Модуль 2. Непрерывная интеграция, доставка и обучение (CI/CD/CT) в проектах ML.

# В этом модуле:

Вы уже знаете из предыдущего модуля, что в программной инженерии придумано много способов повысить эффективность проекта. Одним из самых действенных способов повышения эффективности является автоматизация отдельных рутинных операций, связанных с созданием, сборкой и тестирование сложных проектов программного обеспечения, а также установкой этого программного обеспечения в промышленную среду и организацией эксплуатации. Методы, связанные с созданием, тестированием и сборкой проекта называются «непрерывная интеграция» (Continuous Integration, CI), а все, что связано с выводом решения в производственную среду эксплуатации (production, «прод») называется «непрерывная доставка» (Continuous Delivery, CD). Кроме того, для проектов актуальна автоматизация тестирования (Continuous Testing, CT).

В этом модуле описаны этапы и инструменты для CI/CD/CT для проектов машинного обучения.

Темы, изучаемые в модуле:

1. Автоматизация. Непрерывная интеграция, развертывание и тестирование.
2. Задачи и инструменты CI/CD/CT в проектах машинного обучения.
3. Пример организации CI/CD/CT в проектах машинного обучения с использованием Jenkins.

# Модуль 1. Юнит 1. Автоматизация. Непрерывная интеграция, развертывание и тестирование.

*Введение:* В этом юните вы узнаете о задачах CI/CD /CT. Отдельно более подробно разобрана наиболее важная задача: автоматизация.

*Содержание юнита:*

Современные программные системы состоят из множества отдельных взаимодействующих между собой модулей, над которыми могут работать различные команды проекта или даже разные компании. Вот пример такой системы, имеющей микросервисную архитектуру, в который отдельные части решения реализованы в виде отдельных независимых друг от друга сервисов, общение происходит через служебные интерфейсы и протоколы, например API, telnet, FTP, ssh, сокеты.

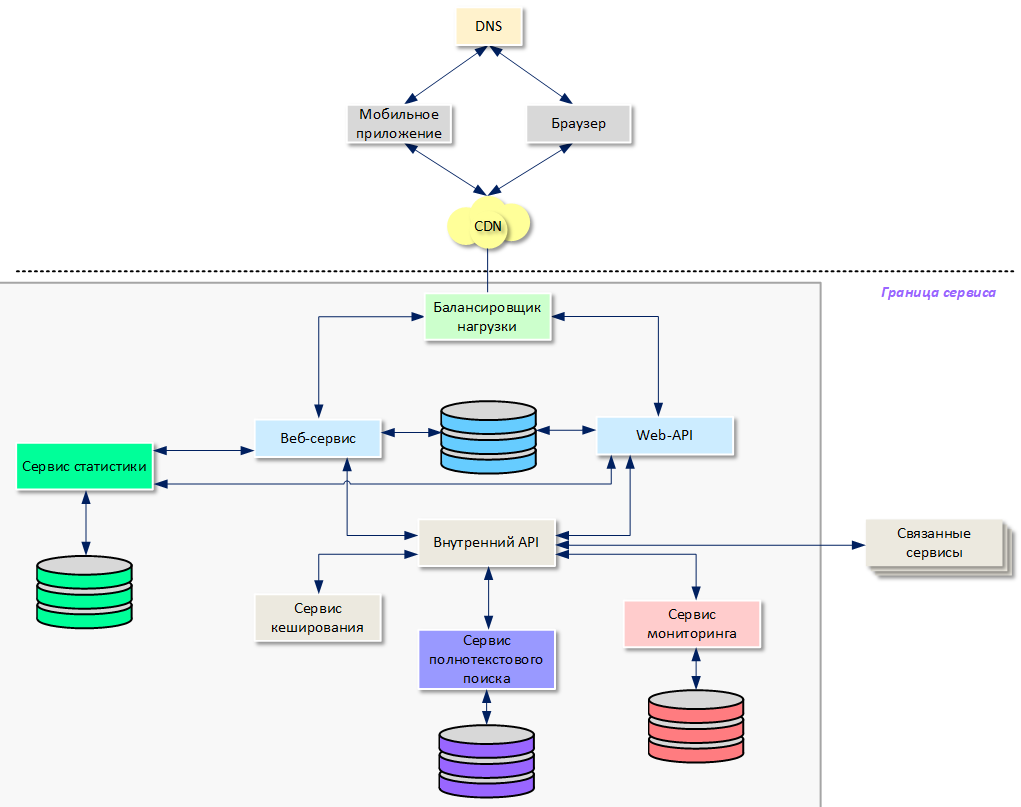
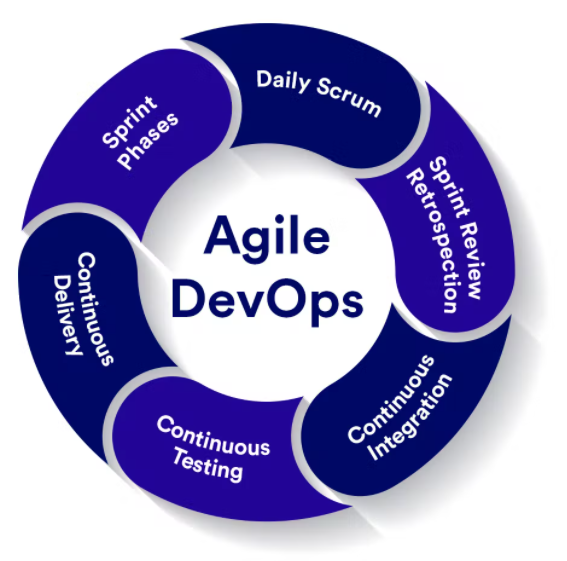


Рисунок из статьи «Архитектура современных веб-сервисов и способы их защиты» https://www.anti-malware.ru/analytics/Technology\_Analysis/Architecture-of-modern-web-services

Количество микросервисов в большом приложении может достигать сотен экземпляров, взаимодействие становится все более мложным. Управление созданием, тестированием, развертыванием и эксплуатацией подобных систем стало невозможным без специальных средств автоматизации, что привело к созданию методологии DevOps. Уже знакомые нам из предыдущего модуля гибкие agile-методологии управления проектами включают в себя несколько стандартных шагов, в том числе относящиеся к DevOps:



Мы помним, что цель любого проекта: предоставить заказчику (пользователю) решение, содержащее как можно меньше проблем, как можно быстрее. Рассмотрим отдельные фазы автоматизации, которые позволяют существенно упростить и ускорить процессы сборки и развертывания программного продукта.

1. Непрерывная интеграция (Continuous Integration, CI)

CI позволяет участникам проекта писать новый и обновлять существующий код разрабатываемой системы и сохранять (commit, «коммитить») эти изменения в едином пространстве для управления кодом. Такое пространство называют репозиторий проекта. На практике участники проекта могут сохранять множество изменений в разных файлах в течении дня. Для обеспечения качества проекта изменения в коде должны быть проверены. Не каждое внесенное изменение удается проверить человеку, особенно если возможные ошибки касаются взаимодействия нескольких модулей. На этом этапе необходимым шагом является автоматизация нескольких задач, которые могут запускаться автоматически при обнаружении факта обновления кода, либо по расписанию:

* проверка правильности написания кода (автоматизированный код ревью)
  + выполнение правил именования функций и переменных,
  + наличие комментариев,
* запуск сценариев тестирования для проверки надежности кода.

Не следует думать, что технологии CI являются средством от всех болезней. Например, CI точно не принесет пользы в следующих случаях:

* разработчики редко «коммитят» в общий репозиторий проекта,
* команды разработчиков отдельных модулей работают в своих отдельных ветках, которые редко интегрируются с главной веткой.

Таким образом CI **непрерывно интегрирует** написанный код в общее хранилище кода проекта, своевременно обнаруживая конфликты между отдельными частями кода.

1. Непрерывная доставка (Continuous Delivery, CD)

CD содержит средства автоматизации для развертывания проекта в среде, в которой эта разработанная система будет использоваться. Иногда этот процесс называют «установка и настройка программного обеспечения», также используют жаргонный термин «выкатить в прод». Процесс развертывания программного обеспечения для промышленной эксплуатации состоит из множества отдельных шагов:

* настроить оборудование,
* создать необходимую среду выполнения программ,
* выполнить установку и настройку,
* создать необходимые пользовательские роли,
* обеспечить мониторинг.

CD обеспечивает **непрерывную доставку** разработанного программного обеспечения в среду эксплуатации, то есть позволяет автоматизировать настройку и установку с использованием специальных скриптов автоматизации, собираемых в единый конвейер.

