

GOTO Copenhagen 2021

#GOTOcph

Diagrams as code 2.0



Simon Brown

 @simonbrown

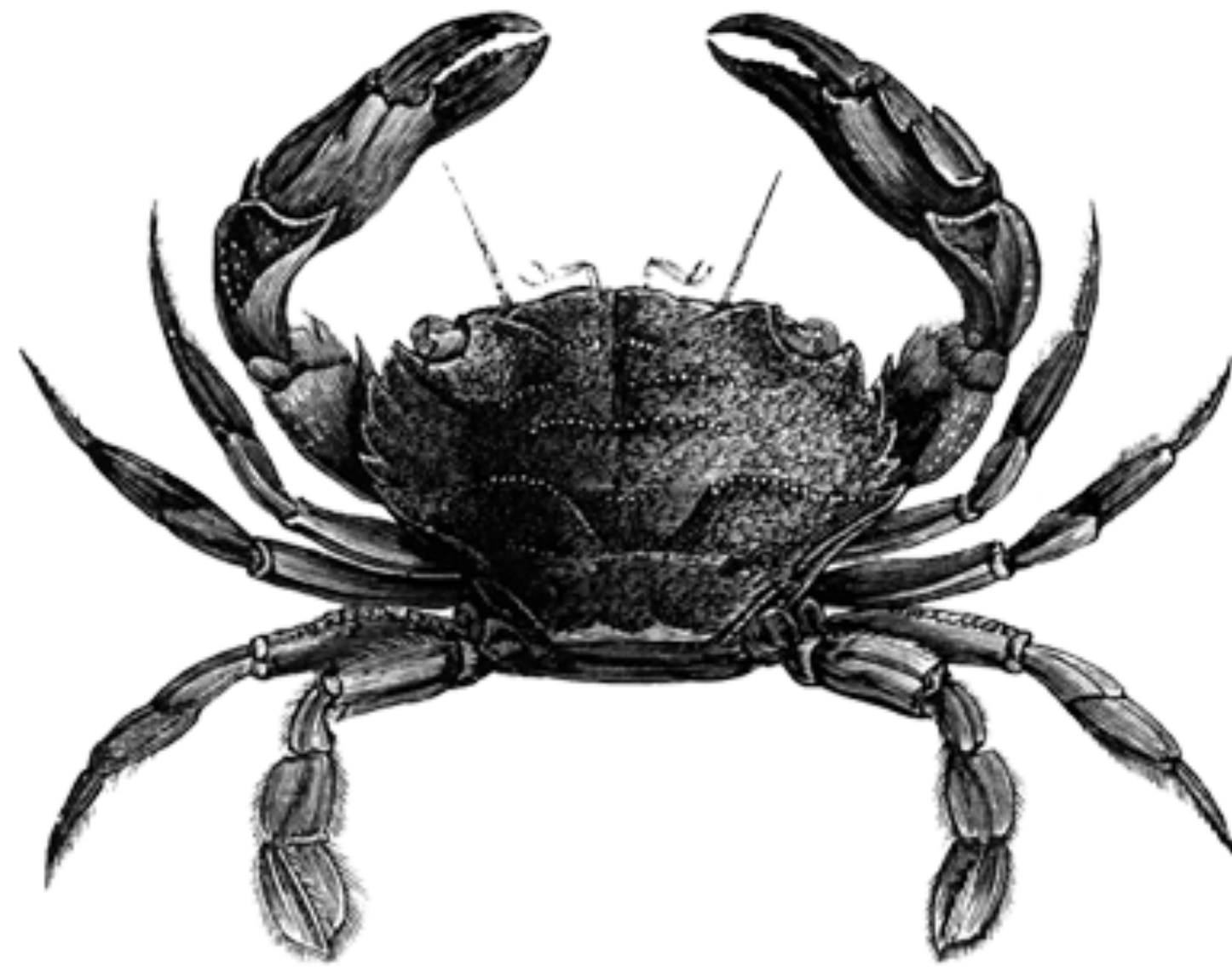


+ some free and open source tooling for creating software architecture diagrams

Teams need a **ubiquitous language**
to communicate effectively



Fewer people are using UML



97 Ways to Sidestep UML

O RLY?

Knowfa Mallity

#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.”

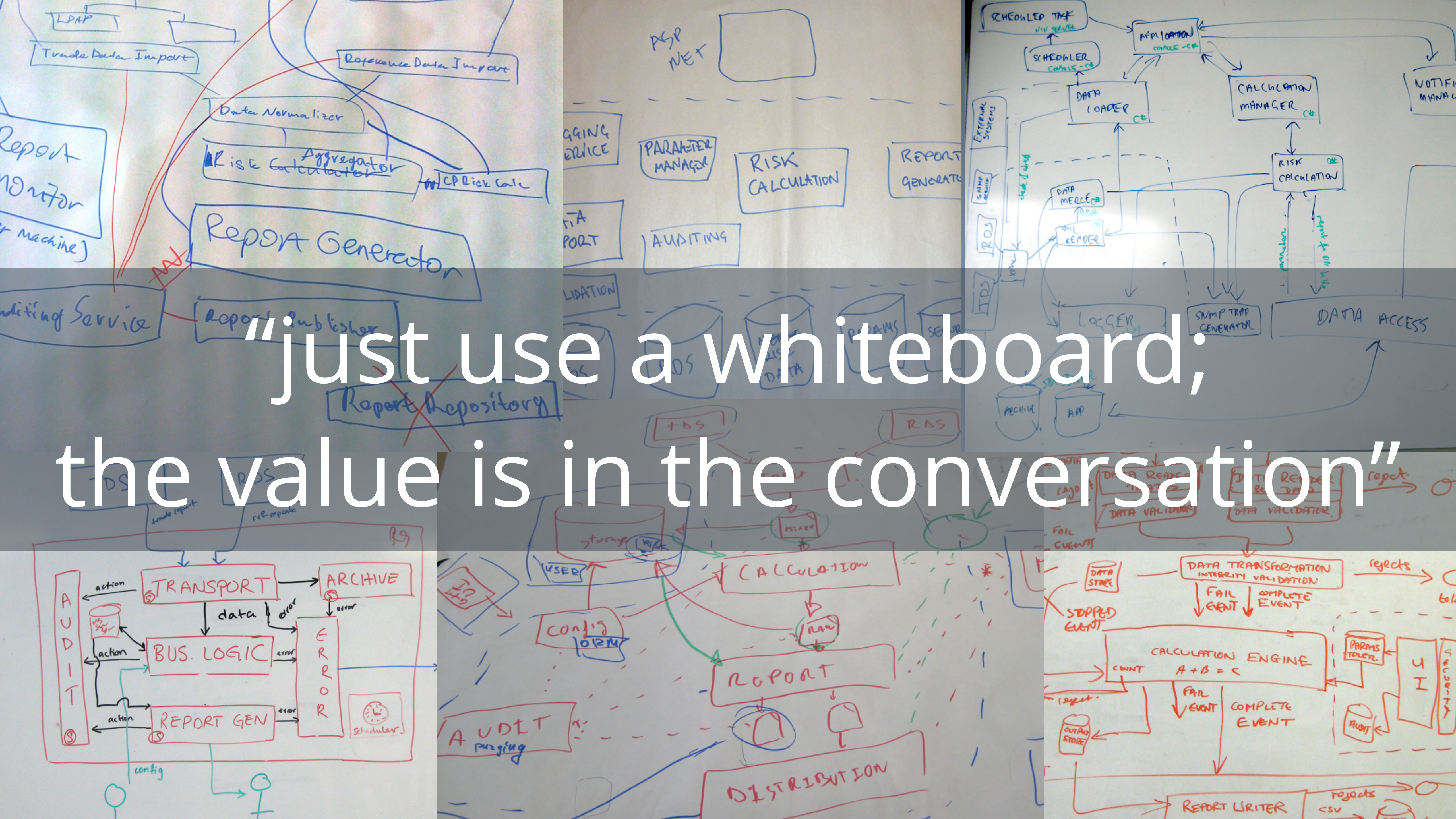
#66 “The tooling sucks.”

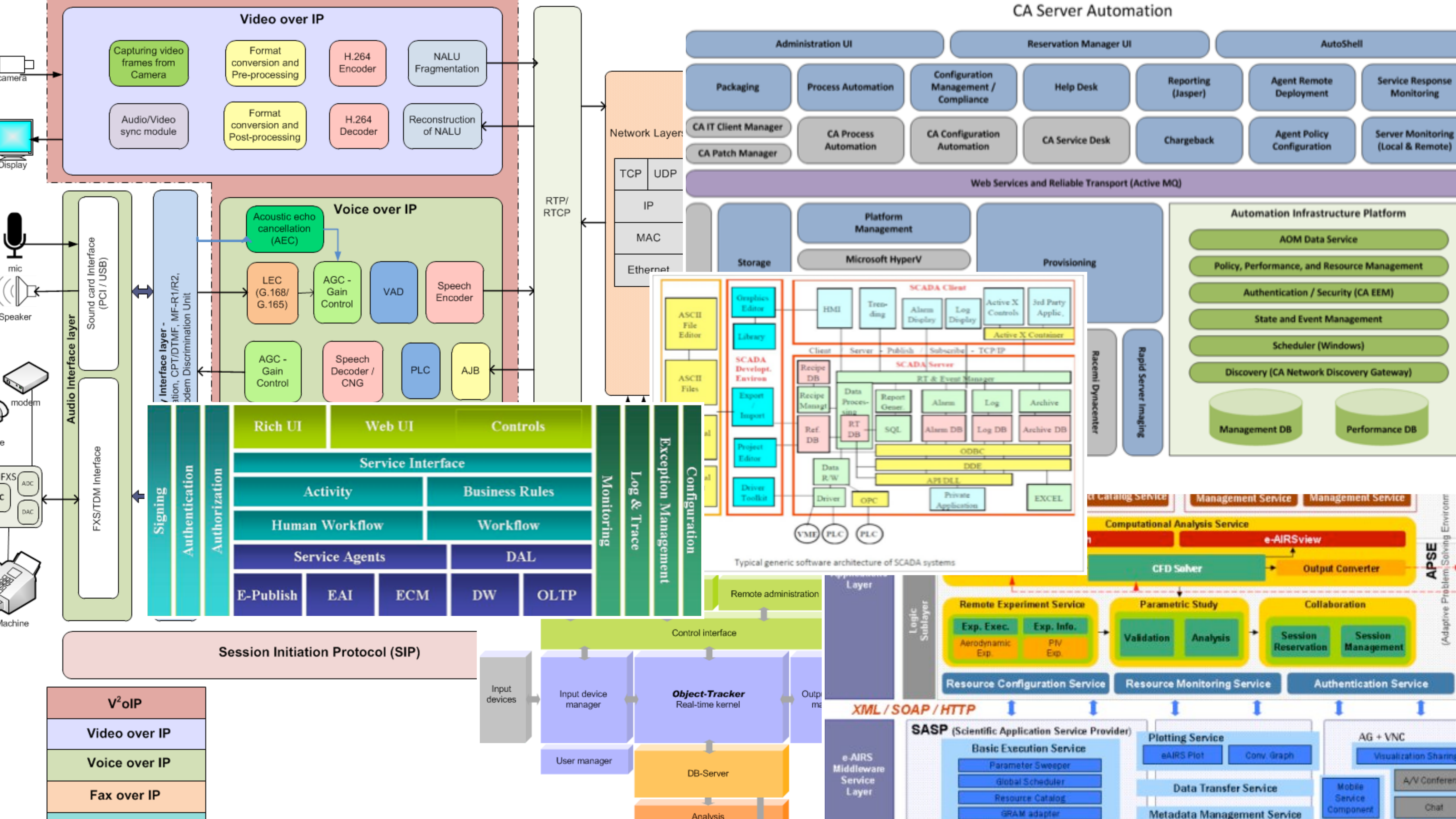
#80 “It’s too detailed.”

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

#92 “It’s not expected in agile.”

#97 “The value is in the conversation.”





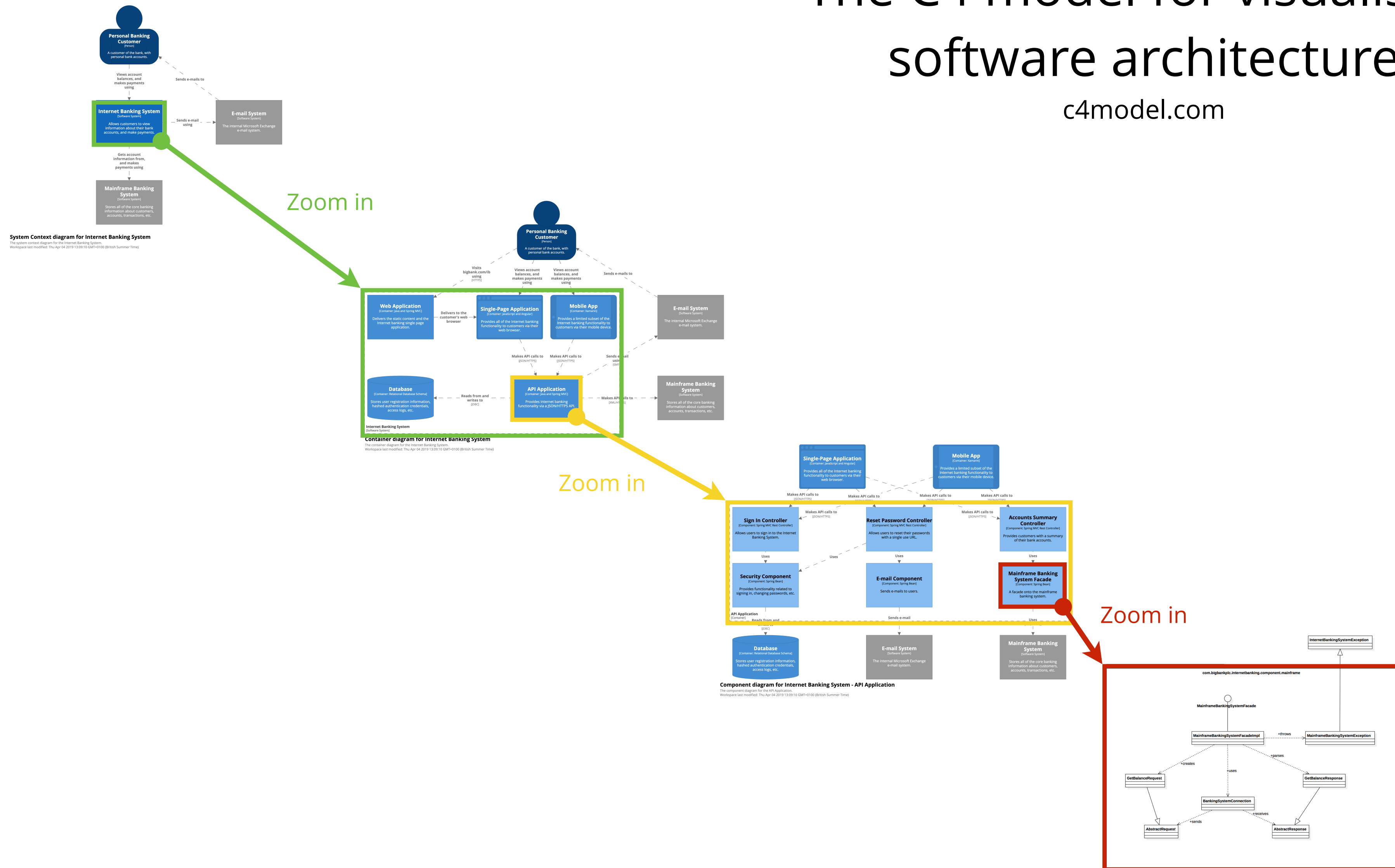
If you're going to use “boxes & lines”,
at least do so in a **structured way**,
using a **self-describing notation**

C4

c4model.com

The C4 model for visualising software architecture

c4model.com

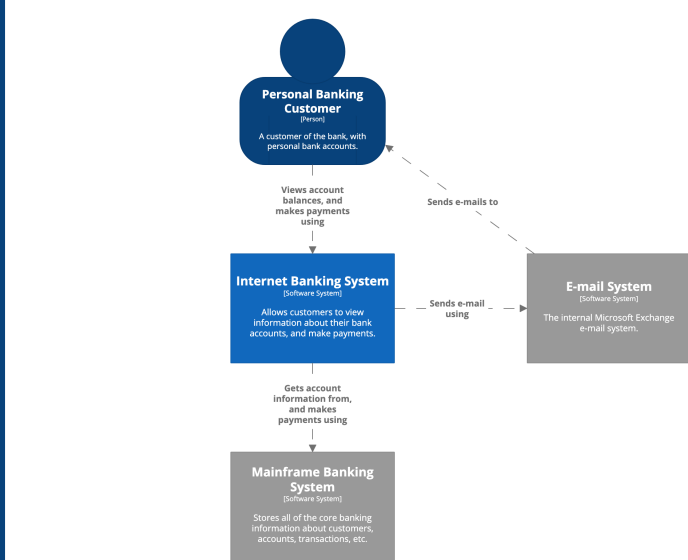
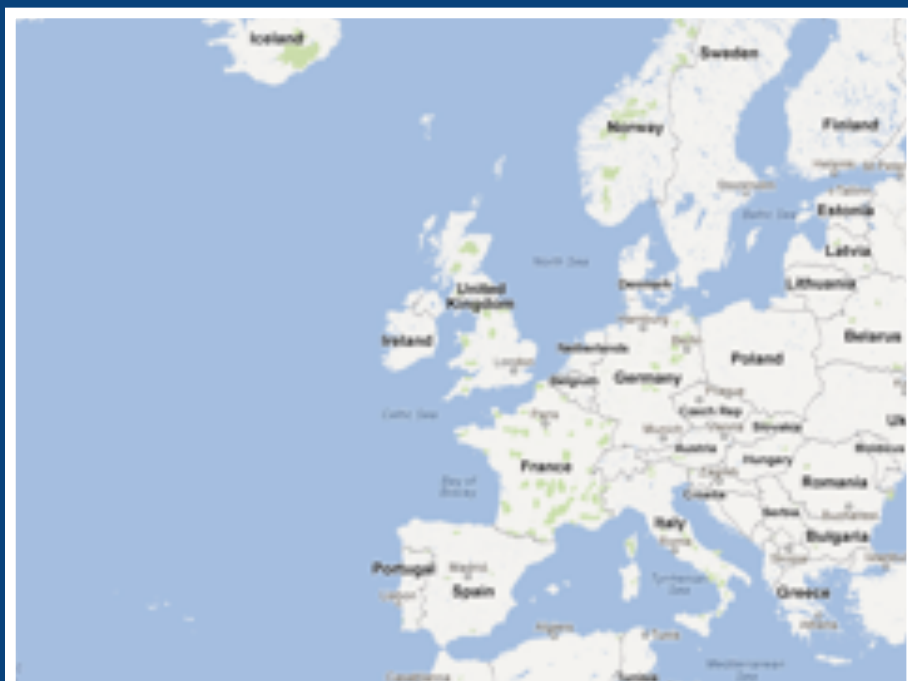


Level 1
Context

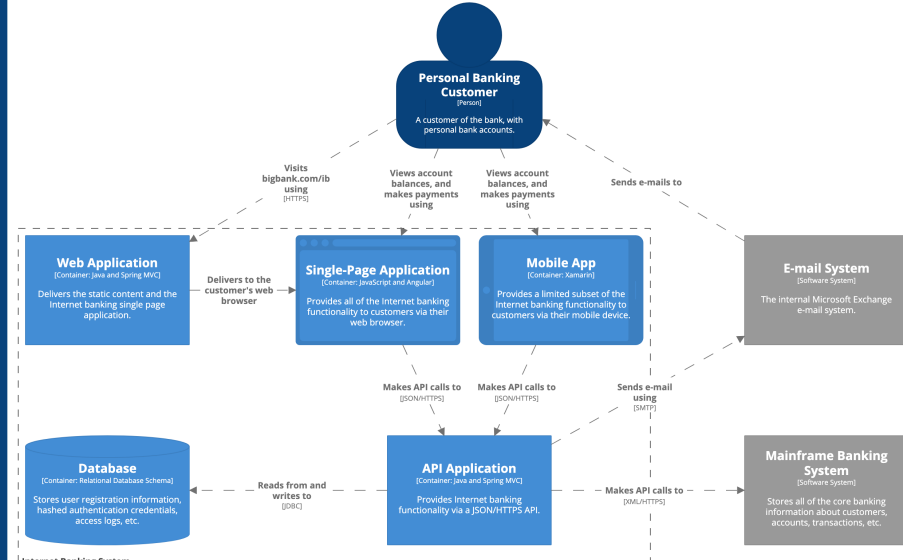
Level 2
Containers

Level 3
Components

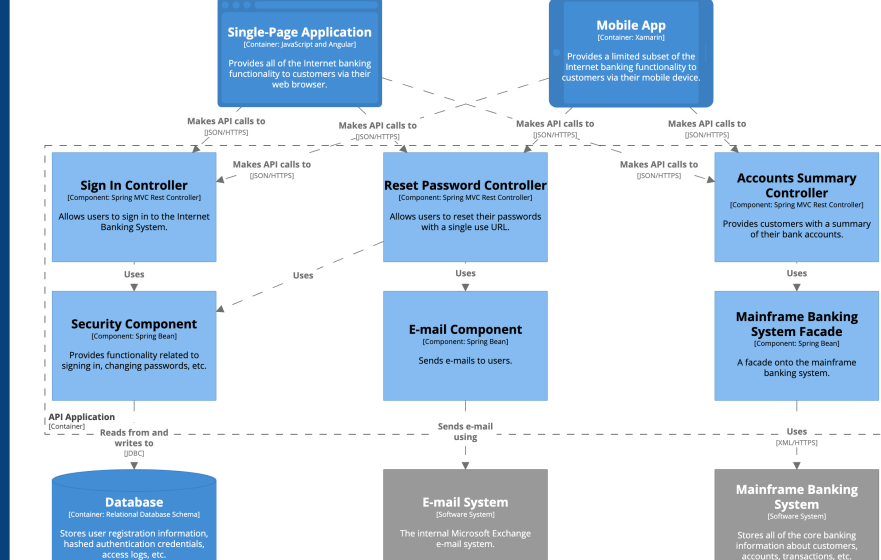
Level 4
Code



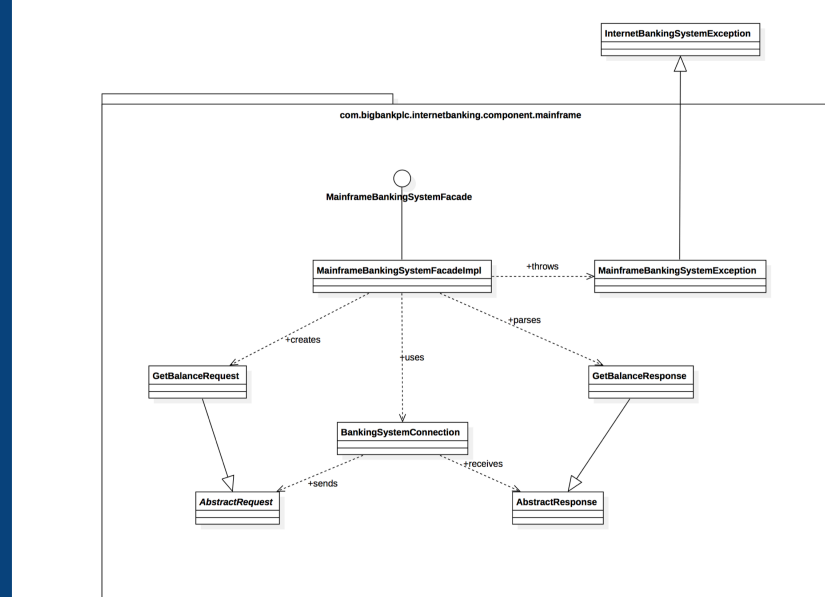
System Context diagram for Internet Banking System
The system context diagram for the Internet Banking System.
Workspace last modified: Thu Apr 04 2019 13:05:10 GMT+0100 (British Summer Time)



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



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



Diagrams are maps

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

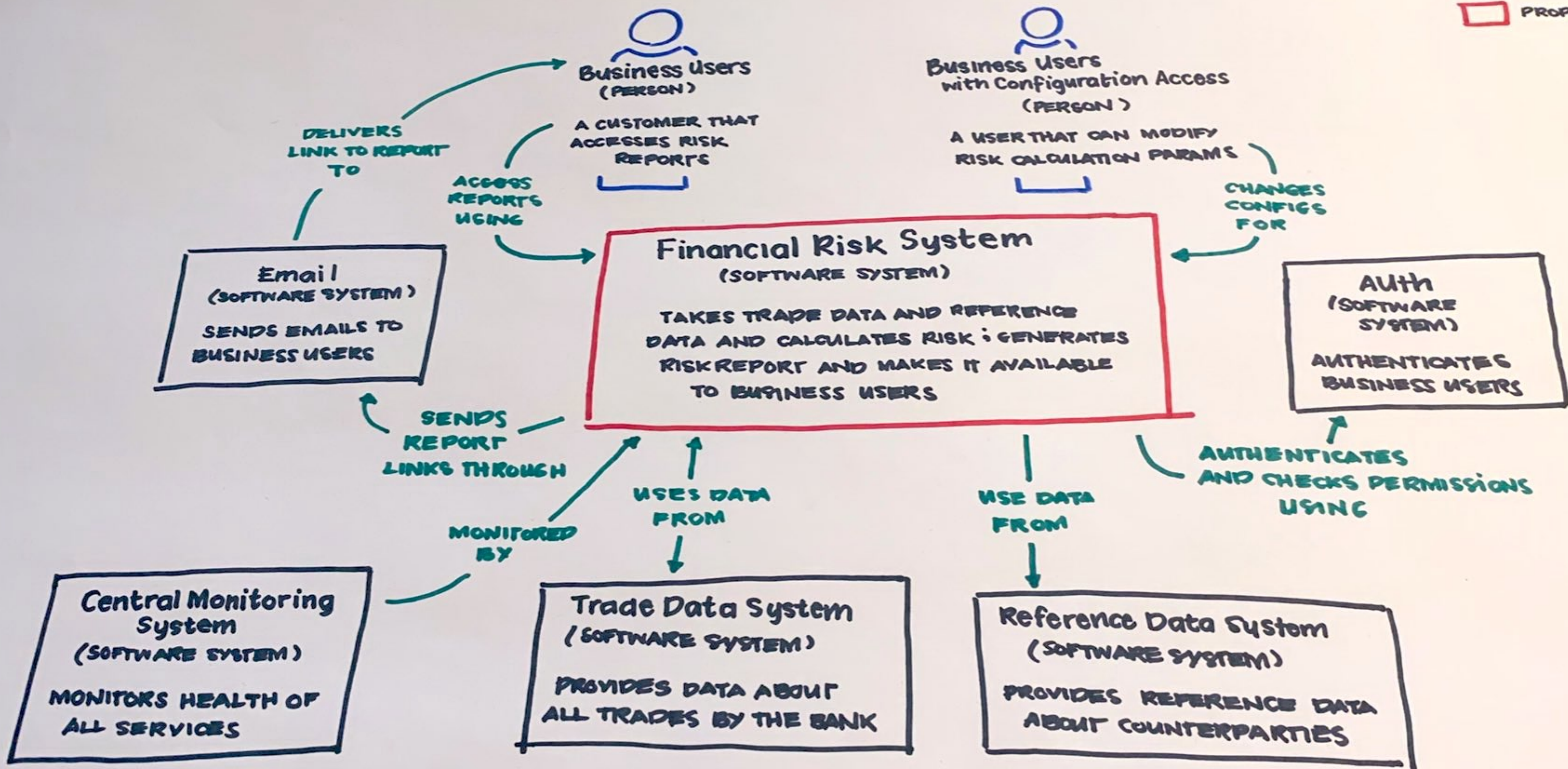
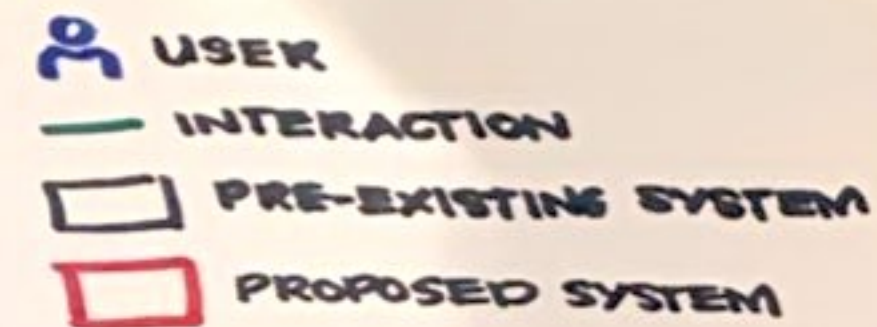
System Context diagram

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

Who is using it? What are they doing?

What system integrations does it need to support?

