Architecting for Scale

A case-study in utilizing a sharded architecture for infinite* scalability

Sumner Evans

27 September 2022

Beeper

A bit about me

My name is Sumner, I'm a **software engineer at Beeper**.

- I graduated from Colorado School of Mines in 2018 with my bachelor's in CS and 2019 with a master's in CS.
- I am an adjunct professor. Currently I'm teaching CSCI 400. I've taught 406, and 564 in the past as well.
- I enjoy skiing, volleyball, and soccer.
- I'm a 4th degree black belt in ATA taekwondo.

Overview

1. A bit about Beeper
2. What we've built
3. What is software architecture?
4. A bit about Beeper's current architecture
5. Our new architecture

This talk is interactive!

If you have questions at any point, feel free to interrupt me.

A bit about Beeper

Beeper's mission

Our mission is to:

> *make it easy for everyone on Earth to chat with each other.*

We specifically chose the word "chat" rather than "communicate" because we are focusing on *people talking to one another*.

Beeper's mission

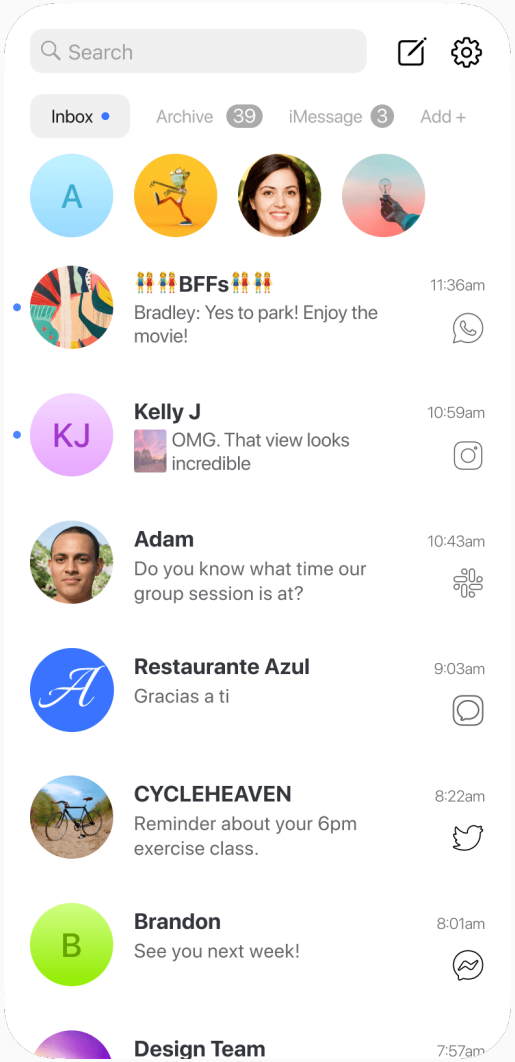
Our mission is to:

> *make it easy for everyone on Earth to chat with each other.*

We specifically chose the word "chat" rather than "communicate" because we are focusing on *people talking to one another*.

What is Beeper?

Beeper is an app that brings all of your chat networks together into a single inbox.



What is Beeper?

Beeper allows you to consolidate messages from 15+ chat apps into a single inbox.

- Beeper is available on on macOS, Windows, Linux iPhone, iPad, Android and Chrome OS.
- Beeper is built on top of the Matrix protocol.
- Beeper encrypts all chats, including bridged chats, by default[1].

[1] We have to momentarily decrypt your chat messages to translate from the other network to Matrix, but we never log or store your unencrypted messages.

What is Beeper?

Beeper allows you to consolidate messages from 15+ chat apps into a single inbox.

- Beeper is available on on macOS, Windows, Linux iPhone, iPad, Android and Chrome OS.
- Beeper is built on top of the Matrix protocol.
- Beeper encrypts all chats, including bridged chats, by default[1].

[1] We have to momentarily decrypt your chat messages to translate from the other network to Matrix, but we never log or store your unencrypted messages.

We are targeting individuals, not enterprises

We want to beat WhatsApp, not Slack.

Our core competency is the 50th highest priority at the big tech companies.

Most chat networks are an afterthought within the ecosystem of a larger company. At Beeper, chat is all we care about.

We are targeting individuals, not enterprises

We want to beat WhatsApp, not Slack.

Our core competency is the 50th highest priority at the big tech companies.

Most chat networks are an afterthought within the ecosystem of a larger company. At Beeper, chat is all we care about.

How we make money

We charge a flat $10/month fee to use Beeper. This fee allows users to connect as many chat networks as they want.

No ads. No data mining. We take advantage of Matrix's E2EE to add additional privacy guarantees.

The transaction is simple:
users pay us money for a service.

How we make money

We charge a flat $10/month fee to use Beeper. This fee allows users to connect as many chat networks as they want.

No ads. No data mining. We take advantage of Matrix's E2EE to add additional privacy guarantees.

The transaction is simple:
users pay us money for a service.

How we make money

We charge a flat $10/month fee to use Beeper. This fee allows users to connect as many chat networks as they want.

No ads. No data mining. We take advantage of Matrix's E2EE to add additional privacy guarantees.

The transaction is simple:
users pay us money for a service.

Bridging to a glorious Matrix future

We see bridges as a way to onboard users into the Matrix ecosystem.

Users can switch to a new app without loosing a single conversation!

We encourage our users to connect *all* of their chat networks to Beeper.

Bridging to a glorious Matrix future

We see bridges as a way to onboard users into the Matrix ecosystem.