1. Непрерывное тестирование (Continuous Testing, CT)

В отличие от рассмотренных CI/CD, главная цель которых это ускорение процессов за счет автоматизации рутинных процедур, назначение CT заключается в обеспечении необходимого качества за счет автоматизации различных тестов на различных этапах. Часто CT не рассматривают отдельно при изучении DevOps, предполагая, что достаточно описать автоматизацию тестов как часть процесса сборки или развертывания. Однако роль тестирования становится все значительнее. И, как мы увидим в дальнейшем, тестирование в проектах машинного обучения является ключевым фактором успеха проекта. В проекте возможны следующие виды тестов:

* юнит тестирование (unit testing),
* нагрузочное тестирование (load testing),
* тестирование соответствия стандартам (conformance тестирование),
* тестирование взаимодействия (interoperability testing)

CT обеспечивает **непрерывное тестирование** проекта на всех этапах.

На каждом этапе жизненного цикла проекта высока вероятность человеческой ошибки, например

* программист может допустить ошибку в коде,
* инженер по работе с данными может написать неверный SQL запрос,
* при сборке в проект может быть включена ошибочная версия библиотеки,
* настройщик системы может установить неправильные значения переменных среды окружения.

CI/CD/CT совместно призваны решить эту проблему, эффективно дополняя друг друга.

*Тест*

1. Что такое непрерывная интеграция
   1. Непрерывно работающий сервис по сбору данных от микросервисов
   2. **Автоматизированный процесс сборки общего программного решения из отдельных элементов**
   3. Постоянный мониторинг всех процессов
   4. Ручная сборка программного пакета
2. Как расшифровывается CT
   1. Common tools
   2. Continuous training
   3. **Continuous testing**
   4. Cost transfer
3. Установите соответствие между CI/CD/CT и задачами

|  |  |
| --- | --- |
| Автоматический запуск тестов | CI, CT |
| Настройка окружения для запуска программы | CD |
| Проверка изменений в программном коде | CI |
| Создание ролей пользователя | CD |

1. Что может быть триггером для запуска процесса сборки программного пакета в CI
   1. **Факт публикации изменений кода в репозитории проекта**
   2. **Расписание**
   3. **Ручной запуск**
   4. Негативное письмо от заказчика

# Итоги/выводы

В этом юните вы узнали о этапах CI/CD/CT, являющихся ключевыми в DevOps. Для каждого их этих этапов существует характерный для них перечень задач, а именно

1. Непрерывная интеграция
   1. Проверка правильности сделанных изменений
   2. Публикация всех изменений программного кода, служебных скриптов, настроек в главном репозитории проекта
   3. Сборка всех изменений в единый проект
   4. Тестирование совместимости измененных частей проекта
   5. Проверка функциональности
2. Непрерывная доставка
   1. Настройка оборудования
   2. Настройка среды выполнения программы
   3. Установка и настройка системы
   4. Создание пользовательских ролей
   5. Организация мониторинга
3. Непрерывное тестирование
   1. Тестирование отдельных модулей (юнит тесты)
   2. Тестирование совместимости
   3. Тестирование производительности
   4. Тестирование функций

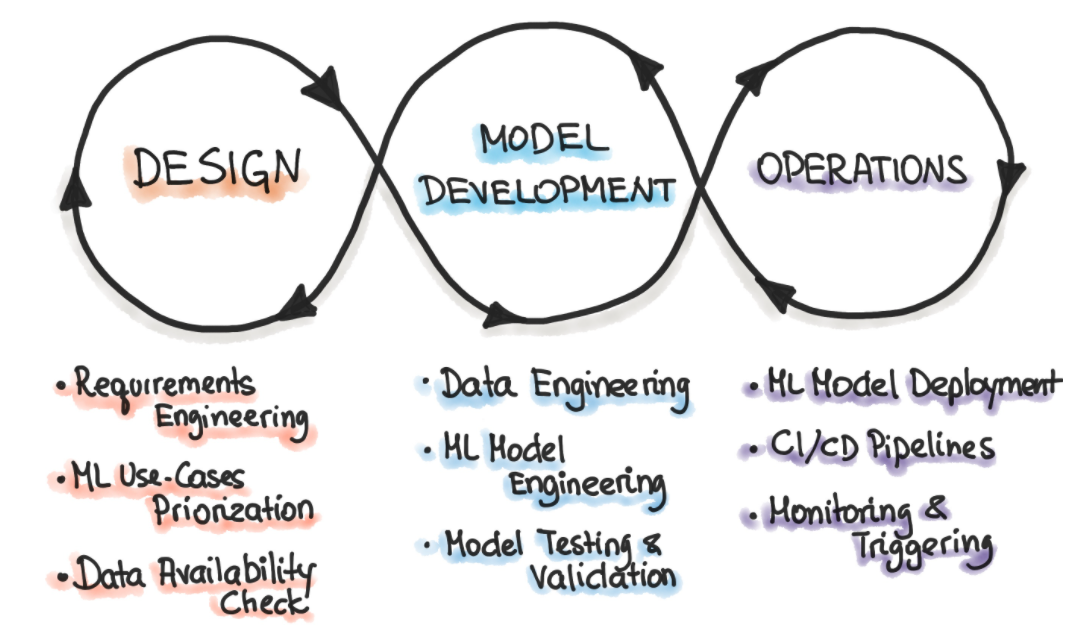
В этом юните речь шла о DevOps для проектов разработки программного обеспечения, а в следующем юните вы узнаете об отличиях в CI/CD/CT в проектах машинного обучения.

# Модуль 1. Юнит 2. Задачи и инструменты CI/CD/CT в проектах машинного обучения.

*Введение:* В этом юните мы посмотрим особенности CI/CD/CT в проектах машинного обучения, отличающие их от классических проектов разработки программного обеспечения.

*Содержание юнита:*

Так же как и в любом проекте разработки программного обеспечения, проект машинного обучения начинается с фазы системного дизайна (system design), на котором внимательно анализируются требования заказчика, полученные в техническом задании, проверяются имеющиеся данные, делаются предположения о том, с помощью каких моделей машинного обучения можно было бы решать задачу. Следующая фаза, разработка модели (model development) состоит из анализа данных, создания и обучения моделей и их валидации для возможности использования в промышленной среде. И следующий, заключительный этап, связан с эксплуатацией модели (operations). На этом этапе происходит развертывание модели, выполнение логики, заложенной в пайплайны CI/CD, организация мониторинга работы модели и качества данных. Важно, что эти фазы проходятся в проекте в цикле.



https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning#ml\_pipeline\_triggers

Практика DevOps предполагает контроль и автоматизацию и на каждом этапе построения системы: интеграция, тестирование, релиз, развертывание и управление инфраструктурой. Давайте обратим внимание на ряд отличий, которые возникают в проекте машинного обучения по сравнению с «классическим» проектом разработки программного обеспечения. Проект машинного обучения предполагает эксперимент для построения модели машинного обучения: с использованием имеющегося датасета для обучения создается система, умеющая делать выводы на основе изучения этого датасета. Однако реальные задачи предполагают большее. Построенная модель должна хорошо работать не только на учебном датасете, но и в реальной среде эксплуатации, на реальных данных. Эти данные имеют свойство меняться со временем, поскольку как в природе, так и в технологических процессах нет ничего постоянного. Как следствие, качество самой хорошей модели машинного обучения начинает ухудшаться практически сразу после ее вывода в эксплуатацию. И для проекта машинного обучения быстрая повторяемость жизненного цикла проекта становится еще более актуальной, чем для разработки обычного программного обеспечения, так как на каждой итерации цикла система машинного обучения получает новую важную информацию, которую использует для улучшения качества работы. Следовательно, построение конвейера, обеспечивающего обучение, интеграцию, развертывание, мониторинг проекта машинного обучения, является ключевым.

Система машинного обучения, используемая для промышленных задач, это комплексное, крупномасштабное программное обеспечение, поэтому для его создания и управления логично использовать принципы DevOps. Однако есть важные отличия, которые необходимо учитывать в проекте ML:

1. Участники команды проекта ML могут не быть высоко квалифицированными программистами, поскольку сосредоточены в основном на изучении данных и эксперименте для построения эффективной модели
2. Разработка модели машинного обучения носит экспериментальный характер, поэтому для получения наиболее эффективной модели приходится пробовать разные методы, конфигурации, алгоритмы, пытаясь найти их оптимальное сочетание. Важнейшая задача MLOps состоит в обеспечении возможности управлять версиями изменяющихся сущностей для того, чтобы была возможность определить наиболее подходящий вариант, обеспечив возможность его быстрого воспроизведения, максимально используя уже созданное (написанный код, посчитанные параметры модели, найденные сочетания версий библиотек).
3. В тестировании к обычному набору тестов, рассмотренному в предыдущем юните, теперь добавляется еще проверка данных, оценка и проверка моделей.
4. Развертывание модели машинного обучения может потребовать больше шагов от MLOps специалиста, поскольку для обеспечения повторяемости работы модели в промышленной среде необходимо скрупулезно по шагам настроить эту среду.
5. При эксплуатации системы машинного обучения необходимо иметь возможность постоянно мониторить качество работы как самой модели, так и данных, на которых она работает.

