



Patroni Quick Start Guide

リリース 3.0.2

EnterpriseDB Corporation

2023 年 04 月 05 日

目次

第 1 章	サポートされているトポロジ	3
1.1	Postgres	3
1.2	パトローニ	4
1.3	EtcD	4
1.4	ウォッチドッグ	4
1.5	HAProxy	4
第 2 章	セキュリティに関する考慮事項	7
2.1	‘Quick start on RHEL 8’	7
2.1.1	EtcD	7
2.1.2	EDB Postgres Advanced または Extended Server	7
2.1.3	ウォッチドッグ	9
2.1.4	パトローニ	9
2.1.5	HAProxy	13
2.2	‘Quick start on Debian 11’	15
2.2.1	EtcD	15
2.2.2	EDB Postgres Advanced または Extended Server	17
2.2.3	ウォッチドッグ	18
2.2.4	パトローニ	18
2.2.5	HAProxy	21
2.3	‘Installing Patroni’	24
2.3.1	Debian/Ubuntu	24
2.3.2	RHEL/CentOS	24
2.4	‘Installing and configuring etcD’	24
2.4.1	インストール	24
2.4.2	構成	26
2.5	‘Managing a cluster’	28
2.5.1	クラスター情報	28
2.5.2	Patroni ログ	29
2.5.3	メンテナンスモード	29
2.5.4	手動フェイルオーバーまたはスイッチオーバー	29
2.6	‘Tips and FAQs’	31
2.6.1	接続文字列	31
2.6.2	VIP & Keepalived	32

Patroni は、Postgres クラスターのレプリケーションとフェイルオーバーを管理するための最も一般的なオープンソースツールの 1 つです。Postgres のストリーミングレプリケーションとホットスタンバイ機能に依存しています。これらの機能により、DBA は、高可用性と読み取りスケーラビリティの要件があるクラスターのセットアップと管理のプロセスを簡素化できます。EDB は、Patroni を PostgreSQL、EDB Postgres Advanced Server、および EDB Postgres Extended Server でテストしました。

Patroni のドキュメントについては、<https://patroni.readthedocs.io> を参照してください。

第1章 サポートされているトポロジ

主な高可用性コンポーネント：

- Postgres クラスター通常、プライマリと2つ以上のレプリカで構成されるデータベースクラスター。

*パトローニフェイルオーバー管理ユーティリティとして使用します。

- etcd 構成、正常性、現在のステータス Etcd のクラスター情報を含む分散構成ストア (DCS) として使用されます。
- HAProxy 読み取り/書き込みノードと読み取り専用ノードに接続するためのプロキシとして使用されます。

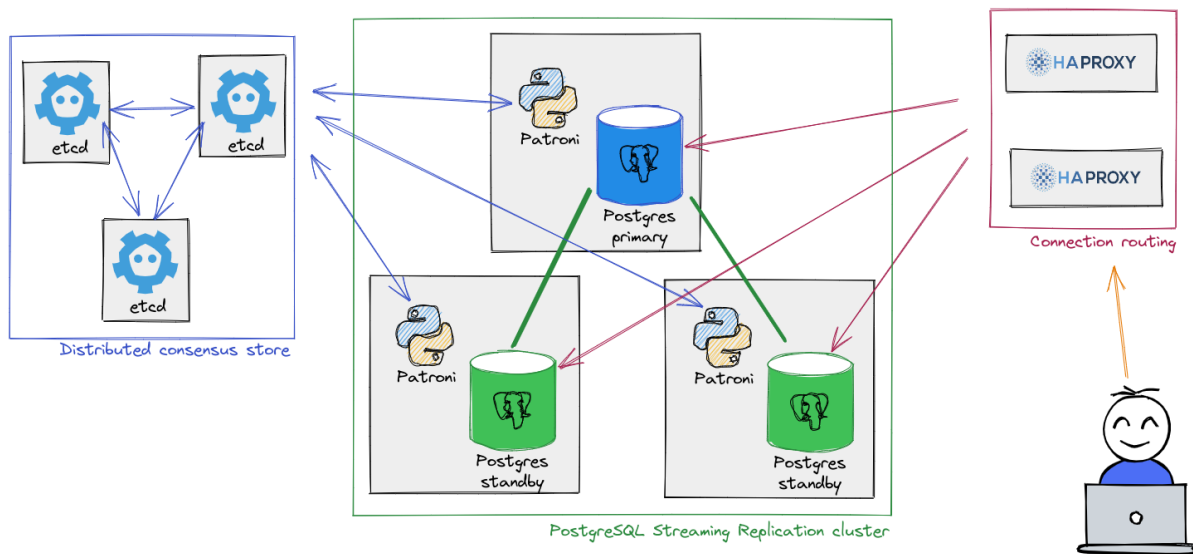


図 1: Supported topology

1.1 Postgres

EDB は現在、プライマリサーバーと2つのレプリカでの Patroni と Postgres の `_physical_` ストリーミングレプリケーションの使用をサポートしています。PostgreSQL、EDB Postgres Advanced Server、または EDB Postgres Extended Server の使用をサポートしています。

1.2 パトローニ

Postgres ノードに Patroni をインストールします。Patroni の [latest release](#) に対応しました。

1.3 Etcd

Etcd は、強い一貫性のある分散キー値ストアです。これは、分散システムまたはマシンのクラスターがアクセスする必要があるデータを保存するための信頼できる方法を提供します。

詳細については、[etcd documentation](#) を参照してください。

注釈: 実稼働環境の場合、3つのノードの etcd クラスターを設定することをお勧めします。1つの etcd クラスターは、複数の Postgres/Patroni クラスターに対応できます。Postgres ノードとは異なるホストに etcd ノードをインストールします。

etcd のインストールと設定については、'Installing and configuring etcd' を参照してください。

1.4 ウォッチドッグ

スプリットブレインの状況を回避するために、Patroni は **ウォッチドッグ** デバイスをサポートしています。

ウォッチドッグデバイスは、指定された時間枠内にキープアライブハートビートを取得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。このデバイスは、通常の Patroni スプリットブレイン保護メカニズムが失敗した場合に備えて、フェイルセーフのレイヤーを追加します。

注釈: Patroni でのウォッチドッグメカニズムの使用はオプションですが、実稼働環境での使用をお勧めします。

1.5 HAProxy

データベースサーバーに直接接続する代わりに、アプリケーションが代わりにプロキシに接続するように HAProxy を設定できます。次に、プロキシは要求を Postgres に転送します。この目的で HAProxy を使用すると、読み取り専用リクエストを 1 つ以上のレプリカにルーティングして負荷を分散できます。

スプリットブレインの場合のデータ損失を避けるために、HAProxy を使用することを強くお勧めします。

HAProxy は独立したサーバーとしてインストールできますが、アプリケーションサーバーまたはデータベースサーバーにインストールすることもできます。

注釈: 実稼働環境の場合、少なくとも 2 つの HAProxy サーバーを設定することをお勧めします。さまざまなポートを使用して、1 つの HAProxy サーバーは複数の Postgres/Patroni クラスターに対応できます。