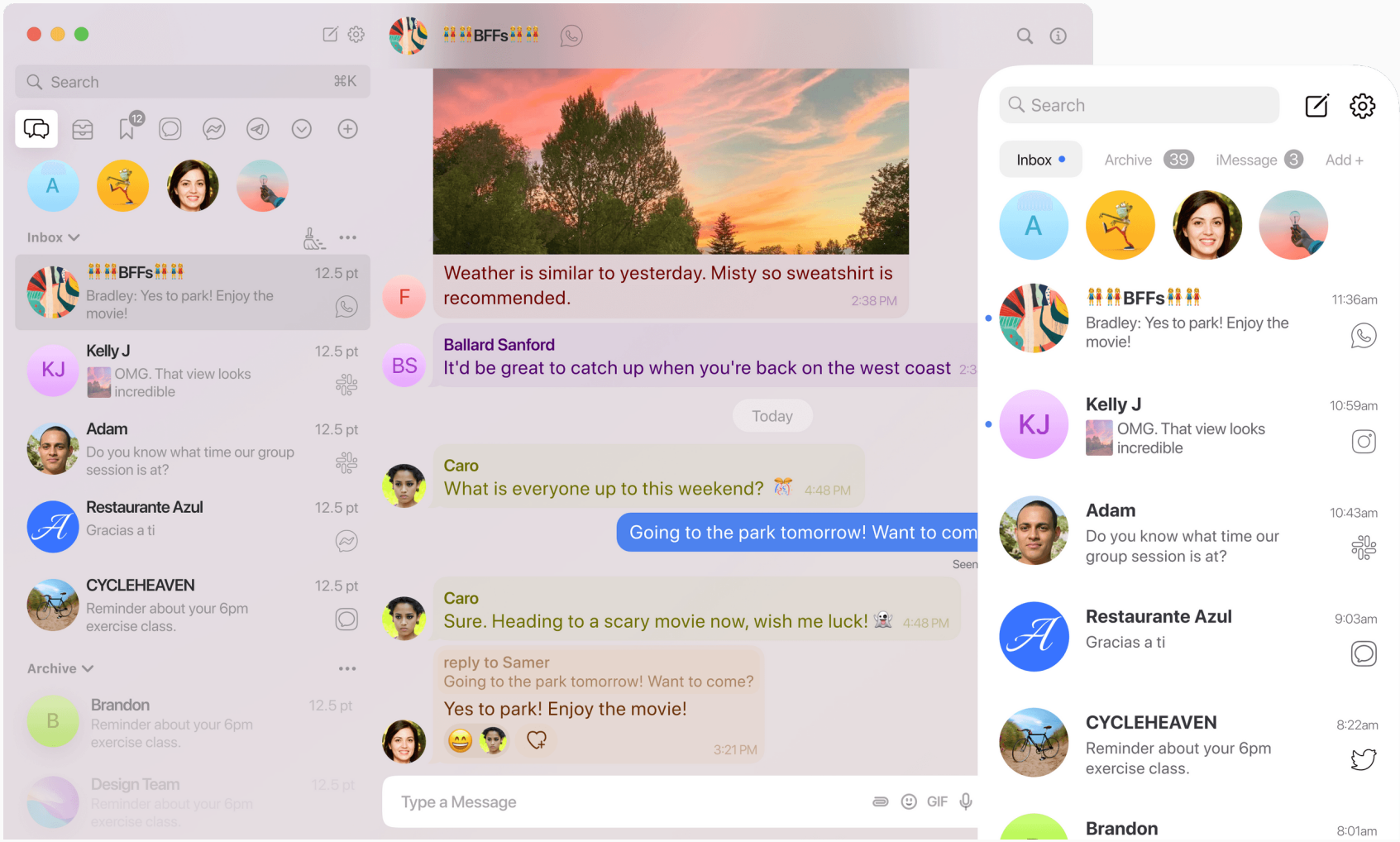
Users can switch to a new app without loosing a single conversation!

We encourage our users to connect *all* of their chat networks to Beeper.

What we’ve built

Custom clients for desktop and mobile

We forked Element, and added custom features.



Bridges, bridges, and more bridges!

We have built bridges to 10 networks:

- iMessage
- WhatsApp
- Facebook Messenger
- SMS (Android)
- Telegram
- Signal
- LinkedIn
- Instagram
- Twitter
- Google Chat

We are actively developing new bridges to Discord and Slack.

You can also connect to IRC (via the Libera.chat bridge) and the rest of the Matrix federation.

Bridges, bridges, and more bridges!

We have built bridges to 10 networks:

- iMessage
- WhatsApp
- Facebook Messenger
- SMS (Android)
- Telegram
- Signal
- LinkedIn
- Instagram
- Twitter
- Google Chat

We are actively developing new bridges to Discord and Slack.

You can also connect to IRC (via the Libera.chat bridge) and the rest of the Matrix federation.

Bridges, bridges, and more bridges!

We have built bridges to 10 networks:

- iMessage
- WhatsApp
- Facebook Messenger
- SMS (Android)
- Telegram
- Signal
- LinkedIn
- Instagram
- Twitter
- Google Chat

We are actively developing new bridges to Discord and Slack.

You can also connect to IRC (via the Libera.chat bridge) and the rest of the Matrix federation.

Demo!

What I work on at Beeper

I am on the newly created *Scaling* team.

Our current objective is to prepare Beeper for rocket-ship growth.

I was previously part of the *Bridges* team.

Notable projects included:

- Writing the LinkedIn bridge
- Adding bridge status reporting and message send status
- Implementing massive stability improvements in the Signal bridge
- Implementing message import in our WhatsApp and Facebook bridges

What I work on at Beeper

I am on the newly created *Scaling* team.

Our current objective is to prepare Beeper for rocket-ship growth.

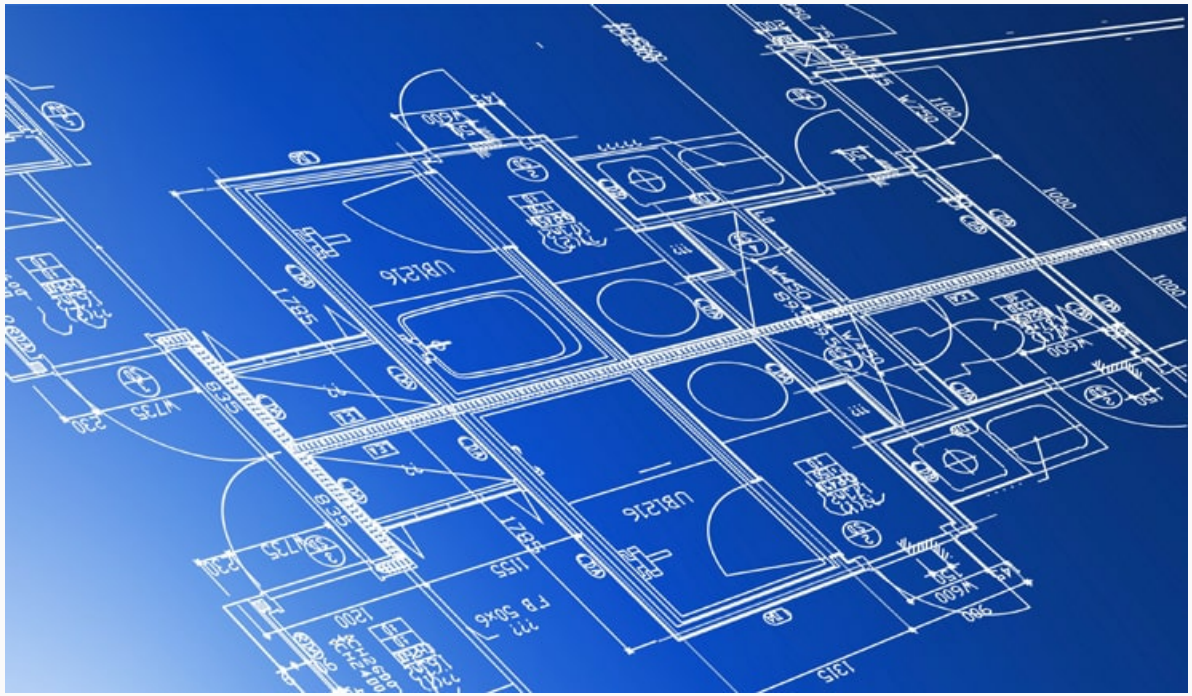
I was previously part of the *Bridges* team.

