BRUNO SCHEUFLER
[AT]GOPHERCON
UK → 2025

INNGEST
.COM
EST...2021

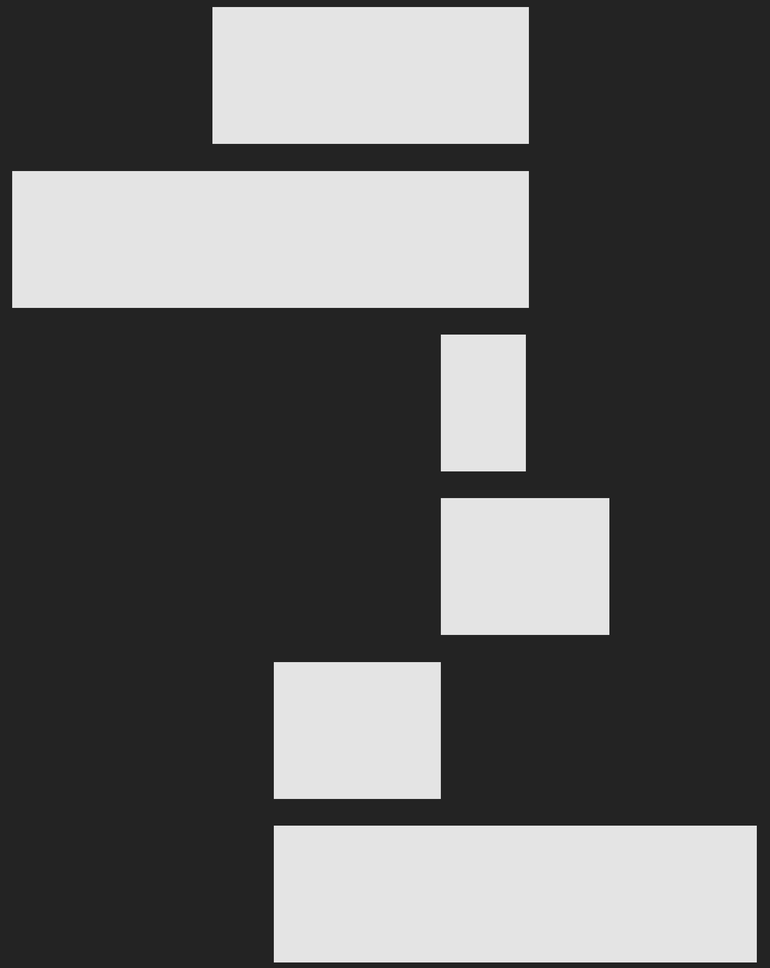
INNGEST

Welcome

↳ Please move up to the front [↑]

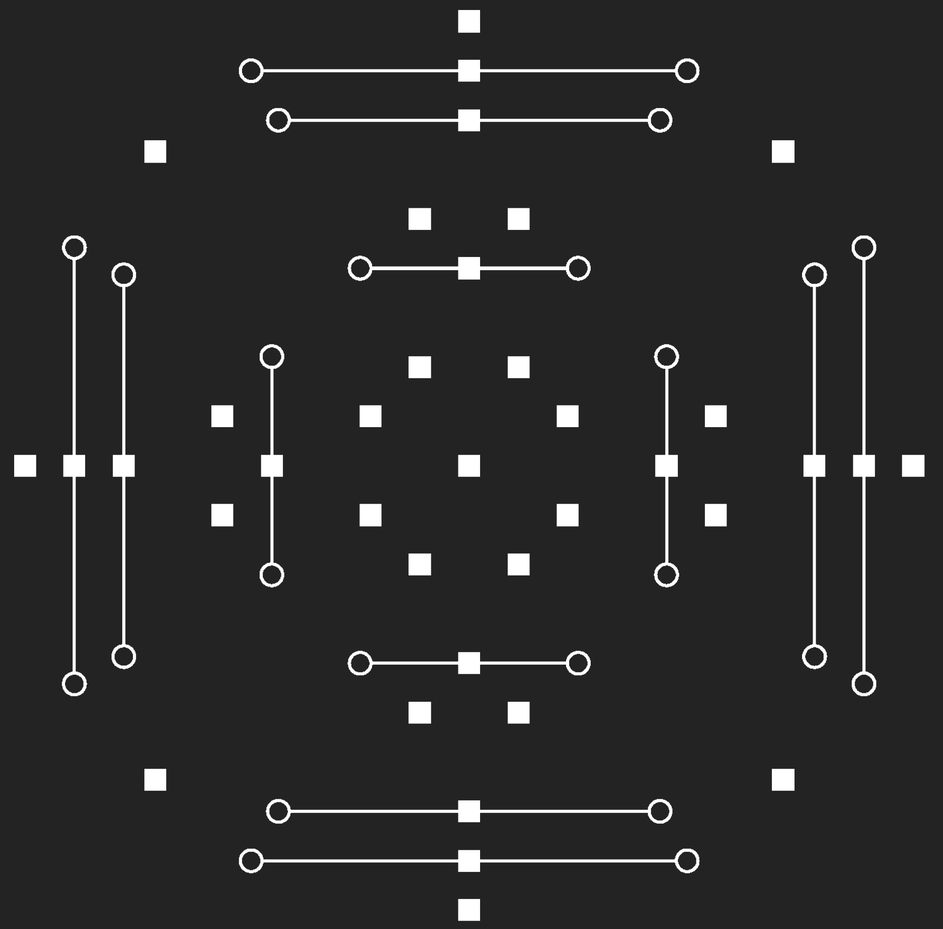
↳ We'll begin shortly...

BRUNO SCHEUFLER
[AT]GOPHERCON
UK → 2025



INNGEST
.COM
EST...2021

Building a framework for reliable data migrations in Go



■ Show of hands

.

.

.

[01]

Who has **written** **a migration** before?

■ Show of hands

.

.

.

[02]

Who has run into issues rolling out a migration?

■ Show of hands

.

.

.

[03]

Who has broken production with a migration?

