
Patroni Documentation

リリース 3.3.0

Zalando SE

2024 年 06 月 14 日

内容

第 1 章	はじめに	3
1.1	開発状況	3
1.2	技術要件/インストール	3
1.3	PostgreSQL ノードの数の計画	3
1.4	実行と構成	4
1.5	YAML 構成	4
1.6	環境設定	4
1.7	レプリケーションの選択	4
1.8	アプリケーションはスーパーユーザーを使用しないでください	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	Psycopg	7
2.3	pip の一般的なインストール	7
2.4	Linux でのパッケージのインストール	8
2.5	アップグレード	10
第 3 章	パトローニ構成	11
3.1	動的構成設定	11
3.2	YAML 構成設定	15
3.3	環境構成設定	31
3.4	重要なルール	42
3.5	構成の生成と検証	46
第 4 章	PatroniRESTAPI	49
4.1	ヘルスチェックエンドポイント	49
4.2	モニタリングエンドポイント	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	64
4.8	エンドポイントを再初期化する	65
第 5 章	patronictl	67
5.1	構成	67
5.2	使用法	68
第 6 章	レプリカ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	ビルディングレプリカ	109
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 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ファクター	115
7.5	同期モードの実装	115
第 8 章	スタンバイクラスター	117
第 9 章	ウォッチドッグのサポート	119
9.1	Linux でのソフトウェアウォッチドッグの設定	120
第 10 章	クラスターの一時停止/再開モード	121
10.1	ゴール	121
10.2	実装	121
10.3	ユーザーガイド	122
第 11 章	DCS フェイルセーフモード	123
11.1	問題	123
11.2	現在の実装の理由	123
11.3	DCS フェイルセーフモード	123
11.4	低レベル実装の詳細	124
11.5	よくある質問	124
第 12 章	Kubernetes での Patroni の使用	127
12.1	エンドポイントを使用する	127
12.2	ConfigMap を使用する	127
12.3	構成	127
12.4	例	129
第 13 章	Citus のサポート	131
13.1	要約	131
13.2	patroni	132
13.3	Citus ワーカーのスイッチオーバー	133
13.4	DCS のピーク	135
13.5	Kubernetes の Citus	135
13.6	Citus アップグレードと PostgreSQL メジャーアップグレード	138
第 14 章	スタンドアロンを Patroni クラスターに変換する	139
14.1	プロシージャ	139
第 15 章	PostgreSQL バージョンのメジャーアップグレード	143
15.1	よくある質問	143
第 16 章	他のツールとの統合	145
16.1	Barman	145
第 17 章	セキュリティに関する考慮事項	147

17.1	DCS の保護	147
17.2	RESTAPI の保護	147
第 18 章	HA マルチデータセンター	149
18.1	同期レプリケーション	149
18.2	非同期レプリケーション	150
第 19 章	よくある質問	151
19.1	他の HA ソリューションとの比較	151
19.2	DCS	152
19.3	patroni	153
19.4	構成	153
19.5	Postgres 管理	156
19.6	概念と要件	157
19.7	自動フェイルオーバー	158
19.8	ブートストラップとスタンバイの作成	159
19.9	モニタリング	159
第 20 章	リリースノート	161
20.1	バージョン 3.3.0	161
20.2	バージョン 3.2.2	162
20.3	バージョン 3.2.1	164
20.4	バージョン 3.2.0	165
20.5	バージョン 3.1.2	166
20.6	バージョン 3.1.1	167
20.7	バージョン 3.1.0	168
20.8	バージョン 3.0.4	170
20.9	バージョン 3.0.3	171
20.10	バージョン 3.0.2	172
20.11	バージョン 3.0.1	173
20.12	バージョン 3.0.0	174
20.13	バージョン 2.1.7	175
20.14	バージョン 2.1.6	175
20.15	バージョン 2.1.5	176
20.16	バージョン 2.1.4	178
20.17	バージョン 2.1.3	180
20.18	バージョン 2.1.2	181
20.19	バージョン 2.1.1	184
20.20	バージョン 2.1.0	184
20.21	バージョン 2.0.2	187
20.22	バージョン 2.0.1	189
20.23	バージョン 2.0.0	190
20.24	バージョン 1.6.5	195
20.25	バージョン 1.6.4	198
20.26	バージョン 1.6.3	199
20.27	バージョン 1.6.2	199
20.28	バージョン 1.6.1	201
20.29	バージョン 1.6.0	204

20.30 バージョン 1.5.6	206
20.31 バージョン 1.5.5	207
20.32 バージョン 1.5.4	208
20.33 バージョン 1.5.3	210
20.34 バージョン 1.5.2	210
20.35 バージョン 1.5.1	211
20.36 バージョン 1.5.0	211
20.37 バージョン 1.4.6	213
20.38 バージョン 1.4.5	213
20.39 バージョン 1.4.4	215
20.40 バージョン 1.4.3	216
20.41 バージョン 1.4.2	217
20.42 バージョン 1.4.1	218
20.43 バージョン 1.4	218
20.44 バージョン 1.3.6	220
20.45 バージョン 1.3.5	221
20.46 バージョン 1.3.4	222
20.47 バージョン 1.3.3	223
20.48 バージョン 1.3.2	223
20.49 バージョン 1.3.1	223
20.50 バージョン 1.3	223
20.51 バージョン 1.2	226
20.52 バージョン 1.1	228
20.53 バージョン 1.0	231
20.54 バージョン 0.90	233
20.55 バージョン 0.80	235
第 21 章 貢献する	237
21.1 貢献するガイドライン	237
第 22 章 Indices and tables	241

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、フリーサイズ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、/"template/"と呼ばれています。独自の注意事項があります。賢く使用してください。PostgreSQL で高可用性を実行するには、多くの方法があります。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。

.0 から、Patroni は Postgres の **Note to Citus users** _データベース拡張機能とうまく統合しました。Citus 分散クラスターと Patroni の高可用性を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** 章を見てください。

第1章 はじめに

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

Patroni を使用した Docker ベースの展開の例については、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 Josh Berkus 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](#) をレポートします。

1.2 技術要件/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き、したがって、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するのは全く問題ありません。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> からクローン化されたものであることを前提としています。つまり、サンプル構成ファイル `postgres0.yml` および `postgres1.yml` が必要です。pip で Patroni をインストールした場合、git リポジトリからこれらのファイルを取得し、以下の `./patroni.py` を `patroni` コマンドに置き換えることができます。

開始するには、さまざまな端末から次のことを行います。:

```
> 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と表示されます。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 kill 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` ファイルを追加して、さらに大規模なクラスターを作成します。

Patroni は、[HAProxy](#) 構成を提供し、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アプリケーションに提供します。構成するには、次を実行します。

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

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

1.5 YAML 構成

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

1.6 環境設定

環境変数を介した設定の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

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の特定のバイト数を超えて後ろにある場合、フェイルオーバーが発生しないことが保証されます。この設定は、ビジネス要件に基づいて増加または減少する必要があります。同期レプリケーションを使用して、耐久性を向上させることもできます。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しないでください

アプリケーションから接続する場合、常に非スーパーユーザーを使用します。Patroni が正常に機能するには、データベースへのアクセスが必要です。アプリケーションからスーパーユーザーを使用することにより、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 `superuser_reserved_connections` 設定で利用できる場合があります。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、動作は望ましくありません。

1.9 HA ソリューションのテスト

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

そうは言っても、ここに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] 自分自身
- ディスク IO
- ファイル制限 Linux では `nofile`
- RAM。oomkiller をオフにしている場合でも、RAM が利用できないことにより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高い
- postgres プロセスの `kill -9 postmaster` を除きます。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、そうすることは実際のシナリオを模倣したものではないためです。インフラストラクチャが安全でなく、攻撃者が `kill -9` を実行する可能性が懸念されている場合、いくら HA プロセスを行ってもそれは修正されません。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、次を実行します。

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

2.2 Psycopg

psycopg2-2.8`*から、*psycopg2* のバイナリバージョン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ができないため、インストール方法を決定する必要があります。

使用可能なオプションがいくつかあります。

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します

```
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、*list of dependencies* で *`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]
```

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

Exhibitor

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

kubernetes

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

raft

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

aws

`boto3` AWS コールバックを使用するため

jsonlogger

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

all

上記のすべて *psycpg* ファミリーを除く

psycpg

`psycpg[binary]>=3.0.0` モジュール

psycpg2

`psycpg2>=2.5.4` モジュール

psycpg2-binary

`psycpg2-binary` モジュール

たとえば、*psycpg3*、DCS としての *Etcd* の依存関係、および AWS コールバックとともに Patroni を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[psycpg3,etcd3,aws]
```

レプリカの作成またはカスタムブートストラップスクリプトつまり WAL-E を呼び出す外部ツールは、パトロニとは無関係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 でのパッケージのインストール

Patroni パッケージは、次の Postgres コミュニティが作成した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。
- SUSEEnterpriseLinux。

公式オペレーティングシステムリポジトリでは使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を見つけることもできます。

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、EPEL からのパッケージが必要になる場合があります。`EPEL リポジトリ`_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メンテナンスされています。サポートが必要な場合は、最初に `Postgresstable`_ 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

PGDG リポジトリがインストールされたら、[above](#) を参照し、aptrun を介して 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

PGDG リポジトリがインストールされたら、[above](#) を参照して、RHEL9 および派生製品に dnf を介して etcdDCS で patroni をインストールします。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では、次のコマンドを実行します。

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

必要に応じて、リポジトリで 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。リポ名は、RockyLinux、AlmaLinux、OracleLinux などでもまだ `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。

PGDG リポジトリがインストールされた SLES15 の場合、[above](#) を参照してください。次を使用して 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

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します。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ー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、パトロニノードの 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 のモニタリングを再開して、メンテナンスモードを解除します。

```
patronictl resume --wait
```

クラスターは、Patroni の新しいバージョンで完全に動作します。

第3章 パトローニ構成

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、`patronictledit-config` ツールまたは Patroni `patronictledit-config` を使用できます。

- **loop/_wait** ループがスリープする秒数。デフォルト値 10、最小可能値 1
- **ttl** リーダーロックを取得する 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、最小可能値 20
- **retry/_timeout** DCS および PostgreSQL オペレーション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は、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、最小可能値 3

警告: `loop_wait`、`loop_wait`、または `loop_wait` の値を変更するときは、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選挙に参加できるまでに遅延す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、正常な非同期フォロワーにスワップされるまでに遅延する可能性のある最大バイト数。パトローニは、複数のフォロワーがいる場合に最大レプリカ lsn を利用します。それ以外の場合、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 は同期の不健康なフォロワーをスワップするアクションを実行しません。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、十分な高い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 に完全な履歴が保持されます。
- **primary/_start/_timeout** フェールオーバーがトリガーされる前に、プライマリが障害から回復できる時間秒単位。デフォルトは 300 秒です。0 に設定すると、フェールオーバーは可能であればクラッシュが検出された直後に実行されます。非同期レプリケーションを使用する場合、フェールオーバーによりトランザクションが失われる場合があります。プライマリ障害のワーストケースのフェールオーバー時間は次のとおりです。`primary/_start/_timeout` がゼロの場合を除き、`loop/_wait+primary/_start/_timeout+loop/_wait`。その場合、それは単なる `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` が有効になっている場合にのみ有効です。>0 に設定され、`synchronous_mode` が有効になっている場合、`primary/_stop/_timeout` で設定された値を超えて `stop` 操作が実行されている場合、Patroni は SIGKILL をポストマスターに送信します。耐久性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 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選挙に参加できます。同期モードでは、Patroni がトランザクションの耐久性を保証できない場合、書き込みの可用性を失いますが、正常にコミットされたトランザクションがフェールオーバー時に失われません。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の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 `false`。
- **postgresql**
 - **use/_pg/_rewind** `pg_rewind` を使用するかどうか。デフォルト `false`。クラスターは `data page checksums initdb` の `--data-checksums` オプションで初期化する必要があるか、`wal_log_hints` を `on` に設定する必要があることに注意してください。そうしないと、`pg_rewind` は機能しないことに注意してください。
 - **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+では、デフォルト `true` に設定されます。
 - **recovery/_conf** フォロワーの構成時に `recovery.conf` に書き込まれる追加の構成設定。PostgreSQL12 には `recovery.conf` はありませんが、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** `pg_hba.conf` を生成するために Patroni が使用する行のリスト。 `hba_file` PostgreSQL パラメーターがデフォルト以外の値に設定されている場合、Patroni はこのパラメーターを無視します。
 - * **-hostall0.0.0.0/0md5**
 - * **-hostreplicationpromoter127.0.0.1/32md5** レプリケーションには、このような行が必要です。
 - **pg/_ident** `pg_ident.conf` を生成するために Patroni が使用する行のリスト。 `ident_file` PostgreSQL パラメーターがデフォルト以外の値に設定されている場合、Patroni はこのパラメーターを無視します。
 - * **-mapname1systemname1pguser1**
 - * **-mapname1systemname2pguser2**
- **standby/_cluster** このセクションが定義されている場合、スタンバイクラスターをブートストラップします。
 - **host** リモートノードのアドレス

- **port** リモートノードのポート
 - **primary/_slot/_name** リモートノードのどのスロットをレプリケーションに使用するか。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。ファンクション ``slot_name_from_member_name`` を参照してください。
 - **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方法の順序付けたリスト。 **create/_replica/_methods** で定義されたリストとは異なる場合があります
 - **restore/_command** リモートプライマリからスタンバイクラスター内のノードに WAL レコードを復元するコマンド。 **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 cleanup コマンド
 - **recovery/_min/_apply/_delay** スタンバイリーダーに WAL レコードを実際に適用するまでの待機時間
- **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、すべてのノードで永続的な物理スロットが作成され、その位置は **slots** 秒ごとに進められます。11 より古い PostgreSQL バージョンの場合、11 の永久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、再起動とともにプライマリからスタンバイにコピーされ、その後 **slots** 秒ごとに位置が進みました必要に応じて。論理スロットファイルのコピーは、**libpq** 接続を介し、リワインドまたはスーパーユーザー資格情報を使用して実行されます **slots** セクションを参照してください。レプリカの論理スロット位置は以前のプライマリより少し後である可能性が常にあります。したがって、アプリケーションは、フェールオーバー後に 2 回目に一部のメッセージを受信できるように準備する必要があります。そのための最も簡単な方法-トラッキング **confirmed_flush_lsn**。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。定義された永続的な論理レプリケーションスロットがある場合、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ールオーバーは安全でなく、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** 永続的レプリケーションスロットの名前。永続的なスロット名が現在のノードの名前と一致する場合、このノードでは作成されません。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ことを保証します。通常は Patroni によってスロットが削除されます。これは、一時的な障害中にメンバーが使用するレプリケーションスロットを持続させたい場合、または既存のメンバーを新しいパトロニクラスター **my/_slot/_name** を参照してください。詳細については、オペレーターがこれらの Patroni の通常の機能に影響を与えるため、スロットが必要でなくなった場合、名前の衝突は DCS に持続されません。
 - * **type** スロットタイプ。 **physical** または **logical** になります。スロットが論理的な場合、**database** および **plugin** を追加して定義する必要があります。
 - * **database** 論理スロットを作成するデータベース名。
 - * **plugin** 論理スロットのプラグイン名。
- **ignore/_slots** Patroni が一致するスロットを無視する必要があるレプリケーションスロットプロパティ

のセットのリスト。この構成/機能/など一部のレプリケーションスロットが Patroni の外部で管理されている場合に役立ちます。一致するプロパティのサブセットにより、スロットが無視されます。

- **name** レプリケーションスロットの名前。
- **type** スロットタイプ。 **physical** または **logical** です。スロットが論理的な場合、さらに **database** および/または **plugin** を定義できます。
- **database** データベース名 **logical** スロットと一致する場合。
- **plugin** 論理デコードプラグイン **logical** スロットと一致する場合。

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

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

注クラスタポロジが静的名前が変更されないノードの固定数の場合、ノードの名前に対応する名前で永続的な物理レプリケーションスロットを構成して、レプリカが一時的にダウンしているときに WAL ファイルのリサイクルを回避できます。

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

警告: 永久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使用することになっています。レプリカノードでそれらを使用すると、クラスター内の他のすべてのノードで **pg_wal** の無制限の増加が発生します。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を構成する場

合です。これらはすべてのノード間で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。

警告: スタンバイで `nostream` タグを設定すると、ノード自分自身とそのすべてのカスケードレプリカある場合は、永久論理レプリケーションスロットのコピーと同期が無効になります。

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

3.2.2 ログ

- **type** ログの形式を設定します。 **type** または **type** のいずれかです。 **type** 形式を使用するには、 **type** をインストールする必要があります。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** を参照してください
- **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する場合は、 **traceback/_level** に設定します。
- **format** ログフォーマット文字列を設定します。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なります。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、`%( 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印刷する場合は、辞書キーがログフィールドであり、値がログに印刷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** を参照してください。
- **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 段階ロギングを使用しています。ログレコードはインメモリキューに書き込まれ、キューから取得し、`stderr` またはファイルに書き込む別のスレッドがあります。内部キューの最大サイズは、デフォルトで **max/_queue/_size** レコードによって制限されます。これは、過去 1 時間 20 分ログを保持するのに十分です。

- **dir** アプリケーションログを書き込むディレクトリ。ディレクトリは存在し、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。`file\_num` および `file\_size` を使用してこれらのリテンション値を調整できます。以下を参照してください。
- **file/_num** 保持するアプリケーションログの数。
- **file/_size** ログ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ギングレベルを再定義できます
 - **patroni.postmaster:** WARNING
 - **urllib3:** DEBUG

次に、json 形式でログを記録するようにパトロニを構成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` セクションへの今後の変更はすべて有効になりません。それらを変更する場合は、`bootstrap.dcs` または Patroni `bootstrap.dcs` のいずれかを使用してください。

- **bootstrap**
 - **dcs** このセクションは、新しいクラスターを初期化した後、指定された構成ストアの ``<namespace>/<scope>/config`` に書き込まれます。クラスターのグローバル動的構成。 **dcs** で説明されているパラメーターを `bootstrap.dcs` の下に配置できます。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` が指定された場合、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` パラメーターが存在しない場合にもトリガーされます。
 - **initdb** : (オプション) `initdb` に渡されるオプションのリスト。

- * **-data-checksums** 9.3 で `pg_rewind` が必要な場合、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** クラスタを初期化した後に実行される追加スクリプト。スクリプトは、接続文字列 URL をユーザー名としてクラスタースーパーユーザーを使用して受け取ります。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、1つのデータベースのみがサポートされています。

3.2.5 Consul

パラメーターのほとんど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** コンスルローカルエージェントの URL、形式は `http(s)://host:port` です。
- **port** オプションの Consul ポート。
- **scheme** オプション **scheme** または **scheme**、デフォルトは **scheme**。
- **token** オプションの ACL トークン。
- **verify** オプション HTTPS 要求の 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** クライアント証明書を含むオプションファイル。
- **key** クライアントキーを含むオプションのファイル。キーが **key** の一部である場合、空にすることができます。
- **dc** オプション通信するデータセンター。デフォルトでは、ホストのデータセンターが使用されます。
- **consistency** : (オプション) consul 一貫性モードを選択します。使用可能な値は **default**、**consistent**、または **stale** です **consistency _**の詳細
- **checks** セッションに使用される Consul ヘルスチェックのオプションリスト。デフォルトでは、空のリストが使用されます。

- **register/_service** オプションスコープパラメーターで定義された名前と、ノードのロールに応じてタグマスター、プライマリ、レプリカ、またはスタンバイリーダーでサービスを登録するかどうか。デフォルトの **register/_service**。
- **service/_tags** 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

パラメーターの大部分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** 形式の etcd エンドポイントのリスト **host1:port1,host2:port2,etc...** コンマ区切りの文字列または実際の yaml リストです。
- **use/_proxies** このパラメーターが **true** に設定されている場合、Patroni は **use/_proxies** をプロキシのリストと見なし、etcd クラスターのトポロジ検出を実行しません。
- **url** etcd の url。
- **proxy** etcd のプロキシ URL。プロキシを使用して etcd に接続する場合は、**proxy** の代わりにこのパラメーターを使用します。
- **srv** クラスター自動検出の SRV レコードを検索するドメイン。Patroni は、指定されたドメインのこれらの SRV サービス名を照会しようとし、最初に成功するまでこの順序で **_etcd-client-ssl**、**_etcd-client**、**_etcd-ssl**、**_etcd**、**_etcd-server-ssl**、**_etcd-server**。**_etcd-server-ssl** または **_etcd-server** の SRV レコードが取得された場合、ETCD ピアプロトコルが使用され、使用可能なメンバーの ETCD を照会します。それ以外の場合、SRV レコードのホストが使用されます。
- **srv/_suffix** 検出中に照会される SRV 名前のサフィックスを構成します。このフラグを使用して、同じドメインの下の複数の etcd クラスターを区別します。**srv/_suffix** と組み合わせてのみ動作します。

たとえば、`srv_suffix: foo` および `srv: example.org` が設定されている場合、次の DNSSRV クエリーが作成されます。 `_etcd-client-ssl-foo._tcp.example.com` および考えられるすべての ETCDsrv サービス名など。

- **protocol** オプション `http` または `https`、指定しない場合、`http` が使用されます。 **protocol** または **protocol** が指定されている場合、-はそれらからプロトコルを取得します。
- **username** オプションの etcd 認証のユーザー名。
- **password** etcd 認証のオプションパスワード。
- **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** クライアント証明書を含むオプションファイル。
- **key** クライアントキーを含むオプションのファイル。キーが **key** の一部である場合、空にすることができます。

3.2.7 etcdv3

Patroni がプロトコルバージョン 3 を介して Etcd クラスターで動作するようにしたい場合は、Patroni 構成ファイルの `etcd3` セクションを使用する必要があります。すべての構成パラメーターは `etcd` 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 では可視されず、その逆であるため、Patroni 構成ファイルを更新するだけで `etcd` から `etcd3` に切り替えることはできません。さらに、Patroni は Etcd の gRPC-gateway プロキシを使用して V3API と通信します。これは、TLS 共通名認証が不可能であることを意味します。

3.2.8 ZooKeeper

- **hosts** 形式 `['host1:port1','host2:port2','etc...']` の ZooKeeper クラスターメンバーのリスト。
- **use_ssl** オプション SSL を使用するかどうか。デフォルト `false`。 `false` に設定されている場合、SSL 固有のパラメーターはすべて無視されます。
- **cacert** : (オプション) CA 証明書。存在する場合、検証が有効になります。
- **cert** オプションクライアント証明書を含むファイル。
- **key** オプションクライアントキーを含むファイル。
- **key_password** オプションクライアントキーのパスワード。
- **verify** オプション証明書を検証するかどうか。デフォルト `true`。
- **set_acls** オプション設定されている場合、作成する各 Z ノードにデフォルトの 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、キーとしてプリンシパルを使用し、値のリストとして 1 つ以上の権限を使用して辞書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 のいずれかです。例 `set_acls: {CN=principal1: [CREATE, READ], CN=principal2: [ALL]}`。

- **auth_data** オプショナル接続に使用する認証資格情報。`scheme` がキーであり、`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 Exhibitor

- **hosts** 'host1,host2,etc...' 形式の ExhibitorZooKeeper ノードの初期リスト。このリストは、Exhibitor-ZooKeeper クラスタポートログが変更されるたびに、自動的に更新されます。
- **poll/_interval** ZooKeeper および attempts ノードのリストを Exhibitor 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** オプショナル KubernetesAPI と通信する場合、Patroni は通常 `kubernetes` サービスに依存しています。そのアドレスは `KUBERNETES\_SERVICE\_HOST` 環境変数を介してポッドで公開されます。`bypass_api_service` が `true` に設定されている場合、Patroni はサービスの背後にある API ノードのリストを解決し、それらに直接接続します。
- **namespace** オプショナル Patroni ポッドが実行されている Kubernetes 名前空間。デフォルト値は `default` です。
- **labels {label1: value1, label2: value2}** フォーマットのラベル。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構成マップを見つけるために使用されます。また、Patroni は、作成するすべてのオブジェクトエンドポイントまたは ConfigMap にそれらを設定します。
- **scope/_label** オプショナルクラスター名を含むラベルの名前。デフォルト値は `cluster-name` です。
- **role/_label** 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、実行されるポッドにこのラベルを設定します。デフォルト値は `role` です。
- **leader/_label/_value** Postgres ロールが `master` の場合のポッドラベルのオプショナル値。デフォルト値は `master` です。
- **follower/_label/_value** Postgres ロールが `replica` の場合のポッドラベルのオプショナル値。デフォルト値は `replica` です。
- **standby/_leader/_label/_value** Postgres ロールが `standby_leader` の場合のポッドラベルのオプショナル値。デフォルト値は `master` です。
- **tmp/_role/_label** オプションのロールマスターまたはレプリカを含む一時ラベルの名前。このラベルの値は、常に対応するロールのデフォルトを使用します。必要な場合にのみ設定します。

- **use_endpoints** オプション。true に設定されている場合、Patroni は ConfigMap の代わりにエンドポイントを使用してリーダー選挙を実行し、クラスター状態を維持します。
- **pod_ip** オプション。パトローニが実行されているポッドの IP アドレス。この値は、`use_endpoints` が有効な場合に必要で、ポッドの PostgreSQL が昇格するときにリーダーエンドポイントのサブセットを設定するために使用されます。
- **ports** オプション。サービスオブジェクトにポートの名前がある場合、同じ名前が Endpoint オブジェクトに表示される必要があります。それ以外の場合、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。Patroni はそれを使用してリーダーエンドポイントのサブセットを更新します。このパラメータは、`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。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 ファイルを指定します。提供されない場合、patroni は ServiceAccount シークレットによって提供される値を使用します。
- **retriable_http_codes** 再試行する K8sAPI からの HTTP ステータスコードのオプションリスト。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、`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 `['ip1:port','ip2:port','etc...']`
- **data_dir** Raft ログとスナップショットを保存するディレクトリ。指定しない場合、現在の作業ディレクトリが使用されます。
- **password** オプション。指定されたパスワードで Raft トラフィックを暗号化します。`cryptography` Python モジュールが必要です。

Raft 実装に関する短い 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方法はありますか?

A `syncobj_admin -conn host:port -status` ここで、`host:port` は、クラスターノードのいずれかのアドレスです

- Q コンセンサスの一部分であったノードはなくなったので、他のノードに同じ IP を再利用できません。このノードをコンセンサスから削除するにはどうすればよいですか?

A `syncobj_admin -conn host:port -remove host2:port2 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: これは、Patroni 依存関係である pysyncobj モジュール pythonRAFT 実装とともにインストールされます。

- Q コンセンサスに追加せずに Patroni ノードを実行することは可能ですか?

A: はい、Patroni 構成から raft.self_addr をコメントアウトするか、削除するだけです。

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

A: はい、3 番目のノードで patroni_raft_controller Patroni および PostgreSQL なしで実行できます。このようなセットアップでは、プライマリに影響を与えることなく 1 つのノードを一時的に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 Patroni が postgres に接続するために使用されます。
- **password** スーパーユーザーのパスワード、初期化 initdb 中に設定します。
- **sslmode** オプション **sslmode** _接続パラメーターにマップします。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** _にアクセスしてください。デフォルトモードは **prefer** です。
- **sslkey** オプション **sslkey** _接続パラメーターにマップします。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** _接続パラメーターにマップします。これは、**sslkey** で指定された秘密キーのパスワードを指定します。
- **sslcert** オプション **sslcert** _接続パラメーターにマップします。これは、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** _接続パラメーターにマップします。これは、クライアントがサーバーの証明書を検証するために使用する 1 つ以上の認証局 CA 証明書を含むファイルの場所を指定します。
- **sslcrl** オプション **sslcrl** _接続パラメーターにマップします。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** _接続パラメーターにマップします。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** _接続パラメーターにマップします。これは、安全な GSSTCP/IP 接続をサーバーとネゴシエートするかどうか、またはその優先順位を決定します
- ・ **channel_binding** オプション **channel_binding** _接続パラメーターにマップします。これは、クライアントのチャンネルバインディングの使用を制御します。

* replication

- ・ **username** レプリケーションユーザー名。ユーザーは初期化中に作成されます。レプリカはこのユーザーを使用して、ストリーミングレプリケーションを介してレプリケーションソースにアクセスします
- ・ **password** レプリケーションパスワード。ユーザーは初期化中に作成されます。
- ・ **sslmode** オプション **sslmode** _接続パラメーターにマップします。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** _にアクセスしてください。デフォルトモードは **prefer** です。
- ・ **sslkey** オプション **sslkey** _接続パラメーターにマップします。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** _接続パラメーターにマップします。これは、**sslkey** で指定された秘密キーのパスワードを指定します。
- ・ **sslcert** オプション **sslcert** _接続パラメーターにマップします。これは、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** _接続パラメーターにマップします。これは、クライアントがサーバーの証明書を検証するために使用する 1 つ以上の認証局 CA 証明書を含むファイルの場所を指定します。
- ・ **sslcrl** オプション **sslcrl** _接続パラメーターにマップします。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** _接続パラメーターにマップします。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** _接続パラメーターにマップします。これは、安全な GSSTCP/IP 接続をサーバーとネゴシエートするかどうか、またはその優先順位を決定します
- ・ **channel_binding** オプション **channel_binding** _接続パラメーターにマップします。これは、クライアントのチャンネルバインディングの使用を制御します。

* rewind

- ・ **username** **pg_rewind** のユーザーのオプションの名前。ユーザーは postgres11+ の初期化中に作成され、必要なすべての **username** _が付与されます。
- ・ **password** **pg_rewind** のユーザーのオプションパスワード。ユーザーは初期化中に作成されます。

- ・ **sslmode** オプション **sslmode** 接続パラメーターにマップします。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** にアクセスしてください。デフォルトモードは **prefer** です。
- ・ **sslkey** オプション **sslkey** 接続パラメーターにマップします。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** 接続パラメーターにマップします。これは、**sslkey** で指定された秘密キーのパスワードを指定します。
- ・ **sslcert** オプション **sslcert** 接続パラメーターにマップします。これは、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** 接続パラメーターにマップします。これは、クライアントがサーバーの証明書を検証するために使用する 1 つ以上の認証局 CA 証明書を含むファイルの場所を指定します。
- ・ **sslcrl** オプション **sslcrl** 接続パラメーターにマップします。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** 接続パラメーターにマップします。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** 接続パラメーターにマップします。これは、安全な GSSTCP/IP 接続をサーバーとネゴシエートするかどうか、またはその優先順位を決定します
- ・ **channel_binding** オプション **channel_binding** 接続パラメーターにマップします。これは、クライアントの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。それらの書き方の例として、`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ているときにこのスクリプトを実行します。
 - * **on/_start** postgres の起動時にこのスクリプトを実行します。
 - * **on/_stop** postgres が停止したときにこのスクリプトを実行します。
- **connect/_address** 他のノードおよび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 の横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 **proxy_url** として DCS のメンバーキーに書き込まれ、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更するための **create** メソッドの順序付けリスト。"/basebackup/" はデフォルトの方法です。他のメソッドは、それぞれが独

自の構成項目として構成されるスクリプトを参照するものとします。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 **pg_ctl**, **initdb**, **pg_controldata**, **pg_basebackup**, **postgres**, **pg_isready**, **pg_rewind** へのパス。提供されない場合、または空の文字列の場合、PATH 環境変数を使用して実行可能ファイルを見つけます。
- **bin/_name** オプションカスタム Postgres ディストリビューションを使用している場合、Postgres バイナリ名をオーバーライドできるようにします。
 - * **pg_ctl** オプション **pg_ctl** バイナリのカスタム名。
 - * **initdb** オプション **initdb** バイナリのカスタム名。
 - * **pg/controldata** オプション **pg_controldata** バイナリのカスタム名。
 - * **pg/_basebackup** オプション **pg_basebackup** バイナリのカスタム名。
 - * **postgres** オプション **postgres** バイナリのカスタム名。
 - * **pg_isready** オプション **pg_isready** バイナリのカスタム名。
 - * **pg/_rewind** オプション **pg_rewind** バイナリのカスタム名。
- **listen** Postgres がリッスンする IP アドレス+ポート。ストリーミングレプリケーションを使用している場合、クラスター内の他のノードからアクセスできる必要があります。ポートコンポーネントがコロンで最後のものつまり **listen: 127.0.0.1,127.0.0.2:5432** の後に追加される限り、複数のコンマ区切りアドレスが許可されます。Patroni は、このリストの最初のアドレスを使用して、PostgreSQL ノードへのローカル接続を確立します。
- **use/_unix/_socket** は、Patroni がクラスターに接続するための UNIX ソケットの使用を優先することを指定します。デフォルト値は **false** です。 **unix_socket_directories** が定義されている場合、Patroni はそこから最初に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。 **postgresql.parameters** で **unix_socket_directories** が指定されていない場合、Patroni はデフォルト値を使用する必要があると想定し、接続パラメーターから **host** を省略します。
- **use/_unix/_socket/_repl** は、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 はそこから最初に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。 **postgresql.parameters** で **unix_socket_directories** が指定されていない場合、Patroni はデフォルト値を使用する必要があると想定し、接続パラメーターから **host** を省略します。
- **pgpass pgpass _**パスワードファイルへのパス。Patroni は、 **pg/_basebackup**, **post_init** スクリプト、およびその他の状況を実行する前にこのファイルを作成します。場所は Patroni によって書き込み可能である必要があります。
- **recovery/_conf** フォロワーの構成時に **recovery.conf** に書き込まれる追加の構成設定。

