

# Patroni

What the blog posts don't tell you...

Cameron Murdoch

Database Operations  
IT Department  
University of Oslo

PGConf.EU, October 2025



**UNIVERSITY  
OF OSLO**

# What are we talking about?

Patroni: what the blog posts don't tell you...

# What are we not talking about?

- Deployments on Kuberetes or similar
- Synchronous replication
- DCS other than Etcid
- Postgres!

# What is a patroni and how do I catch one?

From <https://patroni.readthedocs.io/>

"Patroni is a template for high availability (HA) PostgreSQL solutions using Python"

So Patroni...

- Is a python program
- That manages/runs Postgres for us
- And uses Postgres's streaming replication system to manage read only replicas
- Including automatic failover if the primary fails, etc.

# Why do we want that?

In the beginning...

A computer was just a computer... but things break so we added:

- RAID
- Redundant power/UPS
- Redundant networks
- Virtualisation
- Cloud computing
- etc



# Hot take!

Databases are fairly important!

People tend to get upset quite quickly when their database stops responding.

## Poll

Think of the most important database you maintain...

If you ran the equivalent of:

```
systemctl stop postgresql.service
```

How long before the shouting starts?

# The solution?

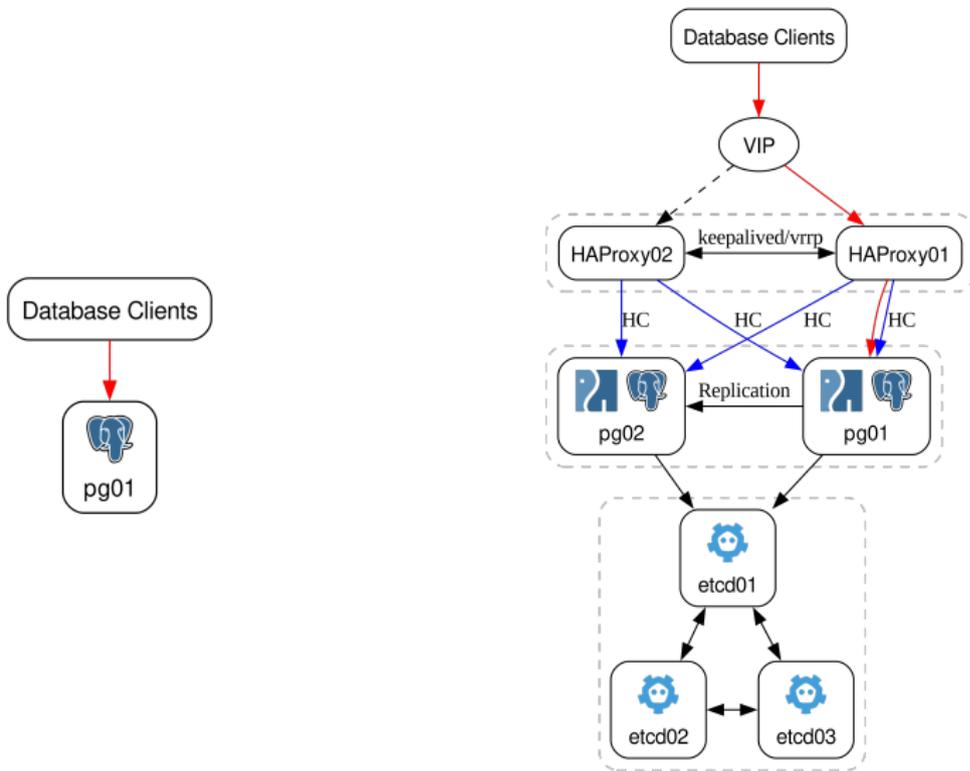


Build a cluster: run your database on several things (computers, virtual machines, containers, etc) at once!

Patroni essentially makes this easier to do.

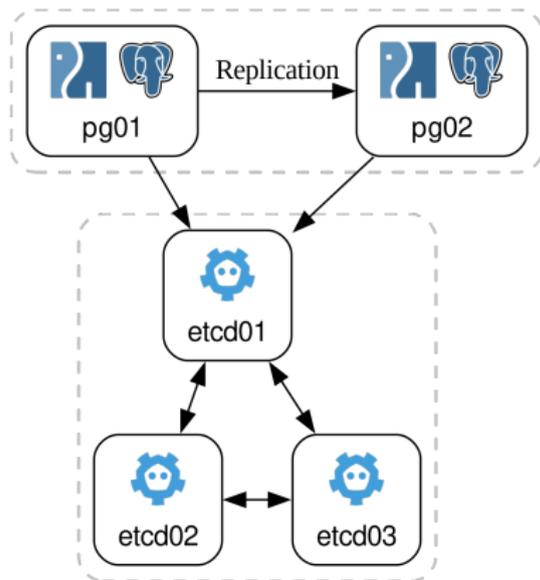
# Spot the difference

Why you might not want Patroni...



