Что такое CI & CD, аналоги Jenkins

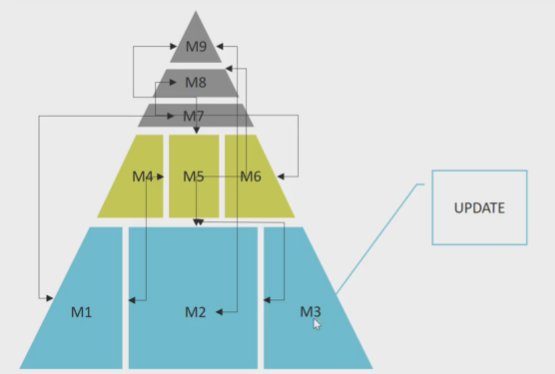
**CI & CD concept**

**Why need CI & CD**

**Component Integration idea**

**Приложение состоит из трех основных уровней:**

1. Бекенд. Там могут быть бд, ядро приложения и т.д;
2. Веб-сервеса. Позволяют юзер-интерфейсу подключаться к ним и выполнять работу с помощью бекенда;
3. Юзер-интерфейс. Приложения (под андроид, айфон, тп).



Обновление одного куска затрагивает другие.

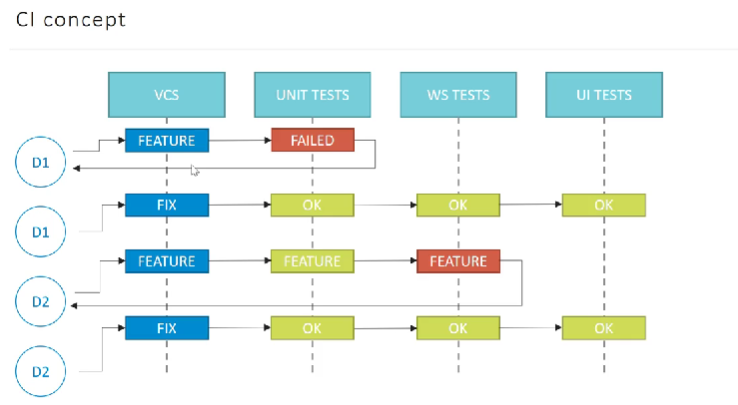
Для того, чтобы решить проблему, когда мы обнаруживаем какие-то дефекты, недопонимания и т.д. и фиксить уже поздно, придумали практику **Continuous Integration** (непрерывная интеграция).

**CI & CD testing**

|  |
| --- |
| **Как работает Continuous Integration и Continuous Delivery? И чем это может решить проблему, которая возникает во время Integration face?** |

**Continuous Integration** дефолятах

Концепция системы непрерывной интеграции. Интеграцию модулей системы мы делаем не во время стадии интеграции, а постоянно.



**Как это работает:**

У нас есть определенная система, которая работает при *триггерах* (определенных событиях).

Есть команда разрабов (D1, D2). VCS (Version Control Server) – например репозиторий на гите.

D1 делает фитчу и коммитает ее в гит и тут происходит *магия(да-да….)* и запускаются наши юнит-тесты, которые есть в коде. Если эти тесты падают, то D1 приходит сообщение мол «все было ок, но вот тут ты сделал какую-то фитчу и тесты начали падать» (это называется *фитбек*). Т.е. девелопер получает этот фитбек спустя 5-7 минут с момента коммита.

**Че в итоге делает D1:**

Он вносит какие-то вправки в код и делает еще один коммит уже с фиксом. Коммит опять начинает проходить юнит-тесты и если они ок, то след тест (тест веб-сервиса), если и там все окей, то запуск след тестов (юзер-интерфейса). В случае успешно пройденных тестов, значит все супер.

Для D2 аналогично.

Если например юнит-тесты прошли, а упали тесты сервиса, то после нового коммита и фикса, код заново проходит юнит-тесты (т.е. всю цепочку).

**В общем, как только мы сделаем коммит, сразу идет интеграция всей системы, и при возникновении ошибки мы сразу об этом узнаем.**

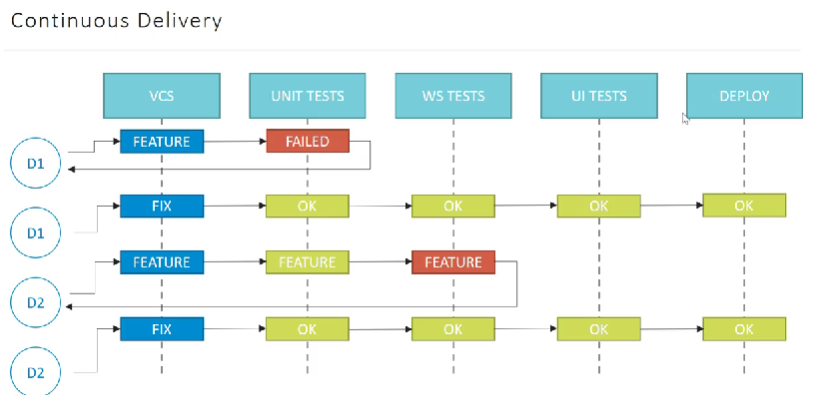
**Требования, чтобы сделать CI:**

1. **Version Control System (Система версионного контроля)**. Код должен где-то храниться.
2. **Build Automation**. Сборка должна быть заавтоматизирована. Ее должна делать какая-то машина, скрипт, но не человек. (ЛЮДИ ЛЕНИВЫЕ И ДОРОГИЕ, А ЕЩЕ НЕНАДЕЖНЫЕ).
3. **Test** **Automation.** Заавтоматизированные тесты**.** Мы не будем ждать, пока какой-нибудь чел пройдет эти тесты, а вдруг у него….
4. **Often commit.** Частые коммиты помогут понять, где была проблема, нежели мы одним коммитом запушим огромный кусок кода и потом не разберемся где напортачили…
5. **Build on change.** Сборка по изменению. Должны стоять триггеры на изменение кода и запуск всей системы.

**Задача CI:** Как можно раньше узнать о дефектах. В конце концов иметь собранную протестированную систему с новыми изменениями.

**Continuous Delivery** – продолжение CI

Когда в самом конце идет доставка сделанной системы пользователю.



**Benefits (выгоды) при CD:**

1. **Defect early discovery**. Раннее обнаруживание дефектов.
2. **Automatic application deploy**. Имеем возможность автоматически установить приложение уже с изменениями.
3. **Transparent health monitor**. Прозрачный монитор здоровья проекта. Т.е. если мы видим в течении дня, что задачи красные, то нужно принимать какое-то решение.
4. **Test pipeline customization**. Есть возможность визуализировать **pipeline** (некая цепочка сборок)
5. **Built-in parallel execution**. Возможность распараллеливания задач.

**Why Jenkins**

* Бесплатный, опен-сорсный (большое количество последователей)
* Кросс-платформенный – на любой платформе (Линукс, винда)
* Огромное количество плагинов.
* 62%

Аналоги:

Travis CI (21%), Gitlab CI, TeamCity, CircltCI, Microsoft TFC / VSTS

**Build triggers**

**Build triggers**

**Триггеры** – события, которые вызывают определенную задачу. Т.е. – автоматизируют джобы.

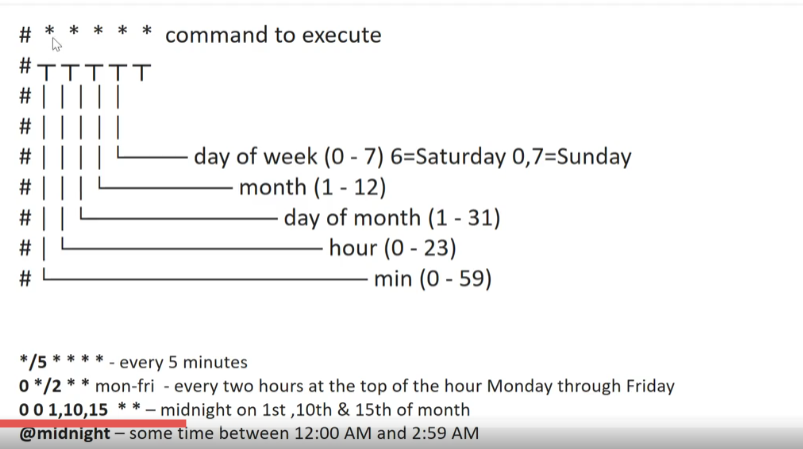
**Триггеры бывают:**

1. **On-demand run.** Запуск по клику.
2. **Other job finish.** Вызов одной задачи после завершения другой.
3. **Cron schedule.** Вызов по расписанию.
4. **SCM check.** Просмотр наличия коммитов.

