Вспомогательные приемы при тестировании μсервисов.

Мартюшов Александр @SIGNAVIO

О себе

- Занимаюсь тестированием около 8 лет.
- Проекты:
  - IPTV системы
  - e-commerce
  - location-based сервисы
- Backend, UI & mobile тестирование
- Jenkins (pipeline implementation)

О компании

SIGNAVIO

signavio.com

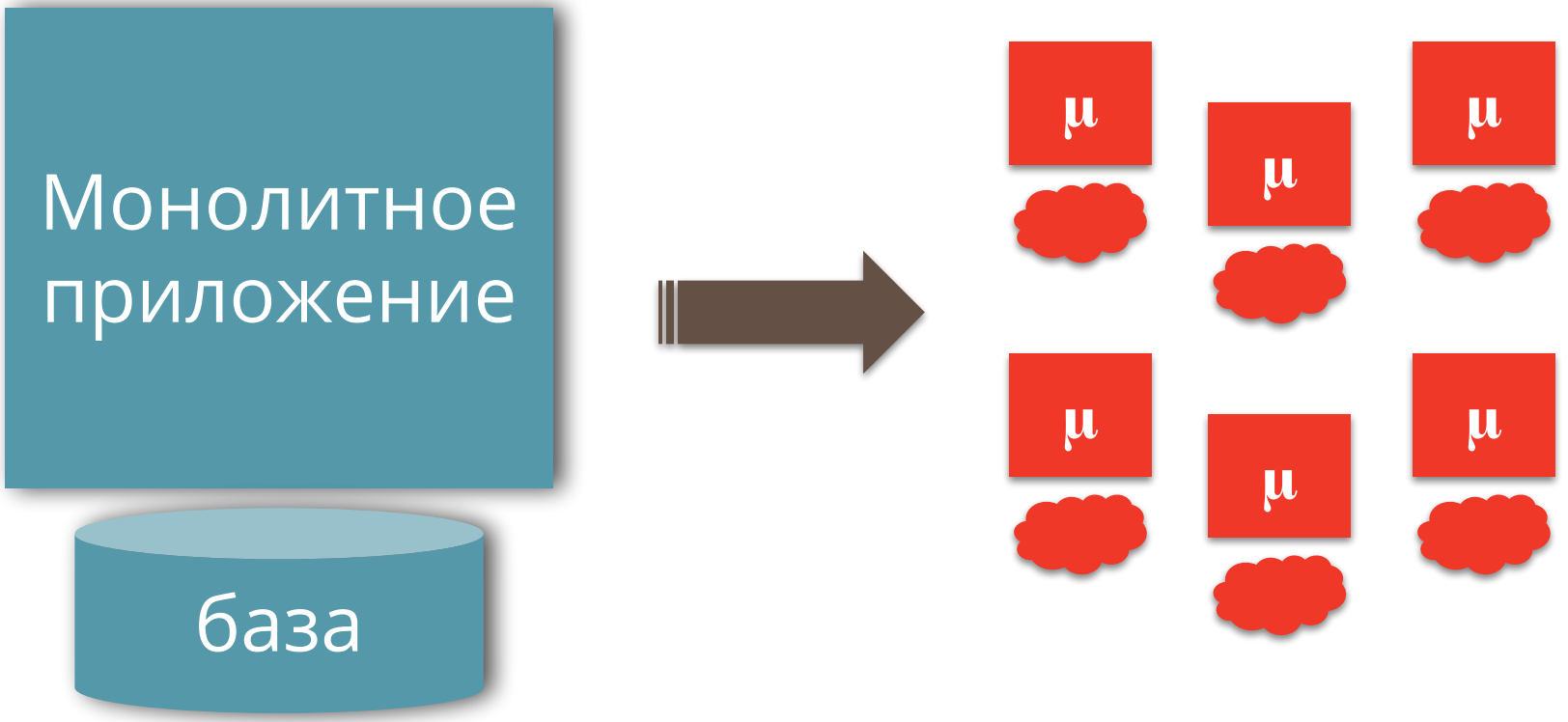
- Продуктовая компания
- Решение для:
  - автоматизации бизнес процессов
  - финансовой оптимизации процессов
- BPMN для описания процессов
  - Business Process Model and Notation

BPMN™

О чем будем говорить сегодня

Это мы хорошо умеем

Надо потестировать µсервисы



Сказано - сделано

Действительно ли нет проблем?

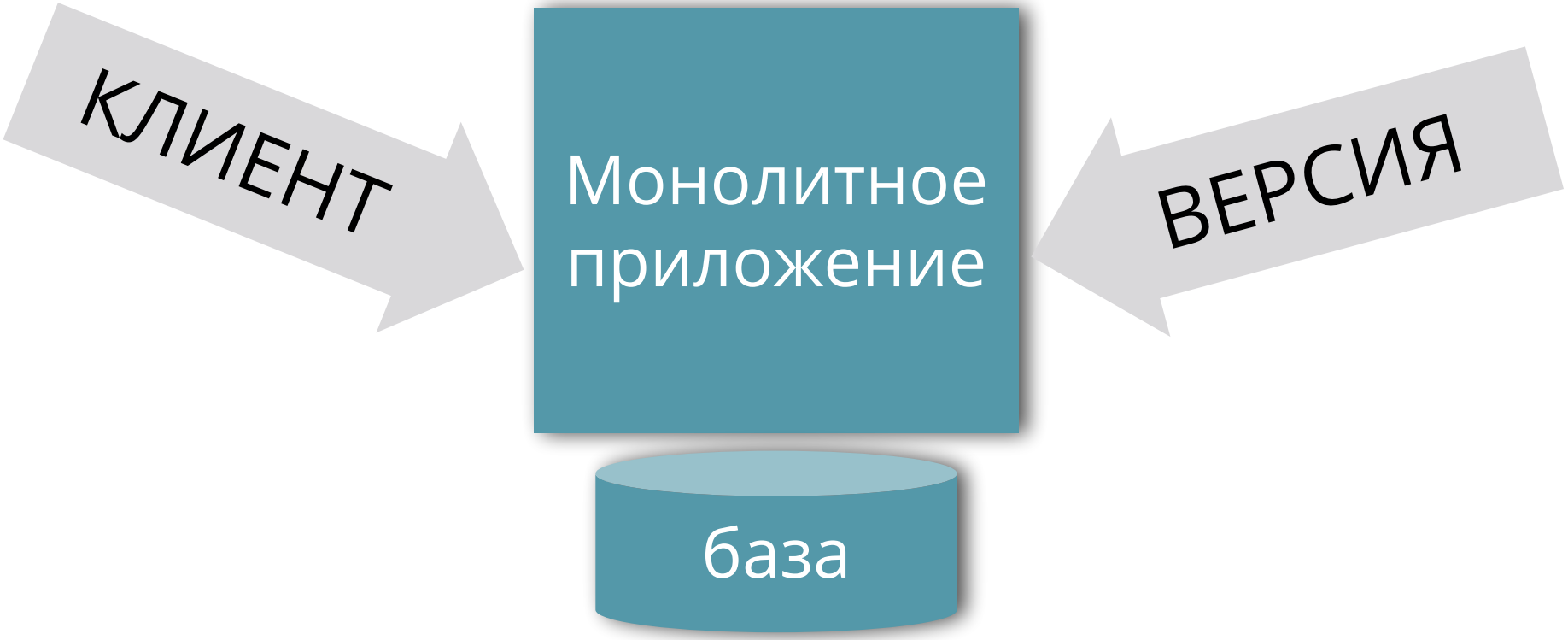
Цель доклада

- На **собственном** опыте
- Как решал проблемы, **специфичные** для μсервисного проекта

Почему подходят не все старые приемы тестирования?

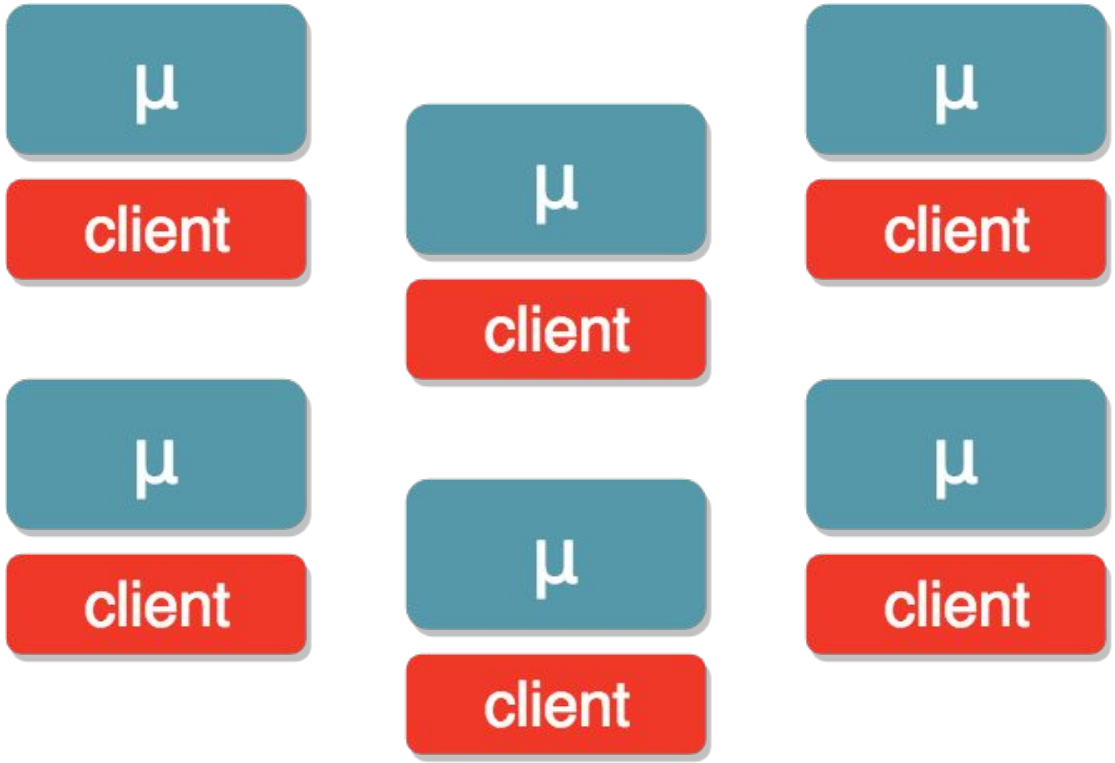
Почему подходят не все старые приемы тестирования?

У μсервисов есть специфика

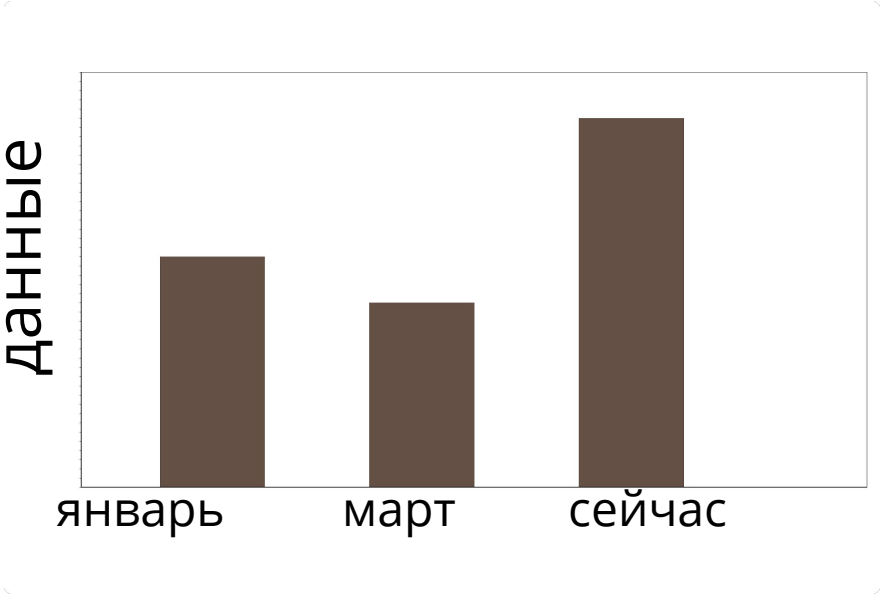

КЛИЕНТ
Монолитное приложение
ВЕРСИЯ
база


BOOM!

