

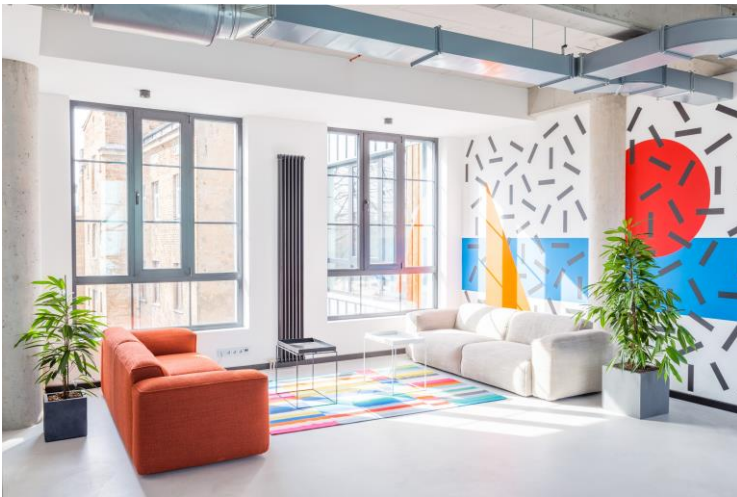


Cloud-native apps
REST API testing

Kirill Shepitko

Konstantins Tarasjuks

Who We Are & What We Do



 neotech.lv

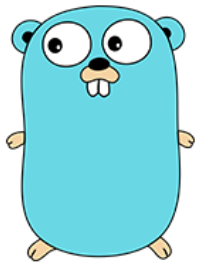


 neotechlv



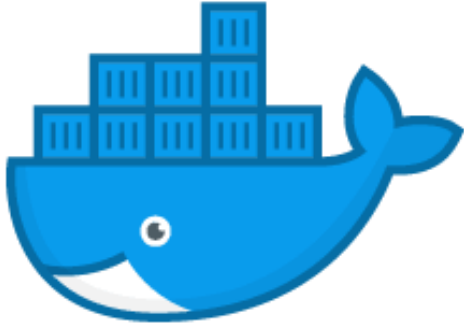
www.neotech.lv





How We Do It





docker




Jenkins



kubernetes

How We Deliver It



What is a
cloud-native
app?

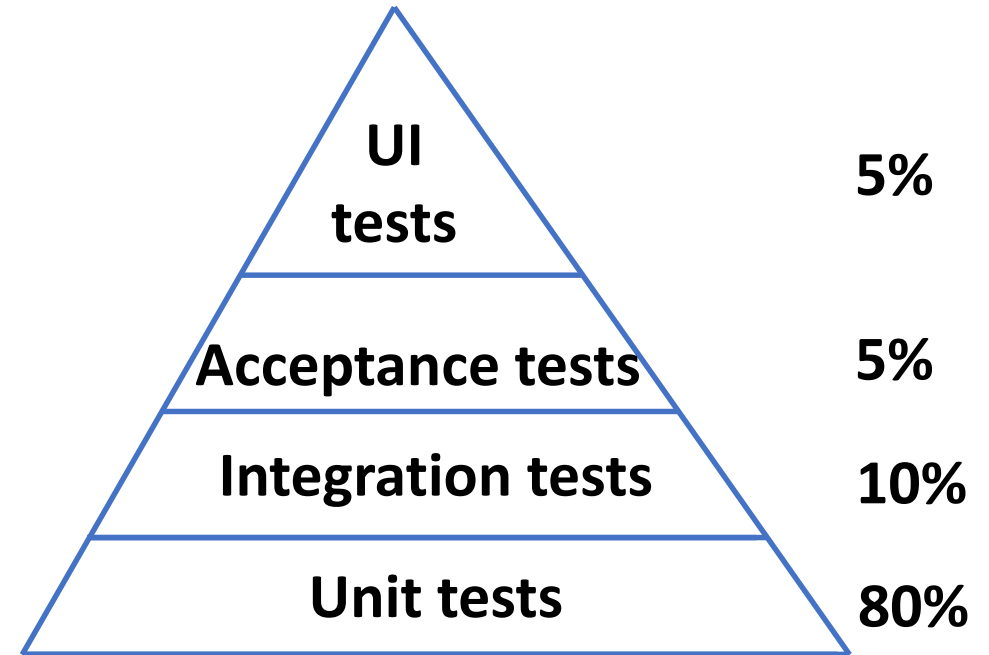
«Cloud native is an approach to building and running applications that fully exploit the advantages of the cloud computing model.»

Source: [What are Cloud-Native Applications? - Pivotal](#)

- **Relies on IaaS**
- **Scalable, resilient, self-healing, stateless**
- **Containerized**

