
Patroni

リリース 3.2.2

Zalando SE

2024 年 01 月 22 日

コンテンツ：

第 1 章 導入	3
1.1 開発状況	3
1.2 技術的要件/インストール	3
1.3 PostgreSQL ノードの数の計画	3
1.4 実行と構成	4
1.5 YAML 構成	4
1.6 環境構成	4
1.7 レプリケーションの選択肢	4
1.8 アプリケーションはスーパーユーザーを使用すべきではありません	5
1.9 HA ソリューションのテスト	5
第 2 章 インストール	7
2.1 MacOS の前提条件	7
2.2 サイコプグ	7
2.3 pip の一般的なインストール	7
2.4 Linux へのパッケージのインストール	8
2.5 アップグレード中	10
第 3 章 パトローニの構成	11
3.1 動的構成設定	11
3.2 YAML 構成設定	15
3.3 環境構成の設定	30
3.4 重要なルール	42
3.5 構成の生成と検証	45
第 4 章 パトローニ RESTAPI	49
4.1 ヘルスチェックエンドポイント	49
4.2 監視エンドポイント	51
4.3 クラスターステータスエンドポイント	57
4.4 構成エンドポイント	59
4.5 スイッチオーバーおよびフェイルオーバーのエンドポイント	62
4.6 エンドポイントを再起動します	64
4.7 エンドポイントのリロード	65
4.8 エンドポイントを再初期化する	65
第 5 章 Patroni	67
5.1 構成	67
5.2 使用法	68
第 6 章 レプリカのイメージングとブートストラップ	107
6.1 ブートストラップ	107

6.2	レプリカの構築	109
6.3	スタンバイクラスタ	112
第7章	レプリケーションモード	113
7.1	非同期モードの耐久性	113
7.2	PostgreSQL の同期レプリケーション	113
7.3	同期モード	114
7.4	同期レプリケーション係数	115
7.5	同期モードの実装	115
第8章	Watchdog のサポート	117
8.1	Linux でのソフトウェア Watchdog のセットアップ	118
第9章	クラスターの一時停止/再開モード	119
9.1	目標	119
9.2	実装	119
9.3	ユーザーガイド	120
第10章	DCS フェールセーフモード	121
10.1	問題	121
10.2	現在の実装の理由	121
10.3	DCS フェールセーフモード	121
10.4	低レベルの実装の詳細	122
10.5	よくある質問。	122
第11章	Kubernetes での Patroni の使用	125
11.1	エンドポイントの使用	125
11.2	ConfigMap を使用する	125
11.3	構成	125
11.4	例	127
第12章	Citus のサポート	129
12.1	TL;DR	129
12.2	Patroni	130
12.3	Citus ワーカーの切り替え	131
12.4	DCS を覗いてみる	133
12.5	Kubernetes 上の Citus	133
12.6	Citus のアップグレードと PostgreSQL のメジャーアップグレード	136
第13章	スタンドアロンを Patroni クラスターに変換する	137
13.1	手順	137
第14章	PostgreSQL バージョンのメジャーアップグレード	141
14.1	よくある質問	141
第15章	セキュリティに関する考慮事項	143
15.1	DCS の保護	143
15.2	RESTAPI の保護	143
第16章	HA マルチデータセンター	145

16.1	同期レプリケーション	145
16.2	非同期レプリケーション	146
第 17 章	よくある質問	147
17.1	他の HA ソリューションとの比較	147
17.2	DCS	148
17.3	Patroni	149
17.4	構成	149
17.5	Postgres 管理	152
17.6	概念と要件	153
17.7	自動フェイルオーバー	154
17.8	ブートストラップとスタンバイの作成	155
17.9	監視	155
第 18 章	リリースノート	157
18.1	バージョン 3.2.2	157
18.2	バージョン 3.2.1	158
18.3	バージョン 3.2.0	159
18.4	バージョン 3.1.2	160
18.5	バージョン 3.1.1	161
18.6	バージョン 3.1.0	162
18.7	バージョン 3.0.4	164
18.8	バージョン 3.0.3	165
18.9	バージョン 3.0.2	166
18.10	バージョン 3.0.1	167
18.11	バージョン 3.0.0	167
18.12	バージョン 2.1.7	169
18.13	バージョン 2.1.6	169
18.14	バージョン 2.1.5	170
18.15	バージョン 2.1.4	172
18.16	バージョン 2.1.3	173
18.17	バージョン 2.1.2	175
18.18	バージョン 2.1.1	177
18.19	バージョン 2.1.0	178
18.20	バージョン 2.0.2	180
18.21	バージョン 2.0.1	182
18.22	バージョン 2.0.0	183
18.23	バージョン 1.6.5	189
18.24	バージョン 1.6.4	191
18.25	バージョン 1.6.3	193
18.26	バージョン 1.6.2	193
18.27	バージョン 1.6.1	194
18.28	バージョン 1.6.0	197
18.29	バージョン 1.5.6	200
18.30	バージョン 1.5.5	201
18.31	バージョン 1.5.4	202
18.32	バージョン 1.5.3	203

18.33 バージョン 1.5.2	204
18.34 バージョン 1.5.1	204
18.35 バージョン 1.5.0	205
18.36 バージョン 1.4.6	206
18.37 バージョン 1.4.5	206
18.38 バージョン 1.4.4	208
18.39 バージョン 1.4.3	209
18.40 バージョン 1.4.2	210
18.41 バージョン 1.4.1	211
18.42 バージョン 1.4	211
18.43 バージョン 1.3.6	213
18.44 バージョン 1.3.5	214
18.45 バージョン 1.3.4	215
18.46 バージョン 1.3.3	216
18.47 バージョン 1.3.2	216
18.48 バージョン 1.3.1	216
18.49 バージョン 1.3	216
18.50 バージョン 1.2	219
18.51 バージョン 1.1	222
18.52 バージョン 1.0	224
18.53 バージョン 0.90	226
18.54 バージョン 0.80	228
第 19 章 貢献する	231
19.1 ガイドラインへの貢献	231
第 20 章 インデックスとテーブル	235

Patroni is a template for high availability (HA) PostgreSQL solutions using Python. For maximum accessibility, Patroni supports a variety of distributed configuration stores like [ZooKeeper](#), [etcd](#), [Consul](#) or [Kubernetes](#). Database engineers, DBAs, DevOps engineers, and SREs who are looking to quickly deploy HA PostgreSQL in datacenters — or anywhere else — will hopefully find it useful.

Patroni は、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とは程遠いため、私たちは Patroni を「テンプレート」と呼んでいます。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 Documentation](#) 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:9.3～16。

Note to Citus users :3.0 以降、Patroni は Postgres の **Note to Citus users** 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する方法の詳細については、Patroni ドキュメントの **Note to Citus users** を確認してください。

Note to Kubernetes users :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 **Note to Kubernetes users** の章を見てください。

第1章 導入

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 のプロジェクトである [Governor](#) _ のフォークとして誕生しました。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 の講演 (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。

新しいリリース情報 [here](#) を報告します。

1.2 技術的要件/インストール

さまざまなプラットフォームでの Patroni のインストールとアップグレードに関するガイダンスについては、[here](#) にアクセスしてください。

1.3 PostgreSQL ノードの数の計画

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)、そのためノードの最小数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まったく問題ありません。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> から複製されたものと想定しています。つまり、サンプル設定ファイル `postgres0.yml` と `postgres1.yml` が必要になります。pip を使用して Patroni をインストールした場合は、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。

開始するには、別の端末から次の操作を実行します:

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

高可用性クラスターが起動するのが表示されます。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。いくつかのコンポーネントを強制終了して、システムがどのように動作するかを確認します。

`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、アプリケーションに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を提供する `HAProxy` 構成を提供します。設定するには、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

環境変数による設定の構成 (オーバーライド) に関する包括的な情報については、[here](#) にアクセスしてください。

1.7 レプリケーションの選択肢

Patroni は Postgres のストリーミングレプリケーションを使用します。これはデフォルトで非同期です。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より一定のバイト数を超えて遅れている場合、フェイルオーバーは発生しません。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。耐久性の保証を向上させ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。詳細については、`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すべきではありません

アプリケーションから接続する場合は、必ず非スーパーユーザーを使用してください。Patroni が正しく機能するには、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と、`superuser_reserved_connections` 設定を使用して、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であるために Patroni がプライマリにアクセスできない場合、動作は望ましくないものになります。

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、多くの変数を伴う、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慮す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。これについては、ドキュメントで詳しく説明することはできません。

そうは言っても、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 (システムの前面のネットワークおよび NIC(物理または仮想) 自体)
- ディスク IO
- ファイル制限 (Linux では `nofile`)
- ラム。oomkiller をオフにしている場合でも、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 (上記に関連する可能性があります)
- 任意の 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適切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、ポストマスタープロセスで `kill -9` を実行することです。これは、そうすることは現実のシナリオを模倣するものではないためです。インフラストラクチャが安全ではなく、攻撃者が `kill -9` を実行する可能性がある懸念している場合、HA プロセスをどれだけ実行しても問題は解決されません。攻撃者は単にプロセスを再度強制終了するか、別の方法で混乱を引き起こします。

第2章 インストール

2.1 MacOS の前提条件

Mac に要件をインストールするには、次のコマンドを実行します。

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

2.2 サイコプグ

psycopg2-2.8`以降、バイナリバージョンの *psycopg2* はデフォルトで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 コンパイラーと *postgres+pythondev* パッケージが必要です。Python の世界では依存関係を `psycopg2 OR psycopg2-binary` として指定することはできないため、それをインストールする方法を決定する必要があります。

いくつかのオプションが利用可能です:

1. ディストリビューションの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
```

2. pip で Patroni をインストールする場合は、*list of dependencies* に `psycopg`、*psycopg2*、または `psycopg2-binary` のいずれかを指定します。

2.3 pip の一般的なインストール

Patroni は pip でインストールできます。

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

ここで、dependencies は空であるか、次の 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

ZooKeeper

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

エキシビター

Exhibitionr 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)

Kubernetes

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ための `boto3`

jsonlogger

json 形式で *logging* を有効にするための `python-json-logger` モジュール

全て

上記のすべて (psycopg ファミリを除く)

サイココップ

`psycopg[binary]>=3.0.0` モジュール

サイココップ 2

`psycopg2>=2.5.4` モジュール

psycopg2-バイナリ

「psycopg2-binary」モジュール

たとえば、Patroni を psycopg3、DCS としての Etc3 の依存関係、および AWS コールバックとともに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[psycopg3,etcd3,aws]
```

レプリカ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。

2.4 Linux へのパッケージのインストール

Patroni パッケージは、次の目的で Postgres コミュニティによって作成され、オペレーティングシステムで利用できる場合があります。

- RHEL、RockyLinux、AlmaLinux。
- Debian と Ubuntu。
- SUSE エンタープライズ Linux。

公式オペレーティングシステムリポジトリでは利用できない可能性がある Python モジュールなど、Patroni の直接依存関係のパッケージも見つけることができます。

詳細については、`PGDG` リポジトリ `ドキュメント` を参照してください。

RedHatEnterpriseLinux 派生オペレーティングシステムを使用している場合は、EPEL のパッケージも必要になる場合があります。「EPEL リポジトリ」のドキュメントを参照してください。

OS の PGDG リポジトリをインストールしたら、patroni をインストールできます。

注釈: Patroni パッケージは Patroni 開発者ではなく、Postgres コミュニティによって保守されています。サポートが必要な場合は、まず「PostgresSlack」_に接続してみてください。

2.4.1 Debian 派生製品へのインストール

PGD   G リポジトリがインストールされたら、[above](#) を参照し、`aptrun` で Patroni をインストールします。

```
apt-get install patroni
```

2.4.2 RedHat 派生製品へのインストール

PGD   G リポジトリがインストールされたら、[above](#) を参照し、RHEL9(および派生製品) 上で `dnf` 経由で `etcdDCS` を含む `patroni` をインストールし、以下を実行します。

```
dnf install patroni patroni-etcd
```

RedHat 派生ディストリビューションがパッケージを提供していない場合は、PGDG から `etcd` をインストールできます。DCS をホストするノード上で以下を実行します。

```
dnf install 'dnf-command(config-manager)'  
dnf config-manager --enable pgdg-rhel9-extras  
dnf install etcd
```

必要に応じて、リポジトリ内の RHEL のバージョンを「8」に置き換えて「`pgdg-rhel8-extras`」を作成できます。RockyLinux、AlmaLinux、OracleLinux などではリポジ名は依然として「`pgdg-rhelN-extras`」です。

2.4.3 SUSEEnterpriseLinux へのインストール

一部の依存関係については、SUSEPackageHub リポジトリを有効にする必要がある場合があります。「SUSEPackageHub」ドキュメントを参照してください。

PGD   G リポジトリがインストールされている SLES15 については、[above](#) を参照してください。以下を使用して `patroni` をインストールできます。

```
zypper install patroni patroni-etcd
```

SUSEPackageHub リポジトリを有効にすると、`etcd` をインストールすることもできます。

```
SUSEConnect -p PackageHub/15.5/x86_64  
zypper install etcd
```

2.5 アップグレード中

patroni のアップグレードは非常に簡単なプロセスで、ソフトウェアのインストールを更新し、クラスター内の各ノードで Patroni デーモンを再起動するだけです。

ただし、Patroni デーモンを再起動すると、Postgres データベースが再起動されます。状況によっては、これによりクラスター内のプライマリノードのフェイルオーバーが発生する可能性があるため、Patroni デーモンの再起動が完了するまでクラスターをメンテナンスモードにすることをお勧めします。

クラスターをメンテナンスモードにするには、patroni ノードの 1 つで次のコマンドを実行します。

```
patronictl pause --wait
```

次に、クラスター内の各ノードで、OS に必要なパッケージのアップグレードを実行します。

```
apt-get update && apt-get install patroni patroni-etcd
```

各ノードで patroni デーモンプロセスを再起動します。

```
systemctl restart patroni
```

次に、最後に patroni を使用して Postgres の監視を再開し、メンテナンスモードを解除します。

```
patronictl resume --wait
```

これで、クラスターは新しいバージョンの Patroni で完全に動作できるようになります。

第3章 パトローニの構成

3.1 動的構成設定

動的構成は 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。

動的構成を変更するには、`patronictl 編集構成` ツールまたは Patroni `patronictl 編集構成` 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、可能な最小値:1
- **ttl** : リーダーロックを取得するための TTL(秒単位)。これは、自動フェイ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、可能な最小値:20
- **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って、パトローニ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、可能な最小値:3

警告: `loop/_wait`、`loop/_wait`、または `loop/_wait` の値を変更する場合は、次のルールに従う必要があります。

```
loop_wait + 2 * retry_timeout <= ttl
```

- **maximum/_lag/_on/_failover** : フォロワーがリーダーの選出に参加できるようになるまでに遅れる可能性のある最大バイト数。
- **maximum/_lag/_on/_syncnode** : 同期フォロワーが異常な候補とみなされ、健全な非同期フォロワーによってスワップされるまでに、同期フォロワーが遅れる可能性がある最大バイト数。Patroni は、複数のフォロワーがいる場合は最大レプリカ lsn を使用します。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、値が 0 以下に設定されている場合、Patroni は同期の不健全なフォロワーを交換するアクションを実行しません。トランザクション量が多いときに Patroni が同期フォロワーを頻繁にスワップしないように、十分大きな値を設定してください。
- **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 に完全な履歴が保存されます。
- **primary/_start/_timeout** : フェイ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 (秒単位)。デフォルトは 300 秒です。0 に設定すると、可能であればクラッシュが検出された直後にフェイ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、フェイ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場合のフェイルオーバー時間は、`loop/_wait+Primary/_start/_timeout+loop/_wait` です。ただし、`primary/_start/_timeout` が 0 の場合は除きます。その場合は、単なる `loop/_wait` です。耐久性と可用性のトレードオフに応じて値を設定します。

- **primary/_stop/_timeout** :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 に設定され、synchronous_mode が有効な場合、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、Patroni はポストマスターに SIGKILL を送信します。耐久性と可用性のトレードオフに応じて値を設定します。パラメーターが設定されていないか、0 以下に設定されている場合、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の選出に参加できます。同期モードでは、Patroni がトランザクションの耐久性を確保できない場合、書き込みの可用性が失われるという代償を払って、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ることはありません。詳細については、**synchronous/_mode** を参照してください。
- **synchronous/_mode/_strict** : 使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を無効にせず、プライマリへのすべてのクライアント書き込みをブロックします。詳細については、**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 は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 はこのパラメータを無視します。
 - * -すべてをホスト 0.0.0.0/md5
 - * -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次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 はこのパラメータを無視します。
 - * -マップ名 1 システム名 1pguser1
 - * -マップ名 1 システム名 2pguser2
- **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のスロット。このパラメータはオプションであり、デフォルト値はインスタンス名から導出されます (関数 `'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きリスト。 **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** : WAL レコードを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復元するコマンド。 **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実際に WAL レコードを適用するまでの待機時間
- **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/フェイルオーバー中に保持されます。存在しない永久スロットはパトローニによって作成されます。PostgreSQL11 以降では、永続的な物理スロットがすべてのノードに作成され、その位置は **slots** 秒ごとに進められます。11 よりも古いバージョンの PostgreSQL の場合、永続的な物理レプリケーション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よってプライマリからスタンバイにコピーされ、その後 (必要に応じて) **slots** 秒ごとに位置が進みます。論理スロットファイルのコピーは、`libpq` 接続経由で、`rewind` 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ます (**slots** セクションを参照)。レプリカ上の論理スロットの位置が以前のプライマリより少し遅れている可能性が常にあります。そのため、アプリケーションは、フェイルオーバー後の 2 回目に一部のメッセージを受信できるように準備する必要があります。これを行う最も簡単な方法は、`confirmed_flush_lsn` を追跡することです。永久レプリケーションスロットを有効にするには、**slots** を `true` に設定する必要があります。永続的な論理レプリケーションスロットが定義されている場合、Patroni は自動的に `hot_standby_feedback` を有効にします。PostgreSQL9.6 以前では論理レプリケーションスロットのフェイルオーバーが安全でなく、PostgreSQL バージョン 10 にはいくつかの重要な機能が欠けているため、この機能は PostgreSQL11 以降でのみ動作します。
- **my/_slot/_name** : 永続レプリケーションスロットの名前。永続スロット名が現在のノードの名前と一致する場合、そのスロットはこのノードには作成されません。Patroni メンバーの名前と一致する名前の永続的な物理レプリケーションスロットを追加すると、Patroni は、対応する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ようにします。この状況では、通常は Patroni によってスロットが削除されます。これは、一時的な障害時にメンバーが使用するレプリケーションスロットを維持したい場合や、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 (詳細については **my/_slot/_name** を参照) など、状況によっては便利ですが、オペレーターはこれらのことが発生することに注意する必要があります。名前の衝突は、スロットが不要になった場合、Patroni の通常の機能に影響を与えるため、DCS には保持されません。
 - * **type** : スロットのタイプ。 `physical` または `logical` の可能性があります。スロットが論理スロットの場合は、 `database` および `plugin` を追加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

ている場合に便利です。一致するプロパティのサブセットがあると、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**: スロットのタイプ。 **physical** または **logical** を指定できます。スロットが論理スロットの場合は、 **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すが、 **slots** は配列です。例えば:

```
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
  ...
```

注: クラスタートポロジが静的 (名前が変更されない固定数のノード) の場合は、レプリカが一時的に停止している間に WAL ファイルのリサイクルを回避するために、ノードの名前に対応する名前を使用して永続的な物理レプリケーションスロットを構成できます。

```
slots:
  node_name1:
    type: physical
  node_name2:
    type: physical
  node_name3:
    type: physical
  ...
```

警告: 永続レプリケーションスロットは、 **primary / standby_leader** からレプリカノードにのみ同期されます。つまり、アプリケーションはリーダーノードからのみそれらを使用することになっています。これらをレプリカノードで使用すると、クラスター内の他のすべてのノードで **pg_wal** が無制限に増加します。この規則の例外は、Patroni メンバー名と一致する永続的な物理スロット (構成している場合) です。これらはノード間のレプリケーションに使用されるため、すべてのノード間で同期

されます。

3.2 YAML 構成設定

3.2.1 グローバル/ユニバーサル

- **name** : ホストの名前。クラスター内で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

3.2.2 ログ

- **type** : ログの形式を設定します。 **type** または **type** のいずれかになります。 **type** 形式を使用するには、 **type** がインストールされている必要があります。デフォルト値は **type** です。
- **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表示したい場合は、 **traceback/_level** に設定します。
- **format** : ログのフォーマット文字列を設定します。ログタイプが **format** の場合、ログ形式は文字列である必要があります。使用可能な属性については、 **format** を参照してください。ログタイプが **format** の場合、ログ形式は文字列に加えてリストにすることもできます。各リスト項目は LogRecord 属性に対応する必要があります。フィールド名のみが必要であり、 **%**(および) は省略する必要がありますことに注意してください。別のキー名でログフィールドを出力する場合は、辞書キーがログフィールドで、値がログに出力するフィールドの名前である辞書を使用します。デフォルト値は **%(asctime)s%(levelname)s: %(message)s** です。
- **dateformat** : 日時フォーマット文字列を設定します。 (**dateformat** を参照)
- **static_fields** : ログにフィールドを追加します。このオプションは、ログタイプが **static_fields** に設定されている場合にのみ使用できます。
- **max/_queue/_size** : Patroni は 2 段階のロギングを使用しています。ログレコードはメモリ内のキューに書き込まれ、キューからログレコードを取得して 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ており、過去 1 時間 20 分までのログを保持するには十分なサイズです。
- **dir** : アプリケーションログを書き込むディレクトリ。ディレクトリは存在し、Patroni を実行す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'` と `'file_size'` を使用して調整できます (以下を参照)。

- **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 `patroni.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**

ここでは、JSON 形式でログを記録するように `patroni` を設定する方法の例を示します。

```
log:
  type: json
  format:
    - message
    - module
    - asctime: '@timestamp'
    - levelname: level
  static_fields:
    app: patroni
```

3.2.3 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変更はすべて反映されなくなります。それらを変更したい場合は、`bootstrap.dcs` 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、新しいクラスターの初期化後に、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` の下に置くことができ、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。

詳細については、[custom bootstrap methods documentation](#) を参照してください。 `initdb` が指定されている場合は、デフォルトの `initdb` コマンドに戻ります。 `initdb` は、構成ファイルに `method` パラメータ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のリスト。
 - * **-data-checksums** : 9.3 で `pg_rewrite` が必要な場合は有効にする必要があります。
 - * **-エンコーディング: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のロケール。

- **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(クラスタースーパーユーザーをユーザー名として含む) 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。

3.2.4 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 などを使用します。
- **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。現在サポートされているデータベースは 1 つだけです。

3.2.5 Consul

ほとんどのパラメータ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ホスト: ポート。
- **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** です。
- **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** の一部である場合は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 整合性モードを選択します。可能な値は **default**、**consistent**、または **stale** です (詳細は **consistency** を参照)
- **checks**: (オプション) セッションに使用される 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) ノードの役割に応じて、スコープパラメーターとタグマスター、プライマリ、レプリカ、またはスタンバイリーダーによって定義された名前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"
}
```

3.2.6 Etcd

ほとんどのパラメータ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 エンドポイントのホスト: ポート。
- **hosts** : host1:port1, host2:port2, etc... の形式の etcd エンドポイントのリスト。カ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 に接続している場合は、**proxy** の代わりにこのパラメータを使用します。
- **srv** : クラスター自動検出のために 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に対して SRV サービス名 (最初に成功するまでこの順序で) のクエリを試行します: **_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は、ETCD ピアプロトコルが使用され、ETCD で利用可能なメンバーをクエリします。それ以外の場合は、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、同じドメイン内の複数の etcd クラスターを区別します。 **srv/_suffix** と組み合わせた場合にのみ機能します。たとえば、**srv_suffix**: **foo** と **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: **_etcd-client-ssl-foo._tcp.example.com** (考えられるすべての ETCD SRV サービス名に対して同様)。

- **protocol** : (オプション) http または https (指定しない場合は http が使用されます)。 **protocol** または **protocol** が指定されている場合、それらからプロトコルを取得します。
- **username** : (オプション) etcd 認証用のユーザー名。
- **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** の一部である場合は空にすることができます。

3.2.7 etcdv3

Patroni をプロトコルバージョン 3 経由で Etcd クラスターと連携させたい場合は、Patroni 設定ファイルの `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は `etcd` の場合と同じです。

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 では表示されず、その逆も同様であるため、Patroni 構成ファイルを更新するだけでは `etcd` から `etcd3` に切り替えることはできません。

3.2.8 ZooKeeper

- **hosts** : ZooKeeper クラスターメンバーのリスト (形式: ['host1:port1', 'host2:port2', 'etc...'])。
- **use_ssl** : (オプション) SSL が使用されるかどうか。デフォルトは `false` です。 `false` に設定すると、SSL 固有のパラメータ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) 設定した場合、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」スキーマを想定しており、プリンシパルをキーとして、1 つ以上のアクセス許可を値のリストとして持つディクショナリとして指定する必要があります。権限は、`CREATE`、`READ`、`WRITE`、`DELETE` または `ADMIN` のいずれかになります。たとえば、`set_acls: {CN=principal1: [CREATE, READ], CN=principal2: [ALL]}`。
- **auth_data** : (オプション) 接続に使用する認証資格情報。`scheme` がキー、`credential` が値である形式の辞書である必要があります。デフォルトは空の辞書です。

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。

3.2.9 エキシビター

- **hosts**: エキシビター (ZooKeeper) ノードの初期リスト (形式: 「host1、host2、etc...」)。このリストは、Exhibitionr(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 Exhibitionr ノードのリストを Exhibitionr から更新する頻度。
- **port**: エキシビターポート。

3.2.10 Kubernetes

- **bypass/_api/_service**: (オプション) KubernetesAPI と通信する場合、Patroni は通常 `kubernetes` サービスに依存し、そのアドレスは `KUBERNETES\_SERVICE\_HOST` 環境変数を介してポッド内で公開されます。「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「デフォルト」です。
- **labels**: {label1: value1, label2: value2} 形式のラベル。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 (ポッドとエンドポイントまたは ConfigMap) を検索す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 (エンドポイントまたは ConfigMap) にそれらを設定します。
- **scope/_label**: (オプション) クラスター名を含むラベルの名前。デフォルト値は「クラスタ名」です。
- **role/_label**: (オプション) ロール (マスター、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、このラベルを実行するポッドに設定します。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 `master` 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。このラベルの値には、対応するロールのデフォルトが常に使用されます。必要な場合のみ設定してください。
- **use/_endpoints**: (オプション) `true` に設定すると、Patroni は ConfigMap の代わりにエンドポイントを使用してリーダーの選出を実行し、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 が昇格され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)Service 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、サービスは機能しません。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ます。これにより、Patroni はリーダーエンドポイントのサブセットを更新するためにそれを使用します。このパラメータは、「`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。指定されない場合、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。
- **retriable/_http/_codes** :(オプション)再試行する 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、または K8sAPI 応答に `retry-after` HTTP ヘッダーがある場合に再試行します。

3.2.11 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。
- **bind/_addr** :(オプション)Raft 接続をリッスンする `ip:port`。指定しない場合は、 `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。指定しない場合は、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 の実装に関する短い 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 1 つのアドレスです。
- Q: コンセンサスに含まれていたノードが消滅したため、同じ IP を他のノードに再利用できません。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 の依存関係である `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 設定から `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 番目のノードでは `patroni_raft_controller` を実行できます (Patroni と PostgreSQL なし)。このような設定では、プライマリに影響を与えることなく、一時的に 1 つのノードを失う可能性があります。

3.2.12 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 Postgres に接続するために Patroni によって使用されます。
 - `password` : スーパーユーザーのパスワード。初期化 (initdb) 中に設定されます。
 - `sslmode` :(オプション) `sslmode` _接続パラメータにマップします。これにより、クライアントはサーバーとの TLS ネゴシエーションモードのタイプを指定できます。各モードの動作の詳細については、`sslmode` _を参照してください。デフォルトのモードは `prefer` です。
 - `sslkey` :(オプション) `sslkey` _接続パラメータにマップします。これは、クライアントの証明書で使用する秘密キーの場所を指定します。
 - `sslpassword` :(オプション) `sslpassword` _接続パラメータにマップします。これは、`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) `sslcert` _接続パラメータにマップされ、クライアント証明書の場所を指定します。
 - `sslrootcert` :(オプション) `sslrootcert` _接続パラメータにマップします。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。
 - `sslcrld` :(オプション) `sslcrld` _接続パラメータにマップします。これは、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、このリストに存在する証明書を持つサーバーへの接続を拒否します。
 - `sslcrldir` :(オプション) `sslcrldir` _接続パラメータにマップします。これは、証明書失効リストを含むファイルが含まれるディレクトリの場所を指定します。クライアントは、このリストに存在する証明書を持つサーバーへの接続を拒否します。
 - `gssencmode` :(オプション) `gssencmode` _接続パラメータにマップします。これにより、セキュアな GSSTCP/IP 接続がサーバとネゴシエートされるかどうか、またはどのような優先順位でネゴシエートされるかを決定します。
 - `channel_binding` :(オプション) `channel_binding` _接続パラメータにマップします。これは、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。

- * `replication` :

- ・ **username** : レプリケーションのユーザー名。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経由で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** _接続パラメータにマップします。これにより、クライアントはサーバーとの TLS ネゴシエーションモードのタイプを指定できます。各モードの動作の詳細については、 **sslmode** _を参照してください。デフォルトのモードは **prefer** です。
- ・ **sslkey** :(オプション) **sslkey** _接続パラメータにマップします。これは、クライアントの証明書で使用する秘密キーの場所を指定します。
- ・ **sslpassword** :(オプション) **sslpassword** _接続パラメータにマップします。これは、**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) **sslcert** _接続パラメータにマップされ、クライアント証明書の場所を指定します。
- ・ **sslrootcert** :(オプション) **sslrootcert** _接続パラメータにマップします。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。
- ・ **sslerl** :(オプション) **sslerl** _接続パラメータにマップします。これは、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、このリストに存在する証明書を持つサーバーへの接続を拒否します。
- ・ **sslerldir** :(オプション) **sslerldir** _接続パラメータにマップします。これは、証明書失効リストを含むファイルが含まれるディレクトリの場所を指定します。クライアントは、このリストに存在する証明書を持つサーバーへの接続を拒否します。
- ・ **gssencmode** :(オプション) **gssencmode** _接続パラメータにマップします。これにより、セキュアな GSSTCP/IP 接続がサーバとネゴシエートされるかどうか、またはどのような優先順位でネゴシエートされるかを決定します。
- ・ **channel_binding** :(オプション) **channel_binding** _接続パラメータにマップします。これは、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 以降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** _接続パラメータにマップします。これにより、クライアントはサーバーとの TLS ネゴシエーションモードのタイプを指定できます。各モードの動作の詳細については、 **sslmode** _を参照してください。デフォルトのモードは **prefer** です。
- ・ **sslkey** :(オプション) **sslkey** _接続パラメータにマップします。これは、クライアントの証明書で使用する秘密キーの場所を指定します。

- ・ **sslpassword** :(オプション) **sslpassword** _接続パラメータにマップします。これは、**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) **sslcert** _接続パラメータにマップされ、クライアント証明書の場所を指定します。
 - ・ **sslrootcert** :(オプション) **sslrootcert** _接続パラメータにマップします。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** _接続パラメータにマップします。これは、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、このリストに存在する証明書を持つサーバーへの接続を拒否します。
 - ・ **sslcrlidir** :(オプション) **sslcrlidir** _接続パラメータにマップします。これは、証明書失効リストを含むファイルが含まれるディレクトリの場所を指定します。クライアントは、このリストに存在する証明書を持つサーバーへの接続を拒否します。
 - ・ **gssencmode** :(オプション) **gssencmode** _接続パラメータにマップします。これにより、セキュアな GSSTCP/IP 接続がサーバとネゴシエートされるかどうか、またはどのような優先順位でネゴシエートされるかを決定します。
 - ・ **channel_binding** :(オプション) **channel_binding** _接続パラメータにマップします。これは、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。
- **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。(記述方法の例として `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 (ロールを変更せずに) このスクリプトを実行します。
 - * **on/_role/_change** : postgres が昇格または降格され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** : 他のノードやアプリケーションから Postgres にアクセスできる IP アドレス+ポート。
 - **proxy/_address** : Postgres の隣で実行されている接続プール (pgbouncer など) 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または役立つ可能性があります。
 - **create/_replica/_methods** : Patroni 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アイテム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です。Patroni によって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`)。指定されない場合、または空の文字列である場合は、`PATH` 環境変数を使用して実行可能ファイルが検索されます。
- **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最後のアドレスの後にコロン付きで追加されている限り、カンマで区切られた複数のアドレスが許可されます (`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。
- **use/_unix/_socket** : Patroni が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 はその中の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、接続パラメータから `host` を省略します。
- **use/_unix/_socket/_repl** : Patroni 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 はその中の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、およびその他の状況下でこのファイルを作成します。この場所は Patroni によって書き込み可能でなければなりません。
- **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` 上の場所からインクルードされます。Patroni はこのファイルの変更を監視したり、バックアップしたりしないことに注意してください。ただし、その設定は Patroni 独自の構成機能によって上書きすることができます。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 はこのパラメータを無視します。このパラメータを **pg/_hba** と組み合わせると、 `pg_hba.conf` の管理が簡素化されます。
 - * -すべてをホスト `0.0.0.0/md5`
 - * -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次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 はこのパラメータを無視します。このパラメータを **pg/_ident** と組み合わせると、 `pg_ident.conf` の管理が簡素化されます。
 - * -マップ名 1 システム名 `1pguser1`
 - * -マップ名 1 システム名 `2pguser2`
- **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** : 前のリーダーが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` を使用してみます。
- **remove/_data/_directory/_on/_rewind/_failure** : このオプションが有効な場合、Patroni 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。そうでなければ、新しいリーダーに従おうとするでしょう。デフォルト値は **remove/_data/_directory/_on/_rewind/_failure** です。
- **remove/_data/_directory/_on/_diverged/_timelines** : Patroni は、タイムラインが分岐し、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の相違チェックを実行している間、Patroni 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` で許可される必要があります。デフォルト値は **remove/_data/_directory/_on/_diverged/_timelines** です。
- **replica/_method** : `basebackup` 以外の `create_replica_method` 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、実行する実際のスクリプトへのフルパスを含む「command」を含める必要があります。他の構成パラメータは、「パラメータ=値」の形式でスクリプトに渡されます。
- **pre/_promote** : フェールオーバー中に、リーダーロックを取得した後、レプリカを昇格する前に実行す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 はレプリカを昇格させず、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、その後シャットダウンが続行されるかどうかには影響しません。

3.2.13 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るため、Patroni セットアップがローカルホスト内のデモを目的としている場合を除き、このアドレスは非「ローカルホスト」またはループバックアドレス (つまり、「ローカルホスト」) である必要があります。"または"/127.0.0.1/"。これは、HTTP ヘルスチェック ("/listen/"RESTAPI パラメータについては以下を参照) のエンドポイントとして機能し、ユーザークエリ (直接または RESTAPI 経由で) や、リーダー選出中のクラスターメンバー (たとえば、リーダーがまだ実行中かどうか、またはクエリを実行しているノードよりも前の WAL 位置を持つノードがあるかどうかを判断するためなど)。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 に接続できるようになります。
- **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-前述のように、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その他のロードバランサ) 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 Basic 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 Basic 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 が指定されていないか空のままの場合、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、信頼された CA の証明書を含む CA_BUNDLE を含む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-AES128-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 の場合、RESTAPI はクライアント証明書をチェックしません。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、クライアント認証は成功します。optional の場合、クライアント証明書は PUT、POST、PATCH、および DELETE リクエストについ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、CIDR 表記を使用したホスト名、IP アドレス、またはネットワークアドレスです。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 が発信接続に別の 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 ヘッダーを使用すると、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):TLS が有効な場合、HTTPS ヘッダーを使用すると、RESTAPI サーバーは 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 で使用される 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、それ以降のリクエストでは「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。

次に、**http_extra_headers** と **https_extra_headers** の両方の例を示します。

```
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。Patroni 内部では、リーダーレース中にこれを使用して、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)、**restapi.verify_client** は **ctl.certfile**、**ctl.keyfile**、**ctl.keyfile_password** にも **restapi.verify_client** を指定します。指定しない場合、Patroni は正しく動作しません。

3.2.14 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。指定されない場合、**username** は RESTAPI の「username」パラメータに指定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。指定されない場合、**password** は RESTAPI の「パスワード」パラメータに指定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、CA_BUNDLE ファイルが含まれるファイル、または信頼された CA の証明書が含まれるディレクトリを指定します。指定されない場合、**cacert** は RESTAPI の「cafile」パラメータに指定された値を使用します。
 - **certfile** :PEM 形式のクライアント証明書が含まれる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** : クライアントの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。

3.2.15 Watchdog

- **mode** : off、automatic または required。off Watchdog が無効になっている場合。automatic の場合、Watchdog が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、Watchdog が正常に有効にならない限り、ノードはリーダーになりません。
- **device** :Watchdog デバイスへのパス。デフォルトは /dev/watchdog です。
- **safety_margin** :Watchdog のトリガーとリーダーキーの有効期限の間の安全マージンの秒数。

3.2.16 タグ

- **clonefrom** : true または false。true に設定すると、他のノードはブートストラップにこのノードの使用を優先する可能性があります (pg_basebackup から取得します)。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が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 に設定すると、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。カスケードレプリケーションをサポートするために使用されます。

