NanoCloud is back in town: тестирование Spring приложений и не только для java-разработчиков

JPoint Spring 2024

Москва, ЦМТ, Апрель 2024

Алексей Рагозин

Занимаюсь высоконагруженными системами на Java с 2006. Разрабатывал софт для торговли на фондовых рынках, телекоме, e-commerce, RTB, здравоохранения. Выступаю на конференциях, организую митапы, провожу тренинги.

Владимир Красильщик

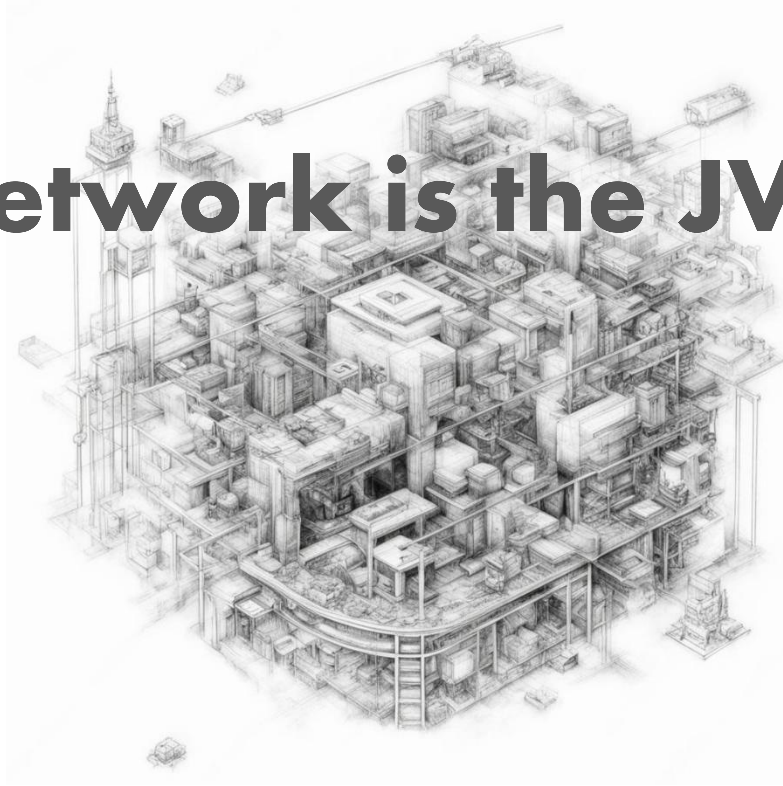
Прагматичный java программист из СПб, имею более 20 лет разнообразного опыта разработки на java. В данное время разрабатываю бекенд в JUG Ru Group. Докладчик на конференциях JUG Ru Group и ПК-шник в JPoint.

Фёдор Бобин

Занимаюсь коммерческой разработкой под JVM с 2009 года. Работал в сфере Forex и других банковсковских услуг, занимался облачной телефонией. В настоящее участвую в развитии платформы "ВК Звонки".

Network is the Computer

Network is the JVM



Немного истории

2009 – первые проекты в AWS

Нужен быстрый способ выкатывать и отлаживать изменения на виртуалках

Ручной процесс

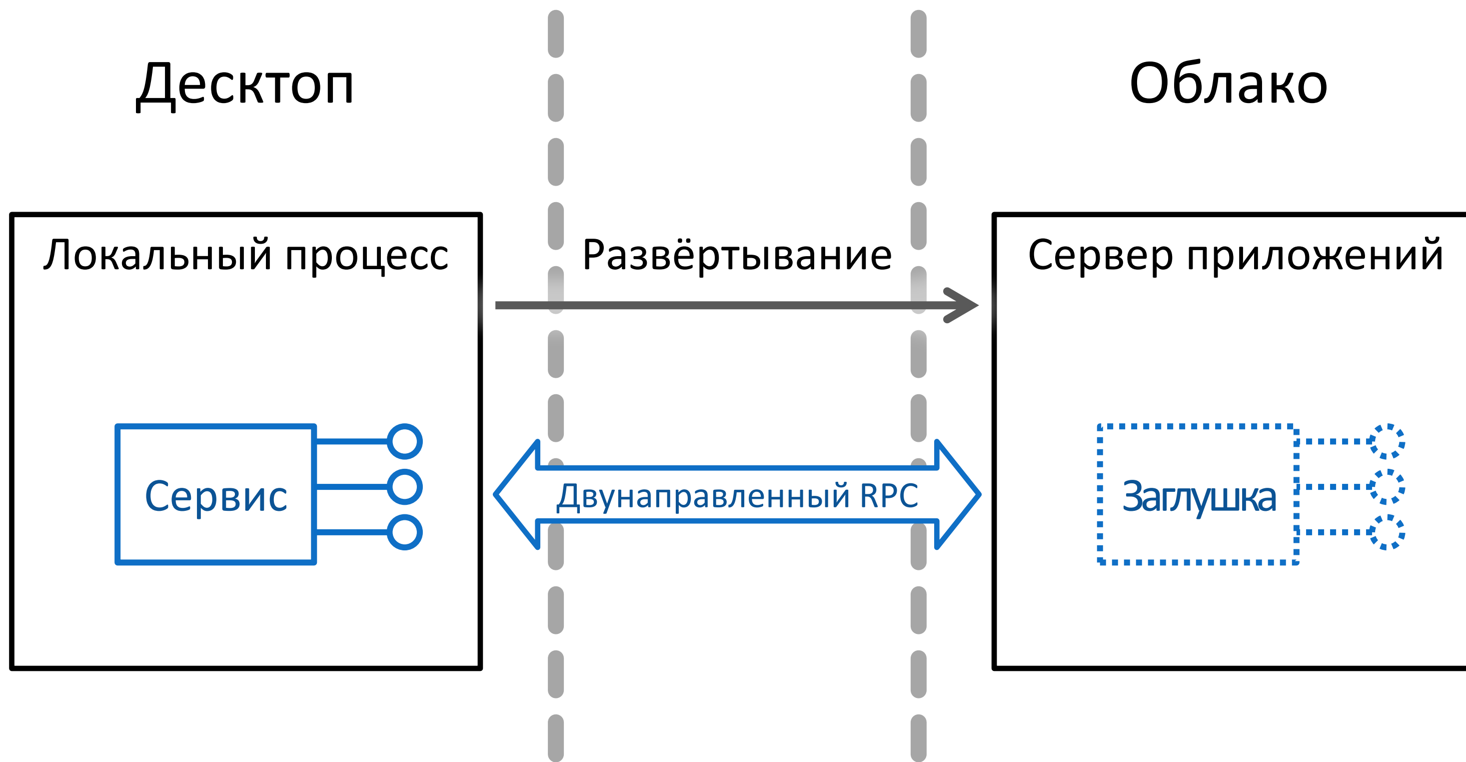
- Сборка
- Копирование артефактов (через Атлантику)
- Перезапуск
- Удалённая отладка ☹

Автоматизация

- Автоматизация обновления кода и перезапуска через SSH
- Прозрачный RMI, для локальной отладки “облачного” кода

Костыль делал своё дело, но был громоздким и не очевидным

“Размазанное” выполнение



Немного истории

2012 – работа с распределёнными кэшами

Нужно поднимать кластер в JUnit, сетевые узлы в рамках JVM эмулируются через ClassLoader

Тестовый “виртуальный” кластер

- Исключительно для локального тестирования
- Абстрактный API для работы с “узлами”
- Простая модель “распределённого” программирования

Получилось элегантно, но чисто для локального тестирования

Немного истории

2012 – работа с распределёнными кэшами

Нужно поднимать кластер в JUnit, сетевые узлы в рамках JVM эмулируются через ClassLoader

Тестовый “виртуальный” кластер

- Исключительно для локального тестирования
- Абстрактный API для работы с “узлами”
- Простая модель “распределённого” программирования

Получилось элегантно, но чисто для локального тестирования

Но затем, вопрос нагрузочного тестирования встал на повестке дня …

Источник вдохновения

```
$ ssh remote 'echo "Hey I am running on `hostname` "'
```

- пляски с кавычками
- синхронное выполнение
- кривая обработка ошибок

Network is the JVM

- У меня есть код
- Я хочу запускать его в “любой точке” сети (или нескольких)
- Выполнение должно быть изоморфно
- Сеть должна быть для программы родной средой!

Network is the JVM

Чем я не хочу заниматься?

