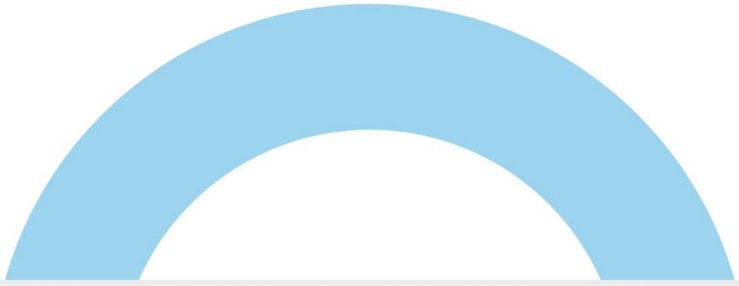
PostgreSQL on AWS RDS: tips and tricks.

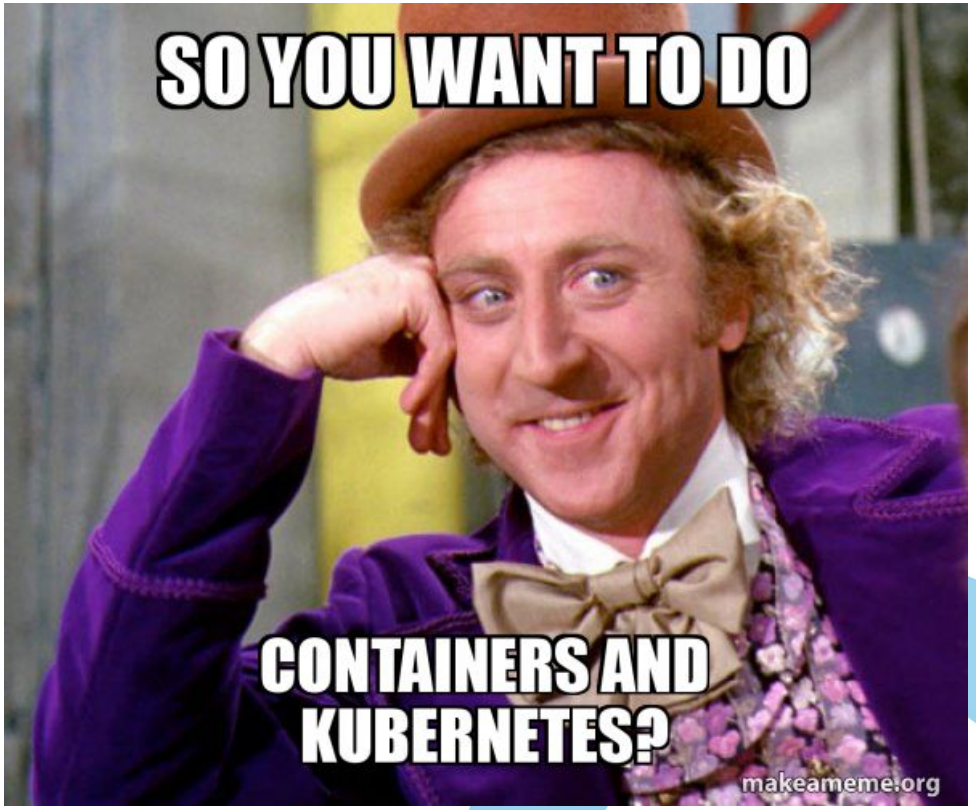
Dmitry Vasiliev, coins.ph

About the company

- Coins.ph - mobile wallet
- Using PostgreSQL since 2014 in AWS
- Dev/Ops: a Russian-speaking team

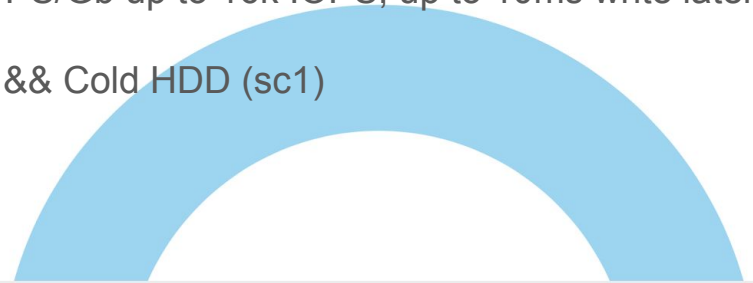


Modern cloud infrastructure



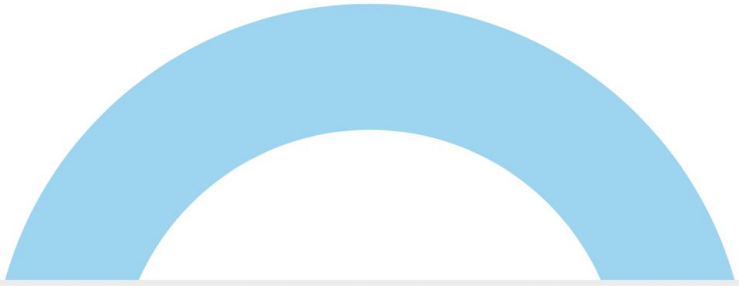
How you can use AWS

- EC2: classic virtual machines
  - Including I3 instances with fast local NVME ephemeral storage
- EBS: network block device
  - Provisioned IOPS SSD (io1): up to 64k IOPS, 1-2ms write latency
  - General Purpose SSD (gp2): 3 IOPS/Gb up to 16k IOPS, up to 10ms write latency
  - Throughput Optimized HDD (st1) && Cold HDD (sc1)
- RDS PostgreSQL



PostgreSQL in AWS

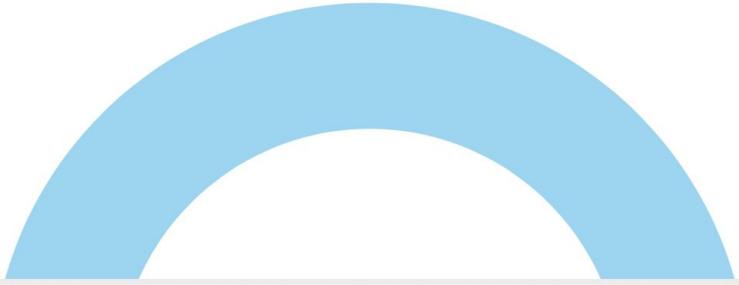
1. AWS EC2 (EBS storage or I3 instances)
2. PostgreSQL in kubernetes
3. PostgreSQL AWS RDS
4. Combination of all these methods



