GeekBrains

Тестирование Backend на Java

Настройка CI в Jenkins и GitLab CI. Документирование API

На этом уроке

1. Поговорим о CI для тестов.
2. Разберём, что важно для конфигурации CI.
3. Рассмотрим виды документации API.

Оглавление

Введение

- Знакомство с Jenkins
- Настройка Jenkins
  - Установка плагинов

Создание задачи на сборку тестов

- Мозговой штурм для конфигурации задачи на сборку
- Конфигурация задачи

Практическое задание

Дополнительные материалы

Используемые источники

Введение

Сегодня мы подведём итоги нашей автоматизации и разберёмся, как развернуть CI в популярном бесплатном инструменте — Jenkins. Затем поговорим о документации для API:

- какая бывает;
- что надо требовать.

Знакомство с Jenkins

Jenkins — сервер непрерывной интеграции, написанный на Java. Это чрезвычайно расширяемая система из-за внушительной экосистемы разнообразных плагинов. Настройка пайплайна осуществляется в декларативном или императивном стиле на языке Groovy, а сам файл конфигурации (Jenkinsfile) располагается в системе контроля версий вместе с исходным кодом.

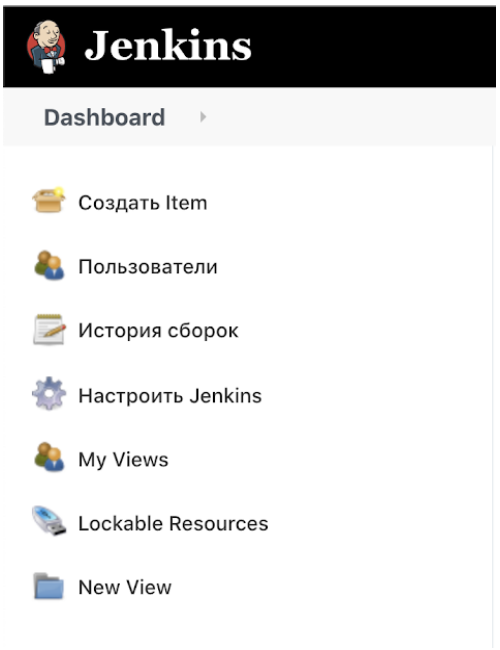
Jenkins устанавливается локально или разворачивается в облаке. Подробнее — на странице скачивания сервиса. Большинство компаний разворачивает Jenkins у себя на сервере, покупая enterprise-версию, которая накладывает некоторые ограничения на используемые ЯП и сервисы.

Например, нельзя автоматически установить Java JDK выше 9 версии. Надо ограничиться ей или самостоятельно установить на сервер JDK подходящей версии и указать Jenkins, где он лежит.

С этого и начнём.

Настройка Jenkins

После авторизации появляется дашборд — главная страница сервиса. Панель инструментов слева содержит требуемый набор опций для настройки. Переходим в «Настроить Jenkins».



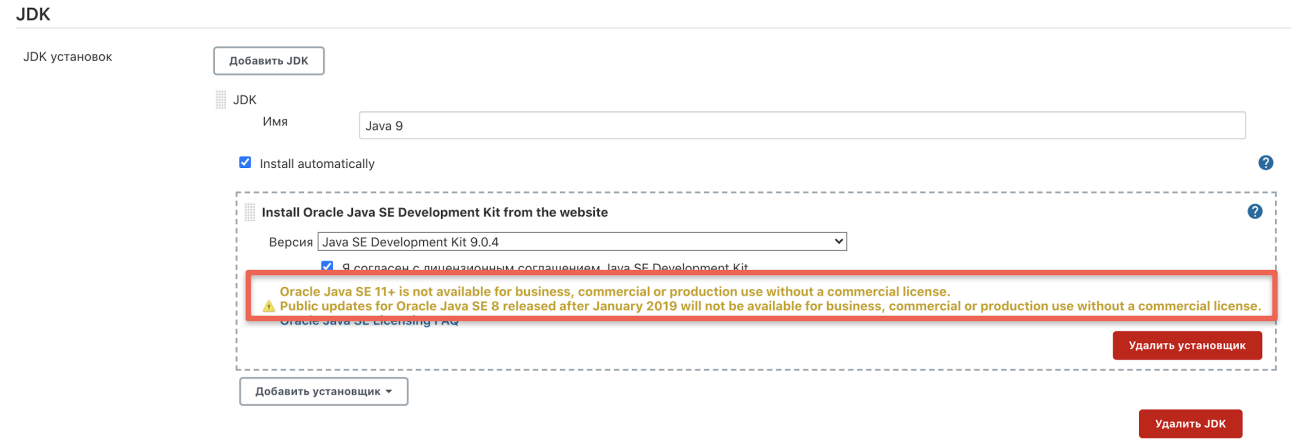
Появляется экран с возможностями для настройки. Сегодня мы поработаем с двумя из них — «Конфигурация глобальных инструментов» и «Управление плагинами». Переходим в «Конфигурацию глобальных инструментов». Здесь настраиваются:

- maven и другие сборщики, такие как Gradle или Ant;
- Git;
- JDK;
- Allure;
- другие сервисы.

