Matrix-Räume lassen sich weder löschen noch verlassen – rette Deine Instanz

Wenn in Matrix-Räumen gar nichts mehr funktioniert – keine Nachrichten mehr ankommen, das Verlassen des Raumes nicht möglich ist und stattdessen eine Fehlermeldung wie `MatrixError: [403] No create event in auth events` erscheint – dann ist meistens der lokale Raum-State beschädigt.

Der Synapse-Server erkennt das „m.room.create“-Event nicht mehr und verweigert dadurch sämtliche Aktionen. Hier zeige ich, wie man solche sogenannten Zombie-Räume loswird – egal ob mit Docker betrieben oder auf einem klassischen Server installiert. Genau das ist mir mit dem Support-Raum von matrix-docker-ansible-deploy passiert. Da meine Installation in Docker-Containern läuft, können die Befehle bei anderen Setups eventuell abweichen.

Symptome

In Clients wie Element werden diese Räume weiterhin als aktiv angezeigt, es kommen aber keine Nachrichten mehr an, und auch das Verlassen schlägt fehl. In den Logs findet sich dann eine Zeile wie:

```
MatrixError: [403] No create event in auth events
```

Das Problem liegt folglich nicht im Client, sondern im Synapse-Backend. Dem Server fehlt das m.room.create-Event, daher weiß er nicht mehr, dass der Raum existiert.

Wie finde ich die Raum-ID heraus?

Um die Raum-ID zu ermitteln, gibt es verschiedene Wege – abhängig davon, ob du Server-Zugang oder nur den Client verwendest.

- **Über den Synapse Admin-Client:** Öffne in Synapse Admin (Weboberfläche unter `https://deinserver/_synapse/`) den entsprechenden Raum. Die Raum-ID findest du in der Adresszeile der URL – sie beginnt immer mit einem Ausrufezeichen, etwa `!cNSQwPqhHKkIZdBnvt:devture.com`.
- Über die API: Du kannst aus dem Container heraus eine Suche nach Räumen durchführen:

  ```
      docker exec -i matrix-synapse \
        curl -sS
  "http://localhost:8008/_synapse/admin/v1/rooms?search_term=raumname" \
        -H "Authorization: Bearer " | jq
  ```

  Damit findest du die vollständige Room-ID auch ohne Client.

- Über den Matrix-Client (wie Element): Gehe in die Raumdetails und schau unter *„Erweiterte*

Informationen“ – dort wird die Room-ID ebenfalls angezeigt.

Die Raum-ID benötigst du für alle Befehle der Admin-API, da Synapse im Hintergrund nicht mit Aliasen (wie `#meinraum:matrixbox.de`), sondern ausschließlich mit der eindeutigen Room-ID arbeitet.

der Grund ist ein fehlerhafter Raum-State

Dies tritt typischerweise auf bei:

- unterbrochenen Föderations-Synchronisationen
- beschädigten Datenbankeinträgen
- veralteten Synapse-Versionen (unter 1.90), die State-Resets nicht korrekt verarbeiten

Auswirkung: Der Raum ist teilweise funktionslos. Sichtbar, aber nicht mehr nutzbar.

Lösung: Bereinigen über die Admin-API

Am einfachsten ist die Bereinigung über die Admin-API, nicht über das Webinterface. Den erforderlichen Admin-Token findet ihr im Element-Client unter Einstellungen » Hilfe und Info (Access Token des Administrators).

1. Raum per API entfernen

Synapse läuft normalerweise im Container mit dem Namen matrix-synapse. Steigt in den Container ein und führt den Löschbefehl aus:

```
docker exec -i matrix-synapse \
  curl -sS -X DELETE
"http://localhost:8008/_synapse/admin/v2/rooms/%21cNSQwPqhHKkIZdBnvt%3Adevtu
re.com" \
  -H "Authorization: Bearer " \
  -H "Content-Type: application/json" \
  -d '{"purge":true,"block":true,"force_purge":true}'
```

Dadurch wird der Raum endgültig aus der Datenbank entfernt und gleichzeitig gesperrt, sodass kein erneuter Beitritt möglich ist. Die API gibt eine `delete_id` zurück, mit der sich der Löschstatus prüfen lässt:

```
docker exec -i matrix-synapse \
  curl -sS "http://localhost:8008/_synapse/admin/v2/rooms/delete_status/" \
  -H "Authorization: Bearer "
```

Warte, bis der Status auf complete steht.

2. Client-Cache leeren

Im Element-Client: **Einstellungen → Hilfe → Cache leeren und neu laden**

Der Raum wird daraufhin aus der Übersicht entfernt.