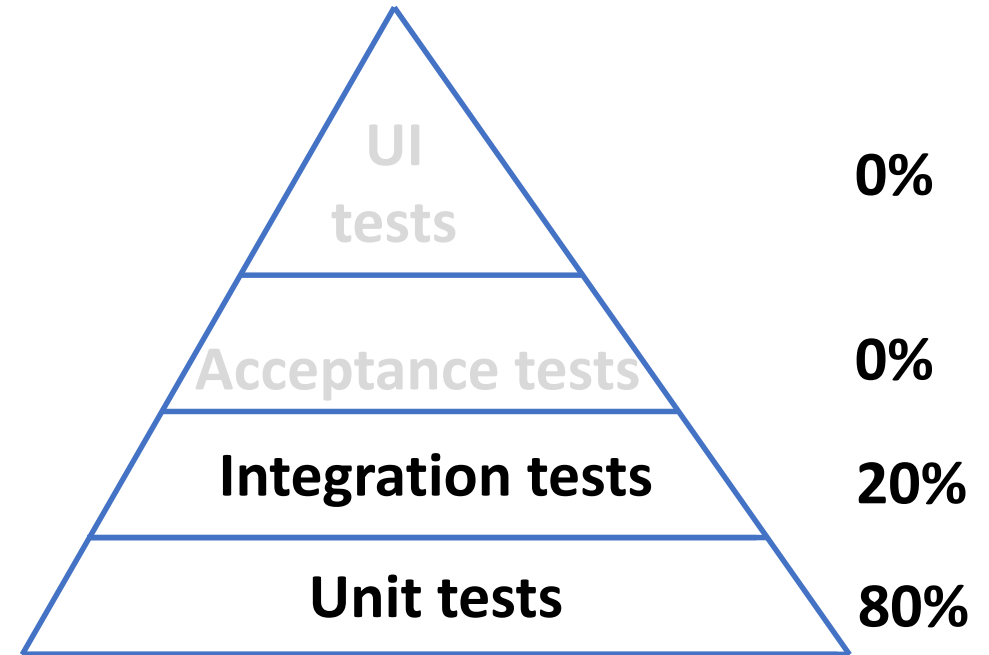
Test pyramid

- UI – 5%
- Acceptance – 5%
- Integration – 10%
- Unit – 80%



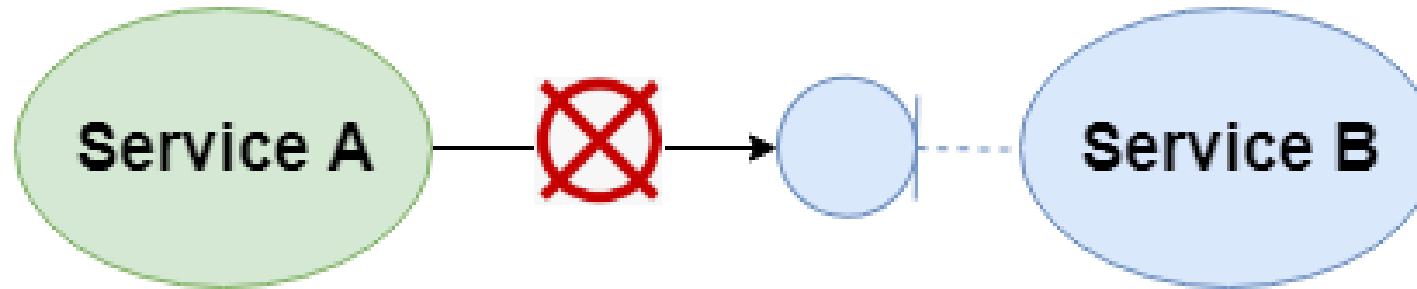
Our test pyramid

- UI – 0%
- Acceptance – 0%
- **Integration – 20%**
- **Unit – 80%**

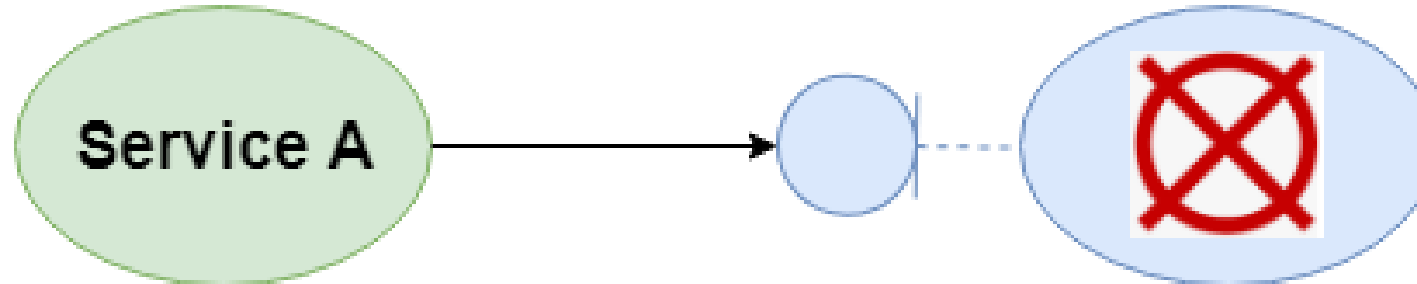


Problems we had

Contract broken



Deployment is broken





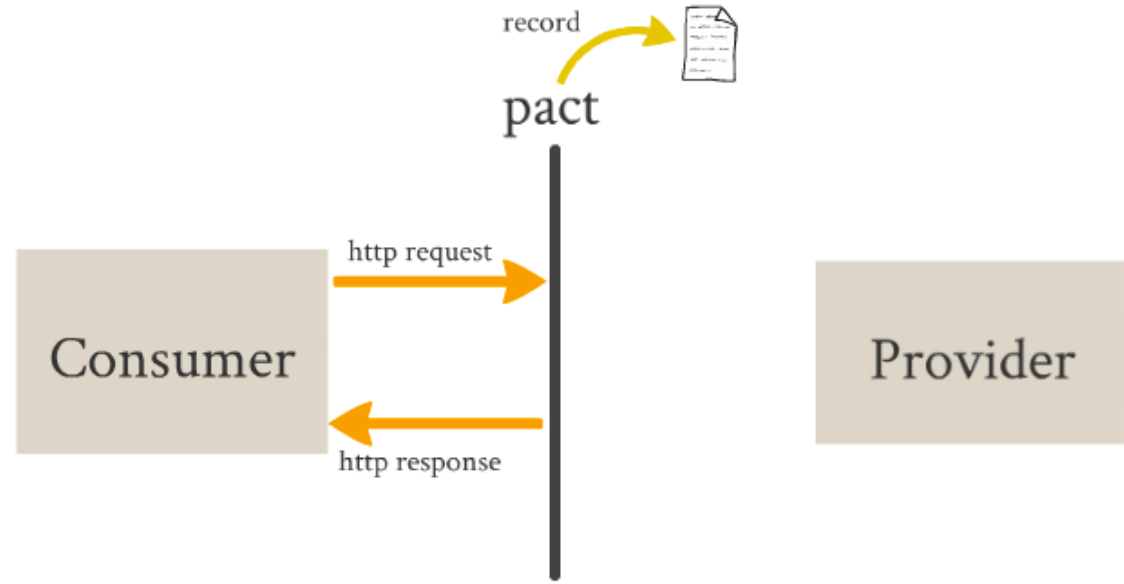
What we needed

- REST API acceptance tests
- Tests should test REST contracts
- Tests should be fast
- Should be able to mimic correspondent systems
- Should be able to detect the deployment problems
- Should be lightweight enough to run tests on dev machines
- Should have nice reporting

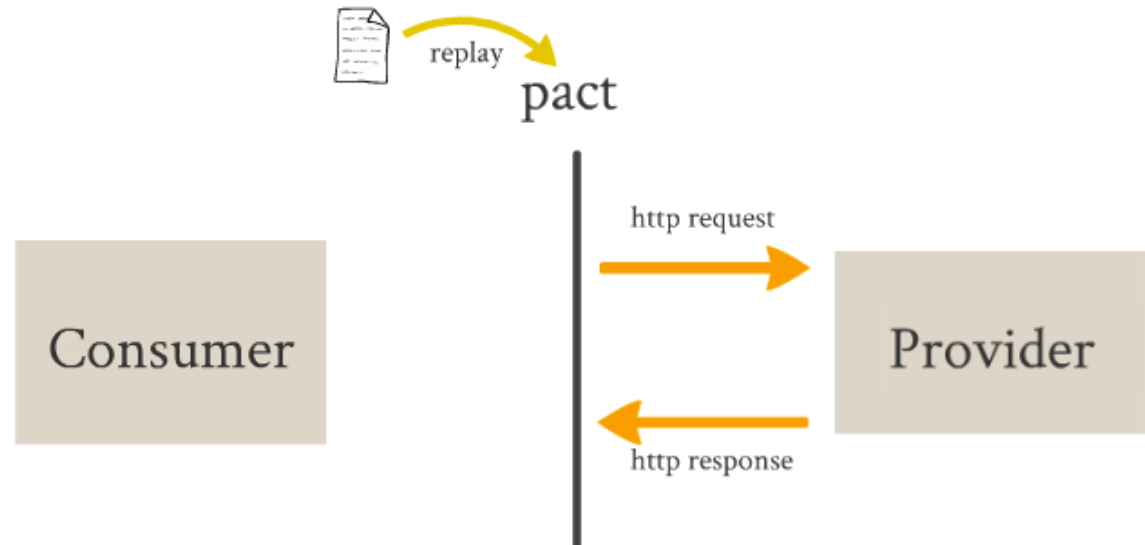
REST API contract testing



Step 1 - Define Consumer expectations



Step 2 - Verify expectations on Provider



Existing Frameworks' Issues

Great for a “Hello world” app

Lots of very similar contracts
(maintain it!)

Tests are generated

Lot of code to write

How Spring Cloud Contract Works



1. Write contract
2. Verify contract from producer side (automated)
3. Publish service stubs to artifactory (there's a plugin)
4. Write a test on client side, using stubs

Writing Spring Cloud Contract

- Groovy
- YAML
- Pact JSON
- Spring REST docs

```
Contract.make {  
    description "should return person by id=1"  
  
    request {  
        url "/person/1"  
        method GET()  
    }  
  
    response {  
        status 200  
