

Documentation for Developers and Friends

Shaw Innes





Proudly sponsored by

Diamond
Sponsors



Platinum Sponsors



Gold Sponsors



Childcare by:



Wi-Fi by:



Coffee Cart by:



Silver Sponsors





Play sponsor bingo and you can win



Developers
Do
Documentation

MELBOURNE

what is this talk about?

Overview

what is this thing?

Get Started

okay, cool, how do I get started?

Old Decisions

that's interesting, why does it...?

New Decisions

how should I add a new thing...?

Architecture

how does this all fit together now?

Sharing

how can my friends see this?

Scaling

document all the things!!

SixPivot

About Me

Currently

Practice Director
Principal Consultant
CTO

Previously

Various Enterprise
Various Startups
Lots of freelance things

And for fun...

Adventurer, dancer, photographer

shawinnes.com



why ^{should we} ~~do we~~ document things?

- Communicate complex systems easier
- Reduce cognitive load
- Streamline new team on-boarding
- Communicate previous decisions
- Architectural Guidelines and Principles
- Disaster recovery
- Future you will forget what the heck you did this week!

- and because...

developers who don't document

...are being jerks!

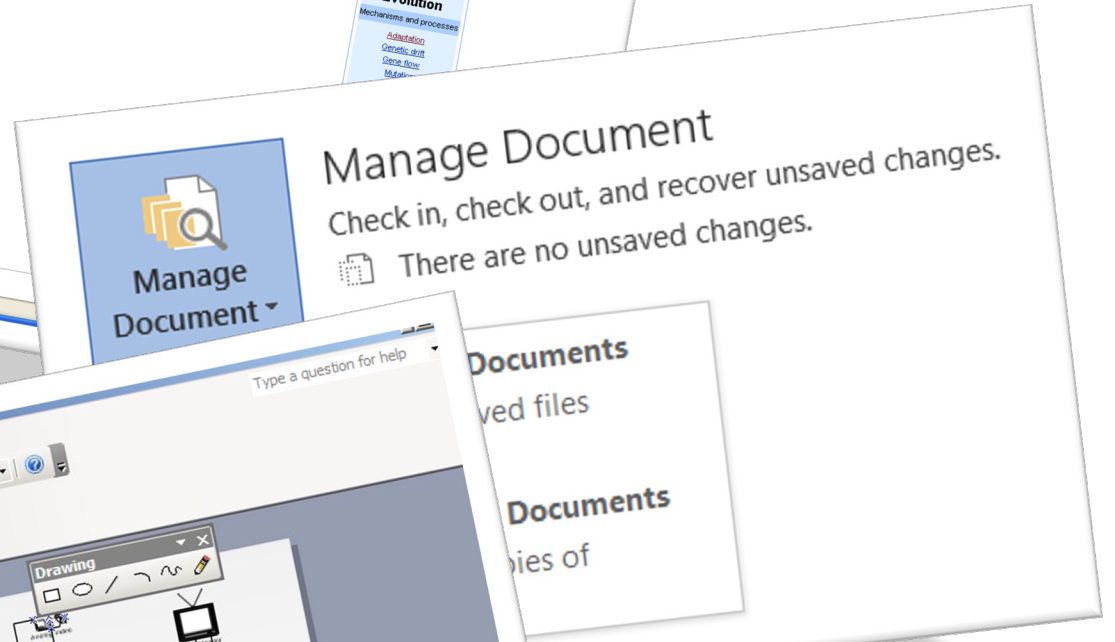
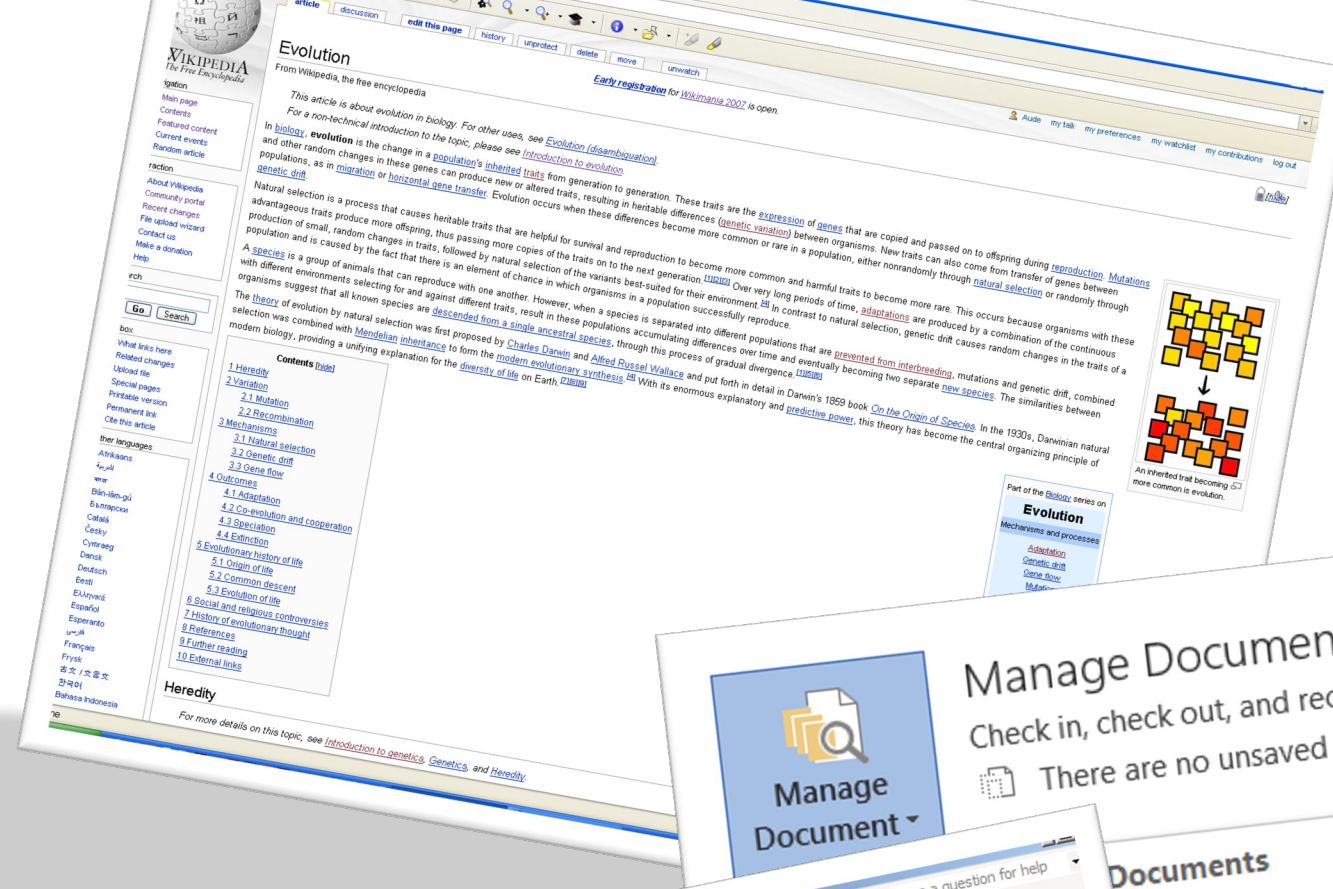
why aren't we doing this?

- It's boring
- It gets out of date and must be maintained
- It breaks the flow of software development
- Often requires special tooling, or licenses
- Diagramming tools are often hard to version control
- Perfect is the enemy of done

Traditional Approaches

- Word Documents
- Wikis, Confluence
- Other specialised tools

- Visio
- Draw.io
- Paint
- Illustration Tools



a better solution?

- Adopt “Documentation as Code”
- Separate structure from presentation
- Minimise requirement for extra tooling
- Version controlled in git
- Be pragmatic, start simple, scale up

PROS

- VERSION CONTROLLABLE / DIFFABLE
- SIMPLE TO USE / LEARN
- USING A DSL VS JUST DIAGRAMS MEANS THERE IS CONTEXT
- CROSS PLATFORM
- MINIMAL TOOLING
- CAN GENERATE VARIOUS OUTPUTS
- NO LICENSING

CONS

- THE LEARNING CURVE
- LESS ACCESSIBLE TO NON-TECH FOLK
- HARDER TO SEARCH & AGGREGATE (SOLVABLE)
- CODE SPRAWL (SOLVABLE)

a consulting story

- A new project
- A handful of repositories
- No documentation
- Where do we start?

what is this thing?

Table of Contents

- [Features](#features)
- [Installation](#installation)
- [Usage](#usage)
- [Screenshots](#screenshots)
- [Contributing](#contributing)
- [License](#license)

Features

- List key features of your React Native app.
- Highlight what makes your project unique.
- Use bullet points for easy readability.

