
Patroni

リリース *3.1.0*

Zalando SE

2023 年 09 月 01 日

内容：

第 1 章	はじめに	3
1.1	開発状況	3
1.2	技術要件/インストール	3
1.3	PostgreSQL ノード数の計画	5
1.4	実行と構成	5
1.5	YAML 構成	5
1.6	環境構成	6
1.7	レプリケーションの選択肢	6
1.8	アプリケーションはスーパーユーザーを使用しないでください	6
1.9	HA ソリューションのテスト	6
第 2 章	パトローニ構成	9
2.1	動的構成設定	9
2.2	YAML 構成設定	12
2.3	環境構成設定	27
2.4	重要なルール	39
第 3 章	PatroniRESTAPI	43
3.1	ヘルスチェックエンドポイント	43
3.2	モニタリングエンドポイント	45
3.3	クラスターステータスエンドポイント	51
3.4	構成エンドポイント	53
3.5	スイッチオーバーとフェイルオーバーのエンドポイント	55
3.6	エンドポイントをリスタート	56
3.7	エンドポイントをリロード	57
3.8	エンドポイントを再初期化する	57
第 4 章	レプリカのイメージングとブートストラップ	59
4.1	ブートストラップ	59
4.2	レプリカの構築	60
4.3	スタンバイクラスタ	62
第 5 章	レプリケーションモード	65
5.1	非同期モードの持続性	65
5.2	PostgreSQL 同期レプリケーション	65
5.3	同期モード	66
5.4	同期レプリケーション係数	67
5.5	同期モードの実装	67
第 6 章	ウォッチドッグのサポート	69
6.1	Linux でソフトウェアウォッチドッグを設定する	70

第 7 章	クラスターの一時停止/再開モード	71
7.1	目標	71
7.2	実装	71
7.3	ユーザーガイド	72
第 8 章	DCS フェイルセーフモード	73
8.1	問題	73
8.2	今回の実施の理由	73
8.3	DCS フェイルセーフモード	73
8.4	低レベル実装の詳細	74
8.5	よくある質問	74
第 9 章	Kubernetes で Patroni を使用する	77
9.1	エンドポイントを使用する	77
9.2	ConfigMap を使用する	77
9.3	構成	77
9.4	例	79
第 10 章	Citus のサポート	81
10.1	TL;DR	81
10.2	パトロン	82
10.3	Citus ワーカーの切り替え	83
10.4	DCS をのぞく	85
10.5	CitusonKubernetes	85
10.6	Citus のアップグレードと PostgreSQL のメジャーアップグレード	88
第 11 章	スタンドアロンを Patroni クラスターに変換する	89
11.1	プロシージャ	89
第 12 章	PostgreSQL バージョンのメジャーアップグレード	93
12.1	よくある質問	93
第 13 章	セキュリティに関する考慮事項	95
13.1	DCS の保護	95
13.2	RESTAPI の保護	95
第 14 章	HA マルチデータセンター	97
14.1	同期レプリケーション	97
14.2	非同期レプリケーション	98
第 15 章	リリースノート	99
15.1	バージョン 3.1.0	99
15.2	バージョン 3.0.4	101
15.3	バージョン 3.0.3	102
15.4	バージョン 3.0.2	103
15.5	バージョン 3.0.1	104
15.6	バージョン 3.0.0	104
15.7	バージョン 2.1.7	105
15.8	バージョン 2.1.6	106

15.9 バージョン 2.1.5	107
15.10 バージョン 2.1.4	109
15.11 バージョン 2.1.3	110
15.12 バージョン 2.1.2	112
15.13 バージョン 2.1.1	114
15.14 バージョン 2.1.0	115
15.15 バージョン 2.0.2	117
15.16 バージョン 2.0.1	119
15.17 バージョン 2.0.0	120
15.18 バージョン 1.6.5	125
15.19 バージョン 1.6.4	128
15.20 バージョン 1.6.3	129
15.21 バージョン 1.6.2	130
15.22 バージョン 1.6.1	131
15.23 バージョン 1.6.0	134
15.24 バージョン 1.5.6	136
15.25 バージョン 1.5.5	137
15.26 バージョン 1.5.4	138
15.27 バージョン 1.5.3	140
15.28 バージョン 1.5.2	140
15.29 バージョン 1.5.1	141
15.30 バージョン 1.5.0	141
15.31 バージョン 1.4.6	143
15.32 バージョン 1.4.5	143
15.33 バージョン 1.4.4	144
15.34 バージョン 1.4.3	146
15.35 バージョン 1.4.2	147
15.36 バージョン 1.4.1	148
15.37 バージョン 1.4	148
15.38 バージョン 1.3.6	150
15.39 バージョン 1.3.5	151
15.40 バージョン 1.3.4	152
15.41 バージョン 1.3.3	152
15.42 バージョン 1.3.2	153
15.43 バージョン 1.3.1	153
15.44 バージョン 1.3	153
15.45 バージョン 1.2	156
15.46 バージョン 1.1	158
15.47 バージョン 1.0	161
15.48 バージョン 0.90	163
15.49 バージョン 0.80	164

第 16 章 貢献する 167

16.1 投稿ガイドライン	167
-------------------------	-----

Patroni は、Python を使用した高可用性 (HA) PostgreSQL ソリューションのテンプレートです。アクセシビリティを最大限に高めるために、Patroni は ZooKeeper、etcd、Consul、Kubernetes などのさまざまな分散構成ストアをサポートしています。データセンターやその他の場所に HA PostgreSQL を迅速に導入したいと考えているデータベース エンジニア、DBA、DevOps エンジニア、SRE は、この機能が役立つことを期待しています。

私たちは Patroni を「テンプレート」と呼んでいます。これは、Patroni が万能のレプリケーション システムやプラグ アンド プレイのレプリケーション システムではないためです。独自の注意事項があります。賢く利用しましょう。PostgreSQL で高可用性を実現するにはさまざまな方法があります。リストについては、[PostgreSQL のドキュメント](#)を参照してください。

現在サポートされている PostgreSQL バージョン: 9.3 ~ 15

Citus ユーザーへの注意: 3.0 以降、Patroni は Postgres への Citus データベース拡張機能とうまく統合されています。Patroni の高可用性を Citus 分散クラスターと組み合わせて使用方法の詳細については、Patroni ドキュメントの Citus サポート ページを確認してください。

Kubernetes ユーザーへの注意: Patroni は Kubernetes 上でネイティブに実行できます。Patroni ドキュメントの Kubernetes の章を見てください。

第1章 はじめに

Patroni は、Python を使用した高可用性 (HA) PostgreSQL ソリューションのテンプレートです。Patroni は、Compose のプロジェクト [Governor](#) _ のフォークとして始まりました。たくさんの新機能が含まれています。

Patroni を使用した Docker ベースのデプロイメントの例については、現在 Zalando で使用されている [Spilo](#) _ を参照してください。

追加の背景情報については、以下を参照してください。

- [PostgreSQL HA with Kubernetes and Patroni](#) _、KubeCon2016 での JoshBerkus による講演 (ビデオ)
- [Feb. 2016 Zalando Tech blog post](#) _

1.1 開発状況

Patroni は積極的に開発中であり、貢献を受け入れています。詳細については、以下の [Contributing](#) セクションを参照してください。

新作情報 [here](#) を報告します。

1.2 技術要件/インストール

Pre-requirements for Mac OS

Mac に要件をインストールするには、次を実行します。

```
brew install postgresql etcd haproxy libyaml python
```

Psycopg

[psycopg2-2.8](#) _以降、psycopg2 のバイナリバージョンはデフォルトではインストールされなくなります。ソースコードからインストールするには、C コンパイラと postgres+pythondev パッケージが必要です。python の世界では psycopg2 OR psycopg2-binary 依存関係を指定できないためインストール方法を決定する必要があります。

利用可能ないくつかのオプションがあります。

1. ディストリビューションからパッケージマネージャーを使用する

```
sudo apt-get install python3-psycopg2 # install psycopg2 module on Debian/Ubuntu
sudo yum install python3-psycopg2     # install psycopg2 on RedHat/Fedora/CentOS
```

- バイナリパッケージから `psycpg2` をインストールします

```
pip install psycpg2-binary
```

- ソースから `psycpg2` をインストールする

```
pip install psycpg2>=2.5.4
```

- `psycpg2` の代わりに `psycpg3.0` を使用してください

```
pip install psycpg[binary]>=3.0.0
```

General installation for pip

Patroni は pip でインストールできます：

```
pip install patroni[dependencies]
```

ここで、dependencies は空か、次のうちの 1 つ以上で構成されます。

etcd または etcd3

Etcd を分散構成ストア（DCS）として使用するための ``python-etcd`` モジュール

Consul

Consul を DCS として使用するための ``python-consul`` モジュール

飼育係

Zookeeper を DCS として使用するための ``kazoo`` モジュール

出展者

Exhibitor を DCS として使用するための ``kazoo`` モジュール（Zookeeper と同じ依存関係）

クベルネテス

Patroni で Kubernetes を DCS として使用するための ``kubernetes`` モジュール

raft

PythonRaft 実装を DCS として使用するための ``pysyncobj`` モジュール

aws

AWS コールバックを使用するには ``boto3``

たとえば、Patroni を Etcd の依存関係とともに DCS および AWS コールバックとしてインストールするためのコマンドは次のとおりです。

```
pip install patroni[etcd,aws]
```

レプリカの作成またはカスタムブートストラップスクリプト（つまり WAL-E）で呼び出す外部ツールは、Patroni とは別にインストールする必要があることに注意してください。

1.3 PostgreSQL ノード数の計画

Patroni/PostgreSQL ノードは DCS ノードから分離されているため (Patroni が独自に RAFT を実装する場合を除く)、したがって、最小数のノードに関する要件はありません。1 つのプライマリと 1 つのスタンバイで構成されるクラスターを実行しても問題ありません。スタンバイノードは後で追加できます。

1.4 実行と構成

次のセクションでは、Patroni リポジトリが <https://github.com/zalando/patroni> から複製されていることを前提としています。つまり、サンプル構成ファイル `postgres0.yml` および `postgres1.yml` が必要になります。pip で Patroni をインストールした場合、git リポジトリからこれらのファイルを取得し、以下の `./patroni.py` を `patroni` コマンドに置き換えることができます。

始めるには、さまざまな端末から次の操作を行います。:

```
> etcd --data-dir=data/etcd --enable-v2=true
> ./patroni.py postgres0.yml
> ./patroni.py postgres1.yml
```

すると、高可用性クラスターが起動します。YAML ファイルのさまざまな設定をテストして、クラスターの動作がどのように変化するかを確認します。一部のコンポーネントを強制終了して、システムの動作を確認します。

`postgres*.yml` ファイルをさらに追加して、さらに大きなクラスターを作成します。

Patroni は [HAProxy](#) 構成を提供します。これにより、アプリケーションはクラスターのリーダーに接続するための単一のエンドポイントになります。構成するには、次を実行します。

```
> haproxy -f haproxy.cfg
```

```
> psql --host 127.0.0.1 --port 5000 postgres
```

1.5 YAML 構成

etcd、consul、および ZooKeeper の設定に関する包括的な情報については、[here](#) にアクセスしてください。例については、[here](#) を参照してください。

1.6 環境構成

環境変数を介した設定の構成（オーバーライド）に関する包括的な情報については、[here](#) に移動してください。

1.7 レプリケーションの選択肢

Patroni は、デフォルトで非同期である Postgres のストリーミングレプリケーションを使用します。Patroni の非同期レプリケーション構成では、`maximum_lag_on_failover` 設定が可能です。この設定により、フォロワーがリーダーの後ろに特定のバイト数を超える場合にフェールオーバーが発生しないことが保証されます。この設定は、ビジネス要件に基づいて増減する必要があります。耐久性を保証するために、同期レプリケーションを使用することもできます。詳細は `maximum_lag_on_failover` を参照してください。

1.8 アプリケーションはスーパーユーザーを使用しないでください

アプリケーションから接続する場合は、常に非スーパーユーザーを使用します。Patroni が適切に機能するには、データベースへのアクセスが必要です。アプリケーションからスーパーユーザーを使用することにより、`superuser_reserved_connections` 設定で、スーパーユーザー用に予約された接続を含む接続プール全体を使用できる可能性があります。接続プールがいっぱいのため Patroni がプライマリにアクセスできない場合、動作は望ましくありません。

1.9 HA ソリューションのテスト

HA ソリューションのテストは、多くの変数を含む時間のかかるプロセスです。これは、クロスプラットフォームアプリケーションを考えると特に当てはまります。この作業を行うには、訓練を受けたシステム管理者またはコンサルタントが必要です。これは、ドキュメントで詳しく説明できるものではありません。

とはいえ、必ずテストする必要があるインフラストラクチャの一部を次に示します。

- ネットワーク（システムの前のネットワーク、および NIC[物理または仮想] 自分自身）
- ディスク IO
- ファイル制限（Linux では `nofile`）
- ラム。oomkiller をオフにしても、RAM が利用できないと問題が発生する可能性があります。
- CPU
- 仮想化の競合（ハイパーバイザーのオーバーコミット）
- cgroup の制限（上記に関連している可能性が高い）
- postgres プロセスの `kill -9`（postmaster を除く！）。これは、セグメンテーション違反のまともなシミュレーションです。

やってはいけないことの 1 つは、postmaster プロセスで `kill -9` を実行することです。これは、そうすることは実際のシナリオを模倣しないためです。インフラストラクチャが安全でなく、攻撃者が `kill -9` を実行する可能性があるかと懸念している場合、どんな量の HA プロセスでもそれを修正することはできません。攻撃者は、単にプロセスを再度強制終了するか、別の方法でカオスを引き起こします。

第2章 パトローニ構成

2.1 動的構成設定

動的構成は DCS（分散構成ストア）に格納され、すべてのクラスターノードに適用されます。

動的構成を変更するには、`patronictl edit-config` ツールまたは `Patroni patronictl edit-config` を使用できます。

- **loop/_wait** : ループがスリープする秒数。デフォルト値：10
- **tli** : リーダーロックを取得する TTL（秒）。自動フェールオーバープロセスが開始されるまでの時間と考えてください。デフォルト値：30
- **retry/_timeout** : DCS および PostgreSQL 操作の再試行のタイムアウト（秒単位）。これより短い DCS またはネットワークの問題により、Patroni がリーダーを降格させることはありません。デフォルト値：10
- **maximum/_lag/_on/_failover** : フォロワーがリーダー選挙に参加できるようになるまでに遅延できる最大バイト。
- **maximum/_lag/_on/_syncnode** : 同期フォロワーが異常な候補と見なされ、正常な非同期フォロワーと交換されるまでに遅延できる最大バイト。Patroni は、フォロワーが複数いる場合は最大レプリカ lsn を利用し、それ以外の場合はリーダーの現在の wallsn を使用します。デフォルトは-1 です。値が 0 以下に設定されている場合、Patroni は同期の異常なフォロワーをスワップするアクションを実行しません。Patroni が大量のトランザクション時に同期フォロワーを頻繁に交換しないように、値を十分に高く設定してください。
- **max/_timelines/_history** : DCS に保持されるタイムライン履歴アイテムの最大数。デフォルト値：0。0 に設定すると、DCS に完全な履歴が保持されます。
- **primary/_start/_timeout** : フェールオーバーがトリガーされる前にプライマリが障害から回復できる時間（秒）。デフォルトは 300 秒です。0 に設定すると、可能な場合はクラッシュが検出された直後にフェールオーバーが実行されます。非同期レプリケーションを使用する場合、フェールオーバーによりトランザクションが失われる可能性があります。プライマリ障害の最悪のフェールオーバー時間は次のとおりです。loop/_wait+primary/_start/_timeout+loop/_wait 耐久性/可用性のトレードオフに従って値を設定します。
- **primary/_stop/_timeout** : Patroni が Postgres を停止するときに待機することが許可され、synchronous_mode が有効な場合にのみ有効な秒数。>0 に設定され、synchronous_mode が有効になっている場合、停止操作が primary/_stop/_timeout で設定された値を超えて実行されている場合、パトローニはポストマスターに SIGKILL を送信します。耐久性/可用性のトレードオフに従って値を設定します。パラメーターが設定されていない場合、または設定されている <=0 の場合、primary/_stop/_timeout は適用されません。

- **synchronous/_mode** : 同期レプリケーションモードをオンにします。このモードでは、レプリカが同期として選択され、最新のリーダーと同期レプリカのみがリーダー選挙に参加できます。同期モードでは、正常にコミットされたトランザクションがフェイルオーバー時に失われないことが保証されますが、Patroni がトランザクションの耐久性を保証できない場合、書き込みの可用性が失われます。詳細は **synchronous/_mode** を参照してください。
- **synchronous/_mode/_strict** : 利用可能な同期レプリカがない場合に同期レプリケーションが無効になるのを防ぎ、プライマリへのすべてのクライアントの書き込みをブロックします。詳細は **synchronous/_mode/_strict** を参照してください。
- **failsafe/_mode** : **failsafe/_mode** を有効にします。デフォルトは `false` です。
- **postgresql** :
 - **use/_pg/_rewind** : `pg_rewind` を使用するかどうか。デフォルトは `false` です。
 - **use/_slots** : レプリケーションスロットを使用するかどうか。PostgreSQL9.4 以降のデフォルトは `true` です。
 - **recovery/_conf** : フォロワーの構成時に `recovery.conf` に書き込まれる追加の構成設定。PostgreSQL12 には `recovery.conf` はもうありませんが、Patroni が透過的に処理するため、このセクションを引き続き使用できます。
 - **parameters** : Postgres の構成設定のリスト。
 - **pg/_hba** : Patroni が `pg_hba.conf` を生成するために使用する行のリスト。 `hba_file` PostgreSQL パラメーターがデフォルト以外の値に設定されている場合、Patroni はこのパラメーターを無視します。
 - * **-hostallall0.0.0.0/0md5**
 - * **-ホストレプリケーションレプリケーター 127.0.0.1/32md5** : レプリケーションにはこのような行が必要です。
 - **pg/_ident** : Patroni が `pg_ident.conf` を生成するために使用する行のリスト。 `ident_file` PostgreSQL パラメーターがデフォルト以外の値に設定されている場合、Patroni はこのパラメーターを無視します。
 - * **-mapname1systemname1pguser1**
 - * **-mapname1systemname2pguser2**
- **standby/_cluster** : このセクションが定義されている場合、スタンバイクラスターをブートストラップします。
 - **host** : リモートノードのアドレス
 - **port** : リモートノードのポート
 - **primary/_slot/_name** : レプリケーションに使用するリモートノード上のスロット。このパラメーターはオプション、デフォルト値はインスタンス名から派生します（ファンクション ``slot_name_from_member_name`` を参照）。
 - **create/_replica/_methods** : リモートプライマリからスタンバイリーダーをブートストラップするために使用できるメソッドの順序付けられたリスト。 **create/_replica/_methods** で定義されたリストとは異なる場合があります

- **restore/_command** : リモートプライマリからスタンバイクラスター内のノードに WAL レコードを復元するコマンドは、**restore/_command** で定義されたリストと異なる場合があります
- **archive/_cleanup/_command** : スタンバイリーダーのクリーンアップコマンド
- **recovery/_min/_apply/_delay** : スタンバイリーダーに WAL レコードを実際に適用するまで待機する時間
- **slots** : 永続的なレプリケーションスロットを定義します。これらのスロットは、スイッチオーバー/フェールオーバー中に保持されます。存在しないパーマネントスロットは Patroni が作成します。物理スロットは、現在のプライマリでのみ維持されます。論理スロットは、再起動に伴いプライマリからスタンバイにコピーされ、その後、それらの位置は **slots** 秒ごとに進められます (必要な場合)。libpq 接続を介して実行され、巻き戻しまたはスーパーユーザーの資格情報を使用して実行される論理スロットファイルのコピー (**slots** セクションを参照)。レプリカの論理スロット位置が以前のプライマリよりも少し遅れる可能性は常にあります。そのため、フェールオーバー後に 2 回目に一部のメッセージを受信できるようにアプリケーションを準備する必要があります。そうする最も簡単な方法- **confirmed_flush_lsn** を追跡します。永続的な論理レプリケーションスロットを有効にするには **slots** を設定する必要があります、**hot_standby_feedback** も自動的に有効になります。論理レプリケーションスロットのフェイルオーバーは PostgreSQL 9.6 以前では安全ではなく、PostgreSQL バージョン 10 にはいくつかの重要な機能がないため、この機能は PostgreSQL 11 以降でのみ動作します。
 - **my/_slot/_name** : 永続的なレプリケーションスロットの名前。パーマネントスロットの名前が現在のリーダーの名前と一致する場合、作成されません。Patroni は、この構成に追加された永続的なスロット名が、Patroni がメンバー用に自動的に作成するものと一致するかどうかをチェックしないことに注意してください。これらの名前が追加された場合、パトローニは、メンバーが応答しなくなった場合でも、作成されたスロットが削除されないことを保証します。通常、パトローニはスロットを削除します。これは、既存のメンバーを新しい Patroni クラスターにインポートする場合などのいくつかの状況では役立ちますが (詳細は **my/_slot/_name** を参照)、名前のこれらの衝突は DCS で持続されないことにオペレーターは注意する必要があります。Patroni の正常な機能。
 - * **type** : スロットタイプ。physical または logical の可能性があります。スロットが論理的な場合は、**database** および **plugin** を追加で定義する必要があります。
 - * **database** : 論理スロットを作成するデータベース名。
 - * **plugin** : 論理スロットのプラグイン名。
- **ignore/_slots** : Patroni が一致するスロットを無視する必要があるレプリケーションスロットプロパティのセットのリスト。この構成/機能/など。一部のレプリケーションスロットが Patroni の外部で管理されている場合に役立ちます。一致するプロパティのサブセットにより、スロットが無視されます。
 - **name** : レプリケーションスロットの名前。
 - **type** : スロットタイプ。physical または logical になります。スロットが論理的な場合は、**database** および/または **plugin** を追加で定義できます。
 - **database** : データベース名 (logical スロットと一致する場合)。
 - **plugin** : 論理デコードプラグイン (logical スロットと一致する場合)。

注: **slots** はハッシュマップであり、**slots** は配列です。例:

```
slots:
  permanent_logical_slot_name:
    type: logical
    database: my_db
    plugin: test_decoding
  permanent_physical_slot_name:
    type: physical
  ...
ignore_slots:
- name: ignored_logical_slot_name
  type: logical
  database: my_db
  plugin: test_decoding
- name: ignored_physical_slot_name
  type: physical
  ...
```

2.2 YAML 構成設定

2.2.1 グローバル/ユニバーサル

- **name** : ホストの名前。クラスターに対して一意である必要があります。
- **namespace** : Patroni がクラスターに関する情報を保持する構成ストア内のパス。デフォルト値: `"/service/"`
- **scope** : クラスター名

2.2.2 ログ

- **level** : 一般的なログレベルを設定します。デフォルト値は **level** です (**level** を参照)
- **traceback/_level** : トレースバックが表示されるレベルを設定します。デフォルト値は **traceback/_level** です。 **traceback/_level** を有効にした場合にのみトレースバックを見たい場合は、 **traceback/_level** に設定します。
- **format** : ログフォーマット文字列を設定します。デフォルト値は `%(asctime)s%(levelname)s:%(message)s` です (**format** を参照)
- **dateformat** : 日時フォーマット文字列を設定します。 (**dateformat** を参照)
- **max/_queue/_size** : Patroni は 2 段階ロギングを使用しています。ログレコードはメモリ内キューに書き込まれ、キューからそれらをプルして stderr またはファイルに書き込む別のスレッドがあります。内部キューの最大サイズは、デフォルトで **max/_queue/_size** レコードによって制限されます。これは、過去 1 時間 20 分のログを保持するのに十分です。

- **dir** : アプリケーションログを書き込むディレクトリ。ディレクトリが存在し、Patroni を実行しているユーザーが書き込み可能である必要があります。この値を設定すると、アプリケーションはデフォルトで4つの25MB ログを保持します。これらの保持値は、`file_num` および `file_size` で調整できます (下記参照)。
- **file/_num** : 保持するアプリケーションログの数。
- **file/_size** : ログのローリングをトリガーする `atoi.log` ファイルのサイズ (バイト単位)。
- **loggers** : このセクションでは、Python モジュールごとにログレベルを再定義できます
 - **patroni.postmaster**: WARNING
 - **urllib3**: DEBUG

2.2.3 ブートストラップ構成

注釈: Patroni が初めてクラスターを初期化し、設定が DCS に保存されると、YAML 構成の `bootstrap.dcs` セクションに対する今後のすべての変更は有効になりません。それらを変更したい場合は、`patronictl edit-config` または `Patroni bootstrap.dcs` を使用してください。

- **bootstrap** :
 - **dcs** : このセクションは、新しいクラスターを初期化した後、指定された構成ストアの `namespace/scope/config` に書き込まれます。クラスターのグローバル動的構成。 **dcs** で説明されているパラメーターのいずれかを `bootstrap.dcs` に置くことができ、Patroni が新しいクラスターを初期化 (ブートストラップ) した後、このセクションを構成ストアの `namespace/scope/config` に書き込みます。
 - **method** : このクラスターのブートストラップに使用するカスタムスクリプト。
 詳細は [custom bootstrap methods documentation](#) を参照してください。 `initdb` が指定されている場合、デフォルトの `initdb` コマンドに戻します。 `initdb` は、構成ファイルに `method` パラメーターが存在しない場合にもトリガーされます。
 - **initdb** : (オプション) `initdb` に渡されるオプションをリストします。
 - * **-data-checksums** : 9.3 で `pg_rewind` が必要な場合は有効にする必要があります。
 - * **-encoding:UTF8** : 新しいデータベースのデフォルトのエンコーディング。
 - * **-ロケール:UTF8** : 新しいデータベースのデフォルトロケール。
 - **users** : 新しいクラスターを初期化した後に作成する必要がある追加のユーザー。以下の **users** を参照してください。
 - **post/_bootstrap** または **post/_bootstrap** : クラスターの初期化後に実行される追加のスクリプト。スクリプトは、接続文字列 URL (ユーザー名としてクラスタースーパーユーザーを使用) を受け取ります。PGPASSFILE 変数は、`pgpass` ファイルの場所に設定されます。

Bootstrap ユーザー構成

クラスターの初期化後に作成する必要があるユーザー：

- **admin** : ユーザーの名前
 - **password** : (オプション) ユーザーのパスワード
 - **options** : CREATEUSER ステートメントのオプションのリスト
 - **-creatorole**
 - **-createdb**

2.2.4 Citus

Enables integration Patroni with [Citus](#). If configured, Patroni will take care of registering Citus worker nodes on the coordinator. You can find more information about Citus support [here](#).

- **group** :Citus グループ ID、整数。コーディネーターには **0** を使用し、ワーカーには 1、2 Etcd を使
用します
- **database** : citus 拡張機能を作成するデータベース。コーディネーターとすべてのワーカーで同じ
である必要があります。現在、1 つのデータベースのみがサポートされています。

2.2.5 Consul

ほとんどのパラメーターはオプションですが、**host** または **host** のいずれかを指定する必要があります

- **host** :Consul ローカルエージェントの host:port。
- **url** : Consul ローカルエージェントの URL、形式：http(s)://host:port。
- **port** : (オプション) Consul ポート。
- **scheme** : (オプション) **scheme** または **scheme**、デフォルトは **scheme**。
- **token** : (オプション) ACL トークン。
- **verify** : (オプション) HTTPS リクエストの SSL 証明書を検証するかどうか。
- **cacert** : (オプション) CA 証明書。存在する場合、検証が有効になります。
- **cert** : (オプション) クライアント証明書を含むファイル。
- **key** : (オプション) クライアントキーを含むファイル。キーが **key** の一部である場合、空にすること
ができます。
- **dc** : (オプション) 通信するデータセンター。デフォルトでは、ホストのデータセンターが使用され
ます。
- **consistency** : (オプション) Consul 一貫性モードを選択します。可能な値は **default**、**consistent**
、または **stale** (詳細は **consistency _**を参照)

- **checks** : (オプション) セッションに使用された Consul ヘルスチェックのリスト。デフォルトでは、空のリストが使用されます。
- **register/_service** : (オプション) scope パラメーターで定義された名前と、ノードのロールに応じてタグ master、primary、replica、または standby-leader でサービスを登録するかどうか。デフォルトは **register/_service** です。
- **service/_tags** : (オプション) ロールとは別に Consul サービスに追加する追加の静的タグ (master / primary / replica / standby-leader)。デフォルトでは、空のリストが使用されます。
- **service/_check/_interval** : (オプション) 登録された URL に対してヘルスチェックを実行する頻度。デフォルトは「5s」です。
- **service/_check/_tls/_server/_name** : (オプション) TLS 経由で接続するときに SNI ホストをオーバーライドします。 **service/_check/_tls/_server/_name** も参照してください。

token には、次の ACL 権限が必要です。

```
service_prefix "${scope}" {
    policy = "write"
}
key_prefix "${namespace}/${scope}" {
    policy = "write"
}
session_prefix "" {
    policy = "write"
}
```

2.2.6 Etcd

ほとんどのパラメーターはオプションですが、**host**、**host**、**host**、**host** または **host** のいずれかを指定する必要があります

- **host** : etcd エンドポイントの host:port。
- **hosts** : host1:port1,host2:port2,etc... の形式の etcd エンドポイントのリストコンマ区切りの文字列または実際の yaml リストにすることができます。
- **use/_proxies** : このパラメーターが true に設定されている場合、Patroni は **use/_proxies** をプロキシのリストと見なし、etcd クラスターのトポロジ検出を実行しません。
- **url** : etcd の URL。
- **proxy** : etcd のプロキシ URL。プロキシを使用して etcd に接続する場合は、**proxy** 代わりにこのパラメーターを使用します。
- **srv** : クラスター自動検出用に SRV レコードを検索するドメイン。Patroni は、指定されたドメインのこれらの SRV サービス名を (最初に成功するまでこの順序で) クエリしようとします: **_etcd-client-ssl**、**_etcd-client**、**_etcd-ssl**、**_etcd**、**_etcd-server-ssl**、**_etcd-server**。 **_etcd-server-ssl** または **_etcd-server** の SRV レコードが取得された場合、ETCD ビアプロトコルが使用され、使用可能なメンバーの ETCD がクエリされます。それ以外の場合、SRV レコードのホストが使用されます。

- **srv/_suffix** : 検出中に照会される SRV 名のサフィックスを構成します。このフラグを使用して、同じドメインの下複数の etcd クラスターを区別します。 **srv/_suffix** と組み合わせてのみ機能します。たとえば、 **srv_suffix**: `foo` および **srv**: `example.org` が設定されている場合、次の DNSSRV クエリが作成されます。
- **protocol** : (オプション) `http` または `https`、指定されていない場合 `http` が使用されます。 **protocol** または **protocol** が指定された場合-それらからプロトコルを取得します。
- **username** : (オプション) etcd 認証のユーザー名。
- **password** : (オプション) etcd 認証のパスワード。
- **cacert** : (オプション) CA 証明書。存在する場合、検証が有効になります。
- **cert** : (オプション) クライアント証明書を含むファイル。
- **key** : (オプション) クライアントキーを含むファイル。キーが **key** の一部である場合、空にすることができます。

2.2.7 Etcdv3

Patroni がプロトコルバージョン 3 を介して Etcd クラスターで動作するようにしたい場合は、Patroni 構成ファイルの `etcd3` セクションを使用する必要があります。すべての構成パラメーターは `etcd` と同じです。

警告: プロトコルバージョン 2 で作成されたキーはプロトコルバージョン 3 では可視ではなく、その逆です。したがって、Patroni 構成ファイルを更新するだけで `etcd` から `etcd3` に切り替えることはできません。

2.2.8 ZooKeeper

- **hosts** : 形式の ZooKeeper クラスターメンバーのリスト: `['host1:port1','host2:port2','etc...']`。
- **use_ssl** : (オプション) SSL を使用するかどうか。デフォルトは `false` です。 `false` に設定されている場合、SSL 固有のパラメーターはすべて無視されます。
- **cacert** : (オプション) CA 証明書。存在する場合、検証が有効になります。
- **cert** : (オプション) クライアント証明書を含むファイル。
- **key** : (オプション) クライアントキーを含むファイル。
- **key_password** : (オプション) クライアントキーのパスワード。
- **verify** : (オプション) 証明書を検証するかどうか。デフォルトは `true` です。
- **set_acls** : (オプション) 設定されている場合、作成する各 ZNode にデフォルトの ACL を適用するように Kazoo を構成します。ACL は「x509」スキーマを想定し、プリンシパルをキーとし、値のリストとして 1 つ以上の権限を含む辞書として指定する必要があります。パーミッションは、`CREATE`、`READ`、`WRITE`、`DELETE` または `ADMIN` のいずれかです。たとえば、`set_acls: {CN=principal1: [CREATE, READ], CN=principal2: [ALL]}`。

注釈: SSL をサポートするには `kazoo>=2.6.0` をインストールする必要があります。

2.2.9 出展者

- **hosts** : 出展者 (ZooKeeper) ノードの初期リスト: 'host1,host2,etc...'. このリストは、Exhibitor (ZooKeeper) クラスタートポロジが変更されるたびに自動的に更新されます。
- **poll/_interval** : ZooKeeper ノードと Exhibitor ノードのリストを Exhibitor から更新する頻度。
- **port** : 出展者ポート。

2.2.10 Kubernetes

- **bypass/_api/_service** : (オプション) KubernetesAPI と通信するとき、Patroni は通常 ``kubernetes`` サービスに依存しています。そのアドレスは、`KUBERNETES_SERVICE_HOST` 環境変数を介してポッドで公開されます。 `bypass_api_service` が ``true`` に設定されている場合、Patroni はサービスの背後にある API ノードのリストを解決し、それらに直接接続します。
- **namespace** : (オプション) Patroni ポッドが実行されている Kubernetes 名前空間。デフォルト値は ``default`` です。
- **labels** : 形式 `{label1: value1, label2: value2}` のラベル。これらのラベルは、現在のクラスターに関連付けられている既存のオブジェクト (Pod と Endpoints または ConfigMap のいずれか) を見つけるために使用されます。また、Patroni は、作成するすべてのオブジェクト (Endpoint または ConfigMap) にそれらを設定します。
- **scope/_label** : (オプション) クラスタ名を含むラベルの名前。デフォルト値は ``cluster-name`` です。
- **role/_label** : (オプション) ロール (マスターまたはレプリカまたはその他のカスタム値) を含むラベルの名前。Patroni は、実行される Pod にこのラベルを設定します。デフォルト値は `role` です。
- **leader/_label/_value** : (オプション) Postgres ロールが `master` の場合のポッドラベルの値。デフォルト値は `master` です。
- **follower/_label/_value** : (オプション) Postgres ロールが `replica` の場合のポッドラベルの値。デフォルト値は `replica` です。
- **standby/_leader/_label/_value** : (オプション) Postgres ロールが `standby-leader` の場合のポッドラベルの値。デフォルト値は `standby-leader` です。
- **tmp/_role/_label** : (オプション) ロール (マスターまたはレプリカ) を含む一時ラベルの名前。このラベルの値は、常に対応するロールのデフォルトを使用します。必要な場合のみ設定してください。
- **use/_endpoints** : (オプション) `true` に設定されている場合、Patroni は ConfigMap の代わりに Endpoints を使用して、リーダー選挙を実行し、クラスターの状態を維持します。
- **pod/_ip** : (オプション) Patroni が実行されているポッドの IP アドレス。この値は ``use_endpoints`` が有効な場合に必須であり、ポッドの PostgreSQL が昇格するときにリーダーエンドポイントサブセットを設定するために使用されます。

- **ports** : (オプション) サービスオブジェクトにポートの名前がある場合、同じ名前が Endpoint オブジェクトに表示される必要があります。そうでない場合、サービスは動作しません。たとえば、サービスが `{Kind: Service, spec: {ports: [{name: postgresql, port: 5432, targetPort: 5432}]}}` として定義されている場合、`kubernetes.ports: [{"/name/": "/postgresql/", "/port/": 5432}]` を設定する必要があり、Patroni はそれを使用してリーダーエンドポイントのサブセットを更新します。このパラメーターは、`'kubernetes.use_endpoints'` が設定されている場合にのみ使用されます。
- **cacert** : (オプション) KubernetesAPISSL 証明書の検証中に使用する信頼できる CA の証明書を含む CA_BUNDLE ファイルを含むファイルを指定します。提供されない場合、パトニは ServiceAccount シークレットによって提供される値を使用します。
- **retriable/_http/_codes** : (オプション) リトライする K8sAPI からの HTTP ステータスコードのリスト。デフォルトでは、Patroni は 500、503、および 504 で再試行しています、または K8sAPI 応答に `retry-after` HTTP ヘッダーがある場合。

2.2.11 Raft (非推奨)

- **self/_addr** : Raft 接続をリッスンする `ip:port`。 `self_addr` は、クラスターの他のノードからアクセスする必要があります。設定されていない場合、ノードはコンセンサスに参加しません。
- **bind/_addr** : (オプション) Raft 接続をリッスンする `ip:port`。指定しない場合、`self_addr` が使用されます。
- **partner/_addrs** : クラスター内の他の Patroni ノードのリスト。形式: `['ip1:port','ip2:port','etc...']`
- **data/_dir** : Raft ログとスナップショットを保存するディレクトリ。指定しない場合、現在の作業ディレクトリが使用されます。
- **password** : (オプション) 指定されたパスワードで Raft トラフィックを暗号化します。 `cryptography python` モジュールが必要です。

Raft 実装に関する簡単な FAQ

- Q: コンセンサスを提供するすべてのノードをリストするにはどうすればよいですか?
A: `syncobj_admin -conn host:port -status` ここで `host:port` は、クラスターノードのいずれかのアドレスです
- Q: コンセンサスの一部であったノードがなくなり、他のノードに同じ IP を再利用できません。このノードをコンセンサスから削除するにはどうすればよいですか?
A: `syncobj_admin -conn host:port -remove host2:port2` ここで `host2:port2` は、コンセンサスから削除するノードのアドレスです。
- Q: `syncobj_admin` ユーティリティはどこで入手できますか?
A: Patroni 依存関係である `pysyncobj` モジュール (pythonRAFT 実装) と一緒にインストールされます。
- Q: コンセンサスに追加せずに Patroni ノードを実行することは可能ですか?
A: はい、Patroni 構成から `raft.self_addr` をコメントアウトするか削除してください。

- Q:Patroni と PostgreSQL を 2 つのノードでのみ実行することは可能ですか?

A : はい、3 番目のノードで `patroni_raft_controller` を実行できます (Patroni と PostgreSQL なしで)。このようなセットアップでは、プライマリに影響を与えずに 1 つのノードを一時的に失う可能性があります。

2.2.12 PostgreSQL

- `postgresql` :

- `authentication` :

- * `superuser` :

- `username` : スーパーユーザーの名前、初期化 (`initdb`) 中に設定され、後で Patroni が postgres に接続するために使用します。
 - `password` : スーパーユーザーのパスワード、初期化 (`initdb`) 中に設定。
 - `sslmode` : (オプション) `sslmode` 接続パラメーターにマップし、クライアントがサーバーとの TLS ネゴシエーションモードのタイプを指定できるようにします。各モードの仕組みの詳細については、`sslmode` をご覧ください。デフォルトモードは `prefer` です。
 - `sslkey` : (オプション) `sslkey` 接続パラメーターにマップし、クライアントの証明書で 사용되는秘密鍵の場所を指定します。
 - `sslpassword` : (オプション) `sslpassword` 接続パラメーターにマップし、`sslkey` で指定された秘密鍵のパスワードを指定します。
 - `sslcert` : (オプション) クライアント証明書の場所を指定する `sslcert` 接続パラメーターにマップします。
 - `sslrootcert` : (オプション) `sslrootcert` 接続パラメーターにマップします。これは、クライアントがサーバーの証明書を検証するために使用する 1 つ以上の認証局 (CA) 証明書を含むファイルの場所を指定します。
 - `sslcrli` : (オプション) `sslcrli` 接続パラメーターにマップし、証明書失効リストを含むファイルの場所を指定します。クライアントは、このリストに証明書が存在するサーバーへの接続を拒否します。
 - `sslcrldir` : (オプション) `sslcrldir` 接続パラメーターにマップし、証明書失効リストを含むファイルがあるディレクトリの場所を指定します。クライアントは、このリストに証明書が存在するサーバーへの接続を拒否します。
 - `gssencmode` : (オプション) `gssencmode` 接続パラメーターにマップします。これは、安全な GSSTCP/IP 接続をサーバーとネゴシエートするかどうか、またはどの優先度でネゴシエートするかを決定します
 - `channel_binding` : (オプション) `channel_binding` 接続パラメーターにマップし、クライアントによるチャンネルバイndingの使用を制御します。

- * `replication` :

- ・ **username** : レプリケーションユーザー名; ユーザーは初期化中に作成されます。レプリカはこのユーザーを使用して、ストリーミングレプリケーションを介してレプリケーションソースにアクセスします
- ・ **password** : レプリケーションパスワード。ユーザーは初期化中に作成されます。
- ・ **sslmode** : (オプション) **sslmode** _接続パラメーターにマップし、クライアントがサーバーとの TLS ネゴシエーションモードのタイプを指定できるようにします。各モードの仕組みの詳細については、**sslmode** _をご覧ください。デフォルトモードは **prefer** です。
- ・ **sslkey** : (オプション) **sslkey** _接続パラメーターにマップし、クライアントの証明書で 사용되는秘密鍵の場所を指定します。
- ・ **sslpassword** : (オプション) **sslpassword** _接続パラメーターにマップし、**sslkey** で指定された秘密鍵のパスワードを指定します。
- ・ **sslcert** : (オプション) クライアント証明書の場所を指定する **sslcert** _接続パラメーターにマップします。
- ・ **sslrootcert** : (オプション) **sslrootcert** _接続パラメーターにマップします。これは、クライアントがサーバーの証明書を検証するために使用する 1 つ以上の認証局 (CA) 証明書を含むファイルの場所を指定します。
- ・ **sslcrli** : (オプション) **sslcrli** _接続パラメーターにマップし、証明書失効リストを含むファイルの場所を指定します。クライアントは、このリストに証明書が存在するサーバーへの接続を拒否します。
- ・ **sslcrldir** : (オプション) **sslcrldir** _接続パラメーターにマップし、証明書失効リストを含むファイルがあるディレクトリの場所を指定します。クライアントは、このリストに証明書が存在するサーバーへの接続を拒否します。
- ・ **gssencmode** : (オプション) **gssencmode** _接続パラメーターにマップします。これは、安全な GSSTCP/IP 接続をサーバーとネゴシエートするかどうか、またはどの優先度でネゴシエートするかを決定します
- ・ **channel_binding** : (オプション) **channel_binding** _接続パラメーターにマップし、クライアントによるチャンネルバインディングの使用を制御します。

* **rewind** :

- ・ **username** : (オプション) **pg_rewind** のユーザーの名前。ユーザーは postgres11+ の初期化中に作成され、必要なすべての **username** _ が付与されます。
- ・ **password** : (オプション) **pg_rewind** のユーザーのパスワード。ユーザーは初期化中に作成されます。
- ・ **sslmode** : (オプション) **sslmode** _接続パラメーターにマップし、クライアントがサーバーとの TLS ネゴシエーションモードのタイプを指定できるようにします。各モードの仕組みの詳細については、**sslmode** _をご覧ください。デフォルトモードは **prefer** です。
- ・ **sslkey** : (オプション) **sslkey** _接続パラメーターにマップし、クライアントの証明書で 사용되는秘密鍵の場所を指定します。

- ・ **sslpassword** : (オプション) **sslpassword** 接続パラメーターにマップし、**sslkey** で指定された秘密鍵のパスワードを指定します。
 - ・ **sslcert** : (オプション) クライアント証明書の場所を指定する **sslcert** 接続パラメーターにマップします。
 - ・ **sslrootcert** : (オプション) **sslrootcert** 接続パラメーターにマップします。これは、クライアントがサーバーの証明書を検証するために使用する 1 つ以上の認証局 (CA) 証明書を含むファイルの場所を指定します。
 - ・ **sslcrl** : (オプション) **sslcrl** 接続パラメーターにマップし、証明書失効リストを含むファイルの場所を指定します。クライアントは、このリストに証明書が存在するサーバーへの接続を拒否します。
 - ・ **sslcrlidir** : (オプション) **sslcrlidir** 接続パラメーターにマップし、証明書失効リストを含むファイルがあるディレクトリの場所を指定します。クライアントは、このリストに証明書が存在するサーバーへの接続を拒否します。
 - ・ **gssencmode** : (オプション) **gssencmode** 接続パラメーターにマップします。これは、安全な GSSTCP/IP 接続をサーバーとネゴシエートするかどうか、またはどの優先度でネゴシエートするかを決定します
 - ・ **channel_binding** : (オプション) **channel_binding** 接続パラメーターにマップし、クライアントによるチャンネルバイndingの使用を制御します。
- **callbacks** : 特定のアクションで実行するコールバックスクリプト。Patroni はアクション、ロール、クラスター名を渡します。(それらを記述する方法の例として、scripts/aws.py を参照してください。)
 - * **on/_reload** : 構成のリロードがトリガーされたときにこのスクリプトを実行します。
 - * **on/_restart** : postgres の再起動時にこのスクリプトを実行します (ロールを変更せずに)。
 - * **on/_role/_change** : postgres が昇格または降格されているときにこのスクリプトを実行します。
 - * **on/_start** : postgres の起動時にこのスクリプトを実行します。
 - * **on/_stop** : postgres が停止したときにこのスクリプトを実行します。
 - **connect/_address** : IP アドレス+他のノードやアプリケーションから Postgres にアクセスするためのポート。
 - **proxy/_address** : Postgres の隣で実行されている接続プール (例 pgbouncer) にアクセスできる IP アドレス+ポート。値は DCS のメンバーキーに **proxy_url** として書き込まれ、サービス検出に使用/役立つ可能性があります。
 - **create/_replica/_methods** : パトロニノードを新しいレプリカに変えるための作成メソッドの順序付きリスト。"/basebackup/"がデフォルトの方法です。他のメソッドはスクリプトを参照すると想定され、それぞれが独自の構成項目として構成されます。詳細については、**create/_replica/_methods** を参照してください。
 - **data/_dir** : Postgres データディレクトリの場所。 **data/_dir** または Patroni によって初期化されます。

