Capítulo 1

Protocolo Matrix: Conceitos, Arquitetura, Aplicações e Desafios

José A. P. Martins, Paulo A. L. Rego, José A. F. de Macêdo, Rossana M. C. Andrade, Michel S. Bonfim, Roberto F. Ivo, Vinícius L. R. da Costa, Rodrigo P. Pacheco e Francisco A. P. da Silva

Abstract

This chapter presents a technical and applied analysis of the Matrix protocol, an open solution for secure and federated digital communication. It begins by discussing the context of digital sovereignty and the limitations of centralized platforms. The architecture of the protocol, its key features, and practical applications using the Element client are then explored. The chapter also covers the deployment of private Matrix servers with Synapse, automation via APIs, and the development of custom bots. Finally, it outlines current technical challenges and research opportunities related to interoperability, performance, and distributed security. The content integrates theoretical foundations with hands-on guidance, providing a comprehensive basis for institutional adoption of Matrix.

Resumo

Este capítulo apresenta uma análise técnico-aplicada do protocolo Matrix, uma solução aberta para comunicação segura e federada. Inicialmente, discute-se o contexto da soberania digital e os limites de plataformas centralizadas. Em seguida, são exploradas a arquitetura do protocolo, suas principais funcionalidades e aplicações práticas por meio do cliente Element. O texto também aborda a implantação de servidores privados com Synapse, automações via APIs e desenvolvimento de bots. Por fim, são discutidos desafios técnicos e oportunidades de pesquisa, com foco em interoperabilidade, desempenho e segurança distribuída. O conteúdo combina fundamentos teóricos com orientações práticas, oferecendo uma base sólida para adoção institucional do Matrix.

1.1. Introdução

Nos últimos anos, o debate sobre soberania digital tem ganhado relevância em diversas esferas, impulsionado por preocupações com privacidade, dependência tecnológica e concentração de poder em plataformas digitais. A comunicação, como componente central

das infraestruturas digitais, passou a depender majoritariamente de sistemas fechados, centralizados e de difícil auditoria [Floridi 2020]. Essa configuração favorece assimetrias de controle, limita a autonomia de organizações públicas e privadas, e dificulta o alinhamento tecnológico a valores locais e institucionais.

Diante disso, surgem como uma das alternativas os sistemas federados e o uso de *open-source software* (OSS) como pilares técnicos e estratégicos. Essas abordagens promovem a descentralização do controle, aumentam a auditabilidade das infraestruturas e ampliam a capacidade de adaptação institucional. A soberania digital, nesse contexto, é compreendida como a capacidade de indivíduos, organizações e Estados de exercerem controle sobre dados e sobre os sistemas que os armazenam e transportam [Klare and Lechner 2023, Moerel and Timmers 2021].

Apesar do avanço dessas discussões, persiste uma dependência estrutural de soluções proprietárias na comunicação digital cotidiana. Ferramentas como WhatsApp, Microsoft Teams ou Telegram não oferecem suporte à hospedagem própria, nem garantias de interoperabilidade[1] ou de segurança auditável. Esse cenário impõe obstáculos à adoção de políticas públicas baseadas em independência tecnológica e proteção de dados sensíveis.

Como resposta a esse problema, o presente capítulo examina o protocolo *Matrix* como um caminho viável e estratégico para comunicação digital federada, segura e auditável. O Matrix é um protocolo aberto para comunicação em tempo real, que adota uma arquitetura federada e descentralizada, permitindo que múltiplas instâncias se comuniquem entre si sem depender de servidores centralizados. [Foundation 2025].

Este capítulo propõe uma abordagem técnico-analítica sobre o protocolo, articulando seus fundamentos arquiteturais, aplicações práticas e potenciais para fortalecer a soberania digital em contextos institucionais e governamentais. O objetivo é apresentar o Matrix não apenas como ferramenta tecnológica, mas como recurso estratégico na formulação de soluções autônomas e auditáveis de comunicação.

O capítulo está estruturado em sete seções. A **Seção 1.1** introduz o tema, expõe o problema e apresenta o escopo do minicurso. As seções seguintes abordam a arquitetura do protocolo (**Seção 1.2**), aplicações práticas (**Seção 1.3**), aspectos avançados em servidores (**Seção 1.4**), uso das APIs para automação (**Seção 1.5**), desafios atuais e frentes de pesquisa (**Seção 1.6**). Finalmente, apresentamos as conclusões na **Seção 1.7**.

1.1.1. Fundamentos e Conceitos Principais

A compreensão do protocolo *Matrix* exige a familiarização com alguns fundamentos que estruturam a comunicação digital descentralizada e o desenvolvimento de software de código aberto. Esses elementos ajudam a situar o Matrix em um ecossistema mais amplo de soluções técnicas orientadas à transparência, autonomia e interoperabilidade.

A comunicação digital, desde seus primórdios, oscilou entre modelos centralizados e descentralizados. Nas arquiteturas centralizadas, amplamente adotadas por plataformas comerciais, a mediação das trocas informacionais depende de um servidor único que controla autenticação, roteamento, armazenamento e acesso. Esse modelo favorece

[1] Capacidade de sistemas distintos se comunicarem e funcionarem em conjunto de forma transparente e eficiente, sem necessidade de adaptações específicas.

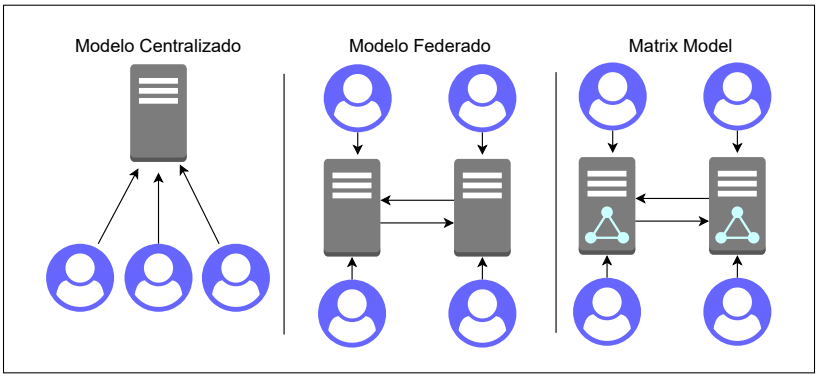

Figura 1.1. Comparação entre modelos de comunicação digital.

desempenho e controle operacional, mas concentra poder e limita a auditabilidade. Em contraste, sistemas descentralizados distribuem essas funções entre múltiplas entidades ou nós, mitigando pontos únicos de falha e ampliando a resiliência e o controle sobre os dados [Moerel and Timmers 2021, Floridi 2020].

Entre os modelos descentralizados, destacam-se as redes federadas, nas quais servidores distintos podem operar de forma autônoma e ainda assim interagir entre si com base em padrões abertos.[2] Esse modelo é amplamente adotado em sistemas como o e-mail e, mais recentemente, em protocolos como o ActivityPub, que sustenta redes sociais distribuídas como o Mastodon. A federação permite que diferentes organizações mantenham sua própria infraestrutura, com regras locais de controle e privacidade, sem abdicar da conectividade entre domínios distintos.

É nesse espectro que se insere o protocolo *Matrix*, projetado para viabilizar comunicação em tempo real entre múltiplas instâncias federadas, com foco em mensagens, chamadas e troca de arquivos. O Matrix busca aliar a flexibilidade de um modelo federado com o rigor técnico necessário para comunicação segura e verificável. A proposta original do protocolo surgiu a partir da constatação de que, embora existissem diversas soluções de mensageria, poucas ofereciam um padrão interoperável, aberto e extensível capaz de sustentar comunicação em tempo real entre domínios distintos [Li et al. 2023, Foundation 2025].

Um dos pilares do projeto é o compromisso com a transparência, refletido na publicação aberta de suas especificações e no desenvolvimento colaborativo de suas implementações. A abertura do código-fonte, aliada a práticas de auditoria e revisão comunitária, permite identificar falhas, adaptar componentes e estender funcionalidades. Em ambientes institucionais, essa característica torna-se particularmente relevante ao permitir que políticas de segurança e privacidade sejam adaptadas às necessidades locais [Bechara and Lechner 2024, Biström et al. 2024].

Ao articular descentralização, federação e software livre, o protocolo Matrix oferece um exemplo consistente de como infraestruturas de comunicação podem ser projeta-

[2]Federação é o modelo em que múltiplas instâncias de um sistema operam de forma independente, mas interoperam entre si segundo protocolos compartilhados. Exemplos incluem o e-mail, o XMPP e o ActivityPub.

das com base em princípios de soberania digital, auditabilidade e controle distribuído.

1.1.2. Características Essenciais do Protocolo

Diante da crescente preocupação com a privacidade, segurança e autonomia digital, torna-se fundamental analisar as características das plataformas de mensageria utilizadas por instituições públicas e privadas. A escolha da tecnologia impacta diretamente o controle sobre os dados e a infraestrutura de comunicação. Nesse cenário, critérios como abertura, criptografia e interoperabilidade ganham relevância estratégica. A Tabela 1.1 apresenta um comparativo entre plataformas populares de mensageria, destacando critérios relevantes no contexto da soberania digital: abertura do código-fonte, suporte à criptografia ponta-a-ponta (E2EE), interoperabilidade e possibilidade de hospedagem própria. Esses aspectos são particularmente sensíveis em ambientes institucionais que buscam controle sobre infraestrutura e dados. Observa-se que o Matrix se diferencia por combinar todos esses atributos em uma única solução, enquanto outras plataformas, como WhatsApp e Slack, mantêm modelos centralizados e de código fechado. O Telegram oferece abertura parcial, mas sem suporte nativo à hospedagem autônoma ou interoperabilidade plena.

Tabela 1.1. Comparativo entre plataformas de mensageria

Plataforma	Código Aberto	E2EE	Interoperável	Hospedagem Própria
Matrix (Element)	Sim	Sim	Sim (via bridges)	Sim
WhatsApp	Não	Sim	Não	Não
Telegram	Parcial	Opcional	Limitada	Não
Signal	Sim	Sim	Não	Sim
Slack (padrão)	Não	Não	Limitada (via API)	Não

O protocolo *Matrix* apresenta um conjunto de características técnicas que o distinguem de outras soluções de comunicação digital. Essas propriedades resultam de escolhas arquiteturais voltadas à interoperabilidade, controle descentralizado, segurança por padrão e extensibilidade. Ainda que a estrutura interna do protocolo seja explorada na seção seguinte, algumas de suas funcionalidades podem ser compreendidas a partir de seu comportamento observável e das possibilidades que oferece a usuários e desenvolvedores.

A interoperabilidade é um dos pilares do Matrix. O protocolo permite a construção de *bridges*, que funcionam como conectores entre o ecossistema Matrix e outras plataformas de comunicação, como Slack, Discord, IRC ou Microsoft Teams. Essas pontes facilitam a transição entre sistemas legados e ambientes federados, promovendo inclusão e coexistência entre diferentes tecnologias [Cui et al. 2023, Foundation 2025].

Outra característica distintiva é a possibilidade de controle efetivo sobre dados e infraestrutura. Ao permitir a hospedagem autônoma de servidores, o Matrix viabiliza que organizações e comunidades mantenham sua comunicação sob políticas locais de segurança, retenção e governança. Essa capacidade se alinha aos princípios da soberania digital e da privacidade institucional, frequentemente negligenciados em soluções centralizadas [Bechara and Lechner 2024, Moerel and Timmers 2021].

A segurança é incorporada ao protocolo desde sua concepção. A criptografia ponta-a-ponta (E2EE) é suportada por padrão em salas privadas, com mecanismos adi-

cionais como *cross-signing* de dispositivos e *key backup* criptografado, permitindo autenticação mútua e recuperação segura de chaves. Esses recursos ampliam a proteção sem comprometer a usabilidade, e posicionam o Matrix como uma alternativa robusta para contextos sensíveis [Albrecht et al. 2023, Authority 2024].

A resiliência do protocolo também é notável. Por operar em um modelo federado e assíncrono, o sistema consegue manter a continuidade da comunicação mesmo em casos de falha ou indisponibilidade temporária de servidores individuais. Essa propriedade é essencial para aplicações que exigem tolerância a falhas e continuidade operacional, como ambientes críticos e redes comunitárias distribuídas.

Finalmente, o Matrix adota especificações públicas e APIs bem documentadas, facilitando sua integração com outras ferramentas e o desenvolvimento de aplicações sob medida. Essa extensibilidade tem viabilizado o surgimento de clientes alternativos, bots, sistemas de autenticação customizados e integrações com sistemas corporativos e educacionais [matrix nio 2025, Rahman and Wang 2024].

Ao combinar interoperabilidade, controle local, segurança nativa e abertura à inovação, o protocolo Matrix apresenta um conjunto coeso de funcionalidades que o tornam adequado a contextos onde a autonomia tecnológica é prioridade.

1.1.3. Soberania Digital e Privacidade

A soberania digital refere-se à capacidade de Estados, organizações e indivíduos de exercerem controle efetivo sobre seus dados, sistemas e infraestruturas tecnológicas. Essa noção ganha relevância diante da crescente interdependência entre decisões políticas e arquiteturas técnicas, especialmente em áreas como segurança nacional, privacidade e autonomia tecnológica [Floridi 2020, Klare and Lechner 2023]. A centralização de serviços digitais sob o controle de grandes empresas transnacionais limita a autodeterminação dos atores locais e expõe vulnerabilidades críticas associadas à vigilância, manipulação de dados e dependência estratégica.

Ainda, a soberania digital deve ser entendida não apenas como uma questão estratégica, mas como um imperativo ético e jurídico em defesa da privacidade, da democracia e dos direitos digitais do cidadão. Além da segurança nacional, a ausência da soberania digital compromete direitos fundamentais, especialmente em contextos de conflitos externos, nos quais nações estrangeiras possam interferir diretamente nos serviços oferecidos. Portanto, torna-se imperativo que Estados e instituições públicas incorporem não apenas soluções técnicas descentralizadas, mas também políticas robustas de fomento à pesquisa, desenvolvimento local e cooperação internacional voltada à construção de uma infraestrutura digital verdadeiramente soberana e democrática.

Nesse sentido, o uso de software livre e arquiteturas federadas surge como base tecnológica para a construção de alternativas soberanas. O *open-source software* (OSS) viabiliza auditoria, adaptação e apropriação institucional dos sistemas, enquanto redes federadas permitem a descentralização operacional, mantendo a interoperabilidade entre entidades autônomas. Essa combinação amplia a capacidade de instituições públicas e comunitárias de construir soluções próprias sem renunciar à conectividade global [Moerel and Timmers 2021, Biström et al. 2024].

O protocolo *Matrix* ilustra esse modelo. Por sua natureza federada, permite que órgãos públicos e organizações mantenham controle sobre servidores, políticas de segurança e dados sensíveis. Um exemplo emblemático é o *Tchap*, sistema de mensageria utilizado pelo governo francês e desenvolvido sobre o Matrix. Essa iniciativa substituiu plataformas comerciais em comunicações oficiais, garantindo criptografia ponta a ponta e hospedagem soberana sob infraestrutura estatal [Government 2020, Bechara and Lechner 2024]. Outras experiências institucionais reforçam o papel estratégico de soluções abertas e federadas como ferramentas de independência tecnológica.

Apesar de seus avanços, a adoção de soluções soberanas enfrenta obstáculos relevantes. Barreiras técnicas, como a complexidade de implantação e manutenção de infraestruturas próprias, coexistem com desafios geopolíticos, incluindo a influência de padrões dominantes impostos por corporações e a dependência de componentes críticos produzidos por terceiros. Riscos como a vigilância extraterritorial e a presença de *backdoors* em sistemas proprietários mantêm a urgência de alternativas auditáveis e adaptáveis [Misra et al. 2025].

Ao integrar princípios técnicos de descentralização com requisitos políticos de autonomia e segurança, o Matrix se consolida como um dos caminhos possíveis para o fortalecimento da soberania digital em ambientes institucionais e governamentais.