- **custom/_conf** postgresql.base.conf の代わりに使用されるオプションのカスタム postgresql.conf ファイルへのパス。ファイルはすべてのクラスターノードに存在し、PostgreSQL で読み取り可能であり、実際の postgresql.conf 上のその場所から含まれる必要があります。Patroni はこのファイルの変更を監視せず、バックアップもしないことに注意してください。ただし、その設定は、Patroni 独自の構成機能によってオーバーライドされる場合があります。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** pg_hba.conf を生成するために Patroni が使用する行のリスト。hba_file PostgreSQL パラメーターがデフォルト以外の値に設定されている場合、Patroni はこのパラメーターを無視します。pg/_hba と合わせて、このパラメーターは pg_hba.conf の管理を簡素化します。
 - * **-hostallall0.0.0.0/0md5**
 - * **-hostreplicationpromoter127.0.0.1/32md5** レプリケーションには、このような行が必要です。
- **pg/_ident** pg_ident.conf を生成するために Patroni が使用する行のリスト。ident_file PostgreSQL パラメーターがデフォルト以外の値に設定されている場合、Patroni はこのパラメーターを無視します。pg/_ident と合わせて、このパラメーターは pg_ident.conf の管理を簡素化します。
 - * **-mapname1systemname1pguser1**
 - * **-mapname1systemname2pguser2**
- **pg/_ctl/_timeout** start、stop または restart を実行するときに pg_ctl が待機する時間。デフォルト値は 60 秒です。
- **use/_pg/_rewind** レプリカとしてクラスターに参加するときに、元のリーダーで pg/_rewind を使用してみてください。クラスターを data page checksums initdb の --data-checksums オプションで初期化するか、wal_log_hints を on に設定する必要があります。そうしないと、pg_rewind は機能しません。
- **remove/_data/_directory/_on/_rewind/_failure** このオプションが有効になっている場合、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。それ以外の場合は、新しいリーダーに従おうとします。デフォルト値は **remove/_data/_directory/_on/_rewind/_failure** です。
- **remove/_data/_directory/_on/_diverged/_timelines** タイムラインが分岐し、元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、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。このオプションは、pg_rewind を使用できない場合に役立ちます。PostgreSQLv10 以前のタイムライン発散チェックを実行しているときに、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ています。したがって、このようなアクセスは pg_hba.conf で許可する必要があります。デフォルト値は **remove/_data/_directory/_on/_diverged/_timelines** です。
- **replica/_method** basebackup 以外の create_replica_methods ごとに、同じ名前の構成セクションを追加します。少なくとも、これには、実行する実際のスクリプトへのフルパスを含む/"command/"が含まれる必要があります。他の構成パラメーターは、/"parameter=value/"の形式でスクリプトに渡されます。

- **pre/_promote** フェイ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されるフェンシングスクリプト。スクリプトがゼロ以外のコードで終了した場合、パトロニはレプリカをプロモートせず、DCS からリーダーキーを削除します。
- **before/_stop postgres** を停止する直前に実行されるスクリプト。コールバックとは対照的に、このスクリプトは同期的に実行され、完了するまでシャットダウンをブロックします。このスクリプトのリターンコードは、その後にシャットダウンを続行するかどうかに影響を与えません。

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/"localhost/"またはループバックアドレスつまり/"localhost/"または/"127.0.0.1/"。HTTP ヘルスチェック/"listen/"RESTAPI パラメーターについて以下を読んでください、ユーザクエリ直接または RESTAPI を介して、およびによって実行されるヘルスチェックのエンドポイントとして機能できます。リーダー選挙中のクラスターメンバーたとえば、リーダーがまだ実行されているかどうか、またはクエリを実行するノードの前にある WAL 位置を持つノードがあるかどうかを判断するため。connect_address はメンバーキーに配置されます 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します。
- **listen** Patroni が RESTAPI をリッスンする IP アドレスまたはホスト名とポート-上記のように、参加ノード間で同じヘルスチェックとクラスターメッセージも提供します。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ーのヘルスチェック情報を提供します。
- **authentication** オプショナル
 - * **username** 安全でない RESTAPI エンドポイントを保護するための Basic-auth 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-auth パスワード。
- **certfile** : (オプショナル) 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** : (オプショナル) PEM 形式の秘密キーを含むファイルを指定します。
- **keyfile/_password** オプショナルキーファイルを復号するためのパスワードを指定します。
- **cafile** オプショナルクライアント証明書の検証中に使用する信頼できる CA の証明書を含む CA_BUNDLE を含むファイルを指定します。
- **ciphers** オプショナル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** オプショナル **none** デフォルト **optional** または **required**。 **none** RESTAPI がクライアント証明書をチェックしない場合。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** を使用する場合、証明書の署名検証が成功した場合、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 要求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 エンドポイントを呼び出すことができ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、含まれないものは拒否されます。
- **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します。
- **https/_extra/_headers** オプション HTTPS ヘッダーにより、TLS が有効になっているときに RESTAPI サーバーは HTTP 応答で追加情報を渡します。これは、 **http_extra_headers** で設定された追加情報も渡します。
- **request_queue_size** : (オプション) PatroniRESTAPI が使用する TCP ソケットの要求キューサイズを設定します。キューがいっぱいになると、それ以降の要求は `"Connectiondenied/"` エラーを取得します。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。内部的に Patroni は、リーダーレース中にそれを使用して、レプリケーションラグを最小限に抑えたノードを見つけます。

- クライアント証明書の検証を有効にしている場合 `restapi.verify_client` が `required` に設定されている場合、`restapi.verify_client` は `ctl.certfile`、`ctl.keyfile`、`ctl.keyfile_password` に `restapi.verify_client` を提供します。提供されない場合、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。提供されない場合、**username** は RESTAPI の `"/username/"` パラメーターで提供された値を使用します。
 - * **password** 保護された RESTAPI エンドポイントにアクセスするための基本認証パスワード。提供されない場合、**password** は RESTAPI の `"/password/"` パラメーターで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信頼できる CA の証明書が含まれる `CA_BUNDLE` ファイルまたはディレクトリを指定します。提供されない場合、**cacert** は RESTAPI の `"/cafile/"` パラメーターで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を含むファイルを指定します。
 - **keyfile** PEM 形式のクライアント秘密キーを含むファイルを指定します。
 - **keyfile/_password** クライアントキーファイルを復号するためのパスワードを指定します。

3.2.15 ウォッチドッグ

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

3.2.16 タグ

- **clonefrom** true または false。true に設定されている場合、他のノードはブートストラップにこのノードを使用することを優先する場合がありますから pg_basebackup を取得します。clonefrom タグが true に設定されているノードが複数ある場合、ブートストラップする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 に設定されている場合、ノードは GET /replica REST API ヘルスチェックの 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。デフォルトの false。
- **replicatefrom** レプリケートする別のレプリカの名前。カスケード複製をサポートするために使用されます。
- **nosync** true または false。true に設定されている場合、ノードは同期レプリカとして選択されません。
- **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されるかどうかを制御します。デフォルトは false、このノードがリーダーレースに `_can_` 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度を制御します。優先度の高いノードが同じ量の WAL を受信/再生した場合、優先度の低いノードよりも優先されます。ただし、優先度に関係なく、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、リーダーになることはできません nofailover: true と同様に。
- **nostream** true または false。true に設定されている場合、ノードはレプリケーションプロトコルを使用して WAL をストリーミングしません。代わりに、アーカイブリカバリ restore_command が構成されている場合および pg_wal / pg_xlog ポーリングに依存します。また、ノード自分自身とそのすべてのカスケードレプリカの永久論理レプリケーションスロットのコピーと同期も無効にします。プライマリノードでこのタグを設定しても効果はありません。

警告: nofailover または failover_priority のいずれかのみを提供します。nofailover: true の提供は failover_priority: 0 と同じであり、nofailover: false を提供するとノード優先度 1 が与えられます。

これらの事前定義されたタグに加えて、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、インスタンスの状態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、または各値が照会値と一致しない場合、HTTP ステータスコード 503 が返されます。

3.3 環境構成設定

システム環境変数を使用して、Patroni 構成ファイルで定義された構成パラメーターの一部をオーバーライドすることが可能です。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のリストを示しています。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` 環境変数を介してパトロニの構成全体を設定することが可能です。この場合、他の環境変数は考慮されません。
- **PATRONI/_NAME** Patroni の現在のインスタンスが実行されているノードの名前。クラスターに対して一意である必要があります。
- **PATRONI/_NAMESPACE** :Patroni がクラスターに関する情報を保持する構成ストア内のパス。デフォルト値 `"/service/"`
- **PATRONI/_SCOPE** クラスター名

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** のいずれかです。 **PATRONI/_LOG/_TYPE** 形式を使用するには、 **PATRONI/_LOG/_TYPE** をインストールする必要があります。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** を参照してください
- **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する場合は、 **PATRONI/_LOG/_TRACEBACK/_LEVEL** に設定します。
- **PATRONI/_LOG/_FORMAT** ログフォーマット文字列を設定します。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なります。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、 `%( 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印刷する場合は、辞書キーがログフィールドであり、値がログに印刷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** を参照してください。

- **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 段階ロギングを使用しています。ログレコードはインメモリキューに書き込まれ、キューから取得し、stderr またはファイルに書き込む別のスレッドがあります。内部キューの最大サイズは、デフォルトで **PATRONI/_LOG/_MAX/_QUEUE/_SIZE** レコードによって制限されます。これは、過去 1 時間 20 分ログを保持するのに十分です。
- **PATRONI/_LOG/_DIR** アプリケーションログを書き込むディレクトリ。ディレクトリは存在し、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。`PATRONI\_LOG\_FILE\_NUM` および `PATRONI\_LOG\_FILE\_SIZE` を使用してこれらのリテンション値を調整できます。以下を参照してください。
- **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ギン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、1 つのデータベースのみがサポートされています。

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、形式は `http(s)://host:port`
- **PATRONI/_CONSUL/_PORT** オプショナルの Consul ポート
- **PATRONI/_CONSUL/_SCHEME** オプショナル **PATRONI/_CONSUL/_SCHEME** または **PATRONI/_CONSUL/_SCHEME**、デフォルトは **PATRONI/_CONSUL/_SCHEME**
- **PATRONI/_CONSUL/_TOKEN** オプショナルの ACL トークン
- **PATRONI/_CONSUL/_VERIFY** オプショナル HTTPS 要求の 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ptionalクライアント証明書を含むファイル
- **PATRONI/_CONSUL/_KEY** オptional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、空にすることができます。
- **PATRONI/_CONSUL/_DC** オptional通信するデータセンター。デフォルトでは、ホストのデータセンターが使用されます。
- **PATRONI/_CONSUL/_CONSISTENCY** : (オプション) consul 一貫性モードを選択します。使用可能な値は default、consistent、または stale です **PATRONI/_CONSUL/_CONSISTENCY_** の詳細
- **PATRONI/_CONSUL/_CHECKS** セッションに使用される Consul ヘルスチェックのオプションリスト。デフォルトでは、空のリストが使用されます。
- **PATRONI/_CONSUL/_REGISTER/_SERVICE** オptionalスコープパラメーターで定義された名前と、ノードのロールに応じてタグマスター、プライマリ、レプリカ、またはスタンバイリーダーでサービスを登録するかどうか。デフォルト **PATRONI/_CONSUL/_REGISTER/_SERVICE**
- **PATRONI/_CONSUL/_SERVICE/_TAGS** 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ptional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ptional 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 etcd

- **PATRONI/_ETCD/_PROXY** etcd のプロキシ URL。プロキシを使用して etcd に接続する場合、 **PATRONI/_ETCD/_PROXY** の代わりにこのパラメーターを使用します
- **PATRONI/_ETCD/_URL** etcd の url、形式は 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見なし、etcd クラスターのトポロジ検出を実行しませんが、 **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、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、-は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 SRV レコードを検索するドメイン。Patroni は、指定されたドメインのこれらの SRV サービス名を照会しようとしまず最初に成功するまでこの順序で

`_etcd-client-ssl`、`_etcd-client`、`_etcd-ssl`、`_etcd`、`_etcd-server-ssl`、`_etcd-server`。
`_etcd-server-ssl` または `_etcd-server` の SRV レコードが取得された場合、ETCD ピアプロトコルが使用され、使用可能なメンバーの ETCD を照会します。それ以外の場合、SRV レコードのホストが使用されます。

- **PATRONI/_ETCD/_SRV/_SUFFIX** 検出中に照会される SRV 名前のサフィックスを構成します。このフラグを使用して、同じドメインの下複数の `etcd` クラスターを区別します。 **PATRONI/_ETCD/_SRV/_SUFFIX** と組み合わせてのみ動作します。たとえば、`PATRONI_ETCD_SRV_SUFFIX=foo` および `PATRONI_ETCD_SRV=example.org` が設定されている場合、次の DNSSRV クエリーが作成されます。 `_etcd-client-ssl-foo._tcp.example.com` および考えられるすべての ETCDSRV サービス名など。
- **PATRONI/_ETCD/_USERNAME** etcd 認証のユーザー名。
- **PATRONI/_ETCD/_PASSWORD** etcd 認証のパスワード。
- **PATRONI/_ETCD/_CACERT** ca 証明書。存在する場合、検証が有効になります。
- **PATRONI/_ETCD/_CERT** クライアント証明書を含むファイル。
- **PATRONI/_ETCD/_KEY** クライアントキーを含むファイル。キーが証明書の一部である場合、空にすることができます。

3.3.6 etcdv3

Etcdv3 の環境名は Etcd の場合と似ていますが、変数名で 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 では可視されず、その逆もあるため、Patroni 構成を更新するだけで Etcd から Etcdv3 に切り替えることはできません。さらに、Patroni は Etcd の gRPC-gateway プロキシを使用して V3API と通信します。これは、TLS 共通名認証が不可能であることを意味します。

3.3.7 ZooKeeper

- **PATRONI/_ZOOKEEPER/_HOSTS** ZooKeeper クラスターメンバーのコンマ区切りリスト `/"host1:port1','host2:port2','etc.../'`。すべてのエンティティを引用することが重要です。
- **PATRONI/_ZOOKEEPER/_USE/_SSL** オプショナル SSL を使用するかどうか。デフォルト `false`。`false` に設定されている場合、SSL 固有のパラメーターはすべて無視されます。
- **PATRONI/_ZOOKEEPER/_CACERT** : (オプショナル) CA 証明書。存在する場合、検証が有効になります。
- **PATRONI/_ZOOKEEPER/_CERT** オプショナルクライアント証明書を含むファイル。
- **PATRONI/_ZOOKEEPER/_KEY** オプショナルクライアントキーを含むファイル。
- **PATRONI/_ZOOKEEPER/_KEY/_PASSWORD** オプショナル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 `true`。
- **PATRONI/_ZOOKEEPER/_SET/_ACLS** オプション設定されている場合、作成する各 Z ノードにデフォルトの 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、キーとしてプリンシパルを使用し、値のリストとして 1 つ以上の権限を使用して辞書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 のいずれかです。例 `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` がキーであり、`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 Exhibitor

- **PATRONI/_EXHIBITOR/_HOSTS** 'host1,host2,etc...' 形式の ExhibitorZooKeeper ノードの初期リスト。このリストは、ExhibitorZooKeeper クラスタポート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` サービスに依存しています。そのアドレスは `KUBERNETES\_SERVICE\_HOST` 環境変数を介してポッドで公開されます。`PATRONI_KUBERNETES_BYPASS_API_SERVICE` が `true` に設定されている場合、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** オプションパトロナポッドが実行されている Kubernetes 名前空間。デフォルト値は `default` です。
- **PATRONI/_KUBERNETES/_LABELS** {label1: value1, label2: value2} フォーマットのラベル。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構成マップを見つけるために使用されます。また、Patroni は、作成するすべてのオブジェクト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 `cluster-name` です。
- **PATRONI/_KUBERNETES/_ROLE/_LABEL** 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、実行されるポッドにこのラベルを設定します。デフォルト値は `role` です。
- **PATRONI/_KUBERNETES/_LEADER/_LABEL/_VALUE** Postgres ロールが `master` の場合のポッドラベルのオプション値。デフォルト値は `master` です。
- **PATRONI/_KUBERNETES/_FOLLOWER/_LABEL/_VALUE** Postgres ロールが `replica` の場合のポッドラベルのオプション値。デフォルト値は `replica` です。

- **PATRONI/_KUBERNETES/_STANDBY/_LEADER/_LABEL/_VALUE** Postgres ロールが standby_leader の場合のポッドラベルのオプション値。デフォルト値は master です。
- **PATRONI/_KUBERNETES/_TMP/_ROLE/_LABEL** オプションのロールマスターまたはレプリカを含む一時ラベルの名前。このラベルの値は、常に対応するロールのデフォルトを使用します。必要な場合にのみ設定します。
- **PATRONI/_KUBERNETES/_USE/_ENDPOINTS** オptional。true に設定されている場合、Patroni は ConfigMap の代わりにエンドポイントを使用してリーダー選挙を実行し、クラスター状態を維持します。
- **PATRONI/_KUBERNETES/_POD/_IP** オptional。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` が有効な場合に必要で、ポッドの PostgreSQL が昇格するときにリーダーエンドポイントのサブセットを設定するために使用されます。
- **PATRONI/_KUBERNETES/_PORTS** オptional。サービスオブジェクトにポートの名前がある場合、同じ名前が Endpoint オブジェクトに表示される必要があります。それ以外の場合、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。Patroni はそれを使用してリーダーエンドポイントのサブセットを更新します。このパラメータは、`PATRONI_KUBERNETES_USE_ENDPOINTS` が設定されている場合にのみ使用されます。
- **PATRONI/_KUBERNETES/_CACERT** オptional。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` ファイルを指定します。提供されない場合、patroni は ServiceAccount シークレットによって提供される値を使用します。
- **PATRONI/_RETRIABLE/_HTTP/_CODES** 再試行する K8sAPI からの HTTP ステータスコードのオプションリスト。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ptional。Raft 接続をリッスンする `ip:port`。指定しない場合、self_addr が使用されます。
- **PATRONI/_RAFT/_PARTNER/_ADDRESSES** クラスター内の他の Patroni ノードのリスト `['ip1:port1', 'ip2:port2']`。すべてのエンティティを引用することが重要です。
- **PATRONI/_RAFT/_DATA/_DIR** Raft ログとスナップショットを保存するディレクトリ。指定しない場合、現在の作業ディレクトリが使用されます。
- **PATRONI/_RAFT/_PASSWORD** オptional。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/_POSTGRESQL/_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コロンで最後のものつまり `listen: 127.0.0.1,127.0.0.2:5432` の後に追加される限り、複数のコンマ区切りアドレスが許可されます。Patroni は、このリストの最初のアドレスを使用して、PostgreSQL ノードへのローカル接続を確立します。
- **PATRONI/_POSTGRESQL/_CONNECT/_ADDRESS** 他のノードおよび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/_POSTGRESQL/_PROXY/_ADDRESS** Postgres の横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 `proxy_url` として DCS のメンバーキーに書き込まれ、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。既存または Patroni によって初期化されます。
- **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、`PATH` 環境変数が実行可能ファイルを見つけるために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その他の状況でこのファイルを作成します。場所は Patroni によって書き込み可能である必要があります。
- **PATRONI/_REPLICATION/_USERNAME** レプリケーションユーザー名。ユーザーは初期化中に作成されます。レプリカはこのユーザーを使用して、ストリーミングレプリケーションを介してレプリケーションソースにアクセスします
- **PATRONI/_REPLICATION/_PASSWORD** レプリケーションパスワード。ユーザーは初期化中に作成されます。
- **PATRONI/_REPLICATION/_SSLMODE** オプション **PATRONI/_REPLICATION/_SSLMODE** 接続パラメーターにマップします。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** にアクセスしてください。デフォルトモードは **prefer** です。

- **PATRONI/_REPLICATION/_SSLKEY** オプション **PATRONI/_REPLICATION/_SSLKEY** 接続パラメーターにマップします。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** 接続パラメーターにマップします。これは、**PATRONI/_REPLICATION/_SSLKEY** で指定された秘密キーのパスワードを指定します。
- **PATRONI/_REPLICATION/_SSLCERT** オプション **PATRONI/_REPLICATION/_SSLCERT** 接続パラメーターにマップします。これは、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** 接続パラメーターにマップします。これは、クライアントがサーバーの証明書を検証するために使用する 1 つ以上の認証局 CA 証明書を含むファイルの場所を指定します。
- **PATRONI/_REPLICATION/_SSLCRL** オプション **PATRONI/_REPLICATION/_SSLCRL** 接続パラメーターにマップします。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** 接続パラメーターにマップします。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** 接続パラメーターにマップします。これは、安全な GSSTCP/IP 接続をサーバーとネゴシエートするかどうか、またはその優先順位を決定します。
- **PATRONI/_REPLICATION/_CHANNEL/_BINDING** オプション **PATRONI/_REPLICATION/_CHANNEL/_BINDING** 接続パラメーターにマップします。これは、クライアントのチャンネルバインディングの使用を制御します。
- **PATRONI/_SUPERUSER/_USERNAME** スーパーユーザーの名前、初期化 initdb 中に設定され、後で Patroni が postgres に接続するために使用されます。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 中に設定します。
- **PATRONI/_SUPERUSER/_SSLMODE** オプション **PATRONI/_SUPERUSER/_SSLMODE** 接続パラメーターにマップします。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** にアクセスしてください。デフォルトモードは **prefer** です。
- **PATRONI/_SUPERUSER/_SSLKEY** オプション **PATRONI/_SUPERUSER/_SSLKEY** 接続パラメーターにマップします。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/_SUPERUSER/_SSLPASSWORD** オプション **PATRONI/_SUPERUSER/_SSLPASSWORD** 接続パラメーターにマップします。これは、**PATRONI/_SUPERUSER/_SSLKEY** で指定された秘密キー

のパスワードを指定します。

- **PATRONI/_SUPERUSER/_SSLCERT** オプション **PATRONI/_SUPERUSER/_SSLCERT** 接続パラメーターにマップします。これは、クライアント証明書の場合を指定します。
- **PATRONI/_SUPERUSER/_SSLROOTCERT** オプション **PATRONI/_SUPERUSER/_SSLROOTCERT** 接続パラメーターにマップします。これは、クライアントがサーバーの証明書を検証するために使用する 1 つ以上の認証局 CA 証明書を含むファイルの場合を指定します。
- **PATRONI/_SUPERUSER/_SSLCRL** オプション **PATRONI/_SUPERUSER/_SSLCRL** 接続パラメーターにマップします。これは、証明書失効リストを含むファイルの場合を指定します。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** 接続パラメーターにマップします。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合を指定します。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** 接続パラメーターにマップします。これは、安全な GSSTCP/IP 接続をサーバーとネゴシエートするかどうか、またはその優先順位を決定します
- **PATRONI/_SUPERUSER/_CHANNEL/_BINDING** オプション **PATRONI/_SUPERUSER/_CHANNEL/_BINDING** 接続パラメーターにマップします。これは、クライアントのチャンネルバインディングの使用を制御します。
- **PATRONI/_REWIND/_USERNAME** `pg_rewind` のユーザーのオプションの名前。ユーザーは `postgres11+` の初期化中に作成され、必要なすべての **PATRONI/_REWIND/_USERNAME** が付与されます。
- **PATRONI/_REWIND/_PASSWORD** `pg_rewind` のユーザーのオプションパスワード。ユーザーは初期化中に作成されます。
- **PATRONI/_REWIND/_SSLMODE** オプション **PATRONI/_REWIND/_SSLMODE** 接続パラメーターにマップします。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** にアクセスしてください。デフォルトモードは `prefer` です。
- **PATRONI/_REWIND/_SSLKEY** オプション **PATRONI/_REWIND/_SSLKEY** 接続パラメーターにマップします。これは、クライアントの証明書で使用する秘密キーの場合を指定します。
- **PATRONI/_REWIND/_SSLPASSWORD** オプション **PATRONI/_REWIND/_SSLPASSWORD** 接続パラメーターにマップします。これは、**PATRONI/_REWIND/_SSLKEY** で指定された秘密キーのパスワードを指定します。
- **PATRONI/_REWIND/_SSLCERT** オプション **PATRONI/_REWIND/_SSLCERT** 接続パラメーターにマップします。これは、クライアント証明書の場合を指定します。
- **PATRONI/_REWIND/_SSLROOTCERT** オプション **PATRONI/_REWIND/_SSLROOTCERT** 接続パラメーターにマップします。これは、クライアントがサーバーの証明書を検証するために使用する 1 つ以上の認証局 CA 証明書を含むファイルの場合を指定します。
- **PATRONI/_REWIND/_SSLCRL** オプション **PATRONI/_REWIND/_SSLCRL** 接続パラメーターにマップします。これは、証明書失効リストを含むファイルの場合を指定します。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** 接続パラメーターにマップします。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合を指定します。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** 接続パラメーターにマップします。これは、安全な GSSTCP/IP 接続をサーバーとネゴシエートするかどうか、またはその優先順位を決定します
- **PATRONI/_REWIND/_CHANNEL/_BINDING** オプション **PATRONI/_REWIND/_CHANNEL/_BINDING** 接続パラメーターにマップします。これは、クライアントの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-auth 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-auth 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 形式の秘密キー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するためのパスワードを指定します。
- **PATRONI/_RESTAPI/_CAFILE** クライアント証明書の検証中に使用する、信頼できる CA の証明書を含む CA_BUNDLE を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 RESTAPI がクライアント証明書をチェックしない場合。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 を使用する場合、証明書の署名検証が成功した場合、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 要求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 エンドポイントを呼び出すことができ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、含まれないものは拒否されます。
- **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** オプションの HTTP ヘッダーにより、RESTAPI サーバーは HTTP 応答で追加情報を渡します。
- **PATRONI/_RESTAPI/_HTTPS/_EXTRA/_HEADERS** オプションの HTTPS ヘッダーにより、TLS が有効になっているときに RESTAPI サーバーは HTTP 応答で追加情報を渡します。これは、`http_extra_headers` で設定された追加情報も渡します。
- **PATRONI/_RESTAPI/_REQUEST/_QUEUE/_SIZE** : (オプション) PatroniRESTAPI が使用する TCP ソケットの要求キューサイズを設定します。キューがいっぱいになると、それ以降の要求は `/"Connectiondenied/"` エラーを取得します。デフォルト値は 5 です。

警告:

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

3.3.13 CTL

- **PATRONICTL/_CONFIG/_FILE** オプションの構成ファイルの場所。
- **PATRONI/_CTL/_USERNAME** オプション保護された RESTAPI エンドポイントにアクセスするための基本認証ユーザー名。提供されない場合、 **PATRONI/_CTL/_USERNAME** は RESTAPI の `/"username/"` パラメーターで提供された値を使用します。
- **PATRONI/_CTL/_PASSWORD** オプション保護された RESTAPI エンドポイントにアクセスするための基本認証パスワード。提供されない場合、 **PATRONI/_CTL/_PASSWORD** は RESTAPI の `/"password/"` パラメーターで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 RESTAPISL 証明書の検証中に使用する信頼できる CA の証明書が含まれる CA_BUNDLE ファイルまたはディレクトリを指定します。提供されない場合、**PATRONI/_CTL/_CACERT** は RESTAPI の `"/cafile/"` パラメーターで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キー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キーファイルを復号するためのパスワードを指定します。

Patroni 構成には 3 種類があります。

- 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 edit-config` ツールまたは Patroni `patronictl edit-config` を使用して、いつでも設定できます。変更されたオプションが起動構成の一部でない場合、それらは次のウェイクアップサイクルですべてのノードに非同期的に適用され、その後リロードされます。ノードが構成を適用するために再起動が必要な場合コンテキストポストマスターを使用した `patronictl edit-config` の場合、値が変更された場合、これを示す特別なフラグ `pending_restart` が `members.dataJSON` に設定されます。さらに、ノードステータスは `"/restart_pending/": true` を表示することによりこれを示します。

- ローカル *configuration file patroni.yml*。

これらのオプションは構成ファイルで定義され、動的構成より優先されます。 `patroni.yml` は、SIGHUP を Patroni プロセスに送信するか、 `POST /reload REST-API` 要求を実行するか、 `patroni.yml` を実行することにより、実行時に Patroni を再起動せずに変更およびリロードできます。ローカル構成は、単一の YAML ファイルまたはディレクトリのいずれかです。ディレクトリの場合、そのディレクトリ内のすべての YAML ファイルは、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、最後のファイルの出現が優先されます。

- *Environment configuration* 。

一部の `"/Local/"` 構成パラメーターを環境変数で設定/オーバーライドすることが可能です。環境構成は、動的環境で実行しており、一部のパラメーターを事前に知らない場合に非常に役立ちますたとえば、 `docker` 内で実行している場合、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

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

- **max_connections**: 100

- **max_locks_per_transaction** :64
- **max_worker_processes** :8
- **max_prepared_transactions** :0
- **wal_level** :hot_standby
- **track_commit_timestamp** :off

以下のパラメーターの場合、PostgreSQL は、プライマリとすべてのレプリカの間で値が等しい必要はありません。ただし、レプリカがいつでもプライマリになる可能性を考慮すると、それらを別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
- MB

これらのパラメーターは検証されて、正常であること、または最小値を満たしていることが確認されます。

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ーは postgresql.conf に書き込まれませんが、引数のリストとして pg_ctl start に渡され、postgresql.conf よりも最高の優先順位が与えられます。

postgresql.listen、postgresql.listen、その postgresql.listen、つまり Patroni postgresql.listen または postgresql.listen 変数を介して postgresql.listen、postgresql.listen のようなパラメーターもあります。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` ファイルが存在するかどうか、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はベース構成として使用され、`postgresql.base.conf` および `postgresql.conf` を無視します。
- custom_conf パラメータが設定されず、`postgresql.base.conf` が存在する場合、名前が変更された /"original/" 構成が含まれ、ベース構成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` の名前が `postgresql.base.conf` に名前変更され、ベース構成として使用されます。

- 動的オプション上記の例外を含むは`postgresql.conf`にダンプされ、インクルードは`postgresql.conf`でベース構成`postgresql.base.conf`または`custom\_conf`のファイルのいずれかに設定されます。したがって、構成ファイルを再度読み取って、インクルードが存在するかどうかを確認することなく、新しいオプションを適用できます。
- Patroni がクラスターを管理するために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る場合`pg\_settings`のコンテキストとこれらのオプションの実際の値を確認する必要があります。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。

パラメーターは次の順序で適用されますランタイムに最高の優先順位が与えられます。

1. ファイル`postgresql.base.conf`または`custom\_conf`ファイル設定されている場合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ー

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、パトローニの実行に不可欠なパラメーターが適用されるように保証 4、および`postgresql`を管理する構成ツールの余地を残します。Patroni を介さずに`conf`を直接呼び出します (1)。

3.4.2 共有メモリに影響を与える PostgreSQL パラメーター

PostgreSQL には、使用する共有メモリのサイズを決定するパラメーター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ーを変更するには、PostgreSQL の再起動を有効にする必要があります、共有メモリ構造をプライマリノードよりもスタンバイノードで小さくすることはできません。

前に説明したように、パトローニは、通常次のもので構成される *dynamic configuration* を介して値の変更を制限します。

1. `patronictl edit-config` または RESTAPI `/config` エンドポイントを介して変更を適用する
2. `patronictl restart` または RESTAPI `/restart` エンドポイントを介したノードの再起動

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 デーモンを再起動することにより PostgreSQL を再起

動しようとしています。たとえば、`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、クラスターでフェー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に追加の注意を払う必要があります。

- これらの設定の値を **increase** する場合
 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
- これらの設定の値を **decrease** する場合
 1. 最初に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: これらの設定の値を **Note:** した後、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` で表示されるものよりも低い値に PostgreSQL は `FATAL` メッセージで終了するため、スタンバイが無限クラッシュループに入ることを防ぐためにこれを行います。言い換えれば、プライマリでのこれらの変更に関してプライマリで `pg_controldata` が最新の状態になった場合にのみ、スタンバイの設定を下げるができます。

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

3.4.3 Patroni 構成パラメーター

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

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

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

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

3.5 構成の生成と検証

Patroni は、Patroni *local configuration* 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、次のことができます。

- 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たとえば *Patroni integration* の準備手順として。
- 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

yaml 形式でサンプルの Patroni 構成ファイルを生成します。パラメーター値は `yaml` を使用して定義されます。それ以外の場合、パトロニで使用されるデフォルトまたはユーザーが後で定義する値の `#FIXME` 文字列。

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** は、PostgreSQL バージョンを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` -結果の保存に使用される構成ファイルへのフルパス。提供されない場合、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 インスタンスの `yaml` 形式で Patroni 構成を生成します。PostgreSQL 接続には、提供された DSN 優先または PostgreSQL `yaml` のいずれかが使用されます。パスワードを提供しない場合、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、構成ファイル、`postmaster` コマンドライン、または環境変数を介して設定された場合とは無関係に、次の Patroni 構成パラメーター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` 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 PostgreSQL GUC。

`scope`、`postgresql.listen` または `postgresql.datadir` が Postgres GUC から設定されていない場合、各 `scope` 値が使用されます。

値定義に適用される他のルール

- `name` `PATRONI_NAME` 環境変数設定されている場合は値、それ以外の場合は、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または `port` GUC 値に使用されるポートの `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ーも構成に含まれます。