Installation

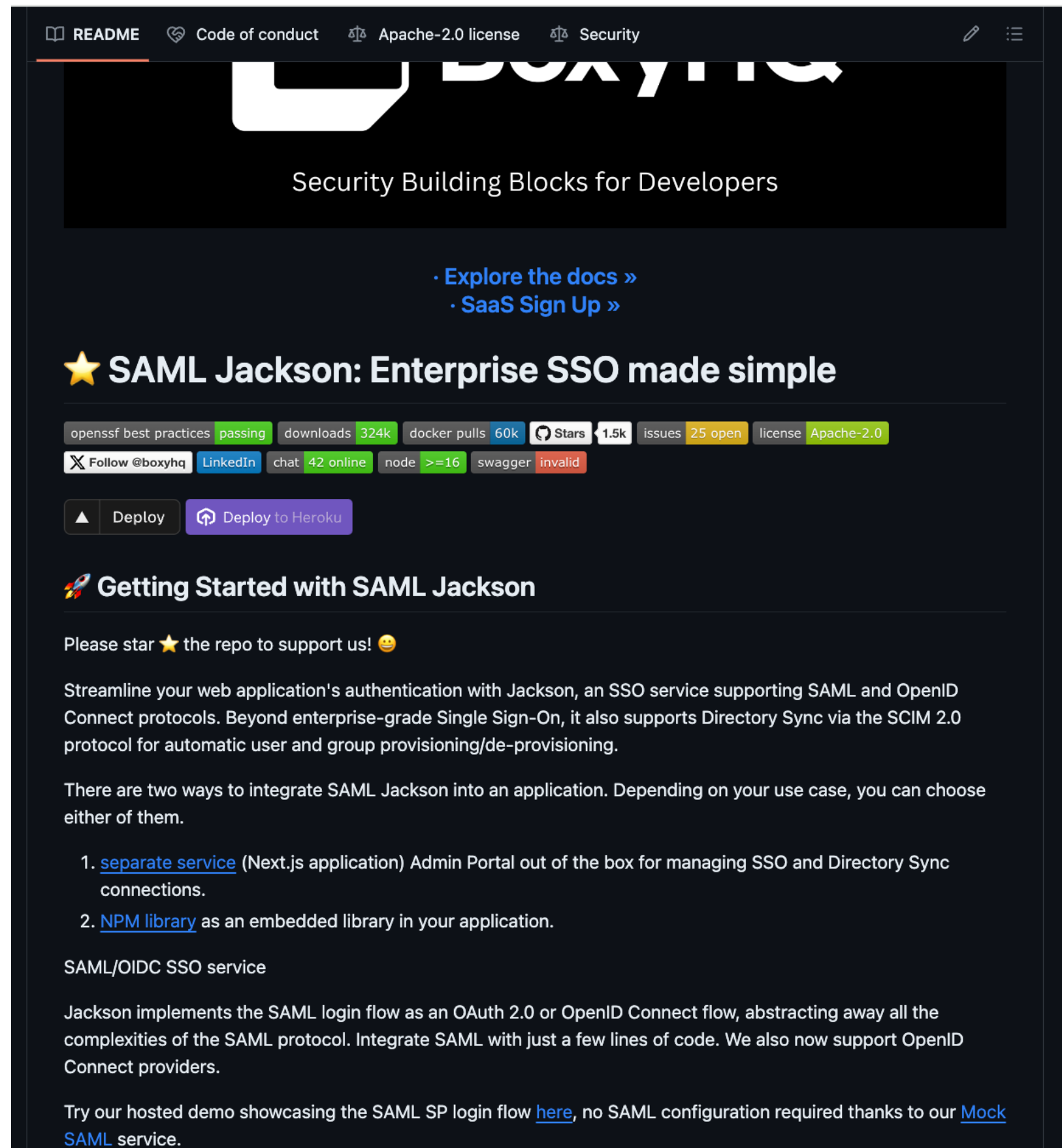
1. Clone the repository:

```
bash
git clone https://github.com/yourusername/your-repo.git
```

README.md

- A quick intro for onboarding
- Use a template for consistency
- Customise for your company
- Use [readme-md-generator](#)
- Google for examples
- Opensource projects

okay, cool, how do i get started?



The screenshot shows the GitHub repository page for SAML Jackson. At the top, there are navigation links for README, Code of conduct, Apache-2.0 license, and Security. Below this is a large header image with the text "Security Building Blocks for Developers". Underneath the header, there are two links: "Explore the docs >>" and "SaaS Sign Up >>". The main heading is "★ SAML Jackson: Enterprise SSO made simple". Below the heading, there are several statistics: "openssf best practices passing", "downloads 324k", "docker pulls 60k", "Stars 1.5k", "issues 25 open", and "license Apache-2.0". There are also social media links for "Follow @boxyhq", "LinkedIn", "chat 42 online", "node >=16", and "swagger invalid". Below these are "Deploy" and "Deploy to Heroku" buttons. The main content area is titled "Getting Started with SAML Jackson" and contains the following text:

Please star ★ the repo to support us! 😊

Streamline your web application's authentication with Jackson, an SSO service supporting SAML and OpenID Connect protocols. Beyond enterprise-grade Single Sign-On, it also supports Directory Sync via the SCIM 2.0 protocol for automatic user and group provisioning/de-provisioning.

There are two ways to integrate SAML Jackson into an application. Depending on your use case, you can choose either of them.

1. [separate service](#) (Next.js application) Admin Portal out of the box for managing SSO and Directory Sync connections.
2. [NPM library](#) as an embedded library in your application.

SAML/OIDC SSO service

Jackson implements the SAML login flow as an OAuth 2.0 or OpenID Connect flow, abstracting away all the complexities of the SAML protocol. Integrate SAML with just a few lines of code. We also now support OpenID Connect providers.

Try our hosted demo showcasing the SAML SP login flow [here](#), no SAML configuration required thanks to our [Mock SAML service](#).

README.md

- Requirements / dependencies
- Step by step instructions for devs
- Development processes
- Where to get more help

that's interesting, why does it...?

Capturing Old Decisions

- Markdown Decision Records (MADR)
 - Context/Problem
 - Considered Options
 - Decision
 - Consequences
- Use a template
- Create an index for easy reference

Mobile App Development Approach

Status

- Draft

Context

- We have been involved in an increasing number of projects where mobile app development has been a consideration.
- To date, we have made ad-hoc decisions on how to approach this, however for the most part the existing projects have been internal line-of-business apps and this has allowed the use of technologies such as [Xamarin](#).
- Ideally we want to have a standard practice for developing mobile apps which can be published to the app store for general consumption.
- The choice of technology should be an industry accepted technology, cross platform, and leverage as much of our existing skillset and experience as possible.
- This review has been made in the context of SixPivot, and comparisons have been made with consideration for sustainability and scalability within our consulting practice with our current team skills and experience.

Decision

- **React Native** is currently the best option for cross-platform mobile app development within SixPivot and we would recommend it for client use where there is no other obvious default.

Rationale

- **Cross-Platform Development**
 - React Native allows us to develop for Android and iOS platforms using a single codebase. This reduces development time and costs.
- **Existing Skillset**
 - Our team already has experience with React and TypeScript, which makes the transition to React Native smoother. The React paradigms transfer through to React Native so the

- 0000-index.md
- 0001-use-markdown-architectural-decision-records.md
- 0002-individual-device-topology.md
- 0003-single-page-applications.md
- 0004-rsa-as-library.md
- 0005-rsa-gui copy.md
- 0008-cmake.md
- 0009-device-authentication.md
- 0010-sms-notification.md
- 0011-email-notification.md
- 0012-rss-deployment.md
- 0013-yaml-pipelines.md
- 0014-message-types.md
- architecture-decision-record.md

how should I add a new thing...?

Architecture Principles

- Capture expectations/assumptions
- Allow developers and teams to make decisions
- Could this meeting have been a Principle?

Ansible

Ansible is a radically simple IT automation system. It handles configuration management, application deployment, cloud provisioning, ad-hoc task execution, network automation, and multi-node orchestration. Ansible makes complex changes like zero-downtime rolling updates with load balancers easy. More information on the Ansible [website](#).

Design Principles

- Have an extremely simple setup process with a minimal learning curve.
- Manage machines quickly and in parallel.
- Avoid custom-agents and additional open ports, be agentless by leveraging the existing SSH daemon.
- Describe infrastructure in a language that is both machine and human friendly.

how should I add a new thing...?

Architectural Principles

Well-defined principles

Name: Succinct, easy to remember, and carries the essence of the rule (few words)

Statement: Unambiguous statement defining the rule (1 to 3 sentences)

Rationale: Why is this rule important, particularly in terms of business benefits.

Implications: Identifies explicitly how the rule affects the business, teams, etc.

