Patroni Documentation

Release 3.1.0

Zalando SE

Sep 20, 2023

CONTENTS:

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.

We call Patroni a "template" because it is far from being a one-size-fits-all or plug-and-play replication system. It will have its own caveats. Use wisely. There are many ways to run high availability with PostgreSQL; for a list, see the PostgreSQL Documentation.

Currently supported PostgreSQL versions: 9.3 to 15.

Note to Citus users: Starting from 3.0 Patroni nicely integrates with the Citus database extension to Postgres. Please check the *Citus support page* in the Patroni documentation for more info about how to use Patroni high availability together with a Citus distributed cluster.

Note to Kubernetes users: Patroni can run natively on top of Kubernetes. Take a look at the *Kubernetes* chapter of the Patroni documentation.

CHAPTER

ONE

INTRODUCTION

Patroni is a template for high availability (HA) PostgreSQL solutions using Python. Patroni originated as a fork of Governor, the project from Compose. It includes plenty of new features.

For an example of a Docker-based deployment with Patroni, see Spilo, currently in use at Zalando.

For additional background info, see:

- PostgreSQL HA with Kubernetes and Patroni, talk by Josh Berkus at KubeCon 2016 (video)
- Feb. 2016 Zalando Tech blog post

1.1 Development Status

Patroni is in active development and accepts contributions. See our *Contributing* section below for more details.

We report new releases information *here*.

1.2 Technical Requirements/Installation

Pre-requirements for Mac OS

To install requirements on a Mac, run the following:

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

Psycopg

Starting from psycopg2-2.8 the binary version of psycopg2 will no longer be installed by default. Installing it from the source code requires C compiler and postgres+python dev packages. Since in the python world it is not possible to specify dependency as `psycopg2 OR psycopg2-binary` you will have to decide how to install it.

There are a few options available:

1. Use the package manager from your distro

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

2. Install psycopg2 from the binary package

```
pip install psycopg2-binary
```

3. Install psycopg2 from source

```
pip install psycopg2>=2.5.4
```

4. Use psycopg 3.0 instead of psycopg2

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

General installation for pip

Patroni can be installed with pip:

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

where dependencies can be either empty, or consist of one or more of the following:

etcd or etcd3
python-etcd module in order to use Etcd as Distributed Configuration Store (DCS)

consul
python-consul module in order to use Consul as DCS

zookeeper
kazoo module in order to use Zookeeper as DCS

exhibitor
kazoo module in order to use Exhibitor as DCS (same dependencies as for Zookeeper)

kubernetes
kubernetes module in order to use Kubernetes as DCS in Patroni

raft
pysyncobj module in order to use python Raft implementation as DCS

aws
boto3 in order to use AWS callbacks

For example, the command in order to install Patroni together with dependencies for Etcd as a DCS and AWS callbacks is:

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

Note that external tools to call in the replica creation or custom bootstrap scripts (i.e. WAL-E) should be installed independently of Patroni.

1.3 Planning the Number of PostgreSQL Nodes

Patroni/PostgreSQL nodes are decoupled from DCS nodes (except when Patroni implements RAFT on its own) and therefore there is no requirement on the minimal number of nodes. Running a cluster consisting of one primary and one standby is perfectly fine. You can add more standby nodes later.

1.4 Running and Configuring

The following section assumes Patroni repository as being cloned from https://github.com/zalando/patroni. Namely, you will need example configuration files *postgres0.yml* and *postgres1.yml*. If you installed Patroni with pip, you can obtain those files from the git repository and replace *./patroni.py* below with *patroni* command.

To get started, do the following from different terminals:

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

You will then see a high-availability cluster start up. Test different settings in the YAML files to see how the cluster's behavior changes. Kill some of the components to see how the system behaves.

Add more `postgres*.yml` files to create an even larger cluster.

Patroni provides an HAProxy configuration, which will give your application a single endpoint for connecting to the cluster's leader. To configure, run:

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

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

1.5 YAML Configuration

Go *here* for comprehensive information about settings for etcd, consul, and ZooKeeper. And for an example, see postgres0.yml.

1.6 Environment Configuration

Go *here* for comprehensive information about configuring(overriding) settings via environment variables.

1.7 Replication Choices

Patroni uses Postgres' streaming replication, which is asynchronous by default. Patroni's asynchronous replication configuration allows for `maximum_lag_on_failover` settings. This setting ensures failover will not occur if a follower is more than a certain number of bytes behind the leader. This setting should be increased or decreased based on business requirements. It's also possible to use synchronous replication for better durability guarantees. See *replication modes documentation* for details.

1.8 Applications Should Not Use Superusers

When connecting from an application, always use a non-superuser. Patroni requires access to the database to function properly. By using a superuser from an application, you can potentially use the entire connection pool, including the connections reserved for superusers, with the `superuser_reserved_connections` setting. If Patroni cannot access the Primary because the connection pool is full, behavior will be undesirable.

1.9 Testing Your HA Solution

Testing an HA solution is a time consuming process, with many variables. This is particularly true considering a cross-platform application. You need a trained system administrator or a consultant to do this work. It is not something we can cover in depth in the documentation.

That said, here are some pieces of your infrastructure you should be sure to test:

- Network (the network in front of your system as well as the NICs [physical or virtual] themselves)
- Disk IO
- file limits (nofile in Linux)
- RAM. Even if you have oomkiller turned off, the unavailability of RAM could cause issues.
- CPU
- Virtualization Contention (overcommitting the hypervisor)
- Any cgroup limitation (likely to be related to the above)
- `kill -9` of any postgres process (except postmaster!). This is a decent simulation of a segfault.

One thing that you should not do is run `kill -9` on a postmaster process. This is because doing so does not mimic any real life scenario. If you are concerned your infrastructure is insecure and an attacker could run `kill -9`, no amount of HA process is going to fix that. The attacker will simply kill the process again, or cause chaos in another way.

CHAPTER

TWO

PATRONI CONFIGURATION

2.1 Dynamic Configuration Settings

Dynamic configuration is stored in the DCS (Distributed Configuration Store) and applied on all cluster nodes.

In order to change the dynamic configuration you can use either `patronictl edit-config` tool or Patroni *REST API*.

- **loop_wait**: the number of seconds the loop will sleep. Default value: 10
- **ttl**: the TTL to acquire the leader lock (in seconds). Think of it as the length of time before initiation of the automatic failover process. Default value: 30
- **retry_timeout**: timeout for DCS and PostgreSQL operation retries (in seconds). DCS or network issues shorter than this will not cause Patroni to demote the leader. Default value: 10
- **maximum_lag_on_failover**: the maximum bytes a follower may lag to be able to participate in leader election.
- **maximum_lag_on_syncnode**: the maximum bytes a synchronous follower may lag before it is considered as an unhealthy candidate and swapped by healthy asynchronous follower. Patroni utilize the max replica lsn if there is more than one follower, otherwise it will use leader's current wal lsn. Default is -1, Patroni will not take action to swap synchronous unhealthy follower when the value is set to 0 or below. Please set the value high enough so Patroni won't swap synchrounous follower fequently during high transaction volume.
- **max_timelines_history**: maximum number of timeline history items kept in DCS. Default value: 0. When set to 0, it keeps the full history in DCS.
- **primary_start_timeout**: the amount of time a primary is allowed to recover from failures before failover is triggered (in seconds). Default is 300 seconds. When set to 0 failover is done immediately after a crash is detected if possible. When using asynchronous replication a failover can cause lost transactions. Worst case failover time for primary failure is: loop_wait + primary_start_timeout + loop_wait, unless primary_start_timeout is zero, in which case it's just loop_wait. Set the value according to your durability/availability tradeoff.
- **primary_stop_timeout**: The number of seconds Patroni is allowed to wait when stopping Postgres and effective only when synchronous_mode is enabled. When set to > 0 and the synchronous_mode is enabled, Patroni sends SIGKILL to the postmaster if the stop operation is running for more than the value set by primary_stop_timeout. Set the value according to your durability/availability tradeoff. If the parameter is not set or set <= 0, primary_stop_timeout does not apply.
- **synchronous_mode**: turns on synchronous replication mode. In this mode a replica will be chosen as synchronous and only the latest leader and synchronous replica are able to participate in leader election. Synchronous mode makes sure that successfully committed transactions will not be lost at failover, at the cost of losing availability for writes when Patroni cannot ensure transaction durability. See *replication modes documentation* for details.
- **synchronous_mode_strict**: prevents disabling synchronous replication if no synchronous replicas are available, blocking all client writes to the primary. See *replication modes documentation* for details.

- **failsafe_mode**: Enables *DCS Failsafe Mode*. Defaults to *false*.
- **postgresql**:
  - **use_pg_rewind**: whether or not to use pg_rewind. Defaults to *false*.
  - **use_slots**: whether or not to use replication slots. Defaults to *true* on PostgreSQL 9.4+.
  - **recovery_conf**: additional configuration settings written to recovery.conf when configuring follower. There is no recovery.conf anymore in PostgreSQL 12, but you may continue using this section, because Patroni handles it transparently.
  - **parameters**: list of configuration settings for Postgres.
  - **pg_hba**: list of lines that Patroni will use to generate `pg_hba.conf`. Patroni ignores this parameter if `hba_file` PostgreSQL parameter is set to a non-default value.
    - **- host all all 0.0.0.0/0 md5**
    - **- host replication replicator 127.0.0.1/32 md5**: A line like this is required for replication.
  - **pg_ident**: list of lines that Patroni will use to generate `pg_ident.conf`. Patroni ignores this parameter if `ident_file` PostgreSQL parameter is set to a non-default value.
    - **- mapname1 systemname1 pguser1**
    - **- mapname1 systemname2 pguser2**
- **standby_cluster**: if this section is defined, we want to bootstrap a standby cluster.
  - **host**: an address of remote node
  - **port**: a port of remote node
  - **primary_slot_name**: which slot on the remote node to use for replication. This parameter is optional, the default value is derived from the instance name (see function *slot_name_from_member_name*).
  - **create_replica_methods**: an ordered list of methods that can be used to bootstrap standby leader from the remote primary, can be different from the list defined in *PostgreSQL*
  - **restore_command**: command to restore WAL records from the remote primary to nodes in a standby cluster, can be different from the list defined in *PostgreSQL*
  - **archive_cleanup_command**: cleanup command for standby leader
  - **recovery_min_apply_delay**: how long to wait before actually apply WAL records on a standby leader
- **slots**: define permanent replication slots. These slots will be preserved during switchover/failover. Permanent slots that don't exist will be created by Patroni. The physical slots are maintained only in the current primary. The logical slots are copied from the primary to a standby with restart, and after that their position advanced every **loop_wait** seconds (if necessary). Copying logical slot files performed via `libpq` connection and using either rewind or superuser credentials (see **postgresql.authentication** section). There is always a chance that the logical slot position on the replica is a bit behind the former primary, therefore application should be prepared that some messages could be received the second time after the failover. The easiest way of doing so - tracking `confirmed_flush_lsn`. Enabling permanent logical replication slots requires **postgresql.use_slots** to be set and will also automatically enable the `hot_standby_feedback`. Since the failover of logical replication slots is unsafe on PostgreSQL 9.6 and older and PostgreSQL version 10 is missing some important functions, the feature only works with PostgreSQL 11+.
  - **my_slot_name**: the name of the permanent replication slot. If the permanent slot name matches with the name of the current leader it will not be created. Please note that Patroni does not make checks for permanent slot names added to this configuration matching those that Patroni creates automatically for members. If those names are added, Patroni will ensure that any slots that were created are not removed even if the member becomes unresponsive, situation which would normally result in the slot's removal by

Patroni. Although this can be useful in some situations, such as when importing existing members to a new Patroni cluster (see *Convert a Standalone to a Patroni Cluster* for details), caution should be exercised by the operator that these clashes in names are not persisted in the DCS due to its effect on normal functioning of Patroni.

  * **type**: slot type. Could be `physical` or `logical`. If the slot is logical, you have to additionally define `database` and `plugin`.
  * **database**: the database name where logical slots should be created.
  * **plugin**: the plugin name for the logical slot.

- **ignore_slots**: list of sets of replication slot properties for which Patroni should ignore matching slots. This configuration/feature/etc. is useful when some replication slots are managed outside of Patroni. Any subset of matching properties will cause a slot to be ignored.
  - **name**: the name of the replication slot.
  - **type**: slot type. Can be `physical` or `logical`. If the slot is logical, you may additionally define `database` and/or `plugin`.
  - **database**: the database name (when matching a `logical` slot).
  - **plugin**: the logical decoding plugin (when matching a `logical` slot).

Note: **slots** is a hashmap while **ignore_slots** is an array. For example:

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

2.2 YAML Configuration Settings

2.2.1 Global/Universal

- **name**: the name of the host. Must be unique for the cluster.
- **namespace**: path within the configuration store where Patroni will keep information about the cluster. Default value: "/service"
- **scope**: cluster name

2.2.2 Log

- **level**: sets the general logging level. Default value is **INFO** (see the docs for Python logging)
- **traceback_level**: sets the level where tracebacks will be visible. Default value is **ERROR**. Set it to **DEBUG** if you want to see tracebacks only if you enable **log.level=DEBUG**.
- **format**: sets the log formatting string. Default value is **%(asctime)s %(levelname)s: %(message)s** (see the LogRecord attributes)
- **dateformat**: sets the datetime formatting string. (see the formatTime() documentation)
- **max_queue_size**: Patroni is using two-step logging. Log records are written into the in-memory queue and there is a separate thread which pulls them from the queue and writes to stderr or file. The maximum size of the internal queue is limited by default by **1000** records, which is enough to keep logs for the past 1h20m.
- **dir**: Directory to write application logs to. The directory must exist and be writable by the user executing Patroni. If you set this value, the application will retain 4 25MB logs by default. You can tune those retention values with *file_num* and *file_size* (see below).
- **file_num**: The number of application logs to retain.
- **file_size**: Size of patroni.log file (in bytes) that triggers a log rolling.
- **loggers**: This section allows redefining logging level per python module
  - **patroni.postmaster: WARNING**
  - **urllib3: DEBUG**

2.2.3 Bootstrap configuration

Note: Once Patroni has initialized the cluster for the first time and settings have been stored in the DCS, all future changes to the `bootstrap.dcs` section of the YAML configuration will not take any effect! If you want to change them please use either `patronictl edit-config` or the Patroni *REST API*.

- **bootstrap**:
  - **dcs**: This section will be written into */<namespace>/<scope>/config* of the given configuration store after initializing the new cluster. The global dynamic configuration for the cluster. You can put any of the parameters described in the *Dynamic Configuration settings* under `bootstrap.dcs` and after Patroni has initialized (bootstrapped) the new cluster, it will write this section into */<namespace>/<scope>/config* of the configuration store.
  - **method**: custom script to use for bootstrapping this cluster.

    See *custom bootstrap methods documentation* for details. When `initdb` is specified revert to the default `initdb` command. `initdb` is also triggered when no `method` parameter is present in the configuration file.
  - **initdb**: (optional) list options to be passed on to initdb.
    - **- data-checksums**: Must be enabled when pg_rewind is needed on 9.3.
    - **- encoding: UTF8**: default encoding for new databases.
    - **- locale: UTF8**: default locale for new databases.
  - **users**: Some additional users which need to be created after initializing new cluster, see *Bootstrap users configuration* below.

  - **post_bootstrap** or **post_init**: An additional script that will be executed after initializing the cluster. The script receives a connection string URL (with the cluster superuser as a user name). The PGPASSFILE variable is set to the location of pgpass file.

Bootstrap users configuration

Users which need to be created after initializing the cluster:

- **admin**: the name of user
  - **password**: (optional) password for the user
  - **options**: list of options for CREATE USER statement
  - **- createrole**
  - **- createdb**

2.2.4 Citus

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

- **group**: the Citus group id, integer. Use `0` for coordinator and `1`, `2`, etc... for workers
- **database**: the database where `citus` extension should be created. Must be the same on the coordinator and all workers. Currently only one database is supported.

2.2.5 Consul

Most of the parameters are optional, but you have to specify one of the **host** or **url**

- **host**: the host:port for the Consul local agent.
- **url**: url for the Consul local agent, in format: http(s)://host:port.
- **port**: (optional) Consul port.
- **scheme**: (optional) **http** or **https**, defaults to **http**.
- **token**: (optional) ACL token.
- **verify**: (optional) whether to verify the SSL certificate for HTTPS requests.
- **cacert**: (optional) The ca certificate. If present it will enable validation.
- **cert**: (optional) file with the client certificate.
- **key**: (optional) file with the client key. Can be empty if the key is part of **cert**.
- **dc**: (optional) Datacenter to communicate with. By default the datacenter of the host is used.
- **consistency**: (optional) Select consul consistency mode. Possible values are `default`, `consistent`, or `stale` (more details in consul API reference)
- **checks**: (optional) list of Consul health checks used for the session. By default an empty list is used.
- **register_service**: (optional) whether or not to register a service with the name defined by the scope parameter and the tag master, primary, replica, or standby-leader depending on the node's role. Defaults to **false**.
- **service_tags**: (optional) additional static tags to add to the Consul service apart from the role (`master`/`primary`/`replica`/`standby-leader`). By default an empty list is used.

- **service_check_interval**: (optional) how often to perform health check against registered url. Defaults to '5s'.
- **service_check_tls_server_name**: (optional) overide SNI host when connecting via TLS, see also consul agent check API reference.

The `token` needs to have the following ACL permissions:

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

2.2.6 Etcd

Most of the parameters are optional, but you have to specify one of the **host**, **hosts**, **url**, **proxy** or **srv**

- **host**: the host:port for the etcd endpoint.
- **hosts**: list of etcd endpoint in format host1:port1,host2:port2,etc... Could be a comma separated string or an actual yaml list.
- **use_proxies**: If this parameter is set to true, Patroni will consider **hosts** as a list of proxies and will not perform a topology discovery of etcd cluster.
- **url**: url for the etcd.
- **proxy**: proxy url for the etcd. If you are connecting to the etcd using proxy, use this parameter instead of **url**.
- **srv**: Domain to search the SRV record(s) for cluster autodiscovery. Patroni will try to query these SRV service names for specified domain (in that order until first success): `_etcd-client-ssl`, `_etcd-client`, `_etcd-ssl`, `_etcd`, `_etcd-server-ssl`, `_etcd-server`. If SRV records for `_etcd-server-ssl` or `_etcd-server` are retrieved then ETCD peer protocol is used do query ETCD for available members. Otherwise hosts from SRV records will be used.
- **srv_suffix**: Configures a suffix to the SRV name that is queried during discovery. Use this flag to differentiate between multiple etcd clusters under the same domain. Works only with conjunction with **srv**. For example, if `srv_suffix: foo` and `srv: example.org` are set, the following DNS SRV query is made:`_etcd-client-ssl-foo._tcp.example.com` (and so on for every possible ETCD SRV service name).
- **protocol**: (optional) http or https, if not specified http is used. If the **url** or **proxy** is specified - will take protocol from them.
- **username**: (optional) username for etcd authentication.
- **password**: (optional) password for etcd authentication.
- **cacert**: (optional) The ca certificate. If present it will enable validation.
- **cert**: (optional) file with the client certificate.
- **key**: (optional) file with the client key. Can be empty if the key is part of **cert**.

2.2.7 Etcdv3

If you want that Patroni works with Etcd cluster via protocol version 3, you need to use the `etcd3` section in the Patroni configuration file. All configuration parameters are the same as for `etcd`.

Warning: Keys created with protocol version 2 are not visible with protocol version 3 and the other way around, therefore it is not possible to switch from `etcd` to `etcd3` just by updating Patroni config file.

2.2.8 ZooKeeper

- **hosts**: List of ZooKeeper cluster members in format: ['host1:port1', 'host2:port2', 'etc...'].
- **use_ssl**: (optional) Whether SSL is used or not. Defaults to `false`. If set to `false`, all SSL specific parameters are ignored.
- **cacert**: (optional) The CA certificate. If present it will enable validation.
- **cert**: (optional) File with the client certificate.
- **key**: (optional) File with the client key.
- **key_password**: (optional) The client key password.
- **verify**: (optional) Whether to verify certificate or not. Defaults to `true`.
- **set_acls**: (optional) If set, configure Kazoo to apply a default ACL to each ZNode that it creates. ACLs will assume 'x509' schema and should be specified as a dictionary with the principal as the key and one or more permissions as a list in the value. Permissions may be one of `CREATE`, `READ`, `WRITE`, `DELETE` or `ADMIN`. For example, `set_acls: {CN=principal1: [CREATE, READ], CN=principal2: [ALL]}`.

Note: It is required to install `kazoo>=2.6.0` to support SSL.

2.2.9 Exhibitor

- **hosts**: initial list of Exhibitor (ZooKeeper) nodes in format: 'host1,host2,etc...'. This list updates automatically whenever the Exhibitor (ZooKeeper) cluster topology changes.
- **poll_interval**: how often the list of ZooKeeper and Exhibitor nodes should be updated from Exhibitor.
- **port**: Exhibitor port.

2.2.10 Kubernetes

- **bypass_api_service**: (optional) When communicating with the Kubernetes API, Patroni is usually relying on the *kubernetes* service, the address of which is exposed in the pods via the *KUBERNETES_SERVICE_HOST* environment variable. If *bypass_api_service* is set to `true`, Patroni will resolve the list of API nodes behind the service and connect directly to them.
- **namespace**: (optional) Kubernetes namespace where Patroni pod is running. Default value is *default*.
- **labels**: Labels in format `{label1: value1, label2: value2}`. These labels will be used to find existing objects (Pods and either Endpoints or ConfigMaps) associated with the current cluster. Also Patroni will set them on every object (Endpoint or ConfigMap) it creates.

- **scope_label**: (optional) name of the label containing cluster name. Default value is *cluster-name*.
- **role_label**: (optional) name of the label containing role (master or replica or other custom value). Patroni will set this label on the pod it runs in. Default value is `role`.
- **leader_label_value**: (optional) value of the pod label when Postgres role is `master`. Default value is `master`.
- **follower_label_value**: (optional) value of the pod label when Postgres role is `replica`. Default value is `replica`.
- **standby_leader_label_value**: (optional) value of the pod label when Postgres role is `standby-leader`. Default value is `standby-leader`.
- **tmp_role_label**: (optional) name of the temporary label containing role (master or replica). Value of this label will always use the default of corresponding role. Set only when necessary.
- **use_endpoints**: (optional) if set to true, Patroni will use Endpoints instead of ConfigMaps to run leader elections and keep cluster state.
- **pod_ip**: (optional) IP address of the pod Patroni is running in. This value is required when *use_endpoints* is enabled and is used to populate the leader endpoint subsets when the pod's PostgreSQL is promoted.
- **ports**: (optional) if the Service object has the name for the port, the same name must appear in the Endpoint object, otherwise service won't work. For example, if your service is defined as `{Kind: Service, spec: {ports: [{name: postgresql, port: 5432, targetPort: 5432}]}}`, then you have to set `kubernetes.ports: [{"name": "postgresql", "port": 5432}]` and Patroni will use it for updating subsets of the leader Endpoint. This parameter is used only if *kubernetes.use_endpoints* is set.
- **cacert**: (optional) Specifies the file with the CA_BUNDLE file with certificates of trusted CAs to use while verifying Kubernetes API SSL certs. If not provided, patroni will use the value provided by the ServiceAccount secret.
- **retriable_http_codes**: (optional) list of HTTP status codes from K8s API to retry on. By default Patroni is retrying on `500`, `503`, and `504`, or if K8s API response has `retry-after` HTTP header.

2.2.11 Raft (deprecated)

- **self_addr**: `ip:port` to listen on for Raft connections. The `self_addr` must be accessible from other nodes of the cluster. If not set, the node will not participate in consensus.
- **bind_addr**: (optional) `ip:port` to listen on for Raft connections. If not specified the `self_addr` will be used.
- **partner_addrs**: list of other Patroni nodes in the cluster in format: ['ip1:port', 'ip2:port', 'etc...']
- **data_dir**: directory where to store Raft log and snapshot. If not specified the current working directory is used.
- **password**: (optional) Encrypt Raft traffic with a specified password, requires `cryptography` python module.

  Short FAQ about Raft implementation

  - Q: How to list all the nodes providing consensus?

    A: `syncobj_admin -conn host:port -status` where the host:port is the address of one of the cluster nodes

  - Q: Node that was a part of consensus and has gone and I can't reuse the same IP for other node. How to remove this node from the consensus?

    A: `syncobj_admin -conn host:port -remove host2:port2` where the `host2:port2` is the address of the node you want to remove from consensus.

  - Q: Where to get the `syncobj_admin` utility?

    A: It is installed together with `pysyncobj` module (python RAFT implementation), which is Patroni dependency.

  - Q: it is possible to run Patroni node without adding in to the consensus?

    A: Yes, just comment out or remove `raft.self_addr` from Patroni configuration.

  - Q: It is possible to run Patroni and PostgreSQL only on two nodes?

    A: Yes, on the third node you can run `patroni_raft_controller` (without Patroni and PostgreSQL). In such a setup, one can temporarily lose one node without affecting the primary.

2.2.12 PostgreSQL

- **postgresql**:
  - **authentication**:
    - **superuser**:
      - **username**: name for the superuser, set during initialization (initdb) and later used by Patroni to connect to the postgres.
      - **password**: password for the superuser, set during initialization (initdb).
      - **sslmode**: (optional) maps to the sslmode connection parameter, which allows a client to specify the type of TLS negotiation mode with the server. For more information on how each mode works, please visit the PostgreSQL documentation. The default mode is `prefer`.
      - **sslkey**: (optional) maps to the sslkey connection parameter, which specifies the location of the secret key used with the client's certificate.
      - **sslpassword**: (optional) maps to the sslpassword connection parameter, which specifies the password for the secret key specified in `sslkey`.
      - **sslcert**: (optional) maps to the sslcert connection parameter, which specifies the location of the client certificate.
      - **sslrootcert**: (optional) maps to the sslrootcert connection parameter, which specifies the location of a file containing one ore more certificate authorities (CA) certificates that the client will use to verify a server's certificate.
      - **sslcrl**: (optional) maps to the sslcrl connection parameter, which specifies the location of a file containing a certificate revocation list. A client will reject connecting to any server that has a certificate present in this list.
      - **sslcrldir**: (optional) maps to the sslcrldir connection parameter, which specifies the location of a directory with files containing a certificate revocation list. A client will reject connecting to any server that has a certificate present in this list.
      - **gssencmode**: (optional) maps to the gssencmode connection parameter, which determines whether or with what priority a secure GSS TCP/IP connection will be negotiated with the server
      - **channel_binding**: (optional) maps to the channel_binding connection parameter, which controls the client's use of channel binding.
    - **replication**:
      - **username**: replication username; the user will be created during initialization. Replicas will use this user to access the replication source via streaming replication
      - **password**: replication password; the user will be created during initialization.

- **sslmode**: (optional) maps to the sslmode connection parameter, which allows a client to specify the type of TLS negotiation mode with the server. For more information on how each mode works, please visit the PostgreSQL documentation. The default mode is `prefer`.
- **sslkey**: (optional) maps to the sslkey connection parameter, which specifies the location of the secret key used with the client's certificate.
- **sslpassword**: (optional) maps to the sslpassword connection parameter, which specifies the password for the secret key specified in `sslkey`.
- **sslcert**: (optional) maps to the sslcert connection parameter, which specifies the location of the client certificate.
- **sslrootcert**: (optional) maps to the sslrootcert connection parameter, which specifies the location of a file containing one ore more certificate authorities (CA) certificates that the client will use to verify a server's certificate.
- **sslcrl**: (optional) maps to the sslcrl connection parameter, which specifies the location of a file containing a certificate revocation list. A client will reject connecting to any server that has a certificate present in this list.
- **sslcrldir**: (optional) maps to the sslcrldir connection parameter, which specifies the location of a directory with files containing a certificate revocation list. A client will reject connecting to any server that has a certificate present in this list.
- **gssencmode**: (optional) maps to the gssencmode connection parameter, which determines whether or with what priority a secure GSS TCP/IP connection will be negotiated with the server
- **channel_binding**: (optional) maps to the channel_binding connection parameter, which controls the client's use of channel binding.

* **rewind**:
  - **username**: (optional) name for the user for `pg_rewind`; the user will be created during initialization of postgres 11+ and all necessary permissions will be granted.
  - **password**: (optional) password for the user for `pg_rewind`; the user will be created during initialization.
  - **sslmode**: (optional) maps to the sslmode connection parameter, which allows a client to specify the type of TLS negotiation mode with the server. For more information on how each mode works, please visit the PostgreSQL documentation. The default mode is `prefer`.
  - **sslkey**: (optional) maps to the sslkey connection parameter, which specifies the location of the secret key used with the client's certificate.
  - **sslpassword**: (optional) maps to the sslpassword connection parameter, which specifies the password for the secret key specified in `sslkey`.
  - **sslcert**: (optional) maps to the sslcert connection parameter, which specifies the location of the client certificate.
  - **sslrootcert**: (optional) maps to the sslrootcert connection parameter, which specifies the location of a file containing one ore more certificate authorities (CA) certificates that the client will use to verify a server's certificate.
  - **sslcrl**: (optional) maps to the sslcrl connection parameter, which specifies the location of a file containing a certificate revocation list. A client will reject connecting to any server that has a certificate present in this list.
  - **sslcrldir**: (optional) maps to the sslcrldir connection parameter, which specifies the location of a directory with files containing a certificate revocation list. A client will reject connecting to any server that has a certificate present in this list.

- **gssencmode**: (optional) maps to the gssencmode connection parameter, which determines whether or with what priority a secure GSS TCP/IP connection will be negotiated with the server
- **channel_binding**: (optional) maps to the channel_binding connection parameter, which controls the client's use of channel binding.

– **callbacks**: callback scripts to run on certain actions. Patroni will pass the action, role and cluster name. (See scripts/aws.py as an example of how to write them.)

* **on_reload**: run this script when configuration reload is triggered.
* **on_restart**: run this script when the postgres restarts (without changing role).
* **on_role_change**: run this script when the postgres is being promoted or demoted.
* **on_start**: run this script when the postgres starts.
* **on_stop**: run this script when the postgres stops.

– **connect_address**: IP address + port through which Postgres is accessible from other nodes and applications.

– **proxy_address**: IP address + port through which a connection pool (e.g. pgbouncer) running next to Postgres is accessible. The value is written to the member key in DCS as `proxy_url` and could be used/useful for service discovery.

– **create_replica_methods**: an ordered list of the create methods for turning a Patroni node into a new replica. "basebackup" is the default method; other methods are assumed to refer to scripts, each of which is configured as its own config item. See *custom replica creation methods documentation* for further explanation.

– **data_dir**: The location of the Postgres data directory, either *existing* or to be initialized by Patroni.

– **config_dir**: The location of the Postgres configuration directory, defaults to the data directory. Must be writable by Patroni.

– **bin_dir**: (optional) Path to PostgreSQL binaries (pg_ctl, initdb, pg_controldata, pg_basebackup, postgres, pg_isready, pg_rewind). If not provided or is an empty string, PATH environment variable will be used to find the executables.

– **bin_name**: (optional) Make it possible to override Postgres binary names, if you are using a custom Postgres distribution:

* **pg_ctl**: (optional) Custom name for `pg_ctl` binary.
* **initdb**: (optional) Custom name for `initdb` binary.
* **pgcontroldata**: (optional) Custom name for `pg_controldata` binary.
* **pg_basebackup**: (optional) Custom name for `pg_basebackup` binary.
* **postgres**: (optional) Custom name for `postgres` binary.
* **pg_isready**: (optional) Custom name for `pg_isready` binary.
* **pg_rewind**: (optional) Custom name for `pg_rewind` binary.

– **listen**: IP address + port that Postgres listens to; must be accessible from other nodes in the cluster, if you're using streaming replication. Multiple comma-separated addresses are permitted, as long as the port component is appended after to the last one with a colon, i.e. `listen: 127.0.0.1,127.0.0.2:5432`. Patroni will use the first address from this list to establish local connections to the PostgreSQL node.

– **use_unix_socket**: specifies that Patroni should prefer to use unix sockets to connect to the cluster. Default value is `false`. If `unix_socket_directories` is defined, Patroni will use the first suitable value from it to connect to the cluster and fallback to tcp if nothing is suitable. If `unix_socket_directories` is not

specified in `postgresql.parameters`, Patroni will assume that the default value should be used and omit `host` from the connection parameters.

