Advanced database testing in CI/CD pipelines

Nikolay Samokhvalov

nik@postgres.ai

Postgres.ai

Nikolay Samokhvalov

PostgreSQL user since 2005

Occasional code contributions (XML, etc.)

Co-founded 3 startups (social media; 2 successful exits)

Helped with Postgres to: GitLab, Chewy, Miro, etc.

Postgres.ai founder

SUBSCRIBE

YouTube **Postgres.tv**

Twitter **@samokhvalov**

@Database_Lab

Postgres.ai

Help companies with PostgreSQL scalability and performance

Database Lab Engine
– thin clones for Postgres

Database Lab

An abstract example:
- DB size: 10 TiB
- DLE – a single VM with 10 TiB of disk space
- A single DLE provides 30-50 thin clones, each is 10 TiB
  - Engineers work independently
  - CI/CD pipelines run automated tests
- To get a new clone:
  - *~10 seconds and $0 (!)*

Used by: GitLab, QIWI, Allianz, BMW, Burger King, MERCURY

Database testing goals

1. Verify that all components of the system work as expected
   Schema and app comply, failover works, etc.
2. Improve quality of the software
   DB schema follows best practices, data integrity is enforced, etc.
3. Achieve good performance
   Queries execution is good, doesn’t degrade; no degradation after upgrades, etc.
4. Help with change management (avoid issues when making changes)
   DB migration deployment will succeed;
   app performance won’t degrade *during* and *after* it

Database testing goals

1. Verify that all components of the system work as expected
   Schema and app comply, failover works, etc.
2. Improve quality of the software
   DB schema follows best practices, data integrity is enforced, etc.
3. Achieve good **performance** – not only benchmarks!
   Queries execution is good, doesn't degrade; no degradation after upgrades, etc.
4. Help with **change management** – not only DB migration test on a small DB!
   DB migration deployment will succeed;
   app performance won't degrade *during* and *after* it

Database Testing Landscape

	Schema	Data
Static analysis	Test schema	Test data
Application change management	Test DDL	Test DML
Dynamic analysis	Single-user DB experiments	*Optimization (EXPLAIN), Application performance regression tests*
	Multi-user DB experiments	*Load and stress testing, regression checks for upgrades*
Infrastructure	Microbenchmarks, Test backups, Corruption check, HA tests, ...	

"app" ↑

"infra" ↓

For database testing in CI/CD pipelines, we want…

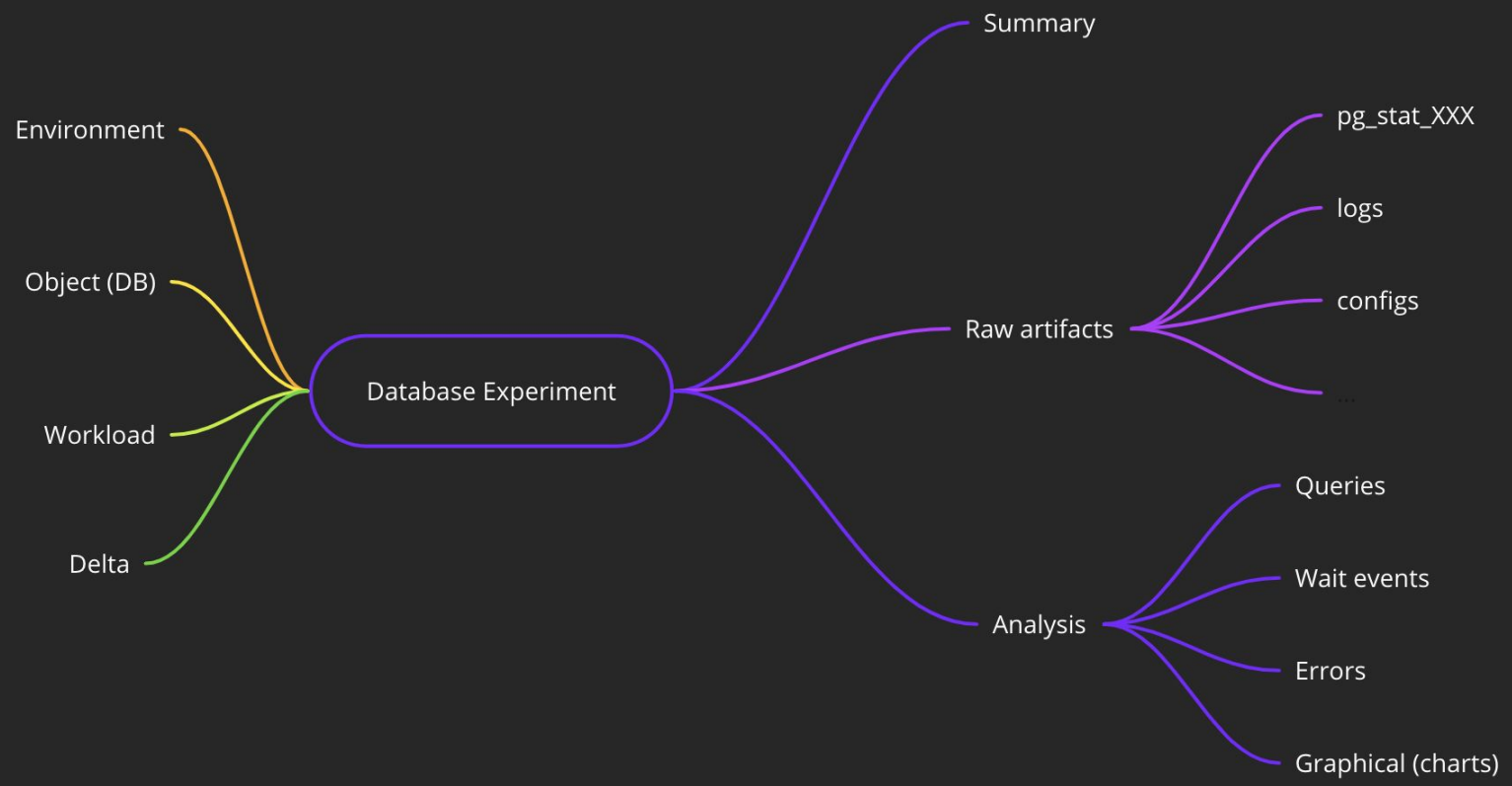
1. Fast (ideally immediate) database cloning
2. Each clone doesn’t cost us a lot of $$$
3. Predictable initial database state, fast reset, PITR
4. Reliable metrics to support decisions
   (*timing* may be not the best metric)

Single- & multi-user DB experiments – the difference

	Workload	CPU cores utilized	Focus	Shared env ok?	Use cases
Single-user experiments	1 query or a sequence – a single DB session	1 or a few	Execution plan: structure, rows, BUFFERS *(not timing!)* `EXPLAIN (ANALYZE, BUFFERS)`	Yes (thin clones work best – Database Lab)	SQL optimization, App changes (schema, data)
Multi-user experiments	Replayed / simulated – many DB sessions	All (or most)	Latency & throughput Utilization, saturation	No (only exclusive use)	Capacity planning Infrastructure changes (hw and sw upgrades, topology change, config tuning, etc.)

Multi-user DB experiments

Multi-user Database Experiment



The trickiest part of multi-user experiments – **Workload**

1. Use existing app-level benchmarking tooling (if any)
   - Works well for smaller envs, not for GitLab.com scale
2. pgreplay, pgreplay-go
   - Requires log collection => “observer effect” risks
   - Can help: log_transaction_sample_rate (PG12+) log_statement_sample_rate (PG13+)
3. Mirror traffic
   - Requires additional tooling that is invasive to production infrastructure
4. “Crafted” workload
   - Determine the most “influential” query groups (time, calls)
   - Define parameters (tricky)
   - Use some tool to generate workload
     i. pgbench: multiple scripts, multiple -f options with @XX, e.g.:

        pgbench -j32 -c32 -R10000 -f file1.sql@12 -f file1.sql@7 …

     ii. or: JMeter, sysbench, etc

“Crafted” workload – where to find parameters?

