

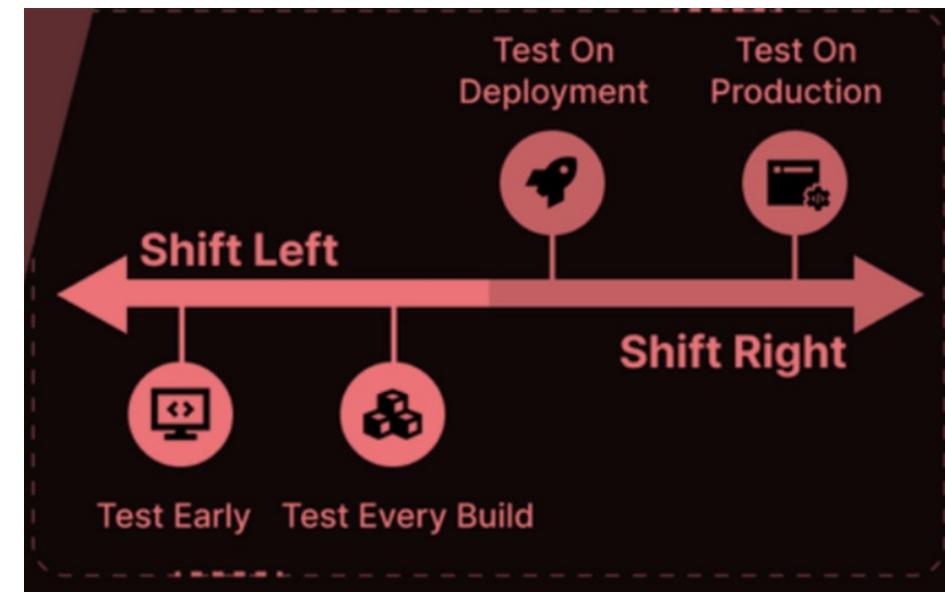
# Тестирование

# Тема 1. Shift-Right Testing

# Определение

→ **Shift-Right Testing** – это продолжение тестирования **после** запуска продукта. Это стратегия, при которой тестирование проводится **весь период жизненного цикла** продукта, не только до его запуска, но и после него. Этот подход основывается на непрерывной оптимизации продукта и сборе отзывов после его внедрения.

- ✓ Расширяет кругозор тестирования и включает в себя цикл жизни продукта
- ✓ Дополняет методики shift-left, ориентированные на раннюю стадию разработки
- ✓ Акцентирует внимание на пользовательском опыте
- ✓ Предполагает непрерывное улучшение продукта на основе пользовательских отзывов



# Почему Shift-Right Testing?

- **Конечный продукт:** Разработка продукта не заканчивается после его запуска. Для обеспечения качества именно конечные пользователи должны быть удовлетворены продуктами и услугами.
- **Непрерывный процесс:** Shift-Right позволяет превратить разработку продукта в непрерывный процесс
- **Измерение реальных ситуаций:** Используя shift-right, команды могут оценить, как продукт ведет себя в реальных условиях
- **Ориентация на пользователей:** При применении этого подхода удовлетворённость конечного пользователя становится приоритетной задачей
- **Постоянное улучшение:** Тестирование продолжается после релиза, что позволяет непрерывно улучшать продукт

# Критерии Shift-Right Testing

- ✓ Стабильность
- ✓ Производительность
- ✓ Удобство использования



## Stability

Ensure that the system remains stable even under unpredictable conditions



## Performance

Measure how the system performs in actual usage scenarios



## Usability

Examine how easy and intuitive it is for users to interact with the system

# Методы Shift-Right Testing

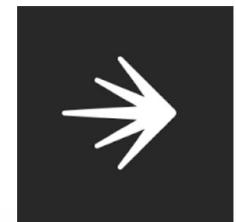
- **A/B тестирование:** Оценка двух версий продукта, чтобы определить наиболее эффективную
- **Распределенное развертывание (Canary releases):** Частичный выпуск новой версии продукта для ограниченного числа пользователей
- **Тестирование в продакшнене:** Проверка продукта в реальной рабочей среде включает в себя мониторинг и тестирование производительности в реальном мире
- **Мониторинг и логирование:** Отслеживание активности пользователей и ошибок в работе продукта, включая оценку производительности продукта в реальном времени
- **Обратная связь от пользователей:** Анализ отзывов пользователей и использование их для улучшения продукта
- **Переключение функциональности (Feature Toggling):** Позволяет включать или отключать определенные функции в продукте без необходимости изменять код
- **Хаос тестирование(Chaos Testing):** это подход, при котором в систему внезапно вводятся сбои, чтобы проверить её устойчивость и восстановление после сбоев.

# Инструменты Shift-Right Testing

- **A/B тестирование:** Optimizely, Mixpanel, VWO
- **Canary releases:** LaunchDarkly, Split.io
- **Тестирование в продакшне:** Honeycomb, Sentry
- **Мониторинг и логирование:** Kibana, Datadog, Dynatrace
- **Обратная связь от пользователей:** UserVoice, Hotjar, AskNicely
- **Feature Toggling:** FeatureFlags, Togglz
- **Chaos Testing:** Gremlin, Chaos Toolkit

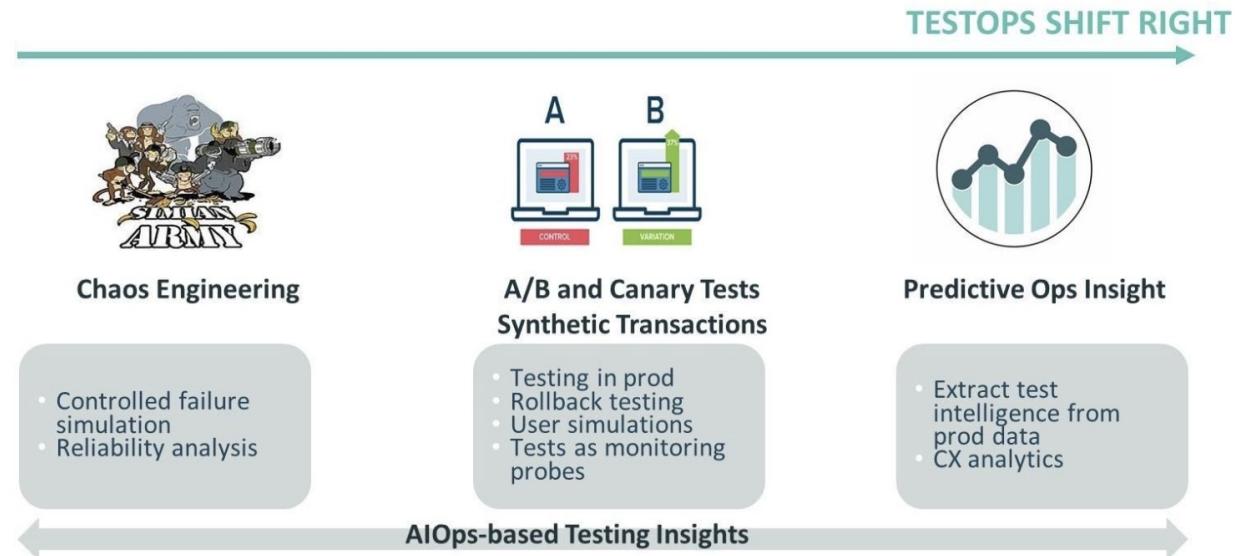


kibana



# Преимущества Shift-Right Testing

- **Постоянная обратная связь:** Быстрое устранение проблем и улучшение продукта
- **Реальное тестирование в реальном времени:** Наиболее точное представление работы продукта
- **Повышение уровня удовлетворенности пользователя:** Возможность обратной связи может повысить уровень удовлетворенности пользователя



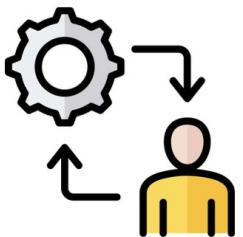
# Вызовы Shift-Right Testing



**Вопросы безопасности и конфиденциальности:** Тестирование в реальной среде может повлечь за собой проблемы с защитой данных



**Трудности при анализе данных:** Большой объем данных может затруднить анализ



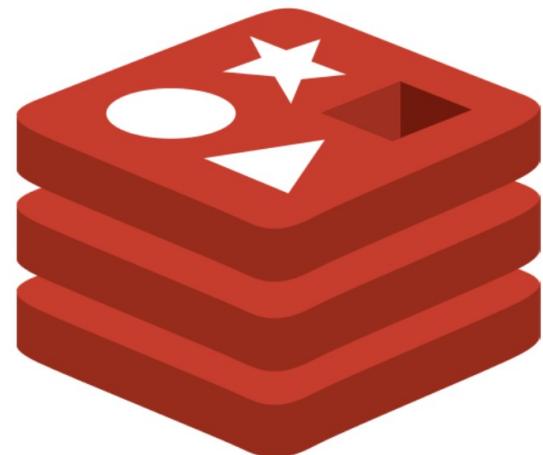
**Управление изменениями в процессе:** Реакции на обратную связь в реальном времени могут создать дополнительные задачи.