Falls Beitritt gesperrt ist: Entsperren

Wenn du beim Löschen block:true gewählt hast (empfohlen), musst du die Sperre bei Bedarf manuell aufheben, bevor ein erneuter Beitritt möglich ist.

```
docker exec -i matrix-synapse \
  curl -sS -X PUT
"http://localhost:8008/_synapse/admin/v1/rooms/%21cNSQwPqhHKkIZdBnvt%3Adevtu
re.com/block" \
  -H "Authorization: Bearer " \
  -H "Content-Type: application/json" \
  -d '{"block": false}'
```

Anschließend kann der Raum wie gewohnt wieder beigetreten werden, beispielsweise so:

```
curl -sS -X POST \
"https://matrix.matrixbox.de/_matrix/client/v3/rooms/%21cNSQwPqhHKkIZdBnvt%3
Adevture.com/join?server_name=devture.com" \
  -H "Authorization: Bearer " \
  -d '{}'
```

Hinweis für nginx-Nutzende

Damit die Admin-APIs künftig auch außerhalb des Containers erreichbar sind, sollte nginx sowohl `/_matrix` als auch `/_synapse` an Synapse weiterleiten:

```
location ~ ^/(_matrix|_synapse/client) {
    proxy_pass http://synapse:8008;
    proxy_read_timeout 600s;
    proxy_send_timeout 600s;
    proxy_request_buffering off;
}
```

Danach einfach nginx -t && systemctl reload nginx ausführen.

Fazit

Wenn Räume bei Matrix hängen bleiben, liegt das fast immer an einem fehlerhaften Raum-State auf

dem Homeserver und selten am Client. Mittels Admin-API lässt sich das zuverlässig beheben – mit neueren Synapse-Versionen (ab 1.110) treten solche Probleme zudem wesentlich seltener auf.

2025/10/17 05:10 · marko

Longhorn - Distributed Block Storage System für Kubernetes

Longhorn ist eine container-native Speicherlösung für Kubernetes-Umgebungen. Als Open-Source-Projekt der Cloud Native Computing Foundation (CNCF) steht Longhorn frei zur Verfügung. Mit Longhorn kann in containerisierten Umgebungen persistenter Blockspeicher bereitgestellt werden.

2025/10/11 07:02 · marko

Kubernetes Cluster Logging mit Loki

Die kontinuierliche Überwachung eines Kubernetes-Clusters erfordert ein effizientes Logmanagement, welches das zentrale Sammeln, Speichern und Analysieren von Protokolldaten ermöglicht. In Kubernetes-Umgebungen wird hierfür traditionell ein Stack aus Logstash oder Fluentd zur Datenerfassung, Elasticsearch als Speichersystem sowie Kibana oder Graylog zur Analyse und Visualisierung eingesetzt. Neben diesem etablierten Ansatz gewinnt der Loki-Grafana-Stack zunehmend an Bedeutung, da er eine ressourcenschonendere und leichter implementierbare Alternative darstellt.

2025/09/04 12:03 · marko

So versetzen Sie einen Cloud Native PostgreSQL Operator Cluster in den Wartungsmodus

Um einen von Cloud Native PostgreSQL (CNPG) Operator verwalteten PostgreSQL-Cluster vollständig in den Wartungsmodus zu versetzen, gibt es keine einzelne, allumfassende Funktion. Stattdessen können je nach Wartungsanforderung zwei Hauptmethoden angewendet werden: die Aktivierung eines Wartungsfensters für Knoten oder das „Fencing“ des gesamten Clusters.

Methode 1: Verwendung des Knoten-Wartungsfensters (nodeMaintenanceWindow)

Diese Methode ist ideal für geplante Wartungsarbeiten an den Kubernetes-Knoten, auf denen die

PostgreSQL-Instanzen laufen. Sie informiert den Operator darüber, dass ein Knoten absichtlich außer Betrieb genommen wird, sodass der Operator keine automatischen Failover-Prozesse einleitet.

Anwendungsfall: Sie planen, die Kubernetes-Knoten zu aktualisieren oder andere Wartungsarbeiten auf der Infrastrukturebene durchzuführen.