1. Поддержка HTTP клиентов

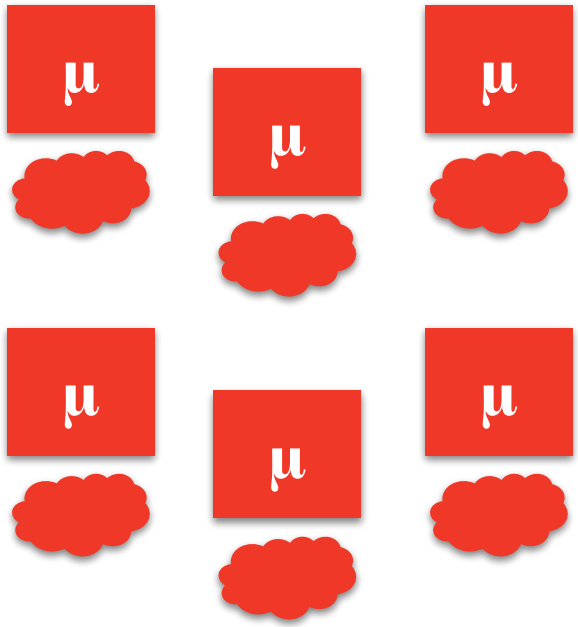


2. Эмуляция тестовых данных за длительные промежутки времени



Подготовка тестовых данных

Подготовка тестовых данных



Этот сервис вообще забыли

А где база?

Памяти мало

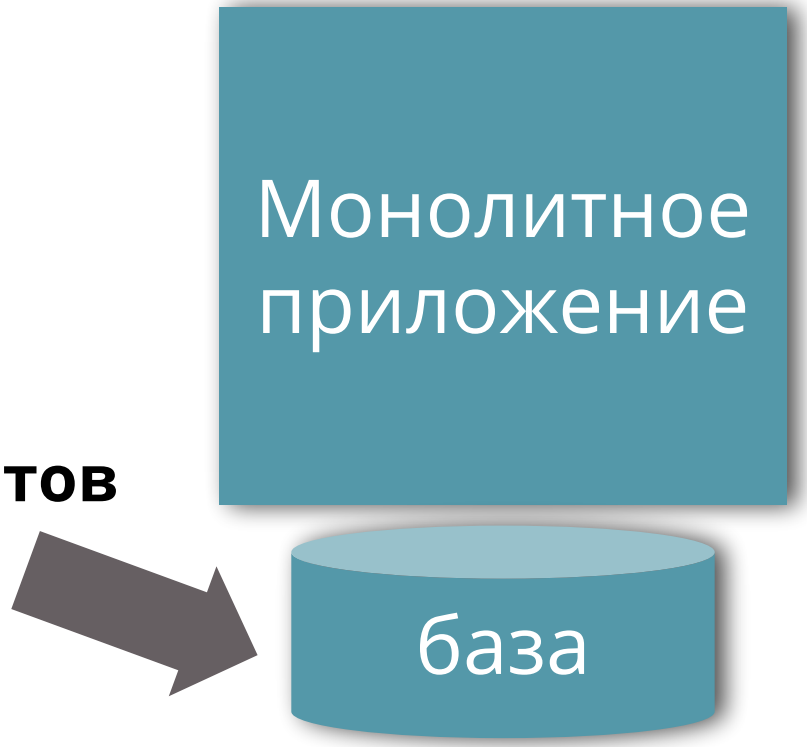
Проперти пропустили

3. Т.е. увеличивается вероятность ошибки при деплое на ci

4. Эффективная очистка данных

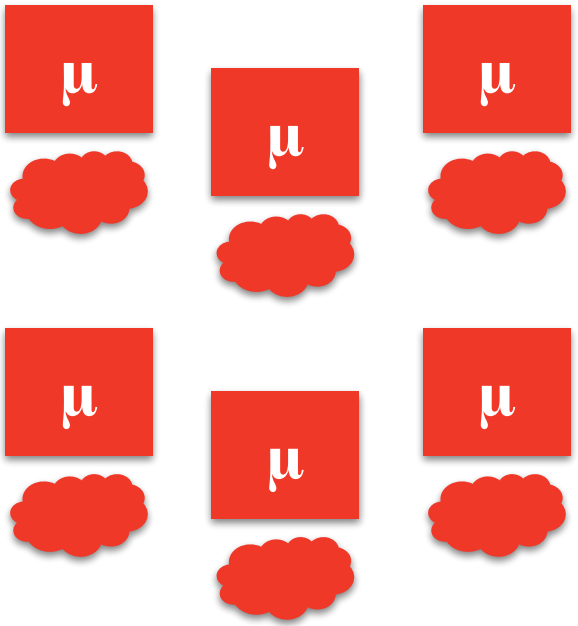
Чистить тут

Без вариантов



4. Эффективная очистка данных

Уже есть варианты!



5. Анализ падения теста сложнее

План доклада

1. Поддержка HTTP клиентов
2. Подготовка данных разнесенных во времени
3. Проверка готовности environment для тестов
4. Экономная очистка тестовых данных
5. Сбор логов при падении теста:
    - QA логи
    - DEV логи

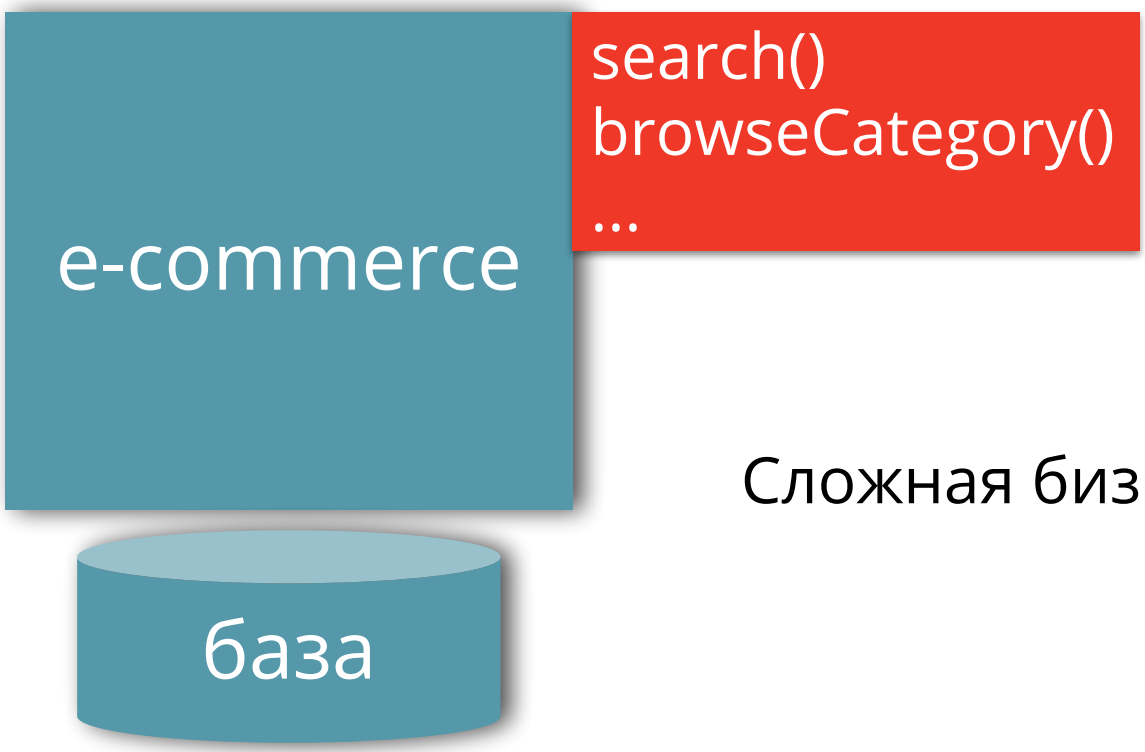
Имплементация решений на

- QA: Java, Cucumber-JVM (+ Spring)