Notable projects included:

- Writing the LinkedIn bridge
- Adding bridge status reporting and message send status
- Implementing massive stability improvements in the Signal bridge
- Implementing message import in our WhatsApp and Facebook bridges

What is software architecture?

Software architecture is about component interactions



Just like architects have to think about the interactions between plumbing, electrical, HVAC, etc., software architects have to think about the interactions of components in the software system.

“The stack”

No, not the one you learn in Data Structures.

In software architecture, the stack is the set of tools, services, and languages used for the various components of the system.

The closer a system is to a user, the further *up* the stack it is.

The further up in the stack you are, the closer to the frontend you are.

“The stack”

No, not the one you learn in Data Structures.

In software architecture, the stack is the set of tools, services, and languages used for the various components of the system.

The closer a system is to a user, the further *up* the stack it is.

The further up in the stack you are, the closer to the frontend you are.

“The stack”

No, not the one you learn in Data Structures.

In software architecture, the stack is the set of tools, services, and languages used for the various components of the system.

The closer a system is to a user, the further *up* the stack it is.

The further up in the stack you are, the closer to the frontend you are.

What do you think software architecture is?

- What questions do software architects ask?
- What concerns do software architects have to take into account?
- What tradeoffs do software architects have to consider?

What do you think software architecture is?

- What questions do software architects ask?
- What concerns do software architects have to take into account?
- What tradeoffs do software architects have to consider?

What do you think software architecture is?

- What questions do software architects ask?
- What concerns do software architects have to take into account?
- What tradeoffs do software architects have to consider?

Examples of questions

- How and where is the code going to be deployed? (CI/CD, Kubernetes, regions, etc.)
- Which systems depend on each other? (DBs, orchestrator process, etc.)
- What dependencies are you using in each component? (security, corporate policies, code quality, etc.)
- What languages are you going to use? (different for front/backend)?
- What components are we going to write ourselves, and which are we going to rely on open source or proprietary solutions for?
- How will the various components communicate with one another (GRPC, Rest, GraphQL, etc.)
- How many resources should you allocate to each system? (memory, disk, CPU, etc.)

Examples of questions

- How and where is the code going to be deployed? (CI/CD, Kubernetes, regions, etc.)
- Which systems depend on each other? (DBs, orchestrator process, etc.)
- What dependencies are you using in each component? (security, corporate policies, code quality, etc.)
- What languages are you going to use? (different for front/backend)?
- What components are we going to write ourselves, and which are we going to rely on open source or proprietary solutions for?
- How will the various components communicate with one another (GRPC, Rest, GraphQL, etc.)
- How many resources should you allocate to each system? (memory, disk, CPU, etc.)

Examples of questions

- How and where is the code going to be deployed? (CI/CD, Kubernetes, regions, etc.)
- Which systems depend on each other? (DBs, orchestrator process, etc.)
- What dependencies are you using in each component? (security, corporate policies, code quality, etc.)
- What languages are you going to use? (different for front/backend)?
- What components are we going to write ourselves, and which are we going to rely on open source or proprietary solutions for?
- How will the various components communicate with one another (GRPC, Rest, GraphQL, etc.)
- How many resources should you allocate to each system? (memory, disk, CPU, etc.)

Examples of questions

- How and where is the code going to be deployed? (CI/CD, Kubernetes, regions, etc.)
- Which systems depend on each other? (DBs, orchestrator process, etc.)
- What dependencies are you using in each component? (security, corporate policies, code quality, etc.)
- What languages are you going to use? (different for front/backend)?
- What components are we going to write ourselves, and which are we going to rely on open source or proprietary solutions for?
- How will the various components communicate with one another (GRPC, Rest, GraphQL, etc.)
- How many resources should you allocate to each system? (memory, disk, CPU, etc.)

Examples of questions

- How and where is the code going to be deployed? (CI/CD, Kubernetes, regions, etc.)
- Which systems depend on each other? (DBs, orchestrator process, etc.)
- What dependencies are you using in each component? (security, corporate policies, code quality, etc.)
- What languages are you going to use? (different for front/backend)?
- What components are we going to write ourselves, and which are we going to rely on open source or proprietary solutions for?
- How will the various components communicate with one another (GRPC, Rest, GraphQL, etc.)
- How many resources should you allocate to each system? (memory, disk, CPU, etc.)

Examples of questions

- How and where is the code going to be deployed? (CI/CD, Kubernetes, regions, etc.)
- Which systems depend on each other? (DBs, orchestrator process, etc.)
- What dependencies are you using in each component? (security, corporate policies, code quality, etc.)
- What languages are you going to use? (different for front/backend)?
- What components are we going to write ourselves, and which are we going to rely on open source or proprietary solutions for?
- How will the various components communicate with one another (GRPC, Rest, GraphQL, etc.)
- How many resources should you allocate to each system? (memory, disk, CPU, etc.)

Examples of questions

- How and where is the code going to be deployed? (CI/CD, Kubernetes, regions, etc.)
- Which systems depend on each other? (DBs, orchestrator process, etc.)
- What dependencies are you using in each component? (security, corporate policies, code quality, etc.)
- What languages are you going to use? (different for front/backend)?
- What components are we going to write ourselves, and which are we going to rely on open source or proprietary solutions for?
- How will the various components communicate with one another (GRPC, Rest, GraphQL, etc.)
- How many resources should you allocate to each system? (memory, disk, CPU, etc.)

Some more common architectural patterns and tools

Patterns

- **Microservices:** split your service into logical smaller services and use IPC between services
- **Load balancing:** split traffic across multiple services
- **Sharding:** split data across independent services

Tools

- **Docker:** allows you to bundle runtime dependencies with your executable in a *container*
- **Kubernetes:** allows you to orchestrate containers and describe things like dependencies between them

Some more common architectural patterns and tools

Patterns

- **Microservices:** split your service into logical smaller services and use IPC between services
- **Load balancing:** split traffic across multiple services
- **Sharding:** split data across independent services

Tools

- **Docker:** allows you to bundle runtime dependencies with your executable in *a container*
- **Kubernetes:** allows you to orchestrate containers and describe things like dependencies between them

Some more common architectural patterns and tools

Patterns

- **Microservices:** split your service into logical smaller services and use IPC between services
- **Load balancing:** split traffic across multiple services
- **Sharding:** split data across independent services

Tools

- **Docker:** allows you to bundle runtime dependencies with your executable in *a container*
- **Kubernetes:** allows you to orchestrate containers and describe things like dependencies between them

Some more common architectural patterns and tools

Patterns

- **Microservices:** split your service into logical smaller services and use IPC between services
- **Load balancing:** split traffic across multiple services
- **Sharding:** split data across independent services

Tools

- **Docker:** allows you to bundle runtime dependencies with your executable in a *container*
- **Kubernetes:** allows you to orchestrate containers and describe things like dependencies between them

