МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет экономики, менеджмента и информационных технологий

Кафедра инноватики и строительной физики имени профессора И.С. Суровцева

Отчет по производственной практике (тип: Проектная)

Выполнила студент группы мТИИ-211 \_\_ Д.А. Москвенков

Подпись, дата Инициалы, фамилия

Принял Д.В. Сысоев

Подпись, дата Инициалы, фамилия

Защищена \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата

Воронеж 2023

СОДЕРЖАНИЕ

[ЗАДАНИЕ 3](#_Toc126619235)

[ВВЕДЕНИЕ 4](#_Toc126619236)

[1.1 Цели CI/CD 5](#_Toc126619237)

1.2 Анализ методологии CI\CD 7

2.1 Введение в Jenkins 8

2.2 Установка Jenkins 9

2.3 Первичная настройка Jenkins 12

[2.3.1 Настройка Jenkins slave node 18](#_Toc126619242)

[2.3.2 Первая сборка (job) в Jenkins 21](#_Toc126619243)

[3.1 Сравнительный анализ Jenkins и альтернативных решений 24](#_Toc126619244)

3.2 Различия между Jenkins и GitLab CI/CD 26

ЗАКЛЮЧЕНИЕ 28

СПИСОК ЛИТЕРАТУРЫ 29

# ЗАДАНИЕ

1. Пройти инструктаж по технике безопасности;
2. Приобрести навыки деятельности в трудовом коллективе;
3. Подготовить анализ методологии непрерывной интеграции ПО;
4. Подготовить инструкцию по развёртыванию выбранного средства автоматизации развёртывания ПО и выполнить тестовую сборку;
5. Изучить альтернативные решения, провести сравнительный анализ одного из них с ранее выбранным.
6. Подготовить и защитить отчет о прохождении практики.

### Введение в CI/CD: что это такое и зачем нужно

Скорость разработки продуктов — одно из главных конкурентных преимуществ в разработке ПО. Поэтому на смену старым моделям программирования пришла новая концепция CI/CD.

**CI (Continuous Integration)** — непрерывная интеграция. Разработчики, применяющие данный паттерн, могут проверять основную ветку репозитория каждый раз, когда что-то замержили в неё. Не просто запускать локальные проверки, а в рамках CI-пайплайна выполнять автоматические тесты, unit-тесты и др.

**CD, (Continuous Delivery)** — непрерывная поставка. На этой стадии происходит автоматическое развертывание на стенды и тестовые окружения. Ещё CD расшифровывают как **Continuous Deployment**— непрерывное развёртывание. Это более продвинутый путь, на шаг дольше, чем непрерывная поставка. При таком подходе каждое изменение, которое мы коммитим в основную ветку репозитория, автоматически проходит все этапы CI и CD и затем попадает на продакшн.

Continuous Deployment Pipeline — высший пилотаж, который редко встречается на практике, потому что всегда есть определённые ограничения. Эти ограничения могут быть как в самом пайплайне, так и в бизнес-процессах с точки зрения безопасности. Но, однозначно, Continuous Deployment Pipeline — то, к чему нужно стремиться.

**1.1 Цели CI/CD**:

* обеспечение последовательного и автоматизированного способа сборки, упаковки и тестирования;
* автоматизация развёртывания в разных окружениях;
* сведение к минимуму ошибок и проблем.

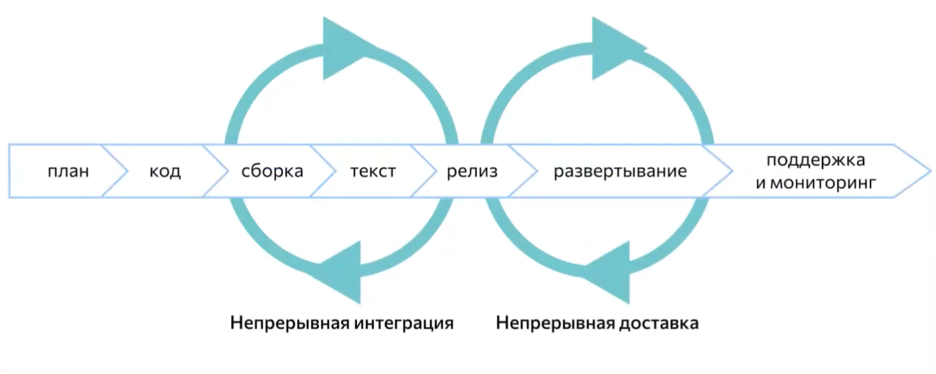
Добиться этих целей помогают четыре принципа, на которых основана концепция CI/CD. Первый принцип — **разделение активности**. Каждый из участников процесса делит ответственность за жизненные циклы продукта. Проектируется бизнес-логика, выбираются сквозные функции, проводятся тесты, организуется доставка кода из одного окружения в другое.

Второй принцип — **снижение рисков**. Чтобы баги не доходили до продакшена, контролируется корректность бизнес-логики, проверяется пользовательский опыт на стендах, улучшается процесс хранения и обработки данных. Чем раньше мы обнаружим риск, тем быстрее идентифицируем проблему и тем меньше средств потратим на её решение.

Третий принцип — **сокращение цикла обратной связи**. В рамках CI/CD мы стремимся увеличить скорость внесения изменений и согласования правок.

Четвертый принцип — **реализация среды**. У разработчиков должно быть общее пространство для работы с основной веткой или со вспомогательными ветками. Это пространство должно быть отказоустойчивым и удобным для работы.

Основные этапы CI/CD выглядят так:



Планирование основывается на пользовательском опыте и бизнес-функционале. Обычно за этот этап отвечают люди из анализа: они переводят требования с языка бизнеса на язык, понятный разработчикам и администраторам. Затем начинается этап работы с кодом — разработчики пишут код, проводят тестирования в ручном режиме и добавляют изменения в основную ветку репозитория.

После того, как изменения попадают в репозиторий, система контроля версии инициирует сборку и тестирование проекта. Тестирование может быть как ручным, так и автоматическим — зависит от того, как работает команда. Далее всё уходит сначала на релиз, а затем на развёртывание. На этапе развёртывания уже протестированная версия приложения отправляется на продакшн и становится доступна пользователям.