# Кто выполняет Shift-Right Testing?

- ✓ **Тестировщики / QA-инженеры:** Они выполняют фактические тесты, анализируют результаты и занимаются устранением найденных проблем.
- ✓ **SRE:** Они особенно важны при внедрении таких методов, как Chaos Testing и мониторинг в реальном времени. Их задача — обеспечить устойчивость и надежность продукта.
- ✓ **Продуктовые менеджеры:** Они могут использовать результаты тестирования для принятия стратегических решений о продукте.
- ✓ **UX-дизайнеры:** Они могут собирать обратную связь от пользователей и использовать результаты тестирования для улучшения пользовательского опыта.
- ✓ **Разработчики:** Они могут использовать результаты тестирования, чтобы исправлять ошибки и улучшать функциональность при внедрении новых функций.

# Тема 2. Performance Testing

# Чем SRE отличается от QA Performance?



# Чем SRE отличается от QA Performance?



SRE – обеспечивает надежность



QA Performance – часть работы SRE, помогает обнаружить проблему



SRE > QA



# Зачем НТ?



# Для клиента

- Пользователь не хочет ждать дольше 100ms (даже с 3g)
- Пользователь не хочет видеть ошибки (даже 11.11)
- Пользователь уходит к конкурентам (>10s)



Что-то пошло не так.

Повторите попытку позже или обратитесь в техническую поддержку.

# Для бизнеса

- Снизить риск прямых и репутационных издержек
- Увеличить число клиентов
- Заранее знать сколько ресурсов ему надо купить чтобы выехать в прод



# Для инженеров

**“ If you haven't tried it, assume it's broken. ”**

Unknown



# Зачем НТ?

- Клиенты довольны
- Бизнес работает бесперебойно и масштабируется
- Инженеры спят по ночам

# Что такое НТ?

**Тестирование производительности** — это тип нагружочного тестирования, используемый для определения **скорости работы** системы или подсистемы, особенно при ее взаимодействии с пользователем. Основной целью тестирования производительности является **убедиться, что система отвечает на запросы в течение заданного времени** на все уровни работоспособности.

**Тестирование производительности** помогает идентифицировать, какую часть системы нужно оптимизировать для улучшения производительности. Оно позволяет **оценивать и улучшать** такие атрибуты производительности, как **время отклика, пропускная способность и ресурсоемкость** системы.



# Что такое НТ?

**Тестирование производительности** — серия экспериментов на объекте тестирования для оценки его поведения под определенной нагрузкой

**Нагрузочное тестирование** – вид тестирования производительности



# Как подготовить НТ?

Ответить на вопросы – стратегия НТ

- Почему?
- Что?
- Как и Где?
- Чем?
- Кто?
- Когда?



# Почему?

## Инициирующие события НТ

- Планирование ресурсов до выхода в прод
- Подготовка к росту нагрузки (заранее)
- Оптимизация узких мест или подбор оптимальной конфигурации
- Миграция на новый стек



# Почему?

Инициирующие события НТ в реальности



- Все уже упало
- Система уже давно тормозит
- Завтра распродажа, потянем?

# Почему?

Определить цели и требования

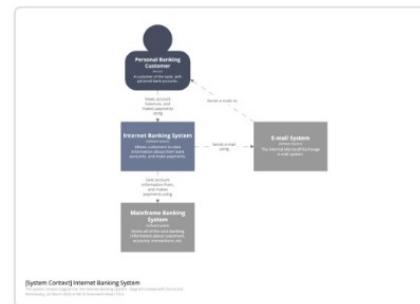
- ~~Нагрузить нагрузочно~~
- Определить максимальную пропускную способность при которой система отвечает нефункциональным требованиям (SLA)
- Определить способность системы масштабироваться под нагрузкой
- Подобрать оптимальную конфигурацию системы
- И тд и тп



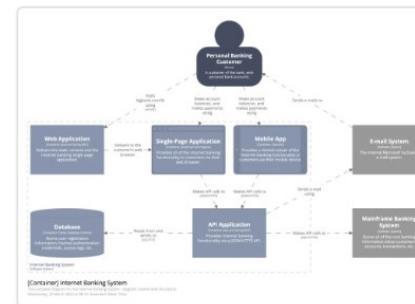
# Что?

Определить объект

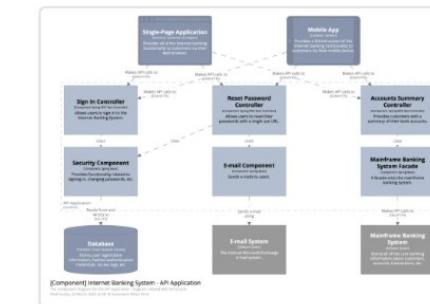
- Код
- Компонент
- Контейнер
- Система



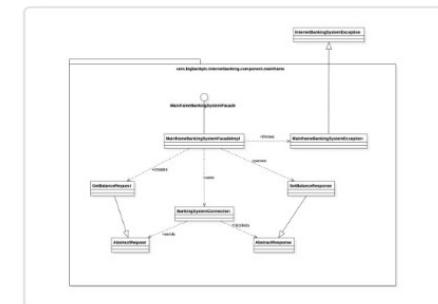
Level 1: A **System Context** diagram provides a starting point, showing how the software system in scope fits into the world around it.



Level 2: A **Container** diagram zooms into the software system in scope, showing the high-level technical building blocks.



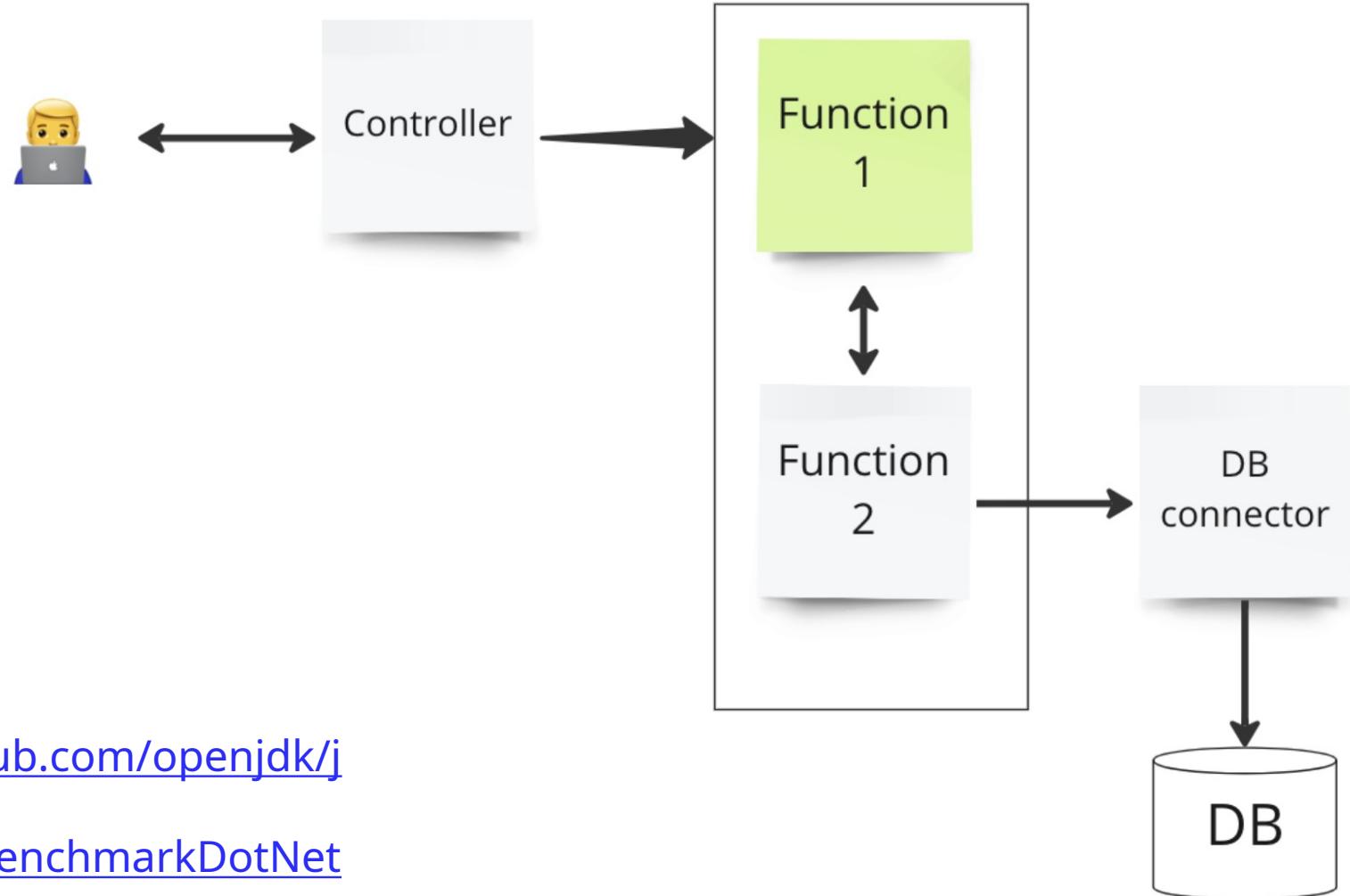
Level 3: A **Component** diagram zooms into an individual container, showing the components inside it.



