CYBERTEC
DATA SCIENCE & POSTGRESQL

Patroni ist keine Pizzeria

Julian Markwort
www.cybertec-postgresql.com

Wieso ist Patroni keine Pizzeria?

CYBERTEC
DATA SCIENCE & POSTGRESQL

Pizzerien heißen...

- San Remo
- Don Camillo
- Napoli
- Patroni (?)

In Münster gibt es den *Pizza Patron* (lecker!).
Google Adsense sagt, dass oft `patroni` und `pizzeria` zusammen gegoogelt werden.

Aber: Patroni ist eben nicht nur eine Pizzeria.
Patroni ist eine Software um HA-Cluster mit PostgreSQL zu betreiben.

Patroni hat den Ruf, eine schlechte Dokumentation zu haben.
Dieser Vortrag soll einen einfacheren Einstieg ermöglichen

Dennoch, zum Nachlesen:

- patroni.readthedocs.io

Vielleicht reichen wir auch mal einen Pull-Request ein mit einer besseren Doku...

CYBERTEC
DATA SCIENCE & POSTGRESQL

- Wir bieten Services für PostgreSQL und Data Science
  - 24x7 support
  - Training
  - Consulting
  - Clustering
  - Scaling
  - Geodata
  - Big Data
  - Machine Learning

CYBERTEC
DATA SCIENCE & POSTGRESQL

- Büros in . . .
  - Wiener Neustadt, Österreich
  - Zürich, Schweiz
  - Tallinn, Estland
  - Montevideo, Uruguay

Inhaltsverzeichnis

Einführung

Postgres schreibt alle Änderungen in ein Transaktionslog (*Write Ahead Log*).
Mit den Tabellendateien nach einem Checkpoint und dem WAL kann man alle Transaktionen rekonstruieren und einen konsistenten Datenstand erreichen.

Folglich kann ein Server sein WAL einem anderen schicken, beide haben dann einen konsistenten Stand.

- seit Postgres 9.0
- Postgres sichert das WAL nicht nur auf der Disk, sondern schickt es auch über das Netzwerk an interessierte Dritte
- natürlich mit Authentifizierung (DB-user `replication` o.ä.)

Sinnvoll für

- Sicherungen
- Hochverfügbarkeit

Streaming Replication funktioniert nur unidirektional.

- Nur an einem Datenbankserver dürfen Daten geändert werden: *Master* (read-write).
- Die zweite Datenbank erzeugt selbst kein WAL: *Replica* (read-only).

- Beide Datenbanken können lesend genutzt werden.
  - quasi horizontale Skalierung
- Backups können auch von der Replica gezogen werden.
- Die Replica kann ohne den Master weiterlaufen.
- Eine Replica kann ersatzweise zum Master befördert werden.

- Die Replica hat zwar konsistente aber ggf. verspätete Daten: *Replication Lag*.
  - Dagegen hilft synchrone Replizierung, das bringt aber wiederum Leistungseinbußen.
  - Bei einer Beförderung muss entschieden werden, wie viel *Lag* akzeptabel ist.
- Wird die Replica auch zum Master befördert und akzeptiert schreibende Verbindungen, können die Datenbestände auseinanderdriften (sog. *Split Brain* Szenario).

Wird der Master durch eine Netzwerkpartition o.ä. unerreichbar, kann man eine Replica befördern.

Der alte Master läuft ggfs. ungehindert weiter

- muss spätestens angehalten werden sobald der Host wieder erreichbar ist
- *Shoot The Other Node In The Head*: STONITH

Man kann theoretisch beliebig viele Replicas betreiben.

- $R : \{r_1, .., r_n\} \qquad n \in \mathbb{N}$

Welche davon befördern?

- $R_c : \{r \in R | r \text{ is healthy } \wedge replication\_lag(r) = min(replication\_lag(r_i))\}$

Problem: beliebig viele Replicas können die Bed. erfüllen.

- n-fach split brain, wenn alle gleichzeitg erkennen, dass kein Master mehr existiert
- wir benötigen einen gegenseitigen Ausschluss, wenn wir die Beförderung automatisieren wollen

verteilter Konsens

Mehrere Knoten wählen einen Anführer.

- Anführer muss sich periodisch melden, sonst erfolgt Neuwahl.

Alle Knoten haben den gleichen Datenstand.

- Änderungen werden von allen Knoten bestätigt.