6. Minimise the total number of communications protocols used across the entire enterprise. MQTT/OPCUA/REST/GRPC/etc.
7. Dependencies for communications should be minimised. For example for intra-machine installation of MQTT brokers on every machine.
8. "In the box" device to device communications should not require an external dependency on an MQTT broker
9. No software components should be interacting with a PLC to instruct it to move the functions, must be controlled by PLC
10. All payloads should contain sufficient information and identifiers so that they are self-describing and should not rely on URL parameters to be processed, or an MQTT payload should be self-describing
11. Where possible, only a single Device Service should be controlling a process since

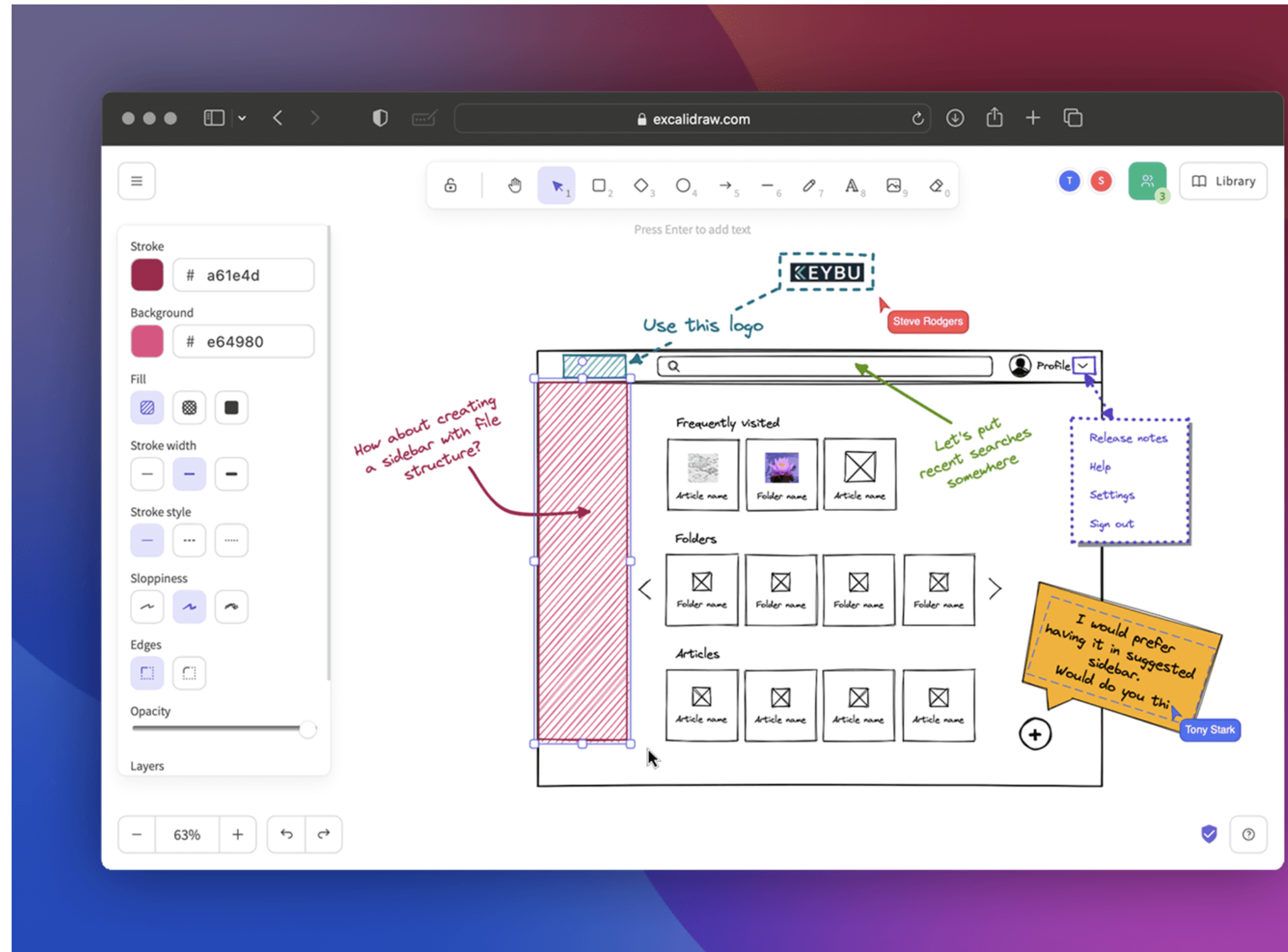
System, Solution, Software Architecture...

how does this all fit together now?

a picture is worth a thousand words...

Excalidraw

- Traditional diagramming
- Browser-based
- Open JSON format file
- Supports symbol libraries

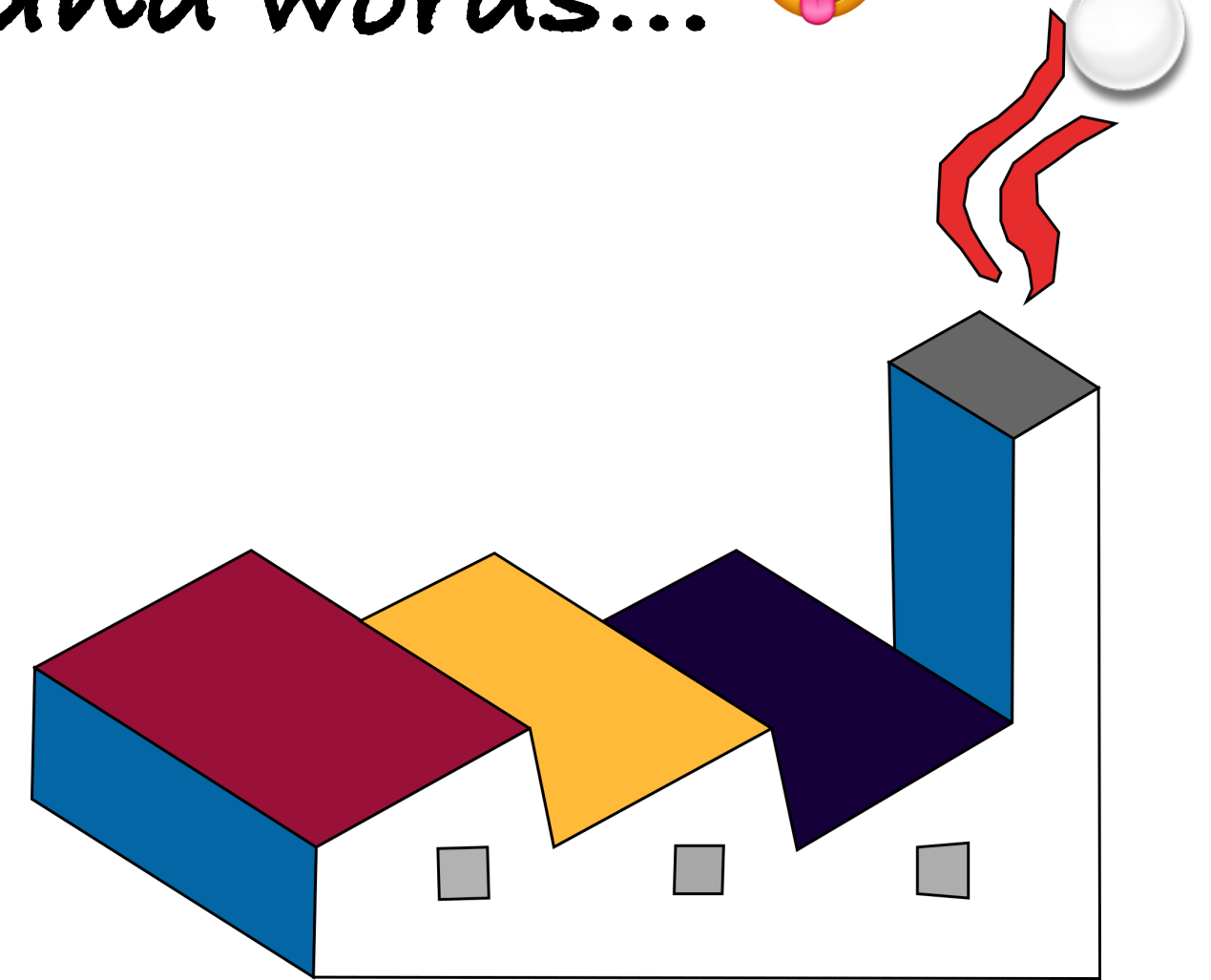


some code is also worth a thousand words...



