# Модуль 4. Непрерывная интеграция и развертывание (CI/CD).

## В этом модуле

В юнитах 1-4 этого модуля вы создали инфраструктуру для проекта, в том числе добавили к уже созданной и настроенной ранее инфраструктуре серверов data-srv (хранение данных, моделей, конвейера проекта с помощью dvc), ml-srv (исследование данных, проверка гипотез, обучение моделей ML) и prod-srv (эксплуатация проекта) виртуальные машины admin-srv для работы администратора инфраструктуры и stage-srv для проведения тестов перед выводом системы в эксплуатацию. Также, дополнительно к уже установленному в предыдущих модулях программному обеспечению, в юнитах 1-4 этого модуля вы установили Jenkins для управления конвейерами операций, pytest для автоматизации тестирования и ansible для автоматизации настроек серверов.

Перечисленные инструменты мы используем далее в юнитах 5-8 для решения практической задачи. В реализуемом сценарии разработчик будет публиковать код в git и dvc репозиторий, который опрашивается Jenkins с определенной периодичностью. Если Jenkins при опросе получит уведомление о наличии обновлений в опубликованном коде в репозитории git, который в том числе может означать изменения в репозитории dvc, то будет автоматически запущено тестирование обновленной модели ML на сервере stage-srv с помощью pytest. При успешном завершении тестирования Jenkins запускает настройку сервера эксплуатации prod-srv с помощью ansible, устанавливает на него обновленное и протестированное программное обеспечение и выполняет запуск web сервиса.

## Содержание юнитов

Последовательность тем, изучаемых в отдельных юнитах модуля:

|  |
| --- |
| 1. Общая постановка задачи, создание инфраструктуры и окружения |
| 1. Установка и настройка Jenkins. |
| 1. Установка и настройка ansible. |
| 1. Установка и настройка pytest. |
| 1. Подробная постановка задачи |
| 1. Настройка конвейера операций в Jenkins, автоматизация взаимодействия с git |
| 1. Настройка сервера тестирования и автоматизированное тестирование с pytest |
| 1. Настройка сервера эксплуатации с ansible и автоматизированное развертывание с Jenkins |
| 1. Задание для самостоятельной работы |

# Модуль 4. Юнит 5. Подробная постановка задачи.

*Введение*: В этом юните подробно описана решаемая в модуле задача и даны рекомендации для ее решения, в том числе описаны инструменты для выполнения задачи модуля.

*Содержание:*

В этом модуле разбирается практический пример создания инфраструктуры для простого проекта машинного обучения: мы продолжаем работать с задачей “Titanic Disaster” с площадки соревнований по машинному обучению Kaggle.com. Несмотря на то, что рассматриваемый проект имеет очень простую структуру и решает учебную, а не производственную, задачу, решаемые при прохождения модуля задачи часто присутствуют и в реальных производственных проектах:

* cобирать, обрабатывать и исследовать данные,
* проводить эксперименты и исследовать гипотезы,
* управлять аппаратным обеспечением для обучения модели,
* контролировать перечень и зависимости используемых библиотек,
* управлять процессом обучения модели, контролировать переобучение,
* обеспечивать версионирование разработанных программ и результатов их работы,
* выводить модель в эксплуатацию и осуществлять мониторинг качества работы.

В юнитах 2-4 вы выполнили создание и настройку простой, но полнофункциональной инфраструктуры для проекта, которая включает основные сегменты и инструменты для всего конвейера проекта от сбора и обработки данных до вывода модели в промышленное окружение и мониторинга эксплуатации.

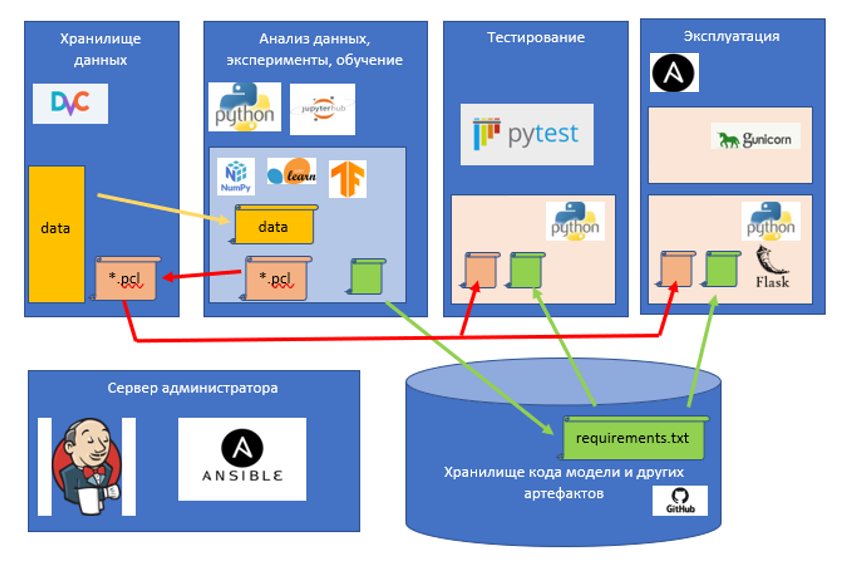
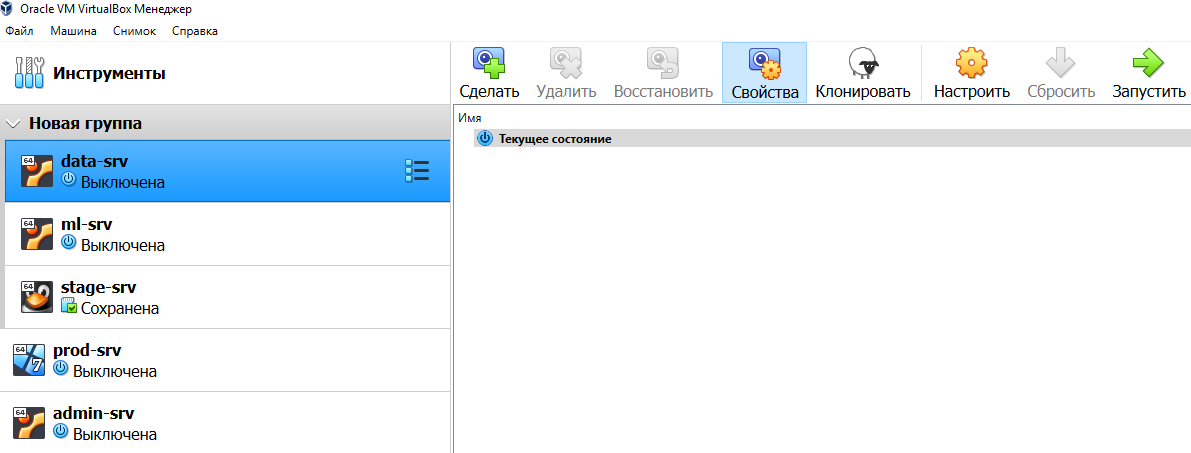


Рисунок «Инфраструктура проекта»

При этом были использованы результаты, полученные при работе с утилитой dvc в Модуле 1, в котором вы осуществили создание и настройку серверов data-srv, ml-srv и prod-srv, а также добавили необходимых пользователей в этих системах. В юнитах 2-4 Модуля 4 вы также создали виртуальные машины admin-srv и stage-srv. В итоге, в гипервизоре virtualbox были созданы следующие виртуальные машины для инфраструктуры проекта:

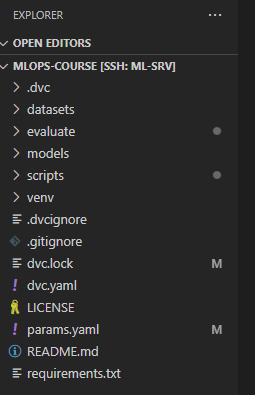


Назначение созданных виртуальных машин следующее:

* data-srv: локальный сервер с доступом по ssh для организации dvc хранилища, используемое программное обеспечение: openssh-server
* ml-srv: основной рабочий сервер для исследования данных, проведения экспериментов, обучения моделей, используемое программное обеспечение: jupyter, python библиотеки для анализа данных и машинного обучения, venv,
* stage-srv: сервер для проведения автоматизированного тестирования, используемое программное обеспечение: pytest, python
* prod-srv: сервер эксплуатации модели,
* admin-srv: сервер для работы администратора, DevOps/MLOps инженера, используемое программное обеспечение: Jenkins, ansible.

В решаемой в модуле задаче необходимо реализовать предсказание выживания (целевая переменная Survived в исходном датасете) пассажира корабля «Титаник» на основе данных о возрасте пассажира (Age), классе каюты (Pclass) и поле (Sex). Для этого необходимо обработать входной датасет, преобразовать признаки, построить и обучить модель классификации. Созданная нами инфраструктура, включая рабочие файлы, скрипты, описанные конвейеры, преобразованные датасеты, модели и метрики, сохраняется с использованием git и dvc. Для этого в git репозитории <https://github.com/CyberLympha/MLOps-course.git> были сохранены файлы скриптов и мета-файлы dvc, описывающие хранение информации в dvc. В созданном конвейере проекта сначала происходит обработка признаков, затем обучение модели и сохранение всех артефактов в dvc хранилище. Модель машинного обучения, решающая задачу, сохраняется в формате pickle в dvc хранилище.

Вид рабочего директория проекта, открытого в VS Code:



В следующих юнитах мы выполним следующие действия, завершающие конвейер проекта машинного обучения:

* В юните 6 вы настроите взаимодействие Jenkins и git, с помощью которого можно увидеть изменения в репозитории и клонировать обновления,
* В юните 7 вы автоматизируете тестирование модели машинного обучения на сервере stage-srv с использованием pytest под управлением Jenkins
* В юните 8 вы автоматизируете вывод модели на сервер эксплуатации, на котором работает приложение на flask и куда подгружается обученная и протестированная модель машинного обучения в формате pickle.

## Тест

1. Какое программное обеспечение будем использовать для автоматизации в проекте в этом модуле? (0.25)
   1. Airflow
   2. MLFlow
   3. ClearML
   4. **Jenkins**
2. Какие утилиты понадобятся для нормального функционирования ansible? (0.25)
   1. **python3**
   2. tensorflow
   3. docker
   4. **ssh**
3. Какой формат используется для сохранения обученной модели машинного обучения для ее последующего тестирования и вывода в эксплуатацию? (0.25)
   1. jpeg
   2. **pickle**
   3. zip
   4. csv
4. Какой фреймворк используется на сервере эксплуатации prod-srv? (0.25)
   1. Django
   2. Fast API
   3. **Flask**
   4. React.js

## Итоги/выводы

В этом юните подробно описана практическая задача модуля, описаны инструменты для ее решения, описано что будем делать в следующих юнитах модуля.