- Dev: Spring Boot экосистема

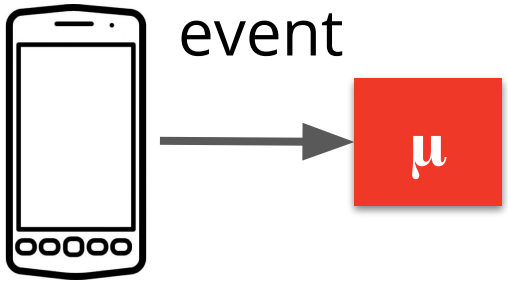
Все решения можно найти на гитхабе

О прошлом проекте

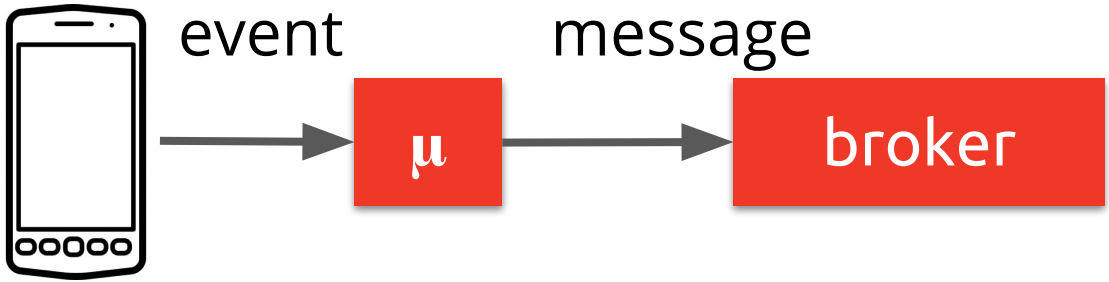


Сложная бизнес логика

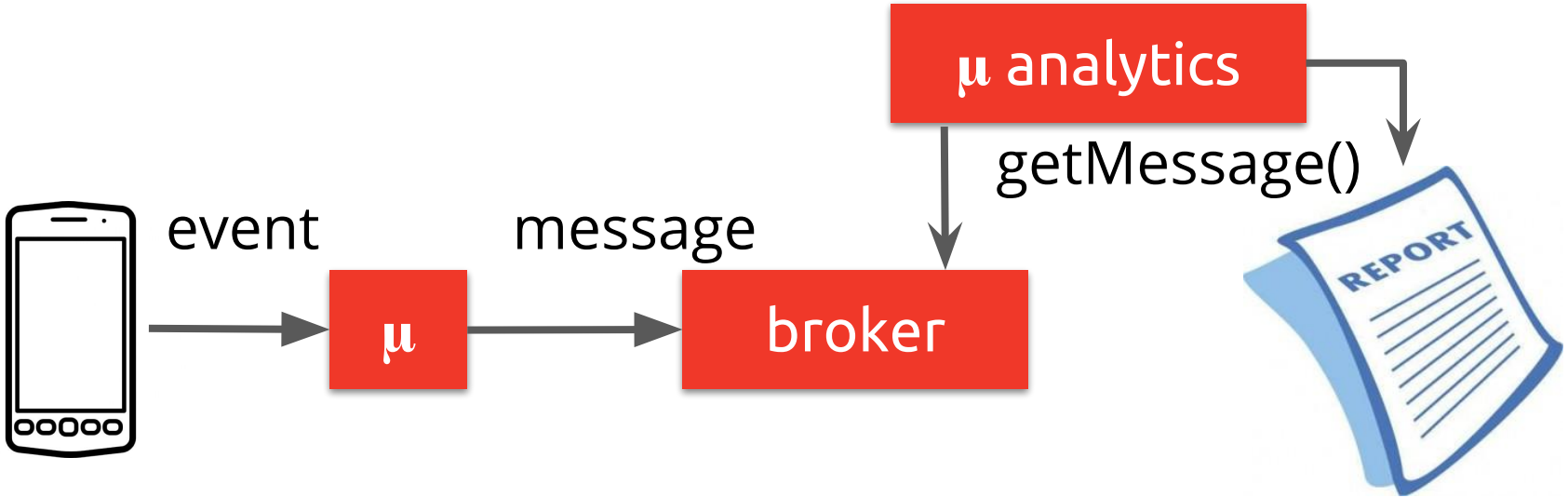
Микросервисный проект



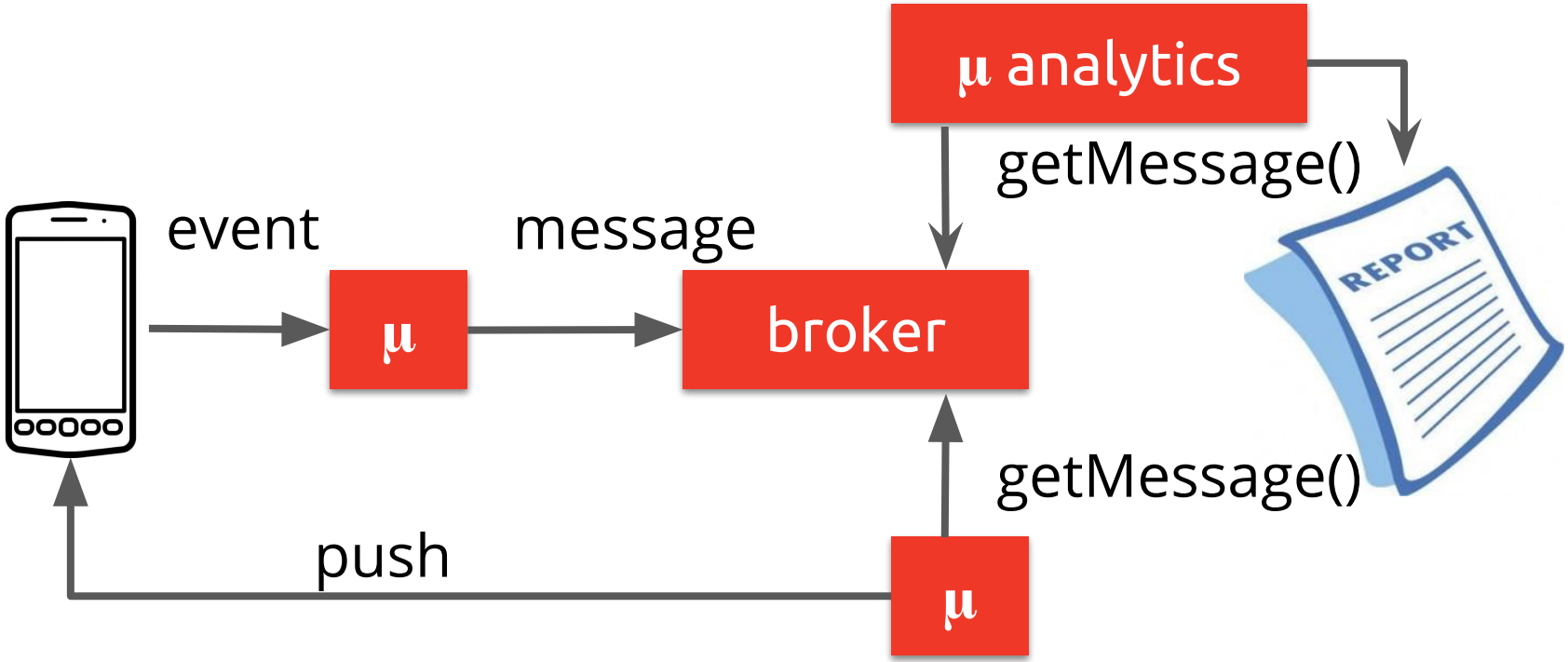
Микросервисный проект



Микросервисный проект



Микросервисный проект



План доклада

1. **Поддержка HTTP клиентов**
2. Подготовка данных разнесенных во времени
3. Проверка готовности environment для тестов
4. Экономная очистка тестовых данных
5. Сбор логов при падении теста:
    - QA логи
    - DEV логи

Объем HTTP клиентов

Было раньше	Стало сейчас
Один клиент	~30 клиентов
Мало методов	~10x30 методов
Редкие изменения	Частые изменения

Ресурсы проектов

Было раньше	Стало сейчас
10 программистов	3 программиста
10 тестировщиков	1 тестировщик

Что хотелось

1. Стоимость **поддержки** клиента должна быть **минимальной**

2. Новая **версия** клиента по каждому **коммиту** в девелоперский код

Что выбрали

https://swagger.io/

Что такое Swagger

Это framework, который позволяет описывать REST API с помощью OpenAPI спецификации.

Swagger инструментарий:

+ Генерация клиентов
+ live документация
+ Swagger UI с возможностью вызывать задеплоеный сервис

Первое знакомство с генерацией

- Девелоперы выдают просто openAPI спецификацию интерфейса

- Этот путь подсмотрели у сторонних команд
  - Например, MailHog

1_client_code_generated_on_qa_side

Как генерирует Swagger

DEV ANNOTATE | OpenAPI JSON | Generate JAX-RS | Create proxy | Call proxy

QA

```
@ApiOperation(nickname = "createUser",
              value = "Create an user",
              response = User.class)
@ApiResponses({
        @ApiResponse(code = 201,
                     message = "Created user")})
public User createUser(@RequestBody User user)
{...}
```

Как генерирует Swagger

DEV ANNOTATE | OpenAPI JSON | Generate JAX-RS | Create proxy | Call proxy

QA

```
"/user" : {
   "post" : {
     "summary" : "Create an user",
     "operationId" : "createUser",
     "parameters" : [ {...
       "schema" : {
         "$ref" : "#/definitions/User"}
     } ],
```

Как генерирует Swagger

Далее

swagger-codegen-maven-plugin
+
dep: connect-openapi-jersey2

Генерятся JAX-RS интерфейсы по JSON файлу

Как генерирует Swagger


```
@Path("/user")
public interface UserService {

    @POST
    public User createUser(User body);
```

Как генерирует Swagger

DEV ANNOTATE | OpenAPI JSON | Generate JAX-RS | Create proxy | Call proxy

QA

```
public class UserJerseyApi {
    private WebTarget webTarget = client
            .target("http://localhost:8080");

    public UserService getUserApi() {
        return WebResourceFactory
                .newResource(UserService.class,
                        webTarget);}
```

Что сразу не понравилось

- Дистрибуция JSON файлов!
- Нет клиента как артефакта
  - Нельзя переиспользовать результат генерации
- Постоянная перегенерация JAX-RS интерфейсов на стороне QA кода

Что захотелось еще

1. Стоимость **поддержки** клиента должна быть **минимальной**
2. Новая **версия** по каждому **коммиту** в девелоперский код

3. Чтобы итоговый клиент был “официальным”:
    a. Использовать в тестировании
    b. Использовать в имплементации

Чтобы это был **отдельный артефакт**

Второе знакомство с генерацией

- Классический путь генерации клиента
- Более привычный в мире разработки

Как генерирует Swagger (ver.2)

DEV ANNOTATE	OpenAPI JSON	Generate sources	Build sources	Use client in QA, DEV

```
@ApiOperation(nickname = "createUser",
              value = "Create an user",
              response = User.class)
@ApiResponses({
        @ApiResponse(code = 201,
                     message = "Created user")})
public User createUser(@RequestBody User user)
{...}
```

Как генерирует Swagger (ver.2)

DEV ANNOTATE	OpenAPI JSON	Generate sources	Build sources	Use client in QA, DEV

Далее

swagger-maven-plugin

генерит JSON с openApi спецификацией endpoints на основании аннотаций

Как генерирует Swagger (ver.2)

DEV ANNOTATE	OpenAPI JSON	Generate sources	Build sources	Use client in QA, DEV

```
"/user" : {
   "post" : {
     "summary" : "Create an user",
     "operationId" : "createUser",
     "parameters" : [ {...
       "schema" : {
         "$ref" : "#/definitions/User"}
     } ],
```

Как генерирует Swagger (ver.2)

DEV ANNOTATE	OpenAPI JSON	Generate sources	Build sources	Use client in QA, DEV

Далее

swagger-codegen-maven-plugin

генерит исходники клиента на Java в виде Maven проекта

Как генерирует Swagger (ver.2)

DEV ANNOTATE	OpenAPI JSON	Generate sources	Build sources	Use client in QA, DEV

```
public class UsersApi {
  private ApiClient apiClient;

  public User createUser(User body) {
    return apiClient.invokeAPI(..., "POST",
body,...);
}
```

Как генерирует Swagger (ver.2)

DEV ANNOTATE	OpenAPI JSON	Generate sources	Build sources	Use client in QA, DEV

Далее

GOTO target/ с исходниками клиента

> mvn clean install

Загрузить артефакт в **Nexus**

Как генерирует Swagger (ver.2)

DEV ANNOTATE	OpenAPI JSON	Generate sources	Build sources	Use client in QA, DEV

```
<dependency>
    <groupId>io.swagger</groupId>
    <artifactId>swagger-java-client</artifactId>
    <version>1.0.0</version>
</dependency>
```

Положительные заметки

- Группировка endpoint по контроллерам сохраняется в классах Swagger клиента

Положительные заметки

- Группировка endpoint по контроллерам сохраняется в классах Swagger клиента
- Клиент может работать на Jersey, OkHttp, Feign

Положительные заметки

- Группировка endpoint по контроллерам сохраняется в классах Swagger клиента
- Клиент может работать на Jersey, OkHttp, Feign
- Стабильно работает

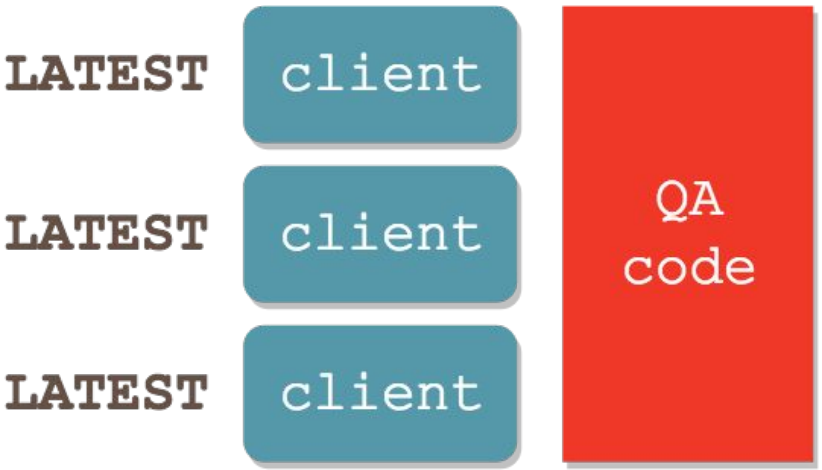
На что обратить внимание

- Версию клиента при сборке надо инкрементировать, т.к. Swagger всегда выдает 1.0.0

На что обратить внимание

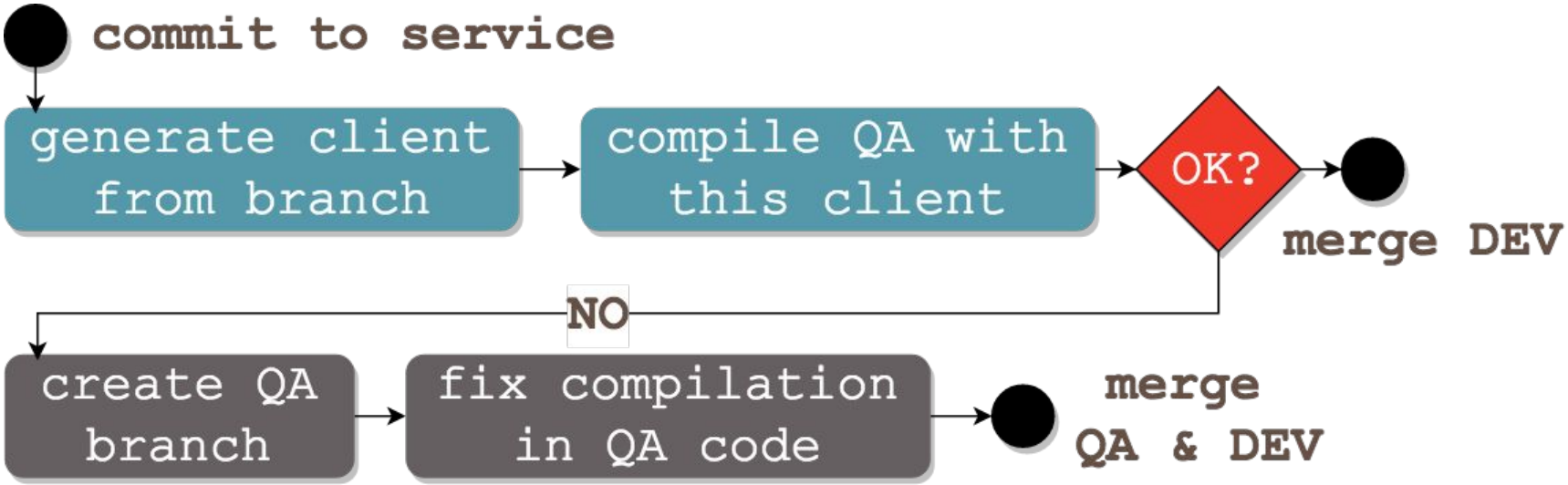
- Версию клиента при сборке надо инкрементировать, т.к. Swagger всегда выдает 1.0.0
- На компиляцию QA кода на CI при множестве клиентов

Компиляция QA кода на CI



1. Клиент генерится по **каждому комиту** в сервис
2. **Версии** клиентов появляются очень **часто**
3. => зависимость от LATEST версии
4. На CI при загрузке последней версии может **ломаться компиляция** QA кода

Доп. шаги в dev Merge Request



Итоги по данному подходу

- Swagger показал себя стабильным решением для генерации клиентов
- Подход работал для ~30 микросервисов

План доклада

1. Поддержка HTTP клиентов
2. **Подготовка данных разнесенных во времени**
3. Проверка готовности environment для тестов
4. Экономная очистка тестовых данных
5. Сбор логов при падении теста:
    - QA логи
    - DEV логи

Бывают такие тестовые сценарии

- Подготовить данные за **позапрошлый** месяц
  - Подготовить данные за **прошлый** месяц
    - Подготовить данные за **данный** месяц

