zalando

Patroni in 2019: What's New and Future Plans

PGConf.EU 2019

Milan

ALEXANDER KUKUSHKIN

DMITRII DOLGOV

16-10-2019

ABOUT ME

Alexander Kukushkin

Database Engineer @ZalandoTech

The Patroni guy

alexander.kukushkin@zalando.de

Twitter: @cyberdemn

zalando

ABOUT ME

?

Dmitrii Dolgov

Software Engineer @ZalandoTech

dmitrii.dolgov@zalando.de

Twitter: @erthalion

WE BRING FASHION TO PEOPLE IN 17 COUNTRIES

17 markets

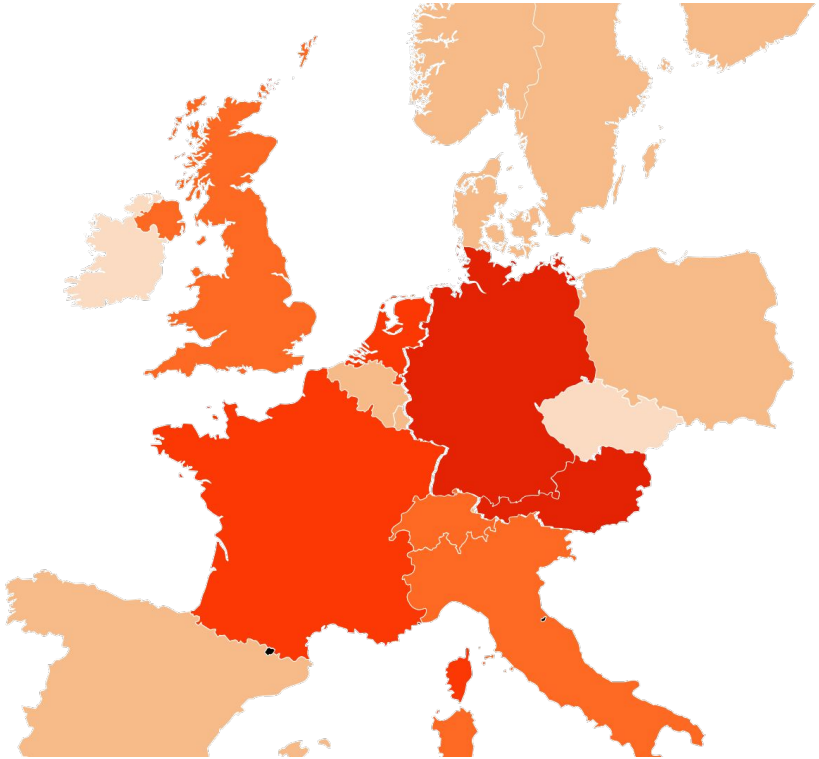
7 fulfillment centers

26.4 million active customers

5.4 billion € net sales 2018

250 million visits per month

15,000 employees in Europe



zalando

PostgreSQL at Zalando

> 300
In the data centers

> 190
Run in the ACID's Kubernetes cluster

> 150
Databases on AWS Managed by DB team

> 1000
