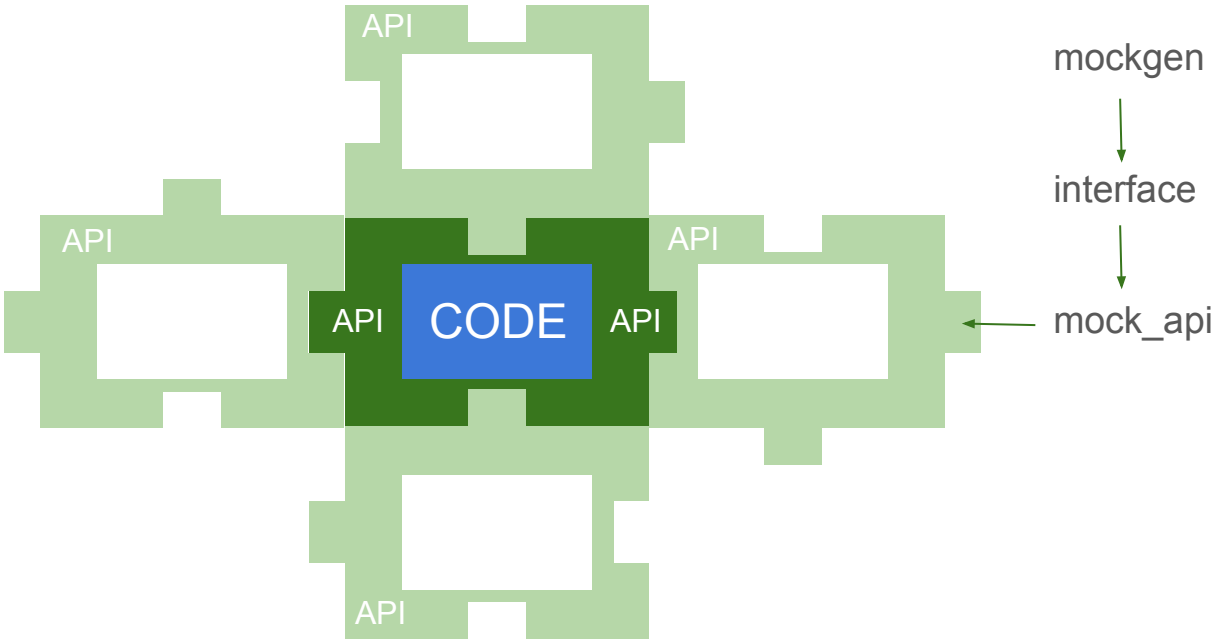
Fake it to Make it!

