GoCardless: Scaling database backups

lawrence@gocardless.com

GoCardless

- Global network for recurring, pull-based payments
- `POST /payments`
- 400 people, 5 offices worldwide

lawrence@gocardless.com

Me

- Tech lead of Core Infrastructure, 6 SREs, PM & EM
- Support GoCardless engineers with reliable infrastructure
- Consult with developers on how to leverage tools

lawrence@gocardless.com

A project in the life of an SRE

- Set the **scene**
- Walkthrough **plan**
- Reflect on **outcomes**

lawrence@gocardless.com

Scene

lawrence@gocardless.com

Scene: Monolith

- Payments-Service powers our API
- Rails app using Postgres, database is 3TB x 2^n
- Payment data is important, should probably back it up...

Scene: Legacy Postgres backups

- Heap `/data/postgres`
- WAL `/data/postgres/pg_{xlog,wal}`
- Barman takes full copies of heap, collects WAL continuously

Scene: Good backups

Last backup taken at 1pm. Database is destroyed at 1:05pm.
Recover at 1:30pm.

lawrence@gocardless.com

Scene: Good backups

- RTO: Recovery time objective (25m, 1:05->1:30)
- RPO: Recovery point objective (5m, 1:00->1:05)

lawrence@gocardless.com

Scene: Goals

- RTO is 30m (99.95%)
- RPO is 1m (1000 payments)

lawrence@gocardless.com

Scene: Problems

- Speed limit, 440MB/s, 2-4hrs
- Data is money

lawrence@gocardless.com

Scene: Breaking the speed limit

- Disk snapshots don't have this limit
- Incremental, scale sub-linerly
- <3m to create, <10m to restore, non-lazy