- **use_unix_socket_repl**: specifies that Patroni should prefer to use unix sockets for replication user cluster connection. Default value is `false`. If `unix_socket_directories` is defined, Patroni will use the first suitable value from it to connect to the cluster and fallback to tcp if nothing is suitable. If `unix_socket_directories` is not specified in `postgresql.parameters`, Patroni will assume that the default value should be used and omit `host` from the connection parameters.
- **pgpass**: path to the .pgpass password file. Patroni creates this file before executing pg_basebackup, the post_init script and under some other circumstances. The location must be writable by Patroni.
- **recovery_conf**: additional configuration settings written to recovery.conf when configuring follower.
- **custom_conf** : path to an optional custom `postgresql.conf` file, that will be used in place of `postgresql.base.conf`. The file must exist on all cluster nodes, be readable by PostgreSQL and will be included from its location on the real `postgresql.conf`. Note that Patroni will not monitor this file for changes, nor backup it. However, its settings can still be overridden by Patroni's own configuration facilities - see *dynamic configuration* for details.
- **parameters**: list of configuration settings for Postgres. Many of these are required for replication to work.
- **pg_hba**: list of lines that Patroni will use to generate `pg_hba.conf`. Patroni ignores this parameter if `hba_file` PostgreSQL parameter is set to a non-default value. Together with *dynamic configuration* this parameter simplifies management of `pg_hba.conf`.
  - **- host all all 0.0.0.0/0 md5**
  - **- host replication replicator 127.0.0.1/32 md5**: A line like this is required for replication.
- **pg_ident**: list of lines that Patroni will use to generate `pg_ident.conf`. Patroni ignores this parameter if `ident_file` PostgreSQL parameter is set to a non-default value. Together with *dynamic configuration* this parameter simplifies management of `pg_ident.conf`.
  - **- mapname1 systemname1 pguser1**
  - **- mapname1 systemname2 pguser2**
- **pg_ctl_timeout**: How long should pg_ctl wait when doing `start`, `stop` or `restart`. Default value is 60 seconds.
- **use_pg_rewind**: try to use pg_rewind on the former leader when it joins cluster as a replica.
- **remove_data_directory_on_rewind_failure**: If this option is enabled, Patroni will remove the PostgreSQL data directory and recreate the replica. Otherwise it will try to follow the new leader. Default value is **false**.
- **remove_data_directory_on_diverged_timelines**: Patroni will remove the PostgreSQL data directory and recreate the replica if it notices that timelines are diverging and the former primary can not start streaming from the new primary. This option is useful when `pg_rewind` can not be used. While performing timelines divergence check on PostgreSQL v10 and older Patroni will try to connect with replication credential to the "postgres" database. Hence, such access should be allowed in the pg_hba.conf. Default value is **false**.
- **replica_method**: for each create_replica_methods other than basebackup, you would add a configuration section of the same name. At a minimum, this should include "command" with a full path to the actual script to be executed. Other configuration parameters will be passed along to the script in the form "parameter=value".
- **pre_promote**: a fencing script that executes during a failover after acquiring the leader lock but before promoting the replica. If the script exits with a non-zero code, Patroni does not promote the replica and removes the leader key from DCS.

  - **before_stop**: a script that executes immediately prior to stopping postgres. As opposed to a callback, this script runs synchronously, blocking shutdown until it has completed. The return code of this script does not impact whether shutdown proceeds afterwards.

2.2.13 REST API

- **restapi**:
  - **connect_address**: IP address (or hostname) and port, to access the Patroni's *REST API*. All the members of the cluster must be able to connect to this address, so unless the Patroni setup is intended for a demo inside the localhost, this address must be a non "localhost" or loopback address (ie: "localhost" or "127.0.0.1"). It can serve as an endpoint for HTTP health checks (read below about the "listen" REST API parameter), and also for user queries (either directly or via the REST API), as well as for the health checks done by the cluster members during leader elections (for example, to determine whether the leader is still running, or if there is a node which has a WAL position that is ahead of the one doing the query; etc.) The connect_address is put in the member key in DCS, making it possible to translate the member name into the address to connect to its REST API.
  - **listen**: IP address (or hostname) and port that Patroni will listen to for the REST API - to provide also the same health checks and cluster messaging between the participating nodes, as described above. to provide health-check information for HAProxy (or any other load balancer capable of doing a HTTP "OPTION" or "GET" checks).
  - **authentication**: (optional)
    - **username**: Basic-auth username to protect unsafe REST API endpoints.
    - **password**: Basic-auth password to protect unsafe REST API endpoints.
  - **certfile**: (optional): Specifies the file with the certificate in the PEM format. If the certfile is not specified or is left empty, the API server will work without SSL.
  - **keyfile**: (optional): Specifies the file with the secret key in the PEM format.
  - **keyfile_password**: (optional): Specifies a password for decrypting the keyfile.
  - **cafile**: (optional): Specifies the file with the CA_BUNDLE with certificates of trusted CAs to use while verifying client certs.
  - **ciphers**: (optional): Specifies the permitted cipher suites (e.g. "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**: (optional): `none` (default), `optional` or `required`. When `none` REST API will not check client certificates. When `required` client certificates are required for all REST API calls. When `optional` client certificates are required for all unsafe REST API endpoints. When `required` is used, then client authentication succeeds, if the certificate signature verification succeeds. For `optional` the client cert will only be checked for `PUT`, `POST`, `PATCH`, and `DELETE` requests.
  - **allowlist**: (optional): Specifies the set of hosts that are allowed to call unsafe REST API endpoints. The single element could be a host name, an IP address or a network address using CIDR notation. By default `allow all` is used. In case if `allowlist` or `allowlist_include_members` are set, anything that is not included is rejected.
  - **allowlist_include_members**: (optional): If set to `true` it allows accessing unsafe REST API endpoints from other cluster members registered in DCS (IP address or hostname is taken from the members `api_url`). Be careful, it might happen that OS will use a different IP for outgoing connections.
  - **http_extra_headers**: (optional): HTTP headers let the REST API server pass additional information with an HTTP response.

  - **https_extra_headers**: (optional): HTTPS headers let the REST API server pass additional information with an HTTP response when TLS is enabled. This will also pass additional information set in `http_extra_headers`.
  - **request_queue_size**: (optional): Sets request queue size for TCP socket used by Patroni REST API. Once the queue is full, further requests get a "Connection denied" error. The default value is 5.

Here is an example of both **http_extra_headers** and **https_extra_headers**:

```
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'
```

Warning:

- The `restapi.connect_address` must be accessible from all nodes of a given Patroni cluster. Internally Patroni is using it during the leader race to find nodes with minimal replication lag.
- If you enabled client certificates validation (`restapi.verify_client` is set to `required`), you also **must** provide **valid client certificates** in the `ctl.certfile`, `ctl.keyfile`, `ctl.keyfile_password`. If not provided, Patroni will not work correctly.

2.2.14 CTL

- **ctl**: (optional)
  - **authentication**:
    - **username**: Basic-auth username for accessing protected REST API endpoints. If not provided patronictl will use the value provided for REST API "username" parameter.
    - **password**: Basic-auth password for accessing protected REST API endpoints. If not provided patronictl will use the value provided for REST API "password" parameter.
  - **insecure**: Allow connections to REST API without verifying SSL certs.
  - **cacert**: Specifies the file with the CA_BUNDLE file or directory with certificates of trusted CAs to use while verifying REST API SSL certs. If not provided patronictl will use the value provided for REST API "cafile" parameter.
  - **certfile**: Specifies the file with the client certificate in the PEM format.
  - **keyfile**: Specifies the file with the client secret key in the PEM format.
  - **keyfile_password**: Specifies a password for decrypting the client keyfile.

2.2.15 Watchdog

- **mode**: `off`, `automatic` or `required`. When `off` watchdog is disabled. When `automatic` watchdog will be used if available, but ignored if it is not. When `required` the node will not become a leader unless watchdog can be successfully enabled.
- **device**: Path to watchdog device. Defaults to `/dev/watchdog`.
- **safety_margin**: Number of seconds of safety margin between watchdog triggering and leader key expiration.

2.2.16 Tags

- **nofailover**: `true` or `false`, controls whether this node is allowed to participate in the leader race and become a leader. Defaults to `false`
- **clonefrom**: `true` or `false`. If set to `true` other nodes might prefer to use this node for bootstrap (take `pg_basebackup` from). If there are several nodes with `clonefrom` tag set to `true` the node to bootstrap from will be chosen randomly. The default value is `false`.
- **noloadbalance**: `true` or `false`. If set to `true` the node will return HTTP Status Code 503 for the `GET /replica` REST API health-check and therefore will be excluded from the load-balancing. Defaults to `false`.
- **replicatefrom**: The IP address/hostname of another replica. Used to support cascading replication.
- **nosync**: `true` or `false`. If set to `true` the node will never be selected as a synchronous replica.

In addition to these predefined tags, you can also add your own ones:

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

Tags are visible in the *REST API* and `patronictl list` You can also check for an instance health using these tags. If the tag isn't defined for an instance, or if the respective value doesn't match the querying value, it will return HTTP Status Code 503.

2.3 Environment Configuration Settings

It is possible to override some of the configuration parameters defined in the Patroni configuration file using the system environment variables. This document lists all environment variables handled by Patroni. The values set via those variables always take precedence over the ones set in the Patroni configuration file.

2.3.1 Global/Universal

- **PATRONI_CONFIGURATION**: it is possible to set the entire configuration for the Patroni via `PATRONI_CONFIGURATION` environment variable. In this case any other environment variables will not be considered!
- **PATRONI_NAME**: name of the node where the current instance of Patroni is running. Must be unique for the cluster.
- **PATRONI_NAMESPACE**: path within the configuration store where Patroni will keep information about the cluster. Default value: "/service"

- **PATRONI_SCOPE**: cluster name

2.3.2 Log

- **PATRONI_LOG_LEVEL**: sets the general logging level. Default value is **INFO** (see the docs for Python logging)
- **PATRONI_LOG_TRACEBACK_LEVEL**: sets the level where tracebacks will be visible. Default value is **ERROR**. Set it to **DEBUG** if you want to see tracebacks only if you enable **PATRONI_LOG_LEVEL=DEBUG**.
- **PATRONI_LOG_FORMAT**: sets the log formatting string. Default value is **%(asctime)s %(levelname)s: %(message)s** (see the LogRecord attributes)
- **PATRONI_LOG_DATEFORMAT**: sets the datetime formatting string. (see the formatTime() documentation)
- **PATRONI_LOG_MAX_QUEUE_SIZE**: Patroni is using two-step logging. Log records are written into the in-memory queue and there is a separate thread which pulls them from the queue and writes to stderr or file. The maximum size of the internal queue is limited by default by **1000** records, which is enough to keep logs for the past 1h20m.
- **PATRONI_LOG_DIR**: Directory to write application logs to. The directory must exist and be writable by the user executing Patroni. If you set this env variable, the application will retain 4 25MB logs by default. You can tune those retention values with *PATRONI_LOG_FILE_NUM* and *PATRONI_LOG_FILE_SIZE* (see below).
- **PATRONI_LOG_FILE_NUM**: The number of application logs to retain.
- **PATRONI_LOG_FILE_SIZE**: Size of patroni.log file (in bytes) that triggers a log rolling.
- **PATRONI_LOG_LOGGERS**: Redefine logging level per python module. Example `PATRONI_LOG_LOGGERS="{patroni.postmaster: WARNING, urllib3: DEBUG}"`

2.3.3 Bootstrap configuration

It is possible to create new database users right after the successful initialization of a new cluster. This process is defined by the following variables:

- **PATRONI_<username>_PASSWORD='<password>'**
- **PATRONI_<username>_OPTIONS='list,of,options'**

Example: defining `PATRONI_admin_PASSWORD=strongpasswd` and `PATRONI_admin_OPTIONS='createrole, createdb'` will cause creation of the user **admin** with the password **strongpasswd** that is allowed to create other users and databases.

2.3.4 Citus

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

- **PATRONI_CITUS_GROUP**: the Citus group id, integer. Use `0` for coordinator and `1`, `2`, etc... for workers
- **PATRONI_CITUS_DATABASE**: the database where `citus` extension should be created. Must be the same on the coordinator and all workers. Currently only one database is supported.

2.3.5 Consul

- **PATRONI_CONSUL_HOST**: the host:port for the Consul local agent.
- **PATRONI_CONSUL_URL**: url for the Consul local agent, in format: http(s)://host:port
- **PATRONI_CONSUL_PORT**: (optional) Consul port
- **PATRONI_CONSUL_SCHEME**: (optional) **http** or **https**, defaults to **http**
- **PATRONI_CONSUL_TOKEN**: (optional) ACL token
- **PATRONI_CONSUL_VERIFY**: (optional) whether to verify the SSL certificate for HTTPS requests
- **PATRONI_CONSUL_CACERT**: (optional) The ca certificate. If present it will enable validation.
- **PATRONI_CONSUL_CERT**: (optional) File with the client certificate
- **PATRONI_CONSUL_KEY**: (optional) File with the client key. Can be empty if the key is part of certificate.
- **PATRONI_CONSUL_DC**: (optional) Datacenter to communicate with. By default the datacenter of the host is used.
- **PATRONI_CONSUL_CONSISTENCY**: (optional) Select consul consistency mode. Possible values are `default`, `consistent`, or `stale` (more details in consul API reference)
- **PATRONI_CONSUL_CHECKS**: (optional) list of Consul health checks used for the session. By default an empty list is used.
- **PATRONI_CONSUL_REGISTER_SERVICE**: (optional) whether or not to register a service with the name defined by the scope parameter and the tag master, primary, replica, or standby-leader depending on the node's role. Defaults to **false**
- **PATRONI_CONSUL_SERVICE_TAGS**: (optional) additional static tags to add to the Consul service apart from the role (`master`/`primary`/`replica`/`standby-leader`). By default an empty list is used.
- **PATRONI_CONSUL_SERVICE_CHECK_INTERVAL**: (optional) how often to perform health check against registered url
- **PATRONI_CONSUL_SERVICE_CHECK_TLS_SERVER_NAME**: (optional) overide SNI host when connecting via TLS, see also consul agent check API reference.

2.3.6 Etcd

- **PATRONI_ETCD_PROXY**: proxy url for the etcd. If you are connecting to the etcd using proxy, use this parameter instead of **PATRONI_ETCD_URL**
- **PATRONI_ETCD_URL**: url for the etcd, in format: http(s)://(username:password@)host:port
- **PATRONI_ETCD_HOSTS**: list of etcd endpoints in format 'host1:port1','host2:port2',etc…
- **PATRONI_ETCD_USE_PROXIES**: If this parameter is set to true, Patroni will consider **hosts** as a list of proxies and will not perform a topology discovery of etcd cluster but stick to a fixed list of **hosts**.
- **PATRONI_ETCD_PROTOCOL**: http or https, if not specified http is used. If the **url** or **proxy** is specified - will take protocol from them.
- **PATRONI_ETCD_HOST**: the host:port for the etcd endpoint.
- **PATRONI_ETCD_SRV**: Domain to search the SRV record(s) for cluster autodiscovery. Patroni will try to query these SRV service names for specified domain (in that order until first success): `_etcd-client-ssl`, `_etcd-client`, `_etcd-ssl`, `_etcd`, `_etcd-server-ssl`, `_etcd-server`. If SRV records for `_etcd-server-ssl` or `_etcd-server` are retrieved then ETCD peer protocol is used do query ETCD for available members. Otherwise hosts from SRV records will be used.

- **PATRONI_ETCD_SRV_SUFFIX**: Configures a suffix to the SRV name that is queried during discovery. Use this flag to differentiate between multiple etcd clusters under the same domain. Works only with conjunction with **PATRONI_ETCD_SRV**. For example, if `PATRONI_ETCD_SRV_SUFFIX=foo` and `PATRONI_ETCD_SRV=example.org` are set, the following DNS SRV query is made:`_etcd-client-ssl-foo._tcp.example.com` (and so on for every possible ETCD SRV service name).
- **PATRONI_ETCD_USERNAME**: username for etcd authentication.
- **PATRONI_ETCD_PASSWORD**: password for etcd authentication.
- **PATRONI_ETCD_CACERT**: The ca certificate. If present it will enable validation.
- **PATRONI_ETCD_CERT**: File with the client certificate.
- **PATRONI_ETCD_KEY**: File with the client key. Can be empty if the key is part of certificate.

2.3.7 Etcdv3

Environment names for Etcdv3 are similar as for Etcd, you just need to use `ETCD3` instead of `ETCD` in the variable name. Example: `PATRONI_ETCD3_HOST`, `PATRONI_ETCD3_CACERT`, and so on.

Warning: Keys created with protocol version 2 are not visible with protocol version 3 and the other way around, therefore it is not possible to switch from Etcd to Etcdv3 just by updating Patroni configuration.

2.3.8 ZooKeeper

- **PATRONI_ZOOKEEPER_HOSTS**: Comma separated list of ZooKeeper cluster members: "'host1:port1','host2:port2','etc…'". It is important to quote every single entity!
- **PATRONI_ZOOKEEPER_USE_SSL**: (optional) Whether SSL is used or not. Defaults to `false`. If set to `false`, all SSL specific parameters are ignored.
- **PATRONI_ZOOKEEPER_CACERT**: (optional) The CA certificate. If present it will enable validation.
- **PATRONI_ZOOKEEPER_CERT**: (optional) File with the client certificate.
- **PATRONI_ZOOKEEPER_KEY**: (optional) File with the client key.
- **PATRONI_ZOOKEEPER_KEY_PASSWORD**: (optional) The client key password.
- **PATRONI_ZOOKEEPER_VERIFY**: (optional) Whether to verify certificate or not. Defaults to `true`.
- **PATRONI_ZOOKEEPER_SET_ACLS**: (optional) If set, configure Kazoo to apply a default ACL to each ZNode that it creates. ACLs will assume 'x509' schema and should be specified as a dictionary with the principal as the key and one or more permissions as a list in the value. Permissions may be one of `CREATE`, `READ`, `WRITE`, `DELETE` or `ADMIN`. For example, `set_acls: {CN=principal1: [CREATE, READ], CN=principal2: [ALL]}`.

Note: It is required to install `kazoo>=2.6.0` to support SSL.

2.3.9 Exhibitor

- **PATRONI_EXHIBITOR_HOSTS**: initial list of Exhibitor (ZooKeeper) nodes in format: 'host1,host2,etc...'. This list updates automatically whenever the Exhibitor (ZooKeeper) cluster topology changes.
- **PATRONI_EXHIBITOR_PORT**: Exhibitor port.

2.3.10 Kubernetes

- **PATRONI_KUBERNETES_BYPASS_API_SERVICE**: (optional) When communicating with the Kubernetes API, Patroni is usually relying on the *kubernetes* service, the address of which is exposed in the pods via the *KUBERNETES_SERVICE_HOST* environment variable. If *PATRONI_KUBERNETES_BYPASS_API_SERVICE* is set to `true`, Patroni will resolve the list of API nodes behind the service and connect directly to them.
- **PATRONI_KUBERNETES_NAMESPACE**: (optional) Kubernetes namespace where the Patroni pod is running. Default value is *default*.
- **PATRONI_KUBERNETES_LABELS**: Labels in format `{label1: value1, label2: value2}`. These labels will be used to find existing objects (Pods and either Endpoints or ConfigMaps) associated with the current cluster. Also Patroni will set them on every object (Endpoint or ConfigMap) it creates.
- **PATRONI_KUBERNETES_SCOPE_LABEL**: (optional) name of the label containing cluster name. Default value is *cluster-name*.
- **PATRONI_KUBERNETES_ROLE_LABEL**: (optional) name of the label containing role (master or replica or other custom value). Patroni will set this label on the pod it runs in. Default value is `role`.
- **PATRONI_KUBERNETES_LEADER_LABEL_VALUE**: (optional) value of the pod label when Postgres role is *master*. Default value is *master*.
- **PATRONI_KUBERNETES_FOLLOWER_LABEL_VALUE**: (optional) value of the pod label when Postgres role is *replica*. Default value is *replica*.
- **PATRONI_KUBERNETES_STANDBY_LEADER_LABEL_VALUE**: (optional) value of the pod label when Postgres role is `standby-leader`. Default value is `standby-leader`.
- **PATRONI_KUBERNETES_TMP_ROLE_LABEL**: (optional) name of the temporary label containing role (master or replica). Value of this label will always use the default of corresponding role. Set only when necessary.
- **PATRONI_KUBERNETES_USE_ENDPOINTS**: (optional) if set to true, Patroni will use Endpoints instead of ConfigMaps to run leader elections and keep cluster state.
- **PATRONI_KUBERNETES_POD_IP**: (optional) IP address of the pod Patroni is running in. This value is required when *PATRONI_KUBERNETES_USE_ENDPOINTS* is enabled and is used to populate the leader endpoint subsets when the pod's PostgreSQL is promoted.
- **PATRONI_KUBERNETES_PORTS**: (optional) if the Service object has the name for the port, the same name must appear in the Endpoint object, otherwise service won't work. For example, if your service is defined as `{Kind: Service, spec: {ports: [{name: postgresql, port: 5432, targetPort: 5432}]}}`, then you have to set `PATRONI_KUBERNETES_PORTS='[{"name": "postgresql", "port": 5432}]'` and Patroni will use it for updating subsets of the leader Endpoint. This parameter is used only if *PATRONI_KUBERNETES_USE_ENDPOINTS* is set.
- **PATRONI_KUBERNETES_CACERT**: (optional) Specifies the file with the CA_BUNDLE file with certificates of trusted CAs to use while verifying Kubernetes API SSL certs. If not provided, patroni will use the value provided by the ServiceAccount secret.
- **PATRONI_RETRIABLE_HTTP_CODES**: (optional) list of HTTP status codes from K8s API to retry on. By default Patroni is retrying on `500`, `503`, and `504`, or if K8s API response has `retry-after` HTTP header.

2.3.11 Raft (deprecated)

- **PATRONI_RAFT_SELF_ADDR**: `ip:port` to listen on for Raft connections. The `self_addr` must be accessible from other nodes of the cluster. If not set, the node will not participate in consensus.
- **PATRONI_RAFT_BIND_ADDR**: (optional) `ip:port` to listen on for Raft connections. If not specified the `self_addr` will be used.
- **PATRONI_RAFT_PARTNER_ADDRS**: list of other Patroni nodes in the cluster in format `"'ip1:port1', 'ip2:port2'"`. It is important to quote every single entity!
- **PATRONI_RAFT_DATA_DIR**: directory where to store Raft log and snapshot. If not specified the current working directory is used.
- **PATRONI_RAFT_PASSWORD**: (optional) Encrypt Raft traffic with a specified password, requires `cryptography` python module.

2.3.12 PostgreSQL

- **PATRONI_POSTGRESQL_LISTEN**: IP address + port that Postgres listens to. Multiple comma-separated addresses are permitted, as long as the port component is appended after to the last one with a colon, i.e. `listen: 127.0.0.1,127.0.0.2:5432`. Patroni will use the first address from this list to establish local connections to the PostgreSQL node.
- **PATRONI_POSTGRESQL_CONNECT_ADDRESS**: IP address + port through which Postgres is accessible from other nodes and applications.
- **PATRONI_POSTGRESQL_PROXY_ADDRESS**: IP address + port through which a connection pool (e.g. pgbouncer) running next to Postgres is accessible. The value is written to the member key in DCS as `proxy_url` and could be used/useful for service discovery.
- **PATRONI_POSTGRESQL_DATA_DIR**: The location of the Postgres data directory, either existing or to be initialized by Patroni.
- **PATRONI_POSTGRESQL_CONFIG_DIR**: The location of the Postgres configuration directory, defaults to the data directory. Must be writable by Patroni.
- **PATRONI_POSTGRESQL_BIN_DIR**: Path to PostgreSQL binaries. (pg_ctl, initdb, pg_controldata, pg_basebackup, postgres, pg_isready, pg_rewind) The default value is an empty string meaning that PATH environment variable will be used to find the executables.
- **PATRONI_POSTGRESQL_BIN_PG_CTL**: (optional) Custom name for `pg_ctl` binary.
- **PATRONI_POSTGRESQL_BIN_INITDB**: (optional) Custom name for `initdb` binary.
- **PATRONI_POSTGRESQL_BIN_PG_CONTROLDATA**: (optional) Custom name for `pg_controldata` binary.
- **PATRONI_POSTGRESQL_BIN_PG_BASEBACKUP**: (optional) Custom name for `pg_basebackup` binary.
- **PATRONI_POSTGRESQL_BIN_POSTGRES**: (optional) Custom name for `postgres` binary.
- **PATRONI_POSTGRESQL_BIN_IS_READY**: (optional) Custom name for `pg_isready` binary.
- **PATRONI_POSTGRESQL_BIN_PG_REWIND**: (optional) Custom name for `pg_rewind` binary.
- **PATRONI_POSTGRESQL_PGPASS**: path to the .pgpass password file. Patroni creates this file before executing pg_basebackup and under some other circumstances. The location must be writable by Patroni.
- **PATRONI_REPLICATION_USERNAME**: replication username; the user will be created during initialization. Replicas will use this user to access the replication source via streaming replication

- **PATRONI_REPLICATION_PASSWORD**: replication password; the user will be created during initialization.
- **PATRONI_REPLICATION_SSLMODE**: (optional) maps to the sslmode connection parameter, which allows a client to specify the type of TLS negotiation mode with the server. For more information on how each mode works, please visit the PostgreSQL documentation. The default mode is `prefer`.
- **PATRONI_REPLICATION_SSLKEY**: (optional) maps to the sslkey connection parameter, which specifies the location of the secret key used with the client's certificate.
- **PATRONI_REPLICATION_SSLPASSWORD**: (optional) maps to the sslpassword connection parameter, which specifies the password for the secret key specified in `PATRONI_REPLICATION_SSLKEY`.
- **PATRONI_REPLICATION_SSLCERT**: (optional) maps to the sslcert connection parameter, which specifies the location of the client certificate.
- **PATRONI_REPLICATION_SSLROOTCERT**: (optional) maps to the sslrootcert connection parameter, which specifies the location of a file containing one ore more certificate authorities (CA) certificates that the client will use to verify a server's certificate.
- **PATRONI_REPLICATION_SSLCRL**: (optional) maps to the sslcrl connection parameter, which specifies the location of a file containing a certificate revocation list. A client will reject connecting to any server that has a certificate present in this list.
- **PATRONI_REPLICATION_SSLCRLDIR**: (optional) maps to the sslcrldir connection parameter, which specifies the location of a directory with files containing a certificate revocation list. A client will reject connecting to any server that has a certificate present in this list.
- **PATRONI_REPLICATION_GSSENCMODE**: (optional) maps to the gssencmode connection parameter, which determines whether or with what priority a secure GSS TCP/IP connection will be negotiated with the server
- **PATRONI_REPLICATION_CHANNEL_BINDING**: (optional) maps to the channel_binding connection parameter, which controls the client's use of channel binding.
- **PATRONI_SUPERUSER_USERNAME**: name for the superuser, set during initialization (initdb) and later used by Patroni to connect to the postgres. Also this user is used by pg_rewind.
- **PATRONI_SUPERUSER_PASSWORD**: password for the superuser, set during initialization (initdb).
- **PATRONI_SUPERUSER_SSLMODE**: (optional) maps to the sslmode connection parameter, which allows a client to specify the type of TLS negotiation mode with the server. For more information on how each mode works, please visit the PostgreSQL documentation. The default mode is `prefer`.
- **PATRONI_SUPERUSER_SSLKEY**: (optional) maps to the sslkey connection parameter, which specifies the location of the secret key used with the client's certificate.
- **PATRONI_SUPERUSER_SSLPASSWORD**: (optional) maps to the sslpassword connection parameter, which specifies the password for the secret key specified in `PATRONI_SUPERUSER_SSLKEY`.
- **PATRONI_SUPERUSER_SSLCERT**: (optional) maps to the sslcert connection parameter, which specifies the location of the client certificate.
- **PATRONI_SUPERUSER_SSLROOTCERT**: (optional) maps to the sslrootcert connection parameter, which specifies the location of a file containing one ore more certificate authorities (CA) certificates that the client will use to verify a server's certificate.
- **PATRONI_SUPERUSER_SSLCRL**: (optional) maps to the sslcrl connection parameter, which specifies the location of a file containing a certificate revocation list. A client will reject connecting to any server that has a certificate present in this list.
- **PATRONI_SUPERUSER_SSLCRLDIR**: (optional) maps to the sslcrldir connection parameter, which specifies the location of a directory with files containing a certificate revocation list. A client will reject connecting to any server that has a certificate present in this list.

- **PATRONI_SUPERUSER_GSSENCMODE**: (optional) maps to the gssencmode connection parameter, which determines whether or with what priority a secure GSS TCP/IP connection will be negotiated with the server
- **PATRONI_SUPERUSER_CHANNEL_BINDING**: (optional) maps to the channel_binding connection parameter, which controls the client's use of channel binding.
- **PATRONI_REWIND_USERNAME**: (optional) name for the user for `pg_rewind`; the user will be created during initialization of postgres 11+ and all necessary permissions will be granted.
- **PATRONI_REWIND_PASSWORD**: (optional) password for the user for `pg_rewind`; the user will be created during initialization.
- **PATRONI_REWIND_SSLMODE**: (optional) maps to the sslmode connection parameter, which allows a client to specify the type of TLS negotiation mode with the server. For more information on how each mode works, please visit the PostgreSQL documentation. The default mode is `prefer`.
- **PATRONI_REWIND_SSLKEY**: (optional) maps to the sslkey connection parameter, which specifies the location of the secret key used with the client's certificate.
- **PATRONI_REWIND_SSLPASSWORD**: (optional) maps to the sslpassword connection parameter, which specifies the password for the secret key specified in `PATRONI_REWIND_SSLKEY`.
- **PATRONI_REWIND_SSLCERT**: (optional) maps to the sslcert connection parameter, which specifies the location of the client certificate.
- **PATRONI_REWIND_SSLROOTCERT**: (optional) maps to the sslrootcert connection parameter, which specifies the location of a file containing one ore more certificate authorities (CA) certificates that the client will use to verify a server's certificate.
- **PATRONI_REWIND_SSLCRL**: (optional) maps to the sslcrl connection parameter, which specifies the location of a file containing a certificate revocation list. A client will reject connecting to any server that has a certificate present in this list.
- **PATRONI_REWIND_SSLCRLDIR**: (optional) maps to the sslcrldir connection parameter, which specifies the location of a directory with files containing a certificate revocation list. A client will reject connecting to any server that has a certificate present in this list.
- **PATRONI_REWIND_GSSENCMODE**: (optional) maps to the gssencmode connection parameter, which determines whether or with what priority a secure GSS TCP/IP connection will be negotiated with the server
- **PATRONI_REWIND_CHANNEL_BINDING**: (optional) maps to the channel_binding connection parameter, which controls the client's use of channel binding.

2.3.13 REST API

- **PATRONI_RESTAPI_CONNECT_ADDRESS**: IP address and port to access the REST API.
- **PATRONI_RESTAPI_LISTEN**: IP address and port that Patroni will listen to, to provide health-check information for HAProxy.
- **PATRONI_RESTAPI_USERNAME**: Basic-auth username to protect unsafe REST API endpoints.
- **PATRONI_RESTAPI_PASSWORD**: Basic-auth password to protect unsafe REST API endpoints.
- **PATRONI_RESTAPI_CERTFILE**: Specifies the file with the certificate in the PEM format. If the certfile is not specified or is left empty, the API server will work without SSL.
- **PATRONI_RESTAPI_KEYFILE**: Specifies the file with the secret key in the PEM format.
- **PATRONI_RESTAPI_KEYFILE_PASSWORD**: Specifies a password for decrypting the keyfile.
- **PATRONI_RESTAPI_CAFILE**: Specifies the file with the CA_BUNDLE with certificates of trusted CAs to use while verifying client certs.

- **PATRONI_RESTAPI_CIPHERS**: (optional) Specifies the permitted cipher suites (e.g. "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` (default), `optional` or `required`. When `none` REST API will not check client certificates. When `required` client certificates are required for all REST API calls. When `optional` client certificates are required for all unsafe REST API endpoints. When `required` is used, then client authentication succeeds, if the certificate signature verification succeeds. For `optional` the client cert will only be checked for `PUT`, `POST`, `PATCH`, and `DELETE` requests.
- **PATRONI_RESTAPI_ALLOWLIST**: (optional): Specifies the set of hosts that are allowed to call unsafe REST API endpoints. The single element could be a host name, an IP address or a network address using CIDR notation. By default `allow all` is used. In case if `allowlist` or `allowlist_include_members` are set, anything that is not included is rejected.
- **PATRONI_RESTAPI_ALLOWLIST_INCLUDE_MEMBERS**: (optional): If set to `true` it allows accessing unsafe REST API endpoints from other cluster members registered in DCS (IP address or hostname is taken from the members `api_url`). Be careful, it might happen that OS will use a different IP for outgoing connections.
- **PATRONI_RESTAPI_HTTP_EXTRA_HEADERS**: (optional) HTTP headers let the REST API server pass additional information with an HTTP response.
- **PATRONI_RESTAPI_HTTPS_EXTRA_HEADERS**: (optional) HTTPS headers let the REST API server pass additional information with an HTTP response when TLS is enabled. This will also pass additional information set in `http_extra_headers`.
- **PATRONI_RESTAPI_REQUEST_QUEUE_SIZE**: (optional): Sets request queue size for TCP socket used by Patroni REST API. Once the queue is full, further requests get a "Connection denied" error. The default value is 5.