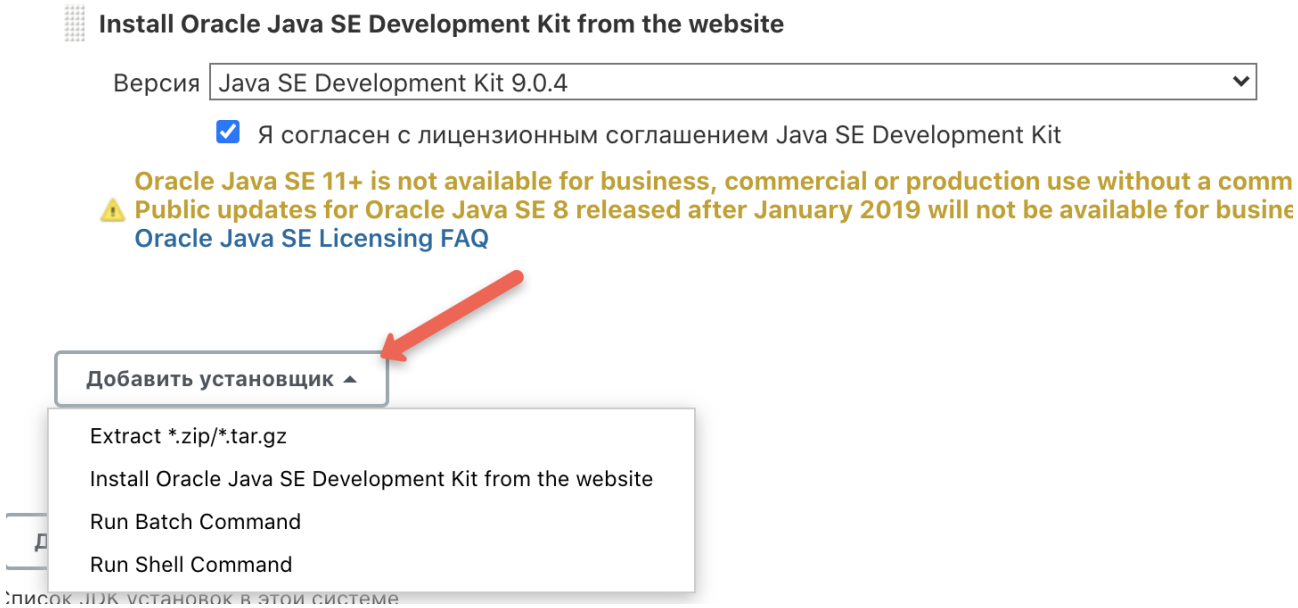
Если что-то из перечисленного нет, обращаемся к разделу «Управление плагинами».

Нажимаем на кнопку «Установка JDK», затем — «Добавить JDK». Далее указываем подходящую версию.

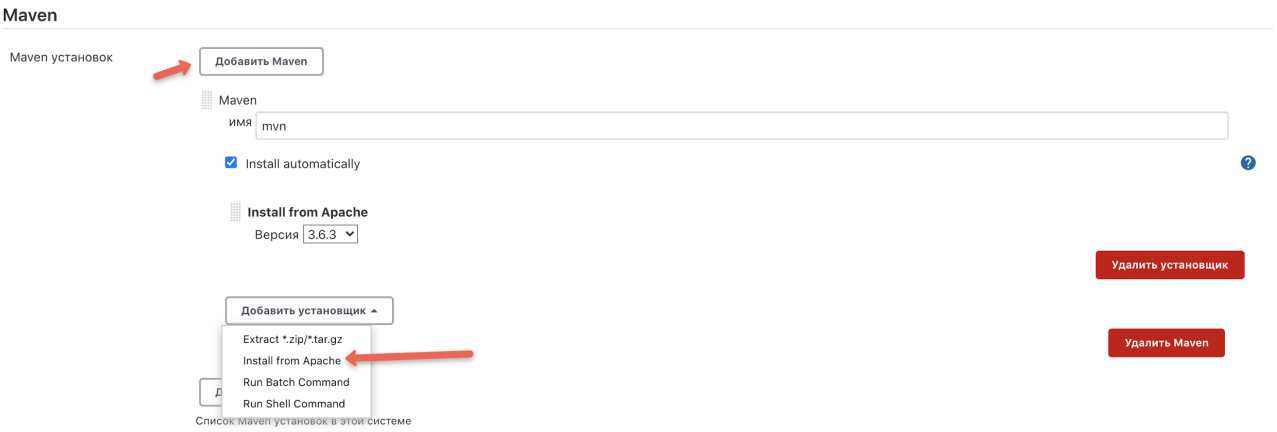
Важно обратить внимание на ограничения enterprise-версии. Для установки старой версии надо указать логин и пароль от личного кабинета:



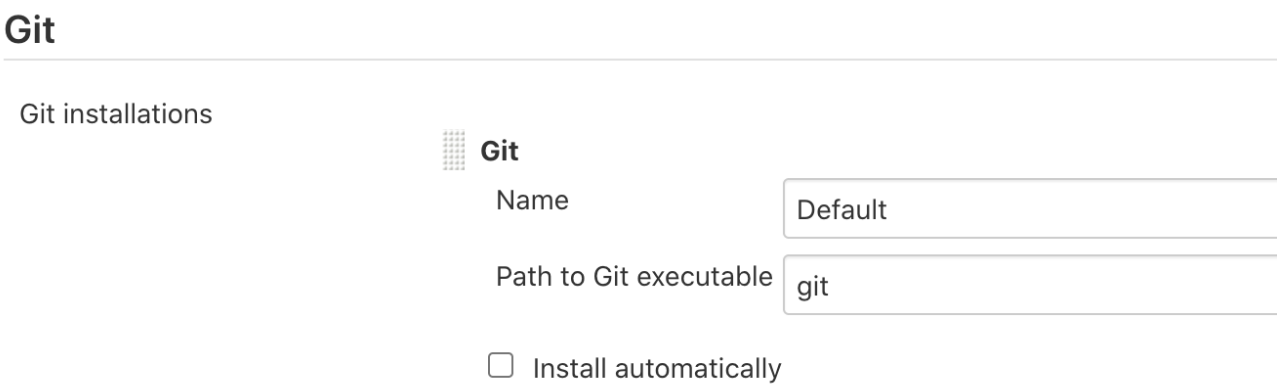
Если потребуется другая версия Java, используем другие опции установщика:



Сразу устанавливаем maven. Рекомендуется выбрать опцию установщика «Установить из Apache» (Install from Apache):



Проверяем установку git:



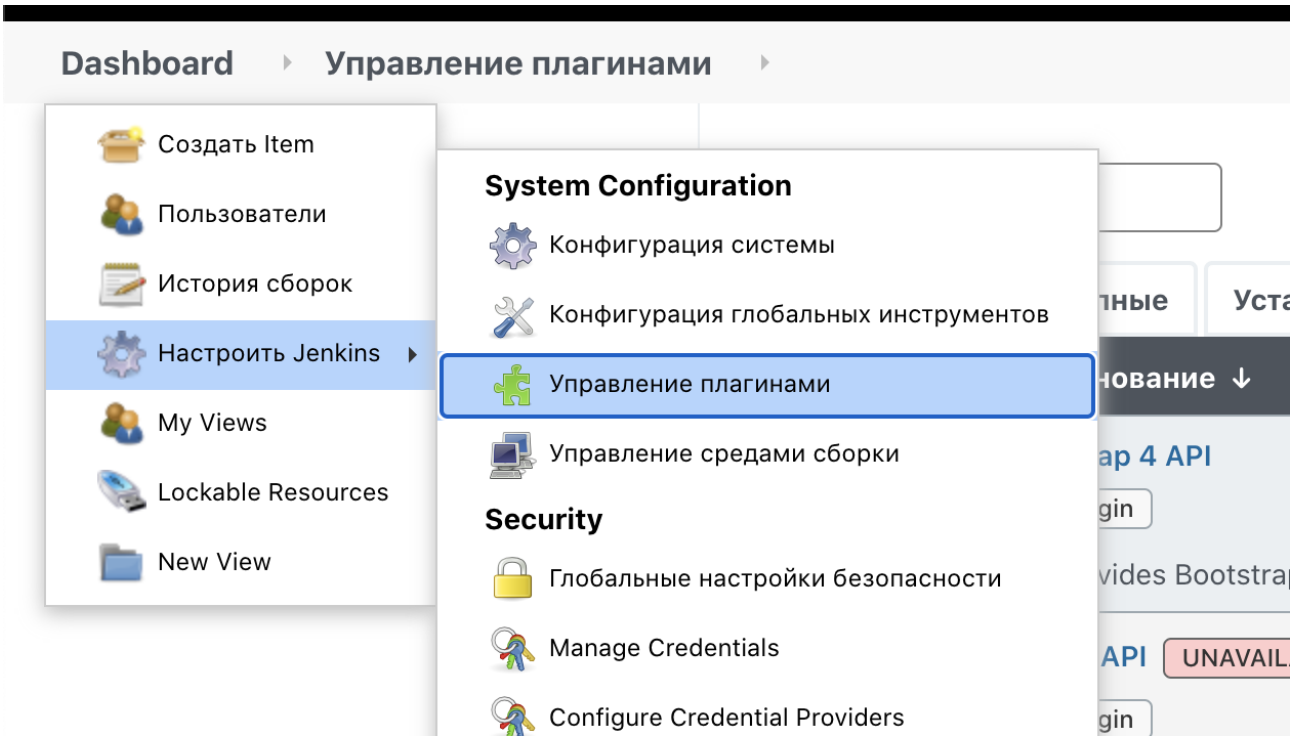
Эти значения выставляются по умолчанию, и их можно оставить.

