PostgreSQL on K8S at Zalando: Two years in production

FOSDEM 2020
PostgreSQL devroom
Brussels
ALEXANDER KUKUSHKIN
02-02-2020
ABOUT ME

Alexander Kukushkin

Database Engineer @ZalandoTech

The Patroni guy

alexander.kukushkin@zalando.de

Twitter: @cyberdemn
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
AGENDA

Brief introduction to Kubernetes
Spilo & Patroni
Postgres-Operator
Typical problems and horror stories
Kubernetes at Zalando

- > 140 Kubernetes clusters
  - 50/50 production/test
- Deployment to production only via CI/CD
- Access to production clusters is possible, but restricted
  - Requires the open incident ticket or approval by a colleague (4 eyes principle)
PostgreSQL on K8s at Zalando

> 1400
Terminology

**Traditional infrastructure**
- Physical server
- Virtual machine
- Individual application
- NAS/SAN
- Load balancer
- Application registry/hardware information
- Password files, certificates

**Kubernetes**
- Node
- Pod
- Container (typically Docker)
- Persistent Volumes
- Service/Endpoint
- Labels
- Secrets
Kubernetes overview

Master Node

- Controller
- Scheduler
- API server
- etcd
- add-ons...

Node

- kubelet
- docker
- kube-proxy
- fluentd

Pod

- Container

Pod

- Container
Stateful applications on Kubernetes

- **PersistentVolumes**
  - Abstracts details how storage is provisioned
  - Supports many different storage types via plugins:
    - EBS, AzureDisk, iSCSI, NFS, CEPH, Glusterfs and so on

- **StatefulSets**
  - Guarantied number of Pods with stable (and unique) identifiers
  - Ordered deployment and scaling
  - Connecting Pods with corresponding persistent storage (PersistentVolumeClaim)
Spilo Docker image

- All supported versions of PostgreSQL inside the single image
- Plenty of extensions (pg_partman, pg_cron, postgis, timescaledb, etc)
- Additional tools (pgq, pgbouncer, wal-e/wal-g)
- PGDATA on an external volume
- **Patroni** for HA
- Environment-variables based configuration
What is Patroni

- Automatic failover solution for PostgreSQL
- A python daemon that manages one PostgreSQL instance
- Uses Kubernetes objects (Endpoint or ConfigMap) for leader elections
  - Makes PostgreSQL 1st class citizen on Kubernetes!
- Helps to automate a lot of things like:
  - A new cluster deployment
  - Scaling out and in
  - PostgreSQL configuration management
Spilo & Patroni on K8S

Node1
- **Pod**: demo-0
  - role: master

Node2
- **Pod**: demo-1
  - role: replica

PersistentVolume
- S3
- PersistentVolume

StatefulSet: demo

Service: demo-repl
- labelSelector: role=replica

Secret: demo

Endpoint: demo

Service: demo-repl

zalando
Manual deployment to Kubernetes

- A few long YAML manifests to write
- Different parts of PostgreSQL configuration spread over multiple manifests
- No easy way to work with a cluster as a whole (update, delete)
- Manual generation of DB objects, i.e. users, and their passwords.
Kubernetes rolling upgrade

● Rotates all worker nodes in the K8s cluster

● Does it in a rolling matter, one-by-one

● If you are unlucky, it will cause the number of failover equal number of pods in your postgres cluster
Please write the title in all capital letters

Kubernetes rolling upgrade

Availability Zone 1

Node
- cluster: A
  - primary
- cluster: B
  - primary
- cluster: C
  - replica

Node (to-be-decommissioned)

Availability Zone 2

Node
- cluster: A
  - primary
- cluster: B
  - replica
- cluster: C
  - primary

Node (new)

Availability Zone 3

Node
- cluster: A
  - replica
- cluster: B
  - primary
- cluster: C
  - replica

Active Pod

Terminated Pod

Node
Kubernetes rolling upgrade

Availability Zone 1
- Node (to-be-decommissioned)
- cluster: A replica
- cluster: B replica
- cluster: C replica

Availability Zone 2
- Node (new)
- cluster: A replica
- cluster: B primary
- cluster: C replica

Availability Zone 3
- Node
- cluster: A replica
- cluster: B primary
- cluster: C replica

Active Pod
Terminated Pod
Kubernetes rolling upgrade

Availability Zone 1
- Node
  - cluster: A (replica)
  - cluster: B (replica)
  - cluster: C (replica)

Availability Zone 2
- Node
  - cluster: A (primary)
  - cluster: B (primary)
  - cluster: C (primary)

