# 🚀 BigBird完全アーキテクチャ設計 - 2025年版

## 📋 BigBird全機能チェックリスト（BigQuery互換）

### 🏗️ 1. データ定義言語（DDL）

#### テーブル操作

- [ ] **CREATE TABLE** - テーブル作成
- [ ] 標準テーブル作成
- [ ] パーティション分割テーブル（日付、整数、列範囲）
- [ ] クラスター化テーブル
- [ ] 外部テーブル（Cloud Storage、Google Drive）
- [ ] CREATE TABLE IF NOT EXISTS
- [ ] CREATE OR REPLACE TABLE
- [ ] CREATE TABLE AS SELECT (CTAS)
- [ ] CREATE TABLE LIKE（既存テーブル構造コピー）
- [ ] 主キー制約（非強制）
- [ ] 外部キー制約（非強制）
- [ ] **ALTER TABLE** - テーブル変更
- [ ] ADD COLUMN（列追加）
- [ ] DROP COLUMN（列削除）
- [ ] RENAME COLUMN（列名変更）
- [ ] ALTER COLUMN SET DATA TYPE（型変更）
- [ ] SET OPTIONS（テーブルオプション設定）
- [ ] RENAME TO（テーブル名変更）
- [ ] **DROP TABLE** - テーブル削除
- [ ] DROP TABLE IF EXISTS

#### ビュー操作

- [ ] **CREATE VIEW** - ビュー作成
- [ ] 標準ビュー
- [ ] マテリアライズドビュー
- [ ] CREATE OR REPLACE VIEW
- [ ] ビューオプション（有効期限、ラベル等）
- [ ] **ALTER VIEW** - ビュー変更
- [ ] **DROP VIEW** - ビュー削除

#### データセット操作

- [ ] **CREATE SCHEMA/DATASET** - データセット作成
- [ ] **ALTER SCHEMA/DATASET** - データセット変更

- [ ] **DROP SCHEMA/DATASET** - データセット削除

#### 関数操作

- [ ] **CREATE FUNCTION** - UDF作成
- [ ] SQL UDF
- [ ] JavaScript UDF
- [ ] 永続UDF
- [ ] 一時UDF
- [ ] リモート関数
- [ ] **ALTER FUNCTION** - 関数変更
- [ ] **DROP FUNCTION** - 関数削除

#### プロシージャ操作

- [ ] **CREATE PROCEDURE** - ストアドプロシージャ作成
- [ ] **ALTER PROCEDURE** - プロシージャ変更
- [ ] **DROP PROCEDURE** - プロシージャ削除

### 🔄 2. データ操作言語（DML）

- [ ] **INSERT** - データ挿入
- [ ] INSERT INTO VALUES
- [ ] INSERT INTO SELECT
- [ ] 複数行一括挿入
- [ ] **UPDATE** - データ更新
- [ ] 条件付き更新
- [ ] JOIN を使った更新
- [ ] サブクエリを使った更新
- [ ] **DELETE** - データ削除
- [ ] 条件付き削除
- [ ] 全行削除（WHERE必須）
- [ ] パーティション削除
- [ ] **MERGE** - データマージ（UPSERT）
- [ ] INSERT、UPDATE、DELETE の組み合わせ
- [ ] 条件付きマージ
- [ ] **TRUNCATE TABLE** - テーブルデータ全削除

### 🔍 3. データクエリ言語（DQL）

#### 基本クエリ

- [ ] **SELECT** - データ選択
- [ ] SELECT *（全列選択）

- [ ] 特定列選択
- [ ] DISTINCT（重複除去）
- [ ] ALL（全データ）
- [ ] AS STRUCT/VALUE（構造体/値として）

### フィルタリング・条件

- [ ] **WHERE** - 条件フィルタリング
- [ ] **HAVING** - 集計後フィルタリング
- [ ] **QUALIFY** - ウィンドウ関数後フィルタリング

### 結合操作

- [ ] **JOIN**
- [ ] INNER JOIN
- [ ] LEFT JOIN / LEFT OUTER JOIN
- [ ] RIGHT JOIN / RIGHT OUTER JOIN
- [ ] FULL JOIN / FULL OUTER JOIN
- [ ] CROSS JOIN
- [ ] 複数テーブル結合

### 集計・グループ化

- [ ] **GROUP BY** - グループ化
- [ ] 単一列グループ化
- [ ] 複数列グループ化
- [ ] ROLLUP（集計ロールアップ）
- [ ] CUBE（キューブ集計）
- [ ] GROUPING SETS（グループセット）

### ソート・制限

- [ ] **ORDER BY** - ソート
- [ ] ASC（昇順）
- [ ] DESC（降順）
- [ ] NULLS FIRST/LAST
- [ ] **LIMIT** - 行数制限
- [ ] **OFFSET** - オフセット

### 集合演算

- [ ] **UNION / UNION ALL** - 和集合
- [ ] **INTERSECT** - 積集合
- [ ] **EXCEPT** - 差集合

### 高度なクエリ構造

- [ ] **WITH** - CTE（共通テーブル式）
- [ ] 非再帰CTE
- [ ] WITH RECURSIVE（再帰CTE）
- [ ] 複数CTE定義
- [ ] **PIVOT** - 行を列に変換
- [ ] **UNPIVOT** - 列を行に変換
- [ ] **TABLESAMPLE** - サンプリング
- [ ] **FOR SYSTEM_TIME AS OF** - 時点クエリ

### 配列・構造体操作

- [ ] **UNNEST** - 配列展開
- [ ] **ARRAY** - 配列生成
- [ ] **STRUCT** - 構造体生成

## 🎯 4. 関数群

### 集計関数

- [ ] COUNT / COUNT(DISTINCT)
- [ ] SUM / AVG
- [ ] MIN / MAX
- [ ] STDDEV / VARIANCE
- [ ] CORR / COVAR_POP / COVAR_SAMP
- [ ] PERCENTILE_CONT / PERCENTILE_DISC
- [ ] ARRAY_AGG
- [ ] STRING_AGG
- [ ] ANY_VALUE

### ウィンドウ関数

- [ ] ROW_NUMBER()
- [ ] RANK() / DENSE_RANK()
- [ ] PERCENT_RANK() / CUME_DIST()
- [ ] NTILE(n)
- [ ] LAG() / LEAD()
- [ ] FIRST_VALUE() / LAST_VALUE()
- [ ] NTH_VALUE()

