



EDB

Patroni Quick Start Guide

Release 3.0.2

EnterpriseDB Corporation

Apr 05, 2023

CONTENTS

1	Supported topology	3
1.1	Postgres	4
1.2	Patroni	4
1.3	EtcD	4
1.4	Watchdog	4
1.5	HAProxy	4
2	Security considerations	7
2.1	‘Quick start on RHEL 8’	7
2.1.1	EtcD	7
2.1.2	EDB Postgres Advanced or Extended Server	7
2.1.3	Watchdog	9
2.1.4	Patroni	9
2.1.5	HAProxy	12
2.2	‘Quick start on Debian 11’	14
2.2.1	EtcD	16
2.2.2	EDB Postgres Advanced or Extended Server	16
2.2.3	Watchdog	16
2.2.4	Patroni	17
2.2.5	HAProxy	20
2.3	‘Installing Patroni’	22
2.3.1	Debian/Ubuntu	22
2.3.2	RHEL/CentOS	22
2.4	‘Installing and configuring etcD’	22
2.4.1	Installation	22
2.4.2	Configuration	24
2.5	‘Managing a cluster’	25
2.5.1	Cluster information	25
2.5.2	Patroni logs	26
2.5.3	Maintenance mode	26
2.5.4	Manual failover or switchover	26
2.6	‘Tips and FAQs’	28
2.6.1	Connection strings	28
2.6.2	VIP & Keepalived	28

Patroni is among the most popular open-source tools for managing replication and failover of Postgres clusters. It relies on the Postgres streaming replication and hot standby capabilities. These capabilities allow DBAs to simplify the process of setting up and managing clusters that have high availability and read scalability requirements. EDB has tested Patroni for use with PostgreSQL, EDB Postgres Advanced Server, and EDB Postgres Extended Server.

For the Patroni documentation, see <https://patroni.readthedocs.io> .

SUPPORTED TOPOLOGY

Main high-availability components:

- Postgres cluster — The database cluster, usually consisting of a primary and two or more replicas.
- Patroni — Used as the failover management utility.
- etcd — Used as a distributed configuration store (DCS) containing cluster information, such as configuration, health, and current status.
- HAProxy — Used as a proxy to connect to the read-write and read-only nodes.

