zalando

POSTGRESQL HA IN THE CLOUDS WITH DOCKER, SPILO & PATRONI

PGCON, OTTAWA

2017-05-26

ALEXANDER KUKUSHKIN

ABOUT ME

Alexander Kukushkin

Database Engineer @ZalandoTech
Email: alexander.kukushkin@zalando.de
Twitter: @cyberdemn

zalando

ZALANDO AT A GLANCE

~3.6 billion EURO
net sales 2016

~165 million
visits per month

~20 million
active customers

~200,000
product choices

>1,500
brands

15
countries

>12,000
employees in Europe

50%
return rate across all categories

zalando

ZALANDO TECHNOLOGY

BERLIN

DORTMUND

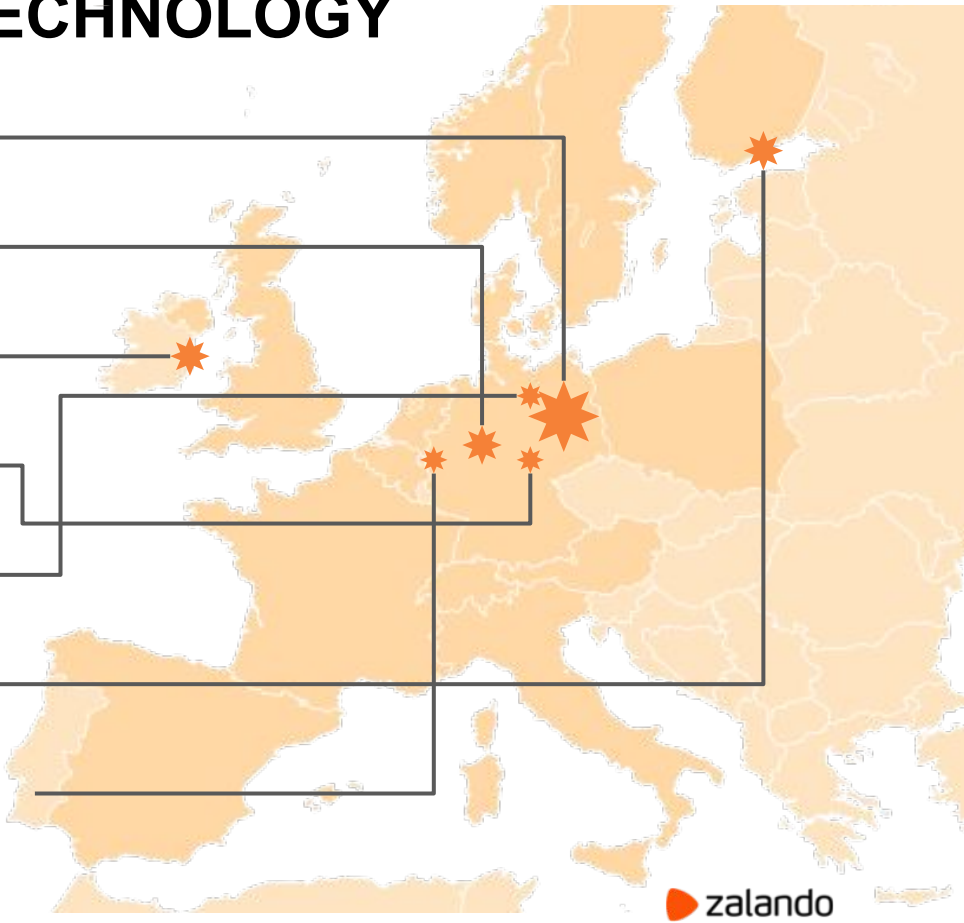
DUBLIN

ERFURT

HAMBURG

HELSINKI

MÖNCHENGLADBACH



zalando

ZALANDO TECHNOLOGY

- > 150 databases in DC
- > 130 databases on AWS
- > 1600 tech employees
- We are hiring!

POSTGRESQL

The world’s most advanced open-source database

- Rock-solid by default
- Transactional DDL
- Standard-compliant modern SQL
- Blazing performance
- PostgreSQL is a community

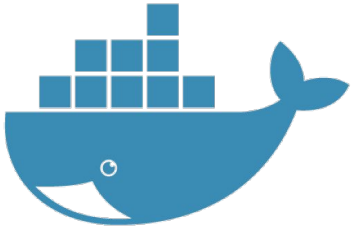
RADICAL AGILITY AND AUTONOMOUS TEAMS

Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations

Conway's Law

AWS, STUPS AND MICROSERVICES

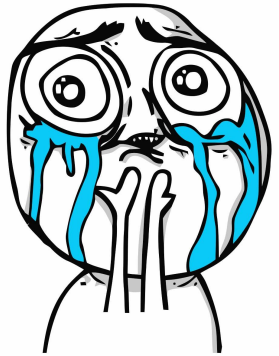
STUPS



- **Rules of Play:**
  - One AWS account per Team
  - Deployment with Docker
  - Managed SSH Access
  - Traceability of changes
- **Problems:**
  - Database must be located close to the Application
  - Migration of existing DB to AWS “without” downtime
  - Microservices != Micro databases
  - Teams don’t have experience in managing their databases

WHY NOT RDS?

- No superuser access
- No replication connection
- No custom extensions
- ..... put your own reason here



IF NOT RDS THEN:

- Run PostgreSQL on your own!
- On EC2 instances!
- In Auto Scaling Group!
- In Docker container!

WHY DOCKER?

- “**Rules of play**” (audit requirements)
- Rapid (and repeatable) deployment
- Nice packaging tool

WHY NOT DOCKER?

- One container per EC2 Instance
- We need to run more than 1 application per container
- Additional layer which could fail
- Nobody really runs production databases in Docker

  those days

POSTGRESQL ON AWS

We just need to solve a few problems:

- Simple and reliable automatic failover
- Automatic node provisioning/configuration
- Backup and disaster recovery

AUTOMATIC FAILOVER

“PostgreSQL does not provide the system software required to identify a failure on the primary and notify the standby database server.”

CC0 Public Domain

EXISTING AUTOMATIC FAILOVER SOLUTIONS

- Promote a replica when the master is not responding
  - Split brain/potentially many masters
- Use one monitor node to make decisions
  - Monitor node is a single point of failure
  - Former master needs to be killed (STONITH)
- Use multiple monitor nodes
  - Distributed consistency problem