第2章 セキュリティに関する考慮事項

セキュリティに関する考慮事項は、このドキュメントの範囲を超えています。

[Patroni documentation](#) で分散構成ストレージ（DCS）と Patroni REST

API の詳細を見つけることができます。

このドキュメントでは説明していませんが、HAProxy で HTTPS を使用できます。

2.1 ‘Quick start on RHEL 8’

このクイックスタートを使用して、RHEL 8 で EDB Postgres Advanced Server または EDB Postgres Extended Server バージョン 15 を使用して高可用性のデモクラスターを作成します。etcd ホストは `etcd1`、`etcd2`、および `etcd3` と呼ばれます。ストリーミングレプリケーションの 2 つのデータベースノードは `pg-patroni1` と `pg-patroni2` です。

2.1.1 Etcd

etcd をインストールして設定するには、'Installing and configuring etcd' を参照してください。

2.1.2 EDB Postgres Advanced または Extended Server

`pg-patroni1` と `pg-patroni2` の両方のホストで、好みのフレーバーの Postgres をインストールします。EDB リポジトリを使用してこれらの製品をインストールする方法の詳細については、

EDB Postgres Advanced Server または EDB Postgres Extended Server を参照してください。

選択した Postgres フレーバーによって環境変数が決まります。

EDB Postgres Advanced Server の場合、次の設定を使用します。

```
export PGPORT=5444
export PGUSER=enterprisedb
export PGGROUP=enterprisedb
export PGDATA="/var/lib/edb/as15/data"
export PGBIN="/usr/edb/as15/bin"
export PGSOCKET="/var/run/edb/as15"
```

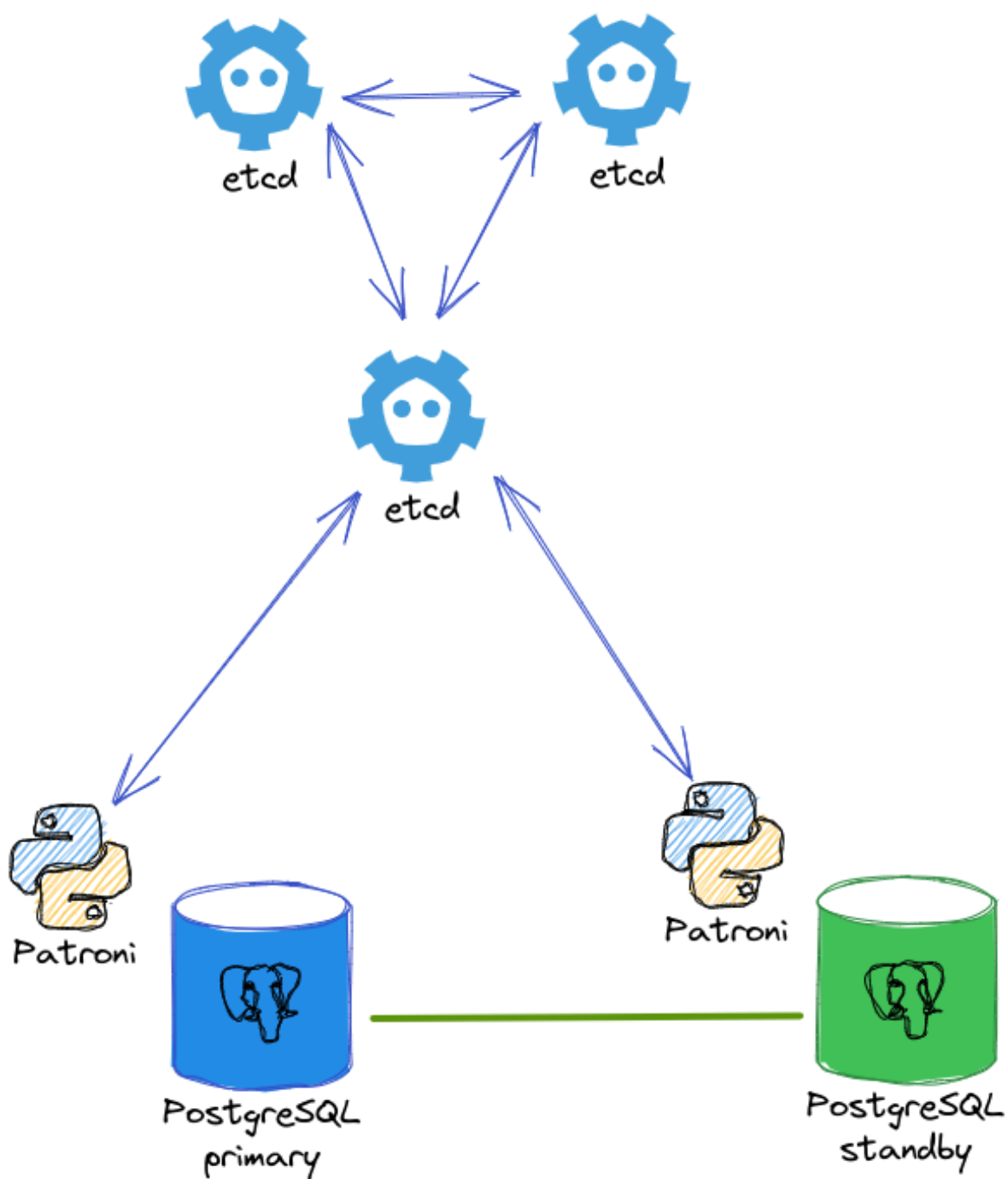


図 1: Demo diagram

EDB Postgres Extended Server の場合、次の設定を使用します。

```
export PGPOR=5432
export PGUSER=postgres
export PGGROUP=postgres
export PGDATA="/var/lib/edb-pge/15/data"
export PGBIN="/usr/edb/pge15/bin"
export PGSOCKET="/var/run/edb-pge"
```

Patroni は最初の Postgres クラスターをブートストラップ（作成）し、サービスの開始を担当するため、このサービスで `systemctl` が無効になっていることを確認します。

```
$ sudo systemctl disable edb-as-15.service
`` または

`` bash
$ sudo systemctl disable edb-pge-15.service
```

Postgres サービスへの接続を受け入れるように、各データベースノードでファイアウォールを構成します。

```
$ sudo firewall-cmd --quiet --zone=public --add-port=$PGPORT/tcp --permanent
$ sudo firewall-cmd --quiet --reload
```

2.1.3 ウォッチドッグ

Patroni は、ウォッチドッグデバイスと対話するコンポーネントです。pg-patroni1 と pg-patroni2 の両方のホストでソフトウェアウォッチドッグの権限を設定します。

```
$ cat <<EOF | sudo tee /etc/udev/rules.d/99-watchdog.rules
KERNEL=="watchdog", OWNER="$PGUSER", GROUP="$PGGROUP"
EOF
$ sudo sh -c echo "softdog" >> /etc/modules-load.d/softdog.conf
$ sudo modprobe softdog
$ sudo chown $PGUSER:$PGGROUP /dev/watchdog
```