*Тест*

1. Что отличает MLOps CI от обычного проекта
   1. **Наличие датасетов**
   2. **Наличие моделей машинного обучения**
   3. Наличие комментариев в коде
   4. Повышенные требования к производительности системы
2. Что отличает MLOps CD от обычного проекта
   1. **Необходимость более скрупулезно организовывать среду выполнения программы для более качественного инференса (работы модели)**
   2. **Необходимость анализировать данные, поступающие при эксплуатации системы, на которых работает модель**
   3. Повышенные требования к работе аппаратного обеспечения
   4. Большее количество ролей в проекте
3. Какой еще смысл возникает у термина CT, который означает Continuous Testing, применительно к проектам машинного обучения
   1. Common Thinking
   2. Collaborative Testing
   3. Continuous Training
   4. Нет никаких изменений у термина CT
4. Какие параметры важно поставить под мониторинг при развертывании и эксплуатации модели машинного обучения
   1. **Оперативную память системы**
   2. **Скорость работы модели**
   3. **Качество поступающих данных**
   4. Температуру окружающей среды

# Итоги/выводы

Мы изучили особенности CI/CD/СT для проектов машинного обучения. Вот наиболее важные отличия CI/CD/CT MLOps от обычного проекта:

* MLOps CI это не только тестирование, валидация кода и компонентов, но еще и тестирование и валидация данных и моделей.
* MLOps CD это не просто пакет или сервис, а система последовательных операций (конвейер машинного обучения, ML training pipeline)
* MLOps CT это новая сущность, уникальная для ML, суть ее в автоматическом переобучении в процессе работы моделей машинного обучения на новых данных.

В следующем юните мы познакомимся с очень популярным в настоящее время инструментом для решения задач CI/CD/CT, системой Jenkins.

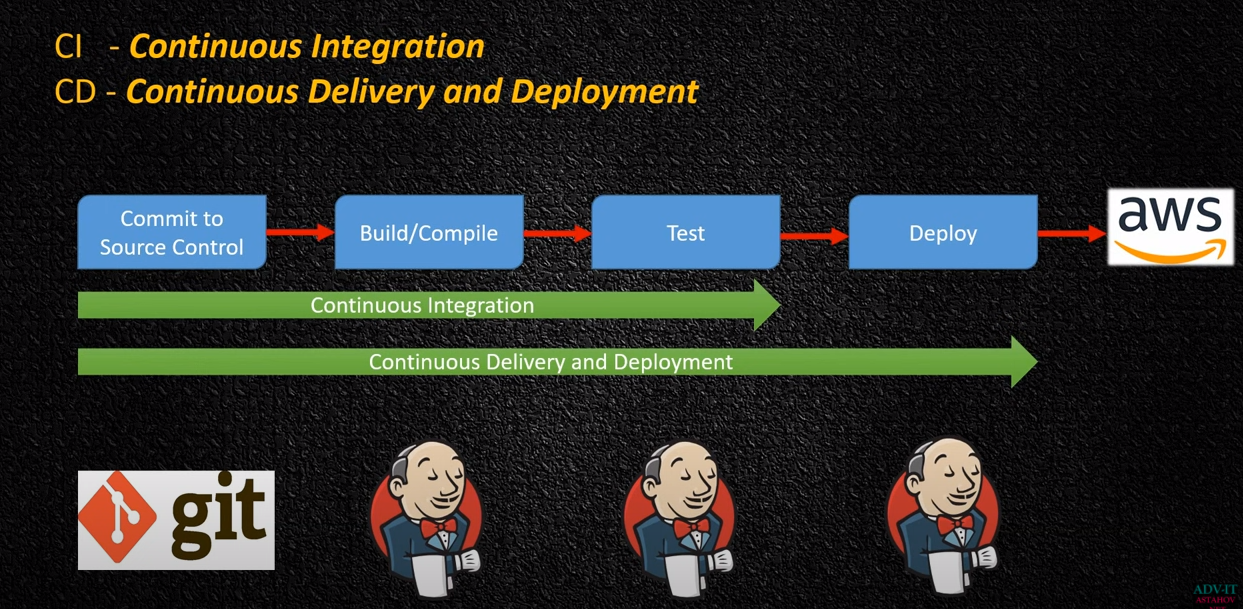
# Модуль 1. Юнит 3. Пример организации CI/CD/CT в проектах машинного обучения с использованием Jenkins.

*Введение:* В этом юните мы познакомимся с Jenkins, позволяющим решать задачи CI/CD/CT, это позволит быстрее переходить к практическим задачам.

*Содержание юнита:*

Итак, приступим к изучению Jenkins, очень популрного инструмента для автоматизации различных операций, возникающих в CI/CD/CT. Как обычно, у любого популярного инструмента есть объяснения этой популярности. Jenkins написан на широко распространенном языке программирования Java, является бесплатным и имеет большое сообщество, которое занимается его развитием, созданием лучших практик использования.

Jenkins можно применять практически на всех этапах жизненного цикла проекта.



Справедливости ради необходимо отметить, что у Jenkins есть альтернативы: Bamboo (Atlassian), Circle CI, Team City, GitLab CI/CD, Travis, Harness.

Установка Jenkins выполняется очень просто и подробно описана на сайте <https://www.jenkins.io/>. На этом сайте можно найти последнюю версию LTS и обновления weekly. Там же можно найти требования к системе, на которой устанавливается Jenkins <https://www.jenkins.io/doc/book/installing/linux/>. Рекомендуем перед началом работы внимательно ознакомиться с этими требованиями.

Важно знать, что поддерживается только Java 8.

***cat /etc/os-release***

***java -version***

***sudo apt-get install openjdk-8-jre***

Перед установкой необходимо добавить ключ в систему

***curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null***

Затем добавить информацию в установщик apt

***echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \***

***https://pkg.jenkins.io/debian-stable binary/ | sudo tee \***

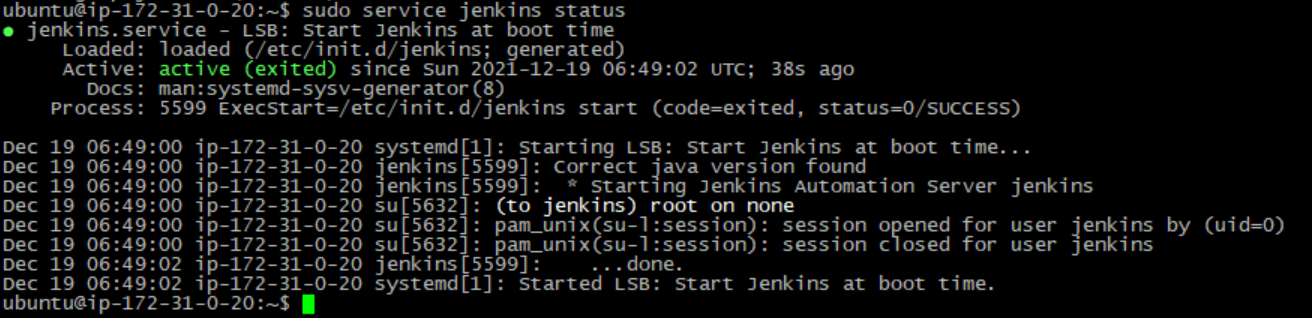
***/etc/apt/sources.list.d/jenkins.list > /dev/null***

