Migrações de dados sem downtimes!

Truques e lições aprendidas com PostgreSQL na Olist

Jéssica Bonson, desenvolvedora no **Olist**

TDC Floripa 2019 - Trilha Big Data

Desenvolvedora na **Olist**

Graduação/Mestrado em Ciências da Computação

+-7 anos como pesquisadora / desenvolvedora

Foco em backend, big data e machine learning

Curto jogos, RPG e artes marciais

Olist

Maior loja nos principais marketplaces do Brasil.

+ 4700 lojistas

+ 500 mil produtos

+ 1 milhão de consumidores únicos

+ 10 marketplaces parceiros

Arquitetura de micro serviços e serverless.

Python. Go. PostgreSQL. AWS. Heroku.

- As migrações de dados são parte do processo de deploy.
  - Alterações nas tabelas ou nos dados

- Como executar as migrações?
  - Downtime X Runtime

Conceitos

DELETEs e UPDATEs no Postgres

- DELETEs não deletam dados, matam
- UPDATEs duplicam linhas
  - Depois da atualização, ambas as linhas estão no BD
- Por quê?
  - Rollback de transações
  - Diferentes visibilidades de dados
  - Permite escrita em paralelo com a leitura
- Resultado: Cemitério de linhas

VACUUM

- É quem deleta ‘de verdade’
  - Na verdade, não deleta, só marca para reuso
  - Alternativa: VACUUM FULL

- Nunca causa exclusive locks nas tabelas
- Só deleta dead rows quando não estão sendo mais usadas

Autovacuum

- PostgreSQL sabe se virar
  - Um daemon checa as tabelas de tempos em tempos
  - Se necessário, roda o VACUUM

- Porém... Transações lentas são um problema
  - VACUUM não pode deletar as linhas
  - Causa table bloat

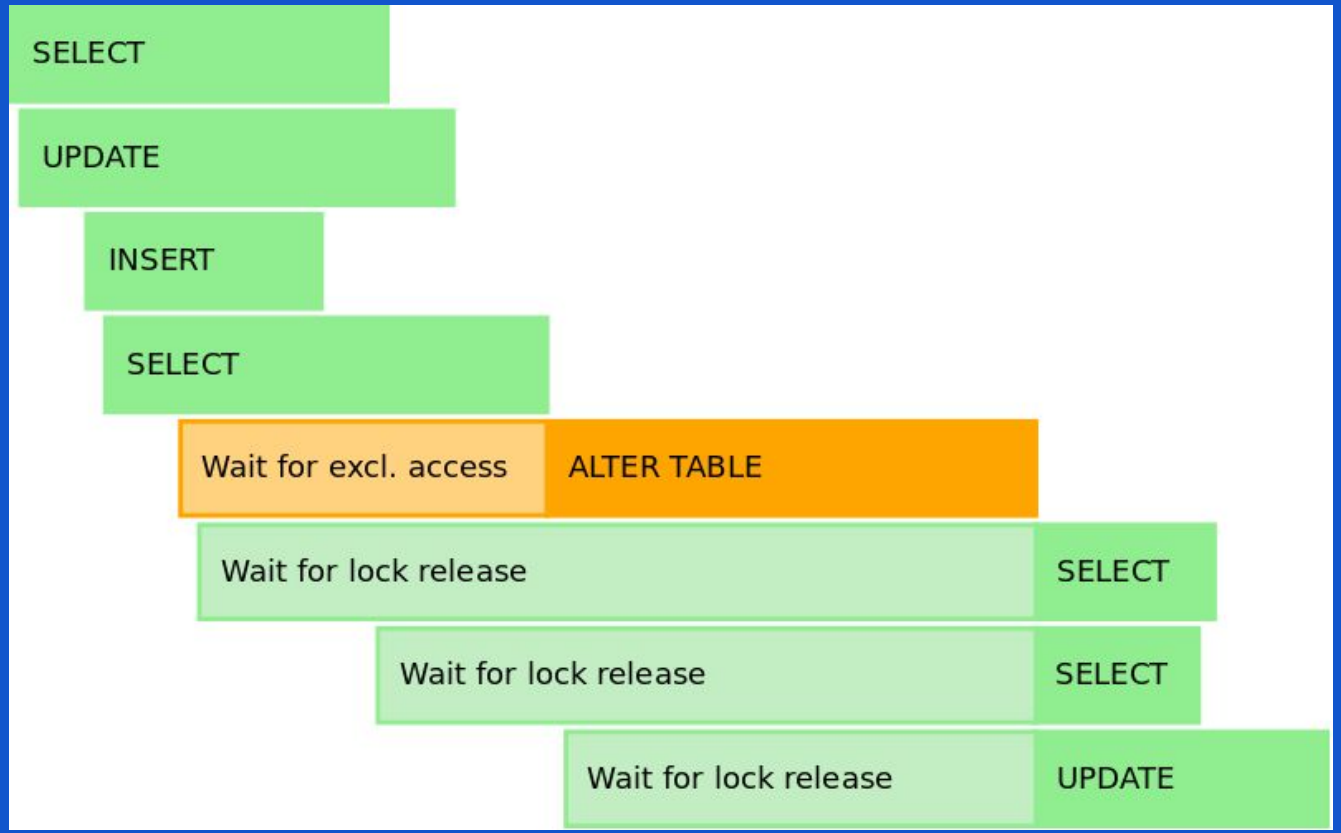
heroku pg:vacuum_stats DATABASE_URL --app <app_name>