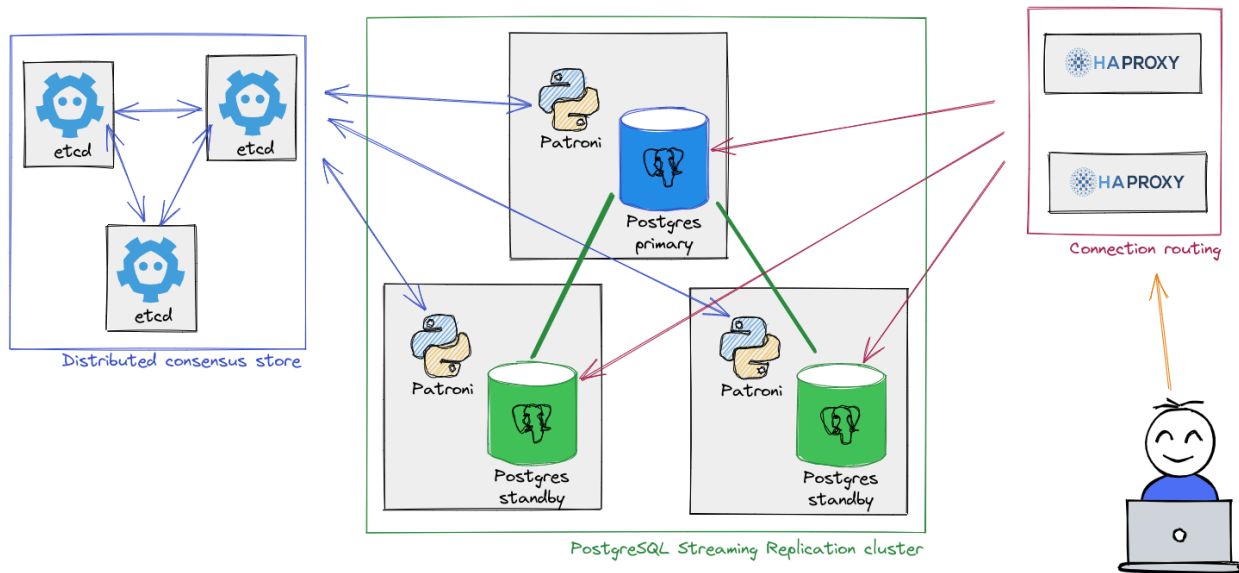


Fig. 1: Supported topology

1.1 Postgres

EDB currently supports using Patroni and Postgres *physical* streaming replication with a primary server and two replicas, replicating directly from the primary. We support using either PostgreSQL, EDB Postgres Advanced Server, or EDB Postgres Extended Server.

1.2 Patroni

Install Patroni on the Postgres nodes. The [latest release](#) of Patroni is supported.

1.3 Etcd

Etcd is a strongly consistent, distributed key-value store. It provides a reliable way to store data that a distributed system or cluster of machines needs to access.

For more information, see the [etcd documentation](#).

Note: For production environments, we recommend setting up an etcd cluster of three nodes. One etcd cluster can serve multiple Postgres/Patroni clusters. Install etcd nodes on different hosts from the Postgres nodes.

See ‘Installing and configuring etcd’ to install and set up etcd.

1.4 Watchdog

To avoid split-brain situations, Patroni supports *Watchdog* devices.

Watchdog devices are software or hardware mechanisms that reset the whole system when they don’t get a keepalive heartbeat within a specified timeframe. This device adds a layer of failsafe in case usual Patroni split-brain protection mechanisms fail.

Note: While the use of a watchdog mechanism with Patroni is optional, we recommend using it for production environments.

1.5 HAProxy

Instead of connecting directly to the database server, you can set up HAProxy so the application connects to the proxy instead. The proxy then forwards the request to Postgres. When you use HAProxy for this purpose, you can route read-only requests to one or more replicas for load balancing.

We strongly suggest using HAProxy to avoid data loss in case of split-brain.

You can install HAProxy as an independent server, but you can also install it on the application server or the database server.

Note: For production environments, we recommend setting up at least two HAProxy servers. Using various ports, one HAProxy server can serve multiple Postgres/Patroni clusters.

SECURITY CONSIDERATIONS

Security considerations are beyond the scope of this documentation. You can find more details about the distributed configuration storage (DCS) and the Patroni REST API in the [Patroni documentation](#).

You can use HTTPS with HAProxy, which is not covered in this documentation.

2.1 ‘Quick start on RHEL 8’

Use this quick start to create a highly available demo cluster using EDB Postgres Advanced Server or EDB Postgres Extended Server version 15 on RHEL 8. The etcd hosts will be called `etcd1`, `etcd2`, and `etcd3`. The two database nodes in streaming replication are `pg-patroni1` and `pg-patroni2`.

2.1.1 Etcd

See ‘Installing and configuring etcd’ to install and set up etcd.

2.1.2 EDB Postgres Advanced or Extended Server

On both `pg-patroni1` and `pg-patroni2` hosts, install your preferred flavor of Postgres flavor. See EDB Postgres Advanced Server or EDB Postgres Extended Server for more information about installing these products using the EDB repository.

The Postgres flavor you choose determines your environment variables.