- **config/_dir** :Postgres 構成ディレクトリの場所、デフォルトはデータディレクトリ。Patroni が書き込み可能である必要があります。
- **bin/_dir** : (オプション) PostgreSQL バイナリへのパス (pg_ctl, initdb, pg_controldata, pg_basebackup, postgres, pg_isready, pg_rewind)。提供されない場合、または空の文字列の場合、PATH 環境変数を使用して実行可能ファイルを見つけます。
- **bin/_name** : (オプション) カスタム Postgres ディストリビューションを使用している場合、Postgres バイナリ名をオーバーライドできるようにします。
 - * **pg/_ctl** : (オプション) pg_ctl バイナリのカスタム名。
 - * **initdb** : (オプション) initdb バイナリのカスタム名。
 - * **pg/controldata** : (オプション) pg_controldata バイナリのカスタム名。
 - * **pg/_basebackup** : (オプション) pg_basebackup バイナリのカスタム名。
 - * **postgres** : (オプション) postgres バイナリのカスタム名。
 - * **pg/_isready** : (オプション) pg_isready バイナリのカスタム名。
 - * **pg/_rewind** : (オプション) pg_rewind バイナリのカスタム名。
- **listen** : Postgres がリッスンする IP アドレス+ポート。ストリーミングレプリケーションを使用している場合は、クラスター内の他のノードからアクセスする必要があります。ポートコンポーネントがコロン、つまり listen: 127.0.0.1,127.0.0.2:5432 で最後のものの後に追加される限り、複数のコンマ区切りアドレスが許可されます。Patroni はこのリストの最初のアドレスを使用して、PostgreSQL ノードへのローカル接続を確立します。
- **use/_unix/_socket** : パトロニは、クラスターへの接続に UNIX ソケットの使用を優先することを指定します。デフォルト値は false です。unix_socket_directories が定義されている場合、Patroni はそこから最初の適切な値を使用してクラスターに接続し、適切なものがない場合は tcp にフォールバックします。unix_socket_directories が postgresql.parameters で指定されていない場合、Patroni はデフォルト値を使用する必要があると想定し、接続パラメーターから host を省略します。
- **use/_unix/_socket/_repl** : パトロニがレプリケーションユーザークラスター接続に UNIX ソケットの使用を優先することを指定します。デフォルト値は false です。unix_socket_directories が定義されている場合、Patroni はそこから最初の適切な値を使用してクラスターに接続し、適切なものがない場合は tcp にフォールバックします。unix_socket_directories が postgresql.parameters で指定されていない場合、Patroni はデフォルト値を使用する必要があると想定し、接続パラメーターから host を省略します。
- **pgpass** : pgpass パスワードファイルへのパス。Patroni は、pg/_basebackup、post_init スクリプトを実行する前 Etc、このファイルを作成します。場所は Patroni が書き込み可能である必要があります。
- **recovery/_conf** : フォロワーの構成時に recovery.conf に書き込まれる追加の構成設定。
- **custom/_conf** : postgresql.base.conf の代わりに使用されるオプションのカスタム postgresql.conf ファイルへのパス。ファイルはすべてのクラスターノードに存在し、PostgreSQL で読み取り可能である必要があります、実際の postgresql.conf 上のその場所から含まれます。Patroni はこのファイルの変更を監視せず、バックアップもしないことに注意してください

い。ただし、その設定は引き続き Patroni 独自の構成機能によってオーバーライドできます。詳細については、**custom/_conf** を参照してください。

- **parameters** : Postgres の構成設定のリスト。これらの多くは、レプリケーションが機能するために必要です。
- **pg/_hba** : Patroni が **pg_hba.conf** を生成するために使用する行のリスト。 **hba_file** PostgreSQL パラメーターがデフォルト以外の値に設定されている場合、Patroni はこのパラメーターを無視します。 **pg/_hba** とともに、このパラメーターは **pg_hba.conf** の管理を簡素化します。
 - * **-hostallall0.0.0.0/0md5**
 - * **-ホストレプリケーションレプリケーター 127.0.0.1/32md5** : レプリケーションにはこのような行が必要です。
- **pg/_ident** : Patroni が **pg_ident.conf** を生成するために使用する行のリスト。 **ident_file** PostgreSQL パラメーターがデフォルト以外の値に設定されている場合、Patroni はこのパラメーターを無視します。 **pg/_ident** とともに、このパラメーターは **pg_ident.conf** の管理を簡素化します。
 - * **-mapname1systemname1pguser1**
 - * **-mapname1systemname2pguser2**
- **pg/_ctl/_timeout** : **start**、**stop**、または **restart** を実行するときに **pg_ctl** が待機する時間。デフォルト値は 60 秒です。
- **use/_pg/_rewind** : レプリカとしてクラスターに参加するときに、元のリーダーで **pg/_rewind** を使用してみてください。
- **remove/_data/_directory/_on/_rewind/_failure** : このオプションを有効にすると、パトロニは PostgreSQL データディレクトリを削除し、レプリカを再作成します。それ以外の場合は、新しいリーダーをフォローしようとします。デフォルト値は **remove/_data/_directory/_on/_rewind/_failure** です。
- **remove/_data/_directory/_on/_diverged/_timelines** : Patroni は、タイムラインが発散しており、以前のプライマリが新しいプライマリからストリーミングを開始できないことに気付いた場合、PostgreSQL データディレクトリを削除し、レプリカを再作成します。このオプションは、**pg_rewind** が使用できない場合に役立ちます。PostgreSQLv10 以前でタイムライン発散チェックを実行しているときに、パトロニはレプリケーション資格情報を使用して/"postgres/"データベースに接続しようとします。したがって、そのようなアクセスは **pg_hba.conf** で許可する必要があります。デフォルト値は **remove/_data/_directory/_on/_diverged/_timelines** です。
- **replica/_method** : **basebackup** 以外の **create_replica_methods** ごとに、同じ名前の構成セクションを追加します。少なくとも、これには、実行される実際のスクリプトへのフルパスを指定した/"command/"を含める必要があります。他の構成パラメーターは、/"parameter=value/"の形式でスクリプトに渡されます。
- **pre/_promote** : リーダーロックを取得した後、レプリカを昇格する前に、フェイルオーバー中に実行されるフェンシングスクリプト。スクリプトがゼロ以外のコードで終了した場合、Patroni はレプリカをプロモートせず、DCS からリーダーキーを削除します。
- **before/_stop** : postgres を停止する直前に実行されるスクリプト。コールバックとは対照的に、このスクリプトは同期的に実行され、完了するまでシャットダウンをブロックします。このス

クリプトのリターンコードは、シャットダウンがその後に進むかどうかには影響しません。

2.2.13 RESTAPI

- **restapi** :

- **connect/_address** : Patroni の **connect/_address** にアクセスするための IP アドレス（またはホスト名）とポート。クラスターのすべてのメンバーがこのアドレスに接続できる必要があります。"または"/127.0.0.1/"）。HTTP ヘルスチェック（"/listen/"RESTAPI パラメーターについては以下を参照）のエンドポイントとして、またユーザークエリ（直接または RESTAPI を介して）、およびリーダー選挙中のクラスターメンバー（たとえば、リーダーがまだ実行されているかどうかを判断するため、またはクエリを実行しているよりも先に WAL 位置を持つノードがあるかどうか Etc）connect_address はメンバーキーに入れられます DCS では、メンバー名をアドレスに変換して RESTAPI に接続できます。
- **listen** : Patroni が RESTAPI をリッスンする IP アドレス（またはホスト名）とポート-上記のように、参加ノード間で同じヘルスチェックとクラスターメッセージングも提供します。HAProxy（または HTTP/"OPTION/"または"/GET/"チェックを実行できる他のロードバランサー）のヘルスチェック情報を提供します。
- **authentication** : (オプション)
 - * **username** : 安全でない RESTAPI エンドポイントを保護するための基本認証ユーザー名。
 - * **password** : 安全でない RESTAPI エンドポイントを保護するための基本認証パスワード。
- **certfile** : (オプション) : PEM 形式の証明書を含むファイルを指定します。certfile を指定しないか空の場合、API サーバーは SSL なしで動作します。
- **keyfile** : (オプション) : PEM 形式の秘密鍵を含むファイルを指定します。
- **keyfile/_password** : (オプション) : キーファイルを復号化するためのパスワードを指定します。
- **cafile** : (オプション) : クライアント証明書の検証中に使用する信頼できる CA の証明書を含む CA_BUNDLE ファイルを指定します。
- **ciphers** : (オプション) : 許可される暗号スイートを指定します（例"/ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:DHE-RSA-128-GCM-SHA256:!SSLv1:!SSLv2:!SSLv3:!TLSv1:!TLSv1.1/"）
- **verify/_client** : (オプション) : none (デフォルト)、optional または required。none RESTAPI の場合、クライアント証明書はチェックされません。すべての RESTAPI 呼び出しに required クライアント証明書が必要な場合。すべての安全でない RESTAPI エンドポイントに optional クライアント証明書が必要な場合。required が使用されている場合、証明書の署名の検証が成功すると、クライアント認証が成功します。optional の場合、クライアント証明書は PUT、POST、PATCH、および DELETE 要求に対してのみチェックされます。
- **allowlist** : (オプション) : 安全でない RESTAPI エンドポイントの呼び出しを許可するホストのセットを指定します。単一の要素は、ホスト名、IP アドレス、または CIDR 表記を使用したネットワークアドレスになります。デフォルトでは allow all が使用されます。allowlist または allowlist_include_members が設定されている場合、含まれていないものはすべて拒否されます。

- **allowlist/_include/_members** : (オプション) : **true** に設定すると、DCS に登録されている他のクラスターメンバーから安全でない RESTAPI エンドポイントにアクセスできます (IP アドレスまたはホスト名はメンバー **api_url** から取得されます)。OS が送信接続に別の IP を使用する場合がありますことに注意してください。
- **http/_extra/_headers** : (オプション) : HTTP ヘッダーにより、RESTAPI サーバーは HTTP 応答で追加情報を渡すことができます。
- **https/_extra/_headers** : (オプション) : HTTPS ヘッダーにより、RESTAPI サーバーは、TLS が有効になっている場合に HTTP 応答で追加情報を渡すことができます。これにより、**http_extra_headers** に設定された追加情報も渡されます。
- **request_queue_size** : (オプション) : PatroniRESTAPI が使用する TCP ソケットのリクエストキューサイズを設定します。キューがいっぱいになると、それ以降の要求には「接続が拒否されました」エラーが発生します。デフォルト値は5です。

これは **http_extra_headers** と **https_extra_headers** の両方の例です。

```
restapi:
  listen: <listen>
  connect_address: <connect_address>
  authentication:
    username: <username>
    password: <password>
  http_extra_headers:
    'X-Frame-Options': 'SAMEORIGIN'
    'X-XSS-Protection': '1; mode=block'
    'X-Content-Type-Options': 'nosniff'
  cafile: <ca file>
  certfile: <cert>
  keyfile: <key>
  https_extra_headers:
    'Strict-Transport-Security': 'max-age=31536000; includeSubDomains'
```

警告:

- **restapi.connect_address** は、特定の Patroni クラスターのすべてのノードからアクセスする必要があります。内部では、Patroni はリーダーレース中にそれを使用して、レプリケーションラグが最小限に抑えられたノードを見つけています。
- クライアント証明書の検証を有効にした場合 (**restapi.verify_client** が **required** に設定されている場合)、**restapi.verify_client** も **ctl.certfile**、**ctl.keyfile**、**ctl.keyfile_password** で **restapi.verify_client** を提供します。提供されていない場合、Patroni は正しく動作しません。

2.2.14 CTL

- **ctl** : (オプション)
 - **authentication** :
 - * **username** : 保護された RESTAPI エンドポイントにアクセスするための基本認証ユーザー名。提供されない場合、`patanictl` は `RESTAPI/"username/"` パラメーターに提供された値を使用します。
 - * **password** : 保護された RESTAPI エンドポイントにアクセスするための基本認証パスワード。提供されない場合、`patanictl` は `RESTAPI/"password/"` パラメーターに提供された値を使用します。
 - **insecure** : SSL 証明書を検証せずに RESTAPI への接続を許可します。
 - **cacert** : RESTAPISSL 証明書の検証中に使用する、信頼できる CA の証明書を含む `CA_BUNDLE` ファイルまたはディレクトリを含むファイルを指定します。提供されない場合、`patanictl` は `RESTAPI/"cafile/"` パラメーターに提供された値を使用します。
 - **certfile** : PEM 形式のクライアント証明書を含むファイルを指定します。
 - **keyfile** : PEM 形式のクライアント秘密鍵を含むファイルを指定します。
 - **keyfile/_password** : クライアントキーファイルを復号化するためのパスワードを指定します。

2.2.15 ウォッチドッグ

- **mode** : `off`、`automatic` または `required`。 `off` ウォッチドッグが無効な場合。 `automatic` の場合、ウォッチドッグは使用可能な場合は使用されますが、使用できない場合は無視されます。 `required` の場合、ウォッチドッグを正常に有効にできない限り、ノードはリーダーになりません。
- **device** : ウォッチドッグデバイスへのパス。デフォルトは `/dev/watchdog` です。
- **safety_margin** : ウォッチドッグのトリガーとリーダーキーの有効期限間のセーフティマージンの秒数。

2.2.16 タグ

- **nofailover** : `true` または `false` は、このノードがリーダーレースに参加してリーダーになることを許可するかどうかを制御します。デフォルトは `false`
- **clonefrom** : `true` または `false`。 `true` に設定されている場合、他のノードはブートストラップにこのノードを使用することを好む場合があります（から `pg_basebackup` を取得します）。 `clonefrom` タグが `true` に設定されたノードが複数ある場合、ブートストラップ元のノードはランダムに選択されます。デフォルト値は `false` です。
- **noloadbalance** : `true` または `false`。 `true` に設定されている場合、ノードは `GET /replica` RESTAPI ヘルスチェックに対して HTTP ステータスコード 503 を返すため、ロードバランシングから除外されます。デフォルトは `false` です。

- **replicatefrom** : 別のレプリカの IP アドレス/ホスト名。カスケード複製をサポートするために使用されます。
- **nosync** : `true` または `false`。 `true` に設定されている場合、ノードは同期レプリカとして選択されません。

これらの定義済みタグに加えて、独自のタグを追加することもできます。

- **key1** : `true`
- **key2** : `false`
- **key3** : `1.4`
- **key4** : `/"RandomString/"`

タグは [REST API](#) および `patronictl list` で表示されますこれらのタグを使用して、インスタンスの状態を確認することもできます。インスタンスに対してタグが定義されていない場合、またはそれぞれの値がクエリ値と一致しない場合、HTTP ステータスコード 503 が返されます。

2.3 環境構成設定

システム環境変数を使用して、Patroni 構成ファイルで定義された構成パラメーターの一部をオーバーライドできます。このドキュメントには、Patroni が処理するすべての環境変数がリストされています。これらの変数を介して設定された値は、常に Patroni 構成ファイルで設定された値よりも優先されます。

2.3.1 グローバル/ユニバーサル

- **PATRONI/_CONFIGURATION** : `PATRONI_CONFIGURATION` 環境変数を介して Patroni の構成全体を設定することが可能です。この場合、他の環境変数は考慮されません。
- **PATRONI/_NAME** : Patroni の現在のインスタンスが実行されているノードの名前。クラスターに対して一意である必要があります。
- **PATRONI/_NAMESPACE** : Patroni がクラスターに関する情報を保持する構成ストア内のパス。デフォルト値 : `/"service/"`
- **PATRONI/_SCOPE** : クラスター名

2.3.2 ログ

- **PATRONI/_LOG/_LEVEL** : 一般的なログレベルを設定します。デフォルト値は `PATRONI/_LOG/_LEVEL` です (`PATRONI/_LOG/_LEVEL` を参照)
- **PATRONI/_LOG/_TRACEBACK/_LEVEL** : トレースバックが表示されるレベルを設定します。デフォルト値は `PATRONI/_LOG/_TRACEBACK/_LEVEL` です。 `PATRONI/_LOG/_TRACEBACK/_LEVEL` を有効にした場合にのみトレースバックを見たい場合は、`PATRONI/_LOG/_TRACEBACK/_LEVEL` に設定します。

- **PATRONI/_LOG/_FORMAT** : ログフォーマット文字列を設定します。デフォルト値は `%(asctime)s%(levelname)s:%(message)s` です (**PATRONI/_LOG/_FORMAT** を参照)
- **PATRONI/_LOG/_DATEFORMAT** : 日時フォーマット文字列を設定します。(**PATRONI/_LOG/_DATEFORMAT** を参照)
- **PATRONI/_LOG/_MAX/_QUEUE/_SIZE** : Patroni は 2 段階ロギングを使用しています。ログレコードはメモリ内キューに書き込まれ、キューからそれらをプルして `stderr` またはファイルに書き込む別のスレッドがあります。内部キューの最大サイズは、デフォルトで **PATRONI/_LOG/_MAX/_QUEUE/_SIZE** レコードによって制限されます。これは、過去 1 時間 20 分のログを保持するのに十分です。
- **PATRONI/_LOG/_DIR** : アプリケーションログを書き込むディレクトリ。ディレクトリが存在し、Patroni を実行しているユーザーが書き込み可能である必要があります。この環境変数を設定すると、アプリケーションはデフォルトで 4 つの 25MB ログを保持します。これらの保持値は、`PATRONI_LOG_FILE_NUM` および `PATRONI_LOG_FILE_SIZE` で調整できます (下記参照)。
- **PATRONI/_LOG/_FILE/_NUM** : 保持するアプリケーションログの数。
- **PATRONI/_LOG/_FILE/_SIZE** : ログのローリングをトリガーする `atoi.log` ファイルのサイズ (バイト単位)。
- **PATRONI/_LOG/_LOGGERS** : Python モジュールごとにログレベルを再定義します。例 `PATRONI_LOG_LOGGERS="/{patroni.postmaster: WARNING, urllib3: DEBUG}/"`

2.3.3 ブートストラップ構成

新しいクラスターの初期化が成功した直後に新しいデータベースユーザーを作成できます。このプロセスは、次の変数によって定義されます。

- **PATRONI/_<username>/_PASSWORD= '<password>'**
- **PATRONI/_<username>/_OPTIONS= 'list,of,options'**

例: `PATRONI_admin_PASSWORD=strongpasswd` および `PATRONI_admin_OPTIONS='createrole, createdb'` を定義すると、他のユーザーとデータベースの作成が許可されているパスワード `PATRONI_admin_PASSWORD=strongpasswd` でユーザー `PATRONI_admin_PASSWORD=strongpasswd` が作成されます。

2.3.4 Citus

Enables integration Patroni with [Citus](#). If configured, Patroni will take care of registering Citus worker nodes on the coordinator. You can find more information about Citus support [here](#).

- **PATRONI/_CITUS/_GROUP** : Citus グループ ID、整数。コーディネーターには 0 を使用し、ワーカーには 1、2 Etc を使用します
- **PATRONI/_CITUS/_DATABASE** : citus 拡張機能を作成するデータベース。コーディネーターとすべてのワーカーで同じである必要があります。現在、1 つのデータベースのみがサポートされています。

2.3.5 Consul

- **PATRONI/_CONSUL/_HOST** : Consul ローカルエージェントの host:port。
- **PATRONI/_CONSUL/_URL** : Consul ローカルエージェントの URL、形式: http(s)://host:port
- **PATRONI/_CONSUL/_PORT** : (オプション) Consul ポート
- **PATRONI/_CONSUL/_SCHEME** : (オプション) **PATRONI/_CONSUL/_SCHEME** または **PATRONI/_CONSUL/_SCHEME**、デフォルトは **PATRONI/_CONSUL/_SCHEME**
- **PATRONI/_CONSUL/_TOKEN** : (オプション) ACL トークン
- **PATRONI/_CONSUL/_VERIFY** : (オプション) HTTPS リクエストの SSL 証明書を検証するかどうか
- **PATRONI/_CONSUL/_CACERT** : (オプション) CA 証明書。存在する場合、検証が有効になります。
- **PATRONI/_CONSUL/_CERT** : (オプション) クライアント証明書を含むファイル
- **PATRONI/_CONSUL/_KEY** : (オプション) クライアントキーを含むファイル。キーが証明書の一部である場合は空にできます。
- **PATRONI/_CONSUL/_DC** : (オプション) 通信するデータセンター。デフォルトでは、ホストのデータセンターが使用されます。
- **PATRONI/_CONSUL/_CONSISTENCY** : (オプション) Consul 一貫性モードを選択します。可能な値は default、consistent、または stale (詳細は **PATRONI/_CONSUL/_CONSISTENCY_** を参照)
- **PATRONI/_CONSUL/_CHECKS** : (オプション) セッションに使用された Consul ヘルスチェックのリスト。デフォルトでは、空のリストが使用されます。
- **PATRONI/_CONSUL/_REGISTER/_SERVICE** : (オプション) scope パラメーターで定義された名前と、ノードのロールに応じてタグ master、primary、replica、または standby-leader でサービスを登録するかどうか。デフォルトは **PATRONI/_CONSUL/_REGISTER/_SERVICE**
- **PATRONI/_CONSUL/_SERVICE/_TAGS** : (オプション) ロールとは別に Consul サービスに追加する追加の静的タグ (master / primary / replica / standby-leader)。デフォルトでは、空のリストが使用されます。
- **PATRONI/_CONSUL/_SERVICE/_CHECK/_INTERVAL** : (オプション) 登録された URL に対してヘルスチェックを実行する頻度
- **PATRONI/_CONSUL/_SERVICE/_CHECK/_TLS/_SERVER/_NAME** : (オプション) TLS 経由で接続するとき SNI ホストをオーバーライドします。 **PATRONI/_CONSUL/_SERVICE/_CHECK/_TLS/_SERVER/_NAME** も参照してください。

2.3.6 Etcd

- **PATRONI/_ETCD/_PROXY** :etcd のプロキシ URL。プロキシを使用して etcd に接続する場合は、**PATRONI/_ETCD/_PROXY** の代わりにこのパラメーターを使用します
- **PATRONI/_ETCD/_URL** :etcd の URL、形式：`http(s)://(username:password@)host:port`
- **PATRONI/_ETCD/_HOSTS** :形式'`host1:port1`'、'`host2:port2`'Etcd の etcd エンドポイントのリスト...
- **PATRONI/_ETCD/_USE/_PROXIES** : このパラメーターが `true` に設定されている場合、Patroni は **PATRONI/_ETCD/_USE/_PROXIES** をプロキシのリストと見なし、etcd クラスターのトポロジ検出を実行せず、**PATRONI/_ETCD/_USE/_PROXIES** の固定リストに固執します。
- **PATRONI/_ETCD/_PROTOCOL** : `http` または `https`、指定されていない場合 `http` が使用されます。**PATRONI/_ETCD/_PROTOCOL** または **PATRONI/_ETCD/_PROTOCOL** が指定された場合-それからプロトコルを取得します。
- **PATRONI/_ETCD/_HOST** :etcd エンドポイントの `host:port`。
- **PATRONI/_ETCD/_SRV** : クラスター自動検出用に SRV レコードを検索するドメイン。Patroni は、指定されたドメインのこれらの SRV サービス名を（最初に成功するまでこの順序で）クエリしようとします：`_etcd-client-ssl`、`_etcd-client`、`_etcd-ssl`、`_etcd`、`_etcd-server-ssl`、`_etcd-server`。`_etcd-server-ssl` または `_etcd-server` の SRV レコードが取得された場合、ETCD ピアプロトコルが使用され、使用可能なメンバーの ETCD がクエリされます。それ以外の場合、SRV レコードのホストが使用されます。
- **PATRONI/_ETCD/_SRV/_SUFFIX** : 検出中に照会される SRV 名のサフィックスを構成します。このフラグを使用して、同じドメインの下複数の etcd クラスターを区別します。**PATRONI/_ETCD/_SRV/_SUFFIX** と組み合わせてのみ機能します。たとえば、`PATRONI_ETCD_SRV_SUFFIX=foo` および `PATRONI_ETCD_SRV=example.org` が設定されている場合、次の DNSSRV クエリが作成されます。
- **PATRONI/_ETCD/_USERNAME** :etcd 認証のユーザー名。
- **PATRONI/_ETCD/_PASSWORD** :etcd 認証のパスワード。
- **PATRONI/_ETCD/_CACERT** : CA 証明書。存在する場合、検証が有効になります。
- **PATRONI/_ETCD/_CERT** : クライアント証明書を含むファイル。
- **PATRONI/_ETCD/_KEY** : クライアントキーを持つファイル。キーが証明書の一部である場合は空にできます。

2.3.7 Etcdv3

Etcdv3 の環境名は Etcd と似ています。変数名に `ETCD` の代わりに `ETCD3` を使用するだけです。例：`PATRONI_ETCD3_HOST`、`PATRONI_ETCD3_CACERT` Etcd。

警告: プロトコルバージョン 2 で作成されたキーはプロトコルバージョン 3 では表示されず、その逆です。したがって、Patroni 構成を更新するだけで Etcd から Etcdv3 に切り替えることはできません。

2.3.8 ZooKeeper

- **PATRONI/_ZOOKEEPER/_HOSTS** :ZooKeeper クラスターメンバーのコンマ区切りリスト:/"host1:port1','host2:port2','etc.../'"。すべてのエンティティを引用することが重要です！
- **PATRONI/_ZOOKEEPER/_USE/_SSL**:(オプション)SSLを使用するかどうか。デフォルトは `false` です。 `false` に設定されている場合、SSL 固有のパラメーターはすべて無視されます。
- **PATRONI/_ZOOKEEPER/_CACERT**:(オプション) CA 証明書。存在する場合、検証が有効になります。
- **PATRONI/_ZOOKEEPER/_CERT**:(オプション) クライアント証明書を含むファイル。
- **PATRONI/_ZOOKEEPER/_KEY**:(オプション) クライアントキーを含むファイル。
- **PATRONI/_ZOOKEEPER/_KEY/_PASSWORD**:(オプション) クライアントキーのパスワード。
- **PATRONI/_ZOOKEEPER/_VERIFY**:(オプション) 証明書を検証するかどうか。デフォルトは `true` です。
- **PATRONI/_ZOOKEEPER/_SET/_ACLS**:(オプション) 設定されている場合、作成する各 ZNode にデフォルトの ACL を適用するように Kazoo を構成します。ACL は「x509」スキーマを想定し、プリンシパルをキーとし、値のリストとして 1 つ以上の権限を含む辞書として指定する必要があります。パーミッションは、`CREATE`、`READ`、`WRITE`、`DELETE` または `ADMIN` のいずれかです。たとえば、`set_acls: {CN=principal1: [CREATE, READ], CN=principal2: [ALL]}`。

注釈: SSL をサポートするには `kazoo>=2.6.0` をインストールする必要があります。

2.3.9 出展者

- **PATRONI/_EXHIBITOR/_HOSTS** :出展者 (ZooKeeper) ノードの初期リスト: 'host1,host2,etc...'。このリストは、Exhibitor (ZooKeeper) クラスタートポロジが変更されるたびに自動的に更新されます。
- **PATRONI/_EXHIBITOR/_PORT** :出展者ポート。

2.3.10 Kubernetes

- **PATRONI/_KUBERNETES/_BYPASS/_API/_SERVICE** : (オプション) KubernetesAPI と通信するとき、Patroni は通常 `kubernetes` サービスに依存しています。そのアドレスは、`KUBERNETES_SERVICE_HOST` 環境変数を介してポッドで公開されます。`PATRONI\_KUBERNETES\_BYPASS\_API\_SERVICE` が `true` に設定されている場合、パトロニはサービスの背後にある API ノードのリストを解決し、それらに直接接続します。
- **PATRONI/_KUBERNETES/_NAMESPACE**:(オプション) Patroni ポッドが実行されている Kubernetes 名前空間。デフォルト値は `default` です。

- **PATRONI/_KUBERNETES/_LABELS** : 形式 {label1: value1, label2: value2} のラベル。これらのラベルは、現在のクラスターに関連付けられている既存のオブジェクト（Pod と Endpoints または ConfigMap のいずれか）を見つけるために使用されます。また、Patroni は、作成するすべてのオブジェクト（Endpoint または ConfigMap）にそれらを設定します。
- **PATRONI/_KUBERNETES/_SCOPE/_LABEL** : (オプション) クラスター名を含むラベルの名前。デフォルト値は `cluster-name` です。
- **PATRONI/_KUBERNETES/_ROLE/_LABEL** : (オプション) ロール（マスターまたはレプリカまたはその他のカスタム値）を含むラベルの名前。Patroni は、実行される Pod にこのラベルを設定します。デフォルト値は `role` です。
- **PATRONI/_KUBERNETES/_LEADER/_LABEL/_VALUE** : (オプション) Postgres ロールが `master` の場合のポッドラベルの値。デフォルト値は `master` です。
- **PATRONI/_KUBERNETES/_FOLLOWER/_LABEL/_VALUE** : (オプション) Postgres ロールが `replica` の場合のポッドラベルの値。デフォルト値は `replica` です。
- **PATRONI/_KUBERNETES/_STANDBY/_LEADER/_LABEL/_VALUE** : (オプション) Postgres ロールが `standby-leader` の場合のポッドラベルの値。デフォルト値は `standby-leader` です。
- **PATRONI/_KUBERNETES/_TMP/_ROLE/_LABEL** : (オプション) ロール（マスターまたはレプリカ）を含む一時ラベルの名前。このラベルの値は、常に対応するロールのデフォルトを使用します。必要な場合のみ設定してください。
- **PATRONI/_KUBERNETES/_USE/_ENDPOINTS** : (オプション) `true` に設定されている場合、Patroni は ConfigMap の代わりに Endpoints を使用して、リーダー選挙を実行し、クラスターの状態を維持します。
- **PATRONI/_KUBERNETES/_POD/_IP** : (オプション) パトロニが実行されているポッドの IP アドレス。この値は `PATRONI\_KUBERNETES\_USE\_ENDPOINTS` が有効な場合に必須であり、ポッドの PostgreSQL が昇格するときにリーダーエンドポイントサブセットを設定するために使用されます。
- **PATRONI/_KUBERNETES/_PORTS** : (オプション) サービスオブジェクトにポートの名前がある場合、同じ名前が Endpoint オブジェクトに表示される必要があります。そうでない場合、サービスは動作しません。たとえば、サービスが {Kind: Service, spec: {ports: [{name: postgresql, port: 5432, targetPort: 5432}]}} として定義されている場合、`PATRONI_KUBERNETES_PORTS='[{"name/": "/postgresql/", "port/": 5432}]'` を設定する必要があります。Patroni はそれを使用してリーダーエンドポイントのサブセットを更新します。このパラメータは、`PATRONI\_KUBERNETES\_USE\_ENDPOINTS` が設定されている場合にのみ使用されます。
- **PATRONI/_KUBERNETES/_CACERT** : (オプション) KubernetesAPISSL 証明書の検証中に使用する信頼できる CA の証明書を含む CA_BUNDLE ファイルを含むファイルを指定します。提供されない場合、Patroni は ServiceAccount シークレットによって提供される値を使用します。
- **PATRONI/_RETRIABLE/_HTTP/_CODES** : (オプション) リトライする K8sAPI からの HTTP ステータスコードのリスト。デフォルトでは、Patroni は 500、503、および 504 で再試行しています、または K8sAPI 応答に `retry-after` HTTP ヘッダーがある場合。

2.3.11 Raft (非推奨)

- **PATRONI/_RAFT/_SELF/_ADDR** : Raft 接続をリッスンする ip:port。self_addr は、クラスターの他のノードからアクセスできる必要があります。設定されていない場合、ノードはコンセンサスに参加しません。
- **PATRONI/_RAFT/_BIND/_ADDR** : (オプション) Raft 接続をリッスンする ip:port。指定しない場合、self_addr が使用されます。
- **PATRONI/_RAFT/_PARTNER/_ADDRESSES** : `/'ip1:port1','ip2:port2'/'` 形式のクラスター内の他の Patroni ノードのリスト。すべてのエンティティを引用することが重要です！
- **PATRONI/_RAFT/_DATA/_DIR** : Raft ログとスナップショットを保存するディレクトリ。指定しない場合、現在の作業ディレクトリが使用されます。
- **PATRONI/_RAFT/_PASSWORD** : (オプション) 指定されたパスワードで Raft トラフィックを暗号化します。cryptography python モジュールが必要です。

2.3.12 PostgreSQL

- **PATRONI/_POSTGRES/_LISTEN** : Postgres がリッスンする IP アドレス+ポート。ポートコンポーネントが最後のものの後にコロンで追加されている限り、複数のコンマ区切りアドレスが許可されます、つまり listen: 127.0.0.1,127.0.0.2:5432。Patroni はこのリストの最初のアドレスを使用して、PostgreSQL ノードへのローカル接続を確立します。
- **PATRONI/_POSTGRES/_CONNECT/_ADDRESS** : IP アドレス+他のノードやアプリケーションから Postgres にアクセスするためのポート。
- **PATRONI/_POSTGRES/_PROXY/_ADDRESS** : Postgres の隣で実行されている接続プール (例 pgbouncer) にアクセスできる IP アドレス+ポート。値は DCS のメンバーキーに proxy_url として書き込まれ、サービス検出に使用/役立つ可能性があります。
- **PATRONI/_POSTGRES/_DATA/_DIR** : Postgres データディレクトリの場所。既存または Patroni によって初期化されます。
- **PATRONI/_POSTGRES/_CONFIG/_DIR** : Postgres 構成ディレクトリの場所、デフォルトはデータディレクトリ。Patroni が書き込み可能である必要があります。
- **PATRONI/_POSTGRES/_BIN/_DIR** : PostgreSQL バイナリへのパス。(pg_ctl, initdb, pg_controldata, pg_basebackup, postgres, pg_isready, pg_rewind) デフォルト値は空の文字列であり、PATH 環境変数を使用して実行可能ファイルを見つけます。
- **PATRONI/_POSTGRES/_BIN/_PG/_CTL** : (オプション) pg_ctl バイナリのカスタム名。
- **PATRONI/_POSTGRES/_BIN/_INITDB** : (オプション) initdb バイナリのカスタム名。
- **PATRONI/_POSTGRES/_BIN/_PG/_CONTROLDATA** : (オプション) pg_controldata バイナリのカスタム名。
- **PATRONI/_POSTGRES/_BIN/_PG/_BASEBACKUP** : (オプション) pg_basebackup バイナリのカスタム名。

- **PATRONI/_POSTGRESQL/_BIN/_POSTGRES** : (オプション) postgres バイナリのカスタム名。
- **PATRONI/_POSTGRESQL/_BIN/_IS/_READY** : (オプション) pg_isready バイナリのカスタム名。
- **PATRONI/_POSTGRESQL/_BIN/_PG/_REWIND** : (オプション) pg_rewind バイナリのカスタム名。
- **PATRONI/_POSTGRESQL/_PGPASS** : **PATRONI/_POSTGRESQL/_PGPASS** _パスワードファイルへのパス。Patroni は、pg/_basebackup を実行する前に、およびその他の状況下でこのファイルを作成します。場所は Patroni が書き込み可能である必要があります。
- **PATRONI/_REPLICATION/_USERNAME** : レプリケーションユーザー名; ユーザーは初期化中に作成されます。レプリカはこのユーザーを使用して、ストリーミングレプリケーションを介してレプリケーションソースにアクセスします
- **PATRONI/_REPLICATION/_PASSWORD** : レプリケーションパスワード。ユーザーは初期化中に作成されます。
- **PATRONI/_REPLICATION/_SSLMODE** : (オプション) **PATRONI/_REPLICATION/_SSLMODE** _接続パラメーターにマップし、クライアントがサーバーとの TLS ネゴシエーションモードのタイプを指定できるようにします。各モードの仕組みの詳細については、**PATRONI/_REPLICATION/_SSLMODE** _をご覧ください。デフォルトモードは prefer です。
- **PATRONI/_REPLICATION/_SSLKEY** : (オプション) **PATRONI/_REPLICATION/_SSLKEY** _接続パラメーターにマップし、クライアントの証明書で使用する秘密鍵の場所を指定します。
- **PATRONI/_REPLICATION/_SSLPASSWORD** : (オプション) **PATRONI/_REPLICATION/_SSLPASSWORD** _接続パラメーターにマップし、**PATRONI/_REPLICATION/_SSLKEY** で指定された秘密鍵のパスワードを指定します。
- **PATRONI/_REPLICATION/_SSLCERT** : (オプション) クライアント証明書の場所を指定する **PATRONI/_REPLICATION/_SSLCERT** _接続パラメーターにマップします。
- **PATRONI/_REPLICATION/_SSLROOTCERT** : (オプション) **PATRONI/_REPLICATION/_SSLROOTCERT** _接続パラメーターにマップします。これは、クライアントがサーバーの証明書を検証するために使用する 1 つ以上の認証局 (CA) 証明書を含むファイルの場所を指定します。
- **PATRONI/_REPLICATION/_SSLCRL** : (オプション) **PATRONI/_REPLICATION/_SSLCRL** _接続パラメーターにマップし、証明書失効リストを含むファイルの場所を指定します。クライアントは、このリストに証明書が存在するサーバーへの接続を拒否します。
- **PATRONI/_REPLICATION/_SSLCRLDIR** : (オプション) **PATRONI/_REPLICATION/_SSLCRLDIR** _接続パラメーターにマップし、証明書失効リストを含むファイルがあるディレクトリの場所を指定します。クライアントは、このリストに証明書が存在するサーバーへの接続を拒否します。
- **PATRONI/_REPLICATION/_GSSENCMODE** : (オプション) **PATRONI/_REPLICATION/_GSSENCMODE** _接続パラメーターにマップします。これは、安全な GSSTCP/IP 接続をサーバーとネゴシエートするかどうか、またはどの優先度でネゴシエートするかを決定します
- **PATRONI/_REPLICATION/_CHANNEL/_BINDING** : (オプション) **PATRONI/_REPLICATION/_CHANNEL/_BINDING** _接続パラメーターにマップし、クライア

ントによるチャンネルバインディングの使用を制御します。

- **PATRONI/_SUPERUSER/_USERNAME** : スーパーユーザーの名前、初期化 (initdb) 中に設定され、後で Patroni が postgres に接続するために使用します。また、このユーザーは pg_rewind によって使用されます。
- **PATRONI/_SUPERUSER/_PASSWORD** : スーパーユーザーのパスワード、初期化 (initdb) 中に設定。
- **PATRONI/_SUPERUSER/_SSLMODE** : (オプション) **PATRONI/_SUPERUSER/_SSLMODE** 接続パラメーターにマップし、クライアントがサーバーとの TLS ネゴシエーションモードのタイプを指定できるようにします。各モードの仕組みの詳細については、**PATRONI/_SUPERUSER/_SSLMODE** をご覧ください。デフォルトモードは prefer です。
- **PATRONI/_SUPERUSER/_SSLKEY** : (オプション) **PATRONI/_SUPERUSER/_SSLKEY** 接続パラメーターにマップし、クライアントの証明書で使用する秘密鍵の場所を指定します。
- **PATRONI/_SUPERUSER/_SSLPASSWORD** : (オプション) **PATRONI/_SUPERUSER/_SSLPASSWORD** 接続パラメーターにマップし、**PATRONI/_SUPERUSER/_SSLKEY** で指定された秘密鍵のパスワードを指定します。
- **PATRONI/_SUPERUSER/_SSLCERT** : (オプション) クライアント証明書の場所を指定する **PATRONI/_SUPERUSER/_SSLCERT** 接続パラメーターにマップします。
- **PATRONI/_SUPERUSER/_SSLROOTCERT** : (オプション) **PATRONI/_SUPERUSER/_SSLROOTCERT** 接続パラメーターにマップします。これは、クライアントがサーバーの証明書を検証するために使用する 1 つ以上の認証局 (CA) 証明書を含むファイルの場所を指定します。
- **PATRONI/_SUPERUSER/_SSLCRL** : (オプション) **PATRONI/_SUPERUSER/_SSLCRL** 接続パラメーターにマップし、証明書失効リストを含むファイルの場所を指定します。クライアントは、このリストに証明書が存在するサーバーへの接続を拒否します。
- **PATRONI/_SUPERUSER/_SSLCRLDIR** : (オプション) **PATRONI/_SUPERUSER/_SSLCRLDIR** 接続パラメーターにマップし、証明書失効リストを含むファイルがあるディレクトリの場所を指定します。クライアントは、このリストに証明書が存在するサーバーへの接続を拒否します。
- **PATRONI/_SUPERUSER/_GSSENCMODE** : (オプション) **PATRONI/_SUPERUSER/_GSSENCMODE** 接続パラメーターにマップします。これは、安全な GSSTCP/IP 接続をサーバーとネゴシエートするかどうか、またはどの優先度でネゴシエートするかを決定します
- **PATRONI/_SUPERUSER/_CHANNEL/_BINDING** : (オプション) **PATRONI/_SUPERUSER/_CHANNEL/_BINDING** 接続パラメーターにマップし、クライアントによるチャンネルバインディングの使用を制御します。
- **PATRONI/_REWIND/_USERNAME** : (オプション) pg_rewind のユーザーの名前。ユーザーは postgres11+ の初期化中に作成され、必要なすべての **PATRONI/_REWIND/_USERNAME** が付与されます。
- **PATRONI/_REWIND/_PASSWORD** : (オプション) pg_rewind のユーザーのパスワード。ユーザーは初期化中に作成されます。
- **PATRONI/_REWIND/_SSLMODE** : (オプション) **PATRONI/_REWIND/_SSLMODE** 接続パラメーターにマップし、クライアントがサーバーとの TLS ネゴシエーションモードのタイプを指定で

きるようにします。各モードの仕組みの詳細については、**PATRONI/_REWIND/_SSLMODE** _をご覧ください。デフォルトモードは **prefer** です。

- **PATRONI/_REWIND/_SSLKEY** : (オプション) **PATRONI/_REWIND/_SSLKEY** _接続パラメーターにマップし、クライアントの証明書で使用する秘密鍵の場所を指定します。
- **PATRONI/_REWIND/_SSLPASSWORD** : (オプション) **PATRONI/_REWIND/_SSLPASSWORD** _接続パラメーターにマップし、**PATRONI/_REWIND/_SSLKEY** で指定された秘密鍵のパスワードを指定します。
- **PATRONI/_REWIND/_SSLCERT** : (オプション) クライアント証明書の場所を指定する **PATRONI/_REWIND/_SSLCERT** _接続パラメーターにマップします。
- **PATRONI/_REWIND/_SSLROOTCERT** : (オプション) **PATRONI/_REWIND/_SSLROOTCERT** _接続パラメーターにマップします。これは、クライアントがサーバーの証明書を検証するために使用する 1 つ以上の認証局 (CA) 証明書を含むファイルの場所を指定します。
- **PATRONI/_REWIND/_SSLCRL** : (オプション) **PATRONI/_REWIND/_SSLCRL** _接続パラメーターにマップし、証明書失効リストを含むファイルの場所を指定します。クライアントは、このリストに証明書が存在するサーバーへの接続を拒否します。
- **PATRONI/_REWIND/_SSLCRLDIR** : (オプション) **PATRONI/_REWIND/_SSLCRLDIR** _接続パラメーターにマップし、証明書失効リストを含むファイルがあるディレクトリの場所を指定します。クライアントは、このリストに証明書が存在するサーバーへの接続を拒否します。
- **PATRONI/_REWIND/_GSSENCMODE** : (オプション) **PATRONI/_REWIND/_GSSENCMODE** _接続パラメーターにマップします。これは、安全な GSSTCP/IP 接続をサーバーとネゴシエートするかどうか、またはどの優先度でネゴシエートするかを決定します
- **PATRONI/_REWIND/_CHANNEL/_BINDING** : (オプション) **PATRONI/_REWIND/_CHANNEL/_BINDING** _接続パラメーターにマップし、クライアントによるチャンネルバインディングの使用を制御します。

2.3.13 RESTAPI

- **PATRONI/_RESTAPI/_CONNECT/_ADDRESS** : RESTAPI にアクセスするための IP アドレスとポート。
- **PATRONI/_RESTAPI/_LISTEN** : Patroni がリッスンする IP アドレスとポート。HAProxy のヘルスチェック情報を提供します。
- **PATRONI/_RESTAPI/_USERNAME** : 安全でない RESTAPI エンドポイントを保護するための基本認証ユーザー名。
- **PATRONI/_RESTAPI/_PASSWORD** : 安全でない RESTAPI エンドポイントを保護するための基本認証パスワード。
- **PATRONI/_RESTAPI/_CERTFILE** : PEM 形式の証明書を含むファイルを指定します。certfile を指定しないか空の場合、API サーバーは SSL なしで動作します。
- **PATRONI/_RESTAPI/_KEYFILE** : PEM 形式の秘密鍵を含むファイルを指定します。