- Examples from Postgres logs
  - slow queries only – above `log_min_duration_statement`
  - Sampling: `log_transaction_sample_rate` (PG12+), `log_statement_sample_rate` (PG13+)

- Sample `pg_stat_activity`
  - Does not work for long queries (`track_activity_query_size = 1024` by default)

- Get popular values from pg_statistic
  - Somewhat random, may be “far from real life”
  - But can cover *all important query groups*

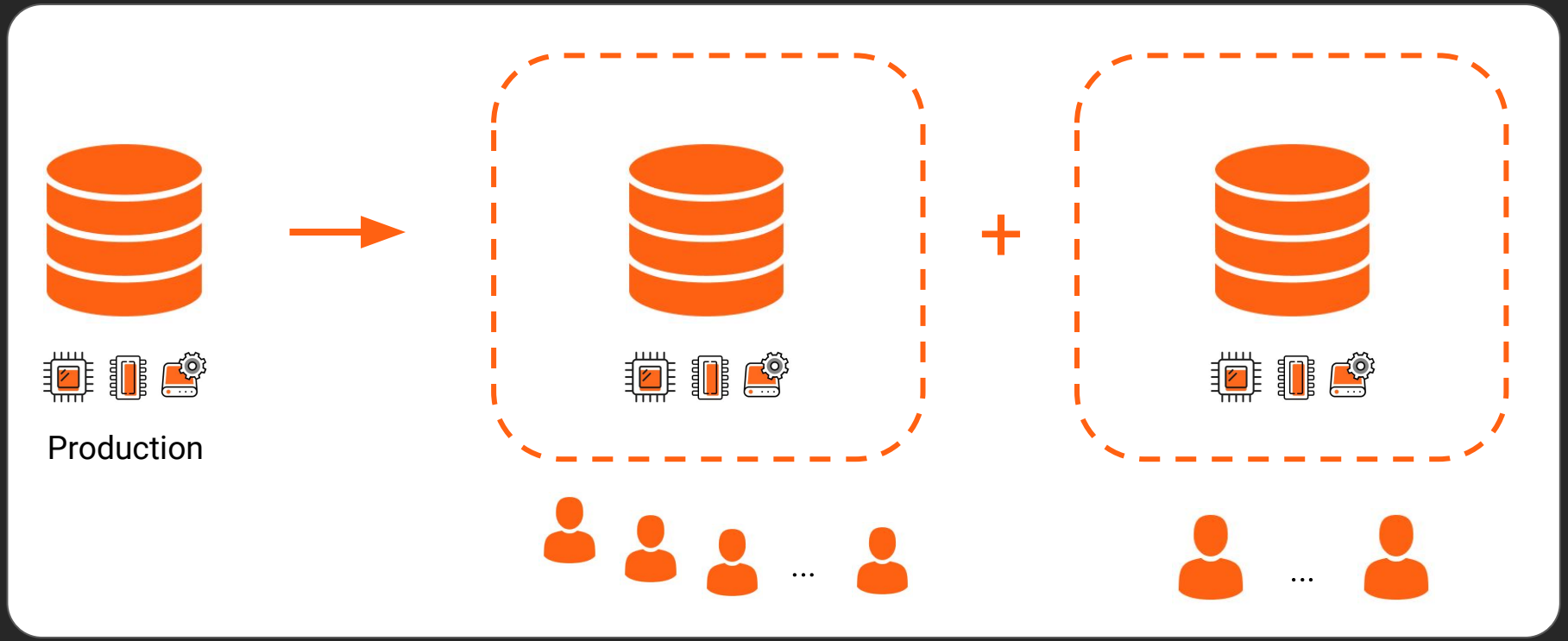
Tools useful for database experiments

	Provisioning and experimentation tools	Metric collection	Visualization and analysis
Single-user experiments	Database Lab Engine (DLE) for thin clones, Joe bot by Postgres.ai Source: Physical backups (e.g., WAL-G) or dumps EXPLAIN (ANALYZE, BUFFERS)	EXPLAIN (ANALYZE, BUFFERS)	Joe bot by Postgres.ai Depesz EXPLAIN PEV2 (Dalibo) pgMustard
Multi-user experiments	Terraform, Ansible, postgresql_cluster Cloud disk snapshots / Physical backups (e.g., WAL-G) pgbench, sysbench, JMeter, etc. + system microbenchmark tools: fio, sysbench, etc.	pg_stat_statements pg_wait_sampling pg_stat_kcache logerrors ad hoc tools: PASH Viewer, pgCenter PG logs, auto_explain /proc/**, perf, etc.	Netdata Pgwatch2 by Cybertec (Postgres.ai edition) pgBadger (HTML & JSON) FlameGraphs

Single-user DB experiments

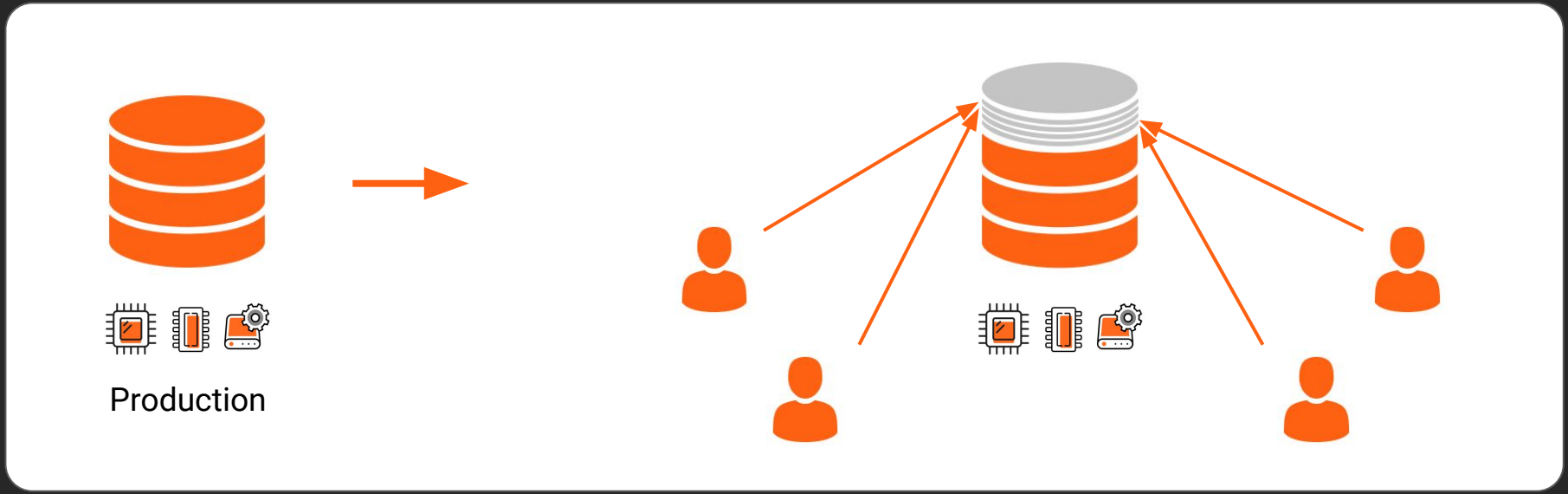

I'M CHECKING NEW INDEXES
FOR OUR POSTGRES DATABASE
NOT ON PRODUCTION, RIGHT?
NOT ON PRODUCTION, RIGHT?
imgflip.com

Traditional DB experiment – thick clones



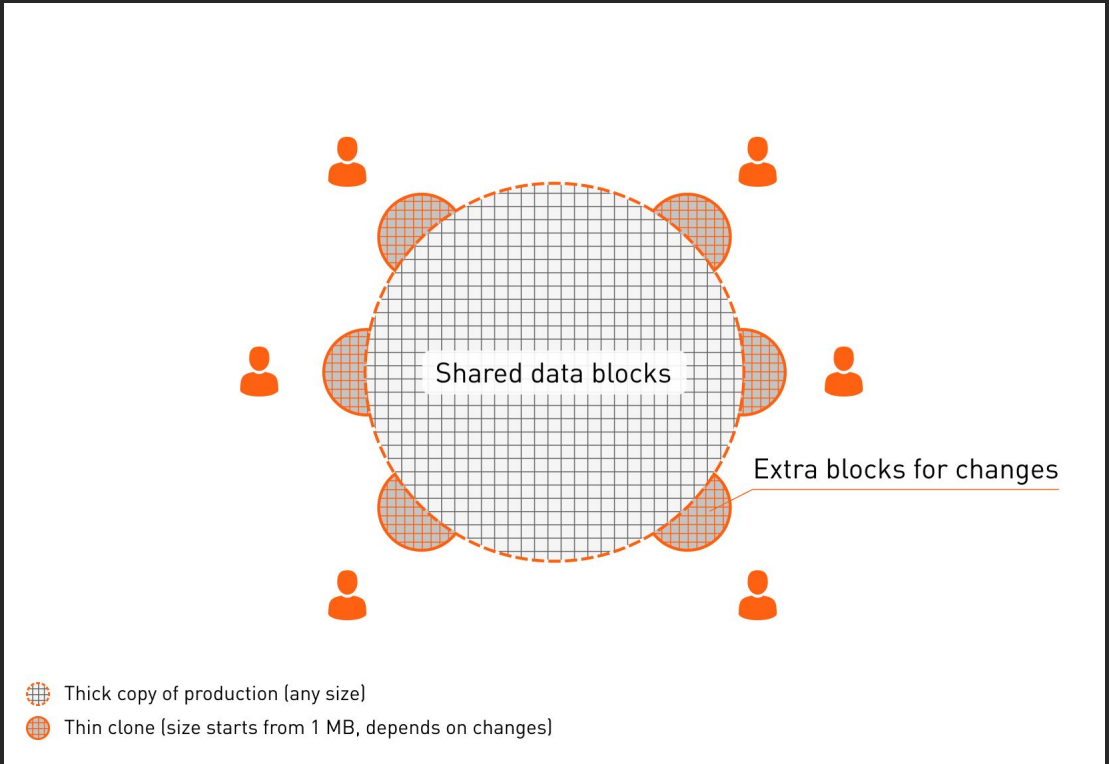
“1 database copy – 10 persons”

Database Lab: use *thin* clones



“1 database copy – 1 person”