# A typical deployment

We are going cover this quickly!

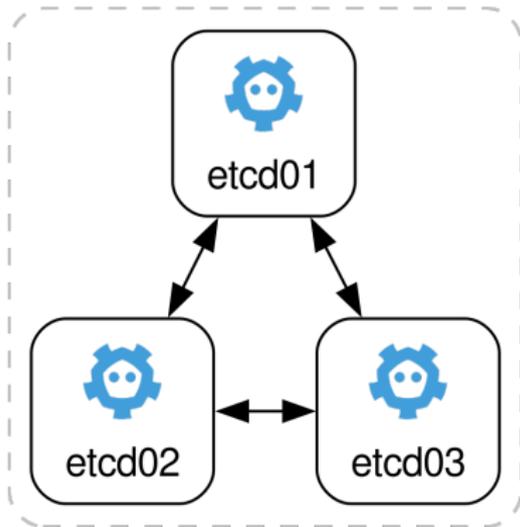


# A typical deployment

## Distributed configuration store

First we need a distributed configuration store (DCS).

- Patroni uses this to store cluster config that has to be same across all nodes
- Several are supported: Etcd, Zookeeper, Consul, and more...
- Unless you have a reason to use something else: use Etcd



# Etcd setup

## Cluster size

### A decision to make!

- Etcd needs to be highly available!
- Needs a quorum (majority) of nodes running and accessible to work

Cluster size	Quorum	Failure Tolerance
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2

This means we should always use an odd number of nodes for etcd:  
probably 3 or 5

# Etcd

## Typical first config

Typically see something like this

### node1

```
name: etcd01
data-dir: /var/lib/etcd

# Bootstrap
initial-cluster-token: "etcd_basic"
initial-cluster-state: "new"
initial-advertise-peer-urls: "http://etcd01:2380"
initial-cluster: "etcd01=http://etcd01:2380,etcd02=http://etcd02:2380,etcd03=http://etcd03:2380"

# Peer
listen-peer-urls: "http://0.0.0.0:2380"

# Client
listen-client-urls: "http://0.0.0.0:2379"
advertise-client-urls: "http://etcd01:2379"
```

Everything that starts with "initial" is only for bootstrapping. Once the cluster is running we use runtime config to make changes.

# Etcd

Up and running

If you run the above config then...

This works!

```
# etcdctl member list -w table
+-----+-----+-----+-----+-----+-----+
| ID | STATUS | NAME | PEER ADDR | CLIENT ADDR | IS LEARNER |
+-----+-----+-----+-----+-----+-----+
| 2d4f4cb7d3809c1d | started | etcd02 | http://etcd02:2380 | http://etcd02:2379 | false |
| 38d9a09e32233f16 | started | etcd03 | http://etcd03:2380 | http://etcd03:2379 | false |
| e753950248f40580 | started | etcd01 | http://etcd01:2380 | http://etcd01:2379 | false |
+-----+-----+-----+-----+-----+-----+

# etcdctl put this works
OK
# etcdctl get this
this
works
```

Yay! Etcd is running, on to Patroni!

# Patroni

## Initial config

### patroni.yml

```
scope: pg-basic
name: pg01

restapi:
  listen: pg01:8008
  connect_address: pg01:8008

etcd3:
  hosts: etcd01:2379,etcd02:2379,etcd03:2379
  protocol: http

bootstrap:
  dcs:
    ttl: 30
    loop_wait: 10
    retry_timeout: 10
    maximum_lag_on_failover: 1048576
  postgresql:
    use_pg_rewind: true
    use_slots: true
    pg_hba:
      - local all postgres peer
      - host replication replicator 0.0.0.0/0 scram-sha-256
      - host all all 0.0.0.0/0 scram-sha-256
    parameters:
      max_connections: 100
  initdb:
    - encoding: UTF8
    - data-checksums

postgresql:
  listen: pg01,127.0.0.1:5432
  connect_address: pg01:5432
  use_unix_socket: true
  data_dir: /var/lib/pgsql/17/data
  bin_dir: /usr/pgsql-17/bin
  authentication:
    superuser:
      username: postgres
      password: putagoodpasswordhere
    replication:
      username: replicator
      password: andheretoo

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

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

# Patroni

## Initial config

### patroni.yml