- **nosync**: true または false。true に設定すると、ノードは同期レプリカとして選択されなくなります。
- **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 で、このノードがリーダーレースに参加できることを意味します。
- **failover_priority**: 整数。フェイルオーバー中にこのノードが持つべき優先順位を制御します。同じ量の WAL を受信/再生した場合、優先度の高いノードが優先度の低いノードよりも優先されます。ただし、優先順位に関係なく、受信/再生 LSN の値が高いノードが優先されます。**failover_priority** が 0 または負の場合、そのようなノードはリーダーレースに参加してリーダーになることはできません (**nofailover**: true と同様)。

警告: **nofailover** または **failover_priority** のいずれか 1 つだけを指定します。 **nofailover**: true を指定すると **failover_priority**: 0 と同じになり、 **nofailover**: false を指定するとノードの優先順位 1 が与えられます。

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 *REST API* に表示されます。これらのタグを使用してインスタンスの健全性を確認することもできます。タグがインスタンスに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は、HTTP ステータスコード 503 が返されます。

3.3 環境構成の設定

システム環境変数を使用して、Patroni 構成ファイルに定義されている構成パラメーターの一部をオーバーライドすることができます。このドキュメントには、Patroni によって処理され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、Patroni 構成ファイルで設定された値よりも常に優先されます。

3.3.1 グローバル/ユニバーサル

- **PATRONI/_CONFIGURATION**: **PATRONI_CONFIGURATION** 環境変数を介して Patroni の構成全体を設定できます。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。クラスター内で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

3.3.2 ログ

- **PATRONI/_LOG/_TYPE** : ログの形式を設定します。 **PATRONI/_LOG/_TYPE** または **PATRONI/_LOG/_TYPE** のいずれかになります。 **PATRONI/_LOG/_TYPE** 形式を使用するには、 **PATRONI/_LOG/_TYPE** がインストールされている必要があります。デフォルト値は **PATRONI/_LOG/_TYPE** です。
- **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表示したい場合は、 **PATRONI/_LOG/_TRACEBACK/_LEVEL** に設定します。
- **PATRONI/_LOG/_FORMAT** : ログのフォーマット文字列を設定します。ログタイプが **PATRONI/_LOG/_FORMAT** の場合、ログ形式は文字列である必要があります。使用可能な属性については、 **PATRONI/_LOG/_FORMAT** を参照してください。ログタイプが **PATRONI/_LOG/_FORMAT** の場合、ログ形式は文字列に加えてリストにすることもできます。各リスト項目は LogRecord 属性に対応する必要があります。フィールド名のみが必要であり、 `%( および )` は省略する必要があることに注意してください。別のキー名でログフィールドを出力する場合は、辞書キーがログフィールドで、値がログに出力するフィールドの名前である辞書を使用します。デフォルト値は `%(asctime)s%(levelname)s: %(message)s` です。
- **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/_STATIC/_FIELDS** : ログにフィールドを追加します。このオプションは、ログタイプが **PATRONI/_LOG/_STATIC/_FIELDS** に設定されている場合にのみ使用できます。例 `PATRONI_LOG_STATIC_FIELDS={"/app: patroni"}`
- **PATRONI/_LOG/_MAX/_QUEUE/_SIZE** : Patroni は 2 段階のロギングを使用しています。ログレコードはメモリ内のキューに書き込まれ、キューからログレコードを取得して `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ており、過去 1 時間 20 分までのログを保持するには十分なサイズです。
- **PATRONI/_LOG/_DIR** : アプリケーションログを書き込むディレクトリ。ディレクトリは存在し、Patroni を実行す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` を使用して調整できます (以下を参照)。
- **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 `patroni.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}"`

3.3.3 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` などを使用します。
- **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。現在サポートされているデータベースは1つだけです。

3.3.4 Consul

- **PATRONI/_CONSUL/_HOST** : Consul ローカルエージェントのホスト: ポート。
- **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** です。
- **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することができます。
- **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 整合性モードを選択します。可能な値は `default`、`consistent`、または `stale` です (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る 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) ノードの役割に応じて、スコープパラメーターとタグマスター、プライマリ、レプリカ、またはスタンバイリーダーによって定義された名前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に 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。

3.3.5 など

- **PATRONI/_ETCD/_PROXY** :etcd のプロキシ URL。プロキシを使用して etcd に接続している場合は、**PATRONI/_ETCD/_PROXY** の代わりにこのパラメータを使用します。
- **PATRONI/_ETCD/_URL** :etcd の URL、形式:http(s)://(username:password@)host:port
- **PATRONI/_ETCD/_HOSTS** :'host1:port1'、'host2:port2' などの形式の 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して考慮し、etcd クラスターのトポロジー検出を実行せず、**PATRONI/_ETCD/_USE/_PROXIES** の固定リストに固執します。
- **PATRONI/_ETCD/_PROTOCOL** :http または https(指定しない場合は http が使用されます)。 **PATRONI/_ETCD/_PROTOCOL** または **PATRONI/_ETCD/_PROTOCOL** が指定されている場合、それらからプロトコルを取得します。
- **PATRONI/_ETCD/_HOST** :etcd エンドポイントのホスト: ポート。
- **PATRONI/_ETCD/_SRV** : クラスター自動検出のために 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に対して SRV サービス名 (最初に成功するまでこの順序で) のクエリを試行します: `_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は、ETCD ピアプロトコルが使用され、ETCD で利用可能なメンバーをクエリします。それ以外の場合は、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、同じドメイン内の複数の etcd クラスターを区別します。 **PATRONI/_ETCD/_SRV/_SUFFIX** と組み合わせた場合にのみ機能します。たとえば、`PATRONI_ETCD_SRV_SUFFIX=foo` と `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: `_etcd-client-ssl-foo._tcp.example.com` (考えられるすべての ETCDsrv サービス名に対して同様)。
- **PATRONI/_ETCD/_USERNAME** :etcd 認証用のユーザー名。
- **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が含まれるファイル。

- **PATRONI/_ETCD/_KEY** : 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することができます。

3.3.6 etcdv3

Etcdv3 の環境名は Etcd の場合と似ており、変数名に ETCD の代わりに ETCD3 を使用する必要があるだけです。例: PATRONI_ETCD3_HOST、PATRONI_ETCD3_CACERT など。

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、その逆も同様であるため、Patroni 設定を更新するだけでは Etcd から Etcdv3 に切り替えることはできません。

3.3.7 ZooKeeper

- **PATRONI/_ZOOKEEPER/_HOSTS** : ZooKeeper クラスターメンバーのカンマ区切りリスト: `"/'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 が使用されるかどうか。デフォルトは `false` です。 `false` に設定すると、SSL 固有のパラメータ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) 設定した場合、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」スキーマを想定しており、プリンシパルをキーとして、1 つ以上のアクセス許可を値のリストとして持つディクショナリとして指定する必要があります。権限は、CREATE、READ、WRITE、DELETE または ADMIN のいずれかになります。たとえば、`set_acls: {CN=principal1: [CREATE, READ], CN=principal2: [ALL]}`。
- **PATRONI/_ZOOKEEPER/_AUTH/_DATA** : (オプション) 接続に使用する認証資格情報。`scheme` がキー、`credential` が値である形式の辞書である必要があります。デフォルトは空の辞書です。

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。

3.3.8 エキシビター

- **PATRONI/_EXHIBITOR/_HOSTS** : エキシビター (ZooKeeper) ノードの初期リスト (形式: 「host1、host2、etc...」)。このリストは、Exhibitionor(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** : エキシビターポート。

3.3.9 Kubernetes

- **PATRONI/_KUBERNETES/_BYPASS/_API/_SERVICE** :(オプション)KubernetesAPI と通信する場合、Patroni は通常 `kubernetes` サービスに依存し、そのアドレスは `KUBERNETES\_SERVICE\_HOST` 環境変数を介してポッド内で公開されます。 *PATRONI_KUBERNETES_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。
- **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「デフォルト」です。
- **PATRONI/_KUBERNETES/_LABELS** : {label1: value1, label2: value2} 形式のラベル。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 (ポッドとエンドポイントまたは ConfigMap) を検索す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 (エンドポイントまたは 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「クラスタ名」です。
- **PATRONI/_KUBERNETES/_ROLE/_LABEL** :(オプション) ロール (マスター、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、このラベルを実行するポッドに設定します。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「マスター」の場合のポッドラベルの値。デフォルト値は「マスター」です。
- **PATRONI/_KUBERNETES/_FOLLOWER/_LABEL/_VALUE** :(オプション)Postgres ロールが「レプリカ」の場合のポッドラベルの値。デフォルト値は「レプリカ」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 `master` 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。このラベルの値には、対応するロールのデフォルトが常に使用されます。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 に設定すると、Patroni は ConfigMap の代わりにエンドポイントを使用してリーダーの選出を実行し、クラスターの状態を維持します。
- **PATRONI/_KUBERNETES/_POD/_IP** :(オプション)Patroni 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 が昇格され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)Service 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、サービスは機能しません。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。これにより、Patroni はリーダーエンドポイントのサブセットを更新するためにそれを使用します。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)KubernetesAPIServer 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。指定されない場合、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)再試行する 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、または K8sAPI 応答に `retry-after` HTTP ヘッダーがある場合に再試行します。

3.3.10 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`。指定しない場合は、`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。指定しない場合は、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。

3.3.11 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アドレスの後ろにコロンで追加されている限り、カンマで区切られた複数のアドレスが許可されます (例: `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**:他のノードやアプリケーションから Postgres にアクセスできる IP アドレス+ポート。
- **PATRONI/_POSTGRES/_PROXY/_ADDRESS**:Postgres の隣で実行されている接続プール (pg-bouncer など) 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または役立つ可能性があります。

- **PATRONI/_POSTGRESQL/_DATA/_DIR** :Postgres データディレクトリの場所。既存の場所、または Patroni によって初期化される場所。
- **PATRONI/_POSTGRESQL/_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です。Patroni によって書き込み可能である必要があります。
- **PATRONI/_POSTGRESQL/_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、実行可能ファイルの検索に PATH 環境変数が使用されることを意味します。
- **PATRONI/_POSTGRESQL/_BIN/_PG/_CTL** :(オプション) pg_ctl バイナリのカスタム名。
- **PATRONI/_POSTGRESQL/_BIN/_INITDB** :(オプション) initdb バイナリのカスタム名。
- **PATRONI/_POSTGRESQL/_BIN/_PG/_CONTROLDATA** :(オプション) pg_controldata バイナリのカスタム名。
- **PATRONI/_POSTGRESQL/_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 を実行する前および他の状況下でこのファイルを作成します。この場所は Patroni によって書き込み可能でなければなりません。
- **PATRONI/_REPLICATION/_USERNAME** : レプリケーションのユーザー名。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経由でレプリケーションソースにアクセスします。
- **PATRONI/_REPLICATION/_PASSWORD** : レプリケーションのパスワード。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** _接続パラメータにマップします。これにより、クライアントはサーバーとの TLS ネゴシエーションモードのタイプを指定できます。各モードの動作の詳細については、**PATRONI/_REPLICATION/_SSLMODE** _を参照してください。デフォルトのモードは prefer です。
- **PATRONI/_REPLICATION/_SSLKEY** :(オプション) **PATRONI/_REPLICATION/_SSLKEY** _接続パラメータにマップします。これは、クライアントの証明書で使用する秘密キーの場所を指定します。
- **PATRONI/_REPLICATION/_SSLPASSWORD** :(オプション) **PATRONI/_REPLICATION/_SSLPASSWORD** _接続パラメータにマップします。これは、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) **PATRONI/_REPLICATION/_SSLCERT** _接続パラメータにマップされ、クライアント証明書の場所を指定します。
- **PATRONI/_REPLICATION/_SSLROOTCERT** :(オプション) **PATRONI/_REPLICATION/_SSLROOTCERT** _接続パラメータにマップします。これは、クラ

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** _接続パラメータにマップします。これは、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、このリストに存在する証明書を持つ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** _接続パラメータにマップします。これは、証明書失効リストを含むファイルが含まれるディレクトリの場所を指定します。クライアントは、このリストに存在する証明書を持つ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** _接続パラメータにマップします。これにより、セキュアな GSSTCP/IP 接続がサーバとネゴシエートされるかどうか、またはどのような優先順位でネゴシエートされるかを決定します。
- **PATRONI/_REPLICATION/_CHANNEL/_BINDING**:(オプション) **PATRONI/_REPLICATION/_CHANNEL/_BINDING** _接続パラメータにマップします。これは、クライアントによるチャンネルバイন্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 Postgres に接続するために Patroni によって使用されます。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) 中に設定されます。
- **PATRONI/_SUPERUSER/_SSLMODE**:(オプション) **PATRONI/_SUPERUSER/_SSLMODE** _接続パラメータにマップします。これにより、クライアントはサーバーとの TLS ネゴシエーションモードのタイプを指定できます。各モードの動作の詳細については、**PATRONI/_SUPERUSER/_SSLMODE** _を参照してください。デフォルトのモードは prefer です。
- **PATRONI/_SUPERUSER/_SSLKEY**:(オプション) **PATRONI/_SUPERUSER/_SSLKEY** _接続パラメータにマップします。これは、クライアントの証明書で 사용되는秘密キーの場所を指定します。
- **PATRONI/_SUPERUSER/_SSLPASSWORD**:(オプション) **PATRONI/_SUPERUSER/_SSLPASSWORD** _接続パラメータにマップします。これは、**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) **PATRONI/_SUPERUSER/_SSLCERT** _接続パラメータにマップされ、クライアント証明書の場所を指定します。
- **PATRONI/_SUPERUSER/_SSLROOTCERT**:(オプション) **PATRONI/_SUPERUSER/_SSLROOTCERT** _接続パラメータにマップします。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** _接続パラメータにマップします。これは、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、このリストに存在する証明書を持つ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** _接続パラメータにマップします。これは、証明書失効リストを含むファイルが含まれるディレクトリ

の場所を指定します。クライアントは、このリストに存在する証明書を持つ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** _接続パラメータにマップします。これにより、セキュアな GSSTCP/IP 接続がサーバとネゴシエートされるかどうか、またはどのような優先順位でネゴシエートされるかを決定します。
- **PATRONI/_SUPERUSER/_CHANNEL/_BINDING**:(オプション) **PATRONI/_SUPERUSER/_CHANNEL/_BINDING** _接続パラメータにマップします。これは、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 以降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** _接続パラメータにマップします。これにより、クライアントはサーバーとの TLS ネゴシエーションモードのタイプを指定できます。各モードの動作の詳細については、**PATRONI/_REWIND/_SSLMODE** _を参照してください。デフォルトのモードは **prefer** です。
- **PATRONI/_REWIND/_SSLKEY**:(オプション) **PATRONI/_REWIND/_SSLKEY** _接続パラメータにマップします。これは、クライアントの証明書で使用する秘密キーの場所を指定します。
- **PATRONI/_REWIND/_SSLPASSWORD**:(オプション) **PATRONI/_REWIND/_SSLPASSWORD** _接続パラメータにマップします。これは、**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) **PATRONI/_REWIND/_SSLCERT** _接続パラメータにマップされ、クライアント証明書の場所を指定します。
- **PATRONI/_REWIND/_SSLROOTCERT**:(オプション) **PATRONI/_REWIND/_SSLROOTCERT** _接続パラメータにマップします。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** _接続パラメータにマップします。これは、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、このリストに存在する証明書を持つ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** _接続パラメータにマップします。これは、証明書失効リストを含むファイルが含まれるディレクトリの場所を指定します。クライアントは、このリストに存在する証明書を持つ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** _接続パラメータにマップします。これにより、セキュアな GSSTCP/IP 接続がサーバとネゴシエートされるかどうか、またはどのような優先順位でネゴシエートされるかを決定します。
- **PATRONI/_REWIND/_CHANNEL/_BINDING**:(オプション) **PATRONI/_REWIND/_CHANNEL/_BINDING** _接続パラメータにマップします。これは、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。

3.3.12 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**: HAProxy のヘルスチェック情報を提供するために Patroni がリスンする IP アドレスとポート。
- **PATRONI/_RESTAPI/_USERNAME**: 安全でない RESTAPI エンドポイントを保護するための Basic 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 Basic 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 が指定されていないか空のままの場合、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時に使用する、信頼された CA の証明書を含む CA_BUNDLE を含む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 の場合、RESTAPI はクライアント証明書をチェックしません。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、クライアント認証は成功します。optional の場合、クライアント証明書は PUT、POST、PATCH、および DELETE リクエストについ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、CIDR 表記を使用したホスト名、IP アドレス、またはネットワークアドレスです。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 が発信接続に別の 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 ヘッダーを使用すると、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 ヘッダーを使用すると、TLS が有効な場合に、RESTAPI サーバーが 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): Patroni RESTAPI で使用される 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、それ以降のリクエストでは「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する必要があります。Patroni 内部では、リーダーレース中にこれを使用して、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)、 **PATRONI_RESTAPI_VERIFY_CLIENT** は **PATRONI_CTL_CERTFILE** 、 **PATRONI_CTL_KEYFILE** 、 **PATRONI_CTL_KEYFILE_PASSWORD** にも **PATRONI_RESTAPI_VERIFY_CLIENT** を指定します。指定しない場合、Patroni は正しく動作しません。

3.3.13 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。指定されない場合、**PATRONI/_CTL/_USERNAME** は RESTAPI の「username」パラメータに指定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。指定されない場合、**PATRONI/_CTL/_PASSWORD** は RESTAPI の「password」パラメータに指定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) RESTAPI SSL 証明書の検証中に使用する、CA_BUNDLE ファイルが含まれるファイル、または信頼された CA の証明書が含まれるディレクトリを指定します。指定されない場合、**PATRONI/_CTL/_CACERT** は RESTAPI の「cafile」パラメータに指定された値を使用します。
- **PATRONI/_CTL/_CERTFILE** :(オプション) 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。
- **PATRONI/_CTL/_KEYFILE** :(オプション) 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。
- **PATRONI/_CTL/_KEYFILE/_PASSWORD** :(オプション) クライアントの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 の構成には3つのタイプがあります。

- 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 編集構成* ツールまたは Patroni *patronictl 編集構成* 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コンフィギュレーションの一部ではない場合、オプションはすべてのノードに非同期的に (次のウェイクアップサイクル時に) 適用され、その後リロードされます。構成を適用するためにノードの再起動が必要な場合 (値が変更されている場合、コンテキスト `postmaster` を持つ `patronictl` [編集構成](#) の場合)、これを示す特別なフラグ `pending_restart` が `members.dataJSON` に設定されます。さらに、ノードのステータスは `/"restart_pending/": true` を表示することでこれを示します。

- 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、動的構成より優先されます。`patroni.yml` は、Patroni プロセスに `SIGHUP` を送信するか、`POST /reload` REST-API リクエストを実行するか、`patroni.yml` を実行することで、実行時に (Patroni を再起動せずに) 変更および再ロードできます。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になります。ディレクトリの場合、そのディレクトリ内のすべての YAML ファイルがソートされた順序で 1 つずつロードされます。キーが複数のファイルで定義されている場合、最後のファイル内のキーが優先されます。

- *Environment configuration* 。

環境変数を使用して、一部の「ローカル」構成パラメータを設定/上書きすることができます。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ってい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)。

3.4 重要なルール

3.4.1 Patroni によって制御される PostgreSQL パラメーター

PostgreSQL パラメータの一部 **must hold the same values on the primary and the replicas**。これらについては、**must hold the same values on the primary and the replicas**。値を変更または設定するには、DCS の共有設定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` : ホットスタンバイ
- `track_commit_timestamp` : オフ

以下のパラメータについては、PostgreSQL はプライマリとすべてのレプリカの間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、レプリカを異なるように設定することはあまり意味がありません。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検証され、それらが正常であるか、最小値を満たしているかが確認されます。

Patroni によって制御される Postgres パラメーターは他にもいくつかあります。

- `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 以降用。

安全のため、上記のリストのパラメータは `postgresql.conf` には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、`postgresql.conf` よりも高い優先順位が与えられます。

また、`postgresql.listen`、`postgresql.listen`、`postgresql.listen` など、Patroni `postgresql.listen` または `postgresql.listen` 変数経由のパラメータもあります。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`」ファイルがあるかどうか、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` パラメータが設定されている場合、指定されたファイルが基本設定として使用され、「`postgresql.base.conf`」と「`postgresql.conf`」は無視されます。
- `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、名前が変更された「元の」設定が含まれており、基本設定として使用されます。
- `custom_conf` も `postgresql.base.conf` も存在しない場合、元の `postgresql.conf` は `postgresql.base.conf` に名前変更され、基本設定として使用されます。
- 動的オプション (上記の例外を除く) は `postgresql.conf` にダンプされ、インクルードは `postgresql.conf` で基本設定 (`postgresql.base.conf` または `custom_conf` のファイル) に設定されます。したがって、インクルードが存在するかどうかを確認するために構成ファイルを再度読み取ることなく、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必要な一部のパラメーターは、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が変更された場合 (`pg_settings` のコンテキストとそ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)、そのノードに `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。

パラメータは次の順序で適用されます (ランタイムに最も高い優先順位が与えられます)。

1. ファイル `postgresql.base.conf` (または設定されている場合は `custom_conf` ファイル) からパラメータをロードします

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` を使用した特定のノードの構成 (3) が可能になり、Patroni の実行に不可欠なパラメーターが確実に適用されるようになります (4)。また、「postgresql」を管理する構成ツールの余地も残ります。Patroni を介さずに `conf` を直接実行します (1)。

3.4.2 共有メモリに影響を与える PostgreSQL パラメータ

PostgreSQL には、使用される共有メモリのサイズを決定するパラメータがいくつかあります。

- `max_connections`
- `max_prepared_transactions`
- `max_locks_per_transaction`
- `max_wal_senders`
- `max_worker_processes`

これらのパラメータの変更を有効にするには PostgreSQL の再起動が必要であり、スタンバイノードの共有メモリ構造をプライマリノードよりも小さくすることはできません。

前に説明したように、Patroni は *dynamic configuration* を介して値の変更を制限します。通常、これは次のもので構成されます。

1. `patronictl 編集構成` を介して (または RESTAPI `/config` エンドポイントを介して) 変更を適用する
2. `Patroni 再起動` 経由 (または RESTAPI `/restart` エンドポイント経由) のノードの再起動

Note: Note: コマンドまたは RESTAPI `/restart` エンドポイントを介して PostgreSQL ノードの再起動を実行する必要があることに注意してください。Patroni デーモンを再起動して PostgreSQL を再起動しようとします。`systemctl restart patroni` を実行すると、プライマリノードを再起動している場合に、クラスター内でフェイルオーバーが発生する可能性があります。

ただし、これらの設定は共有メモリを管理するため、ノードを再起動するときは特に注意する必要があります。

- これらの設定のいずれかの値を **increase** にしたい場合は、次のようにします。
 1. 最初にすべてのスタンバイを再起動します
 2. その後プライマリを再起動します
- これらの設定のいずれかの値を **decrease** にしたい場合は、次のようにします。
 1. まずプライマリを再起動します
 2. その後、すべてのスタンバイを再起動します

Note: これらの設定値の後にすべてのノードを一度に再起動しようとする、Patroni は変更を無視し、元の設定値でスタンバイを再起動するため、後でスタンバイを再度再起動する必要があります。Patroni は、スタンバイノードの `pg_controldata` で表示される値よりも低い値にこれらのパラメータを設定しようすると、PostgreSQL が「FATAL」メッセージを出して終了するため、スタンバイが無限クラッシュループに陥るのを防ぐためにこれを行います。つまり、プライマリでのこれらの変更に関して、スタンバイの `pg_controldata` がプライマリで最新の状態になった場合にのみ、スタンバイの設定を減らすことができます。

詳細については、 [PostgreSQL Administrator's Overview](#) _を参照してください。

3.4.3 Patroni 設定パラメータ

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、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、Postgres データディレクトリにあるファイル `patroni.dynamic.json` にダンプします。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、リーダーのみがディスク上のダンプからこれらのオプションを復元できます。

3.5 構成の生成と検証

Patroni は、Patroni *local configuration* の生成と検証のためのコマンドラインインターフェイスを提供します。 `patroni` 実行可能ファイルを使用すると、次のことが可能になります。

- サンプルのローカル Patroni 構成を作成します。
- ローカルで実行されている PostgreSQL インスタンスの Patroni 構成ファイルを作成します (例: *Patroni integration* の準備ステップとして)。
- 指定された Patroni 構成ファイルを検証します。

3.5.1 Patroni 構成のサンプル

```
patroni --generate-sample-config [configfile]
```

説明

サンプルの Patroni 構成ファイルを `yaml` 形式で生成します。パラメータ値は `yaml` を使用して定義されます。設定されていない場合は、Patroni で使用されるデフォルト値または後でユーザーが定義する値の `#FIXME` 文字列が使用されます。

一部のデフォルト値はローカル設定に基づいて定義されます。

- **postgresql.listen** : 現在のマシンのホスト名と標準の 5432 ポートの `gethostname` 呼び出しによって返された IP アドレス。
- **postgresql.connect_address** : 現在のマシンのホスト名と標準の 5432 ポートの `gethostname` 呼び出しによって返された IP アドレス。
- **postgresql.authentication.rewind** : PostgreSQL のバージョンがバイナリから定義でき、バージョンが 11 以降の場合にのみ定義されます。
- **restapi.listen** : 現在のマシンのホスト名と標準の 8008 ポートの `gethostname` 呼び出しによって返された IP アドレス。
- **restapi.connect_address** : 現在のマシンのホスト名と標準の 8008 ポートの `gethostname` 呼び出しによって返された IP アドレス。

パラメーター

`configfile` - 結果を保存するために使用される構成ファイルへのフルパス。指定しない場合、結果は `stdout` に送信されます。

3.5.2 実行中のインスタンスの Patroni 構成

```
patroni --generate-config [--dsn DSN] [configfile]
```

説明

ローカルで実行されている PostgreSQL インスタンス用の Patroni 構成を `yaml` 形式で生成します。指定された DSN(優先されます) または PostgreSQL `yaml _` のいずれかが PostgreSQL 接続に使用されます。パスワードが指定されていない場合は、プロンプトから入力する必要があります。

ソース Postgres インスタンスで定義されたすべての非内部 GUC は、設定ファイル、ポストマスターコマンドライン、または環境変数を通じて設定された場合には独立して、次の Patroni 設定パラメータのソースとして使用されます。

- **scope** : cluster_name GUC 値。
- **postgresql.listen** : listen_addresses および port の GUC 値。
- **postgresql.datadir** : data_directory GUC 値。
- **postgresql.parameters** : archive_command、restore_command、archive_cleanup_command、recovery_end_command、ssl_passphrase_command、hba_file、ident_file、config_file GUC 値。
- **bootstrap.dcs** : 収集された他のすべての PostgreSQLGUC。

scope、postgresql.listen または postgresql.datadir が PostgresGUC から設定されていない場合は、それぞれの scope 値が使用されます。

値の定義に適用されるその他のルールは次のとおりです。

- **name** : 設定されている場合は PATRONI_NAME 環境変数の値、設定されていない場合は現在のマシンのホスト名。
- **postgresql.bin_dir** : 実行中のインスタンスから収集された Postgres バイナリへのパス。
- **postgresql.connect_address** : 現在のマシンのホスト名とインスタンス接続に使用されるポート、または port GUC 値の gethostname 呼び出しによって返された IP アドレス。
- **postgresql.authentication.superuser** : インスタンス接続に使用される構成。
- **postgresql.pg_hba** : ソース・インスタンスの hba_file から収集された行。
- **postgresql.pg_ident** : ソース・インスタンスの ident_file から収集された行。
- **restapi.listen** : 現在のマシンのホスト名と標準の 8008 ポートの gethostname 呼び出しによって返された IP アドレス。
- **restapi.connect_address** : 現在のマシンのホスト名と標準の 8008 ポートの gethostname 呼び出しによって返された IP アドレス。

Environment configuration を使用して定義された他のパラメータも構成に含まれます。

パラメーター

configfile

結果を保存するために使用される構成ファイルへのフルパス。指定しない場合、結果は stdout に送信されます。

dsn

GUC 値を取得するローカル PostgreSQL インスタンスのオプションの DSN 文字列。

3.5.3 Patroni 構成を検証する

```
patroni --validate-config [configfile]
```

説明

指定された Patroni 構成を検証し、失敗したチェックに関する情報を出力します。

パラメーター

configfile

確認する構成ファイルへのフルパス。指定されない場合、またはファイルが存在しない場合は、PATRONI_CONFIG_VARIABLE 環境変数からの読み取りを試みます。設定されていない場合は、PATRONI_CONFIG_VARIABLE からの読み取りを試みます。

第4章 パトローニ RESTAPI

Patroni には豊富な RESTAPI があり、リーダーレース中に Patroni 自体によって使用され、フェイルオーバー/スイッチオーバー/再初期化/再起動/リロードを実行するために *Patroni* ツールによって使用され、HTTP ヘルスチェックを実行するために HAProxy またはその他の種類のロードバランサーによって使用されます。もちろんモニタリングにも使用できます。以下に Patroni RESTAPI エンドポイントのリストを示します。

4.1 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 リクエストに対して、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場合、または必要ない場合は、GET の代わりに HEAD または OPTIONS メソッドを使用することを検討してください。

- Patroni RESTAPI への次のリクエストは、Patroni ノードがリーダーロック付きの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 の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。前の 2 つのエンドポイントとの主な違いは、PostgreSQL が **primary** として実行されているか **standby_leader** として実行されているかが考慮されていないことです。
- GET /replica : レプリカのヘルスチェックエンドポイント。Patroni ノードが **running** 状態にあり、ロールが **replica** で、**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、レプリケーションの待ち時間もチェックし、指定された値を下回ってい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 からのキー **cluster.last_leader_operation** は、パフォーマンス上の理由から、リーダー wal の位置とレプリカ上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) または人間が判読できる値で指定できます。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 と、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。タグがインスタンスに定義されていない場合、または yaml 設定内の値がクエリ値と一致しない場合は、HTTP ステータスコード 503 が返されます。

次のリクエストでは、リーダーまたはスタンバイリーダーのステータスをチェックしているため、Patroni はユーザー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と似ていますが、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と似ていますが、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、レプリケーション待機時間もチェックし、指定された値を下回ってい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 からのキー cluster.last_leader_operation は、パフォーマンス上の理由から、リーダー wal の位置とレプリカ上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) または人間が判読できる値で指定できます。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 秒以上前である場合、またはレプリカで 2*ttl を超えている場合は GET /liveness 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 が稼働している場合に、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、エンドポイントは `readinessProbe` に使用できます。

`readiness` エンドポイントと `liveness` エンドポイントはどちらも非常に軽量であり、SQL を実行しません。プローブは、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構成する必要があります。デフォルト値の `ttl`、つまり `30s` を使用すると、プローブの例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
```

4.2 監視エンドポイント

`GET /patroni` は、リーダーレース中にパトローニによって使用されます。監視システムでも使用できます。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: 正常なクラスター

```
$ 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
  },
  "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
  }
],
"dcs_last_seen": 1692356718,
"tags": {
  "clonefrom": true
},
"database_system_identifier": "7268616322854375442",
"patroni": {
  "version": "3.1.0",
  "scope": "demo",
  "name": "patroni1"
}
}

```

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",
  "name": "patroni1"
}
}

```

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": [
    {

```

(次のページに続く)

(前のページからの続き)

```

    "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,
"failsafe_mode_is_active": true,
"dcs_last_seen": 1692356928,
"tags": {
  "clonefrom": true
},
"database_system_identifier": "7268616322854375442",
"patroni": {
  "version": "3.1.0",
  "scope": "demo",
  "name": "patroni1"
}
}

```

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",

```

(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
  }
],
"pause": true,
"dcs_last_seen": 1692356928,
"tags": {
  "clonefrom": true
},
"database_system_identifier": "7268616322854375442",
"patroni": {
  "version": "3.1.0",
  "scope": "demo",
  "name": "patroni1"
}
}

```

GET /metrics エンドポイントを通じて、Patroni メトリクスを Prometheus 形式で取得します。

```

$ curl http://localhost:8008/metrics

# HELP patroni_version Patroni semver without periods. \
# TYPE patroni_version gauge
patroni_version{scope="batman",name="patroni1"} 020103
# HELP patroni_postgres_running Value is 1 if Postgres is running, 0 otherwise.
# TYPE patroni_postgres_running gauge
patroni_postgres_running{scope="batman",name="patroni1"} 1
# HELP patroni_postmaster_start_time Epoch seconds since Postgres started.
# TYPE patroni_postmaster_start_time gauge
patroni_postmaster_start_time{scope="batman",name="patroni1"} 1657656955.179243
# HELP patroni_master Value is 1 if this node is the leader, 0 otherwise.
# TYPE patroni_master gauge
patroni_master{scope="batman",name="patroni1"} 1
# HELP patroni_primary Value is 1 if this node is the leader, 0 otherwise.

```

(次のページに続く)

```

# TYPE patroni_primary gauge
patroni_primary{scope="batman",name="patroni1"} 1
# HELP patroni_xlog_location Current location of the Postgres transaction log, 0 if
→this node is not the leader.
# TYPE patroni_xlog_location counter
patroni_xlog_location{scope="batman",name="patroni1"} 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",name="patroni1"} 0
# HELP patroni_replica Value is 1 if this node is a replica, 0 otherwise.
# TYPE patroni_replica gauge
patroni_replica{scope="batman",name="patroni1"} 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",name="patroni1"} 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",name="patroni1"} 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",name="patroni1"} 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",name="patroni1"} 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",name="patroni1"} 0
# HELP patroni_postgres_streaming Value is 1 if Postgres is streaming, 0 otherwise.
# TYPE patroni_postgres_streaming gauge
patroni_postgres_streaming{scope="batman",name="patroni1"} 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",name="patroni1"} 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",name="patroni1"} 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",name="patroni1"} 0

```

(前のページからの続き)

```
# HELP patroni_postgres_timeline Postgres timeline of this node (if running), 0
↳otherwise.
# TYPE patroni_postgres_timeline counter
patroni_failsafe_mode_is_active{scope="batman",name="patroni1"} 0
# HELP patroni_postgres_timeline Postgres timeline of this node (if running), 0
↳otherwise.
# TYPE patroni_postgres_timeline counter
patroni_postgres_timeline{scope="batman",name="patroni1"} 24
# HELP patroni_dcs_last_seen Epoch timestamp when DCS was last contacted successfully
↳by Patroni.
# TYPE patroni_dcs_last_seen gauge
patroni_dcs_last_seen{scope="batman",name="patroni1"} 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",name="patroni1"} 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",name="patroni1"} 1
```

4.3 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",
```

(次のページに続く)

(前のページからの続き)

```

    "api_url": "http://10.89.0.6:8008/patroni",
    "host": "10.89.0.6",
    "port": 5433,
    "timeline": 5,
    "tags": {
      "clonefrom": true
    },
    "lag": 0
  }
],
"scope": "demo",
"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を表示します。この形式は、pg_wal ディレクトリ内の履歴ファイルの内容とよく似ています。唯一の違いは、新しいタイムラインがいつ作成されたか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,

```

(次のページに続く)

(前のページからの続き)

```

    "no recovery target specified",
    "2019-09-24T09:35:52+02:00"
  ]
]

```

4.4 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": {

```

(次のページに続く)

(前のページからの続き)

```

    "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てみましょう。まず、5 秒ごとにログ行の出力を開始します (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": {
    "version": "1.0",
    "scope": "batman",
    "name": "patroni1"
  },
  "state": "running",
  "role": "master",
  "server_version": 90503
}

```

パラメータの削除:

一部の設定を削除 (リセット) したい場合は、null でパッチを適用するだけです。

```

$ 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": {

```

(次のページに続く)

(前のページからの続き)

```

    "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
    }
}
}

```

上記の呼び出しにより、動的構成から `postgresql.parameters.max_connections` が削除されます。

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可能です。

```

$ 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
}

```

4.5 スイッチオーバーおよびフェイルオーバーのエンドポイント

4.5.1 切り替える

`/switchover` エンドポイントは、クラスターが正常な場合 (リーダーがある場合) にのみ機能します。また、特定の時間にスイッチオーバーをスケジュールすることもできます。

`/switchover` エンドポイントを呼び出す場合、`/failover` エンドポイントとは異なり、候補を指定できますが、必須ではありません。候補者が指定されていない場合、リーダーが退任した後、クラスターの適格なすべてのノードがリーダーレースに参加します。

POST リクエストの JSON 本文で、`leader` フィールドを指定する必要があります。 `candidate` フィールドと `scheduled_at` フィールドはオプションであり、特定の時間にスイッチオーバーをスケジュールするために使用できます。

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本文を返す場合があります。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。何か問題が発生した場合、エラーステータスコード (**200**、**200**、または **200** のいずれか) が応答本文の詳細とともに返されます。

DELETE `/switchover` を使用すると、現在スケジュールされているスイッチオーバーを削除できます。

Example: は、正常なスタンバイへのスイッチオーバーを実行します。

```
$ curl -s http://localhost:8008/switchover -XPOST -d '{"leader":"postgresql1"}'
Successfully switched over to "postgresql2"
```

Example: 特定のノードへのスイッチオーバーを実行します

```
$ curl -s http://localhost:8008/switchover -XPOST -d \
    '{"leader":"postgresql1","candidate":"postgresql2"}'
Successfully switched over to "postgresql2"
```

Example: は、特定の時間にリーダーからクラスター内の他の正常なスタンバイ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":"postgresql0","scheduled_at":"2019-09-24T12:00+00"}'
Switchover scheduled
```

