Patroni Documentation

Release 1.4.4

Zalando SE

Jul 09, 2019

CONTENTS:

Patroni is a template for you to create your own customized, high-availability solution using Python and - for maximum accessibility - a distributed configuration store like ZooKeeper, etcd, Consul or Kubernetes. Database engineers, DBAs, DevOps engineers, and SREs who are looking to quickly deploy HA PostgreSQL in the datacenter-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.

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 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
```

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 *python-etcd* module in order to use Etcd as 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

aws *boto* 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 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
> ./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
```

1.4 YAML Configuration

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

1.5 Environment Configuration

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

1.6 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.

1.7 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.

CHAPTER

TWO

PATRONI CONFIGURATION

Patroni configuration is stored in the DCS (Distributed Configuration Store). There are 3 types of configuration:

- **Dynamic configuration.** These options can be set in DCS at any time. 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 options 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 also indicates this, by showing `"restart_pending": true`.
- **Local *configuration* (patroni.yml).** These options are defined in the configuration file and take precedence over dynamic configuration. patroni.yml could be changed and reload in runtime (without restart of Patroni) by sending SIGHUP to the Patroni process or by performing `POST /reload` REST-API request.
- **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`).

Some of the PostgreSQL parameters must hold the same values on the master 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
- wal_log_hints: on
- track_commit_timestamp: off

For the parameters below, PostgreSQL does not require equal values among the master and all the replicas. However, considering the possibility of a replica to become the master 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

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

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

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).

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

- ttl: 30
- loop_wait: 10
- retry_timeouts: 10
- maximum_lag_on_failover: 1048576
- 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 master 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

REST API

We provide a REST API endpoint for working with dynamic configuration.

3.1 GET /config

Get current version of dynamic configuration.

```
$ curl -s 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_log_hints": "on",
      "wal_keep_segments": 8,
      "wal_level": "hot_standby",
      "max_wal_senders": 5,
      "max_replication_slots": 5,
      "max_connections": "100"
    }
  }
}
```

3.2 PATCH /config

Change 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,
```

(continues on next page)

(continued from previous page)

```
  "retry_timeout": 10,
  "postgresql": {
    "use_slots": true,
    "use_pg_rewind": true,
    "parameters": {
      "hot_standby": "on",
      "wal_log_hints": "on",
      "wal_keep_segments": 8,
      "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 "restart_pending" 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_keep_segments": 8,
```

(continues on next page)

(continued from previous page)

```
      "wal_level": "hot_standby",
      "wal_log_hints": "on",
      "max_wal_senders": 5,
      "max_replication_slots": 5
    }
  }
}
```

Above call removes `postgresql.parameters.max_connections` from the dynamic configuration.

3.3 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_log_hints":
↪"on","wal_keep_segments":8,"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_keep_segments": 8,
      "wal_level": "hot_standby",
      "wal_log_hints": "on",
      "max_wal_senders": 5
    },
    "use_pg_rewind": true
  },
  "loop_wait": 3
}
```

CHAPTER

FOUR

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.

4.1 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
- **PATRONI_LOGLEVEL**: sets the general logging level (see the docs for Python logging)
- **PATRONI_REQUESTS_LOGLEVEL**: sets the logging level for all HTTP requests e.g. Kubernetes API calls (see the docs for Python logging)

4.2 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.

4.3 Consul

- **PATRONI_CONSUL_HOST**: the host:port for the Consul endpoint.

- **PATRONI_CONSUL_URL**: url for the Consul, 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_CHECKS**: (optional) list of Consul health checks used for the session. If not specified Consul will use "serfHealth" in additional to the TTL based check created by Patroni. Additional checks, in particular the "serfHealth", may cause the leader lock to expire faster than in *ttl* seconds when the leader instance becomes unavailable.

4.4 Etcd

- **PATRONI_ETCD_HOST**: the host:port for the etcd endpoint.
- **PATRONI_ETCD_HOSTS**: list of etcd endpoints in format host1:port1,host2:port2,etc...
- **PATRONI_ETCD_URL**: url for the etcd, in format: http(s)://(username:password@)host:port
- **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_SRV**: Domain to search the SRV record(s) for cluster autodiscovery.
- **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.

4.5 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.