### 文字列関数

- [ ] CONCAT / CONCAT_WS
- [ ] SUBSTR / SUBSTRING

- [ ] LENGTH / CHAR_LENGTH
- [ ] UPPER / LOWER
- [ ] LTRIM / RTRIM / TRIM
- [ ] REPLACE / REGEXP_REPLACE
- [ ] SPLIT / ARRAY_TO_STRING
- [ ] STARTS_WITH / ENDS_WITH
- [ ] CONTAINS_SUBSTR
- [ ] FORMAT
- [ ] PARSE_URL
- [ ] ASCII / CHR
- [ ] CODE_POINTS_TO_STRING / TO_CODE_POINTS

**数学関数**

- [ ] ABS / SIGN
- [ ] ROUND / CEIL / FLOOR
- [ ] SQRT / POW
- [ ] EXP / LN / LOG
- [ ] SIN / COS / TAN
- [ ] ASIN / ACOS / ATAN / ATAN2
- [ ] RAND / GENERATE_ARRAY
- [ ] MOD / DIV
- [ ] GREATEST / LEAST

**日付・時刻関数**

- [ ] CURRENT_DATE / CURRENT_DATETIME / CURRENT_TIMESTAMP
- [ ] DATE / DATETIME / TIMESTAMP
- [ ] EXTRACT - 日付部分抽出
- [ ] DATE_ADD / DATE_SUB
- [ ] DATETIME_ADD / DATETIME_SUB
- [ ] TIMESTAMP_ADD / TIMESTAMP_SUB
- [ ] DATE_DIFF / DATETIME_DIFF / TIMESTAMP_DIFF
- [ ] FORMAT_DATE / FORMAT_DATETIME / FORMAT_TIMESTAMP
- [ ] PARSE_DATE / PARSE_DATETIME / PARSE_TIMESTAMP
- [ ] DATE_TRUNC / DATETIME_TRUNC / TIMESTAMP_TRUNC
- [ ] GENERATE_DATE_ARRAY / GENERATE_TIMESTAMP_ARRAY

**配列関数**

- [ ] ARRAY - 配列作成
- [ ] ARRAY_CONCAT - 配列結合
- [ ] ARRAY_LENGTH - 配列長さ

- [ ] ARRAY_REVERSE - 配列反転
- [ ] ARRAY_SLICE - 配列切り出し
- [ ] ARRAY_TO_STRING - 配列→文字列
- [ ] GENERATE_ARRAY - 配列生成
- [ ] ARRAY_AGG - 配列集計
- [ ] UNNEST - 配列展開
- [ ] OFFSET / ORDINAL - 配列インデックス

**JSON関数**

- [ ] JSON_EXTRACT - JSON値抽出
- [ ] JSON_EXTRACT_SCALAR - JSONスカラー抽出
- [ ] JSON_EXTRACT_ARRAY - JSON配列抽出
- [ ] JSON_EXTRACT_STRING_ARRAY - JSON文字列配列抽出
- [ ] JSON_QUERY - JSONクエリ
- [ ] JSON_VALUE - JSON値取得
- [ ] JSON_ARRAY - JSON配列作成
- [ ] JSON_OBJECT - JSONオブジェクト作成
- [ ] PARSE_JSON - JSON解析
- [ ] TO_JSON_STRING - JSON文字列変換

**型変換関数**

- [ ] CAST - 型変換
- [ ] SAFE_CAST - 安全な型変換
- [ ] COALESCE - NULL代替
- [ ] IFNULL / NULLIF
- [ ] ISNULL

**条件関数**

- [ ] IF - 条件分岐
- [ ] CASE WHEN - 条件分岐
- [ ] COUNTIF - 条件付きカウント
- [ ] SUMIF - 条件付き合計

**暗号化関数**

- [ ] MD5 / SHA1 / SHA256 / SHA512
- [ ] TO_HEX / FROM_HEX

**統計関数**

- [ ] APPROX_COUNT_DISTINCT
- [ ] APPROX_QUANTILES

- [ ] APPROX_TOP_COUNT
- [ ] STDDEV_POP / STDDEV_SAMP
- [ ] VAR_POP / VAR_SAMP

### 🔐 5. データ制御言語（DCL）

- [ ] **GRANT** - 権限付与
- [ ] **REVOKE** - 権限剥奪

### 🔄 6. トランザクション制御言語（TCL）

- [ ] **BEGIN / START TRANSACTION**
- [ ] **COMMIT**
- [ ] **ROLLBACK**
- [ ] **SAVEPOINT**

### 🌐 7. 分散・クラスター機能

#### クラスター管理

- [ ] **AUTO-DISCOVERY** - 自動ノード発見
- [ ] **AUTO-BOOTSTRAP** - 起動時自動ネットワーク参加
- [ ] **NETWORK SCANNING** - 自動ネットワークスキャン
- [ ] **PEER DISCOVERY** - 自動ピア発見
- [ ] **ZERO-CONFIG JOIN** - ゼロ設定自動参加
- [ ] **CLUSTER JOIN/LEAVE** - 動的クラスター参加・離脱
- [ ] **HEALTH CHECK** - ノード健全性監視
- [ ] **LEADER ELECTION** - リーダー選出
- [ ] **SPLIT-BRAIN PREVENTION** - 脳分裂防止

#### データ分散・複製

- [ ] **SHARDING** - 水平分割
- [ ] **REPLICATION** - データ複製（Master-Slave/Master-Master）
- [ ] **CONSISTENCY LEVELS** - 一貫性レベル（強/弱/最終的）
- [ ] **READ REPLICAS** - 読み取り専用レプリカ
- [ ] **CROSS-REGION SYNC** - 地域間同期

#### 負荷分散・ルーティング

- [ ] **QUERY ROUTING** - クエリルーティング
- [ ] **CONNECTION POOLING** - 接続プール
- [ ] **LOAD BALANCING** - 負荷分散
- [ ] **CIRCUIT BREAKER** - サーキットブレーカー
- [ ] **RATE LIMITING** - レート制限

### 障害対応・復旧

- [ ] **AUTO-FAILOVER** - 自動フェイルオーバー
- [ ] **BACKUP/RESTORE** - バックアップ・復元
- [ ] **POINT-IN-TIME RECOVERY** - 時点復旧
- [ ] **DISASTER RECOVERY** - 災害復旧
- [ ] **ROLLING UPDATES** - ローリングアップデート

## 🛡️ 8. セキュリティ・SQLインジェクション対策

### アクセス制御（独自実装）

- [ ] **CLI-ONLY DIRECT ACCESS** - データベース直接アクセスはCLIのみ
- [ ] **API-ONLY EXTERNAL ACCESS** - 外部アクセスは100%API経由のみ
- [ ] **BIGBIRD API GATEWAY** - 独自APIゲートウェイ強制通過
- [ ] **NO DIRECT DB CONNECTION** - データベース直接接続完全禁止
- [ ] **SECURE API ENDPOINTS** - セキュアAPIエンドポイントのみ
- [ ] **API KEY AUTHENTICATION** - APIキー認証必須