For EDB Postgres Advanced Server, use the following settings:

```
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"
```

For EDB Postgres Extended Server, use the following settings:

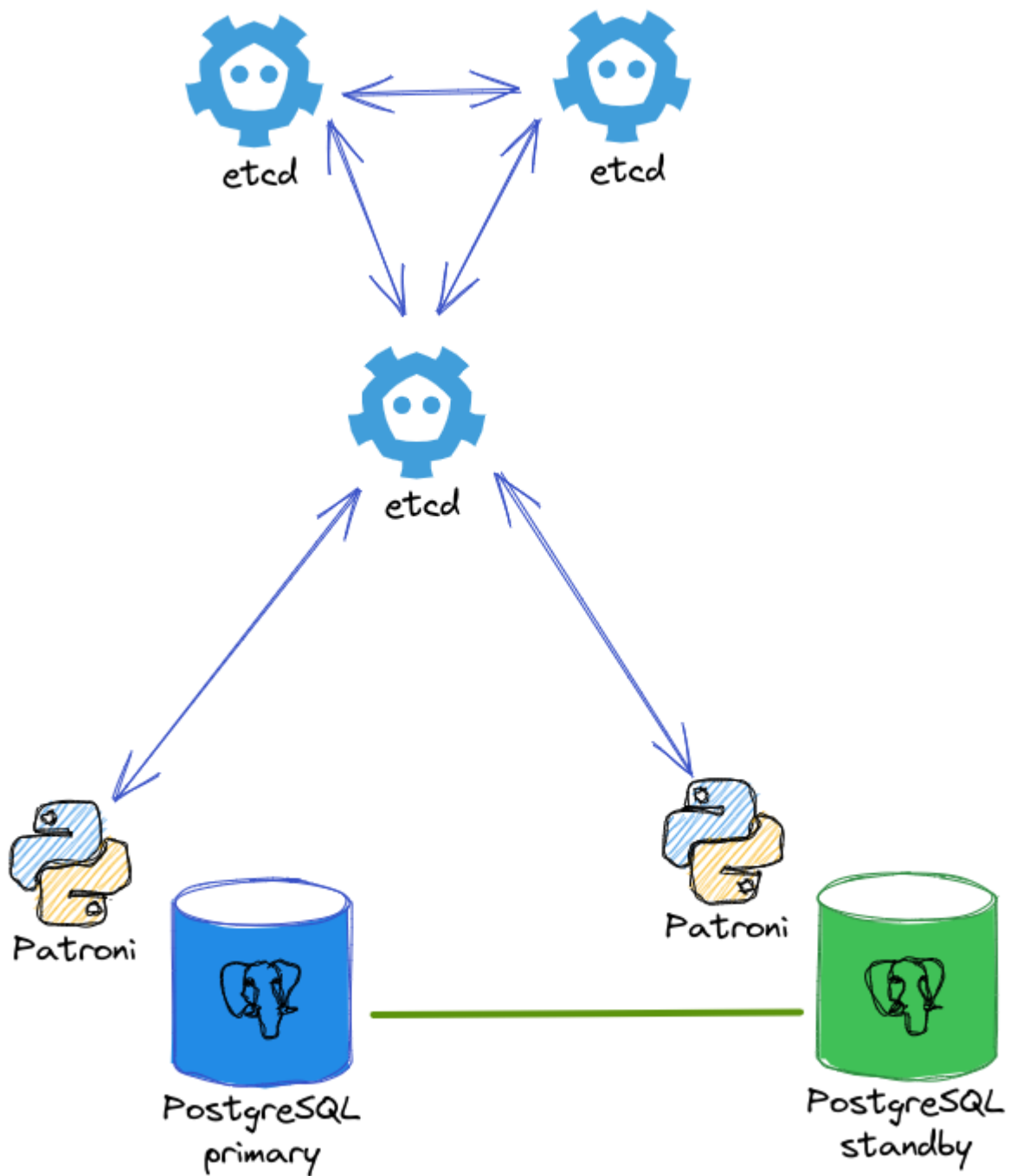


Fig. 1: Demo diagram

```
export PGPORT=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 bootstraps (creates) the initial Postgres cluster and is in charge of starting the service, so make sure `systemctl` is disabled for this service:

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

Configure the firewall on each database node so they accept connections to the Postgres service:

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

2.1.3 Watchdog

Patroni is the component interacting with the watchdog device. Set the permissions of the software watchdog on both `pg-patroni1` and `pg-patroni2` hosts:

```
$ 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 Patroni

On both `pg-patroni1` and `pg-patroni2` hosts, install Patroni and its dependencies for `etcd`. See ‘Installing Patroni’.