Когда продукт попадает к пользователям, мы продолжаем следить за ним — этап поддержки и мониторинга. Мы контролируем, как пользователь идёт по бизнес-процессу, корректно ли работают интеграции. Если на стадии мониторинга мы обнаруживаем ошибку, возвращаемся к самому началу — к планированию. Аналитики разбирают, что пошло не так и предлагают новое решение, разработчики пишут код, и снова начинается процесс сборки.

1.2 Анализ методологии CI\CD

Как и у любой методологии, у CI/CD есть свои плюсы и минусы:

|  |  |
| --- | --- |
| **Плюсы** | **Минусы** |
| Минимальное время от запроса клиента до запуска в использование — мы быстрее доставляем новые фичи | Сложность обеспечения взаимодействия — и DevOps-инженеры, и разработчики должны понимать, что было сделано и зачем |
| Возможность проверки вариантов — можем моментально проверять изменения и при необходимости откатывать назад | Требования к опыту — нужен опыт настройки CI/CD, который почти всегда добывается с болью |
| Качество результата — можем быстро обнаружить и пофиксить ошибки |  |

Реализовать принципы CI/CD, свести к минимуму ошибки интеграции, а также ускорить релизы и повысить их качество помогает Jenkins.

### 2.1 Введение в  Jenkins

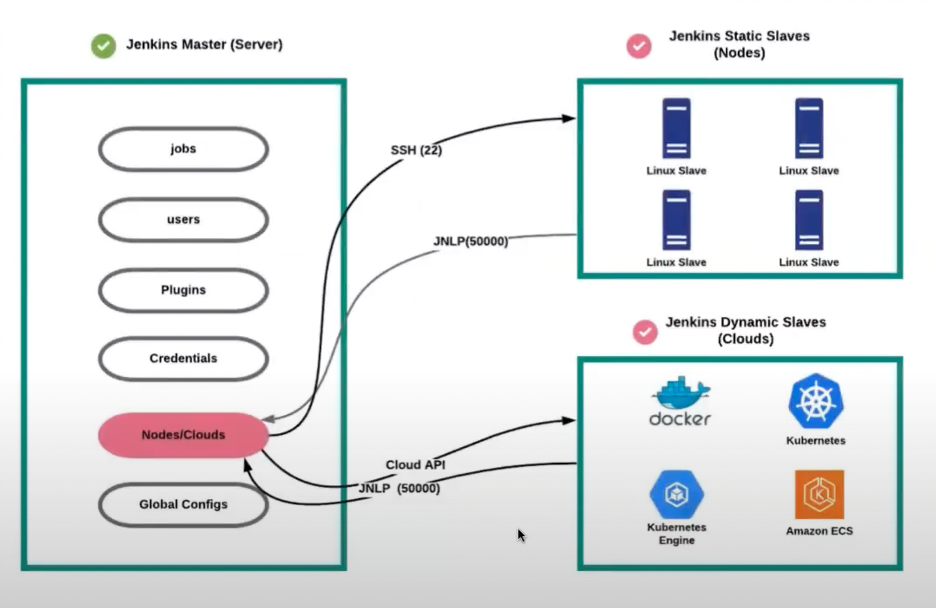
Jenkins – не просто инструмент CI/CD. Это Framework, потому что он:

* **Гибок и расширяем**. Jenkins — опенсорсный проект с множеством внешних расширений.
* **Минимален из коробки**. У Jenkins есть контроллер. Вы можете подключить к нему несколько слоев и уже на этом сетапе собрать минимальный пайплайн, который позволит автоматизировать работу по обновлению сервисов.
* **Требует настройки**. Jenkins — один из кубиков, с помощью которого можно построить большую систему автоматизации. Но прежде чем сделать что-то, его придётся настроить.

Jenkins — это Java-приложение. У него есть контроллер или Master Mode — управляющий центр, который занимается планированием задач. Он запускает задачи согласно установленному расписанию на слэйвах, которые вы к нему прикрепили. Помимо этого контроллер хранит логи наших задач. Вся история хранится только на Master Mode, поэтому важно помнить о настройке правильной ротации логов.

Слэйвы или агенты — это то, что непосредственно выполняет сами задания.

Коротко их взаимодействие можно описать так: контроллер запускает задачу и говорит агенту выполнить её, агент выполняет задачу и возвращает результат контроллеру. Контроллер получает результат и сохраняет его в build-логе.



### 2.2 Установка Jenkins

Разберём, как установить Jenkins, как настроить параметры JVM и почему это важно. Дополнительно познакомимся с Jenkins Home: что это за зверь и с чем его едят. Все действия будем выполнять на Centos.

Перед установкой Jenkins нужно установить Java — без этого никак. Мы проверяем, есть ли на нашем виртуальном энвайронменте Java:

java --version

По умолчанию из коробки Java нет:

Рекомендуется использовать Java 11, потому что у неё более продвинутый Garbage Collector. Поставим её И дополнительные пакеты, требующиеся для работы:

yum install -y wget git openjdk-11-jre-headless fontconfig

Проверим, что Java установилась:

java --version

Теперь нужно сконфигурировать файл limits.conf. В Linux всё — файл, поэтому нужно установить файл-лимит, чтобы снять ограничения и позволить Jenkins генерировать файлы, дампы и др. Если не сделать этого, в каких-то случаях мы не сможем получить данные и понять, что же с Jenkins пошло не так.