Financial Risk System: Context Diagram



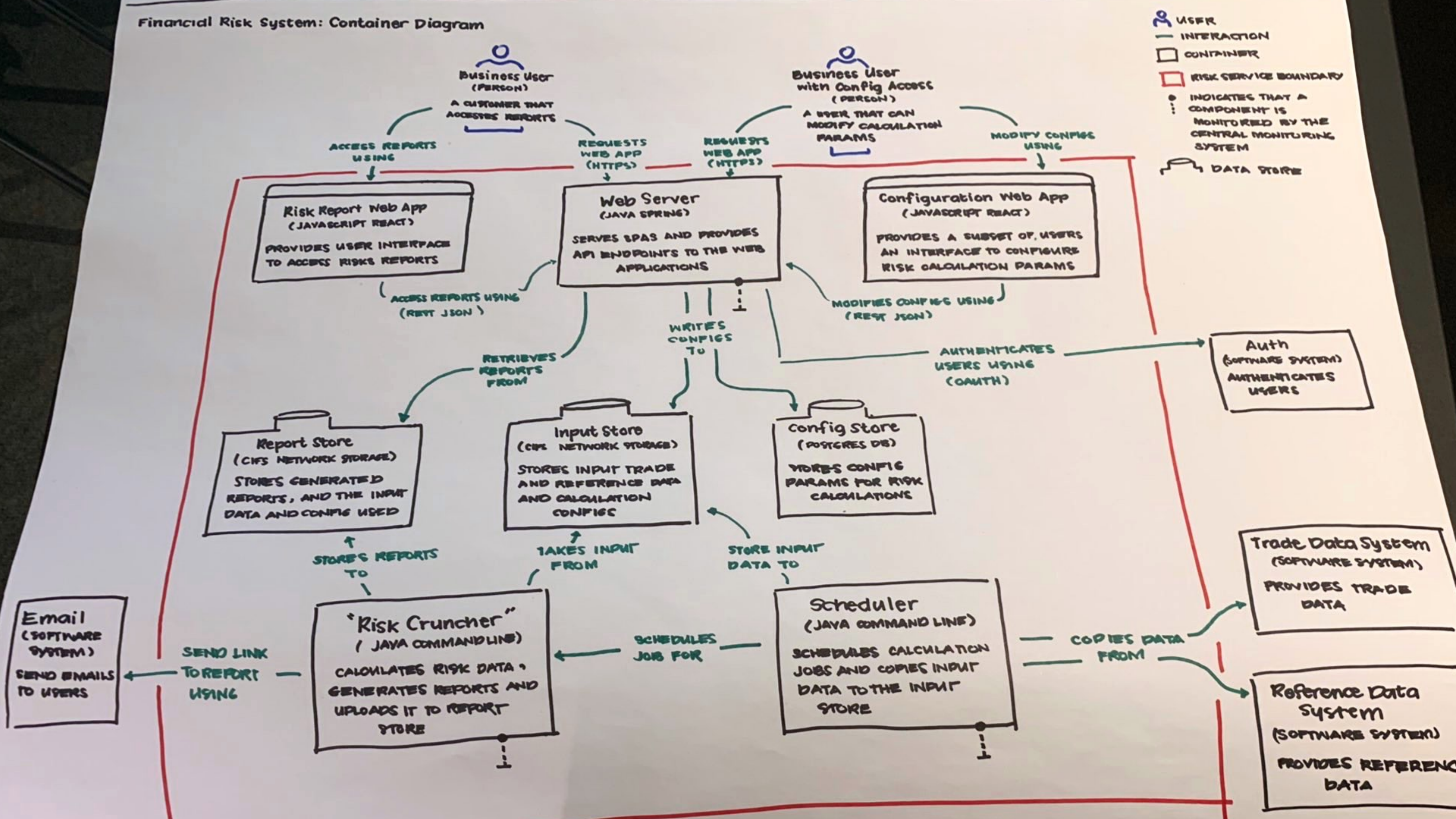
Container diagram

What are the major technology building blocks?

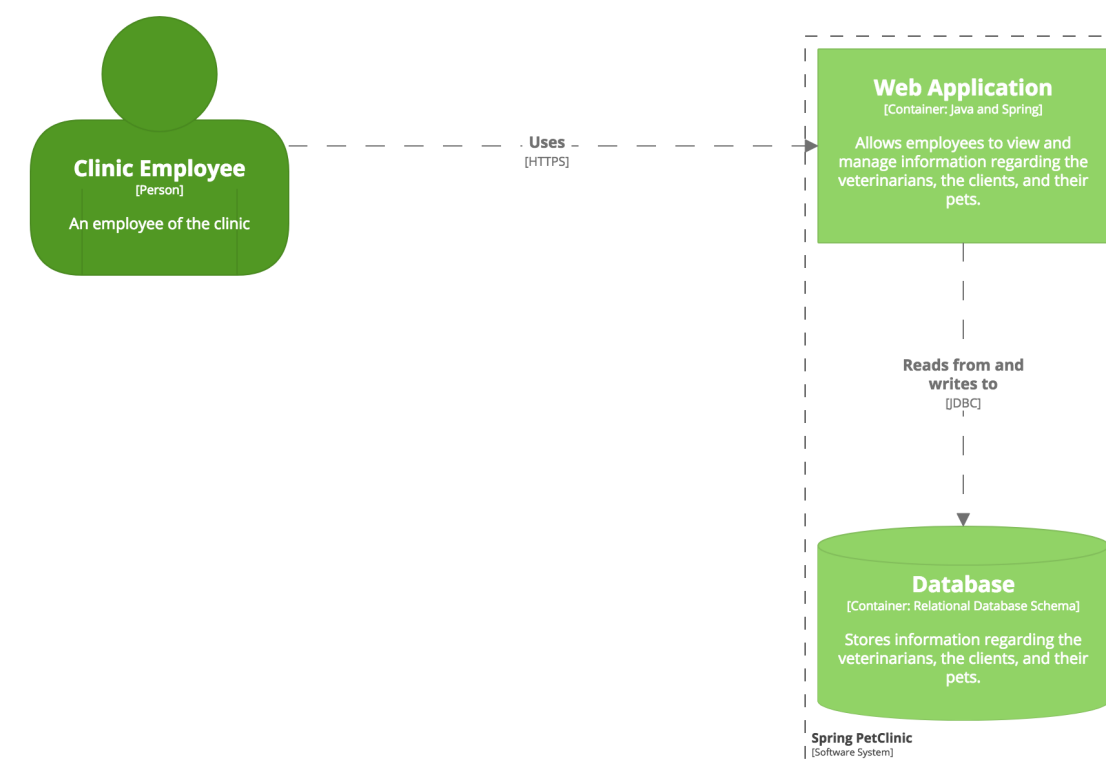
What are their responsibilities?

How do they communicate?

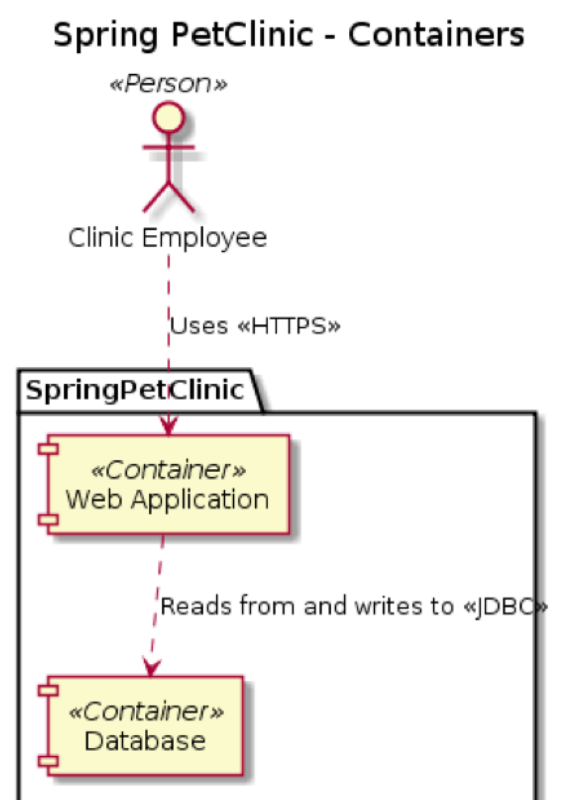
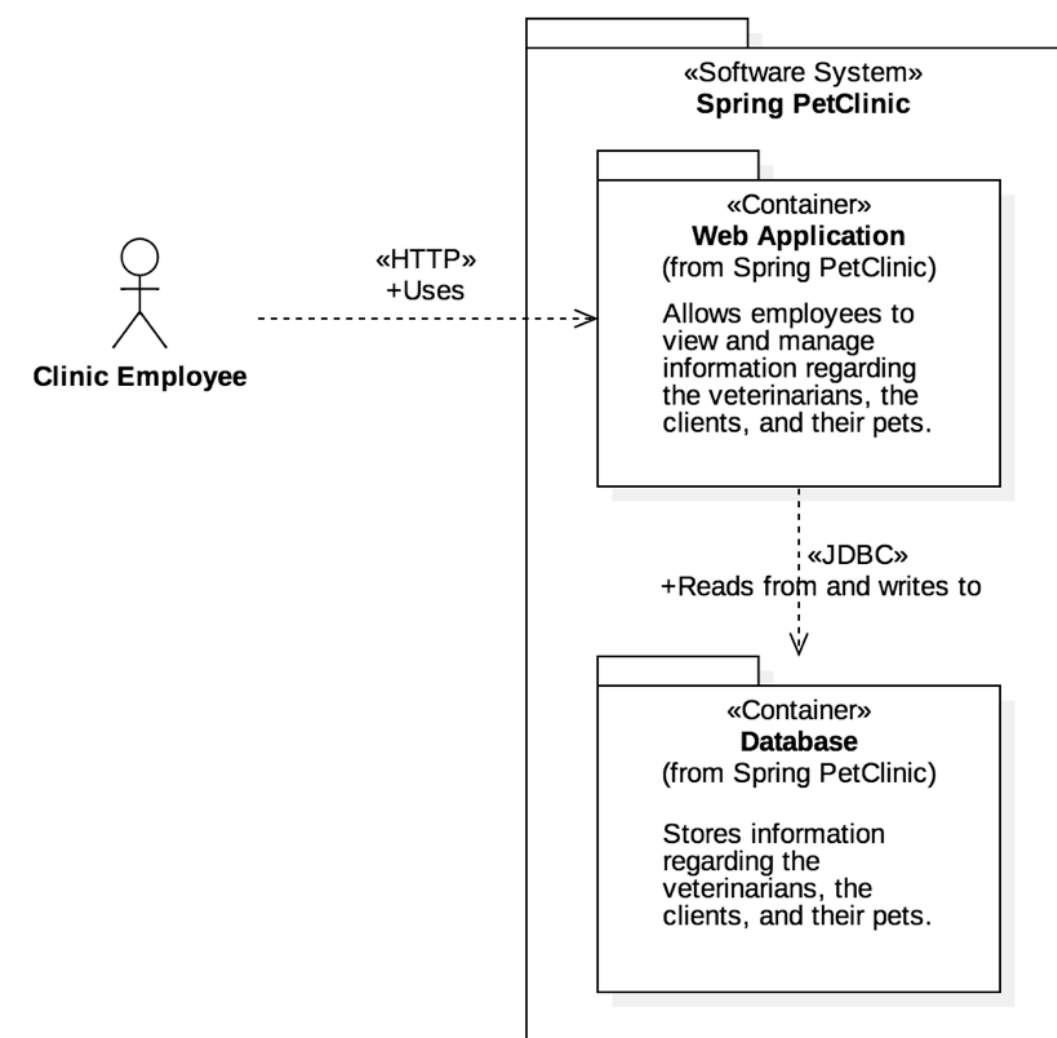
Financial Risk System: Container Diagram



The C4 model is notation independent



Container diagram for Spring PetClinic
The Containers diagram for the Spring PetClinic system.
Last modified: Thursday 17 August 2017 10:15 UTC | Version: 95de1d9f8b6f3560915331664b27a4a75ce1f1f6



The Container diagram for the Spring PetClinic system.

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

Tooling?

TECHNOLOGY RADAR

DownloadSubscribeSearchBuild your RadarAbout

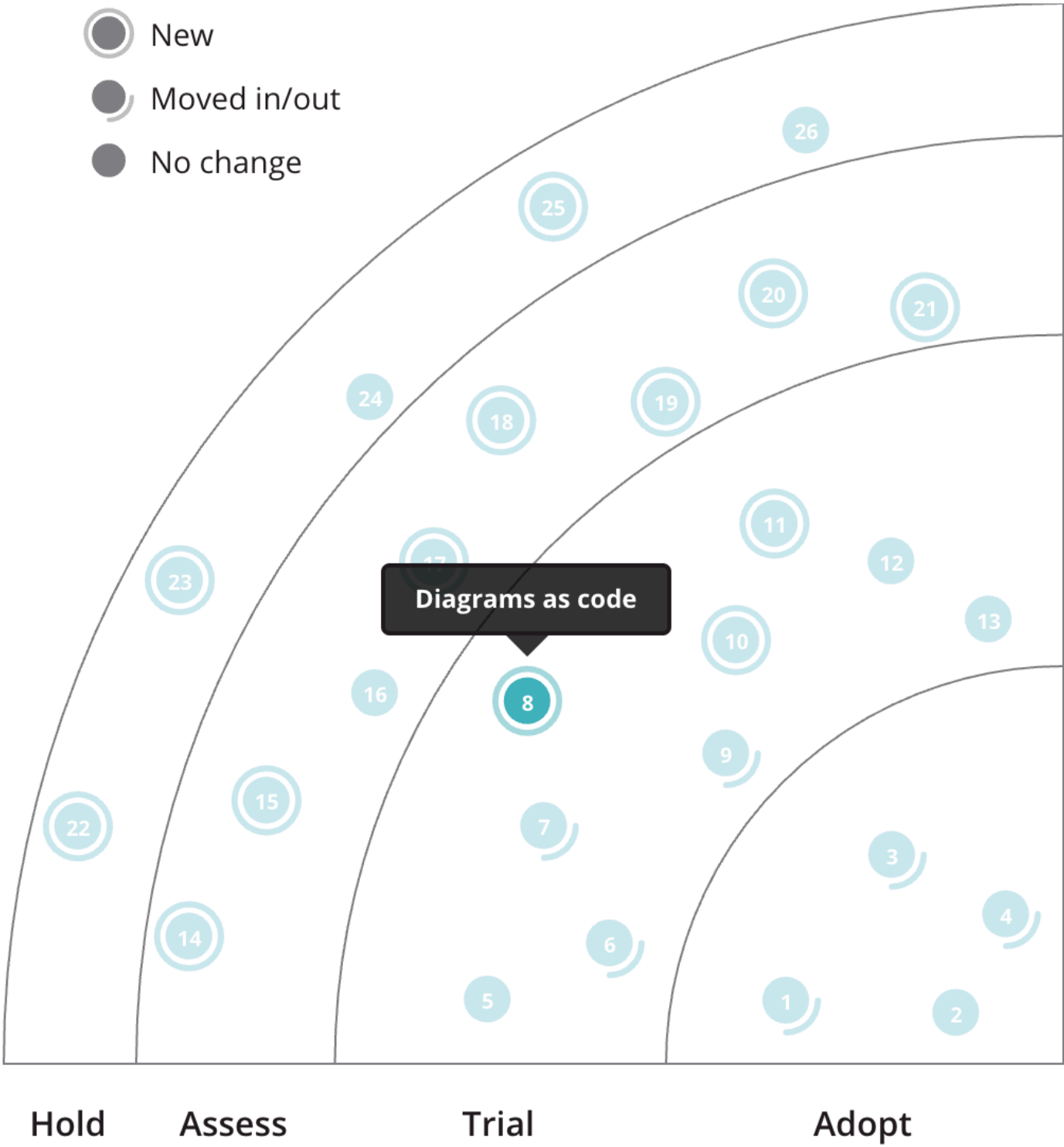


Techniques

Trial ?

- 5. Continuous delivery for machine learning (CD4ML)
- 6. Data mesh
- 7. Declarative data pipeline definition
- 8. **Diagrams as code**

We're seeing more and more tools that enable you to create software architecture and other **diagrams as code**. There are benefits to using these tools over the heavier alternatives, including easy version control and the ability to generate the DSLs from many sources. Tools in this space that we like include Diagrams, Structurizr DSL, AsciiDoctor Diagram and stables such as WebSequenceDiagrams, PlantUML and the venerable Graphviz. It's also fairly simple to generate your own SVG these days, so don't rule out quickly writing your own tool either.

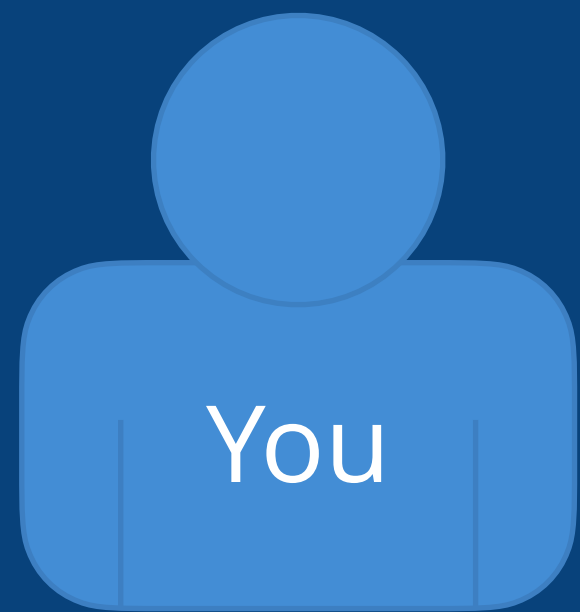


Unable to find something you expected to see?

Each edition of the radar features blips reflecting what we came across during the previous six months. We might have covered

“Diagrams as code” is easy to author,
diff, version control, collaborate on,
integrate into CI/CD, etc

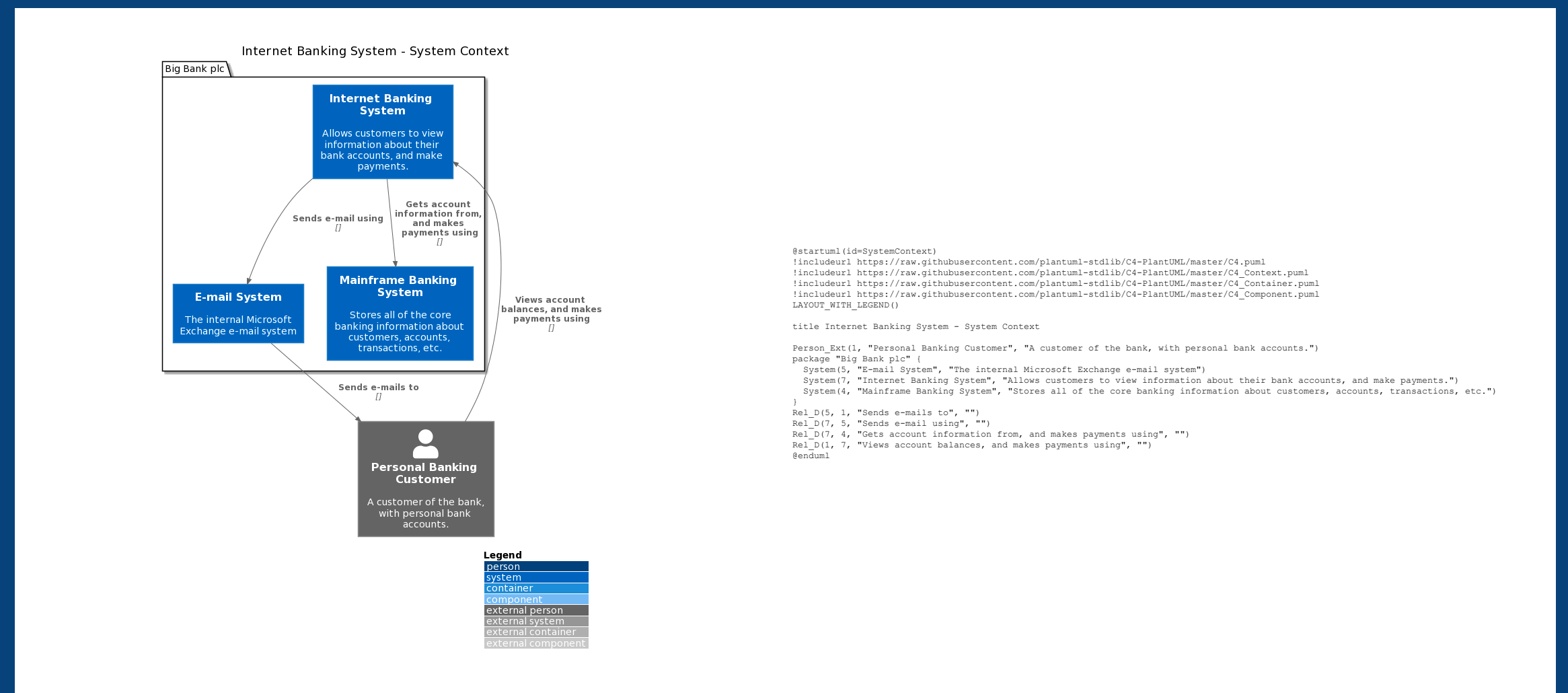
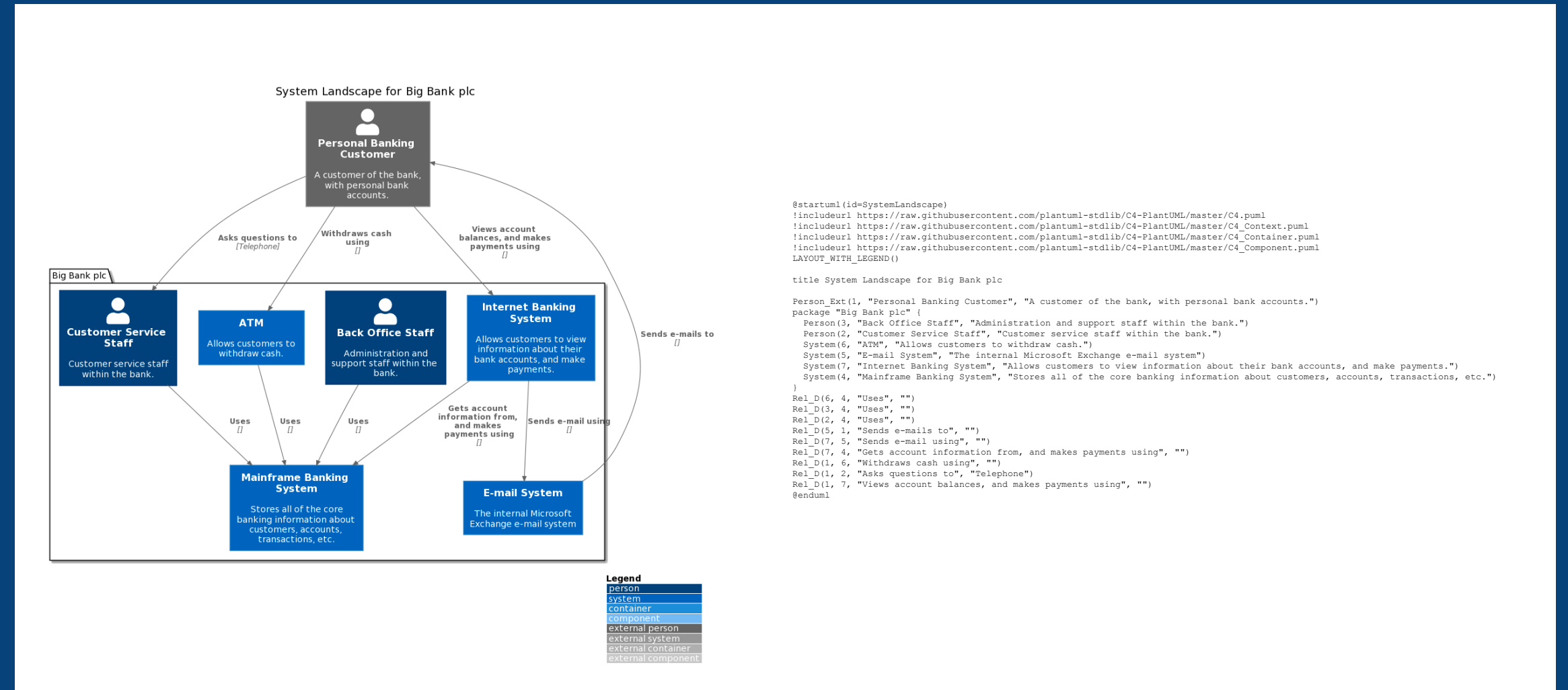
Diagramming vs modelling

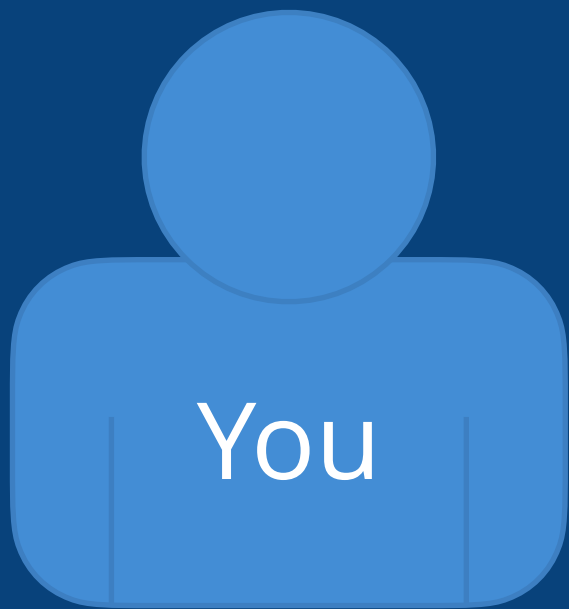


Create and maintain

Diagrams as code 1.0

You create and maintain multiple diagrams, remembering to keep them all in sync whenever you change a diagram





Create and maintain

```
workspace "Big Bank plc" "This is an example workspace to illustrate the key features of Structurizr, via the DSL, based around a fictional online banking system." {
  model {
    customer = person "Personal Banking Customer" "A customer of the bank, with personal bank accounts."

    enterprise "Big Bank plc" {
      supportStaff = person "Customer Service Staff" "Customer service staff within the bank." "Bank Staff"
      backOffice = person "Back Office Staff" "Administration and support staff within the bank." "Bank Staff"

      mainframe = softwareSystem "Mainframe Banking System" "Stores all of the core banking information about customers, accounts, transactions, etc." "Existing System"
      email = softwareSystem "E-mail System" "The internal Microsoft Exchange e-mail system." "Existing System"
      atm = softwareSystem "ATM" "Allows customers to withdraw cash." "Existing System"

      internetBankingSystem = softwareSystem "Internet Banking System" "Allows customers to view information about their bank accounts, and make payments." {
        singlePageApplication = container "Single-Page Application" "Provides all of the Internet banking functionality to customers via their web browser." "JavaScript and Angular" "Web Browser"
        mobileApp = container "Mobile App" "Provides a limited subset of the Internet banking functionality to customers via their mobile device." "React Native" "Mobile App"
        webApplication = container "Web Application" "Delivers the static content and the Internet banking single page application." "Java and Spring MVC"
        apiApplication = container "API Application" "Provides Internet banking functionality via a JSON/HTTPS API." "Java and Spring MVC"
        signInController = component "Sign In Controller" "Allows users to sign in to the Internet Banking System." "Spring MVC Rest Controller"
        accountSummaryController = component "Account Summary Controller" "Provides customers with a summary of their bank accounts." "Spring MVC Rest Controller"
        resetPasswordController = component "Reset Password Controller" "Allows users to reset their passwords with a single use only." "Spring MVC Rest Controller"
        securityComponent = component "Security Component" "Provides functionality related to signing in, changing passwords, etc." "Spring Bean"
        mainframeBankingSystemFacade = component "Mainframe Banking System Facade" "A facade onto the mainframe banking system." "Spring Bean"
        emailComponent = component "E-mail Component" "Sends e-mails to users." "Spring Bean"
      }

      database = container "Database" "Stores user registration information, hashed authentication credentials, access logs, etc." "Oracle Database Schema" "Database"
    }

    # relationships between people and software systems
    uses <customer> -> internetBankingSystem "Views account balances, and makes payments using"
    internetBankingSystem -> mainframe "Gets account information from, and makes payments using"
    internetBankingSystem -> email "Sends e-mail using"
    email -> customer "Sends e-mails to"
    customer -> supportStaff "Asks questions to" "Telephone"
    supportStaff -> mainframe "Uses"
    customer -> atm "Withdraws cash using"
    atm -> mainframe "Uses"
    backOffice -> mainframe "Uses"

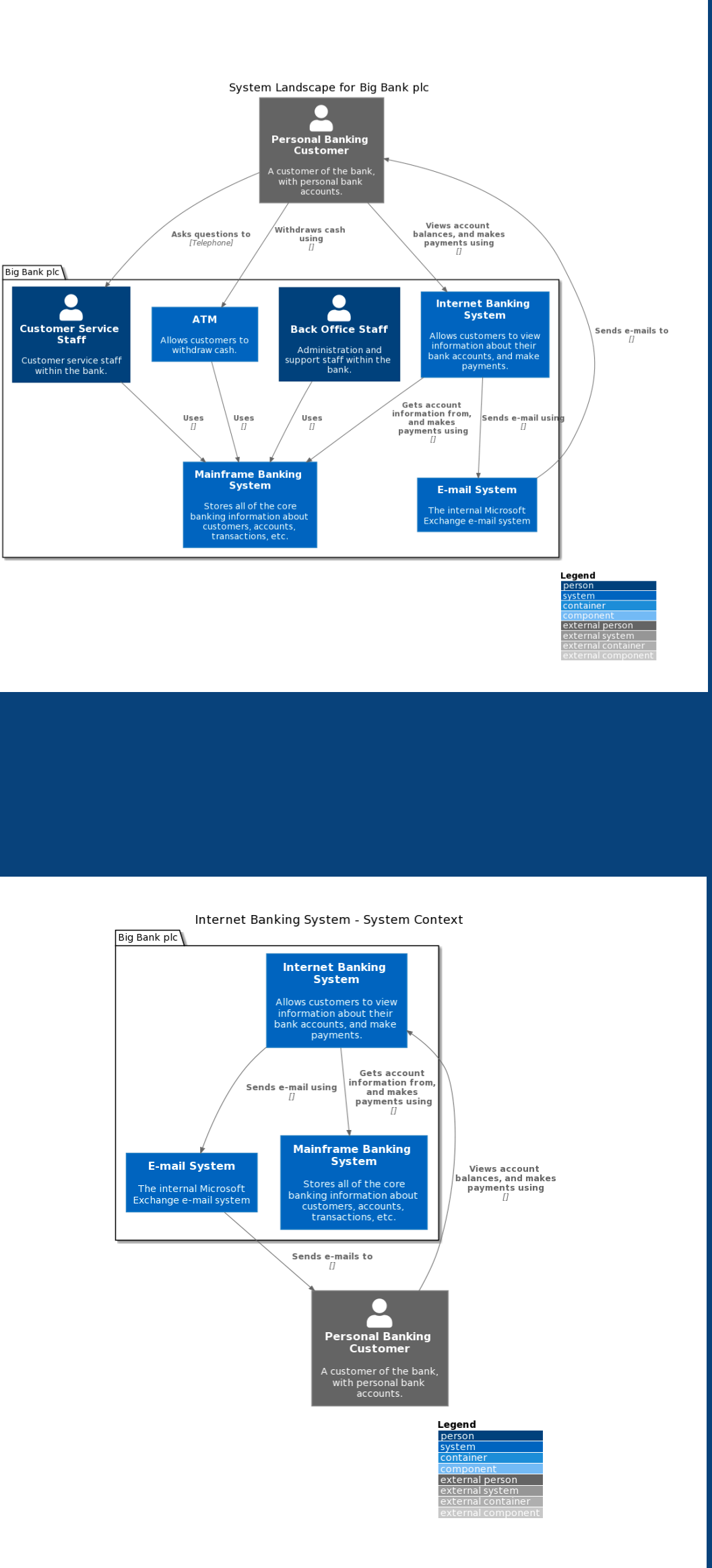
    # relationships to/from containers
    customer -> webApplication "Visits bigbank.com/ib using" "HTTPS"
    customer -> singlePageApplication "Views account balances, and makes payments using"
    customer -> mobileApp "Views account balances, and makes payments using"
    internetBankingSystem -> webApplication "Delivers to the customer's web browser"

    # relationships to/from components
    singlePageApplication -> signInController "Makes API calls to" "JSON/HTTPS"
    singlePageApplication -> accountSummaryController "Makes API calls to" "JSON/HTTPS"
    singlePageApplication -> resetPasswordController "Makes API calls to" "JSON/HTTPS"
    mobileApp -> signInController "Makes API calls to" "JSON/HTTPS"
    mobileApp -> accountSummaryController "Makes API calls to" "JSON/HTTPS"
    mobileApp -> resetPasswordController "Makes API calls to" "JSON/HTTPS"
    apiApplication -> signInController "Makes API calls to" "JSON/HTTPS"
    apiApplication -> accountSummaryController "Makes API calls to" "JSON/HTTPS"
    apiApplication -> resetPasswordController "Makes API calls to" "JSON/HTTPS"
    securityComponent -> database "Reads from and writes to" "JDBC"
    mainframeBankingSystemFacade -> mainframe "Makes API calls to" "JSON/HTTPS"
    emailComponent -> email "Sends e-mail using"
  }
}
```

Automatically generates

Diagrams as code 2.0

You create and maintain a single model, and the tool generates multiple diagrams, automatically keeping them all in sync whenever you change the model





Search or jump to...



