

Яндекс

Яндекс

Как проверить систему, не запуская её

Андрей Сатарин, @asatarin



Вы отвечаете за uptime
в вашем проекте?

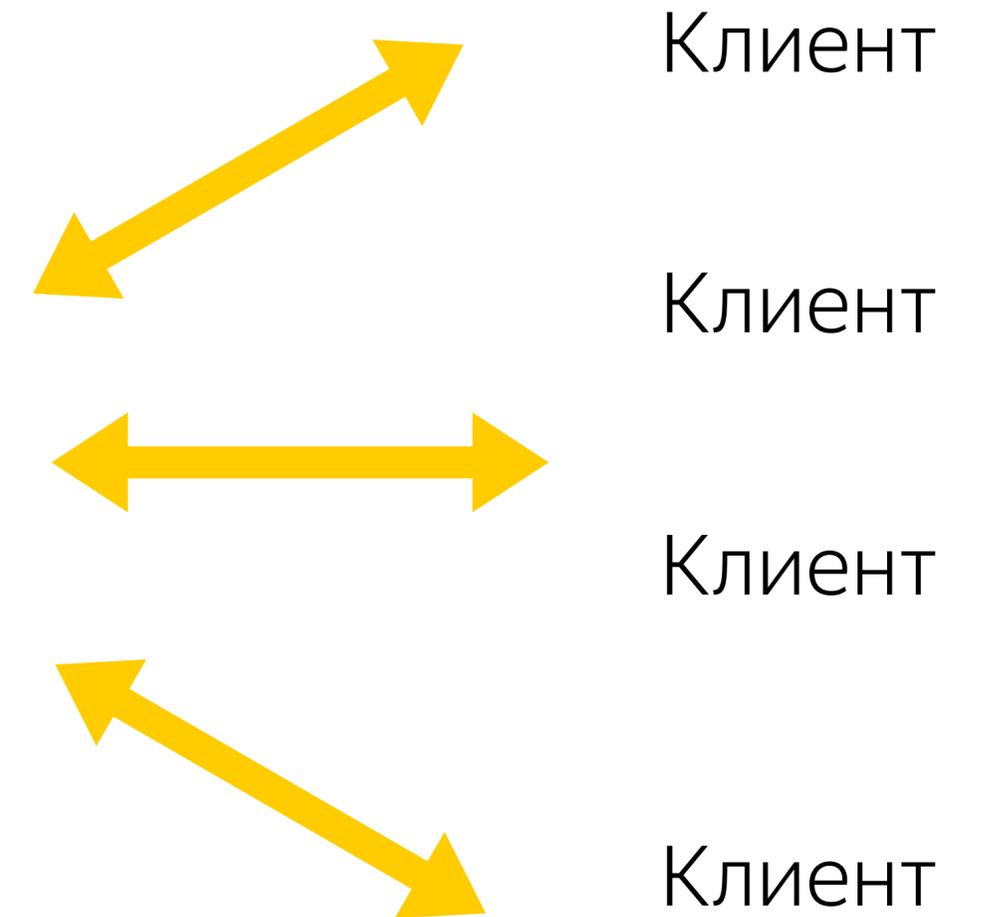
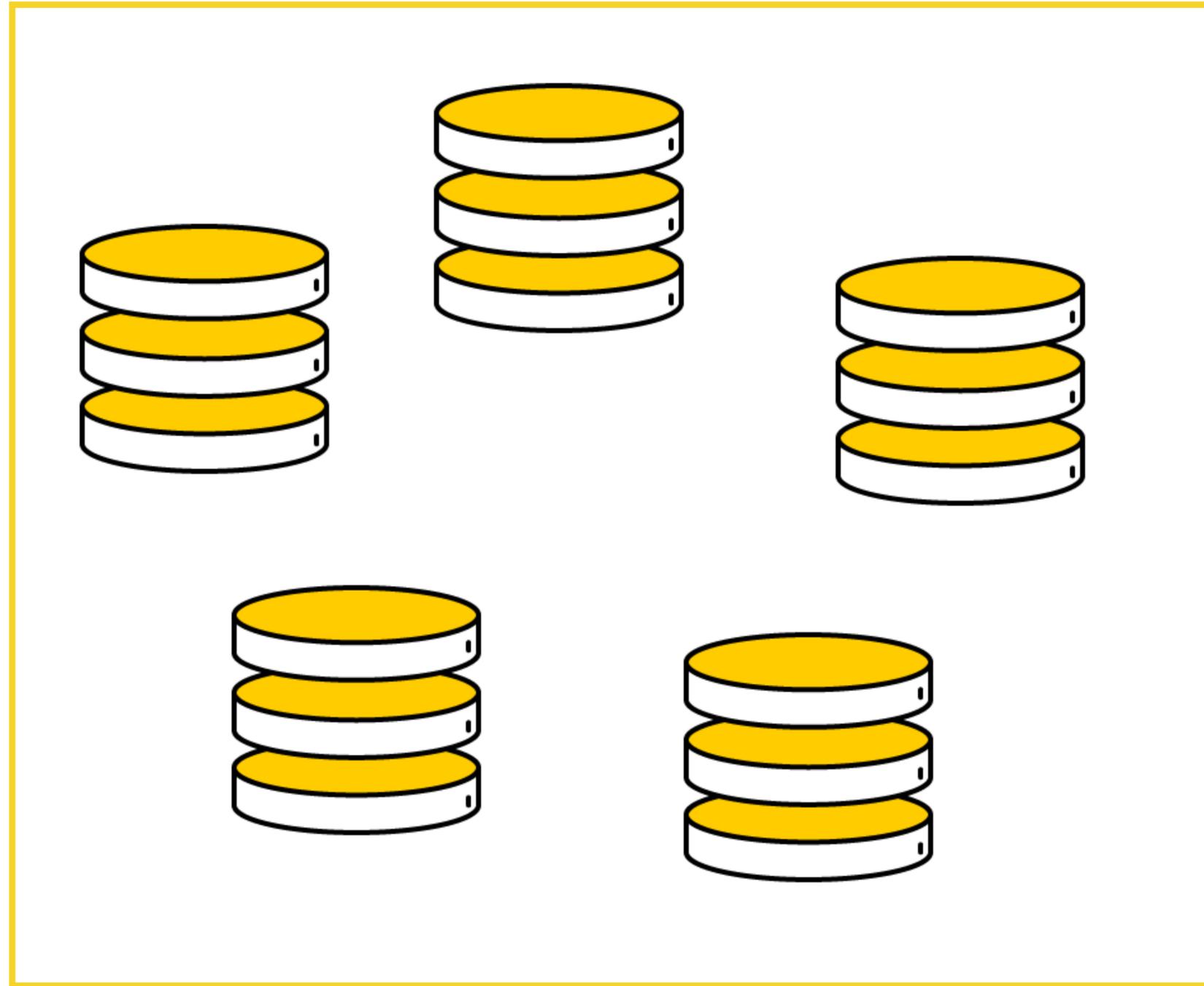
Да — вы в правильном
месте