Databases in other Kubernetes clusters

AGENDA

Brief introduction to automatic failover

Bot pattern and Patroni

New Patroni features

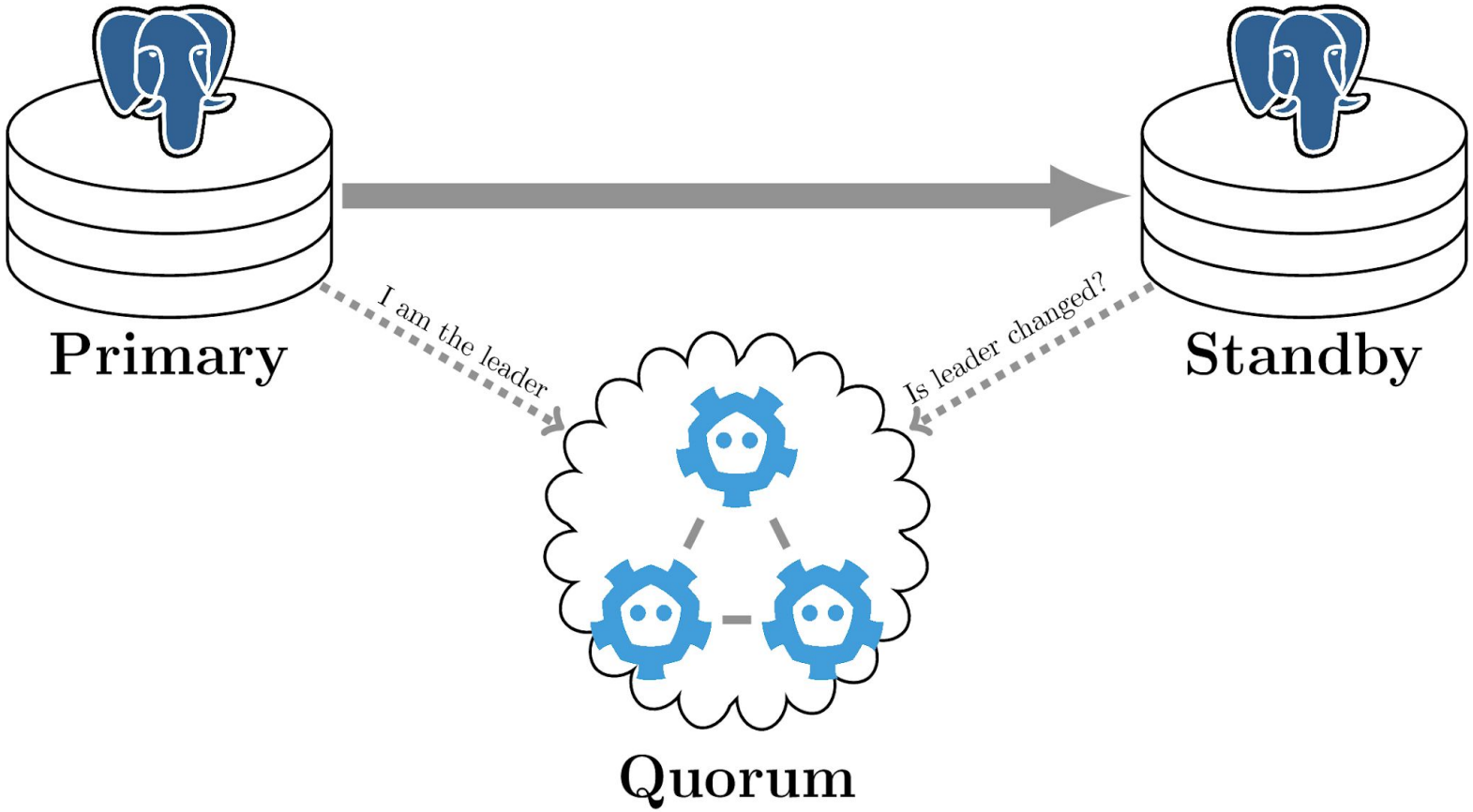
Bug fixes

Plans for future

A good HA system

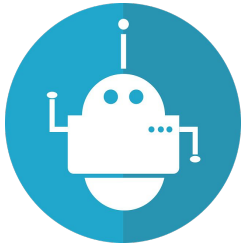
- Quorum
  - Helps to deal with network splits
  - Requires at least 3 nodes
- Fencing
  - Make sure the old primary is unaccessible. STONITH!
- Watchdog
  - Primary should not run if supervising HA process failed

Automatic failover: Patroni

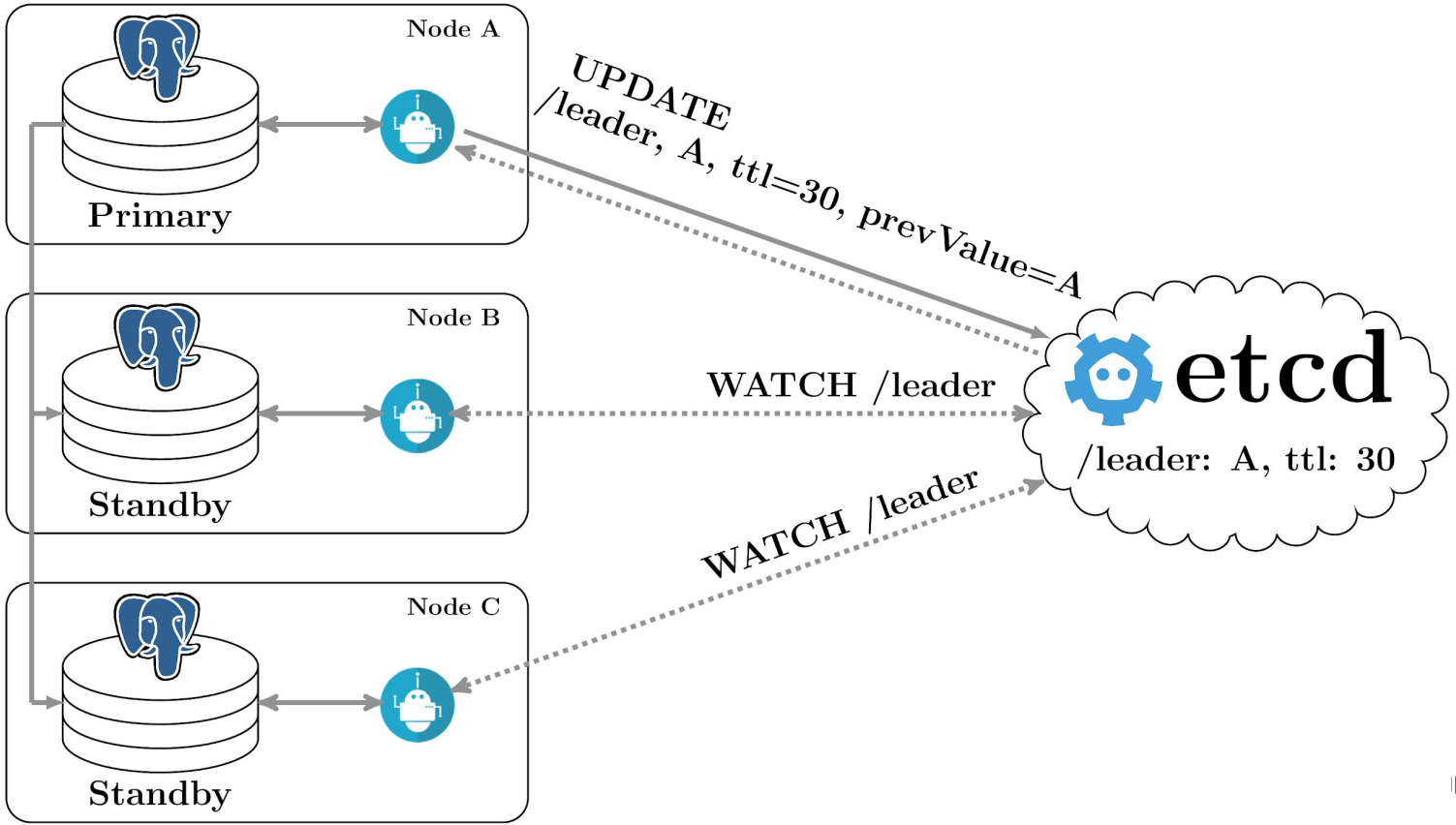


Bot pattern

- PostgreSQL cannot talk to DCS (i.e. Etcd) directly
- Let’s employ a bot alongside PostgreSQL:
  - to manage PostgreSQL
  - to talk to Distributed Consistency Store (DCS)
  - to decide on promotion/demotion