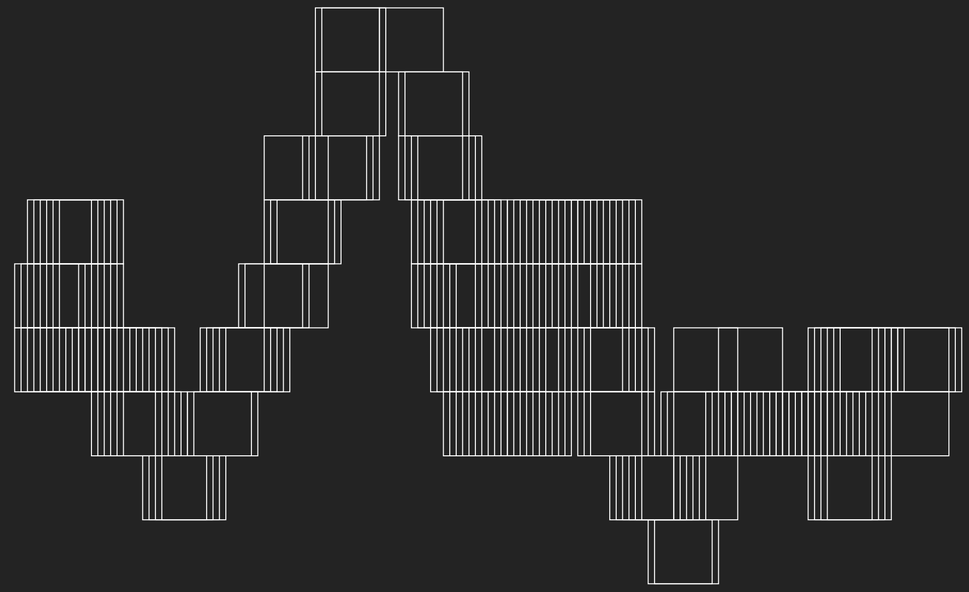
A **story** before we get started

Hello :)
I’m **Bruno**

↳ Software Engineer
↳ @Inngest

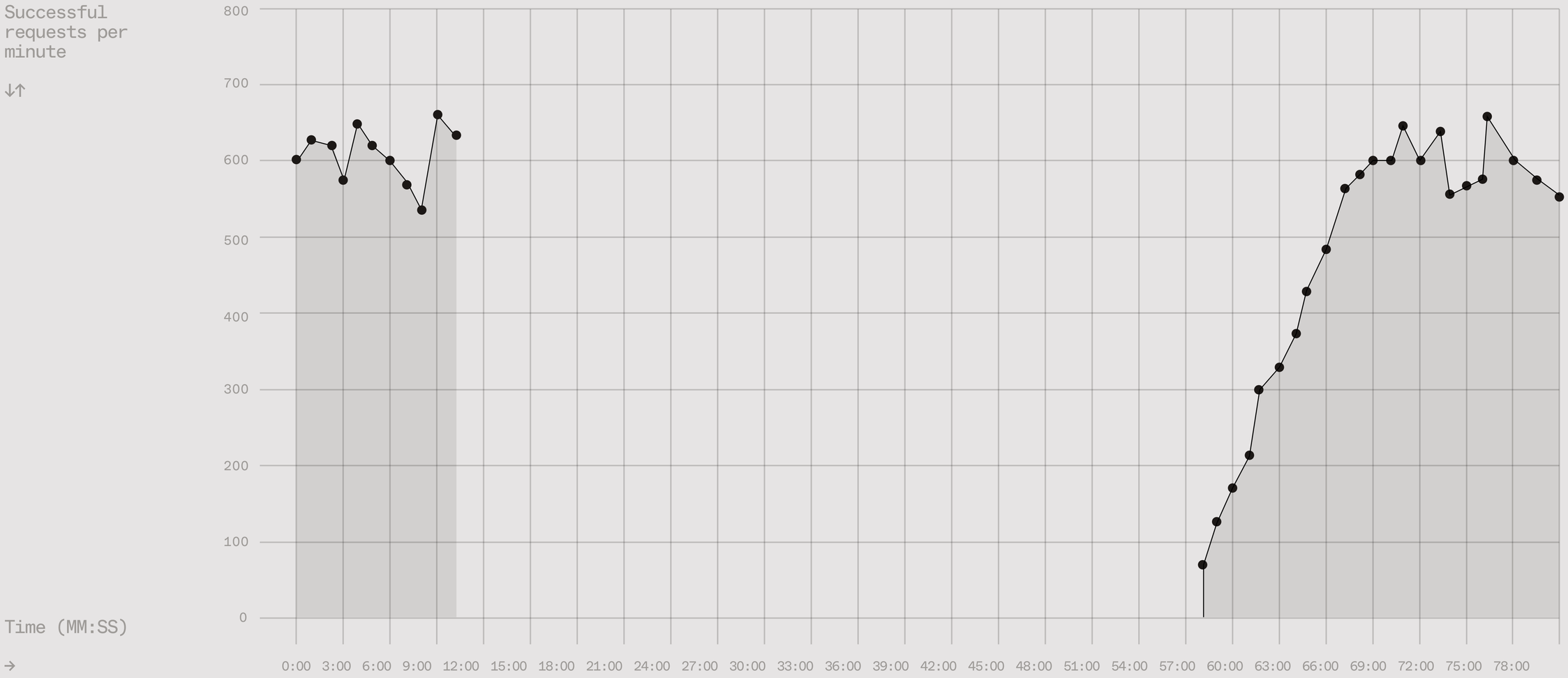
14 August/25

How not to run a migration



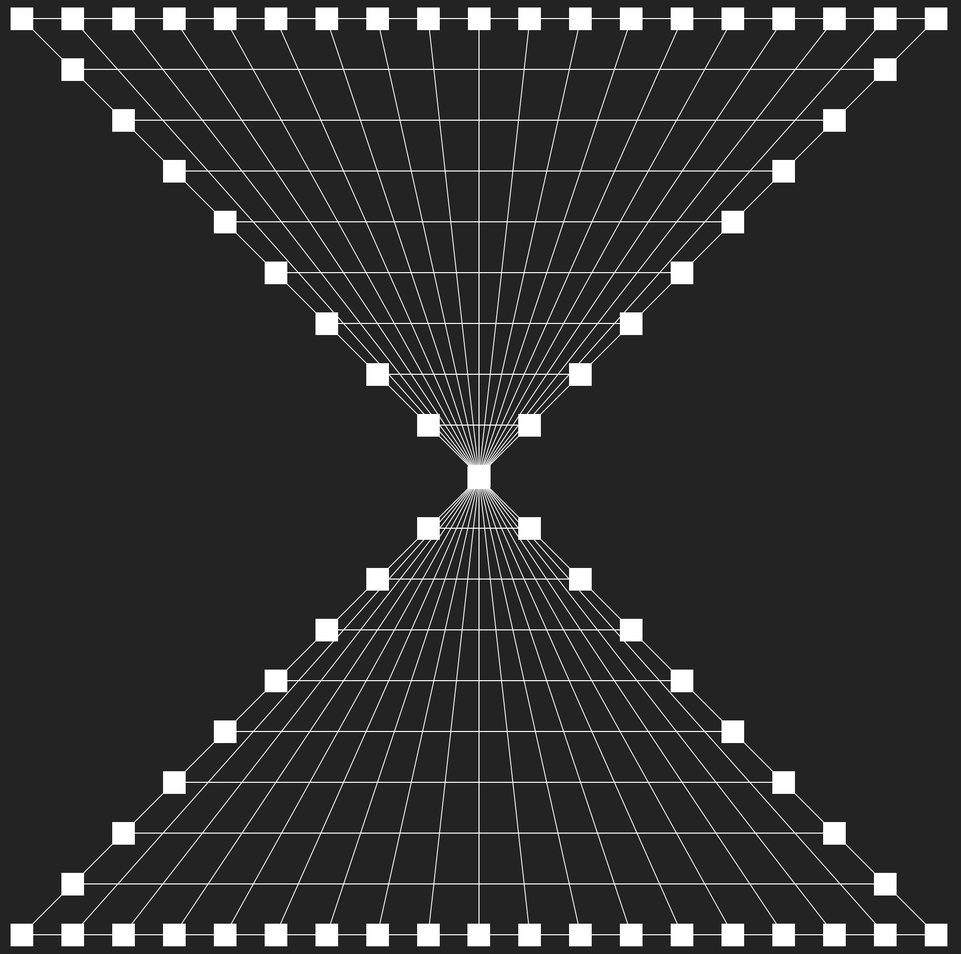
INNGEST EST...2021 / ©2025

+ SLIDES AVAILABLE → BRUNO.SH/GO25


Successful requests per minute
↓↑
800
700
600
500
400
300
200
100
0
Time (MM:SS)
→
0:00 3:00 6:00 9:00 12:00 15:00 18:00 21:00 24:00 27:00 30:00 33:00 36:00 39:00 42:00 45:00 48:00 51:00 54:00 57:00 60:00 63:00 66:00 69:00 72:00 75:00 78:00

14 August/25

Amazing migration



INNGEST EST...2021 / ©2025

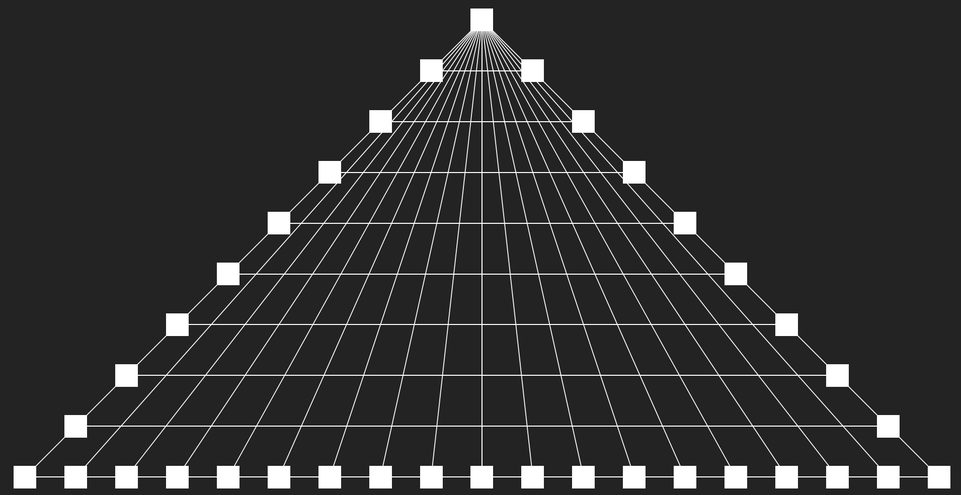
+ SLIDES AVAILABLE → BRUNO.SH/GO25


[01] Initial
Cluster—1
Primary
Accept writes, holds all data
Feature Flag
[02] Configure new primary
Cluster—2
Primary
Empty
Cluster—1
Secondary
Empty
Feature Flag
[03] Toggle flag atomically
Cluster—2
Primary
Accept writes, receive migrated data
Cluster—1
Secondary
Draining
Feature Flag
[04] Wait until migrated
Cluster—2
Primary
Accept writes, receive migrated data
Cluster—1
Secondary
Draining
Feature Flag
[05] Remove secondary
Cluster—2
Primary
Accept writes, holds all data
Feature Flag
[05] Disable flag
Cluster—2
Primary
Accept writes, holds all data
Feature Flag


Debounce
debounce operations (default)
0
Mon 7
12:00
Tue 8
12:00
Wed 9
op in sum:inngest_queue_debounce_operation{qu...
AVG
MIN
MAX
SUM
VALUE
created
deleted
migration_prepared
start
updated
debounce operations (system)
0
Mon 7
12:00
Tue 8
12:00
Wed 9
op in sum:inngest_queue_debounce_operation{qu...
AVG
MIN
MAX
SUM
VALUE
created
deleted
start
updated
Debounce partition size
0
Mon 7
06:00
12:00
18:00
Tue 8
06:00
12:00
18:00
Wed 9
06:00
12:00
Default
System
Debounce creations per shard
0
Mon 7
12:00
Tue 8
12:00
Wed 9
Default
System
Debounce starts per shard
0
Mon 7
12:00
Tue 8
12:00
Wed 9
Default
System

14 August/25

Same goal, (vastly) different outcomes...