- **PATRONI/_RESTAPI/_KEYFILE/_PASSWORD** : キーファイルを復号化するためのパスワードを指定します。
- **PATRONI/_RESTAPI/_CAFILE** : クライアント証明書の検証中に使用する信頼できる CA の証明書を含む CA_BUNDLE を持つファイルを指定します。
- **PATRONI/_RESTAPI/_CIPHERS** : (オプション) 許可される暗号スイートを指定します (例/"ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256:!SSLv1:!SSLv2:!SSLv3:!TLSv1:!TLSv1.1/")
- **PATRONI/_RESTAPI/_VERIFY/_CLIENT** : none (デフォルト)、optional または required。none RESTAPI の場合、クライアント証明書はチェックされません。すべての RESTAPI 呼び出しに required クライアント証明書が必要な場合。すべての安全でない RESTAPI エンドポイントに optional クライアント証明書が必要な場合。required が使用されている場合、証明書の署名の検証が成功すると、クライアント認証が成功します。optional の場合、クライアント証明書は PUT、POST、PATCH、および DELETE 要求に対してのみチェックされます。
- **PATRONI/_RESTAPI/_ALLOWLIST** : (オプション): 安全でない RESTAPI エンドポイントの呼び出しを許可するホストのセットを指定します。単一の要素は、ホスト名、IP アドレス、または CIDR 表記を使用したネットワークアドレスになります。デフォルトでは allow_all が使用されます。allowlist または allowlist_include_members が設定されている場合、含まれていないものはすべて拒否されます。
- **PATRONI/_RESTAPI/_ALLOWLIST/_INCLUDE/_MEMBERS** : (オプション): true に設定すると、DCS に登録されている他のクラスターメンバーから安全でない RESTAPI エンドポイントにアクセスできます (IP アドレスまたはホスト名はメンバー api_url から取得されます)。OS が送信接続に別の IP を使用する可能性があることに注意してください。
- **PATRONI/_RESTAPI/_HTTP/_EXTRA/_HEADERS** : (オプション) HTTP ヘッダーにより、RESTAPI サーバーは HTTP 応答で追加情報を渡すことができます。
- **PATRONI/_RESTAPI/_HTTPS/_EXTRA/_HEADERS** : (オプション) HTTPS ヘッダーにより、RESTAPI サーバーは、TLS が有効になっている場合に HTTP 応答で追加情報を渡すことができます。これにより、http_extra_headers に設定された追加情報も渡されます。
- **PATRONI/_RESTAPI/_REQUEST/_QUEUE/_SIZE** : (オプション): PatroniRESTAPI が使用する TCP ソケットのリクエストキューサイズを設定します。キューがいっぱいになると、それ以降の要求には「接続が拒否されました」エラーが発生します。デフォルト値は 5 です。

警告:

- PATRONI_RESTAPI_CONNECT_ADDRESS は、特定の Patroni クラスターのすべてのノードからアクセスできる必要があります。内部では、Patroni はリーダーレース中にそれを使用して、レプリケーションラグが最小限に抑えられたノードを見つけています。
- クライアント証明書の検証を有効にした場合 (PATRONI_RESTAPI_VERIFY_CLIENT が required に設定されている場合)、PATRONI_RESTAPI_VERIFY_CLIENT も PATRONI_CTL_CERTFILE 、 PATRONI_CTL_KEYFILE 、 PATRONI_CTL_KEYFILE_PASSWORD で PATRONI_RESTAPI_VERIFY_CLIENT を提供します。提供されていない場合、Patroni は正しく動作しません。

2.3.14 CTL

- **PATRONICTL/_CONFIG/_FILE** : (オプション) 構成ファイルの場所。
- **PATRONI/_CTL/_USERNAME** : (オプション) 保護された RESTAPI エンドポイントにアクセスするための基本認証ユーザー名。提供されない場合、patanictl は RESTAPI/"username/"パラメーターに提供された値を使用します。
- **PATRONI/_CTL/_PASSWORD** : (オプション) 保護された RESTAPI エンドポイントにアクセスするための基本認証パスワード。提供されない場合、patanictl は RESTAPI/"password/"パラメーターに提供された値を使用します。
- **PATRONI/_CTL/_INSECURE** : (オプション) SSL 証明書を検証せずに RESTAPI への接続を許可します。
- **PATRONI/_CTL/_CACERT** : (オプション) RESTAPISSL 証明書の検証中に使用する、信頼できる CA の証明書を含む CA_BUNDLE ファイルまたはディレクトリを含むファイルを指定します。提供されない場合、patanictl は RESTAPI/"cafile/"パラメーターに提供された値を使用します。
- **PATRONI/_CTL/_CERTFILE** : (オプション) クライアント証明書を含むファイルを PEM 形式で指定します。
- **PATRONI/_CTL/_KEYFILE** : (オプション) クライアント秘密鍵を含むファイルを PEM 形式で指定します。
- **PATRONI/_CTL/_KEYFILE/_PASSWORD** : (オプション) クライアントキーファイルを復号化するためのパスワードを指定します。

Patroni 構成には 3 種類あります。

- グローバル *dynamic configuration* 。

これらのオプションは DCS (分散構成ストア) に保存され、すべてのクラスターノードに適用されます。動的構成は、`patronictl edit-config tool` または `Patroni patronictl edit-config` を使用して、いつでも設定できます。変更されたオプションがスタートアップ構成の一部でない場合、それらは非同期に (次のウェイクアップサイクル時に) すべてのノードに適用され、続いてリロードされます。ノードが構成を適用するために再起動が必要な場合 (コンテキスト `postmaster` を使用する `patronictl edit-config_` の場合、値が変更された場合)、これを示す特別なフラグ `pending_restart members.dataJSON`。さらに、ノードのステータスは `/"restart_pending/"`: `true` を表示することにより、これを示します。

- ローカル *configuration file* (`patroni.yml`)。

これらのオプションは構成ファイルで定義され、動的構成よりも優先されます。`patroni.yml` は、SIGHUP を Patroni プロセスに送信するか、`POST /reload` REST-API 要求を実行するか、`patronictl reload` を実行することにより、実行時に (Patroni を再起動せずに) 変更してリロードできます。ローカル構成は、単一の YAML ファイルまたはディレクトリのいずれかです。ディレクトリの場合、そのディレクトリ内のすべての YAML ファイルがソート順に 1 つずつ読み込まれます。キーが複数のファイルで定義されている場合、最後のファイルでの出現が優先されます。

- *Environment configuration* 。

環境変数を使用して、`/"Local/"` 構成パラメーターの一部を設定/オーバーライドできます。環境

構成は、動的環境で実行していて、一部のパラメーターが事前にわからない場合に非常に役立ちます（たとえば、`docker` 内で実行しているときに外部 IP アドレスを知ることはできません）。

2.4 重要なルール

2.4.1 Patroni が制御する PostgreSQL パラメーター

PostgreSQL パラメーターの一部 **must hold the same values on the primary and the replicas**。それらの場合、**must hold the same values on the primary and the replicas**。それらの値を変更または設定するには、DCS の共有構成を変更する必要があります。以下は、そのようなパラメーターの実際のリストとデフォルト値です。

- **max_connections** : 100
- **max_locks_per_transaction** : 64
- **max_worker_processes** : 8
- **max_prepared_transactions** : 0
- **wal_level** : hot_standby
- **track_commit_timestamp** : オフ

以下のパラメーターについて、PostgreSQL は、プライマリとすべてのレプリカで同じ値を必要としません。ただし、レプリカがいつでもプライマリになる可能性を考慮すると、それらを異なる方法で設定することは実際には意味がありません。したがって、**Patroni restricts setting their values to the Patroni restricts setting their values to the**。

- **max_wal_senders** : 5
- **max_replication_slots** : 5
- **wal_keep_segments** : 8
- **wal_keep_size** : 128MB

これらのパラメーターは、正常であること、または最小値を満たしていることを確認するために検証されます。

Patroni によって制御される他の Postgres パラメーターがいくつかあります。

- **listen_addresses** - `postgresql.listen` または `PATRONI_POSTGRESQL_LISTEN` 環境変数から設定されます
- **port** - `postgresql.listen` または `PATRONI_POSTGRESQL_LISTEN` 環境変数から設定されます
- **cluster_name** - `scope` または `PATRONI_SCOPE` 環境変数から設定されます
- **hot_standby**: on
- **wal_log_hints**: on -Postgres9.4 以降の場合。

安全な側にあるために、上記のリストのパラメーターは `postgresql.conf` に書き込まれませんが、引数のリストとして `pg_ctl start` に渡されます。

`postgresql.listen`、`postgresql.listen`、`postgresql.listen` のようないくつかのパラメーターもあります、つまり Patroni `postgresql.listen` 内または `postgresql.listen` 変数を介して。ほとんどの場合、ローカル構成は動的構成をオーバーライドします。

ローカルまたは動的構成オプションを適用すると、次のアクションが実行されます。

- ノードは最初に ``postgresql.base.conf`` があるかどうか、または `custom_conf` パラメータが設定されているかどうかを確認します。
- `custom_conf` パラメーターが設定されている場合、指定されたファイルを基本構成として使用し、``postgresql.base.conf`` および ``postgresql.conf`` を無視します。
- `custom_conf` パラメータが設定されておらず、``postgresql.base.conf`` が存在する場合、名前変更された `"/original/"` 構成が含まれ、基本構成として使用されます。
- `custom_conf` も ``postgresql.base.conf`` もない場合、元の ``postgresql.conf`` が取得され、`postgresql.base.conf` に名前が変更されます。
- 動的オプション（上記の例外を除く）は ``postgresql.conf`` and an include is set in `postgresql.conf` to the used base configuration (either `postgresql.base.conf` or what is on ``custom_conf``) にダンプされます。したがって、構成ファイルを再度読み込んでインクルードが存在しないかどうかを確認することなく、新しいオプションを適用できます。
- Patroni がクラスターを管理するために不可欠ないくつかのパラメーターは、コマンドラインを使用してオーバーライドされます。
- 再起動が必要なオプションの一部が変更された場合（`pg_settings` のコンテキストとこれらのオプションの実際の値を確認する必要があります）、特定のノードの `pending_restart` フラグが設定されます。このフラグは、再起動時にリセットされます。

パラメーターは次の順序で適用されます（実行時間が最も優先されます）。

1. パラメータをファイル ``postgresql.base.conf`` から（または設定されている場合は `custom_conf` ファイルから）ロードします
2. ファイル ``postgresql.conf`` からパラメータをロードします
3. ファイル ``postgresql.auto.conf`` からパラメータをロードします
4. ``-o--name=value`` を使用した実行時パラメータ

これにより、すべてのノードの構成 (2)、`ALTER SYSTEM` (3) を使用した特定のノードの構成が可能になり、Patroni の実行に不可欠なパラメーターが強制され (4)、`postgresql` を管理する構成ツール用のスペースが確保されます。Patroni を介さずに `conf` 直接 (1)。

2.4.2 Patroni 構成パラメーター

また、次の Patroni 構成オプション **can be changed only dynamically** :

- `tli` : 30
- `loop_wait` : 10
- `retry_timeouts` : 10
- `maximum_lag_on_failover` : 1048576
- `max_timelines_history` : 0
- `check_timeline` : false
- `postgresql.use_slots` : true

これらのオプションを変更すると、Patroni は DCS に保存されている構成の関連セクションを読み取り、その実行時の値を変更します。

Patroni ノードは、Postgres データディレクトリにあるファイル `patroni.dynamic.json` に構成が変更されるたびに、DCS オプションの状態をディスクにダンプしています。これらのオプションが DCS に完全に存在しない場合、または無効な場合、リーダーのみがオンディスクダンプからこれらのオプションを復元できます。

第3章 PatroniRESTAPI

Patroni には、リーダーレース中に Patroni 自分自身が使用する豊富な RESTAPI があります。、そしてもちろんモニタリングにも使用できます。以下に、PatroniRESTAPI エンドポイントのリストを示します。

3.1 ヘルスチェックエンドポイント

すべてのヘルスチェック GET リクエストについて、Patroni は、ノードのステータスと HTTP ステータスコードを含む JSON ドキュメントを返します。JSON ドキュメントが必要ない、または必要ない場合は、GET の代わりに HEAD または OPTIONS メソッドの使用を検討してください。

- PatroniRESTAPI への次の要求は、Patroni ノードがリーダーロックでプライマリとして実行されている場合にのみ、HTTP ステータスコード **200** を返します。
 - GET /
 - GET /primary
 - GET /read-write
- GET /standby-leader : Patroni ノードが GET /standby-leader リーダーとして実行されている場合にのみ、HTTP ステータスコード GET /standby-leader を返します。
- GET /leader : Patroni ノードがリーダーロックを持っている場合、HTTP ステータスコード GET /leader を返します。以前の 2 つのエンドポイントとの主な違いは、PostgreSQL が **primary** または **standby_leader** として実行されているかどうかを考慮しないことです。
- GET /replica : レプリカヘルスチェックエンドポイント。Patroni ノードの状態が **running**、ロールが **replica**、**noloadbalance** タグが設定されていない場合にのみ、HTTP ステータスコード GET /replica を返します。
- GET /replica?lag=<max-lag> : レプリカチェックエンドポイント。**replica** からのチェックに加えて、レプリケーションレイテンシーもチェックし、指定された値を下回る場合にのみステータスコード GET /replica?lag=<max-lag> を返します。DCS のキー `cluster.last_leader_operation` は、パフォーマンス上の理由から、リーダーウォールの位置とレプリカでのレイテンシーの計算に使用されます。`max-lag` は、バイト（整数）または人間が読める値で指定できます。例 16kB、64MB、1GB。
 - GET /replica?lag=1048576
 - GET /replica?lag=1024kB
 - GET /replica?lag=10MB
 - GET /replica?lag=1GB

- GET /replica?tag_key1=value1&tag_key2=value2 : レプリカチェックエンドポイント。さらに、ユーザー定義タグ key1 および key2 、および yaml 構成管理の GET /replica?tag_key1=value1&tag_key2=value2 セクション内のそれぞれの値もチェックします。インスタンスに対してタグが定義されていない場合、または yaml 構成の値がクエリ値と一致しない場合、HTTP ステータスコード 503 が返されます。

以下のリクエストでは、リーダーまたはスタンバイリーダーのステータスを確認しているため、Patroni はユーザ定義のタグを適用せず、無視されます。

- GET /?tag_key1=value1&tag_key2=value2
 - GET /leader?tag_key1=value1&tag_key2=value2
 - GET /primary?tag_key1=value1&tag_key2=value2
 - GET /read-write?tag_key1=value1&tag_key2=value2
 - GET /standby_leader?tag_key1=value1&tag_key2=value2
 - GET /standby-leader?tag_key1=value1&tag_key2=value2
- GET /read-only : 上記のエンドポイントに似ていますが、プライマリも含まれます。
 - GET /synchronous または GET /sync : Patroni ノードが同期スタンバイとして実行されている場合にのみ、HTTP ステータスコード GET /synchronous を返します。
 - GET /read-only-sync : 上記のエンドポイントに似ていますが、プライマリも含まれます。
 - GET /asynchronous または GET /async : Patroni ノードが非同期スタンバイとして実行されている場合にのみ、HTTP ステータスコード GET /asynchronous を返します。
 - GET /asynchronous?lag=<max-lag> または GET /async?lag=<max-lag> : 非同期スタンバイチェックエンドポイント。 asynchronous または async からのチェックに加えて、レプリケーションレイテンシーもチェックし、指定された値を下回る場合にのみステータスコード GET /asynchronous?lag=<max-lag> を返します。DCS のキー cluster.last_leader_operation は、パフォーマンス上の理由から、リーダーウォールの位置とレプリカでのレイテンシーの計算に使用されます。max-lag は、バイト（整数）または人間が読める値で指定できます。例 16kB、64MB、1GB。
 - GET /async?lag=1048576
 - GET /async?lag=1024kB
 - GET /async?lag=10MB
 - GET /async?lag=1GB
 - GET /health : PostgreSQL が稼働している場合にのみ HTTP ステータスコード GET /health を返します。
 - GET /liveness : Patroni ハートビートループが適切に実行されている場合は HTTP ステータスコード GET /liveness を返し、最後の実行がプライマリで ttl 秒以上前であった場合は GET /liveness 、レプリカで 2*ttl を返します。livenessProbe に使用できます。
 - GET /readiness : Patroni ノードがリーダーとして実行されている場合、または PostgreSQL が起動して実行されている場合、HTTP ステータスコード GET /readiness を返します。リーダー選挙

(OpenShift) に Kubernetes エンドポイントを使用できない場合、エンドポイントを `readinessProbe` に使用できます。

`readiness` および `liveness` エンドポイントは両方とも非常に軽量で、SQL を実行しません。リーダーキーの有効期限が切れる頃に失敗し始めるように、プローブを構成する必要があります。30s の例である `ttl` のデフォルト値を使用すると、プローブは次のようになります。

```
readinessProbe:
  httpGet:
    scheme: HTTP
    path: /readiness
    port: 8008
  initialDelaySeconds: 3
  periodSeconds: 10
  timeoutSeconds: 5
  successThreshold: 1
  failureThreshold: 3
livenessProbe:
  httpGet:
    scheme: HTTP
    path: /liveness
    port: 8008
  initialDelaySeconds: 3
  periodSeconds: 10
  timeoutSeconds: 5
  successThreshold: 1
  failureThreshold: 3
```

3.2 モニタリングエンドポイント

GET `/patroni` は、リーダーレース中にパトローニが使用します。また、監視システムで使用することもできます。このエンドポイントによって生成される JSON ドキュメントは、ヘルスチェックエンドポイントによって生成される JSON と同じ構造を持っています。

Example: 健全なクラスター

```
$ curl -s http://localhost:8008/patroni | jq .
{
  "state": "running",
  "postmaster_start_time": "2023-08-18 11:03:37.966359+00:00",
  "role": "master",
  "server_version": 150004,
  "xlog": {
    "location": 67395656
  },
}
```

(次のページに続く)

(前のページからの続き)

```

"timeline": 1,
"replication": [
  {
    "username": "replicator",
    "application_name": "patroni2",
    "client_addr": "10.89.0.6",
    "state": "streaming",
    "sync_state": "async",
    "sync_priority": 0
  },
  {
    "username": "replicator",
    "application_name": "patroni3",
    "client_addr": "10.89.0.2",
    "state": "streaming",
    "sync_state": "async",
    "sync_priority": 0
  }
],
"dcs_last_seen": 1692356718,
"tags": {
  "clonefrom": true
},
"database_system_identifier": "7268616322854375442",
"patroni": {
  "version": "3.1.0",
  "scope": "demo"
}
}

```

Example: ロック解除されたクラスター

```

$ curl -s http://localhost:8008/patroni | jq .
{
  "state": "running",
  "postmaster_start_time": "2023-08-18 11:09:08.615242+00:00",
  "role": "replica",
  "server_version": 150004,
  "xlog": {
    "received_location": 67419744,
    "replayed_location": 67419744,
    "replayed_timestamp": null,
    "paused": false
  },
  "timeline": 1,

```

(次のページに続く)

(前のページからの続き)

```

"replication": [
  {
    "username": "replicator",
    "application_name": "patroni2",
    "client_addr": "10.89.0.6",
    "state": "streaming",
    "sync_state": "async",
    "sync_priority": 0
  },
  {
    "username": "replicator",
    "application_name": "patroni3",
    "client_addr": "10.89.0.2",
    "state": "streaming",
    "sync_state": "async",
    "sync_priority": 0
  }
],
"cluster_unlocked": true,
"dcs_last_seen": 1692356928,
"tags": {
  "clonefrom": true
},
"database_system_identifier": "7268616322854375442",
"patroni": {
  "version": "3.1.0",
  "scope": "demo"
}
}

```

Example: Example: が有効になっているロック解除されたクラスター

```

$ curl -s http://localhost:8008/patroni | jq .
{
  "state": "running",
  "postmaster_start_time": "2023-08-18 11:09:08.615242+00:00",
  "role": "replica",
  "server_version": 150004,
  "xlog": {
    "location": 67420024
  },
  "timeline": 1,
  "replication": [
    {
      "username": "replicator",

```

(次のページに続く)

(前のページからの続き)

```

    "application_name": "patroni2",
    "client_addr": "10.89.0.6",
    "state": "streaming",
    "sync_state": "async",
    "sync_priority": 0
  },
  {
    "username": "replicator",
    "application_name": "patroni3",
    "client_addr": "10.89.0.2",
    "state": "streaming",
    "sync_state": "async",
    "sync_priority": 0
  }
],
"cluster_unlocked": true,
"failsafe_mode_is_active": true,
"dcs_last_seen": 1692356928,
"tags": {
  "clonefrom": true
},
"database_system_identifier": "7268616322854375442",
"patroni": {
  "version": "3.1.0",
  "scope": "demo"
}
}

```

Example: Example: が有効になっているクラスター

```

$ curl -s http://localhost:8008/patroni | jq .
{
  "state": "running",
  "postmaster_start_time": "2023-08-18 11:09:08.615242+00:00",
  "role": "replica",
  "server_version": 150004,
  "xlog": {
    "location": 67420024
  },
  "timeline": 1,
  "replication": [
    {
      "username": "replicator",
      "application_name": "patroni2",
      "client_addr": "10.89.0.6",

```

(次のページに続く)

(前のページからの続き)

```

    "state": "streaming",
    "sync_state": "async",
    "sync_priority": 0
  },
  {
    "username": "replicator",
    "application_name": "patroni3",
    "client_addr": "10.89.0.2",
    "state": "streaming",
    "sync_state": "async",
    "sync_priority": 0
  }
],
"pause": true,
"dcs_last_seen": 1692356928,
"tags": {
  "clonefrom": true
},
"database_system_identifier": "7268616322854375442",
"patroni": {
  "version": "3.1.0",
  "scope": "demo"
}
}

```

GET /metrics エンドポイントを介して、Prometheus 形式で Patroni メトリックを取得します。

```

$ curl http://localhost:8008/metrics

# HELP patroni_version Patroni semver without periods. \
# TYPE patroni_version gauge
patroni_version{scope="batman"} 020103
# HELP patroni_postgres_running Value is 1 if Postgres is running, 0 otherwise.
# TYPE patroni_postgres_running gauge
patroni_postgres_running{scope="batman"} 1
# HELP patroni_postmaster_start_time Epoch seconds since Postgres started.
# TYPE patroni_postmaster_start_time gauge
patroni_postmaster_start_time{scope="batman"} 1657656955.179243
# HELP patroni_master Value is 1 if this node is the leader, 0 otherwise.
# TYPE patroni_master gauge
patroni_master{scope="batman"} 1
# HELP patroni_primary Value is 1 if this node is the leader, 0 otherwise.
# TYPE patroni_primary gauge
patroni_primary{scope="batman"} 1
# HELP patroni_xlog_location Current location of the Postgres transaction log, 0 if

```

(次のページに続く)

```

→this node is not the leader.
# TYPE patroni_xlog_location counter
patroni_xlog_location{scope="batman"} 22320573386952
# HELP patroni_standby_leader Value is 1 if this node is the standby_leader, 0
→otherwise.
# TYPE patroni_standby_leader gauge
patroni_standby_leader{scope="batman"} 0
# HELP patroni_replica Value is 1 if this node is a replica, 0 otherwise.
# TYPE patroni_replica gauge
patroni_replica{scope="batman"} 0
# HELP patroni_sync_standby Value is 1 if this node is a sync standby replica, 0
→otherwise.
# TYPE patroni_sync_standby gauge
patroni_sync_standby{scope="batman"} 0
# HELP patroni_xlog_received_location Current location of the received Postgres
→transaction log, 0 if this node is not a replica.
# TYPE patroni_xlog_received_location counter
patroni_xlog_received_location{scope="batman"} 0
# HELP patroni_xlog_replayed_location Current location of the replayed Postgres
→transaction log, 0 if this node is not a replica.
# TYPE patroni_xlog_replayed_location counter
patroni_xlog_replayed_location{scope="batman"} 0
# HELP patroni_xlog_replayed_timestamp Current timestamp of the replayed Postgres
→transaction log, 0 if null.
# TYPE patroni_xlog_replayed_timestamp gauge
patroni_xlog_replayed_timestamp{scope="batman"} 0
# HELP patroni_xlog_paused Value is 1 if the Postgres xlog is paused, 0 otherwise.
# TYPE patroni_xlog_paused gauge
patroni_xlog_paused{scope="batman"} 0
# HELP patroni_postgres_streaming Value is 1 if Postgres is streaming, 0 otherwise.
# TYPE patroni_postgres_streaming gauge
patroni_postgres_streaming{scope="batman"} 1
# HELP patroni_postgres_in_archive_recovery Value is 1 if Postgres is replicating
→from archive, 0 otherwise.
# TYPE patroni_postgres_in_archive_recovery gauge
patroni_postgres_in_archive_recovery{scope="batman"} 0
# HELP patroni_postgres_server_version Version of Postgres (if running), 0 otherwise.
# TYPE patroni_postgres_server_version gauge
patroni_postgres_server_version {scope="batman"} 140004
# HELP patroni_cluster_unlocked Value is 1 if the cluster is unlocked, 0 if locked.
# TYPE patroni_cluster_unlocked gauge
patroni_cluster_unlocked{scope="batman"} 0
# HELP patroni_postgres_timeline Postgres timeline of this node (if running), 0
→otherwise.
# TYPE patroni_postgres_timeline counter

```

(前のページからの続き)

```

patroni_failsafe_mode_is_active{scope="batman"} 0
# HELP patroni_postgres_timeline Postgres timeline of this node (if running), 0
# TYPE patroni_postgres_timeline counter
patroni_postgres_timeline{scope="batman"} 24
# HELP patroni_dcs_last_seen Epoch timestamp when DCS was last contacted successfully
# TYPE patroni_dcs_last_seen gauge
patroni_dcs_last_seen{scope="batman"} 1677658321
# HELP patroni_pending_restart Value is 1 if the node needs a restart, 0 otherwise.
# TYPE patroni_pending_restart gauge
patroni_pending_restart{scope="batman"} 1
# HELP patroni_is_paused Value is 1 if auto failover is disabled, 0 otherwise.
# TYPE patroni_is_paused gauge
patroni_is_paused{scope="batman"} 1

```

3.3 クラスターステータスエンドポイント

- GET /cluster エンドポイントは、現在のクラスターのトポロジと状態を説明する JSON ドキュメントを生成します。

```

$ curl -s http://localhost:8008/cluster | jq .
{
  "members": [
    {
      "name": "patroni1",
      "role": "leader",
      "state": "running",
      "api_url": "http://10.89.0.4:8008/patroni",
      "host": "10.89.0.4",
      "port": 5432,
      "timeline": 5,
      "tags": {
        "clonefrom": true
      }
    },
    {
      "name": "patroni2",
      "role": "replica",
      "state": "streaming",
      "api_url": "http://10.89.0.6:8008/patroni",
      "host": "10.89.0.6",
      "port": 5433,

```

(次のページに続く)

```

    "timeline": 5,
    "tags": {
      "clonefrom": true
    },
    "lag": 0
  }
],
"scheduled_switchover": {
  "at": "2023-09-24T10:36:00+02:00",
  "from": "patroni1",
  "to": "patroni3"
}
}

```

- GET /history エンドポイントは、クラスターのスイッチオーバー/フェイルオーバーの履歴に関するビューを提供します。形式は、pg_wal ディレクトリ内の履歴ファイルの内容と非常によく似ています。唯一の違いは、新しいタイムラインが作成された日時を示すタイムスタンプフィールドです。

```

$ curl -s http://localhost:8008/history | jq .
[
  [
    1,
    25623960,
    "no recovery target specified",
    "2019-09-23T16:57:57+02:00"
  ],
  [
    2,
    25624344,
    "no recovery target specified",
    "2019-09-24T09:22:33+02:00"
  ],
  [
    3,
    25624752,
    "no recovery target specified",
    "2019-09-24T09:26:15+02:00"
  ],
  [
    4,
    50331856,
    "no recovery target specified",
    "2019-09-24T09:35:52+02:00"
  ]
]

```

3.4 構成エンドポイント

GET /config: 動的構成の現在のバージョンを取得します。

```
$ curl -s http://localhost:8008/config | jq .
{
  "ttl": 30,
  "loop_wait": 10,
  "retry_timeout": 10,
  "maximum_lag_on_failover": 1048576,
  "postgresql": {
    "use_slots": true,
    "use_pg_rewind": true,
    "parameters": {
      "hot_standby": "on",
      "wal_level": "hot_standby",
      "max_wal_senders": 5,
      "max_replication_slots": 5,
      "max_connections": "100"
    }
  }
}
```

PATCH /config: 既存の構成を変更します。

```
$ curl -s -XPATCH -d \
    '{"loop_wait":5,"ttl":20,"postgresql":{"parameters":{"max_connections":"101"}}}' \
    http://localhost:8008/config | jq .
{
  "ttl": 20,
  "loop_wait": 5,
  "maximum_lag_on_failover": 1048576,
  "retry_timeout": 10,
  "postgresql": {
    "use_slots": true,
    "use_pg_rewind": true,
    "parameters": {
      "hot_standby": "on",
      "wal_level": "hot_standby",
      "max_wal_senders": 5,
      "max_replication_slots": 5,
      "max_connections": "101"
    }
  }
}
```

上記の RESTAPI 呼び出しは、既存の構成にパッチを適用し、新しい構成を返します。

ノードがこの構成を処理したことを確認しましょう。まず、5 秒ごとにログ行の印刷を開始する必要があります (loop_wait=5)。/"max_connections/"の変更には再起動が必要なため、/"pending_restart/"フラグを公開する必要があります。

```
$ curl -s http://localhost:8008/patroni | jq .
{
  "pending_restart": true,
  "database_system_identifier": "6287881213849985952",
  "postmaster_start_time": "2016-06-13 13:13:05.211 CEST",
  "xlog": {
    "location": 2197818976
  },
  "patroni": {
    "scope": "batman",
    "version": "1.0"
  },
  "state": "running",
  "role": "master",
  "server_version": 90503
}
```

パラメータの削除：

いくつかの設定を削除 (リセット) したい場合は、null パッチを当てるだけです：

```
$ curl -s -XPATCH -d \
  '{"postgresql":{"parameters":{"max_connections":null}}}' \
  http://localhost:8008/config | jq .
{
  "ttl": 20,
  "loop_wait": 5,
  "retry_timeout": 10,
  "maximum_lag_on_failover": 1048576,
  "postgresql": {
    "use_slots": true,
    "use_pg_rewind": true,
    "parameters": {
      "hot_standby": "on",
      "unix_socket_directories": ".",
      "wal_level": "hot_standby",
      "max_wal_senders": 5,
      "max_replication_slots": 5
    }
  }
}
```

上記の呼び出しは、動的構成から `postgresql.parameters.max_connections` を削除します。

PUT `/config` : 既存の動的構成の完全な書き換えを無条件に実行することもできます。

```
$ curl -s -XPUT -d \
    '{"maximum_lag_on_failover":1048576,"retry_timeout":10,"postgresql":{"use_
    →slots":true,"use_pg_rewind":true,"parameters":{"hot_standby":"on","wal_level":"hot_
    →standby","unix_socket_directories":".","max_wal_senders":5}},"loop_wait":3,"ttl":20}
    →' \
    http://localhost:8008/config | jq .
{
  "ttl": 20,
  "maximum_lag_on_failover": 1048576,
  "retry_timeout": 10,
  "postgresql": {
    "use_slots": true,
    "parameters": {
      "hot_standby": "on",
      "unix_socket_directories": ".",
      "wal_level": "hot_standby",
      "max_wal_senders": 5
    },
    "use_pg_rewind": true
  },
  "loop_wait": 3
}
```

3.5 スイッチオーバーとフェイルオーバーのエンドポイント

POST `/switchover` または POST `/failover`。これらのエンドポイントは互いに非常に似ています。ただし、いくつかの小さな違いがあります。

1. フェイルオーバーエンドポイントを使用すると、正常なノードがない場合に手動フェイルオーバーを実行できますが、同時にスイッチオーバーをスケジュールすることはできません。
2. スイッチオーバーエンドポイントは反対です。クラスターが正常な場合（リーダーがあります）にのみ機能し、特定の時刻にスイッチオーバーをスケジュールできます。

POST リクエストの JSON 本体では、特定の時刻にスイッチオーバーをスケジュールする場合は、少なくとも `leader` または `candidate` フィールド、およびオプションで `scheduled_at` フィールドを指定する必要があります。

例：特定のノードへのフェイルオーバーを実行します。

```
$ curl -s http://localhost:8009/failover -XPOST -d '{"candidate":"postgresql"}'
Successfully failed over to "postgresql"
```

例：特定の時刻にリーダーからクラスター内の他の正常なレプリカへのスイッチオーバーをスケジュールします。

```
$ curl -s http://localhost:8008/switchover -XPOST -d \
    '{"leader":"postgres10","scheduled_at":"2019-09-24T12:00+00"}'
Switchover scheduled
```

状況に応じて、リクエストは異なる HTTP ステータスコードとボディで終了する場合があります。スイッチオーバーまたはフェイルオーバーが正常に完了すると、ステータスコード **200** が返されます。スイッチオーバーが正常にスケジュールされた場合、Patroni は HTTP ステータスコード **200** を返します。何かがうまくいかなかった場合、エラーステータスコード（**200**、**200** または **200** のいずれか）が応答本文にいくつかの詳細とともに返されます。詳細については、`patroni/api.py:do_POST_failover()` メソッドのソースコードを確認してください。

- **DELETE /switchover** : スケジュールされたスイッチオーバーを削除します

POST /switchover および **POST failover** エンドポイントは、それぞれ `patronictl switchover` および `patronictl failover` によって使用されます。 **DELETE /switchover** は `patronictl flush <cluster-name> switchover` によって使用されます。

3.6 エンドポイントをリスタート

- **POST /restart** : **POST /restart** 呼び出しを実行することにより、特定のノードで Postgres を再起動できます。 **POST** リクエストの JSON 本文では、オプションでいくつかのリスタート条件を指定できます。
 - **restart_pending** : ブール値、**true** に設定されている場合 Patroni は、PostgreSQL 構成のいくつかの変更を適用するために、再起動が保留中の場合にのみ PostgreSQL を再起動します。
 - **role** : ノードの現在のロールが **POST** 要求からのロールと一致する場合にのみ再起動を実行します。
 - **postgres_version** : postgres の現在のバージョンが **POST** リクエストで指定されたものよりも小さい場合にのみ再起動を実行します。
 - **timeout** : PostgreSQL が接続の受け入れを開始するまでに待機する時間。 `primary_start_timeout` をオーバーライドします。
 - **schedule** : タイムゾーン付きのタイムスタンプ、将来のどこかで再起動をスケジュールします。
- **DELETE /restart** : スケジュールされた再起動を削除します

POST /restart および **DELETE /restart** エンドポイントは、それぞれ `patronictl restart` および `patronictl flush <cluster-name> restart` によって使用されます。

3.7 エンドポイントをリロード

POST /reload 呼び出しは、Patroni に構成ファイルを再読み取りして適用するように指示します。これは、SIGHUP シグナルを Patroni プロセスに送信することと同じです。再起動が必要な Postgres パラメーターの一部（POST /reload など）を変更した場合、POST /restart エンドポイントを呼び出すか、`patronictl restart` の助けを借りて、Postgres の再起動を明示的に行う必要があります。

リロードエンドポイントは `patronictl reload` によって使用されます。

3.8 エンドポイントを再初期化する

POST /reinitialize : 指定されたノードの PostgreSQL データディレクトリを再初期化します。レプリカでのみ実行が許可されます。呼び出されると、データディレクトリを削除し、`pg_basebackup` または代替の POST /reinitialize を起動します。

Patroni がグループ内にあり、障害が発生した Postgres を回復（再起動）しようとしている場合、呼び出しは失敗する可能性があります。この問題を克服するために、リクエストボディで `{"force/":true}` を指定できます。

再初期化エンドポイントは、`patronictl reinit` によって使用されます。

第4章 レプリカのイメージングとブートストラップ

Patroni では、新しいレプリカの作成をカスタマイズできます。また、新しい空のクラスターがブートストラップされているときに何が起こるかの定義もサポートしています。2つの違いは明確に定義されています。Patroni は、クラスターの DCS に `initialize` キーが存在する場合にのみレプリカを作成します。`initialize` キーがない場合-Patroni は、初期化キーロックを取得する最初のノードで排他的にブートストラップを呼び出します。

4.1 ブートストラップ

PostgreSQL は、新しいクラスターを初期化する `initdb` コマンドを提供し、Patroni はデフォルトでそれを呼び出します。特定の状況、特に既存のクラスターのコピーとして新しいクラスターを作成する場合、組み込みメソッドをカスタムアクションに置き換える必要があります。Patroni は、新しいクラスターをブートストラップするユーザー定義スクリプトの実行をサポートし、いくつかの必須引数、つまりクラスターの名前とデータディレクトリへのパスを提供します。これは、Patroni 構成の `bootstrap` セクションで構成されます。例：

```
bootstrap:
  method: <custom_bootstrap_method_name>
  <custom_bootstrap_method_name>:
    command: <path_to_custom_bootstrap_script> [param1 [, ...]]
    keep_existing_recovery_conf: False
    no_params: False
    recovery_conf:
      recovery_target_action: promote
      recovery_target_timeline: latest
      restore_command: <method_specific_restore_command>
```

各ブートストラップメソッドは、少なくとも `name` および `command` を定義する必要があります。特別な `initdb` メソッドを使用して、デフォルトの動作をトリガーできます。その場合、`method` パラメーターを完全に省略することができます。`command` は、絶対パス、または `patroni` コマンドの場所からの相対パスを使用して指定できます。構成ファイルで定義された固定パラメーターに加えて、Patroni は2つのクラスター固有のパラメーターを提供します。

<code>--scope</code>	ブートストラップされるクラスターの名前
<code>--datadir</code>	ブートストラップするクラスターインスタンスのデータディレクトリへのパス

これら 2 つの追加フラグを渡すことは、特別な `no_params` パラメーターを `True` に設定することにより無効にできます。

ブートストラップスクリプトが 0 を返した場合、Patroni はそれによって生成された PostgreSQL インスタンスを構成して起動しようとします。中間ステップのいずれかが失敗するか、スクリプトがゼロ以外の値を返した場合、Patroni はブートストラップが失敗したと想定し、自分自身の後をクリーンアップし、初期化ロックを解放して別のノードにブートストラップする機会を与えます。

`recovery_conf` ブロックがカスタムブートストラップメソッドと同じセクションで定義されている場合、Patroni は新しくブートストラップされたインスタンスを開始する前に `recovery.conf` を生成します。通常、このような `recovery.conf` には、`promote` に設定された `recovery_target_timeline` とともに、`recovery_target_*` パラメーターの少なくとも 1 つを含める必要があります。

`keep_existing_recovery_conf` が定義され、`True` に設定されている場合、Patroni は既存の `recovery.conf` ファイルが存在しても削除しません。これは、適切な `recovery.conf` を生成する `pgBackRest` などのツールを使用してバックアップからブートストラップする場合に役立ちます。

注釈: ブートストラップメソッドはチェーンされておらず、プライマリが失敗した場合にデフォルトのメソッドにフォールバックされません

4.2 レプリカの構築

Patroni は、新しいレプリカを作成するために、実証済みの `pg_basebackup` を使用します。その欠点の 1 つは、実行中のリーダーノードが必要であることです。もう 1 つは、バックアップデータの「オンザフライ」圧縮がないこと、および古いバックアップファイルのビルトインクリーンアップがないことです。WAL-E、`pgBackRest`、`Barman` などの他のバックアップソリューションを好む人、または単に独自のスクリプトをロールする人もいます。これらすべてのユースケースに対応するために、Patroni はカスタムスクリプトの実行をサポートして新しいレプリカを複製します。これらは `postgresql` 構成ブロックで構成されます。

```
postgresql:
  create_replica_methods:
    - <method name>
  <method name>:
    command: <command name>
    keep_data: True
    no_params: True
    no_leader: 1
```

例: `wal_e`

```
postgresql:
  create_replica_methods:
    - wal_e
    - basebackup
  wal_e:
```

(次のページに続く)

(前のページからの続き)

```

command: patroni_wale_restore
no_leader: 1
envdir: {{WALE_ENV_DIR}}
use_iam: 1
basebackup:
  max-rate: '100M'

```

例：pgbackrest

```

postgresql:
  create_replica_methods:
    - pgbackrest
    - basebackup
  pgbackrest:
    command: /usr/bin/pgbackrest --stanza=<scope> --delta restore
    keep_data: True
    no_params: True
  basebackup:
    max-rate: '100M'

```

`create_replica_methods` は、使用可能なレプリカ作成方法とそれらの実行順序を定義します。Patroni は、0 を返す最初のメソッドで停止します。各メソッドは、構成ファイルに個別のセクションを定義し、実行するコマンドとそのコマンドに渡す必要があるカスタムパラメーターをリストする必要があります。すべてのパラメーターは `--name=value` 形式で渡されます。ユーザー定義パラメーターに加えて、Patroni はクラスター固有のパラメーターをいくつか提供します。

<code>--scope</code>	このレプリカが属するクラスター
<code>--datadir</code>	レプリカのデータディレクトリへのパス
<code>--role</code>	常に「レプリカ」
<code>--connstring</code>	クローン元のクラスターメンバー（プライマリまたは他のレプリカ）に接続するための接続文字列。接続文字列内のユーザーは、SQL およびレプリケーションプロトコルコマンドを実行できます。

特別な `no_leader` パラメーターが定義されている場合、定義されている場合、実行中のリーダーまたはレプリカがない場合でも、Patroni はレプリカ作成メソッドを呼び出すことができます。その場合、空の文字列が接続文字列に渡されます。これは、バイナリバックアップから以前に実行されていたクラスターを復元する場合に役立ちます。

特別な `keep_data` パラメーターが定義されている場合、定義されている場合、復元を呼び出す前に PGDATA フォルダをクリーンアップしないように Patroni に指示します。

特別な `no_params` パラメーターが定義されている場合、カスタムコマンドにパラメーターを渡すことを制限します。

`basebackup` メソッドは特殊なケースです。 `create_replica_methods` が空の場合に使用されますが、 `create_replica_methods` メソッドの中で明示的にリストすることは可能です。このメソッドは

`pg_basebackup` で新しいレプリカを初期化します。 `clonefrom` タグが付いたレプリカがない限り、ベースバックアップはリーダーから取得されます。その場合、そのようなレプリカの 1 つが `pg_basebackup` のオリジンとして使用されます。構成なしで動作します。ただし、 `basebackup` 構成セクションを指定することは可能です。他のメソッド構成と同じルールが適用されます。つまり、ロング (--付き) オプションのみを指定する必要があります。接続文字列をオーバーライドするか、tar または圧縮されたベースバックアップを作成するオプションを提供する場合、パトニーはレプリカを作成できません。 `basebackup` セクションに渡されたパラメーターの名前または値に対して実行される検証はありません。また、WAL フォルダーにシンボリックリンクが使用される場合、オプションとして正しい `--waldir` パスを指定するのはユーザー次第であることにも注意してください。ただし、このオプションは v10 以降のみサポートされています。

ベースバックアップパラメーターは、マップ（キーと値のペア）または要素のリストとして指定できます。各要素は、キーと値のペアまたは単一のキー（値を受け取らないオプションの場合、たとえば、`--verbose`）。次の 2 つの例を考えてみましょう。

```
postgresql:
  basebackup:
    max-rate: '100M'
    checkpoint: 'fast'
```

そして

```
postgresql:
  basebackup:
    - verbose
    - max-rate: '100M'
    - waldir: /pg-wal-mount/external-waldir
```

すべてのレプリカ作成メソッドが失敗した場合、Patroni は次のイベントループサイクル中にすべてのメソッドを順番に再試行します。

4.3 スタンバイクラスタ

もう 1 つの利用可能なオプションは、`スタンバイクラスタ`を実行することです。これには、リモートノードからレプリケートするスタンバイノードのみが含まれます。このタイプのクラスタには次のものがあります。

- `スタンバイリーダー`。リモートノードからレプリケートすることを除いて、通常のクラスタリーダーとほとんど同じように動作します。
- スタンバイリーダーから複製しているカスケードレプリカ。

スタンバイリーダーは、DCS のリーダーロックを保持および更新します。リーダーロックが期限切れになると、カスケードレプリカは選挙を実行してスタンバイから別のリーダーを選択します。

スタンバイクラスタとレプリケート元のプライマリクラスタの間にこれ以上の関係はありません。特に、同じ DCS を使用する場合、同じ DCS スコープを共有してはなりません。彼らは、レプリケーション

情報を除いて、お互いに他に何も知りません。また、スタンバイクラスターは、プライマリクラスターの `patronictl list` または `patronictl topology` 出力に表示されません。

柔軟性のために、`standby_cluster` セクションで `create_replica_methods` キーを提供することにより、クラスターが「スタンバイモード」にあるときにレプリカを作成し、WAL レコードをリカバリする方法を指定できます。クラスターがデタッチされ、`postgresql` セクションの `create_replica_methods` によって制御される通常のクラスターとして機能する場合、レプリカの作成とは異なります。`postgresql` セクションの `standby` と `normal` の両方の `create_replica_methods` 参照キー。