Die Minderheit der Knoten kann ausfallen.

- Konsens bleibt möglich

In Patroni wird dies DCS (Distributed Configuration Store) genannt.
Verschiedene Optionen:

- etcd
- consul
- zookeeper
- exhibitor
- kubernetes
- aws

- oft genutzt
- in den Repos der verbreiteten Distributionen
- Konfiguration nicht ganz einfach
  - die Alternativen sind kaum besser
- Rollen mit Passwörtern
- Client- & Serverzertifikate

`/etc/default/etcd` (Debianoids)
`/etc/etcd/etcd.conf` (RHELoids)

```
ETCD_NAME="host1"

ETCD_LISTEN_PEER_URLS="http://0.0.0.0:2380"
ETCD_LISTEN_CLIENT_URLS="http://0.0.0.0:2379"

ETCD_INITIAL_ADVERTISE_PEER_URLS="http://10.0.0.1:2380"
ETCD_ADVERTISE_CLIENT_URLS="http://10.0.0.1:2379"

ETCD_INITIAL_CLUSTER="host1=http://10.0.0.1:2380,\
                      host2=http://10.0.0.2:2380, host3=http://10.0.0.3:2380"

ETCD_INITIAL_CLUSTER_STATE="new"
ETCD_INITIAL_CLUSTER_TOKEN="qwkyglxgebju"
```

analog für host2, host3

Auf allen etcd-Knoten gleichzeitg ausführen...

```
systemctl enable etcd
systemctl start etcd
etcdctl cluster-health
journalctl -b -u etcd
```

Wenn der Versuch fehlschlug (weil etcd denkt, es gehöre zu einem anderen Cluster als dem in der Config definierten) kann man so wieder von vorne anfangen:

```
systemctl stop etcd
rm -rf /var/lib/etcd/*
```

Gleiches Vorgehen hilft gegen die nutzlose und automatisch gestartete Standardconfig unter Debian/Ubuntu.

Ein single-node etcd-Cluster ist sinnlos.

Patroni

Unter Ubuntu >= 18.04, Debian >= 9:

```
apt install patroni \
            python3-psycopg2 python3-etcd
```

Unter RHEL/CentOS:

```
curl https://github.com/cybertec-postgresql/\
     patroni-packaging/releases/download/1.5.6-1/\
     patroni-1.5.6-1.rhel7.x86_64.rpm
yum localinstall patroni-*.rpm
```

Unter [hier Lieblingsdistro einfügen]:

```
pip3 install wheel setuptools psycopg2-binary python-etcd
pip3 install patroni
```

patroni (executable)

- überwacht und dirigiert einen Postgres Prozess
- schreibt Zustand in DCS
- befördert oder degradiert Postgres
- erzeugt ein neues Datenverzeichnis (*bootstrap*) oder zieht eine Kopie von einem anderen Patroni (*replica imaging*)
- schreibt die postgresql.conf
  - bei Änderung im DCS
  - bei Beförderung und Degradierung
- verwaltet die Replikation, bei Bedarf auch mit Replication Slots

Patroni config

- geschrieben in *YAML*
- eine config pro Patroni Instanz
- wir benutzen i.d.R. auch ein systemd Service-File pro Instanz

```
scope: {{patroni_cluster_name}}
namespace: /service/
name: {{ansible_hostname}}

restapi:
  listen: 0.0.0.0:8008
  connect_address: {{ansible_default_ipv4.address}}:8008
etcd:
  hosts: {{etcd_hosts_list}} # 127.0.0.1:2379
  protocol: {{proto}}
```

Patroni config ctd.

```
  dcs:
    ttl: 30
    loop_wait: 10
    retry_timeout: 10
    maximum_lag_on_failover: 1048576  # 1MB
#    master_start_timeout: 300
#    synchronous_mode: false
    postgresql:
      use_pg_rewind: true
      use_slots: true
      parameters:
        shared_preload_libraries: pg_stat_statements,auto_explain
        shared_buffers: {{shared_buffers}}
        work_mem: 32MB
        max_wal_size: {{max_wal_size}}
```

Patroni config ctd.

```
initdb:
- encoding: UTF8
- data-checksums
- waldir: {{custom_wal_dir}}
pg_hba:
- host replication replicator 0.0.0.0/0 md5
- host all all 0.0.0.0/0 md5
```

```
postgresql:
  listen: 0.0.0.0:{{pg_port}}
  connect_address: {{ansible_default_ipv4.address}}:{{pg_port}}
  data_dir: {{postgres_data_dir}}
  bin_dir: /usr/pgsql-{{postgres_version}}/bin
  pgpass: {{patroni_config_dir}}/pgpass0
  authentication:
    replication:
      username: replicator
      password: {{replication_db_password}}
    superuser:
      username: postgres
      password: {{postgres_db_password}}
```

Patroni config ctd.