- Заходить на удалённые машины
- Поддерживать “инфраструктуру”
- Переразвёртывать код, после изменения
- Воевать с сетью

Как в bash но на Java

```
$ ssh remote 'echo "Hey I am running on `hostname` "'
```

```
Cloud cloud = Nanocloud.createCloud();
cloud.x(SshConf.SSH)
    .configureSimpleRemoting();

cloud.node(“myserver”).exec(() -> {
    String localhost =
          InetAddress.getLocalHost().toString();
    System.out.println(“Hey I‘m on ” + localhost);
});
```

Как в bash но на Java

```
$ ssh remote 'echo "Hey I am running on `hostname` "'
```

```
Cloud cloud = Nanocloud.createCloud();
cloud.x(SshConf.SSH)
    .configureSimpleRemoting();

cloud.node("myserver").exec(() -> {
    String localhost =
          InetAddress.getLocalHost().toString();
    System.out.println("Hey I'm on " + localhost);
});
```

Код запускается на локальной машине как обычно

Как в bash но на Java

```
$ ssh remote 'echo "Hey I am running on `hostname` "'
```

```
Cloud cloud = Nanocloud.createCloud();
cloud.x(SshConf.SSH)
    .configureSimpleRemoting();

cloud.node("myserver").exec(() -> {
    String localhost =
          InetAddress.getLocalHost().toString();
    System.out.println("Hey I'm on " + localhost);
});
```

Конфигурация, должен быть настроен беспарольный SSH

Код запускается на локальной машине как обычно

Как в bash но на Java

```
$ ssh remote 'echo "Hey I am running on `hostname` "'
```

```
Cloud cloud = Nanocloud.createCloud();
cloud.x(SshConf.SSH)
    .configureSimpleRemoting();

cloud.node(“myserver”).exec(() -> {
    String localhost =
          InetAddress.getLocalHost().toString();
    System.out.println(“Hey I‘m on ” + localhost);
});
```

Конфигурация, должен быть настроен беспарольный SSH

Эти строки будут выполнены удалённо

Код запускается на локальной машине как обычно

Как в bash но на Java

+ параллельное выполнение
+ консистентная обработка ошибок
+ изоморфный код
+ экосистема

```
Cloud cloud = Nanocloud.createCloud();
cloud.x(SshConf.SSH)
    .configureSimpleRemoting();

cloud.node("myserver").exec(() -> {
    String localhost =
          InetAddress.getLocalHost().toString();
    System.out.println("Hey I'm on " + localhost);
});
```

Конфигурация, должен быть настроен беспарольный SSH

Эти строки будут выполнены удалённо

Код запускается на локальной машине как обычно

Чуть больше деталей

Что происходит под капотом?

```
Cloud cloud = Nanocloud.createCloud();
cloud.x(SshConf.SSH)
    .configureSimpleRemoting();


cloud.node("myserver").exec(() -> {
    String localhost =
      InetAddress.getLocalHost().toString();
    System.out.println("Hey! I'm on " + localhost);
});
```

Тип целевого узла (удалённая, SSH)

Используем текущий пользователь / SSH ключ для доступа

Чуть больше деталей

Что происходит под капотом?

```
Cloud cloud = Nanocloud.createCloud();
cloud.x(SshConf.SSH)
    .configureSimpleRemoting();
```

Тип целевого узла (удалённая, SSH)

Используем текущий пользователь / SSH ключ для доступа

Имя узла, используется как сетевое имя удалённого хоста

```
cloud.node("myserver").exec(() -> {
    String localhost =
      InetAddress.getLocalHost().toString();
    System.out.println("Hey! I'm on " + localhost);
});
```

- Соединение по SSH

Чуть больше деталей

Что происходит под капотом?

```
Cloud cloud = Nanocloud.createCloud();
cloud.x(SshConf.SSH)
    .configureSimpleRemoting();
```

Тип целевого узла (удалённая, SSH)

Используем текущий пользователь / SSH ключ для доступа

Имя узла, используется как сетевое имя удалённого хоста

```
cloud.node("myserver").exec(() -> {
    String localhost =
      InetAddress.getLocalHost().toString();
    System.out.println("Hey! I'm on " + localhost);
});
```

- Соединение по SSH
- Репликация локального класспути

Чуть больше деталей

Что происходит под капотом?

```
Cloud cloud = Nanocloud.createCloud();
cloud.x(SshConf.SSH)
    .configureSimpleRemoting();

cloud.node("myserver").exec(() -> {
    String localhost =
      InetAddress.getLocalHost().toString();
    System.out.println("Hey! I'm on " + localhost);
});
```

Тип целевого узла (удалённая, SSH)

Используем текущий пользователь / SSH ключ для доступа

Имя узла, используется как сетевое имя удалённого хоста

- Соединение по SSH
- Репликация локального класспути
- Запуск JVM на удаленном сервере

Чуть больше деталей

Что происходит под капотом?

```
Cloud cloud = Nanocloud.createCloud();
cloud.x(SshConf.SSH)
    .configureSimpleRemoting();
```

Тип целевого узла (удалённая, SSH)

Используем текущий пользователь / SSH ключ для доступа

Имя узла, используется как сетевое имя удалённого хоста

```
cloud.node("myserver").exec(() -> {
    String localhost =
      InetAddress.getLocalHost().toString();
    System.out.println("Hey! I'm on " + localhost);
});
```

- Соединение по SSH
- Репликация локального класспути
- Запуск JVM на удаленном сервере
- Установка соединения с удалённой JVM, завёрнутого в SSH сессию

Чуть больше деталей

```
Cloud cloud = Nanocloud.createCloud();
cloud.x(SshConf.SSH)
    .configureSimpleRemoting();
```

Тип целевого узла (удалённая, SSH)

Используем текущий пользователь / SSH ключ для доступа

Имя узла, используется как сетевое имя удалённого хоста

```
cloud.node("myserver").exec(() -> {
    String localhost =
      InetAddress.getLocalHost().toString();
    System.out.println("Hey! I'm on " + localhost);
});
```

Лямбда будет выполнена на удалённой машине

Что происходит под капотом?

- Соединение по SSH
- Репликация локального класспути
- Запуск JVM на удаленном сервере
- Установка соединения с удалённой JVM, завёрнутого в SSH сессию
- Отправка и запуск сериализованной лямбды на удалённой машине

Чуть больше деталей

```
Cloud cloud = Nanocloud.createCloud();
cloud.x(SshConf.SSH)
    .configureSimpleRemoting();

cloud.node("myserver").exec(() -> {
    String localhost =
      InetAddress.getLocalHost().toString();
    System.out.println("Hey! I'm on " + localhost);
});
```

Тип целевого узла (удалённая, SSH)

Используем текущий пользователь / SSH ключ для доступа

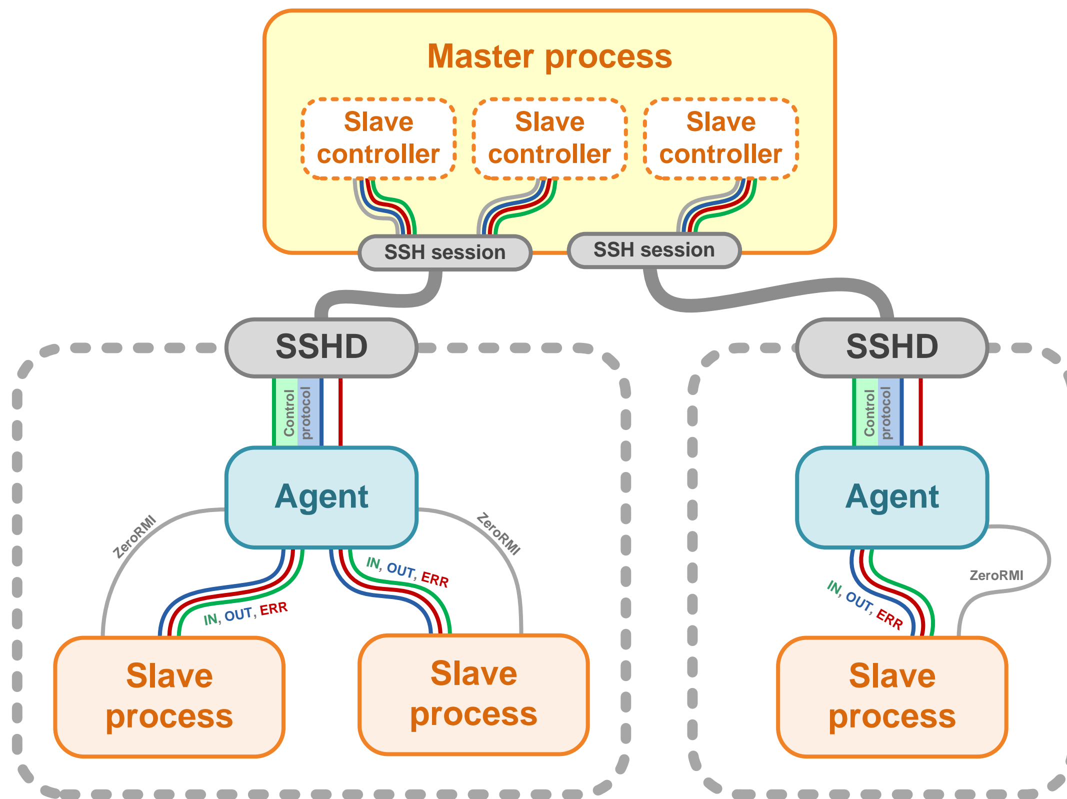
Имя узла, используется как сетевое имя удалённого хоста

Лямбда будет выполнена на удалённой машине

Что происходит под капотом?

- Соединение по SSH
- Репликация локального класспути
- Запуск JVM на удаленном сервере
- Установка соединения с удалённой JVM, завёрнутого в SSH сессию
- Отправка и запуск сериализованной лямбды на удалённой машине
- Перенаправление консольных потоков с удалённой JVM на локальную

Схема коммуникаций (SSH)