Schritte:

1. Aktivieren Sie das Wartungsfenster für den Cluster. Dies geschieht durch Setzen von inProgress: true im nodeMaintenanceWindow-Abschnitt der Cluster-Konfiguration. Sie können auch festlegen, ob das persistente Volume (PVC) wiederverwendet werden soll (reusePVC).
   Mit kubectl patch:

   ```
   kubectl patch cluster <cluster-name> -n <namespace> --type='merge' -p
   '{"spec":{"nodeMaintenanceWindow":{"inProgress":true,"reusePVC":true}}}
   '
   ```

   YAML Beispiel:

   ```
   apiVersion: postgresql.cnpg.io/v1
   kind: Cluster
   metadata:
     name: <cluster-name>
     namespace: <namespace>
   spec:
     # ... andere Cluster-Spezifikationen
     nodeMaintenanceWindow:
       inProgress: true
       reusePVC: true # Empfohlen, um Datenverlust zu vermeiden
   ```

2. **Führen Sie die Wartungsarbeiten** an den Knoten durch (z. B. `kubectl drain <node-name>`). Der CNPG-Operator wird die Pods auf dem Knoten geordnet herunterfahren.
3. **Deaktivieren Sie das Wartungsfenster**, nachdem die Wartungsarbeiten abgeschlossen sind, um den normalen Betrieb wieder aufzunehmen.
   Mit `kubectl patch`**:**

   ```
   kubectl patch cluster <cluster-name> -n <namespace> --type='merge' -p
   '{"spec":{"nodeMaintenanceWindow":{"inProgress":false}}}'
   ```

Methode 2: Fencing des gesamten Clusters

„Fencing“ ist eine drastischere Maßnahme, die den PostgreSQL-Prozess auf einer oder allen Instanzen stoppt, ohne die Pods selbst zu löschen. Dies versetzt die Datenbank effektiv in einen Offline-Zustand und ist nützlich, um den gesamten Cluster für Untersuchungen oder andere kritische Eingriffe anzuhalten.

Anwendungsfall: Sie müssen den gesamten Datenbankbetrieb sofort unterbrechen, um

beispielsweise ein Problem zu diagnostizieren, ohne dass der Operator versucht, den Cluster zu „reparieren“.

Schritte:

1. **Fencen Sie alle Instanzen** des Clusters. Dies wird durch Hinzufügen einer Annotation zum Cluster-Objekt erreicht.
   Mit `kubectl annotate`**:**

   ```
   kubectl annotate cluster <cluster-name> -n <namespace>
   cnpg.io/fencedInstances='["*"]'
   ```

   Das `["*"]` ist ein Wildcard-Zeichen, das alle Instanzen des Clusters betrifft.

2. **Überprüfen Sie den Status.** Die PostgreSQL-Prozesse in den Pods werden beendet, aber die Pods laufen weiter. Anwendungen können keine Verbindung zur Datenbank herstellen.
3. **Heben Sie das Fencing auf**, um den normalen Betrieb wiederherzustellen.

   Mit kubectl annotate:

   ```
   kubectl annotate cluster <cluster-name> -n <namespace>
   cnpg.io/fencedInstances-
   ```

   Das `-` am Ende des Befehls entfernt die Annotation.

Wichtige Überlegungen

- **Anwendungsverbindungen:** Bevor Sie den Cluster in einen Wartungsmodus versetzen, sollten Sie idealerweise die Anwendungen, die auf die Datenbank zugreifen, herunterfahren oder skalieren. Dies verhindert unerwartete Fehler und Verbindungsabbrüche auf der Anwendungsseite.
- **Auswirkungen von Fencing:** Wenn der primäre Server gefenct wird, findet **kein Failover** statt. Der Cluster ist für Schreibvorgänge nicht verfügbar, bis das Fencing aufgehoben wird.
- `kubectl cnpg` **Plugin:** Der CNPG-Operator verfügt über ein `kubectl`-Plugin, das einige dieser Operationen vereinfachen kann. Zum Beispiel kann der Befehl `kubectl cnpg maintenance` verwendet werden, um das `nodeMaintenanceWindow` zu setzen.

Zusammenfassend lässt sich sagen, dass das `nodeMaintenanceWindow` für geplante Infrastrukturwartungen gedacht ist, während das Fencing des gesamten Clusters die beste Methode ist, um den Datenbankbetrieb vollständig und sofort zu stoppen.

2025/08/10 16:15 · marko

Automatisiere Deine Git-Commit-Nachrichten mit ChatGPT

Das Erstellen aussagekräftiger und prägnanter Commit-Nachrichten ist ein wesentlicher Bestandteil eines guten Entwicklungsworkflows. Diese Nachrichten helfen dabei, Änderungen zu verfolgen, den Projektverlauf zu verstehen und mit Teammitgliedern zusammenzuarbeiten. Zugegeben: Das Schreiben von Commit-Nachrichten kann manchmal eine banale Aufgabe sein. In diesem Artikel zeigen ich Dir, wie Du mit ChatGPT von OpenAI automatisch Git-Commit-Nachrichten generieren lassen kannst.