Редактируем лимиты в ОС:

vi /etс/security/limits.conf

Приводим к виду:

jenkins soft core unlimited

jenkins hard core unlimited

jenkins soft fsize unlimited

jenkins hard fsize unlimited

jenkins soft nofile 4096

jenkins hard nofile 8192

jenkins soft nproc 30654

jenkins hard nproc 30654

Следующим шагом будет базовая настройка безопасности, для этого сконфигурируем правила firewall, открыв 22 порт для SSH и порт 8080 для веб-сервера Jenkins:

systemctl start firewalld

firewall-cmd --permanent --add-port=22/tcp

firewall-cmd --permanent --add-port=8080/tcp

firewall-cmd --reload

firewall-cmd --list-all

Добавляем репозиторий Jenkins:

wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo

rpm --import <https://pkg.jenkins.io/redhat-stable/jenkins.io.key>

и выполняем установку:

yum -y install jenkins

после завершения, добавляем в /etc/sysconfig/jenkins следующие параметры запуска:

JENKINS\_JAVA\_OPTIONS="-Djava.awt.headless=true -Djava.io.tmpdir=/var/cache/jenkins/tmp -Dorg.apache.commons.jelly.tags.fmt.timeZone=Europe/Moscow -Duser.timezone= Europe/Moscow "

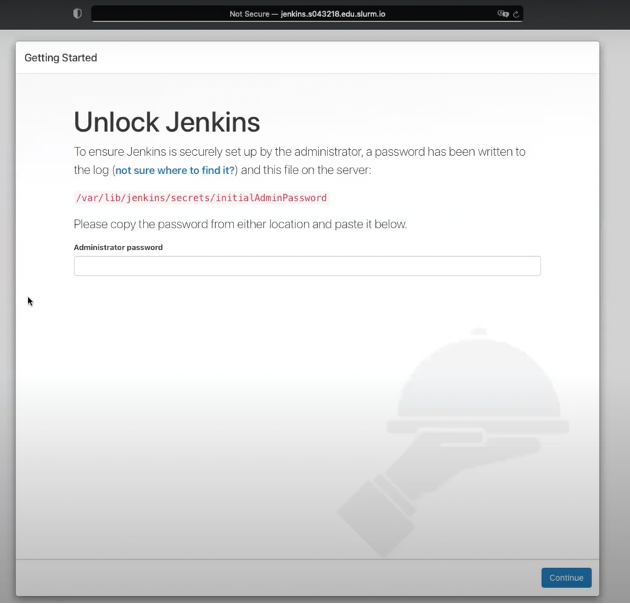
JENKINS\_ARGS="--pluginroot=/var/cache/jenkins/plugins"

Далее активируем system unit, запускаем и проверяем статус:

systemctl enable jenkins && systemctl status Jenkins && systemctl status jenkins

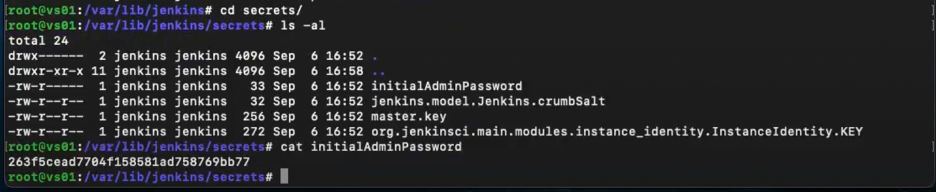
2.3 Первичная настройка Jenkins

Теперь откроем веб-интерфейс <http://localhost:8080>. Он предлагает разблокировать Jenkins:

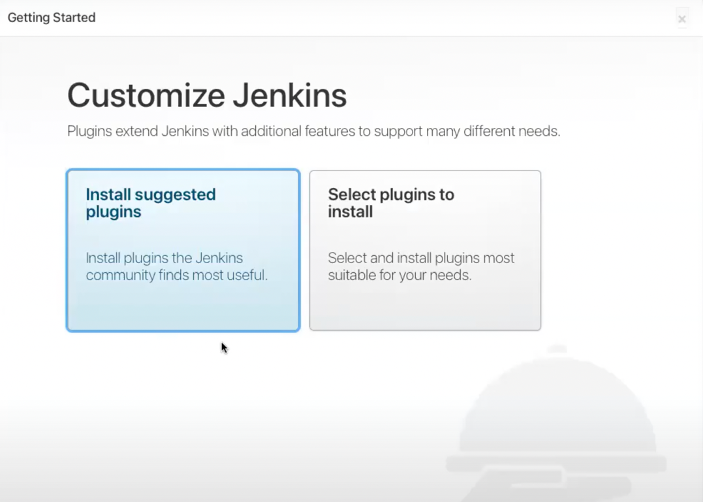


То есть первоначальная установка предполагает, что вы введете некий мастер-пароль. Jenkins сам подсказывает, где этот мастер-пароль можно найти — var/lib/jenkins/secret/initialAdminPassword. Давайте пойдём туда и посмотрим:

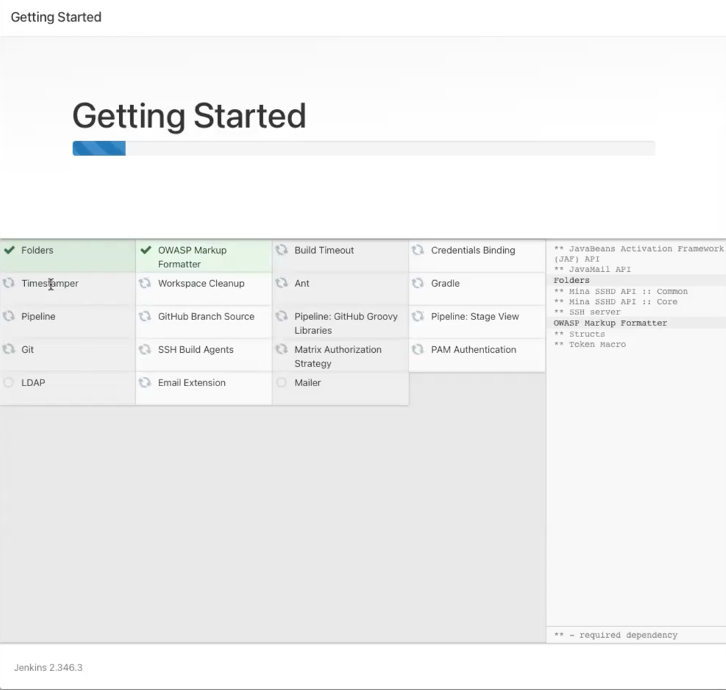
ls -la /var/lib/jenkins/secrets



Следующий шаг — установить Suggested-плагины или самостоятельно выбрать плагины, которые хотите поставить.



Список плагинов, которые используются в Suggested, достаточно правильный, поэтому рекомендуется просто поставить эти плагины автоматом.

