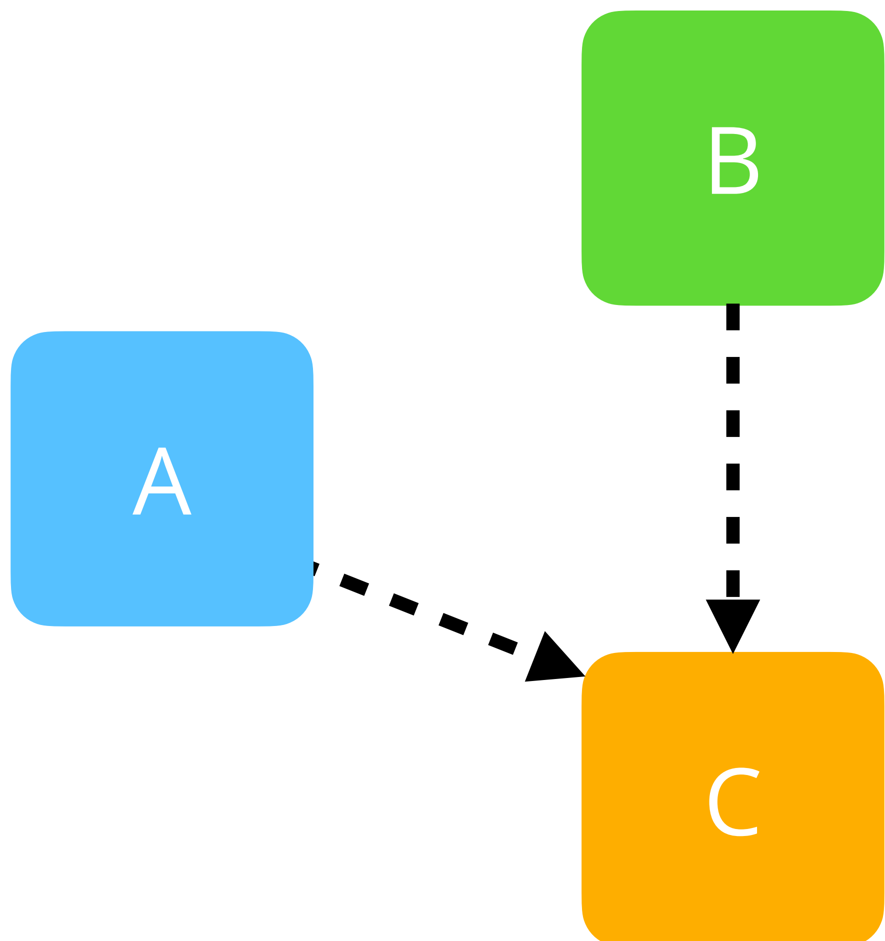



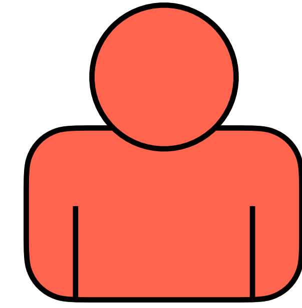
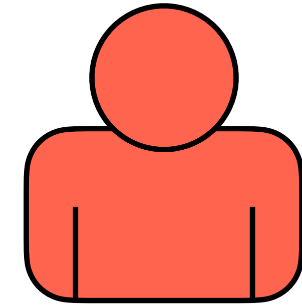
Team B