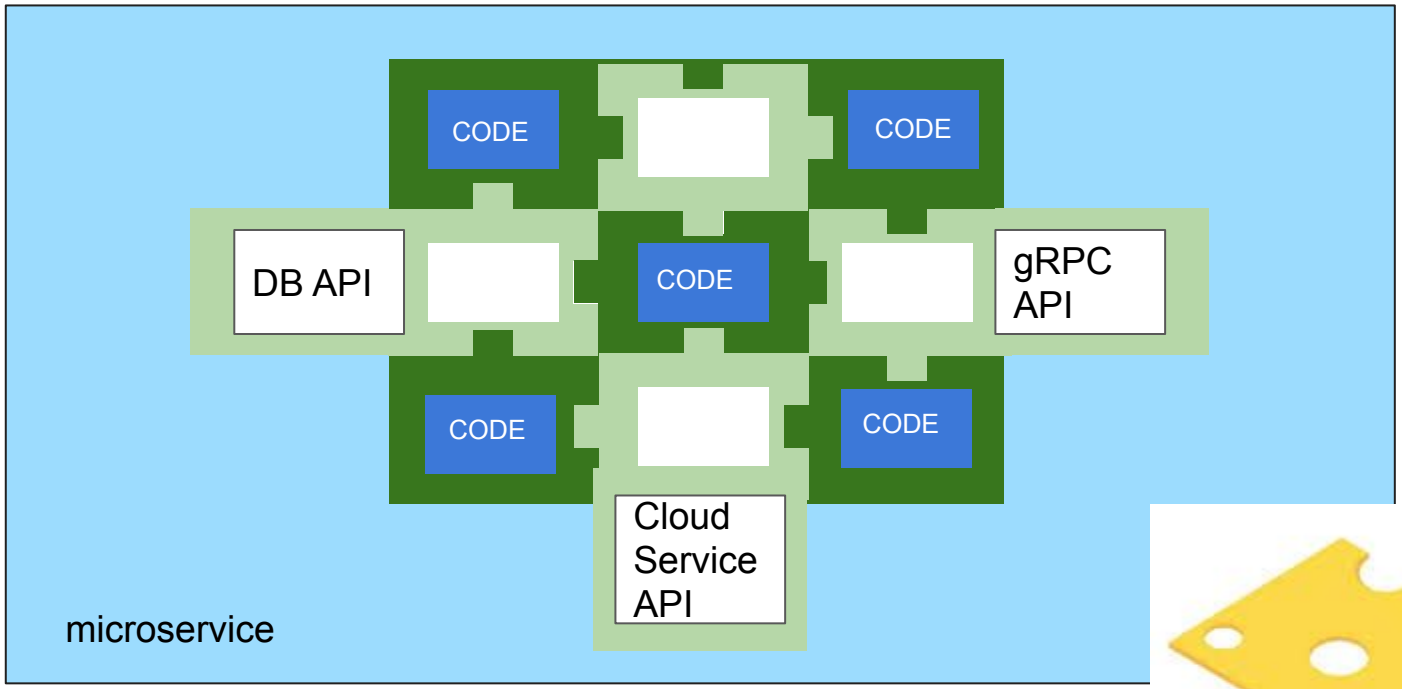
TDD of gRPC Microservices

Ed Crewe - EDB

1. THE ISSUES: Unit tests - in memory (no environment)



Unit tests - memory / disk (no environment)