Nanocloud just works

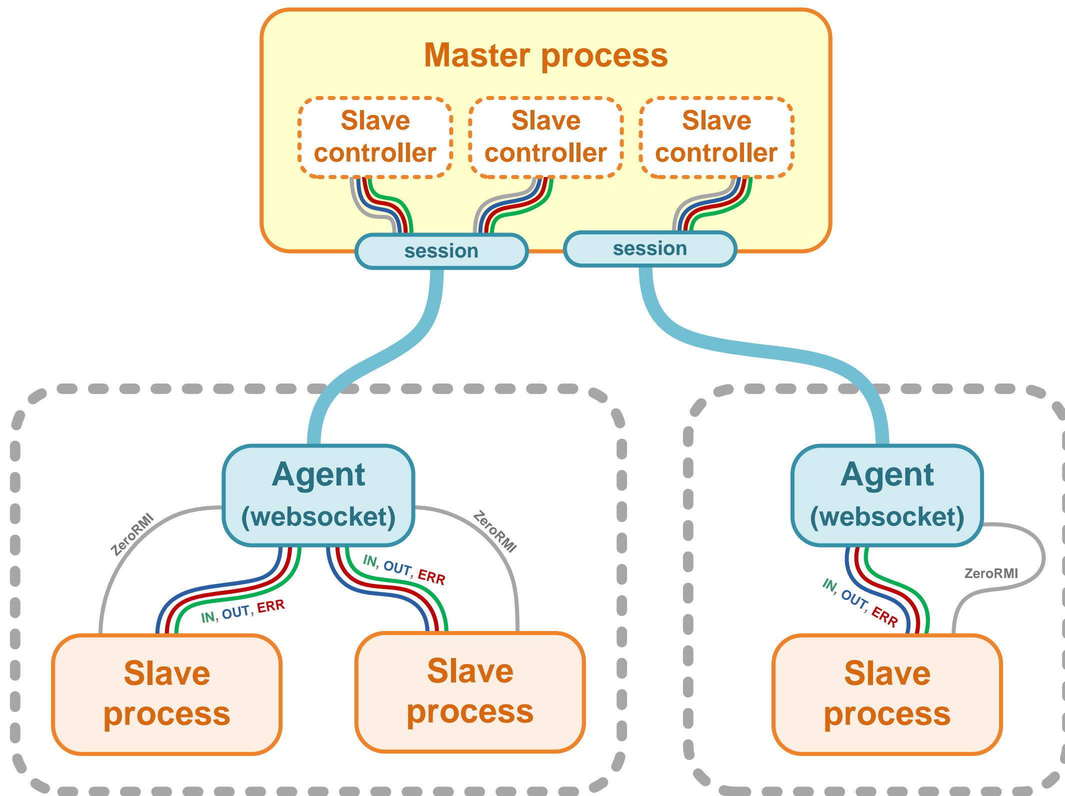
- Нужны только sshd и JRE, установка агента не нужна
- Использование конвенций SSH по умолчанию
- Автоматическое синхронизация classpath
  по всем узлам/процессам распределённой программы
- Одна консоль (std out, std err) для всей распределённой программы
- Завершение распределённой программы, завершает все её процессы

  любое завершение — всегда и все

  - Хартбиты
  - Завершение по разрыву канала связи
  - Принудительное завершение мастером (kill)

SSH is so 90ties

Схема коммуникаций (WS)



Прочие фишки

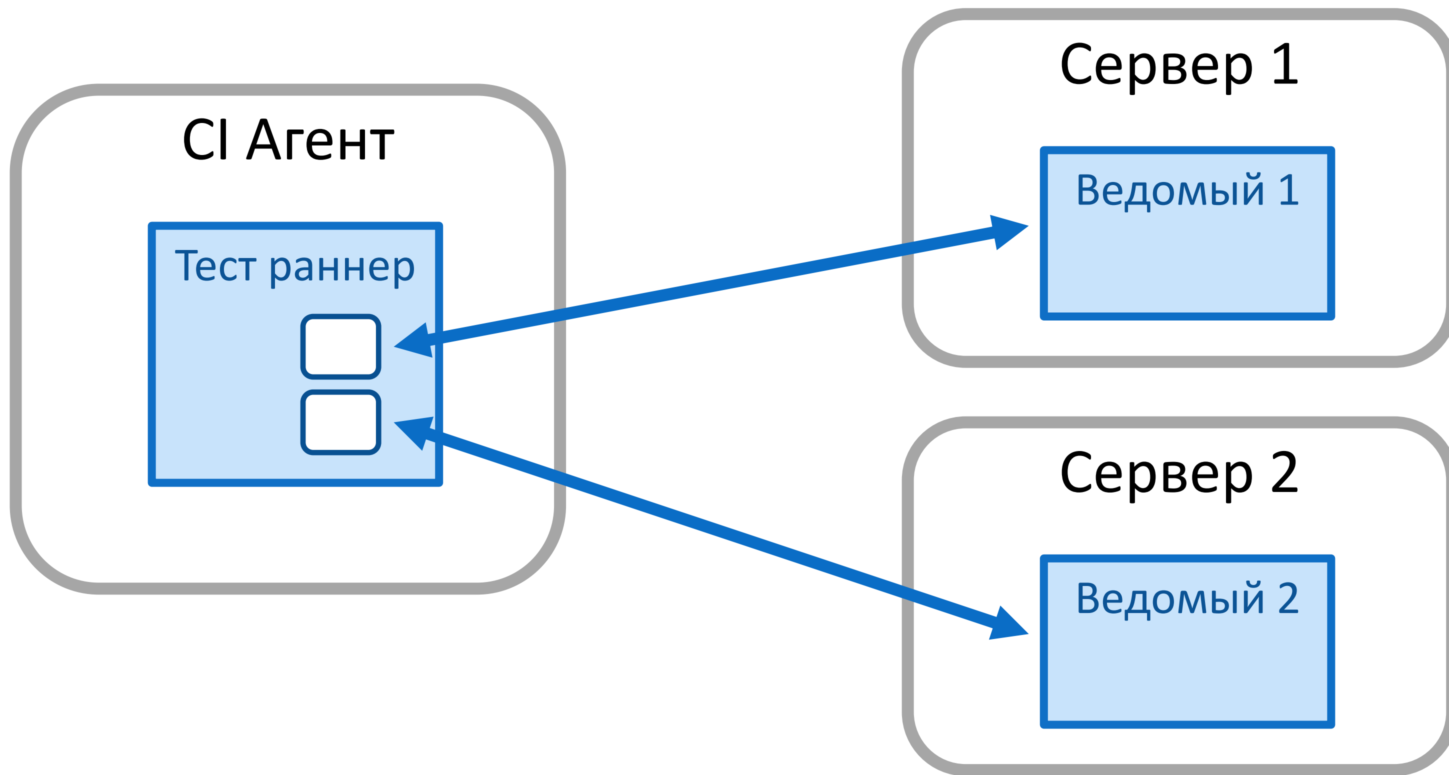
- Три режима целевых узлов
  classloader / локальный процесс / удалённый процесс (SSH, WS)
- Манипуляции с classpath (поддержка Maven)
  добавление/удаление библиотек
- Удаленные интерфейсы и двусторонний вызов удалённых методов
  как Java RMI, но проще и удобнее
- Инструментация кода
  например перехват System.exit
- Манипуляции с конфигурацией JVM
  например параметры памяти
- Библиотека распределённых примитивов

Подробнее в дискуссионной зоне!

Применение

- Сложное функциональное тестирование
  например, сетевой совместимости разных версий продукта
- Развёртывание рабочих сред
  Zookeeper, Hadoop, HBase, Coherence
- Распределённое нагрузочное тестирования
  утилизация кластера в 30 серверов как источника нагрузки
- Расширение возможностей CI
  мастер на Jenkins / ведомые в тестовом кластере

Ведомые CI узлы



Does it scale?

Нагрузочное тестирование
– сервис данных для вычислительного кластера

- Надо выбрать ~4Gbit трафика
- Ферма нагрузки 30 серверов
- ~100 процессов под управлением Nanocloud

Пришлось увеличить интервал heartbeat`ов
чтобы не DDoSить управляющий узел.

Супер-технология?

Nanocloud – оптимизирован по свои задачи

- Проста модель обработки отказов – ***FAIL FAST***
- Упрощённое управление ресурсами
  нет распределённой сборки мусора
- Жёсткий жизненный цикл
  завершение мастера – завершение всех ведомых

NanoCloud для нагрузочного тестирования

План моего куска доклада

План моего куска доклада

1. Контекст задачи нагрузочного тестирования онлайна JUG Ru Group

План моего куска доклада

1. Контекст задачи нагрузочного тестирования онлайна JUG Ru Group
2. Развертывание и запуск распределенной нагрузки на Gatling и NanoCloud

План моего куска доклада

1. Контекст задачи нагрузочного тестирования онлайна JUG Ru Group

2. Развертывание и запуск распределенной нагрузки на Gatling и NanoCloud

3. Результаты и выводы

Исторический контекст

Исторический контекст

1. 2019: Личный Кабинет - “держать 10K уников”

Исторический контекст

1. 2019: Личный Кабинет - “держать 10K уников”

2. 2020: первый онлайн и далее по 2 сезона в год

Исторический контекст

1. 2019: Личный Кабинет - “держать 10K уников”

2. 2020: первый онлайн и далее по 2 сезона в год

3. 2021: первые внешние митапы и конференции

Исторический контекст

1. 2019: Личный Кабинет - “держать 10K уников”
2. 2020: первый онлайн и далее по 2 сезона в год
3. 2021: первые внешние митапы и конференции
4. 2023: задумываемся о ЛК организатора и мультиарендности*

2023: Задача нагрузочного тестирования

2023: Задача нагрузочного тестирования

1. Убедиться что держим 10K одновременных уников

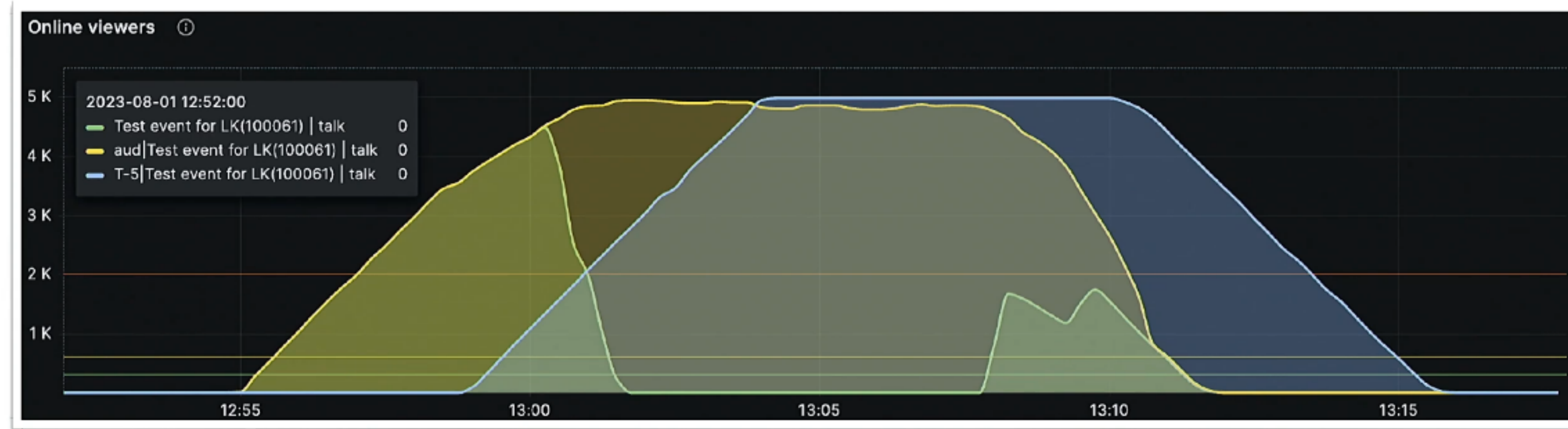
2023: Задача нагрузочного тестирования

1. Убедиться что держим 10K одновременных уников
2. Провести необходимые оптимизации и масштабирования