4.6 Kubernetes

- **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 Postgres role (*master* or *replica*). Patroni will set this label on the pod it is running in. Default value is *role*.
- **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.

4.7 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_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, pg_rewind, pg_basebackup, postgres) The default value is an empty string meaning that PATH environment variable will be used to find the executables.
- **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 master via streaming replication
- **PATRONI_REPLICATION_PASSWORD**: replication password; the user will be created during initialization.
- **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).

4.8 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.

4.9 ZooKeeper

- **PATRONI_ZOOKEEPER_HOSTS**: comma separated list of ZooKeeper cluster members: "'host1:port1','host2:port2','etc...'". It is important to quote every single entity!

CHAPTER

FIVE

YAML CONFIGURATION SETTINGS

5.1 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

5.2 Bootstrap configuration

- **dcs: This section will be written into /<*namespace*>/<*scope*>/*config* of a given configuration store after initializing of new c**
  - **loop_wait**: the number of seconds the loop will sleep. Default value: 10
  - **ttl**: the TTL to acquire the leader lock. 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. 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.
  - **master_start_timeout**: the amount of time a master is allowed to recover from failures before failover is triggered. 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. Best worst case failover time for master failure is: loop_wait + master_start_timeout + loop_wait, unless master_start_timeout is zero, in which case it's just loop_wait. Set the value according to your durability/availability tradeoff.
  - **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.
  - **postgresql:**
    - **use_pg_rewind**: whether or not to use pg_rewind
    - **use_slots**: whether or not to use replication_slots. Must be False for PostgreSQL 9.3. You should comment out max_replication_slots before it becomes ineligible for leader status.

        - **recovery_conf**: additional configuration settings written to recovery.conf when configuring follower.
        - **parameters**: list of configuration settings for Postgres. Many of these are required for replication to work.

- **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: 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.
- **pg_hba: list of lines that you should add to 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.
- **users: Some additional users which need to be created after initializing new cluster**
    - **admin: the name of user**
        - **password: zalando**:
        - **options: list of options for CREATE USER statement**
            - **- createrole**
            - **- createdb**
- **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.

5.3 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 endpoint, in format: http(s)://host:port
- **url**: url for the Consul endpoint
- **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.

- **checks**: (optional) list of Consul health checks used for the session. If not specified Consul will use "serfHealth" in additional to the TTL based check created by Patroni. Additional checks, in particular the "serfHealth", may cause the leader lock to expire faster than in *ttl* seconds when the leader instance becomes unavailable

5.4 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.
- **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.
- **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**.

5.5 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.

5.6 Kubernetes

- **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). Patroni will set this label on the pod it runs in. Default value is `role`.
- **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.

5.7 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).
  - **replication:**
    - **username**: replication username; the user will be created during initialization. Replicas will use this user to access master via streaming replication
    - **password**: replication password; the user will be created during initialization.
- **callbacks: callback scripts to run on certain actions. Patroni will pass the action, role and cluster name. (See scripts/aws.p**
  - **on_reload**: run this script when configuration reload is triggered.
  - **on_restart**: run this script when the cluster restarts.
  - **on_role_change**: run this script when the cluster is being promoted or demoted.
  - **on_start**: run this script when the cluster starts.
  - **on_stop**: run this script when the cluster stops.
- **connect_address**: IP address + port through which Postgres is accessible from other nodes and applications.
- **create_replica_method**: 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**: Path to PostgreSQL binaries (pg_ctl, pg_rewind, pg_basebackup, postgres). The default value is an empty string meaning that PATH environment variable will be used to find the executables.
- **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 definded, Patroni will use 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 default value should be used and omit `host` from 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`. This parameter has higher priority than `bootstra`**
  - **- 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_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 postgres data directory and recreate replica. Otherwise it will try to follow the new leader. Default value is **false**.
- **replica_method**: for each create_replica_method 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".

5.8 REST API

- **connect_address**: IP address and port to access the REST API.
- **listen**: IP address and port that Patroni will listen to, to provide health-check information for HAProxy.
- **Optional:**
  - **authentication:**
    - **username**: Basic-auth username to protect unsafe REST API endpoints.
    - **password**: Basic-auth password to protect unsafe REST API endpoints.
  - **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.
  - **keyfile**: Specifies the file with the secret key in the PEM format.

