Software architecture for developers

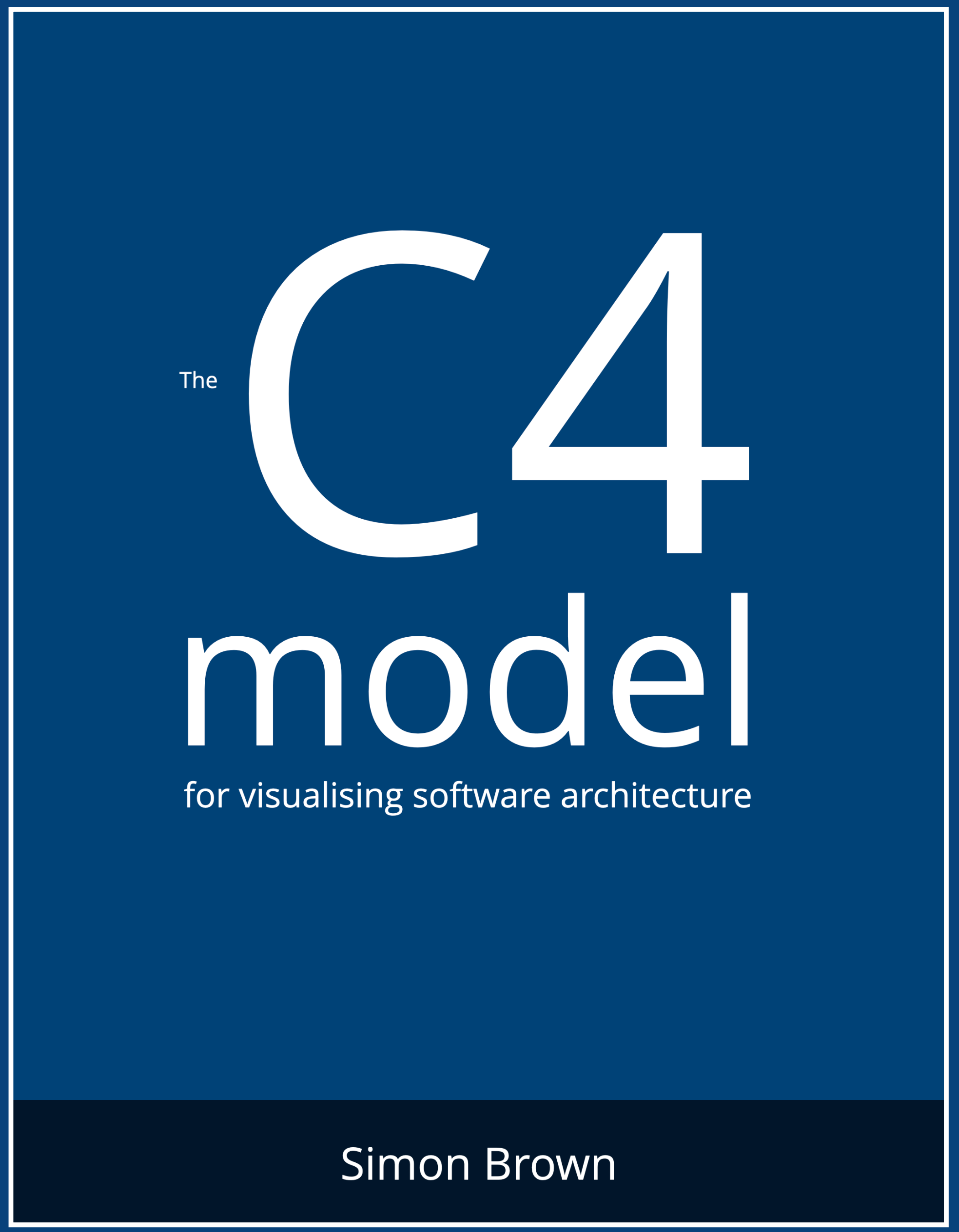
Simon Brown

Simon Brown

Independent consultant specialising in software architecture,
plus the creator of the C4 model and Structurizr

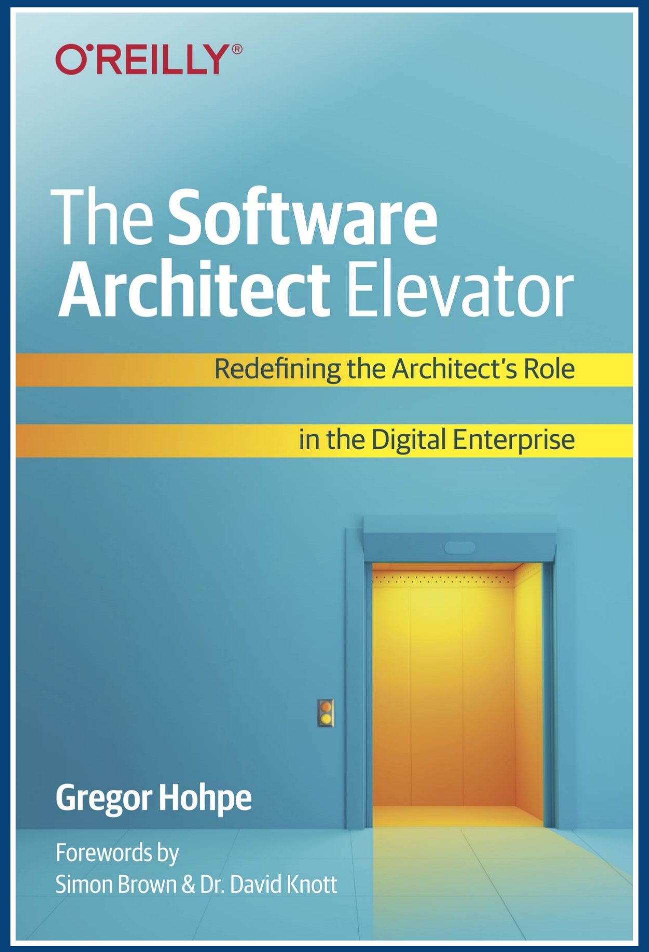
Software architecture for developers

Simon Brown



The C4 model for visualising software architecture

Simon Brown



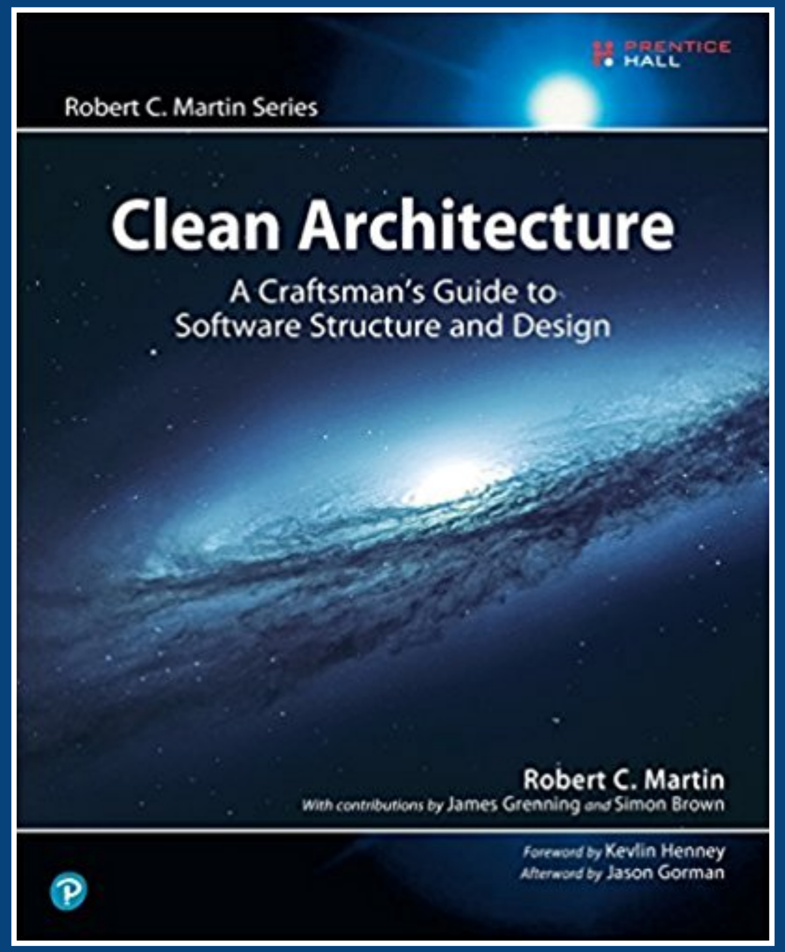
O'REILLY

The Software Architect Elevator

Redefining the Architect's Role in the Digital Enterprise

Gregor Hohpe

Forewords by Simon Brown & Dr. David Knott



Robert C. Martin Series

PRENTICE HALL

Clean Architecture

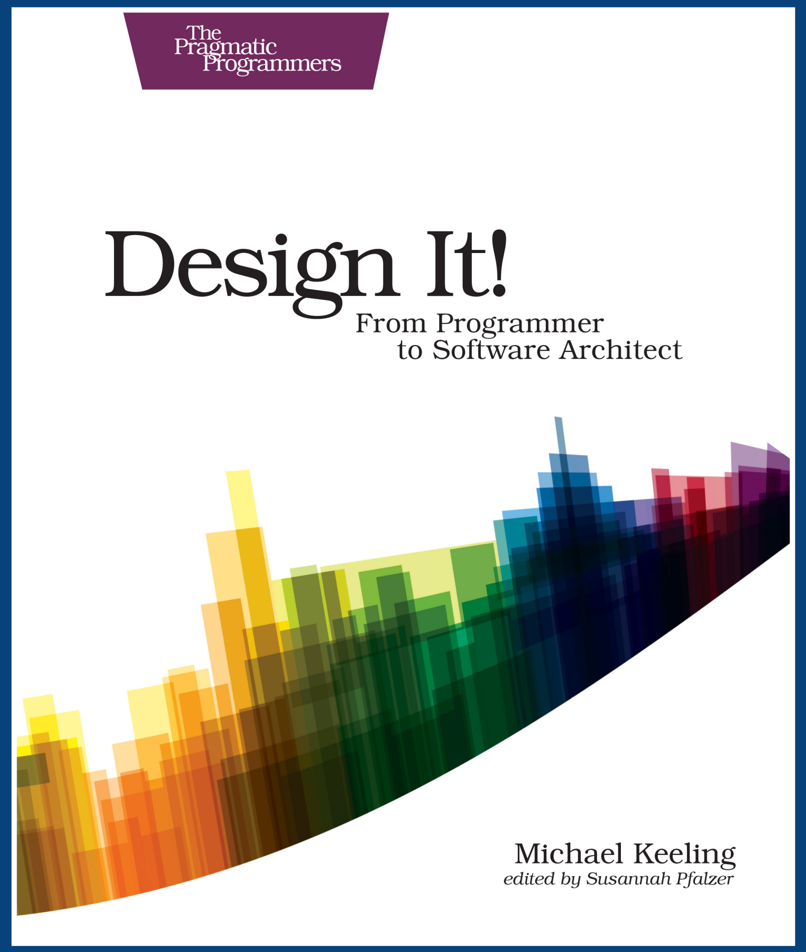
A Craftsman's Guide to Software Structure and Design

Robert C. Martin

With contributions by James Grenning and Simon Brown

Foreword by Kevlin Henney

Afterword by Jason Gorman



The Pragmatic Programmers

Design It!

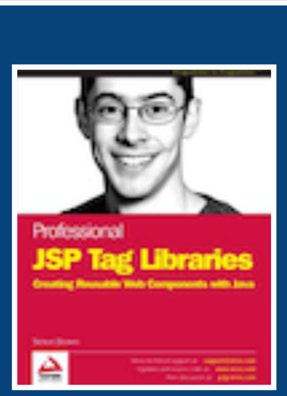
From Programmer to Software Architect

Michael Keeling

edited by Susannah Pfalzer

PROFESSIONAL JSP 2nd Edition

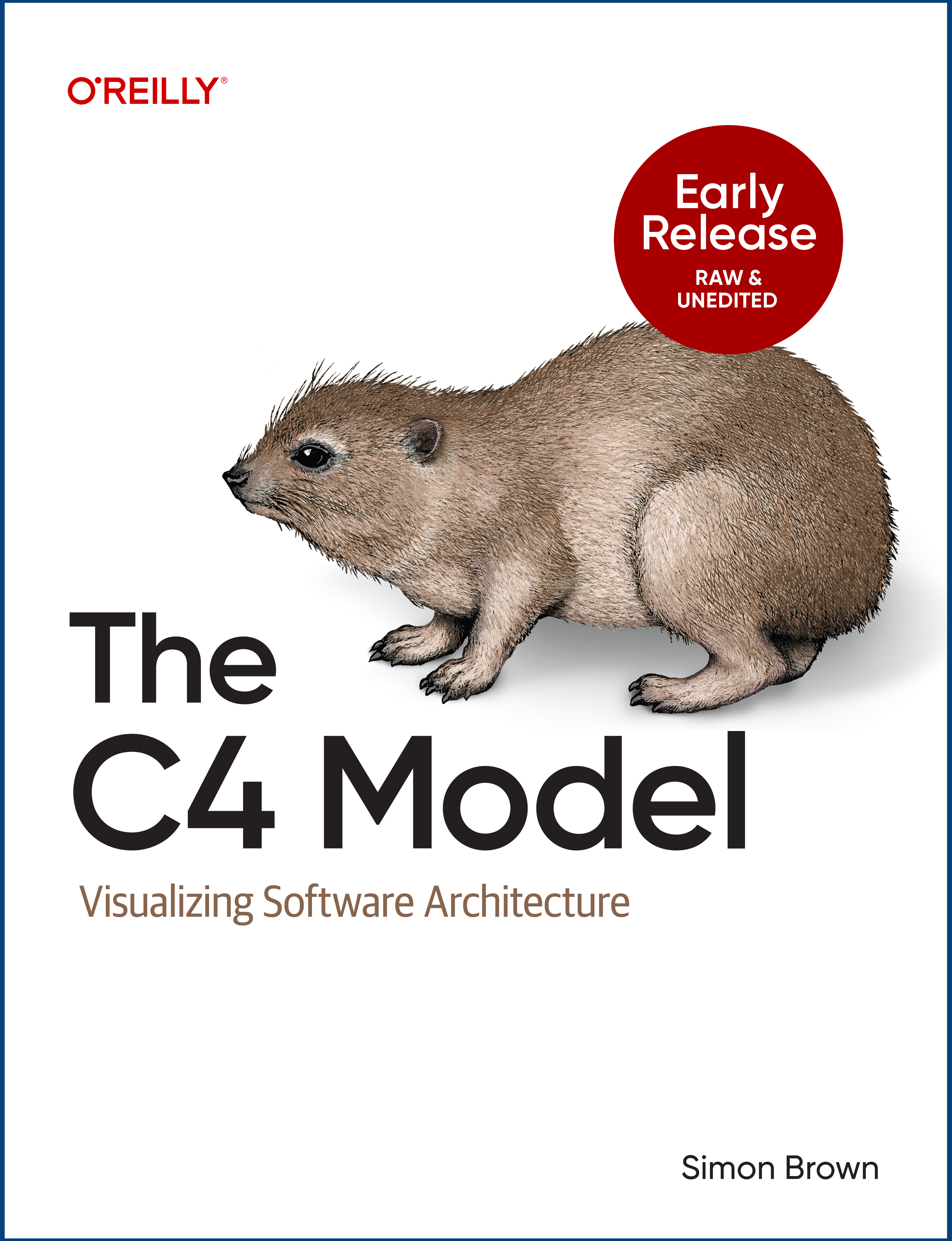
professional Java Servlets 2.3



Professional JSP Tag Libraries

Pro JSP

Pro JSP 2



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

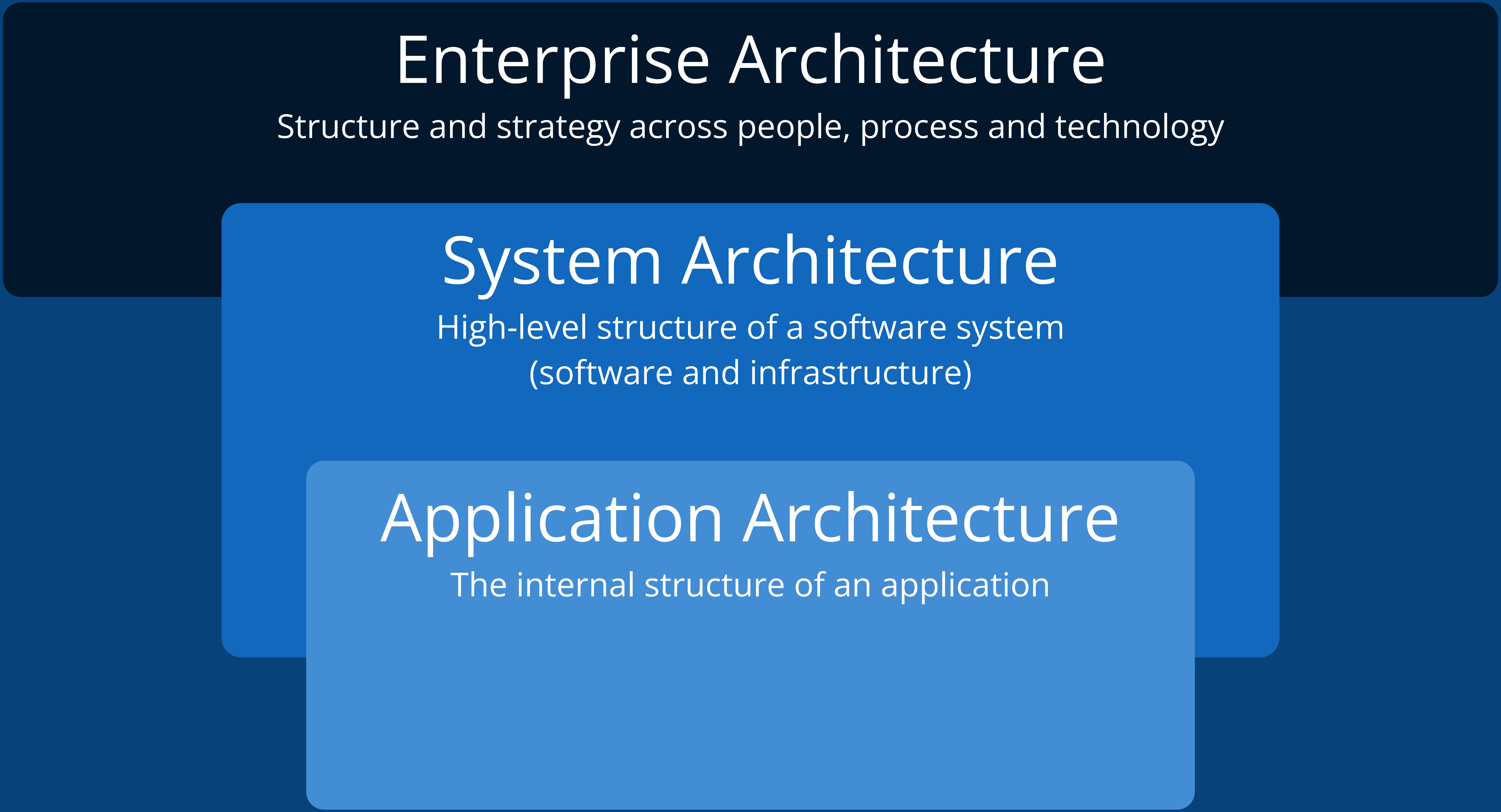
What is software architecture?

Structure

The definition of software in terms of its building blocks and their interactions

Vision

The process of architecting;
making decisions based upon business goals,
requirements and constraints,
plus being able to communicate this to a team



Enterprise Architecture

Structure and strategy across people, process and technology

System Architecture

High-level structure of a software system
(software and infrastructure)

Application Architecture

The internal structure of an application

As a noun, design is the named structure or behaviour of a system … a design thus represents one point in a potential decision space.

Grady Booch

All architecture is design, but
not all design is architecture.

Grady Booch

Architecture represents the **significant decisions**, where significance is measured by **cost of change**.

Grady Booch

As architects, we define
the **significant decisions**

Technology

Programming languages, libraries, frameworks, deployment environments, etc

Elements

How software is decomposed into smaller executable building blocks at different levels of abstraction (e.g. monoliths vs microservices, package by layer vs package by feature) and how data is stored (e.g. data schemas and formats)

Relationships

Dependencies and interactions between elements (e.g. synchronous vs asynchronous communication, data formats, protocols, etc)

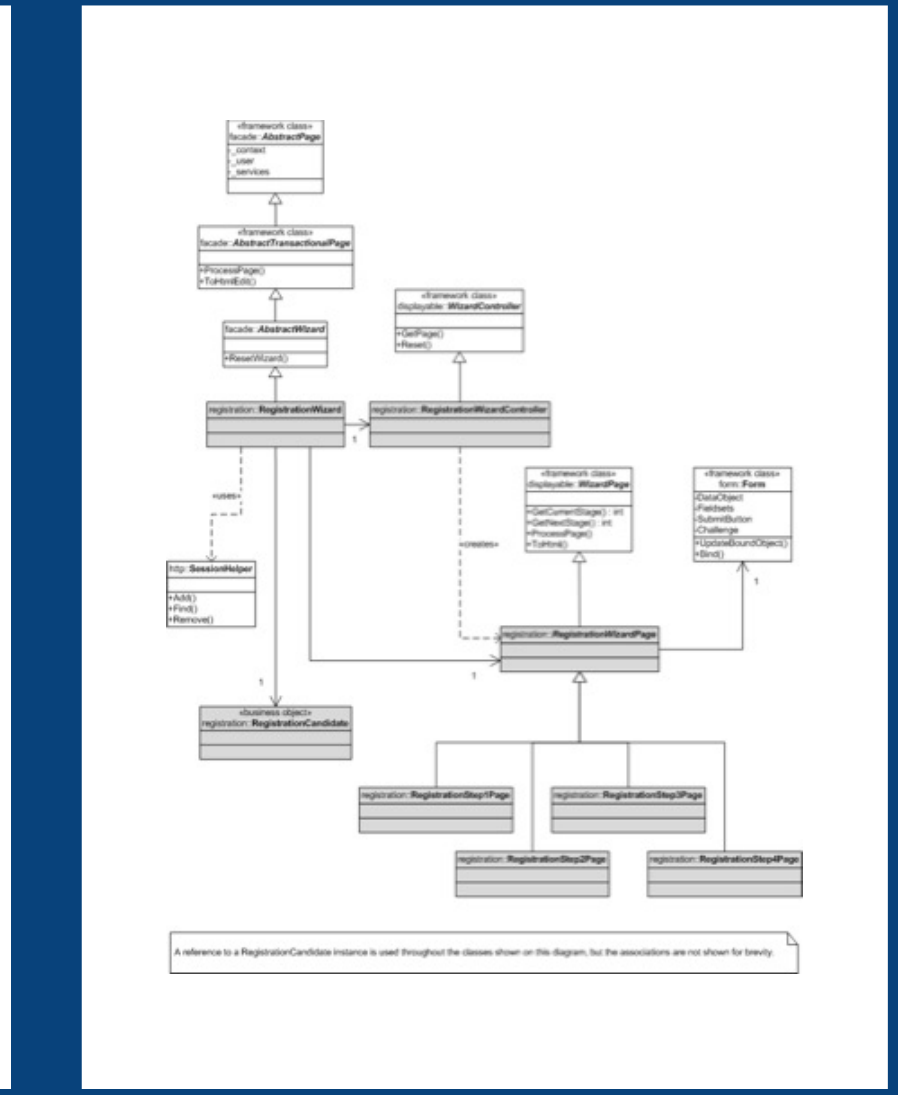
What happens if a software development team **doesn’t think about architecture**?

Chaos

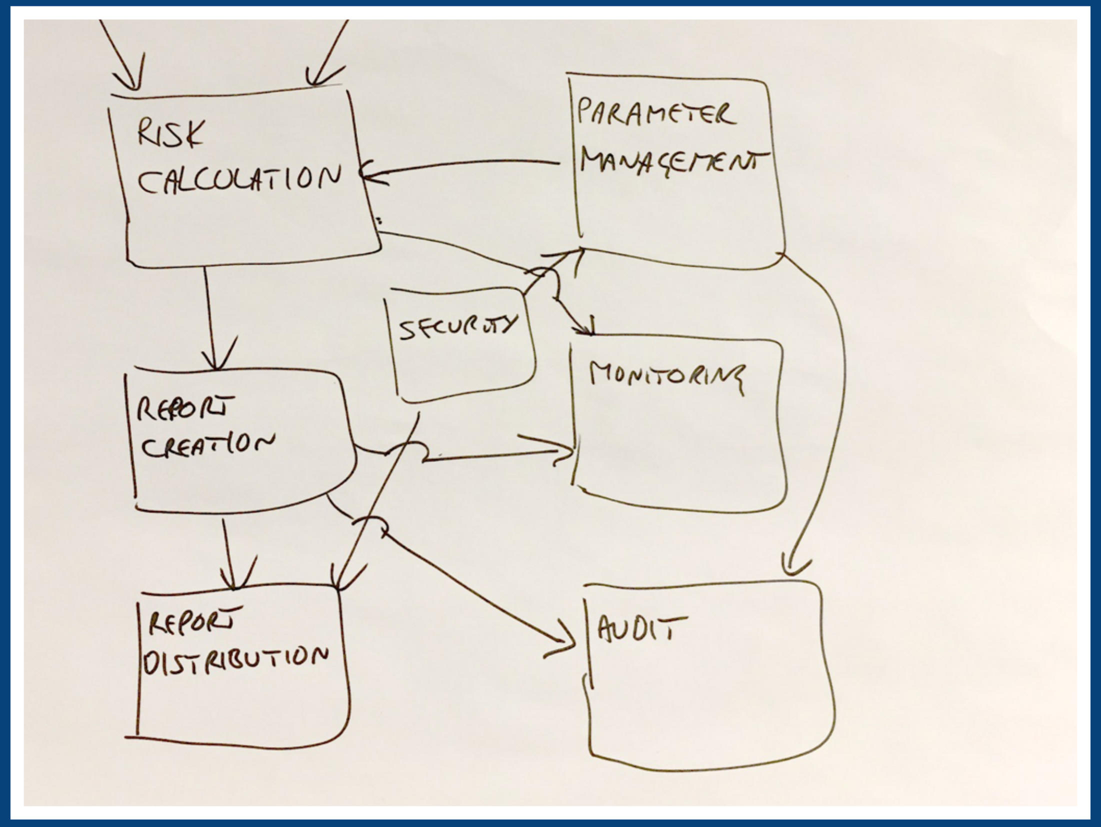
Big ball of mud, spaghetti code, inconsistent approaches to solving the same problems, quality attributes are ignored, deployment problems, maintenance issues, etc

Big design up front

Software Architecture Document



vs



No design up front

> Big design up front is dumb.
> Doing no design up front
> is even dumber.

Dave Thomas

Software architecture helps us avoid chaos

Architectural drivers

Requirements

drive architecture

(use cases, user stories, features, etc)

Requirement

"a thing that is needed or wanted"

(this includes experiments and hypotheses too)

Don’t start designing software if you have no inputs

Quality attributes

(also known as non-functional requirements,
cross-cutting concerns, service-level agreements, etc)

What **quality attributes** might be relevant for the ”Financial Risk System”?

- Performance
- Scalability
- Availability
- Security
- Disaster Recovery
- Accessibility
- Monitoring
- Management
- Audit
- Flexibility
- Extensibility
- Maintainability
- Interoperability
- Legal
- Regulatory
- Compliance
- i18n
- L10n

Create a **checklist** of quality attributes you regularly encounter

Understand how to **capture**, **refine** and **challenge** quality attributes

Software lives in the real world,
and the real world has

constraints

Typical constraints include
time and budget, technology,
people and skills, politics, etc

Constraints can **sometimes** be prioritised

Principles

are selected by the team

Development principles include coding conventions, naming guidelines, testing approaches, review practices, etc

Architecture and design principles typically relate to modularity or crosscutting concerns

(architectural layering, separation of concerns, stateless vs stateful, rich vs anaemic domain, security, error handling, logging, etc)

Ensure you have a good understanding of the requirements, quality attributes, constraints and principles to create

sufficient foundations

What about agile, and agility?

Agile is about moving fast, embracing change, releasing often, getting feedback, …

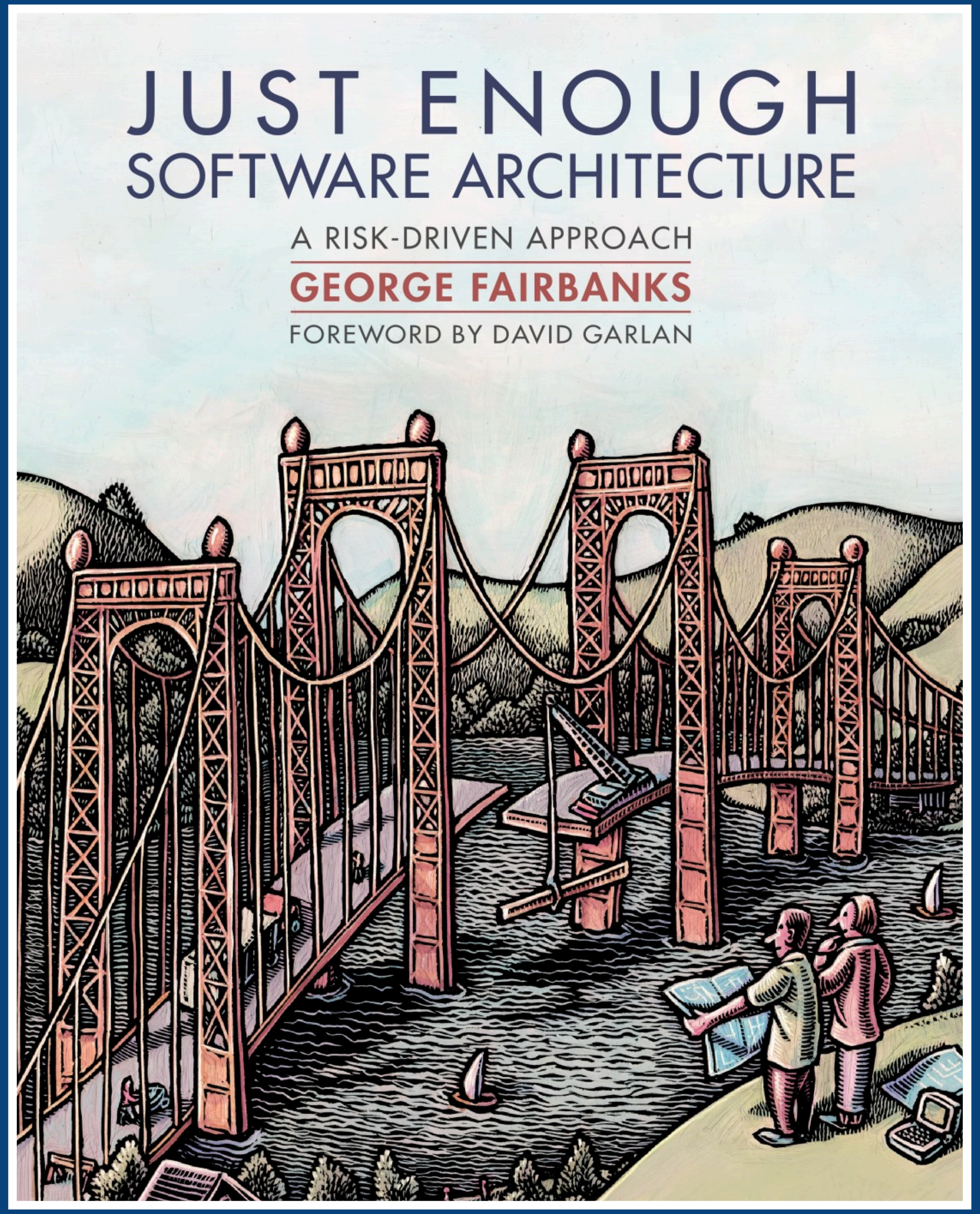
Agile is about a mindset of **continuous improvement**

Inspect and **adapt**

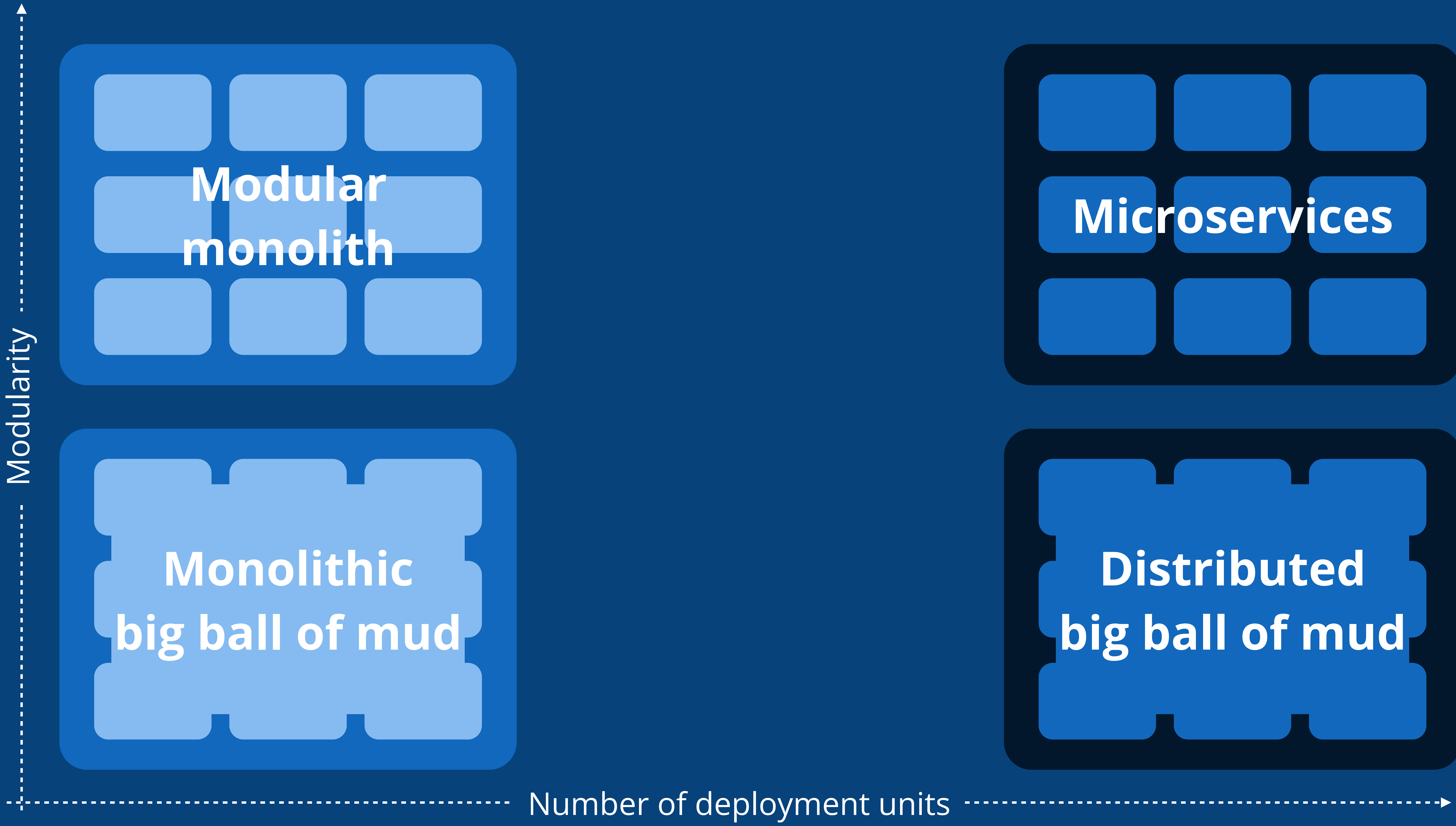
Continuous attention to technical excellence and good design enhances agility.

Principle 9 of the Manifesto for Agile Software Development

A good architecture enables agility



A good architecture rarely happens through **architecture-indifferent design**



Modular monolith

Microservices

Monolithic big ball of mud

Distributed big ball of mud

Modularity

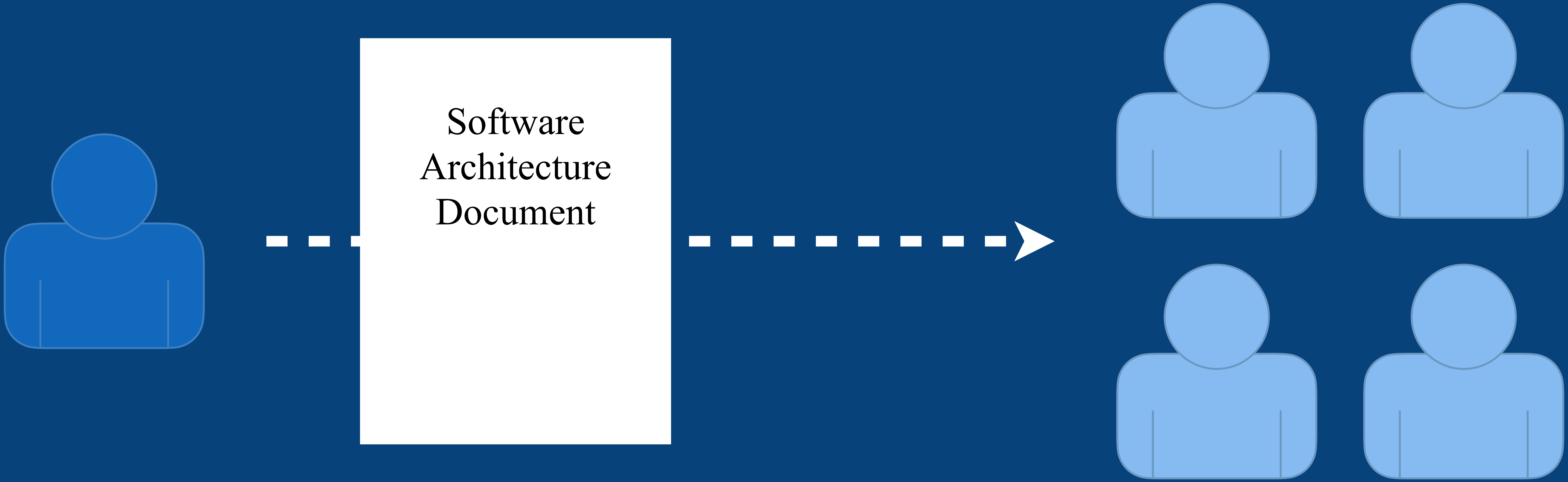
Number of deployment units

Agility is a

quality attribute

The software architecture role

Software development is not a relay sport



Software Architecture Document

AaaS

Architecture as a Service

The software architecture role
is about the “**big picture**”
and, sometimes, this means
stepping away from the code

The software architecture role

(technical leadership, and responsible for the technical success of the project/product)

Architectural drivers

Understanding the goals; capturing, refining, and challenging the requirements and constraints.

Designing software

Creating the technical strategy, vision, alignment, and roadmap.

Technical risks

Identifying, mitigating and owning the technical risks to ensure that the architecture "works".

Technical leadership

Continuous technical leadership and ownership of the architecture throughout the software delivery.

Quality assurance

Introduction and adherence to standards, guidelines, principles, etc plus management of technical debt.

Software development teams don’t need architects

Software development teams do need technical leadership

Every team needs technical leadership

Continuous technical leadership

(somebody needs to continuously steer the ship)

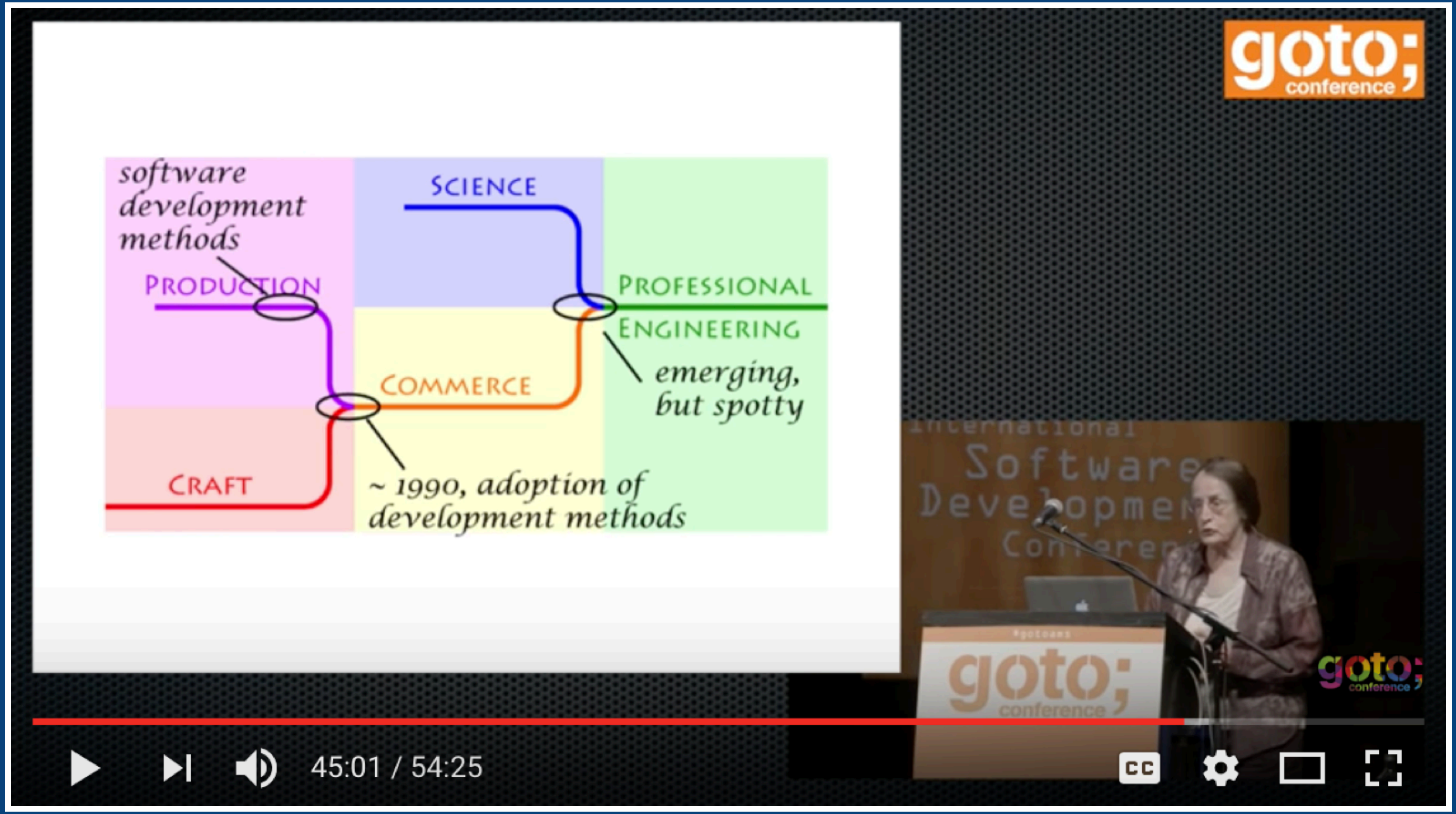
Should software architects write **code**?

Production code, prototypes, frameworks, foundations, code reviews, experimenting, etc

Don’t code all of the time!

There is often a tension between being “senior” and writing code…

Software architects should be **master builders**



Progress Toward an Engineering Discipline of Software

Mary Shaw

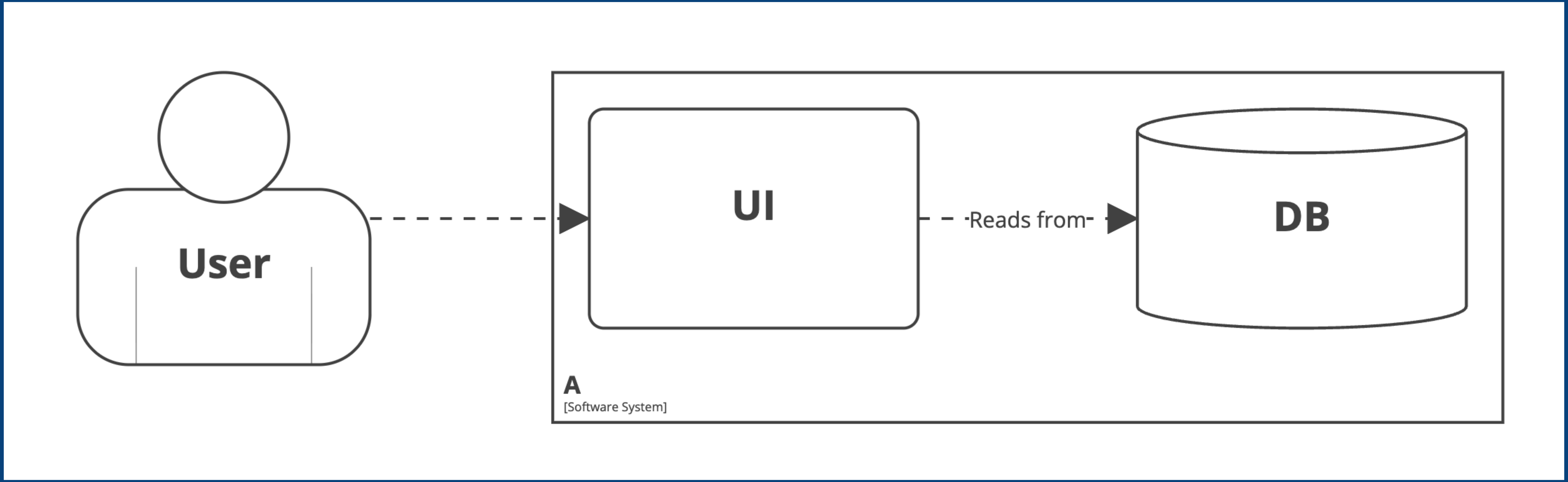
T

Technology skills

Good software architects
are typically
good software developers

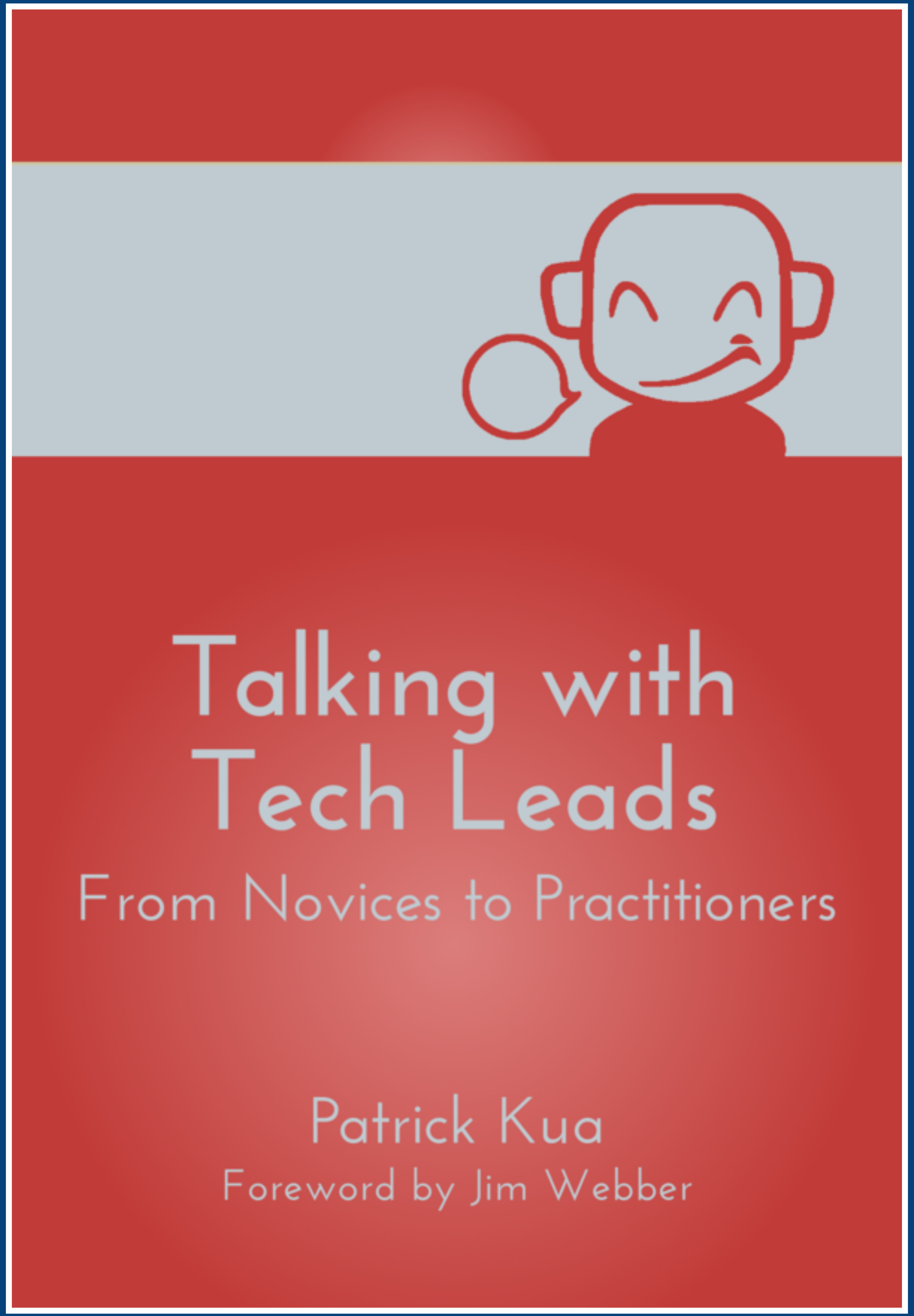
The people designing software must understand technology …

all decisions involve trade-offs


User
UI
Reads from
DB
A
[Software System]

Soft skills

(leadership, communication, presentation, influencing, negotiation, collaboration, coaching and mentoring, motivation, facilitation, political, etc)

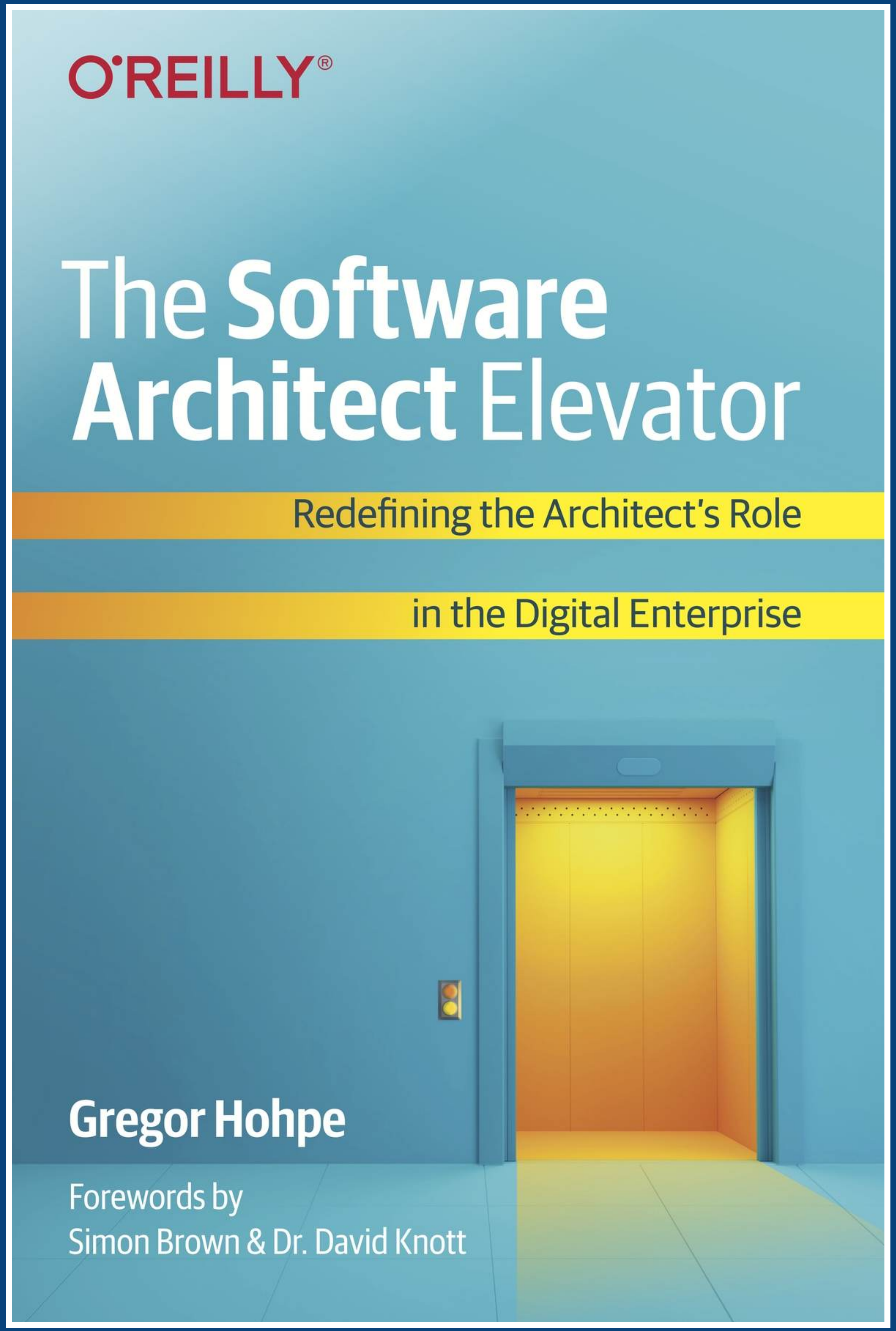


Talking with Tech Leads

From Novices to Practitioners

Patrick Kua

Foreword by Jim Webber



O'REILLY®

The Software Architect Elevator

Redefining the Architect's Role in the Digital Enterprise

Gregor Hohpe

Forewords by
Simon Brown & Dr. David Knott

Domain knowledge

(or the ability to learn quickly)

The software architecture role is **multi-faceted**

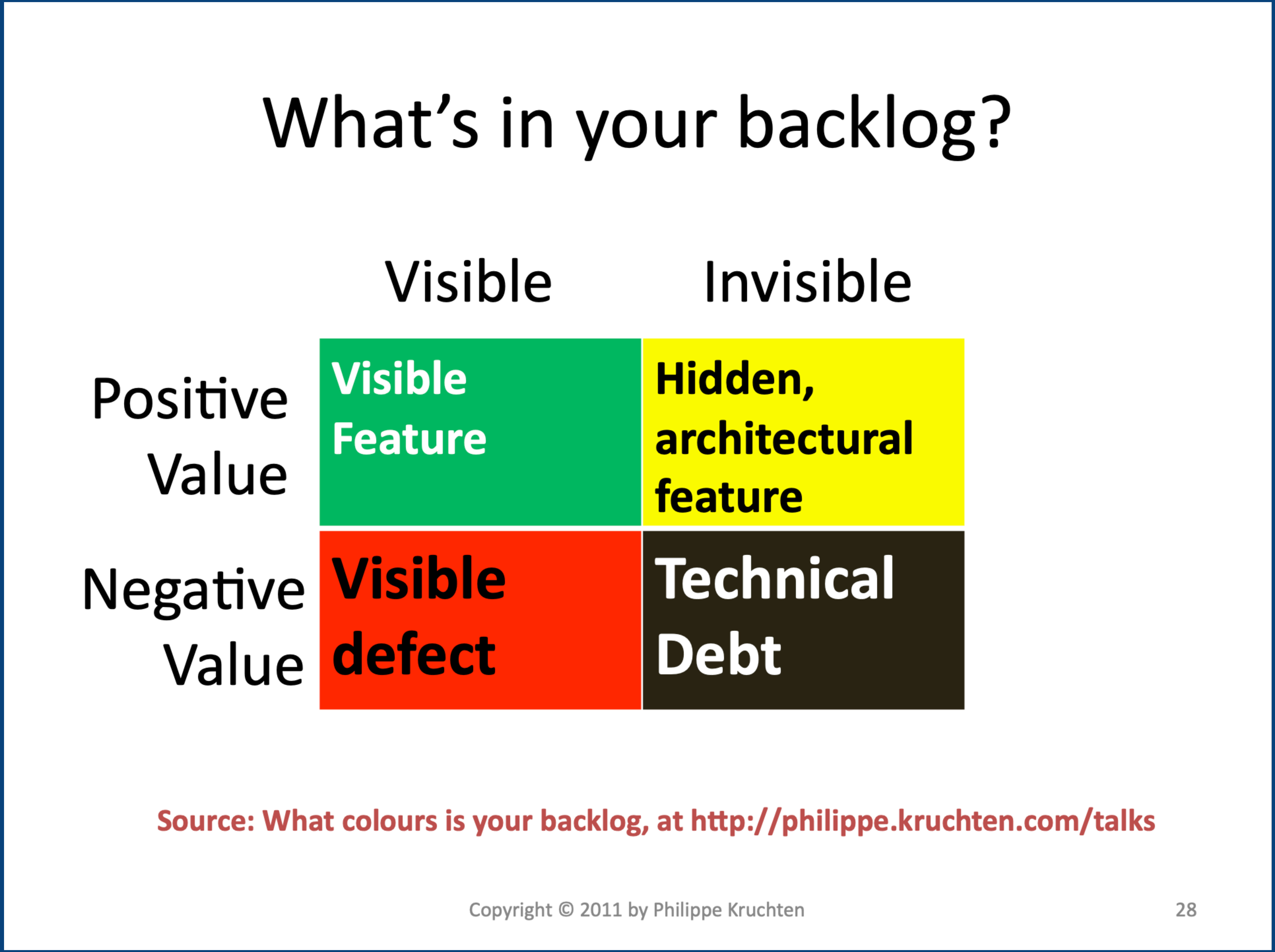
(technology, soft skills, domain knowledge)

Software architects,
solution architects,
tech leads,
principal engineers?

Technical priorities

vs

product priorities?



What’s in your backlog?

	Visible	Invisible
Positive Value	**Visible Feature**	**Hidden, architectural feature**
Negative Value	**Visible defect**	**Technical Debt**

Source: What colours is your backlog, at http://philippe.kruchten.com/talks

https://philippe.kruchten.com/wp-content/uploads/2011/10/kruchten-111027-techdebt.pdf

The product owner(s) and software architect(s) are peers

(“Architecture Owner” is another term you can use)

Everybody should be an architect

“everybody is responsible for architecture”

!=

everybody being responsible for architecture

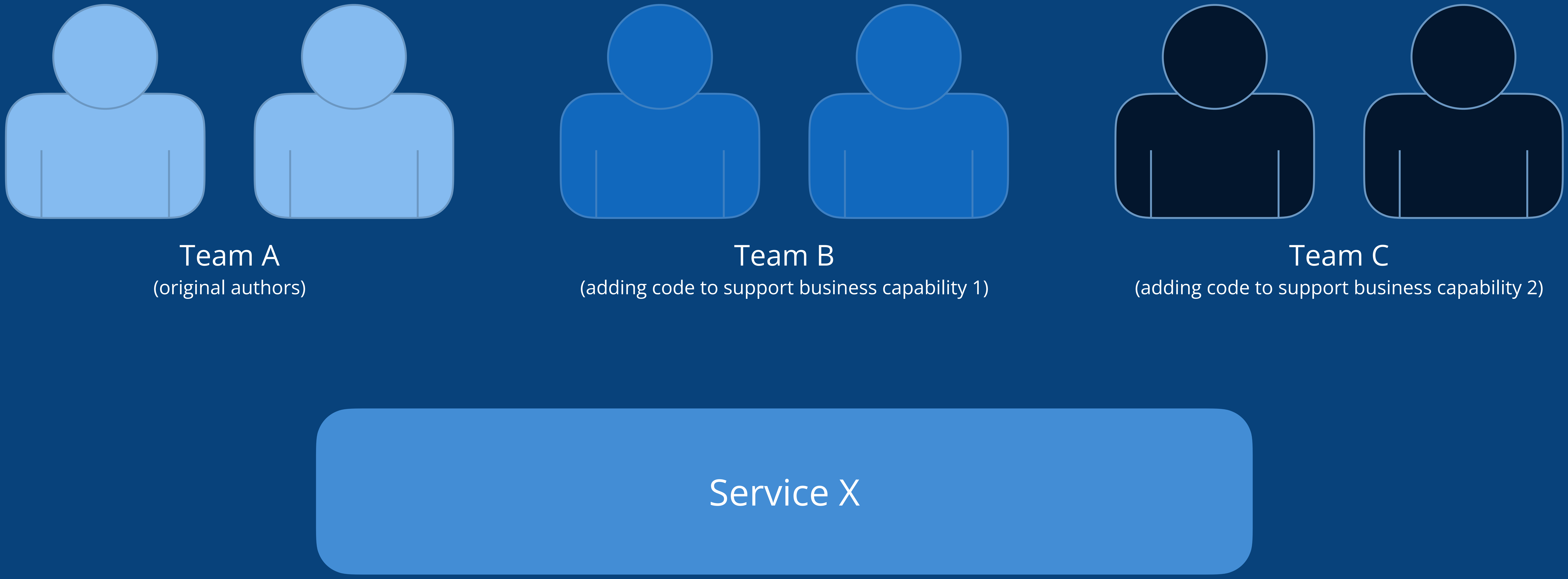
Everybody* should

own the architecture

teams should be agile, autonomous, and self-organising

just hire good people
and trust them to do
the right thing

Does everybody have the skills and motivation to collaborate on the software architecture role?



Team A

(original authors)

Team B

(adding code to support business capability 1)

Team C

(adding code to support business capability 2)

Service X

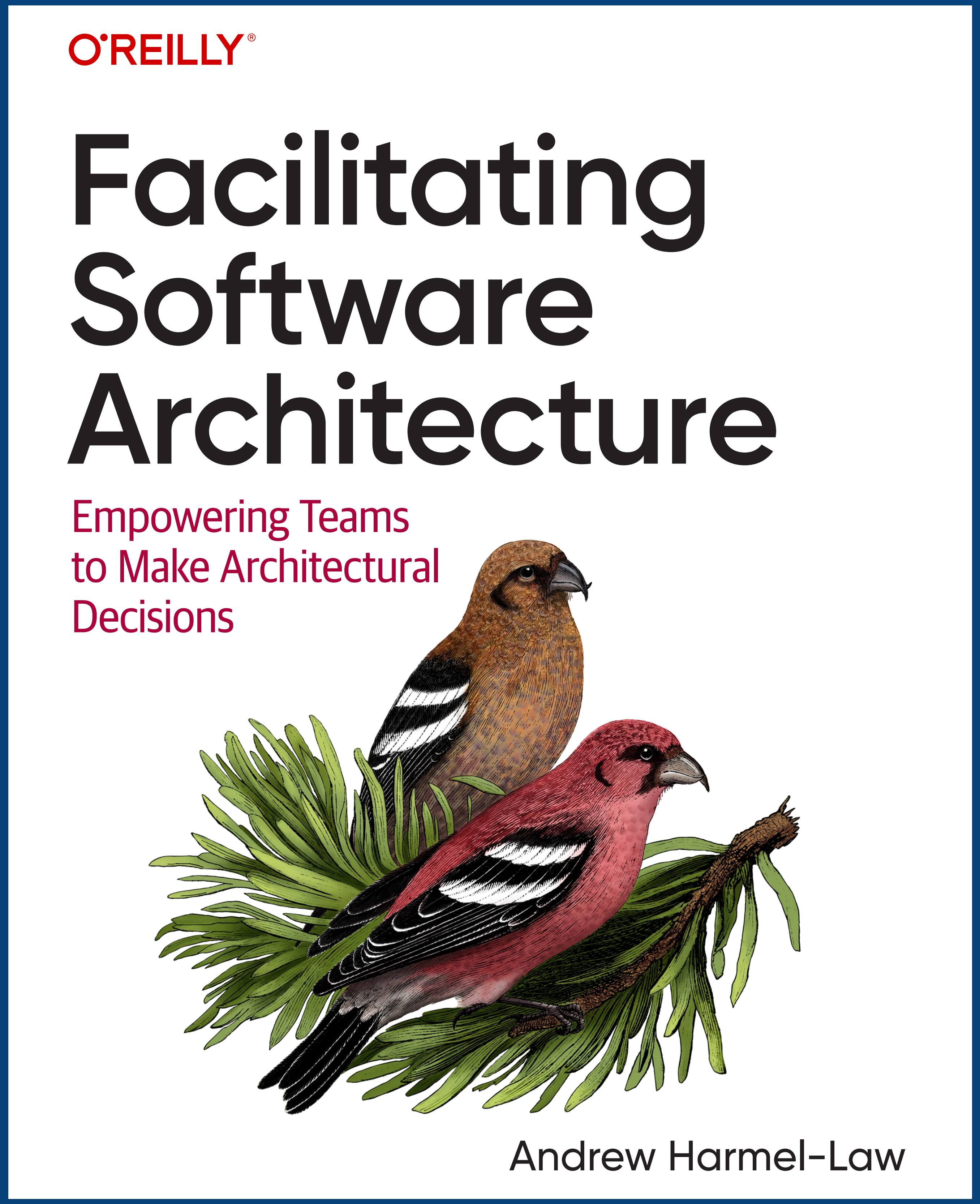
Product vs stream leadership

Hierarchies of architects, central architecture groups, technical design authorities, etc?

Decision making

Centralised vs decentralised

Tactical vs strategic



O'REILLY®

Facilitating Software Architecture

Empowering Teams to Make Architectural Decisions

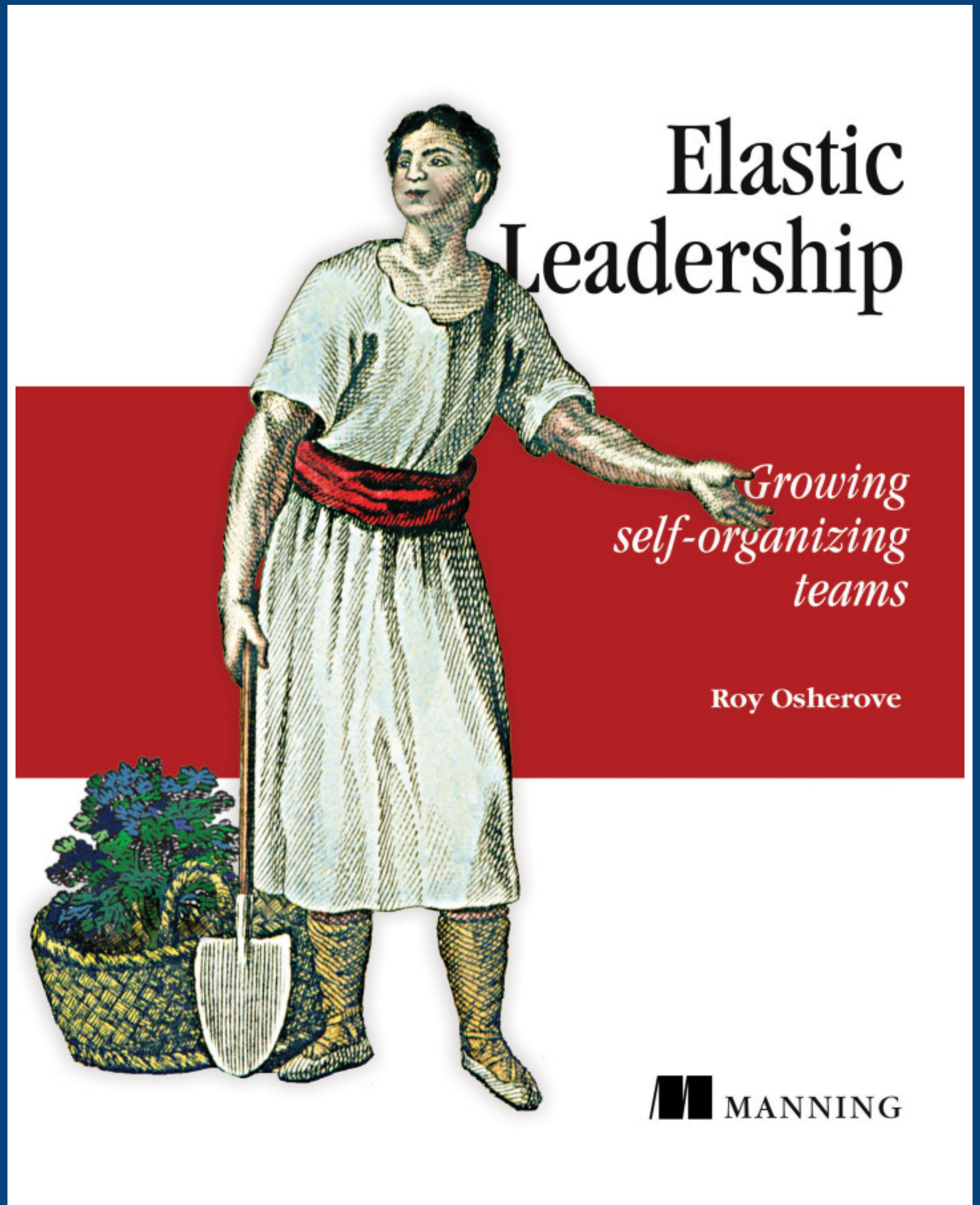
Andrew Harmel-Law

“Architecture advice process”

Introducing control?

Avoiding chaos?

How much control do you need?



Different types of teams need different leadership styles

Pair architecting

Collaborative technical leadership

is not easy

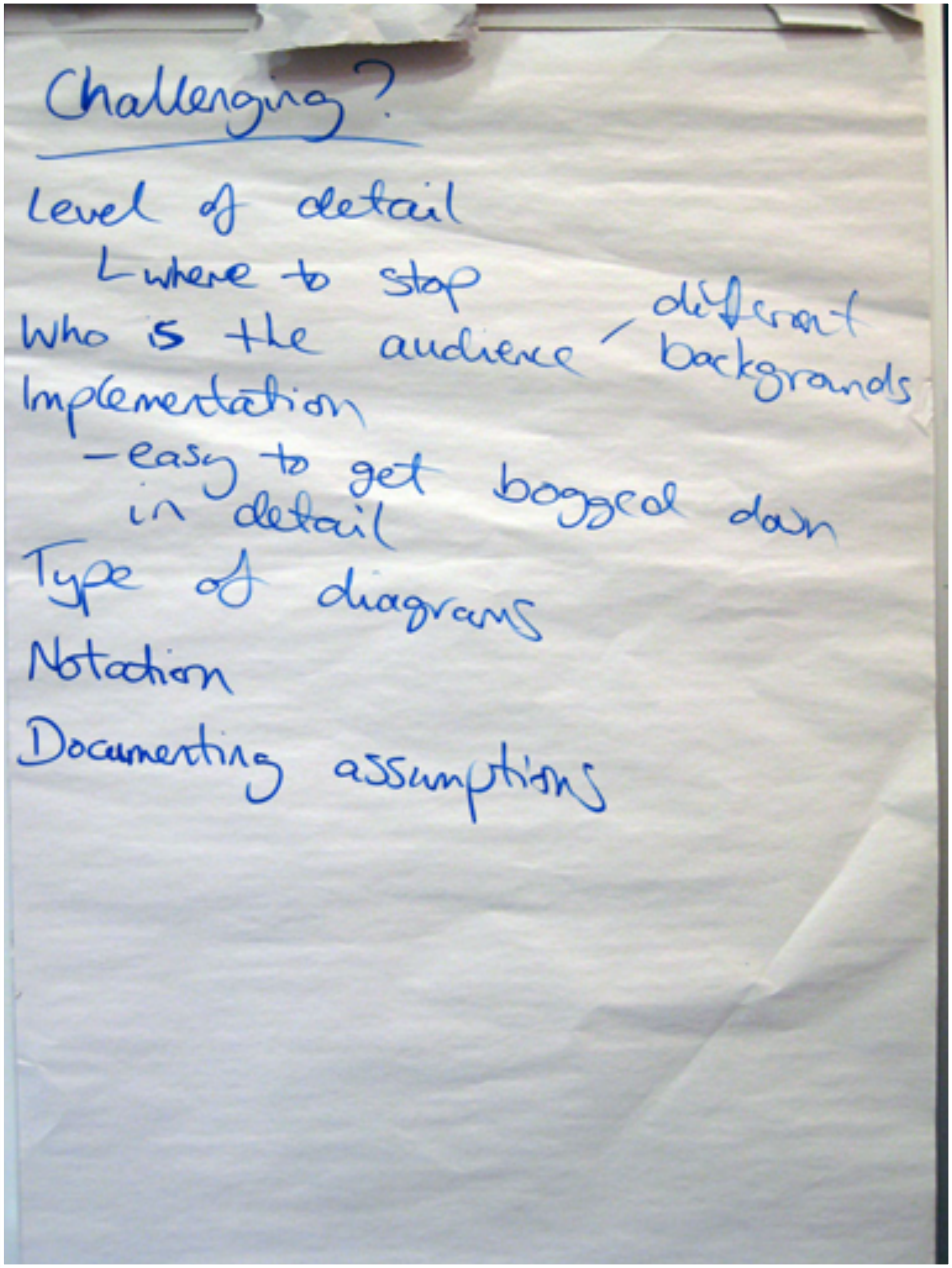
Collaborate or fail

Draw one or more software architecture diagrams to describe a solution for the ”Financial Risk System”

⏱️ = 1.5 hours (15:15)

c4model.com/frs

Did you find anything about this exercise challenging?



Challenging?

Level of detail

└ where to stop

Who is the audience / different backgrounds

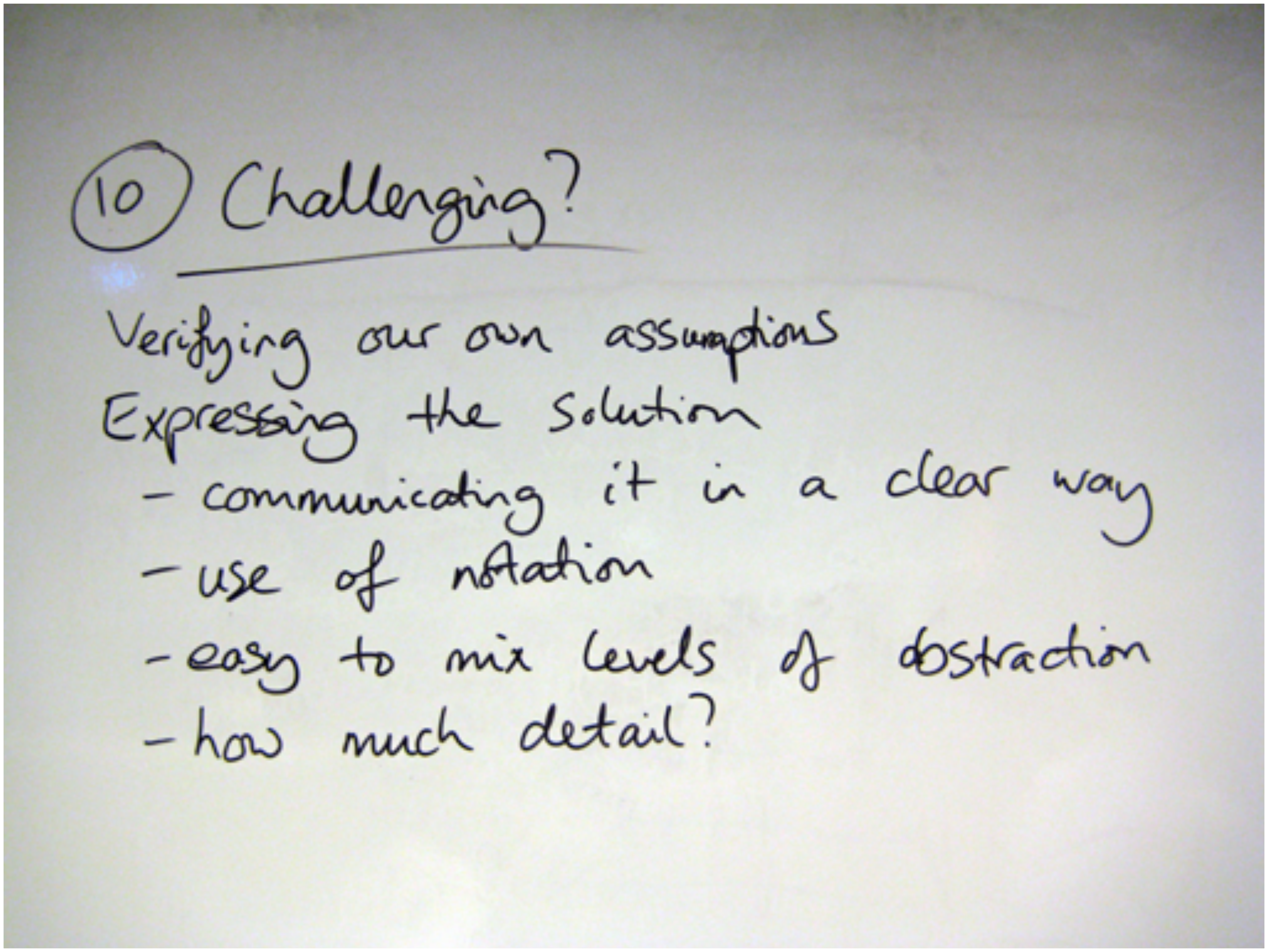
Implementation

- easy to get bogged down in detail

Type of diagrams

Notation

Documenting assumptions

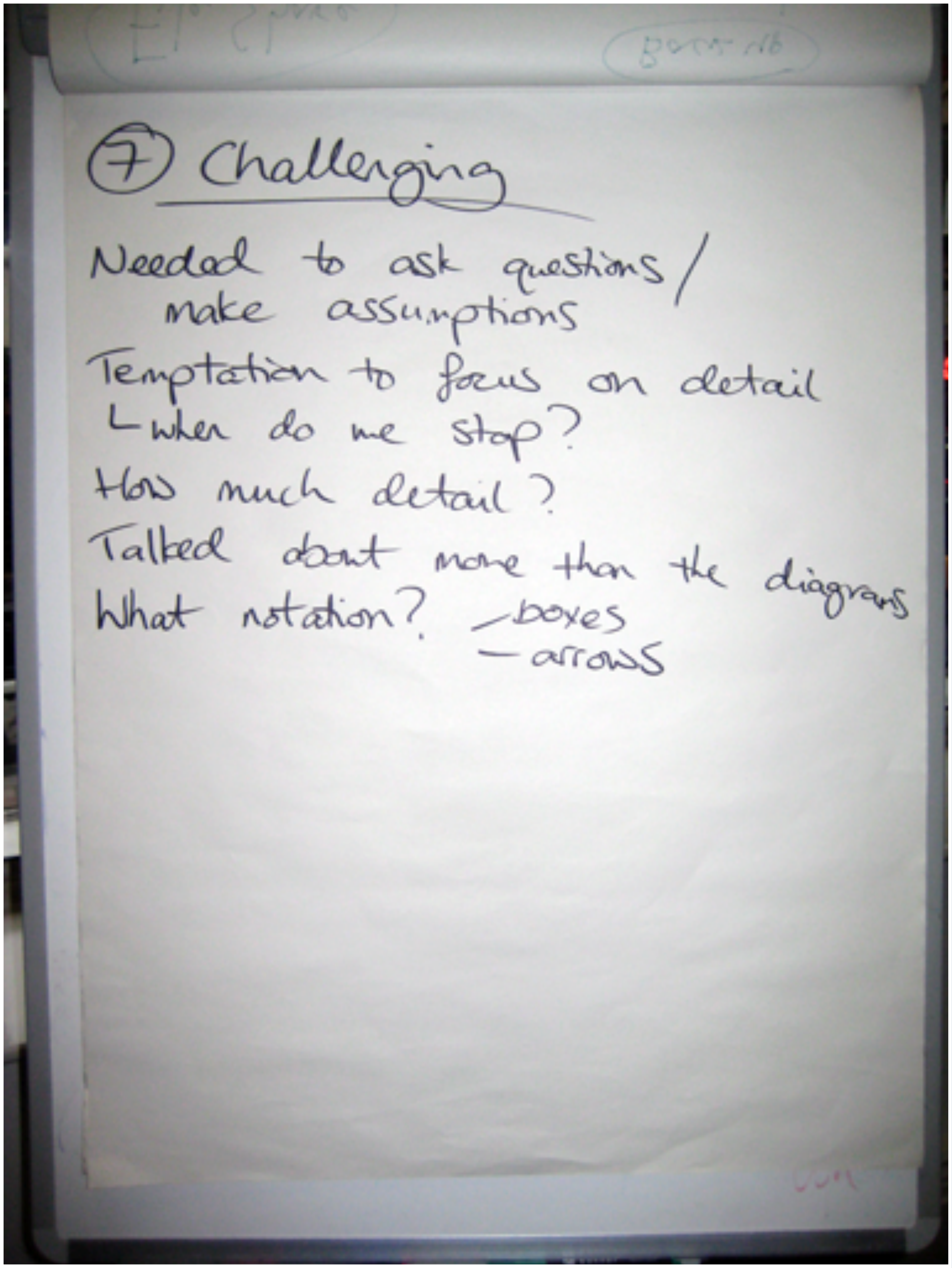


10 Challenging?

Verifying our own assumptions

Expressing the solution

- communicating it in a clear way
- use of notation
- easy to mix levels of dostraction
- how much detail?



7 Challenging

Needed to ask questions / make assumptions

Temptation to focus on detail

└ when do we stop?

How much detail?