PlantUML

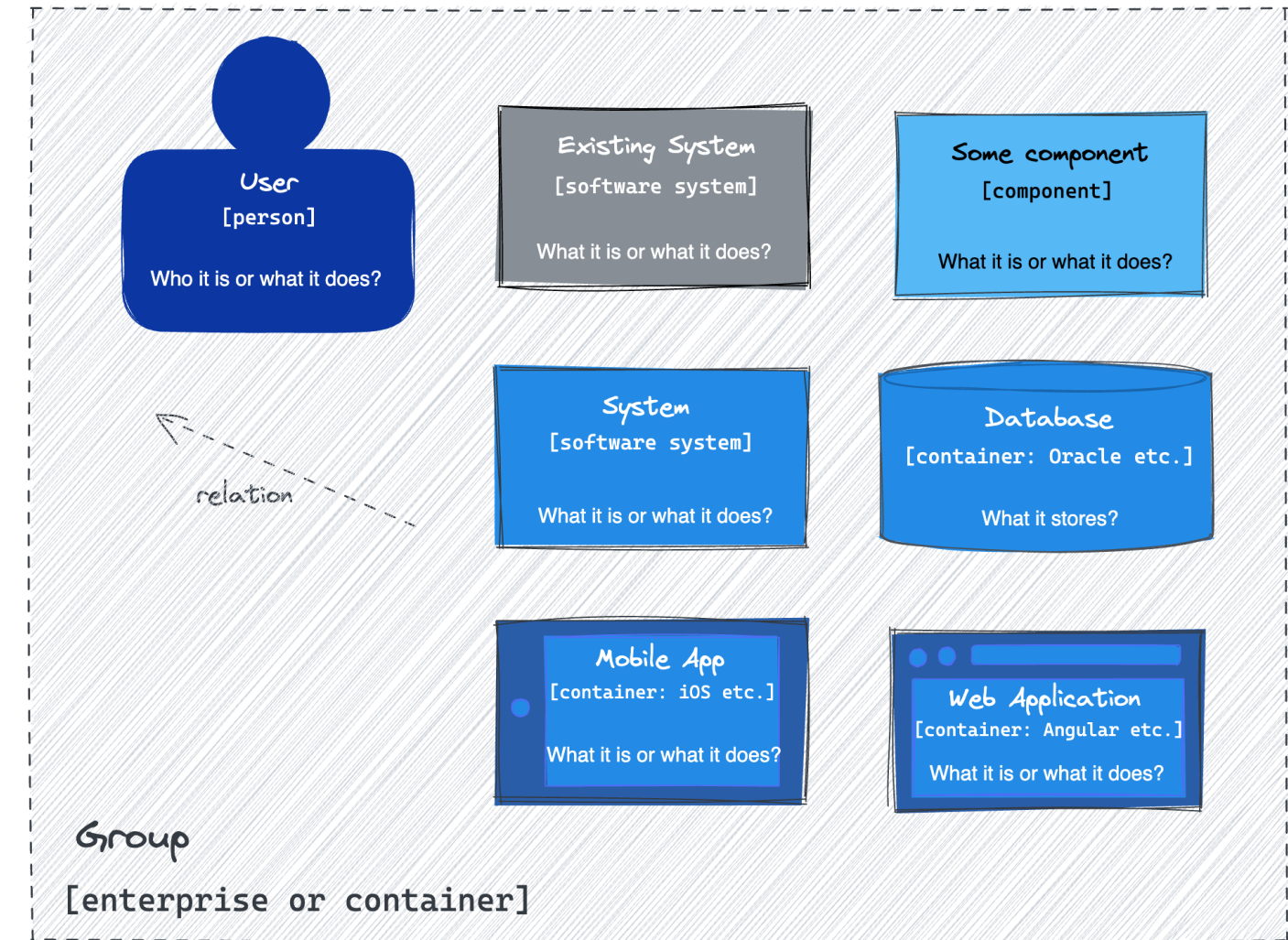
- Text-based diagramming
- Multiple diagram types
 - Sequence, Class, Activity, ERD
- Extensible with Libraries
 - Azure, Kubernetes, AWS, C4...
- Tooling
 - VS Code, JetBrains, Others...
 - Markdown wikis



* Mermaid is another good option

C4 Model

- Simple abstractions & Standard diagram types
 - Helps avoid overly-complex diagrams
1. Context: system scope, users and other systems
 2. Container: represents an application or a data store
 - web application, mobile app, serverless function, embedded device
 3. Component: internals of a container
 4. Code: class-level decomposition



Deployment: optional, and very useful

C4 Context Diagram

```
@startuml Context Demo
```

```
title C4 Context Diagram
```

```
!include <C4/C4_Context>  
!include <C4/C4_Container>
```

```
LAYOUT_WITH_LEGEND()
```

```
Person(user, "App User")  
Person(recipient, "Gift Recipient")
```

```
Container(app, "Hey Lemonade App", "React Native")
```

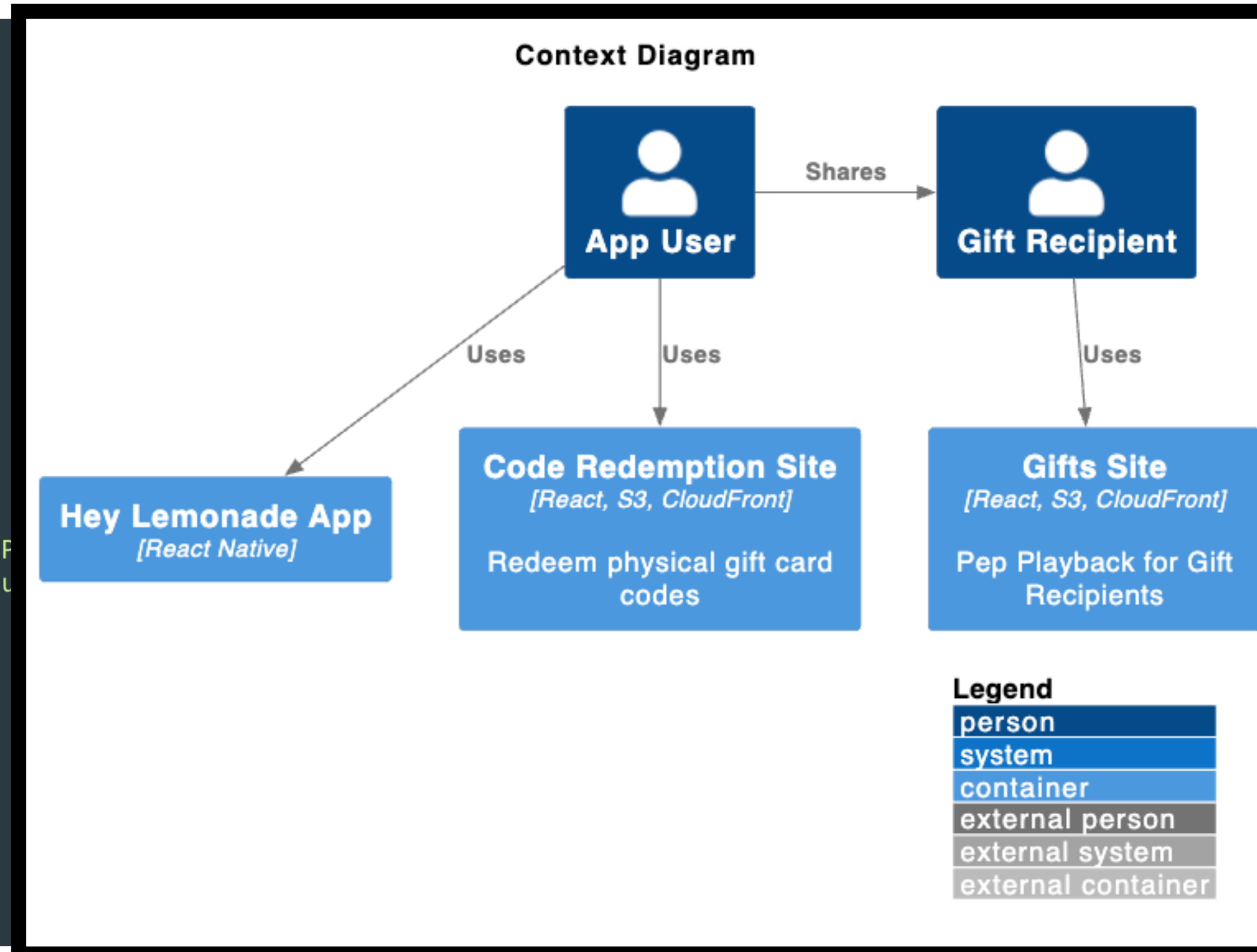
```
Container(gifts, "Gifts Site", "React, S3, CloudFront", "F")  
Container(redeem, "Code Redemption Site", "React, S3, Clou")
```

```
Rel(user, app, "Uses")  
Rel(user, redeem, "Uses")
```

```
Rel_R(user, recipient, "Shares")
```

```
Rel(recipient, gifts, "Uses")
```

```
@enduml
```