5.9 ZooKeeper

- **hosts**: list of ZooKeeper cluster members in format: ['host1:port1', 'host2:port2', 'etc...'].

5.10 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.

CHAPTER

SIX

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.

6.1 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
        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

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

6.2 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 master 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_method:
        - wal_e
        - basebackup
    wal_e:
        command: patroni_wale_restore
        no_master: 1
        envdir: {{WALE_ENV_DIR}}
        use_iam: 1
    basebackup:
        max-rate: '100M'
```

The create_replica_method 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 (master or other replica). The user in the connection string can execute SQL and replication protocol commands.

A special no_master parameter, if defined, allows Patroni to call the replica creation method even if there is no running master 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 basebackup method is a special case: it will be used if create_replica_method is empty, although it is possible to list it explicitly among the create_replica_method methods. This method initializes a new replica with the pg_basebackup, the base backup is taken from the master 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. 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'
```

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

CHAPTER

SEVEN

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.

7.1 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.

7.2 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:

```
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.

[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.

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.

7.3 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.

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.

7.4 Synchronous mode implementation

When in synchronous mode Patroni maintains synchronization state in the DCS, containing the latest primary and current synchronous standby. 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 ever assign one standby to `synchronous_standby_names` because with multiple candidates it is not possible to know which node was acting as synchronous during the failure.

On each HA loop iteration Patroni re-evaluates synchronous standby choice. If the current synchronous standby is 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.

[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.

CHAPTER

EIGHT

PAUSE/RESUME MODE FOR THE CLUSTER

8.1 The goal

Under certain circumstances Patroni needs to temporary 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 the reason unknown to Patroni, some nodes can be even temporary promoted, violating the assumption of running only one master. Therefore, Patroni needs to be able to "detach" from the running cluster, implementing an equivalent of the maintenance mode in Pacemaker.

8.2 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 master with the leader lock Patroni updates the lock. If the node with the leader lock stops being the master (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 master node.
- If 'parallel' masters are detected by Patroni, it emits a warning, but does not demote the masters without the leader lock.
- If there is no leader lock in the cluster, the running master acquires the lock. If there is more than one master node, then the first master to acquire the lock wins. If there are no masters 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 master 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.

8.3 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

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.

There are two ways to direct the traffic to the Postgres master:

- use the callback script provided by Patroni
- 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.

9.4 Examples

- The kubernetes folder of the Patroni repository contains examples of the Docker image, the Kubernetes manifest and the callback script in order 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

WATCHDOG SUPPORT

Having multiple PostgreSQL servers running as master 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 master. If watchdog activation fails and watchdog mode is `required` then the node will refuse to become master. 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.

10.1 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

ELEVEN

RELEASE NOTES

11.1 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)

  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)

  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)

  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)

  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)

  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)

  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)

  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)

  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)

  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.

11.2 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)

  Now it behaves similarly to the usual bootstrap with *initdb*

- Single user mode was waiting for user input and never finish (Alexander)

  Regression was introduced in https://github.com/zalando/patroni/pull/576

11.3 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)

  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)

  It works similar to the *patronictl edit-config*

Stability improvements

- Avoid calling pg_controldata during bootstrap (Alexander)

  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)

  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)

  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)

  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.

11.4 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)

  In case if POST /switchover REST API call has failed with status code 501 it will do it once again, but to /failover endpoint.

11.5 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 explicitely, 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)

  Unconditional removal was generating unnecessary and missleading exceptions.

Improvements in patronictl

- Add version command to patronictl (Ants)

  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, Ants)

  It will work if patronictl is using usual Patroni configuration file with the `scope` defined.

- Show information about scheduled switchover and maintenance mode (Alexander)

  Before that it was possible to get this information only from Patroni logs or directly from DCS.

- Improve `patronictl reinit` (Alexander)

  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)

  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)

  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)

  It will work even if there is no leader, but in that case you will have to explicitely specify a node which should become the new leader.

Expose information about timeline and history

- Expose current timeline in DCS and via API (Alexander)

  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)

  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, Oleksii Kliukin)

  Those endpoints (also accessible as /synchronous and /asynchronous) return 200 only for synchronous and asynchornous replicas correspondingly (exclusing those marked as *noloadbalance*).

Allow multiple hosts for Etcd

- Add a new *hosts* parameter to Etcd configuration (Alexander)

  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.