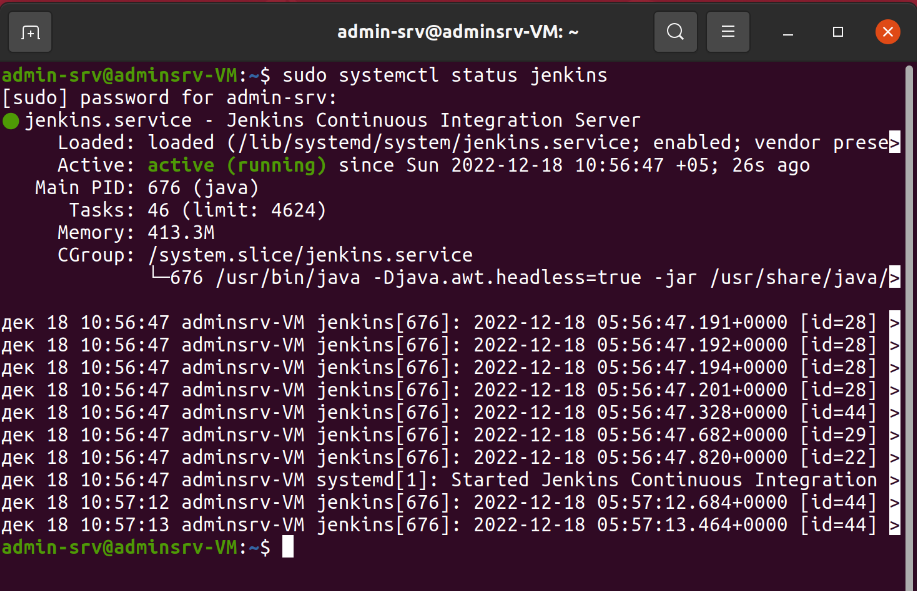
# Модуль 4. Юнит 6. Настройка конвейера операций в Jenkins, автоматизация взаимодействия с git.

## Введение

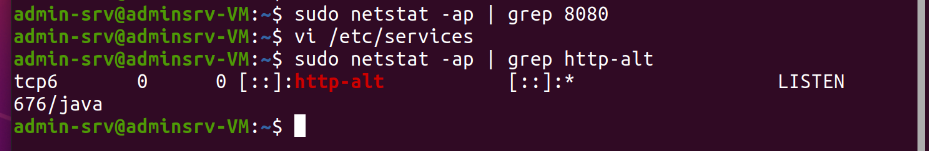
Ранее в этом модуле вы осуществили установку Jenkins, который будет использоваться для автоматизации проекта. В этом юните вы настроите Jenkins для выполнения задания этого модуля. В результате вы настроите управление конвейером операций, а также автоматизацию взаимодействия с git. В следующих юнитах Jenkins будет использован для автоматизации тестирования и вывода в эксплуатацию, что завершит конвейер проекта машинного обучения.

## Содержание

После успешной установки для начала работы с Jenkins необходимо в браузере открыть страницу графического пользовательского интерфейса «http:IP-Address:8080», где IP-Address это адрес виртуальной машины admin-srv, на которой запущено приложение Jenkins, а 8080 это номер порта, которое прослушивает это приложение. Сначала проверьте, что сервис Jenkins запущен на сервере admin-srv

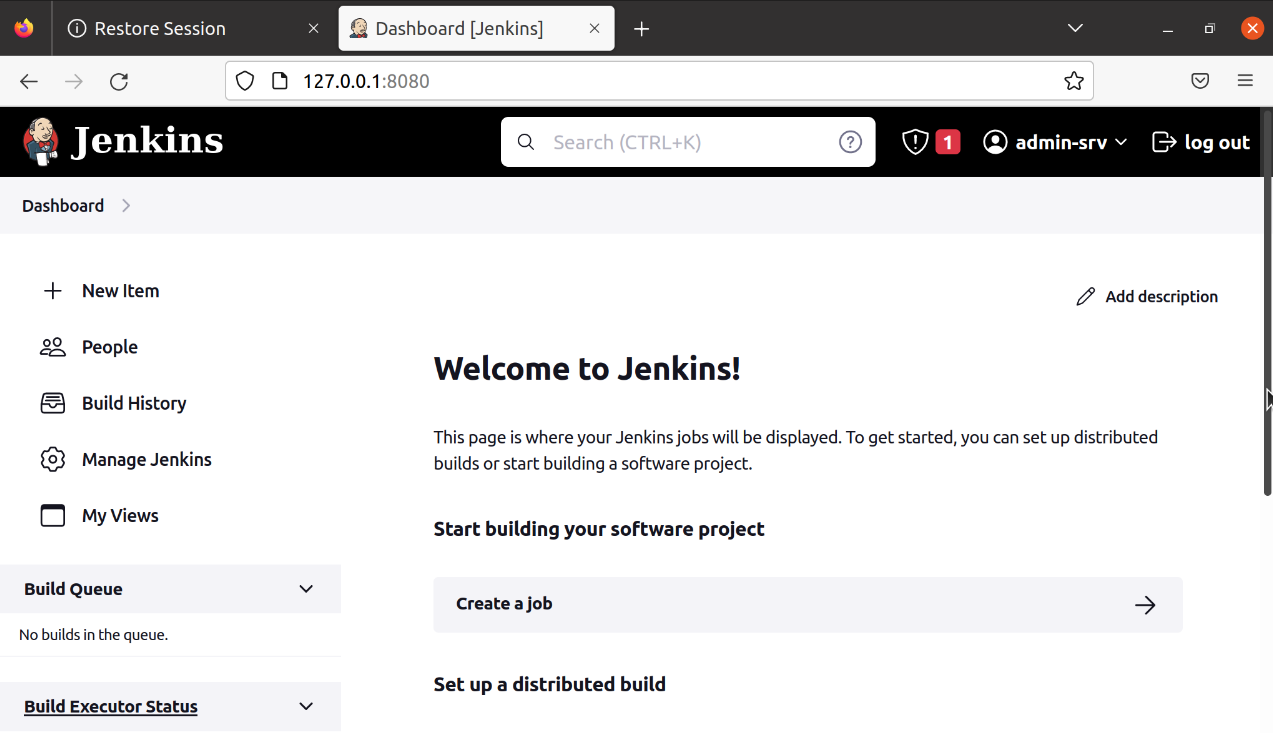


Видим, что сервис находится в состоянии active (running), идентификатор процесса PID=676 (в вашем случае этот идентификатор может иметь другое значение). На всякий случай удостоверимся, что сервис работает на порте 8080

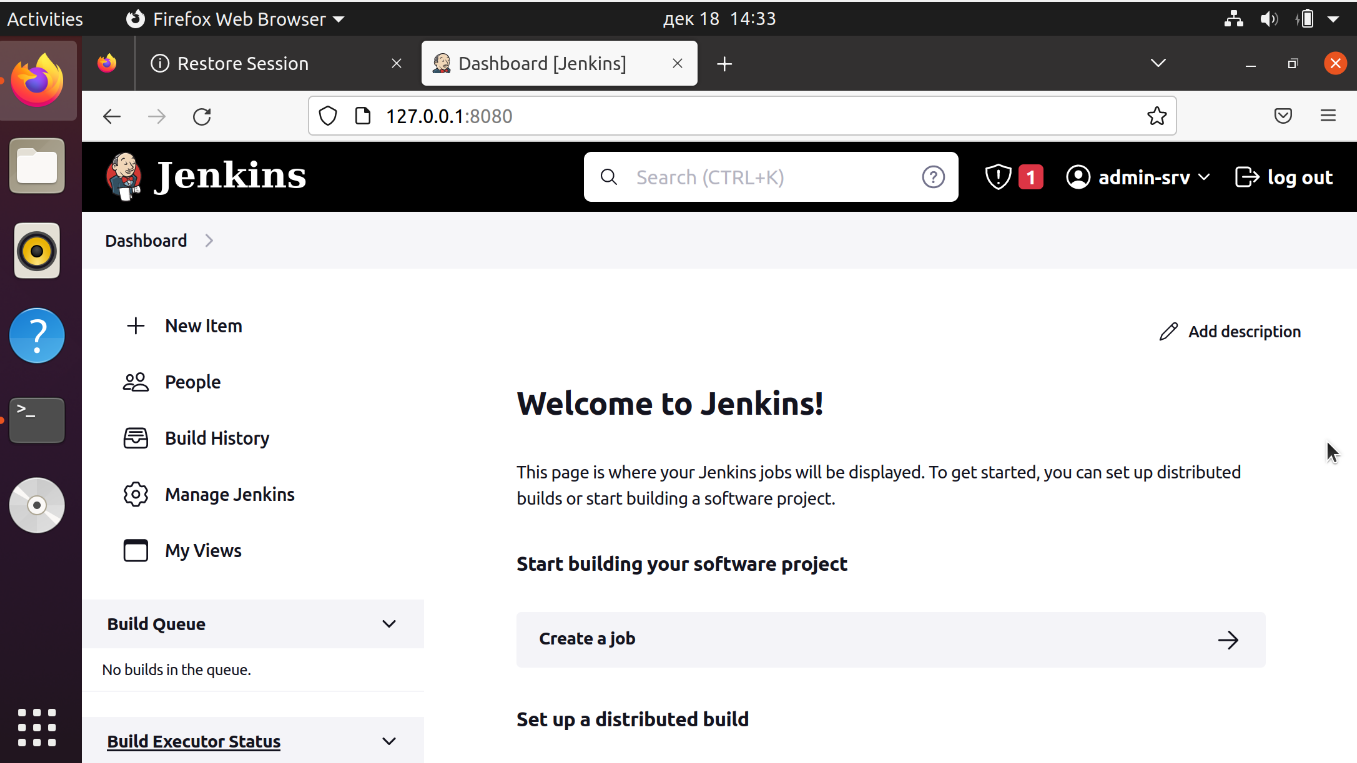


Видим, что в файле /etc/services за номером порта 8080 закреплен сервис http-alt, поэтому при фильтре с помощью grep надо указать имя этого сервиса, после чего мы видим, что процесс java с PID=676 занял этот порт и прослушивает входящие соединения, находится в состоянии LISTEN.