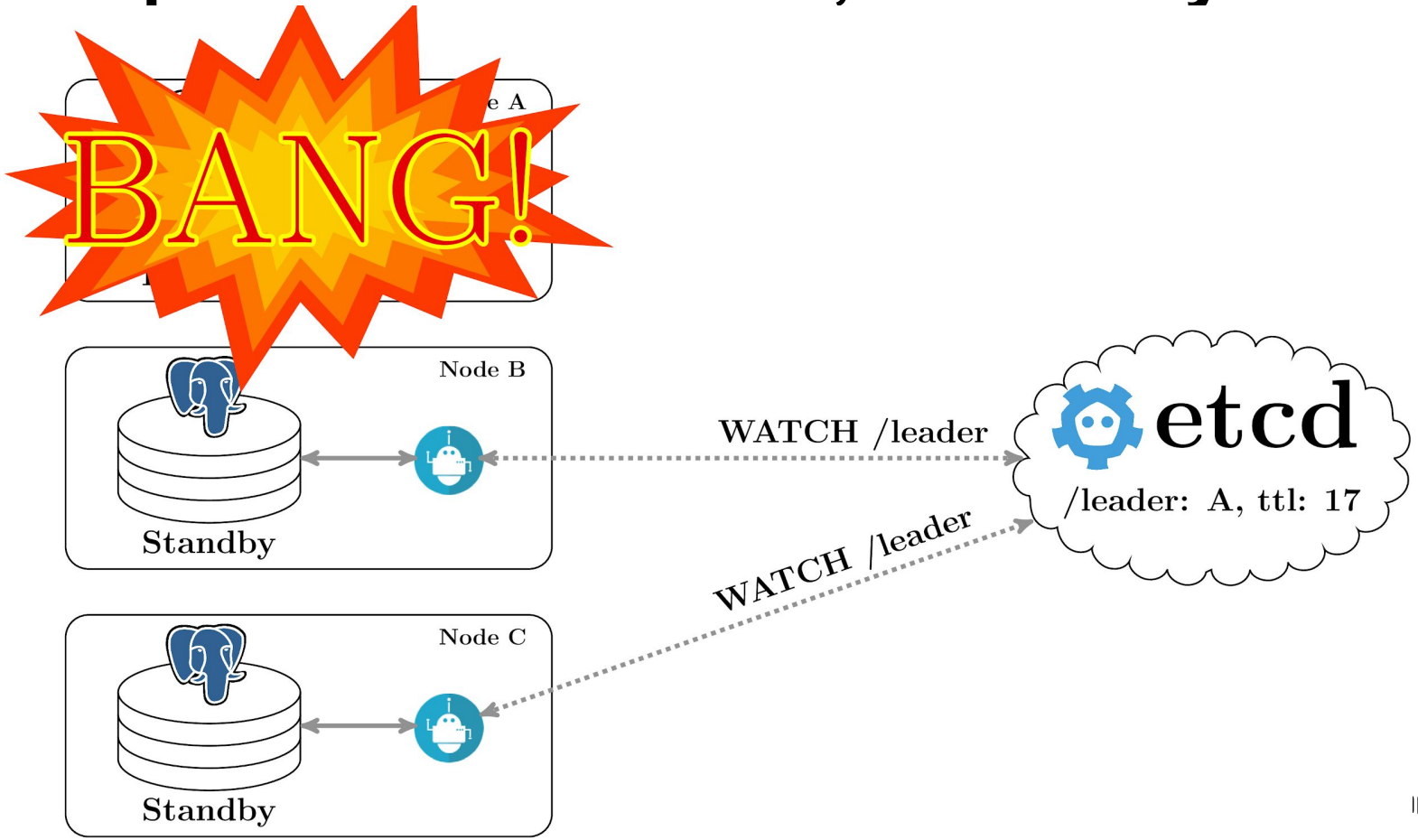


Bot pattern: leader alive

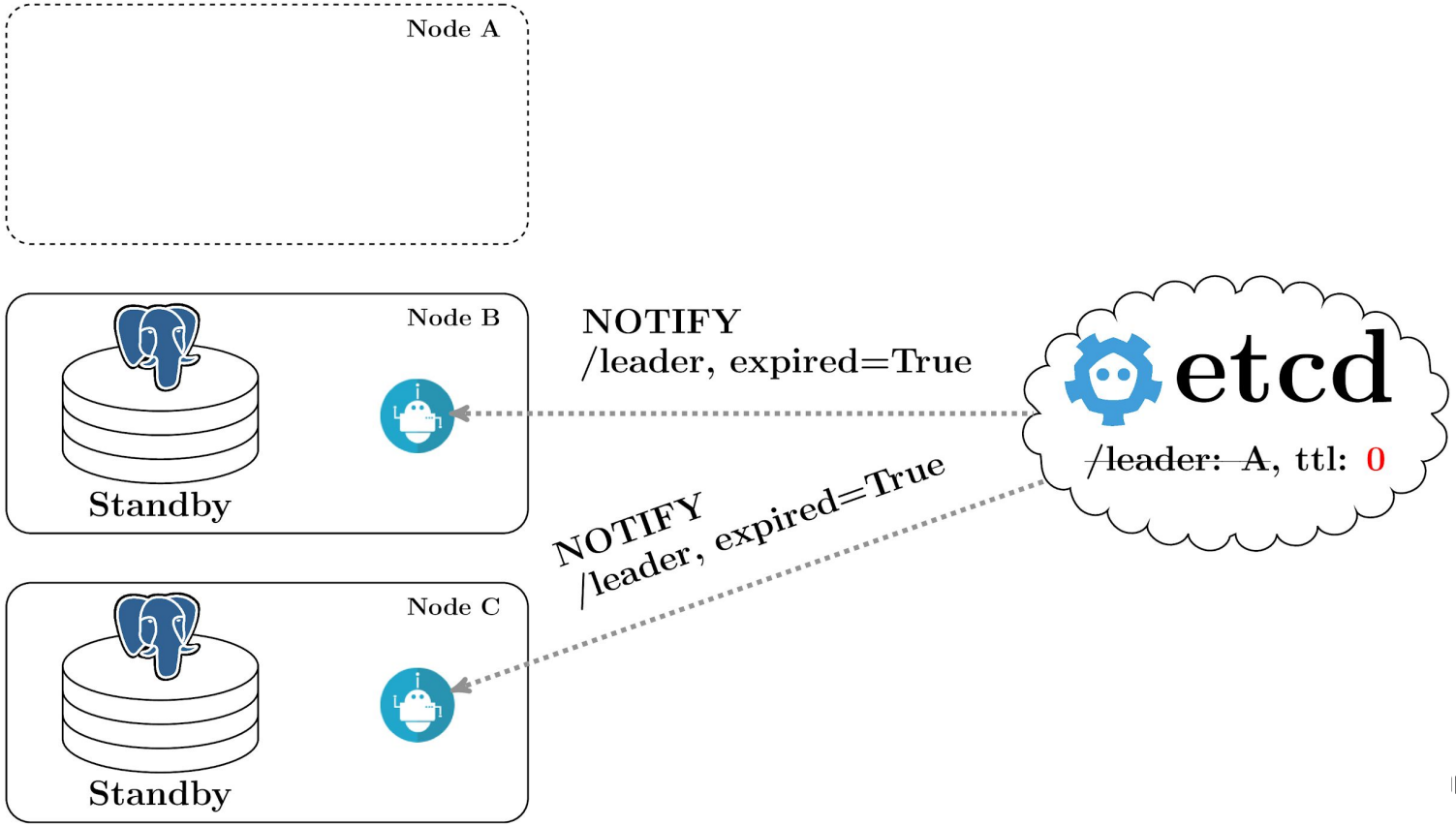


ndo

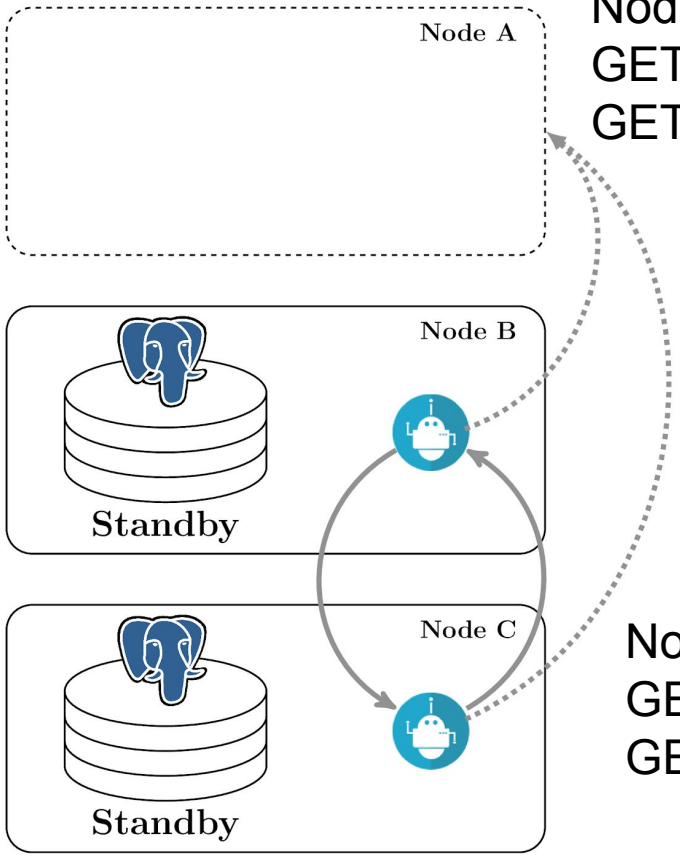
Bot pattern: master dies, leader key holds



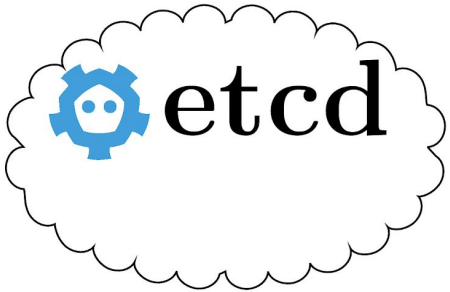