Level 4: A **code** (e.g. UML class) diagram can be used to zoom into an individual component, showing how that component is implemented.

# C4: Уровень кода

## ➤ Функция



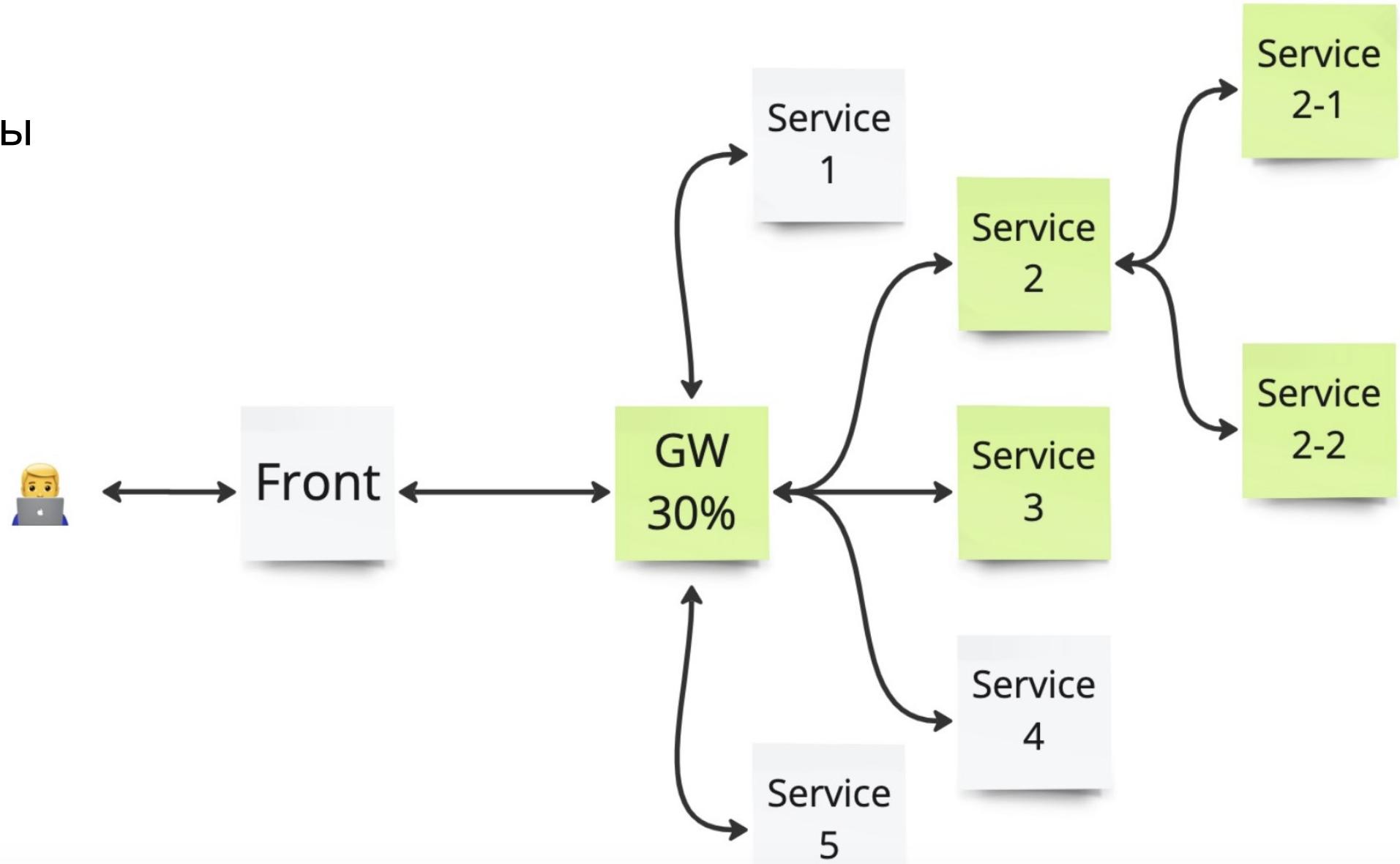
go test -bench=. <https://github.com/openjdk/jmh>

<https://github.com/dotnet/BenchmarkDotNet>

...

# C4: Уровень контейнера

## ➤ Сервисы



# Что?

- Инфраструктура:
  - Сеть
  - Диск
  - И т.д. и т.п.



<https://github.com/cloudmercato/awesome-benchmark>

# Объект тестирования

- Что тестируем
- Что учитываем при анализе результатов

# Объект зависит от цели

Какие виды зависимостей вам знакомы?

Возможные  
варианты ответов

Алкогольная

Наркотическая

Интернет-  
зависимость

Табакокурение

Игровая

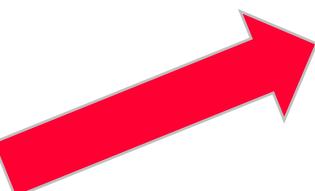
Зависимость объекта  
тестирования от цели

# Что? -> Где?

- Прод
- Препрод
- Тест
- Дев

# А что, если на проде?

- Еще нет реальных пользователей
- Можем ограничить объект тестирования:
  - тестовые данные
  - инфраструктура
  - сервисные окна
- Есть достаточный мониторинг



- Разрешили безопасники, SRE

# Мониторинг

Может и не надо НТ?

- А почему медленно работает?
- Где узкое место?
- Почему произошла авария?

**ЛЮБЛЮ  
СМОТРЕТЬ**



# Deployment strategy

Может и не надо НТ?

- Постепенно переключаем пользователей
- Теневой релиз
- Дублируем трафик

✓ Можем быстро вернуть как было!

[https://youtu.be/VHBB0gIiUgA?list=PLH-XmS0lSi\\_xQtVkWsUMSVUScK\\_3G\\_LUP](https://youtu.be/VHBB0gIiUgA?list=PLH-XmS0lSi_xQtVkWsUMSVUScK_3G_LUP)