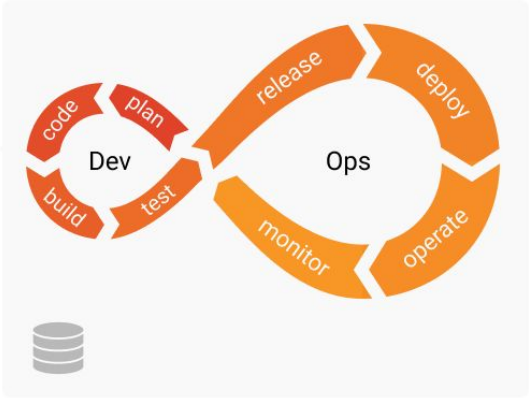
“Thin clones” – Copy-on-Write (CoW)



Thin clones unlock “Shift-left testing”

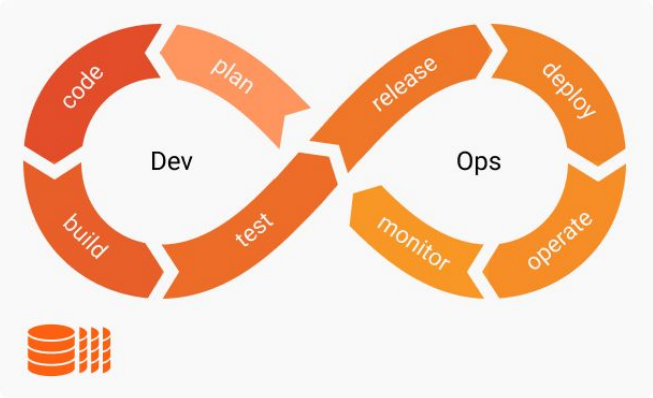
Non-production environment weaknesses are reasons of multiple development problems

Development bottlenecks
(with standard staging DB)

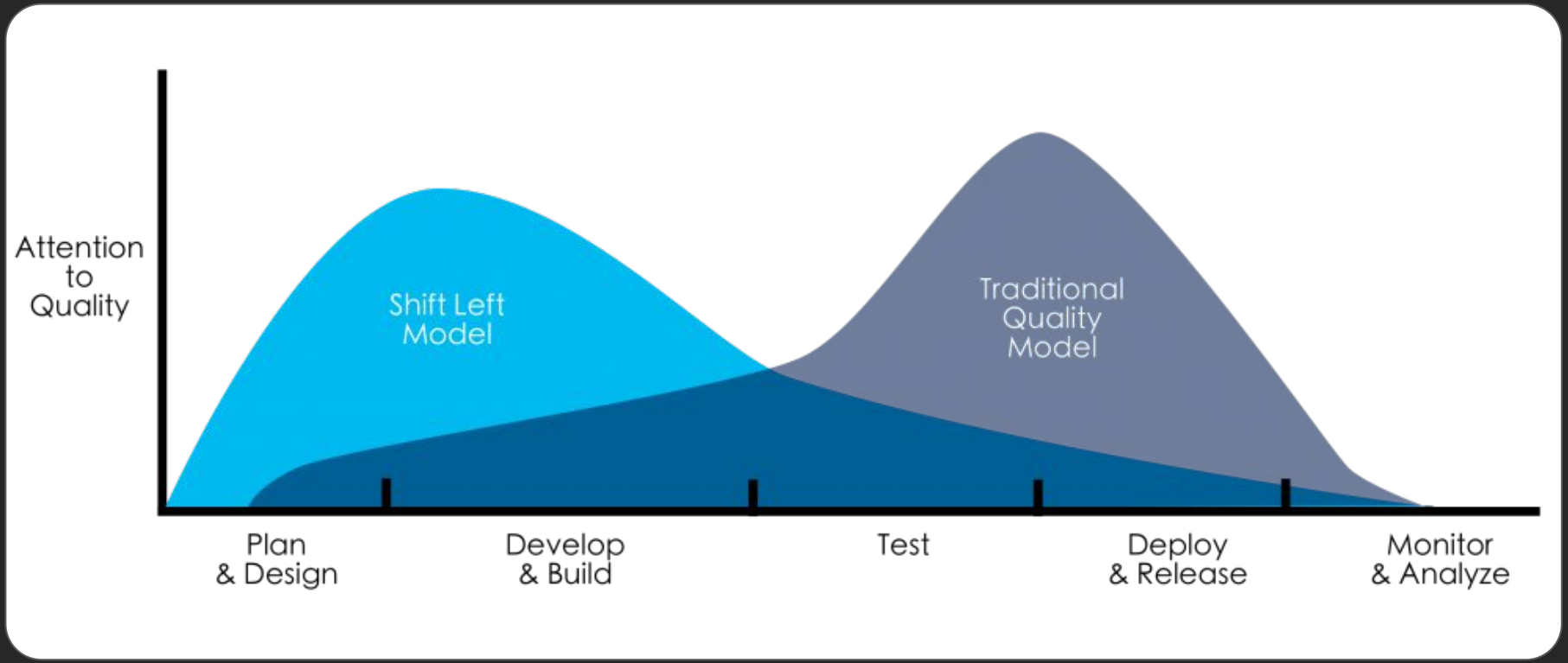


- ✗ Bugs: difficult to reproduce, easy to miss
- ✗ Not 100% of changes are well-verified
- ✗ SQL optimization is hard
- ✗ Each non-prod big DB costs a lot
- ✗ Non-prod DB refresh takes hours, days, weeks

Frictionless development
(with Database Lab)



- ✓ Bugs: easy to reproduce, and fix early
- ✓ 100% of changes are well-verified
- ✓ SQL optimization can be done by anyone
- ✓ Non-prod DB refresh takes seconds
- ✓ Extra non-prod DBs doesn't cost a penny


Attention to Quality
Shift Left Model
Traditional Quality Model
Plan & Design
Develop & Build
Test
Deploy & Release
Monitor & Analyze

Test changes in CI

- Both DO and UNDO steps are supported (can revert)
- CI: test them all
  - Better: DO, UNDO, and DO again

DB schema migration tools

Test changes in CI

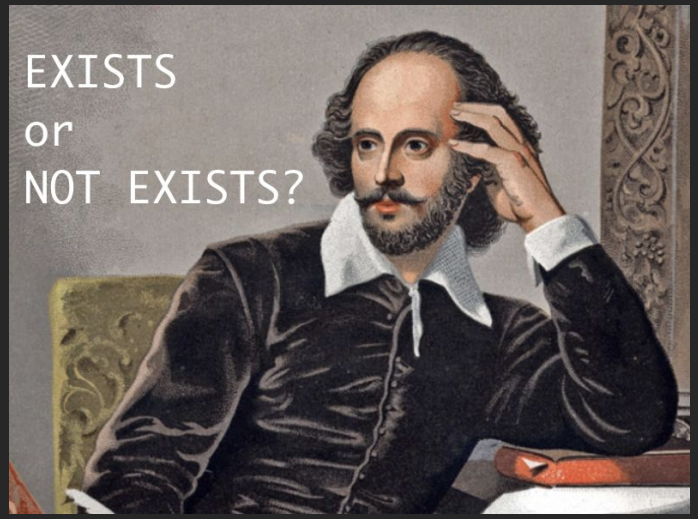
- Both DO and UNDO steps are supported (can revert)
- CI: test them all
  - Better: DO, UNDO, and DO again

Now guess what…

“Thanks” to `IF NOT EXISTS`, we now may leave UNDO empty!

❌ Don't:

- `IF [NOT] EXIST`



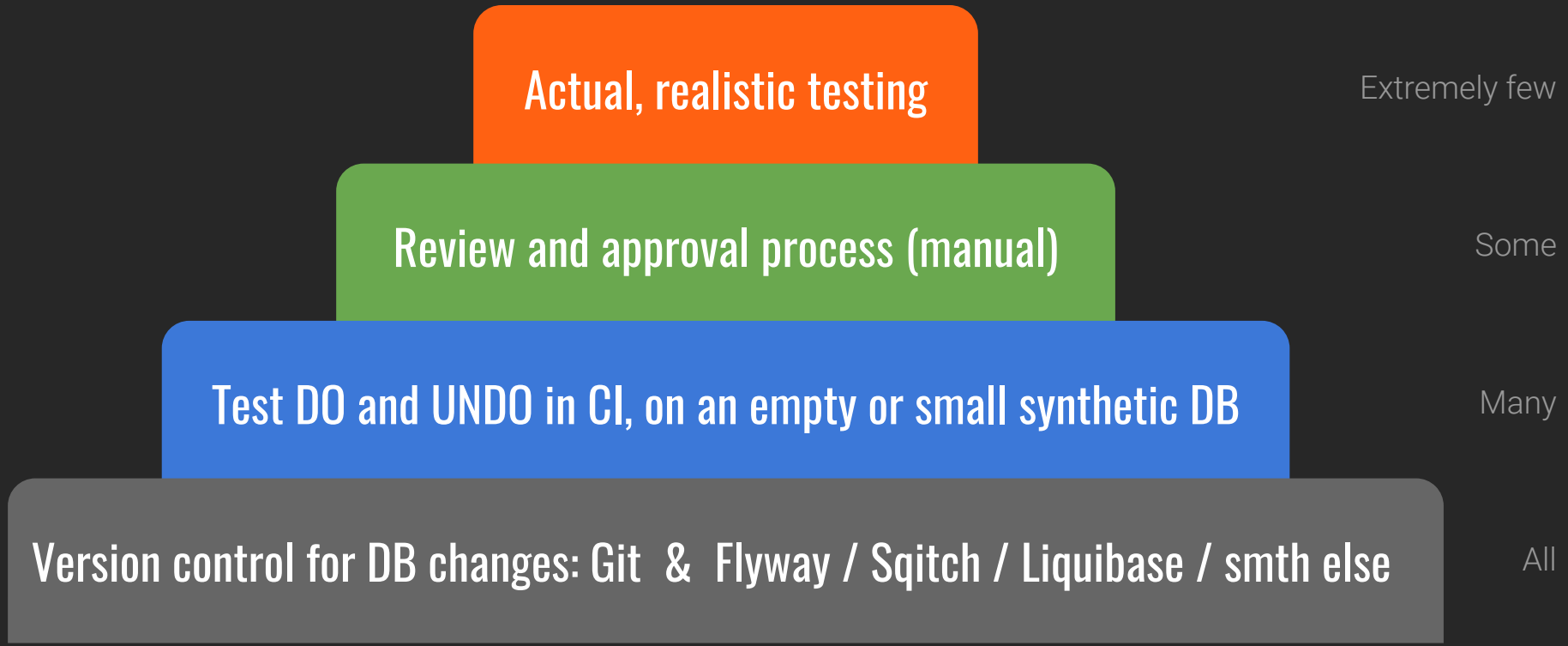
✅ Do:

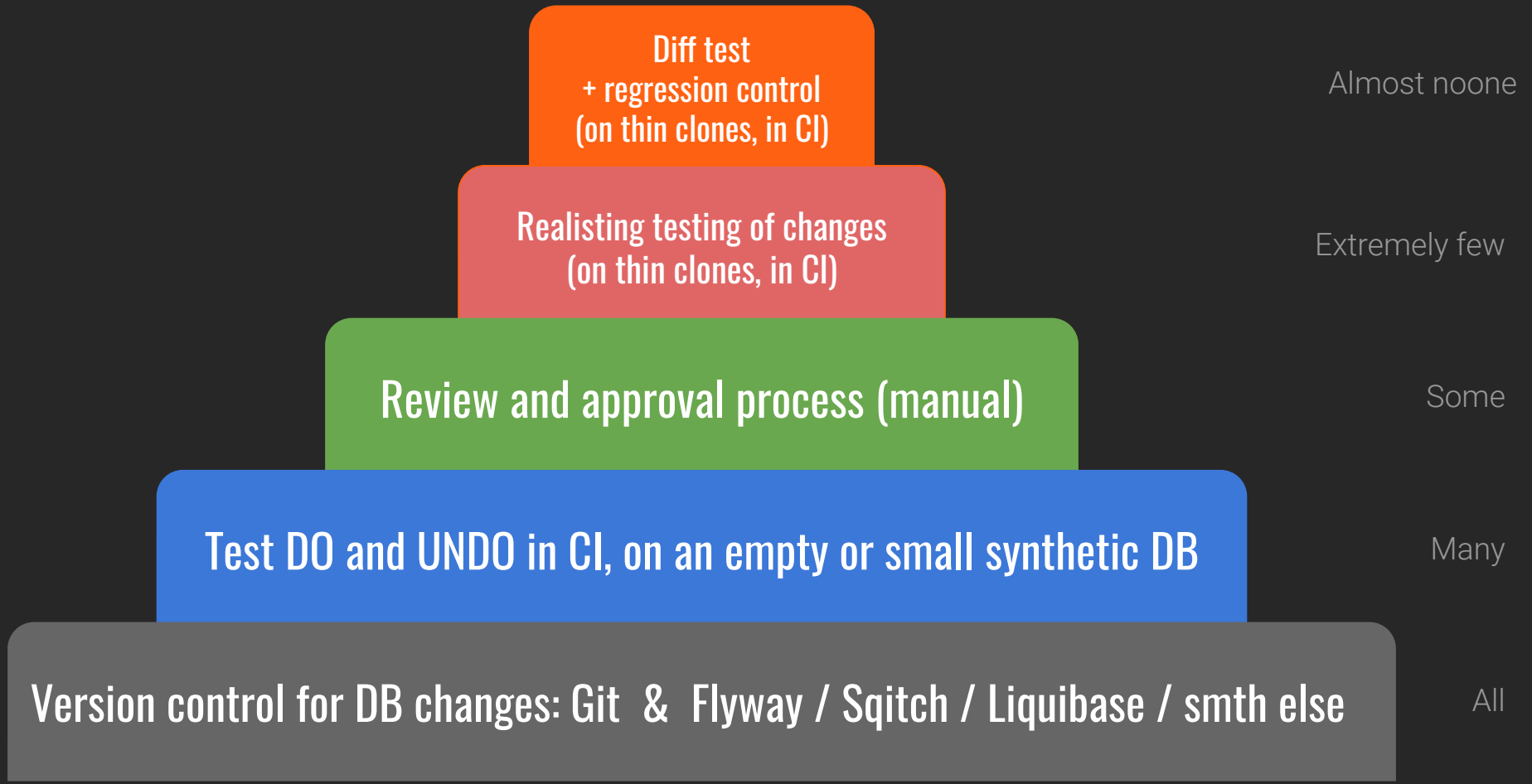
- test DO-UNDO-DO in CI
- keep schema up to date in all envs
- don't ignore or work-around errors

"Three Cases Against IF NOT EXISTS / IF EXISTS in Postgres DDL"

https://postgres.ai/blog/20211103-three-cases-against-if-not-exists-and-if-exists-in-postgresql-ddl

Testing of database changes – the hierarchy of needs

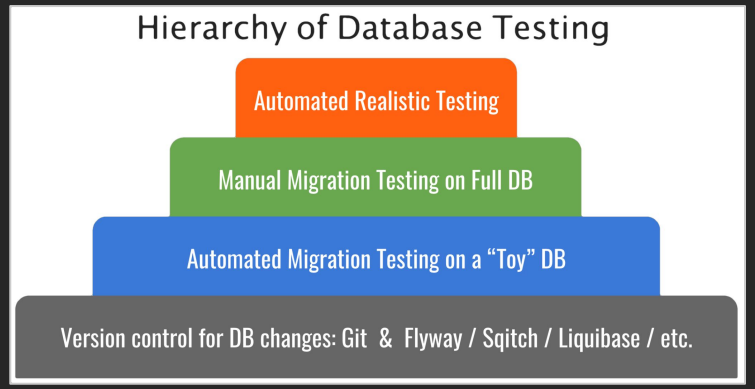



Diff test
+ regression control
(on thin clones, in CI)
Almost noone
Realisting testing of changes
(on thin clones, in CI)
Extremely few
Review and approval process (manual)
Some
Test DO and UNDO in CI, on an empty or small synthetic DB
Many
Version control for DB changes: Git & Flyway / Sqitch / Liquibase / smth else
All