1.1.4. Evolução do protocolo Matrix e eventos da comunidade

Desde sua introdução em 2014, o protocolo Matrix tem passado por uma evolução constante, marcada por melhorias estruturais, expansão da comunidade e consolidação de padrões formais por meio da Matrix Specification (*spec*)[3]. Essa trajetória técnica foi acompanhada pelo fortalecimento da governança aberta e do ecossistema de desenvolvedores e usuários ao redor do protocolo.

Tabela 1.2. Linha do tempo de desenvolvimento do protocolo Matrix

Ano	Evento / Marco
2014	Publicação inicial da spec Matrix 1.0 e criação do Synapse como implementação de referência.
2017	Introdução das primeiras MSCs (Matrix Spec Changes)[4] como processo formal de melhoria da especificação.
2019	Lançamento oficial da **Matrix Foundation**; Element (antigo Riot) se torna cliente principal.
2021	Introdução de *Spaces* (hierarquias de salas) e suporte nativo a *Threads*.
2022	Proposta de **Sliding Sync** e melhoria drástica na sincronização de clientes móveis.
2023	Lançamento do **Element Call** com chamadas de áudio/vídeo nativamente federadas via WebRTC.
2024	Consolidação da visão **Matrix 2.0**, com E2EE por padrão, sincronização rápida e governança aberta de spec.

A Tabela 1.2 apresenta uma síntese da linha do tempo com os marcos principais de desenvolvimento do protocolo. Além disso, evidencia não apenas a continuidade no

[3] `https://spec.matrix.org`

desenvolvimento técnico do protocolo, mas também a consolidação de uma cultura de padronização aberta e evolução incremental. Cada marco representa um avanço específico — desde a estruturação inicial da *spec*, passando pela formalização das MSCs, até a adoção de novos paradigmas de usabilidade e desempenho, como os introduzidos pelo Matrix 2.0. Essa trajetória demonstra o compromisso da comunidade em equilibrar inovação com interoperabilidade e retrocompatibilidade, aspectos essenciais para a adoção institucional de longo prazo.

O desenvolvimento e a disseminação do protocolo também são impulsionados por uma comunidade ativa, articulada em diversos eventos e canais de comunicação. A *Matrix Community Summit*[5], realizada anualmente na Alemanha, é o principal ponto de encontro técnico e institucional da comunidade. Nela, desenvolvedores, pesquisadores e representantes de instituições compartilham avanços, experimentos e casos de adoção. O canal semanal *Matrix Live*[6] oferece atualizações constantes sobre o ecossistema, incluindo entrevistas com autores de MSCs, novidades em clientes, bridges e projetos institucionais. Outros espaços relevantes incluem apresentações na *FOSDEM*[7] e no *Chaos Communication Congress (CCC)*[8], onde o protocolo tem sido debatido como alternativa aberta e soberana para mensageria segura.

O marco mais recente dessa trajetória é o **Matrix 2.0**. Trata-se de uma consolidação conceitual que incorpora avanços recentes como a sincronização ultrarrápida (*Sliding Sync*), a ativação da criptografia ponta-a-ponta (E2EE) por padrão, a integração de chamadas nativas via *Element Call*, e a reestruturação das funcionalidades de salas e espaços. Além de melhorias técnicas, o Matrix 2.0 representa um compromisso com a estabilidade, interoperabilidade e usabilidade, mantendo compatibilidade retroativa com versões anteriores. Esse momento é interpretado pela comunidade como a transição do protocolo de uma base experimental para uma solução madura, pronta para adoções institucionais e em escala.

1.2. Arquitetura e Algoritmos do Matrix

Nesta seção, descrevemos os principais componentes técnicos do protocolo Matrix. São abordadas suas estruturas arquitetônicas, mecanismos de sincronização distribuída, soluções de criptografia, interoperabilidade com outras plataformas e suas APIs fundamentais. O objetivo é oferecer uma visão sistematizada da infraestrutura que sustenta a comunicação federada, segura e extensível no contexto do Matrix.

1.2.1. Arquitetura Descentralizada e Federada

O protocolo Matrix adota uma arquitetura federada e descentralizada como resposta aos desafios impostos por sistemas de comunicação centralizados, que frequentemente sofrem com problemas de escalabilidade, pontos únicos de falha e limitações de controle sobre dados. Na arquitetura federada do Matrix, a responsabilidade pelo armazenamento, processamento e controle dos dados é distribuída entre múltiplos servidores in-

[5] https://matrix.org/events
[6] https://matrix.org/blog/tags/matrix-live
[7] https://fosdem.org
[8] https://events.ccc.de

dependentes, denominados homeservers[9]. Cada homeserver é operado de forma autônoma, mantendo a propriedade e o controle dos dados dos usuários que lhe são associados. Essa abordagem elimina a dependência de uma autoridade central, oferecendo maior resiliência, autonomia institucional e controle sobre os dados trafegados na rede [Li et al. 2023, Foundation 2025].

A arquitetura do protocolo Matrix é estruturada em torno de três componentes principais que interagem para viabilizar a comunicação descentralizada, interoperável e segura: os homeservers, os clientes e os app services.

Os homeservers constituem o núcleo da arquitetura. Eles são responsáveis por armazenar o estado dos usuários e das salas (*rooms*), gerenciar a autenticação e o processamento de eventos, além de coordenar a comunicação com outros homeservers dentro da federação. Cada homeserver mantém o histórico completo das conversas e o estado das salas associadas aos seus usuários, atuando como o guardião do domínio e das identidades que lhe pertencem. Essa autonomia permite que cada servidor opere de forma independente, ao mesmo tempo, em que participa da rede federada por meio de APIs RESTful, que asseguram a replicação do estado das salas e a propagação segura e confiável das mensagens entre diferentes domínios.

Em termos de infraestrutura, os homeservers são serviços backend que expõem interfaces HTTP e utilizam bancos de dados persistentes, geralmente relacionais (como PostgreSQL), para armazenar dados e garantir consistência eventual. Implementações populares de homeservers, como o Synapse (escrito em Python) [Matrix Foundation 2025b], Dendrite (em Go) [Matrix Foundation 2025a] e Conduit (em Rust) [Conduit 2025], exemplificam as abordagens técnicas para lidar com escalabilidade, persistência e federação. A Tabela 1.3 apresenta uma comparação das principais características dessas implementações.

Tabela 1.3. Comparação de Implementações de Homeserver Matrix

Implementação	Linguagem	Banco de Dados	Escalabilidade	Federação	Aberto
Synapse	Python	PostgreSQL	Moderada	Sim	Sim
Dendrite	Go	PostgreSQL/MySQL	Alta	Sim	Sim
Conduit	Rust	SQLite/PostgreSQL	Alta	Sim	Sim

Os clientes são as interfaces utilizadas pelos usuários finais para interagir com a rede Matrix. Eles incluem aplicativos para dispositivos móveis, desktops e navegadores web, tais como Element Web, Element X, FluffyChat e outros. Cada cliente está vinculado a um homeserver específico, com o qual interage através da Client-Server API para realizar operações essenciais como autenticação, envio de mensagens, gerenciamento de salas e sincronização de eventos. Essa comunicação assegura que o estado apresentado ao usuário esteja sempre atualizado e consistente com o servidor. Além disso, os clientes frequentemente implementam criptografia de ponta a ponta, utilizando os protocolos Olm e Megolm, fornecendo confidencialidade e integridade às comunicações. A Tabela 1.4 oferece uma comparação dos principais clientes Matrix, destacando plataformas suportadas, recursos de segurança e modelos de código aberto.

[9]Homeservers são instâncias autônomas do protocolo Matrix, responsáveis por gerenciar usuários, salas e eventos sob seu domínio.

Tabela 1.4. Comparação dos Principais Clientes Matrix

Cliente	Plataforma	E2EE	Aberto	Funcionalidades Notáveis
Element	Web, Desktop, Mobile	Sim	Sim	Interface rica, suporte a VoIP, E2EE
Element X	Mobile (iOS/Android)	Sim	Sim	Versão mais leve, design moderno
FluffyChat	Mobile (iOS/Android)	Sim	Sim	Interface simples, foco em usabilidade
Quaternion	Desktop (Linux/Win)	Sim	Sim	Cliente leve, interface Qt, ideal para desktop
Nheko	Desktop (Linux/Win/macOS)	Sim	Sim	Cliente desktop moderno, suporte a E2EE
Neo	Desktop (Windows)	Sim	Sim	Cliente Windows focado em usabilidade e recursos
SchildiChat	Mobile (Android)	Sim	Sim	Fork do Element focado em privacidade e performance
Mirage	Mobile (iOS)	Sim	Sim	Cliente iOS leve com foco em simplicidade

Paralelamente, a comunicação entre diferentes homeservers é realizada pela Federation API, que mantém a consistência do estado das salas e permite a interoperabilidade entre usuários pertencentes a domínios distintos. A federação é essencial para a arquitetura descentralizada do Matrix, permitindo que servidores independentes compartilhem eventos de forma segura e confiável.

Os *app services* representam componentes externos integrados ao ecossistema Matrix que expandem suas funcionalidades e capacidades. Esses serviços podem atuar como *bridges*, que são responsáveis por conectar a rede Matrix a outras plataformas de comunicação, como IRC, Slack, Discord e XMPP. *Bridges* realizam a tradução bidirecional de mensagens e eventos entre Matrix e os protocolos externos, permitindo a interoperabilidade entre diferentes ecossistemas de comunicação e ampliando o alcance da rede Matrix. Além das *bridges*, os *app services* englobam *bots* e outros agentes programáveis que automatizam tarefas, respondem a comandos, moderam salas ou fornecem funcionalidades adicionais. Esses *bots* podem interagir utilizando a API Client ou operar como *app services*, escutando eventos e respondendo conforme a lógica implementada, como é o caso do `mautrix-python`, uma biblioteca que facilita a criação de *bots* e *bridges* [Mautrix 2025]. Essa arquitetura aberta e extensível possibilita uma variedade de integrações e personalizações que enriquecem a experiência dos usuários e ampliam o potencial do protocolo.

A Figura 1.2 ilustra os principais componentes do ecossistema Matrix e suas interações fundamentais. O diagrama destaca a relação entre os **clientes Matrix** (1), os **homeservers** responsáveis pela mediação e armazenamento das comunicações (2), e os **servidores de identidade** (3), que auxiliam na descoberta e verificação de usuários. Também são representadas as **bridges** (4), que atuam como conectores entre o Matrix e outras plataformas de comunicação, possibilitando a interoperabilidade com **clientes externos** (5) como Slack e Skype.

Ao adotar esse modelo, o Matrix possibilita que organizações públicas, instituições acadêmicas, comunidades ou indivíduos mantenham suas próprias instâncias do pro-

tocolo, com regras locais de autenticação, retenção e segurança [10] A arquitetura federada torna possível que essas instâncias colaborem entre si sem abrir mão de suas políticas de governança e infraestrutura própria, alinhando-se aos princípios de soberania digital.

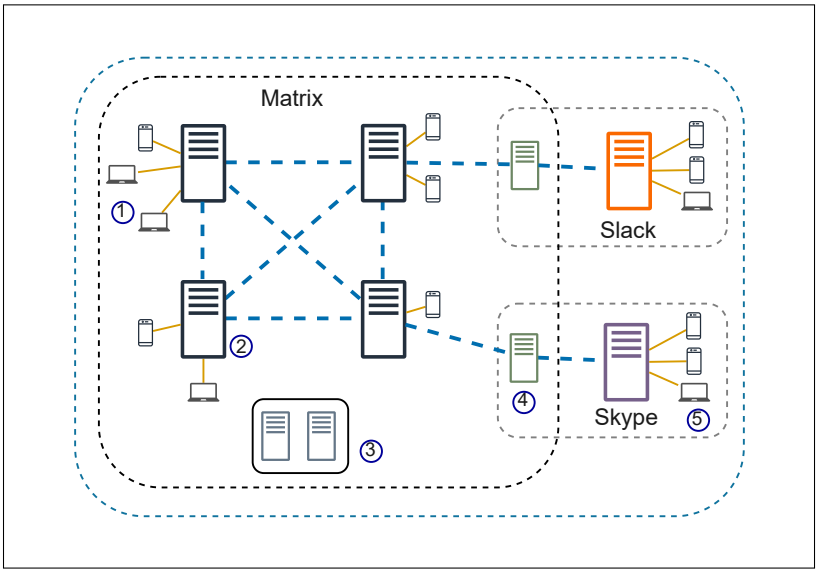

Figura 1.2. Diagrama da arquitetura geral do Matrix: clientes, homeservers e bridges.

1.2.2. Modelo de Dados e Eventos

No cerne da arquitetura do Matrix está o modelo de dados orientado a eventos, que fundamenta toda a comunicação e sincronização dentro do protocolo. Cada ação realizada no sistema — seja o envio de uma mensagem, uma alteração na configuração da sala, ou um evento de presença — é representada como um evento codificado em JSON. Esses eventos são a unidade atômica da comunicação, encapsulando tanto os dados da ação quanto metadados essenciais, como o identificador do remetente, o timestamp, e a relação com eventos anteriores.

O protocolo Matrix define uma variedade de tipos de eventos para suportar diferentes funcionalidades e tipos de mídia. Entre os tipos mais comuns estão:

- Mensagem de texto (`m.room.message`): Eventos que carregam mensagens em texto simples, sendo o tipo mais frequente nas conversas.
- Envio de arquivos (`m.room.message` com `msgtype` como `m.file` ou `m.image`): Eventos que transportam metadados sobre arquivos, imagens ou vídeos compartilhados, incluindo URLs para acesso e informações de tamanho.
- Áudio e Vídeo (`m.room.message` com `msgtype` como `m.audio` ou `m.video`): Eventos similares ao envio de arquivos, mas para conteúdo multimídia de áudio e vídeo, suportando mensagens de voz, clipes e chamadas.

[10] A capacidade de impor políticas locais torna o Matrix adequado a ambientes com requisitos normativos específicos, como órgãos públicos e instituições educacionais.

- Chamadas de voz e vídeo (`m.call.invite`, `m.call.answer`, `m.call.hangup`): Eventos específicos que coordenam a sinalização para iniciar, aceitar ou encerrar chamadas em tempo real, permitindo comunicação multimídia interativa entre usuários.
- Alteração do nome da sala (`m.room.name`): Eventos que definem ou atualizam o nome de uma sala.
- Atualização do tópico da sala (`m.room.topic`): Eventos responsáveis por definir ou modificar o tópico de uma sala, que normalmente serve como descrição ou assunto principal.
- Gerenciamento de membros (`m.room.member`): Eventos usados para adicionar, remover ou modificar permissões de usuários dentro de uma sala.
- Controle de permissões (`m.room.power_levels`): Eventos que definem os níveis de permissão dos membros, determinando quem pode enviar certos tipos de eventos ou alterar configurações da sala.
- Criação da sala (`m.room.create`): Evento inicial que sinaliza a criação da sala e contém informações sobre seu criador e propriedades básicas.
- Estado de presença (`m.presence`): Eventos que informam o estado de presença dos usuários, como online, offline ou ausente.
- Indicador de digitação (`m.typing`): Eventos temporários que sinalizam que um usuário está digitando em uma sala.
- Reações (`m.reaction`): Eventos que registram reações, como emojis, a mensagens existentes.
- Envio de figurinhas (`m.sticker`): Eventos que permitem o envio de figurinhas (stickers) nas conversas.
- Alteração do avatar da sala (`m.room.avatar`): Eventos que definem ou modificam o avatar associado a uma sala.