[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)



[structurizr](#) / [dsl](#) Public

Unwatch ▾

14

Star

362

Fork

75

[Code](#) [Issues](#) 2 [Pull requests](#) 1 [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

master ▾ 1 branch 7 tags

[Go to file](#) [Add file](#) ▾ [Code](#) ▾

Simon Brown Updated to reflect release. ✓ 97b1226 3 days ago 🕒 277 commits		
.github/workflows	GitHub Actions fixes.	5 months ago
docs	Updated to reflect release.	3 days ago
examples	Fixes an issue where <code>this</code> didn't work when defining relationships i...	5 days ago
gradle	Add gradle JAR.	15 months ago
src	Adds support for formatting the branding logo and font as DSL.	4 days ago
.gitignore	Add gradle JAR.	15 months ago
LICENSE	Initial commit	16 months ago
README.md	Added a link to the changelog.	3 months ago
build.gradle	Updated to reflect release.	3 days ago
gradle.properties	Adds a dummy Gradle properties file for GitHub Actions.	5 months ago
gradlew	Initial commit of source code for the DSL parser.	16 months ago
gradlew.bat	Initial commit of source code for the DSL parser.	16 months ago

README.md

About

Structurizr DSL

[dsl](#) [software-architecture](#)

[structurizr](#) [c4model](#)

[architecture-diagrams](#)

[Readme](#)

[Apache-2.0 License](#)

Releases 6

v.1.15.0 Latest
3 days ago

[+ 5 releases](#)

Packages

No packages published
[Publish your first package](#)

Contributors 8



Domain concepts

(not “boxes and lines”)

```
@startuml
title Software System - System Context
```

```
top to bottom direction
```

```
hide stereotype
```

```
rectangle "=="User\n<size:10>[Person]</size>" <<User>> as User
```

```
rectangle "=="Software System\n<size:10>[Software System]</size>" <<SoftwareSystem>> as SoftwareSystem
```

```
User ..> SoftwareSystem : "Uses"
```

```
@enduml
```

Domain language of diagramming

(no rules, no guidance)


```
workspace {  
  
  model {  
    user = person "User"  
    softwareSystem = softwareSystem "Software System"  
  
    user -> softwareSystem "Uses"  
  }  
  
  views {  
    systemContext softwareSystem {  
      include *  
      autoLayout  
    }  
  }  
}
```

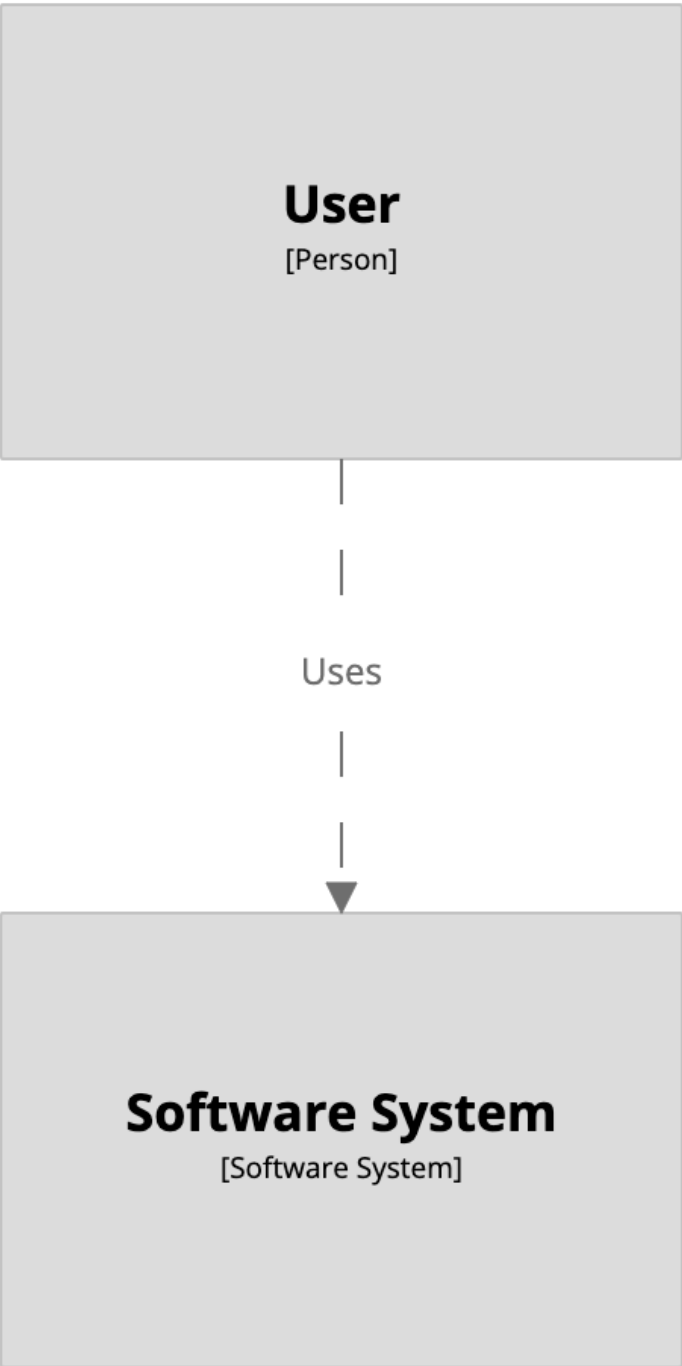
Domain language of software architecture

(metamodel and rules)

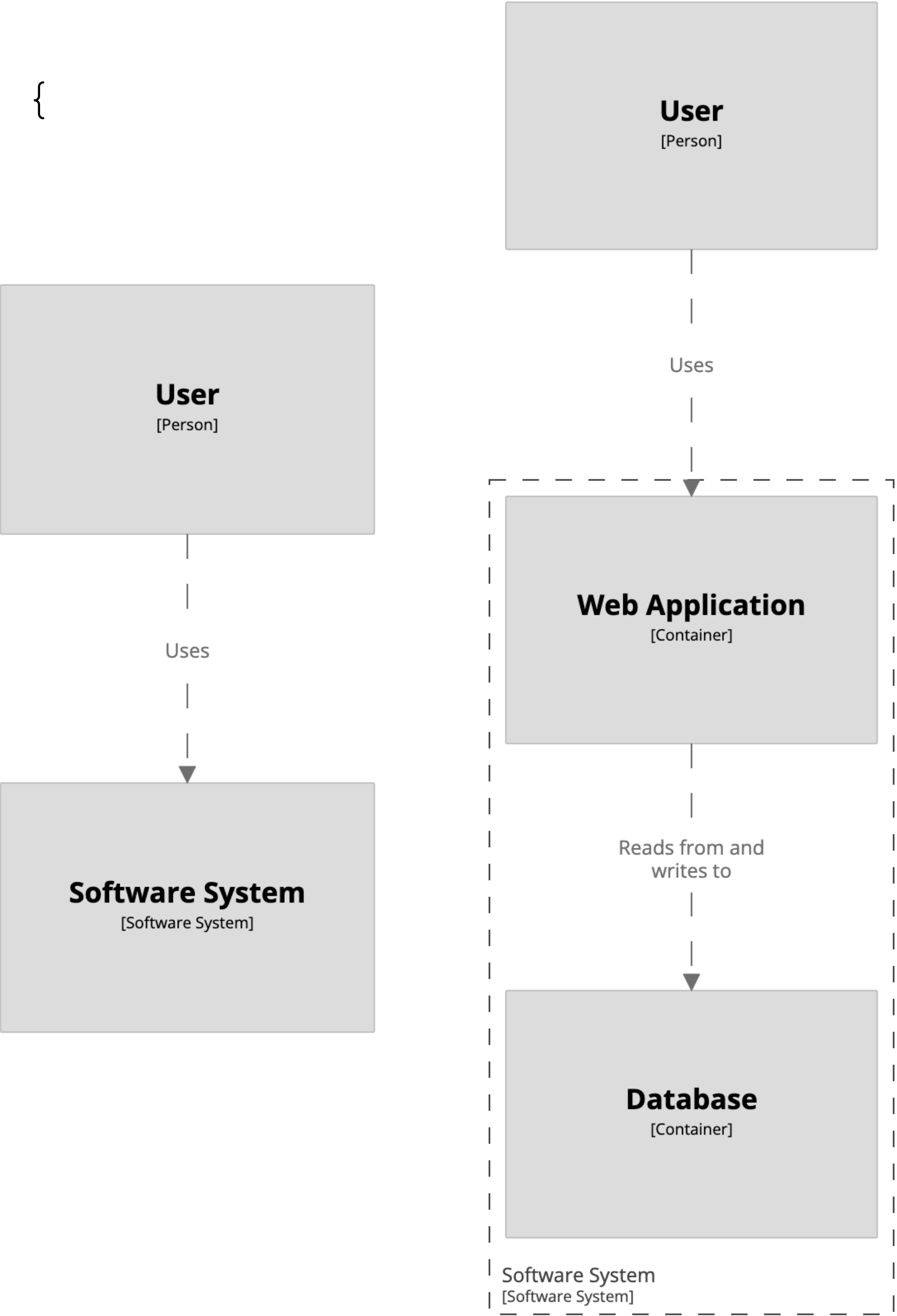
Model-based

(DRY)


```
workspace {  
  
  model {  
    user = person "User"  
    softwareSystem = softwareSystem "Software System"  
  
    user -> softwareSystem "Uses"  
  
  }  
  
  views {  
    systemContext softwareSystem {  
      include *  
      autoLayout  
    }  
  
  }  
  
}
```

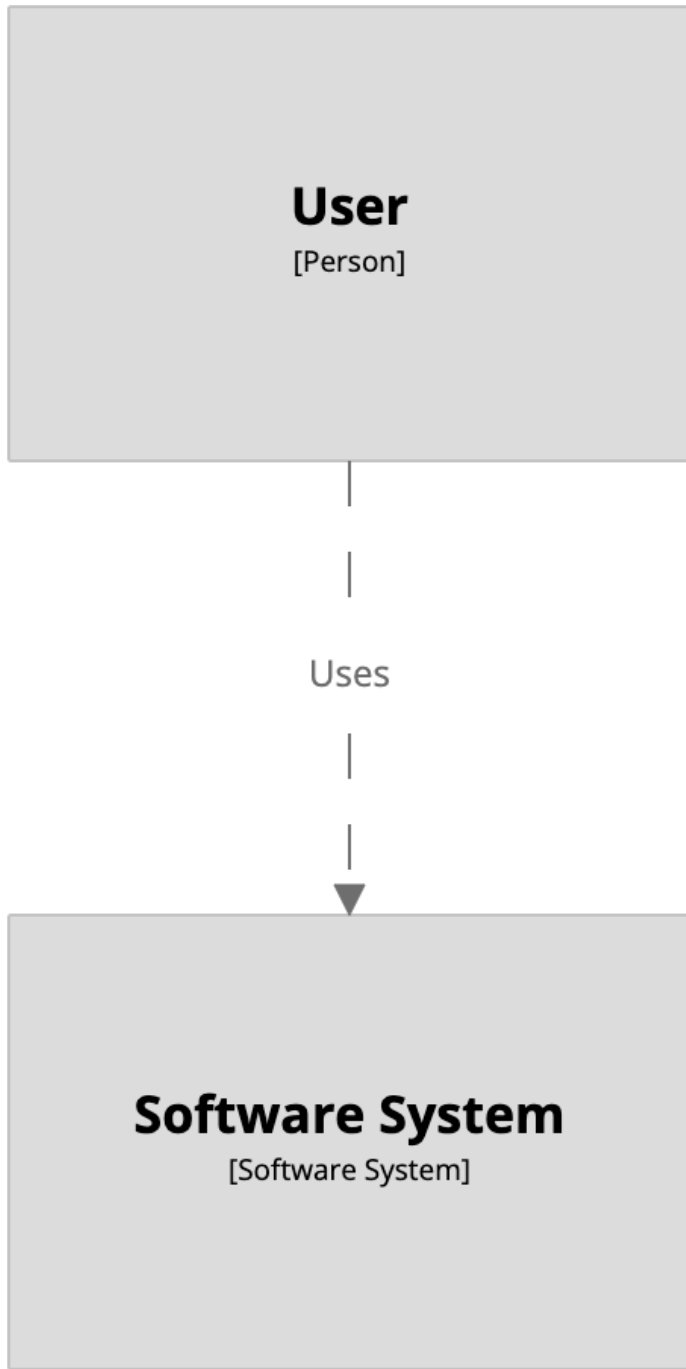


```
workspace {  
  
  model {  
    user = person "User"  
    softwareSystem = softwareSystem "Software System" {  
      webapp = container "Web Application"  
      database = container "Database"  
    }  
  
    user -> webapp "Uses"  
    webapp -> database "Reads from and writes to"  
  }  
  
  views {  
    systemContext softwareSystem {  
      include *  
      autoLayout  
    }  
  
    container softwareSystem {  
      include *  
      autolayout  
    }  
  }  
}
```

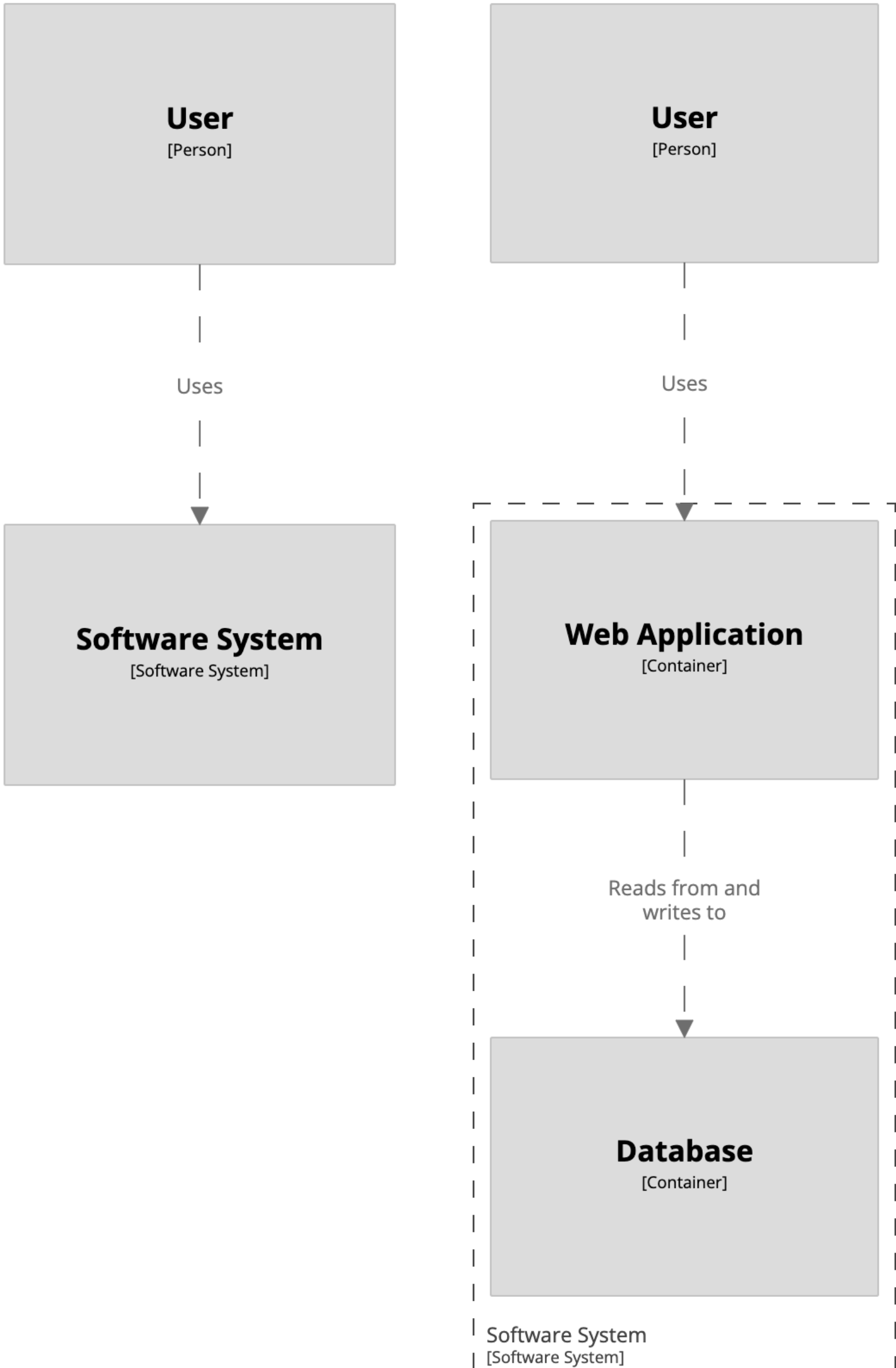


Unspecified relationships can
be implied from the model

user -> softwareSystem "Uses"



user -> webapp "Uses"
webapp -> database "Reads from and writes to"

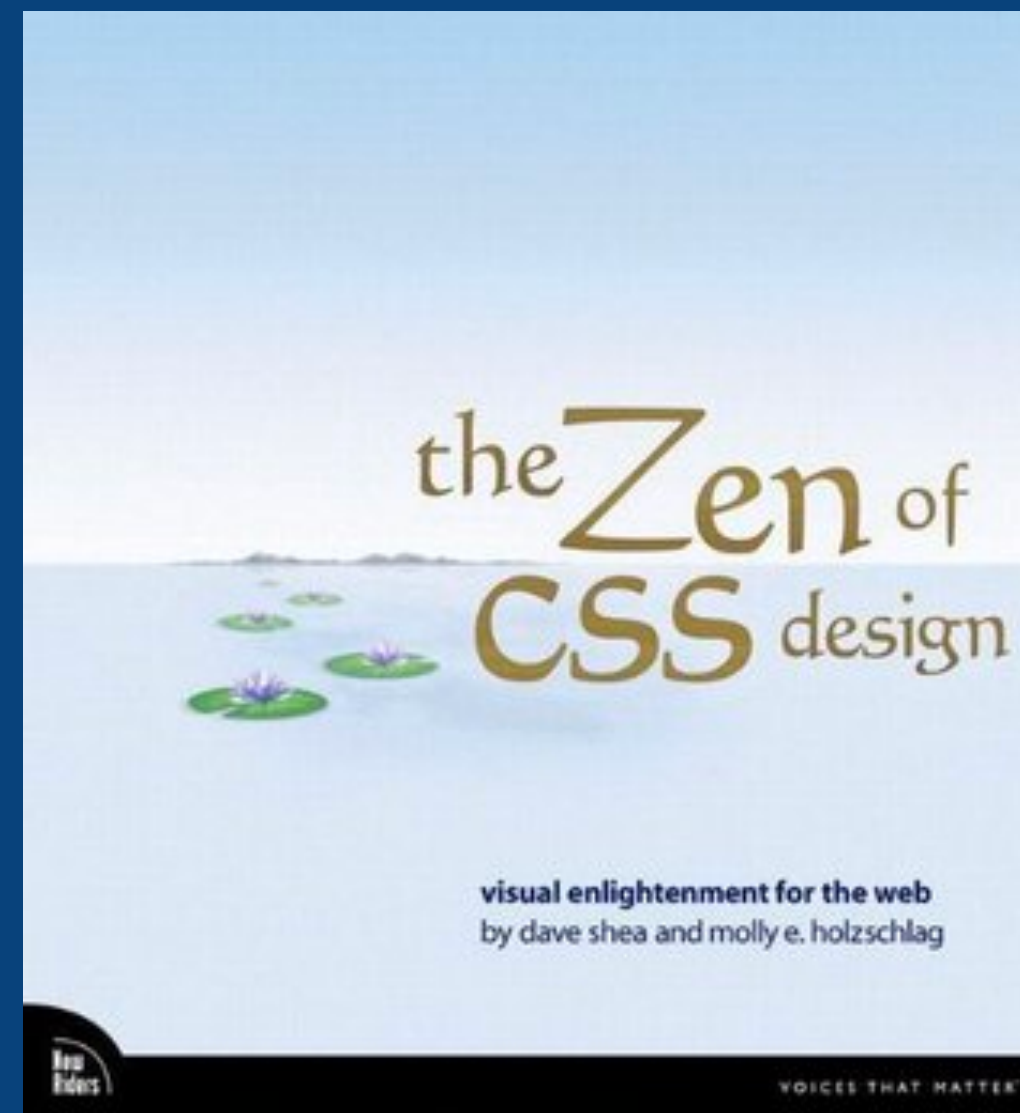


Implied relationships
can be disabled using:

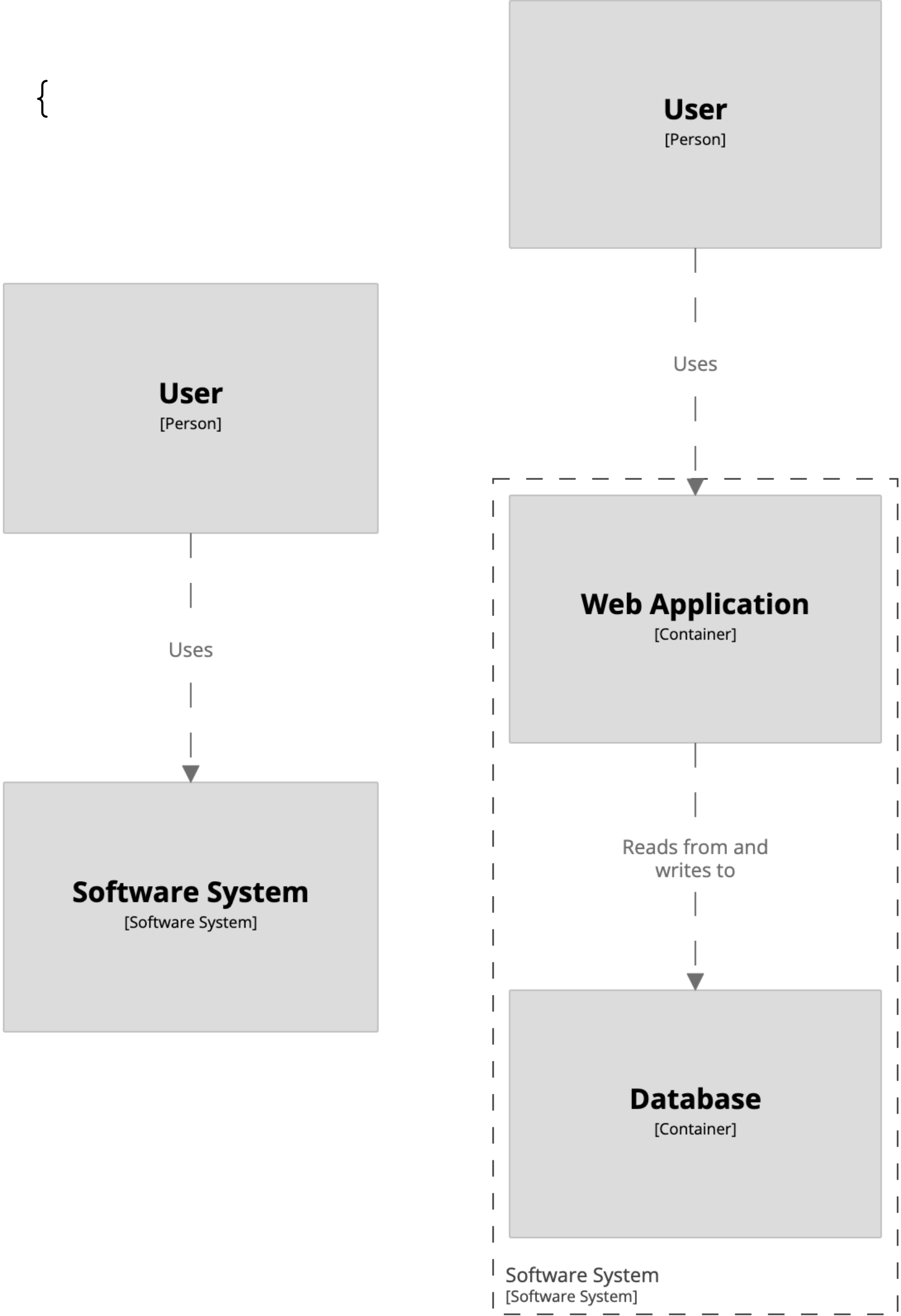
```
!impliedRelationships false
```

Separation of content and presentation

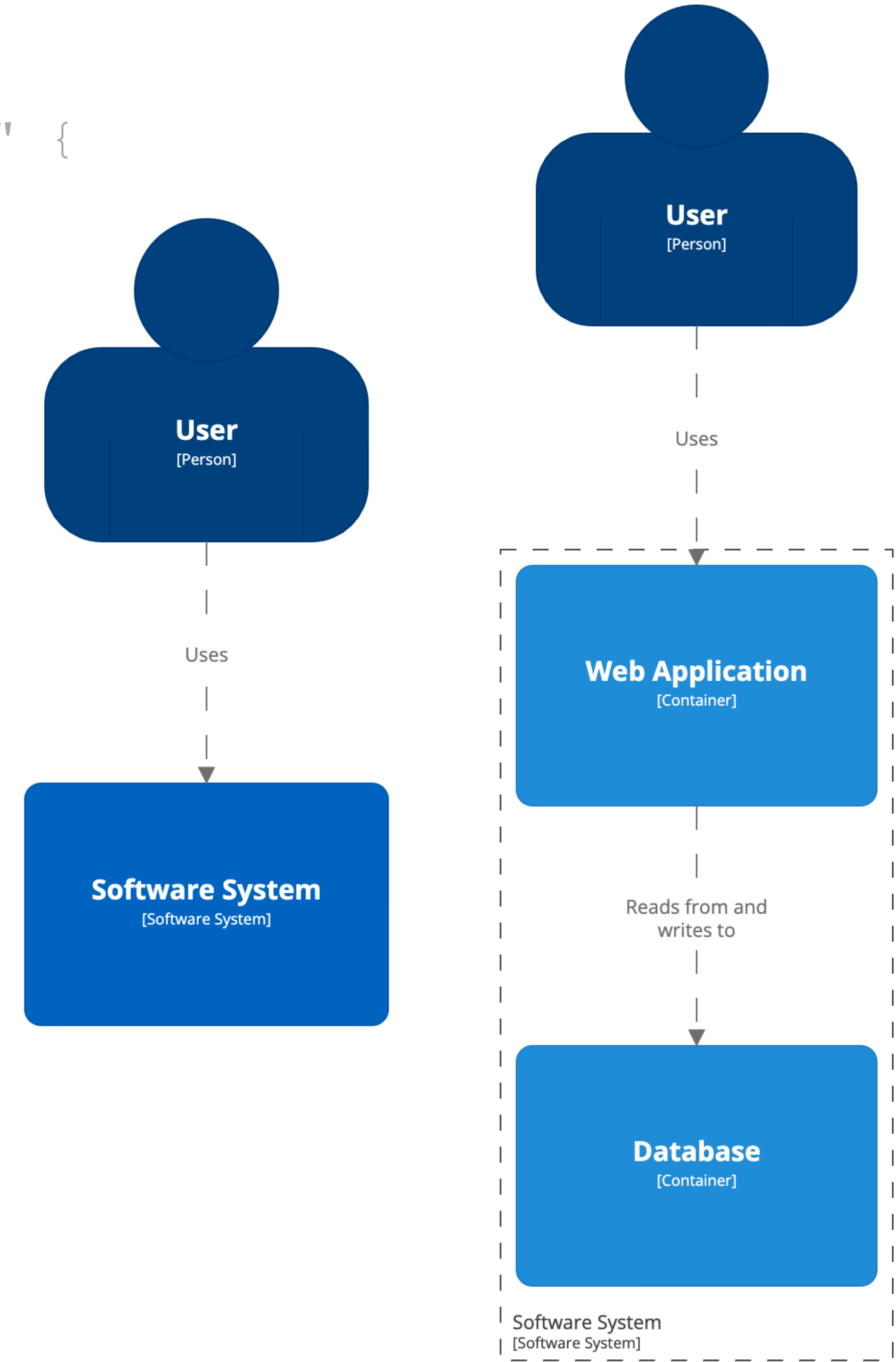
HTML & CSS

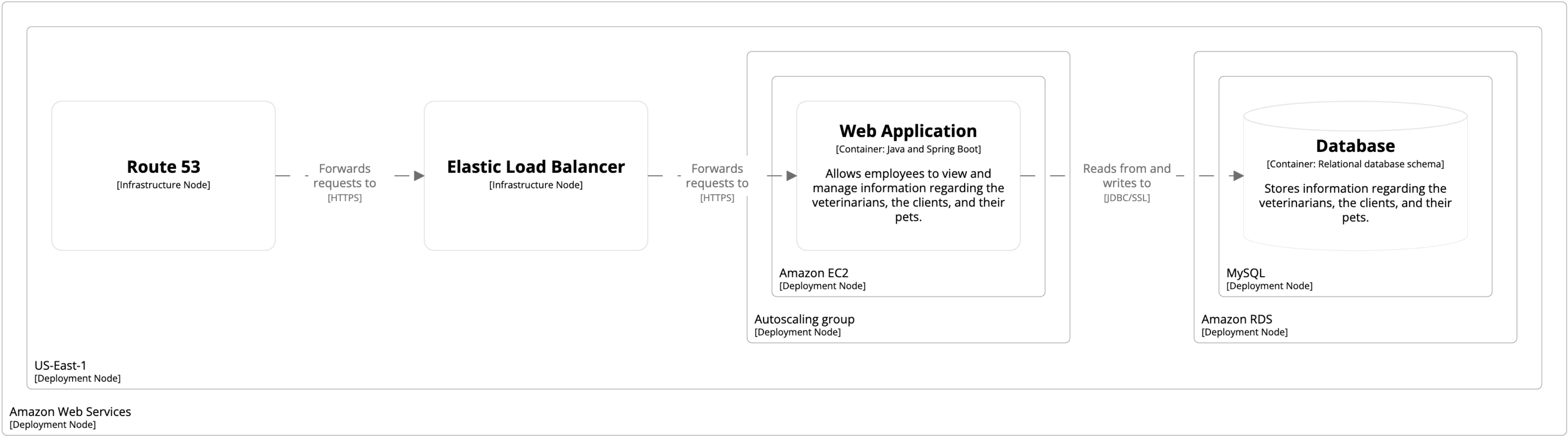


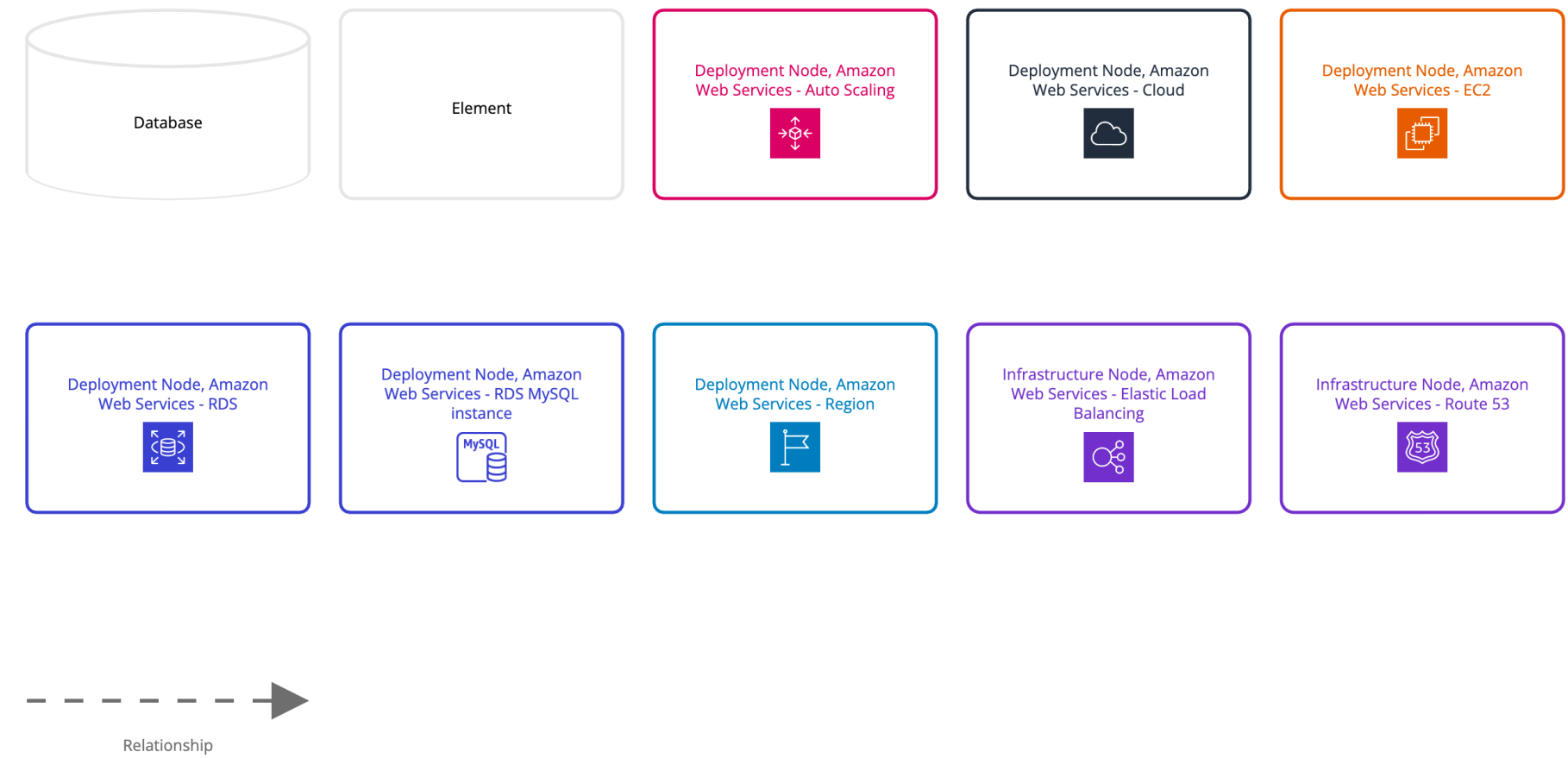
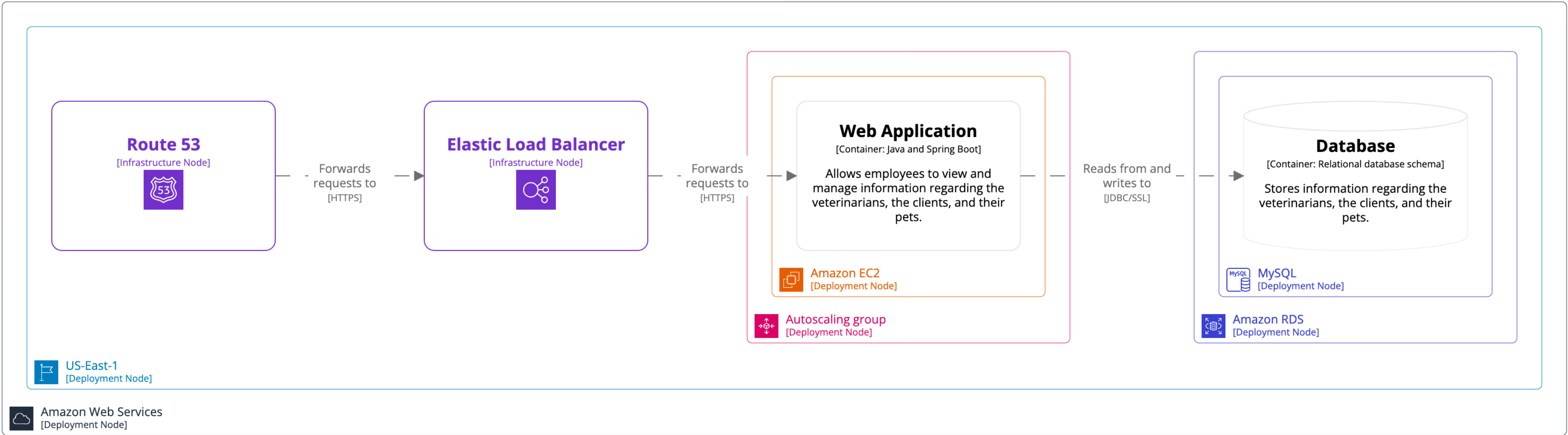
```
workspace {  
  
  model {  
    user = person "User"  
    softwareSystem = softwareSystem "Software System" {  
      webapp = container "Web Application"  
      database = container "Database"  
    }  
  
    user -> webapp "Uses"  
    webapp -> database "Reads from and writes to"  
  }  
  
  views {  
    systemContext softwareSystem {  
      include *  
      autoLayout  
    }  
  
    container softwareSystem {  
      include *  
      autolayout  
    }  
  }  
}
```




```
workspace {  
  
  model {  
    user = person "User"  
    softwareSystem = softwareSystem "Software System" {  
      webapp = container "Web Application"  
      database = container "Database"  
    }  
  
    user -> webapp "Uses"  
    webapp -> database "Reads from and writes to"  
  }  
  
  views {  
    systemContext softwareSystem {  
      include *  
      autoLayout  
    }  
  
    container softwareSystem {  
      include *  
      autolayout  
    }  
  
    theme default  
  }  
}
```







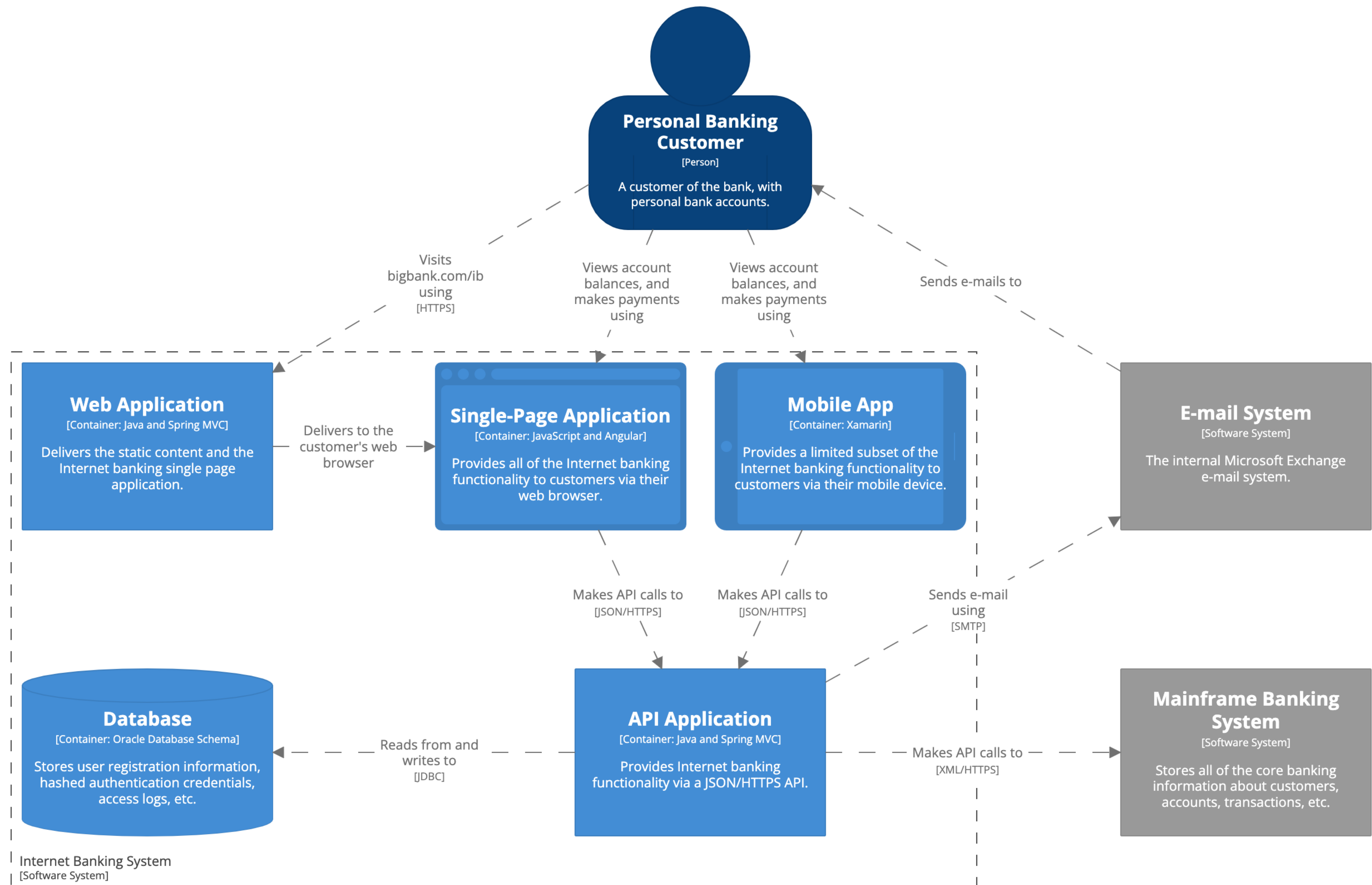
Styling of elements and
relationships is achieved
via tags

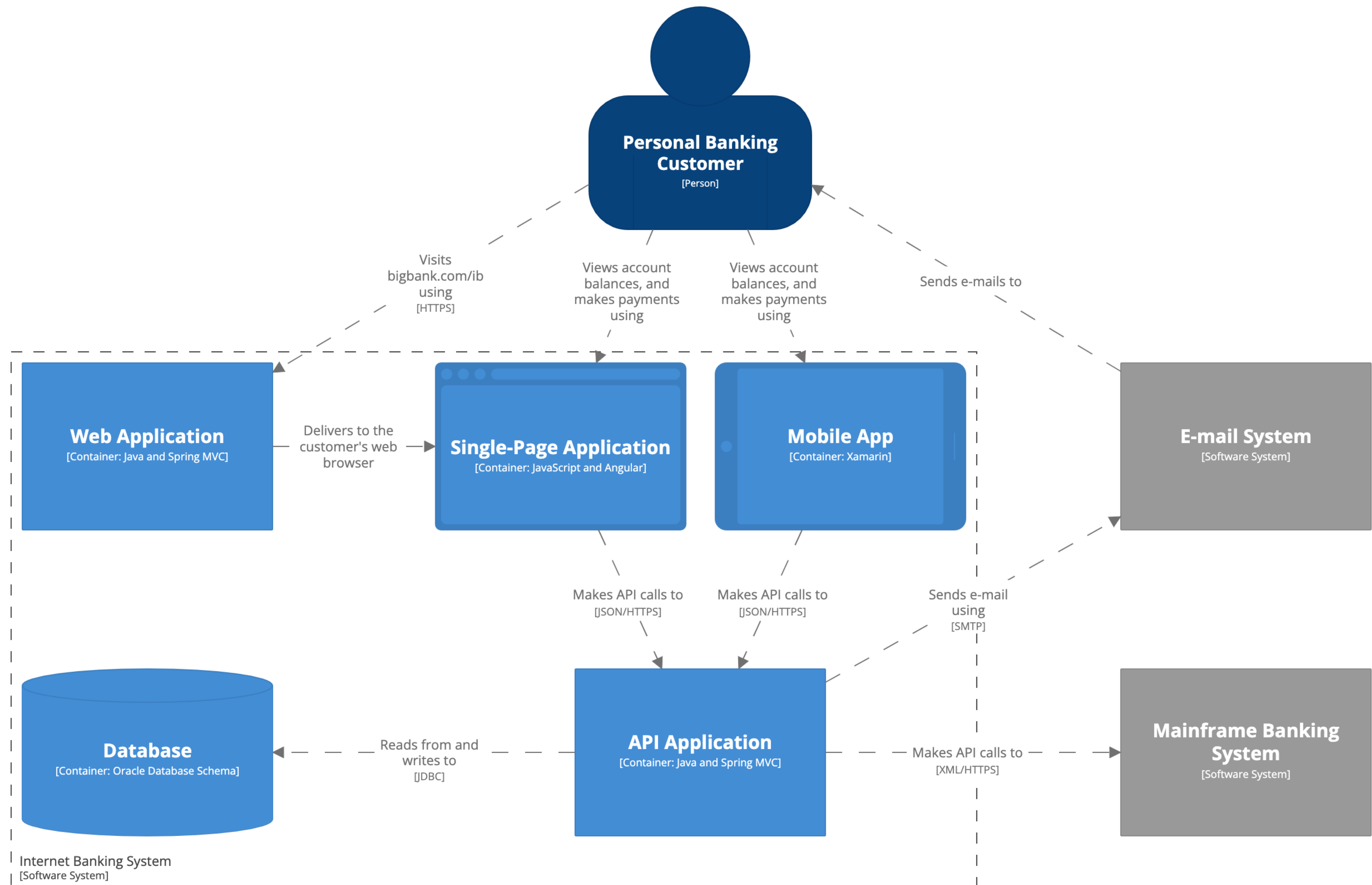
```
workspace {  
  
    model {  
        softwareSystem "Software System"  
    }  
  
    views {  
        systemLandscape {  
            include *  
            autolayout  
        }  
    }  
  
}  
  
}
```




```
workspace {  
  
    model {  
        softwareSystem "Software System"  
    }  
  
    views {  
        systemLandscape {  
            include *  
            autolayout  
        }  
  
        styles {  
            element "Software System" {  
                background #1168bd  
                color #ffffff  
                shape RoundedBox  
            }  
        }  
    }  
}
```







**Rendering tool
independent**

“Diagrams as code 1.0”

PlantUML, Mermaid, etc are **input formats**

“Diagrams as code 2.0”

PlantUML, Mermaid, etc are **output formats**

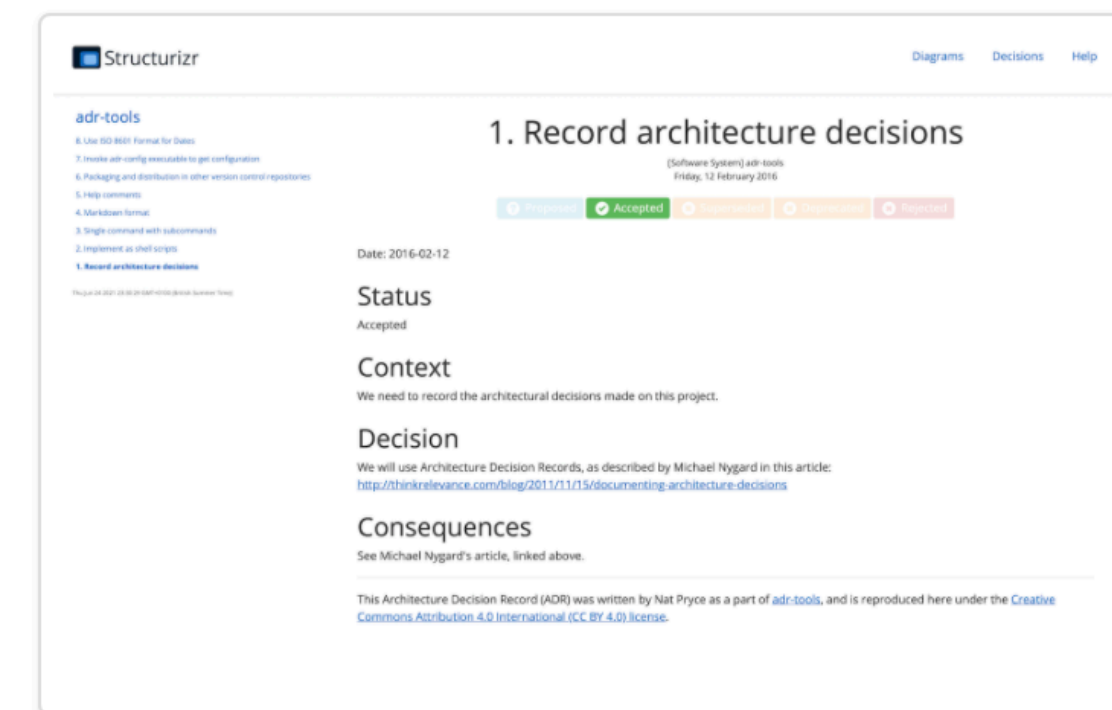
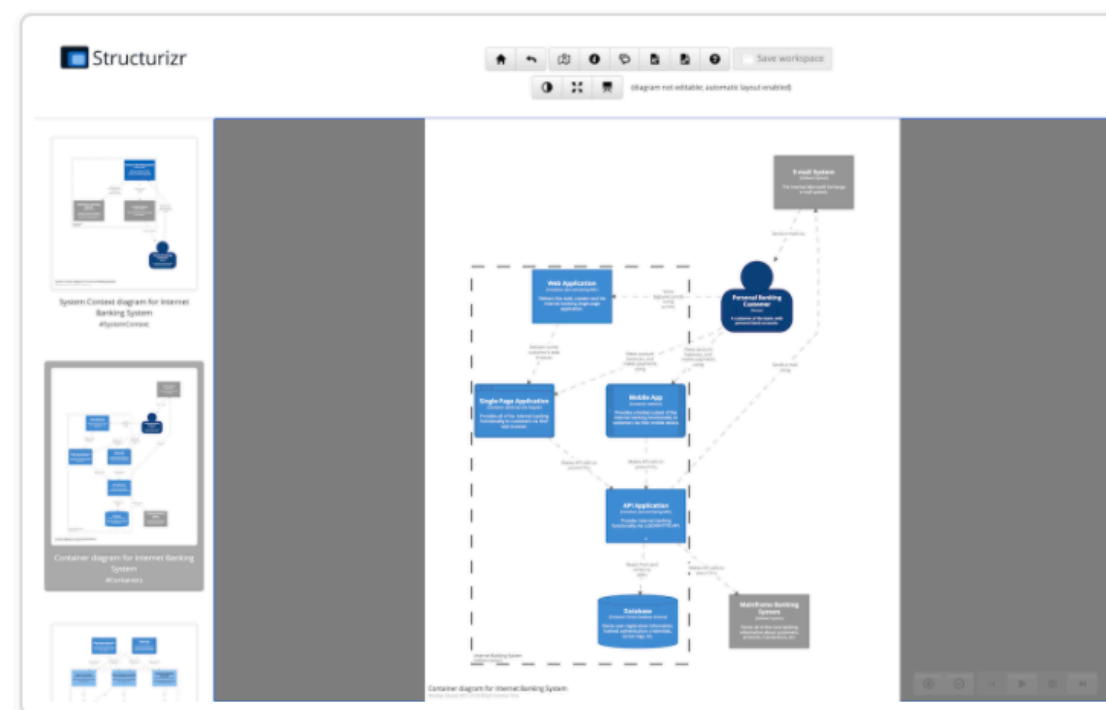
Automatic layout vs manual layout?

Structurizr Lite

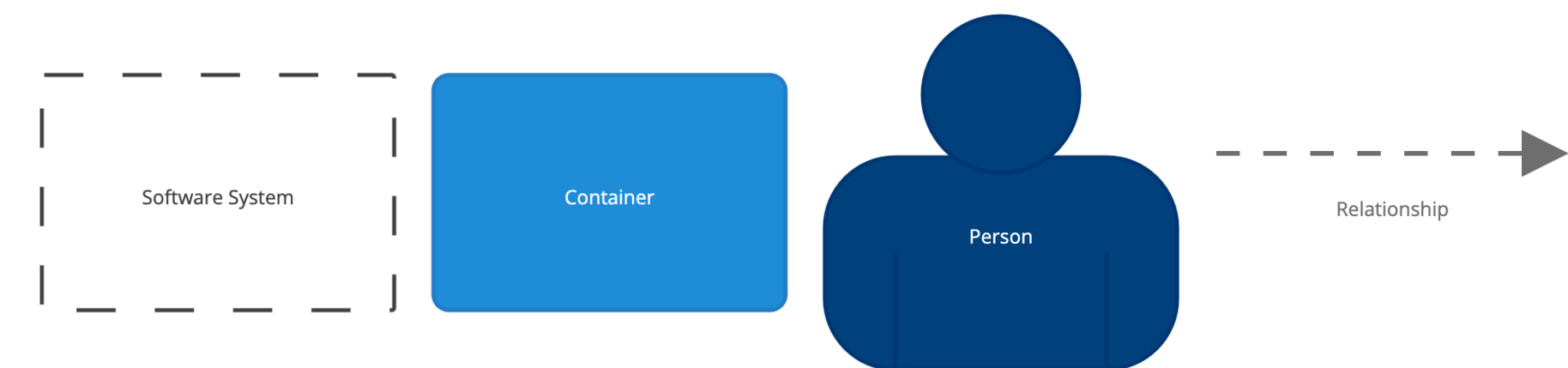
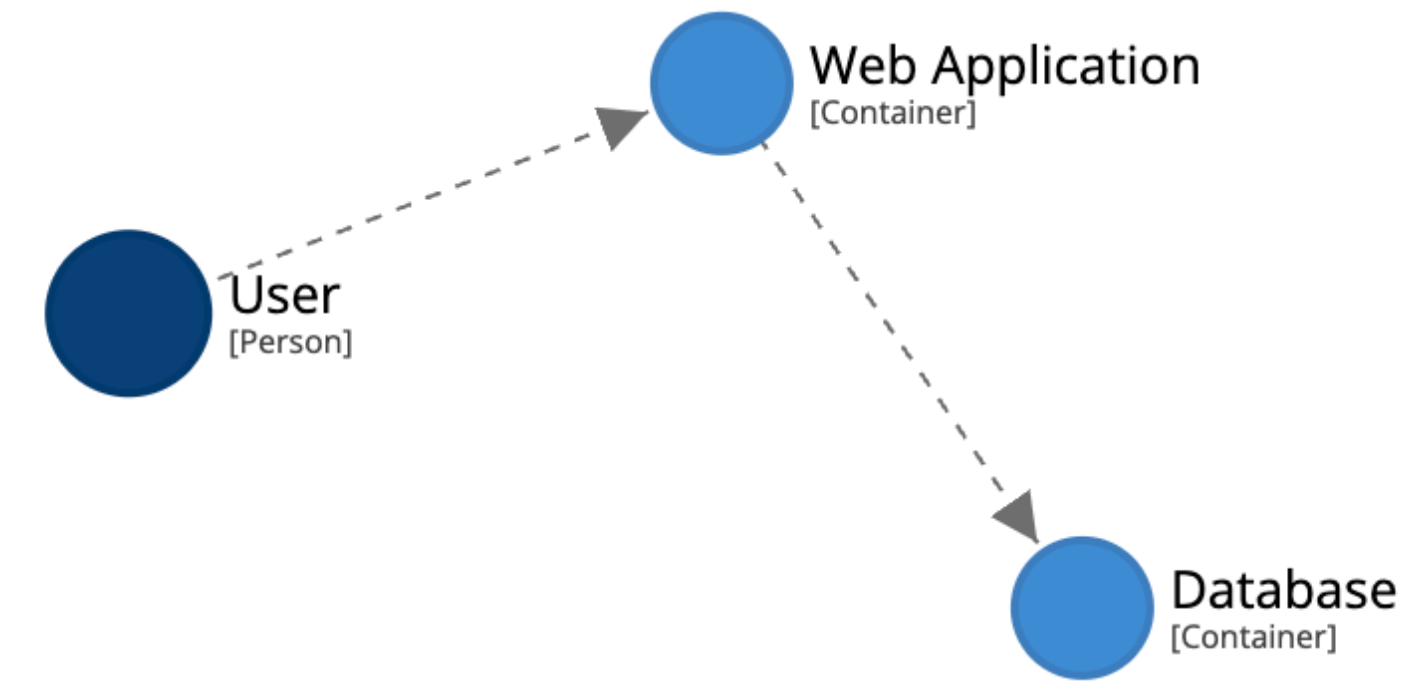
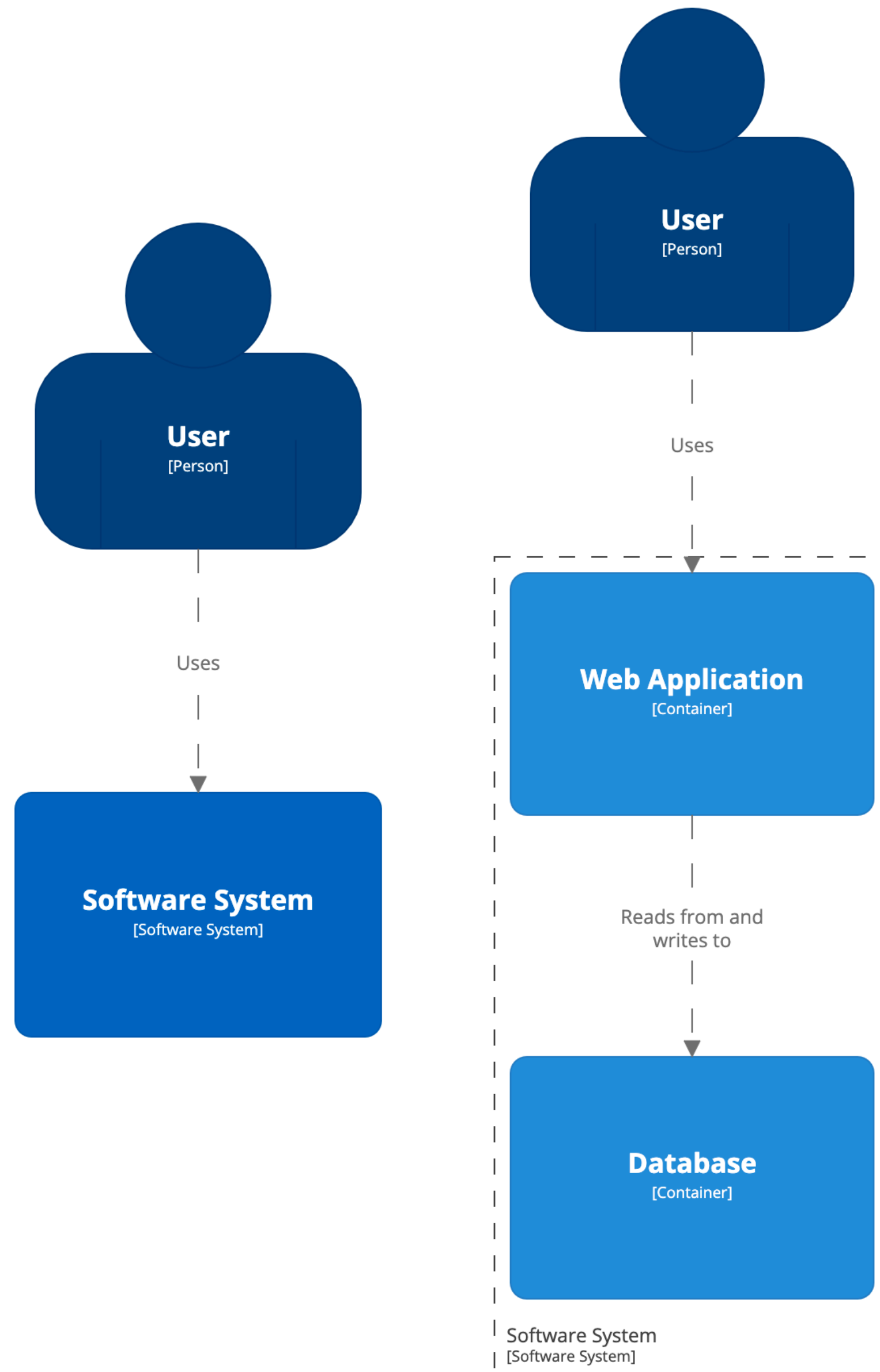
[Overview](#) | [Getting started](#) | [Auto-sync](#) | [Workflow](#) | [Docker Hub](#)

Overview

Packaged as a Docker container, and designed for developers, this version of Structurizr provides a way to quickly work with a single workspace. It's free to use, and allows you to view/edit diagrams, view documentation, and view architecture decision records defined in a DSL or JSON workspace.