Some terms: RDS Multi-AZ

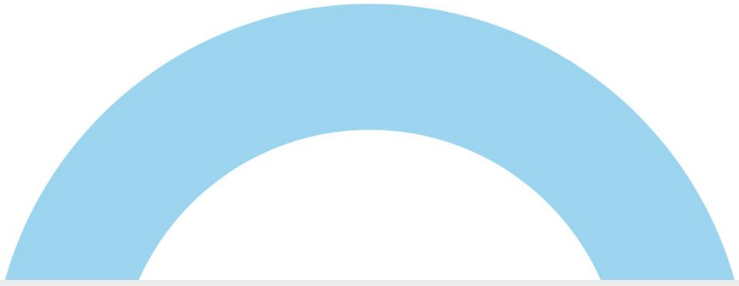
Multi-AZ: automatic failover

1. On the face of it can be thought of as block device replication
2. Hardware is reserved, but PostgreSQL service is not running
3. Failover is managed by DNS record (TTL=5s)
4. From our experience failover works: once a month per 10 instances



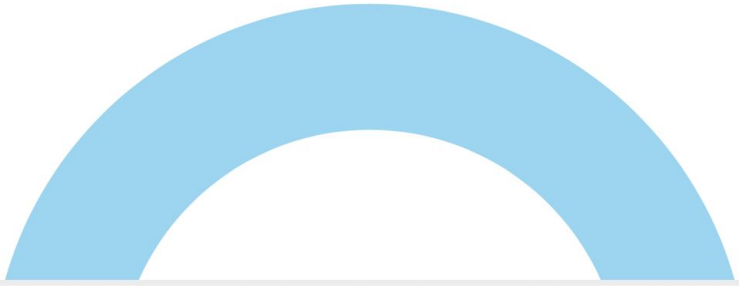
AWS EC2 (cons)

1. Hardware. Not cheaper than Multi-AZ on AWS RDS:
    a. Instances for: master, sync replica and async replica
    b. backup storage and backup test resources
2. Manpower. Harder to maintain:
    a. Failover
    b. Backup



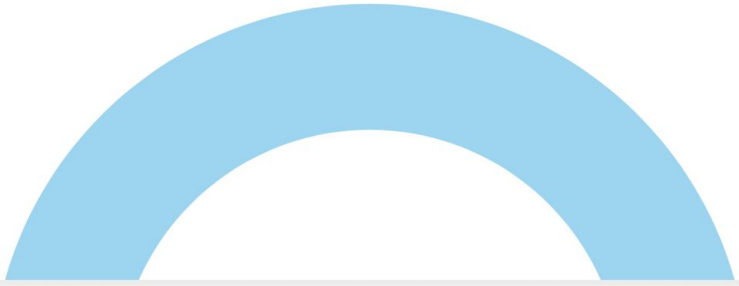
PostgreSQL in kubernetes (cons)

1. Large node instances in kubernetes are required
2. There are problems with “huge_pages = on” (didn’t check)
    a. On RDS we experienced problems on large instances with “huge_pages = off”
3. The complexity of the solution



RDS vs Self-Hosted (cons)

1. No superuser access
2. No streaming replication, except “Read Replica”
3. No access to the operating system (strace, gdb)
4. Single storage for all data: tablespaces, wal, server log.

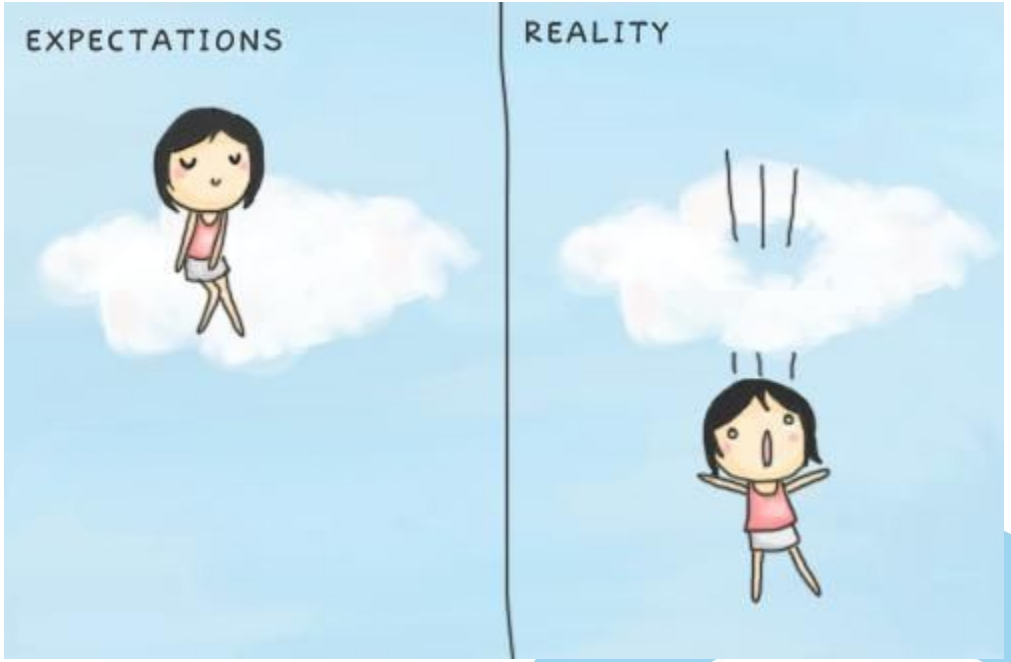


AWS RDS PostgreSQL (pros):

1. SLA 99.95% (*) : 22 minutes per month (higher in real life)
2. Backup: 30 days PITR, (but recovery speed is slow (**): ~ 10MB/s)
3. Failover: 1-5 minutes (depends on instance size)
4. No superuser: stable server versions and extensions

(*) through monthly bill, in fact, may be lower
(**) we are talking about the speed of applying WAL-archive

