ChitChat - Project Proposal

Prepared for: Ethos, IIT Guwahati 2024

Prepared by: Team LinuxLabs

26 September 2024

Team Member: Prasad, Rakshit, Sanjana

EXECUTIVE SUMMARY

Objective

The goal is to develop a scalable and secure architecture for a messaging platform that can handle growing user demands while ensuring efficiency and privacy.

Goals

1. **Scalability**: Enable the system to automatically adapt to traffic and usage fluctuations.
2. **Security**: Implement robust user management and data protection mechanisms.
3. **Interoperability**: Ensure seamless integration with various communication platforms.
4. **Efficiency**: Optimise system performance for faster response times and resource management.
5. **Reliability**: Maintain consistent and secure communication across the platform.

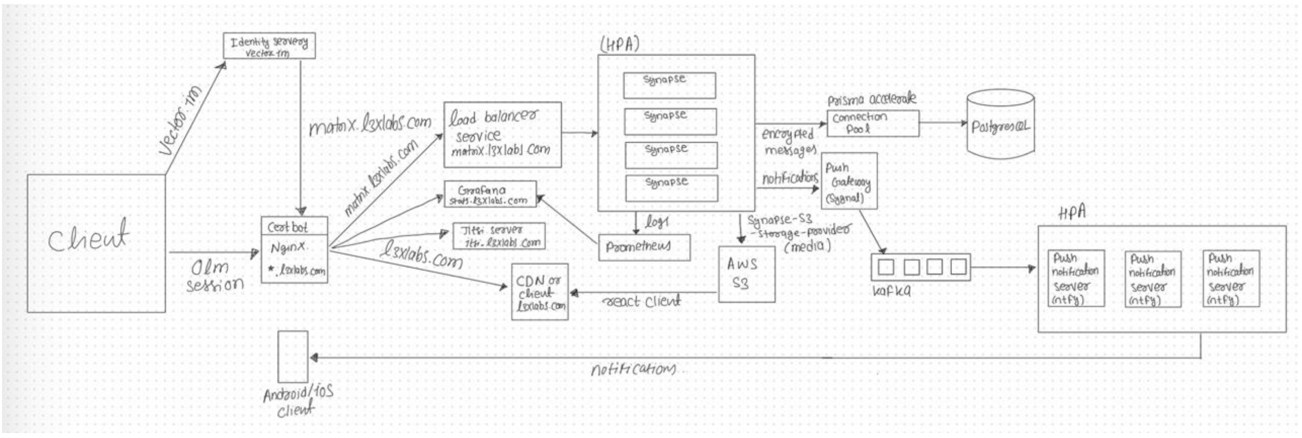
Solution

The proposed architecture focuses on scalability, secure communication, data management, and efficient backend processes, ensuring seamless functionality for both messaging and communication services.

OVERVIEW OF ARCHITECTURE

Components

1. **Synapse Server**: Handles instant messaging, media transfer, and cross-server communication.
2. **Client**: Interfaces with users across different platforms (Web, Android, iOS).
3. **Identity Server**: Manages mappings of third-party identifiers (3PIDs) to Matrix user identifiers.
4. **Jitsi Server**: Facilitates group VOIP communications.



IDENTITY SERVER FUNCTIONS

The identity server plays a crucial role in user registration and validation:

1. **Mapping 3PIDs**: It creates and validates mappings of third-party identifiers (such as email and phone numbers) to Matrix user identifiers. Matrix IDs can be retrieved using 3PIDs, but not vice versa, ensuring privacy.
2. **Inviting People**: Users can invite others using their 3PIDs through individual third-party lookups.
3. **Checking Contacts**: Allows users to check how many existing contacts have Matrix IDs using bulk identifier lookups.

Registration Security Features

- By default, account creation on the Synapse server is blocked to prevent multiple accounts.
- The identity server ensures that the user owns the 3PID before creating a Matrix account.

Validation Mechanism

- Validation typically involves emails or phone numbers, using OTP and short-lived verification tokens stored in the database to manage OTP expiration.

Federation of Identity Servers

- Two identity servers (vector.im and matrix.org) operate in a closed federation, where data created on one server is replicated on the other.
- They also provide MFA (Multi Factor Authentication).

SYNAPSE SERVER FUNCTIONS

The Synapse server is the backbone of the messaging platform:

- It handles instant messaging, media transfer, and allows cross-server communication using Matrix IDs.
- Its federated architecture ensures no single authority owns user data, distinguishing it from centralized messaging services like WhatsApp and Telegram.
- Our Synapse server is hosted on `matrix.l3xlabs.com`, while the client resides on `l3xlabs.com`.

