Erkenntnisse und Erfahrungen rund um Matrix • Am Interneteingang 8

Originalartikel

Backup

<html> Matrix ist ein neues Kommunikationssystem, allerdings nicht mit dem Anspruch, das nächste coole, hippe Teil zu werden, sondern Matrix setzt von Anfang an auf Integration und will Brücken zu anderen Systemen bauen. Die Kommunikation im Internet hat sich leider zu einer zerklüfteten Inselwelt entwickelt, in der jede Gruppe mit ihrem eigenen System arbeitet und inkompatibel zur Außenwelt ist. Da der Umzug von einem Netzwerk in ein anderes umständlich ist und viele Netzwerke ihre speziellen Vorteile haben, will Matrix die Nutzer eben dort nicht wegholen, sondern dort erreichen. Jeder darf das System nutzen, das ihm gefällt – egal ob XMPP, IRC, Whatsapp, Telegram, Hangouts, Facebook oder SMS – und Matrix kommuniziert mit ihnen. Ein Ansatz, der mir wahnsinnig gut gefällt! Wer sich Matrix erst einmal nur ansehen will, kann den Webclient Element benutzen und einen öffentlichen Raum wie <h3 id=„funktionstest“>Funktionstest</h3> Wenn alles durch ist, den Dienst (neu-)starten und dann sollte er auf dem Port 8008 lauschen. <pre>% sudo systemctl restart matrix-synapse.service% sudo ss -tlp |grep 8008LISTEN 0 0 ::1:8008 :::* users:[1)]</pre> Ob der Server für Clients erreichbar ist, kann man per netcat oder telnet prüfen (auch fürs System-Monitoring hilfreich): <pre>% netcat localhost 8008 <<<$'GET /_matrix/client/versions HTTP/1.0\r\n\r'HTTP/1.0 200 OKContent-Length: 54Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept, AuthorizationServer: Synapse/0.29.0Cache-Control: no-cache, no-store, must-revalidateDate: Sun, 20 May 2018 15:16:36 GMTAccess-Control-Allow-Origin: *Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONSContent-Type: application/json{„versions“: [„r0.0.1“, „r0.1.0“, „r0.2.0“, „r0.3.0“]}</pre> Im Log /var/log/matrix-synapse/homeserver.log sollte dann etwas in dieser Art stehen: <pre>2018-05-20 17:12:18,831 - synapse.access.http.8008 - 95 - INFO - GET-65532- - - 8008 - Received request: GET /_matrix/client/versions2018-05-20 17:12:18,832 - synapse.access.http.8008 - 129 - INFO - GET-65532- - - 8008 - {None} Processed request: 1ms (0ms, 0ms) (0ms/0ms/0) 54B 200 „GET /_matrix/client/versions HTTP/1.0“ „None“</pre> Ob der Server für andere Server erreichbar ist, also ob die Federation funktioniert, kann man über die Adresse

```
https://matrix.org/federationtester/api/report?server_name=&#8230;
```

mit dem entsprechenden Servernamen prüfen. In der Ausgabe sollte irgendwo

```
"AllChecksOK": true
```

erscheinen und am Ende sollte

```
"ConnectionErrors": {}
```

leer sein. Alternativ dazu gibt es noch den Fed-tester. <h2 id=„workereinrichten“>Worker einrichten</h2> Worker sind noch eine recht junge Entwicklung (Stand: Mai 2019) von Synapse, aber ihr Einsatz lohnt sich. Ziel der Entwicklung ist es, Teile aus dem Hauptprozess herauszulösen und in eigenständige Prozesse zu gliedern. Auf diese Weise kann man … <h2 id=„elementeinrichten“>Element einrichten</h2> Element ist ein Webclient für Matrix, der vom Matrix-Team entwickelt wird. Man kann ihn direkt von app.element.io aus nutzen oder man richtet sich eine eigene Instanz ein. Es gibt zwar ein Debian-Paket, aber dieses hängt vom Paket gconf ab und zieht auf einem Server einen Rattenschwanz an Paketen hinterher. Deshalb verwende ich das allgemeine Paket und entpacke es in /srv/www/m.jo-so.de. In dem Verzeichnis liegt die Datei config.sample.json, welche nach config.json kopiert werden muss. Darin sollte bei

```
default_hs_url
```

die Adresse des eigenen Matrix-Servers eingetragen werden. Ich habe bei mir die Anmeldung von Gästen deaktiviert

```
"disable_guests": true
```