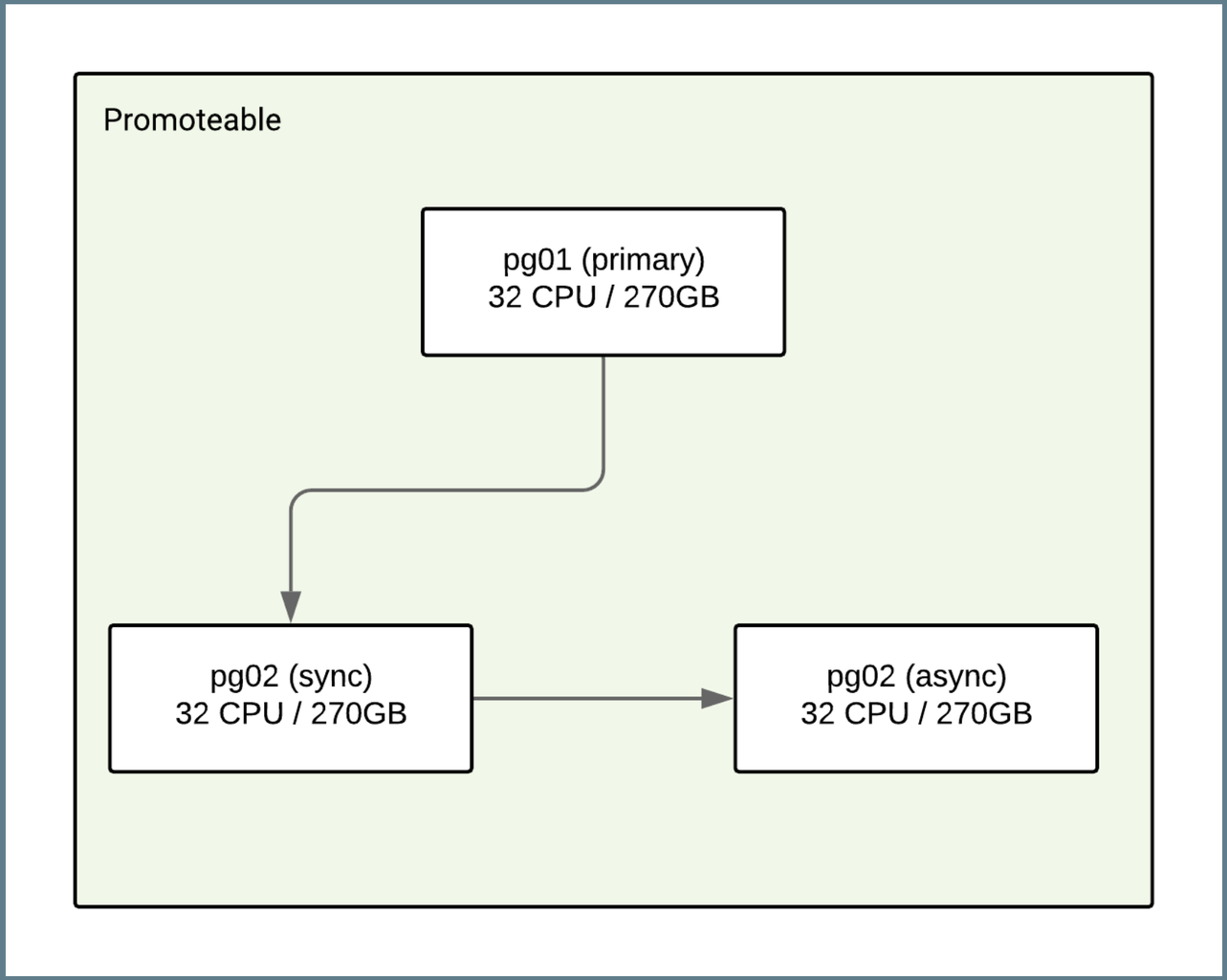
lawrence@gocardless.com

Plan

lawrence@gocardless.com

Plan: Step 1, nominate backup node

lawrence@gocardless.com



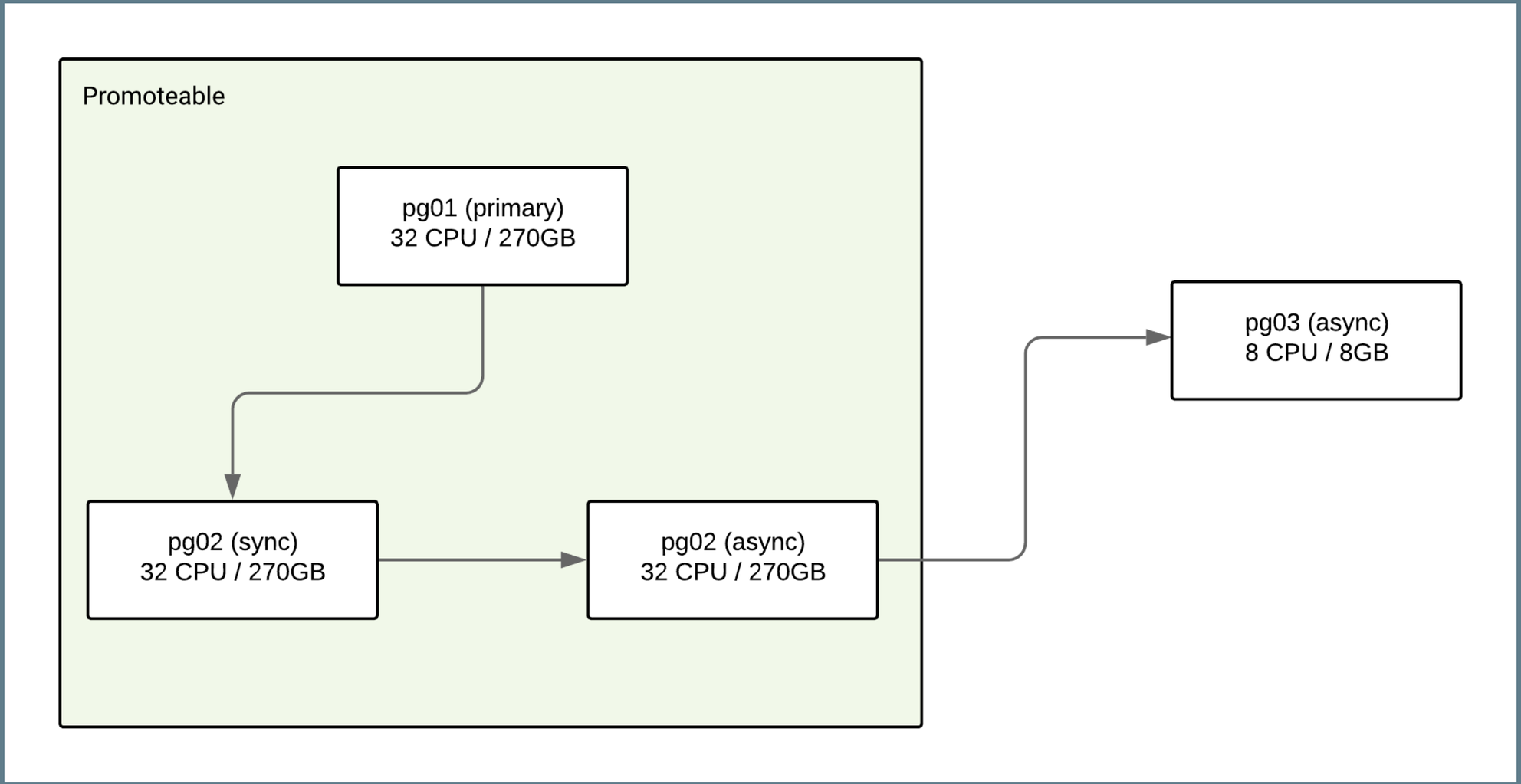
lawrence@gocardless.com

```go
// findBestNewMasters identifies the DBs that are elegible to become a new master. We do
// this by selecting from valid standbys (those keepers that follow the same timeline as
// our master, and have an acceptable replication lag) and also selecting from those nodes
// that are valid to become master by their status.
func (s *Sentinel) findBestNewMasters(cd *cluster.ClusterData, masterDB *cluster.DB) []*cluster.DB {
        bestNewMasters := []*cluster.DB{}
        for _, db := range s.findBestStandbys(cd, masterDB) {
                if k, ok := cd.Keepers[db.Spec.KeeperUID]; ok && k.Status.NeverMaster {
                        log.Infow("ignoring keeper since it cannot be master (--never-master)", "db", db.UID, "keeper", db.Spec
                        continue
                }

                bestNewMasters = append(bestNewMasters, db)
        }