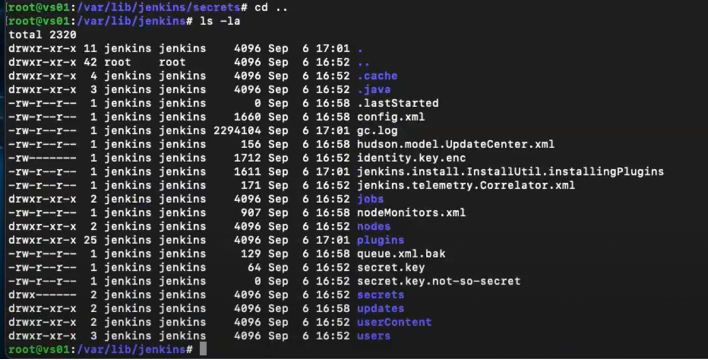


А пока Jenkins делает плагины, посмотрим на Home-директорию:

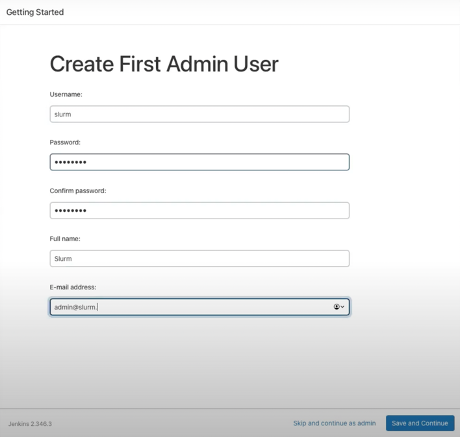
root@vs01:/var/lib/jenkins/secrets# cd ..

root@vs01:/var/lib/jenkins# ls -la

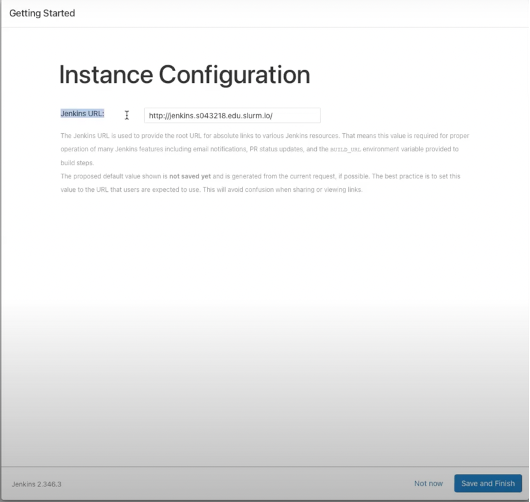
У Jenkins нет выделенной базы, как у некоторых систем. В качестве базы данных он использует директорию Jenkins Home, которая есть в каталоге файловой системы того сервера, на который выставили контролер. Здесь хранятся: конфиги, плагины, задания, которые мы делаем и всё, что связано с ними и т.д.



Вернёмся к Jenkins — он уже поставил домен и предлагает нам настроить первого пользователя. Заполним поля:



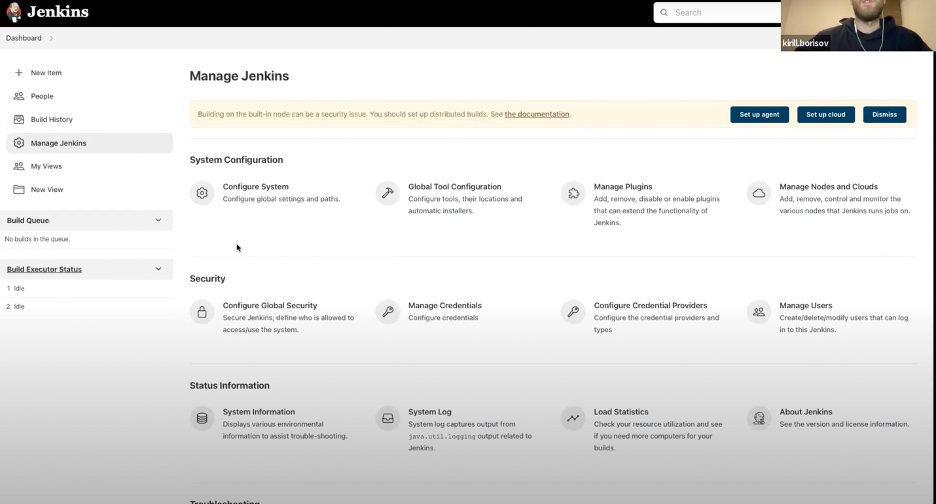
Далее Jenkins предлагает настроить URL:



Этот параметр достаточно важный, потому что его параметр будут использовать различные слэйвы для автоматического коннекта к мастеру.



Итак, мы поставили Jenkins. Пока нет никаких заданий, но мы можем немного пройтись по Manage Jenkins:



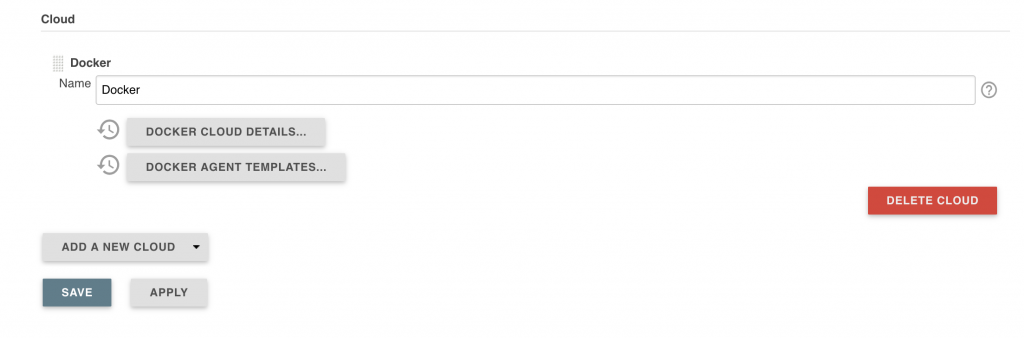
Из наиболее интересного здесь:

* System information — хранит всю основную информацию (Java Home, версия Java, версия Ubuntu и т.д.).
* System log — то, куда нужно смотреть, если непонятно, почему не подключается агент или не работает скрип.
* Load statistics — показывает, сколько экзекьюторов всего онлайн.