Let's talk about expectations vs reality



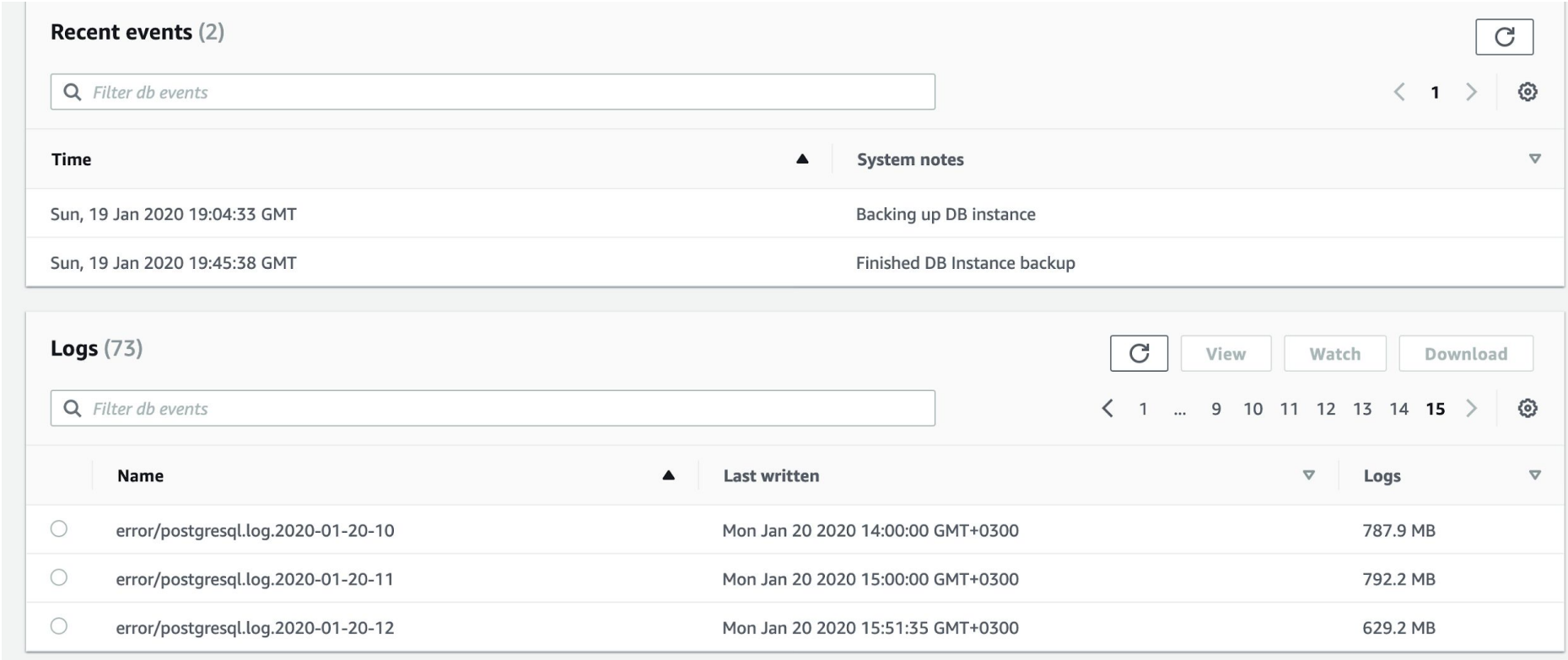
Tools

Tools: terraform

1. aws_db_instance - Encryption, Backup retention policy, ...
2. aws_db_parameter_group - PostgreSQL configuration, value is template (*)
3. data source aws_db_instance - Inventory
4. writing custom providers

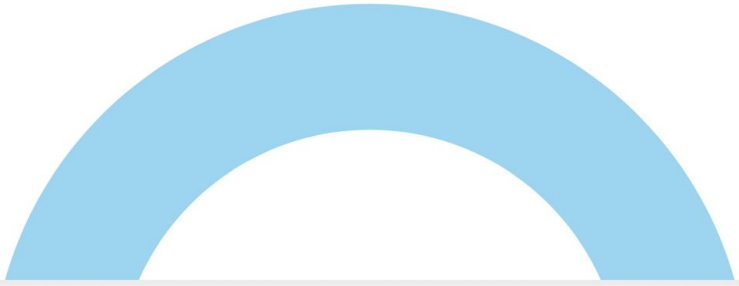
(*) https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithParamGroups.html

Log and Events



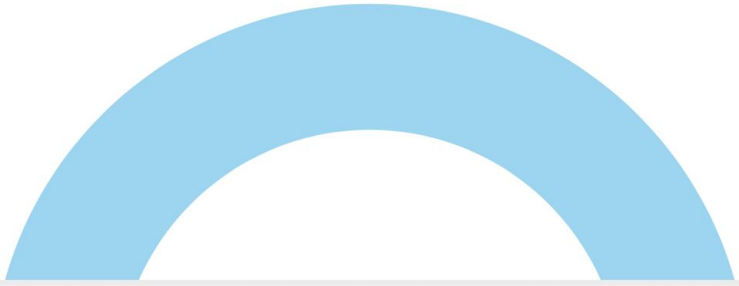
Log and Events

1. Web interface is terrible
2. I strongly recommend publishing logs to CloudWatch:
    a. you can download and get POSIX access to files
    b. you can use additional CloudWatch functionality



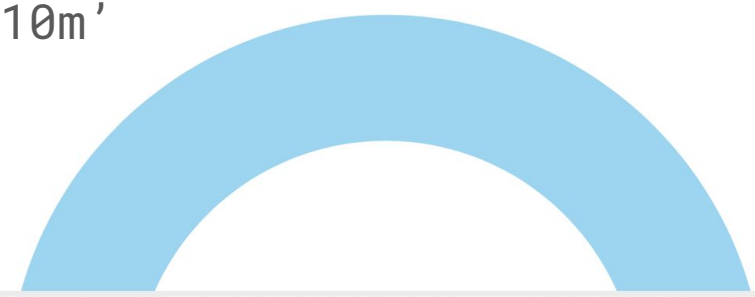
CloudWatch instruments

https://github.com/lucagrulla/cw



cw examples: time seek

```
cw tail -f /aws/rds/instance/<instance-name>/postgresql -b
'2020-01-20T00:00:00’
cw /aws/rds/instance/<instance-name>/postgresql -b ‘20m’ -e
‘10m’
```



cw examples: filter (aka grep), text search

Get ERROR|FATAL messages:

```
cw /aws/rds/instance/<instance-name>/postgresql \

-g '?ERROR ?FATAL'
```

https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/FilterAndPatternSyntax.html

cw examples: filter (aka grep), slow queries

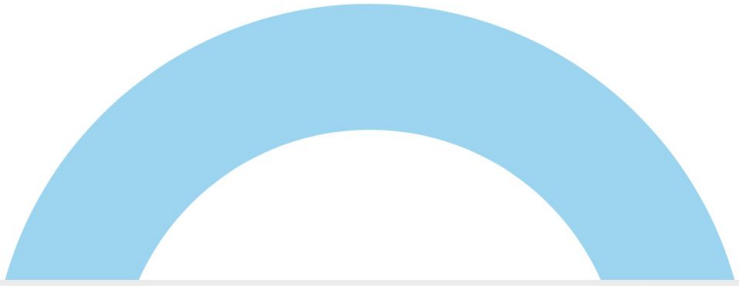
Get queries that took longer than 2 seconds:

```
cw /aws/rds/instance/<instance-name>/postgresql \
-g '[year, time, connection_info, x, duration > 2000, ...]'
```

https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/FilterAndPatternSyntax.html