Talked about more than the diagrams

What notation? – boxes
– arrows

Swap your diagrams with another group

Review the diagrams

Focus on the diagrams: notation, colour coding, symbols, etc

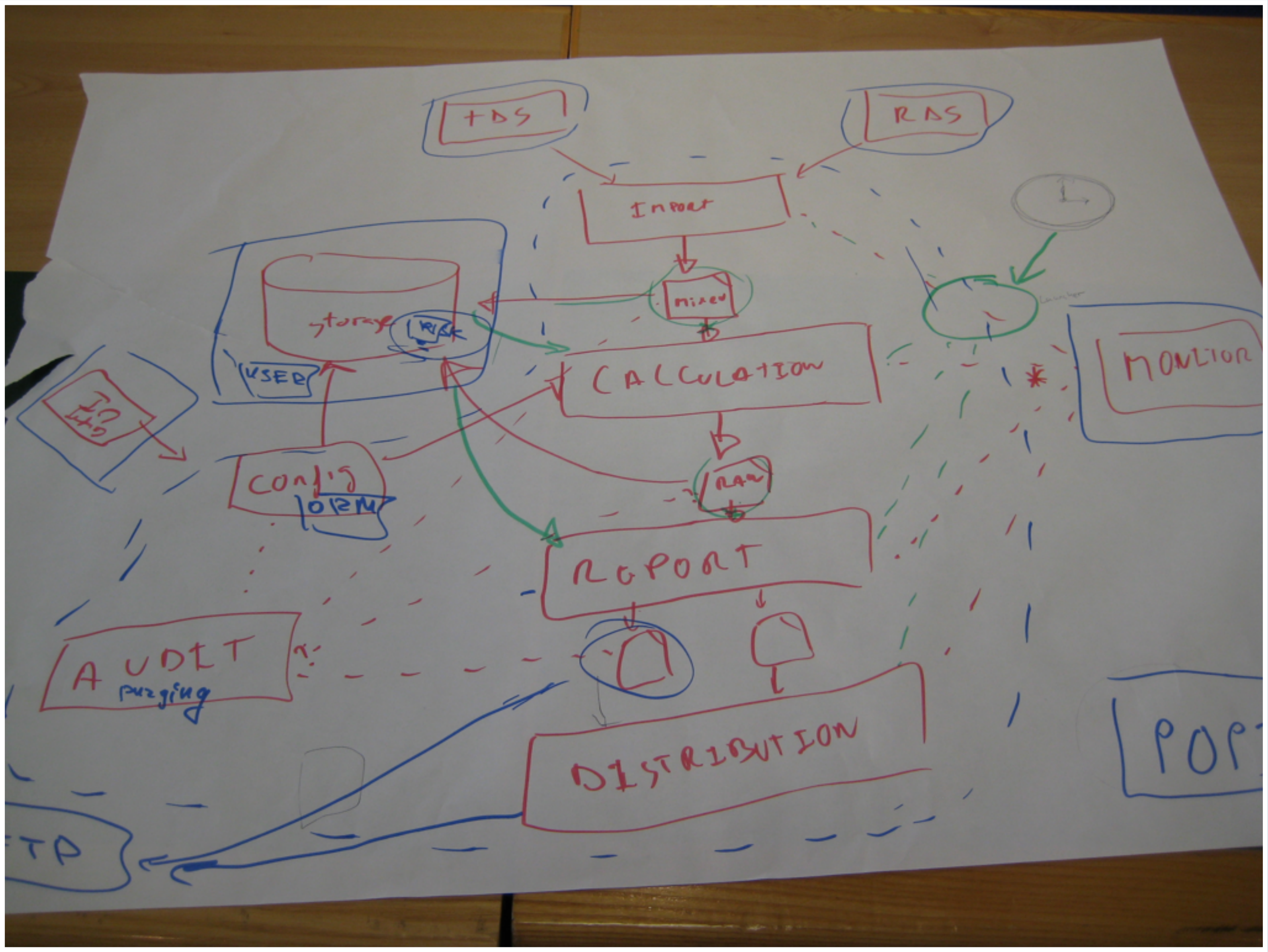
3 things you like 🟩🟩🟩

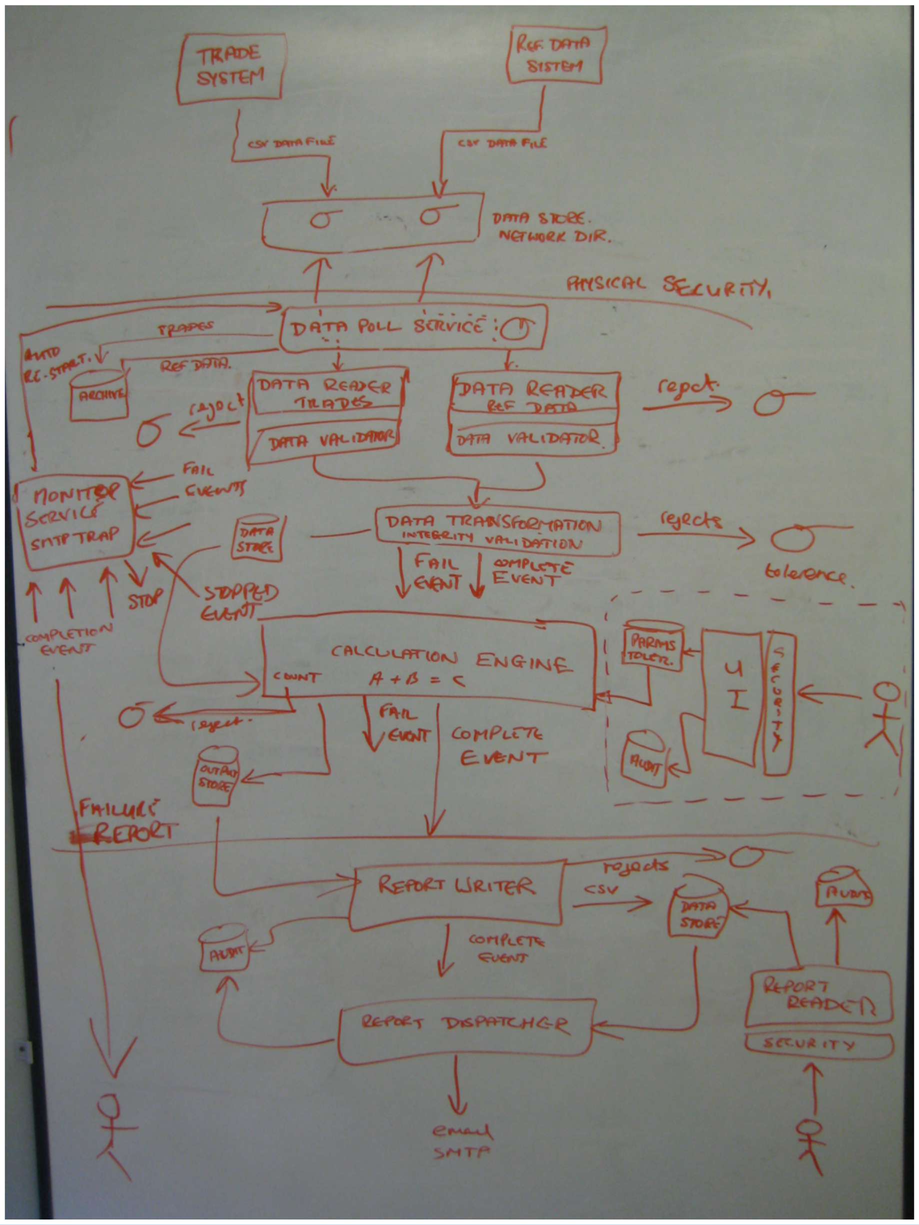
3 things that could be improved 🟥🟥🟥

A score between 1-10 🟨

1 1 1

2 2 2

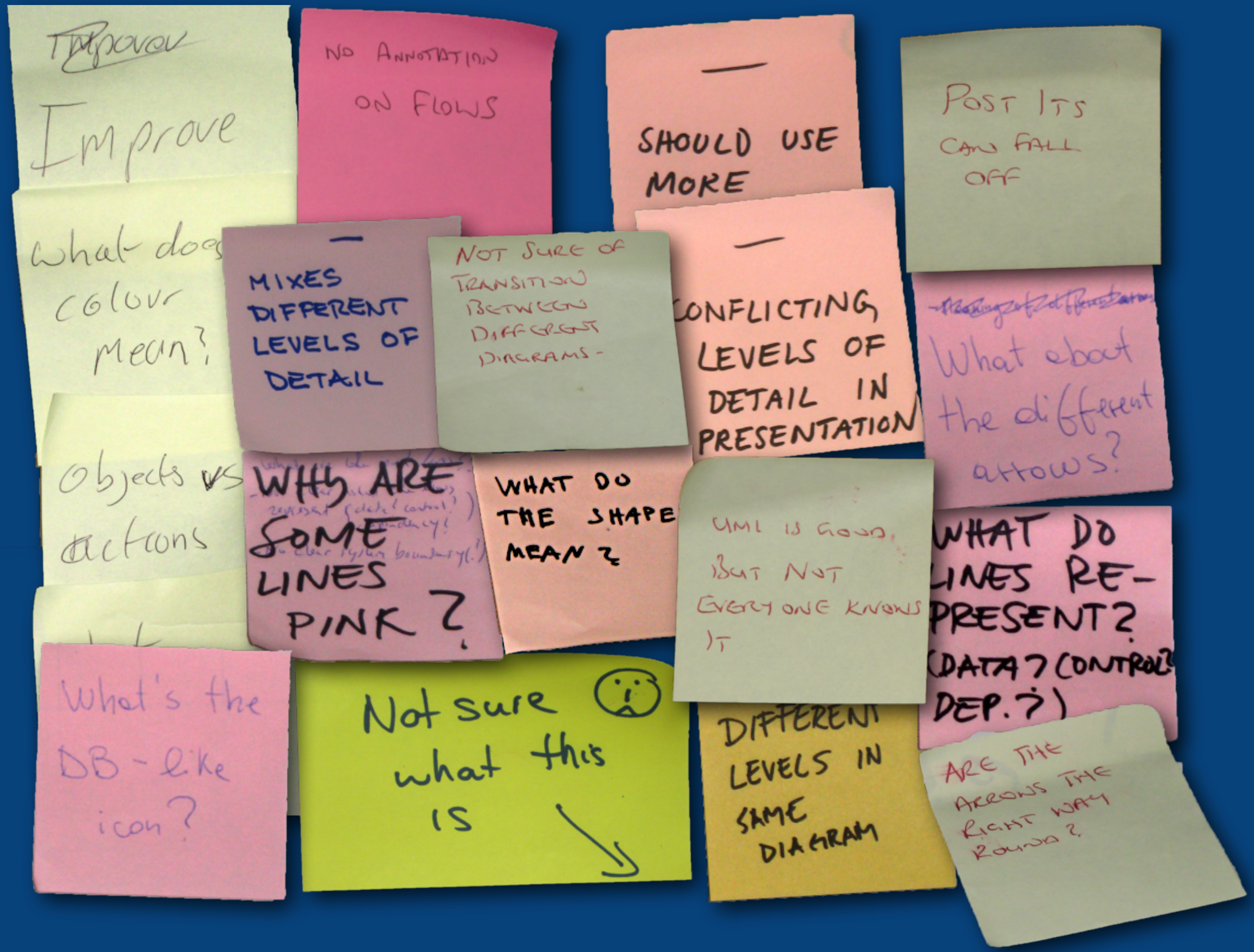

tDS
RDS
Import
mixed
storage
RISK
USER
CALCULATION
MONITOR
config
ORM
RAW
REPORT
AUDIT
purging
DISTRIBUTION
FTP
POP


TRADE SYSTEM
REF DATA SISTEM
CSV DATA FILE
CSV DATA FILE
DATA STORE. NETWORK DIR.
PHYSICAL SECURITY.
DATA POLL SERVICE
TRADES
REF DATA.
AUTO RE-START.
ARCHIVE
DATA READER TRADES
DATA VALIDATOR
DATA READER REF DATA
DATA VALIDATOR
reject
reject.
FAIL EVENTS
MONITOR SERVICE SMTP TRAP
DATA STORE
DATA TRANSFORMATION INTEGRITY VALIDATION
rejects
tolerance.
FAIL EVENT
COMPLETE EVENT
STOP
STOPPED EVENT
COMPLETION EVENT
CALCULATION ENGINE
COUNT
A + B = C
PARMS TOLER.
UI
SECURITY
AUDIT
reject.
FAIL EVENT
COMPLETE EVENT
OUTPUT STORE
FAILURE REPORT
REPORT WRITER
rejects
CSV
DATA STORE
AUDIT
AUDIT
COMPLETE EVENT
REPORT DISPATCHER
REPORT READER
SECURITY
email SMTP

Information is likely
still stuck in your heads

This doesn’t make sense,
but we’ll explain it.

- What is this shape/symbol?
- What is this line/arrow?
- What do the colours mean?
- What level of abstraction is shown?
- Which diagram do we read first?


Improve
No annotation on flows
— Should use more
Post Its can fall off
What does colour mean?
— Mixes different levels of detail
Not sure of transition between different diagrams -
— Conflicting levels of detail in presentation
What about the different arrows?
Objects vs actions
Why are some lines pink?
What do the shape mean?
UML is good, but not everyone knows it
What do lines represent? (Data? Control? Dep.?)
What's the DB-like icon?
Not sure what this is
Different levels in same diagram
Are the arrows the right way round?

FUNCTIONAL VIEW

File Retriever

Scheduler

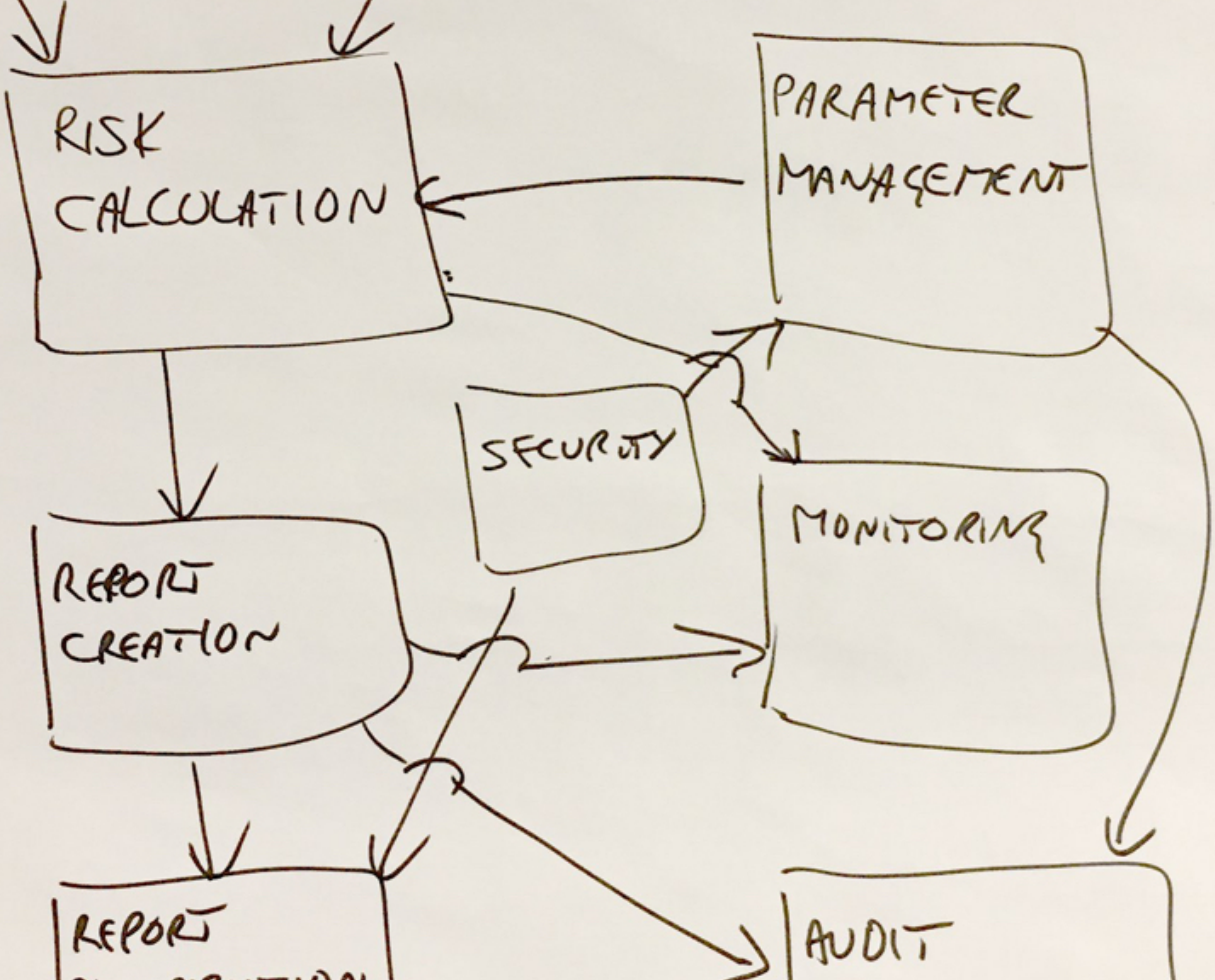
Auditing

Reference Archiver

Risk Assesment Processor

Risk Parameter Configuration

Report

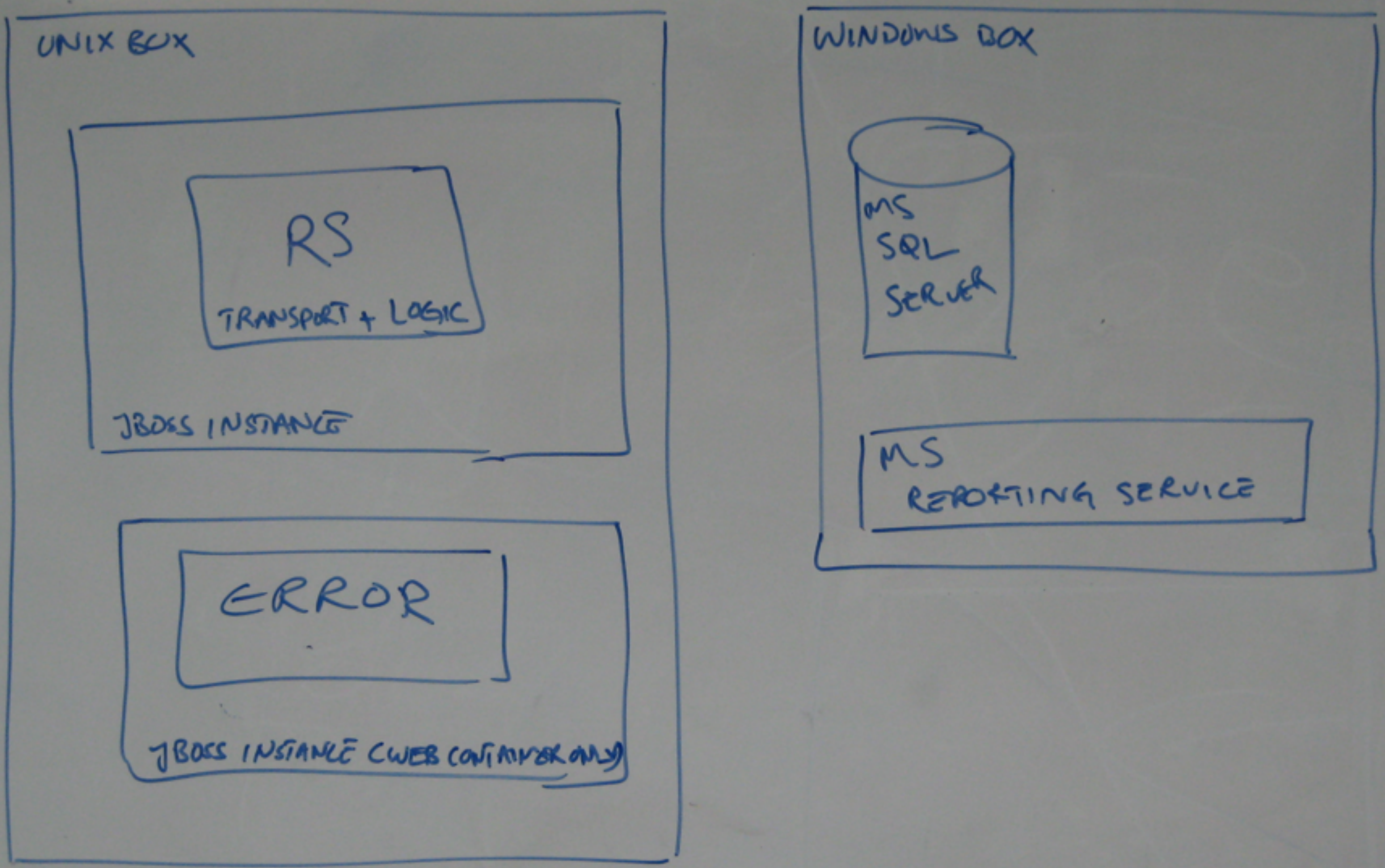

RISK CALCULATION
PARAMETER MANAGEMENT
SECURITY
MONITORING
REPORT CREATION
REPORT
AUDIT

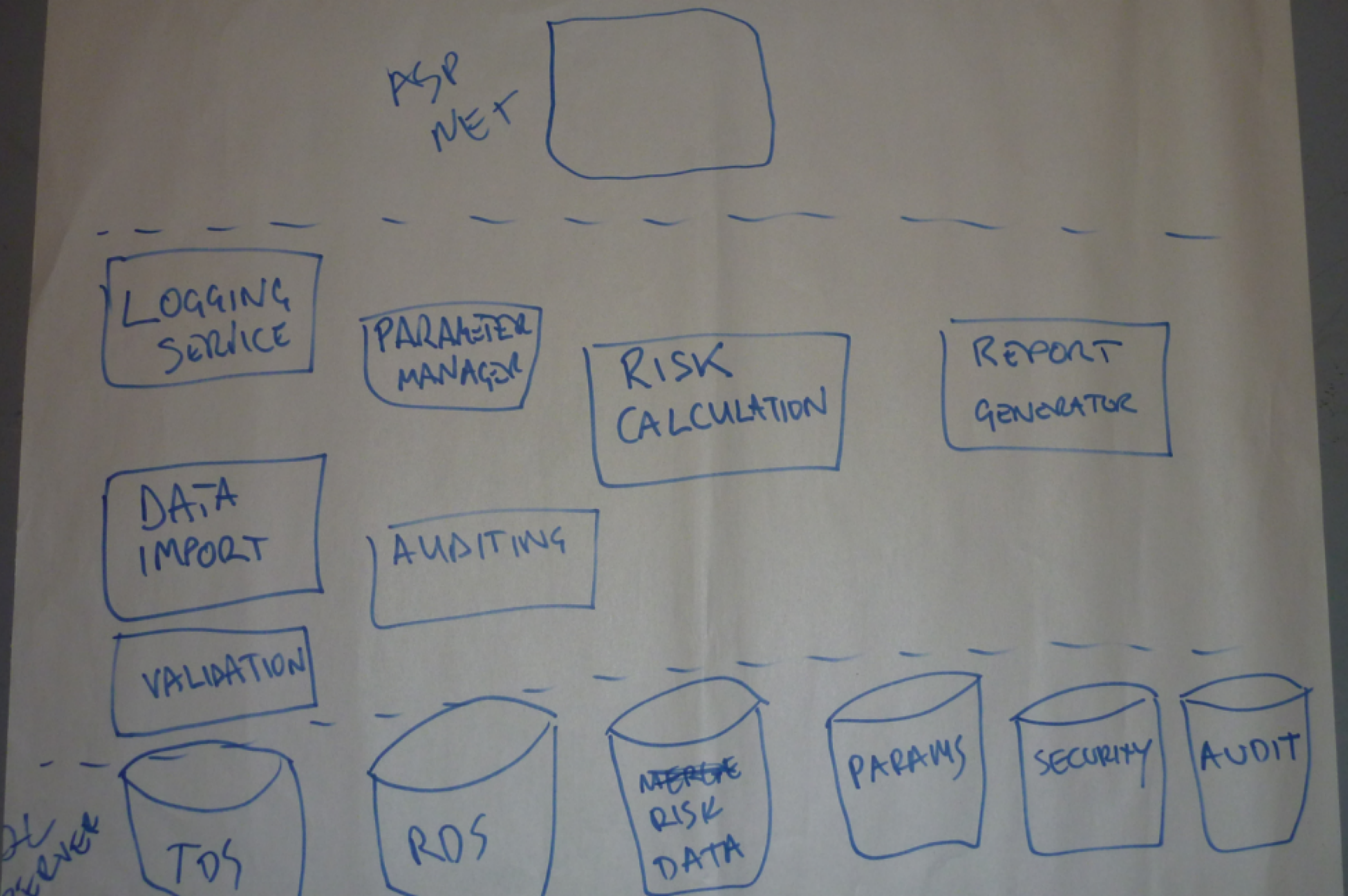
TDS
RDS
trade report
ref report
RS
TRANSPORT
ARCHIVE
S
action
data
error
error
A U D I T
MS
BUS. LOGIC
action
error
E R R O R
MONITOR
REPORT GEN
action
error
S
S
scheduler
config
AL

Import
mixed
storage
RISK
USER
Git UI
config
ORM
CALCULATION
RAW
REPORT
AUDIT
purging
Launcher
TION

DATA POLL SERVICE
TRADES
AUTO RE-START.
REF DATA.
ARCHIVE
DATA READER TRADES
DATA VALIDATOR
reject
DATA READER REF DATA
DATA VALIDATOR.
reject.
FAIL EVENTS
MONITOR SERVICE SMTP TRAP
DATA STORE
DATA TRANSFORMATION INTEGRITY VALIDATION
rejects
tolerence.
FAIL EVENT
COMPLETE EVENT
STOP
STOPPED EVENT
COMPLETION EVENT
CALCULATION ENGINE
COUNT
A + B = C
PARMS TOLER.
UI
SECURITY
AUDIT
reject.
FAIL EVENT
COMPLETE EVENT
OUTPUT STORE
FAILURE

2
SERVICE/BATCH
IMPORT
JOIN
CALC
EXPORT/STORE
REPORT
SNMP
CENTRAL
MONITORING
SERVICE
3
WEBAPP
•GET REPORT
•EDIT
RISK RULES
•AUTH
•AUDIT
RISK RULES


UNIX BOX
RS
TRANSPORT + LOGIC
JBOSS INSTANCE
ERROR
JBOSS INSTANCE (WEB CONTAINER ONLY)
WINDOWS BOX
MS SQL SERVER
MS REPORTING SERVICE

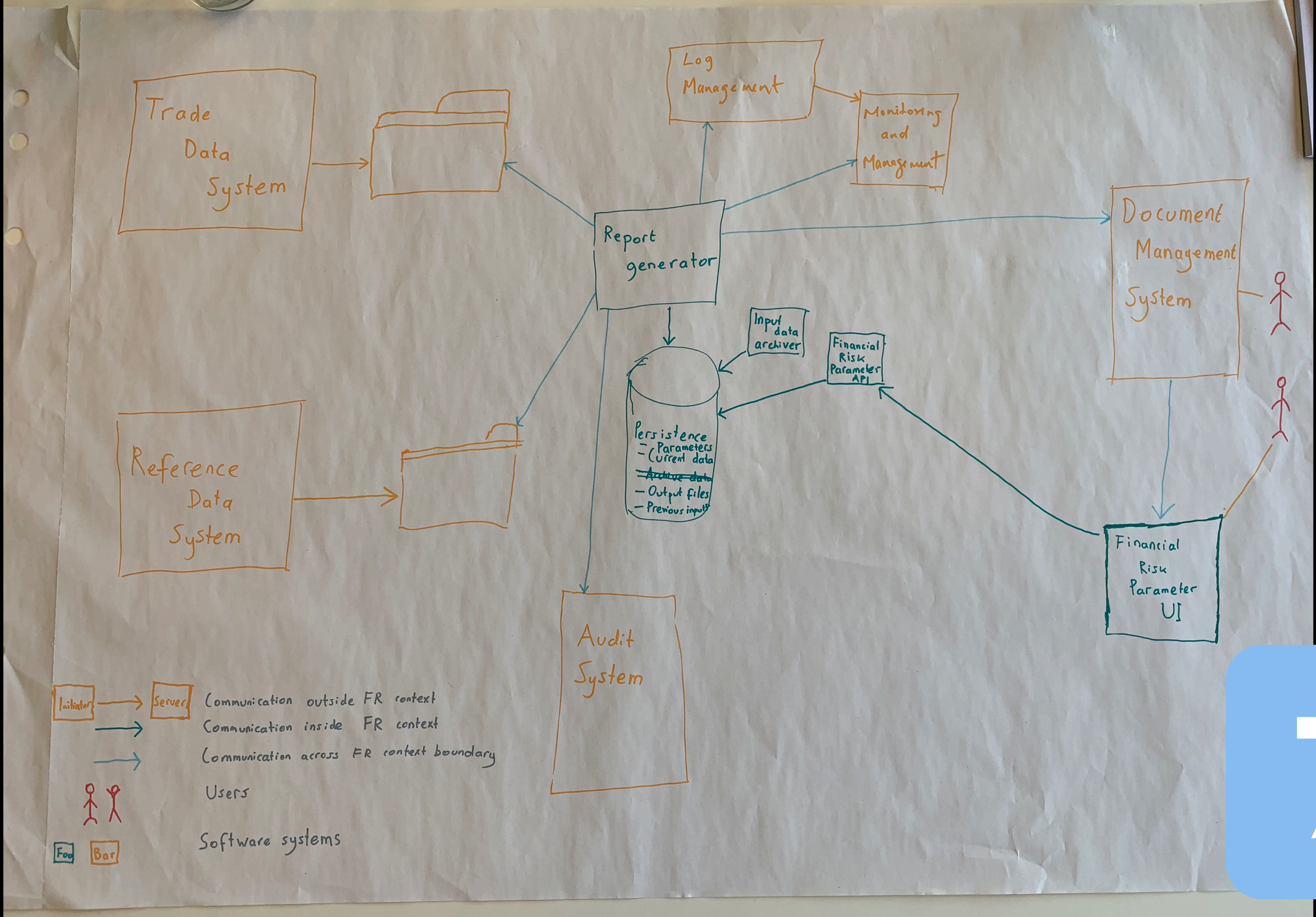

ASP NET
LOGGING SERVICE
PARAMETER MANAGER
RISK CALCULATION
REPORT GENERATOR
DATA IMPORT
AUDITING
VALIDATION
QL SERVER
TDS
RDS
~~MERGE~~ RISK DATA
PARAMS
SECURITY
AUDIT

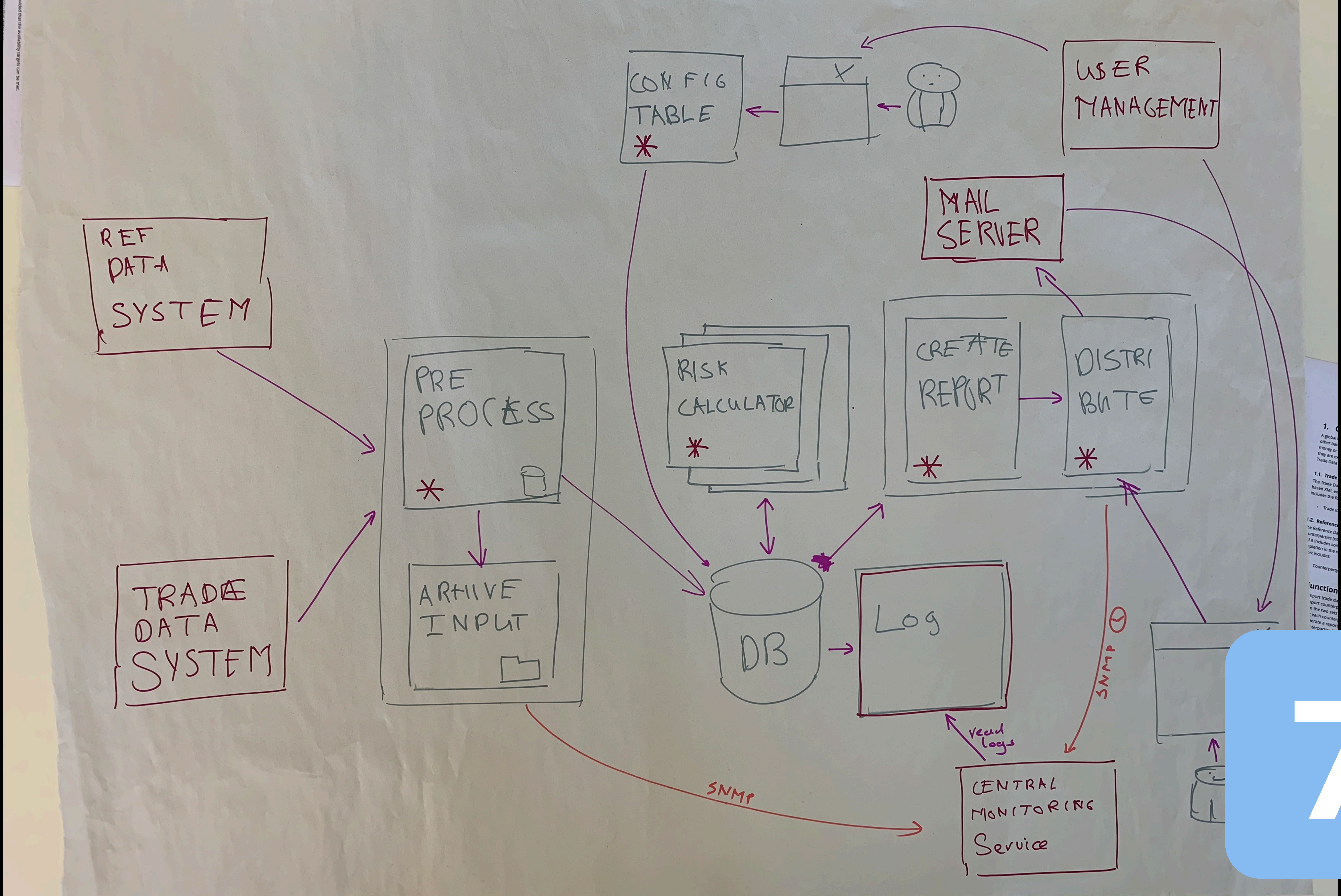
WIN SERVER
APPLICATION
CONSOLE - C#
SCHEDULER
CONSOLE - C#
DATA LOADER
C#
CALCULATION MANAGER
C#
NOTIFICATION MANAGER
C#
EXTERNAL SYSTEMS
SNMP SERVICE
RDS
IPS
XML
check if exists
DATA MERGE
C#
XPath
XML READER
C#
RISK CALCULATION
C#
NOTIFICATION SERVICE
C#
SMTP
parameters
report to DB table
LOGGER
nLog
SNMP TRAP GENERATOR
DATA ACCESS
SQL SERVER

ENGINE
APP SERVER
WHICH DB TECH?
RING
REPORTING SERVICE
WEB SERVER

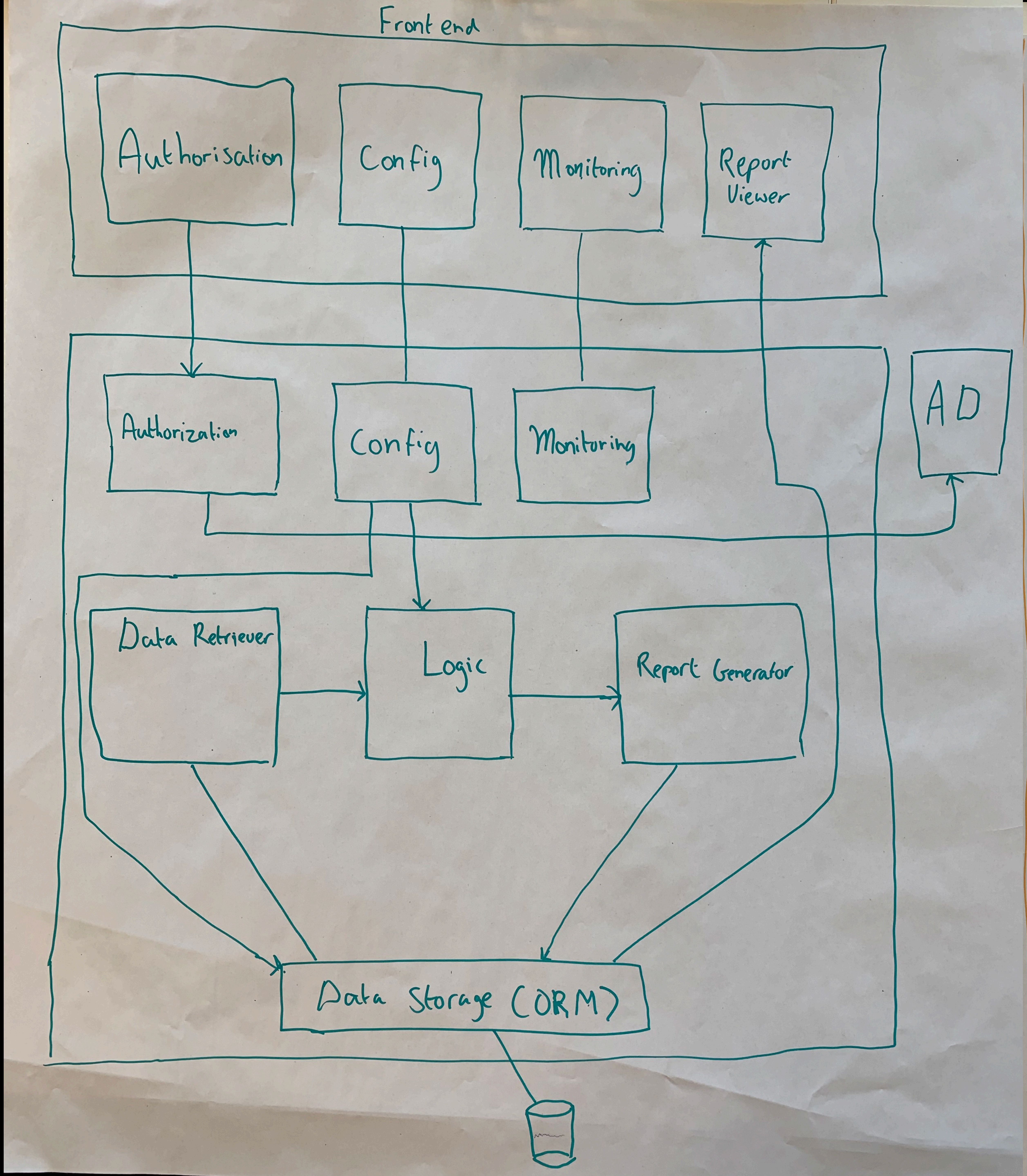
Data Normalizer
Report Monitor
(other machine)
Risk ~~Calculator~~
Aggregator
CP Risk Calc
Report Generator
Auditing Service
Report Publisher
Report Repository

ret_curr
Params
Calcs
Error
App Log
Dag_Calcs_Riskoutpts
EH?
App
Error Log
Date
- RiskSel
- calcs
- Par
Batch
Params_Distrib
Detail
B_id:


Trade Data System
Log Management
Monitoring and Management
Report generator
Document Management System
Input data archiver
Financial Risk Parameter API
Persistence
- Parameters
- Current data
- ~~Archive data~~
- Output files
- Previous input
Reference Data System
Financial Risk Parameter UI
Audit System
Initiator
Server
Communication outside FR context
Communication inside FR context
Communication across FR context boundary
Users
Foo
Bar
Software systems

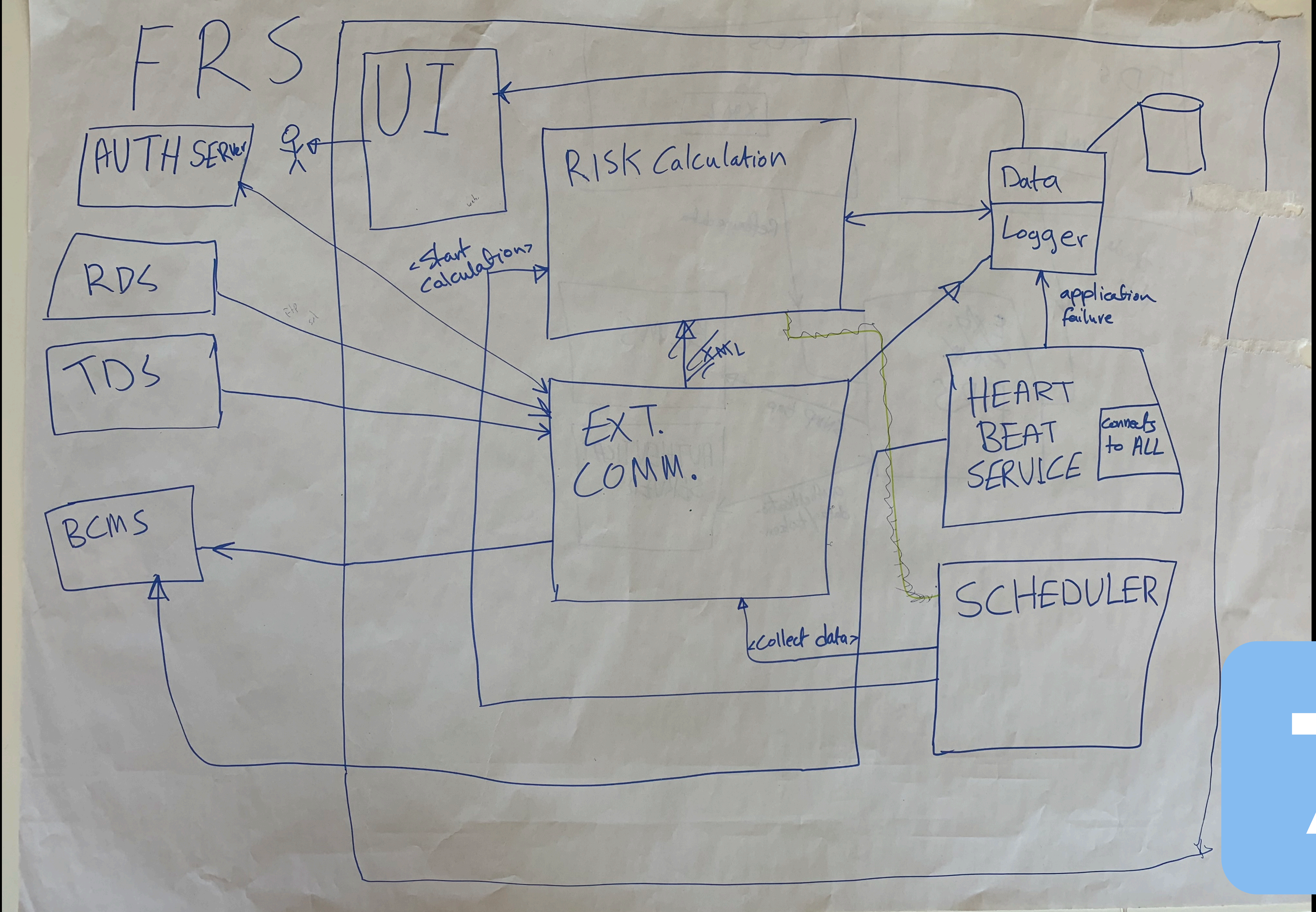

CONFIG TABLE
USER MANAGEMENT
MAIL SERVER
REF DATA SYSTEM
PRE PROCESS
RISK CALCULATOR
CREATE REPORT
DISTRIBUTE
TRADE DATA SYSTEM
ARHIVE INPUT
DB
Log
SNMP
SNMP
read logs
CENTRAL MONITORING Service

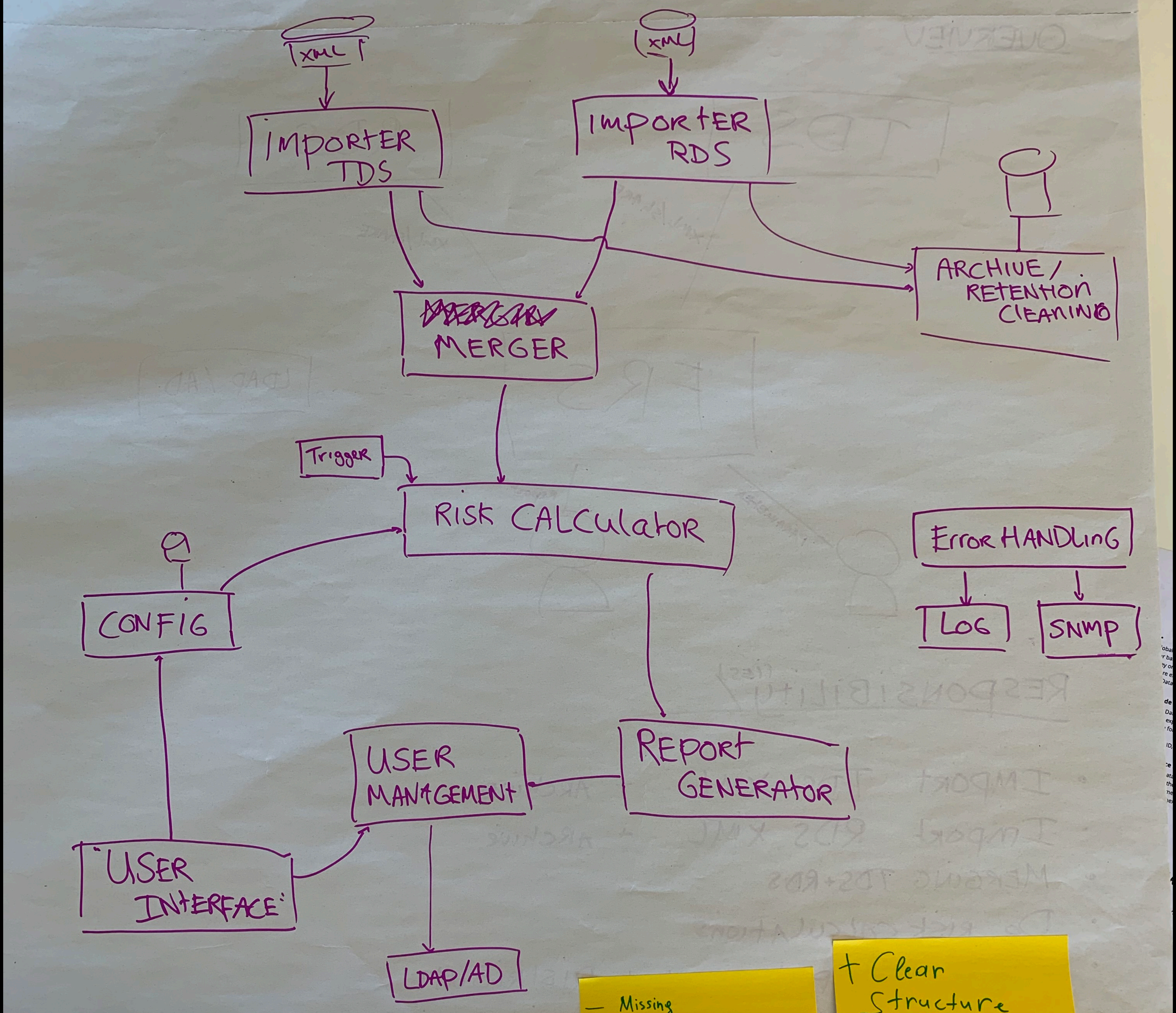
7



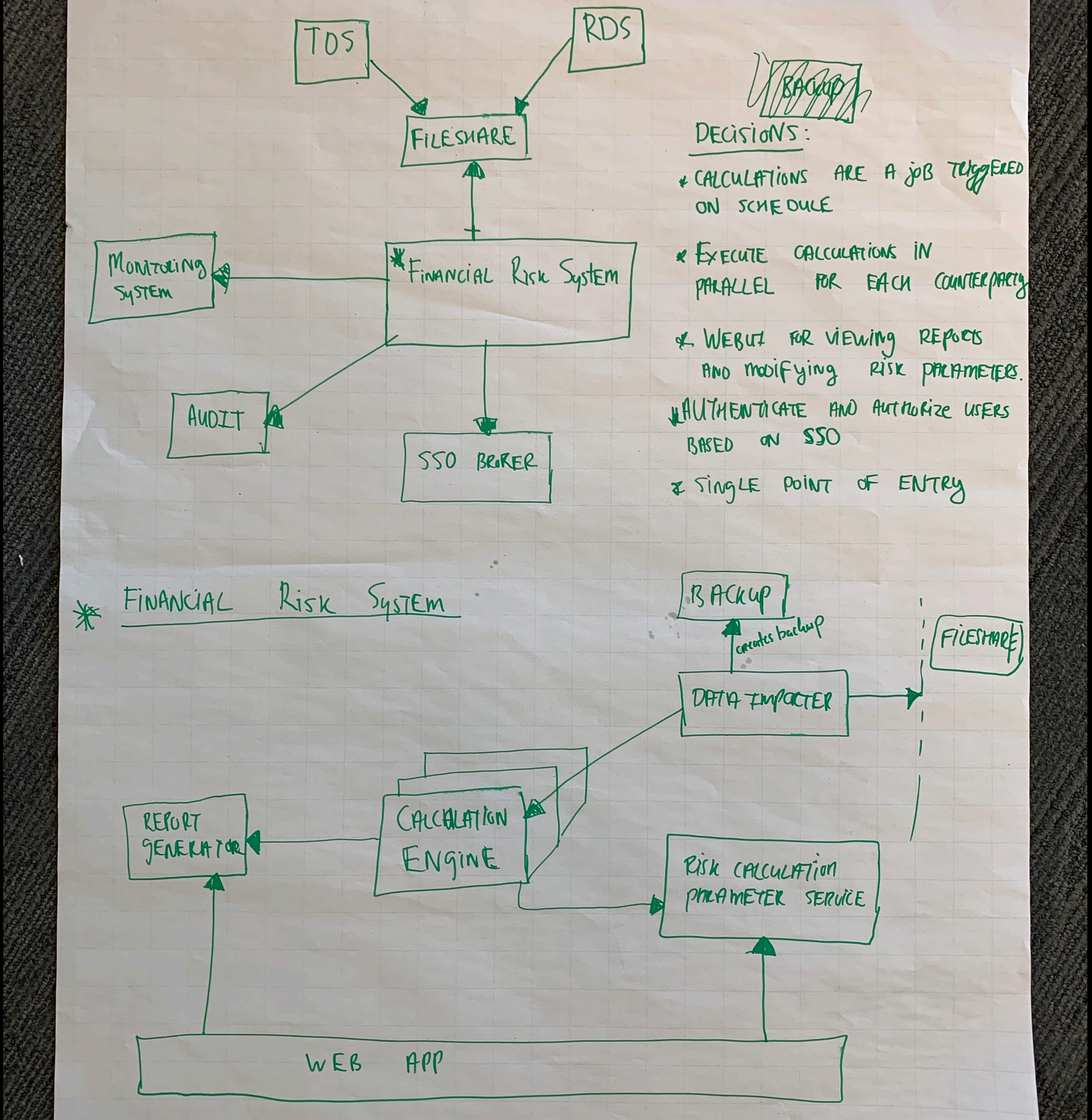
Significant decisions

- F/E <> B/E
- Make use of OS' watchdog mechanism
- Data storage ORM· framework: Entity
- ASP .NET B/E
- Angular F/E


FRS
UI
AUTH SERVER
RDS
TDS
BCMS
RISK Calculation
<Start calculation>
XML
EXT. COMM.
<collect data>
Data
Logger
application failure
HEART BEAT SERVICE
connects to ALL
SCHEDULER


XML
XML
Importer TDS
Importer RDS
Archive / Retention Cleaning
Merger
Trigger
Risk Calculator
Error Handling
Log
SNMP
Config
User Management
Report Generator
"User Interface"
LDAP/AD
– Missing
+ Clear Structure

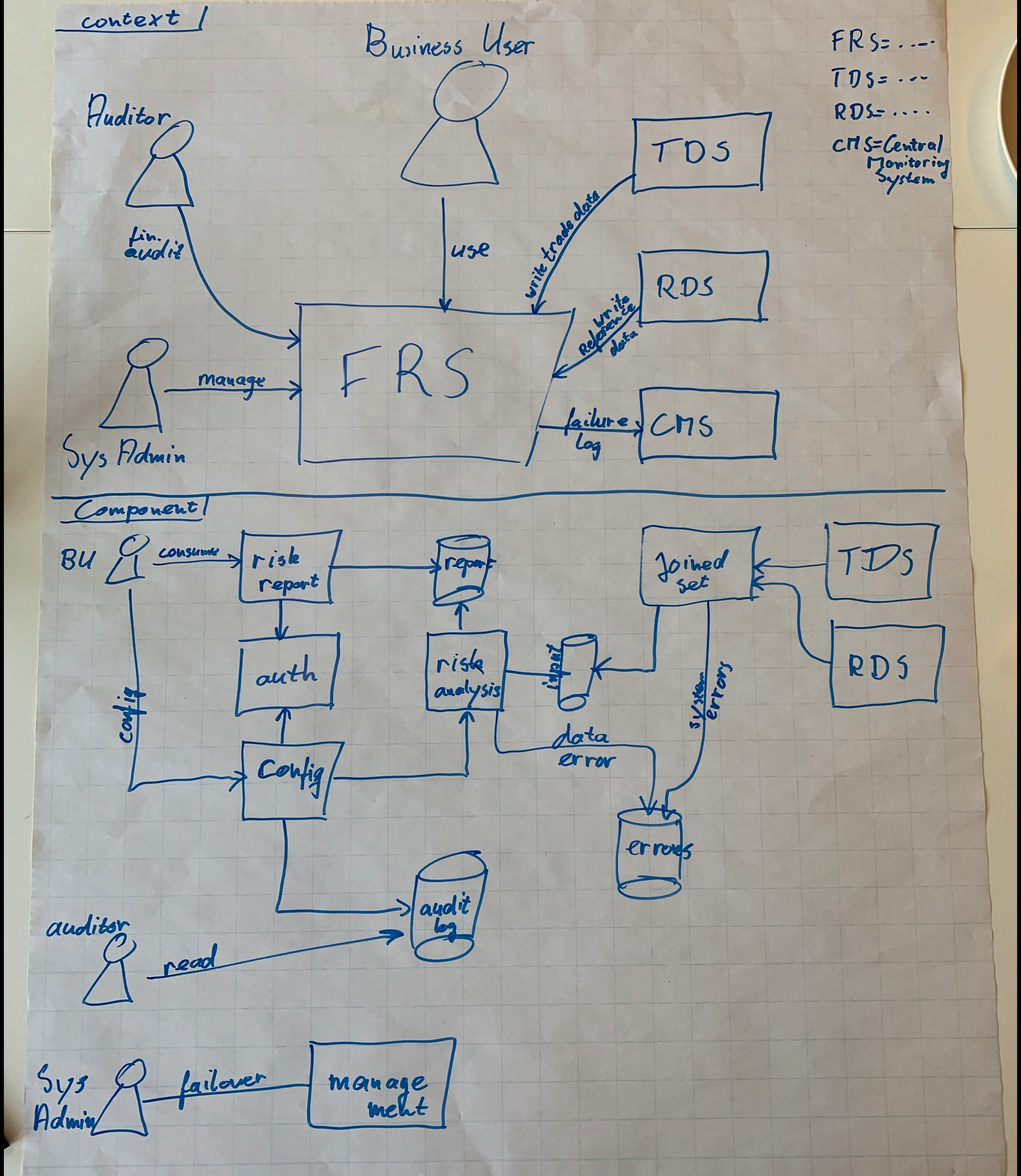
7



DECISIONS:

- CALCULATIONS ARE A JOB TRIGGERED ON SCHEDULE
- EXECUTE CALCULATIONS IN PARALLEL FOR EACH COUNTERPARTY
- WEBUI FOR VIEWING REPORTS AND MODIFYING RISK PARAMETERS.
- AUTHENTICATE AND AUTHORIZE USERS BASED ON SSO
- SINGLE POINT OF ENTRY

7


context
Business User
Auditor
fin. audit
use
TDS
write trade data
RDS
write Reference data
FRS
manage
Sys Admin
failure log
CMS
FRS= ....
TDS= ...
RDS= ....
CMS=Central Monitoring System
Component
BU
consume
risk report
report
Joined set
TDS
auth
risk analysis
input
system errors
RDS
config
Config
data error
errors
audit log
auditor
read
Sys Admin
failover
manage ment

6

software architecture diagram

All · Images · Videos · News · Books · Maps · Web · More · Tools

Software engineering · Visio · System · Uml · Design · Simple · Azure · Component · Tool · Layered · Api · Game

Software Architecture Diagrams
Software Architectural Patterns

Miro
Software Architecture Diagramming

Edrawsoft
Software Architecture Diagram | Ed...

Nulab
What is an architecture diagram, ...

Visual Paradigm Online
Software Architecture Diagram | Visual ...

Medium
Top 9 Architecture diagram software for ...

YouTube
Create Software Architecture Diagrams ...

Lucidchart
Draw 5 Types of Architectural D...

Edrawsoft
Application Architecture Diagram: A ...

ResearchGate
Instrumentation Software Architect...

SlideModel
Four Layers Modern Web Application ...

Lucidchart
Draw 5 Types of Architectural Diagrams ...

Red Hat
5 great diagramming tools for ...

IcePanel - Medium
Top 8 diagramming tools for software ...

LaTeX Stack Exchange
creating software architecture diagram ...

ResearchGate
Software architecture di...

Stack Overflow
tools for architectural diagram ...

predic8
What is Software Architecture

LinkedIn
Software architecture diagramming and ...

If you’re going to use “boxes & lines”, at least do so in a **structured way**, using a **self-describing notation**

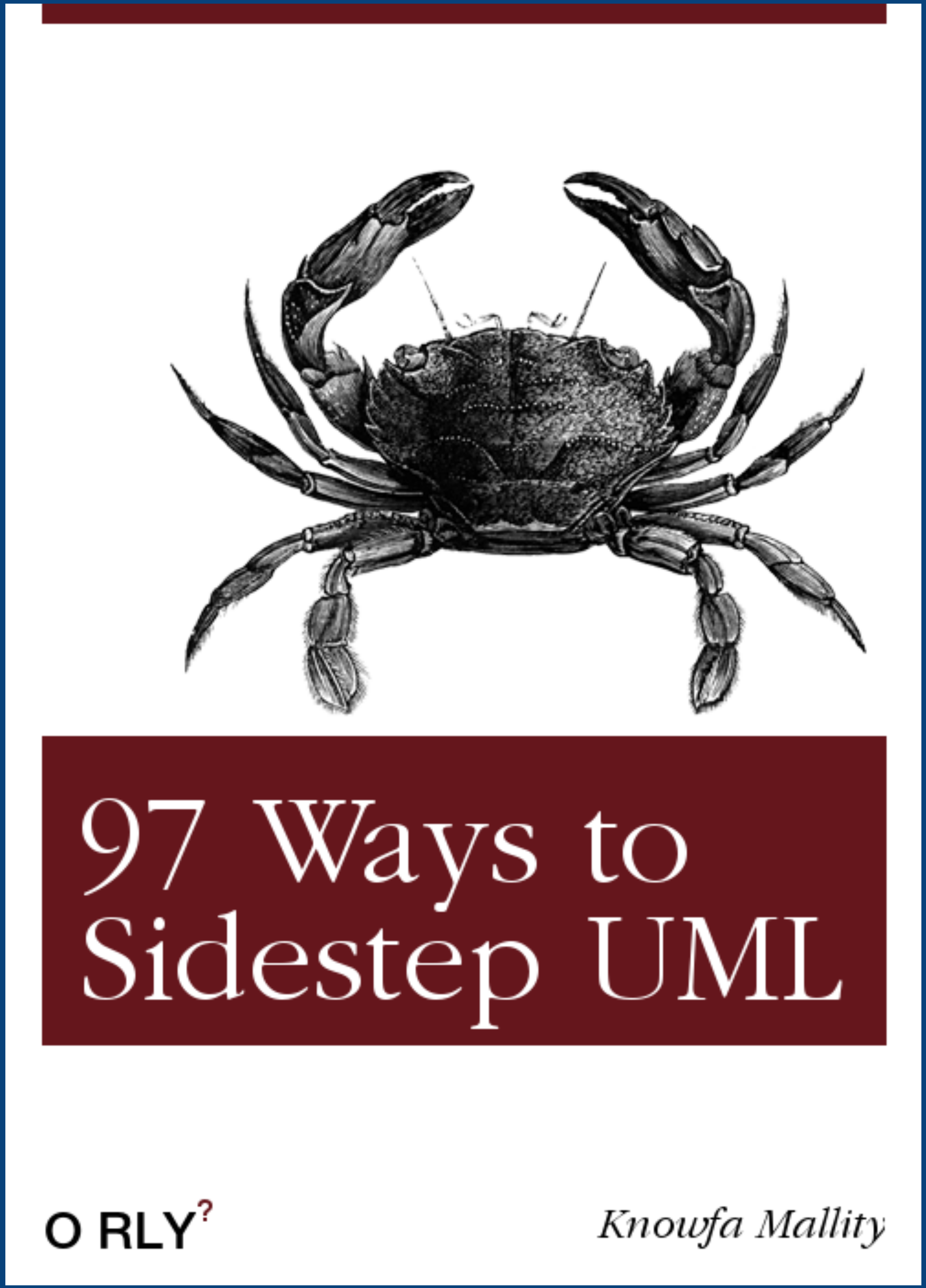
Moving fast in the same direction
as a team requires

good communication

Do **you** use UML?

In my experience,

few people use 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.”

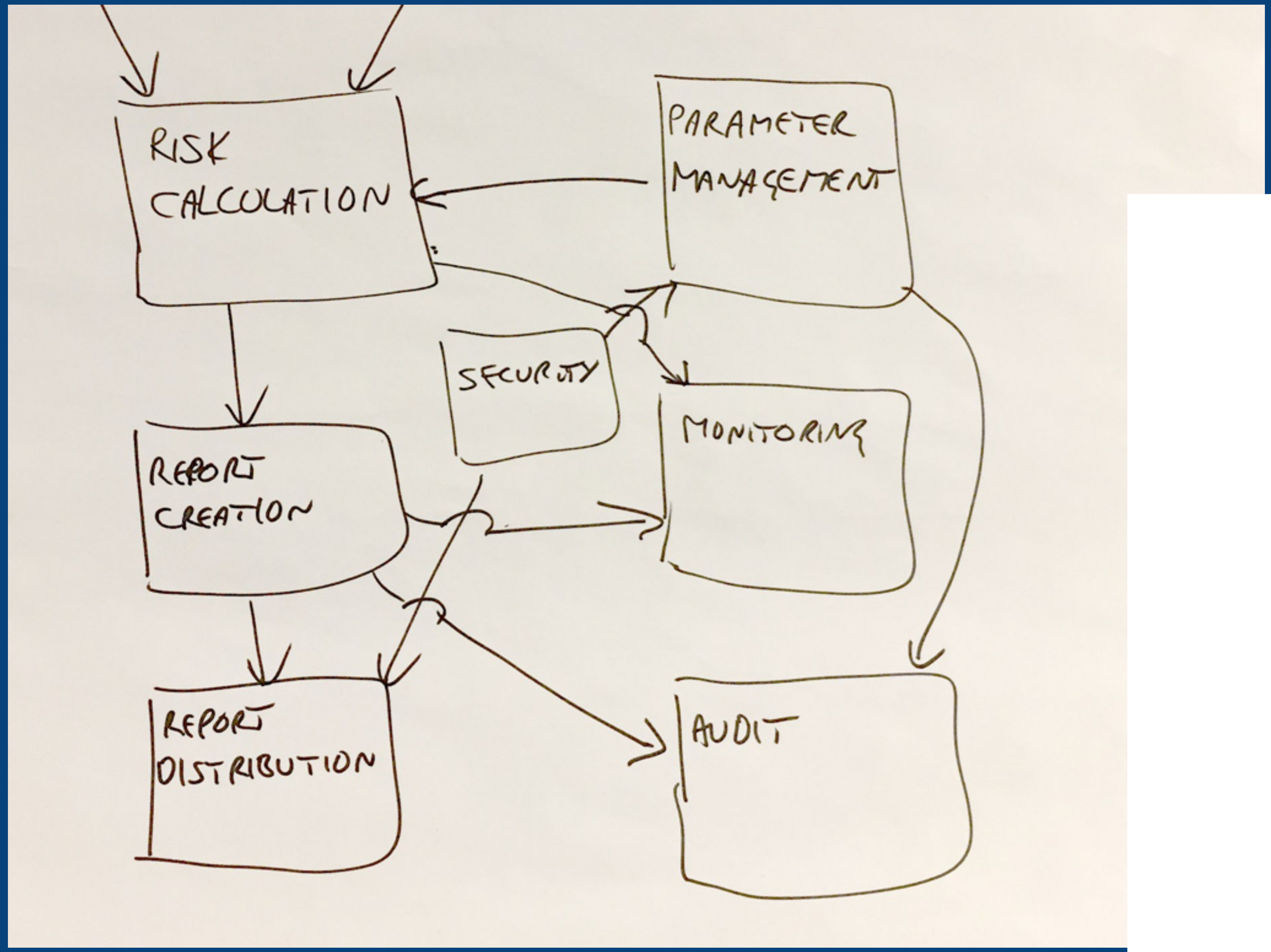
#66 “The tooling sucks.”

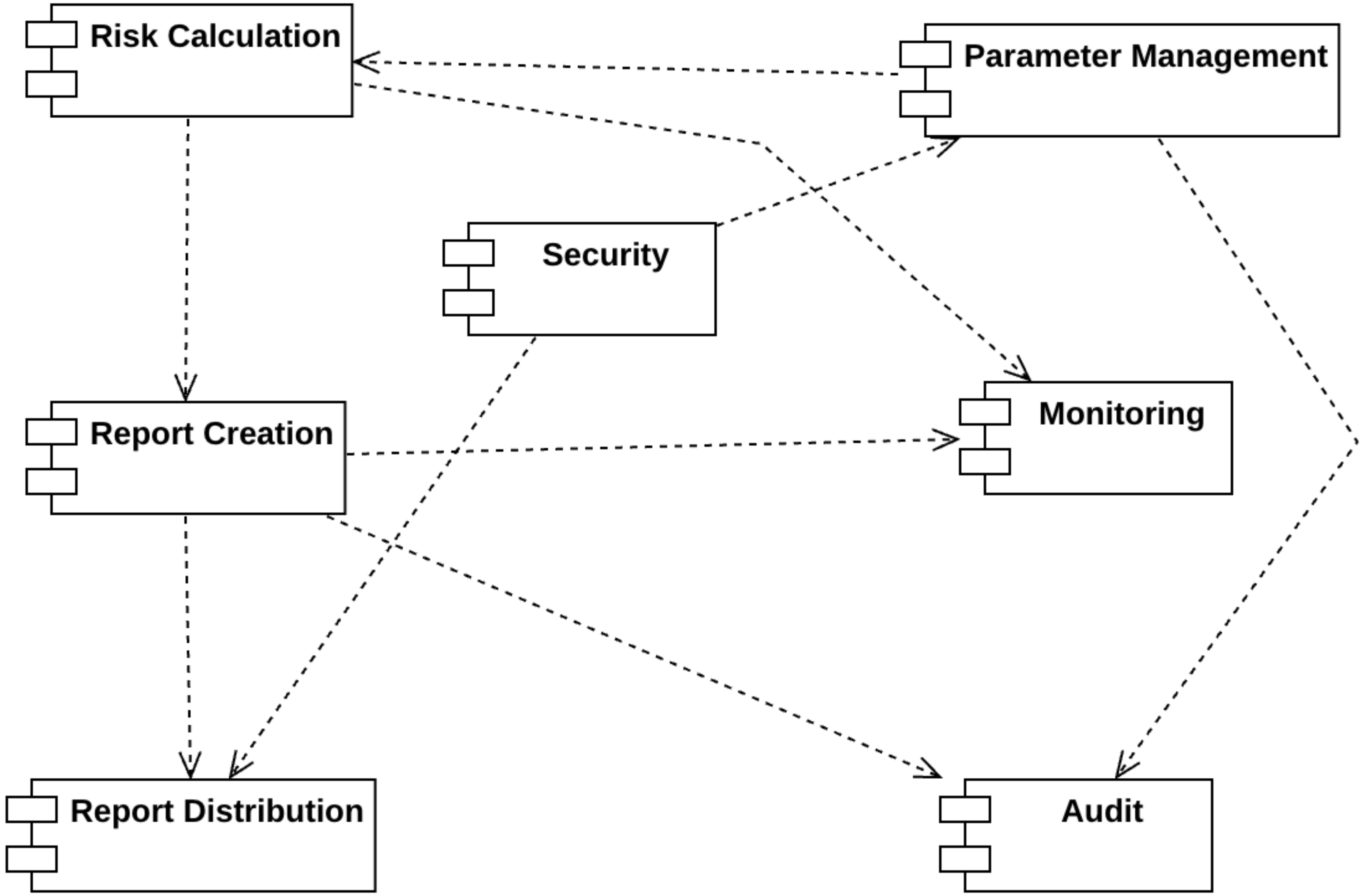
#80 “It’s too detailed.”

#81 “It’s a very elaborate waste of time.”

#92 “It’s not expected in agile.”

#97 “The value is in the conversation.”


RISK CALCULATION
PARAMETER MANAGEMENT
SECURITY
MONITORING
REPORT CREATION
REPORT DISTRIBUTION
AUDIT


Risk Calculation
Parameter Management
Security
Report Creation
Monitoring
Report Distribution
Audit

Who are the **stakeholders** that you need to communicate software architecture to; what **information** do they need?

There are many **different audiences** for diagrams and documentation, all with **different interests**

(software architects, software developers, operations and support staff, testers, Product Owners, project managers, Scrum Masters, users, management, business sponsors, potential customers, potential investors, …)

Our diagramming toolbox should include UML, ArchiMate, SysML, BPML, DFDs, ERDs, etc

To describe a software architecture, we use a model composed of multiple views or perspectives.

Architectural Blueprints - The “4+1” View Model of Software Architecture
Philippe Kruchten

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.

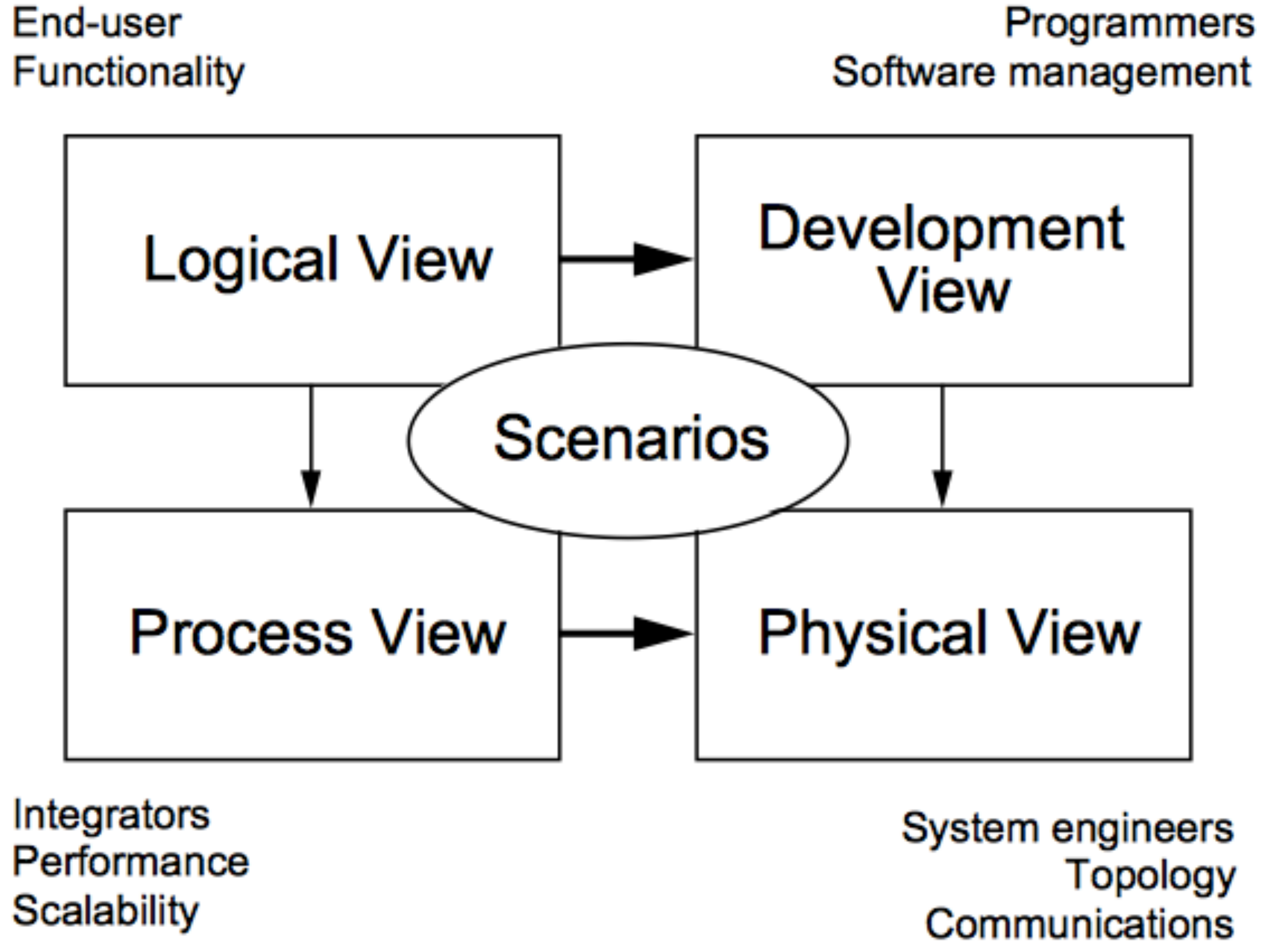
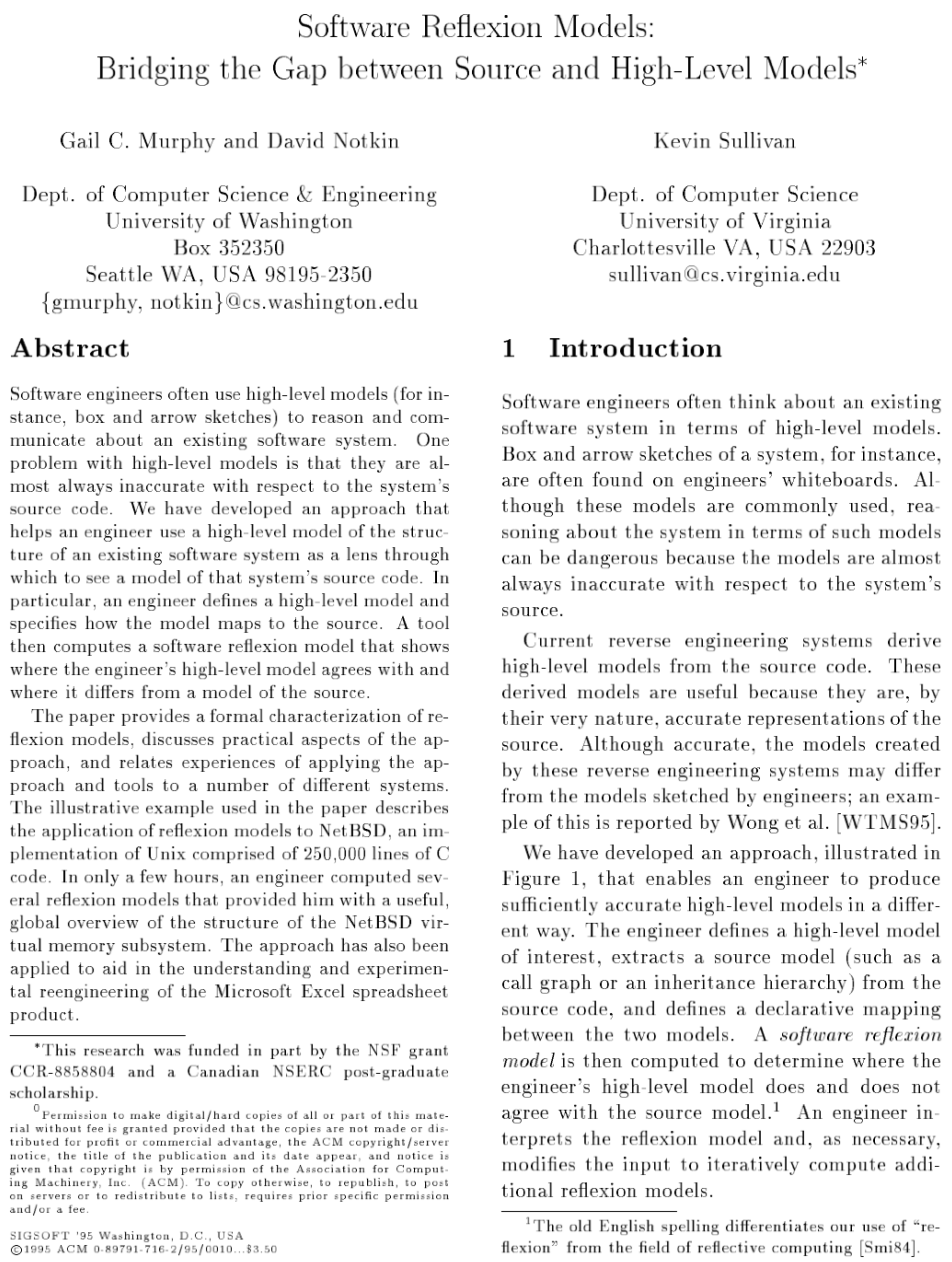


Figure 1 — The “4+1” view model

Our architecture diagrams don’t match the code.



Software Reflexion Models:
Bridging the Gap between Source and High-Level Models*

Gail C. Murphy and David Notkin

Dept. of Computer Science & Engineering
University of Washington
Box 352350
Seattle WA, USA 98195-2350
{gmurphy, notkin}@cs.washington.edu

Kevin Sullivan

Dept. of Computer Science
University of Virginia
Charlottesville VA, USA 22903
sullivan@cs.virginia.edu

Abstract

Software engineers often use high-level models (for instance, box and arrow sketches) to reason and communicate about an existing software system. One problem with high-level models is that they are almost always inaccurate with respect to the system's source code. We have developed an approach that helps an engineer use a high-level model of the structure of an existing software system as a lens through which to see a model of that system's source code. In particular, an engineer defines a high-level model and specifies how the model maps to the source. A tool then computes a software reflexion model that shows where the engineer's high-level model agrees with and where it differs from a model of the source.

The paper provides a formal characterization of reflexion models, discusses practical aspects of the approach, and relates experiences of applying the approach and tools to a number of different systems. The illustrative example used in the paper describes the application of reflexion models to NetBSD, an implementation of Unix comprised of 250,000 lines of C code. In only a few hours, an engineer computed several reflexion models that provided him with a useful, global overview of the structure of the NetBSD virtual memory subsystem. The approach has also been applied to aid in the understanding and experimental reengineering of the Microsoft Excel spreadsheet product.

1 Introduction

Software engineers often think about an existing software system in terms of high-level models. Box and arrow sketches of a system, for instance, are often found on engineers' whiteboards. Although these models are commonly used, reasoning about the system in terms of such models can be dangerous because the models are almost always inaccurate with respect to the system's source.

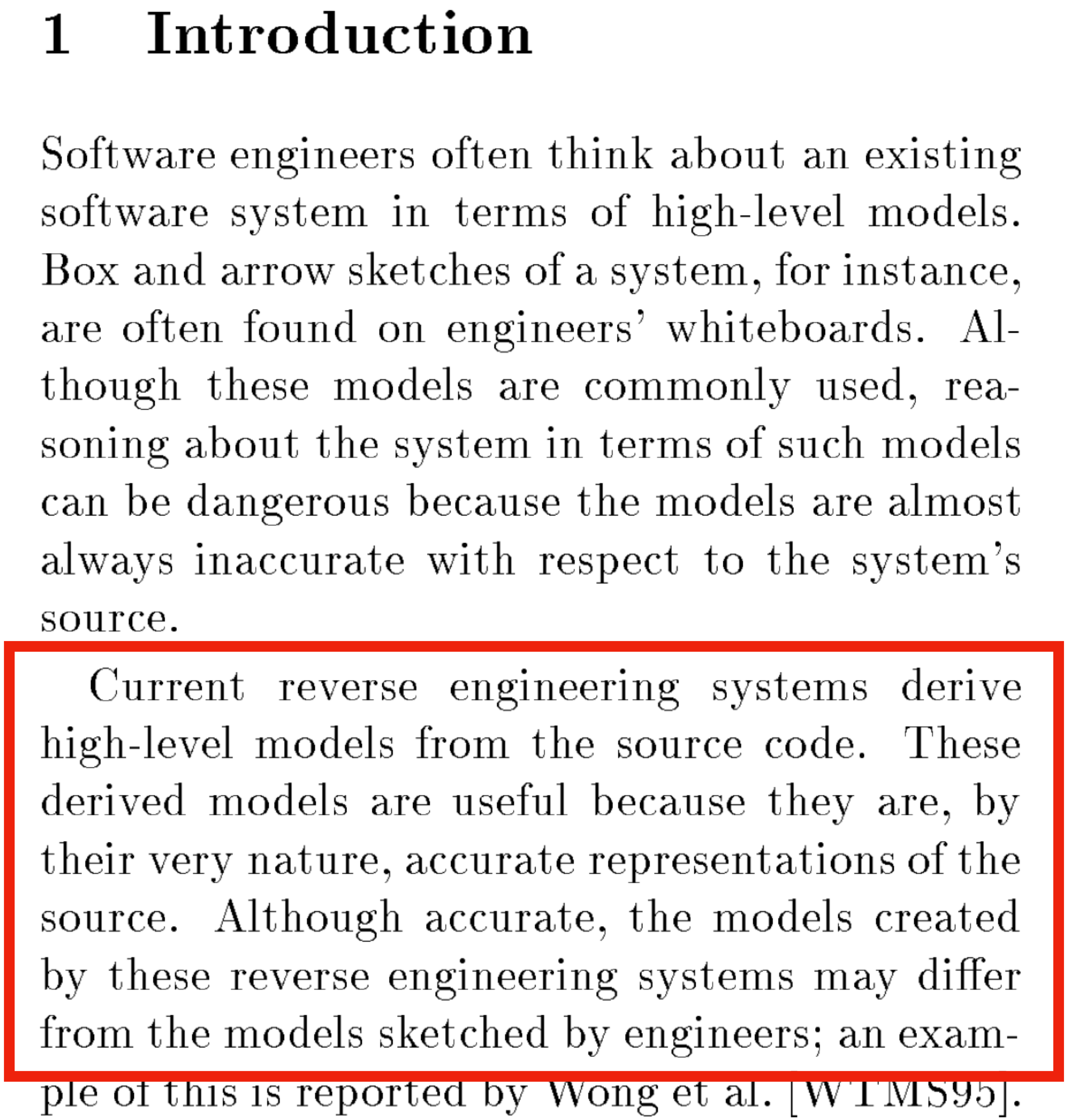
Current reverse engineering systems derive high-level models from the source code. These derived models are useful because they are, by their very nature, accurate representations of the source. Although accurate, the models created by these reverse engineering systems may differ from the models sketched by engineers; an example of this is reported by Wong et al. [WTMS95].

We have developed an approach, illustrated in Figure 1, that enables an engineer to produce sufficiently accurate high-level models in a different way. The engineer defines a high-level model of interest, extracts a source model (such as a call graph or an inheritance hierarchy) from the source code, and defines a declarative mapping between the two models. A *software reflexion model* is then computed to determine where the engineer's high-level model does and does not agree with the source model.[1] An engineer interprets the reflexion model and, as necessary, modifies the input to iteratively compute additional reflexion models.

*This research was funded in part by the NSF grant CCR-8858804 and a Canadian NSERC post-graduate scholarship.

SIGSOFT '95 Washington, D.C., USA
©1995 ACM 0-89791-716-2/95/0010...$3.50

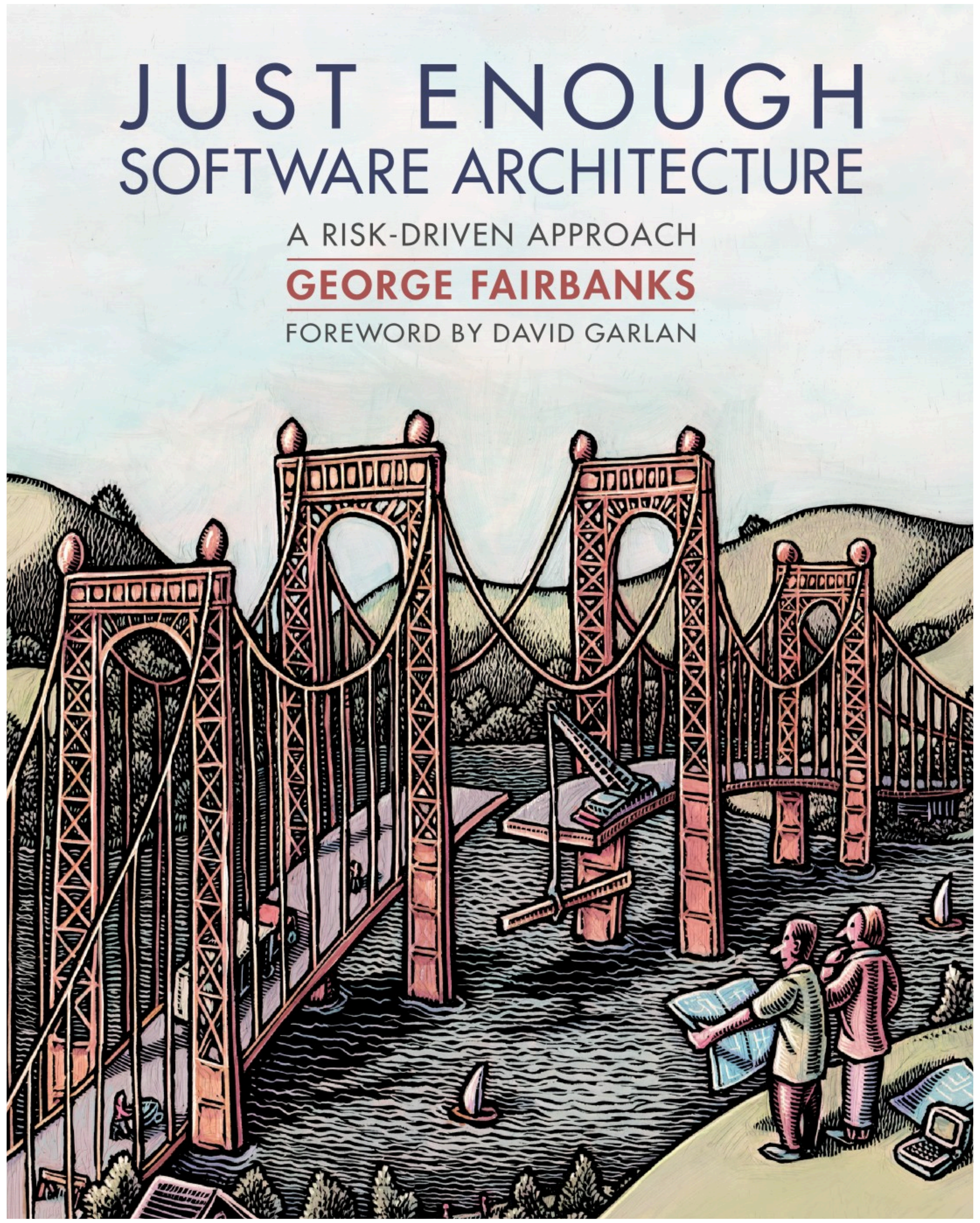
[1] The old English spelling differentiates our use of "reflexion" from the field of reflective computing [Smi84].