```
scope: pg-basic
name: pg01

restapi:
  listen: pg01:8008
  connect_address: pg01:8008

etcd3:
  hosts: etcd01:2379,etcd02:2379,etcd03:2379
```

# Patroni

## Dynamic config

### patroni.yml

```
bootstrap:
  dcs:
    ttl: 30
    loop_wait: 10
    retry_timeout: 10
    maximum_lag_on_failover: 1048576
  postgresql:
    use_pg_rewind: true
    use_slots: true
    pg_hba:
      - local all postgres peer
      - host replication replicator 0.0.0.0/0 scram-sha-256
      - host all all 0.0.0.0/0 scram-sha-256
    parameters:
      max_connections: 100
```

This is written to DCS when cluster is first started up.

# Patroni

## Local config

### patroni.yml

```
postgresql:
  listen: pg01,127.0.0.1:5432
  connect_address: pg01:5432
  use_unix_socket: true
  data_dir: /var/lib/pgsql/17/data
  bin_dir: /usr/pgsql-17/bin
  authentication:
    superuser:
      username: postgres
      password: putagoodpasswordhere
  replication:
    username: replicator
    password: andheretoo

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

This is local config read from patroni.yml

# Patroni

Up and running

If you run the above config then...

This works too!

```
# patronictl -c patroni.yml list
+ Cluster: pg-basic (7563947348454305338) -+-----+-----+-----+-----+
| Member | Host | Role   | State   | TL | Receive LSN | Lag | Replay LSN | Lag |
+-----+-----+-----+-----+---+-----+-----+-----+-----+
| pg01   | pg01 | Replica | streaming | 2 | 0/5000168 | 0 | 0/5000168 | 0 |
| pg02   | pg02 | Leader  | running  | 2 |           |   |           |   |
| pg03   | pg03 | Replica | streaming | 2 | 0/5000168 | 0 | 0/5000168 | 0 |
+-----+-----+-----+-----+---+-----+-----+-----+-----+

# psql -h "pg01,pg02,pg03" -U postgres -c "SELECT 'works' AS this"
this
-----
works
(1 row)
```

### Insert while pg02 is primary

```
psql \  
  "target_session_attrs=primary" \  
  -h "pg01,pg02,pg03" \  
  -U postgres \  
  -c "INSERT INTO t1 (msg,added_time) VALUES ('Hello',now())"
```

### Oh dear! - Something nuked pg02

```
# shutdown
```

### Failover!

Who is the new primary?

### Don't care!

```
# psql \  
  "target_session_attrs=primary" \  
  -h "pg01,pg02,pg03" \  
  -U postgres \  
  -c "INSERT INTO t1 (msg,added_time) VALUES ('world',now())"  
INSERT 0 1
```

As it happens...

p01 is primary

```
+ Cluster: pg-basic (7563947348454305338) -+-----+-----+-----+-----+
| Member | Host | Role   | State | TL | Receive LSN | Lag | Replay LSN | Lag |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| pg01   | pg01 | Leader | running | 7 |              |      |              |      |
| pg02   | pg02 | Replica | stopped |   |          unknown |      |          unknown |      |
| pg03   | pg03 | Replica | streaming | 7 | 0/110453E8 | 0 | 0/110453E8 | 0 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

And...

```
# psql \
  "target_session_attrs=primary" \
  -h "pg01,pg02,pg03" \
  -U postgres \
  -c "SELECT * FROM t1"
id | msg |          added_time
-----+-----+-----
 1 | Hello | 2025-09-22 19:37:49.801598+00
 2 | world | 2025-09-22 19:44:43.913232+00
(2 rows)
```

We did it!



# But...

Here be dragons

Let us go back to etcd.

Any alarm bells ringing?