Database Migration Testing with Database Lab

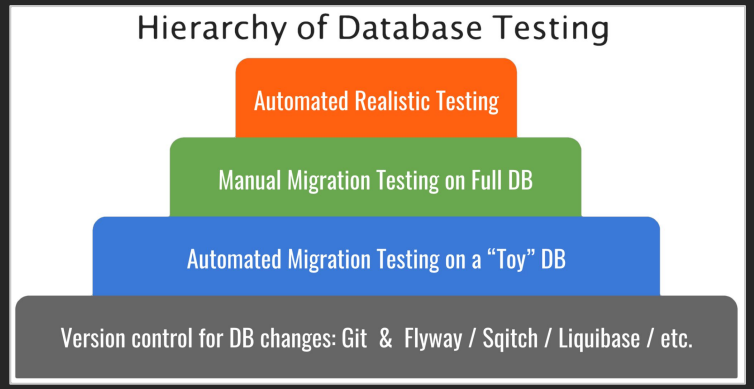
- Realistic migration testing is hard

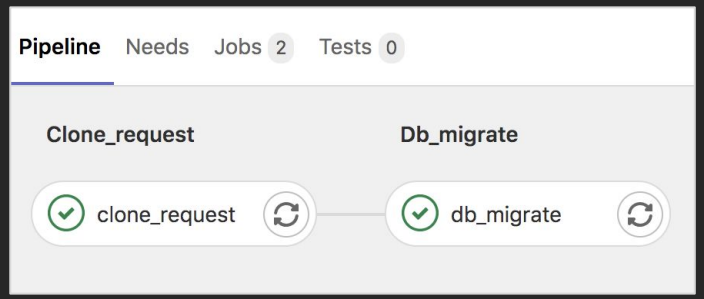
- No testing = unexpected problems



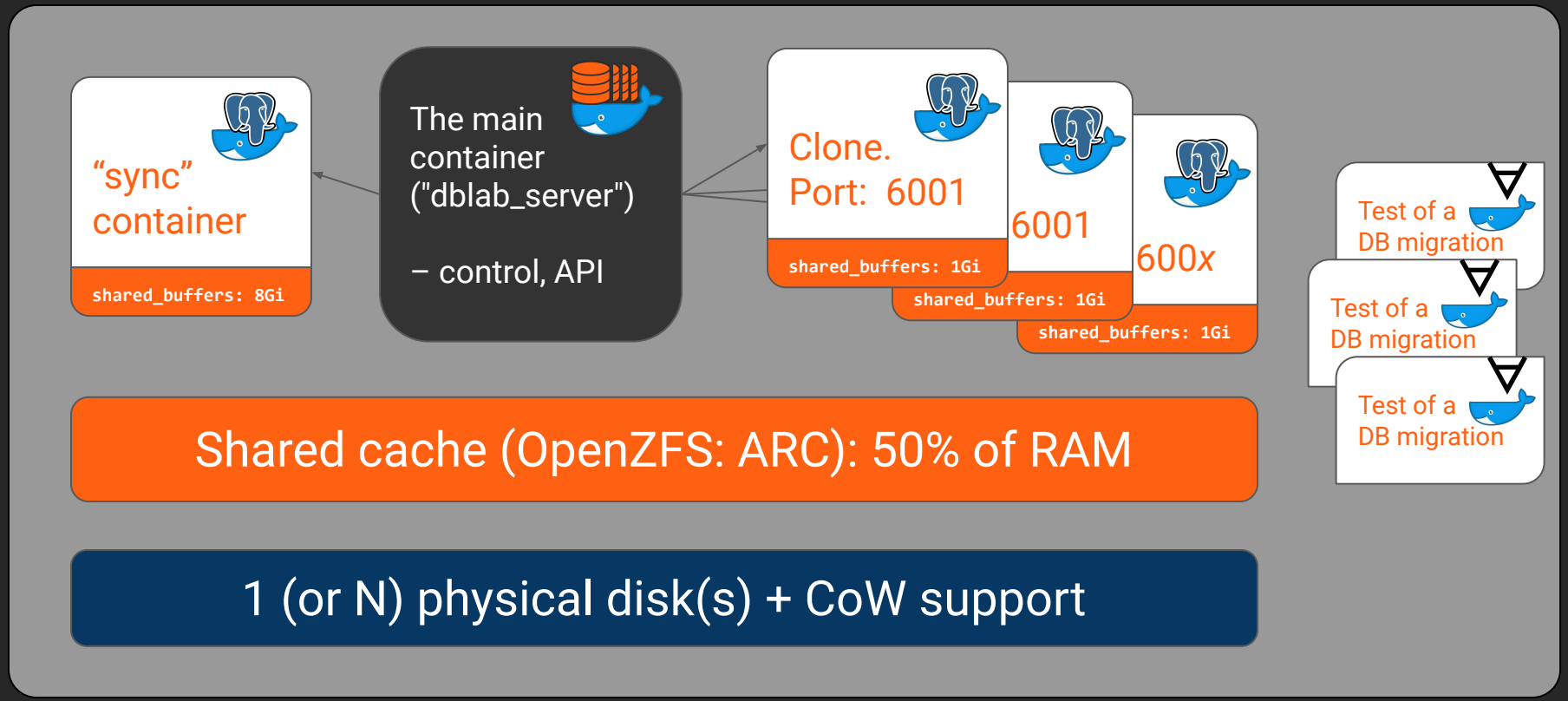
Database Migration Testing with Database Lab

- Realistic migration testing is hard
- No testing = unexpected problems
- Database Lab makes realistic testing *easy*

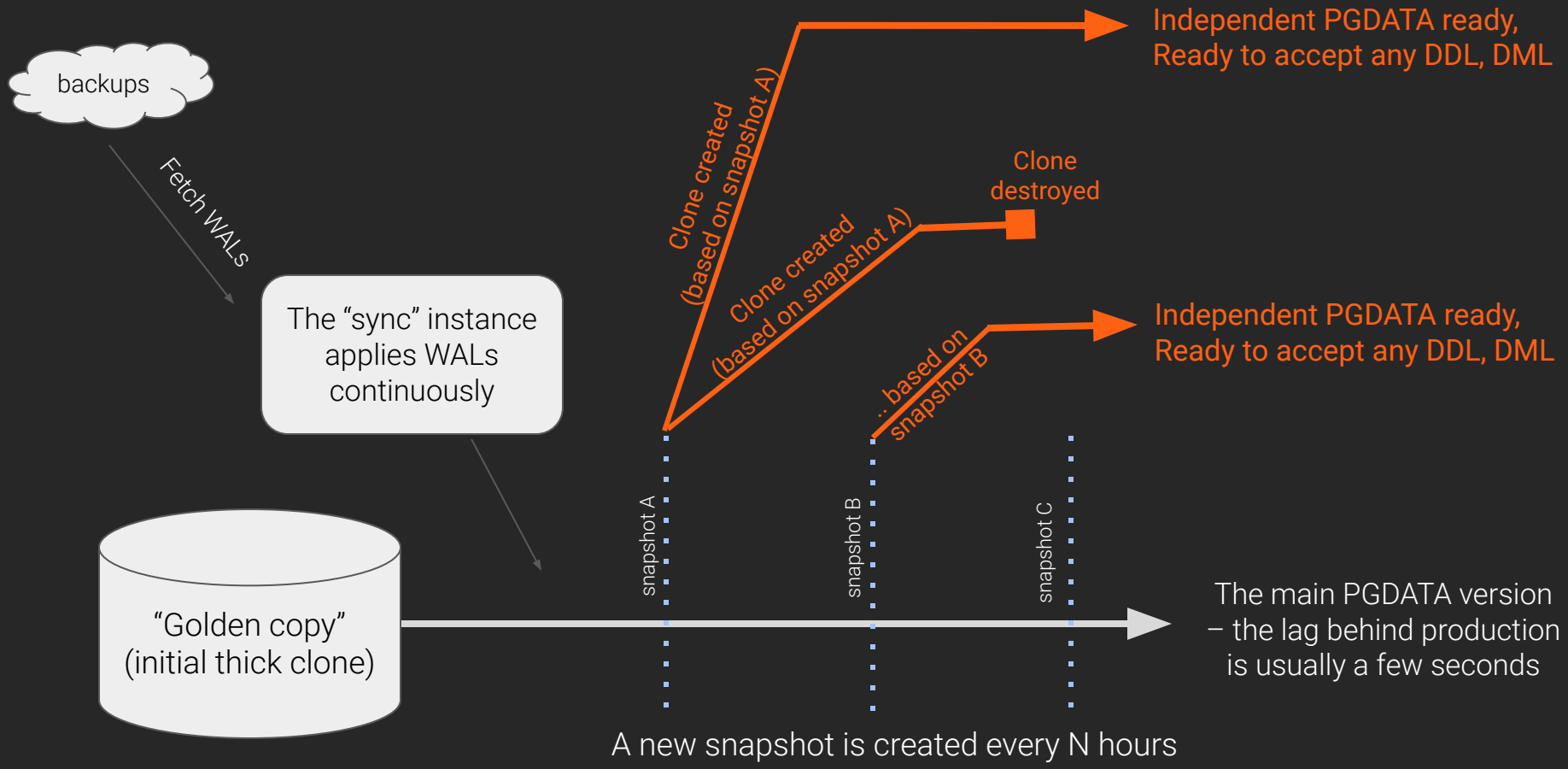




Database Lab Engine (DLE) – what's inside



DLE – the data flow (physical mode)