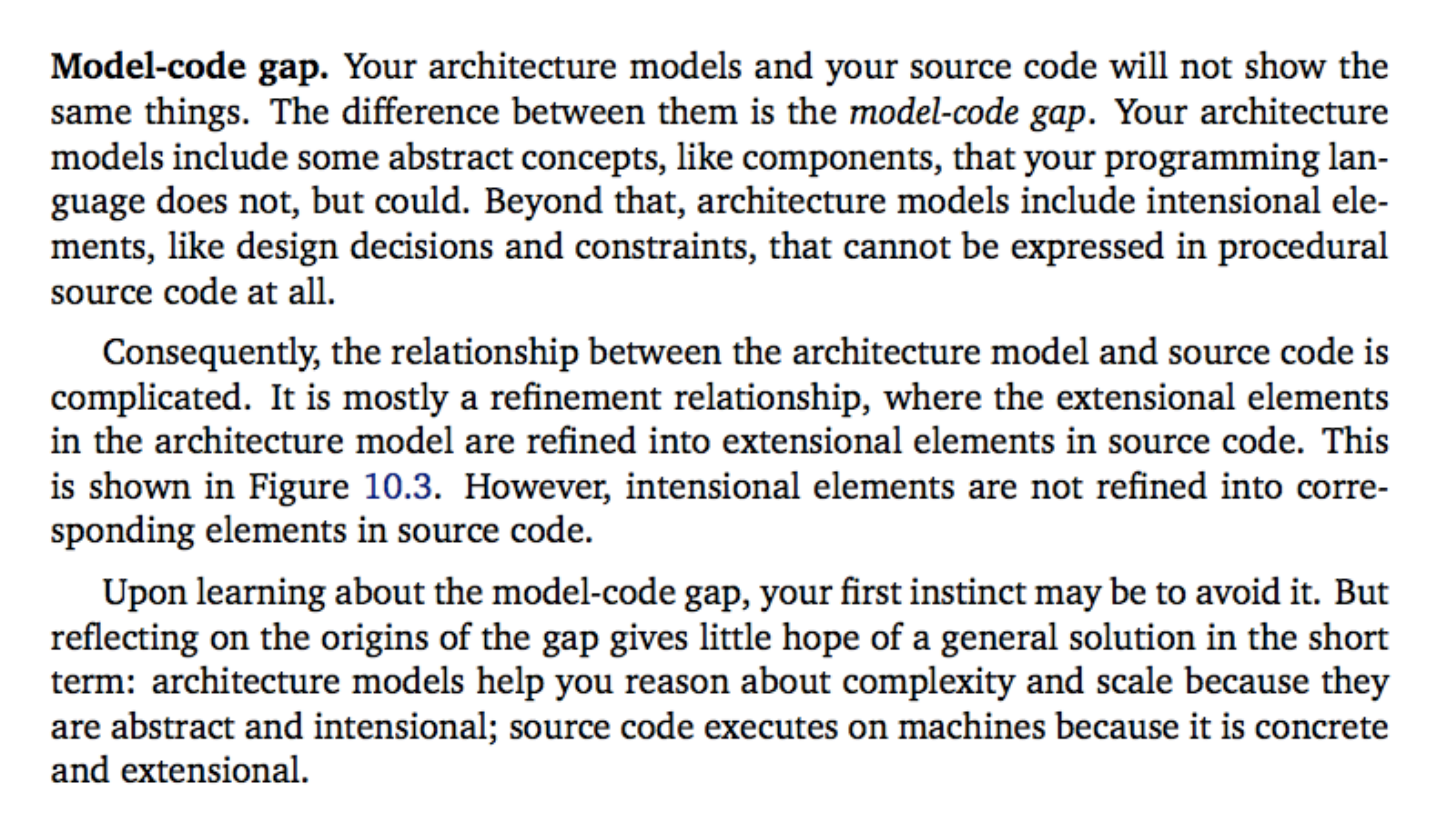


1 Introduction

Software engineers often think about an existing software system in terms of high-level models. Box and arrow sketches of a system, for instance, are often found on engineers' whiteboards. Although these models are commonly used, reasoning about the system in terms of such models can be dangerous because the models are almost always inaccurate with respect to the system's source.

Current reverse engineering systems derive high-level models from the source code. These derived models are useful because they are, by their very nature, accurate representations of the source. Although accurate, the models created by these reverse engineering systems may differ from the models sketched by engineers; an example of this is reported by Wong et al. [WTMS95].





Model-code gap. Your architecture models and your source code will not show the same things. The difference between them is the *model-code gap*. Your architecture models include some abstract concepts, like components, that your programming language does not, but could. Beyond that, architecture models include intensional elements, like design decisions and constraints, that cannot be expressed in procedural source code at all.

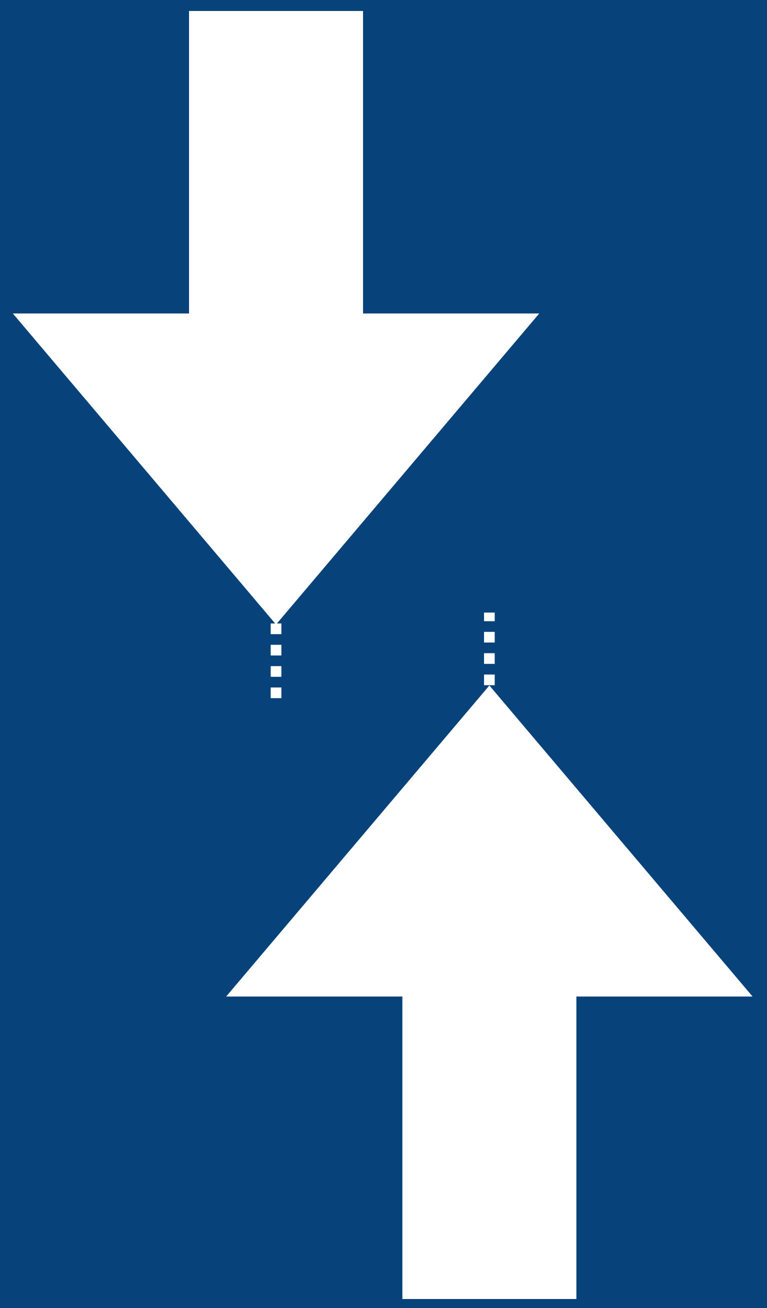
Consequently, the relationship between the architecture model and source code is complicated. It is mostly a refinement relationship, where the extensional elements in the architecture model are refined into extensional elements in source code. This is shown in Figure 10.3. However, intensional elements are not refined into corresponding elements in source code.

Upon learning about the model-code gap, your first instinct may be to avoid it. But reflecting on the origins of the gap gives little hope of a general solution in the short term: architecture models help you reason about complexity and scale because they are abstract and intensional; source code executes on machines because it is concrete and extensional.

“model-code gap”

Top-down view

(components, layers, subsystems, bounded contexts, etc)



Bottom-up view

(classes, interfaces, enums, functions, etc)

Would you code it that way?

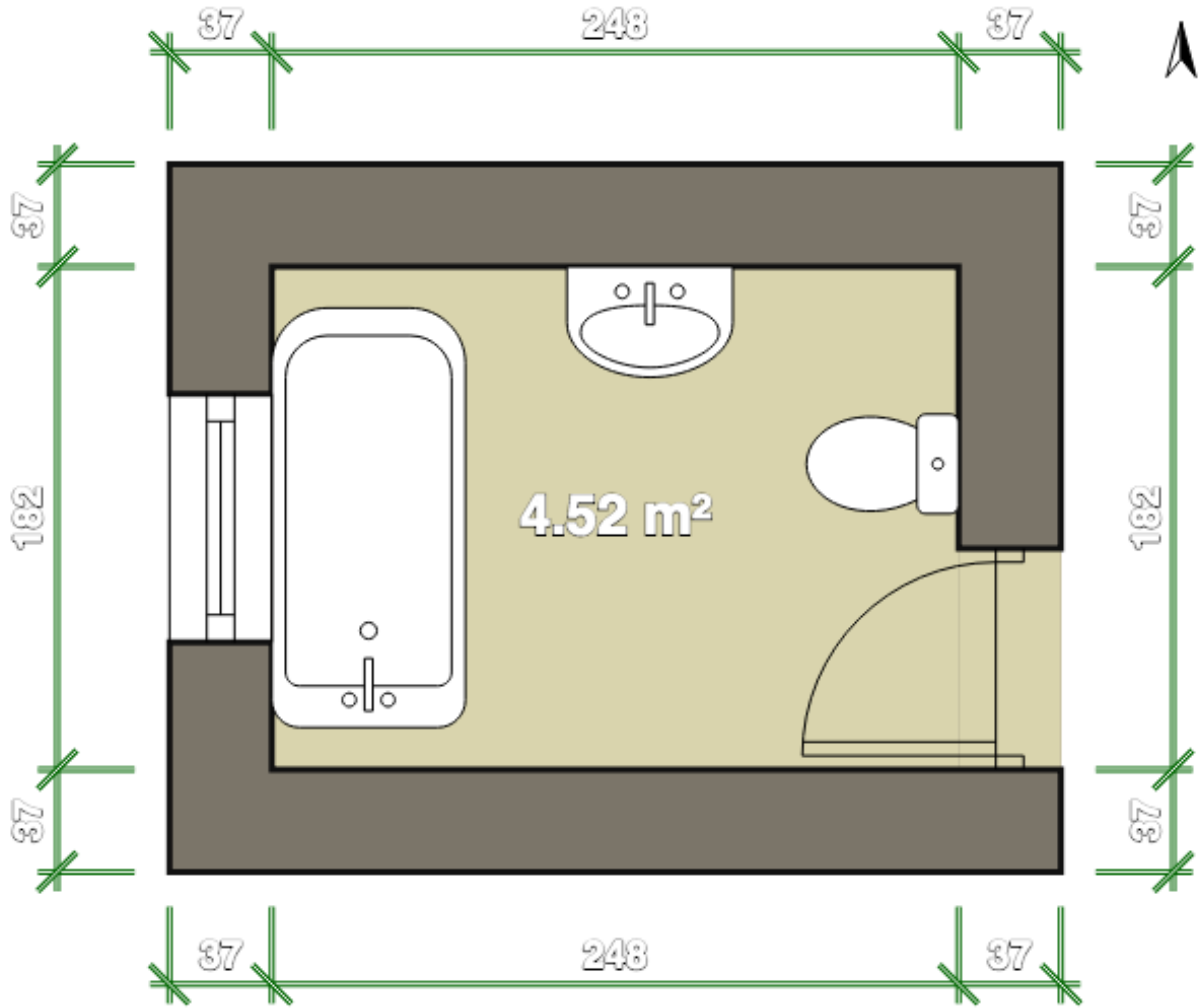
(ensure that your diagrams reflect your implementation intent)

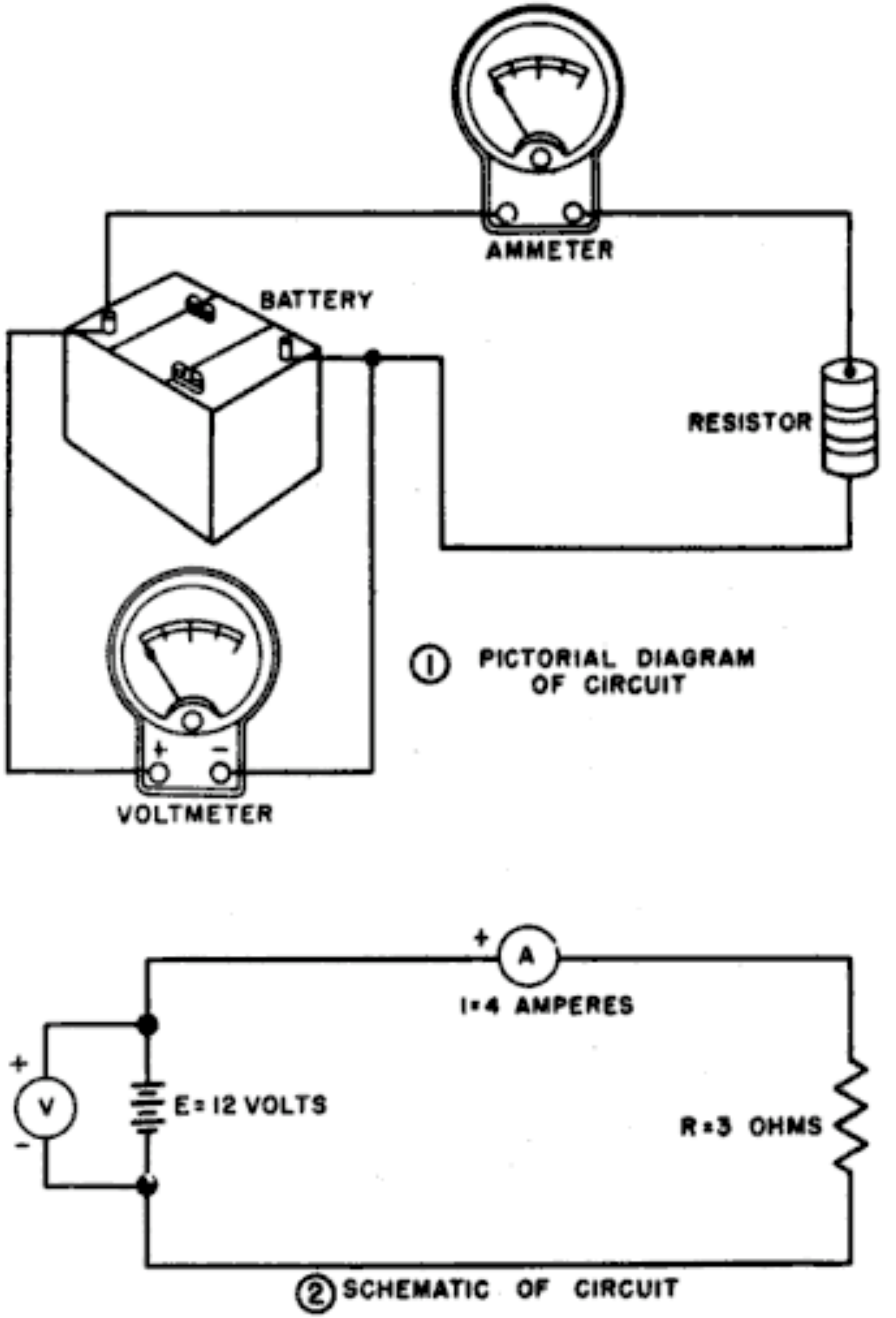
Is that how it really works?

(ensure that your diagrams reflect
your actual codebase)

We lack a **common vocabulary**
to describe software architecture

N Circular Rd
don Way
A5
Finchley Rd
Holloway
Green Lanes
STOKE NEWINGTON
A107
A104
Orient Way
A112
A12
A176
Wanstead Flats
N Circular Rd
Ilford
SEVEN K
t Park
A4088
A407
A5200
A503
A10
A1207
Forest Ln
Romford Rd
Ilford Ln
High St N
Katherine Rd
Green St
STRATFORD
A124
Up
CHALK FARM
CAMDEN TOWN
York Way
A5203
A1
A104
A5
A10
Barking
A123
ZSL London Zoo
Albany St
High St
KENSAL GREEN
A404
A5
A501
A501
A1208
Boleyn Ground
Alfreds W
A12
A1209
A124
High St S
A501
A11
A1205
Newham Way
CLERKENWELL
The British Museum
MARYLEBONE
A5
PADDINGTON
A11
WHITE CITY
SOHO
CITY OF LONDON
WHITECHAPEL
A13
POPLAR
A112
A1020
Bayswater Rd
A1203
e Vale
London
River Thames
SHEPHERD'S BUSH
Kensington Palace
CANARY WHARF
London City Airport
Southwark
KENSINGTON
Royal Albert Hall
Buckingham Palace
A112
River Thames
A219
WESTMINSTER
A2
Isle of Dogs
Lower Rd
King St
A315
A3220
LAMBETH
A1206
PIMLICO
WOOLWICH
River Thames
VAUXHALL
A200
Lonsdale Rd
Castelnau
Fulham Palace Rd
A3218
CHELSEA
A3212
A3
A2
A206
A102
FULHAM
A3036
A206
A3220
A3205
A207
A308
CAMBERWELL
PECKHAM
Brixton Rd
A2
A207
A3
A203
A3205
A2217
A207
Rd
S Circular Rd
BATTERSEA
CLAPHAM
BRIXTON
A3
A2
A2213
Priory Ln
A219
A24
A20
A208
A2
Brixton Hill
HERNE HILL
EAST DULWICH
Dulwich Village
Bexley Rd
Putney Heath
A3
A214
A204
LEE
S Circular Rd
A3

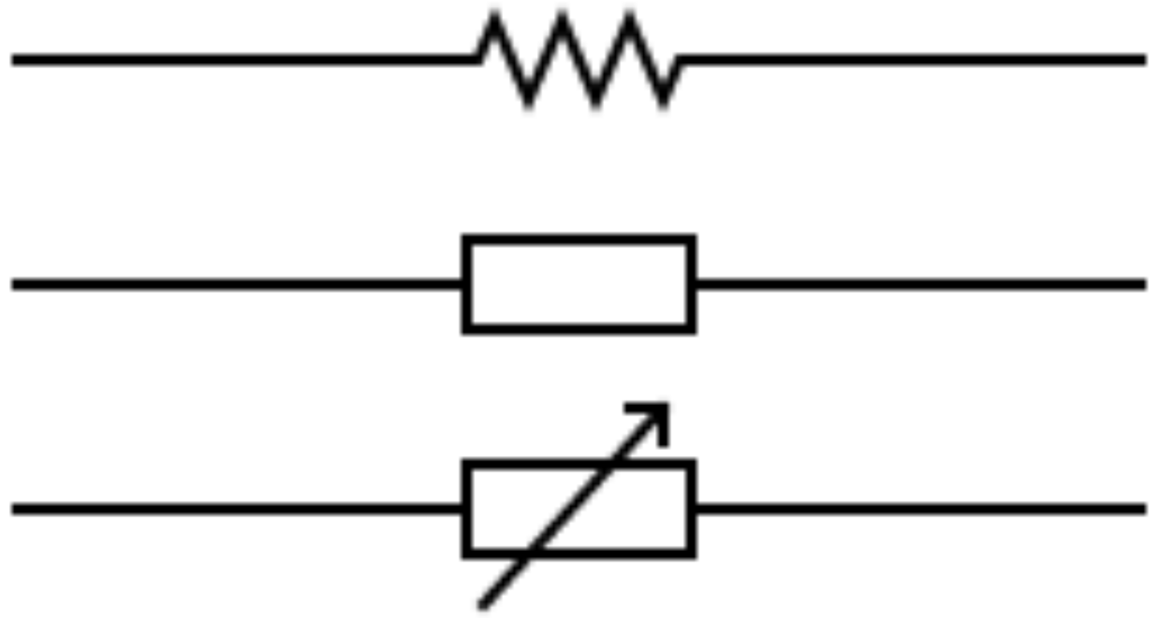

37
248
37
37
182
37
37
182
37
4.52 m²
37
248
37



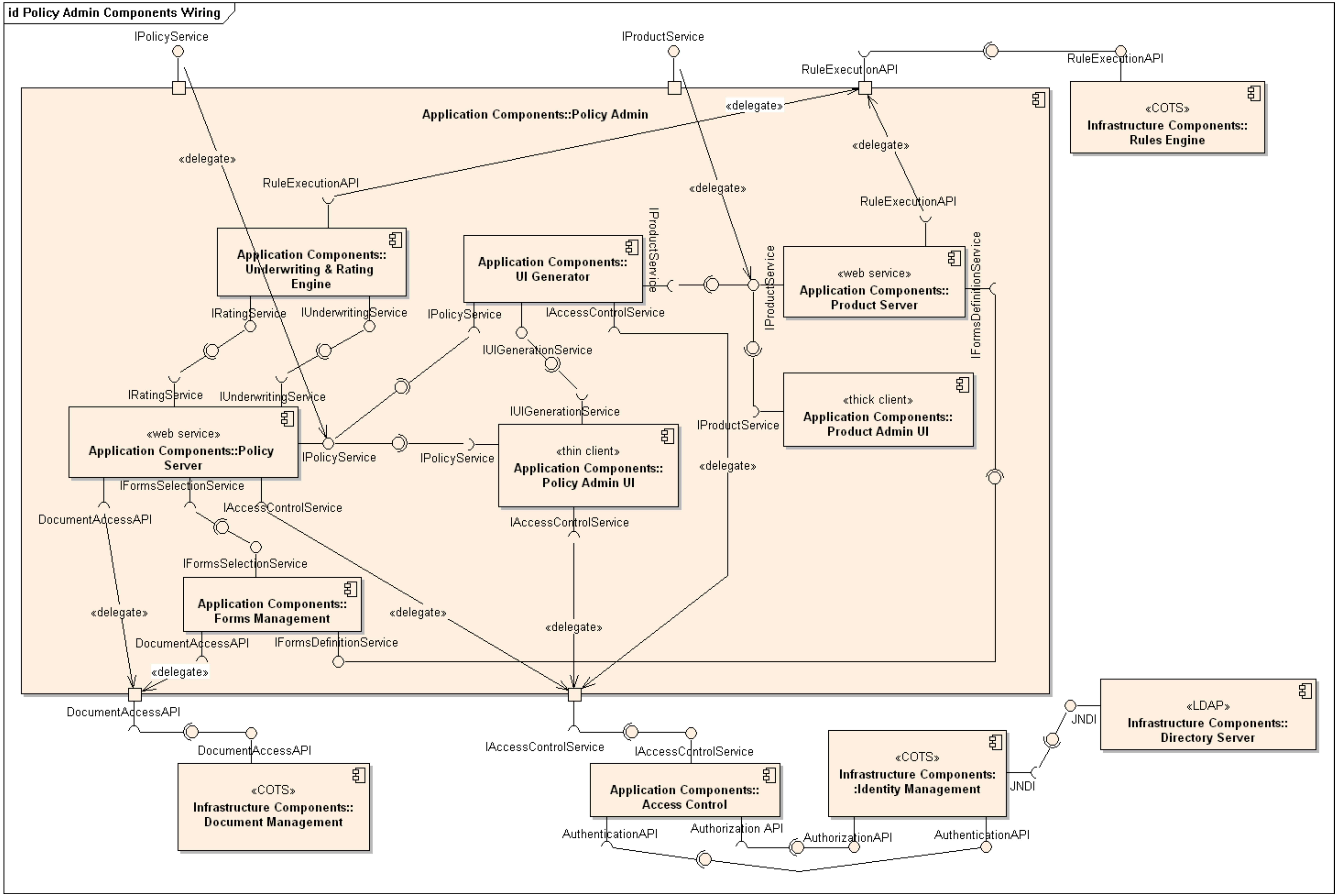
① PICTORIAL DIAGRAM OF CIRCUIT

② SCHEMATIC OF CIRCUIT

Figure 48. Diagram of a basic circuit.



https://en.wikipedia.org/wiki/Circuit_diagram

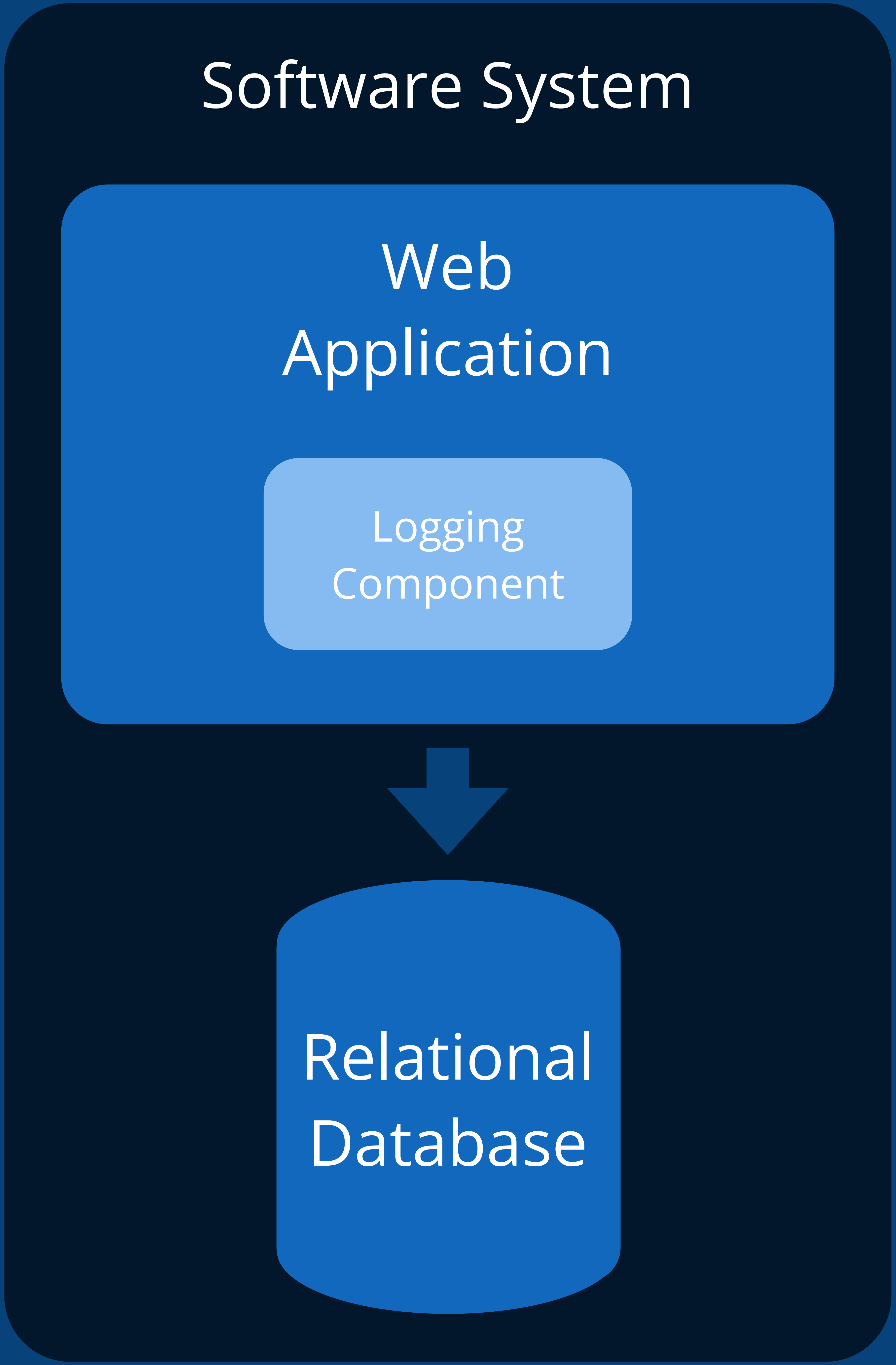


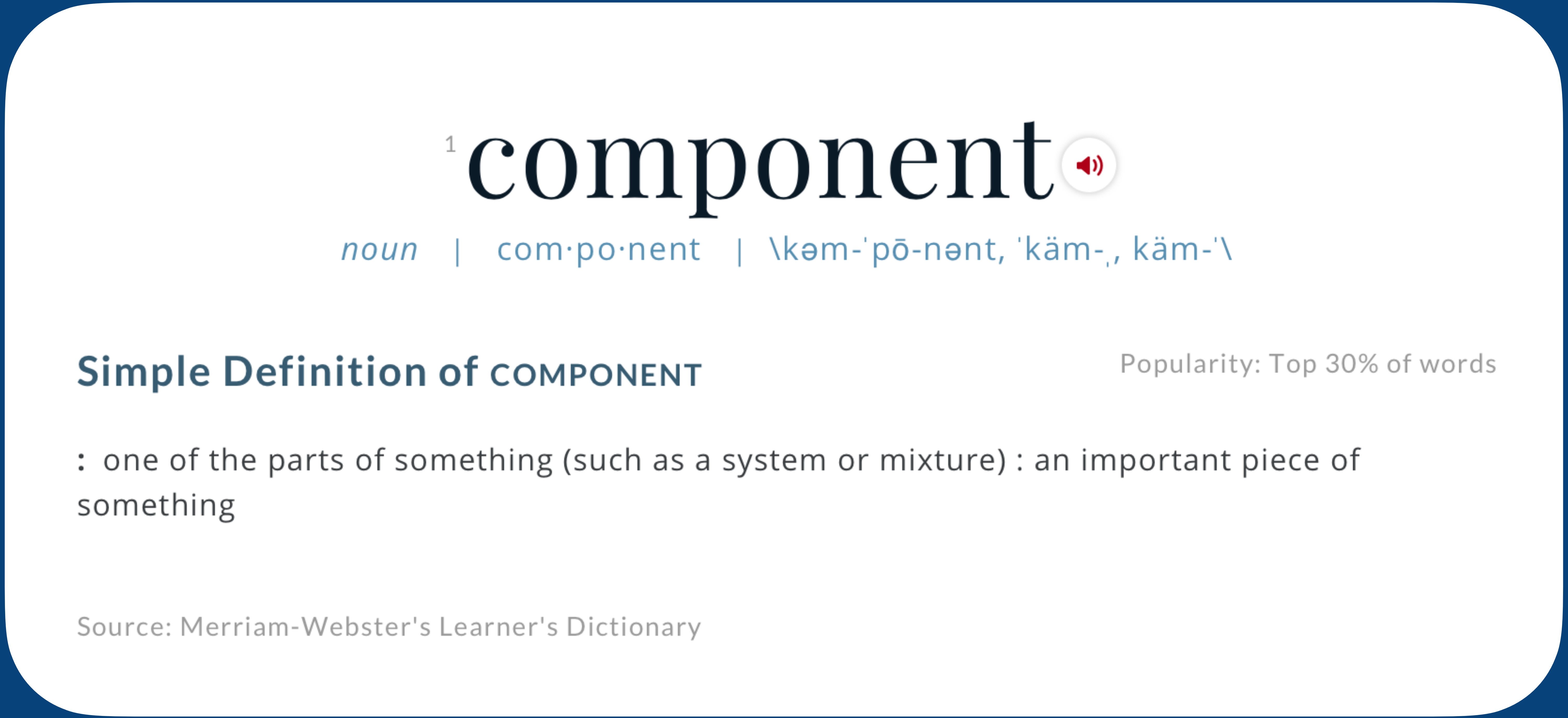
https://en.wikipedia.org/wiki/Component_diagram

Component

a modular unit with well-defined Interfaces that is replaceable within its environment

https://www.omg.org/spec/UML/2.5.1/PDF





[1] component

noun | com·po·nent | \kəm-ˈpō-nənt, ˈkäm-ˌ, käm-ˈ\

Simple Definition of COMPONENT

Popularity: Top 30% of words

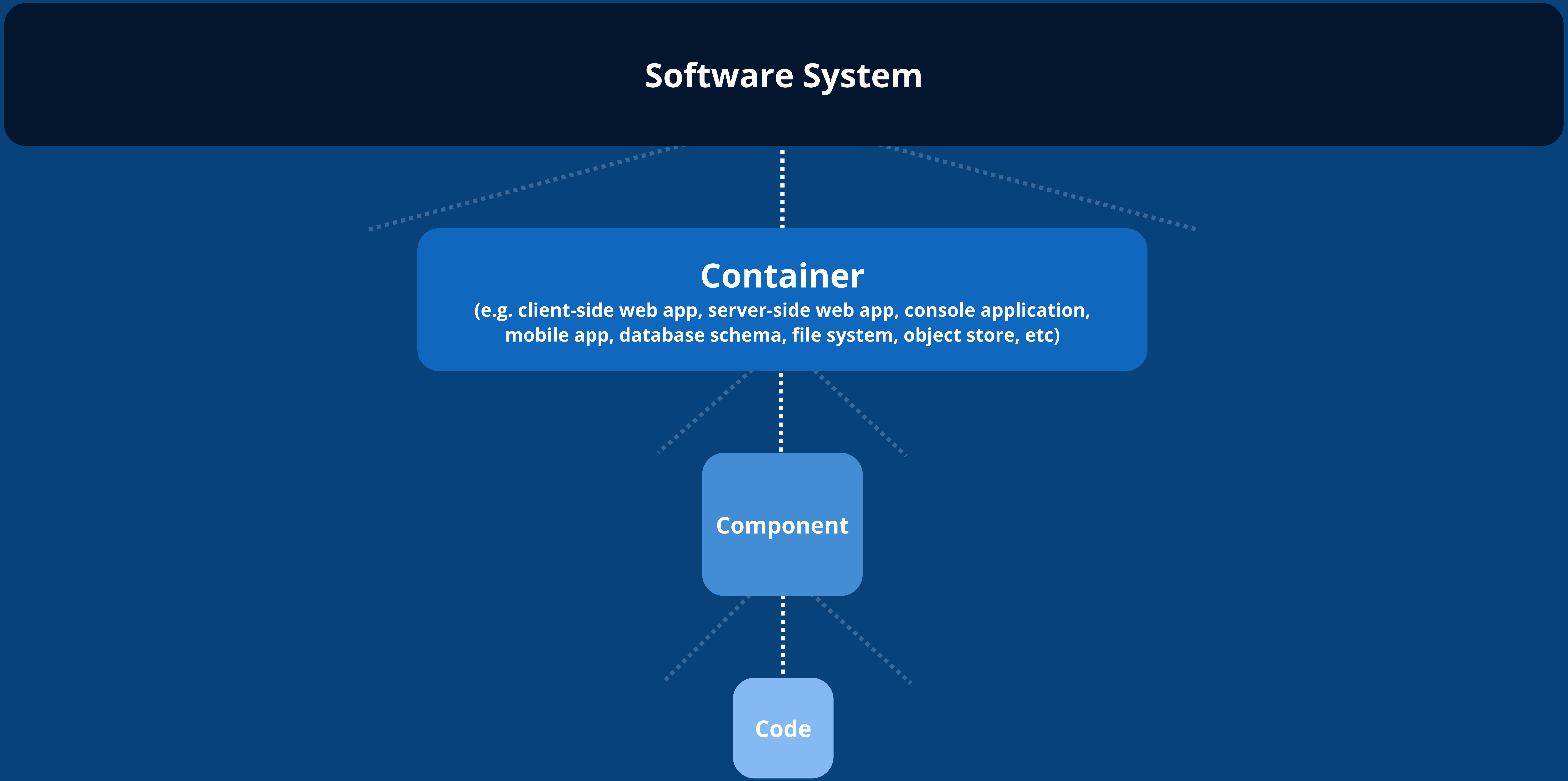
: one of the parts of something (such as a system or mixture) : an important piece of something

Source: Merriam-Webster's Learner's Dictionary

Ubiquitous language

A **common set of abstractions**
is more important
than a common notation

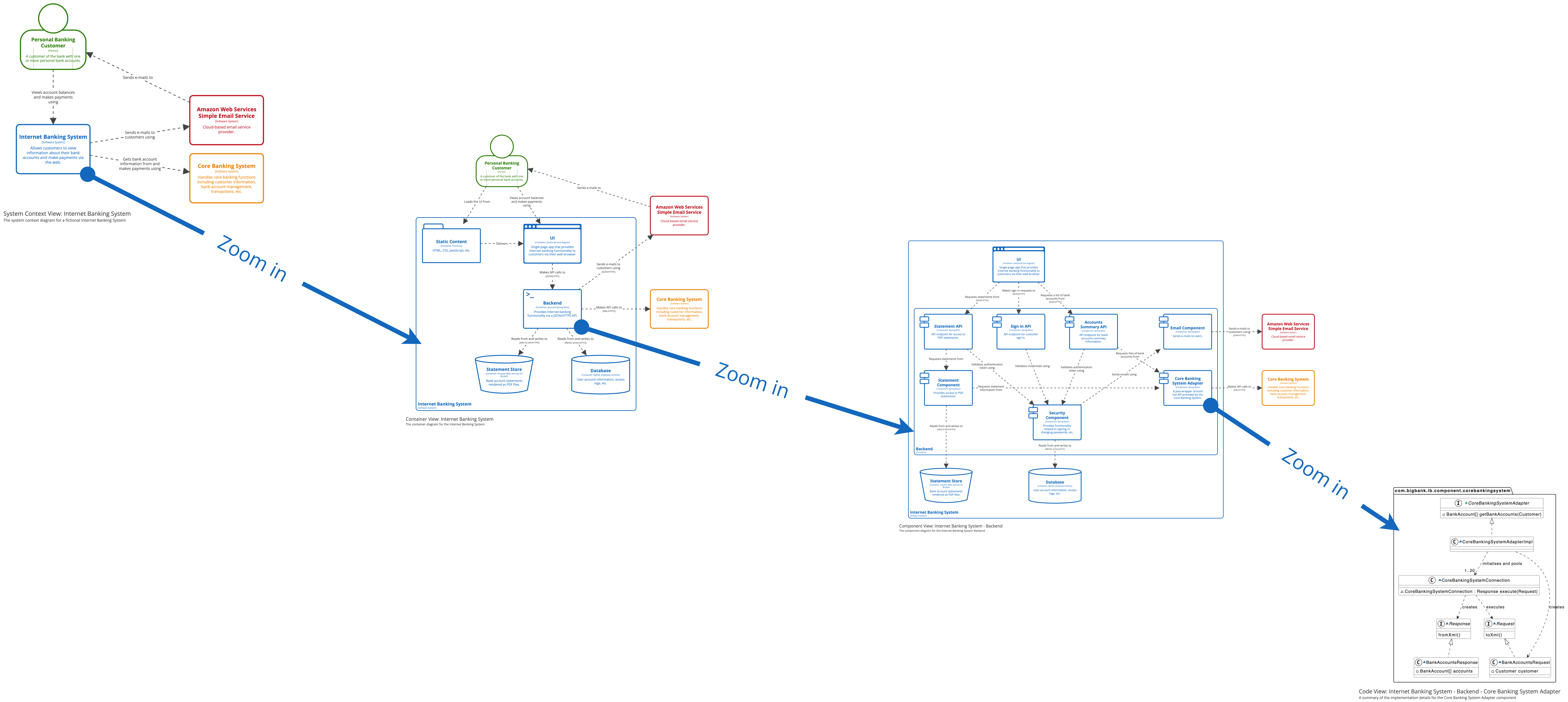
Abstractions


Software System
Container
(e.g. client-side web app, server-side web app, console application, mobile app, database schema, file system, object store, etc)
Component
Code

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


Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.
Views account balances and makes payments using
Sends e-mails to
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
Gets bank account information from and makes payments using
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
Zoom in
Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.
Loads the UI from
Views account balances and makes payments using
Sends e-mails to
Static Content
[Container: Directory]
HTML, CSS, JavaScript, etc.
Delivers
UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Makes API calls to
[JSON/HTTPS]
Sends e-mails to customers using
[JSON/HTTPS]
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.
Internet Banking System
[Software System]
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
Zoom in
UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Requests statements from
[JSON/HTTPS]
Makes sign in requests to
[JSON/HTTPS]
Requests a list of bank accounts from
[JSON/HTTPS]
Statement API
[Component: Spring MVC]
API endpoint for access to PDF statements.
Sign In API
[Component: Spring MVC]
API endpoint for customer sign in.
Accounts Summary API
[Component: Spring MVC]
API endpoint for bank accounts summary information.
Email Component
[Component: Spring Bean]
Sends e-mails to users.
Sends e-mails to customers using
[JSON/HTTPS]
Requests statements from
Validates authentication token using
Validates credentials using
Validates authentication token using
Requests lists of bank accounts from
Sends emails using
Statement Component
[Component: Spring Bean]
Provides access to PDF statements.
Requests statement information from
Core Banking System Adapter
[Component: Spring Bean]
A Java wrapper around the API provided by the Core Banking System.
Makes API calls to
[XML/HTTPS]
Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.
Reads from and writes to
[AWS S3 API/HTTPS]
Reads from and writes to
[MySQL protocol/TLS]
Backend
[Container]
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.
Internet Banking System
[Software System]
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
Zoom in
com.bigbank.ib.component.corebankingsystem
I CoreBankingSystemAdapter
○ BankAccount[] getBankAccounts(Customer)
C CoreBankingSystemAdapterImpl
initialises and pools
1..20
C CoreBankingSystemConnection
△ CoreBankingSystemConnection : Response execute(Request)
creates
executes
creates
I Response
fromXml()
I Request
toXml()
C BankAccountsResponse
□ BankAccount[] accounts
C BankAccountsRequest
□ Customer customer

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

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

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

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

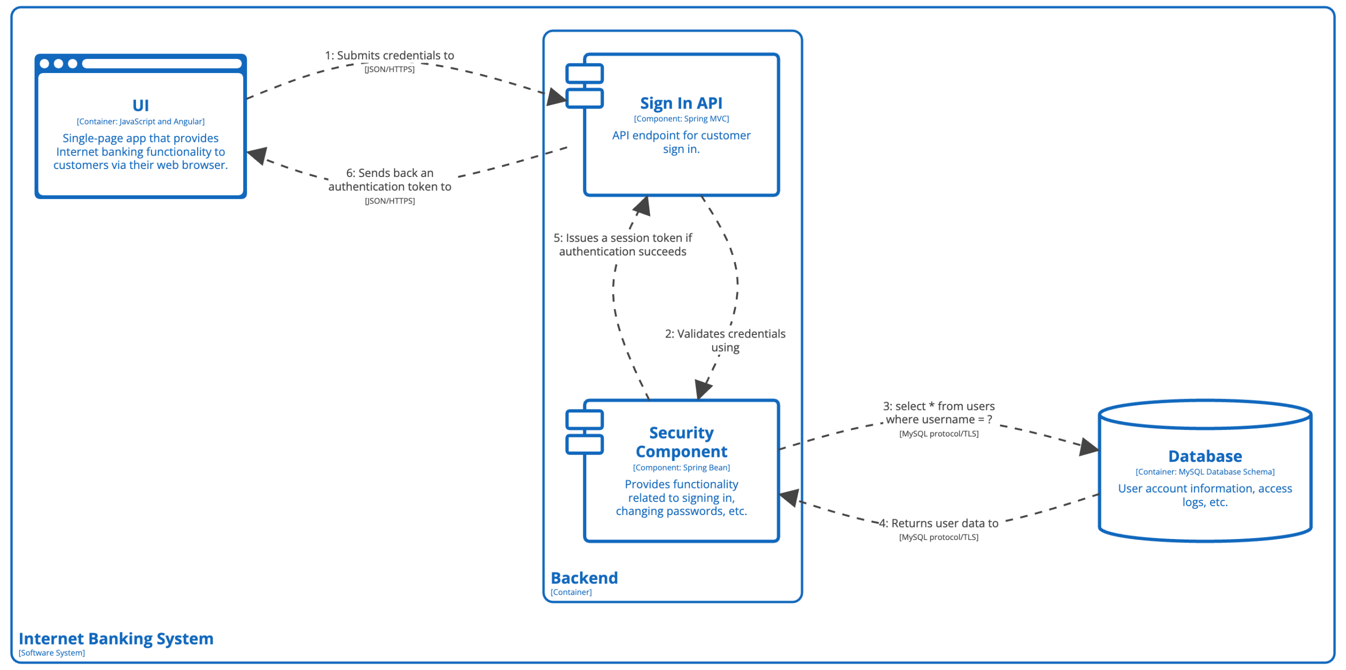
Containers

Components

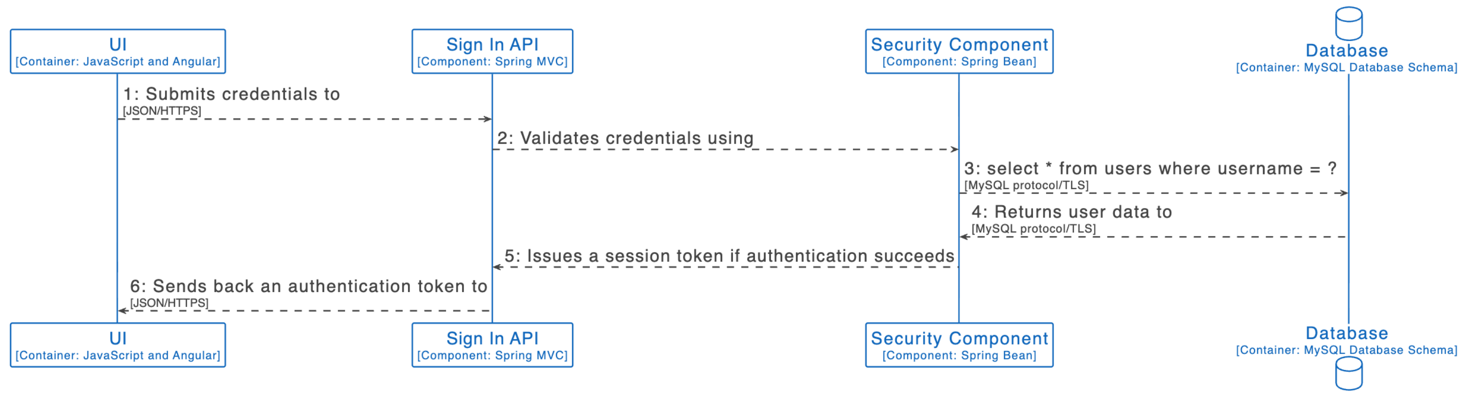
Code

Supporting diagrams

Dynamic

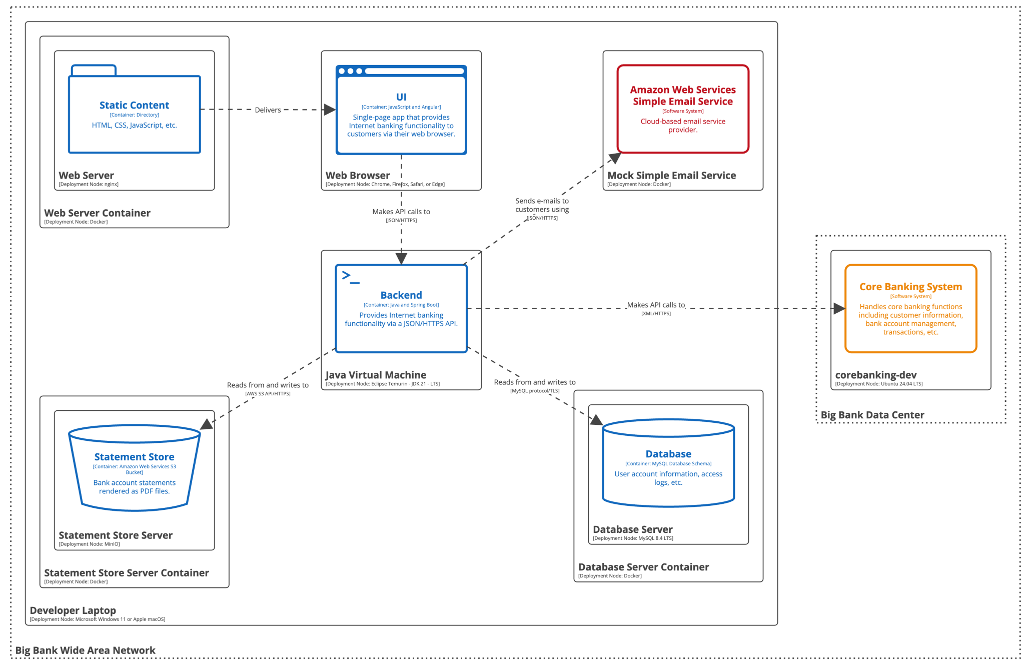

UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
1: Submits credentials to
[JSON/HTTPS]
Sign In API
[Component: Spring MVC]
API endpoint for customer sign in.
6: Sends back an authentication token to
[JSON/HTTPS]
5: Issues a session token if authentication succeeds
2: Validates credentials using
Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.
3: select * from users where username = ?
[MySQL protocol/TLS]
4: Returns user data to
[MySQL protocol/TLS]
Database
[Container: MySQL Database Schema]
User account information, access logs, etc.
Backend
[Container]
Internet Banking System
[Software System]

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

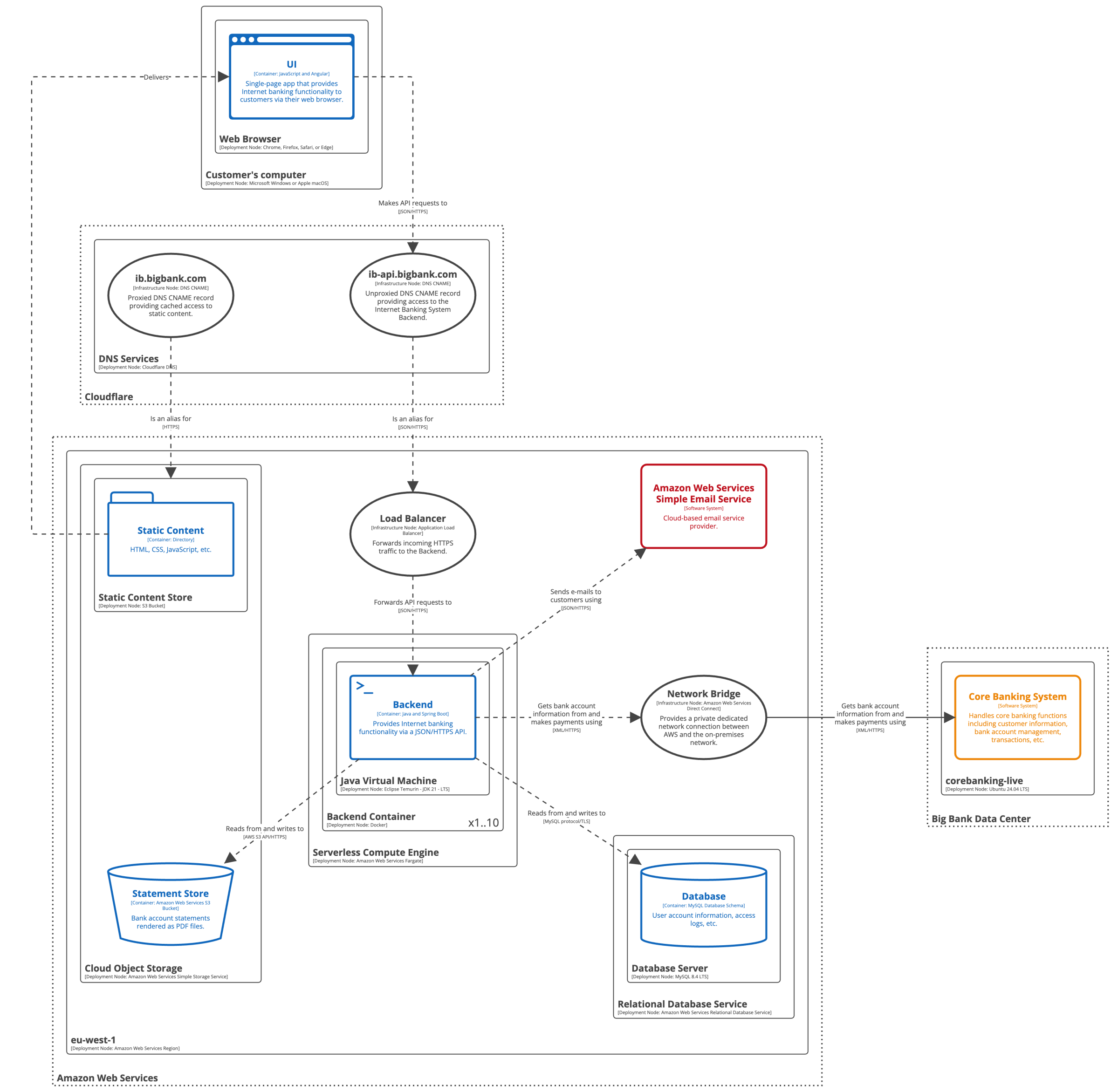

UI
[Container: JavaScript and Angular]
Sign In API
[Component: Spring MVC]
Security Component
[Component: Spring Bean]
Database
[Container: MySQL Database Schema]
1: Submits credentials to
[JSON/HTTPS]
2: Validates credentials using
3: select * from users where username = ?
[MySQL protocol/TLS]
4: Returns user data to
[MySQL protocol/TLS]
5: Issues a session token if authentication succeeds
6: Sends back an authentication token to
[JSON/HTTPS]
UI
[Container: JavaScript and Angular]
Sign In API
[Component: Spring MVC]
Security Component
[Component: Spring Bean]
Database
[Container: MySQL Database Schema]

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

Deployment

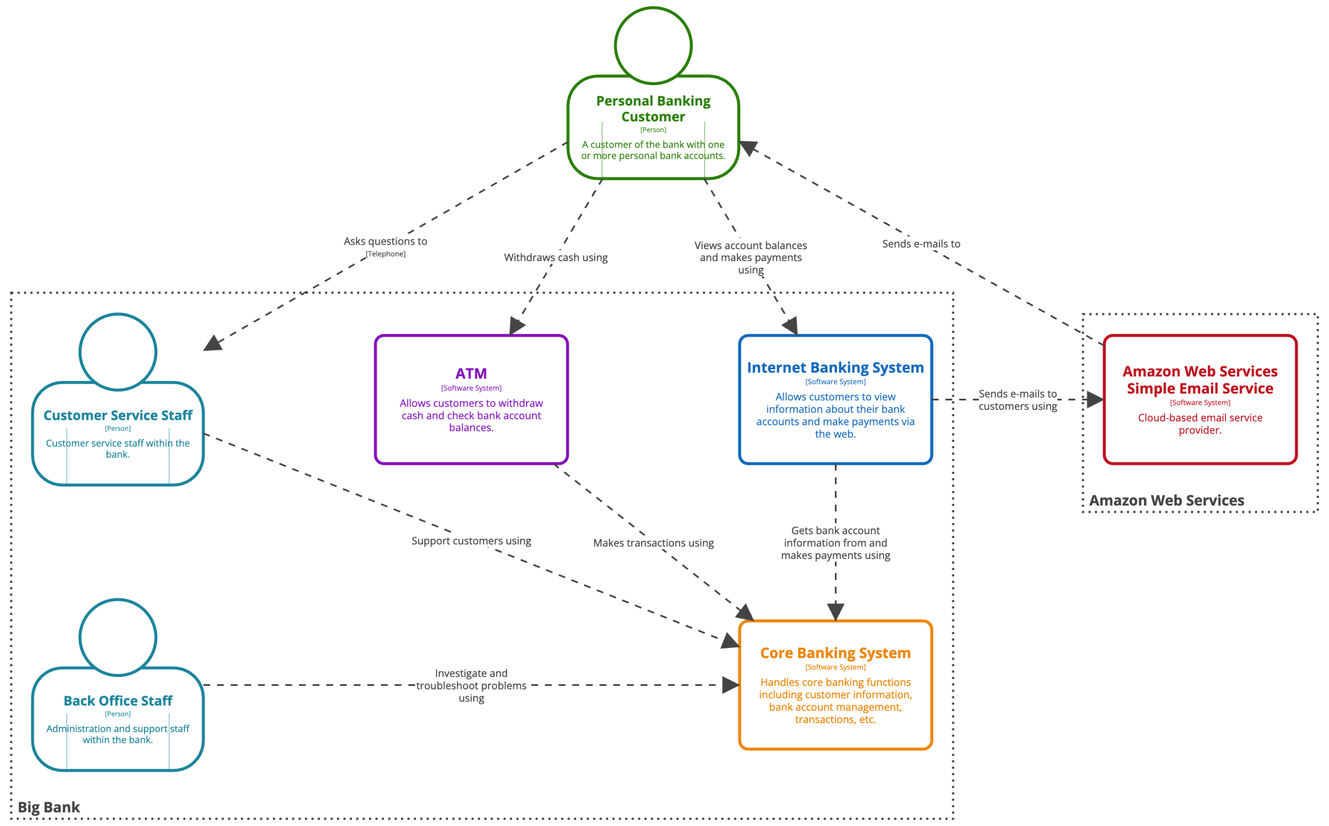

Static Content
[Container: Directory]
HTML, CSS, JavaScript, etc.
Web Server
[Deployment Node: nginx]
Web Server Container
[Deployment Node: Docker]
Delivers
UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Web Browser
[Deployment Node: Chrome, Firefox, Safari, or Edge]
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Mock Simple Email Service
[Deployment Node: Docker]
Makes API calls to
[JSON/HTTPS]
Sends e-mails to customers using
[JSON/HTTPS]
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]
Makes API calls to
[XML/HTTPS]
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
corebanking-dev
[Deployment Node: Ubuntu 24.04 LTS]
Big Bank Data Center
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.
Statement Store Server
[Deployment Node: MinIO]
Statement Store Server Container
[Deployment Node: Docker]
Database
[Container: MySQL Database Schema]
User account information, access logs, etc.
Database Server
[Deployment Node: MySQL 8.4 LTS]
Database Server Container
[Deployment Node: Docker]
Developer Laptop
[Deployment Node: Microsoft Windows 11 or Apple macOS]
Big Bank Wide Area Network

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

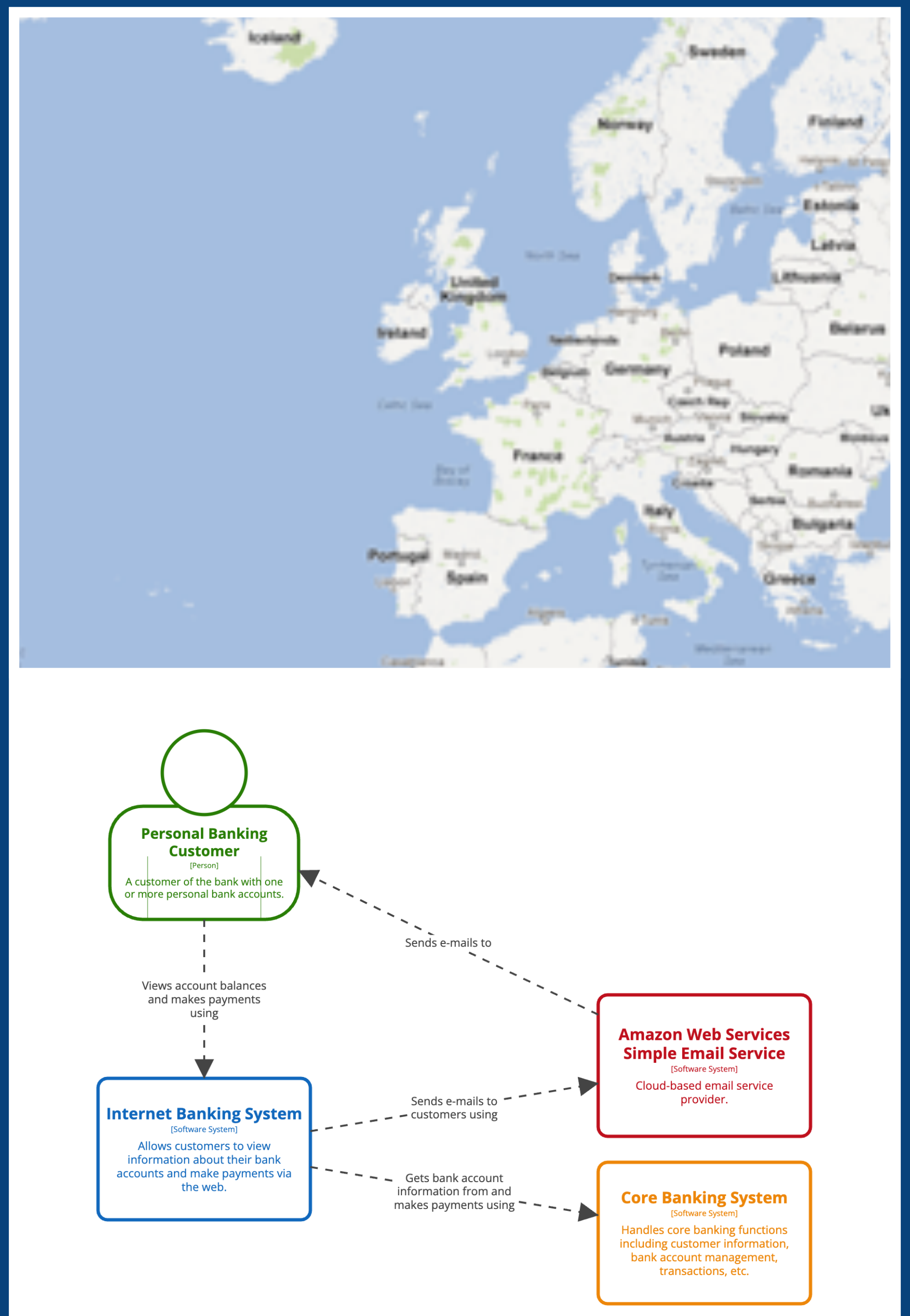

Delivers
UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Web Browser
[Deployment Node: Chrome, Firefox, Safari, or Edge]
Customer's computer
[Deployment Node: Microsoft Windows or Apple macOS]
Makes API requests to
[JSON/HTTPS]
ib.bigbank.com
[Infrastructure Node: DNS CNAME]
Proxied DNS CNAME record providing cached access to static content.
ib-api.bigbank.com
[Infrastructure Node: DNS CNAME]
Unproxied DNS CNAME record providing access to the Internet Banking System Backend.
DNS Services
[Deployment Node: Cloudflare DNS]
Cloudflare
Is an alias for
[HTTPS]
Is an alias for
[JSON/HTTPS]
Static Content
[Container: Directory]
HTML, CSS, JavaScript, etc.
Static Content Store
[Deployment Node: S3 Bucket]
Load Balancer
[Infrastructure Node: Application Load Balancer]
Forwards incoming HTTPS traffic to the Backend.
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Forwards API requests to
[JSON/HTTPS]
Sends e-mails to customers using
[JSON/HTTPS]
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.
Gets bank account information from and makes payments using
[XML/HTTPS]
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
corebanking-live
[Deployment Node: Ubuntu 24.04 LTS]
Big Bank Data Center
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]
eu-west-1
[Deployment Node: Amazon Web Services Region]
Amazon Web Services

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

System Landscape


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

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



Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.

Views account balances and makes payments using

Sends e-mails to

Internet Banking System
[Software System]
Allows customers to view information about their bank accounts and make payments via the web.

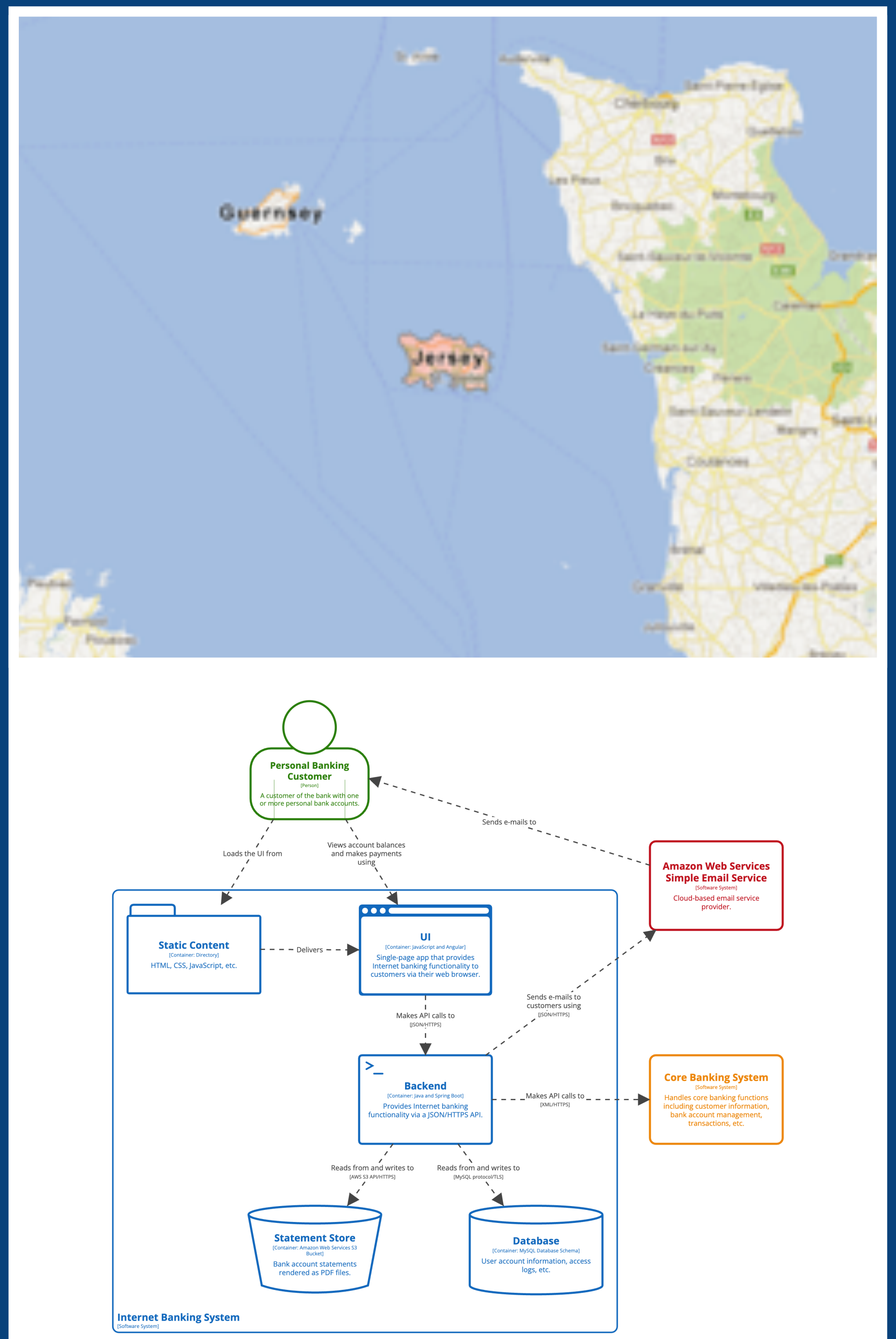
Sends e-mails to customers using

Gets bank account information from and makes payments using

Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.

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

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



Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.

Loads the UI from

Views account balances and makes payments using

Sends e-mails to

Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.

Static Content
[Container: Directory]
HTML, CSS, JavaScript, etc.

Delivers

UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.

Makes API calls to
[JSON/HTTPS]

Sends e-mails to customers using
[JSON/HTTPS]

Backend
[Container: Java and Spring Boot]
Provides Internet banking functionality via a JSON/HTTPS API.

Makes API calls to
[XML/HTTPS]

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

Reads from and writes to
[AWS S3 API/HTTPS]

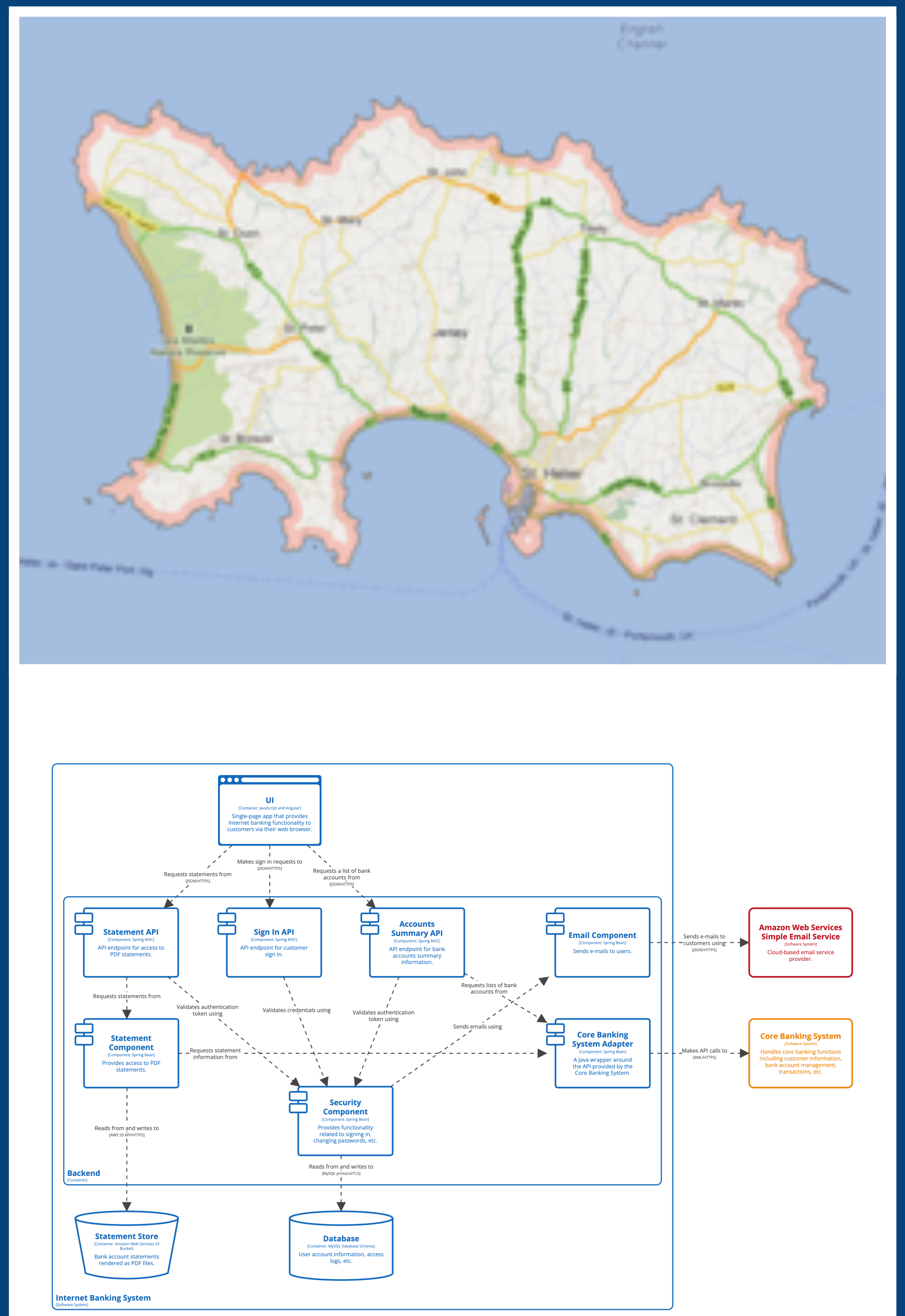
Reads from and writes to
[MySQL protocol/TLS]

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.

Internet Banking System
[Software System]

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



UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.

Requests statements from
[JSON/HTTPS]

Makes sign in requests to
[JSON/HTTPS]

Requests a list of bank accounts from
[JSON/HTTPS]

Statement API
[Component: Spring MVC]
API endpoint for access to PDF statements.

Sign In API
[Component: Spring MVC]
API endpoint for customer sign in.

Accounts Summary API
[Component: Spring MVC]
API endpoint for bank accounts summary information.

Email Component
[Component: Spring Bean]
Sends e-mails to users.

Sends e-mails to customers using
[JSON/HTTPS]

Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.

Requests statements from

Validates authentication token using

Validates credentials using

Validates authentication token using

Requests lists of bank accounts from

Sends emails using

Statement Component
[Component: Spring Bean]
Provides access to PDF statements.

Requests statement information from

Core Banking System Adapter
[Component: Spring Bean]
A Java wrapper around the API provided by the Core Banking System.

Makes API calls to
[XML/HTTPS]

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

Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.

Reads from and writes to
[AWS S3 API/HTTPS]

Reads from and writes to
[MySQL protocol/TLS]

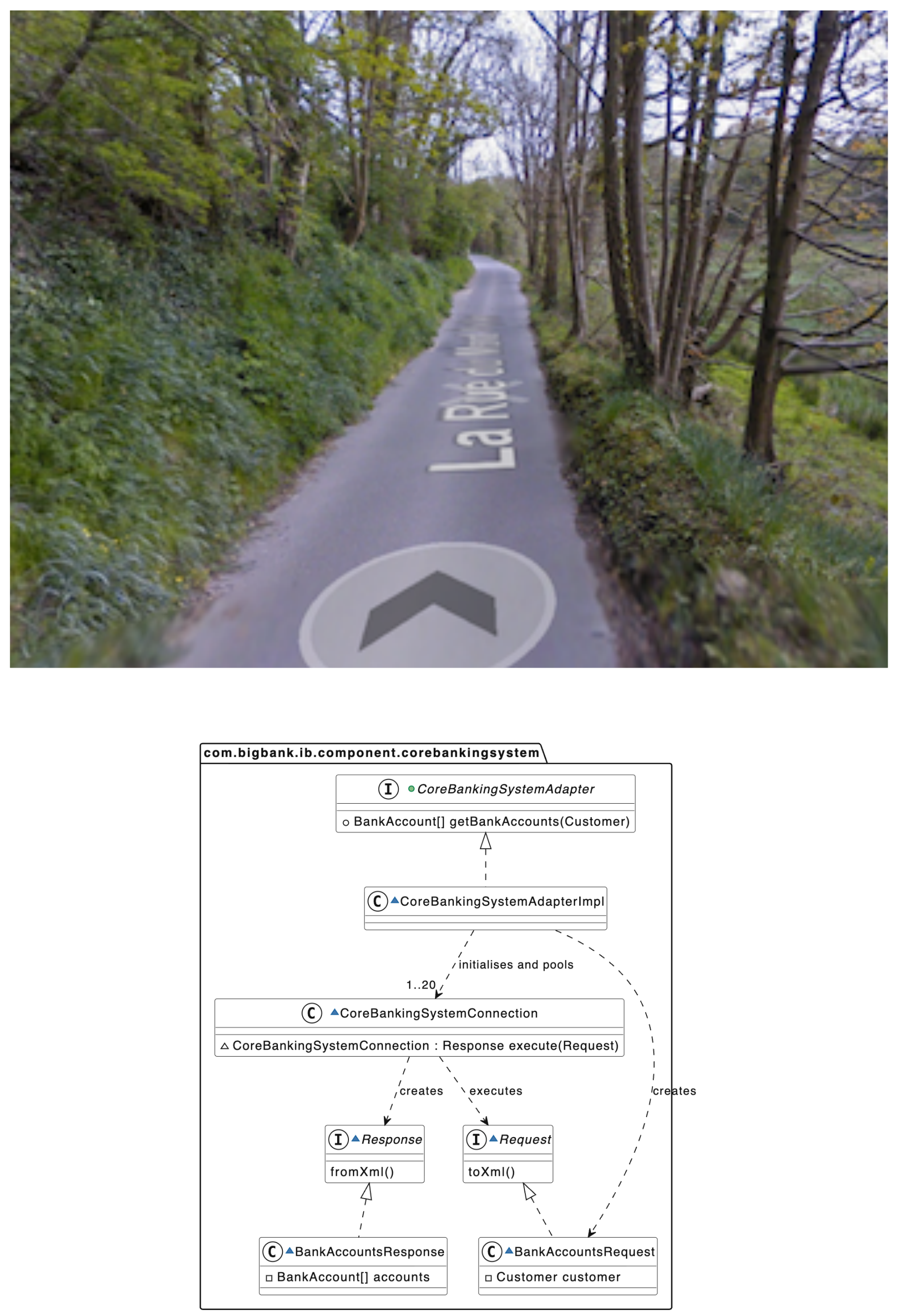
Backend
[Container]

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.

Internet Banking System
[Software System]

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



com.bigbank.ib.component.corebankingsystem

I CoreBankingSystemAdapter
○ BankAccount[] getBankAccounts(Customer)

C CoreBankingSystemAdapterImpl

initialises and pools
1..20

C CoreBankingSystemConnection
△ CoreBankingSystemConnection : Response execute(Request)

creates

executes

creates

I Response
fromXml()

I Request
toXml()

C BankAccountsResponse
□ BankAccount[] accounts

C BankAccountsRequest
□ Customer customer

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

Diagrams are maps

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

1. System Context

The system plus users and system dependencies.

2. Containers

The overall shape of the architecture and technology choices.

3. Components

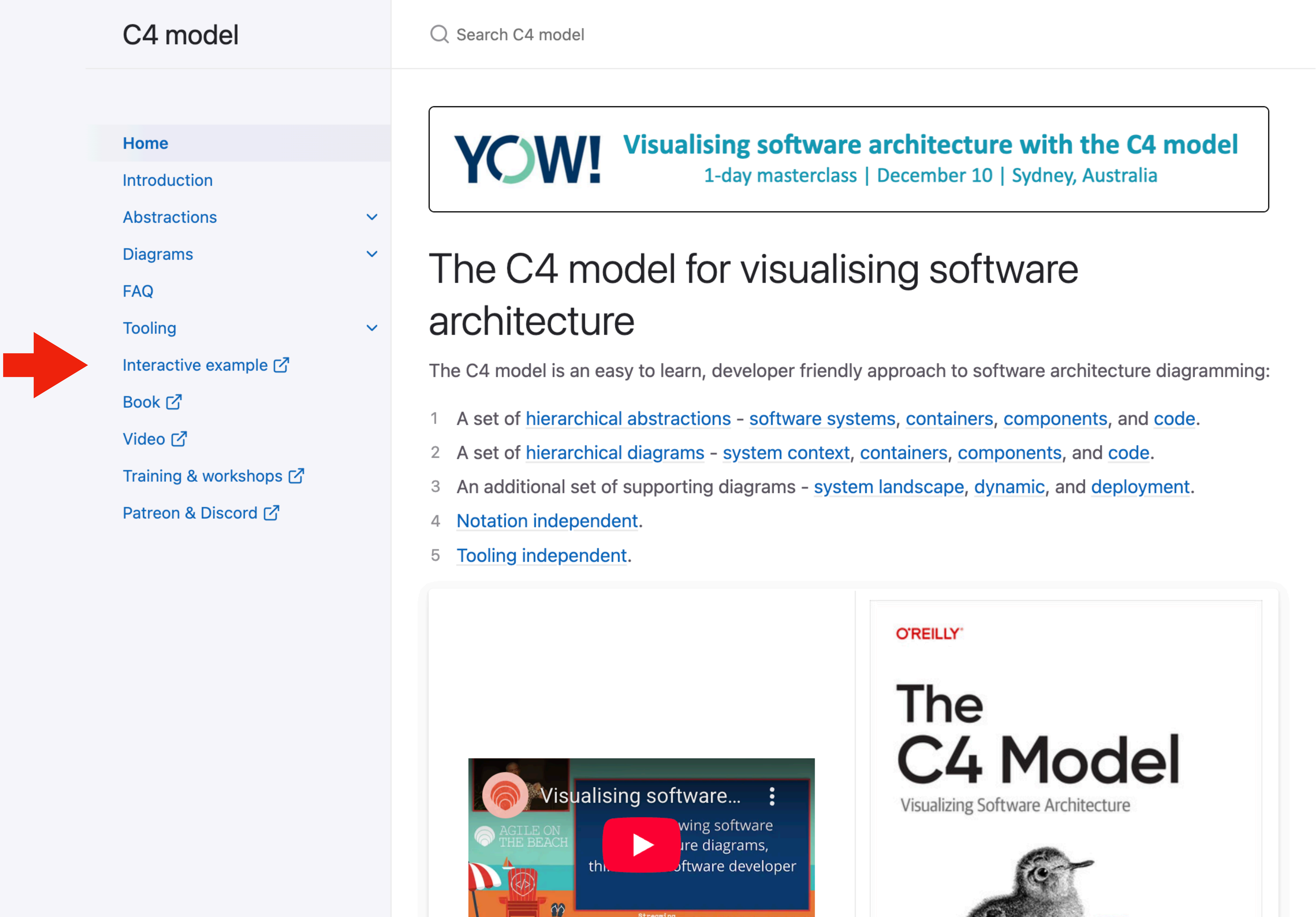
Logical components and their interactions within a container.

4. Code (e.g. classes)

Component implementation details.

Example

(available on c4model.com)



C4 model

Search C4 model

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

YOW! Visualising software architecture with the C4 model
1-day masterclass | December 10 | Sydney, Australia

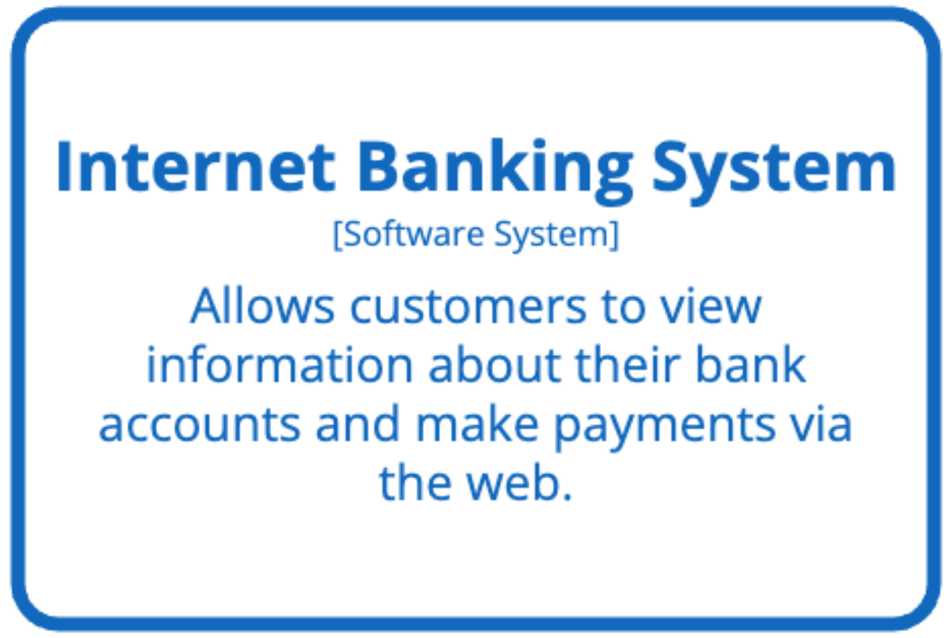
The C4 model for visualising software architecture

The C4 model is an easy to learn, developer friendly approach to software architecture diagramming:

1. A set of hierarchical abstractions - software systems, containers, components, and code.
2. A set of hierarchical diagrams - system context, containers, components, and code.
3. An additional set of supporting diagrams - system landscape, dynamic, and deployment.
4. Notation independent.
5. Tooling independent.