### 独自超強力SSL/TLS実装

- [ ] **BIGBIRD SSL/TLS** - 解読不可能な独自TLSプロトコル
- [ ] **QUANTUM-RESISTANT CRYPTO** - 量子コンピューター耐性暗号
- [ ] **DYNAMIC KEY ROTATION** - 動的キーローテーション（秒単位）
- [ ] **PERFECT FORWARD SECRECY** - 完全前方秘匿性
- [ ] **CUSTOM CIPHER SUITES** - 独自暗号化スイート
- [ ] **ANTI-QUANTUM ALGORITHMS** - 耐量子アルゴリズム

### 入力値検証・サニタイゼーション

- [ ] **PREPARED STATEMENTS** - パラメータ化クエリ強制
- [ ] **INPUT VALIDATION** - 厳格な入力値検証
- [ ] **SQL SANITIZATION** - SQLクエリサニタイズ
- [ ] **ESCAPE PROCESSING** - 特殊文字エスケープ処理
- [ ] **TYPE VALIDATION** - データ型厳密検証
- [ ] **LENGTH VALIDATION** - 入力長制限
- [ ] **API-LEVEL VALIDATION** - API層での完全検証

### クエリ解析・防御

- [ ] **SECURITY PARSER** - セキュリティ特化SQLパーサー
- [ ] **MALICIOUS PATTERN DETECTION** - 悪意あるパターン検出
- [ ] **KEYWORD BLACKLIST** - 危険キーワードブラックリスト
- [ ] **WHITELIST VALIDATION** - 許可リストベース検証
- [ ] **NESTED QUERY ANALYSIS** - ネストクエリ解析

- [ ] **UNION ATTACK PREVENTION** - UNION攻撃防止
- [ ] **API QUERY FILTERING** - API層クエリフィルタリング

#### Bastion WAF（独自実装）

- [ ] **REAL-TIME BLOCKING** - リアルタイム攻撃ブロック
- [ ] **RATE LIMITING** - レート制限・DDoS防止
- [ ] **IP REPUTATION** - IPレピュテーション評価
- [ ] **GEO-BLOCKING** - 地域ベースアクセス制御
- [ ] **SIGNATURE-BASED DETECTION** - シグネチャベース検出
- [ ] **BEHAVIOR ANALYSIS** - 行動分析異常検知
- [ ] **API ATTACK PREVENTION** - API特化攻撃防止

#### 監査・ログ・アラート

- [ ] **COMPREHENSIVE AUDIT LOG** - 包括的監査ログ
- [ ] **REAL-TIME ALERTING** - リアルタイムアラート
- [ ] **FORENSIC ANALYSIS** - フォレンジック分析
- [ ] **THREAT INTELLIGENCE** - 脅威インテリジェンス統合
- [ ] **COMPLIANCE REPORTING** - コンプライアンスレポート
- [ ] **SECURITY DASHBOARD** - セキュリティダッシュボード
- [ ] **API ACCESS LOGGING** - API アクセス完全ログ

#### 権限・アクセス制御

- [ ] **PRINCIPLE OF LEAST PRIVILEGE** - 最小権限の原則
- [ ] **ROLE-BASED ACCESS CONTROL** - ロールベースアクセス制御
- [ ] **DYNAMIC PERMISSIONS** - 動的権限管理
- [ ] **SESSION MANAGEMENT** - セッション管理
- [ ] **MULTI-FACTOR AUTHENTICATION** - 多要素認証
- [ ] **API KEY ROTATION** - APIキー自動ローテーション
- [ ] **CLI-ONLY ADMIN ACCESS** - 管理者アクセスはCLIのみ

### 🎛️ 9. 高度な機能

#### プロシージャル言語

- [ ] **DECLARE** - 変数宣言
- [ ] **SET** - 変数設定
- [ ] **IF...THEN...ELSE** - 条件分岐
- [ ] **WHILE / LOOP** - ループ
- [ ] **FOR...IN** - イテレーション
- [ ] **BREAK / CONTINUE** - ループ制御
- [ ] **CALL** - プロシージャ呼び出し

- [ ] **RETURN** - 戻り値

#### エラーハンドリング

- [ ] **RAISE** - エラー発生
- [ ] **EXCEPTION** - 例外処理

#### 機械学習（BigQuery ML）

- [ ] **CREATE MODEL** - モデル作成
- [ ] **ML.PREDICT** - 予測
- [ ] **ML.EVALUATE** - モデル評価
- [ ] **ML.FEATURE_IMPORTANCE** - 特徴量重要度
- [ ] **ML.EXPLAIN_PREDICT** - 予測説明

#### 地理空間分析

- [ ] **ST_GEOGPOINT** - 地理ポイント
- [ ] **ST_DISTANCE** - 距離計算
- [ ] **ST_AREA** - 面積計算
- [ ] **ST_INTERSECTS** - 交差判定
- [ ] **ST_CONTAINS** - 包含判定

#### 時系列分析

- [ ] **PARSE_TIMEZONE** - タイムゾーン解析
- [ ] **CONVERT_TIMEZONE** - タイムゾーン変換

#### その他の高度な機能

- [ ] **ASSERT** - アサーション
- [ ] **EXPORT DATA** - データエクスポート
- [ ] **LOAD DATA** - データロード

