# Bank of America

## Code to Connect 2022 (GMOT)

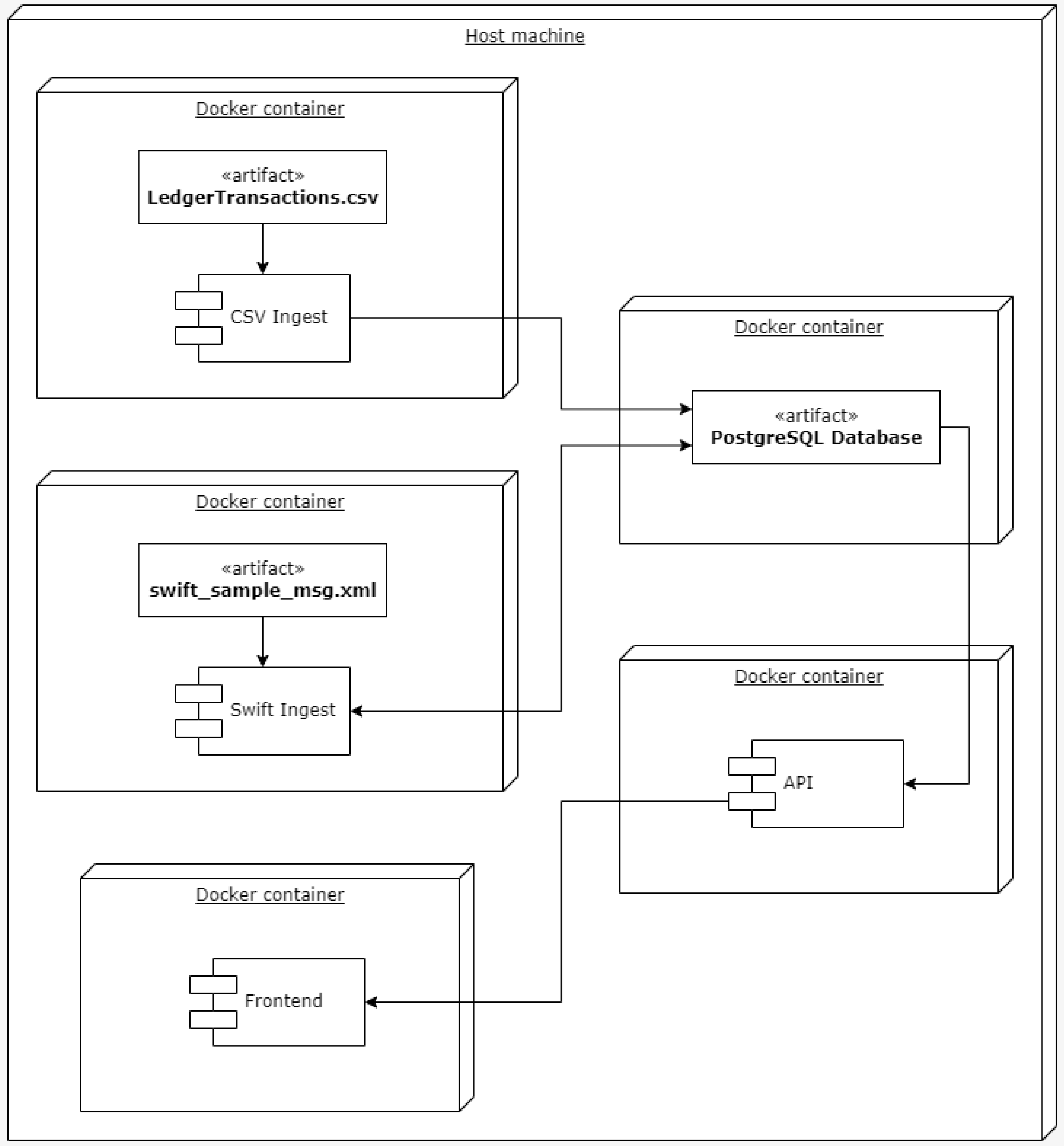
## Solution Presentation

Tan Qi En

## Objective

Design a **fault tolerant** reconciliation system that is **distributed** and capable of scaling to achieve **high throughput**

## Overview


Host machine
Docker container
«artifact»
LedgerTransactions.csv
CSV Ingest
Docker container
«artifact»
swift_sample_msg.xml
Swift Ingest
Docker container
«artifact»
PostgreSQL Database
Docker container
API
Docker container
Frontend