```
# etcdctl member list -w table
```

ID	STATUS	NAME	PEER ADDRS	CLIENT ADDRS	IS LEARNER
2d4f4cb7d3809c1d	started	etcd02	http://etcd02:2380	http://etcd02:2379	false
38d9a09e32233f16	started	etcd03	http://etcd03:2380	http://etcd03:2379	false
e753950248f40580	started	etcd01	http://etcd01:2380	http://etcd01:2379	false

Eve? Can you hear us?

PEER ADDRS	CLIENT ADDRS
http://etcd02:2380	http://etcd02:2379
http://etcd03:2380	http://etcd03:2379
http://etcd01:2380	http://etcd01:2379

# It gets worse

No authentication either

```
# curl -s http://etcd02:2379/v3/kv/put \  
-X POST \  
-d '{"key": "this", "value": "d29ya3MK"}'
```

```
# curl -s http://etcd02:2379/v3/kv/range \  
-X POST \  
-d '{"key": "this"}' \  
| jq -r '.kvs[0].value' \  
| base64 --decode  
works
```

Anyone or anything can read or write to the DCS.  
This **might** be ok.

# What is in the DCS anyway?

```
# patronictl -c /etc/patroni/patroni.yml show-config
loop_wait: 10
maximum_lag_on_failover: 1048576
postgresql:
  parameters:
    max_connections: 100
  pg_hba:
    - local all postgres peer
    - host replication replicator 0.0.0.0/0 scram-sha-256
    - host all all 0.0.0.0/0 scram-sha-256
  use_pg_rewind: true
  use_slots: true
retry_timeout: 10
ttl: 30
```

Do you give everyone write access to postgresql.conf/pg\_hba.conf?

# Ok, let's fix this!

## Enable TLS

```
# Bootstrap
initial-cluster-token: "etcd_fixed"
initial-cluster-state: "new"
initial-advertise-peer-urls: "https://etcd05:2380"
initial-cluster: "etcd04=https://etcd04:2380,etcd05=https://etcd05:2380,etcd06=https://etcd06:2380"

# Common TLS settings
tls-min-version: "TLS1.2"
cipher-suites:
  - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
  - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
  - TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305

# Peers
listen-peer-urls: "https://0.0.0.0:2380"

peer-transport-security:
  cert-file: "/etc/etcd/etcd05_peer.crt"
  key-file: "/etc/etcd/etcd05_peer.key"
  trusted-ca-file: "/etc/etcd/ca_peer.crt"
  client-cert-auth: true

# Clients
listen-client-urls: "https://0.0.0.0:2379"
advertise-client-urls: "https://etcd05:2379"

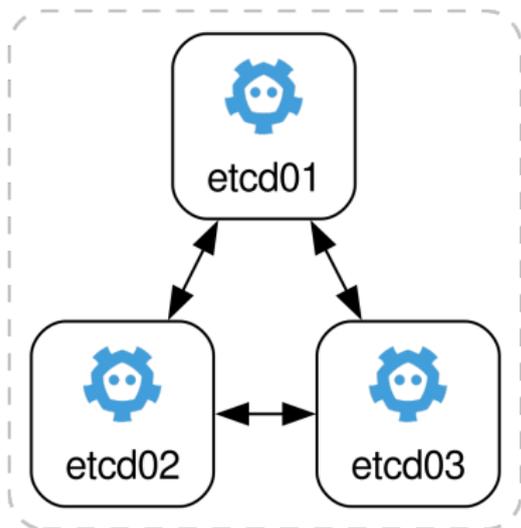
client-transport-security:
  cert-file: "/etc/etcd/etcd05_client.crt"
  key-file: "/etc/etcd/etcd05_client.key"
  client-cert-auth: false
```

# Etcd peers

Etcd peer endpoints are mutually authenticated

```
peer-transport-security:  
  cert-file: "/etc/etcd/etcd05_peer.crt"  
  key-file: "/etc/etcd/etcd05_peer.key"  
  trusted-ca-file: "/etc/etcd/ca_peer.crt"  
  client-cert-auth: true
```

Good use case for internal/private PKI?



# Etcctl

TLS enabled

If we run the above config (as a new cluster):

## Encrypted

```
# etcctl --endpoints="https://etcd04:2379" member list -w table
```

ID	STATUS	NAME	PEER ADDR	CLIENT ADDR	IS LEARNER
6de1706f31916f32	started	etcd06	https://etcd06:2380	https://etcd06:2379	false
c2475baf2574ffe1	started	etcd05	https://etcd05:2380	https://etcd05:2379	false
e73518237edbc109	started	etcd04	https://etcd04:2380	https://etcd04:2379	false

Bit more work to add TLS to existing cluster, see Etcd documentation.

# Etcd

## Enable Authentication

TLS done. Now let's enable authentication!

### Add role