パラメーター

configfile

結果の保存に使用される構成ファイルへのフルパス。提供されない場合、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 構成の検証

```
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、 `PATRONI_CONFIG_VARIABLE` 環境変数から読み取るか、設定されていない場合は `PATRONI_CONFIG_VARIABLE` から読み取ろうとします。

第4章 PatroniRESTAPI

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

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 要求に対して、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 ドキュメントを結果ます。JSON ドキュメントが不要または不要な場合は、GET の代わりに HEAD または OPTIONS メソッドの使用を検討することができます。

- PatroniRESTAPI への次の要求は、Patroni ノードがリーダーロックを使用して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** を結果ます。

- GET /
- GET /primary
- GET /read-write

- GET /standby-leader は、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 パトローニノードがリーダーロックがある場合、HTTP ステータスコード GET /leader を結果ます。以前の 2 つのエンドポイントとの主な違いは、PostgreSQL が **primary** または **standby_leader** として実行されているかどうかを考慮していないことです。
- GET /replica レプリカヘルスチェックエンドポイント。Patroni ノードが **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、レプリケーションレイテンシーもチェックし、指定された値を下回る場合にのみステータスコード GET /replica?lag=<max-lag> を返します。DCS のキー `cluster.last_leader_operation` は、リーダー wal 位置と、パフォーマンス上の理由からレプリカのレイテンシの計算に使用されます。max-lag は、バイト整数または人間が読み取り可能な値で指定できます。たとえば、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、yaml 構成管理の GET /replica?tag_key1=value1&tag_key2=value2 セクションのユーザー定義タグ key1 および key2 と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 構成の値が照会値と一致しない場合、HTTP ステータスコード 503 が返されます。

次の要求では、リーダーまたはスタンバイリーダーステータスをチェックしているため、Patroni はユーザー定義タグを適用せず、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 パトロニ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 を結果ます。
- 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 パトロニ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 を結果ます。
- GET /asynchronous?lag=<max-lag> または GET /async?lag=<max-lag> 非同期スタンバイチェックエンドポイント。 asynchronous または async からのチェックに加えて、レプリケーションレイテンシーもチェックし、指定された値を下回る場合にのみステータスコード GET /asynchronous?lag=<max-lag> を返します。DCS のキー cluster.last_leader_operation は、リーダー wal 位置と、パフォーマンス上の理由からレプリカのレイテンシの計算に使用されます。max-lag は、バイト整数または人間が読み取り可能な値で指定できます。たとえば、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 を結果ます。
- GET /liveness パトロニハートビートループが正常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 秒以上前またはレプリカで 2*ttl である場合は GET /liveness を返します。livenessProbe に使用できます。
- GET /readiness は、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 がアップして実行されている場合、HTTP ステータスコード GET /readiness を結果ます。このエンドポイ

ントは、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、readinessProbe に使用できます。

readiness および liveness エンドポイントは両方とも非常に軽量で、SQL は実行しません。プローブは、リーダーキーの有効期限が間もなくエキスパイアと失敗し始めるように構成する必要があります。デフォルト値が ttl 30s の例では、プローブは次のようになります。

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

4.2 モニタリングエンドポイント

GET /patroni は、リーダーレース中にパトローニによって使用されます。監視システムで使用される場合もあります。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 と同じ構造です。

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:** が有効になった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 エンドポイントを介して Prometheus 形式で Patroni メトリックを取得します。

```

$ curl http://localhost:8008/metrics

# HELP patroni_version Patroni semver without periods. \
# TYPE patroni_version gauge
patroni_version{scope="batman",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、クラスタースイッチオーバー/フェールオーバーの履歴に関するビューを提供します。このフォーマットは、pg_wal ディレクトリの履歴ファイルの内容に非常に似ています。唯一の違いは、新しいタイムラインが作成された時期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,

```

(次のページに続く)

(前のページからの続き)

```

50331856,
"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,

```

(次のページに続く)

(前のページからの続き)

```

    "parameters": {
      "hot_standby": "on",
      "wal_level": "hot_standby",
      "max_wal_senders": 5,
      "max_replication_slots": 5,
      "max_connections": "101"
    }
  }
}

```

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

ノードがこの構成を処理したことを確認してみましょう。最初に、5 秒ごとにログ行の印刷を開始する必要があります `loop_wait=5`。/"max_connections/"の変更には再起動が必要であるため、/"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
}

```

パラメーターの削除

一部の設定を削除または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,

```

(次のページに続く)

(前のページからの続き)

```

"postgresql": {
  "use_slots": true,
  "use_pg_rewind": true,
  "parameters": {
    "hot_standby": "on",
    "unix_socket_directories": ".",
    "wal_level": "hot_standby",
    "max_wal_senders": 5,
    "max_replication_slots": 5
  }
}
}

```

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

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

```

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

```

4.5 スイッチオーバーおよびフェイルオーバーエンドポイント

4.5.1 スイッチオーバー

`/switchover` エンドポイントは、クラスターが正常な場合にのみ機能しますリーダーがあります。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。候補が提供されない場合、クラスターのすべての資格のあるノードは、リーダーが退任した後、リーダーレースに参加します。

POST 要求の JSON 本文で、`leader` フィールドを指定する必要があります。 `candidate` および `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、要求はさまざまな HTTP ステータスコードと本文を返す場合があります。ステータスコード **200** は、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、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` は、現在スケジュールされているスイッチオーバーを削除するために使用できます。

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、特定の時刻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できますたとえば、すべての同期スタンバイが昇格するほど十分に健全でない場合は非同期スタンバイに。ただし、クラスターがリーダーを持たないための要件はありません。フェールオーバーは、正常なクラスターで実行することもできます。

POST 要求の 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: *Be very careful*。特定の状況でデータ損失が発生する可能性があるため、このエンドポイントを使用する場合。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 および POST /failover エンドポイントは、それぞれ POST /switchover および 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	スイッチオーバー
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
一時停止中に実行できます	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、またはフェールオーバー/スイッチオーバー候補としてリーダーになるためには、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 を介して到達可能であること。
- `nofailover` タグが `true` に設定されていません。
- ウォッチドッグが完全に機能するように構成する必要がある場合。
- 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、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、クラスタータイムラインより小さいタイムライン番号を持たない。
- *synchronous mode* で
 - スwitchオーバーの場合候補ありとなしの両方 `/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ールオーバーの場合、このチェックは省略されます。

警告:

リーダーのないクラスターで手動フェイルオーバーが発生する場合、次の場合でも候補者は昇格できません。

- 同期モードが有効になっている場合、`/sync` キーメンバーにはありません。
- 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。
- 最後の既知のクラスタータイムラインよりも小さいタイムライン番号があります。

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ことにより、特定のノードで Postgres を再起動できます。POST 要求の JSON 本文では、オプションでいくつかのリスタート条件を指定できます。
 - **restart_pending** `true` に設定されている場合ブール値、Patroni は PostgreSQL 構成にいくつかの変更を適用するために、再起動が保留中の場合にのみ PostgreSQL を再起動します。
 - **role** ノードの現在のロールが POST 要求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 要求で指定されたものよりも小さい場合にのみ再起動を実行します。
 - **timeout** PostgreSQL が接続の受け入れを開始するまで待機する時間。 `primary_start_timeout` をオーバーライドします。
 - **schedule** タイムゾーン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` および `DELETE /restart` エンドポイントは、それぞれ `POST /restart` および `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取りおよび適用するように指示します。これは、`SIGHUP` シグナルを Patroni プロセスに送信することと同じです。再起動が必要な Postgres パラメーター `POST /reload` などを変更した場合、`POST /restart` エンドポイントを呼び出すか、`POST /reload` のヘルプを使用して、Postgres のリスタートを明示的に行う必要があります。

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

Patroni が障害が発生した Postgres を復旧または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ためには、要求本文で `{/"force/":true}` を指定できます。

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

第5章 patronictl

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。クラスターでの操作を簡単に実行できるよう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` が見つからない場合と `restapi.cacfile` 設定 `ctl.cacert` がみつからない場合に関係します。詳細については、**restapi** を参照してください。
- DCS **etcd** などパトロニが使用する 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て、環境変数または構成ファイルを介してそれらのオプションを設定する方法を理解します。

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。

構成ファイルを使用する場合、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。これは、システムに応じて、これらのパスのいずれかの下にあると予想されます。

- MacOSX `~/Library/Application Support/patroni`
- MacOSXPOSIX `~/.patroni`
- Unix `~/config/patroni`
- UnixPOSIX `~/.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` を設定する。

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

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る場合、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、それぞれを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` 自分自身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 に接続文字列を提供します。

この引数は、`patronictl` 構成から DCS および `namespace` 設定をオーバーライドするために使用でき、または構成に欠落している場合は定義できます。

値は `DCS://HOST:PORT/NAMESPACE` のフォーマットである必要があります。例 `etcd3://localhost:2379/service` は、`service` 名前空間の下に保存されている Patroni クラスターを使用して `localhost` 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 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: これは、概要の構文です。

- 角かっこ間のオプションはオプションナル。
- 中括弧間のオプションは、`/"chooseoneofset/"` 操作を表します。
- `[, ... ]` を使用した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、値を与える必要がある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とき、コマンドの概要は、上記の概要の `SUBCOMMAND` の置き換えとして見られる必要があります。

以下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ます。例のために、Patroni の GitHub リポジトリに存在する構成ファイル `postgres0.yml`、`postgres1.yml` および `postgres2.yml` を使用します。

5.2.1 patronicldsn

概要

```
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と一致する場合、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。

指定されない場合、`patronictl` は、`scope` 構成が存在する場合は、そこからそれを取得しようとします。

-r / --role

指定されたロールを持つメンバーを選択します。

ロールは次のいずれかです。

- `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。このパラメーターを省略するのと同じです。または

-m / --member

指定された名前の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 グループの一部であ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 edit-config

概要

```
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、それで DCS を更新します。

注釈: TTY を介して呼び出されると、コマンドはページャを介して動的構成の差分を表示しようとします。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ます。別のページャが必要な場合は、`PAGER` 環境変数を目的のページャに設定します。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

指定されない場合、`patronictl` は、`scope` 構成が存在する場合は、そこからそれを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変更します。

指定されない場合、`patronictl` は、`citus.group` 構成が存在する場合は、そこからそれを取得しようとします。

`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

指定された動的構成オプションを指定された値で設定します。

`CONFIG` は、`.` で結合されたレベルを含む YAML ツリー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

特定の動的 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、複数の `-s / --set` オプションを `CONFIG_FILE` からの構成ごとに 1 つずつ指定するのと似ています。

`CONFIG_FILE` は、適用される動的構成を YAML 形式で含むファイルへのパスです。 `stdin` から読み取る場合は、`-` 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、有効にする新しい動的構成を YAML 形式で含むファイルへのパスです。stdin から読み取る場合は、- 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。

スクリプトに役立ちます。

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 パトロニクスフェイルオーバー

概要

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

説明

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

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

- リーダーはいません。または
- 同期クラスターで使用可能な同期スタンバイはありません。

同期モードが有効になっている場合、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、これらの場合は、`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昇格し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。

指定されない場合、`patronictl` は、`scope` 構成が存在する場合は、そこからそれを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

フェイ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。

指定された場合、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

フェールオーバー時に昇格するノード。

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。

スクリプトに役立ちます。

例

ノード postgresql2 へのフェイルオーバー

```
$ 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 パトロニカルフラッシュ

概要

```
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る場合は破棄します。

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ない場合、すべてが検討されます。

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

指定されたロール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。

ロールは次のいずれかです。

- **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。このパラメーター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

フラッシュを実行するときに確認プロンプトをスキップするフラグ。

スクリプトに役立ちます。

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 後援者の歴史

概要

```
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とスイッチオーバーイベントの履歴を表示します。

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` ファイルから取得した理由。

Timestamp

イベントが発生した時刻。

New Leader

イベント中に昇格したパトローニメンバー。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

指定されない場合、`patronictl` は、`scope` 構成が存在する場合は、そこからそれを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の履歴を表示します。

CITUS_GROUP は、Citus グループの ID です。

指定されない場合、`patronictl` は、`citus.group` 構成が存在する場合は、そこからそれを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。

フォーマットは次のいずれかです。

- **pretty** 履歴をきれいなテーブルとして印刷します。または
- **tsv / t** で区切られた列を使用して、履歴を表情報として印刷します。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

フラッシュを実行するときに確認プロンプトをスキップするフラグ。

スクリプトに役立ちます。

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
```

(次のページに続く)

(前のページからの続き)

```

Timestamp: '2023-09-11T22:11:27.125527+00:00'
- LSN: 50331864
  New Leader: postgresql0
  Reason: no recovery target specified
  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 後援者リスト

概要

```

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。

Member

パトローニメンバーの名前。

Host

メンバーがいるホスト。

Role

メンバーの現在のロール。

次のいずれかです。

- **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。

考えられる状態のいくつかの例

- **running** Postgres が現在アップして実行されている場合。
- **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 を取得している場合。
- **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 メンバーとその上流間のメガバイト単位のレプリケーション遅延の量。

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、ノードに再起動が必要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` 形式での拡張出力が有効になっている印刷。または
 - ノードの再起動が必要な場合。
-

Scheduled restart

Patroni メンバーによって管理される 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存在し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` 形式での拡張出力が有効になっている印刷。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ています。空の値は、タグが設定されていないか、デフォルト値で構成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` 形式での拡張出力が有効になっている印刷。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含むデフォルトタグがある場合。
-

Scheduled switchover

Patroni クラスターで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: テーブル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` の場合にのみ表示されます。

Maintenance mode

クラスターモニタリングが現在一時停止されている場合。

注釈: テーブルフッターに表示されます。

クラスターが一時停止され、出力形式が `pretty` の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。

指定されない場合、`patronictl` は、`scope` 構成が存在する場合は、そこからそれを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 属性の値が空の場合でも、強制的に表示します。

注釈: `pretty` および `tsv` 出力フォーマット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印刷する前にタイムスタンプを印刷します。

-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。

フォーマットは次のいずれかです。

- `pretty` 履歴をきれいなテーブルとして印刷します。または
- `tsv` / `t` で区切られた列を使用して、履歴を表情報として印刷します。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、秒単位の更新間の間隔です。

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

拡張列を使用してきれいな形式でクラスターに関する情報を表示します。

```
$ 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を使用して 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 patronictl 一時停止

概要

```

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設定し、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。

指定されない場合、`patronictl` は、`scope` 構成が存在する場合は、そこからそれを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。

指定されない場合、`patronictl` は、`citus.group` 構成が存在する場合は、そこからそれを取得しようとします。

--wait

すべての Patroni メンバーが一時停止するまで待ってから、呼び出し元に制御を戻します。

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機します。

```
$ 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 patronictl クエリー

概要

```
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。

指定されない場合、`patronictl` は、`scope` 構成が存在する場合は、そこからそれを取得しようとします。

--group

指定された Citus グループを照会します。

CITUS_GROUP は、Citus グループの ID です。

-r / --role

指定されたロールを持つメンバーを選択します。

ロールは次のいずれかです。

- **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。このパラメーター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

接続してクエリー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、デフォルトの USERNAME。

-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、デフォルトで `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が使用されます。

--password

接続するユーザーのパスワードの入力を求めます。

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。

フォーマットは次のいずれかです。

- **pretty** クエリ出力をきれいなテーブルとして印刷します。または
- **tsv** クエリ出力を表形式の情報として印刷します /t で区切られた列。または
- **json** クエリー出力を JSON 形式で印刷します。または
- **yaml** クエリー出力を YAML 形式で印刷します。

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、実行する SQL コマンドです。

--delimiter

tsv 形式で情報を印刷する場合の区切り文字、または省略した場合は /t。

-W

2 秒ごとにクエリーを自動的に再実行します。

-w / --watch

指定された間隔でクエリーを自動的に再実行します。

TIME は、再実行間の秒単位の間隔です。

例

postgres ユーザーとして SQL コマンドを実行し、パスワードを求めます。

```
$ 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 形式で印刷します。

```
$ 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 形式で印刷します。

```
$ 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 で 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 patronictl reinit

概要

```
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リビルドします。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

MEMBER_NAME

Postgres インスタンスがリビルド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。

スクリプトに役立ちます。

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 のリビルドを要求し、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 |
+-----+-----+-----+-----+-----+-----+
Success: reinitialize for member postgresql2
```

(次のページに続く)

(前のページからの続き)

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

5.2.10 patronictlreload

概要

```
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、何も変更されない場合でも、管理対象 Postgres インスタンスで `pg_ctl reload` もトリガーします。

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ない場合、すべてが検討されます。

--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

指定されたロールを持つメンバーを選択します。

ロールは次のいずれかです。

- **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。このパラメーター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フラグ。

スクリプトに役立ちます。

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 patronictl promote

概要

```
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: この操作は、DCS から Patroni クラスターの情報を破壊します。

パラメーター

CLUSTER_NAME

Patroni クラスターの名前。

--group

特定の 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

確認を求めた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。

フォーマットは次のいずれかです。

- `pretty` メンバーをきれいなテーブルとして印刷します。または
- `tsv /t` で区切られた列を使用して、メンバーを表形式情報として印刷します。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
This cluster currently is healthy. Please specify the leader name to continue:
↪postgresql0
```

5.2.12 patronicrestart

概要

```
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ている 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

特定の 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

指定されたロールを持つメンバーを選択します。

ロールは次のいずれかです。

- **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。このパラメーターを省略した場合と同じです。

--any

指定されたフィルタに一致するノードの中から 1 つのランダムな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 バージョンです。

--pending

Pending restart として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に再起動が発生するようにスケジュールします。

TIMESTAMP は、再起動が発生するときのタイムスタンプです。明確な形式、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フラグ。

スクリプトに役立ちます。

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 |
```

(次のページに続く)

(前のページからの続き)

```

+-----+-----+-----+-----+-----+
| 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 で再起動が発生す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

概要

```

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。

指定されない場合、`patronictl` は、`scope` 構成が存在する場合は、そこからそれを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。

指定されない場合、`patronictl` は、`citus.group` 構成が存在する場合は、そこからそれを取得しようとします。

--wait

すべての Patroni メンバーが一時停止を解除するまで待ってから、呼び出し元に制御を戻します。

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

概要

```
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。

指定されない場合、`patronictl` は、`scope` 構成が存在する場合は、そこからそれを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。

指定されない場合、`patronictl` は、`citus.group` 構成が存在する場合は、そこからそれを取得しようとします。

例

クラスター `batman` の動的構成を表示する

```
$ 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 パトロニクススイッチオーバー

概要

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

説明

`patronictl switchover` は、クラスターでスイッチオーバーを実行します。

クラスターが正常なときに使用するよう設計されています。例

- リーダーがいます。
- 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。

指定されない場合、`patronictl` は、`scope` 構成が存在する場合は、そこからそれを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

スイッチオーバー時に降格す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し、プライマリロールを引き受けるノード。

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に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時点のタイムスタンプです。明確な形式、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すぐ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。

スクリプトに役立ちます。

例

ノード postgresql2 でスイッチオーバーします。

```
$ 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 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 patronictl トポロジ

概要

```
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: テーブルヘッダーに表示されます。

Member

パトローニメンバーの名前。

注釈: この列の情報は、レプリケーション接続の観点から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

メンバーの現在のロール。

次のいずれかです。

- **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。

考えられる状態のいくつかの例

- running Postgres が現在アップして実行されている場合。
- 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 を取得している場合。
- 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 メンバーとその上流間のメガバイト単位のレプリケーション遅延の量。

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、ノードに再起動が必要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。

ノードが再起動が必要な場合に表示されます。

Scheduled restart

Patroni メンバーによって管理される 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存在し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ています。空の値は、タグが設定されていないか、デフォルト値で構成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、またはデフォルト以外の値を含むデフォルトタグがある場合に表示されます。

Scheduled switchover

Patroni クラスターで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: テーブル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

クラスターモニタリングが現在一時停止されている場合。

注釈: テーブル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。

指定されない場合、`patronictl` は、`scope` 構成が存在する場合は、そこからそれを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、秒単位の更新間隔です。

例

クラスターのトポロジを表示する `batman -- postgresql1` および `postgresql2` は `postgresql0` から複製しています。

```
$ 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 保護者バージョン

概要

```
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が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 グループの Patroni クラスターを検討します。

CITUS_GROUP は、Citus グループの ID です。

例

patronictl のみのバージョンを取得します。

```
$ 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 のバージョンと、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 レプリカイメージングとブートストラップ

Patroni では、新しいレプリカの作成をカスタマイズできます。また、新しい空のクラスターがブートストラップされているときに起こること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ブートストラップを排他的に呼び出します。

6.1 ブートストラップ

PostgreSQL は、新しいクラスターを初期化するための `initdb` コマンドを提供し、Patroni はデフォルトでそれを読み出します。特定の場合、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、ビルトインメソッドをカスタムアクションに置き換える必要があります。Patroni は、新しいクラスターをブートストラップするユーザー定義スクリプトの実行をサポートし、それらにいくつかの必要な引数、つまりクラスターの名前とデータディレクトリへのパスを提供します。これは、Patroni 構成の `bootstrap` セクションで構成されます。例

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

各ブートストラップメソッドは、少なくとも `name` および `command` を定義する必要があります。デフォルト動作をトリガーするために特別な `initdb` メソッドが使用できます。この場合、`method` パラメーターを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を使用して指定できます。構成ファイルで定義された固定パラメーターに加えて、Patroni は2つのクラスター固有のパラメーターを提供します。

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

特別な `no_params` パラメーターを `True` に設定することにより、これら2つの追加フラグの受け渡しを無効にできます。

ブートストラップスクリプトが 0 を返す場合、Patroni は、自分自身が生成する PostgreSQL インスタンスを構成および起動しようとします。中間ステップのいずれかが失敗するか、スクリプトがゼロ以外の値を返した場合、Patroni はブートストラップが失敗したと想定し、自分自身の後にクリーンアップし、初期化ロックを解放して別のノードにブートストラップの機会を与えます。

`recovery_conf` ブロックがカスタムブートストラップ方法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起動する前に `recovery.conf` を生成しますまたは、PostgreSQL>=12 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、`recovery_target_*` パラメーターの少なくとも 1 つが含まれる必要があります。

`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せず、構成されたりカバリ設定 PostgreSQL>=12 をオーバーライドしません。これは、適切なリカバリ構成を生成する pgBackRest のようなツールを使用してバックアップからブートストラップする場合に役立ちます。

それに加えて、カスタムブートストラップメソッド構成で通知された追加のキー/値ペアは、引数として `--name=value` 形式で `command` に渡されます。例

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

設定済みの `command` が `--arg1=value1 --arg2=value2` コマンドライン引数でさらに呼び出されるようになります。

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に障害が発生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

例として、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 --api-url https://barman-host:7480 recover
    barman-server: my_server
    ssh-command: ssh postgres@patroni-host
```

注釈: `patroni_barman recover` では、Barman ホストで Barman と pg-backup-api の両方を構成している必要があるため、バックアップ API を介してリモート `barman recover` を実行できます。上記の例では、

使用可能なパラメーター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 ビルディングレプリカ

Patroni は、新しいレプリカを作成するために実績のある `pg_basebackup` を使用します。その欠点の 1 つは、実行中のリーダーノードが必要であることです。もう 1 つは、バックアップデータの「オンザフライ」圧縮と、古いバックアップファイルのビルトインのクリーンアップの欠如です。WAL-E、`pgBackRest`、`Barman` などの他のバックアップソリューション、または単に独自のスクリプトをローリングする人もいます。これらの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
    - basebackup
  pgbackrest:
    command: /usr/bin/pgbackrest --stanza=<scope> --delta restore
    keep_data: True
```

(次のページに続く)

```
no_params: True
basebackup:
  max-rate: '100M'
```

例 Barman

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

注釈: `patroni_barman recover` では、Barman ホストで Barman と `pg-backup-api` の両方を構成している必要があるため、バックアップ API を介してリモート `barman recover` を実行できます。上記の例では、使用可能なパラメーター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方法とそれらの実行順序を定義します。Patroni は、0 を返す最初のメソッドで停止します。各メソッドは、構成ファイルの別のセクションを定義し、実行するコマンドとそのコマンドに渡すカスタムパラメーターをリストする必要があります。すべてのパラメーターは `--name=value` 形式で渡されます。ユーザー定義のパラメーターに加えて、Patroni はいくつかのクラスター固有のパラメーターを提供します。

--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	クローンを作成するクラスターメンバーに接続する接続文字列プライマリまたは他のレプリカ。接続文字列のユーザーは、SQL およびレプリケーションプロトコルコマンドを実行できます。

特別な `no_leader` パラメーターが定義されている場合、Patroni は実行中のリーダーやレプリカが存在しない場合でも、レプリカ作成メソッドを呼び出すことができます。その場合、空の文字列が接続文字列で渡されます。これは、以前に実行され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役立ちます。

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

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

`basebackup` メソッドは特殊なケースです。 `create_replica_methods` が空の場合に使用されますが、 `create_replica_methods` メソッド間で明示的にリストすることも可能です。このメソッドは、 `pg_basebackup` で新しいレプリカを初期化します。ベースバックアップは、 `clonefrom` タグを持つレプリカがない限りリーダーから取得されます。その場合、このようなレプリカのいずれかが `pg_basebackup` のオリジンとして使用されます。構成なしで動作します。ただし、 `basebackup` 構成セクションを指定することは可能です。他のメソッド構成と同じルールが適用されます。つまり、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ー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するか、作成するタールまたは圧縮されたベースバックアップにオプションを提供する場合、パトロニはそれからレプリカを作成できません。 `basebackup` セクションに渡されるパラメーターの名前または値に対して実行される検証はありません。また、シンボリックリンクが WAL フォルダーに使用される場合、オプションとして正しい `--waldir` パスを指定するのはユーザーの責任であるため、レプリカのビルドアップまたは再初期化の後、シンボリックリンクが持続するようにしてください。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` パラメーターを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、たとえば `--verbose` の場合)。これらの 2 つの例を検討します。

```
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方法が失敗した場合、Patroni は次のイベントループサイクル中にすべての方法を順番に再試行します。

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 は非同期レプリケーション用に PostgreSQL を構成します。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ます。

7.1 非同期モードの耐久性

非同期モードでは、クラスターは可用性を保証するために、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するか、他の理由で使用できなくなると、Patroni は十分に正常なスタンバイをプライマリに自動的にプロモートします。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、事実上回復不能です [1]。

失われる可能性のあるトランザクションの量は、`maximum_lag_on_failover` パラメーターを介して制御されます。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、フェールオーバーでの損失データの量は、ワーストケースになります。トランザクションログの `maximum_lag_on_failover` バイトと、最後の `t1` 秒に書き込まれた量 `loop_wait / 2` 秒平均的な場合)。ただし、一般的な定常状態のレプリケーション遅延は 1 秒未満です。

デフォルトでは、リーダー選挙を実行するときに、パトローニ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である可能性があります。 `check_timeline` パラメーターの値を `true` に変更することにより、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 同期レプリケーション

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

ホスト型データセンター環境 AWS、Rackspace、または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、少なくとも 3 つの Postgres データノードを使用して、1 つのホストに障害が発生した場合の書き込み可用性を保証します。

PostgreSQL 同期レプリケーションを使用しても、すべての状況でトランザクション損失ゼロを保証するものではありません。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に同時に障害が発生すると、すべてのトランザクションが含まれていない可能性がある 3 番目のノードが昇格します。

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容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、Patroni は、スタンバイにクライアント [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含まれていることが確実でない限り、スタンバイをプロモートしません。これは、一部のサーバーが利用可能な場合でも、システムは書き込みに利用できない場合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、すべての状況でのコミットのマルチノードの耐久性は保証されません。使用可能な適切なスタンバイがない場合、プライマリサーバーは書き込みを受け入れますが、レプリケーションは保証しません。このモードでプライマリに障害が発生すると、スタンバイは昇格しません。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、自動的に昇格します。この動作により、2 ノードクラスターで同期モードを使用できるようになります。

`synchronous_mode` がオンで、スタンバイがクラッシュすると、パトロニの次の反復が実行され、プライマリをスタンドアロンモードに切り替えるまで、コミットがブロックされます書き込みの最悪の場合の遅延は `ttl` 秒、平均的な場合は `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、コミットサービスの中断は発生しません。スタンバイは、PostgreSQL のシャットダウンが開始される前に、同期スタンバイ業務から自分自身を解放するようにプライマリに信号を送信します。

各書き込みが少なくとも 2 つのノードに永続的に保存されることを保証する必要がある場合は、`synchronous_mode` に加えて `synchronous_mode_strict` を有効にします。このパラメーターは、使用可能な同期スタンバイ候補がない場合、Patroni がプライマリで同期レプリケーションをオフにしないようにします。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` をオフにしていない限り書き込みに利用できず、少なくとも 1 つの同期レプリカが登場するまで、すべてのクライアントの書き込み要求をブロックします。

`nosync` タグを `true` に設定することにより、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、遅い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スタンバイに設定することをお勧めします。タグ `nostream` を `true` に設定しても、同じ効果があります。

同期モードは、PatroniREST インターフェイスを介してオンおよびオフにできます。手順については、[dynamic configuration](#) を参照してください。

注 PostgreSQL で同期レプリケーションが実装される方法のため、`synchronous_mode_strict` を使用す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ときに

PostgreSQL バックエンドがキャンセルされると、クライアントのタイムアウトまたはバックエンドの障害によるパケットキャンセルの結果、トランザクションの変更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、スタンバイプロモーションの場合に失われる可能性があります。

7.4 同期レプリケーションファクター

パラメーター `synchronous_node_count` は、Patroni が同期スタンバイデータベースの数を管理するために使用されます。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、メンバーが参加および離脱するときに DCS および `synchronous_standby_names` の状態を調整します。

7.5 同期モードの実装

同期モードの場合、Patroni は、最新のプライマリおよび現在の同期スタンバイデータベースを含む DCS の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に厳密な順序制約で更新されます。

- ノードは、書き込みトランザクションを受け入れることができるときは常に、最新のリーダーとしてマークされる必要があります。Patroni がクラッシュするか、PostgreSQL がシャットダウンしていない場合、このインバリエントの違反が発生する可能性があります。
- ノードは同期スタンバイとして公開されている限り、PostgreSQL で同期スタンバイとして設定する必要があります。
- リーダーまたは現在の同期スタンバイではないノードは、自分自身を自動的に昇格させることはできません。

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

HA ループ反復ごとに、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、同期ステータスの削除を要求していない場合、選択されたままです。それ以外の場合、レプリケーションで最も進んでいる同期に使用可能なクラスターメンバーが選択されます。

第8章 スタンバイクラスター

Patroni は、`/"スタンバイクラスター/"`と呼ばれる機能を使用して、リモートデータセンターリージョンへのカスケードレプリケーションの実行もサポートしています。このタイプのクラスターには次の特徴があります。

- `/"standbyleader/"` リモートノードから複製することを除き、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しています。

スタンバイリーダーは、DCS のリーダーロックを保持および更新します。リーダーロックの有効期限が切れると、カスケードレプリカは選択を実行して、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、それが複製する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を除き、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 **後援者リスト** または **後援者リスト** 出力に表示されません。

柔軟性を高めるために、`standby_cluster` セクションに `:ref: `create_replica_methods` <custom_replica_creation>` キーを提供することにより、クラスターが`/"スタンバイモード"`のときにレプリカとリカバリ WAL レコードの作成方法を指定できます。クラスターがデタッチされ、`postgresql` セクションの `create_replica_methods`` によって制御される通常のクラスターとして機能する場合、これはレプリカの作成とは異なります。`/"standby/"` と `/"normal/"` `create_replica_methods`` は両方とも、`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、パトロニ構成でセクション `standby_cluster` を指定する必要があります。

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

これらのオプションは、クラスターブートストラップ中に 1 回のみ適用され、その後にそれらを変更する唯一の方法は DCS を介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` を見つけることを想定しており、ベースバックアップ後に見つからない場合は起動しません。リモートプライマリが `postgresql.conf` を他の場所で保持している場合、それを PGDATA にコピーするのはあなたの責任です。

スタンバイクラスターでレプリケーションスロットを使用する場合、プライマリクラスターで対応するレプリケーションスロットも作成する必要があります。これは、スタンバイクラスター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 の永久レプリケーションスロット機能を使用して、`primary_slot_name` と同じ名前のレプリケーションスロット、または `primary_slot_name` が提供されていない場合はそのデフォルト値を維持できます。

リモートサイトがプライマリに接続する単一のエンドポイントを提供しない場合、`standby_cluster.host` セクションにソースクラスターのすべてのホストをリストできます。`standby_cluster.host` にコンマで区切られた複数のホストが含まれる場合、Patroni は次のことを行います。

- スタンバイリーダーノードの `primary_conninfo` に `target_session_attrs=read-write` を追加します。
- `pg_rewind` を実行する必要があるかどうかを判断するとき、またはスタンバイクラスターのすべてのノードで `pg_rewind` を実行するときは、`target_session_attrs=read-write` を使用します。
- `pg_rewind` が正常に動作するには、クラスターを `data page checksums initdb` の `--data-checksums` オプションで初期化する必要があるか、`wal_log_hints` を `on` に設定する必要があることに注意してください。そうしないと、`pg_rewind` は正常に機能しません。

別のスタンバイクラスターまたはプライマリクラスターのスタンバイメンバーからスタンバイクラスターをレプリケートする可能性があります。そのためには、`standby_cluster.host` セクションで単一のホストを定義する必要があります。ただし、この場合、`pg_rewind` はスタンバイクラスターでの実行に失敗することに注意する必要があります。

第9章 ウォッチドッグのサポート

複数の PostgreSQL サーバーをプライマリとして実行すると、タイムラインの分岐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には、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入れないことを確認する必要があります。通常の場合では、Patroni 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、PostgreSQL を停止することによりこれを達成しようとします。ただし、これはさまざまな理由で発生しない場合があります。

- Patroni は、バグ、メモリ不足状態、またはシステム管理者が誤って強制終了したために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って一時停止されている VM、またはその他のインフラストラクチャの問題のため、Patroni は実行できません。

これらの条件下での正しい動作を保証するために、Patroni はウォッチドッグデバイスをサポートしています。ウォッチドッグデバイスは、指定された時間フレーム内にキープアライブハートビートを取得しない場合に、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に障害が発生した場合に備えて、フェイルセーフのレイヤーが追加されます。

Patroni は、PostgreSQL をプライマリに昇格する前に、ウォッチドッグをアクティブにしようとします。ウォッチドッグのアクティブ化が失敗し、ウォッチドッグモードが **required** の場合、ノードはリーダーになることを拒否します。リーダー選挙に参加することを決定するとき、パトローニは、ウォッチドッグ構成でリーダーになることができるかどうかを確認します。PostgreSQL を降格した後たとえば手動フェイルオーバーのため、Patroni はウォッチドッグを再度無効にします。Patroni が一時停止状態にある場合、ウォッチドッグも無効になります。

デフォルトでは、Patroni は TTL の有効期限が切れる 5 秒前に有効期限が切れるようにウォッチドッグを設定します。loop_wait=10 および ttl=30 のデフォルト設定では、これにより、システムが強制的にリセットされる前に HA ループが完了するまでに少なくとも 15 秒 `ttl - safety_margin - loop_wait` が発生します。デフォルトでは、DCS へのアクセスは 10 秒後にタイムアウトするように構成されています。これは、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` を意味します。

安全マージンは、リーダーキーの更新とウォッチドッグキープアライブ間の時間のために Patroni が予約した時間です。Patroni は、リーダーキーの更新を確認したら、すぐにキープアライブを送信しようとします。Patroni プロセスが適切な瞬間に長時間一時停止された場合、キープアライブは、ウォッチドッグをトリガーせずに安全マージンを超えて遅延される場合があります。これにより、リーダーキーの有効期限が発生する前にウォッチドッグ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。あらゆる状況でウォッチドッグがトリガーされるように保証には、`safety_margin` を -1 に設定してウォッチドッグタイム

アウトを `ttdl // 2` に設定することにより、TTL の半分の後に有効期限が切れるようにウォッチドッグを設定します。この保証が必要な場合は、おそらく `ttdl` を増やしたり、`loop_wait` および `retry_timeout` を減らしたりする必要があります。

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

9.1 Linux でのソフトウェアウォッチドッグの設定

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

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

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

テストの場合、`modprobe` コマンドラインに `soft_noboot=1` を追加して、リブートを無効にすることが役立つ場合があります。この場合、ウォッチドッグはカーネルリングバッファに行をログに記録するだけで、`dmesg` を介して可視されます。

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

第10章 クラスターの一時的停止/再開モード

10.1 ゴール

特定の状況では、Patroni は DCS にクラスター状態を保持したまま、クラスターの管理から一時的に辞任する必要があります。考えられるユースケースは、メジャーバージョンのアップグレードや破損リカバリなど、クラスターでの一般的ではないアクティビティです。これらのアクティビティ中に、ノードはパトローニにとって不明な理由で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こともあり、プライマリを1つのみ実行するという仮定に違反します。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、実行中のクラスターから `detach` できる必要があります。

10.2 実装

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

- ノードごとに、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、Patroni は、メンバーが実行中の場合、メンバーノードで読み取り専用クエリを実行します。
- リーダーロックを使用した 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るつまり手動で降格されると、Patroni はノードをプロモーションして戻す代わりにロックを解放します。
- 手動スケジュール外の再起動、手動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。
- 'Parallel' プライマリが Patroni によって検出された場合、警告を発しますが、リーダーロックなしでプライマリを降格しません。
- クラスター内にリーダーロックがない場合、実行中のプライマリがロックを取得します。複数のプライマリノードがある場合、最初にロックを取得したプライマリが勝ちます。プライマリが完全がない場合、Patroni はレプリカをプロモーションしようとしません。このルールには例外があります。古いプライマリが手動昇格のために自分自身を降格したため、リーダーロックがない場合、昇格要求に記載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、レプリカを手動で昇格させた後、パトローニは、以前のリーダーからストリーミングしていたレプリカが新しいリーダーに切り替わることを確認します。
- Postgres が停止している場合、Patroni はそれを起動しようとしません。Patroni は、停止しても、管理している Postgres インスタンスを停止しようとしません。
- Patroni は、他のクラスターメンバーを表さない、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。

10.3 ユーザーガイド

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

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

第11章 DCS フェイルセーフモード

11.1 問題

Patroni は、リーダー選挙のタスクを解決し、ネットワーク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 はすぐに降格され、読み取り専用として起動されます。どの DCS が使用されるかに応じて、/"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 では、可能性はゼロに近いですが、K8sAPIEtcd によってサポートされている場合、より頻繁に観察される可能性があります。

11.2 現在の実装の理由

リーダーロックの更新の失敗は、2つの主な理由によって発生する可能性があります。

1. ネットワークパーティショニング
2. DCS がダウンしている

一般に、単一のノードからこれら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ています。分割されたネットワークの場合、パトロニ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場合があります。スプリットブレインを回避するために、リーダーロックの有効期限が切れる前に、古いプライマリは降格されます。

11.3 DCS フェイルセーフモード

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

11.4 低レベル実装の詳細

- 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/"outage/"の場合、既存のプライマリは `POST /failsafe RESTAPI` を介して `/failsafe` キーで提示されているすべてのメンバーに接続し、すべてのレプリカが確認した場合はプライマリとして実行し続けます。
- メンバーの 1 人が応答しない場合、プライマリは降格されます。
- レプリカは、プライマリがまだ生きていることを示す指標として、着信 `POST /failsafe RESTAPI` 要求を使用しています。この情報は `ttl` 秒間キャッシュされます。

11.5 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?ここでクォーラムに頼ることはできないでしょうか?

これは素晴らしい質問です。問題は、クォーラムに関する見方が 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にわたって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、このようなルールを導入および適用するメカニズムがありません。パトローニ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負けた部分に収まり、少数派ノードが勝った部分にある場合、プライマリを降格する必要があります。他のすべてのメンバーをチェックすることのみが、このような状況を検出できます。

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

DCS にアクセスできない場合、チェック「他のクラスターメンバーはすべてアクセス可能ですか?」ハートビートループのサイクルごとに `loop_wait` 秒ごとに実行されます。ポッド/ノードが終了すると、チェックは失敗し、Postgres は読み取り専用で降格され、DCS が復元されるまで復旧しません。

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

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 Patroni を構成できます。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 へのアクセスを失った場合、レプリカはそうでなかった場合はどうなりますか?

プライマリはフェイルセーフコードを実行し、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、プライマリが生存し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有効期限が切れて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12章 KubernetesでのPatroniの使用

Patroni は、Kubernetes オブジェクトを使用して、クラスターの状態を保存し、リーダーキーを管理できます。これにより、一貫性ストアなしで Kubernetes 環境で Postgres を操作できます。つまり、追加の Etcd 展開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 オブジェクトの2つの異なるタイプがあります。それらは ``kubernetes.use_endpoints`` または ``PATRONI_KUBERNETES_USE_ENDPOINTS`` 環境変数で構成されます。

12.1 エンドポイントを使用する

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

12.2 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、これらの 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、少なくとも2つの更新が必要です。1つはリーダーの ConfigMap に対して、もう1つは各エンドポイントに対してです。

トラフィックを Postgres リーダーに転送するには、``role_label`` patroni 構成で構成した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、ConfigMaps を使用する以外に方法がないことに注意してください。

12.3 構成

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

12.3.1 ロールラベルのカスタマイズ

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

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

1. `kubernetes.tmp_role_label` とポッドの元のロール値を使用して、一時的なラベルを追加します ``tmp_role`` など。ポッドが再起動すると、Patroni によって設定された次のラベルを取得します。