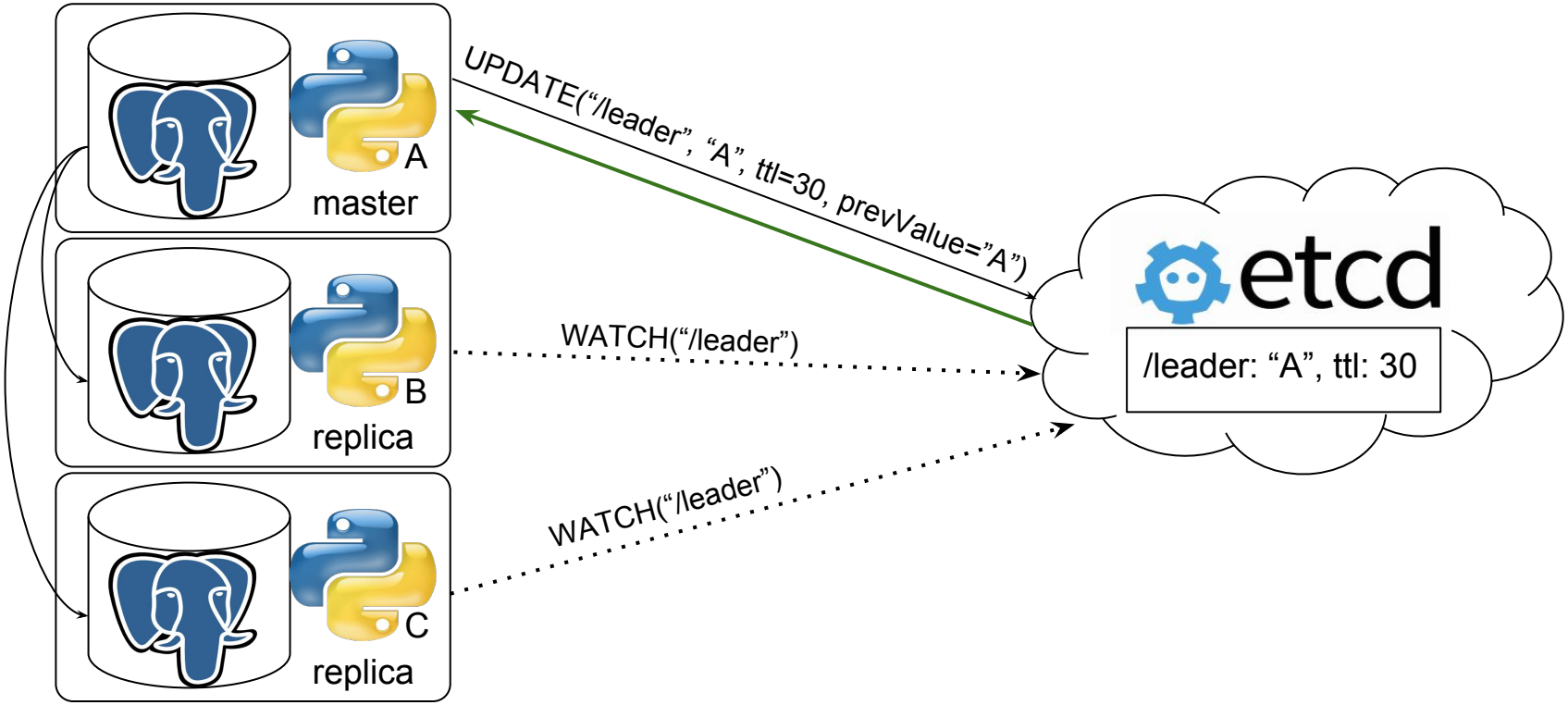
DISTRIBUTED CONSISTENCY PROBLEM

https://www.flickr.com/photos/kevandotorg

PATRONI APPROACH

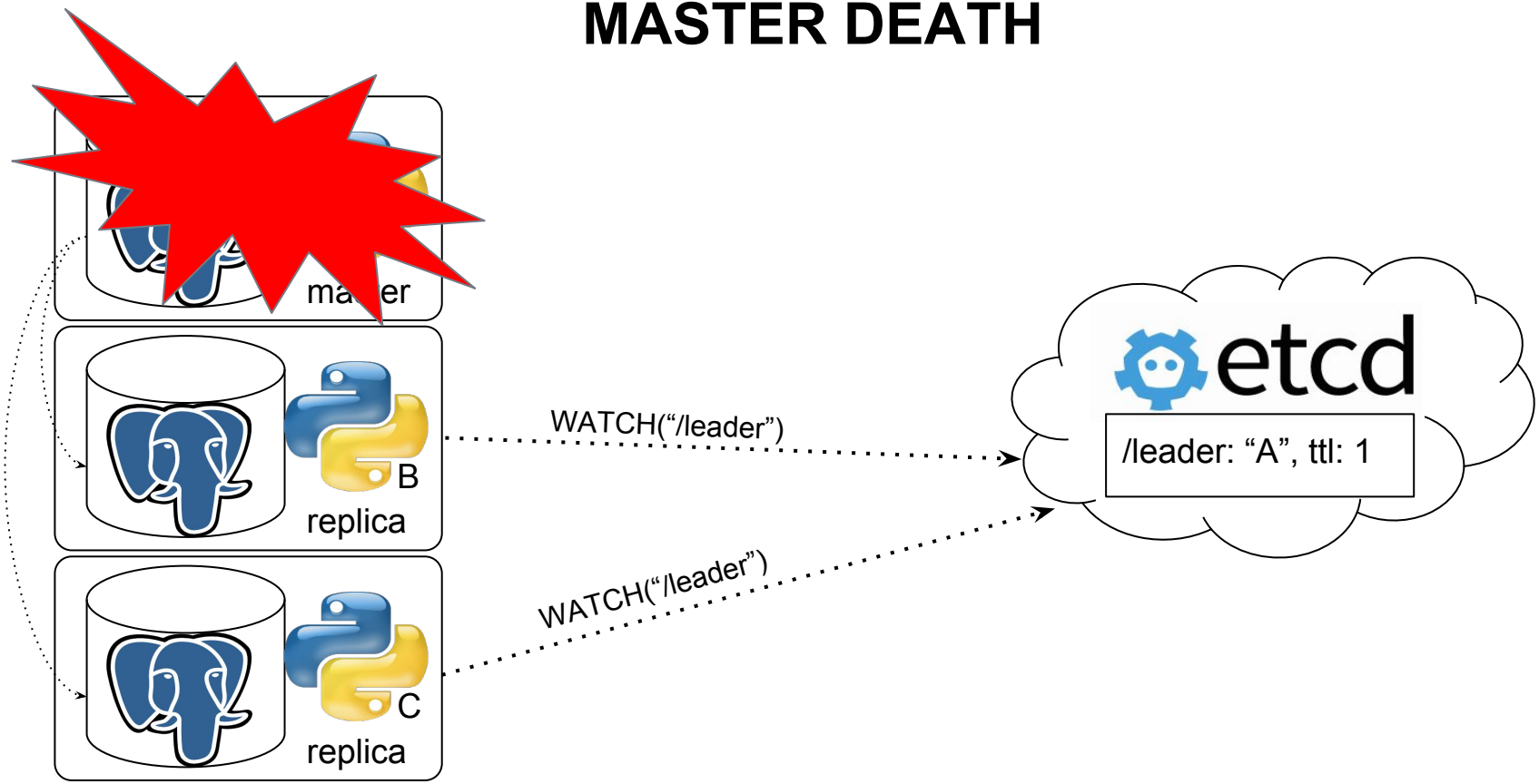
- Use Distributed Configuration System (DCS) for holding the Leader Lock
  - Etcd, Zookeeper or Consul
    - Built-in distributed consensus (RAFT, Zab)
- Run as a Master only if holding the Leader Lock
- Periodically renew the Leader Lock in DCS
- Leader Lock will expire if Master is dead