Bot pattern: leader key expires



Bot pattern: who will be the next master?



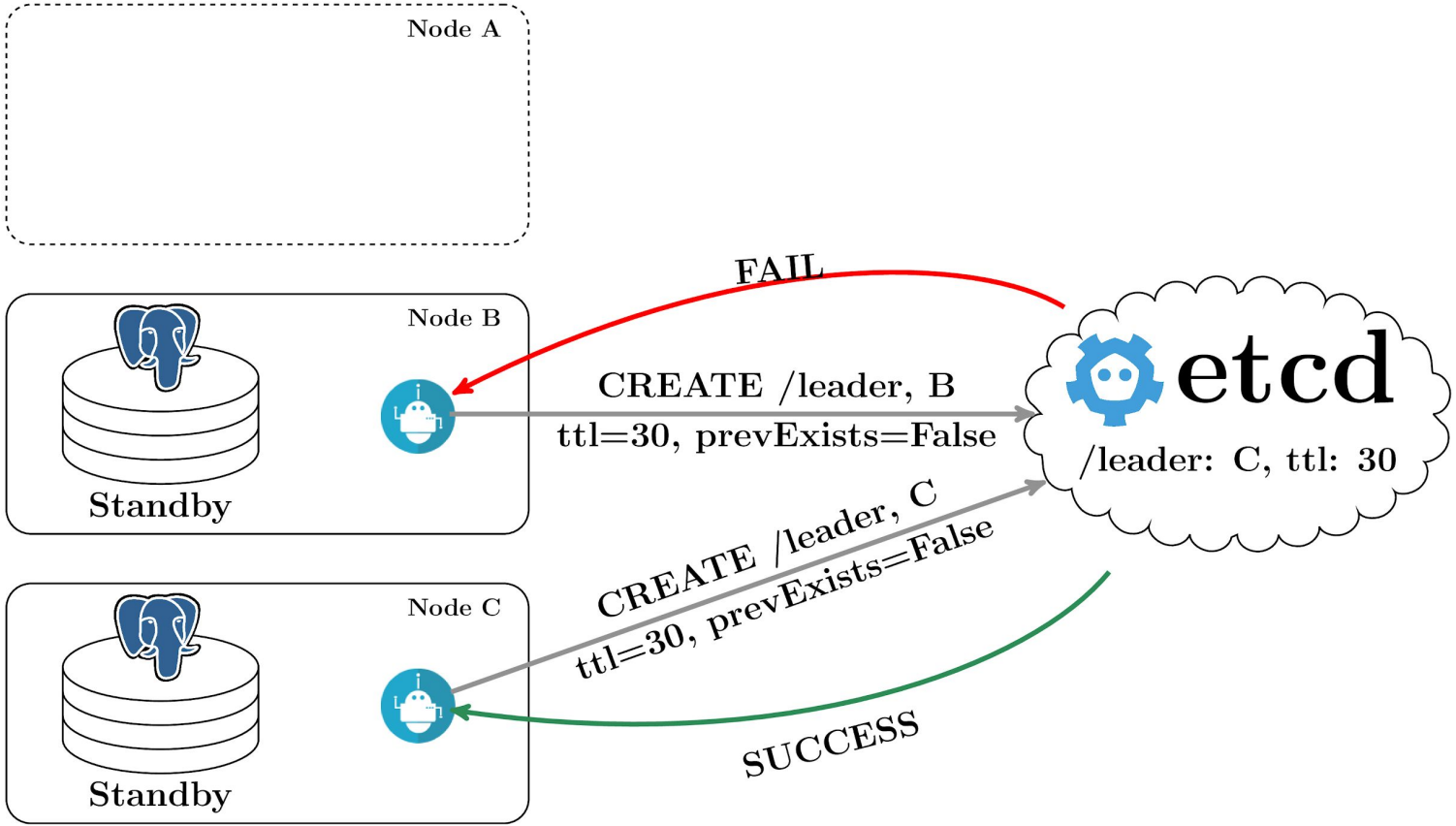
Node **B**:
GET A:8008/patroni -> **failed/timeout**
GET C:8008/patroni -> wal_position: **100**