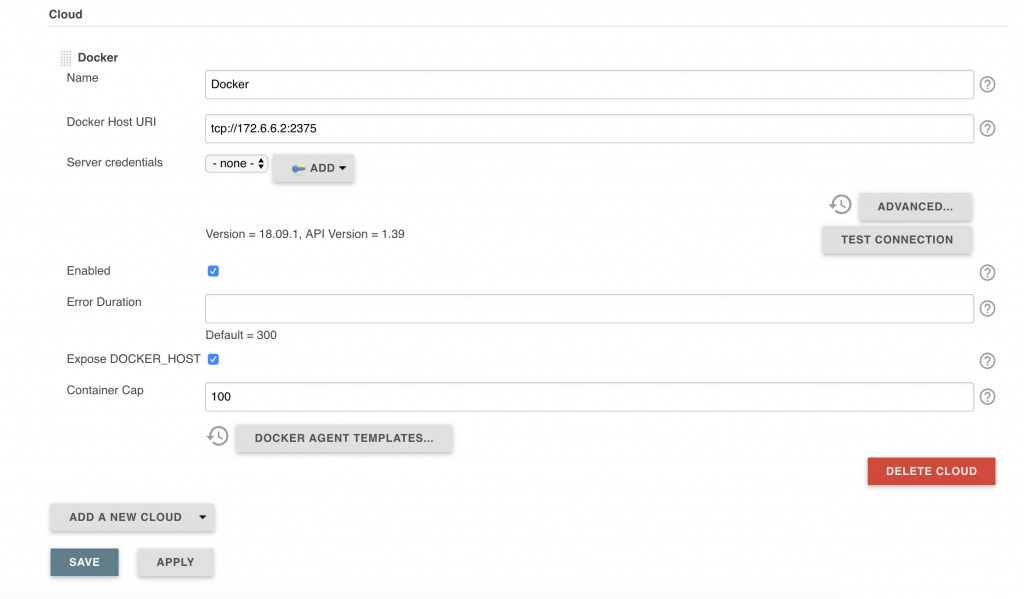
2.3.1 Настройка Jenkins slave node

Открываем дженкинс и переходим:

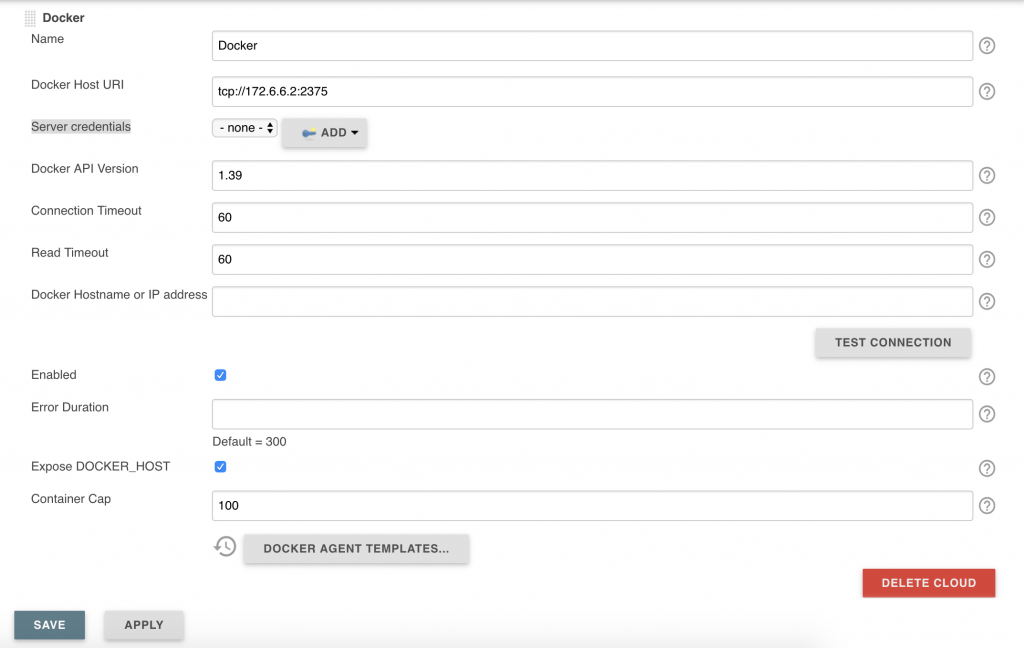
«Manage Jenkins» -> «Configure System» и ищем поле «Cloud»:



Кликаем по «Docker Cloud Details» чтобы ввести необходимые данные:

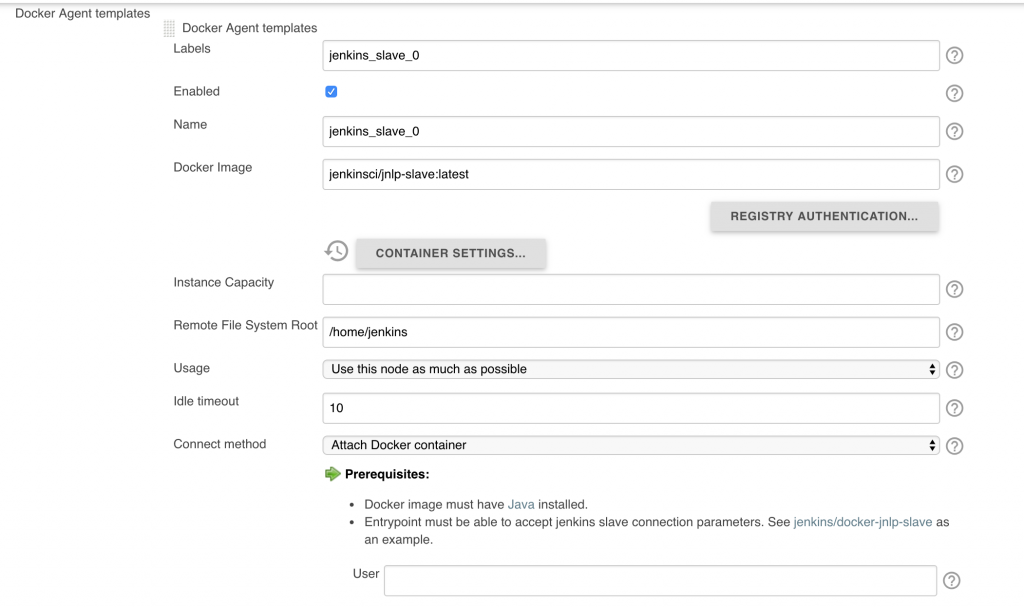


Задаем имя, в нашем примере это — Docker. В поле «Docker Host URI» прописываем хост и порт который использует докер. У нас — «tcp://172.6.6.2:2375». Стоит отметить, если используется авторизация к докер хосту, то стоит заполнить «Server credentials». Если нажать на «Advanced», то выпадет список дополнительных параметров которые можно также заполнить:

* 

Нажмите на «Test connection» чтобы получить тестовое подключение (чтобы убедится что подключение работает).

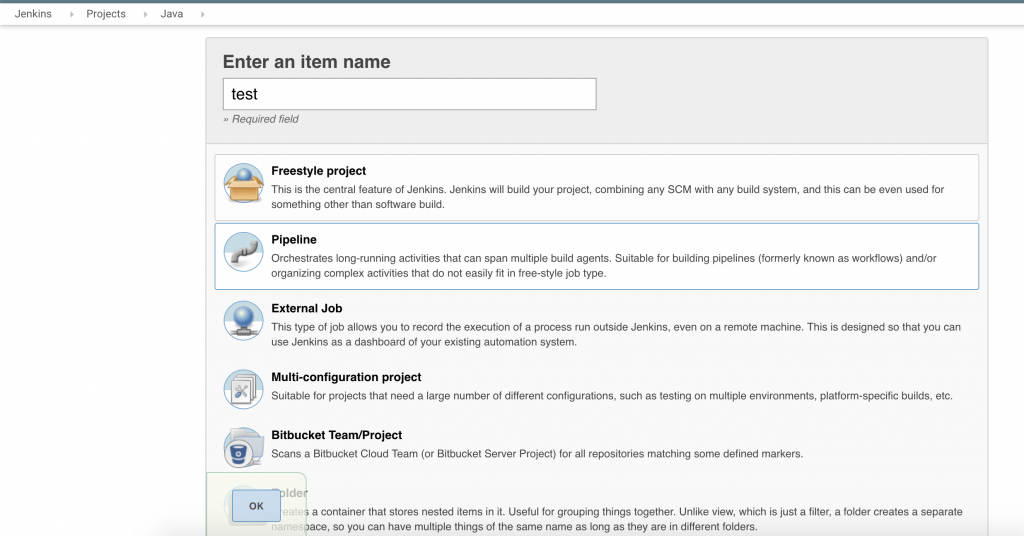
Идем дальше, выбираем «DOCKER AGENT TEMPLATES…» Я привел к виду:



Заполнил поле и лейбл как мне угодно. В поле «Docker image» я прописал Jenkins-Slave образ, который я взял с официального докер-регистра — «jenkinsci/slave». Так же, по необходимости заполните все необходимые поля (креденшелы, дополнительные опции).

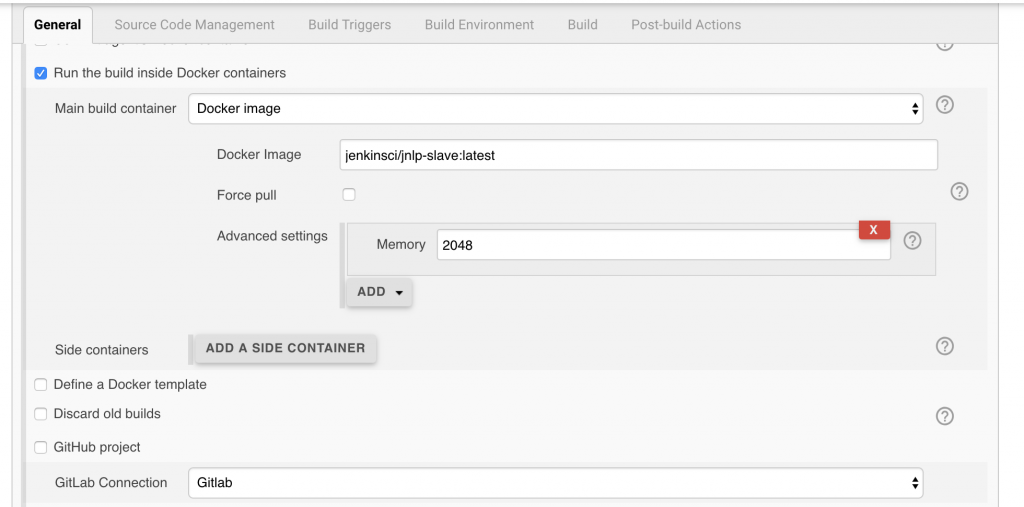
2.3.2 Первая сборка (job) в Jenkins

Создаем проект под свои нужды. Потом, создаем «Pipeline» проект, например:

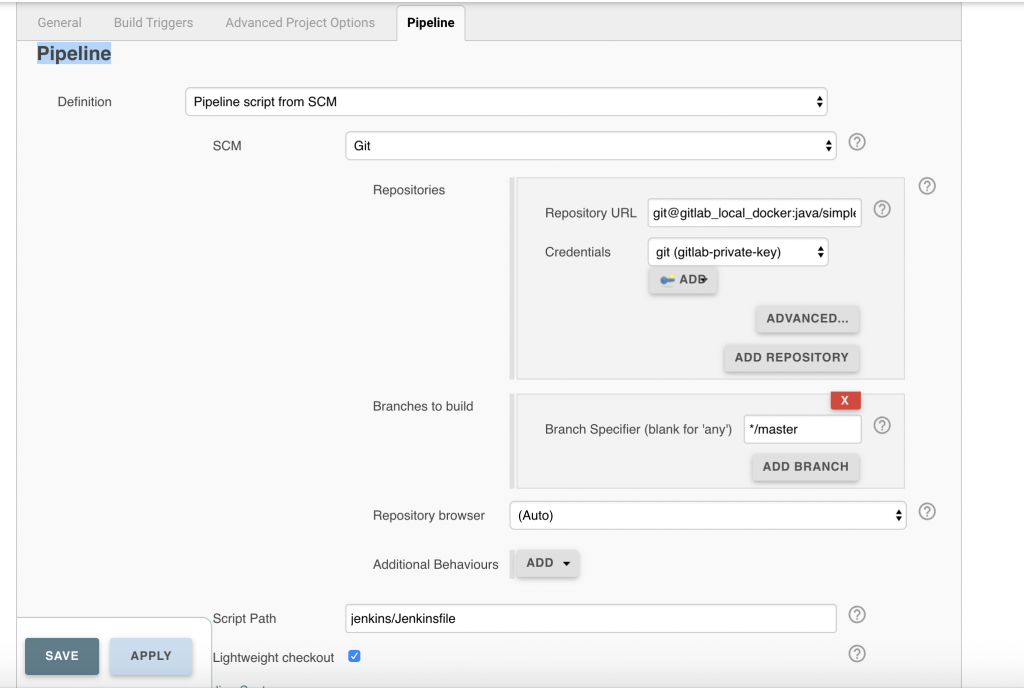
Тестовый pipeline projec для Java

Нажимаем на «OK» и сейчас создадим все необходимое.

Находим «Run the build inside Docker containers» и ставим чекбокс. В поле «Docker Image» ставим наш образ, у меня — «jenkinsci/jnlp-slave:latest». Так же, можно прописать «Advanced settings» опции и выставить использовании по памятиВ нашем случае, для примера, настройки имеют вид:



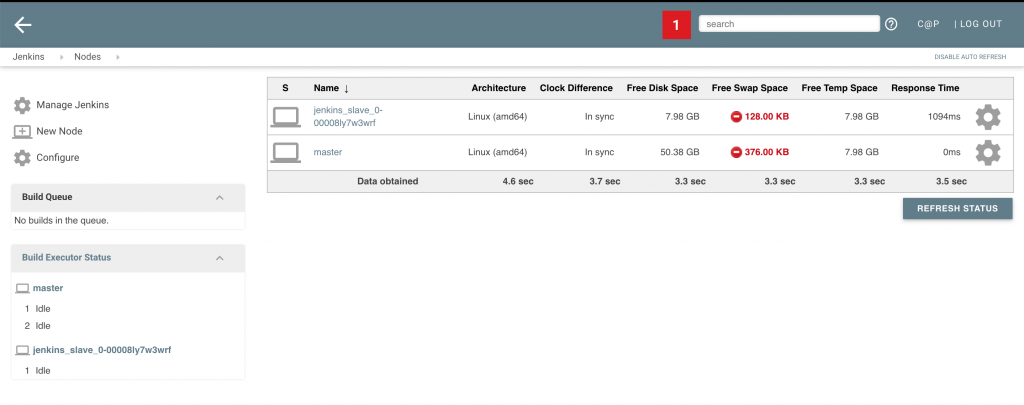
Идем дальше, находим «Pipeline» вкладку и заполняем ее под свои нужды. У меня все приведено и имеет вид:



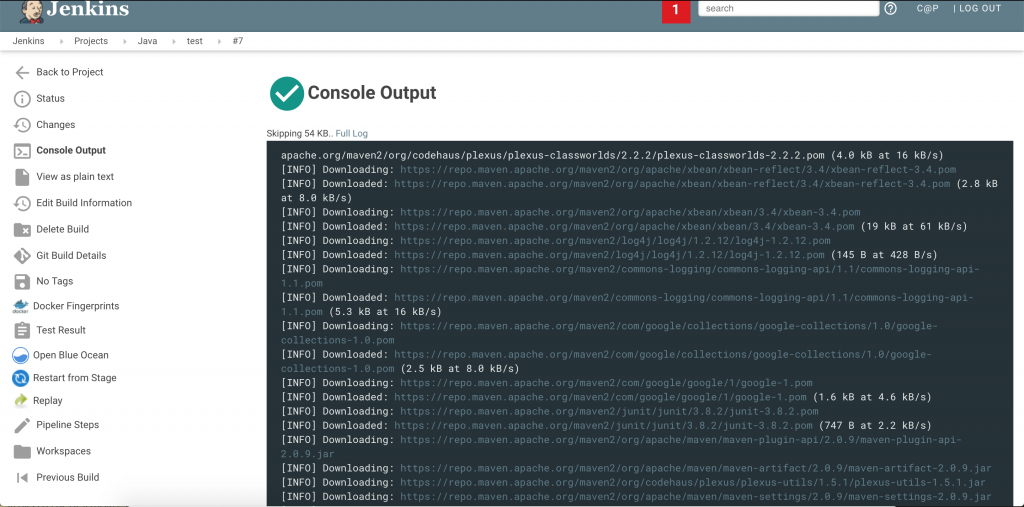
Мы используем свой гитлаб сервер, развёрнутый локально на одной машине с Jenkins. В нем есть репозиторий с проектом. Так же, добавил подключение к гитлабу. Собственно, все готово, можно нажимать на «SAVE»!

Слева вверху, нажимаем на «Build Now» и смотрим что получилось!

Если открыть «Manage Jenkins» -> «Manage Nodes», то появится jenkins-slave:



Видно что поднялся слейв и запустил джобу. Можно открыть ее и увидеть статус выполнения:



3. Сравнительный анализ Jenkins и альтернативных решений.