2023: The show must go on

The show must go on: инструменты нагрузки и рецепты оптимизации онлайн-конференции

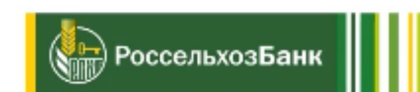
Оптимизации throughput: now_watching



83

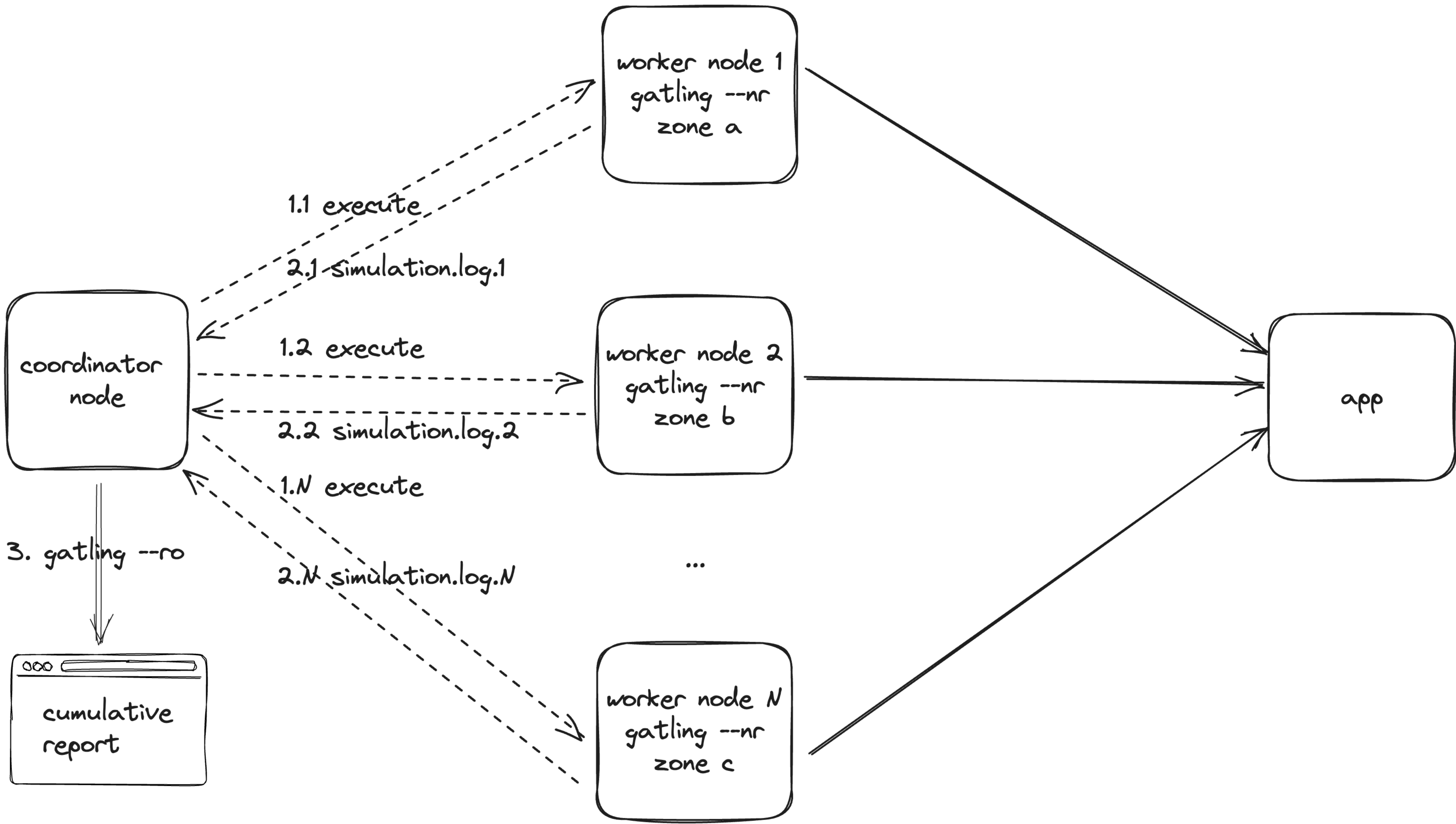
Владимир Красильщик

vlakra

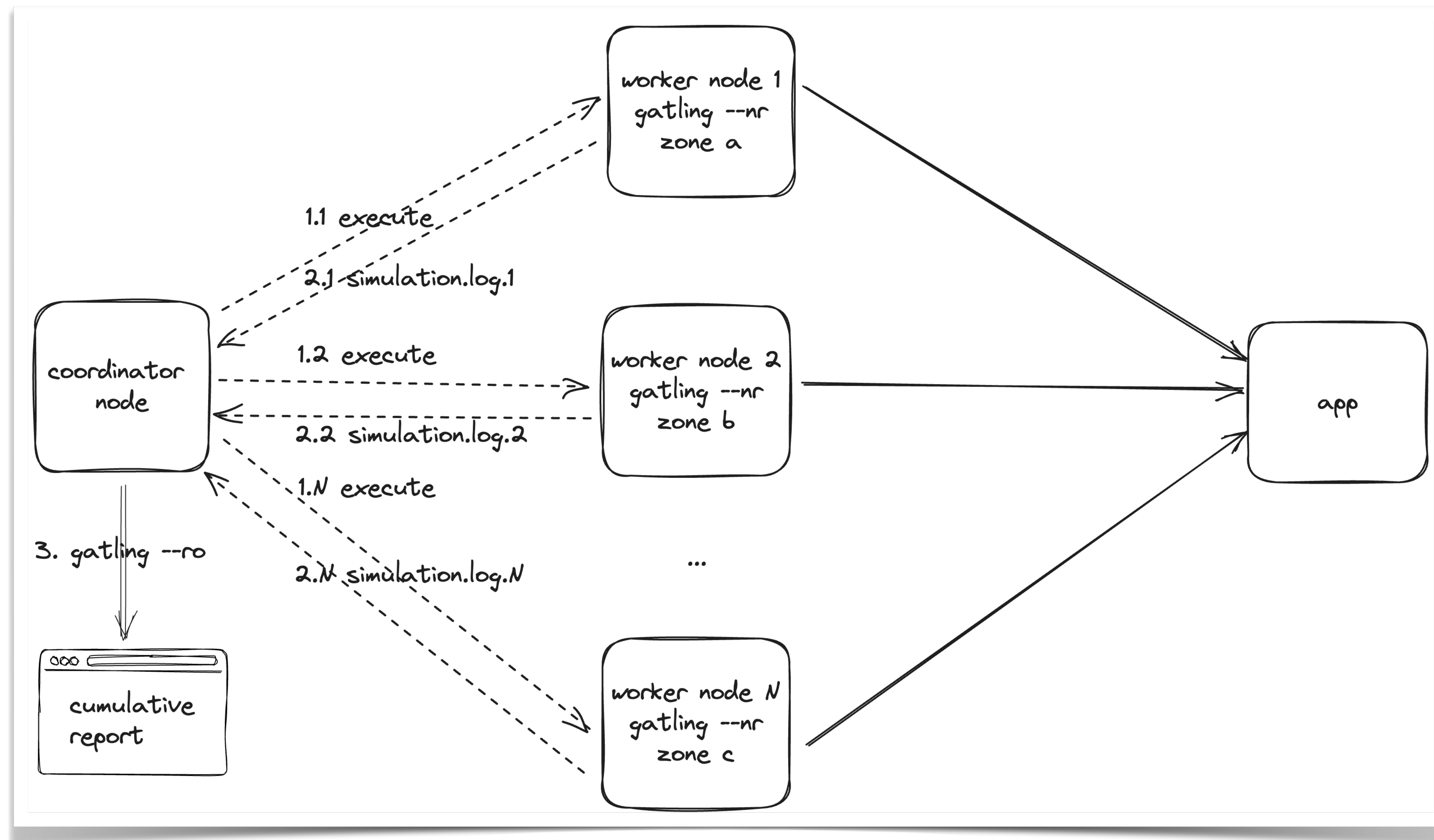


https://youtu.be/77ZWIMDUwfE

Запуск распределенной нагрузки из gatling-ов



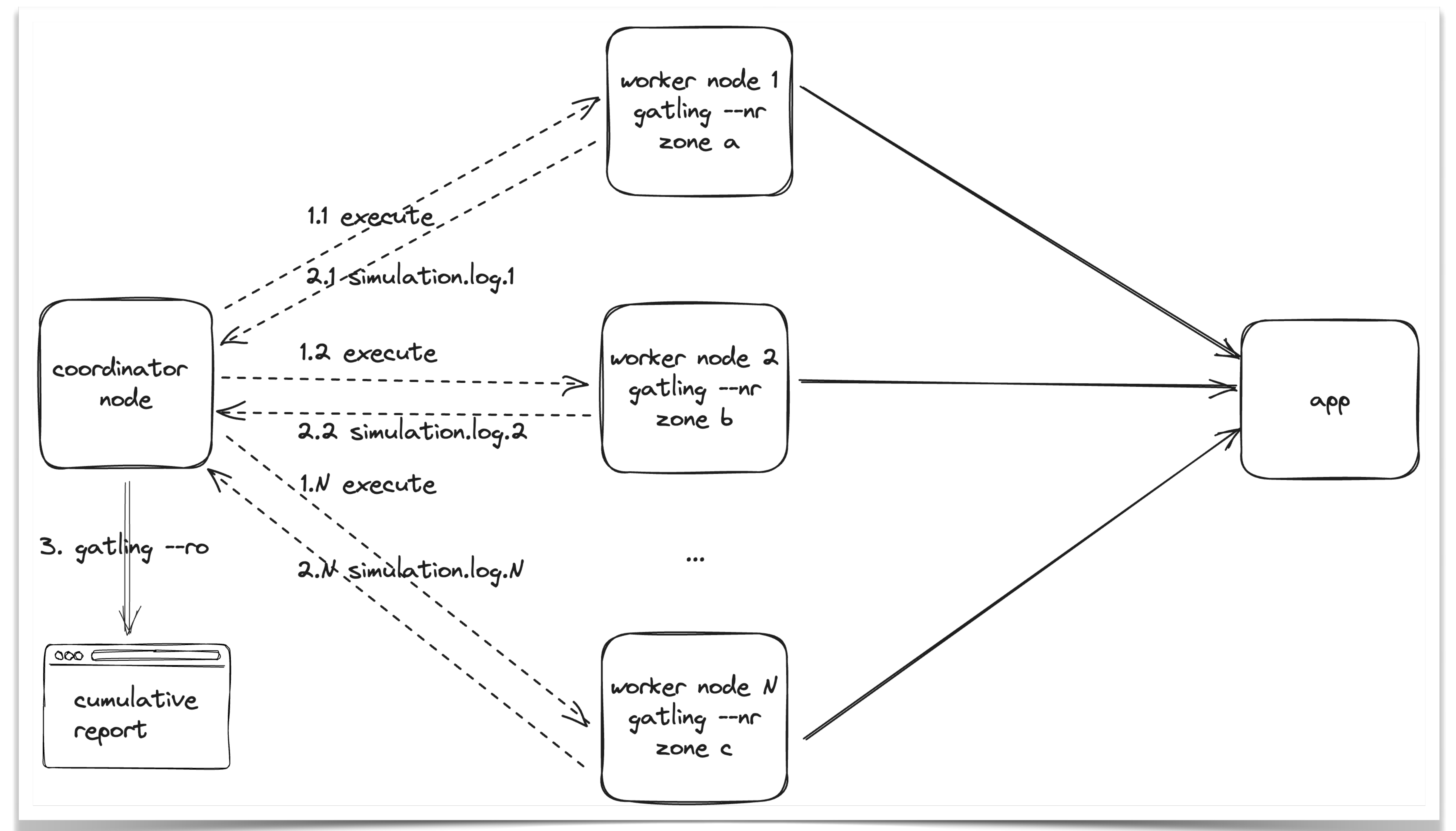
Запуск распределенной нагрузки в k8s?



1. dev -> prod, prod -> dev ?
2. ReplicaSet? Jobs? DaemonSet?
3. CI/CD ?

Запуск распределенной нагрузки с nanocloud

1. 9 виртуалок в Яндекс.Облаке: vCPU=10, RAM=20 ГБ, 50 ГБ, ~18р/ч каждая
2. JDK 17
3. ssh key
4. Координатор локальная тачка
5. yc cli



Конфигурация и запуск NanoCloud на рабочих узлах

```
Cloud cloud = Nanocloud.createCloud();
IntStream.range(0, totalWorkersNumber).forEach(i -> cloud.node("worker" + i));

if (Boolean.parseBoolean(dryRun)) {
    cloud.node("**").x(VX.TYPE).setLocal();
} else {
    IntStream.range(0, totalWorkersNumber).forEach(i ->
            RemoteNode.at(cloud.node("worker" + i)).x(SshConf.SSH)
                .setRemoteNodeType()
                .setRemoteHost(cloudHosts[i])
                .setRemoteAccount(cloudUsers[i])
                .setSshPrivateKey(cloudUsersPrivateKeysLocations[i])
    );
}
Map<String, String> commonEnvPropsProps = prepareCommonEnvProperties();
ViConf.JvmConf jvmConf = cloud.node("**").x(VX.PROCESS);
jvmConf.setProps(commonEnvPropsProps);
```

Конфигурация и запуск NanoCloud на рабочих узлах

```
Cloud cloud = Nanocloud.createCloud();
IntStream.range(0, totalWorkersNumber).forEach(i -> cloud.node("worker" + i));

if (Boolean.parseBoolean(dryRun)) {
    cloud.node("**").x(VX.TYPE).setLocal();
} else {
    IntStream.range(0, totalWorkersNumber).forEach(i ->
            RemoteNode.at(cloud.node("worker" + i)).x(SshConf.SSH)
                    .setRemoteNodeType()
                    .setRemoteHost(cloudHosts[i])
                    .setRemoteAccount(cloudUsers[i])
                    .setSshPrivateKey(cloudUsersPrivateKeysLocations[i])
    );
}
Map<String, String> commonEnvPropsProps = prepareCommonEnvProperties();
ViConf.JvmConf jvmConf = cloud.node("**").x(VX.PROCESS);
jvmConf.setProps(commonEnvPropsProps);
```

Конфигурация и запуск NanoCloud на рабочих узлах

```
Cloud cloud = Nanocloud.createCloud();
IntStream.range(0, totalWorkersNumber).forEach(i → cloud.node("worker" + i));

if (Boolean.parseBoolean(dryRun)) {
    cloud.node("**").x(VX.TYPE).setLocal();
} else {
    IntStream.range(0, totalWorkersNumber).forEach(i →
            RemoteNode.at(cloud.node("worker" + i)).x(SshConf.SSH)