        headers {  
            contentType applicationJson()  
        }  
        body (  
            id: 1,  
            name: "foo",  
            surname: "bee"  
        )  
    }  
}
```

Spring Cloud Contract for Polyglots

```
# Install the required npm packages
$ npm install

# Stop docker infra (mongodb, artifactory)
$ ./stop_infra.sh
# Start docker infra (mongodb, artifactory)
$ ./setup_infra.sh

# Kill & Run app
$ pkill -f "node app"
$ nohup node app &

# Prepare environment variables
$ export SC_CONTRACT_DOCKER_VERSION="1.2.3.RELEASE"
$ export APP_IP="192.168.0.100" # This has to be the IP that is available out
$ export APP_PORT="3000"
$ export ARTIFACTORY_PORT="8081"
$ export APPLICATION_BASE_URL="http://${APP_IP}:${APP_PORT}"
$ export ARTIFACTORY_URL="http://${APP_IP}:${ARTIFACTORY_PORT}/artifactory/li
$ export CURRENT_DIR="$( pwd )"
$ export PROJECT_NAME="bookstore"
$ export PROJECT_GROUP="com.example"
$ export PROJECT_VERSION="0.0.1.RELEASE"

# Execute contract tests
$ docker run --rm -e "APPLICATION_BASE_URL=${APPLICATION_BASE_URL}" \
-e "PUBLISH_ARTIFACTS=true" -e "PROJECT_NAME=${PROJECT_NAME}" \
-e "PROJECT_GROUP=${PROJECT_GROUP}" -e "REPO_WITH_BINARIES_URL=${ARTIFACTORY_
-e "PROJECT_VERSION=${PROJECT_VERSION}" -v "${CURRENT_DIR}/contracts:/contra
-v "${CURRENT_DIR}/node_modules/spring-cloud-contract/output:/spring-cloud-co
springcloud/spring-cloud-contract:${SC_CONTRACT_DOCKER_VERSION}"

# Kill app
$ pkill -f "node app"
```