2.1.4 パトロニー

pg-patroni1 と pg-patroni2 の両方のホストで、etcd の Patroni とその依存関係をインストールします。

'Installing Patroni' を参照してください。

/etc/patroni.yml で Patroni 構成を定義します。

```
$ CLUSTER_NAME="demo-cluster-1"
$ MY_NAME=$(hostname --short)
$ MY_IP=$(hostname -I | awk {print $1})
$ cat <<EOF | sudo tee /etc/patroni.yml
scope: $CLUSTER_NAME
namespace: /db/
name: $MY_NAME

restapi:
  listen: "0.0.0.0:8008"
  connect_address: "$MY_IP:8008"
  authentication:
    username: patroni
    password: mySupeSecretPassword

etcd3:
  hosts:
    - etcd1:2379
    - etcd2:2379
    - etcd3:2379

bootstrap:
  dcs:
    ttl: 30
    loop_wait: 10
    retry_timeout: 10
    maximum_lag_on_failover: 1048576
  postgresql:
    use_pg_rewind: true
    use_slots: true
    parameters:
      archive_mode: "on"
      archive_command: "/bin/true"

  initdb:
    - encoding: UTF8
    - data-checksums
    - auth-local: peer
    - auth-host: scram-sha-256

  pg_hba:
    - host replication replicator 0.0.0.0/0 scram-sha-256
    - host all all 0.0.0.0/0 scram-sha-256

# Some additional users which needs to be created after initializing new cluster
```

(次のページに続く)

(前のページからの続き)

```
users:
  admin:
    password: admin%
    options:
      - createrole
      - createdb

postgresql:
  listen: "0.0.0.0:$PGPORT"
  connect_address: "$MY_IP:$PGPORT"
  data_dir: $PGDATA
  bin_dir: $PGBIN
  pgpass: /tmp/pgpass0
  authentication:
    replication:
      username: replicator
      password: confidential
    superuser:
      username: $PGUSER
      password: my-super-password
  rewind:
    username: rewind_user
    password: rewind_password
  parameters:
    unix_socket_directories: "$PGSOCKET,/tmp"

watchdog:
  mode: required
  device: /dev/watchdog
  safety_margin: 5

tags:
  nofailover: false
  noloadbalance: false
  clonefrom: false
  nosync: false
EOF
```

注釈: この例の `postgresql: authentication:` セクションでは、初期化 (initdb) 中にスーパーユーザーが作成され、後で Patroni がデータベースに接続するために使用します。レプリケーションユーザーは、スタンバイサーバーが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するために作成および使用されます。最後に、巻き戻しユーザーが作成され、必要に応じて `pg_rewind` で使用されます。次に、パスワードポリシーに合わせて構成例のパスワードを変更することをお勧めします。

`$MY_IP` および `$MY_NAME` はローカルホストに固有です。それ以外の場合、`patroni.yml` 構成はすべての Patroni ノードで同じです。

Patroni は、`bin_dir` の場所で `postgres` バイナリを見つけますが、EDB Postgres Advanced Server バイナリは `edb-postgres` と呼ばれます。Patroni がそのバイナリを見つけられるようにするシンボリックリンクを作成します。

```
$ sudo ln -s /usr/edb/as15/bin/edb-postgres /usr/edb/as15/bin/postgres
```

Patroni のインストールソースによっては、`systemd file` が既に作成されている場合があります。正確なシステムユーザー、グループ、および構成パスを使用して独自のファイルを定義します。

```
$ cat <<EOF | sudo tee /etc/systemd/system/patroni.service
[Unit]
Description=Runners to orchestrate a high-availability Postgres
After=syslog.target network.target

[Service]
Type=simple
User=$PGUSER
Group=$PGGROUP
EnvironmentFile=-/etc/patroni_env.conf
ExecStart=patroni /etc/patroni.yml
ExecReload=/bin/kill -s HUP \${MAINPID}
KillMode=process
TimeoutSec=30
Restart=no

[Install]
WantedBy=multi-user.target
EOF

$ sudo systemctl daemon-reload
$ sudo systemctl enable patroni
$ sudo systemctl start patroni
```

最初に `pg-patroni1` で Patroni サービスを開始して、Postgres クラスターのリーダーにします。次に、`pg-patroni2` を起動します。

クラスターのメンバーをリストするには、`patronictl` コマンドを使用します。

```
$ patronictl -c /etc/patroni.yml list
+ Cluster: demo-cluster-1 (7104275668989116902) -----+
| Member          | Host              | Role    | State  | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| pg-patroni1     | 192.168.121.61    | Leader  | running | 1 |           |
| pg-patroni2     | 192.168.121.135   | Replica | running | 1 |           0 |
+-----+-----+-----+-----+-----+-----+

```

Patroni REST API はポート 8008 で利用できます。必要に応じてファイアウォールを構成します。

```
$ sudo firewall-cmd --quiet --zone=public --add-port=8008/tcp --permanent
$ sudo firewall-cmd --quiet --reload
```

2.1.5 HAProxy

PGDG yum extras リポジトリには haproxy パッケージが含まれています。

システムで yum.postgresql.org を設定して使用するには、[PostgreSQL Linux downloads website](#) を参照してください。

この例では、pg-patroni1 と pg-patroni2 の両方のホストに HAProxy をインストールします。

```
$ sudo dnf --enablerepo=pgdg-rhel8-extras install -y haproxy
$ sudo cp /etc/haproxy/haproxy.cfg /etc/haproxy/haproxy.cfg.bck
$ cat <<EOF | sudo tee /etc/haproxy/haproxy.cfg
global
    maxconn 100

defaults
    log      global
    mode     tcp
    retries  2
    timeout  client 30m
    timeout  connect 4s
    timeout  server 30m
    timeout  check 5s

listen stats
    mode http
    bind *:7000
    stats enable
    stats uri /

listen read-write
    bind *:5000
    option httpchk OPTIONS /read-write
    http-check expect status 200
    default-server inter 3s fall 3 rise 2 on-marked-down shutdown-sessions
    server pg-patroni1 pg-patroni1:$PGPORT maxconn 100 check port 8008
    server pg-patroni2 pg-patroni2:$PGPORT maxconn 100 check port 8008

listen read-only
    balance roundrobin
    bind *:5001
```

(次のページに続く)

(前のページからの続き)

```

option httpchk OPTIONS /replica
http-check expect status 200
default-server inter 3s fall 3 rise 2 on-marked-down shutdown-sessions
server pg-patroni1 pg-patroni1:$PGPORT maxconn 100 check port 8008
server pg-patroni2 pg-patroni2:$PGPORT maxconn 100 check port 8008
EOF

$ sudo setsebool -P haproxy_connect_any=1
$ sudo systemctl enable haproxy
$ sudo systemctl start haproxy