このようなクラスターを構成するには、パトロン構成でセクション `standby_cluster` を指定する必要があります。

```
bootstrap:
  dcs:
    standby_cluster:
      host: 1.2.3.4
      port: 5432
      primary_slot_name: patroni
      create_replica_methods:
        - basebackup
```

これらのオプションはクラスターブートストラップ中に 1 回のみ適用され、後でそれらを変更する唯一の方法は DCS を使用することであることに注意してください。

Patroni は、リモートプライマリの PGDATA で `postgresql.conf` または `postgresql.conf.backup` を見つけることを想定しており、ベースバックアップ後に見つからない場合、起動しません。リモートプライマリがその `postgresql.conf` を他の場所に保持している場合、それを PGDATA にコピーするのはあなたの責任です。

スタンバイクラスターでレプリケーションスロットを使用する場合、プライマリクラスターにも対応するレプリケーションスロットを作成する必要があります。スタンバイクラスターの実装では自動的には行われません。プライマリクラスターで Patroni の永続的なレプリケーションスロット機能を使用して、`primary_slot_name`、または `primary_slot_name` が提供されていない場合はそのデフォルト値と同じ名前のレプリケーションスロットを維持できます。

第5章 レプリケーションモード

Patroni は PostgreSQL ストリーミングレプリケーションを使用しています。ストリーミングレプリケーションの詳細については、[Postgres documentation](#) を参照してください。デフォルトでは、Patroni は PostgreSQL を非同期レプリケーション用に構成します。レプリケーションスキーマの選択は、ビジネス上の考慮事項によって異なります。非同期レプリケーションと同期レプリケーションの両方、および他の HA ソリューションを調査して、どのソリューションが最適かを判断します。

5.1 非同期モードの持続性

非同期モードでは、可用性を確保するために、クラスターはコミットされたトランザクションの一部を失うことが許可されます。プライマリサーバーに障害が発生したり、他の理由で使用できなくなった場合、Patroni は自動的に十分に健全なスタンバイをプライマリに昇格させます。そのスタンバイにレプリケートされていないトランザクションは、プライマリの「フォークされたタイムライン」に残り、実質的に回復不能です [1]。

失われる可能性のあるトランザクションの量は、`maximum_lag_on_failover` パラメーターで制御されます。プライマリトランザクションログの位置はリアルタイムでサンプリングされないため、実際にはフェイルオーバーで失われるデータの量は、トランザクションログの `maximum_lag_on_failover` バイトと最後の `tli` 秒 (`loop_wait` / 2 秒に書き込まれた量によって制限される最悪の場合です平均的な場合)。ただし、一般的な定常状態のレプリケーション遅延は 1 秒未満です。

デフォルトでは、リーダー選挙を実行するときに、Patroni はレプリカの現在のタイムラインを考慮しません。これは、場合によっては望ましくない動作になる可能性があります。`check_timeline` パラメーターの値を `true` 変更することにより、元のプライマリが新しいリーダーになるのと同じタイムラインを持たないノードを防ぐことができます。

5.2 PostgreSQL 同期レプリケーション

Patroni で Postgres の `synchronous replication` を使用できます。同期レプリケーションは、接続しているクライアントに成功して戻る前に、書き込みがセカンダリに書き込まれることを確認することにより、クラスター全体の一貫性を保証します。同期レプリケーションのコスト: 書き込みのスループットが低下します。このスループットは、完全にネットワークパフォーマンスに基づいています。

ホストされたデータセンター環境 (AWS、Rackspace、または制御できないネットワークなど) では、同期レプリケーションにより書き込みパフォーマンスの変動が大幅に増加します。フォロワーがリーダーからアクセスできなくなると、リーダーは効果的に読み取り専用になります。

単純な同期レプリケーションテストを有効にするには、YAML 構成ファイルの `parameters` セクションに次の行を追加します。

```
synchronous_commit: "on"
synchronous_standby_names: "*"

```

PostgreSQL 同期レプリケーションを使用する場合は、少なくとも 3 つの Postgres データノードを使用して、1 つのホストに障害が発生した場合の書き込み可用性を保証します。

PostgreSQL 同期レプリケーションを使用しても、あらゆる状況下でトランザクションの損失がゼロになることは保証されません。プライマリと、現在同期レプリカとして機能しているセカンダリに同時に障害が発生すると、すべてのトランザクションが含まれていない可能性のある 3 番目のノードが昇格されます。

5.3 同期モード

コミットされたトランザクションを失うことが許可されないユースケースでは、Patroni の `synchronous_mode` をオンにすることができます。 `synchronous_mode` がオンになると、Patroni はスタンバイに成功したコミットステータスをクライアントに返した可能性のあるすべてのトランザクションが含まれていることが確実でない限り、スタンバイをプロモートしません [2]。これは、一部のサーバーが利用可能であっても、システムが書き込みに利用できない可能性があることを意味します。システム管理者は、トランザクションが失われた場合でも、手動フェールオーバーコマンドを使用してスタンバイを昇格できます。

`synchronous_mode` をオンにしても、あらゆる状況下でのコミットのマルチノード耐久性が保証されるわけではありません。適切なスタンバイが利用できない場合、プライマリサーバーは引き続き書き込みを受け入れますが、レプリケーションは保証しません。このモードでプライマリに障害が発生すると、スタンバイはプロモートされません。システム管理者が手動フェールオーバーを実行しない限り、以前プライマリだったホストが戻ってくると、自動的に昇格します。この動作により、2 ノードクラスターで同期モードを使用できるようになります。

`synchronous_mode` がオンで、スタンバイがクラッシュすると、パトローニの次の反復が実行され、プライマリがスタンドアロンモードに切り替えるまでコミットはブロックされます（書き込みの最悪の場合の遅延 `t1` 秒、平均の場合 `loop_wait/2` 秒）。スタンバイを手動でシャットダウンまたは再起動しても、コミットサービスは中断されません。スタンバイは、PostgreSQL のシャットダウンが開始される前に、プライマリに同期スタンバイ義務から自分自身を解放するように通知します。

各書き込みが少なくとも 2 つのノードに永続的に保存されることを保証する必要がある場合は、`synchronous_mode` に加えて `synchronous_mode_strict` を有効にします。このパラメーターは、利用可能な同期スタンバイ候補がない場合に Patroni がプライマリで同期レプリケーションをオフに切り替えないようにします。マイナス面として、プライマリは書き込みに利用できず（Postgres トランザクションが明示的に `synchronous_mode` をオフにしない限り）、少なくとも 1 つの同期レプリカが起動するまですべてのクライアントの書き込み要求をブロックします。

`nosync` タグを `true` に設定することで、スタンバイが同期スタンバイにならないようにすることができます。これは、低速のネットワーク接続の背後にあり、同期スタンバイになったときにパフォーマンスの低下の原因となるスタンバイに設定することをお勧めします。

同期モードは、PatroniREST インターフェイスを介してオン/オフを切り替えることができます。手順については、[dynamic configuration](#) を参照してください。

注：PostgreSQL での同期レプリケーションの実装方法により、`synchronous_mode_strict` を使用していてもトランザクションを失う可能性があります。レプリケーションの確認を待っている間に PostgreSQL

バックエンドがキャンセルされた場合（クライアントのタイムアウトまたはバックエンドの障害によるパケットキャンセルの結果）、トランザクションの変更は他のバックエンドで可視になります。このような変更はまだレプリケートされておらず、スタンバイプロモーションの場合に失われる可能性があります。

5.4 同期レプリケーション係数

パラメーター `synchronous_node_count` は、Patroni によって使用され、同期スタンバイデータベースの数を管理します。デフォルトでは 1 に設定されています。 `synchronous_mode` が off に設定されている場合、効果はありません。有効にすると、Patroni はパラメーター `synchronous_node_count` に基づいて正確な数の同期スタンバイデータベースを管理し、メンバーが参加および離脱するときに DCS および `synchronous_standby_names` の状態を調整します。

5.5 同期モードの実装

同期モードの場合、Patroni は DCS で同期状態を維持し、最新のプライマリおよび現在の同期スタンバイデータベースを含みます。この状態は、次の不変条件を保証するために、厳密な順序付け制約で更新されます。

- ノードは、書き込みトランザクションを受け入れることができる場合は常に、最新のリーダーとしてマークする必要があります。Patroni がクラッシュしたり、PostgreSQL がシャットダウンしないと、この不変式に違反が発生する可能性があります。
- ノードは、同期スタンバイとして公開されている限り、PostgreSQL で同期スタンバイとして設定する必要があります。
- リーダーまたは現在の同期スタンバイではないノードは、自分自身を自動的に昇格することはできません。

Patroni は、 `synchronous_node_count` パラメーターに基づいて 1 つ以上の同期スタンバイノードのみを `synchronous_standby_names` に割り当てます。

HA ループの反復ごとに、Patroni は同期スタンバイノードの選択を再評価します。同期スタンバイノードの現在のリストが接続されており、その同期ステータスの削除を要求していない場合、選択されたままになります。それ以外の場合、レプリケーションで最も先にある同期に使用可能なクラスターメンバーが選択されます。

第6章 ウォッチドッグのサポート

複数の PostgreSQL サーバーをプライマリとして実行すると、タイムラインがずれるため、トランザクションが失われる可能性があります。この状況は、スプリットブレイン問題とも呼ばれます。スプリットブレインを避けるために、Patroni は、DCS でリーダーキーが期限切れになった後、PostgreSQL がトランザクションコミットを受け入れないことを保証する必要があります。通常の場合では、Patroni は、リーダーロックの更新が何らかの理由で失敗したときに PostgreSQL を停止することにより、これを達成しようとします。ただし、これはさまざまな理由により失敗する場合があります。

- Patroni は、バグ、メモリ不足、またはシステム管理者によって誤って強制終了されたためにクラッシュしました。
- PostgreSQL のシャットダウンは遅すぎます。
- システムの負荷が高い、ハイパーバイザーによって VM が一時停止されている、またはその他のインフラストラクチャの問題が原因で、Patroni を実行できません。

これらの条件下での正しい動作を保証するために、Patroni はウォッチドッグデバイスをサポートしています。ウォッチドッグデバイスは、指定された時間枠内にキープアライブハートビートを取得できない場合にシステム全体をリセットするソフトウェアまたはハードウェアメカニズムです。これにより、通常の Patroni スプリットブレイン保護メカニズムが失敗した場合のフェイルセーフの追加レイヤーが追加されます。

Patroni は、PostgreSQL をプライマリに昇格する前にウォッチドッグをアクティブ化しようとします。ウォッチドッグのアクティブ化が失敗し、ウォッチドッグモードが **required** の場合、ノードはリーダーになることを拒否します。リーダー選挙への参加を決定するとき、Patroni は、ウォッチドッグ構成によりリーダーになることを許可することも確認します。PostgreSQL を降格した後（たとえば、手動フェイルオーバーのため）、Patroni はウォッチドッグを再度無効にします。Patroni が一時停止状態の間、ウォッチドッグも無効になります。

デフォルトでは、Patroni は TTL が期限切れになる 5 秒前にウォッチドッグが期限切れになるように設定します。loop_wait=10 および ttl=30 のデフォルト設定では、これにより、システムが強制的にリセットされる前に HA ループが完了するまでに少なくとも 15 秒（ttl - safety_margin - loop_wait）が与えられます。デフォルトでは、DCS へのアクセスは 10 秒後にタイムアウトするように構成されています。これは、たとえばネットワークの問題のために DCS が利用できない場合、Patroni と PostgreSQL は少なくとも 5 秒（ttl - safety_margin - loop_wait - retry_timeout）、すべてのクライアント接続が終了する状態になることを意味します。

セーフティマージンは、Patroni がリーダーキーの更新とウォッチドッグキープアライブの間の時間のために予約する時間です。Patroni は、リーダーキーの更新の確認後すぐにキープアライブを送信しようとします。Patroni プロセスがまさに適切なタイミングで長時間中断されると、ウォッチドッグをトリガーせずにキープアライブが安全マージンを超えて遅延する場合があります。これにより、リーダーキーの有効期限が切れる前にウォッチドッグがトリガーされず、保証が無効になる時間帯が発生します。ウォッチドッグがすべての状況下でトリガーされることを絶対に保証するには、safety_margin を -1 に設定してウォッチドッグタイムアウトを ttl // 2 に設定することにより、TTL の半分後にウォッチドッグが期限切れになるよう

に設定します。この保証が必要な場合は、おそらく `ttl` を増やしたり、`loop_wait` および `retry_timeout` を減らしたりする必要があります。

現在、ウォッチドッグは Linux ウォッチドッグデバイスインターフェイスを使用した場合にのみサポートされています。

6.1 Linux でソフトウェアウォッチドッグを設定する

デフォルトの Patroni 構成は、Patroni がアクセス可能な場合、Linux で `/dev/watchdog` を使用しようとしています。ほとんどのユースケースでは、Linux カーネルに組み込まれたソフトウェアウォッチドッグを使用することで十分に安全です。

ソフトウェアウォッチドッグを有効にするには、Patroni を起動する前に `root` として次のコマンドを発行します。

```
modprobe softdog
# Replace postgres with the user you will be running patroni under
chown postgres /dev/watchdog
```

テストには、`modprobe` コマンドラインに `soft_noboot=1` を追加して、再起動を無効にすると役立つ場合があります。この場合、ウォッチドッグはカーネルリングバッファに ``dmesg`` を介して表示される行をログに記録するだけです。

Patroni は、ウォッチドッグが正常に有効にされると、ウォッチドッグに関する情報をログに記録します。

第7章 クラスターの一時的停止/再開モード

7.1 目標

特定の状況下では、Patroni は DCS でクラスターの状態を維持しながら、クラスターの管理を一時的に辞任する必要があります。考えられるユースケースは、メジャーバージョンのアップグレードや破損の回復など、クラスターでの一般的ではないアクティビティです。これらのアクティビティ中に、パトローニには不明な理由でノードが起動および停止されることがよくあります。一部のノードは一時的に昇格することさえあり、1つのプライマリのみを実行するという前提に違反します。したがって、Patroni は実行中のクラスターから「デタッチ」でき、Pacemaker のメンテナンスモードに相当するものを実装する必要があります。

7.2 実装

Patroni が一時停止モードで実行されている場合、次の場合を除き、PostgreSQL の状態は変更されません。

- 各ノードについて、DCS のメンバーキーは、クラスターに関する現在の情報で更新されます。これにより、メンバーが実行されている場合、Patroni はメンバーノードで読み取り専用クエリを実行します。
- リーダーロックを持つ Postgres プライマリの場合、Patroni はロックを更新します。リーダーロックを持つノードがプライマリであることを停止した場合（つまり、手動で降格された場合）、Patroni はノードを昇格する代わりにロックを解放します。
- 手動によるスケジュールされていない再起動、再初期化、および手動フェールオーバーが許可されます。手動フェールオーバーは、フェールオーバー先のノードが指定されている場合にのみ許可されます。一時停止モードでは、手動フェールオーバーに実行中のプライマリノードは必要ありません。
- 「パラレル」プライマリが Patroni によって検出された場合、警告を発しますが、リーダーロックなしでプライマリを降格しません。
- クラスター内にリーダーロックがない場合、実行中のプライマリがロックを取得します。複数のプライマリノードがある場合、最初にロックを取得したプライマリが勝ちます。プライマリがまったくない場合、Patroni はレプリカを昇格しようとしません。このルールには例外があります。古いプライマリが手動昇格のために自分自身を降格したためにリーダーロックがない場合、昇格要求で言及された候補ノードのみがリーダーロックを取得できます。新しいリーダーロックが付与されると（つまり、レプリカを手動でプロモートした後）、Patroni は、以前のリーダーからストリーミングしていたレプリカが新しいリーダーに切り替わることを確認します。
- Postgres が停止すると、Patroni は起動しようとしません。Patroni が停止すると、管理している Postgres インスタンスを停止しようとしません。

- Patroni は、他のクラスターメンバーを表していないか、永続スロットの構成にリストされていないレプリケーションスロットを削除しようとしません。

7.3 ユーザーガイド

patronictl は、`pause` および `resume` コマンドをサポートします。

`{"pause/": true/false/null}` を使用して `{namespace}/{cluster}/config` キーに PATCH リクエストを発行することもできます

第8章 DCS フェイルセーフモード

8.1 問題

Patroni は、リーダー選挙のタスクを解決し、ネットワークパーティショニングを検出するために、分散構成ストア（DCS）に大きく依存しています。つまり、ノードは、DCS のリーダーロックを更新できる場合にのみ、プライマリとして Postgres を実行できます。リーダーロックの更新が失敗した場合、Postgres はすぐに降格され、読み取り専用として起動されます。どの DCS を使用するかにによって、「問題」が発生する可能性は異なります。たとえば、Patroni にのみ使用される Etcd を使用すると、チャンスはゼロに近くなりますが、K8sAPI（Etcd が支援）を使用すると、より頻繁に観察される可能性があります。

8.2 今回の実施の理由

リーダーロックの更新の失敗は、主に 2 つの理由が考えられます。

1. ネットワークパーティショニング
2. DCS がダウンしています

一般に、これら 2 つを単一のノードから区別することは不可能であるため、Patroni は最悪のケースであるネットワークパーティショニングを想定しています。パーティション分割されたネットワークの場合、Patroni クラスターの他のノードがリーダーロックを正常に取得し、Postgres をプライマリに昇格させる可能性があります。スプリットブレインを回避するために、古いプライマリはリーダーロックが期限切れになる前に降格されます。

8.3 DCS フェイルセーフモード

新しい特別なオプション `failsafe_mode` を導入します。DCS `/config` キーに保存されているグローバル `failsafe_mode` を介してのみ有効にできます。フェイルセーフモードが有効になっており、DCS でのリーダーロックの更新がバージョン/値/インデックスの不一致とは別の理由で失敗した場合、Postgres は、PatroniRESTAPI を介してクラスターのすべての既知のメンバーにアクセスできる場合、プライマリとして実行を継続できます。

8.4 低レベル実装の詳細

- `/failsafe` という名前の DCS に新しい永久キーを導入します。
- `/failsafe` キーには、特定の時点での特定の Patroni クラスターの既知のメンバーがすべて含まれています。
- 現在のリーダーは `/failsafe` キーを維持します。
- メンバーは、`/failsafe` キーに存在する場合にのみ、リーダーレースに参加して新しいリーダーになることができます。
- クラスターが単一のノードで構成されている場合、`/failsafe` キーには単一のメンバーが含まれます。
- DCS/"停止"/の場合、既存のプライマリは `POST /failsafe RESTAPI` を介して `/failsafe` キーで提示されるすべてのメンバーに接続し、すべてのレプリカが確認した場合、プライマリとして実行を継続できます。
- メンバーの 1 人が応答しない場合、プライマリは降格されます。
- レプリカは、プライマリがまだ生きていることを示すインジケーターとして、着信 `POST /failsafe RESTAPI` 要求を使用しています。この情報は `ttl` 秒間キャッシュされます。

8.5 よくある質問

- なぜ現在のプライマリは他のすべてのメンバーを見なければならないのですか?ここでクォーラムに頼ることはできませんか?

これは素晴らしい質問です!問題は、クォーラムの見方が DCS と Patroni の観点と異なる可能性があることです。DCS ノードはアベイラビリティゾーン全体に均等に分散する必要がありますが、Patroni にはそのようなルールがなく、さらに重要なことに、そのようなルールを導入および適用するメカニズムがありません。マイノリティノードが勝利部分にある間に、Patroni ノードの大多数がパーティションネットワークの敗北部分(プライマリを含む)に終わった場合、プライマリを降格する必要があります。他のすべてのメンバーをチェックするだけで、そのような状況を検出できます。

- DCS がダウンしているときにノード/ポッドが終了するとどうなりますか?

DCS にアクセスできない場合、「他のすべてのクラスターメンバーにアクセスできますか?」チェックハートビートループの周期ごと (`loop_wait` 秒ごと) に実行されます。ポッド/ノードが終了すると、チェックは失敗し、Postgres は読み取り専用で降格され、DCS が復元されるまで回復しません。

- DCS がダウンしているときに Patroni クラスターのすべてのメンバーが失われた場合はどうなりますか?

Patroni は、クラスターにリーダーがない場合でも、バックアップから新しいレプリカを作成するように構成できます。ただし、新しいメンバーが `/failsafe` キーに存在しない場合、リーダーロックを取得してプロモートすることはできません。

- プライマリが DCS へのアクセスを失ったのに対し、レプリカはアクセスできなかった場合はどうなりますか?

プライマリはフェイルセーフコードを実行し、既知のすべてのレプリカに連絡します。これらのレプリカはこの情報を、プライマリが生きていることを示すインジケーターとして使用し、DCS のリーダーロックが期限切れになってもリーダーレースを開始しません。

- フェイルセーフモードを有効にするには?

`failsafe_mode` を有効にする前に、すべてのメンバーの Patroni バージョンが最新であることを確認してください。その後、`PATCH /config failsafe_mode` または `patronictl edit-config -s failsafe_mode=true` のいずれかを使用できます

第9章 KubernetesでPatroniを使用する

Patroni は、Kubernetes オブジェクトを使用して、クラスターの状態を保存し、リーダーキーを管理できます。これにより、コンシステンシーストアなしで Kubernetes 環境で Postgres を操作できるようになります。つまり、追加の Etcd デプロイメントを実行する必要はありません。Patroni がリーダーと構成キーを保存するために使用できる Kubernetes オブジェクトには 2 つの異なるタイプがあります。それらは ``kubernetes.use_endpoints`` または ``PATRONI_KUBERNETES_USE_ENDPOINTS`` 環境変数で構成されます。

9.1 エンドポイントを使用する

これは推奨モードであるにもかかわらず、互換性上の理由からデフォルトでオフになっています。オンの場合、Patroni はクラスター構成とリーダーキーを、作成するそれぞれの ``Endpoints`` の ``metadata: annotations`` フィールドに保存します。リーダー情報を含むアノテーションと、実行中のリーダーポッドを指す実際のアドレスの両方が一度に更新されるため、リーダーの変更は ``ConfigMaps`` を使用する場合よりも安全です。

9.2 ConfigMapを使用する

このモードでは、Patroni は Endpoints の代わりに ConfigMaps を作成し、これらの ConfigMaps のメタデータ内にキーを保存します。リーダーを変更するには、リーダーの ConfigMap とそれぞれのエンドポイントに対する少なくとも 2 つの更新が必要です。

トラフィックを Postgres リーダーに転送するには、``role_label``（パトニ構成で構成）でラベルセクターを使用するように KubernetesPostgres サービスを構成する必要があります。

場合によっては、たとえば、OpenShift で実行している場合、ConfigMap を使用する以外に方法がないことに注意してください。

9.3 構成

PatroniKubernetes *settings* および *settings* は、ドキュメントの一般的な章で説明されています。

9.3.1 ロールラベルをカスタマイズする

デフォルトでは、Patroni は、`role=master` などのノードのロールに基づいて、実行されるポッドに対応するラベルを設定します。ラベルのキーと値は、`kubernetes.role_label`、`kubernetes.leader_label_value`、`kubernetes.follower_label_value` および `kubernetes.standby_leader_label_value` でカスタマイズできます。

デフォルトのロールラベルからカスタムのものに移行する場合、次の移行手順によりダウンタイムを削減できることに注意してください。

1. `kubernetes.tmp_role_label` (``tmp_role`` など) でポッドの元のロール値を使用して、一時ラベルを追加します。ポッドが再起動されると、Patroni によって設定された次のラベルが取得されます。

```
labels:
  cluster-name: foo
  role: master
  tmp_role: master
```

2. すべてのポッドが更新されたら、サービスセクターを変更して一時ラベルを選択します。

```
selector:
  cluster-name: foo
  tmp_role: master
```

3. カスタムロールラベルを追加します (例、`set kubernetes.leader_label_value=primary`)。ポッドが再起動されると、パトローニが設定した次の新しいラベルが取得されます。

```
labels:
  cluster-name: foo
  role: primary
  tmp_role: master
```

4. すべてのポッドが再度更新されたら、新しいロール値を使用するようにサービスセクターを変更します。

```
selector:
  cluster-name: foo
  role: primary
```

5. 最後に、構成から一時ラベルを削除し、すべてのポッドを更新します。

```
labels:
  cluster-name: foo
  role: primary
```

9.4 例

- Patroni リポジトリの `kubernetes_` フォルダーには、Docker イメージの例と、PatroniKubernetes セットアップをテストするための Kubernetes マニフェストが含まれています。現在の状態では、権限の問題のため、PersistentVolumes を使用できないことに注意してください。
- [Spilo Project](#) で永続ボリュームを使用できるフル機能の Docker イメージを見つけることができます。
- Kubernetes を使用して実行されている Patroni で構成された Spilo イメージをデプロイするための [Helm chart](#) もあります。
- Patroni と Spilo を使用してデータベースクラスターを大規模に実行するには、[postgres-operator](#) プロジェクトをご覧ください。Spilo クラスターを管理するオペレーターパターンを実装します。

第10章 Citusのサポート

Patroni を使用すると、`Multi-NodeCitus`__クラスターのデプロイが非常に簡単になります。

10.1 TL;DR

従う必要がある簡単なルールはいくつかあります。

1. `Citus PostgreSQL` のデータベース拡張は、すべてのノードで利用できる必要があります。サポートされる絶対最小の Citus バージョンは 10.0 ですが、透過的なスイッチオーバーとワーカーの再起動からすべての利点を得るには、少なくとも Citus11.2 を使用することをお勧めします。
2. クラスター名（`scope`）は、すべての Citus ノードで同じである必要があります。
3. スーパーユーザーの資格情報は、コーディネーターとすべてのワーカーノードで同じである必要があります、`pg_hba.conf` はすべてのノード間のスーパーユーザーアクセスを許可する必要があります。
4. ワーカーノードからコーディネーターへの [REST API](#) アクセスを許可する必要があります。たとえば、資格情報は同じである必要があります、構成されている場合、ワーカーノードからのクライアント証明書はコーディネーターによって受け入れられる必要があります。
5. 次のセクションを `patroni.yaml` に追加します。

```
citus:
group: X # 0 for coordinator and 1, 2, 3, etc for workers
database: citus # must be the same on all nodes
```

その後、Patroni を起動するだけで、残りは処理されます。

1. `citus` 拡張機能は、`shared_preload_libraries` に自動的に追加されます。
2. `max_prepared_transactions` がグローバル `max_prepared_transactions` で明示的に設定されていない場合、Patroni は自動的に `2*max_connections` に設定します。
3. `citus.database` が自動的に作成され、その後に `CREATE EXTENSION citus` が作成されます。
4. 現在のスーパーユーザー `credentials` が `pg_dist_authinfo` テーブルに追加され、ノード間通信が許可されます。後でスーパーユーザーのユーザー名/パスワード/sslcert/sslkey を変更する場合は、忘れずに更新してください。
5. コーディネータープライマリノードは、ワーカープライマリノードを自動的に検出し、`citus_add_node()` ファンクションを使用して `pg_dist_node` テーブルに追加します。
6. Patroni は、コーディネーターまたはワーカークラスターでフェールオーバー/スイッチオーバーが発生した場合に備えて、`pg_dist_node` も維持します。

10.2 パトロン

コーディネータークラスターとワーカークラスターは、物理的に異なる PostgreSQL/Patroni クラスターであり、PostgreSQL の `Citus` データベース拡張機能を使用して論理的にグループ化されます。したがって、ほとんどの場合、それらを単一のエンティティとして管理することは不可能です。

`patroni.yaml` に通常と比較して `citus` セクションがある場合、`patronictl` の動作に 2 つの大きな違いがあります。

1. `list` および `topology` は、デフォルトで Citus フォーメーションのすべてのメンバー（コーディネーターとワーカー）を出力します。新しい列 `Group` は、それらが属している Citus グループを示します。
2. すべての `patronictl` コマンドには、`--group` という名前の新しいオプションが導入されています。一部のコマンドでは、グループのデフォルト値が `patroni.yaml` から取得される場合があります。たとえば、`patronictl pause` は、`citus` セクションで設定されている `group` のメンテナンスモードをデフォルトで有効にしますが、たとえば `patronictl switchover` または `patronictl remove` の場合、グループを明示的に指定する必要があります。

Citus クラスターの `patronictl list` 出力の例:

```
postgres@coord1:~$ patronictl list demo
+ Citus cluster: demo -----+-----+-----+-----+-----+
| Group | Member | Host          | Role          | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| 0     | coord1 | 172.27.0.10   | Replica       | running | 1  | 0         |
| 0     | coord2 | 172.27.0.6    | Sync Standby  | running | 1  | 0         |
| 0     | coord3 | 172.27.0.4    | Leader        | running | 1  |           |
| 1     | work1-1 | 172.27.0.8    | Sync Standby  | running | 1  | 0         |
| 1     | work1-2 | 172.27.0.2    | Leader        | running | 1  |           |
| 2     | work2-1 | 172.27.0.5    | Sync Standby  | running | 1  | 0         |
| 2     | work2-2 | 172.27.0.7    | Leader        | running | 1  |           |
+-----+-----+-----+-----+-----+-----+
```

`--group` オプションを追加すると、出力は次のようになります:

```
postgres@coord1:~$ patronictl list demo --group 0
+ Citus cluster: demo (group: 0, 7179854923829112860) -----+
| Member | Host          | Role          | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| coord1 | 172.27.0.10   | Replica       | running | 1  | 0         |
| coord2 | 172.27.0.6    | Sync Standby  | running | 1  | 0         |
| coord3 | 172.27.0.4    | Leader        | running | 1  |           |
+-----+-----+-----+-----+-----+-----+

postgres@coord1:~$ patronictl list demo --group 1
+ Citus cluster: demo (group: 1, 7179854923881963547) -----+
| Member | Host          | Role          | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
```

(次のページに続く)

(前のページからの続き)

work1-1	172.27.0.8	Sync Standby	running	1	0
work1-2	172.27.0.2	Leader	running	1	
+-----+-----+-----+-----+-----+					

10.3 Citus ワーカーの切り替え

Citus ワーカーノードのスイッチオーバーが調整されると、Citus は、スイッチオーバーをアプリケーションに対して透過的に近い機会を提供します。アプリケーションはコーディネーターに接続し、コーディネーターはワーカーノードに接続するため、Citus を使用して、ワーカーノードでホストされているシャードのコーディネーターで SQL トラフィックを「一時停止」できます。スイッチオーバーは、トラフィックがコーディネーターに保持されている間に発生し、新しいプライマリワーカーノードが読み取り/書き込みクエリを受け入れる準備が整うとすぐに再開されます。

ワーカークラスターでの `patronictl switchover` の例:

```
postgres@coord1:~$ patronictl switchover demo
+ Citus cluster: demo -----+-----+-----+-----+-----+
| Group | Member | Host       | Role       | State | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| 0 | coord1 | 172.27.0.10 | Replica    | running | 1 | 0 |
| 0 | coord2 | 172.27.0.6  | Sync Standby | running | 1 | 0 |
| 0 | coord3 | 172.27.0.4  | Leader     | running | 1 |  |
| 1 | work1-1 | 172.27.0.8  | Leader     | running | 1 |  |
| 1 | work1-2 | 172.27.0.2  | Sync Standby | running | 1 | 0 |
| 2 | work2-1 | 172.27.0.5  | Sync Standby | running | 1 | 0 |
| 2 | work2-2 | 172.27.0.7  | Leader     | running | 1 |  |
+-----+-----+-----+-----+-----+-----+
Citus group: 2
Primary [work2-2]:
Candidate ['work2-1'] []:
When should the switchover take place (e.g. 2022-12-22T08:02 ) [now]:
Current cluster topology
+ Citus cluster: demo (group: 2, 7179854924063375386) -----+
| Member | Host       | Role       | State | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| work2-1 | 172.27.0.5 | Sync Standby | running | 1 | 0 |
| work2-2 | 172.27.0.7 | Leader     | running | 1 |  |
+-----+-----+-----+-----+-----+-----+
Are you sure you want to switchover cluster demo, demoting current primary work2-2?
→[y/N]: y
2022-12-22 07:02:40.33003 Successfully switched over to "work2-1"
+ Citus cluster: demo (group: 2, 7179854924063375386) -----+
| Member | Host       | Role       | State | TL | Lag in MB |
```

(次のページに続く)

(前のページからの続き)

```

+-----+-----+-----+-----+-----+
| work2-1 | 172.27.0.5 | Leader | running | 1 |      |
| work2-2 | 172.27.0.7 | Replica | stopped |   | unknown |
+-----+-----+-----+-----+-----+

postgres@coord1:~$ patronictl list demo
+ Citus cluster: demo -----+-----+-----+-----+-----+
| Group | Member | Host          | Role          | State | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
|      0 | coord1 | 172.27.0.10   | Replica       | running | 1 |      0 |
|      0 | coord2 | 172.27.0.6    | Sync Standby | running | 1 |      0 |
|      0 | coord3 | 172.27.0.4    | Leader        | running | 1 |      |
|      1 | work1-1 | 172.27.0.8    | Leader        | running | 1 |      |
|      1 | work1-2 | 172.27.0.2    | Sync Standby | running | 1 |      0 |
|      2 | work2-1 | 172.27.0.5    | Leader        | running | 2 |      |
|      2 | work2-2 | 172.27.0.7    | Sync Standby | running | 2 |      0 |
+-----+-----+-----+-----+-----+-----+

```

そして、これはコーディネーター側でどのように見えるかです:

```

# The worker primary notifies the coordinator that it is going to execute "pg_ctl stop
→".
2022-12-22 07:02:38,636 DEBUG: query("BEGIN")
2022-12-22 07:02:38,636 DEBUG: query("SELECT pg_catalog.citus_update_node(3, '172.27.
→0.7-demoted', 5432, true, 10000)")
# From this moment all application traffic on the coordinator to the worker group 2
→is paused.

# The future worker primary notifies the coordinator that it acquired the leader lock
→in DCS and about to run "pg_ctl promote".
2022-12-22 07:02:40,085 DEBUG: query("SELECT pg_catalog.citus_update_node(3, '172.27.
→0.5', 5432)")

# The new worker primary just finished promote and notifies coordinator that it is
→ready to accept read-write traffic.
2022-12-22 07:02:41,485 DEBUG: query("COMMIT")
# From this moment the application traffic on the coordinator to the worker group 2
→is unblocked.

```

10.4 DCS をのぞく

Citus クラスター（コーディネーターとワーカー）は、論理的にグループ化された Patroni クラスターのフリートとして DCS に保存されます。

```
/service/batman/           # scope=batman
/service/batman/0/         # citus.group=0, coordinator
/service/batman/0/initialize
/service/batman/0/leader
/service/batman/0/members/
/service/batman/0/members/m1
/service/batman/0/members/m2
/service/batman/1/         # citus.group=1, worker
/service/batman/1/initialize
/service/batman/1/leader
/service/batman/1/members/
/service/batman/1/members/m3
/service/batman/1/members/m4
...
```

このようなアプローチが選択されたのは、ほとんどの DCS で、1 回の再帰的読み取り要求で Citus クラスター全体をフェッチできるようになるためです。ワーカーノードを検出する必要があるため、Citus コーディネーターノードのみがツリー全体を読み取っています。ワーカーノードは自分のグループのサブツリーのみを読み取っており、場合によってはコーディネーターグループのサブツリーを読み取ることができます。

10.5 CitusonKubernetes

Kubernetes は階層構造をサポートしていないため、Patroni が作成するすべての K8s オブジェクトに citus グループを含める必要がありました:

```
batman-0-leader # the leader config map for the coordinator
batman-0-config # the config map holding initialize, config, and history "keys"
...
batman-1-leader # the leader config map for worker group 1
batman-1-config
...
```

つまり、命名パターンは次のとおりです。\${scope}-\${citus.group}-\${type}。

すべての Kubernetes オブジェクトは、`labelselector`_`` を使用して Patroni によって検出されます。したがって、Patroni & Citus および Endpoints/ConfigMap を使用するすべてのポッドには、同様のラベルが必要であり、Patroni は、`Kubernetes :ref: `settings` <kubernetes_settings>` または settings` を使用してそれらを使用するように構成する必要があります。`

Pods 環境変数を使用した Patroni 構成のいくつかの例：

1. コーディネータークラスター用

```

apiVersion: v1
kind: Pod
metadata:
  labels:
    application: patroni
    citus-group: "0"
    citus-type: coordinator
    cluster-name: citusdemo
  name: citusdemo-0-0
  namespace: default
spec:
  containers:
  - env:
    - name: PATRONI_SCOPE
      value: citusdemo
    - name: PATRONI_NAME
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: metadata.name
    - name: PATRONI_KUBERNETES_POD_IP
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: status.podIP
    - name: PATRONI_KUBERNETES_NAMESPACE
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: metadata.namespace
    - name: PATRONI_KUBERNETES_LABELS
      value: '{application: patroni}'
    - name: PATRONI_CITUS_DATABASE
      value: citus
    - name: PATRONI_CITUS_GROUP
      value: "0"

```

2. グループ 2 のワーカークラスターの場合

```

apiVersion: v1
kind: Pod
metadata:
  labels:
    application: patroni
    citus-group: "2"

```

(次のページに続く)

(前のページからの続き)

```

    citus-type: worker
    cluster-name: citusdemo
name: citusdemo-2-0
namespace: default
spec:
  containers:
  - env:
    - name: PATRONI_SCOPE
      value: citusdemo
    - name: PATRONI_NAME
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: metadata.name
    - name: PATRONI_KUBERNETES_POD_IP
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: status.podIP
    - name: PATRONI_KUBERNETES_NAMESPACE
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: metadata.namespace
    - name: PATRONI_KUBERNETES_LABELS
      value: '{application: patroni}'
    - name: PATRONI_CITUS_DATABASE
      value: citus
    - name: PATRONI_CITUS_GROUP
      value: "2"

```

お気づきかもしれませんが、両方の例には `citus-group` ラベルセットがあります。このラベルにより、Patroni はオブジェクトを特定の Citus グループに属するものとして識別できます。それに加えて、`citus-group` ラベルと同じ値を持つ `PATRONI_CITUS_GROUP` 環境変数もあります。Patroni が新しい Kubernetes オブジェクト ConfigMap または Endpoints を作成すると、それらに `citus-group: ${env.PATRONI_CITUS_GROUP}` ラベルが自動的に配置されます。

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: citusdemo-0-leader # Is generated as ${env.PATRONI_SCOPE}-${env.PATRONI_
↪CITUS_GROUP}-leader
  labels:
    application: patroni # Is set from the ${env.PATRONI_KUBERNETES_LABELS}
    cluster-name: citusdemo # Is automatically set from the ${env.PATRONI_SCOPE}

```

(次のページに続く)

(前のページからの続き)

```
citus-group: '0'           # Is automatically set from the ${env.PATRONI_CITUS_GROUP}
```

Patroni リポジトリの ``kubernetes`` フォルダーに、Citus サポートを備えた Kubernetes への Patroni デプロイメントの完全な例があります。

2つの重要なファイルがあります。

1. `Dockerfile.citus`
2. `citus_k8s.yaml`

10.6 Citus のアップグレードと PostgreSQL のメジャーアップグレード

まず、`documentation`` の Citus バージョンアップについてお読みください。プロセスに 1 つのマイナーな変更があります。アップグレードを実行するときは、``systemctl restart`` ではなく `patronictl restart` を使用して PostgreSQL を再起動する必要があります。

Citus を使用した PostgreSQL のメジャーアップグレードは、もう少し複雑です。メジャーアップグレードに関する Citus のドキュメントと *PostgreSQL major upgrade* に関する Patroni のドキュメントで使用されている手法を組み合わせる必要があります。Citus クラスターは多くの Patroni クラスター（コーディネーターとワーカー）で構成されており、すべて個別にアップグレードする必要があることに注意してください。

第11章 スタンドアロンをPatroniクラスターに変換する

このセクションでは、スタンドアロンの PostgreSQL インスタンスを Patroni クラスターに変換するプロセスについて説明します。

既存の PostgreSQL インスタンスを使用せずに Patroni クラスターをデプロイするには、代わりに *Running and Configuring* を参照してください。

11.1 プロシージャ

以下は、既存の Postgres クラスターを Patroni マネージドクラスターに変換する手順の概要です。手順では、既存のクラスターの一部であるすべてのノードが現在稼働中であり、移行の進行中に Postgres 構成を変更する予定がないことを前提としています。手順：

1. Patroni 構成の *authentication* セクションの説明に従って、Postgres ユーザーを作成します。以下のコードブロックにユーザーを作成するサンプル SQL コマンドがあります。ここでは、環境に応じてユーザー名とパスワードを置き換える必要があります。関連するユーザーが既にいる場合は、この手順をスキップできます。

```
-- Patroni superuser
-- Replace PATRONI_SUPERUSER_USERNAME and PATRONI_SUPERUSER_PASSWORD accordingly
CREATE USER PATRONI_SUPERUSER_USERNAME WITH SUPERUSER ENCRYPTED PASSWORD
↳ 'PATRONI_SUPERUSER_PASSWORD';

-- Patroni replication user
-- Replace PATRONI_REPLICATION_USERNAME and PATRONI_REPLICATION_PASSWORD
↳ accordingly
CREATE USER PATRONI_REPLICATION_USERNAME WITH REPLICATION ENCRYPTED PASSWORD
↳ 'PATRONI_REPLICATION_PASSWORD';

-- Patroni rewind user, if you intend to enable use_pg_rewind in your Patroni
↳ configuration
-- Replace PATRONI_REWIND_USERNAME and PATRONI_REWIND_PASSWORD accordingly
CREATE USER PATRONI_REWIND_USERNAME WITH ENCRYPTED PASSWORD 'PATRONI_REWIND_
↳ PASSWORD';
GRANT EXECUTE ON function pg_catalog.pg_ls_dir(text, boolean, boolean) TO
↳ PATRONI_REWIND_USERNAME;
GRANT EXECUTE ON function pg_catalog.pg_stat_file(text, boolean) TO PATRONI_
```

(次のページに続く)

(前のページからの続き)

```

↪REWIND_USERNAME;
GRANT EXECUTE ON function pg_catalog.pg_read_binary_file(text) TO PATRONI_REWIND_
↪USERNAME;
GRANT EXECUTE ON function pg_catalog.pg_read_binary_file(text, bigint, bigint,
↪boolean) TO PATRONI_REWIND_USERNAME;

```

2. すべての Postgres ノードで次の手順を実行します。次のノードに進む前に、1つのノードですべての手順を実行します。プライマリノードから始めて、各スタンバイノードに進みます。

1. systemd を介して Postgres を実行している場合は、Postgres systemd ユニットを無効にします。これは、Patroni が Postgres デーモンの起動と停止を管理するときに実行されます。

2. Patroni 用の YAML 構成ファイルを作成します。

- **Note (specific for the primary node):** クラスターメンバー間のレプリケーションに使用されているレプリケーションスロットがある場合は、`use_slots` を有効にし、`slots` 構成項目を介して既存のレプリケーションスロットを永続的に構成することをお勧めします。Patroni は、`use_slots` が有効になっている場合、メンバー間のレプリケーション用のレプリケーションスロットを自動的に作成し、認識しないレプリケーションスロットをドロップすることに注意してください。ここで永続的なスロットを使用するという考えは、Patroni への移行が進行中の間、既存のスロットを持続できるようにすることです。詳細は **Note (specific for the primary node):** を参照してください。

3. `patroni systemd` サービスユニットを使用して Patroni を起動します。Postgres が既に実行されていることを自動的に検出し、インスタンスの監視を開始します。

3. Postgres/"起動手順/"を Patroni に引き渡します。これを行うには、`patronictl restart cluster-name member-name` コマンドを使用してクラスターメンバーを再起動する必要があります。ダウンタイムを最小限に抑えるには、この手順を次のように分割できます。

1. スタンバイノードの即時リスタート。
2. メンテナンスウィンドウ内のプライマリノードのスケジュールされた再起動。

4. ステップ 1.2. で永久スロットを構成した場合、Patroni によって作成されたスロットの `restart_lsn` が対応するメンバーの元のスロットの `restart_lsn` に追いつくことができれば、`patronictl edit-config cluster-name member-name` コマンドを介して `slots` 構成からそれらを削除する必要があります。slots 構成からスロットを削除することにより、Patroni は、不要になったらクラスターから元のスロットを削除できます。以下に、いくつかのスロットの `restart_lsn` をチェックするクエリの例があるので、比較できます。

```

-- Assume original_slot_for_member_x is the name of the slot in your original
-- cluster for replicating changes to member X, and slot_for_member_x is the
-- slot created by Patroni for that purpose. You need restart_lsn of
-- slot_for_member_x to be >= restart_lsn of original_slot_for_member_x
SELECT slot_name,
       restart_lsn
FROM pg_replication_slots
WHERE slot_name IN (

```

(次のページに続く)

(前のページからの続き)