Structurizr Lite will look for a `workspace.dsl` and `workspace.json` file in a given directory, in that order, and use the file it finds first. If you change this file (e.g. via your text editor or one of the Structurizr client libraries), you can refresh your web browser to immediately see the changes.



structurizr / cli Public

Unwatch 10


Unstar 251

Fork 30

<> Code Issues 2 Pull requests Actions Projects Wiki Security Insights Settings

master 1 branch 31 tags


Go to file Add file Code

 **simonbrowndotje** Merge pull request #49 from venthur/patch-1 ... ✓ 0ed8450 3 days ago 🕒 153 commits

📁 .github/workflows	Create gradle.yml	10 months ago
📁 docs	Updated to reflect release.	3 days ago
📁 etc	Updated DSL library and changed packaging from Spring Boot to a re...	last month
📁 examples	Split the Big Bank plc example into two workspaces ... "System Lands...	6 months ago
📁 gradle/wrapper	Updated DSL library and changed packaging from Spring Boot to a re...	last month
📁 src/main	Added a more helpful error message when encountering an empty DS...	3 days ago
📄 .gitignore	Include docs and ADRs for the examples via the DSL.	6 months ago
📄 Dockerfile	Updated to reflect release.	3 days ago
📄 LICENSE	Initial commit	16 months ago
📄 README.md	Added missing link to list command in README	last month
📄 build.gradle	Updated to reflect release.	3 days ago
📄 gradlew	Initial commit.	16 months ago
📄 gradlew.bat	Updated DSL library and changed packaging from Spring Boot to a re...	last month
📄 settings.gradle	Modified structurizr-publish tool into a more general cli tool, suppo...	16 months ago

About ⚙️

A command line utility for Structurizr.

 [structurizr.com](#)

markdown plantuml asciidoc

software-architecture architecture-doc

architecture-decision-records

structurizr c4model adrs

websequencediagrams

architecture-diagrams ilograph

 Readme

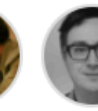

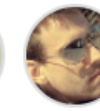



 Apache-2.0 License

Releases 30

 **v1.15.0** Latest 3 days ago

+ 29 releases

Contributors 6

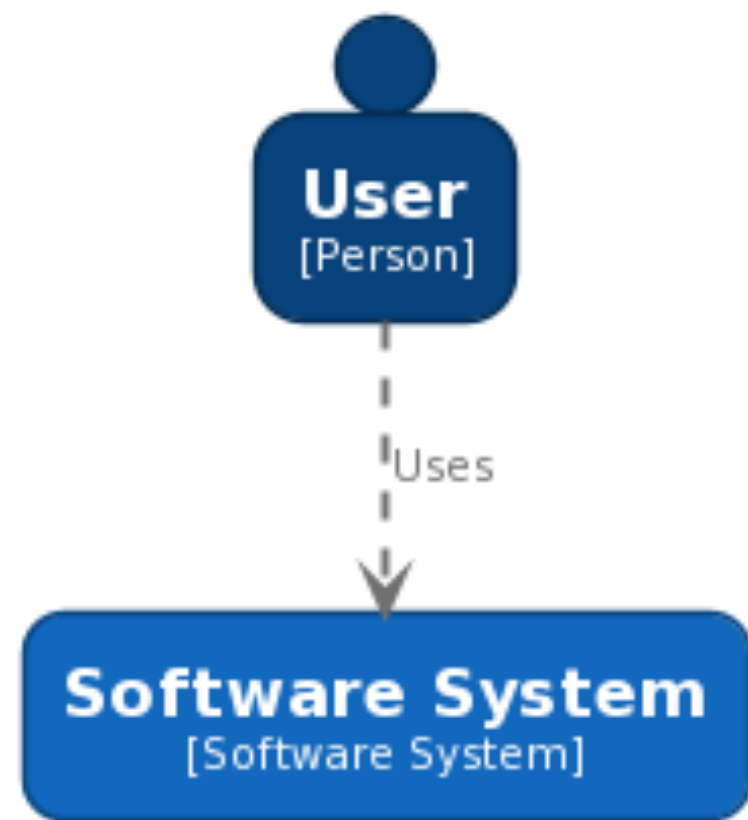


```
./structurizr.sh export -workspace /Users/simon/bigbankplc/workspace.dsl -format plantuml
```

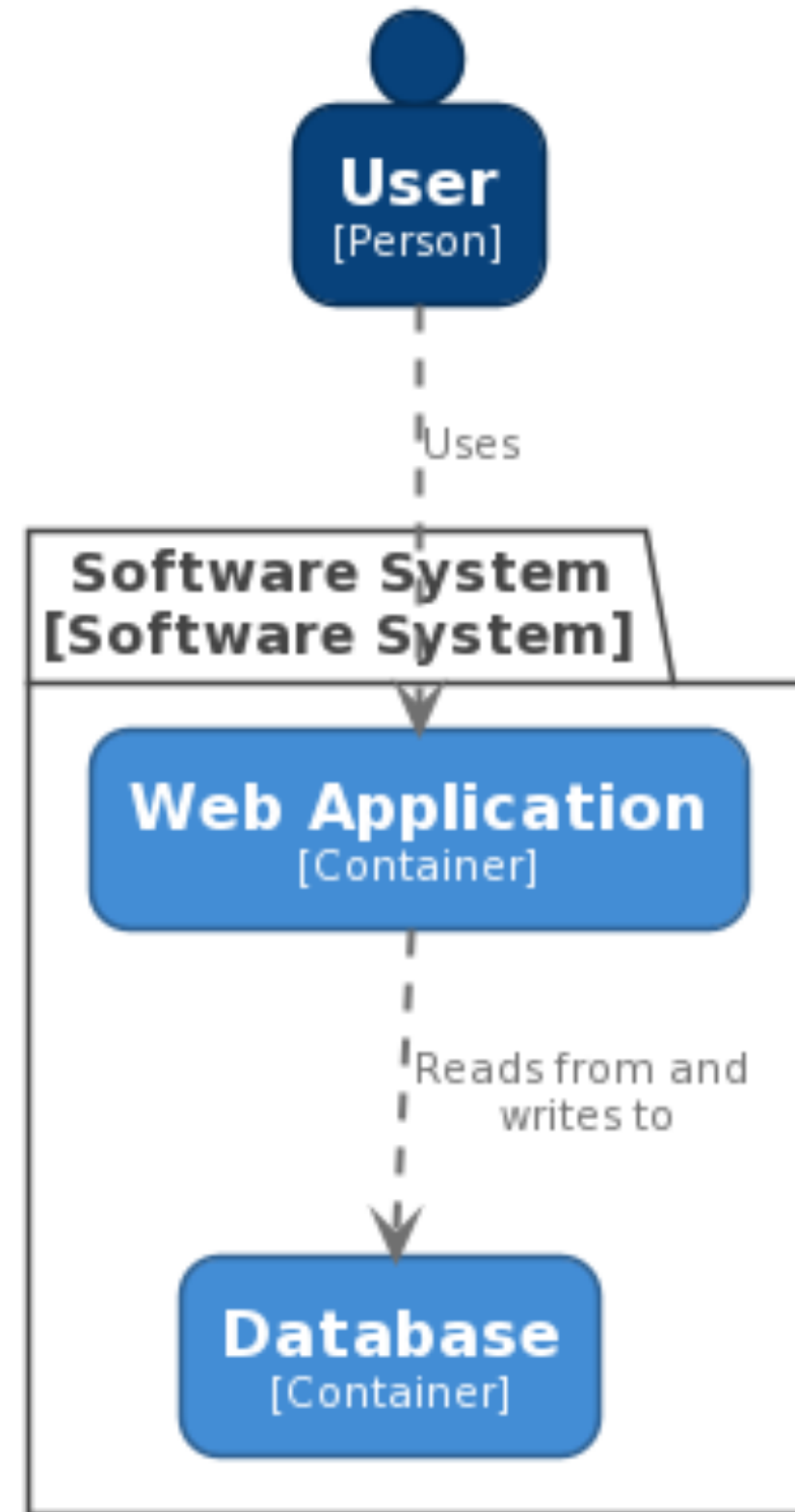
```
Exporting workspace from /Users/simon/bigbankplc/workspace.dsl
```

- loading workspace from DSL
- using StructurizrPlantUMLExporter
- writing /Users/simon/bigbankplc/structurizr-SystemLandscape.puml
- writing /Users/simon/bigbankplc/structurizr-SystemContext.puml
- writing /Users/simon/bigbankplc/structurizr-Containers.puml
- writing /Users/simon/bigbankplc/structurizr-Components.puml
- writing /Users/simon/bigbankplc/structurizr-SignIn.puml
- writing /Users/simon/bigbankplc/structurizr-LiveDeployment.puml
- writing /Users/simon/bigbankplc/structurizr-DevelopmentDeployment.puml
- writing /Users/simon/bigbankplc/structurizr-SignIn-sequence.puml
- finished

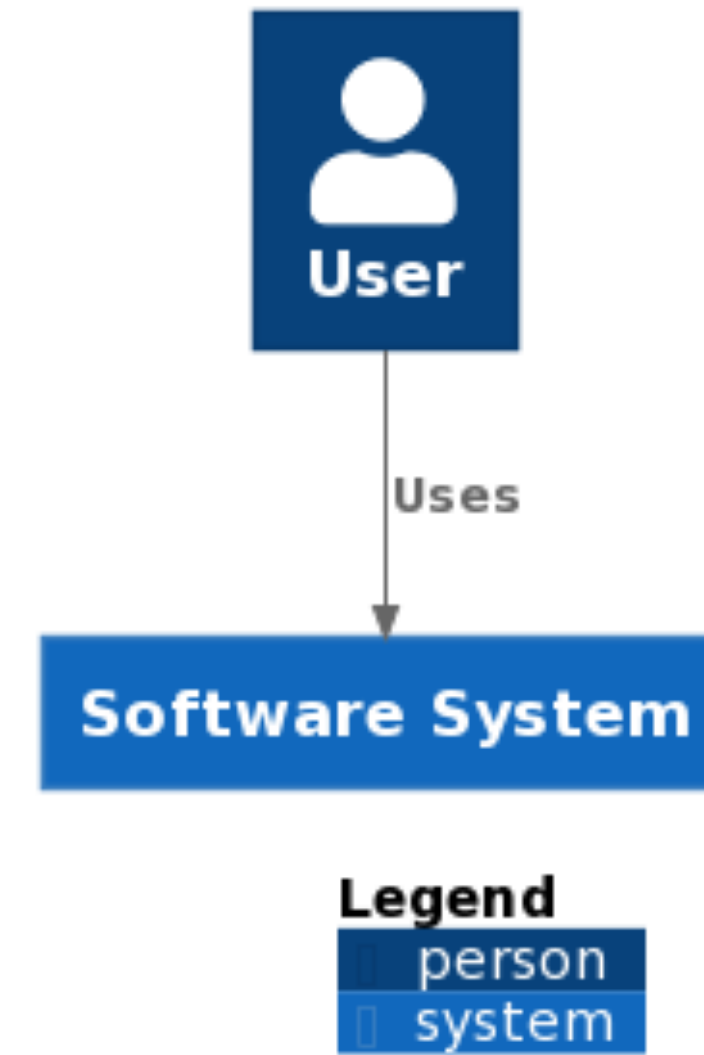
Software System - System Context



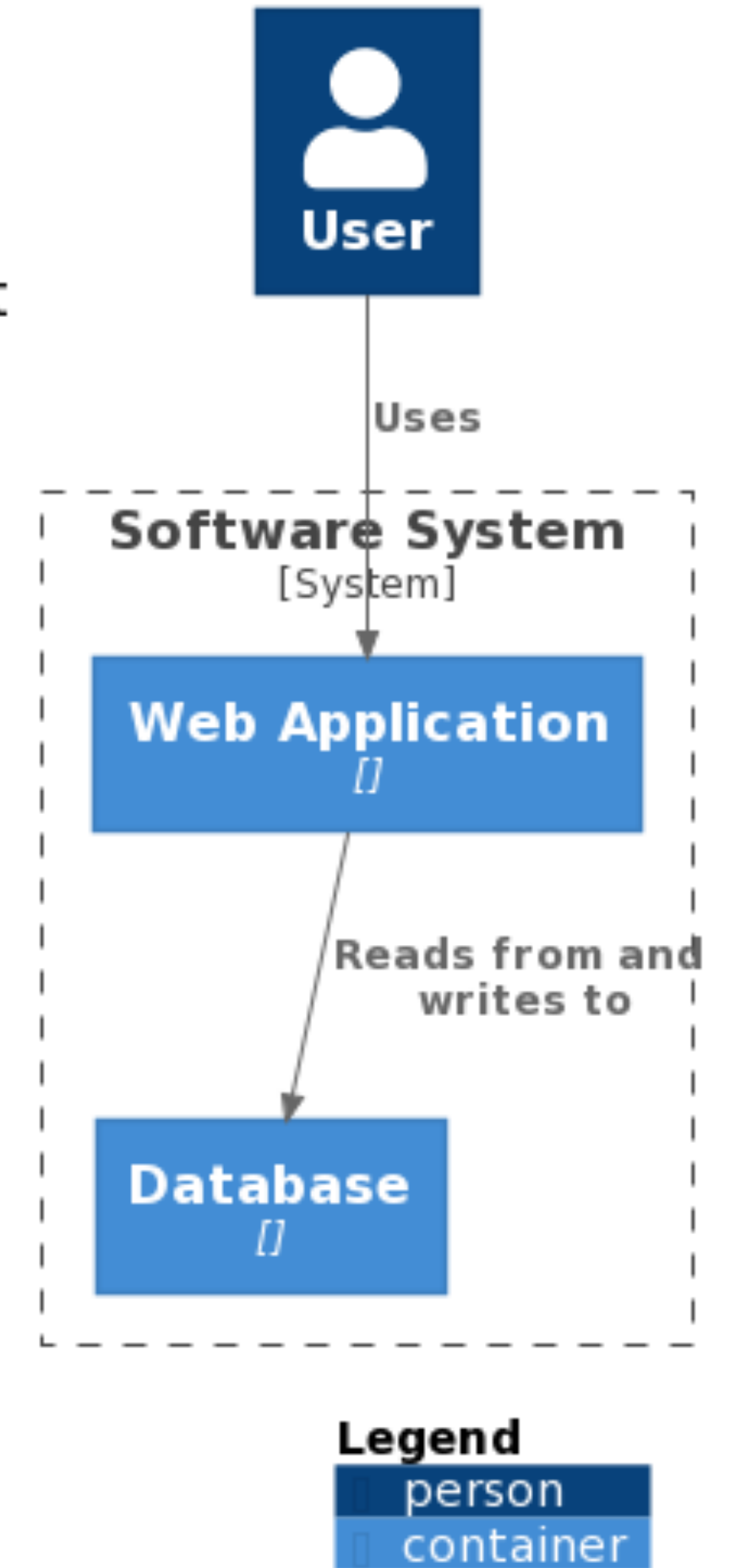
Software System - Containers

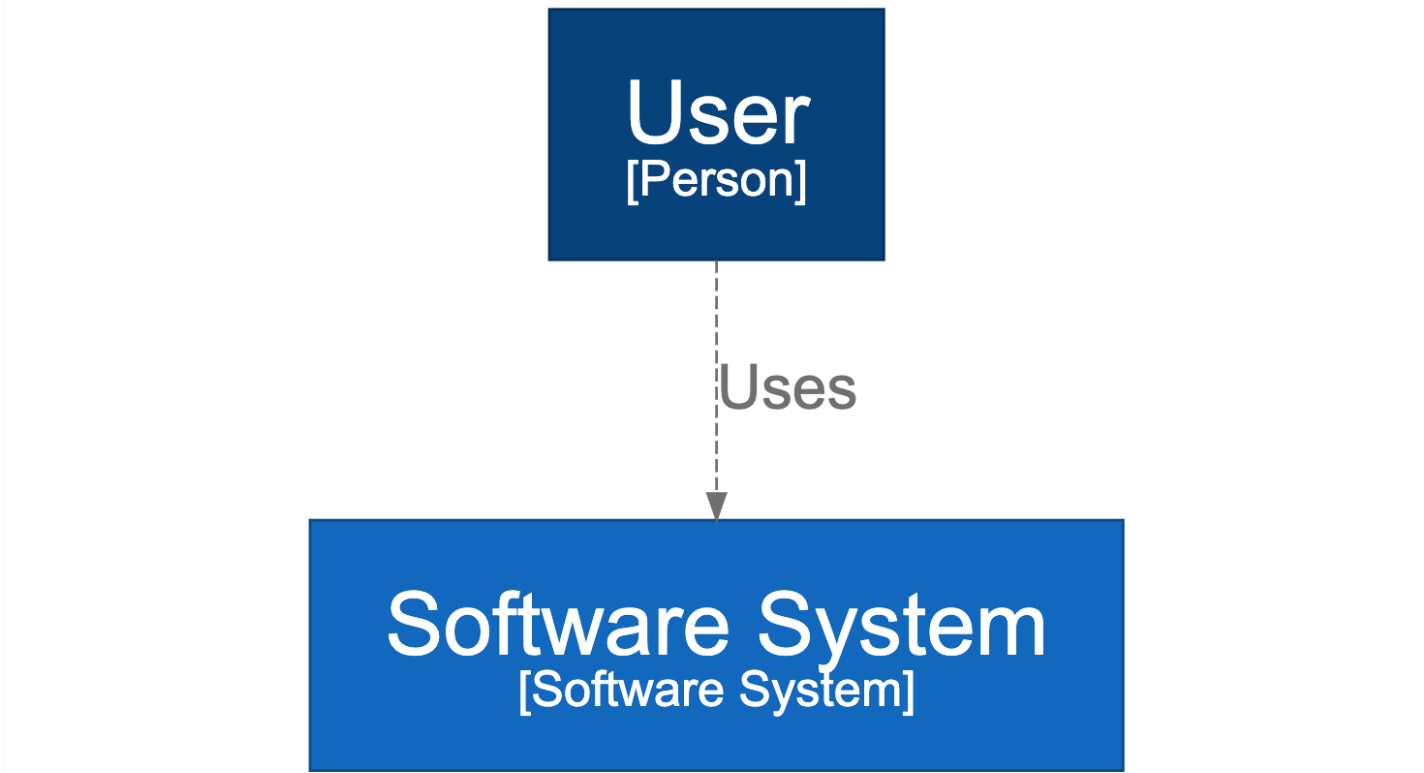


Software System - System Context

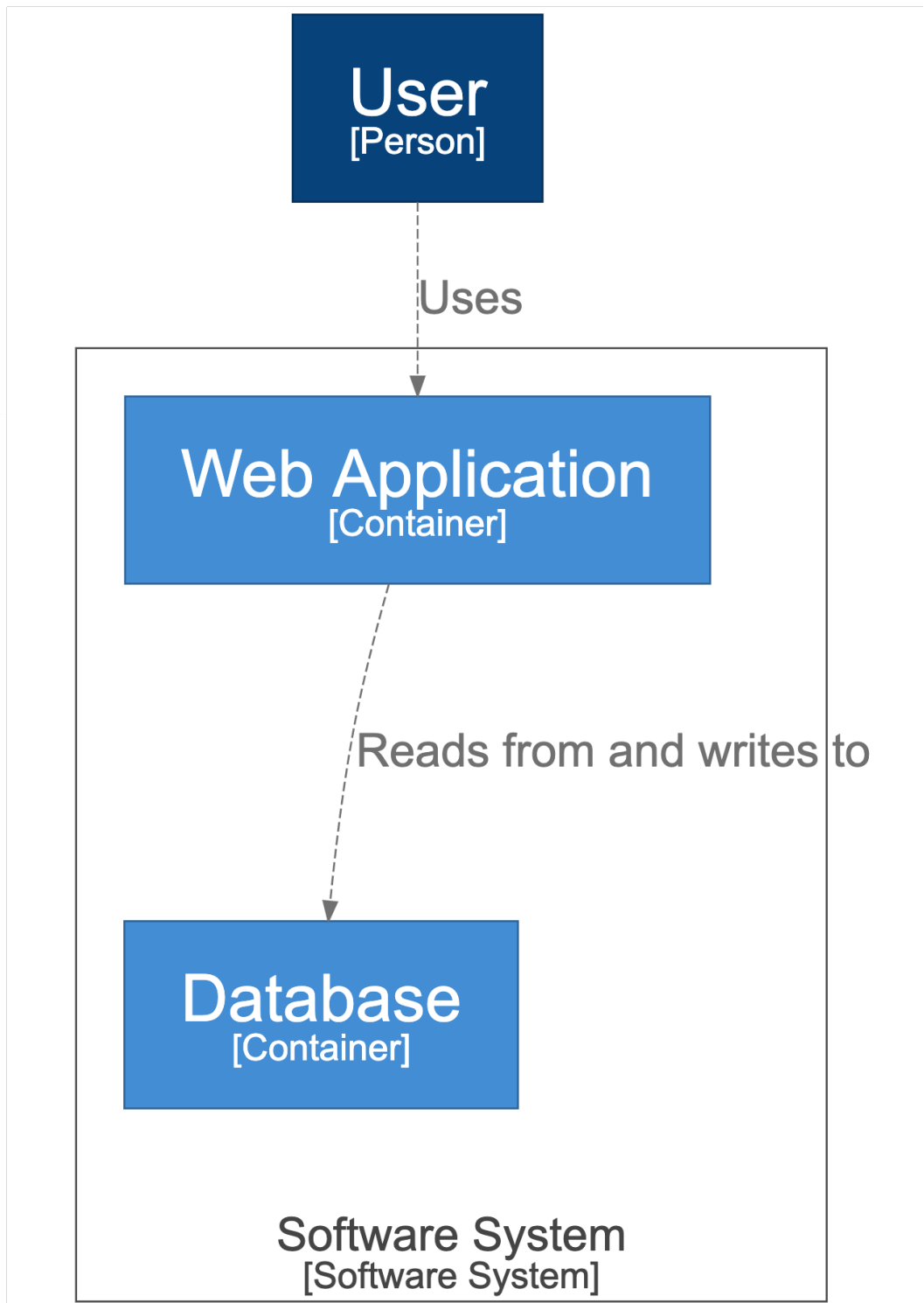


Software System - Containers

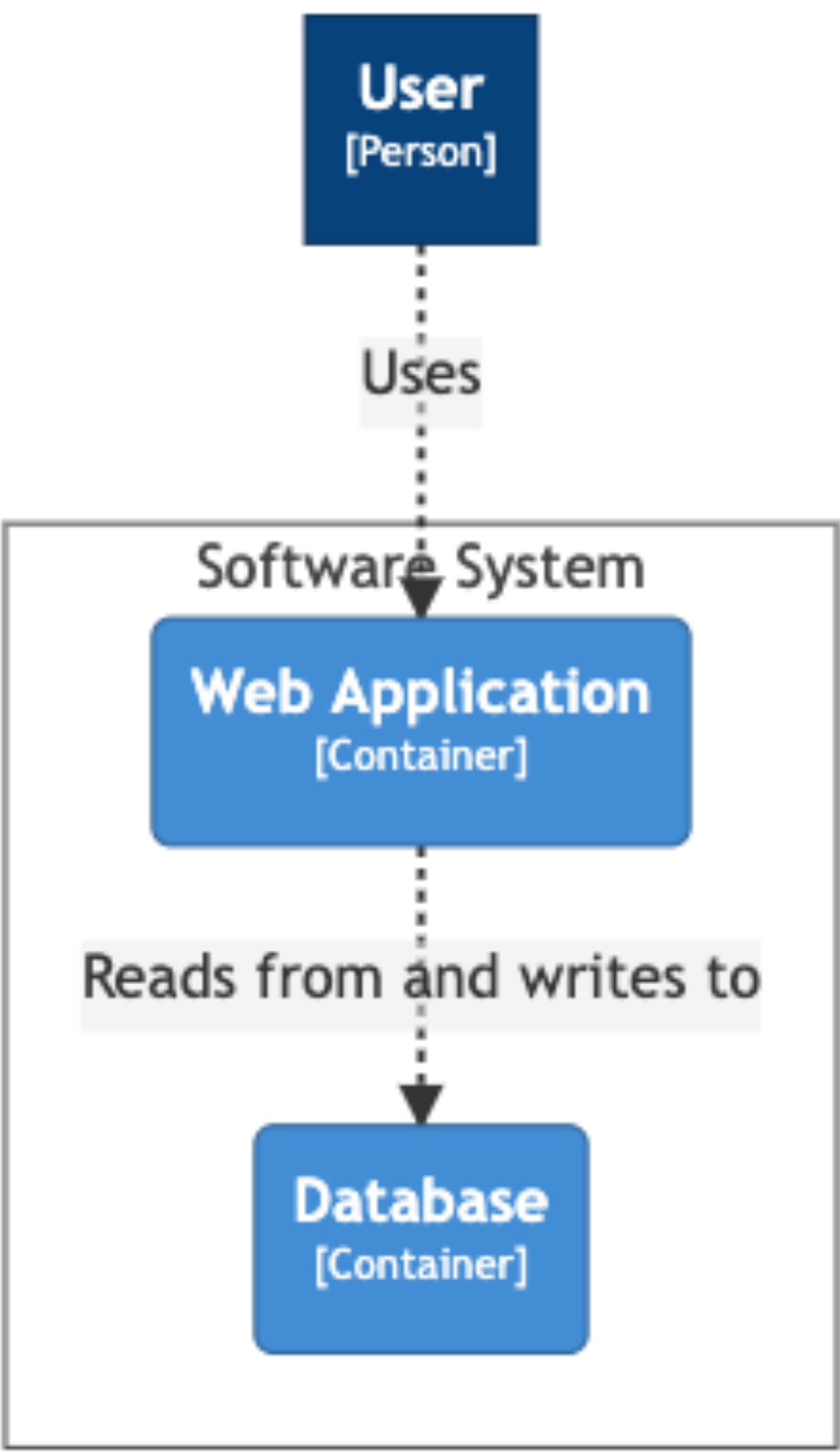
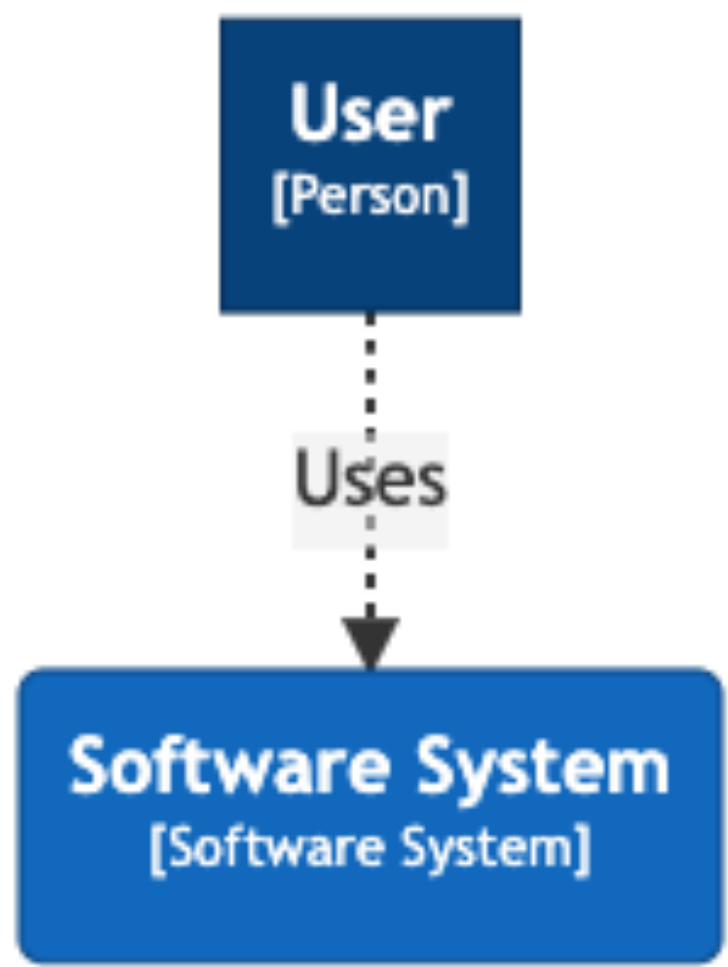