Normal devs deploying a service

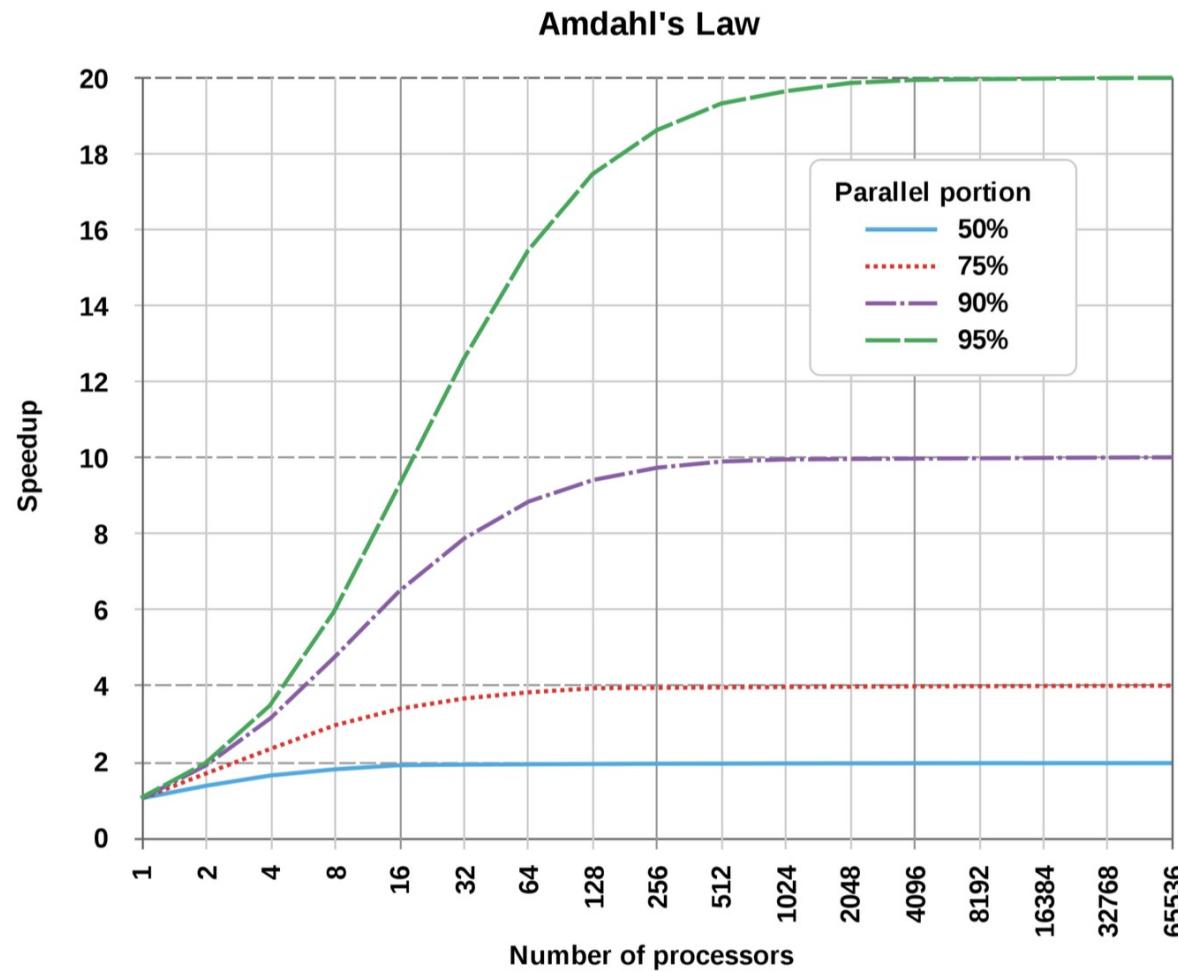


# Если не на проде?

- ❖ Как это повлияет на результат?



# Ограничения



[https://en.wikipedia.org/wiki/Amdahl%27s\\_law](https://en.wikipedia.org/wiki/Amdahl%27s_law)

# Если не на проде?

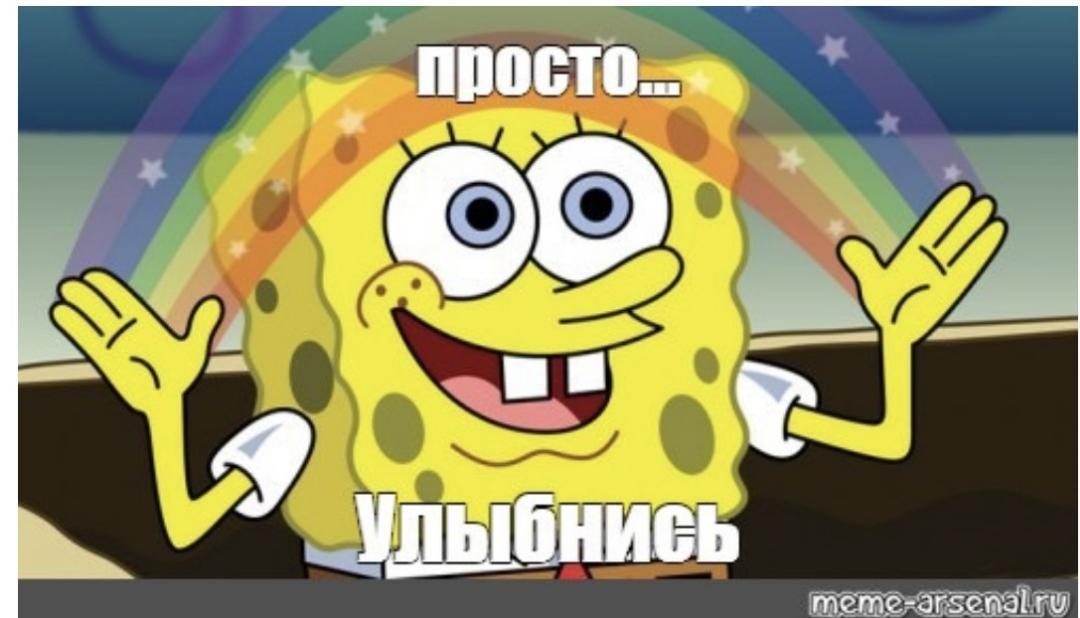
Тестовый стенд должен на  $\geq 50\%$  соответствовать проду

## Рекомендации:

- Вертикально 100%
- Горизонтально 50%
- Конфигурация ==



# Аппроксимация с теста на прод?



# Апроксимация: предположение

- Если на тесте капасити 1000 RPS, значит и на проде будет не меньше.
- Если на тесте утечка памяти, значит и на проде будет.
- Если на тесте деградация, значит и на проде будет.

Прод



Тест



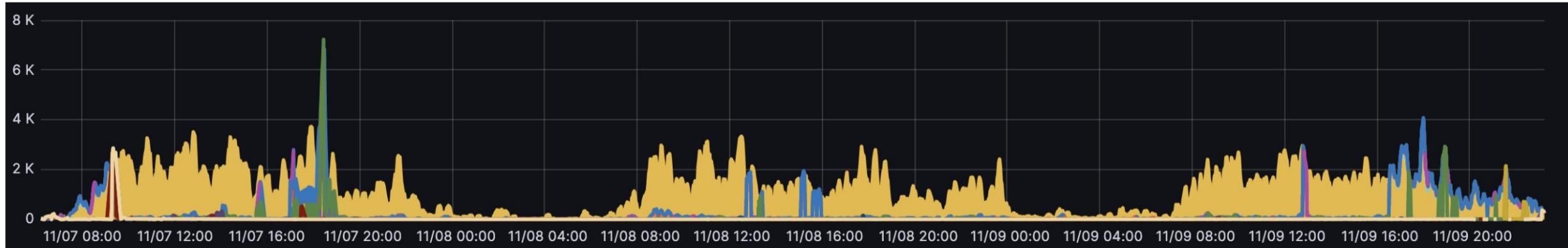
# Где?

- Если можем ограничить объект – тестируем на проде
- Если не можем – копируем конфигурацию на тест и находим 50% ресурсов