```

ポート 50000 を使用する読み取り-書き込みと、ポート 50001 を使用する読み取り専用の 2 つのセクションがあります。すべての Postgres ホストは、すべてプライマリまたはスタンバイの潜在的な候補であるため、両方のセクションに含まれています。HAproxy が各ホストに現在持っているロールを知るには、Patroni REST API を照会します。

```

$ curl -s http://pg-patroni1:8008
{
  "state": "running",
  "postmaster_start_time": "2023-03-21 11:23:54.709344+00:00",
  "role": "master",
  "server_version": 150002,
  "xlog": {
    "location": 67112696
  },
  "timeline": 1,
  "replication": [
    {
      "username": "replicator",
      "application_name": "pg-patroni2",
      "client_addr": "192.168.121.135",
      "state": "streaming",
      "sync_state": "async",
      "sync_priority": 0
    }
  ],
  "dcs_last_seen": 1679401036,
  "database_system_identifier": "7104275668989116902",
  "patroni": {
    "version": "3.0.1",
    "scope": "demo-cluster-1"
  }
}

```

注釈: `/replica` エンドポイントを使用することにより、Patroni はスタンバイサーバーのみにリダイレクト

します。 `/read-only` を使用して、プライマリも含めます。

HAProxy がリッスンするポートにアクセスする必要がある場合は、ファイアウォールを構成します。

```
$ sudo firewall-cmd --quiet --zone=public --add-port=7000/tcp --permanent
$ sudo firewall-cmd --quiet --zone=public --add-port=5000/tcp --permanent
$ sudo firewall-cmd --quiet --zone=public --add-port=5001/tcp --permanent
$ sudo firewall-cmd --quiet --reload
```

ポート 5000 とポート 5001 を使用して、読み書き接続または読み取り専用接続を選択できるようになりました。

```
$ psql -U admin -d postgres -h pg-patroni1 -p 5000 -c "SELECT pg_is_in_recovery();"
pg_is_in_recovery
-----
f
(1 row)

$ psql -U admin -d postgres -h pg-patroni1 -p 5001 -c "SELECT pg_is_in_recovery();"
pg_is_in_recovery
-----
t
(1 row)
```

2.2 ‘Quick start on Debian 11’

このクイックスタートを使用して、Debian 11 で EDB Postgres Advanced Server または EDB Postgres Extended Server バージョン 15 を使用して高可用性のデモクラスターを作成します。etcd ホストは `etcd1`、`etcd2`、および `etcd3` と呼ばれます。ストリーミングレプリケーションの 2 つのデータベースノードは `pg-patroni1` と `pg-patroni2` です。

2.2.1 Etcd

etcd をインストールして設定するには、'Installing and configuring etcd' を参照してください。

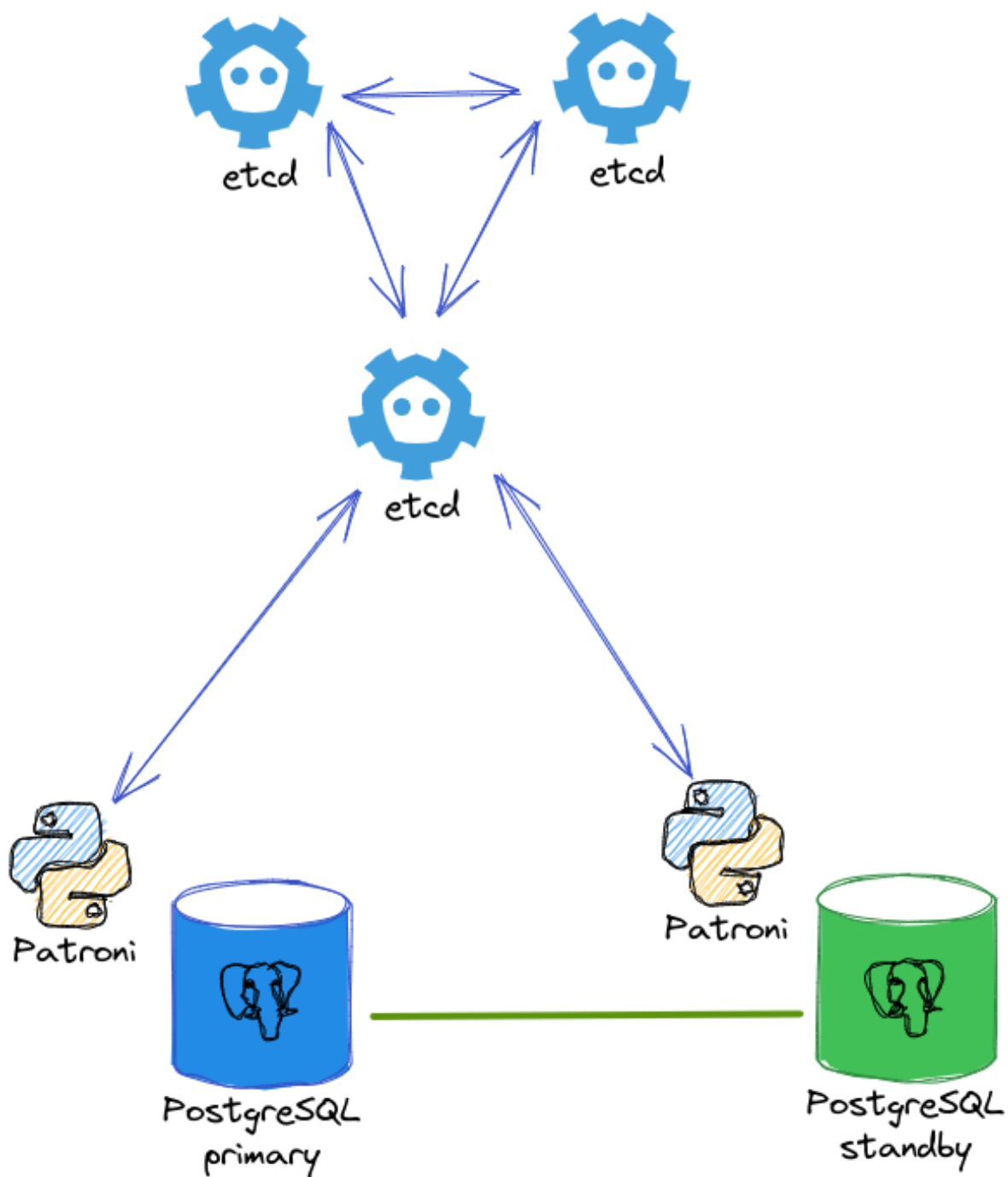


図 2: Demo diagram

2.2.2 EDB Postgres Advanced または Extended Server

pg-patroni1 と pg-patroni2 の両方のホストで、お好みの Postgres フレーバーをインストールします。EDB リポジトリを使用してこれらの製品をインストールする方法の詳細については、