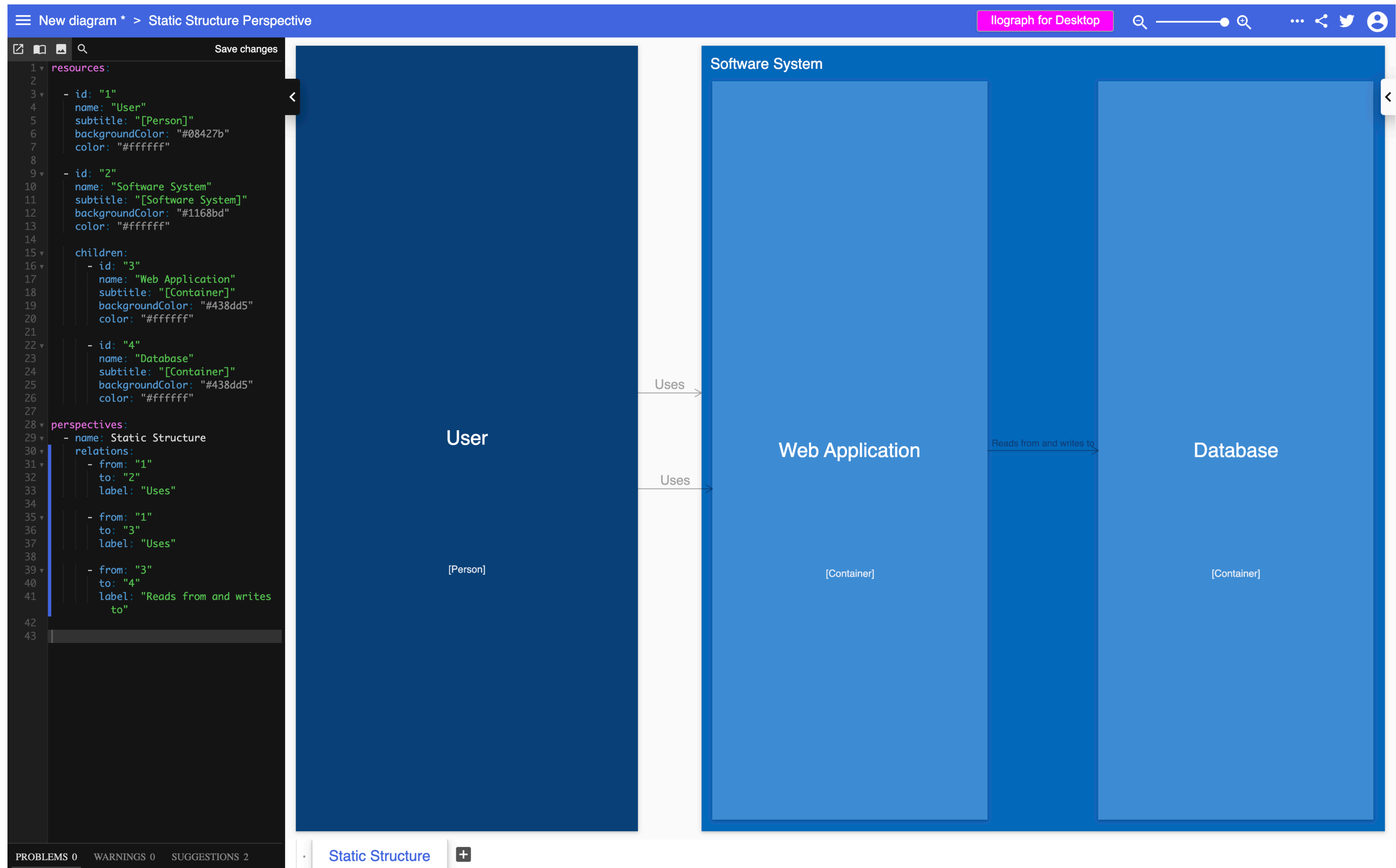


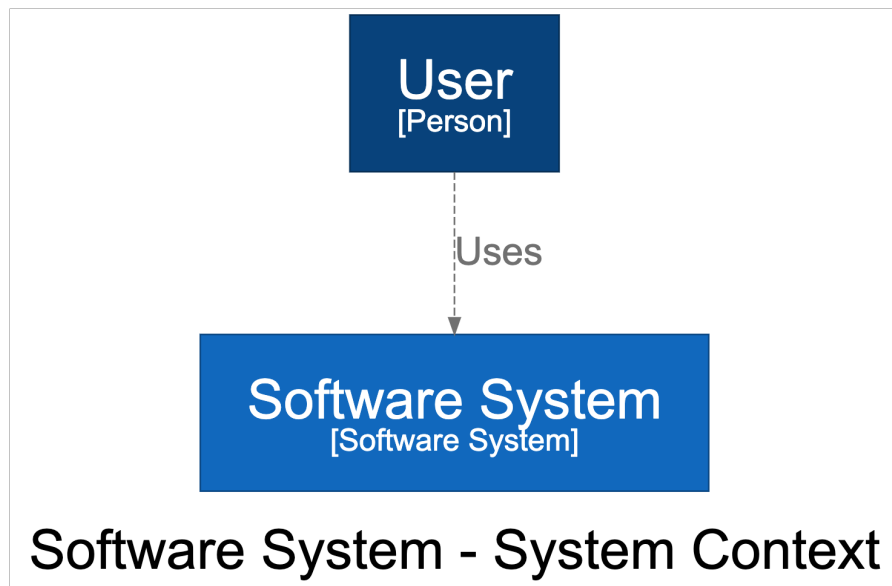
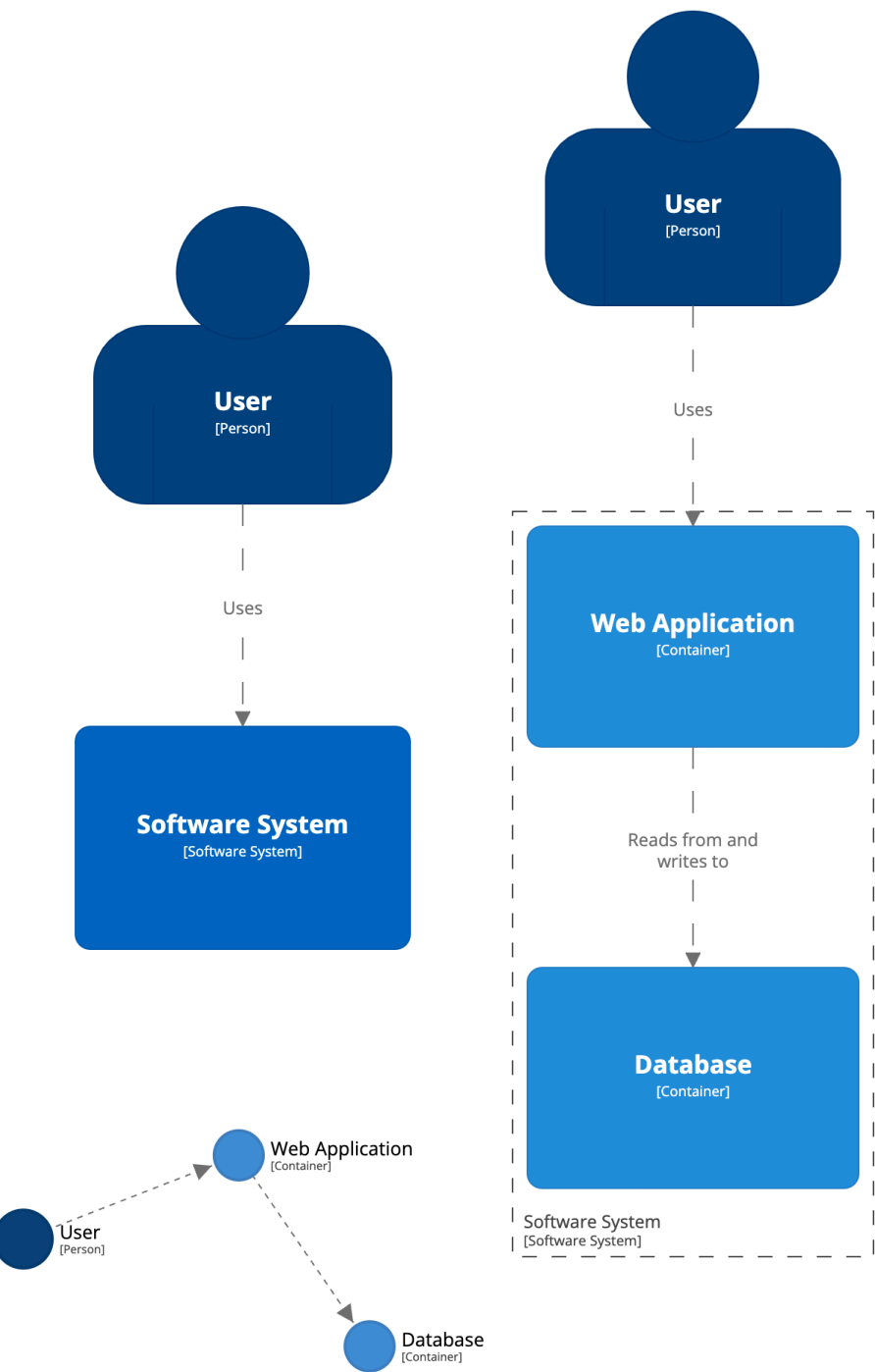
Software System - System Context



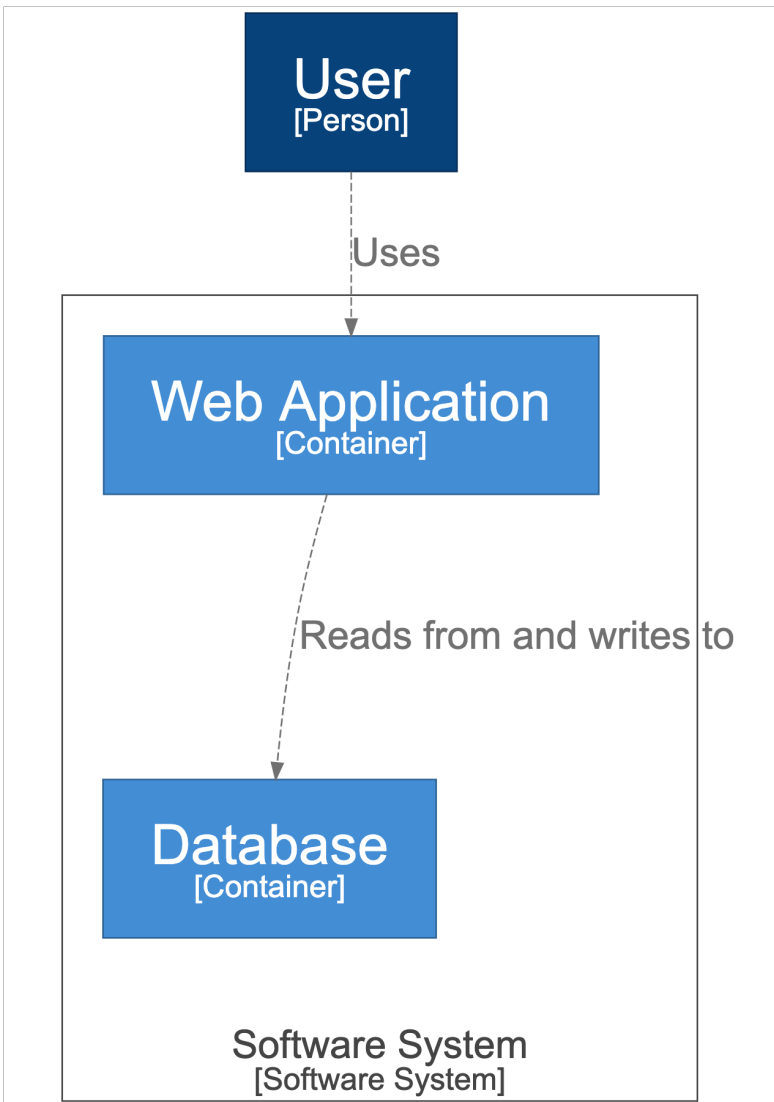
Software System - Containers





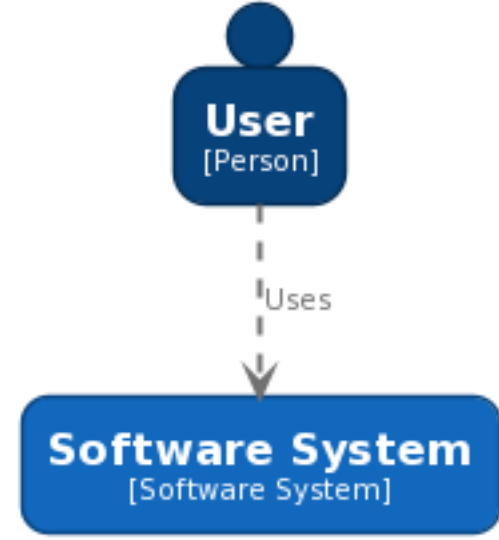


Software System - System Context

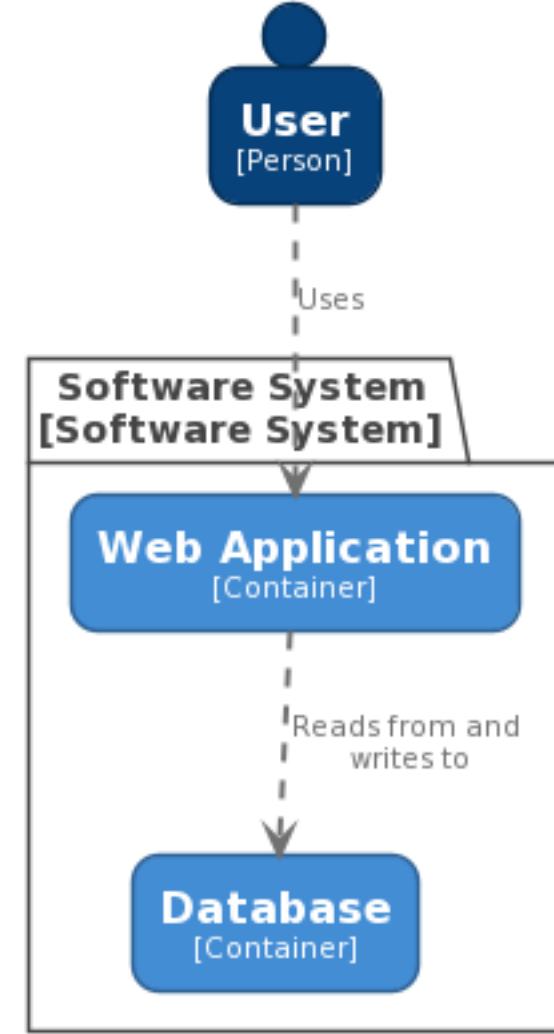


Software System - Containers

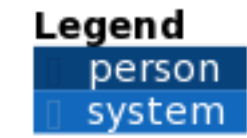
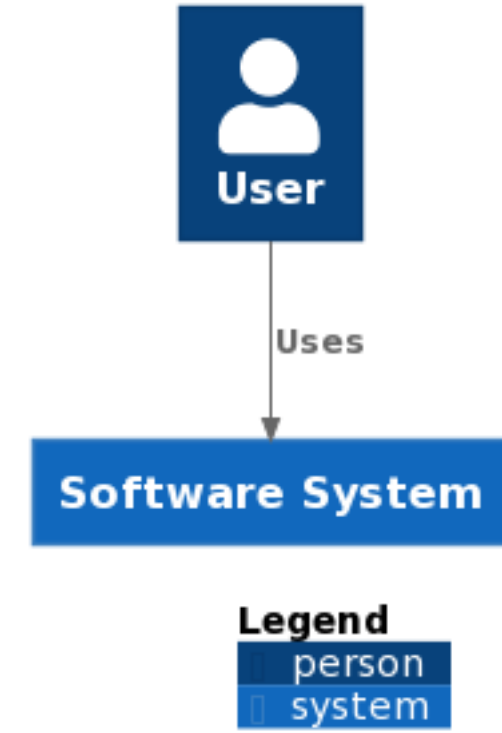
Software System - System Context



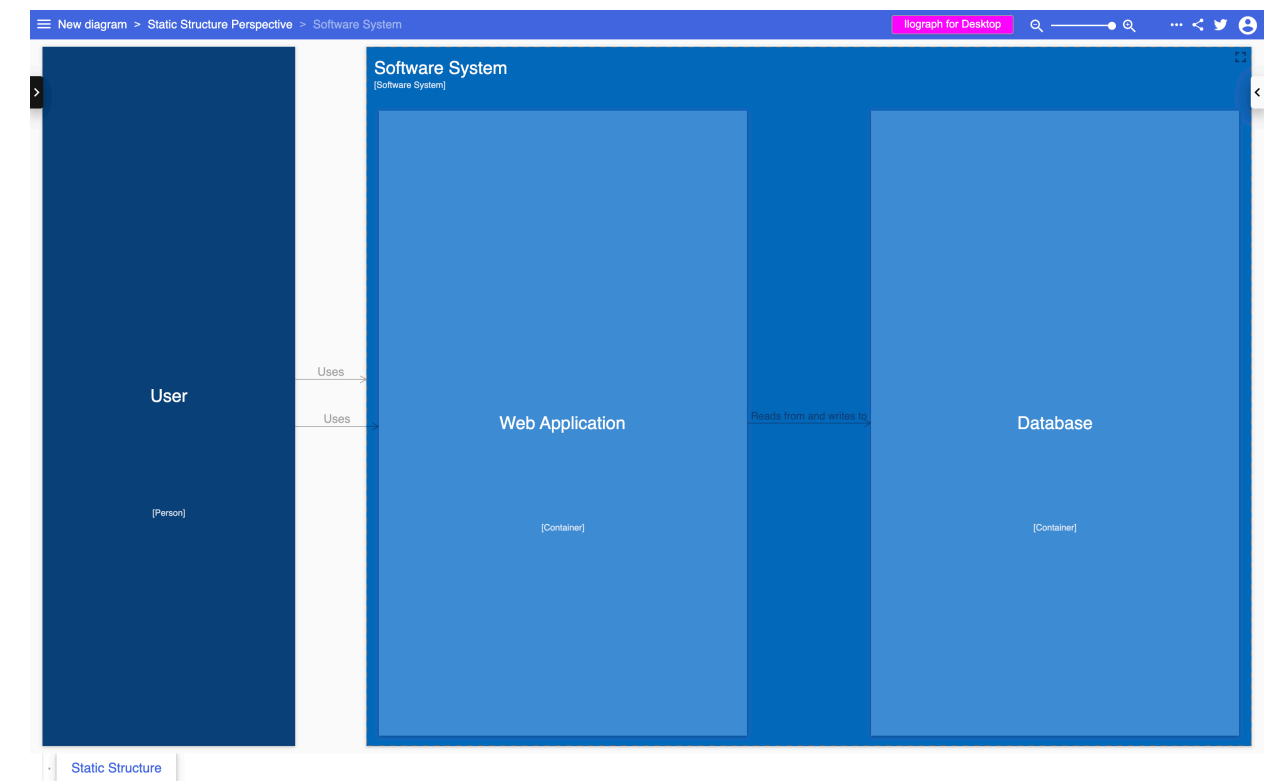
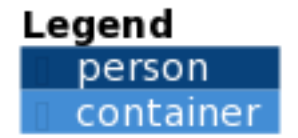
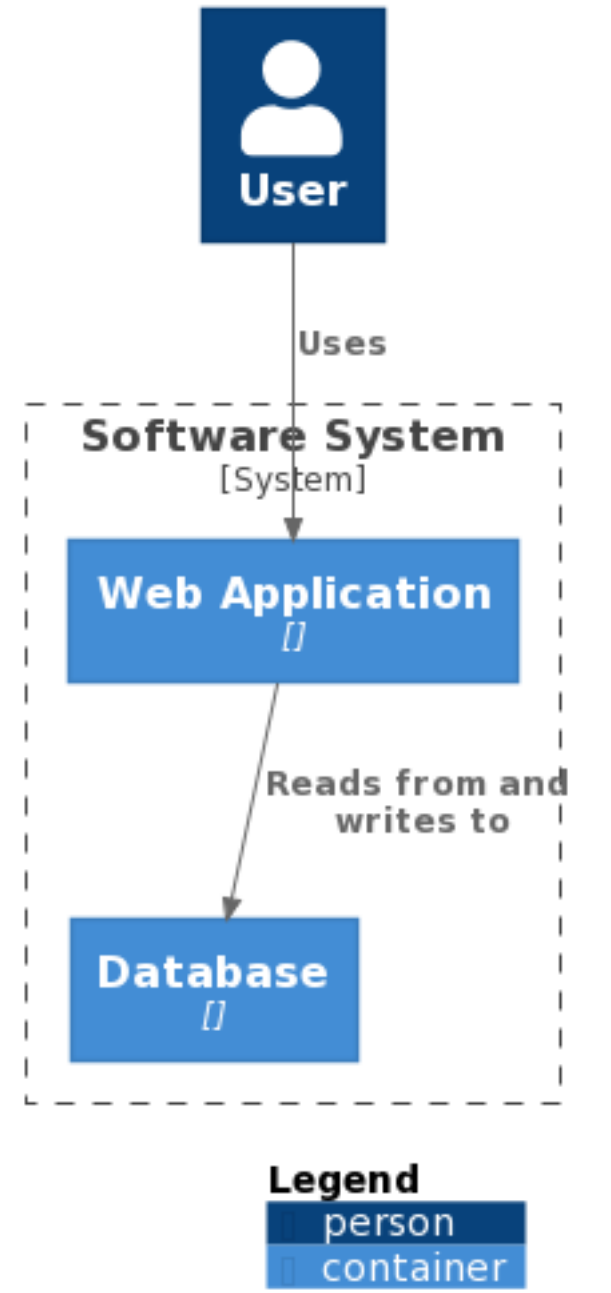
Software System - Containers



Software System - System Context

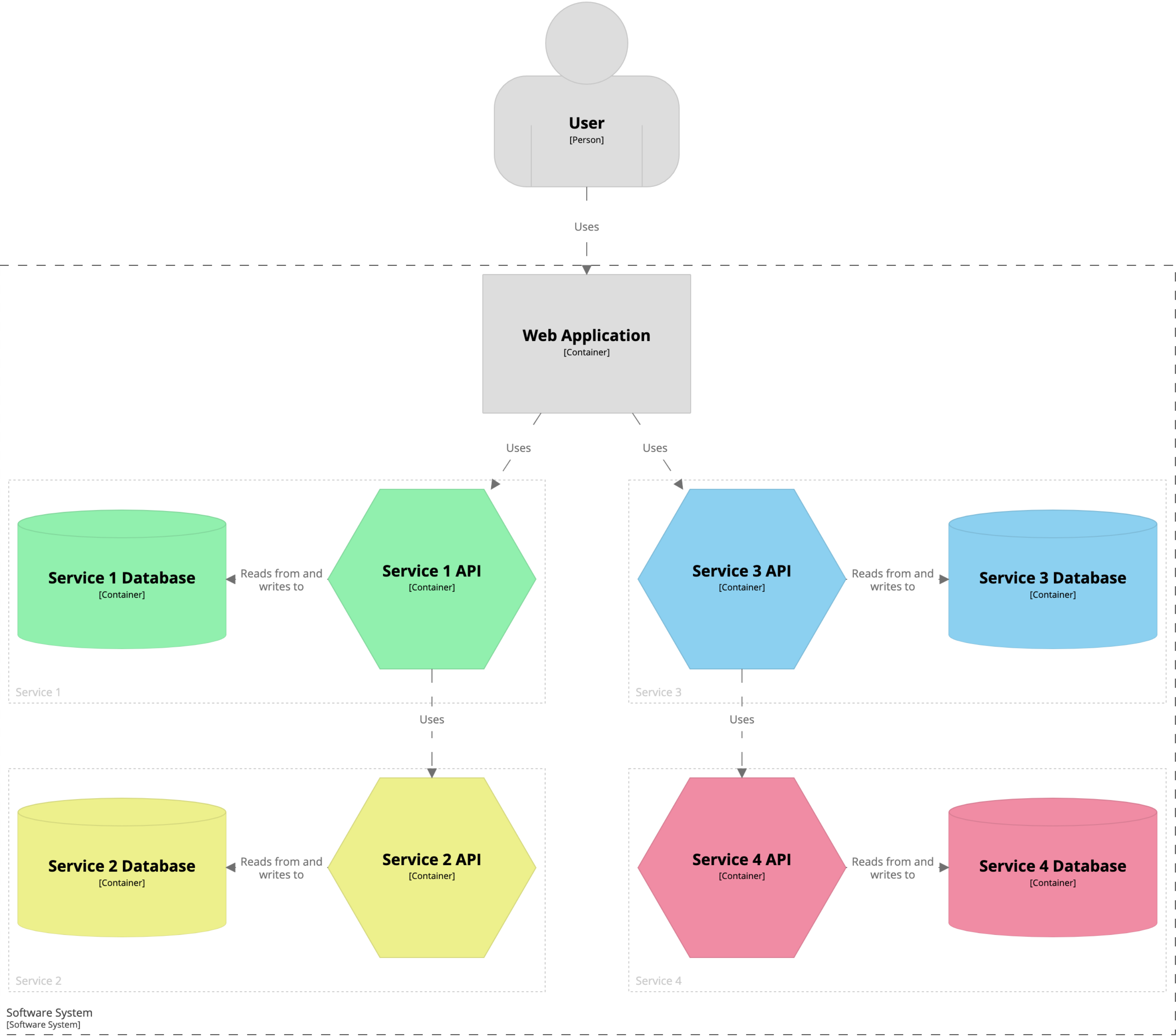


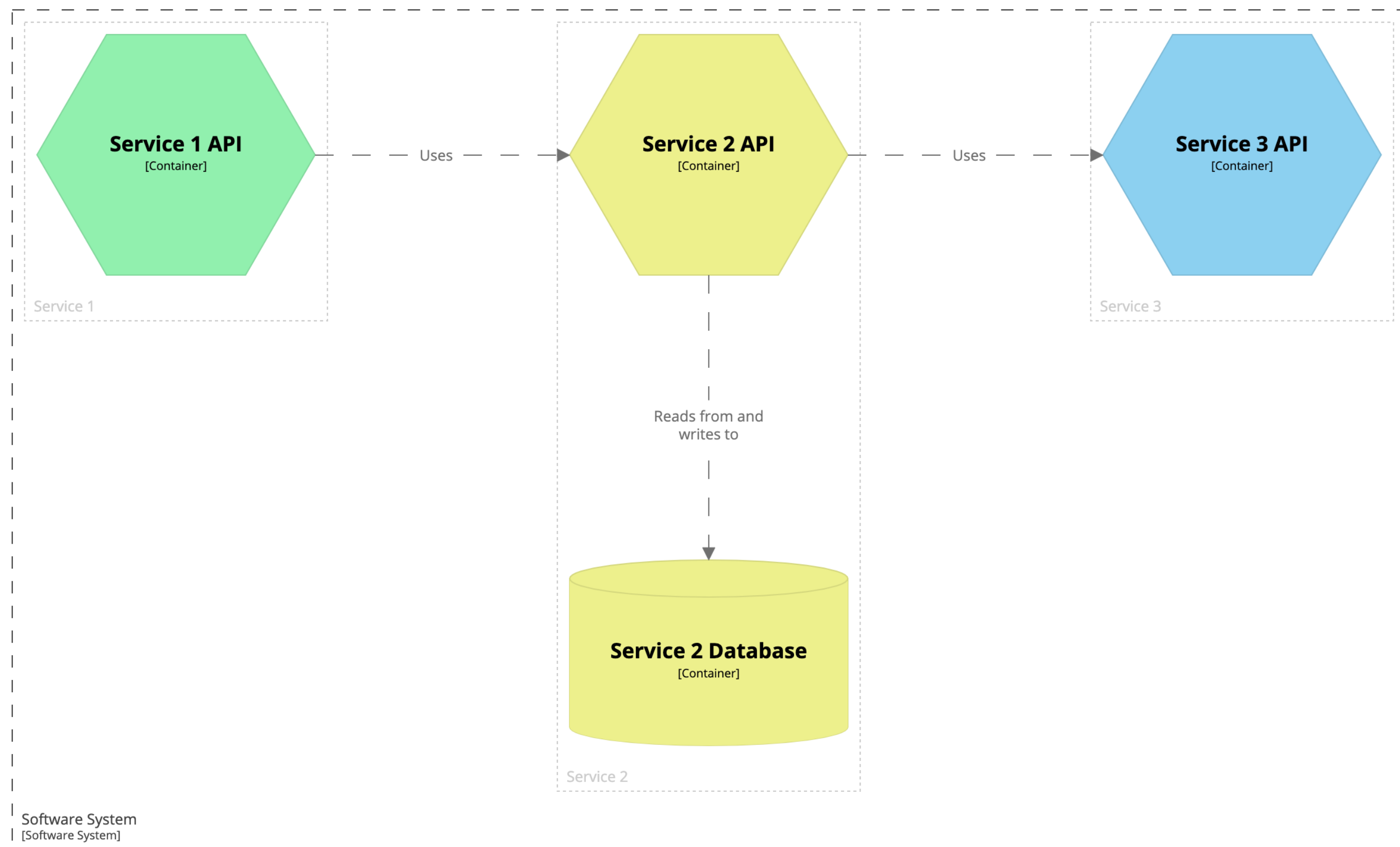
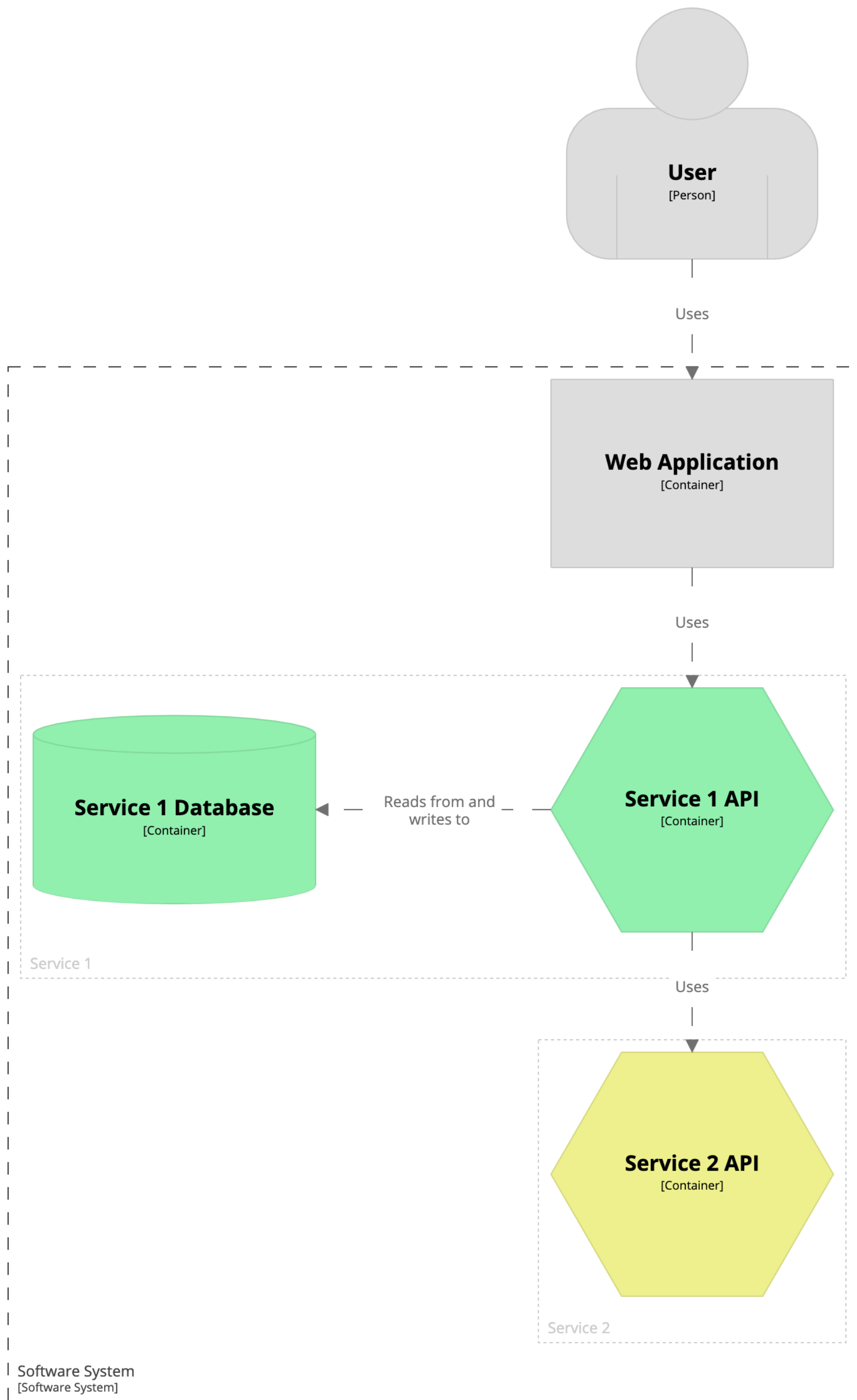
Software System - Containers