**Cron syntax**

Если вдруг не понял синтаксис, на тебе - https://crontab.guru

Задается 5 параметров: MINUTE HOUR DOM MONTH DOW



\* - любое

Если **\* \* \* \* \*** - сборка будет выполняться каждую минуту, час, день, месяц, и каждый день недели (1 раз в минуту)

**\*/5 \* \* \* \*** - каждую мин, которая делится на 5, остальное любое

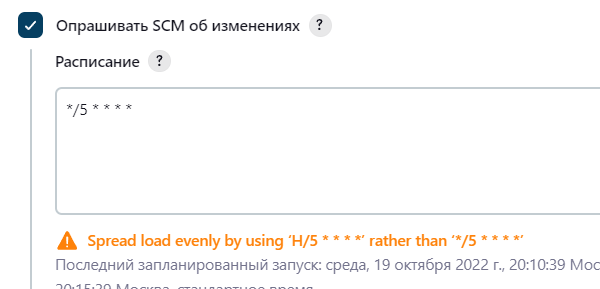
**0 \*/2 \* \*mon-fri** - минута 0, значит каждый час, но час делится на два, значит каждый четный час по будням.

**0 0 1,10,15 \* \*** - раз в сутки, в полночь, день месяца 1, 10 и 15 число, каждый месяц, каждый день недели. Эта джоба запуститься 3 раза в месяц (1, 10 и 15 числа)

**@midnight** – позволяет запускать в случайное время между 00:00 и 02:59

Jenkins предлагает нам заменить \* на Н. У каждой джобы есть свои уник параметры и можно посчитать хеш (должен быть уникальным для каждой джобы)

Крч если джоб много и мы напишем Н, то они будут запускаться с интервалом в 5 минут, но в разные минуты (1ая джоба в 10:01, 2ая в 10:02 и тд. И получится так, что у каждой джобы с таким триггером будет разное время выполнения.



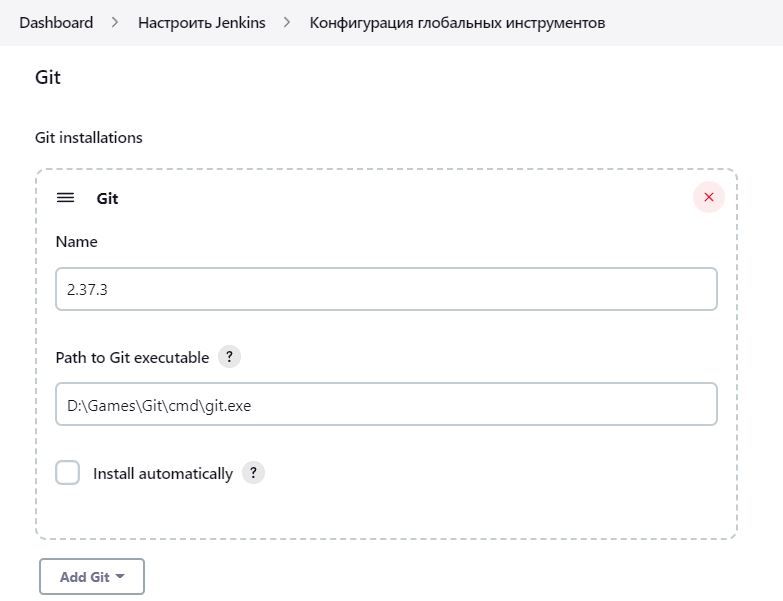
**Jenkins integration with other tools**

**Git integration**

**Как интегрировать с гитом и выкачивать проекты с гита из дженкинса.**

**Че надо:**

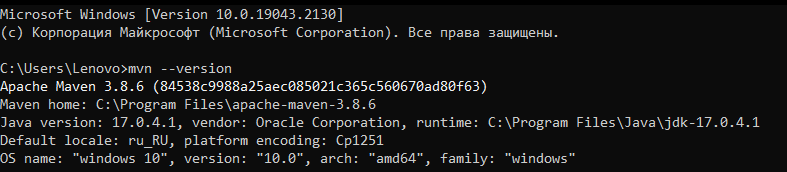
1. Плагин для работы с гитом
2. Объяснить дженкинсу где находится гит.



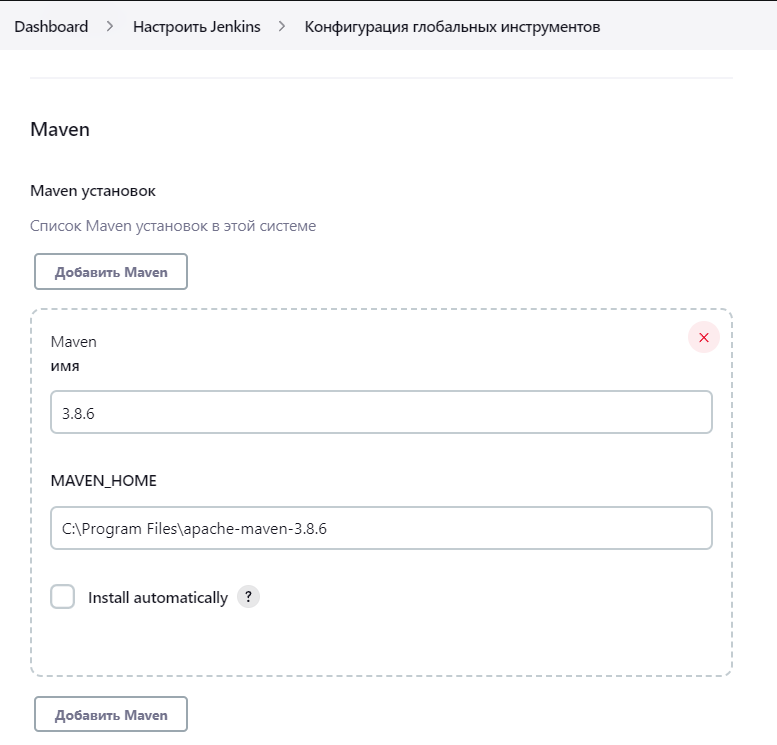
**Maven integration**

Maven - Фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM

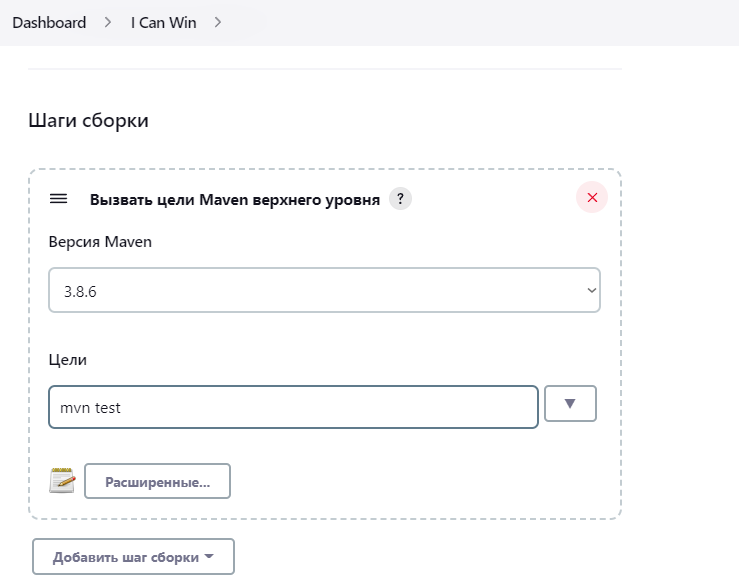
Узнаем где мавен



Так же объясняем дженкинсу где он:



Что бы его использовать, добавляем в качестве билд степа (триггера сборки):

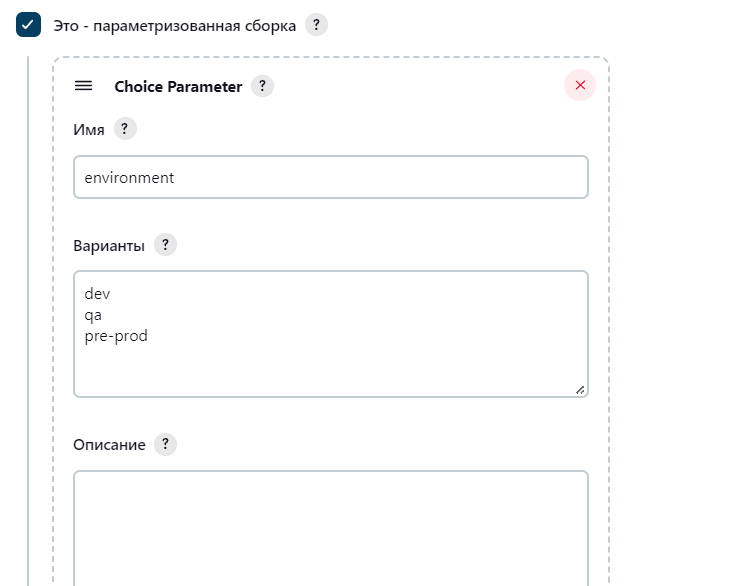


В контексте **Дженкинса** **артефакты**-это конечный продукт или результат, созданный при завершении любого задания **Дженкинса** и сохраненный для дальнейшего использования.

**Parametrized jobs**

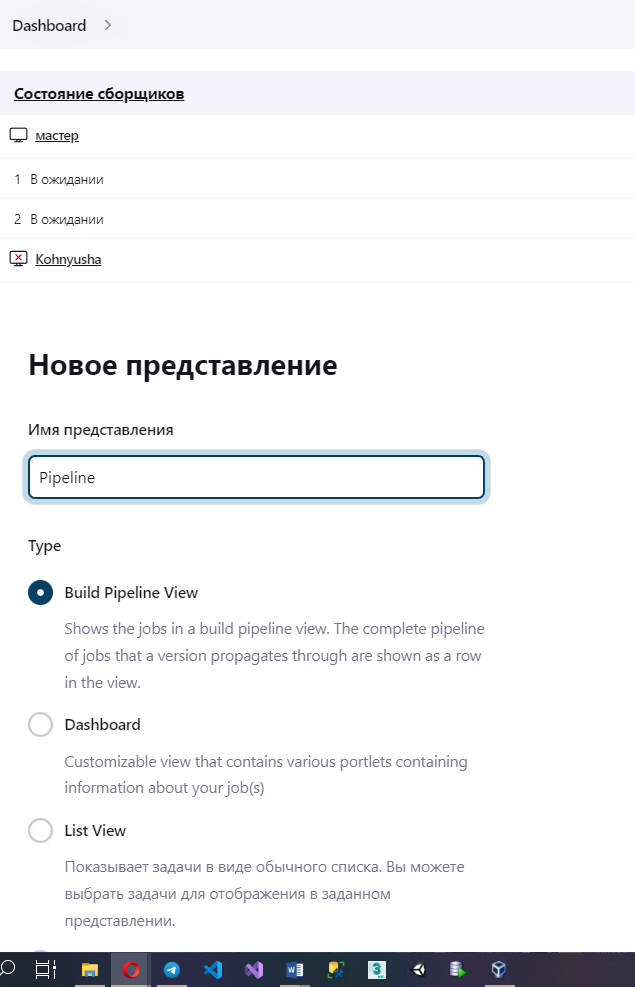
Параметр является переменной средой, который приходит в джобу.

Если джоба будет стартовать по таймеру, то в качестве параметра будет передаваться **dev(по умолчанию 1ый)**

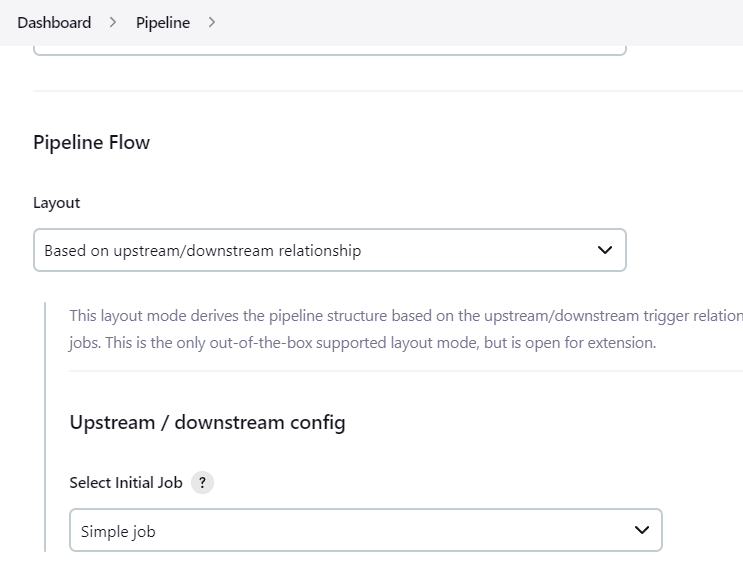


**Pipeline – цепочка**

Крч можем делать свою цепочку задач, которые представляют из себя CI

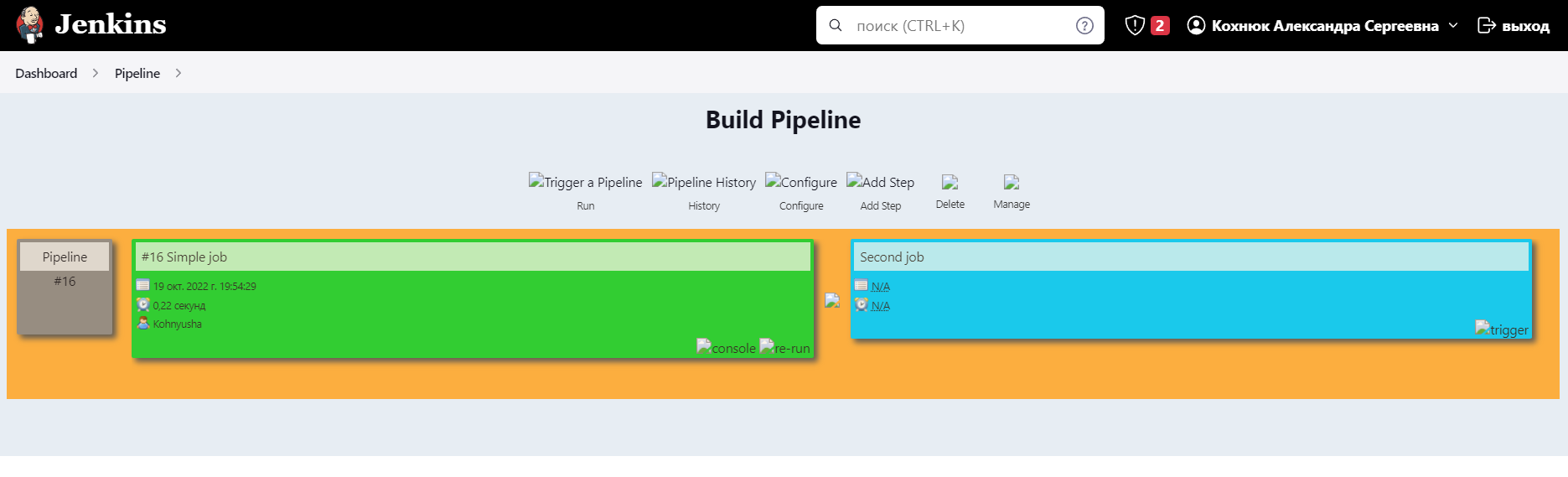
****

Стартовая точка (джоба):

****

Зеленый – прошла

Желтый – идет сейчас

****

**Nodes**

**Node** – это машина, которую можно подключить к серверу, и на ей выполнять реальную работу.

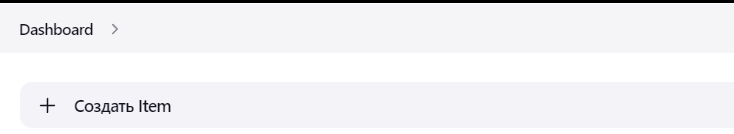
Можно распараллеливать задачи.