Os Listings 1.2 e 1.1 apresentam exemplos de eventos no protocolo Matrix. O Listing 1.1 ilustra a entrada do usuário Maria em uma sala, representada por um evento de alteração de membro, enquanto o Listing 1.2 mostra um evento de envio de mensagem de texto. Ambos os eventos compartilham campos essenciais que fundamentam a estrutura do protocolo: o campo `type` indica o tipo do evento, determinando sua função e interpretação; o campo `sender` identifica o usuário que originou o evento; `room_id` especifica a sala em que o evento ocorre; e `event_id` é um identificador único para cada evento. O campo `origin_server_ts` representa o timestamp do evento no servidor de origem, usado para ordenação e auditoria. Em eventos de estado, como o de membro, o campo `state_key` identifica o usuário ao qual o estado se aplica, e o conteúdo (`content`) detalha a ação ou dado associado, como o corpo da mensagem ou o tipo de participação na sala.

```
{
  "type": "m.room.member",
  "sender": "@maria:example.org",
  "state_key": "@maria:example.org",
  "room_id": "!reuniao12345:example.org",
  "event_id": "$entryevent12345:example.org",
  "origin_server_ts": 1685100000000,
  "content": {
    "membership": "join"
  }
}
```

Listing 1.1. Exemplo de evento representando a entrada do usuário @maria na sala "reuniao"do homeserver example.org

```
{
  "type": "m.room.message",
  "sender": "@maria:example.org",
  "room_id": "!reuniao12345:example.org",
  "event_id": "$sendevent56789:example.org",
  "origin_server_ts": 1685000010000,
  "content": {
    "msgtype": "m.text",
    "body": "Oi!"
  }
}
```

Listing 1.2. Exemplo simplificado de evento de envio de mensagem

1.2.3. Matrix Event Graph (MEG): Algoritmos e Sincronização Distribuída

O mecanismo central de sincronização e consistência no protocolo Matrix é o *Matrix Event Graph* (MEG), uma estrutura baseada em grafos direcionados acíclicos (DAG)[11]. Cada evento gerado por um cliente é representado como um nó no grafo, contendo referências a eventos anteriores. Essa estrutura permite a replicação assíncrona do histórico entre servidores, garantindo que múltiplas instâncias mantenham uma visão consistente dos dados mesmo em presença de desconexões temporárias [Jacob et al. 2021, Foundation 2025].

Cada homeserver mantém uma cópia parcial do MEG para as salas em que participa. Ao receber eventos de outros servidores, o homeserver executa algoritmos de fusão[12], validação de assinaturas, verificação de integridade e resolução de conflitos. A ordenação dos eventos se baseia em vetores causais, utilizando o conceito de `depth` e `prev_events` para garantir uma linearização compatível entre instâncias federadas. A

[11] DAGs (Directed Acyclic Graphs) são estruturas comuns em sistemas distribuídos para representar relações causais sem ciclos.

[12] No contexto do MEG, a fusão de grafos refere-se à incorporação de eventos recebidos de outras réplicas no DAG local, preservando relações de causalidade.

sincronização se dá de forma eventual[13], priorizando consistência sobre disponibilidade imediata.

A Figura 1.3 ilustra a evolução do MEG em duas réplicas distintas de um mesmo ambiente federado. Cada novo evento (vértice) é anexado às extremidades[14] do grafo local, e a sincronização entre réplicas é feita por meio da troca dessas atualizações.

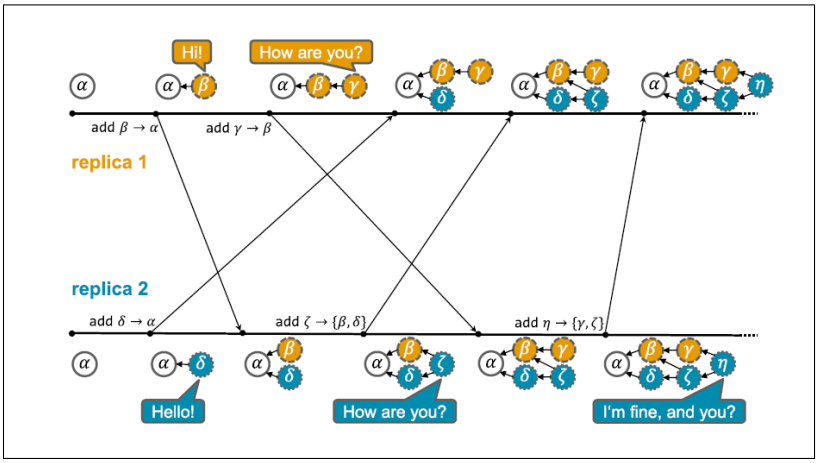

Figura 1.3. Evolução da estrutura MEG em duas réplicas com atualizações concorrentes [Jacob et al. 2021].

A figura mostra como eventos concorrentes são tratados: ao serem detectados por réplicas distintas, são integrados ao DAG por meio de vértices com múltiplos pais, respeitando a aciclicidade e permitindo fusões posteriores que convergem o estado. Esse comportamento caracteriza o MEG como uma estrutura do tipo CRDT[15] orientada a operações.

A Figura 1.4 complementa a visão mostrando a estrutura conceitual do MEG como um CRDT. São destacados os principais componentes: vértices, arestas, regras de geração e aplicação, e o papel das extremidades como pontos de entrada para novos eventos.

A Tabela 1.5 resume as principais características técnicas do Matrix Event Graph no contexto de replicação distribuída. Cada linha destaca uma propriedade fundamental do MEG, combinando aspectos estruturais e operacionais com implicações diretas para a escalabilidade, confiabilidade e consistência do protocolo.

Esse resumo reforça como o MEG opera como uma estrutura de dados robusta para ambientes federados, conciliando replicação assíncrona com convergência eventual. No entanto, também evidencia limitações, como o possível crescimento da largura do DAG e a necessidade de estratégias específicas de otimização para ambientes de alta concorrência.

[13] Consistência eventual é a propriedade de sistemas distribuídos em que, após algum tempo sem novas atualizações, todas as réplicas convergem para o mesmo estado.

[14] As extremidades de um DAG são os vértices que não possuem filhos — ou seja, que ainda não foram referenciados por nenhum evento subsequente.

[15] CRDTs são tipos de dados replicados projetados para funcionar corretamente em sistemas distribuídos, mesmo com atualizações concorrentes, sem necessidade de sincronização ou ordenação global.

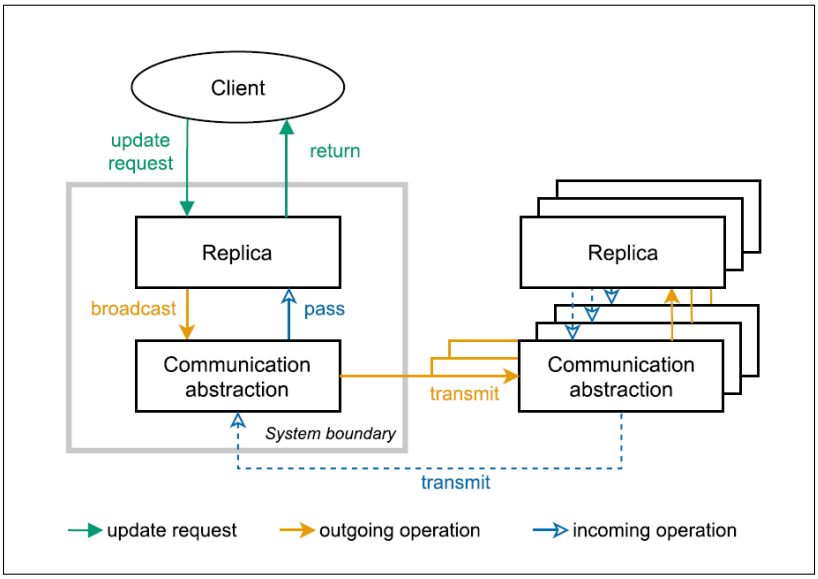

Figura 1.4. Esquema da estrutura do MEG como CRDT orientado a operações [Jacob et al. 2021].

Tabela 1.5. Propriedades do Matrix Event Graph (MEG) como estrutura de replicação distribuída

Propriedade	**Descrição**
Modelo estrutural	DAG (grafo acíclico direcionado), com nós representando eventos e arestas representando causalidade.
Replicação	Assíncrona e eventual, com trocas parciais de estado entre homeservers federados.
Fusão de eventos	Baseada em múltiplos pais; preserva causalidade sem ordenação global.
Detecção de extremidades	Nós sem filhos servem como pontos de referência para anexar novos eventos.
Consistência eventual	As réplicas convergem mesmo com atualizações concorrentes ou perda de conectividade.
Desempenho em larga escala	Potencial crescimento da largura do DAG pode afetar performance; requer otimizações.
Classificação	CRDT orientado a operações com semântica de merge e resolução de conflitos embutida.

Esse modelo de replicação permite alta robustez frente a falhas de conectividade e auditoria do histórico completo de eventos, o que o torna adequado para ambientes federados e descentralizados como os propostos pelo Matrix. No entanto, seu crescimento pode impactar o desempenho de servidores em larga escala. A ausência de uma política padronizada de podagem e a exigência de que todos os pais estejam presentes na réplica antes da aplicação de um evento aumentam a complexidade operacional em ambientes instáveis.

Estratégias como *state resolution algorithms*, compactação de grafos, otimização da propagação de eventos e algoritmos para controle da largura do DAG continuam sendo

objeto de estudo e aprimoramento contínuo [Jacob et al. 2021]. Esses pontos reforçam que o MEG, embora elegante e funcional, ainda representa uma fronteira ativa de pesquisa em replicação distribuída.

1.2.4. Mecanismos Avançados de Criptografia (Olm e Megolm)

A segurança no protocolo Matrix é baseada em E2EE, com suporte dos protocolos Olm[16] e Megolm[17], voltados respectivamente para comunicações *Peer-to-Peer* (P2P) e em grupo [Albrecht et al. 2023, Foundation 2025]. O Olm baseia-se no *Triple Diffie-Hellman* (3DH)[18] e no Double Ratchet[19] e estabelece sessões seguras entre pares. O Megolm, por sua vez, é uma extensão voltada à comunicação em grupo, concebida para maximizar a eficiência em contextos com múltiplos participantes.

Adicionalmente, o Matrix implementa recursos como *cross-signing*, que permite que os próprios usuários autentiquem os dispositivos confiáveis de sua conta, contribuindo para a verificação de identidade e mitigação de ataques do tipo *man-in-the-middle*. Há também suporte a backup criptografado de chaves nos servidores domésticos, permitindo a recuperação de mensagens após perda de dispositivos [Li et al. 2023].

Embora o protocolo forneça E2EE por padrão, também emprega TLS para proteger o tráfego entre clientes e servidores, e entre servidores na federação. A E2EE é conduzida exclusivamente por Olm e Megolm, o primeiro responsável pela distribuição de material criptográfico para os integrantes da sala e o segundo utilizado para troca de mensagens, tanto individuais como em grupos, já que todos os chats são tratados como salas no Matrix.

De forma mais detalhada, o Olm é responsável por estabelecer um canal seguro entre dois dispositivos, oferecendo confidencialidade, autenticidade, *Forward Secrecy* (FS), *Post-Compromise Security* (PCS) e negabilidade. Sua implementação original, em C/C++, foi disponibilizada como uma API, mas atualmente integra a biblioteca vodozemac[20], escrita em Rust.

Já o Megolm implementa sessões unidirecionais entre o dispositivo remetente e cada destinatário, tais sessões distribuídas individualmente por canais Olm. A sessão é composta por uma catraca simétrica que é avançada, derivando uma nova chave de cifra para cada mensagem. A autenticação é fornecida por assinaturas Ed25519 e HMAC-SHA256, e as propriedades de segurança dependem da integridade dos canais Olm usados na distribuição.

A Figura 1.5 ilustra o processo simplificado de envio de uma mensagem criptografada no Matrix entre Alice e Bob. Os passos de ❶ a ❿ abrangem desde a geração e troca de chaves até a cifração, assinatura e posterior decifração no dispositivo do receptor.

No passo ❶, ocorre a geração das chaves de sigilo e de assinatura. As partes

[16] `https://gitlab.matrix.org/matrix-org/olm`

[17] `[https://gitlab.matrix.org/matrix-org/olm/-/blob/master/docs/megolm.md`

[18] O 3DH proporciona uma troca de chaves inicial entre as partes através de curvas elípticas.

[19] O Double Ratchet é um protocolo criptográfico que combina chaves assimétricas e simétricas para garantir sigilo direto e sigilo futuro.

[20] `[https://matrix-org.github.io/vodozemac/vodozemac/index.html`

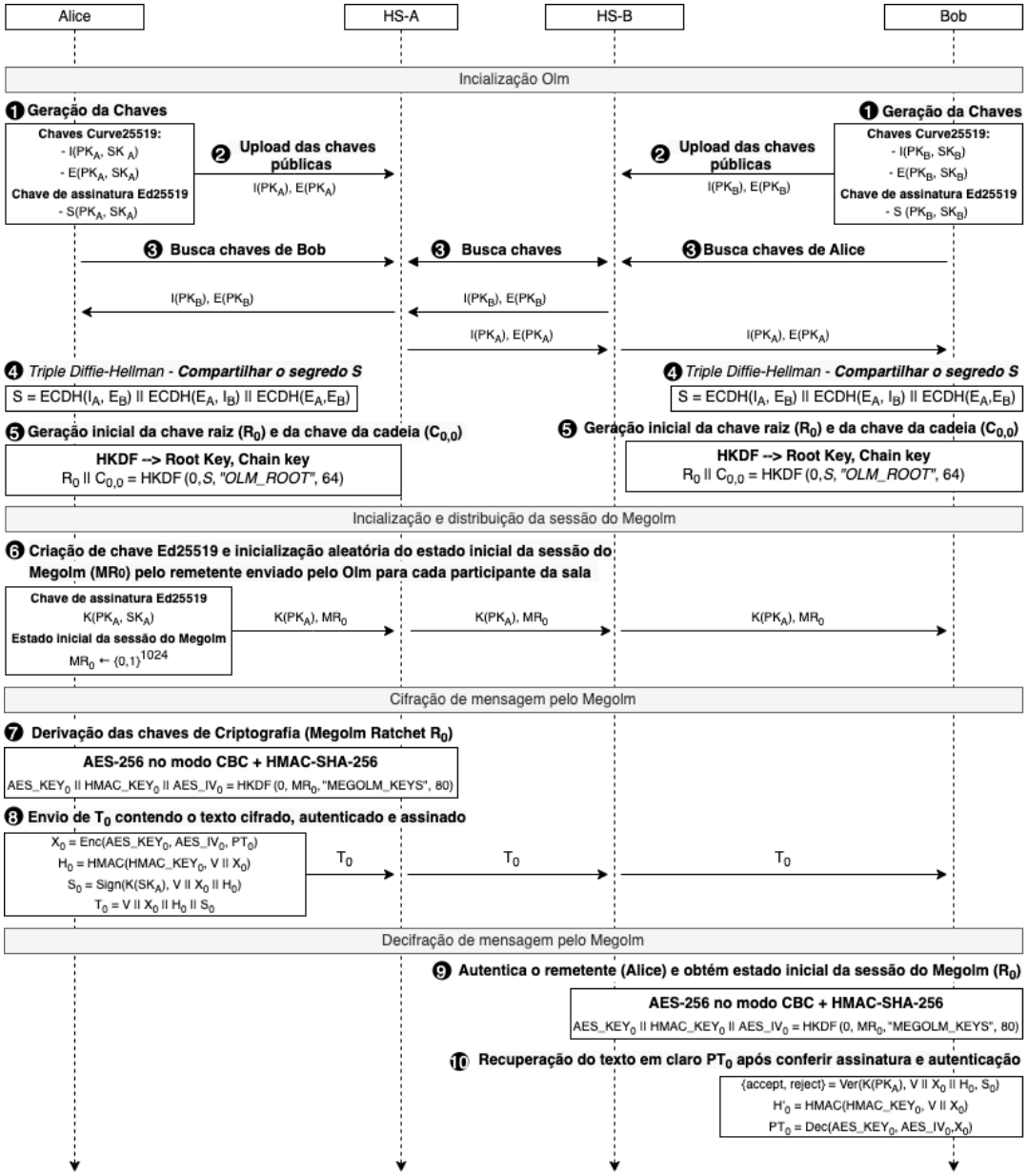

Figura 1.5. E2EE via Olm e Megolm.

públicas dessas chaves são publicadas nos respectivos *homeserver* (HS) de Alice e Bob ❷. Quando Alice e Bob desejam se comunicar, cada um busca nos HS apropriados as chaves da outra parte ❸, para que o protocolo 3DH seja executado ❹. O resultado do 3DH é um segredo compartilhado S, que serve como insumo para a derivação da root key (R_0) e da chain key ($C_{0,0}$) do Double Ratchet presente no Olm ❺, concluindo assim a inicialização da sessão Olm.

