Integration testing with TestContainers-Go

Erdem Toraman

Nikolay Kuznetsov

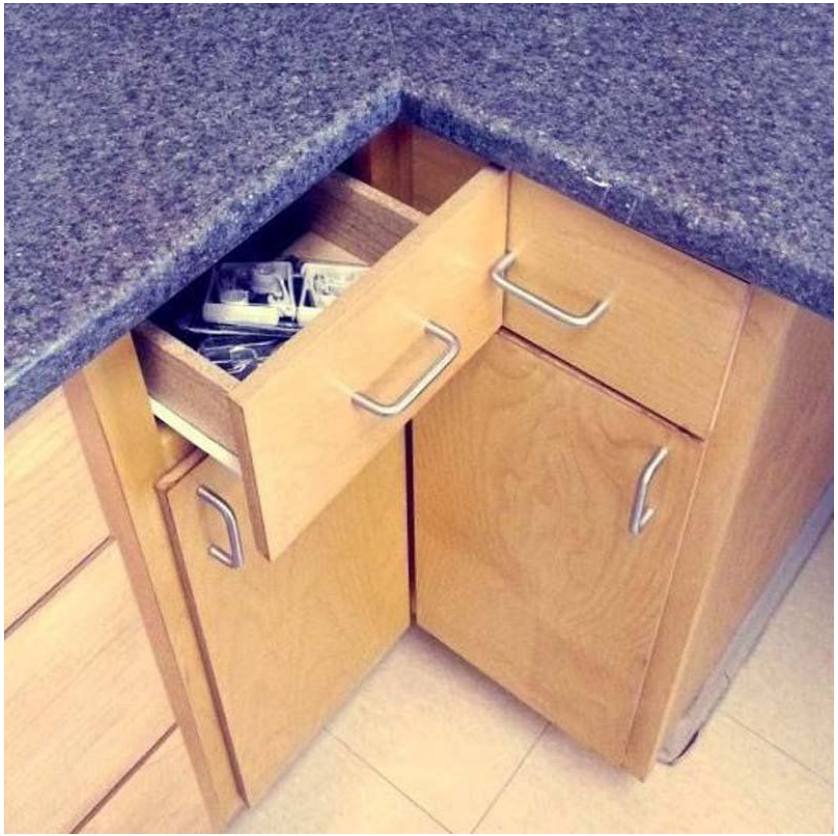
GoDays - Berlin

23 Jan 2020

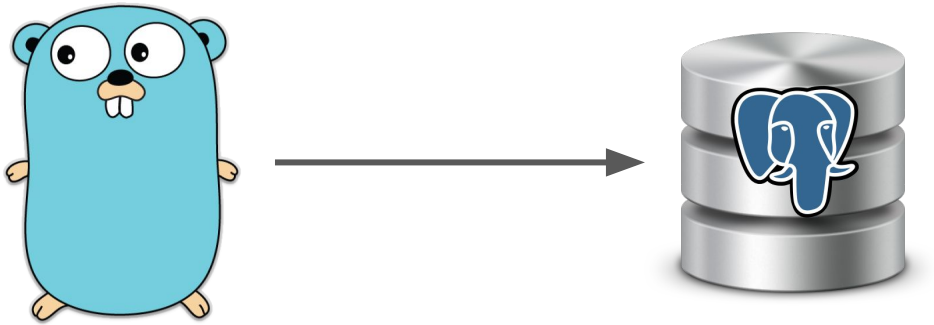
About us

- *Go* developers at **Zalando** in Helsinki
- Project with ~20 microservices in Go
- User perspective of TestContainers-Go library

2 unit tests passed, 0 integration tests



Basic integration test

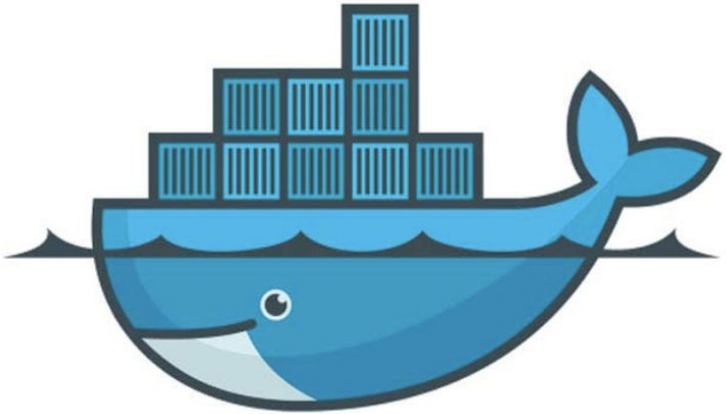


Getting a database for testing

- Local database
- In-memory mock
- **Docker!**

Docker advantages

- 100% compatible databases
- Same version as production
- Empty DB state



Integration testing with Docker

Easy-peasy!

```
docker run -d -p 5432:5432 postgres:12.1
```

What could go wrong?