Проверить систему с такими данными

Пример сценариев для e-commerce

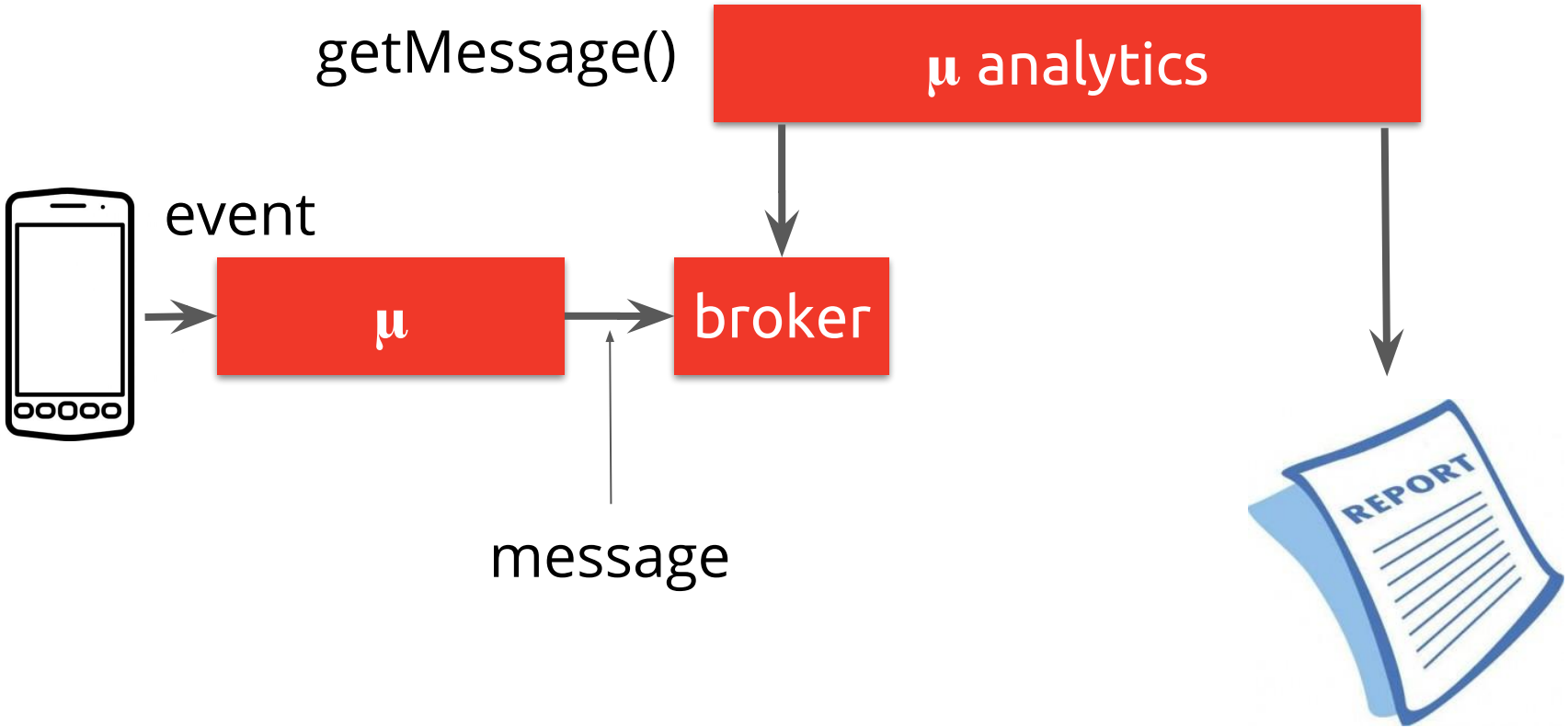
- Скидки на продукты, зависящие от времени
- Продукты с разными стартами продаж
- Категории с продуктами, видимые на сайте только с определенного времени

Как решалось для монолита

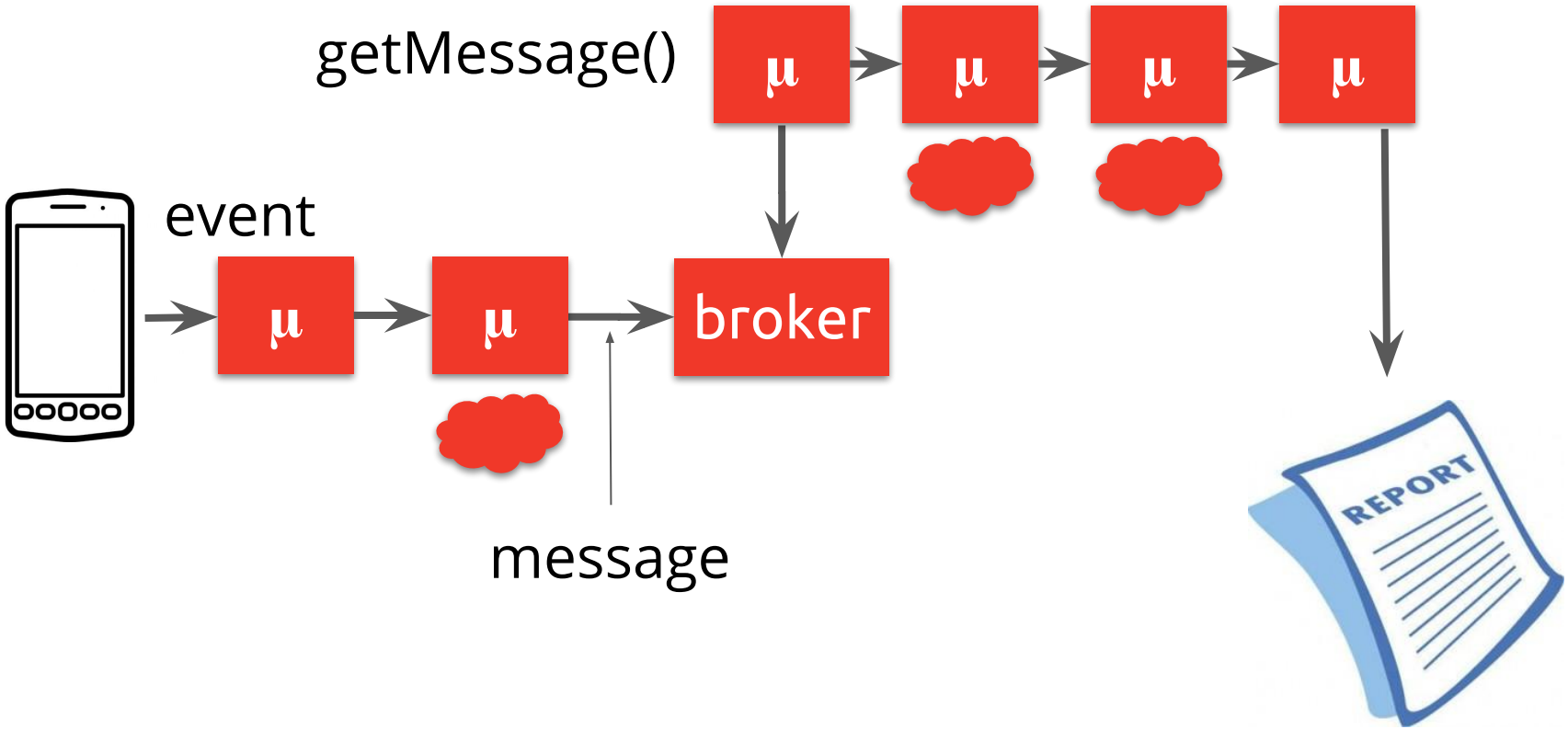
DbUnit инструмент:

- XML файлы с данными
- Прямая загрузка в базу
- Alias для дат:
  - DAY-1
  - WEEK-1

Пример для location-based систем



Пример для location-based систем



Специфика данных в системе

- Данные распределены по многим хранилищам
- Данные связаны между собой

Что стало очень дорого

- Завязываться на имплементацию хранилища
- Загружать данные напрямую в хранилище

Что стало очень дорого

- Завязываться на имплементацию хранилища
- Загружать данные напрямую в хранилище

Теперь готовить данные только через API

Как нагенерить данных через API для прошлого месяца?

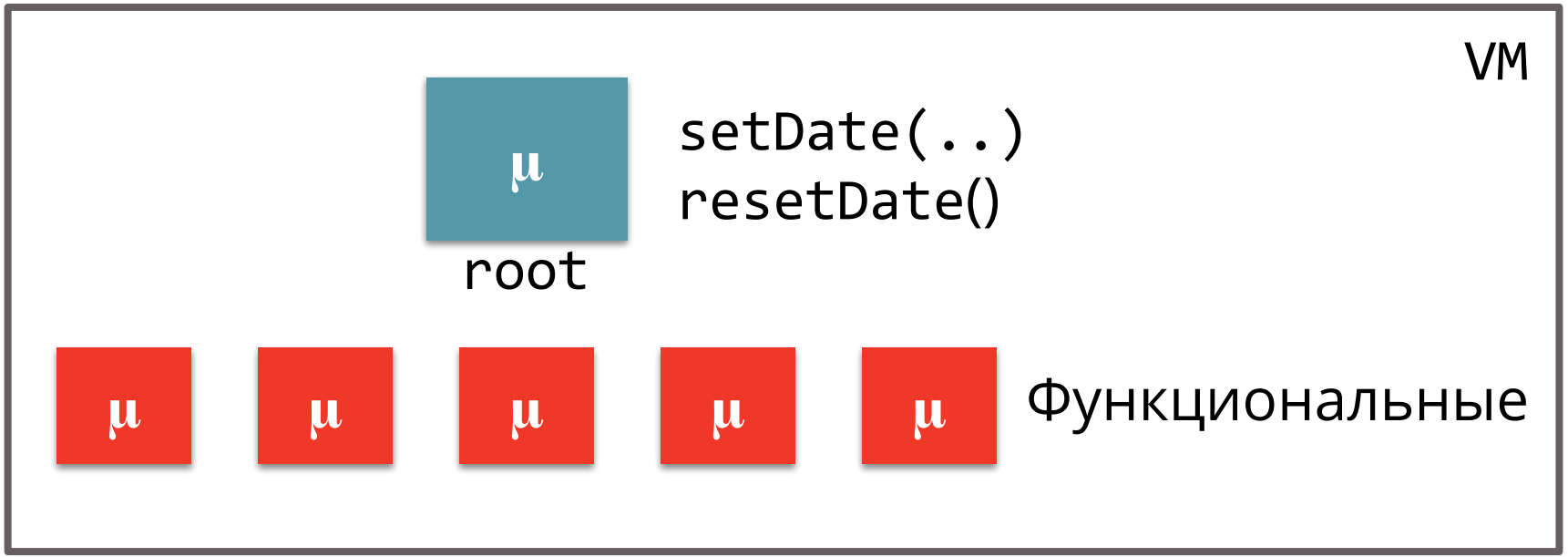
Эксперимент

1. Зайти на виртуалку
2. > `date -s` ДАТА_ВРЕМЯ
3. Послать event в систему
4. Проверить с какой датой везде все сохранилось

Эксперимент

1. Зайти на виртуалку
2. > `date -s` ДАТА_ВРЕМЯ
3. Послать event в систему
4. Проверить с какой датой везде все сохранилось

В итоге решение



Time-machine урезанный код

```
public void setDate(String date) {
   String[] command = {"date", "-s", date};
   execCmd(command);
}
private void execCmd(String[] command) {
   Runtime.getRuntime().exec(command);
}
```

Сценарий выглядит так

- *setDate("MONTH-2")*
- Нагенерить данных
- *setDate("MONTH-1")*
- Нагенерить данных
- *resetDate()*
- скачать отчет (другие проверки)

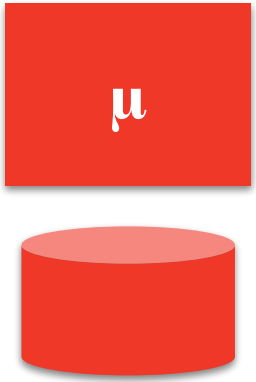
Time-machine итоги

- По performance операция переключения времени дешевая
- Стабильный подход

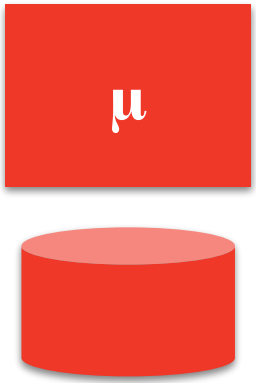
План доклада

1. Поддержка HTTP клиентов
2. Подготовка данных разнесенных во времени
3. **Проверка готовности environment для тестов**
4. Экономная очистка тестовых данных
5. Сбор логов при падении теста:
    - QA логи
    - DEV логи