Далее нажимаем на кнопку Apply для подтверждения изменений и Save, чтобы закрыть страницу настройки. Переходим к плагинам.

Установка плагинов

Jenkins — это open-source ПО. Он устроен так, что практически бесконечно расширяется посредством плагинов. Для поиска плагинов есть отдельный сайт, но можно написать и свой плагин. Пока воспользуемся действующим.

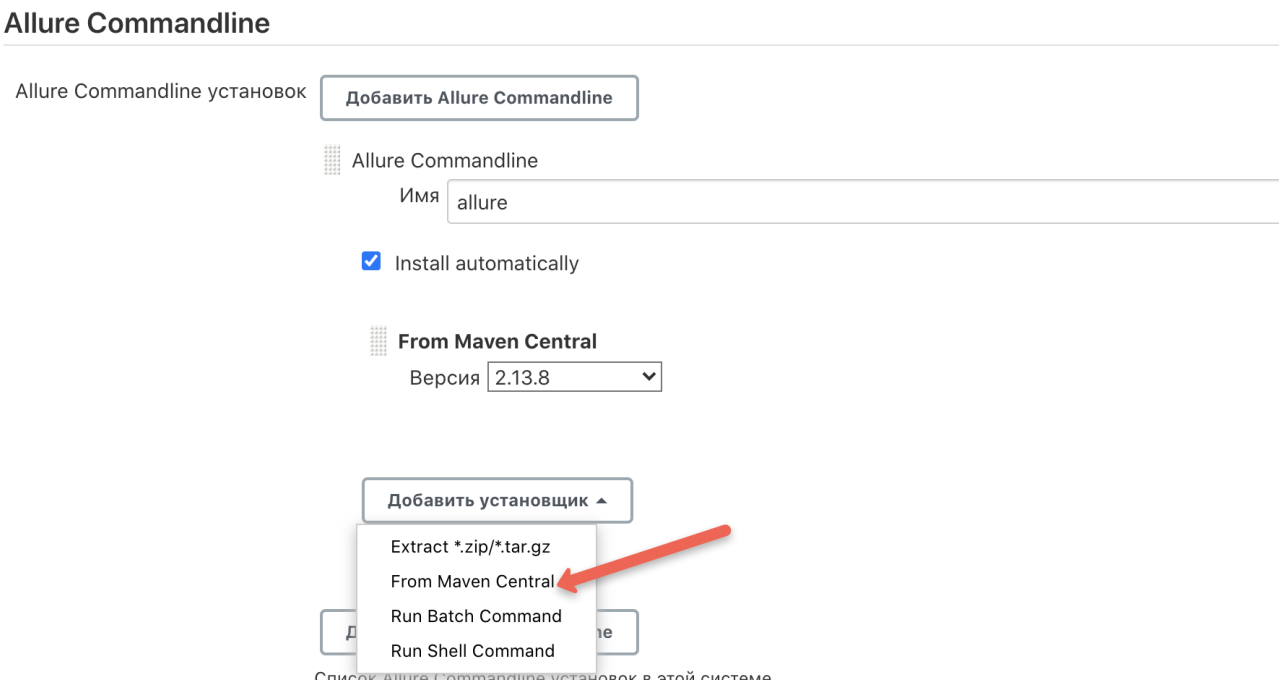
1. Переходим в Dashboard.
2. «Настроить Jenkins».
3. «Управление плагинами».



Чтобы искать и устанавливать плагины, надо перейти во вкладку «Доступные». Нам потребуются следующие плагины:

1. Git Parameter Plug-In.
2. Maven Integration plugin.
3. GitHub Branch Source.
4. Allure Jenkins Plugin.

После установки плагинов надо ещё раз вернуться в настройку глобальных инструментов и выбрать инструмент установки allure. Самый удобный вариант это сделать — настроить автоматическое скачивание с Maven repository:



Теперь создадим джобу, которая будет запускать наши тесты и формировать отчёт.

Создание задачи на сборку тестов

Для успешного создания задачи на сборку (джобы, item) надо иметь чёткое представление, откуда Jenkins будет брать код наших тестов. Самые распространённые варианты:

1. Ссылка на репозиторий с указанием ветки.
2. Указание названия docker-контейнера в docker-реестре. Для этого сначала надо упаковать тесты в контейнер, например, поместив туда executable jar-пакет.