# Как?

Определить модель нагрузки

- Статистика или прогноз – профиль нагрузки
- Набор профилей – модель нагрузки

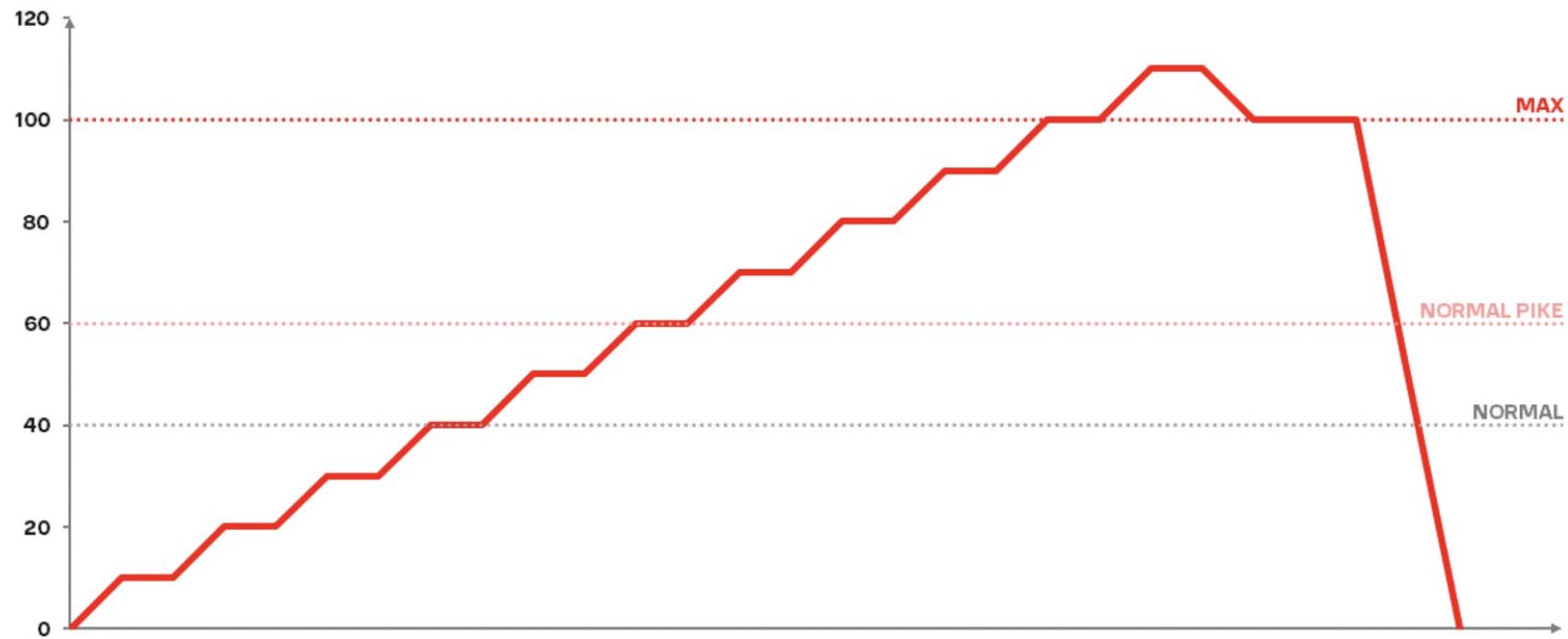


# Kak?

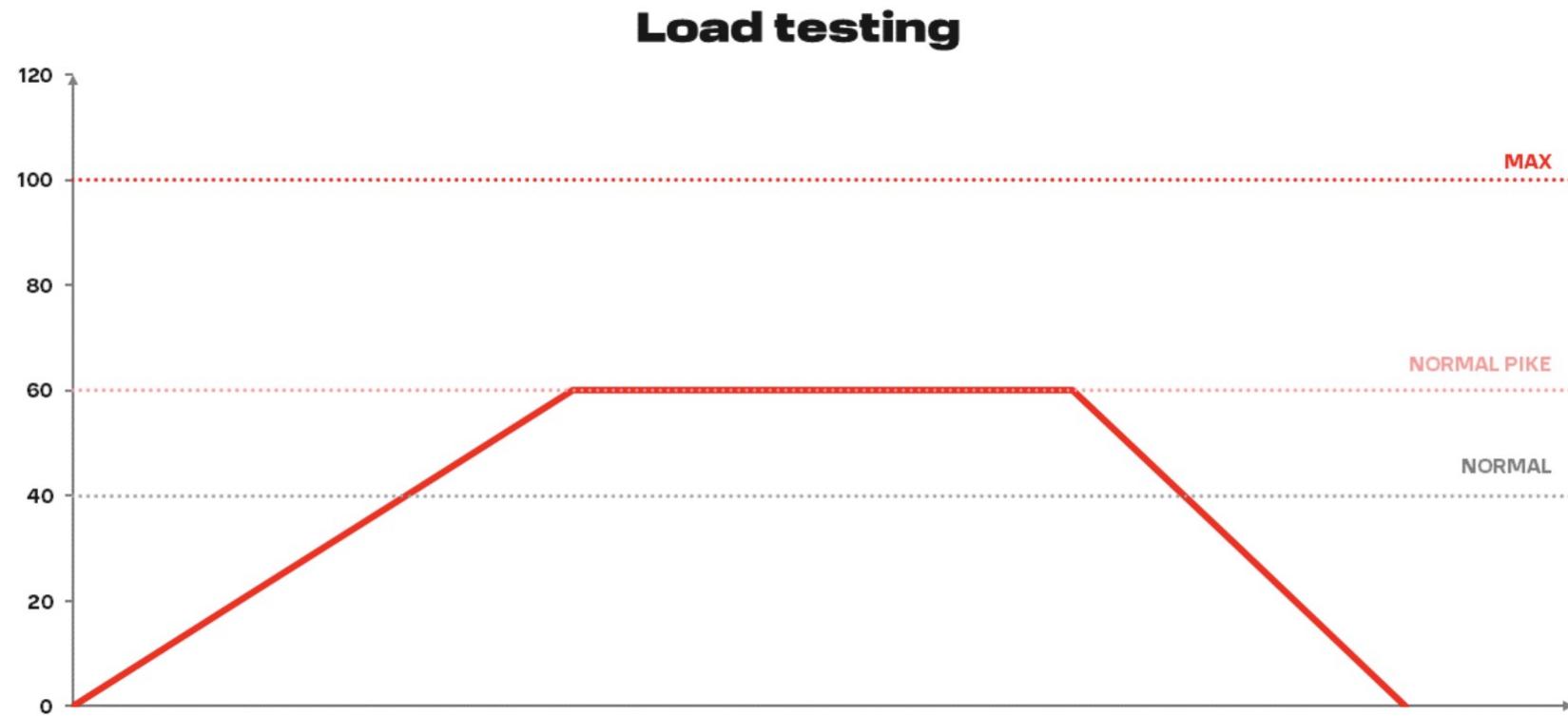
- Capacity
  - Load
  - Destructive
  - Soak/Stability
  - Bottleneck
  - Stress
  - Spike
  - Volume
  - Failover/Recovery
  - Scalability
  - Config
  - Compare
  - Reproduce
  - Baseline
- 
- Chaos