Log Analysis: PgBadger and useful info

1. Download 5-minute segments from CloudWatch
2. Run PgBadger for incremental update
3. We also filter various useful information from this small segment: information about failed authentications and other important server messages
4. GoTo #1



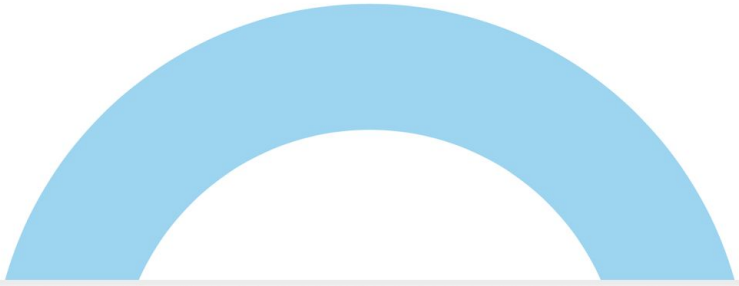
Basic system metrics

1. Basic system metrics: CPU, Disk, Network:
    a. IO latency, IOPS, Queue Depth
    b. CPU usage
2. Storage burst information (gp2)*: 3 iops/gb with burst (limited time) to 3k iops
3. CPU burst information (Tx-instances)

(*) https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html

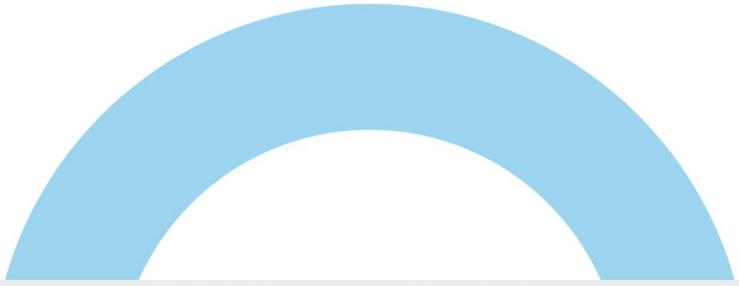
Enhanced system metrics

1. Disk: throughput (MB/s), latency, iostat utilization
2. CPU: la, process information, usage by type (system, user, wait, ...)
3. Memory: usage by type (buffer, dirty, hugepages, ...)



System metrics monitoring tool

https://github.com/percona/rds_exporter

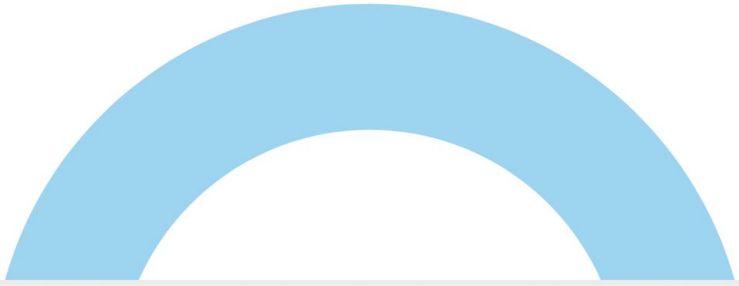


Metrics provided by PostgreSQL monitoring tool

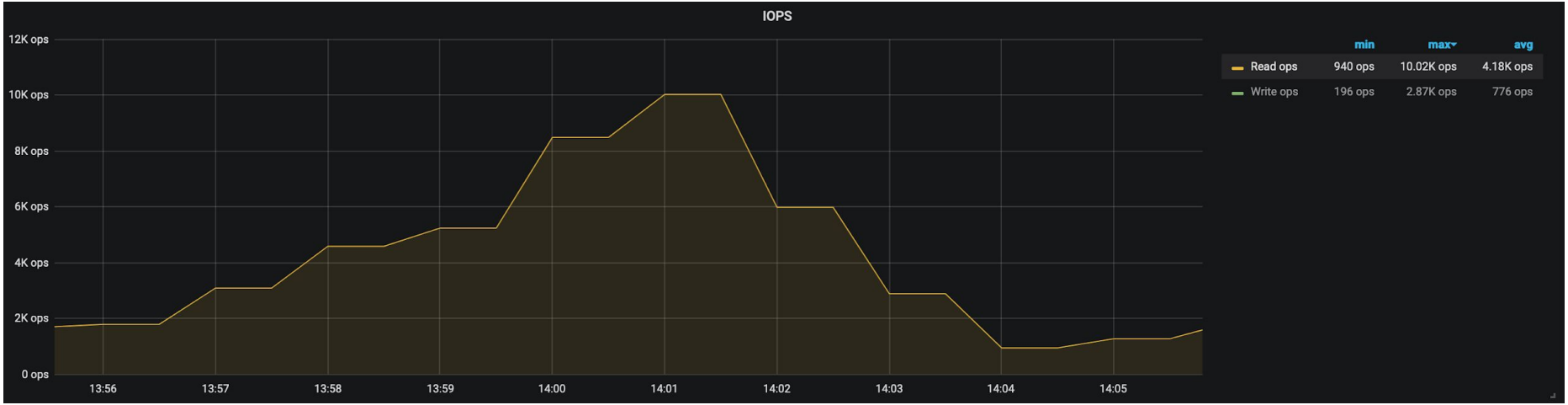
https://github.com/vadv/pg_gatherer

Metrics that helped me out more than once

1. System statistics snapshots:
    a. pg_stat_activity (long queries, waits)
    b. pg_locks
    c. pg_user_tables (seq scans, vacuum, relpages)
2. Buffer pool by relation