Level 1

System Context diagram



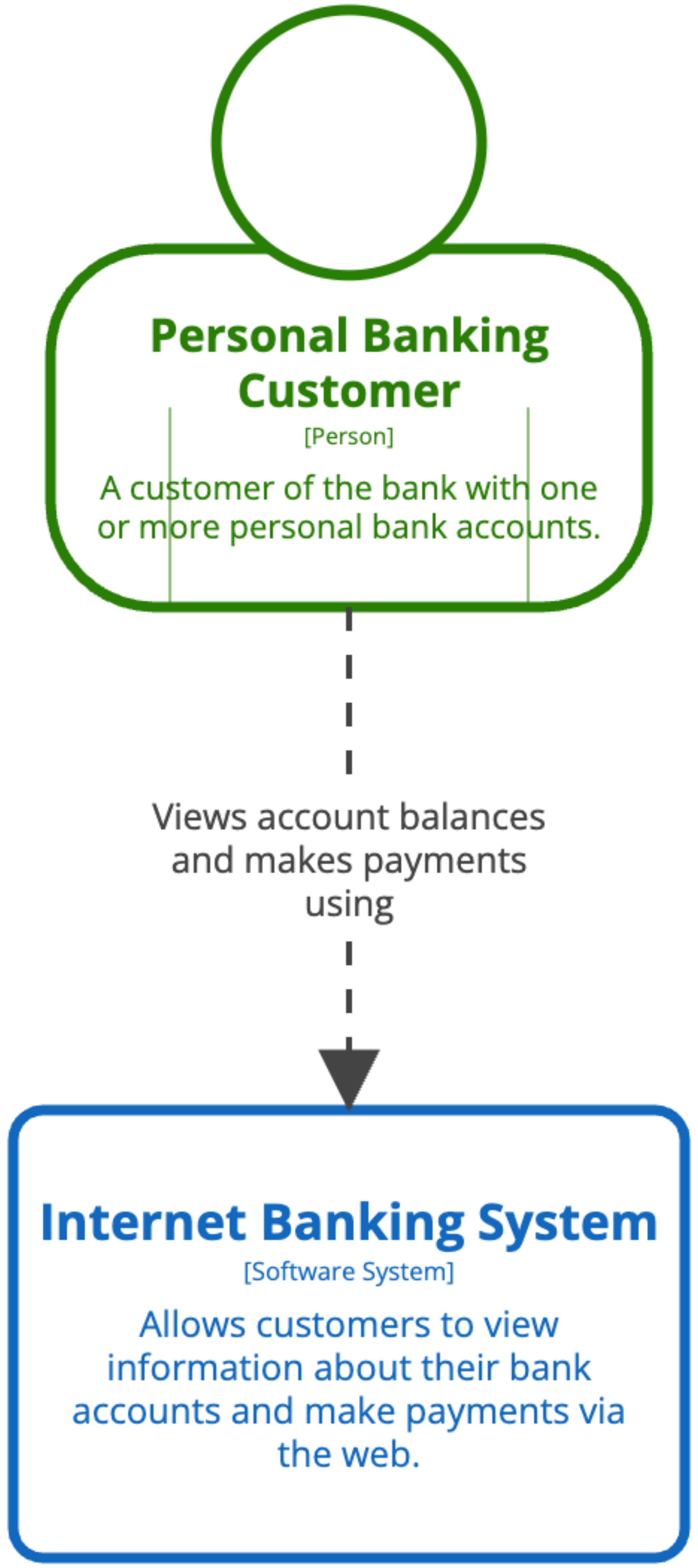
Internet Banking System

[Software System]

Allows customers to view information about their bank accounts and make payments via the web.

System Context View: Internet Banking System

The system context diagram for a fictional Internet Banking System



Personal Banking Customer

[Person]

A customer of the bank with one or more personal bank accounts.

Views account balances and makes payments using

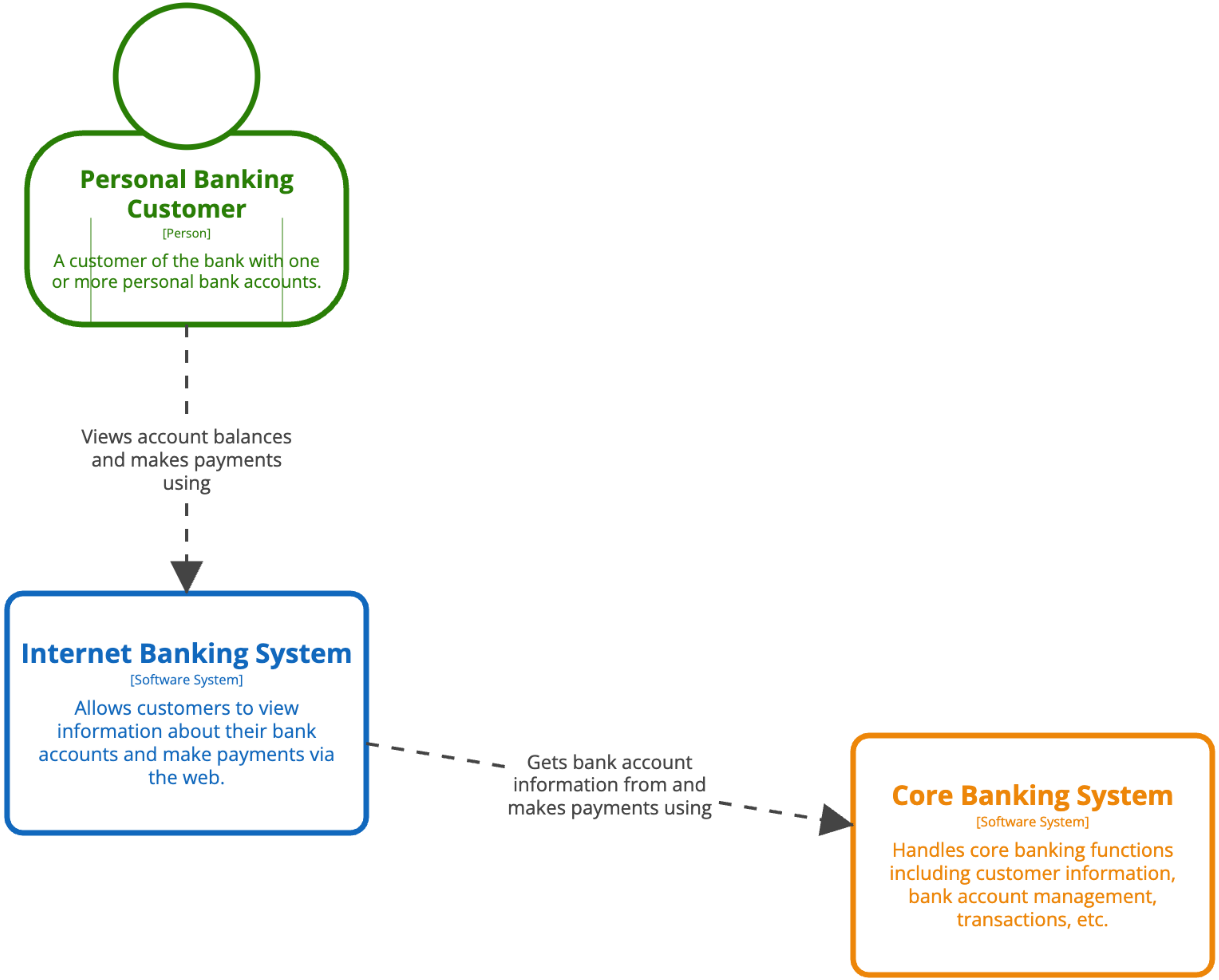
Internet Banking System

[Software System]

Allows customers to view information about their bank accounts and make payments via the web.

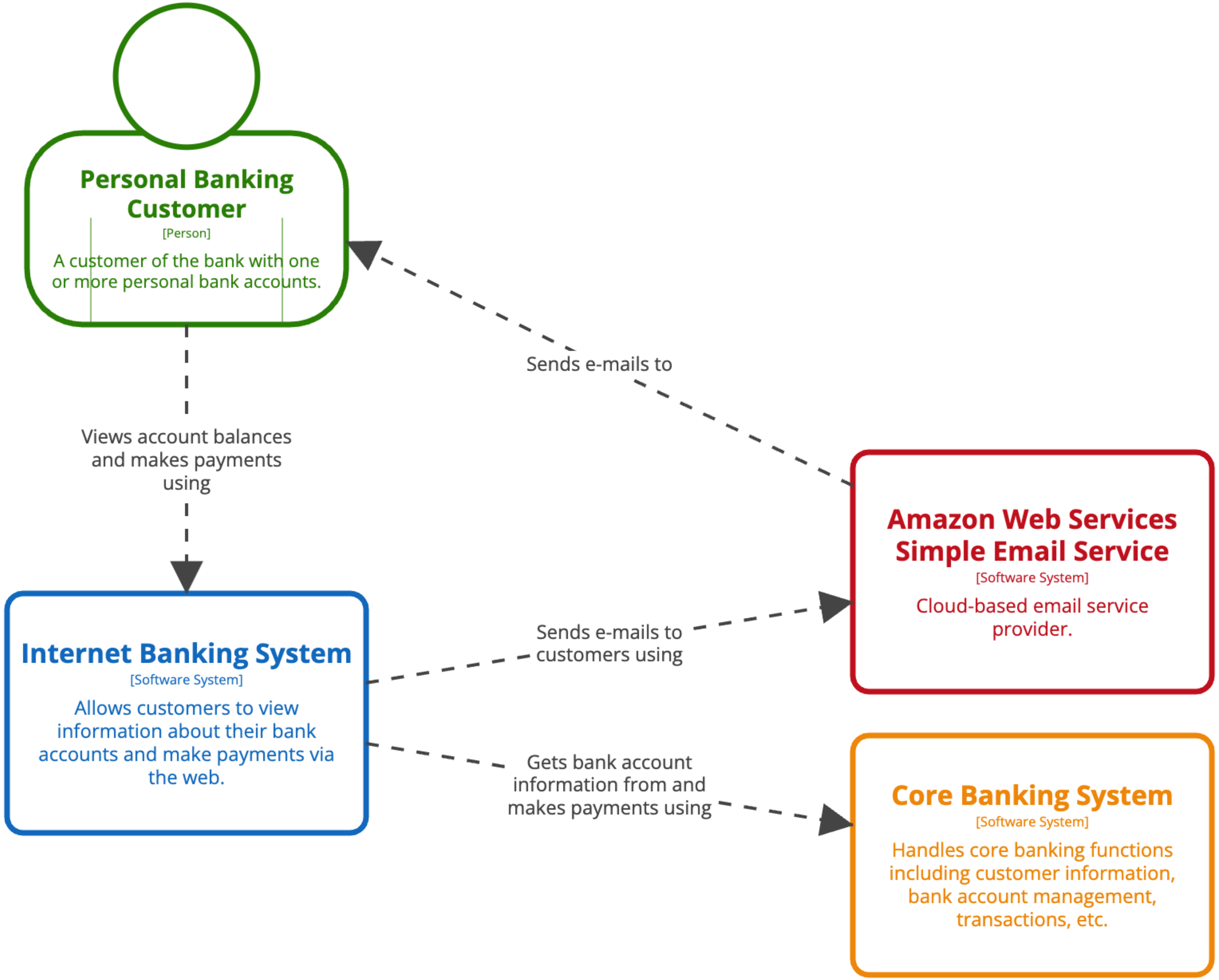
System Context View: Internet Banking System

The system context diagram for a fictional Internet Banking System


Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.
Views account balances and makes payments using
Internet Banking System
[Software System]
Allows customers to view information about their bank accounts and make payments via the web.
Gets bank account information from and makes payments using
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.

System Context View: Internet Banking System

The system context diagram for a fictional Internet Banking System


Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.
Views account balances and makes payments using
Sends e-mails to
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
Gets bank account information from and makes payments using
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.

System Context View: Internet Banking System

The system context diagram for a fictional Internet Banking System

Amazon Web Services
Simple Email Service

Core Banking System

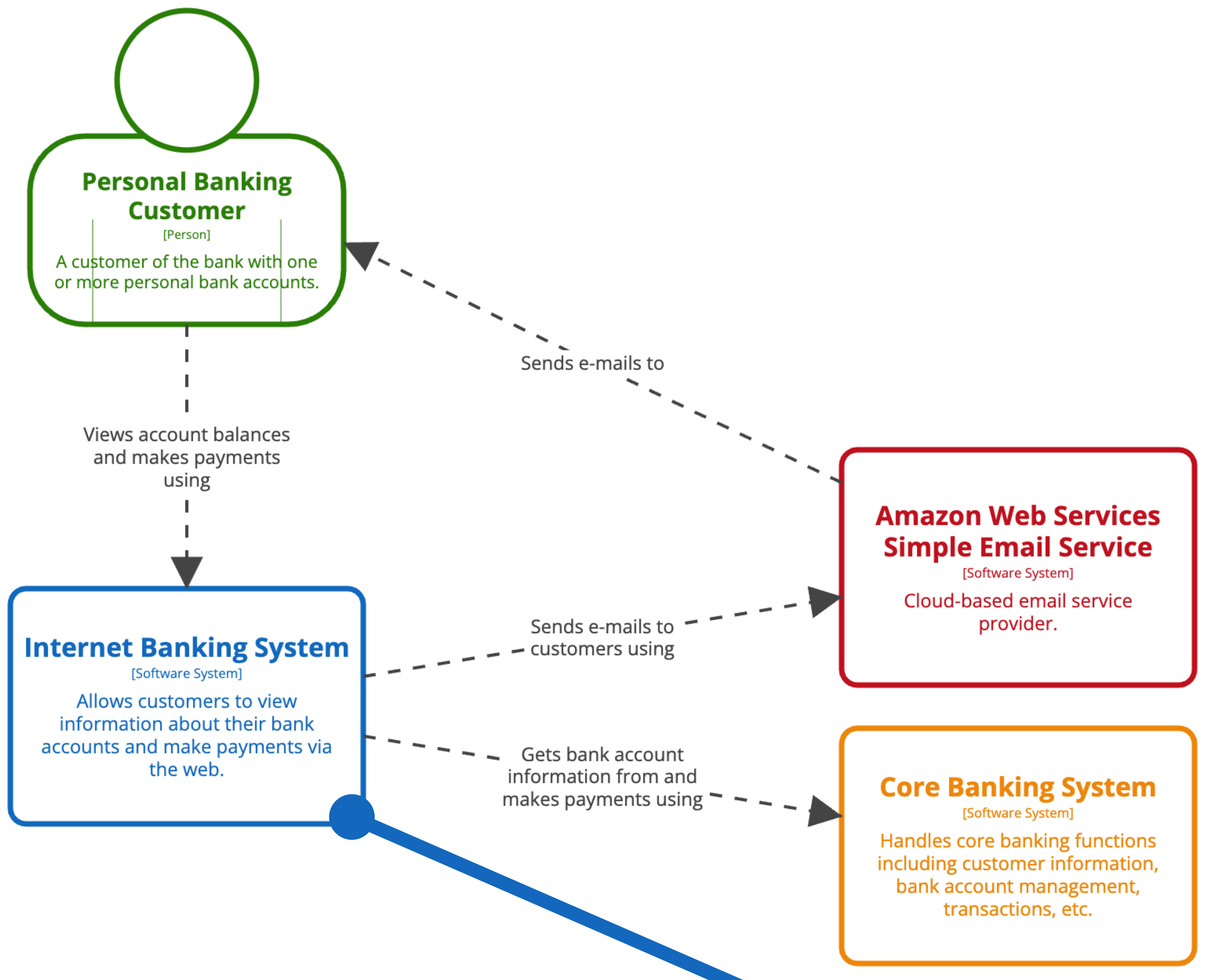
Internet Banking System

Person, Customer

Relationship

Level 2

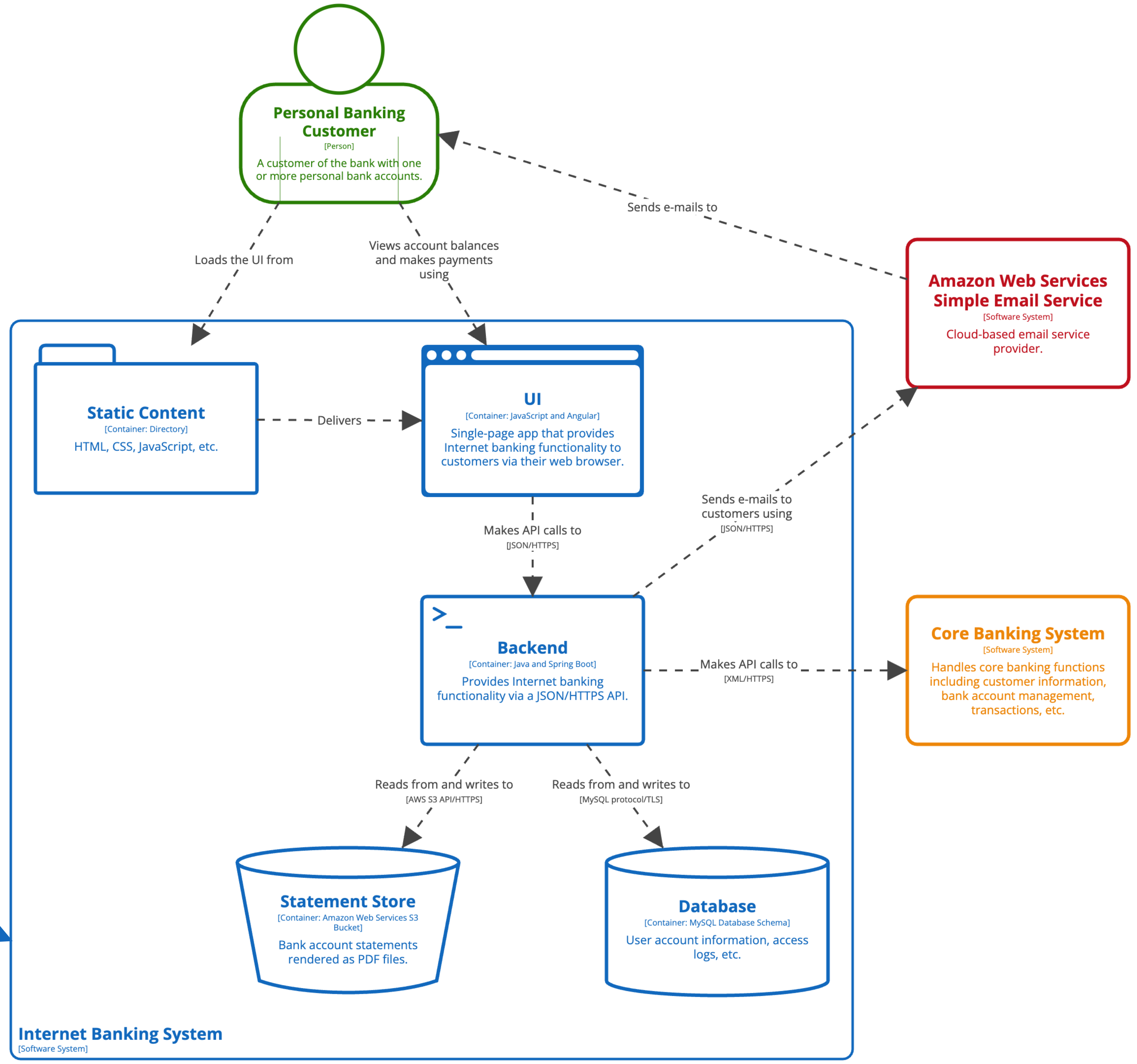
Container diagram


Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.
Sends e-mails to
Views account balances and makes payments using
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
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
Gets bank account information from and makes payments using
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.

System Context View: Internet Banking System

The system context diagram for a fictional Internet Banking System

Zoom in to a software system

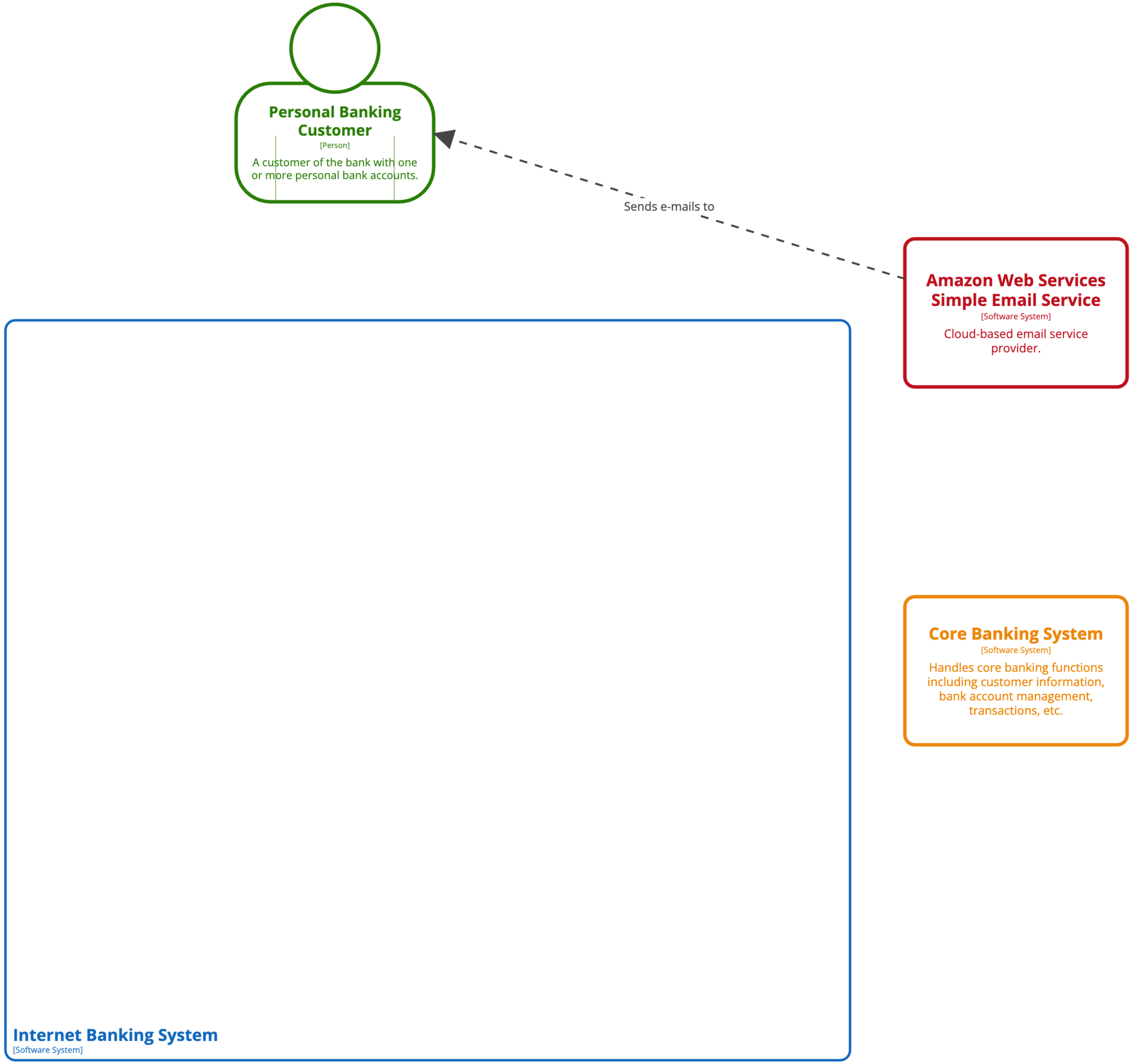

Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.
Sends e-mails to
Loads the UI from
Views account balances and makes payments using
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Static Content
[Container: Directory]
HTML, CSS, JavaScript, etc.
Delivers
UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Makes API calls to
[JSON/HTTPS]
Sends e-mails to customers using
[JSON/HTTPS]
Backend
[Container: Java and Spring Boot]
Provides Internet banking functionality via a JSON/HTTPS API.
Makes API calls to
[XML/HTTPS]
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
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.
Internet Banking System
[Software System]

Container View: Internet Banking System

The container diagram for the Internet Banking System

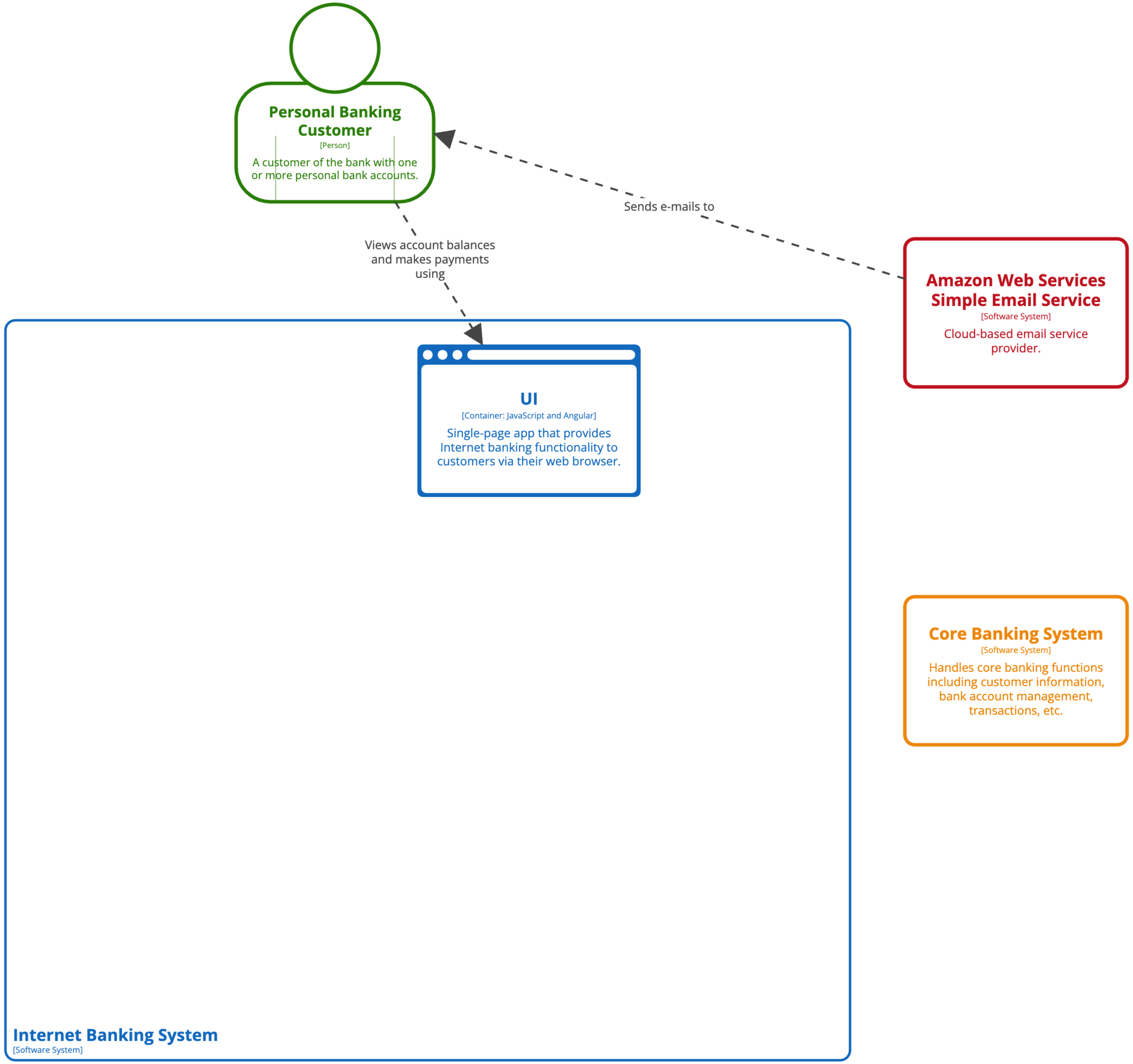
System context diagram

Container diagram


Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.
Sends e-mails to
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
Internet Banking System
[Software System]

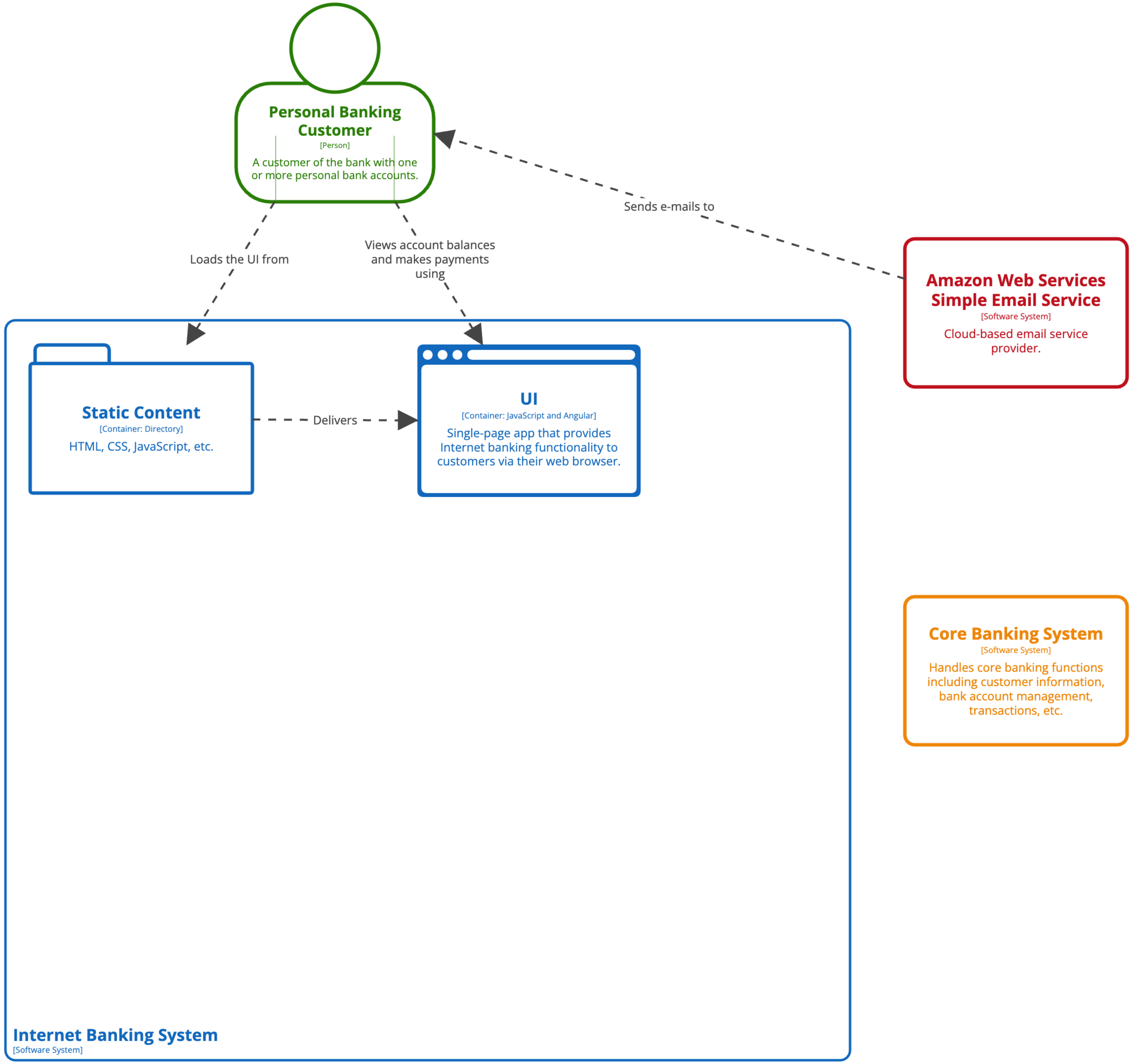
Container View: Internet Banking System

The container diagram for the Internet Banking System


Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.
Sends e-mails to
Views account balances and makes payments using
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
Internet Banking System
[Software System]

Container View: Internet Banking System

The container diagram for the Internet Banking System



Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.

Loads the UI from

Views account balances and makes payments using

Sends e-mails to

Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.

Static Content
[Container: Directory]
HTML, CSS, JavaScript, etc.

Delivers

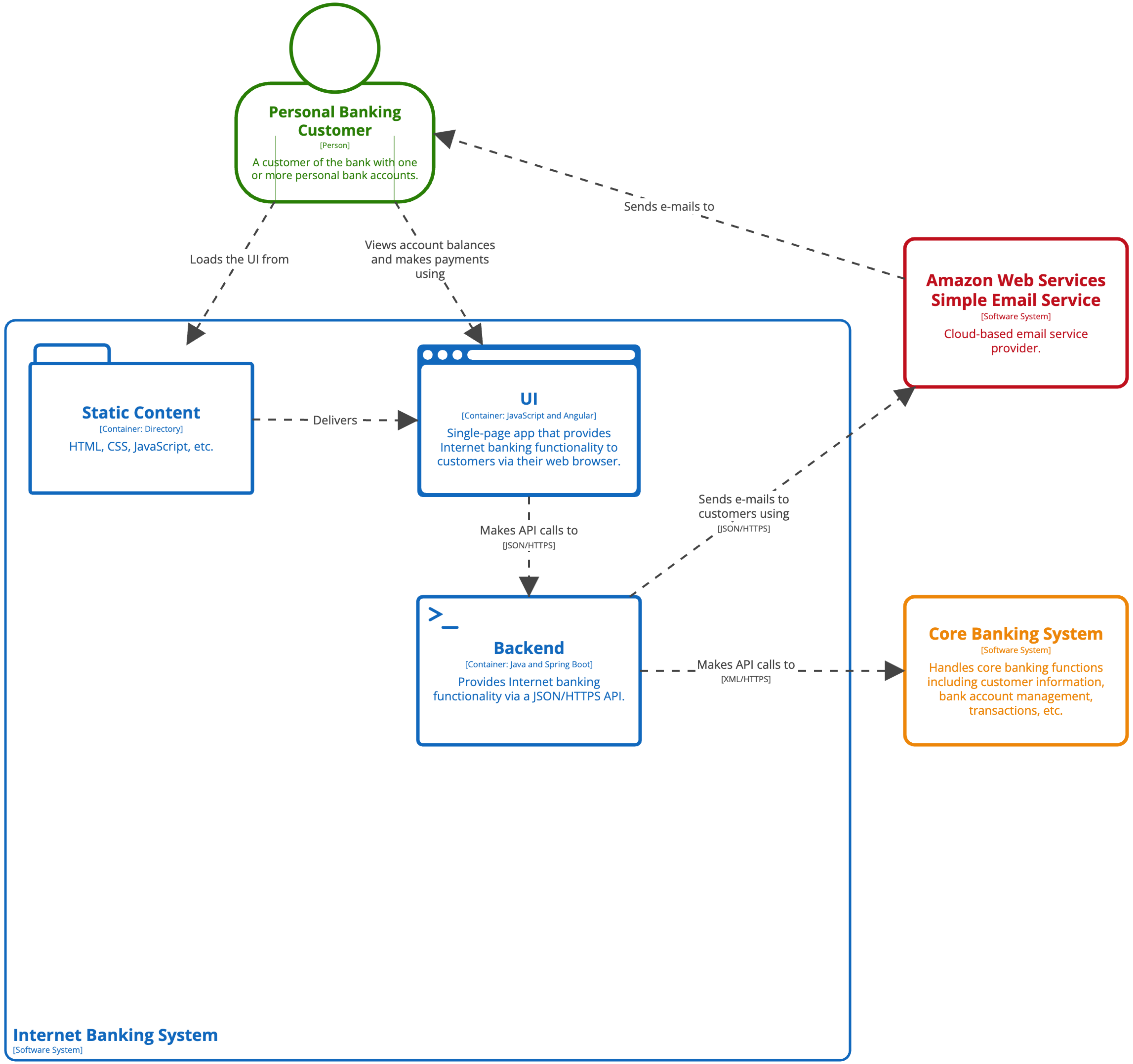
UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.

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

Internet Banking System
[Software System]

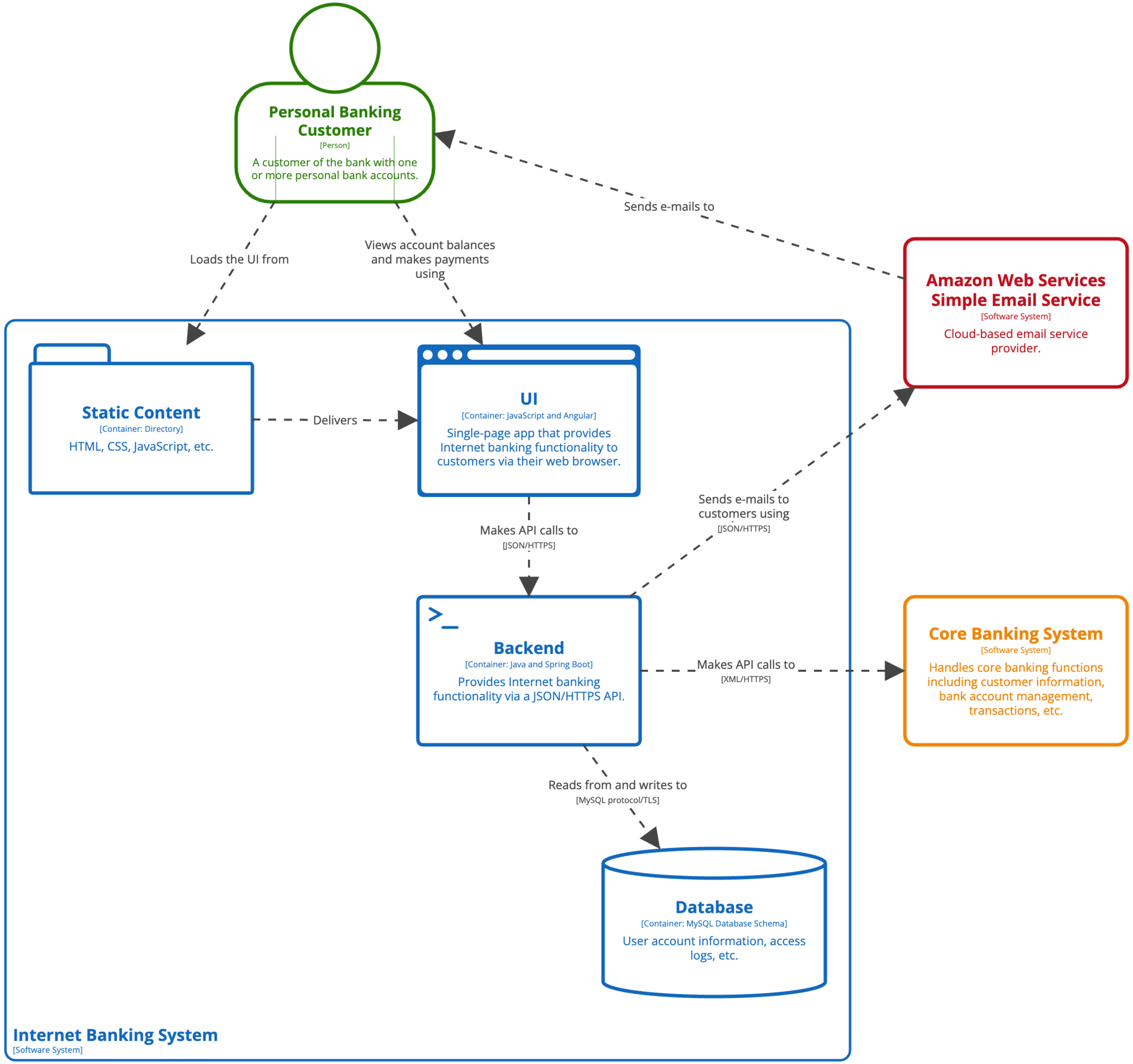
Container View: Internet Banking System

The container diagram for the Internet Banking System


Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.
Sends e-mails to
Loads the UI from
Views account balances and makes payments using
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Static Content
[Container: Directory]
HTML, CSS, JavaScript, etc.
Delivers
UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Makes API calls to
[JSON/HTTPS]
Sends e-mails to customers using
[JSON/HTTPS]
Backend
[Container: Java and Spring Boot]
Provides Internet banking functionality via a JSON/HTTPS API.
Makes API calls to
[XML/HTTPS]
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
Internet Banking System
[Software System]

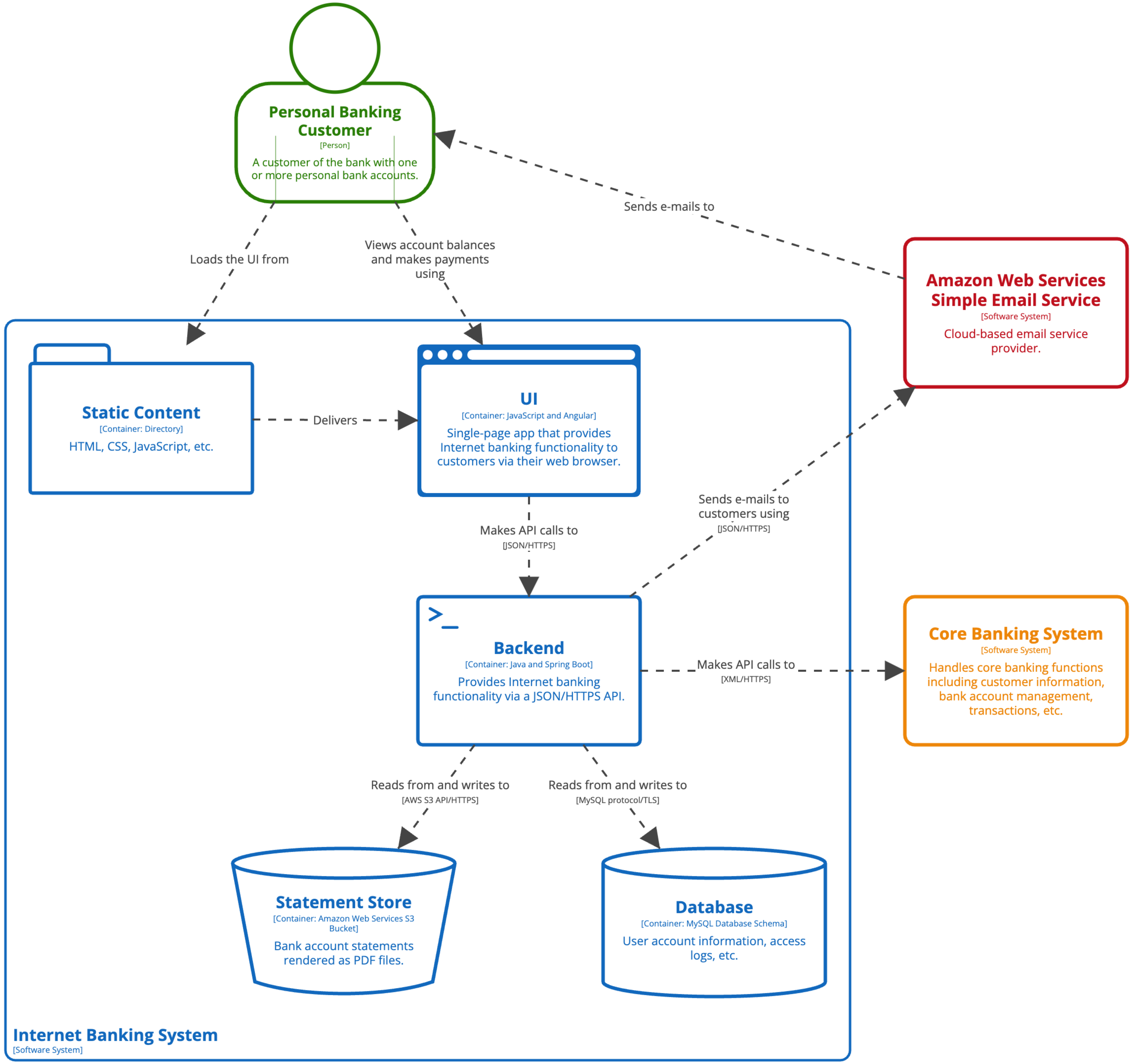
Container View: Internet Banking System

The container diagram for the Internet Banking System


Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.
Loads the UI from
Views account balances and makes payments using
Sends e-mails to
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Static Content
[Container: Directory]
HTML, CSS, JavaScript, etc.
Delivers
UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Makes API calls to
[JSON/HTTPS]
Sends e-mails to customers using
[JSON/HTTPS]
Backend
[Container: Java and Spring Boot]
Provides Internet banking functionality via a JSON/HTTPS API.
Makes API calls to
[XML/HTTPS]
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
Reads from and writes to
[MySQL protocol/TLS]
Database
[Container: MySQL Database Schema]
User account information, access logs, etc.
Internet Banking System
[Software System]

Container View: Internet Banking System

The container diagram for the Internet Banking System



Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.
Loads the UI from
Views account balances and makes payments using
Sends e-mails to
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Static Content
[Container: Directory]
HTML, CSS, JavaScript, etc.
Delivers
UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Makes API calls to
[JSON/HTTPS]
Sends e-mails to customers using
[JSON/HTTPS]
Backend
[Container: Java and Spring Boot]
Provides Internet banking functionality via a JSON/HTTPS API.
Makes API calls to
[XML/HTTPS]
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
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.
Internet Banking System
[Software System]

Container View: Internet Banking System

The container diagram for the Internet Banking System

Amazon Web Services Simple Email Service

Boundary, Internet Banking System

Container, Amazon Web Services S3 Bucket

Container, Directory

Container, Relational Database Schema

Container, Server-side Application

Container, Single-page Application

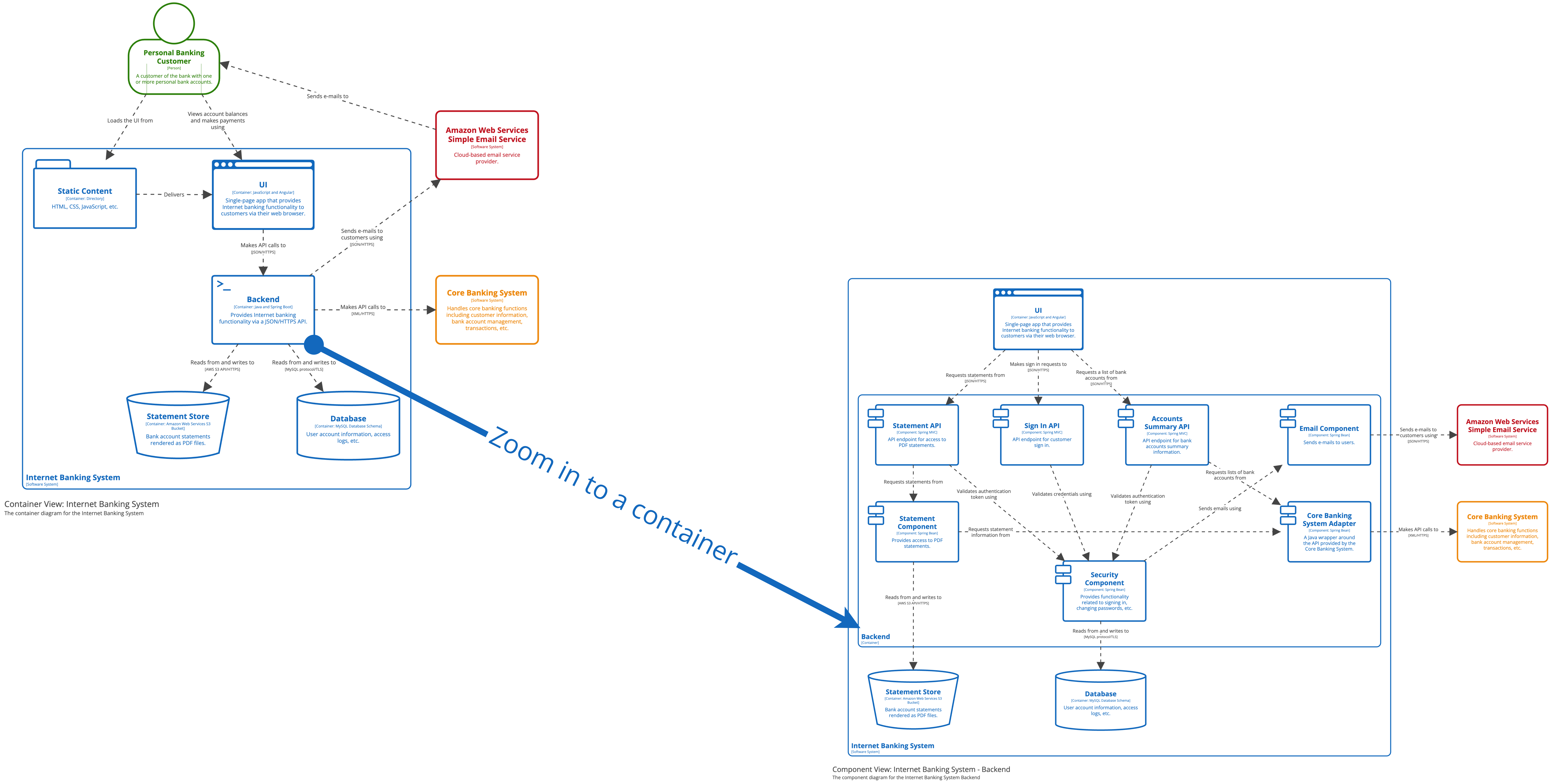
Core Banking System

Person, Customer

Relationship

Level 3

Component diagram



Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.

Loads the UI from

Views account balances and makes payments using

Sends e-mails to

Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.

Static Content
[Container: Directory]
HTML, CSS, JavaScript, etc.

Delivers

UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.

Makes API calls to
[JSON/HTTPS]

Sends e-mails to customers using
[JSON/HTTPS]

Backend
[Container: Java and Spring Boot]
Provides Internet banking functionality via a JSON/HTTPS API.

Makes API calls to
[XML/HTTPS]

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

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.

Internet Banking System
[Software System]

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

Zoom in to a container

UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.

Requests statements from
[JSON/HTTPS]

Makes sign in requests to
[JSON/HTTPS]

Requests a list of bank accounts from
[JSON/HTTPS]

Statement API
[Component: Spring MVC]
API endpoint for access to PDF statements.

Sign In API
[Component: Spring MVC]
API endpoint for customer sign in.

Accounts Summary API
[Component: Spring MVC]
API endpoint for bank accounts summary information.

Email Component
[Component: Spring Bean]
Sends e-mails to users.

Sends e-mails to customers using
[JSON/HTTPS]

Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.

Requests statements from

Validates authentication token using

Validates credentials using

Validates authentication token using

Requests lists of bank accounts from

Sends emails using

Statement Component
[Component: Spring Bean]
Provides access to PDF statements.

Requests statement information from

Core Banking System Adapter
[Component: Spring Bean]
A Java wrapper around the API provided by the Core Banking System.

Makes API calls to
[XML/HTTPS]

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

Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.

Reads from and writes to
[AWS S3 API/HTTPS]

Reads from and writes to
[MySQL protocol/TLS]

Backend
[Container]

Statement Store
[Container: Amazon Web Services S3 Bucket]
Bank account statements rendered as PDF files.

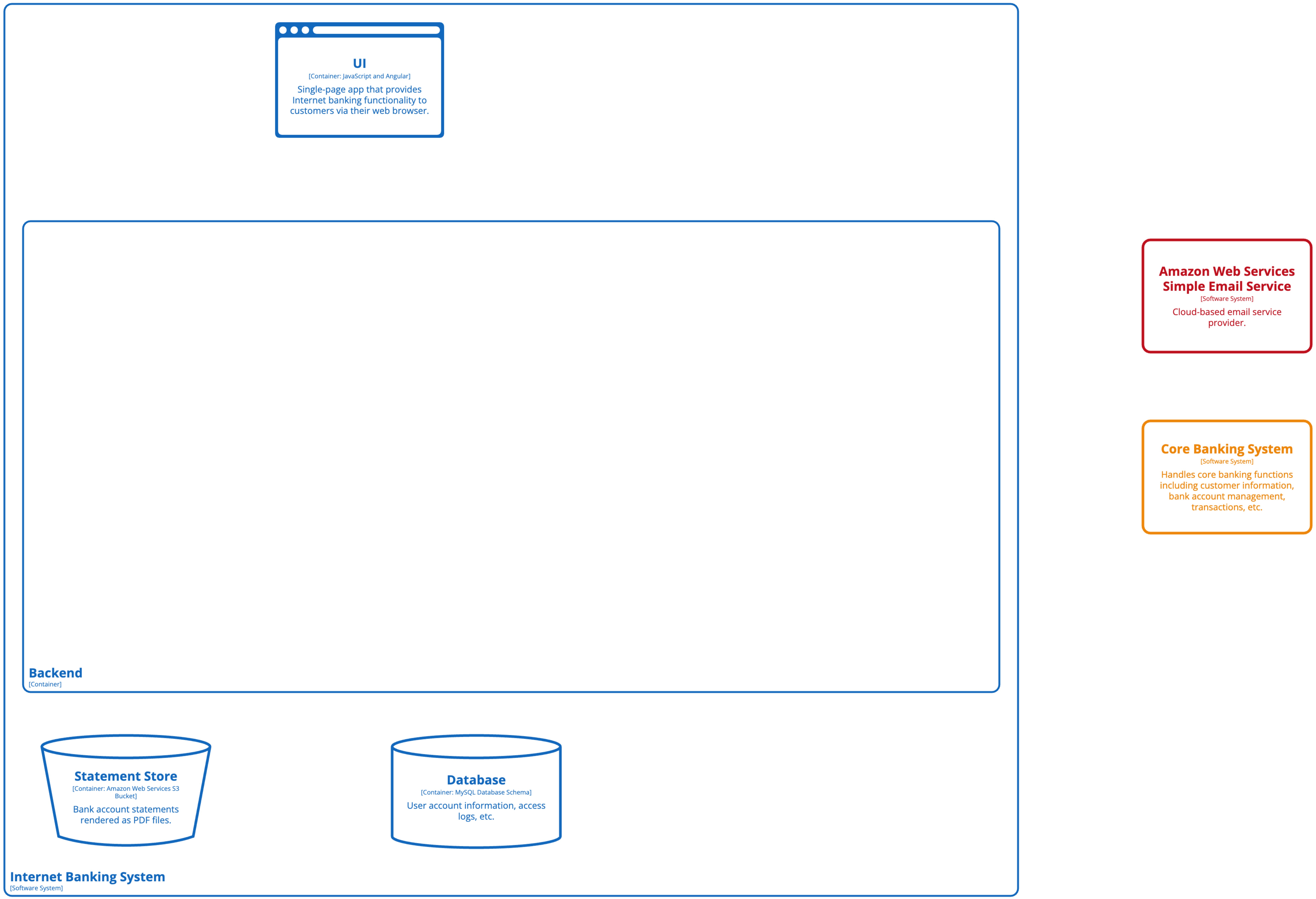
Database
[Container: MySQL Database Schema]
User account information, access logs, etc.

Internet Banking System
[Software System]

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

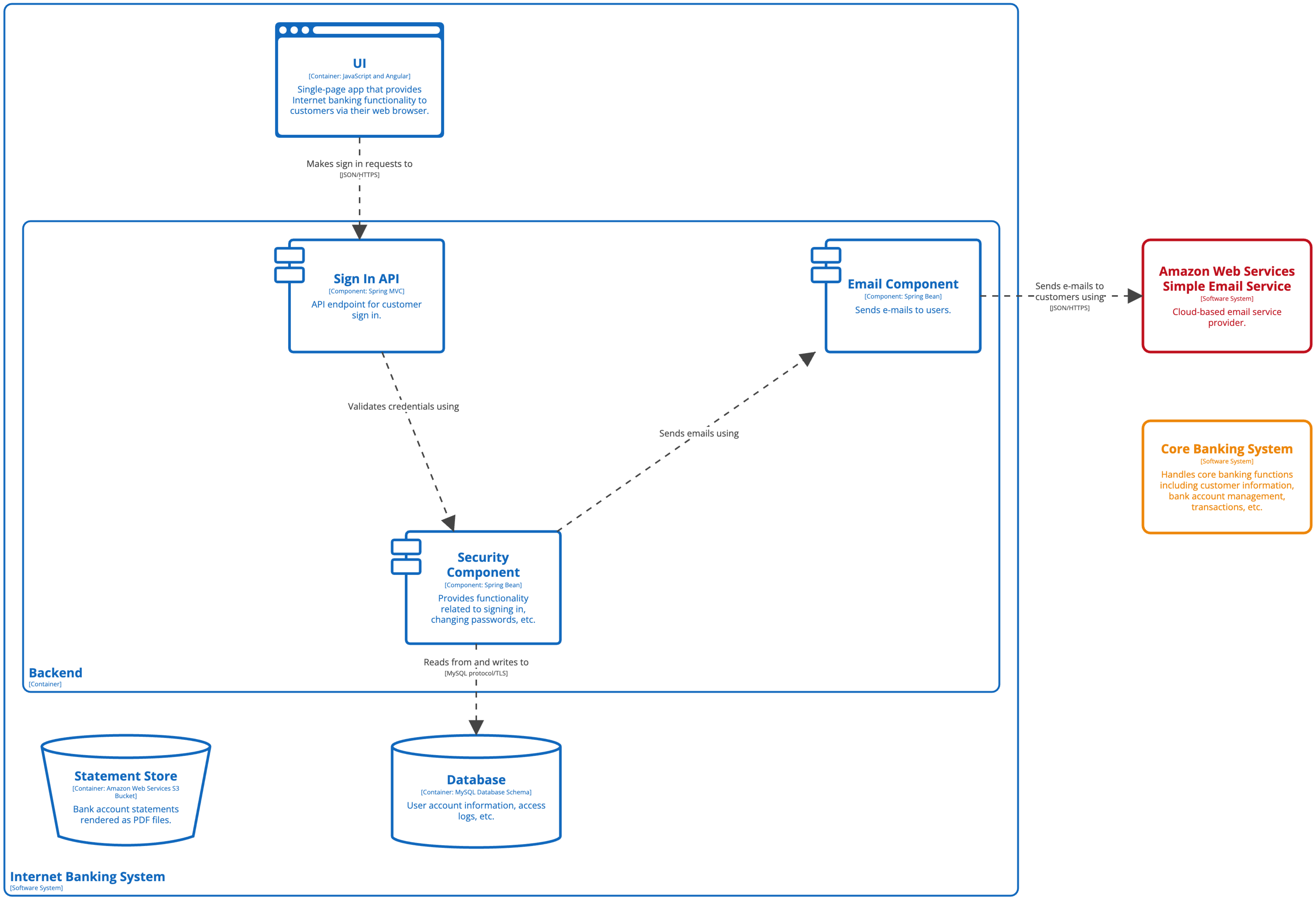
Container diagram

Component diagram


UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Backend
[Container]
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.
Internet Banking System
[Software System]
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.

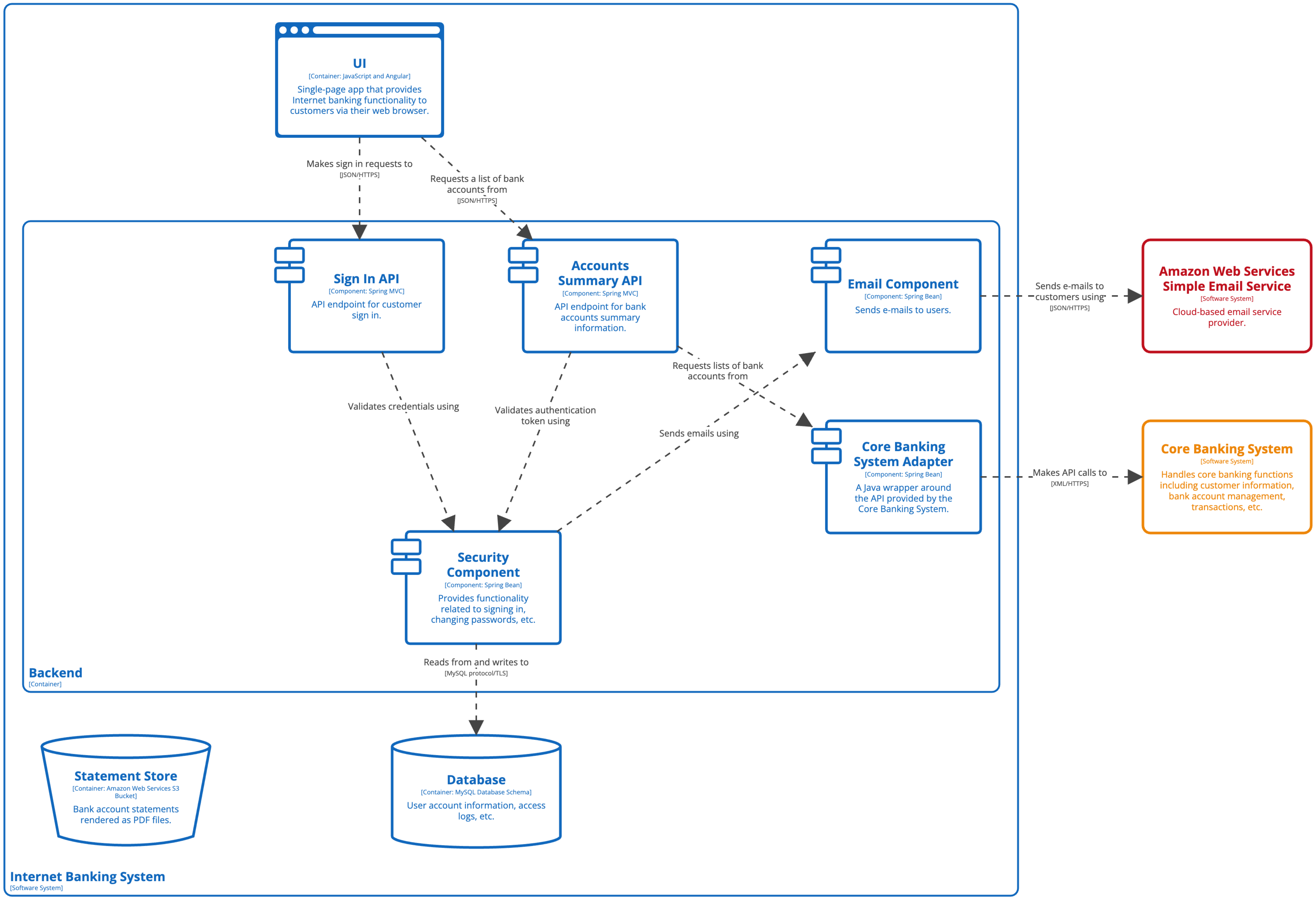
Component View: Internet Banking System - Backend

The component diagram for the Internet Banking System Backend


UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Makes sign in requests to
[JSON/HTTPS]
Sign In API
[Component: Spring MVC]
API endpoint for customer sign in.
Email Component
[Component: Spring Bean]
Sends e-mails to users.
Sends e-mails to customers using
[JSON/HTTPS]
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Validates credentials using
Sends emails using
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.
Reads from and writes to
[MySQL protocol/TLS]
Backend
[Container]
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.
Internet Banking System
[Software System]

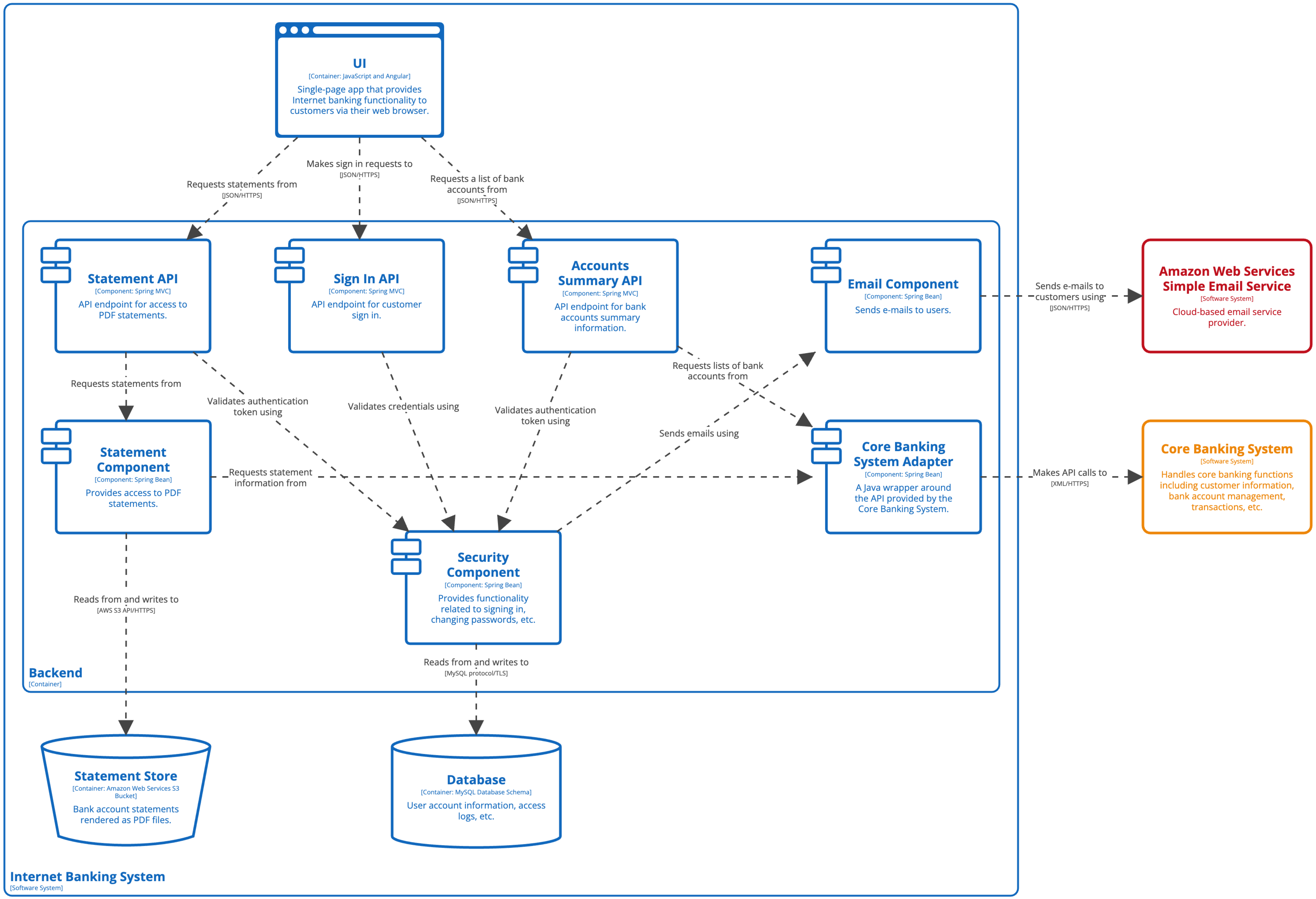
Component View: Internet Banking System - Backend

The component diagram for the Internet Banking System Backend


UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Makes sign in requests to
[JSON/HTTPS]
Requests a list of bank accounts from
[JSON/HTTPS]
Sign In API
[Component: Spring MVC]
API endpoint for customer sign in.
Accounts Summary API
[Component: Spring MVC]
API endpoint for bank accounts summary information.
Email Component
[Component: Spring Bean]
Sends e-mails to users.
Sends e-mails to customers using
[JSON/HTTPS]
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Requests lists of bank accounts from
Validates credentials using
Validates authentication token using
Sends emails using
Core Banking System Adapter
[Component: Spring Bean]
A Java wrapper around the API provided by the Core Banking System.
Makes API calls to
[XML/HTTPS]
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.
Reads from and writes to
[MySQL protocol/TLS]
Backend
[Container]
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.
Internet Banking System
[Software System]

Component View: Internet Banking System - Backend

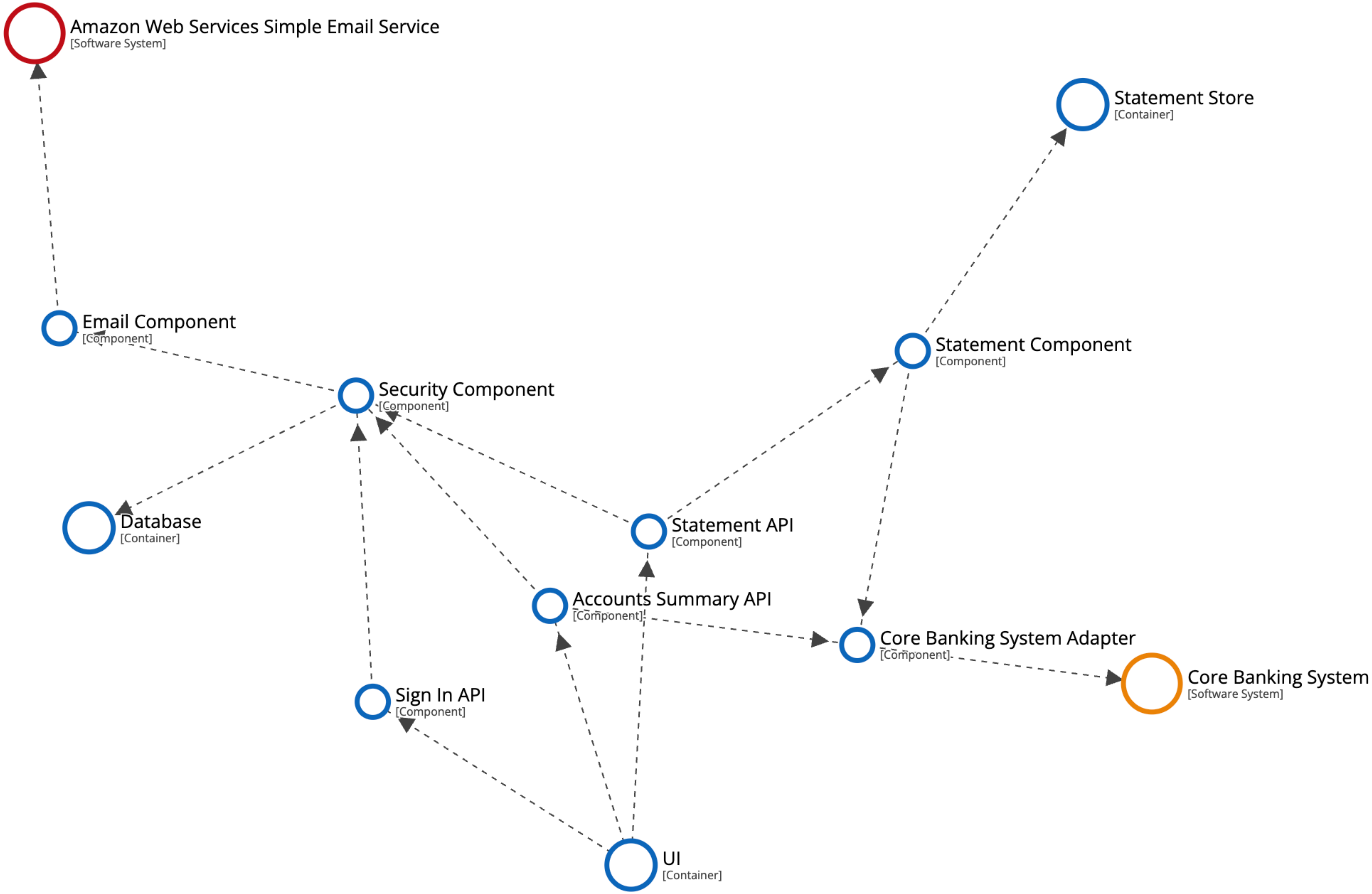
The component diagram for the Internet Banking System Backend


UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Requests statements from
[JSON/HTTPS]
Makes sign in requests to
[JSON/HTTPS]
Requests a list of bank accounts from
[JSON/HTTPS]
Statement API
[Component: Spring MVC]
API endpoint for access to PDF statements.
Sign In API
[Component: Spring MVC]
API endpoint for customer sign in.
Accounts Summary API
[Component: Spring MVC]
API endpoint for bank accounts summary information.
Email Component
[Component: Spring Bean]
Sends e-mails to users.
Sends e-mails to customers using
[JSON/HTTPS]
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Requests statements from
Validates authentication token using
Validates credentials using
Validates authentication token using
Requests lists of bank accounts from
Sends emails using
Statement Component
[Component: Spring Bean]
Provides access to PDF statements.
Requests statement information from
Core Banking System Adapter
[Component: Spring Bean]
A Java wrapper around the API provided by the Core Banking System.
Makes API calls to
[XML/HTTPS]
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.
Reads from and writes to
[AWS S3 API/HTTPS]
Reads from and writes to
[MySQL protocol/TLS]
Backend
[Container]
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.
Internet Banking System
[Software System]

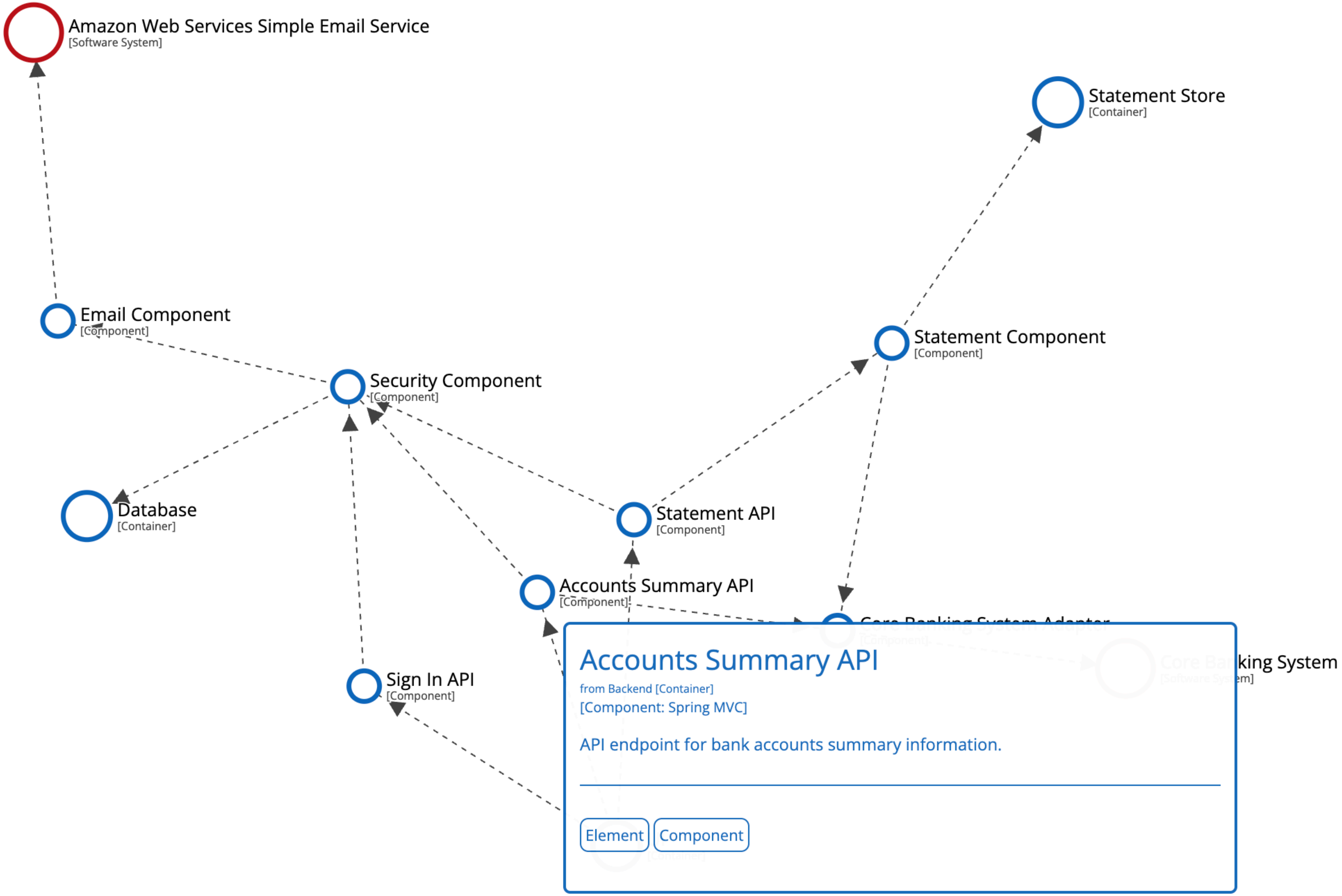
Component View: Internet Banking System - Backend

The component diagram for the Internet Banking System Backend

Amazon Web Services Simple Email Service
Boundary, Container, Server-side Application
Boundary, Internet Banking System
Component
Container, Amazon Web Services S3 Bucket
Container, Relational Database Schema
Container, Single-page Application
Core Banking System
Relationship



Amazon Web Services Simple Email Service
[Software System]
Statement Store
[Container]
Email Component
[Component]
Statement Component
[Component]
Security Component
[Component]
Database
[Container]
Statement API
[Component]
Accounts Summary API
[Component]
Core Banking System Adapter
[Component]
Core Banking System
[Software System]
Sign In API
[Component]
UI
[Container]

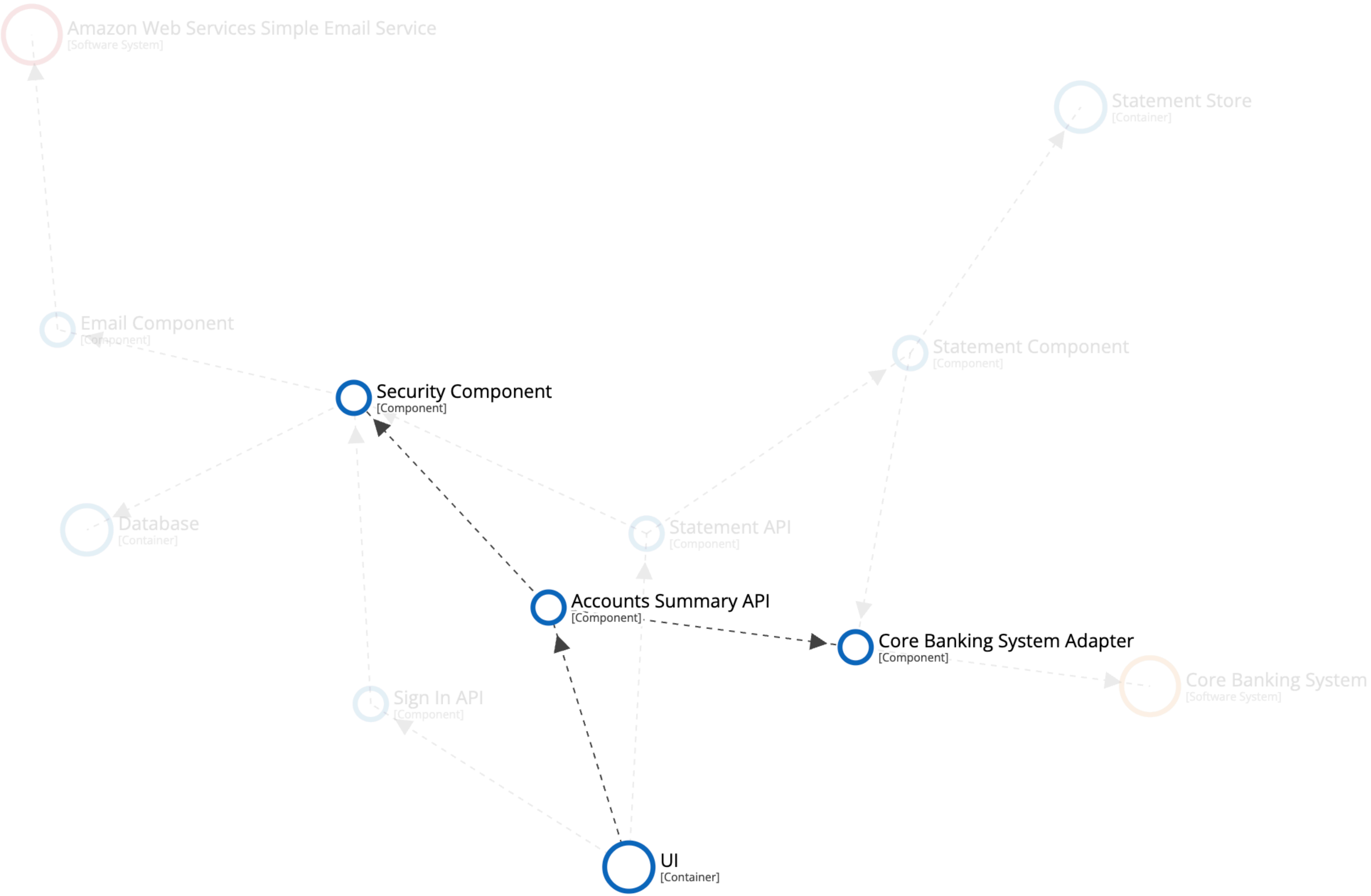


Accounts Summary API

from Backend [Container]

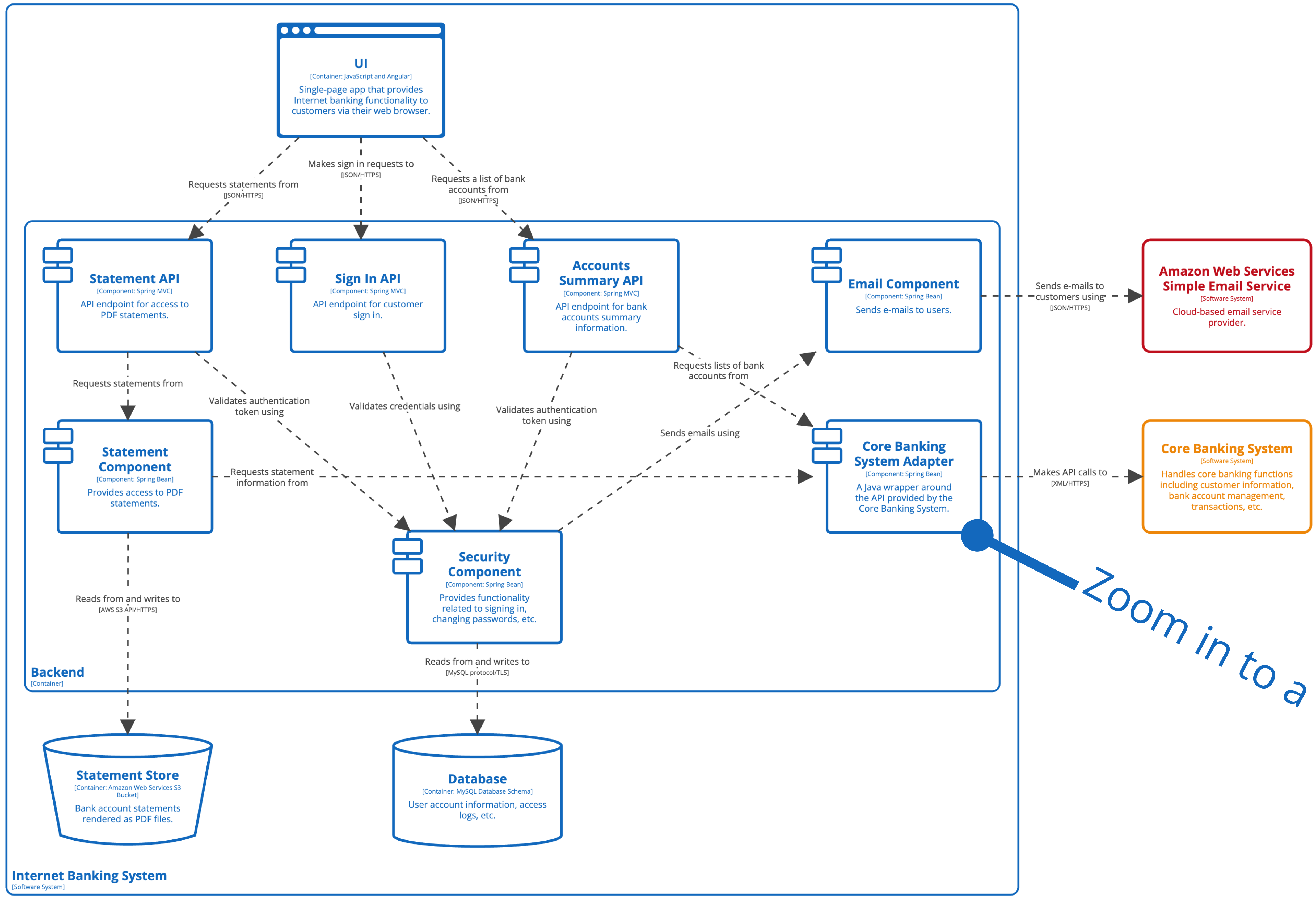
[Component: Spring MVC]

API endpoint for bank accounts summary information.