+ SLIDES AVAILABLE → BRUNO.SH/GO25

■ Background

.

.

.

[↘]

Why should we **care**?

Why should we care?

Software always changes (migrations are inevitable)

[01]	Dealing with intense growth
[02]	Building new features
[03]	Launching new products
[04]	Sunsetting legacy software

Why should we care?

Migrations can speed your team, or slow you down

[01]	Jump to next projects faster
[02]	Sunsetting legacy software
[03]	Phase out old code, technical debt

Why should we care?

Migrations can be fun, really!

[01] Solving a puzzle

[02] Spikes your adrenaline + dopamine

[03] Mixing disciplines: Architecture, Engineering, Ops

■ Why should we care?

What’s in it for me?

☑ After this session you’ll be able to:

[01] Design migrations for production systems, small and large

[02] Set up effective observability to track migration progress

[03] Lead migrations from start to finish

Agenda

Act 1: A primer on migrations (45 min)

[01] Migrations are systems

[02] Do the work upfront

[02] Make it easy to do the right thing

Morning break: (30 min)

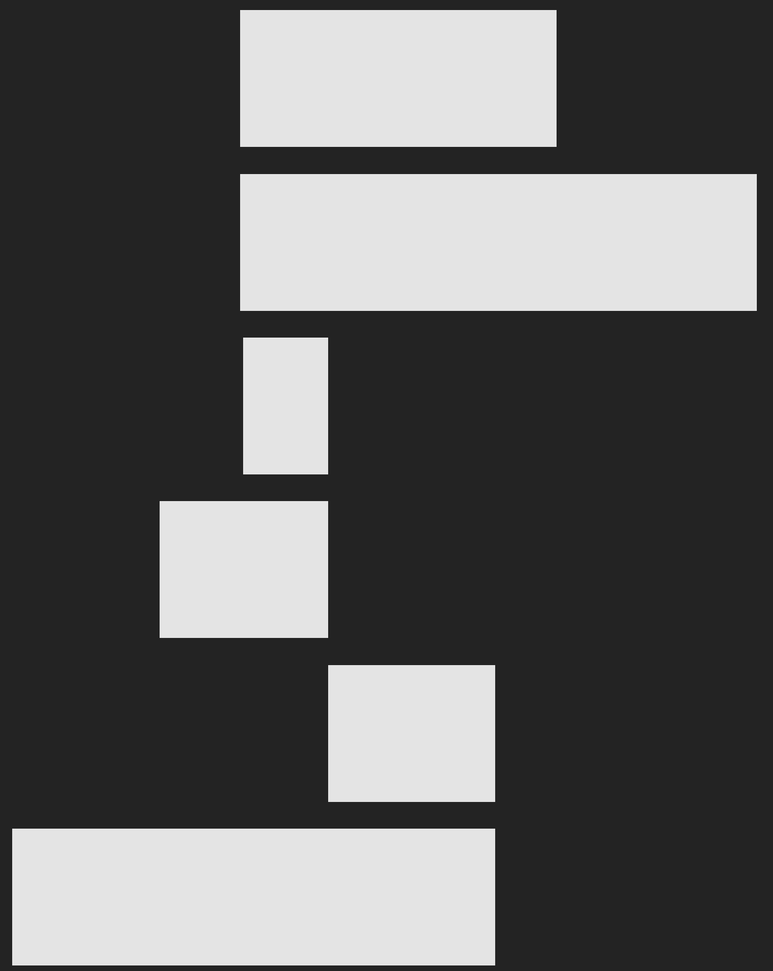
Act 2: Hands-on exercises (45 min)

[01] Introduction

[02] Migrating data stores

[03] Sharding your data

Just ask questions, anytime



ACT →

01

What’s a migration?

THE SHORT ANSWER

A well-defined **unit of change** applied to a system

■ THE LONG ANSWER

.
.
.

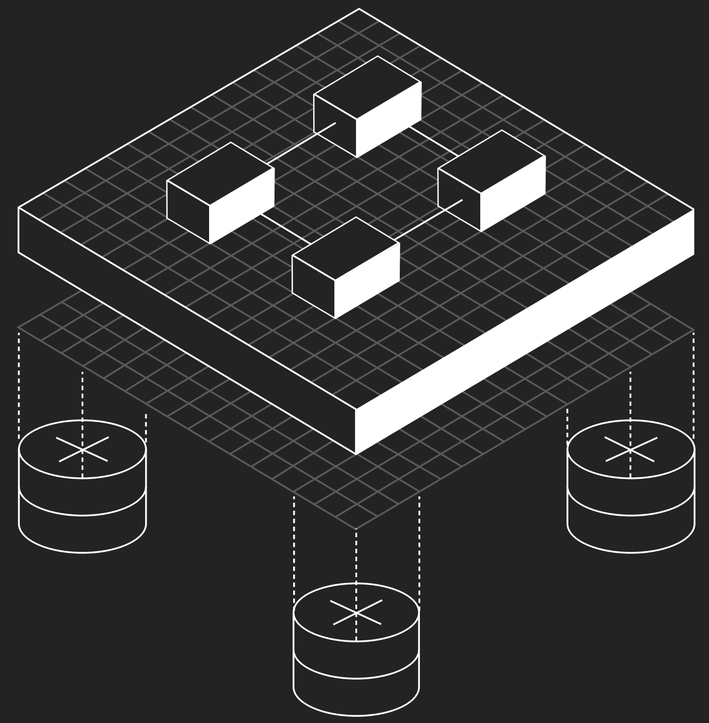
An **orchestrated workflow** made up of multiple stages with the goal of transitioning from an initial to a final state

The long answer

Workflow with multiple stages, transitions

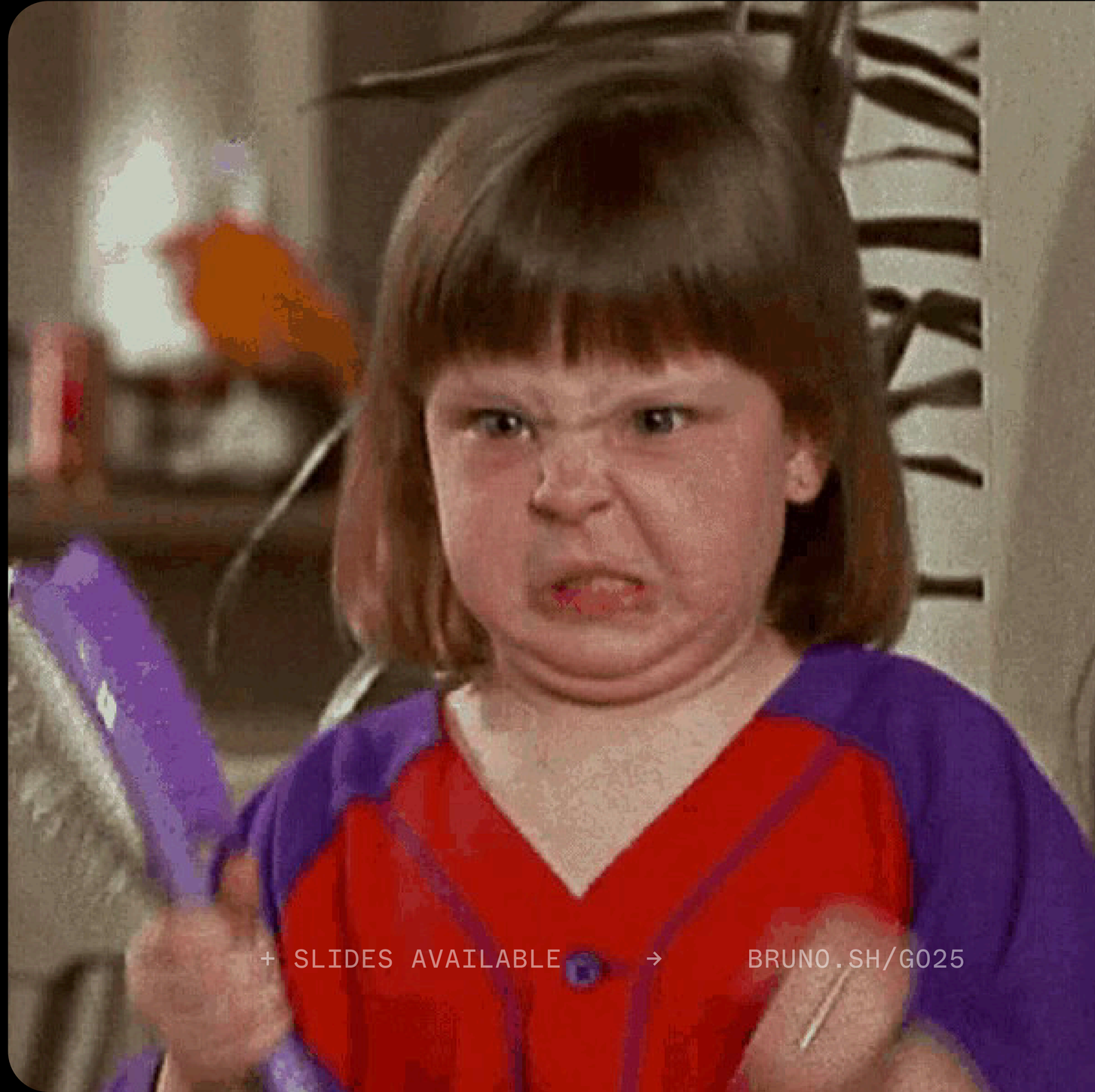
- → Initial state
- → Change read/ write source target
- → Dual read/write
- → Cutover period
- → Backfill
- → Final state

Migrations can apply to **schema, data, systems,** or a combination of both



Migrations can be **online** or **offline**

As a user, scheduled downtime and maintenance periods suck



+ SLIDES AVAILABLE → BRUNO.SH/GO25

Summary

Migrations are workflows similar to state machines.