EDB Postgres Advanced Server または EDB Postgres Extended Server を参照してください。

選択した Postgres フレーバーによって環境変数が決まります。

EDB Postgres Advanced Server の場合、次の設定を使用します。

```
export PGPOR=5444
export PGUSER=enterprisedb
export PGGROUP=enterprisedb
export PGDATA="/var/lib/edb-as/15/main"
export PGBIN="/usr/lib/edb-as/15/bin"
export PGSCKET="/var/run/edb-as"
```

EDB Postgres Extended Server の場合、次の設定を使用します。

```
export PGPOR=5432
export PGUSER=postgres
export PGGROUP=postgres
export PGDATA="/var/lib/edb-pge/15/data"
export PGBIN="/usr/lib/edb-pge/15/bin"
export PGSCKET="/var/run/edb-pge"
```

Patroni は、最初の Postgres クラスターをブートストラップ（作成）し、サービスの開始を担当します。このサービスで systemctl が無効になっていることを確認し、インストール中に作成された Postgres クラスターを削除します。

```
$ sudo systemctl stop edb-as@15-main.service
$ sudo systemctl disable edb-as@15-main.service
$ sudo rm -fr /var/lib/edb-as/15/main
`` または
`` bash
$ sudo systemctl stop edb-pge-15.service
$ sudo systemctl disable edb-pge-15.service
$ sudo rm -fr /var/lib/edb-pge/15/data
```

2.2.3 ウォッチドッグ

Patroni は、ウォッチドッグデバイスと対話するコンポーネントです。pg-patroni1 と pg-patroni2 の両方のホストでソフトウェアウォッチドッグの権限を設定します。

```
$ cat <<EOF | sudo tee /etc/udev/rules.d/99-watchdog.rules
KERNEL=="watchdog", OWNER="$PGUSER", GROUP="$PGGROUP"
EOF
$ sudo sh -c echo "softdog" >> /etc/modules-load.d/softdog.conf
$ sudo modprobe softdog
$ sudo chown $PGUSER:$PGGROUP /dev/watchdog
```

2.2.4 パトローニ

pg-patroni1 と pg-patroni2 の両方のホストで、etcd の Patroni とその依存関係をインストールします。

'Installing Patroni' を参照してください。

/etc/patroni.yml で Patroni 構成を定義します。

```
$ CLUSTER_NAME="demo-cluster-1"
$ MY_NAME=$(hostname --short)
$ MY_IP=$(hostname -I | awk {print $1})
$ cat <<EOF | sudo tee /etc/patroni.yml
scope: $CLUSTER_NAME
namespace: /db/
name: $MY_NAME

restapi:
  listen: "0.0.0.0:8008"
  connect_address: "$MY_IP:8008"
  authentication:
    username: patroni
    password: mySupeSecretPassword

etcd3:
  hosts:
    - etcd1:2379
    - etcd2:2379
    - etcd3:2379

bootstrap:
  dcs:
    ttl: 30
    loop_wait: 10
    retry_timeout: 10
```

(次のページに続く)

(前のページからの続き)

```
maximum_lag_on_failover: 1048576
postgresql:
  use_pg_rewind: true
  use_slots: true
  parameters:
    archive_mode: "on"
    archive_command: "/bin/true"

initdb:
- encoding: UTF8
- data-checksums
- auth-local: peer
- auth-host: scram-sha-256

pg_hba:
- host replication replicator 0.0.0.0/0 scram-sha-256
- host all all 0.0.0.0/0 scram-sha-256

# Some additional users which needs to be created after initializing new cluster
users:
  admin:
    password: admin%
    options:
      - createrole
      - createdb

postgresql:
  listen: "0.0.0.0:$PGPORT"
  connect_address: "$MY_IP:$PGPORT"
  data_dir: $PGDATA
  bin_dir: $PGBIN
  pgpass: /tmp/pgpass0
  authentication:
    replication:
      username: replicator
      password: confidential
    superuser:
      username: $PGUSER
      password: my-super-password
  rewind:
    username: rewind_user
    password: rewind_password
  parameters:
    unix_socket_directories: "$PGSOCKET,/tmp"
```

(次のページに続く)

(前のページからの続き)

```

watchdog:
  mode: required
  device: /dev/watchdog
  safety_margin: 5

tags:
  nofailover: false
  noloadbalance: false
  clonefrom: false
  nosync: false
EOF

```

注釈: この例の ``postgresql: authentication:`` セクションでは、初期化 (`initdb`) 中にスーパーユーザーが作成され、後で Patroni がデータベースに接続するために使用します。レプリケーションユーザーは、スタンバイサーバーが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するために作成および使用されます。最後に、巻き戻しユーザーが作成され、必要に応じて `pg_rewind` で使用されます。次に、パスワードポリシーに合わせて構成例のパスワードを変更することをお勧めします。

`$MY_IP` および `$MY_NAME` はローカルホストに固有です。それ以外の場合、`patroni.yml` 構成はすべての Patroni ノードで同じです。

Patroni は、`bin_dir` の場所で `postgres` バイナリを見つけますが、EDB Postgres Advanced Server バイナリは `edb-postgres` と呼ばれます。Patroni がそのバイナリを見つけられるようにするシンボリックリンクを作成します。

```
$ sudo ln -s /usr/lib/edb-as/15/bin/edb-postgres /usr/lib/edb-as/15/bin/postgres
```

Patroni のインストールソースによっては、`systemd file` が既に作成されている場合があります。正確なシステムユーザー、グループ、および構成パスを使用して独自のファイルを定義します。

```

$ cat <<EOF | sudo tee /etc/systemd/system/patroni.service
[Unit]
Description=Runners to orchestrate a high-availability Postgres
After=syslog.target network.target

[Service]
Type=simple
User=$PGUSER
Group=$PGGROUP
EnvironmentFile=-/etc/patroni_env.conf
ExecStart=patroni /etc/patroni.yml
ExecReload=/bin/kill -s HUP \${MAINPID}
KillMode=process
TimeoutSec=30

```

(次のページに続く)

(前のページからの続き)

```
Restart=no

[Install]
WantedBy=multi-user.target
EOF

$ sudo systemctl daemon-reload
$ sudo systemctl enable patroni
$ sudo systemctl start patroni
```

最初に `pg-patroni1` で Patroni サービスを開始して、Postgres クラスターのリーダーにしてから、`pg-patroni2` を起動します。

クラスターのメンバーをリストするには、`patronictl` コマンドを使用します。

```
$ patronictl -c /etc/patroni.yml list
+ Cluster: demo-cluster-1 (7104275668989116902) ----+-----+-----+
| Member      | Host           | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| pg-patroni1 | 192.168.121.61 | Leader | running | 1  |           |
| pg-patroni2 | 192.168.121.135 | Replica | running | 1  |           0 |
+-----+-----+-----+-----+-----+-----+