Обновить установщик

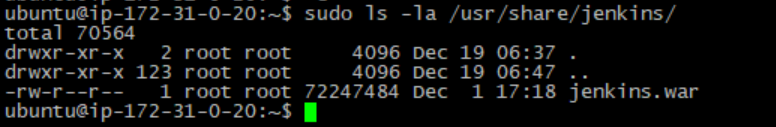
***sudo apt-get update***

Установить Jenkins

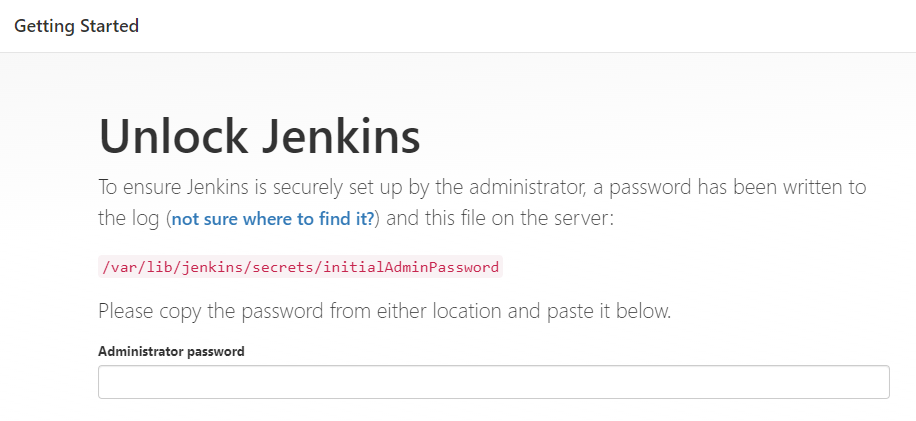
***sudo apt-get install Jenkins***



Jenkins стал работать как web сервер на порте 8080

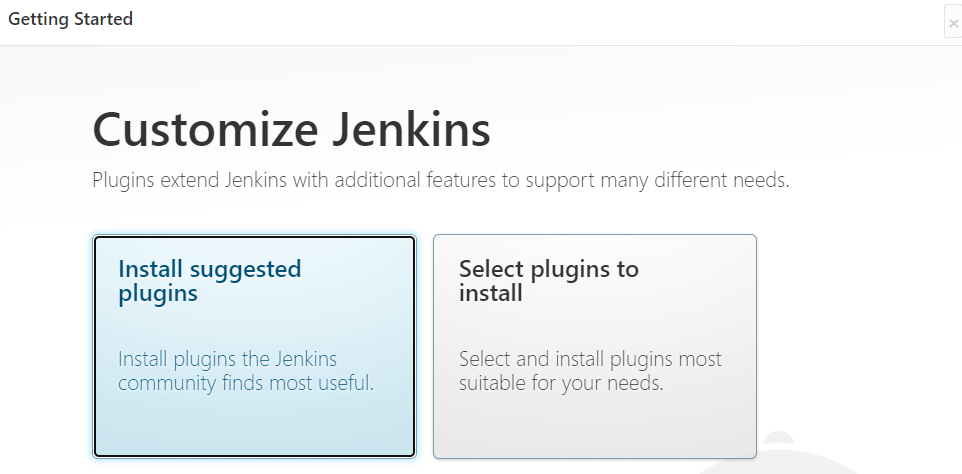


Для обновления версии можно просто обновить этот war файл и перезапустить сервис.



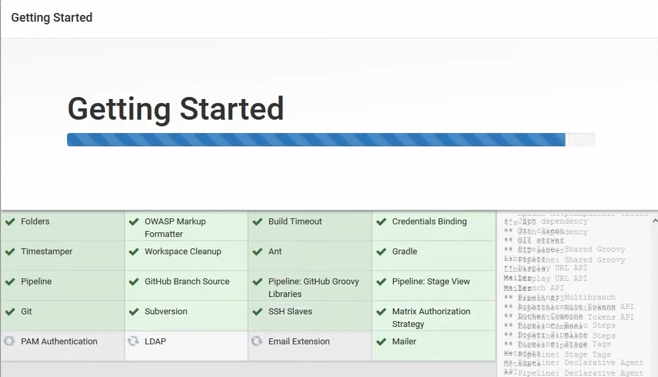
Необходимо скопировать текст с сервера (один раз).

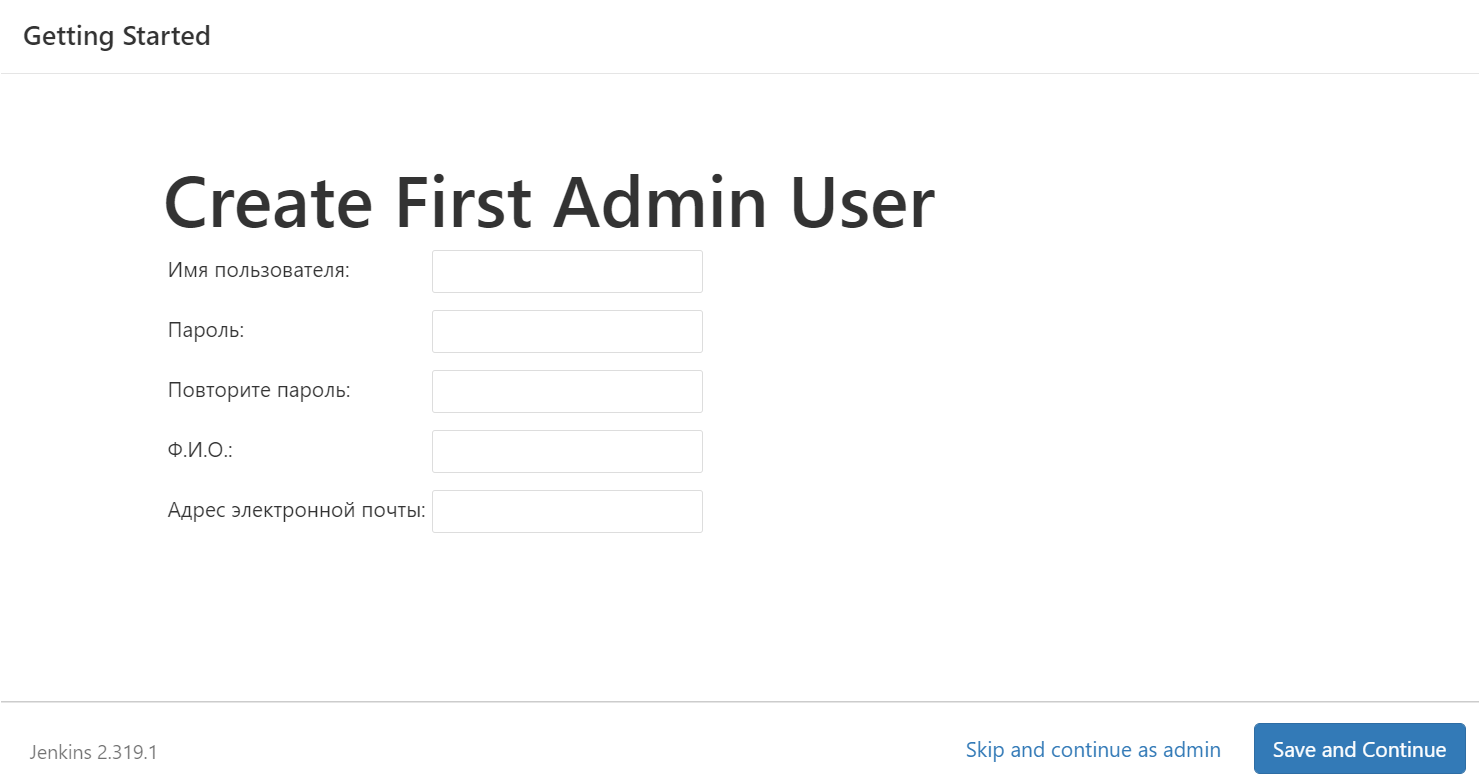
***sudo vi /var/lib/jenkins/secrets/initialAdminPassword***

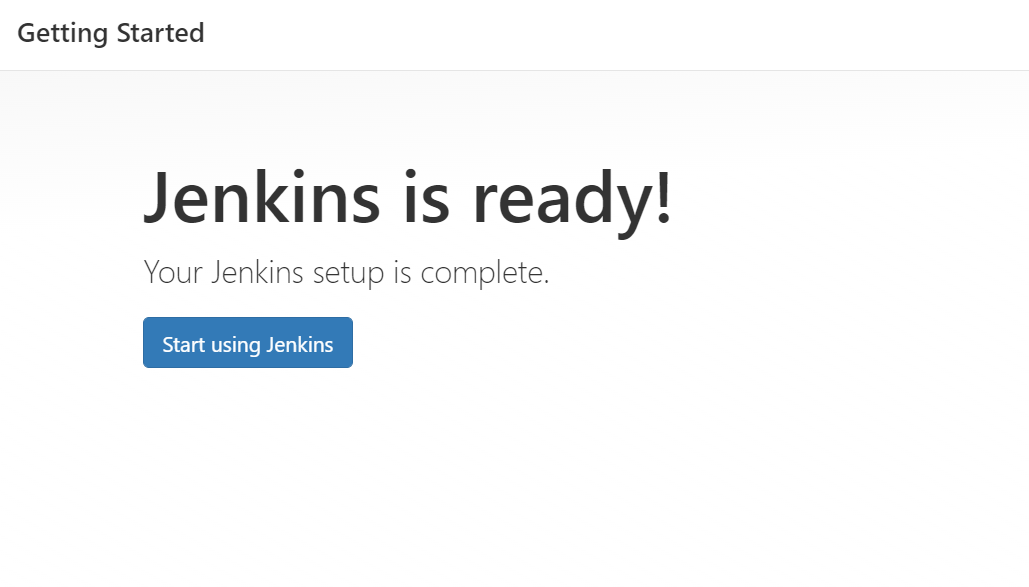


Плагины Jenkins это «коробочка lego», внутри Jenkins не много функций по умолчанию, нужные функции добавляются плагины. Можно делать свои плагины. Плагинов очень много. Пользуются проверенными привычными.