Amazon Web Services Simple Email Service
[Software System]
Statement Store
[Container]
Email Component
[Component]
Statement Component
[Component]
Security Component
[Component]
Database
[Container]
Statement API
[Component]
Accounts Summary API
[Component]
Core Banking System Adapter
[Component]
Core Banking System
[Software System]
Sign In API
[Component]
UI
[Container]

Level 4

Code diagram



UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.

Requests statements from
[JSON/HTTPS]

Makes sign in requests to
[JSON/HTTPS]

Requests a list of bank accounts from
[JSON/HTTPS]

Statement API
[Component: Spring MVC]
API endpoint for access to PDF statements.

Sign In API
[Component: Spring MVC]
API endpoint for customer sign in.

Accounts Summary API
[Component: Spring MVC]
API endpoint for bank accounts summary information.

Email Component
[Component: Spring Bean]
Sends e-mails to users.

Sends e-mails to customers using
[JSON/HTTPS]

Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.

Requests statements from

Validates authentication token using

Validates credentials using

Validates authentication token using

Requests lists of bank accounts from

Sends emails using

Statement Component
[Component: Spring Bean]
Provides access to PDF statements.

Requests statement information from

Core Banking System Adapter
[Component: Spring Bean]
A Java wrapper around the API provided by the Core Banking System.

Makes API calls to
[XML/HTTPS]

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

Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.

Reads from and writes to
[AWS S3 API/HTTPS]

Reads from and writes to
[MySQL protocol/TLS]

Backend
[Container]

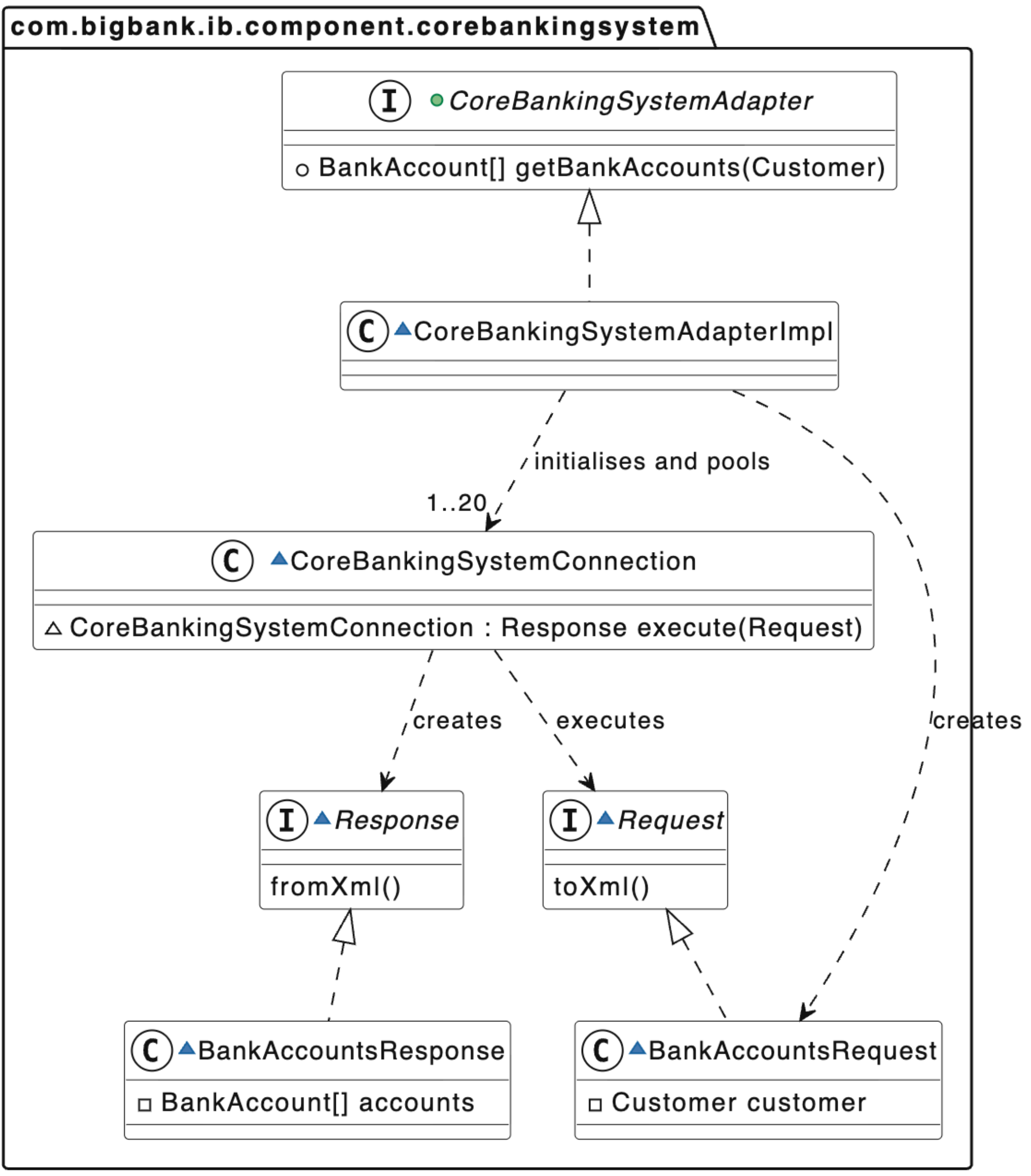
Statement Store
[Container: Amazon Web Services S3 Bucket]
Bank account statements rendered as PDF files.

Database
[Container: MySQL Database Schema]
User account information, access logs, etc.

Internet Banking System
[Software System]

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

Zoom in to a component



com.bigbank.ib.component.corebankingsystem

I CoreBankingSystemAdapter
○ BankAccount[] getBankAccounts(Customer)

C CoreBankingSystemAdapterImpl

initialises and pools
1..20

C CoreBankingSystemConnection
△ CoreBankingSystemConnection : Response execute(Request)

creates

executes

creates

I Response
fromXml()

I Request
toXml()

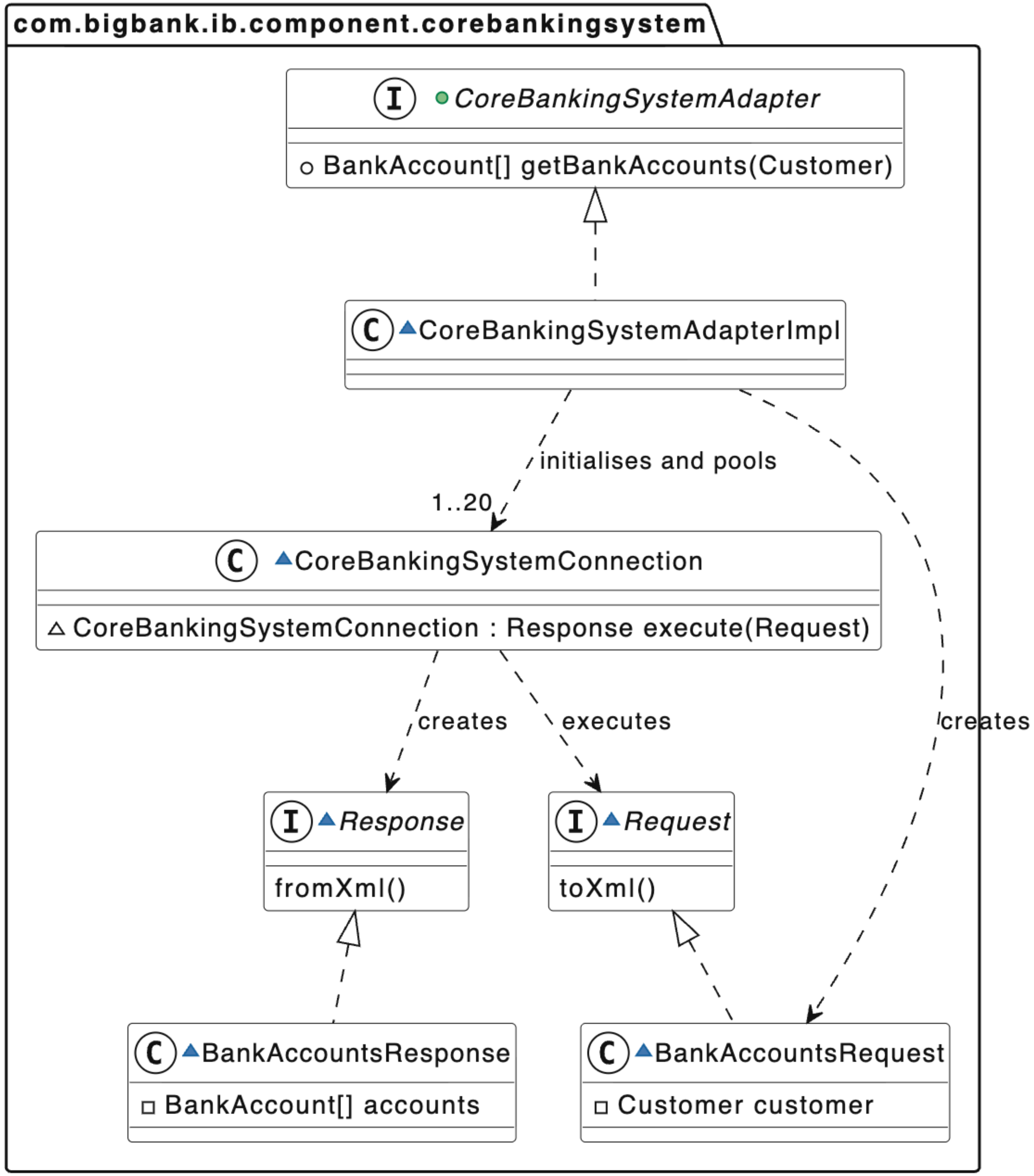
C BankAccountsResponse
□ BankAccount[] accounts

C BankAccountsRequest
□ Customer customer

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


com.bigbank.ib.component.corebankingsystem
I CoreBankingSystemAdapter
BankAccount[] getBankAccounts(Customer)
C CoreBankingSystemAdapterImpl
initialises and pools
1..20
C CoreBankingSystemConnection
CoreBankingSystemConnection : Response execute(Request)
creates
executes
creates
I Response
fromXml()
I Request
toXml()
C BankAccountsResponse
BankAccount[] accounts
C BankAccountsRequest
Customer customer

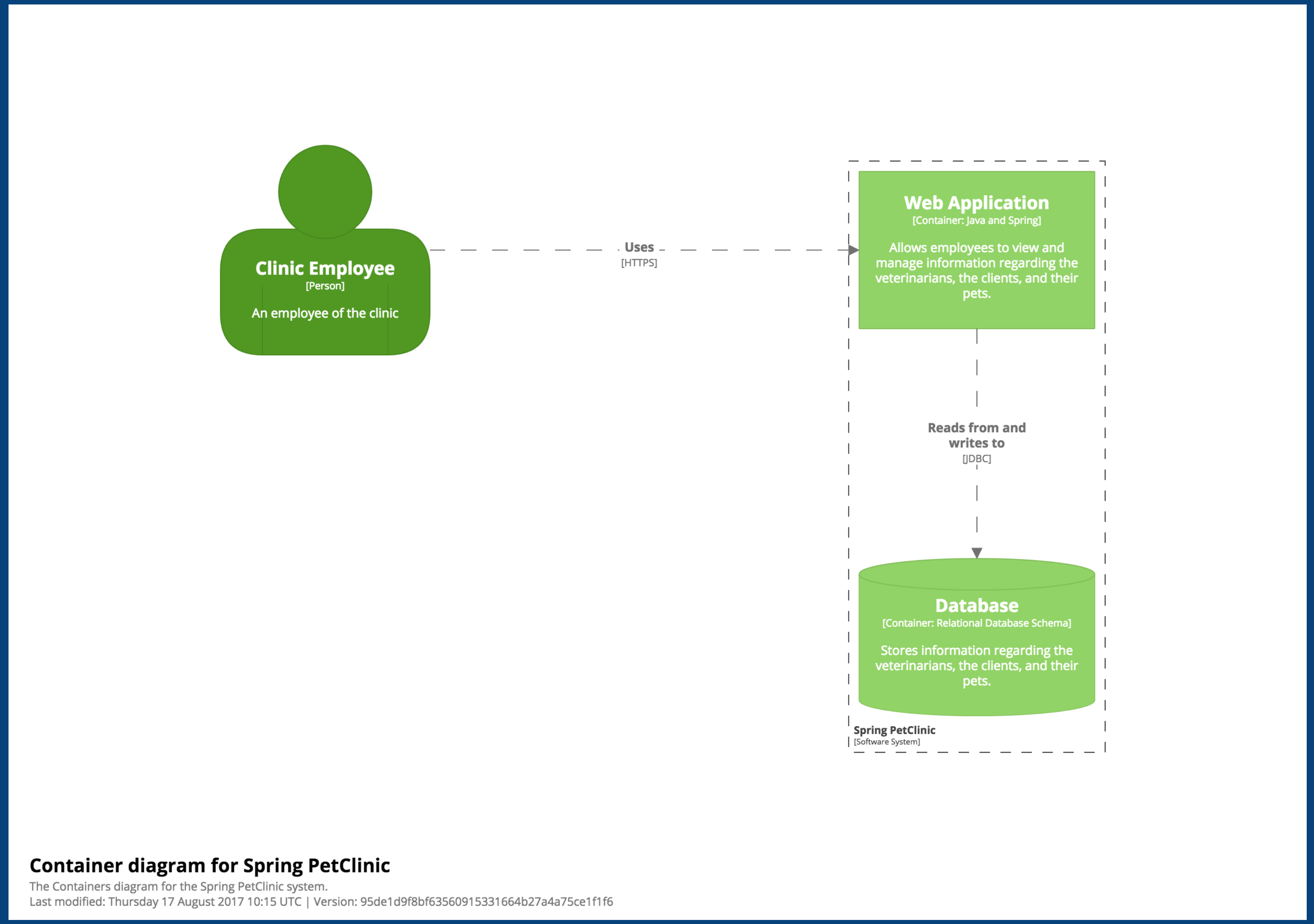
Code View: Internet Banking System - Backend - Core Banking System Adapter

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

Notation

The C4 model is notation independent

The C4 model is notation independent



Clinic Employee
[Person]
An employee of the clinic

Uses
[HTTPS]

Web Application
[Container: Java and Spring]
Allows employees to view and manage information regarding the veterinarians, the clients, and their pets.

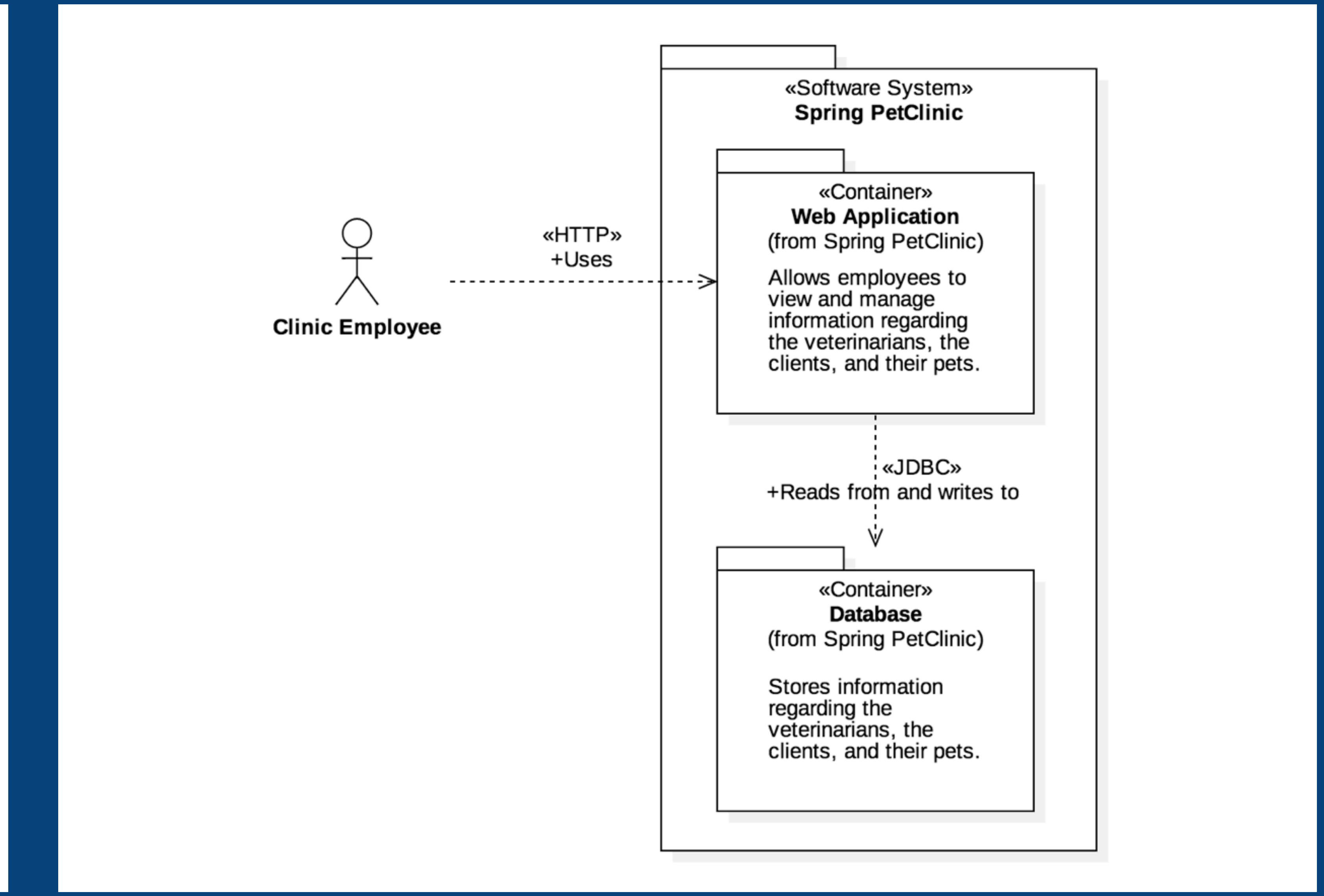
Reads from and writes to
[JDBC]

Database
[Container: Relational Database Schema]
Stores information regarding the veterinarians, the clients, and their pets.

Spring PetClinic
[Software System]

Container diagram for Spring PetClinic

The Containers diagram for the Spring PetClinic system.
Last modified: Thursday 17 August 2017 10:15 UTC | Version: 95de1d9f8bf63560915331664b27a4a75ce1f1f6



«Software System»
Spring PetClinic

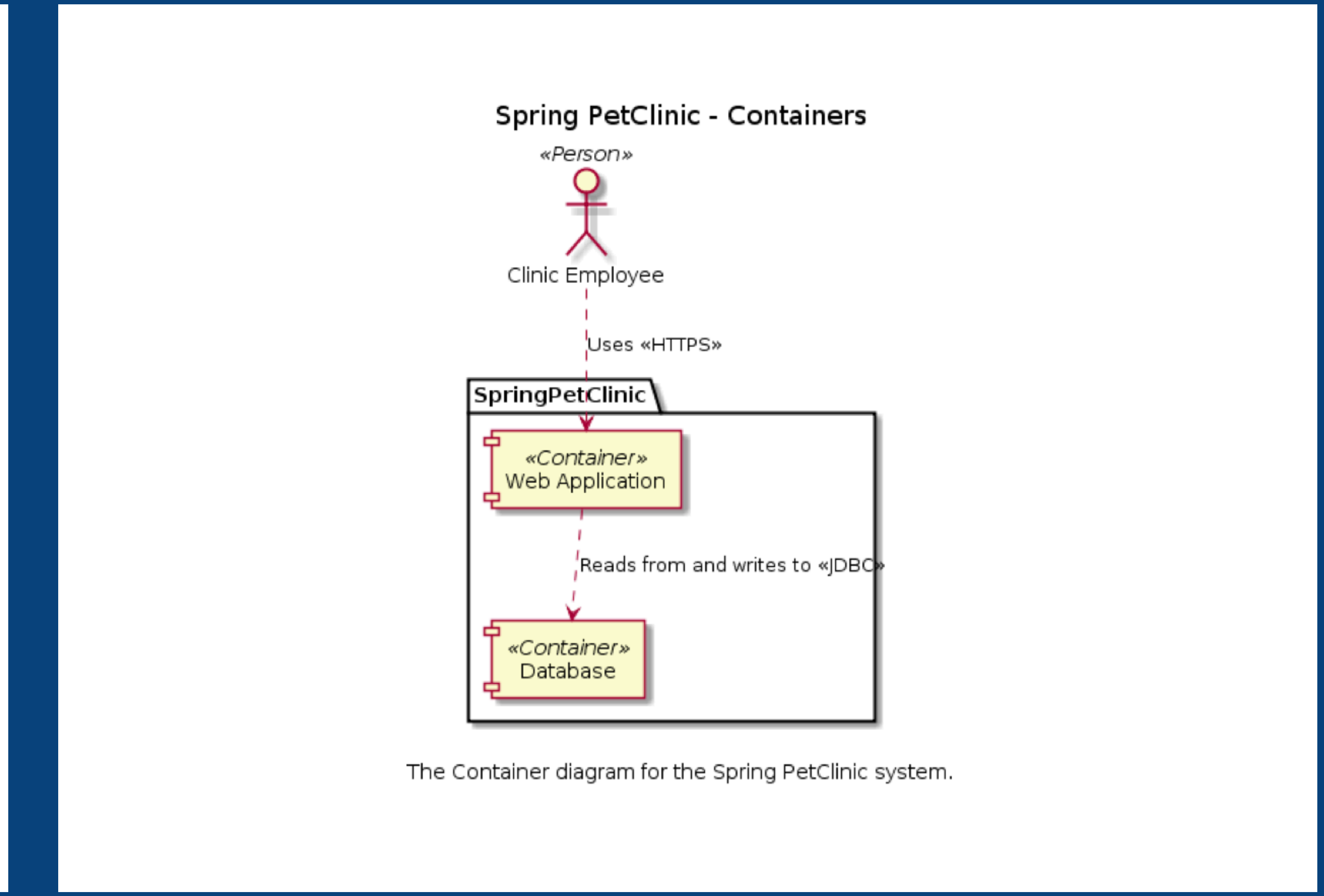
Clinic Employee

«HTTP»
+Uses

«Container»
Web Application
(from Spring PetClinic)
Allows employees to view and manage information regarding the veterinarians, the clients, and their pets.

«JDBC»
+Reads from and writes to

«Container»
Database
(from Spring PetClinic)
Stores information regarding the veterinarians, the clients, and their pets.



Spring PetClinic - Containers

«Person»
Clinic Employee

Uses «HTTPS»

SpringPetClinic

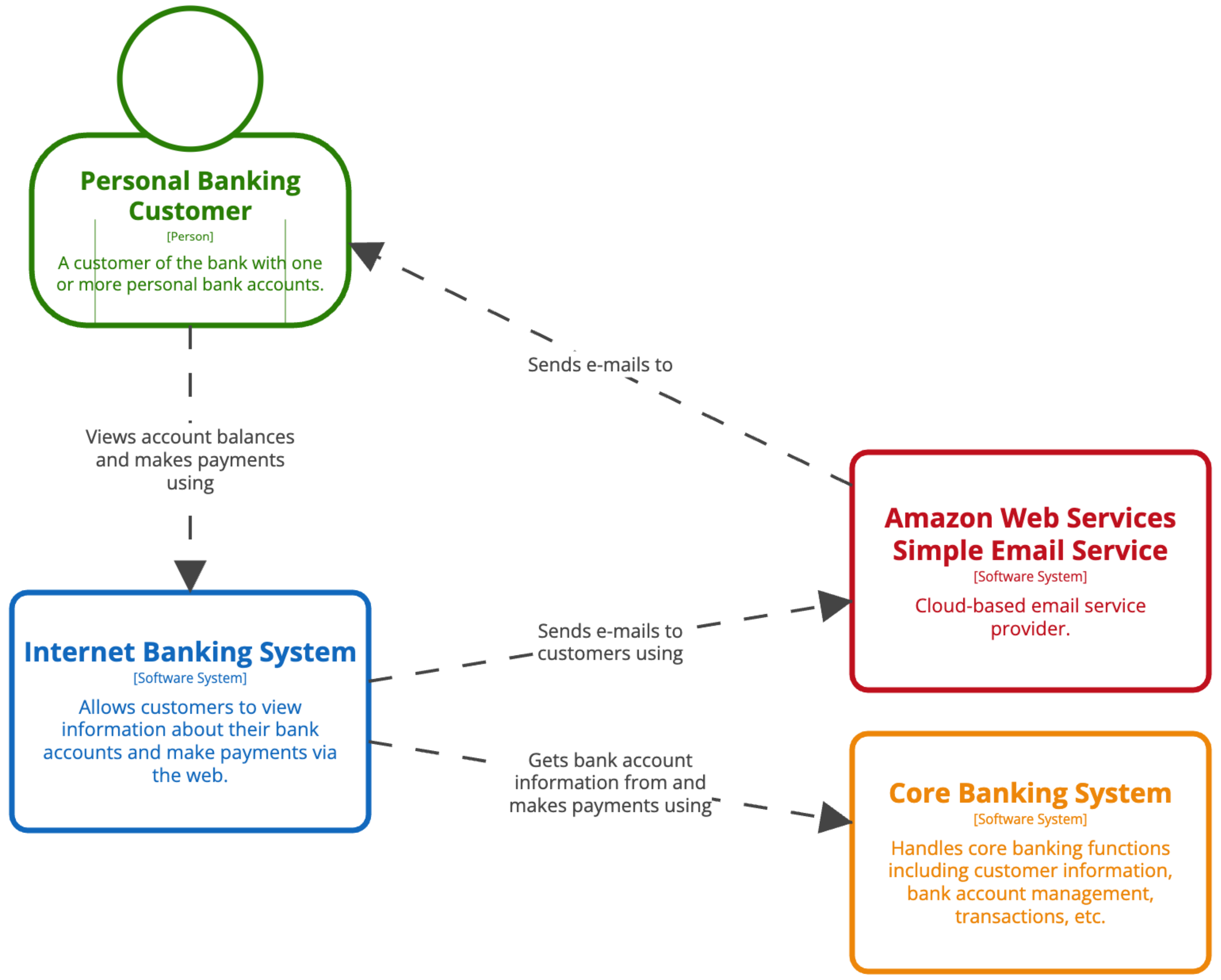
«Container»
Web Application

Reads from and writes to «JDBC»

«Container»
Database

The Container diagram for the Spring PetClinic system.

Should you adopt a standard (visual) notation?



Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.

Views account balances and makes payments using

Sends e-mails to

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

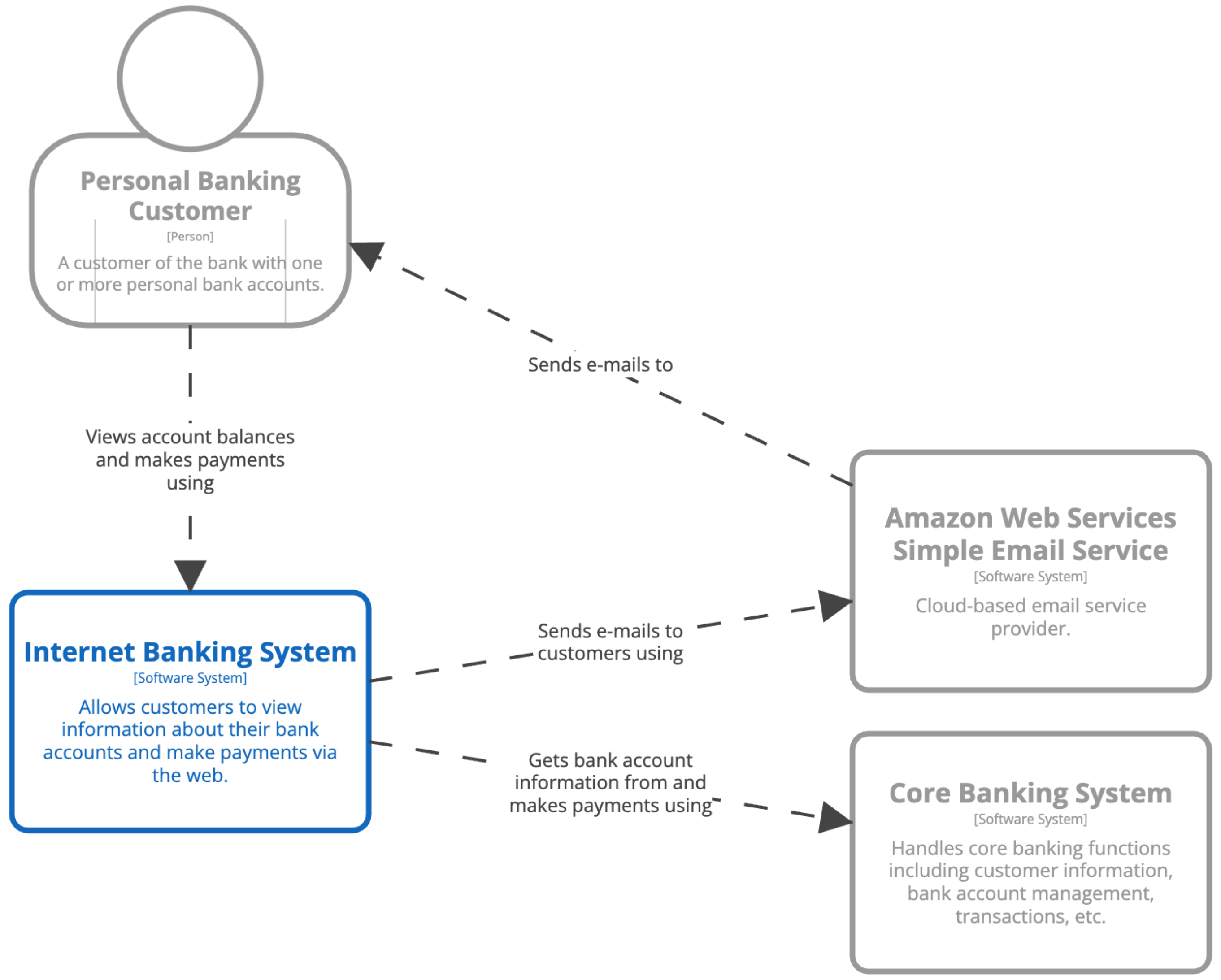
Gets bank account information from and makes payments using

Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.

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

[System Context] Internet Banking System

The system context diagram for a fictional Internet Banking System.



Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.

Views account balances and makes payments using

Sends e-mails to

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

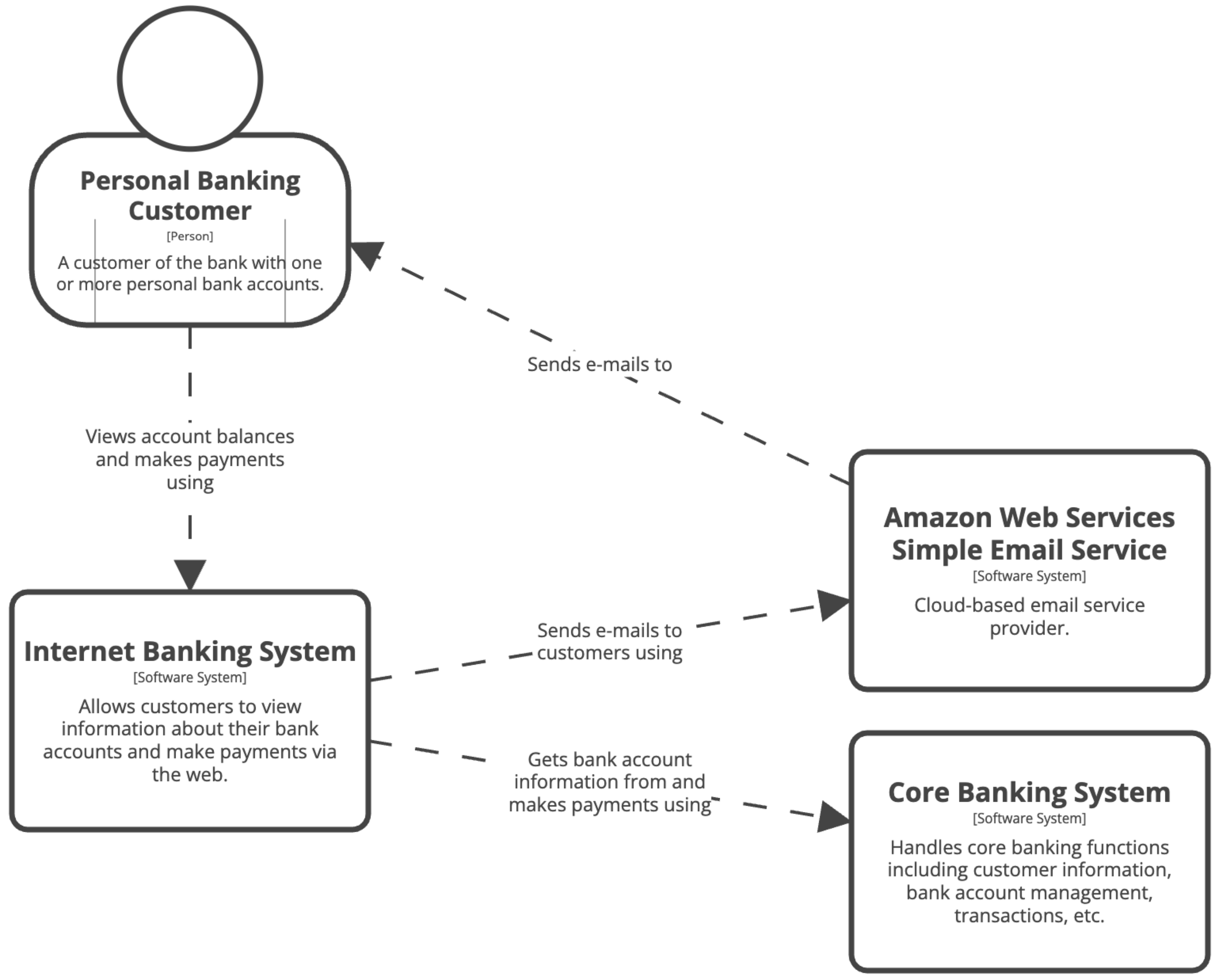
Gets bank account information from and makes payments using

Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.

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

[System Context] Internet Banking System

The system context diagram for a fictional Internet Banking System.



Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.

Views account balances and makes payments using

Sends e-mails to

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

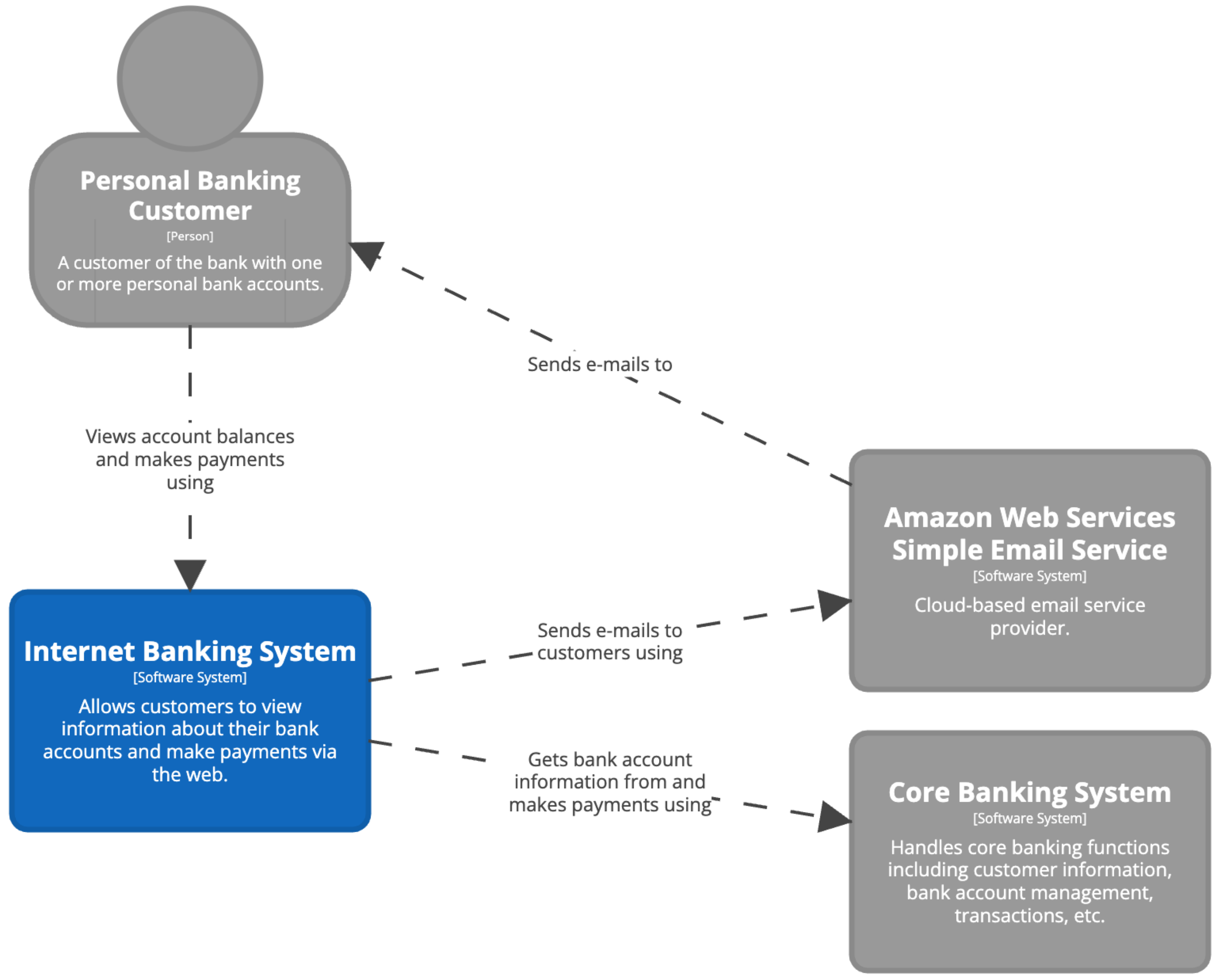
Gets bank account information from and makes payments using

Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.

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

[System Context] Internet Banking System

The system context diagram for a fictional Internet Banking System.



Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.

Views account balances and makes payments using

Sends e-mails to

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

Gets bank account information from and makes payments using

Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.

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

[System Context] Internet Banking System

The system context diagram for a fictional Internet Banking System.

Titles

Short and meaningful, include the **diagram type** and **scope**, numbered if diagram order is important; for example:

System Context View for Internet Banking System

System Context View: Internet Banking System

[System Context View] Internet Banking System

Visual consistency

Try to be consistent with notation
and element positioning across diagrams

Acronyms

Be wary of using acronyms, especially those related to the business/domain that you work in

Boxes

Start with simple boxes containing the element name, type, technology (if appropriate) and a description/responsibilities

Personal Banking Customer

[Person]

A customer of the bank with one or more personal bank accounts.

Internet Banking System

[Software System]

Allows customers to view information about their bank accounts and make payments via the web.

Backend

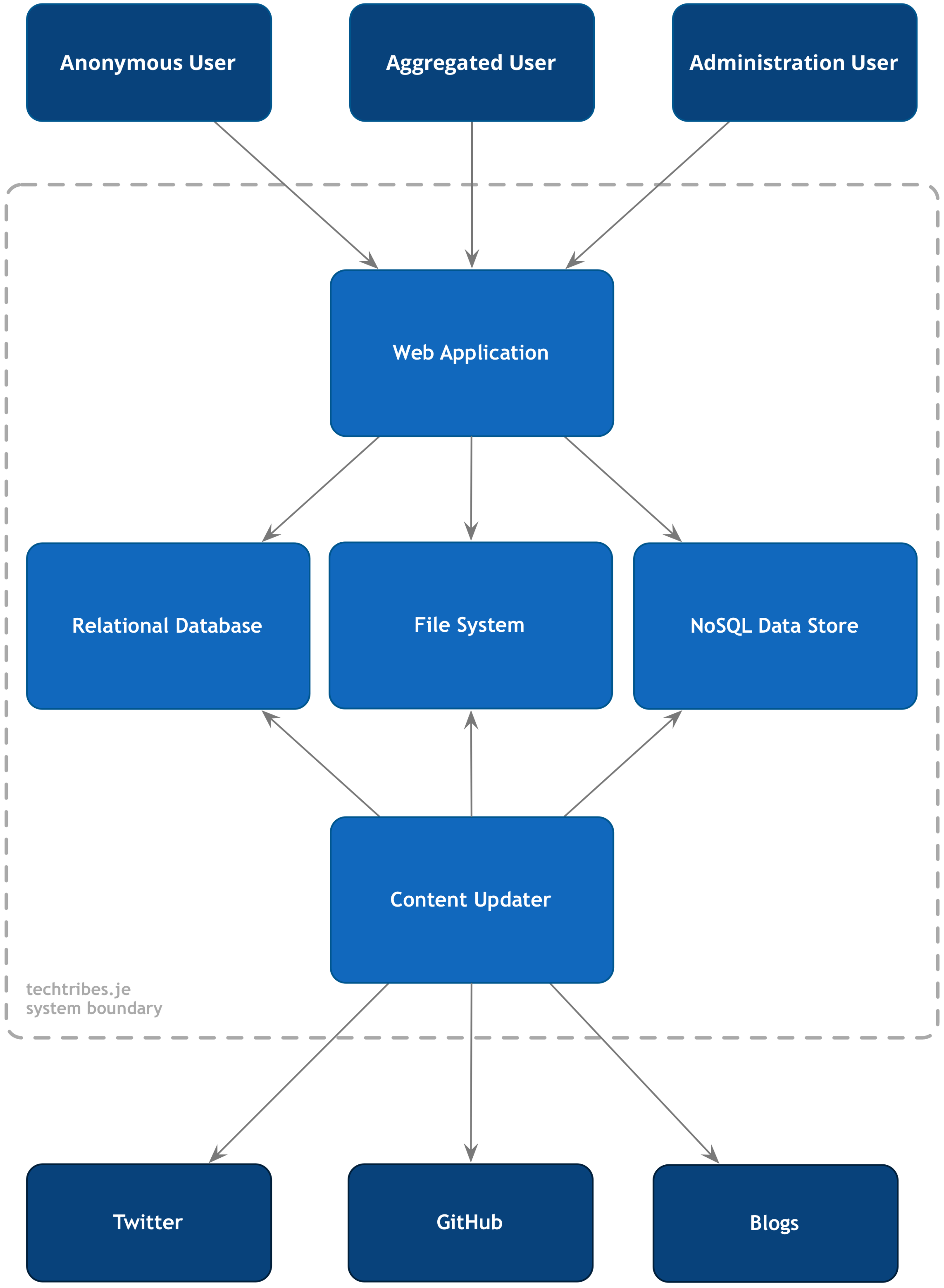
[Container: Java and Spring Boot]

Provides Internet banking functionality via a JSON/HTTPS API.

Core Banking System Adapter

[Component: Spring Bean]

A Java wrapper around the API provided by the Core Banking System.



Anonymous User

Aggregated User

Administration User

Web Application

Relational Database

File System

NoSQL Data Store

Content Updater

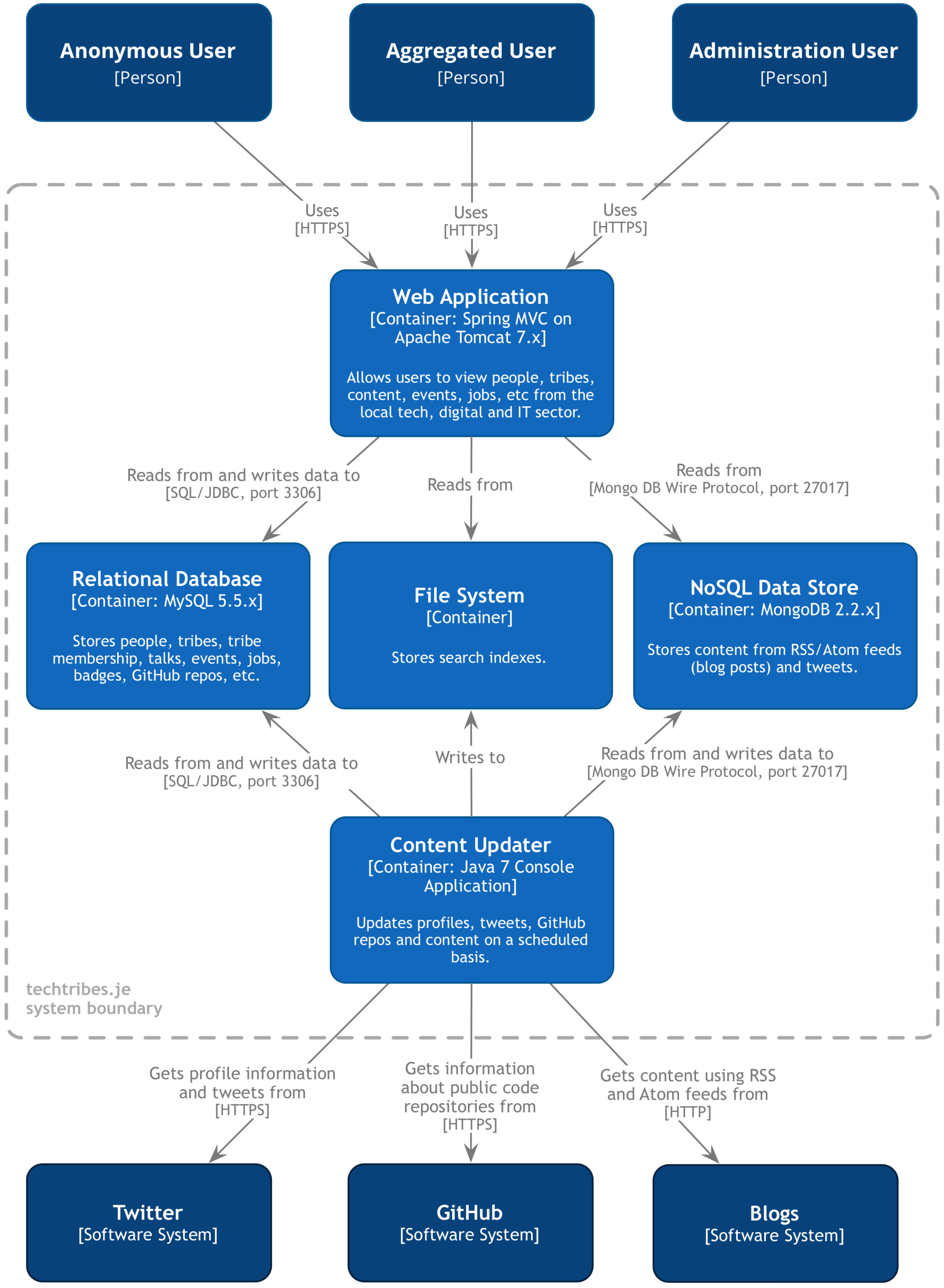
techtribes.je
system boundary

Twitter

GitHub

Blogs

[Containers] techtribes.je



Anonymous User
[Person]

Aggregated User
[Person]

Administration User
[Person]

Uses
[HTTPS]

Uses
[HTTPS]

Uses
[HTTPS]

Web Application
[Container: Spring MVC on Apache Tomcat 7.x]

Allows users to view people, tribes, content, events, jobs, etc from the local tech, digital and IT sector.

Reads from and writes data to
[SQL/JDBC, port 3306]

Reads from

Reads from
[Mongo DB Wire Protocol, port 27017]

Relational Database
[Container: MySQL 5.5.x]

Stores people, tribes, tribe membership, talks, events, jobs, badges, GitHub repos, etc.

File System
[Container]

Stores search indexes.

NoSQL Data Store
[Container: MongoDB 2.2.x]

Stores content from RSS/Atom feeds (blog posts) and tweets.

Reads from and writes data to
[SQL/JDBC, port 3306]

Writes to

Reads from and writes data to
[Mongo DB Wire Protocol, port 27017]

Content Updater
[Container: Java 7 Console Application]

Updates profiles, tweets, GitHub repos and content on a scheduled basis.

techtribes.je
system boundary

Gets profile information and tweets from
[HTTPS]

Gets information about public code repositories from
[HTTPS]

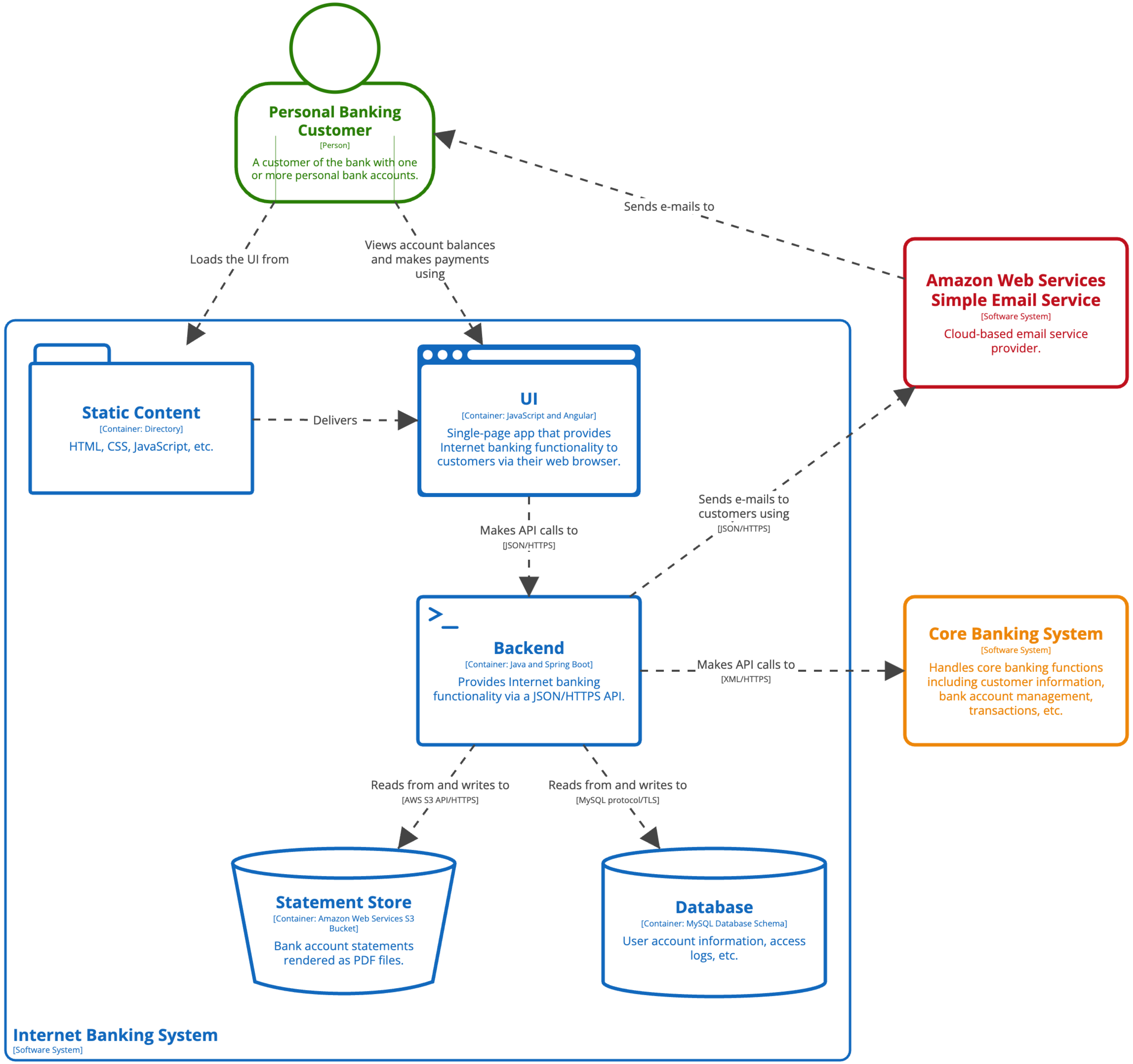
Gets content using RSS and Atom feeds from
[HTTP]

Twitter
[Software System]

GitHub
[Software System]

Blogs
[Software System]

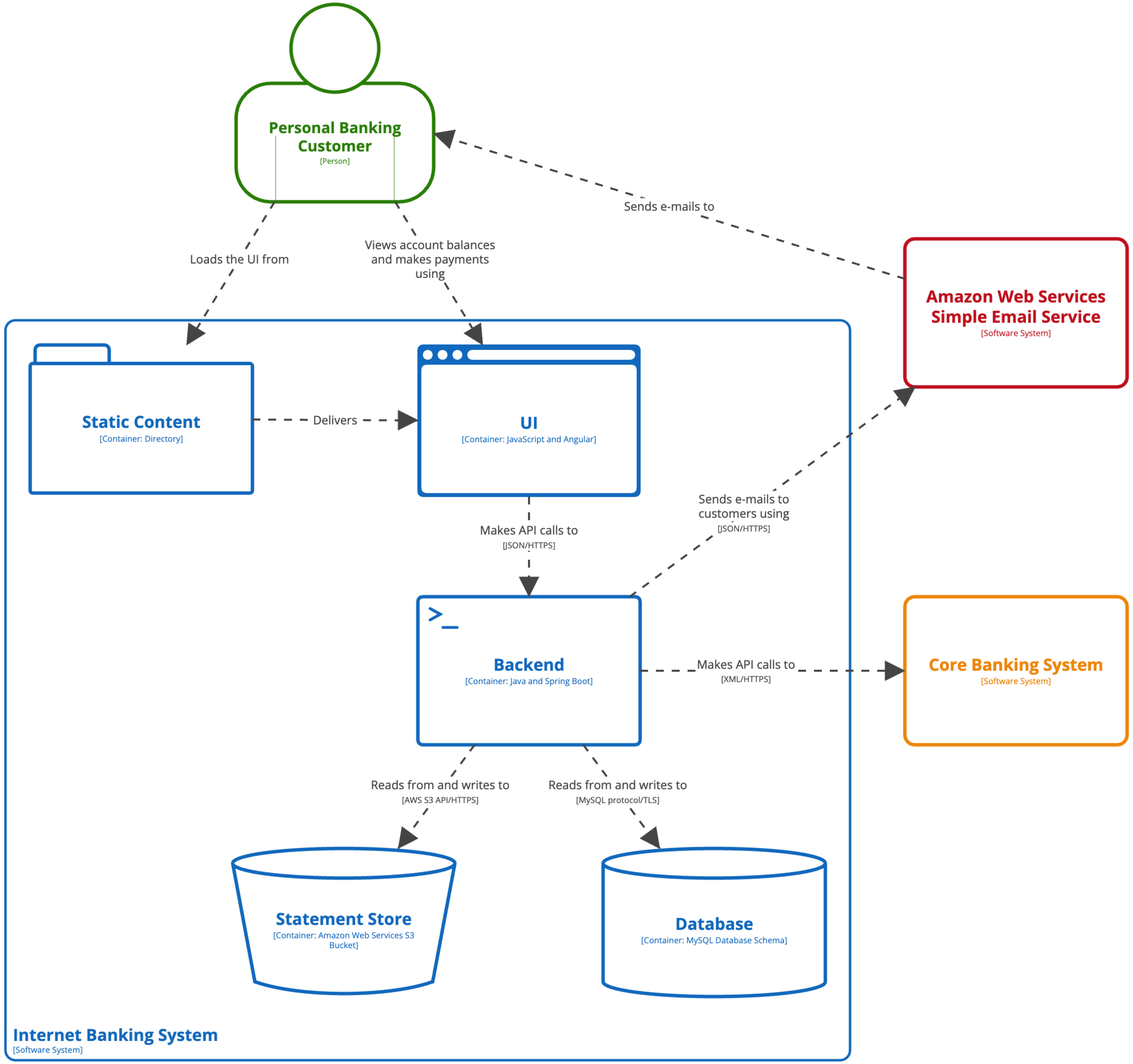
[Containers] techtribes.je



Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.
Sends e-mails to
Loads the UI from
Views account balances and makes payments using
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Static Content
[Container: Directory]
HTML, CSS, JavaScript, etc.
Delivers
UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Sends e-mails to customers using
[JSON/HTTPS]
Makes API calls to
[JSON/HTTPS]
>_
Backend
[Container: Java and Spring Boot]
Provides Internet banking functionality via a JSON/HTTPS API.
Makes API calls to
[XML/HTTPS]
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
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.
Internet Banking System
[Software System]

Container View: Internet Banking System

The container diagram for the Internet Banking System

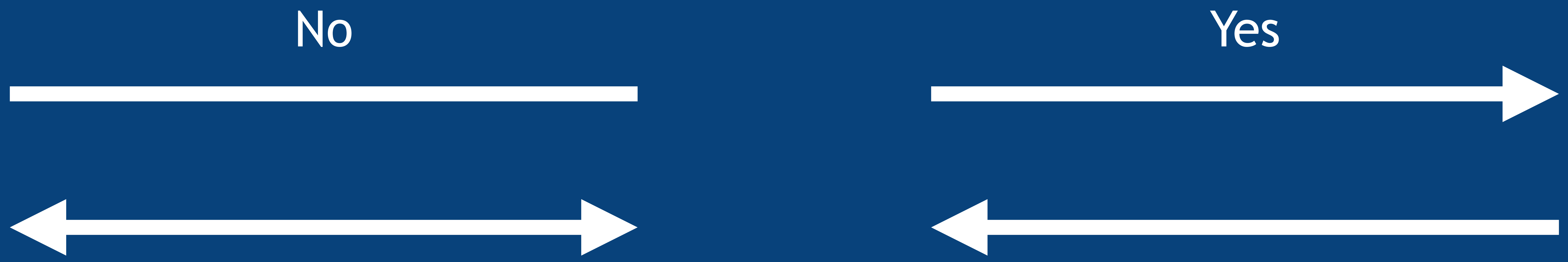


Container View: Internet Banking System

The container diagram for the Internet Banking System

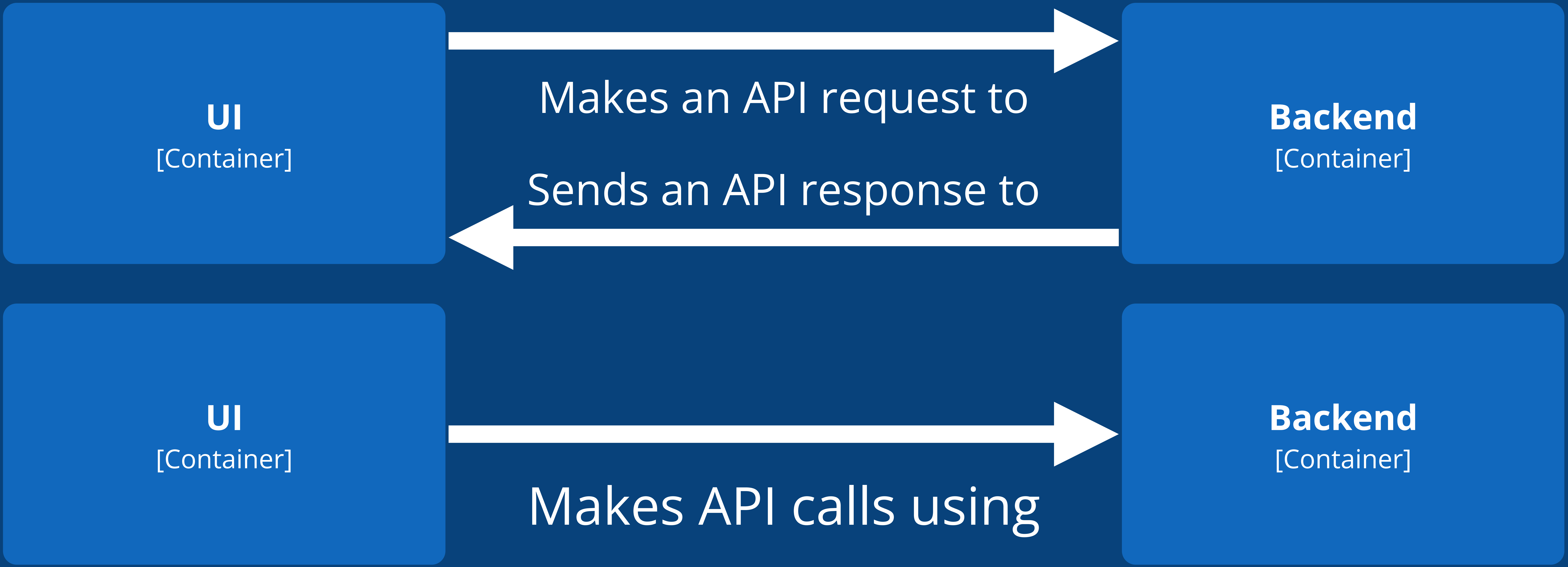
Lines

Favour uni-directional lines showing the most important dependencies or data flow, with an annotation to be explicit about the purpose of the line and direction



No

Yes



UI
[Container]

Makes an API request to

Sends an API response to

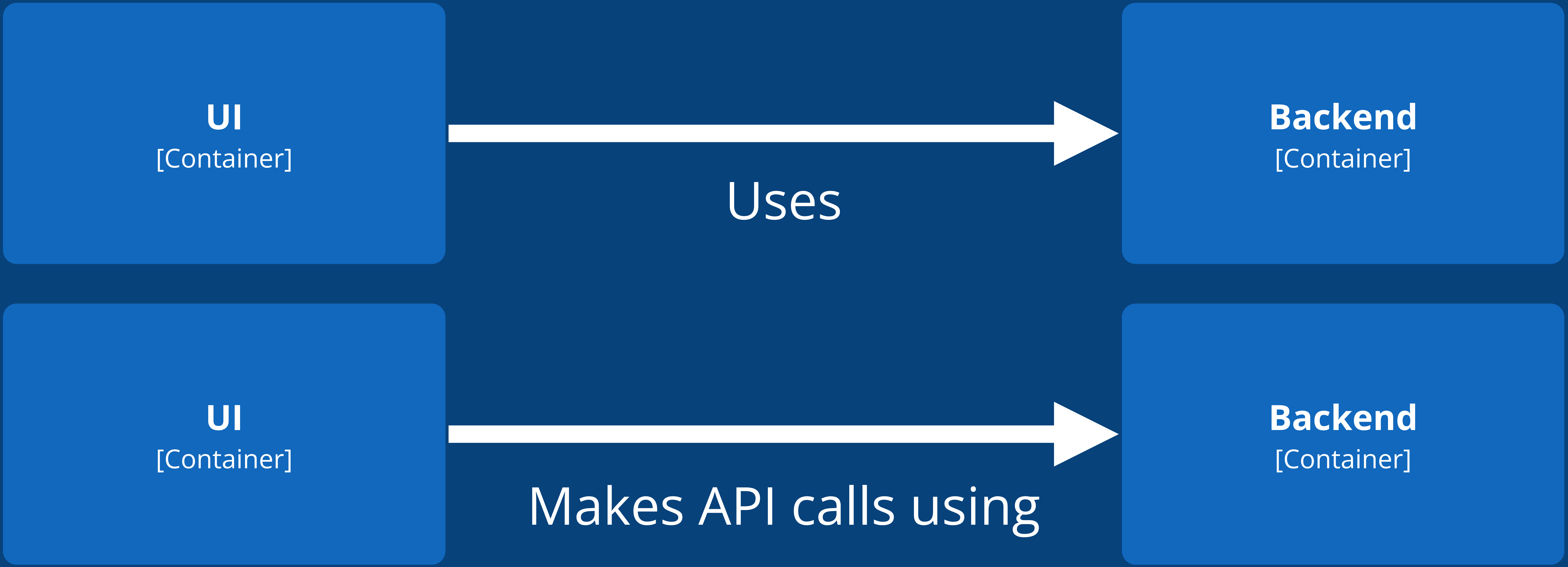
Backend
[Container]

UI
[Container]

Makes API calls using

Backend
[Container]

Summarise the intent of the relationship



UI
[Container]

Uses

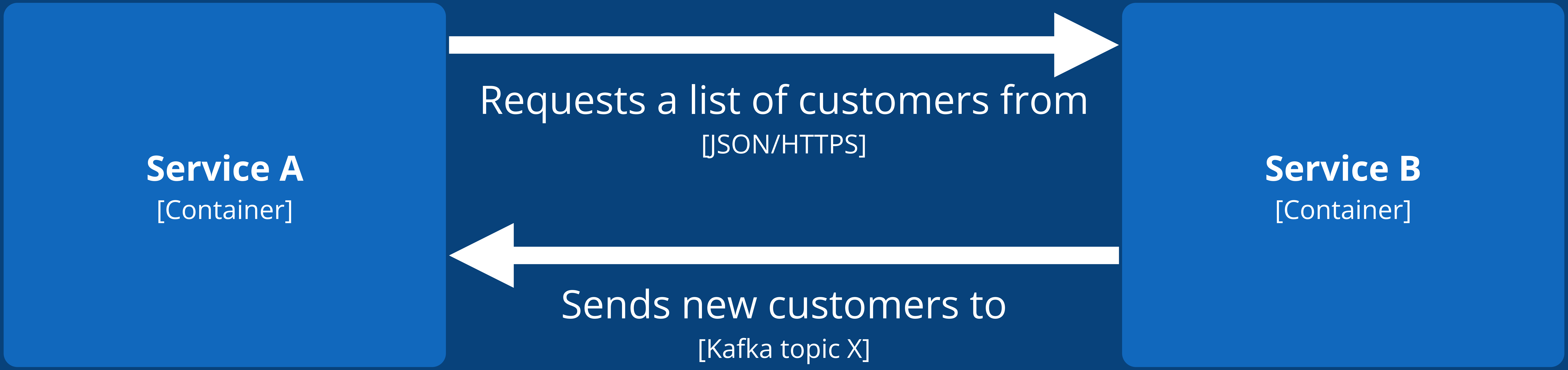
Backend
[Container]

UI
[Container]

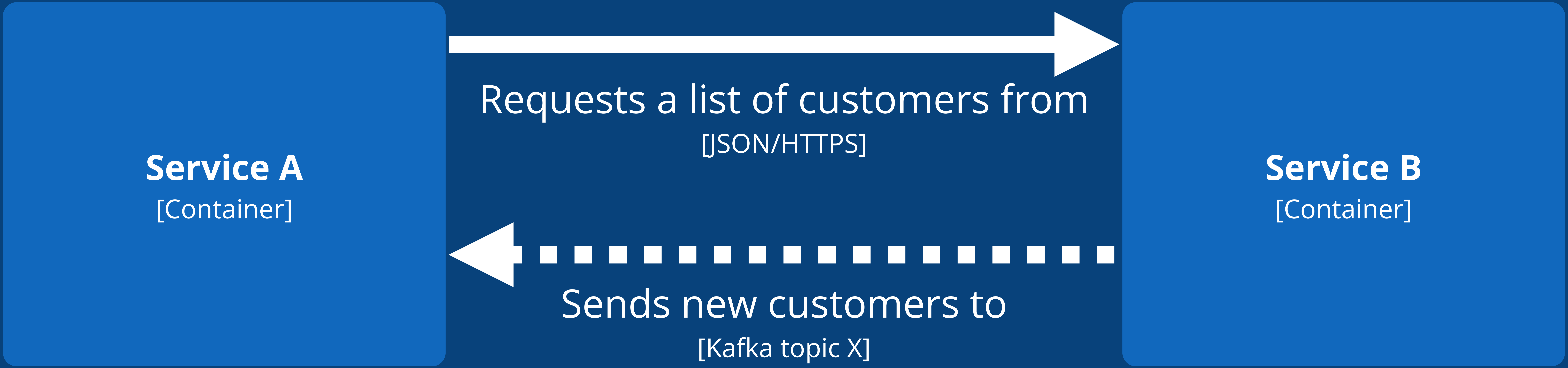
Makes API calls using

Backend
[Container]

Summarise, yet be specific



Show both directions when the intents are different



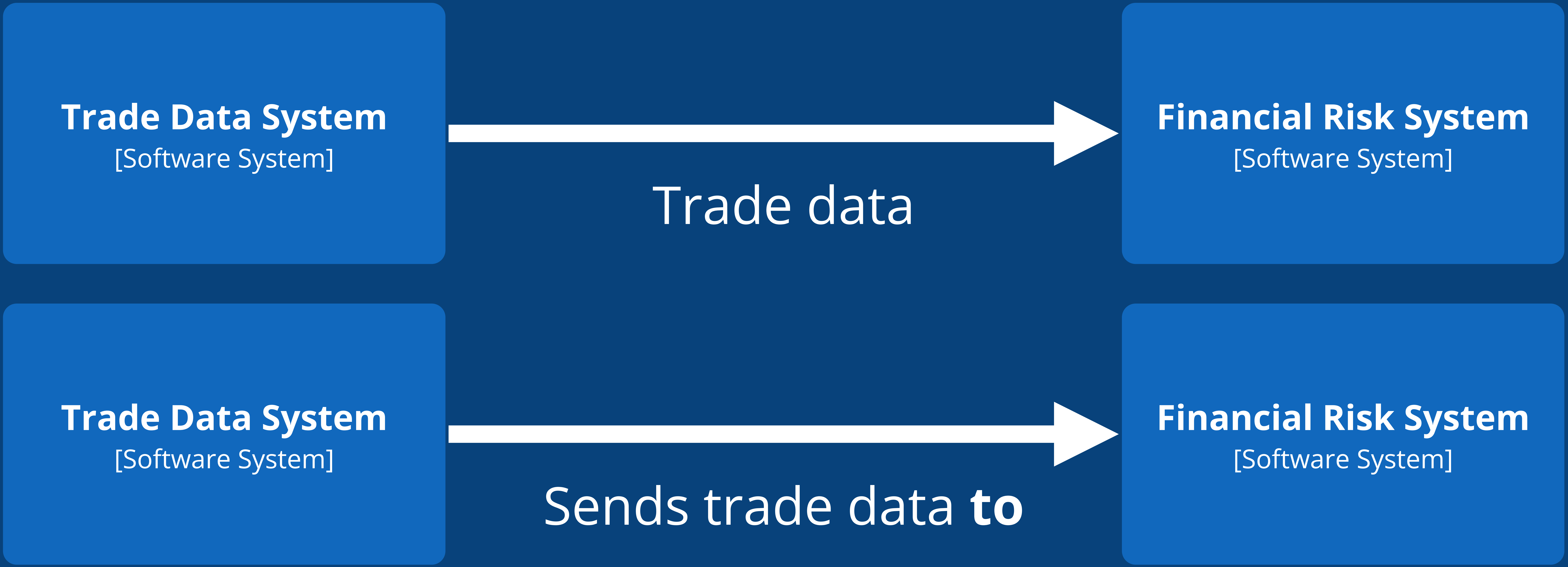
Service A
[Container]

Requests a list of customers from
[JSON/HTTPS]

Sends new customers to
[Kafka topic X]

Service B
[Container]

Show both directions when the intents are different



Trade Data System
[Software System]

Trade data

Financial Risk System
[Software System]

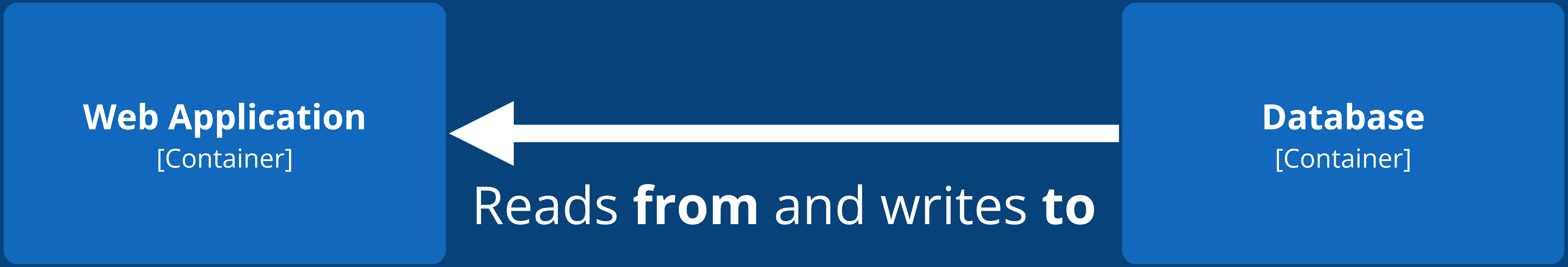
Trade Data System
[Software System]

Sends trade data **to**

Financial Risk System
[Software System]

Add more words to make the intent explicit

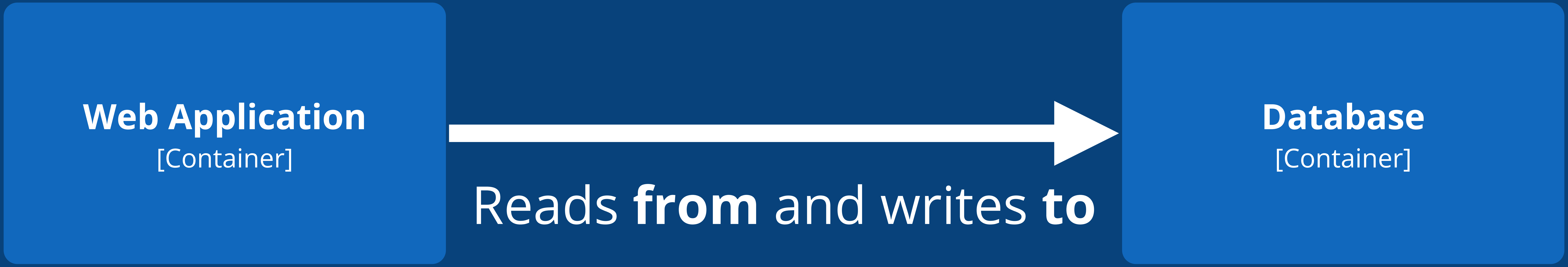
If in doubt, read the relationship



Web Application
[Container]

Reads **from** and writes **to**

Database
[Container]



Web Application
[Container]

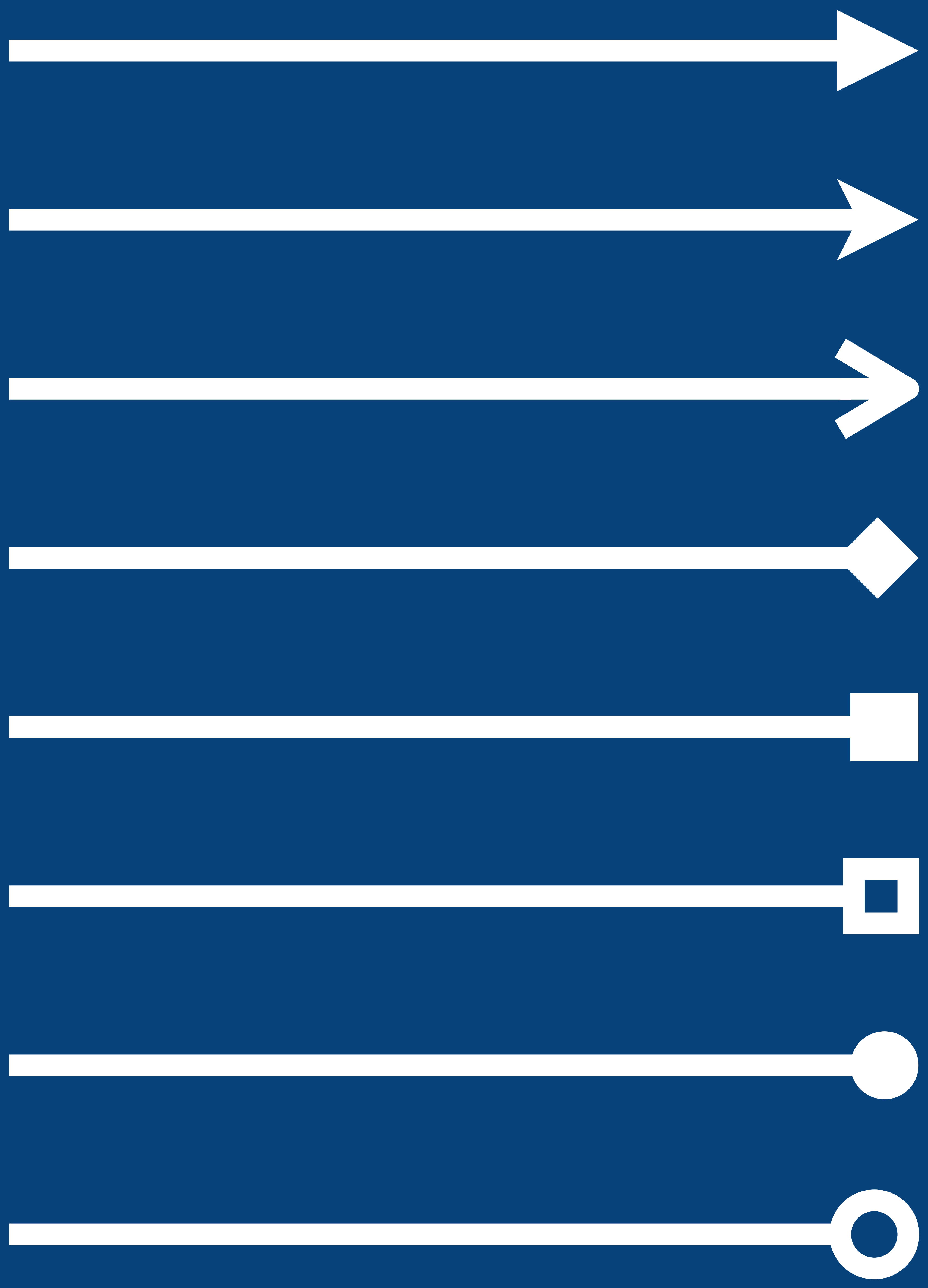
Reads **from** and writes **to**

Database
[Container]

Key/legend

Explain shapes, line styles, colours, borders, acronyms, etc

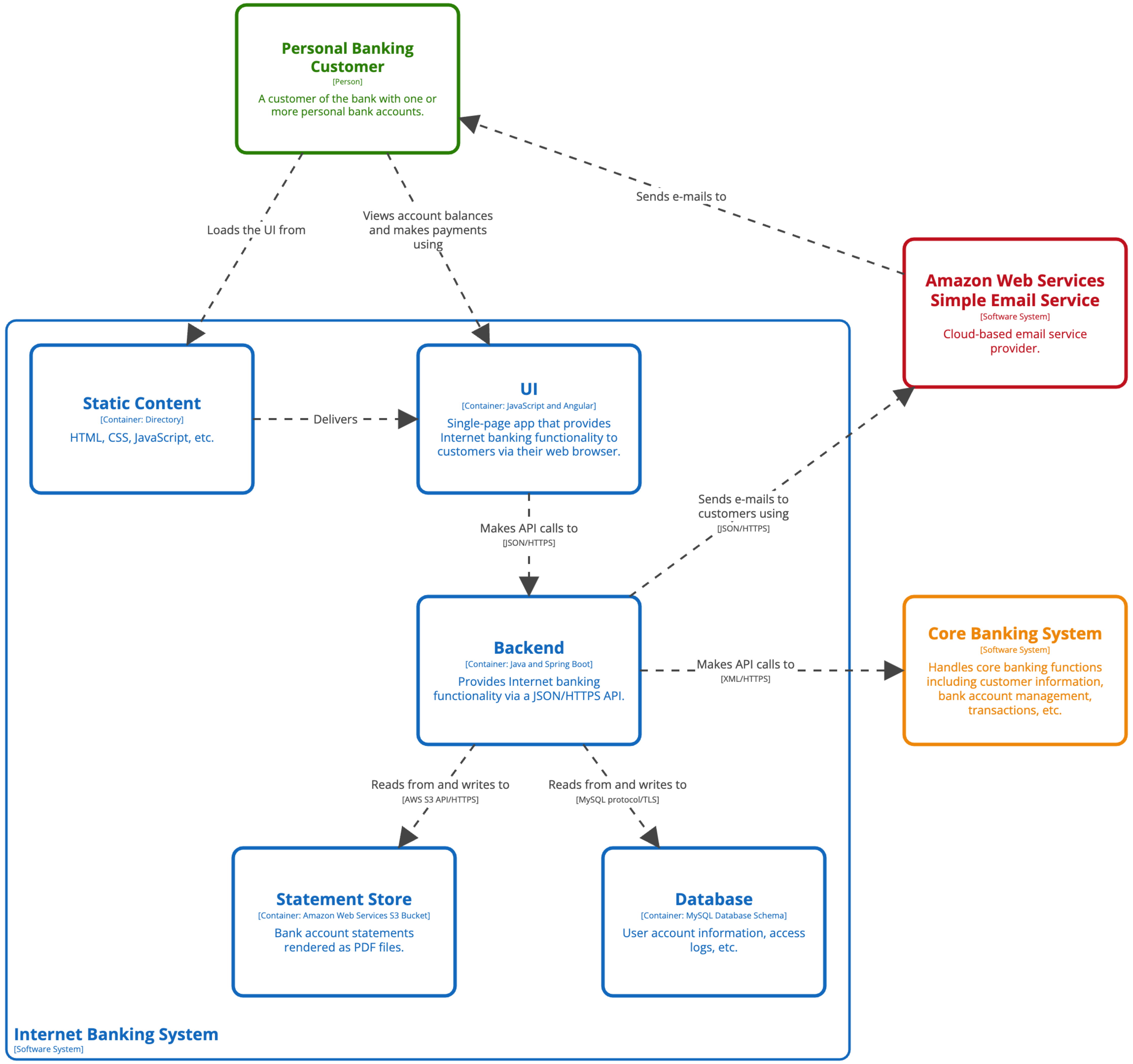
… even if your notation seems obvious!



