Lightning-Fast Database Tests

Using pgtestdb

Why Database Test Speed Matters

- Database tests are usually the slowest tests
- Fast tests lowers barrier to creating new tests
- Reduces CI/CD pipeline bottlenecks
- Faster feedback loop accelerates development

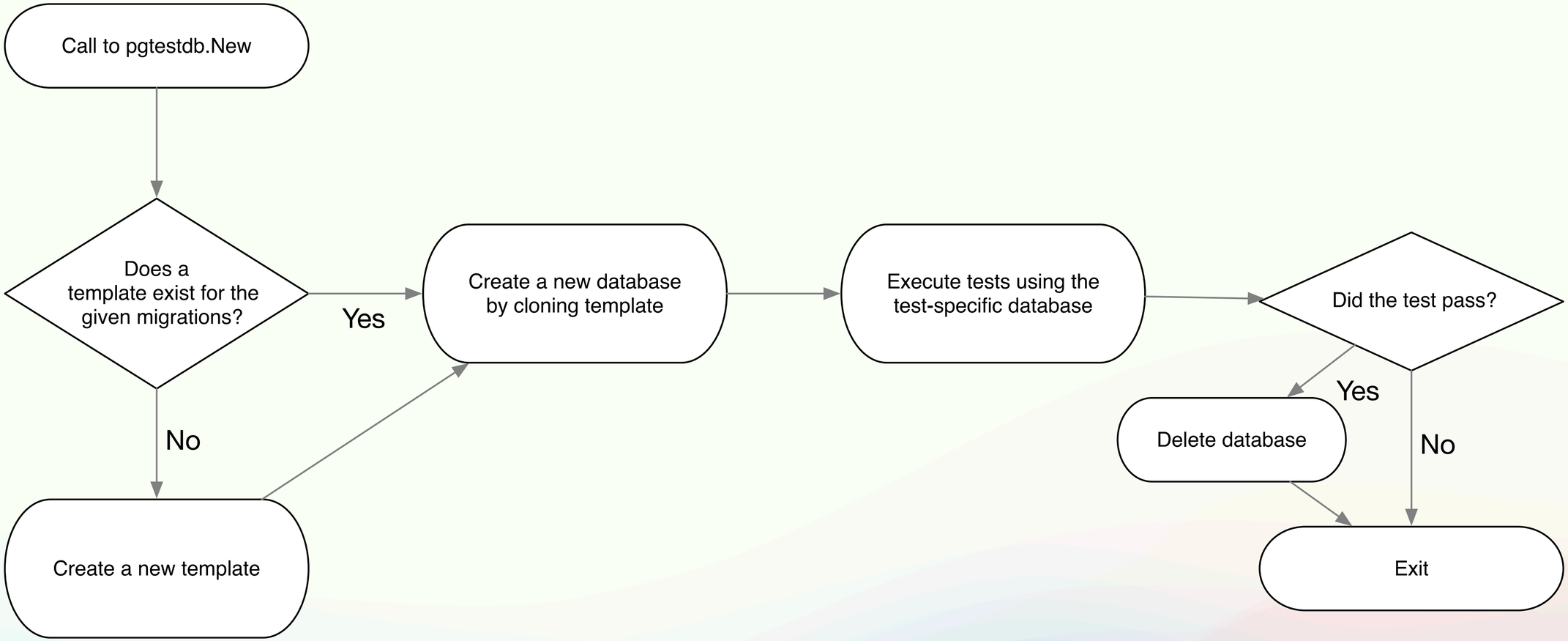
pgtestdb

- Creates ephemeral Postgres databases quickly
- Uses template databases
- Supports parallel testing
- Deletes test databases on successful tests
- Keeps them on failure
- Each test runs in an isolated database
- Other packages (dockertest, testcontainers) are slow

Some quick numbers

- A template (1000 migrations) takes ~500ms to create
- A new database takes ~10ms to create

How It Works


Call to pgtestdb.New
Does a template exist for the given migrations?
Yes
No
Create a new database by cloning template
Create a new template
Execute tests using the test-specific database
Did the test pass?
Yes
No
Delete database
Exit

Example Usage

```
conf := pgtestdb.Config{
    Database:   "postgres",
    DriverName: "pgx",
    User:       "postgres",
    Password:   “password",
    Host:       "localhost",
    Port:       port,
    Options:    "sslmode=disable",
}

//go:embed migrations/*.sql
var exampleFS embed.FS

migrator := golangmigrator.New(
    "migrations",
    golangmigrator.WithFS(exampleFS),
)
```