Разберём первый вариант, а как использовать docker, рассказано здесь.

Слева на панели меню нажимаем на кнопку «Создать item». В появившемся окне есть несколько опций для создания проекта, разберём некоторые из них:

1. Создать задачу со свободной конфигурацией. Это самый распространённый и универсальный тип задач в Jenkins. Если мы не можем установить плагины, например, по требованиям безопасности, то используем эту опцию.
2. Создать проект maven. Идеально подходит, если ресурс тестов для Jenkins — неупакованный код, например, архив или ссылка на репозиторий.
3. Создать мультиконфигурационный проект. Если у нас сложный проект, и надо предусмотреть различные конфигурации, используем этот вариант. А можно создать несколько простых, применив первую опцию: для смоука, регрессионного тестирования и так далее.

4. Создать папку. Это не задача на запуск, а организация нескольких задач в одном месте, как «папка» в проводнике.

Так как мы использовали maven. Самый простой способ настроить сборку автотестов — задача «Создать проект maven». Введём название для задачи и подтвердим создание.

Мозговой штурм для конфигурации задачи на сборку

Перед тем как конфигурировать задачу на сборку, подумайте и/или спросите коллег, что важно для этой задачи. Опишите параметры, которые надо передавать на вход. Что считается источником параметров?

Для примера разберите простую конфигурацию:

1. Код тестов берётся из указанной ветки github-репозитория.
2. Тесты прогоняются стандартным плагином mvn surefire в цели test.
3. После прогона тестов вне зависимости от результата формируется Allure-отчёт.

По каждому пункту задайте себе следующие вопросы:

1. Код тестов берётся из указанной ветки github-репозитория.
    a. Как осуществляется доступ?
    b. Если репозиторий приватный, есть ли авторизация?
    c. По логину или паролю либо SSH?
    d. Все ли ветки должны быть доступны для прогона?
    e. Есть ли триггеры, по которым запускается сборка, например, открытие или мерж pull request?
    f. Требуется ли ссылка на репозиторий?
2. Тесты прогоняются стандартным плагином mvn surefire в цели test.
    a. Достаточно ли просто прогона тестов? Возможно, надо хранить артефакты сборки, тогда удобнее использовать цель install.
    b. Есть ли задачи на другие цели, например, integration-test или package?
    c. Достаточна ли конфигурация surefire-плагина?
3. После прогона тестов вне зависимости от результата формируется Allure-отчёт.
    a. Где хранятся результаты прогона?

b. Есть ли дополнительные параметры, которые надо вывести в отчёт?

c. Есть ли специальная конфигурация allure? Обычно это файл config.yml с плагинами.

d. Всегда ли надо выводить Allure-отчёт?

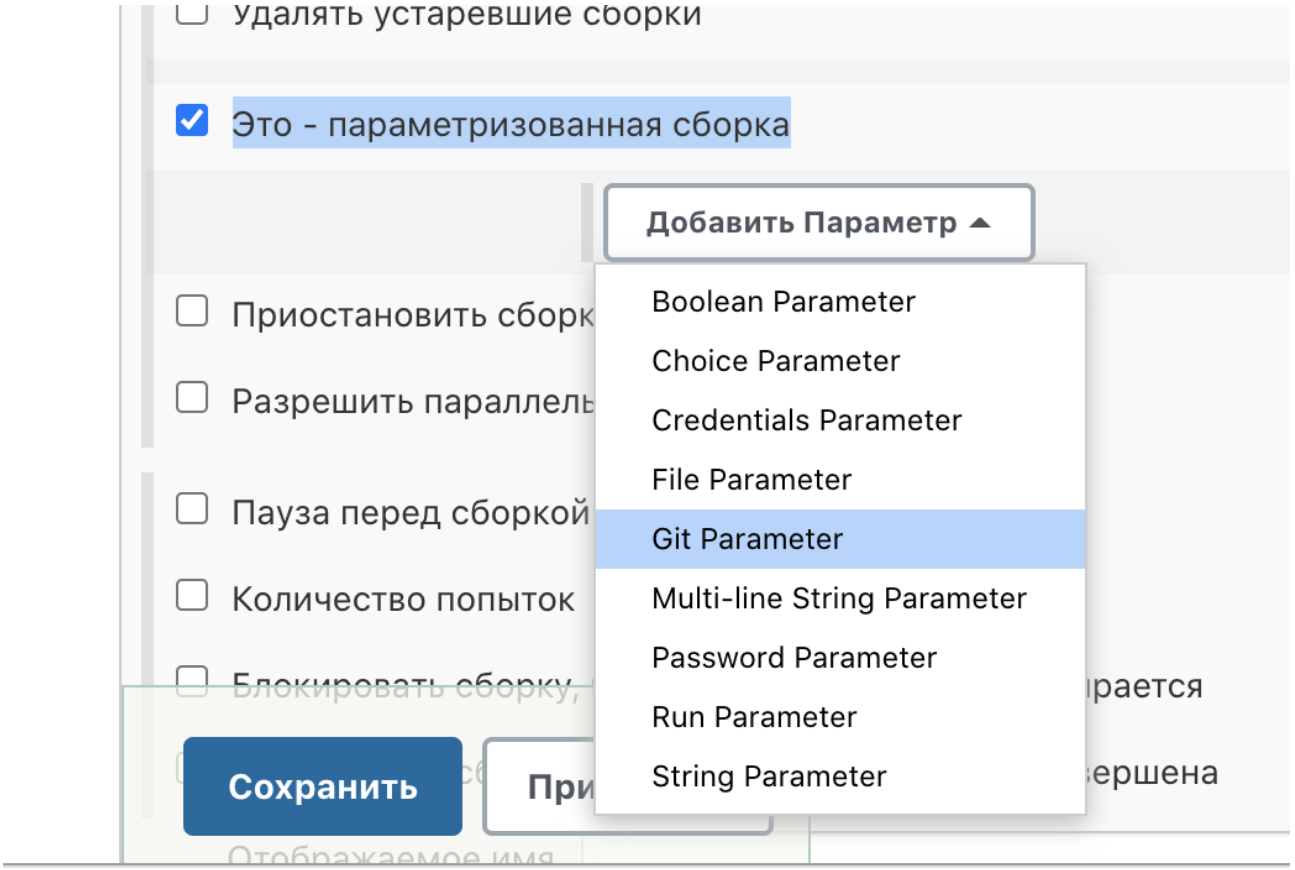
Когда ответите на эти вопросы, приступите к конфигурации.