```

lawrence@gocardless.com



lawrence@gocardless.com

```
=== Cluster Info ===

Master Keeper: stolon_production_1

===== Keepers/DB tree =====

stolon_production_1 (master)
└─stolon_production_2 (sync)
  ├─stolon_production_0 (async)
  └─stolon_async_production_0 (async)
```

lawrence@gocardless.com

Plan: Step 2, schedule & prune backups

lawrence@gocardless.com

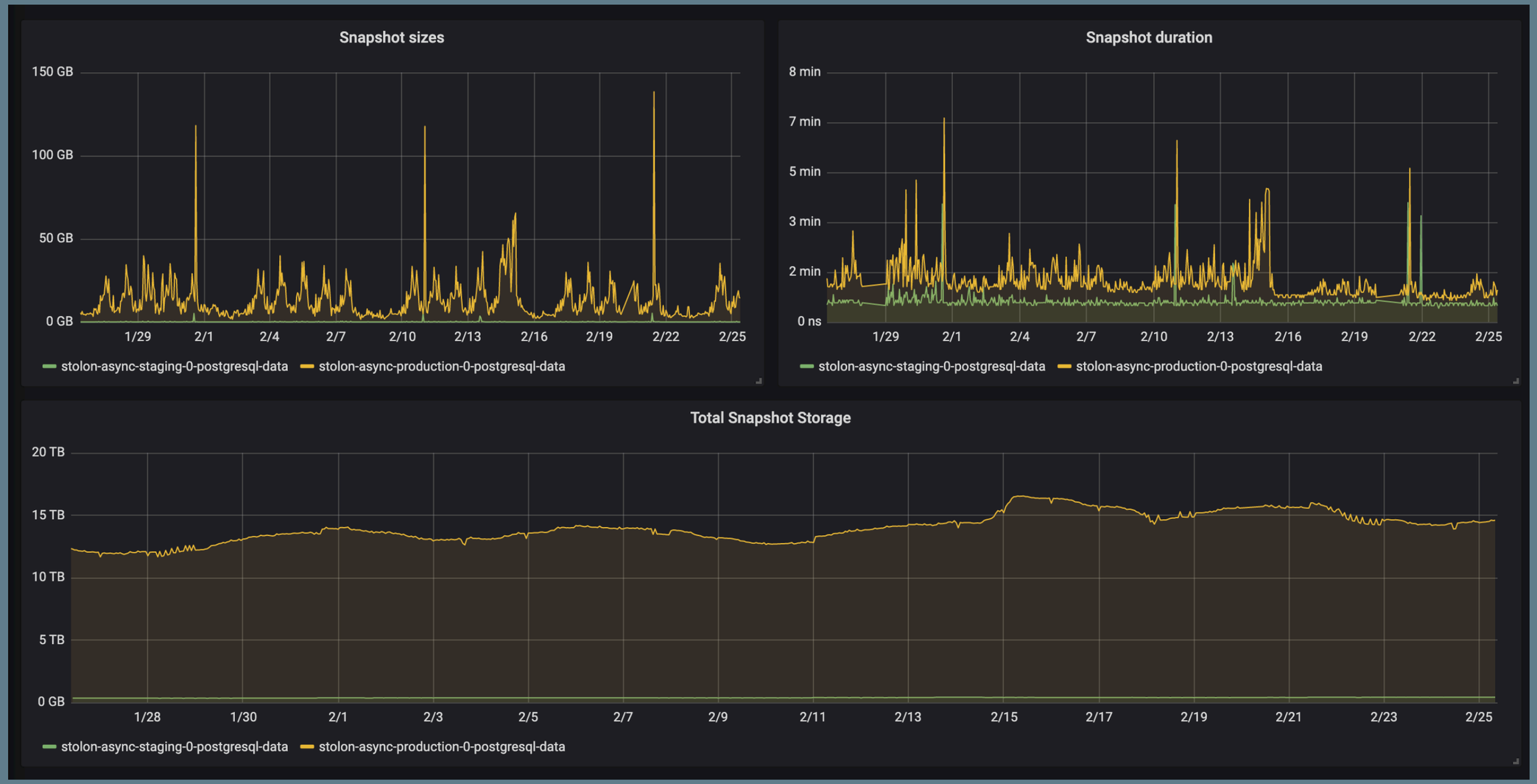
```ruby
# Chef configuration management code (Ruby DSL)
disk_snapshot_schedule("/data") do
  snapshot_frequency("*:0/15")

  # Keep all backups under 3 days old at 15m intervals, ...
  retention_windows(
    "3d": "15m",
    "1w": "1h",
    "4w": "1d",
    "1y": "1w",
    "*":  "4w",
  )
end
```