, bei den Raum-Servern meinen eigenen ergänzt und bei piwik den Schlüssel

```
url
```

durch

```
X-url
```

ersetzt, damit auch sicher kein Piwik angesprochen wird. Matrix nutzt zwei verschiedene Kommunikationswege: 8008 für Clients und 8448 für andere Server, so wie es auch bei XMPP der Fall ist. Da Matrix eine REST-Schnittstelle verwendet, kann man den Verkehr zum eigentlichen Matrix-Dienst durch einen Nginx leiten, so dass der Matrix-Dienst nicht als root laufen muss und man die Möglichkeiten Nginx' nutzen kann: SSL-Zertifikate, HTTP/2.0, Einschränken der IP-Adressbereiche, HTTP-Authentifizierung und die Vermischung mit anderen Inhalten, denn Matrix nutzt nur das Verzeichnis /_matrix. <pre>server { listen 443 http2 ssl; listen [::]:443 http2 ssl; root /srv/www/default; location /_matrix/ { access_log off; proxy_pass http://[::1]:8008; proxy_set_header X-Forwarded-For $remote_addr; } location /.well-known/matrix/server { access_log off; add_header Access-Control-Allow-Origin *; return 200 '{„m.server“: „alea.gnuu.de:443“}'; } location /.well-known/matrix/ { access_log off; proxy_pass http://[::1]:8008; proxy_set_header X-Forwarded-For $remote_addr; } location /m/ { access_log off; alias /srv/www/m.jo-so.de/; }}% sudo systemctl reload nginx.service</pre> Als einfachere Funktionsprüfung kann man im Browser die Adresse

```
/_matrix/client/versions
```

seines Servers aufrufen und sollte eine Ausgabe ähnlich wie bei https://matrix.org/_matrix/client/versions erhalten. Ein kleines Skript für die regelmäßige Aktualisierung von Element habe ich in der Anleitung zu jq beschrieben. <h2 id=„neuenbenutzeranlegen“>Neuen Benutzer anlegen</h2> Man kann es allen Nutzern erlauben, ein neues Konto über die Weboberfläche anzulegen, in dem man in der Serverkonfiguration homeserver.yaml die Option

```
enable_registration: true
```

setzt. Falls man jedoch nur einen geschlossenen Nutzerkreis haben will, kann man in der Serverkonfiguration bei

```
registration_shared_secret: "&lt;SECRET KEY&gt;"
```

eine geheime Zeichenkette (generiert mit

```
pwgen -s 64 1
```

) eintragen und dann mit dem Kommandozeilenprogramm register_new_matrix_user neue Benutzer anlegen: <pre>% register_new_matrix_user -c /etc/matrix-synapse/homeserver.yaml http://localhost:8008New user localpart [root]: userPassword:Confirm password:Make admin [no]:Sending registration request...Success.</pre> Das Zurücksetzen des Passworts gibt es die händische Bearbeitung der Datenbank und eine API-Funktion, die man mit curl aufrufen kann. <h2 id=„turnfrvideo-undaudioverbindungeneinrichten“>TURN für Video- und Audioverbindungen einrichten</h2> Für Video- und Audioverbindungen ist neben dem Matrix-Server noch ein TURN-Server notwendig, der zwischen Geräten vermittelt, die sich nicht direkt erreichen können; z. B. durch NAT. Direkt von Debian gibt es das Paket coturn. Nach der Installation (

```
apt install coturn
```

) muss die Konfigurationsdatei /etc/turnserver.conf wie folgt angepasst werden (Entnommen aus der Doku von synapse): <pre>lt-cred-mechuse-auth-secretstatic-auth-secret=Geheimen Schlüssel mit `pwgen -s 64 1` erzeugenrealm=jo-so.deno-tcp-relaycert=/etc/letsencrypt/live/host/fullchain.pempkey=/etc/letsencrypt/live/host/privkey.pem# Matrix uses only TURNno-stunno-multicast-peersdenied-peer-ip=10.0.0.0-10.255.255.255denied-peer-ip=172.16.0.0-172.31.255.255denied-peer-ip=192.168.0.0-192.168.255.255pidfile=„/dev/null“secure-stunmobilityno-tlsv1</pre> Den

geheimen Schlüssel und die TURN-Verbindungen muss man noch in der homeserver.yaml eintragen und den Synapse-Server neustarten. <pre>turn_uris: - „turn:alea.gnuu.de:3478?transport=udp“ - „turn:alea.gnuu.de:3478?transport=tcp“turn_shared_secret=Geheimer Schlüssel</pre> Die Konfiguration für Systemd wird bei der Installation automatisch generiert, weshalb ich sie durch folgende ersetzt (

```
systemctl edit --full coturn.service
```