Arrowheads

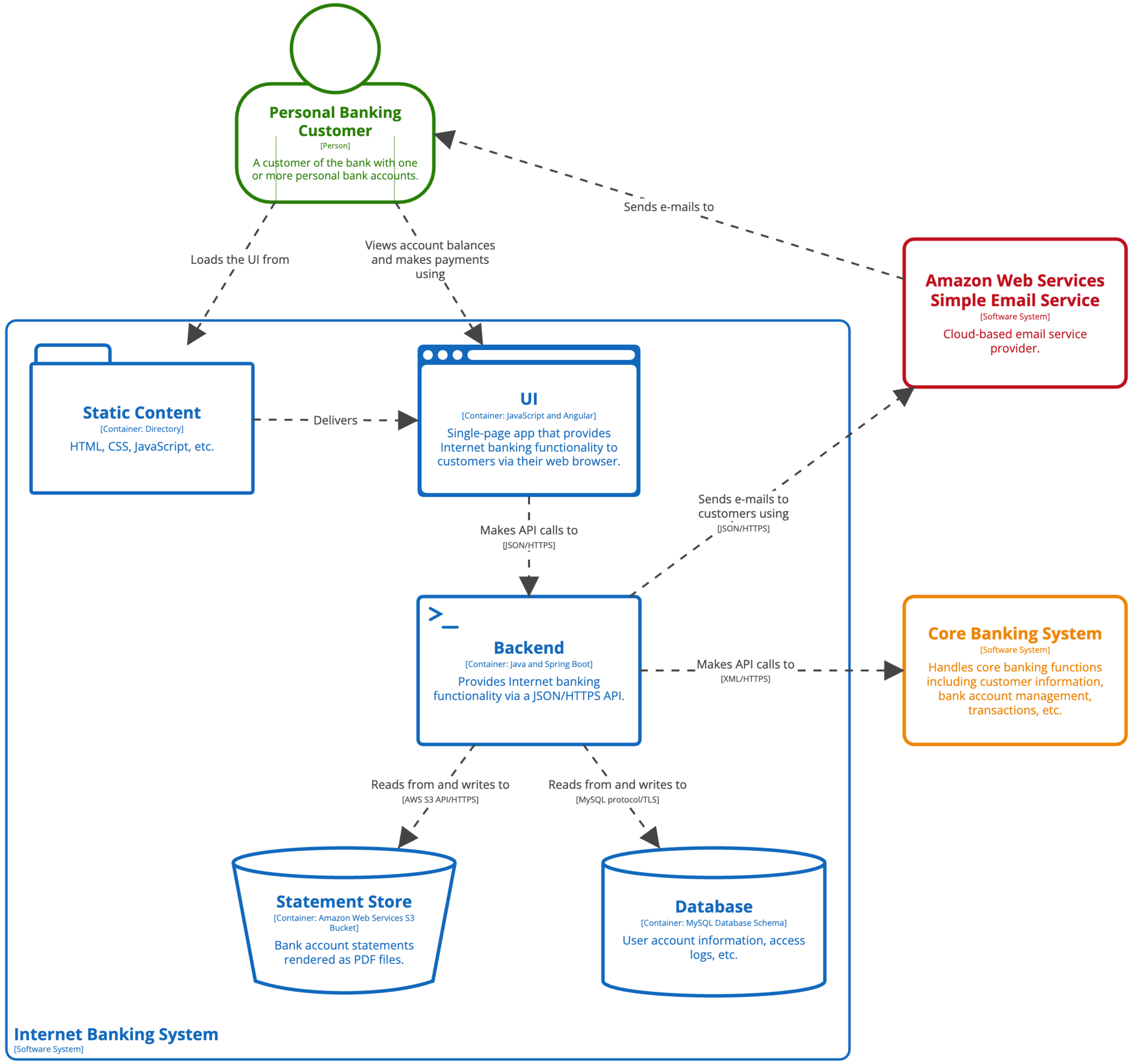
Be careful, using different arrowheads is very subtle; readers may miss them

Use shape, colour and size to **complement** a diagram that already makes sense


Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.
Loads the UI from
Views account balances and makes payments using
Sends e-mails to
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Static Content
[Container: Directory]
HTML, CSS, JavaScript, etc.
Delivers
UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Makes API calls to
[JSON/HTTPS]
Sends e-mails to customers using
[JSON/HTTPS]
Backend
[Container: Java and Spring Boot]
Provides Internet banking functionality via a JSON/HTTPS API.
Makes API calls to
[XML/HTTPS]
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
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.
Internet Banking System
[Software System]

Container View: Internet Banking System

The container diagram for the Internet Banking System


Personal Banking Customer
[Person]
A customer of the bank with one or more personal bank accounts.
Loads the UI from
Views account balances and makes payments using
Sends e-mails to
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Static Content
[Container: Directory]
HTML, CSS, JavaScript, etc.
Delivers
UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Makes API calls to
[JSON/HTTPS]
Sends e-mails to customers using
[JSON/HTTPS]
>_
Backend
[Container: Java and Spring Boot]
Provides Internet banking functionality via a JSON/HTTPS API.
Makes API calls to
[XML/HTTPS]
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
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.
Internet Banking System
[Software System]

Container View: Internet Banking System

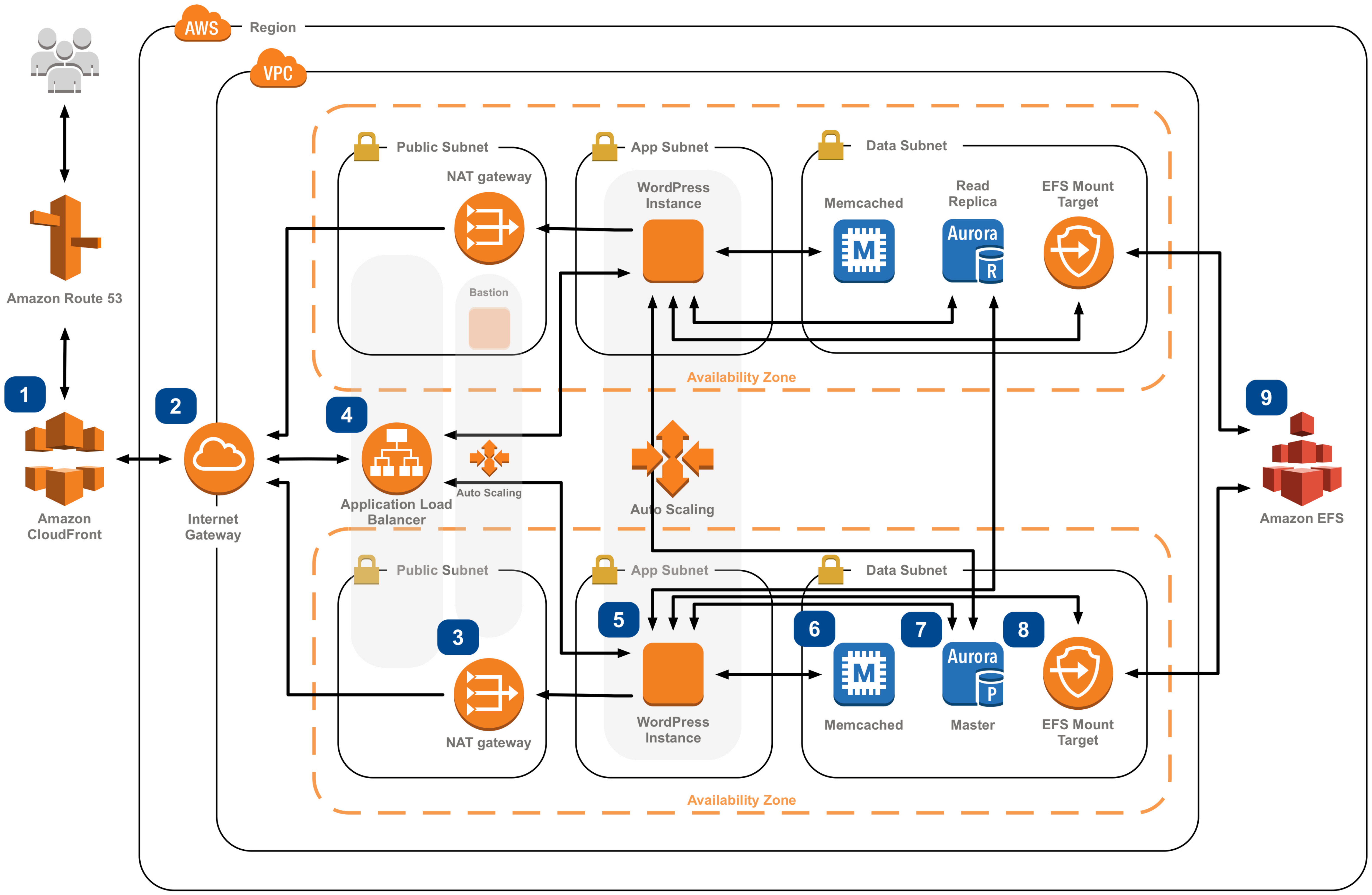
The container diagram for the Internet Banking System

Be careful with **icons**

WordPress Hosting

How to run WordPress on AWS

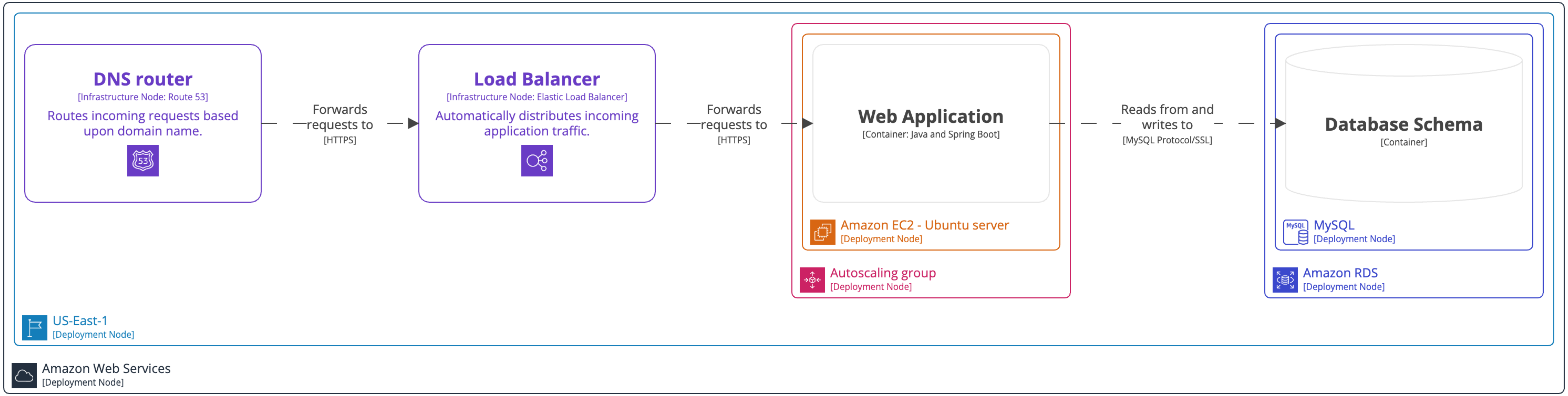
WordPress is one of the world's most popular web publishing platforms, being used to publish 27% of all websites, from personal blogs to some of the biggest news sites. This reference architecture simplifies the complexity of deploying a scalable and highly available WordPress site on AWS.

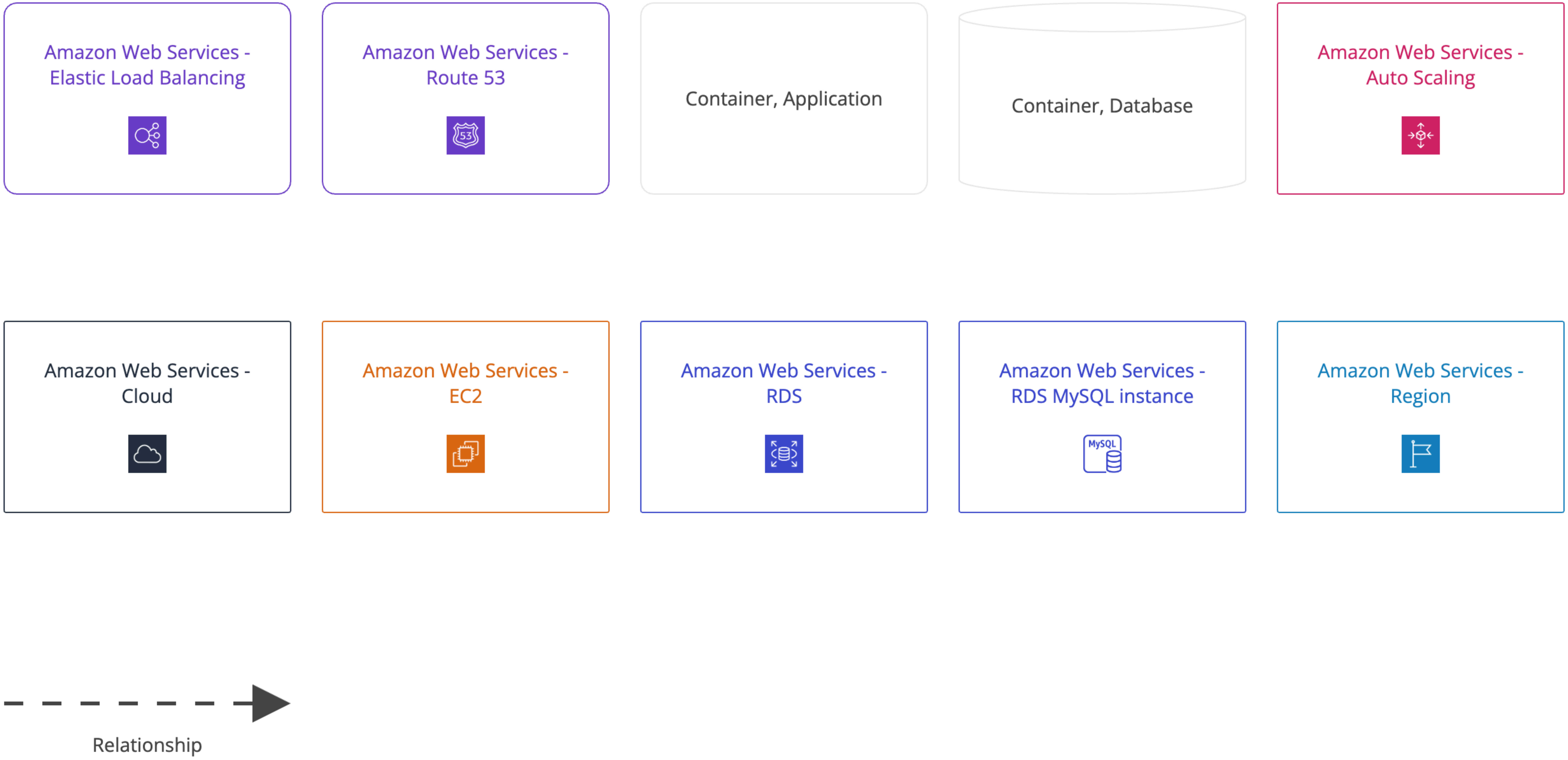


1. Static and dynamic content is delivered by **Amazon CloudFront**.
2. An **Internet gateway** allows communication between instances in your VPC and the Internet.
3. **NAT gateways** in each public subnet enable Amazon EC2 instances in private subnets (App & Data) to access the Internet.
4. Use an **Application Load Balancer** to distribute web traffic across an Auto Scaling Group of Amazon EC2 instances in multiple AZs.
5. Run your WordPress site using an **Auto Scaling group** of **Amazon EC2 instances**. Install the latest versions of WordPress, Apache web server, PHP 7, and OPcache and build an Amazon Machine Image that will be used by the Auto Scaling group launch configuration to launch new instances in the Auto Scaling group.
6. If database access patterns are read-heavy, consider using a WordPress plugin that takes advantage of a caching layer like **Amazon ElastiCache** (**Memcached**) in front of the database layer to cache frequently accessed data.
7. Simplify your database administration by running your database layer in **Amazon RDS** using either Aurora or MySQL.
8. Amazon EC2 instances access shared WordPress data in an Amazon EFS file system using **Mount Targets** in each AZ in your VPC.
9. Use **Amazon EFS**, a simple, highly available, and scalable network file system so WordPress instances have access to your shared, unstructured WordPress data, like php files, config, themes, plugins, etc.

amazon web services

AWS Reference Architectures


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]



Amazon Web Services - Elastic Load Balancing

Amazon Web Services - Route 53

Container, Application

Container, Database

Amazon Web Services - Auto Scaling

Amazon Web Services - Cloud

Amazon Web Services - EC2

Amazon Web Services - RDS

Amazon Web Services - RDS MySQL instance

Amazon Web Services - Region

Relationship

Increase the **readability** of software architecture diagrams, so they can **stand alone**

Any narrative should **complement** the diagram rather than explain it

Abstractions first, notation second

Ensure that your team has a ubiquitous language to describe software architecture

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

Software architecture diagram review checklist

General

Does the diagram have a title?	Yes	No
Do you understand what the diagram type is?	Yes	No
Do you understand what the diagram scope is?	Yes	No
Does the diagram have a key/legend?	Yes	No

Elements

Does every element have a name?	Yes	No
Do you understand the type of every element? (i.e. the level of abstraction; e.g. software system, container, etc)	Yes	No
Do you understand what every element does?	Yes	No
Where applicable, do you understand the technology choices associated with every element?	Yes	No

Draw **System Context** and **Container** diagrams to describe a solution for the ”Financial Risk System”

⏱️ = finish diagrams by 14:00

c4model.com/frs

Designing software is where
the complexity should be,
not communicating it!

Similar levels of abstraction provide a way to easily **compare** solutions

The diagrams should spark **meaningful questions**

No

“What does that arrow mean?”

“Why are some boxes red?”

“Is that a Java application?”

“Is that a monolithic application, or a collection of microservices?”

“How do the users get their reports?”

Yes

“What protocol are your two Java applications using to communicate with each other?”

“Why do you have two separate C# applications instead of one?”

“Why are you using MongoDB?”

“Why are you using MySQL when our standard is Oracle?”

“Should we really build new applications with .NET Framework rather than .NET Core?”

Richer diagrams lead to
richer **design discussions**

Richer diagrams lead to
better communication,
making it easier to scale teams

Runtime/behavioural diagrams

Static structure diagrams are very useful, but they don’t tell the whole story

UI
[Container: JavaScript and Angular]

Sign In API
[Component: Spring MVC]

Security Component
[Component: Spring Bean]

Database
[Container: MySQL Database Schema]

1: Submits credentials to
[JSON/HTTPS]

2: Validates credentials using

3: select * from users where username = ?
[MySQL protocol/TLS]

4: Returns user data to
[MySQL protocol/TLS]

5: Issues a session token if authentication succeeds

6: Sends back an authentication token to
[JSON/HTTPS]

UI
[Container: JavaScript and Angular]

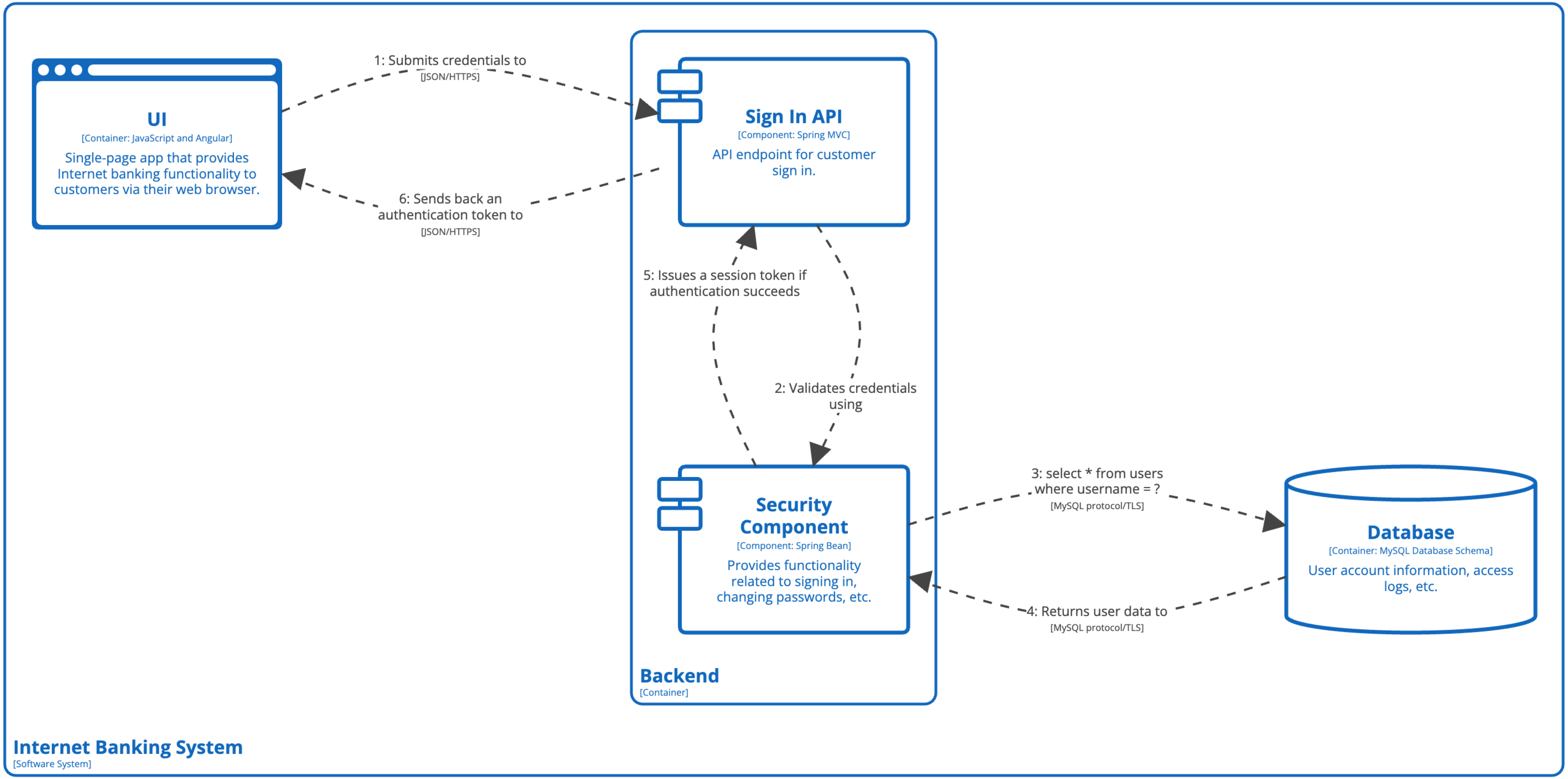
Sign In API
[Component: Spring MVC]

Security Component
[Component: Spring Bean]

Database
[Container: MySQL Database Schema]

Dynamic View: Internet Banking System - Backend

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


UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
1: Submits credentials to
[JSON/HTTPS]
6: Sends back an authentication token to
[JSON/HTTPS]
Sign In API
[Component: Spring MVC]
API endpoint for customer sign in.
5: Issues a session token if authentication succeeds
2: Validates credentials using
Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.
3: select * from users where username = ?
[MySQL protocol/TLS]
4: Returns user data to
[MySQL protocol/TLS]
Database
[Container: MySQL Database Schema]
User account information, access logs, etc.
Backend
[Container]
Internet Banking System
[Software System]

Dynamic View: Internet Banking System - Backend

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

Use dynamic diagrams to describe **patterns** or **complex interactions**

Deployment diagrams

Deployment is about the mapping of containers to infrastructure

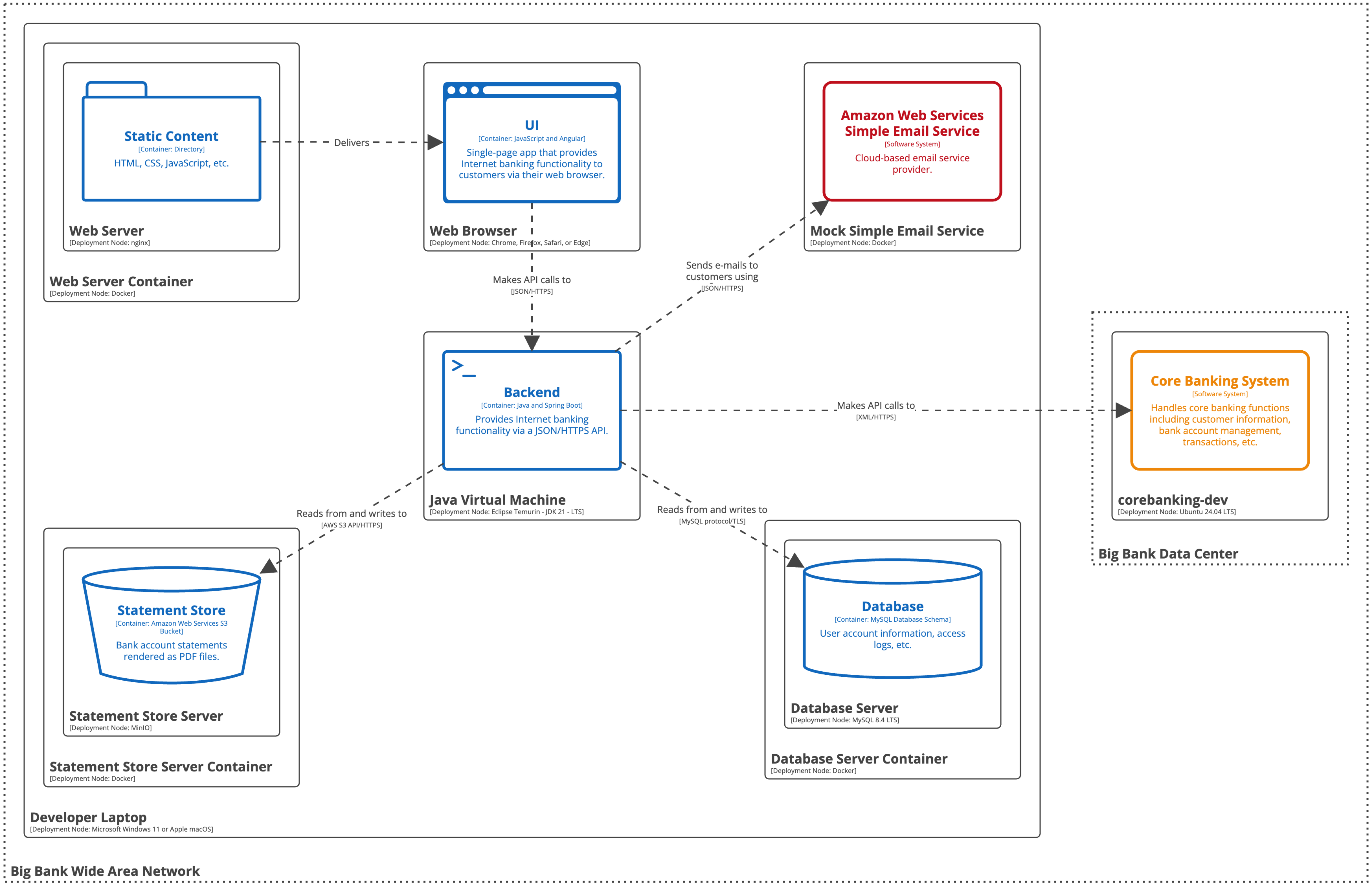
Deployment Node

Physical infrastructure (a physical server or device), virtualised infrastructure (IaaS, PaaS, a virtual machine), containerised infrastructure (a Docker container), database server, Java EE web/application server, Microsoft IIS, etc

A deployment node can contain other **deployment nodes** or software system/container **instances**

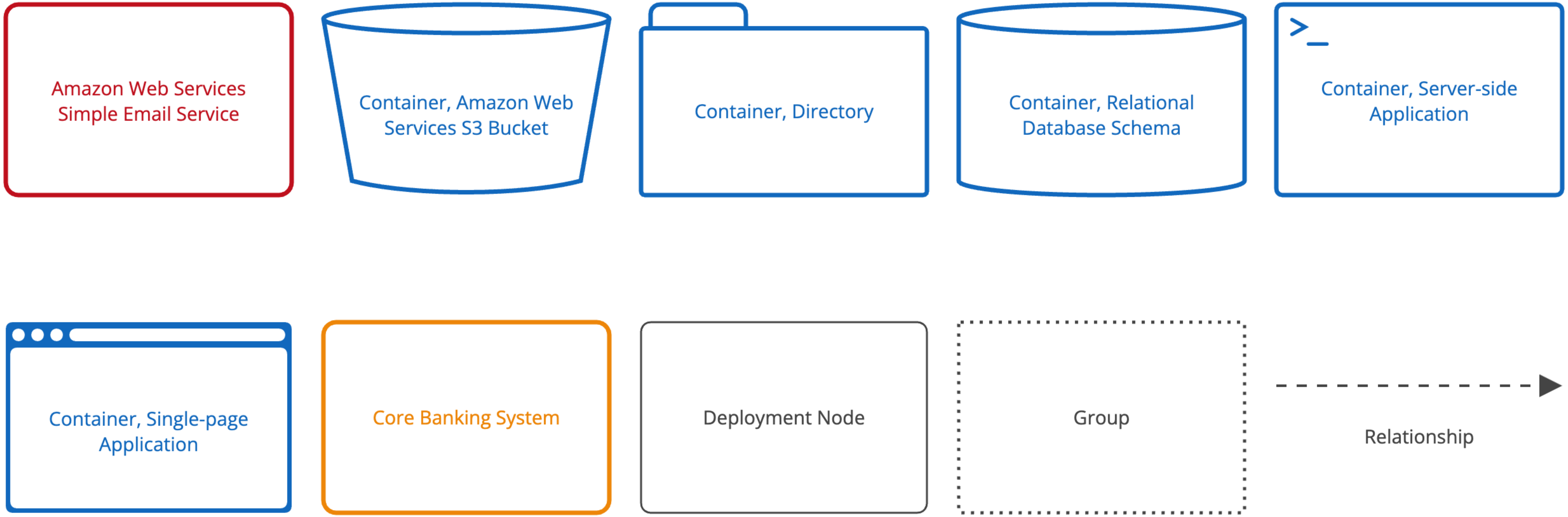
Infrastructure Node

Routers, firewalls, load balancers, DNS providers, edge caches, etc


Static Content
[Container: Directory]
HTML, CSS, JavaScript, etc.
Web Server
[Deployment Node: nginx]
Web Server Container
[Deployment Node: Docker]
Delivers
UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Web Browser
[Deployment Node: Chrome, Firefox, Safari, or Edge]
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Mock Simple Email Service
[Deployment Node: Docker]
Makes API calls to
[JSON/HTTPS]
Sends e-mails to customers using
[JSON/HTTPS]
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]
Makes API calls to
[XML/HTTPS]
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
corebanking-dev
[Deployment Node: Ubuntu 24.04 LTS]
Big Bank Data Center
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.
Statement Store Server
[Deployment Node: MinIO]
Statement Store Server Container
[Deployment Node: Docker]
Database
[Container: MySQL Database Schema]
User account information, access logs, etc.
Database Server
[Deployment Node: MySQL 8.4 LTS]
Database Server Container
[Deployment Node: Docker]
Developer Laptop
[Deployment Node: Microsoft Windows 11 or Apple macOS]
Big Bank Wide Area Network

Deployment View: Internet Banking System - Development

An example development deployment scenario for the Internet Banking System



Amazon Web Services Simple Email Service

Container, Amazon Web Services S3 Bucket

Container, Directory

Container, Relational Database Schema

Container, Server-side Application

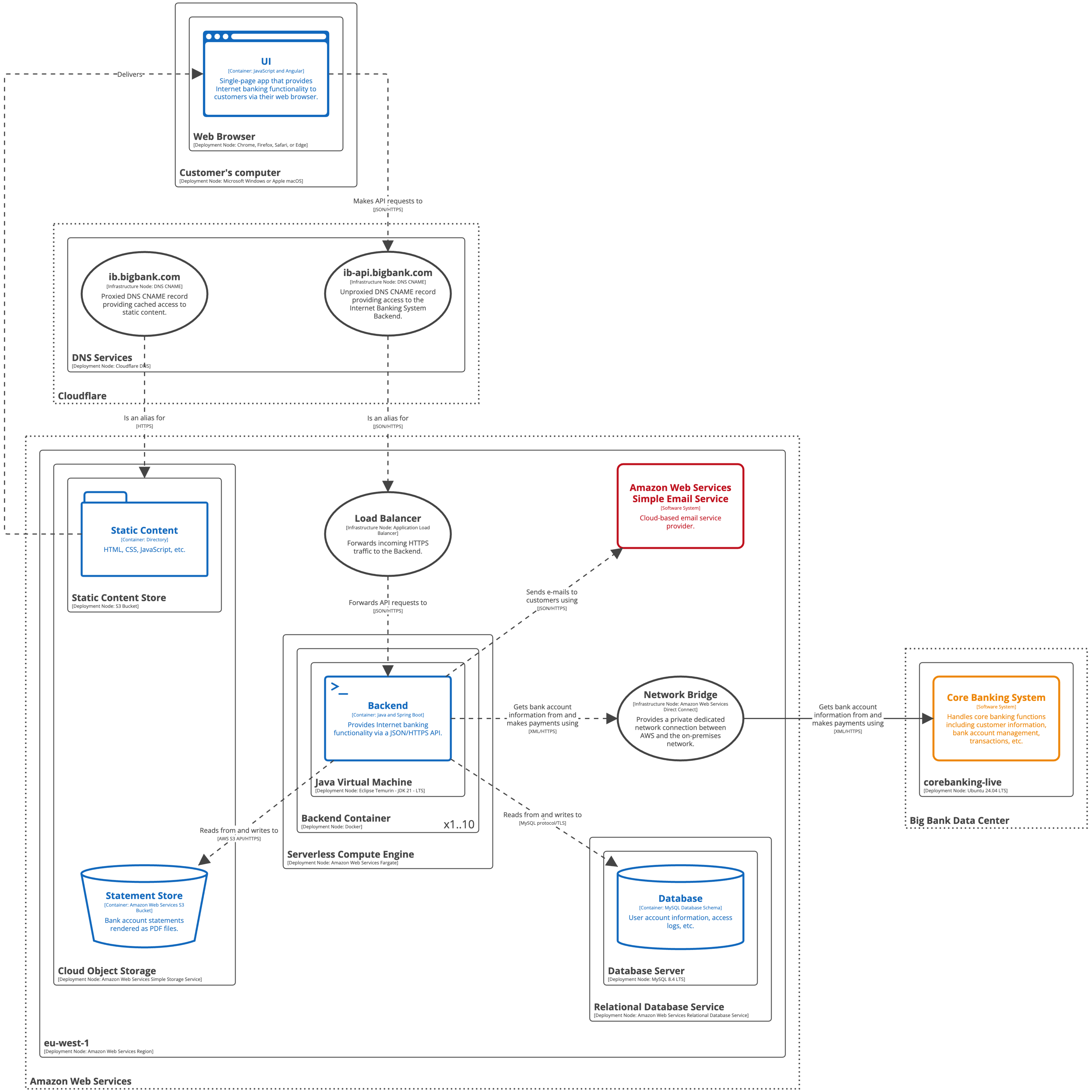
Container, Single-page Application

Core Banking System

Deployment Node

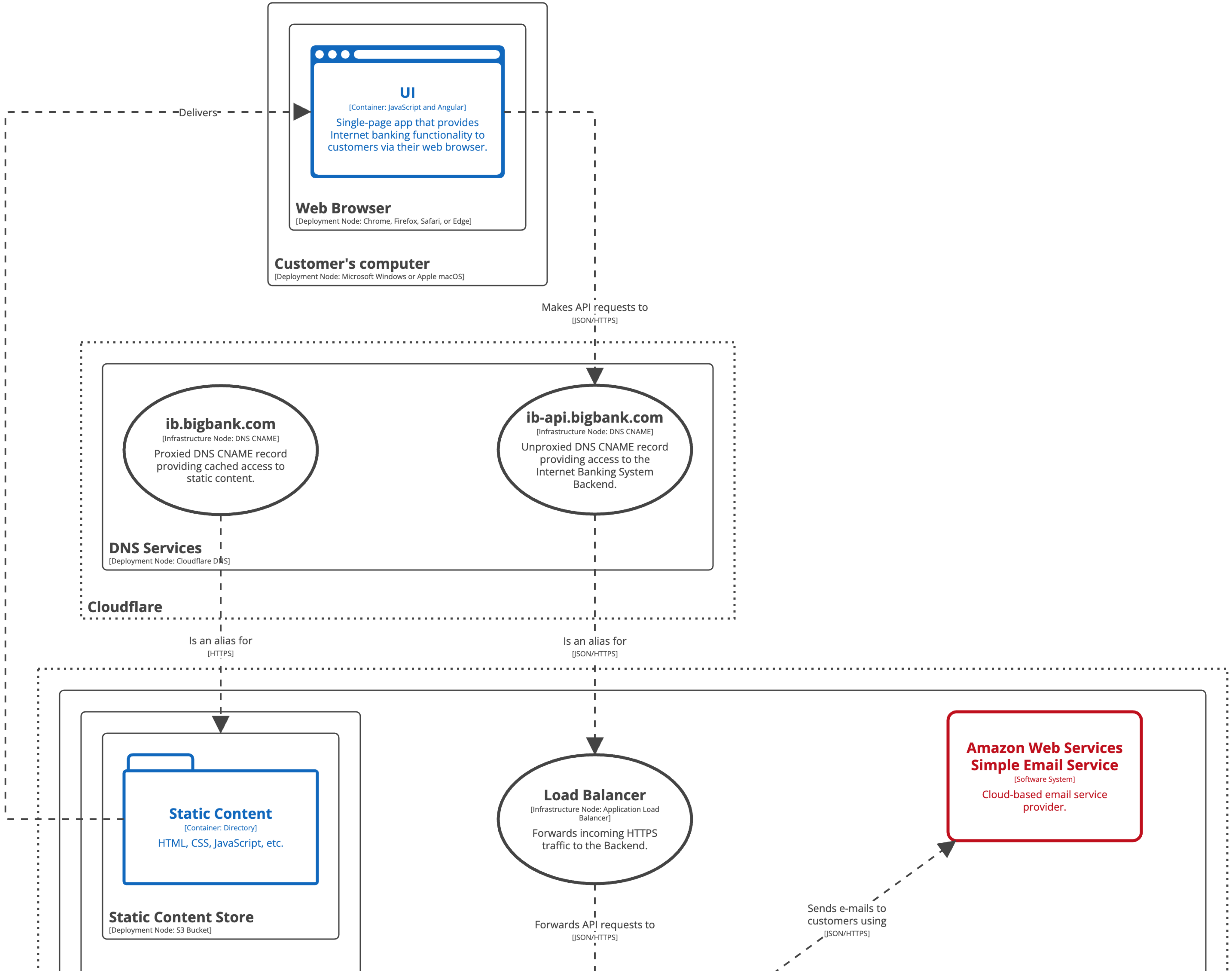
Group

Relationship


UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Web Browser
[Deployment Node: Chrome, Firefox, Safari, or Edge]
Customer's computer
[Deployment Node: Microsoft Windows or Apple macOS]
Delivers
Makes API requests to
[JSON/HTTPS]
ib.bigbank.com
[Infrastructure Node: DNS CNAME]
Proxied DNS CNAME record providing cached access to static content.
ib-api.bigbank.com
[Infrastructure Node: DNS CNAME]
Unproxied DNS CNAME record providing access to the Internet Banking System Backend.
DNS Services
[Deployment Node: Cloudflare DNS]
Cloudflare
Is an alias for
[HTTPS]
Is an alias for
[JSON/HTTPS]
Static Content
[Container: Directory]
HTML, CSS, JavaScript, etc.
Static Content Store
[Deployment Node: S3 Bucket]
Load Balancer
[Infrastructure Node: Application Load Balancer]
Forwards incoming HTTPS traffic to the Backend.
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Forwards API requests to
[JSON/HTTPS]
Sends e-mails to customers using
[JSON/HTTPS]
>_
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.
Gets bank account information from and makes payments using
[XML/HTTPS]
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
corebanking-live
[Deployment Node: Ubuntu 24.04 LTS]
Big Bank Data Center
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]
eu-west-1
[Deployment Node: Amazon Web Services Region]
Amazon Web Services

Deployment View: Internet Banking System - Live

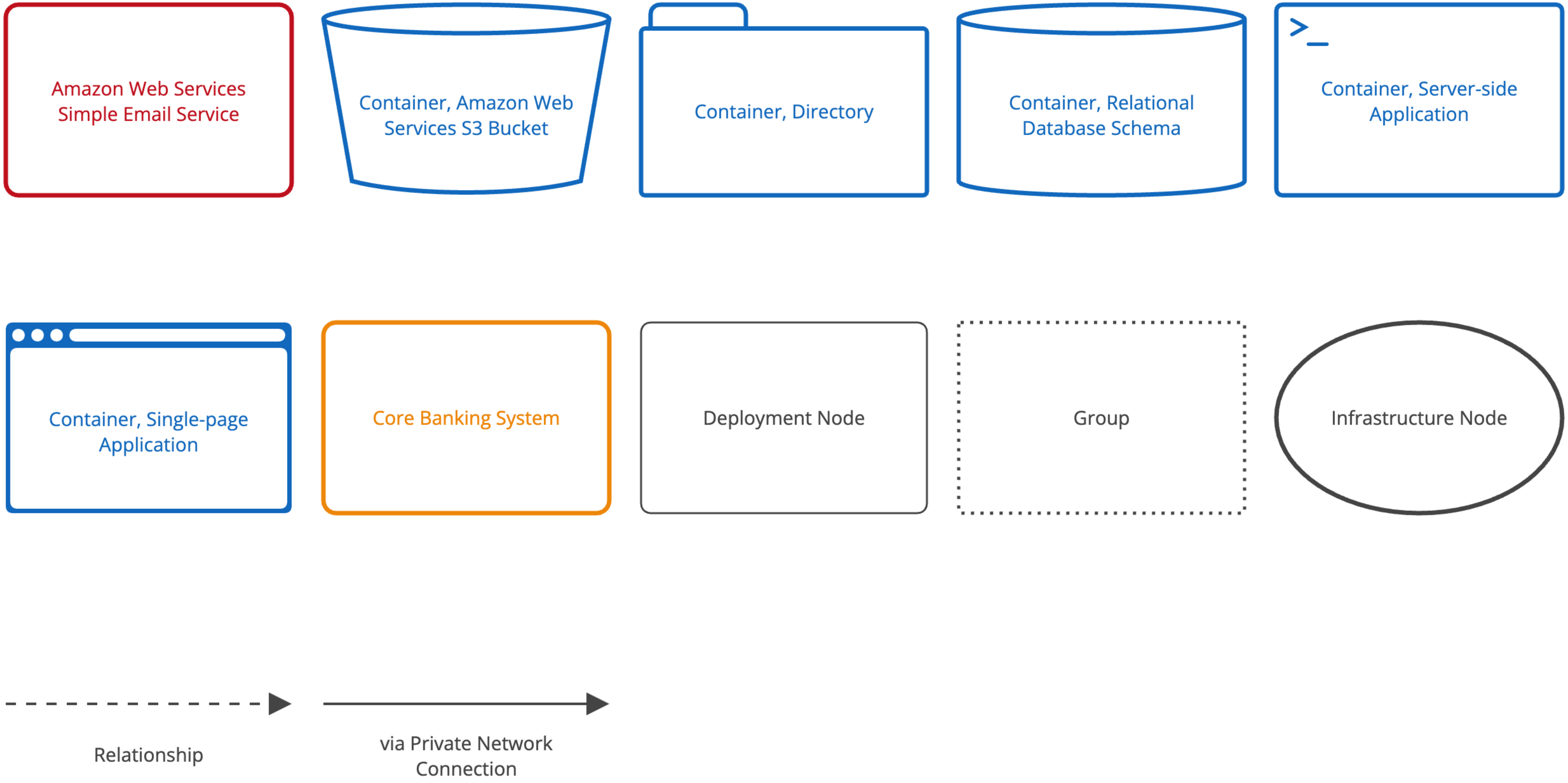
An example live deployment scenario for the Internet Banking System

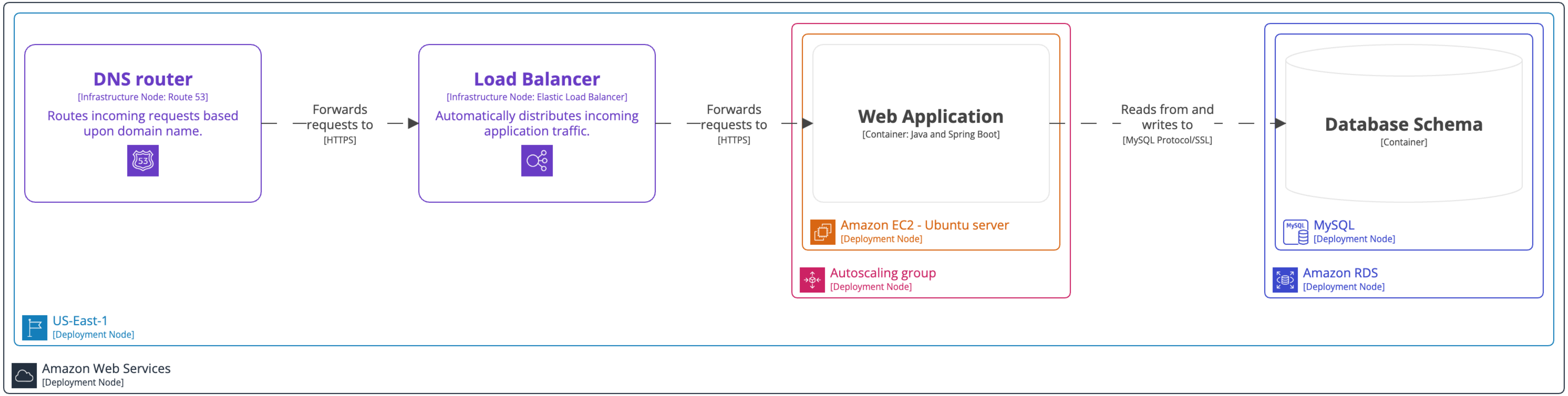

UI
[Container: JavaScript and Angular]
Single-page app that provides Internet banking functionality to customers via their web browser.
Web Browser
[Deployment Node: Chrome, Firefox, Safari, or Edge]
Customer's computer
[Deployment Node: Microsoft Windows or Apple macOS]
Delivers
Makes API requests to
[JSON/HTTPS]
ib.bigbank.com
[Infrastructure Node: DNS CNAME]
Proxied DNS CNAME record providing cached access to static content.
ib-api.bigbank.com
[Infrastructure Node: DNS CNAME]
Unproxied DNS CNAME record providing access to the Internet Banking System Backend.
DNS Services
[Deployment Node: Cloudflare DNS]
Cloudflare
Is an alias for
[HTTPS]
Is an alias for
[JSON/HTTPS]
Static Content
[Container: Directory]
HTML, CSS, JavaScript, etc.
Static Content Store
[Deployment Node: S3 Bucket]
Load Balancer
[Infrastructure Node: Application Load Balancer]
Forwards incoming HTTPS traffic to the Backend.
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Forwards API requests to
[JSON/HTTPS]
Sends e-mails to customers using
[JSON/HTTPS]

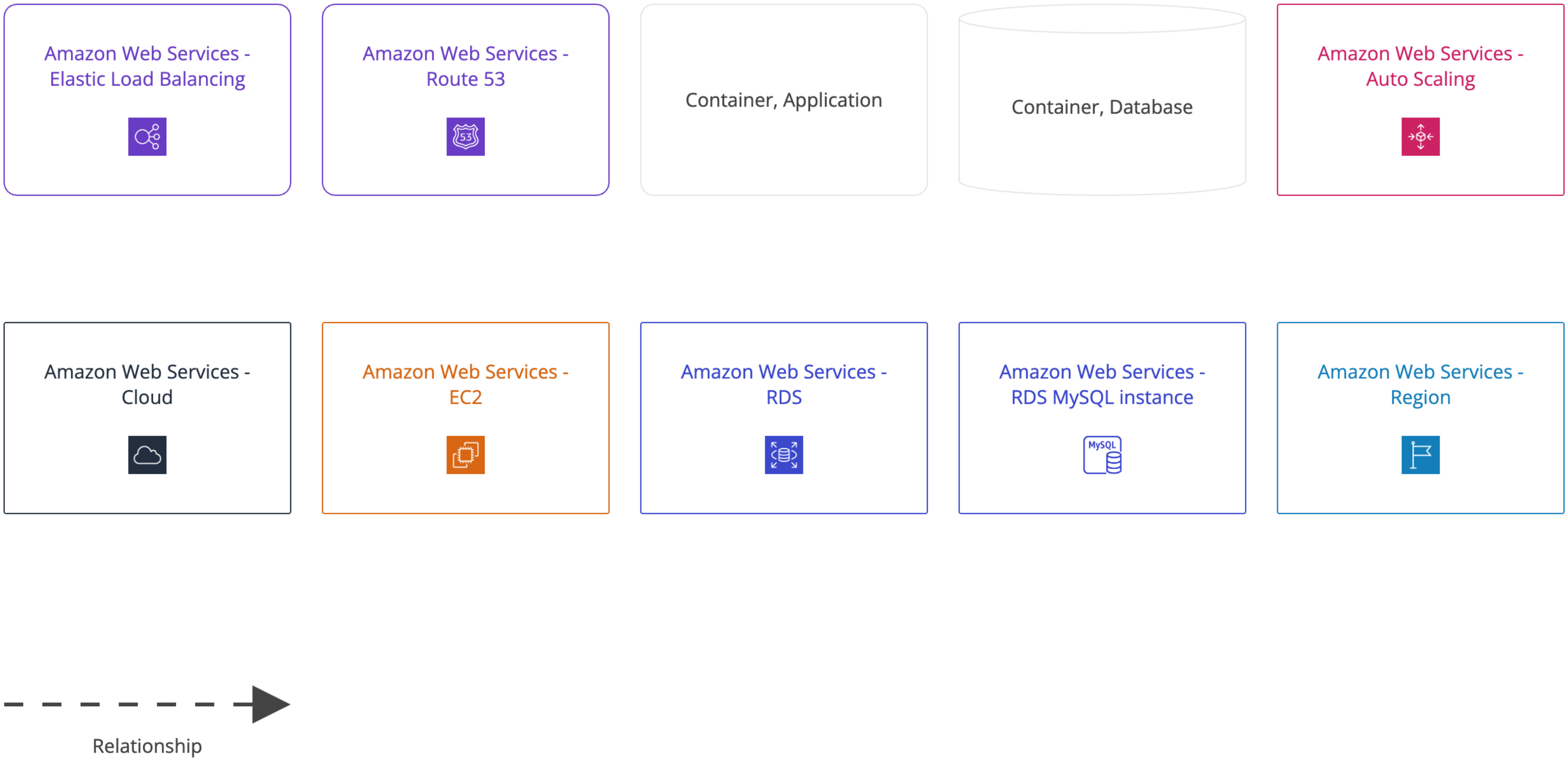
Static Content
[Container: Directory]
HTML, CSS, JavaScript, etc.
Static Content Store
[Deployment Node: S3 Bucket]
Load Balancer
[Infrastructure Node: Application Load Balancer]
Forwards incoming HTTPS traffic to the Backend.
Amazon Web Services Simple Email Service
[Software System]
Cloud-based email service provider.
Forwards API requests to
[JSON/HTTPS]
Sends e-mails to customers using
[JSON/HTTPS]
>_
Backend
[Container: Java and Spring Boot]
Provides Internet banking functionality via a JSON/HTTPS API.
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.
Gets bank account information from and makes payments using
[XML/HTTPS]
Core Banking System
[Software System]
Handles core banking functions including customer information, bank account management, transactions, etc.
corebanking-live
[Deployment Node: Ubuntu 24.04 LTS]
Big Bank Data Center
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]
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]
eu-west-1
[Deployment Node: Amazon Web Services Region]
Amazon Web Services

Deployment View: Internet Banking System - Live

An example live deployment scenario for the Internet Banking System


Amazon Web Services Simple Email Service
Container, Amazon Web Services S3 Bucket
Container, Directory
Container, Relational Database Schema
>_
Container, Server-side Application
Container, Single-page Application
Core Banking System
Deployment Node
Group
Infrastructure Node
Relationship
via Private Network Connection


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
MySQL
[Deployment Node]
Amazon RDS
[Deployment Node]
US-East-1
[Deployment Node]
Amazon Web Services
[Deployment Node]



Amazon Web Services - Elastic Load Balancing

Amazon Web Services - Route 53

Container, Application

Container, Database

Amazon Web Services - Auto Scaling

Amazon Web Services - Cloud

Amazon Web Services - EC2

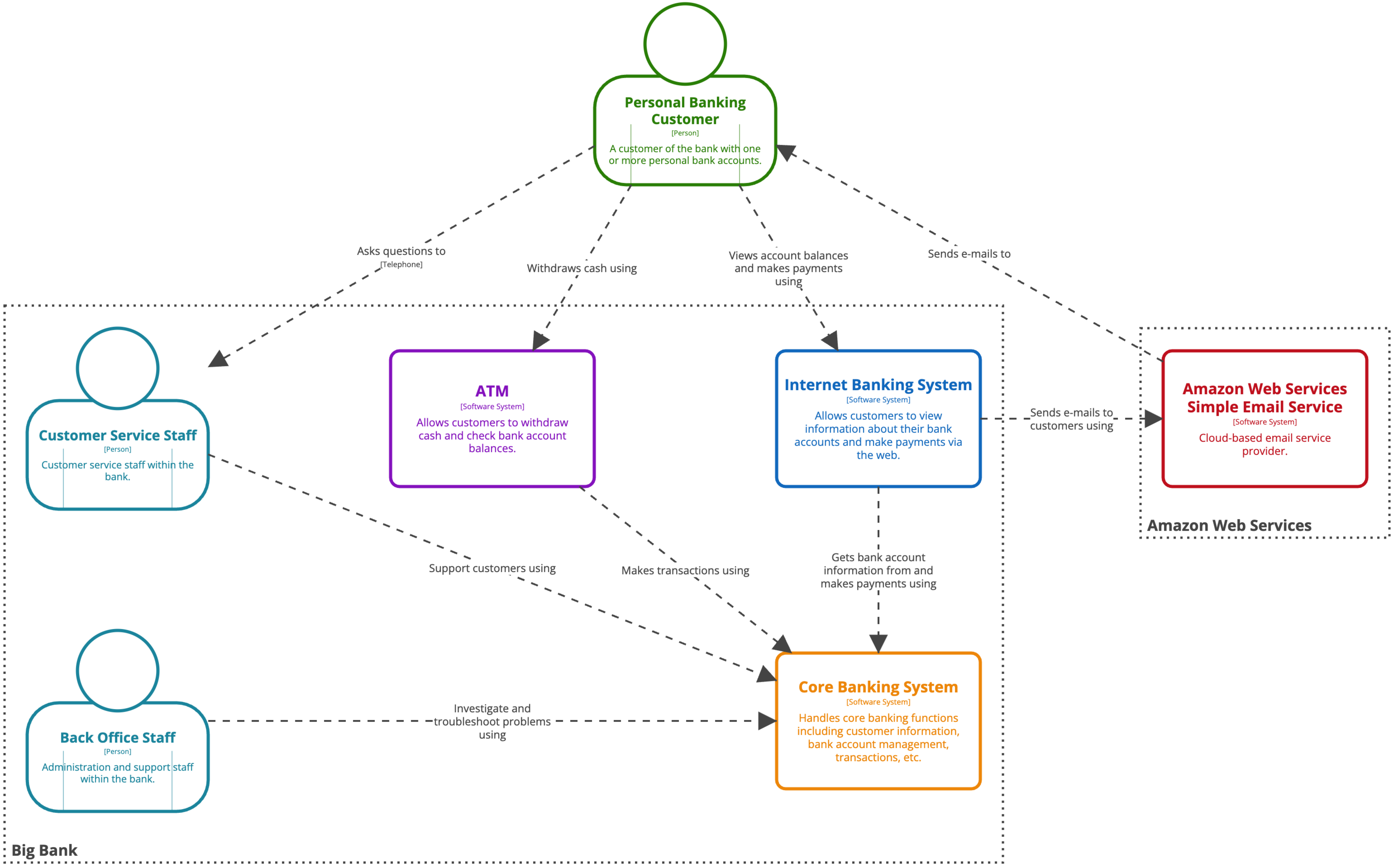
Amazon Web Services - RDS

Amazon Web Services - RDS MySQL instance

Amazon Web Services - Region

Relationship

System landscape diagrams


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

System Landscape View

A partial system landscape diagram for a fictional bank

Amazon Web Services Simple Email Service

ATM

Core Banking System

Group

Internet Banking System

Person, Bank Staff

Person, Customer

Relationship

FAQ

What's the inspiration behind the C4 model?

How widely used is the C4 model?

I’ve run software architecture workshops

in **~40 countries**

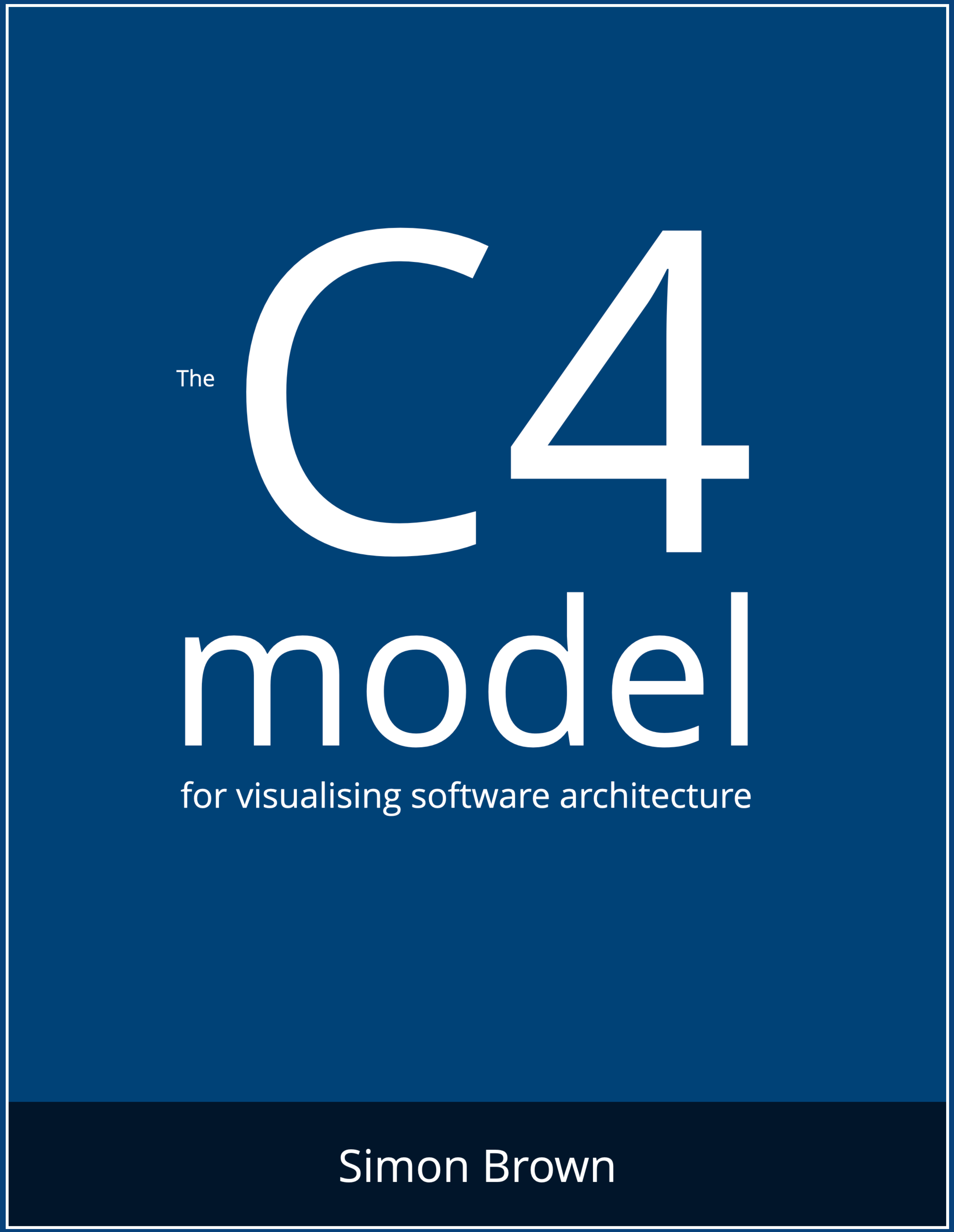
for **10,000+ people**

across most industry sectors

Academic establishments

A free subscription is available for students and staff at academic establishments, **for teaching purposes** (e.g. preparation of teaching material, use in assignments, etc). It's based upon the regular cloud service subscription with 5 workspaces, and is granted automatically to users who sign up with an e-mail address from the following 94 academic establishments:

- Facultad de Ingeniería de la Universidad de Buenos Aires, Argentina (`@fi.uba.ar`)
- Universidad Tecnológica Nacional, Argentina (`@ca.frre.utn.ed.ar`, `@alu.frt.utn.edu.ar`, `@frt.utn.edu.ar`, `@doc.frt.utn.edu.ar`)
- RMIT University, Australia (`@rmit.edu.au`, `@student.rmit.edu.au`)
- University of Queensland, Australia (`@uq.edu.au`, `@uq.net.au`, `@student.uq.edu.au`)
- University of Tasmania, Australia (`@utas.edu.au`)
- Vienna University of Economics and Business, Austria (`@wu.ac.at`, `@s.wu.ac.at`)
- Howest University of Applied Sciences, Belgium (`@howest.be`, `@student.howest.be`)
- PXL University of Applied Sciences and Arts, Belgium (`@pxl.be`, `@student.pxl.be`)
- Universidade Federal de Mato Grosso do Sul, Brazil (`@ufms.br`, `@facom.ufms.br`)
- Universidade Federal do Pará, Brazil (`@ig.ufpa.br`, `@icen.ufpa.br`)
- Universidade federal de Pernambuco, Brazil (`@ufpe.br`, `@cin.ufpe.br`)
- Université de Sherbrooke, Canada (`@usherbrooke.ca`)
- École de Technologie Supérieure, Canada (`@etsmtl.ca`, `@ens.etsmtl.ca`)



The C4 model

for visualising software architecture

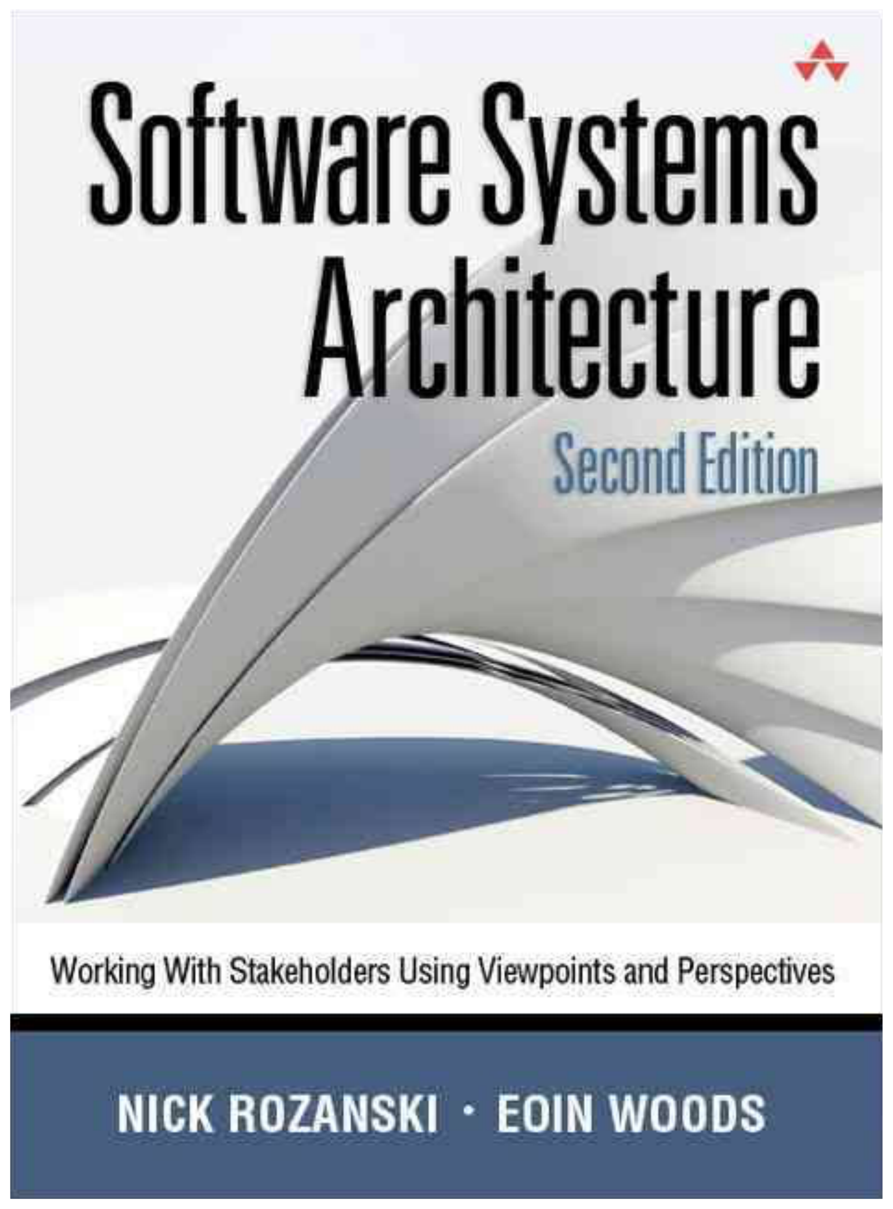
Simon Brown

My C4 model book is also used as course material in many other universities

Are the diagrams for design
or documentation purposes?

Perspectives

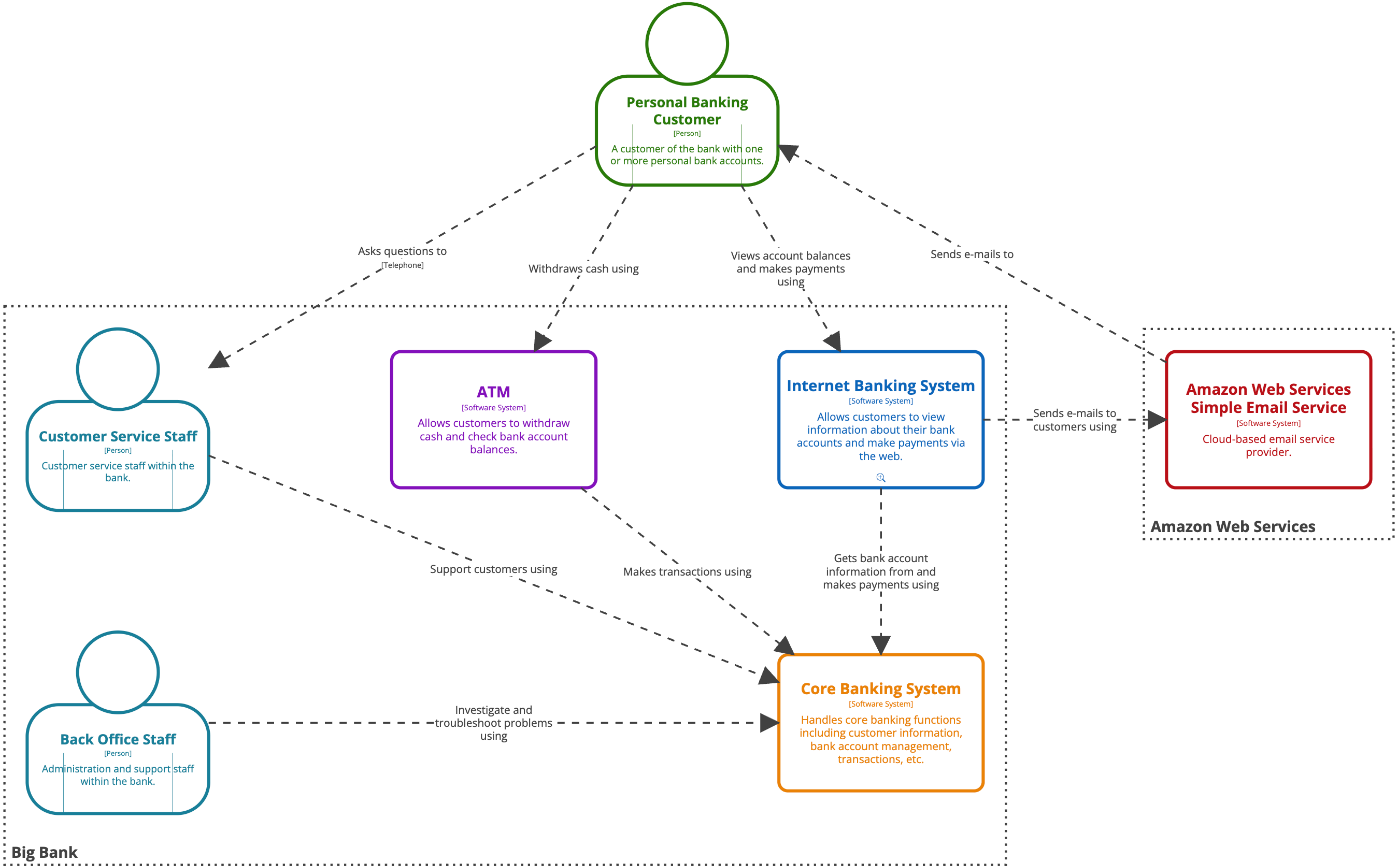
What about ownership, security, technical debt, etc?

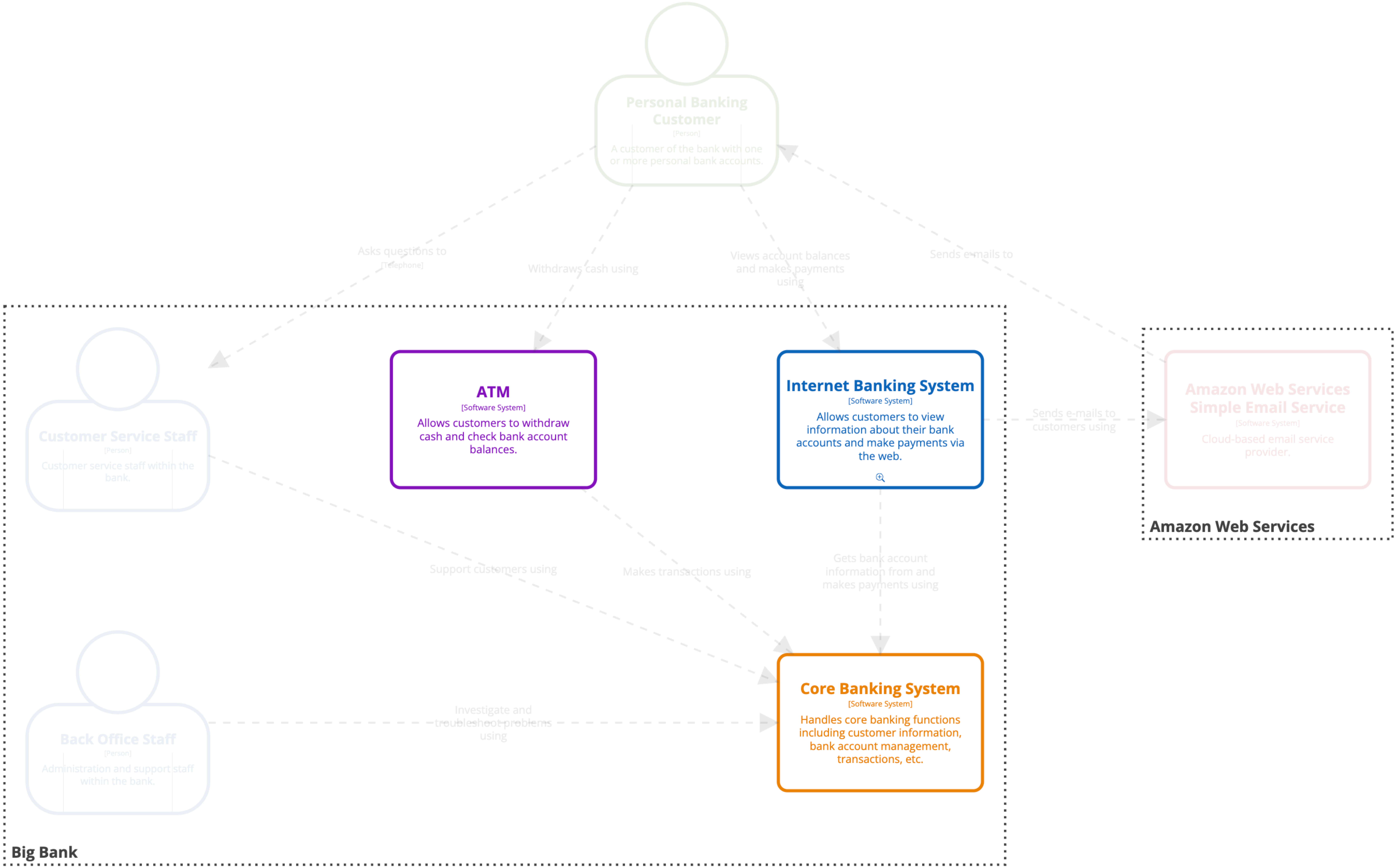

Software Systems Architecture
Second Edition
Working With Stakeholders Using Viewpoints and Perspectives
NICK ROZANSKI · EOIN WOODS

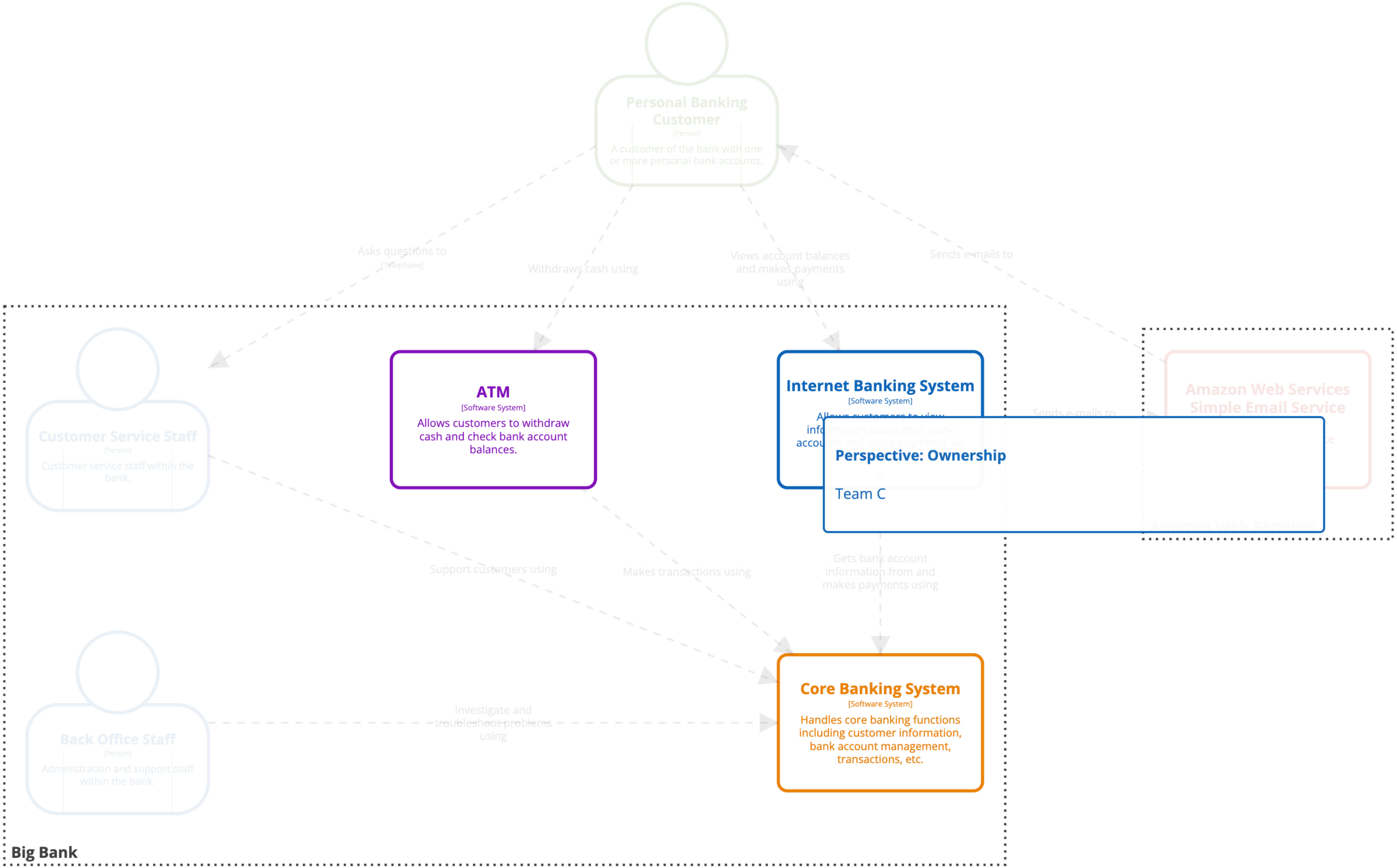
Perspectives

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]
Allows customers to view information about their bank accounts and make payments via

Sends e-mails to

Amazon Web Services Simple Email Service

Amazon Web Services

Perspective: Ownership

Team C

Support customers using

Makes transactions using

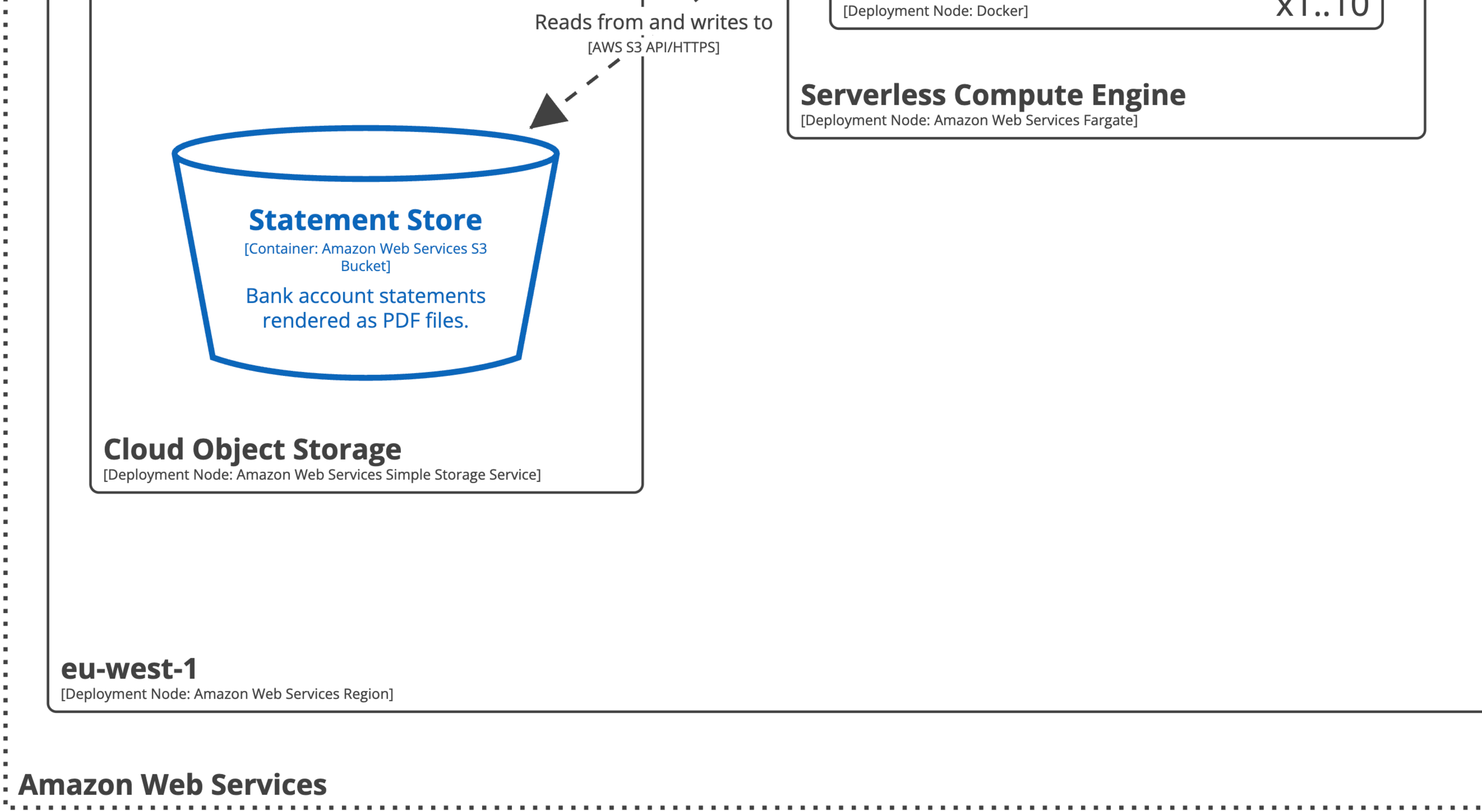
Gets bank account information from and makes payments using

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



Reads from and writes to
[AWS S3 API/HTTPS]

[Deployment Node: Docker]
x1..10

Serverless Compute Engine
[Deployment Node: Amazon Web Services Fargate]

Statement Store
[Container: Amazon Web Services S3 Bucket]
Bank account statements rendered as PDF files.

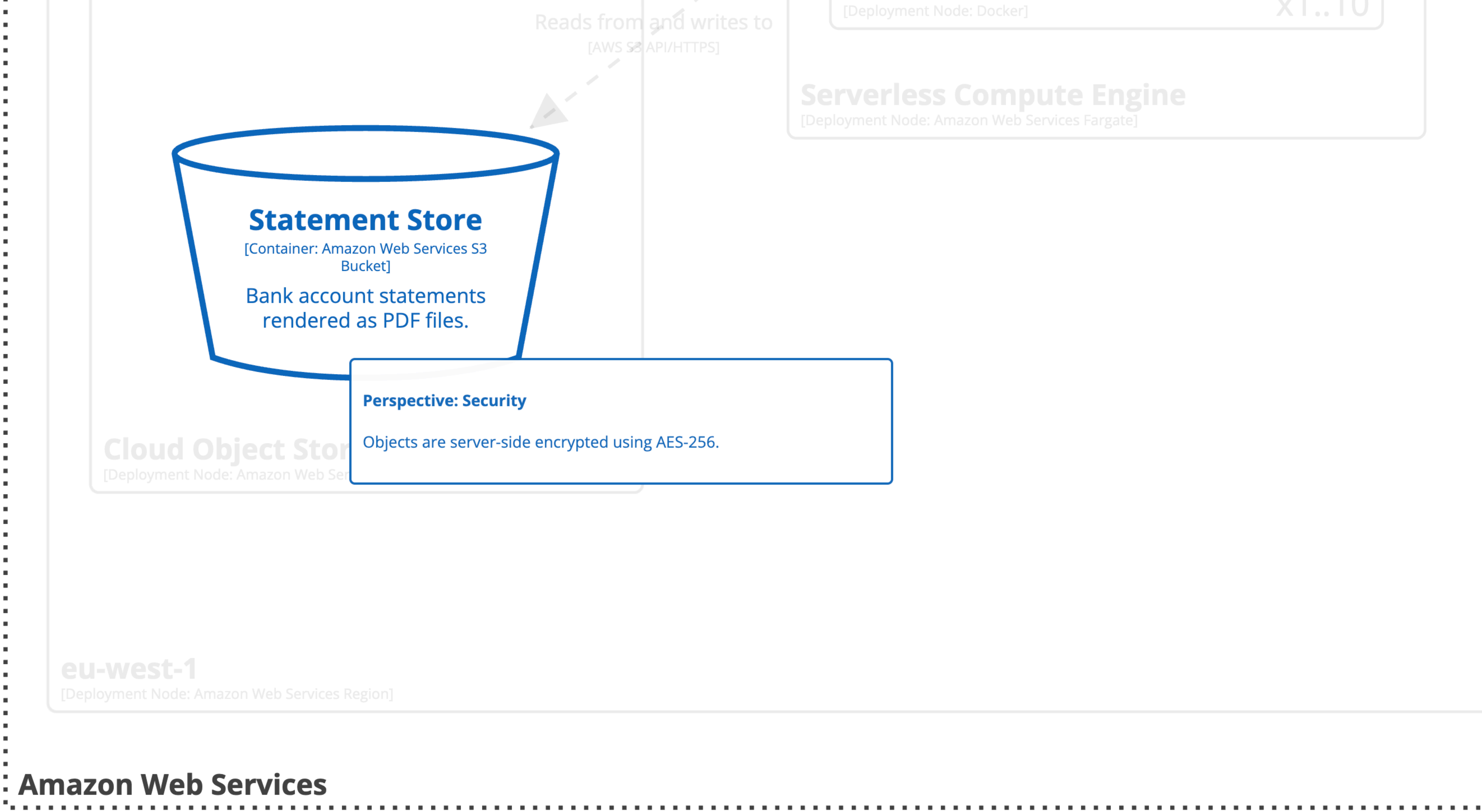
Cloud Object Storage
[Deployment Node: Amazon Web Services Simple Storage Service]

eu-west-1
[Deployment Node: Amazon Web Services Region]

Amazon Web Services

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



Reads from and writes to
[AWS S3 API/HTTPS]

[Deployment Node: Docker]

x1..10

Serverless Compute Engine
[Deployment Node: Amazon Web Services Fargate]

Statement Store
[Container: Amazon Web Services S3 Bucket]
Bank account statements rendered as PDF files.