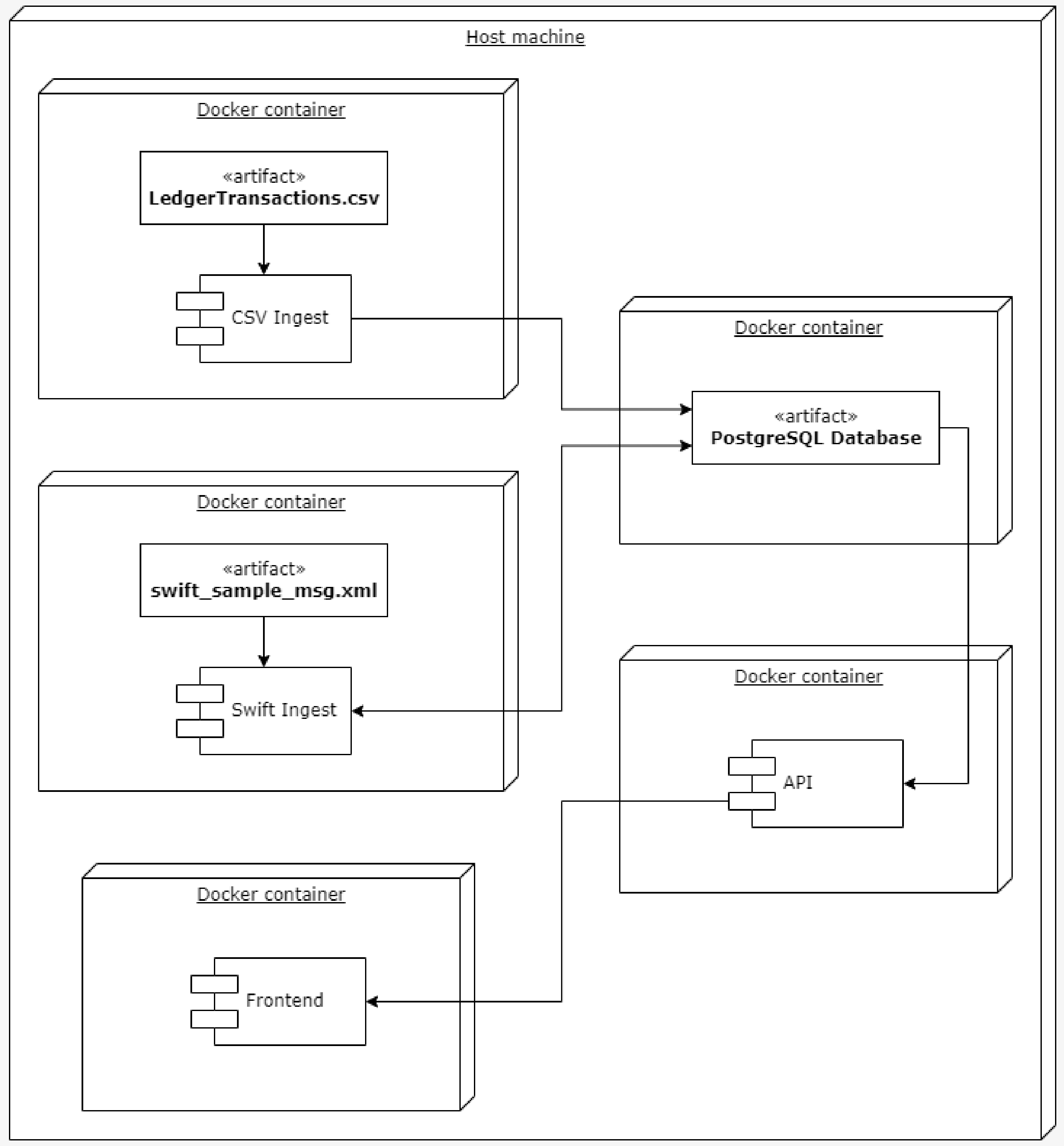
## Overview

**Fault tolerant**

Financial systems must recover from failure without causing discrepancies in data

Individual operations carried out by components are idempotent, so failed containers can be simply restarted without generating dirty data

Containers wait for their dependencies to run successfully before starting, preventing cascading failures where downstream containers fail by requesting unavailable resources