[01] Migrations have well-defined stated & transition rules

[02] Migrations can affect schema, data & systems

[03] Migrations can be online & offline

■ THESIS

Why can’t migration be more like any **normal code rollout**?

PART 1

Migrations are systems

Three principles for migrations

[01] Never impact a running system

[02] Surface migration progress and controls

[03] Always finish migrations, quickly

+ SLIDES AVAILABLE → BRUNO.SH/GO25

Three principles for migrations

[01] Never impact a running system

[02] Surface migration progress and controls

[03] Always finish migrations, quickly

Three principles for migrations

[01] Never impact a running system

[02] Surface migration progress and controls

[03] Always finish migrations, quickly

Principles for distributed systems

[01] Never make assumptions

[02] Example: rolling out a simple producer/ consumer service

[03] Everything can fail

Principles for distributed systems

[01] Never make assumptions

[02] Example: rolling out a simple producer/ consumer service

[03] Everything can fail

Principles for distributed systems

[01] Never make assumptions

[02] Example: rolling out a simple producer/ consumer service

[03] Everything can fail

Why migrations **fail**

[01]	Unexpected state/data in production
[02]	Distributed systems
[03]	Slower than expected
[04]	Forever migrations
[05]	Metastable failure

Why migrations **fail**

[01] Unexpected state/data in production

[02] Distributed systems

[03] Slower than expected

[04] Forever migrations

[05] Metastable failure

+ SLIDES AVAILABLE → BRUNO.SH/GO25

Why migrations **fail**

[01] Unexpected state/data in production

[02] Distributed systems

[03] Slower than expected

[04] Forever migrations

[05] Metastable failure

Why migrations **fail**

[01] Unexpected state/data in production

[02] Distributed systems

[03] Slower than expected

[04] Forever migrations

[05] Metastable failure

Why migrations **fail**

[01] Unexpected state/data in production

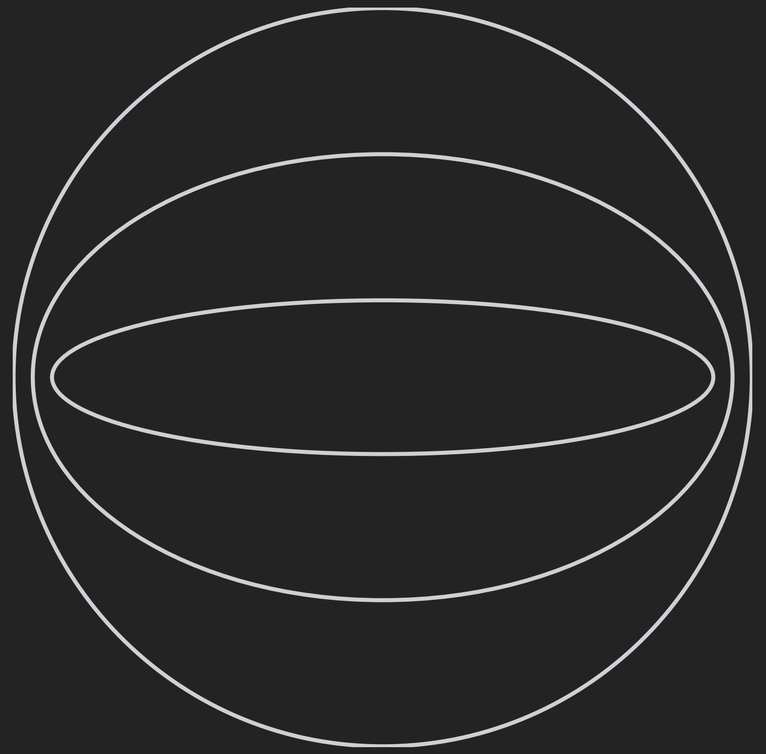
[02] Distributed systems

[03] Slower than expected

[04] Forever migrations

[05] Metastable failure

Types of migrations (a selection)



Service rollouts

- → Intro
- → Rolling releases
- → Blue / green deployments
- → Traffic splitting
- → Shadow traffic (mirroring)
- → Consistency checks

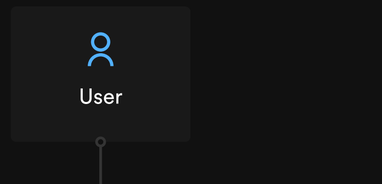
Service rollouts

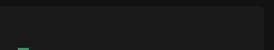
- → Intro
- → Rolling releases
- → Blue / green deployments
- → Traffic splitting
- → Shadow traffic (mirroring)
- → Consistency checks

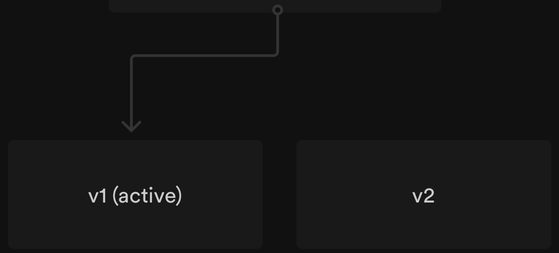
Service rollouts

- → Intro
- → Rolling releases
- → **Blue / green deployments**
- → Traffic splitting
- → Shadow traffic (mirroring)
- → Consistency checks

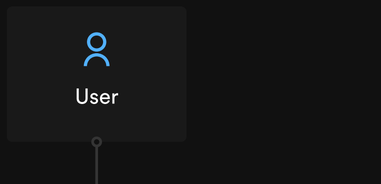
[↱] Before

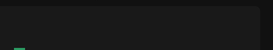


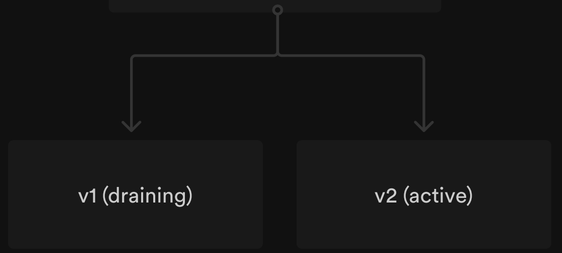




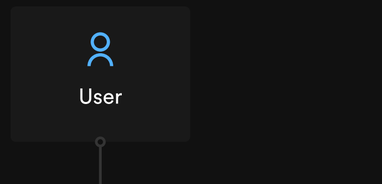
[↱] During

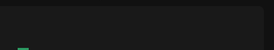


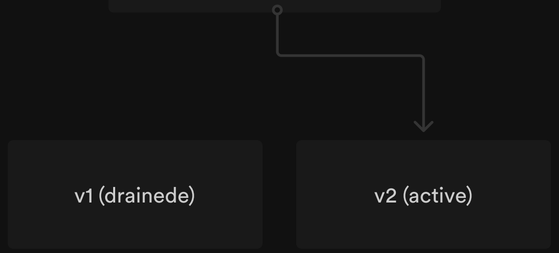




[↱] After







Service rollouts

- → Intro
- → Rolling releases
- → Blue / green deployments
- → Traffic splitting
- → Shadow traffic (mirroring)
- → Consistency checks

Service rollouts

- → Intro
- → Rolling releases
- → Blue / green deployments
- → Traffic splitting
- → **Shadow traffic (mirroring)**
- → Consistency checks

Service rollouts

- → Intro
- → Rolling releases
- → Blue / green deployments
- → Traffic splitting
- → Shadow traffic (mirroring)
- → Consistency checks

■ Types of migrations

Database Schema Migrations

- → Adding new tables
- → Adding required columns
- → Improving data access patterns with indexes, materialized views
- → Backfilling data