A etapa seguinte consiste na inicialização e distribuição da sessão do Megolm. Para isso, Alice — a remetente, neste cenário — cria um par de chaves para assinatura e inicializa aleatoriamente o estado inicial da sessão do Megolm (MR_0) ❻. A parte pública

da chave de assinatura criada por Alice e o estado inicial da sessão do Megolm precisam ser transmitidos de forma segura para Bob, a fim de que ele consiga, posteriormente, verificar e decifrar as mensagens cifradas por Alice. Essa transmissão segura é realizada utilizando o Olm, cuja sessão já foi previamente estabelecida. Assim, Bob recebe de maneira segura o estado inicial da sessão Megolm (MR_0).

Com o estado inicial compartilhado, Alice pode dar início ao processo de cifração da mensagem em claro PT_0. Para isso, ela realiza uma derivação de chaves utilizando *HMAC-based Extract-and-Expand Key Derivation Function* (HKDF) sobre o estado inicial do Megolm (MR_0), obtendo as chaves de cifração, o vetor de inicialização (IV) e o material para autenticação ❼. De posse desses parâmetros, a mensagem é cifrada, autenticada e assinada, sendo então transmitida para Bob ❽.

Bob, por sua vez, como já possui de forma segura o estado inicial da sessão Megolm (MR_0), pode replicar o processo descrito no passo ❼, obtendo exatamente a mesmas chave de cifração, o IV e o material para autenticação ❾. Dessa forma, Bob é capaz de verificar a assinatura da mensagem com a chave pública de Alice previamente recebido em ❻, confirmar sua autenticidade e, em caso de sucesso, proceder com a decifração do conteúdo ❿.

A Tabela 1.6 sintetiza a comparação entre Olm e Megolm em aspectos como finalidade, mecanismo de inicialização, sigilo, autenticação, eficiência e escalabilidade.

Tabela 1.6. Comparação entre os protocolos Olm e Megolm

Característica	**Olm**	**Megolm**
Uso Principal	Mensagens ponto-a-ponto (1:1)	Mensagens em grupo (1 remetente → N receptores)
Inicialização	Triple Diffie-Hellman (3DH)	Distribuição via canais Olm
Ratchet Criptográfico	Double Ratchet simétrico e assimétrico	Megolm Ratchet simétrico
FS	Forte (ratchet por mensagem)	Limitado (receptores podem manter estados anteriores)
PCS	Presente (se pre-keys são assinadas)	Limitado (requer renovação de sessão manual)
Autenticação	Chaves de identidade (Curve25519)	Chave de assinatura do remetente (Ed25519)
Criptografia Utilizada	AES-CBC-256 + HMAC-SHA256	AES-CBC-256 + HMAC-SHA256
Granularidade de Segurança	Por mensagem	Por sessão (unidirecional)
Eficiência	Menos eficiente, mais seguro	Mais eficiente para grupos, com compromissos de segurança
Escalabilidade	Limitada: requer um canal Olm por par de dispositivos	Alta: uma única sessão pode ser compartilhada com vários dispositivos receptores

Enquanto Olm aplica o Ratchet simétrico do Double Ratchet para fornecer FS por mensagem (envio de sessões do Megolm), o Ratchet assimétrico é responsável por prover PCS. Já o Megolm apresenta limitações nesse aspecto, especialmente quando sessões não são atualizadas regularmente. Conforme destacado pela documentação oficial [Matrix.org 2022], é necessário que os clientes descartem sessões antigas — seja avançando os valores da catraca ou reiniciando sessões — para minimizar a reutilização de chaves. Além disso, preservar o PCS requer a criação periódica de novas sessões Megolm, distribuídas por canais Olm seguros.

Por fim, embora Megolm seja menos robusto em certos aspectos de segurança, ele proporciona vantagens significativas em termos de desempenho e escalabilidade em grupos. Uma única sessão pode ser compartilhada com vários dispositivos receptores, enquanto Olm exige um canal separado para cada par de dispositivos, o que o torna menos adequado para ambientes com muitos participantes. A escolha entre os dois protocolos depende, portanto, do cenário de uso e dos requisitos de segurança envolvidos.

O principal desafio, tanto no Olm quanto no Megolm, para o fortalecimento da soberania digital reside na capacidade de integrar, de forma nativa, algoritmos criptográficos proprietários do Estado ao protocolo. Essa medida visa eliminar qualquer vulnerabilidade que comprometa a soberania, especialmente diante da hipótese de que algoritmos comerciais venham a ser quebrados por agências estrangeiras. Nessas circunstâncias, a dependência de algoritmos de mercado poderia configurar riscos reais de influência indevida, vazamento de informações sensíveis ou até mesmo de espionagem externa.

Portanto, a verdadeira soberania digital no contexto da comunicação criptografada não se resume à adoção de software livre ou à descentralização das infraestruturas. Ela exige, também, a capacidade técnica e política de incorporar algoritmos criptográficos sob controle estatal, auditáveis e alinhados às normativas locais de segurança e privacidade. A flexibilidade arquitetural do protocolo Matrix — especialmente nos mecanismos Olm e Megolm — oferece um ponto de partida promissor, mas seu pleno aproveitamento ainda depende de esforços de pesquisa, padronização e engenharia criptográfica soberana.

1.2.5. Interoperabilidade e Integração com Outras Plataformas

Um dos diferenciais do protocolo Matrix em relação a outras soluções de mensageria é sua arquitetura voltada à interoperabilidade. O sistema permite a criação de *bridges* para conectar o ecossistema Matrix a outras plataformas de comunicação como Slack, Telegram, Discord, IRC e até servidores de e-mail. Essas bridges atuam como tradutores de protocolos, possibilitando que usuários interajam entre redes distintas sem a necessidade de múltiplas contas[21] ou interfaces [Cui et al. 2023, Foundation 2025].

A estrutura básica das bridges envolve a escuta e tradução dos eventos Matrix para o formato do protocolo externo, e vice-versa. Elas podem ser implementadas de forma unidirecional, que permite o envio de mensagens em apenas uma direção, geralmente do Matrix para outra plataforma, ou de forma bidirecional, que suportam a troca mútua de mensagens, mantendo a comunicação sincronizada entre os dois sistemas.

[21] Em muitas implementações, as bridges podem mapear identidades externas para usuários Matrix virtuais, criando experiências unificadas.

A Figura 1.6 ilustra a integração realizada por uma bridge entre um homeserver Matrix e a plataforma Slack. As mensagens enviadas na sala `friends` do homeserver `matrix.org` são capturadas pela bridge, traduzidas e encaminhadas para a sala `friends` no Slack, e o processo ocorre igualmente no sentido inverso.

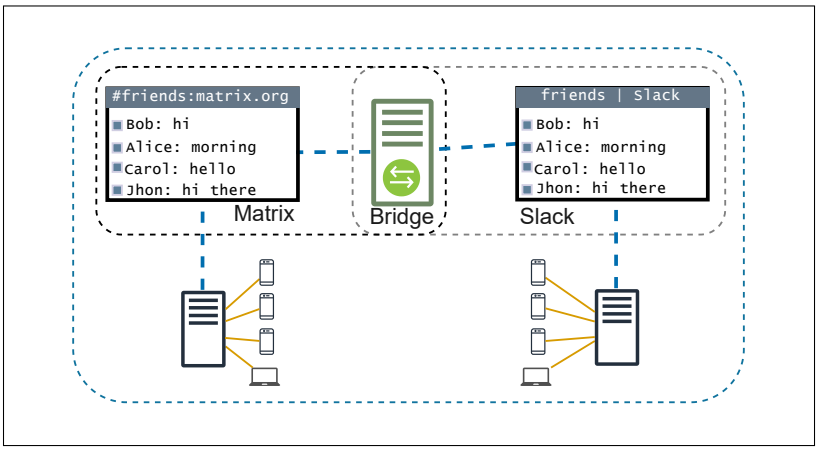

Figura 1.6. Integração entre Matrix e outras plataformas via bridge (ex: Slack, Telegram, IRC)

O gerenciamento dos estados e eventos nas bridges é fundamental para preservar a coerência das conversas e dos históricos, exigindo mecanismos que lidem com diferenças nos modelos de dados e nos formatos dos protocolos envolvidos.

O ecossistema Matrix conta com uma ampla variedade[22] de bridges que permitem a integração com diversas plataformas de comunicação, tais como Slack, Discord, Telegram, WhatsApp, Signal, IRC, XMPP, Messenger, Skype, Google Chat, WeChat, SMS, iMessage, LinkedIn, Twitter, Instagram, Mastodon, entre outros. A diversidade de bridges disponíveis amplia significativamente o alcance e a aplicabilidade do Matrix, tornando-o uma solução versátil para comunicação descentralizada em ambientes heterogêneos. Alguns cenários de uso estratégicos são comunicação segura entre agências com infraestruturas heterogêneas, integração de canais institucionais legados e ambientes de colaboração em tempo real. Além disso, aplicações descentralizadas baseadas no Matrix têm sido exploradas em áreas como Internet das Coisas (IoT), redes comunitárias, jornalismo seguro e plataformas educacionais, reforçando seu papel como protocolo versátil e extensível [Li et al. 2023].

Apesar dos benefícios que a interoperabilidade traz para o protocolo Matrix, essa integração enfrenta desafios técnicos e conceituais que impactam diretamente sua eficiência e escalabilidade. Um dos principais desafios é a complexidade envolvida na tradução entre protocolos heterogêneos. Cada plataforma de comunicação possui seu próprio modelo de presença, formatos específicos para mensagens, regras para gerenciamento de salas, permissões e sinais de estado, além de diferentes capacidades para lidar com recursos multimídia e eventos em tempo real. Traduzir esses modelos distintos de forma fiel e eficiente requer um esforço para preservar a integridade da comunicação e evitar perdas ou distorções de informação.

[22]Lista de bridges: https://matrix.org/ecosystem/bridges/

Outro desafio reside na manutenção da coerência e sincronização dos estados e mensagens entre sistemas diversos. Em ambientes distribuídos e federados, onde múltiplos eventos podem ocorrer simultaneamente ou fora de ordem, garantir que todos os participantes tenham uma visão consistente das conversas torna-se uma tarefa complexa. Isso é especialmente desafiador em cenários de alta concorrência, quando múltiplas bridges e usuários interagem intensamente, e nos casos de eventos assíncronos, que podem introduzir latências e divergências temporárias no estado das salas.

Além disso, questões relacionadas à segurança e privacidade ganham maior complexidade na interoperabilidade. Proteger dados sensíveis enquanto se mantém a compatibilidade entre diferentes sistemas com políticas e mecanismos de segurança variados é uma tarefa delicada e contínua.

1.2.6. Estrutura Conceitual das APIs

O protocolo Matrix é estruturado sobre um conjunto de APIs formais que definem a comunicação entre seus componentes principais. Essas interfaces são publicadas pela Matrix Foundation em uma especificação técnica padronizada, mantida de forma aberta e evolutiva.

A evolução do protocolo Matrix não se dá apenas no plano técnico, mas também na forma como suas especificações são desenvolvidas, debatidas e mantidas publicamente. A governança técnica do ecossistema é estruturada de maneira transparente e colaborativa, com base em dois mecanismos principais:

- **Matrix Spec**: a especificação técnica formal do protocolo, publicada e mantida pela Matrix Foundation[23]. A spec define o comportamento esperado das APIs (Client-Server, Server-Server, Identity Service e Application Service), os tipos de eventos suportados, os fluxos de autenticação, a criptografia ponta-a-ponta e os formatos de comunicação padronizados entre clientes e servidores.

- **Matrix Spec Changes (MSCs)**: propostas públicas de alteração ou extensão da spec, discutidas abertamente pela comunidade técnica via GitHub[24]. As MSCs passam por processo formal de revisão técnica e, quando aprovadas, são incorporadas em versões futuras da spec oficial. Esse processo permite que a evolução do protocolo se mantenha ágil e responsiva às necessidades dos desenvolvedores.

Dentre essas interfaces, destacam-se principalmente a *Client-Server API* (CS API) e a *Federation API* (também chamada de Server-Server API). A CS API é a principal interface utilizada pelos clientes para interagir com seus homeservers. Ela oferece um conjunto rico de funcionalidades, incluindo autenticação de usuários, envio e recebimento de mensagens, criação e gerenciamento de salas, sincronização incremental do estado da rede e verificação de dispositivos para garantir a segurança das comunicações, especialmente em contextos com criptografia de ponta a ponta. Por sua vez, a Federation API

[23]Especificações oficiais disponíveis em `https://spec.matrix.org`

[24]Repositório de propostas MSC: `https://github.com/matrix-org/matrix-spec-proposals`

possibilita a comunicação entre homeservers, lidando com a replicação de eventos, resolução de conflitos de estado, verificação das assinaturas digitais entre servidores e a propagação de atualizações na rede federada. Essa distinção entre CS API e Federation API reflete a separação lógica entre o plano do usuário final e o plano da infraestrutura do servidor, garantindo descentralização e independência tecnológica dos participantes da rede [Li et al. 2023, Foundation 2025].

Além dessas APIs centrais, o ecossistema Matrix define outras interfaces especializadas que expandem suas capacidades. A *Application Service API* é projetada para permitir que *app services* se integrem profundamente ao protocolo. Essa API possibilita que aplicações terceiras monitorem eventos, enviem mensagens e participem da lógica de salas, tudo isso com permissões específicas e controle granular.

A *Identity Service API* complementa a arquitetura ao facilitar a associação e verificação de identidades externas ao Matrix, como endereços de e-mail ou números de telefone, mapeando-os para usuários Matrix. Esse serviço auxilia em fluxos de registro, recuperação de contas e provisionamento de identidades, ampliando a usabilidade e integração do protocolo com sistemas legados[25].

Por fim, a *Push Gateway API* trata da entrega de notificações push para dispositivos móveis e outros clientes que operam em ambientes restritivos, como redes móveis, onde conexões persistentes são inviáveis. Essa API fornece um canal eficiente e padronizado para que os homeservers encaminhem notificações externas, garantindo que os usuários sejam alertados em tempo real sobre mensagens e eventos relevantes, sem comprometer a autonomia da infraestrutura Matrix[26].

Em suma, essa arquitetura baseada em múltiplas APIs especializadas permite que o protocolo Matrix mantenha uma separação clara entre funções, promovendo uma rede federada, extensível e segura, na qual diferentes atores cooperam para viabilizar comunicações abertas e interoperáveis.

1.3. Aplicações Práticas do Protocolo Matrix

A compreensão de um protocolo de comunicação não se limita a seus fundamentos técnicos e arquiteturais. A forma como suas funcionalidades se traduzem em práticas cotidianas — especialmente do ponto de vista do usuário — é essencial para avaliar sua maturidade, segurança e aplicabilidade institucional. Esta seção apresenta uma abordagem prática das operações com o protocolo Matrix, tendo como foco a utilização do cliente Element X[27]. O objetivo é demonstrar, com base em uma perspectiva prática, como usuários podem operar a infraestrutura federada sem a necessidade de instalar ou administrar um servidor próprio.

As subseções abordam, de forma aplicada, a criação de contas, o uso das funcionalidades básicas, o gerenciamento de salas e permissões e, por fim, os recursos de comunicação segura. Ao invés de apresentar um tutorial passo a passo, o foco está na

[25] https://spec.matrix.org/latest/identity-service-api/

[26] https://spec.matrix.org/latest/push-gateway-api/

[27] O Element X é a nova geração de cliente oficial do protocolo Matrix, com ênfase em usabilidade e desempenho, disponível para dispositivos móveis e navegadores.

estrutura conceitual dessas operações e nas boas práticas recomendadas para seu uso em ambientes profissionais, governamentais ou educacionais.