Warning:

- The `PATRONI_RESTAPI_CONNECT_ADDRESS` must be accessible from all nodes of a given Patroni cluster. Internally Patroni is using it during the leader race to find nodes with minimal replication lag.
- If you enabled client certificates validation (`PATRONI_RESTAPI_VERIFY_CLIENT` is set to `required`), you also **must** provide **valid client certificates** in the `PATRONI_CTL_CERTFILE`, `PATRONI_CTL_KEYFILE`, `PATRONI_CTL_KEYFILE_PASSWORD`. If not provided, Patroni will not work correctly.

2.3.14 CTL

- **PATRONICTL_CONFIG_FILE**: (optional) location of the configuration file.
- **PATRONI_CTL_USERNAME**: (optional) Basic-auth username for accessing protected REST API endpoints. If not provided patronictl will use the value provided for REST API "username" parameter.
- **PATRONI_CTL_PASSWORD**: (optional) Basic-auth password for accessing protected REST API endpoints. If not provided patronictl will use the value provided for REST API "password" parameter.
- **PATRONI_CTL_INSECURE**: (optional) Allow connections to REST API without verifying SSL certs.
- **PATRONI_CTL_CACERT**: (optional) Specifies the file with the CA_BUNDLE file or directory with certificates of trusted CAs to use while verifying REST API SSL certs. If not provided patronictl will use the value provided for REST API "cafile" parameter.
- **PATRONI_CTL_CERTFILE**: (optional) Specifies the file with the client certificate in the PEM format.
- **PATRONI_CTL_KEYFILE**: (optional) Specifies the file with the client secret key in the PEM format.

- **PATRONI_CTL_KEYFILE_PASSWORD**: (optional) Specifies a password for decrypting the client keyfile.

There are 3 types of Patroni configuration:

- **Global *dynamic configuration*.**
  These options are stored in the DCS (Distributed Configuration Store) and applied on all cluster nodes. Dynamic configuration can be set at any time using `patronictl edit-config` tool or Patroni *REST API*. If the options changed are not part of the startup configuration, they are applied asynchronously (upon the next wake up cycle) to every node, which gets subsequently reloaded. If the node requires a restart to apply the configuration (for PostgreSQL parameters with context postmaster, if their values have changed), a special flag `pending_restart` indicating this is set in the members.data JSON. Additionally, the node status indicates this by showing `"restart_pending": true`.
- **Local *configuration file* (patroni.yml).**
  These options are defined in the configuration file and take precedence over dynamic configuration. `patroni.yml` can be changed and reloaded at runtime (without restart of Patroni) by sending SIGHUP to the Patroni process, performing `POST /reload` REST-API request or executing `patronictl reload`. Local configuration can be either a single YAML file or a directory. When it is a directory, all YAML files in that directory are loaded one by one in sorted order. In case a key is defined in multiple files, the occurrence in the last file takes precedence.
- ***Environment configuration*.**
  It is possible to set/override some of the "Local" configuration parameters with environment variables. Environment configuration is very useful when you are running in a dynamic environment and you don't know some of the parameters in advance (for example it's not possible to know your external IP address when you are running inside `docker`).

2.4 Important rules

2.4.1 PostgreSQL parameters controlled by Patroni

Some of the PostgreSQL parameters **must hold the same values on the primary and the replicas**. For those, **values set either in the local patroni configuration files or via the environment variables take no effect**. To alter or set their values one must change the shared configuration in the DCS. Below is the actual list of such parameters together with the default values:

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

For the parameters below, PostgreSQL does not require equal values among the primary and all the replicas. However, considering the possibility of a replica to become the primary at any time, it doesn't really make sense to set them differently; therefore, **Patroni restricts setting their values to the** *dynamic configuration*.

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

These parameters are validated to ensure they are sane, or meet a minimum value.

There are some other Postgres parameters controlled by Patroni:

- **listen_addresses** - is set either from `postgresql.listen` or from `PATRONI_POSTGRESQL_LISTEN` environment variable
- **port** - is set either from `postgresql.listen` or from `PATRONI_POSTGRESQL_LISTEN` environment variable
- **cluster_name** - is set either from `scope` or from `PATRONI_SCOPE` environment variable
- **hot_standby: on**
- **wal_log_hints: on** - for Postgres 9.4 and newer.

To be on the safe side parameters from the above lists are not written into `postgresql.conf`, but passed as a list of arguments to the `pg_ctl start` which gives them the highest precedence, even above ALTER SYSTEM

There also are some parameters like **postgresql.listen**, **postgresql.data_dir** that **can be set only locally**, i.e. in the Patroni *config file* or via *configuration* variable. In most cases the local configuration will override the dynamic configuration.

When applying the local or dynamic configuration options, the following actions are taken:

- The node first checks if there is a *postgresql.base.conf* or if the `custom_conf` parameter is set.
- If the `custom_conf` parameter is set, it will take the file specified on it as a base configuration, ignoring *postgresql.base.conf* and *postgresql.conf*.
- If the `custom_conf` parameter is not set and *postgresql.base.conf* exists, it contains the renamed "original" configuration and it will be used as a base configuration.
- If there is no `custom_conf`` nor *postgresql.base.conf*, the original *postgresql.conf*` is taken and renamed to postgresql.base.conf.
- The dynamic options (with the exceptions above) are dumped into the *postgresql.conf*` and an include is set in postgresql.conf to the used base configuration (either *postgresql.base.conf* or what is on `custom_conf`). Therefore, we would be able to apply new options without re-reading the configuration file to check if the include is present not.
- Some parameters that are essential for Patroni to manage the cluster are overridden using the command line.
- If some of the options that require restart are changed (we should look at the context in pg_settings and at the actual values of those options), a pending_restart flag of a given node is set. This flag is reset on any restart.

The parameters would be applied in the following order (run-time are given the highest priority):

1. load parameters from file *postgresql.base.conf* (or from a `custom_conf` file, if set)
2. load parameters from file *postgresql.conf*
3. load parameters from file *postgresql.auto.conf*
4. run-time parameter using *-o –name=value*

This allows configuration for all the nodes (2), configuration for a specific node using `ALTER SYSTEM` (3) and ensures that parameters essential to the running of Patroni are enforced (4), as well as leaves room for configuration tools that manage *postgresql.conf* directly without involving Patroni (1).

2.4.2 Patroni configuration parameters

Also the following Patroni configuration options **can be changed only dynamically**:

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

Upon changing these options, Patroni will read the relevant section of the configuration stored in DCS and change its run-time values.

Patroni nodes are dumping the state of the DCS options to disk upon for every change of the configuration into the file `patroni.dynamic.json` located in the Postgres data directory. Only the leader is allowed to restore these options from the on-disk dump if these are completely absent from the DCS or if they are invalid.

CHAPTER

THREE

PATRONI REST API

Patroni has a rich REST API, which is used by Patroni itself during the leader race, by the `patronictl` tool in order to perform failovers/switchovers/reinitialize/restarts/reloads, by HAProxy or any other kind of load balancer to perform HTTP health checks, and of course could also be used for monitoring. Below you will find the list of Patroni REST API endpoints.

3.1 Health check endpoints

For all health check `GET` requests Patroni returns a JSON document with the status of the node, along with the HTTP status code. If you don't want or don't need the JSON document, you might consider using the `HEAD` or `OPTIONS` method instead of `GET`.

- The following requests to Patroni REST API will return HTTP status code **200** only when the Patroni node is running as the primary with leader lock:
  - `GET /`
  - `GET /primary`
  - `GET /read-write`
- `GET /standby-leader`: returns HTTP status code **200** only when the Patroni node is running as the leader in a *standby cluster*.
- `GET /leader`: returns HTTP status code **200** when the Patroni node has the leader lock. The major difference from the two previous endpoints is that it doesn't take into account whether PostgreSQL is running as the `primary` or the `standby_leader`.
- `GET /replica`: replica health check endpoint. It returns HTTP status code **200** only when the Patroni node is in the state `running`, the role is `replica` and `noloadbalance` tag is not set.
- `GET /replica?lag=<max-lag>`: replica check endpoint. In addition to checks from `replica`, it also checks replication latency and returns status code **200** only when it is below specified value. The key cluster.last_leader_operation from DCS is used for Leader wal position and compute latency on replica for performance reasons. max-lag can be specified in bytes (integer) or in human readable values, for e.g. 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`: replica check endpoint. In addition, It will also check for user defined tags `key1` and `key2` and their respective values in the **tags** section of the yaml configuration management. If the tag isn't defined for an instance, or if the value in the yaml configuration doesn't match the querying value, it will return HTTP Status Code 503.

  In the following requests, since we are checking for the leader or standby-leader status, Patroni doesn't apply any of the user defined tags and they will be ignored.

  - `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`: like the above endpoint, but also includes the primary.
- `GET /synchronous` or `GET /sync`: returns HTTP status code **200** only when the Patroni node is running as a synchronous standby.
- `GET /read-only-sync`: like the above endpoint, but also includes the primary.
- `GET /asynchronous` or `GET /async`: returns HTTP status code **200** only when the Patroni node is running as an asynchronous standby.
- `GET /asynchronous?lag=<max-lag>` or `GET /async?lag=<max-lag>`: asynchronous standby check endpoint. In addition to checks from `asynchronous` or `async`, it also checks replication latency and returns status code **200** only when it is below specified value. The key cluster.last_leader_operation from DCS is used for Leader wal position and compute latency on replica for performance reasons. max-lag can be specified in bytes (integer) or in human readable values, for e.g. 16kB, 64MB, 1GB.
  - `GET /async?lag=1048576`
  - `GET /async?lag=1024kB`
  - `GET /async?lag=10MB`
  - `GET /async?lag=1GB`
- `GET /health`: returns HTTP status code **200** only when PostgreSQL is up and running.
- `GET /liveness`: returns HTTP status code **200** if Patroni heartbeat loop is properly running and **503** if the last run was more than `ttl` seconds ago on the primary or `2*ttl` on the replica. Could be used for `livenessProbe`.
- `GET /readiness`: returns HTTP status code **200** when the Patroni node is running as the leader or when PostgreSQL is up and running. The endpoint could be used for `readinessProbe` when it is not possible to use Kubernetes endpoints for leader elections (OpenShift).

Both, `readiness` and `liveness` endpoints are very light-weight and not executing any SQL. Probes should be configured in such a way that they start failing about time when the leader key is expiring. With the default value of `ttl`, which is `30s` example probes would look like:

```
readinessProbe:
  httpGet:
    scheme: HTTP
    path: /readiness
    port: 8008
  initialDelaySeconds: 3
```

(continues on next page)

(continued from previous page)

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

3.2 Monitoring endpoint

The `GET /patroni` is used by Patroni during the leader race. It also could be used by your monitoring system. The JSON document produced by this endpoint has the same structure as the JSON produced by the health check endpoints.

Example: A healthy cluster

```
$ 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,
  "replication": [
    {
      "usename": "replicator",
      "application_name": "patroni2",
      "client_addr": "10.89.0.6",
      "state": "streaming",
      "sync_state": "async",
      "sync_priority": 0
    },
    {
      "usename": "replicator",
      "application_name": "patroni3",
      "client_addr": "10.89.0.2",
      "state": "streaming",
      "sync_state": "async",
      "sync_priority": 0
    }
  ],
  "dcs_last_seen": 1692356718,
```

(continues on next page)

(continued from previous page)

```
  "tags": {
    "clonefrom": true
  },
  "database_system_identifier": "7268616322854375442",
  "patroni": {
    "version": "3.1.0",
    "scope": "demo"
  }
}
```

Example: An unlocked cluster

```
$ 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,
  "replication": [
    {
      "usename": "replicator",
      "application_name": "patroni2",
      "client_addr": "10.89.0.6",
      "state": "streaming",
      "sync_state": "async",
      "sync_priority": 0
    },
    {
      "usename": "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"
```

(continues on next page)

(continued from previous page)

```
  }
}
```

Example: An unlocked cluster with *DCS failsafe mode* enabled

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

Example: A cluster with the *pause mode* enabled

```
$ curl -s http://localhost:8008/patroni  | jq .
{
  "state": "running",
  "postmaster_start_time": "2023-08-18 11:09:08.615242+00:00",
```

(continues on next page)

(continued from previous page)


```
  "role": "replica",
  "server_version": 150004,
  "xlog": {
    "location": 67420024
  },
  "timeline": 1,
  "replication": [
    {
      "usename": "replicator",
      "application_name": "patroni2",
      "client_addr": "10.89.0.6",
      "state": "streaming",
      "sync_state": "async",
      "sync_priority": 0
    },
    {
      "usename": "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"
  }
}
```


Retrieve the Patroni metrics in Prometheus format through the `GET /metrics` endpoint.

```
$ curl http://localhost:8008/metrics

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

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
patroni_postgres_timeline{scope="batman"} 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"} 1677658321
# HELP patroni_pending_restart Value is 1 if the node needs a restart, 0 otherwise.
# TYPE patroni_pending_restart gauge
patroni_pending_restart{scope="batman"} 1
# HELP patroni_is_paused Value is 1 if auto failover is disabled, 0 otherwise.
# TYPE patroni_is_paused gauge
patroni_is_paused{scope="batman"} 1
```

3.3 Cluster status endpoints

- The `GET /cluster` endpoint generates a JSON document describing the current cluster topology and state:

```
$ curl -s http://localhost:8008/cluster | jq .
{
  "members": [
    {
      "name": "patroni1",
      "role": "leader",
      "state": "running",
      "api_url": "http://10.89.0.4:8008/patroni",
      "host": "10.89.0.4",
      "port": 5432,
      "timeline": 5,
      "tags": {
        "clonefrom": true
      }
    },
    {
      "name": "patroni2",
      "role": "replica",
      "state": "streaming",
      "api_url": "http://10.89.0.6:8008/patroni",
      "host": "10.89.0.6",
      "port": 5433,
      "timeline": 5,
      "tags": {
        "clonefrom": true
      },
      "lag": 0
    }
  ],
  "scheduled_switchover": {
    "at": "2023-09-24T10:36:00+02:00",
    "from": "patroni1",
    "to": "patroni3"
  }
```

(continues on next page)

(continued from previous page)

```
}
```

- The `GET /history` endpoint provides a view on the history of cluster switchovers/failovers. The format is very similar to the content of history files in the `pg_wal` directory. The only difference is the timestamp field showing when the new timeline was created.

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

3.4 Config endpoint

`GET /config`: Get the current version of the dynamic configuration:

```
$ 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",
```

(continues on next page)

(continued from previous page)

```
      "max_wal_senders": 5,
      "max_replication_slots": 5,
      "max_connections": "100"
    }
  }
}
```

`PATCH /config`: Change the existing configuration.

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

The above REST API call patches the existing configuration and returns the new configuration.

Let's check that the node processed this configuration. First of all it should start printing log lines every 5 seconds (loop_wait=5). The change of "max_connections" requires a restart, so the "pending_restart" flag should be exposed:

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

Removing parameters:

If you want to remove (reset) some setting just patch it with `null`:

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

The above call removes `postgresql.parameters.max_connections` from the dynamic configuration.

`PUT /config`: It's also possible to perform the full rewrite of an existing dynamic configuration unconditionally:

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

3.5 Switchover and failover endpoints

`POST /switchover` or `POST /failover`. These endpoints are very similar to each other. There are a couple of minor differences though:

1. The failover endpoint allows to perform a manual failover when there are no healthy nodes, but at the same time it will not allow you to schedule a switchover.
2. The switchover endpoint is the opposite. It works only when the cluster is healthy (there is a leader) and allows to schedule a switchover at a given time.

In the JSON body of the `POST` request you must specify at least the `leader` or `candidate` fields and optionally the `scheduled_at` field if you want to schedule a switchover at a specific time.

Example: perform a failover to the specific node:

```
$ curl -s http://localhost:8009/failover -XPOST -d '{"candidate":"postgresql1"}'
Successfully failed over to "postgresql1"
```

Example: schedule a switchover from the leader to any other healthy replica in the cluster at a specific time:

```
$ curl -s http://localhost:8008/switchover -XPOST -d \
        '{"leader":"postgresql0","scheduled_at":"2019-09-24T12:00+00"}'
Switchover scheduled
```

Depending on the situation the request might finish with a different HTTP status code and body. The status code **200** is returned when the switchover or failover successfully completed. If the switchover was successfully scheduled, Patroni will return HTTP status code **202**. In case something went wrong, the error status code (one of **400**, **412** or **503**) will be returned with some details in the response body. For more information please check the source code of `patroni/api.py:do_POST_failover()` method.

- `DELETE /switchover`: delete the scheduled switchover

The `POST /switchover` and `POST failover` endpoints are used by `patronictl switchover` and `patronictl failover`, respectively. The `DELETE /switchover` is used by `patronictl flush <cluster-name> switchover`.

3.6 Restart endpoint

- `POST /restart`: You can restart Postgres on the specific node by performing the `POST /restart` call. In the JSON body of `POST` request it is possible to optionally specify some restart conditions:
  - **restart_pending**: boolean, if set to `true` Patroni will restart PostgreSQL only when restart is pending in order to apply some changes in the PostgreSQL config.
  - **role**: perform restart only if the current role of the node matches with the role from the POST request.
  - **postgres_version**: perform restart only if the current version of postgres is smaller than specified in the POST request.
  - **timeout**: how long we should wait before PostgreSQL starts accepting connections. Overrides `primary_start_timeout`.
  - **schedule**: timestamp with time zone, schedule the restart somewhere in the future.
- `DELETE /restart`: delete the scheduled restart

`POST /restart` and `DELETE /restart` endpoints are used by `patronictl restart` and `patronictl flush <cluster-name> restart` respectively.

3.7 Reload endpoint

The `POST /reload` call will order Patroni to re-read and apply the configuration file. This is the equivalent of sending the `SIGHUP` signal to the Patroni process. In case you changed some of the Postgres parameters which require a restart (like **shared_buffers**), you still have to explicitly do the restart of Postgres by either calling the `POST /restart` endpoint or with the help of `patronictl restart`.

The reload endpoint is used by `patronictl reload`.

3.8 Reinitialize endpoint

`POST /reinitialize`: reinitialize the PostgreSQL data directory on the specified node. It is allowed to be executed only on replicas. Once called, it will remove the data directory and start `pg_basebackup` or some alternative *replica creation method*.

The call might fail if Patroni is in a loop trying to recover (restart) a failed Postgres. In order to overcome this problem one can specify `{"force":true}` in the request body.

The reinitialize endpoint is used by `patronictl reinit`.

CHAPTER

FOUR

REPLICA IMAGING AND BOOTSTRAP

Patroni allows customizing creation of a new replica. It also supports defining what happens when the new empty cluster is being bootstrapped. The distinction between two is well defined: Patroni creates replicas only if the `initialize` key is present in DCS for the cluster. If there is no `initialize` key - Patroni calls bootstrap exclusively on the first node that takes the initialize key lock.

4.1 Bootstrap

PostgreSQL provides `initdb` command to initialize a new cluster and Patroni calls it by default. In certain cases, particularly when creating a new cluster as a copy of an existing one, it is necessary to replace a built-in method with custom actions. Patroni supports executing user-defined scripts to bootstrap new clusters, supplying some required arguments to them, i.e. the name of the cluster and the path to the data directory. This is configured in the `bootstrap` section of the Patroni configuration. For example:

```
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>
```

Each bootstrap method must define at least a `name` and a `command`. A special `initdb` method is available to trigger the default behavior, in which case `method` parameter can be omitted altogether. The `command` can be specified using either an absolute path, or the one relative to the `patroni` command location. In addition to the fixed parameters defined in the configuration files, Patroni supplies two cluster-specific ones:

--scope	Name of the cluster to be bootstrapped
--datadir	Path to the data directory of the cluster instance to be bootstrapped

Passing these two additional flags can be disabled by setting a special `no_params` parameter to `True`.

If the bootstrap script returns 0, Patroni tries to configure and start the PostgreSQL instance produced by it. If any of the intermediate steps fail, or the script returns a non-zero value, Patroni assumes that the bootstrap has failed, cleans up after itself and releases the initialize lock to give another node the opportunity to bootstrap.

If a `recovery_conf` block is defined in the same section as the custom bootstrap method, Patroni will generate a `recovery.conf` before starting the newly bootstrapped instance. Typically, such recovery.conf should contain at least one of the `recovery_target_*` parameters, together with the `recovery_target_timeline` set to `promote`.

If `keep_existing_recovery_conf` is defined and set to `True`, Patroni will not remove the existing `recovery.conf` file if it exists. This is useful when bootstrapping from a backup with tools like pgBackRest that generate the appropriate `recovery.conf` for you.

> **Note:** Bootstrap methods are neither chained, nor fallen-back to the default one in case the primary one fails

4.2 Building replicas

Patroni uses tried and proven `pg_basebackup` in order to create new replicas. One downside of it is that it requires a running leader node. Another one is the lack of 'on-the-fly' compression for the backup data and no built-in cleanup for outdated backup files. Some people prefer other backup solutions, such as `WAL-E`, `pgBackRest`, `Barman` and others, or simply roll their own scripts. In order to accommodate all those use-cases Patroni supports running custom scripts to clone a new replica. Those are configured in the `postgresql` configuration block:

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

example: 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'
```

example: pgbackrest

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

The `create_replica_methods` defines available replica creation methods and the order of executing them. Patroni will stop on the first one that returns 0. Each method should define a separate section in the configuration file, listing the command to execute and any custom parameters that should be passed to that command. All parameters will be passed in a `--name=value` format. Besides user-defined parameters, Patroni supplies a couple of cluster-specific ones:

--scope	Which cluster this replica belongs to
--datadir	Path to the data directory of the replica
--role	Always 'replica'
--connstring	Connection string to connect to the cluster member to clone from (primary or other replica). The user in the connection string can execute SQL and replication protocol commands.

A special `no_leader` parameter, if defined, allows Patroni to call the replica creation method even if there is no running leader or replicas. In that case, an empty string will be passed in a connection string. This is useful for restoring the formerly running cluster from the binary backup.

A special `keep_data` parameter, if defined, will instruct Patroni to not clean PGDATA folder before calling restore.

A special `no_params` parameter, if defined, restricts passing parameters to custom command.

A `basebackup` method is a special case: it will be used if `create_replica_methods` is empty, although it is possible to list it explicitly among the `create_replica_methods` methods. This method initializes a new replica with the `pg_basebackup`, the base backup is taken from the leader unless there are replicas with `clonefrom` tag, in which case one of such replicas will be used as the origin for pg_basebackup. It works without any configuration; however, it is possible to specify a `basebackup` configuration section. Same rules as with the other method configuration apply, namely, only long (with –) options should be specified there. Not all parameters make sense, if you override a connection string or provide an option to created tar-ed or compressed base backups, patroni won't be able to make a replica out of it. There is no validation performed on the names or values of the parameters passed to the `basebackup` section. Also note that in case symlinks are used for the WAL folder it is up to the user to specify the correct `--waldir` path as an option, so that after replica buildup or re-initialization the symlink would persist. This option is supported only since v10 though.

You can specify basebackup parameters as either a map (key-value pairs) or a list of elements, where each element could be either a key-value pair or a single key (for options that does not receive any values, for instance, `--verbose`). Consider those 2 examples:

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

and

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

If all replica creation methods fail, Patroni will try again all methods in order during the next event loop cycle.

4.3 Standby cluster

Another available option is to run a "standby cluster", that contains only of standby nodes replicating from some remote node. This type of clusters has:

- "standby leader", that behaves pretty much like a regular cluster leader, except it replicates from a remote node.
- cascade replicas, that are replicating from standby leader.

Standby leader holds and updates a leader lock in DCS. If the leader lock expires, cascade replicas will perform an election to choose another leader from the standbys.

There is no further relationship between the standby cluster and the primary cluster it replicates from, in particular, they must not share the same DCS scope if they use the same DCS. They do not know anything else from each other apart from replication information. Also, the standby cluster is not being displayed in `patronictl list` or `patronictl topology` output on the primary cluster.

For the sake of flexibility, you can specify methods of creating a replica and recovery WAL records when a cluster is in the "standby mode" by providing *create_replica_methods* key in *standby_cluster* section. It is distinct from creating replicas, when cluster is detached and functions as a normal cluster, which is controlled by *create_replica_methods* in *postgresql* section. Both "standby" and "normal" *create_replica_methods* reference keys in *postgresql* section.

To configure such cluster you need to specify the section `standby_cluster` in a patroni configuration:

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

Note, that these options will be applied only once during cluster bootstrap, and the only way to change them afterwards is through DCS.

Patroni expects to find *postgresql.conf* or *postgresql.conf.backup* in PGDATA of the remote primary and will not start if it does not find it after a basebackup. If the remote primary keeps its *postgresql.conf* elsewhere, it is your responsibility to copy it to PGDATA.

If you use replication slots on the standby cluster, you must also create the corresponding replication slot on the primary cluster. It will not be done automatically by the standby cluster implementation. You can use Patroni's permanent replication slots feature on the primary cluster to maintain a replication slot with the same name as `primary_slot_name`, or its default value if `primary_slot_name` is not provided.

CHAPTER

FIVE

REPLICATION MODES

Patroni uses PostgreSQL streaming replication. For more information about streaming replication, see the Postgres documentation. By default Patroni configures PostgreSQL for asynchronous replication. Choosing your replication schema is dependent on your business considerations. Investigate both async and sync replication, as well as other HA solutions, to determine which solution is best for you.

5.1 Asynchronous mode durability

In asynchronous mode the cluster is allowed to lose some committed transactions to ensure availability. When the primary server fails or becomes unavailable for any other reason Patroni will automatically promote a sufficiently healthy standby to primary. Any transactions that have not been replicated to that standby remain in a "forked timeline" on the primary, and are effectively unrecoverable[1].

The amount of transactions that can be lost is controlled via `maximum_lag_on_failover` parameter. Because the primary transaction log position is not sampled in real time, in reality the amount of lost data on failover is worst case bounded by `maximum_lag_on_failover` bytes of transaction log plus the amount that is written in the last `ttl` seconds (`loop_wait`/2 seconds in the average case). However typical steady state replication delay is well under a second.

By default, when running leader elections, Patroni does not take into account the current timeline of replicas, what in some cases could be undesirable behavior. You can prevent the node not having the same timeline as a former primary become the new leader by changing the value of `check_timeline` parameter to `true`.

5.2 PostgreSQL synchronous replication

You can use Postgres's synchronous replication with Patroni. Synchronous replication ensures consistency across a cluster by confirming that writes are written to a secondary before returning to the connecting client with a success. The cost of synchronous replication: reduced throughput on writes. This throughput will be entirely based on network performance.

In hosted datacenter environments (like AWS, Rackspace, or any network you do not control), synchronous replication significantly increases the variability of write performance. If followers become inaccessible from the leader, the leader effectively becomes read-only.

To enable a simple synchronous replication test, add the following lines to the `parameters` section of your YAML configuration files:

[1] The data is still there, but recovering it requires a manual recovery effort by data recovery specialists. When Patroni is allowed to rewind with `use_pg_rewind` the forked timeline will be automatically erased to rejoin the failed primary with the cluster.

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

When using PostgreSQL synchronous replication, use at least three Postgres data nodes to ensure write availability if one host fails.

Using PostgreSQL synchronous replication does not guarantee zero lost transactions under all circumstances. When the primary and the secondary that is currently acting as a synchronous replica fail simultaneously a third node that might not contain all transactions will be promoted.

5.3 Synchronous mode

For use cases where losing committed transactions is not permissible you can turn on Patroni's `synchronous_mode`. When `synchronous_mode` is turned on Patroni will not promote a standby unless it is certain that the standby contains all transactions that may have returned a successful commit status to client[2]. This means that the system may be unavailable for writes even though some servers are available. System administrators can still use manual failover commands to promote a standby even if it results in transaction loss.

Turning on `synchronous_mode` does not guarantee multi node durability of commits under all circumstances. When no suitable standby is available, primary server will still accept writes, but does not guarantee their replication. When the primary fails in this mode no standby will be promoted. When the host that used to be the primary comes back it will get promoted automatically, unless system administrator performed a manual failover. This behavior makes synchronous mode usable with 2 node clusters.

When `synchronous_mode` is on and a standby crashes, commits will block until next iteration of Patroni runs and switches the primary to standalone mode (worst case delay for writes `ttl` seconds, average case `loop_wait`/2 seconds). Manually shutting down or restarting a standby will not cause a commit service interruption. Standby will signal the primary to release itself from synchronous standby duties before PostgreSQL shutdown is initiated.

When it is absolutely necessary to guarantee that each write is stored durably on at least two nodes, enable `synchronous_mode_strict` in addition to the `synchronous_mode`. This parameter prevents Patroni from switching off the synchronous replication on the primary when no synchronous standby candidates are available. As a downside, the primary is not be available for writes (unless the Postgres transaction explicitly turns off `synchronous_mode`), blocking all client write requests until at least one synchronous replica comes up.

You can ensure that a standby never becomes the synchronous standby by setting `nosync` tag to true. This is recommended to set for standbys that are behind slow network connections and would cause performance degradation when becoming a synchronous standby.

Synchronous mode can be switched on and off via Patroni REST interface. See *dynamic configuration* for instructions.

Note: Because of the way synchronous replication is implemented in PostgreSQL it is still possible to lose transactions even when using `synchronous_mode_strict`. If the PostgreSQL backend is cancelled while waiting to acknowledge replication (as a result of packet cancellation due to client timeout or backend failure) transaction changes become visible for other backends. Such changes are not yet replicated and may be lost in case of standby promotion.

[2] Clients can change the behavior per transaction using PostgreSQL's `synchronous_commit` setting. Transactions with `synchronous_commit` values of `off` and `local` may be lost on fail over, but will not be blocked by replication delays.

5.4 Synchronous Replication Factor

The parameter `synchronous_node_count` is used by Patroni to manage number of synchronous standby databases. It is set to 1 by default. It has no effect when `synchronous_mode` is set to off. When enabled, Patroni manages precise number of synchronous standby databases based on parameter `synchronous_node_count` and adjusts the state in DCS & synchronous_standby_names as members join and leave.

5.5 Synchronous mode implementation

When in synchronous mode Patroni maintains synchronization state in the DCS, containing the latest primary and current synchronous standby databases. This state is updated with strict ordering constraints to ensure the following invariants:

- A node must be marked as the latest leader whenever it can accept write transactions. Patroni crashing or PostgreSQL not shutting down can cause violations of this invariant.
- A node must be set as the synchronous standby in PostgreSQL as long as it is published as the synchronous standby.
- A node that is not the leader or current synchronous standby is not allowed to promote itself automatically.

Patroni will only assign one or more synchronous standby nodes based on `synchronous_node_count` parameter to `synchronous_standby_names`.

On each HA loop iteration Patroni re-evaluates synchronous standby nodes choice. If the current list of synchronous standby nodes are connected and has not requested its synchronous status to be removed it remains picked. Otherwise the cluster member available for sync that is furthest ahead in replication is picked.

CHAPTER

SIX

WATCHDOG SUPPORT

Having multiple PostgreSQL servers running as primary can result in transactions lost due to diverging timelines. This situation is also called a split-brain problem. To avoid split-brain Patroni needs to ensure PostgreSQL will not accept any transaction commits after leader key expires in the DCS. Under normal circumstances Patroni will try to achieve this by stopping PostgreSQL when leader lock update fails for any reason. However, this may fail to happen due to various reasons:

- Patroni has crashed due to a bug, out-of-memory condition or by being accidentally killed by a system administrator.
- Shutting down PostgreSQL is too slow.
- Patroni does not get to run due to high load on the system, the VM being paused by the hypervisor, or other infrastructure issues.

To guarantee correct behavior under these conditions Patroni supports watchdog devices. Watchdog devices are software or hardware mechanisms that will reset the whole system when they do not get a keepalive heartbeat within a specified timeframe. This adds an additional layer of fail safe in case usual Patroni split-brain protection mechanisms fail.

Patroni will try to activate the watchdog before promoting PostgreSQL to primary. If watchdog activation fails and watchdog mode is `required` then the node will refuse to become leader. When deciding to participate in leader election Patroni will also check that watchdog configuration will allow it to become leader at all. After demoting PostgreSQL (for example due to a manual failover) Patroni will disable the watchdog again. Watchdog will also be disabled while Patroni is in paused state.

By default Patroni will set up the watchdog to expire 5 seconds before TTL expires. With the default setup of `loop_wait=10` and `ttl=30` this gives HA loop at least 15 seconds (`ttl - safety_margin - loop_wait`) to complete before the system gets forcefully reset. By default accessing DCS is configured to time out after 10 seconds. This means that when DCS is unavailable, for example due to network issues, Patroni and PostgreSQL will have at least 5 seconds (`ttl - safety_margin - loop_wait - retry_timeout`) to come to a state where all client connections are terminated.

Safety margin is the amount of time that Patroni reserves for time between leader key update and watchdog keepalive. Patroni will try to send a keepalive immediately after confirmation of leader key update. If Patroni process is suspended for extended amount of time at exactly the right moment the keepalive may be delayed for more than the safety margin without triggering the watchdog. This results in a window of time where watchdog will not trigger before leader key expiration, invalidating the guarantee. To be absolutely sure that watchdog will trigger under all circumstances set up the watchdog to expire after half of TTL by setting `safety_margin` to -1 to set watchdog timeout to `ttl // 2`. If you need this guarantee you probably should increase `ttl` and/or reduce `loop_wait` and `retry_timeout`.

Currently watchdogs are only supported using Linux watchdog device interface.

6.1 Setting up software watchdog on Linux

Default Patroni configuration will try to use `/dev/watchdog` on Linux if it is accessible to Patroni. For most use cases using software watchdog built into the Linux kernel is secure enough.

To enable software watchdog issue the following commands as root before starting Patroni:

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

For testing it may be helpful to disable rebooting by adding `soft_noboot=1` to the modprobe command line. In this case the watchdog will just log a line in kernel ring buffer, visible via *dmesg*.

Patroni will log information about the watchdog when it is successfully enabled.

CHAPTER

SEVEN

PAUSE/RESUME MODE FOR THE CLUSTER

7.1 The goal

Under certain circumstances Patroni needs to temporarily step down from managing the cluster, while still retaining the cluster state in DCS. Possible use cases are uncommon activities on the cluster, such as major version upgrades or corruption recovery. During those activities nodes are often started and stopped for reasons unknown to Patroni, some nodes can be even temporarily promoted, violating the assumption of running only one primary. Therefore, Patroni needs to be able to "detach" from the running cluster, implementing an equivalent of the maintenance mode in Pacemaker.

7.2 The implementation

When Patroni runs in a paused mode, it does not change the state of PostgreSQL, except for the following cases:

- For each node, the member key in DCS is updated with the current information about the cluster. This causes Patroni to run read-only queries on a member node if the member is running.
- For the Postgres primary with the leader lock Patroni updates the lock. If the node with the leader lock stops being the primary (i.e. is demoted manually), Patroni will release the lock instead of promoting the node back.
- Manual unscheduled restart, reinitialize and manual failover are allowed. Manual failover is only allowed if the node to failover to is specified. In the paused mode, manual failover does not require a running primary node.
- If 'parallel' primaries are detected by Patroni, it emits a warning, but does not demote the primary without the leader lock.
- If there is no leader lock in the cluster, the running primary acquires the lock. If there is more than one primary node, then the first primary to acquire the lock wins. If there are no primary altogether, Patroni does not try to promote any replicas. There is an exception in this rule: if there is no leader lock because the old primary has demoted itself due to the manual promotion, then only the candidate node mentioned in the promotion request may take the leader lock. When the new leader lock is granted (i.e. after promoting a replica manually), Patroni makes sure the replicas that were streaming from the previous leader will switch to the new one.
- When Postgres is stopped, Patroni does not try to start it. When Patroni is stopped, it does not try to stop the Postgres instance it is managing.
- Patroni will not try to remove replication slots that don't represent the other cluster member or are not listed in the configuration of the permanent slots.

7.3 User guide

`patronictl` supports `pause` and `resume` commands.

One can also issue a `PATCH` request to the `{namespace}/{cluster}/config` key with `{"pause": true/false/null}`

CHAPTER

EIGHT

DCS FAILSAFE MODE

8.1 The problem

Patroni is heavily relying on Distributed Configuration Store (DCS) to solve the task of leader elections and detect network partitioning. That is, the node is allowed to run Postgres as the primary only if it can update the leader lock in DCS. In case the update of the leader lock fails, Postgres is immediately demoted and started as read-only. Depending on which DCS is used, the chances of hitting the "problem" differ. For example, with Etcd which is only used for Patroni, chances are close to zero, while with K8s API (backed by Etcd) it could be observed more frequently.

8.2 Reasons for the current implementation

The leader lock update failure could be caused by two main reasons:

1. Network partitioning
2. DCS being down

In general, it is impossible to distinguish between these two from a single node, and therefore Patroni assumes the worst case - network partitioning. In the case of a partitioned network, other nodes of the Patroni cluster may successfully grab the leader lock and promote Postgres to primary. In order to avoid a split-brain, the old primary is demoted before the leader lock expires.

8.3 DCS Failsafe Mode

We introduce a new special option, the `failsafe_mode`. It could be enabled only via global *dynamic configuration* stored in the DCS `/config` key. If the failsafe mode is enabled and the leader lock update in DCS failed due to reasons different from the version/value/index mismatch, Postgres may continue to run as a primary if it can access all known members of the cluster via Patroni REST API.

8.4 Low-level implementation details

- We introduce a new, permanent key in DCS, named `/failsafe`.
- The `/failsafe` key contains all known members of the given Patroni cluster at a given time.
- The current leader maintains the `/failsafe` key.
- The member is allowed to participate in the leader race and become the new leader only if it is present in the `/failsafe` key.
- If the cluster consists of a single node the `/failsafe` key will contain a single member.
- In the case of DCS "outage" the existing primary connects to all members presented in the `/failsafe` key via the `POST /failsafe` REST API and may continue to run as the primary if all replicas acknowledge it.
- If one of the members doesn't respond, the primary is demoted.
- Replicas are using incoming `POST /failsafe` REST API requests as an indicator that the primary is still alive. This information is cached for `ttl` seconds.

8.5 F.A.Q.

- Why MUST the current primary see ALL other members? Can't we rely on quorum here?

  This is a great question! The problem is that the view on the quorum might be different from the perspective of DCS and Patroni. While DCS nodes must be evenly distributed across availability zones, there is no such rule for Patroni, and more importantly, there is no mechanism for introducing and enforcing such a rule. If the majority of Patroni nodes ends up in the losing part of the partitioned network (including primary) while minority nodes are in the winning part, the primary must be demoted. Only checking ALL other members allows detecting such a situation.

- What if node/pod gets terminated while DCS is down?

  If DCS isn't accessible, the check "are ALL other cluster members accessible?" is executed every cycle of the heartbeat loop (every `loop_wait` seconds). If pod/node is terminated, the check will fail and Postgres will be demoted to a read-only and will not recover until DCS is restored.

- What if all members of the Patroni cluster are lost while DCS is down?

  Patroni could be configured to create the new replica from the backup even when the cluster doesn't have a leader. But, if the new member isn't present in the `/failsafe` key, it will not be able to grab the leader lock and promote.

- What will happen if the primary lost access to DCS while replicas didn't?

  The primary will execute the failsafe code and contact all known replicas. These replicas will use this information as an indicator that the primary is alive and will not start the leader race even if the leader lock in DCS has expired.

- How to enable the Failsafe Mode?

  Before enabling the `failsafe_mode` please make sure that Patroni version on all members is up-to-date. After that, you can use either the `PATCH /config` *REST API* or `patronictl edit-config -s failsafe_mode=true`

CHAPTER

NINE

USING PATRONI WITH KUBERNETES

Patroni can use Kubernetes objects in order to store the state of the cluster and manage the leader key. That makes it capable of operating Postgres in Kubernetes environment without any consistency store, namely, one doesn't need to run an extra Etcd deployment. There are two different type of Kubernetes objects Patroni can use to store the leader and the configuration keys, they are configured with the *kubernetes.use_endpoints* or *PATRONI_KUBERNETES_USE_ENDPOINTS* environment variable.

9.1 Use Endpoints

Despite the fact that this is the recommended mode, it is turned off by default for compatibility reasons. When it is on, Patroni stores the cluster configuration and the leader key in the *metadata: annotations* fields of the respective *Endpoints* it creates. Changing the leader is safer than when using *ConfigMaps*, since both the annotations, containing the leader information, and the actual addresses pointing to the running leader pod are updated simultaneously in one go.

9.2 Use ConfigMaps

In this mode, Patroni will create ConfigMaps instead of Endpoints and store keys inside meta-data of those ConfigMaps. Changing the leader takes at least two updates, one to the leader ConfigMap and another to the respective Endpoint.

To direct the traffic to the Postgres leader you need to configure the Kubernetes Postgres service to use the label selector with the *role_label* (configured in patroni configuration).

Note that in some cases, for instance, when running on OpenShift, there is no alternative to using ConfigMaps.

9.3 Configuration

Patroni Kubernetes *settings* and *environment variables* are described in the general chapters of the documentation.

9.3.1 Customize role label

By default, Patroni will set corresponding labels on the pod it runs in based on node's role, such as `role=master`. The key and value of label can be customized by *kubernetes.role_label*, *kubernetes.leader_label_value*, *kubernetes.follower_label_value* and *kubernetes.standby_leader_label_value*.

Note that if you migrate from default role labels to custom ones, you can reduce downtime by following migration steps:

1. Add a temporary label using original role value for the pod with *kubernetes.tmp_role_label* (like `tmp_role`). Once pods are restarted they will get following labels set by Patroni:

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

2. After all pods have been updated, modify the service selector to select the temporary label.

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

3. Add your custom role label (e.g., set *kubernetes.leader_label_value=primary*). Once pods are restarted they will get following new labels set by Patroni:

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

4. After all pods have been updated again, modify the service selector to use new role value.

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

5. Finally, remove the temporary label from your configuration and update all pods.

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

9.4 Examples

- The kubernetes folder of the Patroni repository contains examples of the Docker image, and the Kubernetes manifest to test Patroni Kubernetes setup. Note that in the current state it will not be able to use PersistentVolumes because of permission issues.
- You can find the full-featured Docker image that can use Persistent Volumes in the Spilo Project.
- There is also a Helm chart to deploy the Spilo image configured with Patroni running using Kubernetes.
- In order to run your database clusters at scale using Patroni and Spilo, take a look at the postgres-operator project. It implements the operator pattern to manage Spilo clusters.

CHAPTER

TEN

CITUS SUPPORT

Patroni makes it extremely simple to deploy Multi-Node Citus clusters.

10.1 TL;DR

There are only a few simple rules you need to follow:

1. Citus database extension to PostgreSQL must be available on all nodes. Absolute minimum supported Citus version is 10.0, but, to take all benefits from transparent switchovers and restarts of workers we recommend using at least Citus 11.2.
2. Cluster name (`scope`) must be the same for all Citus nodes!
3. Superuser credentials must be the same on coordinator and all worker nodes, and `pg_hba.conf` should allow superuser access between all nodes.
4. *REST API* access should be allowed from worker nodes to the coordinator. E.g., credentials should be the same and if configured, client certificates from worker nodes must be accepted by the coordinator.
5. Add the following section to the `patroni.yaml`:

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

After that you just need to start Patroni and it will handle the rest:

1. `citus` extension will be automatically added to `shared_preload_libraries`.
2. If `max_prepared_transactions` isn't explicitly set in the global *dynamic configuration* Patroni will automatically set it to `2*max_connections`.
3. The `citus.database` will be automatically created followed by `CREATE EXTENSION citus`.
4. Current superuser *credentials* will be added to the `pg_dist_authinfo` table to allow cross-node communication. Don't forget to update them if later you decide to change superuser username/password/sslcert/sslkey!
5. The coordinator primary node will automatically discover worker primary nodes and add them to the `pg_dist_node` table using the `citus_add_node()` function.
6. Patroni will also maintain `pg_dist_node` in case failover/switchover on the coordinator or worker clusters occurs.

10.2 patronictl

Coordinator and worker clusters are physically different PostgreSQL/Patroni clusters that are just logically groupped together using the Citus database extension to PostgreSQL. Therefore in most cases it is not possible to manage them as a single entity.

It results in two major differences in `patronictl` behaviour when `patroni.yaml` has the `citus` section comparing with the usual:

1. The `list` and the `topology` by default output all members of the Citus formation (coordinators and workers). The new column `Group` indicates which Citus group they belong to.
2. For all `patronictl` commands the new option is introduced, named `--group`. For some commands the default value for the group might be taken from the `patroni.yaml`. For example, `patronictl pause` will enable the maintenance mode by default for the `group` that is set in the `citus` section, but for example for `patronictl switchover` or `patronictl remove` the group must be explicitly specified.

An example of `patronictl list` output for the Citus cluster:

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

If we add the `--group` option, the output will change to:

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

postgres@coord1:~$ patronictl list demo --group 1
+ Citus cluster: demo (group: 1, 7179854923881963547) -----------+
| Member  | Host       | Role         | State   | TL | Lag in MB |
+---------+------------+--------------+---------+----+-----------+
| work1-1 | 172.27.0.8 | Sync Standby | running |  1 |         0 |
| work1-2 | 172.27.0.2 | Leader       | running |  1 |           |
+---------+------------+--------------+---------+----+-----------+
```

10.3 Citus worker switchover

When a switchover is orchestrated for a Citus worker node, Citus offers the opportunity to make the switchover close to transparent for an application. Because the application connects to the coordinator, which in turn connects to the worker nodes, then it is possible with Citus to *pause* the SQL traffic on the coordinator for the shards hosted on a worker node. The switchover then happens while the traffic is kept on the coordinator, and resumes as soon as a new primary worker node is ready to accept read-write queries.

An example of `patronictl switchover` on the worker cluster:

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

(continues on next page)

(continued from previous page)

```
|     2 | work2-2 | 172.27.0.7  | Sync Standby | running |  2 |         0 |
+-------+---------+-------------+--------------+---------+----+-----------+
```

And this is how it looks on the coordinator side:

```
# The worker primary notifies the coordinator that it is going to execute "pg_ctl stop".
2022-12-22 07:02:38,636 DEBUG: query("BEGIN")
2022-12-22 07:02:38,636 DEBUG: query("SELECT pg_catalog.citus_update_node(3, '172.27.0.7-
↪demoted', 5432, true, 10000)")
# From this moment all application traffic on the coordinator to the worker group 2 is␣
↪paused.

# The future worker primary notifies the coordinator that it acquired the leader lock in␣
↪DCS and about to run "pg_ctl promote".
2022-12-22 07:02:40,085 DEBUG: query("SELECT pg_catalog.citus_update_node(3, '172.27.0.5
↪', 5432)")

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

10.4 Peek into DCS

The Citus cluster (coordinator and workers) are stored in DCS as a fleet of Patroni clusters logically grouped together:

```
/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
...
```

Such an approach was chosen because for most DCS it becomes possible to fetch the entire Citus cluster with a single recursive read request. Only Citus coordinator nodes are reading the whole tree, because they have to discover worker nodes. Worker nodes are reading only the subtree for their own group and in some cases they could read the subtree of the coordinator group.

10.5 Citus on Kubernetes

Since Kubernetes doesn't support hierarchical structures we had to include the citus group to all K8s objects Patroni creates:

```
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
...
```

I.e., the naming pattern is: `${scope}-${citus.group}-${type}`.

All Kubernetes objects are discovered by Patroni using the label selector, therefore all Pods with Patroni&Citus and Endpoints/ConfigMaps must have similar labels, and Patroni must be configured to use them using Kubernetes *settings* or *environment variables*.

A couple of examples of Patroni configuration using Pods environment variables:

1. for the coordinator cluster

```
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}'
```

(continues on next page)

(continued from previous page)

```
    - name: PATRONI_CITUS_DATABASE
      value: citus
    - name: PATRONI_CITUS_GROUP
      value: "0"
```

2. for the worker cluster from the group 2

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

As you may noticed, both examples have `citus-group` label set. This label allows Patroni to identify object as belonging to a certain Citus group. In addition to that, there is also `PATRONI_CITUS_GROUP` environment variable, which has the same value as the `citus-group` label. When Patroni creates new Kubernetes objects ConfigMaps or Endpoints, it automatically puts the `citus-group: ${env.PATRONI_CITUS_GROUP}` label on them:

```
apiVersion: v1
kind: ConfigMap
metadata:
```

(continues on next page)

(continued from previous page)

```
  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}
```

You can find a complete example of Patroni deployment on Kubernetes with Citus support in the kubernetes folder of the Patroni repository.

There are two important files for you:

1. Dockerfile.citus
2. citus_k8s.yaml

10.6 Citus upgrades and PostgreSQL major upgrades

First, please read about upgrading Citus version in the documentation. There is one minor change in the process. When executing upgrade, you have to use `patronictl restart` instead of `systemctl restart` to restart PostgreSQL.

The PostgreSQL major upgrade with Citus is a bit more complex. You will have to combine techniques used in the Citus documentation about major upgrades and Patroni documentation about *PostgreSQL major upgrade*. Please keep in mind that Citus cluster consists of many Patroni clusters (coordinator and workers) and they all have to be upgraded independently.

CHAPTER

ELEVEN

CONVERT A STANDALONE TO A PATRONI CLUSTER

This section describes the process for converting a standalone PostgreSQL instance into a Patroni cluster.

To deploy a Patroni cluster without using a pre-existing PostgreSQL instance, see *Running and Configuring* instead.

11.1 Procedure

You can find below an overview of steps for converting an existing Postgres cluster to a Patroni managed cluster. In the steps we assume all nodes that are part of the existing cluster are currently up and running, and that you *do not* intend to change Postgres configuration while the migration is ongoing. The steps:

1. Create the Postgres users as explained for *authentication* section of the Patroni configuration. You can find sample SQL commands to create the users in the code block below, in which you need to replace the usernames and passwords as per your environment. If you already have the relevant users, then you can skip this step.

```
-- 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_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. Perform the following steps on all Postgres nodes. Perform all steps on one node before proceeding with the next node. Start with the primary node, then proceed with each standby node:

1. If you are running Postgres through systemd, then disable the Postgres systemd unit. This is performed as Patroni manages starting and stopping the Postgres daemon.
2. Create a YAML configuration file for Patroni.
   - **Note (specific for the primary node):** If you have replication slots being used for replication between cluster members, then it is recommended that you enable `use_slots` and configure the existing replication slots as permanent via the `slots` configuration item. Be aware that Patroni automatically creates replication slots for replication between members, and drops replication slots that it does not recognize, when `use_slots` is enabled. The idea of using permanent slots here is to allow your existing slots to persist while the migration to Patroni is in progress. See *YAML Configuration Settings* for details.
3. Start Patroni using the `patroni` systemd service unit. It automatically detects that Postgres is already running and starts monitoring the instance.

3. Hand over Postgres "start up procedure" to Patroni. In order to do that you need to restart the cluster members through `patronictl restart cluster-name member-name` command. For minimal downtime you might want to split this step into:
   1. Immediate restart of the standby nodes.
   2. Scheduled restart of the primary node within a maintenance window.
4. If you configured permanent slots in step `1.2.`, then you should remove them from `slots` configuration through `patronictl edit-config cluster-name member-name` command once the `restart_lsn` of the slots created by Patroni is able to catch up with the `restart_lsn` of the original slots for the corresponding members. By removing the slots from `slots` configuration you will allow Patroni to drop the original slots from your cluster once they are not needed anymore. You can find below an example query to check the `restart_lsn` of a couple slots, so you can compare them:

```
-- 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 (
    'original_slot_for_member_x',
    'slot_for_member_x'
)
```

CHAPTER

TWELVE

MAJOR UPGRADE OF POSTGRESQL VERSION

The only possible way to do a major upgrade currently is:

1. Stop Patroni
2. Upgrade PostgreSQL binaries and perform pg_upgrade on the primary node
3. Update patroni.yml
4. Remove the initialize key from DCS or wipe complete cluster state from DCS. The second one could be achieved by running `patronictl remove <cluster-name>`. It is necessary because pg_upgrade runs initdb which actually creates a new database with a new PostgreSQL system identifier.
5. If you wiped the cluster state in the previous step, you may wish to copy patroni.dynamic.json from old data dir to the new one. It will help you to retain some PostgreSQL parameters you had set before.
6. Start Patroni on the primary node.
7. Upgrade PostgreSQL binaries, update patroni.yml and wipe the data_dir on standby nodes.
8. Start Patroni on the standby nodes and wait for the replication to complete.

Running pg_upgrade on standby nodes is not supported by PostgreSQL. If you know what you are doing, you can try the rsync procedure described in https://www.postgresql.org/docs/current/pgupgrade.html instead of wiping data_dir on standby nodes. The safest way is however to let Patroni replicate the data for you.

12.1 FAQ

- During Patroni startup, Patroni complains that it cannot bind to the PostgreSQL port.

  You need to verify `listen_addresses` and `port` in `postgresql.conf` and `postgresql.listen` in `patroni.yml`. Don't forget that `pg_hba.conf` should allow such access.

- After asking Patroni to restart the node, PostgreSQL displays the error message `could not open configuration file "/etc/postgresql/10/main/pg_hba.conf": No such file or directory`

  It can mean various things depending on how you manage PostgreSQL configuration. If you specified *postgresql.config_dir*, Patroni generates the `pg_hba.conf` based on the settings in the *bootstrap* section only when it bootstraps a new cluster. In this scenario the `PGDATA` was not empty, therefore no bootstrap happened. This file must exist beforehand.

CHAPTER

THIRTEEN

SECURITY CONSIDERATIONS

A Patroni cluster has two interfaces to be protected from unauthorized access: the distributed configuration storage (DCS) and the Patroni REST API.

13.1 Protecting DCS

Patroni and patronictl both store and retrieve data to/from the DCS.

Despite DCS doesn't contain any sensitive information, it allows changing some of Patroni/Postgres configuration. Therefore the very first thing that should be protected is DCS itself.

The details of protection depend on the type of DCS used. The authentication and encryption parameters (tokens/basic-auth/client certificates) for the supported types of DCS are covered in *settings*.

The general recommendation is to enable TLS for all DCS communication.

13.2 Protecting the REST API

Protecting the REST API is a more complicated task.

The Patroni REST API is used by Patroni itself during the leader race, by the `patronictl` tool in order to perform failovers/switchovers/reinitialize/restarts/reloads, by HAProxy or any other kind of load balancer to perform HTTP health checks, and of course could also be used for monitoring.

From the point of view of security, REST API contains safe (`GET` requests, only retrieve information) and unsafe (`PUT`, `POST`, `PATCH` and `DELETE` requests, change the state of nodes) endpoints.

The unsafe endpoints can be protected with HTTP basic-auth by setting the `restapi.authentication.username` and `restapi.authentication.password` parameters. There is no way to protect the safe endpoints without enabling TLS.

When TLS for the REST API is enabled and a PKI is established, mutual authentication of the API server and API client is possible for all endpoints.

The `restapi` section parameters enable TLS client authentication to the server. Depending on the value of the `verify_client` parameter, the API server requires a successful client certificate verification for both safe and unsafe API calls (`verify_client: required`), or only for unsafe API calls (`verify_client: optional`), or for no API calls (`verify_client: none`).

The `ctl` section parameters enable TLS server authentication to the client (the `patronictl` tool which uses the same config as patroni). Set `insecure: true` to disable the server certificate verification by the client. See *settings* for a detailed description of the TLS client parameters.

Protecting the PostgreSQL database proper from unauthorized access is beyond the scope of this document and is covered in https://www.postgresql.org/docs/current/client-authentication.html

CHAPTER

FOURTEEN

HA MULTI DATACENTER

The high availability of a PostgreSQL cluster deployed in multiple data centers is based on replication, which can be synchronous or asynchronous (replication_modes).

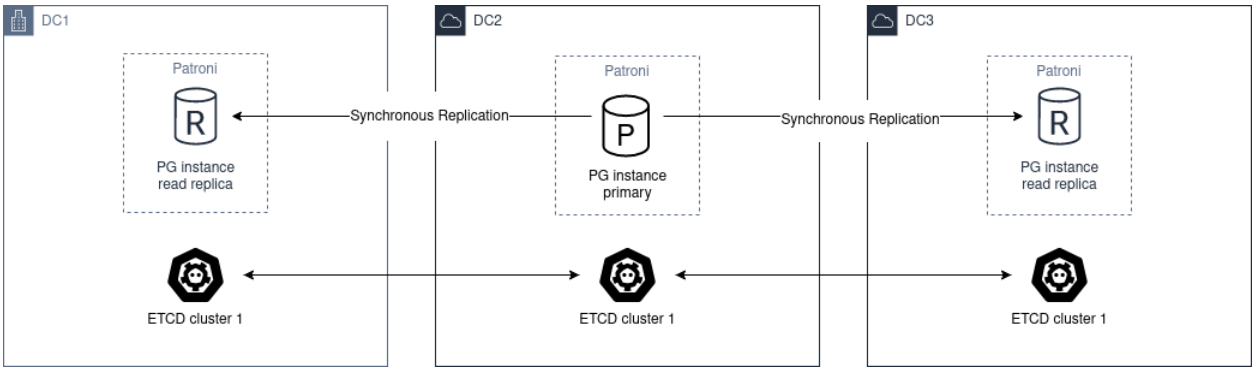
In both cases, it is important to be clear about the following concepts:

- Postgres can run as primary or standby leader only when it owns the leading key and can update the leading key.
- You should run the odd number of etcd, ZooKeeper or Consul nodes: 3 or 5!

14.1 Synchronous Replication

To have a multi DC cluster that can automatically tolerate a zone drop, a minimum of 3 is required.

The architecture diagram would be the following:



We must deploy a cluster of etcd, ZooKeeper or Consul through the different DC, with a minimum of 3 nodes, one in each zone.

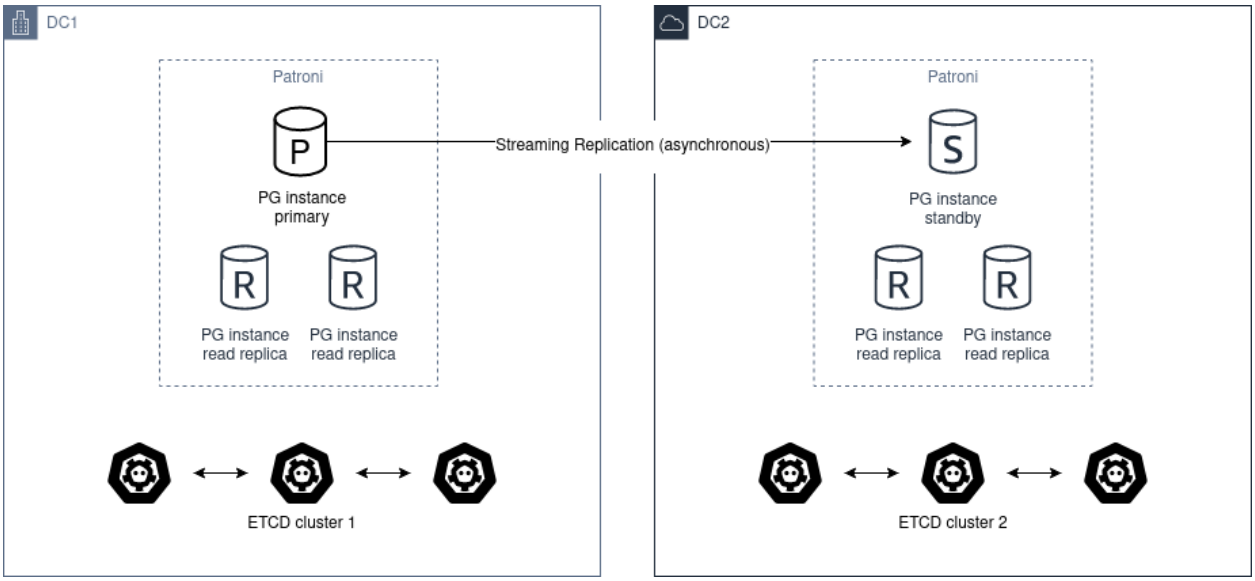
Regarding postgres, we must deploy at least 2 nodes, in different DC. Then you have to set `synchronous_mode: true` in the global *dynamic configuration*.

This enables sync replication and the primary node will choose one of the nodes as synchronous.

14.2 Asynchronous Replication

With only two data centers it would be better to have two independent etcd clusters and run Patroni *standby cluster* in the second data center. If the first site is down, you can MANUALLY promote the `standby_cluster`.

The architecture diagram would be the following:



Automatic promotion is not possible, because DC2 will never able to figure out the state of DC1.

You should not use `pg_ctl promote` in this scenario, you need "manually promote" the healthy cluster by removing `standby_cluster` section from the *dynamic configuration*.

Warning: If the source cluster is still up and running and you promote the standby cluster you create a split-brain.

In case you want to return to the "initial" state, there are only two ways of resolving it:

- Add the standby_cluster section back and it will trigger pg_rewind, but there are chances that pg_rewind will fail.
- Rebuild the standby cluster from scratch.

Before promoting standby cluster one have to manually ensure that the source cluster is down (STONITH). When DC1 recovers, the cluster has to be converted to a standby cluster.

Before doing that you may manually examine the database and extract all changes that happened between the time when network between DC1 and DC2 has stopped working and the time when you manually stopped the cluster in DC1.

Once extracted, you may also manually apply these changes to the cluster in DC2.

CHAPTER

FIFTEEN

RELEASE NOTES

15.1 Version 3.1.0

Breaking changes

- Changed semantic of `restapi.keyfile` and `restapi.certfile` (Alexander Kukushkin)

  Previously Patroni was using `restapi.keyfile` and `restapi.certfile` as client certificates as a fallback if there were no respective configuration parameters in the `ctl` section.

> **Warning:** If you enabled client certificates validation (`restapi.verify_client` is set to `required`), you also **must** provide **valid client certificates** in the `ctl.certfile`, `ctl.keyfile`, `ctl.keyfile_password`. If not provided, Patroni will not work correctly.

New features

- Make Pod role label configurable (Waynerv)

  Values could be customized using `kubernetes.leader_label_value`, `kubernetes.follower_label_value` and `kubernetes.standby_leader_label_value` parameters. This feature will be very useful when we change the `master` role to the `primary`. You can read more about the feature and migration steps *here*.

Improvements

- Various improvements of `patroni --validate-config` (Alexander Kukushkin)

  Improved parameter validation for different DCS, `bootstrap.dcs` , `ctl`, `restapi`, and `watchdog` sections.

- Start Postgres not in recovery if it crashed during recovery while Patroni is running (Alexander Kukushkin)

  It may reduce recovery time and will help to prevent unnecessary timeline increments.

- Avoid unnecessary updates of `/status` key (Alexander Kukushkin)

  When there are no permanent logical slots Patroni was updating the `/status` on every heartbeat loop even when LSN on the primary didn't move forward.

- Don't allow stale primary to win the leader race (Alexander Kukushkin)

  If Patroni was hanging during a significant time due to lack of resources it will additionally check that no other nodes promoted Postgres before acquiring the leader lock.

- Implemented visibility of certain PostgreSQL parameters validation (Alexander Kukushkin, Feike Steenbergen)

If validation of `max_connections`, `max_wal_senders`, `max_prepared_transactions`, `max_locks_per_transaction`, `max_replication_slots`, or `max_worker_processes` failed Patroni was using some sane default value. Now in addition to that it will also show a warning.

- Set permissions for files and directories created in `PGDATA` (Alexander Kukushkin)

  All files created by Patroni had only owner read/write permissions. This behaviour was breaking backup tools that run under a different user and relying on group read permissions. Now Patroni honors permissions on `PGDATA` and correctly sets permissions on all directories and files it creates inside `PGDATA`.

Bugfixes

- Run `archive_command` through shell (Waynerv)

  Patroni might archive some WAL segments before doing crash recovery in a single-user mode or before `pg_rewind`. If the archive_command contains some shell operators, like && it didn't work with Patroni.

- Fixed "on switchover" shutdown checks (Polina Bungina)

  It was possible that specified candidate is still streaming and didn't received shut down checking but the leader key was removed because some other nodes were healthy.

- Fixed "is primary" check (Alexander Kukushkin)

  During the leader race replicas were not able to recognize that Postgres on the old leader is still running as a primary.

- Fixed `patronictl list` (Alexander Kukushkin)

  The Cluster name field was missing in `tsv`, `json`, and `yaml` output formats.

- Fixed `pg_rewind` behaviour after pause (Alexander Kukushkin)

  Under certain conditions, Patroni wasn't able to join the false primary back to the cluster with `pg_rewind` after coming out of maintenance mode.

- Fixed bug in Etcd v3 implementation (Alexander Kukushkin)

  Invalidate internal KV cache if key update performed using `create_revision`/`mod_revision` field due to revision mismatch.

- Fixed behaviour of replicas in standby cluster in pause (Alexander Kukushkin)

  When the leader key expires replicas in standby cluster will not follow the remote node but keep `primary_conninfo` as it is.

15.2 Version 3.0.4

New features

- Make the replication status of standby nodes visible (Alexander Kukushkin)

  For PostgreSQL 9.6+ Patroni will report the replication state as `streaming` when the standby is streaming from the other node or `in archive recovery` when there is no replication connection and `restore_command` is set. The state is visible in `member` keys in DCS, in the REST API, and in `patronictl list` output.

Improvements

- Improved error messages with Etcd v3 (Alexander Kukushkin)

  When Etcd v3 cluster isn't accessible Patroni was reporting that it can't access `/v2` endpoints.

- Use quorum read in `patronictl` if it is possible (Alexander Kukushkin)

  Etcd or Consul clusters could be degraded to read-only, but from the `patronictl` view everything was fine. Now it will fail with the error.