Нет — еще можно уйти

Проблема



Сервис





Как не зафакапить наш
сервис?

Как не зафакапить наш сервис?

- › модульные тесты
- › функциональные тесты
- › тесты производительности
- › еще немного тестов
- › и еще тесты

Functionality

Usability

Reliability

Performance

Supportability

+

Сервис = код

Сервис = код + конфигурация

Сервис = код + конфигурация



Как (не) убить сервис
конфигурацией?



Пример 1

Одноклассники 2013



58,55

Рейтинг

Одноклассники

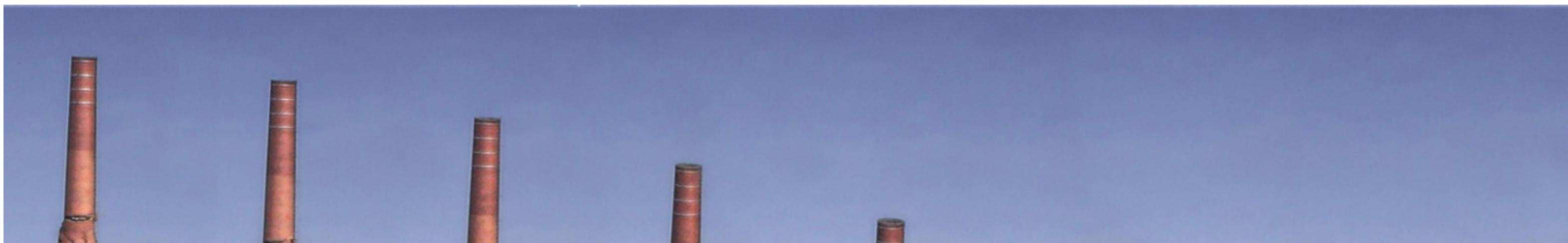
Делимся экспертизой



privately 8 октября 2015 в 08:40

Три дня, которые потрясли нас в 2013

Высокая производительность, Блог компании Одноклассники



Три дня в Одноклассниках — начало

«Однажды вечером система мониторинга зафиксировала незначительную проблему с одним из серверов. Для её устранения нужно было **поправить шаблон конфигурации.**»

Три дня в Одноклассниках — кульминация

«Но дежурный администратор правил файл шаблона в другом редакторе, который **поместил этот символ в конец файла**»

Три дня в Одноклассниках — развязка

«В Одноклассниках, как и во многих других проектах, применяется инцидент-менеджмент. То есть все нештатные ситуации фиксируются и делятся по категориям:

- › баг в нашем коде,
- › **ошибки конфигурации,**

...»



Пример 2

Google Chubby

Paxos Made Live - An Engineering Perspective

Tushar Chandra
Robert Griesemer
Joshua Redstone

June 20, 2007

Abstract

We describe our experience in building a fault-tolerant data-base using the Paxos consensus algorithm. Despite the existing literature in the field, building such a database proved to be non-trivial. We describe selected algorithmic and engineering problems encountered, and the solutions we found for them. Our measurements indicate that we have built a competitive system.

1 Introduction

Chubby — обычная работа

«By their very nature, fault-tolerant systems **try to mask problems**»



Для консенсуса нужно большинство — 3 из 5 нод

Chubby — работа со сбоями

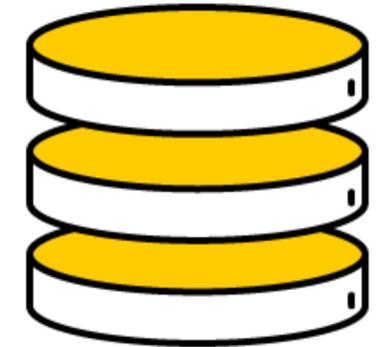
«By their very nature, fault-tolerant systems **try to mask problems**»



Для консенсуса нужно большинство — 3 из 5 нод

Chubby — ошибка в конфигурации

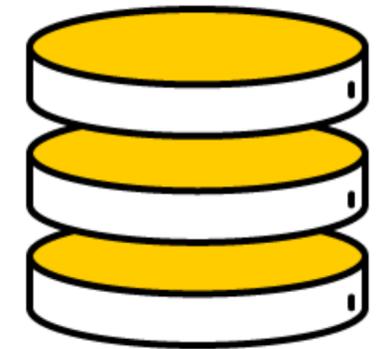
«We once started a system with five replicas, but **misspelled the name** of one of the replicas in the initial group»



Для консенсуса нужно большинство — 3 из 5 нод

Chubby — ошибка в конфигурации + сбой

«We once started a system with five replicas, but **misspelled the name** of one of the replicas in the initial group»



Невозможно собрать большинство из 5 нод

Вывод: ошибки конфигурации

- › Не могут быть найдены модульными/функциональными/и т.д. тестами
- › Могут очень **дорого стоить**
- › Могут быть **незаметны долгое время**
- › Им уделяют мало внимания

Код vs конфигурация

Код	Конфигурация
Java/C++/Python/etc	Protobuf/XML/JSON/YAML/etc
Строгий синтаксис	Строгий синтаксис
Строгая семантика	Семантика не специфицирована
Ревью/тесты/еще тесты/и т.д.	???

| Что делать?

| — Мы очень внимательно
проверим конфигурацию!

Конфигурация «Проект К»

- › Хранится в 18 protobuf файлах
- › Спецификация этих файлов — 400 строк кода
- › Множество внутренних связей
- › Больше нод — больше конфигурация

Нод в кластере	Строк конфигурации
8 нод	700+
32 ноды	1600+
300 нод	10000+



Что делать?