```
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的な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たとえば、`kubernetes.leader_label_value=primary` を設定します。ポッドが再起動すると、Patroni によって設定された次の新しいラベルを取得します。

```
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後、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、構成から一時的なラベルを削除し、すべてのポッドを更新します。

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

12.4 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。権限の問題のため、現在の状態では PersistentVolume を使用できないことに注意してください。
- `Spilo Project` には、永続ボリュームを使用できるフル機能の Docker イメージがあります。
- Kubernetes を使用して実行される Patroni で構成された Spilo イメージを展開するための `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` プロジェクトを見てください。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13章 Citus のサポート

Patroni を使用すると、`Multi-NodeCitus` クラスターの展開が非常に簡単になります。

13.1 要約

従う必要がある簡単なルールがいくつかあります。

1. `Citus PostgreSQL` のデータベース拡張機能は、すべてのノードで利用できる必要があります。サポートされている絶対最小の Citus バージョンは 10.0 ですが、ワーカーの透過的スイッチオーバーと再起動からすべてのメリットを得るには、少なくとも Citus 11.2 以上を使用することをお勧めします。
2. クラスター名 `scope` は、すべての Citus ノードで同じである必要があります。
3. スーパーユーザーの資格情報はコーディネーターとすべてのワーカーノードで同じである必要があります、`pg_hba.conf` はすべてのノード間のスーパーユーザーのアクセスを許可する必要があります。
4. `REST API` ワーカーノードからコーディネーターへのアクセスを許可する必要があります。たとえば、資格情報は同じである必要があります、構成されている場合、ワーカーノードからのクライアント証明書はコーディネーターが受け入れる必要があります。
5. 次のセクションを `patroni.yaml` に追加します。

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

その後、Patroni を起動するだけで、残りは処理されます。

1. `citus` 拡張機能は `shared_preload_libraries` に自動的に追加されます。
2. `max_prepared_transactions` がグローバル `max_prepared_transactions` で明示的に設定されていない場合、Patroni は自動的に `2*max_connections` に設定します。
3. `citus.local_hostname` GUC 値は、`localhost` から Patroni がローカルの PostgreSQL インスタンスに接続するために使用している値に調整されます。PostgreSQL がリッスンしていない可能性があるため、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` は `pg_dist_authinfo` テーブルに追加されて、クロスノード通信が許可されます。後でスーパーユーザーのユーザー名/パスワード/sslcert/sslkey を変更する場合は、それらを更新することを忘れないでください。

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()` ファンクションを使用して `pg_dist_node` テーブルに追加します。
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ールオーバー/スイッチオーバーが発生した場合に、`pg_dist_node` を維持します。

13.2 patronictl

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るだけです。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。

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

1. `list` および `topology` は、デフォルトで、Citus フォーメーションのすべてのメンバーコーディネーターとワーカーを出力します。新しい列 `Group` は、それらが属している Citus グループを示します。
2. すべての `patronictl` コマンドに、`--group` という名前の新しいオプションが導入されます。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た `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、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 [後援者リスト](#) 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更されます。

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

(次のページに続く)

(前のページからの続き)

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

13.3 Citus ワーカーのスイッチオーバー

Citus ワーカーノードのスイッチオーバーがオーケストレーションされると、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トランスペアレントにする機会を提供します。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で SQL トラフィックを `pause` することが可能です。スイッチオーバーは、トラフィックがコーディネーターに保持されたときに発生し、新しいプライマリワーカーノードが読み取り/書き込みクエリを受け入れる準備が整うとすぐに再開されます。

ワーカークラスターでの [パトロニクススイッチオーバー](#) の例

```
postgres@coord1:~$ patronictl switchover demo
+ Citus cluster: demo -----+
| Group | Member | Host | Role | State | TL | Lag in MB |
+-----+-----+-----+-----+-----+
| 0 | coord1 | 172.27.0.10 | Replica | running | 1 | 0 |
| 0 | coord2 | 172.27.0.6 | Sync Standby | running | 1 | 0 |
| 0 | coord3 | 172.27.0.4 | Leader | running | 1 | |
| 1 | work1-1 | 172.27.0.8 | Leader | running | 1 | |
| 1 | work1-2 | 172.27.0.2 | Sync Standby | running | 1 | 0 |
| 2 | work2-1 | 172.27.0.5 | Sync Standby | running | 1 | 0 |
| 2 | work2-2 | 172.27.0.7 | Leader | running | 1 | |
+-----+-----+-----+-----+-----+
Citus group: 2
Primary [work2-2]:
Candidate ['work2-1'] []:
When should the switchover take place (e.g. 2022-12-22T08:02 ) [now]:
Current cluster topology
+ Citus cluster: demo (group: 2, 7179854924063375386) -----+
| Member | Host | Role | State | TL | Lag in MB |
+-----+-----+-----+-----+-----+
| work2-1 | 172.27.0.5 | Sync Standby | running | 1 | 0 |
| work2-2 | 172.27.0.7 | Leader | running | 1 | |
```

(次のページに続く)

(前のページからの続き)

```

+-----+-----+-----+-----+-----+-----+
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          |
+-----+-----+-----+-----+-----+-----+

```

そして、これはコーディネーター側でどのように見えるかです。

```

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

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

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

```

13.4 DCS のピーク

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
...
```

ほとんどの DCS では、単一の再帰的読み取り要求で Citus クラスター全体を取得することが可能になるため、このようなアプローチが選択されました。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ため、ツリー全体を読み取ります。ワーカーノードは自分自身のグループのサブツリーのみを読み取り、場合によっては、コーディネーターグループのサブツリーを読み取ることができます。

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

ポッド環境変数を使用した Patroni 構成のいくつかの例

1. コーディネータークラスターの場合

```

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ーの場合

```

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"

```

お気づきのとおり、両方の例には `citus-group` ラベルセットがあります。このラベルにより、Patroni は特定の Citus グループに属するオブジェクトを識別できます。それに加えて、`citus-group` label と同じ値を持つ `PATRONI_CITUS_GROUP` 環境変数もあります。Patroni は、新しい Kubernetes オブジェクトの ConfigMaps または Endpoint を作成すると、それらに `citus-group: ${env.PATRONI_CITUS_GROUP}` ラベルを自動的に配置します。

```

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}

```

Patroni リポジトリの ``kubernetes`__` フォルダーに、Citus サポートを備えた Kubernetes での Patroni 展開の完全な例があります。

2つの重要なファイルがあります。

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

13.6 Citus アップグレードと PostgreSQL メジャーアップグレード

最初に、`documentation`__` 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ときは、``systemctl restart`` の代わりに [patronictlrestart](#) 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、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14章 スタンドアロンをPatroniクラスターに変換する

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

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

14.1 プロシージャ

以下は、既存の Postgres クラスターを Patroni 管理対象クラスターに変換する手順の概要を示しています。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起動して実行されており、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でユーザーを作成するサンプル SQL コマンドを見つけることができます。環境ごとにユーザー名とパスワードを置き換える必要があります。関連するユーザーが既に存在する場合、この手順をスキップできます。

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

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

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

(次のページに続く)

(前のページからの続き)

```

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

```

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

1. systemd を介して Postgres を実行している場合、Postgressystemd 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。それには *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` 構成項目を介して既存のレプリケーションスロットを永続的なもの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のレプリケーションスロットを自動的に作成し、認識しないレプリケーションスロットをドロップすることに注意してください。ここで永久スロットを使用するというアイデアは、Patroni への移行が進行中である間、既存のスロットを持続できるようにすることです。詳細については、**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 が既に実行されていることを自動的に検出し、インスタンスの監視を開始します。
3. Postgres/"startupprocedure"を Patroni に引き継ぎます。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するには、この手順を次のように分割することができます。

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

4. ステップ 1.2. で永久スロットを構成した場合、パトロニ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たなら、1.2. コマンドを介して `slots` 構成からそれらを削除する必要があります。`slots` 構成からスロットを削除することにより、Patroni は必要がなくなったら元のスロットをクラスターからドロップできます。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、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
WHERE slot_name IN (

```

(次のページに続く)

(前のページからの続き)

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


第15章 PostgreSQL バージョンのメジャーアップグレード

現在、メジャーアップグレードを行う唯一の方法は次のとおりです。

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

スタンバイノードでの `pg_upgrade` の実行は、PostgreSQL ではサポートされていません。自分が何をしているのか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。

15.1 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訴えます。

`patroni.yml` の `postgresql.conf` と `postgresql.listen` の `listen_addresses` と `port` を確認する必要があります。 `pg_hba.conf` はこ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 にノードを再起動するように依頼した後、PostgreSQL はエラーメッセージを表示します `could not open configuration file "/etc/postgresql/10/main/pg_hba.conf": No such file or directory`

PostgreSQL 構成を管理する方法に応じて、さまざまな意味があります。*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場合にのみ `pg_hba.conf` セクションの設定に基づいて `pg_hba.conf` を生成します。このシナリオでは、PGDATA 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する必要があります。

第16章 他のツールとの統合

Patroni は、スタック内の他のツールと統合できます。このセクションでは、例のリストを示します。これは完全なリストではありませんが、Patroni を他のツールと統合する方法に関するアイデアを提供する可能性があります。

16.1 Barman

Patroni は、`pg-backup-api` と通信するロジックを備えた `patroni_barman` という名前のアプリケーションを提供しているため、Barman 操作をリモートで実行できます。

このアプリケーションには、現在いくつかのサブコマンド `recover` および `config-switch` があります。

16.1.1 patroni_barmanrecover

`recover` サブコマンドは、カスタムブートストラップまたはカスタムレプリカの作成方法として使用できます。これに関する詳細については、`recover` を参照してください。

16.1.2 patroni_barmanconfig-switch

`config-switch` サブコマンドは、Patroni で `on_role_change` コールバックとして使用するよう設計されています。例として、現在のプライマリから Barman ホストに WAL をストリーミングしていると仮定します。クラスターでフェールオーバーが発生した場合、新しいプライマリから WAL のストリーミングを開始することができます。これは、`on_role_change` コールバックとして `patroni_barman config-switch` を使用することにより実現できます。

注釈: そのサブコマンドは、`barman config-switch` コマンドに依存しています。このコマンドは、事前定義されたモデルを適用することにより Barman サーバーの構成をオーバーライドします。このコマンドは、Barman3.10 以降で利用できます。詳細については、Barman のドキュメントを参照してください。

これは、この Patroni ノードがプライマリに昇格する場合、構成モデルを適用するように Patroni を構成する方法の例です。

```
postgresql:
  callbacks:
    on_role_change: >
```

(次のページに続く)

(前のページからの続き)

```
patroni_barman
--api-url YOUR_API_URL
config-switch
--barman-server YOUR_BARMAN_SERVER_NAME
--barman-model YOUR_BARMAN_MODEL_NAME
--switch-when promoted
```

注釈: `patroni_barman config-switch` では、Barman ホストで Barman と `pg-backup-api` の両方を構成している必要があるため、バックアップ API を介してリモート `barman config-switch` を実行できます。また、適用する Barman モデルを事前に構成しておく必要があります。上記の例では、使用可能なパラメーターのサブセットを使用しています。`patroni_barman config-switch --help` を実行し、Barman ドキュメントを参照して、詳細情報を取得できます。

第17章 セキュリティに関する考慮事項

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

17.1 DCS の保護

Patroni と *patronictl* は両方とも、DCS にデータを保存および取得します。

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

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

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

17.2 RESTAPI の保護

RESTAPI の保護は、より複雑なタスクです。

PatroniRESTAPI は、リーダーレース中に Patroni 自分自身によって使用され、フェイルオーバー/スイッチオーバー/再初期化/リスタート/リロードを実行するために *patronictl* ツールによって使用され、HAProxy またはその他の種類のロードバランサーによって HTTP ヘルスチェックを実行し、そしてもちろん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 GET 要求、情報の取得のみと安全でない PUT、POST、PATCH および DELETE 要求、ノードの状態を変更するエンドポイントが含まれています。

`restapi.authentication.username` および `restapi.authentication.password` パラメーターを設定することにより、安全でないエンドポイントを HTTP ベーシック認証で保護できます。TLS を有効にすることなく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 が有効になり、PKI が確立されると、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です。

`restapi` セクションパラメーターは、サーバーへの TLS クライアント認証を有効にします。`verify_client` パラメーターの値に応じて、API サーバーは、安全な API 呼び出しと安全でない API 呼び出しの両方 `verify_client: required`、安全でない API 呼び出しのみ `verify_client: optional`、または API 呼び出しなし `verify_client: none` の場合、クライアント証明書の検証が成功する必要があります。

ctl セクションパラメーターは、クライアントへの TLS サーバー認証を有効にします patroni と同じ構成を使用する ctl ツール。insecure: true を設定して、クライアントによるサーバー証明書の検証を無効にします。TLS クライアントパラメータの詳細については、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保護することは、このドキュメントの範囲を超えており、<https://www.postgresql.org/docs/current/client-authentication.html> でカバーされています。

第18章 HA マルチデータセンター

複数のデータセンターに展開された PostgreSQL クラスターの高可用性は、レプリケーションに基づいています。これは同期または非同期 `replication_modes` です。

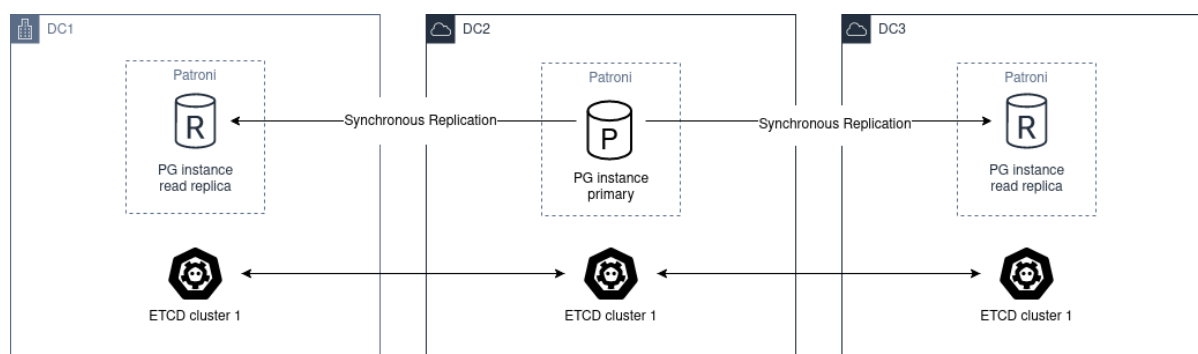
どちらの場合も、次の概念を明確にすることが重要です。

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

18.1 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 クラスターを作成するには、少なくとも 3 つが必要です。

アーキテクチャ図は次のとおりです。



別の DC を介して etcd、ZooKeeper、または Consul のクラスターを展開する必要があります。各ゾーンに 1 つずつ、少なくとも 3 つのノードを使用します。

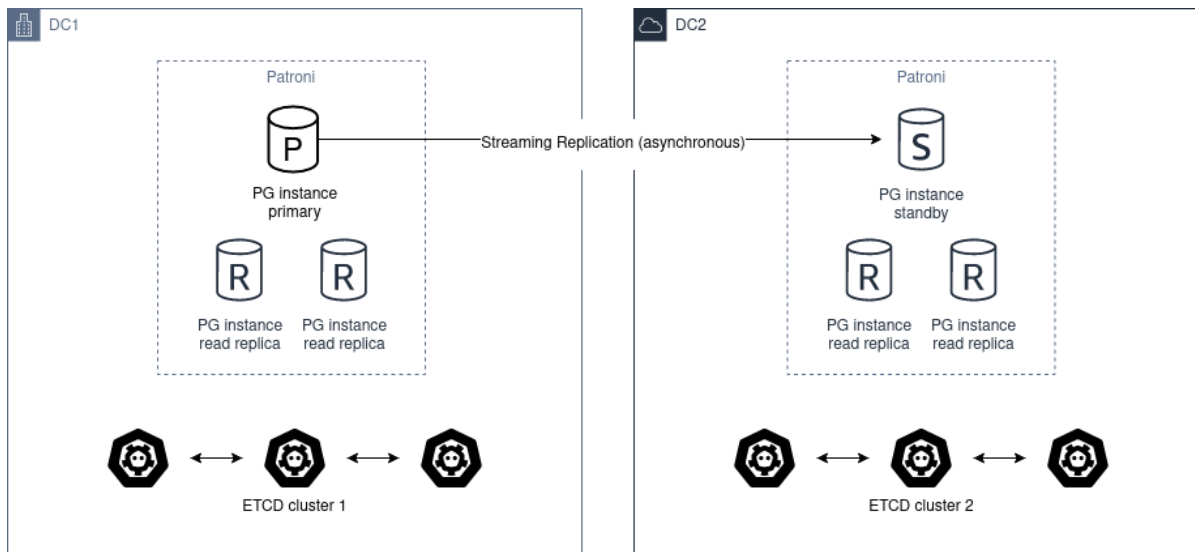
postgres に関しては、少なくとも 2 つのノードを別の DC に展開する必要があります。次に、グローバル `synchronous_mode: true` で `synchronous_mode: true` を設定する必要があります。

これにより、同期レプリケーションが有効になり、プライマリノードはいずれかのノードを同期として選択します。

18.2 非同期レプリケーション

データセンターが2つだけの場合、2つの独立した etcd クラスターを使用し、2 番目のデータセンターで Patroni *standby cluster* を実行することをお勧めします。最初のサイトがダウンしている場合は、`standby_cluster` を手動でプロモートすることができます。

アーキテクチャ図は次のとおりです。



DC2 は DC1 の状態を理解できないため、自動プロモーションは不可能です。

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"手動でプロモート"する必要があります。

警告: ソースクラスターがまだ稼働している場合、スタンバイクラスターを昇格させると、スプリットブレインが作成されます。

"/initial/"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て戻すと、`pg_rewrite` がトリガーされます。ただし、`pg_rewrite` が正常に機能するには、クラスターを `data page checksums initdb` の `--data-checksums` オプションで初期化するか、`wal_log_hints` を `on` に設定する必要がありますが、他の要因により `pg_rewrite` が失敗する可能性があります。
- スタンバイクラスターを最初から再構築します。

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

その前に、データベースを手動で調べて、DC1 と DC2 間のネットワークが動作を停止した時間から DC1 のクラスターを手動で停止した時間まで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19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ようとしています。

これがあなたの質問のほとんどを明確に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、[チャット](#) および [チャット](#) のヘルプを取得または問題を報告する方法を参照してください。

19.1 他の HA ソリューションとの比較

repmgr のような他のソリューションは必要ないのに、**Patroni** は DCS ノードの別のクラスターを必要とするのはなぜですか？

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、アクションを実行する時期を決定します。

一方、**Patroni** は、DCS に保存されている状態に依存します。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。

別の DCS クラスターを持つとアーキテクチャが肥大化する可能性があります。このアプローチでは、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が低くなります。

Postgres 管理に関する 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 自分自身も管理します。

Postgres ノードがまだ存在しない場合、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 ノードが既に存在する場合、**Patroni** は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。言い換えれば、プライマリノードに障害が発生すると、**Patroni** はレプリカにフェー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すると、**Patroni** はそのレプリカに再参加しようとします。

これは、**Patroni** を「HA ソリューションのテンプレート」と呼ぶ方法です。単に物理的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。

19.2 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、DCS の namespace および scope Patroni 設定の接頭辞が付いているパスの下に保存されます。

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 クラスターを指すさまざまな Patroni クラスターで namespace と scope の同じ組み合わせを使用しようとする、どうなりますか？

同じ namespace および scope を使用しようとする 2 番目の Patroni クラスターは、DCS で同じ組み合わせに関連する情報を見つけようとするため、Postgres を管理できません。ただし、互換性のない Postgres システム識別子。システム識別子の不一致により、Patroni は 2 番目のクラスターが別のクラスターを参照し、ユーザーが Patroni の構成を誤っていると仮定して、2 番目のクラスターの管理を中止します。

同じ DCS クラスターを共有するさまざまな Patroni クラスターを扱う場合、必ず別の namespace / scope を使用してください。

DCS クラスターを失うとどうなりますか？

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。

彼らの最初の結果は、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ということです- **DCS フェイルセーフモード** が有効になっていない限り

DCS クラスターを失った場合はどうすればよいですか？

DCS クラスターを失った場合、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復旧しました。これには、Patroni 側からのアクションは必要ありません。DCS クラスターが復旧すると、Patroni も復旧できるはずです。
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です。Patroni 側で変更は必要ありません。
3. 新しい DCS クラスターがさまざまなエンドポイントで作成されます。各 Patroni ノードの Patroni 構成の 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. 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、の 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 で動的構成を再作成します。Patroni クラスターの各メンバー。

DCS クラスターのマジョリティを失うとどうなりますか？

DCS は応答なくなり、Patroni は現在の読み取り/書き込み Postgres ノードを降格します。

注 Patroni は DCS の状態に依存して、クラスターでアクションを実行します。

DCS フェイルセーフモード を使用して、その状況を緩和できます。

19.3 patronicl

Patroni ホストで *patronicl* 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。

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

ただし、*patronicl* 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 と RESTAPI に到達できるように、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 **後援者リスト** コマンドの出力から消えたのはなぜですか？

後援者リスト で表示される情報は、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、そのノードで Patroni エージェントが実行されていないか、DCS と通信できません。

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 **後援者リスト** の出力に表示されません。

後援者リスト コマンドの出力で、Patroni メンバーの 1 つに関する情報が最新でないのはなぜですか？

後援者リスト で表示される情報は、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。言い換えれば、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 "migrate" が表示される場合があります。

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では、DCS 情報がすぐに更新されます。

19.4 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、Patroni クラスターのすべてのメンバーに適用される構成です。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する設定は、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介して指定できます。

パトローニ構成 の詳細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をオーバーライドします。

Note: 一部の PostgresGUC は、グローバル、つまり動的構成を介してのみ設定できます。それに加えて、Patroni がハードコーディングされた値を強制する GUC があります。

[パトローニ構成](#) の詳細を参照してください。

Patroni 構成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、サンプルの Patroni 構成または既存の Postgres インスタンスに基づいて 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` 構成でパラメーターを変更しましたが、Patroni はクラスターメンバーに変更を適用していません。何が間違っているのでしょうか？

`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完了すると、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 の構成を変更する必要があります。これは、次のいずれかを通じて実現されます。

- `patronictledit-config` ; または
- PATCH への PATCH リクエスト。

ローカル構成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、SIGHUP 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行うことができます。

- POST 要求を RESTAPI POST に送信します。または
- `patronictlreload` を実行します。または
- SIGHUP を使用して Patroni プロセスにローカルに通知します。
 - systemd を介して Patroni を起動した場合、次のコマンドを使用できます `systemctl reload PATRONI_UNIT.service`、`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、`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: を介したリロードが機能しない場合があります。

- 有効期限が切れた RESTAPI 証明書 `-k` の `-k` オプションを使用して、これを緩和できます。
- 間違った資格情報たとえば、構成ファイルの `restapi` または `ctl` 資格情報を変更し、Patroni と `restapi` に同じ構成ファイルを使用する場合。

環境構成を変更するにはどうすればよいですか？

環境構成は、起動中に Patroni によってのみ読み取られます。

それを念頭に置いて、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。

クラスターでフェールオーバーが発生しないように注意してください。 [patronictl 一時停止](#) をチェックすることに興味があるかもしれません。

リロードが必要な PostgresGUC を変更するとどうなりますか？

以前の質問で説明したように動的またはローカル構成を変更すると、Patroni は Postgres 構成のリロードを担当します。

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を `pending restart` のフラグでマークします。

メンバーを再起動する時期と方法は、あなたが決定する必要があります。これは、次のいずれかの方法で実現できます。

- [patronictlrestart](#) ; または
- POST への 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の点で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 構成の 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 で保存されている情報は API バージョン 3 では管理できないことに注意してください。その逆も可能です。

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 で完全に削除されます。

Patroni 構成で `use_slots` を有効にしていますが、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するレプリケーションスロットが上流ノードでドロップされます。その問題を回避するにはどうすればよいですか？

メンバーの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 から、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可能になりました。

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ことを確認するだけでなく、メンバーが消費した 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ことにした場合は、永続的なスロット構成を調整するのが **your responsibility** です。それ以外の場合、Patroni はスロットを永久に保持します。

Note: 3.2.0 より古い Patroni 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で作成されますが、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 を使用しても、小さな競合状態が発生する可能性があります。一番最初に、スロットがレプリカで作成されると、リーダーの同じスロットより前にある可能性があり、誰もスロットを消費していない場合、フェールオーバー後にいくつかのファイルが失われる可能性があります。

それを念頭に置いて、継続的アーカイブを構成することをお勧めします。これにより、必要な WAL を復元したり、PITR を実行したりできます。

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実行する場合があります。各 HA サイクルで、クラスターで一連のチェックを実行してその正常性を判断し、ステータスによっては、スタンバイへのフェールオーバーなどのアクションを実行する場合があります。

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。例えば 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場合があります。

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 HA サイクル中に ttl より長くリースを更新できない場合、リースは有効期限が切れ、クラスター内の 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適用することに注意してください。

19.5 Postgres 管理

Postgres 構成で PostgresGUC を直接変更できますか？

できますが、それは避けたほうがいいです。

Postgres の構成は Patroni によって管理され、構成ファイルを編集しようとする、最終的に上書きされる可能性があるため、Patroni によって挫折される場合があります。

Patroni によって実行される管理を克服するために使用できるオプションがいくつかあります。

- `$PGDATA/postgresql.base.conf` を介して 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。

これに関する詳細については、セクション [重要なルール](#) を参照してください。

いずれの場合も、Patroni を介してすべての Postgres 構成を管理することをお勧めします。これにより、管理が一元化され、必要に応じて Patroni のデバッグが簡単になります。

Postgres ノードを直接再起動できますか？

いいえ、**not** Postgres を直接管理してする必要があります。

Patroni なしで Postgres サーバーをバウンスしようすると、クラスターがフェー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 が公開した方法を介してそれを行ってください。

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！

詳しい手順については、[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 のアップとダウンを処理します。

それを念頭に置いて、`systemd` ユニットなどの Postgres クラスターを管理できる他のソースを **MUST** 無効にします。 `postgresql.service` 。 Patroni のみがクラスター内の Postgres インスタンスを起動、停止、および昇格できる必要があります。 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。例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、スプリットブレインが発生する場合があります。

19.6 概念と要件

Patroni の一部を構成するアプリケーションは何ですか？

Patroni は基本的にいくつかのアプリケーションを同梱しています。

- `patroni` これは Patroni エージェントであり、Postgres ノードの管理を担当します。
- `patronictl` これは、Patroni クラスターと対話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。スイッチオーバー、再起動、構成の変更などを実行します。詳細については、`patronictl` をご覧ください。

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、通常は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、リモートの Postgres ノードから変更をレプリケートするスタンバイとなるリーダーが存在します。次に、こ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スタンバイのセットが存在します。

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場合があります。

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` AKA `standby leader` は、リモート Postgres ノードから複製し、こ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 は、クラスター内の最小数の Postgres ノードを必要としますか？

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

Patroni は DCS から分離されていることに注意してください。

パトローニで pause はどういう意味ですか？

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 にステップバックするように要求できます。

これは、クラスターでメンテナンスを実行し、プライマリを停止したときのスタンバイへのフェイルオーバーなど、Patroni が HA に関連する決定を行うことを回避したい場合に役立ちます。

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

19.7 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動作しますか？

Patroni 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、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` ロックを保存します。

その `leader` ロックには、それに関連付けられた有効期限があります。リーダーノードが `leader` ロックのリースを時間内に更新できない場合、キーは最終的に DCS から有効期限が切れます。

`leader` ロックの有効期限が切れると、Patroni が `leader race` と呼ぶものがトリガーされます。すべてのノードがチェックの実行を開始して、`leader` ロールを引き継ぐための最適な候補かどうかを判断します。これらのチェックには、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適な候補として自分自身を見つけたすべての 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、読み取り/書き込みノードまたは `standby leader` に昇格し、他のメンバーはそ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！

これは、クラスターを一時的に一時停止することにより実現できます。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する場合は、一時停止を解除するだけです。

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

19.8 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 ノードはどうなるでしょうか?