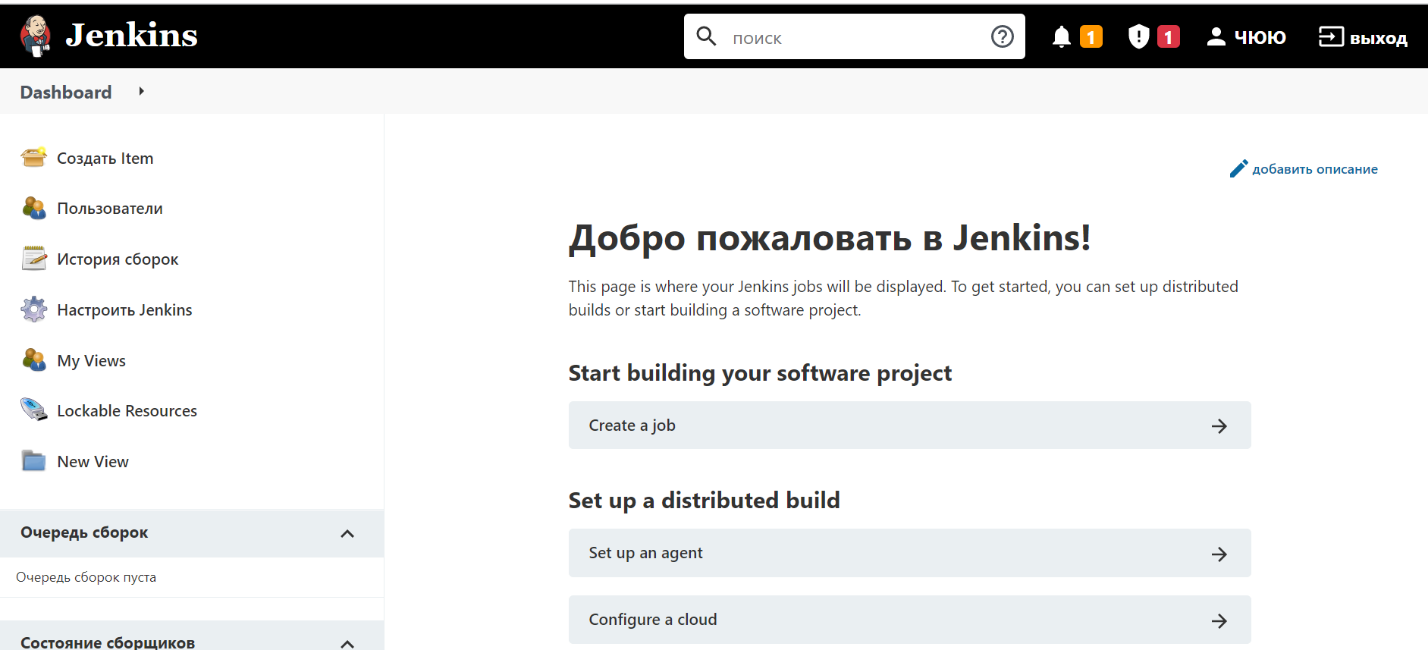
Установка рекомендованных плагинов



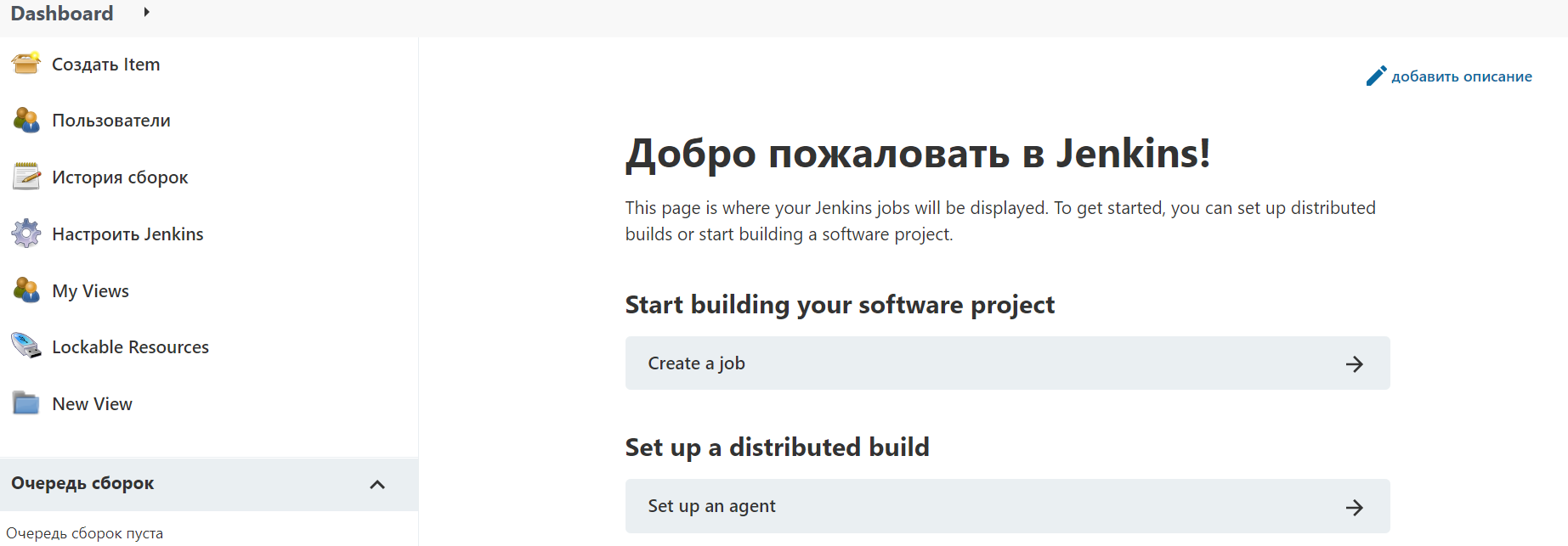


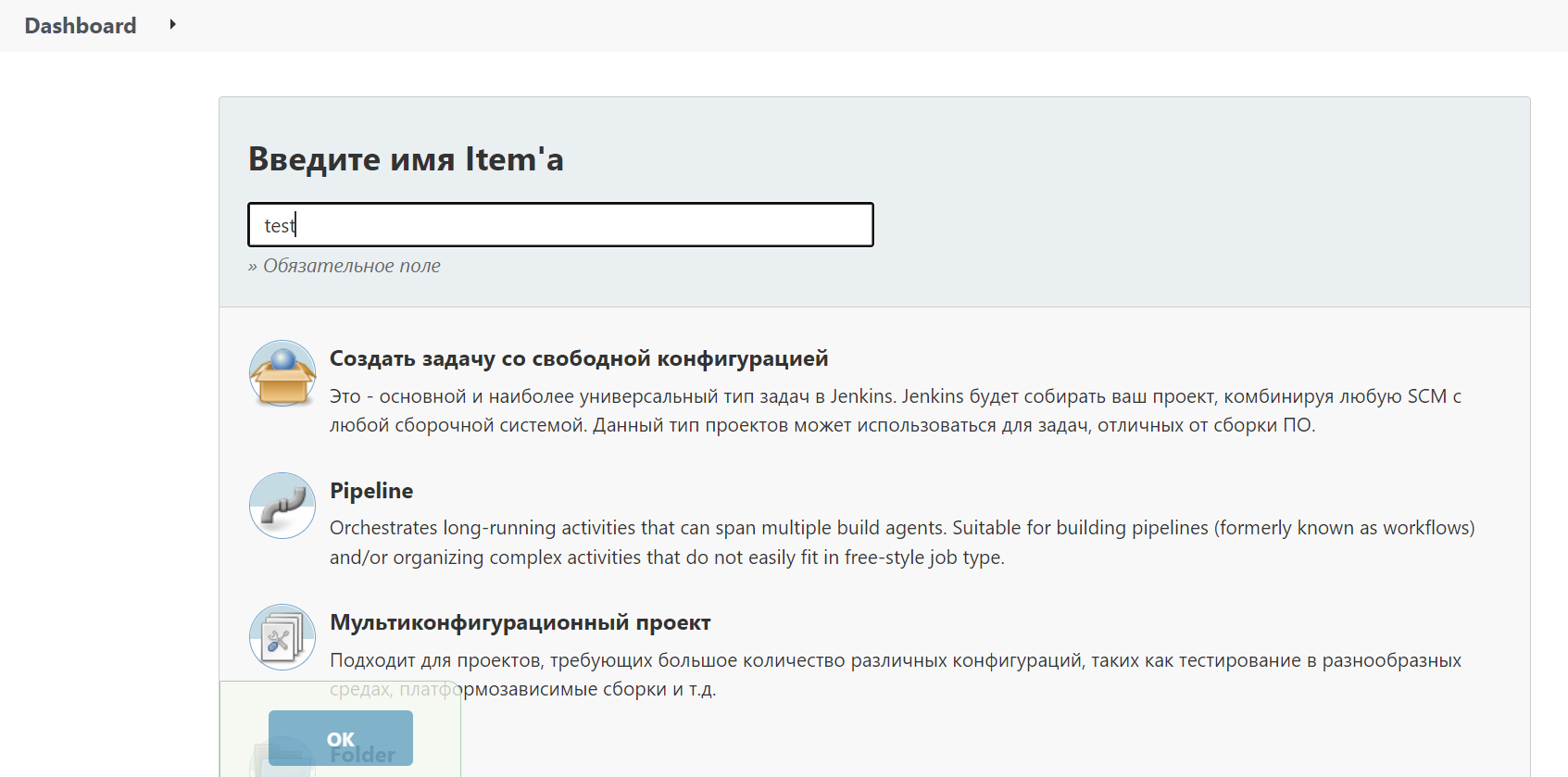


Запуск консоли управления Jenkins

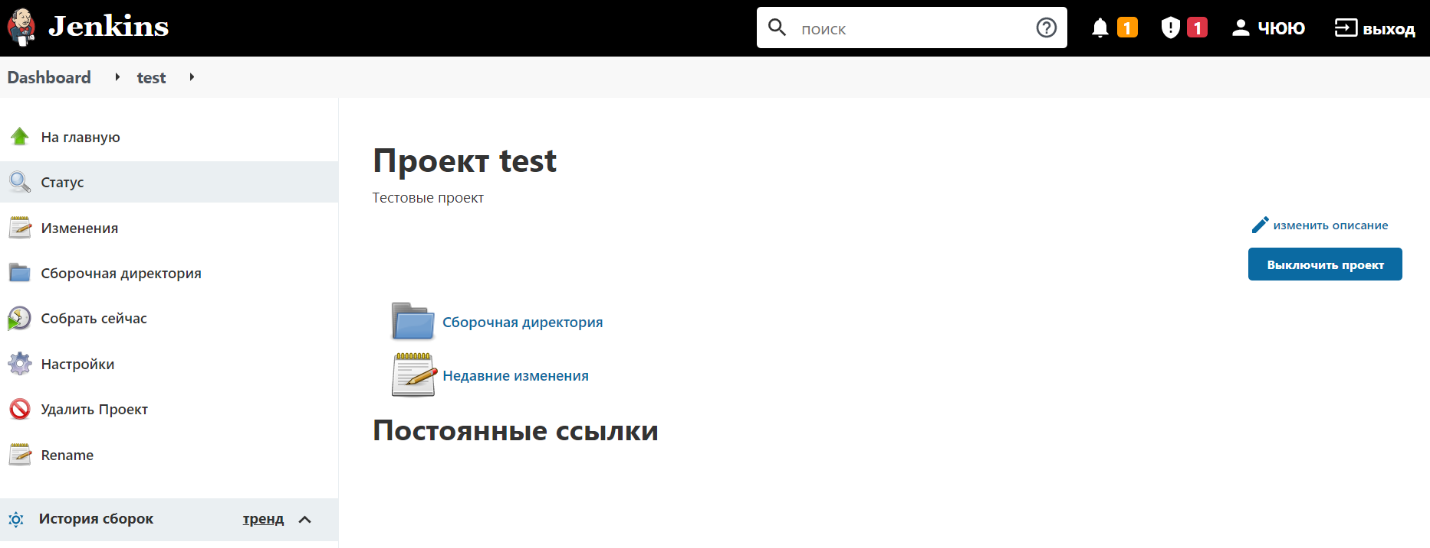


Создание тестового job



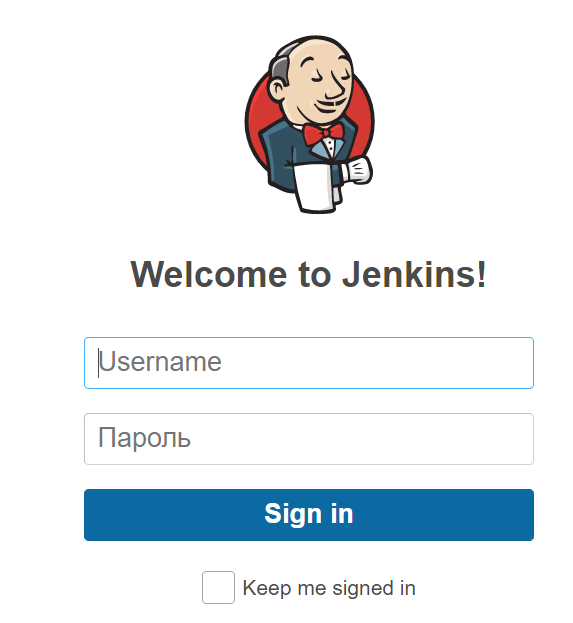


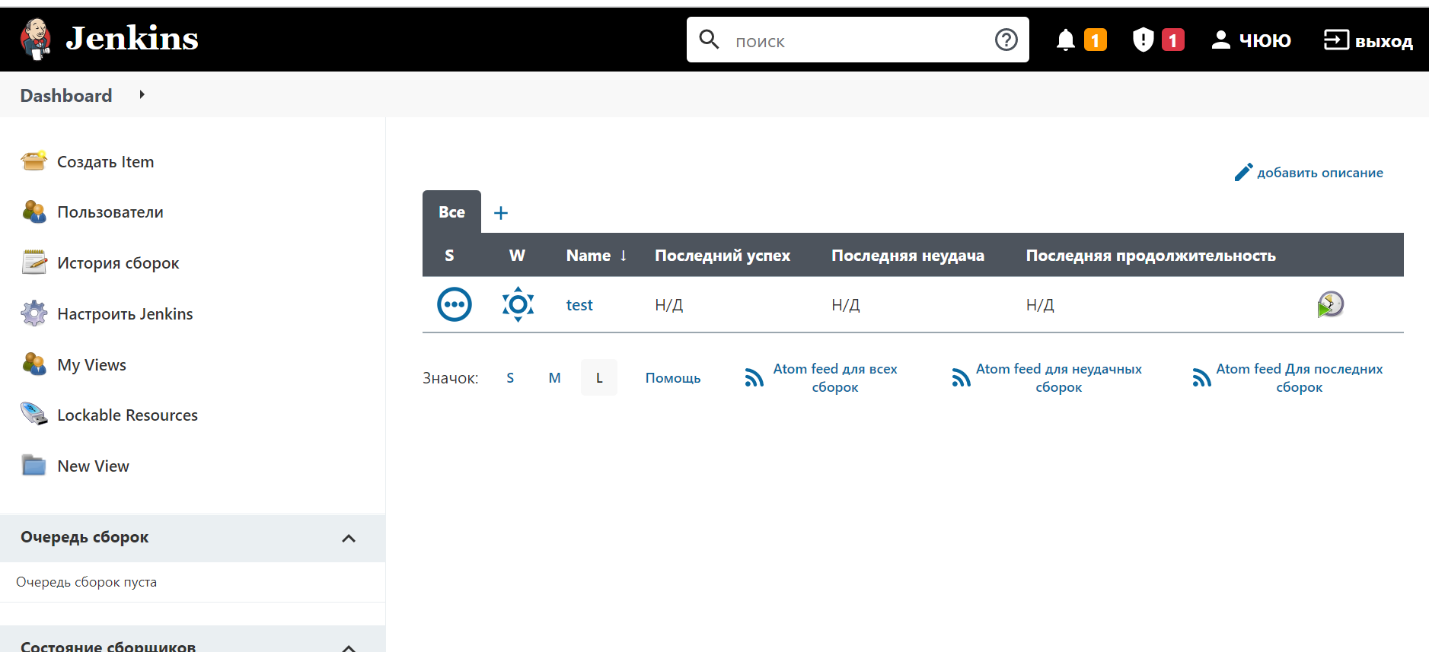
«Создать задачу со свободной конфигурацией» и ОК.