Теория





John Allspaw

@allspaw

Following



Them: "I have to cope with Complex Systems every day!"

Me: "Check out this research on this..."

Them: "ugh, hard PDF...boil it down for me?"

7:04 PM - 23 Oct 2017

<https://twitter.com/allspaw/status/922494066620796928>

Early Detection of Configuration Errors to Reduce Failure Damage

Tianyin Xu, Xinxin Jin, Peng Huang, and Yuanyuan Zhou, *University of California, San Diego*; Shan Lu, *University of Chicago*; Long Jin, *University of California, San Diego*; Shankar Pasupathy, *NetApp, Inc.*

<https://www.usenix.org/conference/osdi16/technical-sessions/presentation/xu>

**This paper is included in the Proceedings of the
12th USENIX Symposium on Operating Systems Design
and Implementation (OSDI '16).**

November 2–4, 2016 • Savannah, GA, USA

ISBN 978-1-931971-33-1

Early Detection of Configuration Errors to Reduce Failure Damage

Tianyin Xu, Xinxin Jin, Peng Huang, and Yuanyuan Zhou, *University of California, San Diego*; Shan Lu, *University of Chicago*; Long Jin, *University of California, San Diego*; Shankar Pasupathy, *NetApp, Inc.*

<https://www.usenix.org/conference/osdi16/technical-sessions/presentation/xu>

**This paper is included in the Proceedings of the
12th USENIX Symposium on Operating Systems Design
and Implementation (OSDI '16).**

November 2–4, 2016 • Savannah, GA, USA

ISBN 978-1-931971-33-1

Early Detection of Configuration Errors to Reduce Failure Damage

Tianyin Xu, Xinxin Jin, Peng Huang, and Yuanyuan Zhou, *University of California, San Diego*; Shan Lu, *University of Chicago*; Long Jin, *University of California, San Diego*; Shankar Pasupathy, *NetApp, Inc.*

<https://www.usenix.org/conference/osdi16/technical-sessions/presentation/xu>

**This paper is included in the Proceedings of the
12th USENIX Symposium on Operating Systems Design
and Implementation (OSDI '16).**

November 2–4, 2016 • Savannah, GA, USA

ISBN 978-1-931971-33-1

Latent Errors

«Our study shows that many of today's mature, widely-used software systems are **subject to latent configuration errors** (referred to as LC errors) in their critically important configurations—those related to the system's **reliability, availability, and serviceability.**»

Latent Errors => серьезные инциденты

«**LC [latent configuration] errors** contribute to **75% of the high-severity** issues and take much longer to diagnose, indicating their high impact and damage.»

Нет никаких проверок

«Finding 3:

Resulting from Findings 1 and 2, 4.7%– 38.6% of the studied RAS [reliability availability serviceability] parameters **do not have any early checks** and are thereby subject to LC [latent configuration] errors which **can cause severe impact** on the system's dependability.»

Нам нужны инструменты!

«Such prevalence of LC [latent configuration] errors indicates the **need for tool support to systematically rule out the threats.**»

«Another option to invoking the early checkers is to create a **standalone checking program** comprised of the checkers, and run it when the configuration file changes.»

Мораль

«This paper **advocates early detection of configuration errors** to minimize failure damage, especially in cloud and datacenter systems.»



В теории разницы
между теорией
и практикой нет,
а на практике есть

Приписывается разным людям

Практика



Что делать?

- › Нужны тесты на **конфигурацию продакшена**
- › Как написать такие тесты?

Типичные тесты в продакшене

- Надо проверить, что банковские переводы работают!
- Как ты себе это представляешь? Отправили 10 миллиардов и ждем?
- ???