# Capacity

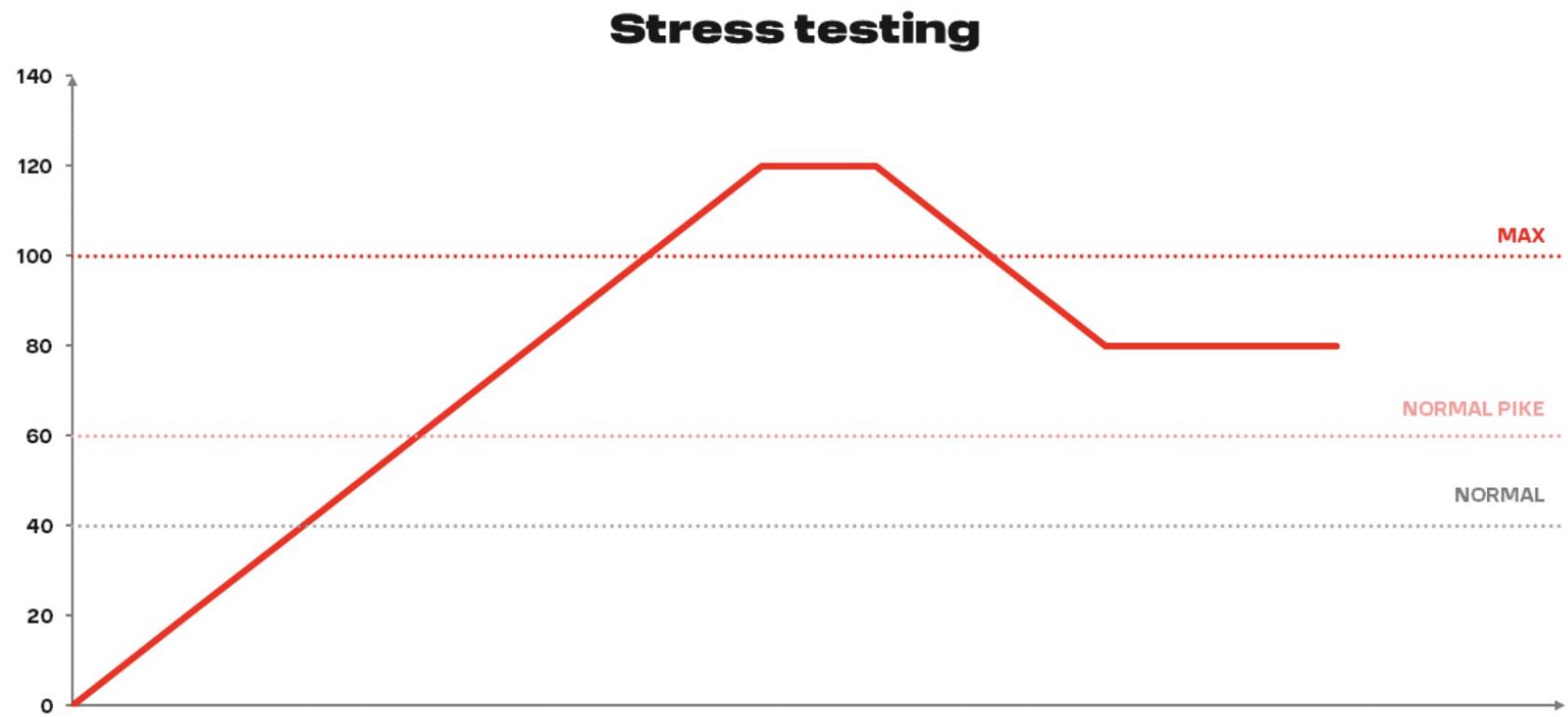
## Capacity testing



# Load

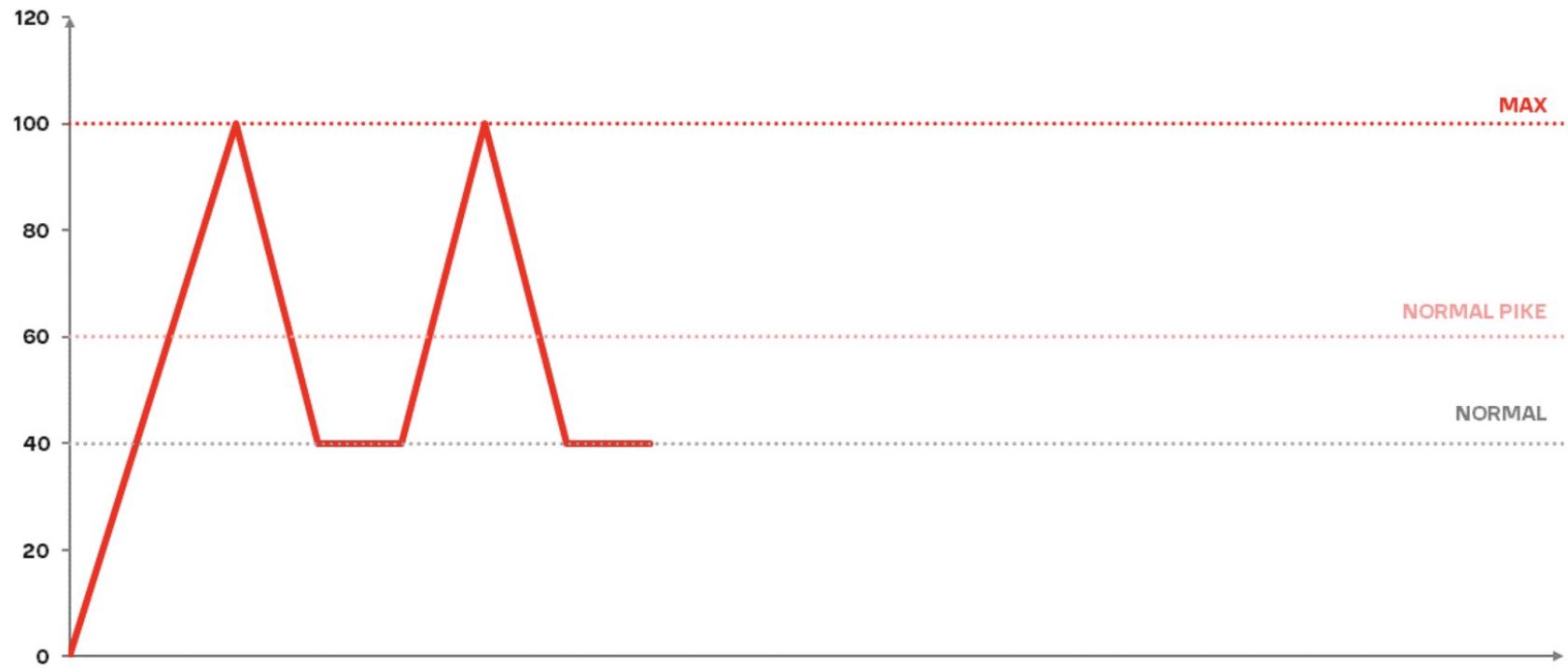


# Stress



# Spike

## Spike Testing



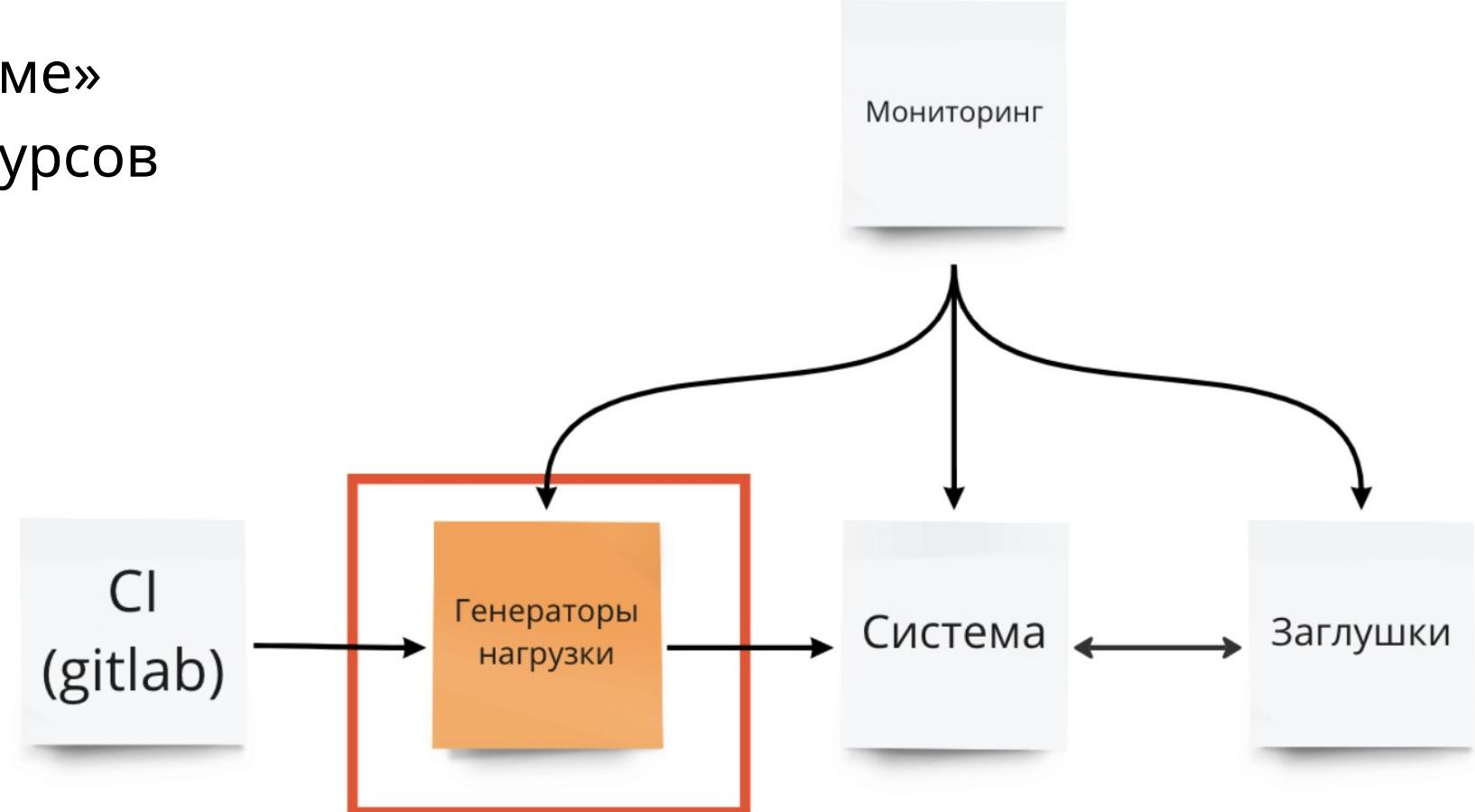
# Performance diagrams



# Тема 3. Инструменты НТ

# Генераторы нагрузки

- «Близко к системе»
- Достаточно ресурсов



Как подготовить генератор нагрузки?

<https://gatling.io/docs/gatling/reference/current/core/operations/>

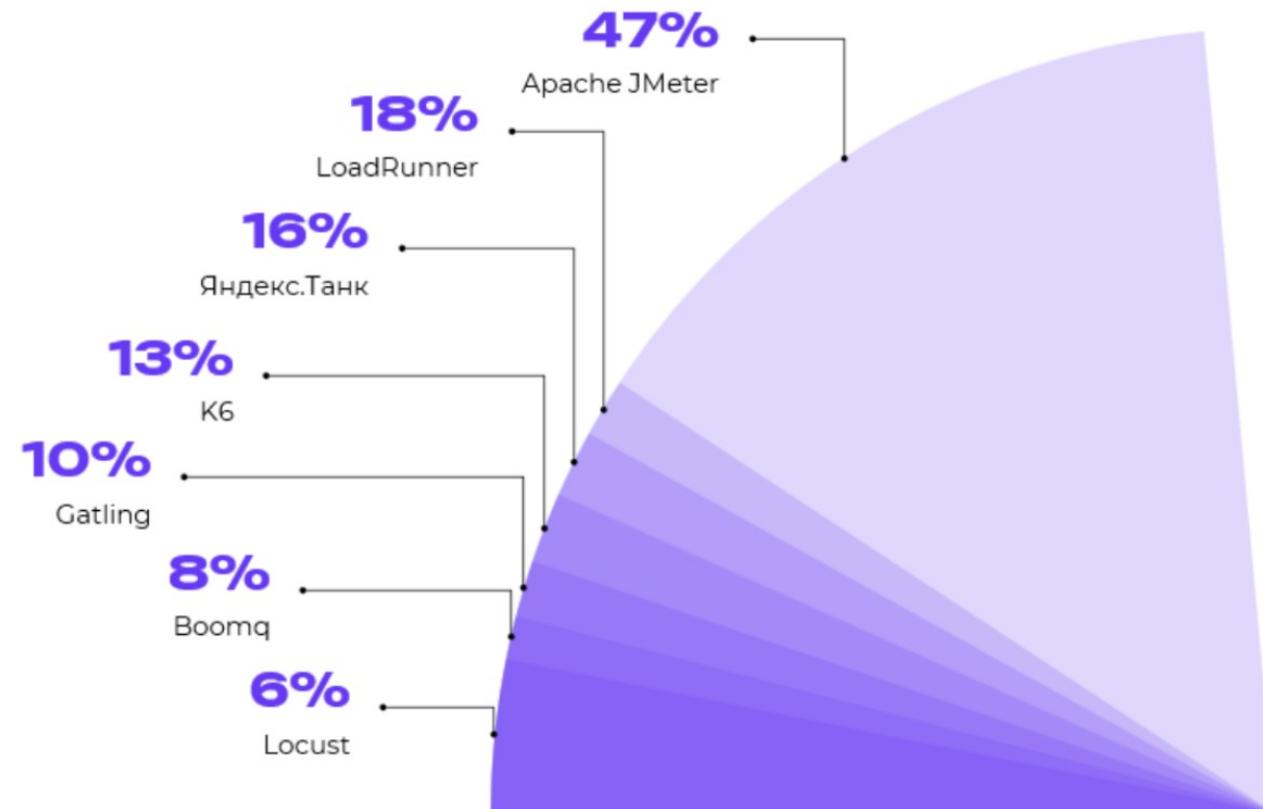
[https://yandextank.readthedocs.io/en/latest/generator\\_tuning.html?highlight=tuning#tuning](https://yandextank.readthedocs.io/en/latest/generator_tuning.html?highlight=tuning#tuning)