Some more common architectural patterns and tools

Patterns

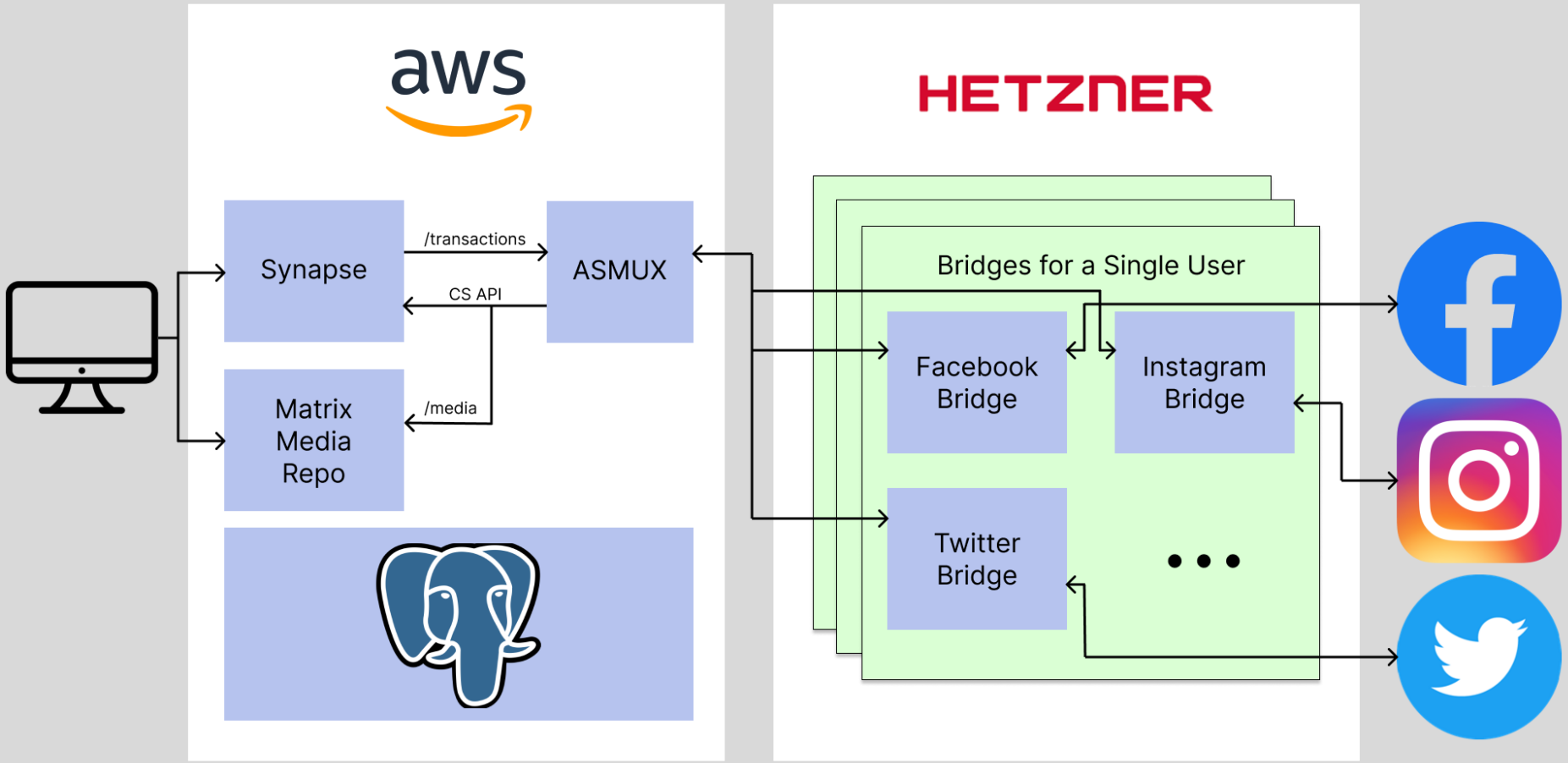
- **Microservices:** split your service into logical smaller services and use IPC between services
- **Load balancing:** split traffic across multiple services
- **Sharding:** split data across independent services

Tools

- **Docker:** allows you to bundle runtime dependencies with your executable in a *container*
- **Kubernetes:** allows you to orchestrate containers and describe things like dependencies between them

A bit about Beeper’s current architecture

A diagram



Let's look at the makeup of the current architecture.

Existing base

We are using existing open source projects for a lot of our architecture.

Synapse is an existing Matrix homeserver and we use the open source **PostgreSQL** database.

All of the bridges are also open source.

Existing base

We are using existing open source projects for a lot of our architecture.

Synapse is an existing Matrix homeserver and we use the open source **PostgreSQL** database.

All of the bridges are also open source.

Deployment

We are deploying on **AWS** and **Hetzner**. We use **Kubernetes** to manage all of the services.

Synapse is where all of the messages (events) are stored, so it has to be fast and reliable. It runs in AWS.

We run one bridge per user for every external network they connect to. This is for security (process isolation) and flexibility.

- We can deploy **different bridge versions** to different sets of users
- Bridges can be stopped and started individually

Deployment

We are deploying on **AWS** and **Hetzner**. We use **Kubernetes** to manage all of the services.

Synapse is where all of the messages (events) are stored, so it has to be fast and reliable. It runs in AWS.

We run one bridge per user for every external network they connect to. This is for security (process isolation) and flexibility.

- We can deploy **different bridge versions** to different sets of users
- Bridges can be stopped and started individually

Deployment

We are deploying on **AWS** and **Hetzner**. We use **Kubernetes** to manage all of the services.

Synapse is where all of the messages (events) are stored, so it has to be fast and reliable. It runs in AWS.

We run one bridge per user for every external network they connect to. This is for security (process isolation) and flexibility.

- We can deploy **different bridge versions** to different sets of users
- Bridges can be stopped and started individually

Deployment

We are deploying on **AWS** and **Hetzner**. We use **Kubernetes** to manage all of the services.

Synapse is where all of the messages (events) are stored, so it has to be fast and reliable. It runs in AWS.

We run one bridge per user for every external network they connect to. This is for security (process isolation) and flexibility.