Node **C**:
GET A:8008/patroni -> **failed/timeout**
GET B:8008/patroni -> wal_position: **100**

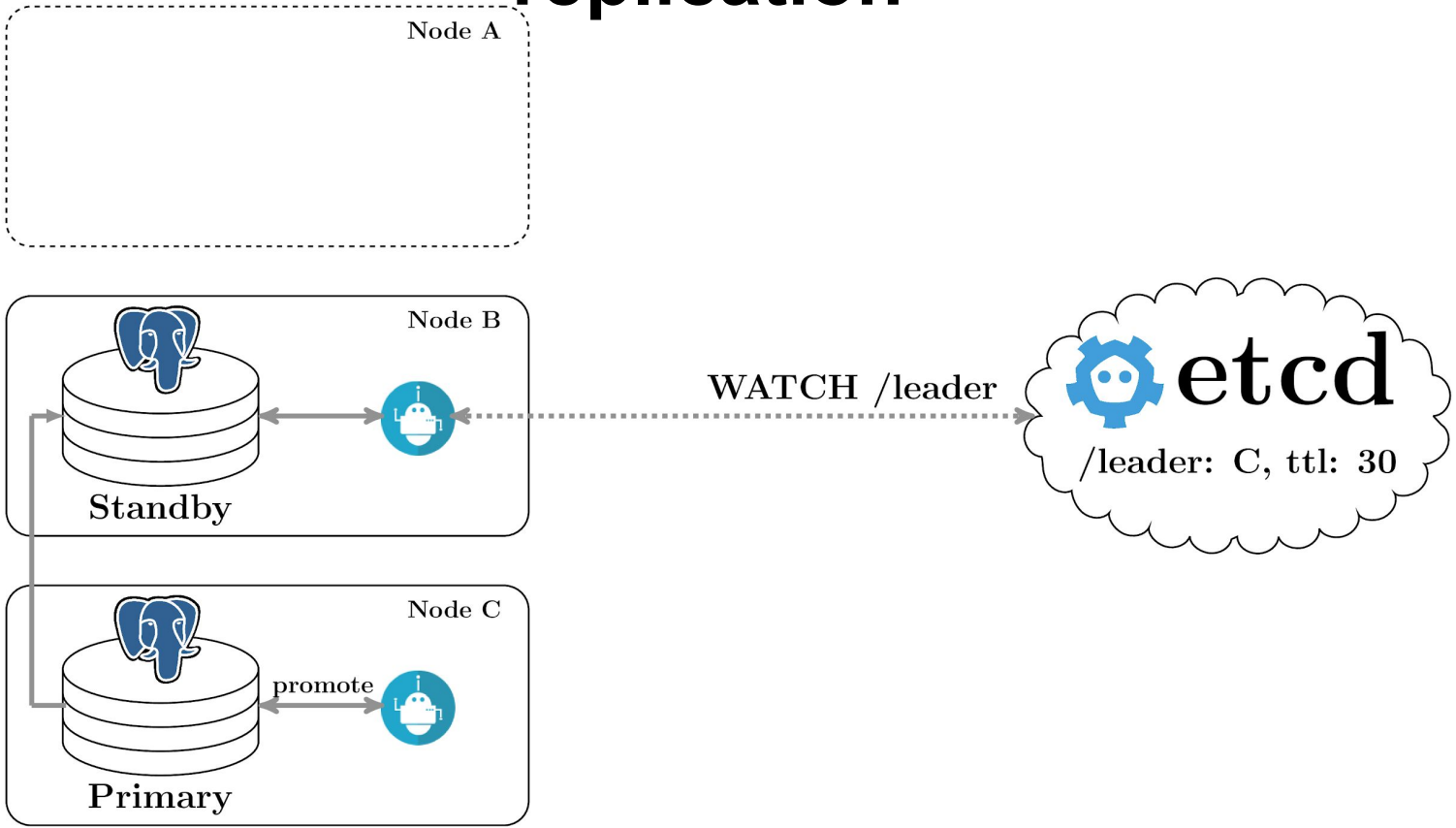
ndo

Bot pattern: leader race among equals



ndo

Bot pattern: promote and continue replication



ndo

Patroni

- Patroni implements bot pattern in Python
- Official successor of Compose Governor
- Developed in the open by Zalando and volunteers all over the world

https://github.com/zalando/patroni

New Patroni Features

PostgreSQL 12 support

- `recovery.conf` moved into `postgresql.conf`
  - All parameters are converted to GUC
  - The `standby.signal` file is used to switch server into non-primary mode
- More flexibility in postgres configuration
  - Allow fractional input for integer server variables
    - For example, `SET work_mem = '30.1GB'`.
  - Time-based units could be specified in micro-seconds

pg_rewind without superuser on pg11+

- **Case:** Patroni needs full control on local postgres (PGDATA, superuser)
- **Before:** Besides that remote superuser access was required
  - For pg_rewind
    - And for CHECKPOINT before calling pg_rewind
- **Now:** Patroni on the new primary exposes information about CHECKPOINT after promote
  - On postgres 11+ we can create a separate user for pg_rewind