NORMAL FLOW



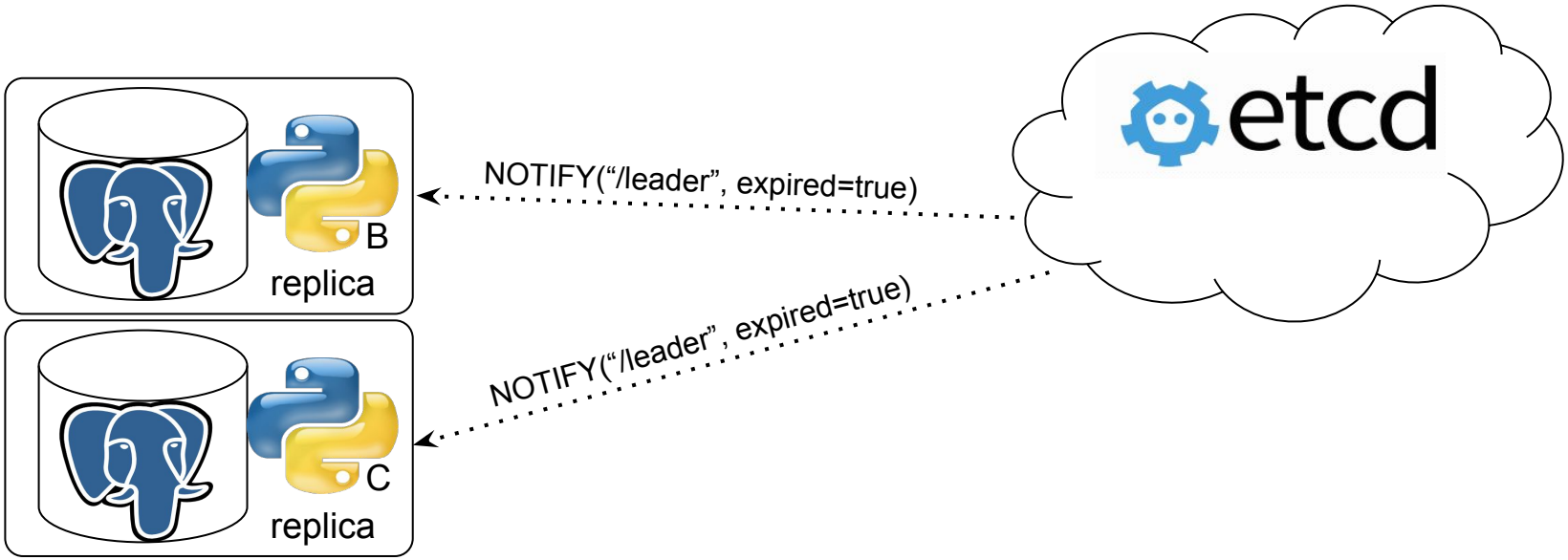
zalando

MASTER DEATH



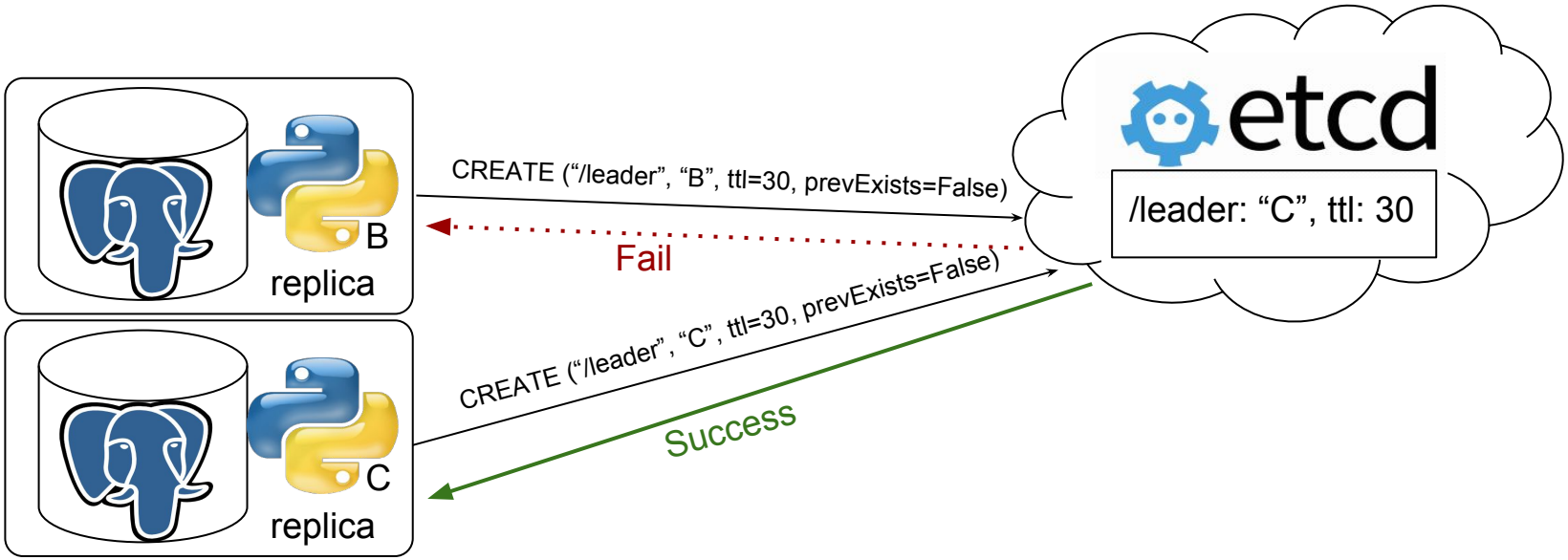
zalando

MASTER DEATH DETECTION



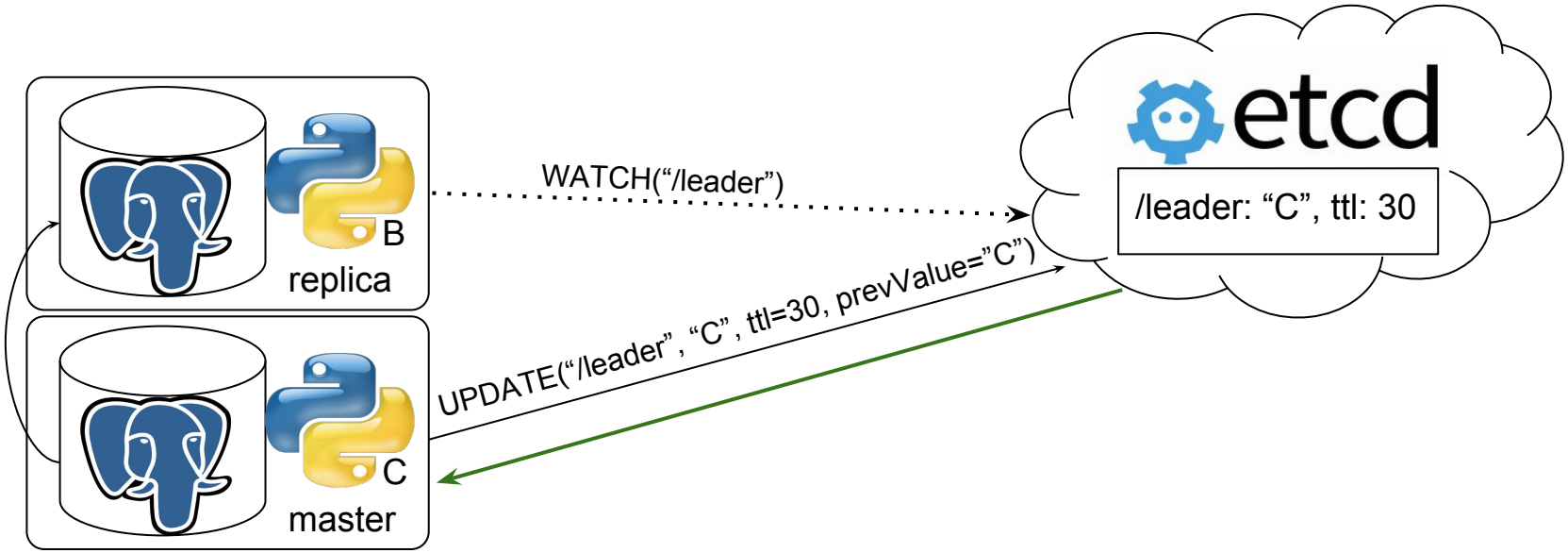
zalando

LEADER RACE



zalando

NORMAL FLOW



zalando

DCS STRUCTURE

- /service/cluster-name/
  - config
  - initialize
  - members/
    - *dbnode1*
    - *dbnode2*
  - *leader*
  - optime/
    - leader