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、およびカスタムレプリカ作成メソッドを記述することにより、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で作成し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。

19.9 モニタリング

Patroni クラスターをモニタするにはどうすればよいですか?

Patroni は、[PatroniRESTAPI](#) 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` モニタリングメトリックを Prometheus が使用できる形式で公開します。
- `/patroni` クラスターの状態を 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もの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、モニタリングチェックを実装できます。

第20章 リリースノート

20.1 バージョン3.3.0

2024-04-04 リリース

警告: 古い Patroni バージョンはすべて `ydiff>=1.3` と互換性がありません。

問題を/"fix"するために使用できる次のオプションがあります。

1. Patroni を最新バージョンにアップグレードする
2. Patroni をインストールした後、`ydiff<1.3` をインストールします
3. `cdiff` モジュールのインストール

New features

- `auth_data` を Zookeeper クライアントに渡す機能を追加します。ArasMumcuyan

接続に使用する認証資格情報を指定できます。

- Barman 統合のための contrib スクリプトを追加しました。IsraelBarthRubio

Barman 操作をリモートで実行でき、カスタムブートストラップ/カスタムレプリカメソッドまたは `on_role_change` コールバックとして使用できるアプリケーション `patroni_barman` を提供します。詳細については、`patroni_barman` を確認してください。

- JSON ログ形式のサポート alisalemmini

`plain` デフォルトとは別に、Patroni は `json` ログ形式もサポートするようになりました。`python-json-logger>=2.0.2` ライブラリをインストールする必要があります。

- `pending_restart_reason` 情報の表示 PolinaBungina

`pending_restart` フラグが設定される原因となった PostgreSQL パラメーターに関する拡張情報を提供します。`patronictl list` と `/patroni` の両方の RESTAPI エンドポイントは、パラメーター名とその/"diff/"を `pending_restart_reason` として表示するようになりました。

- `nostream` タグの実装 GrigorySmolkin

`nostream` タグが `true` に設定されている場合、ノードはレプリケーションプロトコルを使用して WAL をストリーミングせず、代わりにアーカイブリカバリエに依存します `restore_command` が構成されている場合。また、ノード自分自身とそのすべてのカスケードレプリカの永久論理レプリケーションスロットのコピーと同期も無効にします。

Improvements

- log セクションの検証を実装します。AlexanderKukushkin

これまで、バリデーターは、提供されたロギング構成の正しさをチェックしていませんでした。

- PostgreSQL パラメーター変更のロギングを向上させました。PolinaBungina

古い値を人間が判読可能な形式に変換し、`pg_controldata` と Patroni グローバル構成の不一致に関する情報をログに記録します。

Bugfixes

- 許可されない `pg_basebackup` オプションを適切にフィルターして除外する IsraelBarthRubio

バグのため、Patroni は `- setting: value` 形式で提供されている場合、`basebackup` レプリカブートストラップ方法用に構成された許可されていないオプションを適切にフィルタリングしていませんでした。

- `etcd3` 認証エラー処理を修正しました AlexanderKukushkin

要求を実行する直前に認証が行われなかった場合、`etcd3` 認証エラーが常に 1 回リトライします。また、再認証時にウォッチャーを再起動しないでください。

- バリデーターファイル検出のロジックの改善 Waynerv

`importlib` ライブラリを使用して、可能な場合は使用可能な構成パラメーターを含むファイルを検出します Python3.9 以降の場合。この実装はより安定しており、`zip` アーカイブに基づいた Patroni ディストリビューションを中断しません。

- `standby_cluster` セクションで複数のホストが指定されている場合にのみ `target_session_attrs` を使用します。AlexanderKukushkin

`standby_cluster.host` セクションにコンマで区切られた複数のホストが含まれる場合にのみ、`target_session_attrs=read-write` がスタンバイリーダーノードの `primary_conninfo` に追加されるようになりました。

- `ydiff` ライブラリバージョン 1.3+ の互換性コードを追加しました AlexanderKukushkin

Patroni は、Python モジュールではなく単なるターミナルツールであると考えられているため、パブリックではない `ydiff` のいくつかの API に依存しています。残念ながら、1.3 の API の変更により、古い Patroni バージョンが壊れました。

20.2 バージョン 3.2.2

2024-01-17 リリース

Bugfixes

- DCS がワイプされたときにレプリカを復元させないでください AlexanderKukushkin

これは、パトローニ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方法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した読み取りを使用するようになりました。AlexanderKukushkin

Consul は、更新したキーの `ModifyIndex` をすぐに取得する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。古い読み取りはデフォルトで許可されているため、キーの古いバージョンを取得する場合があります。

- 再起動が必要なパラメーターが元の値にリセットされた場合、Postgres 構成をリロードします 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するポリナ・ブンギナ

クラスターが正常な場合、既存のリーダーへのフェールオーバーは `no-op` です。

- Citus データベースと拡張機能を幕等に作成する AlexanderKukushkin、ZhaoJunwang

Citus データベースに依存関係を追加する必要がある場合に、`post_bootstrap` スクリプトでそれらを作成できます。

- 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構成 `{nofailover: false, failover_priority: 0}` では、レースに参加できませんでしたが、`nofailover` タグが優先される必要があるため、レースに参加する必要があります。

- PyInstaller のフリーズした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 構成の構成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ません。

- 実行中のスタンバイに参加するときに、リカバリ GUC と一部のパトローニ管理パラメーターを復元します AlexanderKukushkin

Patroni は、内部構造の 1 つで `port` が欠落していることに関するエラーが発生し、Postgresv12 以降の再起動に失敗していました。

- `pending_restart` フラグ周辺の修正 PolinaBungina

`recovery_target_action = promote` でカスタムブートストラップしている場合、または誰かがたとえば `ALTER SYSTEM` を使用して `hot_standby` または `wal_log_hints` を変更したときは、`pending_restart` を公開しないでください。

20.3 バージョン 3.2.1

2023-11-30 リリース

Bugfixes

- `patronictl` の `--format` 引数の受け入れられる値を制限します AlexanderKukushkin
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、シャットダウン時にレプリカノードがチェックポイント 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になっている場合。その結果、以前のプライマリは場合によって `pg_rewind` を実行する必要がありましたが、データの損失は関係ありません。
- ノード名の一意性チェックを実行するときに実際の HTTP 要求を実行します AlexanderKukushkin
コンテナで Patroni を実行している場合、トラフィックがポートでリッスンし、着信接続を受け入れる `docker-proxy` を使用してルーティングされる可能性があります。それは偽陽性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 クラスターの展開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、必要でない場合でもパトロニによって `pg_rewind` が呼び出されました。
- 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自分自身である `--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に関する問題を修正しました SophiaRuan
接続が適切にクローズされていないため、Postgres の再起動後、Postgres の `unknown` 状態を返す可能性があります。
- `postgres --describe-config` 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。

20.4 バージョン 3.2.0

2023-10-25 リリース

Deprecation notice

- `bootstrap.users` サポートはバージョン 4.0.0 で削除されます。新しいクラスターを展開した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、最小可能値をハードコードする AlexanderKukushkin

最小値 `loop_wait=2`、`retry_timeout=3`、`ttl=20`。値が小さいか、ルールに違反している場合、それらは調整され、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の優先順位を高く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]` および `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 の PostgreSQL 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、メインハートビートループのブロックを回避するのに役立ちます。

- ノード `sskserk` の `name` でいくつかのエンドポイントを強化する

モニタリングエンドポイントの場合、`name` は `scope` の横に追加され、メトリックエンドポイントの場合、`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保証します PolinaBungina

ログメッセージの精度が向上し、正常な同期クラスターの非同期ノードへのフェールオーバーが可能になります。

- 永続的な物理レプリケーションスロットを永続的な論理スロットと同様に動作させます AlexanderKukushkin

リーダーになることができるすべてのノードで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()` ファンクションを使用してスタンバイノード上のスロットの `restart_lsn` を進めます。

- `patronictl` の `--dcs` 引数を介して名前空間を指定する機能を追加しました。IsraelBarthRubio

構成ファイルなしで `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での追加パラメーターのサポートを追加します。IsraelBarthRubio

以前は、カスタム引数を `command` に追加することのみが可能でしたが、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、パトロニが Postgres に接続するために使用する値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` を使用しますが、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は無視されません。

- `on_reload` コールバックの `SIGCHLD` を処理します AlexanderKukushkin

これを行わないと、ゾンビプロセスが発生し、次の `on_reload` が実行された場合にのみ刈り取られます。

- Etcdv3 で作業する際の `AuthOldRevision` エラーを処理します。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、Etcd のユーザデータベースが更新されたときに発生します。

20.5 バージョン 3.1.2

2023-09-26 リリース

Bugfixes

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

`wal_keep_size` は、通常ユニットがある GUC であり、パトロニはその値を `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` 間の不一致を検出および解決するようになりました。AlexanderKukushkin

通常、Patroni は非常に特定の順序で `/sync` および `synchronous_standby_names` を更新しますが、バグの場合、または誰かが手動でリセットした場合 `synchronous_standby_names`、Patroni は一貫性のない状態になってしまいました。その結果、フェールオーバーが非同期ノードで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 の値を読み取ります 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。それを解決し、同様の問題を回避するために、Patroni は、既に実行中の Postgres に参加する場合、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迷惑な警告を沈黙させました AlexanderKukushkin

Patroni がすぐに再起動された場合、**WARNING** メッセージは `urllib3` によって生成されます。

20.6 バージョン 3.1.1

2023-09-20 リリース

Bugfixes

- プロモート時にフェイルセーフ状態をリセットする 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分自身を降格していました。

- `patronictl` の無駄な警告を沈黙させます。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迷惑な警告が表示されていた可能性があります。

- コーナーケースの同期モードを明示的に有効にする 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` 整数値検証のバグを修正しました。IsraelBarthRubio

ほとんどの場合、それは問題を発生させず、警告だけでした。

- スタンバイクラスターの論理スロットを返さないでください 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` 出力での `docstring` の表示を回避します。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

この機能は事実上動作しませんでした。

- 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 のサポートを実装しているときに発生しました。コーディネーターとすべてのワーカーで異なるため、識別子を非表示にする必要があります。

- Kubernetes 実装の `write_leader_optime` メソッドをオーバーライドする AlexanderKukushkin

このメソッドは、新しいプライマリになることが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/構成マップに書き込むことになっています。

- 一時停止中に停止した postgres を開始しないでください AlexanderKukushkin

競合状態のため、パトロニは、一部のリカバリパラメーター `primary_conninfo` または同様のものが変更されたため、スタンバイを再起動する必要があると誤って仮定していました。

- `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` のいずれも提供されない場合、機能しませんでした。

- コマンドラインで使用される整数パラメーターを適切に処理して `postgres` を起動するようにしました。PolinaBungina

値が文字列として提供され、整数にキャストされない場合、Citus クラスターの `max_connections` に基づいて `max_prepared_transactions` の誤った計算が発生しました。

- `pg_rewind` を決定するときは `pg_stat_wal_receiver` に依存しないでください。AlexanderKukushkin

`pg_stat_wal_recevier` によって報告された `received_tli` は、実際の再生タイムラインより進んでいる場合がありますが、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です。

20.7 バージョン 3.1.0

2023-08-03 リリース

Breaking changes

- `restapi.keyfile` および `restapi.certfile` のセマンティックが変更されました。AlexanderKukushkin

以前、Patroni は、`ctl` セクションに各構成パラメーターが存在しない場合、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ている場合 `restapi.verify_client` が `required` に設定されている場合、`restapi.verify_client` は `ctl.certfile`、`ctl.keyfile`、`ctl.keyfile_password` に `restapi.verify_client` を提供します。提供されない場合、Patroni は正しく動作しません。

New features

- ポッドのロールラベルを構成可能にする Waynerv

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

Improvements

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

さまざまな DCS、`bootstrap.dcs`、`ctl`、`restapi`、および `watchdog` セクションのパラメーター検証が向上しました。

- パトローニの実行中にリカバリー中にクラッシュした場合、リカバリーしていないスタート Postgres。AlexanderKukushkin

これにより、リカバリ時間が短縮され、不必要なタイムラインのインクリメントを防ぐことができます。

- `/status` キーの不要な更新を回避します。AlexanderKukushkin

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

- 失効した予備選にリーダー争いを勝たせてはいけないアレクサンダー・ククシュキン

リソースの不足がために重要な時間パトローニが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 を昇格させていないことをさらにチェックします。

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

`max_connections`、`max_wal_senders`、`max_prepared_transactions`、`max_locks_per_transaction`、`max_replication_slots`、または `max_worker_processes` の検証が失敗した場合、Patroni は正常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す。AlexanderKukushkin

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

Bugfixes

- シェル Waynerv を介して `archive_command` を実行する

Patroni は、シングルユーザーモードでクラッシュリカバリーを実行する前、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 `&&` のようなシェル演算子が含まれている場合、Patroni では動作しませんでした。

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

指定された候補は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。他の一部のノードが正常だったため、リーダーキーが削除されました。

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

リーダーレース中に、レプリカは、古いリーダーの 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

クラスター名前フィールドは、`tsv`、`json`、および `yaml` 出力フォーマット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から復帰した後、`pg_rewind` を使用して偽のプライマリをクラスターに参加させる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のため、`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。

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

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

20.8 バージョン 3.0.4

2023-07-13 リリース

New features

- スタンバイノードのレプリケーションステータスを可視化します AlexanderKukushkin

PostgreSQL9.6+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されている場合は `streaming` として、またはレプリケーション接続がなく、`restore_command` が設定されている場合は `in archive recovery` としてレプリケーション状態を報告します。状態は、DCS の `member` キー、RESTAPI、および `patronictl list` 出力に表示されます。

Improvements

- Etcdv3 のエラーメッセージの向上 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` のクォラム読み取りを使用します AlexanderKukushkin

Etcd または Consul クラスターは読み取り専用に低下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。今では、エラーで失敗します。

- 構成内の重複した名前からのスプリットブレインを防止します。MarkPekala

Patroni は、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 を照会しようとします。RESTAPI がアクセス可能な場合、Patroni はエラーで終了します。人的エラーからの保護に役立ちます。

- パトローニの実行中にクラッシュした場合、スタート Postgres がリカバリーしていません。AlexanderKukushkin

リカバリー時間を短縮することができ、不必要なタイムラインのインクリメントから役立ちます。

Bugfixes

- RESTAPISSL 証明書は、SIGHUP の受信時にリロードされませんでした。IsraelBarthRubio

リグレーションは 3.0.3 で導入されました。

- `max_connections` のようなパラメーターの整数 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。リグレッションは 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` が有効になっているときに不在の同期スタンバイを誤って待機しないように、`synchronous_commit=off` とともに `txid_current()` を実行します。

20.9 バージョン 3.0.3

2023-06-22 リリース

New features

- PostgreSQL16beta1 との互換性 AlexanderKukushkin
拡張された GUC のバリデータールール。
- 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。非標準の Postgres ディストリビューション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、AleksiiSukhov
PatroniRESTAPI が使用する TCP ソケットの要求キューサイズを設定します。キューがいっぱいになると、それ以降の要求は `"Connectiondenied/"` エラーを取得します。デフォルト値は 5 です。
- 新しいクラスターを初期化するときに `initdb` を直接呼び出します。MattBaker
以前は `pg_ctl` を介して呼び出され、`initdb` に渡されるパラメーター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影響を与えません。
- 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場合、`patroni --validate-config` は不平を訴えていました。
 - `postgresql.bin_dir` が空であるか、設定されていない場合にエラーを出さないようにします。代わりに、最初にデフォルトの PATH で Postgres バイナリを見つけてみてください。

- `postgresql.authentication.rewind` セクションをオプションにします。そ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` のエラーレポートの向上 IsraelBarthRubio
- `/n` シンボルは、実際の改行シンボルの代わりに、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
- スイッチオーバー中に昇格したワーカーからコーディネーターへの RESTAPI 呼び出しが失敗した場合、特定の Citus グループが無期限にブロックされたままになっていました。
- `patronictl` の `--dcs-url` オプションで ``etcd3`` URL を許可します。IsraelBarthRubio
- ユーザが ``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。

20.10 バージョン 3.0.2

2023-03-24 リリース

警告: バージョン 3.0.2 は、3.6 より古い Python のサポートを終了しました。

New features

- 同期スタンバイレプリカステータスを `/metrics` エンドポイントに追加しました。ThomasvonDein、AlexanderKukushkin
- 以前は、`primary / standby_leader / replica` のみをレポートしていました。
- `patronictl` での `PAGER` の使いやすい処理 IsraelBarthRubio
- `PAGER` 環境変数を介してページャーを構成可能になります。これは、デフォルトの `less` および `more` をオーバーライドします。
- `K8s` を再試行可能な `HTTP` ステータスコードを構成可能にします AlexanderKukushkin
- 一部の管理対象プラットフォームでは、ステータスコード `401 Unauthorized` を取得することができますが、これは数回の再試行後に解決される場合があります。

Improvements

- `recovery_target_action` が `promote` に設定されている場合にのみ、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 `kill` することを許可しないでください AlexanderKukushkin

`on_start/on_stop/on_role_change` は、通常、仮想 IP を追加/削除するために使用され、`on_reload` はそれらに干渉しないはずです。

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

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

Bugfixes

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

`default` とは異なる名前空間では機能しませんでした。

- メジャーバージョンが不明な場合は `PGDATA` に書き込まないでください AlexanderKukushkin

`start` 直後に `PGDATA` が空であった場合まだマウントされていない場合、Patroni は PostgreSQL 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()` 呼び出しはメタデータの同期を発生させず、変更はワーカーノードでは不可視でした。この問題は、`citus_update_node()` に切り替えることにより解決されます。

- すべての `etcd` ノードが `"failed"` した場合、構成ファイルにリストされている `etcd` ホストをフォールバックとして使用します 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場合があります、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、構成からのノードと、再接続しようとするときに最後の既知のトポロジの組み合わせを使用します。

20.11 バージョン 3.0.1

2023-02-16 リリース

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、プロモーション時に `promoted` ロールを `on_role_change` コールバックスクリプトに誤って渡していました。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。

20.12 バージョン 3.0.0

2023-01-30 リリース

このバージョンでは、[Citius](#) との統合を追加し、プライマリを降格せずに一時的な DCS の停止を乗り切ることができます。

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。次のリリースでは、3.7 より古い Python バージョンのサポートを終了します。
- RAFT サポートは非推奨です。メンテナンスには最善を尽くしますが、発生する可能性のあ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、`primary` を優先して、`master` を削除する最初のステップです。次のメジャーリリースへのアップグレードは、少なくとも 3.0.0 を実行している場合にのみ確実に機能します。

New features

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

この機能が有効になっている場合、Patroni クラスターは一時的な DCS の停止を生き延びることができます。詳細については、[documentation](#) をご覧ください。

- Citius のサポート AlexanderKukushkin, PolinaBungina, JelteFenema

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

Improvements

- 不明な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際の再発エラーの抑制 MichaelBankk

Patroni はこれらのログを書き込みますが、デバッグでのみです。

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

同期レプリケーションが有効になっている場合は、そうではありません。

- 最新の障害データディレクトリのみを保持します。WilliamAlbertusDembo

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

- 同期レプリケーション接続のチェックの向上 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` の削除 AlexanderKukushkin

これを使用する唯一の理由は、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。

20.13 バージョン 2.1.7

2023-01-04 リリース

Bugfixes

- 古い Python モジュールとの小さな非互換性を修正しました。AlexanderKukushkin

彼らは、Debian バスター/Ubuntubionic で Patroni をビルド/実行できませんでした。

20.14 バージョン 2.1.6

2022-12-30 リリース

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 の Dockerfile の例を調整する (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 `search_path=pg_catalog` を適用する 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シグネチャを持つ `public` スキーマのユーザー定義ファンクションおよび/または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思います。そのため、`search_path=pg_catalog` は、Patroni によって作成されたすべての接続レプリケーション接続を除くに適用されます。

- パスワードが `pg_stat_statements` に記録されないようにします FeikeSteenbergen

これは、ユーザーを作成するときに `pg_stat_statements.track_utility=off` を設定することにより実現されます。

Bugfixes

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

これは事実上不要なオプションであるため。

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

RESTAPI 要求にクライアント証明書が使用される場合、Ctl の `insecure` オプションは適切に動作していませんでした。

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

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

- WIN32 で実行可能ファイルを検索するときにファイル拡張子が処理される方法を修正しました。MartínMarqués

拡張子がまだない場合、ファイル名にのみ `.exe` を追加します。

- ConsulTTL セットアップを修正しました。AlexanderKukushkin

HTTP クライアントで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ました。その結果、ConsulTTL が 2 倍オフになりました。

Removed functionality

- `patronictl configure` の削除 PolinaBungina

別個の `patronictl` 構成を作成する必要はありません。

20.15 バージョン 2.1.5

2022-11-28 リリース

このバージョンは、PostgreSQL15 との互換性を強化し、Etcdv3 サポートを実稼働対応として宣言します。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、標準エラー出力にエラーを出力します。

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

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

- モニタリングエンドポイントの HEAD 要求メソッドのサポート RobertCutajar

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

- Windows での動作テストのサポート 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 で正常な Patroni 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 `proxy_url` として DCS のメンバーキーに書き込まれ、サービスディスカバリに使用/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 ループに影響を与え、メンバーキーの有効期限が発生する可能性があります。

- 古いプライマリで `pg_rewrite` 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、かなりの時間ダウンしている場合、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_rewrite` は、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する可能性があります。 `ready` WAL ファイルをアーカイブすることにより、この問題を軽減するだけでなく、一般的に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は 403 エラーを無視します。NickHudson、PolinaBungina

Patroni は、サービスを作成しようとして失敗したことによりログをスパム送信していましたが、実際には既に存在する可能性があります。

- liveness プローブの向上 AlexanderKukushkin

ライブネス問題は、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`` より長く実行されている場合に失敗し始めます。これにより、Kubernetes で `watchdog` の代替として使用できます。

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

以前は、ターゲットを指定せずに手動スイッチオーバーが実行された場合、最新の非同期メンバーがリーダーになる可能性はほとんどありませんでした。

- ブートストラップの実行中のクローン作成を回避する AntsAasma

クラスターブートストラップの実行中は、リーダーをトリガーする必要の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 バージョンによっては、閉じたソケットで `select()` が呼び出される場合、`SequentialThreadingHandler.select()` メソッドは `TypeError` および `IOError` 例外を発生させる場合があります。

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

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

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

内部 API の変更が原因で、クラスター名前ヘッダーが誤った行に表示されました。

Bugfixes

- `Etdresume_grant` の有効期限が切れたトークンを処理する monsterxx03

エラーが発生した場合は、新しいトークンを取得して、要求を再試行します。

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

これは、以前のリリースで導入されたもので、事実上動作しませんでした。

- 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 が存在し空ではないことを定期的にチェックしていますが、ストレージに問題が発生した場合、`os.listdir()` は `OSError` 例外を発生させ、ハートビートループを中断しています。

- 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` を適用する AlexanderKukushkin

ユーザーの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 Citus メンテナンスデーモンなどである可能性があります。

- `postgresql.listen` の `*:<port>` を受け入れます DenisLaxalde

`patroni --validate-config` は、無効であることについて苦情を述べていました。

- Raft のタイムアウトの修正 AlexanderKukushkin

Patroni または `patronictl` が起動すると、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、エラー `"rpcerrorMakingcall:rpcerrorMakingcall:ACLnotfound"` が発生します。

20.16 バージョン 2.1.4

2022-06-01 リリース

New features

- 一般的な Debian/Ubuntu システムでの `pg_rewind` の動作を改善します Gunnar/"Nick"/Bluth

`postgresql.conf` をデータディレクトリの外部に保持する *Postgres セットアップ* たとえば、*Ubuntu/Debian* パッケージでは、``pg_rewind --restore-target-wal`` は `restore_command` の値を理解できません。

- Consul サービスチェックで `TLSServerName` の設定を許可するマイケル・グメリン

チェックが IP によって実行され、Consul `node_name` が FQDN ではない場合に役立ちます。

- ウォッチドッグに `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ウォッチドッグのサポートを修正しました。

- `aws.py` コールバックを `boto` から `boto3` に切り替えました。AlexanderKukushkin

`.x` は 2018 年から放棄され、`python3.9` で失敗します。

- K8s 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 時間で有効期限が切れます。

- `/read-only-sync` モニタリングエンドポイント `Dennis4b` を追加しました

`/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セットアップに構成の不一致がある場合、論理レプリケーションスロットをレプリカにコピーしないでください。AlexanderKukushkin

スロットが `plugin` または `database` 構成オプションと一致しない場合、レプリカはプライマリから論理レプリケーションスロットをコピーしません。以前は、スロットがこれらの構成オプションと一致するかどうかのチェックは、レプリカがスロットをコピーして起動するまで実行されず、不要な再起動が繰り返されていました。

- PostgreSQL12+ のリカバリ構成パラメーターの特別な処理 AlexanderKukushkin

レプリカとしてスタートしている間、パトローニは、`pg_settings` から照会する代わりに現在のパラメーター値をキャッシュすることにより、リーダーアドレスが変更された場合、`postgresql.conf` を更新し、リスタート/リロードできる必要があります。

- `postgresql.listen` パラメーターでの IPv6 アドレスの処理の向上 AlexanderKukushkin

`listen` パラメーターにはポートがあるため、IPv6 アドレスを角かっこ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 以前でのみ発散チェックを実行する場合に `replication` 資格情報を使用する AlexanderKukushkin

`rewind` が有効になっている場合、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` のインポートの欠落を修正しました。WesleyMendes

他のモジュールからインポートされたという理由だけで、テストは失敗していませんでした。

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

特定の場合、パトローニはそれを数値として渡そうとしていました。

- 失敗した `pg_rewind` 試行の処理の向上 AlexanderKukushkin

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

- PostgreSQL の準備ができていない場合は、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問題の処理 AlexanderKukushkin

特定の不明な条件下では、ウォッチャーが古くなる場合があります。その結果、`attempt_to_acquire_leader()` メソッドは 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、最初からリスタートします。

20.17 バージョン 2.1.3

2022-02-18 リリース

New features

- `patronictl` の暗号化された TLS キーのサポートを追加しました AlexanderKukushkin
`ctl.keyfile_password` または `PATRONI_CTL_KEYFILE_PASSWORD` 環境変数を介して構成できます。
- `/metrics` エンドポイントにメトリックを追加しました。AlexandrePereira
具体的には、`patroni_pending_restart` および `patroni_is_paused`。
- スタンバイクラス構成で複数のホストを指定できるようにしますマイケル・バンク

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している場合、PostgreSQLv10 以降の `libpq` で利用可能なクライアントサイドのフェールオーバーに依存するのが良いかもしれません。つまり、スタンバイリーダーの `primary_conninfo` と `pg_rewind` 接続文字列の `target_session_attrs=read-write` を設定します。 `pgpass` ファイルは複数行ホストごとに 1 行で生成され、プライマリクラスターノードで `CHECKPOINT` を呼び出す代わりに、スタンバイクラスターは `pg_control` が更新されるのを待機します。

Stability improvements

- 古い `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る場合でも、`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考えていました。このバグは 2.1.2 リリースで導入されました。
- カスケードスタンバイへのフェイルオーバーのバグを修正しました。AlexanderKukushkin
カスケードスタンバイでどのスロットを作成するかを判断するとき、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 構成バリデータの小さな問題を修正しました。AlexanderKukushkin
`validator.py` で最小値と最大値が引用符で囲まれていたため、PostgreSQLv14 で導入された整数パラメーターは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` は設定されている可能性があります、`rewind_credentials` は定義されておらず、ノード間のスーパーユーザーアクセスは許可されていません。
- RESTAPI 証明書の置換時の"/portinuse/"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 要求との競合状態が発生しました。置換期間中にアクティブなものがある場合、置換はポート使用中エラーでエラーになり、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%文字が含まれる場合、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して DO ブロックを実行しますが、`cursor.execute()` メソッドは渡されたパラメーターを含む空のリストを好みませんでした。
- "/AttributeError:noproperty'leader'"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 コンテンツがワイプアウトされた場合に発生する可能性があります。
- ダイバージェンスタイムラインチェックのバグを修正しました。AlexanderKukushkin
パトローニは、タイムラインが分岐したと誤って仮定していました。pg_rewind の場合、問題は発生しませんでした、pg_rewind が許可されず、`remove_data_directory_on_diverged_timelines` が設定されている場合、以前のリーダーが再初期化される結果になりました。

20.18 バージョン 2.1.2

2021-12-03 リリース

New features

- `psycopg>=3.0` との互換性。AlexanderKukushkin
デフォルトでは、`psycopg2` が優先されます。`psycopg>=3.0` は、``psycopg2`` が利用できない場合、またはその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 に追加します MichaelBank
このフィールドは、クラスターメンバーが DCS と正常に通信した最後に UNIX エポックとしてノートします。これは、ネットワークパーティションを特定および分析するのに役立ちます。
- `pg_controldata` が"/shutdown/"を報告したときにリーダーロックを解放します AlexanderKukushkin
`archive_command` が遅い/障害があ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、`pg_controldata` が `shut down` として PGDATA をきれいに報告し始め、すべての変更を受け取った少なくとも 1 つのレプリカが存在することを確認した直後に、パトローニは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、パトローニはロックを更新し続けます。

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

新しい接続パラメーターは PostgreSQLv14 で導入されました。

- Zookeeper で ZNodes の ACL 設定を許可します。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 Z ノードにデフォルトの ACL を適用するようにします。

Stability improvements

- 次の HA ループまで、次のリカバリーの試行を遅らせます。AlexanderKukushkin

ディスク領域たとえば、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ことにあまりにも熱心にログを氾濫させています。

- 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

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

- `"/!am/"`ステータスメッセージを改善しました MichaelBank

no action. I am a secondary ($\{0\}$) vs no action. I am ($\{0\}$), a secondary

- `wal_keep_size` に変換するときに `int wal_keep_segments` にキャストするようになりました。
- JorgeSolórzano

グローバル wal_keep_segments の文字列として wal_keep_segments を指定することが可能であり、Python は動的型付け言語であるため、文字列は単に乗算されました。例 wal_keep_segments: /"100/" は 100100100100100100100100100100100100100100100100100100100MB に変換されました。