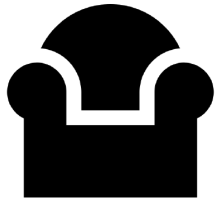
```
postgresql:
  authentication:
    rewind:  # Has no effect on postgres 10 and lower
      username: rewind_user
      password: rewind_password
```

IPv6 support

- IPv4 address pool is limited and soon or later will be depleted
- Databases are usually hosted in private networks
- But, there are IPv6 only systems
  - Hello from Kubernetes

- Patroni fully supports IPv6 starting from 1.6.0

Better integration with pgBackRest



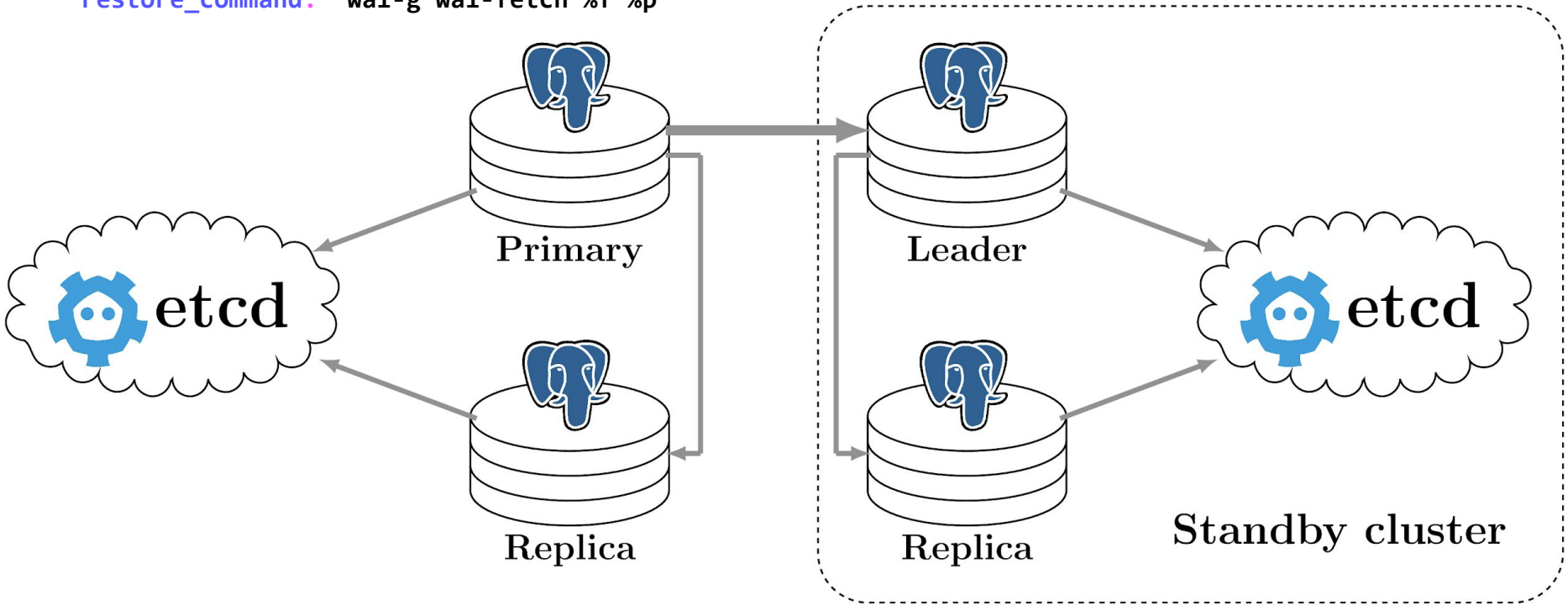
- keep_existing_recovery_conf
  - use the **recovery.conf** file generated by pgBackRest
    - Simplifies Patroni configuration
  - Contributed by **@Brad Nicholson**

- delta restore support
  - Don’t remove PGDATA when reinitializing the node
    - Can significantly speed up resync of large clusters
  - Contributed by **@Yogesh Sharma**

zalando

Standby cluster

```
standby_cluster:
  host: remote-cluster.fqdn.or.ip
  port: 5432
  restore_command: 'wal-g wal-fetch %f %p'
```



Permanent replication slots

Case: An application uses replication slots e.g. for logical decoding

Before: It can experience issues during switchover, when slots were not synchronized yet

Now: One can define a permanent replication slots, that are preserved during switchover/failover, Patroni will try to create slots ***before*** opening connections to the cluster.

```
slots:
  permanent_logical_1:
    type: logical
    database: foo
    plugin: pgoutput
```

Flexible logging

Case: Patroni writes logs

Before: Patroni was writing logs only to **stderr** wit only configurable ***global*** log level

Now: You can choose between stderr and files. It is also possible to change logging configuration on the fly and fine-tune log level per python module.

```
log:
 level: INFO
 dir: /var/log/patroni
 file_size: 50000000
 file_num: 10
 format: '%(asctime)s
%(levelname)s: %(message)s'
 dateformat: '%Y-%m-%d %H:%M:%S'
 loggers:
   etcd.client: DEBUG
   urllib3: DEBUG
```