DCS STRUCTURE

- /service/cluster-name/
  - config — {"postgresql":{"parameters":{"max_connections":300}}}
  - initialize — ”6303731710761975832” (database system identifier)
  - members/
    - *dbnode1* — {"role":"replica","state":"running”,"conn_url":"postgres://172.17.0.2:5432/postgres"}
    - *dbnode2* — {"role":"**master**","state":"running”,"conn_url":"postgres://172.17.0.3:5432/postgres"}
  - *leader* — ***dbnode2***
  - optime/
    - leader — “67393608” # ← absolute wal positition

PATRONI FEATURES

- Automatic failover
- REST API (status, health-check, reinit, restart, reload, switchover)
- Manual and Scheduled Failover (switchover)
- patronictl (reinit, restart, reload, switchover, pause/resume)
- Callbacks (on_start, on_stop, on_restart, on_reload, on_role_change)
- Customizable replica creation methods
- Tags (nofailover, clonefrom, replicatefrom, noloadbalance, nosync)

PATRONI FEATURES

- pg_rewind
- Cascading replication
- Dynamic configuration
- Pause (maintenance) mode
- Data durability vs. High-Availability
- Synchronous mode
- Linux watchdog (coming soon)

DYNAMIC CONFIGURATION

- Store Patroni/PostgreSQL parameters in DCS and apply them dynamically
- Ensure identical configuration of the following parameters on all members:
  - ttl, loop_wait, retry_timeout, maximum_lag_on_failover
  - wal_level, hot_standby
  - max_connections, max_prepared_transactions, max_locks_per_transaction, max_worker_processes, track_commit_timestamp, wal_log_hints
  - wal_keep_segments, max_replication_slots
- Inform the user that PostgreSQL needs to be restarted (**pending_restart** flag)

BUILDING HA POSTGRESQL BASED ON PATRONI

- Client traffic routing
  - Patroni callbacks + Floating IP
  - confd + haproxy, pgbouncer
- Backup and recovery
  - WAL-E, barman, pgBackRest
- Monitoring
  - Nagios, zabbix, zmon

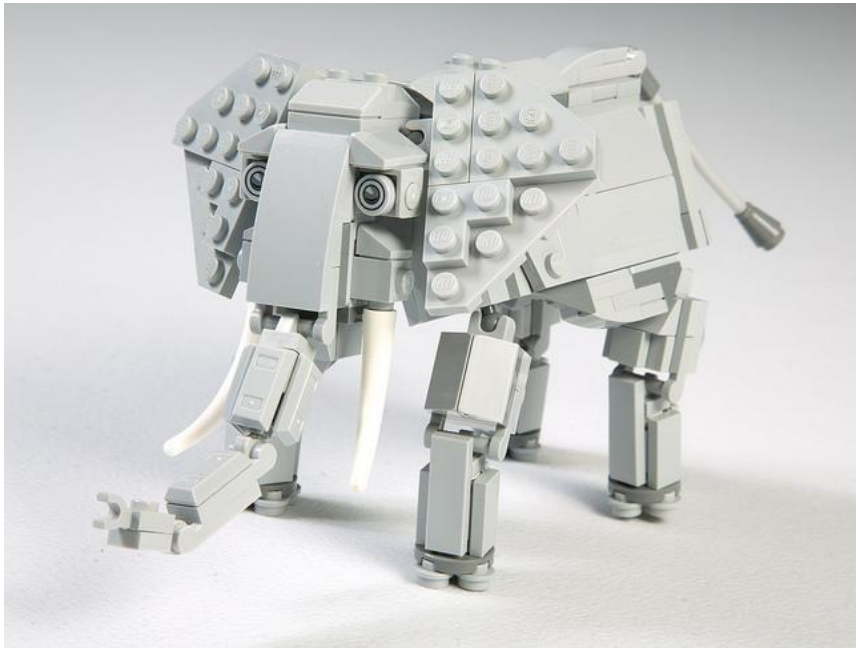


Image by flickr user https://www.flickr.com/photos/brickset/

SPILO:

Dockerfile

+ PostgreSQL

+ Patroni

+ WAL-E

+ Scripts to build Patroni configuration from environment

+ Custom extensions

SPILO:

+ Cron daemon
+ Different versions of PostgreSQL in the same image
+ callback script for Kubernetes
+ pam-oauth2
+ bg_mon

DEPLOYMENT

AWS DEPLOYMENT

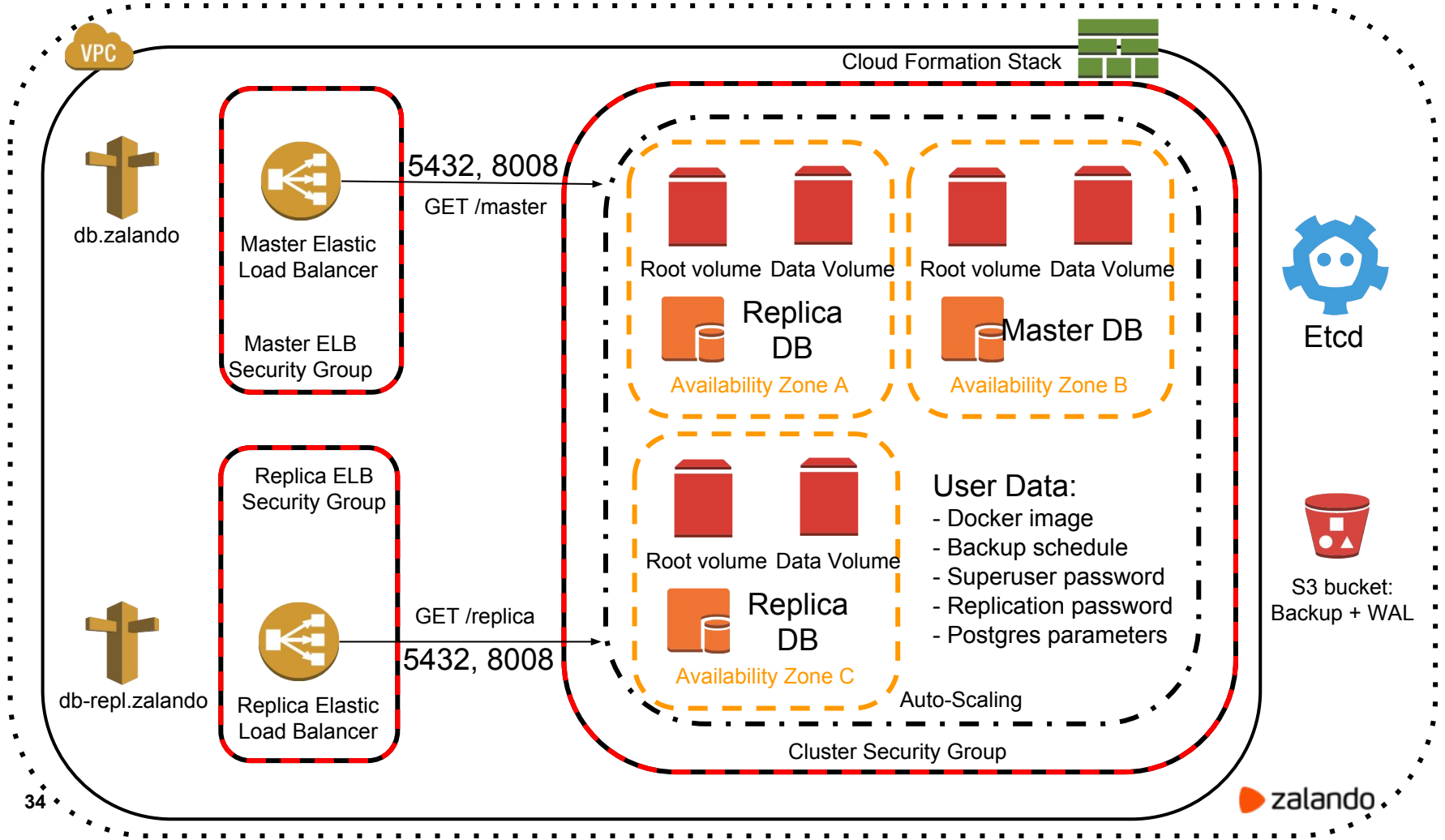
- One CloudFormation stack per PostgreSQL cluster
- Custom AMI (Taupage)
- One Docker container per EC2 Instance
- PGDATA + pg_xlog (pg_wal) on the same volume (usually EBS)
- Passwords are encrypted with KMS
- ELB for traffic routing