) habe. Damit läuft der Prozess auch nicht mehr als Benutzer root. <pre># https://github.com/coturn/coturn/blob/master/rpm/turnserver.service.fc[Unit]Description=coturnDocumentation=man:coturn(1) man:turnadmin(1) man:turnserver(1)After=network.target[Service]User=turnserverGroup=turnserverSupplementaryGroups=ssl-certPIDFile=/run/turnserver.pidExecStart=/usr/bin/turnserver -c /etc/turnserver.conf –log-file stdoutRestart=on-abortLimitNOFILE=999999LimitNPROC=60000LimitRTPRIO=infinityLimitRTTIME=7000000CPUSchedulingPolicy=otherUMask=0007NoNewPrivileges=yes[Install]WantedBy=multi-user.target</pre> AppArmor /etc/apparmor.d/usr.bin.turnserver <pre>include <tunables/global>profile /usr/bin/turnserver { include <abstractions/base> include <abstractions/ssl_certs> include <abstractions/ssl_keys> /etc/turnserver.conf r, owner /var/lib/turn/turndb rwk,}</pre> Sicherheitshinweis: Trotz der Einstellungen für Verschlüsselung des TURN-Servers konnte ich mit Wireshark noch die Inhalte der Verbindung einsehen. Ich habe nichts gefunden, wie ich die TURN-Verbindung weiter absichern kann. <pre>Source: laptop (Port: 59697)Destination: alea.gnuu.de (Port: 3478)Protocol: STUNInfo: Allocate Request UDP lifetime: 3600 user: 1527072691:@test:alea.gnuu.de realm: alea.gnuu.de with nonceSource: laptop (Port: 59697)Destination: alea.gnuu.de (Port: 3478)Protocol: STUNInfo: CreatePermission Request XOR-PEER-ADDRESS: 192.168.178.21:48261 user: 1527072691:@test:alea.gnuu.de realm: alea.gnuu.de with nonce</pre> Im Wireshark konnte ich aber auch beobachten, dass meine beiden Testgeräte direkt per IPv6 kommuniziert haben, ohne den Weg über den TURN-Server. Ich habe ursprünglich mit sqlite begonnen und irgendwann die Datenbank auf Postgres umgestellt. Hier die Befehle zum Einrichten des Benutzers und der Datenbank. Dabei ist darauf zu achten, dass der Benutzername identisch mit dem Systembenutzer seien muss, unter dem auch der Dienst läuft, sonst funktioniert die passwortlose Anmeldung per Socket nicht;

```
systemctl show -pUser matrix-synapse.service
```

. <pre>% sudo apt install python-psycopg2% sudo -u postgres sh -c 'createuser matrix-synapse && createdb -O matrix-synapse -l C.UTF-8 -T template0 matrix'</pre> In der Konfiguration homeserver.yaml müssen noch die Einstellungen wie folgt geändert werden: <pre># Database configurationdatabase: # The database engine name name: psycopg2 # Arguments to pass to the engine args: database: matrix user: matrix-synapse host: /run/postgresql application_name: matrix-synapse</pre> Zusätzlich zu den Parametern für connect kann man noch Parameter für das Connection-Pooling angeben. Leider ist zu dem von Synapse

genutzten Connection-Pool zu sagen, dass es ungenutzte (idle) Verbindungen nicht automatisch schließt. Eine Lösung hierfür habe ich noch nicht gefunden. Bei Systemd sollte man dann noch angeben, dass der Server erst nach Postgres gestartet wird: <pre>% sudo systemctl edit matrix-synapse.service[Unit]After=network-online.target postgresql.service</pre> Für die Datenmigration von sqlite zu Postgres gibt es das Programm synapse_port_db. Dieses kann auch mehrfach ausgeführt werden, falls man den Dienst weiterlaufen lässt und eine Kopie der sqlite-Datenbank verwendet hat oder es zu Fehlern kam. Ich habe aber einfach für die gesamte Zeit den Dienst bei mir abgeschaltet. <pre>% cd /tmp && sudo -u matrix-synapse synapse_port_db -v –curses \ –sqlite-database /var/lib/matrix-synapse/homeserver.db \ –postgres-config /etc/matrix-synapse/homeserver.yaml</pre> Wenn es zu einem Fehler kommt und in der Datei port-synapse.log etwas von

```
Failed to insert: room_depth
```

steht, dann zuerst einmal nachsehen, ob der Fehler 3214 bereits gelöst ist oder ggf. den

```
ALTER TABLE
```