C4 Container Diagram

```

@startuml
!include <C4/C4_Component>

title Remote Service Agent

System_Boundary(product_system, "MediCorp Product") {
  Container_Ext(product, "Product Software", "MediCorp Product")
  Container_Ext(sks, "Secure Key Store", "x509 Key")
  Container_Ext(cert_store, "Certificate Store", "Trusted CA Root")
}

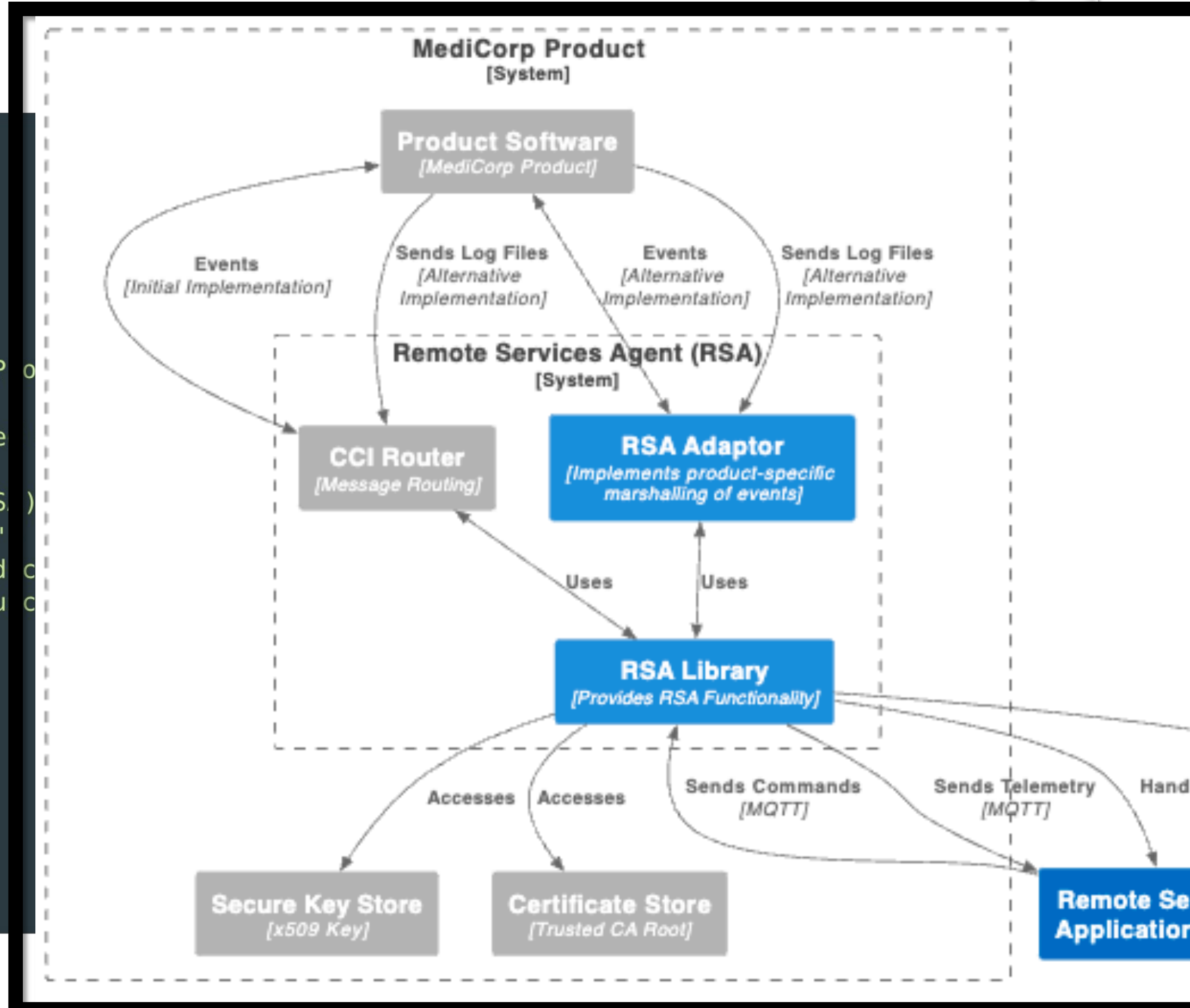
System_Boundary(rsa_system, "Remote Services Agent (RSA)") {
  Container_Ext(cci, "CCI Router", "Message Routing")
  Container_Ext(adaptor, "RSA Adaptor", "Implements product-specific marshalling of events")
  Container_Ext(library, "RSA Library", "Provides RSA Functionality")
}

System(rss, "Remote Services Application (RSS)", "")

BiRel(adaptor, library, "Uses", "")
BiRel(cci, library, "Uses", "")
Rel(library, sks, "Accesses", "")
Rel(library, cert_store, "Accesses", "")
Rel(library, remote_app, "Sends Commands [MQTT]", "")
Rel(library, remote_app, "Sends Telemetry [MQTT]", "")
Rel(remote_app, library, "Handled", "")

Product_Software -->|Events [Initial Implementation]| CCI_Router
Product_Software -->|Sends Log Files [Alternative Implementation]| RSA_Adaptor
CCI_Router -->|Events [Alternative Implementation]| Product_Software
RSA_Adaptor -->|Sends Log Files [Alternative Implementation]| Product_Software

```



Architecture Diagram

```
@startuml Deployment Demo
```

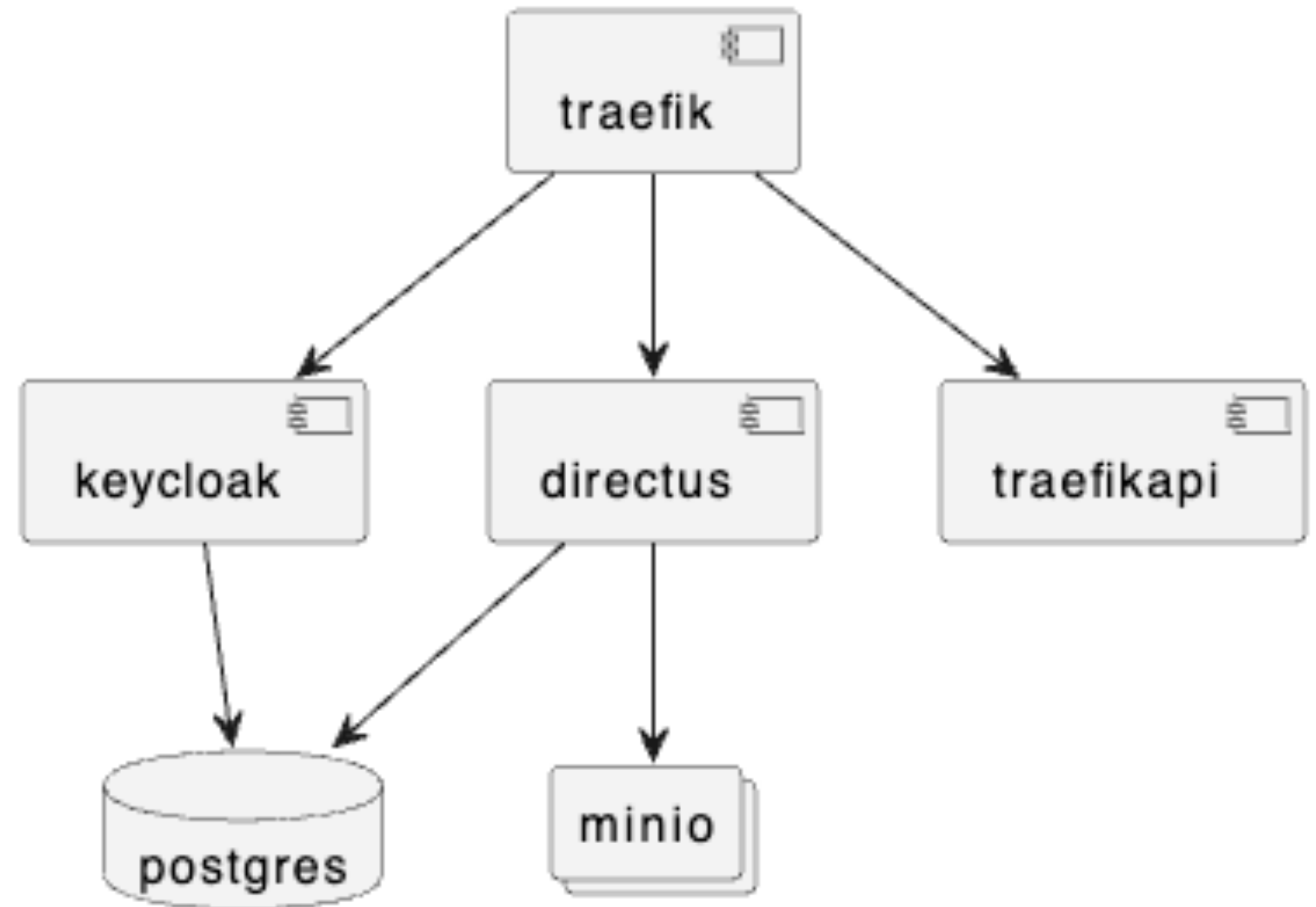
```
database postgres  
collections minio
```

```
[traefik] --> [keycloak]  
[traefik] --> [directus]  
[traefik] --> [traefikapi]
```

```
[keycloak] --> [postgres]  
[directus] --> [postgres]
```

```
[directus] --> [minio]
```

```
@enduml
```