table	last_vacuum	last_autovacuum	rowcount	dead_rowcount	autovacuum_threshold	expect_autovacuum
jobs		2013-05-20 16:54	82,617	36,056	16,573	yes
logs		2013-05-20 16:27	1	18	50	
reviews		2013-05-20 01:36	87	0	67	
users		2013-05-20 16:28	0	23	50	
...						

Table Bloat

- Live rows X Dead rows
- Se o VACUUM continuar sem conseguir rodar...
  - A tabela fica cada vez maior
  - VACUUM leva mais tempo
  - Performance da API piora

- Se for um problema recorrente, tunar o autovacuum


SELECT
UPDATE
INSERT
SELECT
Wait for excl. access
ALTER TABLE
Wait for lock release
SELECT
Wait for lock release
SELECT
Wait for lock release
UPDATE

Locks

Migrações para atualização de tabelas

Antes

```
UPDATE product
SET currency = 'BRL'
WHERE currency is Null;
```

Como resolver?

- Fazer a atualização registro a registro
- Dividir a migração em lotes

Depois

```
SELECT COUNT(*) FROM product WHERE currency is Null;
```

```
SELECT id
FROM product
WHERE currency is Null
LIMIT :limit;
```

```
UPDATE product
SET currency = 'BRL'
WHERE id = :id;
```

Migrações para alteração de tabelas

Antes

```
ALTER TABLE freight

ADD COLUMN enable_subsidy boolean NOT NULL DEFAULT FALSE;
```

Como resolver?

- Dividir a migração em partes:
  - Exemplo: Criar campo com default False
    - Criar campo novo nullable
    - Setar default da coluna para False
    - Setar valores do campo para False
    - Remover nullable