                .setRemoteNodeType()
                .setRemoteHost(cloudHosts[i])
                .setRemoteAccount(cloudUsers[i])
                .setSshPrivateKey(cloudUsersPrivateKeysLocations[i])
    );
}
Map<String, String> commonEnvPropsProps = prepareCommonEnvProperties();
ViConf.JvmConf jvmConf = cloud.node("**").x(VX.PROCESS);
jvmConf.setProps(commonEnvPropsProps);
```

Конфигурация и запуск NanoCloud на рабочих узлах

```
Cloud cloud = Nanocloud.createCloud();
IntStream.range(0, totalWorkersNumber).forEach(i → cloud.node("worker" + i));

if (Boolean.parseBoolean(dryRun)) {
    cloud.node("**").x(VX.TYPE).setLocal();
} else {
    IntStream.range(0, totalWorkersNumber).forEach(i →
            RemoteNode.at(cloud.node("worker" + i)).x(SshConf.SSH)
                    .setRemoteNodeType()
                    .setRemoteHost(cloudHosts[i])
                    .setRemoteAccount(cloudUsers[i])
                    .setSshPrivateKey(cloudUsersPrivateKeysLocations[i])
    );
}
Map<String, String> commonEnvPropsProps = prepareCommonEnvProperties();
ViConf.JvmConf jvmConf = cloud.node("**").x(VX.PROCESS);
jvmConf.setProps(commonEnvPropsProps);
```

Конфигурация и запуск NanoCloud на рабочих узлах

```
Cloud cloud = Nanocloud.createCloud();
IntStream.range(0, totalWorkersNumber).forEach(i → cloud.node("worker" + i));

if (Boolean.parseBoolean(dryRun)) {
    cloud.node("**").x(VX.TYPE).setLocal();
} else {
    IntStream.range(0, totalWorkersNumber).forEach(i →
            RemoteNode.at(cloud.node("worker" + i)).x(SshConf.SSH)
                .setRemoteNodeType()
                .setRemoteHost(cloudHosts[i])
                .setRemoteAccount(cloudUsers[i])
                .setSshPrivateKey(cloudUsersPrivateKeysLocations[i])
    );
}
Map<String, String> commonEnvPropsProps = prepareCommonEnvProperties();
ViConf.JvmConf jvmConf = cloud.node("**").x(VX.PROCESS);
jvmConf.setProps(commonEnvPropsProps);
```

Конфигурация и запуск nanocloud на рабочих узлах

```
String nowAsIsoString =
      DateFormatUtils.format(Date.from(Instant.now()), "yyyy-MM-dd-HH-mm-ss");

IntStream.range(0, totalWorkersNumber).forEach(i -> {
  Path resFolder = Path.of("results", nowAsIsoString + "-worker" + i + "-result");
  cloud.node("worker" + i).setProp("loadtool.thisWorkerNumber", String.valueOf(i));
  cloud.node("worker" + i).setProp("gatling.resultsDirectory", resFolder.toString());
});

cloud.node("**").touch();

MassResult<Map<String, Object>> result = cloud.
        .node("**")
        .massCalc(CloudLauncher::runGatlingSimulationAndReturnGzippedLogs);
Iterable<Map<String, Object>> similationGzippedLogsResult = result.results();
```