Конфигурация задачи

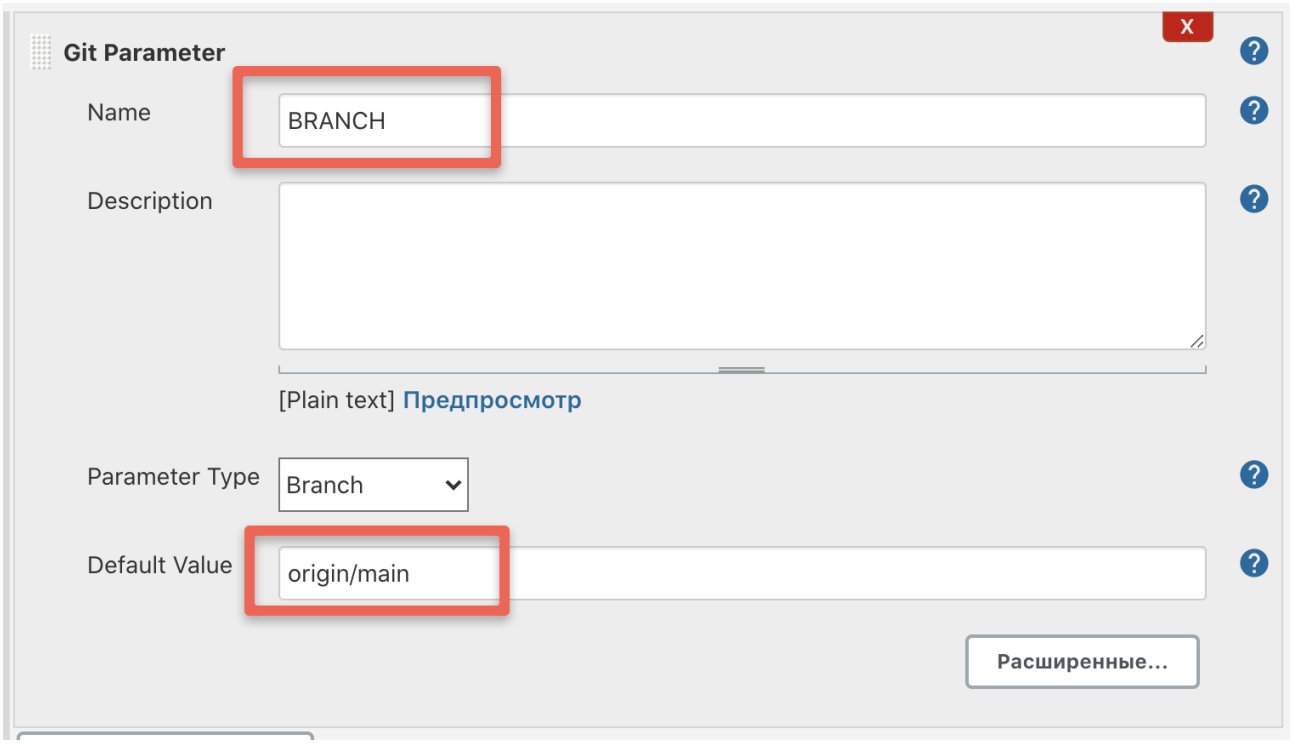
В General настраиваются параметры запуска: пауза перед сборкой, количество попыток, ссылка на репозиторий и так далее. Для нашей задачи полезны две функции:

1. Чекбокс GitHub project позволяет указать ссылку на репозиторий, чтобы открыть её в браузере.
2. Чекбокс — параметризованная сборка, которая позволяет добавлять параметры к сборке (ставим всегда).