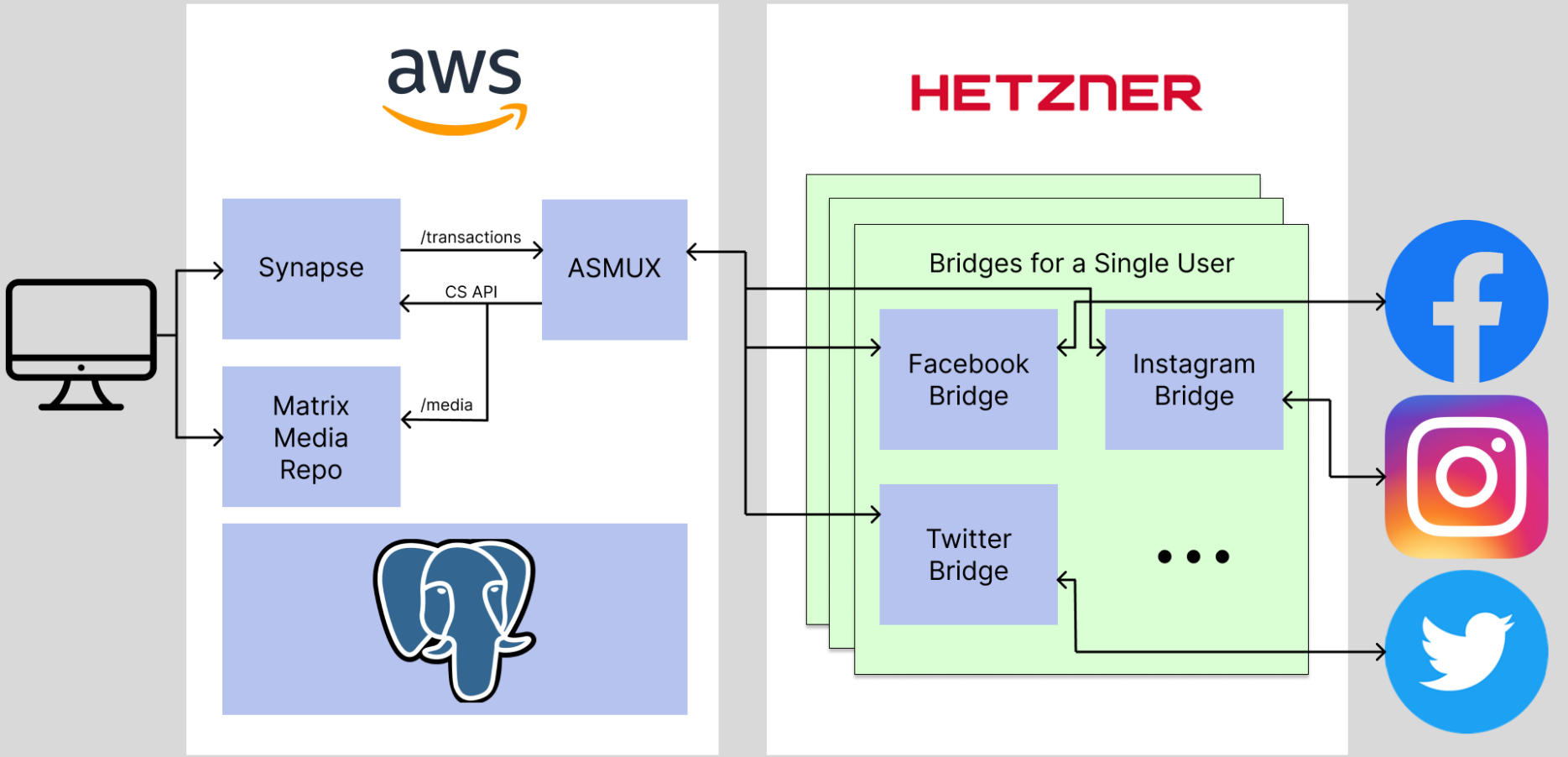
- We can deploy **different bridge versions** to different sets of users
- Bridges can be stopped and started individually

Custom software

Bridges normally connect directly to Synapse. However, Synapse has no way to hotswap bridges.

We wrote ASMUX as an intermediate service to allow for hotswapping of bridges.

Back to the diagram



There are some disadvantages to this architecture...

Disadvantages of our architecture

- We have to run a **lot** of bridges
- We run Synapse on AWS which is relatively expensive
- Synapse becomes a bottleneck for all traffic

We've reached the extent of how far we can scale with our current architecture.

Don't scale until you have to

Don't architect your system for scale until you need to scale.

Disadvantages of our architecture

- We have to run a **lot** of bridges
- We run Synapse on AWS which is relatively expensive
- Synapse becomes a bottleneck for all traffic

We've reached the extent of how far we can scale with our current architecture.

Don't scale until you have to

Don't architect your system for scale until you need to scale.

Disadvantages of our architecture

- We have to run a **lot** of bridges
- We run Synapse on AWS which is relatively expensive
- Synapse becomes a bottleneck for all traffic

We've reached the extent of how far we can scale with our current architecture.

Don't scale until you have to

Don't architect your system for scale until you need to scale.

Disadvantages of our architecture

- We have to run a **lot** of bridges
- We run Synapse on AWS which is relatively expensive
- Synapse becomes a bottleneck for all traffic

We've reached the extent of how far we can scale with our current architecture.

Don't scale until you have to

Don't architect your system for scale until you need to scale.

Stats

Because we encourage users to connect every chat network, we have a lot of puppet users.

On average, each user brings 4716 puppeted users.

All of these puppets represent real people on other networks, so they each generate their own traffic.

On average, each user and their puppets collectively account for 100k events!

Stats

Because we encourage users to connect every chat network, we have a lot of puppet users.

On average, each user brings 4716 puppeted users.

All of these puppets represent real people on other networks, so they each generate their own traffic.

On average, each user and their puppets collectively account for 100k events!

We need to find a way to scale!

Observation

Bridged rooms are unique

Most (chat) rooms in Matrix are *federated* which means that users on different servers can participate.

Bridged rooms are only for a **single user** on Beeper.com.

Observation

Bridged rooms are unique

Most (chat) rooms in Matrix are *federated* which means that users on different servers can participate.

Bridged rooms are only for a **single user** on Beeper.com.

Observation

Bridged rooms are unique

Most (chat) rooms in Matrix are *federated* which means that users on different servers can participate.