---

## 🏗️ 実装アーキテクチャ設計

### 📦 1. コア・アーキテクチャ・コンポーネント

```
BigBird/
├── cmd/
│   ├── bigbird-server/        # メインサーバー実行可能ファイル
│   ├── bigbird-cli/           # CLIクライアント
│   ├── guardian-proxy/        # Guardian Proxy サーバー
│   ├── maestro-orchestrator/  # Maestro Orchestrator
│   └── bigbird-tools/         # 管理ツール群
├── core/
│   ├── phoenix/
│   │   ├── engine.go              # Phoenix Engine本体
│   │   ├── memory_manager.go      # メモリ管理
│   │   ├── execution_engine.go    # 実行エンジン
│   │   └── performance_tuner.go   # パフォーマンスチューナー
│   ├── talon/
│   │   ├── sql_parser.go          # Talon SQLパーサー
│   │   ├── lexer.go               # 字句解析
│   │   ├── ast.go                 # 抽象構文木
│   │   └── syntax_validator.go    # 構文検証
│   ├── optimizer/
│   │   ├── query_optimizer.go     # Query Optimizer
│   │   ├── query_planner.go       # クエリプランナー
│   │   ├── cost_estimator.go      # コスト推定
│   │   └── plan_selector.go       # プラン選択
│   ├── storage/
│   │   ├── feather_format.go      # FeatherFormat (.bbf)
│   │   ├── swift_index.go         # SwiftIndex (.bbi)
│   │   ├── flight_log.go          # FlightLog (.bbl)
│   │   ├── talon_compress.go      # TalonCompress圧縮
│   │   ├── iron_vault.go          # IronVault暗号化
│   │   ├── storage_engine.go      # ストレージエンジン
│   │   ├── page_manager.go        # ページ管理
│   │   └── buffer_pool.go         # バッファプール
│   ├── cluster/
│   │   ├── harmony_protocol.go    # Harmony Protocol合意
│   │   ├── skymesh_protocol.go    # SkyMesh通信プロトコル
│   │   ├── eagleeye_discovery.go  # EagleEye 自動ノード発見
│   │   ├── bigbird_autonet.go     # BigBird AutoNet 自動P2Pメッシュ
│   │   ├── whisperwave.go         # WhisperWave ゴシップ
│   │   ├── auto_bootstrap.go      # 自動ブートストラップ
│   │   ├── network_scanner.go     # ネットワーク自動スキャン
│   │   ├── peer_discovery.go      # 自動ピア発見
│   │   ├── zero_config_join.go    # ゼロ設定自動参加
│   │   ├── node_manager.go        # ノード管理
│   │   ├── leader_election.go     # リーダー選出
│   │   └── membership.go          # メンバーシップ管理
│   ├── maestro/
```

```
│   │   ├── orchestrator.go        # Maestro Orchestrator
│   │   ├── featherbox_runtime.go  # FeatherBox Runtime
│   │   ├── beacon_discovery.go    # Beacon Discovery
│   │   ├── config_nest.go         # ConfigNest設定管理
│   │   ├── pulse_monitor.go       # PulseMonitor ヘルス監視
│   │   ├── scheduler.go           # スケジューラー
│   │   ├── resource_manager.go    # リソース管理
│   │   └── auto_scaler.go         # オートスケーラー
│   ├── replication/
│   │   ├── replication_manager.go # 複製管理
│   │   ├── log_replication.go     # ログ複製
│   │   ├── state_machine.go       # ステートマシン
│   │   ├── snapshot.go            # スナップショット
│   │   └── conflict_resolution.go # 競合解決
│   ├── sharding/
│   │   ├── shard_manager.go       # シャード管理
│   │   ├── consistent_hash.go     # 一貫性ハッシュ
│   │   ├── data_distribution.go   # データ分散
│   │   ├── rebalancer.go          # リバランサー
│   │   └── partition_key.go       # パーティションキー
│   ├── guardian/
│   │   ├── proxy.go               # Guardian Proxy
│   │   ├── turbo_balance.go       # TurboBalance ロードバランサー
│   │   ├── circuit_guard.go       # CircuitGuard サーキットブレーカー
│   │   ├── flow_control.go        # FlowControl レート制限
│   │   ├── connection_pool.go     # 接続プール
│   │   └── failover.go            # フェイルオーバー
│   ├── network/
│   │   ├── swiftrpc_protocol.go   # SwiftRPC Protocol
│   │   ├── binary_stream.go       # BinaryStream プロトコル
│   │   ├── cloud_queue.go         # CloudQueue メッセージキュー
│   │   ├── packet_manager.go      # パケット管理
│   │   └── connection_manager.go  # 接続管理
│   ├── security/
│   │   ├── bigbird_ssl.go         # BigBird SSL/TLS
│   │   ├── trustforge_ca.go       # TrustForge CA
│   │   ├── vault_keeper.go        # VaultKeeper キー管理
│   │   ├── bastion_waf.go         # Bastion WAF
│   │   ├── sentinel_ids.go        # SentinelIDS 侵入検知
│   │   ├── crypto_engine.go       # 暗号化エンジン
│   │   ├── threat_detector.go     # 脅威検知エンジン
│   │   └── audit_logger.go        # セキュリティ監査ログ
│   ├── functions/
│   │   ├── aggregate_functions.go # 集計関数
│   │   ├── window_functions.go    # ウィンドウ関数
│   │   ├── string_functions.go    # 文字列関数
│   │   ├── math_functions.go      # 数学関数
```

```
│   │   ├── date_functions.go      # 日付関数
│   │   ├── array_functions.go     # 配列関数
│   │   ├── json_functions.go      # JSON関数
│   │   └── user_defined_functions.go # UDF
│   ├── transaction/
│   │   ├── distributed_tx.go      # 分散トランザクション
│   │   ├── two_phase_commit.go    # 2フェーズコミット
│   │   ├── transaction_manager.go # トランザクション管理
│   │   ├── lock_manager.go        # 分散ロック管理
│   │   └── mvcc.go                # 分散MVCC
│   └── backup/
│       ├── backup_manager.go      # バックアップ管理
│       ├── incremental_backup.go  # 増分バックアップ
│       ├── point_in_time.go       # 時点復旧
│       ├── cross_region_backup.go # 地域間バックアップ
│       └── disaster_recovery.go   # 災害復旧
├── api/
│   ├── portal/
│   │   ├── gateway.go             # Portal Gateway
│   │   ├── auth_middleware.go     # 認証ミドルウェア
│   │   ├── rate_limit_middleware.go # レート制限ミドルウェア
│   │   └── security_middleware.go # セキュリティミドルウェア
│   ├── handlers/
│   │   ├── query_handler.go       # クエリハンドラー
│   │   ├── admin_handler.go       # 管理ハンドラー
│   │   ├── health_handler.go      # ヘルスハンドラー
│   │   └── metrics_handler.go     # メトリクスハンドラー
│   └── protocol/
│       ├── bigbird_api.go         # BigBird API定義（製品API）
│       ├── request_parser.go      # リクエストパーサー
│       ├── response_builder.go    # レスポンスビルダー
│       └── error_handler.go       # エラーハンドラー
├── monitoring/
│   ├── metrichawk/
│   │   ├── metrics_collector.go   # MetricHawk メトリクス
│   │   ├── performance_monitor.go # パフォーマンス監視
│   │   └── system_monitor.go      # システム監視
│   ├── tracehunter/
│   │   ├── trace_collector.go     # TraceHunter トレーシング
│   │   ├── span_processor.go      # スパン処理
│   │   └── trace_exporter.go      # トレース出力
│   ├── logstream/
│   │   ├── log_aggregator.go      # LogStream ログ集約
│   │   ├── log_processor.go       # ログ処理
│   │   └── log_shipper.go         # ログ転送
│   └── cockpitdash/
│       ├── dashboard.go           # CockpitDash ダッシュボード
```

```
│       ├── chart_generator.go     # チャート生成
│       └── alert_manager.go       # アラート管理
├── sql/
│   ├── parser/
│   │   ├── talon_lexer.go         # Talon 字句解析
│   │   ├── talon_parser.go        # Talon 構文解析
│   │   ├── ast.go                 # 抽象構文木
│   │   ├── validator.go           # 構文検証
│   │   └── syntax_highlighter.go # シンタックスハイライト
│   ├── ddl/
│   │   ├── create_table.go        # CREATE TABLE
│   │   ├── alter_table.go         # ALTER TABLE
│   │   ├── drop_table.go          # DROP TABLE
│   │   ├── create_view.go         # CREATE VIEW
│   │   └── create_function.go     # CREATE FUNCTION
│   ├── dml/
│   │   ├── insert.go              # INSERT
│   │   ├── update.go              # UPDATE
│   │   ├── delete.go              # DELETE
│   │   ├── merge.go               # MERGE
│   │   └── truncate.go            # TRUNCATE
│   ├── dql/
│   │   ├── select.go              # SELECT
│   │   ├── join.go                # JOIN処理
│   │   ├── aggregation.go         # 集計処理
│   │   ├── window.go              # ウィンドウ関数
│   │   ├── cte.go                 # CTE処理
│   │   ├── pivot.go               # PIVOT/UNPIVOT
│   │   └── recursive_cte.go       # 再帰CTE
│   └── dcl/
│       ├── grant.go               # GRANT
│       └── revoke.go              # REVOKE
├── types/
│   ├── phoenix_types.go           # Phoenix Engine独自データ型定義
│   ├── schema.go                  # スキーマ定義
│   ├── table.go                   # テーブル定義
│   ├── column.go                  # カラム定義
│   ├── index.go                   # インデックス定義
│   └── constraint.go              # 制約定義
├── cli/
│   ├── bigbird_cli.go             # BigBird CLI実装
│   ├── command_parser.go          # コマンドパーサー
│   ├── interactive_shell.go       # 対話シェル
│   ├── script_runner.go           # スクリプト実行
│   └── output_formatter.go        # 出力フォーマッター
├── config/
│   ├── server_config.go           # サーバー設定
```

```
│   ├── cluster_config.go          # クラスター設定
│   ├── security_config.go         # セキュリティ設定
│   ├── performance_config.go      # パフォーマンス設定
│   └── storage_config.go          # ストレージ設定
├── utils/
│   ├── phoenix_logger.go          # Phoenix 独自ログシステム
│   ├── crypto_utils.go            # 暗号化ユーティリティ
│   ├── network_utils.go           # ネットワークユーティリティ
│   ├── file_utils.go              # ファイルユーティリティ
│   ├── memory_utils.go            # メモリユーティリティ
│   └── math_utils.go              # 数学ユーティリティ
├── data/
│   ├── databases/                 # データベースファイル (.bbf)
│   ├── indexes/                   # インデックスファイル (.bbi)
│   ├── logs/                      # ログファイル (.bbl)
│   ├── backups/                   # バックアップファイル
│   └── temp/                      # 一時ファイル
├── certs/
│   ├── trustforge/                # TrustForge 独自認証局
│   ├── server/                    # サーバー証明書
│   ├── client/                    # クライアント証明書
│   └── vault/                     # VaultKeeper 暗号化キー
├── docs/
│   ├── README.md                  # プロジェクト概要
│   ├── ARCHITECTURE.md            # アーキテクチャ詳細
│   ├── API_REFERENCE.md           # API リファレンス
│   ├── SQL_REFERENCE.md           # SQL リファレンス
│   ├── SECURITY.md                # セキュリティガイド
│   ├── DEPLOYMENT.md              # デプロイメントガイド
│   └── CONTRIBUTING.md            # 貢献ガイド
├── scripts/
│   ├── build.sh                   # ビルドスクリプト
│   ├── deploy.sh                  # デプロイスクリプト
│   ├── test.sh                    # テストスクリプト
│   ├── benchmark.sh               # ベンチマークスクリプト
│   └── setup.sh                   # セットアップスクリプト
├── tests/
│   ├── unit/                      # ユニットテスト
│   ├── integration/               # 統合テスト
│   ├── performance/               # パフォーマンステスト
│   ├── security/                  # セキュリティテスト
│   └── load/                      # 負荷テスト
├── web/
│   ├── cockpitdash/               # CockpitDash ダッシュボード
│   ├── api_docs/                  # API ドキュメント
│   └── monitoring/                # 監視UI
└── go.mod                        # Go モジュール定義
```

```
├── go.sum                      # Go モジュール依存関係
├── Makefile                    # ビルド設定
├── Dockerfile                  # Docker設定（FeatherBox用）
├── LICENSE                     # ライセンス
└── .gitignore                  # Git除外設定
```