Sequence Diagram

```

@startuml Sequence Demo

server -> aei      : BEGIN_PALLET_LOAD_SAGA
aei -> server     : PALLET_LOAD_SAGA_IN_PROGRESS

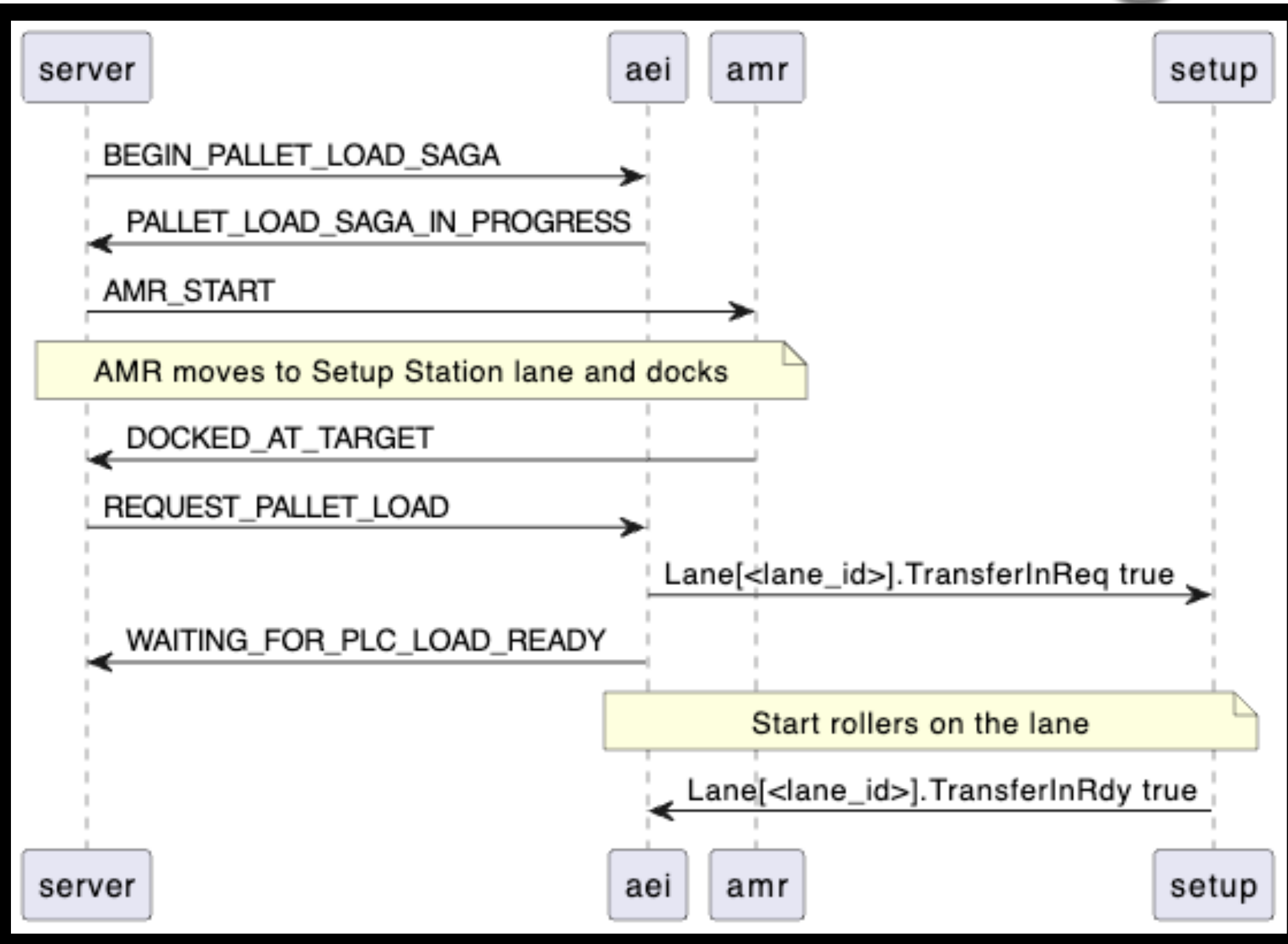
server -> amr     : AMR_START
note over server, amr: AMR moves to Setup Station lane

amr -> server     : DOCKED_AT_TARGET

server -> aei     : REQUEST_PALLET_LOAD
aei -> setup     : Lane[<lane_id>].TransferInReq true
aei -> server     : WAITING_FOR_PLC_LOAD_READY
note over aei,setup: Start rollers on the lane
setup -> aei     : Lane[<lane_id>].TransferInRdy true

@enduml

```



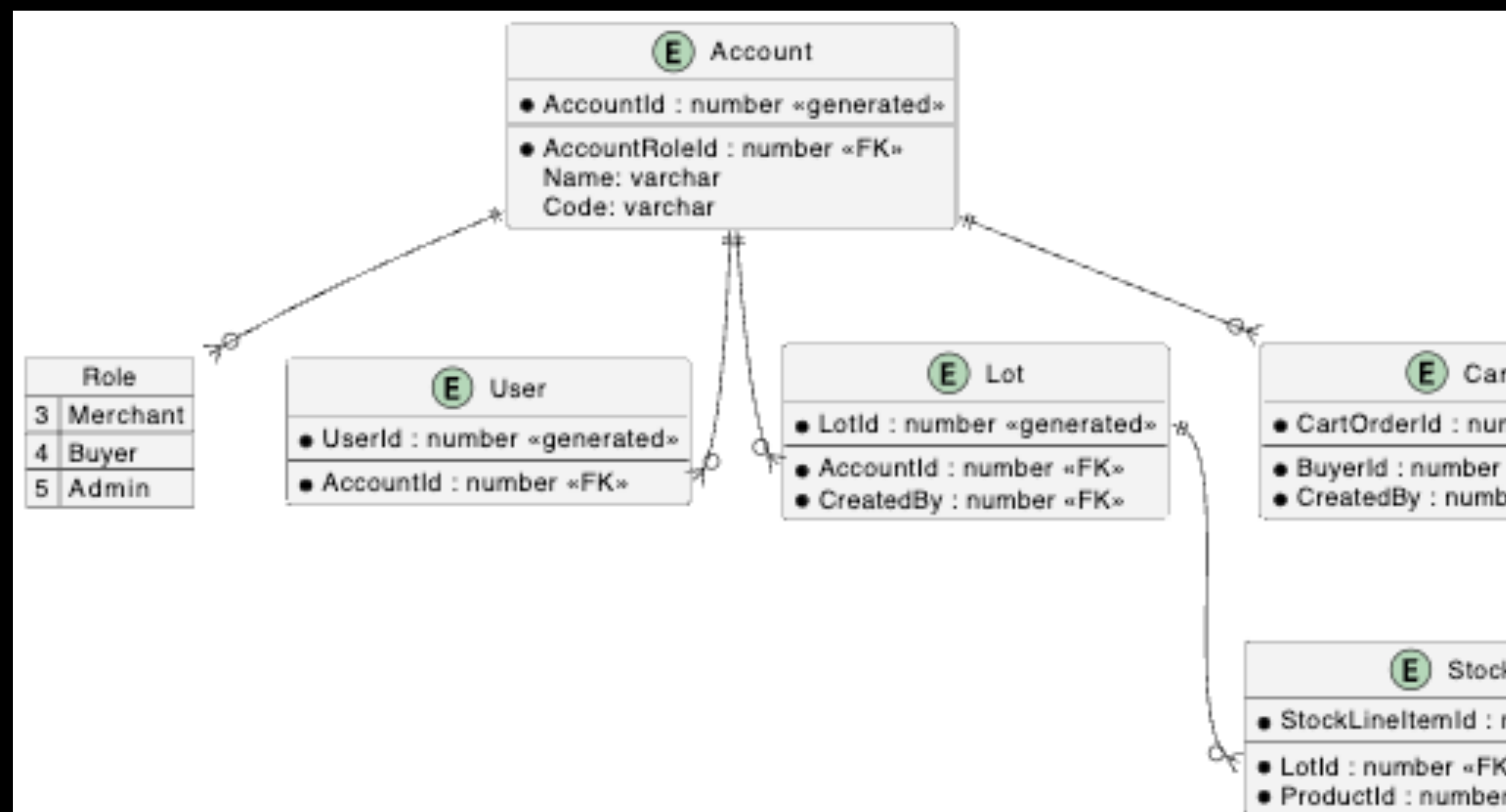
Entity Relationship Diagram

```
entity "Account" as account {
  *AccountId : number <<generated>>
  --
  *AccountRoleId : number <<FK>>
  Name: varchar
  Code: varchar
}

map "Role" as role {
  3 => Merchant
  4 => Buyer
  5 => Admin
}

account ||--o{ role

entity "User" as user {
  *UserId : number <<generated>>
  --
  *AccountId : number <<FK>>
}
```



Cloud Deployment Diagram

@startuml Deployment Diagram

```
Chat(webhooks_fb, "Facebook", "comments webhook")
Chat(webhooks_ig, "Instagram", "comments webhook")
```