Depois

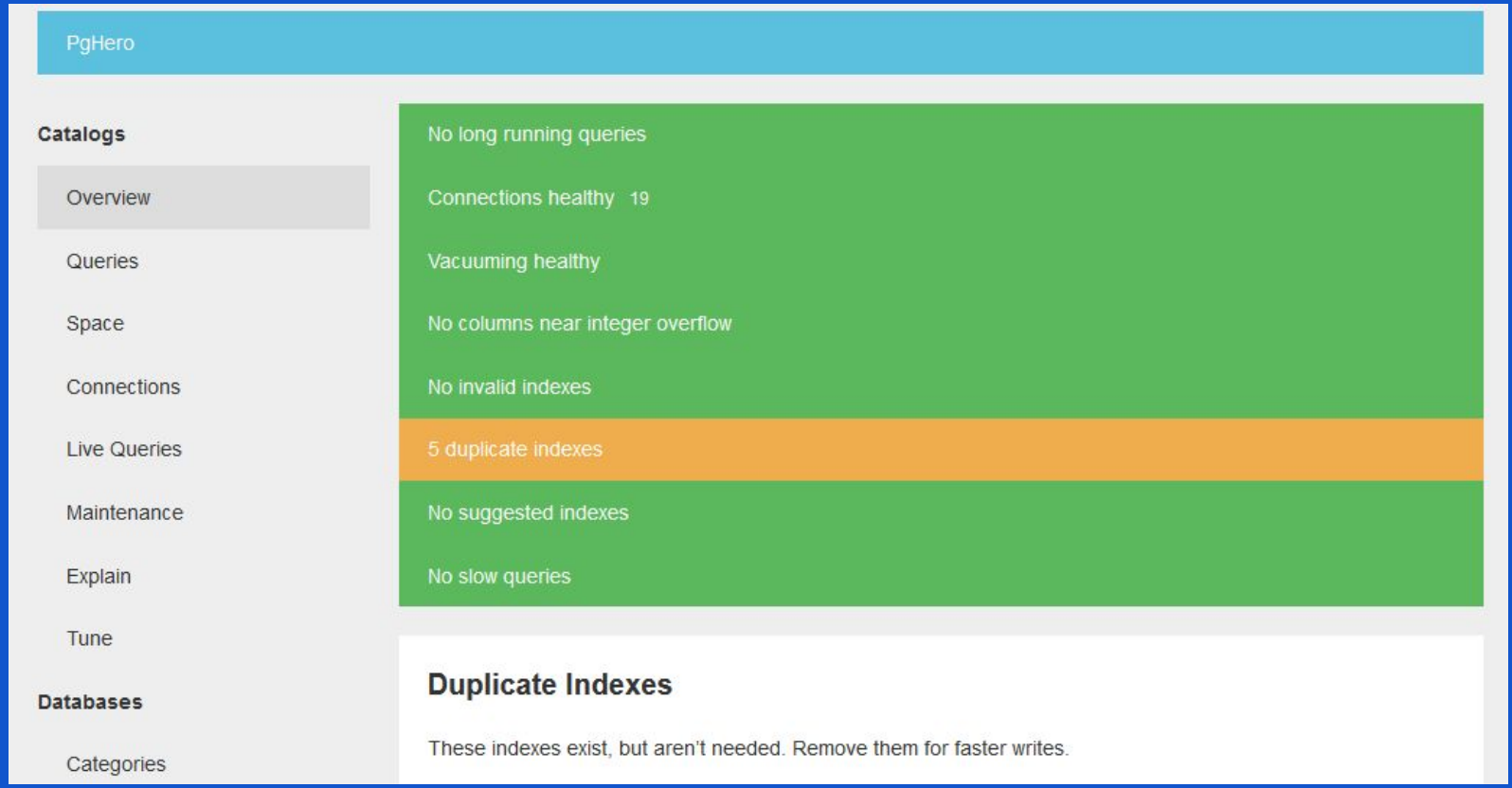
```
ALTER TABLE freight ADD COLUMN enable_subsidy boolean;

ALTER TABLE freight ALTER COLUMN enable_subsidy SET DEFAULT FALSE;

< Migração para alterar os valores do campo para FALSE >

ALTER TABLE freight ALTER COLUMN enable_subsidy SET NOT NULL;
```