lawrence@gocardless.com

Time	component	source_disk	event	source_disk_size_gb	snapshot_size_gb
2020-02-25T08:01:13.857+00:00	SnapshotSchedule	stolon-async-production-0-postgresql-data	creating_snapshot	-	-
2020-02-25T08:01:14.801+00:00	SnapshotSchedule	stolon-async-production-0-postgresql-data	pending_creation	-	-
2020-02-25T08:01:20.129+00:00	SnapshotSchedule	stolon-async-production-0-postgresql-data	pending_creation	-	-
2020-02-25T08:01:25.450+00:00	SnapshotSchedule	stolon-async-production-0-postgresql-data	pending_creation	-	-
2020-02-25T08:01:30.788+00:00	SnapshotSchedule	stolon-async-production-0-postgresql-data	pending_creation	-	-
2020-02-25T08:01:36.054+00:00	SnapshotSchedule	stolon-async-production-0-postgresql-data	pending_creation	-	-
2020-02-25T08:01:41.307+00:00	SnapshotSchedule	stolon-async-production-0-postgresql-data	pending_creation	-	-
2020-02-25T08:01:46.640+00:00	SnapshotSchedule	stolon-async-production-0-postgresql-data	pending_creation	-	-
2020-02-25T08:01:51.952+00:00	SnapshotSchedule	stolon-async-production-0-postgresql-data	pending_creation	-	-
2020-02-25T08:01:57.285+00:00	SnapshotSchedule	stolon-async-production-0-postgresql-data	pending_creation	-	-
2020-02-25T08:02:02.498+00:00	SnapshotSchedule	stolon-async-production-0-postgresql-data	snapshot_created	3,300	16.385

lawrence@gocardless.com



lawrence@gocardless.com

Plan: Step 3, ship WAL

lawrence@gocardless.com

```
# Ship new WAL segments to GCS
archive_command = "gsutil %p gc-prd-postgresql-wal/%f"
```

```
# Google Cloud Storage -> Amazon S3
resource "google_cloudfunctions_function" "wal_to_s3" {
  name        = "wal-to-s3"
  runtime     = "go113"
  description = "Copy WAL segments to S3 off-site storage"

  event_trigger {
    event_type = "google.storage.object.finalize"
    resource   = "projects/${var.project}/buckets/${module.wal_archive.bucket}"
  }

  ...
}
```

Outcomes

lawrence@gocardless.com

Outcomes: RTO & RPO

- Recovery within 20m
- Loses <1m data
- Success?

lawrence@gocardless.com

Outcomes: ABR

- Your backups are broken
- Test them, or don't bother at all!
- ABR: Automated backup recovery...

lawrence@gocardless.com

gocardless / abr-stolon-cluster Private

Configure Sourcegraph | Watch 16 | Star 0 | Fork 0

generated from gocardless/golang-template

Code | Issues 0 | Pull requests 8 | Actions | Projects 0 | Wiki | Security | Insights | Settings