## 🔧 2. 技術スタック（完全独自開発）

### データベースエンジン（独自開発）

- **Go言語** - 高性能&並行処理
- **Phoenix Engine** - 完全独自データベースエンジン
- **Talon SQL Parser** - 独自SQLパーサー・レクサー
- **Query Optimizer** - 独自クエリオプティマイザー

### データベースストレージ（独自開発）

- **FeatherFormat (.bbf)** - 独自バイナリファイル形式
- **SwiftIndex (.bbi)** - 独自インデックスファイル
- **FlightLog (.bbl)** - 独自トランザクションログ
- **TalonCompress** - 独自圧縮アルゴリズム
- **IronVault** - ファイルレベル独自暗号化

### 分散システム（独自開発）

- **Harmony Protocol** - 独自合意アルゴリズム
- **SkyMesh Protocol** - 独自クラスター通信プロトコル
- **EagleEye Discovery** - 独自自動ノード発見システム
- **BigBird AutoNet** - 独自自動P2Pメッシュネットワーク
- **WhisperWave** - 独自ゴシッププロトコル
- **Auto-Bootstrap** - 起動時自動ネットワーク参加
- **Zero-Config Join** - ゼロ設定自動クラスター参加

### クラスター管理（独自開発）

- **Maestro Orchestrator** - 独自コンテナオーケストレーション
- **FeatherBox Runtime** - 独自軽量コンテナ
- **Beacon Discovery** - 独自サービス発見
- **ConfigNest** - 独自分散設定管理
- **PulseMonitor** - 独自ヘルスモニタリング