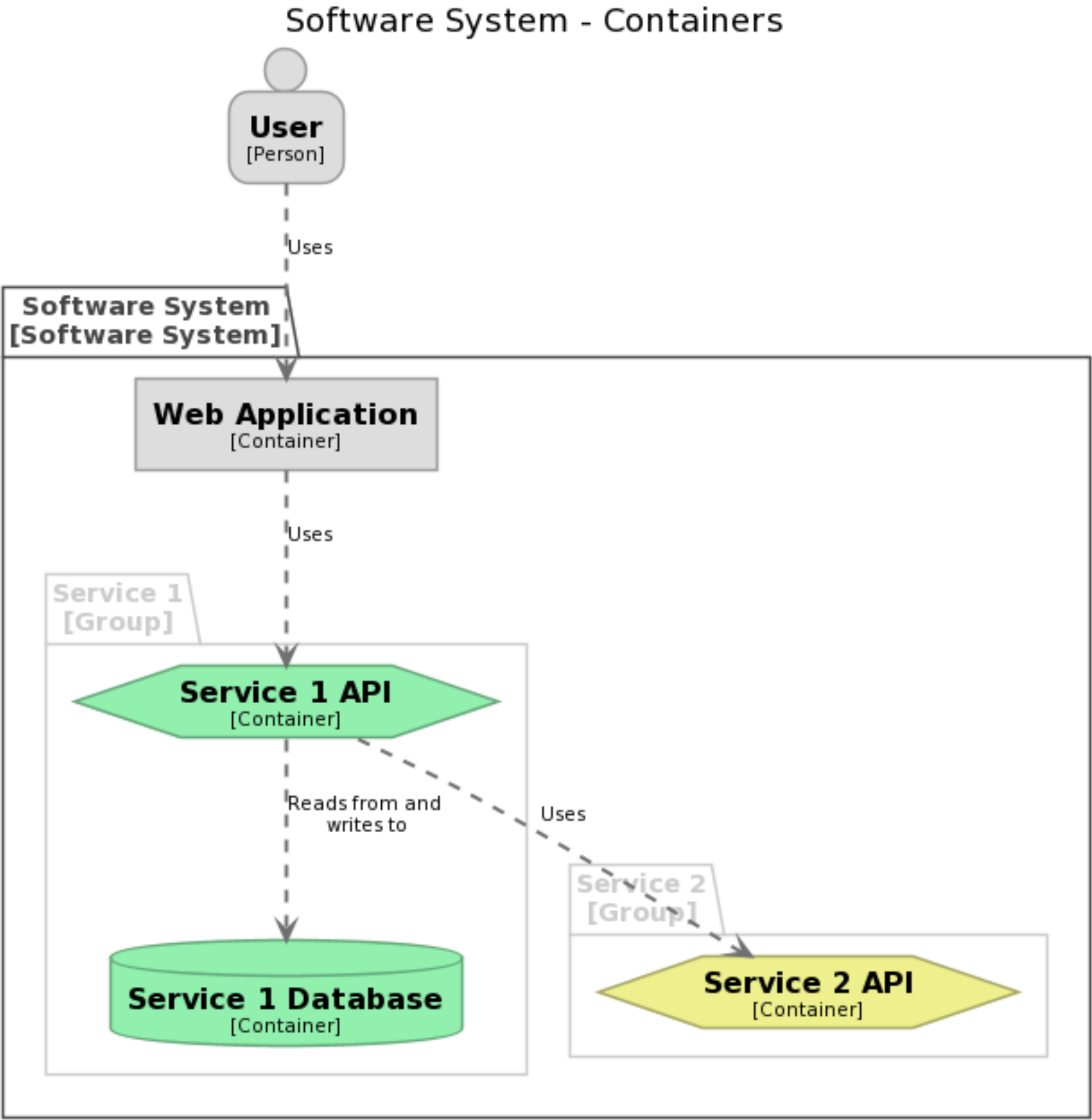
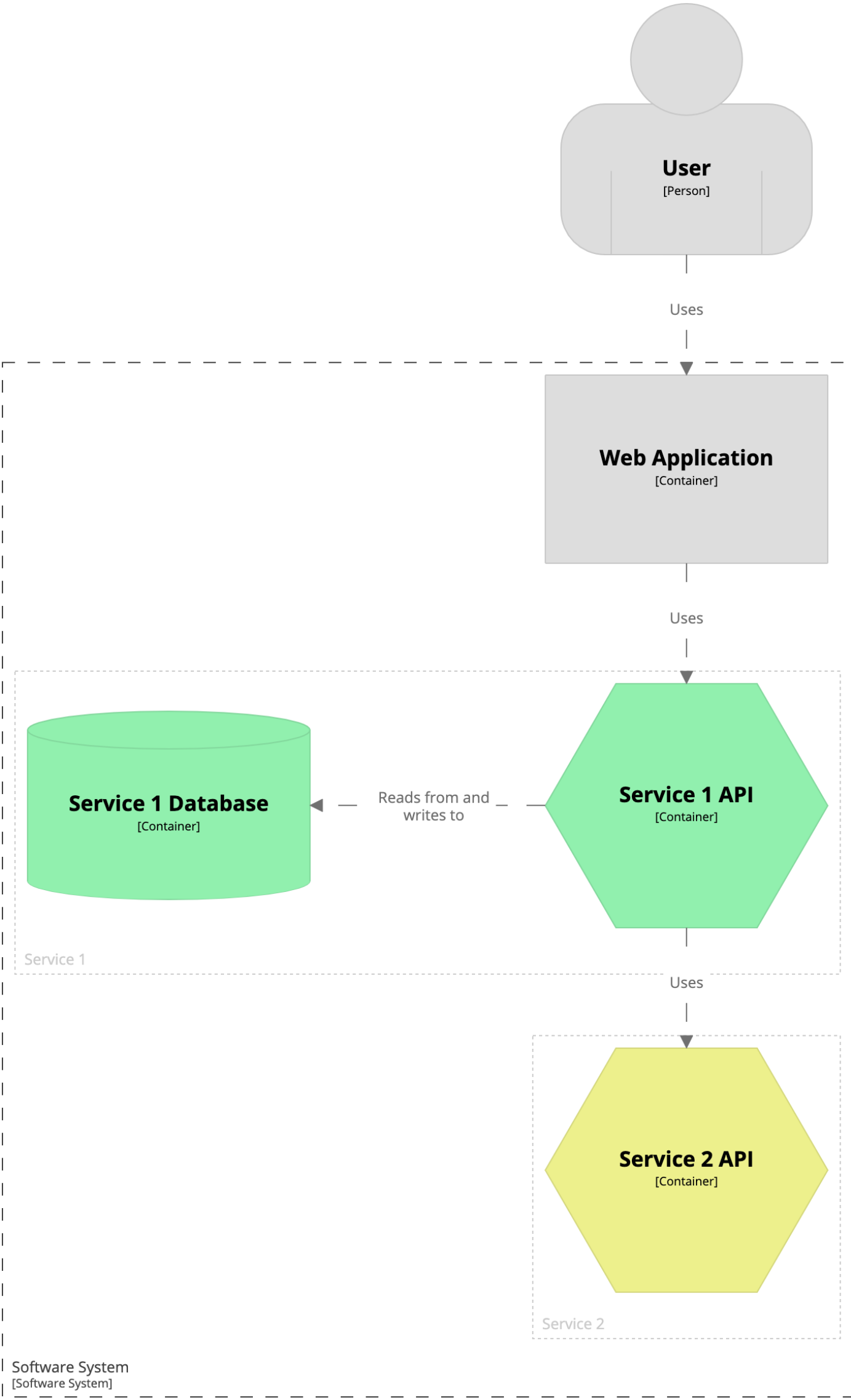
More advanced features

How do you diagram large and complex software systems?

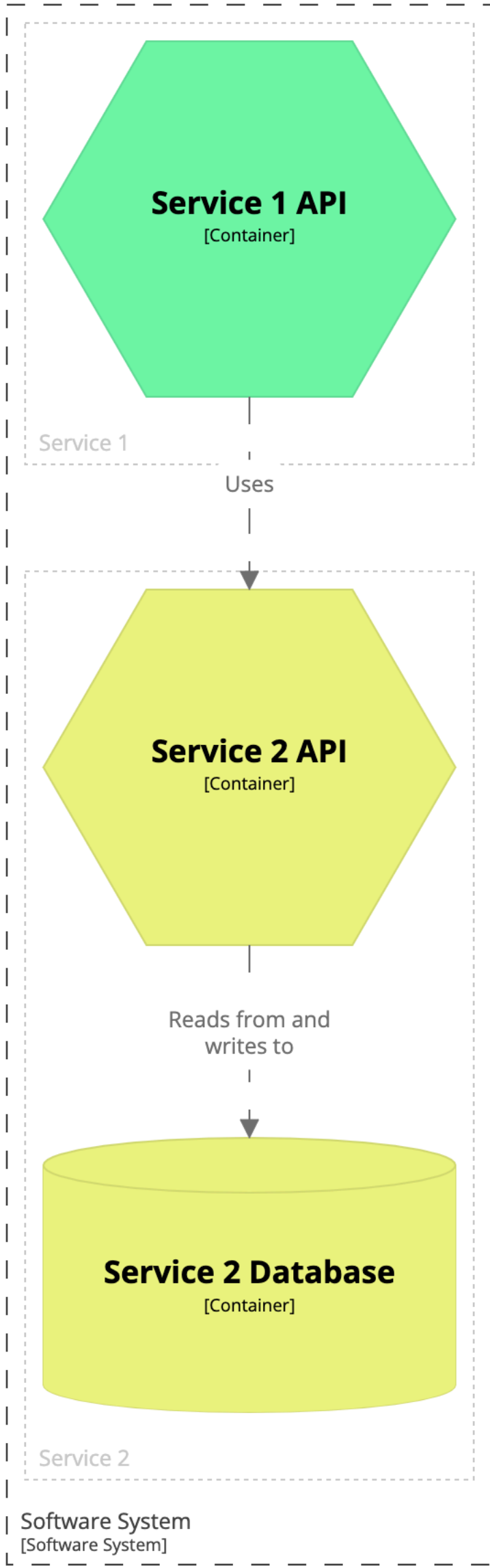




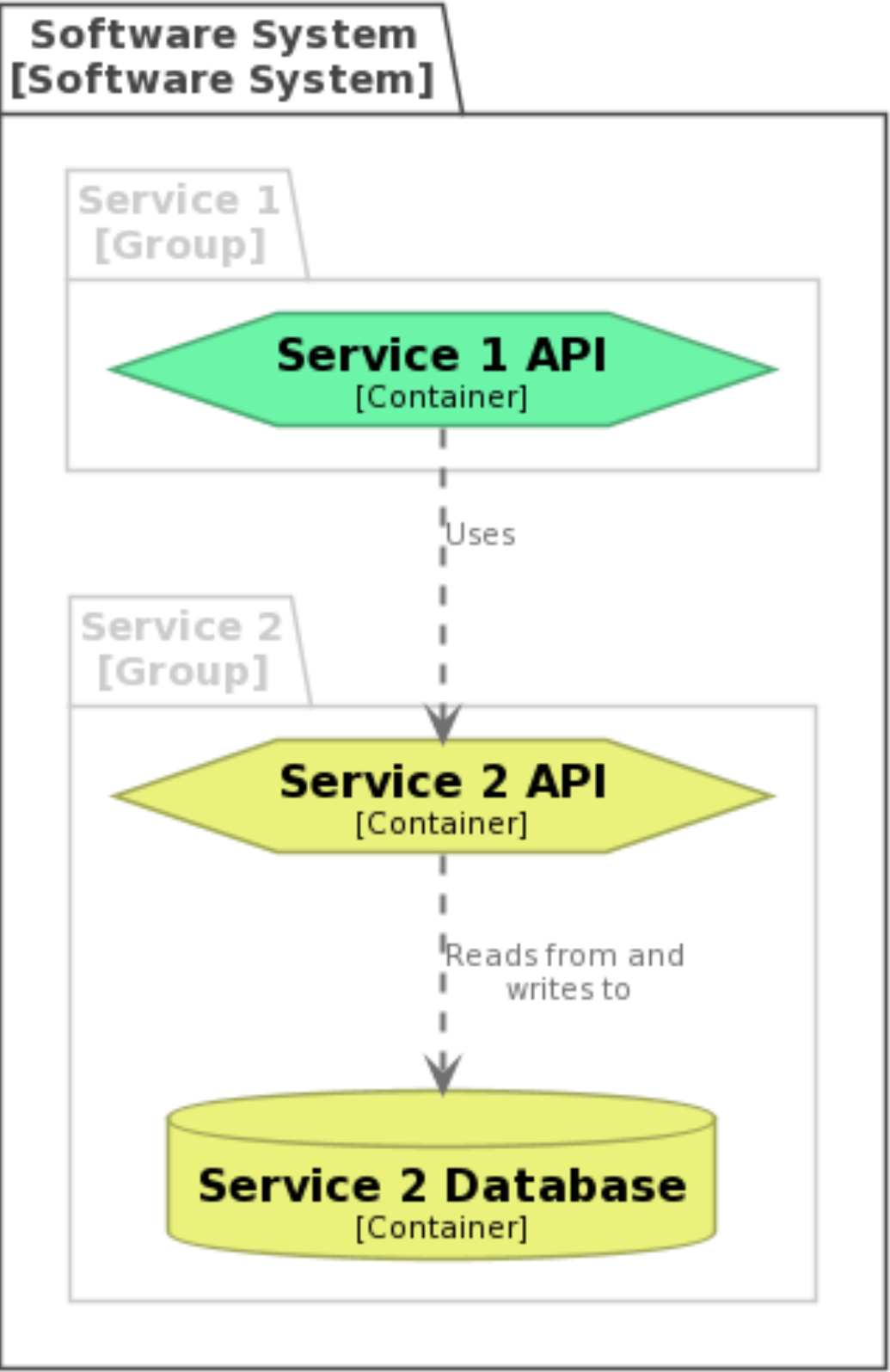
```
container softwareSystem {
  include user ->service1->
  autolayout
}
```

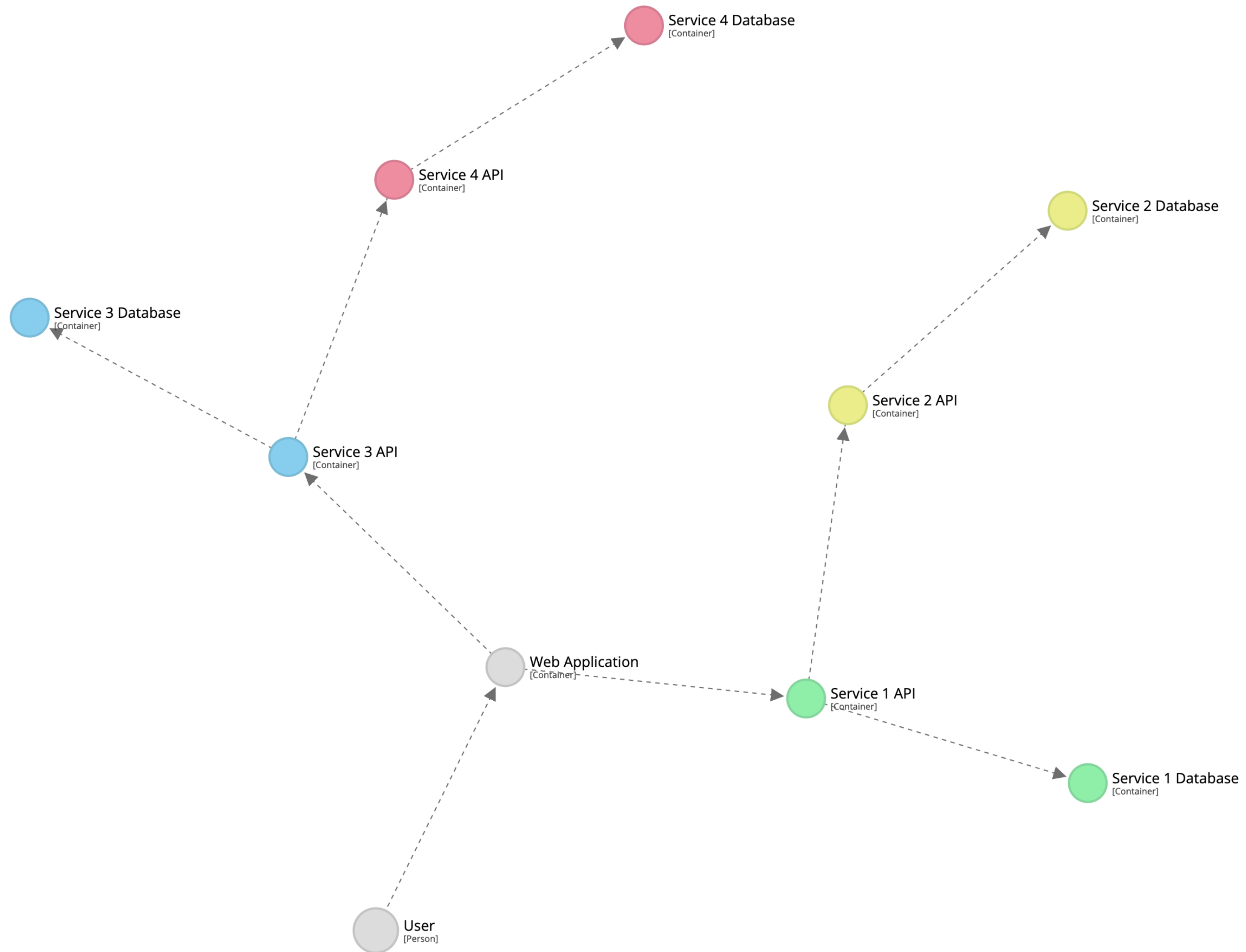


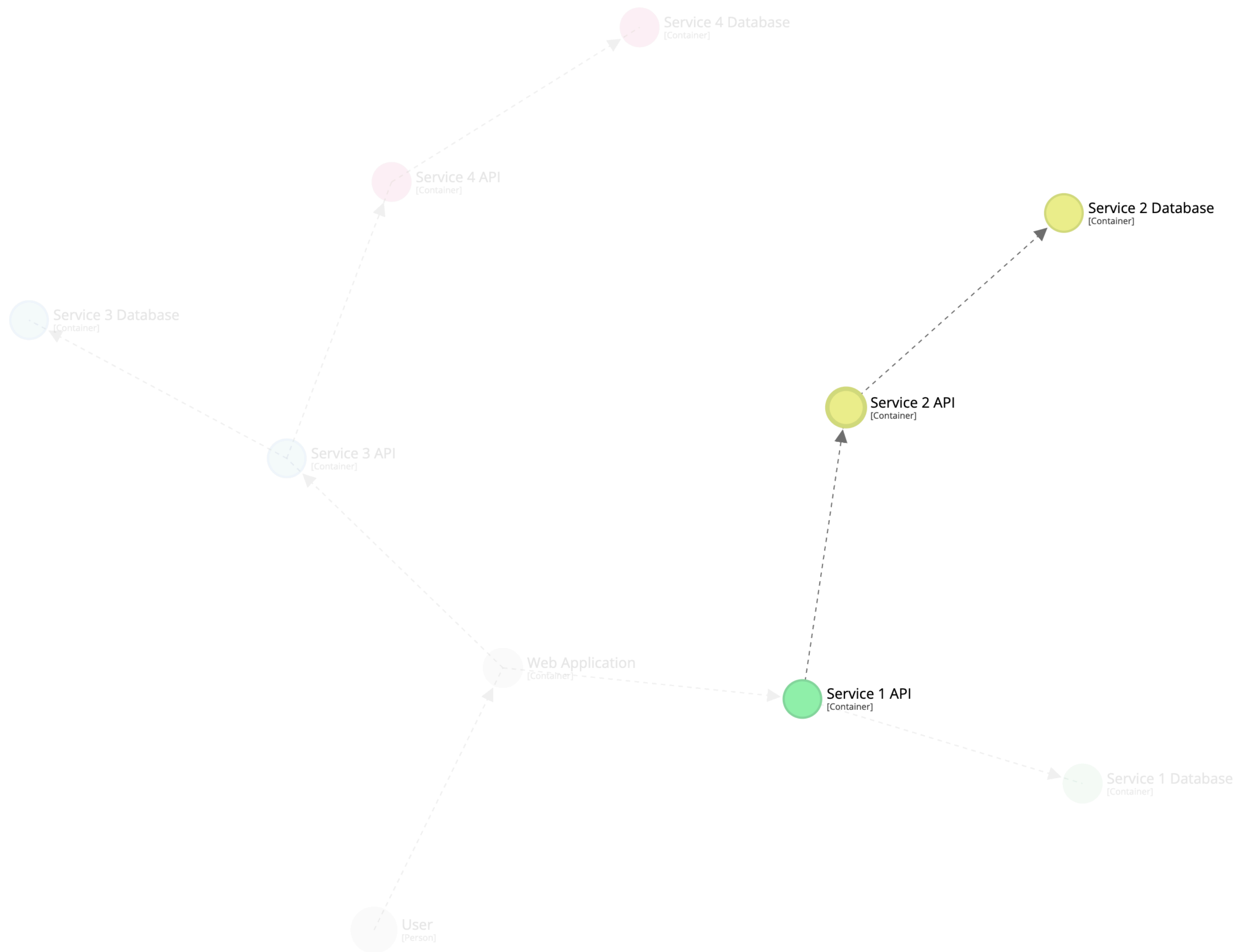
```
container softwareSystem {
  include ->service2->
  autolayout
}
```

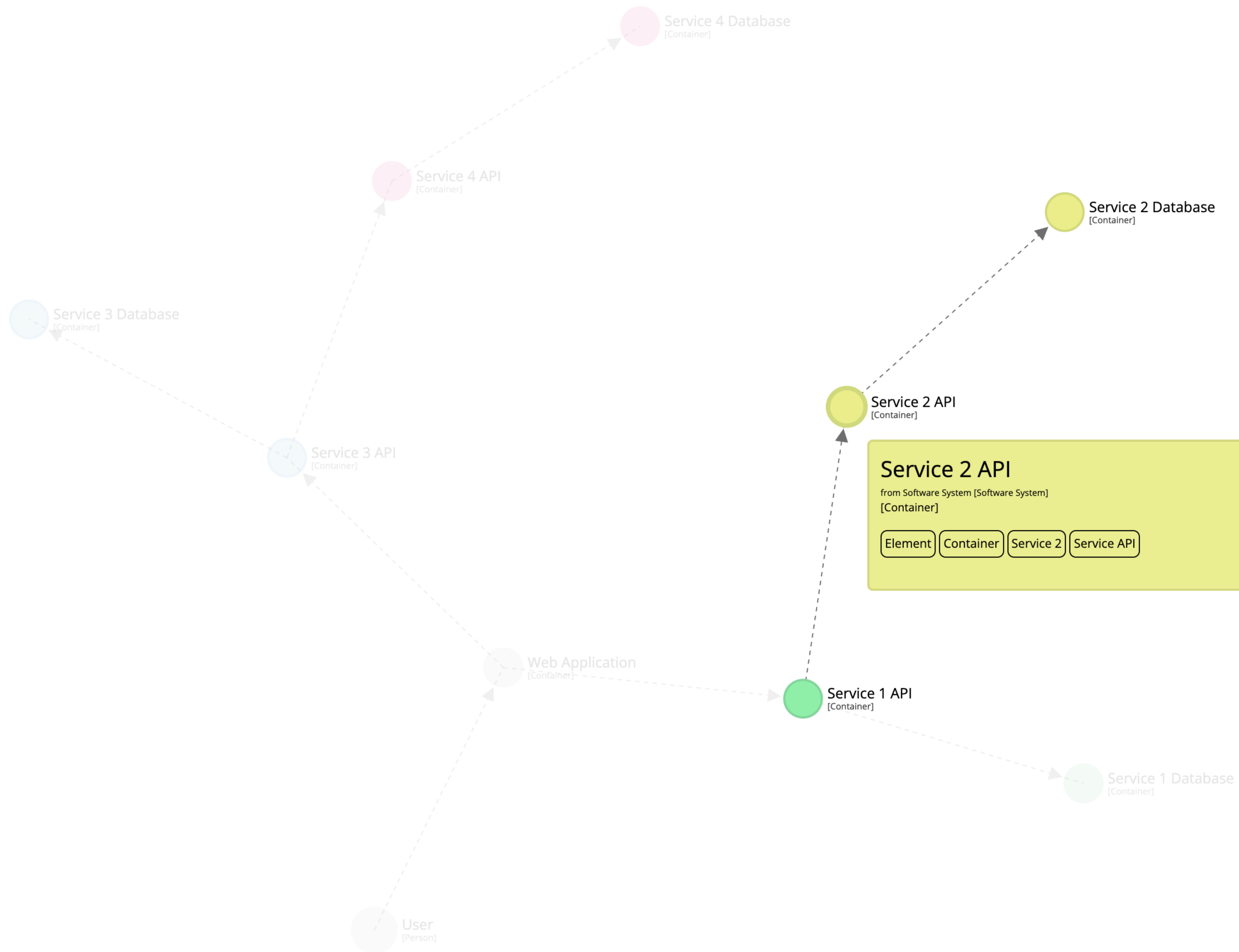


Software System - Containers









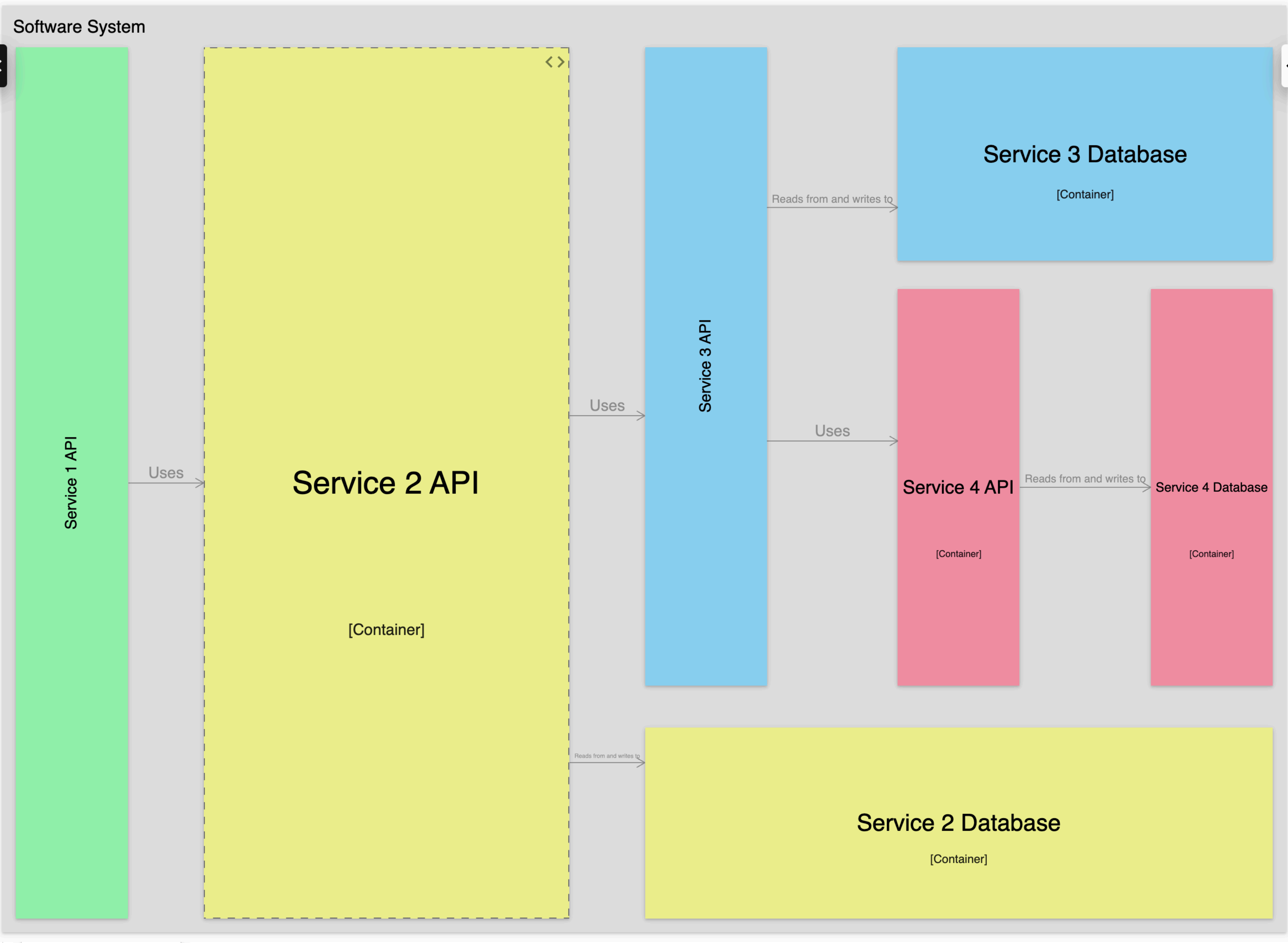
Save changes

```
1 resources:
2
3 - id: "1"
4   name: "User"
5   subtitle: "[Person]"
6   backgroundColor: "#dddddd"
7   color: "#000000"
8
9 - id: "2"
10  name: "Software System"
11  subtitle: "[Software System]"
12  backgroundColor: "#dddddd"
13  color: "#000000"
14
15  children:
16  - id: "10"
17    name: "Service 3 API"
18    subtitle: "[Container]"
19    backgroundColor: "#8cd0f0"
20    color: "#000000"
21
22  - id: "11"
23    name: "Service 3 Database"
24    subtitle: "[Container]"
25    backgroundColor: "#8cd0f0"
26    color: "#000000"
27
28  - id: "13"
29    name: "Service 4 API"
30    subtitle: "[Container]"
31    backgroundColor: "#f08ca4"
32    color: "#000000"
33
34  - id: "14"
35    name: "Service 4 Database"
36    subtitle: "[Container]"
37    backgroundColor: "#f08ca4"
38    color: "#000000"
39
40  - id: "3"
41    name: "Web Application"
42    subtitle: "[Container]"
43    backgroundColor: "#dddddd"
44    color: "#000000"
45
46  - id: "4"
47    name: "Service 1 API"
48    subtitle: "[Container]"
49    backgroundColor: "#91f0ae"
50    color: "#000000"
51
52  - id: "5"
53    name: "Service 1 Database"
54    subtitle: "[Container]"
55    backgroundColor: "#91f0ae"
56    color: "#000000"
57
58  - id: "7"
59    name: "Service 2 API"
```



🗒️ 📄 Save changes

```
35      name: "Service 2 Database"
36      subtitle: "[Container]"
37      backgroundColor: "#edf08c"
38      color: "#000000"
39
40  - id: "6"
41    name: "Service 2 API"
42    subtitle: "[Container]"
43    backgroundColor: "#edf08c"
44    color: "#000000"
45
46  - id: "8"
47    name: "Service 3 Database"
48    subtitle: "[Container]"
49    backgroundColor: "#8cd0f0"
50    color: "#000000"
51
52  - id: "9"
53    name: "Service 3 API"
54    subtitle: "[Container]"
55    backgroundColor: "#8cd0f0"
56    color: "#000000"
57
58  perspectives:
59  - name: Static Structure
60    relations:
61    - from: "12"
62      to: "11"
63      label: "Reads from and writes to"
64
65    - from: "3"
66      to: "6"
67      label: "Uses"
68
69    - from: "3"
70      to: "2"
71      label: "Reads from and writes to"
72
73    - from: "6"
74      to: "9"
75      label: "Uses"
76
77    - from: "6"
78      to: "5"
79      label: "Reads from and writes to"
80
81    - from: "9"
82      to: "8"
83      label: "Reads from and writes to"
84
85    - from: "9"
86      to: "12"
87      label: "Uses"
88
89
```



Enterprise-wide modelling?