Federation and Special Endpoints

- To solve federation issues, two special endpoints are used:

1. `.well-known/matrix/server` - Returns:

```
{
  "m.server": "matrix.l3xlabs.com:443"
}
```

2. `.well-known/matrix/client` - Returns:

```
{
    "m.homeserver":{
        "base_url": "https://matrix.l3xlabs.com"
    }
}
```

- These endpoints help other clients and servers in the federation reach the Synapse server at `matrix.l3xlabs.com`.

- We can disable federation by using domain whitelisting, but we are not doing that to avoid becoming the sole owner of user data.
- Our Synapse server stores data in a PostgreSQL database. All messages are encrypted using the Olm library with Matrix's session keys.

VOIP

- For 1-on-1 VOIP, we use Synapse TURN Server along with an eturnal TURN server configuration and TURN REST API. All communication is encrypted using the Olm library.
- For group VOIP, we use a self-hosted Jitsi server with DTLS enabled.

CLIENT

- For Desktop/Web, we use `matrix-js-sdk` along with `matrix-react-sdk` and the Olm library.
- For Android and iOS, we use the respective SDKs provided by Matrix, along with the Olm library.
- Each login is treated as a new device with its own set of keys.

Device Management

- 1. The device management system is robust. A verified device can see all other devices that logged in with credentials but without passphrase and can remove them.
- 2. Unverified sessions cannot access message history unless cross-verified using a passphrase or QR code.

MEDIA STORAGE IN SYNAPSE

- By default, Synapse stores media in a local folder, which is not scalable.
- The alternative is to use `Synapse-S3-Storage`, which stores media in AWS S3.

PUSH NOTIFICATIONS

For push notifications on mobile devices, we use Sygnal as the push gateway and NTFY for notifications.

MONITORING

We use Prometheus along with Grafana for logging and monitoring.

FEATURES

- E2EE (End-to-End Encryption) for one-on-one and group chats.
- Encrypted VOIP for 1-on-1 and group calls.
- MFA, OIDC, SSO, and multi-device support.
- Device management and key setup using passphrase or QR code.
- Multi-platform support (Web, Android, iOS).

Additional Features

- One client (l3xlabs.com) connects almost every communication service (e.g., Discord, WhatsApp, Telegram, Instagram, Slack, LinkedIn, Signal, SMS, email, Twitter, etc.).
- Instant messaging support from the single client is done using bridges for these services. Email is handled using Exim, the default SMTP for Synapse.

SCALING INFRASTRUCTURE

Client Serving

Since the web client is created using React, all build files are uploaded to AWS S3 and distributed using CloudFront (CDN).

Horizontal Pod Autoscaling (HPA)

Horizontal Pod Autoscaling (HPA) is designed to automatically scale the number of pods in a Kubernetes cluster based on the current workload. This ensures optimal resource utilization and responsiveness to varying traffic loads. HPA can dynamically adjust the number of running pods to maintain performance levels and handle peak loads effectively.

Database Architecture

- The master-slave architecture for the database can significantly improve performance by implementing proper indexing across all slave databases. This setup allows for both faster reads and writes:

4. **Master Database**: Responsible for all write operations and data modifications. It serves as the authoritative source of truth.
5. **Slave Databases**: These replicate the master database and handle read operations. By distributing the read load across multiple slaves, we reduce the latency and bottleneck typically associated with a single database.

- **Indexing Strategy**

To enhance the performance of the master-slave architecture, a comprehensive indexing strategy should be applied:

- **Read Optimization**: Indexing the slave databases can drastically speed up read queries, allowing for faster data retrieval and reduced response times.
- **Write Performance**: Although writes are handled by the master, ensuring that the master is well-indexed can also improve the speed at which changes are propagated to the slaves.

Backend Communication and Caching

Redis Implementation

Redis can be leveraged for asynchronous backend communication and caching:

- **Asynchronous Communication**: Using Redis Pub/Sub or Redis Streams can facilitate efficient communication between different microservices without blocking operations, enhancing overall responsiveness.
- **Caching Layer**: By implementing Redis as a caching layer, frequently accessed data can be stored in memory, reducing the load on the database and significantly decreasing response times for read operations.