```
# ETCDCCTL_ENDPOINTS="https://etcd04:2379,https://etcd05:2379,https://etcd06:2379"  
# etcdctl role add root  
Role root created
```

### Add user

```
# etcdctl user add root --new-user-password="rootpass"  
User root created
```

# Etcctl

## Enable Authentication

### Grant role to user

```
# etcdctl user grant-role root root  
Role root is granted to user root
```

### Enable auth

```
# etcdctl auth enable  
Authentication Enabled
```

# Etcd

## Enable Authentication

### Authenticaiton required

```
# etcdctl get this
"error": "rpc error: code = InvalidArgument desc = etcdserver: user name is empty"
```

```
# export ETCDCCTL_USER=root
# export ETCDCCTL_PASSWORD=rootpass
# etcdctl get this
this
works
```

We need a user in Etcd for Patroni to use.

- We could use root user with Patroni
- Better: create a user for all Patroni clusters
- Better still?: create a user per Patroni cluster

# Etcctl

## Add a user for Patroni

### Add role

```
# etcctl role add pg_fixed_role
Role pg_fixed_role created
```

### Grant role access to /service/(scope)

```
# etcctl role grant-permission pg_fixed_role readwrite \  
    /service/pg-fixed --prefix=true  
Role pg_fixed_role updated
```

"pg-fixed" is Patroni's "scope" setting.

# Etcctl Auth

Patroni

## Add user

```
# etcdctl user add pg_fixed_user --new-user-password="pg_fixed_pass"  
User pg_fixed_user created
```

## Grant role to user

```
# etcdctl user grant-role pg_fixed_user pg_fixed_role  
Role pg_fixed_role is granted to user pg_fixed_user
```

We now have a user that can read and write to the keyspace under `"/service/pg-fixed"`.

# Patroni

Re-configure to use TLS and authentication with Etcd

## Patroni Config

```
scope: pg-fixed
name: pg04

restapi:
  listen: pg04:8008
  connect_address: pg04:8008

etcd3:
  hosts: - etcd04:2379,etcd05:2379,etcd06:2379
  protocol: https
  cacert: /etc/ssl/certs/ca-bundle.crt
  username: pg_fixed_user
  password: pg_fixed_pass
```

If we run this then it should work. But...

# Patroni

## Rest api

### Patroni's rest api also has no encryption or authentication

```
restapi:  
  listen: pg04:8008  
  connect_address: pg04:8008
```

### Anyone can do this:

```
# curl -s http://pg04:8008/switchover -XPOST -d '{"leader":"pg06"}'  
Successfully switched over to "pg05"
```

## Or this:

```
# curl -s -XPATCH \  
-d '{"postgresql": {"pg_hba": ["host all all all trust"]}}' \  
http://pg05:8008/config | jq .  
{  
  "ttl": 30,  
  "loop_wait": 10,  
  "retry_timeout": 10,  
  "maximum_lag_on_failover": 1048576,  
  "postgresql": {  
    "use_pg_rewind": true,  
    "use_slots": true,  
    "pg_hba": [  
      "host all all all trust"  
    ],  
    "parameters": {  
      "max_connections": 100  
    }  
  }  
}
```

This probably isn't a good idea.

# Patroni

## Rest api TLS and Authentication

### Add TLS and authentication

```
restapi:  
  listen: pg05:8008  
  connect_address: pg05:8008  
  authentication:  
    username: patroni_user  
    password: patroni_pass  
  certfile: /etc/patroni/ssl/server.crt  
  keyfile: /etc/patroni/ssl/server.key  
  https_extra_headers:  
    'Strict-Transport-Security': 'max-age=31536000; includeSubDomains'
```

See documentation for more settings

### Access denied

```
# curl http://pg04:8008/switchover -XPOST -d '{"leader":"pg04"}'  
curl: (1) Received HTTP/0.9 when not allowed
```

# Patroni

## TLS and Authentication

- Basic auth is probably minimum you would want
- Can also use certificates for authentication here
- Also look at `allowlist` and `allowlist_include_members` in the documentation
- Set ciphers like we did for Etcd

# More Patroni config

## DCS Failsafe Mode

You probably want this

Increases robustness of Patroni/Postgres in the event of DCS outage

If the DCS is down:

**Without Failsafe mode**

Primary is demoted to avoid split brain

**With Failsafe mode**

As long as all the Patroni/Postgres nodes are running and have connectivity, then the primary continues to run.

# More Patroni config

## DCS Failsafe Mode

How to enable:

Enabled during bootstrap with:

```
bootstrap:  
  dcs:  
    failsafe_mode: true
```

Or on a working cluster with:

```
# patronictl edit-config -s failsafe_mode=true
```

# To DCS or not to DCS

That is the question!

Patroni has 3 places you can place config:

- Global dynamic config, stored in the DCS
- Local config file, `patroni.yml`, overrides DCS
- Environment variables, overrides local config

# To DCS or not to DCS

That is the question!

Although there are a few config options that can **ONLY** be changed in the DCS:

- `max_connections`
- `max_locks_per_transaction`
- `max_worker_processes`
- `max_prepared_transactions`
- `wal_level`
- `track_commit_timestamp`

Patroni ignores these in local config or environment variables

# To DCS or not to DCS

That is the question!

But where should other config go? DCS or local?

# To DCS or not to DCS

That is the question!

## Answer - it depends

- Dynamic config is applied to all nodes
- This can be good: dynamic application of Postgres config!
- But might be a problem if a node gets config you didn't actually want it to get.

# Other things to mention

## Don't run Etcd and Patroni on the same nodes

- Bad if Postgres eats all the resources and Etcd can't work
- Easy to make mistakes later, eg go from 3 to 4 Etcd nodes
- Can run several Patroni clusters from one Etcd cluster

Not time to properly talk about proxys. But maybe you don't need one.

## Primary connection

```
"host=pg01,pg02,pg03  
target_session_attrs=primary  
connect_timeout=2"
```

Newer (from pg16) libpq based clients can load balance the connection.

## Replica connection

```
"host=pg01,pg02,pg03  
target_session_attrs=prefer-standby  
connect_timeout=2  
load_balance_hosts=random"
```

# Backups

Etcd

Even though Etcd doesn't contain "real" data, you probably want a backup.

## Snapshot

```
# etcdctl snapshot save snapshot.db  
---snip---  
Snapshot saved at snapshot.db  
Server version 3.6.0
```

This could help speed up disaster recovery.

# Backups

## Postgres

We can run Postgres backups through the proxy (if you have one). But this seems like something that can go direct.

### pg\_dump

```
# pg_dump \  
-F p \  
-f postgres.sql \  
"host=pg01,pg02,pg03 user=postgres dbname=postgres target_session_attrs=primary"
```

Can take backup from secondary too, see documentation for target\_session\_attrs

# Backups

## Postgres

### PITR, eg barman

```
# Barman config
conninfo = host=pg01,pg02,pg03
          target_session_attrs=primary
          user=barman
          dbname=postgres

streaming_conninfo = host=pg01,pg02,pg03
                    target_session_attrs=primary
                    user=streaming_barman
```

### This works

On failover, barman reconnects to new primary and starts streaming wals on the new timeline.

Could also take backup from replica. Can also use barman as a Patroni bootstrap method for creating replicas, etc.

# Backups

Postgres

## Note

There will be new timelines created when working with Patroni. When testing backups:

- Make sure you can restore correctly to earlier timelines
- Test full disaster recovery

If I ran `rm -rf *` for Patroni, Postgres and Etcd on all nodes, can you recover? How quickly can your highly available system be back up again?

# Upgrades

Etc

## Etc Upgrade

- Can upgrade a running cluster without downtime
- Stop, install new binary, start
- But you should read the release notes for each new version
- And you **must** upgrade one node at a time

We have ansible automation that upgrades one node at a time, checking cluster health at each step.

# Upgrades

## Postgres

### Postgres minor upgrade

- Bit more work, but Patroni helps here.
- Upgrade each replica as normal, one at a time
- Controlled switchover at a sensible/agreed time
- Upgrade old primary

This can be (semi)-automated.

# Upgrades

## Postgres

### Postgres major upgrade

Not so straightforward. Difficult to do without some downtime

### Options

Option	Downtime Primary	Downtime Replicas	Complexity
pg_dump and pg_restore into new cluster	High	High	Low
pg_upgrade primary, reinit DCS and replicas	Lower	High	Higher
pg_upgrade primary with 'link' and rsync trick	Lower	Low	Little bit higher
New cluster, logical replication	Low	Low	Very high

# The end - thanks very much!

Please give feedback