Perspective: Security

Objects are server-side encrypted using AES-256.

Cloud Object Stor
[Deployment Node: Amazon Web Ser

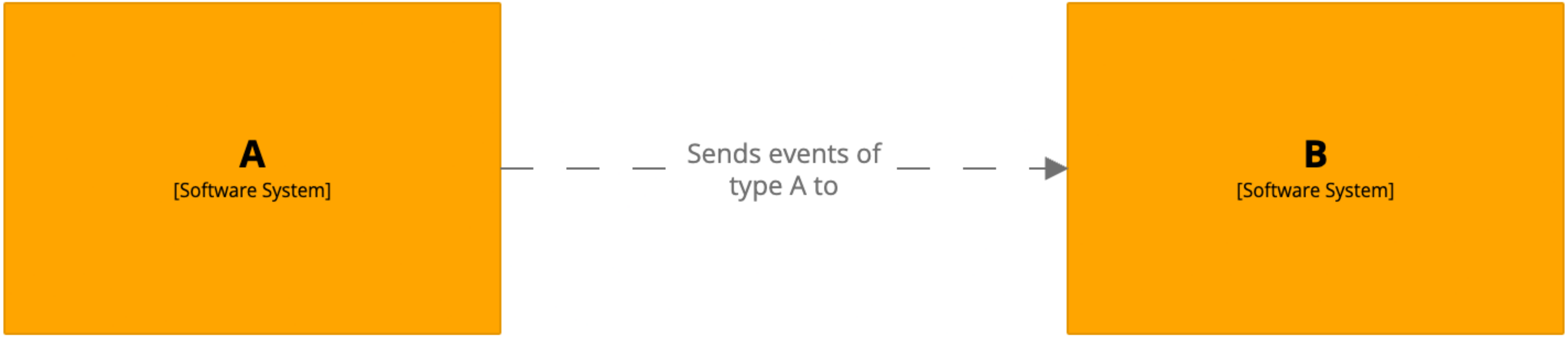
eu-west-1
[Deployment Node: Amazon Web Services Region]

Amazon Web Services

Deployment View: Internet Banking System - Live

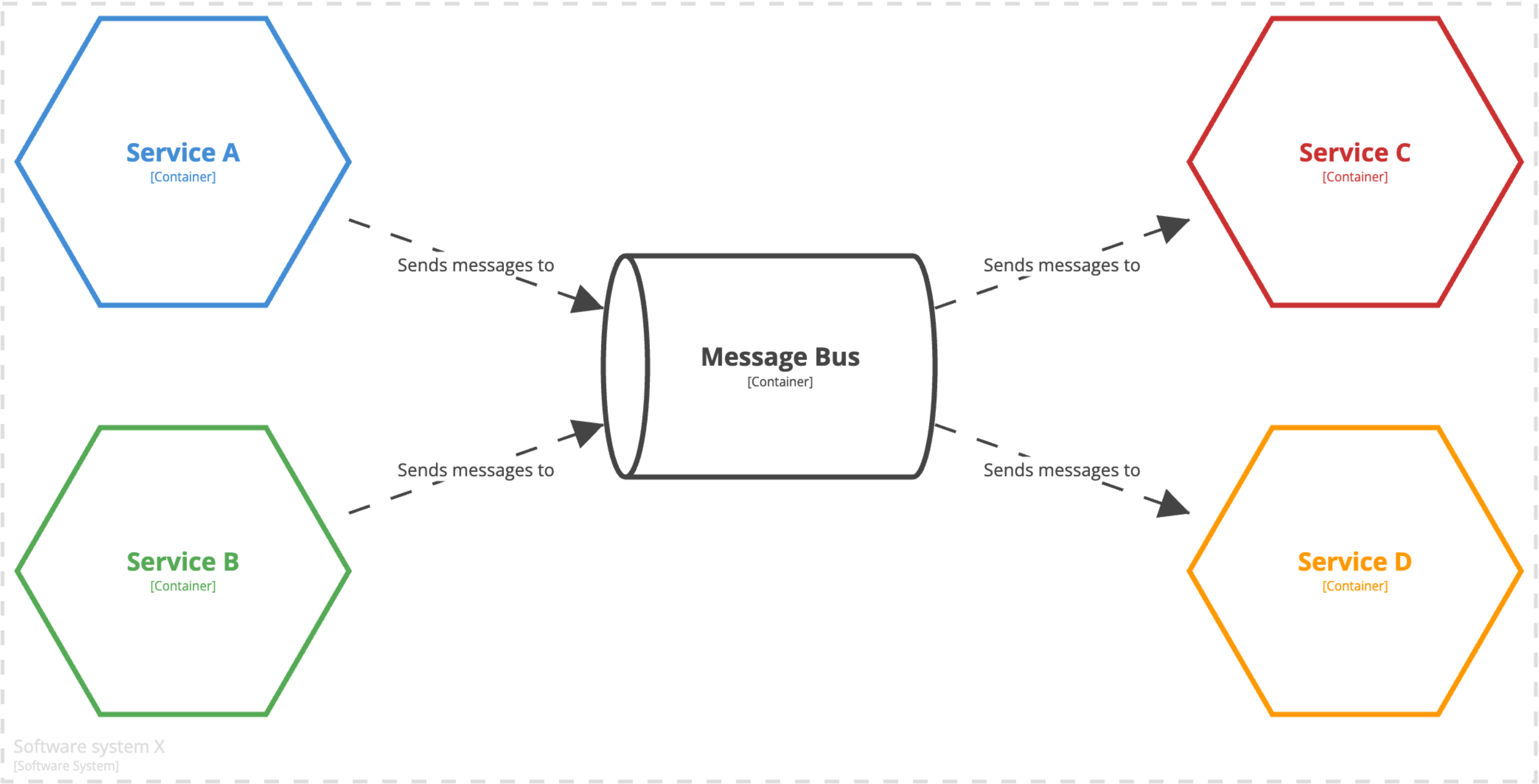
An example live deployment scenario for the Internet Banking System | Simon Brown | c4model.com | License: CC BY 4.0

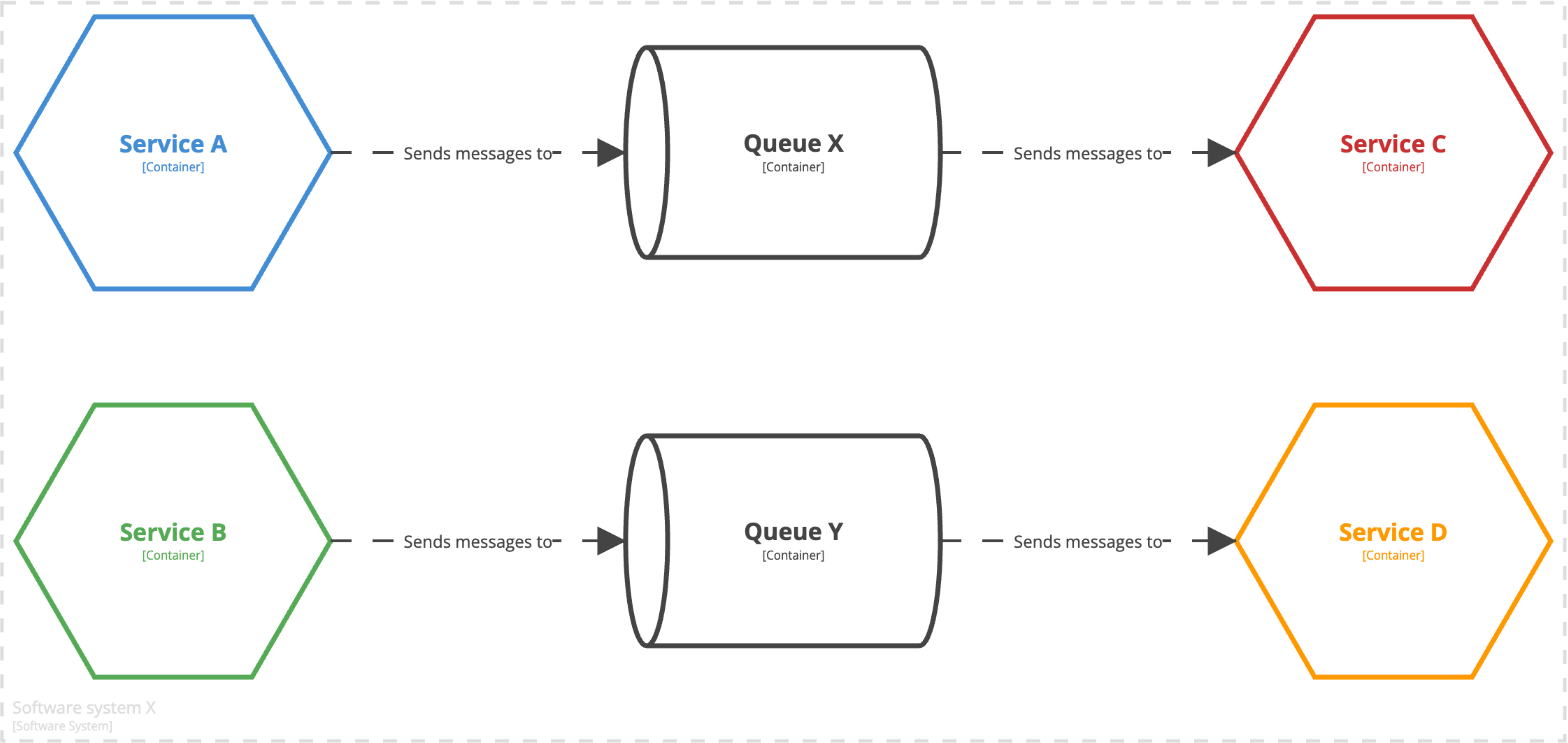
Event-driven architectures

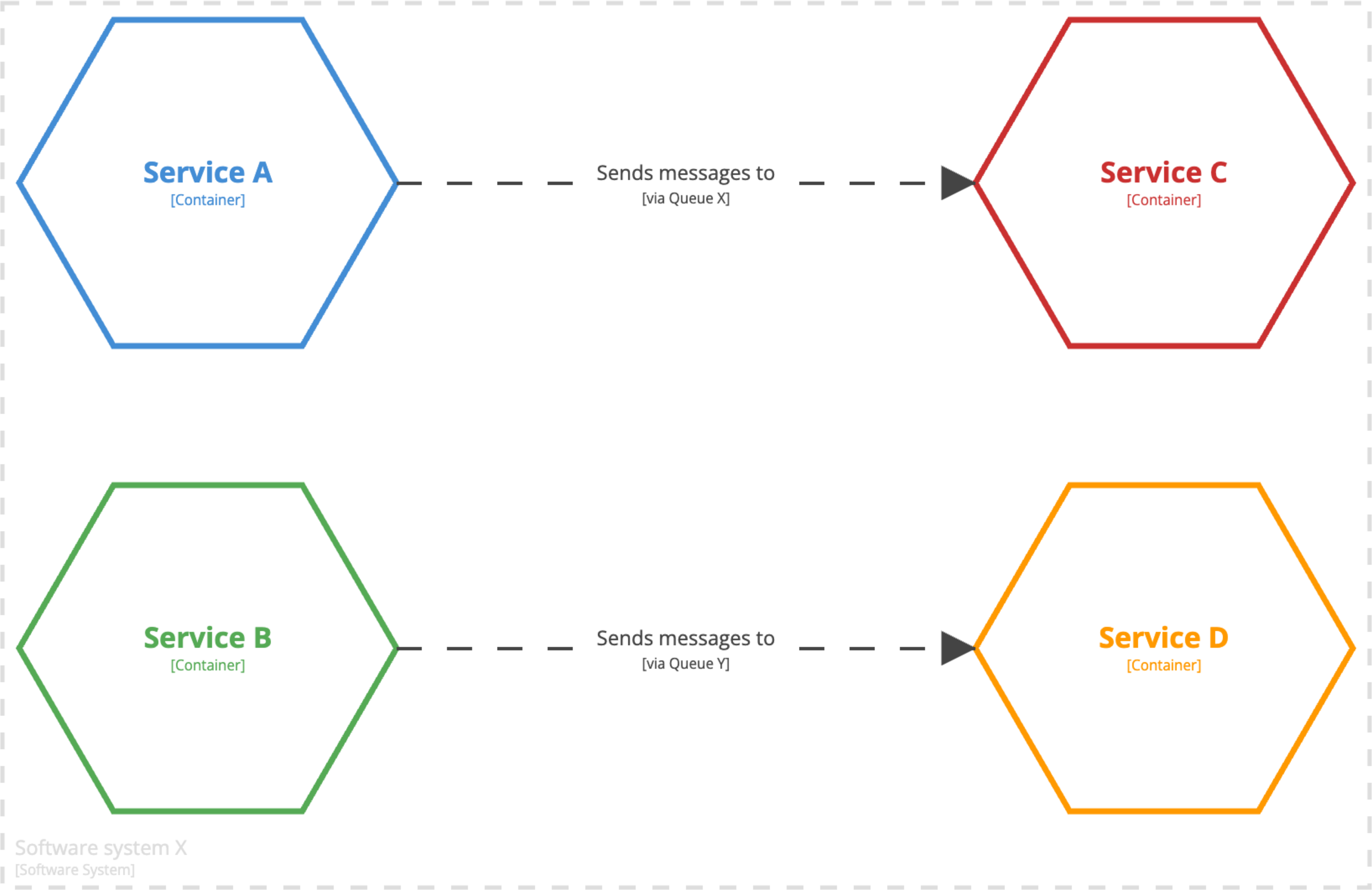

A
[Software System]
Sends events of type A to
B
[Software System]

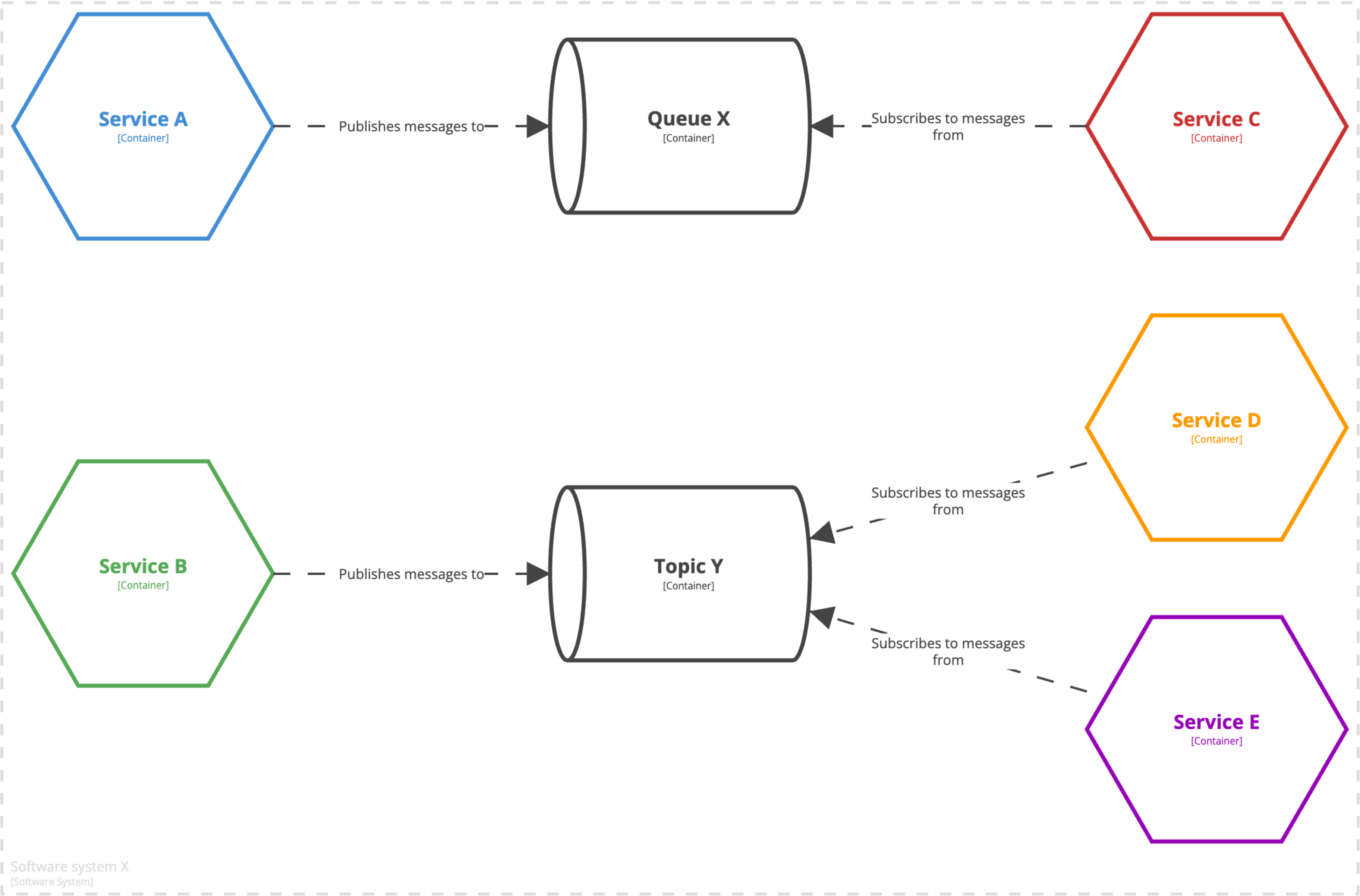
Message-driven architectures

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]


Service A
[Container]
Sends messages to
Queue X
[Container]
Sends messages to
Service C
[Container]
Service B
[Container]
Sends messages to
Queue Y
[Container]
Sends messages to
Service D
[Container]
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]

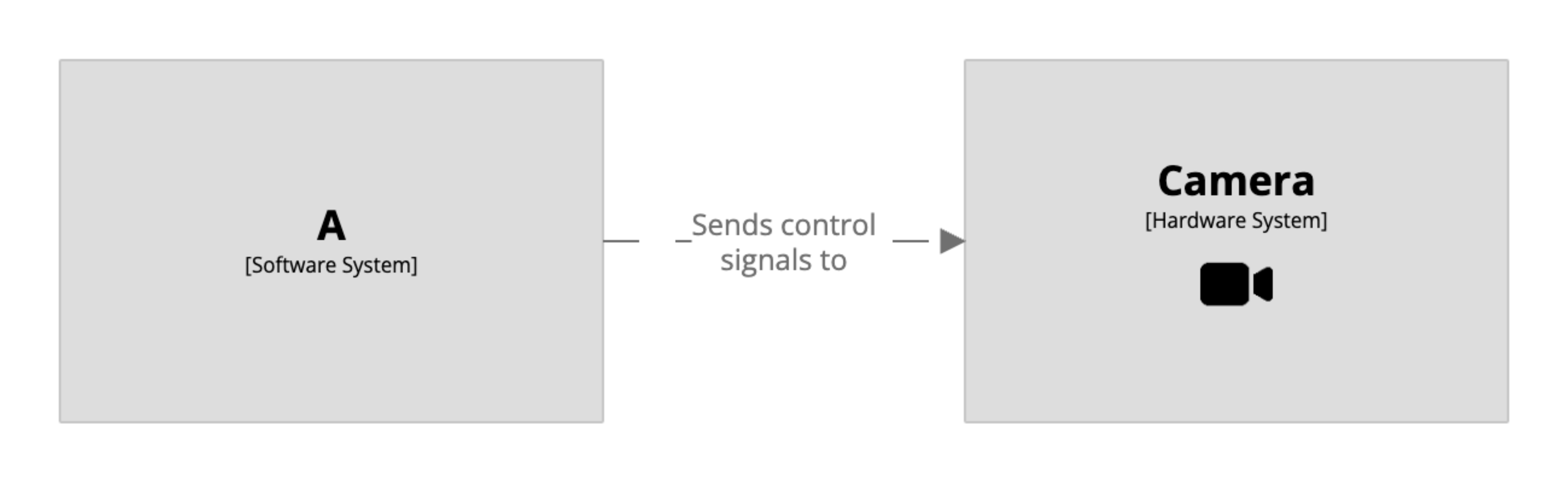

Service A
[Container]
Publishes messages to
Queue X
[Container]
Subscribes to messages from
Service C
[Container]
Service D
[Container]
Subscribes to messages from
Service B
[Container]
Publishes messages to
Topic Y
[Container]
Subscribes to messages from
Service E
[Container]
Software system X
[Software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

Abstraction

vs

organisation

What are your thoughts on modelling additional abstractions?



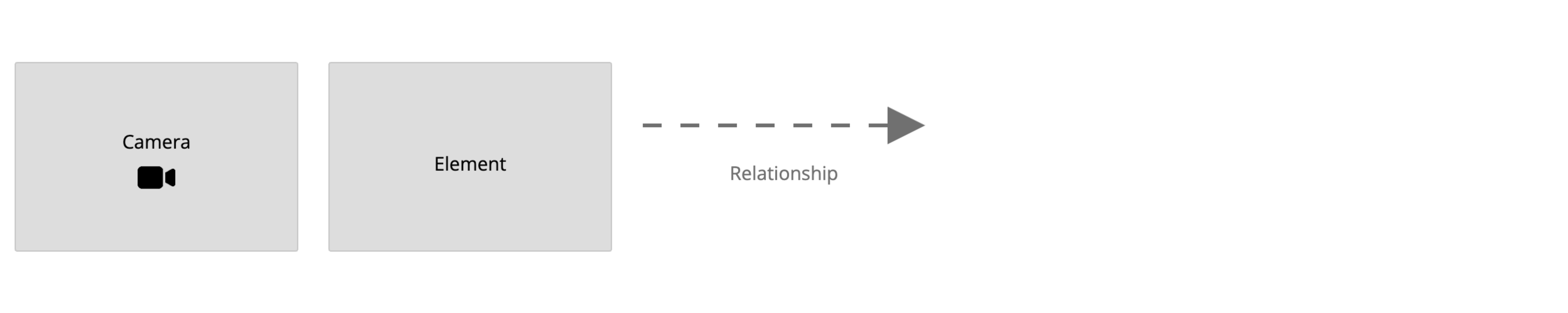
A

[Software System]

_Sends control signals to

Camera

[Hardware System]

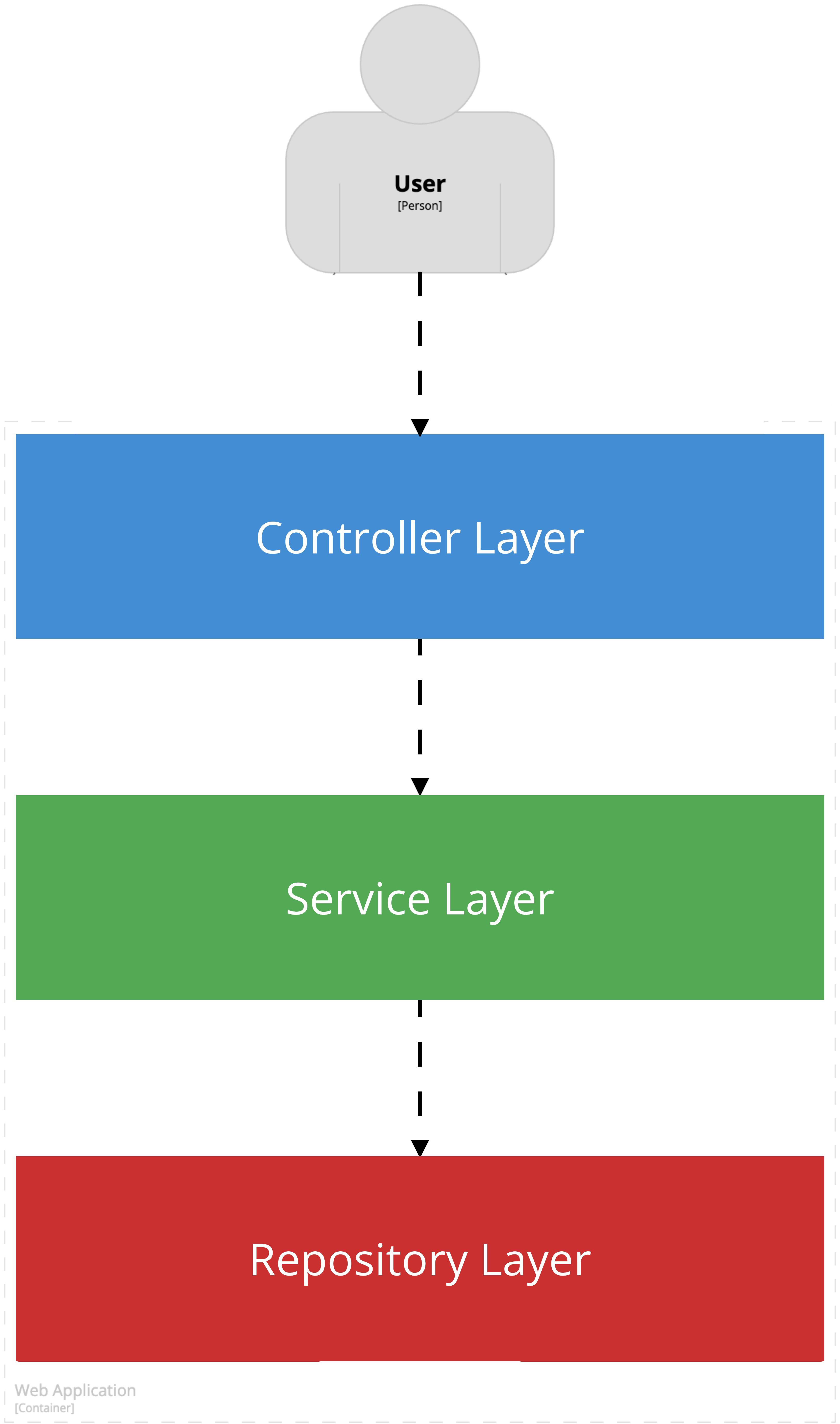


Camera

Element

Relationship

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



User
[Person]

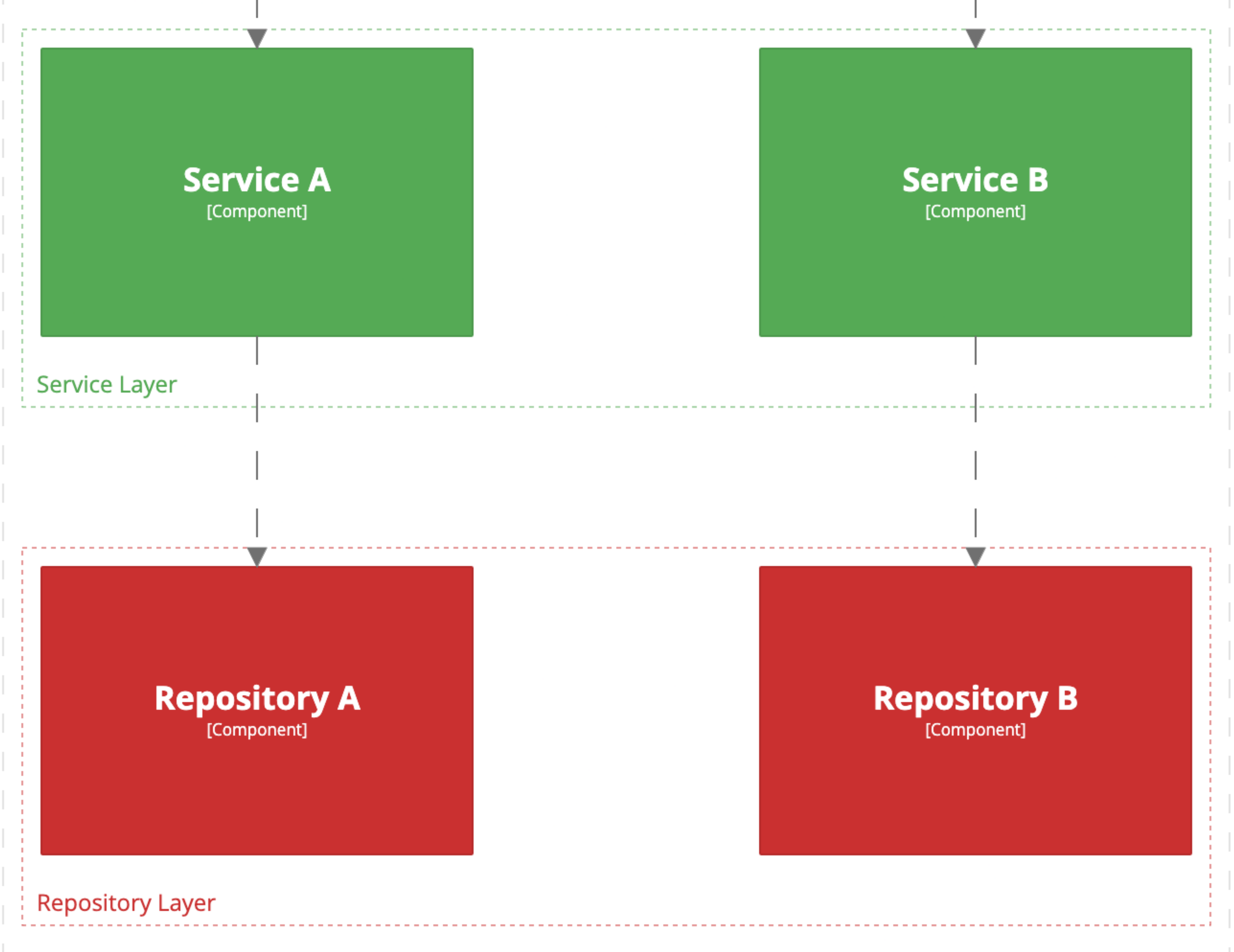
Controller Layer

Service Layer

Repository Layer

Web Application
[Container]

✗



Service A
[Component]

Service B
[Component]

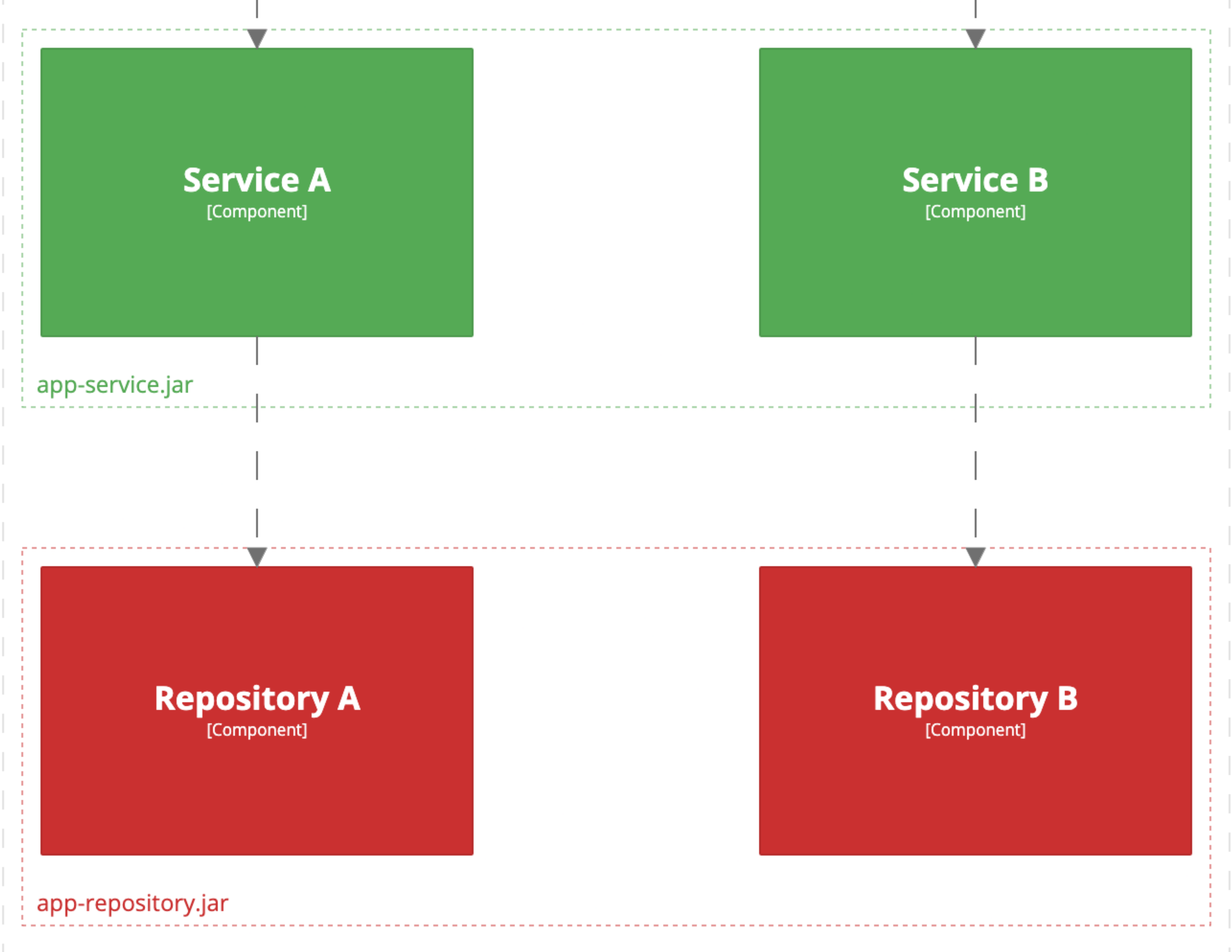
Service Layer

Repository A
[Component]

Repository B
[Component]

Repository Layer

✅



Service A
[Component]

Service B
[Component]

app-service.jar

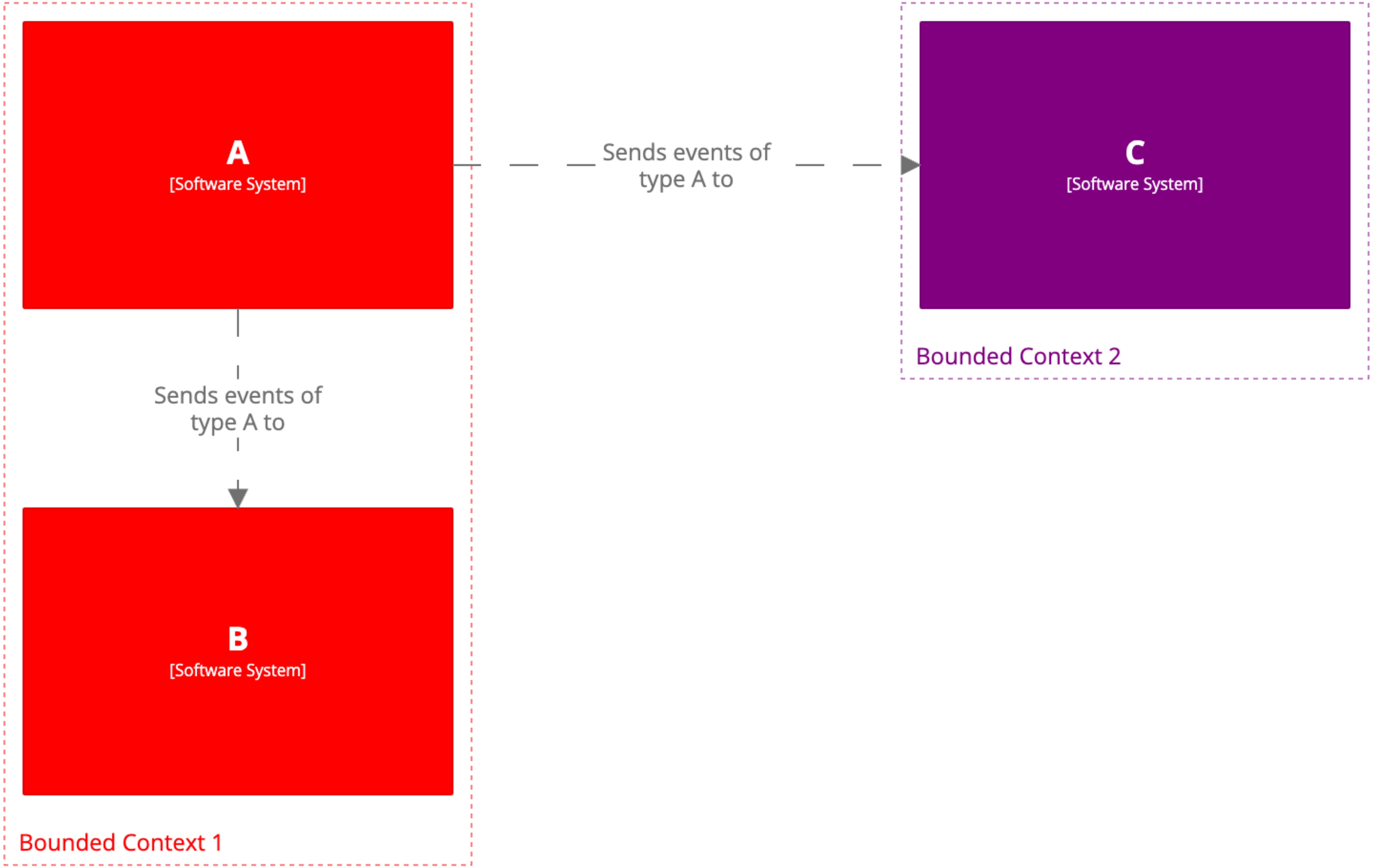
Repository A
[Component]

Repository B
[Component]

app-repository.jar

✅

Apply this concept to subsystems, bounded contexts, etc…

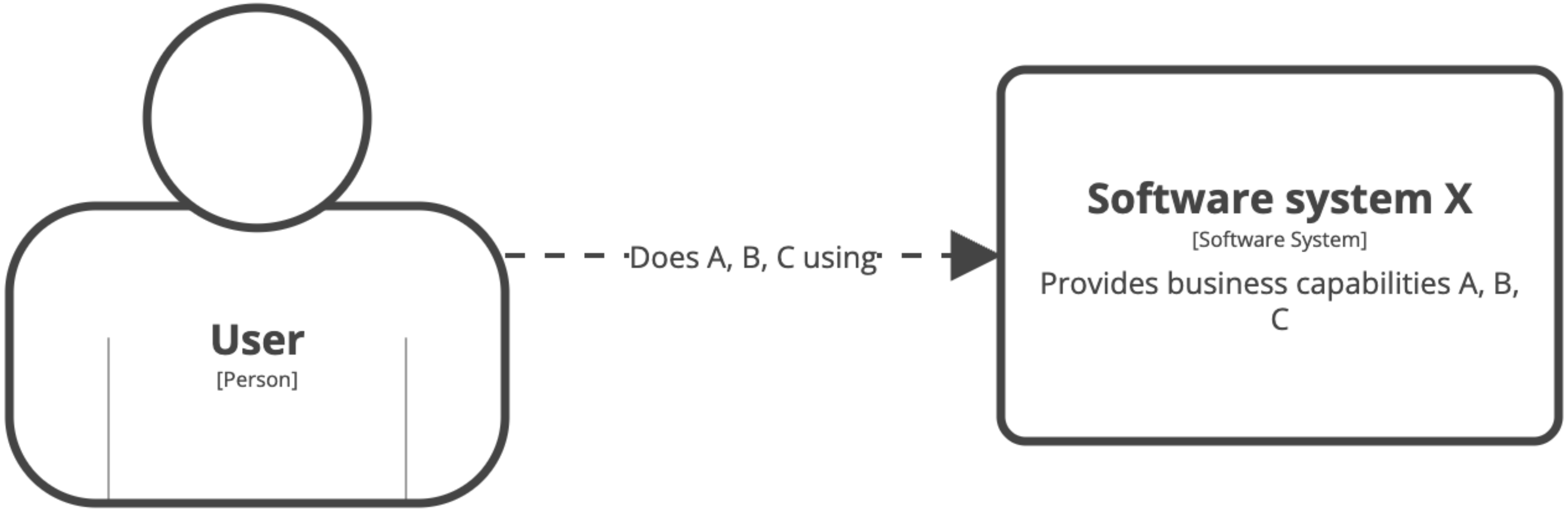

A
[Software System]
Sends events of type A to
C
[Software System]
Bounded Context 2
Sends events of type A to
B
[Software System]
Bounded Context 1

Microservices

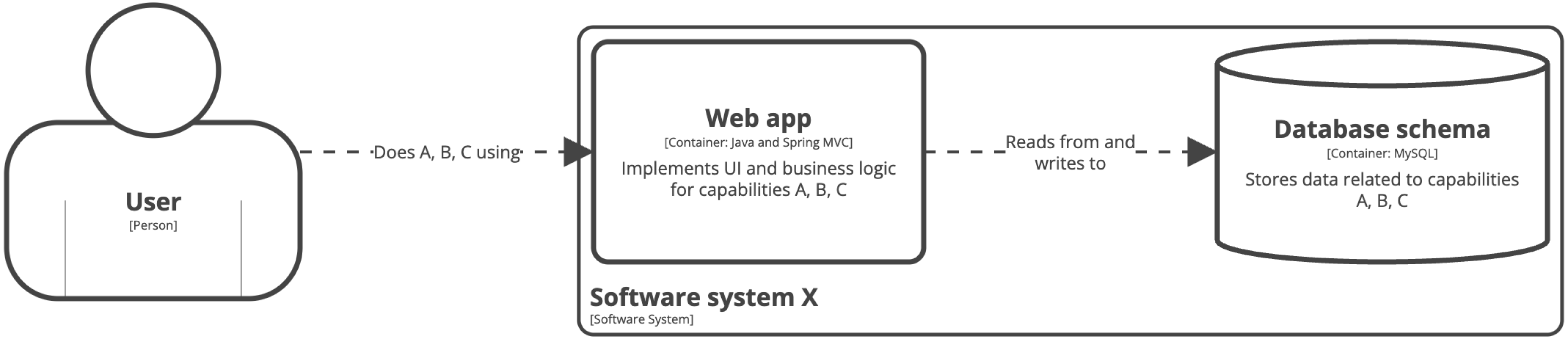
A microservice should be modelled as a **software system** or a **group of containers**

Stage 1: 💵

(monolithic architecture)



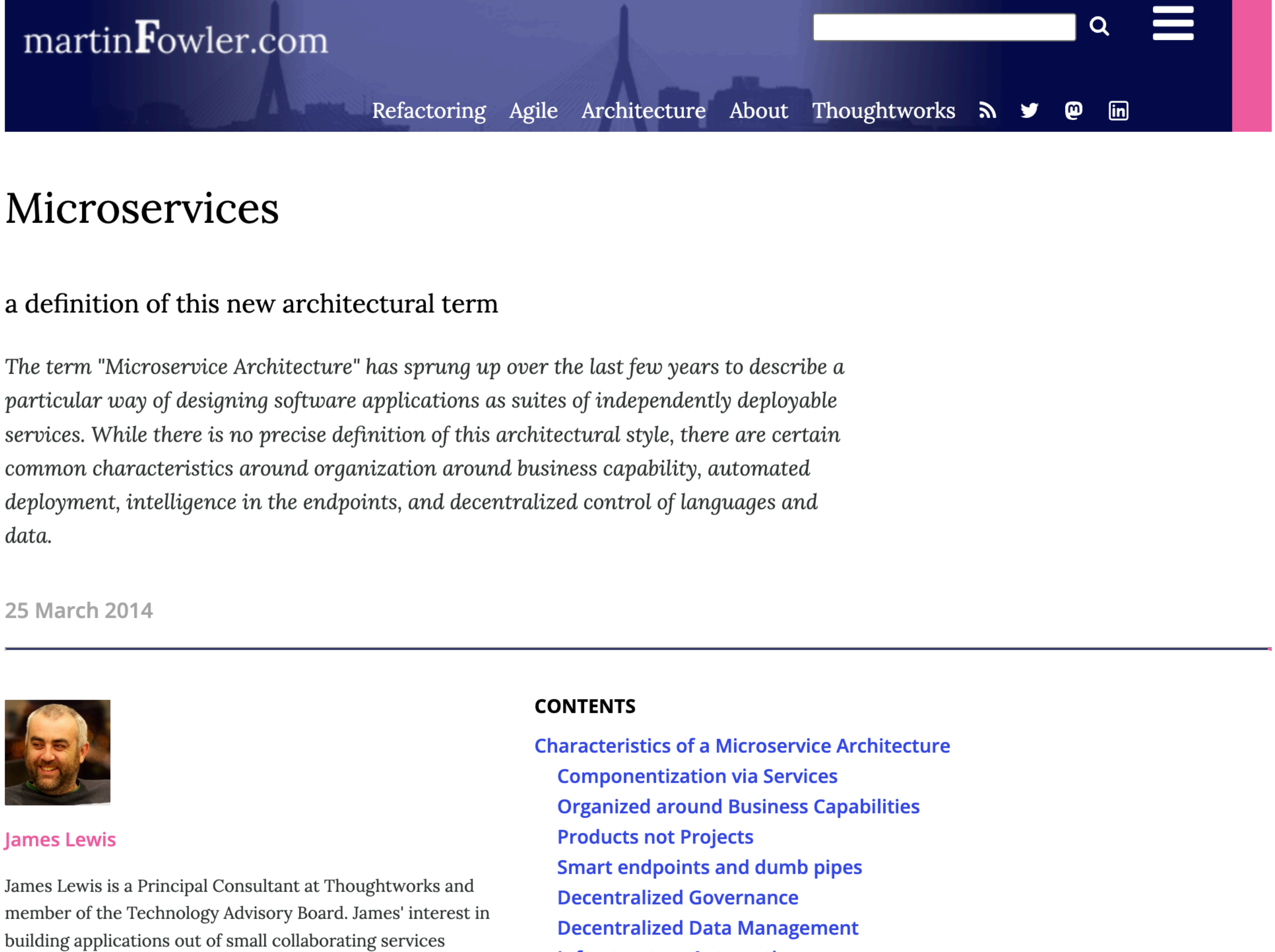
System Context View: Software system X


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

Container View: Software system X

Stage 2: 💵 💵

(microservices)



Microservices

a definition of this new architectural term

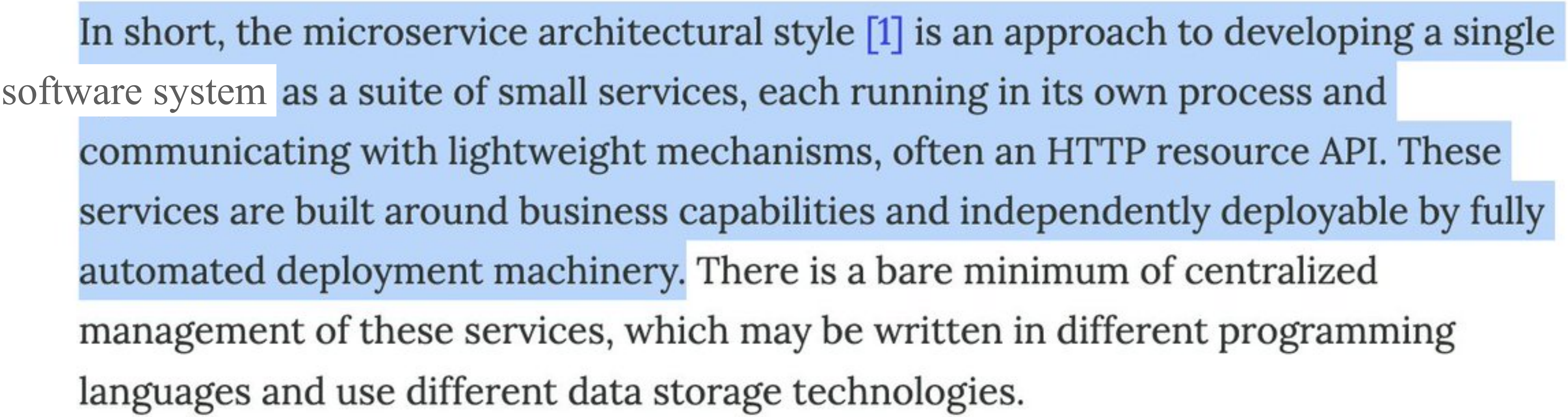
The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around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

25 March 2014

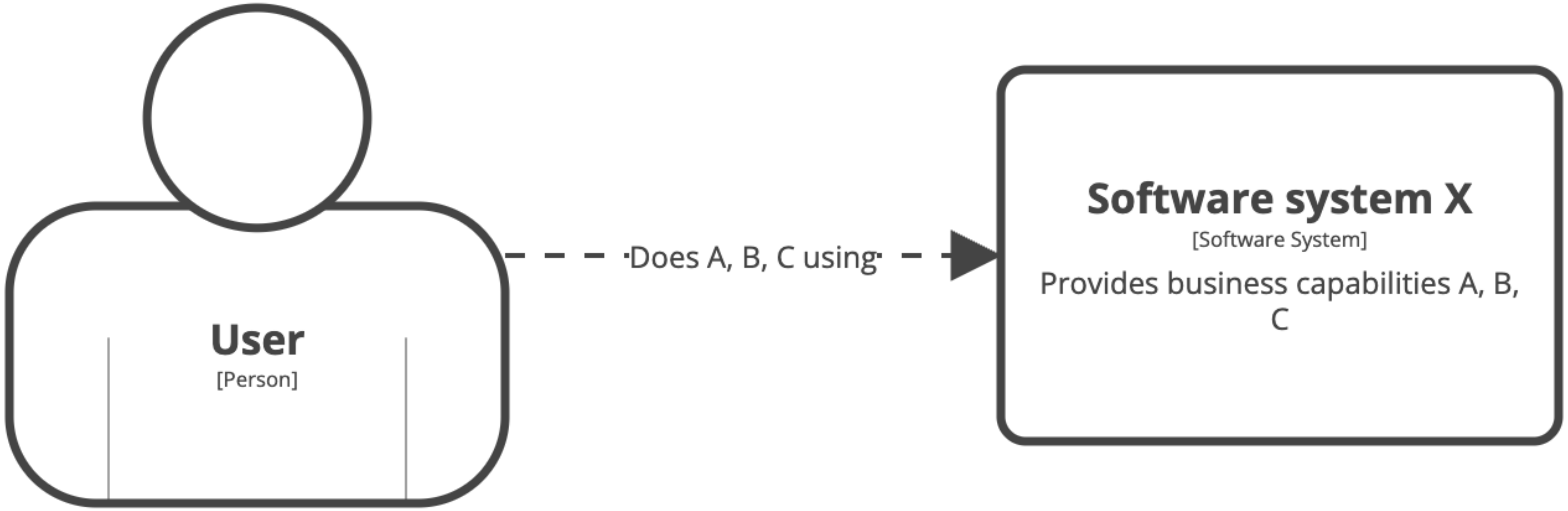
James Lewis

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



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.



User

[Person]

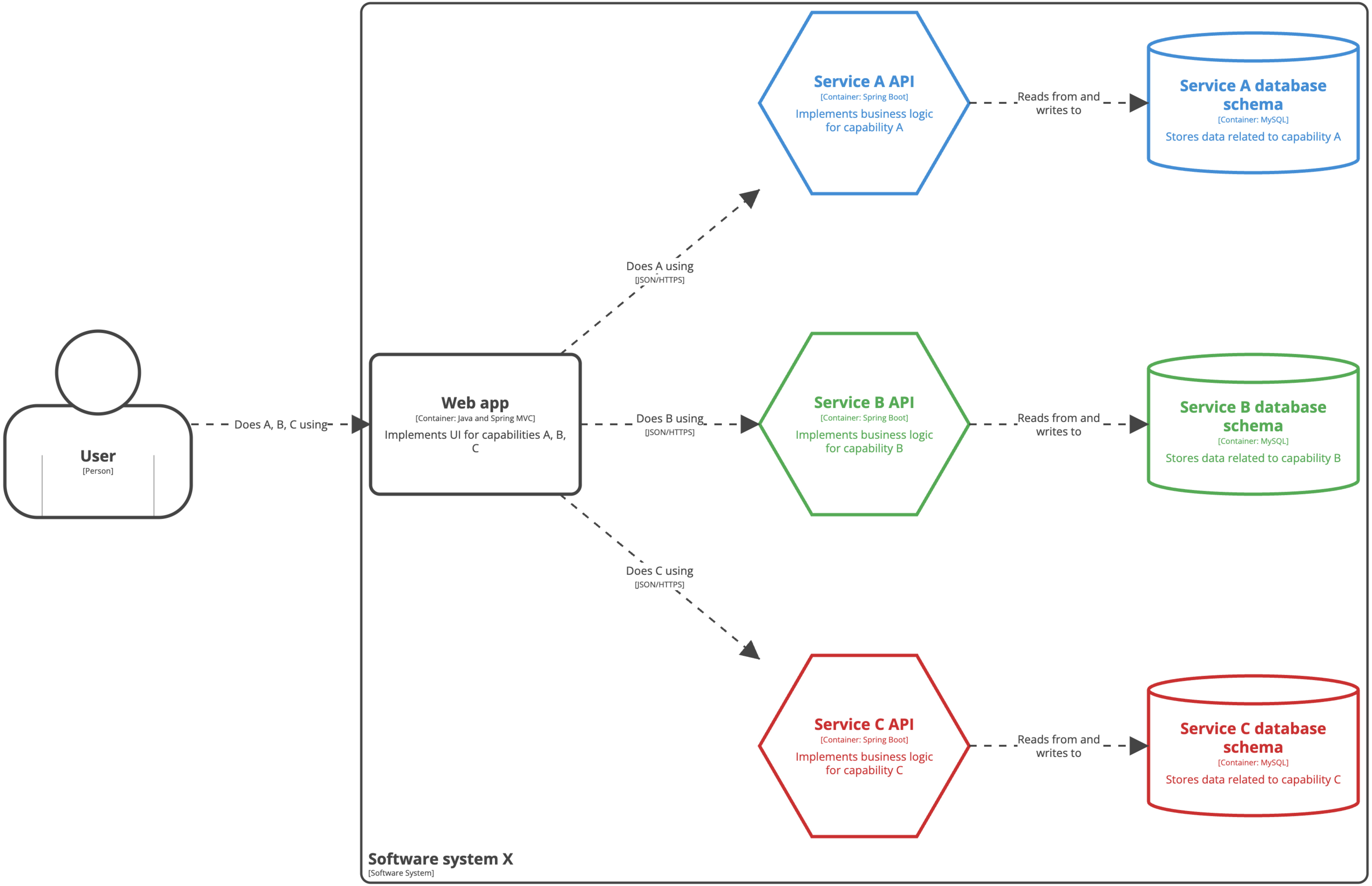
Does A, B, C using

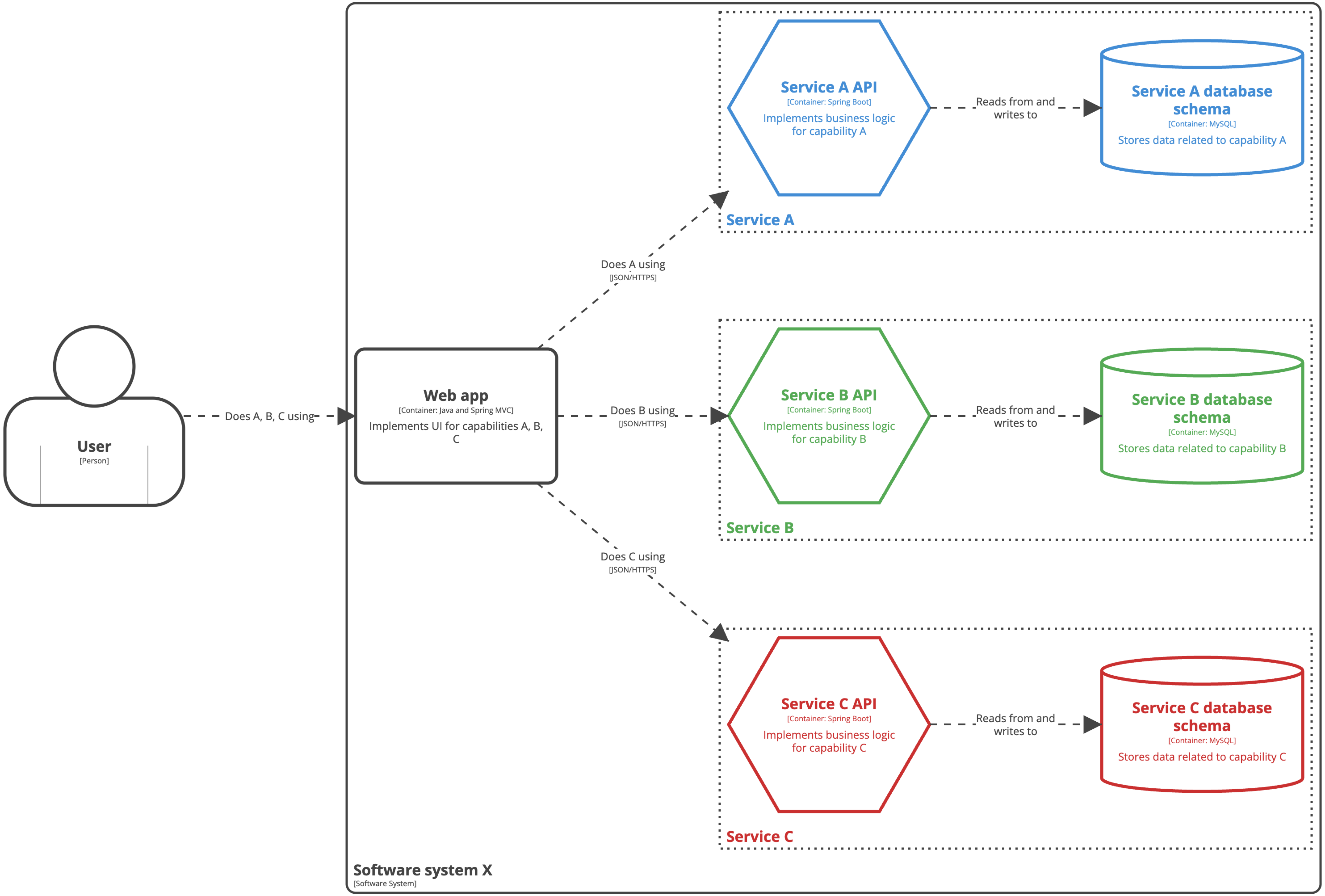
Software system X

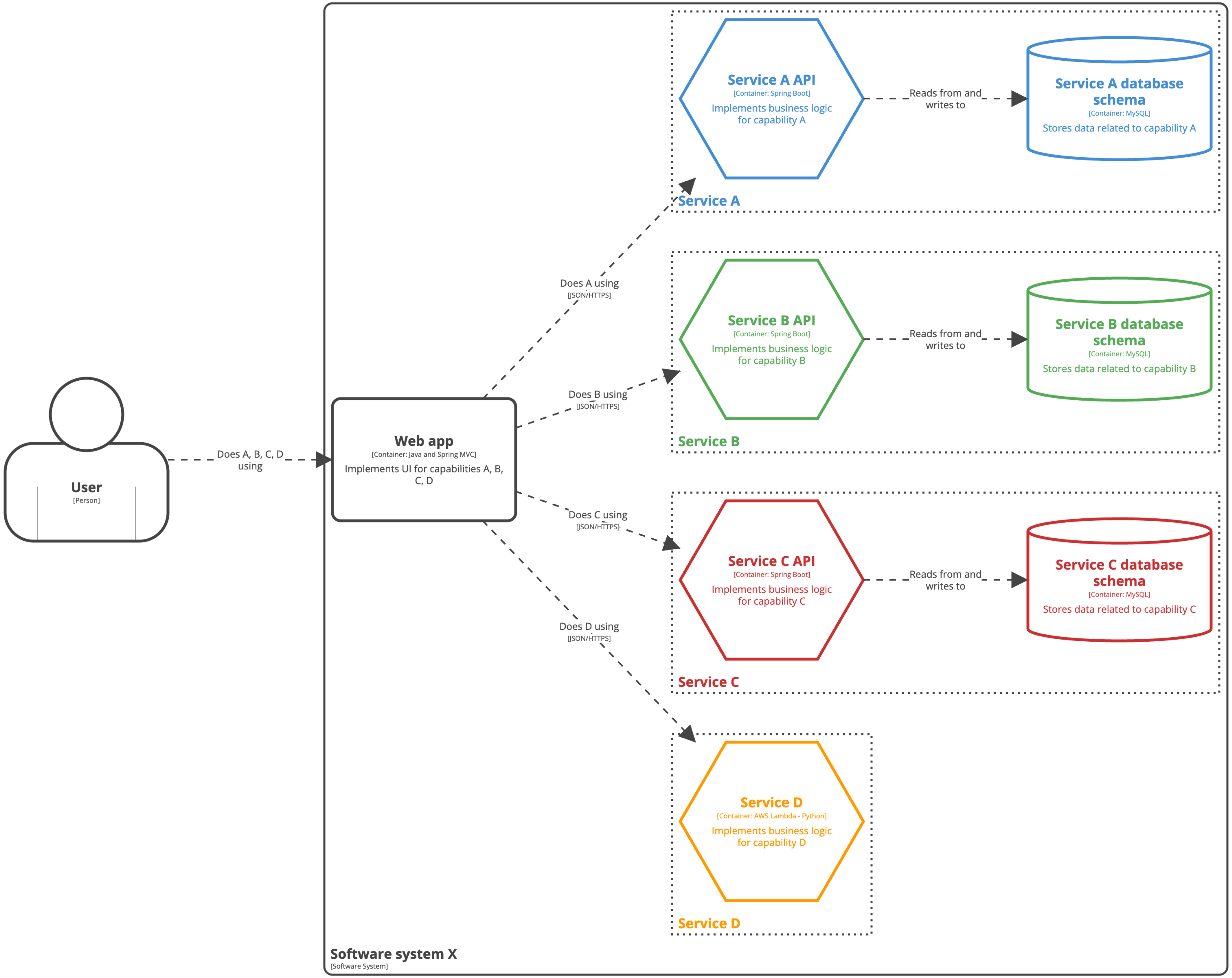
[Software System]

Provides business capabilities A, B, C

System Context View: Software system X

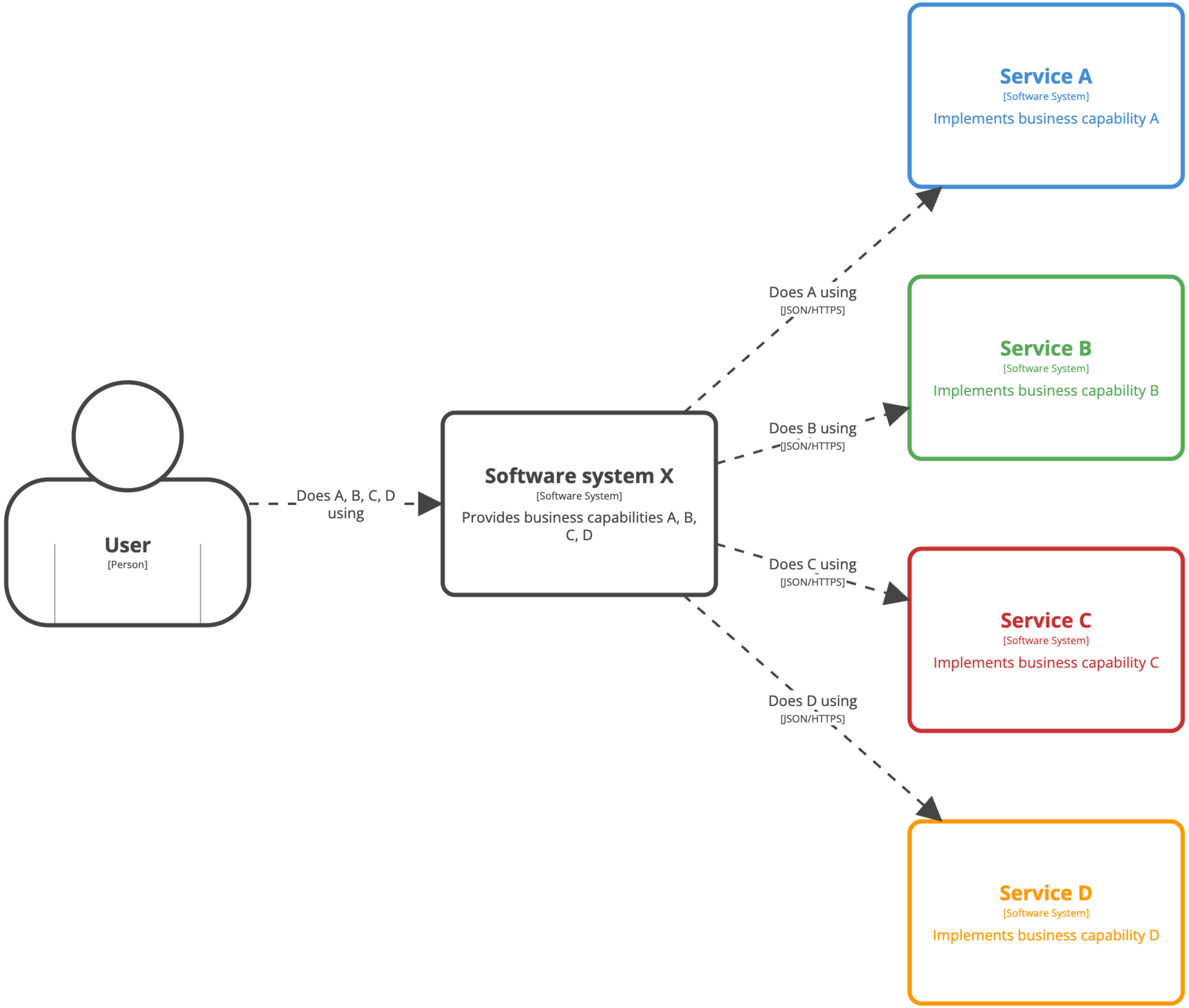

User
[Person]
Does A, B, C using
Web app
[Container: Java and Spring MVC]
Implements UI for capabilities A, B, C
Does A using
[JSON/HTTPS]
Does B using
[JSON/HTTPS]
Does C using
[JSON/HTTPS]
Service A API
[Container: Spring Boot]
Implements business logic for capability A
Service B API
[Container: Spring 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]


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
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
Software system X
[Software System]

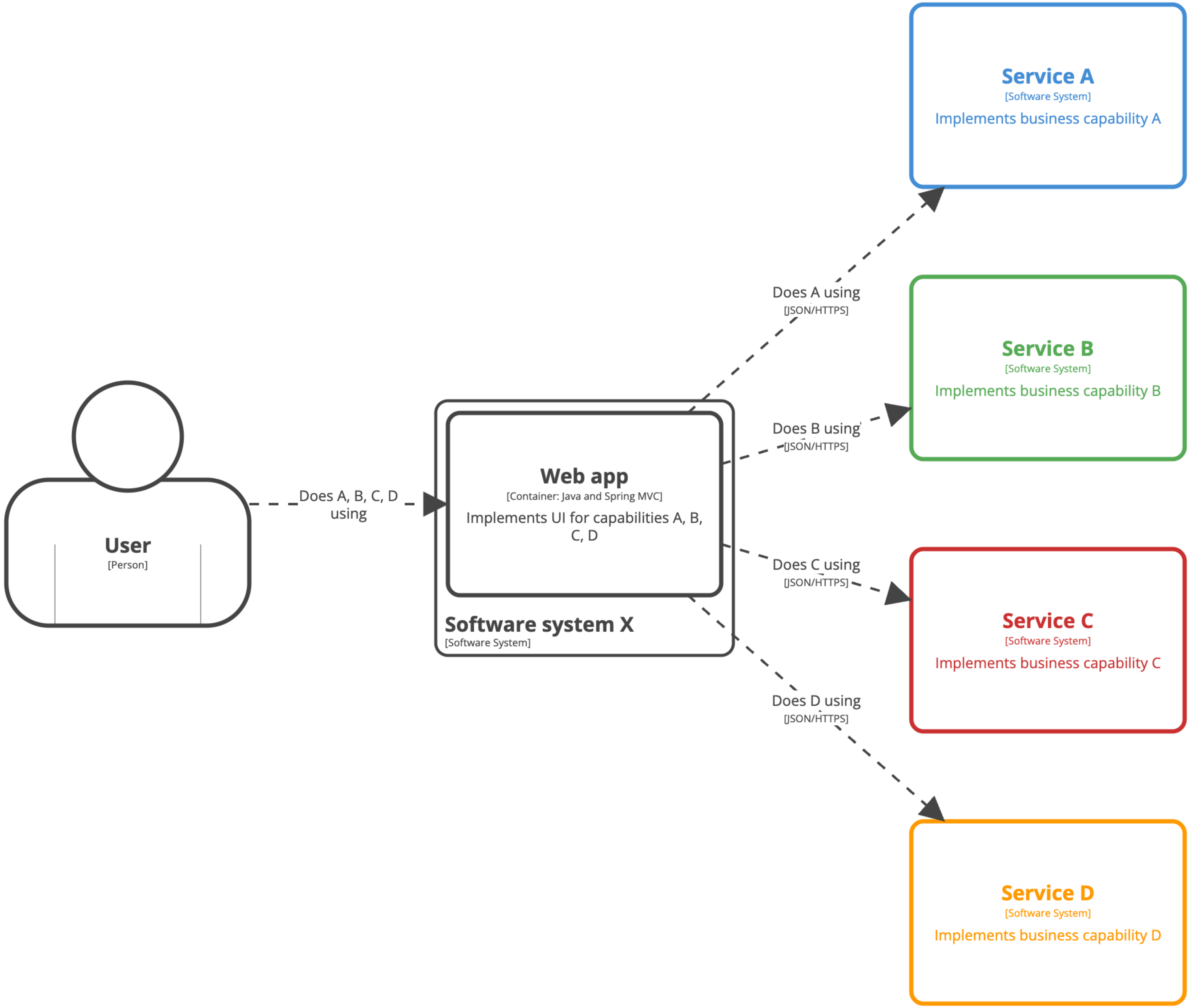

User
[Person]
Does A, B, C, D using
Web app
[Container: Java and Spring MVC]
Implements UI for capabilities A, B, C, D
Does A using
[JSON/HTTPS]
Does B using
[JSON/HTTPS]
Does C using
[JSON/HTTPS]
Does D using
[JSON/HTTPS]
Service A API
[Container: Spring Boot]
Implements business logic for capability 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]

Stage 3: 💵 💵 💵

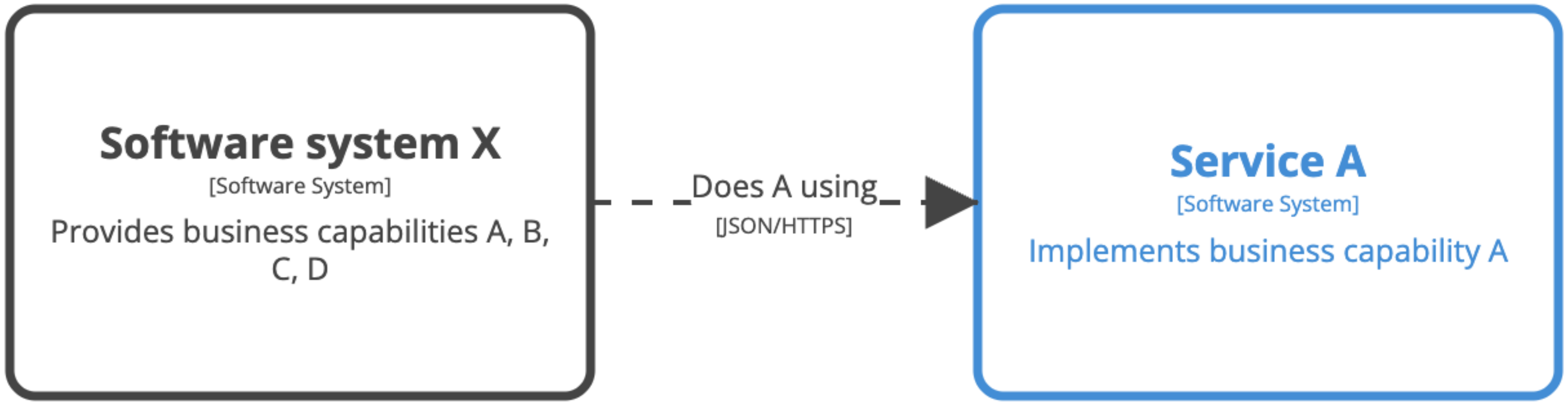
(Conway’s Law)



System Context View: Software system X



Container View: Software system X



Software system X

[Software System]

Provides business capabilities A, B, C, D

Does A using

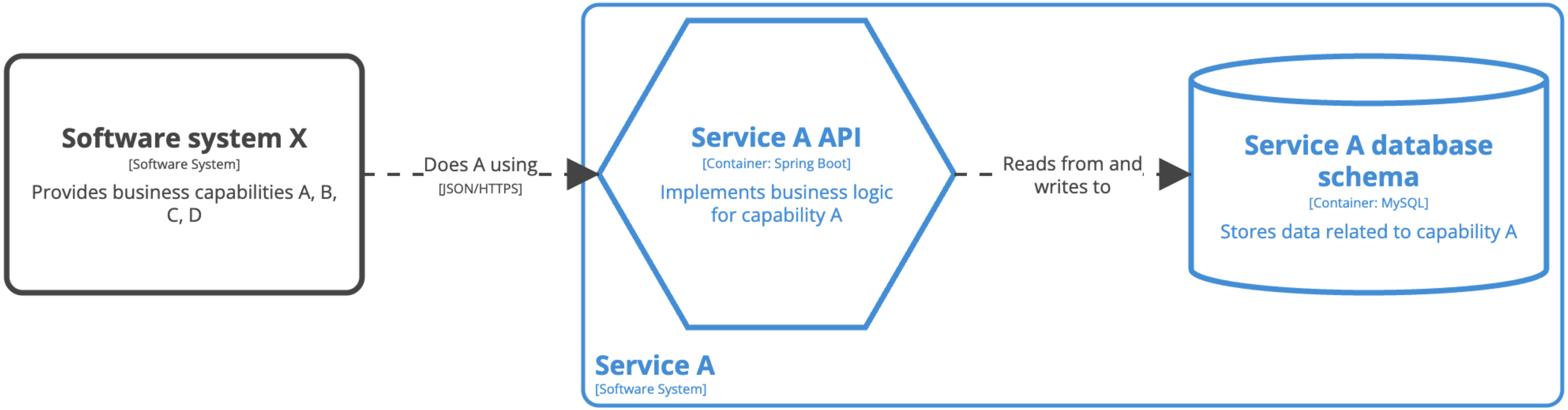
[JSON/HTTPS]

Service A

[Software System]

Implements business capability A

System Context View: Service A



Software system X
[Software System]
Provides business capabilities A, B, C, D

Does A 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
[Software System]

Container View: Service A

Dependencies to “external” containers

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 for capability C
Reads from and writes to
Service C database
[Container: MySQL]
Stores data related to capability C

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

Tooling?