Two step logging

Case: Extreme resource exhaustion on the node, where Patroni is running

Before: In rare situations it could lead to direct logging blocking HA loop

Now: There is an in-memory queue for logging messages, that are asynchronously flushed to a log destination

patronictl reload

- Patroni can read config.yaml without restart
  - That requires either:
    - Sending the **SIGHUP** to the Patroni process
    - Doing **POST /reload** REST API call
  - Good for automation
    - Not so handy for humans
- **`patronictl reload`** is a human-friendly interface
  - Contributed by **@Don Seiler**

Register Services in Consul

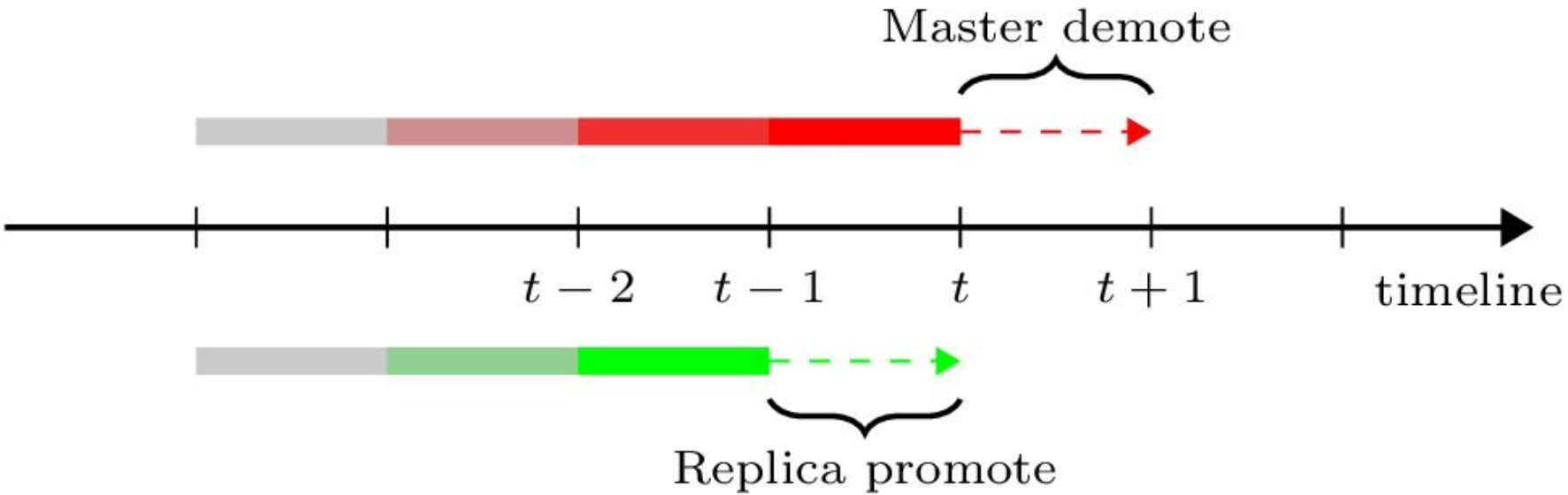
- Consul provides a service discovery via DNS
  - We can use it instead of VIP or HAProxy to find the primary/replica
    - Set consul.register_service: **true** to enable it
- Contributed by **@Pavel Kirillov**

```
$host -t SRV master.pgsql-pgpi.service.consul.
master.pgsql-pgpi.service.consul has SRV record 1 1 5432 pgpi2.node.dc.consul.

$ host -t SRV replica.pgsql-pgpi.service.consul.
replica.pgsql-pgpi.service.consul has SRV record 1 1 5432 pgpi1.node.dc.consul.
replica.pgsql-pgpi.service.consul has SRV record 1 1 5432 pgpi3.node.dc.consul.
```

Stability improvements

Check current timeline



zalando

Check current timeline

Case: Something went wrong, and leader election is happening

Before: Patroni by default do not consider the current timeline of potential candidates, which could lead to undesired result

Now: There is an option that allow to enforce for a new master to not have the same timeline as previous

```
bootstrap:
  dcs:
    check_timeline: true
```

Automatic reinitialize

Case: After switchover/failover pg_rewind is not allowed/failed

Before: The former master could fail to start as a replica due to diverged timelines and the only possible fix would be to reinit it

Now: Patroni can do this automatically if the following option is set:

```
remove_data_directory_on_diverged_timelines: true
```

Converting existing clusters to Patroni

- **Case:** Patroni can attach itself to already running postgres
  - This is very convenient if you want implement HA on already existing primary-standby setup
  - It is imperative to start with primary and continue with replicas!
- **Before**: Patroni was “happily” promoting the replica
- **Now:** Patroni notice that postgres is running as a standby and DCS has no information about this cluster and aborts start.

Take some parameters from controldata

- **Case:** Patroni makes sure that values of `max_connections`, `max_worker_processes` and so on are unified across all cluster nodes
- **Before:** When building a new replica from basebackup (wal-e/wal-g/pgBackrest) it might happen that the value of `max_connections` was higher than the current value stored in DCS
  - `FATAL:  hot standby is not possible because max_connections = X is a lower setting than on the master server (its value was Y)`
- **Now:** Patroni takes current values from `pg_controldata` output:

```
max_connections setting:             99
max_worker_processes setting:        8
max_prepared_xacts setting:          0
max_locks_per_xact setting:          64
```

Fixed bugs

(the most interesting)

Case insensitive parameter names

- Patroni manages postgresql.conf
- It needs to compare the current and the new value in order to figure out if restart is needed or reload is enough
- Most of postgres parameter names are in **snake_case**
  - But, there are some in **CamelCase: DateStyle**, **IntervalStyle** and **TimeZone** (why?)
- For the postgres **timezone = UTC** and **TimeZone = UTC** are the same
  - But in **pg_settings** parameter name visible as a **TimeZone**!

Starting from 1.4.4 Patroni treats all parameter names as case insensitive.