4.5.2 フェイルオーバー

`/failover` エンドポイントは、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するために使用できます (たとえば、すべての同期スタンバイが昇格できるほど正常でない場合は非同期スタンバイに)。ただし、クラスターにリーダーがない必要はありません。正常なクラスターでもフェイルオーバーを実行できます。

POST リクエストの JSON 本文で、`candidate` フィールドを指定する必要があります。 `leader` フィールドが指定されている場合、代わりにスイッチオーバーがトリガーされます。

Example:

```
$ curl -s http://localhost:8008/failover -XPOST -d '{"candidate":"postgresql1"}'
Successfully failed over to "postgresql1"
```

警告: このエンドポイントを使用する場合は *Be very careful* を使用します。これにより、特定の状況でデータ損失が発生する可能性があります。ほとんどの場合、 *Be very careful* は管理者のニーズを満たします。

POST `/switchover` エンドポイントと POST `/failover` エンドポイントは、それぞれ POST `/switchover` と POST `/switchover` によって使用されます。

DELETE `/switchover` は DELETE `/switchover` によって使用されます。

表 1: フェイルオーバー/スイッチオーバーの比較

	フェイルオーバー	切り替える
リーダーの指定が必要です	いいえ	はい
候補者の指定が必要です	はい	いいえ
一時停止中でも実行可能	はい	はい (特定の候補者にのみ)
スケジュール可能	いいえ	はい (一時停止中でない場合)

4.5.3 正常スタンバイ

スイッチオーバー中にリーダーレースに参加できるようにする、またはフェイルオーバー/スイッチオーバーの候補としてリーダーになるために、クラスターのメンバーが合格する必要があるチェックがいくつかあります。

- PatroniAPI 経由でアクセス可能であること。
- `nofailover` タグが `true` に設定されていません。
- Watchdog が完全に機能するようにします (構成で必要な場合)。
- 正常なクラスターでのスイッチオーバーまたは自動フェイルオーバーの場合は、最大レプリケーションラグ (`maximum_lag_on_failover` `maximum_lag_on_failover`) を超えません。

- 正常なクラスターでのスイッチオーバーまたは自動フェイルオーバーの場合、`check_timeline` `check_timeline` が `true` に設定されている場合は、クラスターのタイムラインよりも小さいタイムライン番号を持たないでください。
- *synchronous mode* 内:
 - スwitchオーバーの場合 (候補がある場合とない場合の両方): `/sync` キーメンバーにリストされます。
 - 正常なクラスターと異常なクラスターの両方でのフェイルオーバーの場合、このチェックは省略されます。

警告:

リーダーのいないクラスターで手動フェイルオーバーが行われる場合、次の場合でも候補者は昇格できます。

- 同期モードが有効な場合、これは `/sync` キーメンバーには含まれません。
- その遅延が、許容される最大レプリケーション遅延を超えています。
- タイムライン番号は、最後に知られているクラスタータイムラインよりも小さいです。

4.6 エンドポイントを再起動します

- `POST /restart`: `POST /restart` 呼び出しを実行すると、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再起動条件を指定できます。
 - **restart_pending**: ブール値、`true` に設定すると、Patroni は PostgreSQL 構成に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 リクエスト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バージョン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待機する必要がある時間。`primary_start_timeout` をオーバーライドします。
 - **schedule**: タイムゾーンを含むタイムスタンプ。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` エンドポイントと `DELETE /restart` エンドポイントは、それぞれ `POST /restart` と `POST /restart` によって使用されます。

4.7 エンドポイントのリロード

POST /reload 呼び出しは、Patroni に構成ファイルを再読み込みして適用するように命令します。これは、SIGHUP シグナルを Patroni プロセスに送信するのと同じです。再起動が必要な Postgres パラメータの一部 (POST /reload など) を変更した場合でも、POST /restart エンドポイントを呼び出すか、POST /reload を使用して Postgres を明示的に再起動する必要があります。

リロードエンドポイントは [パトロニトルリロード](#) によって使用されます。

4.8 エンドポイントを再初期化する

POST /reinitialize: 指定されたノード上の PostgreSQL データディレクトリを再初期化します。レプリカ上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が削除され、pg_basebackup または代替の POST /reinitialize が開始されます。

Patroni が障害が発生した Postgres を回復 (再起動) しようとするループに入っている場合、呼び出しは失敗する可能性があります。この問題を解決するには、リクエスト本文に {"force": true} を指定します。

再初期化エンドポイントは [Patroni の再起動](#) によって使用されます。

第5章 Patroni

Patroni には `patronictl` という名前のコマンドラインインターフェイスがあり、これは基本的に Patroni の RESTAPI および DCS と対話するために使用されます。これは、クラスター内での操作の実行を容易にすることを目的としており、人間またはスクリプトで簡単に使用できます。

5.1 構成

`patronictl` は、構成の 3 つのセクションを使用します。

- **ctl**: PatroniRESTAPI に対して認証する方法、およびサーバー ID を検証する方法。詳細については、**ctl** を参照してください。
- **restapi**: PatroniRESTAPI に対して認証する方法、およびサーバー ID を検証する方法。ctl 構成が十分でない場合にのみ使用されます。patronictl は主に `restapi.authentication` セクション (`ctl.authentication` が欠落している場合) と `restapi.cacfile` 設定 (`ctl.cacert` が欠落している場合) に関係します。詳細については、**restapi** を参照してください。
- DCS(例: **etcd**): Patroni が使用する DCS に接続して認証する方法。

これらの構成オプションは、環境変数または構成ファイルから取得できます。 *Environment Configuration Settings* または *Environment Configuration Settings* で上記のセクションを探し、環境変数または構成ファイルを通じてそれらのオプションを設定する方法を理解してください。

環境変数の使用を選択した場合、それは簡単なアプローチです。Patronictl は環境変数を読み取り、その値を使用します。

構成ファイルの使用を選択した場合、使用するファイルについて `patronictl` に通知するさまざまな方法があります。デフォルトでは、`patronictl` は `patronictl.yaml` という名前の構成ファイルをロードしようとします。このファイルは、システムに応じて次のパスのいずれかにあると想定されます。

- MacOSX: `~/Library/Application Support/patroni`
- MacOSX(Posix): `~/.patroni`
- Unix: `~/.config/patroni`
- Unix(Posix): `~/.patroni`
- Windows(ローミング): `C:/Users/<user>/AppData/Roaming/patroni`
- Windows(非ローミング): `C:/Users/<user>/AppData/Local/patroni`

次のいずれかの方法でその動作をオーバーライドできます。

- カスタム構成ファイルへのパスを使用して環境変数 `PATRONICTL_CONFIG_FILE` を設定します。

- `patronictl` の `-c / --config-file` コマンドライン引数をカスタム構成ファイルへのパスとともに使用します。

注釈: `patroni` デーモンが実行されているのと同じホストで `patronictl` を実行している場合、`patronictl` に必要なすべての構成セクションが含まれていれば、同じ構成ファイルを使用できます。

5.2 使用法

`patronictl` は、いくつかの便利な操作を公開します。このセクションでは、それぞれについて説明することを目的としています。

`patronictl` の各サブコマンドに入る前に、`patronictl` 自体に次のコマンドライン引数があることに注意してください。

`-c / --config-file`

前に説明したように、`patronictl` の構成ファイルへのパスを提供するために使用されます。

`-d / --dcs-url / --dcs`

Patroni が使用する DCS に接続文字列を指定します。

この引数は、`patronictl` 構成の DCS および `namespace` 設定をオーバーライドするか、構成にない場合に定義するために使用できます。

値は `DCS://HOST:PORT/NAMESPACE` の形式である必要があります。 `etcd3://localhost:2379/service` は、`service` 名前空間に保存されている Patroni クラスターを使用して、`localhost` 上で実行されている `etcdv3` に接続します。引数の値に欠落している部分は、構成に存在する値またはそのデフォルトに置き換えられます。

`-k / --insecure`

RESTAPI サーバーの SSL 証明書の検証をバイパスするフラグ。

これは、`patronictl` からコマンドを実行する場合の概要です。

```
patronictl [ { -c | --config-file } CONFIG_FILE ]
[ { -d | --dcs-url | --dcs } DCS_URL ]
[ { -k | --insecure } ]
SUBCOMMAND
```

注釈: 概要の構文は次のとおりです。

- 角括弧内のオプションはオプションです。
- 中括弧内のオプションは、「設定の 1 つを選択する」操作を表します。
- `[, ... ]` のオプションは複数回指定できます。
- 大文字で書かれたものは値を与えるべきリテラルを表します。

以下のサブセクションで `patronictl` サブコマンドを説明するときに、これと同じ構文を使用します。また、以下のサブセクションでサブコマンドを説明する場合、コマンドの概要は、上記の概要の `SUBCOMMAND` を置き換えるものと見なす必要があります。

次のサブセクションでは、`patronictl` によって実装される各コマンドの説明を見つけることができます。例として、Patroni の GitHub リポジトリに存在する構成ファイル(ファイル `postgres0.yml`、`postgres1.yml`、および `postgres2.yml`) を使用します。

5.2.1 保護者 DSN

あらすじ

```
dsn
[ CLUSTER_NAME ]
[ { { -r | --role } { leader | primary | standby-leader | replica | standby | any } }
↪ { { -m | --member } MEMBER_NAME } ]
[ --group CITUS_GROUP ]
```

説明

`patronictl dsn` は、Patroni クラスターの 1 つのメンバーの接続文字列を取得します。

複数のメンバーがこのコマンドのパラメーターに一致する場合、プライマリノードを優先してそのうちの 1 つが選択されます。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

指定しない場合、`patronictl` は `scope` 構成が存在する場合、そこから取得しようとします。

-r / --role

指定された役割を持つメンバーを選択します。

役割は次のいずれかになります。

- `leader`: 通常の Patroni クラスターまたはスタンバイ Patroni クラスターのリーダー。または
- `primary`: 通常の Patroni クラスターのリーダー。または
- `standby-leader`: スタンバイ Patroni クラスターのリーダー。または
- `replica`: Patroni クラスターのレプリカ。または
- `standby`: `replica` と同じ。または

- any: 任意のロール。このパラメータを省略した場合と同じです。または

-m / --member

指定された名前を持つクラスターのメンバーを選択します。

MEMBER_NAME はメンバーの名前です。

--group

指定された Citus グループに属するメンバーを選択します。

CITUS_GROUP は Citus グループの ID です。

例

プライマリノードの DSN を取得します。

```
$ patronictl -c postgres0.yml dsn batman -r primary
host=127.0.0.1 port=5432
```

postgresql1 という名前のノードの DSN を取得します。

```
$ patronictl -c postgres0.yml dsn batman --member postgresql1
host=127.0.0.1 port=5433
```

5.2.2 patronictl 編集構成

あらすじ

```
edit-config
[ CLUSTER_NAME ]
[ --group CITUS_GROUP ]
[ { -q | --quiet } ]
[ { -s | --set } CONFIG="VALUE" [, ... ] ]
[ { -p | --pg } PG_CONFIG="PG_VALUE" [, ... ] ]
[ { --apply | --replace } CONFIG_FILE ]
[ --force ]
```

説明

`patronictl edit-config` はクラスターの動的構成を変更し、それによって DCS を更新します。

注釈: TTY を通じて呼び出された場合、コマンドはページャーを通じて動的構成の差分を表示しようとしています。デフォルトでは、`less` または `more` のいずれかを使用しようとしています。別のページャーが必要な場合は、`PAGER` 環境変数を目的のページャーに設定します。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

指定しない場合、`patronictl` は `scope` 構成が存在する場合、そこから取得しようとしています。

--group

指定された Citus グループの動的構成を変更します。

指定しない場合、`patronictl` は `citus.group` 構成が存在する場合、そこから取得しようとしています。

`CITUS_GROUP` は Citus グループの ID です。

-q / --quiet

構成の差分の表示をスキップするフラグ。

-s / --set

特定の動的構成オプションを特定の値で設定します。

`CONFIG` は、`.` によって結合されたレベルを持つ YAML ツリー内の動的構成パスの名前です。

`VALUE` は `CONFIG` の値です。 `null` の場合、`CONFIG` は動的構成から削除されます。

-p / --pg

指定された動的 Postgres 構成オプションを指定された値で設定します。

これは基本的に、`--s / --set` の短縮形であり、`CONFIG` の先頭に `postgresql.parameters.` が付加されています。

`PG_CONFIG` は、設定する Postgres 構成の名前です。

`PG_VALUE` は `PG_CONFIG` の値です。 `null` の場合、`PG_CONFIG` は動的構成から削除されます。

--apply

指定されたファイルから動的構成を適用します。

これは、`CONFIG_FILE` の構成ごとに 1 つずつ、複数の `-s / --set` オプションを指定するのと似ています。

`CONFIG_FILE` は、適用される動的構成を含むファイルへのパス (YAML 形式) です。 `stdin` から読み取りたい場合は、`-` を使用します。

--replace

DCS の動的構成を、指定されたファイルで指定された動的構成に置き換えます。

CONFIG_FILE は、有効になる新しい動的構成を含むファイルへのパス (YAML 形式) です。stdin から読み取りたい場合は、- を使用します。

--force

動的構成を変更するときに確認プロンプトをスキップするフラグ。

スクリプトに便利です。

例

max_connections PostgresGUC を変更します。

```
patronictl -c postgres0.yml edit-config batman --pg max_connections="150" --force
---
+++
@@ -1,6 +1,8 @@
loop_wait: 10
maximum_lag_on_failover: 1048576
postgresql:
+ parameters:
+   max_connections: 150
pg_hba:
- host replication replicator 127.0.0.1/32 md5
- host all all 0.0.0.0/0 md5

Configuration changed
```

loop_wait および ttl 設定を変更します。

```
patronictl -c postgres0.yml edit-config batman --set loop_wait="15" --set ttl="45" --
→force
---
+++
@@ -1,4 +1,4 @@
-loop_wait: 10
+loop_wait: 15
maximum_lag_on_failover: 1048576
postgresql:
pg_hba:
@@ -6,4 +6,4 @@
- host all all 0.0.0.0/0 md5
use_pg_rewind: true
retry_timeout: 10
-ttl: 30
```

(次のページに続く)

(前のページからの続き)

`+ttl: 45`

Configuration changed

`maximum_lag_on_failover` 設定を動的構成から削除します。

```
patronictl -c postgres0.yml edit-config batman --set maximum_lag_on_failover="null" --  
→force
```

+++

`@@ -1,5 +1,4 @@``loop_wait: 10``-maximum_lag_on_failover: 1048576``postgresql:` `pg_hba:` `- host replication replicator 127.0.0.1/32 md5`

Configuration changed

5.2.3 patronictl フェイルオーバー

あらすじ

```
failover  
[ CLUSTER_NAME ]  
[ --group CITUS_GROUP ]  
[ { --leader | --primary } LEADER_NAME ]  
--candidate CANDIDATE_NAME  
[ --force ]
```

説明

`patronictl failover` は、クラスター内で手動フェイルオーバーを実行します。

これは、クラスターが正常でない場合に使用するように設計されています。例:

- リーダーはいない。または
- 同期クラスターでは使用できる同期スタンバイはありません。

同期モードが有効な場合は、非同期ノードにフェイルオーバーすることもできます。

注釈: 正常なクラスターで `patronictl failover` を実行することを妨げるものではありません。ただし、そのような場合には `patronictl switchover` を使用することをお勧めします。

警告: プライマリと比較してプロモートされたレプリカがどの程度最新であるかによっては、フェールオーバーをトリガーするとデータ損失が発生する可能性があります。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

指定しない場合、`patronictl` は `scope` 構成が存在する場合、そこから取得しようとします。

--group

指定された Citus グループでフェイルオーバーを実行します。

CITUS_GROUP は Citus グループの ID です。

--leader / --primary

フェールオーバー時に予想されるリーダーが誰であることを示します。

指定した場合、フェイルオーバーの代わりにスイッチオーバーが実行されます。

LEADER_NAME は、クラスター内の現在のリーダーの名前と一致する必要があります。

警告: この引数は非推奨であり、将来のリリースでは削除される予定です。

--candidate

フェイルオーバー時に昇格されるノード。

CANDIDATE_NAME は、昇格するノードの名前です。

--force

フェイルオーバーの実行時に確認プロンプトをスキップするためのフラグ。

スクリプトに便利です。

例

ノード postgresql2 にフェイルオーバーします。

```
$ patronictl -c postgres0.yml failover batman --candidate postgresql2 --force
Current cluster topology
+ Cluster: batman (7277694203142172922) -+-----+-----+
| Member      | Host              | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+
| postgresql0 | 127.0.0.1:5432    | Leader | running | 3  |           |
| postgresql1 | 127.0.0.1:5433    | Replica | streaming | 3  | 0         |
| postgresql2 | 127.0.0.1:5434    | Replica | streaming | 3  | 0         |
+-----+-----+-----+-----+-----+
2023-09-12 11:52:27.50978 Successfully failed over to "postgresql2"
+ Cluster: batman (7277694203142172922) -+-----+-----+
| Member      | Host              | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+
| postgresql0 | 127.0.0.1:5432    | Replica | stopped |    | unknown   |
| postgresql1 | 127.0.0.1:5433    | Replica | running | 3  | 0         |
| postgresql2 | 127.0.0.1:5434    | Leader | running | 3  |           |
+-----+-----+-----+-----+-----+
```

5.2.4 パトロニトルフラッシュ

あらすじ

```
flush
  CLUSTER_NAME
  [ MEMBER_NAME [, ... ] ]
  { restart | switchover }
  [ --group CITUS_GROUP ]
  [ { -r | --role } { leader | primary | standby-leader | replica | standby | any } ]
  [ --force ]
```

説明

patronictl flush は、スケジュールされたイベントがあればそれを破棄します。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

MEMBER_NAME

指定された Patroni メンバーのスケジュールされたイベントを破棄します。

複数のメンバーを指定できます。メンバーが指定されていない場合は、すべてのメンバーが考慮されます。

注釈: スケジュールされた再起動イベントを破棄する場合にのみ使用されます。

restart

スケジュールされた再起動イベントを破棄します。

switchover

スケジュールされたスイッチオーバーイベントを破棄します。

--group

指定された Citus グループからスケジュールされたイベントを破棄します。

CITUS_GROUP は Citus グループの ID です。

-r / --role

指定された役割を持つメンバーのスケジュールされたイベントを破棄します。

役割は次のいずれかになります。

- **leader**: 通常の Patroni クラスターまたはスタンバイ Patroni クラスターのリーダー。または
- **primary**: 通常の Patroni クラスターのリーダー。または
- **standby-leader**: スタンバイ Patroni クラスターのリーダー。または
- **replica**: Patroni クラスターのレプリカ。または
- **standby: replica** と同じ。または
- **any**: 任意のロール。このパラメータを省略した場合と同じです。

注釈: スケジュールされた再起動イベントを破棄する場合にのみ使用されます。

--force

フラッシュの実行時に確認プロンプトをスキップするためのフラグ。

スクリプトに便利です。

例

スケジュールされたスイッチオーバーイベントを破棄します。

```
$ patronictl -c postgres0.yml flush batman switchover --force
Success: scheduled switchover deleted
```

すべてのスタンバイノードのスケジュールされた再起動を破棄します。

```
$ patronictl -c postgres0.yml flush batman restart -r replica --force
+ Cluster: batman (7277694203142172922) +-----+-----+-----+-----+
↪-----+
| Member      | Host          | Role   | State   | TL | Lag in MB | Scheduled_
↪restart      |
+-----+-----+-----+-----+-----+-----+-----+
↪-----+
| postgresql0 | 127.0.0.1:5432 | Leader | running | 5 |          | 2023-09-
↪12T17:17:00+00:00 |
| postgresql1 | 127.0.0.1:5433 | Replica | streaming | 5 |          0 | 2023-09-
↪12T17:17:00+00:00 |
| postgresql2 | 127.0.0.1:5434 | Replica | streaming | 5 |          0 | 2023-09-
↪12T17:17:00+00:00 |
+-----+-----+-----+-----+-----+-----+-----+
↪-----+
Success: flush scheduled restart for member postgresql1
Success: flush scheduled restart for member postgresql2
```

ノード postgresql0 および postgresql1 のスケジュールされた再起動を破棄します。

```
$ patronictl -c postgres0.yml flush batman postgresql0 postgresql1 restart --force
+ Cluster: batman (7277694203142172922) +-----+-----+-----+-----+
↪-----+
| Member      | Host          | Role   | State   | TL | Lag in MB | Scheduled_
↪restart      |
+-----+-----+-----+-----+-----+-----+-----+
↪-----+
| postgresql0 | 127.0.0.1:5432 | Leader | running | 5 |          | 2023-09-
↪12T17:17:00+00:00 |
| postgresql1 | 127.0.0.1:5433 | Replica | streaming | 5 |          0 | 2023-09-
↪12T17:17:00+00:00 |
| postgresql2 | 127.0.0.1:5434 | Replica | streaming | 5 |          0 | 2023-09-
↪12T17:17:00+00:00 |
+-----+-----+-----+-----+-----+-----+-----+
↪-----+
Success: flush scheduled restart for member postgresql0
Success: flush scheduled restart for member postgresql1
```

5.2.5 Patroni の歴史

あらすじ

```
history
[ CLUSTER_NAME ]
[ --group CITUS_GROUP ]
[ { -f | --format } { pretty | tsv | json | yaml } ]
```

説明

`patronictl history` は、クラスターからのフェイルオーバーおよびスイッチオーバーイベントの履歴を表示します (存在する場合)。

出力には次の情報が含まれます。

TL

イベントが発生した Postgres タイムライン。

LSN

イベントが発生した PostgresLSN。

Reason

Postgres `.history` ファイルから取得された理由。

Timestamp

イベントが発生した時刻。

New Leader

イベント期間中に昇格したパトローニ会員。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

指定しない場合、`patronictl` は `scope` 構成が存在する場合、そこから取得しようとします。

--group

指定された Citus グループのイベント履歴を表示します。

CITUS_GROUP は Citus グループの ID です。

指定しない場合、`patronictl` は `citus.group` 構成が存在する場合、そこから取得しようとします。

-f / --format

出力内のイベントのリストをフォーマットする方法。

形式は次のいずれかになります。

- `pretty`: 履歴を美しいテーブルとして出力します。または
- `tsv`: 履歴を `/t` で区切られた列を含む表形式の情報として出力します。または
- `json`: 履歴を JSON 形式で出力します。または
- `yaml`: 履歴を YAML 形式で出力します。

デフォルトは `pretty` です。

`--force`

フラッシュの実行時に確認プロンプトをスキップするためのフラグ。

スクリプトに便利です。

例

イベントの履歴を表示します。

```
$ patronictl -c postgres0.yml history batman
```

TL	LSN	Reason	Timestamp
New Leader			
1	24392648	no recovery target specified	2023-09-11T22:11:27.125527+00:00
2	50331864	no recovery target specified	2023-09-12T11:34:03.148097+00:00
3	83886704	no recovery target specified	2023-09-12T11:52:26.948134+00:00
4	83887280	no recovery target specified	2023-09-12T11:53:09.620136+00:00

イベントの履歴を YAML 形式で表示します。

```
$ patronictl -c postgres0.yml history batman -f yaml
```

```
- LSN: 24392648
  New Leader: postgresql0
  Reason: no recovery target specified
  TL: 1
  Timestamp: '2023-09-11T22:11:27.125527+00:00'
- LSN: 50331864
  New Leader: postgresql0
  Reason: no recovery target specified
```

(次のページに続く)

```

TL: 2
Timestamp: '2023-09-12T11:34:03.148097+00:00'
- LSN: 83886704
New Leader: postgresql2
Reason: no recovery target specified
TL: 3
Timestamp: '2023-09-12T11:52:26.948134+00:00'
- LSN: 83887280
New Leader: postgresql0
Reason: no recovery target specified
TL: 4
Timestamp: '2023-09-12T11:53:09.620136+00:00'

```

5.2.6 Patroni リスト

あらすじ

```

list
[ CLUSTER_NAME [, ... ] ]
[ --group CITUS_GROUP ]
[ { -e | --extended } ]
[ { -t | --timestamp } ]
[ { -f | --format } { pretty | tsv | json | yaml } ]
[ { -W | { -w | --watch } TIME } ]

```

説明

`patronictl list` は、Patroni クラスターとそのメンバーに関する情報を示します。

出力には次の情報が含まれます。

Cluster

Patroni クラスターの名前。

Member

パトローニのメンバーの名前。

Host

メンバーがいるホスト。

Role

メンバーの現在の役割。

次のいずれかになります:

- **Leader**: 通常の Patroni クラスターの現在のリーダー。または

- **Standby Leader** : Patroni スタンバイクラスターの現在のリーダー。または
- **Sync Standby** : 同期モードが有効になっている Patroni クラスターの同期スタンバイ。または
- **Replica** : Patroni クラスターの通常のスタンバイ。

State

Patroni メンバーの Postgres の現在の状態。

考えられる状態の例をいくつか示します。

- **running** : Postgres が現在稼働している場合。
- **streaming** : レプリカと Postgres が現在プライマリノードから WAL をストリーミングしている場合。
- **in archive recovery** : レプリカと Postgres が現在アーカイブから WAL をフェッチしている場合。
- **stopped** : Postgres がシャットダウンされていた場合。
- **crashed** : Postgres がクラッシュした場合。

TL

Patroni メンバーの現在の Postgres タイムライン。

Lag in MB

Patroni メンバーとそのアップストリームの間のレプリケーションラグの量 (メガバイト単位)。

それに加えて、次の情報が出力に含まれる場合があります。

System identifier

Postgres システム識別子。

注釈: テーブルのヘッダーに表示されます。

出力形式が **pretty** の場合にのみ表示されます。

Group

Citus グループ ID。

注釈: テーブルのヘッダーに表示されます。

Citus クラスターの場合にのみ表示されます。

Pending restart

* は、一部の Postgres 構成を有効にするためにノードの再起動が必要であることを示します。空の値は、ノードを再起動する必要がないことを示します。

注釈: メンバー属性として表示されます。

次の場合に表示されます:

- `pretty` または `tsv` 形式で拡張出力を有効にして印刷します。または
 - ノードの再起動が必要な場合。
-

Scheduled restart

Patroni メンバーが管理する Postgres インスタンスの再起動がスケジュールされたタイムスタンプ。空の値は、メンバーにスケジュールされた再起動がないことを示します。

注釈: メンバー属性として表示されます。

次の場合に表示されます:

- `pretty` または `tsv` 形式で拡張出力を有効にして印刷します。または
 - ノードにスケジュールされた再起動がある場合。
-

Tags

Patroni メンバーに設定されたタグが含まれます。空の値は、タグが設定されていないか、タグがデフォルト値で設定されていることを示します。

注釈: メンバー属性として表示されます。

次の場合に表示されます:

- `pretty` または `tsv` 形式で拡張出力を有効にして印刷します。または
 - ノードにカスタムタグ、またはデフォルト以外の値を持つデフォルトタグがある場合。
-

Scheduled switchover

Patroni クラスターのスイッチオーバーがスケジュールされている場合のタイムスタンプ。

注釈: 表のフッターに表示されます。

スケジュールされたスイッチオーバーがあり、出力形式が `pretty` である場合にのみ表示されます。

Maintenance mode

クラスター監視が現在一時停止されている場合。

注釈: 表のフッターに表示されます。

クラスターが一時停止されており、出力形式が `pretty` である場合にのみ表示されます。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

指定しない場合、`patronictl` は `scope` 構成が存在する場合、そこから取得しようとします。

--group

指定された Citus グループのメンバーに関する情報を表示します。

CITUS_GROUP は Citus グループの ID です。

-e / --extended

拡張情報を表示します。

Pending restart、Scheduled restart、および Tags 属性を、値が空であっても強制的に表示します。

注釈: `pretty` および `tsv` 出力形式にのみ適用されます。

-t / --timestamp

クラスターとそのメンバーに関する情報を出力する前に、タイムスタンプを出力します。

-f / --format

出力内のイベントのリストをフォーマットする方法。

形式は次のいずれかになります。

- `pretty`: 履歴を美しいテーブルとして出力します。または
- `tsv`: 履歴を `/t` で区切られた列を含む表形式の情報として出力します。または
- `json`: 履歴を JSON 形式で出力します。または
- `yaml`: 履歴を YAML 形式で出力します。

デフォルトは `pretty` です。

-W

2 秒ごとに情報を自動的に更新します。

-w / --watch

指定した間隔で情報を自動的に更新します。

TIME は、更新間隔 (秒単位) です。

例

クラスターに関する情報をきれいな形式で表示します。

```
$ patronictl -c postgres0.yml list batman
+ Cluster: batman (7277694203142172922) -+-----+-----+-----+
| Member      | Host          | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| postgresql0 | 127.0.0.1:5432 | Leader | running | 5 |           |
| postgresql1 | 127.0.0.1:5433 | Replica | streaming | 5 | 0 |
| postgresql2 | 127.0.0.1:5434 | Replica | streaming | 5 | 0 |
+-----+-----+-----+-----+-----+-----+
```

拡張列を使用して、クラスターに関する情報をわかりやすい形式で表示します。

```
$ patronictl -c postgres0.yml list batman -e
+ Cluster: batman (7277694203142172922) -+-----+-----+-----+
↪+-----+-----+
| Member      | Host          | Role   | State   | TL | Lag in MB | Pending↪
↪restart | Scheduled restart | Tags |
+-----+-----+-----+-----+-----+-----+
↪+-----+-----+
| postgresql0 | 127.0.0.1:5432 | Leader | running | 5 |           |
|             |                 |       |         |   |           |
| postgresql1 | 127.0.0.1:5433 | Replica | streaming | 5 | 0 |
|             |                 |       |         |   |           |
| postgresql2 | 127.0.0.1:5434 | Replica | streaming | 5 | 0 |
|             |                 |       |         |   |           |
+-----+-----+-----+-----+-----+-----+
↪+-----+-----+
```

クラスターに関する情報を実行のタイムスタンプとともに YAML 形式で表示します。

```
$ patronictl -c postgres0.yml list batman -f yaml -t
2023-09-12 13:30:48
- Cluster: batman
  Host: 127.0.0.1:5432
  Member: postgresql0
  Role: Leader
  State: running
  TL: 5
- Cluster: batman
  Host: 127.0.0.1:5433
  Lag in MB: 0
  Member: postgresql1
  Role: Replica
  State: streaming
```

(次のページに続く)

(前のページからの続き)

```
TL: 5
- Cluster: batman
Host: 127.0.0.1:5434
Lag in MB: 0
Member: postgresql2
Role: Replica
State: streaming
TL: 5
```

5.2.7 一時停止

あらすじ

```
pause
[ CLUSTER_NAME ]
[ --group CITUS_GROUP ]
[ --wait ]
```

説明

`patronictl pause` は、Patroni クラスターを一時的にメンテナンスモードにし、自動フェイルオーバーを無効にします。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

指定しない場合、`patronictl` は `scope` 構成が存在する場合、そこから取得しようとします。

--group

指定された Citus グループを一時停止します。

CITUS_GROUP は Citus グループの ID です。

指定しない場合、`patronictl` は `citus.group` 構成が存在する場合、そこから取得しようとします。

--wait

すべての Patroni メンバーが一時停止されるまで待ってから、呼び出し元に制御を返します。

例

クラスターをメンテナンスモードにし、すべてのノードが一時停止されるまで待ちます。

```
$ patronictl -c postgres0.yml pause batman --wait
'pause' request sent, waiting until it is recognized by all nodes
Success: cluster management is paused
```

5.2.8 Patroni クエリ

あらすじ

```
query
[ CLUSTER_NAME ]
[ --group CITUS_GROUP ]
[ { { -r | --role } { leader | primary | standby-leader | replica | standby | any } }
→| { -m | --member } MEMBER_NAME } ]
[ { -d | --dbname } DBNAME ]
[ { -U | --username } USERNAME ]
[ --password ]
[ --format { pretty | tsv | json | yaml } ]
[ { { -f | --file } FILE_NAME | { -c | --command } SQL_COMMAND } ]
[ --delimiter ]
[ { -W | { -w | --watch } TIME } ]
```

説明

`patronictl query` は、Patroni クラスターのメンバーに対して SQL コマンドまたはスクリプトを実行します。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

指定しない場合、`patronictl` は `scope` 構成が存在する場合、そこから取得しようとします。

--group

指定された Citus グループをクエリします。

CITUS_GROUP は Citus グループの ID です。

-r / --role

指定された役割を持つメンバーを選択します。

役割は次のいずれかになります。

- **leader**: 通常の Patroni クラスターまたはスタンバイ Patroni クラスターのリーダー。または
- **primary**: 通常の Patroni クラスターのリーダー。または
- **standby-leader**: スタンバイ Patroni クラスターのリーダー。または
- **replica**: Patroni クラスターのレプリカ。または
- **standby**: **replica** と同じ。または
- **any**: 任意のロール。このパラメータを省略した場合と同じです。

-m / --member

指定された名前を持つメンバーを選択します。

MEMBER_NAME は、選択されるメンバーの名前です。

-d / --dbname

接続してクエリを実行するデータベース。

DBNAME はデータベースの名前です。指定しない場合、デフォルトは USERNAME になります。

-U / --username

データベースに接続するユーザー。

USERNAME ユーザーの名前。指定しない場合、デフォルトは `patronictl query` を実行しているオペレーティングシステムユーザーになります。

--password

接続するユーザーのパスワードの入力を求められます。

Patroni は `libpq` を使用するため、代わりに `~/.pgpass` ファイルを作成するか、`PGPASSWORD` 環境変数を設定できます。

--format

クエリの出力をフォーマットする方法。

形式は次のいずれかになります。

- **pretty**: クエリ出力を美しいテーブルとして出力します。または
- **tsv**: クエリ出力を `/t` で区切られた列を含む表形式の情報として出力します。または
- **json**: クエリ出力を JSON 形式で出力します。または
- **yaml**: クエリ出力を YAML 形式で出力します。

デフォルトは `tsv` です。

-f / --file

ファイルをコマンドのソースとして使用してクエリを実行します。

FILE_NAME はソースファイルへのパスです。

-c / --command

クエリで指定された SQL コマンドを実行します。

SQL_COMMAND は、実行される SQL コマンドです。

--delimiter

情報を tsv 形式で出力する場合の区切り文字。省略した場合は /t。

-W

2 秒ごとにクエリを自動的に再実行します。

-w / --watch

指定された間隔でクエリを自動的に再実行します。

TIME は、再実行の間隔 (秒単位) です。

例

postgres ユーザーとして SQL コマンドを実行し、パスワードを要求します。

```
$ patronictl -c postgres0.yml query batman -U postgres --password -c "SELECT now()"
Password:
now
2023-09-12 18:10:53.228084+00:00
```

postgres ユーザーとして SQL コマンドを実行し、libpq 環境変数からパスワードを取得します。

```
$ PGPASSWORD=zalando patronictl -c postgres0.yml query batman -U postgres -c "SELECT
↪now()"
now
2023-09-12 18:11:37.639500+00:00
```

SQL コマンドを実行し、2 秒ごとに pretty 形式で出力します。

```
$ patronictl -c postgres0.yml query batman -c "SELECT now()" --format pretty -W
+-----+
| now                |
+-----+
| 2023-09-12 18:12:16.716235+00:00 |
+-----+
+-----+
| now                |
+-----+
| 2023-09-12 18:12:18.732645+00:00 |
+-----+
+-----+
| now                |
+-----+
```

(次のページに続く)

(前のページからの続き)

```
| 2023-09-12 18:12:20.750573+00:00 |
+-----+
```

データベース test で SQL コマンドを実行し、出力を YAML 形式で出力します。

```
$ patronictl -c postgres0.yml query batman -d test -c "SELECT now() AS column_1, 'test
→' AS column_2" --format yaml
- column_1: 2023-09-12 18:14:22.052060+00:00
  column_2: test
```

メンバー postgresql2 に対して SQL コマンドを実行します。

```
$ patronictl -c postgres0.yml query batman -m postgresql2 -c "SHOW port"
port
5434
```

いずれかのスタンバイで SQL コマンドを実行します。

```
$ patronictl -c postgres0.yml query batman -r replica -c "SHOW port"
port
5433
```

5.2.9 Patroni の再起動

あらすじ

```
reinit
  CLUSTER_NAME
  [ MEMBER_NAME [, ... ] ]
  [ --group CITUS_GROUP ]
  [ --wait ]
  [ --force ]
```

説明

`patronictl reinit` は、Patroni クラスターのレプリカメンバーによって管理される Postgres スタンバイインスタンスを再構築します。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

MEMBER_NAME

Postgres インスタンスが再構築されるレプリカメンバーの名前。

複数のレプリカメンバーを指定できます。メンバーが指定されていない場合、コマンドは何も行いません。

--group

指定された Citus グループのレプリカメンバーを再構築します。

CITUS_GROUP は Citus グループの ID です。

--wait

Postgres スタンバイノードの再初期化が完了するまで待ちます。

--force

Postgres スタンバイインスタンスを再構築するときに確認プロンプトをスキップするためのフラグ。
スクリプトに便利です。

例

Patroni クラスターのすべてのレプリカメンバーの再構築を要求し、すぐに呼び出し元に制御を返します。

```
$ patronictl -c postgres0.yml reinit batman postgresql1 postgresql2 --force
+ Cluster: batman (7277694203142172922) +-----+-----+-----+
| Member      | Host          | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| postgresql0 | 127.0.0.1:5432 | Leader | running | 5 |           |
| postgresql1 | 127.0.0.1:5433 | Replica | streaming | 5 |           0 |
| postgresql2 | 127.0.0.1:5434 | Replica | streaming | 5 |           0 |
+-----+-----+-----+-----+-----+-----+
Success: reinitialize for member postgresql1
Success: reinitialize for member postgresql2
```

postgresql2 の再構築をリクエストし、完了するまで待ちます。

```
$ patronictl -c postgres0.yml reinit batman postgresql2 --wait --force
+ Cluster: batman (7277694203142172922) +-----+-----+-----+
| Member      | Host          | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| postgresql0 | 127.0.0.1:5432 | Leader | running | 5 |           |
| postgresql1 | 127.0.0.1:5433 | Replica | streaming | 5 |           0 |
| postgresql2 | 127.0.0.1:5434 | Replica | streaming | 5 |           0 |
+-----+-----+-----+-----+-----+-----+
```