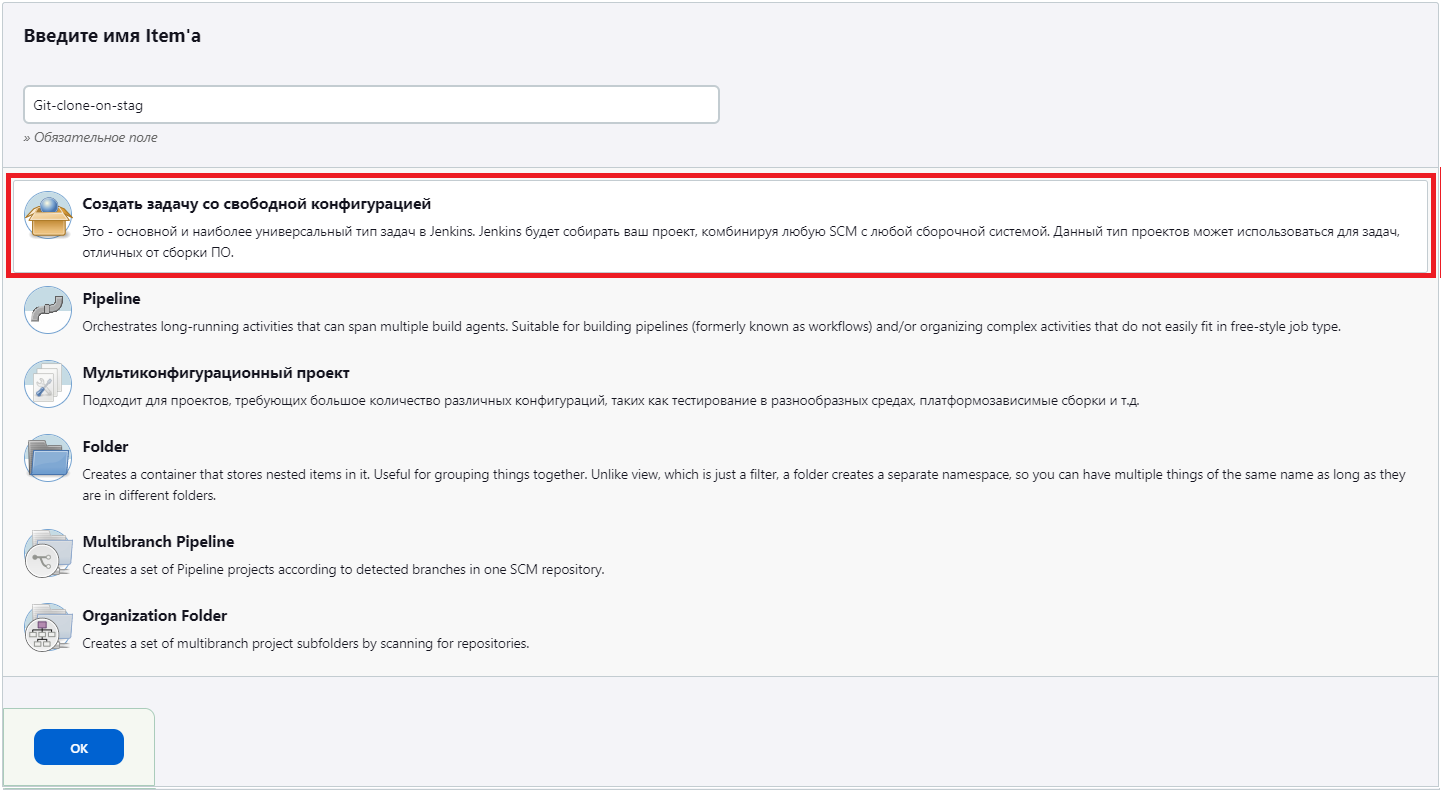
Теперь вы можете зайти через браузер на порт 8080 и открыть графический интерфейс пользователя Jenkins (все необходимые настройки доступа вы сделали ранее, в юните 2 данного модуля)



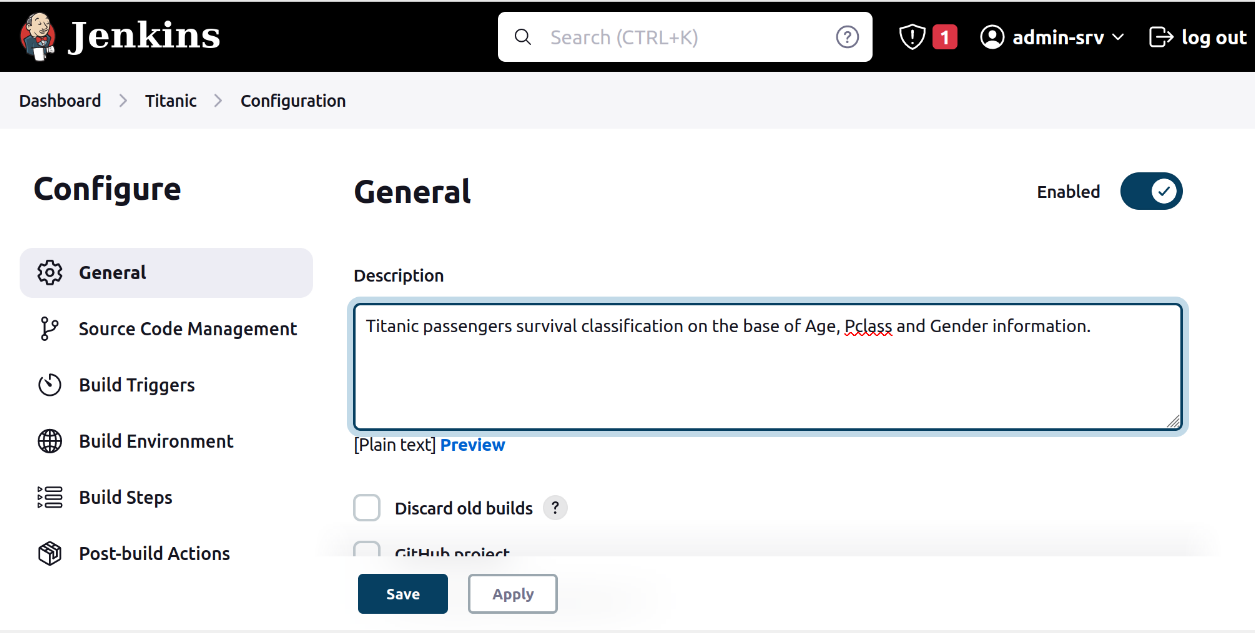
Давайте опишем конвейер операций проекта, с помощью которого мы будем автоматизировать этапы сборки, тестирования и развертывания. Для этого вам необходимо создать Задачу или Сборку (Job), для этого в главном меню надо выбрать опцию «Create a job». Термины «Задача» и «Сборка» являются общеупотребительными для Jenkins, далее будем использовать термин «Задача» или «Job» (в англоязычном варианте интерфейса).



Далее вам надо задать имя задачи и выбрать тип задачи. Например, вы можете выбрать опцию «Создать задачу со свободной конфигурацией» (в англоязычном варианте «Freestyle project»).



Далее вам будет предложена возможность сделать настройки задачи, описывающие собственно те операции, которые будет выполнять Jenkins.

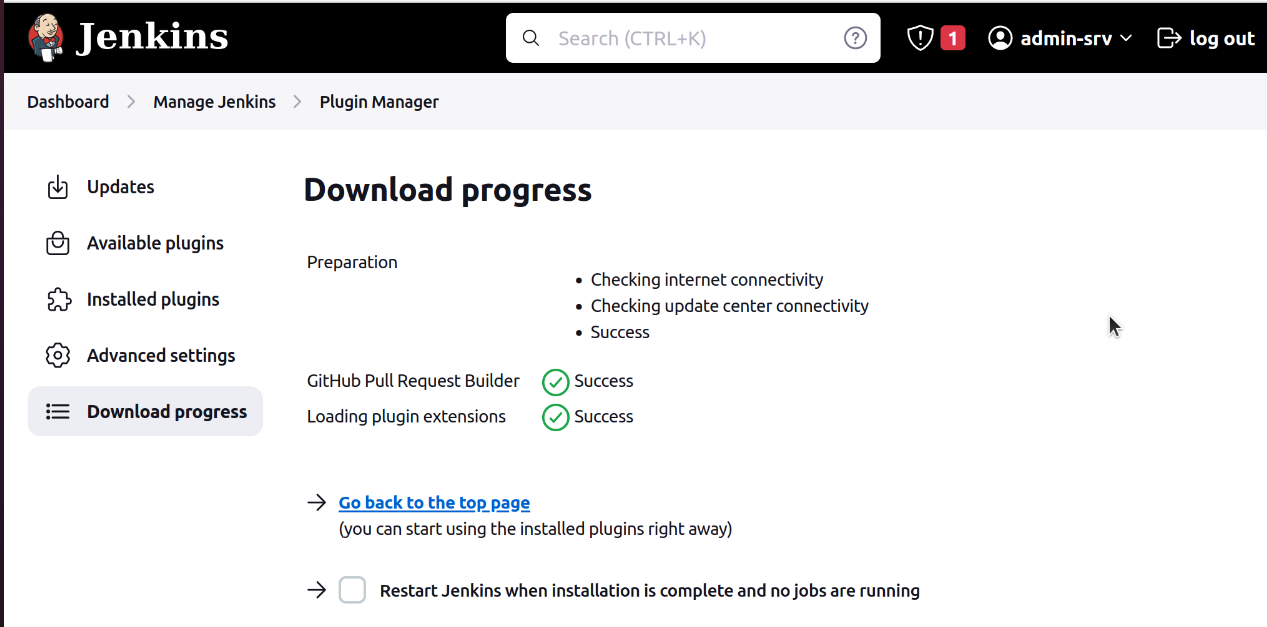


Используя соответствующие вкладки в панели меню действий можно настроить общие параметры (General) задачи (jobs) Jenkins, работу с исходным кодом (Source Code Management), управление триггерами (Build Triggers), управление настройкой среды окружения (environment) для выполнения программы (Build Environment), описать шаги сборки проекта (Build Steps) и задать действия, выполняемые после сборки проекта (Post-build Actions).