- Prevent splitbrain from duplicate names in configuration (Mark Pekala)

  When starting Patroni will check if node with the same name is registered in DCS, and try to query its REST API. If REST API is accessible Patroni exits with an error. It will help to protect from the human error.

- Start Postgres not in recovery if it crashed while Patroni is running (Alexander Kukushkin)

  It may reduce recovery time and will help from unnecessary timeline increments.

Bugfixes

- REST API SSL certificate were not reloaded upon receiving a SIGHUP (Israel Barth Rubio)

  Regression was introduced in 3.0.3.

- Fixed integer GUCs validation for parameters like `max_connections` (Feike Steenbergen)

  Patroni didn't like quoted numeric values. Regression was introduced in 3.0.3.

- Fix issue with `synchronous_mode` (Alexander Kukushkin)

  Execute `txid_current()` with `synchronous_commit=off` so it doesn't accidentally wait for absent synchronous standbys when `synchronous_mode_strict` is enabled.

15.3 Version 3.0.3

New features

- Compatibility with PostgreSQL 16 beta1 (Alexander Kukushkin)

  Extended GUC's validator rules.

- Make PostgreSQL GUC's validator extensible (Israel Barth Rubio)

  Validator rules are loaded from YAML files located in `patroni/postgresql/available_parameters/` directory. Files are ordered in alphabetical order and applied one after another. It makes possible to have custom validators for non-standard Postgres distributions.

- Added `restapi.request_queue_size` option (Andrey Zhidenkov, Aleksei Sukhov)

  Sets request queue size for TCP socket used by Patroni REST API. Once the queue is full, further requests get a "Connection denied" error. The default value is 5.

- Call `initdb` directly when initializing a new cluster (Matt Baker)

  Previously it was called via `pg_ctl`, what required a special quoting of parameters passed to `initdb`.

- Added before stop hook (Le Duane)

  The hook could be configured via `postgresql.before_stop` and is executed right before `pg_ctl stop`. The exit code doesn't impact shutdown process.

- Added support for custom Postgres binary names (Israel Barth Rubio, Polina Bungina)

  When using a custom Postgres distribution it may be the case that the Postgres binaries are compiled with different names other than the ones used by the community Postgres distribution. Custom binary names could be configured using `postgresql.bin_name.*` and `PATRONI_POSTGRESQL_BIN_*` environment variables.

Improvements

- Various improvements of `patroni --validate-config` (Polina Bungina)
  - Make `bootstrap.initdb` optional. It is only required for new clusters, but `patroni --validate-config` was complaining if it was missing in the config.
  - Don't error out when `postgresql.bin_dir` is empty or not set. Try to first find Postgres binaries in the default PATH instead.
  - Make `postgresql.authentication.rewind` section optional. If it is missing, Patroni is using the superuser.
- Improved error reporting in `patronictl` (Israel Barth Rubio)

  The `\n` symbol was rendered as it is, instead of the actual newline symbol.

Bugfixes

- Fixed issue in Citus support (Alexander Kukushkin)

  If the REST API call from the promoted worker to the coordinator failed during switchover it was leaving the given Citus group blocked during indefinite time.
- Allow *etcd3* URL in *–dcs-url* option of *patronictl* (Israel Barth Rubio)

  If users attempted to pass a *etcd3* URL through *–dcs-url* option of *patronictl* they would face an exception.

15.4 Version 3.0.2

Warning: Version 3.0.2 dropped support of Python older than 3.6.

New features

- Added sync standby replica status to `/metrics` endpoint (Thomas von Dein, Alexander Kukushkin)

  Before were only reporting `primary/standby_leader/replica`.
- User-friendly handling of `PAGER` in `patronictl` (Israel Barth Rubio)

  It makes pager configurable via `PAGER` environment variable, which overrides default `less` and `more`.
- Make K8s retriable HTTP status code configurable (Alexander Kukushkin)

  On some managed platforms it is possible to get status code `401 Unauthorized`, which sometimes gets resolved after a few retries.

Improvements

- Set `hot_standby` to `off` during custom bootstrap only if `recovery_target_action` is set to `promote` (Alexander Kukushkin)

  It was necessary to make `recovery_target_action=pause` work correctly.
- Don't allow `on_reload` callback to kill other callbacks (Alexander Kukushkin)

  `on_start/on_stop/on_role_change` are usually used to add/remove Virtual IP and `on_reload` should not interfere with them.
- Switched to `IMDSFetcher` in aws callback example script (Polina Bungina)

  The `IMDSv2` requires a token to work with and the `IMDSFetcher` handles it transparently.

Bugfixes

- Fixed `patronictl switchover` on Citus cluster running on Kubernetes (Lukáš Lalinský)

  It didn't work for namespaces different from `default`.

- Don't write to `PGDATA` if major version is not known (Alexander Kukushkin)

  If right after the start `PGDATA` was empty (maybe wasn't yet mounted), Patroni was making a false assumption about PostgreSQL version and falsely creating `recovery.conf` file even if the actual major version is v10+.

- Fixed bug with Citus metadata after coordinator failover (Alexander Kukushkin)

  The `citus_set_coordinator_host()` call doesn't cause metadata sync and the change was invisible on worker nodes. The issue is solved by switching to `citus_update_node()`.

- Use etcd hosts listed in the config file as a fallback when all etcd nodes "failed" (Alexander Kukushkin)

  The etcd cluster may change topology over time and Patroni tries to follow it. If at some point all nodes became unreachable Patroni will use a combination of nodes from the config plus the last known topology when trying to reconnect.

15.5 Version 3.0.1

Bugfixes

- Pass proper role name to an `on_role_change` callback script'. (Alexander Kukushkin, Polina Bungina)

  Patroni used to erroneously pass `promoted` role to an `on_role_change` callback script on promotion. The passed role name changed back to `master`. This regression was introduced in 3.0.0.

15.6 Version 3.0.0

This version adds integration with Citus and makes it possible to survive temporary DCS outages without demoting primary.

Warning:

- Version 3.0.0 is the last release supporting Python 2.7. Upcoming release will drop support of Python versions older than 3.7.
- The RAFT support is deprecated. We will do our best to maintain it, but take neither guarantee nor responsibility for possible issues.
- This version is the first step in getting rid of the "master", in favor of "primary". Upgrading to the next major release will work reliably only if you run at least 3.0.0.

New features

- DCS failsafe mode (Alexander Kukushkin, Polina Bungina)

  If the feature is enabled it will allow Patroni cluster to survive temporary DCS outages. You can find more details in the *documentation*.

- Citus support (Alexander Kukushkin, Polina Bungina, Jelte Fennema)

  Patroni enables easy deployment and management of Citus clusters with HA. Please check *here* page for more information.

Improvements

- Suppress recurring errors when dropping unknown but active replication slots (Michael Banck)

  Patroni will still write these logs, but only in DEBUG.

- Run only one monitoring query per HA loop (Alexander Kukushkin)

  It wasn't the case if synchronous replication is enabled.

- Keep only latest failed data directory (William Albertus Dembo)

  If bootstrap failed Patroni used to rename $PGDATA folder with timestamp suffix. From now on the suffix will be `.failed` and if such folder exists it is removed before renaming.

- Improved check of synchronous replication connections (Alexander Kukushkin)

  When the new host is added to the `synchronous_standby_names` it will be set as synchronous in DCS only when it managed to catch up with the primary in addition to `pg_stat_replication.sync_state = 'sync'`.

Removed functionality

- Remove `patronictl scaffold` (Alexander Kukushkin)

  The only reason for having it was a hacky way of running standby clusters.

15.7 Version 2.1.7

Bugfixes

- Fixed little incompatibilities with legacy python modules (Alexander Kukushkin)

  They prevented from building/running Patroni on Debian buster/Ubuntu bionic.

15.8 Version 2.1.6

Improvements

- Fix annoying exceptions on ssl socket shutdown (Alexander Kukushkin)

  The HAProxy is closing connections as soon as it got the HTTP Status code leaving no time for Patroni to properly shutdown SSL connection.

- Adjust example Dockerfile for arm64 (Polina Bungina)

  Remove explicit `amd64` and `x86_64`, don't remove `libnss_files.so.*`.

Security improvements

- Enforce `search_path=pg_catalog` for non-replication connections (Alexander Kukushkin)

  Since Patroni is heavily relying on superuser connections, we want to protect it from the possible attacks carried out using user-defined functions and/or operators in `public` schema with the same name and signature as the corresponding objects in `pg_catalog`. For that, `search_path=pg_catalog` is enforced for all connections created by Patroni (except replication connections).

- Prevent passwords from being recorded in `pg_stat_statements` (Feike Steenbergen)

  It is achieved by setting `pg_stat_statements.track_utility=off` when creating users.

Bugfixes

- Declare `proxy_address` as optional (Denis Laxalde)

  As it is effectively a non-required option.

- Improve behaviour of the insecure option (Alexander Kukushkin)

  Ctl's `insecure` option didn't work properly when client certificates were used for REST API requests.

- Take watchdog configuration from `bootstrap.dcs` when the new cluster is bootstrapped (Matt Baker)

  Patroni used to initially configure watchdog with defaults when bootstrapping a new cluster rather than taking configuration used to bootstrap the DCS.

- Fix the way file extensions are treated while finding executables in WIN32 (Martín Marqués)

  Only add `.exe` to a file name if it has no extension yet.

- Fix Consul TTL setup (Alexander Kukushkin)

  We used `ttl/2.0` when setting the value on the HTTPClient, but forgot to multiply the current value by 2 in the class' property. It was resulting in Consul TTL off by twice.

Removed functionality

- Remove `patronictl configure` (Polina Bungina)

  There is no more need for a separate `patronictl` config creation.

15.9 Version 2.1.5

This version enhances compatibility with PostgreSQL 15 and declares Etcd v3 support as production ready. The Patroni on Raft remains in Beta.

New features

- Improve `patroni --validate-config` (Denis Laxalde)

  Exit with code 1 if config is invalid and print errors to stderr.

- Don't drop replication slots in pause (Alexander Kukushkin)

  Patroni is automatically creating/removing physical replication slots when members are joining/leaving the cluster. In pause slots will no longer be removed.

- Support the `HEAD` request method for monitoring endpoints (Robert Cutajar)

  If used instead of `GET` Patroni will return only the HTTP Status Code.

- Support behave tests on Windows (Alexander Kukushkin)

  Emulate graceful Patroni shutdown (`SIGTERM`) on Windows by introduce the new REST API endpoint `POST /sigterm`.

- Introduce `postgresql.proxy_address` (Alexander Kukushkin)

  It will be written to the member key in DCS as the `proxy_url` and could be used/useful for service discovery.

Stability improvements

- Call `pg_replication_slot_advance()` from a thread (Alexander Kukushkin)

  On busy clusters with many logical replication slots the `pg_replication_slot_advance()` call was affecting the main HA loop and could result in the member key expiration.

- Archive possibly missing WALs before calling `pg_rewind` on the old primary (Polina Bungina)

  If the primary crashed and was down during considerable time, some WAL files could be missing from archive and from the new primary. There is a chance that `pg_rewind` could remove these WAL files from the old primary

making it impossible to start it as a standby. By archiving `ready` WAL files we not only mitigate this problem but in general improving continues archiving experience.

- Ignore `403` errors when trying to create Kubernetes Service (Nick Hudson, Polina Bungina)

  Patroni was spamming logs by unsuccessful attempts to create the service, which in fact could already exist.

- Improve liveness probe (Alexander Kukushkin)

  The liveness problem will start failing if the heartbeat loop is running longer than *ttl* on the primary or *2*ttl* on the replica. That will allow us to use it as an alternative for *watchdog* on Kubernetes.

- Make sure only sync node tries to grab the lock when switchover (Alexander Kukushkin, Polina Bungina)

  Previously there was a slim chance that up-to-date async member could become the leader if the manual switchover was performed without specifying the target.

- Avoid cloning while bootstrap is running (Ants Aasma)

  Do not allow a create replica method that does not require a leader to be triggered while the cluster bootstrap is running.

- Compatibility with kazoo-2.9.0 (Alexander Kukushkin)

  Depending on python version the `SequentialThreadingHandler.select()` method may raise `TypeError` and `IOError` exceptions if `select()` is called on the closed socket.

- Explicitly shut down SSL connection before socket shutdown (Alexander Kukushkin)

  Not doing it resulted in `unexpected eof while reading` errors with OpenSSL 3.0.

- Compatibility with *prettytable>=2.2.0* (Alexander Kukushkin)

  Due to the internal API changes the cluster name header was shown on the incorrect line.

Bugfixes

- Handle expired token for Etcd lease_grant (monsterxx03)

  In case of error get the new token and retry request.

- Fix bug in the `GET /read-only-sync` endpoint (Alexander Kukushkin)

  It was introduced in previous release and effectively never worked.

- Handle the case when data dir storage disappeared (Alexander Kukushkin)

  Patroni is periodically checking that the PGDATA is there and not empty, but in case of issues with storage the `os.listdir()` is raising the `OSError` exception, breaking the heart-beat loop.

- Apply `master_stop_timeout` when waiting for user backends to close (Alexander Kukushkin)

  Something that looks like user backend could be in fact a background worker (e.g., Citus Maintenance Daemon) that is failing to stop.

- Accept `*:<port>` for `postgresql.listen` (Denis Laxalde)

  The `patroni --validate-config` was complaining about it being invalid.

- Timeouts fixes in Raft (Alexander Kukushkin)

  When Patroni or patronictl are starting they try to get Raft cluster topology from known members. These calls were made without proper timeouts.

- Forcefully update consul service if token was changed (John A. Lotoski)

  Not doing so results in errors "rpc error making call: rpc error making call: ACL not found".

15.10 Version 2.1.4

New features

- Improve `pg_rewind` behavior on typical Debian/Ubuntu systems (Gunnar "Nick" Bluth)

  On Postgres setups that keep *postgresql.conf* outside of the data directory (e.g. Ubuntu/Debian packages), `pg_rewind --restore-target-wal` fails to figure out the value of the `restore_command`.

- Allow setting `TLSServerName` on Consul service checks (Michael Gmelin)

  Useful when checks are performed by IP and the Consul `node_name` is not a FQDN.

- Added `ppc64le` support in watchdog (Jean-Michel Scheiwiler)

  And fixed watchdog support on some non-x86 platforms.

- Switched aws.py callback from `boto` to `boto3` (Alexander Kukushkin)

  `boto` 2.x is abandoned since 2018 and fails with python 3.9.

- Periodically refresh service account token on K8s (Haitao Li)

  Since Kubernetes v1.21 service account tokens expire in 1 hour.

- Added `/read-only-sync` monitoring endpoint (Dennis4b)

  It is similar to the `/read-only` but includes only synchronous replicas.

Stability improvements

- Don't copy the logical replication slot to a replica if there is a configuration mismatch in the logical decoding setup with the primary (Alexander Kukushkin)

  A replica won't copy a logical replication slot from the primary anymore if the slot doesn't match the `plugin` or `database` configuration options. Previously, the check for whether the slot matches those configuration options was not performed until after the replica copied the slot and started with it, resulting in unnecessary and repeated restarts.

- Special handling of recovery configuration parameters for PostgreSQL v12+ (Alexander Kukushkin)

  While starting as replica Patroni should be able to update `postgresql.conf` and restart/reload if the leader address has changed by caching current parameters values instead of querying them from `pg_settings`.

- Better handling of IPv6 addresses in the `postgresql.listen` parameters (Alexander Kukushkin)

  Since the `listen` parameter has a port, people try to put IPv6 addresses into square brackets, which were not correctly stripped when there is more than one IP in the list.

- Use `replication` credentials when performing divergence check only on PostgreSQL v10 and older (Alexander Kukushkin)

  If `rewind` is enabled, Patroni will again use either `superuser` or `rewind` credentials on newer Postgres versions.

Bugfixes

- Fixed missing import of `dateutil.parser` (Wesley Mendes)

  Tests weren't failing only because it was also imported from other modules.

- Ensure that `optime` annotation is a string (Sebastian Hasler)

  In certain cases Patroni was trying to pass it as numeric.

- Better handling of failed `pg_rewind` attempt (Alexander Kukushkin)

  If the primary becomes unavailable during `pg_rewind`, `$PGDATA` will be left in a broken state. Following that, Patroni will remove the data directory even if this is not allowed by the configuration.

- Don't remove `slots` annotations from the leader `ConfigMap/Endpoint` when PostgreSQL isn't ready (Alexander Kukushkin)

  If `slots` value isn't passed the annotation will keep the current value.

- Handle concurrency problem with K8s API watchers (Alexander Kukushkin)

  Under certain (unknown) conditions watchers might become stale; as a result, `attempt_to_acquire_leader()` method could fail due to the HTTP status code 409. In that case we reset watchers connections and restart from scratch.

15.11 Version 2.1.3

New features

- Added support for encrypted TLS keys for `patronictl` (Alexander Kukushkin)

  It could be configured via `ctl.keyfile_password` or the `PATRONI_CTL_KEYFILE_PASSWORD` environment variable.

- Added more metrics to the /metrics endpoint (Alexandre Pereira)

  Specifically, `patroni_pending_restart` and `patroni_is_paused`.

- Make it possible to specify multiple hosts in the standby cluster configuration (Michael Banck)

  If the standby cluster is replicating from the Patroni cluster it might be nice to rely on client-side failover which is available in `libpq` since PostgreSQL v10. That is, the `primary_conninfo` on the standby leader and `pg_rewind` setting `target_session_attrs=read-write` in the connection string. The `pgpass` file will be generated with multiple lines (one line per host), and instead of calling `CHECKPOINT` on the primary cluster nodes the standby cluster will wait for `pg_control` to be updated.

Stability improvements

- Compatibility with legacy `psycopg2` (Alexander Kukushkin)

  For example, the `psycopg2` installed from Ubuntu 18.04 packages doesn't have the `UndefinedFile` exception yet.

- Restart `etcd3` watcher if all Etcd nodes don't respond (Alexander Kukushkin)

  If the watcher is alive the `get_cluster()` method continues returning stale information even if all Etcd nodes are failing.

- Don't remove the leader lock in the standby cluster while paused (Alexander Kukushkin)

  Previously the lock was maintained only by the node that was running as a primary and not a standby leader.

Bugfixes

- Fixed bug in the standby-leader bootstrap (Alexander Kukushkin)

  Patroni was considering bootstrap as failed if Postgres didn't start accepting connections after 60 seconds. The bug was introduced in the 2.1.2 release.

- Fixed bug with failover to a cascading standby (Alexander Kukushkin)

  When figuring out which slots should be created on cascading standby we forgot to take into account that the leader might be absent.

- Fixed small issues in Postgres config validator (Alexander Kukushkin)

  Integer parameters introduced in PostgreSQL v14 were failing to validate because min and max values were quoted in the validator.py

- Use replication credentials when checking leader status (Alexander Kukushkin)

  It could be that the `remove_data_directory_on_diverged_timelines` is set, but there is no `rewind_credentials` defined and superuser access between nodes is not allowed.

- Fixed "port in use" error on REST API certificate replacement (Ants Aasma)

  When switching certificates there was a race condition with a concurrent API request. If there is one active during the replacement period then the replacement will error out with a port in use error and Patroni gets stuck in a state without an active API server.

- Fixed a bug in cluster bootstrap if passwords contain `%` characters (Bastien Wirtz)

  The bootstrap method executes the `DO` block, with all parameters properly quoted, but the `cursor.execute()` method didn't like an empty list with parameters passed.

- Fixed the "AttributeError: no attribute 'leader'" exception (Hrvoje Milković)

  It could happen if the synchronous mode is enabled and the DCS content was wiped out.

- Fix bug in divergence timeline check (Alexander Kukushkin)

  Patroni was falsely assuming that timelines have diverged. For pg_rewind it didn't create any problem, but if pg_rewind is not allowed and the `remove_data_directory_on_diverged_timelines` is set, it resulted in reinitializing the former leader.

15.12 Version 2.1.2

New features

- Compatibility with `psycopg>=3.0` (Alexander Kukushkin)

  By default `psycopg2` is preferred. *psycopg>=3.0* will be used only if `psycopg2` is not available or its version is too old.

- Add `dcs_last_seen` field to the REST API (Michael Banck)

  This field notes the last time (as unix epoch) a cluster member has successfully communicated with the DCS. This is useful to identify and/or analyze network partitions.

- Release the leader lock when `pg_controldata` reports "shut down" (Alexander Kukushkin)

  To solve the problem of slow switchover/shutdown in case `archive_command` is slow/failing, Patroni will remove the leader key immediately after `pg_controldata` started reporting PGDATA as `shut down` cleanly and it verified that there is at least one replica that received all changes. If there are no replicas that fulfill this condition the leader key is not removed and the old behavior is retained, i.e. Patroni will keep updating the lock.

- Add `sslcrldir` connection parameter support (Kostiantyn Nemchenko)

  The new connection parameter was introduced in the PostgreSQL v14.

- Allow setting ACLs for ZNodes in Zookeeper (Alwyn Davis)

  Introduce a new configuration option `zookeeper.set_acls` so that Kazoo will apply a default ACL for each ZNode that it creates.

Stability improvements

- Delay the next attempt of recovery till next HA loop (Alexander Kukushkin)

  If Postgres crashed due to out of disk space (for example) and fails to start because of that Patroni is too eagerly trying to recover it flooding logs.

- Add log before demoting, which can take some time (Michael Banck)

  It can take some time for the demote to finish and it might not be obvious from looking at the logs what exactly is going on.

- Improve "I am" status messages (Michael Banck)

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

- Cast to int `wal_keep_segments` when converting to `wal_keep_size` (Jorge Solórzano)

  It is possible to specify `wal_keep_segments` as a string in the global *dynamic configuration* and due to Python being a dynamically typed language the string was simply multiplied. Example: `wal_keep_segments: "100"` was converted to `100100100100100100100100100100100100100100MB`.

- Allow switchover only to sync nodes when synchronous replication is enabled (Alexander Kukushkin)

  In addition to that do the leader race only against known synchronous nodes.

- Use cached role as a fallback when Postgres is slow (Alexander Kukushkin)

  In some extreme cases Postgres could be so slow that the normal monitoring query does not finish in a few seconds. The `statement_timeout` exception not being properly handled could lead to the situation where Postgres was not demoted on time when the leader key expired or the update failed. In case of such exception Patroni will use the cached `role` to determine whether Postgres is running as a primary.

- Avoid unnecessary updates of the member ZNode (Alexander Kukushkin)

  If no values have changed in the members data, the update should not happen.

- Optimize checkpoint after promote (Alexander Kukushkin)

  Avoid doing `CHECKPOINT` if the latest timeline is already stored in `pg_control`. It helps to avoid unnecessary `CHECKPOINT` right after initializing the new cluster with `initdb`.

- Prefer members without `nofailover` when picking sync nodes (Alexander Kukushkin)

  Previously sync nodes were selected only based on the replication lag, hence the node with `nofailover` tag had the same chances to become synchronous as any other node. That behavior was confusing and dangerous at the same time because in case of a failed primary the failover could not happen automatically.

- Remove duplicate hosts from the etcd machine cache (Michael Banck)

  Advertised client URLs in the etcd cluster could be misconfigured. Removing duplicates in Patroni in this case is a low-hanging fruit.

Bugfixes

- Skip temporary replication slots while doing slot management (Alexander Kukushkin)

  Starting from v10 `pg_basebackup` creates a temporary replication slot for WAL streaming and Patroni was trying to drop it because the slot name looks unknown. In order to fix it, we skip all temporary slots when querying `pg_stat_replication_slots` view.

- Ensure `pg_replication_slot_advance()` doesn't timeout (Alexander Kukushkin)

  Patroni was using the default `statement_timeout` in this case and once the call failed there are very high chances that it will never recover, resulting in increased size of `pg_wal` and `pg_catalog` bloat.

- The `/status` wasn't updated on demote (Alexander Kukushkin)

After demoting PostgreSQL the old leader updates the last LSN in DCS. Starting from `2.1.0` the new `/status` key was introduced, but the optime was still written to the `/optime/leader`.

- Handle DCS exceptions when demoting (Alexander Kukushkin)

  While demoting the master due to failure to update the leader lock it could happen that DCS goes completely down and the `get_cluster()` call raises an exception. Not being handled properly it results in Postgres remaining stopped until DCS recovers.

- The `use_unix_socket_repl` didn't work is some cases (Alexander Kukushkin)

  Specifically, if `postgresql.unix_socket_directories` is not set. In this case Patroni is supposed to use the default value from `libpq`.

- Fix a few issues with Patroni REST API (Alexander Kukushkin)

  The `clusters_unlocked` sometimes could be not defined, what resulted in exceptions in the `GET /metrics` endpoint. In addition to that the error handling method was assuming that the `connect_address` tuple always has two elements, while in fact there could be more in case of IPv6.

- Wait for newly promoted node to finish recovery before deciding to rewind (Alexander Kukushkin)

  It could take some time before the actual promote happens and the new timeline is created. Without waiting replicas could come to the conclusion that rewind isn't required.

- Handle missing timelines in a history file when deciding to rewind (Alexander Kukushkin)

  If the current replica timeline is missing in the history file on the primary the replica was falsely assuming that rewind isn't required.

15.13 Version 2.1.1

New features

- Support for ETCD SRV name suffix (David Pavlicek)

  Etcd allows to differentiate between multiple Etcd clusters under the same domain and from now on Patroni also supports it.

- Enrich history with the new leader (huiyalin525)

  It adds the new column to the `patronictl history` output.

- Make the CA bundle configurable for in-cluster Kubernetes config (Aron Parsons)

  By default Patroni is using `/var/run/secrets/kubernetes.io/serviceaccount/ca.crt` and this new feature allows specifying the custom `kubernetes.cacert`.

- Support dynamically registering/deregistering as a Consul service and changing tags (Tommy Li)

  Previously it required Patroni restart.

Bugfixes

- Avoid unnecessary reload of REST API (Alexander Kukushkin)

  The previous release added a feature of reloading REST API certificates if changed on disk. Unfortunately, the reload was happening unconditionally right after the start.

- Don't resolve cluster members when `etcd.use_proxies` is set (Alexander Kukushkin)

  When starting up Patroni checks the healthiness of Etcd cluster by querying the list of members. In addition to that, it also tried to resolve their hostnames, which is not necessary when working with Etcd via proxy and was causing unnecessary warnings.

- Skip rows with NULL values in the `pg_stat_replication` (Alexander Kukushkin)

  It seems that the `pg_stat_replication` view could contain NULL values in the `replay_lsn`, `flush_lsn`, or `write_lsn` fields even when `state = 'streaming'`.

15.14 Version 2.1.0

This version adds compatibility with PostgreSQL v14, makes logical replication slots to survive failover/switchover, implements support of allowlist for REST API, and also reducing the number of logs to one line per heart-beat.

New features

- Compatibility with PostgreSQL v14 (Alexander Kukushkin)

  Unpause WAL replay if Patroni is not in a "pause" mode itself. It could be "paused" due to the change of certain parameters like for example `max_connections` on the primary.

- Failover logical slots (Alexander Kukushkin)

  Make logical replication slots survive failover/switchover on PostgreSQL v11+. The replication slot if copied from the primary to the replica with restart and later the pg_replication_slot_advance() function is used to move it forward. As a result, the slot will already exist before the failover and no events should be lost, but, there is a chance that some events could be delivered more than once.

- Implemented allowlist for Patroni REST API (Alexander Kukushkin)

  If configured, only IP's that matching rules would be allowed to call unsafe endpoints. In addition to that, it is possible to automatically include IP's of members of the cluster to the list.

- Added support of replication connections via unix socket (Mohamad El-Rifai)

  Previously Patroni was always using TCP for replication connection what could cause some issues with SSL verification. Using unix sockets allows exempt replication user from SSL verification.

- Health check on user-defined tags (Arman Jafari Tehrani)

  Along with *predefined tags:* it is possible to specify any number of custom tags that become visible in the `patronictl list` output and in the REST API. From now on it is possible to use custom tags in health checks.

- Added Prometheus `/metrics` endpoint (Mark Mercado, Michael Banck)

  The endpoint exposing the same metrics as `/patroni`.

- Reduced chattiness of Patroni logs (Alexander Kukushkin)

  When everything goes normal, only one line will be written for every run of HA loop.

Breaking changes

- The old `permanent logical replication slots` feature will no longer work with PostgreSQL v10 and older (Alexander Kukushkin)

  The strategy of creating the logical slots after performing a promotion can't guaranty that no logical events are lost and therefore disabled.

- The `/leader` endpoint always returns 200 if the node holds the lock (Alexander Kukushkin)

  Promoting the standby cluster requires updating load-balancer health checks, which is not very convenient and easy to forget. To solve it, we change the behavior of the `/leader` health check endpoint. It will return 200 without taking into account whether the cluster is normal or the `standby_cluster`.

Improvements in Raft support

- Reliable support of Raft traffic encryption (Alexander Kukushkin)

  Due to the different issues in the `PySyncObj` the encryption support was very unstable

- Handle DNS issues in Raft implementation (Alexander Kukushkin)

  If `self_addr` and/or `partner_addrs` are configured using the DNS name instead of IP's the `PySyncObj` was effectively doing resolve only once when the object is created. It was causing problems when the same node was coming back online with a different IP.

Stability improvements

- Compatibility with `psycopg2-2.9+` (Alexander Kukushkin)

  In `psycopg2` the `autocommit = True` is ignored in the `with connection` block, which breaks replication protocol connections.

- Fix excessive HA loop runs with Zookeeper (Alexander Kukushkin)

  Update of member ZNodes was causing a chain reaction and resulted in running the HA loops multiple times in a row.

- Reload if REST API certificate is changed on disk (Michael Todorovic)

  If the REST API certificate file was updated in place Patroni didn't perform a reload.

- Don't create pgpass dir if kerberos auth is used (Kostiantyn Nemchenko)

  Kerberos and password authentication are mutually exclusive.

- Fixed little issues with custom bootstrap (Alexander Kukushkin)

  Start Postgres with `hot_standby=off` only when we do a PITR and restart it after PITR is done.

Bugfixes

- Compatibility with `kazoo-2.7+` (Alexander Kukushkin)

  Since Patroni is handling retries on its own, it is relying on the old behavior of `kazoo` that requests to a Zookeeper cluster are immediately discarded when there are no connections available.

- Explicitly request the version of Etcd v3 cluster when it is known that we are connecting via proxy (Alexander Kukushkin)

  Patroni is working with Etcd v3 cluster via gPRC-gateway and it depending on the cluster version different endpoints (`/v3`, `/v3beta`, or `/v3alpha`) must be used. The version was resolved only together with the cluster topology, but since the latter was never done when connecting via proxy.

15.15 Version 2.0.2

New features

- Ability to ignore externally managed replication slots (James Coleman)

  Patroni is trying to remove any replication slot which is unknown to it, but there are certainly cases when replication slots should be managed externally. From now on it is possible to configure slots that should not be removed.

- Added support for cipher suite limitation for REST API (Gunnar "Nick" Bluth)

  It could be configured via `restapi.ciphers` or the `PATRONI_RESTAPI_CIPHERS` environment variable.

- Added support for encrypted TLS keys for REST API (Jonathan S. Katz)

  It could be configured via `restapi.keyfile_password` or the `PATRONI_RESTAPI_KEYFILE_PASSWORD` environment variable.

- Constant time comparison of REST API authentication credentials (Alex Brasetvik)

  Use `hmac.compare_digest()` instead of ==, which is vulnerable to timing attack.

- Choose synchronous nodes based on replication lag (Krishna Sarabu)

  If the replication lag on the synchronous node starts exceeding the configured threshold it could be demoted to asynchronous and/or replaced by the other node. Behaviour is controlled with `maximum_lag_on_syncnode`.

Stability improvements

- Start postgres with `hot_standby = off` when doing custom bootstrap (Igor Yanchenko)

  During custom bootstrap Patroni is restoring the basebackup, starting Postgres up, and waiting until recovery finishes. Some PostgreSQL parameters on the standby can't be smaller than on the primary and if the new value (restored from WAL) is higher than the configured one, Postgres panics and stops. In order to avoid such behavior we will do custom bootstrap without `hot_standby` mode.

- Warn the user if the required watchdog is not healthy (Nicolas Thauvin)

  When the watchdog device is not writable or missing in required mode, the member cannot be promoted. Added a warning to show the user where to search for this misconfiguration.

- Better verbosity for single-user mode recovery (Alexander Kukushkin)

  If Patroni notices that PostgreSQL wasn't shutdown clearly, in certain cases the crash-recovery is executed by starting Postgres in single-user mode. It could happen that the recovery failed (for example due to the lack of space on disk) but errors were swallowed.

- Added compatibility with `python-consul2` module (Alexander Kukushkin, Wilfried Roset)

  The good old `python-consul` is not maintained since a few years, therefore someone created a fork with new features and bug-fixes.