(次のページに続く)

(前のページからの続き)

```
Success: reinitialize for member postgresql2
Waiting for reinitialize to complete on: postgresql2
Reinitialize is completed on: postgresql2
```

5.2.10 パトロニトルリロード

あらすじ

```
reload
  CLUSTER_NAME
  [ MEMBER_NAME [, ... ] ]
  [ --group CITUS_GROUP ]
  [ { -r | --role } { leader | primary | standby-leader | replica | standby | any } ]
  [ --force ]
```

説明

`patronictl reload` は、1 つ以上の Patroni メンバーのローカル構成のリロードを要求します。

また、何も変更されていない場合でも、管理対象 Postgres インスタンスで `pg_ctl reload` がトリガーされます。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

MEMBER_NAME

指定された Patroni メンバーのローカル構成のリロードを要求します。

複数のメンバーを指定できます。メンバーが指定されていない場合は、すべてのメンバーが考慮されます。

--group

指定された Citus グループのメンバーのリロードを要求します。

CITUS_GROUP は Citus グループの ID です。

-r / --role

指定された役割を持つメンバーを選択します。

役割は次のいずれかになります。

- **leader** : 通常の Patroni クラスターまたはスタンバイ Patroni クラスターのリーダー。または
- **primary** : 通常の Patroni クラスターのリーダー。または

- `standby-leader`: スタンバイ Patroni クラスターのリーダー。または
- `replica`: Patroni クラスターのレプリカ。または
- `standby`: `replica` と同じ。または
- `any`: 任意のロール。このパラメータを省略した場合と同じです。

--force

ローカル構成のリロードを要求するときに確認プロンプトをスキップするためのフラグ。

スクリプトに便利です。

例

Patroni クラスターのすべてのメンバーのローカル構成のリロードを要求します。

```
$ patronictl -c postgres0.yml reload batman --force
+ Cluster: batman (7277694203142172922) -+-----+-----+-----+
| Member          | Host                | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| postgresql0     | 127.0.0.1:5432      | Leader | running | 5  |           |
| postgresql1     | 127.0.0.1:5433      | Replica | streaming | 5  | 0         |
| postgresql2     | 127.0.0.1:5434      | Replica | streaming | 5  | 0         |
+-----+-----+-----+-----+-----+-----+
Reload request received for member postgresql0 and will be processed within 10 seconds
Reload request received for member postgresql1 and will be processed within 10 seconds
Reload request received for member postgresql2 and will be processed within 10 seconds
```

5.2.11 Patroni を削除します

あらすじ

```
remove
  CLUSTER_NAME
  [ --group CITUS_GROUP ]
  [ { -f | --format } { pretty | tsv | json | yaml } ]
```

説明

`patronictl remove` は、DCS からクラスターの情報を削除します。

インタラクティブなアクションです。

警告: この操作により、DCS から Patroni クラスターの情報が破棄されます。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

--group

指定された Citus グループに関連する Patroni クラスターに関する情報を削除します。

CITUS_GROUP は Citus グループの ID です。

-f / --format

確認を求めるプロンプトが表示されるときに、出力内のメンバーのリストをフォーマットする方法。

形式は次のいずれかになります。

- `pretty`: メンバーを美しいテーブルとして出力します。または
- `tsv`: メンバーを `/t` で区切られた列を含む表形式の情報として出力します。または
- `json`: メンバーを JSON 形式で出力します。または
- `yaml`: メンバーを YAML 形式で出力します。

デフォルトは `pretty` です。

例

Patroni クラスター `batman` に関する情報を DCS から削除します。

```
$ patronictl -c postgres0.yml remove batman
+ Cluster: batman (7277694203142172922) -+-----+-----+-----+
| Member      | Host              | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| postgresql0 | 127.0.0.1:5432    | Leader | running | 5  |           |
| postgresql1 | 127.0.0.1:5433    | Replica | streaming | 5  | 0         |
| postgresql2 | 127.0.0.1:5434    | Replica | streaming | 5  | 0         |
+-----+-----+-----+-----+-----+-----+
Please confirm the cluster name to remove: batman
You are about to remove all information in DCS for batman, please type: "Yes I am
↪aware": Yes I am aware
```

(次のページに続く)

(前のページからの続き)

```
This cluster currently is healthy. Please specify the leader name to continue:  
→ postgresql0
```

5.2.12 Patroni 再起動

あらすじ

```
restart
  CLUSTER_NAME
  [ MEMBER_NAME [, ...] ]
  [ --group CITUS_GROUP ]
  [ { -r | --role } { leader | primary | standby-leader | replica | standby | any } ]
  [ --any ]
  [ --pg-version PG_VERSION ]
  [ --pending ]
  [ --timeout TIMEOUT ]
  [ --scheduled TIMESTAMP ]
  [ --force ]
```

説明

`patronictl restart` は、Patroni クラスターのメンバーによって管理される Postgres インスタンスの再起動を要求します。

再起動はすぐに実行することも、後で実行するようにスケジュールすることもできます。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

--group

指定された Citus グループに関連する Patroni クラスターを再起動します。

CITUS_GROUP は Citus グループの ID です。

-r / --role

与えられた役割を持つメンバーを選択します。

役割は次のいずれかになります。

- **leader**: 通常の Patroni クラスターまたはスタンバイ Patroni クラスターのリーダー。または
- **primary**: 通常の Patroni クラスターのリーダー。または

- **standby-leader**: スタンバイ Patroni クラスターのリーダー。または
- **replica**: Patroni クラスターのレプリカ。または
- **standby**: **replica** と同じ。または
- **any**: 任意のロール。このパラメータを省略した場合と同じです。

--any

指定されたフィルターに一致するノードの中からランダムな 1 つのノードを再起動します。

--pg-version

管理対象 Postgres インスタンスのバージョンが指定されたバージョンよりも古いメンバーのみを選択します。

PG_VERSION は、比較する Postgres のバージョンです。

--pending

Pending restart というフラグが付いているメンバーのみを選択します。

timeout

指定されたタイムアウトを超えて再起動がかかる場合は再起動を中止し、問題がプライマリにある場合はレプリカにフェールオーバーします。

TIMEOUT は、再起動を中止するまでに待機する秒数です。

--scheduled

指定されたタイムスタンプで再起動が行われるようにスケジュールを設定します。

TIMESTAMP は、再起動が行われるときのタイムスタンプです。明確な形式で、できればタイムゾーンを使用して指定します。リテラル **now** を使用して、再起動をすぐに実行することもできます。

--force

再起動操作を要求するときに確認プロンプトをスキップするためのフラグ。

スクリプトに便利です。

例

クラスターのすべてのメンバーをすぐに再起動します。

```
$ patronictl -c postgres0.yml restart batman --force
+ Cluster: batman (7277694203142172922) -+-----+-----+-----+
| Member          | Host              | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| postgresql0     | 127.0.0.1:5432    | Leader | running | 6  |           |
| postgresql1     | 127.0.0.1:5433    | Replica | streaming | 6  | 0         |
| postgresql2     | 127.0.0.1:5434    | Replica | streaming | 6  | 0         |
+-----+-----+-----+-----+-----+-----+
Success: restart on member postgresql0
Success: restart on member postgresql1
Success: restart on member postgresql2
```

クラスターのランダムなメンバーをすぐに再起動します。

```
$ patronictl -c postgres0.yml restart batman --any --force
+ Cluster: batman (7277694203142172922) -+-----+-----+
| Member      | Host           | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| postgresql0 | 127.0.0.1:5432 | Leader | running | 6 |          |
| postgresql1 | 127.0.0.1:5433 | Replica | streaming | 6 |          0 |
| postgresql2 | 127.0.0.1:5434 | Replica | streaming | 6 |          0 |
+-----+-----+-----+-----+-----+-----+
Success: restart on member postgresql1
```

2023-09-13T18:00-03:00 で再起動が行われるようにスケジュールします。

```
$ patronictl -c postgres0.yml restart batman --scheduled 2023-09-13T18:00-03:00 --
→force
+ Cluster: batman (7277694203142172922) -+-----+-----+
| Member      | Host           | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| postgresql0 | 127.0.0.1:5432 | Leader | running | 6 |          |
| postgresql1 | 127.0.0.1:5433 | Replica | streaming | 6 |          0 |
| postgresql2 | 127.0.0.1:5434 | Replica | streaming | 6 |          0 |
+-----+-----+-----+-----+-----+-----+
Success: restart scheduled on member postgresql0
Success: restart scheduled on member postgresql1
Success: restart scheduled on member postgresql2
```

5.2.13 保護者の履歴書

あらすじ

```
resume
[ CLUSTER_NAME ]
[ --group CITUS_GROUP ]
[ --wait ]
```

説明

`patronictl resume` は、Patroni クラスターのメンテナンスモードを解除し、自動フェイルオーバーを再度有効にします。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

指定しない場合、`patronictl` は `scope` 構成が存在する場合、そこから取得しようとします。

--group

指定された Citus グループを再開します。

CITUS_GROUP は Citus グループの ID です。

指定しない場合、`patronictl` は `citus.group` 構成が存在する場合、そこから取得しようとします。

--wait

すべての Patroni メンバーの一時停止が解除されるまで待ってから、呼び出し元に制御を返します。

例

クラスターをメンテナンスモードから解除します。

```
$ patronictl -c postgres0.yml resume batman --wait
'resume' request sent, waiting until it is recognized by all nodes
Success: cluster management is resumed
```

5.2.14 patronictlshow-config

あらすじ

```
show-config
[ CLUSTER_NAME ]
[ --group CITUS_GROUP ]
```

説明

`patronictl show-config` は、DCS に保存されているクラスターの動的構成を示します。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

指定しない場合、`patronictl` は `scope` 構成が存在する場合、そこから取得しようとします。

--group

指定された Citus グループの動的構成を表示します。

CITUS_GROUP は Citus グループの ID です。

指定しない場合、`patronictl` は `citus.group` 構成が存在する場合、そこから取得しようとします。

例

クラスター `batman` の動的構成を表示します。

```
$ patronictl -c postgres0.yml show-config batman
loop_wait: 10
postgresql:
  parameters:
    max_connections: 250
  pg_hba:
    - host replication replicator 127.0.0.1/32 md5
    - host all all 0.0.0.0/0 md5
  use_pg_rewind: true
retry_timeout: 10
ttl: 30
```

5.2.15 Patroni 切り替え

あらすじ

```
switchover
[ CLUSTER_NAME ]
[ --group CITUS_GROUP ]
[ { --leader | --primary } LEADER_NAME ]
--candidate CANDIDATE_NAME
[ --force ]
```

説明

`patronictl switchover` はクラスター内でスイッチオーバーを実行します。

これは、クラスターが正常な場合に使用されるように設計されています。例:

- リーダーがいる。
- 同期クラスターでは使用可能な同期スタンバイがあります。

注釈: クラスターが正常でない場合は、代わりに `patronictl failover` に興味があるかもしれません。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

指定しない場合、`patronictl` は `scope` 構成が存在する場合、そこから取得しようとします。

--group

指定された Citus グループでスイッチオーバーを実行します。

CITUS_GROUP は Citus グループの ID です。

--leader / --primary

切り替え時に降格されるリーダーを指定します。

LEADER_NAME は、クラスター内の現在のリーダーの名前と一致する必要があります。

--candidate

スイッチオーバー時に昇格され、プライマリの役割を担うノード。

CANDIDATE_NAME は、昇格するノードの名前です。

--scheduled

指定されたタイムスタンプでスイッチオーバーが発生するようにスケジュールします。

TIMESTAMP は、スイッチオーバーが発生するときのタイムスタンプです。明確な形式で、できればタイムゾーンを使用して指定します。リテラル `now` を使用してスイッチオーバーを即時に実行することもできます。

--force

スイッチオーバーの実行時に確認プロンプトをスキップするためのフラグ。

スクリプトに便利です。

例

ノード postgresql2 に切り替えます。

```
$ patronictl -c postgres0.yml switchover batman --leader postgresql0 --candidate_
→ postgresql2 --force
Current cluster topology
+ Cluster: batman (7277694203142172922) -+-----+-----+-----+
| Member      | Host          | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| postgresql0 | 127.0.0.1:5432 | Leader | running | 6 |           |
| postgresql1 | 127.0.0.1:5433 | Replica | streaming | 6 |          0 |
| postgresql2 | 127.0.0.1:5434 | Replica | streaming | 6 |          0 |
+-----+-----+-----+-----+-----+-----+
2023-09-13 14:15:23.07497 Successfully switched over to "postgresql2"
+ Cluster: batman (7277694203142172922) -+-----+-----+-----+
| Member      | Host          | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| postgresql0 | 127.0.0.1:5432 | Replica | stopped |   | unknown   |
| postgresql1 | 127.0.0.1:5433 | Replica | running | 6 |          0 |
| postgresql2 | 127.0.0.1:5434 | Leader | running | 6 |           |
+-----+-----+-----+-----+-----+-----+
```

postgresql0 と postgresql2 の間のスイッチオーバーが 2023-09-13T18:00:00-03:00 で発生するようにスケジュールします。

```
$ patronictl -c postgres0.yml switchover batman --leader postgresql0 --candidate_
→ postgresql2 --scheduled 2023-09-13T18:00-03:00 --force
Current cluster topology
+ Cluster: batman (7277694203142172922) -+-----+-----+-----+
| Member      | Host          | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| postgresql0 | 127.0.0.1:5432 | Leader | running | 8 |           |
| postgresql1 | 127.0.0.1:5433 | Replica | streaming | 8 |          0 |
| postgresql2 | 127.0.0.1:5434 | Replica | streaming | 8 |          0 |
+-----+-----+-----+-----+-----+-----+
2023-09-13 14:18:11.20661 Switchover scheduled
+ Cluster: batman (7277694203142172922) -+-----+-----+-----+
| Member      | Host          | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| postgresql0 | 127.0.0.1:5432 | Leader | running | 8 |           |
| postgresql1 | 127.0.0.1:5433 | Replica | streaming | 8 |          0 |
| postgresql2 | 127.0.0.1:5434 | Replica | streaming | 8 |          0 |
+-----+-----+-----+-----+-----+-----+
Switchover scheduled at: 2023-09-13T18:00:00-03:00
                        from: postgresql0
                        to: postgresql2
```

5.2.16 守護トポロジー

あらすじ

```
topology
[ CLUSTER_NAME [, ... ] ]
[ --group CITUS_GROUP ]
[ { -W | { -w | --watch } TIME } ]
```

説明

`patronictl topology` は、Patroni クラスターとそのメンバーに関する情報をツリービューアプローチで表示します。

出力には次の情報が含まれます。

Cluster

Patroni クラスターの名前。

注釈: テーブルのヘッダーに表示されます。

System identifier

Postgres システム識別子。

注釈: テーブルのヘッダーに表示されます。

Member

パトローニのメンバーの名前。

注釈: この列の情報は、レプリケーション接続に関するメンバーのツリービューとして表示されます。

Host

メンバーがいるホスト。

Role

メンバーの現在の役割。

次のいずれかになります:

- **Leader** : 通常の Patroni クラスターの現在のリーダー。または
- **Standby Leader** : Patroni スタンバイクラスターの現在のリーダー。または
- **Sync Standby** : 同期モードが有効になっている Patroni クラスターの同期スタンバイ。または
- **Replica** : Patroni クラスターの通常のスタンバイ。

State

Patroni メンバーの Postgres の現在の状態。

考えられる状態の例をいくつか示します。

- **running** :Postgres が現在稼働している場合。
- **streaming** :レプリカと Postgres が現在プライマリノードから WAL をストリーミングしている場合。
- **in archive recovery** :レプリカと Postgres が現在アーカイブから WAL をフェッチしている場合。
- **stopped** :Postgres がシャットダウンされていた場合。
- **crashed** :Postgres がクラッシュした場合。

TL

Patroni メンバーの現在の Postgres タイムライン。

Lag in MB

Patroni メンバーとそのアップストリームの間のレプリケーションラグの量 (メガバイト単位)。

それに加えて、次の情報が出力に含まれる場合があります。

Group

Citus グループ ID。

注釈: テーブルのヘッダーに表示されます。

Citus クラスターの場合にのみ表示されます。

Pending restart

* は、一部の Postgres 構成を有効にするにはノードの再起動が必要であることを示します。空の値は、ノードを再起動する必要がないことを示します。

注釈: メンバー属性として表示されます。

ノードの再起動が必要な場合に表示されます。

Scheduled restart

Patroni メンバーが管理する Postgres インスタンスの再起動がスケジュールされたタイムスタンプ。空の値は、メンバーにスケジュールされた再起動がないことを示します。

注釈: メンバー属性として表示されます。

ノードにスケジュールされた再起動がある場合に表示されます。

Tags

Patroni メンバーに設定されたタグが含まれます。空の値は、タグが設定されていないか、タグがデフォルト値で設定されていることを示します。

注釈: メンバー属性として表示されます。

ノードにカスタムタグがある場合、またはデフォルト以外の値を持つデフォルトタグがある場合に表示されます。

Scheduled switchover

Patroni クラスターのスイッチオーバーがスケジュールされている場合のタイムスタンプ。

注釈: 表のフッターに表示されます。

スケジュールされたスイッチオーバーがある場合にのみ表示されます。

Maintenance mode

クラスター監視が現在一時停止されている場合。

注釈: 表のフッターに表示されます。

クラスターが一時停止されている場合にのみ表示されます。

パラメーター**CLUSTER_NAME**

Patroni クラスターの名前。

指定しない場合、`patronictl` は `scope` 構成が存在する場合、そこから取得しようとします。

--group

指定された Citus グループのメンバーに関する情報を表示します。

CITUS_GROUP は Citus グループの ID です。

-W

2 秒ごとに情報を自動的に更新します。

-w / --watch

指定した間隔で情報を自動的に更新します。

TIME は、更新間隔 (秒単位) です。

例

クラスター batman のトポロジーを表示します-- postgresql1 と postgresql2 は postgresql0 から複製されています。

```
$ patronictl -c postgres0.yml topology batman
+ Cluster: batman (7277694203142172922) ---+-----+-----+-----+
| Member          | Host              | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| postgresql0     | 127.0.0.1:5432    | Leader | running | 8  |           |
| + postgresql1   | 127.0.0.1:5433    | Replica | streaming | 8  | 0         |
| + postgresql2   | 127.0.0.1:5434    | Replica | streaming | 8  | 0         |
+-----+-----+-----+-----+-----+-----+
```

5.2.17 守護バージョン

あらすじ

```
version
[ CLUSTER_NAME [, ... ] ]
[ MEMBER_NAME [, ... ] ]
[ --group CITUS_GROUP ]
```

説明

patronictl version は、patronictl アプリケーションのバージョンを取得します。それに加えて、Patroni クラスターとそのメンバーに関するバージョン情報も含まれる場合があります。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

MEMBER_NAME

Patroni クラスターのメンバーの名前。

--group

指定された Citus グループを持つ Patroni クラスターを考えてみましょう。

CITUS_GROUP は Citus グループの ID です。

例

patronictl のバージョンのみを取得します。

```
$ patronictl -c postgres0.yml version
patronictl version 3.1.0
```

patronictl とクラスター batman のすべてのメンバーのバージョンを取得します。

```
$ patronictl -c postgres0.yml version batman
patronictl version 3.1.0

postgresql0: Patroni 3.1.0 PostgreSQL 15.2
postgresql1: Patroni 3.1.0 PostgreSQL 15.2
postgresql2: Patroni 3.1.0 PostgreSQL 15.2
```

patronictl と、クラスター batman のメンバー postgresql1 および postgresql2 のバージョンを取得します。

```
$ patronictl -c postgres0.yml version batman postgresql1 postgresql2
patronictl version 3.1.0

postgresql1: Patroni 3.1.0 PostgreSQL 15.2
postgresql2: Patroni 3.1.0 PostgreSQL 15.2
```


第6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るときに何が起こるかの定義もサポートします。2つの区別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。

6.1 ブートストラップ

PostgreSQL には、新しいクラスターを初期化するための `initdb` コマンドが用意されており、Patroni はデフォルトでそれ呼び出します。場合によっては、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ためのユーザー定義スクリプトの実行をサポートし、クラスターの名前とデータディレクトリへのパスなど、いくつかの必要な引数をそれらに提供します。これは、Patroni 構成の `bootstrap` セクションで構成されます。例えば：

```
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では、少なくとも `name` と `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。この場合、`method` パラメータは完全に省略できます。`command` は、絶対パス、または `patroni` コマンドの場所を基準とした相対パスのいずれか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 はクラスター固有の2つ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>	ブートストラップされ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 つの追加フラグの受け渡しを無効にするには、特別な `no_params` パラメータを `True` に設定します。

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ています。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ものとみなし、その後クリーンアップして初期化ロックを解放し、別のノードにブートストラップの機会を与えます。

`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 (または、PostgreSQL12 以上を実行している場合は、Postgres 構成でリカバリ設定を設定します)。通常、このようなりカバリ構成には、`promote` に設定された `recovery_target_timeline` とともに、少なくとも 1 つの `recovery_target_*` パラメータが含まれている必要があります。

`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` ファイルが存在する場合は削除しません (PostgreSQL<=11)。同様に、その場合、Patroni は既存の `recovery.signal` または `standby.signal` が存在する場合は削除しません。また、構成された回復設定 (PostgreSQL>=12) をオーバーライドすることもあります。これは、適切なりカバリ構成を生成する 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便利です。

これに加えて、カスタムブートストラップメソッド設定で通知された追加のキーと値のペアは、引数として `--name=value` 形式で `command` に渡されます。例えば：

```
bootstrap:
  method: <custom_bootstrap_method_name>
  <custom_bootstrap_method_name>:
    command: <path_to_custom_bootstrap_script>
    arg1: value1
    arg2: value2
```

構成された `command` が `--arg1=value1 --arg2=value2` コマンドライン引数を使用して追加で呼び出されるようにします。

注釈： ブートストラップメソッドはチェーンされず、プライマリメソッド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。

たとえば、次のような構成を使用して、Barman バックアップから新しい Patroni クラスターをブートストラップできます。

```
bootstrap:
  method: barman
  barman:
    keep_existing_recovery_conf: true
    command: patroni_barman_recover
    api-url: https://barman-host:7480
    barman-server: my_server
    ssh-command: ssh postgres@patroni-host
```

注釈: `patroni_barman_recover` では、バックアップ API を通じてリモート `barman recover` を実行できるように、Barman ホストに Barman と `pg-backup-api` の両方が構成されている必要があります。上の例では、使用可能なパラメータのサブセットを使用しています。 `patroni_barman_recover --help` を実行すると、詳細情報を取得できます。

6.2 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。欠点の 1 つは、実行中のリーダーノードが必要なことです。もう 1 つは、バックアップデータの「オンザフライ」圧縮がないことと、古いバックアップファイルのクリーンアップが組み込まれていないことです。WAL-E、`pgBackRest`、`Barman` などの他のバックアップソリューションを好んだり、単に独自のスクリプトを使用したりする人もいます。これらすべてのユースケースに対応するために、Patroni は新しいレプリカのクローンを作成するためのカスタムスクリプトの実行をサポートしています。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:
    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
```

(次のページに続く)

(前のページからの続き)

```

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

```

例:Barman

```

postgresql:
  create_replica_methods:
    - barman
    - basebackup
  barman:
    command: patroni_barman_recover
    api-url: https://barman-host:7480
    barman-server: my_server
    ssh-command: ssh postgres@patroni-host
  basebackup:
    max-rate: '100M'

```

注釈: `patroni_barman_recover` では、バックアップ API を通じてリモート `barman recover` を実行できるように、Barman ホストに Barman と `pg-backup-api` の両方が構成されている必要があります。上の例では、使用可能なパラメータのサブセットを使用しています。 `patroni_barman_recover --help` を実行すると、詳細情報を取得できます。

`create_replica_methods` は、使用可能なレプリカ作成メソッドとそ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、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をリストする構成ファイル内に個別のセクションを定義する必要があります。すべてのパラメータは `--name=value` 形式で渡されます。ユーザー定義のパラメータの他に、Patroni はクラスター固有のパラメータをいくつか提供します。

--scope	このレプリカが属するクラスター
--datadir	レプリカのデータディレクトリへのパス
--role	常に「レプリカ」
--connstring	クローン作成元のクラスターメンバーに接続するための接続文字列 (プライマリまたは他のレプリカ)。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` パラメータが定義されている場合、実行中のリーダーや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、接続文字列には空の文字列が渡されます。これは、以前に実行していたクラスターをバイナリバックアップから復元する場合に便利です。

特別な `keep_data` パラメータが定義されている場合、Patroni は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指示されます。

特別な `no_params` パラメータが定義されている場合、パラメータをカスタムコマンドに渡すことが制限されます。

`basebackup` メソッドは特殊なケースです。 `create_replica_methods` メソッドの中に明示的にリストすることは可能ですが、 `create_replica_methods` が空の場合に使用されます。このメソッドは、 `pg_basebackup` を使用して新しいレプリカを初期化し、 `clonefrom` タグを持つ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。設定なしで動作します。ただし、 `basebackup` 構成セクションを指定することは可能です。他のメソッド設定と同じルールが適用されます。つまり、長い (--付き) オプションのみを指定する必要があります。すべてのパラメータが意味をなすわけではありません。接続文字列をオーバーライドしたり、tar 圧縮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したりすると、patroni はそのバックアップからレプリカを作成できなくなります。 `basebackup` セクションに渡されるパラメータの名前または値に対しては検証は実行されません。また、WAL フォルダにシンボリックリンクが使用されている場合、レプリカの構築または再初期化後にシンボリックリンクが保持されるように、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の責任であることに注意してください。ただし、このオプションは v10 以降でのみサポートされています。

Basebackup パラメータは、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` の場合) のいずれかになります。)。これら 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。

6.3 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、リモートノードから複製されたスタンバイノードのみを含む「スタンバイクラスタ」を実行することです。このタイプのクラスター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複製する点を除けば、通常のクラスターリーダーとほぼ同じように動作します。
- カスケードレプリカ。スタンバイリーダーから複製されます。

スタンバイリーダーは DCS のリーダーロックを保持し、更新します。リーダーの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ーとその複製元のプライマリクラスターの間にはそ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。これらは、レプリケーション情報以外にはお互いに何も知りません。また、スタンバイクラスターは、プライマリクラスターの *Patroni リスト* または *Patroni リスト* 出力に表示されません。

柔軟性を高めるため、「standby_cluster」セクションに「create_replica_methods」キーを指定することで、クラスターが「スタンバイモード」のときにレプリカとリカバリ WAL レコードを作成する方法を指定できます。これは、クラスターが切り離されて通常のクラスターとして機能する場合のレプリカの作成とは異なります。これは、「postgresql」セクションの「create_replica_methods」によって制御されます。「standby」と「normal」の両方の「create_replica_methods」は、「postgresql」セクションのキーを参照します。

このようなクラスターを構成するには、patroni 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のブートストラップ中に 1 回だけ適用され、その後変更する唯一の方法は DCS を使用す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」が見つかることを期待しており、ベースバックアップ後に見つからない場合は起動しません。リモートプライマリがその「postgresql.conf」を別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は、プライマリクラスターで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。これは、スタンバイクラスターの実装によって自動的に実行されません。プライマリクラスターで Patroni の永続レプリケーションスロット機能を使用すると、`primary_slot_name` と同じ名前、または `primary_slot_name` が指定されていない場合はそのデフォルト値のレプリケーションスロットを維持できます。

第7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であるかを判断してください。

7.1 非同期モードの耐久性

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できます。プライマリサーバーに障害が発生したり、その他の理由で使用できなくなった場合、Patroni は十分に正常なスタンバイを自動的に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、プライマリ上の「フォークされたタイムライン」に残り、実質的に回復できなくなります [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` パラメータによって制御されます。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、フェイルオーバー時に失われるデータの量は、最悪の場合、トランザクションログの `maximum_lag_on_failover` バイトと、最後の `t1` 秒間に書き込まれた量 (例では `loop_wait / 2` 秒) によって制限されます。平均的な場合)。ただし、典型的な定常状態のレプリケーション遅延は 1 秒をはるかに下回ります。

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となる可能性があります。 `check_timeline` パラメータの値を `true` に変更することで、元のプライマリと同じタイムラインを持たないノードが新しいリーダーになるのを防ぐことができます。

7.2 PostgreSQL の同期レプリケーション

Postgres の `synchronous replication` を Patroni で使用できます。同期レプリケーションは、書き込みが成功して接続クライアントに戻る前にセカンダリに書き込まれたことを確認することで、クラスター全体の一貫性を確保します。同期レプリケーションのコスト: 書き込み時のスループットの低下。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きます。

ホスト型データセンター環境 (AWS、Rackspace、またはお客様が制御してい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性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、リーダーは事実上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、1つのホストに障害が発生した場合でも書き込みができるように、少なくとも3つの Postgres データノードを使用します。

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損失がゼロになることが保証されるわけではありません。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で同時に障害が発生した場合、すべてのトランザクションが含まれていない可能性がある3番目のノードが昇格されます。

7.3 同期モード

コミットされたトランザクションの損失が許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` がオンになっている場合、クライアント [2] に成功したコミットステータスを返した可能性のあるすべてのトランザクションがスタンバイに含まれていることが確実でない限り、Patroni はスタンバイを昇格しません。これは、一部のサーバーが使用可能であっても、システムが書き込みに使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、トランザクション損失が発生した場合でも、手動フェイルオーバーコマンドを使用してスタンバイを昇格させることができます。

`synchronous_mode` をオンにしても、あらゆる状況下で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。使用可能な適切なスタンバイがない場合でも、プライマリサーバーは書き込みを受け入れますが、そのレプリケーションは保証されません。このモードでプライマリに障害が発生した場合、スタンバイは昇格されません。プライマリであったホストが復帰すると、システム管理者が手動フェイルオーバーを実行しない限り、そのホストは自動的に昇格されます。この動作により、2ノードクラスターで同期モードが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、Patroni の次の反復が実行され、プライマリがスタンダアロンモードに切り替わるまで、コミットはブロックされます (書き込みの遅延は最悪の場合 `ttl` 秒、平均的な場合 `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業務から解放するよう信号を送ります。

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ことが絶対に必要な場合は、`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、使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できないようにします。欠点として、プライマリは (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) 書き込みに利用できないため、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書き込みリクエストがブロックされます。

`nosync` タグを `true` に設定すると、スタンバイが同期スタンバイにならないようにできます。これは、低速ネットワーク接続の背後にあり、同期スタンバイになるとパフォーマンスの低下を引き起こす可能性があるスタンバイに対して設定することをお勧めします。

同期モードは、PatroniREST インターフェイス経由でオン/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` を使用している場合でもトランザクションが失われる可能性があります。レプリケーションの確認を待機している間に 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として)、トランザクションの変更が他のバックエンドに表示されるようになります。このような変更はまだ複製されていないため、スタンバイ昇格の場合には失われる可能性があります。

7.4 同期レプリケーション係数

パラメータ `synchronous_node_count` は、同期スタンバイデータベースの数を管理するために Patroni によって使用されます。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` に基づいて同期スタンバイデータベースの正確な数を管理し、メンバーの参加および脱退に応じて DCS および `synchronous_standby_names` の状態を調整します。

7.5 同期モードの実装

同期モードの場合、Patroni は最新のプライマリデータベースと現在の同期スタンバイデータベースを含む DCS で同期状態を維持します。この状態は、次の不変条件を保証するための厳密な順序制約で更新されます。

- 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、ノードを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、この不変条件に違反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、それ自体を自動的に昇格させることはできません。

Patroni は、`synchronous_node_count` パラメータに基づいて 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、同期ステータスの削除を要求していない場合は、選択されたままになります。それ以外の場合は、レプリケーションで最も先にある同期に利用可能なクラスターメンバーが選択されます。

第8章 Watchdog 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、タイムラインの相違により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回避するために、Patroni は、DCS でリーダーキーの有効期限が切れた後、PostgreSQL がトランザクションコミットを受け付けなくする必要があります。通常の場合では、何らかの理由でリーダーロックの更新が失敗した場合、Patroni は PostgreSQL を停止することでこれを達成しようとします。ただし、次のようなさまざまな理由により、これが実行されない場合があります。

- Patroni は、バグ、メモリ不足、またはシステム管理者による誤った強制終了によりクラッシュしました。
- PostgreSQL のシャットダウンが遅すぎます。
- システムの高負荷、ハイパーバイザーによる VM の一時停止、またはその他のインフラストラクチャの問題により、Patroni が実行できません。

このような状況下で正しい動作を保証するために、Patroni は Watchdog デバイスをサポートしています。Watchdog 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に備えて、フェイルセーフの層が追加されます。

Patroni は、PostgreSQL をプライマリに昇格する前に、Watchdog をアクティブ化しようとします。Watchdog のアクティブ化が失敗し、Watchdog 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。リーダー選挙に参加することを決定するとき、Patroni は Watchdog 構成によってリーダーになれるのかも確認します。PostgreSQL を降格した後(手動フェイルオーバーなどにより)、Patroni は Watchdog を再び無効にします。Patroni が一時停止状態にある間は、Watchdog 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 Watchdog 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、システムが強制的にリセットされるまでに HA ループが完了するまでに少なくとも 15 秒かかります (`ttl - safety_margin - loop_wait`)。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設定されています。これは、ネットワークの問題などにより DCS が利用できない場合、Patroni と PostgreSQL がすべてのクライアント接続が終了する状態になるまでに少なくとも 5 秒 (`ttl - safety_margin - loop_wait - retry_timeout`) かかることを意味します。

安全マージンは、リーダーキーの更新と Watchdog キープアライブの間の時間として Patroni が予約する時間です。Patroni は、リーダーキーの更新を確認した後、すぐにキープアライブの送信を試みます。Patroni プロセスがまさに適切な瞬間に長時間中断された場合、Watchdog がトリガーされずにキープアライブが安全マージンを超えて遅延する可能性があります。その結果、リーダーキーの有効期限が切れる前に Watchdog がトリガーされない時間が発生し、保証が無効になります。あらゆる状況で Watchdog が確実にトリガーされるようにするには、`safety_margin` を -1 に設定して Watchdog のタイムアウトを `ttl // 2` に設定し、TTL の半分後に Watchdog 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` を増やすか、`loop_wait` と `retry_timeout` を減らす必要があります。

現在、Watchdog は LinuxWatchdog デバイスインターフェイスを使用した場合のみサポートされています。

8.1 Linux でのソフトウェア Watchdog のセットアップ

Patroni にアクセスできる場合、デフォルトの Patroni 設定は Linux 上で `/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ているソフトウェア Watchdog を使用するだけで十分に安全です。