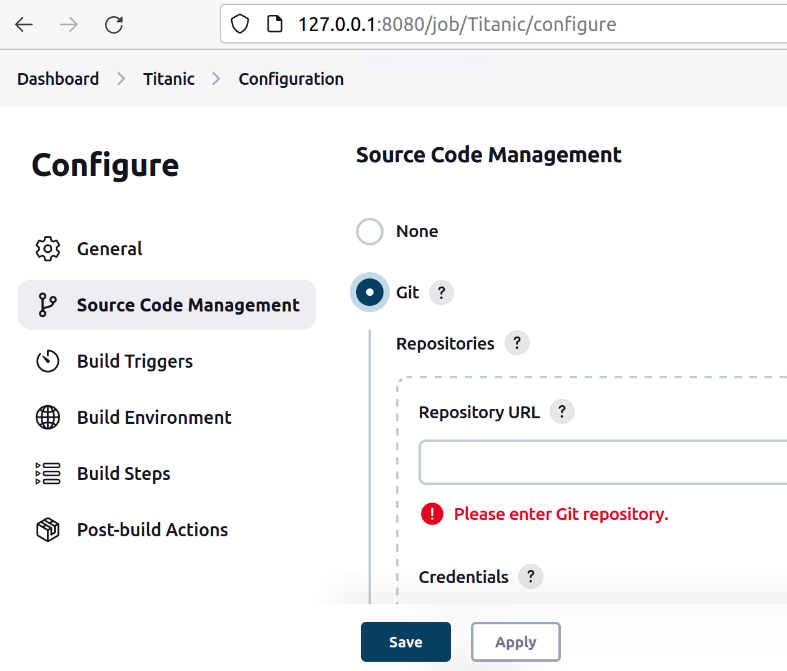
В этом юните будем настраивать взаимодействие Jenkins и github.com, где находится рабочий репозиторий нашего проекта машинного обучения. Для настройки и корректной работы такого взаимодействия требуется установить дополнительные плагины, обеспечивающие интеграцию git и Jenkins:

* плагин клиента git ([плагин](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://wiki.jenkins-ci.org/display/JENKINS/Git%2BClient%2BPlugin&usg=ALkJrhgqWYS5CxBTpKWJsiVOw1I00j2TNw) совместно используемой библиотеки для других плагинов Jenkins, связанных с Git),
* плагин git (для интеграции git с Jenkins),
* плагин GitHub API (этот плагин предоставляет GitHub API для других плагинов),
* плагин GitHub (этот плагин интегрирует GitHub в Jenkins),
* плагин GitHub Pull Request Builder (этот плагин создает запросы на извлечение из GitHub и сообщает о результатах).

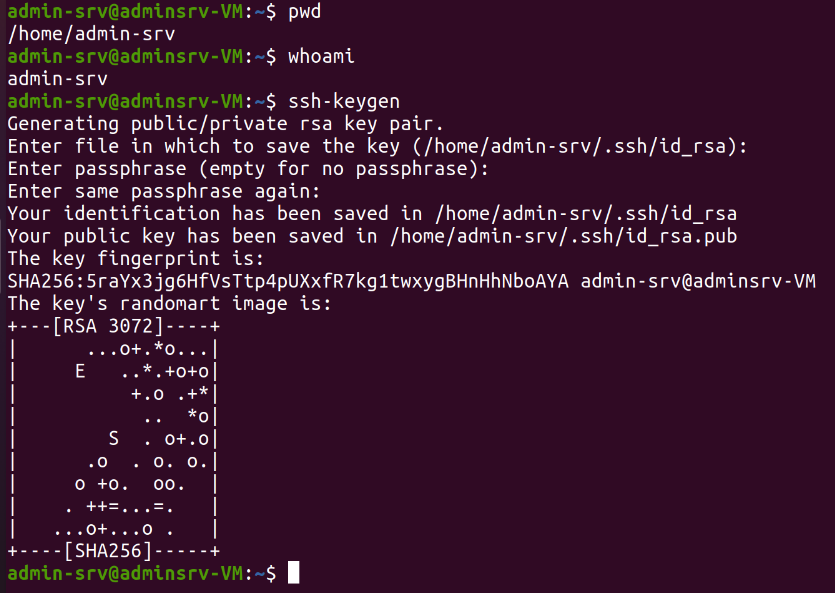
Для установки плагинов необходимо в главном меню Jenkins выбрать раздел «Manage Jenkins», в котором перейти в раздел «Plugin Manager». Далее поиском можно найти требуемый плагин, проверить его статус и, при необходимости, установить. Вот как выглядит результат успешной установки плагина “GitHub Pull Request Builder”



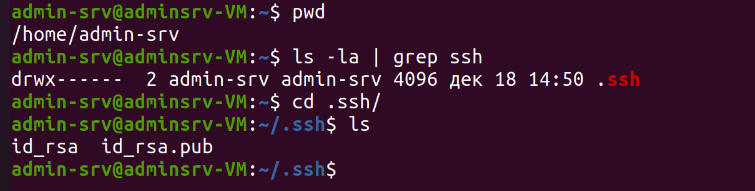
В Jenkins можно настроить взаимодействие с git репозиторием, для этого в пункте «Управление исходным кодом» (Source Code Management), необходимо выбрать пункт «Git».



Как вы уже знаете из Модуля 1, для взаимодействия с репозиторием на github.com вам понадобится настроить аутентификацию с помощью ssh ключей. Для этого необходимо создать публичный и приватный ключи на admin-srv и настроить соответствующий интерфейс доступа в настройках безопасности репозитория проекта на github.com. Создайте публичный (public) и приватный (private) ключи ssh на виртуальном сервере admin-srv, на котором установлен Jenkins, с помощью команды ssh-keygen.



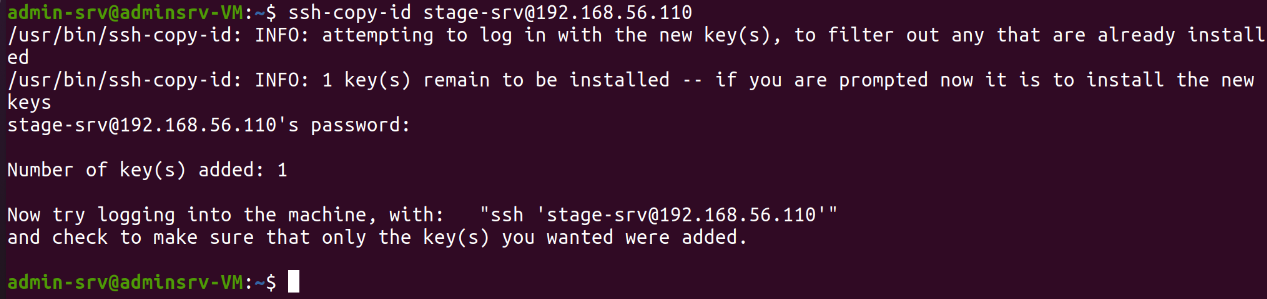
Дальше перейдите в каталог .ssh, который создался в домашней директории, по умолчанию ключи генерируются в данной директории.



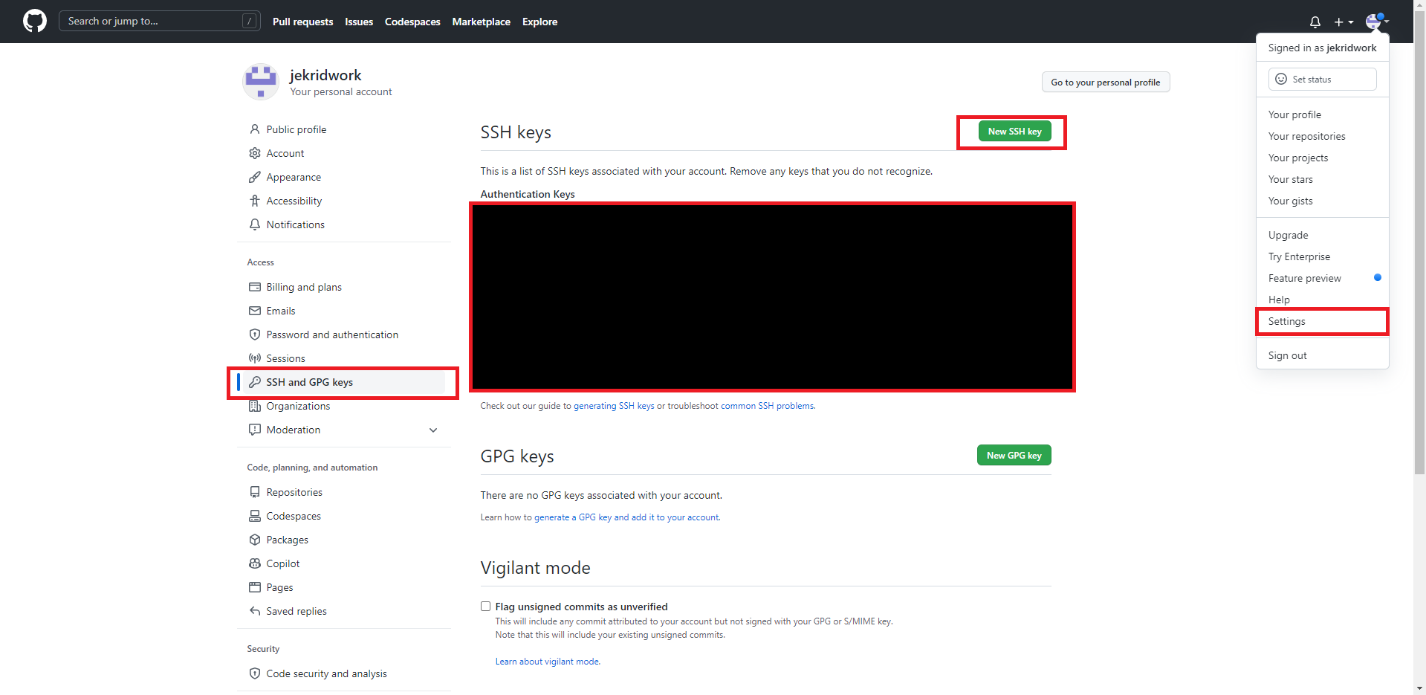
Файл публичного ключа id\_rsa.pub можно открыть в текстовом редакторе, данный ключ необходимо добавить в настройки репозитория на github.com, чтобы можно было взаимодействовать между проектом в github.com и Jenkins. Далее передаем созданный ключ, для авторизации на виртуальный сервер stage-srv, используя команду ssh-copy-id, чтобы можно было взаимодействовать между сервером администратора admin-srv, на котором установлен Jenkins, и сервером тестирования stage-srv, на котором будет проводиться автоматическое тестирование с использованием pytest. Обратите внимание, что для наличия сетевой связности между виртуальными машинами stage-srv и admin-srv необходимо правильно настроить виртуальные сетевые адаптеры этих виртуальных машин, а именно, сетевой адаптер должен иметь тип «Виртуальный адаптер хоста». Проверить сетевую связность можно с помощью команды ping. Также обратите внимание, что на stage-srv должен работать ssh сервер, в его наличии можно убедиться командой