Réplicas

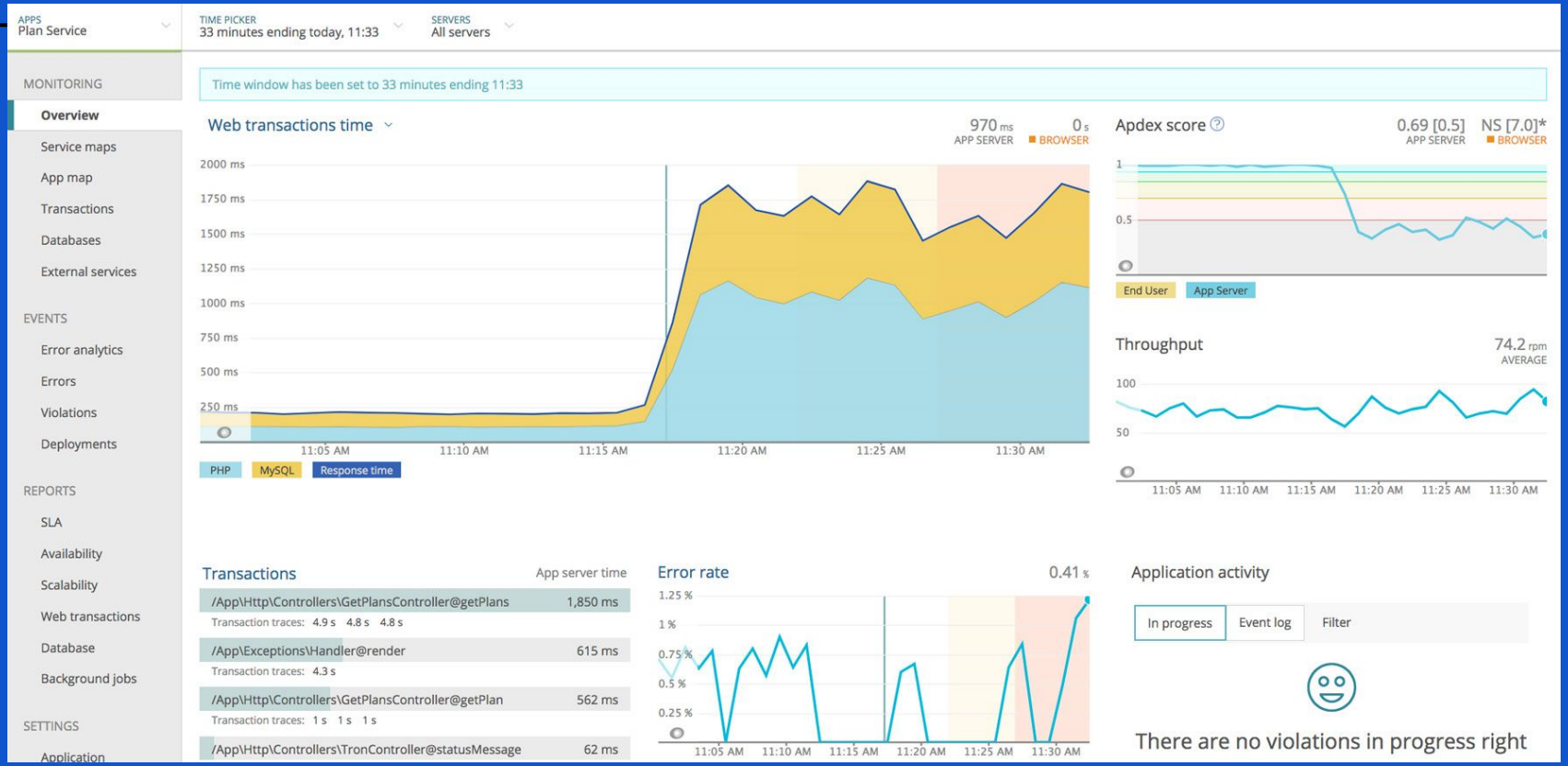
Trade-offs

- Custo x Benefício
  - Melhor performance de leitura
  - Alta disponibilidade e tolerância a falhas
  - Custo maior
  - Performance de escrita vs Consistência dos dados
- Caso da API de cálculo de frete
  - 1 Master Postgres na Amazon RDS com 2 Réplicas assíncronas
    - Read/Write na Master para sistemas internos
    - Read-only nas Réplicas
      - Sistemas externos
      - BI