ソフトウェア Watchdog 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
```

テストの場合は、`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、Watchdog はカーネルリングバッファに行を記録するだけで、`dmesg` 経由で表示されます。

Patroni は、Watchdog が正常に有効になると、Watchdog に関する情報をログに記録します。

第9章 クラスターの一時的停止/再開モード

9.1 目標

特定の状況では、Patroni はクラスターの状態を DCS に保持したまま、一時的にクラスターの管理から降りる必要があります。考えられる使用例は、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、クラスター上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、Patroni が不明な理由でノードが起動および停止されることが多く、一部のノードは一時的に昇格する可能性があり、プライマリを 1 つだけ実行するという前提に違反します。したがって、Patroni は、Pacemaker のメンテナンスモードと同等の機能を実装して、実行中のクラスターから「切り離す」ことができる必要があります。

9.2 実装

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

- 各ノードについて、DCS のメンバーキーが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、メンバーが実行中の場合、Patroni はメンバーノードに対して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なくなった場合 (つまり、手動で降格された場合)、Patroni はノードを昇格させずにロックを解放します。
- 手動によるスケジュール外の再起動、手動によるスケジュール外のフェイルオーバー/スイッチオーバー、および再初期化が許可されます。スケジュールされたアクションは許可されません。手動スイッチオーバーは、スイッチ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。
- 「並列」プライマリが Patroni によって検出された場合、警告が発行されますが、リーダーロックがなければプライマリは降格されません。
- クラスター内にリーダーロックがない場合は、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は、最初にロックを取得したプライマリが優先されます。プライマリがまったく存在しない場合、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。手動昇格により古いプライマリが降格したためにリーダーロックがない場合、昇格リクエストで指定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、レプリカを手動で昇格させた後)、Patroni は前のリーダーからストリーミングされ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 が停止している場合、Patroni は Postgres を起動しようとしません。Patroni が停止しても、管理している Postgres インスタンスは停止しません。
- Patroni は、他のクラスターメンバーを表していないレプリケーションスロット、または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。

9.3 ユーザーガイド

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

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

第10章 DCS フェールセーフモード

10.1 問題

Patroni は、リーダー選出のタスクを解決し、ネットワークの分割を検出するために、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、Postgres をプライマリとして実行できます。リーダーロックの更新が失敗した場合、Postgres は直ちに降格され、読み取り専用として開始されます。どの DCS が使用されているかに応じて、「問題」が発生する可能性が異なります。たとえば、Patroni にのみ使用される Etcd の場合、可能性はゼロに近いですが、K8sAPI(Etcd によってサポートされる)を使用すると、より頻繁に観察される可能性があります。

10.2 現在の実装の理由

リーダーロックの更新エラーは、次の2つの主な理由によって発生する可能性があります。

1. ネットワークのパーティショニング
2. DCS がダウンしている

一般に、単一ノードからこれら2つを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分割を想定します。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、リーダーのロックが期限切れになる前に古いプライマリが降格されます。

10.3 DCS フェールセーフモード

新しい特別なオプション `failsafe_mode` を導入します。これは、DCS `/config` キーに格納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。フェールセーフモードが有効になっており、バージョン/値/インデックスの不一致とは異なる理由で DCS のリーダーロックの更新が失敗した場合でも、PatroniRESTAPI 経由でクラスターの既知のすべてのメンバーにアクセスできれば、Postgres はプライマリとして実行し続ける可能性があります。

10.4 低レベルの実装の詳細

- DCS に `/failsafe` という名前の新しい永続キーを導入します。
- `/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、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` キーに示されているすべてのメンバーに接続し、すべてのレプリカがそれを認識すると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、受信した `POST /failsafe RESTAPI` リクエストを、プライマリがまだ生きていることを示す指標として使用します。この情報は `ttl` 秒間キャッシュされます。

10.5 よくある質問。

- 現在のプライマリが他のすべてのメンバーを参照しなければならないのはなぜですか?ここでは定足数に頼ることはできないのでしょうか?

これは素晴らしい質問です。問題は、定足数に対する考え方が DCS とパトローニの観点から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はなく、さらに重要なことに、そのようなルールを導入および強制するメカニズムがありません。Patroni ノードの大部分が分割されたネットワーク (プライマリを含む) の負けた部分に属し、少数のノードが勝った部分にある場合は、プライマリを降格する必要があります。このような状況を検出できるのは、他のすべてのメンバーをチェックする場合のみです。

- DCS がダウンしている間にノード/ポッドが終了した場合はどうなりますか?

DCS にアクセスできない場合は、「他のすべてのクラスターメンバーがアクセス可能か?」がチェックされます。ハートビートループのサイクルごと (`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 は読み取り専用以降格され、DCS が復元されるまで回復しません。

- DCS がダウンしている間に 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昇格することはできません。

- プライマリが DCS へのアクセスを失い、レプリカがアクセスできなかった場合はどうなりますか?

プライマリはフェールセーフコードを実行し、すべての既知のレプリカに接続します。これらのレプリカは、プライマリが生きていることを示す指標としてこの情報を使用し、DCS のリーダーロックの期限が切れた場合でもリーダーレースを開始しません。

- フェールセーフモードを有効にするにはどうすればよいですか？

`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 `failsafe_mode` のいずれかを使用できます。

第11章 Kubernetes での Patroni の使用

Patroni は、クラスターの状態を保存し、リーダーキーを管理するために Kubernetes オブジェクトを使用できます。これにより、整合性ストアなしで 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、これらは ``kubernetes.use_endpoints`` または ``PATRONI_KUBERNETES_USE_ENDPOINTS`` 環境変数で構成されます。

11.1 エンドポイントの使用

これは推奨モードですが、互換性の理由からデフォルトではオフになっています。これがオンの場合、Patroni は作成するそれぞれの「エンドポイント」の「メタデータ: アノテーション」フィールドにクラスター構成とリーダーキーを保存します。リーダーの変更は、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同時に更新されるため、「ConfigMaps」を使用する場合よりも安全です。

11.2 ConfigMap を使用する

このモードでは、Patroni はエンドポイントの代わりに ConfigMap を作成し、それらの ConfigMap 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 に対する更新と、それぞれのエンドポイントに対する更新の少なくとも2回の更新が必要です。

トラフィックを Postgres リーダーに送信するには、「role_label」(patroni 構成で構成)を含む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。

OpenShift で実行している場合など、場合によっては 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。

11.3 構成

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

11.3.1 役割ラベルをカスタマイズする

デフォルトでは、Patroni は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 (`role=master` など) を設定します。label のキーと値は、`kubernetes.role_label`、`kubernetes.leader_label_value`、`kubernetes.follower_label_value`、`kubernetes.standby_leader_label_value` によってカスタマイズできます。

デフォルトの役割ラベルからカスタムの役割ラベルに移行する場合、次の移行手順によってダウンタイムを短縮できることに注意してください。

1. `kubernetes.tmp_role_label` (``tmp_role`` など) を使用して、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: `kubernetes.leader_label_value=primary` を設定します)。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
```

11.4 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、権限の問題により PersistentVolume を使用できないことに注意してください。
- 永続ボリュームを使用できるフル機能の Docker イメージは [Spilo Project](#) にあります。
- Kubernetes を使用して実行される 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](#) プロジェクトを確認してください。Spilo クラスターを管理するための演算子パターンを実装します。

第12章 Citusのサポート

Patroni を使用すると、「マルチノード Citus」__クラスターの展開が非常に簡単になります。

12.1 TL;DR

従う必要がある簡単なルールはいくつかだけです。

1. **Citus** PostgreSQL のデータベース拡張機能は、すべてのノードで使用できる必要があります。サポートされる絶対最小 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のメリットを最大限に活用するには、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ノード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は、ワーカーノードからのクライアント証明書が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
```

その後は Patroni を起動するだけで残りはすべて処理されます。

1. `citus` 拡張子は `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` に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.local_hostname` GUC 値は、`localhost` からローカル PostgreSQL インスタンスに接続するために Patroni が使用している値に調整されます。PostgreSQL がこの値をリッスンしていない可能性があるため、この値は `localhost` とは異なる必要がある場合があります。
4. `citus.database` が自動的に作成され、続いて `CREATE EXTENSION citus` が作成されます。
5. 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、クロスノード通信が可能になります。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。

6. 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()` 関数を使用してそれらを `pg_dist_node` テーブルに追加します。
7. 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イルオーバー/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。

12.2 Patroni

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ています。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できません。

`patroni.yaml` に `citus` セクションがある場合、通常と比較して *Patroni* の動作に 2 つの大きな違いが生じます。

1. `list` と `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、それらがどの Citus グループに属しているかを示します。
2. すべての `patronictl` コマンドに、`--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` 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` または `patronictl` の場合は、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 *Patroni* リスト 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、出力は次のように変わります。

```
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         |
```

(次のページに続く)

(前のページからの続き)

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

12.3 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 はアプリケーションに対してスイッチオーバーをほぼ透過的に行う機会を提供します。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 では、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上の SQL トラフィックを「一時停止」することができます。その後、トラフィックがコーディネーター上に保持されている間にスイッチオーバーが発生し、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できるとすぐに再開されます。

ワーカークラスター上の *Patroni* 切り替え 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 |
```

(次のページに続く)

(前のページからの続き)

```
| 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 |
+-----+-----+-----+-----+-----+
| 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         |
+-----+-----+-----+-----+-----+
```

コーディネーター側では次のようになります:

```
# 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.
```

12.4 DCS を覗いてみる

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 では、単一の再帰読み取りリクエストで Citus クラスター全体をフェッチできるためです。Citus コーディネーターノードのみがワーカーノードを検出する必要があるため、ツリー全体を読み取ります。ワーカーノードは、自身のグループのサブツリーのみを読み取りますが、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可能性があります。

12.5 Kubernetes 上の Citus

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 `${scope}-${citus.group}-${type}` です。

すべての Kubernetes オブジェクトは、「ラベルセクター」__を使用して Patroni によって検出されます。そのため、Patroni&Citus および Endpoints/ConfigMap を含むすべての Pod には同様のラベルが必要であり、Kubernetes *settings* または *settings* を使用してラベルを使用するように Patroni を構成する必要があります。

Pods 環境変数を使用した Patroni 構成の例をいくつか示します。

1. コーディネータークラスターの場合

```

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の場合

```

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

```

(次のページに続く)

(前のページからの続き)

```

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"

```

お気づきかと思いますが、どちらの例にも `citus-group` ラベルが設定されています。このラベルにより、Patroni はオブジェクトが特定の Citus グループに属するものとして識別できるようになります。これに加えて、`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 ConfigMaps または Endpoints を作成すると、それらに `citus-group: ${env.PATRONI_CITUS_GROUP}` ラベルが自動的に付けられます。

```

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

```

Citus をサポートする Kubernetes 上での Patroni デプロイメントの完全な例は、Patroni リポジトリの `kubernetes` フォルダーにあります。

次の 2 つの重要なファイルがあります。

1. Dockerfile.citus
2. citus_k8s.yaml

12.6 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、「ドキュメント」__の Citus バージョンのアップグレードについてお読みください。プロセスには小さな変更が 1 つあります。アップグレードを実行する場合、`systemctl restart` の代わりに [Patroni 再起動](#) 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 ドキュメントと [PostgreSQL major upgrade](#) に関する Patroni ドキュメントで使用されているテクニック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、それらは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13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 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。

13.1 手順

既存の Postgres クラスターを Patroni 管理クラスターに変換する手順の概要を以下に示します。この手順では、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 構成を変更するつもりは「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ためのサンプル SQL コマンドがあり、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すでに存在する場合は、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てください。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 の YAML 構成ファイルを作成します。これには *Patroni configuration generation and validation tooling* を使用できます。

- **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` 構成項目を使用して既存のレプリケーションスロットを永続的なスロットとして構成することをお勧めします。`use_slots` が有効になっている場合、Patroni はメンバー間のレプリケーション用に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削除することに注意してください。ここで永久スロットを使用するという考えは、Patroni への移行の進行中に既存のスロットを維持できるようにすることです。詳細については、**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 がすで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 の「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ために、このステップを次のように分割することをお勧めします。

1. スタンバイノードを即時に再起動します。
2. メンテナンス時間枠内で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. で永続スロットを構成した場合は、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、1.2. コマンドを使用して永続スロットを `slots` 構成から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と、元のスロットが不要になったときに Patroni がクラスターから元のスロットを削除できるようになります。以下に、いくつかのスロットの `restart_lsn` を確認するクエリの例を示しますので、それらを比較できます。

```

-- 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

```

(次のページに続く)

(前のページからの続き)