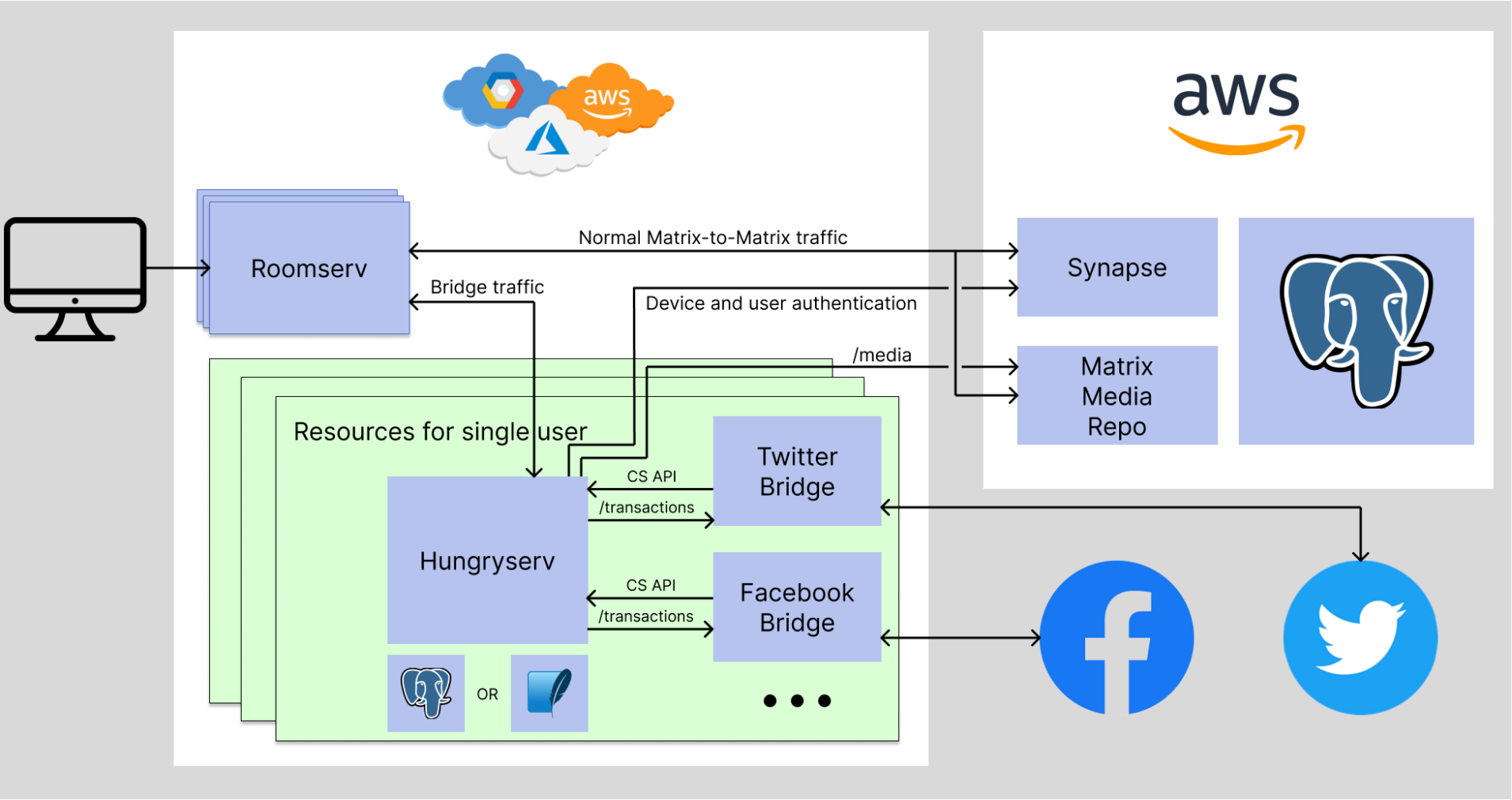
Bridged rooms are only for a **single user** on Beeper.com.

Finding scalability

Shard and load-balance our homeserver based on bridged vs. unbridged traffic

Our new architecture

A sharded future



Let's talk about the differences from the old architecture...

Splitting the homeserver, keeping the protocol

This new architecture allows us to handle non-bridge traffic separately from our bridge traffic.

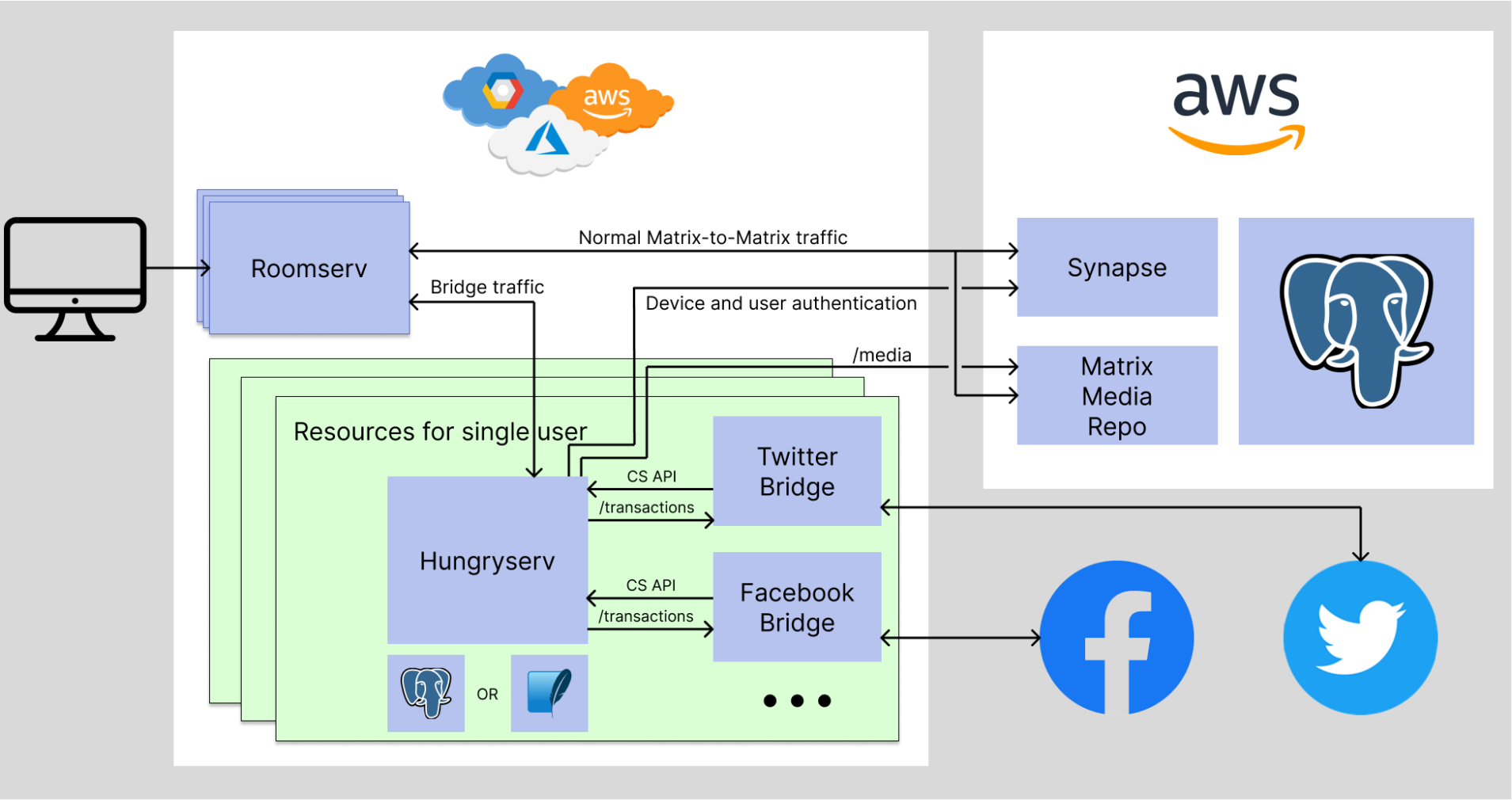
All while maintaining API compatibility with clients, bridges, and the Matrix federation!

Splitting the homeserver, keeping the protocol

This new architecture allows us to handle non-bridge traffic separately from our bridge traffic.

All while maintaining API compatibility with clients, bridges, and the Matrix federation!

A sharded future



Let's talk about the differences from the old architecture...

Hungryserv

We run a **Hungryserv** instance for every user to handle that user's bridge traffic. We can run infinite[2] Hungryserv instances for our users.