- Host port conflicts
- Not ready container / service
- Resource leak (the container keeps running)
- Stale data (if reusing the same container)
- Starting mechanism both for CI and a local machine

Solving some issues

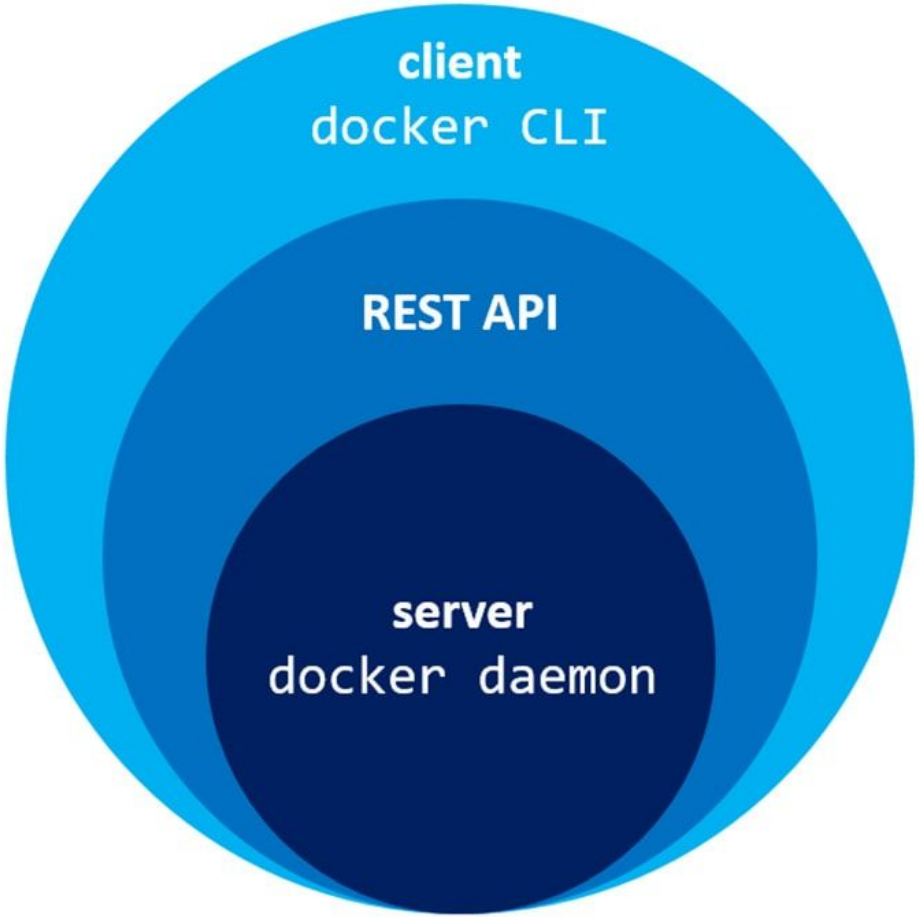
```
#!/bin/sh

DB_NAME=users
TABLE_NAME=users

if [[ ! $(docker ps -q -f name=user-postgres) ]]; then
    if [[ $(docker ps -aq -f status=exited -f name=user-postgres) ]]; then
        # cleanup
        docker rm user-postgres
    fi
    docker run -d --name user-postgres -p 5432:5432 postgres:9.6.13
    sleep 5
    docker exec user-postgres psql --user=postgres -p 5432 -c "CREATE DATABASE $DB_NAME;"
else
    echo "it's already running"
fi
```