После активации последнего чекбокса появляется возможность добавлять параметры. Добавим параметр, позволяющий подключать действующие ветки из репозитория:



Укажем требуемые параметры. По конвенции имён все параметры пишутся заглавными буквами:

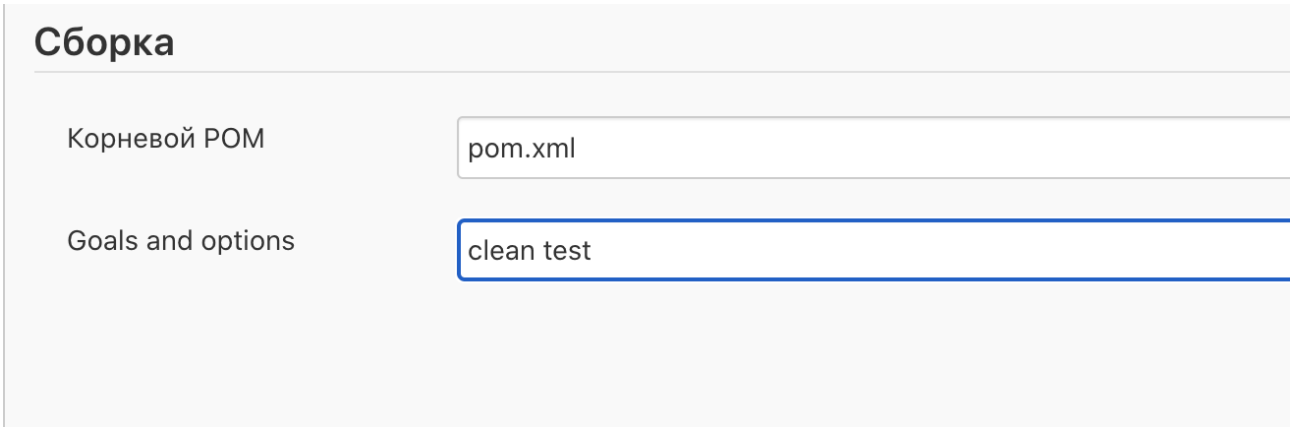


Теперь указываем, что будем использовать значение из переменной Branch. Переходим в раздел «Управление исходным кодом».

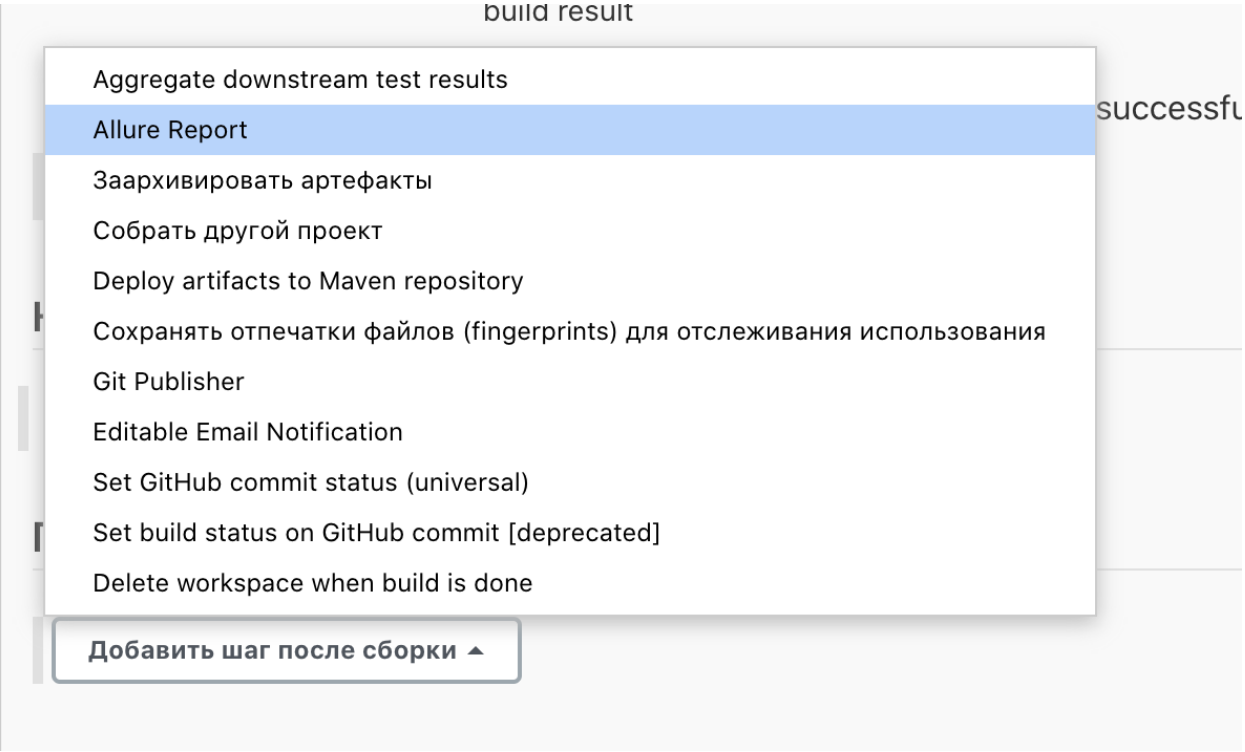
Активируем радиобаттон Git. В поле Repository URL указываем ссылку на репозиторий так, как она указывается в команде git clone. Если потребуется, указываем Credentials — данные для авторизации. В разделе Branches specified to build указываем значение ${BRANCH} — показываем, что надо использовать значение из заданной выше переменной.