```
'original_slot_for_member_x',  
'slot_for_member_x'  
)
```


第12章 PostgreSQL バージョンのメジャーアップグレード

現在メジャーアップグレードを行う唯一の方法は次のとおりです。

1. ストップパトローニ
2. PostgreSQL バイナリをアップグレードし、プライマリノードで `pg_upgrade` を実行します
3. `Patani.yml` を更新する
4. DCS から初期化キーを削除するか、DCS から完全なクラスター状態をワイプします。2 番目は `patronictl remove <cluster-name>` を実行することで達成できます。 `pg_upgrade` は `initdb` を実行し、実際に新しい PostgreSQL システム識別子で新しいデータベースを作成するためです。
5. 前の手順でクラスターの状態をワイプした場合、`パトローニ.dynamic.json` を古いデータディレクトリから新しいデータディレクトリにコピーすることをお勧めします。以前に設定したいくつかの PostgreSQL パラメーターを保持するのに役立ちます。
6. プライマリノードで Patroni を起動します。
7. PostgreSQL バイナリをアップグレードし、`atoani.yml` を更新して、スタンバイノードの `data_dir` をワイプします。
8. スタンバイノードで Patroni を起動し、レプリケーションが完了するまで待ちます。

スタンバイノードでの `pg_upgrade` の実行は、PostgreSQL ではサポートされていません。自分が何をしているのか知っている場合は、スタンバイノードで `data_dir` を消去する代わりに、<https://www.postgresql.org/docs/current/pgupgrade.html> で説明されている `rsync` 手順を試すことができます。ただし、最も安全な方法は、Patroni にデータを複製させることです。

12.1 よくある質問

- Patroni の起動中に、Patroni は PostgreSQL ポートにバインドできないと不平を言っています。

`postgresql.conf` の `listen_addresses` と `port`、`patroni.yml` の `postgresql.listen` を確認する必要があります。 `pg_hba.conf` はそのようなアクセスを許可する必要があることを忘れないでください。

- Patroni にノードの再起動を要求した後、PostgreSQL はエラーメッセージ `could not open configuration file "/etc/postgresql/10/main/pg_hba.conf/": No such file or directory` を表示します

PostgreSQL 構成を管理する方法に応じて、さまざまなことを意味します。*postgresql.config_dir* を指定した場合、*Patroni* は新しいクラスターをブートストラップするときにのみ、`pg_hba.conf` セクションの設定に基づいて `pg_hba.conf` を生成します。このシナリオでは、PGDATA が空ではなかったため、ブートストラップは発生しませんでした。このファイルは事前に存在する必要があります。

第13章 セキュリティに関する考慮事項

Patroni クラスターには、不正アクセスから保護する 2 つのインターフェイスがあります。分散構成ストレージ (DCS) と PatroniRESTAPI です。

13.1 DCS の保護

Patroni とパトロニ `ctl` はどちらも DCS との間でデータを保存および取得します。

DCS には機密情報が含まれていませんが、Patroni/Postgres 構成の一部を変更できます。したがって、最初に保護する必要があるのは DCS 自分自身です。

保護の詳細は、使用される DCS の種類によって異なります。サポートされているタイプの DCS の認証および暗号化パラメーター (トークン/基本認証/クライアント証明書) は、[settings](#) でカバーされています。

一般的な推奨事項は、すべての DCS 通信で TLS を有効にすることです。

13.2 RESTAPI の保護

RESTAPI の保護は、より複雑なタスクです。

PatroniRESTAPI は、リーダーレース中に Patroni 自分自身、フェイルオーバー/スイッチオーバー/再初期化/リスタート/リロードを実行するために `patronictl` ツール、HTTP ヘルスチェックを実行するために HAProxy または他の種類のロードバランサー、そしてもちろんモニタリングにも使えます。

セキュリティの観点から、RESTAPI には安全な (GET リクエスト、情報を取得のみ) および安全でない (PUT、POST、PATCH および DELETE リクエスト、ノードの状態を変更する) エンドポイントが含まれています。

`restapi.authentication.username` および `restapi.authentication.password` パラメーターを設定することにより、安全でないエンドポイントを HTTP 基本認証で保護できます。TLS を有効にせずに安全なエンドポイントを保護する方法はありません。

RESTAPI の TLS を有効にし、PKI を確立すると、すべてのエンドポイントで API サーバーと API クライアントの相互認証が可能になります。

`restapi` セクションのパラメーターは、サーバーに対する TLS クライアント認証を有効にします。`verify_client` パラメーターの値に応じて、API サーバーは、安全な API 呼び出しと安全でない API 呼び出しの両方 (`verify_client: required`)、または安全でない API 呼び出しのみ (`verify_client: optional`)、または API 呼び出しがない (`verify_client: none`) クライアント証明書の検証に成功する必要があります。

`ctl` セクションのパラメーターは、クライアントへの TLS サーバー認証を有効にします（パトローニと同じ構成を使用する `patronictl` ツール）。`insecure: true` を設定して、クライアントによるサーバー証明書の検証を無効にします。TLS クライアントパラメーターの詳細については、`ctl` を参照してください。

不正アクセスから PostgreSQL データベースを適切に保護することはこのドキュメントの範囲を超えており、<https://www.postgresql.org/docs/current/client-authentication.html> でカバーされています。

第14章 HA マルチデータセンター

複数のデータセンターにデプロイされた PostgreSQL クラスターの高可用性は、レプリケーションに基づいています。レプリケーションは、同期または非同期（`replication_modes`）の場合があります。

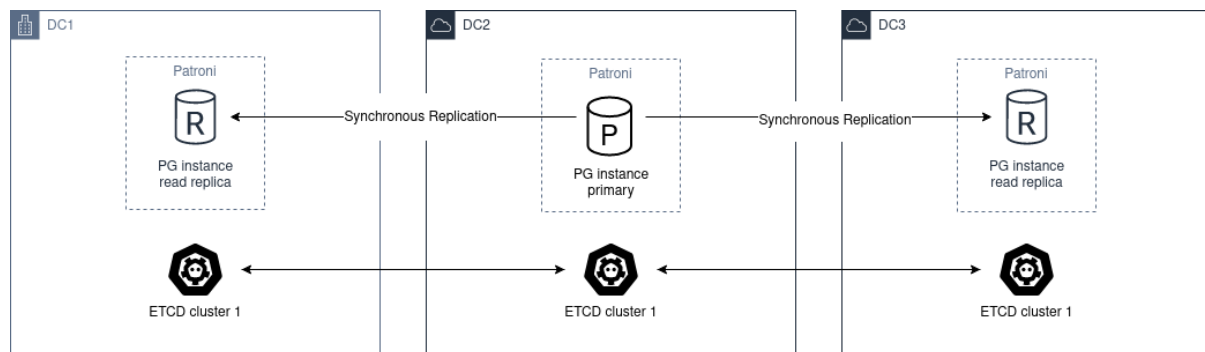
どちらの場合も、次の概念について明確にすることが重要です。

- Postgres は、リーディングキーを所有し、リーディングキーを更新できる場合にのみ、プライマリまたはスタンバイリーダーとして実行できます。
- 奇数の etcd、ZooKeeper、または Consul ノードを実行する必要があります：3 または 5！

14.1 同期レプリケーション

ゾーンのドロップを自動的に許容できるマルチ DC クラスターを持つには、少なくとも 3 つが必要です。

アーキテクチャ図は次のようになります。



etcd、ZooKeeper、または Consul のクラスターを、異なる DC を介して、各ゾーンに 1 つずつ、少なくとも 3 つのノードでデプロイする必要があります。

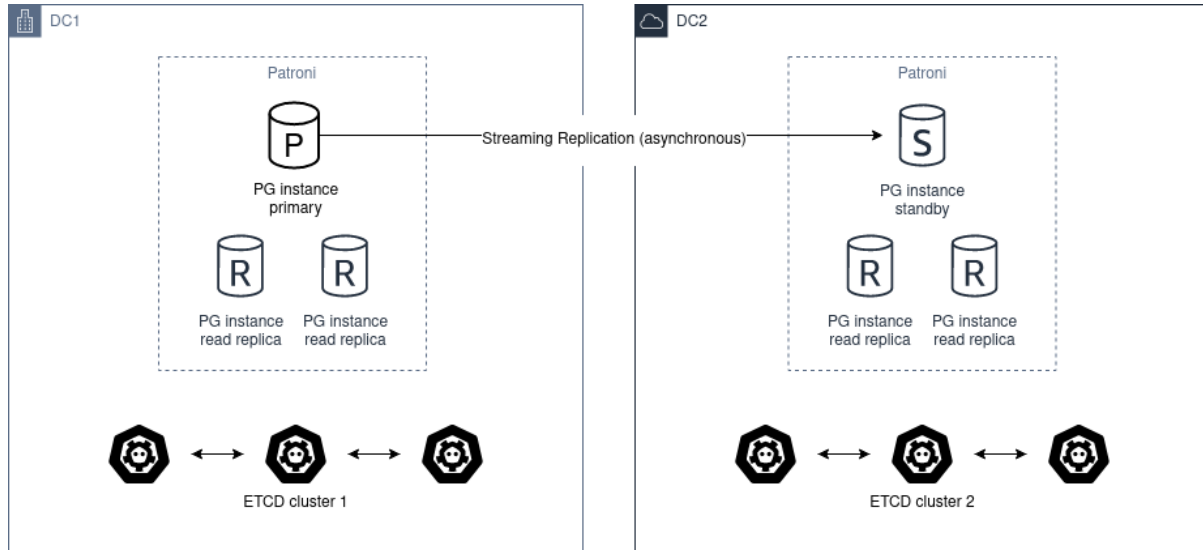
postgres に関しては、異なる DC に少なくとも 2 つのノードを展開する必要があります。次に、グローバル `synchronous_mode: true` で `synchronous_mode: true` を設定する必要があります。

これにより、同期レプリケーションが有効になり、プライマリノードはノードの 1 つを同期として選択します。

14.2 非同期レプリケーション

データセンターが2つしかない場合、2つの独立した etcd クラスターを用意し、2 番目のデータセンターで Patroni *standby cluster* を実行する方が良いでしょう。最初のサイトがダウンしている場合は、`standby_cluster` 手動で昇格できます。

アーキテクチャ図は次のようになります。



DC2 は DC1 の状態を決して理解できないため、自動昇格はできません。

このシナリオでは、`pg_ctl promote` を使用しないでください。`pg_ctl promote` から `standby_cluster` セクションを削除して、正常なクラスターを「手動でプロモート」する必要があります。

警告: ソースクラスターがまだ稼働していて、スタンバイクラスターを昇格させる場合、スプリットブレインを作成します。

初期の状態に戻したい場合、解決するには2つの方法しかありません。

- `standby_cluster` セクションを追加し直すと `pg_rewrite` がトリガーされますが、`pg_rewrite` が失敗する可能性があります。
- スタンバイクラスターを最初から再構築します。

スタンバイクラスターを昇格する前に、ソースクラスターがダウンしていることを手動で確認する必要があります (STONITH)。DC1 が回復したら、クラスターをスタンバイクラスターに変換する必要があります。

それを行う前に、データベースを手動で調べ、DC1 と DC2 間のネットワークが動作を停止してから DC1 のクラスターを手動で停止するまでの間に発生したすべての変更を抽出できます。

抽出したら、これらの変更を DC2 のクラスターに手動で適用することもできます。

第15章 リリースノート

15.1 バージョン3.1.0

Breaking changes

- `restapi.keyfile` と `restapi.certfile` のセマンティックを変更しました。(AlexanderKukushkin)

以前は、Patroni は `ctl` セクションにそれぞれの構成パラメーターがない場合、フォールバックとして `restapi.keyfile` および `restapi.certfile` をクライアント証明書として使用していました。

警告: クライアント証明書の検証を有効にした場合 (`restapi.verify_client` が `required` に設定されている場合)、`restapi.verify_client` も `ctl.certfile`、`ctl.keyfile`、`ctl.keyfile_password` で `restapi.verify_client` を提供します。提供されていない場合、Patroni は正しく動作しません。

New features

- Pod ロールラベルを構成可能にします。(Waynerv)

`kubernetes.leader_label_value`、`kubernetes.follower_label_value` および `kubernetes.standby_leader_label_value` パラメーターを使用して値をカスタマイズできます。この機能は、`master` ロールを `primary` に変更するときに非常に役立ちます。機能と移行手順 `kubernetes.leader_label_value` の詳細を読むことができます。

Improvements

- `patroni --validate-config` のさまざまな改善 (AlexanderKukushkin)

さまざまな DCS、`bootstrap.dcs`、`ctl`、`restapi`、および `watchdog` セクションのパラメーター検証が改善されました。

- Patroni の実行中にリカバリ中にクラッシュした場合、Postgres がリカバリしていない状態で起動します。(AlexanderKukushkin)

回復時間を短縮し、不必要なタイムラインの増分を防ぐのに役立ちます。

- `/status` キーの不必要な更新を回避しました。(AlexanderKukushkin)

永続的な論理スロットがない場合、Patroni は、プライマリの LSN が前進しなかった場合でも、ハートビートループごとに `/status` を更新していました。

- 古いプライマリーがリーダーレースに勝つことを許可しないでください (AlexanderKukushkin)

リソース不足のために Patroni がかなりの時間ハングアップしていた場合、リーダーロックを取得する前に、他のノードが Postgres を昇格させていないことをさらに確認します。

- 特定の PostgreSQL パラメーター検証の可視性を実装しました。(AlexanderKukushkin、FeikeSteenbergen)

`max_connections`、`max_wal_senders`、`max_prepared_transactions`、`max_locks_per_transaction`、`max_replication_slots`、または `max_worker_processes` の検証が失敗した場合 Patroni は、まともなデフォルト値を使用していました。それに加えて、警告も表示されます。

- PGDATA で作成されたファイルとディレクトリのアクセス許可を設定します。(AlexanderKukushkin)

Patroni によって作成されたすべてのファイルには、所有者の読み取り/書き込み権限のみがありました。この動作は、別のユーザーで実行され、グループの読み取り権限に依存しているバックアップツールを壊していました。Patroni は PGDATA のパーミッションを尊重し、PGDATA 内に作成するすべてのディレクトリとファイルのパーミッションを正しく設定します。

Bugfixes

- シェルを介して `archive_command` を実行します (Waynerv)

Patroni は、シングルユーザーモードでクラッシュリカバリを実行する前、または `pg_rewrite` の前に、いくつかの WAL セグメントをアーカイブする場合があります。 `archive_command` に `&&` のようないくつかのシェルオペレーターが含まれている場合、Patroni では動作しませんでした。

- `/"onswitchover/"` シャットダウンチェックを修正しました (PolinaBungina)

指定された候補がまだストリーミング中であり、シャットダウンチェックを受信しなかった可能性があります、他のいくつかのノードが正常であったため、リーダーキーが削除されました。

- `/"isprimary/"` チェックを修正しました (AlexanderKukushkin)

リーダーレース中、レプリカは古いリーダー上の Postgres がまだプライマリとして実行されていることを認識できませんでした。

- `patronictl list` を修正 (AlexanderKukushkin)

`tsv`、`json`、および `yaml` 出力形式にクラスター名フィールドがありませんでした。

- 一時停止後の `pg_rewrite` の動作を修正しました。(AlexanderKukushkin)

特定の条件下では、Patroni は、メンテナンスモードを終了した後、`pg_rewrite` を使用して `false` プライマリをクラスターに参加できませんでした。

- Etcdv3 実装のバグを修正しました (AlexanderKukushkin)

リビジョンの不一致が原因で `create_revision/mod_revision` フィールドを使用してキーの更新が実行された場合、内部 KV キャッシュを無効にします。

- 一時停止中のスタンバイクラスターのレプリカの動作を修正しました。(AlexanderKukushkin)

リーダーキーの有効期限が切れると、スタンバイクラスターのレプリカはリモートノードを追跡せず、`primary_conninfo` をそのまま維持します。

15.2 バージョン 3.0.4

New features

- スタンバイノードのレプリケーションステータスを可視化する (AlexanderKukushkin)

PostgreSQL9.6 以降の場合、Patroni は、スタンバイが他のノードからストリーミングしている場合は `streaming`、レプリケーション接続がなく、`restore_command` が設定されている場合は `in archive recovery` としてレプリケーション状態を報告します。状態は、DCS の `member` キー、RESTAPI、および `patronictl list` 出力に表示されます。

Improvements

- Etcdv3 のエラーメッセージを改善しました (AlexanderKukushkin)

Etcdv3 クラスターにアクセスできない場合、パトローニは `/v2` エンドポイントにアクセスできないことを報告していました。

- 可能であれば `patronictl` でクォーラム読み取りを使用します。 (AlexanderKukushkin)

Etcd または Consul クラスターは読み取り専用に低下させることができましたが、`patronictl` ビューからはすべて問題ありませんでした。これで、エラーで失敗します。

- `splitbrain` が構成内の名前を重複しないようにしました。 (MarkPekala)

Patroni の起動時に、同じ名前のノードが DCS に登録されているかどうかを確認し、その RESTAPI をクエリしようとします。RESTAPI にアクセスできる場合 Patroni はエラーで終了します。ヒューマンエラーから身を守るのに役立ちます。

- Patroni の実行中にクラッシュした場合、Postgres が復旧していない状態で起動します。 (AlexanderKukushkin)

回復時間を短縮し、不必要なタイムラインの増分から役立ちます。

Bugfixes

- SIGHUP を受信しても RESTAPISSL 証明書がリロードされませんでした。 (IsraelBarthRubio)

回帰は 3.0.3 で導入されました。

- `max_connections` などのパラメーターの整数 GUC 検証を修正しました。 (FeikeSteenbergen)

Patroni は、引用符で囲まれた数値が好きではありませんでした。回帰は 3.0.3 で導入されました。

- `synchronous_mode` の問題を修正 (AlexanderKukushkin)

`txid_current()` を `synchronous_commit=off` で実行して、`synchronous_mode_strict` が有効になっているときに誤って不在の同期スタンバイを待機しないようにします。

15.3 バージョン 3.0.3

New features

- PostgreSQL16beta1 との互換性 (AlexanderKukushkin)

拡張 GUC のバリデータールール。

- PostgreSQLGUC のバリデータを拡張可能にしました。 (IsraelBarthRubio)

バリデータールールは、`patroni/postgresql/available_parameters/` ディレクトリにある YAML ファイルからロードされます。ファイルはアルファベット順に並べられ、次々と適用されます。非標準の Postgres ディストリビューション用のカスタムバリデーターを持つことが可能になります。

- `restapi.request_queue_size` オプションを追加しました。 (AndreyZhidenkov、AleksseiSukhov)

PatroniRESTAPI が使用する TCP ソケットのリクエストキューサイズを設定します。キューがいっぱいになると、それ以降の要求には「接続が拒否されました」エラーが発生します。デフォルト値は 5 です。

- 新しいクラスターを初期化するときに `initdb` を直接呼び出します。 (MattBaker)

以前は `pg_ctl` を介して呼び出されましたが、`initdb` に渡されるパラメーターの特別な引用が必要でした。

- ストップフックの前に追加しました (LeDuane)

フックは `postgresql.before_stop` を介して構成でき、`pg_ctl stop` の直前に実行されます。終了コードは、シャットダウンプロセスには影響しません。

- カスタム Postgres バイナリ名のサポートを追加しました。 (IsraelBarthRubio、PolinaBungina)

カスタム Postgres ディストリビューションを使用する場合、コミュニティ Postgres ディストリビューションで使用される名前とは異なる名前で Postgres バイナリがコンパイルされる場合があります。カスタムバイナリ名は、`postgresql.bin_name`、* および ``PATRONI\_POSTGRESQL\_BIN\_`` 環境変数を使用して構成できます。

Improvements

- `patroni --validate-config` のさまざまな改善 (PolinaBungina)

- `bootstrap.initdb` をオプションにします。新しいクラスターにのみ必要ですが、`patroni --validate-config` は、構成に欠落していると不平を言っていました。
- `postgresql.bin_dir` が空または設定されていないときにエラーになりません。代わりに、最初にデフォルトの PATH で Postgres バイナリを見つけてみてください。
- `postgresql.authentication.rewind` セクションをオプションにします。欠落している場合、Patroni はスーパーユーザーを使用しています。

- `patronictl` のエラー報告を改善しました。 (IsraelBarthRubio)

`/n` シンボルは、実際の改行記号ではなく、そのままレンダリングされました。

Bugfixes

- Citus サポートの問題を修正しました (AlexanderKukushkin)

昇格したワーカーからコーディネーターへの RESTAPI 呼び出しがスイッチオーバー中に失敗した場合、指定された Citus グループは無期限にブロックされたままになりました。

- `patronictl` の --dcs-url` オプションで `etcd3` URL を許可しました。 (IsraelBarthRubio)`

ユーザーが `patronictl` の --dcs-url` オプションを介して `etcd3` URL を渡そうとすると、例外に直面します。`

15.4 バージョン 3.0.2

警告: バージョン 3.0.2 では、3.6 より古い Python のサポートが終了しました。

New features

- `/metrics` エンドポイントに同期スタンバイレプリカステータスを追加しました。 (ThomasvonDein、AlexanderKukushkin)

以前は `primary / standby_leader / replica` のみを報告していました。

- `patronictl` の `PAGER` のユーザーフレンドリーな処理 (IsraelBarthRubio)

これは、デフォルトの `less` および `more` をオーバーライドする `PAGER` 環境変数を介してページャーを構成可能にします。

- K8s の再試行可能な HTTP ステータスコードを構成可能にしました。 (AlexanderKukushkin)

一部の管理対象プラットフォームでは、ステータスコード `401 Unauthorized` を取得することができます。これは、数回の再試行後に解決される場合があります。

Improvements

- `recovery_target_action` が `promote` に設定されている場合のみ、カスタムブートストラップ中に `hot_standby` を `off` に設定します。 (AlexanderKukushkin)

`recovery_target_action=pause` を正しく動作させる必要がありました。

- `on_reload` コールバックが他のコールバックを強制終了できないようにしました。 (AlexanderKukushkin)

`on_start / on_stop / on_role_change` は通常、仮想 IP の追加/削除に使用され、`on_reload` はそれらを干渉しません。

- `aws` コールバックサンプルスクリプトで `IMDSFetcher` に切り替えました。 (PolinaBungina)

`IMDSv2` は使用するトークンを必要とし、`IMDSFetcher` はそれを透過的に処理します。

Bugfixes

- Kubernetes で実行されている Citus クラスターの `patronictl switchover` を修正しました。 (LukášLalinský)

default とは異なる名前空間では機能しませんでした。

- メジャーバージョンがわからない場合は PGDATA に書き込まないでください。(AlexanderKukushkin)

開始直後に PGDATA が空だった場合（おそらくまだマウントされていませんでした）、Patroni は PostgreSQL バージョンについて誤った仮定を行い、実際のメジャーバージョンが v10+であっても recovery.conf ファイルを誤って作成していました。

- コーディネーターフェイルオーバー後の Citus メタデータのバグを修正しました (AlexanderKukushkin)

citus_set_coordinator_host() 呼び出しはメタデータの同期を引き起こさず、変更はワーカーノードで表示されませんでした。この問題は、citus_update_node() に切り替えることで解決されます。

- すべての etcd ノードが「失敗」した場合のフォールバックとして、構成ファイルにリストされている etcd ホストを使用します。(AlexanderKukushkin)

etcd クラスターは時間の経過とともにトポロジを変更する可能性があり、Patroni はそれに追従しようとします。ある時点ですべてのノードが到達不能になった場合、Patroni は再接続しようとするときに、構成と最後の既知のトポロジのノードの組み合わせを使用します。

15.5 バージョン 3.0.1

Bugfixes

- 適切なロール名を on_role_change コールバックスクリプトに渡します。(AlexanderKukushkin, PolinaBungina)

Patroni は、昇格時に誤って promoted ロールを on_role_change コールバックスクリプトに渡していました。渡されたロール名は master に戻りました。このリグレッションは 3.0.0 で導入されました。

15.6 バージョン 3.0.0

このバージョンでは Citus_との統合が追加され、プライマリを降格せずに一時的な DCS の停止を生き残ることが可能になります。

警告:

- バージョン 3.0.0 は、Python2.7 をサポートする最後のリリースです。今後のリリースでは、3.7 より前の Python バージョンのサポートが終了します。
- RAFT サポートは非推奨です。メンテナンスには万全を期しておりますが、いかなる保証も責任も負えません。
- このバージョンは、"/primary/"を優先して、"/master/"を取り除くための最初のステップです。次のメジャーリリースへのアップグレードは、少なくとも 3.0.0 を実行している場合にのみ確実に機能します。

New features

- DCS フェイルセーフモード (AlexanderKukushkin, PolinaBungina)

この機能が有効になっている場合、Patroni クラスターは一時的な DCS の停止から生き残ることができます。 [documentation](#) で詳細を見つけることができます。

- Citus のサポート (AlexanderKukushkin, PolinaBungina, JelteFennema)

Patroni enables easy deployment and management of [Citus](#) clusters with HA. Please check [here](#) page for more information.

Improvements

- 不明だがアクティブなレプリケーションスロットをドロップするときに繰り返し発生するエラーを抑制しました。(MichaelBanck)

Patroni は引き続きこれらのログを書き込みますが、DEBUG にのみ書き込みます。

- HA ループごとに 1 つのモニタリングクエリのみを実行します。(AlexanderKukushkin)

同期レプリケーションが有効になっている場合はそうではありませんでした。

- 最新の失敗データディレクトリのみを保持します。(WilliamAlbertusDembo)

ブートストラップが失敗した場合、Patroni はタイムスタンプサフィックスを使用して \$PGDATA フォルダーの名前を変更していました。今後、サフィックスは `.failed` になり、そのようなフォルダーが存在する場合、名前を変更する前に削除されます。

- 同期レプリケーション接続のチェックを改善しました。(AlexanderKukushkin)

新しいホストが `synchronous_standby_names` に追加されると、`pg_stat_replication.sync_state` = 'sync' に加えてプライマリに追いつくことができた場合にのみ、DCS で同期として設定されます。

Removed functionality

- `patronictl scaffold` を削除します (AlexanderKukushkin)

それを持っている唯一の理由は、スタンバイクラスターを実行するハックな方法でした。

15.7 バージョン 2.1.7

Bugfixes

- レガシー Python モジュールとの小さな非互換性を修正しました (AlexanderKukushkin)

彼らは、Debianbuster/Ubuntubionic で Patroni をビルド/実行することを防ぎました。

15.8 バージョン 2.1.6

Improvements

- SSL ソケットのシャットダウン時の迷惑な例外を修正しました。(AlexanderKukushkin)

HAProxy は、HTTP ステータスコードを取得するとすぐに接続を閉じます。パトローニが SSL 接続を適切にシャットダウンする時間はありません。

- arm64 のサンプル Dockerfile を調整します (PolinaBungina)

明示的な amd64 および x86_64 を削除します。libnss_files.so.* を削除しないでください。

Security improvements

- 非レプリケーション接続に `search_path=pg_catalog` を強制します。(AlexanderKukushkin)

Patroni はスーパーユーザー接続に大きく依存しているため、`pg_catalog` の対応するオブジェクトと同じ名前と署名を持つ `public` スキーマのユーザー定義関数および/またはオペレーターを使用して実行される攻撃から保護したいと思います。そのため、`search_path=pg_catalog` は Patroni によって作成されたすべての接続（レプリケーション接続を除く）に適用されます。

- `pg_stat_statements` にパスワードが記録されないようにしました。(FeikeSteenbergen)

これは、ユーザーの作成時に `pg_stat_statements.track_utility=off` を設定することで実現されます。

Bugfixes

- `proxy_address` をオプションとして宣言します。(DenisLaxalde)

それは事実上非必須オプションであるため。

- 安全でないオプションの動作を改善しました。(AlexanderKukushkin)

RESTAPI リクエストにクライアント証明書が使用された場合、ctl の `insecure` オプションが正しく機能しませんでした。

- 新しいクラスターがブートストラップされるときに `bootstrap.dcs` からウォッチドッグ構成を取得します。(MattBaker)

Patroni は、DCS のブートストラップに使用される構成を取得するのではなく、新しいクラスターをブートストラップするときにデフォルトでウォッチドッグを最初に構成していました。

- WIN32 で実行可能ファイルを見つける際のファイル拡張子の処理方法を修正しました。(MartínMarqués)

まだ拡張子がない場合にのみ、ファイル名に `.exe` を追加します。

- ConsulTTL セットアップを修正 (AlexanderKukushkin)

HTTPClient に値を設定するときに `ttl/2.0` を使用しましたが、クラスのプロパティで現在の値を 2 倍するのを忘れていました。その結果、ConsulTTL が 2 回オフになりました。

Removed functionality

- `patronictl configure` を削除します (PolinaBungina)

個別の `patronictl` 構成を作成する必要はもうありません。

15.9 バージョン 2.1.5

このバージョンでは、PostgreSQL15 との互換性が強化され、Etcdv3 サポートが本番環境対応として宣言されています。いかなどのパトローニはベータのままです。

New features

- `patroni --validate-config` を改善 (DenisLaxalde)

`config` が無効な場合はコード 1 で終了し、エラーを `stderr` に出力します。

- 一時停止中にレプリケーションスロットをドロップしないでください (AlexanderKukushkin)

Patroni は、メンバーがクラスターに参加/離脱するときに、物理レプリケーションスロットを自動的に作成/削除しています。一時停止中のスロットは削除されなくなります。

- エンドポイントをモニタリングするための `HEAD` リクエストメソッドをサポートしました。 (Robert-Cutajar)

`GET` の代わりに使用された場合、Patroni は HTTP ステータスコードのみを返します。

- Windows での動作テストをサポートしました。 (AlexanderKukushkin)

新しい RESTAPI エンドポイント `POST /sigterm` を導入することにより、Windows での Patroni のグレースフルシャットダウン (`SIGTERM`) をエミュレートします。

- `postgres.proxy_address` 紹介 (AlexanderKukushkin)

DCS のメンバーキーに `proxy_url` として書き込まれ、サービス検出に使用/役立ちます。

Stability improvements

- スレッドから `pg_replication_slot_advance()` を呼び出す (AlexanderKukushkin)

多くの論理レプリケーションスロットがあるビジー状態のクラスターでは、`pg_replication_slot_advance()` 呼び出しがメインの HA ループに影響を与え、メンバーキーの有効期限が切れる可能性があります。

- 古いプライマリで `pg_rewind` を呼び出す前に、欠落している可能性のある WAL をアーカイブしました。 (PolinaBungina)

プライマリがクラッシュし、かなりの時間ダウンしていた場合、一部の WAL ファイルがアーカイブおよび新しいプライマリから欠落している可能性があります。`pg_rewind` が古いプライマリからこれらの WAL ファイルを削除し、スタンバイとして起動できなくなる可能性があります。`ready` WAL ファイルをアーカイブすることにより、この問題を軽減するだけでなく、一般的にアーカイブエクスペリエンスを継続的に改善します。

- KubernetesService を作成しようとするときに `403` エラーを無視します。 (NickHudson, PolinaBungina)

Patroni は、サービスを作成しようとして失敗し、ログをスパムしていましたが、実際にはすでに存在している可能性があります。

- liveness プローブを改善する (AlexanderKukushkin)

ハートビートループがプライマリで `ttl` またはレプリカで `2\*ttl` よりも長く実行されている場合、活性の問題は失敗し始めます。これにより、Kubernetes で *watchdog* の代替として使用できます。

- スイッチオーバー時に同期ノードのみがロックを取得しようとするようにしました。 (AlexanderKukushkin, PolinaBungina)

以前は、ターゲットを指定せずに手動スイッチオーバーが実行された場合、最新の非同期メンバーがリーダーになる可能性はわずかでした。

- ブートストラップの実行中のクローン作成を回避しました。 (AntsAasma)

クラスターブートストラップの実行中にトリガーされるリーダーを必要としないレプリカ作成メソッドを許可しないでください。

- kazoo-2.9.0 (AlexanderKukushkin) との互換性

python のバージョンによっては、`select()` が閉じられたソケットで呼び出された場合、`SequentialThreadingHandler.select()` メソッドは `TypeError` および `IOError` 例外を発生させる場合があります。

- ソケットをシャットダウンする前に SSL 接続を明示的にシャットダウンしました。 (AlexanderKukushkin)

それを行わないと、OpenSSL3.0 で `unexpected eof while reading` エラーが発生しました。

- `prettytable>=2.2.0` との互換性 (AlexanderKukushkin)

内部 API の変更により、クラスター名ヘッダーが間違っただけに表示されました。

Bugfixes

- Etcd の期限切れのトークンを処理する `lease_grant` (monsterxx03)

エラーが発生した場合は、新しいトークンを取得して要求を再試行します。

- GET `/read-only-sync` エンドポイントのバグを修正しました。 (AlexanderKukushkin)

以前のリリースで導入され、効果的に機能しませんでした。

- `datadirstorage` が消えた場合の対処法 (AlexanderKukushkin)

Patroni は、PGDATA が存在し、空ではないことを定期的にチェックしていますが、ストレージに問題がある場合、`os.listdir()` は `OSError` 例外を発生させ、ハートビートループを壊しています。

- ユーザーのバックエンドが閉じるのを待っているときに `master_stop_timeout` を適用しました。 (AlexanderKukushkin)

ユーザーバックエンドのように見えるものは、実際には停止に失敗したバックグラウンドワーカー (例、`CitusMaintenanceDaemon`) である可能性があります。

- `postgresql.listen` に `*:<port>` を受け入れました (DenisLaxalde)

`patroni --validate-config` は、無効であると不平を言っていました。

- Raft でのタイムアウトの修正 (AlexanderKukushkin)

Patroni または `atonicctl` が起動すると、既知のメンバーから Raft クラスタトポロジーを取得しようとします。これらの呼び出しは、適切なタイムアウトなしで行われました。

- トークンが変更された場合、Consul サービスを強制的に更新します。(JohnA.Lotoski)

そうしないと、エラー「`"/rpcerror メーキングコール：rpc エラーメーキングコール：ACL が見つかりません/"`」が発生します。

15.10 バージョン 2.1.4

New features

- 典型的な Debian/Ubuntu システムでの `pg_rewind` の動作を改善しました。(Gunnar/"Nick"/Bluth)
`postgresql.conf` をデータディレクトリの外部に保持する *Postgres セットアップ (Ubuntu/Debian パッケージなど)* では、``pg_rewind --restore-target-wal`` は `restore_command` の値を把握できません。
- Consul サービスチェックで `TLSServerName` 設定を許可しました。(MichaelGmelin)
 チェックが IP によって実行され、Consul `node_name` が FQDN でない場合に役立ちます。
- ウォッチドッグに `ppc64le` のサポートを追加しました。(Jean-MichelScheiwiler)
 また、一部の非 x86 プラットフォームでのウォッチドッグサポートを修正しました。
- `aws.py` コールバックを `boto` から `boto3` に切り替えました。(AlexanderKukushkin)
`.x` は 2018 年以來放棄され、`python3.9` で失敗します。
- K8s (HaitaoLi) のサービスアカウントトークンを定期的に更新します
 Kubernetes v1.21 以降、サービスアカウントトークンの有効期限は 1 時間です。
- `/read-only-sync` モニタリングエンドポイントを追加しました (Dennis4b)
`/read-only` に似ていますが、同期レプリカのみが含まれています。

Stability improvements

- プライマリとの論理デコードセットアップに構成の不一致がある場合、論理レプリケーションスロットをレプリカにコピーしないでください。(AlexanderKukushkin)
 スロットが `plugin` または `database` 構成オプションと一致しない場合、レプリカはプライマリから論理レプリケーションスロットをコピーしません。以前は、スロットがこれらの構成オプションと一致するかどうかのチェックは、レプリカがスロットをコピーして開始するまで実行されなかったため、不要な再起動が繰り返されました。
- PostgreSQL v12+ のリカバリ構成パラメーターの特別な処理 (AlexanderKukushkin)
 レプリカとして起動している間、Patroni は `postgresql.conf` を更新し、リーダーアドレスが変更された場合に再起動/リロードできるはずですが `pg_settings` から照会する代わりに、現在のパラメーター値をキャッシュすることにより。
- `postgresql.listen` パラメーターでの IPv6 アドレスの処理を改善しました。(AlexanderKukushkin)

`listen` パラメーターにはポートがあるため、人々は IPv6 アドレスを角括弧に入れようとしませんが、リストに複数の IP がある場合に正しく削除されませんでした。

- PostgreSQLv10 以前でのみダイバージェンスチェックを実行するときに `replication` 資格情報を使用します。(AlexanderKukushkin)

`rewind` が有効になっている場合、Patroni は再び新しい Postgres バージョンで `superuser` または `rewind` 資格情報を使用します。

Bugfixes

- `dateutil.parser` の欠落しているインポートを修正しました (WesleyMendes)

他のモジュールからインポートされたという理由だけで、テストが失敗したわけではありません。

- `optime` アノテーションが文字列であることを確認しました。(SebastianHasler)

特定の場合、Patroni はそれを数値として渡そうとしていました。

- `pg_rewind` の試行が失敗した場合の処理を改善しました。(AlexanderKukushkin)

`pg_rewind` 中にプライマリが利用できなくなった場合、`$PGDATA` は壊れた状態のままになります。その後、Patroni は、構成で許可されていない場合でも、データディレクトリを削除します。

- PostgreSQL の準備ができていないときにリーダー `ConfigMap / Endpoint` から `slots` アノテーションを削除しないでください。(AlexanderKukushkin)

`slots` 値が渡されない場合、注釈は現在の値を保持します。

- K8sAPI ウォッチャーで同時実行の問題を処理します。(AlexanderKukushkin)

特定の（不明な）条件下では、ウォッチャーが古くなる場合があります。その結果、HTTP ステータスコード 409 が原因で `attempt_to_acquire_leader()` メソッドが失敗する可能性があります。その場合、ウォッチャー接続をリセットし、最初から再起動します。

15.11 バージョン 2.1.3

New features

- `patronictl` の暗号化された TLS キーのサポートを追加しました。(AlexanderKukushkin)

`ctl.keyfile_password` または `PATRONI_CTL_KEYFILE_PASSWORD` 環境変数を介して構成できます。

- `/metrics` エンドポイントにメトリックを追加しました。(AlexandrePereira)

具体的には、`patroni_pending_restart` および `patroni_is_paused`。

- スタンバイクラスター構成で複数ホストを指定できるようにしました。(MichaelBanck)

スタンバイクラスターが Patroni クラスターから複製している場合、PostgreSQLv10 以降の `libpq` で利用可能なクライアント側のフェールオーバーに依存するといいかもかもしれません。つまり、スタンバイリーダーの `primary_conninfo` と `pg_rewind` は接続文字列に `target_session_attrs=read-write` を設定します。 `pgpass` ファイルは複数行（ホストごとに 1 行）で生成され、プライマリクラスターノードで `CHECKPOINT` を呼び出す代わりに、スタンバイクラスターは `pg_control` が更新されるのを待ちます。

Stability improvements

- レガシー `psycpg2` との互換性 (AlexanderKukushkin)

たとえば、Ubuntu18.04 パッケージからインストールされた `psycpg2` にはまだ `UndefinedFile` 例外がありません。

- すべての Etcd ノードが応答しない場合は `etcd3` ウォッチャーをリスタートしました。(AlexanderKukushkin)

ウォッチャーが生きている場合、すべての Etcd ノードに障害が発生していても、`get_cluster()` メソッドは古い情報を返し続けます。

- 一時停止中にスタンバイクラスターのリーダーロックを削除しないでください。(AlexanderKukushkin)

以前は、ロックはスタンバイリーダーではなく、プライマリとして実行されていたノードによってのみ維持されました。

Bugfixes

- スタンバイリーダーブートストラップのバグを修正しました (AlexanderKukushkin)

Patroni は、Postgres が 60 秒後に接続の受け入れを開始しない場合、ブートストラップが失敗したと考えていました。このバグは 2.1.2 リリースで導入されました。

- カスケードスタンバイへのフェイルオーバーに関するバグを修正しました (AlexanderKukushkin)

カスケードスタンバイでどのスロットを作成するかを判断するときに、リーダーが不在の場合があることを考慮するのを忘れていました。

- Postgres 構成バリデーターの小さな問題を修正しました。(AlexanderKukushkin)

PostgreSQLv14 で導入された整数パラメーターは、最小値と最大値が `validator.py` で引用されていたため、検証に失敗していました

- リーダーのステータスを確認するときにレプリケーション資格情報を使用するようになりました。(AlexanderKukushkin)

`remove_data_directory_on_diverged_timelines` が設定されている可能性があります、`rewind_credentials` が定義されておらず、ノード間のスーパーユーザーアクセスは許可されていません。

- RESTAPI 証明書の置き換えに関する `"/portinuse/"` エラーを修正しました。(AntsAasma)

証明書を切り替えると、同時 API 要求で競合状態が発生しました。置換期間中にアクティブなものと、置換は使用中のポートエラーでエラーになり、Patroni はアクティブな API サーバーのない状態でスタックします。

- パスワードに % 文字が含まれる場合のクラスターブートストラップのバグを修正しました。(Bastien-Wirtz)

ブートストラップメソッドは、すべてのパラメーターを適切に引用して `DO` ブロックを実行しますが、`cursor.execute()` メソッドは、パラメーターが渡された空のリストが好きではありませんでした。

- `"/AttributeError:noattribute'leader'/"` 例外を修正しました。(HrvojeMilković)

同期モードが有効になっていて、DCS コンテンツが一掃された場合に発生する可能性があります。

- ダイバージェンスタイムラインチェックのバグを修正しました。(AlexanderKukushkin)

Patroni は、タイムラインがずれていると誤って想定していました。pg_rewind の場合、問題は発生しませんでした。pg_rewind が許可されておらず、remove_data_directory_on_diverged_timelines が設定されている場合、元のリーダーが再初期化されました。

15.12 バージョン 2.1.2

New features

- psycopg>=3.0 (AlexanderKukushkin) との互換性

デフォルトでは psycopg2 が優先されます。psycopg>=3.0 は、`psycopg2` が利用できないか、そのバージョンが古すぎる場合にのみ使用されます。

- RESTAPI に dcs_last_seen フィールドを追加しました。(MichaelBanck)

このフィールドは、クラスターメンバーが DCS と正常に通信した最後に (UNIX エポックとして) 記録します。これは、ネットワークパーティションを特定および/または分析するのに役立ちます。

- pg_controldata が "shutdown/" を報告したらリーダーロックを解放します。(AlexanderKukushkin)

archive_command が遅い/失敗した場合のスイッチオーバー/シャットダウンが遅い問題を解決するために、Patroni は pg_controldata が PGDATA を shut down として報告し始め、すべての変更を受け取ったレプリカが少なくとも 1 つあることを確認した直後にリーダーキーを削除します。この条件を満たすレプリカがない場合、リーダーキーは削除されず、古い動作が保持されます。つまり、Patroni はロックを更新し続けます。

- sslcrldir 接続パラメータのサポートを追加しました。(KostiantynNemchenko)

新しい接続パラメーターは、PostgreSQLv14 で導入されました。

- Zookeeper で ZNode の ACL を設定できるようになりました。(AlwynDavis)

新しい構成オプション zookeeper.set_acls を導入して、Kazoo が作成する各 ZNode にデフォルトの ACL を適用します。

Stability improvements

- 次の HA ループまで次の回復の試みを遅らせる (AlexanderKukushkin)

Postgres がディスク容量不足 (たとえば) のためにクラッシュし、そのために起動に失敗した場合、Patroni が熱心に回復しようとしてログにあふれます。

- 降格する前にログを追加します。これには時間がかかる場合があります。(MichaelBanck)

降格が完了するまでに時間がかかる場合があり、ログを見ても正確に何が起きているのかが明らかではない場合があります。

- 「私は」ステータスメッセージを改善しました。(MichaelBanck)

no action. I am a secondary ({0}) と no action. I am ({0}), a secondary

- `wal_keep_segments` をグローバルな `wal_keep_segments` の文字列として指定することは可能です。Python は動的に型指定される言語であるため、文字列は単純に乗算されました。例：
wal_keep_segments: /"100/" 100100100100100100100100100100100100100100100100100100100100MB に変換されました。

- etcd クラスター内のアドバタイズされたクライアント URL が誤って構成されている可能性があります。この場合、Patroni で重複を削除するのは簡単な作業です。

Bugfixes

- Patroni はこの場合デフォルトの `statement_timeout` を使用していましたが、呼び出しが失敗すると回復しない可能性が非常に高く、`pg_wal` および `pg_catalog` のサイズが増加します。

- `/status` が demote 時に更新されませんでした。(AlexanderKukushkin)

PostgreSQL を降格した後、古いリーダーは DCS の最後の LSN を更新します。2.1.0 から始めて、新しい `/status` キーが導入されましたが、`optime` はまだ `/optime/leader` に書き込まれました。

- 降格時に DCS 例外を処理する (AlexanderKukushkin)

リーダーロックの更新に失敗したためにマスターを降格しているときに、DCS が完全にダウンし、`get_cluster()` 呼び出しで例外が発生する可能性があります。適切に処理されないと、DCS が回復するまで Postgres が停止したままになります。

- `use_unix_socket_repl` が機能しなかった場合があります。(AlexanderKukushkin)

具体的には、`postgresql.unix_socket_directories` が設定されていない場合。この場合、Patroni は `libpq` のデフォルト値を使用することになっています。

- PatroniRESTAPI に関するいくつかの問題を修正 (AlexanderKukushkin)

`clusters_unlocked` を定義できないことがあり、`GET /metrics` エンドポイントで例外が発生しました。それに加えて、エラー処理方法は `connect_address` タプルには常に 2 つの要素があると想定していましたが、実際には IPv6 の場合はもっとある可能性があります。

- 巻き戻しを決定する前に、新しく昇格したノードが回復を完了するのを待ちます。(AlexanderKukushkin)

実際のプロモートが行われ、新しいタイムラインが作成されるまでには、時間がかかる場合があります。レプリカは、待機することなく、巻き戻しは必要ないという結論に達する可能性があります。

- 巻き戻しを決定するときに履歴ファイルで欠落しているタイムラインを処理する (AlexanderKukushkin)

現在のレプリカのタイムラインがプライマリの履歴ファイルにない場合、レプリカは巻き戻しは必要ないと誤って想定していました。

15.13 バージョン 2.1.1

New features

- ETCDSRV 名前サフィックスのサポート (DavidPavlicek)

Etcd を使用すると、同じドメインの下複数の Etcd クラスターを区別できます。今後は Patroni もサポートします。

- 新しいリーダーで歴史を豊かにする (huiyalin525)

`patronictl history` 出力に新しい列を追加します。

- CA バンドルをクラスター内 Kubernetes 構成用に構成可能にします。(AronParsons)

デフォルトでは、Patroni は `/var/run/secrets/kubernetes.io/serviceaccount/ca.crt` を使用しており、この新機能によりカスタム `kubernetes.cacert` を指定できます。

- Consul サービスとしての動的な登録/登録解除とタグの変更をサポートしました。(TommyLi)

以前は、Patroni の再起動が必要でした。

Bugfixes

- RESTAPI の不必要なリロードを回避しました。(AlexanderKukushkin)

前のリリースでは、ディスク上で変更された場合に RESTAPI 証明書をリロードする機能が追加されました。残念ながら、スタート直後に無条件にリロードが発生していました。

- `etcd.use_proxies` が設定されているときにクラスターメンバーを解決しないでください。(AlexanderKukushkin)

起動時に、Patroni はメンバーのリストを照会して、Etcd クラスターの健全性をチェックします。それに加えて、ホスト名の解決も試みましたが、これはプロキシ経由で Etcd を操作する場合は不要であり、不要な警告を引き起こしていました。

- `pg_stat_replication` の NULL 値を持つ行をスキップします。(AlexanderKukushkin)

`pg_stat_replication` ビューは、`state = 'streaming'` 場合でも `replay_lsn`、`flush_lsn`、または `write_lsn` フィールドに NULL 値を含む可能性があるようです。

15.14 バージョン 2.1.0

このバージョンでは、PostgreSQLv14 との互換性が追加され、フェールオーバー/スイッチオーバーに耐える論理レプリケーションスロットが作成され、RESTAPI の許可リストのサポートが実装され、ハートビートごとにログ数が 1 行に削減されます。

New features

- PostgreSQLv14 (AlexanderKukushkin) との互換性

Patroni が "一時停止" モード自分自身でない場合は、WAL リプレイの一時停止を解除します。たとえば、プライマリ上の `max_connections` のような特定のパラメーターの変更が原因で、"一時停止" される可能性があります。

- フェールオーバー論理スロット (AlexanderKukushkin)

PostgreSQLv11 以降でのフェールオーバー/スイッチオーバーに耐える論理レプリケーションスロットを作成します。レプリケーションスロットは、再起動してプライマリからレプリカにコピーされ、後で `pg_replication_slot_advance()` 関数を使用して前方に移動します。その結果、スロットはフェールオーバー前にすでに存在し、イベントが失われることはありませんが、一部のイベントが複数回配信される可能性があります。

- PatroniRESTAPI の許可リストを実装しました (AlexanderKukushkin)

構成されている場合、ルールに一致する IP のみが安全でないエンドポイントを呼び出すことができます。それに加えて、クラスターのメンバーの IP をリストに自動的に含めることができます。

- UNIX ソケットを介したレプリケーション接続のサポートを追加しました。(MohamadEl-Rifai)

以前は、Patroni はレプリケーション接続に常に TCP を使用していたため、SSL 検証で問題が発生する可能性があります。UNIX ソケットを使用すると、レプリケーションユーザーを SSL 検証から免除できます。

- ユーザー定義タグのヘルスチェック (ArmanJafariTehrani)

predefined tags: とともに、`patronictl list` 出力および RESTAPI で可視になるカスタムタグをいくつでも指定できます。今後は、ヘルスチェックでカスタムタグを使用できるようになります。

- Prometheus `/metrics` エンドポイントを追加しました。(MarkMercado、MichaelBanck)

`/patroni` と同じメトリックを公開するエンドポイント。

- Patroni ログのおしゃべりの減少 (AlexanderKukushkin)

すべてが正常に進むと、HA ループの実行ごとに 1 行のみが書き込まれます。

Breaking changes

- 古い `permanent logical replication slots` 機能は、PostgreSQLv10 以前では動作しません。(AlexanderKukushkin)

プロモーションの実行後に論理スロットを作成する戦略では、論理イベントが失われず、無効にならないことを保証できません。

- `/leader` エンドポイントは、ノードがロックを保持している場合常に 200 を返します。(AlexanderKukushkin)

スタンバイクラスターを昇格するには、ロードバランサーのヘルスチェックを更新する必要がありますが、これはあまり便利ではなく、忘れがちです。それを解決するために、`/leader` ヘルスチェックエンドポイントの動作を変更します。クラスターがノーマルか `standby_cluster` かを考慮せずに 200 を返します。

Improvements in Raft support

- Raft トラフィック暗号化の信頼できるサポート (AlexanderKukushkin)

`PySyncObj` のさまざまな問題により、暗号化のサポートは非常に不安定でした

- Raft 実装で DNS の問題を処理する (AlexanderKukushkin)

`self_addr` および/または `partner_addrs` が IP の代わりに DNS 名を使用して構成されている場合、`PySyncObj` はオブジェクトが作成されるときに 1 回だけ効果的に解決を実行していました。同じノードが別の IP でオンラインに戻ると、問題が発生していました。

Stability improvements

- `psycpg2-2.9+` (AlexanderKukushkin) との互換性

`psycpg2` では、`autocommit = True` は `with connection` ブロックで無視され、レプリケーションプロトコル接続が切断されます。

- Zookeeper で過剰な HA ループの実行を修正しました。(AlexanderKukushkin)

メンバー `ZNode` の更新により連鎖反応が発生し、HA ループが複数回連続して実行されました。

- ディスク上で RESTAPI 証明書が変更された場合にリロードします。(MichaelTodorovic)

RESTAPI 証明書ファイルがその場で更新された場合、パトローニはリロードを実行しませんでした。

- `kerberosauth` が使用されている場合、`pgpassdir` を作成しないでください。(KostiantynNemchenko)

Kerberos とパスワード認証は相互に排他的です。

- カスタムブートストラップの小さな問題を修正しました (AlexanderKukushkin)

PITR を実行する場合にのみ `hot_standby=off` で Postgres を起動し、PITR の完了後に再起動します。

Bugfixes

- kazoo-2.7+ (AlexanderKukushkin) との互換性

Patroni は自分で再試行を処理しているため、利用可能な接続がない場合、Zookeeper クラスターへのリクエストはすぐに破棄されるという `kazoo` の古い動作に依存しています。

- プロキシを介して接続していることがわかっている場合、Etcdv3 クラスターのバージョンを明示的に要求します。(AlexanderKukushkin)

Patroni は gPRC-gateway を介して Etcdv3 クラスターを操作しており、クラスターバージョンに応じて異なるエンドポイント (`/v3`、`/v3beta`、または `/v3alpha`) を使用する必要があります。バージョンはクラスタートポロジと一緒にのみ解決されましたが、プロキシを介して接続するときに後者が実行されなかったため。

15.15 バージョン 2.0.2

New features

- 外部で管理されるレプリケーションスロットを無視できるようになりました。(JamesColeman)

Patroni は、不明なレプリケーションスロットを削除しようとしていますが、レプリケーションスロットを外部で管理する必要がある場合は確かにあります。これからは、削除しないスロットを構成できます。

- RESTAPI の暗号スイート制限のサポートを追加しました (Gunnar/"Nick"/Bluth)

`restapi.ciphers` または `PATRONI_RESTAPI_CIPHERS` 環境変数を介して構成できます。

- RESTAPI の暗号化された TLS キーのサポートを追加しました。(JonathanS.Katz)

`restapi.keyfile_password` または `PATRONI_RESTAPI_KEYFILE_PASSWORD` 環境変数を介して構成できます。

- RESTAPI 認証資格情報の常時比較 (AlexBrasetvik)

タイミング攻撃に対して脆弱な `==` の代わりに `hmac.compare_digest()` を使用します。

- レプリケーションラグに基づいて同期ノードを選択します。(KrishnaSarabu)

同期ノードのレプリケーションラグが構成されたしきい値を超え始めた場合、非同期に降格およびまたは他のノードに置き換えられる可能性があります。動作は `maximum_lag_on_syncnode` で制御されます。

Stability improvements

- カスタムブートストラップを行うときに `hot_standby = off` で postgres を起動します。(IgorYanchenko)

カスタムブートストラップ中に、Patroni はベースバックアップを復元し、Postgres を起動し、リカバリが完了するまで待機しています。スタンバイの一部の PostgreSQL パラメーターは、プライマリ

よりも小さくすることはできません。新しい値（WAL から復元）が構成された値よりも高い場合、Postgres はパニックして停止します。このような動作を回避するために、`hot_standby` モードなしでカスタムブートストラップを実行します。

- 必要なウォッチドッグが正常でない場合にユーザーに警告します。(NicolasThauvin)

ウォッチドッグデバイスが必要なモードで書き込みできないか、見つからない場合、メンバーを昇格できません。この構成ミスを検索する場所をユーザーに示す警告を追加しました。

- シングルユーザーモードリカバリの冗長性が向上しました。(AlexanderKukushkin)

Patroni が PostgreSQL が明確にシャットダウンされなかったことに気付いた場合、特定の状況、シングルユーザーモードで Postgres を起動することによりクラッシュリカバリが実行されます。リカバリに失敗した場合（ディスクの空き容量不足など）、エラーが飲み込まれた可能性があります。

- `python-consul2` モジュールとの互換性を追加しました。(AlexanderKukushkin, WilfriedRoset)

古き良き `python-consul` は数年以来維持されていないため、誰かが新機能とバグ修正を含むフォークを作成しました。

- `patronictl` の実行時に `bypass_api_service` を使用しないでください。(AlexanderKukushkin)

K8s ポッドが `default` 以外の名前空間で実行されている場合、必ずしも `kubernetes` エンドポイントを照会するための十分な権限があるとは限りません。この場合、Patroni は警告を表示し、`bypass_api_service` 設定を無視します。`patronictl` の場合、警告は少し面倒でした。

- `raft.data_dir` が存在しない場合は作成するか、書き込み可能であることを確認します。(MarkMercado)

使いやすさと使いやすさを向上。

Bugfixes

- ポーズ中にリーダーロックを失った場合、リスタートを中断したり昇格したりしないでください。(AlexanderKukushkin)

一時停止中、ロックなしで postgres をプライマリとして実行できます。

- RESTAPI の `shutdown_request()` の問題を修正しました (NicolasLimage)

SSL 接続の処理を改善し、スレッドが開始されるまでハンドシェイクを遅らせるために、Patroni は `HTTPServer` のいくつかのメソッドをオーバーライドします。`shutdown_request()` メソッドは忘れられました。

- Zookeeper (AlexanderKukushkin) 使用時の睡眠時間の問題を修正

Patroni が HA コードを実行してから実行するまでに最大 2 倍長くスリープしている可能性があります。

- ブートストラップが失敗した後にデータディレクトリを移動するときの無効な `os.symlink()` 呼び出しを修正しました。(AndrewL'Ecuyer)

ブートストラップが失敗した場合、Patroni はデータディレクトリ、`pg_wal`、およびすべてのテーブルスペースの名前を変更しています。その後、シンボリックリンクを更新して、ファイルシステムの一貫性を維持します。`src` および `dst` 引数が交換されているため、シンボリックリンクの作成に失敗していました。

- `post_bootstrap()` メソッドのバグを修正しました。(AlexanderKukushkin)

スーパーユーザーのパスワードが構成されていない場合、Patroni は `post_init` スクリプトの呼び出しに失敗したため、ブートストラップ全体が失敗していました。

- スタンバイクラスターの `pg_rewind` に関する問題を修正しました (AlexanderKukushkin)

スーパーユーザー名が Postgres と異なる場合、接続文字列にデータベース名が含まれていないため、スタンバイクラスターの `pg_rewind` は失敗していました。

- Etcdv3 による認証が明示的に失敗した場合にのみ終了します。(AlexanderKukushkin)

起動時に、Patroni は ETCD クラスタトポロジの検出を実行し、必要な場合は認証します。Patroni はこのサーバーで認証を実行しようとしていて、次のノードで再試行する代わりに失敗しました。

- `psutil.cmdline()` で空のリストを返すケースを処理しました。(AlexanderKukushkin)

ゾンビプロセスはまだ `postmasters` の子ですが、`cmdline()` はありません

- `PATRONI_KUBERNETES_USE_ENDPOINTS` 環境変数をブール値として扱います。(AlexanderKukushkin)

そうしないと、環境を介して `kubernetes.use_endpoints` を無効にすることが不可能になりました。

- 同時エンドポイント更新エラーの処理を改善しました。(AlexanderKukushkin)

Patroni は、現在のエンドポイントオブジェクトを明示的にクエリし、現在のポッドがまだリーダーロックを保持していることを確認して、更新を繰り返します。

15.16 バージョン 2.0.1

New features

- `less` が利用できない場合、`patronictl edit-config` のページャーとして `more` を使用します。(PavelGolub)

Windows `more.com` です。それに加えて、`requirements.txt` で `cdiff` は `ydifff` に変更されましたが、`patronictl` はまだ互換性のために両方をサポートしています。

- `raft bind_addr` および `password` のサポートを追加しました (AlexanderKukushkin)

`raft.bind_addr` は、NAT の背後で実行する場合に役立つ場合があります。`raft.password` は、トラフィックの暗号化を有効にします (`cryptography` モジュールが必要です)。

- `sslpassword` 接続パラメーターのサポートを追加しました。(KostiantynNemchenko)

接続パラメーターは PostgreSQL13 で導入されました。

Stability improvements

- ポーズ時の動作を変更しました (AlexanderKukushkin)

1. PGDATA ディレクトリが見つからない/空の場合、Patroni は `bootstrap` メソッドを呼び出しません。
2. Patroni は一時停止中に `sysid` の不一致で終了せず、警告のみをログに記録します。

3. Postgres がリカバリーで実行されていない（書き込みを受け入れます）が、sysid が初期化キーと一致しない場合、ノードは一時停止モードでリーダーキーを取得しようとしません。

- クラッシュリカバリー実行時に `master_start_timeout` を適用しました。(AlexanderKukushkin)

Postgres がリーダーノードでクラッシュした場合、Patroni は Postgres をシングルユーザーモードで起動してクラッシュリカバリを行います。クラッシュリカバリ中に、リーダーロックが更新されています。クラッシュリカバリが `master_start_timeout` 秒で終了しなかった場合、Patroni は強制的に停止し、リーダーロックを解放します。

- `urllib3` 要件から `secure` エクストラを削除しました。(AlexanderKukushkin)

そこに追加する唯一の理由は、python2.7 の `ipaddress` 依存関係でした。

Bugfixes

- `Kubernetes.update_leader()` のバグを修正しました (AlexanderKukushkin)

リーダーオブジェクトの更新が失敗したときに、未処理の例外により、プライマリの降格が妨げられていました。

- RAFT 使用時の `patronictl` のハングを修正しました。(AlexanderKukushkin)

`Patroniconfig` で `patronictl` を使用する場合、`self_addr` `partner_addrs` に追加する必要があります。

- `get_guc_value()` のバグを修正 (AlexanderKukushkin)

Patroni は PostgreSQL12 で `restore_command` の値を取得できませんでした。したがって、`pg_rewind` の欠落している WAL をフェッチできませんでした。

15.17 バージョン 2.0.0

このバージョンでは、PostgreSQL13 との互換性が強化され、複数の同期スタンバイのサポートが追加され、`pg_rewind` の処理が大幅に改善され、純粋な RAFT (Etcd、Consul、または Zookeeper なし) での Etcdv3 と Patroni のサポートが追加され、オプションで `pre_promote` (フェンシング) スクリプト。

PostgreSQL 13 support

- PostgreSQL13+で `standby_leader` に昇格するときに `on_reload` を起動しないでください。(AlexanderKukushkin)

`standby_leader` に昇格するときは、`primary_conninfo` を変更し、ロールを更新して Postgres をリロードします。 `on_role_change` と `on_reload` は効果的にお互いを複製するため、Patroni は `on_role_change` のみを呼び出します。

- `gssencmode` および `channel_binding` 接続パラメーターのサポートを追加しました。(AlexanderKukushkin)

PostgreSQL12 では `gssencmode` および `channel_binding` 接続パラメーターが導入され、`postgresql.authentication` セクションで定義されている場合に使用できるようになりました。

- `wal_keep_segments` から `wal_keep_size` への名前変更を処理しました。(AlexanderKukushkin)
構成に誤りがある場合 (13 では `wal_keep_segments`、古いバージョンでは `wal_keep_size`) Patroni は自動的に構成を調整します。
- 可能であれば 13 で `pg_rewrite` と `--restore-target-wal` を使用します。(AlexanderKukushkin)
PostgreSQL13 では、Patroni は `restore_command` が構成されているかどうかを確認し、`pg_rewrite` にそれを使用するように指示します。

New features

- [BETA]PureRAFT に Patroni のサポートを実装しました (AlexanderKukushkin)
これにより、Etcd、Consul、Zookeeper などのサードパーティの依存関係なしで Patroni を実行できます。HA の場合、3 つの Patroni ノード、または Patroni で 2 つのノードと `patroni_raft_controller` で 1 つのノードを実行する必要があります。詳細については、`patroni_raft_controller` を確認してください。
- [BETA]gPRC-gateway を介して Etcdv3 プロトコルのサポートを実装しました (AlexanderKukushkin)
Etcd3.0 は 4 年以上前にリリースされ、Etcd3.4 では v2 がデフォルトで無効になっています。v2 が Etcd から完全に削除される可能性もあるため、Patroni に Etcdv3 のサポートを実装しました。それを使い始めるには、`etcd3` セクションを明示的に作成する必要があります Patroni 構成ファイルです。
- 複数の同期スタンバイのサポート (クリシュナサラブ)
これにより、複数の同期レプリカでクラスターを実行できます。同期レプリカの最大数は、新しいパラメーター `synchronous_node_count` によって制御されます。デフォルトで 1 に設定され、`synchronous_mode` が `off` に設定されている場合は効果がありません。
- `pre_promote` スクリプトを呼び出す可能性を追加しました。(SergeyDudoladov)
コールバックとは異なり、`pre_promote` スクリプトは、リーダーロックを取得した後、ただし Postgres を昇格する前に同期的に呼び出されます。スクリプトが失敗するか、ゼロ以外の終了コードで終了した場合、現在のノードはリーダーロックを解放します。
- 構成ディレクトリのサポートを追加しました。(FlorisvanNee)
ディレクトリ内の YAML ファイルがロードされ、アルファベット順に適用されます。
- PostgreSQL パラメータの高度な検証 (AlexanderKukushkin)
特定のパラメーターが現在の PostgreSQL バージョンでサポートされていない場合、またはその値が正しくない場合、Patroni はパラメーターを完全に削除するか、値の修正を試みます。
- プロモート後の強制チェックポイントが完了したときにメインスレッドを起こします。(AlexanderKukushkin)
レプリカは、DCS のリーダーのメンバーキーを介したチェックポイントの指示を待っています。キーは通常、HA ループごとに 1 回のみ更新されます。メインスレッドを起こさないと、レプリカは必要以上に最大 `loop_wait` 秒待つ必要があります。
- 9.6 以降での `pg_stat_wal_recevier` ビューの使用 (AlexanderKukushkin)
ビューには `primary_conninfo` および `primary_slot_name` の最新の値が含まれていますが、`recovery.conf` の内容は古い可能性があります。

- Patroni 構成ファイルでの IPv6 アドレスの処理を改善しました。(MateuszKowalski)

IPv6 アドレスは角括弧で囲むことになっていますが、Patroni はそれが平易になることを期待していました。現在、両方のフォーマットがサポートされています。

- Consul `service_tags` 構成パラメーターを追加しました。(RobertEdström)

これらは、ロードバランサーなどによる動的サービス検出に役立ちます。

- Zookeeper の SSL サポートを実装しました (KostiantynNemchenko)

kazoo>=2.6.0 が必要です。

- カスタムブートストラップメソッドの `no_params` オプションを実装しました。(KostiantynNemchenko)

`wal-g`、`pgBackRest`、およびその他のバックアップツールをシェルスクリプトにラップせずに呼び出すことができます。

- `init` が失敗した後に WAL とテーブルスペースを移動します。(FeikeSteenbergen)

`reinit` を実行するとき、Patroni はすでに PGDATA だけでなく、シンボリックリンクされた WAL ディレクトリとテーブルスペースも削除していました。これで、`move_data_directory()` メソッドは同様のジョブ、つまり WAL ディレクトリとテーブルスペースの名前変更、PGDATA のシンボリックリンクの更新を行います。

Improved in pg_rewind support

- タイムラインの相違チェックの改善 (AlexanderKukushkin)

レプリカで再生された場所がスイッチポイントよりも進んでいない場合、または元のプライマリのチェックポイントレコードの終わりがスイッチポイントと同じである場合、巻き戻す必要はありません。チェックポイントレコードの最後を取得するには、`pg_waldump` を使用し、その出力を解析します。

- `pg_rewind` がそれについて不平を言う場合、行方不明の WAL を取得してみてください。(AlexanderKukushkin)

`pg_rewind` に必要な WAL セグメントが `pg_wal` ディレクトリに存在しないため、`pg_rewind` が分岐点の前のチェックポイントの場所を見つけられない場合があります。PostgreSQL13 以降、`pg_rewind` は、欠落している WAL をフェッチするために `restore_command` を使用できました。古い PostgreSQL バージョンの場合、Patroni は失敗した巻き戻しの試行のエラーを解析し、独自に `restore_command` を呼び出すことにより、欠落している WAL をフェッチしようとします。

- スタンバイクラスターで新しいタイムラインを検出し、必要に応じて巻き戻し/再初期化をトリガーします。(AlexanderKukushkin)

`standby_cluster` はプライマリクラスターから分離されているため、リーダー選挙とタイムラインスイッチについてすぐには知りません。事実を検出するために、`standby_leader` は定期的に `pg_wal` の新しい履歴ファイルをチェックします。

- 履歴ログ出力の短縮と美化 (AlexanderKukushkin)

Patroni が `pg_rewind` の必要性を把握しようとしているとき、履歴ファイルの内容をプライマリからログに書き込む可能性があります。履歴ファイルは、フェールオーバー/スイッチオーバーのたびに大きくなり、最終的にはあまりにも多くの行を占有し始めます。そのほとんどはあまり役に立ちませ

ん。生データを表示する代わりに、Patroni は現在のレプリカのタイムラインの前 3 行と後 2 行のみを表示します。

Improvements on K8s

- `kubernetes python` モジュールを削除します。(AlexanderKukushkin)

公式の `pythonkubernetes` クライアントには多くの自動生成コードが含まれているため、非常に重いです。Patroni は K8sAPI エンドポイントのごく一部のみを使用しており、それらのサポートを実装するのは難しくありませんでした。

- `kubernetes` サービスをバイパスできるようにします。(AlexanderKukushkin)

K8s で実行している場合、Patroni は通常、`kubernetes` サービスを介して K8sAPI と通信します。そのアドレスは `KUBERNETES_SERVICE_HOST` 環境変数で公開されます。他のサービスと同様に、`kubernetes` サービスは `kube-proxy` によって処理され、構成に応じて、トラフィックルーティングのためにユーザー空間プログラムまたは `iptables` に依存しています。中間コンポーネントをスキップして K8s マスターノードに直接接続すると、より良い再試行戦略を実装し、K8s マスターノードのアップグレード時に Postgres を降格するリスクを軽減できます。

- Patroni クラスターのすべてのポッドの HA ループを同期します (AlexanderKukushkin)

そうしないと、障害検出時間が `t1` から `t1 + loop_wait` に増加していました。

- K8 のサブセットアドレスに `references` と `nodename` を入力します。(AlexanderKukushkin)

一部のロードバランサーはこの情報に依存しています。

- `update_leader()` で発生する可能性のある競合状態を修正しました。(AlexanderKukushkin)

Patroni の外部で発生するリーダー構成マップまたはエンドポイントの同時更新により、`update_leader()` 呼び出しが失敗する場合があります。この場合、Patroni は現在のノードがまだリーダーロックを所有していることを再確認し、更新を繰り返します。

- 存在しない設定へのパッチ適用を明示的に禁止しました。(AlexanderKukushkin)

`kubernetes` 以外の DCS の場合、`cluster.config` が `None` であるため、例外が発生して PATCH 呼び出しが失敗しますが、Kubernetes では、喜んで構成アノテーションを作成し、ブートストラップが終了した後にブートストラップ構成の書き込みを防止していました。

- `pause` のバグを修正 (AlexanderKukushkin)

リーダーキーが存在しない場合、レプリカは `primary_conninfo` を削除して Postgres を再起動していましたが、何もしないはずでした。

Improvements in REST API

- ワーカースレッドが開始されるまで TLS ハンドシェイクを延期しました。(AlexanderKukushkin, BenHarris)

TLS ハンドシェイクが API スレッドで行われ、クライアント側がデータを送信しなかった場合、API スレッドはブロックされました (DoS のリスクがあります)。

- RESTAPI のクライアント証明書とは別に `basic-auth` を確認します。(AlexanderKukushkin)

以前は、クライアント証明書のみが検証されました。2 つのチェックを個別に行うことは、絶対に有効なユースケースです。

- OPTIONS リクエストの HTTP ヘッダーの後に double CRLF を書き込みました。(SergeyBurladyan)

HAProxy は単一の CRLF に満足していましたが、Consul ヘルスチェックは接続の切断と予期しない EOF について不平を言いました。

- GET /cluster は、Zookeeper (AlexanderKukushkin) の古いメンバー情報を表示していました

エンドポイントは Patroni 内部クラスタービューを使用していました。Patroni 自分自身にとっては問題は発生しませんでした、外部の世界にさらされた場合、最新の情報、特にレプリケーションラグを表示する必要があります。

- スタンバイクラスターのヘルスチェックを修正しました (AlexanderKukushkin)

マスターの GET /standby-leader と standby_leader の GET /master が 200 で誤って応答していました。

- DELETE /switchover を実装しました (AlexanderKukushkin)

RESTAPI 呼び出しは、スケジュールされたスイッチオーバーを削除します。

- /readiness および /liveness エンドポイントを作成しました。(AlexanderKukushkin)

K8s サービスがラベルセクターで使用されている場合、サブセットアドレスから「異常な」ポッドを排除するのに役立つ可能性があります。

- GET /replica および GET /async RESTAPI ヘルスチェックの強化 (KrishnaSarabu, AlexanderKukushkin)

チェックはオプションのキーワード ?lag=<max-lag> をサポートするようになり、ラグが指定された値よりも小さい場合にのみ 200 で応答します。この機能に依存している場合、リーダー上の WAL 位置に関する情報は loop_wait 秒ごとにのみ更新されることに注意してください。

- RESTAPI レスポンスにユーザー定義の HTTP ヘッダーのサポートを追加しました。(YogeshSharma)

この機能は、ブラウザから要求が行われる場合に役立つ場合があります。

Improvements in patronictl

- patronictl pause に存在しないリーダーを呼び出そうとしないでください。(AlexanderKukushkin)

K8s でリーダーのないクラスターを一時停止しているときに、patronictl は、メンバー/"None/"にアクセスできないという警告を表示していました。

- メンバー conn_url が欠落している場合を処理します。(AlexanderKukushkin)

K8s では、Patroni がまだ実行されていないため、ポッドに必要なアノテーションがない可能性があります。patronictl を失敗させていました。

- ASCII クラスタートポロジを印刷する機能を追加しました。(MaximFedotov, AlexanderKukushkin)

カスケードレプリケーションを使用してクラスターの概要を取得すると非常に役立ちます。

- patronictl flush switchover を実装する (AlexanderKukushkin)

それ以前は、patronictl flush はスケジュールされた再起動のキャンセルのみをサポートしていました。

Bugfixes

- 既存の PGDATA を使用したクラスターのブートストラップ中の属性エラー (KrishnaSarabu)
/history キーを作成/更新しようとする、パトロニは DCS でまだ作成されていない ClusterConfig オブジェクトにアクセスしていました。
- Consul での例外処理の改善 (AlexanderKukushkin)
touch_member() メソッドの未処理の例外により、Patroni プロセス全体がクラッシュしました。
- post_init スクリプトに synchronous_commit=local を適用しました。 (AlexanderKukushkin)
Patroni は、ユーザー (replication、rewind) を作成するときに既にそれを行っていましたが、post_init の場合にそれを見逃したのは見落としでした。その結果、スクリプトが自分で内部的に実行していない場合、synchronous_mode のブートストラップは終了できませんでした。
- Consul プールマネージャー (ponvenkates) の maxsize を増やしました
デフォルトの size=1 では、いくつかの警告が生成されました。
- Patroni は Postgres を実行中として誤って報告していました (AlexanderKukushkin)
たとえば、ディスク不足エラーにより Postgres がクラッシュした場合、状態は更新されませんでした。
- 欠損値または空の値の代わりに * を pgpass に入れました。 (AlexanderKukushkin)
たとえば、standby_cluster.port が指定されていない場合、pgpass ファイルが誤って生成されました。
- 特殊キャラクターを使用してリーダーノードでの物理レプリケーションスロットの作成をスキップします (クリシュナサラブ)
Patroni は、名前に「-」などの特殊文字 (たとえば、「abc-us-1/」) が含まれている場合、リーダーノードの休止スロット (slots が定義されている場合) を作成しているように見えました。
- カスタムブートストラップで存在しない pg_hba.conf を削除しないようにしました。 (KrishnaSarabu)
カスタムブートストラップ後に pg_hba.conf がたまたま pgdata dir の外部にある場合、Patroni は失敗していました。

15.18 バージョン 1.6.5

New features

- マスターストップタイムアウト (クリシュナサラブ)
Postgres を停止するときに Patroni が待機できる秒数。synchronous_mode が有効な場合にのみ有効です。0 より大きい値に設定され、synchronous_mode が有効になっている場合、master_stop_timeout で設定された値を超える値で停止操作が実行されている場合、Patroni は SIGKILL をポストマスターに送信します。耐久性/可用性のトレードオフに従って値を設定します。パラメーターが設定されていないか、非正の値に設定されている場合、master_stop_timeout は効果がありません。
- プライマリの名前で永続的な物理スロットを作成しないでください (AlexanderKukushkin)
レプリカがダウンしているときにプライマリが WAL セグメントをリサイクルするのは一般的な問題です。これで、固定数のノードと名前が変更されない静的クラスターの優れたソリューションがで

きました。slots のすべてのノードの名前をリストするだけでよいので、ノードがダウンしている (DCS に登録されていない) ときにプライマリがスロットを削除しません。

- ConfigValidator の最初のドラフト (IgorYanchenko)

Patroni 構成を検証するには、`patroni --validate-config patroni.yaml` を使用します。

- タイムライン履歴の最大長を構成する可能性 (KrishnaSarabu)

Patroni は、フェールオーバー/スイッチオーバーの履歴を DCS の `/history` キーに書き込みます。時間の経過とともにこのキーのサイズは大きくなりますが、ほとんどの場合、興味深いのは最後の数行だけです。max_timelines_history パラメーターを使用すると、DCS に保持するタイムライン履歴アイテムの最大数を指定できます。

- Kazoo2.7.0 互換性 (DanyalProut)

Kazoo の一部の非公開メソッドは署名を変更しましたが、Patroni はそれらに依存していました。

Improvements in patronictl

- メンバーのタグを表示する (KostiantynNemchenko, AlexanderKukushkin)

タグはすべてのノードに対して個別に構成されており、それらの概要を取得する簡単な方法はありませんでした

- メンバーのアウトプットを改善する (AlexanderKukushkin)

冗長なクラスター名は、すべての行に表示されなくなり、テーブルヘッダーにのみ表示されます。