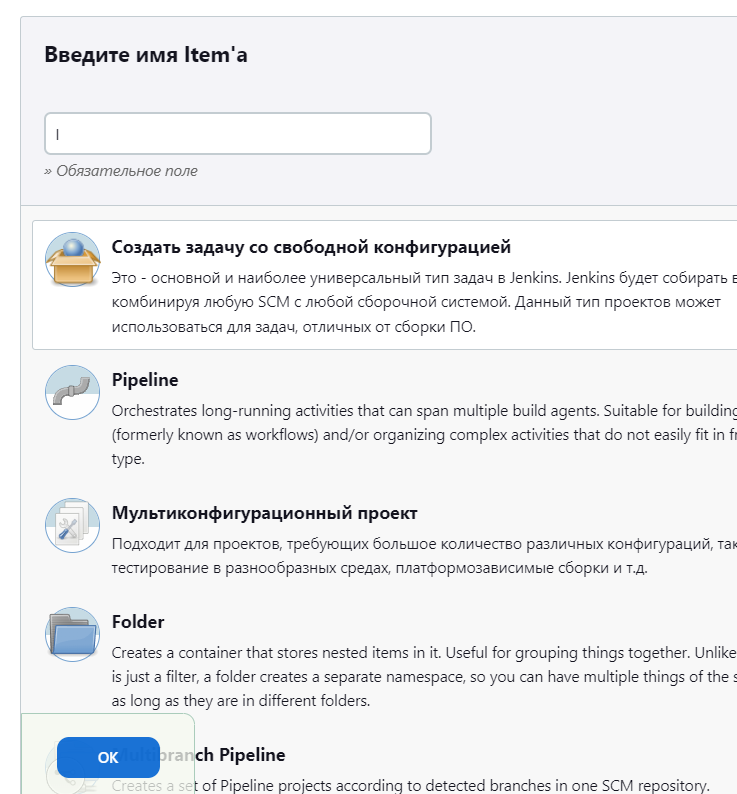
**I can win**

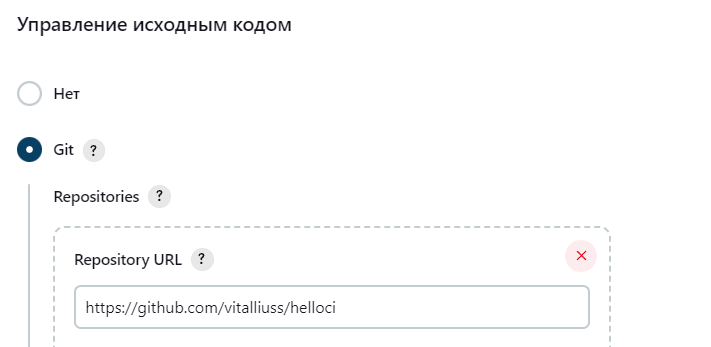
1. Установите Jenkins

2. Создайте задачу, которая будет делать следующее:

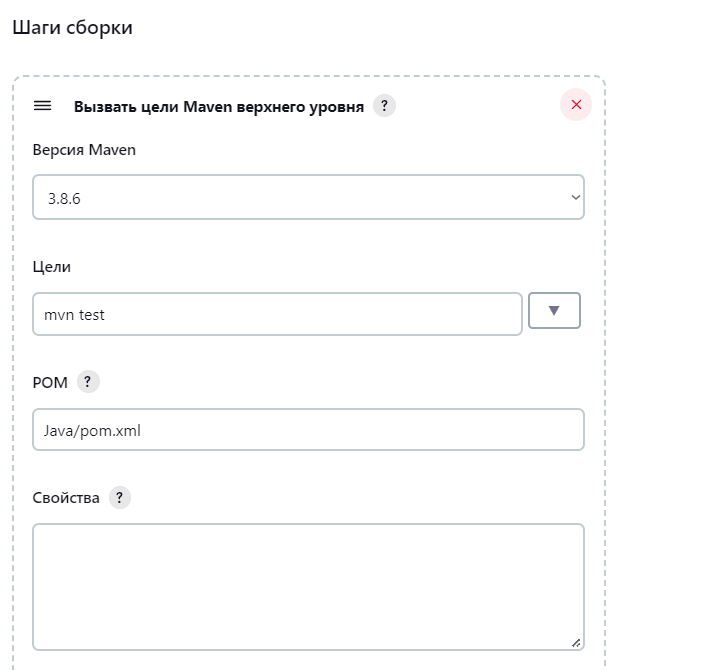
* Клонировать проект <https://github.com/vitalliuss/helloci>



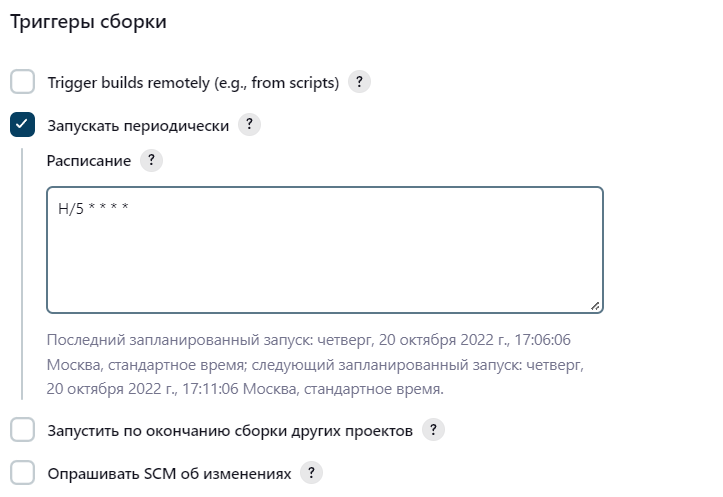




* Запускать тесты из проекта в директори Java с помощью цели mvn test



1. Настройте билд триггеры таким образом, чтобы задача выполнялась раз в 5 минут



**Bring It On**

**Первые 2 задания по аналогии с I Can Win**

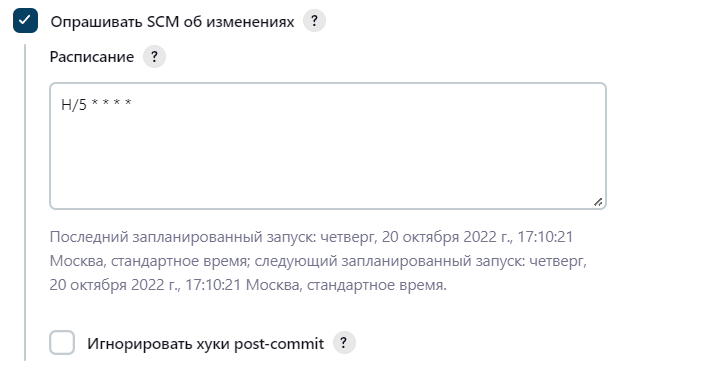
1. Установите Jenkins

2. Создайте задачу которая будет делать следующее:

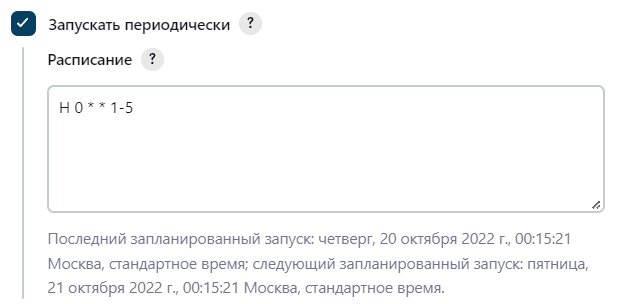
* Клонировать проект <https://github.com/vitalliuss/helloci>
* Запускать тесты из проекта в директори Java с помощью цели mvn test

3. Настройте билд триггеры:

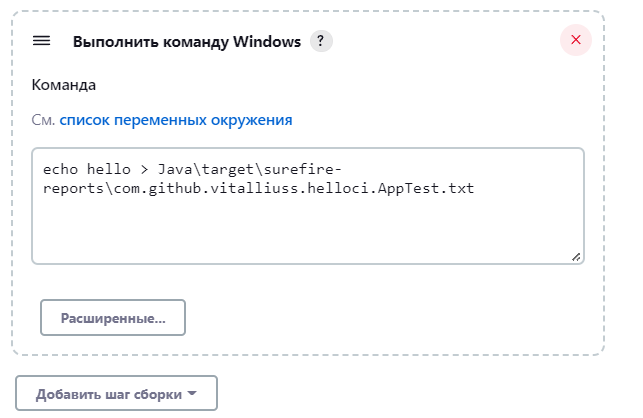
* Запуск тестов не позднее чем через 5 минут после коммита в git



* Каждый будний день в полночь



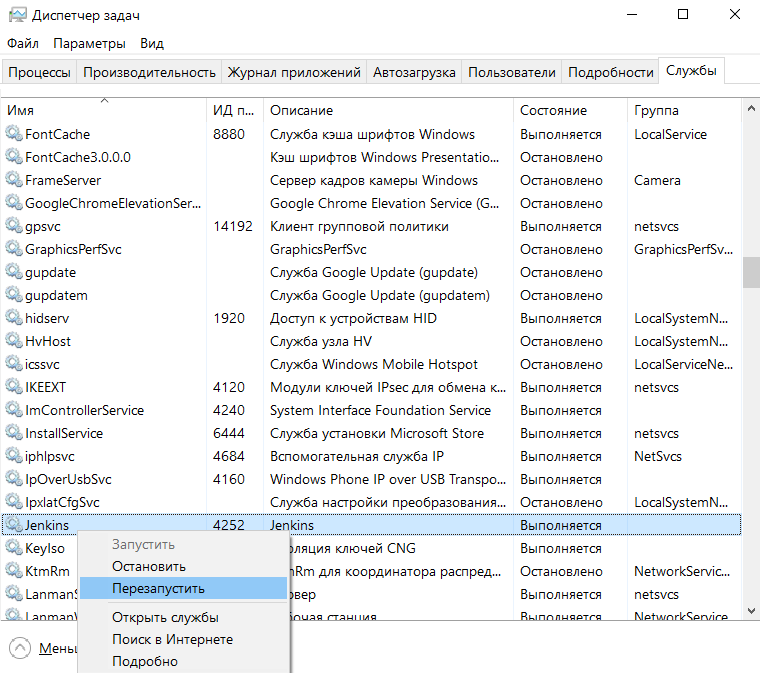
4. Опубликуйте файл “Java\target\surefire-reports\com.github.vitalliuss.helloci.AppTest.txt” как артефакт



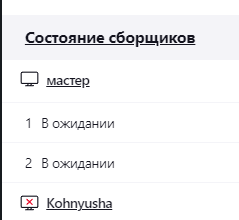
**Hurt Me Plenty**

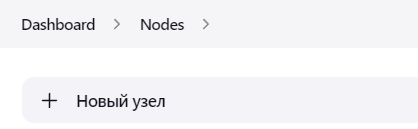
Данное задание выполняется на основе Bring It On задачи.

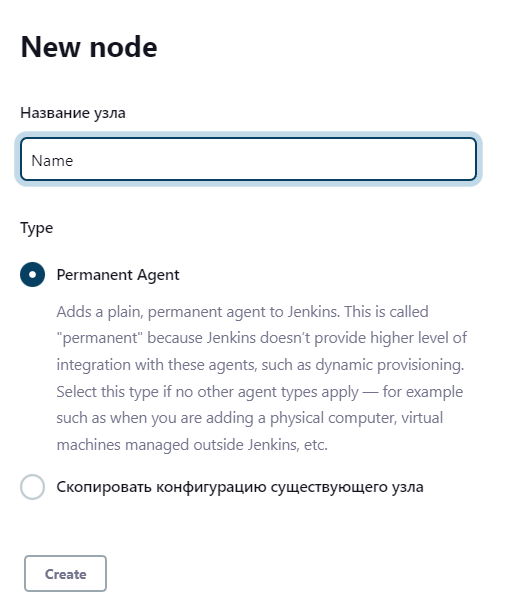
* 1. Сменить порт сервера на 8081
* Открываем файл jenkins.xml
* Меняем порт на 8081
* Перезапускаем дженкинс:



* 1. Создать ноду и настроить сервер так, чтобы джоба выполнялась только на slave ноде
* Кликнуть на «состояние сборщиков»

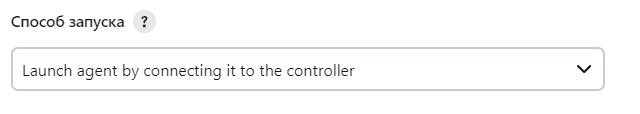




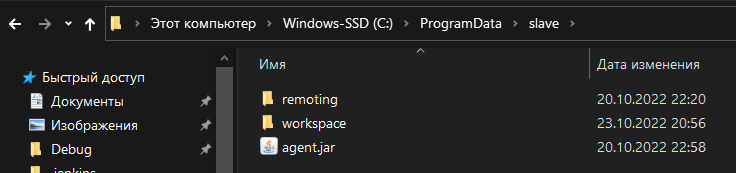


* В настройках указываем:

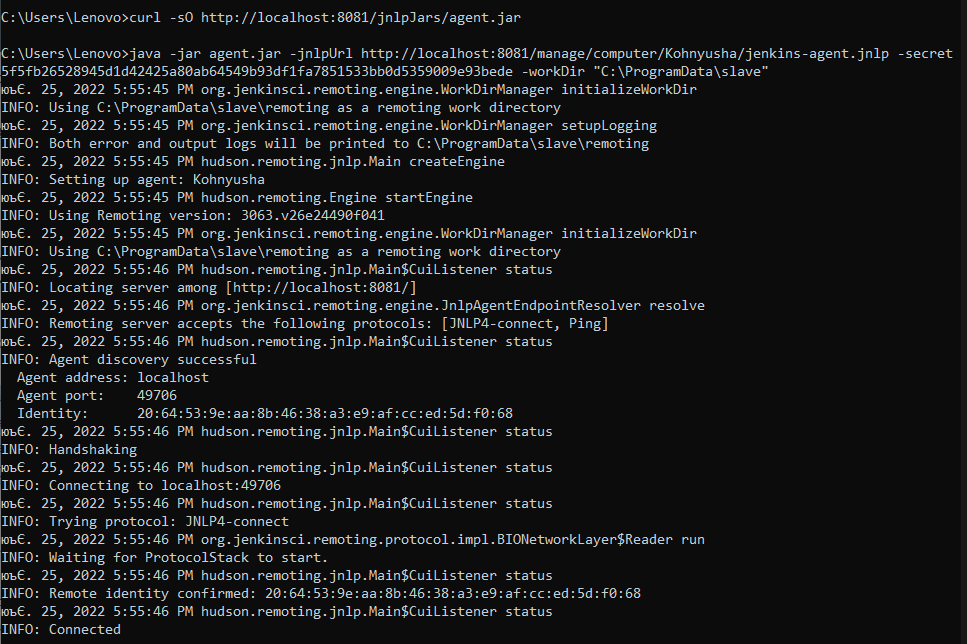




В папку slave загружаем файл агента.