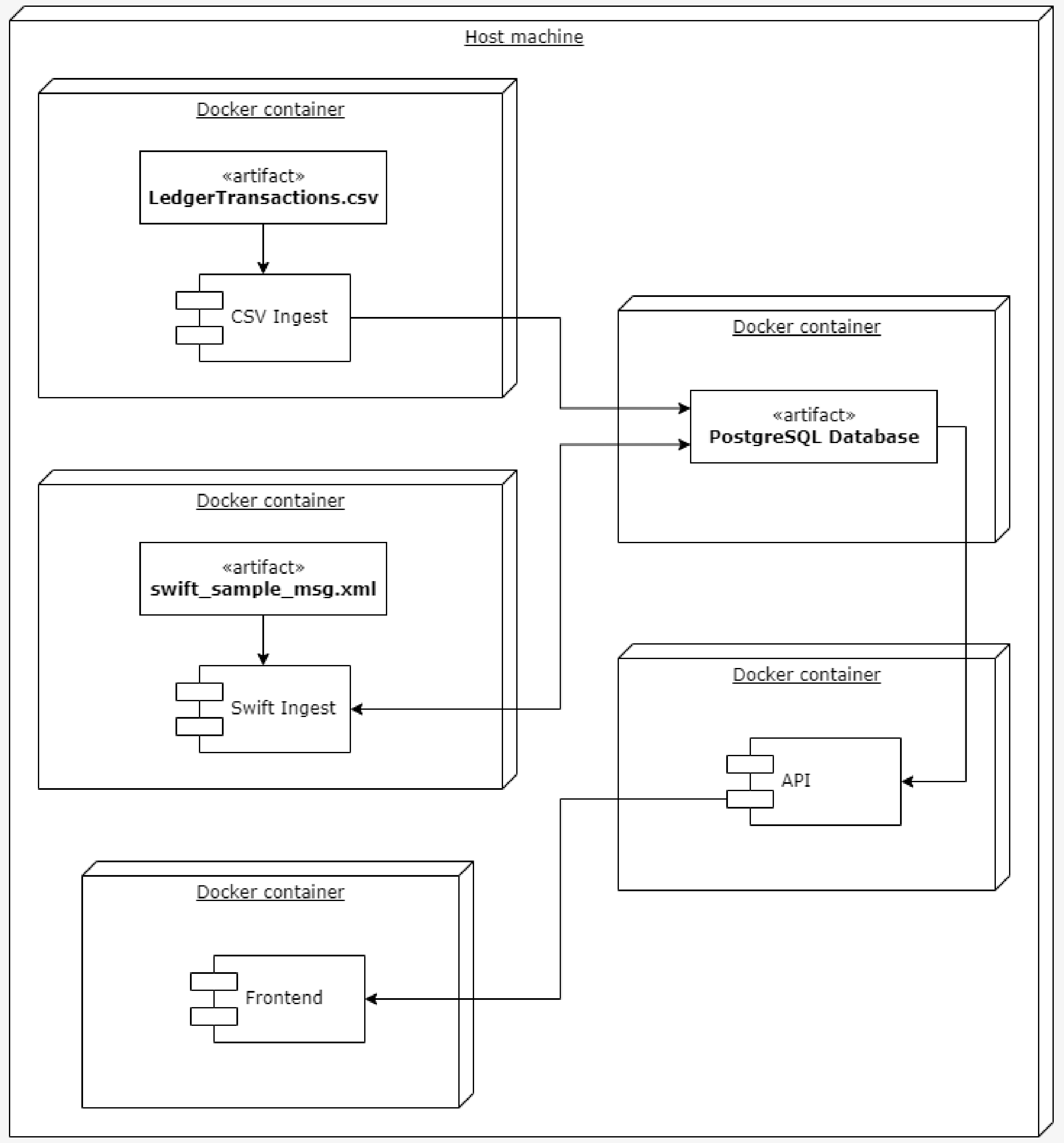
## Overview

### Distributed

With underlying systems that may themselves be distributed, this system must also be distributed to consume source data most efficiently

Containerization provides reproducibility of the system across different deployment environments

Containers can be easily configured to run on different physical machines, and orchestrated with technologies like Kubernetes