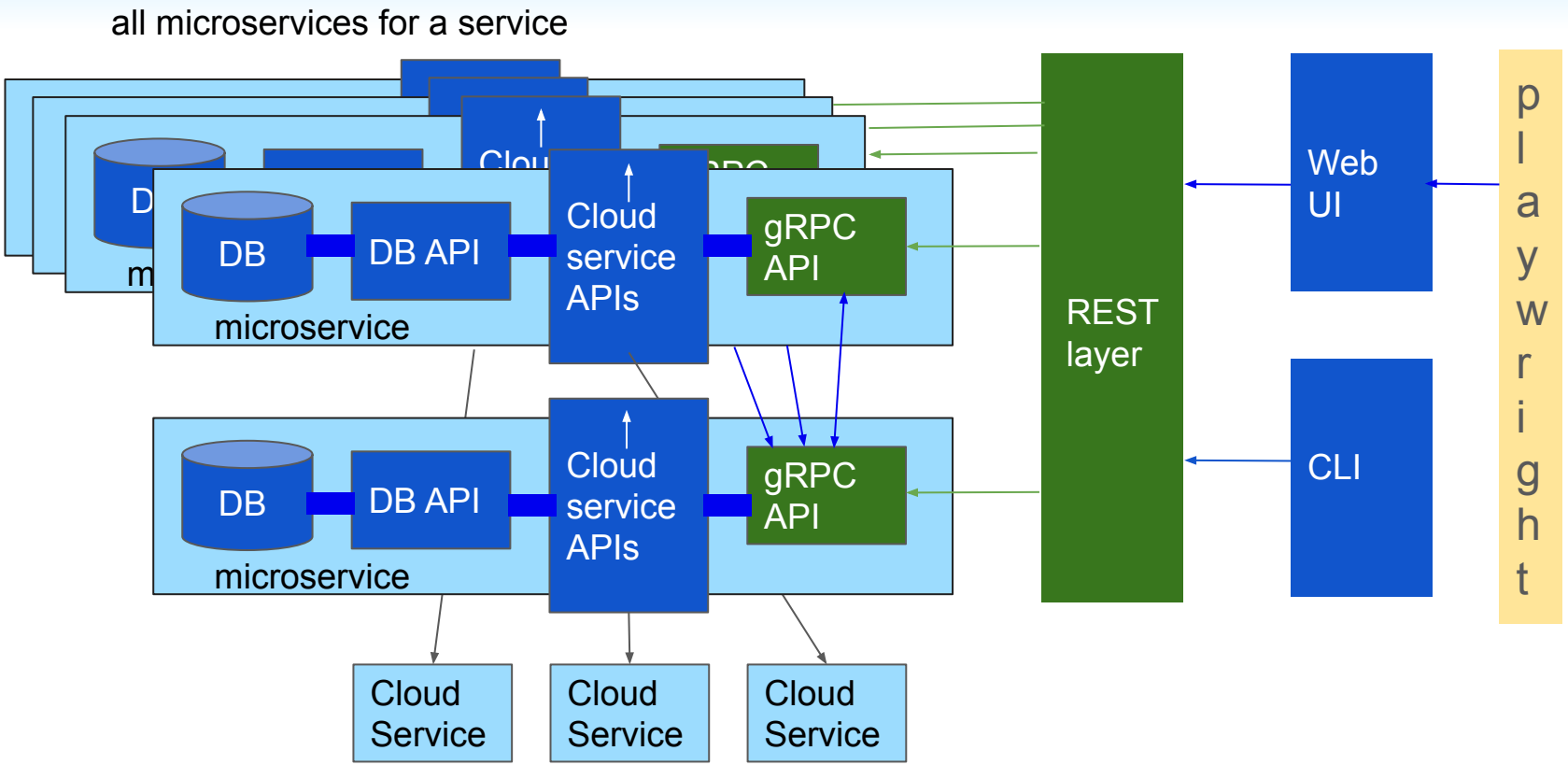


Unit tests - issues wrt. Test-driven Development (TDD)

1. Tests depend more on API details than functional reality
2. Cry Wolf Effect when code is rewritten
3. Unreliable for TDD
4. Written after the functional code
5. Encourages creation of replica tests purely for coverage

… but even so, they are still required.

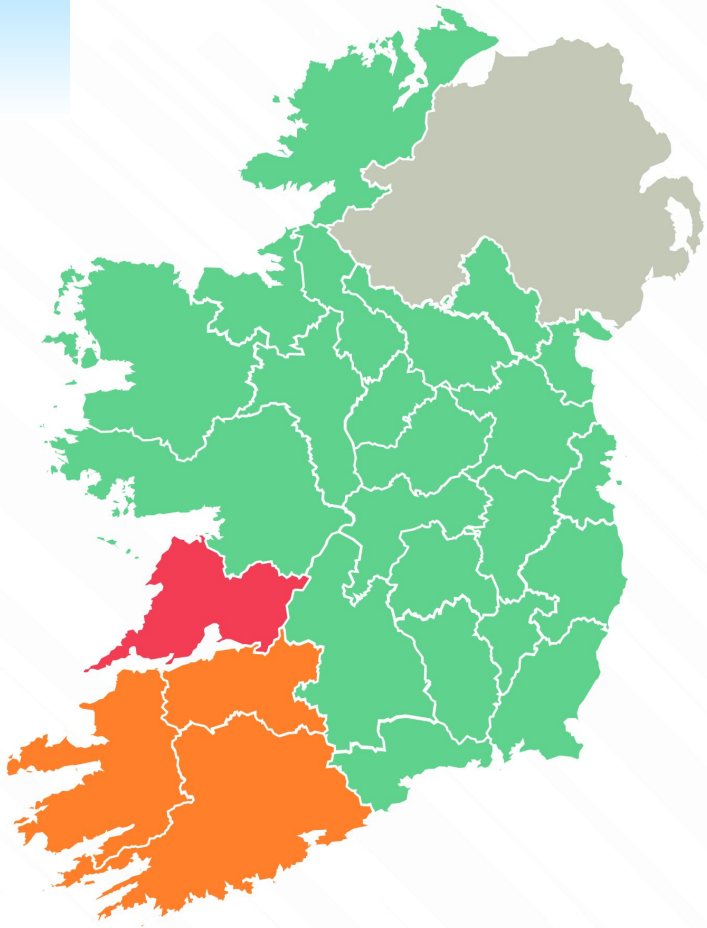
E2E tests - full stack, ie kubernetes (prod-like test environment)