The Solution

Describe
REST endpoints



Create scenario
(pact)



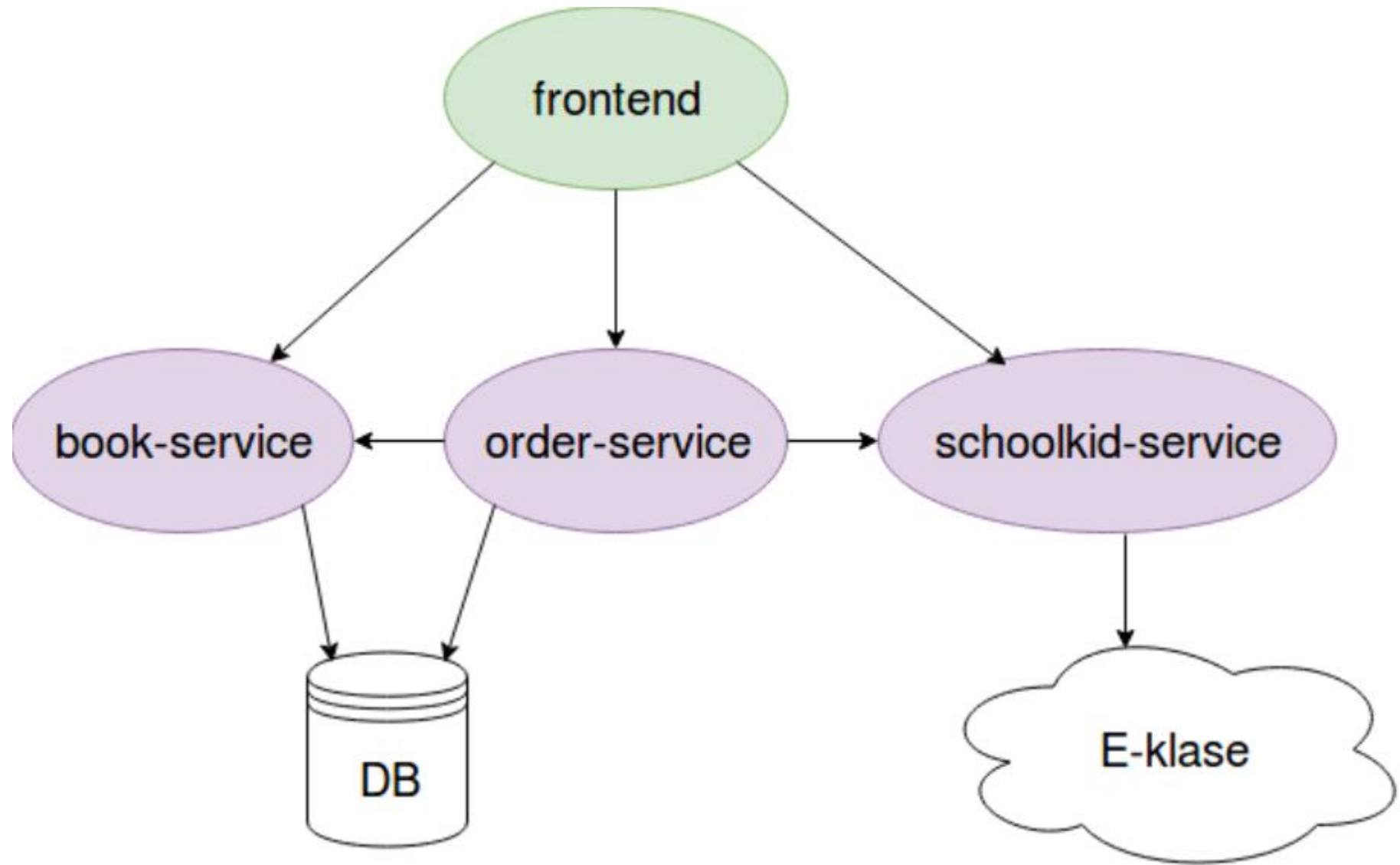
Author test
steps



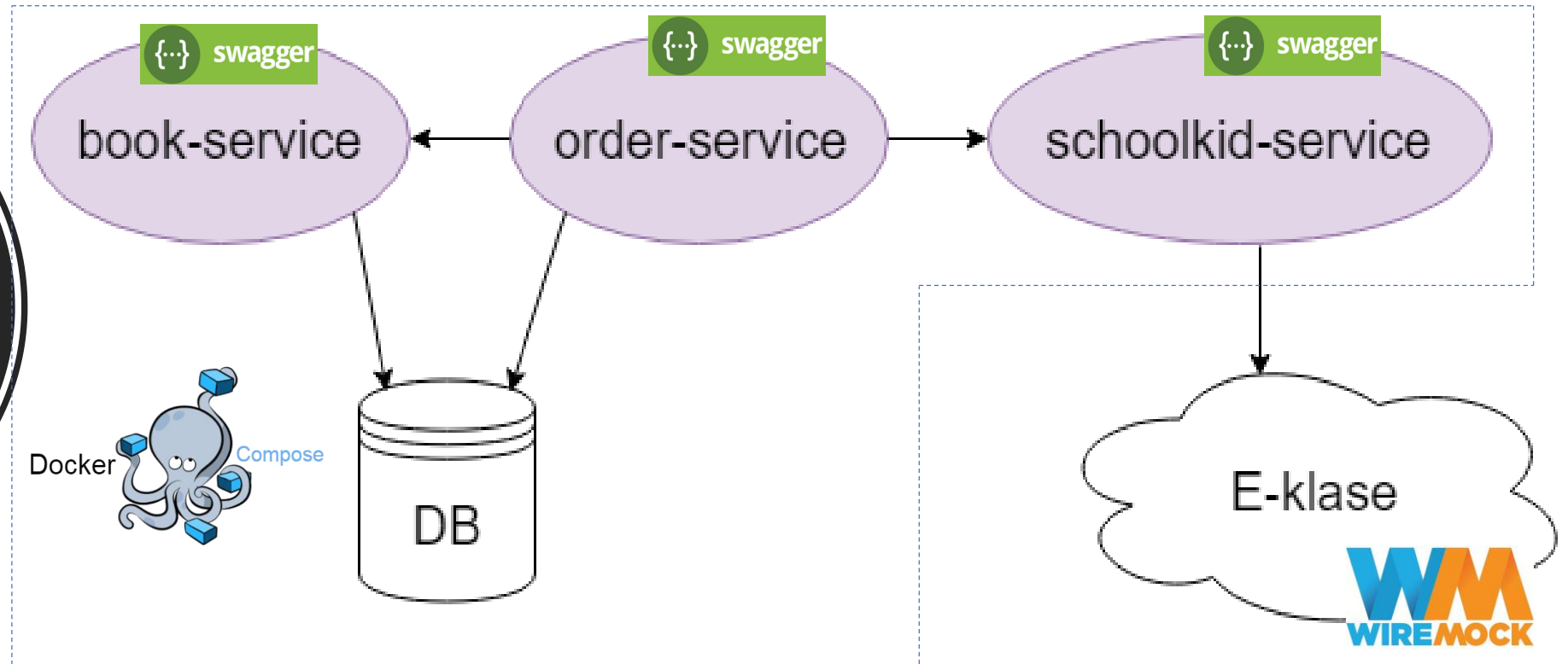
Run tests



Imagine
a sample
app...



How we test REST API



Before Tests

1

Manually start
up the app
ecosystem

2

Pull Swagger
specs

3

Save Swagger
specs to tests
Git repo

Test Flow

1

Generate REST
clients

2

Run tests

- Launch ecosystem via *TestContainers*
- Launch mocks
- Run scenarios

3

Generate reports

Benefits

- Scenario is a contract
- WireMock is useful for mocks and trouble emulation
- Test steps and mocks can be reused
- Easy to bring up the whole app ecosystem
- REST client generation approach is polyglot
- Great reports

Lessons Learned

- QA needs a faster solution



- REST client may be stale

- Supporting different test profiles

- Regenerating REST clients on the fly

Updated Test Flow

1

Start up the app ecosystem and download Swagger JSON

2

Generate REST clients (Git-ignored)