Documenting software architecture

Working software

over

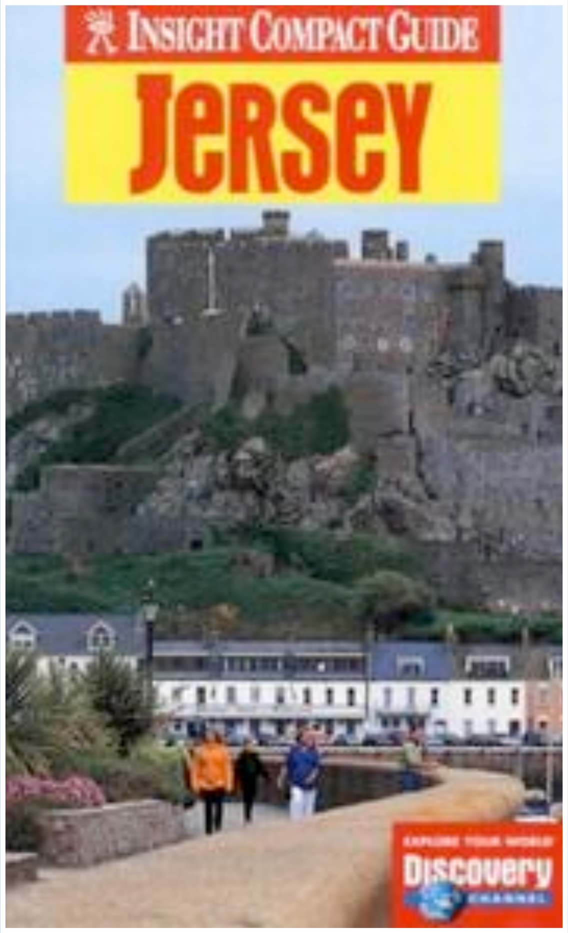
comprehensive documentation

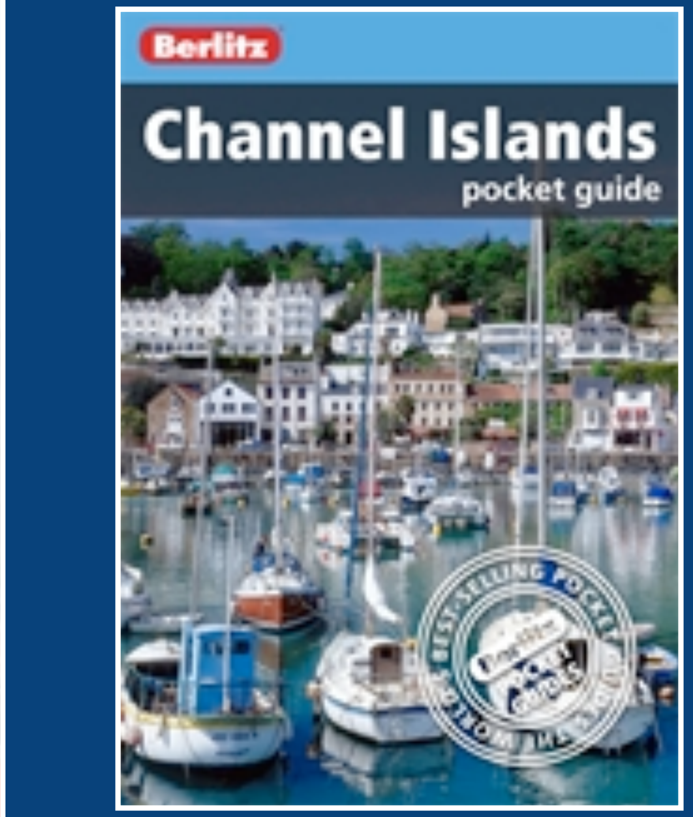
Manifesto for Agile Software Development

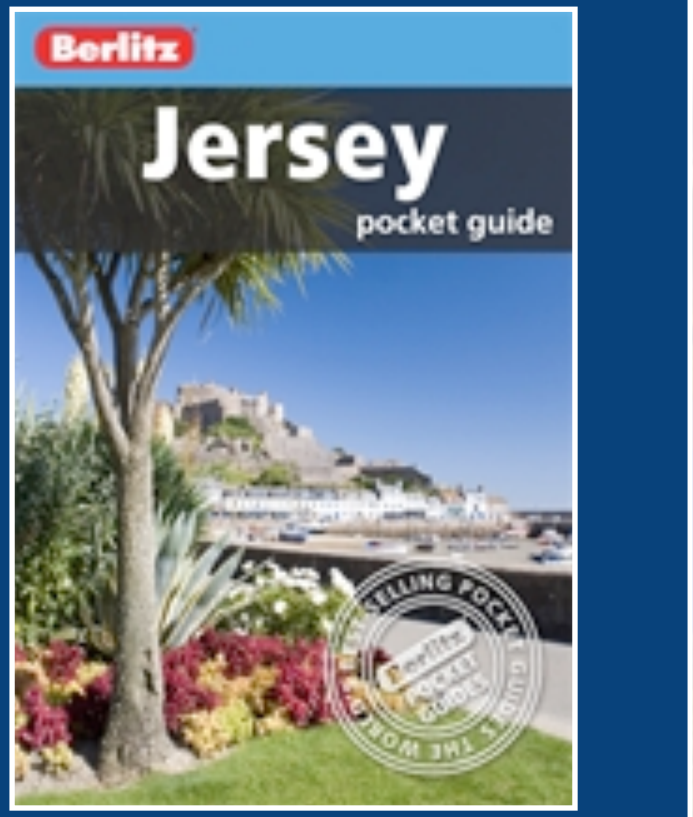
The code doesn’t tell the whole story

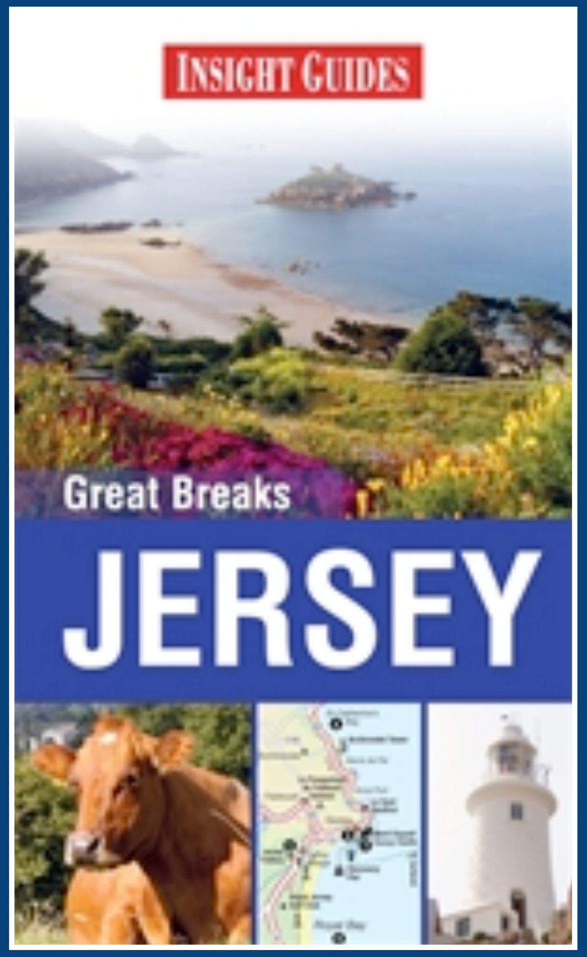
Software Architecture Document

Useful information spread across hundreds of pages; rarely read or updated



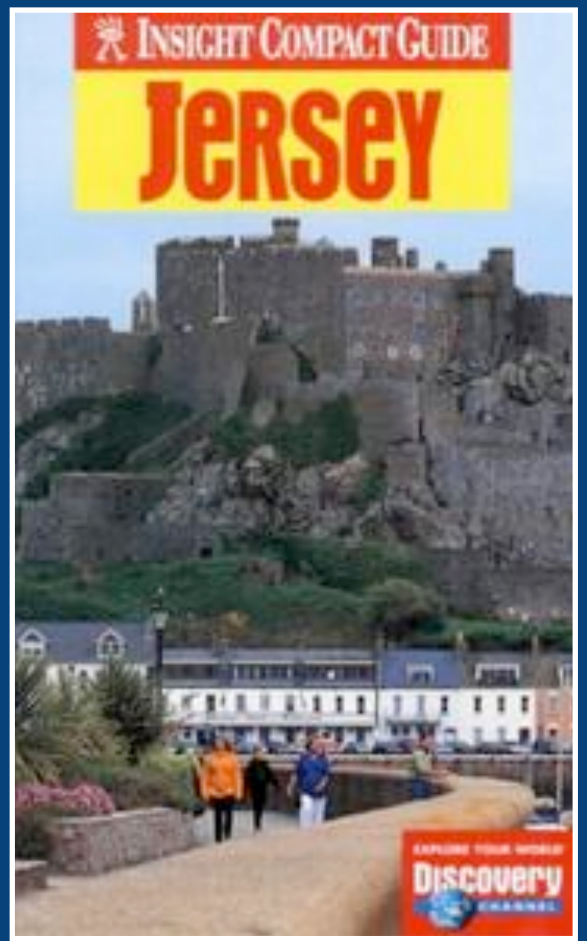


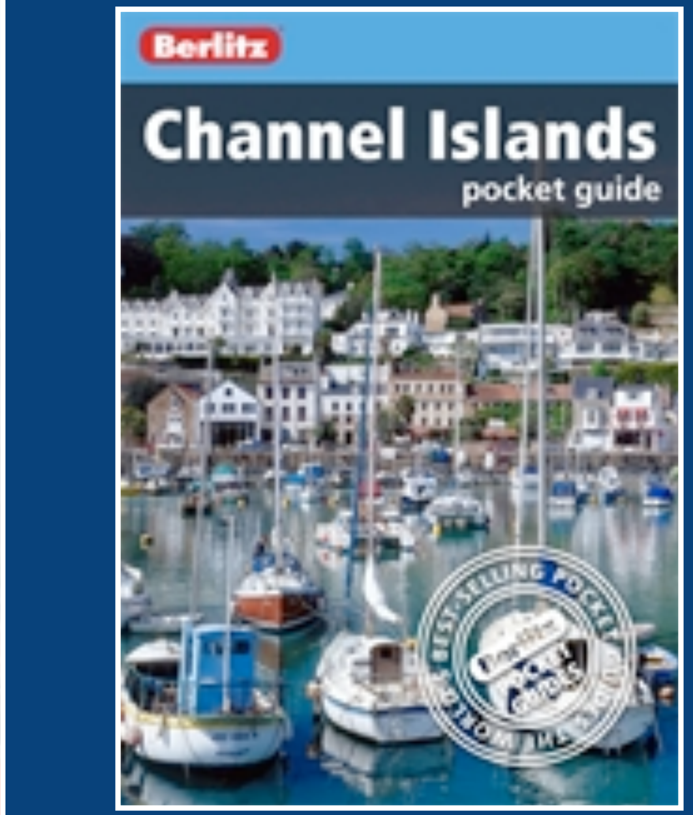


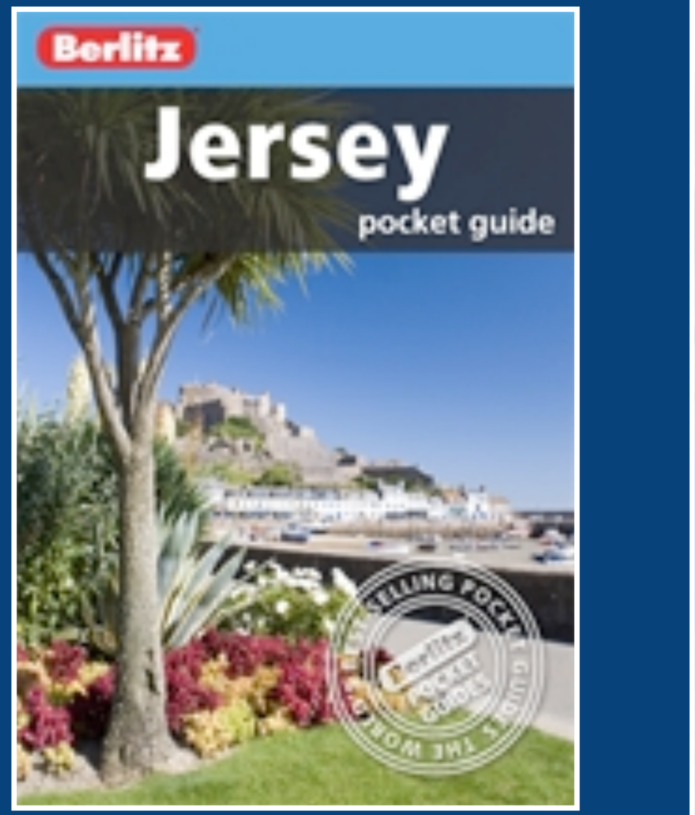


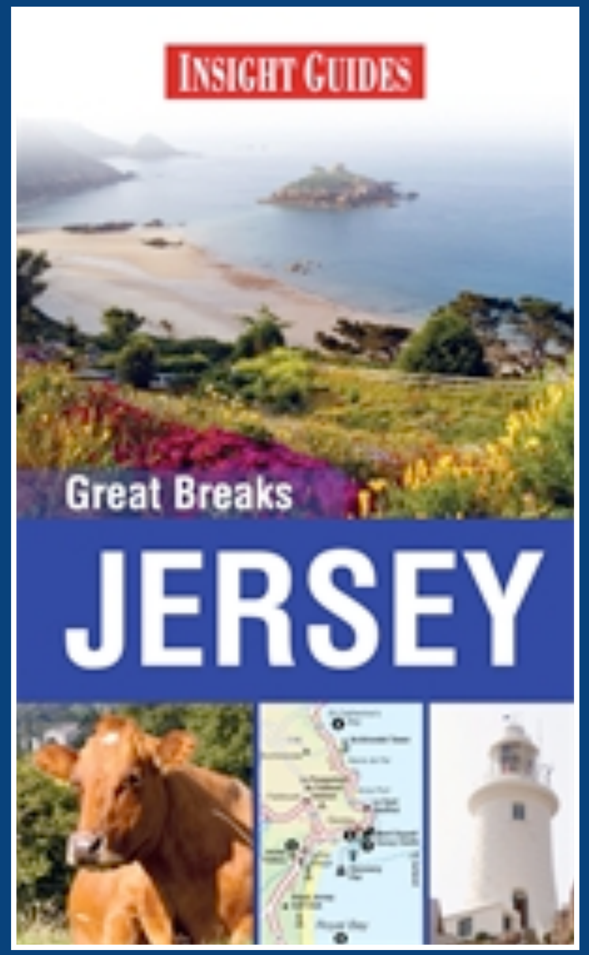
Travel Guidebook

(maps, points of interest, sights, itineraries, history, culture, practical information, etc)



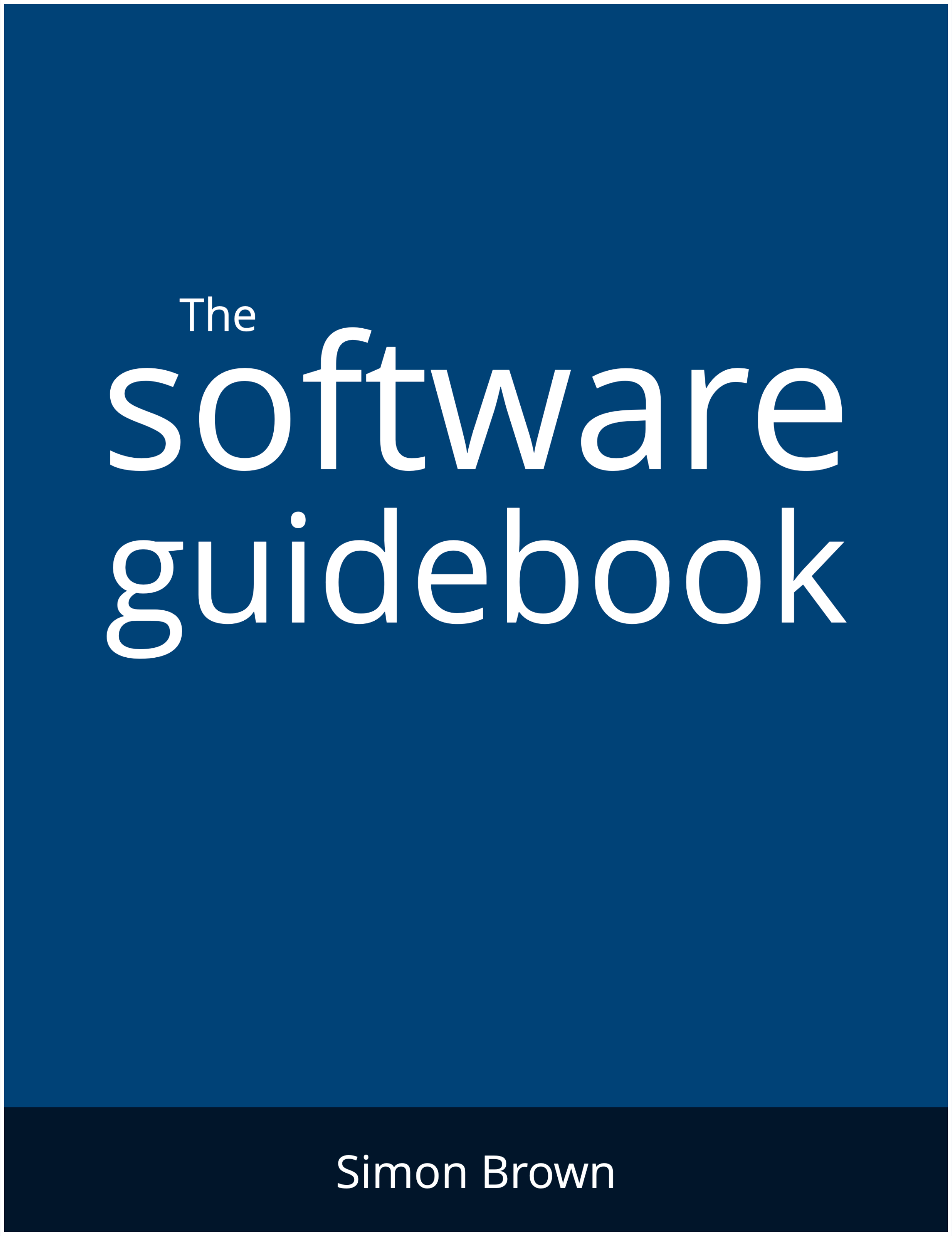






Software Guidebook

(maps, points of interest, sights, itineraries, history, culture, practical information, etc)



The
software
guidebook

Simon Brown

https://leanpub.com/documenting-software-architecture/c/free

The scope is a single **software system**

Describe what you

can’t get from the code

Documentation should be **constantly evolving**

Context

A system context diagram, plus some narrative text to “set the scene”.

Functional Overview

An overview of the software system; perhaps including wireframes, UI mockups, screenshots, workflow diagrams, business process diagrams, etc.

Quality Attributes

A list of the quality attributes (non-functional requirements; e.g. performance, scalability, security, etc).

Constraints

A list of the environmental constraints (e.g. timescales, budget, technology, team size/skills, etc).

Principles

A list of the development and architecture principles (e.g. coding conventions, separation of concerns, patterns, etc).

Software Architecture

A description of the software architecture, including static structure (e.g. containers and components) and dynamic/runtime behaviour.

Code

A description of important or complicated component implementation details, patterns, frameworks, etc.

Data

Data models, entity relationship diagrams, security, data volumes, archiving strategies, backup strategies, etc.

This is a **starting point**; add and remove sections as necessary.

Infrastructure Architecture

A description of the infrastructure available to run the software system.

Deployment

The mapping of software (e.g. containers) to infrastructure.

Development Environment

A description of how a new developer gets started.

Operation and Support

An overview of how the software system is operated, supported, monitored, etc.

Decision Log

A log of the major decisions made; e.g. as free format text or a collection of “Architecture Decision Records”.

arc42
Overview
Docs
FAQ
Examples
Training
Download
About

arc42 Template Overview

arc42 is a template for architecture communication and documentation.

Photo credit: unsplash

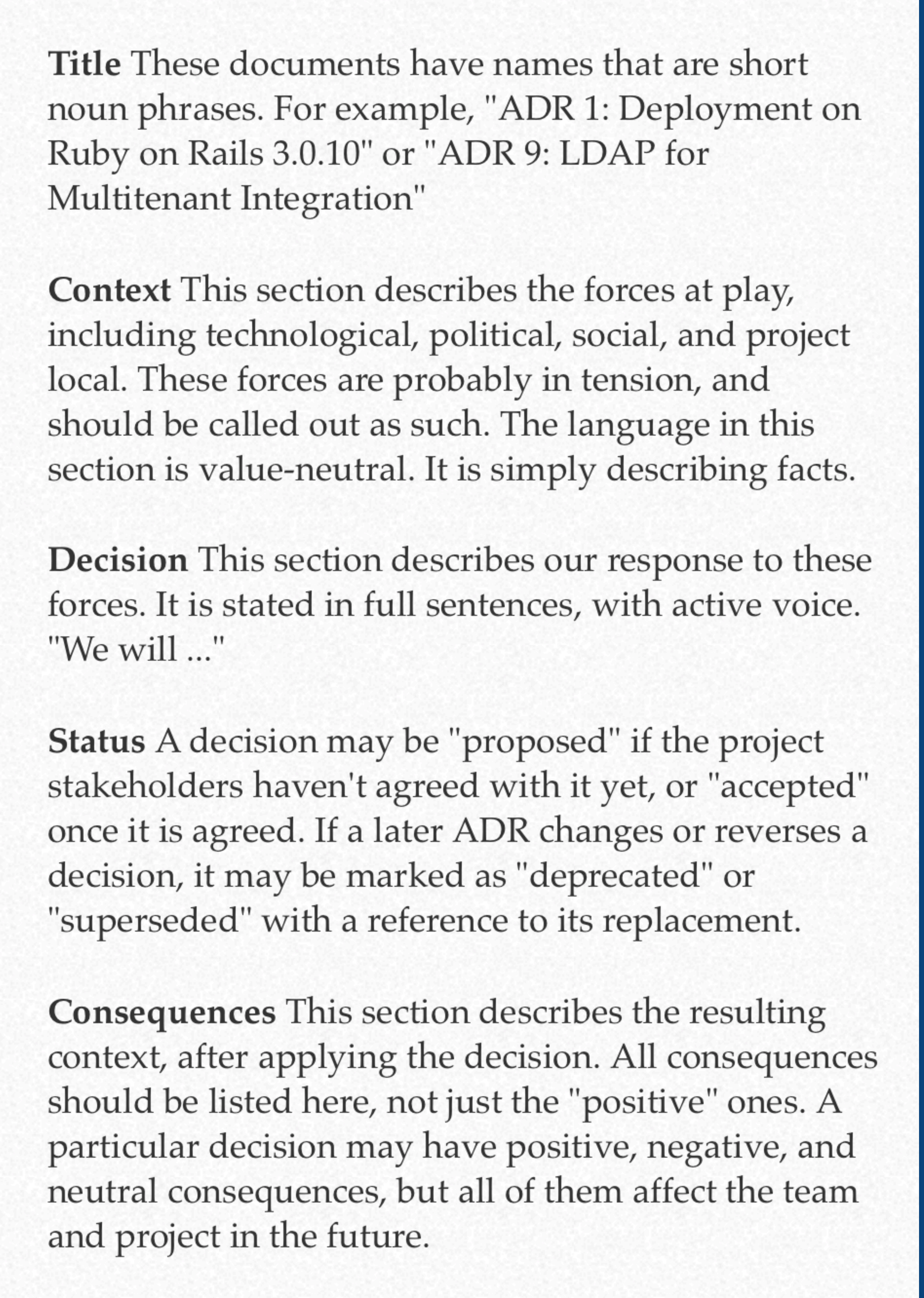
arc42 answers the following two questions in a pragmatic way, but can be tailored to your specific needs:

- *What* should we document/communicate about our architecture?
- *How* should we document/communicate?

Introduction and Goals
Required Functions
Quality Goals
Goal
Description
...
...
Stakeholder
Who?
Expectation?
...
...

1. Introduction and Goals

Short description of the **requirements**, driving forces, extract (or abstract) of requirements. Top three (max five) **quality goals** for the architecture which have highest priority for the major stakeholders. A table of important **stakeholders** with their expectation regarding architecture.



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

http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions (Michael Nygard)

Immutable vs mutable ADRs?

Documentation format?

Microsoft Word, Microsoft SharePoint,
Atlassian Confluence, Markdown or AsciiDoc, etc

How long?

Something I can read in 1-2 hours;

a good starting point for exploring the code

How do you keep software architecture documentation up to date?

C4 model diagrams

+

software guidebook/arc42

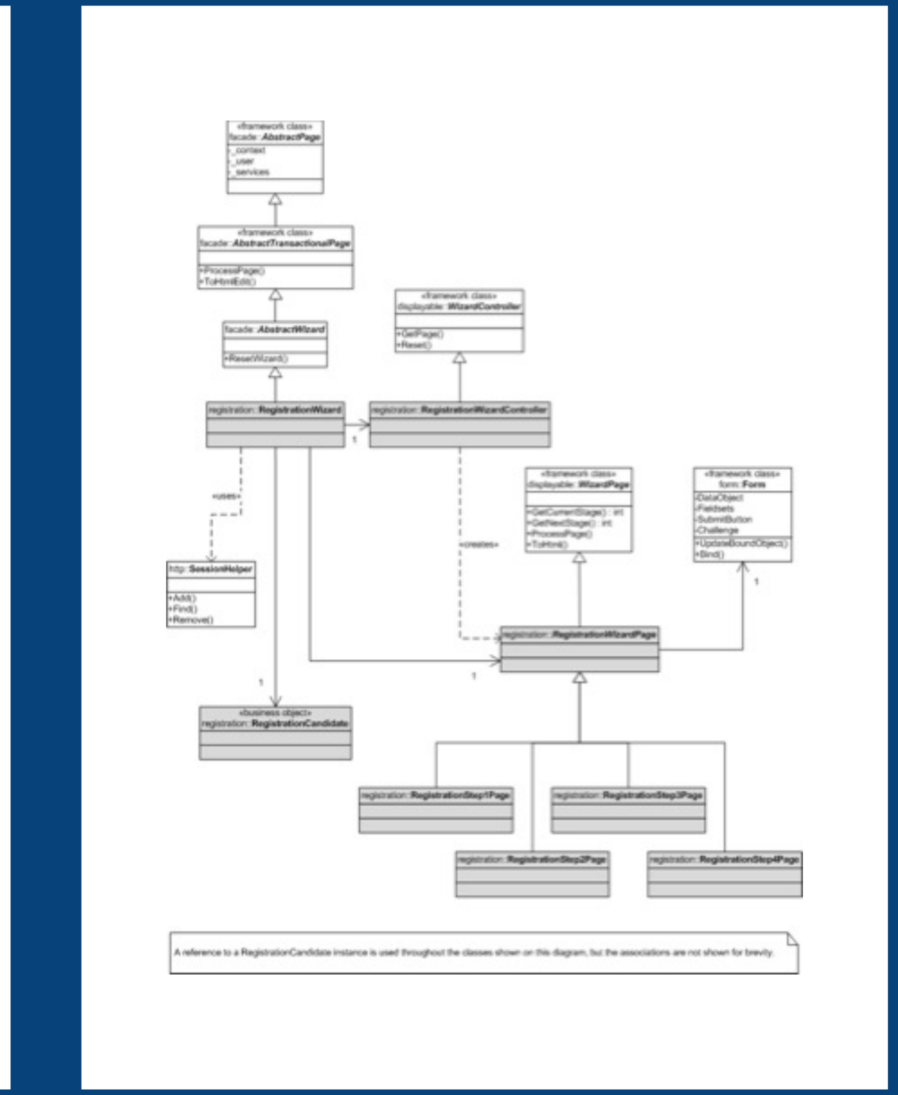
+

architecture decision records

Software architecture in practice

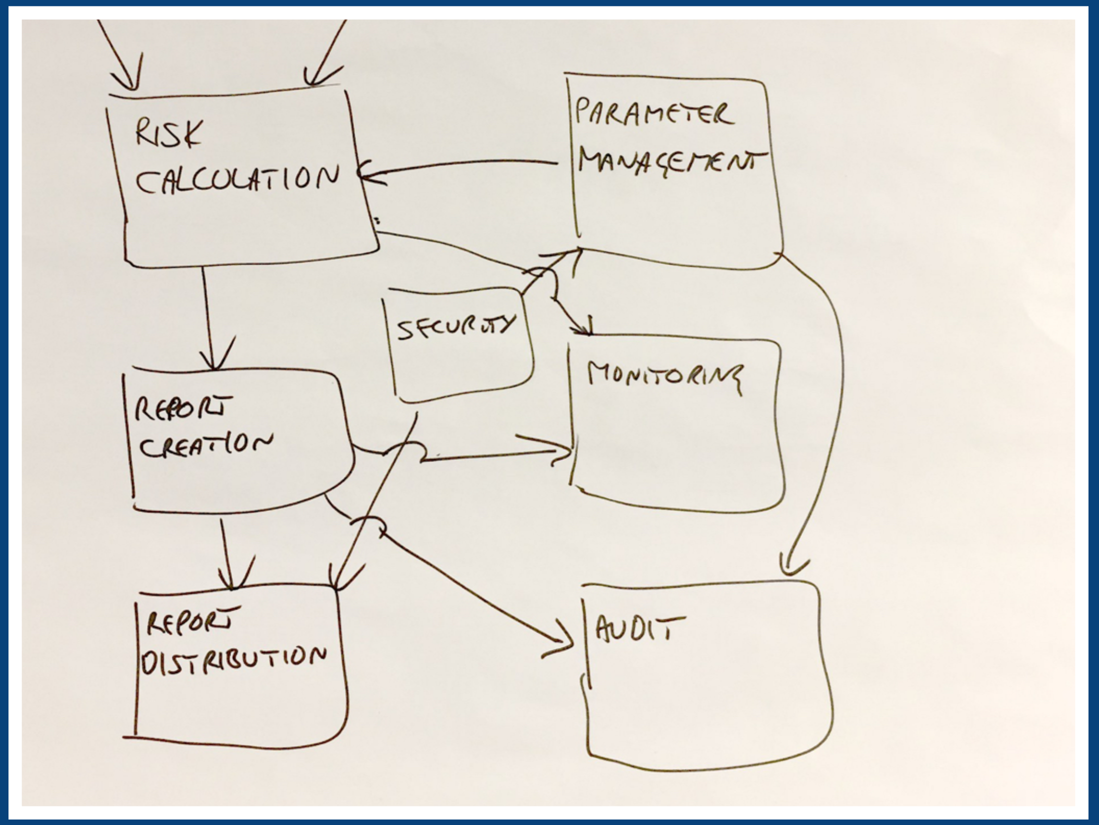
Big design up front

Software Architecture Document



A reference to a RegistrationCandidate instance is used throughout the classes shown on this diagram, but the associations are not shown for brevity.

vs



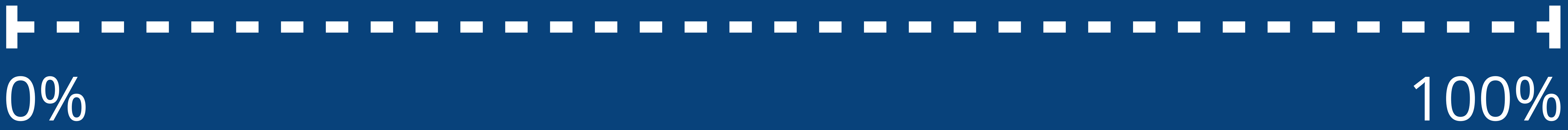
No design up front

Big design up front is dumb.
Doing no design up front
is even dumber.

Dave Thomas

Evolutionary architecture

How much **up front design** should you do?



0%

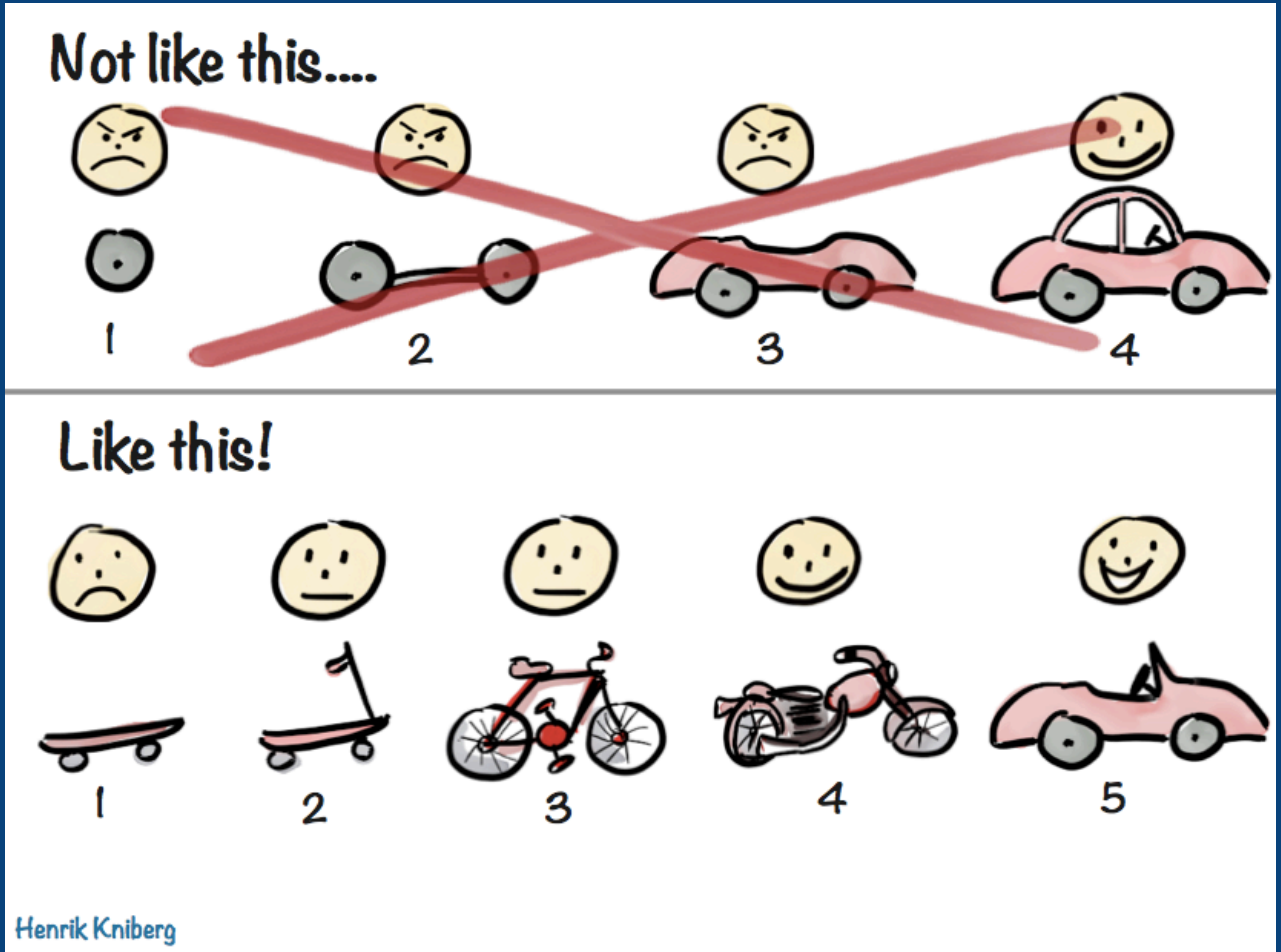
100%

it depends

Sometimes requirements are known, and sometimes they aren’t

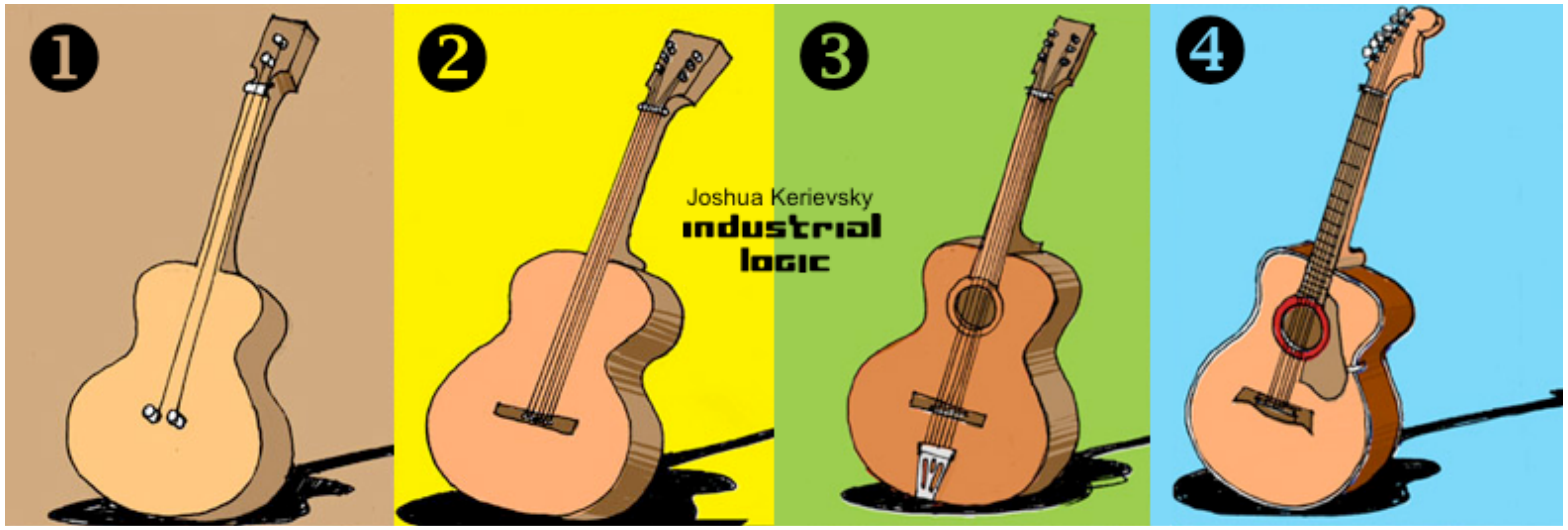
(enterprise software development vs product companies and startups) just enough

Up front design is not necessarily about creating a perfect end-state or complete architecture



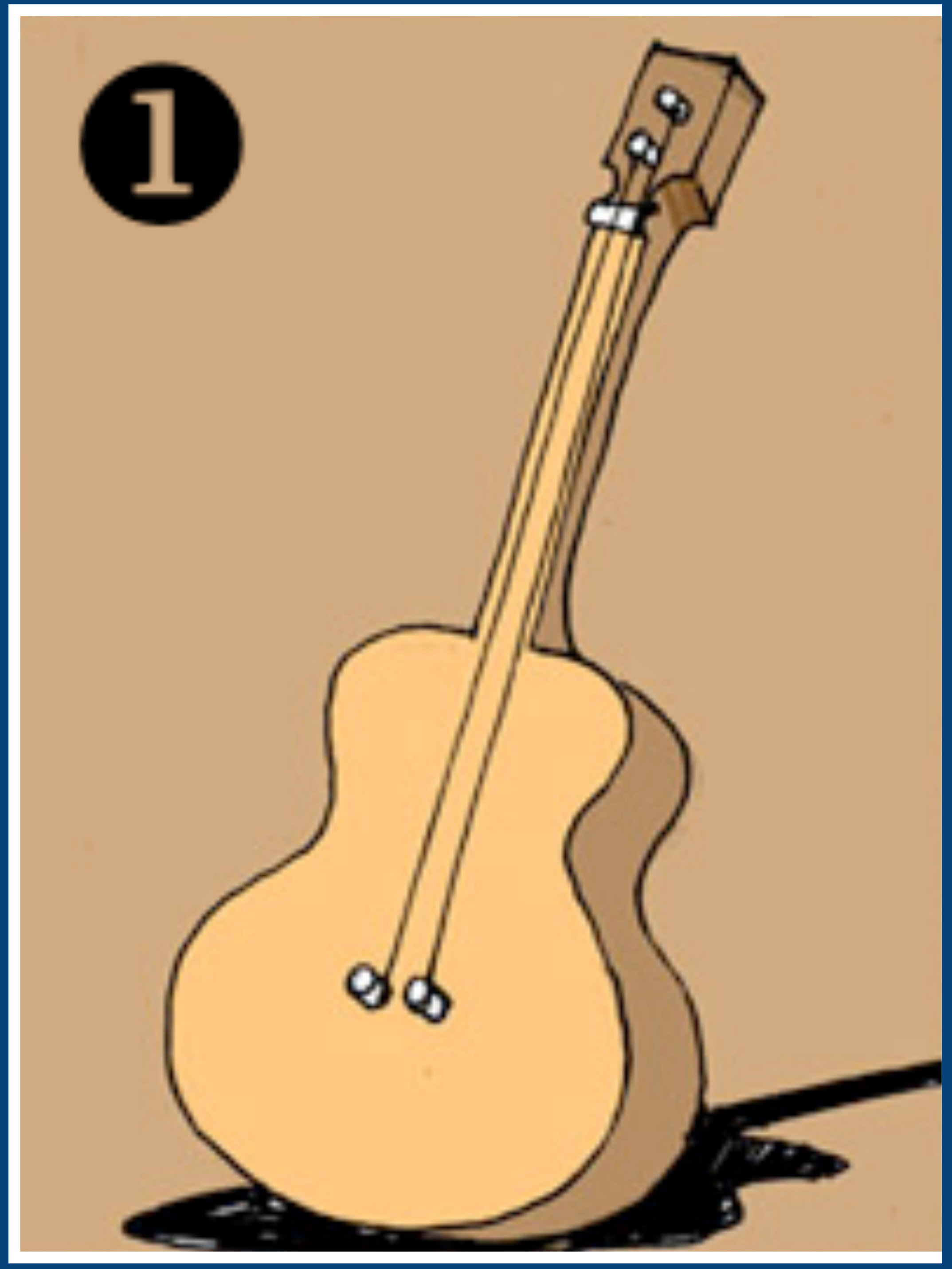
Iteration (via prototyping and experimentation) is great for product design but...

you don’t just “build the car”



Evolutionary Design

Beginning With A Primitive Whole



Evolutionary Design

Beginning With A Primitive Whole

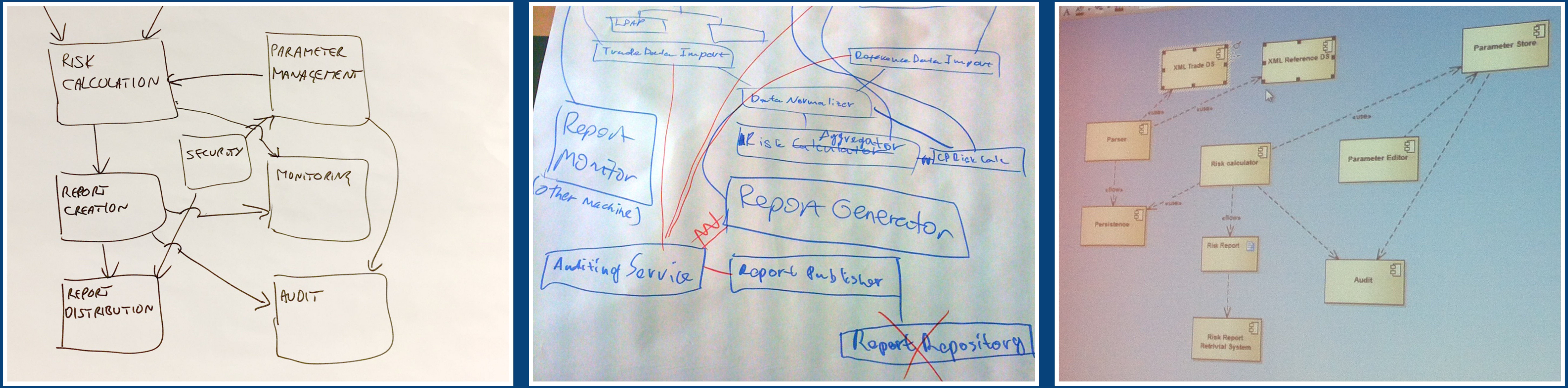
We’re not trying to make every decision

I think there is a role for a broad starting point architecture. Such things as stating early on how to layer the application, how you'll interact with the database (if you need one), what approach to use to handle the web server.

Martin Fowler

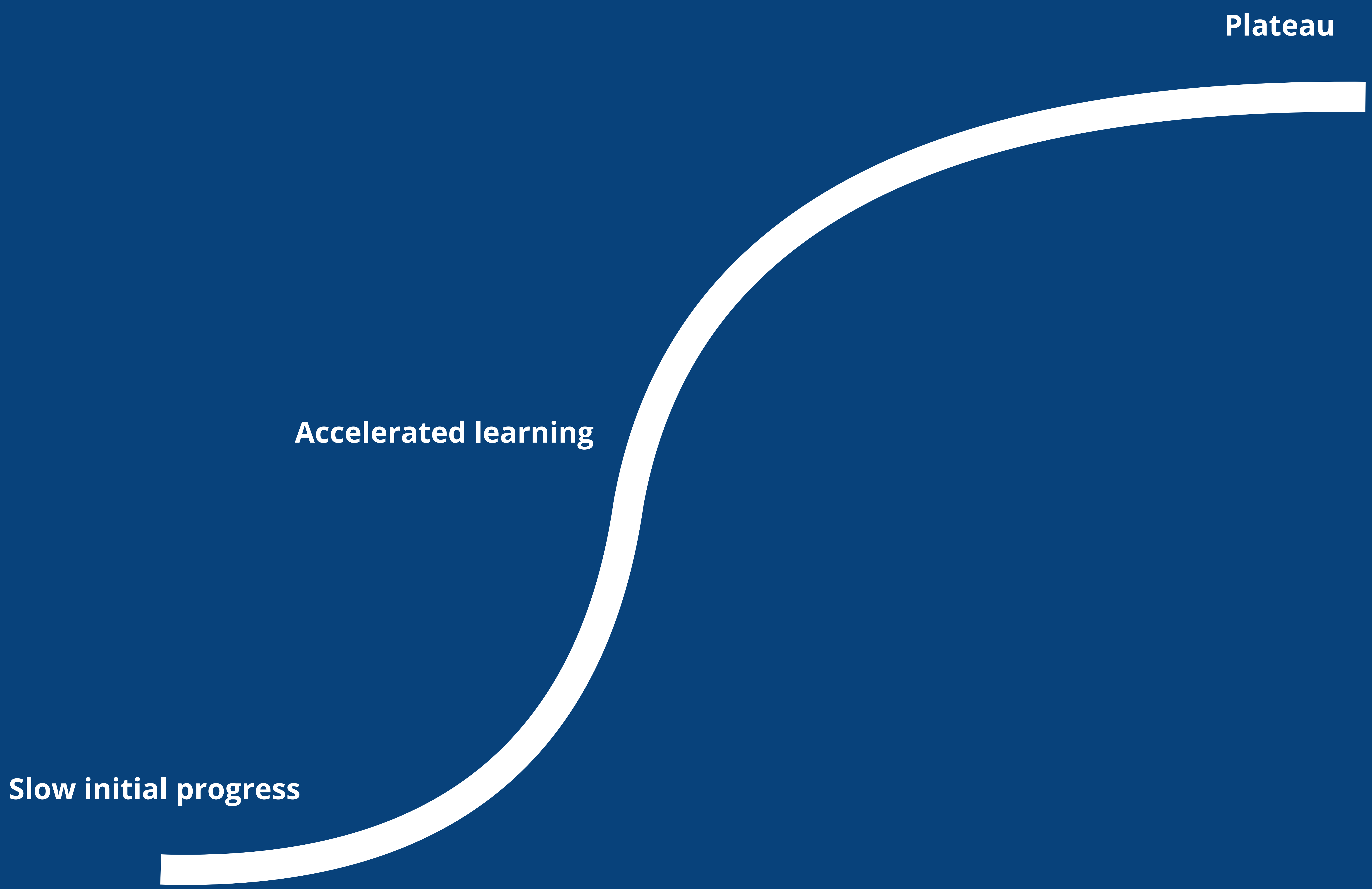
https://martinfowler.com/articles/designDead.html

A **starting point** adds value



If you don’t **engage** in the problem, you end up with a very simplified and superficial view of the solution

Part of the design activity is about discovering “unknown unknowns”



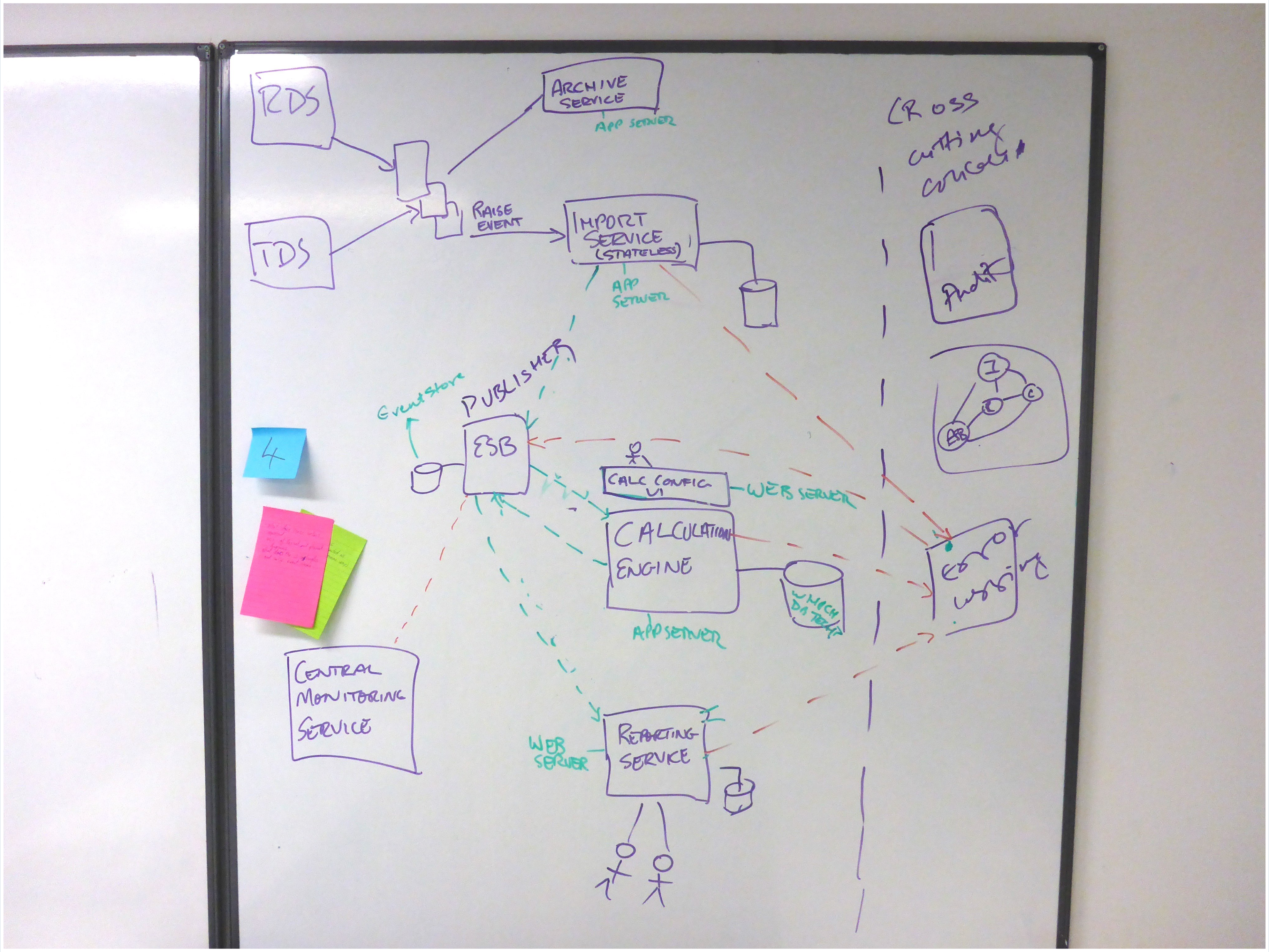
Plateau

Accelerated learning

Slow initial progress

The typical s-curve of learning

1. Is that what we're going to **build**?



2. Is it going to **work**?

Diagrams are a visual checklist for design decisions

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

USER
INTERACTION
PRE-EXISTING SYSTEM
PROPOSED SYSTEM
Business Users
(PERSON)
A CUSTOMER THAT ACCESSES RISK REPORTS
Business Users with Configuration Access
(PERSON)
A USER THAT CAN MODIFY RISK CALCULATION PARAMS
DELIVERS LINK TO REPORT TO
ACCESS REPORTS USING
CHANGES CONFIGS FOR
Email
(SOFTWARE SYSTEM)
SENDS EMAILS TO BUSINESS USERS
Financial Risk System
(SOFTWARE SYSTEM)
TAKES TRADE DATA AND REFERENCE DATA AND CALCULATES RISK; GENERATES RISK REPORT AND MAKES IT AVAILABLE TO BUSINESS USERS
Auth
(SOFTWARE SYSTEM)
AUTHENTICATES BUSINESS USERS
SENDS REPORT LINKS THROUGH
MONITORED BY
USES DATA FROM
USE DATA FROM
AUTHENTICATES AND CHECKS PERMISSIONS USING
Central Monitoring System
(SOFTWARE SYSTEM)
MONITORS HEALTH OF ALL SERVICES
Trade Data System
(SOFTWARE SYSTEM)
PROVIDES DATA ABOUT ALL TRADES BY THE BANK
Reference Data System
(SOFTWARE SYSTEM)
PROVIDES REFERENCE DATA ABOUT COUNTERPARTIES

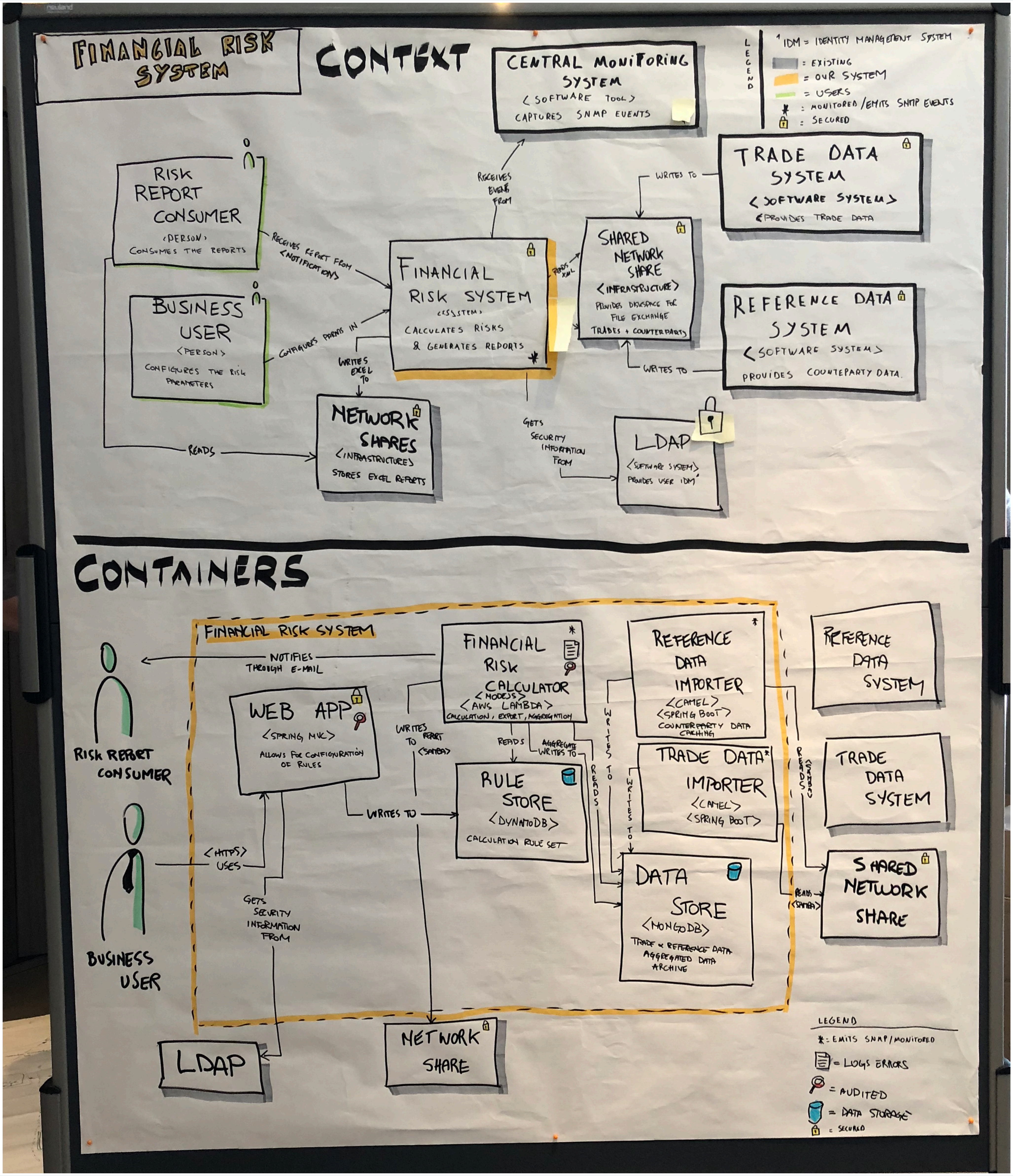
Container diagram

What are the major technology building blocks?

What are their responsibilities?

How do they communicate?

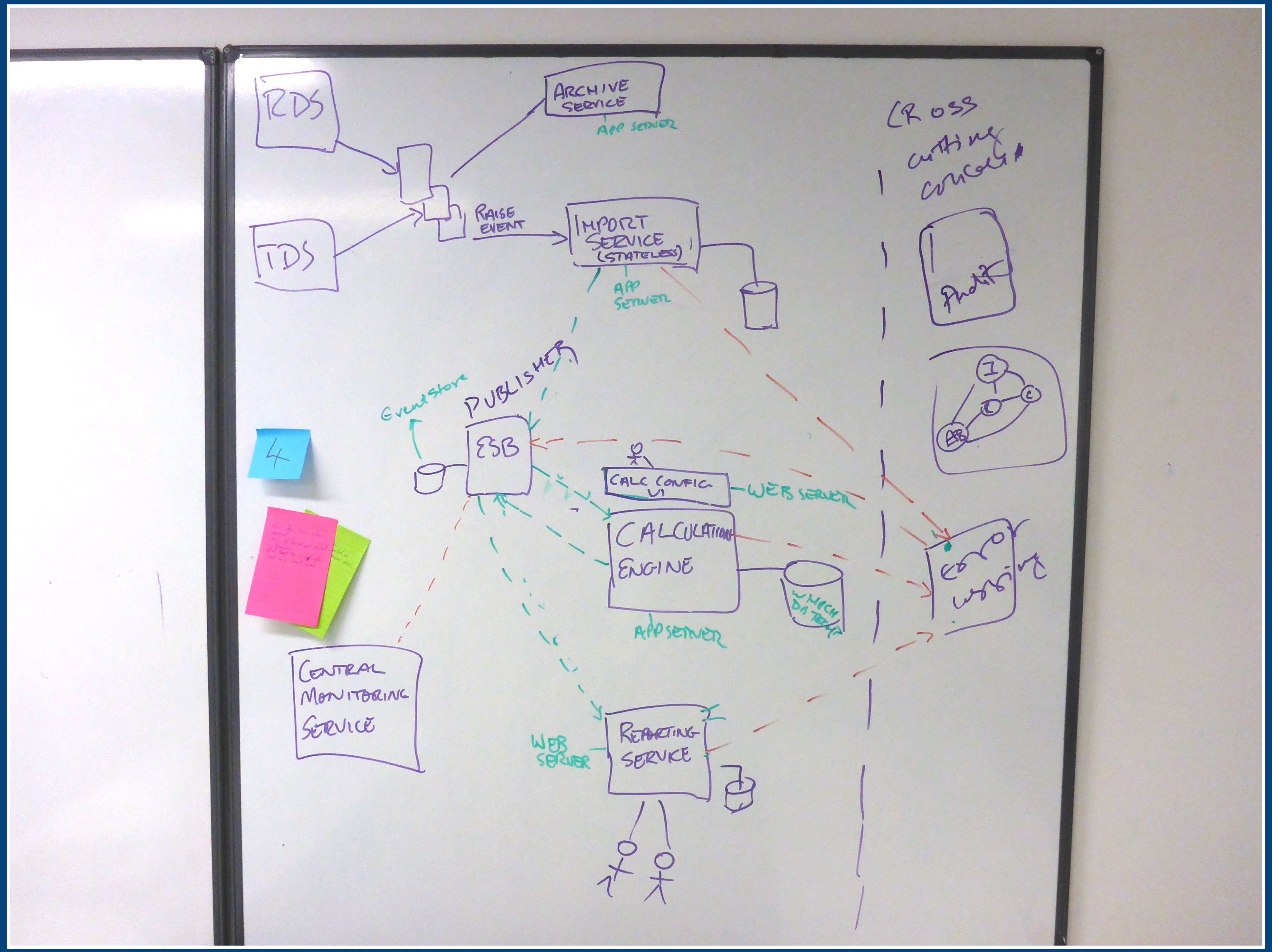
Financial Risk System: Container Diagram
USER
INTERACTION
CONTAINER
RISK SERVICE BOUNDARY
INDICATES THAT A COMPONENT IS MONITORED BY THE CENTRAL MONITORING SYSTEM
DATA STORE
Business User (PERSON)
A CUSTOMER THAT ACCESSES REPORTS
Business User with Config Access (PERSON)
A USER THAT CAN MODIFY CALCULATION PARAMS
ACCESS REPORTS USING
REQUESTS WEB APP (HTTPS)
REQUESTS WEB APP (HTTPS)
MODIFY CONFIGS USING
Risk Report Web App (JAVASCRIPT REACT)
PROVIDES USER INTERFACE TO ACCESS RISKS REPORTS
Web Server (JAVA SPRING)
SERVES SPAS AND PROVIDES API ENDPOINTS TO THE WEB APPLICATIONS
Configuration Web App (JAVASCRIPT REACT)
PROVIDES A SUBSET OF USERS AN INTERFACE TO CONFIGURE RISK CALCULATION PARAMS
ACCESS REPORTS USING (REST JSON)
MODIFIES CONFIGS USING (REST JSON)
WRITES CONFIGS TO
RETRIEVES REPORTS FROM
AUTHENTICATES USERS USING (OAUTH)
Auth (SOFTWARE SYSTEM)
AUTHENTICATES USERS
Report Store (CIFS NETWORK STORAGE)
STORES GENERATED REPORTS, AND THE INPUT DATA AND CONFIG USED
Input Store (CIFS NETWORK STORAGE)
STORES INPUT TRADE AND REFERENCE DATA AND CALCULATION CONFIGS
Config Store (POSTGRES DB)
STORES CONFIG PARAMS FOR RISK CALCULATIONS
STORES REPORTS TO
TAKES INPUT FROM
STORE INPUT DATA TO
Trade Data System (SOFTWARE SYSTEM)
PROVIDES TRADE DATA
Email (SOFTWARE SYSTEM)
SEND EMAILS TO USERS
SEND LINK TO REPORT USING
"Risk Cruncher" (JAVA COMMANDLINE)
CALCULATES RISK DATA + GENERATES REPORTS AND UPLOADS IT TO REPORT STORE
SCHEDULES JOB FOR
Scheduler (JAVA COMMAND LINE)
SCHEDULES CALCULATION JOBS AND COPIES INPUT DATA TO THE INPUT STORE
COPIES DATA FROM
Reference Data System (SOFTWARE SYSTEM)
PROVIDES REFERENC DATA



Understand the structure and create a shared vision

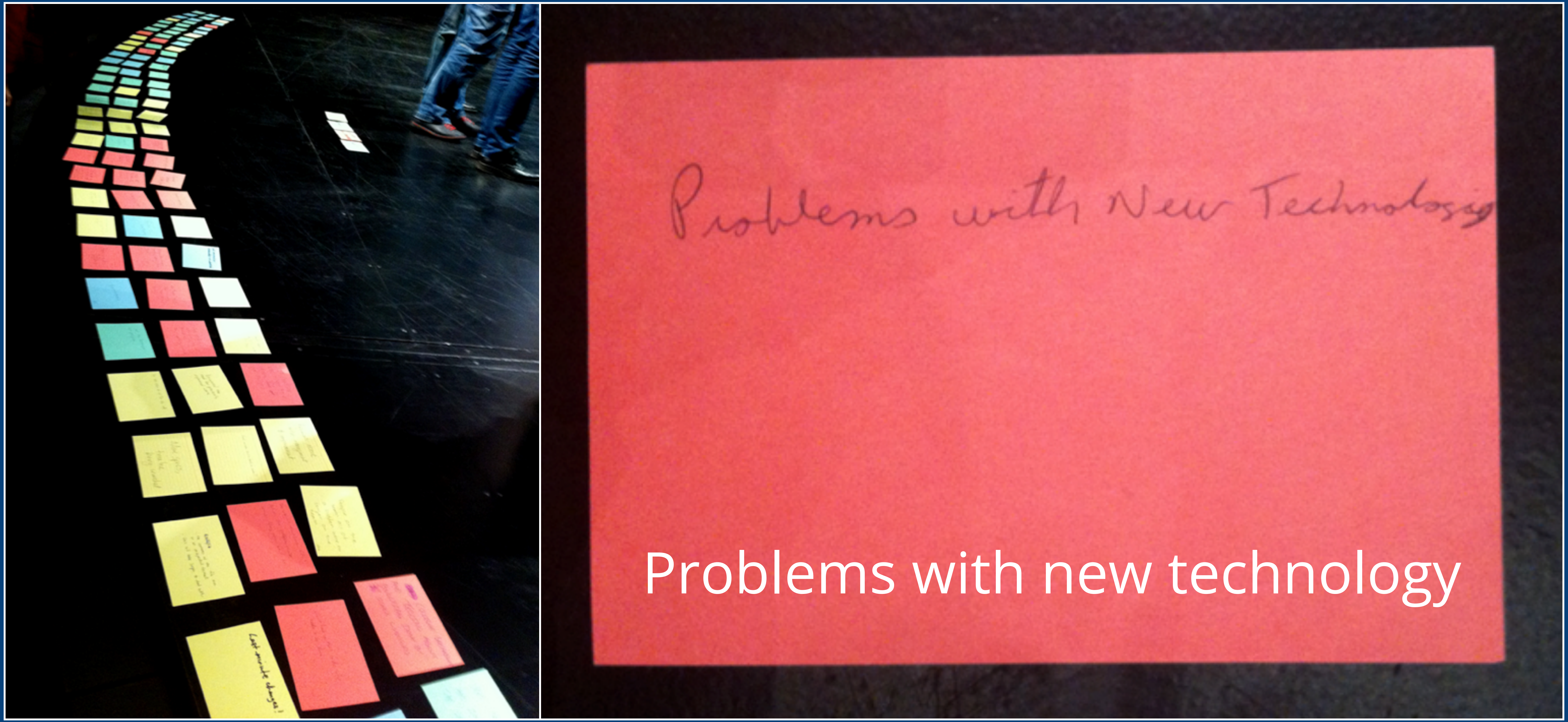
Did the “Financial Risk System”
exercise feel like
big design up front?

1. Is that what we're going to **build**?



2. Is it going to **work**?

Teams need to explicitly manage technical risk



Problems with new technology

An example timeline from “Beyond Retrospectives”
Linda Rising, GOTO Aarhus 2011

Identify and **mitigate** your **highest priority risks**

Probability

		Low 1	Medium 2	High 3
Impact	Low 1	1	2	3
	Medium 2	2	4	6
	High 3	3	6	9

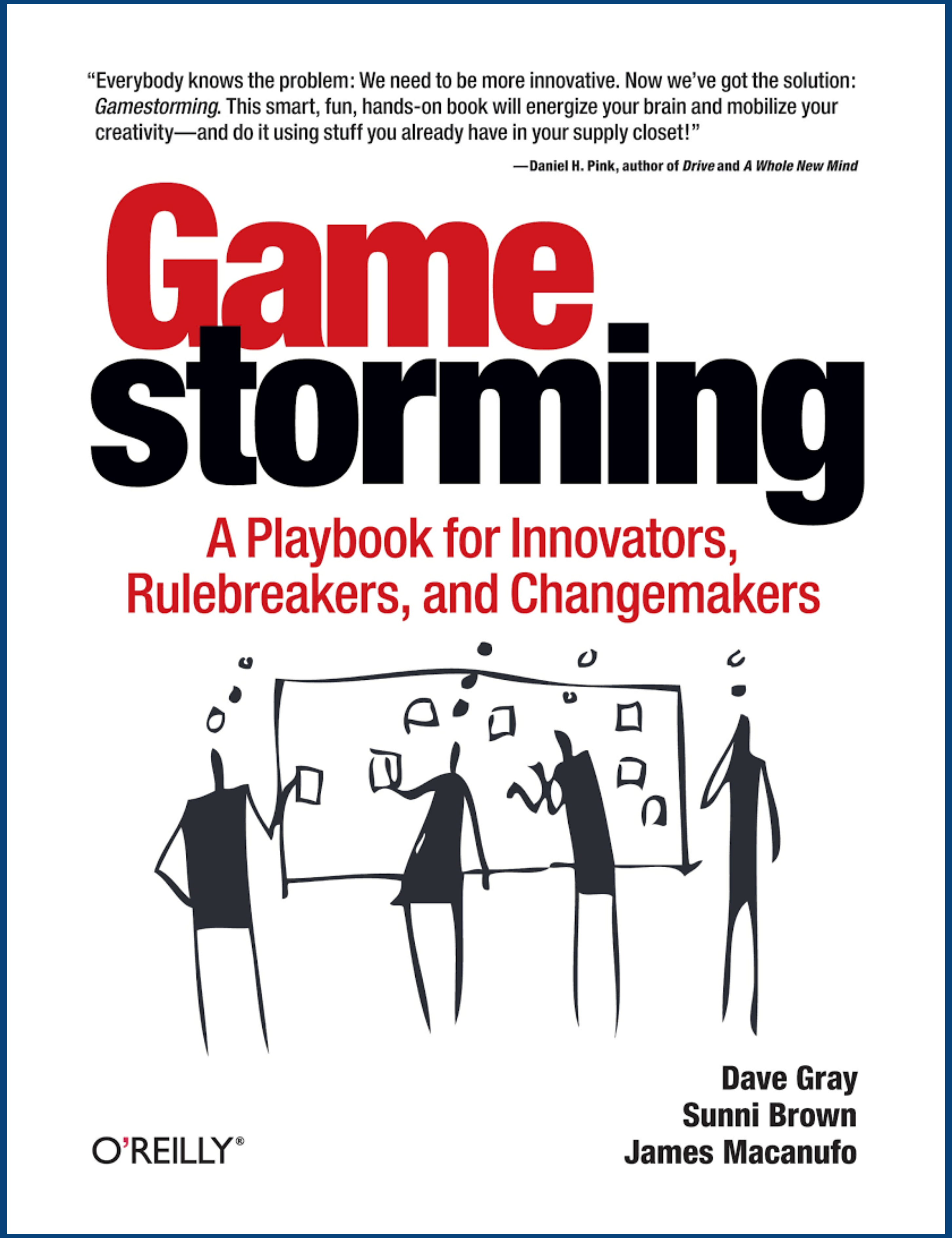
The software architecture role
should own the technical risks

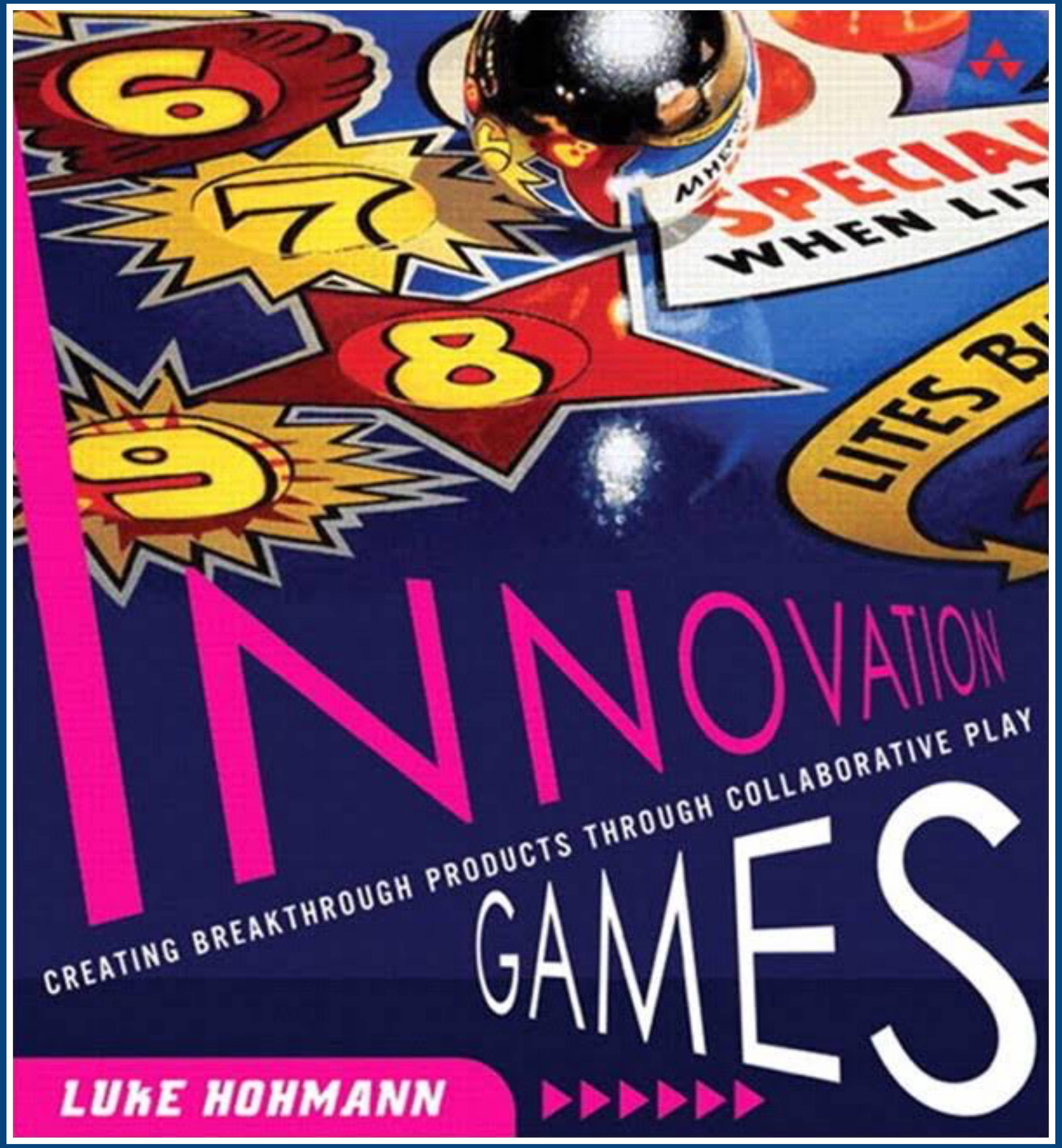
Architecturally significant?

costly to change | complicated | new

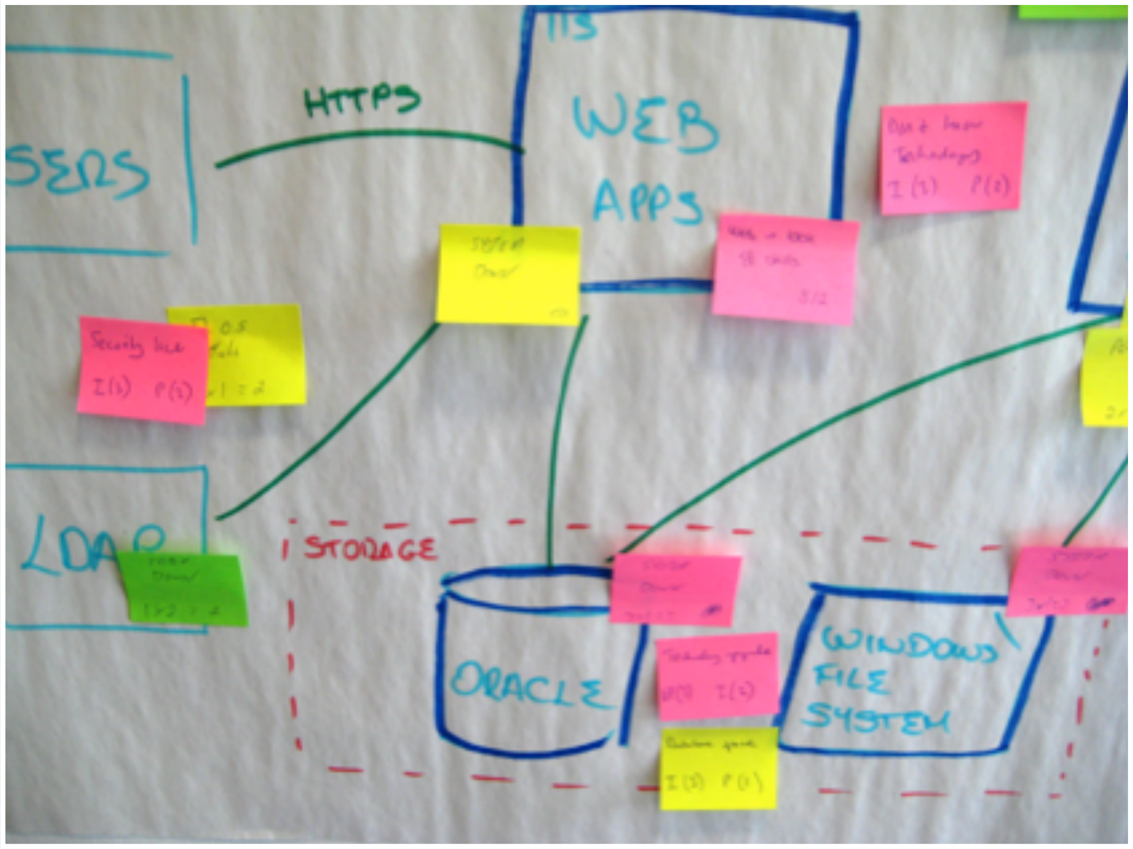
Like estimates,

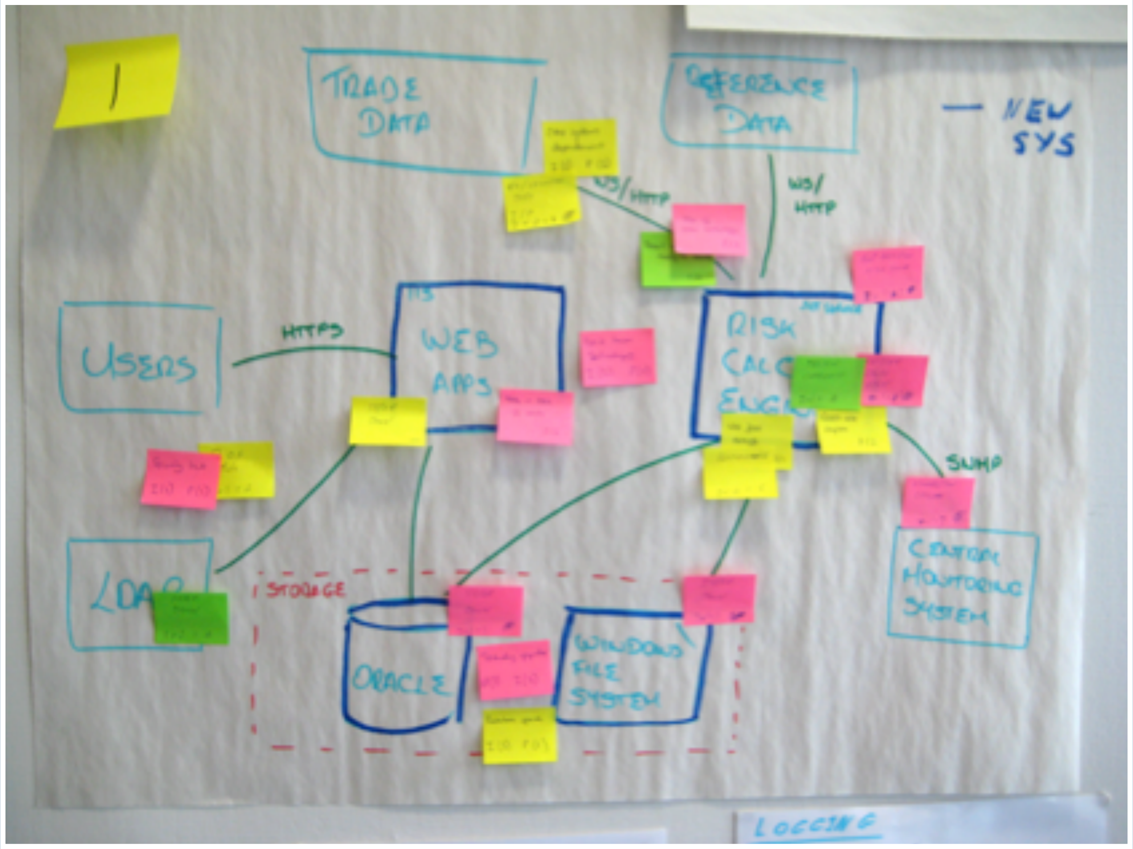
risks are subjective

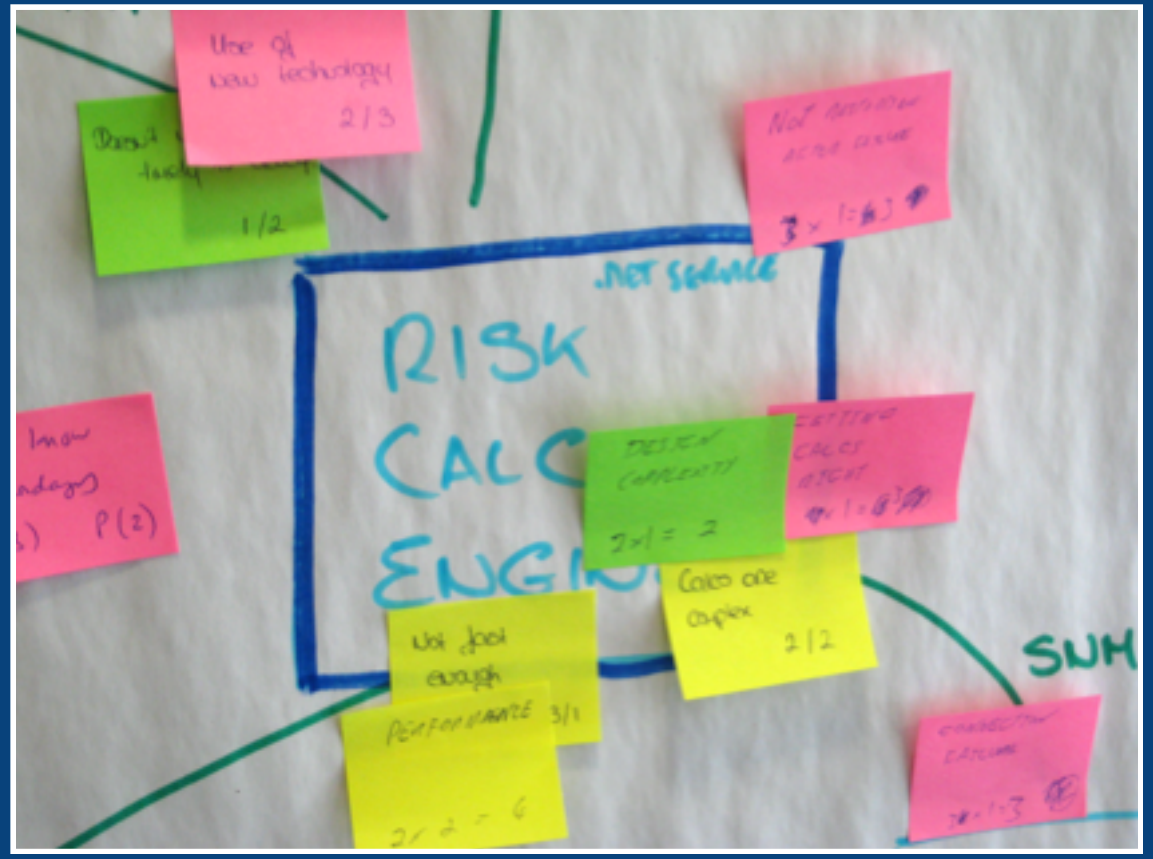

"Everybody knows the problem: We need to be more innovative. Now we've got the solution: Gamestorming. This smart, fun, hands-on book will energize your brain and mobilize your creativity—and do it using stuff you already have in your supply closet!"
—Daniel H. Pink, author of Drive and A Whole New Mind
Game storming
A Playbook for Innovators, Rulebreakers, and Changemakers
Dave Gray
Sunni Brown
James Macanufo
O'REILLY



Visual and collaborative "games"







Risk-storming

A visual and collaborative technique for identifying risk

Threat modelling

(STRIDE, LINDDUN, Attack Trees, etc)

Base your architecture on requirements, travel light and prove your architecture with concrete experiments.

Agile Architecture: Strategies for Scaling Agile Development

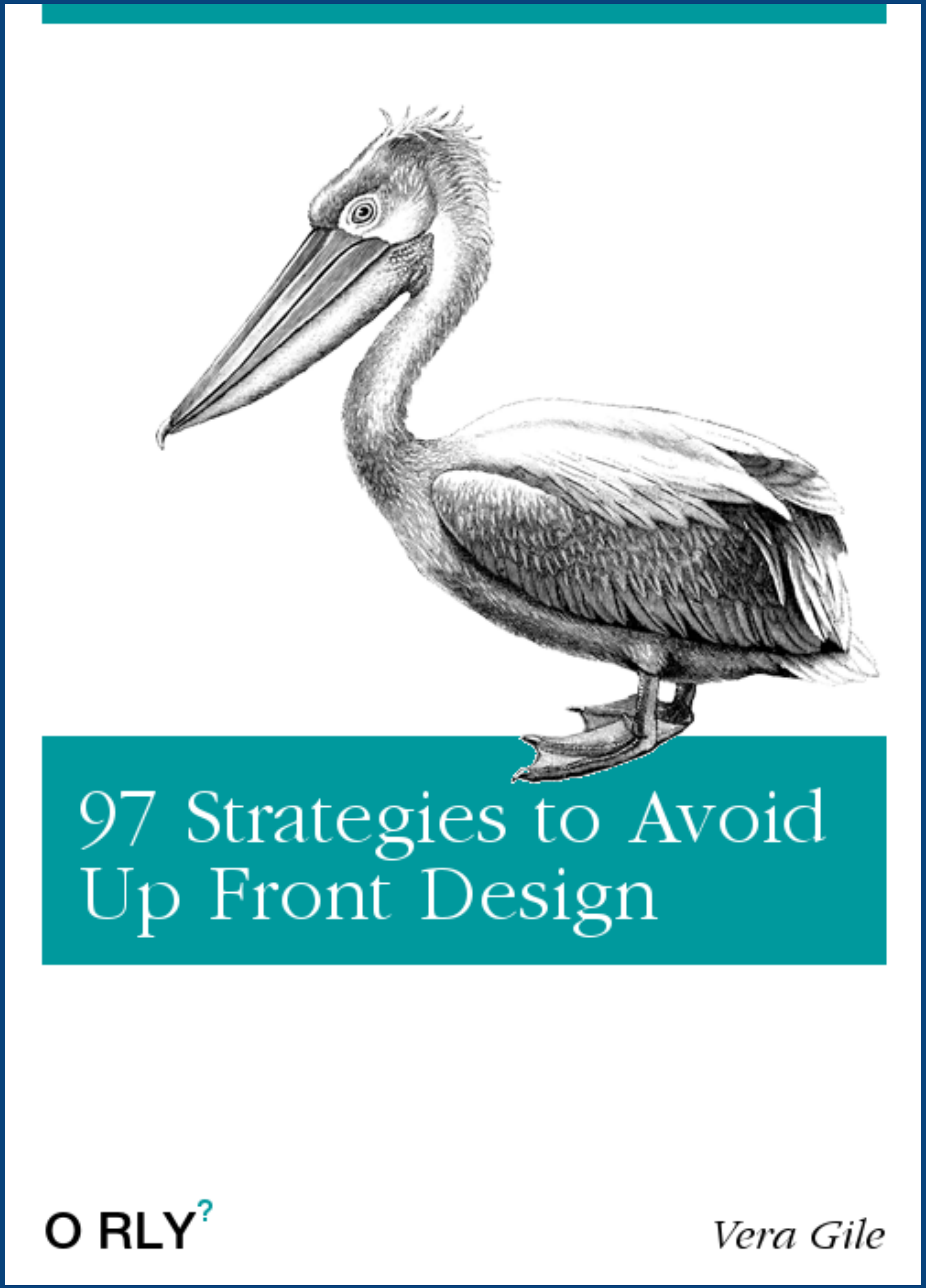
Scott Ambler

Concrete experiment

Proof of concept, prototype, spike, tracer, vertical slice, walking skeleton, executable reference architecture, …

Just enough up front design to create **firm and sufficient foundations**

How much up front design should you do?



97 Strategies to Avoid Up Front Design

O RLY?

Vera Gile

#52

“I’m good with maybe a day for a one-year effort.”

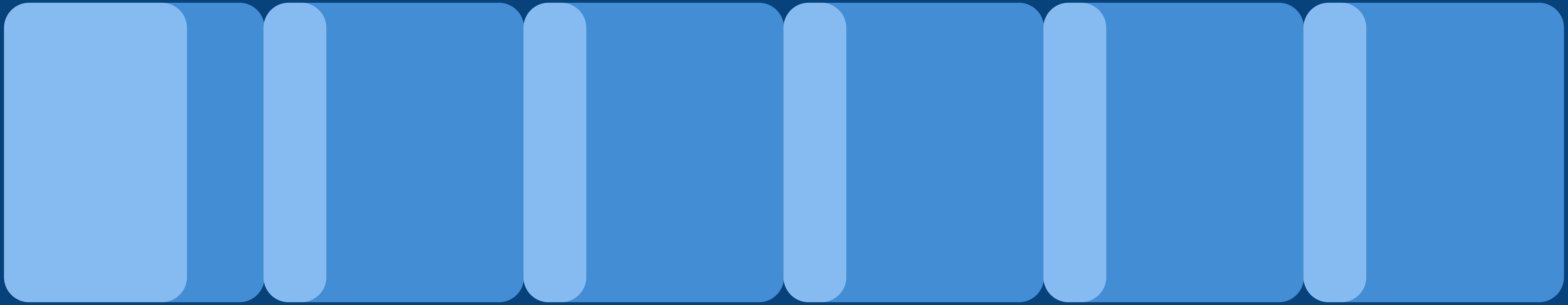
Up front design is an iterative and incremental process; stop when:

- ✓ You understand the significant architectural drivers (requirements, quality attributes, constraints).
- ✓ You understand the context and scope of what you're building.
- ✓ You understand the significant design decisions (i.e. technology, modularity, etc).
- ✓ You have a way to communicate your technical vision to other people.
- ✓ You are confident that your design satisfies the key architectural drivers.
- ✓ You have identified, and are comfortable with, the risks associated with building the software.

Techniques: Workshops, interviews, Event Storming, Impact Mapping, domain modelling, OOAD, CRC, DDD, architecture reviews, ATAM, architecture dry runs, Risk-storming, concrete experiments, C4 model, ADRs, etc.

How long?

Hours, days or weeks … not months or years



Some Design Up Front
+ Evolutionary Design

Some up front design to create a **starting point** and **direction** for further **evolutionary design**

Estimates?

Adopt an agile mindset

Choose a starting point and continuously improve
to discover what works for you

Thank you!

Simon Brown