-Befehl von dort ausführen und die Migration noch einmal starten. Auf der Webseite von Matrix werden diverse Anleitungen zur Wartung von Synapse und Moderation von Matrix angeboten. Wichtig für eine Wiederherstellung einer Matrix-Installation sind die Konfigurationdateien unter /etc/matrix-synapse, die Dateien unter /var/lib/matrix-synapse und die Datenbank. Um das Backup zu verkleinern kann man die Verzeischnisse /var/lib/matrix-synapse/media/(remote_|url_cache)*/* auslassen. Deren Inhalt lässt sich im Nachhinein über andere Server wiederherstellen. Bei der Wiederherstellung des Systems muss man mit dem Zugangsschlüssel (Access-Token) eines Admin-Benutzers den Cache zurücksetzen, damit Synapse die Dateien, die vom Backup ausgeschlossen waren, neu beschafft. Den Zugangsschlüssel findet man in Element unter Einstellungen, Hilfe, Fortgeschritten und nutzt diesen für purge_remote_media mit aktuellem Zeitstempel: <pre>curl -H „Authorization: Bearer <access_token>“ -d '' „http://localhost:8086/_synapse/admin/v1/purge_media_cache?before_ts=$(date +%s000)“</pre><h2 id=„ungenutzterumeentfernen“>Ungenutzte Räume entfernen</h2> Wenn Nutzer einen Raum verlassen, der über mehrere Server verteilt liegt, bleiben auf dem eigenen Server die alten Nachrichten zurück. Da diese nicht mehr genutzt werden, ist es sinnvoll, diese zu entfernen: <pre>mx_token=<Token eines Admins>for room in $(curl -s –header „Authorization: Bearer $mx_token“ \ 'http://localhost:8008/_synapse/admin/v1/rooms?limit=800' \ |jq -r '[.rooms[] | select(.joined_local_members == 0)] |sort_by(.state_events) | .[].room_id')do echo -n „Purging $room: “ time curl -s -o /dev/null –header „Authorization: Bearer $mx_token“ \ -X POST -H „Content-

Type: application/json“ -d „{ \room_id\“: \„$room\“ }„“ \ 'http://localhost:8008/_synapse/admin/v1/purge_room'done</pre> In der Tabelle state_groups_state legt Synapse irgendwelche Daten ab (weiß jemand welche? Stift oben rechts nutzen), die zum Teil redundant sind. Daher kann man diese Tabelle mit dem Programm rust-synapse-compress-state aufräumen und somit den Speicherverbrauch eindämmen. Wer Rust installiert hat, kompiliert das Programm besser selbst aus den Quellen, denn Releases passieren eher selten: <pre>cargo install –git https://github.com/matrix-org/rust-synapse-compress-state</pre> Danach liegt das Programm im Verzeichnis ~/.cargo/bin/synapse-compress-state. Beim Aufruf erstellt es aber nur eine SQL-Dateien, die die eigentlichen Änderungen beschreiben. Man muss also nach jedem Durchlauf des Programms noch das entstandene SQL-Skript ausführen. Daher habe ich mir das Hilfsprogramm synapse-clear-state erstellt, um auch alle anderen Schritte dafür durchzuführen. Dieses Hilfsprogramm rufe ich mithilfe von Systemd auf, damit es im Hintergrund läuft (bei mir dauert es über 2 Stunden) und ich die Meldungen im Nachhinein prüfen kann <pre>% sudo systemd-run –nice=4 -pCPUSchedulingPolicy=batch \ -pIOSchedulingClass=idle –uid=matrix-synapse –collect \ –unit=synapse-clear-state ~/bin/synapse-clear-state% journalctl -Stoday -u synapse-clear-state</pre> Das Programm hat nämlich die Macke und erzeugt ein größeres Ergebnis als der Ausgangszustand. Da auch die Prüfung auf die Korrektheit des neuen Zustands teilweise sehr lange dauert, kann man mit der Option

```
-m
```

bei synapse-compress-state die Mindestanzahl an Tabellenzeilen angeben, für die der neue Zustand genommen wird. Abgesehen von dieser Macke liefert das Programm aber teilweise eine Reduktion auf 10 % (sic!) bei einigen Räumen. Synapse muss während der Ausführung nicht angehalten werden, sodass es egal ist, wie lange das Programm läuft oder ob man es zwischendurch abbricht. Ich rufe es bei mir unregelmäßig alle paar Wochen auf. <h2 id=„„device_inboxaufrumen““>device_inbox aufräumen</h2> Die Schema-Definition der Datenbank von Synapse ist piep. Daher werden Einträge in der Tabelle device_inbox nicht gelöscht, wenn die dazugehörigen Geräte gelöscht werden. Mit folgendem SQL-Befehl kann man dies händisch erledigen: <pre>DELETE– SELECT COUNT(*)FROM device_inboxWHERE device_id IN (SELECT device_id FROM devices WHERE hidden = true) OR device_id NOT IN (SELECT device_id FROM devices) </html>

1)

„python“,pid=24088,fd=10