Основано на реальных событиях



Мы не будет запускать
систему



Пример 3

Тривиальные проверки

Проверяем, что конфигурация загружается

```
@pytest.mark.parametrize(  
    ['path_to_config'],  
    all_production_config_paths()  
)  
  
def test_config_is_loadable(path_to_config):  
    config = load_config(path_to_config)  
    assert_that(config, not_(none()))
```

Проверяем, что конфигурация загружается

```
@pytest.mark.parametrize(  
    ['path_to_config'],  
    all_production_config_paths()  
)  
  
def test_config_is_loadable(path_to_config):  
    config = load_config(path_to_config)  
    assert_that(config, not_(none()))
```

```
CONFIG_FOLDER = 'path/to/config/folder'  
CONFIG_FILE_PATTERN = 'config.txt'
```

```
def all_production_config_paths():  
    return map(  
        os.path.dirname,  
        list_all_files_for_pattern(  
            CONFIG_FOLDER,  
            CONFIG_FILE_PATTERN  
        )  
    )
```

Простая проверка инвариантов

```
@pytest.mark.parametrize(  
    ['config'],  
    all_production_configs()  
)  
def test_unique_node_ids(config):  
    assert_that(  
        config.NodeIds,  
        sequence_has_unique_elements()  
    )
```

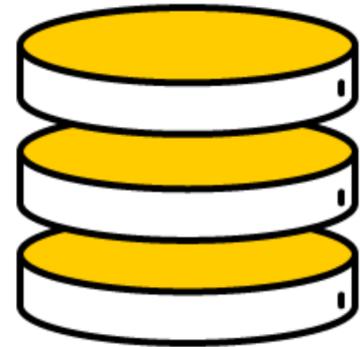


Пример 4

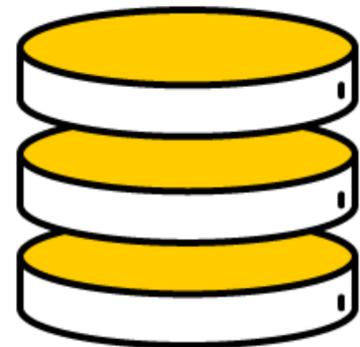
«Проект R» и порты

«Проект R» и порты

Primary
host1:port1



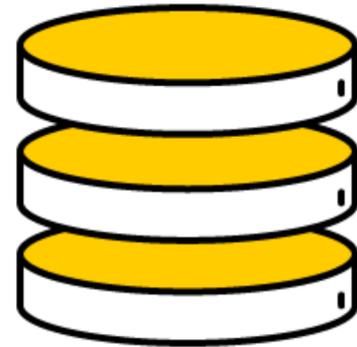
Secondary
host2:port2



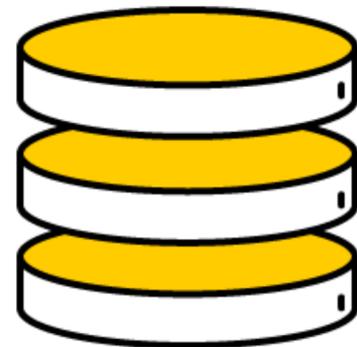
Клиент
host1:port1
host2:port2

«Проект R» и порты

Primary
host1:port1



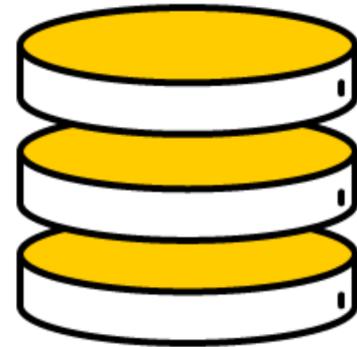
Secondary
host2:port2



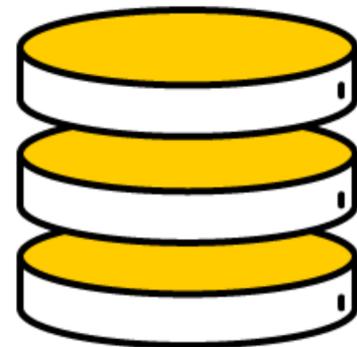
Клиент
host1:port1
host2:port3

«Проект R» и порты

Primary
host1:port1



Secondary
host2:port2



Клиент
host1:port1
host2:port3

```
@pytest.mark.parametrize(...)
def test_ports_are_valid(client, server_pri, server_sec):
    client_primary = client['primary']['connection']
    client_secondary = client['secondary']['connection']

    server_primary = server_pri['connection']
    server_secondary = server_sec['connection']

    assert_that(client_primary, equal_to(server_primary))
    assert_that(client_secondary, equal_to(server_secondary))
```

Мораль «Проект R»