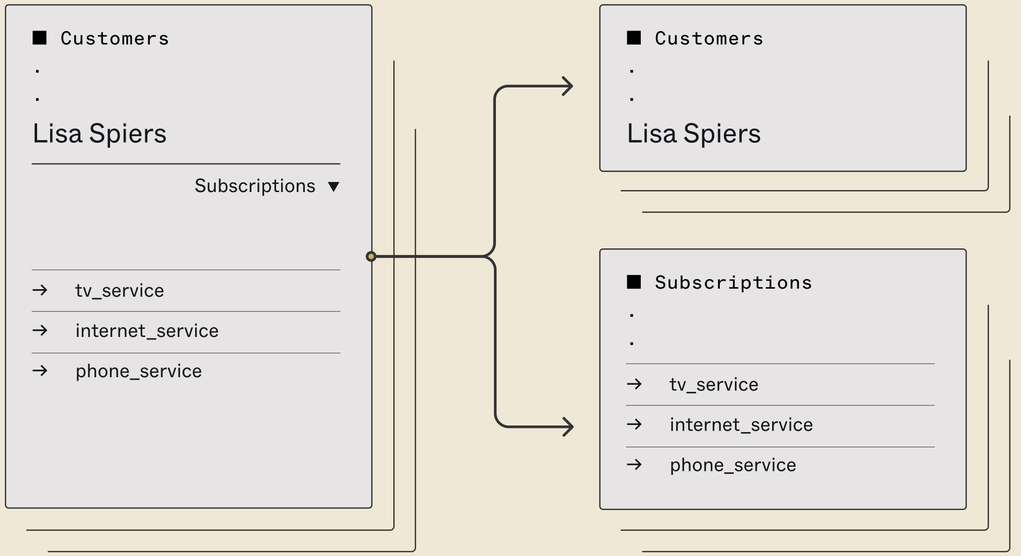
■ Database schema migrations

Adding a required column (if you can't use a default value)

- → Create nullable column
- → Update INSERTs to always include column value for new rows
- → Backfill value for existing rows
- → Mark as non-nullable

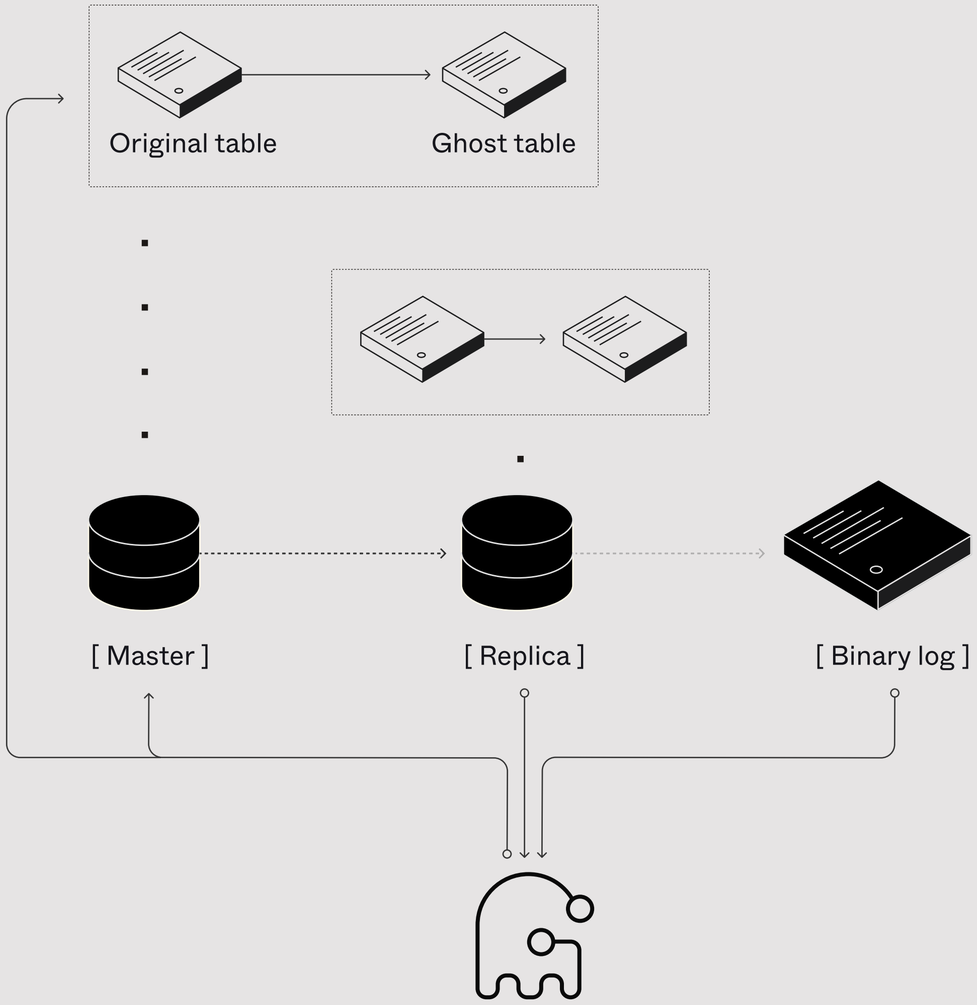
■ Example

Dual Writes (Stripe)



■ Example

Shadow tables (gh-ost) →



■ gh-ost migration:

→ Creates ghost table on migrated server

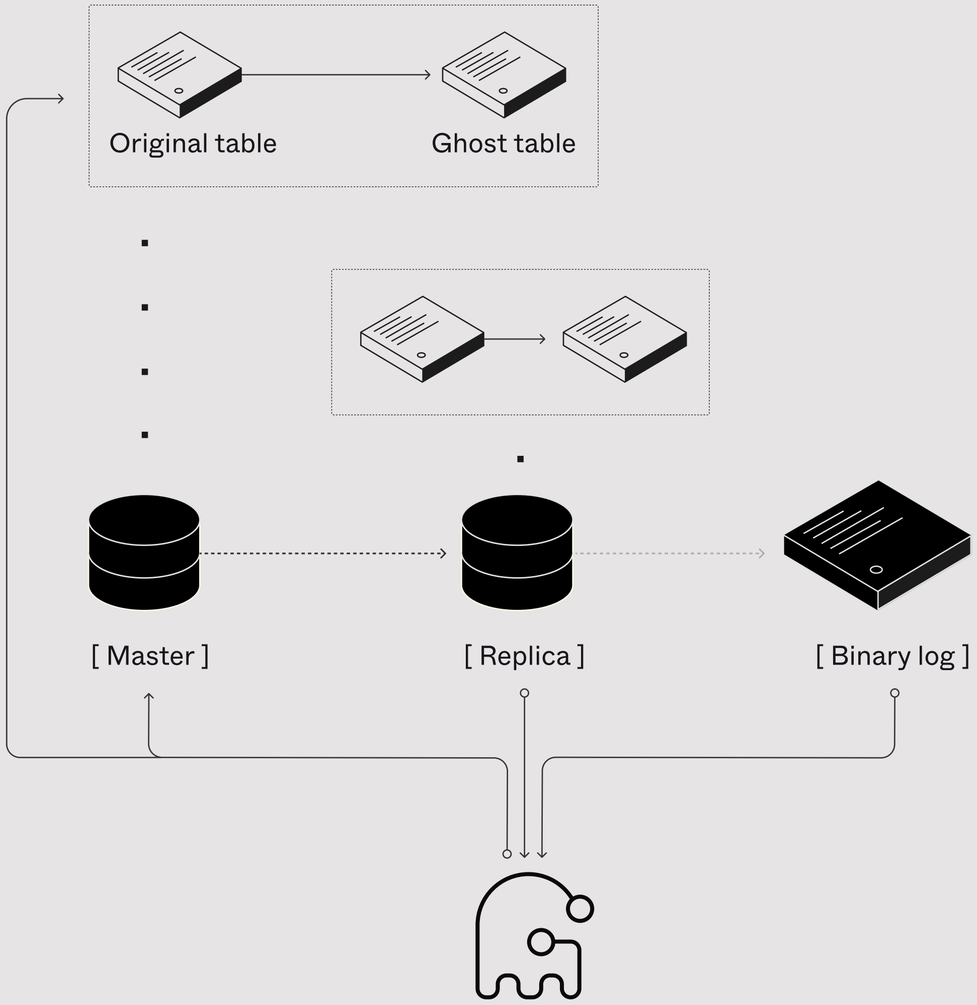
→ Alters ghost table

→ Hooks up as a MySQL replica, streams binary log events

→ Interchangeably:

[+] Applies events on ghost table

[+] Copies rows from original table onto shost table



■ Preferred setup:

- → Connects to replica
- → Inspects table structure, table dimensions on replica
- → Hooks as replica onto replica
- → Apply all changes on master
- → Writes internal & heartbeat events onto master expects them on replica