Поговорим про деплоймент на CI

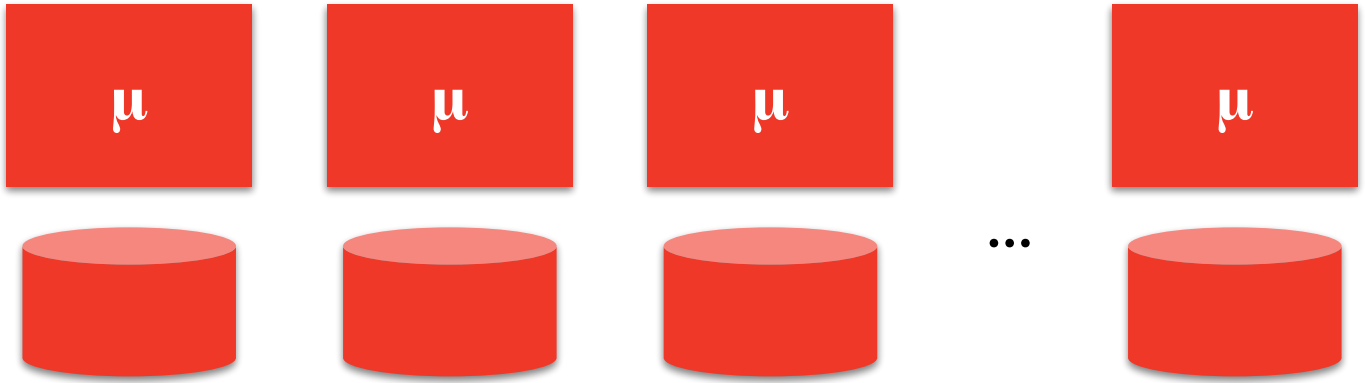


Что может пойти не так

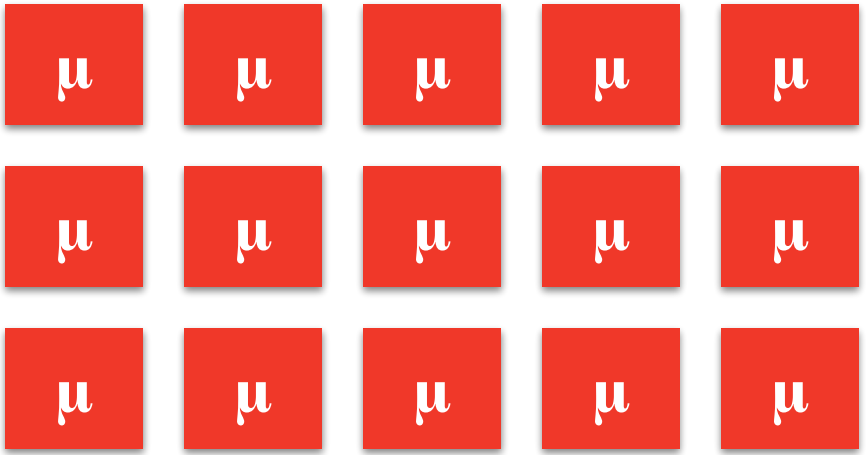


- Забыли про сервис
- Старая схема хранилища
- Проперти
- Память
- И т.д.

Вероятность ошибки больше

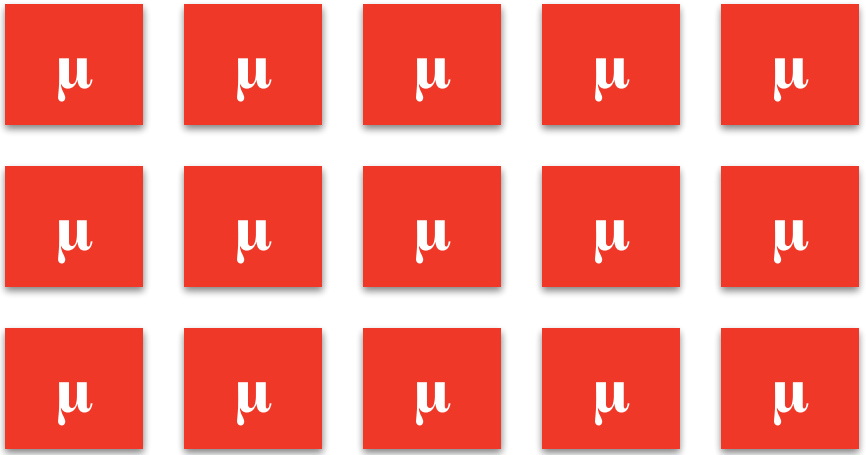


Это время прогона джобы



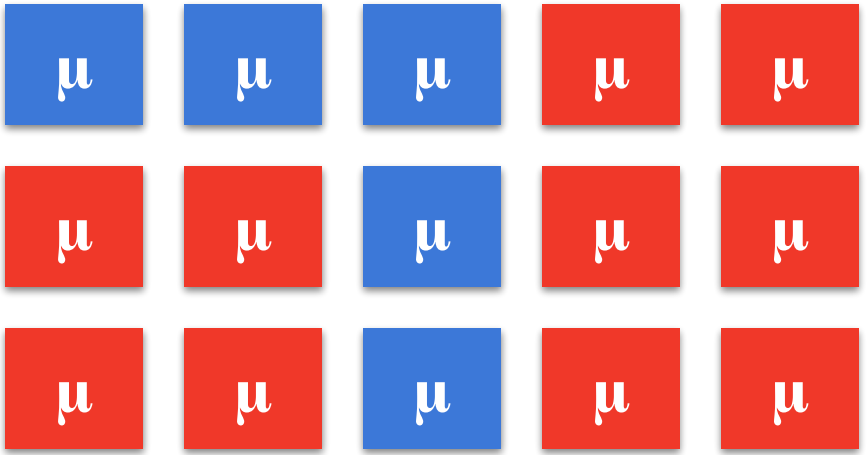
Это время прогона джобы

Опросить “здоровье” всех микросервисов в начале джобы



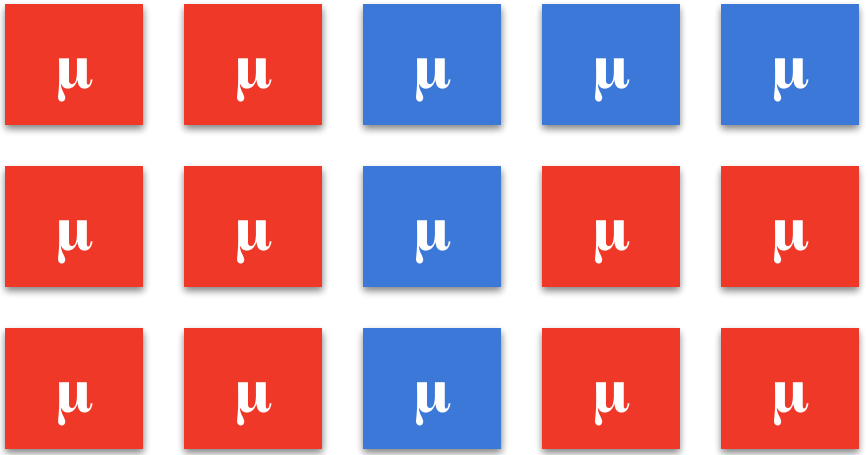
Это время прогона джобы

А лучше перед тестом опрашивать только релевантные сервисы

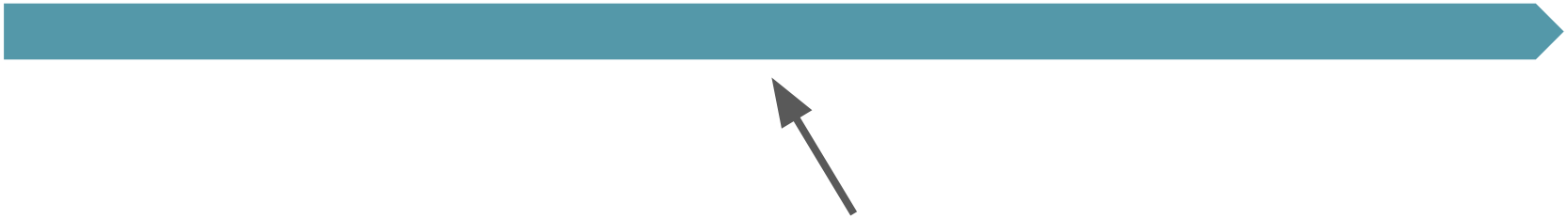


Это время прогона джобы

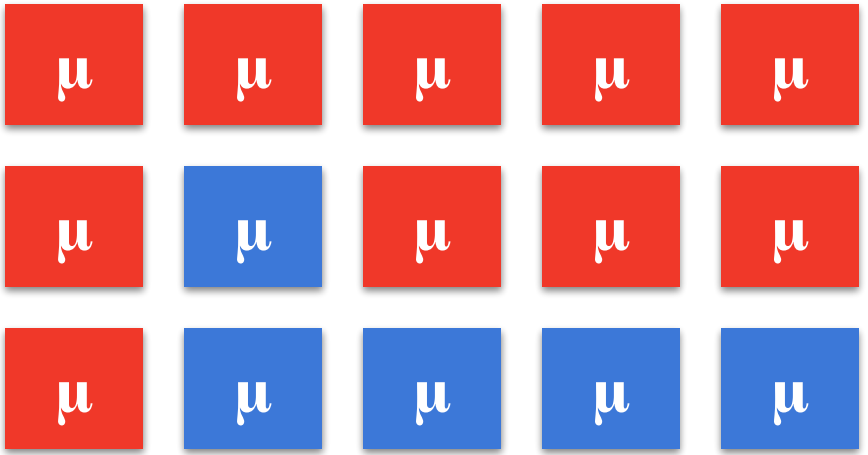
А лучше перед тестом опрашивать только релевантные сервисы



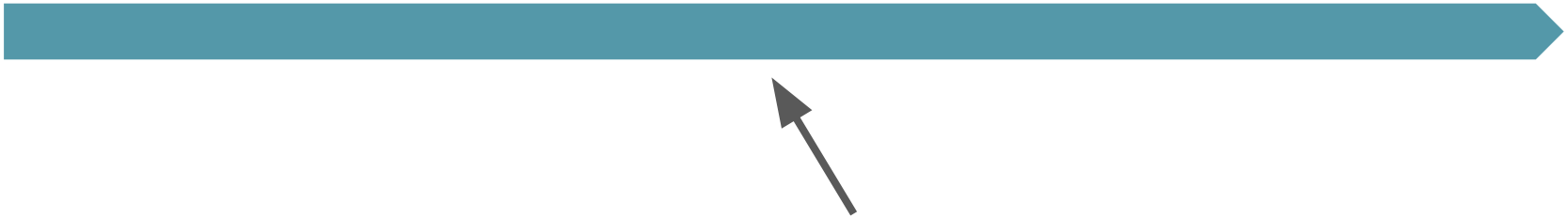
Это время прогона джобы



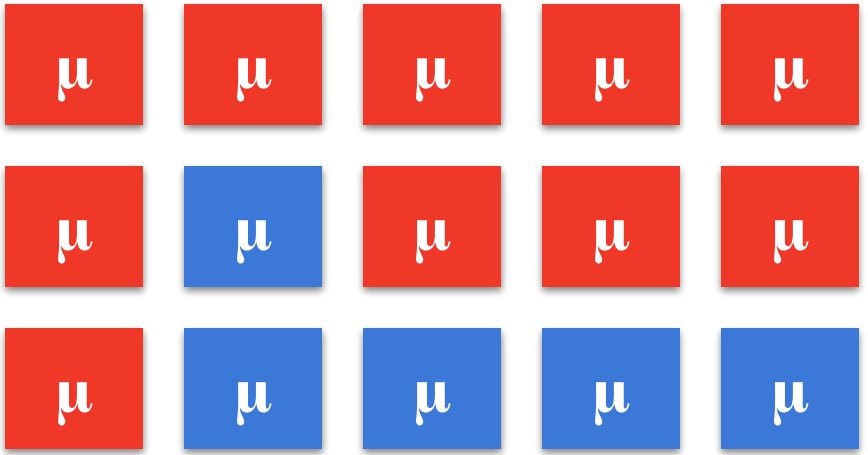
А лучше перед тестом опрашивать только релевантные сервисы