- › Не все клиенты смотрели на правильный secondary
- › Проблема была **не видна** до фактического отказа primary
- › Поиск проблемы в продакшне **чрезвычайно дорог**
- › Тесты на конфигурацию легко масштабируются на новых клиентов



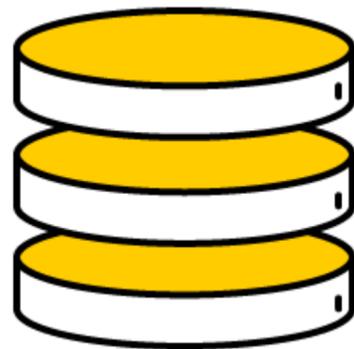
Пример 5

«Проект К»

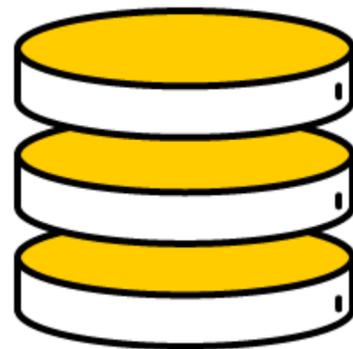
и отказоустойчивость

«Проект К» — отказоустойчивость

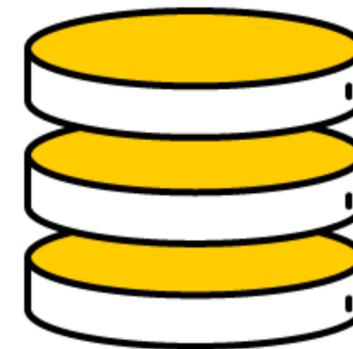
Каждая нода в отдельной стойке



Стойка 1



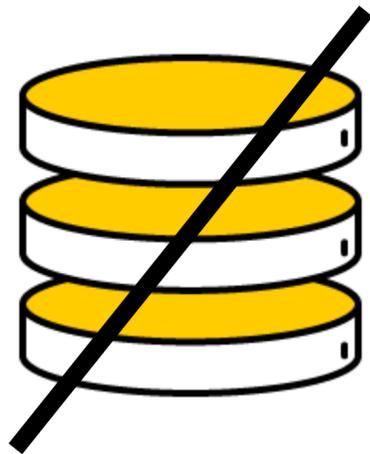
Стойка 2



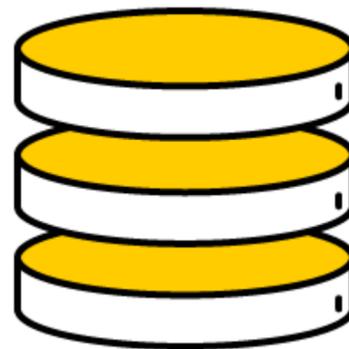
Стойка 3

«Проект К» — отказоустойчивость

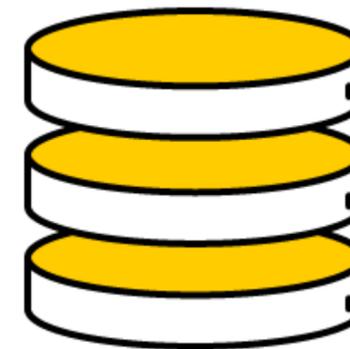
Отказала стойка — полет нормальный



Стойка 1



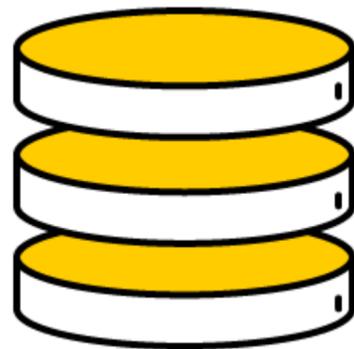
Стойка 2



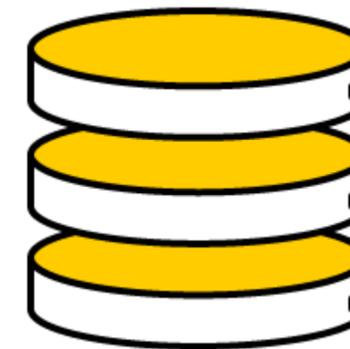
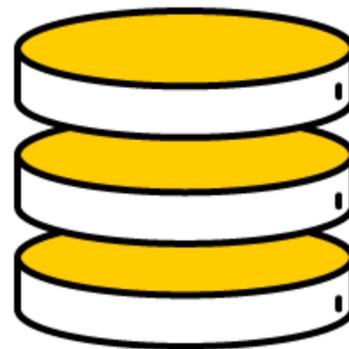
Стойка 3