Example Usage

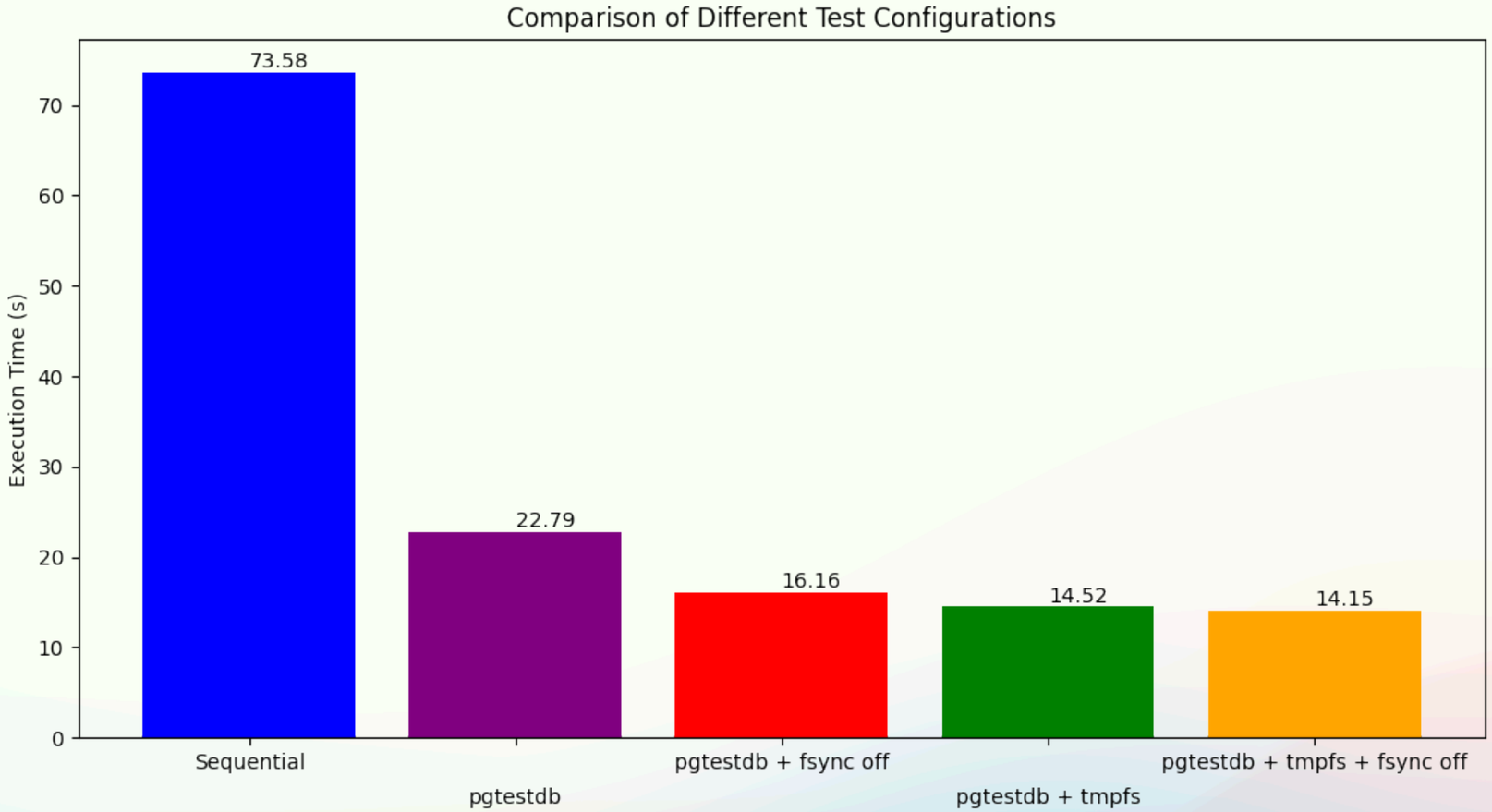
```
db := pgtestdb.New(t, conf, migrator)

var message string
err := db.QueryRow("select 'hello world'").Scan(&message)
require.NoError(t, err)
assert.Equal(t, "hello world", message)
```

Benchmark

- Test suite that inserts and lists 1000 rows
- 5 different configurations
  - Sequentially (not using pgtestdb)
  - In parallel using pgtestdb
  - In parallel using pgtestdb + fsync=off
  - In parallel using pgtestdb + tmpfs
  - In parallel using pgtestdb + fsync=off + tmpfs

Benchmark

Best case scenario is ~5x times faster



Quick recap

- You can safely run your tests in parallel
- They’re fast
- All your database tests run in isolation

Thanks for watching

Link to code, slides, pgtestdb and more.