Availability Zone 3
- Node
  - cluster: A (primary)
  - cluster: B (primary)
  - cluster: C (primary)

Node (to-be-decommissioned)
Node (new)
Active Pod
Terminated Pod
Kubernetes rolling upgrade

Availability Zone 1
- Node
  - cluster: A replica
  - cluster: B replica
  - cluster: C replica

Availability Zone 2
- Node
  - cluster: A replica
  - cluster: B replica
  - cluster: C replica

Availability Zone 3
- Node
  - cluster: A primary
  - cluster: B primary
  - cluster: C primary

Node (to-be-decommissioned)
- Node (new)
Active Pod
Terminated Pod
Kubernetes rolling upgrade

Availability Zone 1
- Node
  - cluster: A primary
  - cluster: B replica
  - cluster: C primary

Availability Zone 2
- Node
  - cluster: A replica
    - cluster: B primary
    - cluster: C replica

Availability Zone 3
- Node
  - cluster: A primary
  - cluster: B primary
  - cluster: C primary

Node (to-be-decommissioned)
Node (new)
Active Pod
Terminated Pod
Kubernetes rolling upgrade

Availability Zone 1
- Node
  - cluster: A primary
  - cluster: B replica
  - cluster: C primary

Node (to-be-decommissioned)

Availability Zone 2
- Node
  - cluster: A replica
  - cluster: B primary
  - cluster: C replica

Node (new)

Active Pod

Availability Zone 3
- Node
  - cluster: A replica
  - cluster: B replica
  - cluster: C replica

Terminated Pod
# Kubernetes rolling upgrade

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Number of failovers</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
</tbody>
</table>
We need automation!
PostgreSQL cluster life-cycle

- create/update cluster config
- decommission
- provision/sync db user (periodically)
- deploy or do a rolling upgrade
- create/update

Please write the title in all capital letters. Use bullet points to summarize information rather than writing long paragraphs in the text box.
Goals

● Fully automated:
  ○ deployments
  ○ cluster upgrades
  ○ user management
  ○ minimize a number of failovers
Zalando Postgres-Operator

- Defines a custom Postgresql resource
- Watches instances of Postgresql, creates/updates/deletes corresponding Kubernetes objects
- Allows updating running-cluster resources (memory, cpu, volumes), postgres configuration
- Creates databases, users and automatically generates passwords
- Auto-repairs, smart rolling updates (switchover to replicas before updating the master)
apiVersion: "acid.zalan.do/v1"
kind: postgresql
metadata:
  name: acid-minimal-cluster
spec:
  teamId: "ACID"  # is used to provision human users
  volume:
    size: 1Gi
  numberOfInstances: 2
  users:
    zalando: # database owner
      - createdb
    foo_app_user: # role for application foo
  databases: # name->owner
    foo: zalando
postgreSQL:
  version: "11"
Stateful set

Spilo pod

PATRONI

Kubernetes cluster

deploy

cluster manifest

create

watch

operator pod

operator config map

create

Infrastructure roles

create

Cluster secrets

Client application

Endpint

Service

DB deployer

zalando
Rolling upgrade with Postgres-Operator

- Detect the to-be-decommissioned node by lack of the ready label and SchedulingDisabled status
- Move replicas to the already updated (new) node
- Trigger switchover to those replicas
Smart rolling upgrade (start)

Availability Zone 1
- Node
  - cluster: A primary
  - cluster: B primary
  - cluster: C replica

Availability Zone 2
- Node
  - cluster: A replica
  - cluster: B replica
  - cluster: C primary

Availability Zone 3
- Node
  - cluster: A replica
  - cluster: B replica
  - cluster: C replica

Node (to-be-decommissioned)
Node (new)
Active Pod
Terminated Pod
availability zone 1
node
cluster: a primary
class: b primary
class: c replica
node (to-be-decommissioned)
availability zone 2
node
cluster: a replica
class: b replica
class: c replica
availability zone 3
node
cluster: a replica
class: b replica
class: c replica
node (new)
active pod
terminated pod
Smart rolling upgrade (step 1)

Availability Zone 1
- Node (primary)
- Node (replica)
- Node (to-be-decommissioned)

Availability Zone 2
- Node (primary)
- Node (replica)

Availability Zone 3
- Node (replica)
availability zone 1
node
cluster: A
primary
cluster: B
primary
cluster: C
replica
node
availability zone 2
node
cluster: C
primary
cluster: A
replica
cluster: B
replica
node
availability zone 3
node
cluster: A
replica
cluster: B
replica
cluster: C
replica
node
node (to-be-decommissioned)
node (new)
terminated pod
active pod
Smart rolling upgrade (switchover)