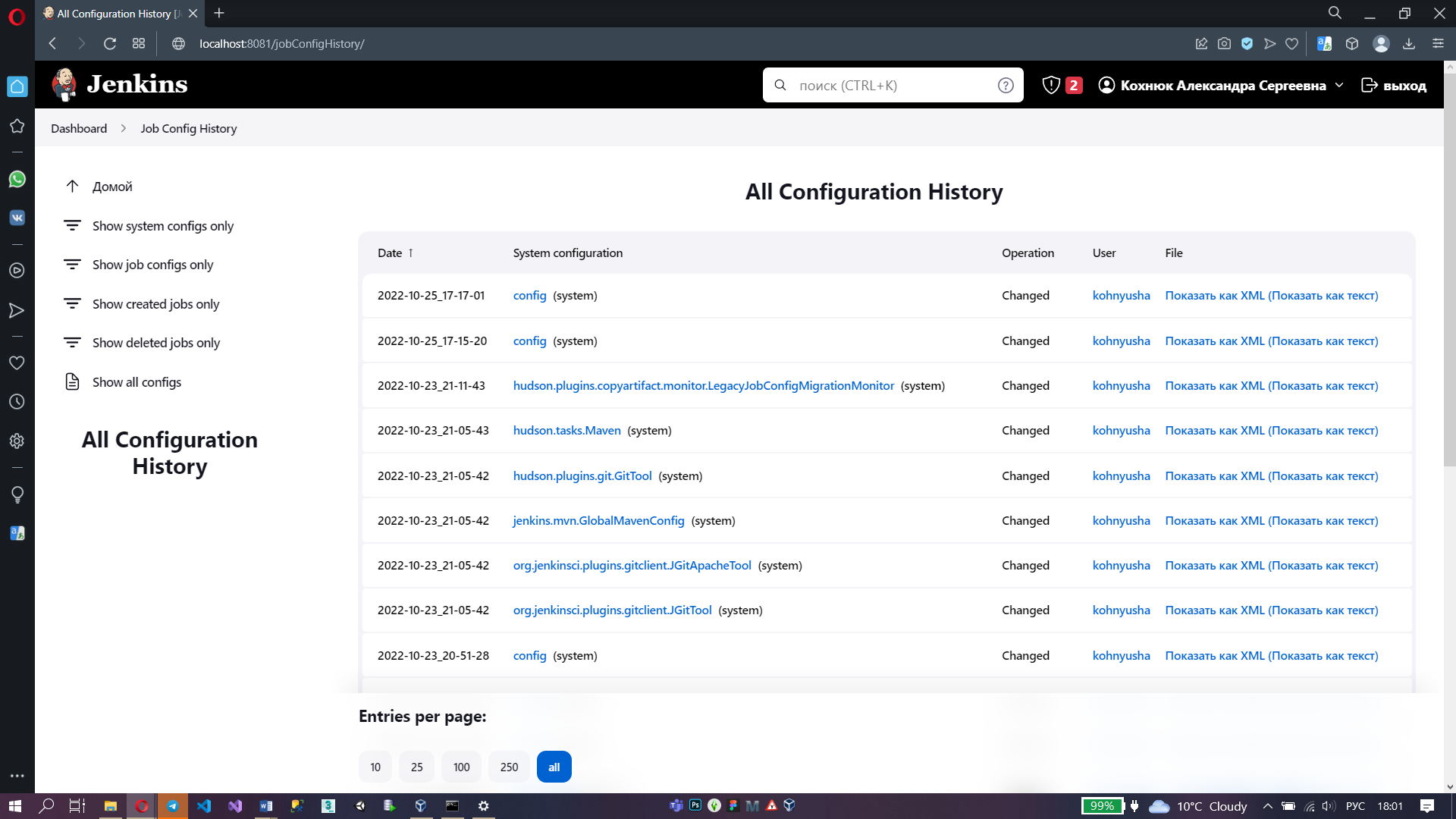
Чтобы запустить ноду:



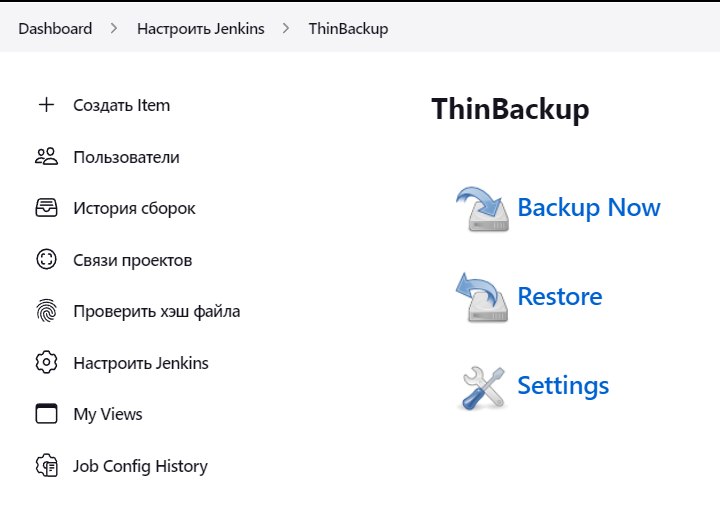
* 1. Настроить Job Config History и thinBackup

Устанавливаем плагинs «Job Config History» и «ThinBackup»

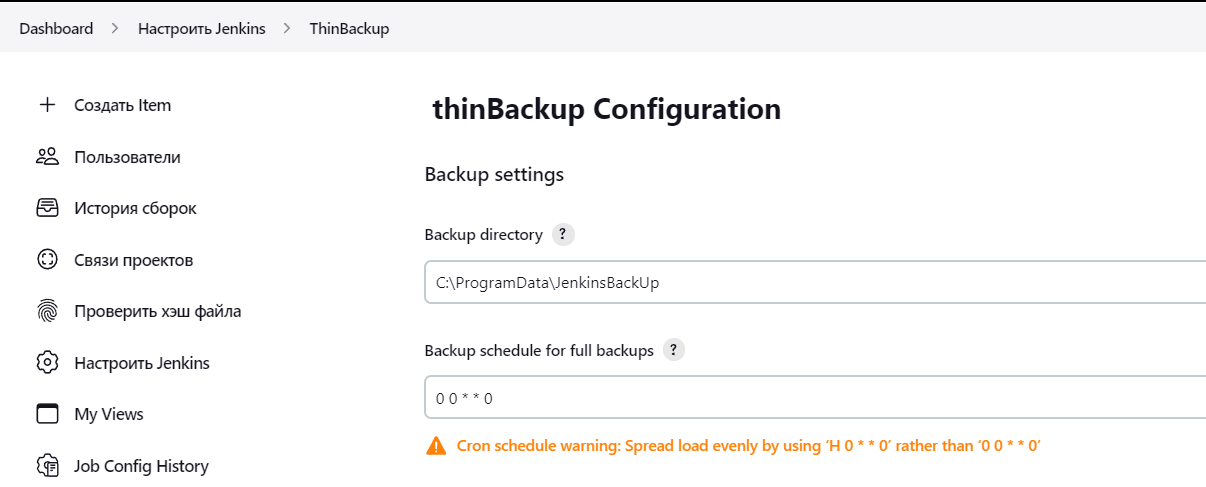
**Job Config History** – для записи истории конфигураций задачи



**ThinBackup -** для управления резервным копированием.



Далее указываем директорию для копий и расписание выполнения (в 00:00 по воскресеньям каждого месяца)