Define the Patroni configuration in `/etc/patroni.yml`:

```
$ 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"
```

(continues on next page)

(continued from previous page)

```

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
  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:

```

(continues on next page)

(continued from previous page)

```

    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

```

Note: In the *postgresql: authentication:* section in the example, a superuser is created during initialization (initdb) and later used by Patroni to connect to the database. The replication user is created and used by the standby servers to access the replication source by way of streaming replication. Finally, a rewind user is created and is used with *pg_rewind* when needed. We then recommend changing those passwords in the example configuration to match your password policy.

\$MY_IP and \$MY_NAME are specific to the local host. Otherwise, the `patroni.yml` configuration is the same on all Patroni nodes.

Patroni expects to find the `postgres` binary in the `bin_dir` location, while the EDB Postgres Advanced Server binary is called `edb-postgres`. Create a symbolic link to let Patroni find that binary:

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

Depending on the Patroni installation source, a `systemd` file might have already been created. Define your own file with the accurate system user, group, and configuration path:

```

$ 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

```

(continues on next page)

(continued from previous page)

```
Restart=no

[Install]
WantedBy=multi-user.target
EOF

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

Start the Patroni service first on pg-patroni1 so it becomes the leader of your Postgres cluster. Then start pg-patroni2.

To list the members of the cluster, use the patronictl command:

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

```

The Patroni REST API is available on port 8008 . Configure the firewall if needed:

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

2.1.5 HAProxy

The PGDG yum extras repositories contain haproxy packages.

To set up and use yum.postgresql.org on your systems, see the [PostgreSQL Linux downloads website](#) .

For the purpose of this example, install HAProxy on both pg-patroni1 and pg-patroni2 hosts:

```
$ 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
```

(continues on next page)

(continued from previous page)

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

There are two sections: read-write, using port 5000, and read-only, using port 5001. All Postgres hosts are included in both sections because they are all potential candidates to be either primary or standby. For HAProxy to know the role each host currently has, it queries the Patroni REST API:

```
$ 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
    }
  ],
}
```

(continues on next page)

(continued from previous page)

```
"dcs_last_seen":1679401036,
"database_system_identifier":"7104275668989116902",
"patroni":{
  "version":"3.0.1",
  "scope":"demo-cluster-1"
}
```

Note: By using the `/replica` endpoint, Patroni redirects only to a standby server. Use `/read-only` to also include the primary.

Configure the firewall if needed to access the ports for HAProxy to listen on:

```
$ 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
```