- Don't use `bypass_api_service` when running `patronictl` (Alexander Kukushkin)

  When a K8s pod is running in a non-`default` namespace it does not necessarily have enough permissions to query the `kubernetes` endpoint. In this case Patroni shows the warning and ignores the `bypass_api_service` setting. In case of `patronictl` the warning was a bit annoying.

- Create `raft.data_dir` if it doesn't exists or make sure that it is writable (Mark Mercado)

  Improves user-friendliness and usability.

Bugfixes

- Don't interrupt restart or promote if lost leader lock in pause (Alexander Kukushkin)

  In pause it is allowed to run postgres as primary without lock.

- Fixed issue with `shutdown_request()` in the REST API (Nicolas Limage)

  In order to improve handling of SSL connections and delay the handshake until thread is started Patroni overrides a few methods in the `HTTPServer`. The `shutdown_request()` method was forgotten.

- Fixed issue with sleep time when using Zookeeper (Alexander Kukushkin)

  There were chances that Patroni was sleeping up to twice longer between running HA code.

- Fixed invalid `os.symlink()` calls when moving data directory after failed bootstrap (Andrew L'Ecuyer)

If the bootstrap failed Patroni is renaming data directory, pg_wal, and all tablespaces. After that it updates symlinks so filesystem remains consistent. The symlink creation was failing due to the `src` and `dst` arguments being swapped.

- Fixed bug in the post_bootstrap() method (Alexander Kukushkin)

  If the superuser password wasn't configured Patroni was failing to call the `post_init` script and therefore the whole bootstrap was failing.

- Fixed an issues with pg_rewind in the standby cluster (Alexander Kukushkin)

  If the superuser name is different from Postgres, the `pg_rewind` in the standby cluster was failing because the connection string didn't contain the database name.

- Exit only if authentication with Etcd v3 explicitly failed (Alexander Kukushkin)

  On start Patroni performs discovery of Etcd cluster topology and authenticates if it is necessarily. It could happen that one of etcd servers is not accessible, Patroni was trying to perform authentication on this server and failing instead of retrying with the next node.

- Handle case with psutil cmdline() returning empty list (Alexander Kukushkin)

  Zombie processes are still postmasters children, but they don't have cmdline()

- Treat `PATRONI_KUBERNETES_USE_ENDPOINTS` environment variable as boolean (Alexander Kukushkin)

  Not doing so was making impossible disabling `kubernetes.use_endpoints` via environment.

- Improve handling of concurrent endpoint update errors (Alexander Kukushkin)

  Patroni will explicitly query the current endpoint object, verify that the current pod still holds the leader lock and repeat the update.

15.16 Version 2.0.1

New features

- Use `more` as pager in `patronictl edit-config` if `less` is not available (Pavel Golub)

  On Windows it would be the `more.com`. In addition to that, `cdiff` was changed to `ydiff` in `requirements.txt`, but `patronictl` still supports both for compatibility.

- Added support of `raft` `bind_addr` and `password` (Alexander Kukushkin)

  `raft.bind_addr` might be useful when running behind NAT. `raft.password` enables traffic encryption (requires the `cryptography` module).

- Added `sslpassword` connection parameter support (Kostiantyn Nemchenko)

  The connection parameter was introduced in PostgreSQL 13.

Stability improvements

- Changed the behavior in pause (Alexander Kukushkin)

  1. Patroni will not call the `bootstrap` method if the `PGDATA` directory is missing/empty.
  2. Patroni will not exit on sysid mismatch in pause, only log a warning.
  3. The node will not try to grab the leader key in pause mode if Postgres is running not in recovery (accepting writes) but the sysid doesn't match with the initialize key.

- Apply `master_start_timeout` when executing crash recovery (Alexander Kukushkin)

  If Postgres crashed on the leader node, Patroni does a crash-recovery by starting Postgres in single-user mode. During the crash-recovery the leader lock is being updated. If the crash-recovery didn't finish in `master_start_timeout` seconds, Patroni will stop it forcefully and release the leader lock.

- Removed the `secure` extra from the `urllib3` requirements (Alexander Kukushkin)

  The only reason for adding it there was the `ipaddress` dependency for python 2.7.

Bugfixes

- Fixed a bug in the `Kubernetes.update_leader()` (Alexander Kukushkin)

  An unhandled exception was preventing demoting the primary when the update of the leader object failed.

- Fixed hanging `patronictl` when RAFT is being used (Alexander Kukushkin)

  When using `patronictl` with Patroni config, `self_addr` should be added to the `partner_addrs`.

- Fixed bug in `get_guc_value()` (Alexander Kukushkin)

  Patroni was failing to get the value of `restore_command` on PostgreSQL 12, therefore fetching missing WALs for `pg_rewind` didn't work.

15.17 Version 2.0.0

This version enhances compatibility with PostgreSQL 13, adds support of multiple synchronous standbys, has significant improvements in handling of `pg_rewind`, adds support of Etcd v3 and Patroni on pure RAFT (without Etcd, Consul, or Zookeeper), and makes it possible to optionally call the `pre_promote` (fencing) script.

PostgreSQL 13 support

- Don't fire `on_reload` when promoting to `standby_leader` on PostgreSQL 13+ (Alexander Kukushkin)

  When promoting to `standby_leader` we change `primary_conninfo`, update the role and reload Postgres. Since `on_role_change` and `on_reload` effectively duplicate each other, Patroni will call only `on_role_change`.

- Added support for `gssencmode` and `channel_binding` connection parameters (Alexander Kukushkin)

  PostgreSQL 12 introduced `gssencmode` and 13 `channel_binding` connection parameters and now they can be used if defined in the `postgresql.authentication` section.

- Handle renaming of `wal_keep_segments` to `wal_keep_size` (Alexander Kukushkin)

  In case of misconfiguration (`wal_keep_segments` on 13 and `wal_keep_size` on older versions) Patroni will automatically adjust the configuration.

- Use `pg_rewind` with `--restore-target-wal` on 13 if possible (Alexander Kukushkin)

  On PostgreSQL 13 Patroni checks if `restore_command` is configured and tells `pg_rewind` to use it.

New features

- [BETA] Implemented support of Patroni on pure RAFT (Alexander Kukushkin)

  This makes it possible to run Patroni without 3rd party dependencies, like Etcd, Consul, or Zookeeper. For HA you will have to run either three Patroni nodes or two nodes with Patroni and one node with `patroni_raft_controller`. For more information please check the *documentation*.

- [BETA] Implemented support for Etcd v3 protocol via gPRC-gateway (Alexander Kukushkin)

  Etcd 3.0 was released more than four years ago and Etcd 3.4 has v2 disabled by default. There are also chances that v2 will be completely removed from Etcd, therefore we implemented support of Etcd v3 in Patroni. In order to start using it you have to explicitly create the `etcd3` section is the Patroni configuration file.

- Supporting multiple synchronous standbys (Krishna Sarabu)

  It allows running a cluster with more than one synchronous replicas. The maximum number of synchronous replicas is controlled by the new parameter `synchronous_node_count`. It is set to 1 by default and has no effect when the `synchronous_mode` is set to `off`.

- Added possibility to call the `pre_promote` script (Sergey Dudoladov)

  Unlike callbacks, the `pre_promote` script is called synchronously after acquiring the leader lock, but before promoting Postgres. If the script fails or exits with a non-zero exitcode, the current node will release the leader lock.

- Added support for configuration directories (Floris van Nee)

  YAML files in the directory loaded and applied in alphabetical order.

- Advanced validation of PostgreSQL parameters (Alexander Kukushkin)

  In case the specific parameter is not supported by the current PostgreSQL version or when its value is incorrect, Patroni will remove the parameter completely or try to fix the value.

- Wake up the main thread when the forced checkpoint after promote completed (Alexander Kukushkin)

  Replicas are waiting for checkpoint indication via member key of the leader in DCS. The key is normally updated only once per HA loop. Without waking the main thread up, replicas will have to wait up to `loop_wait` seconds longer than necessary.

- Use of `pg_stat_wal_recevier` view on 9.6+ (Alexander Kukushkin)

  The view contains up-to-date values of `primary_conninfo` and `primary_slot_name`, while the contents of `recovery.conf` could be stale.

- Improved handing of IPv6 addresses in the Patroni config file (Mateusz Kowalski)

  The IPv6 address is supposed to be enclosed into square brackets, but Patroni was expecting to get it plain. Now both formats are supported.

- Added Consul `service_tags` configuration parameter (Robert Edström)

  They are useful for dynamic service discovery, for example by load balancers.

- Implemented SSL support for Zookeeper (Kostiantyn Nemchenko)

  It requires `kazoo>=2.6.0`.

- Implemented `no_params` option for custom bootstrap method (Kostiantyn Nemchenko)

  It allows calling `wal-g`, `pgBackRest` and other backup tools without wrapping them into shell scripts.

- Move WAL and tablespaces after a failed init (Feike Steenbergen)

  When doing `reinit`, Patroni was already removing not only `PGDATA` but also the symlinked WAL directory and tablespaces. Now the `move_data_directory()` method will do a similar job, i.e. rename WAL directory and tablespaces and update symlinks in PGDATA.

Improved in pg_rewind support

- Improved timeline divergence check (Alexander Kukushkin)

We don't need to rewind when the replayed location on the replica is not ahead of the switchpoint or the end of the checkpoint record on the former primary is the same as the switchpoint. In order to get the end of the checkpoint record we use `pg_waldump` and parse its output.

- Try to fetch missing WAL if `pg_rewind` complains about it (Alexander Kukushkin)

  It could happen that the WAL segment required for `pg_rewind` doesn't exist in the `pg_wal` directory anymore and therefore `pg_rewind` can't find the checkpoint location before the divergence point. Starting from PostgreSQL 13 `pg_rewind` could use `restore_command` for fetching missing WALs. For older PostgreSQL versions Patroni parses the errors of a failed rewind attempt and tries to fetch the missing WAL by calling the `restore_command` on its own.

- Detect a new timeline in the standby cluster and trigger rewind/reinitialize if necessary (Alexander Kukushkin)

  The `standby_cluster` is decoupled from the primary cluster and therefore doesn't immediately know about leader elections and timeline switches. In order to detect the fact, the `standby_leader` periodically checks for new history files in `pg_wal`.

- Shorten and beautify history log output (Alexander Kukushkin)

  When Patroni is trying to figure out the necessity of `pg_rewind`, it could write the content of the history file from the primary into the log. The history file is growing with every failover/switchover and eventually starts taking up too many lines, most of which are not so useful. Instead of showing the raw data, Patroni will show only 3 lines before the current replica timeline and 2 lines after.

Improvements on K8s

- Get rid of `kubernetes` python module (Alexander Kukushkin)

  The official python kubernetes client contains a lot of auto-generated code and therefore very heavy. Patroni uses only a small fraction of K8s API endpoints and implementing support for them wasn't hard.

- Make it possible to bypass the `kubernetes` service (Alexander Kukushkin)

  When running on K8s, Patroni is usually communicating with the K8s API via the `kubernetes` service, the address of which is exposed in the `KUBERNETES_SERVICE_HOST` environment variable. Like any other service, the `kubernetes` service is handled by `kube-proxy`, which in turn, depending on the configuration, is either relying on a userspace program or `iptables` for traffic routing. Skipping the intermediate component and connecting directly to the K8s master nodes allows us to implement a better retry strategy and mitigate risks of demoting Postgres when K8s master nodes are upgraded.

- Sync HA loops of all pods of a Patroni cluster (Alexander Kukushkin)

  Not doing so was increasing failure detection time from `ttl` to `ttl + loop_wait`.

- Populate `references` and `nodename` in the subsets addresses on K8s (Alexander Kukushkin)

  Some load-balancers are relying on this information.

- Fix possible race conditions in the `update_leader()` (Alexander Kukushkin)

  The concurrent update of the leader configmap or endpoint happening outside of Patroni might cause the `update_leader()` call to fail. In this case Patroni rechecks that the current node is still owning the leader lock and repeats the update.

- Explicitly disallow patching non-existent config (Alexander Kukushkin)

  For DCS other than `kubernetes` the PATCH call is failing with an exception due to `cluster.config` being `None`, but on Kubernetes it was happily creating the config annotation and preventing writing bootstrap configuration after the bootstrap finished.

- Fix bug in `pause` (Alexander Kukushkin)

Replicas were removing `primary_conninfo` and restarting Postgres when the leader key was absent, but they should do nothing.

Improvements in REST API

- Defer TLS handshake until worker thread has started (Alexander Kukushkin, Ben Harris)

  If the TLS handshake was done in the API thread and the client-side didn't send any data, the API thread was blocked (risking DoS).

- Check `basic-auth` independently from client certificate in REST API (Alexander Kukushkin)

  Previously only the client certificate was validated. Doing two checks independently is an absolutely valid use-case.

- Write double `CRLF` after HTTP headers of the `OPTIONS` request (Sergey Burladyan)

  HAProxy was happy with a single `CRLF`, while Consul health-check complained about broken connection and unexpected EOF.

- `GET /cluster` was showing stale members info for Zookeeper (Alexander Kukushkin)

  The endpoint was using the Patroni internal cluster view. For Patroni itself it didn't cause any issues, but when exposed to the outside world we need to show up-to-date information, especially replication lag.

- Fixed health-checks for standby cluster (Alexander Kukushkin)

  The `GET /standby-leader` for a master and `GET /master` for a `standby_leader` were incorrectly responding with 200.

- Implemented `DELETE /switchover` (Alexander Kukushkin)

  The REST API call deletes the scheduled switchover.

- Created `/readiness` and `/liveness` endpoints (Alexander Kukushkin)

  They could be useful to eliminate "unhealthy" pods from subsets addresses when the K8s service is used with label selectors.

- Enhanced `GET /replica` and `GET /async` REST API health-checks (Krishna Sarabu, Alexander Kukushkin)

  Checks now support optional keyword `?lag=<max-lag>` and will respond with 200 only if the lag is smaller than the supplied value. If relying on this feature please keep in mind that information about WAL position on the leader is updated only every `loop_wait` seconds!

- Added support for user defined HTTP headers in the REST API response (Yogesh Sharma)

  This feature might be useful if requests are made from a browser.

Improvements in patronictl

- Don't try to call non-existing leader in `patronictl pause` (Alexander Kukushkin)

  While pausing a cluster without a leader on K8s, `patronictl` was showing warnings that member "None" could not be accessed.

- Handle the case when member `conn_url` is missing (Alexander Kukushkin)

  On K8s it is possible that the pod doesn't have the necessary annotations because Patroni is not yet running. It was making `patronictl` to fail.

- Added ability to print ASCII cluster topology (Maxim Fedotov, Alexander Kukushkin)

  It is very useful to get overview of the cluster with cascading replication.

- Implement `patronictl flush switchover` (Alexander Kukushkin)

  Before that `patronictl flush` only supported cancelling scheduled restarts.

Bugfixes

- Attribute error during bootstrap of the cluster with existing PGDATA (Krishna Sarabu)

  When trying to create/update the `/history` key, Patroni was accessing the `ClusterConfig` object which wasn't created in DCS yet.

- Improved exception handling in Consul (Alexander Kukushkin)

  Unhandled exception in the `touch_member()` method caused the whole Patroni process to crash.

- Enforce `synchronous_commit=local` for the `post_init` script (Alexander Kukushkin)

  Patroni was already doing that when creating users (`replication`, `rewind`), but missing it in the case of `post_init` was an oversight. As a result, if the script wasn't doing it internally on it's own the bootstrap in `synchronous_mode` wasn't able to finish.

- Increased `maxsize` in the Consul pool manager (ponvenkates)

  With the default `size=1` some warnings were generated.

- Patroni was wrongly reporting Postgres as running (Alexander Kukushkin)

  The state wasn't updated when for example Postgres crashed due to an out-of-disk error.

- Put * into `pgpass` instead of missing or empty values (Alexander Kukushkin)

  If for example the `standby_cluster.port` is not specified, the `pgpass` file was incorrectly generated.

- Skip physical replication slot creation on the leader node with special characters (Krishna Sarabu)

  Patroni appeared to be creating a dormant slot (when `slots` defined) for the leader node when the name contained special chars such as '-' (for e.g. "abc-us-1").

- Avoid removing non-existent `pg_hba.conf` in the custom bootstrap (Krishna Sarabu)

  Patroni was failing if `pg_hba.conf` happened to be located outside of the `pgdata` dir after custom bootstrap.

15.18 Version 1.6.5

New features

- Master stop timeout (Krishna Sarabu)

  The number of seconds Patroni is allowed to wait when stopping Postgres. Effective only when `synchronous_mode` is enabled. When set to value greater than 0 and the `synchronous_mode` is enabled, Patroni sends `SIGKILL` to the postmaster if the stop operation is running for more than the value set by `master_stop_timeout`. Set the value according to your durability/availability tradeoff. If the parameter is not set or set to non-positive value, `master_stop_timeout` does not have an effect.

- Don't create permanent physical slot with name of the primary (Alexander Kukushkin)

  It is a common problem that the primary recycles WAL segments while the replica is down. Now we have a good solution for static clusters, with a fixed number of nodes and names that never change. You just need to list the names of all nodes in the `slots` so the primary will not remove the slot when the node is down (not registered in DCS).

- First draft of Config Validator (Igor Yanchenko)

  Use `patroni --validate-config patroni.yaml` in order to validate Patroni configuration.

- Possibility to configure max length of timelines history (Krishna Sarabu)

  Patroni writes the history of failovers/switchovers into the `/history` key in DCS. Over time the size of this key becomes big, but in most cases only the last few lines are interesting. The `max_timelines_history` parameter allows to specify the maximum number of timeline history items to be kept in DCS.

- Kazoo 2.7.0 compatibility (Danyal Prout)

  Some non-public methods in Kazoo changed their signatures, but Patroni was relying on them.

Improvements in patronictl

- Show member tags (Kostiantyn Nemchenko, Alexander Kukushkin)

  Tags are configured individually for every node and there was no easy way to get an overview of them

- Improve members output (Alexander Kukushkin)

  The redundant cluster name won't be shown anymore on every line, only in the table header.

```
$ 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 |                     ␣
↪|
+--------------+----------------+--------+---------+----+-----------+---------------------
↪+
```

- Fail if a config file is specified explicitly but not found (Kaarel Moppel)

  Previously `patronictl` was only reporting a `DEBUG` message.

- Solved the problem of not initialized K8s pod breaking patronictl (Alexander Kukushkin)

  Patroni is relying on certain pod annotations on K8s. When one of the Patroni pods is stopping or starting there is no valid annotation yet and `patronictl` was failing with an exception.

Stability improvements

- Apply 1 second backoff if LIST call to K8s API server failed (Alexander Kukushkin)

  It is mostly necessary to avoid flooding logs, but also helps to prevent starvation of the main thread.

- Retry if the `retry-after` HTTP header is returned by K8s API (Alexander Kukushkin)

  If the K8s API server is overwhelmed with requests it might ask to retry.

- Scrub `KUBERNETES_` environment from the postmaster (Feike Steenbergen)

  The `KUBERNETES_` environment variables are not required for PostgreSQL, yet having them exposed to the postmaster will also expose them to backends and to regular database users (using pl/perl for example).

- Clean up tablespaces on reinitialize (Krishna Sarabu)

  During reinit, Patroni was removing only `PGDATA` and leaving user-defined tablespace directories. This is causing Patroni to loop in reinit. The previous workarond for the problem was implementing the *custom bootstrap* script.

- Explicitly execute `CHECKPOINT` after promote happened (Alexander Kukushkin)

  It helps to reduce the time before the new primary is usable for `pg_rewind`.

- Smart refresh of Etcd members (Alexander Kukushkin)

  In case Patroni failed to execute a request on all members of the Etcd cluster, Patroni will re-check `A` or `SRV` records for changes of IPs/hosts before retrying the next time.

- Skip missing values from `pg_controldata` (Feike Steenbergen)

  Values are missing when trying to use binaries of a version that doesn't match PGDATA. Patroni will try to start Postgres anyway, and Postgres will complain that the major version doesn't match and abort with an error.

Bugfixes

- Disable SSL verification for Consul when required (Julien Riou)

  Starting from a certain version of `urllib3`, the `cert_reqs` must be explicitly set to `ssl.CERT_NONE` in order to effectively disable SSL verification.

- Avoid opening replication connection on every cycle of HA loop (Alexander Kukushkin)

  Regression was introduced in 1.6.4.

- Call `on_role_change` callback on failed primary (Alexander Kukushkin)

  In certain cases it could lead to the virtual IP remaining attached to the old primary. Regression was introduced in 1.4.5.

- Reset rewind state if postgres started after successful pg_rewind (Alexander Kukushkin)

  As a result of this bug Patroni was starting up manually shut down postgres in the pause mode.

- Convert `recovery_min_apply_delay` to `ms` when checking `recovery.conf`

  Patroni was indefinitely restarting replica if `recovery_min_apply_delay` was configured on PostgreSQL older than 12.

- PyInstaller compatibility (Alexander Kukushkin)

  PyInstaller freezes (packages) Python applications into stand-alone executables. The compatibility was broken when we switched to the `spawn` method instead of `fork` for `multiprocessing`.

15.19 Version 1.6.4

New features

- Implemented `--wait` option for `patronictl reinit` (Igor Yanchenko)

  Patronictl will wait for `reinit` to finish is the `--wait` option is used.

- Further improvements of Windows support (Igor Yanchenko, Alexander Kukushkin)

  1. All shell scripts which are used for integration testing are rewritten in python
  2. The `pg_ctl kill` will be used to stop postgres on non posix systems
  3. Don't try to use unix-domain sockets

Stability improvements

- Make sure `unix_socket_directories` and `stats_temp_directory` exist (Igor Yanchenko)

  Upon the start of Patroni and Postgres make sure that `unix_socket_directories` and `stats_temp_directory` exist or try to create them. Patroni will exit if failed to create them.

- Make sure `postgresql.pgpass` is located in the place where Patroni has write access (Igor Yanchenko)

  In case if it doesn't have a write access Patroni will exit with exception.

- Disable Consul `serfHealth` check by default (Kostiantyn Nemchenko)

  Even in case of little network problems the failing `serfHealth` leads to invalidation of all sessions associated with the node. Therefore, the leader key is lost much earlier than `ttl` which causes unwanted restarts of replicas and maybe demotion of the primary.

- Configure tcp keepalives for connections to K8s API (Alexander Kukushkin)

  In case if we get nothing from the socket after TTL seconds it can be considered dead.

- Avoid logging of passwords on user creation (Alexander Kukushkin)

  If the password is rejected or logging is configured to verbose or not configured at all it might happen that the password is written into postgres logs. In order to avoid it Patroni will change `log_statement`, `log_min_duration_statement`, and `log_min_error_statement` to some safe values before doing the attempt to create/update user.

Bugfixes

- Use `restore_command` from the `standby_cluster` config on cascading replicas (Alexander Kukushkin)

  The `standby_leader` was already doing it from the beginning the feature existed. Not doing the same on replicas might prevent them from catching up with standby leader.

- Update timeline reported by the standby cluster (Alexander Kukushkin)

  In case of timeline switch the standby cluster was correctly replicating from the primary but `patronictl` was reporting the old timeline.

- Allow certain recovery parameters be defined in the custom_conf (Alexander Kukushkin)

  When doing validation of recovery parameters on replica Patroni will skip `archive_cleanup_command`, `promote_trigger_file`, `recovery_end_command`, `recovery_min_apply_delay`, and `restore_command` if they are not defined in the patroni config but in files other than `postgresql.auto.conf` or `postgresql.conf`.

- Improve handling of postgresql parameters with period in its name (Alexander Kukushkin)

  Such parameters could be defined by extensions where the unit is not necessarily a string. Changing the value might require a restart (for example `pg_stat_statements.max`).

- Improve exception handling during shutdown (Alexander Kukushkin)

  During shutdown Patroni is trying to update its status in the DCS. If the DCS is inaccessible an exception might be raised. Lack of exception handling was preventing logger thread from stopping.

15.20 Version 1.6.3

Bugfixes

- Don't expose password when running `pg_rewind` (Alexander Kukushkin)

  Bug was introduced in the #1301

- Apply connection parameters specified in the `postgresql.authentication` to `pg_basebackup` and custom replica creation methods (Alexander Kukushkin)

  They were relying on url-like connection string and therefore parameters never applied.

15.21 Version 1.6.2

New features

- Implemented `patroni --version` (Igor Yanchenko)

  It prints the current version of Patroni and exits.

- Set the `user-agent` http header for all http requests (Alexander Kukushkin)

  Patroni is communicating with Consul, Etcd, and Kubernetes API via the http protocol. Having a specifically crafted `user-agent` (example: `Patroni/1.6.2 Python/3.6.8 Linux`) might be useful for debugging and monitoring.

- Make it possible to configure log level for exception tracebacks (Igor Yanchenko)

  If you set `log.traceback_level=DEBUG` the tracebacks will be visible only when `log.level=DEBUG`. The default behavior remains the same.

Stability improvements

- Avoid importing all DCS modules when searching for the module required by the config file (Alexander Kukushkin)

  There is no need to import modules for Etcd, Consul, and Kubernetes if we need only e.g. Zookeeper. It helps to reduce memory usage and solves the problem of having INFO messages `Failed to import smth`.

- Removed python `requests` module from explicit requirements (Alexander Kukushkin)

  It wasn't used for anything critical, but causing a lot of problems when the new version of `urllib3` is released.

- Improve handling of `etcd.hosts` written as a comma-separated string instead of YAML array (Igor Yanchenko)

  Previously it was failing when written in format `host1:port1, host2:port2` (the space character after the comma).

Usability improvements

- Don't force users to choose members from an empty list in `patronictl` (Igor Yanchenko)

  If the user provides a wrong cluster name, we will raise an exception rather than ask to choose a member from an empty list.

- Make the error message more helpful if the REST API cannot bind (Igor Yanchenko)

  For an inexperienced user it might be hard to figure out what is wrong from the Python stacktrace.

Bugfixes

- Fix calculation of `wal_buffers` (Alexander Kukushkin)

  The base unit has been changed from 8 kB blocks to bytes in PostgreSQL 11.

- Use `passfile` in `primary_conninfo` only on PostgreSQL 10+ (Alexander Kukushkin)

  On older versions there is no guarantee that `passfile` will work, unless the latest version of `libpq` is installed.

15.22 Version 1.6.1

New features

- Added `PATRONICTL_CONFIG_FILE` environment variable (msvechla)

  It allows configuring the `--config-file` argument for `patronictl` from the environment.

- Implement `patronictl history` (Alexander Kukushkin)

  It shows the history of failovers/switchovers.

- Pass `-c statement_timeout=0` in `PGOPTIONS` when doing `pg_rewind` (Alexander Kukushkin)

  It protects from the case when `statement_timeout` on the server is set to some small value and one of the statements executed by pg_rewind is canceled.

- Allow lower values for PostgreSQL configuration (Soulou)

  Patroni didn't allow some of the PostgreSQL configuration parameters be set smaller than some hardcoded values. Now the minimal allowed values are smaller, default values have not been changed.

- Allow for certificate-based authentication (Jonathan S. Katz)

  This feature enables certificate-based authentication for superuser, replication, rewind accounts and allows the user to specify the `sslmode` they wish to connect with.

- Use the `passfile` in the `primary_conninfo` instead of password (Alexander Kukushkin)

  It allows to avoid setting `600` permissions on postgresql.conf

- Perform `pg_ctl reload` regardless of config changes (Alexander Kukushkin)

  It is possible that some config files are not controlled by Patroni. When somebody is doing a reload via the REST API or by sending SIGHUP to the Patroni process, the usual expectation is that Postgres will also be reloaded. Previously it didn't happen when there were no changes in the `postgresql` section of Patroni config.

- Compare all recovery parameters, not only `primary_conninfo` (Alexander Kukushkin)

  Previously the `check_recovery_conf()` method was only checking whether `primary_conninfo` has changed, never taking into account all other recovery parameters.

- Make it possible to apply some recovery parameters without restart (Alexander Kukushkin)

  Starting from PostgreSQL 12 the following recovery parameters could be changed without restart: `archive_cleanup_command`, `promote_trigger_file`, `recovery_end_command`, and `recovery_min_apply_delay`. In future Postgres releases this list will be extended and Patroni will support it automatically.

- Make it possible to change `use_slots` online (Alexander Kukushkin)

  Previously it required restarting Patroni and removing slots manually.

- Remove only `PATRONI_` prefixed environment variables when starting up Postgres (Cody Coons)

  It will solve a lot of problems with running different Foreign Data Wrappers.

Stability improvements

- Use LIST + WATCH when working with K8s API (Alexander Kukushkin)

  It allows to efficiently receive object changes (pods, endpoints/configmaps) and makes less stress on K8s master nodes.

- Improve the workflow when PGDATA is not empty during bootstrap (Alexander Kukushkin)

  According to the `initdb` source code it might consider a PGDATA empty when there are only `lost+found` and `.dotfiles` in it. Now Patroni does the same. If `PGDATA` happens to be non-empty, and at the same time not valid from the `pg_controldata` point of view, Patroni will complain and exit.

- Avoid calling expensive `os.listdir()` on every HA loop (Alexander Kukushkin)

  When the system is under IO stress, `os.listdir()` could take a few seconds (or even minutes) to execute, badly affecting the HA loop of Patroni. This could even cause the leader key to disappear from DCS due to the lack of updates. There is a better and less expensive way to check that the PGDATA is not empty. Now we check the presence of the `global/pg_control` file in the PGDATA.

- Some improvements in logging infrastructure (Alexander Kukushkin)

  Previously there was a possibility to loose the last few log lines on shutdown because the logging thread was a `daemon` thread.

- Use `spawn` multiprocessing start method on python 3.4+ (Maciej Kowalczyk)

  It is a known issue in Python that threading and multiprocessing do not mix well. Switching from the default method `fork` to the `spawn` is a recommended workaround. Not doing so might result in the Postmaster starting process hanging and Patroni indefinitely reporting `INFO: restarting after failure in progress`, while Postgres is actually up and running.

Improvements in REST API

- Make it possible to check client certificates in the REST API (Alexander Kukushkin)

  If the `verify_client` is set to `required`, Patroni will check client certificates for all REST API calls. When it is set to `optional`, client certificates are checked for all unsafe REST API endpoints.

- Return the response code 503 for the `GET /replica` health check request if Postgres is not running (Alexander Anikin)

  Postgres might spend significant time in recovery before it starts accepting client connections.

- Implement `/history` and `/cluster` endpoints (Alexander Kukushkin)

  The `/history` endpoint shows the content of the `history` key in DCS. The `/cluster` endpoint shows all cluster members and some service info like pending and scheduled restarts or switchovers.

Improvements in Etcd support

- Retry on Etcd RAFT internal error (Alexander Kukushkin)

  When the Etcd node is being shut down, it sends `response code=300, data='etcdserver: server stopped'`, which was causing Patroni to demote the primary.

- Don't give up on Etcd request retry too early (Alexander Kukushkin)

  When there were some network problems, Patroni was quickly exhausting the list of Etcd nodes and giving up without using the whole `retry_timeout`, potentially resulting in demoting the primary.

Bugfixes

- Disable `synchronous_commit` when granting execute permissions to the `pg_rewind` user (kremius)

If the bootstrap is done with `synchronous_mode_strict: true` the *GRANT EXECUTE* statement was waiting indefinitely due to the non-synchronous nodes being available.

- Fix memory leak on python 3.7 (Alexander Kukushkin)

  Patroni is using `ThreadingMixIn` to process REST API requests and python 3.7 made threads spawn for every request non-daemon by default.

- Fix race conditions in asynchronous actions (Alexander Kukushkin)

  There was a chance that `patronictl reinit --force` could be overwritten by the attempt to recover stopped Postgres. This ended up in a situation when Patroni was trying to start Postgres while basebackup was running.

- Fix race condition in `postmaster_start_time()` method (Alexander Kukushkin)

  If the method is executed from the REST API thread, it requires a separate cursor object to be created.

- Fix the problem of not promoting the sync standby that had a name containing upper case letters (Alexander Kukushkin)

  We converted the name to the lower case because Postgres was doing the same while comparing the `application_name` with the value in `synchronous_standby_names`.

- Kill all children along with the callback process before starting the new one (Alexander Kukushkin)

  Not doing so makes it hard to implement callbacks in bash and eventually can lead to the situation when two callbacks are running at the same time.

- Fix 'start failed' issue (Alexander Kukushkin)

  Under certain conditions the Postgres state might be set to 'start failed' despite Postgres being up and running.

15.23 Version 1.6.0

This version adds compatibility with PostgreSQL 12, makes is possible to run pg_rewind without superuser on PostgreSQL 11 and newer, and enables IPv6 support.

New features

- Psycopg2 was removed from requirements and must be installed independently (Alexander Kukushkin)

  Starting from 2.8.0 `psycopg2` was split into two different packages, `psycopg2`, and `psycopg2-binary`, which could be installed at the same time into the same place on the filesystem. In order to decrease dependency hell problem, we let a user choose how to install it. There are a few options available, please consult the *documentation*.