CYBERTEC
DATA SCIENCE & POSTGRESQL

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

```
systemctl enable patroni
systemctl start patroni
patronictl -c /etc/patroni/config.yml
```

Bei Fehler ggf. Patroni config korrigieren, dann Service restarten.

1. Postgres `bin_dir` falsch

- produziert einen File not found stacktrace (`pg_ctl` u.a.)

2. etcd/consul unerreichbar
3. Datenbanken unerreichbar (`pg_hba.conf`, `listen_port`, firewall, Passwörter)

- alle müssen mit allen kommunizieren können (Postgres zu Postgres, Patroni zu etcd, Patroni zu Patroni)

4. Config wird nicht übernommen oder überschrieben (in postgresql.conf)

- Der richtige Weg wäre, die Config per `patronictl edit-config` oder per http PUT zu verändern. Andernfalls kann man auch noch die postgresql.base.conf nutzen. Gewisse Parameter sollten auch ausschließlich von Patroni verwaltet werden, etwa `hot_standby`.

5. Patroni kann die Datenbank nach einem reboot nicht mehr verwalten. (postmaster.pid existiert bereits)

- Überflüssiger Service, der vor Patroni die Gelegenheit bekommt, Postgres im Standarddir zu starten. (*Warum, Debian??*)

6. Waiting for master to bootstrap

- Der Cluster lief schon einmal, ist dann aber abgestürzt.
- Das Replication Lag sämtlicher Replicas ist zu groß, um befördert zu werden.
  - Ggf. `maximum_lag_on_failover` in der Patroni config anpassen, wenn trotzdem ein Failover gewünscht ist.
- Der alte Master muss reanimiert werden.

Versucht man, einen Cluster in einem zuvor genutzten Namespace zu starten, dann muss man das entsprechende Verzeichnis aus etcd/consul löschen oder ein `patronictl remove` (für den alten Namespace) durchführen.

7. Replica holt nie auf

- Oft nach reboots.
- Die replica war lange offline, der Master hat (von der Replica noch benötigte) WAL Files gelöscht.
- Mehr `wal_keep_segments` oder replication slots benutzen!

8. Patroni cluster member meldet: start failed

- Vermutlich ist irgendetwas in der postgresql.conf falsch geschrieben.
- `patronictl edit-config` zum korrigieren.
  Deshalb: Konfigurationsänderungen immer erst an einer Replica ausprobieren (`patronictl restart clustername replicaname`)

- Überwachen eines Patroni clusters
  - `patronictl list`
- Einsehen und Ändern der config
  - `patronictl show-config` / `patronictl edit-config`
- Switchover
  - `patronictl switchover`
- Wartungsmodus
  - `patronictl pause` / `patronictl resume`
- Replicas neu initialisieren
  - `patronictl reinit`
- Clusterinformationen komplett aus DCS entfernen
  - `patronictl remove`

CYBERTEC
DATA SCIENCE & POSTGRESQL

patronictl muss wissen, wie DCS zu erreichen ist.

`patronictl.yml`:

```
dcs_api: etcd://127.0.0.1:2379
namespace: /service/
#scope: pgcluster
```

- `patronictl -c patronictl.yml list pgcluster`
- `patronictl -c patronictl.yml list pgcluster2`

Das steht (mit scope) auch so in der config für `patroni`...