Major topics of automated (CI) testing on thin clones

- Security

  https://postgres.ai/docs/platform/security

- Capturing dangerous locks

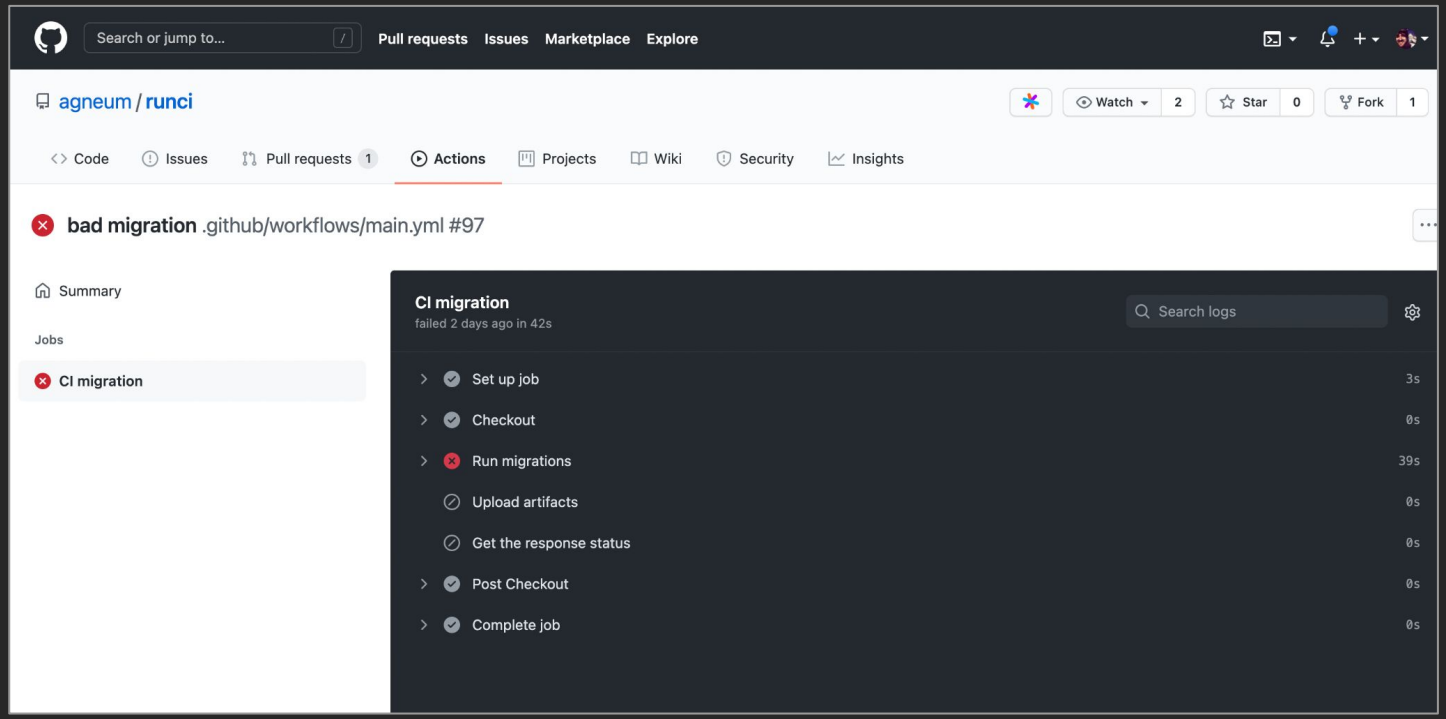
  DB Migration Checker Observer: https://postgres.ai/docs/db-migration-checker

- Forecast production timing

  Timing estimator: https://postgres.ai/docs/database-lab/timing-estimator

How it looks like: the CI part

Example: GitHub Actions:
https://github.com/agneum/runci/runs/2519607920?check_suite_focus=true



Postgres.ai / Database Lab Platform

Postgres.ai Console β

Nikolay

Organization Switch

Demo

This is a Demo organization, once you've explored Database Lab features: Create new organization

Organizations / Demo / Database Lab observed sessions

- Dashboard
- Database Lab
  - Instances
  - Observed sessions
- SQL Optimization
  - Ask Joe BOT
  - History
- Checkup
  - Reports
- Settings
  - General
  - Members
  - Access tokens
  - Billing
  - Audit
- Documentation
- Ask support

Database Lab observed sessions Experimental

Status	Session	Project/Instance	Commit	Checklist	
Passed	#352	-/-	pgbench-account-index/b98e2978e93ca6ef9527aec762d275bc9f52c9a5	✓✓✓	45s created 21 minutes ago by NikolayS
Passed	#351	-/-	pgbench-account-index/b696bf5cd75188eb0f9c21bd3f2297a424547d7e	✓✓✓	43s created 5 hours ago by agneum
Failed	#350	-/-	pgbench-account-index/18c91f6a50cb5b8c6df7466b5a0fbc0779b880d8	✗✓✓	2s created 5 hours ago by agneum
Failed	#349	-/-	pgbench-account-index/bea1571746ce48552eebbee3b33fb9d346971bb9	✓✗✓	27s created 6 hours ago by agneum
Passed	#348	-/-	master/ee898e6ae8b0fdbb3351d1ea79a3698015a861a3	✓✓✓	3s created 6 hours ago by agneum
Passed	#347	-/-	set-up-workflow/c4d2432aa6f39a7aeb02e5ae7729b1ee68ea4c85	✓✓✓	3s created 6 hours ago by agneum

Postgres.ai / Database Lab Platform

Database Lab observed sessions Experimental

Status	Session	Project/Instance	Commit	Checklist	
Passed	#352	-/-	pgbench-account-index/b98e2978e93ca6ef9527aec762d275bc9f52c9a5	✓✓✓	45s created 21 minutes ago by NikolayS
Passed	#351	-/-	pgbench-account-index/b696bf5cd75188eb0f9c21bd3f2297a424547d7e	✓✓✓	43s created 5 hours ago by agneum
Failed	#350	-/-	pgbench-account-	✗✓✓	2s created 5 hours ago by

Database Lab observed session #349 Experimental

Summary

Status:	✕ Failed
Session:	#349
Project:	-
DLE instance:	
DLE version:	2.4.0-beta.3-3-g8b57d6d-20210707-0301
Data state at:	2021-07-07 11:39:05 UTC
Duration:	27s
Created:	6 hours ago
Branch:	pgbench-account-index
Commit:	bea1571746ce48552eebbee3b33fb9d346971bb9
Triggered by:	agneum (akartasov)
PR/MR:	-
Changes:	-

Checklist

- Passed — overall_success
- Failed — Dangerous locks are not observed during the session
  (3 intervals with locks of 10 allowed)
- Passed — Session duration is within allowed interval
  (spent 27s of the allowed 10m)

Observed intervals and details

Show intervals

Code | Issues | Pull requests 1 | Actions | Security | Insights | Settings

test -- this should fail

pgbench-account-in… 62554c7

Re-run jobs

.github/workflows/main.yml
on: push

CI migration

CI migration

failed 24 minutes ago in 58s

Search logs

DB migrations checker with DLE 50s


```
73          "warning": "
{\"datname\":\"test_small\",\"relation\":\"16780\",\"transactionid\":null,\"mode\":\"AccessExclusiveLock\",\"locktype\":\"relation\",\"granted\":tru
e,\"usename\":\"ci_NikolayS\",\"query\":\"create index /****concurrently****/ bid_idx on pgbench_accounts(bid);\",\"query_start\":\"2021-07-
08T14:08:50.276435+00:00\",\"state\":\"active\",\"wait_event_type\":\"LWLock\",\"wait_event\":\"WALWriteLock\",\"xact_start\":\"2021-07-
08T14:08:50.276435+00:00\",\"xact_duration\":\"00:00:18.398015\",\"query_start\":\"2021-07-
08T14:08:50.276435+00:00\",\"query_duration\":\"00:00:18.398017\",\"state_change\":\"2021-07-
08T14:08:50.276438+00:00\",\"state_changed_ago\":\"00:00:18.398015\",\"pid\":44}\n"
74        },
75        {
76          "started_at": "2021-07-08T14:09:08.675785043Z",
77          "duration": 8.688464841,
78          "warning": ""
79        }
80      ],
81      "summary": {
82        "total_duration": 29.056247725,
83        "total_intervals": 3,
84        "warning_intervals": 2,
85        "checklist": {
86          "overall_success": true,
87          "session_duration_acceptable": true,
88          "no_long_dangerous_locks": false
89        }
90      }
91    }
92  }
93 }
```

Case study: GitLab.com, testing database changes using Database Lab

- Full automation
- GitLab CI/CD pipelines securely work with Database Lab
- Database Lab clones ~14 TiB database in ~15 seconds

More:

- https://docs.gitlab.com/ee/architecture/blueprints/database_testing/
- https://postgres.ai/resources/case-studies/gitlab

Overview 26 | Commits 5 | Pipelines 7 | Changes 7

6 unresolved threads

Dmytro Zaporozhets (DZ) @dzaporozhets · 1 week ago

Owner

@abrandl as per !54466 (comment 511910471) can you please review this merge request?

gitlab-org/database-team/gitlab-com-database-testing @project_278964_bot2 · 1 week ago

Maintainer

Database migrations

Migrations included in this change have been executed on gitlab.com data for testing purposes. For details, please see the migration testing pipeline (limited access). Note that this includes pending migrations from `master`.

Migration	Total runtime	Result	DB size change
20210215144909	1.2 s	✅	+0.00 B
20210218105431	0.6 s	💥	+0.00 B

Migration: 20210215144909

- Duration: 1.2 s
- Database size change: +0.00 B

Migration: 20210218105431

- Duration: 0.6 s
- Database size change: +0.00 B

Query	Calls	Total Time	Max Time	Mean Time	Rows
`ALTER TABLE "ci_builds" DROP COLUMN "artifacts_file" /*application:test*/` •••	1	12.9 ms	12.9 ms	12.9 ms	0

Artifacts

- Database testing statistics
- Database Lab Instance

February 19, 2021 – https://gitlab.com/gitlab-org/gitlab/-/merge_requests/54564#note_512678910

Summary – available in PR/MR and visible to whole team

- When, who, status
- Duration (in the Lab + estimated for production)
- Size changes, new objects
- Dangerous locks (!!)
- Error stats
- Transaction stats
- Query analysis summary
- Tuple stats
- WAL generated, checkpoitner/bgwriter stats
- Temp files stats

Example: https://gitlab.com/postgres-ai/database-lab/-/snippets/2083427

More artifacts, details – restricted access

- System monitoring (resources utilization)
- `pg_stat_*`
- `pg_stat_statements`, `pg_stat_kcache`
- `logerrors`
- Postgres log
- pgBadger (html, json)
- wait event sampling
- perf tracing, flamegraphs; or eBPF
- Estimated production timing

https://gitlab.com/postgres-ai/database-lab/-/issues/226

Diff is not enough!