- Compatibility with PostgreSQL 12 (Alexander Kukushkin)

  Starting from PostgreSQL 12 there is no `recovery.conf` anymore and all former recovery parameters are converted into GUC. In order to protect from `ALTER SYSTEM SET primary_conninfo` or similar, Patroni will parse `postgresql.auto.conf` and remove all standby and recovery parameters from there. Patroni config remains backward compatible. For example despite `restore_command` being a GUC, one can still specify it in the `postgresql.recovery_conf.restore_command` section and Patroni will write it into `postgresql.conf` for PostgreSQL 12.

- Make it possible to use `pg_rewind` without superuser on PostgreSQL 11 and newer (Alexander Kukushkin)

  If you want to use this feature please define `username` and `password` in the `postgresql.authentication.rewind` section of Patroni configuration file. For an already existing cluster you will have to create the user manually and `GRANT EXECUTE` permission on a few functions. You can find more details in the PostgreSQL documentation.

- Do a smart comparison of actual and desired `primary_conninfo` values on replicas (Alexander Kukushkin)

  It might help to avoid replica restart when you are converting an already existing primary-standby cluster to one managed by Patroni

- IPv6 support (Alexander Kukushkin)

  There were two major issues. Patroni REST API service was listening only on `0.0.0.0` and IPv6 IP addresses used in the `api_url` and `conn_url` were not properly quoted.

- Kerberos support (Ajith Vilas, Alexander Kukushkin)

  It makes possible using Kerberos authentication between Postgres nodes instead of defining passwords in Patroni configuration file

- Manage `pg_ident.conf` (Alexander Kukushkin)

  This functionality works similarly to `pg_hba.conf`: if the `postgresql.pg_ident` is defined in the config file or DCS, Patroni will write its value to `pg_ident.conf`, however, if `postgresql.parameters.ident_file` is defined, Patroni will assume that `pg_ident` is managed from outside and not update the file.

Improvements in REST API

- Added `/health` endpoint (Wilfried Roset)

  It will return an HTTP status code only if PostgreSQL is running

- Added `/read-only` and `/read-write` endpoints (Julien Riou)

  The `/read-only` endpoint enables reads balanced across replicas and the primary. The `/read-write` endpoint is an alias for `/primary`, `/leader` and `/master`.

- Use `SSLContext` to wrap the REST API socket (Julien Riou)

  Usage of `ssl.wrap_socket()` is deprecated and was still allowing soon-to-be-deprecated protocols like TLS 1.1.

Logging improvements

- Two-step logging (Alexander Kukushkin)

  All log messages are first written into the in-memory queue and later they are asynchronously flushed into the stderr or file from a separate thread. The maximum queue size is limited (configurable). If the limit is reached, Patroni will start losing logs, which is still better than blocking the HA loop.

- Enable debug logging for GET/OPTIONS API calls together with latency (Jan Tomsa)

  It will help with debugging of health-checks performed by HAProxy, Consul or other tooling that decides which node is the primary/replica.

- Log exceptions caught in Retry (Daniel Kucera)

  Log the final exception when either the number of attempts or the timeout were reached. It will hopefully help to debug some issues when communication to DCS fails.

Improvements in patronictl

- Enhance dialogues for scheduled switchover and restart (Rafia Sabih)

  Previously dialogues did not take into account scheduled actions and therefore were misleading.

- Check if config file exists (Wilfried Roset)

  Be verbose about configuration file when the given filename does not exists, instead of ignoring silently (which can lead to misunderstanding).

- Add fallback value for `EDITOR` (Wilfried Roset)

  When the `EDITOR` environment variable was not defined, `patronictl edit-config` was failing with *PatroniCtlException*. The new strategy is to try `editor` and than `vi`, which should be available on most systems.

Improvements in Consul support

- Allow to specify Consul consistency mode (Jan Tomsa)

  You can read more about consistency mode here.

- Reload Consul config on SIGHUP (Cameron Daniel Kucera, Alexander Kukushkin)

  It is especially useful when somebody is changing the value of `token`.

Bugfixes

- Fix corner case in switchover/failover (Sharoon Thomas)

  The variable `scheduled_at` may be undefined if REST API is not accessible and we are using DCS as a fallback.

- Open trust to localhost in `pg_hba.conf` during custom bootstrap (Alexander Kukushkin)

  Previously it was open only to unix_socket, which was causing a lot of errors: `FATAL: no pg_hba.conf entry for replication connection from host "127.0.0.1", user "replicator"`

- Consider synchronous node as healthy even when the former leader is ahead (Alexander Kukushkin)

  If the primary loses access to the DCS, it restarts Postgres in read-only, but it might happen that other nodes can still access the old primary via the REST API. Such a situation was causing the synchronous standby not to promote because the old primary was reporting WAL position ahead of the synchronous standby.

- Standby cluster bugfixes (Alexander Kukushkin)

  Make it possible to bootstrap a replica in a standby cluster when the standby_leader is not accessible and a few other minor fixes.

15.24 Version 1.5.6

New features

- Support work with etcd cluster via set of proxies (Alexander Kukushkin)

  It might happen that etcd cluster is not accessible directly but via set of proxies. In this case Patroni will not perform etcd topology discovery but just round-robin via proxy hosts. Behavior is controlled by *etcd.use_proxies*.

- Changed callbacks behavior when role on the node is changed (Alexander Kukushkin)

  If the role was changed from *master* or *standby_leader* to *replica* or from *replica* to *standby_leader*, *on_restart* callback will not be called anymore in favor of *on_role_change* callback.

- Change the way how we start postgres (Alexander Kukushkin)

  Use *multiprocessing.Process* instead of executing itself and *multiprocessing.Pipe* to transmit the postmaster pid to the Patroni process. Before that we were using pipes, what was leaving postmaster process with stdin closed.

Bug fixes

- Fix role returned by REST API for the standby leader (Alexander Kukushkin)

  It was incorrectly returning *replica* instead of *standby_leader*

- Wait for callback end if it could not be killed (Julien Tachoires)

  Patroni doesn't have enough privileges to terminate the callback script running under *sudo* what was cancelling the new callback. If the running script could not be killed, Patroni will wait until it finishes and then run the next callback.

- Reduce lock time taken by dcs.get_cluster method (Alexander Kukushkin)

  Due to the lock being held DCS slowness was affecting the REST API health checks causing false positives.

- Improve cleaning of PGDATA when *pg_wal/pg_xlog* is a symlink (Julien Tachoires)

  In this case Patroni will explicitly remove files from the target directory.

- Remove unnecessary usage of os.path.relpath (Ants Aasma)

  It depends on being able to resolve the working directory, what will fail if Patroni is started in a directory that is later unlinked from the filesystem.

- Do not enforce ssl version when communicating with Etcd (Alexander Kukushkin)

  For some unknown reason python3-etcd on debian and ubuntu are not based on the latest version of the package and therefore it enforces TLSv1 which is not supported by Etcd v3. We solved this problem on Patroni side.

15.25 Version 1.5.5

This version introduces the possibility of automatic reinit of the former master, improves patronictl list output and fixes a number of bugs.

New features

- Add support of *PATRONI_ETCD_PROTOCOL*, *PATRONI_ETCD_USERNAME* and *PATRONI_ETCD_PASSWORD* environment variables (Étienne M)

  Before it was possible to configure them only in the config file or as a part of *PATRONI_ETCD_URL*, which is not always convenient.

- Make it possible to automatically reinit the former master (Alexander Kukushkin)

  If the pg_rewind is disabled or can't be used, the former master could fail to start as a new replica due to diverged timelines. In this case, the only way to fix it is wiping the data directory and reinitializing. This behavior could be changed by setting *postgresql.remove_data_directory_on_diverged_timelines*. When it is set, Patroni will wipe the data directory and reinitialize the former master automatically.

- Show information about timelines in patronictl list (Alexander Kukushkin)

  It helps to detect stale replicas. In addition to that, *Host* will include ':{port}' if the port value isn't default or there is more than one member running on the same host.

- Create a headless service associated with the $SCOPE-config endpoint (Alexander Kukushkin)

  The "config" endpoint keeps information about the cluster-wide Patroni and Postgres configuration, history file, and last but the most important, it holds the *initialize* key. When the Kubernetes master node is restarted or upgraded, it removes endpoints without services. The headless service will prevent it from being removed.

Bug fixes

- Adjust the read timeout for the leader watch blocking query (Alexander Kukushkin)

  According to the Consul documentation, the actual response timeout is increased by a small random amount of additional wait time added to the supplied maximum wait time to spread out the wake up time of any concurrent requests. It adds up to *wait / 16* additional time to the maximum duration. In our case we are adding *wait / 15* or 1 second depending on what is bigger.

- Always use replication=1 when connecting via replication protocol to the postgres (Alexander Kukushkin)

  Starting from Postgres 10 the line in the pg_hba.conf with database=replication doesn't accept connections with the parameter replication=database.

- Don't write primary_conninfo into recovery.conf for wal-only standby cluster (Alexander Kukushkin)

  Despite not having neither *host* nor *port* defined in the *standby_cluster* config, Patroni was putting the *primary_conninfo* into the *recovery.conf*, which is useless and generating a lot of errors.

15.26 Version 1.5.4

This version implements flexible logging and fixes a number of bugs.

New features

- Improvements in logging infrastructure (Alexander Kukushkin, Lucas Capistrant, Alexander Anikin)

  Logging configuration could be configured not only from environment variables but also from Patroni config file. It makes it possible to change logging configuration in runtime by updating config and doing reload or sending SIGHUP to the Patroni process. By default Patroni writes logs to stderr, but now it becomes possible to write logs directly into the file and rotate when it reaches a certain size. In addition to that added support of custom dateformat and the possibility to fine-tune log level for each python module.

- Make it possible to take into account the current timeline during leader elections (Alexander Kukushkin)

  It could happen that the node is considering itself as a healthiest one although it is currently not on the latest known timeline. In some cases we want to avoid promoting of such node, which could be achieved by setting *check_timeline* parameter to *true* (default behavior remains unchanged).

- Relaxed requirements on superuser credentials

  Libpq allows opening connections without explicitly specifying neither username nor password. Depending on situation it relies either on pgpass file or trust authentication method in pg_hba.conf. Since pg_rewind is also using libpq, it will work the same way.

- Implemented possibility to configure Consul Service registration and check interval via environment variables (Alexander Kukushkin)

  Registration of service in Consul was added in the 1.5.0, but so far it was only possible to turn it on via patroni.yaml.

Stability Improvements

- Set archive_mode to off during the custom bootstrap (Alexander Kukushkin)

  We want to avoid archiving wals and history files until the cluster is fully functional. It really helps if the custom bootstrap involves pg_upgrade.

- Apply five seconds backoff when loading global config on start (Alexander Kukushkin)

  It helps to avoid hammering DCS when Patroni just starting up.

- Reduce amount of error messages generated on shutdown (Alexander Kukushkin)

  They were harmless but rather annoying and sometimes scary.

- Explicitly secure rw perms for recovery.conf at creation time (Lucas Capistrant)

  We don't want anybody except patroni/postgres user reading this file, because it contains replication user and password.

- Redirect HTTPServer exceptions to logger (Julien Riou)

  By default, such exceptions were logged on standard output messing with regular logs.

Bug fixes

- Removed stderr pipe to stdout on pg_ctl process (Cody Coons)

  Inheriting stderr from the main Patroni process allows all Postgres logs to be seen along with all patroni logs. This is very useful in a container environment as Patroni and Postgres logs may be consumed using standard tools (docker logs, kubectl, etc). In addition to that, this change fixes a bug with Patroni not being able to catch postmaster pid when postgres writing some warnings into stderr.

- Set Consul service check deregister timeout in Go time format (Pavel Kirillov)

  Without explicitly mentioned time unit registration was failing.

- Relax checks of standby_cluster cluster configuration (Dmitry Dolgov, Alexander Kukushkin)

  It was accepting only strings as valid values and therefore it was not possible to specify the port as integer and create_replica_methods as a list.

15.27 Version 1.5.3

Compatibility and bugfix release.

- Improve stability when running with python3 against zookeeper (Alexander Kukushkin)

  Change of *loop_wait* was causing Patroni to disconnect from zookeeper and never reconnect back.

- Fix broken compatibility with postgres 9.3 (Alexander Kukushkin)

  When opening a replication connection we should specify replication=1, because 9.3 does not understand replication='database'

- Make sure we refresh Consul session at least once per HA loop and improve handling of consul sessions exceptions (Alexander Kukushkin)

  Restart of local consul agent invalidates all sessions related to the node. Not calling session refresh on time and not doing proper handling of session errors was causing demote of the primary.

15.28 Version 1.5.2

Compatibility and bugfix release.

- Compatibility with kazoo-2.6.0 (Alexander Kukushkin)

  In order to make sure that requests are performed with an appropriate timeout, Patroni redefines create_connection method from python-kazoo module. The last release of kazoo slightly changed the way how create_connection method is called.

- Fix Patroni crash when Consul cluster loses the leader (Alexander Kukushkin)

  The crash was happening due to incorrect implementation of touch_member method, it should return boolean and not raise any exceptions.

15.29 Version 1.5.1

This version implements support of permanent replication slots, adds support of pgBackRest and fixes number of bugs.

New features

- Permanent replication slots (Alexander Kukushkin)

  Permanent replication slots are preserved on failover/switchover, that is, Patroni on the new primary will create configured replication slots right after doing promote. Slots could be configured with the help of *patronictl edit-config*. The initial configuration could be also done in the *bootstrap.dcs*.

- Add pgbackrest support (Yogesh Sharma)

  pgBackrest can restore in existing $PGDATA folder, this allows speedy restore as files which have not changed since last backup are skipped, to support this feature new parameter *keep_data* has been introduced. See *replica creation method* section for additional examples.

Bug fixes

- A few bugfixes in the "standby cluster" workflow (Alexander Kukushkin)

  Please see https://github.com/zalando/patroni/pull/823 for more details.

- Fix REST API health check when cluster management is paused and DCS is not accessible (Alexander Kukushkin)

  Regression was introduced in https://github.com/zalando/patroni/commit/90cf930036a9d5249265af15d2b787ec7517cf57

15.30 Version 1.5.0

This version enables Patroni HA cluster to operate in a standby mode, introduces experimental support for running on Windows, and provides a new configuration parameter to register PostgreSQL service in Consul.

New features

- Standby cluster (Dmitry Dolgov)

  One or more Patroni nodes can form a standby cluster that runs alongside the primary one (i.e. in another datacenter) and consists of standby nodes that replicate from the master in the primary cluster. All PostgreSQL nodes in the standby cluster are replicas; one of those replicas elects itself to replicate directly from the remote master, while the others replicate from it in a cascading manner. More detailed description of this feature and some configuration examples can be found at *here*.

- Register Services in Consul (Pavel Kirillov, Alexander Kukushkin)

  If *register_service* parameter in the consul *configuration* is enabled, the node will register a service with the name *scope* and the tag *master*, *replica* or *standby-leader*.

- Experimental Windows support (Pavel Golub)

  From now on it is possible to run Patroni on Windows, although Windows support is brand-new and hasn't received as much real-world testing as its Linux counterpart. We welcome your feedback!

Improvements in patronictl

- Add patronictl -k/–insecure flag and support for restapi cert (Wilfried Roset)

  In the past if the REST API was protected by the self-signed certificates *patronictl* would fail to verify them. There was no way to disable that verification. It is now possible to configure *patronictl* to skip the certificate verification altogether or provide CA and client certificates in the *ctl:* section of configuration.

- Exclude members with nofailover tag from patronictl switchover/failover output (Alexander Anikin)

  Previously, those members were incorrectly proposed as candidates when performing interactive switchover or failover via patronictl.

Stability improvements

- Avoid parsing non-key-value output lines of pg_controldata (Alexander Anikin)

  Under certain circuimstances pg_controldata outputs lines without a colon character. That would trigger an error in Patroni code that parsed pg_controldata output, hiding the actual problem; often such lines are emitted in a warning shown by pg_controldata before the regular output, i.e. when the binary major version does not match the one of the PostgreSQL data directory.

- Add member name to the error message during the leader election (Jan Mussler)

  During the leader election, Patroni connects to all known members of the cluster and requests their status. Such status is written to the Patroni log and includes the name of the member. Previously, if the member was not accessible, the error message did not indicate its name, containing only the URL.

- Immediately reserve the WAL position upon creation of the replication slot (Alexander Kukushkin)

  Starting from 9.6, *pg_create_physical_replication_slot* function provides an additional boolean parameter *immediately_reserve*. When it is set to *false*, which is also the default, the slot doesn't reserve the WAL position until it receives the first client connection, potentially losing some segments required by the client in a time window between the slot creation and the initial client connection.

- Fix bug in strict synchronous replication (Alexander Kukushkin)

  When running with *synchronous_mode_strict: true*, in some cases Patroni puts * into the *synchronous_standby_names*, changing the sync state for most of the replication connections to *potential*. Previously, Patroni couldn't pick a synchronous candidate under such curcuimstances, as it only considered those with the state *async*.

15.31 Version 1.4.6

Bug fixes and stability improvements

This release fixes a critical issue with Patroni API /master endpoint returning 200 for the non-master node. This is a reporting issue, no actual split-brain, but under certain circumstances clients might be directed to the read-only node.

- Reset is_leader status on demote (Alexander Kukushkin, Oleksii Kliukin)

  Make sure demoted cluster member stops responding with code 200 on the /master API call.

- Add new "cluster_unlocked" field to the API output (Dmitry Dolgov)

  This field indicates whether the cluster has the master running. It can be used when it is not possible to query any other node but one of the replicas.

15.32 Version 1.4.5

New features

- Improve logging when applying new postgres configuration (Don Seiler)

  Patroni logs changed parameter names and values.

- Python 3.7 compatibility (Christoph Berg)

  async is a reserved keyword in python3.7

- Set state to "stopped" in the DCS when a member is shut down (Tony Sorrentino)

  This shows the member state as "stopped" in "patronictl list" command.

- Improve the message logged when stale postmaster.pid matches a running process (Ants Aasma)

  The previous one was beyond confusing.

- Implement patronictl reload functionality (Don Seiler)

  Before that it was only possible to reload configuration by either calling REST API or by sending SIGHUP signal to the Patroni process.

- Take and apply some parameters from controldata when starting as a replica (Alexander Kukushkin)

  The value of *max_connections* and some other parameters set in the global configuration may be lower than the one actually used by the primary; when this happens, the replica cannot start and should be fixed manually. Patroni takes care of that now by reading and applying the value from *pg_controldata*, starting postgres and setting *pending_restart* flag.

- If set, use LD_LIBRARY_PATH when starting postgres (Chris Fraser)

  When starting up Postgres, Patroni was passing along PATH, LC_ALL and LANG env vars if they are set. Now it is doing the same with LD_LIBRARY_PATH. It should help if somebody installed PostgreSQL to non-standard place.

- Rename create_replica_method to create_replica_methods (Dmitry Dolgov)

  To make it clear that it's actually an array. The old name is still supported for backward compatibility.

Bug fixes and stability improvements

- Fix condition for the replica start due to pg_rewind in paused state (Oleksii Kliukin)

  Avoid starting the replica that had already executed pg_rewind before.

- Respond 200 to the master health-check only if update_lock has been successful (Alexander Kukushkin)

  Prevent Patroni from reporting itself a master on the former (demoted) master if DCS is partitioned.

- Fix compatibility with the new consul module (Alexander Kukushkin)

  Starting from v1.1.0 python-consul changed internal API and started using *list* instead of *dict* to pass query parameters.

- Catch exceptions from Patroni REST API thread during shutdown (Alexander Kukushkin)

  Those uncaught exceptions kept PostgreSQL running at shutdown.

- Do crash recovery only when Postgres runs as the master (Alexander Kukushkin)

  Require *pg_controldata* to report 'in production' or 'shutting down' or 'in crash recovery'. In all other cases no crash recovery is necessary.

- Improve handling of configuration errors (Henning Jacobs, Alexander Kukushkin)

  It is possible to change a lot of parameters in runtime (including *restapi.listen*) by updating Patroni config file and sending SIGHUP to Patroni process. This fix eliminates obscure exceptions from the 'restapi' thread when some of the parameters receive invalid values.

15.33 Version 1.4.4

Stability improvements

- Fix race condition in poll_failover_result (Alexander Kukushkin)

  It didn't affect directly neither failover nor switchover, but in some rare cases it was reporting success too early, when the former leader released the lock, producing a 'Failed over to "None"' instead of 'Failed over to "desired-node"' message.

- Treat Postgres parameter names as case insensitive (Alexander Kukushkin)

  Most of the Postgres parameters have snake_case names, but there are three exceptions from this rule: DateStyle, IntervalStyle and TimeZone. Postgres accepts those parameters when written in a different case (e.g. timezone = 'some/tzn'); however, Patroni was unable to find case-insensitive matches of those parameter names in pg_settings and ignored such parameters as a result.

- Abort start if attaching to running postgres and cluster not initialized (Alexander Kukushkin)

  Patroni can attach itself to an already running Postgres instance. It is imperative to start running Patroni on the master node before getting to the replicas.

- Fix behavior of patronictl scaffold (Alexander Kukushkin)

  Pass dict object to touch_member instead of json encoded string, DCS implementation will take care of encoding it.

- Don't demote master if failed to update leader key in pause (Alexander Kukushkin)

  During maintenance a DCS may start failing write requests while continuing to responds to read ones. In that case, Patroni used to put the Postgres master node to a read-only mode after failing to update the leader lock in DCS.

- Sync replication slots when Patroni notices a new postmaster process (Alexander Kukushkin)

  If Postgres has been restarted, Patroni has to make sure that list of replication slots matches its expectations.

- Verify sysid and sync replication slots after coming out of pause (Alexander Kukushkin)

  During the *maintenance* mode it may happen that data directory was completely rewritten and therefore we have to make sure that *Database system identifier* still belongs to our cluster and replication slots are in sync with Patroni expectations.

- Fix a possible failure to start not running Postgres on a data directory with postmaster lock file present (Alexander Kukushkin)

  Detect reuse of PID from the postmaster lock file. More likely to hit such problem if you run Patroni and Postgres in the docker container.

- Improve protection of DCS being accidentally wiped (Alexander Kukushkin)

  Patroni has a lot of logic in place to prevent failover in such case; it can also restore all keys back; however, until this change an accidental removal of /config key was switching off pause mode for 1 cycle of HA loop.

- Do not exit when encountering invalid system ID (Oleksii Kliukin)

  Do not exit when the cluster system ID is empty or the one that doesn't pass the validation check. In that case, the cluster most likely needs a reinit; mention it in the result message. Avoid terminating Patroni, as otherwise reinit cannot happen.

Compatibility with Kubernetes 1.10+

- Added check for empty subsets (Cody Coons)

  Kubernetes 1.10.0+ started returning *Endpoints.subsets* set to *None* instead of *[]*.

Bootstrap improvements

- Make deleting recovery.conf optional (Brad Nicholson)

  If *bootstrap.<custom_bootstrap_method_name>.keep_existing_recovery_conf* is defined and set to `True`, Patroni will not remove the existing `recovery.conf` file. This is useful when bootstrapping from a backup with tools like pgBackRest that generate the appropriate *recovery.conf* for you.

- Allow options to the basebackup built-in method (Oleksii Kliukin)

  It is now possible to supply options to the built-in basebackup method by defining the *basebackup* section in the configuration, similar to how those are defined for custom replica creation methods. The difference is in the format accepted by the *basebackup* section: since pg_basebackup accepts both *–key=value* and *–key* options, the contents of the section could be either a dictionary of key-value pairs, or a list of either one-element dictionaries or just keys (for the options that don't accept values). See *replica creation method* section for additional examples.

15.34 Version 1.4.3

Improvements in logging

- Make log level configurable from environment variables (Andy Newton, Keyvan Hedayati)

  PATRONI_LOGLEVEL - sets the general logging level *PATRONI_REQUESTS_LOGLEVEL* - sets the logging level for all HTTP requests e.g. Kubernetes API calls See *the docs for Python logging <https://docs.python.org/3.6/library/logging.html#levels>* to get the names of possible log levels

Stability improvements and bug fixes

- Don't rediscover etcd cluster topology when watch timed out (Alexander Kukushkin)

  If we have only one host in etcd configuration and exactly this host is not accessible, Patroni was starting discovery of cluster topology and never succeeding. Instead it should just switch to the next available node.

- Write content of bootstrap.pg_hba into a pg_hba.conf after custom bootstrap (Alexander Kukushkin)

  Now it behaves similarly to the usual bootstrap with *initdb*

- Single user mode was waiting for user input and never finish (Alexander Kukushkin)

  Regression was introduced in https://github.com/zalando/patroni/pull/576

15.35 Version 1.4.2

Improvements in patronictl

- Rename scheduled failover to scheduled switchover (Alexander Kukushkin)

  Failover and switchover functions were separated in version 1.4, but *patronictl list* was still reporting *Scheduled failover* instead of *Scheduled switchover*.

- Show information about pending restarts (Alexander Kukushkin)

  In order to apply some configuration changes sometimes it is necessary to restart postgres. Patroni was already giving a hint about that in the REST API and when writing node status into DCS, but there were no easy way to display it.

- Make show-config to work with cluster_name from config file (Alexander Kukushkin)

  It works similar to the *patronictl edit-config*

Stability improvements

- Avoid calling pg_controldata during bootstrap (Alexander Kukushkin)

  During initdb or custom bootstrap there is a time window when pgdata is not empty but pg_controldata has not been written yet. In such case pg_controldata call was failing with error messages.

- Handle exceptions raised from psutil (Alexander Kukushkin)

  cmdline is read and parsed every time when *cmdline()* method is called. It could happen that the process being examined has already disappeared, in that case *NoSuchProcess* is raised.

Kubernetes support improvements

- Don't swallow errors from k8s API (Alexander Kukushkin)

  A call to Kubernetes API could fail for a different number of reasons. In some cases such call should be retried, in some other cases we should log the error message and the exception stack trace. The change here will help debug Kubernetes permission issues.

- Update Kubernetes example Dockerfile to install Patroni from the master branch (Maciej Szulik)

  Before that it was using *feature/k8s*, which became outdated.

- Add proper RBAC to run patroni on k8s (Maciej Szulik)

  Add the Service account that is assigned to the pods of the cluster, the role that holds only the necessary permissions, and the rolebinding that connects the Service account and the Role.

15.36 Version 1.4.1

Fixes in patronictl

- Don't show current leader in suggested list of members to failover to. (Alexander Kukushkin)

  patronictl failover could still work when there is leader in the cluster and it should be excluded from the list of member where it is possible to failover to.

- Make patronictl switchover compatible with the old Patroni api (Alexander Kukushkin)

  In case if POST /switchover REST API call has failed with status code 501 it will do it once again, but to /failover endpoint.

15.37 Version 1.4

This version adds support for using Kubernetes as a DCS, allowing to run Patroni as a cloud-native agent in Kubernetes without any additional deployments of Etcd, Zookeeper or Consul.

Upgrade notice

Installing Patroni via pip will no longer bring in dependencies for (such as libraries for Etcd, Zookeper, Consul or Kubernetes, or support for AWS). In order to enable them one need to list them in pip install command explicitly, for instance *pip install patroni[etcd,kubernetes]*.

Kubernetes support

Implement Kubernetes-based DCS. The endpoints meta-data is used in order to store the configuration and the leader key. The meta-data field inside the pods definition is used to store the member-related data. In addition to using Endpoints, Patroni supports ConfigMaps. You can find more information about this feature in the *Kubernetes chapter of the documentation*

Stability improvements

- Factor out postmaster process into a separate object (Ants Aasma)

  This object identifies a running postmaster process via pid and start time and simplifies detection (and resolution) of situations when the postmaster was restarted behind our back or when postgres directory disappeared from the file system.

- Minimize the amount of SELECT's issued by Patroni on every loop of HA cylce (Alexander Kukushkin)

  On every iteration of HA loop Patroni needs to know recovery status and absolute wal position. From now on Patroni will run only single SELECT to get this information instead of two on the replica and three on the master.

- Remove leader key on shutdown only when we have the lock (Ants Aasma)

  Unconditional removal was generating unnecessary and misleading exceptions.

Improvements in patronictl

- Add version command to patronictl (Ants Aasma)

  It will show the version of installed Patroni and versions of running Patroni instances (if the cluster name is specified).

- Make optional specifying cluster_name argument for some of patronictl commands (Alexander Kukushkin, Ants Aasma)

  It will work if patronictl is using usual Patroni configuration file with the `scope` defined.

- Show information about scheduled switchover and maintenance mode (Alexander Kukushkin)

  Before that it was possible to get this information only from Patroni logs or directly from DCS.

- Improve `patronictl reinit` (Alexander Kukushkin)

  Sometimes `patronictl reinit` refused to proceed when Patroni was busy with other actions, namely trying to start postgres. *patronictl* didn't provide any commands to cancel such long running actions and the only (dangerous) workarond was removing a data directory manually. The new implementation of *reinit* forcefully cancells other long-running actions before proceeding with reinit.

- Implement `--wait` flag in `patronictl pause` and `patronictl resume` (Alexander Kukushkin)

  It will make `patronictl` wait until the requested action is acknowledged by all nodes in the cluster. Such behaviour is achieved by exposing the `pause` flag for every node in DCS and via the REST API.

- Rename `patronictl failover` into `patronictl switchover` (Alexander Kukushkin)

  The previous `failover` was actually only capable of doing a switchover; it refused to proceed in a cluster without the leader.

- Alter the behavior of `patronictl failover` (Alexander Kukushkin)

  It will work even if there is no leader, but in that case you will have to explicitly specify a node which should become the new leader.

Expose information about timeline and history

- Expose current timeline in DCS and via API (Alexander Kukushkin)

  Store information about the current timeline for each member of the cluster. This information is accessible via the API and is stored in the DCS

- Store promotion history in the /history key in DCS (Alexander Kukushkin)

  In addition, store the timeline history enriched with the timestamp of the corresponding promotion in the /history key in DCS and update it with each promote.

Add endpoints for getting synchronous and asynchronous replicas

- Add new /sync and /async endpoints (Alexander Kukushkin, Oleksii Kliukin)

  Those endpoints (also accessible as /synchronous and /asynchronous) return 200 only for synchronous and asynchronous replicas correspondingly (exclusing those marked as *noloadbalance*).

Allow multiple hosts for Etcd

- Add a new *hosts* parameter to Etcd configuration (Alexander Kukushkin)

  This parameter should contain the initial list of hosts that will be used to discover and populate the list of the running etcd cluster members. If for some reason during work this list of discovered hosts is exhausted (no available hosts from that list), Patroni will return to the initial list from the *hosts* parameter.

15.38 Version 1.3.6

Stability improvements

- Verify process start time when checking if postgres is running. (Ants Aasma)

  After a crash that doesn't clean up postmaster.pid there could be a new process with the same pid, resulting in a false positive for is_running(), which will lead to all kinds of bad behavior.

- Shutdown postgresql before bootstrap when we lost data directory (ainlolcat)

  When data directory on the master is forcefully removed, postgres process can still stay alive for some time and prevent the replica created in place of that former master from starting or replicating. The fix makes Patroni cache the postmaster pid and its start time and let it terminate the old postmaster in case it is still running after the corresponding data directory has been removed.

- Perform crash recovery in a single user mode if postgres master dies (Alexander Kukushkin)

  It is unsafe to start immediately as a standby and not possible to run `pg_rewind` if postgres hasn't been shut down cleanly. The single user crash recovery only kicks in if `pg_rewind` is enabled or there is no master at the moment.

Consul improvements

- Make it possible to provide datacenter configuration for Consul (Vilius Okockis, Alexander Kukushkin)

  Before that Patroni was always communicating with datacenter of the host it runs on.

- Always send a token in X-Consul-Token http header (Alexander Kukushkin)

  If `consul.token` is defined in Patroni configuration, we will always send it in the 'X-Consul-Token' http header. python-consul module tries to be "consistent" with Consul REST API, which doesn't accept token as a query parameter for session API, but it still works with 'X-Consul-Token' header.

- Adjust session TTL if supplied value is smaller than the minimum possible (Stas Fomin, Alexander Kukushkin)

  It could happen that the TTL provided in the Patroni configuration is smaller than the minimum one supported by Consul. In that case, Consul agent fails to create a new session. Without a session Patroni cannot create member and leader keys in the Consul KV store, resulting in an unhealthy cluster.

Other improvements

- Define custom log format via environment variable `PATRONI_LOGFORMAT` (Stas Fomin)

  Allow disabling timestamps and other similar fields in Patroni logs if they are already added by the system logger (usually when Patroni runs as a service).

15.39 Version 1.3.5

Bugfix

- Set role to 'uninitialized' if data directory was removed (Alexander Kukushkin)

  If the node was running as a master it was preventing from failover.

Stability improvement

- Try to run postmaster in a single-user mode if we tried and failed to start postgres (Alexander Kukushkin)

  Usually such problem happens when node running as a master was terminated and timelines were diverged. If `recovery.conf` has `restore_command` defined, there are really high chances that postgres will abort startup and leave controldata unchanged. It makes impossible to use `pg_rewind`, which requires a clean shutdown.