```

Patroni REST API はポート `8008` で利用できます。

EDB Postgres Advanced Server では、データベースクラスターの実行後、他の `systemd` の依存関係のリセットが必要になる場合があります。次のコマンドを使用して、リセットを実行します。

```
$ sudo systemctl restart edb-as15-pgagent.service
$ sudo systemctl reset-failed
```

2.2.5 HAProxy

この例では、`pg-patroni1` と `pg-patroni2` の両方のホストに HAProxy をインストールします。

```
$ sudo apt-get install -y haproxy
$ sudo systemctl stop haproxy
$ sudo cp /etc/haproxy/haproxy.cfg /etc/haproxy/haproxy.cfg.bck
$ cat <<EOF | sudo tee /etc/haproxy/haproxy.cfg
global
    maxconn 100

defaults
    log      global
    mode     tcp
```

(次のページに続く)

(前のページからの続き)

```

    retries 2
    timeout client 30m
    timeout connect 4s
    timeout server 30m
    timeout check 5s

listen stats
    mode http
    bind *:7000
    stats enable
    stats uri /

listen read-write
    bind *:5000
    option httpchk OPTIONS /read-write
    http-check expect status 200
    default-server inter 3s fall 3 rise 2 on-marked-down shutdown-sessions
    server pg-patroni1 pg-patroni1:$PGPORT maxconn 100 check port 8008
    server pg-patroni2 pg-patroni2:$PGPORT maxconn 100 check port 8008

listen read-only
    balance roundrobin
    bind *:5001
    option httpchk OPTIONS /replica
    http-check expect status 200
    default-server inter 3s fall 3 rise 2 on-marked-down shutdown-sessions
    server pg-patroni1 pg-patroni1:$PGPORT maxconn 100 check port 8008
    server pg-patroni2 pg-patroni2:$PGPORT maxconn 100 check port 8008
EOF

$ sudo systemctl enable haproxy
$ sudo systemctl start haproxy

```

ポート 5000 を使用する読み取り-書き込みと、ポート 5001 を使用する読み取り専用の 2 つのセクションがあります。すべての Postgres ホストは、すべてプライマリまたはスタンバイの潜在的な候補であるため、両方のセクションに含まれています。HAProxy は、各ホストが現在持っているロールを知るために、Patroni REST API を照会します。

```

$ curl -s http://pg-patroni1:8008
{
  "state": "running",
  "postmaster_start_time": "2022-06-30 07:45:55.427895+00:00",
  "role": "master",
  "server_version": 140400,
  "xlog": {

```

(次のページに続く)

(前のページからの続き)

```

    "location": 67108960
  },
  "timeline": 1,
  "replication": [
    {
      "username": "replicator",
      "application_name": "pg-patroni2",
      "client_addr": "192.168.121.229",
      "state": "streaming",
      "sync_state": "async",
      "sync_priority": 0
    }
  ],
  "dcs_last_seen": 1656575356,
  "database_system_identifier": "7114936110138668110",
  "patroni": {
    "version": "2.1.3",
    "scope": "demo-cluster-1"
  }
}

```

注釈: */replica* エンドポイントを使用することにより、Patroni はスタンバイサーバーのみにリダイレクトします。*/read-only* を使用して、プライマリも含めます。

ポート 5000 とポート 5001 を使用して、読み書き接続または読み取り専用接続を選択できるようになりました。

```

$ psql -U admin -d postgres -h pg-patroni1 -p 5000 -c "SELECT pg_is_in_recovery();"
pg_is_in_recovery
-----
f
(1 row)

$ psql -U admin -d postgres -h pg-patroni1 -p 5001 -c "SELECT pg_is_in_recovery();"
pg_is_in_recovery
-----
t
(1 row)

```

2.3 ‘Installing Patroni’

Patroni パッケージは、PGDG apt および yum リポジトリを介して提供されます。

サポートされている OS リストについては、

[Platform Compatibility](#) を参照してください（現在 Linux x86-64 (amd64) のみ）。

2.3.1 Debian/Ubuntu

Patroni をインストールするには、PostgreSQL apt リポジトリを構成します。次に、次を実行します。

```
$ sudo apt-get install -y patroni
```

より詳細な構成例については、'Quick start on Debian 11' を参照してください。

2.3.2 RHEL/CentOS

PostgreSQL yum リポジトリから Patroni をインストールできます。

Enterprise Linux 用の Extra Packages（[EPEL](#)）が必要です。

リポジトリを構成したら、次のコマンドを実行して、etcd の Patroni とその依存関係をインストールします。

```
$ sudo dnf install -y patroni patroni-etcd
```

より詳細な構成例については、Quick start on RHEL8 を参照してください。

2.4 ‘Installing and configuring etcd’

2.4.1 インストール

Debian/Ubuntu パッケージ

公式の Debian/Ubuntu apt リポジトリから etcd をインストールするには：

```
$ sudo apt-get -y install etcd
```

インストールされているバージョンと利用可能な最新バージョンを確認できます。インストールされているバージョンが古く、最新のバージョンをインストールする場合は、[ビルド済みバイナリ](#) メソッドを使用します。

RedHat 互換パッケージ

PGDG yum [extras](#) リポジトリには etcd パッケージが含まれています。

システムで yum.postgresql.org を設定して使用するには、[PostgreSQL Linux downloads website](#) を参照してください。

システムに応じて、次のリポジトリを有効にします。

- RHEL 9 : pgdg-rhel9-extras
- RHEL 8 : pgdg-rhel8-extras
- RHEL 7 : pgdg-rhel7-extras

たとえば、次のコマンドは etcd を RHEL 8 にインストールします。

```
$ sudo dnf --enablerepo=pgdg-rhel8-extras install -y etcd
```

インストールされているバージョンと利用可能な最新バージョンを確認できます。インストールされているバージョンが古く、最新のバージョンをインストールする場合は、[ビルド済みバイナリ](#) メソッドを使用します。

ビルド済みバイナリ

これらのコマンドは、インストールされた etcd バイナリのバージョンを出力し、etcd-io Web サイトから最新バージョンを取得します。

```
$ etcd --version |head -1
etcd Version: 3.5.7

$ ETCD_RELEASE=$(curl -s https://api.github.com/repos/etcd-io/etcd/releases/
→latest|grep tag_name | cut -d " " -f 4)
$ echo "Latest etcd version released: $ETCD_RELEASE"
Latest etcd version released: v3.5.7
```

etcd の最新リリースをインストールする最も簡単な方法は、ビルド済みバイナリからのものです。

amd64 ホストでは、次のコマンドを使用します。

```
$ ETCD_RELEASE=$(curl -s https://api.github.com/repos/etcd-io/etcd/releases/
→latest|grep tag_name | cut -d " " -f 4)
$ curl -sL https://github.com/etcd-io/etcd/releases/download/${ETCD_RELEASE}/etcd-
→${ETCD_RELEASE}-linux-amd64.tar.gz \
| sudo tar xz -C /usr/bin --strip=1 --wildcards --no-anchored etcdctl etcd
$ etcd --version |head -1
etcd Version: 3.5.7
```