EDB Postgres AI

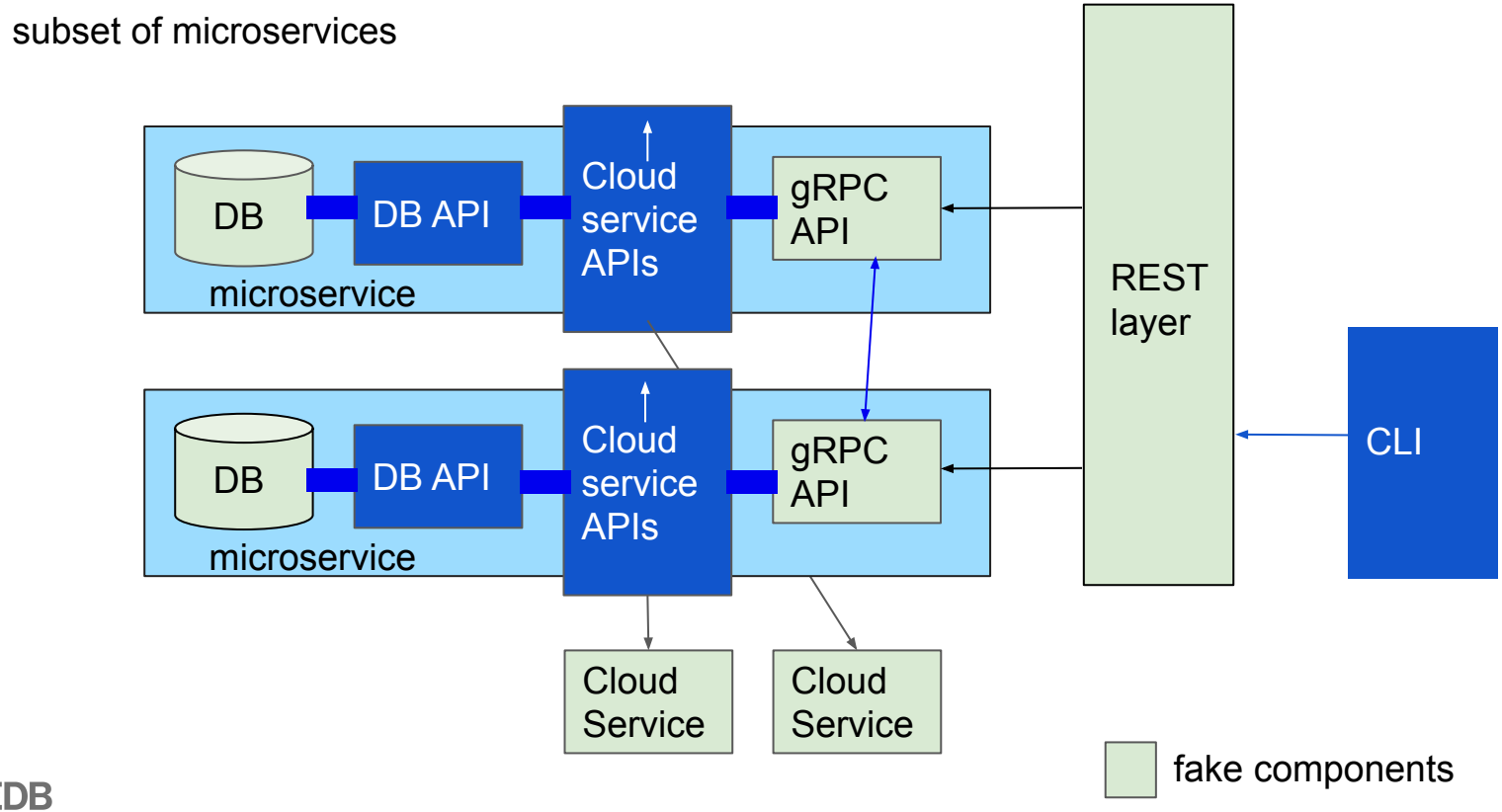
E2E tests - Sustainability

Ireland is a popular location for global CSPs European cloud data centres. They consume a quarter of its metered electricity.



https://www.vecteezy.com/

2. SOLUTIONS: Functional tests - memory / disk (fake env)



Mocks, dummys, stubs, spies & fakes

Real world Dependency

Dummy objects are unused but just fulfil the API

Mocks simulate objects with test hard coded responses

Spies are partial Mocks that patch / inspect real objects

Stubs provide canned answers to calls

Fakes working implementations, but not fit for production

- **canned service** = fixture responses for requests, may include recorder eg cloud.google.com/go/rpcreplay, many rest ones
- **partial service** = the functional test framework
- **service emulators** = local deploy. Eg. dbs, pubsub or kind for k8s