```
$ patronictl list
+ Cluster: batman (6813309862653668387) +-----+-----+-----+-----+
↪---+
| Member | Host | Role | State | TL | Lag in MB | Tags |
|         |      |      |      |    |           |      |
+-----+-----+-----+-----+-----+
↪---+
| postgresql0 | 127.0.0.1:5432 | Leader | running | 3 | | clonefrom: true | |
|         |      |      |      |    |           |      |
|         |      |      |      |    |           |      | noloadbalance: true |
↪true |
|         |      |      |      |    |           |      | nosync: true |
|         |      |      |      |    |           |      |
+-----+-----+-----+-----+-----+
↪---+
| postgresql1 | 127.0.0.1:5433 |      | running | 3 | 0.0 |
|         |      |      |      |    |           |      |
+-----+-----+-----+-----+-----+
↪---+
```

- 設定ファイルが明示的に指定されたが見つからない場合に失敗します。 (KaarelMoppel)

以前は、`patronictl` は DEBUG メッセージのみを報告していました。

- 初期化されていない K8s ポッドがパトロールを破損する問題を解決しました (AlexanderKukushkin)

Patroni は、K8 の特定のポッドアノテーションに依存しています。Patroni ポッドの 1 つが停止または開始するとき、有効なアノテーションはまだなく、`patronictl` は例外で失敗していました。

Stability improvements

- K8sAPI サーバーへの LIST 呼び出しが失敗した場合、1 秒のバックオフを適用します。(AlexanderKukushkin)

ほとんどの場合、ログのフラッディングを回避する必要がありますが、メインスレッドの枯渇を防ぐのにも役立ちます。

- `retry-after` HTTP ヘッダーが K8sAPI によって返された場合に再試行します。(AlexanderKukushkin)

K8sAPI サーバーがリクエストで圧倒されている場合、再試行を求める場合があります。

- `postmaster` から `KUBERNETES_` 環境をスクラブしました。(FeikeSteenbergen)

`KUBERNETES_` 環境変数は PostgreSQL には必要ありませんが、`postmaster` に公開すると、バックエンドと通常のデータベースユーザー (`pl/perl` などを使用) にも公開されます。

- 再初期化時にテーブルスペースをクリーンアップします。(KrishnaSarabu)

再初期化中に、Patroni は PGDATA のみを削除し、ユーザー定義のテーブルスペースディレクトリを残しました。これにより、Patroni が `reinit` でループします。この問題の以前の作業は、PGDATA スクリプトを実装することでした。

- プロモートが発生した後に `CHECKPOINT` を明示的に実行しました。(AlexanderKukushkin)

新しいプライマリが `pg_rewind` で使用可能になるまでの時間を短縮するのに役立ちます。

- Etcd メンバーのスマートリフレッシュ (AlexanderKukushkin)

Patroni が Etcd クラスターのすべてのメンバーでリクエストの実行に失敗した場合、Patroni は次回再試行する前に、IP/ホストの変更について A または SRV レコードを再チェックします。

- `pg_controldata` から欠損値をスキップしました。(FeikeSteenbergen)

PGDATA と一致しないバージョンのバイナリを使用しようとすると、値が欠落します。Patroni はとにかく Postgres を起動しようとしますが、Postgres はメジャーバージョンが一致しないと不平を言い、エラーで中止します。

Bugfixes

- 必要に応じて Consul の SSL 検証を無効にします。(JulienRiou)

`urllib3` の特定のバージョン以降、SSL 検証を効果的に無効にするには、`cert_reqs` を明示的に `ssl.CERT_NONE` に設定する必要があります。

- HA ループのすべてのサイクルでレプリケーション接続を開かないようにしました。(AlexanderKukushkin)

回帰は 1.6.4 で導入されました。

- プライマリに障害が発生したときに `on_role_change` コールバックを呼び出します。(AlexanderKukushkin)

場合によっては、仮想 IP が古いプライマリに接続されたままになる可能性があります。回帰は 1.4.5 で導入されました。

- `pg_rewind` が成功した後に `postgres` が起動した場合、リワインド状態をリセットします。(AlexanderKukushkin)

このバグの結果、Patroni は起動して `postgres` を一時停止モードで手動でシャットダウンしていました。

- `recovery.conf` をチェックするときに `recovery_min_apply_delay` を `ms` に変換します

`recovery_min_apply_delay` が 12 より古い PostgreSQL で構成されている場合、Patroni はレプリカを無期限にリスタートしていました。

- PyInstaller の互換性 (AlexanderKukushkin)

PyInstaller は、Python アプリケーションをスタンドアロンの実行可能ファイルにフリーズ (パッケージ) します。 `multiprocessing` の `fork` ではなく `spawn` メソッドに切り替えたとき、互換性が壊れました。

15.19 バージョン 1.6.4

New features

- `patronictl reinit` の `--wait` オプションを実装しました。(IgorYanchenko)

`--wait` オプションが使用されている場合、Patronictl は `reinit` が終了するのを待ちます。

- Windows サポートのさらなる改善 (IgorYanchenko, AlexanderKukushkin)

1. 統合テストに使用されるすべてのシェルスクリプトは `python` で書き換えられます
2. `pg_ctl kill` は、非 `posix` システムで `postgres` を停止するために使用されます
3. `unix-domain` ソケットを使用しようとししないでください

Stability improvements

- `unix_socket_directories` と `stats_temp_directory` が存在することを確認しました。(IgorYanchenko)

Patroni と `Postgres` の起動時に、`unix_socket_directories` と `stats_temp_directory` が存在することを確認するか、作成しようとします。作成に失敗した場合、Patroni は終了します。

- `postgresql.pgpass` が Patroni に書き込みアクセスがある場所にあることを確認します。(IgorYanchenko)

書き込みアクセスがない場合、Patroni は例外で終了します。

- Consul `serfHealth` チェックをデフォルトで無効にします。(KostiantynNemchenko)

ネットワークの問題がほとんどない場合でも、`serfHealth` が失敗すると、ノードに関連付けられているすべてのセッションが無効になります。したがって、リーダーキーは `ttl` よりもはるかに早く失われ、レプリカの不要な再起動と、場合によってはプライマリの降格が発生します。

- K8sAPI への接続用に TCP キープアライブを構成します。(AlexanderKukushkin)

TTL 秒後にソケットから何も得られない場合、デッドと見なすことができます。

- ユーザー作成時にパスワードのログ記録を回避しました。(AlexanderKukushkin)

パスワードが拒否された場合、またはロギングが詳細に構成されているか、まったく構成されていない場合、パスワードが postgres ログに書き込まれる可能性があります。それを避けるために、Patroni は `log_statement`、`log_min_duration_statement`、および `log_min_error_statement` をいくつかの安全な値に変更しますユーザーを作成/更新しようとする前に。

Bugfixes

- カスケードレプリカで `standby_cluster` 構成から `restore_command` を使用します。(AlexanderKukushkin)

`standby_leader` は、機能が存在する最初からすでにそれを行っていました。レプリカで同じことをしないと、スタンバイリーダーに追いつくことができない場合があります。

- スタンバイクラスターによって報告された更新タイムライン (AlexanderKukushkin)

タイムラインスイッチの場合、スタンバイクラスターはプライマリから正しく複製していましたが、`patronictl` は古いタイムラインを報告していました。

- `custom_conf` で特定の回復パラメーターを定義できるようになりました。(AlexanderKukushkin)

レプリカでリカバリパラメーターの検証を行うとき、Patroni は `archive_cleanup_command`、`promote_trigger_file`、`recovery_end_command`、`recovery_min_apply_delay`、および `restore_command` パトロニ構成で定義されておらず、`postgresql.auto.conf` または `postgresql.conf` 以外のファイルで定義されている場合、スキップします。

- 名前にピリオドを含む postgresql パラメーターの処理を改善しました。(AlexanderKukushkin)

このようなパラメーターは、ユニットが必ずしも文字列ではない拡張機能によって定義できます。値を変更すると、再起動が必要になる場合があります (`pg_stat_statements.max` など)。

- シャットダウン中の例外処理を改善しました。(AlexanderKukushkin)

シャットダウン中、Patroni は DCS のステータスを更新しようとしています。DCS にアクセスできない場合、例外が発生する可能性があります。例外処理の欠如により、ログスレッドが停止していませんでした。

15.20 バージョン 1.6.3

Bugfixes

- `pg_rewind` の実行時にパスワードを公開しないでください。(AlexanderKukushkin)

#1301 でバグが導入されました

- `postgresql.authentication` で指定された接続パラメーターを `pg_basebackup` に適用し、カスタムレプリカ作成方法を適用しました。(AlexanderKukushkin)

彼らは url のような接続文字列に依存していたため、パラメーターは適用されませんでした。

15.21 バージョン 1.6.2

New features

- `patroni --version` を実装しました (IgorYanchenko)

Patroni の現在のバージョンを印刷して終了します。

- すべての http リクエストに `user-agent` http ヘッダーを設定します。 (AlexanderKukushkin)

Patroni は、http プロトコルを介して Consul、Etcd、および KubernetesAPI と通信しています。特別に細工された `user-agent` (例: `Patroni/1.6.2 Python/3.6.8 Linux`) を持つと、デバッグとモニタリングに役立つ場合があります。

- 例外トレースバックのログレベルを設定できるようにしました。 (IgorYanchenko)

`log.traceback_level=DEBUG` を設定すると、`log.level=DEBUG` の場合にのみトレースバックが表示されます。デフォルトの動作は同じです。

Stability improvements

- 構成ファイルに必要なモジュールを検索するときに、すべての DCS モジュールをインポートしないようにしました。 (AlexanderKukushkin)

例のみが必要な場合、Etcd、Consul、および Kubernetes のモジュールをインポートする必要はありません。飼育係。メモリ使用量を削減し、INFO メッセージ `Failed to import smth` の問題を解決します。

- 明示的な要件から `python requests` モジュールを削除しました。 (AlexanderKukushkin)

重要なことには使用されませんでしたが、新しいバージョンの `urllib3` がリリースされると多くの問題が発生します。

- YAML 配列ではなくコンマ区切りの文字列として記述された `etcd.hosts` の処理を改善しました。 (IgorYanchenko)

以前は、`host1:port1, host2:port2` 形式 (コンマの後のスペース文字) で書き込むと失敗していました。

Usability improvements

- `patronictl` の空のリストからメンバーを選択することをユーザに強制しないようにしました。 (IgorYanchenko)

ユーザーが間違ったクラスター名を指定した場合、空のリストからメンバーを選択するように求めるのではなく、例外が発生します。

- RESTAPI がバインドできない場合のエラーメッセージをより役立つようにしました。 (IgorYanchenko)

経験の浅いユーザーにとっては、Python スタックトレースから何が問題なのかを判断するのは難しいかもしれません。

Bugfixes

- `wal_buffers` の計算を修正しました (AlexanderKukushkin)

PostgreSQL11 では、基本単位が 8kB ブロックからバイトに変更されました。

- PostgreSQL10+でのみ `primary_conninfo` の `passfile` を使用します。(AlexanderKukushkin)

古いバージョンでは、最新バージョンの `libpq` がインストールされていない限り、`passfile` が動作する保証はありません。

15.22 バージョン 1.6.1

New features

- `PATRONICTL_CONFIG_FILE` 環境変数を追加しました (msvechla)

環境から `patronictl` の `--config-file` 引数を構成できます。

- `patronictl history` を実装する (AlexanderKukushkin)

フェールオーバー/スイッチオーバーの履歴を表示します。

- `pg_rewind` を実行するときに `PGOPTIONS` に `-c statement_timeout=0` を渡します。(AlexanderKukushkin)

サーバー上の `statement_timeout` が小さな値に設定され、`pg_rewind` によって実行されるステートメントの1つがキャンセルされた場合から保護します。

- PostgreSQL 構成のより低い値を許可します。(Soulou)

Patroni は、一部の PostgreSQL 構成パラメーターを一部のハードコードされた値よりも小さく設定することを許可しませんでした。最小許容値が小さくなりました。デフォルト値は変更されていません。

- 証明書ベースの認証を許可します。(JonathanS.Katz)

この機能により、スーパーユーザー、レプリケーション、巻き戻しアカウントの証明書ベースの認証が有効になり、ユーザーは接続する `sslmode` を指定できます。

- パスワードの代わりに `primary_conninfo` の `passfile` を使用します。(AlexanderKukushkin)

`postgresql.conf` に `600` 権限を設定するのを避けることができます

- 構成の変更に関係なく `pg_ctl reload` を実行します。(AlexanderKukushkin)

一部の構成ファイルが Patroni によって制御されていない可能性があります。誰かが RESTAPI を介して、または `SIGHUP` を Patroni プロセスに送信することによりリロードを行っている場合、通常は Postgres もリロードされることが期待されます。以前は、Patroni 構成の `postgresql` セクションに変更がない場合は発生しませんでした。

- `primary_conninfo` だけでなく、すべての回復パラメーターを比較します (AlexanderKukushkin)

以前は、`check_recovery_conf()` メソッドは `primary_conninfo` が変更されたかどうかをチェックするだけで、他のすべての回復パラメーターは考慮されませんでした。

- 再起動せずにいくつかの回復パラメーターを適用できるようにします。(AlexanderKukushkin)

PostgreSQL12 以降、次の回復パラメータを再起動せずに変更できました: `archive_cleanup_command`、`promote_trigger_file`、`recovery_end_command`、および `recovery_min_apply_delay`。将来の Postgres リリースでは、このリストは拡張され、Patroni は自動的にサポートします。

- `use_slots` をオンラインで変更できるようにしました。(AlexanderKukushkin)

以前は、Patroni を再起動し、スロットを手動で削除する必要がありました。

- Postgres の起動時に `PATRONI_` プレフィックス付きの環境変数のみを削除します (CodyCoons)
- さまざまな ForeignDataWrappers の実行に関する多くの問題を解決します。

Stability improvements

- K8sAPI で作業するときに `LIST+WATCH` を使用してください (AlexanderKukushkin)

オブジェクトの変更 (ポッド、エンドポイント/構成マップ) を効率的に受信でき、K8s マスターノードのストレスが軽減されます。

- ブートストラップ中に PGDATA が空でない場合のワークフローを改善しました。(AlexanderKukushkin)
- `initdb` ソースコードによると、`lost+found` と `.dotfiles` のみがある場合、PGDATA が空であると考えられる場合があります。現在、パトローニは同じことをしています。PGDATA がたまたま空ではなく、同時に `pg_controldata` の観点から有効でない場合、Patroni は文句を言って終了します。

- すべての HA ループで高価な `os.listdir()` を呼び出さないようにしました。(AlexanderKukushkin)
- システムに IO ストレスがかかっている場合、`os.listdir()` の実行には数秒 (または場合によっては数分) かかり、Patroni の HA ループに悪影響を与える可能性があります。これにより、更新がないためにリーダーキーが DCS から消えることさえあります。PGDATA が空でないことを確認するより良い、より安価な方法があります。次に、PGDATA 内の `global/pg_control` ファイルの存在を確認します。

- ログインフラストラクチャのいくつかの改善 (AlexanderKukushkin)

以前は、ロギングスレッドが `daemon` スレッドだったため、シャットダウン時にログの最後の数行が失われる可能性がありました。

- python3.4 以降で `spawn` マルチプロセッシングスタートメソッドを使用します。(MaciejKowalczyk)
- スレッド化とマルチプロセッシングがうまく混ざらないことは、Python で知られている [issue](#) です。デフォルトの方法 `fork` から `spawn` への切り替えは、推奨される回避策です。そうしないと、Postgres が実際に起動して実行されているときに、Postmaster の開始プロセスがハングし、Patroni が無期限に `INFO: restarting after failure in progress` を報告する可能性があります。

Improvements in REST API

- RESTAPI でクライアント証明書を確認できるようにしました。(AlexanderKukushkin)

`verify_client` が `required` に設定されている場合、Patroni はすべての RESTAPI 呼び出しのクライアント証明書を確認します。 `optional` に設定されている場合、クライアント証明書は、安全でないすべての RESTAPI エンドポイントについてチェックされます。

- Postgres が実行されていない場合、`GET /replica` ヘルスチェック要求の応答コード 503 を返します。(AlexanderAnikin)

Postgres は、クライアント接続の受け入れを開始する前に、リカバリにかなりの時間を費やす場合があります。

- `/history` および `/cluster` エンドポイントを実装しました。(AlexanderKukushkin)

`/history` エンドポイントは、DCS の `history` キーの内容を示します。 `/cluster` エンドポイントは、すべてのクラスターメンバーと、保留中およびスケジュールされた再起動またはスイッチオーバーなどのサービス情報を表示します。

Improvements in Etcd support

- `RetryonEtcdRAFTInternalerror` (AlexanderKukushkin)

Etcd ノードがシャットダウンされると、 `response code=300, data='etcdserver: server stopped'` を送信します、これにより、Patroni はプライマリを降格させました。

- Etcd 要求の再試行をあまりにも早くあきらめないでください。(AlexanderKukushkin)

ネットワークに問題が発生したとき、Patroni は Etcd ノードのリストをすぐに使い果たし、 `retry_timeout` 全体を使用せずにあきらめていました。

Bugfixes

- `pg_rewind` ユーザーに実行権限を付与するときに `synchronous_commit` を無効にします。(kremius)

ブートストラップが `synchronous_mode_strict: true` で完了すると、非同期ノードが利用可能になったため、`'GRANTEXECUTE'` ステートメントは無期限に待機していました。

- python3.7 のメモリリークを修正しました。(AlexanderKukushkin)

Patroni は `ThreadingMixIn` を使用して RESTAPI リクエストを処理しており、python3.7 では、デフォルトでデーモン以外のすべてのリクエストに対してスレッドが生成されました。

- 非同期アクションの競合状態を修正しました。(AlexanderKukushkin)

停止した Postgres を回復しようとする試みによって `patronictl reinit --force` が上書きされる可能性があります。これは、ベースバックアップの実行中に Patroni が Postgres を起動しようとしている状況になりました。

- `postmaster_start_time()` メソッドの競合状態を修正しました。(AlexanderKukushkin)

メソッドが RESTAPI スレッドから実行される場合、別のカーソルオブジェクトを作成する必要があります。

- 名前に大文字が含まれる同期スタンバイが昇格しない問題を修正 (AlexanderKukushkin)

`application_name` を `synchronous_standby_names` の値と比較しながら Postgres が同じことをしていたため、名前を小文字に変換しました。

- 新しいものを開始する前に、コールバックプロセスと一緒にすべての子供たちを殺します。(AlexanderKukushkin)

そうしないと、`bash` でコールバックを実装するのが難しくなり、最終的に 2 つのコールバックが同時に実行されている状況につながります。

- 「スタートに失敗しました」問題を修正しました (AlexanderKukushkin)

特定の条件下では、Postgres が起動して実行されているにもかかわらず、Postgres の状態が「`startfailed`」に設定される場合があります。

15.23 バージョン 1.6.0

このバージョンでは、PostgreSQL12 との互換性が追加され、PostgreSQL11 以降でスーパーユーザーなしで `pg_rewind` を実行できるようになり、IPv6 サポートが有効になります。

New features

- Psycopg2 は要件から削除され、個別にインストールする必要があります。(AlexanderKukushkin)

2.8.0 以降 `psycopg2` は 2 つの異なるパッケージ `psycopg2` および `psycopg2-binary` に分割され、ファイルシステム上の同じ場所に同時にインストールできました。依存関係地獄の問題を軽減するために、ユーザーにインストール方法を選択させます。利用可能なオプションがいくつかあります。`psycopg2` を参照してください。
- PostgreSQL12 (AlexanderKukushkin) との互換性

PostgreSQL12 以降、`recovery.conf` はなくなり、以前のすべての回復パラメーターは `recovery.conf` に変換されます。ALTER SYSTEM SET `primary_conninfo` などから保護するために、Patroni は `postgresql.auto.conf` を解析し、そこからすべてのスタンバイおよびリカバリパラメーターを削除します。Patroniconfig は引き続き下位互換性があります。たとえば、`restore_command` が GUC であるにもかかわらず、`postgresql.recovery.conf.restore_command` セクションで指定できます。パトロニは PostgreSQL12 の `postgresql.conf` にそれを書き込みます。
- PostgreSQL11 以降でスーパーユーザーなしで `pg_rewind` を使用できるようにしました。(AlexanderKukushkin)

この機能を使用する場合は、Patroni 構成ファイルの `postgresql.authentication.rewind` セクションで `username` および `password` を定義してください。既存のクラスターの場合、ユーザーを手動で作成し、いくつかの機能に対する GRANT EXECUTE 権限を付与する必要があります。詳細については、PostgreSQL `username _` を参照してください。
- レプリカで実際の `primary_conninfo` 値と目的の `primary_conninfo` 値をスマートに比較します。(AlexanderKukushkin)

既存のプライマリ-スタンバイクラスターを Patroni が管理するクラスターに変換する場合、レプリカの再起動を回避すると役立つ場合があります
- IPv6 サポート (AlexanderKukushkin)

2 つの大きな問題がありました。PatroniRESTAPI サービスは `0.0.0.0` のみリスンしており、`api_url` および `conn_url` で使用される IPv6IP アドレスが適切に引用されていませんでした。
- Kerberos サポート (AjithVilas, AlexanderKukushkin)

Patroni 構成ファイルでパスワードを定義する代わりに、Postgres ノード間で Kerberos 認証を使用できるようにします
- Manage `pg_ident.conf` (AlexanderKukushkin)

この機能は `pg_hba.conf` と同様に機能します: `postgresql.pg_ident` が構成ファイルまたは DCS で定義されている場合、Patroni はその値を `pg_ident.conf` に書き込みますが、`postgresql.parameters.ident_file` が定義されている場合、Patroni は `pg_ident` が外部から管理され、ファイルを更新しないと想定します。

Improvements in REST API

- `/health` エンドポイントを追加しました。(WilfriedRoset)

PostgreSQL が実行されている場合にのみ HTTP ステータスコードを返します

- `/read-only` および `/read-write` エンドポイントを追加しました。(JulienRiou)

`/read-only` エンドポイントは、レプリカとプライマリ全体でバランスの取れた読み取りを有効にします。`/read-write` エンドポイントは、`/primary`、`/leader`、および `/master` のエイリアスです。

- `SSLContext` を使用して RESTAPI ソケットをラップします。(JulienRiou)

`ssl.wrap_socket()` の使用は非推奨であり、TLS1.1 のような間もなく非推奨になるプロトコルをまだ許可していました。

Logging improvements

- 二段階ロギング (AlexanderKukushkin)

すべてのログメッセージは、最初にメモリ内キューに書き込まれ、後で非同期に別のスレッドから `stderr` またはファイルにフラッシュされます。最大キューサイズは制限されています（設定可能）。制限に達すると、Patroni はログを失い始めますが、これは HA ループをブロックするよりも優れています。

- `GET/OPTIONSAPI` 呼び出しとレイテンシーのデバッグログを有効にします。(JanTomsa)

HAProxy、Consul、またはどのノードがプライマリ/レプリカであるかを決定するその他のツールによって実行されるヘルスチェックのデバッグに役立ちます。

- `Retry` でキャッチされた例外をログに記録します。(DanielKucera)

試行回数またはタイムアウトに達した場合、最後の例外をログに記録します。DCS への通信が失敗したときの問題のデバッグに役立つことを願っています。

Improvements in patronictl

- スケジュールされたスイッチオーバーとリスタートのダイアログを強化しました (RafiaSabih)

以前のダイアログでは、スケジュールされたアクションが考慮されなかったため、誤解を招きました。

- 設定ファイルが存在するかどうかを確認します。(WilfriedRoset)

指定されたファイル名が存在しない場合、静かに無視するのではなく（誤解につながる可能性があります）、構成ファイルについて冗長になります。

- `EDITOR` のフォールバック値を追加しました。(WilfriedRoset)

`EDITOR` 環境変数が定義されていない場合、`patronictl edit-config` は `PatroniCtlException` で失敗していました。新しい戦略は、`editor` を試すことです `vi`、ほとんどのシステムで利用できるはずです。

Improvements in Consul support

- Consul の一貫性モードを指定できるようになりました。(JanTomsa)

一貫性モード [here](#) _の詳細を読むことができます。

- SIGHUP で Consul の設定をリロードします。(CameronDanielKucera、AlexanderKukushkin)

誰かが token の値を変更している場合に特に役立ちます。

Bugfixes

- スイッチオーバー/フェイルオーバーのコーナーケースを修正しました (SharoonThomas)

RESTAPI にアクセスできず、フォールバックとして DCS を使用している場合、変数 `scheduled_at` は未定義になる可能性があります。

- カスタムブートストラップ中に `pg_hba.conf` の localhost への信頼をオープンしました。(AlexanderKukushkin)

以前は `unix_socket` に対してのみ開いていたため、多くのエラーが発生していました: `FATAL: no pg_hba.conf entry for replication connection from host /"127.0.0.1/", user /"replicator/"`

- 前のリーダーが先行している場合でも、同期ノードを正常と見なします。(AlexanderKukushkin)

プライマリが DCS へのアクセスを失った場合、Postgres を読み取り専用で再起動しますが、他のノードが RESTAPI を介して古いプライマリに引き続きアクセスできる場合があります。古いプライマリが同期スタンバイよりも前に WAL 位置を報告していたため、このような状況により、同期スタンバイがプロモートしませんでした。

- スタンバイクラスターのバグ修正 (AlexanderKukushkin)

`standby_leader` にアクセスできない場合にスタンバイクラスターでレプリカをブートストラップできるようにし、その他のマイナーな修正を行いました。

15.24 バージョン 1.5.6

New features

- プロキシのセットを介した etcd クラスターとのサポート作業 (AlexanderKukushkin)

etcd クラスターに直接アクセスできず、一連のプロキシを介してアクセスできる場合があります。この場合、Patroni は etcd トポロジー検出を実行せず、プロキシホストを介したラウンドロビンのみを実行します。動作は ``etcd.use_proxies`` によって制御されます。

- ノード上のロールが変更されたときのコールバックの動作を変更しました。(AlexanderKukushkin)

ロールが ``master`` または ``standby_leader`` から ``replica`` に、または ``replica`` から ``standby_leader`` に変更された場合、``on_restart`` コールバックは、``on_role_change`` コールバックを優先して呼び出されなくなります。

- postgres の起動方法を変更します (AlexanderKukushkin)

自分自身を実行する代わりに ``multiprocessing.Process`` を使用し、``multiprocessing.Pipe`` を使用して、ポストマスター pid を Patroni プロセスに送信します。それ以前はパイプを使用していたですが、stdin を閉じた状態で postmaster プロセスを終了していました。

Bug fixes

- スタンバイリーダーの RESTAPI によって返されるロールを修正しました。(AlexanderKukushkin)

・`standby\_leader`の代わりに`replica`を誤って返していました

- キルできなかった場合はコールバックの終了を待ちます。(JulienTachaires)

Patroni には、新しいコールバックをキャンセルしていた`sudo`で実行されているコールバックスクリプトを終了するための十分な特権がありません。実行中のスクリプトを強制終了できなかった場合、Patroni は終了するまで待機してから、次のコールバックを実行します。

- `dcs.get_cluster` メソッドにかかるロック時間を削減しました。(AlexanderKukushkin)

ロックが保持されているため、DCS の速度が RESTAPI ヘルスチェックに影響し、誤検知が発生していました。

- `pg_wal/pg_xlog` がシンボリックリンクの場合の PGDATA のクリーニングを改善しました。(JulienTachaires)

この場合、Patroni はターゲットディレクトリからファイルを明示的に削除します。

- `os.path.relpath` の不必要な使用を削除しました。(AntsAasma)

作業ディレクトリを解決できるかどうか依存しますが、後でファイルシステムからリンク解除されるディレクトリで Patroni が起動された場合、何が失敗しますか。

- Etcd と通信するときに ssl バージョンを強制しないでください。(AlexanderKukushkin)

いくつかの不明な理由のため、debian および ubuntu の python3-etcd はパッケージの最新バージョンに基づいていないため、Etcdv3 でサポートされていない TLSv1 が強制されます。Patroni 側でこの問題を解決しました。

15.25 バージョン 1.5.5

このバージョンでは、以前のマスターの自動再初期化の可能性が導入され、`patanictl` リストの出力が改善され、多数のバグが修正されました。

New features

- `PATRONI_ETCD_PROTOCOL`、`PATRONI_ETCD_USERNAME` および `PATRONI_ETCD_PASSWORD` 環境変数のサポートを追加しました。(ÉtienneM)

構成ファイルでのみ、または `PATRONI_ETCD_URL` の一部としてのみ構成することが可能になる前は、必ずしも便利ではありません。

- 前のマスターを自動的に再初期化できるようにします。(AlexanderKukushkin)

`pg_rewind` が無効になっているか使用できない場合、タイムラインの相違が原因で、以前のマスターが新しいレプリカとして起動できない可能性があります。この場合、それを修正する唯一の方法は、データディレクトリを消去して再初期化することです。この動作は、`postgresql.remove_data_directory_on_diverged_timelines` を設定することにより変更できます。設定されると、Patroni はデータディレクトリを消去し、元のマスターを自動的に再初期化します。

- パトロンリストにタイムラインに関する情報を表示します。(AlexanderKukushkin)

古いレプリカの検出に役立ちます。それに加えて、ポート値がデフォルトでない場合、または同じホストで複数のメンバーが実行されている場合、`Host` には `:{port}` が含まれます。

- `$$SCOPE-config` エンドポイントに関連付けられたヘッドレスサービスを作成します。(AlexanderKukushkin)

`"/config/"` エンドポイントは、クラスター全体の Patroni および Postgres 構成、履歴ファイルに関する情報を保持し、最後に最も重要なことに、`initialize` キーを保持します。Kubernetes マスターノードが再起動またはアップグレードされると、サービスのないエンドポイントが削除されます。ヘッドレスサービスは、削除されないようにします。

Bug fixes

- リーダーウォッチブロッキングクエリの読み取りタイムアウトを調整しました。(AlexanderKukushkin)

Consul のドキュメントによると、実際の応答タイムアウトは、同時要求のウェイクアップ時間を分散するために、指定された最大待機時間に追加されたランダムな少量の追加待機時間だけ増加します。最大持続時間に `wait/16` の追加時間を加算します。私たちの場合、大きいものに依拠して `wait/15` または 1 秒を追加します。

- レプリケーションプロトコルを介して postgres に接続するときは、常に `replication=1` を使用してください。(AlexanderKukushkin)

Postgres10 以降、`database=replication` を含む `pg_hba.conf` の行は、パラメータ `replication=database` を持つ接続を受け入れません。

- `wal-only` スタンバイクラスターの `recovery.conf` に `primary_conninfo` を書き込まないでください。(AlexanderKukushkin)

`standby_cluster` 構成で `host` も `port` も定義されていないにもかかわらず、パトロンは `primary_conninfo` を `recovery.conf` に入れていました。これは役に立たず、多くのエラーを生成していました。

15.26 バージョン 1.5.4

このバージョンでは、柔軟なロギングを実装し、多くのバグが修正されています。

New features

- ロギングインフラストラクチャの改善 (AlexanderKukushkin, LucasCapistrant, AlexanderAnikin)

ロギング構成は、環境変数だけでなく、Patroni 構成ファイルからも構成できます。構成を更新してリロードを実行するか、`SIGHUP` を Patroni プロセスに送信することにより、実行時にロギング構成を変更できるようにします。デフォルトでは、Patroni はログを `stderr` に書き込みますが、ログをファイルに直接書き込み、特定のサイズに達するとローテーションできるようになりました。それに加えて、カスタム `dateformat` のサポートと、各 Python モジュールのログレベルを微調整する可能性が追加されました。

- リーダー選挙中に現在のタイムラインを考慮に入れられるようにします。(AlexanderKukushkin)

現在、既知の最新のタイムライン上にはありませんが、ノードが自分自身を最も健全なノードと見なしている可能性があります。場合によっては、そのようなノードの昇格を回避したい場合があります。これは、`check_timeline` パラメータを `true` に設定することで達成できます（デフォルトの動作は変更されません）。

- スーパーユーザーの資格情報の緩和された要件

Libpq では、ユーザー名もパスワードも明示的に指定せずに接続を開くことができます。状況に応じて、pgpass ファイルに依存するか、pg_hba.conf の認証方法を信頼します。pg_rewind も libpq を使用しているため、同じように動作します。

- 環境変数を介して Consul サービスの登録とチェック間隔を構成する可能性を実装しました。(AlexanderKukushkin)

Consul でのサービスの登録は 1.5.0 で追加されましたが、これまでのところ、atoani.yaml を介してのみオンにすることができました。

Stability Improvements

- カスタムブートストラップ中に archive_mode を off に設定します。(AlexanderKukushkin)

クラスターが完全に機能するまで、wals と history ファイルのアーカイブは避けたいと思います。カスタムブートストラップに pg_upgrade が含まれる場合、本当に役立ちます。

- スタート時にグローバル設定をロードするときに 5 秒のバックオフを適用します。(AlexanderKukushkin)

Patroni の起動時に DCS をハンマーするのを避けるのに役立ちます。

- シャットダウン時に生成されるエラーメッセージの量を減らしました。(AlexanderKukushkin)

それらは無害ですが、むしろ迷惑であり、時には恐ろしいものでした。

- 作成時に recovery.conf の rwpem を明示的に保護しました。(LucasCapistrant)

このファイルにはレプリケーションユーザーとパスワードが含まれているため、atoni/postgres ユーザー以外は誰も読まないでください。

- HTTPServer 例外をロガーにリダイレクトしました。(JulienRiou)

デフォルトでは、このような例外は標準出力に記録され、通常のログが混乱していました。

Bug fixes

- pg_ctl プロセスの stdout への stderr パイプを削除しました (CodyCoons)

メインの Patroni プロセスから stderr を継承すると、すべての Postgres ログをすべてのパトロニログとともに表示できます。Patroni および Postgres のログは標準ツール (docker ログ、kubectrl など) を使用して消費される可能性があるため、これはコンテナ環境で非常に役立ちます。それに加えて、この変更により、postgres がいくつかの警告を stderr に書き込むときに Patroni が postmasterpid をキャッチできないというバグが修正されます。

- Consul サービスのチェック登録解除タイムアウトを Go 時間形式で設定します。(PavelKirillov)

明示的に言及されていない場合、時間単位の登録は失敗していました。

- standby_cluster クラスター構成のチェックを緩めます。(DmitryDolgov、AlexanderKukushkin)

有効な値として文字列のみを受け入れていたため、ポートを整数として指定し、create_replica_methods をリストとして指定することはできませんでした。

15.27 バージョン 1.5.3

互換性とバグ修正リリース。

- Zookeeper に対して python3 で実行する際の安定性を改善しました。(AlexanderKukushkin)

‘loop_wait’の変更により、Patroni は zookeeper から切断し、再接続しませんでした。

- postgres9.3 との壊れた互換性を修正しました (AlexanderKukushkin)

9.3 は replication='database' を理解しないため、レプリケーション接続を開くときは replication=1 を指定する必要があります

- HA ループごとに少なくとも 1 回は Consul セッションを更新し、consul セッションの例外の処理を改善してください。(AlexanderKukushkin)

ローカル Consul エージェントを再起動すると、ノードに関連するすべてのセッションが無効になります。セッションリフレッシュを時間通りに呼び出さず、セッションエラーの適切な処理を行わないと、プライマリの降格が発生しました。

15.28 バージョン 1.5.2

互換性とバグ修正リリース。

- kazoo-2.6.0 (AlexanderKukushkin) との互換性

リクエストが適切なタイムアウトで実行されるようにするために、Patroni は python-kazoo モジュールから create_connection メソッドを再定義します。kazoo の最後のリリースでは、create_connection メソッドの呼び出し方法がわずかに変更されました。

- Consul クラスターがリーダーを失ったときの Patroni のクラッシュを修正しました (AlexanderKukushkin)

touch_member メソッドの実装が正しくないためにクラッシュが発生していました。ブール値を返し、例外を発生させないはずです。

15.29 バージョン 1.5.1

このバージョンでは、永続的なレプリケーションスロットのサポートを実装し、pgBackRest のサポートを追加し、多数のバグを修正します。

New features

- 永久複製スロット (AlexanderKukushkin)

永続的なレプリケーションスロットは、フェールオーバー/スイッチオーバー時に保持されます。つまり、新しいプライマリの Patroni は、プロモートの直後に構成済みのレプリケーションスロットを作成します。スロットは、`patronictledit-config` を使用して構成できます。初期構成は、`:ref: bootstrap.dcs <yaml_configuration>` で行うこともできます。

- pgbackrest のサポートを追加しました。(YogeshSharma)

pgBackrest は既存の \$PGDATA フォルダに復元できます。これにより、最後のバックアップから変更されていないファイルはスキップされるため、迅速な復元が可能になります。この機能をサポートするために、新しいパラメーター `keep_data` が導入されました。追加の例については、[replica creation method](#) セクションを参照してください。

Bug fixes

- `/"standbycluster/"` ワークフローのいくつかのバグ修正 (AlexanderKukushkin)

詳細については、<https://github.com/zalando/patroni/pull/823> を参照してください。

- クラスター管理が一時停止し、DCS にアクセスできない場合の RESTAPI ヘルスチェックを修正しました。(AlexanderKukushkin)

回帰は <https://github.com/zalando/patroni/commit/90cf930036a9d5249265af15d2b787ec7517cf57> で導入されました

15.30 バージョン 1.5.0

このバージョンでは、PatroniHA クラスターがスタンバイモードで動作できるようになり、Windows で実行するための実験的なサポートが導入され、Consul に PostgreSQL サービスを登録するための新しい構成パラメーターが提供されます。

New features

- スタンバイクラスター (DmitryDolgov)

1 つ以上の Patroni ノードは、プライマリクラスターと一緒に（つまり別のデータセンターで）実行されるスタンバイクラスターを形成でき、プライマリクラスターのマスターから複製するスタンバイノードで構成されます。スタンバイクラスター内のすべての PostgreSQL ノードはレプリカです。これらのレプリカの 1 つは自分自身をリモートマスターから直接複製することを選択し、他のレプリカはカスケード方法でそこから複製します。この機能のより詳細な説明といくつかの構成例は、[here](#) にあります。

- Consul にサービスを登録する (PavelKirillov, AlexanderKukushkin)

consul *configuration* の ``register_service`` パラメーターが有効になっている場合、ノードは名前 ``scope`` とタグ ``master``、``replica`` または ``standby-leader`` を使用してサービスを登録します。

- 実験的な Windows のサポート (PavelGolub)

これからは、Windows で Patroni を実行することが可能になりますが、Windows のサポートは新しく、Linux のサポートほど実際のテストは受けていません。フィードバックをお待ちしております。

Improvements in patronictl

- パトロンを追加-k/--insecure フラグと restapi 証明書のサポート。(WilfriedRoset)

以前は、RESTAPI が自己署名証明書 ``patronictl`` で保護されている場合、それらを検証できませんでした。その検証を無効にする方法はありませんでした。*patronictl* を構成して、**証明書の検証を完全にスキップするか、構成の `:ref: `ctl: <patronictl_settings>` セクションで CA およびクライアント証明書を提供できるようになりました。**

- Patanictl スイッチオーバー/フェールオーバー出力から nofailover タグを持つメンバーを除外しました。(AlexanderAnikin)

以前は、これらのメンバーは、patanictl を介して対話型スイッチオーバーまたはフェイルオーバーを実行するときに、候補として誤って提案されました。

Stability improvements

- pg_controldata の非キー値出力行の解析を回避しました。(AlexanderAnikin)

特定の状況下で、pg_controldata はコロン文字なしで行を出力します。それは pg_controldata 出力を解析した Patroni コードでエラーを引き起こし、実際の問題を隠しました。多くの場合、このような行は、通常の出力の前に pg_controldata によって表示される警告、つまりバイナリメジャーバージョンが PostgreSQL データディレクトリのバージョンと一致しない場合に出力されます。

- リーダー選挙中のエラーメッセージにメンバー名を追加しました。(JanMussler)

リーダー選挙中に、Patroni はクラスターの既知のすべてのメンバーに接続し、そのステータスを要求します。このようなステータスは Patroni ログに書き込まれ、メンバーの名前が含まれます。以前は、メンバーにアクセスできない場合、エラーメッセージには名前が示されず、URL のみが含まれていました。

- レプリケーションスロットの作成時に WAL ポジションをすぐに予約しました。(AlexanderKukushkin)

9.6 以降、``pg_create_physical_replication_slot`` ファンクションは、追加のブールパラメータ ``immediately_reserve`` を提供します。これがデフォルトでもある ``false`` に設定されている場合、スロットは最初のクライアント接続を受信するまで WAL ポジションを予約せず、スロットの作成と初期クライアント接続。

- 厳密な同期レプリケーションのバグを修正しました。(AlexanderKukushkin)

``synchronous_mode_strict: true`` で実行している場合、場合によってはパトロンは ``*`` を ``synchronous_standby_names`` に入れ、ほとんどのレプリケーション接続の同期状態を ``potential`` に変更します。以前は、Patroni はそのような状況下で同期候補を選択できませんでした。状態が ``async`` のもののみを考慮したためです。

15.31 バージョン 1.4.6

Bug fixes and stability improvements

このリリースでは、PatroniAPI/master エンドポイントが非マスターノードに対して 200 を返す重大な問題が修正されています。これはレポートの問題であり、実際のスプリットブレインではありませんが、特定の状況下では、クライアントが読み取り専用ノードに転送される場合があります。

- 降格時に is_leader ステータスをリセットします。(AlexanderKukushkin、OleksiiKliukin)

降格されたクラスターメンバーが、/masterAPI 呼び出しでコード 200 で応答を停止することを確認します。

- 新しい"/cluster_unlocked/"フィールドを API 出力に追加します。(DmitryDolgov)

このフィールドは、クラスターでマスターが実行されているかどうかを示します。レプリカの 1 つ以外の他のノードを照会できない場合に使用できます。

15.32 バージョン 1.4.5

New features

- 新しい postgres 構成を適用するときのロギングを改善しました。(DonSeiler)

Patroni は、変更されたパラメーターの名前と値をログに記録します。

- Python3.7 の互換性 (ChristophBerg)

async は python3.7 の予約キーワードです

- メンバーがシャットダウンされたときに DCS で状態を"/stopped/"に設定します。(TonySorrentino)

これは、"/patronictl/"コマンドでメンバーの状態が"/stopped/"として表示されます。

- 古い postmaster.pid が実行中のプロセスと一致したときに記録されるメッセージを改善しました。(AntsAasma)

前のものは混乱を超えるものでした。

- パトロンのリロード機能を実装しました。(DonSeiler)

それ以前は、RESTAPI を呼び出すか、SIGHUP シグナルを Patroni プロセスに送信することによってのみ、構成をリロードすることができました。

- レプリカとして起動するときに controldata からいくつかのパラメーターを取得して適用します。(AlexanderKukushkin)

`max\_connections` の値およびグローバル構成で設定された他のいくつかのパラメーターは、プライマリで実際に使用される値よりも低い場合があります。これが発生すると、レプリカは起動できないため、手動で修正する必要があります。Patroni は、`pg\_controldata` から値を読み取って適用し、postgres を起動して `pending\_restart` フラグを設定することにより、これを処理します。

- 設定されている場合、postgres の起動時に LD_LIBRARY_PATH を使用します。(ChrisFraser)

Postgres の起動時に、Patroni は PATH、LC_ALL、および LANG 環境変数が設定されている場合、それらを渡していました。現在、LD_LIBRARY_PATH でも同じことをしています。誰かが PostgreSQL を非標準の場所にインストールした場合に役立つはずです。

- create_replica_method の名前を create_replica_methods に変更しました。(DmitryDolgov)

実際には配列であることを明確にするため。古い名前は、下位互換性のために引き続きサポートされています。

Bug fixes and stability improvements

- 一時停止状態の pg_rewind によるレプリカの開始の条件を修正しました。(OleksiiKliukin)

以前に pg_rewind を実行したレプリカを起動しないでください。

- update_lock が成功した場合にのみ、マスターヘルスチェックに 200 を応答します。(AlexanderKukushkin)

DCS がパーティション分割されている場合、Patroni が以前の（降格した）マスターのマスターであると自分自身を報告しないようにします。

- 新しい consul モジュールとの互換性を修正しました。(AlexanderKukushkin)

v1.1.0 から python-consul は内部 API を変更し、`dict` の代わりに `list` を使用してクエリパラメーターを渡すようになりました。

- シャットダウン中に PatroniRESTAPI スレッドからの例外をキャッチします。(AlexanderKukushkin)

これらのキャッチされない例外により、シャットダウン時に PostgreSQL が実行され続けました。

- Postgres がマスターとして実行されている場合にのみクラッシュリカバリを行います。(AlexanderKukushkin)

「生産中」、「シャットダウン中」、または「クラッシュリカバリ中」を報告するには `pg\_controldata` が必要です。他のすべての場合、クラッシュリカバリは必要ありません。

- 構成エラーの処理を改善しました。(HenningJacobs、AlexanderKukushkin)

Patroni 構成ファイルを更新し、SIGHUP を Patroni プロセスに送信することにより、実行時に多くのパラメーター（`restapi.listen` を含む）を変更することが可能です。この修正により、一部のパラメーターが無効な値を受け取ったときに、「restapi」スレッドからあいまいな例外が排除されます。

15.33 バージョン 1.4.4

Stability improvements

- poll_failover_result の競合状態を修正しました。(AlexanderKukushkin)

フェールオーバーにもスイッチオーバーにも直接影響しませんでした。まれに、前のリーダーがロックを解放したときに成功を報告するのが早すぎ、「/に Failedovertor/"None/"」が生成されました。"desired-node/" メッセージ。

- Postgres パラメーター名を大文字と小文字を区別しないものとして扱います。(AlexanderKukushkin)

Postgres パラメーターのほとんどには snake_case 名前がありますが、このルールには 3 つの例外があります。DateStyle、IntervalStyle、および TimeZone です。Postgres は、別のケースで記述された場合 (例: `timezone='some/tzn'`)、これらのパラメーターを受け入れます。ただし、Patroni は `pg_settings` でこれらのパラメーター名の大文字と小文字を区別しない一致を見つけることができず、結果としてそのようなパラメーターを無視しました。

- 実行中の postgres にアタッチし、クラスターが初期化されていない場合、スタートを中止しました。(AlexanderKukushkin)

Patroni は、すでに実行中の Postgres インスタンスに自分自身をアタッチできます。レプリカに到達する前に、マスターノードで Patroni の実行を開始することが不可欠です。

- Patanictlscaffold の動作を修正しました。(AlexanderKukushkin)

json エンコードされた文字列の代わりに dict オブジェクトを `touch_member` に渡すと、DCS 実装がエンコードを処理します。

- 一時停止中にリーダーキーの更新に失敗した場合、マスターを降格しないでください。(AlexanderKukushkin)

メンテナンス中に、DCS は読み取り要求への応答を継続しながら、書き込み要求に失敗し始める場合があります。その場合、Patroni は、DCS でリーダーロックの更新に失敗した後、Postgres マスターノードを読み取り専用モードにしていました。

- Patroni が新しい postmaster プロセスに気付いたときにレプリケーションスロットを同期します。(AlexanderKukushkin)

Postgres が再起動された場合、Patroni はレプリケーションスロットのリストが期待と一致することを確認する必要があります。

- 一時停止から抜けた後に `sysid` を確認し、レプリケーションスロットを同期します。(AlexanderKukushkin)

‘maintenance’モード中にデータディレクトリが完全に書き換えられた可能性があるため、‘データベースシステム識別子’がまだクラスターに属し、レプリケーションスロットが Patroni の期待と同期していることを確認する必要があります。

- postmaster ロックファイルが存在するデータディレクトリで Postgres を実行していないときに発生する可能性のある障害を修正しました。(AlexanderKukushkin)

postmaster ロックファイルから PID の再利用を検出します。docker コンテナで Patroni と Postgres を実行すると、このような問題が発生する可能性が高くなります。

- DCS が誤ってワイプされる保護を改善しました。(AlexanderKukushkin)

Patroni には、そのような場合のフェイルオーバーを防ぐための多くのロジックがあります。すべてのキーを元に戻すこともできます。ただし、この変更まで、`/config` キーを誤って削除すると、HA ループの 1 サイクルの一時停止モードがオフになりました。

- 無効なシステム ID が発生しても終了しないでください。(OleksiiKliukin)

クラスタシステム ID が空の場合、または検証チェックに合格しない場合は終了しないでください。その場合、クラスターは再初期化を必要とする可能性が最も高いです。結果メッセージでそれを言及

します。Patroni を終了しないでください。終了しないと、再初期化できません。

Compatibility with Kubernetes 1.10+

- 空のサブセットのチェックを追加しました (CodyCoons)

Kubernetes 1.10.0 以降は、`[]`ではなく `None`に設定された `Endpoints.subsets`を返すようになりました。

Bootstrap improvements

- `recovery.conf` の削除をオプションにしました。 (BradNicholson)