Конфигурация и запуск nanocloud на рабочих узлах

```java
String nowAsIsoString =
      DateFormatUtils.format(Date.from(Instant.now()), "yyyy-MM-dd-HH-mm-ss");

IntStream.range(0, totalWorkersNumber).forEach(i -> {
  Path resFolder = Path.of("results", nowAsIsoString + "-worker" + i + "-result");
  cloud.node("worker" + i).setProp("loadtool.thisWorkerNumber", String.valueOf(i));
  cloud.node("worker" + i).setProp("gatling.resultsDirectory", resFolder.toString());
});

cloud.node("**").touch();

MassResult<Map<String, Object>> result = cloud.
        .node("**")
        .massCalc(CloudLauncher::runGatlingSimulationAndReturnGzippedLogs);
Iterable<Map<String, Object>> similationGzippedLogsResult = result.results();
```

Конфигурация и запуск nanocloud на рабочих узлах

```
String nowAsIsoString =
      DateFormatUtils.format(Date.from(Instant.now()), "yyyy-MM-dd-HH-mm-ss");

IntStream.range(0, totalWorkersNumber).forEach(i -> {
  Path resFolder = Path.of("results", nowAsIsoString + "-worker" + i + "-result");
  cloud.node("worker" + i).setProp("loadtool.thisWorkerNumber", String.valueOf(i));
  cloud.node("worker" + i).setProp("gatling.resultsDirectory", resFolder.toString());
});
```

```
cloud.node("**").touch();
```

```
MassResult<Map<String, Object>> result = cloud.
        .node("**")
        .massCalc(CloudLauncher::runGatlingSimulationAndReturnGzippedLogs);
Iterable<Map<String, Object>> similationGzippedLogsResult = result.results();
```

Конфигурация и запуск nanocloud на рабочих узлах

```
String nowAsIsoString =
      DateFormatUtils.format(Date.from(Instant.now()), "yyyy-MM-dd-HH-mm-ss");

IntStream.range(0, totalWorkersNumber).forEach(i -> {
  Path resFolder = Path.of("results", nowAsIsoString + "-worker" + i + "-result");
  cloud.node("worker" + i).setProp("loadtool.thisWorkerNumber", String.valueOf(i));
  cloud.node("worker" + i).setProp("gatling.resultsDirectory", resFolder.toString());
});
```

```
cloud.node("**").touch();
```

```
MassResult<Map<String, Object>> result = cloud.
        .node("**")
        .massCalc(CloudLauncher::runGatlingSimulationAndReturnGzippedLogs);
Iterable<Map<String, Object>> similationGzippedLogsResult = result.results();
```

Запуск gatling на рабочем узле

```
public static Map<String, Object> runGatlingSimulationAndReturnGzippedLogs()
throws Exception {

    String jvmName = ManagementFactory.getRuntimeMXBean().getName();

    launchGatling();                                                          //1

    Path simulationLogPathGzip = locateAndGzipSimulationLog();                //2

    byte[] simulationLogGzipBytes = Files.readAllBytes(simulationLogPathGzip);//3

    return Map.of(                                                            //4
        "jvmName", jvmName,
        "simulationLogGzipBytes", simulationLogGzipBytes
    );

}
```

Запуск gatling на рабочем узле

```
public static void launchGatling() {
    GatlingPropertiesBuilder props = new GatlingPropertiesBuilder()
            .simulationClass(gatlingSimulationClass)
            .runDescription(gatlingRunDescription)
            .noReports()
            .resultsDirectory(gatlingResultsDirectory);
   Gatling.fromMap(props.build());
}
```

Запуск gatling на рабочем узле

```
public static void launchGatling() {
    GatlingPropertiesBuilder props = new GatlingPropertiesBuilder()
            .simulationClass(gatlingSimulationClass)
            .runDescription(gatlingRunDescription)
            .noReports()
            .resultsDirectory(gatlingResultsDirectory);
   Gatling.fromMap(props.build());
}
```

Запуск gatling на рабочем узле

```
public static Map<String, Object> runGatlingSimulationAndReturnGzippedLogs()
throws Exception {

    String jvmName = ManagementFactory.getRuntimeMXBean().getName();

    launchGatling();                                                     //1

    Path simulationLogPathGzip = locateAndGzipSimulationLog();           //2

    byte[] simulationLogGzipBytes = Files.readAllBytes(simulationLogPathGzip);//3

    return Map.of(                                                       //4
        "jvmName", jvmName,
        "simulationLogGzipBytes", simulationLogGzipBytes
    );

}
```

Построение кумулятивного отчета

```
MassResult<Map<String, Object>> result = cloud.
        .node("**")
        .massCalc(CloudLauncher::runGatlingSimulationAndReturnGzippedLogs);
Iterable<Map<String, Object>> similationGzippedLogsResult = result.results();

ungzipRawSimulationsLogsFromAllWorkers(simLogGzipList, execResultsDir); //1

Gatling.fromMap(new GatlingPropertiesBuilder()                           //2
        .reportsOnly(execResultsDir)
        .build());

cloud.shutdown();                                                         //3
```

Построение кумулятивного отчета

```
MassResult<Map<String, Object>> result = cloud.
        .node("**")
        .massCalc(CloudLauncher::runGatlingSimulationAndReturnGzippedLogs);
Iterable<Map<String, Object>> similationGzippedLogsResult = result.results();

ungzipRawSimulationsLogsFromAllWorkers(simLogGzipList, execResultsDir); //1

Gatling.fromMap(new GatlingPropertiesBuilder()                          //2
        .reportsOnly(execResultsDir)
        .build());

cloud.shutdown();                                                       //3
```

Построение кумулятивного отчета

```
MassResult<Map<String, Object>> result = cloud.
        .node("**")
        .massCalc(CloudLauncher::runGatlingSimulationAndReturnGzippedLogs);
Iterable<Map<String, Object>> similationGzippedLogsResult = result.results();

ungzipRawSimulationsLogsFromAllWorkers(simLogGzipList, execResultsDir); //1

Gatling.fromMap(new GatlingPropertiesBuilder()                           //2
        .reportsOnly(execResultsDir)
        .build());

cloud.shutdown();                                                        //3
```

Выводы и результаты

Выводы и результаты

1. Не k8s единым: KISS and NanoCloud

Выводы и результаты

1. Не k8s единым: KISS and NanoCloud

2. Gatling + NanoCloud + 9 виртуалок = “My Gatling Enterprise”

Выводы и результаты

1. Не k8s единым: KISS and NanoCloud

2. Gatling + NanoCloud + 9 виртуалок = “My Gatling Enterprise”

3. “My Gatling Enterprise” = Min (тест -> поиск узкого места -> оптимизация -> тест)

Nanocloud для интеграционного тестирования

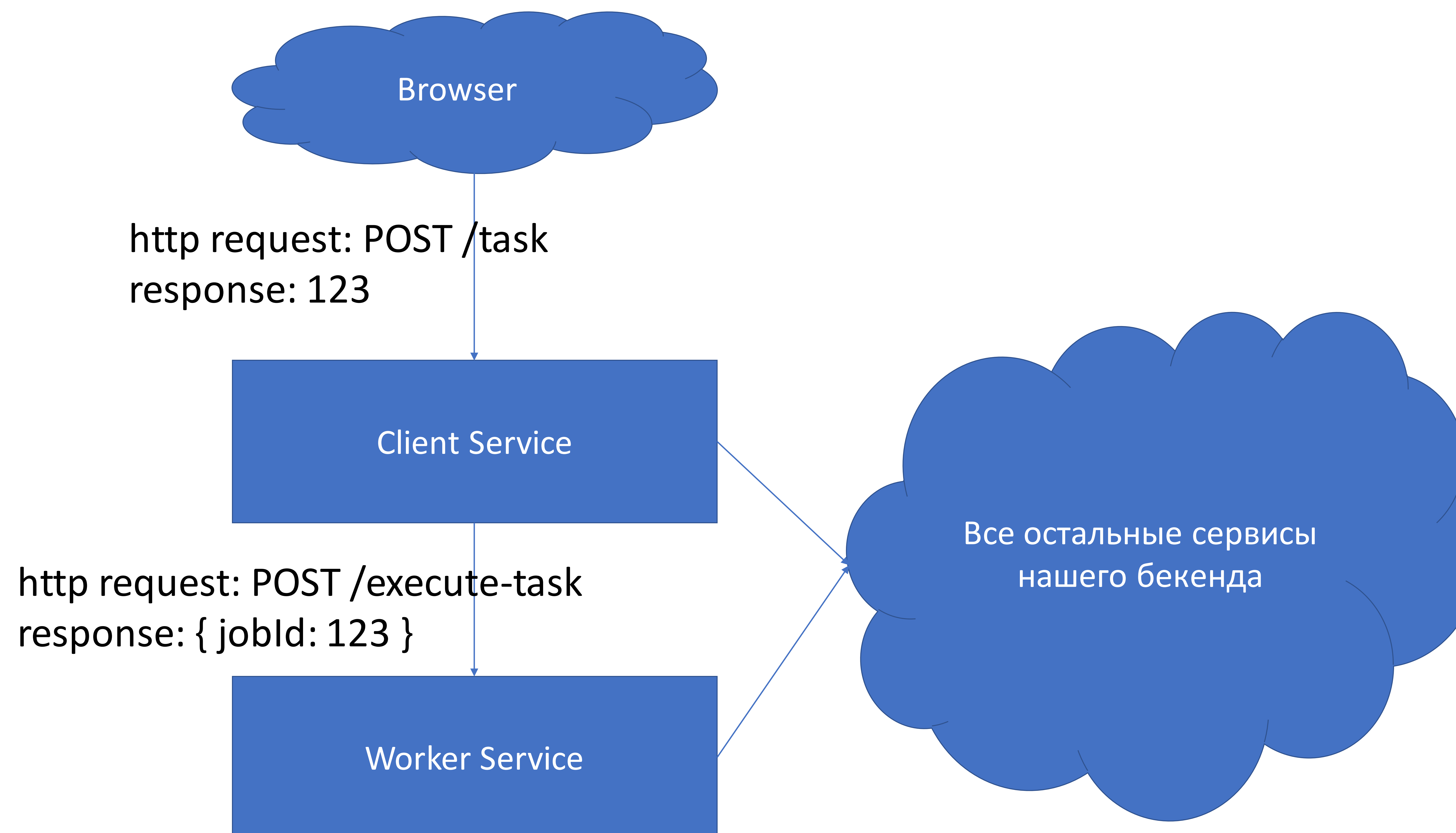
Интеграционное тестирование во время разработки