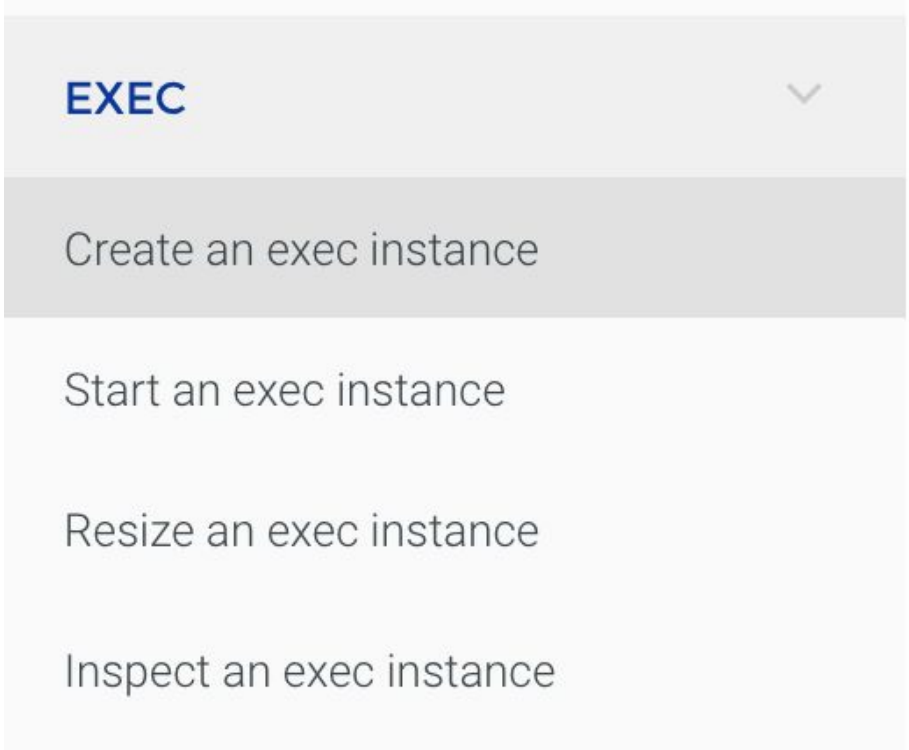

BETTER WAY

Docker API



docs.docker.com/engine/api/latest

Exec example



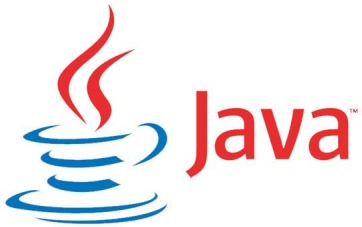
POST /containers/{id}/exec

REQUEST SAMPLES

```
{
    "AttachStdin": false,
    "AttachStdout": true,
    "AttachStderr": true,
    "DetachKeys": "ctrl-p,ctrl-q",
    "Tty": false,
  - "Cmd": [
        "date"
    ],
  - "Env": [
        "FOO=bar",
        "BAZ=quux"
    ]
}
```


TESTCONTAINERS

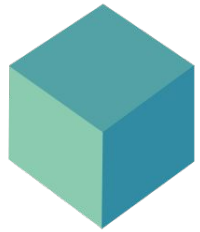
TestContainers flavors



TestContainers-Go

github.com/testcontainers/testcontainers-go

- *Docker Go client* under the hood
- Host port randomization
- Containers clean up at the test shutdown
- Readiness waiting strategies



As simple as

```
pgContainer, err := tc.GenericContainer(ctx,
  tc.GenericContainerRequest{
    ContainerRequest: tc.ContainerRequest{
      Image:        "postgres:12.1",
      ExposedPorts: []string{"5432/tcp"},
    },
  })
```

Host port randomization

```
IMAGE                PORTS
postgres:12.1        0.0.0.0:32770->5432/tcp
```

- API to get a host port:

```
port, err := pgContainer.MappedPort(ctx, "5432/tcp")
```

- Prevents port conflicts
- Enables parallel builds

Containers cleanup: Ryuk

github.com/testcontainers/moby-ryuk

- Ryuk kills containers (networks, volumes) by labels
- TC assigns labels to started containers
- TC keeps a connection to Ryuk open

```
IMAGE                                  golang                 sessionId
postgres:12.1                          true                   2a0770fa-e
quay.io/testcontainers/ryuk:0.2.2      true
```

Waiting strategies

- Host port
- HTTP
- Logs
- *Custom*
- Multi

Host port waiting strategy

```
tc.ContainerRequest{
  Image:        "postgres:12.1",
  ExposedPorts: []string{"5432/tcp"},
  WaitingFor: wait.ForListeningPort("5432/tcp"),
},
```

- Default (customizable) timeout is 60 seconds
- Impl checks both from outside and **inside** container