Availability Zone 1
- Node
  - cluster: A replica
  - cluster: B replica
  - cluster: C primary

Availability Zone 2
- Node
  - cluster: C replica

Availability Zone 3
- Node
  - cluster: A replica
  - cluster: B replica
  - cluster: C primary

Node (to-be-decommissioned)
Node (new)
Active Pod
Terminated Pod
Smart rolling upgrade (finish)

Availability Zone 1

Node

cluster: A replica

cluster: B replica

cluster: C replica

Availability Zone 2

Node

cluster: A replica

cluster: B replica

cluster: C replica

cluster: C primary

Availability Zone 3

Node

cluster: A replica

cluster: B replica

cluster: C replica

cluster: B primary

cluster: C primary

Node (to-be-decommissioned)

Node (new)

Active Pod

Terminated Pod

Node (to-be-decommissioned)
Most common issues on K8s
Problems with AWS infrastructure

- AWS API Rate Limit Exceeded
  - Prevents or delays attaching/detaching persistent volumes (EBS) to/from Pods
    - Delays recovery of failed Pods
  - Might delay a deployment of a new cluster
- Sometimes EC2 instances fail and being shutdown by AWS
  - Shutdown might take ages
  - All EBS volumes remain attached until instance is shutted down
    - Pods can’t be rescheduled
Lack of Disk space

- Single volume for PGDATA, *pg_wal* and *logs*
- FATAL, 53100, could not write to file "pg_wal/xlogtemp.22993": **No space left on device**
  - Usually ends up with postgres being self shutdown
- Patroni tries to recover the primary which isn’t running
  - “start->promote->No space left->shutdown” loop

Disk space MUST be monitored!
Why not auto-extend volumes?

- Excessive logging
  - slow queries, human access, application errors, connections/disconnections

- pg_wal growth
  - archive_command is slow/failing
  - Unconsumed changes on the replication slot
    - Replica is not streaming? Replica is slow?
    - Logical replication slot?
  - checkpoints taking too long due to throttled IOPS

- PGDATA growth
  - Table and index bloat!
    - Useless updates of unchanged data?
    - Autovacuum tuning? Zheap?
  - Natural growth of data
    - Lack of retention policies?
    - Broken cleanup jobs?
ORM can cause wal-e to fail!

wal_e.main ERROR MSG: Attempted to archive a file that is too large. HINT: There is a file in the postgres database directory that is larger than 1610612736 bytes. If no such file exists, please report this as a bug. In particular, check pg_stat/pg_stat_statements.stat.tmp, which appears to be 2010822591 bytes.

Meanwhile in pg_stat_statements:

UPDATE foo SET bar = $1 WHERE id IN ($2, $3, $4, ..., $10500);
UPDATE foo SET bar = $1 WHERE id IN ($2, $3, $4, ..., $100500);
.... and so on
Exclusive backup issues

PANIC,XX000,"online backup was canceled, recovery cannot continue","xlog redo at D45/EB000028 for XLOG/CHECKPOINT_SHUTDOWN: redo D45/EB000028; tli 237; prev tli 237; fpw true; xid 0:105446371; oid 187558; multi 1; offset 0; oldest xid 544 in DB 1; oldest multi 1 in DB 1; oldest/newest commit timestamp xid: 0/0; oldest running xid 0; shutdown",""

- There is no way to join back such failed primary as a replica without rebuilding (reinitializing) it!
  - wal-g supports non-exclusive backups, but not yet stable enough
Out-Of-Memory Killer

$ postgres.log:
server process (PID 10810) was terminated by signal 9: Killed

$ dmesg -T:
[Wed Jul 31 01:35:35 2019] Memory cgroup out of memory: Kill process 14208 (postgres) score 606 or sacrifice child
Out-Of-Memory Killer

- Pids in the container (10810) and on the host are different (14208)
  - Hard to investigate!
- `oom_score_adj` trick doesn’t really make sense in the container
  - There is only Patroni+PostgreSQL running
- It is not really clear how memory accounting in the container works:
  - memory: usage 8388392kB, limit 8388608kB, failcnt 1
  - cache:2173896KB `rss:6019692kB` rss_huge:0KB shmem:2173428KB
    mapped_file:2173512KB dirty:132KB writeback:0KB swap:0KB
    inactive_anon:15732KB active_anon:8177696KB inactive_file:320KB active_file:184KB
    unevictable:0KB
Yet another OOM

```bash
$ kubectl get pods my-cluster-0
NAME           READY  STATUS     RESTARTS  AGE
my-cluster-0   1/1    Running 7  42d
```

```bash
$ kubectl describe pods my-cluster-0
...
Events:
Normal SandboxChanged 30m (x7 over 14d) kubelet, node1 Pod sandbox changed, it will be killed and re-created.
Normal Killing 30m (x4 over 12d) kubelet, node1 Stopping container postgres
```
Yet another OOM