**負荷分散・プロキシ（独自開発）**

- **Guardian Proxy** - 独自高性能プロキシサーバー
- **TurboBalance** - 独自ロードバランサー
- **CircuitGuard** - 独自サーキットブレーカー
- **FlowControl** - 独自レート制限システム

**セキュリティ（独自開発）**

- **IronClad-TLS** - 独自超強力暗号化プロトコル
- **TrustForge CA** - 独自認証局
- **Bastion WAF** - 独自Web Application Firewall
- **SentinelIDS** - 独自侵入検知システム
- **VaultKeeper** - 独自キー管理システム

**API・通信（独自開発）**

- **Portal Gateway** - 独自APIゲートウェイ
- **SwiftRPC Protocol** - 独自RPC通信プロトコル
- **CloudQueue** - 独自メッセージキューシステム
- **BinaryStream** - 独自バイナリ通信プロトコル

**監視・ログ（独自開発）**

- **MetricHawk** - 独自メトリクス収集システム
- **TraceHunter** - 独自分散トレーシング
- **LogStream** - 独自ログ集約システム
- **CockpitDash** - 独自監視ダッシュボード

## 🎯 3. 実装優先度

**Phase 1: 基礎機能（1-2ヶ月）**

1. Phoenix Engine独自エンジン基盤
2. Talon独自SQLパーサー・レクサー
3. FeatherFormat独自ファイル形式（.bbf/.bbi/.bbl）
4. 基本DDL（CREATE/DROP TABLE）
5. 基本DML（INSERT/UPDATE/DELETE/SELECT）
6. BigBird CLI実装
7. Portal Gateway基本実装

8. 基本SQLインジェクション対策

**Phase 2: セキュリティ基盤（2-3ヶ月）**

1. IronClad-TLS独自実装
2. TrustForge独自証明書管理システム
3. APIアクセス制限システム
4. Bastion WAF実装
5. 量子耐性暗号化
6. 基本クラスター機能
7. 分散セキュリティ基盤

**Phase 3: 分散機能（3-4ヶ月）**

1. Harmony Protocol独自合意アルゴリズム
2. SkyMesh独自クラスター通信プロトコル
3. EagleEye独自自動ノード発見システム
4. BigBird AutoNet自動P2Pメッシュネットワーク
5. Auto-Bootstrap起動時自動ネットワーク参加
6. Zero-Config Join ゼロ設定自動参加
7. Maestro独自オーケストレーター
8. シャーディング・データ分散
9. Guardian Proxy + TurboBalance
10. 分散トランザクション

**Phase 4: 高可用性（4-5ヶ月）**

1. 地域間レプリケーション
2. 災害復旧・自動バックアップ
3. CockpitDash独自監視・ダッシュボード
4. 自動スケーリング
5. 分散キャッシュ
6. ウィンドウ関数・CTE
7. 配列・JSON操作
8. SentinelIDS AI脅威検知・異常検知

**Phase 5: 最高級機能（5-6ヶ月）**

1. 完全分散クエリ最適化

2. ML分散学習
3. リアルタイムストリーミング
4. 再帰CTE・PIVOT/UNPIVOT
5. UDF（SQL/JavaScript）
6. 地理空間分析
7. 完全ゼロトラスト・セキュリティ
8. 独自プロトコル完成

## 💡 4. パフォーマンス最適化戦略

### 分散システム最適化

- **Smart Routing** - クエリ最適ルーティング
- **Data Locality** - データ局所性最適化
- **Cross-Region Optimization** - 地域間最適化
- **Network Compression** - ネットワーク圧縮
- **Connection Multiplexing** - 接続多重化

### クエリ最適化

- **分散クエリプランニング** - 複数ノードでの最適実行計画
- **統計ベース最適化** - 分散テーブル統計情報活用
- **インデックス最適化** - 分散インデックス推奨
- **パーティション最適化** - 効率的分散データアクセス
- **並列実行** - マルチノード・マルチコア活用

### ストレージ最適化

- **分散カラムナーストレージ** - 地域分散分析クエリ高速化
- **圧縮** - LZ4/Snappy分散圧縮
- **分散キャッシュ** - 多層・多地域キャッシュ戦略
- **プリフェッチ** - 分散先読み最適化
- **Data Tiering** - ホット・コールドデータ階層化

### メモリ最適化

- **分散ストリーミング処理** - 大データ地域分散対応
- **メモリプール** - ノード間オブジェクト再利用
- **GC最適化** - 分散ガベージコレクション最適化

- **Memory Balancing** - ノード間メモリ負荷分散

**可用性・信頼性最適化**

- **Graceful Degradation** - 段階的性能劣化
- **Circuit Breaker Pattern** - 障害連鎖防止
- **Bulkhead Pattern** - 障害隔離
- **Retry Strategy** - 指数バックオフ再試行
- **Health-based Routing** - 健全ノード優先ルーティング

**セキュリティ最適化**

- **Zero-Copy Security** - ゼロコピーセキュリティ処理
- **JIT Compilation** - セキュリティルールJITコンパイル
- **Bloom Filter** - 高速ブラックリスト検索
- **ML-based Detection** - 機械学習異常検知
- **Hardware Acceleration** - ハードウェア暗号化加速
- **Security Caching** - セキュリティ判定結果キャッシュ
- **Parallel Validation** - 並列セキュリティ検証
- **Streaming Security** - ストリーミングセキュリティ処理

---

## 🎯 実装ロードマップ

### 📅 開発スケジュール

| フェーズ | 期間 | 主要機能 | 独自実装レベル | セキュリティレベル | 完成度 |
|---|---|---|---|---|---|
| Phase 1 | 1-2ヶ月 | 独自エンジン基盤 | 基本独自実装 | 基本防御 | 15% |
| Phase 2 | 2-3ヶ月 | セキュリティ基盤 | 独自暗号化実装 | BigBird-TLS実装 | 35% |
| Phase 3 | 3-4ヶ月 | 分散機能 | 独自プロトコル実装 | API制御完成 | 55% |
| Phase 4 | 4-5ヶ月 | 高可用性 | 独自オーケストレーション | AI脅威検知 | 75% |
| Phase 5 | 5-6ヶ月 | 最高級機能 | 完全独自実装 | 量子耐性暗号 | 95% |