Выбираем опции управления проектом на разных этапах (чем больше плагинов, тем больше опций). И сохраняем проект.

# Администрирование Jenkins

Необходимо залогироваться





Создать Item

Пользователи

История сборок

Настроить Jenkins

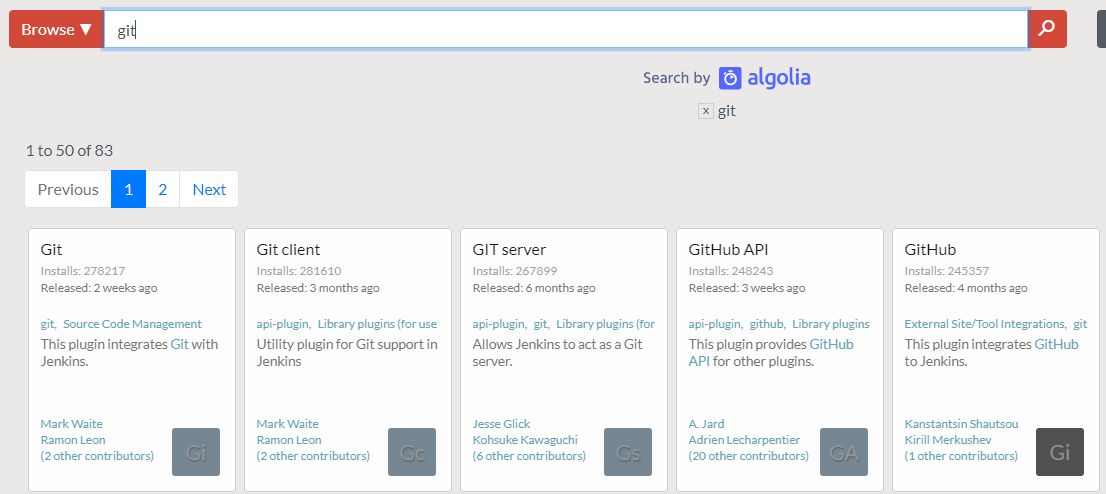
MyViews

Очередь сборок: указываются сборки (jobs)

Состояние сборщиков: можно определить количество одновременно выполняемых сборок (jobs), в соответствии с имеющимися ресурсами.