```
WHERE slot_name IN (  
    'original_slot_for_member_x',  
    'slot_for_member_x'  
)
```


第14章 PostgreSQL バージョンのメジャーアップグレード

現時点でメジャーアップグレードを実行できる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. `patroni.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消去します。2 番目の方法は、`patronictl remove cluster-name` を実行することで実現できます。これは、`pg_upgrade` が実際に新しい PostgreSQL システム識別子を持つ新しいデータベースを作成する `initdb` を実行するために必要です。
5. 前の手順でクラスターの状態を消去した場合は、古いデータディレクトリから `patroni.dynamic.json` を新しいデータディレクトリにコピーするとよいでしょう。これは、以前に設定したいいくつかの PostgreSQL パラメータ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、`patroni.yml` を更新し、スタンバイノードの `data_dir` を消去します。
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。何をしようとしているのかわかっている場合は、スタンバイノードで `data_dir` をワイプする代わりに、<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。

14.1 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いうメッセージを出します。

`postgresql.conf` の `listen_addresses` と `port`、および `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すると、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。

これは、PostgreSQL 構成の管理方法に応じてさまざまな意味を持ちます。*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 は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しておく必要があります。

第15章 セキュリティに関する考慮事項

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

15.1 DCS の保護

Patroni と *Patroni* は両方とも、DCS との間でデータを保存および取得します。

DCS には機密情報は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、最初に保護されるべきものは DCS 自体です。

保護の詳細は、使用する DCS の種類によって異なります。サポートされている種類の 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。

一般に、すべての DCS 通信に対して TLS を有効にすることが推奨されます。

15.2 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 自体によって使用され、フェイルオーバー/スイッチオーバー/再初期化/再起動/リロードを実行するために *Patroni* ツールによって使用され、HTTP ヘルスチェックを実行するために HAProxy またはその他の種類のロードバランサーによって使用されます。モニタリングにも使用できます。

セキュリティの観点から、RESTAPI には安全なエンドポイント (GET リクエスト、情報の取得のみ) と安全でないエンドポイント (PUT、POST、PATCH および DELETE リクエスト、ノードの状態の変更) 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 を有効にしない限り、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`)、安全でない API 呼び出しのみ (`verify_client: optional`)、または API 呼び出しがない場合 (`verify_client: none`) のクライアント証明書検証の成功を必要とします。

ctl セクションのパラメーターは、クライアント (patroni と同じ構成を使用する ctl ツール) に対する TLS サーバー認証を有効にします。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の詳細については、ctl を参照してください。

PostgreSQL データベースを不正アクセスから適切に保護することは、このドキュメントの範囲外であり、<https://www.postgresql.org/docs/current/client-authentication.html> で説明されています。

第16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、同期または非同期 (`replication_modes`) のレプリケーションに基づいています。

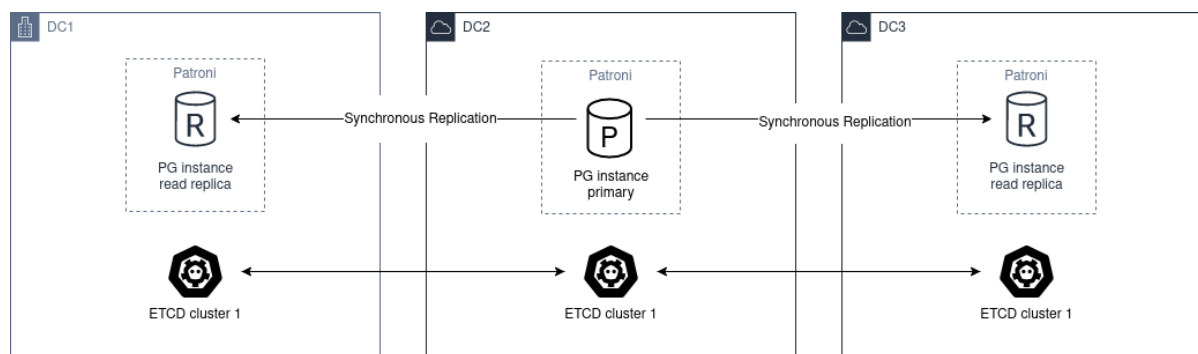
どちらの場合も、次の概念を明確にすることが重要です。

- Postgres は、先頭のキーを所有し、先頭のキーを更新できる場合にのみ、プライマリリーダーまたはスタンバイリーダーとして実行できます。
- etcd、ZooKeeper、または Consul ノードの奇数の数 (3 または 5) を実行する必要があります。

16.1 同期レプリケーション

ゾンドロップを自動的に許容できるマルチ DC クラスターを構築するには、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 のクラスターを別の DC 経由でデプロイする必要があります。各ゾーンに 1 つずつ、少なくとも 3 つのノードが必要です。

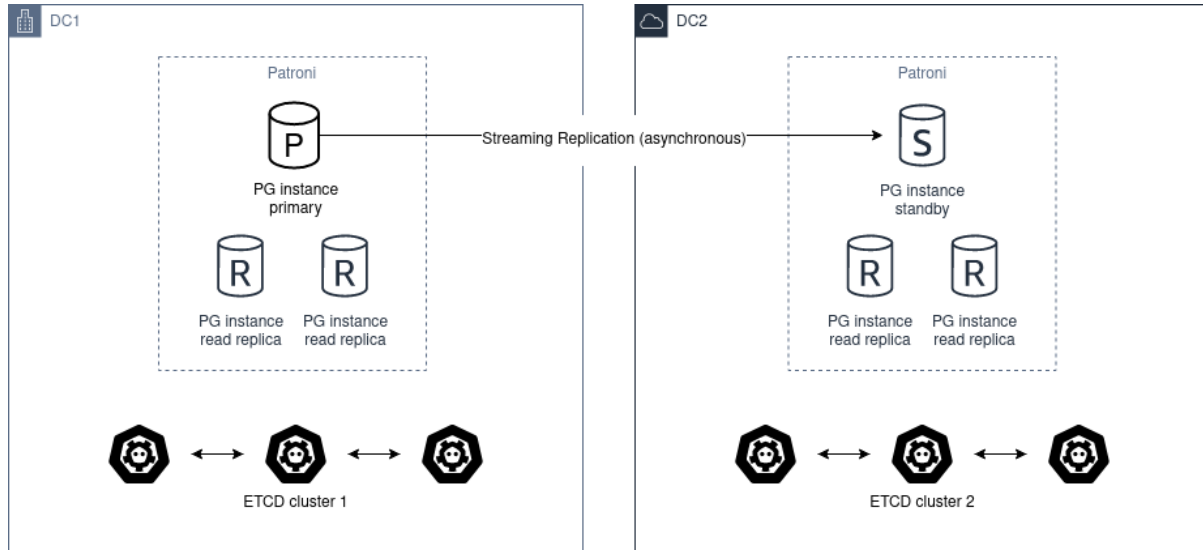
postgres に関しては、異なる DC に少なくとも 2 つのノードをデプロイする必要があります。次に、グローバル `synchronous_mode: true` に `synchronous_mode: true` を設定する必要があります。

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いずれかのノードを同期として選択します。

16.2 非同期レプリケーション

データセンターが2つしかない場合は、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* を実行する方がよいでしょう。最初のサイトがダウンしている場合は、*standby_cluster* を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 の状態を決して把握できないため、自動昇格は不可能です。

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「手動で昇格」する必要があります。

警告: ソースクラスターがまだ稼働中で、スタンバイクラスターを昇格すると、スプリットブレインが作成されます。

「初期」状態に戻したい場合、それを解決する方法は2つしかありません。

- *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させる前に、ソースクラスターが停止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。

これを行う前に、データベースを手動で調べて、DC1 と DC2 の間のネットワークが動作を停止した時点から DC1 のクラスターを手動で停止した時点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17章 よくある質問

このセクションでは、パトローニに関してよくある質問への回答を示します。各サブセクションでは、さまざまな種類の質問に焦点を当てます。

これがあなたの疑問のほとんどを解決するのに役立つことを願っています。さらに懸念がある場合、または予期しない問題に直面している場合は、[おしゃべり](#) および [おしゃべり](#) を参照して、ヘルプを取得したり問題を報告したりする方法を参照してください。

17.1 他の HA ソリューションとの比較

repmgr などの他のソリューションでは必要ないのに、**Patroni** では DCS ノードの別個のクラスターが必要なのはなぜですか？

HA ソリューションを実装するにはさまざまな方法があり、それぞれに長所と短所があります。

repmgr のようなソフトウェアは、ノード間で通信を実行して、いつアクションを実行するかを決定します。

一方、**Patroni** は DCS に保存された状態に依存します。DCS は、パトローニが何をすべきかを決定するための真実の情報源として機能します。

別個の DCS クラスターを使用するとアーキテクチャが肥大化する可能性があります。このアプローチにより、Postgres クラスターでスプリットブレインシナリオが発生する可能性も低くなります。

Postgres 管理に関して、**Patroni** と他の HA ソリューションの違いは何ですか？

Patroni は、Postgres クラスターの高可用性を管理するだけでなく、Postgres 自体も管理します。

Postgres ノードがまだ存在しない場合は、プライマリノードとスタンバイノードのブートストラップを処理し、ノードの Postgres 構成も管理します。Postgres ノードがすでに存在する場合、**Patroni** がクラスターの管理を引き継ぎます。

上記に加えて、**Patroni** には自己修復機能もあります。つまり、プライマリノードに障害が発生した場合、**Patroni** はレプリカにフェイルオーバーするだけでなく、新しいプライマリのレプリカとして元のプライマリに再参加しようとします。同様に、レプリカに障害が発生した場合、**Patroni** はそのレプリカに再参加しようとします。

これが、**Patroni** を「HA ソリューションのテンプレート」と呼ぶ方法です。単に物理レプリケーションを管理するだけでなく、Postgres 全体を管理します。

17.2 DCS

同じ etcd クラスターを使用して、2 つ以上の Patroni クラスターからのデータを保存できますか？

はい、できます！

Patroni クラスターに関する情報は、Patroni 設定の `namespace` および `scope` というプレフィックスが付いたパスの下に DCS に保存されます。

異なる Patroni クラスター間で競合する名前空間とスコープがない限り、同じ DCS クラスターを使用して複数の Patroni クラスターからの情報を保存できるはずです。

同じ DCS クラスターを指す異なる Patroni クラスターに対して `namespace` と `scope` の同じ組み合わせを使用しようとするとうなりますか？

同じ `namespace` と `scope` を使用しようとする 2 番目の Patroni クラスターは、DCS 内の同じ組み合わせに関連するが互換性のない Postgres システム ID を持つ情報を見つけるため、Postgres を管理できません。システム識別子の不一致により、Patroni は 2 番目のクラスターの管理を中止します。これは、2 番目のクラスターが別のクラスターを参照しており、ユーザーが Patroni の構成を誤っていると想定するためです。

同じ DCS クラスターを共有する異なる Patroni クラスターを扱う場合は、必ず異なる `namespace` / `scope` を使用してください。

DCS クラスターを失った場合はどうなりますか？

DCS は、基本的に Patroni クラスターのステータスと動的構成を保存するために使用されます。

その最初の結果は、[DCS フェールセーフモード](#) が有効になっていない限り、その DCS に依存するすべての Patroni クラスターが読み取り専用モードになることです。

DCS クラスターを紛失した場合はどうすればよいですか？

DCS クラスターが失われた場合、考えられる結果は 3 つあります。

1. DCS クラスターは完全に回復されています。これには、Patroni 側からのアクションは必要ありません。DCS クラスターが回復すると、Patroni も回復できるはずです。
2. DCS クラスターは適切な場所に再作成され、エンドポイントは同じままになります。Patroni 側では変更は必要ありません。
3. 新しい DCS クラスターが異なるエンドポイントで作成されます。各 Patroni ノードの Patroni 構成で DCS エンドポイントを更新する必要があります。

2. または 3. のシナリオに直面した場合、Patroni はクラスターの現在のステータスに基づいてステータス情報を再度作成し、Postgres データディレクトリ内に保存されている `patroni.dynamic.json` という名前のバックアップファイルに基づいて DCS 上に動的構成を再作成します。Patroni クラスターの各メンバー。

DCS クラスターの過半数を失った場合はどうなりますか？

DCS が応答しなくなるため、Patroni は現在の読み取り/書き込み Postgres ノードを降格させます。

覚えておいてください: Patroni は DCS の状態に依存してクラスターに対してアクションを実行します。

[DCS フェールセーフモード](#) を使用すると、この状況を軽減できます。

17.3 Patroni

Patroni ホストで *Patroni* を実行する必要がありますか？

いいえ、その必要はありません。

Patroni ホストにアクセスできる場合は、Patroni ホストで *Patroni* を実行すると便利です。これは、*patroni* エージェントからのまったく同じ構成ファイルを *Patroni* アプリケーションに使用できるためです。

ただし、*Patroni* は基本的にクライアントであり、リモートマシンから実行できます。Patroni メンバーの DCS および RESTAPI にアクセスできるように、十分な構成を提供するだけです。

Patroni メンバーの 1 人からの情報が *Patroni リスト* コマンドの出力から消えたのはなぜですか？

Patroni リスト で示される情報は DCS の内容に基づいています。

メンバーに関する情報が DCS から消えた場合は、そのノード上の Patroni エージェントが実行されていないか、DCS と通信できない可能性が非常に高くなります。

メンバーは情報を更新できないため、最終的に DCS からの情報の有効期限が切れ、その結果、メンバーは *Patroni リスト* の出力に表示されなくなります。

Patroni リスト コマンドの出力にある Patroni メンバーの 1 つに関する情報が最新ではないのはなぜですか？

Patroni リスト で示される情報は DCS の内容に基づいています。

デフォルトでは、その情報は Patroni によっておよそ `loop_wait` 秒ごとに更新されます。つまり、すべてが正常に機能している場合でも、DCS に保存されている情報に最大 `loop_wait` 秒の「遅延」が発生する可能性があります。

ただし、それはルールではないことに注意してください。Patroni によって実行される一部の操作により、DCS 情報が即座に更新されます。

17.4 構成

動的構成とローカル構成の違いは何ですか？

動的構成 (またはグローバル構成) は、DCS に保存される構成であり、Patroni クラスターのすべてのメンバーに適用されます。これは主に構成を保存する場所です。

ノードに固有の設定、またはグローバル構成を上書きしたい設定は、目的の Patroni メンバーにのみローカル構成として設定する必要があります。そのローカル構成は、構成ファイルまたは環境変数を通じて指定できます。

詳細については、[パトローニの構成](#) を参照してください。

Patroni の構成の種類と優先順位は何ですか？

種類は次のとおりです。

- 動的構成: すべてのメンバーに適用されます。
- ローカル構成: ローカルメンバーに適用され、動的構成をオーバーライドします。

- 環境構成: ローカルメンバーに適用され、動的構成とローカル構成の両方がオーバーライドされます。

Note: 一部の PostgresGUC は、グローバルに、つまり動的構成を通じてのみ設定できます。それに加えて、Patroni がハードコードされた値を強制する GUC もあります。

詳細については、[パトローニの構成](#) を参照してください。

Patroni 設定ファイルの作成に役立つ機能はありますか？

はいあります。

`patroni --generate-sample-config` または `patroni --generate-config` コマンドを使用して、それぞれサンプルの Patroni 構成または既存の Postgres インスタンスに基づく Patroni 構成を生成できます。

詳細については、[Patroni 構成のサンプル](#) および [Patroni 構成のサンプル](#) を参照してください。

`bootstrap.dcs` 設定でパラメータを変更しましたが、Patroni は変更をクラスタメンバーに適用していません。なにが問題ですか？

`bootstrap.dcs` で構成された値は、新しいクラスターをブートストラップする場合にのみ使用されます。これらの値はブートストラップ中に DCS に書き込まれます。

ブートストラップフェーズが終了した後は、DCS を介してのみ動的構成を変更できます。

詳細については、次の質問を参照してください。

動的構成を変更するにはどうすればよいですか？

DCS の設定を変更する必要があります。これは、次のいずれかの方法で実現されます。

- [patronictl 編集構成](#) ; または
- PATCH に対する PATCH リクエスト。

ローカル設定を変更するにはどうすればよいですか？

対応する Patroni メンバーの構成ファイルを変更し、SIHGUP で Patroni エージェントに通知する必要があります。これは、次のいずれかのアプローチを使用して実行できます。

- POST リクエストを RESTAPI POST に送信します。または
- [パトロニトルリロード](#) を実行します。または
- SIGHUP を使用してローカルで Patroni プロセスに信号を送ります。
 - `systemd` を通じて Patroni を起動した場合は、コマンド `systemctl reload PATRONI_UNIT.service` を使用できます。PATRONI_UNIT は Patroni サービスの名前です。または
 - 他の方法で Patroni を起動した場合は、`patroni` プロセスを特定し、`kill -s HUP PID` を実行する必要があります。PID は `patroni` プロセスのプロセス ID です。

Note: Note: によるリロードが機能しない場合があります。

- 期限切れの RESTAPI 証明書: `-k` の `-k` オプションを使用することで、これを軽減できます。
- 間違った資格情報: たとえば、構成ファイル内の `restapi` または `ctl` 資格情報を変更し、同じ構成ファイルを Patroni と `restapi` に使用する場合。

環境構成を変更するにはどうすればよいですか？

環境設定は、起動時に Patroni によってのみ読み取られます。

これを念頭に置いて、環境構成を変更した場合は、対応する Patroni エージェントを再起動する必要があります。

クラスター内でフェイルオーバーが発生しないように注意してください。一時停止をチェックしてみるとよいでしょう。

リロードが必要な PostgresGUC を変更するとどうなりますか？

前の質問で説明したように動的構成またはローカル構成を変更すると、Patroni が Postgres 構成の再ロードを処理します。

再起動が必要な PostgresGUC を変更するとどうなりますか？

Patroni は、影響を受けるメンバーに `pending restart` のフラグを付けます。

メンバーをいつ、どのように再起動するかを決定するのはあなた次第です。これは、次のいずれかの方法で実現できます。

- Patroni 再起動; または
- POST に対する POST リクエスト。

Note: 一部の PostgresGUC では、Postgres ノードを再起動する順序に関して特別な管理が必要です。詳細については、**Note:** を参照してください。

Patroni 設定における etcd と etcd3 の違いは何ですか？

etcd は etcd の API バージョン 2 を使用し、etcd3 は etcd の API バージョン 3 を使用します。

API バージョン 2 で保存された情報は API バージョン 3 では管理できず、その逆も同様であることに注意してください。

次の理由から、etcd ではなく etcd3 を構成することをお勧めします。

- API バージョン 2 は、Etcdv3.4 以降、デフォルトで無効になっています。
- API バージョン 2 は Etcdv3.6 では完全に削除されます。

Patroni 構成で use_slots を有効にしましたが、クラスターメンバーがしばらくオフラインになると、そのメンバーが使用していたレプリケーションスロットが上流ノードで削除されます。その問題を回避するにはどうすればよいでしょうか？

メンバーに対して永続的な物理レプリケーションスロットを構成できます。

Patroni 3.2.0 以降、メンバースロットを Patroni が管理する永久スロットとして持つことができるようになりました。

Patroni はすべてのノードに永続的な物理スロットを作成し、スロットを削除しないようにするだけでなく、メンバーによって消費された LSN に従ってすべてのノードのスロットの LSNを進めます。

後で、対応するメンバーを削除する場合は、**your responsibility** で永続スロット構成を調整します。そうでない場合、Patroni はスロットを永久に保持します。

3.2.0 よりも古い Patroni の **Note:** では、メンバースロットを永続的な物理スロットとして構成できますが、それらは現在のリーダーでのみ管理されます。つまり、フェイルオーバー/スイッチオーバーの場合、これらのスロットは新しいリーダー上に作成されますが、存在しないノードのすべての WAL セグメントがあることは保証されません。

Note: Patroni 3.2.0 を使用した場合でも、小さな競合状態が発生する可能性があります。最初は、スロットがレプリカ上に作成されるときに、リーダー上の同じスロットよりも先に作成される可能性があります。誰もスロットを使用していない場合、フェイルオーバー後に一部のファイルが失われる可能性があります。これを念頭に置いて、必要な WAL の復元や PITR の実行を可能にする継続的アーカイブを構成することをお勧めします。

loop_wait、retry_timeout、ttl の違いは何ですか？

Patroni は、HA サイクルと呼ばれるものを時々実行します。各 HA サイクルでは、クラスター上で一連のチェックを実行してその健全性を判断し、ステータスに応じてスタンバイへのフェイルオーバーなどのアクションを実行します。

loop_wait は、HA チェックの新しいサイクルを実行する前に Patroni がスリープする時間を秒単位で決定します。

retry_timeout は、DCS および Postgres での再試行操作のタイムアウトを設定します。たとえば、DCS が retry_timeout 秒を超えて応答しない場合、Patroni はセキュリティアクションとしてプライマリノードを降格する可能性があります。

ttl は、DCS の leader ロックのリース時間を設定します。クラスターの現在のリーダーが ttl を超えて HA サイクル中にリースを更新できない場合、リースは期限切れになり、クラスター内で leader race がトリガーされます。

Note: これらの設定を変更するときは、Patroni がドキュメントの **Note:** セクションで説明されているルールと最小値を強制することに注意してください。

17.5 Postgres 管理

Postgres 構成で PostgresGUC を直接変更できますか？

可能ですが、それは避けるべきです。

Postgres 構成は Patroni によって管理されており、構成ファイルを編集しようとする、最終的に構成ファイルが上書きされる可能性があるため、Patroni によって失敗する可能性があります。

Patroni による管理を克服するために利用できるオプションがいくつかあります。

- \$PGDATA/postgresql.base.conf を通じて PostgresGUC を変更します。または
- postgresql.base.conf の代わりに使用される postgresql.custom_conf を定義して、外部で管理できるようにします。または
- ALTER SYSTEM / ALTER DATABASE / ALTER USER を使用して GUC を変更します。

詳細については、[重要なルール](#) セクションを参照してください。

いずれの場合も、すべての Postgres 構成を Patroni を通じて管理することをお勧めします。これにより、管理が集中化され、必要なときに Patroni をデバッグしやすくなります。

Postgres ノードを直接再起動できますか？

いいえ、**not** で Postgres を直接管理する必要があります。

Patroni を使用せずに Postgres サーバーをバウンスしようすると、クラスターがフェイルオーバーに直面する可能性があります。

Postgres サーバーを管理する必要がある場合は、Patroni が公開している方法で管理してください。

Patroni は既存の Postgres クラスターの管理を引き継ぐことができますか？

はい、できます！

詳細な手順については、[スタンドアロンを Patroni クラスターに変換する](#) を参照してください。

Patroni は Postgres をどのように管理していますか？

Patroni は、`pg_ctl` や `postgres` などの Postgres バイナリを実行することにより、Postgres の起動と停止を処理します。

これを念頭に置いて、postgres クラスターを管理できる他のソース (systemd ユニットなど) を **MUST** で無効にします。 `postgresql.service`。Patroni のみがクラスター内の Postgres インスタンスを開始、停止、昇格できるようにする必要があります。そうしないと、スプリットブレインシナリオが発生する可能性があります。たとえば、プライマリとして実行されているノードに障害が発生し、ユニット `postgresql.service` が有効になっている場合、Postgres がバックアップされてスプリットブレインが発生する可能性があります。

17.6 概念と要件

Patroni の一部を構成するアプリケーションはどれですか？

Patroni は基本的に次の 2 つのアプリケーションを同梱しています。

- `patroni` : これは、Postgres ノードの管理を担当する Patroni エージェントです。
- `patronictl` : これは、Patroni クラスターと対話する (スイッチオーバー、再起動、構成の変更などを実行する) ために使用されるコマンドラインユーティリティです。詳細については、`patronictl` を参照してください。

Patroni の standby cluster とは何ですか？

これは、プライマリ Postgres ノードが実行されていないクラスターです。つまり、クラスター内に読み取り/書き込みメンバーがありません。

これらの種類のクラスターは、別のクラスターからデータをレプリケートするために存在し、通常、データセンター間でデータをレプリケートする場合に役立ちます。

クラスター内には、リモート Postgres ノードからの変更のレプリケーションを担当するスタンバイとなるリーダーが存在します。次に、そのようなリーダーメンバーからのカスケードレプリケーションで構成された一連のスタンバイが存在します。

Note: スタンバイクラスターは、複製元のソースクラスターについて何も知りません。WAL ストリーミングの代わりに `restore_command` を使用することもでき、完全に独立した DCS クラスターを使用することもあります。

詳細については、[スタンバイクラスター](#) を参照してください。

Patroni の leader とは何ですか？

Patroni の leader は、クラスターのコーディネーターのようなものです。

通常の Patroni クラスターでは、leader が読み取り/書き込みノードになります。

スタンバイ Patroni クラスターでは、`leader` (別名 `standby leader`) がリモート Postgres ノードからのレプリケーションを担当し、それらの変更をスタンバイクラスターの他のメンバーにカスケードします。

Patroni ではクラスター内に最小限の Postgres ノードが必要ですか?

いいえ、Patroni は任意の数の Postgres ノードで実行できます。

覚えておいてください: Patroni は DCS から切り離されています。

パトローニで `pause` は何を意味しますか?

一時停止は Patroni によって公開される操作であるため、ユーザーは Postgres 管理に関して Patroni に手を引くように要求できます。

これは主に、クラスターでメンテナンスを実行する必要がある、プライマリを停止したときにスタンバイにフェイルオーバーするなど、Patroni が HA に関連する決定を下すことを避けたい場合に役立ちます。

詳細については、[クラスターの一時停止/再開モード](#) を参照してください。

17.7 自動フェイルオーバー

Patroni の自動フェイルオーバーメカニズムはどのように機能しますか?

Patroni の自動フェイルオーバーは、`leader race` と呼ばれるものに基づいています。

Patroni はクラスターのステータスを DCS に保存します。その中には、クラスターの現在の `leader` である Patroni メンバーの名前を保持する `leader` ロックがあります。

その `leader` ロックには有効期限が関連付けられています。リーダーノードが `leader` ロックのリースを期限内に更新できなかった場合、キーは最終的に DCS から期限切れになります。

`leader` ロックの有効期限が切れると、Patroni で `leader race` と呼ばれるものがトリガーされます。つまり、すべてのノードがチェックの実行を開始して、`leader` の役割を引き継ぐ最適な候補であるかどうかを判断します。これらのチェックの一部には、他のすべての Patroni メンバーの RESTAPI への呼び出しが含まれます。

`leader` ロックを引き継ぐための最良の候補であると判断したすべての Patroni メンバーは、そうしようとします。`leader` ロックを取得できる最初の Patroni メンバーは、自身を読み取り/書き込みノード (または `standby leader`) に昇格し、他のメンバーはこれに従うように構成されます。

Patroni クラスターで自動フェイルオーバーを一時的に無効にすることはできますか?

はい、できます！

これを実現するには、クラスターを一時的に停止します。これは通常、メンテナンスを実行する場合に役立ちます。

クラスターの自動フェイルオーバーを再開したい場合は、一時停止を解除するだけです。

詳細については、[クラスターの一時停止/再開モード](#) を参照してください。

17.8 ブートストラップとスタンバイの作成

Patroni はどのようにしてプライマリ Postgres ノードを作成しますか? スタンバイ Postgres ノードについてはどうですか?

デフォルトでは、Patroni は `initdb` を使用して新しいクラスターをブートストラップし、`pg_basebackup` を使用して leader メンバーのコピーからスタンバイノードを作成します。

カスタムブートストラップメソッドとカスタムレプリカ作成メソッドを作成することで、その動作をカスタマイズできます。

カスタムメソッドは通常、`pgBackRest` や `Barman` などのバックアップツールによって作成されたバックアップを復元する場合に役立ちます。

詳細については、[ブートストラップ](#) および [ブートストラップ](#) を参照してください。

17.9 監視

Patroni クラスターを監視するにはどうすればよいですか?

Patroni は、[パトローニ RESTAPI](#) でいくつかの便利なエンドポイントを公開しています。

- `/metrics`: Prometheus で使用できる形式でモニタリングメトリクスを公開します。
- `/patroni`: クラスターのステータスを JSON 形式で公開します。ここに表示される情報は、`/metrics` エンドポイントによって表示される情報と非常に似ています。

これらのエンドポイントを使用して、監視チェックを実装できます。

第18章 リリースノート

18.1 バージョン 3.2.2

Bugfixes

- DCS がワイプされたときにレプリカ復元で初期化キーを実行させないようにしました。(AlexanderKukushkin)

これは、Patroni がスタンドアロンの PG クラスターを引き継ぐことになっているメソッドで発生していました。

- Consul から更新されたばかりの同期キーを取得するときに一貫性のある読み取りを使用するようになりました。(AlexanderKukushkin)

Consul は、更新したばかりのキーの `ModifyIndex` をすぐに取得するためのインターフェイスを提供していないため、明示的な読み取り操作を実行する必要があります。デフォルトでは古い読み取りが許可されているため、古いバージョンのキーを取得することがありました。

- 再起動が必要なパラメータが元の値にリセットされた場合、Postgres 設定をリロードします。(PolinaBungina)

以前は、Patroni は構成を更新せず、`pending_restart` をリセットするだけでした。

- 同期モードで非同期候補へのフェイルオーバーを実行する際の確認プロンプトメッセージの誤った反転ロジックを修正しました。(PolinaBungina)

この問題は `patronictl` にのみ存在しました。

- `patronictl` のフェイルオーバー候補からリーダーを除外しました (PolinaBungina)

クラスターが正常な場合、既存のリーダーへのフェイルオーバーは何も行われません。

- Citus データベースと拡張機能をベキ等に作成する (AlexanderKukushkin、ZhaoJunwang)

Citus データベースにさらに依存関係を追加する必要がある場合に備えて、`post_bootstrap` スクリプトでそれらを作成できるようになります。

- 矛盾する `nofailover` タグをフィルタリングしないでください (PolinaBungina)

ノードに設定された `{nofailover: false, failover_priority: 0}` 設定では、ノードはレースに参加できませんでしたが、`nofailover` タグが優先されるため、レースに参加できるはずでした。

- PyInstaller のフリーズ問題を修正しました (SophiaRuan)

`freeze_support()` は `argparse` の後に呼び出されたため、Patroni は Postgres を起動できませんでした。

- `patronictl` および Citus 設定の設定ジェネレーターのバグを修正しました (IsraelBarthRubio)

これにより、環境変数を介して設定された `patronictl` および Citus 構成パラメーターが、生成された構成に書き込まれることが妨げられました。

- 実行中のスタンバイに参加するときに、リカバリ GUC と一部の Patroni 管理パラメータを復元します (AlexanderKukushkin)

Patroni は、内部構造の 1 つに `port` が欠落しているというエラーが発生し、Postgresv12 以降の再起動に失敗していました。

- `pending_restart` フラグ周りの修正 (PolinaBungina)

`recovery_target_action = promote` を使用したカスタムブートストラップの場合、または誰かが `ALTER SYSTEM` などを使用して `hot_standby` または `wal_log_hints` を変更した場合は、`pending_restart` を公開しないでください。

18.2 バージョン 3.2.1

Bugfixes

- `patronictl` の `--format` 引数に受け入れられる値を制限しました (AlexanderKukushkin)

以前は任意の文字列を受け入れ、値が認識されない場合は出力を生成していませんでした。

- リーダーキーを解放する前に、レプリカノードがシャットダウン時にチェックポイント LSN を受信したことを確認します。 (AlexanderKukushkin)

以前は、場合によっては、CHECKPOINT が後に続く SWITCH レコードの LSN を使用していました (アーカイブモードが有効な場合)。その結果、以前のプライマリは `pg_rewind` を実行する必要がある場合がありますが、データ損失は発生しませんでした。

- ノード名の一意性チェックを実行するときに実際の HTTP リクエストを実行します。 (AlexanderKukushkin)

Patroni をコンテナで実行する場合、トラフィックが `docker-proxy` を使用してルーティングされる可能性があります。これは、ポートをリッスンして受信接続を受け入れます。それが誤検知を引き起こしていました。

- Etcdv2 での Citus サポートを修正しました (AlexanderKukushkin)

Patroni は Etcdv2 を使用した新しい Citus クラスターのデプロイに失敗していました。

- Postgresv16+での `pg_rewind` の動作を修正しました。 (AlexanderKukushkin)

`pg_waldump` のエラーメッセージ形式が v16 で変更されたため、不要な場合でも Patroni によって `pg_rewind` が呼び出されるようになりました。

- カスタムブートストラップのバグを修正しました (AlexanderKukushkin)

Patroni は、ブートストラップコマンドそのものである `--command` 引数を誤って適用していました。

- RESTAPI ヘルスチェックエンドポイントの問題を修正しました (SophiaRuan)

接続が適切に閉じられなかったために、Postgres の再起動後に Postgres の `unknown` 状態が返される可能性があります。

- `postgres --describe-config` の出力結果をキャッシュする (Waynerv)

これらは、PostgreSQL 構成の検証にどの GUC が利用できるかを判断するために使用され、Patroni の実行中にこのリストが変更されることは想定されていません。

18.3 バージョン 3.2.0

Deprecation notice

- `bootstrap.users` のサポートはバージョン 4.0.0 で削除されます。新しいクラスターのデプロイ後にユーザーを作成する必要がある場合は、`bootstrap.post_bootstrap` フックを使用してください。

Breaking changes

- `loop_wait + 2*retry_timeout <= ttl` ルールを適用し、可能な最小限の値をハードコードします。(AlexanderKukushkin)

最小値: `loop_wait=2`、`retry_timeout=3`、`ttl=20`。値が小さい場合、またはルールに違反している場合は、値が調整され、警告が Patroni ログに書き込まれます。

New features

- フェイルオーバーの優先順位 (MarkPekala)

`tags.failover_priority` の助けを借りて、リーダーレース中にノードをより優先させることができるようになりました。詳細については、ドキュメント (ref タグ) を参照してください。

- `patroni --generate-config [--dsn DSN]` と `patroni --generate-sample-config` を実装しました (PolinaBungina)

実行中の PostgreSQL クラスターの構成ファイル、または新しい Patroni クラスターのサンプル構成ファイルを生成できます。

- PatroniRESTAPI 用に Postgres への専用接続を使用する (AlexanderKukushkin)

システムにストレスがかかっている場合に、メインのハートビートループがブロックされるのを避けるのに役立ちます。

- ノードの `name` を使用して一部のエンドポイントを強化します (sskserk)

モニタリングエンドポイントの場合は `name` が `scope` の隣に追加され、メトリクスエンドポイントの場合は `name` がタグに追加されます。

- 厳密なフェイルオーバー/スイッチオーバーの違いを確保する (PolinaBungina)

ログメッセージをより正確にし、正常な同期クラスター内の非同期ノードへのフェイルオーバーを許可します。

- 永続的な物理レプリケーションスロットが永続的な論理スロットと同様に動作するようにしました。(AlexanderKukushkin)

リーダーになることが許可されているすべてのノードに永続的な物理レプリケーションスロットを作成し、`pg_replication_slot_advance()` 関数を使用してスタンバイノードのスロットの `restart_lsn` を進めます。

- `patronictl` の `--dcs` 引数を通じて名前空間を指定する機能を追加しました。(IsraelBarthRubio)

設定ファイルなしで `patronictl` を使用すると便利です。

- カスタムブートストラップ構成に追加パラメータのサポートを追加しました (IsraelBarthRubio)

以前は、カスタム引数を `command` に追加することしかできませんでしたが、現在はそれらをマッピングとしてリストできるようになりました。

Improvements

- `citus.local_hostname` GUC を、Patroni が Postgres に接続するために使用するのと同じ値に設定します。(AlexanderKukushkin)

Citus がローカル Postgres に接続したい場合があります。デフォルトでは `localhost` が使用されますが、常に使用できるわけではありません。

Bugfixes

- スタンバイクラスターの `synchronous_mode` 設定を無視する (PolinaBungina)

Postgres はカスケード同期レプリケーションをサポートしておらず、`synchronous_mode` がスタンバイクラスターでのスイッチオーバーを中断していたことを無視していません。

- `on_reload` コールバックの `SIGCHLD` を処理しました (AlexanderKukushkin)

そうしないとゾンビプロセスが生成され、次の `on_reload` が実行されるときにのみ取得されます。

- Etcdv3 を使用する場合の `AuthOldRevision` エラーを処理しました (AlexanderKukushkin, KennyDo)

このエラーは、Etcd が JWT を使用するよう構成されている場合、および Etcd のユーザーデータベースが更新されたときに発生します。

18.4 バージョン 3.1.2

Bugfixes

- `wal_keep_size` チェックのバグを修正しました (AlexanderKukushkin)

`wal_keep_size` は通常ユニットを持つ GUC ですが、Patroni はその値を `int` にキャストできませんでした。その結果、`bootstrap.dcs` の値はその後 `/config` キーに書き込まれませんでした。

- `/sync` キーと `synchronous_standby_names` の間の不一致を検出して解決しました。(AlexanderKukushkin)

通常、Patroni は非常に特定の順序で `/sync` と `synchronous_standby_names` を更新しますが、バグが発生した場合、または誰かが `synchronous_standby_names` を手動でリセットした場合、Patroni は一貫性のない状態になってしまいました。その結果、非同期ノードでフェイルオーバーが発生する可能性があります。

- 実行中の Postgres に参加するときに GUC の値を読む (AlexanderKukushkin)

`pause` で再起動すると、Patroni は `postgresql.conf` から `synchronous_standby_names` GUC を破棄していました。これを解決し、同様の問題を回避するために、Patroni は、すでに実行中の Postgres に参加している場合に GUC の値を読み取ります。

- ノードの一意性をチェックする際の煩わしい警告を黙らせました (AlexanderKukushkin)

Patroni がすぐに再起動されると、`WARNING` メッセージが `urllib3` によって生成されます。

18.5 バージョン 3.1.1

Bugfixes

- プロモート時にフェイルセーフ状態をリセット (ChenChangAo)

フェールセーフモードがアクティブになった直後にスイッチオーバー/フェールオーバーが発生した場合、フェールセーフが非アクティブになった後、新しく昇格したプライマリは自らを降格していました。

- `patronictl` の無用な警告をサイレントにしました (AlexanderKukushkin)

`patronictl` が Patroni と同じ `patroni.yaml` ファイルを使用し、`PGDATA` ディレクトリにアクセスできる場合、グローバル構成の不正な値に関する煩わしい警告が表示されていた可能性があります。

- 特殊なケースに対して同期モードを明示的に有効にしました。 (AlexanderKukushkin)

プライマリからストリーミングするレプリカがない場合、同期モードは事実上アクティブ化されません。

- 0 整数値の検証に関するバグを修正しました (IsraelBarthRubio)

ほとんどの場合、警告が表示されるだけで、問題は発生しませんでした。

- スタンバイクラスターの論理スロットを返さないようにしました。 (AlexanderKukushkin)

Patroni はスタンバイクラスターに論理レプリケーションスロットを作成できないため、グローバル構成で定義されている場合は無視する必要があります。

- `patronictl --help` 出力で `docstring` を表示しないようにしました。 (IsraelBarthRubio)

`click` モジュールは、そのための特別なヒントを取得する必要があります。

- `kubernetes.standby_leader_label_value` のバグを修正しました (AlexanderKukushkin)

この機能は事実上、まったく機能しませんでした。

- クラスターシステム識別子が `patronictl list` 出力に返されました (PolinaBungina)

この問題は、Citrus のサポートを実装するときに発生しました。識別子はコーディネーターとすべてのワーカーで異なるため、識別子を隠す必要があります。

- Kubernetes 実装で `write_leader_optime` メソッドをオーバーライドしました (AlexanderKukushkin)

このメソッドは、新しいプライマリとなるために使用できる正常なレプリカがない場合に、シャットダウン LSN をリーダーエンドポイント/ConfigMap に書き込むことになっています。

- 停止した postgres を一時停止中に開始しないでください。(AlexanderKukushkin)

競合状態のため、Patroni は、一部の回復パラメーター (`primary_conninfo` など) が変更されたため、スタンバイを再起動する必要があると誤って想定していました。

- `patronictl query` コマンドのバグを修正しました (IsraelBarthRubio)

`-m` 引数のみが指定された場合、または `-r` または `-m` のいずれも指定されなかった場合は機能しませんでした。

- postgres を起動するためにコマンドラインで使用する整数パラメータを適切に処理するようになりました。(PolinaBungina)

値が文字列として提供され、整数にキャストされていない場合、Citus クラスターの `max_connections` に基づいた `max_prepared_transactions` の計算が正しくなくなりました。

- `pg_rewrite` を決定するときに `pg_stat_wal_receiver` に依存しないでください。(AlexanderKukushkin)

`pg_stat_wal_recevier` によって報告される `received_tli` が実際に再生されるタイムラインよりも進んでいる一方で、レプリケーション接続を介して `DENTIFY_SYSTEM` によって報告されるタイムラインは常に正しい場合があります。

18.6 バージョン 3.1.0

Breaking changes

- `restapi.keyfile` と `restapi.certfile` のセマンティクスを変更しました (AlexanderKukushkin)

以前、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)、`restapi.verify_client` は `ctl.certfile`、`ctl.keyfile`、`ctl.keyfile_password` にも `restapi.verify_client` を指定します。指定しない場合、Patroni は正しく動作しません。

New features

- ポッドの役割ラベルを構成可能にする (Waynerv)

値は、`kubernetes.leader_label_value`、`kubernetes.follower_label_value`、および `kubernetes.standby_leader_label_value` パラメータを使用して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。機能と移行手順の詳細については、`kubernetes.leader_label_value` を参照してください。

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 の実行中に回復中にクラッシュした場合、回復中でない状態で Postgres を起動します。(AlexanderKukushkin)

これにより、回復時間が短縮され、不必要なタイムラインの増加を防ぐことができます。

- /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、プライマリ上の LSN が前進していない場合でも、Patroni はハートビートループごとに /status を更新していました。

- 行き詰まった予備選が党首争いに勝つことを許さない (アレクサンダー・ククシュキン)

リソース不足により Patroni がかなりの時間ハングしていた場合、リーダーロックを取得する前に他のノードが Postgres を昇格していないかどうかさらにチェックされます。

- 特定の PostgreSQL パラメータ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 は適切なデフォルト値を使用していました。これに加えて、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した (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。この動作により、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するバックアップツールが機能しなくなります。現在、Patroni は PGDATA の権限を尊重し、PGDATA 内に作成されるすべてのディレクトリとファイルに権限を正しく設定します。

Bugfixes

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

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_rewind` の前に、一部の WAL セグメントをアーカイブする場合があります。archive_command に `&&` などのシェル演算子が含まれている場合、Patroni では機能しませんでした。

- 「スイッチオーバー時」の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て ☐ ☐ いない可能性があります。他のいくつかのノードが正常であったためにリーダーキーが削除されました。

- 「プライマリである」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` を修正しました (AlexanderKukushkin)

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

- 一時停止後の `pg_rewind` 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_rewind` を使用して偽のプライマリをクラスターに戻すことができませんでした。

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

リビジョンの不一致により、`create_revision/mod_revision` フィールドを使用してキー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。

- スタンバイクラスターの一時停止中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に従わず、`primary_conninfo` をそのまま保持します。

18.7 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表示できるようにしました (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レプリケーション状態を `streaming` として報告し、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 でのエラーメッセージの改善 (AlexanderKukushkin)

Etcdv3 クラスターにアクセスできない場合、Patroni は `/v2` エンドポイントにアクセスできないと報告していました。

- 可能であれば、`patronictl` のクォーラム読み取りを使用してください。 (AlexanderKukushkin)

Etcd または Consul クラスターは読み取り専用に低下する可能性があります、`patronictl` ビューからはすべて問題ありませんでした。今はエラーが出て失敗します。

- スプリットブレインの構成内での名前の重複を防止しました (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。人的ミスから守るのに役立ちます。

- Patroni の実行中にクラッシュした場合に Postgres を起動すると回復しない (AlexanderKukushkin)

これにより、回復時間が短縮され、不必要なタイムラインの増加を防ぐことができます。

Bugfixes

- RESTAPISSL 証明書が SIGHUP の受信時に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` などのパラメータの整数 GUC 検証を修正しました (FeikeSteenbergen)

パトローニは引用符で囲まれた数値を好みませんでした。回帰は 3.0.3 で導入されました。

- `synchronous_mode` の問題を修正しました。 (AlexanderKukushkin)

`synchronous_mode_strict` が有効なときに、欠落している同期スタンバイを誤って待機しないように、`txid_current()` を `synchronous_commit=off` とともに実行します。

18.8 バージョン 3.0.3

New features

- PostgreSQL16beta1 との互換性 (AlexanderKukushkin)

GUC のバリデータールールを拡張しました。

- PostgreSQLGUC のバリデーターを拡張可能にする (IsraelBarthRubio)

バリデーター・ルールは、`patroni/postgresql/available_parameters/` ディレクトリーにある 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、次々に適用されます。これにより、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使用できるようになります。

- `restapi.request_queue_size` オプションを追加しました (AndreyZidenkov、AlekseiSukhov)

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、それ以降のリクエストでは「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` 経由で呼び出されていましたが、そのためには `initdb` に渡されるパラメータの特別な引用符が必要でした。

- 停止フックの前に追加 (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、Postgres バイナリがコミュニティ Postgres ディストリビューションで使用されるものとは異なる名前で作成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。これは新しいクラスターにのみ必要ですが、構成に欠落していると `patroni --validate-config` がエラーを出しました。
- `postgresql.bin_dir` が空の場合、または設定されていない場合でも、エラーが発生しません。代わりに、まずデフォルトの PATH で Postgres バイナリを検索してみてください。
- `postgresql.authentication.rewind` セクションをオプションにします。これが見つからない場合、Patroni はスーパーユーザーを使用しています。