AWS DEPLOYMENT

$ senza create mycluster.yaml mycluster

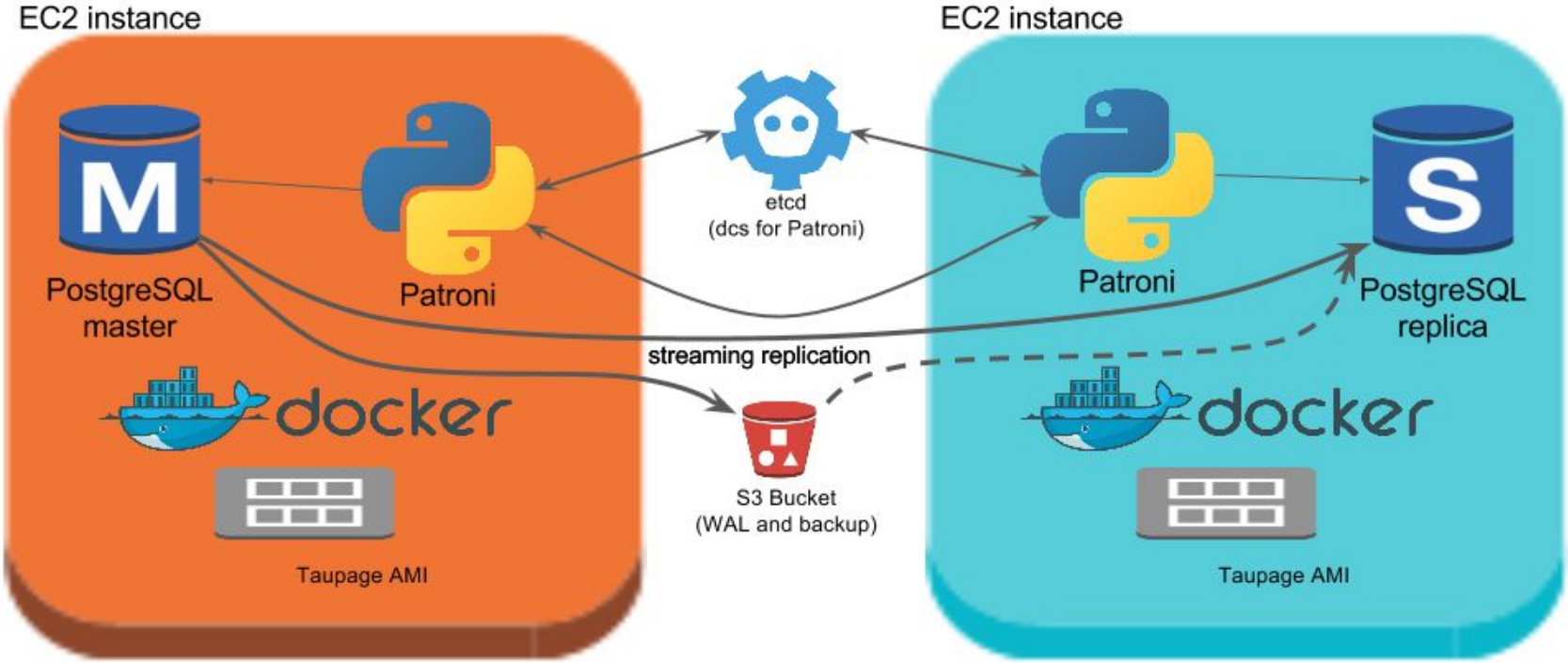
Senza is a command line tool that enables you to generate and execute AWS CloudFormation templates in a sane, simple way.

```
SenzaInfo:
  StackName: spilo
  Tags:
    - SpiloCluster: "{{Arguments.version}}"
    - Environment: live
SenzaComponents:
  - AppServer: # will create a launch configuration and auto scaling group
      Type: Senza::TaupageAutoScalingGroup
      AutoScaling: {Minimum: 3, Maximum: 3}
      InstanceType: m4.large
      BlockDeviceMappings:
        - DeviceName: /dev/xvdk
          Ebs: {VolumeSize: 1200, VolumeType: gp2}
      TaupageConfig:
        mounts:
          /home/postgres/pgdata: {partition: /dev/xvdk, filesystem: ext4}
        runtime: Docker
        source: registry.opensource.zalan.do/acid/spilo-9.6:1.2-p22
        ports: {5432: 5432, 8008: 8008}
        environment:
          SCOPE: "{{Arguments.version}}"
          PGPASSWORD_SUPERUSER: "aws:kms:<long encrypted string>"
          PGPASSWORD_STANDBY: "aws:kms:<long encrypted string>"
          BACKUP_SCHEDULE: "00 01 * * *"
```


VPC
Cloud Formation Stack
db.zalando
Master Elastic
Load Balancer
Master ELB
Security Group
5432, 8008
GET /master
Root volume
Data Volume
Replica
DB
Availability Zone A
Root volume
Data Volume
Master DB
Availability Zone B
Etcd
Replica ELB
Security Group
Root volume
Data Volume
Replica
DB
Availability Zone C
User Data:
- Docker image
- Backup schedule
- Superuser password
- Replication password
- Postgres parameters
S3 bucket:
Backup + WAL
db-repl.zalando
GET /replica
5432, 8008
Replica Elastic
Load Balancer
Auto-Scaling
Cluster Security Group

zalando

AWS DEPLOYMENT



zalando

PROBLEMS

- **Rules of play** and tooling were developed for stateless applications!
  - Upgrade Taupage AMI every two weeks
- Hard to upgrade EBS volume size / InstanceType:
  - It wasn’t possible to expand EBS volumes until Jan 2017
- In most cases replicas are doing nothing
- WAL-E wal-fetch/wal-prefetch is terribly slow

ALTERNATIVE:

Run instances with AutoRecovery (without ASG)

Pros

- EBS volume won't be destroyed
- One EC2 instance could be used to run multiple replicas of different clusters

Cons

- Fire drills are hard
- Need to manage EBS volumes and ELB on your own
- New tooling must be developed (not supported by **Stups**)

ANOTHER ALTERNATIVE:

Run it on Kubernetes

- **StatefulSets** + **volumeClaimTemplates** for volumes
- Use Kubernetes **Services** + **Endpoints** for traffic routing
- Use Patroni callbacks for updating **Endpoint** and **Pod** labels
  - Update **Endpoint** when **Pod** becomes Master
  - Label every **Pod** with its current **role**
- **Service** with **labelSelector role=replica** for read-only queries

PROBLEMS WITH KUBERNETES

- Only a few parameters in a **StatefulSet** could be changed:
  - **replicas** and **spec.template.containers**
- Changes are not propagated to running **Pods**
- **Kubernetes** can kill master during the downscaling
- **volumeClaimTemplates** can’t be changed:
  - Kubernetes doesn’t provide tools to extend volumes

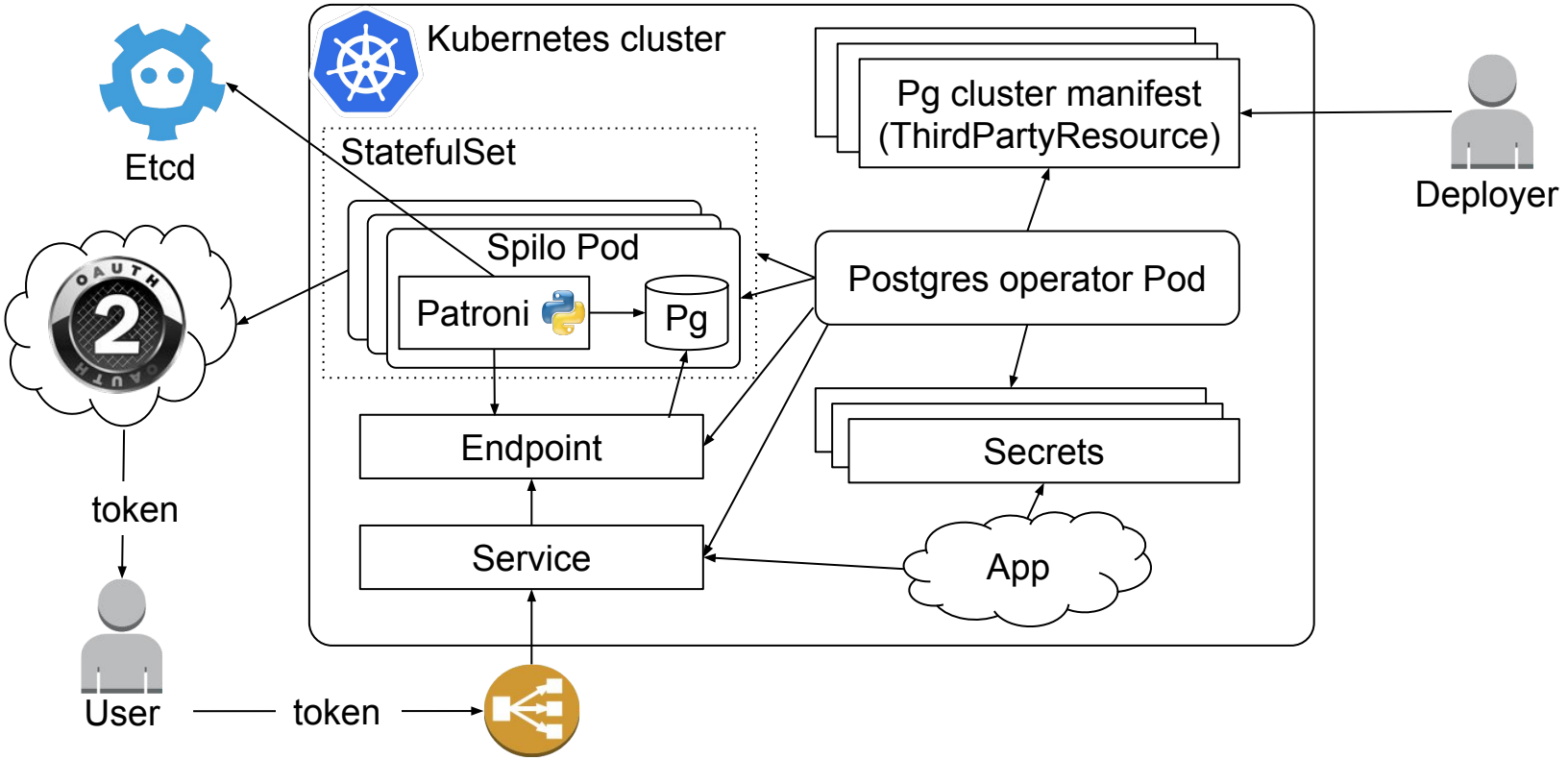
HOW TO DEPLOY ON KUBERNETES

- kubectl create -f your-cluster.yaml
- Use Patroni helm chart
- Use postgres-operator

INTRODUCING POSTGRES-OPERATOR

- Creates **ThirdPartyResource Postgresql** and watches it
- When new **Postgresql** object is created - deploys a new cluster
  - Creates **Secrets**, **Endpoints**, **Services** and **StatefulSet**
- When **Postgresql** object is updated - updates **StatefulSet**
  - and does a rolling upgrade
- Periodically syncs running clusters with the manifests
- When **Postgresql** object is deleted - cleans everything up

KUBERNETES DEPLOYMENT



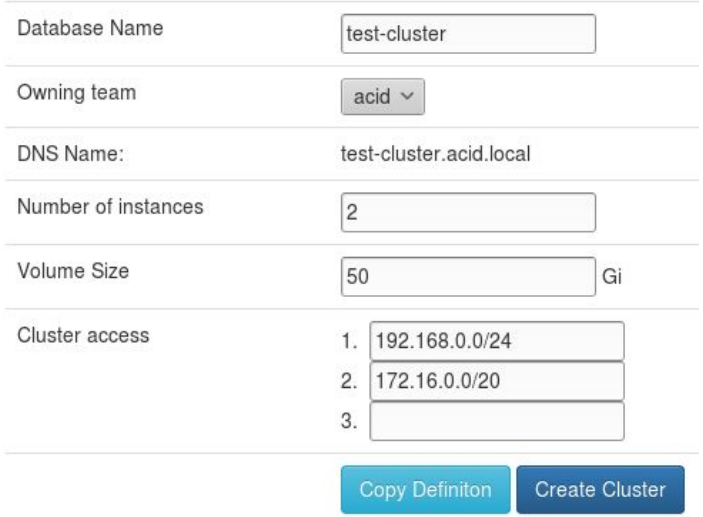
zalando

DEPLOYMENT WITH OPERATOR

Cluster YAML definition

```yaml
kind: "Postgresql"
apiVersion: "acid.zalan.do/v1"

metadata:
    name: "acid-test-cluster"

spec:
    teamId: "acid"
    volume:
        size: "50Gi"
    numberOfInstances: 2

    postgresql:
        version: "9.6"

    allowedSourceRanges:
    # IP ranges to access your cluster go here
    - 192.168.0.0/24
    - 172.16.0.0/20
```

Cluster configuration



zalando

CLUSTER STATUS

Cluster YAML definition

```yaml
apiVersion: acid.zalan.do/v1
kind: Postgresql
metadata:
  creationTimestamp: '2017-05-03T11:53:17Z'
  labels:
    team:acid
  name: acid-test-cluster
  namespace: default
spec:
  allowedSourceRanges:
    - 192.168.0.0/24
    - 172.16.0.0/20
  numberOfInstances: 2
  postgresql:
    version: '9.6'
  teamId: acid
  volume:
    size: 50Gi
status: Running
```