«Проект К» — отказоустойчивость

Ошибка конфигурации — две ноды в одной стойке



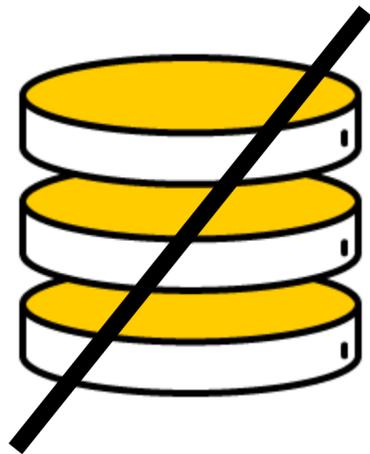
Стойка 1



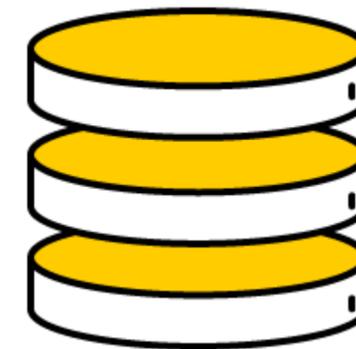
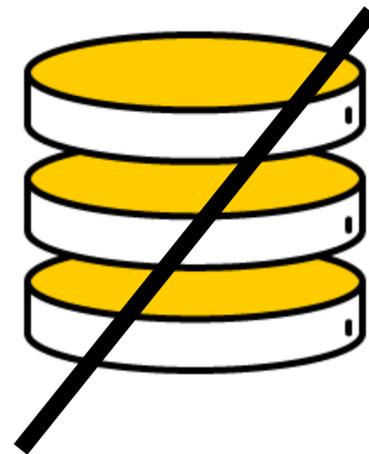
Стойка 3

«Проект К» — отказоустойчивость

Отказала стойка — часть системы стала недоступной



Стойка 1



Стойка 3

```
@pytest.mark.parametrize(...)
def test_every_node_is_in_a_separate_rack(config):
    hostname_to_rack_id = {
        hostname: get_rack_id(h)
        for h in config.hostnames
    }
    rack_ids = hostname_to_rack_id.values()
    assert_that(rack_ids, sequence_has_unique_elements())
```

```
@pytest.mark.parametrize(...)
def test_every_node_is_in_a_separate_rack(config):
    hostname_to_rack_id = {
        hostname: get_rack_id(h)
        for h in config.nostnames
    }
    rack_ids = hostname_to_rack_id.values()
    assert_that(rack_ids, sequence_has_unique_elements())
```

Где узнать как ноды в стойках стоят?

Как написать метод **get_rack_id()**?

- › Разметить руками — подходит, если нод мало (< 10)
- › Получить информацию из внешней системы (если она у вас есть)

Мораль «Проект К»

- › Система не знает про расположение серверов в стойках
- › Но мы то знаем, как сервера расположены в стойках
- › «Внешние» проверки на основе этого знания **легко нашли проблему**
- › Такой **тест легко масштабируется** на новые инсталляции

Выводы



Выводы

- › Конфигурация это как код, **только хуже**
- › Конфигурацию можно и **нужно тестировать**
- › Тесты на конфигурацию **легко масштабируются** на множество инсталляций/клиентов/и т.д.
- › Тесты на конфигурацию просто написать и получить **МНОГО ПОЛЬЗЫ**



Напиши свой первый
тест на конфигурацию,
username@

Андрей Сатарин

Ведущий инженер по автоматизации тестирования



asatarin@yandex-team.ru



<https://twitter.com/asatarin>

Ссылки

- › [Три дня, которые потрясли нас в 2013](#)
- › [«Paxos Made Live – An Engineering Perspective»](#)
- › [«Early detection of configuration errors to reduce failure damage»](#)
- › [Gixy — open source от Яндекса, который сделает конфигурирование Nginx безопасным](#)