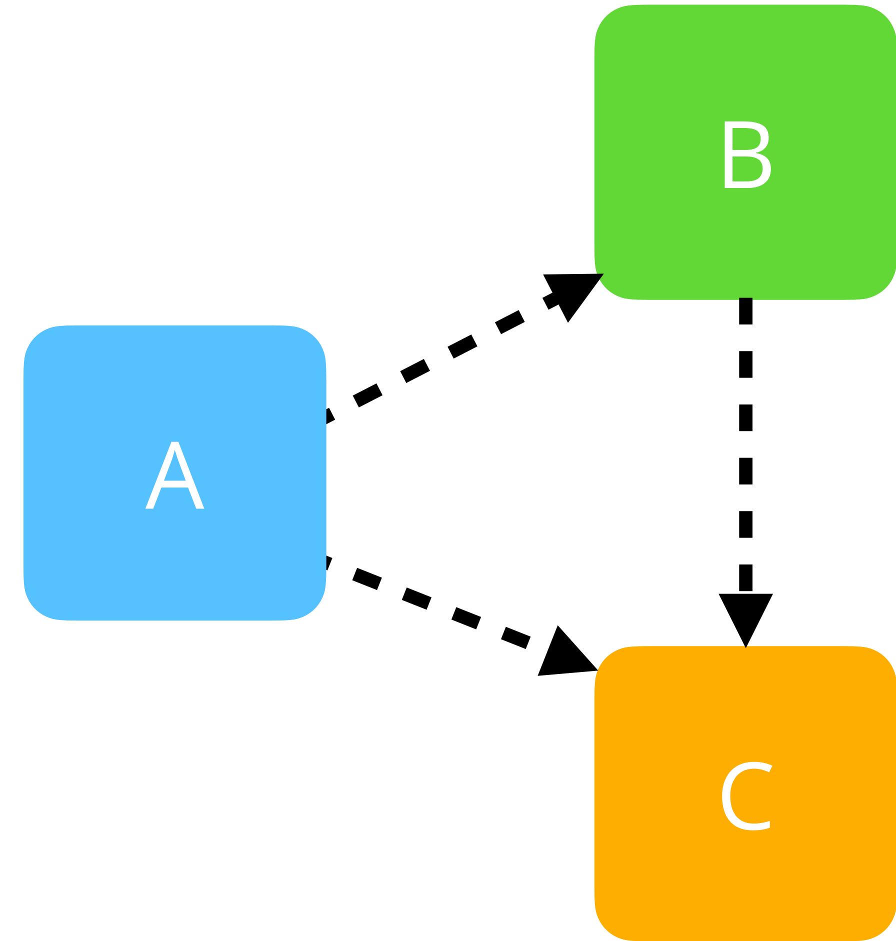



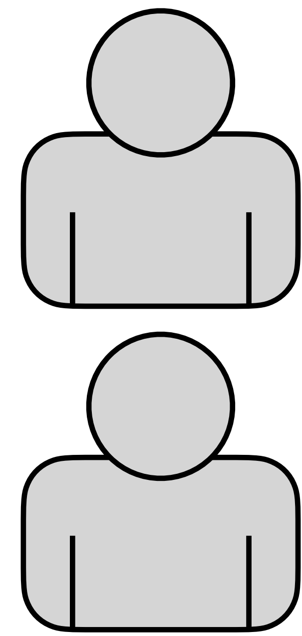
Team C



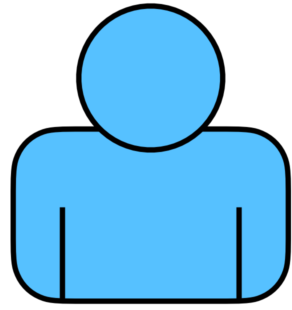
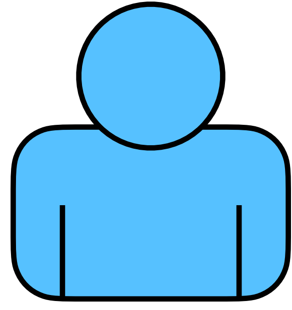
Enterprise Architecture Team





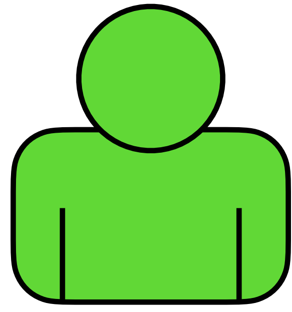
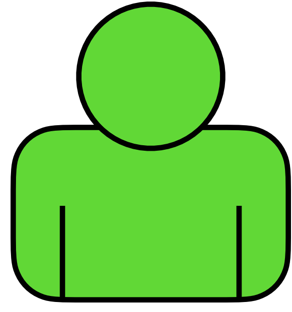

```
model {
  a = softwareSystem "A"
  b = softwareSystem "B"
  c = softwareSystem "C"
}
```

system catalog

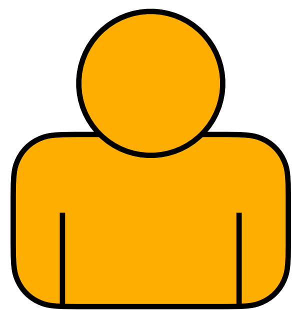
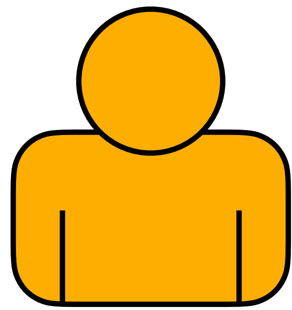
Team A

```
model extends “system catalog” {
  a {
    app = container "App"
  }
  a.app -> b
  a.app -> c
}
```