- Если у вас микросервисная архитектура
- И задачи как правило затрагивают несколько сервисов
- То чтобы протестировать результат, надо запустить больше одного процесса

Пример микросервисного проекта

Git: client-service,
получает запрос от клиента,
валидирует, обогащает,
отправляет в worker-service

Git: worker-service
получает запрос,
что-то делает



Как обычно устроен процесс

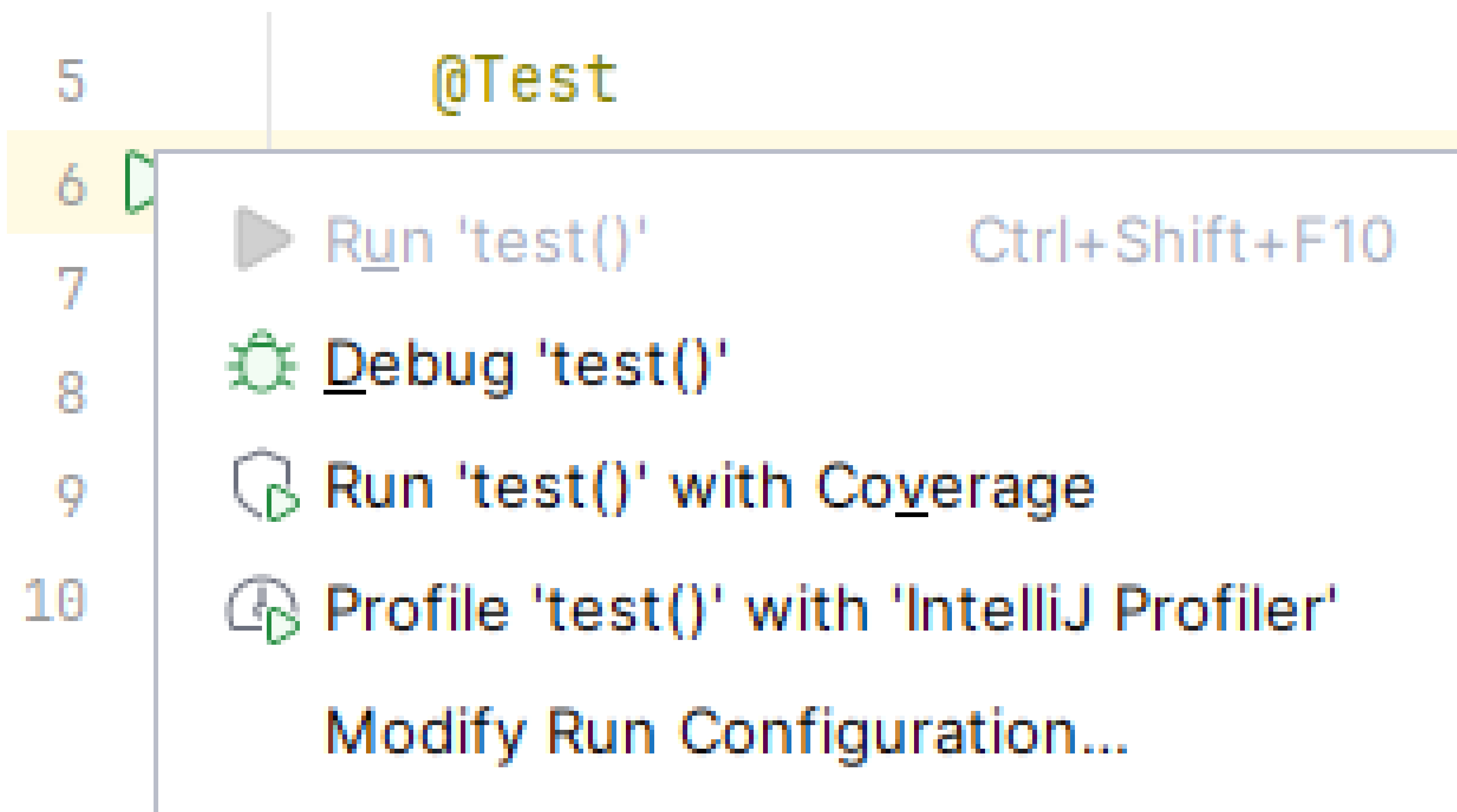
1. Написание кода в каждом сервисе
2. Цикл: юнит тестирование <-> исправления… *секунды*
3. Коммит и сборка… *минуты*
4. Деплой на тестовое окружение… *часы, если занято*
5. Тестирование на тестовом окружении.
6. goto 1

Собрать и задеплоить на какое-то окружение?

- Долго
- Должно быть личное окружение или их должно быть много (а то будем мешать друг другу)
- Не очень просто подпихнуть моки, чтобы управлять поведением внешнего сервиса (тестируем как правило хорошие сценарии)

Интеграционные тесты в идеальном мире

- Должны запускаться одной кнопкой run test из IDE



Интеграционные тесты в идеальном мире

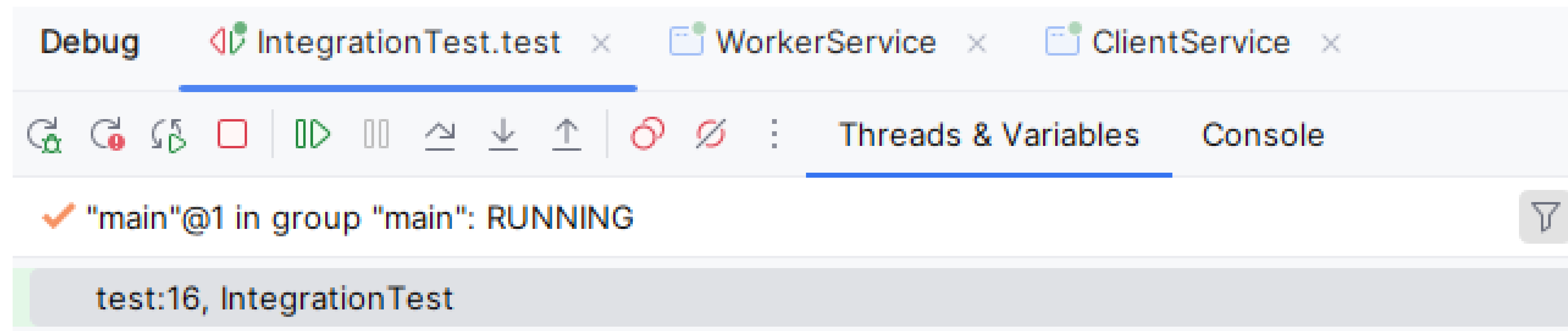
- Написание не должно представлять трудностей и не должно сильно отличаться от написания юнит-тестов.

```
@Test
public void test() throws Exception {
    ClientService client = createClient();
    WorkerService worker = createWorker();

    client.post("/task", "{...}");
    List<Task> tasks = worker.getExecutedTasks();

    Assertions.assertThat(tasks).hasSize(1);
}
```

Интеграционные тесты в идеальном мире

- Должны поддерживать отладку отдельных микросервисов

```
2    public void test(){
3        ClientService client = createClient();   client: ClientService@776
4        WorkerService worker = createWorker();   worker: WorkerService@777
5
         client.post(url: "/task", json: "{...}");   client: ClientService@776
```



Интеграционные тесты в идеальном мире

- Должны быть изолированы друг от друга

```
@Test
public void test() throws Exception {
    client.post("/task", "{...}");
    List<Task> tasks = worker.getExecutedTasks();
    Assertions.assertThat(tasks).hasSize(1);
}

@Test
public void test2() throws Exception {
    client.post("/task", "{...}");
    List<Task> tasks = worker.getExecutedTasks();
    Assertions.assertThat(tasks).hasSize(1);
}
```

Интеграционные тесты в идеальном мире

- Запуск теста должен быть достаточно быстрый, чтобы при разработке можно было пользоваться практикой TDD

Test	Time
IntegrationTest	25 sec 74 ms
testFlow1	10 sec 68 ms
testFlow2	15 sec 6 ms

Интеграционные тесты в идеальном мире

- Надо уметь мокать «ненужные сервисы»
- Надо уметь проверять внутренние параметры

```
public void test() throws Exception {
    /*...*/
      Mockito.when(worker.usersDao.findUserByUsername("agent.smith"))
           .thenReturn(new User("agent.smith"));
}
```

Интеграционные тесты в идеальном мире

Должны быть интегрированы с CI. Идеально, чтобы их можно было прогонять при проверке пул реквестов.

- То есть желательна независимость от другой инфраструктуры
- Возможность гнать в параллель
  - Как тесты одной бранчи
  - Так и тесты с разных бранчей

Пример микросервисного проекта:

Backend, делается нами

Client Service

Worker Service

client-service
- src/main
- src/resources
- tests/main
- …
- build.gradle/pom.xml

worker-service
- src/main
- src/resources
- tests/main
- …
- build.gradle/pom.xml

Объединяем в один проект

- Используем монорепу
- Или используем git-submodule, чтобы получить один репозиторий
  - git submodule add https://..../client-service
  - git submodule add https://..../worker-service
- Или просто договариваемся о правилах чекаута

- Итоговая структура:
  - integration-tests-project
    - client-service
    - worker-service
    - integration-tests
    - pom.xml / build.gradle

Объединяем в один проект (maven)

- Главный pom.xml

```
<modules>
    <module>client-service</module>
    <module>worker-service</module>
    <module>integration-tests</module>
</modules>
```

- Тестовый pom.xml

```
<dependency>
    <artifactId>client-service</artifactId>
</dependency>
<dependency>
    <artifactId>worker-service</artifactId>
</dependency>
```

Объединяем в один проект (gradle)

- Главный settings.gradle

```
rootProject.name=parentTests
include ‘worker-service’, ‘client-service’, ‘integration-tests’
```

- Тестовый build.gradle

```
dependencies {
    implementation project(':worker-service’)
    implementation project(':client-service’)
}
```

Все готово?