**sudo systemctl status sshd**

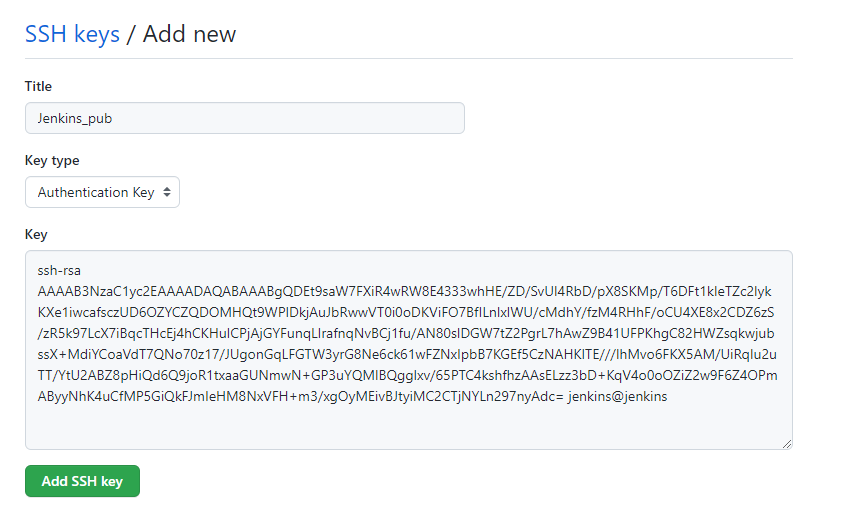
Если сетевая связь есть, и ssh сервер работает, то можно воспользоваться ssh-copy-id для передачи созданных ключей на сервер stage-srv



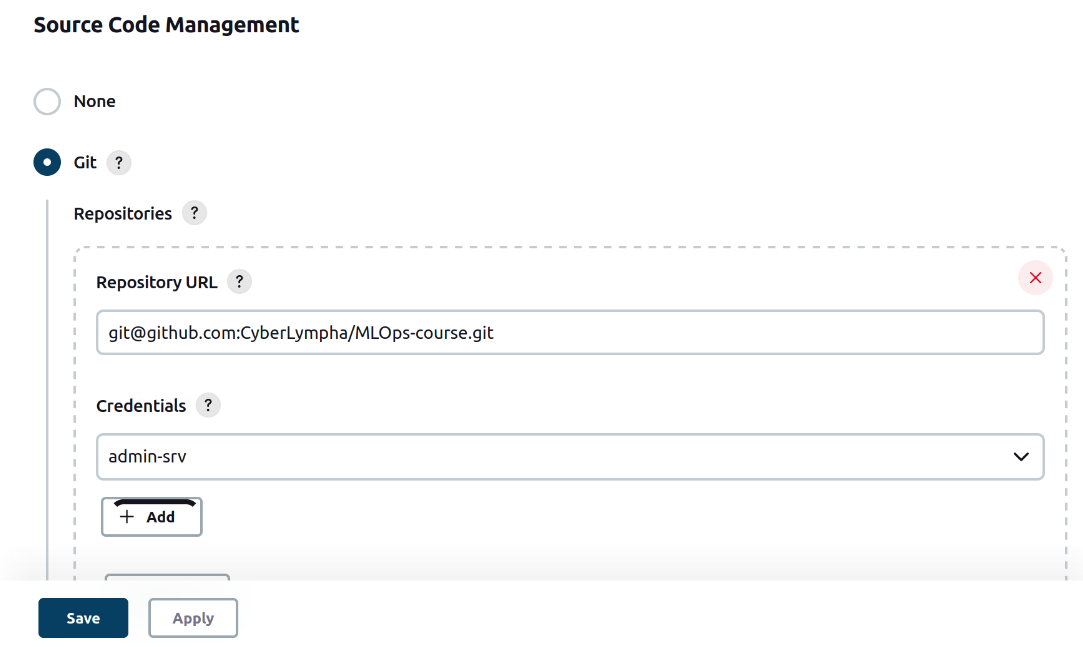
Чтобы добавить ключ в настройки безопасности репозитория на github.com, необходимо перейти по следующему пути «Settings» > «SSH and GPG keys» > New SSH key».



Далее необходимо указать заголовок (Title) и скопированный ключ (Key), в вашем случае будет другое значение ssh ключа.

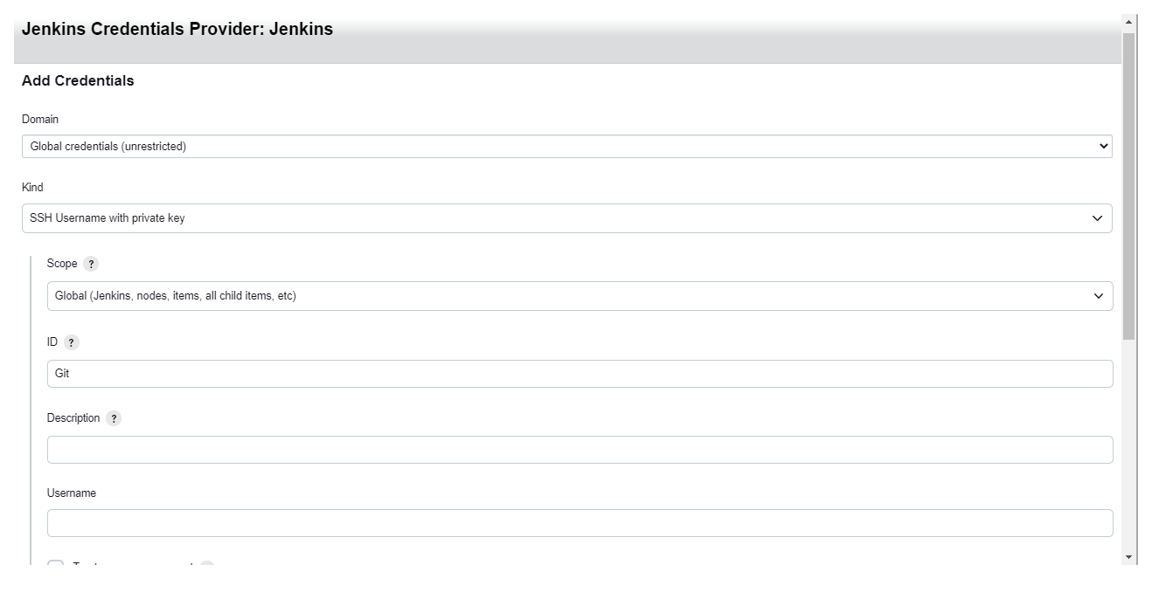


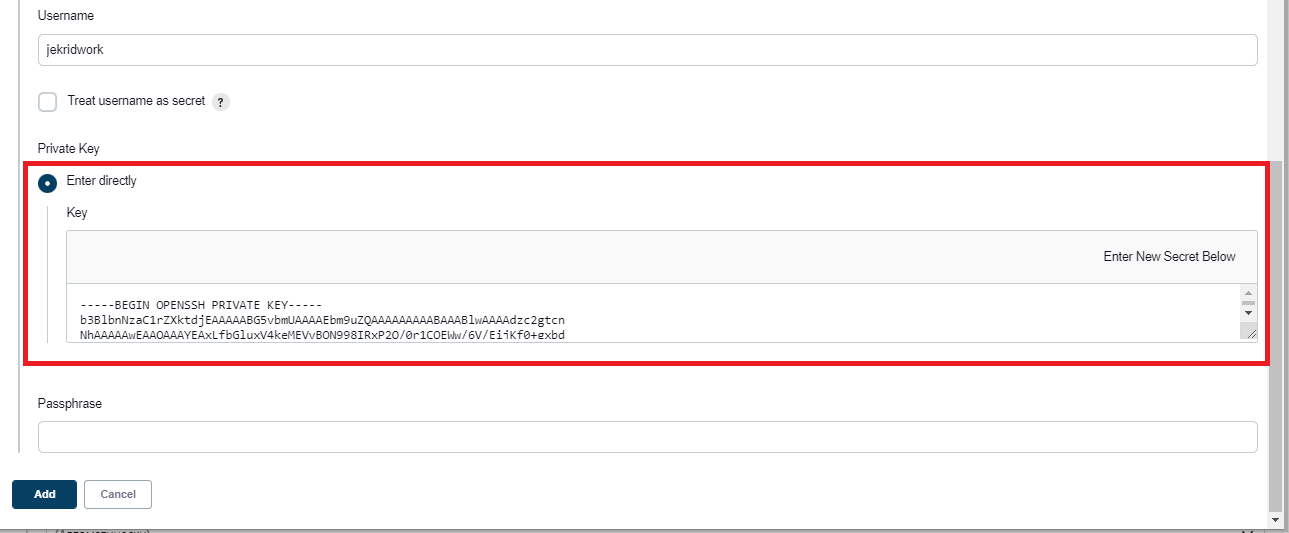
Теперь можно работать с git репозиторием, например, клонировать его или копировать из него отдельные файлы. После того как публичный ключ и файл были добавлены в Git, возвращаемся на сайт Jenkins. После выбора пункта «Git», появилась форма для сопряжения репозитория Git и Jenkins. В форме указываем ssh ссылку на Git, после нам необходимо добавить данные для подключения (Credentials). Для этого нажимаем на кнопку «Add».



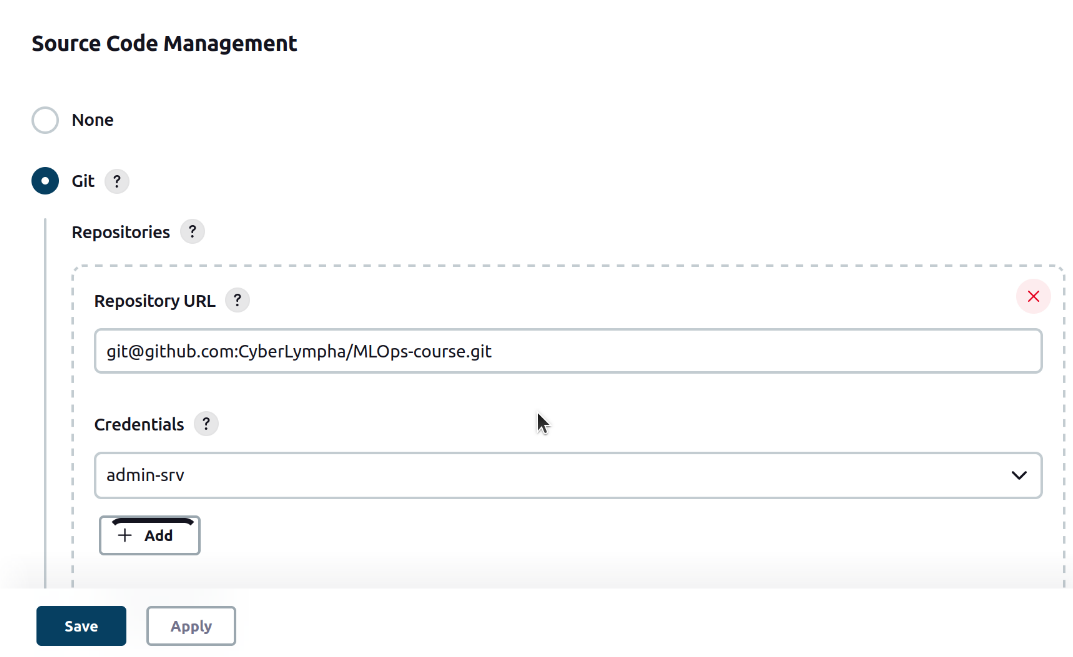
Дальше появится форма для заполнения данными чтобы произвести подключение к проекту на Git. Необходимые для заполнения поля:

* Kind – выбор способа авторизации (надо выбрать «SSH user name with private key»),
* Username – указывается имя для взаимодействия с git,
* Private key – приватный ключ который создался при использовании команды «ssh-keygen» (в примере приватный ключ лежит в файле git).