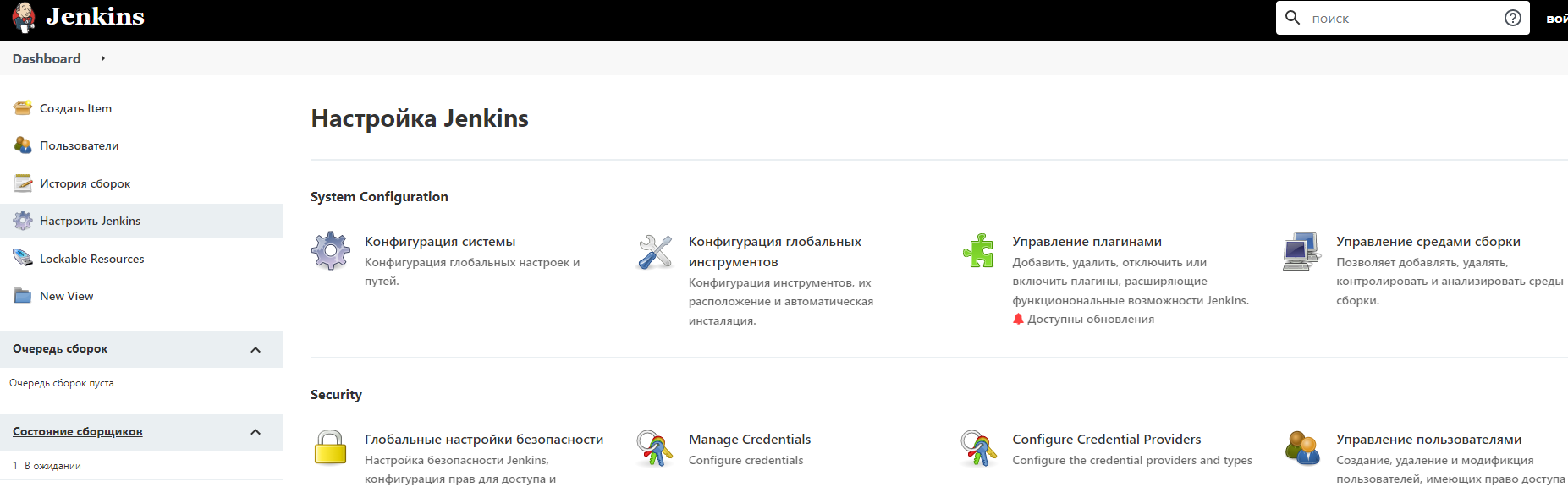
Можно к главному серверу Jenkins (master) добавить вспомогательные серверы (slaves, agents), на которых будут выполняться сборки (jobs)

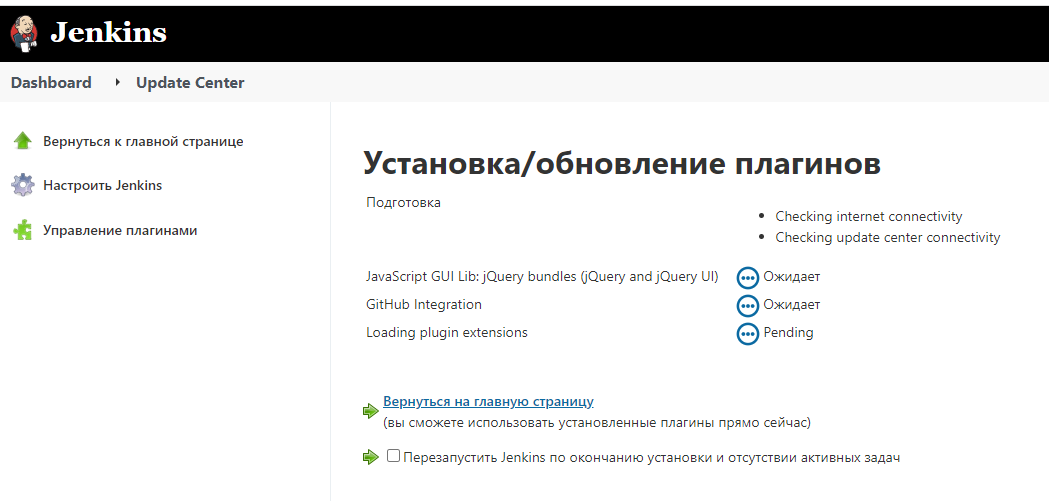
Плагины можно найти на сайте <https://plugins.jenkins.io/>



Лучше выбирать самый популярный плагин.

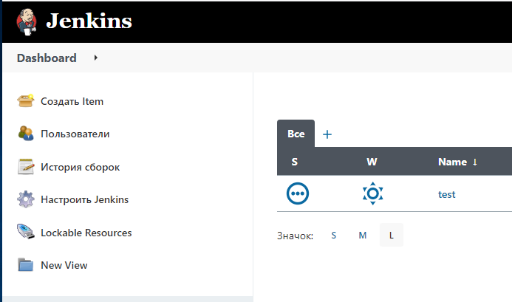
Все дополнительные функции в Jenkins реализуются через плагины.

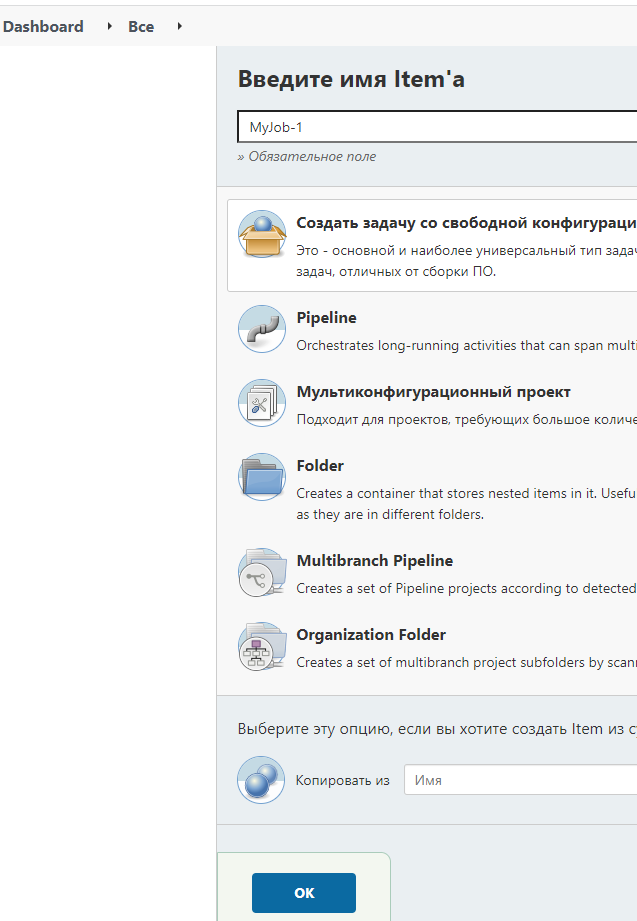




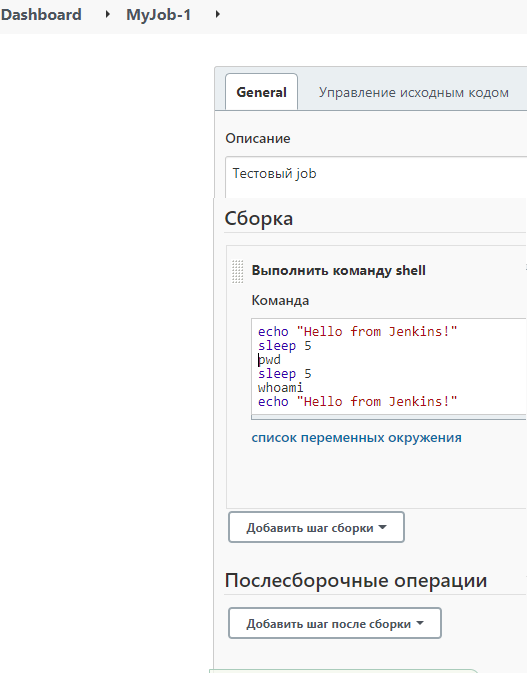
# Создание Jobs

## Создание «игрушечного» job

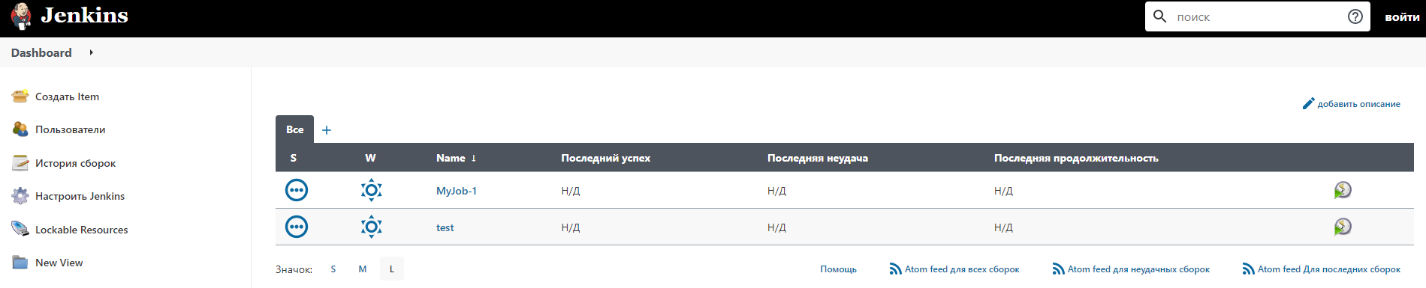




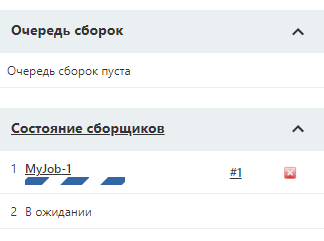
Можно пропустить все пункты, добавить только выполнение shell команд на этапе «Сборка».