Pg_gatherer: iops + pg_stat_activity snapshot



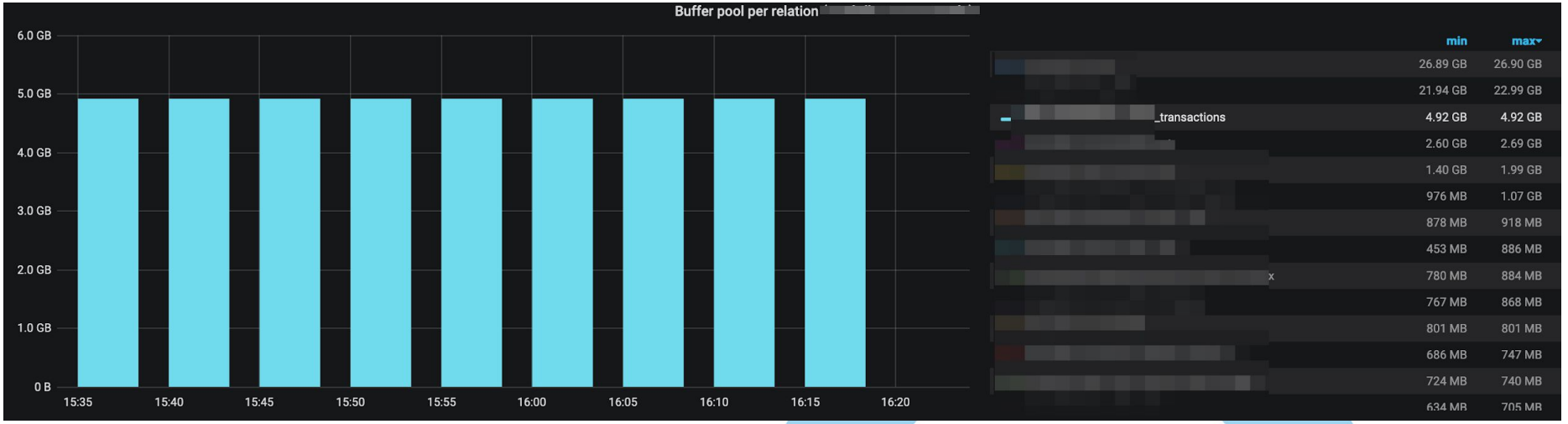
Long queries

datname	query_id	application_name	first seen	last seen	duration	query
core	a205054e-f51b-48e8-49f0-e3259151859d	psql	2020-01-21 16:56:02	2020-01-21 17:03:23	7.35 min	create index CONCURRENTLY [redacted];

Pg_gatherer: sequential scans

Sequential scans tables > 256Mb

timestamp	table	table size	count
2020-01-21 19:20:00	_transactions	4.92 GB	2
2020-01-21 19:15:00	_transactions	4.92 GB	2
2020-01-21 19:10:00	_transactions	4.92 GB	2



Sequential scans on “small tables”

```
-> Parallel Seq Scan on [redacted]_transactions  (cost=0.00..629556.76 rows=2907876 width=8) (actual time=0.003..1500.353 rows=2492326 loops=2)
     Output: status
     Buffers: shared hit=600478 dirtied=1
     Worker 0: actual time=0.003..1493.825 rows=2520207 loops=1
       Buffers: shared hit=303601 dirtied=1
```

The ring buffer is not used for “small” (shared_buffers/4) tables.

The target table is completely immersed into shared_buffers.

Queries are executed quickly, and they are not visible in the slow query log.

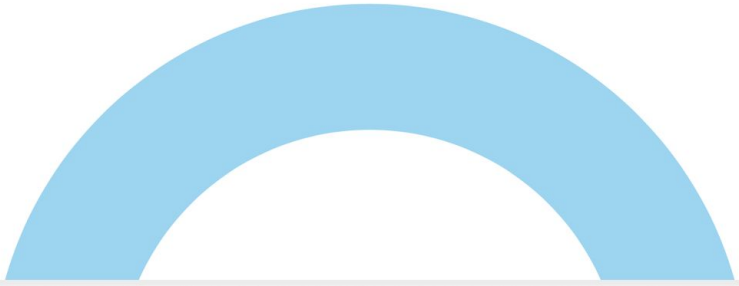
Perhaps it would be better if PostgreSQL used OS PageCache.

http://www.interdb.jp/pg/pgsql08.html

Balancer

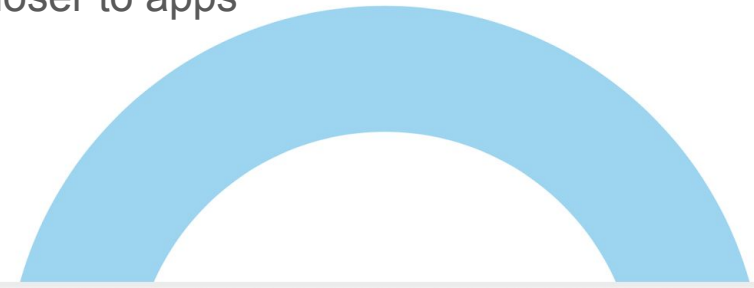
There are not many options:

1. PgBouncer (not without problems, but good old one)
2. Odyssey (too novel, no PAUSE functionality)



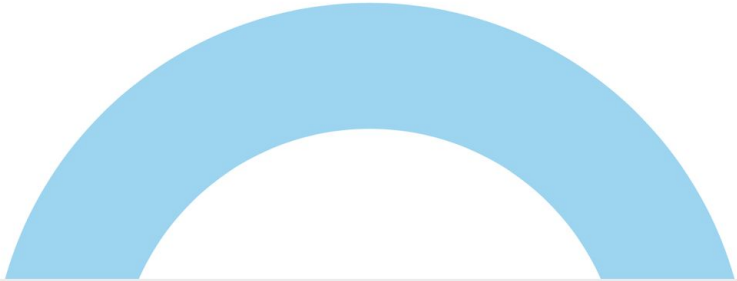
Balancer: PgBouncer

1. EC2 (pros)
    a. Stability (PODs are often re-scheduled to another k8s nodes)
2. Kubernetes (pros)
    a. Config is located closer to apps
    b. Scaling