- `patronictl -c /etc/patroni_pgcluster/config.yml list`

patronictl list


```
[root@centos-server-101 ~]# patronictl -c /etc/patroni_pgcluster/config.yml list
+-----------+-------------------+---------------+--------+---------+----+-----------+
|  Cluster  |       Member      |      Host     |  Role  |  State  | TL | Lag in MB |
+-----------+-------------------+---------------+--------+---------+----+-----------+
| pgcluster | centos-server-101 | 192.168.178.35 |        | running |  4 |         0 |
| pgcluster | centos-server-102 | 192.168.178.36 | Leader | running |  4 |         0 |
+-----------+-------------------+---------------+--------+---------+----+-----------+
```

patronictl restart

CYBERTEC
DATA SCIENCE & POSTGRESQL

```
[root@centos-server-101 ~]# patronictl -c /etc/patroni_pgcluster/config.yml \
                                    restart pgcluster centos-server-101
+-----------+-------------------+----------------+--------+---------+----+-----------+-----------------+
|  Cluster  |       Member      |      Host      |  Role  |  State  | TL | Lag in MB | Pending restart |
+-----------+-------------------+----------------+--------+---------+----+-----------+-----------------+
| pgcluster | centos-server-101 | 192.168.178.35 |        | running |  4 |         0 |        *        |
| pgcluster | centos-server-102 | 192.168.178.36 | Leader | running |  4 |         0 |        *        |
+-----------+-------------------+----------------+--------+---------+----+-----------+-----------------+
Are you sure you want to restart members centos-server-101? [y/N]: y
Restart if the PostgreSQL version is less than provided (e.g. 9.5.2)  []:
When should the restart take place (e.g. 2015-10-01T14:30)  [now]:
Success: restart on member centos-server-101
```

Patroni ReST API

CYBERTEC
DATA SCIENCE & POSTGRESQL

```
[root@centos-101 ~]# curl -s http://192.168.178.36:8008/patroni | jq .
```


```
{
  "state": "running",
  "postmaster_start_time": "2019-05-03 11:34:44.215 CEST",
  "role": "master",
  "server_version": 100007,
  "cluster_unlocked": false,
  "xlog": {
    "location": 50332352
  },
  "timeline": 2,
  "replication": [
    {
      "usename": "replicator",
      "application_name": "centos-101",
      "client_addr": "192.168.178.35",
      "state": "streaming",
      "sync_state": "async",
      "sync_priority": 0
    }
  ],
  "database_system_identifier": "6686123827190300460",
  "patroni": {
    "version": "1.5.6",
    "scope": "pgcluster"
  }
}
```

Was steht im DCS?

Was steht im DCS?

CYBERTEC
DATA SCIENCE & POSTGRESQL

- Welche Member gibt es gerade?
  - Wo sind die zu erreichen?
  - Sind die lebendig?
  - Bei welcher WAL Nummer sind die?
- Wer ist der Leader (Master)?
  - Wann hat sich der Leader das letzte mal gemeldet?

Was steht im DCS?

CYBERTEC
DATA SCIENCE & POSTGRESQL

```
[root@centos-101 ~]# curl -s http://127.0.0.1:2379/v2/keys/service/ | jq . \
                          | grep -e "\{" -e "\}" -e "\[" -e "\]" -e node -e key -e value -e dir
```


```
{
  "node": {
    "key": "/service",
    "dir": true,
    "nodes": [
      {
        "key": "/service/pgcluster",
        "dir": true,
      },
      {
        "key": "/service/pgcluster2",
        "dir": true,
      }
    ],
  }
}
```

Was steht im DCS? ctd.

CYBERTEC
DATA SCIENCE & POSTGRESQL

```
[root@centos-101 ~]# curl -s http://127.0.0.1:2379/v2/keys/service/pgcluster | jq . \
                           | grep -e "\{" -e "\}" -e "\[" -e "\]" -e node -e key
```


```
{
  "node": {
    "key": "/service/pgcluster",
    "nodes": [
      {
        "key": "/service/pgcluster/config",
      },
      {
        "key": "/service/pgcluster/history",
      },
      {
        "key": "/service/pgcluster/initialize",
      },
      {
        "key": "/service/pgcluster/leader",
      },
      {
        "key": "/service/pgcluster/members",
      },
      {
        "key": "/service/pgcluster/optime",
      }
    ],
  }
}
```

Patroni wird gestartet...