- 同期レプリケーションが有効になっている場合、同期ノードへのスイッチオーバーのみを許可します
- AlexanderKukushkin

それに加えて、リーダーは既知の同期ノードに対してのみレースを行います。

- Postgres が遅い場合に、キャッシュされたロールをフォールバックとして使用する AlexanderKukushkin

一部の極端なケースでは、Postgres が非常に遅いため、通常の監視クエリが数秒で終了しない場合があります。 `statement_timeout` 例外が適切に処理されない場合、リーダーキーの有効期限が切れた場合、または更新が失敗したときに Postgres が時間通りに降格されない状況が発生する可能性があります。このような例外が発生した場合、Patroni はキャッシュされた `role` を使用して、Postgres がプライマリとして実行されているかどうかを判断します。

- メンバー ZNode の不必要な更新を回避します。AlexanderKukushkin

メンバーデータの値が変更されていない場合、更新は発生しないはずです。

- プロモート後のチェックポイントの最適化 AlexanderKukushkin

最新のタイムラインが既に `pg_control` に保存されている場合は、CHECKPOINT の実行を避けてください。 `initdb` で新しいクラスターを初期化した直後に、不必要な CHECKPOINT を回避するのに役立ちます。

- 同期ノードを選択するときに `nofailover` のないメンバーを優先する AlexanderKukushkin

以前の同期ノードはレプリケーションラグにのみ基づいて選択されていたため、`nofailover` タグが付いているノードは他のノードと同じ同期になる機会がありました。プライマリに障害が発生した場合、フェールオーバーは自動的に発生しないため、この動作は混乱を招くと同時に危険でした。

- `etcd` マシンキャッシュから重複ホストを削除します MichaelBank

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

Bugfixes

- スロット管理中に一時的なレプリケーションスロットをスキップする AlexanderKukushkin

v10 から `pg_basebackup` は、WAL ストリーミング用の一時的なレプリケーションスロットを作成します。パトローニは、スロット名が不明に見えるため、それをドロップしようとしていました。それを修正するために、`pg_stat_replication_slots` ビューを照会するとき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ことを確認します AlexanderKukushkin

この場合、パトローニはデフォルトの `statement_timeout` を使用していましたが、呼び出しが失敗すると、回復しない可能性が非常に高いため、`pg_wal` および `pg_catalog` のサイズが増加します。

- `/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` に書き込まれます。

- 降格時のハンドル DCS 例外 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()` 呼び出しが例外が発生する場合があります。適切に処理されない場合、DCS が回復するまで Postgres は停止したままになります。

- `use_unix_socket_repl` は場合によっては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。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 のいくつかの問題を修正しました AlexanderKukushkin

`clusters_unlocked` は定義できない場合があります、`GET /metrics` エンドポイントで例外が発生しました。それに加えて、エラー処理方法は、`connect_address` タプルに常に2つの要素があることを前提としていましたが、実際には、IPv6 の場合にはもっと多くの要素がある可能性があります。

- 巻き戻しを決定する前に、新しく昇格したノードの復旧が完了するのを待ってください AlexanderKukushkin

実際のプロモートが発生し、新しいタイムラインが作成されるまでに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ときに履歴ファイルの欠落したタイムラインを処理する AlexanderKukushkin

現在のレプリカのタイムラインがプライマリの履歴ファイルにない場合、レプリカはリワインドが必要でないという誤った想定をしていました。

20.19 バージョン 2.1.1

2021-08-19 リリース

New features

- ETCDSRV 名前サフィックスのサポート DavidPavlicek

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

- 新しいリーダー huiyalin525 で歴史を豊かにします

新しい列を `patronictl history` 出力に追加します。

- クラスター内の Kubernetes 構成用に CA バンドルを構成可能にします AronParsons

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

- Consul サービスとしての動的な登録/登録解除とタグの変更をサポート 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、ディスクで変更された場合に RESTAPI 証明書をリロードする機能を追加しました。残念ながら、リロードは開始直後に無条件で発生していました。

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

Patroni は、起動時に、メンバーのリストを照会することにより、Etcd クラスターの正常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すが、これはプロキシを介して Etcd で作業する場合に必要ではなく、不要な警告を発生させていました。

- `pg_stat_replication` の NULL 値を含む行をスキップする AlexanderKukushkin

`pg_stat_replication` ビューは、`state = 'streaming'` の場合でも、`replay_lsn`、`flush_lsn`、または `write_lsn` フィールドに NULL 値を含めることができます。

20.20 バージョン 2.1.0

2021-07-06 リリース

このバージョンでは、PostgreSQLv14 との互換性を追加し、フェールオーバー/スイッチオーバーを生き抜くための論理レプリケーションスロットを作成し、RESTAPI のホワイトリストのサポートを実装し、ログ数をハートビートごとに 1 行に削減します。

New features

- PostgreSQLv14 との互換性 AlexanderKukushkin

Patroni が `/"pause/"` モード自分自身にない場合、WAL 再生の一時停止を解除します。プライマリの `max_connections` などの特定のパラメーターの変更により、`/"paused/"` になる可能性があります。

- フェールオーバー論理スロット AlexanderKukushkin

PostgreSQLv11 以降で論理レプリケーションスロットがフェールオーバー/スイッチオーバーを生き残るようにします。レプリケーションスロットは、リスタートでプライマリからレプリカにコピーされた場合、後で `pg_replication_slot_advance()` _ファンクションを使用して前方に移動します。その結果、スロットはフェールオーバー前に既に存在し、イベントが失われることはありませんが、一部のイベントは複数回配信される可能性があります。

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

構成されている場合、ルールと一致する IP のみが安全でないエンドポイントを呼び出すことができます。それに加えて、クラスタのメンバーの IP をリストに自動的に含めることができます。

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

以前、Patroni はレプリケーション接続に常に TCP を使用していましたが、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: と合わせて、`patronictl list` 出力および RESTAPI で表示されるカスタムタグを指定できます。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` と同じメトリックを公開するエンドポイント。

- Patroni ログのおしゃべり性の低下 AlexanderKukushkin

すべてが正常な場合、HA ループの実行ごとに 1 行のみが書き込まれます。

Breaking changes

- 古い `permanent logical replication slots` 機能は、PostgreSQLv10 以前では動作しなくなりました。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ないこと、したがって無効になる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 を結果ます。AlexanderKukushkin

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

Improvements in Raft support

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

PySyncObj のさまざまな問題のため、暗号化のサポートは非常に不安定でした

- Raft 実装での 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 回のみ効果的に解決を行っていました。同じノードが別の IP でオンラインに戻ったときに問題が発生していました。

Stability improvements

- `psycopg2-2.9+` との互換性。AlexanderKukushkin

`psycopg2` 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 での過剰な HA ループの実行を修正しました。AlexanderKukushkin

メンバー Z ノードの更新は連鎖反応を引き起こし、その結果、HA ループが連続して複数回実行されました。

- 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 証明書ファイルがインプレースで更新された場合、Patroni はリロードを実行しませんでした。

- kerberos 認証が使用される場合、`pgpass` ディレクトリを作成しないでください KostiantynNemchenko

Kerberos とパスワード認証は相互に排他的です。

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

PITR を実行する場合にのみ `hot_standby=off` で Postgres を起動し、PITR が完了したら再起動します。

Bugfixes

- `kazoo-2.7+` との互換性。AlexanderKukushkin

Patroni は自分自身で再試行を処理しているため、使用可能な接続がない場合に Zookeeper クラスターへの要求がすぐに破棄される `kazoo` の古い動作に依存しています。

- プロキシを介して接続していることが知られている場合、Etcdv3 クラスターのバージョンを明示的に要求します AlexanderKukushkin

Patroni は gPRC ゲートウェイを介して Etcdv3 クラスターと連携しており、クラスターバージョンに応じて、異なるエンドポイント `/v3`、`/v3beta`、または `/v3alpha` を使用する必要があります。バージョンはクラスタートポロジとともにのみ解決されましたが、プロキシを介して接続するときに後者は決して実行されないため。

20.21 バージョン 2.0.2

2021-02-22 リリース

New features

- 外部管理対象レプリケーションスロットを無視する機能 JamesColeman

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

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

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

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

- RESTAPI 認証資格情報の定数時間比較 AlexBrasetvik

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

- 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

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

Stability improvements

- カスタムブートストラップを行うときに `hot_standby = off` で postgres を開始する IgorYanchenko

カスタムブートストラップ中に、Patroni はベースバックアップを復元し、Postgres を起動し、復旧が完了するまで待機しています。スタンバイの一部の PostgreSQL パラメーターは、プライマリより小さくすることはできません。新しい値 WAL から復元され、構成された値より大きい場合、Postgres はパニックして停止します。このような動作を回避するために、`hot_standby` モードを使用せずにカスタムブートストラップを実行します。

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

ウォッチドッグデバイスが書き込み可能でない場合、または必須モードで欠落している場合、メンバーは昇格できません。この誤った構成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のリカバリーの詳細度の向上 AlexanderKukushkin

PostgreSQL が明確にシャットダウンされていないことに Patroni が気づいた場合、特定の状況、シングルユーザーモードで Postgres を起動することによりクラッシュリカバリーが実行されます。ディスク上の領域の不足などにより、復旧に失敗した場合がありますが、エラーは飲み込まれました。

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

古き良き `python-consul` は数年以来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` を実行するときに `bypass_api_service` を使用しないでください AlexanderKukushkin
K8s ポッドが非 `default` 名前空間で実行されている場合、`kubernetes` エンドポイントを照会するための十分な権限があるとは限りません。この場合、Patroni は警告を表示し、`bypass_api_service` 設定を無視します。`patronictl` の場合、警告は少し迷惑でした。
- `raft.data_dir` が存在しない場合は作成するか、書き込み可能であることを確認します。MarkMercado
使いやすさと使いやすさが向上します。

Bugfixes

- 一時停止中にリーダーロックを失った場合、再起動またはプロモーションを中断しないでください AlexanderKukushkin
一時停止中に、ロックなしでプライマリとして `postgres` を実行できます。
- 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延するために、Patroni は `HTTPServer` のいくつかのメソッドをオーバーライドします。`shutdown_request()` メソッドは忘れられました。
- Zookeeper 使用時のスリープ時間の問題を修正しました。AlexanderKukushkin
HA コードの実行の間に、パトローニが最大 2 倍長く眠っていた可能性があります。
- ブートストラップが失敗した後にデータディレクトリを移動するときの無効な `os.symlink()` 呼び出しを修正しました。AndrewL'Ecuyer
ブートストラップが失敗した場合、Patroni はデータディレクトリ、`pg_wal`、およびすべてのテーブルスペースの名前を変更しています。その後、シンボリックリンクを更新して、ファイルシステムの一貫性を保ちます。`src` および `dst` 引数がスワップされたため、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
スーパーユーザーパスワードが構成されていない場合、Patroni は `post_init` スクリプトの呼び出しに失敗し、したがって、ブートストラップ全体が失敗していました。
- スタンバイクラスターの `pg_rewind` に関する問題を修正しました。AlexanderKukushkin
スーパーユーザー名が `Postgres` と異なる場合、接続文字列にデータベース名が含まれていないため、スタンバイクラスターの `pg_rewind` は障害が発生していました。
- `Etcdv3` での認証が明示的に失敗した場合にのみ終了します AlexanderKukushkin
起動時に、Patroni は `Etcd` クラスタトポロジの検出を実行し、必要に応じて認証します。`etcd` サーバーの 1 つがアクセスできない場合があり、パトローニはこのサーバーで認証を実行しようとしたましたが、次のノードで再試行する代わりに失敗しました。
- 空のリストを返す `psutilcmdline()` でケースを処理します。AlexanderKukushkin
ゾンビプロセスはまだポストマスターの子ですが、`cmdline()` がありません。
- `PATRONI_KUBERNETES_USE_ENDPOINTS` 環境変数をブール値として扱います AlexanderKukushkin
そうしないと、環境を介して `kubernetes.use_endpoints` を無効にすることが不可能でした。

- 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照会し、現在のポッドがリーダーロックをまだ保持していることを確認して、更新を繰り返します。

20.22 バージョン 2.0.1

リリース 2020-10-01

New features

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

Windows、`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 の背後で実行されている場合に役立つ場合があります。`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` 接続パラメーターのサポートを追加しました KostiantynNemchenko

Connection パラメーターは PostgreSQL13 で導入されました。

Stability improvements

- 一時停止中の動作を変更しました AlexanderKukushkin

1. PGDATA ディレクトリが見つからない/空の場合、Patroni は `bootstrap` メソッドを呼び出しません。
2. Patroni は一時停止中に `sysid` 不一致でも終了せず、警告のみをログに記録します。
3. Postgres がリカバリーでなく書き込みを受け入れて実行されていないが、`sysid` が初期化キーと一致しない場合、ノードは一時停止モードでリーダーキーを取得しようとしません。

- クラッシュリカバリーを実行するときに `master_start_timeout` を適用する AlexanderKukushkin

Postgres がリーダーノードでクラッシュした場合、Patroni はシングルユーザーモードで Postgres を起動することによりクラッシュリカバリーを実行します。クラッシュリカバリー中に、リーダーロックが更新されます。クラッシュリカバリーが `master_start_timeout` 秒以内に終了しなかった場合、Patroni は強制的に停止し、リーダーロックを解放します。

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

そこに追加する唯一の理由は、Python2.7 の `ipaddress` 依存関係でした。

Bugfixes

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

リーダーオブジェクトの更新が失敗した場合、未処理の例外がプライマリの降格を妨げていました。

- RAFT が使用されているときにハングする `patronictl` を修正しました。AlexanderKukushkin

Patroni 構成で `patronictl` を使用する場合、`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` の値の取得に失敗していたため、`pg_rewind` の欠落している WAL の取得は機能しませんでした。

20.23 バージョン 2.0.0

2020-09-02 リリース

このバージョンは、PostgreSQL13 との互換性を強化し、複数の同期スタンバイのサポートを追加し、`pg_rewind` の処理が大幅に向上し、純粋な RAFT、Etcd、Consul、または Zookeeper なしで Etcdv3 および Patroni のサポートを追加し、オプションで呼び出しが可能です `pre_promote` フェンシングスクリプト。

PostgreSQL 13 support

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

`standby_leader` に昇格するときに、`primary_conninfo` を変更し、ロールを更新し、Postgres をリロードします。 `on_role_change` および `on_reload` は実質的に互いに重複するため、Patroni は `on_role_change` のみを呼び出します。

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

PostgreSQL12 では、`gssencmode` および `channel_binding` 接続パラメーターを導入し、`postgresql.authentication` セクションで定義された場合に使用できるようになりました。

- `wal_keep_segments` から `wal_keep_size` へのハンドル名前変更 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 `--restore-target-wal` とともに `pg_rewind` を使用します AlexanderKukushkin

PostgreSQL13 では、Patroni は `restore_command` が構成されているかどうかを確認し、`pg_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 でのパトローニ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に依存せずに Patroni を実行できます。HA の場合、3 つの Patroni ノード、または Patroni で 2 つのノードと 1 つのノードで `patroni_raft_controller` 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 ゲートウェイを介した 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 ではデフォルトで v2 が無効になっています。また、v2 が Etcd から完全に削除される可能性があるため、Patroni に Etcdv3 のサポートを実装しました。それを使用し始めるには、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でクラスターを実行できます。同期レプリカの最大数は、新しいパラメーター `synchronous_node_count` によって制御されます。デフォルトで 1 に設定され、`synchronous_mode` が `off` に設定されている場合、効果はありません。

- `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 ファイルは、アルファベット順にロードおよび適用されます。

- PostgreSQL パラメーターの高度な検証 AlexanderKukushkin

特定のパラメーターが現在の PostgreSQL バージョンでサポートされていない場合、またはその値が正しくない場合、Patroni はパラメーターを完全に削除するか、値を修正しようとします。

- プロモート後の強制チェックポイントが完了したときにメインスレッドを起動します AlexanderKukushkin

レプリカは、DCS のリーダーのメンバーキーを介してチェックポイントの指示を待機しています。キーは通常、HA ループごとに 1 回のみ更新されます。メインスレッドをウェイクアップしないと、レプリカは必要よりも長く `loop_wait` 秒まで待機する必要があります。

- 9.6+での `pg_stat_wal_recevier` ビューの使用 AlexanderKukushkin

ビューには、`primary_conninfo` および `primary_slot_name` の最新の値が含まれていますが、`recovery.conf` の内容は古い可能性があります。

- Patroni 構成ファイルでの IPv6 アドレスの処理の改善 MateuszKowalski

IPv6 アドレスは角かっこで囲まれることになっていますが、パトローニはそれをプレーンに取得することを期待していました。両方の形式がサポートされるようになりました。

- Consul `service_tags` 設定パラメーターを追加しました。RobertEdström

これらは、ロードバランサーなどによる動的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方法の `no_params` オプションを実装しました。KostiantynNemchenko

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

- `init` に失敗した後に WAL とテーブルスペースを移動する FeikeSteenbergen

`reinit` を実行したときに、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既に削除していました。 `move_data_directory()` メソッドは、同様のジョブ、つまり WAL ディレクトリとテーブルスペースの名前を変更し、PGDATA のシンボリックリンクを更新します。

Improved in `pg_rewind` support

- タイムライン発散チェックの改善 AlexanderKukushkin

レプリカ上の再生された場所がスイッチポイントの前でない場合、または元のプライマリのチェックポイントレコードの最後がスイッチポイントと同じである場合、巻き戻す必要はありません。チェックポイントレコードの最後を取得するために、 `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ている場合は、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` は分岐ポイントの前にチェックポイントの場所を見つけることができません。PostgreSQL13 以降、`pg_rewind` は、不足している WAL を取得するために `restore_command` を使用できます。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、自分自身で `restore_command` を呼び出すことにより欠落している 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分離されているため、リーダー選挙とタイムラインのスイッチについてすぐには認識しません。ファクトを検出するために、 `standby_leader` は、`pg_wal` 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および美化します AlexanderKukushkin

Patroni が `pg_rewind` の必要性を判断しようとしている場合、プライマリから履歴ファイルの内容をログに書き込む可能性があります。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成長し、最終的にはあまりにも多くの行を占有し始めますが、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に 3 行と後に 2 行のみを表示します。

Improvements on K8s

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

公式の `Pythonkubernetes` クライアントには、大量の自動生成コードが含まれているため、非常に重くなっています。Patroni は K8sAPI エンドポイントのごく一部のみを使用し、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 で実行されている場合、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、これは、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

ブし、K8s マスターノードに直接接続することにより、より良い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` に増加しました。

- K8 のサブセットアドレスに `references` および `nodename` を設定します。AlexanderKukushkin

一部のロードバランサーはこの情報に依存しています。

- `update_leader()` で考えられる競合状態を修正しました。AlexanderKukushkin

パトロニの外部で発生するリーダー構成マップまたはエンドポイントの同時更新により、`update_leader()` 呼び出しが失敗する可能性があります。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。

- 存在しない構成のパッチの適用を明示的に禁止します 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ていますが、Kubernetes では、喜んで構成アノテーションを作成し、ブートストラップが完了した後にブートストラップ構成の書き込みを防止していました。

- `pause` のバグを修正しました。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、Postgres を再起動していましたが、何もしないはずでした。

Improvements in REST API

- ワーカースレッドが開始されるまで TLS ハンドシェイクを遅延します。AlexanderKukushkin, BenHarris

TLS ハンドシェイクが API スレッドで実行され、クライアントサイド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 のクライアント証明書とは別に `basic-auth` をチェックします 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、完全に有効なユースケースです。

- OPTIONS 要求の HTTP ヘッダーの後にダブル CRLF を書き込みます。SergeyBurladyan

HAProxy は単一の CRLF で満足していましたが、Consul のヘルスチェックは、切断された接続と予期しない EOF について不平を言いました。

- `GET /cluster` は、ZookeeperAlexanderKukushkin の古いメンバー情報を表示していました

エンドポイントは、Patroni 内部クラスタービューを使用していました。Patroni 自分自身の場合、それは問題を引き起こしませんでした。外部に公開される場合、最新の情報、特にレプリケーションラグを表示する必要があります。

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

マスターの `GET /standby-leader` と `standby_leader` の `GET /master` は、誤って 200 で応答していました。

- `DELETE /switchover` を実装しました AlexanderKukushkin

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

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

これらは、K8s サービスがラベルセクターで使用される場合、サブセットアドレスから `/"unhealthy"` ポッドを削除するのに役立つ可能性があります。

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

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

- RESTAPI 応答でのユーザー定義の HTTP ヘッダーのサポートを追加しました YogeshSharma

この機能は、ブラウザから要求が行われた場合に役立つ場合があります。

Improvements in patronictl

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

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

- メンバー `conn_url` が見つからない場合の処理 AlexanderKukushkin

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

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

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

- `patronictl flush switchover` の実装 AlexanderKukushkin

それまでは `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、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 の例外処理の向上 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` を適用する AlexanderKukushkin

Patroni はユーザー `replication`、`rewind` を作成するときに既にそれを行っていましたが、`post_init` の場合にそれを欠いていたのは見落としでした。その結果、スクリプトが自分自身で内部的に実行していない場合、`synchronous_mode` のブートストラップは完了できませんでした。

- Consul プールマネージャー ponvenkates の `maxsize` の増加

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。

- パトローニは Postgres が実行されていると誤って報告していました AlexanderKukushkin

たとえば、Postgres がディスク不足エラーのために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 に入れます AlexanderKukushkin

たとえば `standby_cluster.port` が指定されていない場合、`pgpass` ファイルは誤って生成されました。

- 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する KrishnaSarabu

Patroni は、名前に `'` などの特殊文字が含まれている場合、リーダーノードの休止スロット `slots` 定義時を作成しているようですたとえば、`/abc-us-1/` の場合。

- カスタムブートストラップで存在しない `pg_hba.conf` の削除を回避します KrishnaSarabu

カスタムブートストラップ後に `pg_hba.conf` がたまたま `pgdata` ディレクトリの外部にある場合、Patroni は失敗していました。

20.24 バージョン 1.6.5

2020-08-23 リリース

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、Patroni は stop オペレーションが `master_stop_timeout` で設定された値を超えて実行されている場合に 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ときにプライマリが WAL セグメントをリサイクルするのは一般的な問題です。これで、固定数のノードと名前が変更されない静的クラスターに適したソリューションができました。 `slots` のすべてのノードの名前をリストする必要があるため、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します。

- ConfigValidator の最初のドラフト IgorYanchenko

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

- タイムライン履歴の最大長を構成する機能 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、最後の数行のみが関心があります。 `max_timelines_history` パラメーターを使用すると、DCS に保持するタイムライン履歴項目の最大数を指定できます。

- Kazoo2.7.0 互換性 DanyalProut

Kazoo の一部の非パブリックメソッドはシグネチャを変更しましたが、パトローニ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するコスティアンティン・ネムチェンコ、アレクサンダー・ククシュキン
タグはノードごとに個別に構成され、それらの概要を取得する簡単な方法はありませんでした
- メンバーの成果の向上 AlexanderKukushkin

冗長クラスター名はすべての行に表示され、テーブルヘッダーにのみ表示されます。

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

- 構成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に失敗する KaarelMoppel

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

- 初期化されていない K8s ポッドが保護者を破壊する問題を解決しました。AlexanderKukushkin

Patroni は、K8 の特定のポッドアノテーションに依存しています。Patroni ポッドのいずれかが停止または起動している場合、有効なアノテーションはまだなく、`patronictl` は例外で失敗していました。

Stability improvements

- K8sAPI サーバーへの LIST 呼び出しが失敗した場合に 1 秒のバックオフを適用する AlexanderKukushkin
これは主に、ログの氾濫を回避するために必要ですが、メインスレッドのスタベーションを防ぐことにも役立ちます。
- 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 サーバーが要求で圧倒されている場合、再試行を求める場合があります。

- ポストマスターから KUBERNETES_ 環境をスクラブする FeikeSteenbergen

KUBERNETES_ 環境変数は PostgreSQL には必要ありませんが、ポストマスターに公開すると、バックエンドおよび通常のデータベースユーザーたとえば pl/perl を使用して公開されます。

- 再初期化時にテーブルスペースをクリーンアップする 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。この問題の以前の回避策は、PGDATA スクリプトを実装することでした。

- プロモートが発生した後に CHECKPOINT を明示的に実行するようにしました AlexanderKukushkin

新しいプライマリが pg_rewind で使用できるようになるまでの時間を短縮するのに役立ちます。

- etcd メンバーのスマートリフレッシュ AlexanderKukushkin

Patroni が Etcd クラスターのすべてのメンバーで要求の実行に失敗した場合、Patroni は、次回再試行する前に IP/ホストの変更が A または SRV レコードを再チェックします。

- pg_controldata の欠損値をスキップする (FeikeSteenbergen)

PGDATA と一致しないバージョンのバイナリを使用しようとする、値が欠落します。Patroni はとにかく Postgres を起動しようとします、そして Postgres はメジャーバージョンが一致しないと不平を言い、エラーで中断します。

Bugfixes

- 必要に応じて Consul の SSL 検証を無効にします。JulienRiou

urllib3 の特定のバージョンから、SSL 検証を効果的に無効にするには、cert_reqs を明示的に ssl.CERT_NONE に設定する必要があります。

- HA ループのすべてのサイクルでレプリケーション接続を開くことを避けます。AlexanderKukushkin

回帰は 1.6.4 で導入されました。

- 障害が発生したプライマリで on_role_change コールバックを呼び出します AlexanderKukushkin

特定の場合では、仮想 IP が古いプライマリに接続されたままになる場合があります。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、リワインド状態をリセットする 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

12 より前の PostgreSQL で recovery_min_apply_delay が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) します。multiprocessing の fork ではなく spawn メソッドに切り替えたときに、互換性が壊れました。

20.25 バージョン 1.6.4

2020-01-27 リリース

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 で書き換えられます
 2. `pg_ctl kill` は、非 `posix` システムで `postgres` を停止するために使用されます
 3. Unix ドメインソケットを使用しないでください

Stability improvements

- `unix_socket_directories` および `stats_temp_directory` が存在することを確認します IgorYanchenko

Patroni と Postgres の起動時に、`unix_socket_directories` および `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てください。Patroni は、作成に失敗すると終了します。
- `postgresql.pgpass` がパトローニが書き込みアクセスがある場所にあることを確認します。IgorYanchenko

書き込みアクセスがない場合、Patroni は例外で終了します。
- デフォルトで Consul `serfHealth` チェックを無効にします KostiantynNemchenko

ネットワークに小さな問題が発生した場合でも、`serfHealth` の障害は、ノードに関連付けられているすべてのセッションの無効化につながります。したがって、リーダーキーは `ttl` よりもはるかに早く失われ、レプリカの不要な再起動と、場合によってはプライマリの降格が発生します。
- K8sAPI への接続の TCP キープアライブを構成する AlexanderKukushkin

TTL 秒後にソケットから何も取得しない場合、デッドであると考えられます。
- ユーザー作成時にパスワードのロギングを回避する AlexanderKukushkin

パスワードが拒否されるか、ロギング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。それを回避するために、Patroni はユーザーを作成/更新しようとする前に、`log_statement`、`log_min_duration_statement`、および `log_min_error_statement` をいくつかの安全な値に変更します。

Bugfixes

- カスケードレプリカで `standby_cluster` 構成の `restore_command` を使用する AlexanderKukushkin

`standby_leader` は、機能が存在する最初から既に行っていました。レプリカで同じことをしないと、スタンバイリーダーに追いつけない可能性があります。
- スタンバイクラスターによって報告された更新タイムライン AlexanderKukushkin

タイムラインスイッチの場合、スタンバイクラスターはプライマリから正しく複製していましたが、`patronictl` は古いタイムラインを報告していました。

- `custom_conf` で特定のリカバリパラメーターを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ーの検証を行うとき、パトロニは、パトロニ構成で定義されていないが、`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、`archive_cleanup_command`、`promote_trigger_file`、`recovery_end_command`、`recovery_min_apply_delay`、および `restore_command` をスキップします。

- 名前にピリオドを含む `postgresql` パラメーターの処理を改善しました AlexanderKukushkin

このようなパラメーターは、ユニットが必ずしも文字列ではない拡張機能によって定義できます。値の変更には、再起動が必要になる場合がありますたとえば `pg_stat_statements.max`。

- シャットダウン中の例外処理を改善しました。AlexanderKukushkin

シャットダウン中に、Patroni は DCS 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、例外が発生する場合があります。例外処理の欠如が、ログスレッドの停止を妨げていました。

20.26 バージョン 1.6.3

2019-12-05 リリース

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` ～ `pg_basebackup` およびカスタムレプリカ作成方法で指定された接続パラメーターを適用します。AlexanderKukushkin

彼らは URL のような接続文字列に依存しているため、パラメーターは適用されません。

20.27 バージョン 1.6.2

2019-12-05 リリース

New features

- `patroni --version` を実装しましたイゴール・ヤンチェンコ

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 要求の `user-agent` http ヘッダーを設定します AlexanderKukushkin

Patroni は、http プロトコルを介して Consul、Etcd、および KubernetesAPI と通信しています。特別に作られた `user-agent` 例 `Patroni/1.6.2 Python/3.6.8 Linux` があると、デバッグとモニタリングに役立つ場合があります。

- 例外トレースバックのログレベルを構成できるようにします IgorYanchenko

`log.traceback_level=DEBUG` を設定すると、トレースバックは `log.level=DEBUG` の場合にのみ可視されます。デフォルトの動作は同じままです。

Stability improvements

- 構成ファイルに必要なモジュールを検索するときに、すべての DCS モジュールのインポートを回避します。AlexanderKukushkin

たとえば、ファイルだけがが必要な場合、Etcd、Consul、および Kubernetes のモジュールをインポートする必要はありません。zookeeper。メモリ使用量を削減し、INFO メッセージ `Failed to import smth` の問題を解決します。

- 明示的な要件から `python requests` モジュールを削除しました AlexanderKukushkin

これは重要な目的には使用されませんでしたが、`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発生させました。

- YAML 配列の代わりにコンマ区切りの文字列として書き込まれた `etcd.hosts` の処理を改善しました。IgorYanchenko

以前は、`host1:port1, host2:port2` フォーマットで書き込まれると失敗していましたコンマの後のスペース文字。

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、例外を発生させます。

- RESTAPI がバインドできない場合にエラーメッセージをもっと役立つようにしました 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悪いのかを理解するのは難しい場合があります。

Bugfixes

- `wal_buffers` の計算を修正しました。AlexanderKukushkin

PostgreSQL11 では、基本単位が 8kB ブロックからバイトに変更されました。

- PostgreSQL10+でのみ `primary_conninfo` で `passfile` を使用する AlexanderKukushkin

古いバージョンでは、最新バージョンの `libpq` がインストールされていない限り、`passfile` が動作する保証はありません。

20.28 バージョン 1.6.1

2019-11-15 リリース

New features

- `PATRONICTL_CONFIG_FILE` 環境変数 `msvechla` を追加しました
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を表示します。
- `pg_rewind` を実行するときに `PGOPTIONS` で `-c statement_timeout=0` を渡すようにしました。AlexanderKukushkin
サーバー上の `statement_timeout` が小さな値に設定され、`pg_rewind` によって実行されたステートメントのいずれかがキャンセルされた場合から保護します。
- PostgreSQL 構成のより低い値を許可します Soulou
Patroni は、一部の PostgreSQL 構成パラメーターを一部のハードコードされた値より小さく設定することを許可していませんでした。最小許可値が小さくなりました、デフォルト値は変更されていません。
- 証明書ベースの認証を許可します JonathanS.Katz
この機能は、スーパーユーザー、レプリケーション、リワインドアカウントの証明書ベースの認証を有効にし、ユーザーは接続する `sslmode` を指定できます。
- パスワードの代わりに `primary_conninfo` で `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
- 構成の変更に関係なく `pg_ctl reload` を実行します AlexanderKukushkin
一部の構成ファイルは Patroni によって制御されていない可能性があります。誰かが RESTAPI を介してまたは 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、通常は Postgres もリロードされると予想されます。以前は、Patroni 構成の `postgresql` セクションに変更がない場合、これは発生しませんでした。
- `primary_conninfo` だけでなくすべての復旧パラメーターを比較します AlexanderKukushkin
以前は、`check_recovery_conf()` メソッドは `primary_conninfo` が変更されたかどうかをチェックするだけで、他のすべてのリカバリパラメーターは考慮されませんでした。
- 再起動せずにいくつかの回復パラメーターを適用できるようにします。AlexanderKukushkin
PostgreSQL12 から、次のリカバリパラメーターは、再起動せずに変更できます。
`archive_cleanup_command`、`promote_trigger_file`、`recovery_end_command`、および `recovery_min_apply_delay`。将来の Postgres リリースでは、このリストは拡張され、Patroni はそれを自動的にサポートします。
- `use_slots` をオンラインで変更できるようにしました AlexanderKukushkin
以前は、Patroni を再起動し、スロットを手動で削除する必要がありました。

- Postgres の起動時に `PATRONI_` 接頭辞の環境変数のみを削除する CodyCoons

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を解決します。

Stability improvements

- K8sAPI を使用する場合は LIST+WATCH を使用します。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、K8s マスターノードのストレスが軽減されます。

- ブートストラップ中に PGDATA が空でない場合のワークフローを改善しました。AlexanderKukushkin

`initdb` ソースコードによると、`lost+found` および `.dotfiles` のみが存在する場合、PGDATA が空であると見なす場合があります。今、パトロニも同じことをしています。PGDATA がたまたま空でなく、同時に `pg_controldata` の観点から有効でない場合、Patroni は文句を言って終了します。