- `patronictl` のエラーレポートを改善しました (IsraelBarthRubio)

`/n` シンボルは、実際の改行シンボルではなく、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すると、指定された Citus グループが無期限にブロックされたままになります。

- `patronictl`の`--dcs-url`オプションで`etcd3`URL を許可します。 (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、例外が発生します。

18.9 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` のみをレポートしていました。

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

これにより、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できるようになります。

- K8s を再試行可能な HTTP ステータスコードを設定可能にしました (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` を取得する可能性があります。数回再試行すると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にのみ、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することを許可し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` がそれらを妨げるべきではありません。

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

`IMDSv2` は動作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 上で実行されている 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不明な場合は `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ない可能性があります)、Patroni は PostgreSQL のバージョンについて誤った仮定を立て、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()` 呼び出しではメタデータの同期は発生せず、変更はワーカーノードでは認識されませんでした。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 はそれに従おうとします。ある時点ですべてのノードが到達不能になった場合、Patroni は再接続を試行するときに、構成のノードと最後に知られているトポロジの組み合わせを使用します。

18.10 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。 (アレクサンダー・ククシュキン、ポリーナ・ブングナ)

Patroni は、昇格時に `promoted` ロールを誤って `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。この回帰は 3.0.0 で導入されました。

18.11 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、プライマリを降格することなく一時的な DCS 停止に耐え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 より古い Python バージョンのサポートが終了します。
- RAFT サポートは非推奨になりました。メンテナンスには最善を尽くしますが、起こり得る問題については保証も責任も負いません。

- このバージョンは、「マスター」を削除し、「プライマリ」を優先する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 DCS の一時的な停止に耐えることができます。詳細については、[documentation](#) を参照してください。

- Citus 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 (MichaelBank)

Patroni は引き続きこれらのログを書き込みますが、それは DEBUG でのみです。

- HA ループごとに監視クエリを 1 つだけ実行します (AlexanderKukushkin)

同期レプリケーションが有効な場合はそうではありませんでした。

- 最新の失敗したデータディレクトリのみを保持する (WilliamAlbertusDembo)

ブートストラップが失敗した場合、Patroni は \$PGDATA フォルダーの名前をタイムスタンプサフィックスを付けて変更していました。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は、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いた場合にのみ、DCS で同期として設定されます。

Removed functionality

- `patronictl scaffold` を削除 (AlexanderKukushkin)

これを導入した唯一の理由は、スタンバイクラスターを実行するハッキングな方法でした。

18.12 バージョン 2.1.7

Bugfixes

- 従来の Python モジュールとのわずかな非互換性を修正しました (AlexanderKukushkin)
これらにより、DebianBuster/Ubuntubionic 上で Patroni をビルド/実行できなくなりました。

18.13 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ているため、Patroni が SSL 接続を適切にシャットダウンする時間がありません。
- arm64 用のサンプル Dockerfile を調整 (PolinaBungina)
明示的な amd64 および x86_64 を削除します。 `libnss_files.so.*` は削除しないでください。

Security improvements

- 非レプリケーション接続に対して `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や演算子を使用して実行される可能性のある攻撃から保護したいと考えています。そのため、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) に対して `search_path=pg_catalog` が適用されます。
- パスワードが `pg_stat_statements` に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)
これは事実上必須ではないオプションであるためです。
- 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ている場合、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 Watchdog 構成を取得します (MattBaker)
Patroni は、新しいクラスターをブートストラップするときに、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、最初にデフォルトで Watchdog を構成していました。
- WIN32 で実行可能ファイルを検索する際のファイル拡張子の処理方法を修正しました。 (MartínMarqués)

ファイル名に拡張子がまだない場合のみ、ファイル名に `.exe` を追加します。

- ConsulTTL セットアップを修正しました (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 で乗算するのを忘れていました。その結果、ConsulTTL が 2 倍低下しました。

Removed functionality

- `patronictl configure` を削除 (PolinaBungina)

`patronictl` 構成を個別に作成する必要はなくなりました。

18.14 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、Etcdev3 のサポートが実稼働対応として宣言されています。Raft のパトローニはベータ版のままです。

New features

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

構成が無効な場合はコード 1 で終了し、エラーを標準エラー出力に出力します。

- 一時停止中にレプリケーションスロットを削除しないでください。 (AlexanderKukushkin)

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

- エンドポイントを監視するための `HEAD` リクエストメソッドをサポートしました。 (RobertCutajar)

GET の代わりに使用すると、Patroni は HTTP ステータスコードのみを返します。

- Windows での動作テストのサポート (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 上で Patroni の正常なシャットダウン (SIGTERM) をエミュレートします。

- `postgresql.proxy_address` の紹介 (AlexanderKukushkin)

これは DCS のメンバーキーに `proxy_url` として書き込まれ、サービスの検出に使用または役立つ可能性があります。

Stability improvements

- スレッドから `pg_replication_slot_advance()` を呼び出します (AlexanderKukushkin)

多くの論理レプリケーションスロットを持つビジーなクラスターでは、`pg_replication_slot_advance()` 呼び出しがメイン HA ループに影響を及ぼし、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 をアーカイブします (PolinaBungina)

プライマリがクラッシュして長時間ダウンした場合、一部の WAL ファイルがアーカイブと新しいプライマリから失われ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で、この問題が軽減されるだけでなく、一般的に継続的なアーカイブエクスペリエンスが向上します。

- Kubernetes サービスを作成しようとするときに 403 エラーを無視する (NickHudson、PolinaBungina)

Patroni はサービスを作成しようとして失敗し、実際にはすでに存在している可能性があるサービスを作成しようとしてログをスパム送信していました。

- liveness プローブの改善 (AlexanderKukushkin)

ハートビートループがプライマリの「ttl」またはレプリカの「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 上の *watchdog* の代替として使用できるようになります。

- 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ほとんどありませんでした。

- ブートストラップの実行中にクローンを作成しないようにします (AntsAasma)

クラスターのブートストラップの実行中に、リーダーのトリガ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 との互換性 (AlexanderKukushkin)

Python のバージョンによっては、閉じられたソケットで `select()` が呼び出された場合、`SequentialThreadingHandler.select()` メソッドによって `TypeError` 例外と `IOError` 例外が発生する可能性があります。

- ソケットのシャットダウン前に SSL 接続を明示的にシャットダウンしました。 (AlexanderKukushkin)

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

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

内部 API の変更により、クラスター名のヘッダーが間違った行に表示されました。

Bugfixes

- Etcd リースグラントの期限切れトークンの処理 (monsterxx03)

エラーが発生した場合は、新しいトークンを取得してリクエストを再試行します。

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

これは以前のリリースで導入されましたが、実際には機能しませんでした。

- データディレクトリストレージが消失した場合の処理 (AlexanderKukushkin)

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

- ユーザーのバックエンドが閉じるのを待機するときに `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ているバックグラウンドワーカー (Citux メンテナンスデーモンなど) である可能性があります。

- *:<port> を postgresql.listen として受け入れます (DenisLaxalde)

patroni --validate-config は無効であると訴えていました。

- Raft のタイムアウトを修正 (AlexanderKukushkin)

Patroni または patronictl が開始されると、既知のメンバーから Raft クラスタートポロジ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に Consul サービスを強制的に更新します (JohnA.Lotoski)

そうしないと、「呼び出し実行中の rpc エラー: 呼び出し実行中の rpc エラー:ACL が見つかりません」というエラーが発生します。

18.15 バージョン 2.1.4

New features

- 一般的な 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 の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 ではない場合に便利です。

- Watchdog に ppc64le 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 Watchdog のサポートが修正されました。

- 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 年以降廃止され、Python3.9 で失敗します。

- K8s 上のサービスアカウントトークンを定期的に更新します (HaitaoLi)

Kubernetes v1.21 以降、サービスアカウントトークンは 1 時間で期限切れになります。

- /read-only-sync モニタリングエンドポイントを追加しました (Dennis4b)

これは /read-only に似ていますが、同期レプリカのみが含まれます。

Stability improvements

- プライマリとの論理デコード設定に構成の不一致がある場合は、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なくなります。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それを使用して開始するまで実行されず、不必要な再起動が繰り返されていました。

- PostgreSQL12+のリカバリ設定パラメータの特別な処理 (AlexanderKukushkin)

レプリカとして起動している間、Patroni は `pg_settings` からクエリするのではなく現在のパラメータ値をキャッシュすることで、リーダーアドレスが変更された場合に `postgresql.conf` を更新して再起動/再ロードできる必要があります。

- `postgresql.listen` パラメータでの IPv6 アドレスの処理が改善されました (AlexanderKukushkin)

`listen` パラメータにはポートがあるため、IPv6 アドレスを角括弧内に入力しようとしませんが、リストに複数の IP がある場合、これは正しく削除されませんでした。

- PostgreSQL10 以前でのみ相違チェックを実行する場合は `replication` 認証情報を使用するようにしました。 (AlexanderKukushkin)

`rewind` が有効な場合、Patroni は新しい Postgres バージョンで `superuser` または `rewind` 資格情報を再度使用します。

Bugfixes

- `dateutil.parser` のインポートが欠落していたのを修正しました (WesleyMendes)

テストが失敗したのは、他のモジュールからもインポートされていたためだけではありません。

- `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)

場合によっては、Patroni がそれを数値として渡そうとしていました。

- 失敗した `pg_rewind` 試行の処理を改善しました (AlexanderKukushkin)

`pg_rewind` 中にプライマリが使用できなくなると、`$PGDATA` は壊れた状態のままになります。その後、構成で許可されていない場合でも、Patroni は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場合は、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、アノテーションは現在の値を保持します。

- K8sAPI ウォッチャーによる同時実行の問題の処理 (AlexanderKukushkin)

特定の (未知の) 条件下では、ウォッチャーが期限切れになる可能性があります。その結果、`attempt_to_acquire_leader()` メソッドが HTTP ステータスコード 409 により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。

18.16 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)

これは、`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クスを追加しました。 (AlexandrePereira)

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

- 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。(MichaelBank)

スタンバイクラスターが Patroni クラスターからレプリケートされている場合は、PostgreSQLv10 以降の `libpq` で利用できるクライアント側のフェイルオーバーを利用するとよいでしょう。つまり、スタンバイリーダー上の `primary_conninfo` と、接続文字列内の `pg_rewind` 設定 `target_session_attrs=read-write` です。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 `psycopg2` 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 `psycopg2` には、`UndefinedFile` 例外がまだありません。

- すべての Etcd ノードが応答しない場合は、`etcd3` ウォッチャーを再起動します。(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るノードによってのみ維持されていました。

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みなしていました。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際、リーダーが存在しない可能性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は、`validator.py` で最小値と最大値が引用符で囲まれていたため、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 (AlexanderKukushkin)

`remove_data_directory_on_diverged_timelines` は設定されているものの、`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が許可されていない可能性があります。

- RESTAPI 証明書の置換時の「ポートが使用中」エラーを修正しました (AntsAasma)

証明書を切り替えるときに、同時 API リクエストで競合状態が発生しました。交換期間中にアクティブなサーバーがある場合、交換はポート使用中エラーでエラーとなり、Patroni はアクティブな API サーバーがない状態でスタックします。

- パスワードに % 文字が含まれてい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。(BastienWirtz)

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符で囲んで 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を好みませんでした。

- 「属性エラー: 属性'リーダー'がありません」例外を修正しました (HrvojeMilković)

これは、同期モードが有効になっており、DCS コンテンツが消去された場合に発生する可能性があります。

- 発散タイムラインチェックのバグを修正しました。(AlexanderKukushkin)

パトローニはタイムラインが分岐していると誤って仮定していた。`pg_rewind` の場合は問題は発生しませんでした。が、`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は、前のリーダーが再初期化されることになります。

18.17 バージョン 2.1.2

New features

- `psycpg>=3.0` との互換性 (AlexanderKukushkin)

デフォルトでは、`psycpg2` が優先されます。`psycpg>=3.0` は、``psycpg2`` が利用できない場合、またはバージョンが古すぎる場合にのみ使用されます。

- `dcs_last_seen` フィールドを RESTAPI に追加しました (MichaelBank)

このフィールドには、クラスタメンバーが最後に DCS と正常に通信した時刻 (UNIX エポックとして) が記録されます。これは、ネットワークパーティションの特定や分析に役立ちます。

- `pg_controldata` が「シャットダウン」を報告したら、リーダーのロックを解除します。(AlexanderKukushkin)

`archive_command` が遅い/障害が発生している場合の遅いスイッチオーバー/シャットダウンの問題を解決するために、Patroni は、`pg_controldata` が PGDATA を `shut down` として正確に報告し始め、すべての変更を受信し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。

- `sslcrlidir` 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は PostgreSQLv14 で導入されました。

- Zookeeper で ZNode の ACL 設定を許可しました (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 を適用できるようにします。

Stability improvements

- 次の回復の試みを次の HA ループまで遅らせます (AlexanderKukushkin)

たとえば、ディスク容量不足が原因で Postgres がクラッシュし、そのために起動に失敗した場合、Patroni はそれを回復しようとするあまりにも熱心にログを溢れさせます。

- 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 (MichaelBank)

正するために、`pg_stat_replication_slots` ビューをクエリするときにすべての一時スロットをスキップします。

- `pg_replication_slot_advance()` がタイムアウトしないようにしました (AlexanderKukushkin)

この場合、Patroni はデフォルトの `statement_timeout` を使用していましたが、一度呼び出しが失敗すると回復できない可能性が非常に高く、結果として `pg_wal` と `pg_catalog` のサイズが増大してしまいます。

- `/status` は降格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、新しい `/status` キーが導入されましたが、`optime` は引き続き `/optime/leader` に書き込まれました。

- 降格時の DCS 例外の処理 (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。

- 一部のケースでは `use_unix_socket_repl` が機能しませんでした (AlexanderKukushkin)

具体的には、`postgresql.unix_socket_directories` が設定されていない場合です。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しました (AlexanderKukushkin)

`clusters_unlocked` を定義できない場合があり、その結果、GET `/metrics` エンドポイントで例外が発生しました。さらに、エラー処理方法では、`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 の場合はさらに多くの要素がある可能性があります。

- 新しく昇格したノードの回復が完了するまで待つから、巻き戻しを決定します。 (AlexanderKukushkin)

実際のプロモーションが行われて新しいタイムラインが作成されるまでには、しばらく時間がかかる場合があります。待機せずに、レプリカは巻き戻しが必要では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際に、履歴ファイル内の欠落しているタイムラインを処理します。 (AlexanderKukushkin)

現在のレプリカ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が必要ないと誤って想定しています。

18.18 バージョン 2.1.1

New features

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

Etcd を使用すると、同じドメイン内の複数の Etcd クラスターを区別できます。今後、Patroni もこれをサポートします。

- 新しいリーダーとともに歴史を豊かにする (huiyalin525)

新しい列が `patronictl history` 出力に追加されます。

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

以前のリリースでは、ディスク上で変更された場合に RESTAPI 証明書を再ロードする機能が追加されました。残念ながら開始直後に無条件でリロードが発生してしまいました。

- `etcd.use_proxies` が設定されている場合、クラスターメンバーを解決しません。 (AlexanderKukushkin)

Patroni は起動時にメンバーのリストをクエリして Etcd クラスターの健全性をチェックします。それに加えて、ホスト名を解決しようとしたましたが、これはプロキシ経由で Etcd を操作する場合には必要なく、不要な警告を引き起こしていました。

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

`state = 'streaming'` の場合でも、`pg_stat_replication` ビューの `replay_lsn`、`flush_lsn`、または `write_lsn` フィールドに NULL 値が含まれる可能性があるようです。

18.19 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、フェイルオーバー/スイッチオーバーを生き延びるための論理レプリケーションスロットが作成され、RESTAPI のホワイトリストのサポートが実装され、ログの数もハートビートごとに 1 行に削減されます。

New features

- PostgreSQLv14 との互換性 (AlexanderKukushkin)

Patroni 自体が「一時停止」モードになっていない場合は、WAL 再生の一時停止を解除します。プライマリ上の `max_connections` などの特定のパラメータの変更により、「一時停止」される可能性があります。

- フェイルオーバー論理スロット (AlexanderKukushkin)

PostgreSQLv11 以降で論理レプリケーションスロットがフェイルオーバー/スイッチオーバーを存続できるようにします。レプリケーションスロットは、再起動によってプライマリからレプリカにコピーされ、その後 `pg_replication_slot_advance()` 関数を使用して前に移動されます。その結果、スロットはフェイルオーバー前にすでに存在しており、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 のホワイト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許可されます。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もできます。

- UNIX ソケット経由の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 で表示されるカスタムタグを任意の数だけ指定できます。今後、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, MichaelBank)

`/patroni` と同じメトリクスを公開するエンドポイント。

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

すべてが正常に完了すると、HA ループの実行ごとに 1 行だけが書き込まれます。

Breaking changes

- 古い `permanent logical replication slots` 機能は PostgreSQLv10 以前では動作しなくなります。 (AlexanderKukushkin)

プロモーションを実行した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、無効になることがないことを保証できません。

- ノードがロックを保持している場合、`/leader` エンドポイントは常に 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。これを解決するには、`/leader` ヘルスチェック エンドポイントの動作を変更します。クラスターが正常であるか `standby_cluster` であるかは考慮されず、200 が返されます。

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、暗号化サポートは非 $\square$ 常に不安定でした

- Raft 実装における DNS の問題の処理 (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 はオブジェクトの作成時に 1 回だけ効果的に解決を実行します。同じノードが別の IP でオンラインに戻るときに問題が発生していました。

Stability improvements

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

`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るようになりました。

- ディスク上の RESTAPI 証明書が変更された場合はリロードします (MichaelTodorovic)

RESTAPI 証明書ファイルが適切に更新された場合、Patroni はリロードを実行しませんでした。

- Kerberos 認証が使用されている場合は pgpass ディレクトリを作成しないでください。 (Kostiantyn-Nemchenko)

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に関する小さな問題を修正しました (AlexanderKukushkin)

PITR を実行するときのみ `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+ との互換性 (AlexanderKukushkin)

Patroni は再試行を独自に処理するため、使用可能な接続がない場合は Zookeeper クラスターへのリクエストが直ちに破棄されるという kazoo の古い動作に依存しています。

- プロキシ経由で接続していることがわかっている場合に、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 ゲートウェイ経由で Etcdv3 クラスターと連携しており、クラスターの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。この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、プロキシ経由で接続する場合は後者が実行されなかったためです。

18.20 バージョン 2.0.2

New features

- 外部で管理されているレプリケーションスロットを無視する機能 (JamesColeman)

Patroni は、未知の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。今後は、削除してはならないスロットを設定できるようになります。

- 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)

これは、`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)

これは、`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 認証資格情報の定時比較 (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する (KrishnaSarabu)

同期ノード上のレプリケーションラグが設定されたしきい値を超え始めた場合、そのノードは非同期に降格されるか、他のノードに置き換えられるか、あるいはその両方にな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実行するときに `postgres` を `hot_standby = off` で開始します (IgorYanchenko)

カスタムブートストラップ中、Patroni はベースバックアップを復元し、Postgres を起動し、回復が完了するまで待機します。スタンバイ上の一部の PostgreSQL パラメーターはプライマリよりも小さくすることができず、新しい値 (WAL から復元された) が構成された値より大きい場合、Postgres はパニックを起こして停止します。このような動作を回避するために、`hot_standby` モードを使用せずにカスタムブートストラップを実行します。

- 必要な Watchdog が正常でない場合にユーザーに警告します。 (NicolasTuvvin)

Watchdog デバイス書き込み可能でない場合、または必須モードで見つからない場合、メンバーを昇格することはできません。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の向上 (AlexanderKukushkin)

PostgreSQL が明確にシャットダウンされていないことに Patroni が気付いた場合、場合によっては、Postgres をシングルユーザーモードで起動することによってクラッシュ回復が実行されます。(ディスク上のスペース不足などにより) リカバリが失敗しても、エラーが飲み込まれる可能性があります。

- `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` は数年前からメンテナンスされていないため、誰かが新機能とバグ修正を備えたフォークを作成しました。

- `patronictl` を実行する場合は `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、`kubernetes` エンドポイントをクエリ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てください。 (MarkMercado)

使いやすさと使いやすさを向上させます。

Bugfixes

- 一時停止中のリーダーロックを失った場合、再起動や昇格を中断しないでください (AlexanderKukushkin)

一時停止中は、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()` に関する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` 内のいくつかのメソッドをオーバーライドします。 `shutdown_request()` メソッドが忘れられていました。

- Zookeeper 使用時のスリープ時間の問題を修正しました (AlexanderKukushkin)

パトローニが HA コードを実行する間に最大 2 倍長く眠っ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保ちます。 `src` 引数と `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は失敗していました。

- `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設定されていない場合、Patroni は `post_init` スクリプトの呼び出しに失敗し、その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` の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` が失敗していました。

- 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 クラスタポートの検出を実行し、必要に応じて認証を行います。etcd サーバーの 1 つがアクセスできず、Patroni がこのサーバーで認証を実行しようとして、次のノードで再試行せずに失敗することが発生する可能性があります。

- 空のリストを返す `psutilcmdline()` でケースを処理します (AlexanderKukushkin)

ゾンビプロセスは依然としてポストマスターの子ですが、`cmdline()` を持っていません。

- `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、環境経由で `kubernetes.use_endpoints` を無効にすることができなくなりました。

- 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。

18.21 バージョン 2.0.1

New features

- `less` が利用できない場合は、`patronictl edit-config` のポケットベルとして `more` を使用します。 (PavelGolub)

Windows では、`more.com` になります。それに加えて、`requirements.txt` では `cdiff` が `ydiff` に変更されましたが、`patronictl` は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便利です。 `raft.password` はトラフィック暗号化を有効にします (`cryptography` モジュールが必要です)。

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

接続パラメータは PostgreSQL13 で導入されました。

Stability improvements

- 一時停止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か空の場合、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` の不一致が発生しても終了せず、警告をログに記録するだけです。
 3. Postgres がリカバリ中ではなく実行されている (書き込みを受け入れている) 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の実行時に `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することでクラッシュ回復を行います。クラッシュリカバリ中に、リーダーのロックが更新されます。クラッシュリカバリが `master_start_timeout` 秒以内に終了しなかった場合、Patroni はクラッシュリカバリを強制的に停止し、リーダーのロックを解除します。

- `urllib3` の要件から `secure` の追加要素を削除しました。 (AlexanderKukushkin)

これを追加した唯一の理由は、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場合、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)

Patroni 構成で `patronictl` を使用する場合は、`self_addr` を `partner_addrs` に追加する必要があります。

- `get_guc_value()` のバグを修正しました (AlexanderKukushkin)

Patroni は PostgreSQL12 で `restore_command` の値を取得できなかったため、`pg_rewind` の欠落している WAL を取得できませんでした。

18.22 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 および Patroni のサポートが追加され、オプションで次の呼び出しが可能になります。 `pre_promote` (フェンシング) スクリプト。

PostgreSQL 13 support

- PostgreSQL13 以降で `standby_leader` に昇格するときに `on_reload` を起動しないようにしました。(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` は実質的に相互に重複するため、Patroni は `on_role_change` のみを呼び出します。

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

PostgreSQL12 では `gssencmode` および 13 `channel_binding` 接続パラメータが導入され、`postgresql.authentication` セクションで定義されている場合はそれらを使用できるようになりました。

- `wal_keep_segments` から `wal_keep_size` への名前変更ハンドル (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 では `pg_rewind` を `--restore-target-wal` とともに使用してください。(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_rewind` にそれを使用するように指示します。

New features

- [ベータ] 純粋な RAFT 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 を実行できるようになります。HA の場合は、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。

- [ベータ] gPRC ゲートウェイ経由で 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。これを使用するには、Patroni 構成ファイルである `etcd3` セクションを明示的に作成する必要があります。

- 複数の同期スタンバイのサポート (KrishnaSarabu)

これにより、複数の同期レプリカを含む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 `synchronous_node_count` によって制御されます。デフォルトでは 1 に設定されており、`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、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すると、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 ファイルがアルファベット順にロードされ、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らメインスレッドを起動するようにしました。(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。通常、キーは HA ループごとに 1 回だけ更新されます。メインスレッドをウェイクアップしないと、レプリカは必要以上に最大 `loop_wait` 秒ほど長く待機する必要があります。

- 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 設定ファイルでの 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まれることになっていますが、Patroni はプレーンなアドレスを期待していました。現在は両方の形式がサポートされています。

- Consul `service_tags` 設定パラメータを追加しました (RobertEdström)

これらは、ロードバランサーなどによる動的な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)

これにより、`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` の失敗後に WAL とテーブルスペースを移動する (FeikeSteenbergen)

`reinit` を実行すると、Patroni は 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すでに削除していました。`move_data_directory()` メソッドは同様のジョブを実行します。つまり、WAL ディレクトリとテーブルスペースの名前を変更し、PGDATA のシンボリックリンクを更新します。

Improved in `pg_rewind` support

- タイムラインの発散チェックの改善 (AlexanderKukushkin)

レプリカ上の再生位置がスイッチポイントより前でない場合、または元のプライマリ上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は、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終わりを取得するために、`pg_waldump` を使用し、その出力を解析します。

- `pg_rewind` がそれについて文句を言ったら、不足している WAL を取得してみます。(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` が分岐点より前のチェックポイントの場所を見つけることができない場合があります。PostgreSQL13 以降、`pg_rewind` は欠落している WAL を取得するために `restore_command` を使用できるようになり

ました。古い PostgreSQL バージョンの場合、Patroni は失敗した巻き戻し試行のエラーを解析し、独自に `restore_command` を呼び出して不足している WAL を取得しようとします。

- 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切り離されているため、リーダーの選出やタイムラインの切り替えについてはすぐには認識されません。この事実を検出するために、`standby_leader` は `pg_wal` 内の新しい履歴ファイルを定期的にチェックします。

- 履歴ログ出力を短縮して美しくしました (AlexanderKukushkin)

Patroni が `pg_rewind` の必要性を判断しようとする、プライマリからの履歴ファイルの内容がログに書き込まれる可能性があります。履歴ファイルはフェイルオーバー/スイッチオーバーのたびに増大し、最終的には多くの行を占めるようになり、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 は現在のレプリカタイムラインの前の 3 行と後の 2 行だけを表示します。

Improvements on K8s

- `kubernetes` Python モジュールを削除しました (AlexanderKukushkin)

公式の `PythonKubernetes` クライアントには自動生成されたコードが多数含まれているため、非常に重くなっています。Patroni は K8sAPI エンドポイントのほんの一部のみを使用しており、それらのサポートの実装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した。 (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にユーザー空間プログラムまたは `iptables` に依存します。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ことで、より適切な再試行戦略を実装し、K8s マスターノードがアップグレードされたときに 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` を設定しました。 (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()` で起こり得る競合状態を修正しました。 (AlexanderKukushkin)

Patroni の外部でリーダー構成マップまたはエンドポイントの同時更新が行われると、`update_leader()` 呼び出しが失敗する可能性があります。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 では、`cluster.config` が `None` であるため PATCH 呼び出しは例外で失敗しますが、Kubernetes では構成アノテーションが問題なく作成され、ブートストラップ終了後にブートストラップ構成を書き込むことができませんでした。

- `pause` のバグを修正しました。 (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、何も行わないはずでした。

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す。(AlexanderKukushkin, Ben-Harris)

TLS ハンドシェイクが API スレッドで実行され、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 の危険があります)。

- RESTAPI でクライアント証明書から独立して `basic-auth` を確認しました。(AlexanderKukushkin)

以前は、クライアント証明書のみが検証されていました。2つのチェックを独立して実行することは、完全に有効な使用例です。

- OPTIONS リクエストの HTTP ヘッダーの後に CRLF を二重に書き込みます (SergeyBurladyan)

HAProxy は 1 つの CRLF で問題ありませんでしたが、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` が動物園飼育員 (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 自体では問題は発生しませんでしたでしたが、外部に公開される場合は、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、誤って 200 と応答していました。

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

RESTAPI 呼び出しにより、スケジュールされたスイッチオーバーが削除されます。

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

これらは、K8s サービスがラベルセクターとともに使用されている場合に、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)

チェックではオプションのキーワード `?lag=<max-lag>` がサポートされるようになり、ラグが指定された値より小さい場合にのみ 200 で応答します。この機能を利用する場合は、リーダー上の WAL の位置に関する情報が `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、リクエストがブラウザから行われる場合に役立つ可能性があります。

Improvements in patronictl

- `patronictl pause` に存在しないリーダーに電話をかけないでください。(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「なし」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の処理 (AlexanderKukushkin)

K8 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含まれていない可能性があります。 `patronictl` が失敗していました。

- ASCII クラスタートポロジー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把握するのに非常に役立ちます。

- `patronictl flush switchover` の実装 (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/更新しようとしたときに、Patroni 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 の例外処理を改善しました (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`) を作成するときにすでにそれを行っていましたが、`post_init` の場合にそれが欠けていたのは見落としでした。その結果、スクリプトが内部で独自に実行していない場合、`synchronous_mode` のブートストラップを完了できませんでした。

- Consul プールマネージャーの `maxsize` を増加 (ponvenkates)

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 が Postgres が実行中であると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` ファイルが正しく生成されません。

- 特殊文字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 (KrishnaSarabu)

Patroni は、名前に「-」などの特殊文字が含まれている場合 (例: `/"abc-us-1/"`) に、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` ディレクトリの外にあった場合、Patroni は失敗していました。

18.23 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、停止操作が `master_stop_timeout` で設定された値を超えて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。耐久性と可用性のトレードオフに応じて値を設定します。パラメーターが設定されていないか、正以外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間にプライマリが 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、ノード数が固定され、名前が決して変更されない、静的クラスターに対する優れたソリューションが得られました。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ないように、 `slots` 内のすべてのノードの名前をリストする必要があるだけです。

- ConfigValidator の初稿 (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設定できるようになりました (KrishnaSarabu)

Patroni は、フェイルオーバー/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、重要なのは最後の数行だけです。 `max_timelines_history` パラメータを使用すると、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 の互換性 (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

- メンバータグを表示 (コスティアンティン・ネムチェンコ、アレクサンダー・ククシキン)

タグはノードごとに個別に構成されており、タグの概要を簡単に把握する方法はありませんでした。

- メンバーの成果を向上させる (AlexanderKukushkin)

冗長なクラスター名は各行に表示されなくなり、テーブルの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
```