Race conditions around postmaster.pid

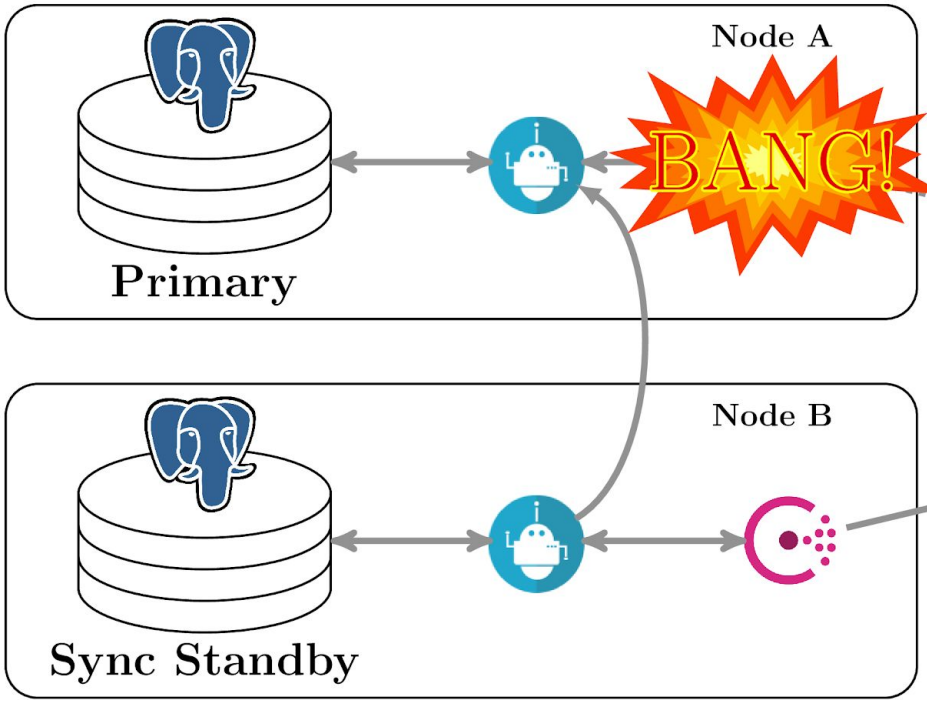
- **Case:** Postgres “locks” data directory when starts
  - It uses **postmaster.pid** for that
    - The PID from the lock file should not be alive
    - The shared memory should not be used
- **Before:** On newly started host/instance/vm/container it is highly likely that the PID will be already taken by existing process
  - Postgres was refusing to start
- **Now:** Patroni does some sophisticated checks and might set environment variable **PG_GRANDPARENT_PID=XYZ**
  - **XYZ** is the PID from from postmaster.pid

Bug is synchronous mode strict

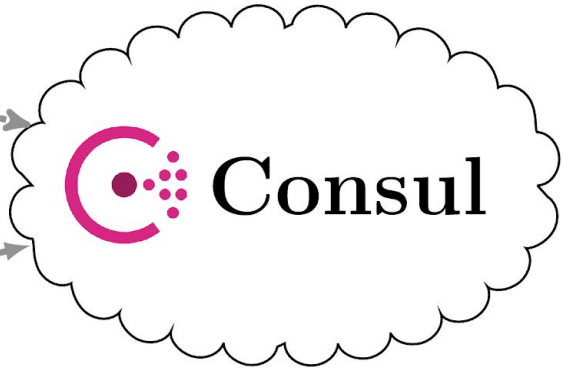
- **Case:** in sync mode Patroni choses sync replica and puts its name into `synchronous_standby_names`
  - Replica is chosen from `pg_stat_replication` view
  - In strict mode Patroni sets `synchronous_standby_names='*'` when there are no replicas available
    - `pg_receivewal` (barman) can become a ***sync replica***
- **Before:** when the real replica was coming back Patroni never stick to it
  - It was considering only connections with `sync_state = 'async'`!
- **Now:** Patroni is choosing sync replica among connections with `sync_state IN ('async', 'potential')`

Bug in sync mode & Consul

Node B: my_wal_position: **100**
GET A:8008/patroni -> wal_position: **101**



Promote of **sync** standby doesn't happen due to **100** < **101**



Fixed in **1.6.0**

Leader watch in Consul

- **Case:** To get a quick notification about leader key expiration Patroni is relying on Blocking Queries: `GET /kv/:cluster/leader?wait=10s`
  - Patroni calls requests.get() with timeout=11 (safety measure)
    - `11 = loop_wait + 1s` (hard-coded constant)
- **Before:** everything working fine with default value of **`loop_wait=10`** and getting **Read timed out** exception when **`loop_wait>=20`**
  - RTFM! *A small random amount of additional wait time is added to the supplied maximum wait time to **spread out** the wake up time of any concurrent requests. This adds **up to wait/16 additional time!***
- **Now:** Hard-coded constant is replaced with a calculated value, **`wait/15`**

What’s next?

Patroni on pure RAFT

- Patroni is relying on a consensus provided by external system (DCS)
- What if we implement RAFT support into Patroni?
  - PySyncObj - RAFT protocol implementation in python
    - Created and battle-tested (literally) by Wargaming
- #375 implements it.
- You have to run either:
  - At least **three** nodes with Patroni and Postgres
  - Two nodes with Patroni and Postgres and one node with **patroni_raft_controller**

Quorum commit

- Patroni support synchronous mode
  - The leader chooses a synchronous node and sticks to it
- PostgreSQL 10 implements quorum commit: `ANY k (*)`
  - #672 is an attempt to make use of it in Patroni
    - Big thanks to **@Ants Aasma** CYBERTEC DATA SCIENCE & POSTGRESQL

Etcd v3 protocol support

- Etcd 3.4.0+ doesn’t enable v2 protocol by default
  - First step to deprecation
  - Workaround: `etcd --enable-v2=true`
- It would be nice to support v3 natively, but…
  - python-etcd3 module still doesn’t provide failover out-of-the-box
    - gRPC is hard
- Luckily there is a JSON gRPC gateway in Etcd
  - #1162 - POC, Etcd v3 API support

Thank you!

Questions?