3

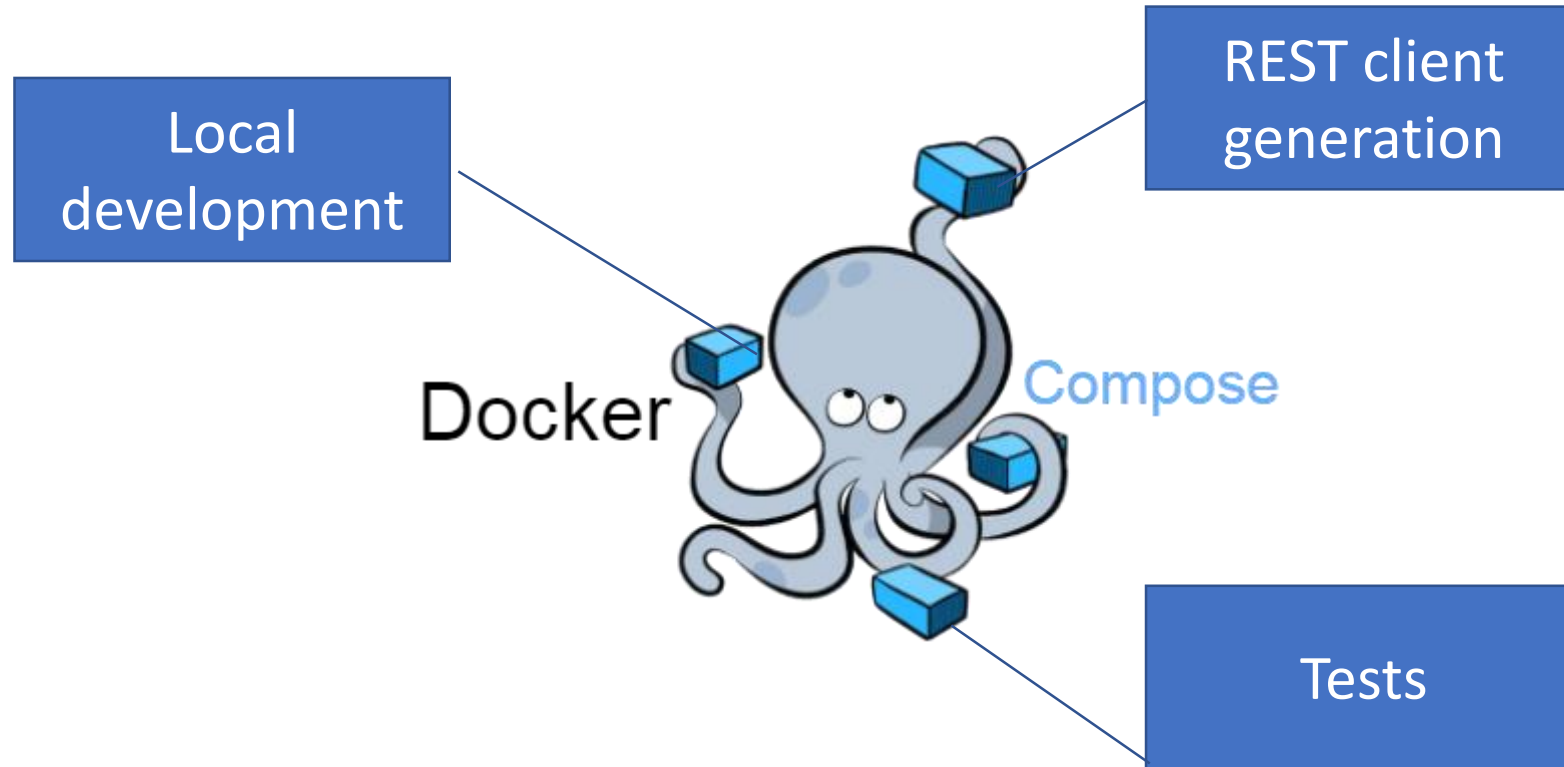
Run tests

- Launch ecosystem again
- Launch mocks
- Run scenarios

4

Generate reports

Docker-compose Is a God



Problems

- No clear boundary between contract and acceptance testing
- TestContainers bind the solution to JUnit 4
- Springfox (generates Swagger spec) is quite limited

Swagger

Api Documentation

Api Documentation

[Apache 2.0](#)

orders-controller : Orders Controller

Show/Hide

List Operations

Expand Operations

GET	/api/v1/orders	Gets orders
POST	/api/v1/orders	Creates a new order order
DELETE	/api/v1/orders/{id}	Deletes an order
GET	/api/v1/orders/{id}	Gets order by id
PUT	/api/v1/orders/{id}	Updates order info

[BASE URL: / , API VERSION: 1.0]

Scenario

@orders
@severity=critical
Feature: Order creation

Background:

Given School library service is up and running

Scenario: Successful order creation by adding one book to one person

Given verification at `eclass` succeeds

When adding a book "War and peace" for order for Maria Curie and it is taken until "02/12/2020"

Then order is successfully created

Scenario: Successful order creation by adding two books to one person

Given verification at `eclass` succeeds

When adding a book

book	name
War and peace	Maria Curie
The Lord of Flies	Maria Curie

Then order is successfully created

Scenario Outline: Order creation fails due invalid date and not existent person

Given verification at `eclass` succeeds

When adding a book "<book>" for order for <name> <surname> and it is taken until "<date>"

Then error message is displayed "Can not create order"

Examples:

book	name	surname	date
War and peace	Maria	Sinatra	02/12/2019
War and peace	Maria	Curie	02/12/2013

Generated REST client in use

```
Response<BookInfoDto> addNewBook(Map<String, String> params) {  
    UpdateBookDto item = createNewBook(params);  
    BooksControllerApi booksControllerApi = apiClient.createService(BooksControllerApi.class);  
  
    Call<BookInfoDto> call = booksControllerApi.addUsingPOST(item);  
    try {  
        return call.execute();  
    } catch (IOException e) {  
        throw new RuntimeException(e);  
    }  
}
```

```
Response<BookInfoDto> updateNewBook(String bookNameOld, String bookNameNew) {  
  
    String id = getBookIdByName(bookNameOld);  
    UpdateBookDto item = updateBookName(id, bookNameNew);  
    BooksControllerApi booksControllerApi = apiClient.createService(BooksControllerApi.class);  
    Call<BookInfoDto> call = booksControllerApi.updateUsingPUT(id, item);  
    try {  
        return call.execute();  
    } catch (IOException e) {  
        throw new RuntimeException(e);  
    }  
}
```

Service mock

```
@Singleton
public class EklaseMock extends AbstractServiceMock {

    private static final int PORT = 8103;

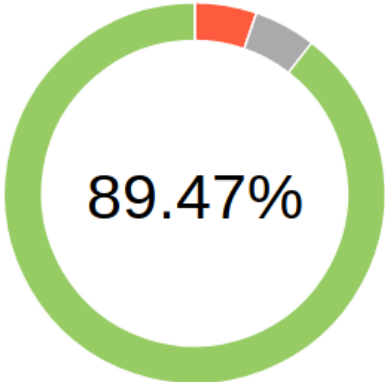
    @Inject
    public void init() {
        start(PORT, new ClasspathFileSource("eclass"));
    }

    public void mockVerify(Boolean noError) {
        mock.stubFor(
            post(urlPathEqualTo( testUrl: "/e-class/v1/api"))
                .willReturn(aResponse()
                    .withHeader( key: "Content-Type", ...values: "application/json")
                    .withBodyFile(noError ? "eclass_response_success.json" : "eclass_response_failure.json")
                    .withStatus( noError ? 200 : 501)
                ));
    }
}
```