Team B

```
model extends “system catalog”{
  b {
    api = container "API"
  }
  a -> b.api
  b.api -> c
}
```

Team C

```
model extends “system catalog”{
  c {
    api = container "API"
  }
  a -> c.api
  b -> c.api
}
```

push

push

push

Architecture repository

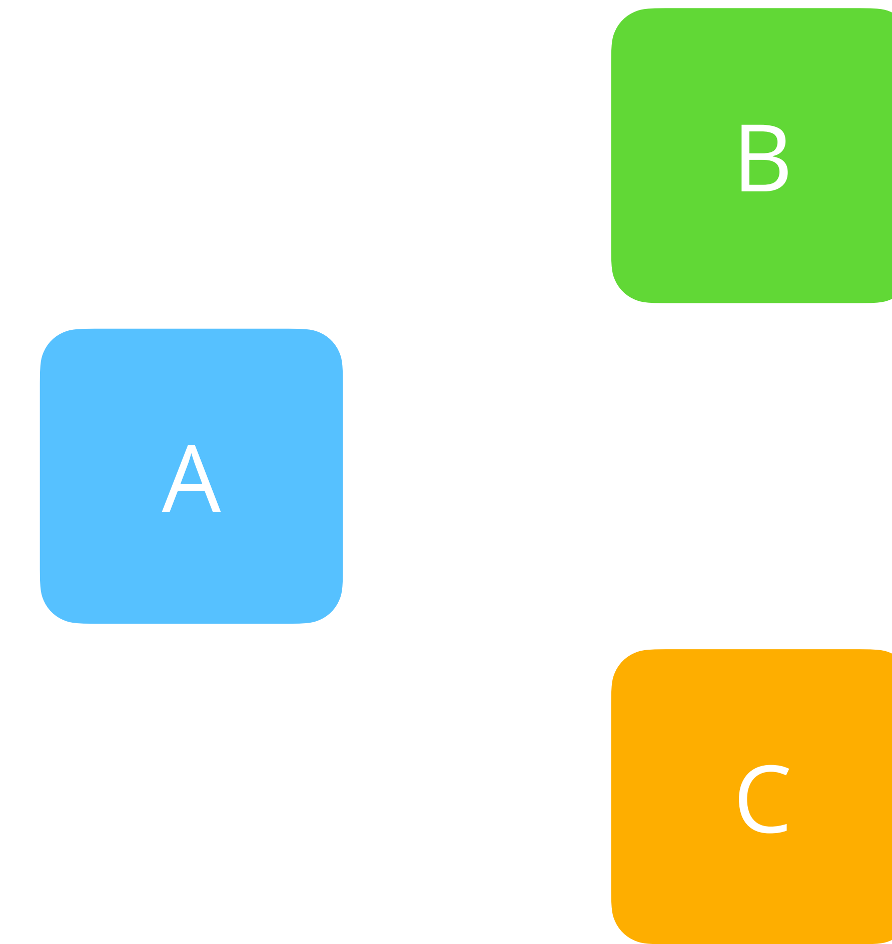
Model 1
Software System A

Model 2
Software System B

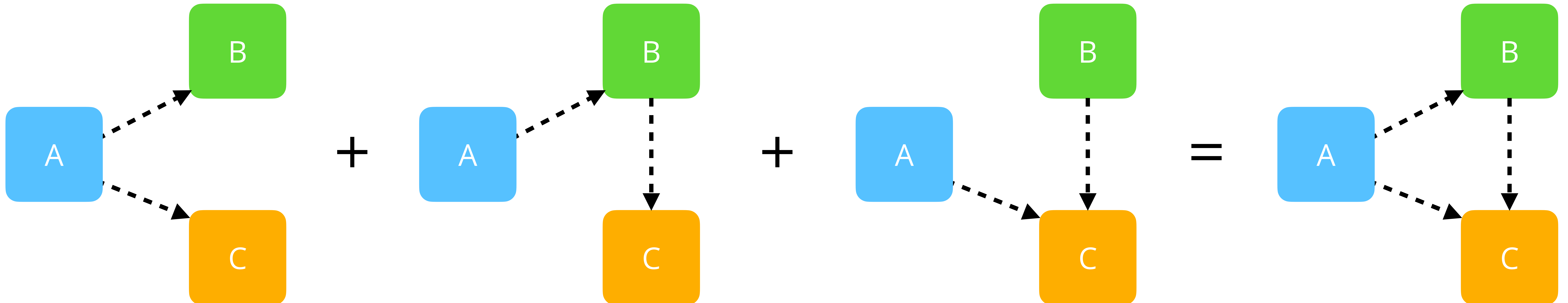
Model 3
Software System C

1. Start with the system catalog

```
model {
  a = softwareSystem "A"
  b = softwareSystem "B"
  c = softwareSystem "C"
}
```



2. Clone relationships from the team-based models



A final note…

Level 1	Level 2	Level 3	Level 4	Level 5
Initial No software architecture diagrams.	**Ad hoc** Software architecture diagrams with ad hoc abstractions and notation, in a general purpose diagramming tool.	**Defined** Software architecture diagrams with defined abstractions and notation, in a general purpose diagramming tool.	**Modelled** Software architecture diagrams with defined abstractions and notation, in a modelling tool, authored manually.	**Optimising** - Model elements are shared between teams. - Centralised system landscape views are generated by aggregating decentralised team-based models. - Model elements are reverse-engineered from source code, deployment environment, logs, etc. - Alternative visualisations are used for different use cases (e.g. communication vs exploration). - Models are used as queryable datasets. ...

Software architecture diagramming maturity model

Level 1	Level 2	Level 3	Level 4	Level 5
Initial No software architecture diagrams.	**Ad hoc** Software architecture diagrams with ad hoc abstractions and notation, in a general purpose diagramming tool.	**Defined** Software architecture diagrams with defined abstractions and notation, in a general purpose diagramming tool.	**Modelled** Software architecture diagrams with defined abstractions and notation, in a modelling tool, authored manually.	**Optimising** - Model elements are shared between teams. - Centralised system landscape views are generated by aggregating decentralised team-based models. - Model elements are reverse-engineered from source code, deployment environment, logs, etc. - Alternative visualisations are used for different use cases (e.g. communication vs exploration). - Models are used as queryable datasets. ...