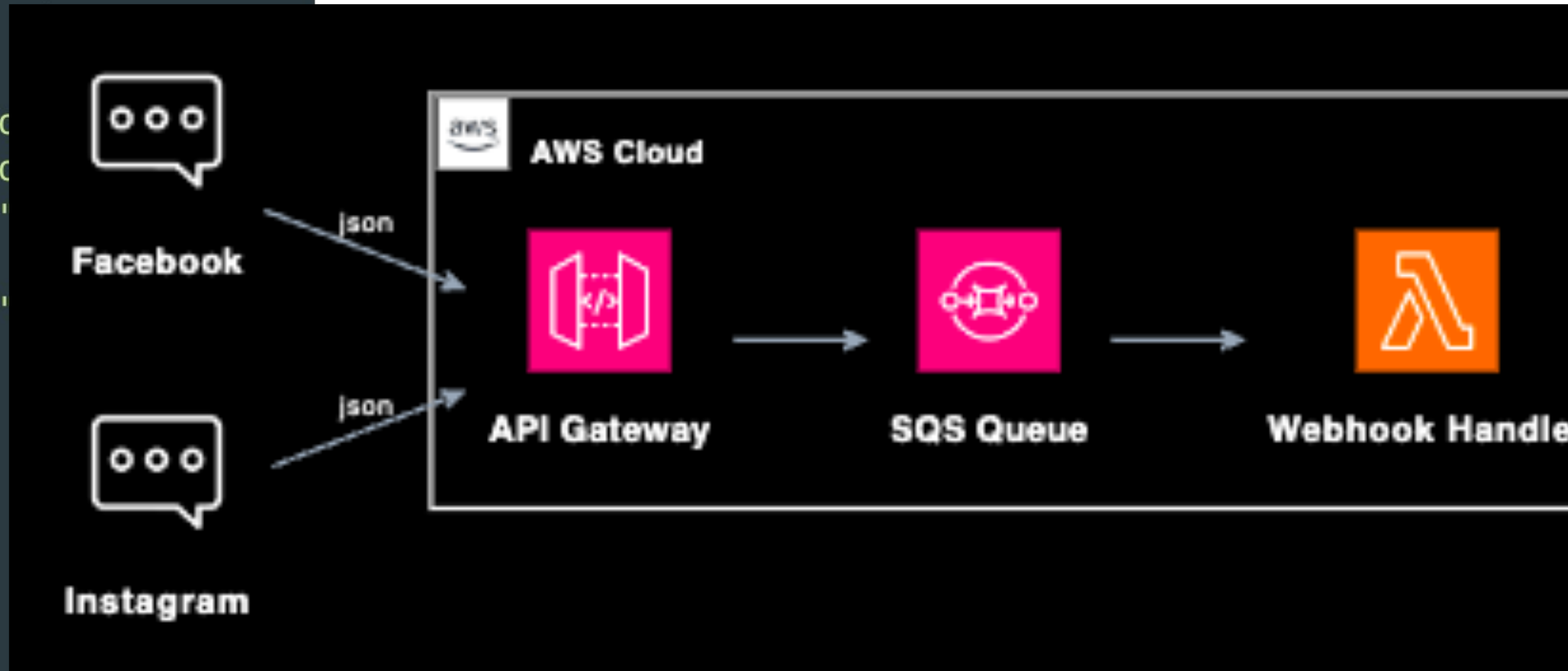
```
AWSCloudGroup(cloud) {
  APIGateway(webhookgw, "API Gateway", "webhook")
  SimpleQueueService(sqs, "SQS Queue", "webhook")
  Lambda(lambda, "Webhook Handler", "webhooks")
  AppRunner(api, "AppRunner API", "webhooks")
  AuroraPostgreSQLInstance(rds, "RDS Database")
}
```

```
webhooks_fb --> webhookgw : json
webhooks_ig --> webhookgw : json
```

```
webhookgw --> sqs : ? ?
sqs --> lambda : ? ?
lambda --> api #ff0000: ✨ urlencoded text ✨
```

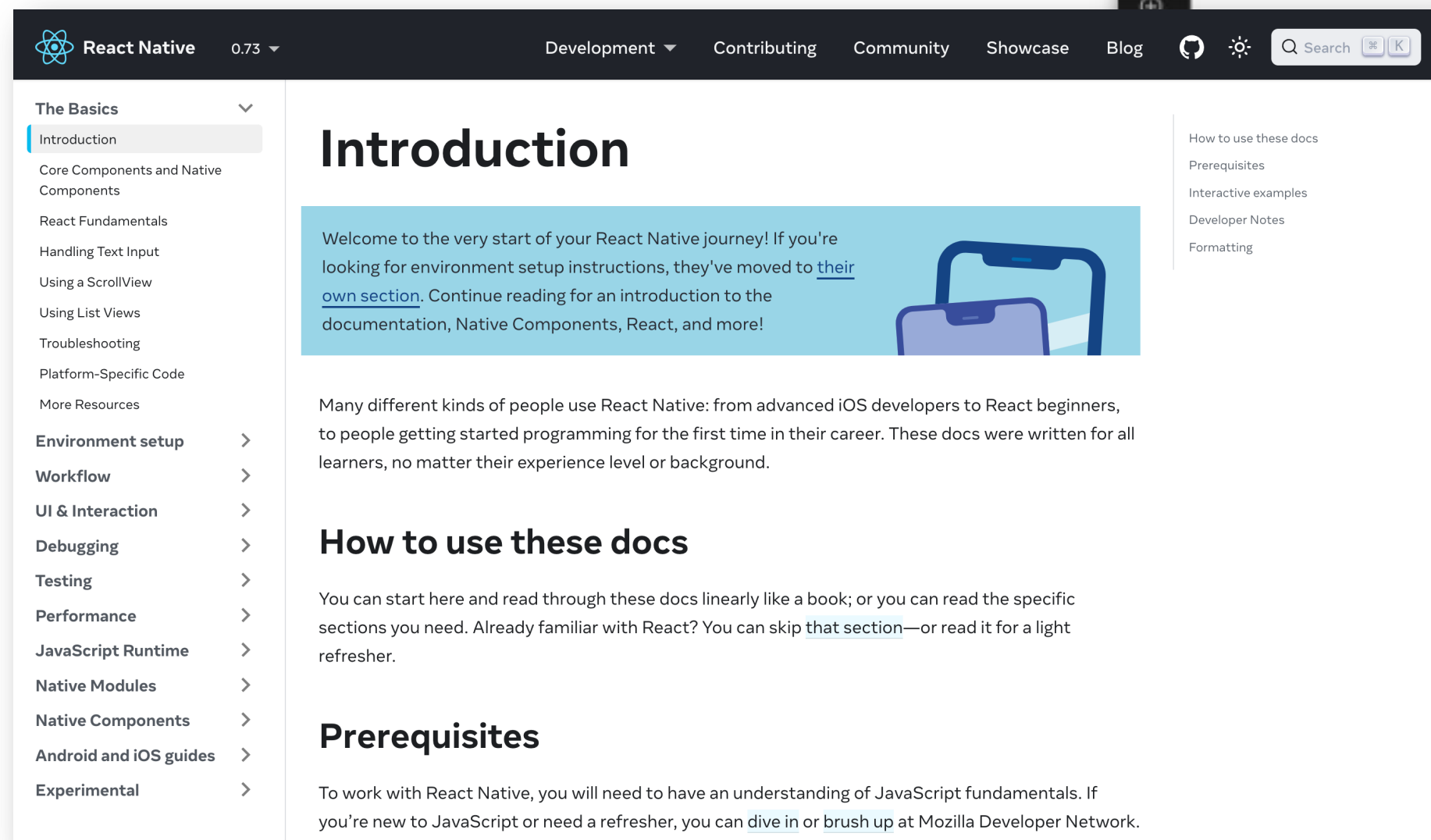
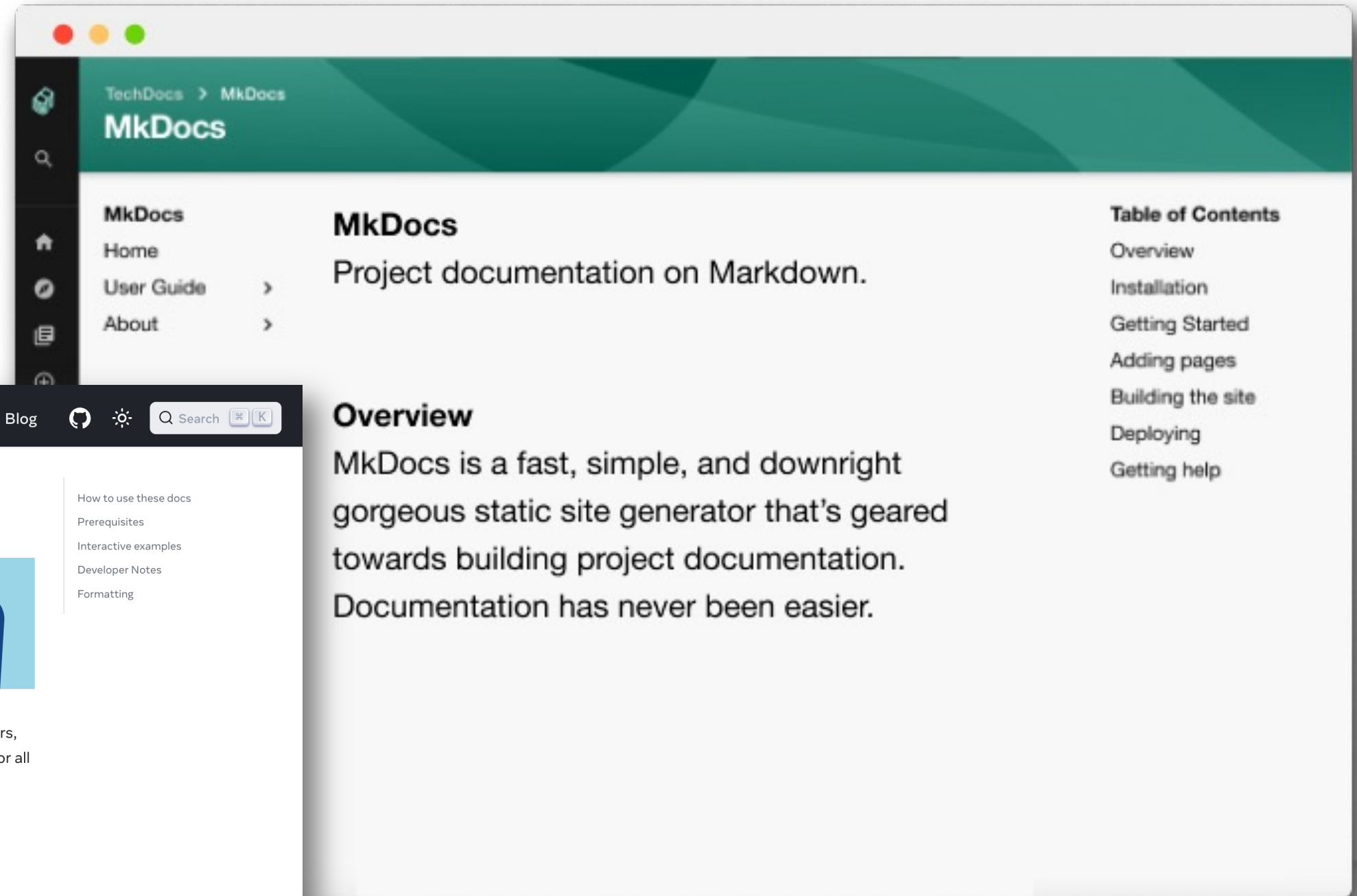
```
api --> rds
```