### 🏆 目標

**2025年末までに：**

- ✅ BigQuery SQL 95%互換
- ✅ Phoenix Engine完全独自実装
- ✅ FeatherFormat独自ファイル形式（.bbf/.bbi/.bbl）
- ✅ BigBird SSL/TLS量子耐性暗号

- ✅ 毎秒100万クエリ処理（分散）
- ✅ エクサバイト級データ対応
- ✅ 99.99%可用性（4-9s）
- ✅ SQLインジェクション100%防御
- ✅ Portal Gateway API-Only外部アクセス
- ✅ BigBird CLI-Only管理アクセス
- ✅ ゼロデイ攻撃完全防御
- ✅ BigBird AutoNet独自自動P2Pメッシュネットワーク
- ✅ Auto-Bootstrap起動時自動ネットワーク参加
- ✅ Zero-Config Join完全ゼロ設定自動参加
- ✅ EagleEye自動ノード発見システム
- ✅ Maestro独自オーケストレーター
- ✅ Guardian Proxy独自プロキシ・ロードバランサー
- ✅ 完全オープンソース
- ✅ プロジェクトフォルダ内完結
- ✅ 既存技術ゼロ依存
- ✅ 地域間自動レプリケーション
- ✅ ゼロダウンタイム運用
- ✅ 企業利用可能レベル

このアーキテクチャで、BigQueryの全機能を網羅した **BigBird - 世界最高レベルの分散オープンソースデータベース** を構築しましょう！🚀✨

---

## 🌐 BigBird分散システムの特徴

### 🔗 どこからでもアクセス可能

- **Any Node Access** - どのサーバーに接続しても同じデータ
- **Transparent Routing** - ユーザーに透明なクエリルーティング
- **Global Consistency** - 世界中で一貫したデータ

### 🛡️ 究極の冗長性

- **Multi-Region Replication** - 複数地域での自動レプリケーション
- **Zero-Downtime Operations** - メンテナンス中も稼働継続
- **Disaster Recovery** - 災害時の自動復旧

### ⚡ スケーラブル性能

- **Horizontal Scaling** - ノード追加で性能向上
- **Auto-Sharding** - データ増加に応じた自動分散
- **Load Distribution** - 負荷の自動分散

### 🔒 エンタープライズセキュリティ

- **End-to-End Encryption** - 通信・保存データ暗号化
- **Distributed Authentication** - 分散認証システム
- **Audit Logging** - 完全な監査ログ
- **SQLインジェクション完全防御** - 多層防御システム
- **AI脅威検知** - 機械学習による異常検知
- **ゼロトラスト・アーキテクチャ** - すべてを検証

---

## 🛡️ BigBird SQLインジェクション対策詳細

### 🔍 多層防御システム

#### 📝 Layer 1: 入力値検証

go

```
// 厳格な型検証
func ValidateInput(value interface{}, expectedType DataType) error {
    switch expectedType {
    case STRING:
        return validateString(value.(string))
    case INTEGER:
        return validateInteger(value)
    case FLOAT:
        return validateFloat(value)
    }
}

// 危険パターン検出
var DANGEROUS_PATTERNS = []string{
    `(?i)(union|select|insert|update|delete|drop|exec|script)`,
    `(?i)(or\s+1\s*=\s*1)`,
    `(?i)(and\s+1\s*=\s*1)`,
    `(?i)(\-\-|\#|\/\*)`,
    `(?i)(xp_|sp_)`,
}
```

#### 🔒 Layer 2: パラメータ化クエリ強制

go

```go
// BigBird独自のセーフクエリビルダー
type SafeQuery struct {
    SQL        string
    Parameters []Parameter
}

func (q *SafeQuery) AddCondition(column string, operator string, value interface{}) {
    // 列名とオペレーターをホワイトリストで検証
    if !isValidColumn(column) || !isValidOperator(operator) {
        panic("Invalid column or operator")
    }
    q.Parameters = append(q.Parameters, Parameter{Value: value})
}
```

### 🧠 Layer 3: AI異常検知

go

```go
// 機械学習ベースの異常検知
type ThreatDetector struct {
    model        *MLModel
    baselines    map[string]float64
    alertChannel chan SecurityAlert
}

func (td *ThreatDetector) AnalyzeQuery(query string, user User) ThreatLevel {
    features := extractFeatures(query, user)
    score := td.model.Predict(features)

    if score > THREAT_THRESHOLD {
        td.alertChannel <- SecurityAlert{
            Type:  SQL_INJECTION_ATTEMPT,
            Query: query,
            User:  user,
            Score: score,
        }
        return HIGH_THREAT
    }
    return LOW_THREAT
}
```

### ⚡ Layer 4: リアルタイムWAF

go

```go
// 高速WAFエンジン
type WAFEngine struct {
    rules       []SecurityRule
    bloomFilter *BloomFilter
    cache       *ThreatCache
}

func (waf *WAFEngine) ProcessRequest(req *SQLRequest) (*Response, error) {
    // 超高速ブルームフィルターで既知の脅威をチェック
    if waf.bloomFilter.Contains(req.Query) {
        return nil, fmt.Errorf("Known malicious pattern detected")
    }

    // ルールエンジンで詳細検証
    for _, rule := range waf.rules {
        if rule.Matches(req) {
            return waf.handleThreat(req, rule)
        }
    }

    return waf.processCleanRequest(req), nil
}
```

## 🚨 リアルタイム脅威対応

### 📊 異常検知ダッシュボード

- **リアルタイム脅威マップ** - 世界中の攻撃を可視化
- **攻撃パターン分析** - SQLインジェクション手法の傾向
- **ユーザー行動分析** - 異常なクエリパターン検出
- **自動レスポンス** - 脅威に対する自動対応

### 🔧 自動防御機能

- **動的IPブロック** - 攻撃元IPの自動ブロック
- **アカウント一時停止** - 疑わしいアカウントの自動停止
- **クエリレート制限** - 異常なクエリ頻度の制限
- **セッション無効化** - 危険なセッションの即座な無効化

## 🏆 セキュリティ認証・準拠

### 📜 準拠予定規格

- **OWASP Top 10** - Webアプリケーションセキュリティ
- **SOC 2 Type II** - セキュリティ・可用性・機密性
- **ISO 27001** - 情報セキュリティマネジメント
- **PCI DSS** - 決済カード業界データセキュリティ標準
- **GDPR準拠** - EU一般データ保護規則

### 🔐 ゼロトラスト実装

- **すべてのリクエストを検証** - 内部・外部問わず全検証
- **最小権限の原則** - 必要最小限のアクセス権のみ
- **継続的な監視** - 24/7リアルタイム監視
- **マイクロセグメンテーション** - ネットワーク細分化