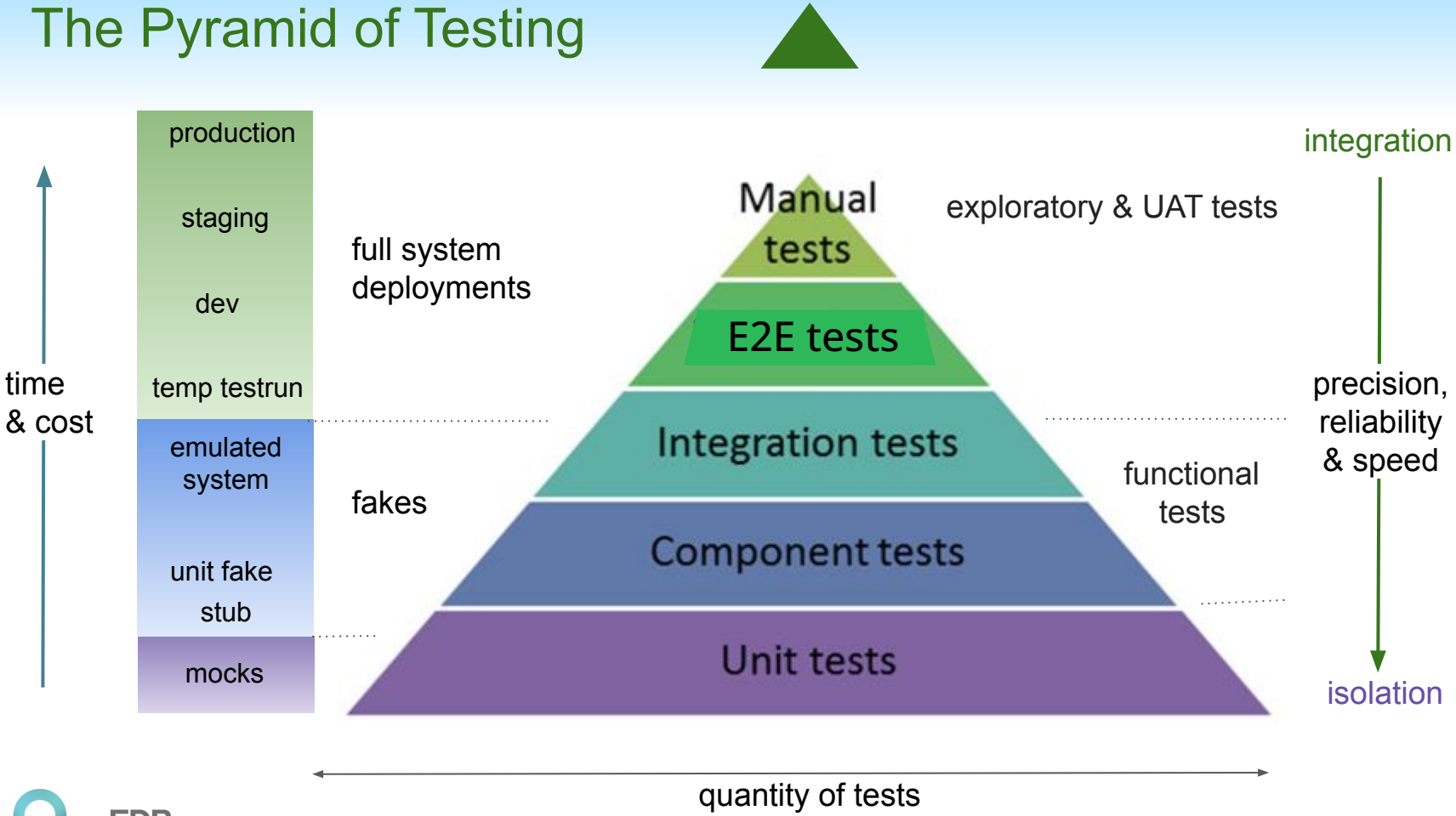
EDB Postgres AI

Emulators & Fake libraries

Examples of a generic cloud lib, a cloud provider service and storage emulators

Tool	Emulator	Fake library (Go examples)
Kubernetes	Kind or minikube	k8s.io/client-go/**kubernetes/fake**
GCloud emulators	> gcloud beta emulators, eg pubub	cloud.google.com/go/**pubsub/pstest/fake**.go
AWS localstack	> AWS emulators, eg S3	github.com/aws/aws-sdk-go-v2/service/s3
Azure NoSQL store	SQLExpress Azure storage emulator	https://pkg.go.dev/github.com/displague/crossplane/pkg/clients/**azure/storage/fake** *
Postgresql RDBMS	SQLite, Embedded Postgres	https://github.com/pashagolub/**pgxmock**

