Patroni - HA PostgreSQL with Zookeeper, Etcd or Consul Documentation

Release 1.1

Zalando SE

October 11, 2016

Contents

Patroni is an open-source High-Availability solution for PostgreSQL. It provides PostgreSQL automatic failovers and more.

Contents:

CHAPTER 1

Introduction

1.1 What is Patroni

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 or Consul. 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.

**Note to Kubernetes users: We're currently developing Patroni to be as useful as possible for teams running Kubernetes on **top of Google Compute Engine; Patroni can be the HA solution for Postgres in such an environment. Please contact us via our **Issues Tracker if this describes your team's current setup, and we'll follow up.

Patroni originated as a fork of Governor, the project from Compose. It includes plenty of new features.

There are many ways to run high availability with PostgreSQL; for a list, see the PostgreSQL Documentation.

1.2 Usage examples and talks

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

CHAPTER 2

Installation and Configuration

2.1 Installation

For Mac

To install requirements on a Mac, run the following:

```
brew install postgresql etcd haproxy libyaml python
pip install psycopg2 pyyaml
```

2.2 Running

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
```

2.3 Configuration Settings

2.3.1 Global/Universal

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

2.3.2 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
    - **maximum_lag_on_failover**: the maximum bytes a follower may lag to be able to participate in leader election.
    - **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.
- **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 users which needs 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_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.

2.3.3 Consul

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

2.3.4 Etcd

- **host**: the host:port for the etcd endpoint.

2.3.5 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.

2.3.6 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_methods**: an ordered list of the create methods for turning a Patroni node into a new replica. "basebackup" is the default method; other methods are assumed to refer to scripts, each of which is configured as its own config item.
- **data_dir**: The location of the Postgres data directory, either existing or to be initialized 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.
- **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 overriden 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_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".

2.3.7 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.

2.3.8 ZooKeeper

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

2.4 Environment variables

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

2.4.1 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

2.4.2 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,create` will cause creation of the user **admin** with the password **strongpasswd** that is allowed to create other users and databases.

2.4.3 Consul

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

2.4.4 Etcd

- **PATRONI_ETCD_HOST**: the host:port for the etcd endpoint.

2.4.5 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.

2.4.6 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_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).

2.4.7 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.

2.4.8 ZooKeeper

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

CHAPTER 3

Usage

3.1 Replication Choices

Patroni uses Postgres' streaming replication, which is asynchronous by default. For more information, see the Postgres documentation on streaming replication.

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 follower. This setting should be increased or decreased based on business requirements.

When asynchronous replication is not optimal for your use case, investigate Postgres's synchronous replication. 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 follow lines to the `parameters` section of your YAML configuration files:

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

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

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.

3.2 Security

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.

3.3 Dynamic 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 you 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 `PATRRONI_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.

3.3.1 REST API

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

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"
    }
  }
}
```

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,
  "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,
      "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.

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
        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
}
```

3.4 Pause/Resume mode

3.4.1 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.

3.4.2 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 to stop Postgres instance it is managing.

3.4.3 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 4

Development

4.1 Contributing

Wanna contribute to Patroni? Yay - here is how!

4.2 Filing issues

If you have a question about patroni or have a problem using it, please read the user documentation before filing an issue. Also double check with the current issues on our Issues Tracker.

4.3 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 5

Indices and tables

- genindex
- modindex
- search