■ Types of migrations

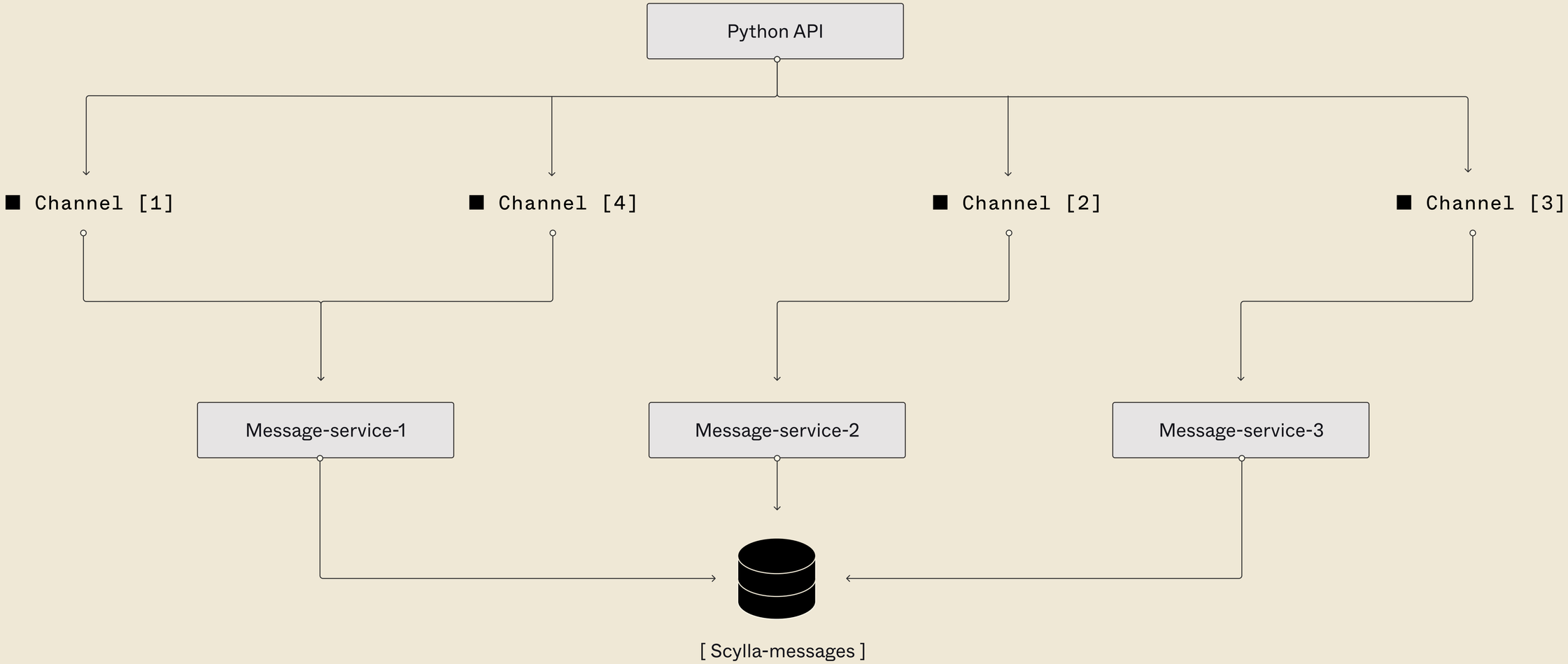
Data Migrations

- → Moving data from one data store to another
- → Sharding data across multiple physical nodes
- → Moving customer data from one region to another (compliance, performance)
- → Isolating noisy neighbors

■ Example

Discord migrated **to** ScyllaDB

- → Introduced new Rust-based data (proxy) service to coalesce requests, using consistent hash-based routing
- → Adding migration logic to the same data service
- → Dual-wrote to both data stores during migration
- → Used shadow traffic pattern to compare results between old and new clusters


Python API
Channel [1]
Channel [4]
Channel [2]
Channel [3]
Message-service-1
Message-service-2
Message-service-3
[Scylla-messages]

Recap

- What are migrations
- Principles for migrations+distributed systems
- Why migrations fail
- Types of migrations

THESIS

Why don’t we **treat migrations the same way we plan regular systems?**

Do the work upfront

- → Creating a migration plan
- → Never leave your migrations unattended
- → Always communicate
- → Make migrations a priority

■ Do the work upfront
.
.
.
[01]

“Time spent upfront **is time saved during an incident**”

■ Do the work upfront

.

.

.

[02]

Creating a migration plan

- → Which systems are involved
- → What’s the order of operations throughout the migration?
- → Can the system be migrated gradually? What factor do you decide on?
- → Can the migration be rolled back? How?

■ Do the work upfront

.

.

.

[03]

Never leave your migrations unattended

→ Set and communicate a clear timeline (ahead of time)

→ Always complete (or roll back) migrations

→ Avoid “forever migrations” at all costs

■ Do the work upfront

.

.

.

[04]

Always communicate

→ Ensure all stakeholders are briefed in advance

→ Share knowledge and tools to observe systems during migration

→ Reduce everyone’s stress levels!

■ Do the work upfront

.

.

.

[05]

Make migrations a priortity (For the team leads among us)

- → Allocate time to plan, test, and talk about migrations
- → Support engineers over the entire migration lifecycle
- → Do not: cut timelines, force individuals to bear the responsibility

■ Do the work upfront

.

.

.

[06]

Recap

→ Every minute you spend planning will be paid back tenfold during execution

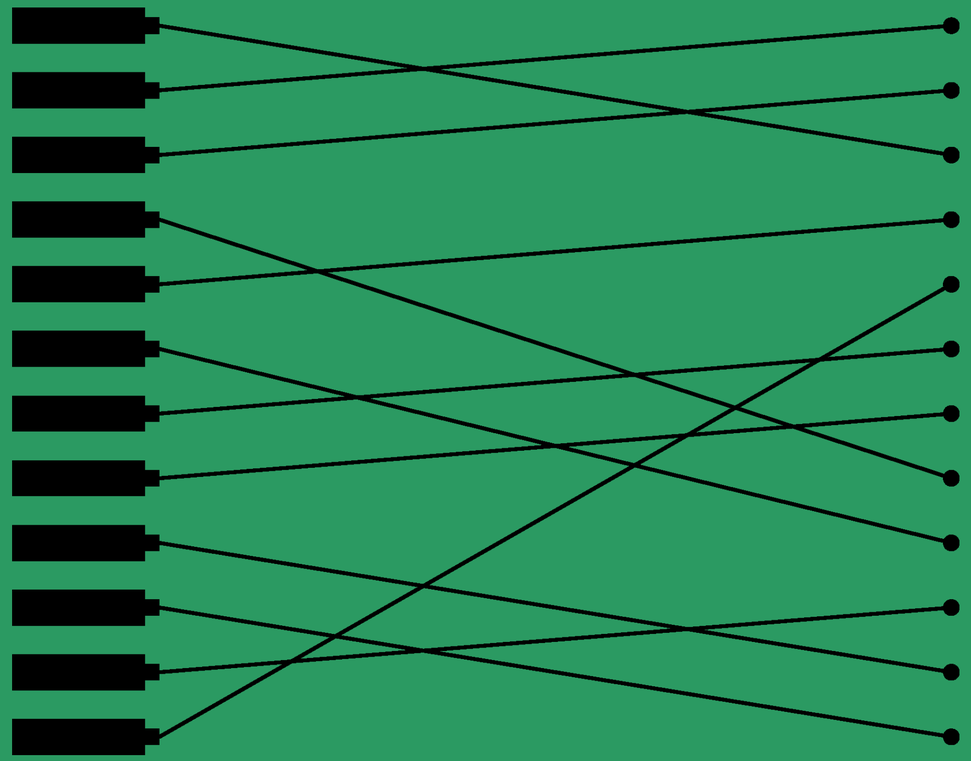
→ Prepare a migration plan

→ Set clear timelines, communicate to keep everyone in the loop

→ Always finish migrations

Act 1 / Part 3

Make it easy (to do the right thing)



+ SLIDES AVAILABLE → BRUNO.SH/GO25

■ Make it easy

.

.

.

[01]

Design principles (revisited)

- → Keep the scope as focused as possible
- → Find the smallest-possible unit to migrate
- → Use transactions, atomic operations

■ Make it easy

.

.

.

[02]

Make your migrations visible

- → Set up dashboards to follow along
- → Collect fine-grained metrics to track progress
- → Create alerts in case of failure

■ Make it easy

.

.

.

[03]

Most migrations don’t work on the first try

- → Test your migrations in production-like environments befor rolling out
- → Aim for gradual migrations, even if it’s complex to implement
- → Design migrations to be rolled back

■ Make it easy

.

.

.

[04]

Feature **flags** are your best friend

→ Roll out migrations across a subset of customers in production

01 Start with internal accounts

02 Enroll free users

03 Wrap up with individual enterprise customers

■ Make it easy

.

.

.

[05]

Share knowledge early

→ Every engineer should be able to take over

→ Every customer-facing role should know the current status

■ Make it easy

.

.

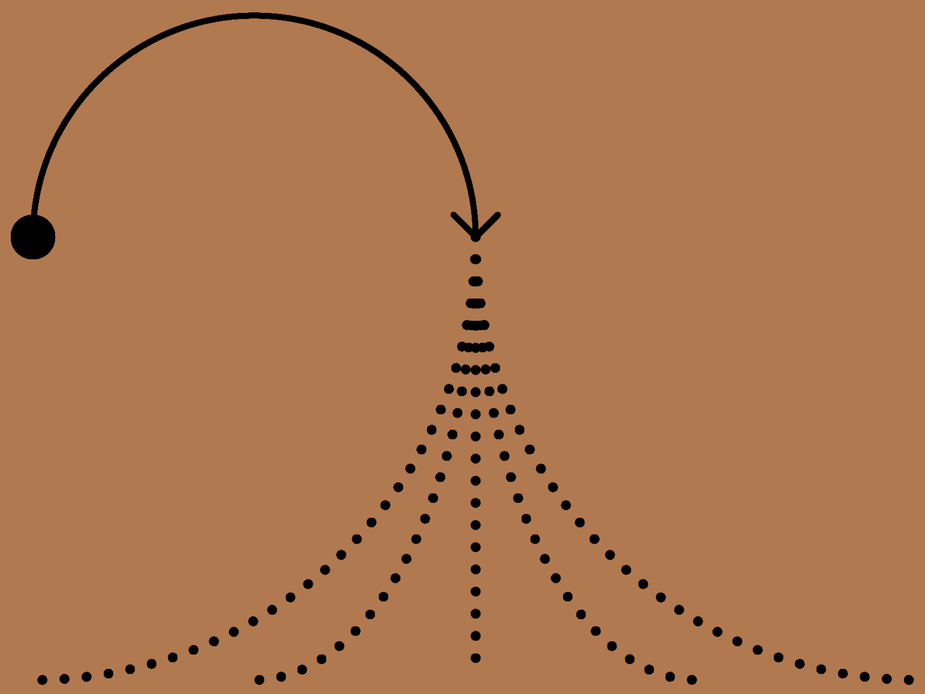
.