1.3.1. Aspectos Conceituais na Configuração e Operação de Clientes Matrix

O protocolo Matrix foi projetado para ser acessível tanto por usuários finais quanto por desenvolvedores, permitindo sua adoção em larga escala sem a necessidade de operar servidores próprios. Essa característica é particularmente importante para instituições que desejam experimentar o protocolo de forma imediata, avaliando sua experiência de uso e funcionalidades antes de realizar implantações locais.

Uma das formas mais diretas de explorar o Matrix é por meio do serviço público `matrix.org`, que disponibiliza um homeserver gratuito e funcional mantido pela fundação responsável pelo protocolo. Usuários podem registrar-se nesse serviço e utilizar clientes como o *Element X*, o novo cliente oficial do ecossistema Matrix voltado para dispositivos móveis e desktops [Foundation 2025].

O processo de entrada envolve a criação de uma conta no domínio `matrix.org`, seguida da autenticação no aplicativo. O Element X oferece uma interface moderna e intuitiva, compatível com práticas atuais de design de aplicativos de comunicação. Após o login, o cliente realiza a sincronização inicial com o homeserver, permitindo ao usuário acessar suas salas, mensagens, notificações e dispositivos registrados.

O menu de configurações do Element X inclui opções para alterar nome de exibição, gerenciar dispositivos ativos, habilitar ou desabilitar notificações, e ativar E2EE em salas privadas. É possível também buscar por salas públicas por meio de diretórios federados, ingressar em discussões abertas e iniciar conversas privadas com outros usuários do ecossistema.

A Tabela 1.7 apresenta uma seleção representativa dos principais clientes Matrix disponíveis atualmente, com destaque para suas plataformas-alvo, linguagens de implementação e características técnicas mais relevantes. O *Element*, nas versões para desktop e dispositivos móveis, destaca-se como cliente oficial e mais amplamente adotado no ecossistema, oferecendo suporte completo à criptografia ponta a ponta, Spaces, Threads e a sincronização rápida introduzida pelo Sliding Sync. Sua variante móvel, o *Element X*, foi redesenhada para melhorar a experiência em smartphones, com ênfase em desempenho, fluidez e integração nativa.

Entre as alternativas, o *FluffyChat* se sobressai pela leveza, portabilidade e proposta de interface amigável, ideal para novos usuários. O *SchildiChat*, por sua vez, adapta o Element com foco em usabilidade tradicional e múltiplas contas. Já o *Nheko* é reconhecido por sua leveza, desempenho nativo e foco em segurança, sendo especialmente atrativo para usuários avançados em ambientes desktop. Essa diversidade de clientes evidencia a flexibilidade e maturidade do protocolo Matrix, permitindo adoções personalizadas em diferentes contextos de uso, desde ambientes institucionais até comunidades autônomas.

Como exercício prático, recomenda-se que os leitores criem uma conta no `matrix.org`, acessem o cliente Element X, localizem uma sala pública e realizem o envio de sua primeira mensagem. Essa atividade fornece uma compreensão imediata das capacidades do

Tabela 1.7. Comparativo entre clientes Matrix populares

Cliente	Plataforma	Linguagem	Destaques
Element X	Android, iOS	Kotlin, Swift	Interface moderna, Sliding Sync, suporte E2EE por padrão
Element	Desktop, Web, Linux, Android, macOS, Windows	JavaScript/React	Interface web baseada em Electron, suporte completo a Spaces e Threads
FluffyChat	Android, iOS, Web	Dart (Flutter)	Leve, amigável, alternativa ao Element, código aberto
SchildiChat	Android, Web, Desktop	Kotlin	Derivado do Element, interface estilo "mensageiros clássicos", suporte a múltiplas contas
Nheko	Linux, Windows	C++/Qt	Cliente leve, nativo, foco em desempenho e segurança

protocolo e do fluxo básico de uso de seus clientes oficiais.

1.3.2. Gerenciamento de Salas, Grupos e Permissões

No protocolo Matrix, a comunicação ocorre principalmente por meio de salas, que funcionam como espaços lógicos para interação entre usuários. As salas podem ser públicas, privadas ou protegidas, dependendo de seus parâmetros de visibilidade, acesso e autenticação. Compreender a estrutura dessas salas e a lógica de permissões associada é essencial para garantir segurança, organização e governança nas comunicações institucionais.

Salas públicas são acessíveis por qualquer usuário da federação Matrix. Elas podem ser descobertas por meio de diretórios globais ou locais e permitem ingresso livre, sem convite. Já as salas privadas são visíveis apenas a seus membros, exigindo convite para entrada. Salas protegidas funcionam como intermediárias: são visíveis publicamente, mas exigem autenticação ou aprovação manual para ingresso [Foundation 2025].

A gestão de salas envolve a atribuição de papéis aos participantes. O protocolo define níveis hierárquicos de poder, como administrador (gerencia configurações e usuários), moderador (controla permissões e conteúdo) e participante comum. Esses papéis são atribuídos por meio de valores numéricos chamados *power levels*, configuráveis em cada sala. A Tabela 1.8 resume os principais tipos de sala e suas configurações típicas de permissão.

Tabela 1.8. Resumo dos tipos de sala no Matrix e suas propriedades de acesso

Tipo de sala	Visível publicamente	Entrada livre	Criptografia E2EE
Pública	Sim	Sim	Opcional
Privada	Não	Não (convite)	Recomendado
Protegida	Sim	Requer aprovação/autenticação	Recomendado

É recomendável que salas privadas ou protegidas adotem E2EE, recurso suportado nativamente pelo Matrix, garantindo que somente os dispositivos autorizados possam descriptografar o conteúdo. A ativação do E2EE é feita por sala, e pode ser combinada com verificação de dispositivos e autenticação cruzada para ambientes mais sensíveis.

A governança local torna-se ainda mais relevante em contextos organizacionais. A capacidade de definir políticas de acesso, criar grupos de usuários com papéis distintos e aplicar mecanismos de moderação adequados é essencial para garantir integridade, privacidade e conformidade legal na comunicação digital [Bechara and Lechner 2024].

1.3.3. Comunicação Segura: Chamadas de Áudio e Vídeo (Element X)

O suporte a chamadas de áudio e vídeo é uma funcionalidade estratégica do protocolo Matrix, especialmente relevante em contextos que demandam privacidade, segurança e descentralização. Por meio do cliente Element X, é possível realizar chamadas diretas entre usuários com suporte à E2EE, sem intermediação de servidores centrais de mídia.

A tecnologia base utilizada para o transporte de mídia é o WebRTC (Web Real-Time Communication), que permite comunicação peer-to-peer entre dispositivos. O papel do protocolo Matrix é limitado à troca de sinalização (session initiation) entre os clientes envolvidos na chamada. Uma vez estabelecida, a chamada utiliza canais diretos com negociação de rota realizada via ICE, incluindo o uso de servidores STUN/TURN para contornar firewalls e NATs [Foundation 2025, Rahman and Wang 2024].

Do ponto de vista de segurança, o protocolo Matrix adota os algoritmos Olm (mensagens individuais) e Megolm (grupos) para proteger as comunicações. Chamadas realizadas entre usuários em salas com E2EE ativado herdam os mecanismos de proteção criptográfica já definidos para a sessão. Além disso, recomenda-se que os dispositivos envolvidos realizem verificação cruzada (cross-signing), assegurando que os pares são legítimos e não foram adulterados durante o processo de emparelhamento.

Por fim, o Element X também permite ativar backups criptografados de chaves, garantindo que sessões anteriores possam ser restauradas com segurança em caso de reinstalação ou perda de dispositivos. Esse recurso, aliado à verificação entre dispositivos, fortalece a proteção contra ataques de intermediário (MITM) e amplia a confiabilidade da comunicação em tempo real.

Como exercício opcional, sugere-se que os participantes iniciem uma chamada com outro usuário confiável e testem a verificação cruzada de dispositivos, observando as etapas de emparelhamento, ativação de E2EE e início da sessão multimídia.

1.4. Aspectos Avançados em Servidores Privados Matrix

A adoção do protocolo Matrix em contextos institucionais demanda, em muitos casos, a operação de servidores próprios. Essa prática permite maior controle sobre políticas de segurança, retenção de dados e conformidade regulatória — aspectos centrais da soberania digital. Nesta seção, exploramos a arquitetura do *Synapse*, o servidor de referência para Matrix, e discutimos desafios técnicos associados à sua implantação em ambientes reais.

1.4.1. Arquitetura do Servidor Synapse

Embora o protocolo Matrix permita o uso de servidores públicos como o `matrix.org`, uma das principais vantagens da arquitetura federada está na possibilidade de implantação de servidores privados. Essa abordagem garante maior controle sobre políticas de auten-

ticação, armazenamento e segurança dos dados, além de possibilitar conformidade com requisitos institucionais e regulatórios. Esta seção aborda os principais aspectos envolvidos na operação de servidores Matrix, com foco na arquitetura do Synapse, nos desafios técnicos relacionados à sua implantação em escala e nos requisitos para comunicação síncrona segura.

1.4.1.1. Visão Geral

A Figura 1.7 ilustra a arquitetura de um homeserver básico baseado no Synapse. O Synapse é a principal implementação de referência do protocolo Matrix, desenvolvido originalmente pela equipe da Matrix Foundation. *Open Source* (licença Affero GNU AGPLv3) e escrito em Python, ele apresenta uma arquitetura monolítica e modular, com suporte à decomposição por *workers*, o que possibilita escalabilidade horizontal em implantações com alta demanda. Seus principais componentes incluem o servidor de aplicação, banco de dados, serviços auxiliares de armazenamento, autenticação e replicação federada [Foundation 2025, Synapse 2025].

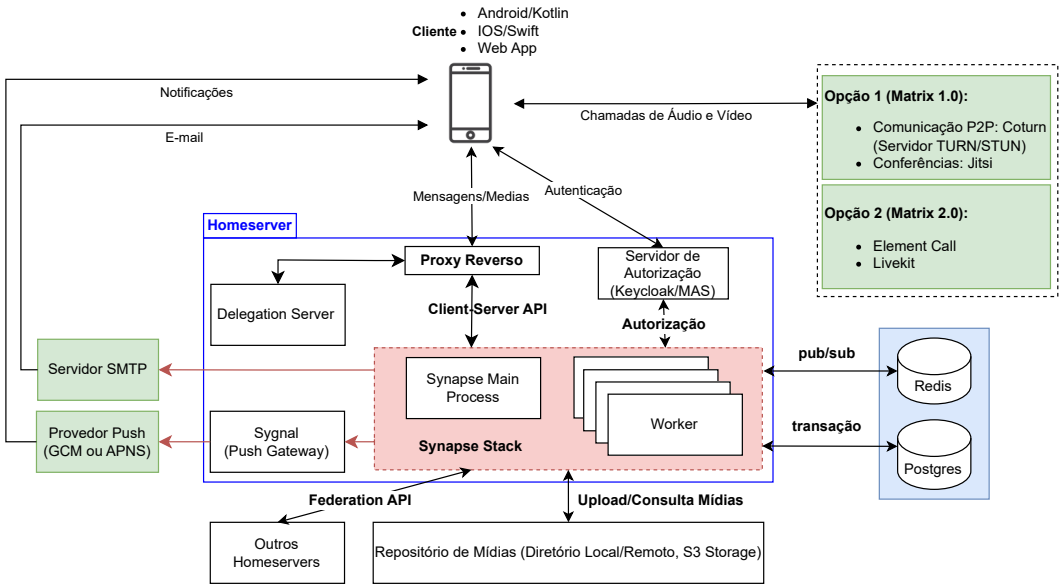

Figura 1.7. Arquitetura de um Homeserver baseado no Synapse.

A seguir descrevemos cada um dos componentes da arquitetura, que devem ser implantados nos diferentes ambientes.

- **Cliente Matrix**: Desenvolvido nas versões mobile (Android/Kotlin e iOS/Swift) e Web. Permite envio de mensagens de texto e mídias, além da realização de chamadas de voz.

- **Proxy Reverso**: Realiza as funções de API Gateway e Proxy Reverso. Gerencia o tráfego entre os clientes e os serviços *backend*. Redireciona tráfego para os serviços da *Stack* do Synapse ou para o *Delegation Server*.

- **Delegation Server**: Antes de tudo, vale ressaltar que necessitamos de duas URLs para o funcionamento do sistema. São elas: **(i) URL do Matrix**: permite que um administrador de servidor doméstico mantenha um server_name de "example.com" para que IDs de usuário, aliases de sala, etc. continuem parecendo: "example.com"; e (ii) **URL dos Serviços**: permite acessar os *endpoints* dos diferentes serviços na arquitetura. Por exemplo, "matrix.example.com". Neste cenário, o *Delegation Server* é um recurso do Matrix que permite que um administrador de servidor doméstico mantenha um server_name de example.com para que IDs de usuário, aliases de sala, etc. continuem parecendo ":example.com", enquanto o tráfego de federação é roteado para um servidor e/ou porta diferente (por exemplo, "synapse.example.com:443"). O Delegation Server pode ser implementado em servidores Web tradicionais, como Apache e NGINX.

- **Synapse Stack**: Compõe o ecossistema Synapse os seguintes processos:

  - **Main Process**: Processo principal do servidor Synapse. Responsável pelas funções administrativas do ecossistema Synapse (Synpase Admin API), bem como pelo processo de sincronização entre os diferentes workers do ecossistema. Em uma instalação típica, o Synapse mantém somente um *Main Process* que processa eventos de clientes locais, sincroniza dados com outros servidores da federação, aplica lógica de acesso a salas e persiste informações em bancos relacionais como PostgreSQL. A comunicação federada é realizada por meio da Federation API, enquanto o relacionamento com os clientes é feito pela Client-Server API.
  - **Workers**: Os "workers" são aplicados quando é necessário escalar o Synapse, ajudando a distribuir a carga de trabalho do *Main Process* e melhorar a escalabilidade e o desempenho do servidor. Detalharemos os diferentes tipos de *workers* mais à frente.

- **Push Gateway (Sygnal)**: Sygnal é um Push Gateway de referência para Matrix. Ele envia mensagens via Google/Firebase Cloud Messaging (GCM/FCM) e Apple Push Notification Service (APNS) e, portanto, pode ser usado para entregar notificações para aplicativos Android e iOS.

- **Camada de Dados**:

  - **Postgres**: Banco de dados relacional para persistência de dados do Synapse (todos os tipos de instâncias).
  - **Redis**: Cache e armazenamento de dados em memória, voltado para a implementação de cache, armazenamento de dados temporários e para a sincronização entre as diferentes instâncias do Synapse.
  - **Repositório de mídias**: voltado para o armazenamento de arquivos de mídia (imagens, vídeos, documentos, etc). Podem ser utilizados diretórios locais e

remotos (via *Network File System* - NFS, por exemplo), bem como um *storage* S3.

- **Servidor de Autorização**: Sua principal função é verificar a identidade dos usuários e conceder ou negar permissão para que eles realizem ações específicas dentro da rede Matrix. Essencialmente, ele atua como um porteiro digital, garantindo que apenas usuários autenticados e com as devidas permissões possam: acessar salas específicas, enviar mensagens, modificar configurações, etc. No Synapse, o servidor de autorização utiliza os protocolos modernos OAuth 2.0 e OpenID Connect (OIDC). Neste sentido, até então, temos dois serviços de autorização suportados pelo Synapse (considerando o Matrix 2.0):

  - O **MAS (Matrix Authentication Service)**: Agora mais conhecido como matrix-authentication-service, é um serviço de autenticação e autorização independente e autônomo projetado para o ecossistema Matrix;
  - **Keycloak**: É uma solução de software de código aberto para Gerenciamento de Identidade e Acesso (IAM). Ele permite que você proteja aplicações e serviços com pouco ou nenhum código, adicionando funcionalidades de autenticação e autorização.