The Pyramid of Testing

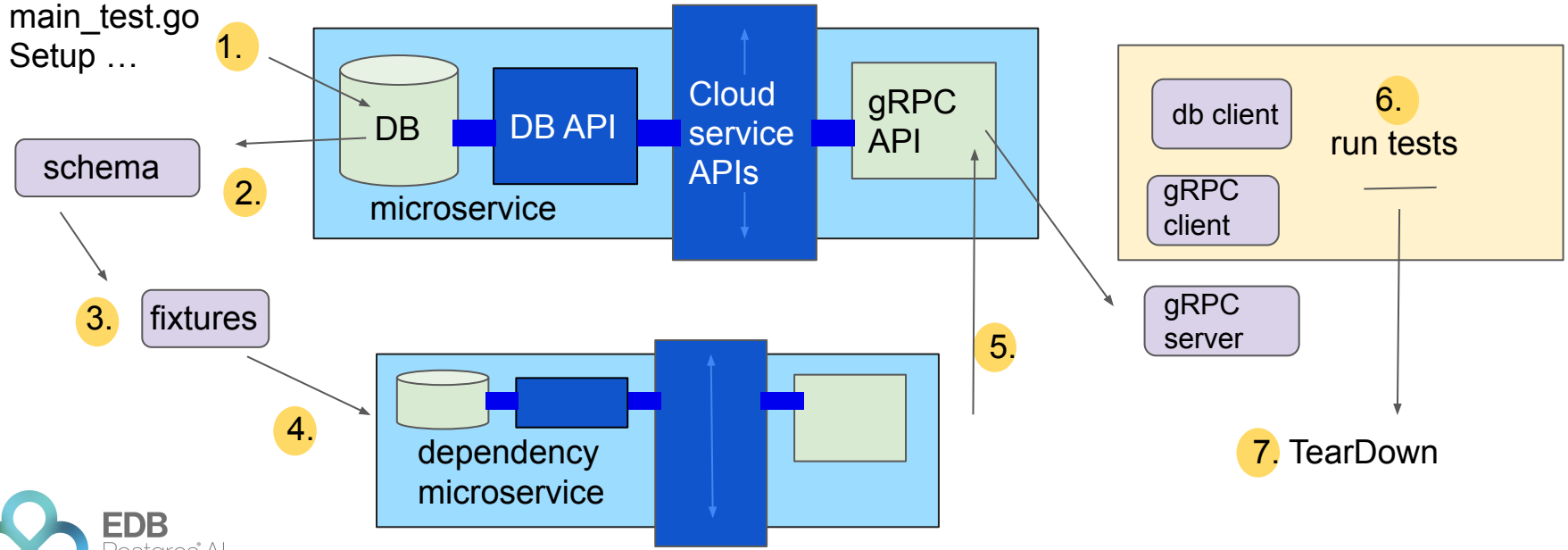


3. DEMO - Test runner steps

0. sequence

main_test.go creates and runs embedded Postgres, creates the schema via tern, populates it with fixtures then runs up in memory gRPC fake microservice (and any dependent microservices) using

`"google.golang.org/grpc/test/bufconn"`



Demo Functional testing for TDD

- Run functional tests in seconds from your IDE
- Demonstrate usage for TDD / refactoring code
- Run at the command line, showing functional test coverage
- Run against a ‘full deploy’ instead, eg. kind