- すべての HA ループで高価な `os.listdir()` の呼び出しを回避します AlexanderKukushkin

システムが IO ストレス下にある場合、`os.listdir()` の実行に数秒または数分かかる場合があります、パトロニの HA ループに悪影響を与える可能性があります。これにより、更新がないためにリーダーキーが DCS から消失する可能性もあります。PGDATA が空ではないことを確認するための、より適切で安価な方法があります。次に、PGDATA に `global/pg_control` ファイルの存在を確認します。

- ログインフラストラクチャの一部の改善 AlexanderKukushkin

以前は、ロギングスレッドが `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行が失われる可能性がありました。

- Python3.4+で `spawn` マルチプロセッシングスタートメソッドを使用する MaciejKowalczyk

Python では、スレッド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 [issue _](#)。デフォルトの方法 `fork` から `spawn` への切り替えが推奨される回避策です。そうしないと、Postgres が実際に起動して実行されているにもかかわらず、Postmaster 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す 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` に設定されている場合、クライアント証明書は、安全でないすべての RESTAPI エンドポイントがチェックされます。

- Postgres が実行されていない場合、`GET /replica` ヘルスチェック要求の応答コード 503 を返します。AlexanderAnikin

Postgres は、クライアント接続の受け入れを開始する前の復旧にかなりの時間を費やす場合があります。

- `/history` および `/cluster` エンドポイントを実装します AlexanderKukushkin

`/history` エンドポイントは、DCS の `history` キーのコンテンツを表示します。 `/cluster` エンドポイントは、すべてのクラスターメンバーと、保留中およびスケジュールされた再起動またはスイッチオーバーなどのサービス情報を表示します。

Improvements in Etcd support

- EtcdRAFT 内部エラーでの再試行 AlexanderKukushkin

Etcd ノードはシャットダウンされるときに、`response code=300, data='etcdserver: server stopped'` を送信します。これが、Patroni がプライマリを降格する原因となっていました。

- Etcd 要求の再試行を早めにあきらめないでください AlexanderKukushkin

いくつかのネットワーク問題が発生すると、パトローニはすぐに Etcd ノードのリストを使い果たし、`retry_timeout` 全体を使用せずに断念しました。その結果、プライマリが降格する可能性があります。

Bugfixes

- `pg_rewind` ユーザーに実行権限を付与するときに `synchronous_commit` を無効にする kremius

`synchronous_mode_strict: true` でブートストラップが行われる場合、`GRANTEXECUTE` ステートメントは、使用可能な非同期ノードのため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 要求を処理し、Python3.7 はデフォルトで非デーモン以外のすべての要求に対してスレッドを生成します。

- 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 を復旧しようとするにより、`patronictl reinit --force` が上書きされる可能性があります。これは、ベースバックアップの実行中にパトローニが Postgres を起動しようとした場合に発生しました。

- `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、別のカーソルオブジェクトを作成する必要があります。

- 大文字を含む名前を持つ同期スタンバイを昇格させない問題を修正しました。AlexanderKukushkin

Postgres は、`application_name` と `synchronous_standby_names` の値を比較するときに同じことをしていたため、名前を小文字に変換しました。

- 新しいものを開始する前に、コールバックプロセスとともにすべての子を `kill` 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困難になり、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。

- 'startfailed' 問題を修正しました AlexanderKukushkin

特定の条件下では、Postgres が起動して実行されているにもかかわらず、Postgres の状態が「startfailed」に設定される場合があります。

20.29 バージョン 1.6.0

2019-08-05 リリース

このバージョンでは、PostgreSQL12 との互換性を追加し、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し、IPv6 サポートを有効にします。

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 から `psycopg2` は、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 2 つの異なるパッケージ `psycopg2`、および `psycopg2-binary` に分割されました。依存関係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。使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 からは `recovery.conf` はなくなり、以前のすべてのリカバリパラメーターは `recovery.conf` に変換されます。 `ALTER SYSTEM SET primary_conninfo` または同様のものから保護するために、Patroni は `postgresql.auto.conf` を解析し、そこからすべてのスタンバイおよびリカバリパラメーターを削除します。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` セクションで指定でき、Patroni は PostgreSQL12 の `postgresql.conf` にそれを書き込みます。

- PostgreSQL11 以降でスーパーユーザーなしで `pg_rewind` を使用できるようにしました AlexanderKukushkin

この機能を使用する場合は、Patroni 構成ファイルの `postgresql.authentication.rewind` セクションで `username` および `password` を定義してください。既存のクラスターの場合、ユーザーを手動で作成し、いくつかの機能で `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 _` をご覧ください。

- レプリカで実際の `primary_conninfo` 値と目的の `primary_conninfo` 値をスマートに比較します AlexanderKukushkin

既存のプライマリスタンバイクラスターをパトロニによって管理されているクラスターに変換する場合、レプリカのリスタートを回避するのに役立つ場合があります

- IPv6 のサポート AlexanderKukushkin

2 つの大きな問題がありました。PatroniRESTAPI サービスは `0.0.0.0` のみリスンしており、`api_url` および `conn_url` で使用される IPv6IP アドレスは適切に引用されませんでした。

- Kerberos のサポート AjithVilas、AlexanderKukushkin

Patroni 構成ファイルでパスワードを定義する代わりに、Postgres ノード間で Kerberos 認証を使用することが可能になります

- Manage `pg_ident.conf` AlexanderKukushkin

この機能は `pg_hba.conf` と同様に動作します。 `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 WilfredRoset

PostgreSQL が実行されている場合にのみ HTTP ステータスコードを結果ます

- `/read-only` および `/read-write` エンドポイントを追加しました。JulienRiou

`/read-only` エンドポイントは、レプリカとプライマリ全体で読み取りバランスのとれた読み取りを有効にします。 `/read-write` エンドポイントは、 `/primary`、 `/leader`、および `/master` のエイリアスです。

- `SSLContext` を使用して RESTAPI ソケットをラップしました。JulienRiou

`ssl.wrap_socket()` の使用は非推奨であり、TLS1.1 のような間もなく非推奨になるプロトコルはまだ許可されていました。

Logging improvements

- 2 段階ロギング AlexanderKukushkin

すべてのログメッセージは、最初にメモリ内のキューに書き込まれ、後で別のスレッドから `stderr` またはファイルに非同期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。制限に到達すると、Patroni はログを失い始めますが、これは HA ループをブロックするよりはまだ良いです。

- レイテンシーとともに GET/OPTIONSAPI 呼び出しのデバッグログを有効にします。JanTomsa

HAProxy、Consul、またはどのノードがプライマリ/レプリカであるかを決定するその他のツールによって実行されるヘルスチェックのデバッグに役立ちます。

- 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到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、いくつかの問題のデバッグに役立つことを願っています。

Improvements in patronictl

- スケジュールされたスイッチオーバーと再起動のためのダイアログの強化 RafiaSabih

以前のダイアログはスケジュールされたアクションを考慮していないため、誤解を招くものでした。

- 構成ファイルが存在するかどうかを確認します WilfredRoset

指定されたファイル名が存在しない場合、サイレントに無視する代わりに、構成ファイルについて詳しく説明します誤解を招く可能性があります。

- EDITOR のフォールバック値を追加しましたウィルフリード・ロゼット

EDITOR 環境変数が定義されていない場合、`patronictl edit-config` は `PatroniCtlException` で失敗していました。新しい戦略は、`editor` および `vi` を試すことです。これは、ほとんどのシステムで利用できるはずです。

Improvements in Consul support

- Consul 一貫性モードを指定できるようにしました。JanTomsa
一貫性モード [here](#) _の詳細については、こちらをご覧ください。
- SIGHUP で Consul 構成をリロードキャメロンダニエルクセラ、アレクサンダーククシュキン
これは、誰かが `token` の値を変更する場合に特に役立ちます。

Bugfixes

- スイッチオーバー/フェイルオーバーのコーナーケースを修正しました。SharoonThomas
RESTAPI にアクセスできず、フォールバックとして DCS を使用している場合、変数 `scheduled_at` は未定義になる場合があります。
- カスタムブートストラップ中に `pg_hba.conf` のローカルホストへのオープントラスト AlexanderKukushkin
以前は `unix_socket` のみにオープンされていたため、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/"`
- 元のリーダーがリードしている場合でも、同期ノードを正常であると考えます AlexanderKukushkin
プライマリが DCS へのアクセスを失うと、Postgres を読み取り専用で再起動しますが、他のノードが RESTAPI を介して古いプライマリにアクセスできる場合があります。古いプライマリが同期スタンバイより先に WAL 位置を報告していたため、このような状況により、同期スタンバイが昇格しませんでした。
- スタンバイクラスターのバグ修正 AlexanderKukushkin
`standby_leader` がアクセスできない場合に、スタンバイクラスターでレプリカをブートストラップできるようにし、その他のいくつかのマイナー修正。

20.30 バージョン 1.5.6

2019-08-03 リリース

New features

- プロキシのセットを介した etcd クラスターでの作業のサポート 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、プロキシホストを介したラウンドロビンだけを実行します。動作は `etcd.use_proxies` によって制御されます。
- ノードのロールが変更されたときのコールバック動作が変更されました AlexanderKukushkin
ロールが `master` または `standby_leader` から `replica` に、または `replica` から `standby_leader` に変更された場合、`on_restart` コールバックは、`on_role_change` コールバックを優先して呼び出されません。
- postgres の起動方法を変更します AlexanderKukushkin

自分自身と ``multiprocessing.Pipe`` を実行する代わりに ``multiprocessing.Process`` を使用して、ポストマスター pid を Patroni プロセスに送信します。その前はパイプを使用していましたが、標準入力クロードされた状態でポストマスタープロセスを残していました。

Bug fixes

- スタンバイリーダーの RESTAPI から返されたロールを修正しました AlexanderKukushkin

``standby_leader`` の代わりに ``replica`` を誤って返していました

- Kill できなかった場合はコールバック終了を待機します JulienTachaires

Patroni には、新しいコールバックをキャンセルしたもの ``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ための十分な特権がありません。実行中のスクリプトを強制終了できなかった場合、Patroni は完了するまで待機して、次のコールバックを実行します。

- `dc.get_cluster` メソッドにかかるロック時間を削減します。AlexanderKukushkin

ロックが保持されているため、DCS の遅延が RESTAPI ヘルスチェックに影響を与え、誤検出を発生させていました。

- `pg_wal/pg_xlog` がシンボリックリンクの場合、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、後でファイルシステムからリンクが解除されるディレクトリで Patroni が起動された場合に何が失敗するかによって異なります。

- Etcd との通信時に ssl バージョンを強制しないでください AlexanderKukushkin

何らかの不明な理由で、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 を適用します。Patroni 側でこの問題を解決しました。

20.31 バージョン 1.5.5

2019-02-15 リリース

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を導入し、`patronictl` リスト出力を改善し、多くのバグを修正します。

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す。ÉtienneM

以前は、構成ファイルまたは `PATRONI_ETCD_URL` の一部としてのみそれらを構成することが可能でしたが、これは必ずしも便利ではありません。

- 元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しているため、元のマスターは新しいレプリカとして起動できない場合があります。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ワイプし、再初期化することです。この動作は、`postgresql.remove_data_directory_on_diverged_timelines` を設定することにより変更できます。設定されると、Patroni はデータディレクトリをワイプし、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する AlexanderKukushkin

古いレプリカの検出に役立ちます。それに加えて、ポート値がデフォルトでない場合、または同じホストで実行されている複数のメンバーが存在する場合、`Host` は `{port}` をインクルードします。

- `$$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 の構成、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、サービスなしでエンドポイントが削除されます。headless サービスは削除されません。

Bug fixes

- リーダーウォッチブロッキングクエリの読み取りタイムアウトを調整します 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、指定された最大待機時間に追加される少量のランダムな追加待機時間だけ増加して、同時要求のウェイクアップ時間を分散します。`wait/16` の追加時間を最大期間に加算します。私たちの場合、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を介して postgres に接続するときは、常に `replication=1` を使用します。AlexanderKukushkin

Postgres10 以降、`pg_hba.conf` 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ーの `recovery.conf` に `primary_conninfo` を書き込まないでください AlexanderKukushkin

`standby_cluster` 構成で `host` も `port` も定義されていないにもかかわらず、Patroni は `primary_conninfo` を `recovery.conf` に配置していましたが、これは役に立たず、多くのエラーを生成していました。

20.32 バージョン 1.5.4

リリース 2019-01-15

このバージョンは、柔軟なロギングを実装し、多くのバグを修正します。

New features

- 伐採インフラの改善アレクサンダー・ククシュキン、ルーカス・キャピストラント、アレクサンダー・アニキン

ロギング構成は、環境変数からだけでなく、Patroni 構成ファイルから構成できます。構成を更新してリロードを実行するか、Patroni プロセスに SIGHUP を送信することにより、ランタイムにログイン

グ構成を変更できます。デフォルトでは、Patroni はログを stderr に書き込みますが、ログをファイルに直接書き込み、特定のサイズに達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。それに加えて、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可能性が追加されました。

- リーダー選挙中に現在のタイムラインを考慮できるようにしますアレクサンダー・ククシュキン

ノードは、現在最新の既知のタイムラインにないにもかかわらず、自分自身を最も正常なノードと考えている場合があります。場合によっては、このようなノードのプロモーションを回避したい場合があります。これは、`check_timeline` パラメータを `true` に設定することにより実現できますデフォルトの動作は変更されません。

- スーパーユーザーの資格情報の要件の緩和

Libpq では、ユーザー名もパスワードも明示的に指定せずに接続を開くことができます。状況に応じて、pgpass ファイルまたは pg_hba.conf の信頼できる認証方法に依存します。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構成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 を介して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 off に設定します AlexanderKukushkin

クラスターが完全に機能するまで、wals と履歴ファイルのアーカイブは避けたいです。カスタムブートストラップに pg_upgrade が関係する場合、これは非常に役立ちます。

- 起動時にグローバル構成をロードするときに 5 秒のバックオフを適用します AlexanderKukushkin

Patroni が起動したばかりのときに DCS のハンマリングを回避するのに役立ちます。

- 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す。AlexanderKukushkin

それらは無害ですが、かなり迷惑で、時には怖かったです。

- 作成時に recovery.conf の rw パーマ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 ユーザー以外の者にこのファイルを読んでほしくない。

- HTTPServer 例外をロガーにリダイレクトします。JulienRiou

デフォルトでは、このような例外は通常のログを台無しにして標準出力にログに記録されました。

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 Patroni プロセスから stderr を継承すると、すべての Postgres ログがすべての patroni ログとともに表示されます。Patroni および Postgres ログは、標準のツール 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、この変更は、postgres がいくつかの警告を stderr 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バグを修正します。

- Consul サービスチェックの登録解除タイムアウトを Go 時間形式で設定します PavelKirillov
明示的に記載されていない限り、タイムユニット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す。DmitryDolgov、AlexanderKukushkin
有効な値として文字列のみを受け入れていたため、ポートを整数として指定し、create_replica_methods をリストとして指定することはできません。

20.33 バージョン 1.5.3

2018-12-03 リリース

互換性とバグ修正リリース。

- zookeeper に対して python3 で実行する際の安定性が向上しました。AlexanderKukushkin
`loop\_wait` の変更により、パトローニは zookeeper から切断され、再接続されませんでした。
- postgres9.3 との壊れた互換性を修正しました。AlexanderKukushkin
9.3 は replication='database' を理解しないため、レプリケーション接続を開くときは replication=1 を指定する必要があります。
- HA ループごとに少なくとも 1 回は Consul セッションをリフレッシュし、Consul セッション例外の処理を改善するようにしました。AlexanderKukushkin
ローカルコンスルエージェントを再起動すると、ノードに関連するすべてのセッションが無効になります。セッションリフレッシュを時間通りに呼び出さず、セッションエラーの適切な処理を実行しないことが、プライマリの降格を発生させていました。

20.34 バージョン 1.5.2

2018-11-26 リリース

互換性とバグ修正リリース。

- kazoo-2.6.0 との互換性 AlexanderKukushkin
要求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 モジュールから create_connection メソッドを再定義します。kazoo の最後のリリースでは、create_connection メソッドを呼び出す方法が少し変更されました。
- Consul クラスターがリーダーを失ったときの Patroni クラッシュを修正しました。AlexanderKukushkin
クラッシュは、touch_member メソッドの誤った実装が原因で発生していました。ブール値を返し、例外を発生させないはずです。

20.35 バージョン 1.5.1

2018-11-01 リリース

このバージョンは、永続的なレプリケーションスロットのサポートを実装し、pgBackRest のサポートを追加し、多くのバグを修正します。

New features

- 永久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、フェールオーバー/スイッチオーバーで保持されます。つまり、新しいプライマリの Patroni は、プロモートの直後に構成済みのレプリケーションスロットを作成します。*patronictl edit-config* の助けを借りてスロットは設定できます。初期構成は `:ref: bootstrap.dcs <yaml_configuration>` で行うこともできます。

- pgbackrest のサポートを追加しました YogeshSharma

pgBackrest は、既存の \$PGDATA フォルダに復元できます。これにより、最後のバックアップから変更されていないファイル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ー"/ワークフローのいくつかのバグ修正 AlexanderKukushkin

詳細については、<https://github.com/zalando/patroni/pull/823> を参照してください。

- クラスター管理が一時停止され、DCS が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

回帰は以下で紹介されました <https://github.com/zalando/patroni/commit/90cf930036a9d5249265af15d2b787ec7517cf57>

20.36 バージョン 1.5.0

2018-09-20 リリース

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 での実行の実験的なサポートを導入し、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、プライマリクラスターと並行してつまり別のデータセンターで実行され、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るスタンバイクラスターを形成できます。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複製するように自分自身を選択し、他のレプリカはカスケード方法でそこから複製します。この機能のより詳細な説明といくつかの構成例については、[here](#) を参照してください。

- Consul 館の登録サービス PavelKirillov、AlexanderKukushkin

consul *configuration* の ``register_service`` パラメータが有効になっている場合、ノードは名前 ``scope`` とタグ ``master``、``replica`` または ``standby-leader`` でサービスを登録します。

- 実験的な Windows サポート PavelGolub

今後、Patroni を Windows で実行できるようになりますが、Windows のサポートは新しいもので、Linux 対応するものほど多くの現実世界でのテストは受けていません。皆様のフィードバックをお待ちしております。

Improvements in patronictl

- patronictl-k/--insecure フラグと restapi 証明書のサポートを追加しました。WilfriedRoset

以前は、RESTAPI が自己署名証明書で保護されていた場合、*patronictl* はそれらの検証に失敗しました。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するように、または構成の `:ref: `ctl: <patronictl_settings>` セクションで CA およびクライアント証明書を提供するように *patronictl* を構成できるようになりました。

- patronicl スイッチオーバー/フェールオーバー出力から nofailover タグを持つメンバーを除外する 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 の非キーと値の出力行の解析を回避する AlexanderAnikin

特定の状況では、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 コードのエラーがトリガーされ、実際の問題が隠蔽されます。多くの場合、このような行は、通常の出力前、つまりバイナリメジャーバージョンが PostgreSQL データディレクトリのバージョンと一致しない場合、pg_controldata によって表示される警告で出力されます。

- リーダー選挙中のエラーメッセージにメンバー名を追加します。JanMussler

リーダー選挙中に、パトローニはクラスターのすべての既知のメンバーに接続し、それらのステータスを要求します。このようなステータスは Patroni ログに書き込まれ、メンバーの名前が含まれます。以前は、メンバーにアクセスできない場合、エラーメッセージには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、WAL ポジションをすぐに予約します。AlexanderKukushkin

9.6 以降、``pg_create_physical_replication_slot`` ファンクションは、追加のブールパラメータ ``immediately_reserve`` を提供します。``false`` に設定されている場合、デフォルトでもありますが、スロットは、最初のクライアント接続を受信するまで WAL ポジションを予約せず、スロットの作成と間の時間ウィンドウでクライアントが必要とする一部のセグメントを失う可能性があります。初期クライアント接続。

- 厳密な同期レプリケーションのバグを修正しました。AlexanderKukushkin

``synchronous_mode_strict:true`` で実行している場合、場合によっては、Patroni は ``*`` を ``synchronous_standby_names`` に配置し、ほとんどのレプリケーション接続の同期状態を ``potential`` に変

更します。以前、Patroni は、`async` 状態を持つもののみを考慮するため、このような状況では同期候補を選択できませんでした。

20.37 バージョン 1.4.6

リリース 2018-08-14

Bug fixes and stability improvements

このリリースでは、非マスターノードの 200 を返す PatroniAPI/マスターエンドポイントに関する重大な問題を修正します。これは報告の問題であり、実際のスプリットブレインではありませんが、特定の状況では、クライアントが読み取り専用ノードにリダイレクトされる場合があります。

- 降格時に `is_leader` ステータスをリセット AlexanderKukushkin、OleksiiKliukin

降格されたクラスターメンバーが `/masterAPI` 呼び出しでコード 200 で応答を停止することを確認します。

- 新しい `"/cluster_unlocked/"` フィールドを API 出力に追加します DmitryDolgov

このフィールドは、クラスターでマスターが実行されているかどうかを示します。これは、レプリカの 1 つを除き、他のノードを照会できない場合に使用できます。

20.38 バージョン 1.4.5

2018-08-03 リリース

New features

- 新しい postgres 構成を適用する際のロギングを改善しました。DonSeiler

Patroni は、変更されたパラメーターの名前と値をログに記録します。

- Python3.7 互換性 ChristophBerg

`async` は python3.7 の予約キーワードです

- メンバーがシャットダウンされたときに DCS で状態を `"/stopped/"` に設定します。TonySorrentino

これは、メンバーの状態を `"/patronictl/"` コマンドの `"/stopped/"` として表示します。

- 古い `postmaster.pid` が実行中のプロセスと一致する場合にログに記録されるメッセージを改善します AntsAasma

以前のものは混乱を超えていました。

- `patronictl` リロード機能の実装 DonSeiler

それまでは、RESTAPI を呼び出すか、パトロニプロセスに SIGHUP 信号を送信することによってのみ構成をリロードすることができました。

- レプリカとして起動するときに、制御データ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` の値とグローバル構成で設定されるその他のパラメーターは、プライマリが実際に使用するものよりも低い場合があります。これが発生すると、レプリカは起動できないため、手動で修正する必要があります。Patroni は、`pg\_controldata` から値を読んで適用し、postgres を起動し、`pending\_restart` フラグを設定することにより、それを処理します。
- 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、Patroni は PATH、LC_ALL、および 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。次に、LD_LIBRARY_PATH で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すると役立つはずです。
- create_replica_method の名前を create_replica_methods に変更します DmitryDolgov

それが実際に配列であることを明確にするため。古い名前は、下位互換性のために引き続きサポートされています。

Bug fixes and stability improvements

- 一時停止状態の pg_rewind によるレプリカ起動の条件を修正しました OleksiiKliukin

以前に pg_rewind を既に実行したレプリカを起動しないようにします。
- 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 応答する AlexanderKukushkin

DCS がパーティション化されている場合、Patroni が以前の降格マスターのマスターを報告しないようにします。
- 新しいコンスルモジュールとの互換性を修正しました。AlexanderKukushkin

v1.1.0 から python-consul は内部 API を変更し、`dict` の代わりに `list` を使用してクエリパラメーターを渡すようになりました。
- シャットダウン中に PatroniRESTAPI スレッドから例外をキャッチする AlexanderKukushkin

これらのキャッチされない例外により、シャットダウン時に PostgreSQL が実行され続けました。
- Postgres がマスターとして実行されている場合にのみクラッシュリカバリーを実行します。AlexanderKukushkin

`pg\_controldata` が `inProduction` または `shuttingdown` または `incrashrecovery` をレポートする必要があります。その他のすべての場合、クラッシュリカバリーは必要ありません。
- 構成エラーの処理を向上させました。HenningJacobs、AlexanderKukushkin

Patroni 構成ファイルを更新し、SIGHUP を Patroni プロセスに送信することにより、ランタイム `restapi.listen` を含む多くのパラメーターを変更できます。この修正では、一部のパラメーターが無効な値を受け取ったときに、`restapi` スレッドからあいまいな例外を削除します。

20.39 バージョン 1.4.4

2018-05-22 リリース

Stability improvements

- `poll_failover_result` の競合状態を修正しました。AlexanderKukushkin

フェイルオーバーもスイッチオーバーにも直接影響しませんでしたでしたが、まれに、元のリーダーがロックを解放したときに成功を報告するのが早すぎ、「Failedoverto/」の代わりに「Failedoverto/"None/"」が生成されました。`"desired-node/"` メッセージ。

- Postgres パラメーター名を大文字と小文字を区別しないように扱う AlexanderKukushkin

Postgres パラメーターのほとんどは `snake_case` 名前を持っていますが、このルールには3つの例外があります `DateStyle`、`IntervalStyle`、`TimeZone`。Postgres は、別のケースたとえば `timezone='some/tzn'` で書かれた場合、これらのパラメーターを受け入れます。ただし、Patroni は `pg_settings` でこれらのパラメーター名の大文字と小文字を区別しない一致を見つけることができず、結果としてこのようなパラメーターを無視しました。

- 実行中の postgres にアタッチし、クラスターが初期化されていない場合、スタートを中止します。AlexanderKukushkin

Patroni は、既に実行されている Postgres インスタンスに自分自身をアタッチできます。レプリカに到達する前に、マスターノードで Patroni の実行を開始することが不可欠です。

- `patronictlscaffold` の動作を修正しました AlexanderKukushkin

`json` エンコードされた文字列の代わりに `dict` オブジェクトを `touch_member` に渡すと、DCS 実装がそのエンコードを担当します。

- 一時停止中にリーダーキーの更新に失敗した場合、マスターを降格しないでください AlexanderKukushkin

メンテナンス中に、DCS は読み取り要求に応答し続けながら、書き込み要求の失敗を開始する場合があります。その場合、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します AlexanderKukushkin

Postgres が再起動された場合、Patroni はレプリケーションスロットのリストが予想と一致することを確認する必要があります。

- 一時停止から復帰した後の `sysid` と同期レプリケーションスロットを検証する AlexanderKukushkin

``maintenance`` モード中に、データディレクトリが完全に書き換えられる場合があるため、``Database-system-identifier`` がまだクラスターに属し、レプリケーションスロットがパトローニ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 を実行せずに開始すると失敗する可能性を修正しました。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 を実行している場合、このような問題が発生する可能性が高くなります。

- 誤ってワイプされた DCS の保護を向上させました 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多くのロジックが用意されています。すべてのキーを復元することもできます。ただし、この変更が発生するまでは、`/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っていました。

- 無効なシステム ID が発生しても終了しない 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、クラスターは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ます。そうしないと、再初期化が発生しないため、Patroni を終了しないでください。

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+は、`[]`の代わりに``None``に設定された``Endpoints.subsets``を返し始めました。

Bootstrap improvements

- `recovery.conf` の削除をオプションにします。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` のようなツールを使用してバックアップからブートストラップする場合に役立ちます。

- `basebackup` ビルトインメソッドのオプションを許可します OleksiiKliukin

カスタムレプリカ作成方法で定義される方法と同様に、構成で ``basebackup`` セクションを定義することにより、ビルトイン `basebackup` メソッドにオプションを提供できるようになりました。違いは、`basebackup` セクションで受け入れられるフォーマットにあります。`pg_basebackup` は `--key=value` と `--key` オプションの両方を受け入れるため、セクションのコンテンツはキーと値のペアの辞書、または 1 要素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。

20.40 バージョン 1.4.3

2018-03-05 リリース

Improvements in logging

- 環境変数からログレベルを設定できるようにします。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 要求のログレベルを設定します。KubernetesAPI 呼び出し考えられるログレベルの名前を取得するには、``thedocsforPythonlogging<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ないでください AlexanderKukushkin

etcd 構成にホストが 1 つだけあり、正確にこのホストにアクセスできない場合、Patroni はクラスタトポロジの検出を開始していましたが、成功しませんでした。代わりに、次の利用可能なノードに切り替える必要があります。

- カスタムブートストラップの後に、bootstrap.pg_hba のコンテンツを pg_hba.conf に書き込みます。AlexanderKukushkin
`initdb` を使用した通常のブートストラップと同様に動作するようになりました
- シングルユーザーモードはユーザー入力を待機していて、終了しませんでした。AlexanderKukushkin
回帰は以下で紹介されました <https://github.com/zalando/patroni/pull/576>

20.41 バージョン 1.4.2

2018-01-30 リリース

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す AlexanderKukushkin
フェールオーバー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` は引き続き `Scheduledswitchover` の代わりに `Scheduledfailover` を報告していました。
- 保留中の再起動に関する情報を表示する AlexanderKukushkin
一部の構成の変更を適用するには、postgres を再起動する必要がある場合があります。Patroni は、RESTAPI で、およびノードのステータスを DCS に書き込むときに、既にそれに関する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。
- 構成ファイルの cluster_name を操作するように show-config を作成しました。AlexanderKukushkin
`patronictl edit-config` と同様に動作します

Stability improvements

- ブートストラップ中に pg_controldata の呼び出しを回避する 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枠があります。このような場合、pg_controldata 呼び出しはエラーメッセージで失敗していました。
- psutil から発生した例外を処理します。AlexanderKukushkin
cmdline は、`cmdline()` メソッドが呼び出されるたびに読み込まれ、解析されます。調べられるプロセスが既に消失している場合もあります。その場合、`NoSuchProcess` が発生します。

Kubernetes support improvements

- k8sAPI からのエラーを飲み込まないでください。AlexanderKukushkin
KubernetesAPI の呼び出しは、さまざまな理由で失敗する場合があります。場合によっては、このような呼び出しを再試行する必要があります、他の場合には、エラーメッセージと例外スタックトレース

スをログに記録する必要があります。ここでの変更は、Kubernetes の権限の問題をデバッグするのに役立ちます。

- Kubernetes の例の Docker ファイル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

その前は `feature/k8s` を使用していましたが、時代遅れになりました。

- k8s でパトリニを実行するための適切な RBAC を追加します。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。

20.42 バージョン 1.4.1

2018-01-17 リリース

Fixes in patronictl

- フェイルオーバー先のメンバーの提案リストに現在のリーダーを表示しません。(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存在する場合でも機能する可能性があるため、フェイルオーバーが可能な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、再度実行されますが、/failover エンドポイントに対して。

20.43 バージョン 1.4

2018-01-10 リリース

このバージョンでは、Kubernetes を DCS として使用するサポートを追加し、Etcd、Zookeeper、または Consul を追加展開せずに、Kubernetes でクラウドネイティブエージェントとして Patroni を実行できます。

Upgrade notice

pip を介して Patroni をインストールすると、Etcd、Zookeeper、Consul、または Kubernetes のライブラリ、または AWS のサポートなどの依存関係は導入されません。それらを有効にするには、`pipinstallpatroni[etcd,kubernetes]` などの pipinstall コマンドで明示的にリストする必要があります。

Kubernetes support

Kubernetes ベースの DCS を実装します。エンドポイントのメタデータは、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 ConfigMaps をサポートしています。この機能の詳細については、[Kubernetes chapter of the documentation](#) をご覧ください。

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する 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 サイクルのすべてのループで 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、パトローニは、リカバリステータスと絶対的な wal 位置を知る必要があります。今後、Patroni は、レプリカで 2 つ、マスターで 3 つの代わりに、1 つの SELECT のみを実行してこの情報を取得します。

- ロックがある場合にのみシャットダウン時にリーダーキーを削除します。AntsAasma

無条件削除は、不要で誤解を招く例外を生成していました。

Improvements in patronictl

- version コマンドを patronictl に追加します。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 コマンドの cluster_name 引数をオプションに指定するようにします。AlexanderKukushkin、AntsAasma

patronictl が scope が定義された通常の Patroni 構成ファイルを使用している場合に動作します。

- スケジュールされたスイッチオーバーとメンテナンスモードに関する情報を表示します AlexanderKukushkin

それまでは、Patroni ログからまたは DCS から直接この情報を取得することができました。

- patronictl reinit の改善 AlexanderKukushkin

Patroni が他のアクション、つまり postgres を起動しようとしていたときに、patronictl reinit が続行を拒否する場合があります。patronictl は、このような長時間実行アクションをキャンセルするコマンドを提供せず、唯一の危険な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、reinit 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 および patronictl resume に --wait フラグを実装します。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確認されるまで、patronictl は待機します。このような動作は、DCS および RESTAPI を介してすべてのノードの pause フラグを公開することにより実現されます。

- patronictl failover の名前を patronictl switchover に変更します。AlexanderKukushkin

以前の failover は、実際にはスイッチオーバーのみが可能でした。リーダーなしでクラスターで進むことを拒否しました。

- patronictl failover の動作を変更します。AlexanderKukushkin

リーダーがない場合でも機能しますが、その場合、新しいリーダーになるノードを明示的に指定する必要があります。

Expose information about timeline and history

- DCS および API 経由で現在のタイムラインを公開する AlexanderKukushkin

クラスターのメンバーごとに、現在のタイムラインに関する情報を保存します。この情報は API を介してアクセスでき、DCS に保存されます

- DCS の/history キーにプロモーション履歴を保存する AlexanderKukushkin

さらに、対応するプロモーションのタイムスタンプで強化されたタイムライン履歴を DCS の/history キーに保存し、各プロモートで更新します。

Add endpoints for getting synchronous and asynchronous replicas

- 新しい/sync および/async エンドポイントを追加します AlexanderKukushkin、OleksiiKliukin

これらのエンドポイント/synchronous および/asynchronous としてもアクセスできますは、対応する同期および非同期レプリカの場合のみ 200 を返します `noloadbalance` としてマークされたものを除きます。