1.4.1.2. Tipos de Workers

Workers são processos separados que executam tarefas específicas na infraestrutura do servidor Matrix. Eles são responsáveis por lidar com diferentes aspectos do funcionamento do Synapse, como processamento de eventos, manipulação de solicitações de API, replicação de dados, entre outros. Os "workers" são uma parte essencial para implementar uma arquitetura distribuída do Synapse, ajudando a distribuir a carga de trabalho e melhorar a escalabilidade e o desempenho do servidor. Até a data da escrita deste documento, existem os seguintes tipos de *workers*:

- **pusher**: É possível designar *workers* para enviar notificações *push* para um *gateway push*, como Sygnal e E-mail. Isso interromperá o processo principal de enviar notificações *push*. Vários *pushers* podem ser adicionados, caso em que a carga de trabalho é balanceado entre eles. Ou seja, suporta balanceamento de carga via *Homeserver*. Importante destacar a necessidade de reiniciar o *Main Process* e todos os *workers* após alterar esta opção. Esses *workers* não precisam aceitar solicitações HTTP de entrada para enviar notificações *push*;

- **user_dir**: Um diretório de usuários é mantido com base nos usuários que são "visíveis" para o servidor doméstico, ou seja, aqueles locais no servidor e aqueles com os quais qualquer usuário local compartilha uma sala. Além disso, permite pesquisa de texto completo com base em IDs de usuário e nomes de exibição. Esse *worker* é responsável por executar trabalhos em segundo plano relacionados à atualização do diretório do usuário e executar uma busca por usuários;

- **media_repository**: Clientes pode fazer *upload* e *download* de conteúdos de um repositório de mídias. Este *worker* trata as requisições direcionadas para o gerenciamento do repositório de mídias. Se estiver executando vários repositórios deste tipo, eles devem estar no mesmo servidor e deve-se especificar uma única instância para executar as tarefas em segundo plano na configuração compartilhada. Suporta balanceamento de carga com escalabilidade horizontal automática;

- **appservice**: Lida com o envio de tráfego de saída para *Application Services*. Este *worker* não pode ter carga balanceada. Portanto, deve haver apenas um *worker appservice*.

- **federation_sender**: É possível delegar os processos que manipulam o envio de solicitações de federação de saída. Isso removerá a necessidade de realizar essa função pelo *Main Process*. Vários *workers* deste tipo podem ser adicionados, onde o trabalho é balanceado entre eles. A maneira como o balanceamento de carga funciona é que qualquer solicitação de federação de saída será atribuída a um *worker* remetente de federação com base no *hash* do nome do servidor de destino. Isso significa que todas as solicitações enviadas ao mesmo destino serão processadas pela mesma instância de *worker*. Vários federation_sender_instances são úteis se houver uma federação com vários servidores. Essa configuração deve ser compartilhada entre todos os *workers* que manipulam o envio de federação e, se alterada, todos os *workers* remetentes de federação devem ser interrompidos ao mesmo tempo e, em seguida, iniciados para garantir que todas as instâncias estejam sendo executadas com a mesma configuração (caso contrário, os eventos podem ser descartados).

- **client_reader**: Responsável pelo Client-Server API. Suporta balanceamento de carga com escalabilidade horizontal automática. Contudo, solicitações de registro/login podem ser tratadas separadamente (Main Process ou um Client Reader só para elas) apenas para garantir que cargas inesperadas não afetem novos logins e inscrições;

- **federation_reader**: lida com o envio de solicitações de federação (Server-Server API). Suporta balanceamento de carga com escalabilidade horizontal automática;

- **federation_inbound**: Responsável por enviar EDUs (Ephemeral Data Unit) e PDUs entre homeservers, que representam atividades ao vivo. Cada PDU contém um único evento de Sala que o servidor de origem deseja enviar ao destino. EDUs, em comparação com PDUs, não têm um ID, um ID de sala ou uma lista de IDs "anteriores". Elas são destinadas a serem dados não persistentes, como presença do usuário, notificações de digitação, etc. Suporta balanceamento de carga via proxy reverso.

- **event_persister**: Responsável por receber novos eventos, vinculá-los aos que já estão no DAG (gráfico acíclico direcionado) da sala, persisti-los no banco de dados e, finalmente, atualizar o fluxo de eventos. O fluxo de eventos suporta ter vários *Event Persisters*, onde a carga é dividida entre eles por ID da sala. Ou seja, suporta balanceamento de carga via *Homeserver*. Certifique-se de que o *Main Process* e

todos os *workers* sejam reiniciados após alterar esta opção. Esses *workers* não precisam aceitar solicitações HTTP de entrada;

- **background_worker**: O *worker* é usado para executar tarefas em segundo plano (por exemplo, limpeza de dados expirados). Se não for fornecido, o padrão será o processo principal. Este *worker* não pode ser fragmentado. Portanto, deve haver apenas um *background worker*. Isso é imposto para a segurança do seu banco de dados.

- **event_creator**: Escuta solicitações de clientes para criar novos eventos. Ele então passará esses eventos por replicação HTTP para quaisquer *Event Persisters* configurados (ou para o *Main Process*, se nenhum estiver configurado). Suporta balanceamento de carga com escalabilidade horizontal automática.

- **frontend_proxy**: O cliente envia requisições para este *worker* para publicar as chaves públicas de criptografia de ponta a ponta e algoritmos suportados pelo seu dispositivo.

- **account_data**: Este *worker* persiste as alterações nos dados da conta de usuários locais. Funciona como um escritor para o fluxo de notificações de atualizações de dados de conta.

- **presence**: Cada usuário tem o conceito de informação de presença. Isso codifica: (1) Se o usuário está online no momento; (2) Quão recentemente o usuário esteve ativo pela última vez (conforme visto pelo servidor); (3) Se um determinado cliente considera o usuário como ocioso no momento; (4) Informações arbitrárias sobre o status atual do usuário (por exemplo, “em uma reunião”). Essas informações são coletadas de dados por dispositivo (online, ocioso, last_active) e por usuário (status), agregados pelo *Homeserver* e transmitidos como um evento “m.presence”. Eventos de presença são enviados para partes interessadas onde os usuários compartilham uma associação de sala. Este *worker* funciona como um escritor para o fluxo de notificações de presença.

- **receipts**: Recibos são uma forma de reconhecimento de um evento. Os clientes devem enviar recibos quando houver alguma certeza de que o evento em questão foi exibido ao usuário. Este *worker* funciona como um escritor para o fluxo de notificações de recibos.

- **to_device**: Este *worker* fornece um meio pelo qual os clientes podem trocar mensagens de sinalização sem que elas sejam armazenadas permanentemente como parte de um histórico de comunicação compartilhado. Uma mensagem é entregue exatamente uma vez para cada dispositivo cliente. A principal motivação para esta API é trocar dados que não têm sentido ou que não são desejados para persistir no DAG da sala - por exemplo, tokens de autenticação única ou dados de chave. Não se destina a dados de conversação, que devem ser enviados usando a API normal “/rooms/<room_id>/send” para consistência em todo o Matrix.

- **typing**: Os usuários podem desejar ser informados quando outro usuário estiver digitando em uma sala. Isso pode ser feito usando notificações de digitação. Este *worker* é responsável por lidar com a escrita das notificações de digitação.

- **push_rules**: Os usuários também podem configurar várias regras que determinam quais eventos geram notificações *push*. Uma regra *push* (push-rule) é uma regra única que define sob quais condições um evento deve ser passado para um *push gateway* e como a notificação deve ser apresentada. Todas elas são armazenadas e gerenciadas pelo *homeserver* do usuário. Isso permite que as configurações de *push* específicas do usuário sejam reutilizadas entre aplicativos clientes. Este *worker* funciona como um escritor para o fluxo de notificações de *push-rules*.

Cada tipo de *worker* no Synapse desempenha um papel específico no processamento e na entrega de mensagens e eventos na rede Matrix. Eles trabalham juntos para garantir que o servidor Synapse funcione de forma eficiente e confiável, mesmo em ambientes de alta carga e escalabilidade.

1.4.1.3. Componentes para Comunicação em Tempo Real

Tradicionalmente, a operação de um servidor Matrix privado envolvia a integração com ferramentas externas para viabilizar funcionalidades complementares, como chamadas de voz e vídeo. Nesse modelo, os componentes mais utilizados eram o *Coturn*, um servidor TURN/STUN necessário para permitir a comunicação em redes com NAT, e o *Jitsi Meet*, utilizado para videoconferência por meio da ponte `jitsi-meet-tokens` e integração via bot/bridge com salas Matrix [Foundation 2020b, Foundation 2020a].

Nesse cenário, a sinalização de chamadas era realizada por meio do protocolo Matrix, enquanto a mídia era transmitida fora do domínio Matrix, através de conexões diretas entre clientes mediados por servidores Jitsi e Coturn. Tal separação trazia limitações à implementação de E2EE em chamadas em grupo, comprometendo a privacidade em certos cenários institucionais.

Recentemente, com o surgimento do padrão MSC3401 — que especifica a sinalização WebRTC diretamente sobre o protocolo Matrix — surgiu a possibilidade de um novo paradigma: o de chamadas nativamente integradas à rede Matrix [Foundation 2023]. Essa mudança foi operacionalizada pela ferramenta *Element Call*, que implementa chamadas individuais e em grupo com suporte completo a E2EE, utilizando o próprio Matrix como meio de sinalização e gerenciamento de sessões. Para o transporte eficiente da mídia em larga escala, o *Element Call* pode ser integrado ao *LiveKit*, uma solução moderna baseada em WebRTC com suporte a SFU (Selective Forwarding Unit), melhorando o desempenho e escalabilidade de videoconferências [Element 2025a, Element 2025c]. A Tabela 1.9 resume as diferenças principais entre as duas arquiteturas.

Tabela 1.9. Comparativo entre arquiteturas de mídia no ecossistema Matrix.

Componente	Arquitetura Tradicional	Arquitetura Moderna
Sinalização	Matrix + Bot Jitsi	Matrix MSC3401 (nativo)
Serviço de Mídia	Jitsi Meet	LiveKit (via Element Call)
E2EE Suportado	Parcial (chamadas diretas)	Sim (grupo e P2P)
Relay de Mídia	Coturn	Coturn (ainda necessário)
Integração com Matrix	Indireta (bot externo)	Direta via Client/Server API

1.4.2. Desafios Técnicos: Segurança, Desempenho e Escalabilidade

A implantação de um servidor Synapse em ambiente institucional requer uma abordagem técnica cuidadosa. Desde a instalação inicial até ajustes finos de desempenho e segurança, cada etapa influencia diretamente a confiabilidade do sistema e a proteção dos dados trafegados. Esta subseção apresenta orientações práticas e discute os principais desafios técnicos enfrentados durante a operação de um homeserver privado Matrix.

O processo de instalação pode ser realizado por meio de pacotes pré-compilados, imagens Docker ou scripts automatizados mantidos pela Matrix Foundation[28]. Após a instalação, é necessário configurar arquivos como `homeserver.yaml`, onde são definidas as portas de escuta, domínios autorizados, políticas de autenticação e parâmetros de retenção.

A segurança do Synapse envolve múltiplos aspectos. Primeiramente, recomenda-se ativar o suporte a TLS por meio de um proxy reverso (como Nginx ou Caddy), garantindo que toda comunicação seja cifrada. Além disso, a autenticação pode ser integrada a serviços externos como LDAP ou OpenID Connect, permitindo unificação de credenciais e autenticação de múltiplos fatores. A configuração de permissões por sala e a limitação de recursos por usuário são práticas recomendadas para evitar abusos e contenção de recursos.

Em relação ao armazenamento, o Synapse oferece suporte a SQLite e PostgreSQL. Para ambientes de produção, é altamente recomendado utilizar PostgreSQL, que oferece melhor desempenho, suporte a concorrência e maior robustez em operações complexas [Synapse 2025]. O SQLite pode ser suficiente para testes locais ou pequenas implantações de avaliação, mas não é adequado para uso contínuo com múltiplos usuários simultâneos.

A escalabilidade do Synapse pode ser alcançada por meio de técnicas como separação de workers, cache persistente, tuning de banco de dados e otimização do uso de CPU e disco. A documentação oficial oferece guias detalhados sobre como distribuir a carga de trabalho entre processos distintos, possibilitando o crescimento progressivo da infraestrutura conforme a demanda [Foundation 2025].

[28] Ver documentação oficial: `https://matrix-org.github.io/synapse/latest/`.

Quadro – Checklist para configuração segura e escalável do Synapse

- Configurar proxy reverso com TLS.
- Integrar autenticação externa (LDAP, SSO).
- Habilitar verificação de dispositivos e E2EE.
- Utilizar PostgreSQL como backend.
- Limitar conexões, tamanho de mensagens e eventos.
- Aplicar separação de workers e tuning de banco.

Recomenda-se que os leitores instalem um servidor local de teste, configurem um usuário administrativo e validem a comunicação entre clientes Matrix e o homeserver implantado. Essa experiência prática permite compreender o ciclo completo de autenticação, envio e sincronização de mensagens em uma rede federada.

1.4.3. Comunicação em Tempo Real: Considerações Práticas

O suporte a comunicação síncrona, como chamadas de voz e vídeo, é um dos diferenciais do protocolo Matrix em relação a outras soluções federadas. Esse recurso é especialmente importante em contextos institucionais que demandam autonomia na infraestrutura de comunicação. Nesta subseção, discutimos os requisitos técnicos, desafios e boas práticas relacionados à operação de chamadas em tempo real em servidores Synapse.

As chamadas no Matrix são viabilizadas por meio da tecnologia WebRTC, que permite a comunicação direta entre dispositivos sem a necessidade de servidores intermediários. O papel do Synapse nesse processo se limita à sinalização entre os participantes da chamada — o transporte de mídia ocorre por conexões peer-to-peer sempre que possível [Foundation 2025, Rahman and Wang 2024].

Para que essa comunicação seja bem-sucedida, é necessário configurar corretamente a rede e os serviços auxiliares. Em especial, o uso de servidores STUN e TURN é essencial para atravessar NATs e firewalls. O protocolo ICE (Interactive Connectivity Establishment) é utilizado para negociar o melhor caminho entre os pares. A ausência ou má configuração desses elementos pode impedir completamente o estabelecimento de chamadas[29].

O desempenho das chamadas também depende da infraestrutura de rede e dos recursos computacionais do servidor. Embora a mídia não transite diretamente pelo Synapse, sobrecargas no sistema — como lentidão na sincronização de eventos ou picos de carga no banco de dados — podem prejudicar a experiência do usuário, especialmente no tempo de sinalização inicial. Recomenda-se monitorar o consumo de CPU, memória e uso de rede, além de aplicar limites a salas ou usuários com uso abusivo.

Do ponto de vista da segurança, recomenda-se ativar criptografia ponta-a-ponta

[29] Ver guia de configuração de chamadas em `https://matrix-org.github.io/synapse/latest/usage/voip.html`.

em salas privadas e realizar verificação cruzada de dispositivos antes do início das sessões. Em ambientes institucionais, é desejável restringir a funcionalidade de chamadas a salas ou domínios autorizados, mitigando riscos relacionados a chamadas não autenticadas ou a possíveis abusos da rede federada.

Como atividade prática, sugere-se realizar chamadas entre contas registradas em diferentes homeservers, observando o comportamento da sinalização e dos logs do servidor durante o processo de estabelecimento e encerramento da sessão.

1.4.4. Tutorial de Instalação do Matrix

Uma instalação mínima funcional de um servidor Matrix utilizando Synapse e Element pode ser facilmente implementada com Docker Compose. Um exemplo prático e bem documentado dessa configuração encontra-se no projeto oficial `element-docker-demo`, mantido pela equipe do Element [Element 2025b].