Это время прогона джобы

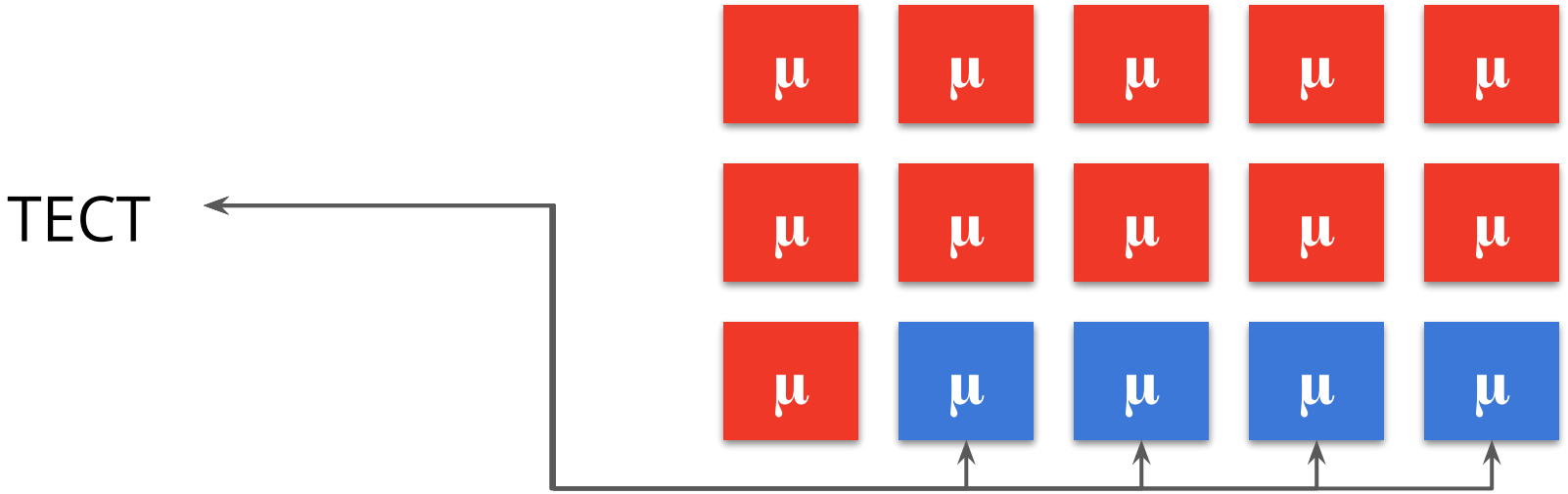


→ Т.е. каждый тест будет уверен, что для него все задеплоено правильно



Промежуточная задача

Ассоциировать тест с микросервисами, которые он затрагивает.



ТЕСТ

Последовательность действий

Сохранить список микросервисов в сундучок

Опросить здоровье микросервисов

Запустить тест

Как?

Сохранить список микросервисов в сундучок

@μ_1 @μ_2 @μ_4 @μ_6
Scenario: Проверка отсылки сообщения
When Подхожу к кафе
Then Получаю пригласительную скидку

Что будет сундучком

```
public class Сундучок {
   private Set<String> tags;

   public void init(Scenario scenario) {
      tags = Sets.newHashSet(
               scenario.getSourceTagNames());
   }
}
```

Заполняем сундучок

Гарантируем, что это будет вначале каждого теста

```
@Before(order = 1)
public void saving(Scenario scenario) {
   сундучок.init(scenario);
}
```

Запуск проверки здоровья

Для отчетов шаги разносим на два хука

```
@Before(order = 2)
public void checkHealth() {
   healthChecker.check(сундучок.getTags());
}
```

Проверка здоровья

```
public void check(Set<String> tags) {
   ExecutorService executor = ...;
   Runnable task = () -> {
      getServiceLinks(tags)
         .forEach(this::checkStatus);
   };
   executor.submit(task);
}
```

Проверка здоровья

```
public void check(Set<String> tags) {
   ExecutorService executor = ...;
   Runnable task = () -> {
      getServiceLinks(tags)
         .forEach(this::checkStatus);
   };
   executor.submit(task);
}
```

Преобразуем теги в линки

```
private List<String> getServiceLinks(Set<String> tags) {
    for (String tag : tags) {
        switch (tag) {
            case "@µ_1":
                urls.add("http://localhost:8080/health");
                break;
            case "@µ_2":
                urls.add("http://localhost:9090/health");
                break; } }
    return urls;
}
```

Ранее подготовленный набор значений

Проверка здоровья

```
public void check(Set<String> tags) {
   ExecutorService executor = ...;
   Runnable task = () -> {
      getServiceLinks(tags)
         .forEach(this::checkStatus);
   };
   executor.submit(task);
}
```

```
private void checkStatus(String healthLink) {
    try {
        String response = Request.Get(healthLink)
            .execute().returnContent().asString();
        assertThat(response).contains("UP");
    } catch (IOException e) {
        log.error("Connection to Health service refused", e);
        System.exit(1);
    } catch (ComparisonFailure e) {
        log.error("Health status is not UP", e);
        System.exit(1);
    }
}
```

```
private void checkStatus(String healthLink) {
    try {
        String response = Request.Get(healthLink)
            .execute().returnContent().asString();
        assertThat(response).contains("UP");
    } catch (IOException e) {
        log.error("Connection to Health service refused", e);
        System.exit(1);
    } catch (ComparisonFailure e) {
        log.error("Health status is not UP", e);
        System.exit(1);
    }
}
```

```
private void checkStatus(String healthLink) {
    try {
        String response = Request.Get(healthLink)
            .execute().returnContent().asString();
        assertThat(response).contains("UP");
    } catch (IOException e) {
        log.error("Connection to Health service refused", e);
        System.exit(1);
    } catch (ComparisonFailure e) {
        log.error("Health status is not UP", e);
        System.exit(1);
    }
}
```

Комментарии к подходу

- if кол-во µсервисов ↑ ⇒ `getServiceLinks(`**`tags`**`)` сложнее
- При многопоточных тестах сундучок должен иметь нужный scope (thread, scenario)

Итоги по проверке env

- **Fail fast** механизм => Jenkins **не** выполняет **пустую** работу
- **Опрашиваются** только **необходимые** сервисы, а не весь кластер

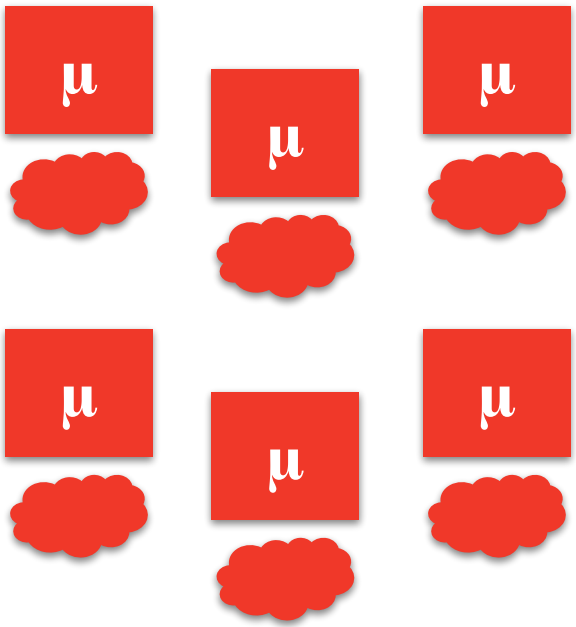
План доклада

1. Поддержка HTTP клиентов
2. Подготовка данных разнесенных во времени
3. Проверка готовности environment для тестов
4. **Экономная очистка тестовых данных**
5. Сбор логов при падении теста:
    - QA логи
    - DEV логи

Как было у монолита

Вариантов не так много

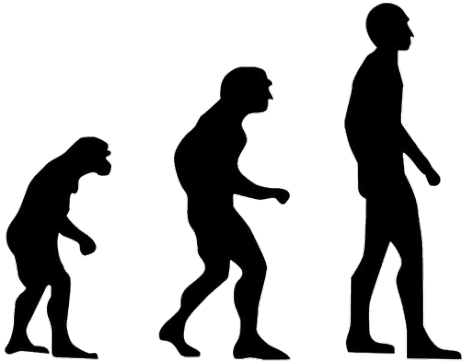
Как стало в микросервисах



- Больше **соединений** с хранилищами
- **Поддержка** кода для очистки данных сложнее

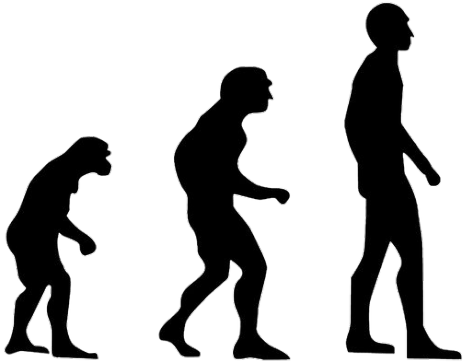
Как быть?

Эволюция решения



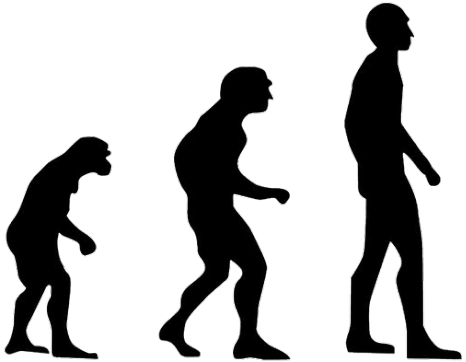
Перед каждой подготовкой данных чистить все хранилища

Эволюция решения



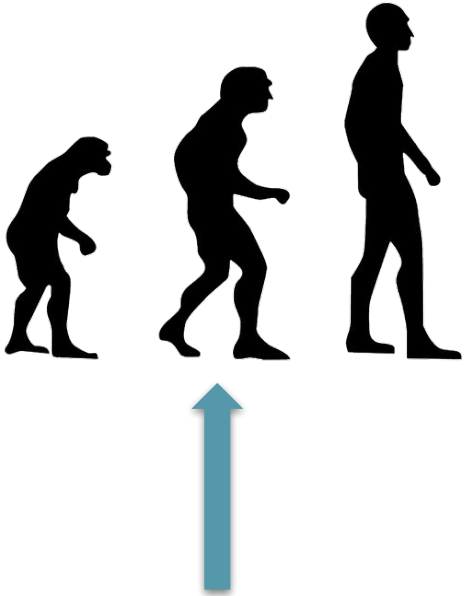
- Топорно, но дешево поддерживать

Эволюция решения



- Полагаешь, что все хранилища всегда присутствуют
- По времени дороже

Эволюция решения

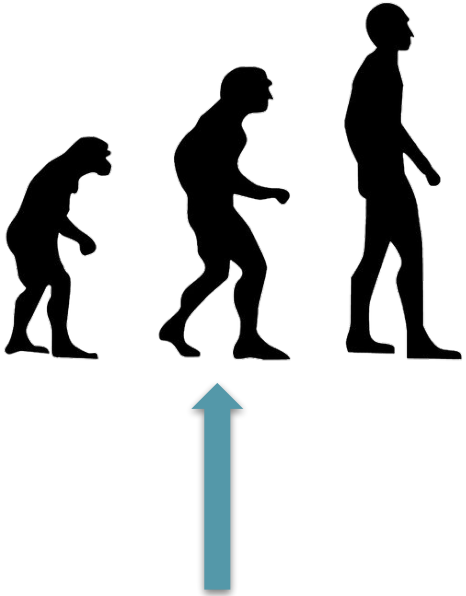


Чистить только

определенные хранилища

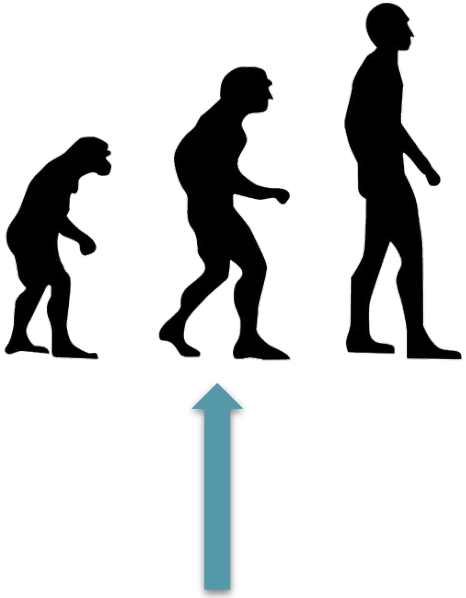
при подготовке данных

Эволюция решения



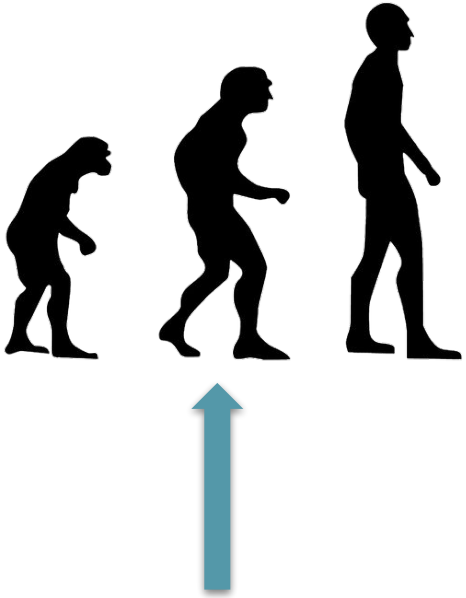
```
@Before("@delete1")
public void delete1(){
    mysqlCleanup.clean();
    cassandraCleanup.clean();
    redisCleanup.clean();
}

@Before("@delete2")
public void delete2(){
    mysqlCleanup.clean();
    redisCleanup.clean();
    elasticCleanup.clean();
}
```