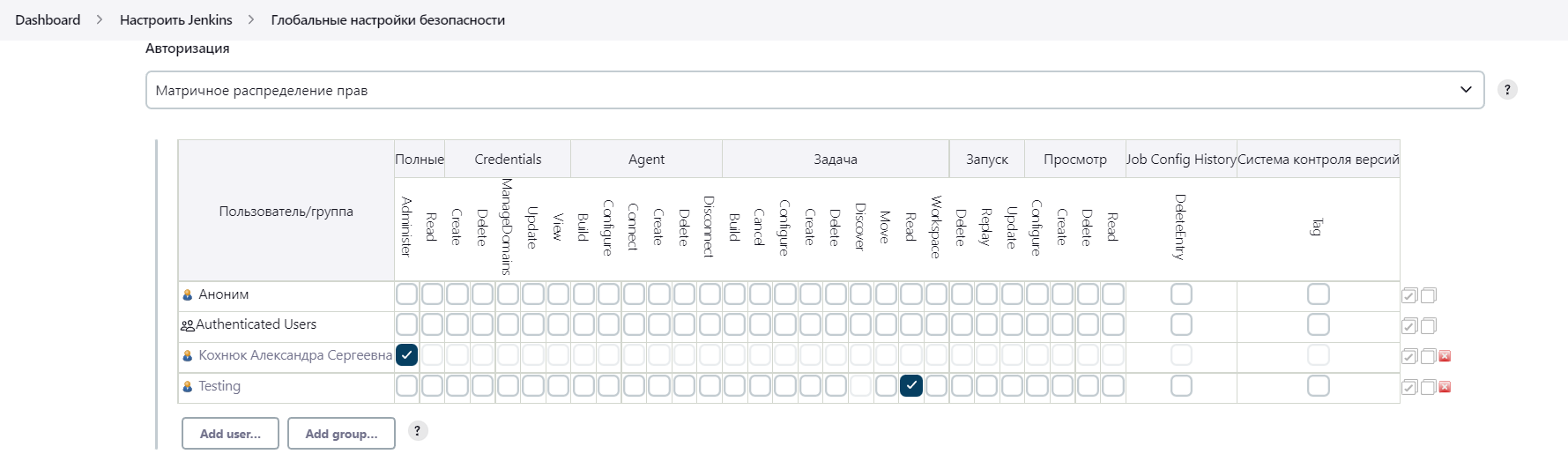
**Hardcore**

Данное задание выполняется на основе Hurt Me Plenty

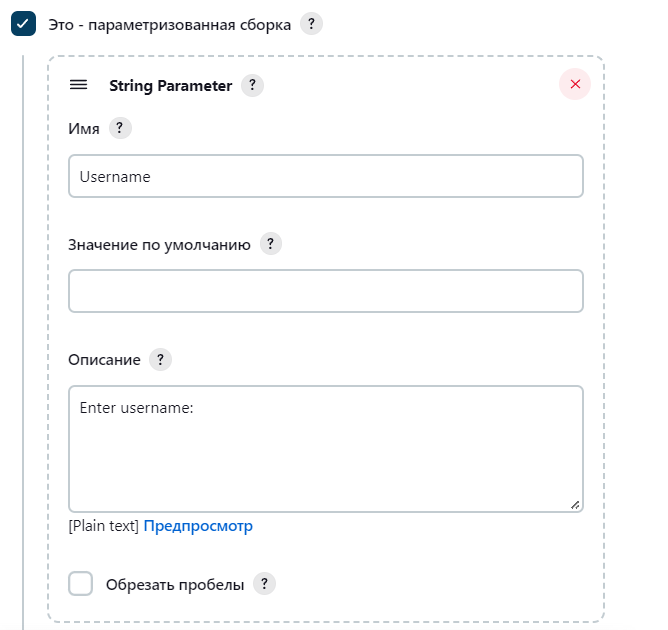
1.Создать пользователя user и дать ему права на просмотр джоб Jenkins, но без возможности записи или смены настроек

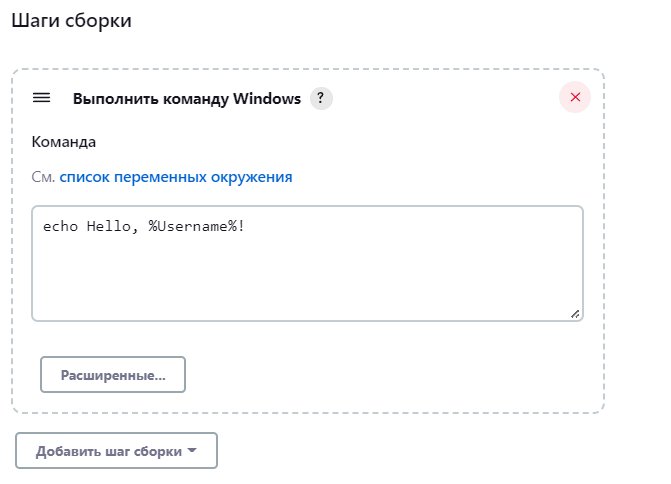
* User testing - login
* Usertesting1 - password

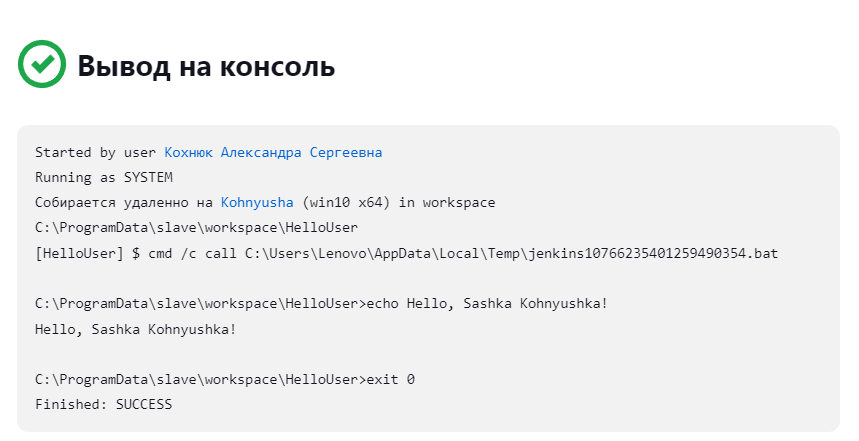
Устанавливаем плагин «Matrix Authorization Plugin».



2. Создать параметризованную джобу HelloUser, которая будет спрашивать в качестве параметра имя пользователя (username) и писать в консоль "Hello, username!".