(次のページに続く)

(前のページからの続き)

```
|
|                               |               |               |               |               | 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ているが見つからない場合は失敗します。(KaarelMoppel)

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

- K8s ポッドが初期化されていないと patronictl が壊れ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 つが停止または起動しているときは、有効なアノテーションがまだなく、`patronictl` が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に 1 秒のバックオフを適用しました。(AlexanderKukushkin)

これは主にログの氾濫を避けるために必要ですが、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 から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ると、再試行を要求されることがあります。

- KUBERNETES_環境をポストマスターからスクラブしました (FeikeSteenbergen)

KUBERNETES_ 環境変数は PostgreSQL には必要ありませんが、環境変数をポストマスターに公開すると、バックエンドや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ていました。これにより、Patroni が再初期化でループする原因となります。この問題に対する以前の回避策は、PGDATA スクリプトを実装することでした。

- プロモートが行われた後に CHECKPOINT を明示的に実行しました。(AlexanderKukushkin)

これは、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なリフレッシュ (AlexanderKukushkin)

Patroni が Etcd クラスターのすべてのメンバーに対する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、**A** または **SRV** レコードで IP/ホストの変更を再チェックします。

- pg_controldata からの欠損値をスキップします (FeikeSteenbergen)

PGD  ATA 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エラーを出して異常終了します。

Bugfixes

- 必要に応じて Consul の SSL 検証を無効にする (JulienRiou)

特定のバージョンの `urllib3` 以降では、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 ループのサイクルごとにレプリケーション接続を開かないようにします。(AlexanderKukushkin)
回帰は 1.6.4 で導入されました。

- 失敗したプライマリで `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 が開始された場合に巻き戻し状態をリセットしました。(AlexanderKukushkin)

このバグの結果、Patroni は一時停止モードで postgres を手動でシャットダウンして起動していました。

- `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再起動していました。

- 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 (パッケージ化) します。 `multiprocessing` の `fork` メソッドではなく `spawn` メソッドに切り替えたときに、互換性が失われていました。

18.24 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` が完了するまで待機します。

- 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 で書き直されます。
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 ドメインソケットを使用しないでください

Stability improvements

- `unix_socket_directories` と `stats_temp_directory` が存在することを確認してください (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、それらを作成してみます。作成に失敗すると Patroni は終了します。

- `postgresql.pgpass` が Patroni が書き込みアクセス権を持っている場所にあることを確認してください。 (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した (KostiantynNemchenko)

ネットワークに小さな問題が発生した場合でも、`serfHealth` に障害が発生すると、ノードに関連付けられた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 `t1` よりもはるかに早く失われ、レプリカの不要な再起動が発生し、場合によってはプライマリが降格されます。

- K8sAPI への接続用に `tcp` キープアライブを構成しました (AlexanderKukushkin)

TTL 秒後にソケットから何も取得されない場合、ソケットは停止していると見なすことができます。

- ユーザー作成時にパスワードを記録しないようにします。 (AlexanderKukushkin)

パスワードが拒否された場合、またはログ記録が冗長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。これを回避するために、Patroni はユーザーの作成/更新を試行する前に `log_statement`、`log_min_duration_statement`、および `log_min_error_statement` を安全な値に変更します。

Bugfixes

- カスケードレプリカで `standby_cluster` 設定の `restore_command` を使用します。 (AlexanderKukushkin)

`standby_leader` は、この機能が存在する最初からすでに実行していました。レプリカで同じことを行わないと、スタンバイリーダーに追いつけなくなる可能性があります。

- スタンバイクラスターによって報告される更新タイムライン (AlexanderKukushkin)

タイムライン切り替えの場合、スタンバイクラスターはプライマリから正しくレプリケートしていましたが、`patronictl` は古いタイムラインを報告していました。

- 特定の回復パラメータを `custom_conf` で定義できるようにしました。 (AlexanderKukushkin)

レプリカでリカバリパラメータの検証を行うとき、Patroni は `archive_cleanup_command`、`promote_trigger_file`、`recovery_end_command`、`recovery_min_apply_delay`、および `restore_command` が `patroni` 構成ではなく `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、それらをスキップします。

- 名前にピリオドが含まれる `postgresql` パラメータの処理を改善しました。 (AlexanderKukushkin)

このようなパラメータは、単位が必ずしも文字列であるとは限ら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 (例: `pg_stat_statements.max`)。

- シャットダウン時の例外処理を改善しました (AlexanderKukushkin)

シャットダウン中に、Patroni は DCS 内のステータスを更新しようとします。DCS にアクセスできない場合は、例外が発生する可能性があります。例外処理が不足しているため、ログスレッドが停止できませんでした。

18.25 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_ に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を `pg_basebackup` に適用し、カスタムレプリカ作成メソッドを適用しました。(AlexanderKukushkin)

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。

18.26 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出力して終了します。

- すべての http リクエストに `user-agent` http ヘッダーを設定しました (AlexanderKukushkin)

Patroni は、http プロトコルを介して Consul、Etcd、および KubernetesAPI と通信します。特別に作成された `user-agent` (例: Patroni/1.6.2 Python/3.6.8 Linux) があると、デバッグや監視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、トレースバックは `log.level=DEBUG` の場合にのみ表示されます。デフォルトの動作は変わりません。

Stability improvements

- 設定ファイルで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)

たとえば、必要な場合のみ 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。ZooKeeper。これにより、メモリ使用量が削減され、INFO メッセージ `Failed to import smth` の問題が解決されます。

- 明示的な要件から Python `requests` モジュールを削除しました (AlexanderKukushkin)

これは重要なことには使用されませんでした、`urllib3` の新しいバージョンがリリースされると多くの問題を引き起こしました。

- YAML 配列ではなくカンマ区切り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` (カンマの後のスペース文字) 形式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ーに強制しないでください。(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わかりやすくしました。(IgorYanchenko)

経験の浅いユーザーにとって、Python スタックトレースから何が問題なのかを特定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 以降でのみ `primary_conninfo` で `passfile` を使用する (AlexanderKukushkin)

最新バージョンの `libpq` がインストールされていない限り、古いバージョンでは `passfile` が動作する保証はありません。

18.27 バージョン 1.6.1

New features

- `PATRONICTL_CONFIG_FILE` 環境変数 (msvechla) を追加しました

これにより、環境から `patronictl` の `--config-file` 引数を構成できます。

- `patronictl history` の実装 (AlexanderKukushkin)

フェイルオーバー/スイッチオーバーの履歴が表示されます。

- `pg_rewind` を実行するときに `PGOPTIONS` に `-c statement_timeout=0` を渡します (AlexanderKukushkin)

これは、サーバー上の `statement_timeout` が小さな値に設定され、`pg_rewind` によって実行されたステートメントの 1 つがキャンセルされた場合から保護します。

- PostgreSQL 構成でより低い値を許可する (Soulou)

Patroni では、一部の PostgreSQL 構成パラメータを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ていませんでした。許容される最小値が小さくなり、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する (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、リwind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` を指定できるようになります。

- `primary_conninfo` のパスワードの代わりに `passfile` を使用します。(AlexanderKukushkin)

`postgresql.conf` での `600` 権限の設定を回避できます。

- 設定の変更に関係なく `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 経由で、または Patroni プロセスに `SIGHUP` を送信してリロードを実行している場合、通常、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にはこの問題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を比較します。(AlexanderKukushkin)

以前の `check_recovery_conf()` メソッドは、`primary_conninfo` が変更されたかどうかのみをチェックし、他のすべての回復パラメータを考慮していませんでした。

- 再起動せずにいくつかの回復パラメータを適用できるようにしました。(AlexanderKukushkin)

PostgreSQL12 以降、リカバリパラメータ `archive_cleanup_command`、`promote_trigger_file`、`recovery_end_command`、および `recovery_min_apply_delay` は再起動せずに変更できます。将来の Postgres リリースでは、このリストが拡張され、Patroni が自動的にサポートする予定です。

- `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 を再起動して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_` プレフィックスが付いた環境変数のみを削除する (CodyCoons)

これにより、さまざまな外部データラッパーの実行に関する多くの問題が解決されます。

Stability improvements

- K8sAPI を使用する場合は `LIST+WATCH` を使用してください (AlexanderKukushkin)

これにより、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るようになり、K8 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、PGDATA に `lost+found` と `.dotfiles` しかない場合、PGDATA は空であるとみなされる可能性があります。今、パトローニも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 はエラーを出して終了します。

- すべての HA ループで高価な `os.listdir()` を呼び出すのを避けてください。(AlexanderKukushkin)

システムが IO ストレスにさらされている場合、`os.listdir()` の実行に数秒 (または数分) かかる可能性があり、Patroni の HA ループに悪影響を及ぼします。これにより、更新がないためにリーダーキーが DCS から消えてしまう可能性もあります。PGD [X] ATA が空でないことを確認する、より適切で安価な方法があります。ここで、PGDATA に `global/pg_control` ファイルが存在するかどうかを確認します。

- ログインフラストラクチャのいくつかの改善 (AlexanderKukushkin)

以前は、ログスレッドが `daemon` スレッドであったため、シャットダウン時に最後の数行のログが失われる可能性があります。

- Python3.4 以降で `spawn` マルチプロセッシング開始メソッドを使用する (MaciejKowalczyk)

Python ではスレッド処理とマルチプロセッシングがうまく混合しないことが知られています。デフォルトのメソッド `fork` から `spawn` に切り替えることが推奨される回避策です。そうしないと、Postgres が実際には稼働しているにもかかわらず、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チェックします。 `optional` に設定すると、安全でないすべての RESTAPI エンドポイントについてクライアント証明書がチェックされます。

- Postgres が実行されていない場合は、`GET /replica` ヘルスチェックリクエストに対して応答コード 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までの回復にかなりの時間を費やす可能性があります。

- `/history` および `/cluster` エンドポイントを実装します (AlexanderKukushkin)

`/history` エンドポイントは、DCS の `history` キーの内容を示します。 `/cluster` エンドポイントには、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やスイッチオーバーなどのサービス情報が表示されます。

Improvements in Etcd support

- EtcdRAFT 内部エラーでの再試行 (AlexanderKukushkin)

Etcd ノードがシャットダウンされると、 `response code=300, data='etcdserver: server stopped'` が送信されます。これにより、Patroni はプライマリを降格させていました。

- Etcd リクエストの再試行を早すぎて諦めないでください。 (AlexanderKukushkin)

ネットワークに問題が発生すると、Patroni はすぐに Etcd ノードのリストを使い果たし、`retry_timeout` 全体を使用せずに諦めたため、プライマリが降格される可能性があります。

Bugfixes

- `pg_rewind` ユーザー (kremius) に実行権限を付与するときに `synchronous_commit` を無効にする

ブートストラップが `synchronous_mode_strict: true` で実行される場合、非同期ノードが使用可能なため、`'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、Python3.7 ではデフォルトで非デーモンリクエストごとにスレッドが生成されるようにしました。

- 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、`patronictl reinit --force` が上書きされる可能性があります。これにより、Patroni が Basebackup の実行中に Postgres を起動しようとしたという状況が発生しました。

- `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は、別のカーソルオブジェクトを作成する必要があります。

- 大文字を含む名前の同期スタンバイが昇格されない問題を修正しました。(AlexanderKukushkin)

Postgres が `application_name` と `synchronous_standby_names` の値を比較するときに同じことを行っていたため、名前を小文字に変換しました。

- 新しいプロセスを開始する前に、コールバックプロセスとともにすべての子を強制終了します。(AlexanderKukushkin)

そうしないと、`bash` でコールバックを実装することが難しくなり、最終的に 2 つのコールバックが同時に実行される状況が発生する可能性があります。

- 「起動に失敗しました」問題を修正しました (AlexanderKukushkin)

特定の条件下では、Postgres が起動して実行中であるにもかかわらず、Postgres の状態が「開始に失敗しました」に設定される場合があります。

18.28 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たため、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 `psycopg2` と `psycopg2-binary` という 2 つの異なるパッケージに分割され、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るようになりました。依存関係による地獄の問題を減らすために、ユーザーがインストール方法を選択できるようにしました。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 との互換性 (AlexanderKukushkin)

PostgreSQL12 以降では `recovery.conf` はなくなり、以前のすべての回復パラメータは `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を削除します。Patroni 構成は下位互換性を維持します。たとえば、`restore_command` は GUC であるにもかかわらず、`postgresql.recovery.conf.restore_command` セクションで指定することができ、Patroni はそれを PostgreSQL12 の `postgresql.conf` に書き込みます。

- 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)

この機能を使用したい場合は、Patroni 設定ファイルの `postgresql.authentication.rewind` セクションで `username` と `password` を定義してください。既存のクラスターの場合は、ユーザーを手動

で作成し、いくつかの機能に対する `GRANT EXECUTE` 権限を作成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。

- レプリカ上で実際の `primary_conninfo` 値と必要な `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 によって管理されるクラスターに変換する場合、レプリカの再起動を回避できる場合があります。

- IPv6 サポート (AlexanderKukushkin)

大きな問題が2つありました。Patroni RESTAPI サービスは `0.0.0.0` でのみリスンしており、`api_url` と `conn_url` で使用される IPv6 IP アドレスが適切に引用されていませんでした。

- Kerberos のサポート (AjithVilas, AlexanderKukushkin)

Patroni 設定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なります。

- `pg_ident.conf` の管理 (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ていると想定し、ファイルを更新しません。

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` エンドポイントにより、レプリカとプライマリ間で読み取りバランスがとれます。 `/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()` の使用は非推奨になりましたが、TLS1.1 など、間もなく非推奨となるプロトコルをまだ許可していました。

Logging improvements

- 2 段階のログ記録 (AlexanderKukushkin)

すべてのログメッセージは、最初にメモリ内のキューに書き込まれ、その後、別のスレッドから標準エラー出力またはファイルに非同期的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 (構成可能)。制限に達すると、Patroni はログを失い始めますが、HA ループをブロックするよりはまだマシです。

- `GET/OPTIONSAPI` 呼び出しとレイテンシのデバッグログを有効にしました (JanTomsa)

これは、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。

- 再試行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ときの最後の例外をログに記録します。DCS への通信が失敗した場合に、いくつかの問題をデバッグするのに役立つことが期待されます。

Improvements in patronictl

- スケジュールされたスイッチオーバーと再起動のためのダイアログを強化 (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ていなかったため、誤解を招く恐れがありました。

- 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は、黙って無視するのではなく (誤解を招く可能性があります)、構成ファイルについて詳しく説明してください。

- EDITOR のフォールバック値を追加 (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、ほとんどのシステムで利用できるはずの `vi` よりも `editor` を試すことです。

Improvements in Consul support

- Consul 整合性モードを指定できるようにしました (JanTomsa)

整合性モード [here](#) _について詳しくは、こちらをご覧ください。

- SIGHUP で Consul 設定をリロード (CameronDanielKucera, AlexanderKukushkin)

これは、誰かが `token` の値を変更する場合に特に便利です。

Bugfixes

- スwitchオーバー/フェイルオーバーの例外的なケースを修正しました (SharoonThomas)

RESTAPI にアクセスできず、DCS をフォールバックとして使用している場合、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す (AlexanderKukushkin)

以前は `unix_socket` に対してのみ開かれていたため、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/"`

- 元のリーダーが先行している場合でも、同期ノードは健全であるとみなします。 (AlexanderKukushkin)

プライマリが DCS へのアクセスを失った場合、Postgres は読み取り専用で再起動されますが、他のノードが RESTAPI 経由で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。このような状況により、古いプライマリが同期スタンバイよりも先に WAL 位置を報告していたため、同期スタンバイが昇格できませんでした。

- 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するほか、いくつかのマイナーな修正を加えました。

18.29 バージョン 1.5.6

New features

- プロキシのセットを介した etcd クラスターのサポート作業 (AlexanderKukushkin)

etcd クラスターに直接アクセスできず、プロキシのセットを介してアクセスできない場合があります。この場合、Patroni は etcd トポロジ検出を実行せず、プロキシホスト経由で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が変更されました。(AlexanderKukushkin)

ロールが「master」または「standby_leader」から「replica」、または「replica」から「standby_leader」に変更された場合、「on_restart」コールバックは呼び出されなくなり、「on_role_change」コールバックが優先されます。

- postgres の起動方法を変更する (AlexanderKukushkin)

それ自体を実行する代わりに ``multiprocessing.Process`` を使用し、ポストマスター PID を Patroni プロセスに送信するには ``multiprocessing.Pipe`` を使用します。その前はパイプを使用していて、stdin が閉じられた状態でポストマスタープロセスを終了していました。

Bug fixes

- RESTAPI によって返されるスタンバイリーダーのロールを修正しました。(AlexanderKukushkin)

``standby_leader`` ではなく、誤って ``replica`` を返していました。

- コールバックを強制終了できなかった場合は終了を待ちます (JulienTachaires)

Patroni には、新しいコールバックをキャンセルしていた `sudo` の下で実行されているコールバックスクリプトを終了するのに十分な権限がありません。実行中のスクリプトを強制終了できなかった場合、Patroni はスクリプトが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。

- `dcg.get_cluster` メソッドにかかるロック時間を短縮しました (AlexanderKukushkin)

ロックが保持されているため、DCS の速度が低下し、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である場合の 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` の不要な使用を削除します (AntsAasma)

それは作業ディレクトリを解決できるかどうかに依存します。後でファイルシステムからリンクが解除されたディレクトリで Patroni が起動された場合に何が失敗するかによって決まります。

- Etcd との通信時に SSL バージョンを強制しないようにしました。(AlexanderKukushkin)

何らかの理由で、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 を強制します。この問題はパトローニ側で解決しました。

18.30 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、`patronictl` リストの出力が改善され、多くのバグが修正されています。

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)

以前は、設定ファイルまたは「`PATRONI_ETCD_URL`」の一部としてのみ設定できましたが、必ずしも便利ではありませんでした。

- 以前のマスターを自動的に再初期化できるようにしました (AlexanderKukushkin)

`pg_rewind` が無効になっているか使用できない場合、タイムラインが分岐するため、元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`」を設定することで変更できます。これが設定されると、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。

- `patronictl` リストのタイムラインに関する情報を表示 (AlexanderKukushkin)

古いレプリカを検出するのに役立ちます。それに加えて、ポート値がデフォルトでない場合、または同じホスト上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 構成に関する情報、履歴ファイル、そして最後に最も重要な `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により、削除されなくなります。

Bug fixes

- リーダー監視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ランダムな少量の追加待機時間を加えることによって増加します。最大持続時間に「待機/16」時間が追加されます。この場合、どちらが大きいかに応じて、「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経由で postgres に接続する場合は、常に `replication=1` を使用してください。 (AlexanderKukushkin)

Postgres10 以降、`pg_hba.conf` 内の `database=replication` を含む行は、パラメーター `replication=database` を使用した接続を受け入れません。

- wal 専用スタンバイクラスタの `Recovery.conf` に `Primary_conninfo` を書き込まないでください。 (AlexanderKukushkin)

「`standby_cluster`」設定に「`host`」も「`port`」も定義されていないにもかかわらず、Patroni は「`primary_conninfo`」を「`recovery.conf`」に入れていましたが、これは役に立たず、多くのエラーを生成し

ていました。

18.31 バージョン 1.5.4

このバージョンでは、柔軟なロギングが実装され、多くのバグが修正されています。

New features

- 伐採インフラの改善 (AlexanderKukushkin、LucasCapistrant、AlexanderAnikin)

ロギング設定は、環境変数からだけでなく、Patroni 設定ファイルからも設定できます。構成を更新してリロードを実行するか、Patroni プロセスに SIGHUP を送信することで、実行時にロギング構成を変更できるようになります。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、一定のサイズに達したときにローテーションすることが可能になりました。これに加えて、カスタム日付形式のサポートと、各 Python モジュールのログレベルを微調整する機能が追加されました。

- リーダー選挙中に現在のタイムラインを考慮できるようにする (AlexanderKukushkin)

現在、既知の最新のタイムラインに載っていないにもかかわらず、ノードが自身を最も健全なノードであるとみ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避けたいことがあります。これは、`check_timeline` パラメータを `true` に設定することで実現できます (デフォルトの動作は変更されません)。

- スーパーユーザー資格情報の要件が緩和されました

Libpq を使用すると、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 ファイルまたは pg_hba.conf の信頼認証方法のいずれかに依存します。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 ConsulService の登録とチェック間隔を設定する機能を実装しました (AlexanderKukushkin)

Consul でのサービスの登録は 1.5.0 で追加されましたが、これまでは patroni.yaml 経由でのみ有効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 をオフに設定します。 (AlexanderKukushkin)

クラスターが完全に機能するまでは、WAL と履歴ファイルのアーカイブは避けたいと考えています。カスタムブートストラップに pg_upgrade が含まれる場合、これは非常に役立ちます。

- 起動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)

これは、Patroni が起動したばかりのときに DCS を叩くの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削減しました (AlexanderKukushkin)

それらは無害ですが、かなり迷惑で、時には恐ろしいものでした。

- 作成時に、recovery.conf の rw 権限を明示的に保護します (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、patroni/postgres ユーザー以外にはこのファイルを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、このような例外は通常のログを混乱させて標準出力に記録されていました。

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 Patroni ログとともに表示できるようになります。Patroni および Postgres のログは標準ツール (docker ログ、kubectl など) を使用して使用できるため、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標準エラー出力に書き込むときに、Patroni がポストマスター pid 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した (PavelKirillov)

明示的に言及されていない限り、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和しました (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たり、create_replica_methods をリストとして指定したりすることはできませんでした。

18.32 バージョン 1.5.3

互換性とバグ修正のリリース。

- ZooKeeper に対して Python3 を使用して実行するときの安定性が向上しました (AlexanderKukushkin)

「loop_wait」の変更により、Patroni が飼育員から切断され、再接続できなくなりました。

- postgres9.3 との壊れた互換性を修正しました。 (AlexanderKukushkin)

9.3 は replication='database' を認識しないため、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 Consul セッションを更新するようにし、Consul セッション例外の処理を改善しました。 (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。セッション更新を時間通りに呼び出さず、セッションエラーを適切に処理しなかったことが、プライマリの降格を引き起こしていました。

18.33 バージョン 1.5.2

互換性とバグ修正のリリース。

- kazoo-2.6.0 との互換性 (AlexanderKukushkin)

リクエストが適切なタイムアウトで実行されることを確認するために、Patroni は python-kazoo モジュールの `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少し変更されました。

- Consul クラスタ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)

クラッシュは `touch_member` メソッドの不正な実装が原因で発生していました。これはブール値を返し、例外は発生しません。

18.34 バージョン 1.5.1

このバージョンでは、永続レプリケーションスロットのサポートが実装され、pgBackRest のサポートが追加され、いくつかのバグが修正されています。

New features

- 永続的なレプリケーションスロット (AlexanderKukushkin)

永続的なレプリケーションスロットは、フェイルオーバー/スイッチオーバーでも保持されます。つまり、新しいプライマリ上の Patroni は、プロモートの実行直後に、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「`patronictl edit-config`」を使用して設定できます。初期設定は *[bootstrap.dcs](#)* で行うこともできます。

- pgbackrest サポートを追加 (YogeshSharma)

pgBackrest は既存の \$PGDATA フォルダーに復元できます。これにより、最後のバックアップ以降変更されていないファイルが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

- 「スタンバイクラスタ」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。

- クラスタ管理が一時停止され、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。

18.35 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し、Windows 上での実行の実験的サポートを導入し、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を提供します。

New features

- スタンバイクラスター (DmitryDolgov)

1 つ以上の Patroni ノードは、プライマリノードと並行して (つまり、別のデータセンター内で) 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 つはリモートマスターから直接レプリケートすることを選択し、他のレプリカはカスケード方式でリモートマスターからレプリケートします。この機能の詳細な説明といくつかの構成例は、[here](#) にあります。

- Consul でのサービスの登録 (PavelKirillov, AlexanderKukushkin)

consul *configuration* の `register_service` パラメータが有効な場合、ノードは名前 `scope` とタグ `master`、`replica`、または `standby-leader` を持つサービスを登録します。

- 実験的な Windows サポート (PavelGolub)

今後は Windows 上で Patroni を実行できるようになりますが、Windows のサポートは新しく、Linux の対応物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

- `patronictl-k/--insecure` フラグを追加し、`restapicert` をサポートしました。(WilfriedRoset)

以前は、RESTAPI が自己署名証明書によって保護されていた場合、`patronictl` は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。証明書検証を完全にスキップしたり、設定の `:ref:ctl: <patronictl_settings>` セクションで CA 証明書とクライアント証明書を提供したりするように「`patronictl`」を設定できるようになりました。

- `patronictl` スイッチオーバー/フェイルオーバー出力から `nofailover` タグを持つメンバーを除外します。(AlexanderAnikin)

以前は、`patronictl` 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、これらのメンバーが候補として誤って提案されていました。

Stability improvements

- `pg_controldata` のキーと値以外の出力行を解析しないようにしました。(AlexanderAnikin)

特定の状況下では、`pg_controldata` はコロン文字のない行を出力します。これにより、`pg_controldata` 出力を解析する Patroni コードでエラーが発生し、実際の問題が隠蔽されてしまいます。多くの場合、このような行は、通常の出力の前に、つまりバイナリの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、`pg_controldata` によって示される警告として出力されます。

- リーダー選出時のエラーメッセージにメンバー名を追加しました (JanMussler)

リーダーの選出中、Patroni はクラスターの既知のメンバーすべてに接続し、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、メンバーの名前も含まれます。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その名前は示されず、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すぐに WAL ポジションを予約します (AlexanderKukushkin)

9.6 以降、「pg_create_physical_replication_slot」関数には追加のブール値パラメータ「immediately_reserve」が提供されます。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 位置を予約しないため、スロットの作成と接続の間の時間枠でクライアントが必要とする一部のセグメントが失われる可能性があります。最初のクライアント接続。

- 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」で実行すると、Patroni が「synchronous_standby_names」に「*」を入れて、ほとんどのレプリケーション接続の同期状態を「潜在的」に変更する場合があります。以前は、Patroni は状態が「async」の候補のみを考慮していたため、このような状況では同期候補を選択できませんでした。

18.36 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 を返すという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誘導される可能性があります。

- 降格時に is_leader ステータスをリセット (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 で応答を停止していることを確認します。

- 新しい「cluster_unlocked」フィールドを API 出力に追加しました (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。これは、レプリカの 1 つ以外の他のノードをクエリできない場合に使用できます。

18.37 バージョン 1.4.5

New features

- 新しい postgres 設定を適用する際のログ記録を改善しました (DonSeiler)

Patroni は、変更されたパラメータ名と値をログに記録します。

- Python3.7 との互換性 (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場合、DCS の状態を「停止」に設定します。(TonySorrentino)
これにより、「patronictl list」コマンドでメンバーの状態が「停止」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する場合に記録されるメッセージを改善しました (AntsAasma)
前回のものは混乱を超えていました。
- patronictl リロード機能を実装しました (DonSeiler)
それまでは、RESTAPI を呼び出すか、Patroni プロセスに SIGHUP シグナルを送信することによってのみ構成をリロードできました。
- レプリカとして開始するときに、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」の値と、グローバル設定で設定されたその他のパラメータの値は、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。この問題が発生すると、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、`pending\_restart` フラグを設定することで、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 を起動するとき、Patroni は PATH、LC_ALL、LANG 環境変数が設定されている場合はそれらを渡していました。現在は LD_LIBRARY_PATH でも同じことを行っています。誰かが PostgreSQL を標準以外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)
それが実際には配列であることを明確にするためです。下位互換性のために、古い名前も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 によるレプリカ起動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ようにしてください。
- 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 がパーティション化されている場合、Patroni が以前の (降格された) マスター上のマスター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、クエリパラメータを渡すために `dict` の代わりに `list` を使用するようになりました。
- 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かった例外により、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回復を実行します。(AlexanderKukushkin)
`pg\_controldata` が「運用中」、「シャットダウン中」、または「クラッシュ回復中」を報告する必要があります。それ以外の場合はすべて、クラッシュからの回復は必要ありません。

- 構成エラーの処理を改善しました (HenningJacobs、AlexanderKukushkin)

Patroni 設定ファイルを更新し、Patroni プロセスに SIGHUP を送信することで、実行時に多くのパラメーター (`restapi.listen`` を含む) を変更することができます。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場合に、「restapi」スレッドから不明瞭な例外が排除されます。

18.38 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)

フェイルオーバーにもスイッチオーバーにも直接的な影響はありませんでしたが、まれに、前のリーダーがロックを解放したときに成功が早すぎて報告され、「/にフェイルオーバーしました」ではなく「/なし/にフェイルオーバーされました」が生成されました。「desired-node」メッセージ。

- Postgres パラメータ名の大文字と小文字を区別しないようにしました。 (AlexanderKukushkin)

Postgres パラメータのほとんどには、`snake_case` 名が付いていますが、このルールには 3 つの例外があります (`DateStyle`、`IntervalStyle`、`TimeZone`)。Postgres は、別の大文字と小文字で記述された場合 (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、開始を中止します。 (AlexanderKukushkin)

Patroni は、すでに実行されている Postgres インスタンスに自身をアタッチできます。レプリカにアクセスする前に、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。

- `patronictlscaffold` の動作を修正しました (AlexanderKukushkin)

`json` で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にマスターを降格しません。 (AlexanderKukushkin)

メンテナンス中に、DCS は読み取り要求には応答し続けながら、書き込み要求は失敗し始める場合があります。この場合、Patroni は DCS の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設定していました。

- Patroni が新しいポストマスタープロセスに気づいたときにレプリケーションスロットを同期するようになりました。 (AlexanderKukushkin)

Postgres が再起動されている場合、Patroni はレプリケーションスロットのリストが期待どおりであることを確認する必要があります。

- 一時停止から復帰した後、`sysid` を確認し、レプリケーションスロットを同期します。 (AlexanderKukushkin)

「メンテナンス」モード中に、データディレクトリが完全に書き換えられる可能性があるため、「データベースシステム識別子」がまだクラスターに属しており、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。

- ポストマスターロックファイルが存在するデータディレクトリで Postgres を実行しないと開始できない可能性があった問題を修正しました。(AlexanderKukushkin)

ポストマスター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。

- 誤って消去された DCS の保護を改善しました (AlexanderKukushkin)

Patroni には、このような場合にフェイルオーバーを防ぐためのロジックが多数用意されています。すべてのキーを元に戻すこともできます。ただし、この変更が行われるまでは、/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 が発生した場合は終了しない (OleksiiKliukin)

クラスターシステム 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、クラスターの再初期化が必要になる可能性が高くなります。結果メッセージにそれを記載してください。Patroni を終了しないでください。終了しないと再初期化が行われません。

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)

Kubernetes1.10.0 以降では、「[]」の代わりに「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便利です。

- Basebackup 組み込みメソッドへのオプションを許可 (OleksiiKliukin)

カスタムレプリカ作成メソッドの定義方法と同様に、構成内で「`basebackup`」セクションを定義することで、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形式にあります。`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。

18.39 バージョン 1.4.3

Improvements in logging

- ログレベルを環境変数から設定できるようにしました (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

- 監視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ません。(AlexanderKukushkin)

etcd 構成にホストが 1 つだけあり、そ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必要があります。
- カスタムブートストラップの後に、bootstrap.pg_hba の内容を 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> で紹介されました

18.40 バージョン 1.4.2

Improvements in patronictl

- スケジュールされたフェイ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)

フェイルオーバー機能とスイッチオーバー機能はバージョン 1.4 で分離されましたが、「patronictl」は依然として「スケジュールされたスイッチオーバー」ではなく「スケジュールされたフェイルオーバー」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 (AlexanderKukushkin)

一部の構成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、RESTAPI や DCS にノードのステータスを書き込むときに、すでにそのヒントを提供していましたが、それを表示する簡単な方法はありませんでした。
- 設定ファイルの cluster_name で動作するように show-config を作成しました。(AlexanderKukushkin)

これは「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す。(AlexanderKukushkin)

initdb またはカスタムブートストラップ中に、pgdata は空ではないが、pg_controldata がまだ書き込まれていない時間帯があります。このような場合、pg_controldata 呼び出しはエラーメッセージとともに失敗していました。
- psutil から発生した例外を処理しました (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。検査対象のプロセスがすでに消滅している可能性もあり、その場合は `NoSuchProcess` が発生します。

Kubernetes support improvements

- k8sAPI からのエラーを鵜呑みにしないでください (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が、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記録する必要があります。ここでの変更は、Kubernetes 権限の問題のデバッグに役立ちます。

- Kubernetes サンプルの 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)

それまでは「feature/k8s」を使用していましたが、それは時代遅れになりました。

- k8s で patroni を実行するために適切な 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、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。

18.41 バージョン 1.4.1

Fixes in patronictl

- フェイルオーバー先のメンバーの推奨リストに現在のリーダーを表示しません。(アレクサンダー・ククシュキン)

patronictl フェイルオーバーは、クラスター内にリーダーがある場合でも機能する可能性があるため、フェイルオーバー先のメンバーのリストから除外する必要があります。

- patronictl スイッチオーバーを古い PatroniAPI と互換性のあるものにしました。(AlexanderKukushkin)

POST/switchoverRESTAPI 呼び出しがステータスコード 501 で失敗した場合は、/failover エンドポイントに対して再度実行されます。

18.42 バージョン 1.4

このバージョンでは、Kubernetes を DCS として使用するためのサポートが追加されており、Etcd、Zookeeper、または Consul を追加展開することなく、Patroni を Kubernetes のクラウドネイティブエージェントとして実行できるようになります。

Upgrade notice

pip 経由で Patroni をインストールすると、依存関係 (Etcd、Zookeeper、Consul、Kubernetes のライブラリ、または AWS のサポートなど) が取り込まれなくなります。こ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 (例: `pipinstallpatroni[etcd,kubernetes]`)。

Kubernetes support

Kubernetes ベースの DCS を実装します。エンドポイントのメタデータは、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。エンドポイントの使用に加えて、Patroni は ConfigMap も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](#) を参照してください。

Stability improvements

- ポストマスタープロセスを別のオブジェクトに分解します (AntsAasma)

このオブジェクトは、pid と開始時刻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 サイクルの各グループで Patroni によって発行される SELECT の量を最小限に抑えます。(AlexanderKukushkin)

HA ループの反復ごとに、Patroni は回復ステータスと絶対的な壁の位置を知る必要があります。今後、Patroni はこの情報を取得するために、レプリカで 2 つ、マスターで 3 つの SELECT を実行するのではなく、1 つの SELECT のみを実行します。

- ロックを持っている場合にのみ、シャットダウン時にリーダーキーを削除します (AntsAasma)

無条件の削除により、不要で誤解を招く例外が生成されていました。

Improvements in patronictl

- patronictl に version コマンドを追加 (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。

- 一部の patronictl コマンドで、cluster_name 引数の指定をオプションにしました (AlexanderKukushkin, AntsAasma)

patronictl が scope が定義された通常の Patroni 設定ファイルを使用している場合は機能します。

- 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)

それまでは、この情報は Patroni ログから取得するか、DCS から直接取得するしかありませんでした。

- patronictl reinit の改善 (AlexanderKukushkin)

Patroni が他のアクション、つまり postgres を開始しようとして忙しいときに、patronictl reinit が続行を拒否することがありました。patronictl は、そのような長時間実行されるアクションをキャンセルするコマンドを提供しておらず、唯一の (危険な) 回避策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、再初期化を続行する前に、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)

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、patronictl は待機します。このような動作は、DCS 内のすべてのノードの pause フラグを RESTAPI 経由で公開することによって実現されます。

- patronictl failover の名前を patronictl switchover に変更しました (AlexanderKukushkin)

以前の failover は、実際にはスイッチオーバーのみを行うことができました。リーダーなしでは集団で進むことを拒否した。

- patronictl failover の動作を変更しました (AlexanderKukushkin)

リーダーがない場合でも機能しますが、その場合は新しいリーダーと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 経由で現在のタイムラインを公開します (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 経由でアクセスでき、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を付加し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した (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、対応する同期レプリカと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ものを除く)。

Allow multiple hosts for Etcd

- Etcd 設定に新しい「hosts」パラメータを追加しました。 (AlexanderKukushkin)

このパラメータには、実行中の etcd クラスタメンバーのリストを検出して設定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。作業中に何らかの理由で、この検出されたホストのリストが使い果たされた場合 (そのリストに利用可能なホストがない場合)、Patroni は「hosts」パラメータから初期リストに戻ります。

18.43 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。 (アリズ・アスマ)

postmaster.pid がクリーンアップされ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() の誤検知が発生し、あらゆる種類の不正な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)

マスター上のデータディレクトリが強制的に削除されると、postgres プロセスがしばらくの間生きたままになり、元のマスターの代わりに作成されたレプリカの起動や複製が妨げられることがあります。この修正により、Patroni はポストマスターの PID 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古いポストマスターがまだ実行されている場合に備えて古いポストマスターを終了できるようになります。

- postgres マスターが停止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は `pg_rewind` を実行できません。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存在しない場合にのみ開始されます。

Consul improvements

- Consul にデータセンター構成を提供できるようにしました (ViliusOkockis, AlexanderKukushkin)

それまでは、Patroni は常に、それが実行されてい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-Token` http ヘッダーでトークンを送信します。 (AlexanderKukushkin)

`consul.token` が Patroni 設定で定義されている場合、常に「`X-Consul-Token`」http ヘッダーで送信されます。python-consul モジュールは ConsulRESTAPI と「一貫性」を保とうとしますが、`consul.token` のクエリパラメータとしてトークンを受け入れませんが、「`X-Consul-Token`」ヘッダーでは動作します。

- 指定された値が可能な最小値より小さい場合、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 がサポートする最小 TTL 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、クラスターが異常な状態になります。

Other improvements

- 環境変数 `PATRONI_LOGFORMAT` を介してカスタムログ形式を定義します (StasFomin)

システムロガーによってすで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)、Patroni ログ内のタイムスタンプやその他の同様のフィールドを無効にすることができます。

18.44 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にロールを「初期化されていない」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、フェイ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は、シングルユーザーモードで `postmaster` を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、制御データが変更されないままになる可能性が非常に高くなります。クリーンな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)

指定しない場合、Consul は「serfHealth」を使用します。一方では、孤立し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 (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 バッファを返します。int と str の比較でエラーが返されなかったため、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 を発生させることによって行われます。

18.45 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](#) に渡すための推奨される方法になりました。

- Consul の詳細設定 (AlexanderKukushkin)

scheme、token、クライアントおよび CA 証明書 scheme を指定する可能性があります。

- 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

- 「TTL ロックを解除できませんでした」というメッセージが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、適切なログが記録されていないと、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、その値を引用符で囲む必要があります。(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事実上無効にし、動作を継続します。

Different bugfixes around pause state, mostly related to watchdog (アレクサンダー・ククシュキン)

- Watchdog 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 Watchdog をアクティブにしないようにする
- 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は API からクエリを実行しないでください

18.46 バージョン 1.3.3

Bugfixes

- `synchronous_mode_strict` がオンになっている場合でも、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空の `pg_ident.conf` ファイルが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)
- `pg_hba.conf` で postgres だけでなくすべてのデータベースにアクセスできるようにしました (FrancoBellagamba)

18.47 バージョン 1.3.2

Bugfix

- `patronictl edit-config` が ZooKeeper で動作しませんでした。(AlexanderKukushkin)

18.48 バージョン 1.3.1

Bugfix

- `_MemberStatus` の変更により API 経由のフェイルオーバーが壊れていました。(AlexanderKukushkin)

18.49 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、`patronictl` に構成編集が追加され、Linux での Watchdog サポートが実装されています。さらに、これは PostgreSQL 10 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互換性既知の問題はありません。バージョン 1.2 からの設定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 が再起動する) か、最初に Patroni を *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 (一時停止モードの Patroni は停止しようとしません) ことでアップグレードできます。PostgreSQL を開始)、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の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、`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。ブートストラップコマンドは、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、バックアップからブートストラップしてポイントインタイムリカバリを実行できるようになります。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)

以前の Patroni には、pg_rewind をトリガーする条件が固定されていました。つまり、元のマスターを起動するとき、クラスター内の他のすべてのノードで指定されたノードへのスイッチオーバーを実行するとき、または nofailover タグを持つレプリカが存在するときです。これらすべてのケースに共通するのは、一部のレプリカが新しいマスターよりも先に存在する可能性があるということです。場合によっては、pg_rewind が何も行わなかったり、必要なときに実行されなかったりする場合があります。この限られたルールのリストに依存する代わりに、Patroni は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) 比較し、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

- 厳密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 が有効で、マスターにレプリカが接続されていない場合、Patroni はマスターを書き込み可能な状態に保つために同期レプリケーションを無効にします。synchronous_mode_strict オプションは、このオプションが設定されている場合、レプリカが不足している場合でも Patroni が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。自動フェイ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、ストリクトモードでは、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つしかない場合には完全に書き込みが行われないことが保証されます。

Configuration editing with patronictl

- patronictl に設定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のいずれかをサポートします。

Linux watchdog support

- Linux 用の Watchdog サポートを実装する (AntsAasma)

Patroni が実行されていない、または応答していないノード (高負荷などの理由) を再起動するために、Linux ソフトウェア Watchdog をサポートします。Linux ソフトウェア Watchdog は、応答しないノードを再起動します。Patroni 設定の Watchdog セクションから、使用する Watchdog デバイス (デフォルトでは `/dev/watchdog`) とモード (オン、自動、オフ) を設定できます。[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 PostgreSQL 10 のすべてのベータ版と互換性があり、リリースされる PostgreSQL 10 と互換性があることが期待されます。

PostgreSQL-related minor improvements

- Patroni 設定ファイルまたは DCS の動的設定を介して 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` の内容を定義できるようにします。これにより、すべてのノードにログを記録し、手動で変更して設定をリロードするのではなく、DCS で `pg_hba.conf` を定義するだけで済むため、複数のノードでの `pg_hba.conf` の管理が簡素化されます。

定義すると、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 パラメータが設定されている場合、Patroni はこれを無視します。

- 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)

`use_unix_socket` オプションを Patroni 構成の `postgresql` セクションに追加します。 `true` に設定され、PostgreSQL `unix_socket_directories` オプションが空でない場合、Patroni はその最初の値を使用してローカル PostgreSQL クラスターに接続できるようになります。 `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 (@jour) 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` 構成ディレクティブ `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 によって書き込み可能である必要があります。

Bug fixes and stability improvements

- `EtcdEventIndexCleared` 例外と `EtcdWatcherCleared` 例外を処理します (AlexanderKukushkin)

無駄な再試行を回避することで、Etcd によって監視操作が終了された場合の回復が高速化されます。

- Etcd 障害時に回転するエラーを削除し、ログスパムを削減します (AntsAasma)

2 回目以降の Etcd 接続失敗時に、すぐに再試行してスタックトレースをログに出力することは避けてください。

- 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する (OleksiiKliukin)

NLS で構築された PostgreSQL の英語以外のロケールでの「起動中にポストマスターがマルチスレッド化されました」という致命的なエラーを回避します。

- レプリケーションスロットを削除する際の追加チェック (AlexanderKukushkin)

場合によっては、WAL 送信者によって Patroni がレプリケーションスロットを削除できないことがあります。

- 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 クラスターに対して余分な接続が開かれるという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する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)

これを行わないと、ノードが別のクラスターに属しているというエラーが発生する可能性があります。Patroni はクラスター内にリーダーが存在する必要のないブートストラップ方式でブートストラップされながら通常の業務を続行したためです。

- WAL-E レプリカ作成方法を改善しました (JoarWantborg、AlexanderKukushkin)。
 - WAL-E ベースのバックアップを解析するときに csv.DictReader を使用し、スペースで区切られた日付と時刻を持つ 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、レプリカから現在の WAL 位置を取得する機能をサポートします。以前は、コードはマスターノードでのみ利用可能なシステム情報関数を呼び出すために使用されていました。

18.50 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イ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。(アリズ・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ときは常に同期レプリケーションを有効にします。同期モードが有効な場合、Patroni はマスター障害時に同期レプリケーションを行っていたスタンバイにのみ自動的にフェイルオーバーします。これは事実上、そのような場合に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% 正常でない場合は、`leader optime` キーに保存されているリーダーの位置を更新しないでください。リーダーキーの更新に失敗した場合は直ちに降格します。(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 に対して再試行およびタイムアウト戦略を実装します。(アレクサンダー・ククシュキン)
 - `--dcs` と `--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 パラメータを `postgresql.conf` に書き込みます。(アレクサンダー・ククシュキン)
- Patroni によって構成された PostgreSQL を `pg_ctl` だけで起動できるようになります。
- 構成にユーザーが存在しない場合は例外を回避します。(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。この修正が行われる前は、`patronictl` が一時停止を実行しようとするノードが異常な場合、`patronictl` はバイルアウトしていました。(アレクサンダー・ククシュキン)
- リーダーウォッチ機能を改善します。(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(タイムアウトになるかリーダーキーが変更されるまでスリープ状態でした)。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され、PostgreSQL が停止/起動または再起動しているときは監視されません。

- SIGCHILD を PID1 として処理するときに競合状態に陥るのを避けてください。(AlexanderKukushkin)

以前は、Patroni 内の同じプロセスが新しいプロセスを生成し、そこから SIGCHILD を処理したため、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/execs` を使用し、元の PID1 プロセスに子からの信号の処理を任せます。

- WAL-E の復元を修正しました。(オレクシー・クルーキン)

以前の WAL-E 復元では、マスターとの協議を完全に回避するために `no_master` フラグが使用されていたため、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、レプリケーション方法として Patroni WAL-E リストアが選択される可能性があります。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確認されます。さらに、再試行メカニズムがより堅牢になり、他の細かい点も処理されます。

- 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(たとえば、ノードが受信する過剰なトラフィックが原因)の失敗を回避します。

- 開始状態とマスター開始タイムアウトを実装します。(アリツ・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、PostgreSQL が実行中であるとみなして喜んで踏み続けました。これにより、実際には実行されていないのに PostgreSQL がリスト上で実行中として表示され、競合状態が発生してフェイルオーバー、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たクラッシュリカバリと巻き戻しのミスが発生しました。この変更により、`master_start_timeout` パラメータが追加され、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、マスターがクラッシュしたときにフェイルオーバー候補が存在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 はマスター上で PostgreSQL を起動しようとした後、タイムアウトの間待機します。有効期限が切れると、可能であればフェイルオーバーされます。手動フェイルオーバー要求は、マスターのクラッシュ中にタイムアウト期限が切れる前であっても受け付けられます。

`timeout` パラメータを `restart` API エンドポイントと `patronictl` に導入します。これが設定され、再起動にタイムアウトよりも長い時間がかかると、PostgreSQL は異常とみなされ、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。(アリズ・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できない場合(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。`superuser` 認証が `pg_rewind` 関

連の Patroni 設定セクションにない場合、`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します。

- 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が生成され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 に設定されているにもかかわらず、このリーダーキーへの更新が失敗する場合は、以前のマスターを昇格させないようにしてください。(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ともにパーティション化されている場合に、その役割を維持し続けるという問題が回避されます。

Miscellaneous

- ブートストラップに `post_init` 構成オプションを追加します。(アレハンドロ・マルティネス)

Patroni は、`initdb` を実行し、新しいクラスターの PostgreSQL を起動した直後に、このオプションのスクリプト引数呼び出します。スクリプトは、`superuser` で接続 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、新しいクラスターに拡張機能を作成したりする場合に便利です。

- 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、あるフラグから別のフラグに変わるときに `pending_restart` 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 メイン `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しました。(ローリ・アップル)
- Patroni 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。(オレクシー・クルーキン)

さまざまなデバイス (スマートフォンを含む) からドキュメントを簡単に表示し、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。(オレクシー・クルーキン)

Patroni は、小規模だが重大なバグ修正に関する新しいマイナーバージョンのリリースを避けるために、`major.minor.patch` バージョンスキーマに従います。すべてのパッチを含む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。

18.51 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で 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再起動によるメンテナンスが改善され、Patroni と Etcd または Zookeeper の対話の回復力が向上し、`patronictl` が大幅に強化されました。

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 リリースノートで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

- Patroni を PostgreSQL インスタンスの管理から一時的に切り離すための一時停止モードを導入しました (MuratKabilov、AlexanderKukushkin、OleksiiKliukin)。

以前は、PostgreSQL を終了せずに Patroni を停止するには、`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全体で PostgreSQL から Patroni を切り離します。これは、Pacemaker のメンテナンスモードに似ています。Patroni は DCS でメンバーキーとリーダーキーを更新する責任を負いますが、そのプロセスで PostgreSQL サーバーを起動、停止、再起動することはありません。いくつかの例外があります。たとえば、手動フェイルオーバー、再初期化、再起動は引き続き許可されます。 [a detailed description of this feature](#) を読み取ることができます。

さらに、`patronictl` は、一時停止モードを切り替えるための新しい `pause` および `resume` コマンドをサポートしています。

Scheduled and conditional restarts

- 再起動 API コマンドに条件を追加します (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が追加され、Patroni の再起動が強化されました。条件の中には、PostgreSQL ロールがマスターまたはレプリカの場合に再起動する、PostgreSQL のバージョン番号を確認する、構成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などがあります。

- スケジュールされた再起動を追加 (OleksiiKliukin)

将来の再起動をスケジュールできるようになりました。スケジュールされた再起動はノードごとに 1 つだけサポートされます。スケジュールされた再起動がなくなっただけの場合は、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、PostgreSQL のマイナーアップグレードを夜間にスケジュールして、管理スクリプトに `postgres` 固有のロジックを追加せずに、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することが可能になります。

- `patronictl` に条件付きおよびスケジュールされた再起動のサポートを追加しました (MuratKabilov)。

`patronictlrestart` 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消去するための `patronictlflash` コマンドもあります。

Robust DCS interaction

- `loop_wait` に応じて Kazoo タイムアウトを設定しました。 (AlexanderKukushkin)

当初、`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 の `loop_wait` は考慮されていませんでした。その結果、1 回の再試行にセッションタイムアウトよりも長い時間がかかる可能性があり、Patroni はロックを解除して降格する必要がありました。

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、ロックが失われる前に接続試行を再試行するのに十分な時間が残されました。

- 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)

クライアントに認識されている Etcd トポロジの更新を、元のリクエストが完了するまで延期します。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、クライアントは最新のノードのリストを取得するよりも、リクエストの結果を取得することを優先します。

どちらの変更も、ネットワークの問題に直面しても Patroni から DCS への接続をより堅牢にします。

Patronictl, monitoring and configuration

- API 経由で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、(接続の問題などにより) 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 に問い合わせる信頼性の高い方法はありませんでした。この変更により、`pg_stat_replication` の内容が `/patroni` エンドポイント経由で公開されます。

- `patronictlscaffold` コマンドを追加 (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` とリーダーを使用して作成され、リーダーキーとメンバーキーの両方が永続化されます。このコマンドは、レプリカのみで構成される Patroni クラスターが Patroni を認識しない外部マスターノードから複製される、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。その後、リーダーキーを削除して、Patroni ノードの 1 つを昇格し、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。

- PostgreSQL バイナリを見つけるための構成オプション `bin_dir` を追加 (AntsAasma)

複数の PostgreSQL バージョンの同時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しました。

カスタム構成ファイルのパスを許可します。これは Patroni によって管理されません ([details](#))。

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のあるものにします (FeikeSteenbergen)

PostgreSQL のバージョンに基づいて条件付き再起動を行う場合、Patroni が 2 桁のバージョン番号を理解していることを確認してください。

- `pkgutil` を使用して DCS モジュールを検索します (AlexanderKukushkin)

DCS モジュールを見つけるためにディレクトリを手動で走査する代わりに、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、正しいロールですで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で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、すでに実行中のノードにアタッチしている場合でも `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削除しない (MuratKabilov、OleksiiKliukin)

マスター上のアクティブな物理レプリケーションスロットをドロップしないようにします。いずれにしても、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上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 接続を閉じるようにしました。(AlexanderKukushkin)

PostgreSQL ノードの起動時に、Patroni に以前の接続をすべて強制的に閉じます。postmaster が SIGKILL で強制終了された場合に、以前の接続が再利用されるという罫を回避します。

- 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ていないスタンバイ名によってスロットの作成や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、スロット名に使用できない他のすべての文字を Unicode コードポイントに置き換えます。

18.52 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ています。また、多数のバグ修正も提供されます。

Upgrade notice

v0.90 以前からアップグレードする場合は、必ず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。DCS にはレプリケーション資格情報が保存されなくなったため、古いレプリカは新しいマスターに接続できなくなります。

Dynamic Configuration

- 動的な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 (マスターおよびすべてのレプリカ) 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は DCS で設定され、多くの場合、PostgreSQL や 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に、API 経由で表示される「再起動保留中」と呼ばれる特別なフラグを設定します。この場合、API を介して手動で再起動を発行する必要があります。

PatroniSIGHUP または `/reload` への POST により、構成ファイルが再読み取りされます。

変更できるパラメータ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 から **変更されました**。Patroni は依然として古い設定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 の設定と Patroni が接続するデータベース名を設定可能にしました (MisjaHoebe)
「database」および「config_base_name」設定パラメータを導入します。特に、PipelineDB やその他の PostgreSQL フォークで Patroni を実行できるようになります。
- 環境経由で一部の Patroni 設定パラメータを設定できる機能を実装しました (AlexanderKukushkin)
これらには、スコープ、ノード名、名前空間、およびシークレットが含まれており、動的環境、つまり Kubernetes での Patroni の実行が容易になります。詳細については、[supported environment variables](#) を参照してください。
- 環境ベースの構成を活用するために、組み込みの PatroniDocker コンテナを更新します (FeikeSteenbergen)。
- Zookeeper サポートを PatroniDocker イメージに追加しました (AlexanderKukushkin)
- Zookeeper と Exhibitionr の構成オプションを分割する (AlexanderKukushkin)
- patronicl が Patroni のコードを再利用して設定を読み取るようにしました。 (AlexanderKukushkin)
これにより、patronicctl は環境ベースの構成を利用できるようになります。
- 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の進行中に pg_controldata を呼び出さないようにしました。 (AlexanderKukushkin)
この変更により、バックアップからの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減少します。
- 多数の pg_rewind のコーナーケースを修正しました (AlexanderKukushkin)
ソースクラスターがマスターではない場合は、pg_rewind を実行しないでください。
さらに、新しいパラメータ `remove_data_directory_on_rewind_failure` が true に設定されていない限り、巻き戻しが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した。 (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、patroni 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) は DCS で公開されなくなりました。
- 降格呼び出し周りの非同期機構を修正しました。 (AlexanderKukushkin)
降格は、DCS 対話をブロックすることなく完全に非同期で実行されるようになりました。

- `patronictl` が設定されている場合、常に認証ヘッダーを送信するようにしました。(AlexanderKukushkin)
これにより、Patroni が承認を必要とするように設定されている場合、`patronictl` 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るようになります。
- `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` 用のサンプル `haproxy` テンプレート (AlexanderKukushkin)
`confide` を使用して、DCS の `patroni` 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、新規ユーザーにとってより使いやすいものにします (LauriApple)
- API は `pg_ctl` の停止中に `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が高まります。さらに、「`pg_ctl`」を介して開始、停止、再起動呼び出しのタイムアウトを設定する「`pg_ctl_timeout`」オプションを導入します。
- `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かつ堅牢にします。
- Zookeeper コードの短いネットワーク障害に対する回復力を強化しました (AlexanderKukushkin)
接続タイムアウトを減らして、Zookeeper の接続試行の頻度を増やします。

18.53 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ています。

Consul support

- Consul サポートの実装 (AlexanderKukushkin)
パトローニは Etcd と Zookeeper に加えて Consul と対戦します。接続パラメータは 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に利用できないことを返すようになります。
- `clonefrom` タグの実装を変更しました (AlexanderKukushkin)
以前は、ノード名を `clonefrom` に指定する必要があり、タグ付きレプリカが特定のノードから強制的に複製されました。新しい実装では `clonefrom` がブール型タグになります。これが `true` に設定されている場合、レプリカは他のレプリカがそこから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、フェイ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、以前のリーダーの降格、および再接続に関するいくつかの特殊なケースに対処します。

- 停止中にパトローニの子供たちが殺されないようシステムスクリプトを改善しました (JanKeirse、AlexanderKukushkin)

以前は、Patroni を停止するときに、`systemd` も PostgreSQL にシグナルを送信していました。Patroni は PostgreSQL を単独で停止しようとしたため、別のシャットダウン要求 (スマートシャットダウン、その後の高速シャットダウン) が送信されることになりました。その結果、レプリカの切断が早すぎて、元のマスターが降格後に再参加できなくなりました。Alexander による事前調査をもとに Jan によって修正されました。

- 元のマスターがレプリカとして再参加する前に `pg_rewind` を呼び出すことができなかったいくつかのケースを排除しました。 (OleksiiKliukin)

以前は、前のマスターがクラッシュした場合にのみ `pg_rewind` を呼び出していました。 `pg_rewind` がシステムに存在する限り、元のマスターに対して常に `pg_rewind` を実行するようにこれを変更します。これにより、レプリカが最新の変更を取得する前にマスターがシャットダウンされた場合 (つまり「スマート」シャットダウン中) が修正されます。

- 特に、単体テストと受け入れテストに対する数多くの改善により、Zookeeper と Consul (AlexanderKukushkin) のサポートが可能になりました。

- TravisCI を高速化し、Zookeeper (エキシビター) と Consul (AlexanderKukushkin) に対するテスト実行のサポートを実装します。

単体テストと受け入れテストは両方とも、各コミットまたはプルリクエストで Etcd、Zookeeper、および 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 クラスターに接続することによってシステム環境変数が読み取られる可能性が防止されます。

Configuration and control changes

- `patronictl` と Patroni の設定を統合しました (FeikeSteenbergen)

`patronictl` は、Patroni 自体と同じ設定ファイルを使用できます。

- Patroni が環境変数から設定を読み取れるようにします (OleksiiKliukin)

これにより、Patroni の構成の自動生成や、異なる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)

`patronictl` で Etcd 以外の DCS のサポートを有効にします。

18.54 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イルオーバー**を提供することで Patroni 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) をこのバージョンと組み合わせて使用☒☒する場合があります。スケジュールされたフェイ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

- patroni ノード (OleksiiKliukin) に *replicatefrom* タグと *clonefrom* タグのサポートを追加します。

replicatefrom タグを使用すると、レプリカはマスターではなくソースとして任意のノードを使用できるようになります。 *clonefrom* は最初のバックアップに対して同じことを行います。これらを組み合わせることで、Patroni は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なります。

- レプリケーション接続を実行していなくてもレプリカを初期化できる、レプリケーションメソッドの実行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場合に便利です。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では、yaml 設定で *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番に呼び出されます。

Patronictl, API and DCS improvements

- スケジュールされたフェイルオーバーを実装します (FeikeSteenbergen)。

patronictl 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。

- patronictl に *dbuser* および *password* パラメータのサポートを追加しました (FeikeSteenbergen)。
- PostgreSQL バージョンをヘルスチェック出力に追加しました (FeikeSteenbergen)。
- patronictl での Zookeeper のサポートを改善しました (OleksandrShulgin)
- python-etcd0.43 への移行 (AlexanderKukushkin)

Configuration

- Patroni(JanKeirse) 用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 接続用の設定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☒☒修正しました。

Tests

- Patroni(AlexanderKukushkin, OleksiiKliukin) を実行する実際のシナリオをチェックするために、*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、プルリクエストや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19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。

19.1 ガイドラインへの貢献

19.1.1 おしゃべり

If you have a question, looking for an interactive troubleshooting help or want to chat with other Patroni users, join us on channel `#patroni` in the [PostgreSQL Slack](#).

19.1.2 バグの報告

バグを報告する前に、必ず **reproduce it with the latest Patroni version** を確認してください。また、問題が **reproduce it with the latest Patroni version** _にすでに存在するかどうかを再確認してください。

19.1.3 テストの実行

動作テストを実行するための要件:

1. `contrib_` モジュールを含む PostgreSQL パッケージをインストールする必要があります。
2. PostgreSQL バイナリは「PATH」で使用できる必要があります。
`PATH=/usr/lib/postgresql/11/bin:\$PATHpython-mbehave` のような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(Etcd、Consul、Zookeeper など)でテストしたい場合は、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されていない/保護されていない接続を受け入れる必要があります。Etcd 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

# Moreover, you may want to run tests in different scopes for debugging purposes,
# the -s option include print output during test execution.
# Tests in pytest typically follow the pattern: FILEPATH::CLASSNAME::TESTNAME.
pytest -s tests/test_api.py
pytest -s tests/test_api.py::TestRestApiHandler
pytest -s tests/test_api.py::TestRestApiHandler::test_do_GET

# 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
```

19.1.4 Tox を使った検査

tox テストを実行するには、依存関係 (Python 以外) を 1 つインストールするだけで済みます。

```
pip install tox>=4
```

「behave」テストを実行したい場合は、docker もインストールする必要があります。

「tox.ini」の Tox 設定には、次のタスクを実行するための「環境」が含まれています。

- lint: `flake8` を使用した Python コード lint
- test: `pytest` を使用して利用可能なすべての Python インタープリターの単体テストを行い、TTY が検出された場合は XML レポートまたは HTML レポートを生成します。
- dep: `pipdeptree` を使用してパッケージの依存関係の競合を検出します
- type: `pyright` による静的型チェック
- black: `black` を使用したコードの書式設定
- 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: 上記のイメージで任意のコマンドを実行します
- docker-behave-etc: 上記の画像を使用して動作テスト用の tox を実行します
- py*behave: 利用可能な Python インタープリターを使用して behave を実行します (docker は使用しませんが、これは docker コンテナ内で呼ばれるものです)。
- docs: `sphinx` を使用してドキュメントをビルドします

ランニングトックス

デフォルトの環境リストを実行するには;dep、lint、test、docs の場合は、次を実行するだけです。

```
tox
```

「test」環境は、「test」というラベルを付けて実行できます。

```
tox -m test
```

「behave」の Docker テストは、「behave」というラベルを使用して実行できます。

```
tox -m behave
```

同様に、docs には「docs」というラベルが付いています。

他のすべての環境は、それぞれの環境名を使用して実行できます。

```
tox -e lint
tox -e py39-test-lin
```

「factors」を使用して部分的な環境リストを選択することもできます。たとえば、Python3.10 のすべての環境を実行したい場合:

```
tox -f py310
```

これは、以下にリストされているすべての環境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`と`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する開発者の利益を目的としています。Mac では「open」を実行し、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。この機能を無効にしたい場合は、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

「-mbehave」を指定した動作テストは、PG_MAJOR バージョン 11～16 に基づいて Docker イメージを構築し、すべての動作テストを実行します。これは実行に非常に長い時間がかかる可能性があるため、範囲を Postgres の選択したバージョン、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完全名と動作環境名を含めます。たとえば、Postgres14 が必要な場合は、次を使用します。

```
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、位置引数を渡して動作させることができます。これにより、Postgres のすべてのバージョンで Watchdog 動作機能のテストシナリオが実行されます。

```
tox -m behave -- features/watchdog.feature
```

もちろん、この 2 つを組み合わせることもできます。

19.1.5 プリリクエストの提供

1. リポジトリをフォークし、コードの変更を開発してテストします。
2. ユーザードキュメントの変更を反映します。
3. 変更の目的を明確に説明したプリリクエストを送信します。必要に応じて、既存の問題をリンクします。

プリリクエストに関するフィードバックはできるだけ早く受け取れます。

パトローニのハッキングを楽しんでください;-)

第20章 インデックスとテーブル