Эволюция решения



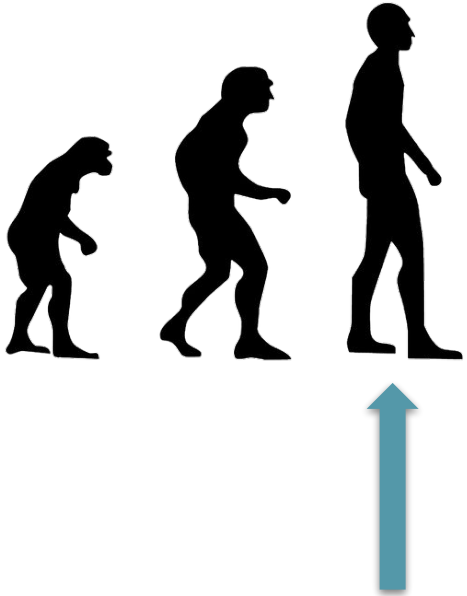
- Экономим время

Эволюция решения



- Дорогая поддержка кода
- Для теста хардкодишь шаги для очищения

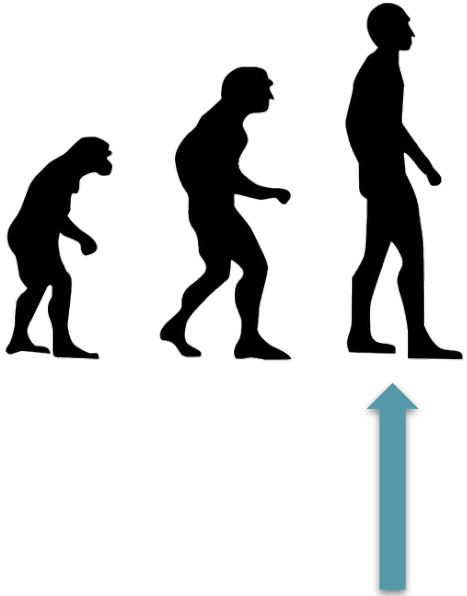
Эволюция решения



Чистить только определенные хранилища при подготовке данных

- Иметь абстрактный метод deleteAll()

Эволюция решения



```
public class CleanUpExecutor {

    Сундучок сундучок;

    Set<Runnable> toExecute;

    public void deleteAll() {..}

}
```

Что внутри deleteAll(). часть1

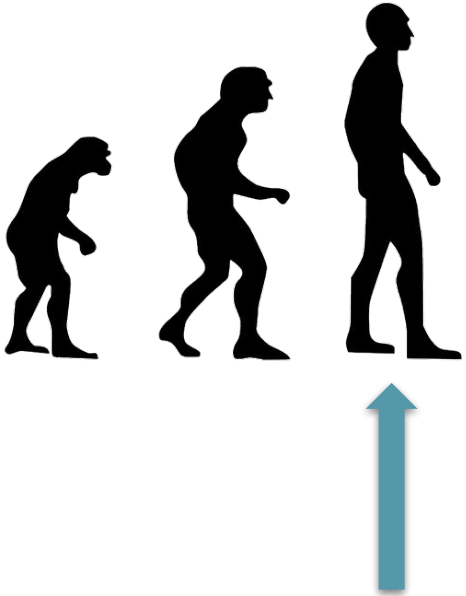
```
for (String tag : сундучок.getTags()) {
    switch (tag) {
        case "@µ_1":
            toExecute.add(cleanCassandra);
            break;
        case "@µ_2":
            toExecute.add(cleanRedis);
            break;
    }
    ...
}
```

Ранее подготовленный набор значений

Что внутри deleteAll(). часть2

```
toExecute.forEach (p -> {
   try {
      p.run();
   } catch (Throwable e) {
      log.error("Unable to clean storage", e);
      System.exit(1);
   }
});
toExecute.clear();
```

Эволюция решения



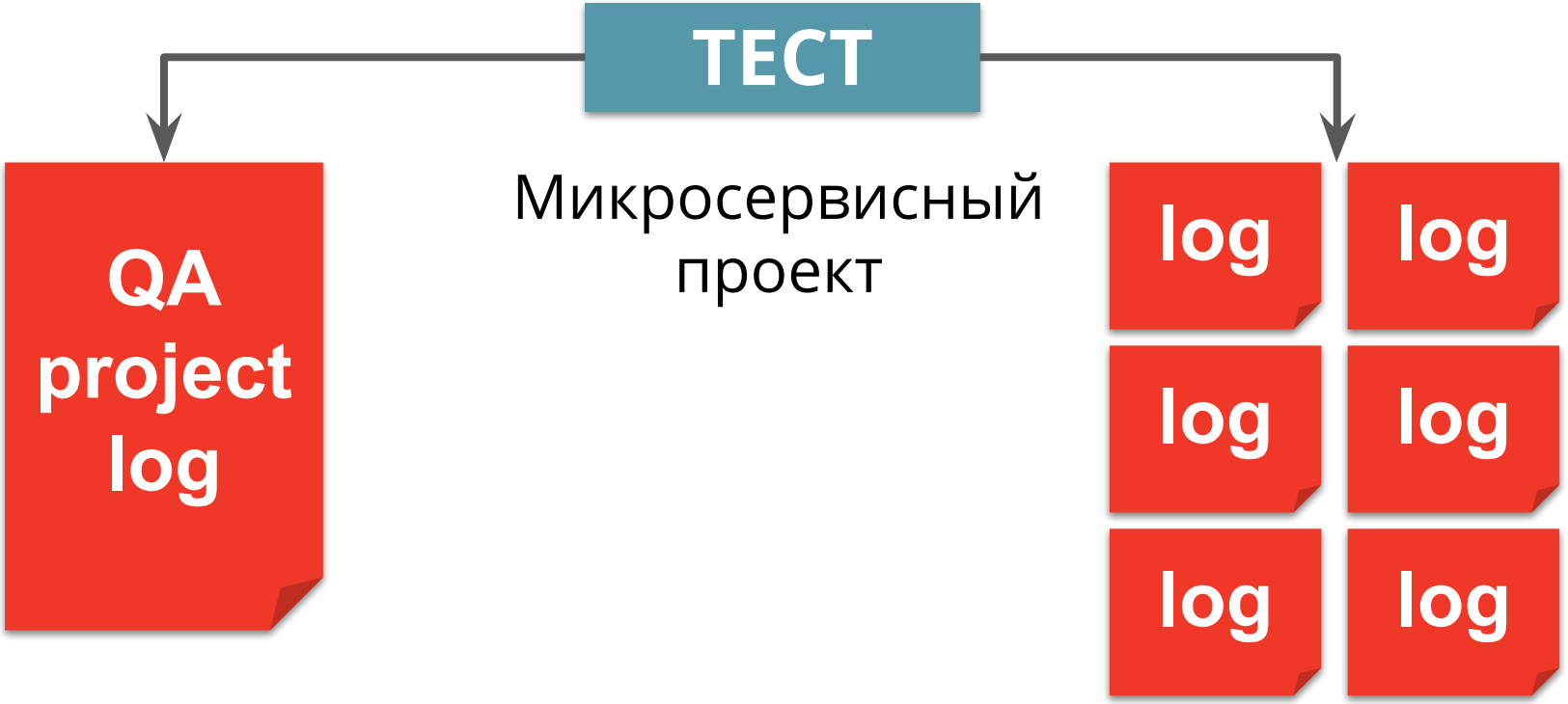
- **Не** чистим **ненужное** - экономим время
- **Поддержка** очистки в целом **не дорогая**
- **Переиспользую** теги из сундучка

План доклада

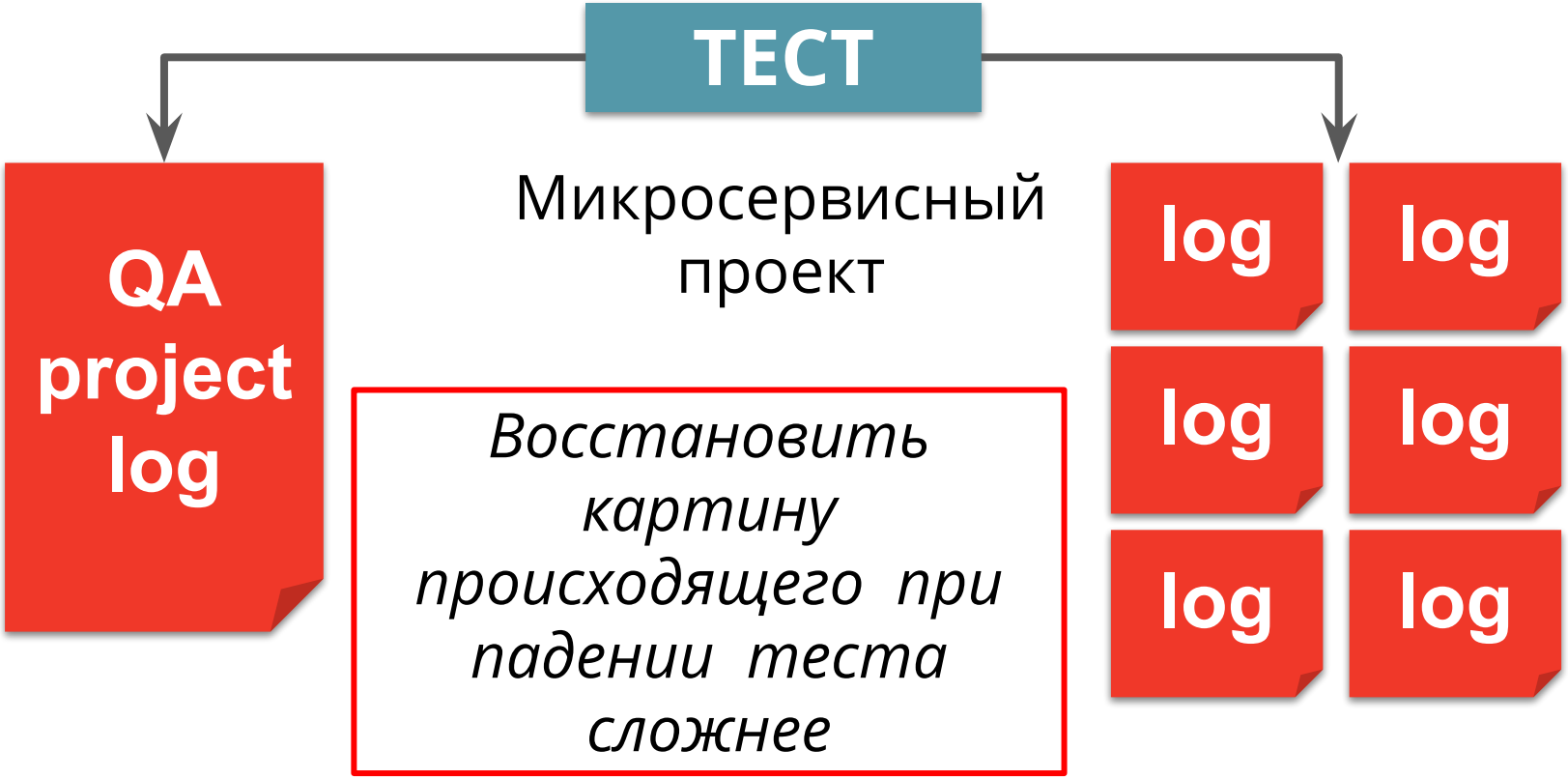
1. Поддержка HTTP клиентов
2. Подготовка данных разнесенных во времени
3. Проверка готовности environment для тестов
4. Экономная очистка тестовых данных
5. **Сбор логов при падении теста:**
    - **QA логи**
    - **DEV логи**

Какие логи пишутся при тесте

Какие логи пишутся при тесте



Какие логи пишутся при тесте

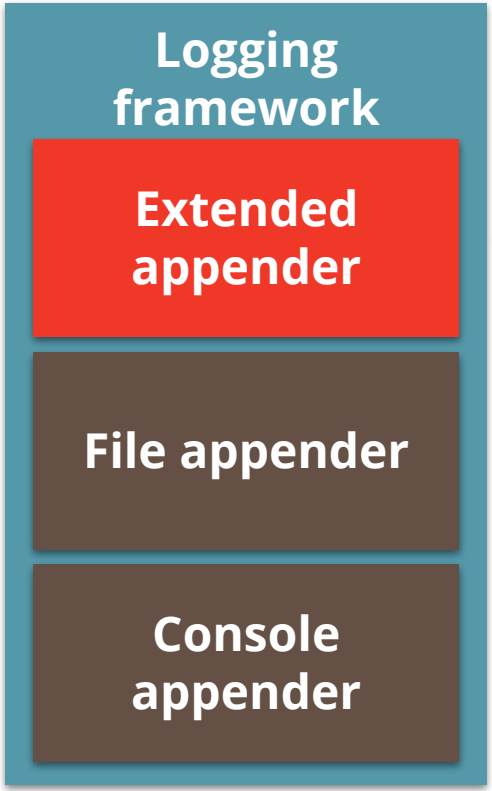


Задача 1

- Сохранить **детальную** последовательность **действий** теста со стороны **QA** кода