11.6 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)

  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)

  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)

  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)

  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).

11.7 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)

  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)

  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.

11.8 Version 1.3.4

Different Consul improvements

- Pass the consul token as a header (Andrew Colin Kissa)

  Headers are now the prefered 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)

  new python-consul module has changed signature of some methods

- "Could not take out TTL lock" message was never logged (Alexander)

  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)

  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)

- 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

11.9 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)
- open access in `pg_hba.conf` to all databases, not only postgres (Franco Bellagamba)

11.10 Version 1.3.2

Bugfix

- patronictl edit-config didn't work with ZooKeeper (Alexander Kukushkin)

11.11 Version 1.3.1

Bugfix

- failover via API was broken due to change in `_MemberStatus` (Alexander Kukushkin)

11.12 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)

  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)

  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)

  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)

  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)

  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)

  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)

- 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)

  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)

  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)

  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)

- 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)

  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).

  - 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.

11.13 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)
- Implement retry and timeout strategy for Consul similar to how it is done for Etcd. (Alexander)
- Make `--dcs` and `--config-file` apply to all options in `patronictl`. (Alexander)
- Write all postgres parameters into postgresql.conf. (Alexander)

  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)
- Improve the leader watch functionality. (Alexander)

  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)

  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)

  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, Alexander)

  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)

  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)

- Avoid promoting a former master when the leader key is set in DCS but update to this leader key fails. (Alexander)

  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)

  Use `wal_level = replica` as a synonym for `hot_standby`, avoiding pending_restart flag when it changes from one to another. (Alexander)

Documentation improvements

- Add a Patroni main loop workflow diagram. (Alejandro, Alexander)
- 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)

  Makes the documentation easily viewable from different devices (including smartphones) and searchable.

- Move the package to the semantic versioning. (Oleksii)

  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.

11.14 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)

  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)

  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).

  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)

  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 loosing the lock.

- Update Etcd topology only after original request succeed (Alexander)

  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)

  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)

  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)

  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)

  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, Oleksii)

  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)

  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)

  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.

11.15 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 *dynamic 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 configuraton 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)

  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)
- Split the Zookeeper and Exhibitor configuration options (Alexander)
- Make patronictl reuse the code from Patroni to read configuration (Alexander)

  This allows patronictl to take advantage of environment-based configuration.

- Set application name to node name in primary_conninfo (Alexander)

  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)

  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)

  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)

  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)

  Demote now runs totally asynchronously without blocking the DCS interactions.

- Make patronictl always send the authorization header if it is configured (Alexander)

  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)

  Avoids the issues of Patroni not stopping properly when receiving the SIGTERM

- Sample haproxy templates for confd (Alexander)

  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)

  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)

  Make retries more predictable and robust.

- Make Zookeeper code more resilient against short network hiccups (Alexander)

  Reduce the connection timeouts to make Zookeeper connection attempts more frequent.

11.16 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)

  This tag makes Patroni always return that the replica is not available to the load balancer.

- Change the implementation of the *clonefrom* tag (Alexander)

  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)

  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).

- Make Travis CI faster and implement support for running tests against Zookeeper (Exhibitor) and Consul (Alexander)

  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)

  patronictl can use the same configuration file as Patroni itself.

- Enable Patroni to read the configuration from the environment variables (Oleksii)

  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)

- Implement *delete_cluster* for all available DCSs (Alexander)

  Enables support for DCSs other than Etcd in patronictl.

11.17 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).

  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).
- Add PostgreSQL version to the health check output (Feike).
- 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).
- Fix the handling of CTRL-C by creating a separate session ID and process group for the postmaster launched by Patroni (Alexander).

Tests

- Add acceptance tests with *behave* in order to check real-world scenarios of running Patroni (Alexander, Oleksii).

  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

TWELVE

CONTRIBUTING GUIDELINES

Wanna contribute to Patroni? Yay - here is how!

12.1 Chatting

Just want to chat with other Patroni users? Looking for interactive troubleshooting help? Join us on channel #patroni in the PostgreSQL Slack.

12.2 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.

12.3 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

THIRTEEN

INDICES AND TABLES

- genindex
- modindex
- search