`bootstrap.<custom_bootstrap_method_name>.keep_existing_recovery_conf`が定義され、`True`に設定されている場合、Patroni は既存の `recovery.conf` ファイルを削除しません。これは、適切な `recovery.conf` を生成する `pgBackRest` などのツールを使用してバックアップからブートストラップする場合に役立ちます。

- `basebackup` ビルトインメソッドのオプションを許可します。 (OleksiiKliukin)

カスタムレプリカ作成メソッドに対して定義される方法と同様に、構成の `basebackup` セクションを定義することにより、組み込みの `basebackup` メソッドにオプションを提供できるようになりました。違いは `basebackup` セクションで受け入れられるフォーマットにあります。 `pg_basebackup` は `--key=value` と `--key` オプションの両方を受け入れるため、セクションの内容はキーと値のペアの辞書、または 1 要素の辞書またはキーのみのリスト (値を受け入れないオプションの場合)。追加の例については、 [replica creation method](#) セクションを参照してください。

15.34 バージョン 1.4.3

Improvements in logging

- 環境変数からログレベルを設定可能にしました。 (AndyNewton、KeyvanHedayati)

`PATRONI_LOGLEVEL`-一般的なログレベルを設定します `PATRONI_REQUESTS_LOGLEVEL`-すべての HTTP リクエストのログレベルを設定します。 `KubernetesAPI` 呼び出し可能なログレベルの名前を取得するには、 `Python` ロギングのドキュメント <<https://docs.python.org/3.6/library/logging.html#levels>>を参照してください

Stability improvements and bug fixes

- ウォッチがタイムアウトしたときに `etcd` クラスタートポロジを再検出しないでください。 (AlexanderKukushkin)

`etcd` 構成にホストが 1 つしかなく、まさにこのホストにアクセスできない場合、Patroni はクラスタートポロジの検出を開始していましたが、成功しませんでした。代わりに、次の利用可能なノードに切り替えるだけです。

- カスタムブートストラップの後に `bootstrap.pg_hba` のコンテンツを `pg_hba.conf` に書き込みます。 (AlexanderKukushkin)

これで、`initdb`を使用した通常のブートストラップと同様に動作します

- シングルユーザーモードはユーザー入力を待っていて、終了しませんでした。(AlexanderKukushkin)
 回帰は <https://github.com/zalando/patroni/pull/576> で導入されました

15.35 バージョン 1.4.2

Improvements in patronictl

- スケジュールされたフェールオーバーの名前をスケジュールされたスイッチオーバーに変更しました。(AlexanderKukushkin)

フェールオーバーとスイッチオーバーの機能はバージョン 1.4 で分離されましたが、``patronictl list`` は依然として ``Scheduledswitchover`` ではなく ``Scheduledfailover`` を報告していました。

- 保留中の再起動に関する情報を表示する (AlexanderKukushkin)

構成の変更を適用するには、postgres を再起動する必要がある場合があります。Patroni はすでに RESTAPI とノードのステータスを DCS に書き込むときにヒントを出していましたが、簡単に表示する方法がありませんでした。

- 構成ファイルの `cluster_name` で動作するように `show-config` を作成しました。(AlexanderKukushkin)
``patronictl edit-config`` と同様に機能します

Stability improvements

- ブートストラップ中に `pg_controldata` を呼び出さないようにしました。(AlexanderKukushkin)

`initdb` またはカスタムブートストラップ中に、`pgdata` が空ではないが `pg_controldata` がまだ書き込まれていない時間枠があります。このような場合、`pg_controldata` 呼び出しはエラーメッセージで失敗していました。

- `psutil` から発生した例外を処理します。(AlexanderKukushkin)

`cmdline` は、``cmdline()`` メソッドが呼び出されるたびに読み取られて解析されます。調査中のプロセスがすでに消えている可能性があります。その場合は ``NoSuchProcess`` が発生します。

Kubernetes support improvements

- k8sAPI からのエラーを飲み込まないでください。(AlexanderKukushkin)

KubernetesAPI への呼び出しは、さまざまな理由で失敗する可能性があります。場合によっては、そのような呼び出しを再試行する必要があります。他の場合には、エラーメッセージと例外スタックトレースをログに記録する必要があります。ここでの変更は、Kubernetes のアクセス許可の問題のデバッグに役立ちます。

- Kubernetes の例 Dockerfile を更新して、マスターブランチから Patroni をインストールします。(MaciejSzulik)

それ以前は ``feature/k8s`` を使用していましたが、これは時代遅れになりました。

- k8s でパトニーを実行するための適切な RBAC を追加しました。(MaciejSzulik)

クラスターのポッドに割り当てられるサービスアカウント、必要な権限のみを保持するロール、およびサービスアカウントとロールを接続するロールバインディングを追加します。

15.36 バージョン 1.4.1

Fixes in patronictl

- フェールオーバー先のメンバーの提案リストに現在のリーダーを表示しないでください。(アレクサンダー・ククシュキン)

クラスターにリーダーが存在し、フェールオーバーが可能なメンバーのリストから除外する必要がある場合でも、`patitanictl` フェールオーバーは引き続き機能します。

- パトローニのスイッチオーバーを古い Patroniapi (AlexanderKukushkin) と互換性のあるものにします

POST/switchoverRESTAPI 呼び出しがステータスコード 501 で失敗した場合、もう一度呼び出しますが、`/failover` エンドポイントに。

15.37 バージョン 1.4

このバージョンでは、DCS として Kubernetes を使用するためのサポートが追加され、Etdcd、Zookeeper、または Consul を追加デプロイせずに、Kubernetes で Patroni をクラウドネイティブエージェントとして実行できます。

Upgrade notice

pip を介して Patroni をインストールすると、(Etdcd、Zookeeper、Consul または Kubernetes のライブラリ、または AWS のサポートなど) の依存関係が取り込まれなくなります。それらを有効にするには、`pipinstall` コマンドで明示的にリストする必要があります。

Kubernetes support

Kubernetes ベースの DCS を実装します。エンドポイントメタデータは、構成とリーダーキーを保存するために使用されます。ポッド定義内のメタデータフィールドは、メンバー関連のデータを保存するために使用されます。Endpoints の使用に加えて、Patroni は ConfigMap をサポートしています。この機能の詳細については、[Kubernetes chapter of the documentation](#)

Stability improvements

- postmaster プロセスを別のオブジェクトに分解しました (AntsAasma)

このオブジェクトは、pid と開始時間を介して実行中のポストマスタープロセスを識別し、ポストマスターが背後で再起動されたとき、または postgres ディレクトリがファイルシステムから消えたときの状況の検出（および解決）を簡素化します。

- HA サイクルのすべてのループで Patroni が発行する SELECT の量を最小限に抑えます。(AlexanderKukushkin)

HA ループのすべての反復で、Patroni はリカバリーステータスと絶対ウォールポジションを知る必要があります。今後、Patroni は、レプリカで 2 回、マスターで 3 回ではなく、単一の SELECT のみを実行してこの情報を取得します。

- ロックがある場合にのみ、シャットダウン時にリーダーキーを削除します (AntsAasma)

無条件削除は、不要で誤解を招く例外を生成していました。

Improvements in patronictl

- パトロンにバージョンコマンドを追加しました。(AntsAasma)

インストールされている Patroni のバージョンと実行中の Patroni インスタンスのバージョン（クラスター名が指定されている場合）が表示されます。

- パトロンの一部のコマンドで `cluster_name` 引数を指定するオプションを作成しました。(AlexanderKukushkin、AntsAasma)

Patronictl が `scope` が定義された通常の Patroni 構成ファイルを使用している場合に動作します。

- スケジュールされたスイッチオーバーとメンテナンスモードに関する情報を表示します。(AlexanderKukushkin)

それ以前は、Patroni ログまたは DCS から直接この情報を取得することのみが可能でした。

- `patronictl reinit` を改善 (AlexanderKukushkin)

Patroni が他のアクション、つまり `postgres` を起動しようとしているときに、`patronictl reinit` が続行を拒否することがありました。`patronictl` は、このような長時間実行アクションをキャンセルするコマンドを提供せず、唯一の（危険な）作業は、データディレクトリを手動で削除することでした。`reinit` の新しい実装は、再初期化に進む前に、他の長時間実行アクションを強制的にキャンセルします。

- `patronictl pause` と `patronictl resume` に `--wait` フラグを実装しました。(AlexanderKukushkin)

`patronictl` は、要求されたアクションがクラスター内のすべてのノードによって承認されるまで待機します。このような動作は、DCS のすべてのノードの `pause` フラグを公開し、RESTAPI を介して実現されます。

- `patronictl failover` を `patronictl switchover` に名前変更しました (AlexanderKukushkin)

以前の `failover` は実際にはスイッチオーバーのみを実行できました。リーダーのいないクラスターでの続行を拒否しました。

- `patronictl failover` の動作を変更しました (AlexanderKukushkin)

リーダーがなくても機能しますが、その場合、新しいリーダーになるノードを明示的に指定する必要があります。

Expose information about timeline and history

- DCS および API を介して現在のタイムラインを公開しました。(AlexanderKukushkin)

クラスターの各メンバーの現在のタイムラインに関する情報を保存します。この情報は API を介してアクセスでき、DCS に保存されます

- DCS の `/history` キーにプロモーション履歴を保存します (AlexanderKukushkin)

さらに、対応するプロモーションのタイムスタンプで強化されたタイムライン履歴を DCS の `/history` キーに保存し、プロモートごとに更新します。

Add endpoints for getting synchronous and asynchronous replicas

- 新しい `/sync` および `/async` エンドポイントを追加します。(AlexanderKukushkin、OleksiiKliukin)

これらのエンドポイント (`/synchronous` および `/asynchronous` としてもアクセス可能) は、対応する同期および非同期レプリカに対してのみ 200 を返します (``noloadbalance`` としてマークされたエンドポイントを除く)。

Allow multiple hosts for Etcd

- Etcd 構成に新しい ``hosts`` パラメーターを追加します。(AlexanderKukushkin)

このパラメーターには、実行中の etcd クラスターメンバーのリストを検出して入力するために使用されるホストの初期リストが含まれている必要があります。何らかの理由で作業中に検出されたホストのこのリストが使い果たされた場合 (そのリストに使用可能なホストがない場合)、Patroni は ``hosts`` パラメーターから最初のリストに戻ります。

15.38 バージョン 1.3.6

Stability improvements

- postgres が実行されているかどうかを確認するときに、プロセスの開始時間を確認します。(アリアスマ)

`postmaster.pid` をクリーンアップしないクラッシュの後、同じ pid を持つ新しいプロセスが存在する可能性があり、その結果、`is_running()` が誤検出され、あらゆる種類の悪い動作につながります。

- データディレクトリを失った場合、ブートストラップの前に `postgresql` をシャットダウンします。(ainlolcat)

マスターのデータディレクトリが強制的に削除されると、postgres プロセスはしばらくの間生きたままで、その以前のマスターの代わりに作成されたレプリカの開始または複製を防ぐことができます。この修正により、Patroni は `postmasterpid` とその開始時刻をキャッシュし、対応するデータディレクトリが削除された後もまだ実行されている場合に備えて、古い `postmaster` を終了させます。

- postgres マスターが死亡した場合にシングルユーザーモードでクラッシュリカバリを実行します。(AlexanderKukushkin)

スタンバイとしてすぐに起動するのは安全ではなく、postgres が正常にシャットダウンされていない場合、`pg_rewind` を実行することはできません。シングルユーザークラッシュリカバリは、`pg_rewind` が有効になっている場合、または現時点でマスターがない場合にのみ始まります。

Consul improvements

- Consul にデータセンター構成を提供できるようにします (ViliusOkockis、AlexanderKukushkin)

その前に、Patroni は常に実行されるホストのデータセンターと通信していました。

- 常に `X-Consul-Tokenhttp` ヘッダーでトークンを送信します。(AlexanderKukushkin)

`consul.token` が Patroni 構成で定義されている場合、常に `'X-Consul-Token'http` ヘッダーで送信します。python-consul モジュールは、ConsulRESTAPI で「一貫性」を保とうとします。これは、`consul.token_` のクエリパラメーターとしてトークンを受け入れませんが、`'X-Consul-Token'` ヘッダーで引き続き動作します。

- 指定された値が可能な最小値より小さい場合は、セッション TTL を調整します。(StasFomin、AlexanderKukushkin)

Patroni 構成で提供される TTL が、Consul でサポートされる最小値よりも小さい場合があります。その場合、Consul エージェントは新しいセッションの作成に失敗します。セッションがないと、Patroni は ConsulKV ストアにメンバーキーとリーダーキーを作成できず、クラスターが異常終了します。

Other improvements

- 環境変数 `PATRONI_LOGFORMAT` を介してカスタムログフォーマットを定義します。(StasFomin)

Patroni ログのタイムスタンプやその他の同様のフィールドがシステムロガーによって既に追加されている場合（通常は Patroni がサービスとして実行されている場合）、無効にすることを許可します。

15.39 バージョン 1.3.5

Bugfix

- データディレクトリが削除された場合、ロールを「未初期化」に設定しました。(AlexanderKukushkin)
- ノードがマスターとして実行されている場合、フェールオーバーが妨げられていました。

Stability improvement

- postgres を起動しようとして失敗した場合、postmaster をシングルユーザーモードで実行してみてください。(AlexanderKukushkin)

通常、このような問題は、マスターとして実行されているノードが終了し、タイムラインが分岐したときに発生します。`recovery.conf` に `restore_command` が定義されている場合、postgres が起動を中止し、制御データを変更しない可能性が非常に高くなります。クリーンシャットダウンが必要な `pg_rewind` を使用できなくなります。

Consul improvements

- セッション作成時にヘルスチェックを指定できるようにしました。(AlexanderKukushkin)

指定されない場合、Consul は `"/serfHealth/"` を使用します。一方で、分離されたマスターを迅速に検出できますが、別の側面では、Patroni が短いネットワークラグを許容できなくなります。

Bugfix

- Python3 の watchdog を修正しました (AntsAasma)

`ioctl()` 呼び出しインターフェースの誤解。`mutable=False` の場合、`fcntl.ioctl()` は実際に `arg` バッファーを返します。`int` と `str` の比較がエラーを返さなかったため、これは Python2 で誤って機能しました。エラー報告は、実際には Python2 では `IOError` を、Python3 では `OSError` を送出することによって行われます。

15.40 バージョン 1.3.4

Different Consul improvements

- Consul トークンをヘッダーとして渡します。(AndrewColinKissa)
ヘッダーは、トークンを consul [API](#) に渡すための好ましい方法になりました。
- Consul の高度な構成 (AlexanderKukushkin)
scheme、token、クライアントおよび CA 証明書 scheme を指定する可能性。
- python-consul-0.7.1 以降との互換性 (AlexanderKukushkin)
新しい python-consul モジュールはいくつかのメソッドのシグネチャを変更しました
- `"/CouldnottakeoutTTLlock/"` メッセージはログに記録されませんでした。(AlexanderKukushkin)
重大なバグではありませんが、適切なログ記録がないため、問題が発生した場合の調査が複雑になります。

Quote synchronous_standby_names using quote_ident

- synchronous_standby_names を postgresql.conf に書き込むときは、その値をクォートする必要があります。(AlexanderKukushkin)
適切に引用符で囲まれていない場合、PostgreSQL は同期レプリケーションを効果的に無効にして、動作し続けます。

Different bugfixes around pause state, mostly related to watchdog (AlexanderKukushkin)

- ウォッチドッグがアクティブでない場合はキープアライブを送信しない
- 一時停止モードでウォッチドッグをアクティブにしない
- 一時停止モードで正しい postgres 状態を設定する
- postgres が停止している場合、API からクエリを実行しようとしないでください

15.41 バージョン 1.3.3

Bugfixes

- synchronous_mode_strict がオンになっていても、昇格直後に同期レプリケーションが無効になりました。(AlexanderKukushkin)
- バックアップから復元した後に見つからない場合は、空の pg_ident.conf ファイルを作成します。(AlexanderKukushkin)
- postgres だけでなく、すべてのデータベースへの pg_hba.conf でのオープンアクセス (FrancoBel-lagamba)

15.42 バージョン 1.3.2

Bugfix

- Patanictledit-config が ZooKeeper で動作しませんでした。(AlexanderKukushkin)

15.43 バージョン 1.3.1

Bugfix

- `_MemberStatus` の変更により、API を介したフェイルオーバーが壊れました。(AlexanderKukushkin)

15.44 バージョン 1.3

バージョン 1.3 では、カスタムブートストラップの可能性が追加され、`pg_rewind` のサポートが大幅に改善され、同期モードのサポートが強化され、`patanictl` に構成編集が追加され、Linux でウォッチドッグサポートが実装されます。さらに、これは PostgreSQL10 で正しく動作する最初のバージョンです。

Upgrade notice

Patroni の新しいバージョンとの既知の互換性問題はありません。バージョン 1.2 からの構成は、変更せずに機能するはずです。新しいパッケージをインストールして Patroni を再起動する (PostgreSQL の再起動が発生します)、または Patroni を最初に *pause mode* に入れてからクラスター内のすべてのノードで Patroni を再起動することにより、アップグレードできます (一時停止モードの Patroni は停止/startPostgreSQL)、最後に一時停止モードから再開します。

Custom bootstrap

- クラスターのブートストラッププロセスを構成可能にしました。(AlexanderKukushkin)

クラスター内の最初のノードを初期化するときに、`initdb` の代わりにカスタムブートストラップスクリプトを許可します。`bootstrap` コマンドは、クラスターの名前とデータディレクトリへのパスを受け取ります。結果として得られるクラスターは、リカバリを実行するように構成できるため、バックアップからブートストラップし、ポイントインタイムリカバリを実行できます。この機能の詳細については、`initdb` を参照してください。

Smarter pg_rewind support

- 現在のマスターとのタイムラインの違いを見て `pg_rewind` を実行するかどうかを決定します。(AlexanderKukushkin)

以前は、Patroni には `pg_rewind` をトリガーするための一定の条件セットがありました。これらすべての場合に共通して、一部のレプリカが新しいマスターよりも進んでいる可能性があります。場合によっては、`pg_rewind` は何もしませんでした。他の場合には、必要なときに実行されませんでした。この限られたルールリストに依存する代わりに、Patroni にマスターとレプリカの WAL 位置を (ストリーミングレプリケーションプロトコルを使用して) 比較させ、レプリカに巻き戻しが必要かどうかを確実に判断します。

Synchronous replication mode strict

- strict モードを追加して、同期レプリケーションのサポートを強化しました。(JamesSewell、AlexanderKukushkin)

通常、`synchronous_mode` が有効になっていて、マスターにレプリカが接続されていない場合、Patroni は同期レプリケーションを無効にして、マスターを書き込みに使用できるようにします。`synchronous_mode_strict` オプションは、設定されている場合、Patroni がレプリカがない場合に同期レプリケーションを無効にしないように変更し、マスターにデータを書き込むすべてのクライアントを効果的にブロックします。自動フェールオーバーによるデータ損失を防止する同期モードの保証に加えて、strict モードは、各書き込みが 2 つのノードに永続的に保存されるか、クラスター内にノードが 1 つしかない場合に完全に行われないことを保証します。

Configuration editing with patranictl

- Patanictl に構成編集を追加しました。(AntsAasma、AlexanderKukushkin)

DCS に保存されている動的クラスター構成を編集する機能を `patanictl` に追加します。コマンドラインからのパラメーター/値の指定、`$EDITOR` の呼び出し、または `yaml` ファイルからの構成の適用をサポートします。

Linux watchdog support

- Linux のウォッチドッグサポートを実装します (AntsAasma)

Patroni が実行されていないか応答していないノードをリポートするために、Linux ソフトウェアウォッチドッグをサポートしてください (高負荷のためなど) Linux ソフトウェアウォッチドッグは、応答しないノードをリポートします。Patroni 構成のウォッチドッグセクションから、使用するウォッチドッグデバイス (デフォルトは `/dev/watchdog`) とモード (`on`、`automatic`、`off`) を構成できます。[watchdog documentation](#) から詳細情報を取得できます。

Add support for PostgreSQL 10

- Patroni は、これまでにリリースされた PostgreSQL10 のすべてのベータ版と互換性があり、リリースされる PostgreSQL10 と互換性があると期待しています。

PostgreSQL-related minor improvements

- Patroni 構成ファイルまたは DCS の動的構成を介して `pg_hba.conf` を定義します。(AlexanderKukushkin)

構成の `postgresql` セクションの `pg_hba` サブセクションで `pg_hba.conf` の内容を定義できます。これにより、すべてのノードにログインし、手動で変更して構成をリロードする代わりに、DCS で 1 つだけを定義する必要があるため、複数のノードでの `pg_hba.conf` の管理が簡単になります。

定義されると、このセクションの内容は現在の `pg_hba.conf` を完全に置き換えます。`hba_file` PostgreSQL パラメーターが設定されている場合、Patroni はそれを無視します。

- UNIX ソケットを介したローカル PostgreSQL クラスターへの接続をサポートしました。(AlexanderKukushkin)

Patroni 構成の `postgresql` セクションに `use_unix_socket` オプションを追加します。`true` に設定され、PostgreSQL `unix_socket_directories` オプションが空でない場合、Patroni は最初の値を使用してローカル PostgreSQL クラスターに接続できます。`unix_socket_directories` が定義されていない場合、Patroni はそのデフォルト値を想定し、PostgreSQL 接続文字列の `host` パラメーターを完全に省略します。

- リロード時のスーパーユーザーとレプリケーション資格情報の変更をサポートしました。(AlexanderKukushkin)
- PostgreSQL データディレクトリ (@jouis) の外部への構成ファイルの保存をサポート

新しい構成 `postgresql` 構成指示子 `config_dir` を追加します。デフォルトはデータディレクトリであり、Patroni が書き込み可能である必要があります。

Bug fixes and stability improvements

- EtcdEventIndexCleared および EtcdWatcherCleared 例外を処理します。(AlexanderKukushkin)
無駄な再試行を回避することにより、Etcd によってウォッチ操作が終了したときのリカバリを高速化します。
- Etcd 障害時のエラースピニングを削除し、ログスパムを減らしました。(AntsAasma)
2 回目以降の Etcd 接続障害で、すぐに再試行してログにスタックトレースを発行しないようにします。
- PostgreSQL プロセスをフォークするときにロケール変数をエクスポートします。(OleksiiKliukin)
NLS でビルドされた PostgreSQL の英語以外のロケールで `postmastergotmultithreadedduringstartup` 致命的エラーを回避します。
- レプリケーションスロットをドロップする際の追加チェック (AlexanderKukushkin)
場合によっては、Patroni が WAL 送信者によってレプリケーションスロットをドロップできない場合があります。
- PostgreSQL の命名規則に準拠するために、レプリケーションスロット名を 63 (NAMEDATALEN-1) 文字に切り詰めます。(NickScott)
- Patroni から PostgreSQL クラスターへの余分な接続が開かれる競合状態を修正しました。(AlexanderKukushkin)
- ノードが空のデータディレクトリで再起動したときにリーダーキーを解放します。(AlexKerney)
- リーダーなしでブートストラップを実行するときに非同期エグゼキューターをビジーに設定します。(AlexanderKukushkin)
そうしないと、ノードが別のクラスターに属していることを示すエラーが発生する可能性があります。
- WAL-E レプリカの作成方法を改善しました (JoarWandborg、AlexanderKukushkin)。
 - WAL-E ベースバックアップを解析するときに `csv.DictReader` を使用し、スペース区切りの日時で ISO 日付を受け入れます。
 - 復元する WAL の量を見積もるために、レプリカからの現在の WAL 位置のフェッチをサポートします。以前は、マスターノードでのみ使用可能なシステム情報機能呼び出すために使用されるコード。

15.45 バージョン 1.2

このバージョンでは、同期レプリケーションの処理が大幅に改善され、起動プロセスとフェールオーバーの信頼性が向上し、PostgreSQL9.6 サポートが追加され、多くのバグが修正されています。さらに、これらのリリースノートを含むドキュメントは <https://patroni.readthedocs.io> に移動されました。

Synchronous replication

- 同期レプリケーションのサポートを追加します。(アリアスマ)

新しい構成変数 `synchronous_mode` を追加します。有効にすると、Patroni は `synchronous_standby_names` を管理して、利用可能な正常なスタンバイがあるたびに同期レプリケーションを有効にします。同期モードが有効になっている場合、Patroni は、マスター障害時に同期複製していたスタンバイにのみ自動的にフェールオーバーします。これは、このような場合、ユーザーに表示されるトランザクションが失われないことを効果的に意味します。詳細な説明と実装の詳細については、`synchronous_mode` を参照してください。

Reliability improvements

- PostgreSQL が 100 % 正常ではない場合、`leader optime` キーに保存されているリーダーの位置を更新しようとししないでください。リーダーキーの更新に失敗したら即降格。(アレクサンダー・ククシュキン)
- 新しいレプリカのクローンを作成するターゲットのリストから異常なノードを除外します。(アレクサンダー・ククシュキン)
- Etcd の場合と同様に、Consul のリトライとタイムアウトの戦略を実装します。(アレクサンダー・ククシュキン)
- `--dcs` および `--config-file` を `patronictl` のすべてのオプションに適用します。(アレクサンダー・ククシュキン)
- すべての postgres パラメーターを `postgresql.conf` に書き込みます。(アレクサンダー・ククシュキン)
`pg_ctl` だけで Patroni が構成した PostgreSQL を起動できます。
- 構成にユーザーがいない場合の例外を回避します。(キリルプーシキン)
- 異常なクラスターの一時停止を許可します。この修正の前は、`patronictl` は、一時停止を実行しようとするノードが異常な場合に救済されました。(アレクサンダー・ククシュキン)
- リーダーウォッチの機能を改善。(アレクサンダー・ククシュキン)

以前は、レプリカは常にリーダーキーを監視していました(タイムアウトまたはリーダーキーが変更されるまでスリープ)。この変更により、レプリカの PostgreSQL が `running` 状態にあるときのみ監視し、PostgreSQL の停止/起動または再起動中は監視しません。

- SIGCHILD を PID1 として処理するときに競合状態に遭遇しないようにしました。(AlexanderKukushkin)

Patroni 内の同じプロセスが新しいプロセスを生成し、それらから SIGCHILD を処理したため、以前は Docker コンテナ内で実行すると競合状態が発生する可能性があります。この変更では、Patroni に `fork/exec` を使用し、元の PID1 プロセスに子からのシグナルの処理を任せます。

- WAL-E 復元を修正。(OleksiiKliukin)

以前は、WAL-E 復元は `no_master` フラグを使用してマスターとのコンサルティングを完全に回避していたため、Patroni は常に `pg_basebackup` ではなく WAL からの復元を選択していました。この変更により、`no_master` の元の意味に戻ります。つまり、マスターが実行されていない場合、PatroniWAL-Erestore はレプリケーション方法として選択できます。後者は、メソッドに渡された接続文字列を調べることによってチェックされます。さらに、リトライメカニズムをより堅牢にし、他の細目を処理します。

- 非同期 DNS リゾルバーキャッシュを実装します。(アレクサンダー・ククシュキン)

DNS が一時的に利用できない場合の失敗を避けます (たとえば、ノードが受信した過剰なトラフィックが原因で)。

- 起動状態とマスター起動タイムアウトを実装します。(アリ・アスマ、アレクサンダー・ククシュキン)

以前は `pg_ctl` はタイムアウトを待ってから、喜んで PostgreSQL が実行されていることを検討しました。これにより、PostgreSQL は実際には実行されていないときに実行中としてリストに表示され、競合状態を引き起こし、フェイルオーバー、クラッシュリカバリ、またはフェイルオーバーによって中断されてリワインドに失敗したクラッシュリカバリを引き起こしました。この変更により、`master_start_timeout` パラメーターが追加され、メイン HA ループに新しい状態 `starting` が導入されます。`master_start_timeout` が 0 の場合、フェイルオーバー候補があるとすぐにマスターがクラッシュするとすぐにフェイルオーバーします。それ以外の場合、Patroni はマスターで PostgreSQL の起動を試行した後、タイムアウトの間待機します。有効期限が切れると、可能であればフェールオーバーします。手動フェイルオーバー要求は、タイムアウトの有効期限が切れる前であっても、マスターのクラッシュ中に受け入れられます。

`timeout` パラメーターを `restart` API エンドポイントと `patronictl` に導入します。設定されており、再起動にタイムアウトよりも時間がかかる場合、PostgreSQL は異常であると見なされ、他のノードがリーダーロックを取得できるようになります。

- 一時停止モードでの `pg_rewind` の動作を修正しました。(アリアスマ)

Patroni が巻き戻しが必要であると考えているが、巻き戻しが不可能な場合 (つまり、`pg_rewind` が存在しない場合)、一時停止モードでの不必要な再起動を避けます。`pg_rewind` 関連の Patroni 構成セクションに `superuser` 認証が欠落している場合、`superuser` (デフォルトの OS ユーザー) のデフォルトの `libpq` 値にフォールバック。

- コールバックの実行をシリアル化します。新しいコールバックが実行されようとしているときに、同じタイプの以前のコールバックを強制終了します。コールバックの実行時にゾンビプロセスを生成する問題を修正します。(アレクサンダー・ククシュキン)

- リーダーキーが DCS で設定されているが、このリーダーキーへの更新が失敗する場合、以前のマスターの昇格を避けます。(アレクサンダー・ククシュキン)

これにより、現在のマスターが Etcd および/"一貫性のない読み取り/"を許可する他の DCS 内の少数のノードと一緒にパーティション分割されたときに、その役割を維持し続ける問題を回避できます。

Miscellaneous

- ブートストラップに `post_init` 構成オプションを追加。(アレハンドロ・マルティネス)

Patroni は、`initdb` を実行し、新しいクラスター用に PostgreSQL を起動した直後に、このオプションのスクリプト引数を呼び出します。スクリプトは `superuser` を含む接続 URL を受け取り、パス

ワードを含む `.pgpass` ファイルを指すように `PGPASSFILE` を設定します。スクリプトが失敗すると、Patroni の初期化も失敗します。これは、新しいユーザーを追加したり、新しいクラスターで拡張機能を作成したりする場合に役立ちます。

- PostgreSQL9.6 サポートを実装します。(アレクサンダー・ククシュキン)

`wal_level = replica` を `hot_standby` の同義語として使用し、`pending_restart` フラグが変更されたときにフラグを回避します。(アレクサンダー・ククシュキン)

Documentation improvements

- Patroni main loop workflow diagram _を追加します。(アレハンドロ・マルティネス、アレクサンダー・ククシュキン)
- README を改善し、Helm チャートとリリースノートへのリンクを追加。(ラウリアップル)
- Patroni のドキュメントを Read the Docs に移動します。最新のドキュメントは、<https://patroni.readthedocs.io> で入手できます。(OleksiiKliukin)

ドキュメントをさまざまなデバイス（スマートフォンを含む）から簡単に表示および検索できるようにします。

- パッケージをセマンティックバージョニングに移動します。(OleksiiKliukin)

Patroni は、major.minor.patch バージョンスキーマに従って、小さいながらも重大なバグ修正で新しいマイナーバージョンがリリースされないようにします。すべてのパッチが含まれるマイナーバージョンのリリースノートのみを公開します。

15.46 バージョン 1.1

このリリースでは、一時停止モードを導入することにより Patroni クラスターの管理が改善され、スケジュールされた条件付きの再起動によるメンテナンスが改善され、Patroni と Etcd または Zookeeper との相互作用がより回復力があり、`patronictl` が大幅に強化されます。

Upgrade notice

1.0 より前のリリースからアップグレードする場合は、1.0 リリースノートの資格情報と構成形式の変更についてお読みください。

Pause mode

- 一時停止モードを導入して、Patroni を PostgreSQL インスタンス (MuratKabilov、AlexanderKukushkin、OleksiiKliukin) の管理から一時的にデタッチします。

以前は、PostgreSQL を終了せずに停止するには、Patroni に SIGKILL シグナルを送信する必要がありました。新しい一時停止モードは、Patroni を終了せずに、クラスター全体で PostgreSQL から Patroni をデタッチします。Pacemaker のメンテナンスモードに似ています。Patroni は、DCS のメンバーキーとリーダーキーの更新を引き続き担当しますが、その過程で PostgreSQL サーバーを起動、停止、または再起動しません。いくつかの例外があります。たとえば、手動フェールオーバー、再初期化、および再起動は引き続き許可されます。 [a detailed description of this feature](#) を読むことができます。

さらに、`patanictl` は、一時停止モードを切り替える新しい `pause` および `resume` コマンドをサポートします。

Scheduled and conditional restarts

- リスタート API コマンドに条件を追加しました。(OleksiiKliukin)

この変更により、再起動を行うために検証できるいくつかの条件を追加することにより、Patroni の再起動が強化されます。条件には、PostgreSQL ロールがマスターまたはレプリカの場合の再起動、PostgreSQL バージョン番号の確認、または構成の変更を適用するために再起動が必要な場合にのみ再起動することです。

- スケジュールされた再起動を追加します。(OleksiiKliukin)

将来的に再起動をスケジュールできるようになりました。ノードごとに 1 つのスケジュールされた再起動のみがサポートされています。スケジュールされた再起動が不要になった場合は、クリアすることができます。スケジュールされた再起動と条件付きの再起動の組み合わせがサポートされているため、たとえば、夜間に PostgreSQL のマイナーアップグレードをスケジュールし、postgres 固有のロジックを管理スクリプトに追加せずに、古いマイナーバージョンを実行しているインスタンスのみを再起動できます。

- 条件付き再起動とスケジュールされた再起動のサポートをパトロンに追加しました (MuratKabilov)。

パトロンリスタートは、いくつかの新しいオプションをサポートします。スケジュールされたアクションをクリーンアップするための `patanictlflush` コマンドもあります。

Robust DCS interaction

- `loop_wait` に応じて Kazoo のタイムアウトを設定しました。(AlexanderKukushkin)

もともと、`ping_timeout` および `connect_timeout` の値は、ネゴシエートされたセッションタイムアウトから計算されました。Patroni `loop_wait` は考慮されませんでした。その結果、1 回の再試行にはセッションタイムアウトよりも時間がかかる可能性があり、Patroni は強制的にロックを解放して降格します。

この変更により、`ping` と接続のタイムアウトが `loop_wait` の値の半分に設定され、接続の問題の検出が高速化され、ロックを失う前に接続試行を再試行するための十分な時間が残されます。

- 元のリクエストが成功した後にのみ Etcd トポロジを更新します。(AlexanderKukushkin)

元のリクエストが完了するまで、クライアントが知っている ETCD トポロジの更新を延期します。クラスタートポロジを取得するときは、Etcd クラスター内の既知のノード数に応じて再試行タイムアウトを実装します。これにより、クライアントは、ノードの最新のリストを取得するための要求の結果を取得することを好みます。

どちらの変更も、ネットワークの問題に直面した際の DCS への Patroni 接続をより堅牢にします。

Patronictl, monitoring and configuration

- API を介してストリーミングレプリカに関する情報を返します。(FeikeSteenbergen)

以前は、変更のストリーミングに失敗した PostgreSQL インスタンスについて Patroni にクエリを実行する信頼できる方法はありませんでした（たとえば、接続の問題が原因で）。この変更により、`/patroni` エンドポイントを介して `pg_stat_replication` の内容が公開されます。

- パトロンを追加しました。scaffold コマンド (OleksiiKliukin)

Etcd にクラスター構造を作成するコマンドを追加します。クラスターはユーザー指定の `sysid` と `leader` で作成され、リーダーとメンバーの両方のキーが永続的にされます。このコマンドは、いわゆるマス

ターレス構成を作成する場合に役立ちます。レプリカのみで構成される Patroni クラスターは、Patroni を認識しない外部マスターノードから複製します。その後、リーダーキーを削除し、Patroni ノードの 1 つを昇格させ、元のマスターを Patroni ベースの HA クラスターに置き換えることができます。

- 構成オプション `bin_dir` を追加して、PostgreSQL バイナリを見つけます。(AntsAasma)

同時に複数の PostgreSQL バージョンのインストールをサポートする Linux ディストリビューションの場合、PostgreSQL バイナリの場所を明示的に指定できると便利です。

- (AlejandroMartínez) の `custom_conf` を使用して構成ファイルのパスをオーバーライドできるようになります

Patroni、[details](#) によって管理されないカスタム構成ファイルパスを許可します。

Bug fixes and code improvements

- Patroni を PostgreSQL10 以降の新しいバージョンのスキーマと互換性を持たせるようにしました。(FeikeSteenbergen)

PostgreSQL バージョンに基づいて条件付きリスタートを実行するときに、Patroni が 2 桁のバージョン番号を理解できることを確認してください。

- `pkgutil` を使用して DCS モジュールを見つけます。(AlexanderKukushkin)

DCS モジュールを見つけるには、ディレクトリを手動でトラバースする代わりに、専用の `python` モジュールを使用します。

- Patroni を起動するときは常に `on_start` コールバックを呼び出します (AlexanderKukushkin)

以前は、Patroni は正しいロールで既に実行中のノードにアタッチするときにコールバックを呼び出しませんでした。コールバックはクライアント接続のルーティングによく使用されるため、接続ルーティングスキームに実行中のノードを登録できなくなる可能性があります。この修正により、Patroni は、既に実行中のノードにアタッチする場合でも、`on_start` コールバックを呼び出します。

- アクティブなレプリケーションスロットをドロップしないでください (MuratKabilov、OleksiiKliukin)

マスターでアクティブな物理レプリケーションスロットを削除しないでください。とにかく PostgreSQL はそのようなスロットを削除できません。この変更により、Patroni が管理していないレプリカ/コンシューマーをマスターで実行できるようになります。

- PostgreSQL インスタンスの起動中に Patroni 接続を閉じます。(AlexanderKukushkin)

PostgreSQL ノードの起動時に、Patroni に以前のすべての接続を強制的に閉じさせます。postmaster が SIGKILL で強制終了された場合、以前の接続を再利用するトラップを回避します。

- メンバー名からスロット名を構築するときに無効な文字を置き換えます。(AntsAasma)

スロットの命名規則に準拠しないスタンバイ名によってスロットの作成とスタンバイの起動が失敗しないようにしてください。スロット名のダッシュをアンダースコアに置き換え、スロット名で許可されない他のすべての文字をその Unicode コードポイントに置き換えます。

15.47 バージョン 1.0

このリリースでは、HA クラスター全体の PostgreSQL および Patroni 構成パラメーターを動的に変更できるグローバル動的構成が導入されました。また、多数のバグ修正を提供します。

Upgrade notice

v0.90 以下からアップグレードする場合は、常にマスターより前にすべてのレプリカをアップグレードしてください。レプリケーション資格情報を DCS に保存しないため、古いレプリカは新しいマスターに接続できません。

Dynamic Configuration

- 動的グローバル構成を実装する (AlexanderKukushkin)

新しい RESTAPI エンドポイント/config を導入して、HA クラスター全体（マスターとすべてのレプリカ）に対してグローバルに設定する必要がある PostgreSQL および Patroni 構成パラメーターを提供します。これらのパラメーターは DCS で設定され、多くの場合、PostgreSQL または Patroni を中断することなく適用できます。Patroni は、一部の値で PostgreSQL の再起動が必要な場合、API を介して見える/"pendingrestart/"と呼ばれる特別なフラグを設定します。その場合、API を介して手動でリスタートを発行する必要があります。

PatroniSIGHUP または POSTto/reload は、構成ファイルを再読み取りします。

変更できるパラメーターと差分構成ソースの処理順序の詳細については、[Patroni configuration](#) を参照してください。

構成ファイルの形式は、v0.90 以降 **変更されました**。Patroni は引き続き古い構成ファイルと互換性がありますが、ブートストラップパラメーターを利用するには変更する必要があります。ユーザーは、[dynamic configuration documentation page](#) を参照して更新することをお勧めします。

More flexible configuration *

- postgresql 構成とデータベース名を作成します Patroni が構成可能に接続します。(MisjaHoebe)
`database` および `config\_base\_name` 構成パラメーターを導入します。とりわけ、PipelineDB および他の PostgreSQL フォークで Patroni を実行できます。
- 環境を介していくつかの Patroni 構成パラメーターを構成する可能性を実装しました。(AlexanderKukushkin)
これらには、スコープ、ノード名、名前空間、ならびにシークレットが含まれ、動的環境、つまり Kubernetes で Patroni を実行しやすくします。詳細については、[supported environment variables](#) を参照してください。
- 組み込みの Patronidocker コンテナを更新して、環境ベースの構成を利用します (FeikeSteenbergen)。
- Patronidockerimage に Zookeeper のサポートを追加しました。(AlexanderKukushkin)
- Zookeeper と Exhibitor の構成オプションを分割しました。(AlexanderKukushkin)
- Patroni に Patroni のコードを再利用して構成を読み取らせるようにしました。(AlexanderKukushkin)

これにより、patanictl は環境ベースの構成を利用できます。

- アプリケーション名を `primary_conninfo` のノード名に設定します。(AlexanderKukushkin)

これにより、特定のノードの同期レプリケーションの識別と構成が簡素化されます。

Stability, security and usability improvements

- `sysid` をリセットし、バックアップの復元が進行中に `pg_controldata` を呼び出さないでください。(AlexanderKukushkin)

この変更により、バックアップからこのノードの長時間の初期化中に PatroniAPI ヘルスチェックによって生成されるノイズの量が削減されます。

- `pg_rewind` のコーナーケースの束を修正しました。(AlexanderKukushkin)

ソースクラスターがマスターでない場合は、`pg_rewind` の実行を避けてください。

さらに、新しいパラメーター `remove_data_directory_on_rewind_failure` が `true` に設定されていない限り、リワインドに失敗したときにデータディレクトリを削除しないでください。デフォルトでは `false` です。

- DCS のレプリケーション接続文字列からパスワードを削除します。(AlexanderKukushkin)

以前は、Patroni は常に DCS の `PostgresURL` からのレプリケーション資格情報を使用していました。これは、パトロン構成から資格情報を取得するように変更されました。シークレット (レプリケーションユーザー名とパスワード) であり、DCS で公開されなくなりました。

- 降格コールの周りの非同期機構を修正しました。(AlexanderKukushkin)

`Demote` は、DCS の相互作用をブロックすることなく、完全に非同期に実行されるようになりました。

- パトロンが設定されている場合、常に承認ヘッダーを送信するようにしました。(AlexanderKukushkin)

これにより、Patroni がそれらに対する承認を必要とするように構成されている場合、`patroni` は `/"` 保護された `/"` リクエスト、つまり再起動または再初期化を発行できます。

- `SystemExit` 例外を正しく処理します。(AlexanderKukushkin)

`SIGTERM` を受信したときに Patroni が適切に停止しない問題を回避します

- `confd` のサンプル `haproxy` テンプレート (AlexanderKukushkin)

`confide` を使用して、DCS のパトローニ状態から `haproxy` 構成を生成し、動的に変更します

- ドキュメントを改善および再構成して、新しいユーザーにとってより使いやすくしました。(LauriApple)

- API は `pg_ctlstop` 中に `role=master` を報告する必要があります。(AlexanderKukushkin)

特にクラスター停止の場合、コールバック呼び出しの信頼性を高めます。さらに、`'pg_ctl_timeout'` オプションを導入して、`'pg_ctl'` を介した開始、停止、およびリスタート呼び出しのタイムアウトを設定します。

- `etcd` の再試行ロジックを修正しました。(AlexanderKukushkin)

再試行をより予測可能で堅牢にします。

- Zookeeper コードを短いネットワークのしゃっくりに対してより回復力のあるものにしました。(AlexanderKukushkin)

接続タイムアウトを減らして、Zookeeper 接続の試行をより頻繁にします。

15.48 バージョン 0.90

このリリースでは、Consul のサポートが追加され、新しい *noloadbalance* タグが含まれ、*clonefrom* タグの動作が変更され、*pg_rewind* 処理が改善され、*patronictl* 制御プログラムが改善されます。

Consul support

- Consul サポートの実装 (AlexanderKukushkin)

Patroni は、Etcd と Zookeeper に加えて、Consul と対戦します。接続パラメーターは YAML ファイルで構成できます。

New and improved tags

- *noloadbalance* タグを実装します (AlexanderKukushkin)

このタグにより、Patroni は常にレプリカがロードバランサーで利用できないことを返します。

- *clonefrom* タグの実装を変更しました。 (AlexanderKukushkin)

以前は、ノード名を *clonefrom* に提供する必要があり、タグ付けされたレプリカに特定のノードからのクローンを強制していました。新しい実装は、*clonefrom* をブールタグにします。true に設定されている場合、レプリカは、他のレプリカがクローンを作成する候補になります。複数の候補が存在する場合、レプリカはランダムに 1 つを選択します。

Stability and security improvements

- 多数の信頼性の向上 (AlexanderKukushkin)

いくつかの偽のエラーメッセージを削除し、フェールオーバーの安定性を向上させ、DCS からのデータの読み取り、シャットダウン、降格、および前のリーダーの再接続でいくつかのコーナーケースに対処します。

- システムスクリプトを改善して、停止時に Patroni の子供たちを殺さないようにしました。 (JanKeirse, AlexanderKukushkin)

以前は、Patroni を停止すると、*systemd* も PostgreSQL にシグナルを送信しました。Patroni も PostgreSQL を自分で停止しようとしたため、さまざまなシャットダウン要求 (スマートシャットダウン、それに続く高速シャットダウン) に送信されました。その結果、レプリカの切断が早すぎ、以前のマスターは降格後に再参加できませんでした。Jan による修正、Alexander による事前調査。

- 以前のマスターがレプリカとして再参加する前に *pg_rewind* を呼び出せなかったいくつかのケースを排除しました。 (OleksiiKliukin)

以前は、以前のマスターがクラッシュした場合にのみ *pg_rewind* を呼び出しました。これを変更して、*pg_rewind* がシステムに存在する限り、以前のマスターに対して常に *pg_rewind* を実行します。これにより、レプリカが最新の変更を取得する前に ("スマート" シャットダウン中)、マスターがシャットダウンされた場合が修正されます。

- 特に、ユニットテストと受け入れテストに対する多数の改善により、Zookeeper と Consul (AlexanderKukushkin) のサポートが有効になります。
- TravisCI を高速化し、Zookeeper (出展者) と Consul (AlexanderKukushkin) に対するテスト実行のサポートを実装

ユニットテストと受け入れテストの両方が、コミットまたはプルリクエストごとに Etcd、Zookeeper、Consul に対して自動的に実行されます。

- Patroni から PostgreSQL コマンドを呼び出す前に環境変数をクリアしました。(FeikeSteenbergen)

これにより、Patroni が管理する PostgreSQL クラスターに接続して、システム環境変数を読み取る可能性を防ぎます。

Configuration and control changes

- パトロンとパトローニの構成を統一しました。(FeikeSteenbergen)

Patronictl は、Patroni 自分自身と同じ構成ファイルを使用できます。

- Patroni が環境変数から構成を読み取れるようにします。(OleksiiKliukin)

これにより、Patroni の構成の自動生成、またはさまざまなソースからの単一の構成のマージが簡素化されます。

- API によって返される情報にデータベースシステム識別子を含めます。(FeikeSteenbergen)
- 利用可能なすべての DCS に `delete_cluster` を実装しました。(AlexanderKukushkin)

パトロールで Etcd 以外の DCS のサポートを有効にします。

15.49 バージョン 0.80

このリリースでは、**カスケードレプリケーション**のサポートが追加され、**スケジュールされたフェールオーバー**を提供することにより、Patroni の管理が簡素化されます。新しいリリースに移行するために、Patroni の古いバージョン（特に 0.78）とこれを組み合わせて使用できます。スケジュールされたフェールオーバーとカスケードレプリケーション関連の機能は、Patroni0.80 以降でのみ動作することに注意してください。

Cascading replication

- パトローニード (OleksiiKliukin) の `replicatefrom` および `clonefrom` タグのサポートを追加します。

タグ `replicatefrom` を使用すると、レプリカはマスターではなく、任意のノードをソースとして使用できます。 `clonefrom` は、初期バックアップと同じことを行います。一緒に、これらは Patroni がカスケードレプリケーションを完全にサポートできるようにします。

- 実行中のレプリケーション接続がなくてもレプリカを初期化するレプリケーションメソッドを実行するためのサポートを追加しました。(OleksiiKliukin)

これは、S3 または FTP に保存されているスナップショットからレプリカを作成するために役立ちます。実行中のレプリケーション接続を必要としないレプリケーションメソッドは、yaml 構成で `no_master:true` を指定する必要があります。レプリケーション接続が存在する場合、これらのスクリプトは引き続き順に呼び出されます。

Patronictl, API and DCS improvements

- スケジュールされたフェールオーバーを実装します (FeikeSteenbergen)。

パトロンまたは API 呼び出しを使用して、将来の特定の時刻にフェールオーバーが発生するようにスケジュールできます。

- パトロンに *dbuser* と *password* パラメーターのサポートを追加しました。(FeikeSteenbergen)
- ヘルスチェックの出力に PostgreSQL バージョンを追加しました (FeikeSteenbergen)。
- パトロンの Zookeeper サポートを改善しました。(OleksandrShulgin)
- python-etcd0.43 (AlexanderKukushkin) への移行

Configuration

- Patroni (JanKeirse) のサンプルシステム構成スクリプトを追加します。
- Patroni が DB 接続の構成ファイルで指定されたスーパーユーザー名を無視する問題を修正しました (AlexanderKukushkin)。
- Patroni (AlexanderKukushkin) によって起動されたポストマスター用に別のセッション ID とプロセスグループを作成して、CTRL-C の処理を修正します。

Tests

- Patroni (AlexanderKukushkin、OleksiiKliukin) を実行する実際のシナリオを確認するために、*behave* を使用して受け入れテストを追加します。

テストは、*behave* コマンドを使用して手動で起動できます。また、プルリクエストおよびコミット後に自動的に起動されます。

一部の古いバージョンのリリースノートは、[project's github page](#) _にあります。

第16章 貢献する

開発者向けのリソースと情報は、以下のページにあります。

16.1 投稿ガイドライン

Patroni に貢献したいですか？ イェーイ-これがその方法です！

16.1.1 おしゃべり

Just want to chat with other Patroni users? Looking for interactive troubleshooting help? Join us on channel [#patroni](#) in the [PostgreSQL Slack](#).

16.1.2 テストの実行

動作テストを実行するための要件：

1. PostgreSQL パッケージをインストールする必要があります。
2. `PATH`で PostgreSQL バイナリが利用できる必要があります。
`PATH=/usr/lib/postgresql/11/bin:\$PATHpython-mBehavior`のようなものでパスにそれらを追加する必要がある場合があります。
3. 外部 DCS (Etd、Consul、Zookeeper など) でテストする場合は、パッケージをインストールし、それぞれのサービスを実行し、ローカルホストとデフォルトポートで暗号化/保護されていない接続を受け入れる必要があります。Etd または Consul の場合、`PATH`でバイナリが利用可能な場合、動作テストスイートはそれらを起動できます。

依存関係をインストールします。