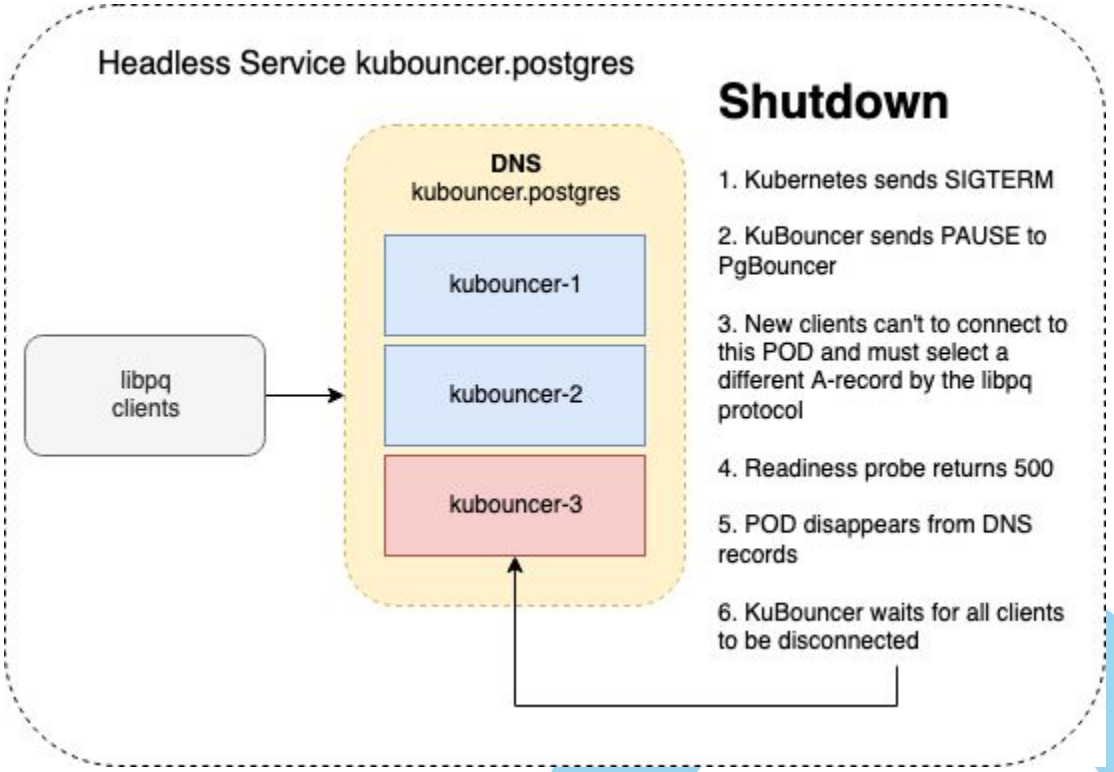


KuBouncer: PgBouncer + Kubernetes agent

1. Monitors Kubernetes Secret updates (updated via terraform or manually)
2. Monitors database availability
3. Exposes metrics to Prometheus
4. Performs “Graceful Shutdown”



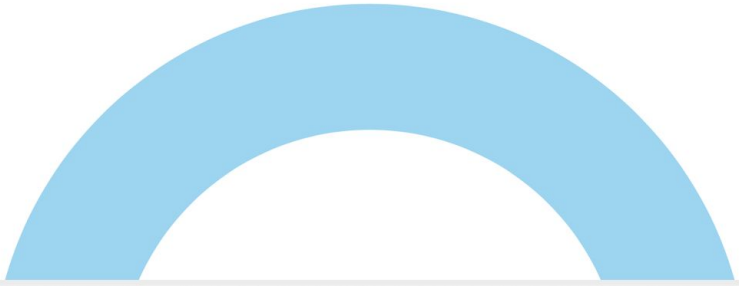
KuBouncer: Graceful shutdown



KuBouncer: automation

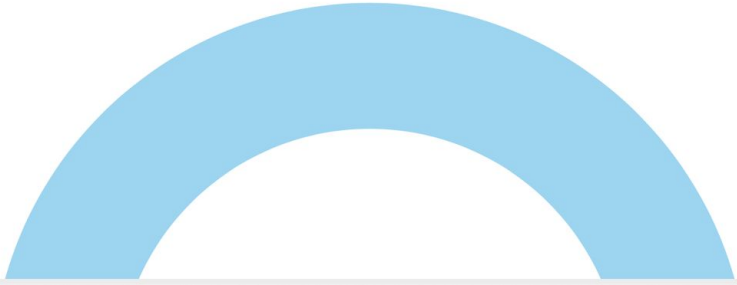
1. Config is easily created by terraform provider
2. Database per application: each application uses own virtual database:

   applicationname_databasename
3. Temporary virtual user is created for each virtual database

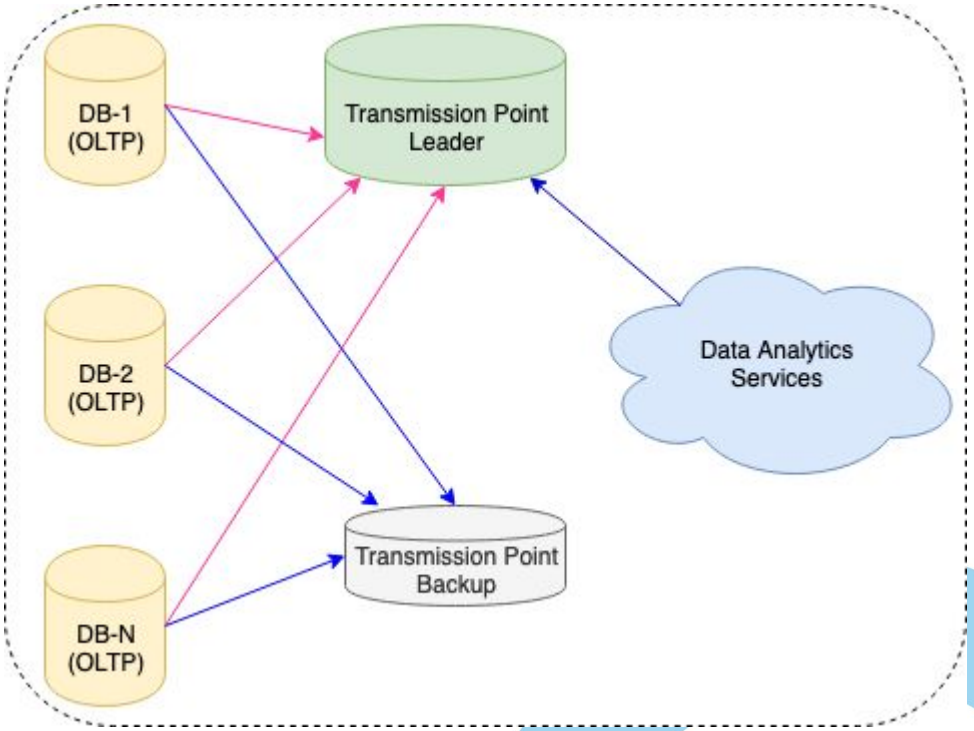


Data analytics

1. We use logical replication to collect data from a variety of instances in one instance (Transmission Point)
2. TP deployed in EC2 (two instance: leader && backup)
3. We give RO access to the TP instance for Data analytics team and this is no longer our headache (Debezium, Airflow, AWS Athena, ...)