Software systems and people

`system-landscape.dsl`

Software System 1

`software-system-1.dsl`
extends
`system-landscape.dsl`

Software System 2

`software-system-2.dsl`
extends
`system-landscape.dsl`

Software System 3

`software-system-3.dsl`
extends
`system-landscape.dsl`

Scripting support

(via JSR-223: Java Scripting API)

```
workspace {  
  
    model {  
        ...  
    }  
  
    !script groovy {  
        workspace.views.createDefaultViews()  
        workspace.views.views.each { it.disableAutomaticLayout() }  
    }  
  
}
```

Plugin support

(via Java)

Hybrid usage

(DSL and Java)


```
workspace {  
  
    model {  
        s = softwareSystem "Software System" {  
            webapp = container "Web Application"  
            database = container "Database" {  
                webapp -> this "Reads from and writes to"  
            }  
        }  
    }  
  
    views {  
        systemContext s {  
            include *  
            autoLayout lr  
        }  
  
        container s {  
            include *  
            autoLayout lr  
        }  
    }  
  
}
```

```
StructurizrDslParser parser = new StructurizrDslParser();
parser.parse(new File("workspace.dsl"));

Workspace workspace = parser.getWorkspace();
Container webApplication = workspace.getModel()
    .getSoftwareSystemWithName("Software System")
    .getContainerWithName("Web Application");

// add components manually or via automatic extraction
...

// add a component view
ComponentView componentView = workspace.getViews()
    .createComponentView(webApplication, "Components", "Description");
componentView.addDefaultElements();
componentView.enableAutomaticLayout();
```

Custom tooling

Authoring tool

Create diagrams as code (Java, .NET, TypeScript, Python, PHP, etc) or text (DSL, YAML) via a number of different authoring tools.

```
private static final long WORKSPACE_ID = 25441;
private static final String API_KEY = "";
private static final String API_SECRET = "";

public static void main(String[] args) throws Exception {
    // all software architecture models belong to a workspace
    Workspace workspace = new Workspace("Getting Started", "This is a model of my software system.");
    Model model = workspace.getModel();

    // create a model to describe a user using a software system
    Person user = model.addPerson("User", "A user of my software system.");
    SoftwareSystem softwareSystem = model.addSoftwareSystem("Software System", "My software system.");
    user.uses(softwareSystem, "Uses");

    // create a system context diagram showing people and software systems
    ViewSet views = workspace.getViews();
    SystemContextView contextView = views.createSystemContextView(softwareSystem, "SystemContext", "An example of contextView.addSoftwareSystems();
    contextView.addSoftwareSystems();
    contextView.addPeople();

    // add some styling to the diagram elements
    Styles styles = views.getConfiguration().getStyles();
    styles.addElementStyle(Tags.SOFTWARE_SYSTEM).background("#1168bd").color("ffffff");
    styles.addElementStyle(Tags.PERSON).background("#08427b").color("ffffff").shape(Shape.Person);

    // upload to structurizr.com (you'll need your own workspace ID, API key and API secret)
    StructurizrClient structurizrClient = new StructurizrClient(API_KEY, API_SECRET);
    structurizrClient.putWorkspace(WORKSPACE_ID, workspace);
}
```

```
workspace "Getting Started" "This is a model of my software system." {

    model {
        user = person "User" "A user of my software system."
        softwareSystem = softwareSystem "Software System" "My software system."

        user -> softwareSystem "Uses"
    }

    views {
        systemContext softwareSystem "SystemContext" "An example of a System Context diagram." {
            include *
        }

        styles {
            element "Software System" {
                background #1168bd
                color #ffffff
            }
            element "Person" {
                shape person
                background #08427b
                color #ffffff
            }
        }
    }
}
```

Workspace

A workspace is the wrapper for a software architecture model and views, described using the C4 model and an open JSON data format.

Creates

Renders

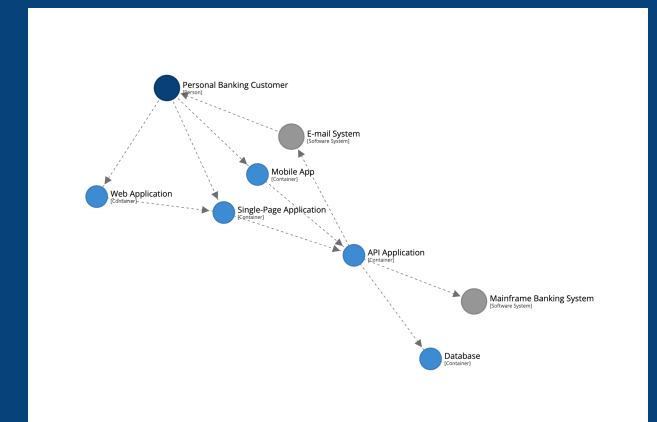
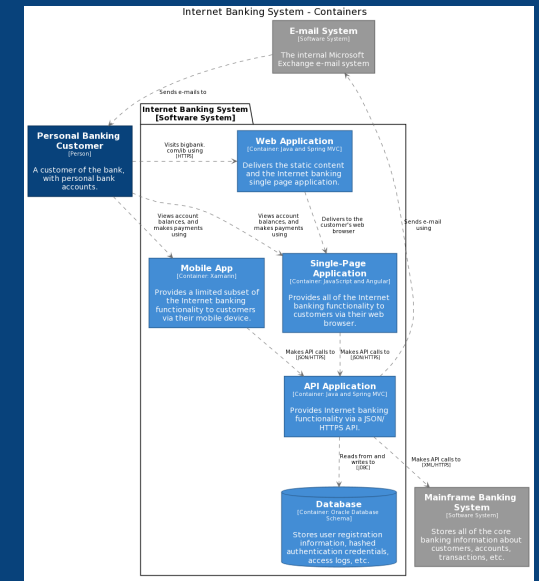
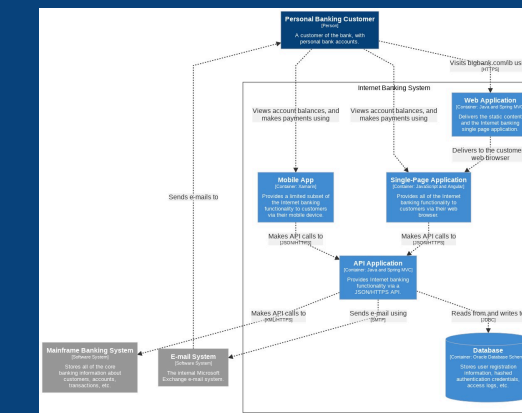
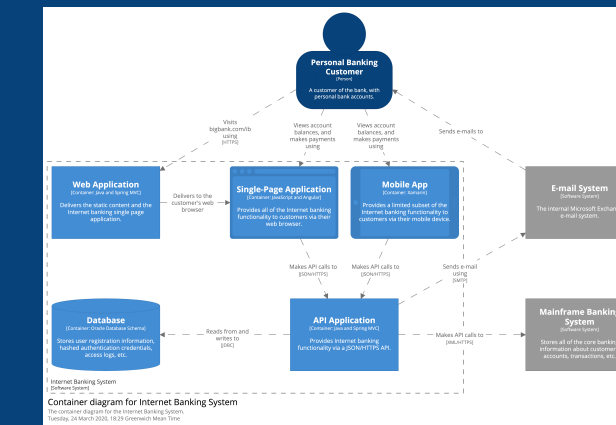
Consumes

Custom tool

Your own tooling to parse the model and views; for integration into other rendering tools, dashboards, service catalogs, etc.

Rendering tool

Render views using multiple diagramming tools and formats (Structurizr cloud service/on-premises installation, PlantUML, Mermaid, WebSequenceDiagrams, Ilograph, etc).



Usage scenarios

Static diagrams

(e.g. PNG/SVG)

Interactive diagrams

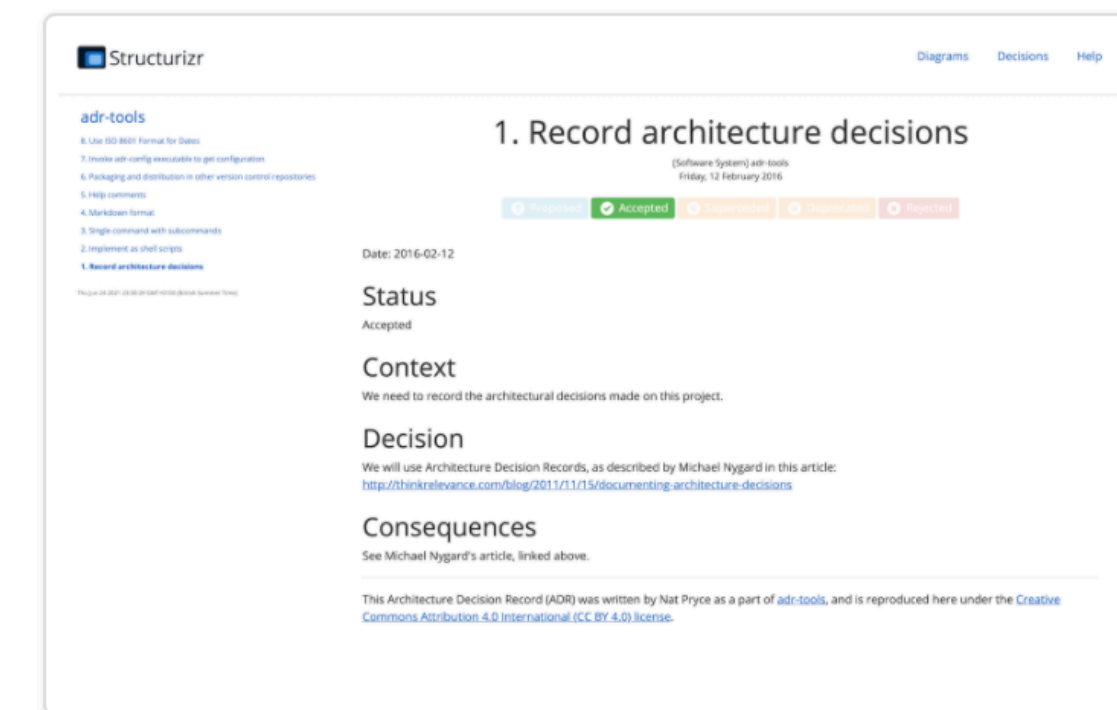
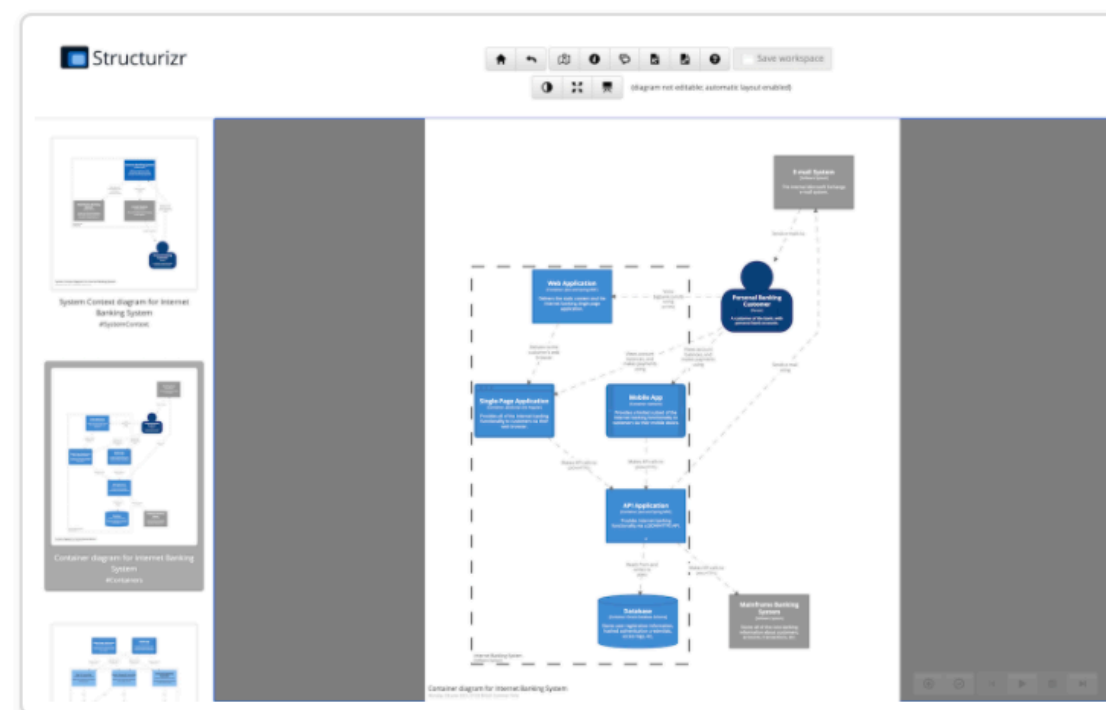
(e.g. browser-based)

Structurizr Lite

[Overview](#) | [Getting started](#) | [Auto-sync](#) | [Workflow](#) | [Docker Hub](#)

Overview

Packaged as a Docker container, and designed for developers, this version of Structurizr provides a way to quickly work with a single workspace. It's free to use, and allows you to view/edit diagrams, view documentation, and view architecture decision records defined in a DSL or JSON workspace.



Structurizr Lite will look for a `workspace.dsl` and `workspace.json` file in a given directory, in that order, and use the file it finds first. If you change this file (e.g. via your text editor or one of the Structurizr client libraries), you can refresh your web browser to immediately see the changes.

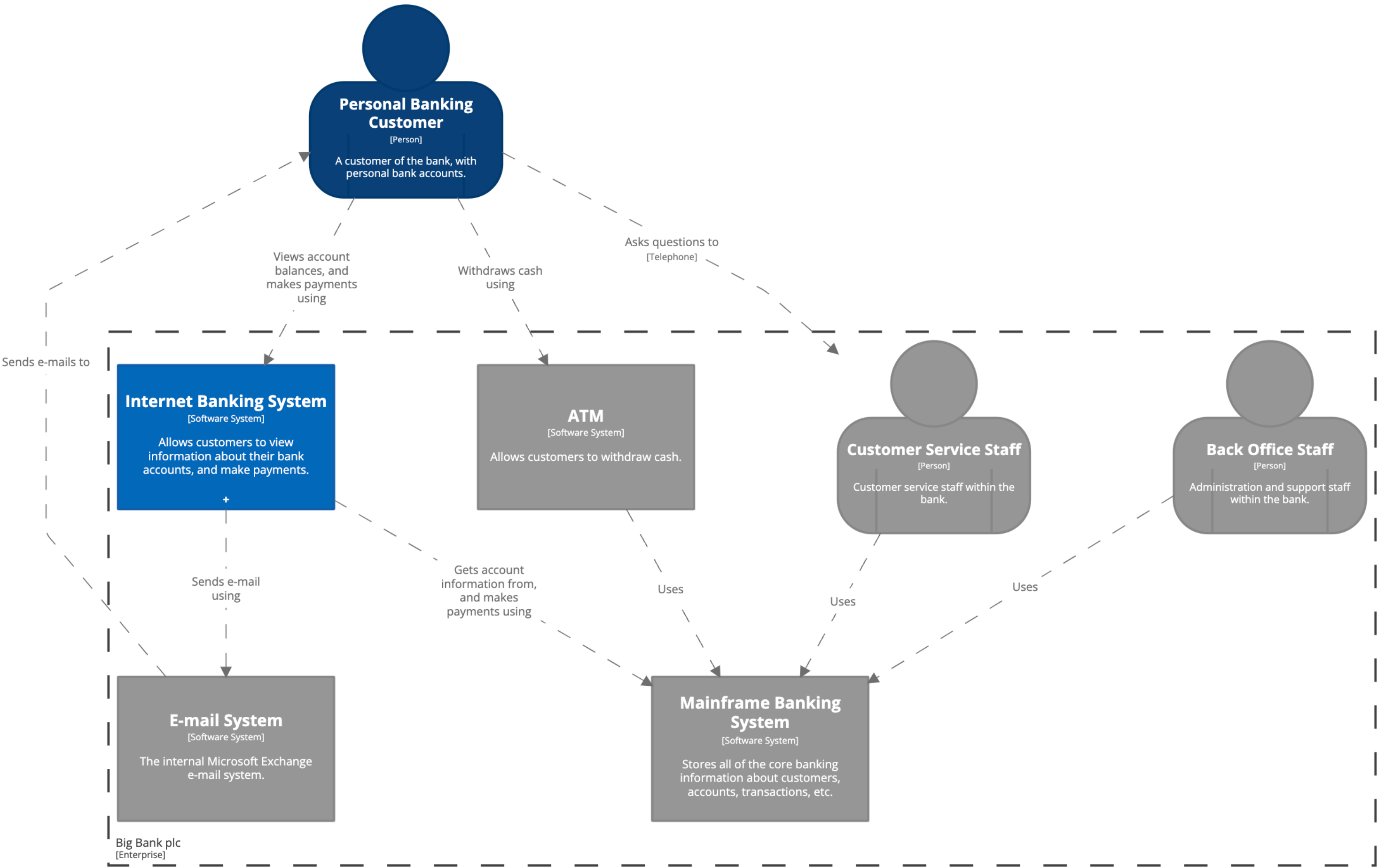
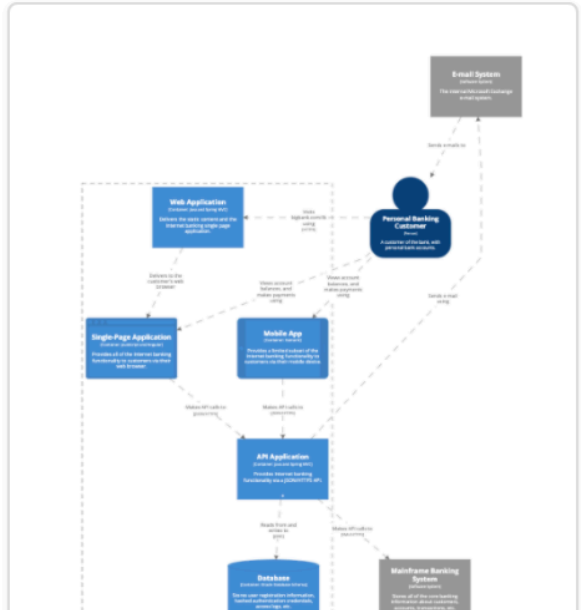
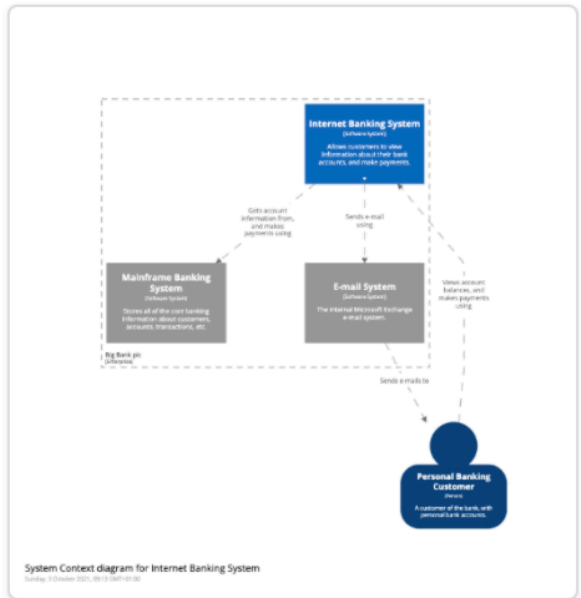
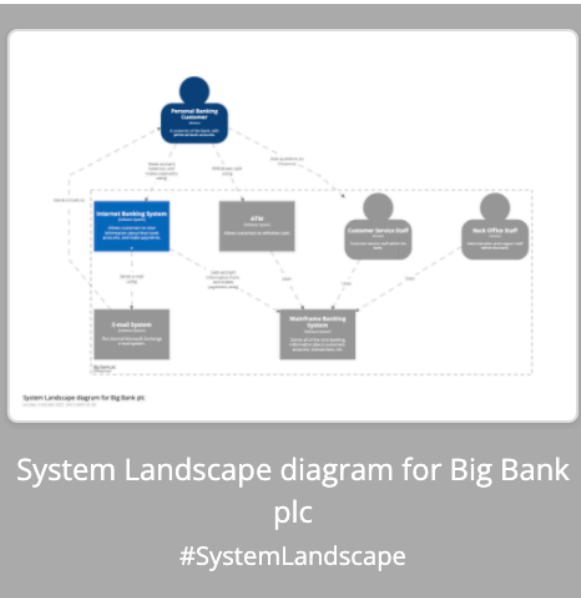
<https://structurizr.com/help/lite>


```
docker run -it --rm -p 8080:8080 -v /Users/simon/bigbankplc/:/usr/local/structurizr structurizr/lite
```



Save workspace

(diagram not editable; automatic layout enabled)



System Landscape diagram for Big Bank plc
Monday, 11 October 2021, 08:40 GMT+01:00



Financial Risk System

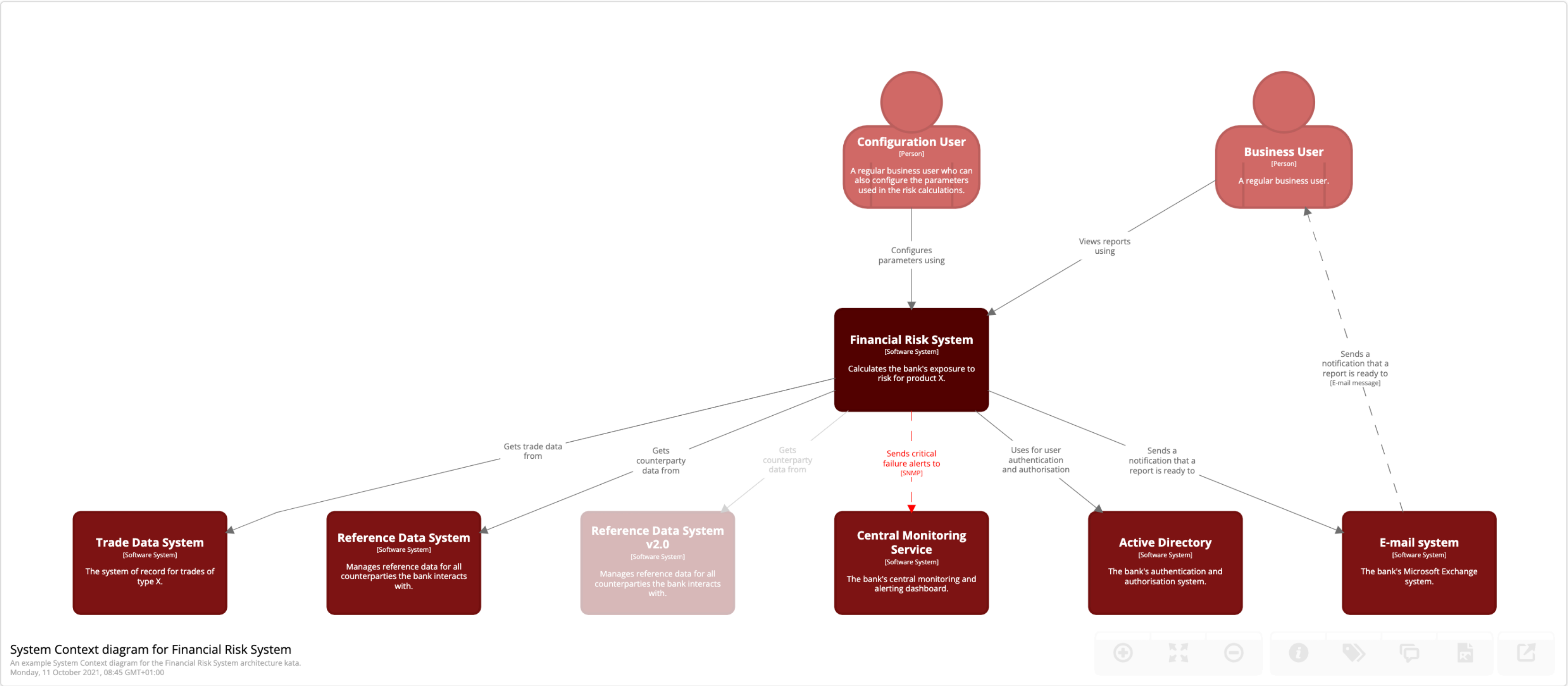
- 1 Context
 - 1.1 Trade Data System
 - 1.2 Reference Data System
- 2 Functional Overview
- 3 Quality Attributes
 - 3.1 Performance
 - 3.2 Scalability
 - 3.3 Availability
 - 3.4 Failover
 - 3.5 Security
 - 3.6 Audit
 - 3.7 Fault Tolerance and Resilience
 - 3.8 Internationalization and Localization
 - 3.9 Monitoring and Management
 - 3.10 Data Retention and Archiving
 - 3.11 Interoperability

Monday, 11 October 2021, 08:45 GMT+01:00

Financial Risk System

1 Context

A global investment bank based in London, New York and Singapore trades (buys and sells) financial products with other banks (counterparties). When share prices on the stock markets move up or down, the bank either makes money or loses it. At the end of the working day, the bank needs to gain a view of how much risk they are exposed to (e.g. of losing money) by running some calculations on the data held about their trades. The bank has an existing Trade Data System (TDS) and Reference Data System (RDS) but need a new Risk System.



1.1 Trade Data System

!adrs <directory name>

adr-tools

- 8. Use ISO 8601 Format for Dates
- 7. Invoke adr-config executable to get configuration
- 6. Packaging and distribution in other version control repositories
- 5. Help comments
- 4. Markdown format
- 3. Single command with subcommands
- 2. Implement as shell scripts
- 1. Record architecture decisions

Sun Oct 03 2021 10:08:10 GMT+0100 (GMT+01:00)

adr-tools - Summary

[Software System] adr-tools



2017

8. Use ISO 8601 Format for Dates

Tuesday, 21 February 2017

Accepted

2016

7. Invoke adr-config executable to get configuration

Saturday, 17 December 2016

Accepted

6. Packaging and distribution in other version control repositories

Tuesday, 16 February 2016

Accepted

5. Help comments

Saturday, 13 February 2016

Accepted

4. Markdown format

Friday, 12 February 2016

Accepted

3. Single command with subcommands

Friday, 12 February 2016

Accepted

2. Implement as shell scripts

Friday, 12 February 2016

Accepted

1. Record architecture decisions

Friday, 12 February 2016

Accepted

Closing thoughts

“Diagrams as code” is easy to author,
diff, version control, collaborate on,
integrate into CI/CD, etc

“Diagrams as code 2.0”
makes this model based,
separating content from presentation

Developers
vs
non-developers?

Store your diagrams and docs
in version control,
next to your source code

“Publish” the diagrams and
documentation if necessary

Up front design
VS
long-lived documentation?

Think about diagrams as being
“disposable” artefacts

DSL language reference

Render

Structurizr Diagram

Structurizr Graph

Export PlantUML

Export C4-PlantUML

Export Mermaid

Export DOT

Export WebSequenceDiagrams

Export Ilograph

Structurizr JSON

1 workspace {
2
3 model {
4 user = person "User"
5 softwareSystem = softwareSystem "Software System"
6
7 user -> softwareSystem "Uses"
8 }
9
10 views {
11 systemContext softwareSystem {
12 include *
13 autolayout
14 }
15
16 theme default
17 }
18
19 }

[System Context] Software System (#SoftwareSystem-SystemContext)

(diagram not editable; automatic layout enabled)

```
graph TD; User((User  
[Person])) -- Uses --> SoftwareSystem[Software System  
[Software System]]
```


Structurizr DSL cookbook

Creating software architecture diagrams from a textual definition is becoming more popular, but it's easy to introduce inconsistencies into your diagrams if you don't keep the multiple source files in sync. This cookbook is a guide to the Structurizr DSL, an open source tool for creating diagrams as code from a single consistent model.

Table of contents

- [Introduction](#)
- [Workspace](#)
- [Workspace extension](#)
- [System Context view](#)
- [Container view](#)
- [Component view](#)
- [Filtered view](#)
- [Dynamic view](#)
- [Deployment view](#)
- [Amazon Web Services](#)
- [Element styles](#)
- [Relationship styles](#)
- [Themes](#)
- [Implied relationships](#)
- [Scripts](#)
- [DSL and code \(hybrid usage pattern\)](#)

Thank you!



Simon Brown

 @simonbrown

Don't forget to
vote for this session
in the **GOTO Guide app**