[06]

Recap

- → Break your migrations down to the smallest units
- → Track activity using metrics and visualize migration progress with dashboards
- → Reduce blast radius with feature flags
- → Make sure you can roll back

Wrapping up Act 1



+ SLIDES AVAILABLE → BRUNO.SH/GO25

Act 1

Recap

→ Treat migrations as systems

→ Do the work upfront

→ Make it easy to do the right thing

Before we break
(What to expect next)

Questions?

Enjoy the break,

see you back @11:30

■ Hands-on exercises

.

.

.

[↘]

Pilots, start your engines.

01 - Check out the example repository

→ bruno.sh/go25

02 - Clone the hands-on exercise code:

$ git clone github.com/brunoscheufler/gopherconuk25

03 - Start the app

→ $ go run . --cli --gen

ACT →

02

■ Show of hands

.

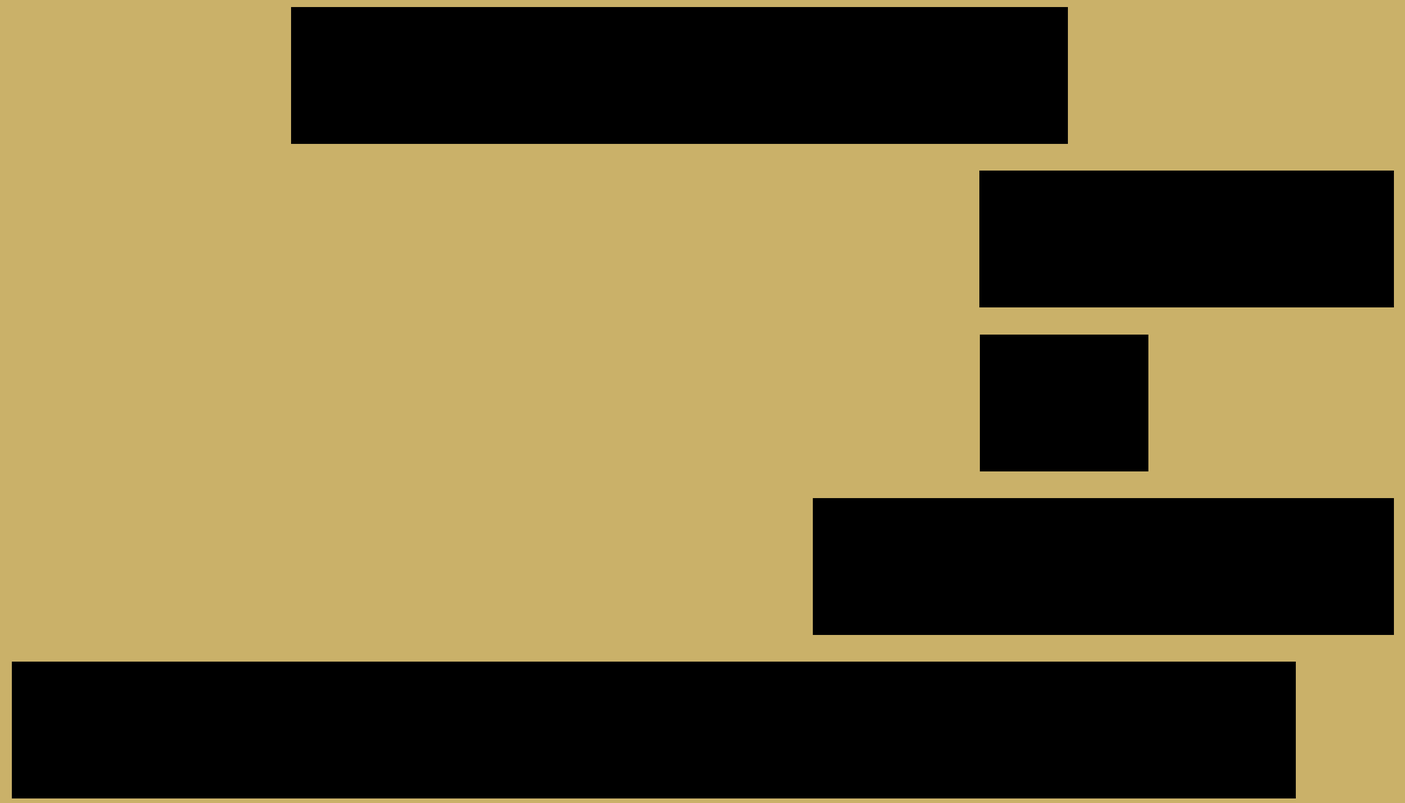
.

.

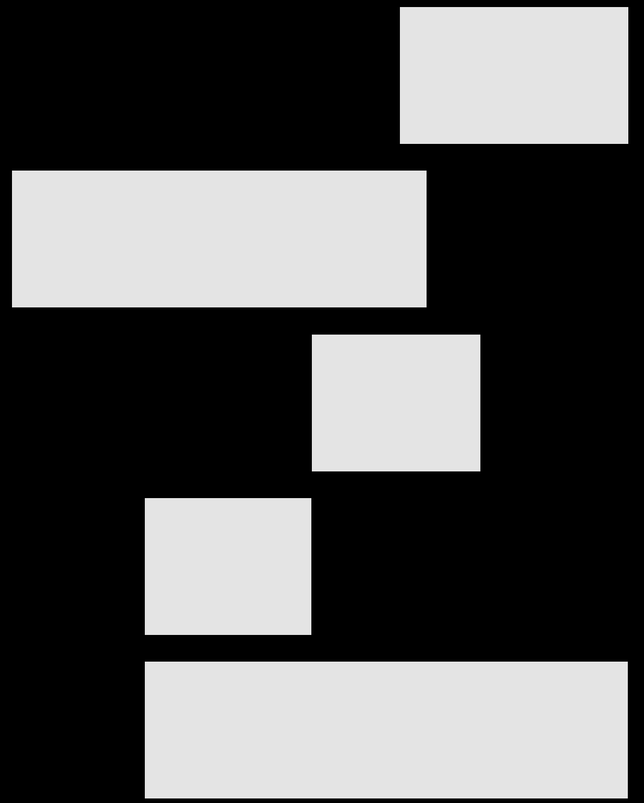
[01]

How much time have you **spent** writing Go?

Setting up



Please help each other, work together



Our example app:
Meet Notely

The codebase:
Meet the CLI

The codebase:
Load gen

The codebase:
The proxy

Moving to a **new** data store

EXERCISE: [01]

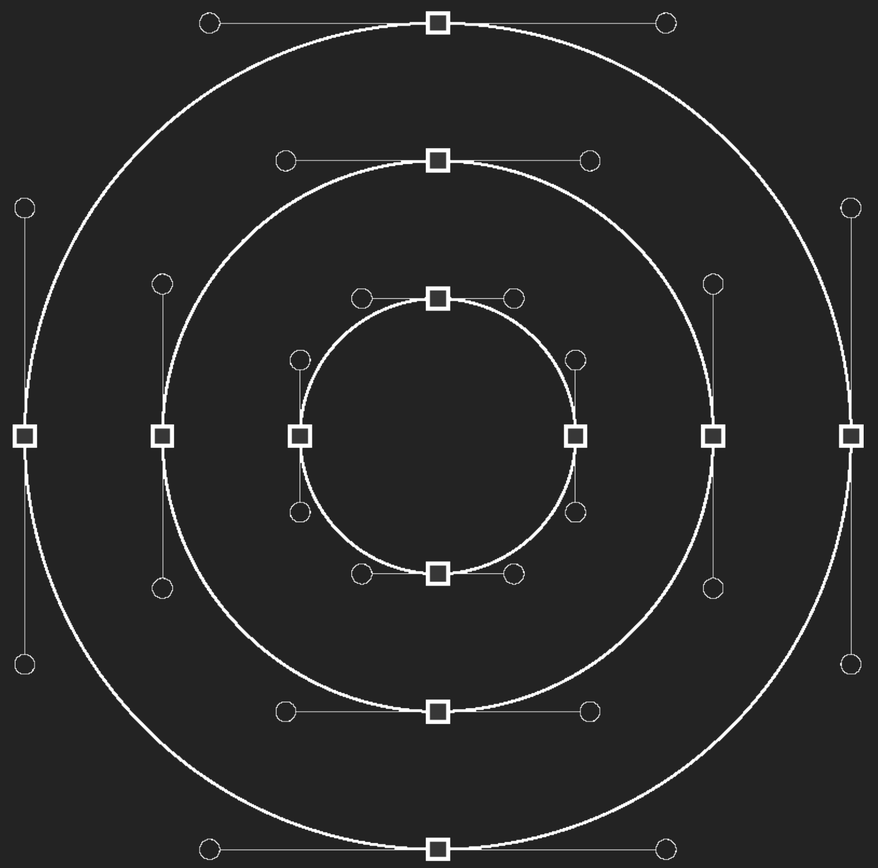
Game plan

EXERCISE: [01]