- If there's a message flood from a bridge, it only affects that user.
- We don't have to store all of the bridge events on Hungryserv's Synapse database.
- Synapse is less resource constrained and can do what it does best: federate with other Matrix homeservers.

[2] theoretically

Hungryserv

We run a **Hungryserv** instance for every user to handle that user's bridge traffic. We can run infinite[2] Hungryserv instances for our users.

- If there's a message flood from a bridge, it only affects that user.
- We don't have to store all of the bridge events and users in our Synapse database.
- Synapse is less resource constrained and can do what it does best: federate with other Matrix homeservers.

[2] theoretically

Hungryserv

We run a **Hungryserv** instance for every user to handle that user's bridge traffic. We can run infinite[2] Hungryserv instances for our users.

- If there's a message flood from a bridge, it only affects that user.
- We don't have to store all of the bridge events and users in our Synapse database.
- Synapse is less resource constrained and can do what it does best: federate with other Matrix homeservers.

[2] theoretically

Hungryserv

We run a **Hungryserv** instance for every user to handle that user's bridge traffic. We can run infinite[2] Hungryserv instances for our users.

- If there's a message flood from a bridge, it only affects that user.
- We don't have to store all of the bridge events and users in our Synapse database.
- Synapse is less resource constrained and can do what it does best: federate with other Matrix homeservers.

[2] theoretically

Hungryserv

We run a **Hungryserv** instance for every user to handle that user's bridge traffic. We can run infinite[2] Hungryserv instances for our users.

- If there's a message flood from a bridge, it only affects that user.
- We don't have to store all of the bridge events and users in our Synapse database.
- Synapse is less resource constrained and can do what it does best: federate with other Matrix homeservers.

[2] theoretically

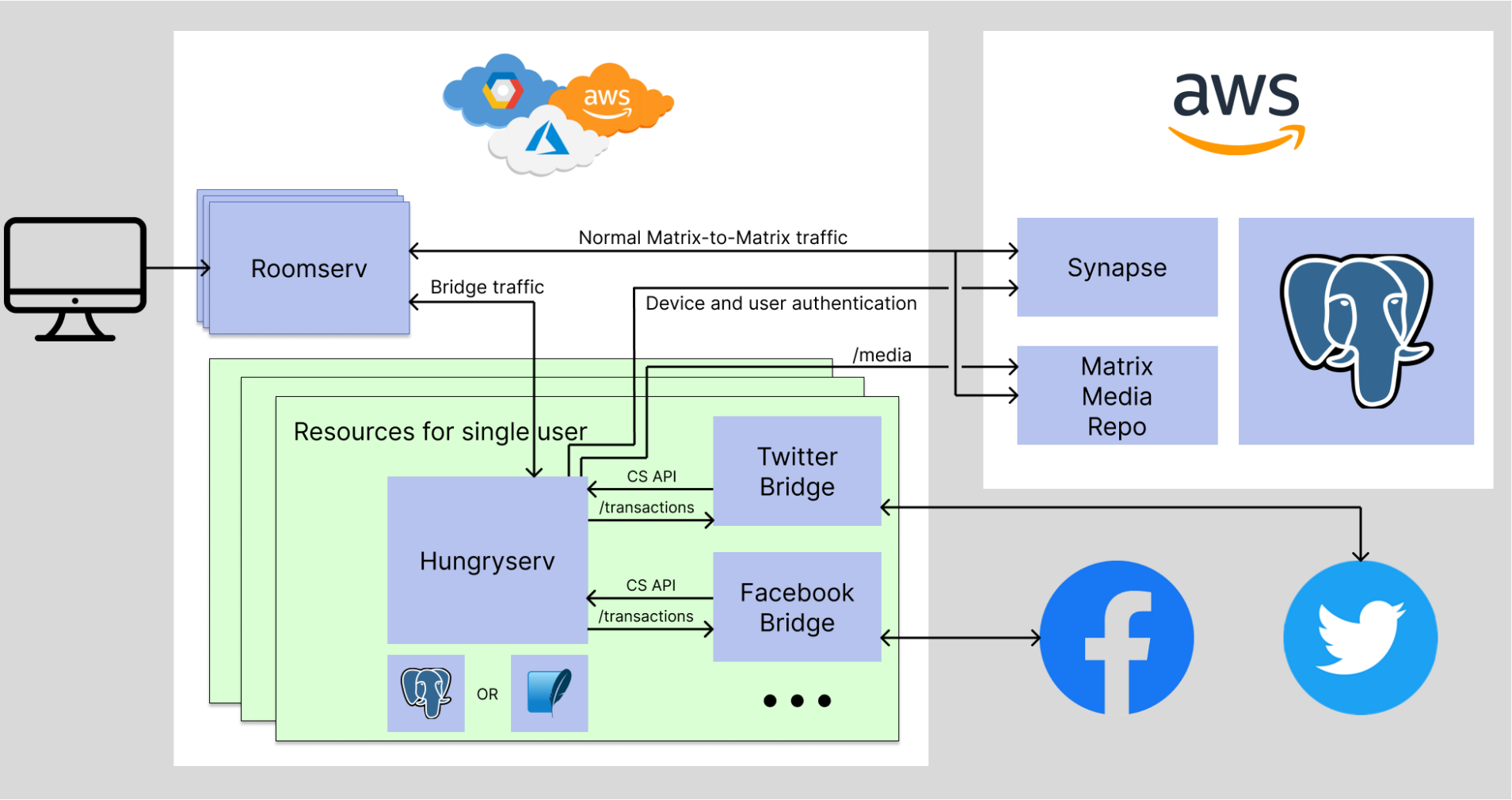
Hungryserv: why is it hungry?

Because it’s unfederated!

Hungryserv: why is it hungry?

Because it’s unfederated!