- У нас появился модуль, зависящий от наших сервисов
- Если мы в нем напишем тест и запустим его, то IDE сперва построит наши сервисы
- Можем ли мы запустить прямо в тесте несколько приложений?

```
@Test
public void testSomething() {
    ClientServiceApplication.main(new String[0]);
    WorkerServiceApplication.main(new String[0]);
    … // some assertions
}
```

Все готово?

- У нас появился модуль, зависящий от наших сервисов
- Можем ли мы запустить прямо в тесте несколько приложений? Скорее всего нет!
  - Ресурсы (в т.ч. application.yml) будут пересекаться
  - Различие в класспасе приведет к тому, что мы будем тестировать монстра Франкенштейна
    - Например, добавление spring-security в один сервис, изменит и поведение второго в тестах
- Нам надо запускать сервисы в отдельных процессах с правильными classpath!

Решаем проблему с classpath

- Нам надо запускать сервисы в отдельных процессах с правильными classpath!
  - Надо получить список зависимостей для каждого проекта в отдельности
    - Обратимся к системе сборки
  - Надо запустить процессы с правильным списком зависимостей
    - Тут поможет nanocloud
  - Надо мокать чужие сервисы
    - Тут тоже поможет nanocloud

Узнаем список зависимостей каждого проекта

- Maven
  - maven-dependency-plugin:build-classpath
  - Прописываем в pom в предположении, что класспасс меняется редко
  - Вычитываем в тесте из получившегося файла.
  - Можно еще: скопировать pom в target и проверять, что не изменился

- Gradle
  - Добавляем таску

```
task writeClasspath << {
   buildDir.mkdirs()
   new File(buildDir, "classpath.txt").text = configurations.runtime.asPath
}
```

  - Или ставим таску тестов в зависимость от таски сборки библиотек и используем результат сборки

Запускаем сервисы в отдельных процессах

```
Cloud cloud = Nanocloud.createCloud(); //1

ViNode clientService = cloud.node("client-service"); //2

ViProps.at(clientService).setLocalType(); //3

clientService.x(VX.CLASSPATH).inheritClasspath(false); //4

classpathEntries.forEach(clientService.x(VX.CLASSPATH)::add); //5

clientService.exec(() -> ClientServiceApplication.main(new String[0])); //6
```

Мокаем сервисы

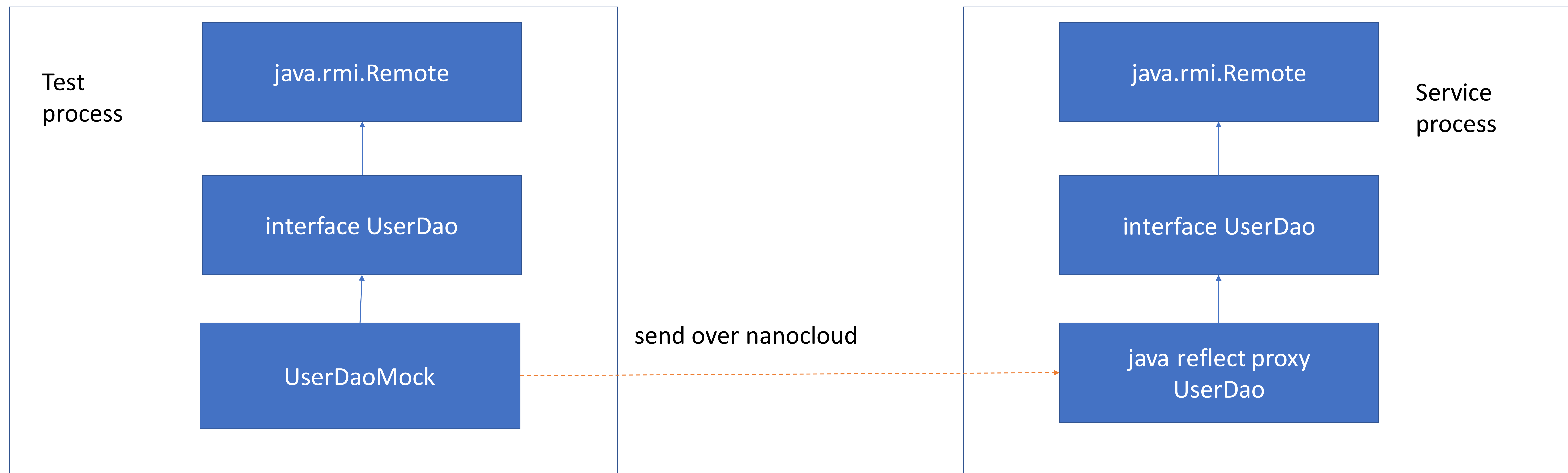
```java
interface UsersDao {
    User findUserByUsername(String id);
}

@Primary
class UserDaoDelegate implements IntegrationTest.UsersDao {
    static IntegrationTest.UsersDao delegate;

    @Override
    public User findUserByUsername(String id) {
        return delegate.findUserByUsername(id);
    }
}
```

Мокаем сервисы: моки на стороне сервиса

```
clientService.exec(() -> {

    RemoteUsersDao remoteUsersDao = Mockito.mock(RemoteUsersDao.class);
    Mockito
            .when(remoteUsersDao.findUserByUsername("agent.smith"))
            .thenReturn(new User());

    UserDaoDelegate.delegate = remoteUsersDao;
    Assert.assertNotNull(remoteUsersDao.findUserByUsername("agent.smith"));
});
```

Мокаем сервисы: моки на стороне теста

- Два вида сериализации в nanocloud:
  - java.io.Serializable – будет сериализовано обычно
  - java.rmi.Remote – будет сериализовано в виде rmi заглушки



Мокаем сервисы: моки на стороне теста

```
interface RemoteUsersDao extends UsersDao, Remote {}


@Test
public void testFlow1() throws InterruptedException {
    RemoteUsersDao remoteUsersDao = Mockito.mock(RemoteUsersDao.class);

    Mockito.when(remoteUsersDao.findUserByUsername("agent.smith")).thenReturn(new User());

    clientService.exec(() -> {
        UserDaoDelegate.delegate = remoteUsersDao;
        Assert.assertNotNull(remoteUsersDao.findUserByUsername("agent.smith"));
    });
…
```

Отлаживаем сервисы

```
clientService.x(VX.JVM)
    .addJvmArg("-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=30602");
```

Console

Tests passed: 0 of 1 test

```
[client-service] Listening for transport dt_socket at address: 30602 Attach debugger
```

Запускаем интеграционный тест

- Запускаем как обычный unit-test
- IDE сама скомпилирует все зависимости
- Если тест упал, исправляем и быстро запускаем снова
  - не нужно собирать и деплоить
- Дебажим при необходимости
- Если нужны внешние системы:
  - мокаем
  - или дополняем testcontainers
- Бонус: можно тестировать отказы и падения не ломая окружения

Полезные дополнения и ссылки

- Полезные дополнения
  - Testcontainers https://testcontainers.com/
    - Позволят добавить базу, очередь и многое другое в тест
  - Sniffy https://www.sniffy.io/
    - Позволит добавить сетевые задержки, разрывы соединения и другое
    - Если хватает функциональности, то более удобно, чем tc

- Статья с примерами кода по интеграционному тестированию
  - https://habr.com/ru/articles/682420/

Nanocloud для тестирования в специфичном окружении

Тестирование в специфичном окружении

- Если нужна конкретная операционная система или конкретное железо
  - Определенная операционная система или предустановленное ПО
  - Определенная видеокарта или другое железо
- Два сценария
  - Нельзя запускать тесты во время разработки (машина разработчика не подходит)
  - Нельзя запускать тесты во время CI (дефицит специфических боксов)

Если не подходит машина девелопера

- Тесты запускаются вручную, а значит не очень часто
- Можно подготовить несколько боксов, шарить их между сотрудниками
- Вручную настроить,
  - запустить там nanocloud ws agent
  - или установить несекретный пароль для ssh
  - или прописать сертификаты всех разработчиков в ssh

Если мало боксов с нужным железом

- Не хочется прогонять весь билд на дефицитном железе
- Решение от ДевОпс:
  - Настроить TeamCity Build Chain – довольно сложно, но можно решить все проблемы
- Решение с nanocloud зависит от количества тестов и конкуренции
  - Если мало – делаем как для девелоперских тачек
  - Если много – нужно ограничение.

Ограничение конкурентности

- Например, с помощью File Locks

```
testBox.exec(() -> {
    try (RandomAccessFile file = new RandomAccessFile("lockfile.txt", "rw");
         FileChannel channel = file.getChannel();
         FileLock lock = channel.lock()) {

        doTest();

    }
});
```

Итог

Nanocloud позволяет создавать java процессы локально и удаленно и управлять ими

Мы показали вам несколько примеров использования

- Распределение нагрузки при нагрузочном тестировании
- Изолирование сервисов при интеграционном тестировании
- Выполнение кода на специализированном окружении

Это только несколько идей, как использовать nanocloud.

Пробуйте!

Спасибо за внимание
и до встречи на дискуссии!

alexey.ragozin@gmail.com

blog.ragozin.info

https://github.com/aragozin

https://github.com/gridkit/nanocloud

https://aragozin.timepad.ru

vladimir.krasilschik@gmail.com

@vlakra

fuudtorrentsru@gmail.com