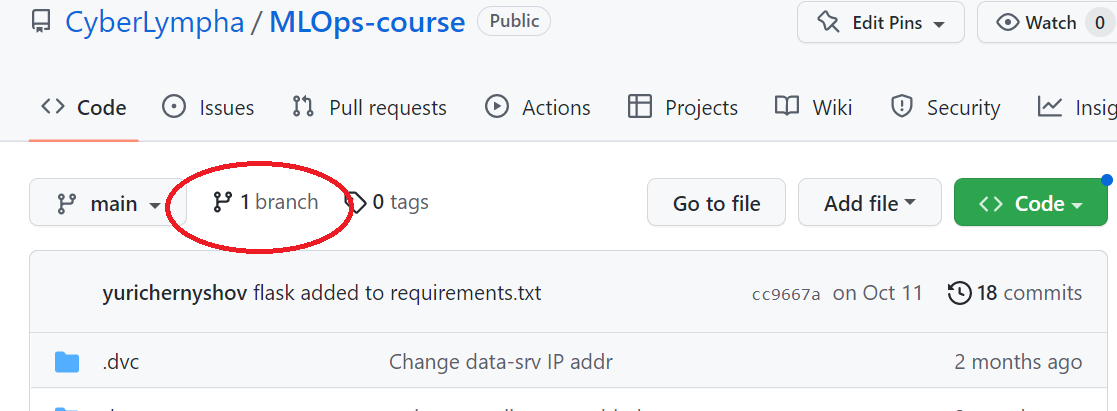


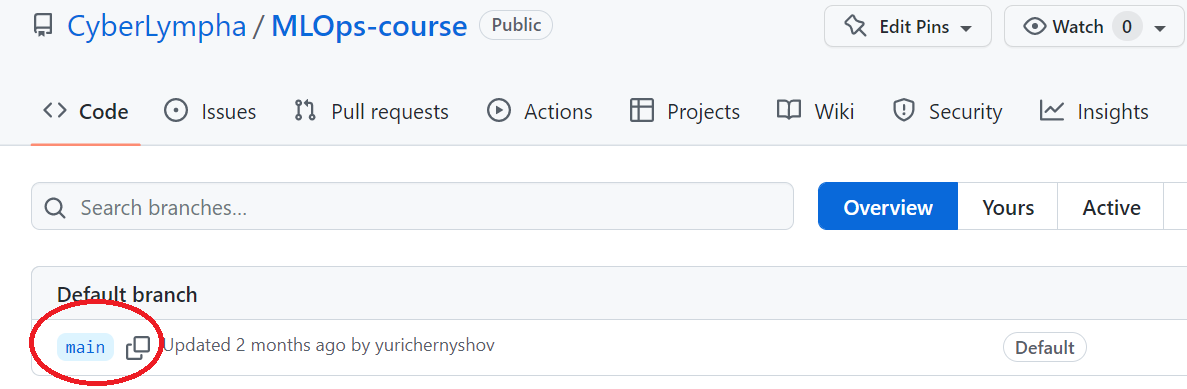


После добавления данных в поле «Credentials», появится новый пункт, с созданным параметрами.



Обратите внимание, что в связи с тем, что в github.com главная ветка может называться main или master может возникать ошибка при попытке клонирования git репозитория, если имя ветки указано неправильно. Чтобы исключить ошибку необходимо в git репозитории проверить название ветки

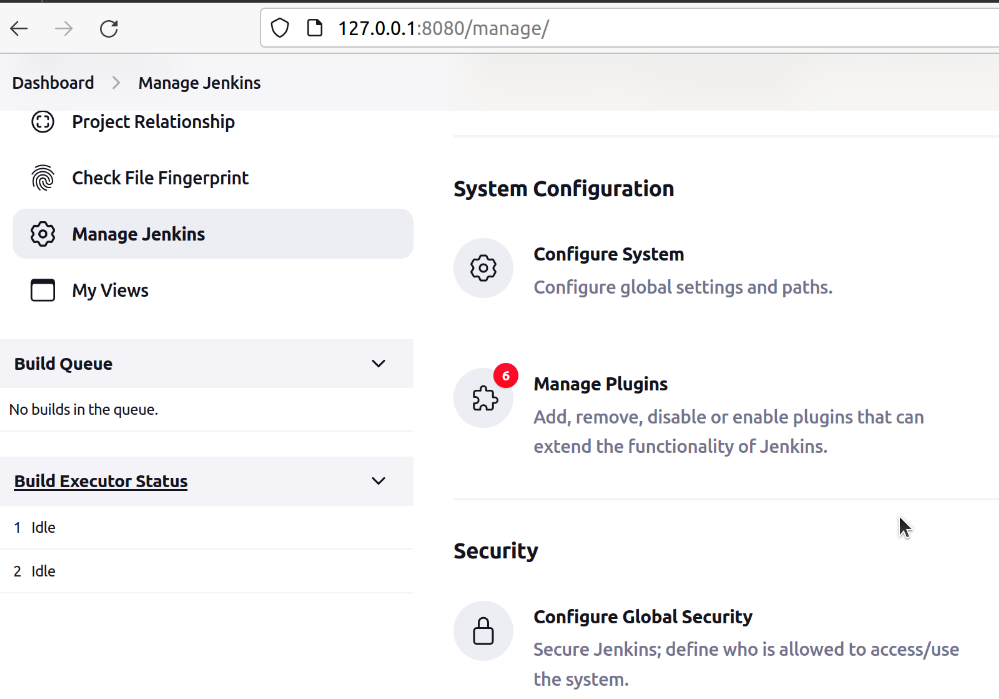




А затем уточнить название ветки в конфигурации задачи (job) в Jenkins, в разделе “Source Code Management”, “Branches to build”

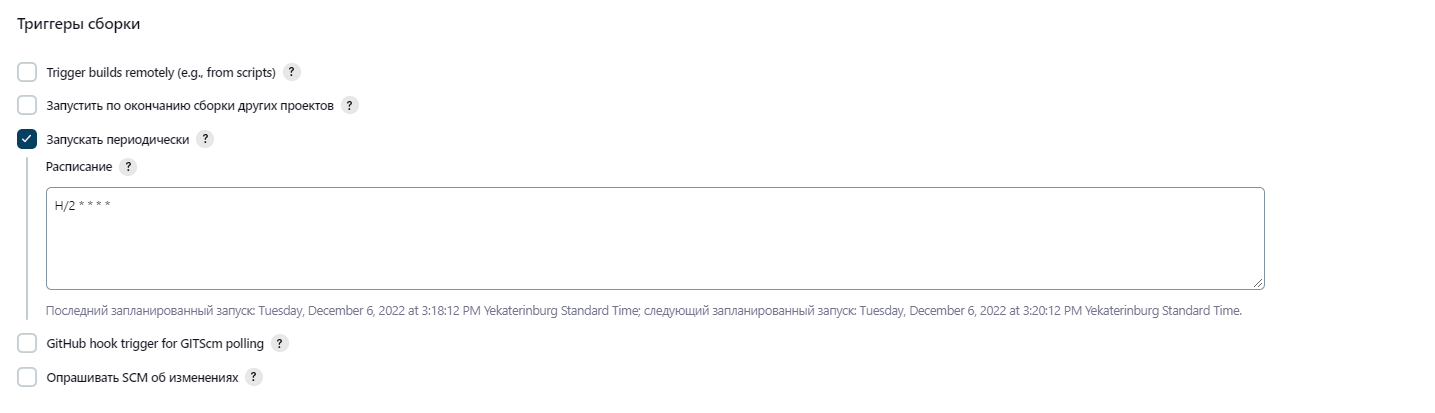


Еще одной ошибкой, которая может привести к сбою взаимодействия Jenkins и git по протоколу ssh, может быть отсутствие имени хоста для соединения в файле known\_hosts, служебном файле ssh, в который записываются имена известных хостов. Эта часть настраивается в разделе “Manage Jenkins”, “Configure Global Security”

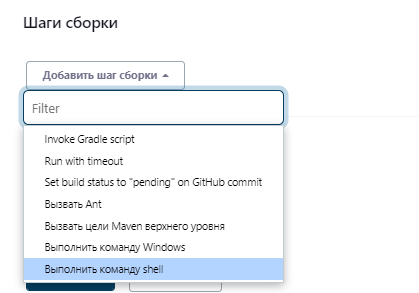


Вас могут подстерегать и другие неточности, ошибки, или особенности работы конкретной версии Jenkins, поэтому рекомендуем внимательно читать логфайл, в который пишутся результаты выполнения сборки. Обычно в этом файле можно найти описание и номер ошибки, что позволить найти решение в документации или в тематических форумах.