A sharded future



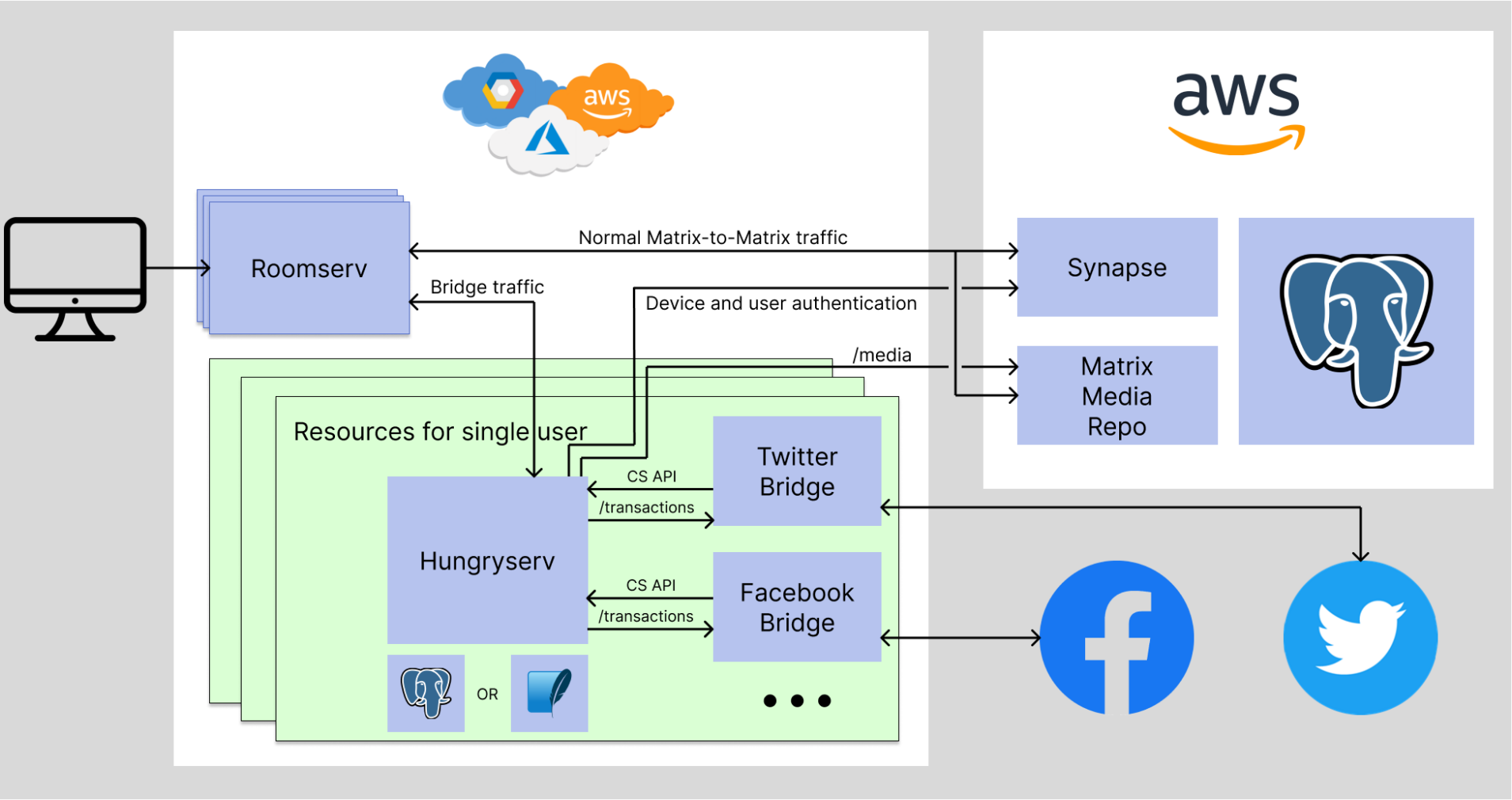
Let's talk about the differences from the old architecture...

Roomserv

Roomserv is a service which splits client requests between Synapse and Hungryserv.

Roomserv merges the responses from the two homeservers to make it appear as a single homeserver to the client.

A sharded future



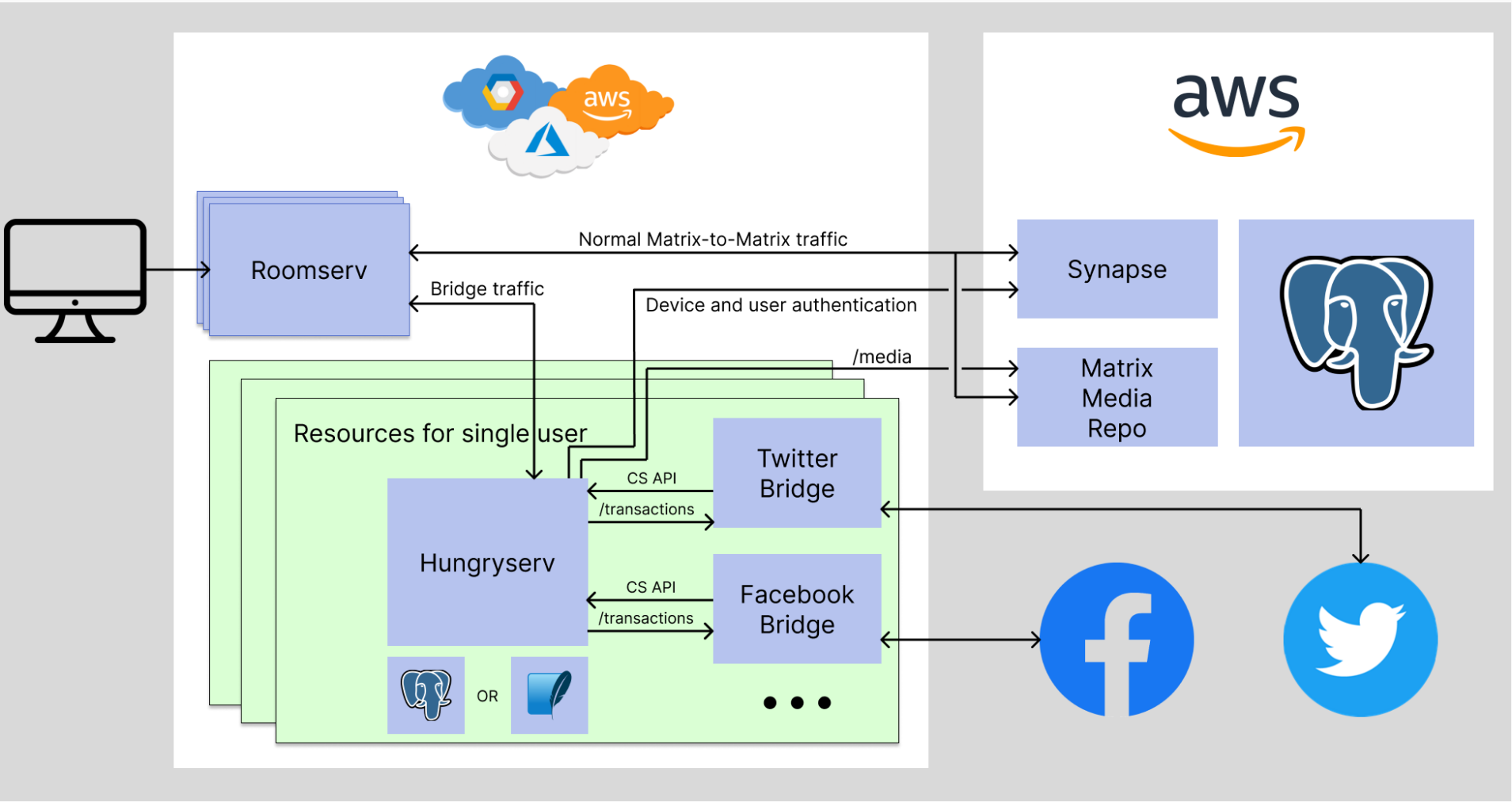
Let's talk about the differences from the old architecture...

A multi-cloud future?

The resources for each user can be moved closer to them using a multi-cloud, or multi-region strategy.

This will help reduce latency (especially to bridged networks) and allow for us to deploy in more flexible and cost-effective environments.

A sharded future



Let's talk about the differences from the old architecture...

Questions?