Ideal testing of DB changes:

1. Regression test 1 (“pre”)
2. Test the change
3. Regression test 2 (“post)
4. Compare “pre” vs. “post”

Diff is not enough!

Ideal testing of DB changes:

1. Regression test 1 (“pre”)
2. Test the change
3. Regression test 2 (“post)
4. Compare “pre” vs. “post”

We can apply the same methodology to reproduce the workload as for multi-user testing

Diff is not enough!

Ideal testing of DB changes:

1. Regression test 1 (“pre”)
2. Test the change
3. Regression test 2 (“post)
4. Compare “pre” vs. “post”

We can apply the same methodology to reproduce the workload as for multi-user testing

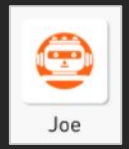
BUT: remain in “single-user testing” area – test using 1 connection

…in CI/CD pipelines!

Database experiments on thin clones – yes and no

✅ Yes

- Check execution plan
  - EXPLAIN (ANALYZE, BUFFERS)
    - (timing is different; structure and buffer numbers – the same)
  - Example – Joe chatbot



- DB migration testing
  - DDL, DML - in CI/CD
  - Example: Database Lab's DB Migration Checker
- Performance regression testing
  - Control performance of specific set of queries – in CI/CD

Database Lab

❌ No

- Stress testing

  (but single-user experiments are OK!)
- Tests related to HA/DR
  - backups
    - (but useful to check WAL stream, recover records by mistake)
  - hot standby
    - (but useful to offload very long-running SELECTs)

Single- & multi-user DB experiments – the difference

	Workload	CPU cores utilized	Focus	Shared env ok?	Use cases
Single-user experiments	1 query or a sequence – a single DB session	1 or a few	Execution plan: structure, rows, BUFFERS *(not timing!)* `EXPLAIN (ANALYZE, BUFFERS)`	Yes (thin clones work best – Database Lab)	SQL optimization, App changes (schema, data)
Multi-user experiments	Replayed / simulated – many DB sessions	All (or most)	Latency & throughput Utilization, saturation	No (only exclusive use)	Capacity planning Infrastructure changes (hw and sw upgrades, topology change, config tuning, etc.)

Summary – two key ideas

1. Use **large** databases for testing in CI/CD pipelines

   Consider: Database Lab with DB Migration Checker

2. For performance testing (and optimization), focus on:

   - **execution plans**
   - I/O: rows, **buffer** numbers (not timing!)

Thank you!

Slack (EN): **Slack.Postgres.ai**

Twitter: **@Database_Lab**

TO BE CONTINUED...

Some examples of failures due to lack of testing

- Incompatible changes – production has different DB schema than dev & test
- Cannot deploy – hitting `statement_timeout` – too heavy operations

- During deployment, we've got a failover
- Deployment lasted 10 minutes, the app was very slow (or even down)

- Two weeks after deployment, we realize that the high bloat growth we have now has been introduced by that deployment
- Deployment succeeded, but then we have started to see errors

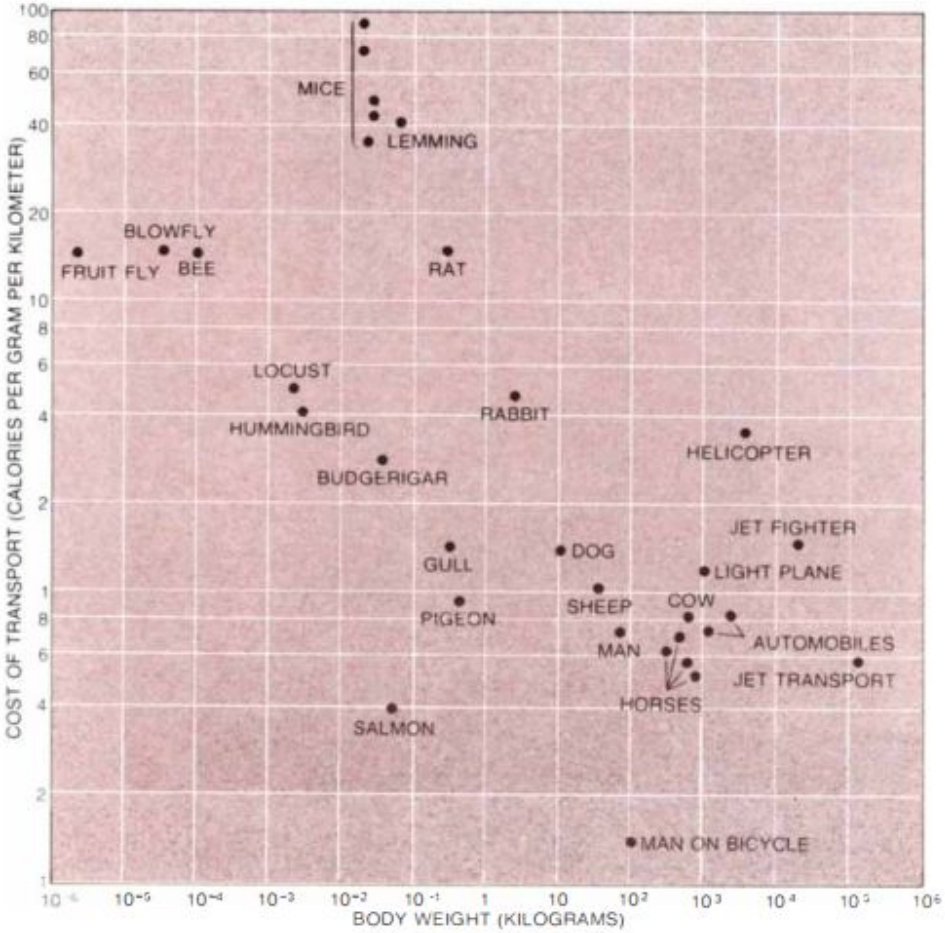
We need better tools

SCIENTIFIC AMERICAN

BICYCLE TECHNOLOGY

ONE DOLLAR

March 1973



Steve Jobs (1980)

1) We, humans, are great tool-makers.
   We amplify human abilities.

2) Something special happens
   when you have 1 computer and 1 person.

   It's very different that having 1 computer and 10 persons.

DB migration testing – “stateful tests in CI”

What we want from testing of DB changes:

- Ensure the change is valid
- It will be executed in appropriate time
- It won’t put the system down

…and:

- What to expect? (New objects, size change, duration, etc.)

Perfect Lab for database experiments

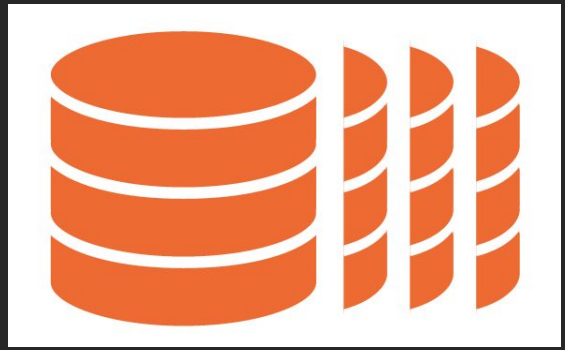
- Realistic conditions – as similar to production as possible
  - The same schema, data, environment as on production
  - Very similar background workload
- Full automation
- “Memory” (store, share details)
- Low iteration overhead (time & money)
- Everyone can test independently

allowed to fail → allowed to learn

Database experiments with Database Lab today (2021)

- Realistic conditions – as similar to production as possible
  - The same schema, data, environment as on production
  - ~~Very similar background workload~~
- Fine automation
- “Memory” (store, share details)
- Low iteration overhead (time & money)
- Everyone can test independently

  able to fail → able to learn



Why Database Lab was created

- Containers, OverlayFS (file-level CoW)

  CI: `docker pull ... && docker run ...`

  – OK only for tiny (< a few GiB) databases

- Existing solutions: Oracle Snap Clones, Delphix, Actifio, etc. $$$$, not open

  – OK only for very large enterprises

Companies that do need it today

- 10+ engineers
- Multiple backend teams (or plans to split soon)
- Microservices (or plans to move to them)
- 100+ GiB databases
- Frequent releases

How snapshots are created (ZFS version)

- Create a “pre” ZFS snapshot (R/O)
- Create a “pre” ZFS clone (R/W)
- DLE launches a temporary “promote” container
  - If needed, performs “preprocessing” steps (bash)
  - Uses “pre” clone to run Postgres and promote it to primary state
  - If needed, performs “preprocessing” SQL queries
  - Performs a clean shutdown of Postgres
- Create a final ZFS snapshot that will be used for cloning

Making the process secure: where to place the DLE?