Das Skript

```
#!/bin/bash

check_git_repo() {
    if ! git rev-parse --is-inside-work-tree >/dev/null 2>&1; then
        exit 1
    fi
}

check_changes() {
    if [ -z "$(git status --porcelain)" ]; then
        exit 0
    fi
}

generate_commit_message() {
    local diff_content=$(git diff --cached)
    local files_changed=$(git status --porcelain)

    echo -e "Files changed:\n$files_changed\n\nChanges:\n$diff_content" | \
        llm -m anthropic/claude-3-5-sonnet-latest \
        "Generate a git commit message for these changes. The message must
have:

        1. TITLE LINE: A specific, concise summary (max 50 chars) begin
without special characters
           that clearly describes the primary change or feature. This should
not be generic like
           'Update files' but rather describe the actual change like 'Add
user
           authentication to API endpoints'

        2. BLANK LINE

        3. DETAILED DESCRIPTION: A thorough explanation including:
```

```
            - What changes were made
            - Why they were necessary
            - Any important technical details
            - Breaking changes or important notes
            Wrap this at 72 chars.

        IMPORTANT:
        - Output ONLY the commit message
        - Make sure the title is specific to these changes
        - Focus on the what and why, not just the how"
}

# Main execution
main() {
    check_git_repo
    check_changes
    git add --all
    commit_message=$(generate_commit_message)
    git commit -m "$commit_message"
}

main "$@"
```

Aufschlüsselung

Repository-Validierung

```
check_git_repo() {
    if ! git rev-parse --is-inside-work-tree >/dev/null 2>&1; then
        exit 1
    fi
}
```

Diese Funktion stellt sicher, dass wir in einem Git-Repository arbeiten.

Änderungserkennung

```
check_changes() {
    if [ -z "$(git status --porcelain)" ]; then
        exit 0
    fi
}
```

Überprüft, ob tatsächlich Änderungen zum Festschreiben vorhanden sind.

KI-gestützte Nachrichtengenerierung

```
generate_commit_message() {
    local diff_content=$(git diff --cached)
    local files_changed=$(git status --porcelain)

    echo -e "Files changed:\n$files_changed\n\nChanges:\n$diff_content" | \
        llm -m anthropic/claude-3-5-sonnet-latest \
        "Generate a git commit message for these changes. The message must
have:

        1. TITLE LINE: A specific, concise summary (max 50 chars) that
clearly
           describes the primary change or feature. This should not be
generic like
           'Update files' but rather describe the actual change like 'Add
user
           authentication to API endpoints'

        2. BLANK LINE

        3. DETAILED DESCRIPTION: A thorough explanation including:
           - What changes were made
           - Why they were necessary
           - Any important technical details
           - Breaking changes or important notes
           Wrap this at 72 chars.

        IMPORTANT:
        - Output ONLY the commit message
        - Make sure the title is specific to these changes
        - Focus on the what and why, not just the how"
}
```

Hier geschieht die Magie – das Skript analysiert Deine Änderungen und verwendet KI, um eine aussagekräftige Commit-Nachricht zu generieren.

Das Skript verwendet Simon Willisons Kommandozeilentool llm , ein äußerst nützliches Dienstprogramm für die Interaktion mit verschiedenen KI-Modellen direkt von Deinem Terminal aus. Weitere Informationen dazu, wie Du es einrichtest und tatsächlich verwendest, findest Du in seiner Dokumentation.

Bitte beachte, dass ich in diesem Skript das Modell von Anthropic verwende. Das bedeutet, dass Du das Plugin „llm-anthropic“ einrichten musst.

Einrichten

Um das Skript am Ende auch auszuführen zu können, erstellst Du einfach eine ausführbare Commit-Datei und fügst diese in Dein Bin-Verzeichnis hinzu, sodass sie in Deinem PATH landet.

Vergiss nicht:

```
chmod +x ~/.local/bin/commit
```

um das Skript ausführbar zu machen. Ändere einfach den Pfad zum Skript, je nachdem, wo Du es speichern möchtest.

Juhuu

Nachdem Du hart an Deinem Code gearbeitet hast, musst Du ihn nur noch `**commit**` ausführen und erhälst eine von der KI generierte Commit-Nachricht.

2025/06/21 09:43 · marko

Ältere Einträge >>