# Инструменты НТ



# JMeter

Какие инструменты нагрузочного тестирования используются в вашей организации?



**35%**

компаний использовали  
LoadRunner по данным 2020-  
2021 года (-17%)

**42%**

компаний использовали  
Apache JMeter по данным  
на 2020-2021 год (+5%)

# JMeter

Apache JMeter (5.6.2)

00:00:00 0/0 0/0

Test Plan  
CSV Data Set Config  
HTTP Cookie Manager  
Thread Group  
GET /api/user

**HTTP Request**

Name: GET /api/user  
Comments:

**Basic Advanced**

Web Server  
Protocol [http]: Server Name or IP: Port Number:

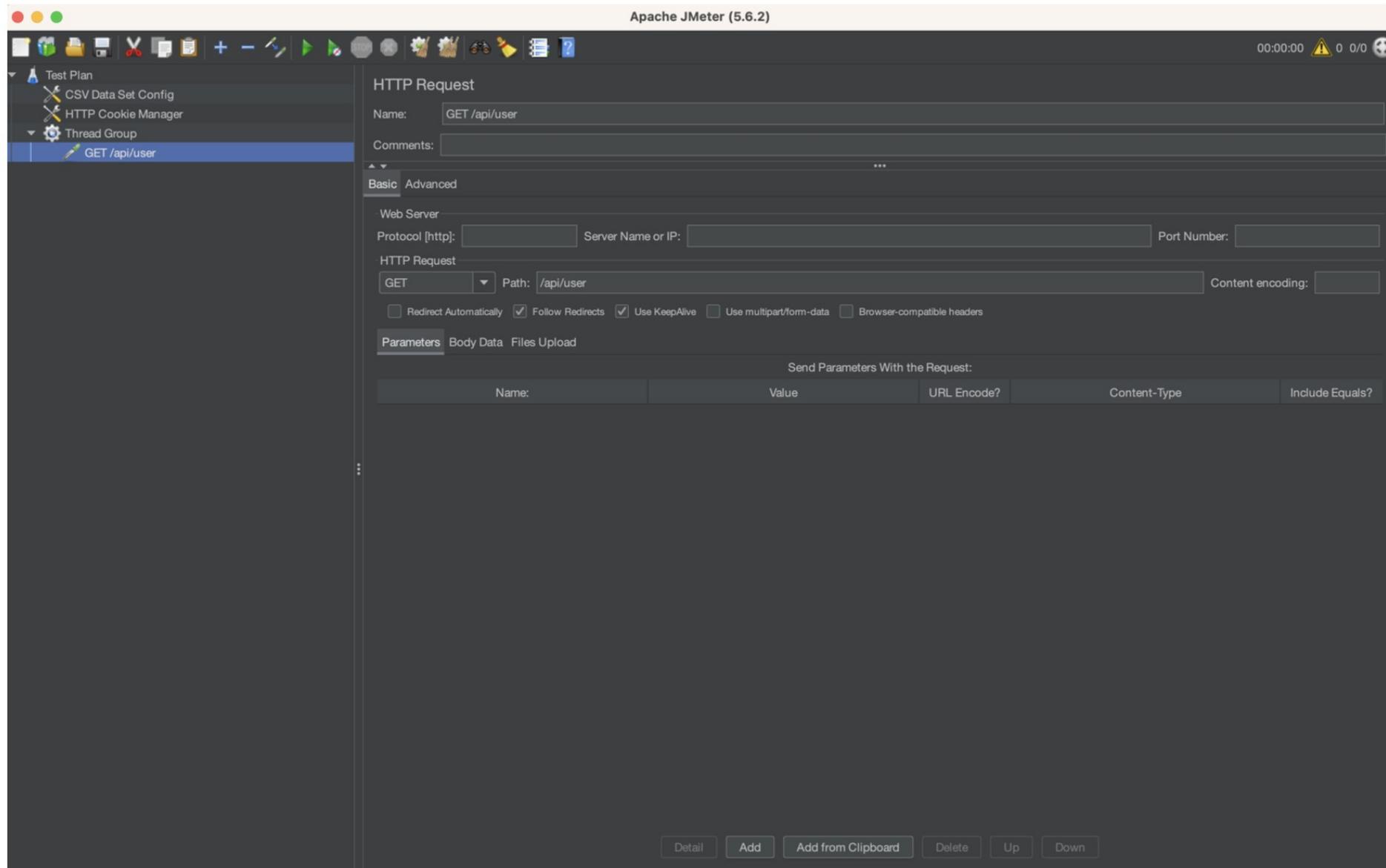
HTTP Request  
Method: GET Path: /api/user Content encoding:  
 Redirect Automatically  Follow Redirects  Use KeepAlive  Use multipart/form-data  Browser-compatible headers

Parameters Body Data Files Upload

Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include Equals?
-------	-------	-------------	--------------	-----------------

Detail Add Add from Clipboard Delete Up Down



# JMeter

- Простой на старте
- Очень популярен
- Почти любой протокол
- Открытая модель нагрузки –  
появилась
- Есть jmeter-java-dsl
- Конфиг – XML (git diff?)
- Как собирать?
- Проблемы с потоками

# JMeter

- [https://jmeter.apache.org/download\\_jmeter.cgi](https://jmeter.apache.org/download_jmeter.cgi)
- <https://jmeter.apache.org/usermanual/get-started.html>
- <https://www.blazemeter.com/university/jmetertm-intro>
- <https://habr.com/ru/articles/518702/>
- <https://grafana.com/grafana/dashboards/1152-jmeter-load-test/>
- <https://github.com/apache/jmeter>
- <https://abstracta.github.io/jmeter-java-dsl/>



## Notes

1. One year after jmeter-java-dsl release, on November 2021, Gatling released [3.7 version](#), including a Java friendly API for existing Gatling Scala API. This greatly simplifies usage for Java users and is a great addition to Gatling.

As a side note, take into consideration that the underlying code is still Scala and async model-based, which makes debugging and understanding it harder for Java developers than JMeter code. Additionally, the model is still tied to `Simulator` classes and maven (gradle or sbt) plugin to be able to run the tests, compared to the simplicity and flexibility of jmeter-java-dsl tests execution.