@enduml



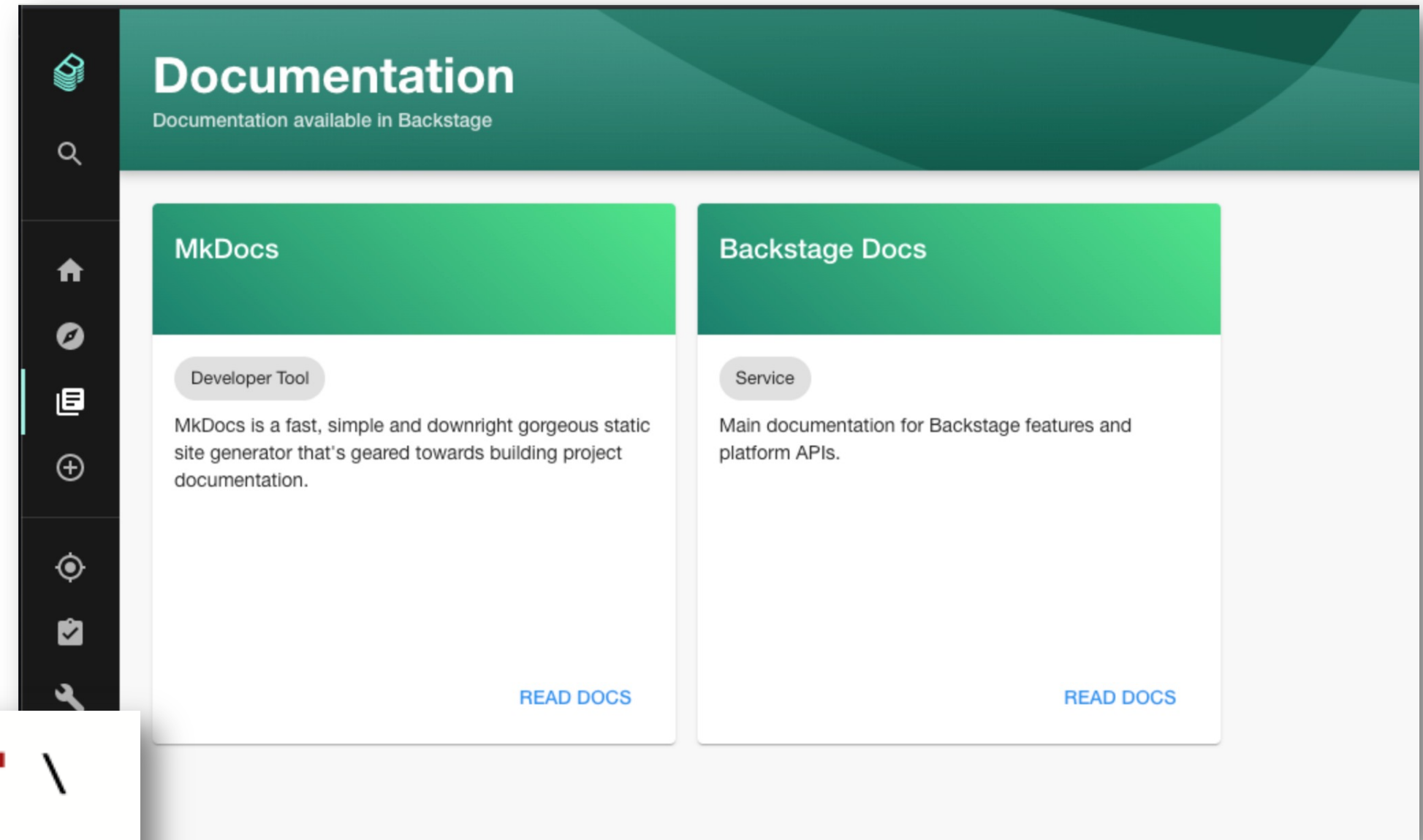
how can my friends see this?

- Mkdocs & Mkdocs Material 🥰
- Docusaurus 🦖



what if i have lots of friends?

- Render to PDF or DOCX - **Pandoc**
- Developer platform - **Backstage**
- Build and publish - **CI/CD**



```
pandoc --toc --reference-doc "$TEMPLATE" \  
-o "$OUTPUT" \  
--highlight-style "$CODE_STYLE" \  
--from=markdown+grid_tables \  
"$SOURCE"
```


Documentation for Developers

Key takeaways

Keep it simple with text-based approaches

Focus on content, not presentation

Minimise licensing requirements, reduce friction

Focus on documenting the gaps, and onboarding

Don't be "that developer", help your friends

Overview & Getting Started – Markdown READMEs

Old Decisions – Architecture Decision Records

New Decisions – Architectural Principles

System Architecture – Diagrams as Code, PlantUML & C4

Sharing & Scaling – Mkdocs, Docusaurus, Pandoc, CI/CD



Proudly sponsored by

Diamond
Sponsors



Platinum Sponsors



Gold Sponsors



Childcare by:



Wi-Fi by:

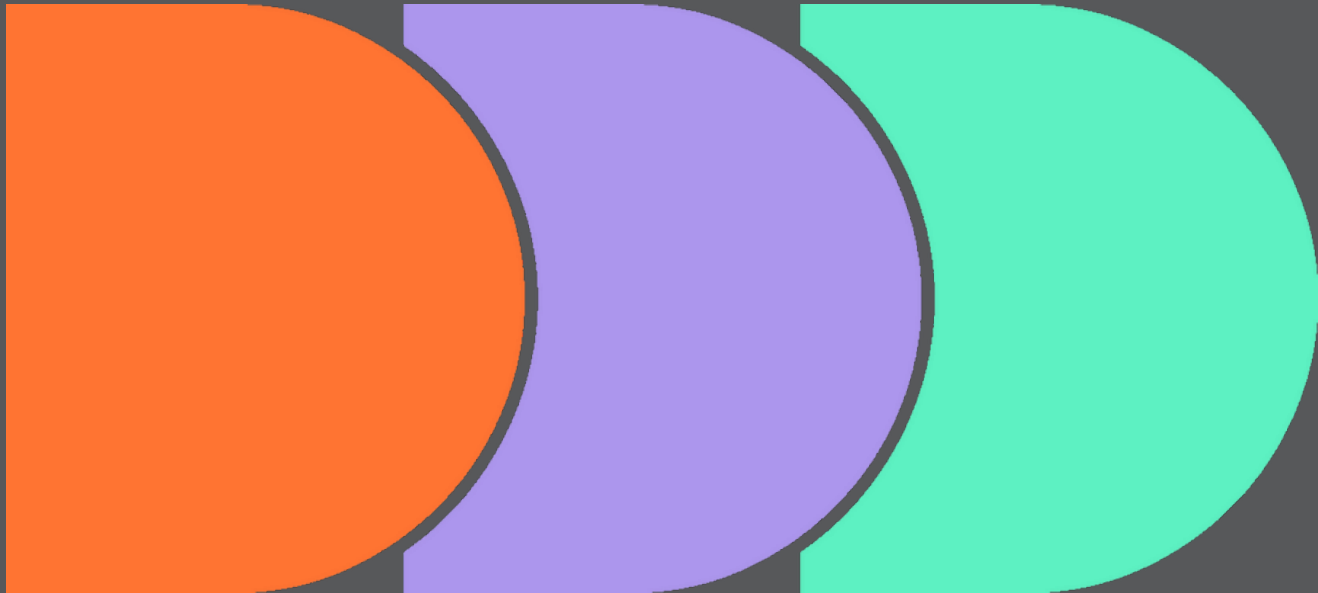


Coffee Cart by:



Silver Sponsors





MELBOURNE 2024