- Existiert ein Verzeichnis für cluster in DCS?
- Wenn nicht, anlegen, init key dazu.
- Der Patroni, der den init key anlegt, darf initdb oder custom bootstrap ausführen.

leader key im DCS

- Enthält als value den Namen des Leaders.
- Schlüssel läuft nach einiger Zeit automatisch ab.
- Patroni am Leader muss periodisch Lebendigkeit prüfen, dann die TTL neu setzen.

```
{
  "node": {
    "key": "/service/pgcluster/leader",
    "value": "centos-102",
    "expiration": "2019-05-03T10:21:56.234644744Z",
    "ttl": 29,
  }
}
```

So wird STONITH implizit umgesetzt:

- Ist der leader von den übrigen Knoten getrennt, so kann er den key nicht aktualisieren und muss dann degradiert werden.
- Im Grunde genommen muss man sich hierbei darauf verlassen, dass Patroni den alten Master dann wirklich abschaltet. Für Paranoide gibt es auch noch die Möglichkeit, die TTL eines watchdog Devices, ähnlich wie den leader key, periodisch zu aktualisieren. Bleibt die Aktualisierung aus (wenn z.B. Patroni unvermittelt stirbt), wird der Host angehalten.

Wenn kein leader key da:

- alle gesunden Replicas mit hinreichend geringem replication lag versuchen, den leader key an sich zu reißen.
- Konsensprotokoll verhindert Konflikte.
  - Zu jedem key ist der value über den gesamten DCS gleich.
  - Etcd bietet beispielsweise einen konditionellen PUT, *setze den key nur, wenn er noch nicht gesetzt ist*.

Der Gewinner des leader race wird zum Master befördert.

members dir im DCS


```
{
  "node": {
    "key": "/service/pgcluster/members",
    "dir": true,
    "nodes": [
      {
        "key": "/service/pgcluster/members/centos-102",
        "value": "{\"conn_url\":\"postgres://192.168.178.36:5432/postgres\",\"a
        "expiration": "2019-05-03T10:25:16.242427345Z",
        "ttl": 27,
      },
      {
        "key": "/service/pgcluster/members/centos-101",
        "value": "{\"conn_url\":\"postgres://192.168.178.35:5432/postgres\",\"a
        "expiration": "2019-05-03T10:25:16.249577163Z",
        "ttl": 27,
      }
    ],
  }
}
```

member details im DCS

CYBERTEC
DATA SCIENCE & POSTGRESQL

```
[root@centos-101 ~]# curl -s \
                          http://127.0.0.1:2379/v2/keys/service/pgcluster/members/centos-102 \
                          | jq -rc .node.value | jq

{
  "conn_url": "postgres://192.168.178.36:5432/postgres",
  "api_url": "http://192.168.178.36:8008/patroni",
  "state": "running",
  "role": "master",
  "xlog_location": 50332352,
  "timeline": 2
}
```

cluster config im DCS

CYBERTEC
DATA SCIENCE & POSTGRESQL

```
[root@centos-101 ~]# curl -s http://127.0.0.1:2379/v2/keys/service/pgcluster/config \
                          | jq -rc .node.value | jq .
```


```
{
  "ttl": 30,
  "loop_wait": 10,
  "retry_timeout": 10,
  "maximum_lag_on_failover": 1048576,
  "postgresql": {
    "use_pg_rewind": true,
    "use_slots": true,
    "parameters": {
      "effective_cache_size": "1469MB",
      "maintenance_work_mem": "320MB",
      "max_wal_size": "2GB",
      "pg_stat_statements.track": "all",
      "shared_buffers": "459MB",
      "shared_preload_libraries": "pg_stat_statements,auto_explain",
      "work_mem": "32MB"
    }
  }
}
```


Hier könnte man auch PUTten...

Client-Konfiguration

```
[root@centos-101 ~]# psql -U postgres $(curl -s \
http://127.0.0.1:2379/v2/keys/service/pgcluster/members/centos-102 \
| jq -rc .node.value | jq -r .conn_url)
Password for user postgres:
psql (10.7)
Type "help" for help.

postgres=# \q
```

- Man könnte sich natürlich auch zuerst mit curl den `leader` value holen und dann dazu die `conn_url`, dann erreicht man immer den Master

```
[root@centos-101 ~]# psql -U postgres \
$(curl -s http://127.0.0.1:2379/v2/keys/service/pgcluster/members/\
$(curl -s http://127.0.0.1:2379/v2/keys/service/pgcluster/leader\
| jq -r .node.value)| jq -rc .node.value | jq -r .conn_url)
Password for user postgres:
psql (10.7)
Type "help" for help.

postgres=# \q
```

Man kann libpq mehrere Hosts liefern, es wird dann der erste erreichbare ausgewählt.

