Witnz for PostgreSQL

分散型データベース改竄検知システム 技術設計書

1. 概要

witnz は、既存の PostgreSQL データベースに対して、アプリケーションサーバーを活用した分散型の改竄検知機能を提供するシステムである。ブロックチェーン技術の考え方を応用しつつ、軽量で導入しやすい設計を目指す。

1.1 解決する課題

- データベース管理者による内部不正の検知が困難
- RDS への直接攻撃時に改竄を検知する手段がない
- 既存の改竄防止ソリューション（Hyperledger 等）は導入が重い
- 監査要件（SOC2, ISO27001）への対応に手間がかかる

1.2 アプローチ

既存のアプリケーションサーバーに軽量ノードをサイドカーとして配置し、保護対象テーブルのデータハッシュを分散管理する。RDS が改竄されても、アプリサーバー群が「正しい状態」を保持しているため、検知・復旧が可能となる。

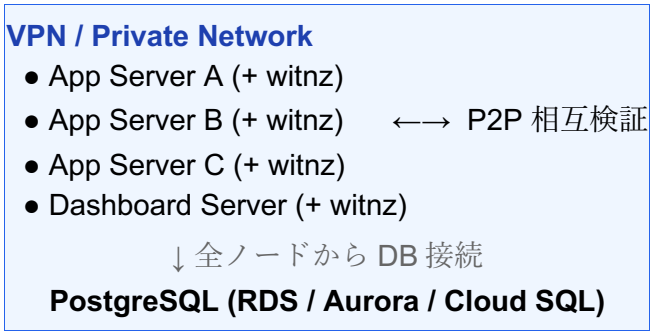
2. システムアーキテクチャ

2.1 構成要素

コンポーネント	説明
軽量ノード	各アプリサーバーにインストールする Go 製バイナリ。ハッシュチェーンの保持と相互検証を行う。
ダッシュボード	管理 UI。ノードの一つとしてコンセンサスに参加しつつ、状態監視とアラート管理を提供。
PostgreSQL	既存のデータベース。RDS、Aurora、Cloud SQL、Supabase 等に対応。変更なしで利用可能。

2.2 ネットワーク構成

VPN またはプライベートネットワーク内に閉じた構成を想定。



3. 保護モード

テーブルの性質に応じて 2 つの保護モードを提供する。

3.1 append_only モード（履歴・監査系）

用途: 変更ログ、監査証跡、契約履歴、同意履歴

保証: 過去のレコードが改竄されていないこと

動作:

- 新規 INSERT 時にハッシュを計算し、チェーンに追加
- UPDATE/DELETE が発生した場合は即アラート
- 各ノードがチェーンを保持し、整合性を相互検証

3.2 state_integrity モード（マスタ・設定系）

用途: 権限テーブル、料金テーブル、ライセンス情報

保証: 現在の値が正しい（改竄されていない）こと

動作:

- テーブル全体の Merkle Root を定期的に計算
- 各ノード間で照合し、不一致があれば検知
- どのレコードが変更されたかも特定可能

4. 技術スタック

レイヤー	技術	理由
軽量ノード	Go	シングルバイナリ、クロスコンパイル容易
DB 変更検知	Logical Replication / CDC	PostgreSQL 標準機能、低オーバーヘッド
コンセンサス	Raft	軽量、実装が容易、十分な耐障害性
ローカルストレージ	SQLite / BoltDB	組み込み可能、追加インフラ不要
ハッシュ構造	Merkle Tree	効率的な整合性検証、差分特定が可能
ダッシュボード	React + Go API	モダン UI、ノードと同一バイナリに統合可能

5. データフロー

5.1 書き込み時のフロー

1. アプリケーションが PostgreSQL に INSERT/UPDATE/DELETE を実行
2. Logical Replication により変更がローカルノードに伝播
3. ノードがデータのハッシュを計算
4. Raft コンセンサスにより他ノードに伝播・合意
5. 各ノードがローカルストレージにハッシュチェーンを保存

5.2 検証時のフロー

1. 定期的またはオンデマンドで検証を実行
2. PostgreSQL から対象データを取得し、ハッシュを計算
3. 各ノードが保持するハッシュと照合
4. 不一致があれば、多数決で「正しい状態」を特定
5. アラート発報または自動修復（設定による）

6. セキュリティ考慮事項

6.1 攻撃シナリオと対策

攻撃シナリオ	リスク	対策
RDS のみ改竄	低	ノード群のハッシュと不一致で即検知
単一ノード侵害	低	他ノードとの多数決で検知
全ノード同時侵害	中	外部アンカーで対応（後述）
秘密鍵漏洩	中	HSM 利用、鍵ローテーション
長期潜伏攻撃	高	外部アンカーで時点証明

6.2 外部アンカー（オプション）

VPN 内だけで完結する基本構成に加え、より強固な証明が必要な場合は外部アンカーを利用：

レベル	方式	用途
Basic	VPN 内ノード間検証のみ	内部不正の抑止
Standard	S3 Object Lock に定期保存	監査対応、時点証明
Strict	パブリックチェーンにアンカリング	法的証拠能力が必要な場合

7. 設定例

YAML 形式での設定ファイル例：

```yaml
# witnz.yaml
database:
  host: your-rds-endpoint.amazonaws.com
  port: 5432
  name: myapp_production

protected_tables:
  - table: audit_logs
    mode: append_only
  - table: user_permissions
    mode: state_integrity
  - table: pricing_plans
    mode: state_integrity
    alert: slack

nodes:
  - host: app-server-1.internal
  - host: app-server-2.internal
  - host: dashboard.internal
```

8. 導入手順

1. 各アプリサーバーに witnz バイナリを配置
2. PostgreSQL の Logical Replication を有効化
3. witnz.yaml で保護対象テーブルを指定
4. 各ノードで tmpr start を実行
5. ダッシュボードで状態を確認

```
# インストール
curl -sSL https://witnz.tech/install.sh | sh

# 初期化
witnz init --db postgres://user:pass@rds/myapp

# 保護対象を指定して起動
witnz protect user_permissions pricing_plans
witnz start
```

9. 競合との差別化

ソリューション	課題	witnz の優位性
Hyperledger Fabric	導入が重い、専門知識必要	バイナリ 1 つで導入可能
immudb	別 DB への移行が必要	既存 PostgreSQL をそのまま利用
Amazon QLDB	AWS ロックイン、分散検証なし	クラウド非依存、分散検証あり
pgaudit + S3	分散検証がない	複数ノードで相互検証
ScalarDL	エンタープライズ向け、高コスト	OSS、スタートアップでも利用可能

10. 今後の開発計画

Phase 1: MVP

- Go による軽量ノードの実装
- PostgreSQL Logical Replication との連携
- 基本的なハッシュチェーン機能
- CLI による状態確認
- Raft コンセンサスの実装

Phase 2: コア機能

- ロギング
- Raft リトライ機能
- API 統合
- ダッシュボード UI
- アラート連携（Slack, PagerDuty）

Phase 3: エンタープライズ機能

- gRPC API サーバー
- 外部アンカー連携
- 監査レポート出力
- SSO / SAML 対応
- マルチ環境管理