Software architecture diagramming maturity model

Level 1	Level 2	Level 3	Level 4	Level 5
Initial No software architecture diagrams.	**Ad hoc** Software architecture diagrams with ad hoc abstractions and notation, in a general purpose diagramming tool.	**Defined** Software architecture diagrams with defined abstractions and notation, in a general purpose diagramming tool.	**Modelled** Software architecture diagrams with defined abstractions and notation, in a modelling tool, authored manually.	**Optimising** - Model elements are shared between teams. - Centralised system landscape views are generated by aggregating decentralised team-based models. - Model elements are reverse-engineered from source code, deployment environment, logs, etc. - Alternative visualisations are used for different use cases (e.g. communication vs exploration). - Models are used as queryable datasets. ...

Software architecture diagramming maturity model

Level 1	Level 2	Level 3	Level 4	Level 5
Initial No software architecture diagrams.	**Ad hoc** Software architecture diagrams with ad hoc abstractions and notation, in a general purpose diagramming tool.	**Defined** Software architecture diagrams with defined abstractions and notation, in a general purpose diagramming tool.	**Modelled** Software architecture diagrams with defined abstractions and notation, in a modelling tool, authored manually.	**Optimising** - Model elements are shared between teams. - Centralised system landscape views are generated by aggregating decentralised team-based models. - Model elements are reverse-engineered from source code, deployment environment, logs, etc. - Alternative visualisations are used for different use cases (e.g. communication vs exploration). - Models are used as queryable datasets. ...