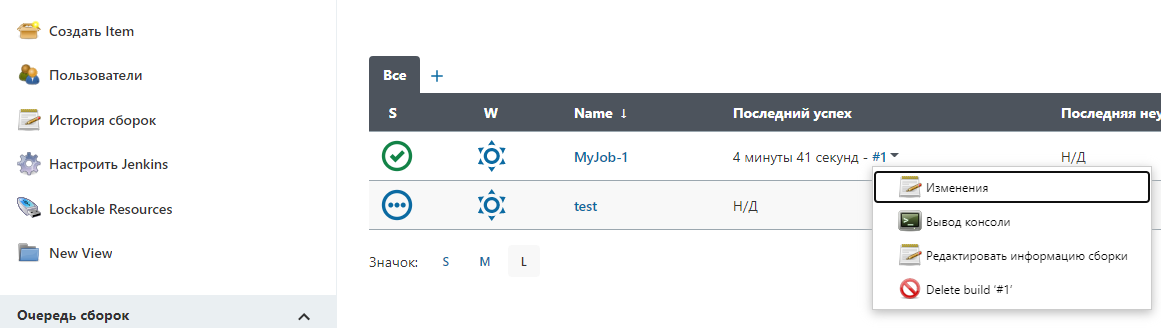
Новый job появился в списке и мы можем запустить его.

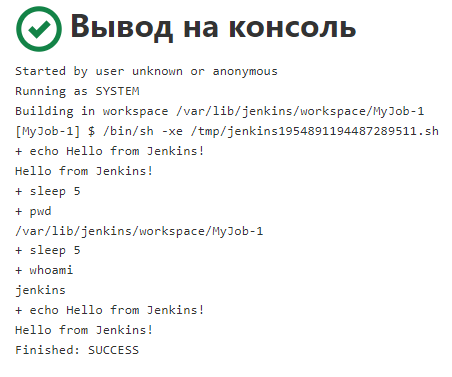


Видим, что начал выполняться.



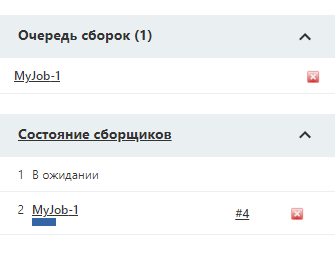
Консоль вывода:



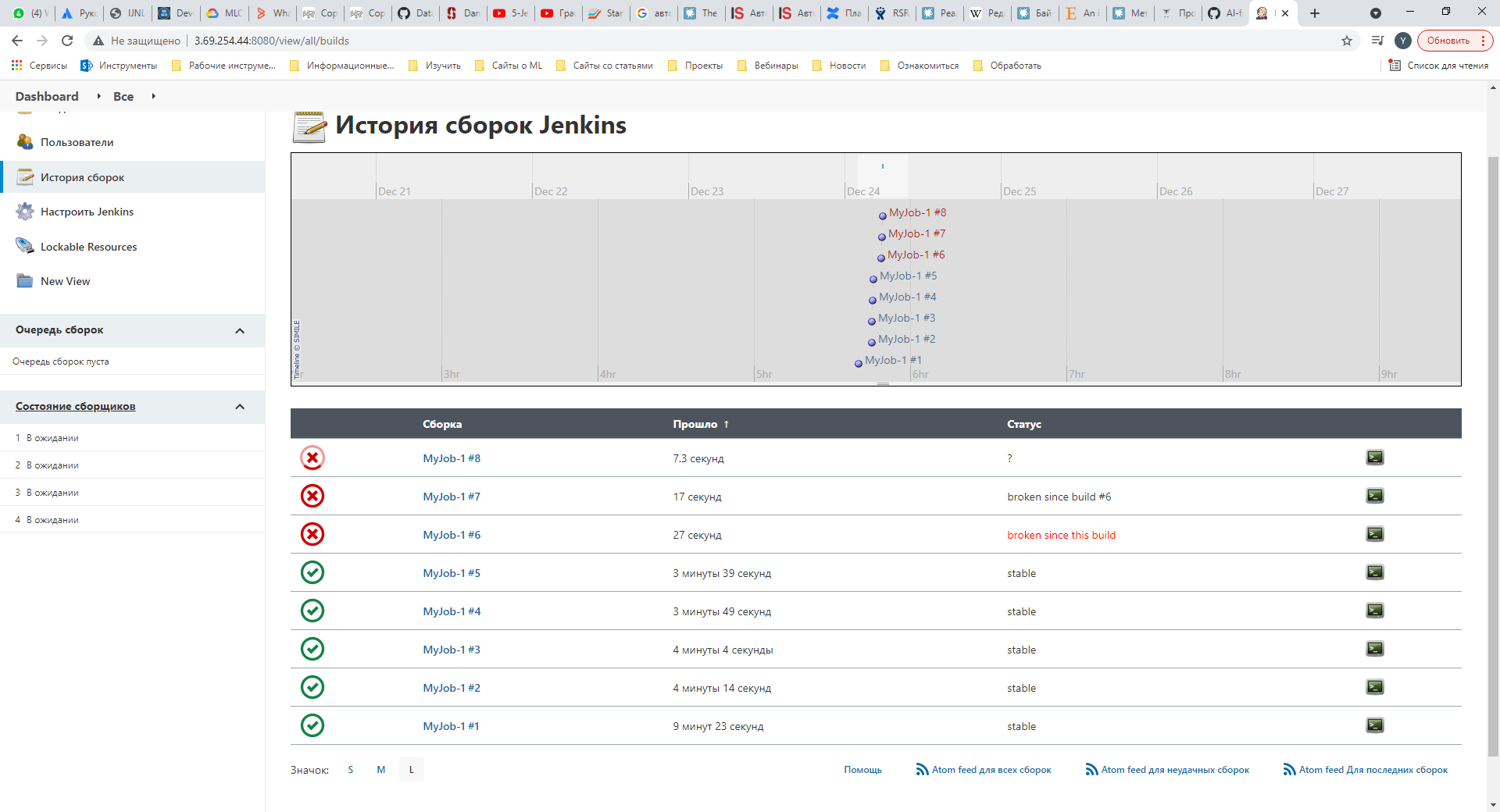


Можно запустить несколько раз и увидим несколько в очереди выполнения.

Количество job в очереди и в текущей обработке можно настраивать.



История сборок



Создание job для модели машинного обучения аналогично уже описанному процессу, за исключением того, что отдельными блоками автоматизации будут стандартные шаги в проекте машинного обучения.

Обучение модели

Вывод модели в эксплуатацию

Предобработка данных

И, в заключении, несколько подсказок на разные непредвиденные ситуации.

1. Если забыли пользовательский пароль от Jenkins.
   1. Остановить службу

***sudo service Jenkins stop***

* 1. Скорректировать параметр useSecurity в файле /var/lib/Jenkins/config.xml
  2. Запустить службу Jenkins

***sudo service Jenkins start***

*Тест/задание*

Создание и автоматизация пайплайна разработки и тестирования модели машинного обучения.

Шаги пайплайна

1. Скачать датасет из github
2. Провести обработку датасета. Разделить модель на тренировочные данные и тестовые.
3. Создать и обучить на данных датасета модель машинного обучения
4. Сохранить модель.
5. Запустить модель на тестовых данных.

# Итоги/выводы

В этом юните мы рассмотрели инструмент автоматизации задач Jenkins.

# Итоги/выводы по модулю

В этом модуле были рассмотрены технологии непрерывной интеграции, доставки и тестирования.

# Практическое задание

проверяет достижение ОР всего модуля.

При формулировании задания придерживайся правил:

* Цель задания: что оно даст студенту и зачем его делать
* Содержание задания: что нужно сделать
* Критерии: на что опираться при выполнении, на что будет опираться проверяющий, что является правильно выполненным заданием
* Рекомендации/подсказки

1. git
2. jenkins
3. dvc

# Список источников

github.com

dvc.org

# Дополнительные материалы

есть что посоветовать студентам для самостоятельного изучения и подогрева интереса - это сюда)

# 