次に、データディレクトリとシステムユーザーを作成します。

```
$ sudo mkdir -p /var/lib/etcd/
$ sudo mkdir /etc/etcd
$ sudo groupadd --system etcd
$ sudo useradd -s /sbin/nologin --system -g etcd etcd
$ sudo chown -R etcd:etcd /var/lib/etcd/
$ sudo chmod -R a+rw /var/lib/etcd
```

systemd ユニットファイルを構成します。

```
$ cat <<EOF | sudo tee /etc/systemd/system/etcd.service
[Unit]
Description=etcd key-value store
Documentation=https://github.com/etcd-io/etcd
After=network-online.target local-fs.target remote-fs.target time-sync.target
Wants=network-online.target local-fs.target remote-fs.target time-sync.target

[Service]
User=etcd
Type=notify
Environment=ETCD_DATA_DIR=/var/lib/etcd
Environment=ETCD_NAME=%H
EnvironmentFile=-/etc/etcd/etcd.conf
ExecStart=/usr/bin/etcd
Restart=always
RestartSec=10s
LimitNOFILE=40000

[Install]
WantedBy=multi-user.target
EOF

$ sudo systemctl daemon-reload
```

2.4.2 構成

必要に応じてファイアウォールを構成します。

```
$ sudo firewall-cmd --quiet --zone=public --add-port=2379/tcp --permanent
$ sudo firewall-cmd --quiet --zone=public --add-port=2380/tcp --permanent
$ sudo firewall-cmd --quiet --reload
```

新しい etcd クラスターを構成して作成するには、ETCD_INITIAL_CLUSTER 変数を設定して初期クラスター構成を定義する必要があります。そのリストには、すべての etcd ホストとその IP アドレスが含まれている必要があります。

この例では、3 つの etcd ホストを示しています。

```
$ ETCD_INITIAL_CLUSTER="etcd1=http://192.168.121.35:2380,etcd2=http://192.168.121.
→245:2380,etcd3=http://192.168.121.191:2380"
```

動的に取得するには、次の単純なシェルスクリプトを使用できます。etcd1 etcd2 etcd3 は 3 つの etcd ホストです。

```
# Fetch the IP addresses of all etcd hosts

etcd_nodes=( etcd1 etcd2 etcd3 )
i=0
for node in "${etcd_nodes[@]}"
do
    i=$((i+1))
    target_ip=$((dig +short $node))
    target_array[$i]="$node=http://$target_ip:2380"
done
ETCD_CLUSTER_URL=$(printf ",%s" "${target_array[@]}")
export ETCD_CLUSTER_URL=${ETCD_CLUSTER_URL:1}
echo "ETCD_CLUSTER_URL=\\"$ETCD_CLUSTER_URL\\""
```

次に、ローカル etcd 構成を設定し、サービスを開始します。

```
$ MY_IP=$(hostname -I | awk {print $1})
$ MY_NAME=$(hostname --short)
$ cat <<EOF | sudo tee /etc/etcd/etcd.conf
# [Member]
ETCD_LISTEN_PEER_URLS="http://$MY_IP:2380"
ETCD_LISTEN_CLIENT_URLS="http://localhost:2379,http://$MY_IP:2379"
ETCD_NAME="$MY_NAME"
# [Clustering]
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://$MY_IP:2380"
ETCD_INITIAL_CLUSTER="$ETCD_CLUSTER_URL"
ETCD_ADVERTISE_CLIENT_URLS="http://$MY_IP:2379"
ETCD_INITIAL_CLUSTER_TOKEN="etcd-cluster-1"
ETCD_INITIAL_CLUSTER_STATE="new"
EOF
$ sudo systemctl enable etcd.service
$ sudo systemctl start etcd.service
```

注釈: ビルド済みバイナリ方法または Red Hat 互換パッケージを使用する場合、構成ファイルは `/etc/etcd/etcd.conf` です。Debian/Ubuntu パッケージを使用する場合、構成ファイルは `/etc/default/etcd` です。

サービスを開始し、ポート 2379 および 2380 でリッスンします。クラスターのステータスを確認します。

```
$ ss -tunelp | grep 2379
tcp    LISTEN 0      4096      127.0.0.1:2379      0.0.0.0:*    ...
tcp    LISTEN 0      4096    192.168.121.7:2379  0.0.0.0:*    ...

$ ss -tunelp | grep 2380
tcp    LISTEN 0      4096    192.168.121.245:2380  0.0.0.0:*    ...

$ etcdctl member list
3078f511def8f5a9, started, etcd1, http://192.168.121.7:2380, http://192.168.121.
→7:2379, false
493bb0ea8338d60a, started, etcd2, http://192.168.121.174:2380, http://192.168.121.
→174:2379, false
bfdda4bcab677625, started, etcd3, http://192.168.121.226:2380, http://192.168.121.
→226:2379, false

$ etcdctl endpoint health
127.0.0.1:2379 is healthy: successfully committed proposal: took = 2.103463ms

$ etcdctl endpoint status
127.0.0.1:2379, 3078f511def8f5a9, 3.5.4, 29 kB, true, false, 2, 12, 12,
```

2.5 ‘Managing a cluster’

2.5.1 クラスター情報

クラスターに関する情報を取得するには、list または topology エンドポイントを使用します。

```
$ patronictl -c /etc/patroni.yml list
+ Cluster: demo-cluster-1 (7106835957012965359) -----+-----+-----+
| Member          | Host                | Role    | State  | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| pg-patroni1     | xxx.xxx.xxx.xxx     | Leader  | running | 1 |           |
| pg-patroni2     | xxx.xxx.xxx.xxx     | Replica | running | 1 |           0 |
+-----+-----+-----+-----+-----+-----+

$ patronictl -c /etc/patroni.yml topology
+ Cluster: demo-cluster-1 (7106835957012965359) -----+-----+-----+
| Member          | Host                | Role    | State  | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| pg-patroni1     | xxx.xxx.xxx.xxx     | Leader  | running | 1 |           |
| + pg-patroni2   | xxx.xxx.xxx.xxx     | Replica | running | 1 |           0 |
+-----+-----+-----+-----+-----+-----+
```

2.5.2 Patroni ログ

Patroni ログは systemd によってシステムログに送信されます。journalctl システムユーティリティを使用して取得します。

```
$ sudo journalctl -u patroni.service -n 100 -f
Jun 08 12:26:29 pg-patroni1 patroni[26589]: 2022-06-08 12:26:29,000 INFO: no action.
→ I am (pg-patroni1), a secondary, and following a leader (pg-patroni2)
...
```