BANK OF AMERICA

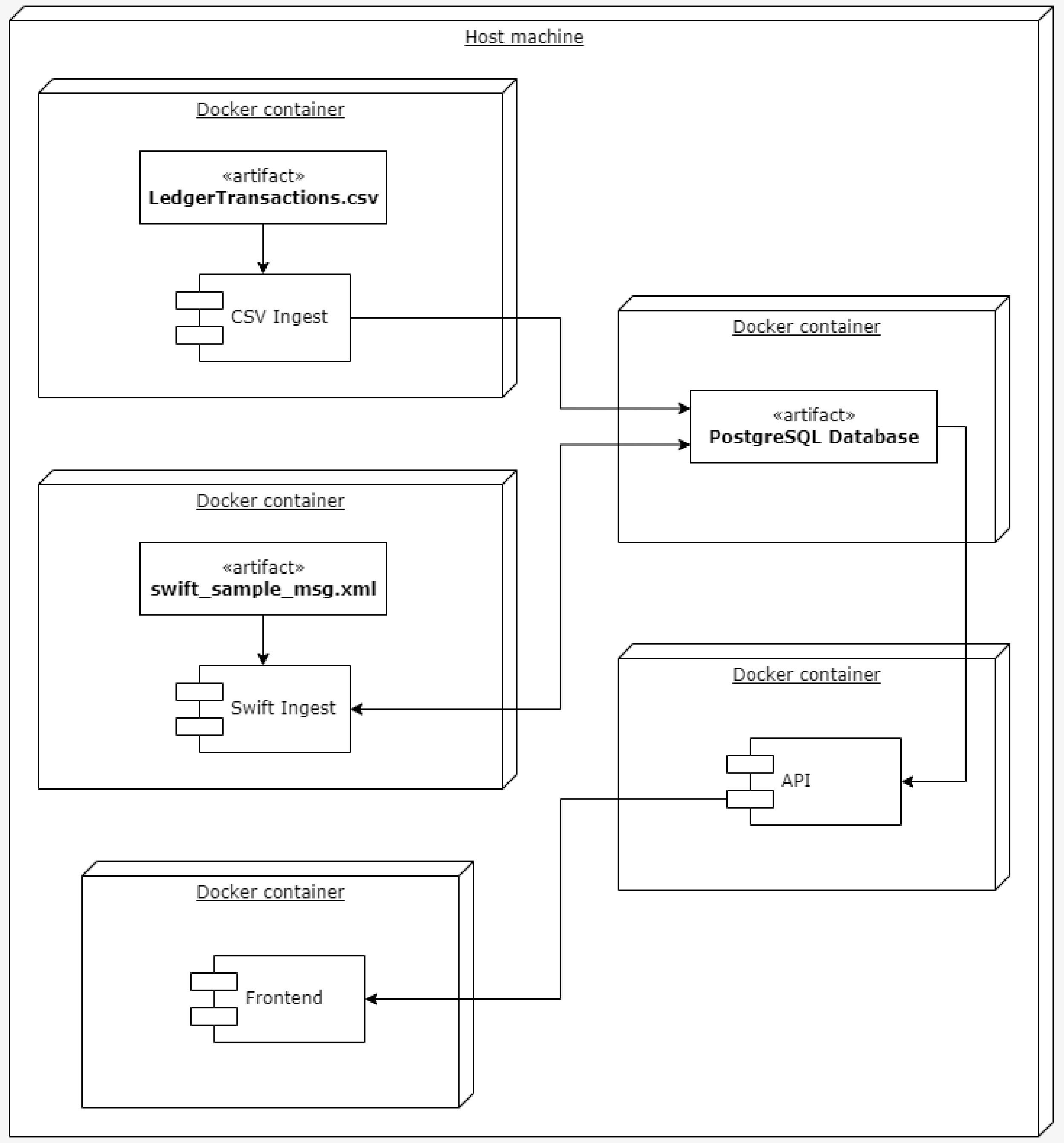
## Overview

### Scalability

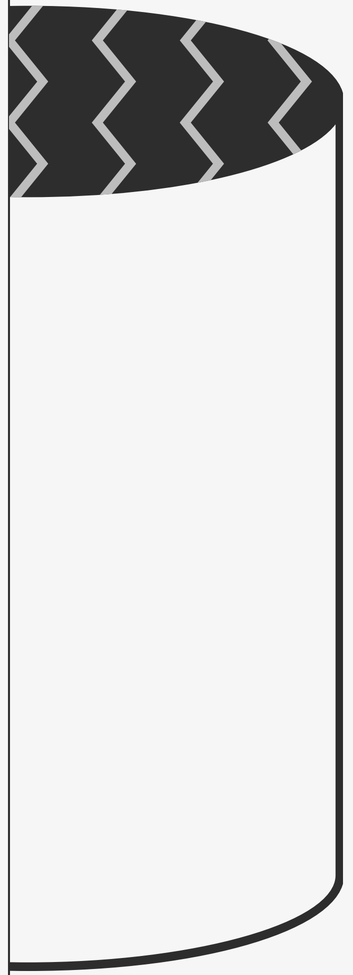
The system needs to be able to scale up to process millions of input files in batches

Services are separated into their own containers, and can be scaled independently to meet demand