Os componentes essenciais utilizados neste projeto são:

- **Element Web:** Interface web oficial do Matrix para usuários finais.
- **Element Call:** Aplicação para chamadas de voz e vídeo com criptografia ponta-a-ponta.
- **Synapse:** Servidor Matrix responsável pela autenticação, sinalização e federação.
- **Matrix Authentication Service (MAS):** Serviço dedicado para autenticação integrada e OpenID Connect.
- **LiveKit:** Plataforma SFU para WebRTC, gerenciando mídia em chamadas grupais.
- **PostgreSQL:** Banco de dados relacional para armazenamento persistente e eficiente dos dados.
- **Nginx + Let's Encrypt ou mkcert (para TLS):** Proxy reverso configurado para segurança e criptografia HTTPS.
- **Coturn:** Servidor TURN/STUN para viabilizar chamadas através de redes restritivas.

A Figura 1.8 ilustra um diagrama simplificado desta instalação mínima.

Passos para execução:

1. Instale Docker Compose.
2. Caso utilize localmente, instale o `mkcert` para gerenciamento local de certificados TLS.

Execute o seguinte comando:

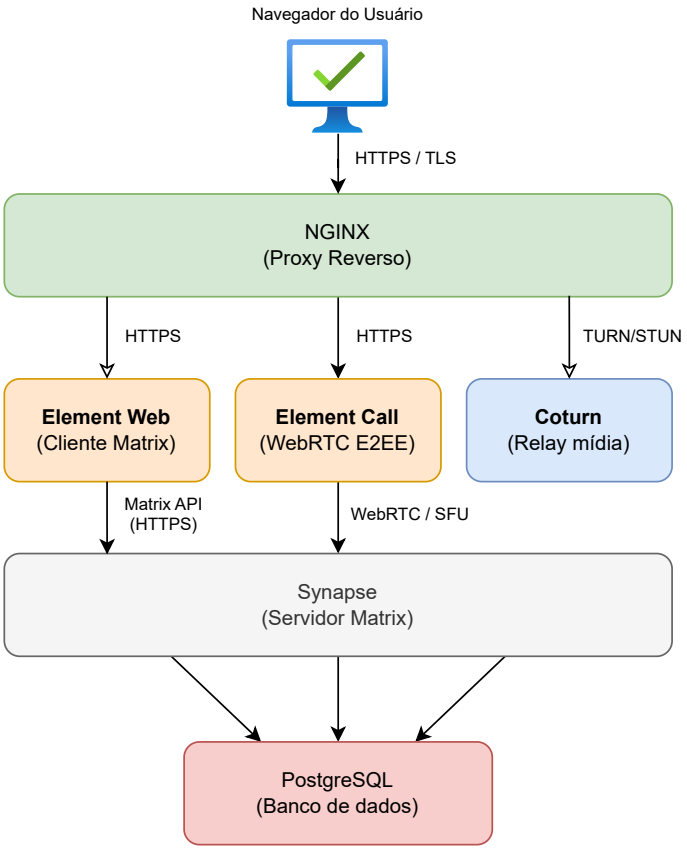

Figura 1.8. Diagrama simplificado da infraestrutura instalada

```
./setup.sh
```

Configure o DNS para apontar `*.seu-dominio` para o host Docker. Caso utilize localmente com `mkcert`, execute:

```
source .env
sudo sh -c "echo 127.0.0.1 $DOMAINS >> /etc/hosts"
```

Em seguida, inicie os containers:

```
docker compose up
```

Após iniciar, acesse pelo navegador em `https://element.seu-dominio`.

Configuração personalizada: Verifique o arquivo `.env` ou personalize os templates em `/data-templates`. Reinicie com os comandos:

```
docker compose down
docker compose up -d
```

Os dados dos containers são armazenados no diretório `./data`, e segredos em `./secrets`. Note que arquivos de configuração em `./data` são recriados pelos templates a cada inicialização.

Administração do sistema: Para atualizar os containers:

```
docker compose pull
```

Registrar um novo usuário:

```
docker compose exec mas mas-cli -c /data/config.yaml manage
   register-user
```

Diagnóstico e resolução de problemas: Para verificar a configuração OIDC e diagnosticar problemas com TLS:

```
docker compose exec mas mas-cli -c /data/config.yaml doctor
```

Agora temos uma infraestrutura Matrix pronta, funcional e fácil de utilizar.

1.5. Automação com APIs e Bots no Matrix

Apresentar o uso prático das APIs do Matrix para automação de tarefas e criação de bots, utilizando Python. A seção foca na interação programática com homeservers via Client-Server API, com exemplos reais e reprodutíveis, aplicáveis a contextos institucionais e educacionais.

1.5.1. Automação e Gerenciamento Através das APIs Matrix

A automação de interações com servidores Matrix pode ser realizada por meio de sua *Client-Server API* (CS API), que fornece uma interface RESTful [30] padronizada para envio de mensagens, autenticação, gerenciamento de salas e outras operações essenciais. Essa API é amplamente documentada e permite que scripts, aplicações e bots interajam programaticamente com um homeserver, sem depender da interface gráfica de clientes tradicionais como o Element.

Para fins de automação com Python, é possível utilizar diretamente bibliotecas como `requests` [31] , construindo manualmente as requisições HTTP. No entanto, para

[30] RESTful é um estilo arquitetural para comunicação entre sistemas baseado em operações HTTP padronizadas (como `GET`, `POST`, `PUT`, `DELETE`). Interfaces RESTful são amplamente utilizadas em APIs modernas pela simplicidade e compatibilidade com a Web.

[31] `requests` é uma biblioteca HTTP popular do ecossistema Python que permite enviar requisições REST com sintaxe simplificada.

simplificar o desenvolvimento e garantir conformidade com a especificação, é recomendável adotar bibliotecas especializadas como a `matrix-nio` [32] , que encapsula as chamadas da CS API e fornece uma estrutura orientada a eventos e sessões persistentes.

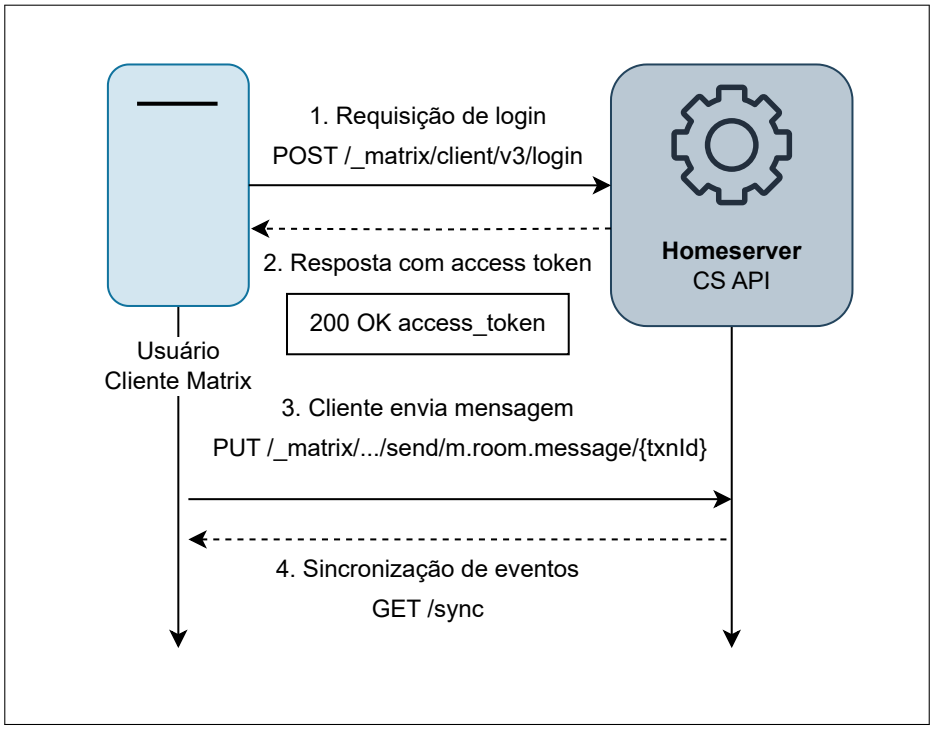

Figura 1.9. Fluxo de autenticação e envio de mensagem via CS API.

A seguir, apresenta-se um exemplo simples de script em Python que autentica um usuário com **token de acesso**[33] e envia uma mensagem a uma sala:

```
import requests

access_token = "YOUR_ACCESS_TOKEN"
room_id = "!roomid:matrix.org"
homeserver = "https://matrix.org"

message = {
    "msgtype": "m.text",
    "body": "Ola, esta e uma mensagem automatica!"
}

url = f"{homeserver}/_matrix/client/v3/rooms/{room_id}/send/m
   .room.message"
headers = {
    "Authorization": f"Bearer {access_token}"
```

[32]Disponível em: `https://github.com/poljar/matrix-nio`. Biblioteca Python para interação com a Client-Server API do Matrix, com suporte assíncrono, criptografia e gerenciamento de sessões.

[33]Tokens de acesso são credenciais temporárias que identificam um usuário autenticado. No Matrix, eles são usados em requisições HTTP via cabeçalhos `Authorization`.

```
}

response = requests.post(url, json=message, headers=headers)
print("Status:", response.status_code)
```

Listing 1.3. Envio de mensagem via CS API com Python

Comentários sobre o código:

O Listing 1.3 utiliza a biblioteca `requests` para interagir diretamente com o endpoint da Client-Server API. Primeiramente, são definidos o token de acesso, o *room_id* da sala e a URL base do homeserver. Em seguida, o corpo da mensagem é montado no formato JSON com o tipo de mensagem `m.text` e o texto desejado. A requisição POST é então enviada para o endpoint específico de envio de mensagens, com o token incluído no cabeçalho de autenticação (`Authorization: Bearer <token>`). A resposta HTTP indica o sucesso ou falha da operação.

A seguir uma lista de atividades uteis que podem ser executadas com **requests**:

- **Listar salas públicas disponíveis no servidor** (`GET /publicRooms`): Permite recuperar a lista de salas abertas em um homeserver Matrix, útil para explorar ambientes públicos sem autenticação completa.
- **Convidar um usuário para uma sala** (`POST /rooms/{roomId}/invite`): Envia um convite para outro usuário ingressar em uma sala. É uma forma prática de simular a gestão de participantes por bots ou scripts administrativos.
- **Modificar o nome da sala** (`PUT /rooms/{roomId}/state/m.room.name`): Atualiza o nome visível da sala para todos os participantes. Permite estudar os mecanismos de alteração de estado via API REST.
- **Definir ou alterar o tópico da sala** (`PUT /rooms/{roomId}/state/m.room.topic`): Define uma mensagem de topo visível no cabeçalho da sala. Essencial para sinalizar objetivos ou orientações de uso.
- **Encerrar a participação do usuário em uma sala** (`POST /rooms/{roomId}/leave`): Realiza a operação de saída voluntária da sala. Útil para estudar o ciclo de entrada e saída de ambientes federados.

Os exemplos com requests realizam o envio de uma mensagem simples com autenticação via token. Em contextos mais complexos, é possível usar a biblioteca `matrix-nio` para gerenciar sessões, receber eventos e reagir a comandos em tempo real, conforme será explorado na próxima subseção.

Exemplo com a biblioteca `matrix-nio`

A seguir, apresenta-se um exemplo simples de cliente usando a biblioteca assíncrona `matrix-nio` para enviar uma mensagem de forma assíncrona a uma sala. Essa abordagem é recomendada para aplicações persistentes e bots mais robustos, pois a biblioteca gerencia sessões, estado e eventos com maior controle.

```python
from nio import AsyncClient, RoomMessageText
import asyncio

async def main():
    homeserver = "https://matrix.org"
    user_id = "@seu_usuario:matrix.org"
    access_token = "YOUR_ACCESS_TOKEN"
    room_id = "!roomid:matrix.org"

    client = AsyncClient(homeserver, user_id)
    client.access_token = access_token

    await client.room_send(
        room_id=room_id,
        message_type="m.room.message",
        content={
            "msgtype": "m.text",
            "body": "Mensagem enviada com matrix-nio!"
        }
    )
    await client.close()

asyncio.run(main())
```

Listing 1.4. Envio de mensagem com matrix-nio

Comentários sobre o código:

No Listing 1.4, utilizamos a biblioteca assíncrona `matrix-nio` para enviar uma mensagem. Após instanciar o cliente com o endereço do homeserver e o ID do usuário, o token de acesso é atribuído diretamente. A função `room_send` é usada para enviar a mensagem com os parâmetros necessários: ID da sala, tipo da mensagem e o conteúdo JSON. A execução ocorre dentro de uma função assíncrona principal controlada por `asyncio`, garantindo que o envio aconteça corretamente. Por fim, o cliente é fechado com `client.close()`.

Esse exemplo assume que o token de acesso já foi obtido previamente. A biblioteca `matrix-nio` também permite lidar com eventos, respostas de usuários e múltiplos tipos de mensagens, sendo amplamente utilizada no desenvolvimento de bots e clientes personalizados.

Comparativo entre uso direto da API e uso de `matrix-nio`

Para automatizações simples, o uso direto da API via `requests` pode ser suficiente. Entretanto, para aplicações que exigem persistência, interação com eventos ou múltiplos recursos do protocolo Matrix, a biblioteca `matrix-nio` oferece uma base mais robusta. A Tabela 1.10 resume as principais diferenças entre as abordagens.

Tabela 1.10. Comparativo entre uso direto da CS API com `requests` e uso de `matrix-nio`

Aspecto	`requests` (HTTP direto)	`matrix-nio` (cliente dedicado)
Facilidade inicial	Simples para scripts pontuais e diretos	Requer conhecimento em programação assíncrona
Controle do fluxo HTTP	Total, com headers e JSON manualmente configurados	Abstraído por métodos prontos da biblioteca
Gerenciamento de estado	Manual (tokens, sessões, eventos)	Gerenciado internamente com persistência opcional
Recepção de eventos	Difícil (requere polling ou webhooks)	Nativo via callbacks e handlers reativos
Desempenho	Adequado para tarefas simples	Escalável e eficiente para bots e serviços contínuos
Complexidade do código	Reduzida, porém repetitiva e limitada	Inicialmente mais verboso, mas modular e reutilizável
Escalabilidade	Limitada a casos simples	Projetada para ambientes persistentes e multiusuário

Como mostrado na Tabela 1.10, a escolha entre as abordagens depende do escopo da aplicação. Para scripts pontuais, `requests` pode ser a melhor escolha. Já para bots reativos ou integrações mais complexas, a estrutura fornecida por `matrix-nio` proporciona maior robustez, modularidade e desempenho.

1.5.2. Desenvolvimento Conceitual e Prático de Bots

Bots são aplicações autônomas que interagem com o protocolo Matrix de maneira persistente e reativa. Embora scripts manuais também possam realizar operações programadas, os bots diferenciam-se por sua capacidade de escutar eventos, responder a mensagens em tempo real e manter sessões contínuas com o servidor.

Utilizando bibliotecas como `matrix-nio`, é possível desenvolver bots que monitoram salas, executam comandos personalizados, aplicam regras de moderação e coletam métricas de uso. Essa automação é útil em diversos contextos: comunidades técnicas, ambientes educacionais, suporte automatizado e gestão de canais institucionais.

A estrutura de um bot em Matrix geralmente inclui três elementos centrais: conexão ao servidor, escuta de eventos e lógica de resposta. A seguir, apresenta-se um exemplo simples de bot que responde a comandos iniciados com `!`:

```
from nio import AsyncClient, RoomMessageText
import asyncio

class SimpleBot:
    def __init__(self, client):
        self.client = client

    async def message_callback(self, room, event):
        if isinstance(event, RoomMessageText):
            if event.body.startswith("!ping"):
                await self.client.room_send(
```

```
                    room_id=room.room_id,
                    message_type="m.room.message",
                    content={"msgtype": "m.text", "body": "
                        pong!"}
                )

    async def run(self):
        self.client.add_event_callback(self.message_callback,
            RoomMessageText)
        await self.client.sync_forever(timeout=30000)

client = AsyncClient("https://matrix.org", "@seu_usuario:
    matrix.org")
client.access_token = "YOUR_ACCESS_TOKEN"
bot = SimpleBot(client)

asyncio.run(bot.run())
```

Listing 1.5. Bot simples com resposta automática a comandos

Comentários sobre o código:

Esse exemplo define uma classe `SimpleBot` que registra um *callback* para eventos de mensagens do tipo `RoomMessageText`. Quando uma mensagem que começa com `!ping` é detectada, o bot responde com `pong!`. A função `sync_forever` mantém a escuta ativa, garantindo reatividade contínua. O uso de `add_event_callback` permite modularizar o comportamento do bot com base em diferentes tipos de evento.