3. Code refactoring and functional test coverage

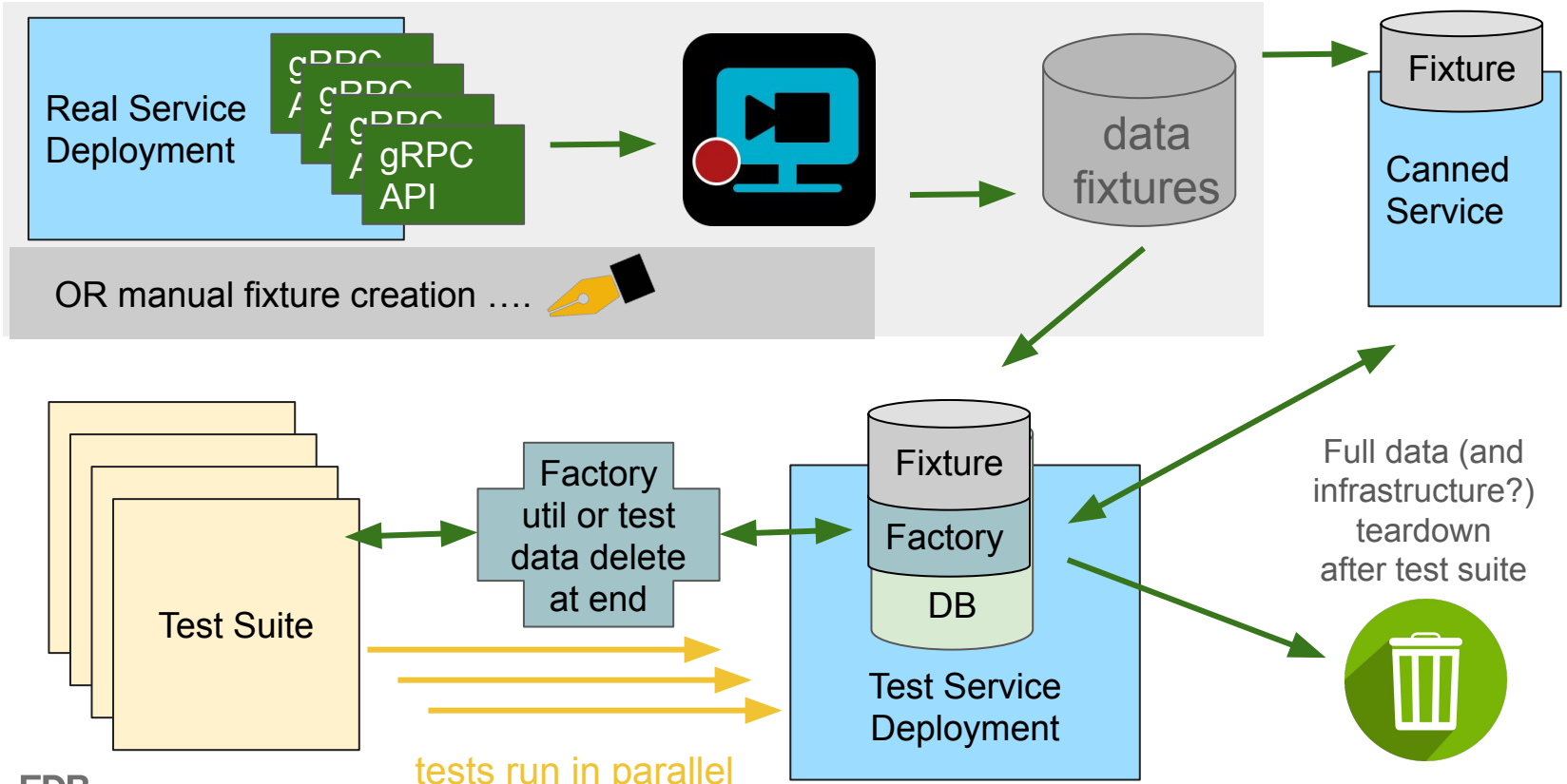
- Code refactoring requires good functional test coverage
- Using a fake framework to run func tests they can provide coverage
- Good functional test coverage already from TDD dev, supports frequent refactoring -> higher code quality

```
go test ./... -tags=functest \
-coverpkg=github.com/EnterpriseDB/$app/pkg/...  \
-count=1 -coverprofile ./coverage.out

go tool cover -html=coverage.out
```

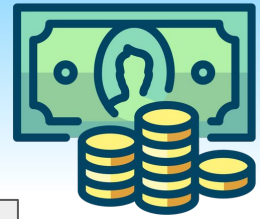
github.com/EnterpriseDB/upm-api-admin/pkg/service/beaco/account/v1/project.go (64.7%) not tracked not covered covered