Data analytics



Logical replication

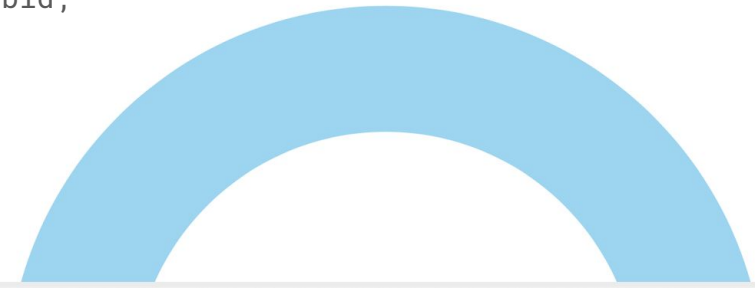
1. Initial copy of logical replication is an expensive operation for both: publisher and subscriber
2. We have developed a way to create a copy of subscriber using pg_basebackup (*)

(*) Based on: https://medium.com/avitotech/recovery-use-cases-for-logical-replication-in-postgresql-10-a1e6bab03072

Cheap way to create a copy of a logical subscriber

1. Backup: pg_basebackup -h Leader -U replica -W -D data -X stream -R
2. Backup: `select pg_create_logical_replication_slot('replica_slot_name', 'pgoutput')`
3. Leader: `alter subscription sub_name disable;`
4. Leader (make snapshot):

```
with subscriptions (select 'pg_'||(oid::bigint)::text as external_id, subname, subconninfo,
subdbid from pg_subscription)
select s.subname, s.subconninfo, status.remote_lsn, d.datname
from pg_replication_origin_status status
inner join subscriptions s on s.external_id = status.external_id
inner join pg_database d on d.oid = s.subdbid;
```

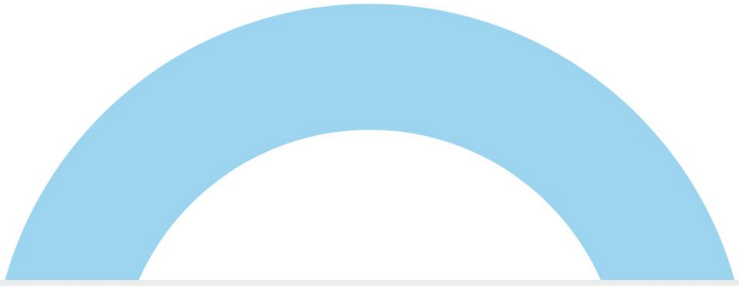


Cheap way to create a copy of a logical subscriber

5. Backup: `pg_ctl promote -D /var/lib/pgsql/11/data`
6. Backup: `create subscription … publication tp_pub with (enabled=false, copy_data=false, create_slot=false);`
7. Backup (from 4):

```
with subscriptions as (select 'pg_'||(oid::bigint)::text as external_id from pg_subscription
where subname = “replica_slot_name” )
select pg_replication_origin_advance(s.external_id, remote_lsn) from subscriptions s;
```

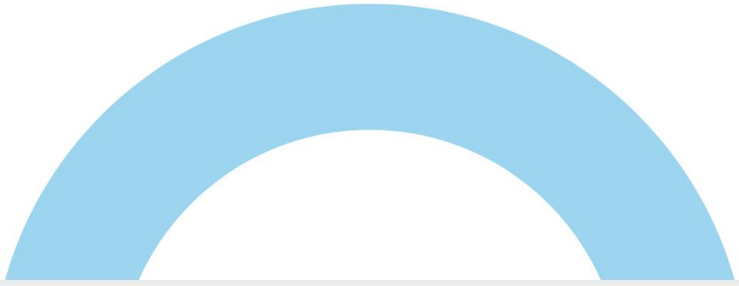
8. Backup && Leader: `alter subscription ... enable;`



Tricks: Row-Level locks

Row-level locks are extremely slow on RDS, possibly due to the increased write latency on Multi-AZ.

To reduce numbers of failed or hot row-level locks, we increased the total number of locks: we added an additional advisory lock before each row-level lock.

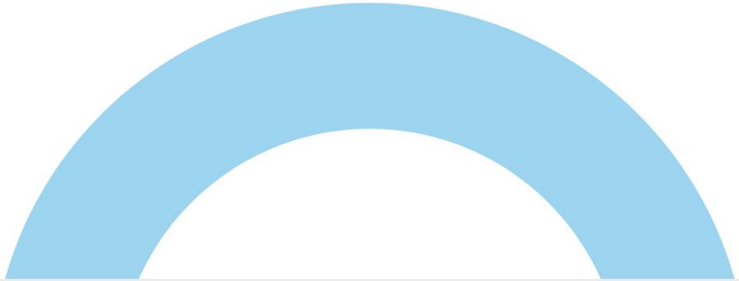


Tricks: Row-Level locks

```
select id from “order” for
update where id = “X” nowait;
```



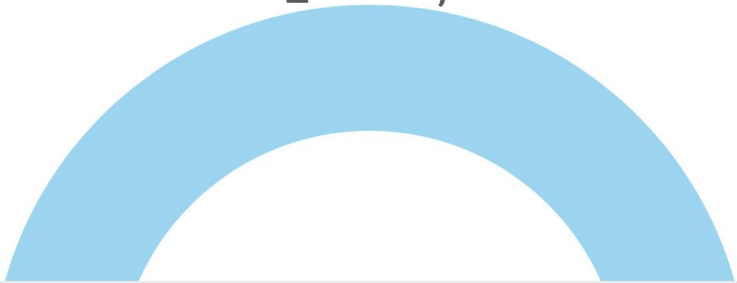
```
select
pg_try_advisory_xact_lock(“X”);

select id from “order” for
update where id = “X” nowait;
```