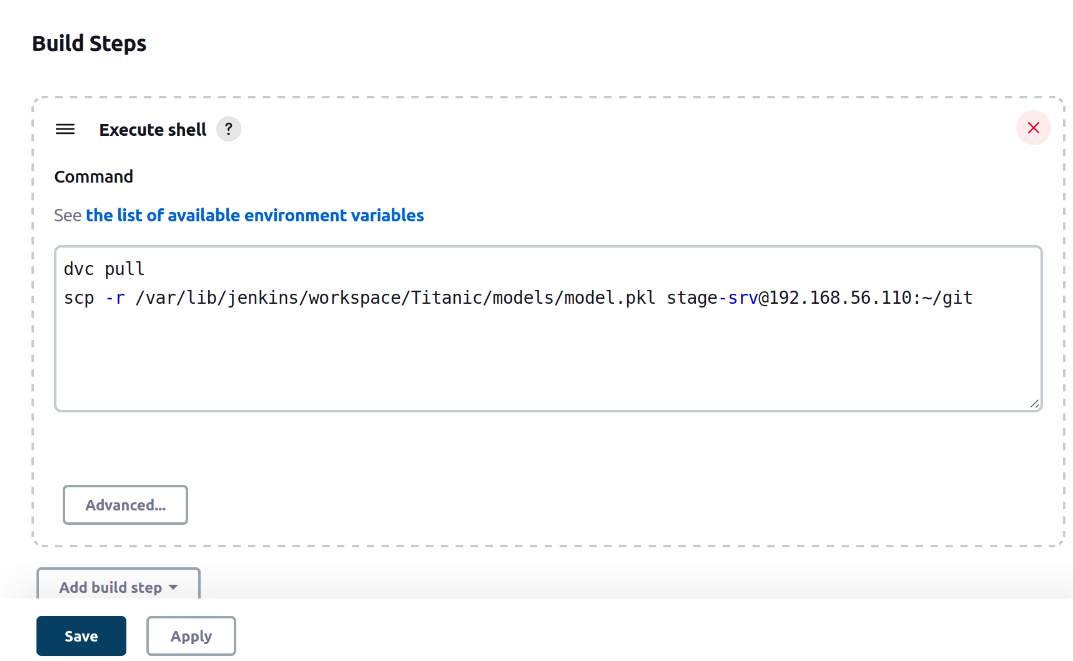
После успешной настройки взаимодействия Jenkins и git укажем, чтобы наша сборка запускалась каждые 2 минуты. Конечно, для реальных проектов это избыточно, обычно сборка всего проекта стартует один раз в сутки, например, в ночное время. Мы поставили такой кратковременный промежуток между сборками в учебных целях.



Теперь укажите команду, которую необходимо исполнять при взаимодействии с проектом git. В данном случае, части проекта, который был выгружен с git, будут передаваться на виртуальный сервер stage-srv, для дальнейшей сборки и тестирования проекта.

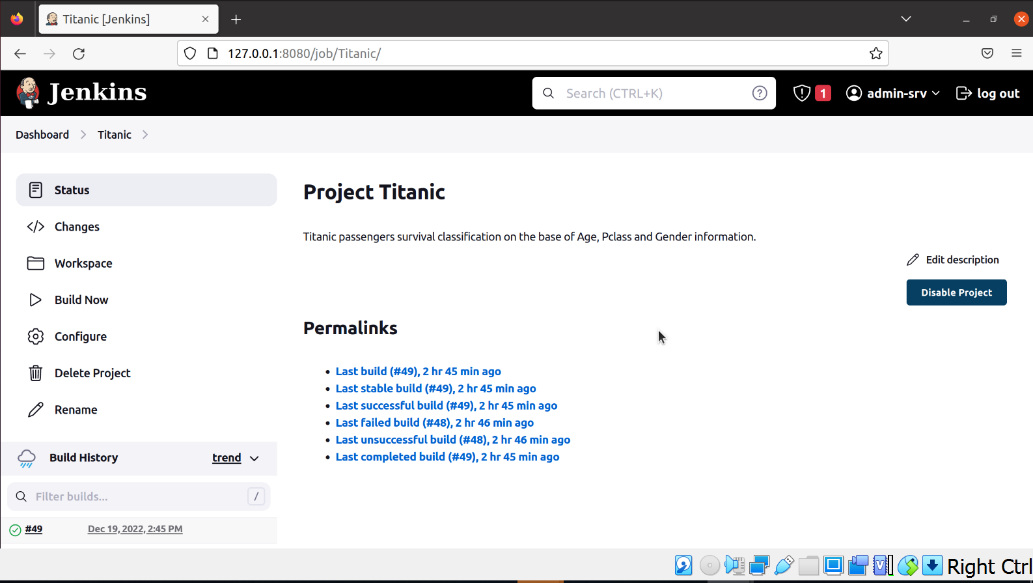


Используемые команды для обновления данных (при необходимости) из dvc хранилища и передачи данных из репозитория github.com на виртуальный сервер stage-srv

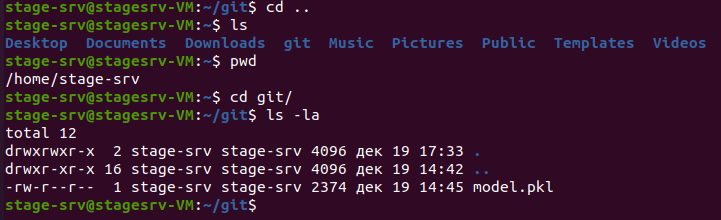


Как видно из содержания, сначала выполняется обновление данных в рабочем репозитории. Мы настроили интеграцию Jenkins и github.com и теперь в /var/lib/Jenkins/workspace/Titanic находятся файлы из git репозитория. Однако обученная модель в формате pickle хранится в dvc хранилище на сервере data-srv, в git репозитории сохраняются только метафайлы, описывающие правила хранения в dvc. Поэтому для получения данных из dvc хранилища на сервере admin-srv необходимо установить и настроить dvc, как вы это делали в Модуле 1. В том числе, не забудьте установить локальный параметр – пароль для пользователя сервера data-srv, который не сохраняется в git из соображений безопасности. После того как установлен и настроен dvc между admin-srv и data-srv, а также ssh между admin-srv и stage-srv, вы можете запустить задачу (job) Jenkins, которая сначала обновить данные из dvc, а затем загрузит модель /models/model.pkl на сервер stage-srv для тестирования.

Теперь вам необходимо сохранить созданную задачу после чего вы можете попробовать запустить эту задачу вручную, без ожидания 2 минут, для быстрой проверки. Для этого надо выбрать опцию “Build Now”



Теперь если вы зайдете на виртуальный сервер stage-srv в директорию «git» (директория заранее создается), то после выполнения созданной вами задачи в Jenkins модель для тестирования появится в указанной директории.



Итак, вы настроили интеграцию Jenkins и git репозитория и создали простые скрипты автоматизации в Jenkins, которые выполняют shell команды для выгрузки данных из dvc и передачи данных на удаленный сервер.

## Тест

1. Какой способ авторизации используется на github.com и который необходимо настроить для нормальной работы конвейера проекта и операций по взаимодействию с git репозиторием? (0.25)
   1. Шлюз защиты по IP адресу
   2. Доступ по паролю
   3. **ssh ключи**
   4. Нет специального способа авторизации
2. Какой тип задачи (job) Jenkins мы использовали в юните? (0.25)
   1. Папка (Folder)
   2. Конвейер (Pipeline)
   3. **Задача со свободной конфигурацией**
   4. Мультиконфигурационный проект
3. Какая утилита используется для генерации пары ключей ssh? (0.25)
   1. ssh-keycreate
   2. **ssh-keygen**
   3. ssh-keymake
   4. ssh-keyload
4. Какой шаблон задает запуск выполнения операции в Jenkins каждые 2 минуты? (0.25)
   1. \* 2 \* \* \*
   2. **2 \* \* \* \***
   3. \* \* \* \* 2
   4. \* \* \* 2 \*

## Итоги/выводы

В этом юните вы настроили в Jenkins взаимодействие с git репозиторием. Вы создали сценарий, по которому Jenkins циклически проводит клонирование кода репозитория по заданному расписанию, один раз в две минуты. В качестве примера описан процесс клонирования рабочего git репозитория на сервер тестирования stage-srv.

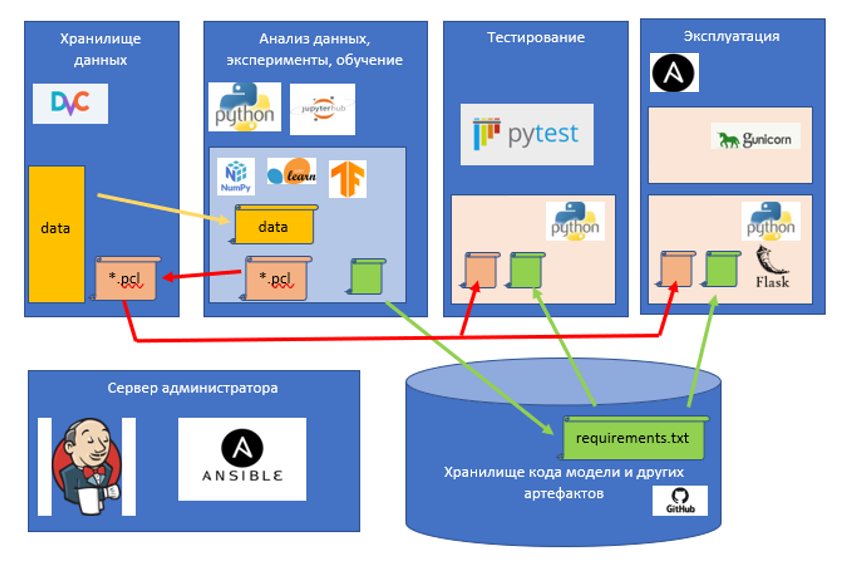
# Модуль 2. Юнит 7. Настройка сервера тестирования и автоматизированное тестирование с pytest