Если потребуется, устанавливаем триггеры сборки, среду сборки и предварительные шаги.

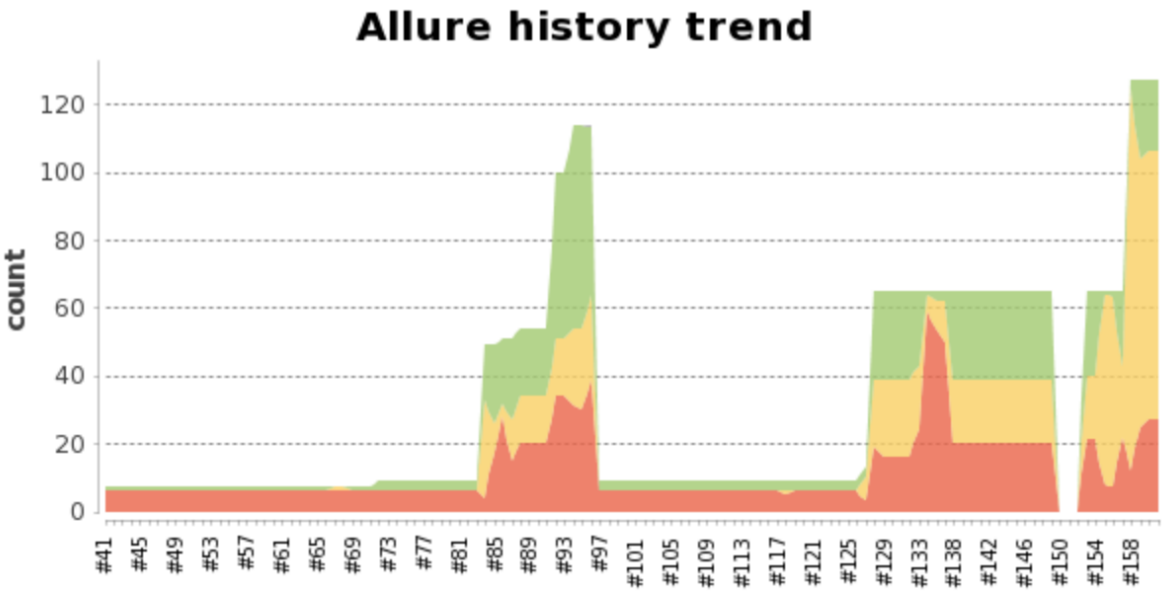
Переходим в раздел «Сборка». Так как проект создан специально под maven, здесь можно указать корневой pom (для многомодульных проектов), а также цели, которые надо выполнить. Слово mvn уже писать не требуется, достаточно указать сами цели, например:



Осталось добавить Allure-отчётность. Для этого в разделе «Послесборочные операции» выбираем Allure Report:



Указываем папку с результатами, в тестовом проекте это обычно target/allure-results. Далее сохраняем джобу и пытаемся её запустить на панели слева в пункте меню «Собрать с параметрами». Логи есть в пункте меню «Вывод консоли», Allure-отчёт — в пункте меню Allure report. На странице джобы справа появится тренд прогонов, при увеличении количества прогонов он выглядит так:



Практическое задание

1. Разверните задачу на сборку своего проекта.

2. Сдайте ссылку на джобу преподавателю.

Дополнительные материалы

1. Официальная документация.

2. Документация Allure.

3. Статья «Automated GUI testing: пошаговая инструкция».

4. Видеодоклад «Selenoid: запускаем Selenium-тесты в Docker-контейнерах».

Используемые источники

1. Официальная документация.

2. Jenkins: The Definitive Guide: Continuous Integration for the Masses by John Ferguson Smart.