---

## 🌐 BigBird自動ネットワーク参加システム

### 🚀 Auto-Bootstrap（自動ブートストラップ）

BigBirdは起動時に完全自動でネットワークを発見し、既存のBigBirdクラスターに参加します。

```go
// 自動ブートストラップシステム
type AutoBootstrap struct {
    networkScanner   *NetworkScanner
    peerDiscovery    *PeerDiscovery
    autoJoin         *ZeroConfigJoin
    eagleEye         *EagleEyeDiscovery
    autoNet          *BigBirdAutoNet
}

func (ab *AutoBootstrap) StartAutoDiscovery() error {
    // 1. ローカルネットワークスキャン
    localPeers := ab.networkScanner.ScanLocalNetwork()

    // 2. 既知のBigBirdノード発見
    bigbirdNodes := ab.eagleEye.FindBigBirdNodes(localPeers)

    // 3. インターネット経由でのピア発見
    internetPeers := ab.peerDiscovery.DiscoverInternetPeers()

    // 4. 最適なクラスターを選択して自動参加
    return ab.autoJoin.JoinBestCluster(bigbirdNodes, internetPeers)
}
```

## 🔍 EagleEye自動発見システム

go

```go
// EagleEye - 超高速ノード発見
type EagleEyeDiscovery struct {
    multicastAddr   string
    broadcastPort   int
    discoveryBeacon *DiscoveryBeacon
}

func (ee *EagleEyeDiscovery) FindBigBirdNodes(targets []string) []Node {
    // マルチキャスト探索
    multicastNodes := ee.scanMulticast()

    // ブロードキャスト探索
    broadcastNodes := ee.scanBroadcast()

    // BigBirdシグネチャ検証
    return ee.validateBigBirdSignatures(multicastNodes, broadcastNodes)
}

// BigBird独自の発見シグネチャ
const BIGBIRD_DISCOVERY_SIGNATURE = "BB:Phoenix:Talon:2025"
```

## 🕸️ BigBird AutoNet（自動P2Pメッシュ）

go

```go
// BigBird AutoNet - 完全自動P2Pメッシュネットワーク
type BigBirdAutoNet struct {
    nodeID          string
    meshTopology    *MeshTopology
    peerTable       *PeerTable
    whisperWave     *WhisperWave
}

func (ban *BigBirdAutoNet) AutoJoinMesh(discoveredPeers []Node) error {
    // 1. 最適なエントリーポイント選択
    entryPoint := ban.selectBestEntryPoint(discoveredPeers)

    // 2. ハンドシェイク実行
    if err := ban.performHandshake(entryPoint); err != nil {
        return err
    }

    // 3. メッシュトポロジーに自動参加
    return ban.meshTopology.IntegrateNode(ban.nodeID)
}
```

## ⚡ Zero-Config Join（ゼロ設定参加）

go

```go
// 完全ゼロ設定でのクラスター参加
type ZeroConfigJoin struct {
    clusterAnalyzer *ClusterAnalyzer
    autoConfig      *AutoConfig
    joinProtocol    *JoinProtocol
}

func (zcj *ZeroConfigJoin) JoinBestCluster(localNodes, internetNodes []Node) error {
    // 1. 利用可能なクラスターを分析
    clusters := zcj.clusterAnalyzer.AnalyzeClusters(localNodes, internetNodes)

    // 2. 最適なクラスターを自動選択
    bestCluster := zcj.selectOptimalCluster(clusters)

    // 3. 自動設定生成
    config := zcj.autoConfig.GenerateConfig(bestCluster)

    // 4. 参加プロトコル実行
    return zcj.joinProtocol.JoinCluster(bestCluster, config)
}

func (zcj *ZeroConfigJoin) selectOptimalCluster(clusters []Cluster) Cluster {
    // 選択基準：
    // - ネットワーク遅延（最低）
    // - ノード数（安定性）
    // - バージョン互換性
    // - 地理的距離
    // - セキュリティレベル
    return zcj.scoreAndRankClusters(clusters)[0]
}
```

## 📡 自動ネットワークスキャンシステム

go

```go
// ネットワーク自動スキャン
type NetworkScanner struct {
    portRange       []int
    scanTimeout     time.Duration
    parallelWorkers int
}

func (ns *NetworkScanner) ScanLocalNetwork() []string {
    // 1. ローカルサブネット検出
    subnets := ns.detectLocalSubnets()

    // 2. BigBird標準ポートスキャン
    bigbirdPorts := []int{8734, 8735, 8736} // BigBird専用ポート

    // 3. 並列スキャン実行
    return ns.parallelPortScan(subnets, bigbirdPorts)
}

func (ns *NetworkScanner) detectLocalSubnets() []string {
    // ローカルネットワークインターフェース自動検出
    interfaces, _ := net.Interfaces()
    var subnets []string

    for _, iface := range interfaces {
        addrs, _ := iface.Addrs()
        for _, addr := range addrs {
            if ipnet, ok := addr.(*net.IPNet); ok && !ipnet.IP.IsLoopback() {
                subnets = append(subnets, ipnet.String())
            }
        }
    }
    return subnets
}
```

## 🎯 自動参加フロー

```
[BigBird起動]
      ↓
[NetworkScanner実行]
      ↓
[EagleEye自動発見]
      ↓
[BigBirdノード検証]
      ↓
[クラスター分析]
      ↓
[最適クラスター選択]
      ↓
[Zero-Config参加]
      ↓
[AutoNet統合]
      ↓
[完全動作開始]
```

## 🔧 設定オプション

yaml

```yaml
# bigbird-auto.yaml（オプション設定）
auto_discovery:
  enabled: true                    # 自動発見有効
  scan_local_network: true         # ローカルネットワークスキャン
  scan_internet: true              # インターネットピア発見
  join_automatically: true         # 自動参加
  preferred_regions: ["asia", "us"] # 優先地域
  max_join_attempts: 5             # 最大参加試行回数
  discovery_timeout: 30s           # 発見タイムアウト

network_preferences:
  prefer_local_cluster: true       # ローカルクラスター優先
  min_cluster_size: 3              # 最小クラスターサイズ
  max_latency_ms: 100              # 最大遅延許容値
  require_ssl: true                # SSL必須
```

## 🌟 特徴

- **完全自動化** - 設定ファイル不要で即座にクラスター参加
- **インテリジェント選択** - 最適なクラスターを自動判定
- **フォールバック機能** - 参加失敗時の自動リトライ
- **セキュア接続** - すべての通信が暗号化済み

- **ゼロダウンタイム** - 既存クラスターへの無停止参加