Checking status of Cluster

PostgreSQL ready: **test-cluster.acid.local** (DNS may be slow)

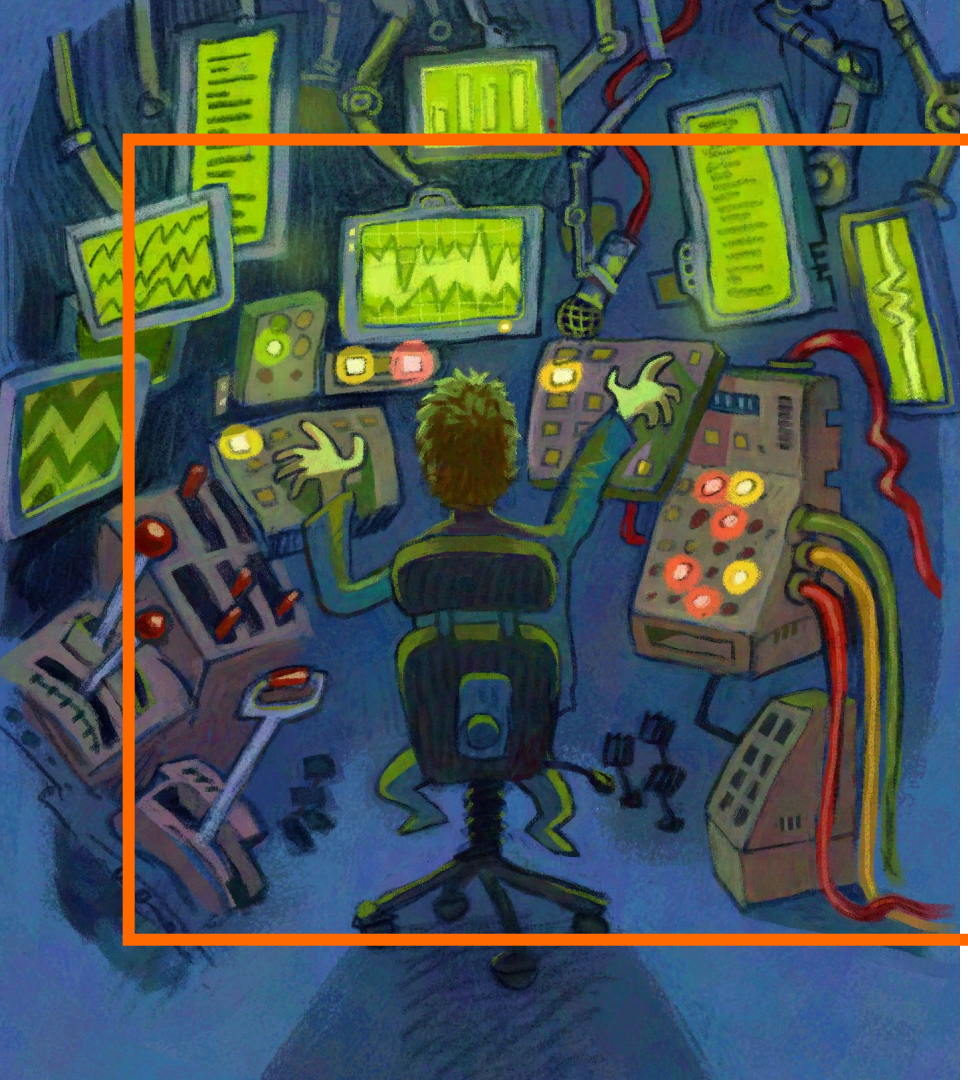
PostgreSQL master available, label is attached

First PostgreSQL cluster container spawned

StatefulSet created

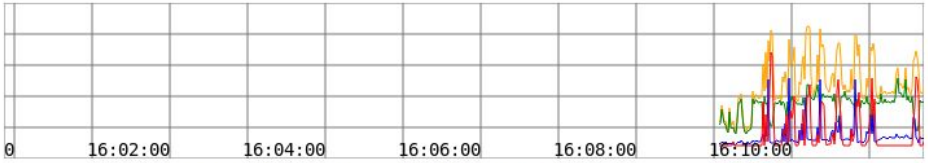
PostgreSQL 3rd party object created

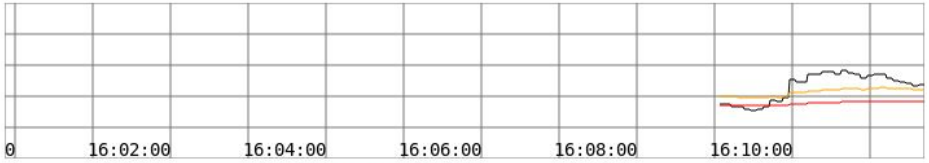
Create request successful



MONITORING

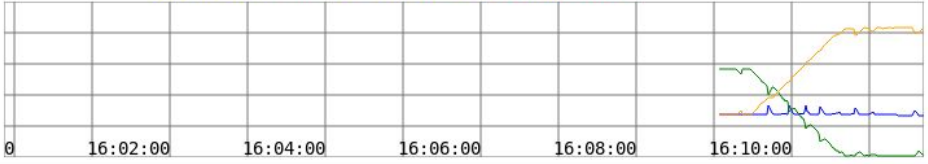
wmd, up 11 days 23:28:19, Linux: 4.9.0-040900-generic, 4 cores, running processes: 2, blocked processes: 0, ctxt 7461