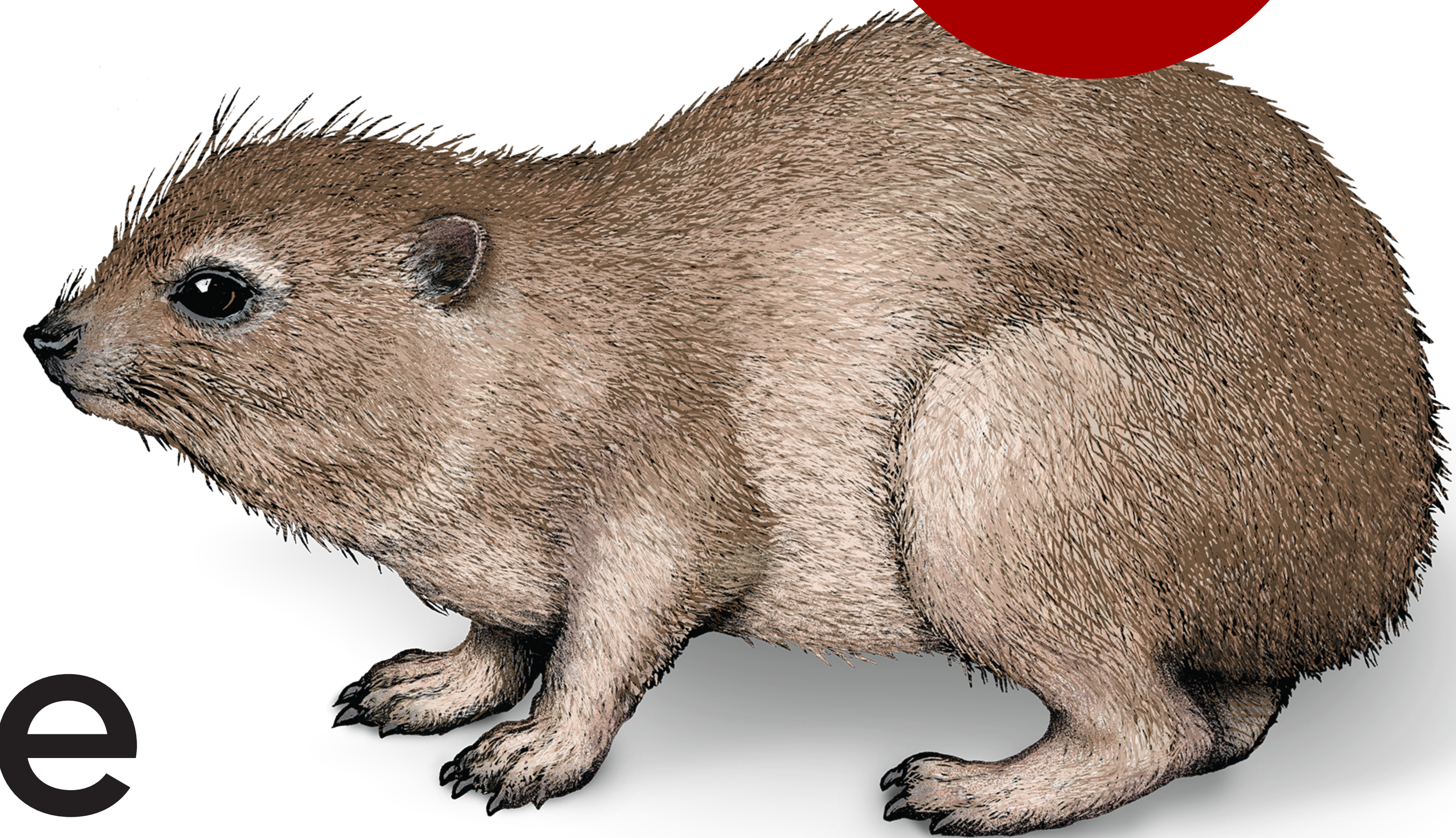
Software architecture diagramming maturity model

Level 1	Level 2	Level 3	Level 4	Level 5
Initial No software architecture diagrams.	**Ad hoc** Software architecture diagrams with ad hoc abstractions and notation, in a general purpose diagramming tool.	**Defined** Software architecture diagrams with defined abstractions and notation, in a general purpose diagramming tool.	**Modelled** Software architecture diagrams with defined abstractions and notation, in a modelling tool, authored manually.	**Optimising** - Model elements are shared between teams. - Centralised system landscape views are generated by aggregating decentralised team-based models. - Model elements are reverse-engineered from source code, deployment environment, logs, etc. - Alternative visualisations are used for different use cases (e.g. communication vs exploration). - Models are used as queryable datasets. ...

Software architecture diagramming maturity model

O'REILLY

Early Release
RAW & UNEDITED



The C4 Model

Visualizing Software Architecture

Simon Brown

Thank you!

c4model.com
simonbrown.je