Tricks: Security

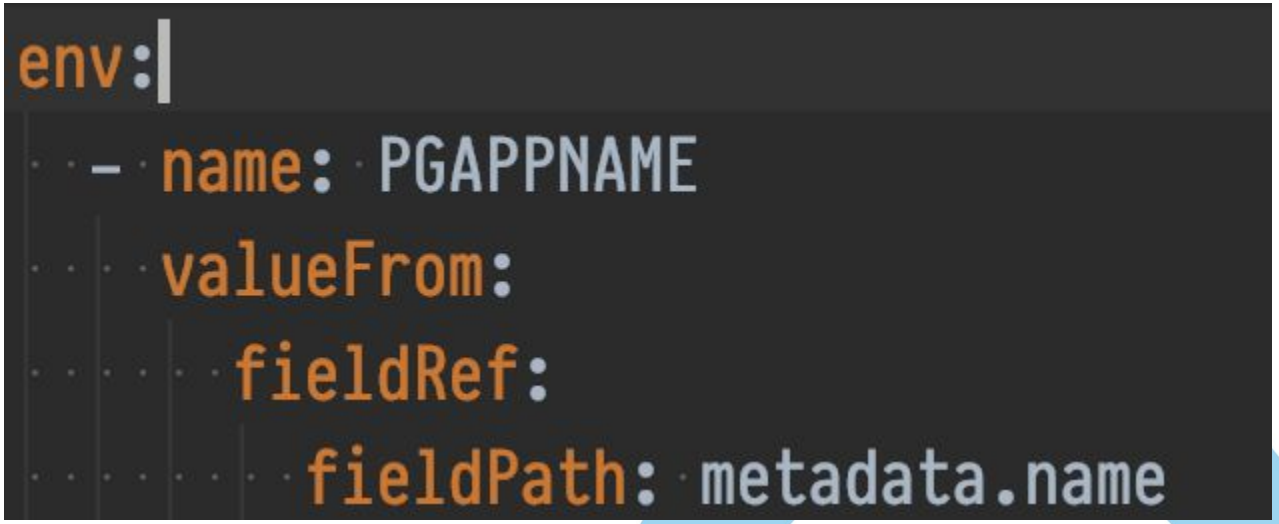
Custom terraform provider:

1. `alter database X owner owner_role;`
2. `create role rotated_role with login valid until ‘1 month’ in role owner_role;`
3. `alter rotated_role set role to owner_role;`



Tricks: Observability in Kubernetes

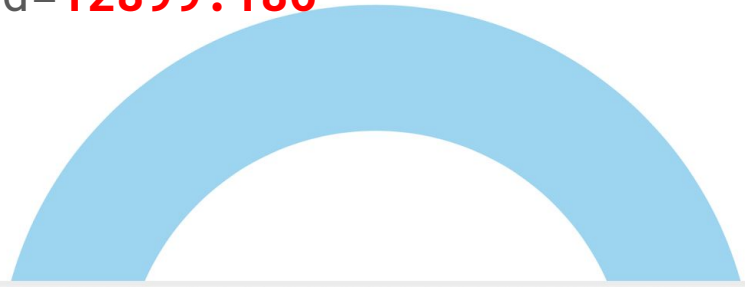
Not all applications are connected via PgBouncer, for these applications we set PGAPPNAME to POD Name:



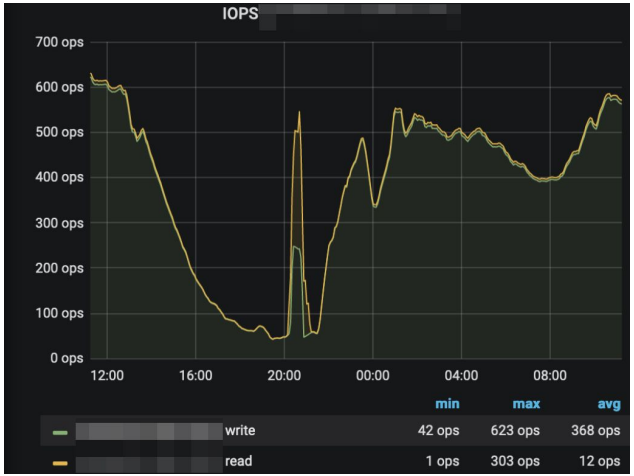
RDS Problems

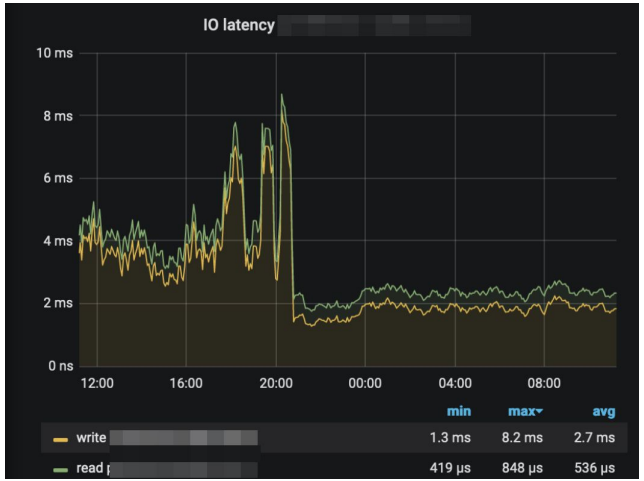
Rare IO problems:

```
Index Cond: ((x.user_id)::text = 'x'::text)
      Filter: ((x.y)::text ~~ 'z%'::text)
      Buffers: shared hit=3 read=1
      I/O Timings: read=12899.186
```



RDS Problems

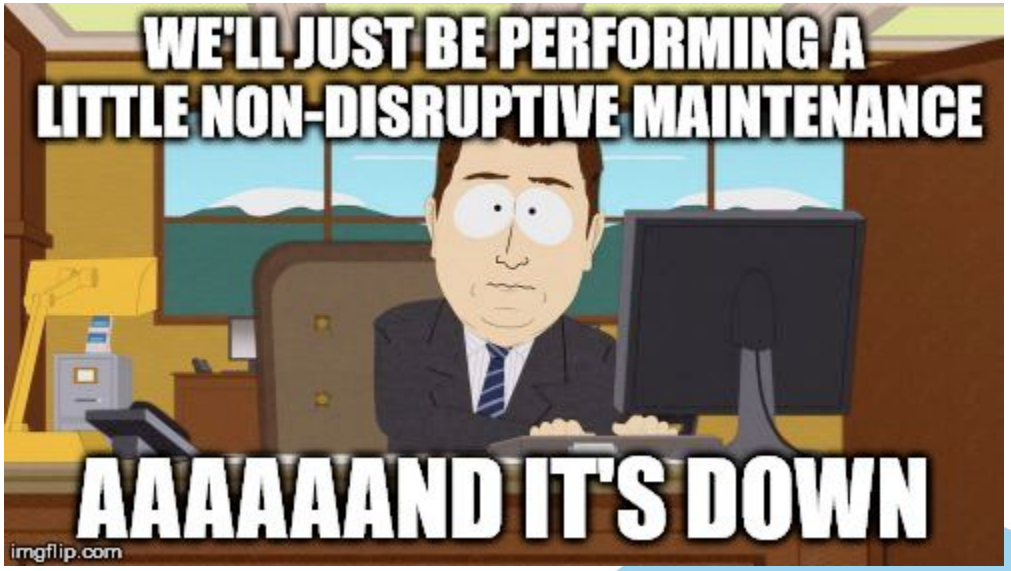




Sometimes you have to change the instance type to run away

from troublesome host

RDS Problems



https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ModifyPostgreSQLInstance.html

Thank you!

Questions?