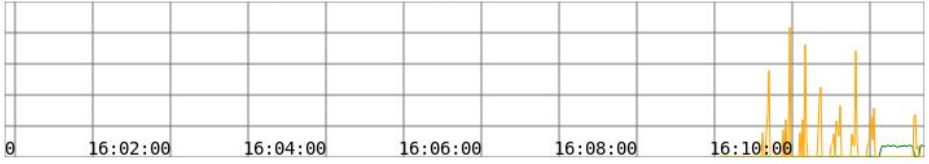




CPU: Idle 55.2% System 6.3% User 34.9% Total 44.8% IOWait 3.5%

Load average: 1 min 2.53 5 min 2.32 15 min 1.75





Mem: Total 19.3GB Used 4.7GB Free 239.2MB Buffers+Cache 14.4GB

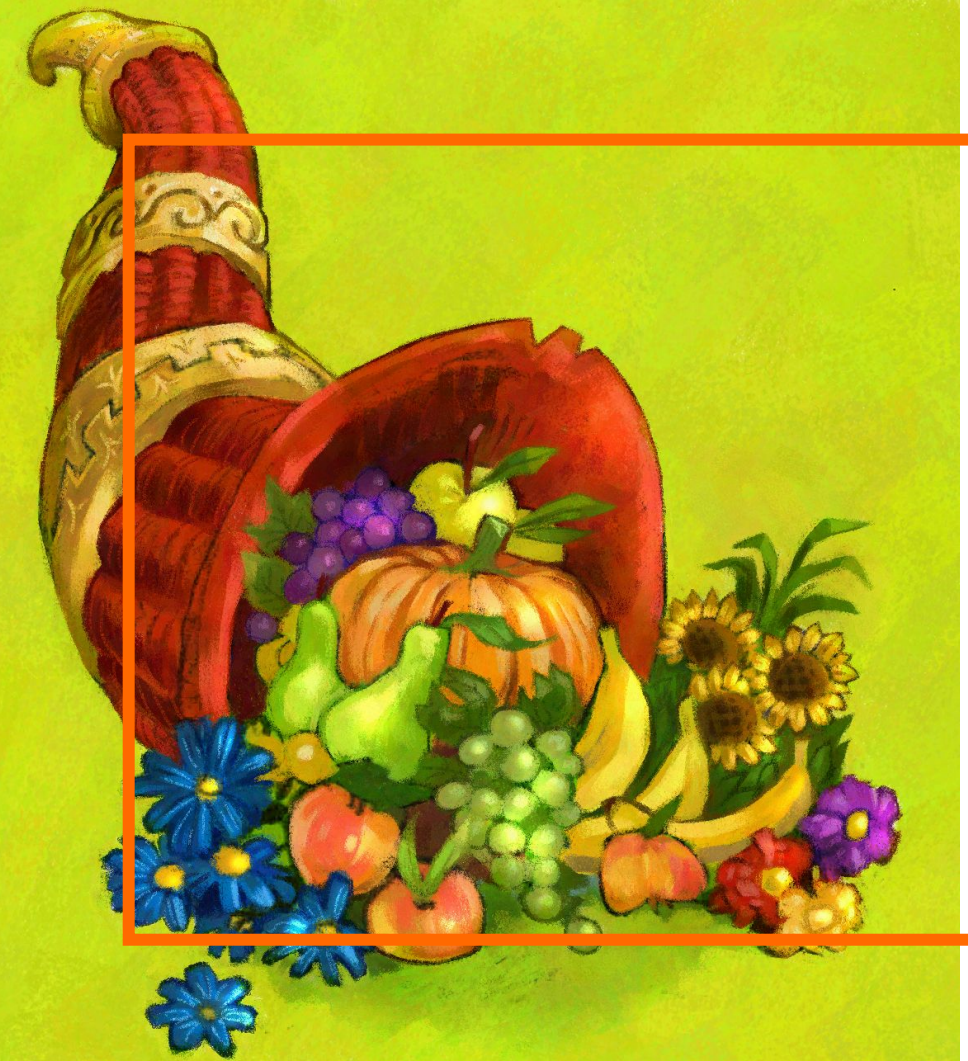
data/dm-1: Read: 98.5MB/s Write: 39.4MB/s Reads/s: 791 Writes/s: 44 Await: 3.93 %Util: 45.2

Partitions

Device	Read	Reads/s	Write	Write/s	Await	%Util	Total	Free	Size	Path
dm-1	98.5MB/s	791	39.4MB/s	44	3.93	45.2	214.7GB	118.8GB	14.0GB	/home/postgres/patroni/data/postgresql0
dm-1	98.5MB/s	791	39.4MB/s	44	3.93	45.2	214.7GB	118.8GB	960.2MB	/home/postgres/patroni/data/postgresql0/pg_xlog

master, 9.6.3, started 2017-05-14T10:33:47.000Z, connections 2/4 of maximum 100

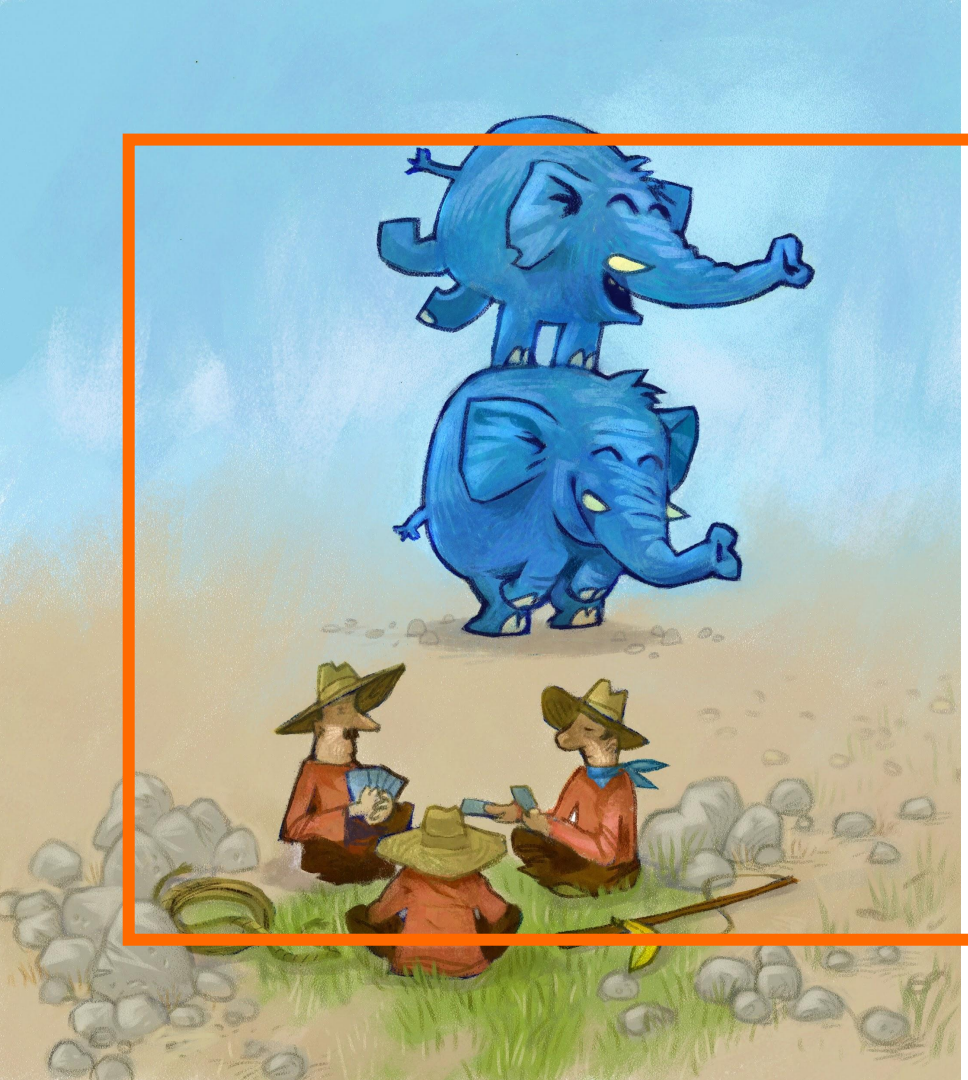
Pid	Lock	Type	State	utime	stime	Read	Write	uss	Age	Database	User	Query
18253		checkpointer	S	0.0	0.0	0.0KB/s	0.0KB/s	1.7MB				
18254		writer	S	0.0	0.0	0.0KB/s	0.0KB/s	1.7MB				
18255		stats collector	S	0.0	0.0	0.0KB/s	0.0KB/s	1.7MB				
18266		wal writer	S	0.0	0.0	0.0KB/s	0.0KB/s	1.7MB				
18267		autovacuum launcher	S	0.0	0.0	0.0KB/s	0.0KB/s	1.9MB				
20128		backend	R	81.0	3.0	98.6MB/s	0.0KB/s	7.7MB	00:02:13	postgres	postgres	select sum(n) FROM generate_series(1, 1000000000) n;
20826	20128	backend	S	0.0	0.0	0.0KB/s	0.0KB/s	2.3MB	00:03:22	postgres	postgres	select pg_advisory_lock(1);



OPEN SOURCE

LINKS

- Patroni: https://github.com/zalando/patroni
- Spilo: https://github.com/zalando/spilo
- Helm chart: https://github.com/kubernetes/charts/tree/master/incubator/patroni
- Postgres-operator: https://github.com/zalando-incubator/postgres-operator
- pam-oauth2: https://github.com/zalando-incubator/pam-oauth2
- bg_mon: https://github.com/CyberDem0n/bg_mon



QUESTIONS?