Allow multiple hosts for Etcd

- 新しい `hosts` パラメータを Etcd 構成に追加します AlexanderKukushkin

このパラメーターには、実行中の etcd クラスターメンバーのリストを検出および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ます。作業中に何らかの理由で、検出されたホストのこのリストが使い果たされると、そのリストに使用可能なホストがない場合、Patroni は `hosts` パラメータから初期リストに戻ります。

20.44 バージョン 1.3.6

2017-11-10 リリース

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検証します。(AntsAasma)

postmaster.pid をクリーンアップしないクラッシュの後、同じ pid を持つ新しいプロセスが存在する可能性があり、is_running() の誤検出が発生し、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 ainlolcat を失ったときにブートストラップする前に postgresql をシャットダウンする

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 プロセスはしばらく生きたまま、その元のマスターの代わりに作成されるレプリカの起動またはレプリケートを防ぐことができます。この修正により、Patroni はポストマスター 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実行されている場合に古いポストマスターを終了させます。

- postgres マスターが死亡した場合に、シングルユーザーモードでクラッシュリカバリーを実行します AlexanderKukushkin

スタンバイとしてすぐに起動するのは安全でなく、postgres が完全にシャットダウンしていない場合は `pg_rewind` を実行できません。シングルユーザーのクラッシュリカバリーは、`pg_rewind` が有効になっている場合、または現在マスターが存在しない場合にのみキックインされます。

Consul improvements

- Consul のデータセンター構成を提供できるようにしました ViliusOkockis、AlexanderKukushkin

その前に、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

Patroni 構成で `consul.token` が定義されている場合、常に 'X-Consul-Token' http ヘッダーで送信します。python-consul モジュールは、`consul.token` のクエリパラメータとしてトークンを受け入れない Consul REST API と/"一貫性"を持とうとしますが、'X-Consul-Token' ヘッダーで動作します。

- 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 でサポートされている最小 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 Consul KV ストアにメンバーキーとリーダーキーを作成できません。その結果、異常なクラスタが発生します。

Other improvements

- 環境変数 `PATRONI_LOGFORMAT` を介してカスタムログ形式を定義するスタスフォミン

システムロガーによって既に追加されている場合、Patroni ログのタイムスタンプおよびその他の同様のフィールドを無効にすることを許可します通常は Patroni がサービスとして実行されている場合。

20.45 バージョン 1.3.5

2017-10-12 リリース

Bugfix

- データディレクトリが削除された場合、ロールを '未初期化' に設定します AlexanderKukushkin

ノードがマスターとして実行されている場合、フェールオーバーを妨げていました。

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` を実行してみます。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され、タイムラインが分岐されたときに発生します。`recovery.conf` に `restore_command` が定義されている場合、postgres が起動を中止し、制御データを変更しないままにする可能性が非常に高いです。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

- セッションを作成するとき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、分離されたマスターの迅速な検出が可能になりますが、別の側面では、パトローニが短いネットワーク遅延を許容することが不可能になります。

Bugfix

- Python3 のウォッチドッグを修正する 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 の比較がエラーを返さなかったため、これは誤って Python2 で動作しました。エラー報告は、実際には、Python2 で IOError を発生させ、Python3 で OSError を発生させることにより実行されます。

20.46 バージョン 1.3.4

2017-09-08 リリース

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 館 [API](#) に渡すための優先的な方法になりました。

- Consul の詳細な構成 AlexanderKukushkin

scheme、token、クライアントおよび ca 証明書 scheme を指定する可能性。

- python-consul-0.7.1 以降との互換性 AlexanderKukushkin

新しい python-consul モジュールは、一部のメソッドのシグネチャを変更しました

- "/TTL ロックを取り出すことができませんでした"メッセージはログに記録されませんでした。AlexanderKukushkin

重大なバグではありませんが、適切なログの欠如により、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 に書き込む場合、その値は引用符で囲む必要があります。AlexanderKukushkin

適切に引用されていない場合、PostgreSQL は同期レプリケーションを効果的に無効にして、動作を継続します。

Different bugfixes around pause state, mostly related to watchdog アレクサンダー・ククシュキン

- ウォッチドッグがアクティブでない場合はキープアライブを送信しない
- 一時停止モードでウォッチドッグをアクティブにしない
- 一時停止モードで正しい postgres 状態を設定する
- postgres が停止している場合は、API からクエリを実行しないでください

20.47 バージョン 1.3.3

2017-08-04 リリース

Bugfixes

- `synchronous_mode_strict` がオンになっている場合でも、昇格の直後に同期レプリケーションが無効になっていました。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、作成します AlexanderKukushkin
- postgres だけでなくすべてのデータベースへの `pg_hba.conf` のオープンアクセス FrancoBellagamba

20.48 バージョン 1.3.2

2017-07-31 リリース

Bugfix

- `patronictl edit-config` は ZooKeeper で動作しませんでした。AlexanderKukushkin

20.49 バージョン 1.3.1

2017-07-28 リリース

Bugfix

- `_MemberStatus` の変更により、API を介したフェイルオーバーが中断されました。AlexanderKukushkin

20.50 バージョン 1.3

2017-07-27 リリース

バージョン 1.3 は、カスタムブートストラップの可能性を追加し、`pg_rewind` のサポートを大幅に向上させ、同期モードのサポートを強化し、`patronictl` に構成編集を追加し、Linux でウォッチドッグサポートを実装します。さらに、これは PostgreSQL10 で正常に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既知の互換性問題はありません。バージョン 1.2 からの構成は変更せずに動作するはずですが、新しいパッケージをインストールし、Patroni を再起動する PostgreSQL の再起動が発生するか、最初に Patroni を *pause mode* に入れて、クラスター内のすべてのノードで Patroni を再起動することにより、アップグレードが可能です。一時停止モードの Patroni は `stop/startPostgreSQL`、最後に一時停止モードから再開します。

Custom bootstrap

- クラスターのブートストラップのプロセスを構成可能にする 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。`bootstrap` コマンドは、クラスターの名前とデータディレクトリへのパスを受け取ります。生成されるクラスターは、リカバリを実行するように構成できるため、バックアップからブートストラップし、ポイントインタイムリカバリーを可能にします。この機能の詳細な説明については、`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` をトリガーするための固定条件セット、つまり元のマスターを起動したとき、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行ったとき、または `nofailover` タグを持つレプリカが存在する場合があります。これらすべての場合に共通して、一部のレプリカが新しいマスターよりも先に行く可能性があります。場合によっては、`pg_rewind` は何もませんでした、他のいくつかの場合、必要に応じて実行されませんでした。この限定されたルールのリストに依存するのではなく、Patroni はマスターとレプリカの WAL 位置を比較する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、レプリカに巻き戻しが必要かどうかを確実に決定します。

Synchronous replication mode strict

- `strict` モードを追加することにより、同期レプリケーションのサポートを強化します。JamesSewell, AlexanderKukushkin

通常、`synchronous_mode` が有効で、マスターに接続されているレプリカがない場合、Patroni はマスターを書き込みに利用できるようにする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、設定されている場合、Patroni はレプリカが不足している場合に同期レプリケーションを無効にせず、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ぐ同期モードの保証に加えて、`strict` 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つだけの場合は完全に発生しないことが保証されます。

Configuration editing with patronictl

- `patronictl` に構成編集を追加しました。AntsAasma, AlexanderKukushkin

DCS に保存されている動的クラスター構成の編集を保護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 のウォッチドッグサポートを実装する AntsAasma

パトローニが実行されていないノード、または応答していないノードをリブートするために Linux ソフトウェアウォッチドッグをサポートたとえば、高負荷のため Linux ソフトウェアウォッチドッグは、非応答ノードを再起動します。Patroni 構成の `watchdog` セクションからデフォルトで `/dev/watchdog` とモード `on`、`automatic`、`off` を使用するようにウォッチドッグデバイスを構成することが可能です。[watchdog documentation](#) から詳細情報を取得できます。

Add support for PostgreSQL 10

- Patroni は、これまでにリリースされた PostgreSQL10 のすべてのベータバージョンと互換性があり、リリースされる PostgreSQL10 と互換性がある予定です。

PostgreSQL-related minor improvements

- Patroni 構成ファイルまたは DCS の動的構成を介して `pg_hba.conf` を定義します AlexanderKukushkin
 構成の `postgresql` セクションの `pg_hba` サブセクションで `pg_hba.conf` の内容を定義できます。これにより、すべてのノードにロギングし、手動で変更して、構成をリロードするのではなく、DCS でのみ定義する必要があるため、マルチプルのノードでの `pg_hba.conf` の管理が簡素化されます。
 定義されると、このセクションの内容は、現在の `pg_hba.conf` を完全に置き換えます。 `hba_file` PostgreSQL パラメーターが設定されている場合、Patroni はそれを無視します。
- UNIX ソケットを介したローカル PostgreSQL クラスターへの接続のサポート 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の PostgreSQL クラスターに接続できます。 `unix_socket_directories` が定義されていない場合、Patroni はデフォルト値を想定し、PostgreSQL 接続文字列の `host` パラメーターを完全に省略します。
- リロード時のスーパーユーザーとレプリケーション資格情報の変更のサポート AlexanderKukushkin
- PostgreSQL データディレクトリの外部への構成ファイルの保存のサポート @jouis
 新しい構成 `postgresql` 構成指示子 `config_dir` を追加します。これはデフォルトでデータディレクトリにあり、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

- HandleEtcdEventIndexCleared および EtcdWatcherCleared 例外 AlexanderKukushkin
 無駄な再試行を回避することにより、Etcd によって監視オペレーションが終了されたときのリカバリーが速くなります。
- Etcd 障害時にスピン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 接続失敗時の即時再試行とログへのスタックトレースの発行は避けてください。
- 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 でビルドされた PostgreSQL の英語以外のロケールでの ``postmasterbecomemultithreaded-whilestartup`` の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ドロップする際の追加チェック AlexanderKukushkin
 場合によっては、Patroni は WAL 送信者によってレプリケーションスロットをドロップできない場合があります。
- PostgreSQL の命名ルールに従うように、レプリケーションスロット名を 63(NAMEDATALEN-1) 文字に切り詰めます。 NickScott
- Patroni から PostgreSQL クラスターへの追加の接続が開かれる競合状態を修正しました。 AlexanderKukushkin
- ノードが空のデータディレクトリで再起動したときにリーダーキーを解放します。 AlexKerney

- リーダーなしでブートストラップを実行するときに非同期エグゼキューターをビジーに設定します。
AlexanderKukushkin

これを行わないと、パトロニは、クラスターにリーダーが存在することを必要としないブートストラップ方法でブートストラップされている間、通常のビジネスを続行したため、ノードが別のクラスターに属していることを示すエラーが発生する可能性があります。
- WAL-E レプリカ作成方法の改善 JoarWandborg、AlexanderKukushkin。
 - WAL-E ベースバックアップを解析するときに csv.DictReader を使用し、スペース区切りの日付と時刻で ISO 日付を受け入れます。
 - レプリカから現在の WAL 位置を取得して、復元する WAL の量を推定することをサポートします。以前は、コードは、マスターノードでのみ使用可能なシステム情報機能呼び出すために使用されていました。

20.51 バージョン 1.2

2016-12-13 リリース

このバージョンでは、同期レプリケーションの処理を大幅に改善し、起動プロセスとフェールオーバーの信頼性を高め、PostgreSQL9.6 サポートを追加し、多くのバグを修正します。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。(AntsAasma)

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。同期モードが有効になっている場合、Patroni は、マスター障害が発生したときに同期複製していたスタンバイにのみ自動的にフェールオーバーします。これは、このような場合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効果的に意味します。詳細な説明と実装の詳細については、`synchronous_mode` を参照してください。

Reliability improvements

- PostgreSQL が 100%正常でない場合は、`leader optime` キーに保存されているリーダー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、すぐに降格します。(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 で行う方法と同様に、Consul の再試行と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` および `--config-file` を `patronictl` のすべてのオプションに適用します。(アレクサンダー・ククシュキン)
- すべての postgres パラメーターを `postgresql.conf` に書き込みます。(アレクサンダー・ククシュキン)
これにより、パトロニによって構成された PostgreSQL を `pg_ctl` のみで起動できます。

- 構成内にユーザーが存在しない場合、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` は、一時停止を実行しようとしたノードに異常がある場合に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またはリーダーキーが変更されるまでスリープしていました。この変更では、レプリカの PostgreSQL が `running` 状態にある場合のみを監視し、PostgreSQL が停止/開始、または再起動しているときは監視しません。

- SIGCHILD を PID1 として処理するときに、競合状態に陥ることを回避します。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、それらから SIGCHILD を処理したため、Docker コンテナ内で実行されているときに競合状態が発生する可能性があります。この変更では、Patroni の `fork/exec` を使用し、元の PID1 プロセスに子からの信号を処理させます。

- WAL-E の復元を修正します。(オレクシー・クリウキン)

以前は、WAL-E 復元は `no_master` フラグを使用して、マスターとのコンサルテーションを完全に回避し、Patroni は常に `pg_basebackup` ではなく WAL からの復元を選択します。この変更は、`no_master` の元の意味に戻します。つまり、マスターが実行されていない場合、Patroni WAL-E リストアはレプリケーション方法として選択できます。後者は、メソッドに渡された接続文字列を調べることで確認されます。さらに、再試行メカニズムをより堅牢にし、他の細部を処理します。

- 非同期 DNS リゾルバーキャッシュを実装します。(アレクサンダー・ククシュキン)

DNS が一時的に使用できない場合の失敗を回避しますたとえば、ノードが受信した過剰なトラフィックのため。

- 開始状態とマスター開始タイムアウトを実装します。(AntsAasma, AlexanderKukushkin)

以前は `pg_ctl` はタイムアウトを待って、PostgreSQL が実行されていると考えて喜んで踏み続けました。これにより、PostgreSQL は実際には実行されていないときに実行されているとしてリストに表示され、フェールオーバー、クラッシュリカバリー、またはフェールオーバーとリワインドの欠落によって中断されたクラッシュリカバリーが発生する競合状態が発生しました。この変更では、`master_start_timeout` パラメーターを追加し、メイン HA ループ `starting` の新しい状態を導入します。`master_start_timeout` が 0 の場合、マスターがクラッシュしたときに、フェールオーバー候補が存在するとすぐにフェールオーバーします。それ以外の場合、Patroni はマスターで 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、可能であればフェールオーバーします。手動フェールオーバー要求は、マスターのクラッシュ中にタイムアウトの有効期限が発生する前でも受け入れられます。

`timeout` パラメーターを `restart` API エンドポイントと `patronictl` に導入します。設定され、再起動がタイムアウトより長くかかる場合、PostgreSQL は異常であると考えられ、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。(AntsAasma)

Patroni が巻き戻す必要があると考えているが、巻き戻しが不可能な場合、つまり `pg_rewind` が存在しない場合、一時停止モードでの不必要な再起動を避けます。`superuser` 認証が `pg_rewind` 関連の Patroni 構成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、同じタイプの以前のコールバックを強制終了します。コールバックの実行時にゾンビプロセス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 で設定されているが、このリーダーキーの更新が失敗する場合、以前のマスターの昇格を避けてください。(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「一貫性のない読み取り」を許可する Etcd およびその他の DCS 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、ロールを維持し続けてしまう問題を回避します。

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、新しいクラスターに対して PostgreSQL を起動した直後に、このオプションのスクリプト引数を呼び出します。スクリプトは、`superuser` で接続 URL を受け取り、パスワードを含む `.pgpass` ファイルをポイントするように `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役立ちます。

- PostgreSQL9.6 サポートを実装します。(アレクサンダー・ククシュキン)

`wal_level = replica` を `hot_standby` の同義語として使用し、あるものから別のものに変更されときの `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

- `Patronimain 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す。(ラウリ・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・クリウキン)

ドキュメントをさまざまなデバイススマートフォンを含むから簡単に表示し、検索可能にします。

- 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・クリウキン)

Patroni は、`major.minor.patch` バージョンスキーマに従い、小さいが重要なバグ修正で新しいマイナーバージョンのリリースを回避します。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。

20.52 バージョン 1.1

2016-09-07 リリース

このリリースでは、一時停止モードを導入することにより Patroni クラスターの管理を向上させ、スケジューラされたおよび条件付き再起動によるメンテナンスを向上させ、Etcd または Zookeeper との Patroni の相互作用をより弾力的にし、`patronictl` を大幅に強化します。

Upgrade notice

1.0 より前のリリースからアップグレードする場合、1.0 リリースノートで資格情報と構成フォーマットの変更についてお読みください。

Pause mode

- 一時停止モードを導入して、PostgreSQL インスタンスの管理から Patroni を一時的に切断します MuratKabilov, AlexanderKukushkin, OleksiiKliukin。

以前は、PostgreSQL を終了せずに Patroni に SIGKILL シグナルを送信して停止する必要がありました。新しい一時停止モードは、Patroni を終了せずに、クラスター全体で PostgreSQL から Patroni を切断します。これは、Pacemaker のメンテナンスモードに似ています。Patroni は引き続き DCS のメンバーとリーダーキーを更新しますが、プロセス中に PostgreSQL サーバーを起動、停止、または再起動しません。いくつかの例外があります。たとえば、手動フェイルオーバー、再初期化、および再起動は引き続き許可されます。 [a detailed description of this feature](#) を読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

- `restartAPI` コマンドに条件を追加します OleksiiKliukin

この変更は、再起動を行うために検証できるいくつかの条件を追加することにより、Patroni の再起動を強化します。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の再起動、PostgreSQL バージョン番号の確認、または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が含まれます。

- スケジュールされた再起動を追加します。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。ノードごとに 1 つのスケジュールされた再起動のみがサポートされています。スケジュールされた再起動が必要でない場合は、クリアすることが可能です。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おり、たとえば、夜間にマイナー PostgreSQL のアップグレードをスケジュールし、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できます。

- 条件付きおよびスケジュールされたリスタートのサポートを `patronictl` に追加します。MuratKabilov。

`patronictlrestart` は、いくつかの新しいオプションをサポートしています。スケジュールされたアクションをクリーンアップする `patronictlflush` 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す AlexanderKukushkin

もともと、`ping_timeout` および `connect_timeout` 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。Patronistop_wait は考慮されませんでした。その結果、1 回の再試行でセッションタイムアウトよりも時間がかかる可能性があり、パトローニはロックを解放して降格することを余儀なくされました。

この変更では、`ping` と接続タイムアウトを `loop_wait` の値の半分に設定し、接続問題の検出を高速化し、ロックを失う前に接続試行を再試行するための十分な時間を残します。

- 元の要求が成功した場合にのみ `etcd` トポロジを更新します AlexanderKukushkin

元の要求の後まで、クライアントに知られている Etcd トポロジの更新を延期します。クラスターポロジを取得するときは、Etcd クラスター内のノードの既知の数に応じて、再試行タイムアウトを実装します。これにより、クライアントはノードの最新のリストよりも要求の結果を取得することを優先します。

どちらの変更も、ネットワークに問題が発生した場合に DCS への Patroni 接続をより堅牢にします。

Patronictl, monitoring and configuration

- API を介してストリーミングレプリカに関する情報を返します FeikeSteenbergen

以前は、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 を照会する信頼できる方法がありませんでした。この変更により、/patroni エンドポイントを介して pg_stat_replication の内容が公開されます。

- patronictlscaffold コマンドを追加 OleksiiKliukin

Etcd で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 とリーダーで作成され、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レプリケートする、いわゆるマスターレス構成を作成するのに役立ちます。次に、リーダーキーを削除し、Patroni ノードの 1 つを昇格させ、元のマスターを Patroni ベースの HA クラスターに置き換えることができます。

- PostgreSQL バイナリを見つけるための構成オプション bin_dir を追加します AntsAasma

複数の PostgreSQL バージョンのインストールを同時にサポートする Linux ディストリビューションを使用する場合、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パスを許可します。これは、Patroni [details](#) によって管理対象外になります。

Bug fixes and code improvements

- Patroni を PostgreSQL 10 以降の新しいバージョンスキーマと互換性のあるものにします。FeikeSteenbergen

PostgreSQL バージョンに基づいて条件付きリスタートを実行するときは、Patroni が 2 桁のバージョン番号を理解することを確認します。

- pkgutil を使用して DCS モジュールを見つけます。AlexanderKukushkin

DCS モジュールを見つけるためには、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 を起動するときに常に on_start コールバックを呼び出します。AlexanderKukushkin

以前は、Patroni は、正しいロールで既に実行されているノードに接続する場合、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をルーティングするために使用されることが多いため、接続ルーティングスキームで実行中のノードの登録に失敗する可能性があります。この修正により、Patroni は、既に実行中のノードに接続している場合でも、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ドロップしないでください MuratKabilov, OleksiiKliukin

マスターでアクティブな物理レプリケーションスロットをドロップしないようにします。いずれにせよ、PostgreSQL はそのようなスロットをドロップできません。この変更により、マスターで非 Patroni 管理のレプリカ/コンシューマを実行できるようになります。

- PostgreSQL インスタンスの起動中に Patroni 接続を閉じます AlexanderKukushkin

PostgreSQL ノードが起動されたときに、Patroni に以前のすべての接続を強制的に閉じます。postmaster が SIGKILL で強制終了された場合、以前の接続を再利用するトラップを回避します。

- メンバー名からスロット名を構成するときに無効な文字を置き換えます AntsAasma

スロット命名ルールに準拠していないスタンバイ名がスロットの作成とスタンバイの起動に失敗しないことを確認します。スロット名のダッシュをアンダースコアに置き換え、スロット名で使用できないその他のすべての文字を Unicode コードポイントに置き換えます。

20.53 バージョン 1.0

2016-07-05 リリース

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を導入します。また、多数のバグ修正も提供します。

Upgrade notice

v0.90 以前からアップグレードする場合、マスターの前に常にすべてのレプリカをアップグレードします。DCS にレプリケーション資格情報を保存しないため、古いレプリカは新しいマスターに接続できません。

Dynamic Configuration

- 動的グローバル構成の実装 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ーは DCS で設定され、多くの場合、PostgreSQL または Patroni を中断せずに適用できます。Patroni は、一部の値で PostgreSQL の再起動が必要な場合、API を介して表示される"/pendingrestart/"と呼ばれる特別なフラグを設定します。その場合、API を介して再起動を手動で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ーと差分構成ソースの処理順序の詳細については、[Patroni configuration](#) を参照してください。

構成ファイルの形式は v0.90 から **変更されました**。Patroni はまだ古い構成ファイルとの互換性がありますが、ブートストラップパラメーターを利用するには、変更する必要があります。ユーザーは、[dynamic configuration documentation page](#) を参照してそれらを更新することをお勧めします。

More flexible configuration *

- postgresql 構成とデータベース名を作成する Patroni は構成可能に接続します MisjaHoebe

`database` および `config\_base\_name` 構成パラメーターを導入します。とりわけ、PipelineDB およびその他の PostgreSQL フォークで Patroni を実行できるようになります。

- 環境を介して一部の Patroni 構成パラメーターを構成する機能を実装します。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、動的環境、つまり Kubernetes でのパトロニの実行が簡単になります。詳細については、[supported environment variables](#) を参照してください。
- ビルトインの PatroniDocker コンテナを更新して、環境ベースの構成 FeikeSteenbergen を利用します。
- PatroniDocker イメージに Zookeeper サポートを追加しました AlexanderKukushkin
- Zookeeper と Exhibitionr 構成オプションの分割 AlexanderKukushkin
- Patronicl が Patroni から構成を読み取るためにコードを再利用できるようにしました。AlexanderKukushkin
これにより、patronicl は環境ベースの構成を利用できます。
- primary_conninfo でアプリケーション名を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、バックアップのリストアップが進行中のときに pg_controldata を呼び出さないようにしました AlexanderKukushkin
この変更により、バックアップからこのノードの時間のかかる初期化中に PatroniAPI ヘルスチェックによって生成されるノイズの量が削減されます。
- 多数の pg_rewind コーナーケースを修正しました。AlexanderKukushkin
ソースクラスタがマスターでない場合、pg_rewind の実行を避けてください。
さらに、新しいパラメーター `remove_data_directory_on_rewind_failure` が true に設定されていない限り、リワインド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す AlexanderKukushkin
以前は、Patroni は常に DCS の PostgresURL からのレプリケーション資格情報を使用していました。これは、パトロニ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の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。
- デモートコール周りの非同期機構を修正しました。AlexanderKukushkin
Demote は、DCS 相互作用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、承認ヘッダーを常に送信するようにしました AlexanderKukushkin
これにより、Patroni がこれらに承認を必要とするように構成されている場合、patronicl は"/protected/" 要求を発行することができます。つまり、再起動または再初期化。
- SystemExit 例外を正しく処理します AlexanderKukushkin
SIGTERM を受信したときに Patroni が適切に停止しない問題を回避します
- confd のサンプル haproxy テンプレート AlexanderKukushkin
confide を使用して DCS のパトロニ状態から ha プロキシ構成を生成および動的に変更します

- ドキュメントを改善および再構成して、新しいユーザーにとってより親しみやすいものにします。
LauriApple
- API は `pg_ctl` 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を高めます。さらに、`pg_ctl_timeout` オプションを導入して、`pg_ctl` を介してスタート、停止、およびリスタート呼び出しのタイムアウトを設定します。
- `etcd` の再試行ロジックを修正しました。AlexanderKukushkin
再試行をより予測可能かつ堅牢にします。
- Zookeeper コードの短いネットワークの中断に対する回復力を高めます 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。

20.54 バージョン 0.90

2016-04-27 リリース

このリリースでは、Consul のサポートを追加し、新しい *noloadbalance* タグを含み、*clonefrom* タグの動作を変更し、*pg_rewind* 処理を改善し、*patronictl* 制御プログラムを改善します。

Consul support

- Consul サポートの実装 AlexanderKukushkin

Patroni は、Etcd と Zookeeper に加えて、Consul に対して実行されます。接続パラメーターは YAML ファイルで構成できます。

New and improved tags

- noloadbalance* タグを実装します。AlexanderKukushkin

このタグにより、Patroni は常にレプリカがロードバランサーで利用できないことを返すようになります。

- clonefrom* タグの実装を変更します。AlexanderKukushkin

以前は、ノード名を *clonefrom* に提供する必要があり、タグ付けされたレプリカは特定のノードからクローンを作成する必要がありました。新しい実装では、*clonefrom* がブールタグになります。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、フェールオーバーの安定性を向上させ、DCS からのデータの読み取り、元のリーダーのシャットダウン、降格、および再接続に関するいくつかのコーナーケースに対処します。

- 停止中にパトローニの子供を殺さないように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、*systemd* も PostgreSQL にシグナルを送信していました。Patroni も PostgreSQL を自分自身で停止しようとしたため、さまざまなシャットダウン要求スマートシャットダウン、それに続く高速シャットダウンに送信されます。その結果、レプリカの切断が早すぎ、元のマスターは降格後に再参加できなくなりました。Alexander による事前調査を使用して 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一部のケースを排除します。OleksiiKliukin

以前は、元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。システムに *pg_rewind* が存在する限り、以前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、つまり/"スマート/"のシャットダウン中にマスターがシャットダウンされた場合が修正されます。

- 特に、単体テストおよび受け入れテストの多数の改善により、Zookeeper と ConsulAlexanderKukushkin のサポートが有効になりました。
- TravisCI を高速化し、Zookeeper エキシビターと ConsulAlexanderKukushkin に対する実行テスト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要求で 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す。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り、システム環境変数を読み取る可能性を防ぎます。

Configuration and control changes

- *patronictl* と Patroni 構成の統合 FeikeSteenbergen

patronictl は、Patroni 自分自身と同じ構成ファイルを使用できます。

- Patroni が環境変数から構成を読み取れるようにします OleksiiKliukin

これにより、Patroni の構成を自動的に生成したり、さまざまなソースから単一の構成をマージしたりすることが簡単になります。

- API から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
- 使用可能なすべての DCS に *delete_cluster* を実装します。AlexanderKukushkin

patronictl での Etcd 以外の DCS のサポートを有効にします。

20.55 バージョン 0.80

2016-03-14 リリース

このリリースでは、**カスケードレプリケーション** のサポートを追加し、**スケジュールされたフェイルオーバー** を提供することにより Patroni 管理を簡素化します。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 をこれと組み合わせて使用することができます。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、Patroni 0.80 以降でのみ動作することに注意してください。

Cascading replication

- パトロニノード OleksiiKliukin の *replicatefrom* および *clonefrom* タグのサポートを追加します。

タグ *replicatefrom* により、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ます。*clonefrom* は、初期バックアップに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ます。

- 実行中のレプリケーション接続 OleksiiKliukin なしでもレプリカを初期化する実行レプリケーションメソッドのサポートを追加します。

これは、S3 または FTP に保存されているスナップショットからレプリカを作成するために役立ちます。実行中のレプリケーション接続を必要としないレプリケーション方法は、yaml 構成で *no_master:true* を提供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* パラメーター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す FeikeSteenbergen
- patronictl* の Zookeeper サポートを向上させました。OleksandrShulgin
- python-etcd 0.43 への移行 AlexanderKukushkin

Configuration

- Patroni(JanKeirse) のサンプルシステム構成スクリプトを追加します。
- Patroni が DB 接続の構成ファイルで指定されたスーパーユーザー名を無視する問題を修正しました。AlexanderKukushkin
- PatroniAlexanderKukushkin によって起動されたポストマスターの別のセッション ID とプロセスグループを作成することにより、CTRL-C の処理を修正します。

Tests

- 実行中の PatroniAlexanderKukushkin、OleksiiKliukin の現実のシナリオを確認するために、*behave* を使用して受け入れテストを追加します。

テストは、*behave* コマンドを使用して手動で起動できます。また、プル要求の場合およびコミット後にも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](#) _で見つけることができます。

第21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。

21.1 貢献するガイドライン

21.1.1 チャット

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](#).

21.1.2 バグの報告

バグを報告する前に、必ず **reproduce it with the latest Patroni version** を行ってください。また、問題が既に **reproduce it with the latest Patroni version** _に存在するかどうかを再確認してください。

21.1.3 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``で利用できる必要があります。
``PATH=/usr/lib/postgresql/11/bin:$PATHpython-mbehave`` のようなものを使用してパスに追加する必要がある場合があります。
3. 外部 DCS たとえば Etcd、Consul、Zookeeper などではテストする場合は、パッケージをインストールし、各サービスを実行してローカルホストとデフォルトポートで非暗号化/非保護の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、`behave` 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
```

21.1.4 tox を使用したテスト

tox テストを実行するには、1 つの依存関係 Python 以外をインストールするだけで済みます

```
pip install tox>=4
```

`behave` テストを実行する場合は、docker のインストールも必要です。

`tox.ini` の Tox 構成には、次のタスクを実行するための `"/environments/"` があります。

- lint`flake8` を使用した Python コード lint
- test`pytest` を使用して使用可能なすべての Python インタープリターの単体テスト、TTY が検出された場合に 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` を使用した静的型チェック
- black`black` を使用したコードフォーマット
- docker-build`behave` 環境に使用される docker イメージをビルドします
- docker-cmd 上記のイメージで任意のコマンドを実行します
- docker-behave-etcd 上記の画像を使用して動作テストの tox を実行します
- py*behave 使用可能な Python インタープリターで動作を実行します Docker なし、ただし、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する

ランニング tox

デフォルトの env リストを実行するには dep、lint、test、および docs、実行するだけです。

```
tox
```

test 環境は、ラベル test で実行できます。

```
tox -m test
```

behave の docker テストは、ラベル behave を使用して実行できます。

```
tox -m behave
```

同様に、docs には docs ラベルがあります。

他のすべての env は、それぞれの env 名で実行できます。

```
tox -e lint
tox -e py39-test-lin
```

factors を使用して部分的な env リストを選択することもできます。たとえば、Python3.10 のすべての env を実行したい場合

```
tox -f py310
```

これは、以下にリストされているすべての env を実行することと同じです。

```
$ 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) を使用して環境の設定済みの組み合わせをすべてリストすることができます

```
tox l
```

envs test および docs は、tox がアクティブなターミナルで実行されている場合、ジョブが完了すると HTML 出力ファイルを開こうとします。これは、この env をローカルで実行している開発者の利益を目的としています。Mac では open 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た場合、全体的な実行は失敗しません。この機能を無効にする場合は、環境変数 OPEN_CMD を no-op コマンドに設定します。

```
OPEN_CMD=: tox -m docs
```

動作テスト

`-mbehave`を使用した動作テストは、PG_MAJOR バージョン 11~16 に基づいて Docker イメージをビルドし、すべての動作テストを実行します。これは実行にかなりの時間がかかる場合があるため、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手順にスコープを制限することができます。

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な名前と behaveenv 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する場合は、動作するように位置引数を渡すことができます。これにより、Postgres のすべてのバージョンでウォッチドッグ動作機能テストシナリオが実行されます。

```
tox -m behave -- features/watchdog.feature
```

もちろん、2つを組み合わせることもできます。

21.1.5 プルリクエストの寄稿

1. リポジトリをフォークし、コードの変更を開発およびテストします。
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ル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。

プルリクエストに関するフィードバックをできるだけ早く取得します。

ハッピーパトロニのハッキング;-)

第 22 章 Indices and tables

- `genindex`
- `search`