Adding **sharding**

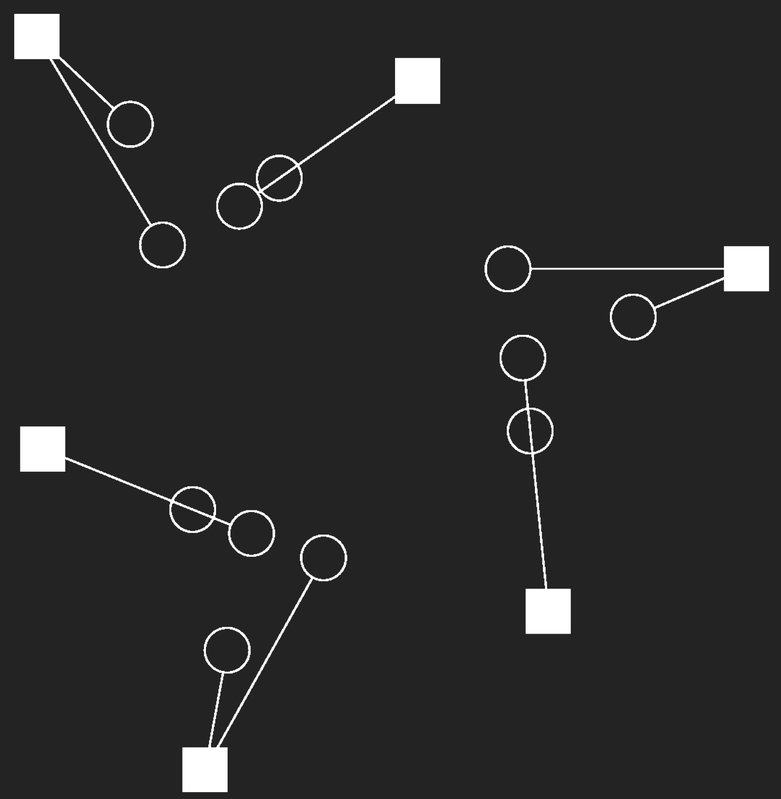
EXERCISE: [02]

Wrapping up...



+ SLIDES AVAILABLE → BRUNO.SH/GO25

What we learned today



■ What we learned today

.

.

.

[01]

Migrations are **everywhere** + necessary

■ What we learned today

.

.

.

[02]

Migrations can be **stressful**, but with the right preparation, **they don’t have to be**

■ What we learned today

.

.

.

[03]

Can borrow from systems: atomic operations, observability, feature flags, traffic splitting, etc.

■ What we learned today

.

.

.

[04]

Can plan migrations like systems: documentation/runbooks

■ What we learned today

.

.

.

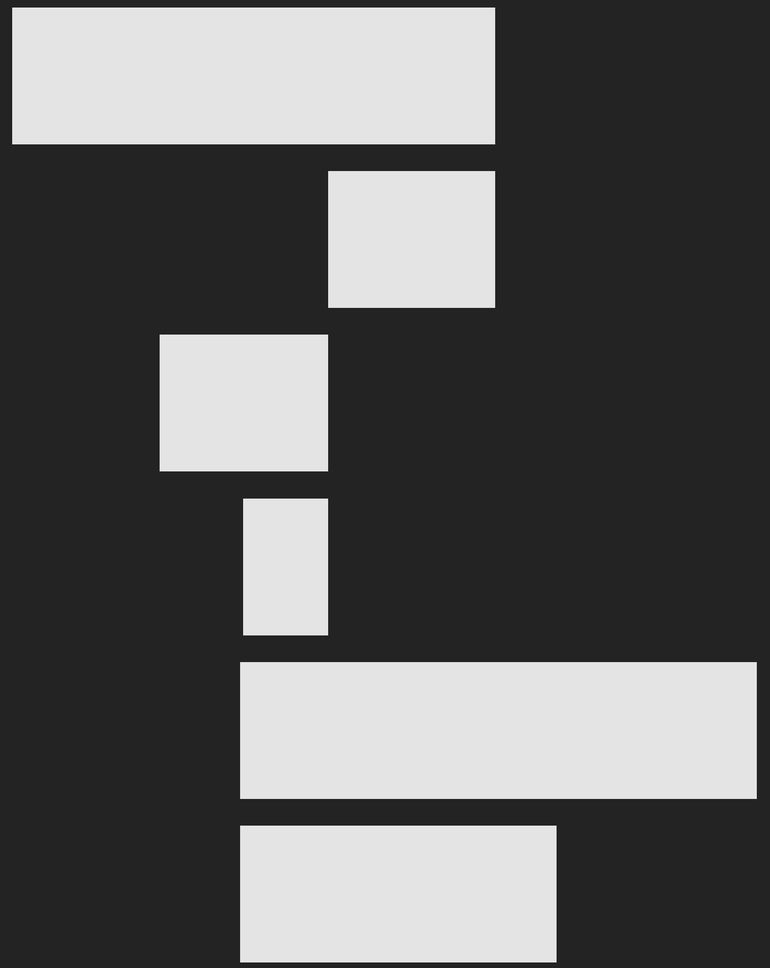
[05]

To ship faster, write better migrations

Thank you :)

↳ Slides + code available
↳ bruno.sh/go25

BRUNO SCHEUFLER
[AT]GOPHERCON
UK → 2025



INNGEST
.COM
EST...2021

Questions?

It takes a village <3

Thanks to all GopherCon UK organizers!

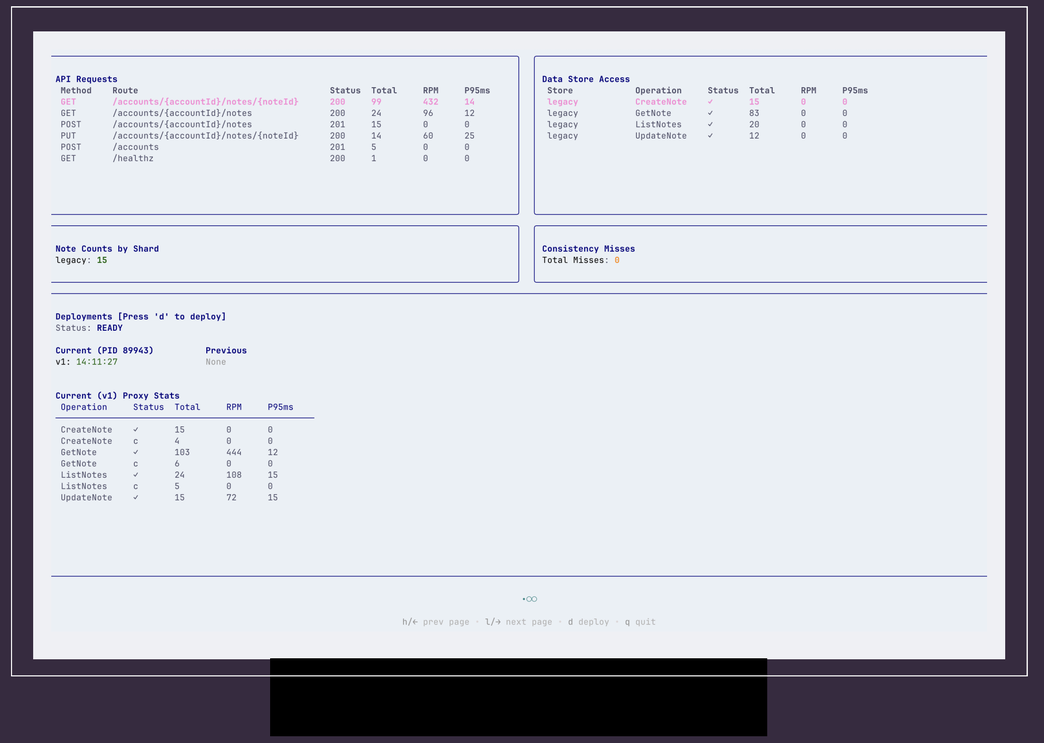
Design: Sophia Umansky
↳ Brand Design / Inngest

Thanks to the amazing team at Inngest for doing everything to make this talk come to life.

+ SLIDES AVAILABLE → BRUNO.SH/GO25

BRUNO SCHEUFLER
[AT]GOPHERCON
UK → 2025
SOFTWARE ENGINEER
@INNGEST

INNGEST



INNGEST
.COM
EST...2021