⇒ т.е. кусок QA лога

Описание подхода



Расширяем appender

```
public class CustomFileAppender extends
    FileAppender<LoggingEvent> {

    @Override
    public void doAppend(LoggingEvent eventObject) {
        Сундучок.addLogEntry(eventObject);
    }
}
```

Как собираем логи в сундучок

```
public void init(Scenario scenario) {
   insideOfFeature = true; }

public static void addLogEntry(LoggingEvent event) {
   if (insideOfFeature)
       accumulatedLogs.add(event);   }

public void reset() {
   insideOfFeature = false;
   accumulatedLogs.clear(); }
```

Используем сохраненные QA логи

```
@After(order = 1)
public void publishQaLogs(Scenario scenario) {
    if (scenario.failed()) {
        recordInternalLogs(scenario);
    }
    сундучок.reset();
}

private void recordInternalLogs(Scenario scenario) {
    scenario.write(сундучок.getAccumulatedLogs());
}
```

Пример без логов

▼ @deleteAll **Feature**: create user

mvn clean test -Dcucumber.options="--tags "

▼ **Scenario**: check user is not null

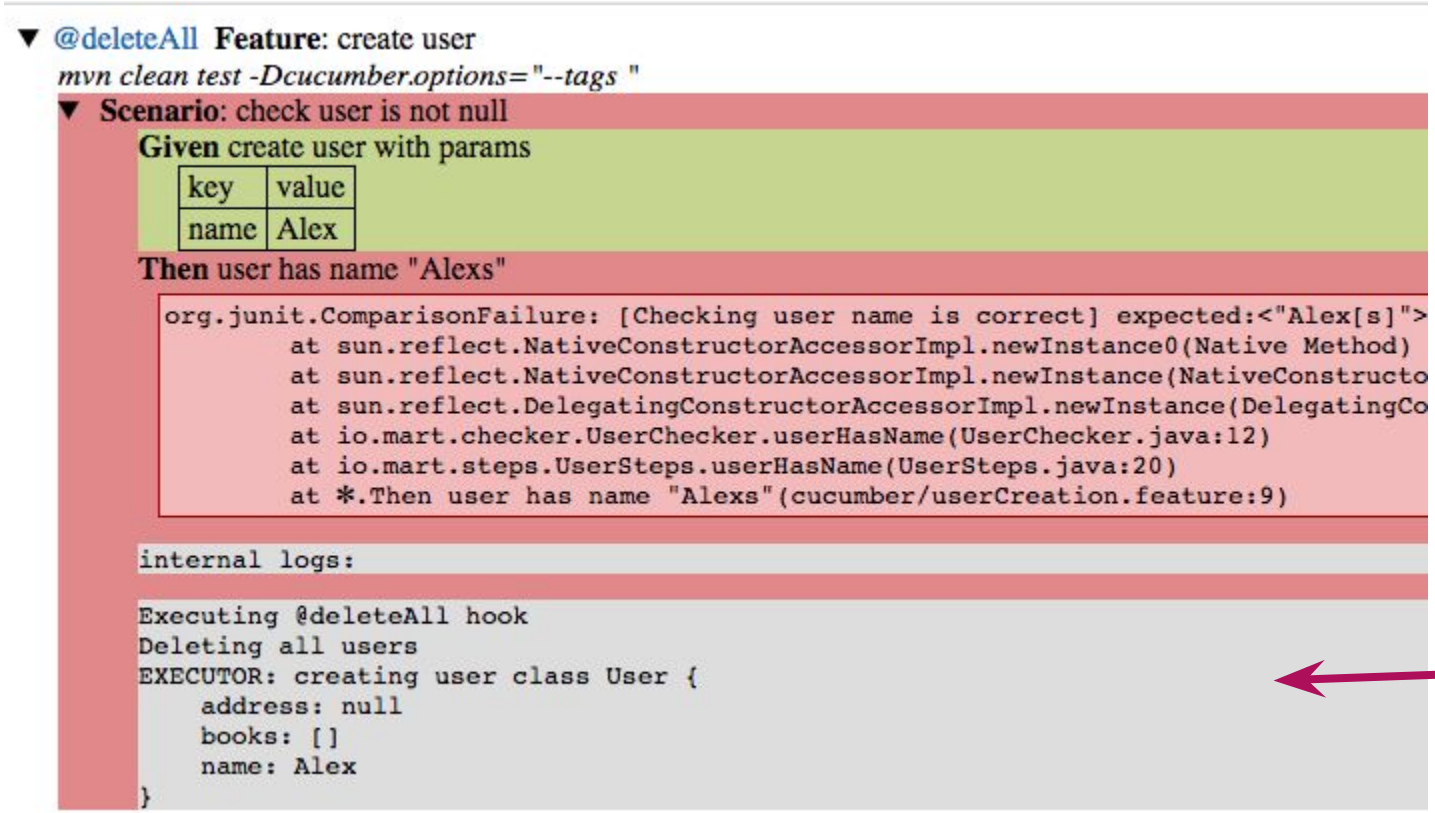
Given create user with params

key	value
name	Alex

Then user has name "Alexs"

```
org.junit.ComparisonFailure: [Checking user name is correct] expected:<"Alex[s]">
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructo
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingCo
        at io.mart.checker.UserChecker.userHasName(UserChecker.java:12)
        at io.mart.steps.UserSteps.userHasName(UserSteps.java:20)
        at *.Then user has name "Alexs"(cucumber/userCreation.feature:9)
```

Пример с логами



▼ @deleteAll **Feature**: create user

mvn clean test -Dcucumber.options="--tags "

▼ **Scenario**: check user is not null

Given create user with params

key	value
name	Alex

Then user has name "Alexs"

```
org.junit.ComparisonFailure: [Checking user name is correct] expected:<"Alex[s]">
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructo
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingCo
        at io.mart.checker.UserChecker.userHasName(UserChecker.java:12)
        at io.mart.steps.UserSteps.userHasName(UserSteps.java:20)
        at *.Then user has name "Alexs"(cucumber/userCreation.feature:9)
```

```
internal logs:

Executing @deleteAll hook
Deleting all users
EXECUTOR: creating user class User {
    address: null
    books: []
    name: Alex
}
```

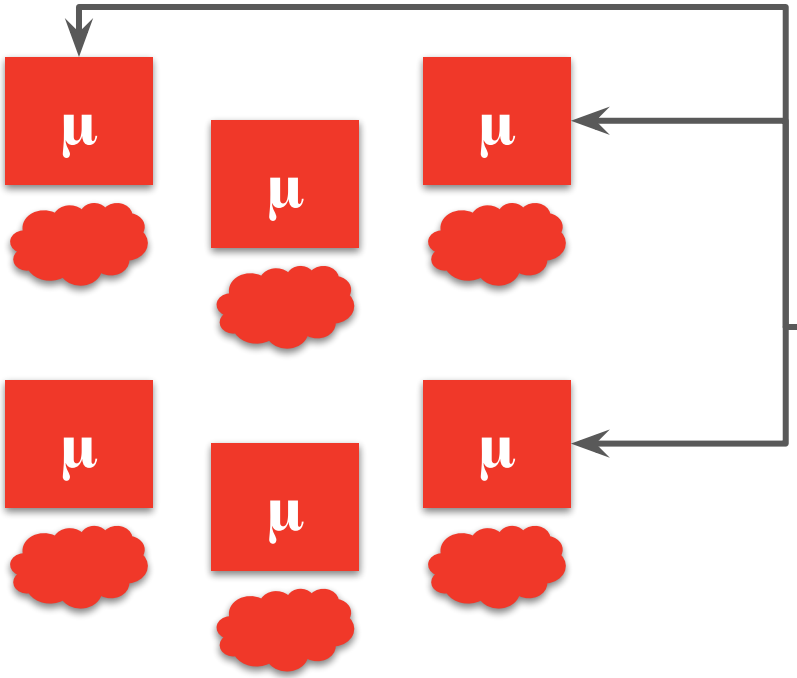
Логи QA проекта

Задача 2

- Сохранить последовательность **действий** микросервисов за **время** работы **теста**

⇒ т.е. куски DEV логов

Собрать DEV логи за Δt



Вариант 1 (Dev code: Spring Boot)

Log collector

Вариант 1 (Dev code: Spring Boot)

1. getTags()

Log collector

Вариант 1 (Dev code: Spring Boot)

1. getTags()

Log collector

2. GET /logfile

μ_1
/logfile

μ_4
/logfile

μ_7
/logfile

spring boot

Вариант 1 (Dev code: Spring Boot)

1. getTags()

Log collector

2. GET /logfile

3. extract

List<String>

μ_1
/logfile

μ_4
/logfile

μ_7
/logfile

spring boot

Вариант 1 (Dev code: Spring Boot)

1. getTags()

Log collector

2. GET /logfile

μ_1 /logfile

μ_4 /logfile

μ_7 /logfile

3. extract

List<String>

4. append

Scenario

spring boot

Как собираем логи

```
public void recordDevLogs(_ startTime, Set<String> tags) {

    List<String> serviceLinks = getLinks(tags);
    for (String link : serviceLinks) {
        try {
            String s = Request.Get(link).execute()
                        .returnContent()
                        .asString();
            String extractedLog = parse(s, startTime);
        }
    }
...
```

Шаг сбора логов

```
@After(order = 1)
public void collectDevLogs() {
   if (scenario.isFailed()) {
      logCollector.recordDevLogs(
         scenarioStartTime,
         сундучок.getTags());
   }
}
```

Пример без логов

▼ @deleteAll **Feature**: create user

mvn clean test -Dcucumber.options="--tags "

▼ **Scenario**: check user is not null

Given create user with params

key	value
name	Alex

Then user has name "Alexs"

```
org.junit.ComparisonFailure: [Checking user name is correct] expected:<"Alex[s]"> but was:<"Alex[]">
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
        at io.mart.checker.UserChecker.userHasName(UserChecker.java:12)
        at io.mart.steps.UserSteps.userHasName(UserSteps.java:20)
        at ✻.Then user has name "Alexs"(cucumber/userCreation.feature:9)
```

Пример с логами

▼ @deleteAll **Feature**: create user

mvn clean test -Dcucumber.options="--tags "

▼ **Scenario**: check user is not null

Given create user with params

key	value
name	Alex

Then user has name "Alexs"

```
org.junit.ComparisonFailure: [Checking user name is correct] expected:<"Alex[s]"> but was:<"Alex[]">
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
        at io.mart.checker.UserChecker.userHasName(UserChecker.java:12)
        at io.mart.steps.UserSteps.userHasName(UserSteps.java:20)
        at ✻.Then user has name "Alexs"(cucumber/userCreation.feature:9)
```

```
logs from: (http://localhost:8080/logfile) after 2018-03-22T21:32:32.528 - are

2018-03-22 21:32:32.866  INFO 2195 --- [http-nio-8080-exec-10] io.mart.UserRepository                : Users were delete
2018-03-22 21:32:33.067  INFO 2195 --- [http-nio-8080-exec-1] io.mart.UserRepository                 : User was created U
```

Логи, собранные с приложений

Комментарии

- Помнить про часовые пояса на CI и cloud

Альтернативный путь сбора логов

DEV код не на Spring Boot

1. getTags()

Log collector

DEV код не на Spring Boot

1. getTags()

Log collector

2. read from socket

μ_1
socket appender

μ_4
socket appender

μ_7
socket appender

DEV код не на Spring Boot

1. getTags()

Log collector

2. read from socket

3. extract

List<String>

μ_1
socket appender

μ_4
socket appender

μ_7
socket appender

DEV код не на Spring Boot

1. getTags()

Log collector

2. read from socket

μ_1
socket appender

μ_4
socket appender

μ_7
socket appender

3. extract

List<String>

4. append

Scenario

Итоги

Поговорили **зачем и как**

- генерировать HTTP клиенты
- контекстно
  - проверять environment
  - чистить тестовые данные
  - собирать логи

Вспомогательные приемы при тестировании микросервисов.

Мартюшов Александр
amartyushov@gmail.com

Все примеры:
https://github.com/amartyushov/heisenbug2018