You can now use port 5000 and port 5001 to choose between read-write or read-only connections:

```
$ 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’

Use this quick start to create a highly available demo cluster using EDB Postgres Advanced Server or EDB Postgres Extended Server version 15 on Debian 11. The etcd hosts will be called `etcd1`, `etcd2`, and `etcd3`. The two database nodes in streaming replication are `pg-patroni1` and `pg-patroni2`.

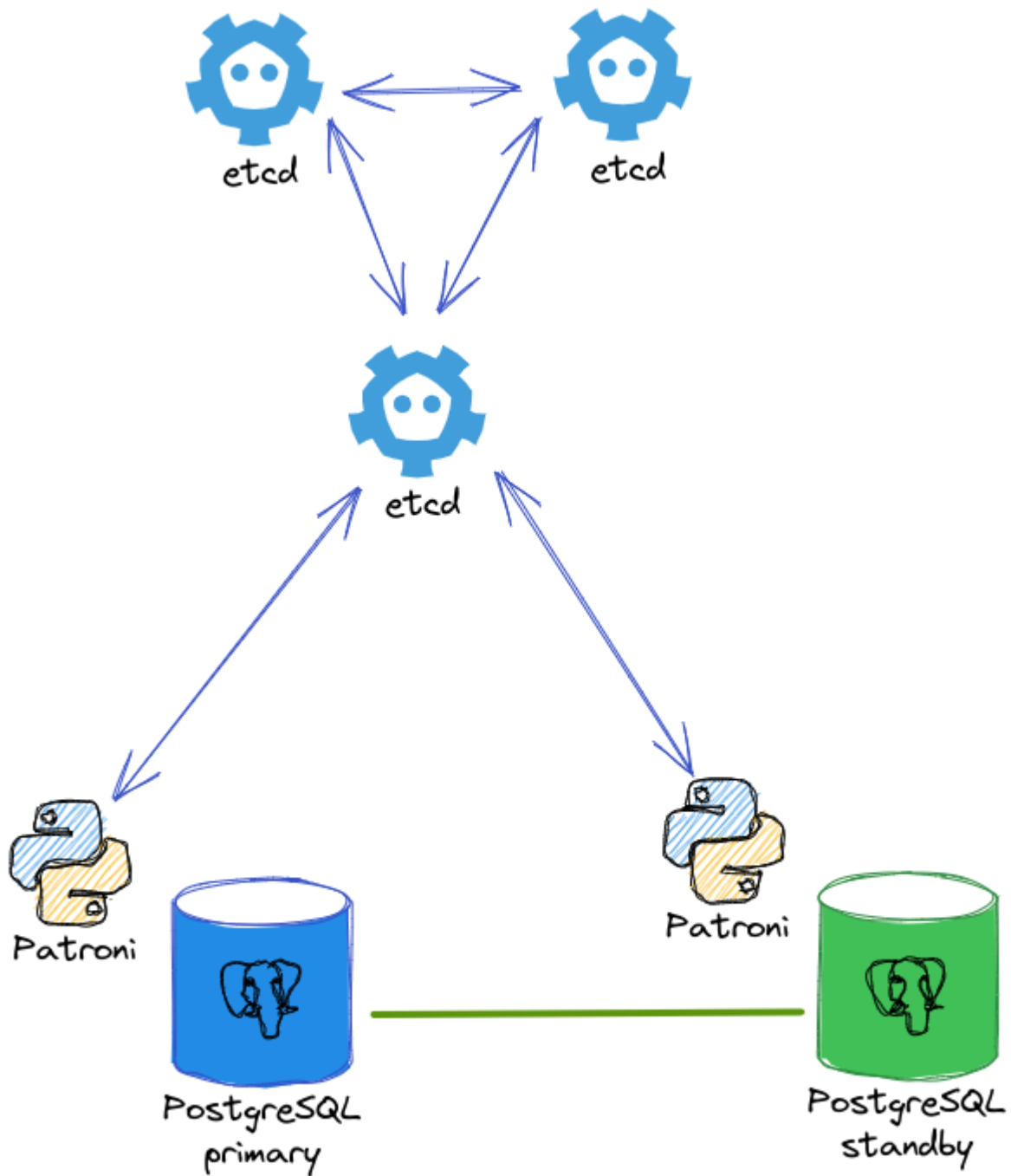


Fig. 2: Demo diagram

2.2.1 Etcd

See ‘Installing and configuring etcd’ to install and set up etcd.

2.2.2 EDB Postgres Advanced or Extended Server

On both `pg-patroni1` and `pg-patroni2` hosts, install your preferred Postgres flavor. See EDB Postgres Advanced Server or EDB Postgres Extended Server for more information about installing these products using the EDB repository.

The Postgres flavor you choose determines your environment variables.

For EDB Postgres Advanced Server, use the following settings:

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

For EDB Postgres Extended Server, use the following settings:

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

Patroni bootstraps (creates) the initial Postgres cluster and is in charge of starting the service. Make sure `systemctl` is disabled for this service, and remove the Postgres cluster created during the installation:

```
$ 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
``` or
```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 Watchdog

Patroni is the component interacting with the watchdog device. Set the permissions of the software watchdog on both `pg-patroni1` and `pg-patroni2` hosts:

```
$ 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 Patroni

On both `pg-patroni1` and `pg-patroni2` hosts, install Patroni and its dependencies for `etcd`. See ‘Installing Patroni’

Define the Patroni configuration in `/etc/patroni.yml` :

```
$ 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
users:
  admin:
```

(continues on next page)

(continued from previous page)

```
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
```

Note: In the *postgresql: authentication:* section in the example, a superuser is created during initialization (initdb) and later used by Patroni to connect to the database. The replication user is created and used by the standby servers to access the replication source by way of streaming replication. Finally, a rewind user is created and is used with *pg_rewind* when needed. We then recommend changing those passwords in the example configuration to match your password policy.

`$MY_IP` and `$MY_NAME` are specific to the local host. Otherwise, the `patroni.yml` configuration is the same on all Patroni nodes.

Patroni expects to find the `postgres` binary in the `bin_dir` location, while the EDB Postgres Advanced Server binary is called `edb-postgres`. Create a symbolic link to let Patroni find that binary:

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

Depending on the Patroni installation source, a `systemd` file might already have been created. Define your own file with the accurate system user, group, and configuration path:

```
$ 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
```

Start the Patroni service first on `pg-patroni1` so it becomes the leader of your Postgres cluster, and then start `pg-patroni2`.

To list the members of the cluster, use the `patronictl` command:

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

```

The Patroni REST API is available on port `8008`.

With EDB Postgres Advanced Server, after the database cluster is running, some other systemd dependencies might need a reset. Use the following commands to perform the resets:

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

2.2.5 HAProxy

For the purpose of this example, install HAProxy on both pg-patroni1 and pg-patroni2 hosts:

```
$ 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
    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
```

There are two sections: read-write, using port 5000 , and read-only, using port 5001 . All Postgres hosts are included in both sections because they are all potential candidates to be either primary or standby. For HAProxy to know the role each host currently has, it queries the Patroni REST API:

```
$ curl -s http://pg-patroni1:8008
{
```

(continues on next page)

(continued from previous page)

```

"state": "running",
"postmaster_start_time": "2022-06-30 07:45:55.427895+00:00",
"role": "master",
"server_version": 140400,
"xlog": {
  "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"
}
}

```

Note: By using the `/replica` endpoint, Patroni redirects only to a standby server. Use `/read-only` to also include the primary.

You can now use port 5000 and port 5001 to choose between read-write or read-only connections:

```

$ 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 packages are provided through the PGDG apt and yum repositories.

See [Platform Compatibility](#) for the supported OS list (only Linux x86-64 (amd64) currently).

2.3.1 Debian/Ubuntu

To install Patroni, configure the [PostgreSQL apt](#) repository. Then run:

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

See ‘Quick start on Debian 11’ for a more detailed configuration example.

2.3.2 RHEL/CentOS

You can install Patroni from the [PostgreSQL yum](#) repository. It requires Extra Packages for Enterprise Linux ([EPEL](#)).

After you configured the repositories, run the following command to install Patroni and its dependencies for etcd:

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

See Quick start on RHEL8 for a more detailed configuration example.

2.4 ‘Installing and configuring etcd’

2.4.1 Installation

Debian/Ubuntu packages

To install etcd from the official Debian/Ubuntu apt repositories:

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

You can check the version installed and the latest available version. If the version installed is out of date and you want to install the latest one, use the [Prebuilt binaries](#) method.

RedHat-compatible packages

The PGDG yum [extras](#) repositories contains etcd packages.

To set up and use [yum.postgresql.org](#) on your systems, see the [PostgreSQL Linux downloads website](#) .

Depending on your system, enable the following repositories:

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

For example, this command installs etcd on RHEL 8:

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

You can check the version installed and the latest available version. If the version installed is out of date and you want to install the latest one, use the *Prebuilt binaries* method.

Prebuilt binaries

These commands output the version of the installed etcd binary and get the latest version from the etcd-io website:

```
$ 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
```

The easiest way to install the etcd latest release is from prebuilt binaries.

On amd64 hosts, use the following commands:

```
$ 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
```

Then, create the data directory and system user:

```
$ 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
```

Configure the systemd unit file:

```
$ 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
```

(continues on next page)

(continued from previous page)

```
ExecStart=/usr/bin/etcd
Restart=always
RestartSec=10s
LimitNOFILE=40000

[Install]
WantedBy=multi-user.target
EOF

$ sudo systemctl daemon-reload
```

2.4.2 Configuration

Configure the firewall if needed:

```
$ 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
```

To configure and create a new etcd cluster, you need to define the initial cluster configuration by setting the `ETCD_INITIAL_CLUSTER` variable. That list must contain all the etcd hosts and their IP addresses.

This example shows three etcd hosts:

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

To fetch it dynamically, you can use this simple shell script, where `etcd1 etcd2 etcd3` are the three etcd hosts:

```
# 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\""
```

Then, set up the local etcd configuration and start the service:

```
$ 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"
```

(continues on next page)

(continued from previous page)

```
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
```

Note: When using the prebuilt binaries method or the Red Hat-compatible packages, the configuration file is */etc/etcd/etcd.conf* . When using the Debian/Ubuntu packages, the configuration file is */etc/default/etcd* .

Start the service and listen on ports 2379 and 2380 . Check the cluster status:

```
$ 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 Cluster information

To get information about the cluster, use the list or topology endpoints:

```
$ 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  |           |
```

(continues on next page)

(continued from previous page)

```

| 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 logs

Patroni logs are sent to system logs by systemd. Use the `journalctl` system utility to fetch it:

```

$ 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 Maintenance mode

To pause or resume cluster management, use:

```

$ 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 Manual failover or switchover

The failover endpoint allows you to perform a manual failover when there are no healthy nodes. It doesn't allow you to schedule a switchover.

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

Failover to a replica

(continues on next page)

(continued from previous page)

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.
```

The switchover endpoint works only when the cluster is healthy, that is, there's a leader. It allows you to schedule a switchover.

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.
```

Example:

```
$ 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
```

(continues on next page)

(continued from previous page)

-----+-----+-----+-----+-----				
↪-----+				
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 Connection strings

PostgreSQL [client libraries](#) like libpq and jdbc support client-connection failover. The connection string contains multiple servers, for example, `host=srv1,srv2`. The client library loops over the available hosts to find a connection that's available and capable of read-write or read-only operations. This configuration allows clients to follow the primary cluster during a switchover.

Example:

```
$ psql "host=srv1,srv2 dbname=postgres user=admin target_session_attrs=read-write" -c
↪"SELECT pg_is_in_recovery();"
pg_is_in_recovery
-----
f
(1 row)

$ psql "host=srv1,srv2 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

When using multiple HAProxy instances and you need a way to decide which proxy host to connect to, the easiest solution is to use Keepalived and a VIP (virtual IP).

RHEL8 example

This example shows installing and configuring on RHEL8.

First, install Keepalived on each hosts running HAProxy:

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

For this example, use 192.168.121.121 as the virtual IP. Configure Keepalived:

```
$ 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
```

Configure the firewall if needed:

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

Finally, enable and start the Keepalived service:

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

Once started, the virtual IP is attached to one of the hosts. To verify, try to reach it:

```
$ 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
```

(continues on next page)

(continued from previous page)

```
- -- 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
```