2.5.3 メンテナンスモード

クラスター管理を一時停止または再開するには、次を使用します。

```
$ patronictl -c /etc/patroni.yml pause
Success: cluster management is paused

$ patronictl -c /etc/patroni.yml list
+ Cluster: demo-cluster-1 (7106835957012965359) -----+
| Member          | Host              | Role    | State  | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| pg-patroni1     | xxx.xxx.xxx.xxx   | Leader  | running | 1 |           |
| pg-patroni2     | xxx.xxx.xxx.xxx   | Replica | running | 1 |           0 |
+-----+-----+-----+-----+-----+
Maintenance mode: on

$ patronictl -c /etc/patroni.yml resume
Success: cluster management is resumed
```

2.5.4 手動フェイルオーバーまたはスイッチオーバー

フェールオーバーエンドポイントを使用すると、正常なノードがない場合に手動フェールオーバーを実行できます。スイッチオーバーをスケジュールすることはできません。

```
Usage: patronictl failover [OPTIONS] [CLUSTER_NAME]

Failover to a replica

Options:
  --master TEXT      The name of the current master
  --candidate TEXT   The name of the candidate
  --force            Do not ask for confirmation at any point
  --help            Show this message and exit.
```

スイッチオーバーエンドポイントは、クラスターが正常な場合、つまりリーダーがいる場合にのみ機能します。これにより、スイッチオーバーをスケジュールできます。

Usage: patronictl switchover [OPTIONS] [CLUSTER_NAME]

Switchover to a replica

Options:

```
--master TEXT      The name of the current master
--candidate TEXT   The name of the candidate
--scheduled TEXT   Timestamp of a scheduled switchover in unambiguous format
                   (e.g. ISO 8601)
--force            Do not ask for confirmation at any point
--help            Show this message and exit.
```

例:

```
$ patronictl -c /etc/patroni.yml switchover --candidate pg-patroni2
Master [pg-patroni1]:
When should the switchover take place (e.g. 2022-06-08T13:14 ) [now]:
Current cluster topology
+ Cluster: demo-cluster-1 (7106835957012965359) ----+-----+
| Member      | Host           | Role   | State  | TL | Lag in MB |
+-----+-----+
| pg-patroni1 | xxx.xxx.xxx.xxx | Leader | running | 1 |          |
| pg-patroni2 | xxx.xxx.xxx.xxx | Replica | running | 1 |          0 |
+-----+-----+
Are you sure you want to switchover cluster demo-cluster-1, demoting current master
pg-patroni1? [y/N]: y
2022-06-08 12:14:48.01625 Successfully switched over to "pg-patroni2"
+ Cluster: demo-cluster-1 (7106835957012965359) ----+-----+
| Member      | Host           | Role   | State  | TL | Lag in MB |
+-----+-----+
| pg-patroni1 | xxx.xxx.xxx.xxx | Replica | stopped |   | unknown   |
| pg-patroni2 | xxx.xxx.xxx.xxx | Leader | running | 1 |          |
+-----+-----+

$ patronictl -c /etc/patroni.yml list
+ Cluster: demo-cluster-1 (7106835957012965359) ----+-----+
| Member      | Host           | Role   | State  | TL | Lag in MB |
+-----+-----+
| pg-patroni1 | xxx.xxx.xxx.xxx | Replica | running | 2 |          0 |
| pg-patroni2 | xxx.xxx.xxx.xxx | Leader | running | 2 |          |
+-----+-----+

$ patronictl -c /etc/patroni.yml history
```

(次のページに続く)

(前のページからの続き)

TL	LSN	Reason	Timestamp
New Leader			
1	67109024	no recovery target specified	2022-06-08T12:14:47.785857+00:00
pg-patroni2			

2.6 ‘Tips and FAQs’

2.6.1 接続文字列

libpq および jdbc のような PostgreSQL

`client libraries` は、クライアント接続フェイルオーバーをサポートしています。接続文字列には、複数のサーバーが含まれています（例、`host=svr1,svr2`

）。クライアントライブラリは、使用可能なホストをループして、使用可能で読み取り/書き込みまたは読み取り専用操作が可能な接続を見つけます。この構成により、クライアントはスイッチオーバー中にプライマリクラスタをたどることができます。

例：

```
$ psql "host=svr1,svr2 dbname=postgres user=admin target_session_attrs=read-write" -c
->"SELECT pg_is_in_recovery();"
pg_is_in_recovery
-----
f
(1 row)

$ psql "host=svr1,svr2 dbname=postgres user=admin target_session_attrs=read-only" -c
->"SELECT pg_is_in_recovery();"
pg_is_in_recovery
-----
t
(1 row)
```

2.6.2 VIP & Keepalived

複数の HAProxy インスタンスを使用していて、接続するプロキシホストを決定する方法が必要な場合、最も簡単な解決策は、Keepalived と VIP（仮想 IP）を使用することです。

RHEL8 の例

この例では、RHEL8 でのインストールと構成を示しています。

最初に、HAProxy を実行している各ホストに Keepalived をインストールします。

```
$ sudo dnf install -y keepalived
$ sudo cp /etc/keepalived/keepalived.conf /etc/keepalived/keepalived.conf.bck
```

この例では、仮想 IP として 192.168.121.121 を使用します。Keepalived を構成します。

```
$ VIP=192.168.121.121
$ MY_IP=$(hostname -I | awk {print $1})
$ DEVICE=$(ip -br address | grep $MY_IP | awk {print $1})
$ cat <<EOF | sudo tee /etc/keepalived/keepalived.conf
global_defs {
    vrrp_garp_master_delay 3
    vrrp_garp_master_repeat 4
    vrrp_garp_master_refresh 60
    vrrp_garp_master_refresh_repeat 4
}
vrrp_instance patroni {
    state BACKUP
    interface $DEVICE
    virtual_router_id 100
    priority 100
    advert_int 1
    virtual_ipaddress {
        $VIP
    }
    authentication {
        auth_type PASS
        auth_pass secret
    }
}
EOF
```

必要に応じてファイアウォールを構成します。

```
$ sudo firewall-cmd --add-rich-rule=rule protocol value="vrrp" accept --permanent
$ sudo firewall-cmd --reload
```

最後に、Keepalived サービスを有効にして開始します。

```
$ sudo systemctl enable keepalived
$ sudo systemctl start keepalived
```

起動すると、仮想 IP がいずれかのホストに接続されます。確認するには、それに到達してみてください。

```
$ ping -c 3 $VIP
PING 192.168.121.121 (192.168.121.121) 56(84) bytes of data.
64 bytes from 192.168.121.121: icmp_seq=1 ttl=64 time=0.196 ms
64 bytes from 192.168.121.121: icmp_seq=2 ttl=64 time=0.410 ms
64 bytes from 192.168.121.121: icmp_seq=3 ttl=64 time=0.322 ms

- -- 192.168.121.121 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2052ms
rtt min/avg/max/mdev = 0.196/0.309/0.410/0.087 ms
```