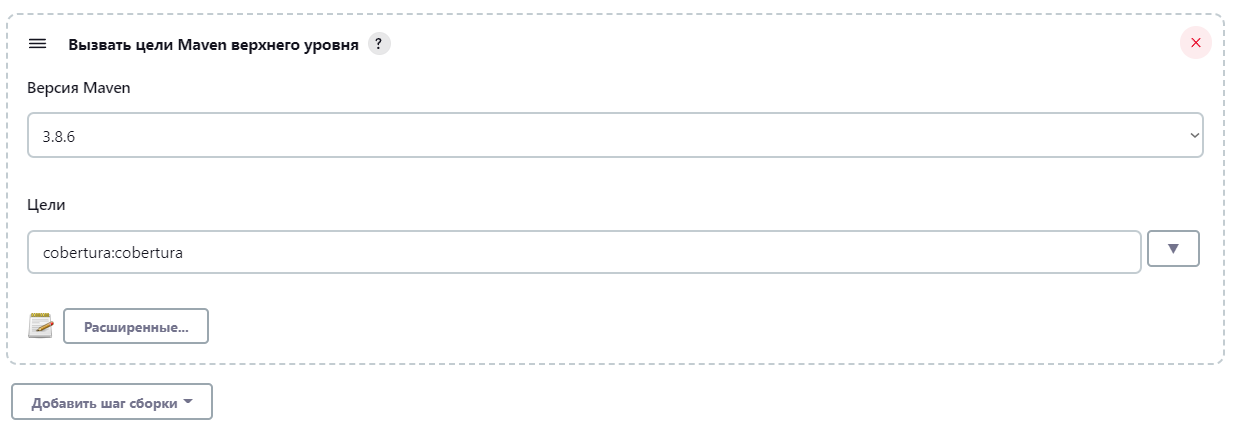


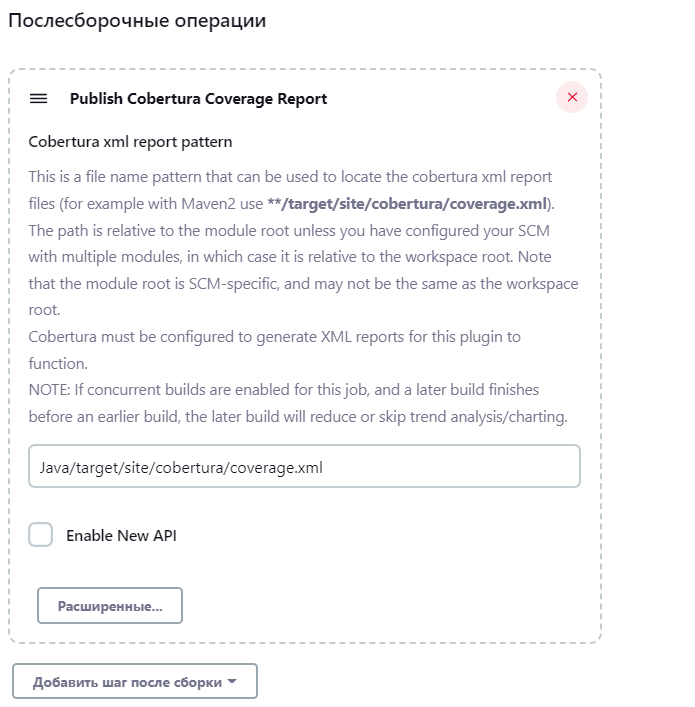




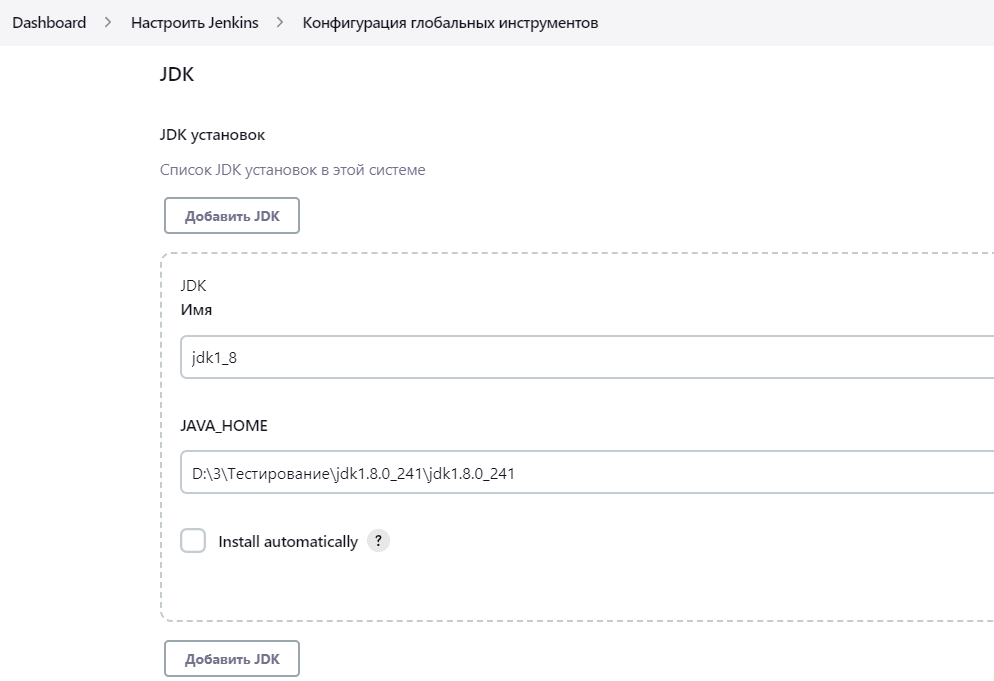
3. С помощью цели  mvn cobertura:cobertura измерьте покрытие кода юнит-тестами (code coverage) и опубликуйте на странице джобы в виде графика

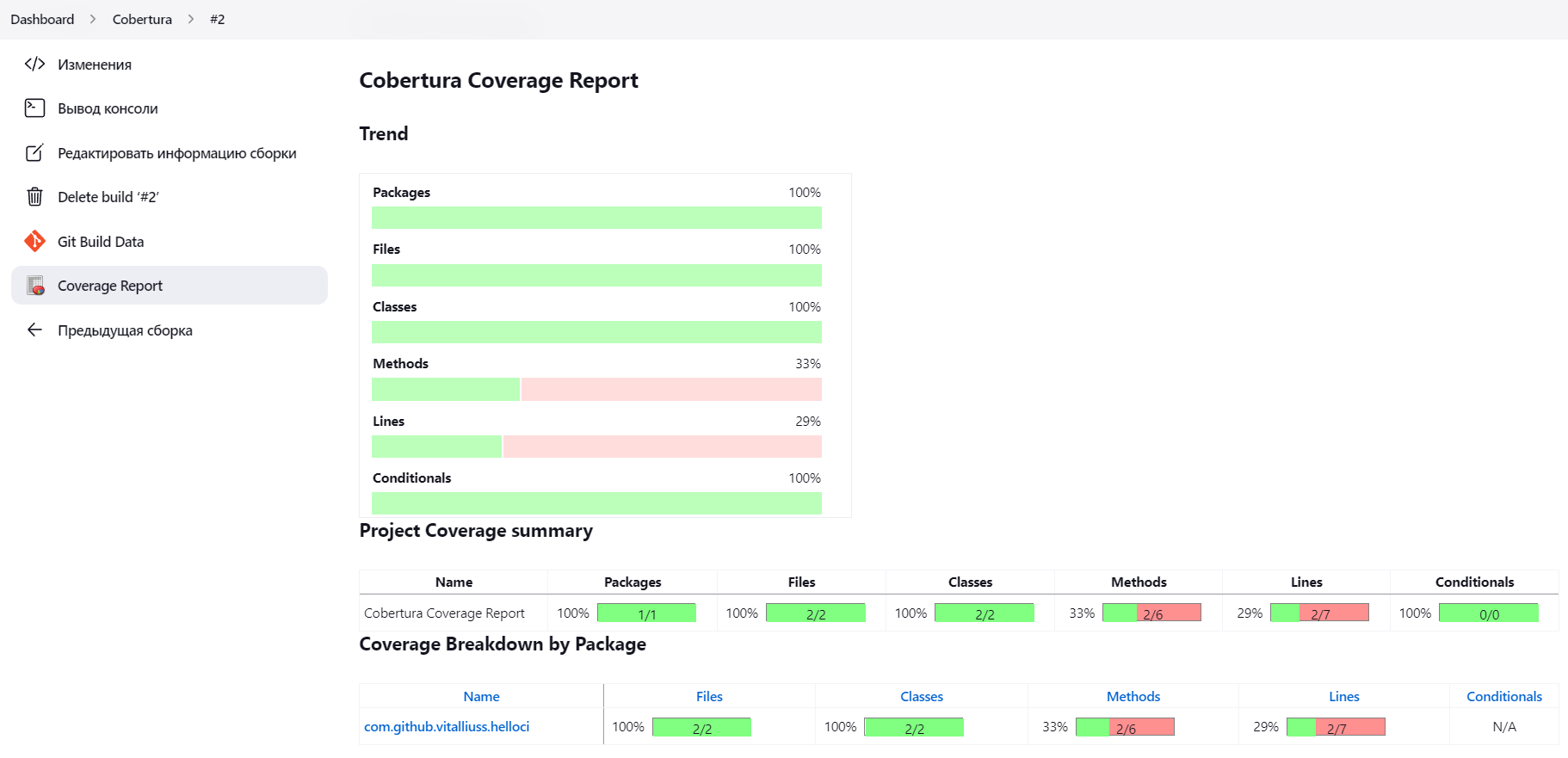
* Устанавливаем плагин Cobertura
* Создаем новую джобу “Cobertura”
* Настраиваем:





В конфигурации глобальных настроек дженкинса:





**Cobertura** – это бесплатный инструмент Java, который вычисляет процент кода, доступного тестам. Его можно использовать для определения того, в каких частях вашей Java-программы отсутствует тестовое покрытие.