PII here

Production

The big wall

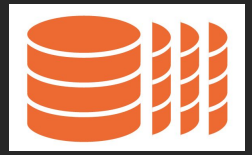
Dev & Test

CI runners here

Where to place the DLE? Current approach

PII here

Production



High-level API calls

The big wall

Dev & Test

CI runners here

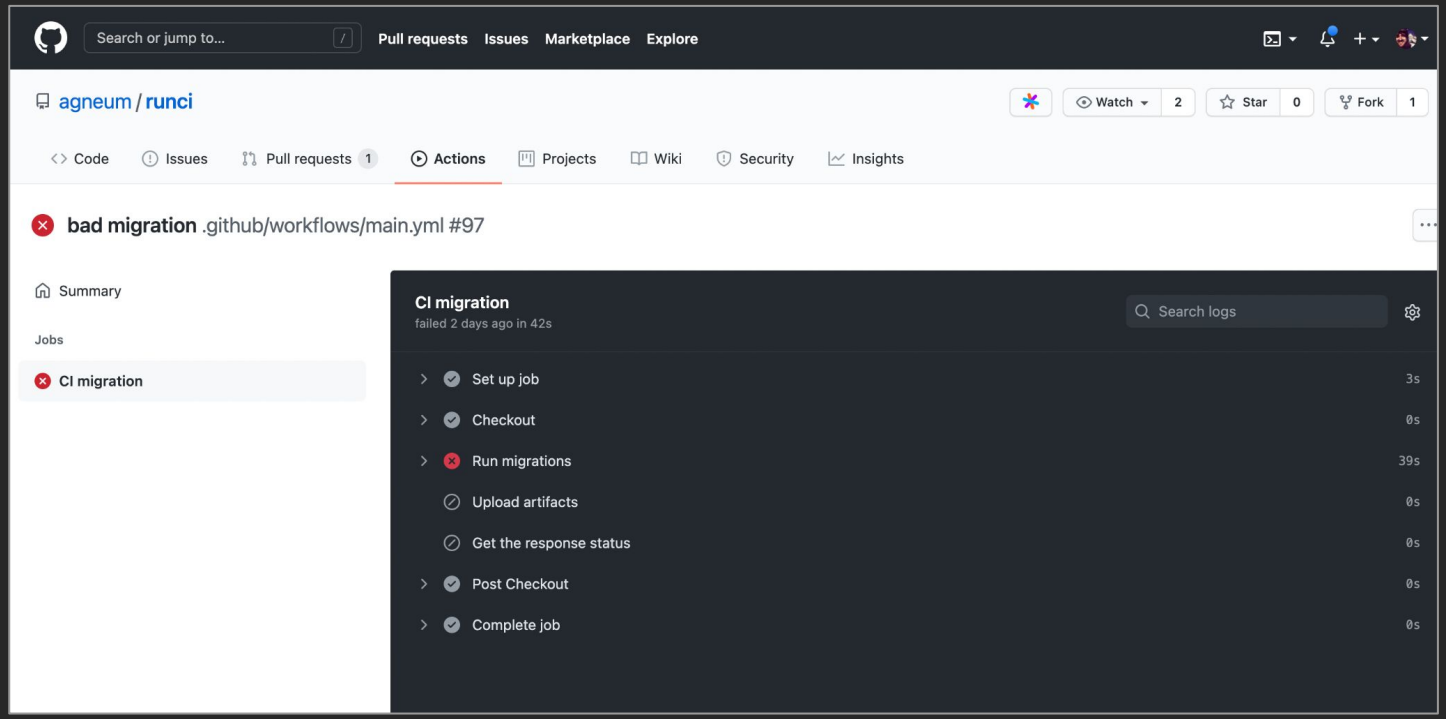
GitHub

GitLab

Jenkins

How it looks like: CI part

Example: GitHub Actions:
https://github.com/agneum/runci/runs/2519607920?check_suite_focus=true



More about dangerous lock detection

Postgres.ai Console β

Nikolay

Organization Switch

Demo

- Dashboard
- Database Lab
  - Instances
  - Observed sessions
- SQL Optimization
  - Ask Joe BOT
  - History
- Checkup
  - Reports
- Settings
  - General
  - Members
  - Access tokens
  - Billing
  - Audit
- Documentation
- Ask support

Organizations / Demo / Observed sessions / Database Lab observed session #166

Database Lab observed session #166 Experimental

Summary

Status:	✕ Failed
Session:	#166
Project:	-
DLE instance:	
Duration:	2m, 5s
Created:	2 days ago
Branch:	master
Commit:	-
Triggered by:	-
PR/MR:	-

Checklist

✕ Failed — Dangerous locks are not observed during the session
(125 intervals with locks of 1 allowed)

✓ Passed — Session duration is within allowed interval
(spent 2m, 5s of the allowed 5m)

Observed intervals and details

Hide intervals

Started at	Duration
✓ 2021-02-26 16:18:16 UTC	1s
✓ 2021-02-26 16:18:17 UTC	1s
! 2021-02-26 16:18:18 UTC	1s

{"datname":"test","relation":"pgbench_branches","transactionid":null,"mode":"AccessExclusiveLock","locktype":"relation","granted":true,"usename":"dblab_user_1"
,"query":"drop table pgbench_branches;","query_start":"2021-02-26T16:18:18.021939+00:00","state":"idle in

Overview 26 | Commits 5 | Pipelines 7 | Changes 7

6 unresolved threads

Dmytro Zaporozhets (DZ) @dzaporozhets · 1 week ago

Owner

@abrandl as per !54466 (comment 511910471) can you please review this merge request?

gitlab-org/database-team/gitlab-com-database-testing @project_278964_bot2 · 1 week ago

Maintainer

Database migrations

Migrations included in this change have been executed on gitlab.com data for testing purposes. For details, please see the migration testing pipeline (limited access). Note that this includes pending migrations from `master`.

Migration	Total runtime	Result	DB size change
20210215144909	1.2 s	✅	+0.00 B
20210218105431	0.6 s	💥	+0.00 B

Migration: 20210215144909

- Duration: 1.2 s
- Database size change: +0.00 B

Migration: 20210218105431

- Duration: 0.6 s
- Database size change: +0.00 B

Query	Calls	Total Time	Max Time	Mean Time	Rows
`ALTER TABLE "ci_builds" DROP COLUMN "artifacts_file" /*application:test*/` •••	1	12.9 ms	12.9 ms	12.9 ms	0

Artifacts

- Database testing statistics
- Database Lab Instance

February 19, 2021 – https://gitlab.com/gitlab-org/gitlab/-/merge_requests/54564#note_512678910

Example: GitLab.com, testing database changes using Database Lab

- Full automation
- GitLab CI/CD pipelines securely work with Database Lab
- Database Lab clones ~10 TiB database in ~10 seconds

Read their blueprint:

https://docs.gitlab.com/ee/architecture/blueprints/database_testing/

More about production timing estimation

Experimental, WIP: https://postgres.ai/docs/database-lab/timing-estimator

```
estimator:
    readRatio: 1
    writeRatio: 1
    profilingInterval: 20ms
    sampleThreshold: 100
```

```
LOG: Profiling process 63 with 10ms sampling
% time      seconds wait_event
------ ------------ ----------------------------
57.30     17.715111 IO.DataFileRead
25.53      7.893916 Running
3.55       1.097738 IO.DataFileExtend
2.55       0.787341 LWLock.WALWriteLock
2.25       0.696663 IO.BufFileRead
2.14       0.662457 IO.BufFileWrite
2.12       0.654081 IO.WALInitWrite
1.62       0.499461 IO.WALInitSync
1.09       0.335660 IO.WALWrite
0.98       0.301637 IO.DataFileImmediateSync
0.81       0.250249 IO.WALSync
0.07       0.020805 LWLock.WALBufMappingLock
------ ------------ ----------------------------
100.00    30.915119
```

Summary:

```
Time: 3.148 s
  - planning: 0.168 ms
  - execution: 3.147 s (estimated* for prod: 2.465...2.693 s)
    - I/O read: 627.267 ms
    - I/O write: 3.644 ms

Shared buffers:
  - hits: 1016393 (~7.80 GiB) from the buffer pool
  - reads: 16395 (~128.10 MiB) from the OS file cache, including disk I/O
  - dirtied: 16395 (~128.10 MiB)
  - writes: 280 (~2.20 MiB)
```

Summary – available in PR/MR and visible to whole team

- When, who, status
- Duration (in the Lab + estimated for production)
- Size changes, new objects
- Dangerous locks
- Error stats
- Transaction stats
- Query analysis summary
- Tuple stats
- WAL generated, checkpointer/bgwriter stats
- Temp files stats

Example (WIP): https://gitlab.com/postgres-ai/database-lab/-/snippets/2083427

More artifacts, details – restricted access

- System monitoring (resources utilization)
- pg_stat_*
- pg_stat_statements, pg_stat_kcache
- logerrors
- Postgres log
- pgBadger (html, json)
- wait event sampling
- perf tracing, flamegraphs; or eBPF
- Estimated production timing

https://gitlab.com/postgres-ai/database-lab/-/issues/226

Database Lab Roadmap

https://postgres.ai/docs/roadmap

- Lower the entry bar
  - Simplify installation
  - Simplify the use
  - Easy to integrate
  - *** **** * *******

Where to start

Postgres.ai/docs/

Slack.Postgres.ai