$ dmesg

postgres invoked oom-killer: gfp_mask=0x14200ca(GFP_HIGHUSER_MOVABLE), nodemask=(null), order=0, oom_score_adj=-998

[ pid ]   uid tgid    total_vm  rss  pgtables_bytes swapents oom_score_adj name
[29203]   0  29203  256   1  32768  0  -998     pause
[29308]   0  29308  1096  190  49152  0  -998     dumb-init
[29419] 101  29419 154759  5592 442368 0  -998     patroni
[29420] 101  29420  27011  784  241664 0  -998     pgqd
[29474] 101  29474 162244  7861 417792 0  -998     postgres

Memory cgroup out of memory: Kill process 29203 (pause) score 0 or sacrifice child
Killed process 29203 (pause) total-vm:1024kB, anon-rss:4kB, file-rss:0kB, shmem-rss:0kB
How to mitigate Out-Of-Memory Killer?

- Reduce `shared_buffers` from 25% to 20%

- `vm.dirty_background_bytes = 67108864`
- `vm.dirty_bytes = 134217728`

Could be set only per Node :(
Kubernetes+Docker

● **ERROR:** could not resize shared memory segment "/PostgreSQL.1384046013" to 8388608 bytes: **No space left on device**
● PostgreSQL 11 (due to the “parallel hash join”)
● Docker limits `/dev/shm` to 64MB by default
● How to fix?
  ○ Mount custom dshm tmpfs volume to `/dev/shm`
    ■ Or set `enableShmVolume: true` in the cluster manifest
Problems with PostgreSQL

● Logical decoding on the replica? Failover slots?
  ○ Patroni does sort of a hack by not allowing connections until logical slot is created.
    ■ Consumer might still lose some events.
● “FATAL too many connections”
  ○ Prevents replica from starting streaming
    ■ Solved in PostgreSQL 12 (wal_senders not count as part of max_connections)
  ○ Built-in connection pooler?
Human errors

- Inadequate resource requests and limits
  - Pod can’t be scheduled due to the node weakness
  - Processes are terminated by oom-killer
- Deleted Postgres-Operator/Spilo ServiceAccount by employees
- YAML formatting :)

49
Please write the title in all capital letters.

Use bullet points to summarize information rather than writing long paragraphs in the text box.

https://www.reddit.com/r/ProgrammerHumor/comments/9fhvyl/writing_yaml/
Cluster YAML definition

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

metadata:
  name: "acid-minimal-cluster"
  namespace: "default"
  labels:
    team: acid

spec:
  teamId: "acid"
  postgresql:
    version: "11"
    numberOfInstances: 2
    volume:
      size: "100Gi"
    users:
      app_owner: []
    databases:
      prod_app_db: app_owner
  allowedSourceRanges:
    # IP ranges to access your cluster go here

resources:
  requests:
    cpu: 1000m
    memory: 1Gi
  limits:
    cpu: 1000m
    memory: 1Gi
```

New cluster configuration

- **Name**: minimal-cluster
- **Namespace**: default
- **Owning team**: acid
- **PostgreSQL version**: 11
- **DNS name**: acid-minimal-cluster.default
- **Number of Instances**: 2
- **Volume size**: 10 Gi
- **Users**: app_owner
- **Databases**: prod_app_db
- **CPU**:
  - Request: 1000
  - Limit: 1000
- **Memory**:
  - Request: 1 Gi
  - Limit: 1 Gi
Conclusion

● Postgres-Operator helps us to manage more than 1500 PostgreSQL clusters distributed in 80+ K8s accounts with minimal effort.
  ○ It wouldn’t be possible without high level of automation
● In the cloud and on K8s you have to be ready to deal with absolutely new problems and failure scenarios
  ○ Find the solution and implement a permanent fix
Open-source

- Postgres-operator: https://github.com/zalando/postgres-operator
- Patroni: https://github.com/zalando/patroni
- Spilo: https://github.com/zalando/spilo
Thank you!

Questions?