В качестве аналога Jenkins предлагаю рассмотреть очень популярное решение - GitLab

Jenkins и GitLab CI/CD — это очень хорошие инструменты, каждый из которых способен обеспечить нормальную работу CI/CD-конвейера. Но, если их сравнить, окажется, что они, хотя и во многом похожи, кое-чем друг от друга отличаются.

|  |  |  |
| --- | --- | --- |
| **Характеристика** | **Jenkins** | **GitLab CI/CD** |
| Открытый или закрытый код | Открытый код | Открытый код |
| Установка | Требуется. | Не требуется, так как это — встроенная возможность платформы GitLab. |
| Уникальные особенности | Поддержка плагинов. | Глубокая интеграция в систему управления версиями. |
| Поддержка | Отсутствует. | Имеется. |
| Установка и настройка | Сложностей не вызывают | Сложностей не вызывают |
| Самостоятельное развёртывание системы | Это — единственный вариант использования системы. | Поддерживается. |
| Создание CI/CD-конвейеров | Поддерживается, используется Jenkins Pipeline. | Поддерживается. |
| Мониторинг производительности приложений | Отсутствует. | Имеется. |
| Экосистема | Существует более 1000 плагинов. | Система развивается в рамках GitLab. |
| API | Поддерживает развитую систему API. | Предлагает API для более глубокой интеграции в проекты. |
| Поддержка JavaScript | Имеется. | Имеется. |
| Интеграция с другими инструментами | Поддерживается интеграция с другими инструментами и платформами (Slack, GitHub). | Множество средств для интеграции со сторонними системами, в частности — с GitHub и Kubernetes. |
| Контроль качества кода | Поддерживается — с помощью плагина SonarQube и других плагинов. | Поддерживается. |

3.2 Различия между Jenkins и GitLab CI/CD

Описав и сравнив Jenkins и GitLab CI/CD, давайте сосредоточимся на различиях этих DevOps-инструментов. Знание об этих различиях позволит понять тех, кто предпочитает один из этих инструментов другому.

* GitLab CI/CD может полностью контролировать Git-репозитории. Речь идёт об управлении ветками репозиториев и о некоторых других возможностях. А вот Jenkins, хотя и умеет работать с репозиториями, не даёт такого же уровня контроля над ними, как GitLab CI/CD.
* Jenkins — это бесплатный опенсорсный проект. Тот, кто его выбирает, разворачивает его самостоятельно. А GitLab CI/CD включён в состав платформы GitLab, это готовое решение.
* GitLab CI/CD поддерживает развитые средства управления задачами, работающие на уровне проектов. Эта сторона Jenkins развита слабее.

Jenkins и GitLab CI/CD: сильные и слабые стороны

Сейчас у вас сложилось некоторое представление о Jenkins и GitLab CI/CD. Теперь, чтобы вы ещё лучше познакомились с этими инструментами, давайте разберём их сильные и слабые стороны. Полагаем, что вы уже приняли решение о том, какой именно инструмент вам нужен. Хочется надеяться, этот раздел позволит вам проверить себя.

- Сильные стороны Jenkins

* Большое количество плагинов.
* Полный контроль над установкой инструмента.
* Простая отладка раннеров.
* Простая настройка узлов.
* Простое развёртывание кода.
* Очень хорошая система управления учётными данными.
* Гибкость и универсальность.
* Поддержка различных языков программирования.
* Система понятна на интуитивном уровне.

- Слабые стороны Jenkins

* При использовании плагинов могут возникать сложности.
* При использовании Jenkins в маленьких проектах затраты времени, необходимые на его самостоятельную настройку, могут оказаться неоправданно большими.
* Отсутствие общих аналитических сведений по CI/CD-цепочкам.

- Сильные стороны GitLab CI/CD

* Хорошая интеграция с Docker.
* Простое масштабирование раннеров.
* Параллельное выполнение задач, входящих в состав стадий CI/CD-конвейера.
* Использование модели ориентированного ациклического графа при настройке взаимоотношений задач.
* Высокий уровень масштабируемости за счёт возможности параллельного выполнения раннеров.
* Лёгкость добавления задач.
* Простое разрешение конфликтов.
* Надёжная система безопасности.

- Слабые стороны GitLab CI/CD

* Для каждой задачи нужно описывать и загружать/выгружать артефакты.
* Нельзя протестировать результаты объединения веток до их фактического объединения.
* При описании стадий CI/CD-конвейера в них пока нельзя выделять отдельные этапы.

Заключение

И Jenkins, и GitLab CI/CD имеют сильные и слабые стороны. Ответ на вопрос о том, что именно выбрать, зависит от нужд и особенностей конкретного проекта. Каждый из рассмотренных сегодня CI/CD-инструментов отличается определёнными особенностями, хотя созданы эти инструменты для решения одной и той же задачи. При этом Jenkins — это автономный инструмент, а GitLab CI/CD — это часть платформы, предназначенной для совместной работы над кодом.  
  
Выбирая CI/CD-систему стоит, помимо её возможностей, принимать во внимание и те затраты, которые могут быть с ней связаны, и то, с чем именно привыкли работать DevOps-инженеры, поддерживающие проект.

Среди всех существующих CI/CD инструментов существуют два проекта, на которые, определённо, стоит обратить внимание тому, кто ищет что-то из этой сферы. Речь идёт о Jenkins и об инструменте GitLab CI/CD, который является частью платформы GitLab. У Jenkins имеется более [16000](https://github.com/jenkinsci/jenkins) звёзд на GitHub. Репозиторий GitLab на gitlab.com набрал чуть больше 2000 звёзд. Если сравнить популярность репозиториев, то окажется, что Jenkins набрал в 8 раз больше звёзд, чем платформа, в состав которой входит GitLab CI/CD. Но при выборе CI/CD-инструмента это — далеко не единственный показатель, на который стоит обращать внимание. Есть и масса других, и это объясняет то, что во многих сравнениях Jenkins и GitLab CI/CD оказываются очень близко друг к другу.

Список используемой литературы.

* 1. Официальная документация Jenkins: <https://www.jenkins.io/doc/book/>
  2. Официальная документация GitLab

<https://docs.gitlab.com/ee/ci/>

* 1. John Ferguson Smart “Jenkins: The Definitive Guide”, O'Reilly Media, Inc, 2011
  2. Adam O'Grady “GitLab Quick Start Guide”, Packt Publishing, 2018
  3. Брент Ластер: “Jenkins 2. Приступаем к работе. Создайте свой конвейер развертывания для автоматизации следующего проекта.“ ДМК-Пресс, 2019 г.