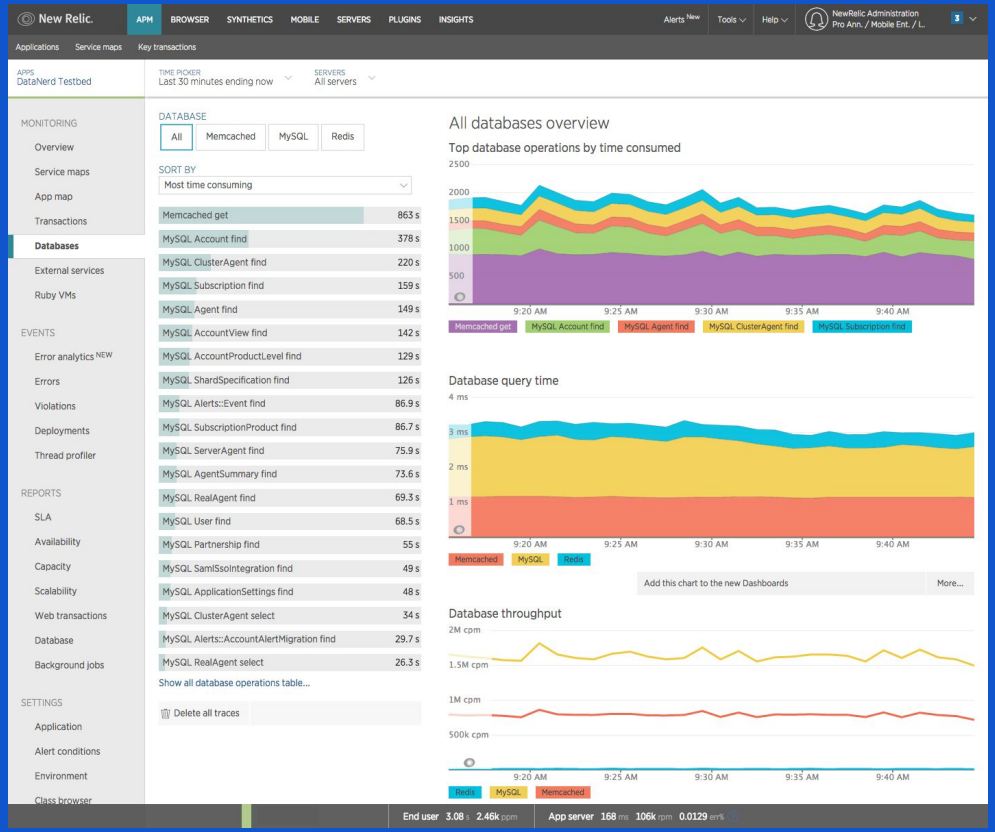
Monitoramento



PgHero: A performance dashboard for Postgres



NewRelic APM



NewRelic APM


opbeans-python
Integrations
Beta
context.tags.customer_tier : "$$$$"
Request
Errors
Response times
Avg. 6 ms
95th percentile
99th percentile
45 ms
23 ms
0 ms
03 AM
06 AM
09 AM
12 PM
03 PM
06 PM
09 PM
Sat 04
Requests per minute
HTTP 2xx 18.2 rpm
HTTP 4xx 1.6 rpm
HTTP 5xx 1.3 rpm
22 rpm
11 rpm
0 rpm
Request
Name
Avg. resp. time
95th percentile
Req. per minute
Impact
GET opbeans.views.oopsie
17 ms
22 ms
1.3 rpm
GET opbeans.views.product_customers
9 ms
12 ms
2.0 rpm
GET opbeans.views.customers
11 ms
14 ms
1.4 rpm
GET opbeans.views.orders
10 ms
14 ms
1.3 rpm

Elastic APM

Dicas e Precauções

- Testar tempo da migração localmente
  - time python manage.py migrate
- Verificar o SQL que será executado
  - python manage.py sqlmigrate <app> <número da migração>
- É melhor rodar migrações longas fora do horário de pico
- Criar backup do banco de dados
- Avisar os outros developers da migração

- Antes da migração, lidar com queries longas em execução
- Dividir migrações grandes em pequenas
- CREATE INDEX CONCURRENTLY vs CREATE INDEX
- Tabela pg_catalog.pg_locks
- UPDATE vs INSERT
- Estrutura das tabelas de histórico

WE'RE HIRING

desenvol-vedor(a) python

olist

Valeu!

Perguntas?

@jpbonson

Referências

- https://www.citusdata.com/blog/2018/02/15/when-postgresql-blocks/
- http://pankrat.github.io/2015/django-migrations-without-downtimes/
- https://momjian.us/main/writings/pgsql/nulls.pdf
- https://www.citusdata.com/blog/2018/02/22/seven-tips-for-dealing-with-postgres-locks/
- https://www.cybertec-postgresql.com/en/a-beginners-guide-to-postgresqls-update-and-autovacuum/
- https://devcenter.heroku.com/articles/managing-vacuum-on-heroku-postgres
- https://www.datadoghq.com/blog/postgresql-vacuum-monitoring/
- https://www.cybertec-postgresql.com/en/reasons-why-vacuum-wont-remove-dead-rows/
- https://www.percona.com/blog/2018/08/06/basic-understanding-bloat-vacuum-postgresql-mvcc/
- https://blog.timescale.com/scalable-postgresql-high-availability-read-scalability-streaming-replication-fb95023e2af/