Allure report: overview

ALLURE REPORT
28/09/2018
13:56:36 - 13:56:37 (1s 226ms)

19
test cases



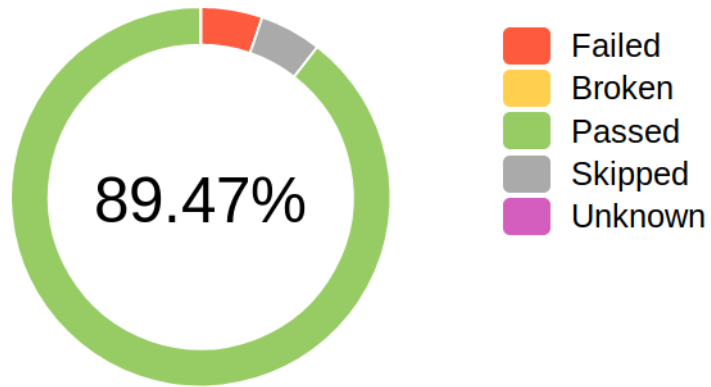
SUITES 4 items total

Books management	1	5
Order management	6	1
Order creation	5	
Schoolkids management	1	

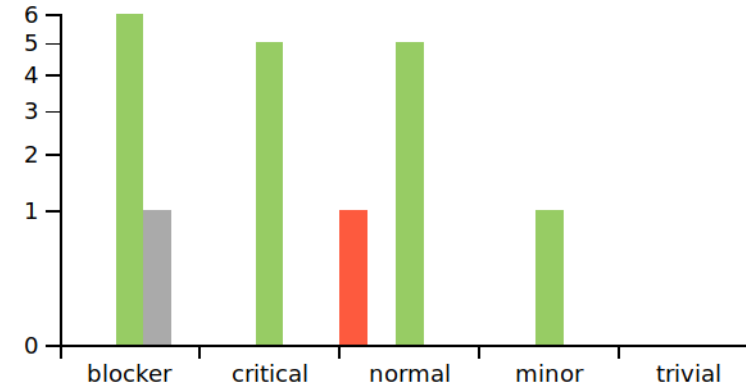
Show all

Allure report: graphs

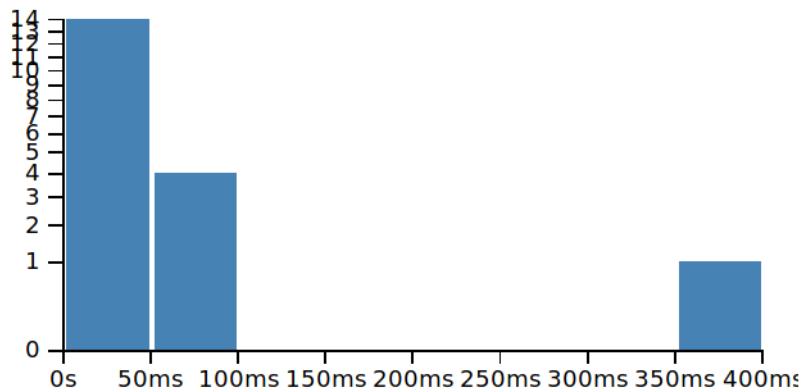
STATUS



SEVERITY



DURATION



Allure report: behaviours

Behaviors



order name duration status

Filter by status: 1 0 17 1 0

Books management

1 5

> All books exist

1

> Fails to delete book due nonexistent book

1

✓ #1 Fails to delete book due nonexistent book 33ms

> Fails to update book due invalid book data

1

> Fails to update book due nonexistent book

1

> Successful book deletion

1

✗ #1 Successful book deletion 78ms

> Successful book update

1

> Order creation

5

Passed

Fails to delete book due nonexistent book

Overview

History

Retries

Tags: books bookDelete bookUpdate

Severity: normal

Duration: ⌚ 33ms

Execution

> Test body

✓ Given School library service is up and running 27ms

✓ Given delete book "123988" 1ms

✓ And error message is displayed "Book can not be deleted" 0s

Demo

ature: Books management

Background:

Given School library service is up and running

Scenario: All books exist

Given all books

Then following books exist:

name	isbn
War and Peace	978-3-16-148410-0
The Getaway	778-3-16-148410-0
Moby Dick	723-3-56-121410-0
The Lord of Flies	778-3-23-141234-0
Animal Farm	778-2-16-345871-0

Scenario: Successful book update

Given adding a book "The Shining" for Elvis Presley with id "123997"

And book is successfully created

When update book "123997" with book name "Learn Java In 30 Days"

And book is successfully updated

Scenario: Fails to update book due nonexistent book

Given update book "123999" with book name "Learn Java In 30 Days"

And error message is displayed "Book can not be updated"

Scenario: Fails to update book due invalid book data

Given adding a book "Hello World" for Elvis Presley with id "123998"

And book is successfully created

When update book "123998" with book name "The Getaway"

And error message is displayed "Book can not be updated"