Exemplos de funcionalidades típicas de bots Matrix

Tabela 1.11. Exemplos de funcionalidades típicas de bots Matrix

Gatilho	Ação realizada	Finalidade / aplicação
`!ping`	Enviar resposta `pong!`	Testar conectividade e funcionamento do bot
`!contar`	Contabilizar número de mensagens na sala	Coleta de métricas de uso e atividade
`!ajuda`	Listar comandos disponíveis	Fornecer instruções aos usuários
Mensagem com palavra-chave proibida	Remover ou sinalizar mensagem	Moderação de conteúdo
Evento de horário específico	Enviar lembrete ou notificação	Agendamento de avisos ou atividades programadas
Entrada de novo usuário	Enviar mensagem de boas-vindas	Recepção automática de novos participantes

As funcionalidades listadas na Tabela 1.11 ilustram como bots podem ser adaptados a diferentes necessidades institucionais e comunitárias. A combinação de comandos

simples com ações programadas permite automatizar tarefas rotineiras, reduzir a sobrecarga de moderação manual e oferecer suporte contínuo a usuários. Além disso, a capacidade de interagir com eventos em tempo real torna os bots ferramentas versáteis para notificações contextuais, integração com sistemas externos e reforço de políticas de uso em salas de comunicação.

Bots podem ser estendidos para executar uma ampla variedade de ações. A combinação de `matrix-nio`, estrutura assíncrona e integração com outras bibliotecas Python torna o desenvolvimento acessível e poderoso mesmo em ambientes institucionais.

A seguir uma lista de bots úteis utilizando **matrix-nio**:

- **Bot que responde a comandos simples**: Detecta mensagens iniciadas por comandos prefixados como `!ajuda`, respondendo com instruções úteis. Base para interações automáticas com usuários.
- **Bot que envia mensagens automáticas em intervalos**: Utiliza tarefas assíncronas (`asyncio`) para enviar lembretes, notificações ou relatórios regulares para uma sala específica.
- **Bot que registra mensagens recebidas**: Armazena localmente o histórico das mensagens em arquivos texto. Serve como base para auditoria, monitoramento ou coleta de dados de uso.
- **Bot que saúda automaticamente novos participantes**: Detecta eventos de entrada (`m.room.member`) e envia mensagens de boas-vindas personalizadas. Útil para experiências de onboarding automatizado.
- **Bot moderador que remove mensagens com conteúdo ofensivo**: Filtra palavras-chave indesejadas e utiliza a API de `redact` para apagar mensagens de forma automática, simulando políticas de moderação.

Os códigos-fonte dos exemplos de automação com *matrix-nio* e **requests** apresentados neste capítulo estão disponíveis publicamente em `https://github.com/atlabufc/matrixjai2025`.

1.6. Desafios e Oportunidades de Pesquisa Relacionados ao Matrix

A maturidade técnica e a crescente adoção do protocolo Matrix não eliminam os desafios ainda presentes em sua evolução. Muitos dos problemas enfrentados em ambientes reais decorrem justamente de sua ambição técnica: ser federado, auditável, seguro e interoperável. Esta seção apresenta um panorama crítico sobre os obstáculos mais relevantes enfrentados atualmente e propõe caminhos promissores para a investigação científica.

1.6.1. Desafios Técnicos e Perspectivas Futuras

A interoperabilidade segura por meio de *bridges* permanece como um dos desafios mais relevantes para o avanço do Matrix. Embora a arquitetura aberta facilite a criação de conectores com outras plataformas, como Slack, IRC e Discord, manter equivalência

semântica e segurança na troca de mensagens entre sistemas heterogêneos é complexo. Questões como preservação de identidade, sincronização de histórico, e compatibilidade com E2EE impõem limites técnicos e operacionais [Cui et al. 2023, Li et al. 2023].

Outro aspecto importante envolve o desempenho e a escalabilidade, especialmente em grandes redes federadas. O custo computacional da replicação de eventos, da resolução de conflitos e do gerenciamento de chaves criptográficas cresce rapidamente em salas com alta atividade ou número elevado de participantes. O próprio modelo do Matrix Event Graph (MEG) apresenta desafios relacionados ao aumento da largura do grafo e à latência de convergência entre servidores [Jacob et al. 2021].

Além disso, ambientes regulados como instituições governamentais, sistemas de justiça ou redes hospitalares impõem requisitos adicionais de conformidade, armazenamento seguro e auditoria de acessos. A adoção plena do protocolo em tais contextos exige aprimoramentos no suporte a políticas de retenção, mecanismos de logging auditável e ferramentas de governança descentralizada.

1.6.2. Desafios em Aberto na Operação de Servidores Matrix

Embora o protocolo Matrix e sua implementação de referência, o Synapse, ofereçam uma base robusta para comunicação federada, a operação de servidores privados ainda envolve uma série de desafios técnicos e estratégicos. Essas limitações afetam tanto a adoção institucional quanto a manutenção de longo prazo em ambientes autônomos.

Um dos principais desafios diz respeito à **escalabilidade vertical e horizontal**. Apesar do suporte à separação de processos via workers, o Synapse ainda depende de um núcleo monolítico que concentra grande parte das operações críticas. A decomposição do servidor em serviços independentes está em andamento, mas requer maior maturidade para facilitar implantações em larga escala com balanceamento eficiente de carga [Synapse 2025].

Outro ponto importante envolve a **observabilidade e gerenciamento operacional**. Embora o Synapse ofereça suporte à exportação de métricas por meio de Prometheus, a configuração é pouco acessível e carece de ferramentas integradas de visualização e alerta. Isso dificulta a análise proativa de desempenho e o diagnóstico de falhas [Foundation 2024].

A **resiliência a falhas federadas** também é uma preocupação. Em ambientes descentralizados, a instabilidade de um servidor federado pode afetar a experiência global de usuários em diferentes domínios. Mecanismos mais robustos de isolamento, cache inteligente e fallback de federação são áreas em que o protocolo pode evoluir.

Do ponto de vista de segurança, permanece o desafio da **atualização contínua de políticas criptográficas**. A gestão segura de chaves, autenticação de dispositivos e proteção contra abusos ainda requer intervenção manual significativa. A automação desses processos, aliada à detecção de anomalias, é um campo promissor para pesquisa e desenvolvimento.

Por fim, o processo de instalação e manutenção ainda apresenta **curva de aprendizagem elevada**, o que limita o uso por organizações com menor infraestrutura técnica. A criação de instaladores simplificados, interfaces gráficas de configuração e documen-

tação contextualizada por casos de uso são aspectos que podem contribuir decisivamente para a ampliação da adoção do Matrix em ambientes governamentais, educacionais e comunitários.

1.6.3. Potenciais Áreas para Inovação e Investigação Científica

O protocolo Matrix apresenta um campo fértil para pesquisa aplicada e desenvolvimento experimental. A otimização da estrutura de replicação MEG é uma das frentes mais promissoras, com investigações voltadas à compactação de grafos, reordenamento eficiente de eventos e resolução distribuída de conflitos com menor custo computacional [Jacob et al. 2021].

Outra área em expansão envolve a interoperabilidade avançada entre mensageiros, com foco em modelos de conversão semântica entre protocolos distintos, autenticação cruzada e criptografia ponta-a-ponta mantida mesmo entre domínios heterogêneos [Cui et al. 2023]. Essa abordagem demanda novos modelos formais para tradução de eventos, bem como algoritmos de mediação que garantam integridade contextual das interações.

A segurança distribuída também segue como vetor central de inovação. Pesquisas sobre esquemas aprimorados de cross-signing, autenticação baseada em identidades descentralizadas (DIDs) e delegação criptográfica entre dispositivos podem consolidar o Matrix como padrão de mensageria segura e interoperável em escala global.

Tabela 1.12. Principais áreas de pesquisa e inovação no ecossistema Matrix

Área	Descrição	Objetivos de pesquisa
Replicação e MEG	Otimização do Matrix Event Graph (MEG) em redes de grande escala	Compactação, ordenação eficiente de eventos, resolução de conflitos com menor custo computacional
Interoperabilidade segura	Integração confiável entre Matrix e outras plataformas heterogêneas via *bridges*	Tradução semântica, preservação de E2EE, autenticação cruzada
Segurança distribuída	Fortalecimento da confiança e autenticação entre múltiplos dispositivos e domínios	Cross-signing, DIDs, delegação criptográfica, gerenciamento de dispositivos
Governança e conformidade	Adaptação do protocolo para ambientes regulados e institucionais	Auditoria de acessos, retenção de dados, logging e governança descentralizada

A Tabela 1.12 sintetiza essas frentes de pesquisa em quatro grandes áreas que refletem tanto os desafios técnicos quanto as exigências institucionais para a consolidação do protocolo Matrix. No eixo da replicação e desempenho, a estrutura do MEG demanda soluções escaláveis para ambientes federados extensos. Já a interoperabilidade segura segue como prioridade diante da diversidade de plataformas legadas. Na dimensão da segurança distribuída, aspectos como autenticação entre dispositivos e identidades descentralizadas se mostram essenciais para contextos sensíveis. Por fim, a governança e

conformidade regulatória emergem como requisitos críticos à adoção institucional, exigindo suporte nativo a políticas de auditoria, retenção e transparência operacional. Tais vetores indicam que o Matrix não é apenas um protocolo em evolução, mas também um campo estratégico para inovação em sistemas distribuídos.

1.7. Conclusão

Este curso proporcionou aos participantes uma compreensão abrangente e crítica sobre o protocolo Matrix, combinando fundamentos teóricos com aplicações práticas. Foram discutidas as bases conceituais e técnicas que sustentam o funcionamento do protocolo, incluindo sua arquitetura federada, os algoritmos de sincronização distribuída utilizados no MEG e os mecanismos de segurança criptográfica baseados nos protocolos Olm e Megolm.

Além dos aspectos conceituais, o curso explorou aplicações práticas do Matrix em cenários de comunicação segura, descentralizada e automatizada. Foram apresentados casos de uso em ambientes institucionais e exemplos de integração com ferramentas externas, incluindo automações via APIs e desenvolvimento de bots. Essas aplicações evidenciam o potencial do protocolo para atender demandas reais de privacidade, controle de dados e interoperabilidade.

Por fim, o curso destacou desafios técnicos ainda em aberto no desenvolvimento do Matrix, que representam oportunidades relevantes para pesquisas de mestrado e doutorado. Foram abordados temas como estratégias de sincronização federada em larga escala, interoperabilidade segura com outras plataformas e estudos de desempenho e escalabilidade em ambientes distribuídos. Ao apresentar essas questões de forma crítica e contextualizada, espera-se incentivar investigações acadêmicas e soluções inovadoras no campo da comunicação descentralizada, segurança digital e soberania tecnológica.

Agradecimentos

Os autores agradecem aos pesquisadores da Universidade Federal do Ceará (UFC), em especial a José G. R. Maia e Marcos D. Ortiz, cujas contribuições técnicas e discussões foram fundamentais para o amadurecimento e escrita deste capítulo.

Referências

[Albrecht et al. 2023] Albrecht, M. R., Celi, S., Dowling, B., and Jones, D. (2023). Practically-exploitable cryptographic vulnerabilities in matrix. *Proceedings of the IEEE Symposium on Security and Privacy*, pages 164–181.

[Authority 2024] Authority, L. (2024). Security audit report: vodozemac. Technical report, Least Authority.

[Bechara and Lechner 2024] Bechara, J. and Lechner, U. (2024). Digital sovereignty and open-source software: A discussion paper. *Digital Policy, Regulation and Governance*.

[Biström et al. 2024] Biström, M., Adolfsson, K. K., and Stocchetti, M. (2024). Open-source software and digital sovereignty: A technical case study on alternatives to mainstream tools. *Technology and Regulation*.

[Conduit 2025] Conduit (2025). Conduit: Lightweight matrix homeserver in rust. https://github.com/matrix-org/conduit. Acessado em 2025.

[Cui et al. 2023] Cui, M., Gu, C., and Nan, G. (2023). Crosschat: Instant messaging across different apps on mobile devices. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*.

[Element 2025a] Element (2025a). Element call integration with livekit. Accessed: 2025-05-22.

[Element 2025b] Element (2025b). Element docker demo. Accessed: 2025-05-22.

[Element 2025c] Element (2025c). Self-hosting element call. Accessed: 2025-05-22.

[Floridi 2020] Floridi, L. (2020). The fight for digital sovereignty: What it is, and why it matters, especially for the eu. *Philosophy and Technology*.

[Foundation 2020a] Foundation, M. (2020a). Docker-jitsi. Accessed: 2025-05-22.

[Foundation 2020b] Foundation, M. (2020b). Running matrix with jitsi. Accessed: 2025-05-22.

[Foundation 2023] Foundation, M. (2023). Docker-jitsi. Accessed: 2025-05-22.

[Foundation 2024] Foundation, M. (2024). Synapse metrics and monitoring. Acesso em 2025-05-16.

[Foundation 2025] Foundation, M. (2025). An open network for secure, decentralised communication. Acessado em 2025.

[Government 2020] Government, F. (2020). Tchap: The official messaging platform for french civil servants. Accessed: 2024-08-25.

[Jacob et al. 2021] Jacob, F., Beer, C., Henze, N., and Hartenstein, H. (2021). Analysis of the matrix event graph replicated data type. *IEEE Access*, 9:28317–28333.

[Klare and Lechner 2023] Klare, M. and Lechner, U. (2023). A reference model to strengthen digital sovereignty in companies. In *2023 IEEE International Conference on Digital Ecosystems (ICDE)*.

[Li et al. 2023] Li, H., Wu, Y., Huang, R., Mi, X., Hu, C., and Guo, S. (2023). Demystifying decentralized matrix communication network: Ecosystem and security. In *2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 260–267.

[Matrix Foundation 2025a] Matrix Foundation (2025a). Dendrite: Next generation matrix homeserver. https://github.com/matrix-org/dendrite. Acessado em 2025.

[Matrix Foundation 2025b] Matrix Foundation (2025b). Synapse: Reference matrix homeserver implementation. https://github.com/matrix-org/synapse. Acessado em 2025.

[matrix nio 2025] matrix nio (2025). nio: a multilayered matrix client library. Acessado em 2025.

[Matrix.org 2022] Matrix.org (2022). Megolm group ratchet. `https://gitlab.matrix.org/matrix-org/olm/-/blob/master/docs/megolm.md`. libolm repository.

[Mautrix 2025] Mautrix (2025). mautrix-python: Matrix client library for bots and bridges. `https://github.com/mautrix/python`. Acessado em 2025.

[Misra et al. 2025] Misra, S., Barik, K., and Kvalvik, P. (2025). Digital sovereignty in the era of industry 5.0: Challenges and opportunities. *Procedia Computer Science*.

[Moerel and Timmers 2021] Moerel, L. and Timmers, P. (2021). Data sovereignty and open source governance.

[Rahman and Wang 2024] Rahman, M. and Wang, Y. (2024). Implementation of dew-inspired matrix-mesh communication protocol. In De, D. and Roy, S., editors, *Dew Computing: The Sustainable IoT Perspectives*, pages 105–115. Springer Nature Singapore.

[Synapse 2025] Synapse (2025). Synapse: Welcome to the documentation repository for synapse, a matrix homeserver implementation developed by element. Accessed: 2025-01-09.