Consul improvements

- Make it possible to specify health checks when creating session (Alexander Kukushkin)

  If not specified, Consul will use "serfHealth". From one side it allows fast detection of isolated master, but from another side it makes it impossible for Patroni to tolerate short network lags.

Bugfix

- Fix watchdog on Python 3 (Ants Aasma)

  A misunderstanding of the ioctl() call interface. If mutable=False then fcntl.ioctl() actually returns the arg buffer back. This accidentally worked on Python2 because int and str comparison did not return an error. Error reporting is actually done by raising IOError on Python2 and OSError on Python3.

15.40 Version 1.3.4

Different Consul improvements

- Pass the consul token as a header (Andrew Colin Kissa)

  Headers are now the preferred way to pass the token to the consul API.

- Advanced configuration for Consul (Alexander Kukushkin)

  possibility to specify `scheme`, `token`, client and ca certificates *details*.

- compatibility with python-consul-0.7.1 and above (Alexander Kukushkin)

  new python-consul module has changed signature of some methods

- "Could not take out TTL lock" message was never logged (Alexander Kukushkin)

  Not a critical bug, but lack of proper logging complicates investigation in case of problems.

Quote synchronous_standby_names using quote_ident

- When writing `synchronous_standby_names` into the `postgresql.conf` its value must be quoted (Alexander Kukushkin)

  If it is not quoted properly, PostgreSQL will effectively disable synchronous replication and continue to work.

Different bugfixes around pause state, mostly related to watchdog (Alexander Kukushkin)

- Do not send keepalives if watchdog is not active
- Avoid activating watchdog in a pause mode
- Set correct postgres state in pause mode
- Do not try to run queries from API if postgres is stopped

15.41 Version 1.3.3

Bugfixes

- synchronous replication was disabled shortly after promotion even when synchronous_mode_strict was turned on (Alexander Kukushkin)
- create empty `pg_ident.conf` file if it is missing after restoring from the backup (Alexander Kukushkin)
- open access in `pg_hba.conf` to all databases, not only postgres (Franco Bellagamba)

15.42 Version 1.3.2

Bugfix

- patronictl edit-config didn't work with ZooKeeper (Alexander Kukushkin)

15.43 Version 1.3.1

Bugfix

- failover via API was broken due to change in `_MemberStatus` (Alexander Kukushkin)

15.44 Version 1.3

Version 1.3 adds custom bootstrap possibility, significantly improves support for pg_rewind, enhances the synchronous mode support, adds configuration editing to patronictl and implements watchdog support on Linux. In addition, this is the first version to work correctly with PostgreSQL 10.

Upgrade notice

There are no known compatibility issues with the new version of Patroni. Configuration from version 1.2 should work without any changes. It is possible to upgrade by installing new packages and either restarting Patroni (will cause PostgreSQL restart), or by putting Patroni into a *pause mode* first and then restarting Patroni on all nodes in the cluster (Patroni in a pause mode will not attempt to stop/start PostgreSQL), resuming from the pause mode at the end.

Custom bootstrap

- Make the process of bootstrapping the cluster configurable (Alexander Kukushkin)

  Allow custom bootstrap scripts instead of `initdb` when initializing the very first node in the cluster. The bootstrap command receives the name of the cluster and the path to the data directory. The resulting cluster can be configured to perform recovery, making it possible to bootstrap from a backup and do point in time recovery. Refer to the *documentaton page* for more detailed description of this feature.

Smarter pg_rewind support

- Decide on whether to run pg_rewind by looking at the timeline differences from the current master (Alexander Kukushkin)

  Previously, Patroni had a fixed set of conditions to trigger pg_rewind, namely when starting a former master, when doing a switchover to the designated node for every other node in the cluster or when there is a replica with the nofailover tag. All those cases have in common a chance that some replica may be ahead of the new master. In some cases, pg_rewind did nothing, in some other ones it was not running when necessary. Instead of relying on this limited list of rules make Patroni compare the master and the replica WAL positions (using the streaming replication protocol) in order to reliably decide if rewind is necessary for the replica.

Synchronous replication mode strict

- Enhance synchronous replication support by adding the strict mode (James Sewell, Alexander Kukushkin)

  Normally, when `synchronous_mode` is enabled and there are no replicas attached to the master, Patroni will disable synchronous replication in order to keep the master available for writes. The `synchronous_mode_strict` option changes that, when it is set Patroni will not disable the synchronous replication in a lack of replicas, effectively blocking all clients writing data to the master. In addition to the synchronous mode guarantee of preventing any data loss due to automatic failover, the strict mode ensures that each write is either durably stored on two nodes or not happening altogether if there is only one node in the cluster.

Configuration editing with patronictl

- Add configuration editing to patronictl (Ants Aasma, Alexander Kukushkin)

  Add the ability to patronictl of editing dynamic cluster configuration stored in DCS. Support either specifying the parameter/values from the command-line, invoking the $EDITOR, or applying configuration from the yaml file.

Linux watchdog support

- Implement watchdog support for Linux (Ants Aasma)

  Support Linux software watchdog in order to reboot the node where Patroni is not running or not responding (e.g because of the high load) The Linux software watchdog reboots the non-responsive node. It is possible to configure the watchdog device to use (*/dev/watchdog* by default) and the mode (on, automatic, off) from the watchdog section of the Patroni configuration. You can get more information from the *watchdog documentation*.

Add support for PostgreSQL 10

- Patroni is compatible with all beta versions of PostgreSQL 10 released so far and we expect it to be compatible with the PostgreSQL 10 when it will be released.

PostgreSQL-related minor improvements

- Define pg_hba.conf via the Patroni configuration file or the dynamic configuration in DCS (Alexander Kukushkin)

  Allow to define the contents of `pg_hba.conf` in the `pg_hba` sub-section of the `postgresql` section of the configuration. This simplifies managing `pg_hba.conf` on multiple nodes, as one needs to define it only ones in DCS instead of logging to every node, changing it manually and reload the configuration.

  When defined, the contents of this section will replace the current `pg_hba.conf` completely. Patroni ignores it if `hba_file` PostgreSQL parameter is set.

- Support connecting via a UNIX socket to the local PostgreSQL cluster (Alexander Kukushkin)

  Add the `use_unix_socket` option to the `postgresql` section of Patroni configuration. When set to true and the PostgreSQL `unix_socket_directories` option is not empty, enables Patroni to use the first value from it to connect to the local PostgreSQL cluster. If `unix_socket_directories` is not defined, Patroni will assume its default value and omit the `host` parameter in the PostgreSQL connection string altogether.

- Support change of superuser and replication credentials on reload (Alexander Kukushkin)

- Support storing of configuration files outside of PostgreSQL data directory (@jouir)

  Add the new configuration `postgresql` configuration directive `config_dir`. It defaults to the data directory and must be writable by Patroni.

Bug fixes and stability improvements

- Handle EtcdEventIndexCleared and EtcdWatcherCleared exceptions (Alexander Kukushkin)

  Faster recovery when the watch operation is ended by Etcd by avoiding useless retries.

- Remove error spinning on Etcd failure and reduce log spam (Ants Aasma)

  Avoid immediate retrying and emitting stack traces in the log on the second and subsequent Etcd connection failures.

- Export locale variables when forking PostgreSQL processes (Oleksii Kliukin)

  Avoid the *postmaster became multithreaded during startup* fatal error on non-English locales for PostgreSQL built with NLS.

- Extra checks when dropping the replication slot (Alexander Kukushkin)

  In some cases Patroni is prevented from dropping the replication slot by the WAL sender.

- Truncate the replication slot name to 63 (NAMEDATALEN - 1) characters to comply with PostgreSQL naming rules (Nick Scott)

- Fix a race condition resulting in extra connections being opened to the PostgreSQL cluster from Patroni (Alexander Kukushkin)

- Release the leader key when the node restarts with an empty data directory (Alex Kerney)

- Set asynchronous executor busy when running bootstrap without a leader (Alexander Kukushkin)

  Failure to do so could have resulted in errors stating the node belonged to a different cluster, as Patroni proceeded with the normal business while being bootstrapped by a bootstrap method that doesn't require a leader to be present in the cluster.

- Improve WAL-E replica creation method (Joar Wandborg, Alexander Kukushkin).

  - Use csv.DictReader when parsing WAL-E base backup, accepting ISO dates with space-delimited date and time.
  - Support fetching current WAL position from the replica to estimate the amount of WAL to restore. Previously, the code used to call system information functions that were available only on the master node.

15.45 Version 1.2

This version introduces significant improvements over the handling of synchronous replication, makes the startup process and failover more reliable, adds PostgreSQL 9.6 support and fixes plenty of bugs. In addition, the documentation, including these release notes, has been moved to https://patroni.readthedocs.io.

Synchronous replication

- Add synchronous replication support. (Ants Aasma)

  Adds a new configuration variable `synchronous_mode`. When enabled, Patroni will manage `synchronous_standby_names` to enable synchronous replication whenever there are healthy standbys available. When synchronous mode is enabled, Patroni will automatically fail over only to a standby that was synchronously replicating at the time of the master failure. This effectively means that no user visible transaction gets lost in such a case. See the *feature documentation* for the detailed description and implementation details.

Reliability improvements

- Do not try to update the leader position stored in the `leader optime` key when PostgreSQL is not 100% healthy. Demote immediately when the update of the leader key failed. (Alexander Kukushkin)
- Exclude unhealthy nodes from the list of targets to clone the new replica from. (Alexander Kukushkin)
- Implement retry and timeout strategy for Consul similar to how it is done for Etcd. (Alexander Kukushkin)
- Make `--dcs` and `--config-file` apply to all options in `patronictl`. (Alexander Kukushkin)
- Write all postgres parameters into postgresql.conf. (Alexander Kukushkin)

  It allows starting PostgreSQL configured by Patroni with just `pg_ctl`.
- Avoid exceptions when there are no users in the config. (Kirill Pushkin)
- Allow pausing an unhealthy cluster. Before this fix, `patronictl` would bail out if the node it tries to execute pause on is unhealthy. (Alexander Kukushkin)
- Improve the leader watch functionality. (Alexander Kukushkin)

  Previously the replicas were always watching the leader key (sleeping until the timeout or the leader key changes). With this change, they only watch when the replica's PostgreSQL is in the `running` state and not when it is stopped/starting or restarting PostgreSQL.
- Avoid running into race conditions when handling SIGCHILD as a PID 1. (Alexander Kukushkin)

  Previously a race condition could occur when running inside the Docker containers, since the same process inside Patroni both spawned new processes and handled SIGCHILD from them. This change uses fork/execs for Patroni and leaves the original PID 1 process responsible for handling signals from children.
- Fix WAL-E restore. (Oleksii Kliukin)

  Previously WAL-E restore used the `no_master` flag to avoid consulting with the master altogether, making Patroni always choose restoring from WAL over the `pg_basebackup`. This change reverts it to the original meaning of `no_master`, namely Patroni WAL-E restore may be selected as a replication method if the master is not running. The latter is checked by examining the connection string passed to the method. In addition, it makes the retry mechanism more robust and handles other minutia.

- Implement asynchronous DNS resolver cache. (Alexander Kukushkin)

  Avoid failing when DNS is temporary unavailable (for instance, due to an excessive traffic received by the node).

- Implement starting state and master start timeout. (Ants Aasma, Alexander Kukushkin)

  Previously `pg_ctl` waited for a timeout and then happily trodded on considering PostgreSQL to be running. This caused PostgreSQL to show up in listings as running when it was actually not and caused a race condition that resulted in either a failover, or a crash recovery, or a crash recovery interrupted by failover and a missed rewind. This change adds a `master_start_timeout` parameter and introduces a new state for the main HA loop: `starting`. When `master_start_timeout` is 0 we will failover immediately when the master crashes as soon as there is a failover candidate. Otherwise, Patroni will wait after attempting to start PostgreSQL on the master for the duration of the timeout; when it expires, it will failover if possible. Manual failover requests will be honored during the crash of the master even before the timeout expiration.

  Introduce the `timeout` parameter to the `restart` API endpoint and `patronictl`. When it is set and restart takes longer than the timeout, PostgreSQL is considered unhealthy and the other nodes becomes eligible to take the leader lock.

- Fix `pg_rewind` behavior in a pause mode. (Ants Aasma)

  Avoid unnecessary restart in a pause mode when Patroni thinks it needs to rewind but rewind is not possible (i.e. `pg_rewind` is not present). Fallback to default `libpq` values for the `superuser` (default OS user) if `superuser` authentication is missing from the `pg_rewind` related Patroni configuration section.

- Serialize callback execution. Kill the previous callback of the same type when the new one is about to run. Fix the issue of spawning zombie processes when running callbacks. (Alexander Kukushkin)

- Avoid promoting a former master when the leader key is set in DCS but update to this leader key fails. (Alexander Kukushkin)

  This avoids the issue of a current master continuing to keep its role when it is partitioned together with the minority of nodes in Etcd and other DCSs that allow "inconsistent reads".

Miscellaneous

- Add `post_init` configuration option on bootstrap. (Alejandro Martínez)

  Patroni will call the script argument of this option right after running `initdb` and starting up PostgreSQL for a new cluster. The script receives a connection URL with `superuser` and sets `PGPASSFILE` to point to the `.pgpass` file containing the password. If the script fails, Patroni initialization fails as well. It is useful for adding new users or creating extensions in the new cluster.

- Implement PostgreSQL 9.6 support. (Alexander Kukushkin)

  Use `wal_level = replica` as a synonym for `hot_standby`, avoiding pending_restart flag when it changes from one to another. (Alexander Kukushkin)

Documentation improvements

- Add a Patroni main loop workflow diagram. (Alejandro Martínez, Alexander Kukushkin)
- Improve README, adding the Helm chart and links to release notes. (Lauri Apple)
- Move Patroni documentation to `Read the Docs`. The up-to-date documentation is available at https://patroni.readthedocs.io. (Oleksii Kliukin)

  Makes the documentation easily viewable from different devices (including smartphones) and searchable.

- Move the package to the semantic versioning. (Oleksii Kliukin)

  Patroni will follow the major.minor.patch version schema to avoid releasing the new minor version on small but critical bugfixes. We will only publish the release notes for the minor version, which will include all patches.

15.46 Version 1.1

This release improves management of Patroni cluster by bring in pause mode, improves maintenance with scheduled and conditional restarts, makes Patroni interaction with Etcd or Zookeeper more resilient and greatly enhances patronictl.

Upgrade notice

When upgrading from releases below 1.0 read about changing of credentials and configuration format at 1.0 release notes.

Pause mode

- Introduce pause mode to temporary detach Patroni from managing PostgreSQL instance (Murat Kabilov, Alexander Kukushkin, Oleksii Kliukin).

  Previously, one had to send SIGKILL signal to Patroni to stop it without terminating PostgreSQL. The new pause mode detaches Patroni from PostgreSQL cluster-wide without terminating Patroni. It is similar to the maintenance mode in Pacemaker. Patroni is still responsible for updating member and leader keys in DCS, but it will not start, stop or restart PostgreSQL server in the process. There are a few exceptions, for instance, manual failovers, reinitializes and restarts are still allowed. You can read *a detailed description of this feature*.

In addition, patronictl supports new `pause` and `resume` commands to toggle the pause mode.

Scheduled and conditional restarts

- Add conditions to the restart API command (Oleksii Kliukin)

  This change enhances Patroni restarts by adding a couple of conditions that can be verified in order to do the restart. Among the conditions are restarting when PostgreSQL role is either a master or a replica, checking the PostgreSQL version number or restarting only when restart is necessary in order to apply configuration changes.

- Add scheduled restarts (Oleksii Kliukin)

  It is now possible to schedule a restart in the future. Only one scheduled restart per node is supported. It is possible to clear the scheduled restart if it is not needed anymore. A combination of scheduled and conditional restarts is supported, making it possible, for instance, to scheduled minor PostgreSQL upgrades in the night, restarting only the instances that are running the outdated minor version without adding postgres-specific logic to administration scripts.

- Add support for conditional and scheduled restarts to patronictl (Murat Kabilov).

  patronictl restart supports several new options. There is also patronictl flush command to clean the scheduled actions.

Robust DCS interaction

- Set Kazoo timeouts depending on the loop_wait (Alexander Kukushkin)

  Originally, ping_timeout and connect_timeout values were calculated from the negotiated session timeout. Patroni loop_wait was not taken into account. As a result, a single retry could take more time than the session timeout, forcing Patroni to release the lock and demote.

  This change set ping and connect timeout to half of the value of loop_wait, speeding up detection of connection issues and leaving enough time to retry the connection attempt before losing the lock.

- Update Etcd topology only after original request succeed (Alexander Kukushkin)

  Postpone updating the Etcd topology known to the client until after the original request. When retrieving the cluster topology, implement the retry timeouts depending on the known number of nodes in the Etcd cluster. This makes our client prefer to get the results of the request to having the up-to-date list of nodes.

  Both changes make Patroni connections to DCS more robust in the face of network issues.

Patronictl, monitoring and configuration

- Return information about streaming replicas via the API (Feike Steenbergen)

Previously, there was no reliable way to query Patroni about PostgreSQL instances that fail to stream changes (for instance, due to connection issues). This change exposes the contents of pg_stat_replication via the /patroni endpoint.

- Add patronictl scaffold command (Oleksii Kliukin)

  Add a command to create cluster structure in Etcd. The cluster is created with user-specified sysid and leader, and both leader and member keys are made persistent. This command is useful to create so-called master-less configurations, where Patroni cluster consisting of only replicas replicate from the external master node that is unaware of Patroni. Subsequently, one may remove the leader key, promoting one of the Patroni nodes and replacing the original master with the Patroni-based HA cluster.

- Add configuration option `bin_dir` to locate PostgreSQL binaries (Ants Aasma)

  It is useful to be able to specify the location of PostgreSQL binaries explicitly when Linux distros that support installing multiple PostgreSQL versions at the same time.

- Allow configuration file path to be overridden using `custom_conf` of (Alejandro Martínez)

  Allows for custom configuration file paths, which will be unmanaged by Patroni, *details*.

Bug fixes and code improvements

- Make Patroni compatible with new version schema in PostgreSQL 10 and above (Feike Steenbergen)

  Make sure that Patroni understand 2-digits version numbers when doing conditional restarts based on the PostgreSQL version.

- Use pkgutil to find DCS modules (Alexander Kukushkin)

  Use the dedicated python module instead of traversing directories manually in order to find DCS modules.

- Always call on_start callback when starting Patroni (Alexander Kukushkin)

  Previously, Patroni did not call any callbacks when attaching to the already running node with the correct role. Since callbacks are often used to route client connections that could result in the failure to register the running node in the connection routing scheme. With this fix, Patroni calls on_start callback even when attaching to the already running node.

- Do not drop active replication slots (Murat Kabilov, Oleksii Kliukin)

  Avoid dropping active physical replication slots on master. PostgreSQL cannot drop such slots anyway. This change makes possible to run non-Patroni managed replicas/consumers on the master.

- Close Patroni connections during start of the PostgreSQL instance (Alexander Kukushkin)

  Forces Patroni to close all former connections when PostgreSQL node is started. Avoids the trap of reusing former connections if postmaster was killed with SIGKILL.

- Replace invalid characters when constructing slot names from member names (Ants Aasma)

  Make sure that standby names that do not comply with the slot naming rules don't cause the slot creation and standby startup to fail. Replace the dashes in the slot names with underscores and all other characters not allowed in slot names with their unicode codepoints.

15.47 Version 1.0

This release introduces the global dynamic configuration that allows dynamic changes of the PostgreSQL and Patroni configuration parameters for the entire HA cluster. It also delivers numerous bugfixes.

Upgrade notice

When upgrading from v0.90 or below, always upgrade all replicas before the master. Since we don't store replication credentials in DCS anymore, an old replica won't be able to connect to the new master.

Dynamic Configuration

- Implement the dynamic global configuration (Alexander Kukushkin)

  Introduce new REST API endpoint /config to provide PostgreSQL and Patroni configuration parameters that should be set globally for the entire HA cluster (master and all the replicas). Those parameters are set in DCS and in many cases can be applied without disrupting PostgreSQL or Patroni. Patroni sets a special flag called "pending restart" visible via the API when some of the values require the PostgreSQL restart. In that case, restart should be issued manually via the API.

  Patroni SIGHUP or POST to /reload will make it re-read the configuration file.

  See the *Patroni configuration* for the details on which parameters can be changed and the order of processing difference configuration sources.

  The configuration file format *has changed* since the v0.90. Patroni is still compatible with the old configuration files, but in order to take advantage of the bootstrap parameters one needs to change it. Users are encourage to update them by referring to the *dynamic configuration documentation page*.

More flexible configuration*

- Make postgresql configuration and database name Patroni connects to configurable (Misja Hoebe)

  Introduce *database* and *config_base_name* configuration parameters. Among others, it makes possible to run Patroni with PipelineDB and other PostgreSQL forks.

- Implement possibility to configure some Patroni configuration parameters via environment (Alexander Kukushkin)

  Those include the scope, the node name and the namespace, as well as the secrets and makes it easier to run Patroni in a dynamic environment, i.e. Kubernetes Please, refer to the *supported environment variables* for further details.

- Update the built-in Patroni docker container to take advantage of environment-based configuration (Feike Steenbergen).
- Add Zookeeper support to Patroni docker image (Alexander Kukushkin)
- Split the Zookeeper and Exhibitor configuration options (Alexander Kukushkin)
- Make patronictl reuse the code from Patroni to read configuration (Alexander Kukushkin)

  This allows patronictl to take advantage of environment-based configuration.

- Set application name to node name in primary_conninfo (Alexander Kukushkin)

  This simplifies identification and configuration of synchronous replication for a given node.

Stability, security and usability improvements

- Reset sysid and do not call pg_controldata when restore of backup in progress (Alexander Kukushkin)

  This change reduces the amount of noise generated by Patroni API health checks during the lengthy initialization of this node from the backup.

- Fix a bunch of pg_rewind corner-cases (Alexander Kukushkin)

  Avoid running pg_rewind if the source cluster is not the master.

  In addition, avoid removing the data directory on an unsuccessful rewind, unless the new parameter *remove_data_directory_on_rewind_failure* is set to true. By default it is false.

- Remove passwords from the replication connection string in DCS (Alexander Kukushkin)

  Previously, Patroni always used the replication credentials from the Postgres URL in DCS. That is now changed to take the credentials from the patroni configuration. The secrets (replication username and password) and no longer exposed in DCS.

- Fix the asynchronous machinery around the demote call (Alexander Kukushkin)

  Demote now runs totally asynchronously without blocking the DCS interactions.

- Make patronictl always send the authorization header if it is configured (Alexander Kukushkin)

  This allows patronictl to issue "protected" requests, i.e. restart or reinitialize, when Patroni is configured to require authorization on those.

- Handle the SystemExit exception correctly (Alexander Kukushkin)

  Avoids the issues of Patroni not stopping properly when receiving the SIGTERM

- Sample haproxy templates for confd (Alexander Kukushkin)

  Generates and dynamically changes haproxy configuration from the patroni state in the DCS using confide

- Improve and restructure the documentation to make it more friendly to the new users (Lauri Apple)

- API must report role=master during pg_ctl stop (Alexander Kukushkin)

  Makes the callback calls more reliable, particularly in the cluster stop case. In addition, introduce the *pg_ctl_timeout* option to set the timeout for the start, stop and restart calls via the *pg_ctl*.

- Fix the retry logic in etcd (Alexander Kukushkin)

  Make retries more predictable and robust.

- Make Zookeeper code more resilient against short network hiccups (Alexander Kukushkin)

  Reduce the connection timeouts to make Zookeeper connection attempts more frequent.

15.48 Version 0.90

This releases adds support for Consul, includes a new *noloadbalance* tag, changes the behavior of the *clonefrom* tag, improves *pg_rewind* handling and improves *patronictl* control program.

Consul support

- Implement Consul support (Alexander Kukushkin)

  Patroni runs against Consul, in addition to Etcd and Zookeeper. the connection parameters can be configured in the YAML file.

New and improved tags

- Implement *noloadbalance* tag (Alexander Kukushkin)

  This tag makes Patroni always return that the replica is not available to the load balancer.

- Change the implementation of the *clonefrom* tag (Alexander Kukushkin)

  Previously, a node name had to be supplied to the *clonefrom*, forcing a tagged replica to clone from the specific node. The new implementation makes *clonefrom* a boolean tag: if it is set to true, the replica becomes a candidate for other replicas to clone from it. When multiple candidates are present, the replicas picks one randomly.

Stability and security improvements

- Numerous reliability improvements (Alexander Kukushkin)

  Removes some spurious error messages, improves the stability of the failover, addresses some corner cases with reading data from DCS, shutdown, demote and reattaching of the former leader.

- Improve systems script to avoid killing Patroni children on stop (Jan Keirse, Alexander Kukushkin)

  Previously, when stopping Patroni, *systemd* also sent a signal to PostgreSQL. Since Patroni also tried to stop PostgreSQL by itself, it resulted in sending to different shutdown requests (the smart shutdown, followed by the fast shutdown). That resulted in replicas disconnecting too early and a former master not being able to rejoin after demote. Fix by Jan with prior research by Alexander.

- Eliminate some cases where the former master was unable to call pg_rewind before rejoining as a replica (Oleksii Kliukin)

  Previously, we only called *pg_rewind* if the former master had crashed. Change this to always run pg_rewind for the former master as long as pg_rewind is present in the system. This fixes the case when the master is shut down before the replicas managed to get the latest changes (i.e. during the "smart" shutdown).

- Numerous improvements to unit- and acceptance- tests, in particular, enable support for Zookeeper and Consul (Alexander Kukushkin).

- Make Travis CI faster and implement support for running tests against Zookeeper (Exhibitor) and Consul (Alexander Kukushkin)

  Both unit and acceptance tests run automatically against Etcd, Zookeeper and Consul on each commit or pull-request.

- Clear environment variables before calling PostgreSQL commands from Patroni (Feike Steenbergen)

  This prevents a possibility of reading system environment variables by connecting to the PostgreSQL cluster managed by Patroni.

Configuration and control changes

- Unify patronictl and Patroni configuration (Feike Steenbergen)

  patronictl can use the same configuration file as Patroni itself.

- Enable Patroni to read the configuration from the environment variables (Oleksii Kliukin)

  This simplifies generating configuration for Patroni automatically, or merging a single configuration from different sources.

- Include database system identifier in the information returned by the API (Feike Steenbergen)

- Implement *delete_cluster* for all available DCSs (Alexander Kukushkin)

  Enables support for DCSs other than Etcd in patronictl.

15.49 Version 0.80

This release adds support for *cascading replication* and simplifies Patroni management by providing *scheduled failovers*. One may use older versions of Patroni (in particular, 0.78) combined with this one in order to migrate to the new release. Note that the scheduled failover and cascading replication related features will only work with Patroni 0.80 and above.

Cascading replication

- Add support for the *replicatefrom* and *clonefrom* tags for the patroni node (Oleksii Kliukin).

  The tag *replicatefrom* allows a replica to use an arbitrary node a source, not necessary the master. The *clonefrom* does the same for the initial backup. Together, they enable Patroni to fully support cascading replication.

- Add support for running replication methods to initialize the replica even without a running replication connection (Oleksii Kliukin).

  This is useful in order to create replicas from the snapshots stored on S3 or FTP. A replication method that does not require a running replication connection should supply *no_master: true* in the yaml configuration. Those scripts will still be called in order if the replication connection is present.

Patronictl, API and DCS improvements

- Implement scheduled failovers (Feike Steenbergen).

  Failovers can be scheduled to happen at a certain time in the future, using either patronictl, or API calls.

- Add support for *dbuser* and *password* parameters in patronictl (Feike Steenbergen).
- Add PostgreSQL version to the health check output (Feike Steenbergen).
- Improve Zookeeper support in patronictl (Oleksandr Shulgin)
- Migrate to python-etcd 0.43 (Alexander Kukushkin)

Configuration

- Add a sample systems configuration script for Patroni (Jan Keirse).
- Fix the problem of Patroni ignoring the superuser name specified in the configuration file for DB connections (Alexander Kukushkin).
- Fix the handling of CTRL-C by creating a separate session ID and process group for the postmaster launched by Patroni (Alexander Kukushkin).

Tests

- Add acceptance tests with *behave* in order to check real-world scenarios of running Patroni (Alexander Kukushkin, Oleksii Kliukin).

  The tests can be launched manually using the *behave* command. They are also launched automatically for pull requests and after commits.

  Release notes for some older versions can be found on project's github page.

CHAPTER

SIXTEEN

CONTRIBUTING

Resources and information for developers can be found in the pages below.

16.1 Contributing guidelines

Wanna contribute to Patroni? Yay - here is how!

16.1.1 Chatting

Just want to chat with other Patroni users? Looking for interactive troubleshooting help? Join us on channel #patroni in the PostgreSQL Slack.

16.1.2 Running tests

Requirements for running behave tests:

1. PostgreSQL packages need to be installed.
2. PostgreSQL binaries must be available in your *PATH*. You may need to add them to the path with something like *PATH=/usr/lib/postgresql/11/bin:$PATH python -m behave*.
3. If you'd like to test with external DCSs (e.g., Etcd, Consul, and Zookeeper) you'll need the packages installed and respective services running and accepting unencrypted/unprotected connections on localhost and default port. In the case of Etcd or Consul, the behave test suite could start them up if binaries are available in the *PATH*.

Install dependencies:

```
# You may want to use Virtualenv or specify pip3.
pip install -r requirements.txt
pip install -r requirements.dev.txt
```

After you have all dependencies installed, you can run the various test suites:

```
# 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
```

(continues on next page)

(continued from previous page)

```
# Run the behave (https://behave.readthedocs.io/en/latest/) test suite in features/;
# modify DCS as desired (raft has no dependencies so is the easiest to start with):
DCS=raft python -m behave
```

16.1.3 Testing with tox

To run tox tests you only need to install one dependency (other than Python)

```
pip install tox>=4
```

If you wish to run *behave* tests then you also need docker installed.

Tox configuration in *tox.ini* has "environments" to run the following tasks:

- lint: Python code lint with *flake8*
- test: unit tests for all available python interpreters with *pytest*, generates XML reports or HTML reports if a TTY is detected
- dep: detect package dependency conflicts using *pipdeptree*
- type: static type checking with *pyright*
- black: code formatting with *black*
- docker-build: build docker image used for the *behave* env
- docker-cmd: run arbitrary command with the above image
- docker-behave-etcd: run tox for behave tests with above image
- py*behave: run behave with available python interpreters (without docker, although this is what is called inside docker containers)
- docs: build docs with *sphinx*

Running tox

To run the default env list; dep, lint, test, and docs, just run:

```
tox
```

The *test* envs can be run with the label *test*:

```
tox -m test
```

The *behave* docker tests can be run with the label *behave*:

```
tox -m behave
```

Similarly, docs has the label *docs*.

All other envs can be run with their respective env names:

```
tox -e lint
tox -e py39-test-lin
```

It is also possible to select partial env lists using *factors*. For example, if you want to run all envs for python 3.10:

```
tox -f py310
```

This is equivalent to running all the envs listed below:

```
$ 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
```

You can list all configured combinations of environments with tox (>=v4) like so

```
tox l
```

The envs *test* and *docs* will attempt to open the HTML output files when the job completes, if tox is run with an active terminal. This is intended to be for benefit of the developer running this env locally. It will attempt to run *open* on a mac and *xdg-open* on Linux. To use a different command set the env var *OPEN_CMD* to the name or path of the command. If this step fails it will not fail the run overall. If you want to disable this facility set the env var *OPEN_CMD* to the *:* no-op command.

```
OPEN_CMD=: tox -m docs
```

Behave tests

Behave tests with *-m behave* will build docker images based on PG_MAJOR version 11 through 15 and then run all behave tests. This can take quite a long time to run so you might want to limit the scope to a select version of Postgres or to a specific feature set or steps.

To specify the version of postgres include the full name of the dependent image build env that you want and then the behave env name. For instance if you want Postgres 15 use:

```
tox -e pg14-docker-build,pg14-docker-behave-etcd-lin
```

If on the other hand you want to test a specific feature you can pass positional arguments to behave. This will run the watchdog behave feature test scenario with all versions of Postgres.

```
tox -m behave -- features/watchdog.feature
```

Of course you can combine the two.

16.1.4 Reporting issues

If you have a question about patroni or have a problem using it, please read the *README* before filing an issue. Also double check with the current issues on our Issues Tracker.

16.1.5 Contributing a pull request

1) Submit a comment to the relevant issue or create a new issue describing your proposed change.
2) Do a fork, develop and test your code changes.
3) Include documentation
4) Submit a pull request.

You'll get feedback about your pull request as soon as possible.

Happy Patroni hacking ;-)

CHAPTER

SEVENTEEN

INDICES AND TABLES

- genindex
- search