Scenario: Successful book deletion

Given adding a book "Learn to Draw In 30 Days" for John Doe with id "123987"

And book is successfully created

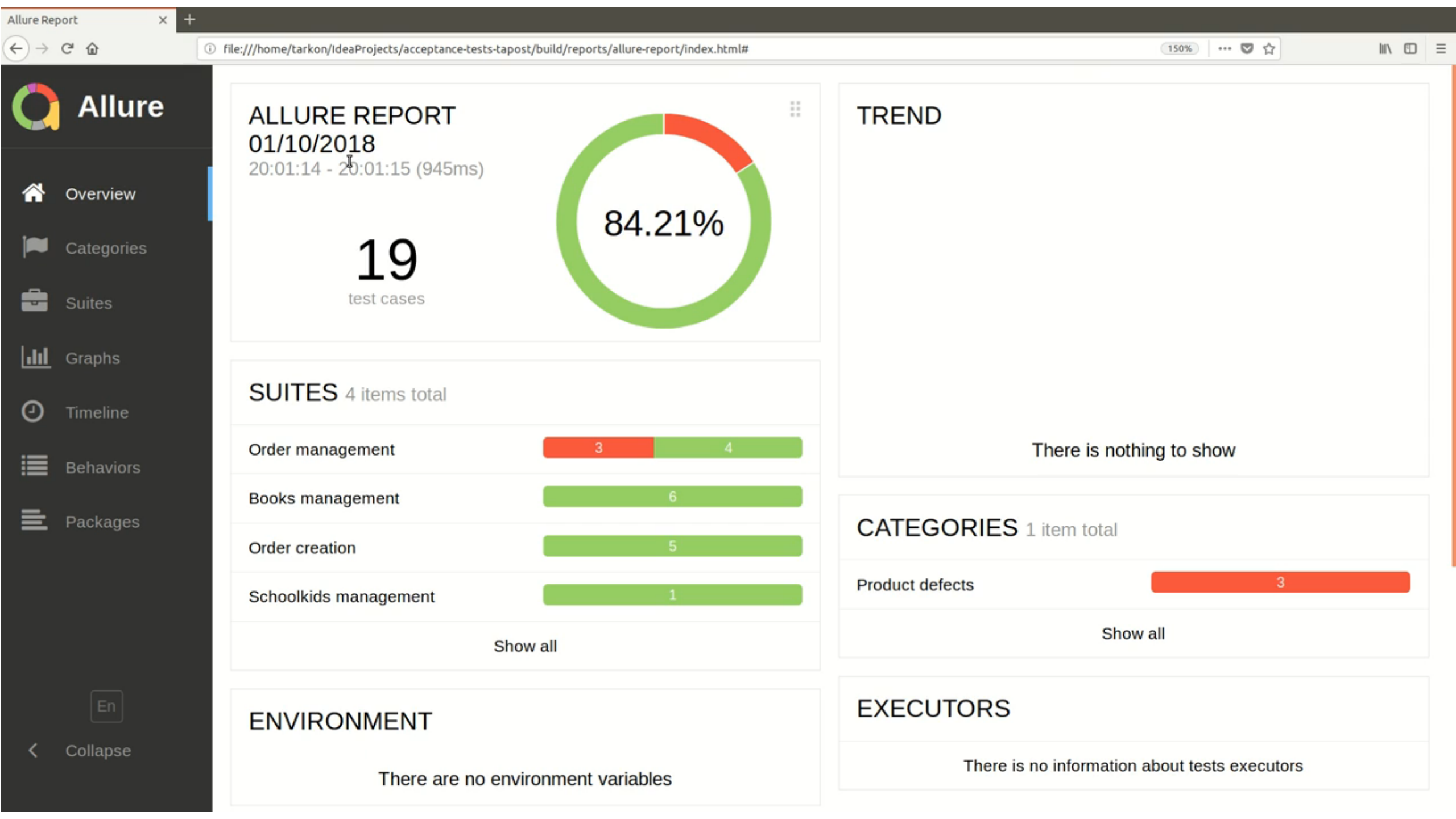
When delete book "123987"

And book is successfully deleted

Scenario: Fails to delete book due nonexistent book

Given delete book "123988"

And error message is displayed "Book can not be deleted"



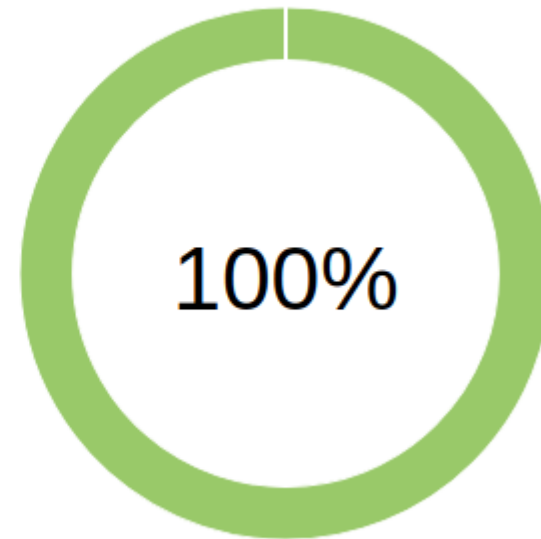
Real performance

ALLURE REPORT 20/09/2018

16:16:20 - 16:24:02 (7m 41s)

1642

test cases





Our framework is open-source



<https://github.com/neotechlv>



We're done.



Questions?

Thank you for listening!