Database can be configured to use sharding and read replicas to provide additional performance



## Assumptions



Reconciliation happens as a batch job periodically (weekly, monthly etc)

Balance records cover a booking period from 0000h to 2359h

If balances on a SWIFT message cannot be reconciled for a certain date, none of the transactions on that date should be used for reconciliation

Ledger CSV and SWIFT XML files are well formed with a consistent format

## Challenges & Solutions

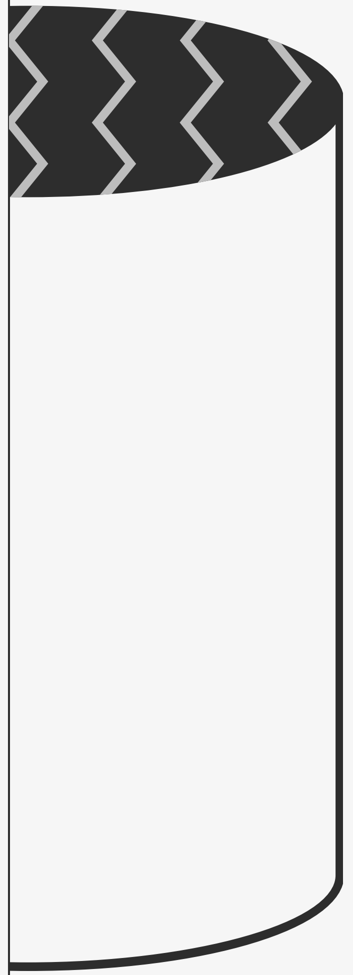
### Challenge

Managing multiple components and their dependencies individually became very challenging as they grew in number

### Solution

Used docker compose to package components into containers and orchestrate their deployment

## Challenges & Solutions



**Challenge**

Unit tests could not be run at container build time because modules to perform matching had dependencies on external components like database

**Solution**

Refactored modules to have a more cleanly defined separation of concern, placing database connection in a separate module

BANK OF AMERICA

## Challenges & Solutions

**Challenge**

Limited time for development due to short duration of challenge

**Solution**

Utilized existing frameworks and modules to minimize reinventing the wheel, focusing more on unique business logic. Wrote unit tests strategically for areas of code with more complicated logic.

## Bonus Questions

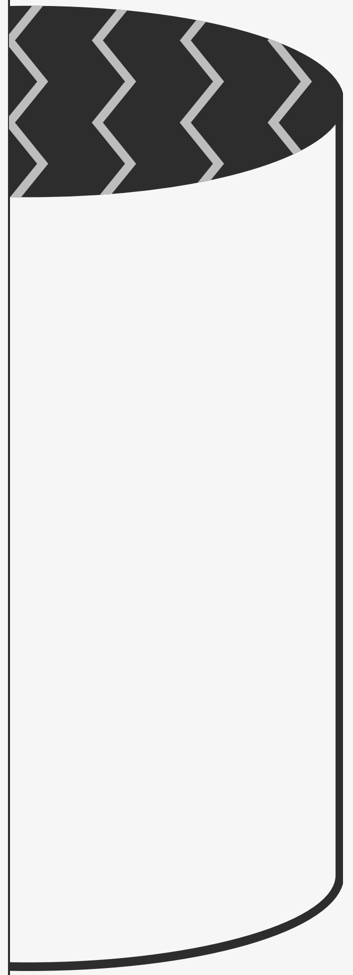
**What other types of SWIFT data validations can be performed?**

Ensure internal consistency, eg total amount for batch transaction is correct

Ensure the XML file is correctly structured

Ensure that field correspond to a list of permitted values

## Bonus Questions



**How can you ensure that there are no duplicate SWIFT messages?**

Assuming the SWIFT message ID is guaranteed to be unique, we can store a list of IDs received in a database, and refuse to process SWIFT messages with IDs already in the database.

Otherwise, the same principle can be applied to a hash of the entire message file instead of the message ID.

## Bonus Questions

**What are some things you will consider for matching performance?**

We perform 1 to 1 matching first as it can be done with a significantly faster algorithm (O(n) time vs O(2^n) for many to many)

This eliminates (hopefully) most of the transactions and leaves a smaller dataset for the slower algorithm to process

## Bonus Questions

**What are some things you will consider for matching performance?**

We consider 1 to many matching and many to many matching together since this may increase the number of potential matches.

We find all combinations within ledger and swift transactions, and attempt to match them simultaneously. Multiple many to many matches can be aggregated, allowing us to make all matches in a single pass.

BANK OF AMERICA

## Thank you!