Additional Features

Hosted Bridges

To enhance communication across different platforms and services, hosted bridges can be integrated:

1. **Interoperability**: Hosted bridges allow for seamless communication between various messaging services, enabling users to interact across platforms like Discord, WhatsApp, and others.
2. **Custom Features**: These bridges can be customized to support specific use cases and features, improving the overall user experience.

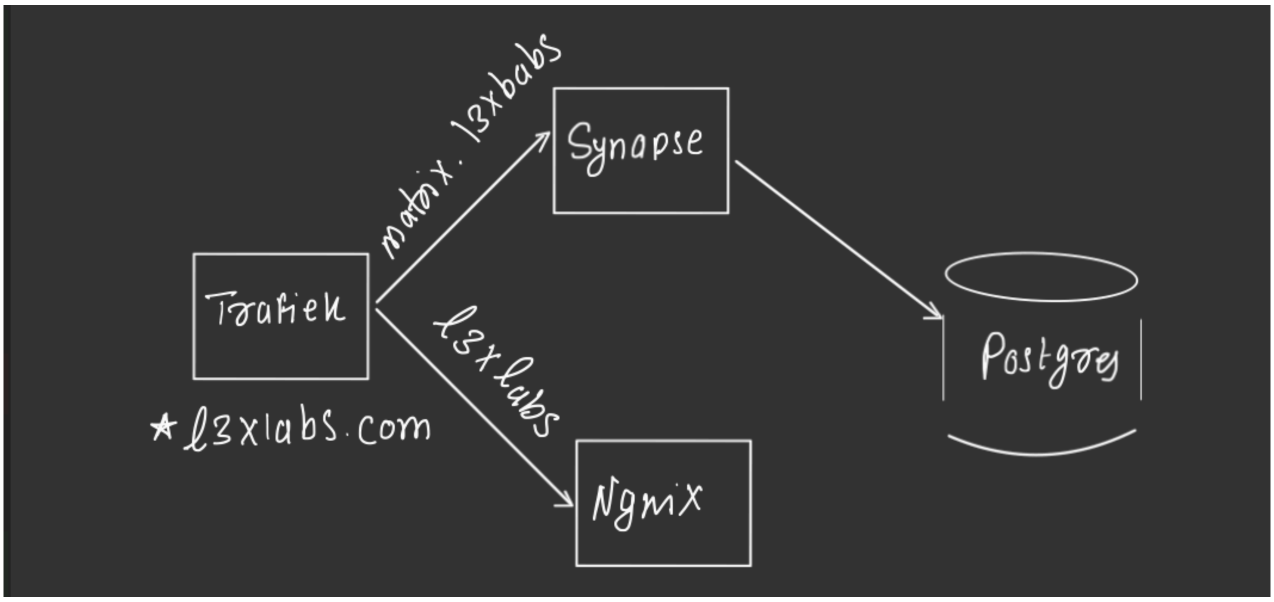
Email Communication

For email functionalities, we utilize the Exim SMTP server:

1. **Email Delivery**: Exim provides a robust mail transfer agent for sending and receiving emails reliably within the messaging architecture.
2. **Integration with Messaging Services**: Exim can be configured to handle notifications and alerts, bridging the gap between email communication and real-time messaging.

PROOF OF CONCEPT

Architecture



- **Nginx**: Used for serving the React client along with two special endpoints:

1. `./well-known/matrix/client`

2. `./well-known/matrix/server`

- **Synapse Server**: Handles all messaging, media transfer, etc.
- **Traefik**: Acts as a reverse proxy for Synapse and Nginx and manages SSL/TLS certificates

Features Already implemented

1. **E2EE Federated/Decentralized Instant Messaging**: End-to-end encrypted group messaging and direct messaging (DM).
2. **E2EE Media Transfer**: Includes encrypted voice notes, location sharing, poll creation (in groups), and more.
3. **Exporting Chats**: Individual and group chats can be exported in various formats.
4. **Key Setup and Sync**: Cross-device synchronization of keys on web, Android, and iOS using a passphrase.
5. **Device Management**: Displays a verified and unverified device list for better control over user sessions.

Test Credentials(Proof Of Concept)

Deployed at: https://l3xlabs.com
GitHub: https://github.com/L3xLabs/ChitChat

- User1:- `moonpie`, pass1:- `l3xlabs@2024`
- User2:- `honeybun`, pass:- `l3xlabs@2024`

Browser Recommendation: Google Chrome on Desktop **YouTube Demo Link**: https://youtu.be/D8bRmr8u9yM