Automated backup recovery and verification for our Stolon clusters

Edit

tier-1 team-core-infrastructure archive-no Manage topics

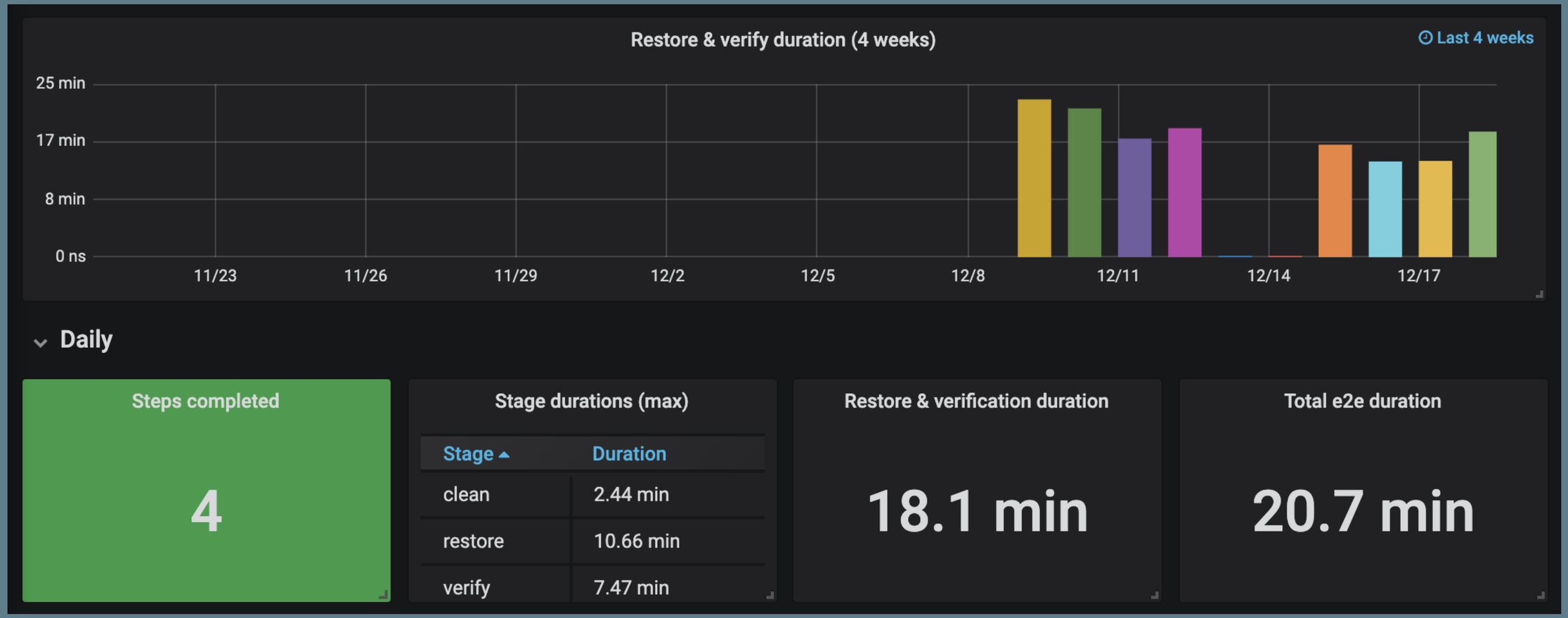
56 commits | 14 branches | 0 packages | 0 releases | 4 environments | 6 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

dyson Merge pull request #53 from gocardless/dyson-bump-anu-sha ... Latest commit f9a7c48 6 days ago

.circleci	Fix check-updated-dependencies CI job	4 months ago
.dependabot	Initial commit	4 months ago
cmd/abr-stolon-cluster	Log verification errors	2 months ago
pkg/terraform	Change each command into a stage	4 months ago

lawrence@gocardless.com



lawrence@gocardless.com

Postgres (Stolon) [docs]

GoCardless in-house database solution. We'll run a highly-available Postgres cluster on behalf of your team in the same manner as the database that powers our primary API (payments-service).

- Core-Infrastructure required for setup and maintenance
- SRE on-call provided when powering a tier-1 service
- >99.95% uptime SLO (<21m downtime per month)
- <1m failover for node failures
- Zero-downtime patch version upgrades
- Provides the latest version of Postgres
- Backups and point-in-time recovery with daily testing
- Configuration optimised for large variable usage pattern databases
- Direct integration with GoCardless observability stack
- Optional shipping of data to BigQuery
- Optional anonymised production databases available via Draupnir

Take-aways

- Establish constraints, work until you meet them
- When things are fast, you open new doors
- Correct technology choice can have big impact (5x savings)
- Work to a technical vision

lawrence@gocardless.com

- We're hiring! (all eng roles)
- More content at:
  - https://gocardless.com/blog/debugging-the-postgres-query-planner/
  - https://blog.lawrencejones.dev/building-a-postgresql-load-tester/

lawrence@gocardless.com