- ab Postgres 10
- `psql -d "host=host1,host2 target_session_attrs=read-write"`

https://github.com/cybertec-postgresql/vip-manager

- eine vip-manager Instanz pro Patroni Instanz
- schaut, ob der zugehörige Patroni gerade Leader ist
  - wenn ja
    - war vorher leader? nichts tun!
    - war vorher kein leader? `ip addr add VIP dev IFACE`, gratuitous ARP reply senden!
  - wenn nein
    - war vorher kein leader? nichts tun!
    - war vorher leader? `ip addr del VIP dev IFACE`

Integration in diverse Hostingplattformen möglich dank modularem Aufbau des Projekts.

Patroni for runaways (Patroni für Fortgeschrittene)

/etc/pgbackrest.conf auf jedem cluster member.

```
[global]
repo1-host=192.168.178.37
[pgcluster]
pg1-path=/var/lib/pgsql/10/data_pgcluster
pg1-port=5432
```

`/etc/pgbackrest.conf` auf dem *repository* host.

```
[global]
repo1-path=/var/lib/pgbackrest
backup-standby=y
start-fast=y
[pgcluster]
pg1-host=192.168.178.35
pg1-path=/var/lib/pgsql/10/data_pgcluster
pg1-port=5432
pg2-host=192.168.178.36
pg2-path=/var/lib/pgsql/10/data_pgcluster
pg2-port=5432
```

Weitere Replicas einfach mit pg*n*-host usw. angeben.

pgBackRest sucht sich beim Sichern einfach einen der (erreichbaren) Hosts aus. Wenn `backup-standby` aktiviert ist, werden die meisten Daten von einer Replica gezogen.

Damit beim `reinit` oder neuerlichen Hinzufügen eines Patroni cluster members kein pgbasebackup vom Master gezogen werden muss, können wir auch pgBackRest nutzen.

Patroni config (Ausschnitt):

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

Für den Fall, dass `pgbackrest` nicht funktioniert würde hier immer noch `pg_basebackup` als Fallback genutzt, allerdings gedrosselt.

Wir können auch einen Patroni Cluster von einem vorhandenen Backup bootstrappen. Patroni Config (Ausschnitt):

```
bootstrap:
  method: pgbackrest
  pgbackrest:
    command: /etc/patroni_pgcluster2/bootstrap_pgbackrest.sh
    keep_existing_recovery_conf: False
    recovery_conf:
      restore_command: '/usr/bin/pgbackrest \
                  --pg1-path=/var/lib/pgsql/10/data_pgcluster2 \
                  --log-level-file=off --stanza=pgcluster archive-get %f "%p"'
      recovery_target_action: promote
      recovery_target_timeline: latest
      recovery_target_time: '2019-05-08 15:40'
```

Da Patroni dem command ein paar Parameter mitgibt, mit denen pgBackRest nichts anfangen kann, brauchen wir ein kleines Skript...

/etc/patroni_pgcluster2/bootstrap_pgbackrest.sh :

```
#!/usr/bin/bash
mkdir /var/lib/pgsql/10/data_pgcluster2
chmod 0700 /var/lib/pgsql/10/data_pgcluster2
/usr/bin/pgbackrest --pg1-path=/var/lib/pgsql/10/data_pgcluster2 \
        --stanza=pgcluster --type=time '--target=2019-05-08 15:40' restore
```

Abschluss

Patroni ist ein tolles Tool.
Man kann sehr schnell einen robusten HA-Cluster aufziehen.

Man kann aber auch sehr komplexe Szenarien erfüllen.

- Das resultiert schnell in ganz schön viel Konfigurationsaufwand!
- Wir empfehlen, größere Projekte mit vielen Clustern deswegen per Ansible umzusetzen.
- Zum Glück habe ich schon knapp 2000 Zeilen Playbooks und knapp 500 Zeilen Templates geschrieben.

Danke für Ihre Aufmerksamkeit.

Gibt es noch Fragen?

- Diskussionen gerne am Stand.
- Wenn Sie Python / C / Go / ansible / YAML / bash mögen, machen wir Ihnen ein Angebot, dass Sie nicht ablehnen können!

Kontakt

Meine Mail: julian.markwort@cybertec.at
Ich auf Twitter: jukklies

Cybertec Schönig & Schönig GmbH
Gröhrmühlgasse 26
2700 Wiener Neustadt
Austria

Email: office@cybertec.at
www.cybertec-postgresql.com

Cybertec auf Twitter: @PostgresSupport