HTTP waiting strategy

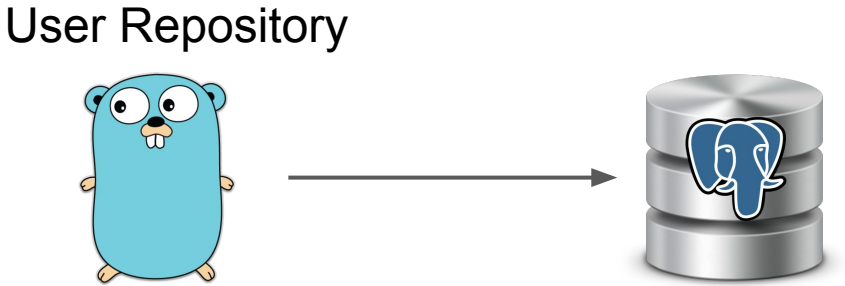
```
WaitingFor: wait.ForHTTP("/health").
    WithPort("8080/tcp").
    WithStatusCodeMatcher(
        func(status int) bool { return status == http.StatusOK }),
```

Demo

Demo I: integration testing

github.com/erdemtoraman/godays-testcontainers-demo

- Testing interaction with a database
- Create and Get users from Postgres

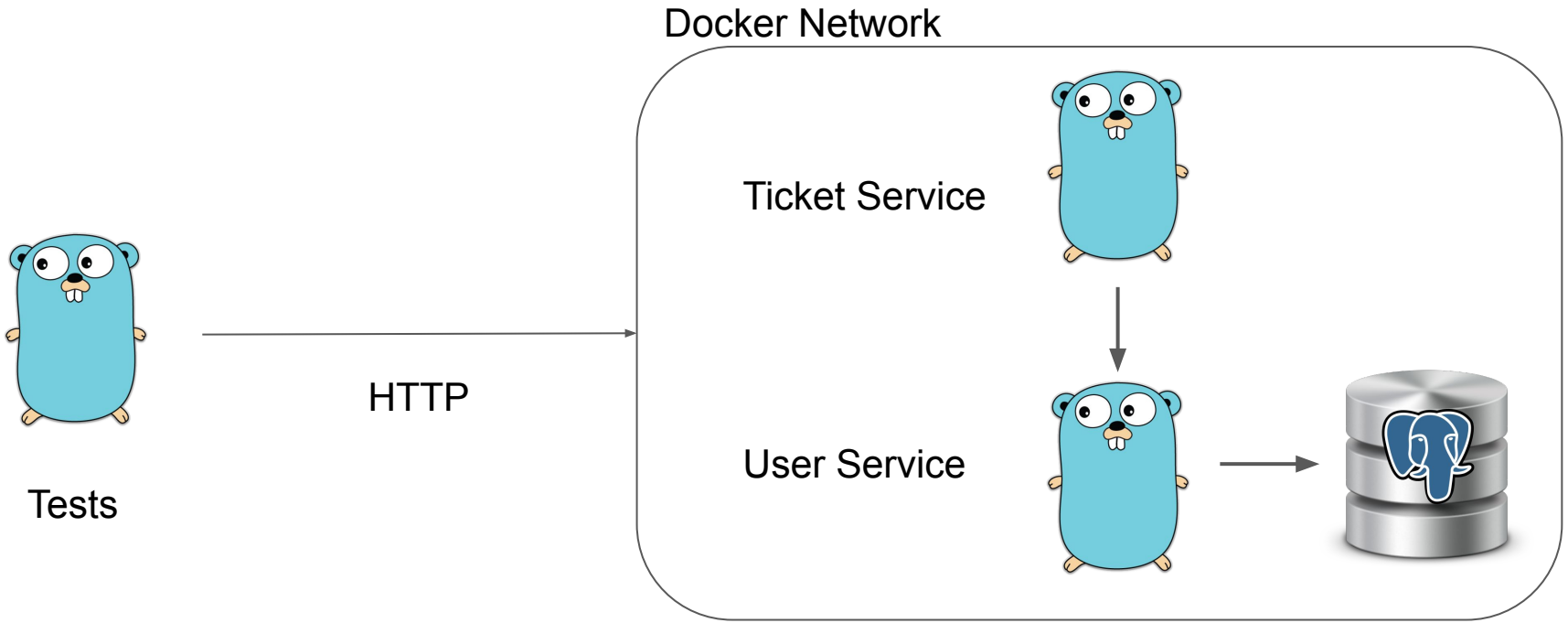


What happened?

- Ran a Postgres container
- Mapped a random port to the host machine
- Ran tests against it
- Cleaned up the containers

Demo II: end-to-end testing

github.com/erdemtoraman/godays-testcontainers-demo



Takeaways

- testcontainers.org
- Balance between flexibility, speed and features
- Great for integration tests
- Possible to use for end-to-end tests

Thank you!

@erdem_toraman, @nikolayk812

erdemtoraman