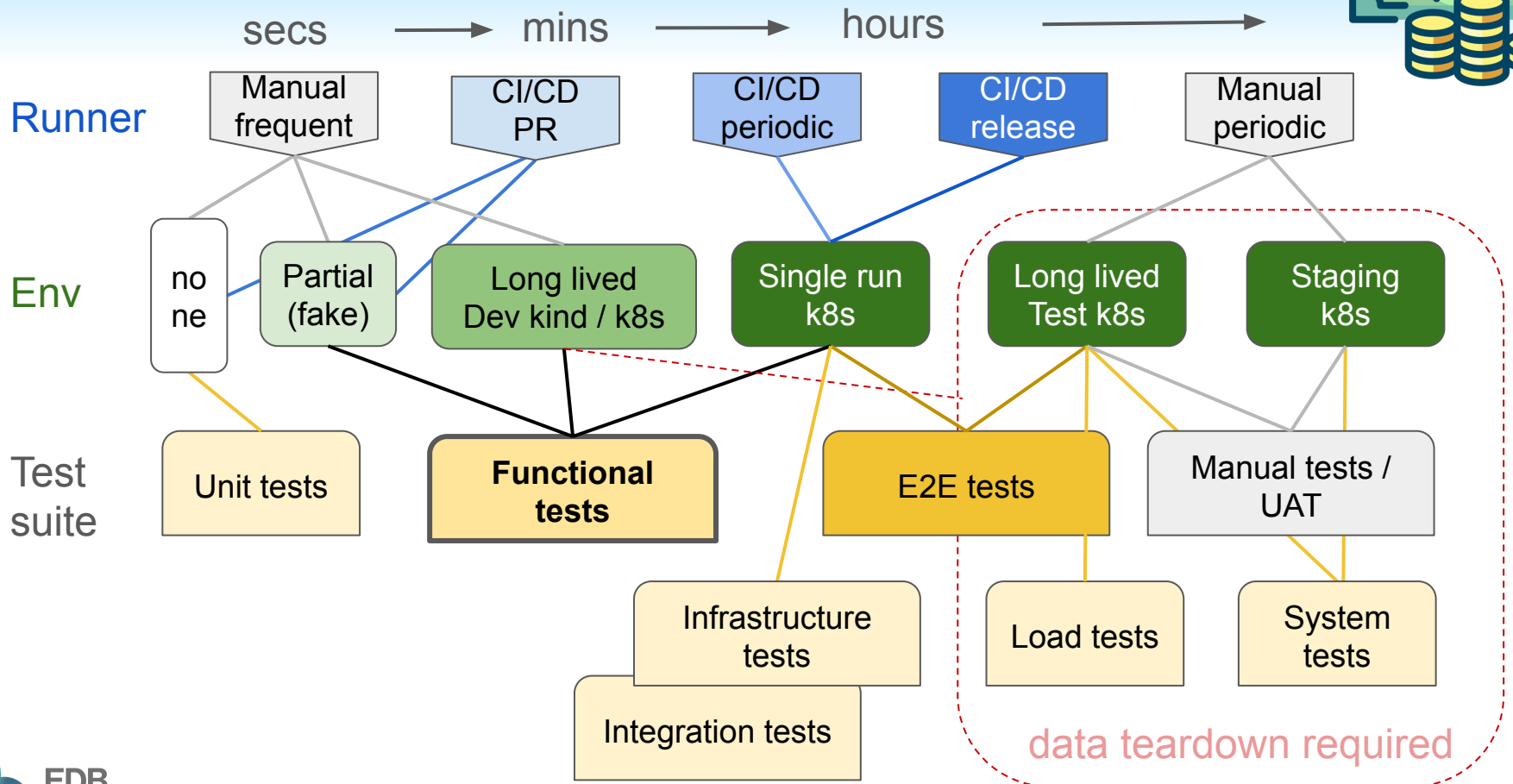
```
        case len(in.ProjectName) > constant.MAX_PROJECT_NAME_LENGTH:
                return fmt.Errorf("project_name should not exceed %d characters", constant.MAX_PROJECT_NAME_LENGTH)
        }
        in.ProjectName = "naming broken" // Demo line
        _, err := doer.ValidateProjectTags(in.Tags)
        return err
}

func toV1Projects(ctx context.Context, projects []*types.Project) ([]*v1.Project, upmerror.UpmError) {
        var (
                rets       = make([]*v1.Project, len(projects))
                projectIds []string
        )
        // find all these projects' tags in a batch
        for _, p := range projects {
                projectIds = append(projectIds, p.ID)
        }
        // doer.ListProjectsTags() ensures tags for the projectIds order
        allTags, err := doer.ListProjectsTags(ctx, projectIds)
        if err != nil {
                log.Get().ErrorWithFields(ctx, map[string]interface{}{"func": "toV1Projects"}, "failed retrieving projects tags", err)
                return nil, upmerror.New(upmerror.StatusInternalServerError, "failed retrieving projects tags", err)
        }

        for i, project := range projects {
                rets[i] = &v1.Project{
                        ProjectId:   project.ID,
                        ProjectName: project.Name,
                        UserCount:   project.UserCount,
                        Tags:        allTags[i],
                }
        }
        return rets, nil
}

// UpdateProject - update an existing project
func (p *Provider) UpdateProject(ctx context.Context, in *v1.UpdateProjectRequest) (out *v1.UpdateProjectResponse, err error) {
        fields := map[string]interface{}{"func": "UpdateProject"}

        if err := validateUpdateProjectRequest(in); err != nil {
                return nil, log.LogAndBuildError(
                        ctx, upmerror.StatusBadRequest, fields, err.Error(), nil,
                )
        }

        // check if project exist or not
        project, err := p.getProject(ctx, p.organizationId, in.ProjectId)
        if err != nil {
```

4. TEST DATA: data life cycle



5. Test types vs. Run environments





Conclusion

1. THE ISSUES of relying on unit and E2E tests for functional code testing.
2. THE SOLUTION of a fake test framework for functional testing, TDD and code refactor
3. DEMO of such a framework
4. RUNNING YOUR TESTS data issues, functional testing against a range of env types

… if you don’t do so already, I hope you will consider Faking It to Make It, too!

Questions?

Ed Crewe - EDB
Bristol, UK

edmundcrewe@gmail.com

https://edcrewe.blogspot.com/
https://www.enterprisedb.com/