```
# You may want to use Virtualenv or specify pip3.  
pip install -r requirements.txt  
pip install -r requirements.dev.txt
```

すべての依存関係をインストールしたら、さまざまなテストスイートを実行できます。

```
# You may want to use Virtualenv or specify python3.

# Run flake8 to check syntax and formatting:
python setup.py flake8

# Run the pytest suite in tests/:
python setup.py test

# Run the behave (https://behave.readthedocs.io/en/latest/) test suite in features/;
# modify DCS as desired (raft has no dependencies so is the easiest to start with):
DCS=raft python -m behave
```

16.1.3 tox でのテスト

tox テストを実行するには、1 つの依存関係（Python 以外）をインストールするだけです

```
pip install tox>=4
```

`behave` テストを実行する場合は、docker もインストールする必要があります。

`tox.ini` の Tox 構成には、次のタスクを実行するための `environments/` があります。

- lint: `flake8` を使用した Python コード lint
- test: `pytest` を使用した利用可能なすべての Python インタープリターの単体テスト、TTY が検出された場合は XML レポートまたは HTML レポートを生成
- dep: `pipdeptree` を使用してパッケージの依存関係の競合を検出する
- type: `pyright` による静的型チェック
- black: `black` を使用したコードフォーマット
- docker-build: `behave` 環境に使用される docker イメージをビルドします
- docker-cmd: 上記の画像で任意のコマンドを実行
- docker-behave-etcd: 上記の画像で動作テストのために tox を実行します
- py*behave: 使用可能な python インタープリターで動作します (docker なし、ただしこれは docker コンテナ内で呼び出されます)
- docs: `sphinx` でドキュメントをビルドする

ランニングトックス

デフォルトの env リストを実行するには;dep、lint、test、および docs、実行するだけです：

```
tox
```

test 環境は、ラベル test で実行できます。

```
tox -m test
```

behave ドッカーテストは、ラベル behave で実行できます。

```
tox -m behave
```

同様に、docs にはラベル docs があります。

他のすべての env は、それぞれの env 名で実行できます。

```
tox -e lint
tox -e py39-test-lin
```

factors を使用して、部分的な env リストを選択することもできます。たとえば、python3.10 のすべての env を実行する場合：

```
tox -f py310
```

これは、以下にリストされているすべての env を実行することと同等です。

```
$ tox -l -f py310
py310-test-lin
py310-test-mac
py310-test-win
py310-type-lin
py310-type-mac
py310-type-win
py310-behave-etcd-lin
py310-behave-etcd-win
py310-behave-etcd-mac
```

次のように、tox (>=v4) を使用して環境の構成済みの組み合わせをすべてリストできます

```
tox l
```

tox がアクティブな端末で実行されている場合、環境 test および docs は、ジョブの完了時に HTML 出力 ファイルを開こうとします。これは、この環境をローカルで実行している開発者の利益を目的としています。Mac では open、Linux では xdg-open を実行しようとしています。別のコマンドを使用するには、環境変数 OPEN_CMD をコマンドの名前またはパスに設定します。このステップが失敗しても、実行全体は失敗しません。この機能を無効にする場合は、環境変数 OPEN_CMD を no-op コマンドに設定します。

```
OPEN_CMD=: tox -m docs
```

動作テスト

、`-mBehavior`を使用した動作テストは、PG_MAJOR バージョン 11~15 に基づいて docker イメージをビルドし、すべての動作テストを実行します。これは実行にかなり時間がかかる場合があるため、スコープを選択したバージョンの Postgres、または特定の機能セットまたはステップに制限することをお勧めします。

postgres のバージョンを指定するには、必要な依存イメージビルド環境のフルネームと動作環境名を含めます。たとえば、Postgres15 が必要な場合：

```
tox -e pg14-docker-build,pg14-docker-behave-etcd-lin
```

一方、特定の機能をテストしたい場合は、動作に位置引数を渡すことができます。これにより、すべてのバージョンの Postgres でウォッチドッグの動作機能テストシナリオが実行されます。

```
tox -m behave -- features/watchdog.feature
```

もちろん、2つを組み合わせることもできます。

16.1.4 問題の報告

パトニについて質問がある場合、または使用中に問題がある場合は、問題を提出する前に [README](#) をお読みください。また、[README](#) _の現在の問題を再確認してください。

16.1.5 プルリクエストを貢献する

- 1) 関連する問題にコメントを送信するか、提案された変更を説明する新しい問題を作成します。
- 2) フォークを行い、コード変更を開発およびテストします。
- 3) ドキュメントを含める
- 4) プルリクエストを送信します。

プルリクエストに関するフィードバックができるだけ早く届きます。

ハッピーパトローニハッキング;-)