# Gatling

- Maven/Gradle/SBT/Bundle
- Java/Kotlin/Scala DSL
- Тесты «как код»
- Netty – неблокирующее IO
- Akka – оркестрация сценариев

# Gatling

- ✓ Удобно хранить в git
  - ✓ Хорошая интеграция в CI
  - ✓ Удобная организация проекта
  - ✓ Поддержка любого протокола для которого есть JVM библиотека
  - ✓ Производительный
- 
- Из коробки только graphite метрики (и других официально не будет)
  - Нет автостопов
  - «Тяжелые» проекты

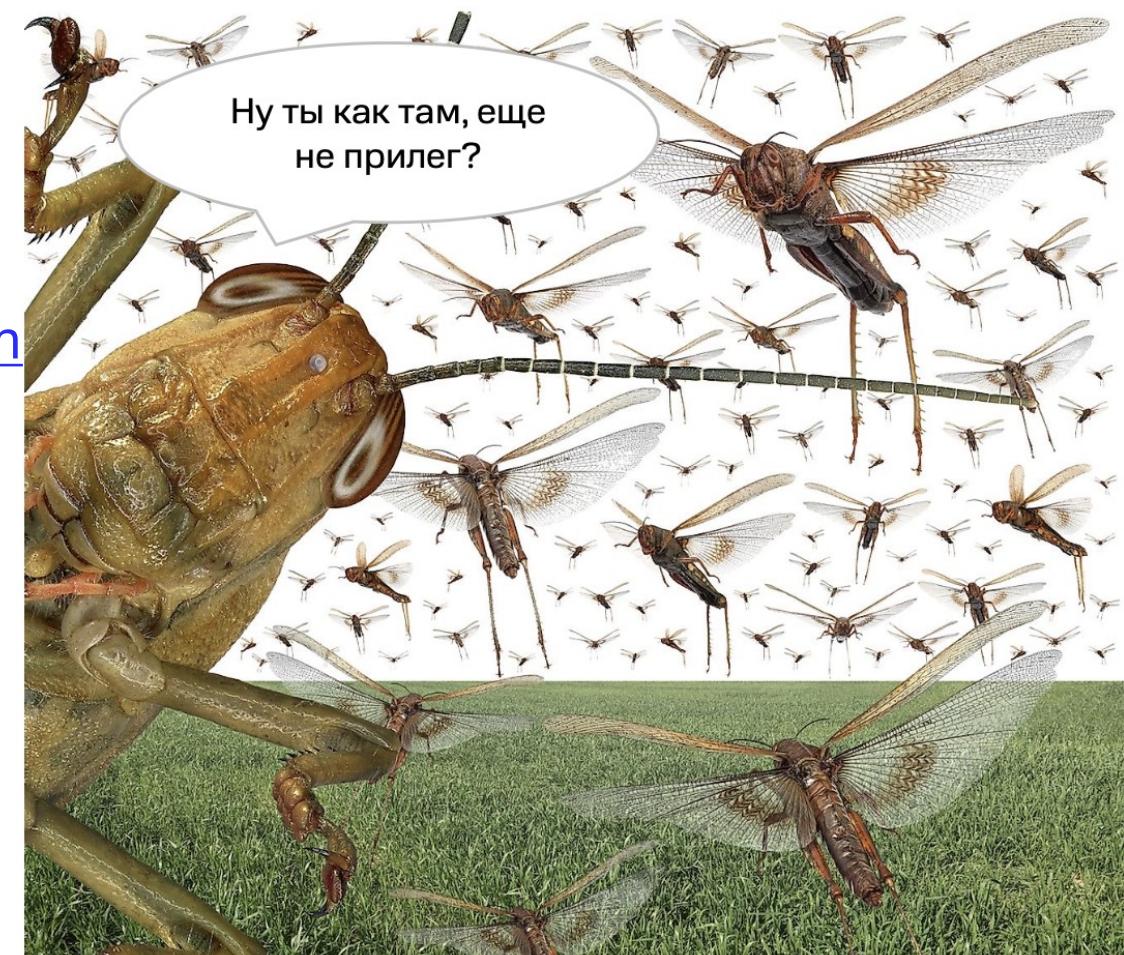
# Gatling

- Обучение: <https://gatling.io/academy/>
- Доки: <https://gatling.io/docs/gatling/reference/current/>
- Netty: <https://www.youtube.com/watch?v=NvnOg6g4114>
- Шаблон: <https://github.com/Tinkoff/gatling-template.g8>
- Полезные утилиты: <https://github.com/Tinkoff/gatling-picatinny>

# Locust

**Идея** заключается в том, что во время тестирования сайт подвергается «**атаке стаи саранчи**» (от англ. locust — саранча).

- <https://locust.io/>
- <https://docs.locust.io/en/stable/installation.html>
- <https://www.gevent.org/>
- <https://greenlet.readthedocs.io/en/latest/>
- <https://loadtestweb.info/2017/08/23/pacing/>



# Locust

- Для Python 3

```
pip install locust
```

или

```
pip3 install locust
```

- Последняя master ветка из git

```
pip install -e git://github.com/locustio/locust.git@master#egg=locust
```

- Дополнительный фикс для Windows (если что-то пошло не так)

- Загрузите предварительно собранные бинарные пакеты

- [pyzmq](#)

- [gevent](#)

- [greenlet](#)

- Установите каждый файл wheel (whl):

```
pip install name-of-the-package.whl
```

- Повторите установку locust



# Locust

- Напишем базовый тест на две «ручки»

```
from locust import HttpUser, between, task

class MyUser(HttpUser):
    wait_time = between(0.1, 0.5)

    @task(1)
    def get_status(self):
        self.client.get("/status/")

    @task(1)
    def get_index(self):
        self.client.get("/", name="index")
```

Интервал ожидания  
между запросами, с  
Есть еще constant()  
и constant\_pacing()

Веса запросов из  
общей массы

Более красивое имя  
нашей «ручки»

- Запуск

```
locust -f perf_test.py --headless --host http://localhost:8000 -u 1000 -r 100 -t 3m
# OR
locust -f perf_test.py --master
locust -f perf_test.py --worker
```

k6



k6

Friendship ended with  
Now

Gatling

k6

is my  
best friend



# k6

- Генератор на Go, скрипты простые
- Активно развивается + поддержка от Grafana
- Метрики в Prometheus/Victoria metrics/Influxdb
- k8s оператор
- Интегрируется в Gitlab
- Автостопы «из коробки» (+для тестов на проде)

# k6

- Доки: <https://k6.io/docs/>
- Репо: <https://github.com/grafana/k6>
- Собрать с плагинами: <https://github.com/grafana/xk6>
- Плагины: <https://github.com/topics/xk6>
- Chaos: <https://k6.io/docs/javascript-api/xk6-disruptor/>

# Что еще?

- HTTP: <https://github.com/yandex/pandora> + <https://github.com/yandex/yandex-tank> (большая нагрузка + автостопы)
- <https://gettaurus.org/> (автостопы + тесты как yaml)
- HTTP: <https://github.com/giltene/wrk2> (бенчмарк HTTP + Lua)
- L3-7: <https://trex-tgn.cisco.com/>
- S3: <https://github.com/minio/warp>
- DB: <https://www.hammerdb.com/>



# Литература и полезные ссылки

- <https://katalon.com/resources-center/blog/shift-right-testing/>
- <https://help.mindbox.ru/docs/ab-test-google-optimize/>
- <https://www.optimizely.com/>
- <https://vwo.com/>
- <https://cleverics.ru/digital/2020/07/sdvig-testirovaniya-vpravo-vozniknovenie-testops/>
- <https://www.lambdatest.com/learning-hub/chaos-testing/>
- <https://www.oreilly.com/library/view/chaos-engineering/9781492043850/>
- <https://www.reg.ru/blog/polnyj-gajd-po-b-testam/>

# Еще ссылки

- Если надумаете разрабатывать свой инструмент НТ:  
<https://www.scylladb.com/2021/04/22/on-coordinated-omission/>
- Пример работы с Gatling: <https://gitlab.com/tinkoffperfworkshop>
- Пример организации проекта с k6: <https://gitlab.com/gitlab-org/quality/performance>
- Статьи и книжки про performance: <https://www.brendangregg.com/>, воркшоп:  
<https://youtu.be/FJW8nGV4jxY>
- От сообщества: <https://load.qa/>
- Настроить генератор:  
<https://gatling.io/docs/gatling/reference/current/core/operations/>  
[https://yandextank.readthedocs.io/en/latest/generator\\_tuning.html?highlight=tuning#tuning](https://yandextank.readthedocs.io/en/latest/generator_tuning.html?highlight=tuning#tuning)

# Больше ссылок

- <https://www.nngroup.com/articles/website-response-times>
- [https://services.google.com/fh/files/blogs/google\\_delayexp.pdf](https://services.google.com/fh/files/blogs/google_delayexp.pdf)
- <https://www.oreilly.com/library/view/designing-for-performance/9781491903704/ch01.html>
- <https://sre.google/sre-book/testing-reliability/>
- <https://c4model.com/>
- <https://github.com/openjdk/jmh>
- <https://github.com/cloudmercato/awesome-benchmark>
- [https://en.wikipedia.org/wiki/Amdahl%27s\\_law](https://en.wikipedia.org/wiki/Amdahl%27s_law)
- <https://www.apdex.org/>

# Еще больше ссылок

- <https://github.com/aliesbelik/awesome-gatling>
- <https://github.com/aliesbelik/awesome-jmeter>
- <https://github.com/aliesbelik/awesome-locust>
- <https://github.com/grafana/awesome-k6>