## Введение:

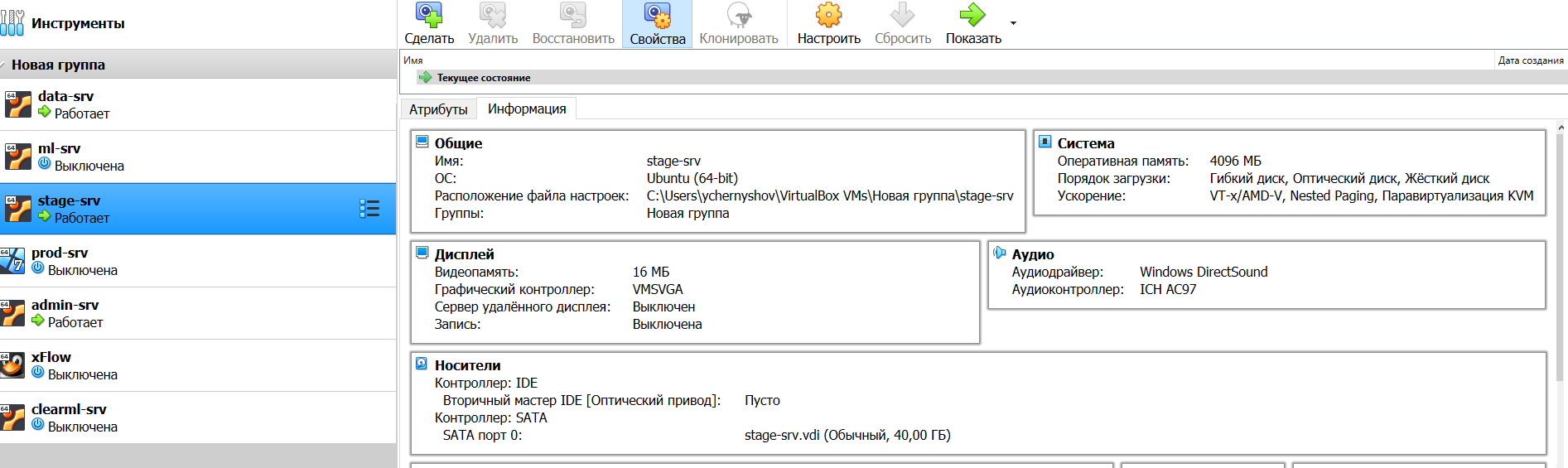
Тестирование является важнейшим этапом в любом проекте разработки программного обеспечения, в том числе в проектах машинного обучения. Перед выводом разработанной системы в эксплуатацию необходимо убедиться в работоспособности программного обеспечения, минимизировать риски сбоев, проверить производительность. Существует множество видов тестирования и подходов к выполнению тестов. Автоматизация тестирования позволяет исключить ошибки, связанные с человеческим фактором, выполнить большее количество тестов за меньшее время, может выполняться в нерабочее время, например, ночью, чтобы эффективно использовать аппаратные мощности. В этом юните вы осуществите настройку сервера автоматизированного тестирования, на котором будет тестироваться разработанная вами модель машинного обучения перед выводом в эксплуатацию.

## Содержание:

Для тестирования в созданной вами инфраструктуре проекта предназначен сервер stage-srv, на котором используется утилита pytest для автоматизации тестирования. На сервере stage-srv должен быть установлен интерпретатор python для выполнения скриптов.

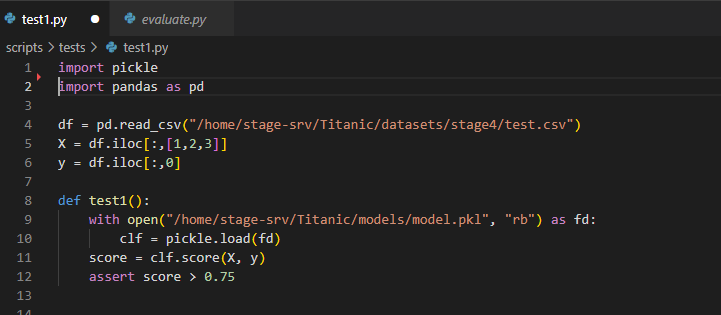


Ранее вы создали виртуальную машину stage-srv с операционной системой Ubuntu 20.04 с предустановленной версией интерпретатора python 3.8.



На виртуальный сервер stage-srv будет загружаться обученная модель машинного обучения model.pkl в формате pickle, а также тестовые скрипты, которые будут выполняться с использованием pytest. Перенос необходимых файлов (модели в формате pickle и скриптов автоматического тестирования), а также их использование на stage-srv будет управляться с помощью Jenkins.

Давайте сначала в рабочем репозитории создадим папки /scripts/tests для python скриптов, описывающих тесты. Эту папку надо будет также загрузить на сервер stage-srv и выполнять эти тесты автоматически с использованием pytest. Вы уже настроили ранее IDE VS Code для редактирования и удаленного запуска скриптов на рабочем сервере. С его помощью создайте папку и необходимый скрипт автоматизации тестирования. Создадим скрипт /scripts/tests/test1.py, который оценивает качество работы модели машинного обучения на тестовых данных.



Тестовый скрипт /scripts/tests/test1.py и модель в формате pickle /models/model.pkl переносим с помощью команды scp в сборке Jenkins с сервера admin-srv на сервер тестирования stage-srv. В сценарий Jenkins для этого надо добавить команды

***scp -r /var/lib/jenkins/workspace/Titanic/datasets/stage4/test.csv stage-srv@192.168.56.110:~/Titanic/datasets/stage4/test.csv***

***scp -r /var/lib/jenkins/workspace/Titanic/models/model.pkl stage-srv@192.168.56.110:~/Titanic/models/model.pkl***

***scp -r /var/lib/jenkins/workspace/Titanic/scripts/tests/test1.py stage-srv@192.168.56.110:~/Titanic/tests/test1.py***

***scp -r /var/lib/jenkins/workspace/Titanic/requirements.txt stage-srv@192.168.56.110:~/Titanic/requirements.txt***

***ssh stage-srv@192.168.56.110 source /home/stage-srv/Titanic/venv/bin/activate***

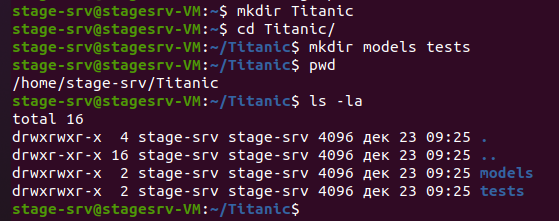
***ssh stage-srv@192.168.56.110 pip install -r /home/stage-srv/Titanic/requirements.txt***

***ssh stage-srv@192.168.56.110 pip install pytest***

***ssh stage-srv@192.168.56.110 python3 -m pytest --rootdir=/home/stage-srv/Titanic/tests /home/stage-srv/Titanic/tests/test1.py***

***ssh stage-srv@192.168.56.110 deactivate***

На сервере stage-srv в директории проекта, например, ~/Titanic, должны быть созданы папки для переноса модели ~/Titanic/models/ и для тестовых скриптов ~/Titanic/ tests/



Теперь можно запустить задачу в Jenkins на выполнение

**Started by user** [**admin-srv**](http://127.0.0.1:8080/user/admin-srv)

**Running as SYSTEM**

**Building in workspace /var/lib/jenkins/workspace/Titanic**

**The recommended git tool is: NONE**

**using credential Git**

**> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/Titanic/.git # timeout=10**

**Fetching changes from the remote Git repository**

**> git config remote.origin.url git@github.com:CyberLympha/MLOps-course.git # timeout=10**

**Fetching upstream changes from git@github.com:CyberLympha/MLOps-course.git**

**> git --version # timeout=10**

**> git --version # 'git version 2.25.1'**

**using GIT\_SSH to set credentials**

**> git fetch --tags --force --progress -- git@github.com:CyberLympha/MLOps-course.git +refs/heads/\*:refs/remotes/origin/\* # timeout=10**

**> git rev-parse refs/remotes/origin/main^{commit} # timeout=10**

**Checking out Revision 170fb147a5650acbdc4dcf63eed76fe581c2c527 (refs/remotes/origin/main)**

**> git config core.sparsecheckout # timeout=10**

**> git checkout -f 170fb147a5650acbdc4dcf63eed76fe581c2c527 # timeout=10**

**Commit message: "test scripts changed"**

**> git rev-list --no-walk 11028711e0e136312b81b38fc8e98521e8cdfc87 # timeout=10**

**[Titanic] $ /bin/sh -xe /tmp/jenkins6339381885510545336.sh**

**+ dvc pull**

**Everything is up to date.**

**[Titanic] $ /bin/sh -xe /tmp/jenkins1130229151100658543.sh**

**+ scp -r /var/lib/jenkins/workspace/Titanic/datasets/stage4/test.csv stage-srv@192.168.56.110:~/Titanic/datasets/stage4/test.csv**

**+ scp -r /var/lib/jenkins/workspace/Titanic/models/model.pkl stage-srv@192.168.56.110:~/Titanic/models/model.pkl**

**+ scp -r /var/lib/jenkins/workspace/Titanic/scripts/tests/test1.py stage-srv@192.168.56.110:~/Titanic/tests/test1.py**

**+ scp -r /var/lib/jenkins/workspace/Titanic/requirements.txt stage-srv@192.168.56.110:~/Titanic/requirements.txt**

**+ ssh stage-srv@192.168.56.110 source /home/stage-srv/Titanic/venv/bin/activate**

**+ ssh stage-srv@192.168.56.110 pip install -r /home/stage-srv/Titanic/requirements.txt**

**Requirement already satisfied: click==8.1.3 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 1)) (8.1.3)**

**Requirement already satisfied: Flask==2.2.2 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 2)) (2.2.2)**

**Requirement already satisfied: importlib-metadata==5.0.0 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 3)) (5.0.0)**

**Requirement already satisfied: itsdangerous==2.1.2 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 4)) (2.1.2)**

**Requirement already satisfied: Jinja2==3.1.2 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 5)) (3.1.2)**

**Requirement already satisfied: joblib==1.2.0 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 6)) (1.2.0)**

**Requirement already satisfied: MarkupSafe==2.1.1 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 7)) (2.1.1)**

**Requirement already satisfied: numpy==1.23.3 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 8)) (1.23.3)**

**Requirement already satisfied: pandas==1.5.0 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 9)) (1.5.0)**

**Requirement already satisfied: python-dateutil==2.8.2 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 10)) (2.8.2)**

**Requirement already satisfied: pytz==2022.4 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 11)) (2022.4)**

**Requirement already satisfied: PyYAML==6.0 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 12)) (6.0)**

**Requirement already satisfied: scikit-learn==1.1.2 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 13)) (1.1.2)**

**Requirement already satisfied: scipy==1.9.2 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 14)) (1.9.2)**

**Requirement already satisfied: six==1.16.0 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 15)) (1.16.0)**

**Requirement already satisfied: threadpoolctl==3.1.0 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 16)) (3.1.0)**

**Requirement already satisfied: Werkzeug==2.2.2 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 17)) (2.2.2)**

**Requirement already satisfied: zipp==3.9.0 in ./.local/lib/python3.8/site-packages (from -r /home/stage-srv/Titanic/requirements.txt (line 18)) (3.9.0)**

**+ ssh stage-srv@192.168.56.110 pip install pytest**

**Requirement already satisfied: pytest in /usr/lib/python3/dist-packages (4.6.9)**

**Requirement already satisfied: more-itertools>=4.0.0 in /usr/lib/python3/dist-packages (from pytest) (4.2.0)**

**+ ssh stage-srv@192.168.56.110 python3 -m pytest --rootdir=/home/stage-srv/Titanic/tests /home/stage-srv/Titanic/tests/test1.py**

**============================= test session starts ==============================**

**platform linux -- Python 3.8.10, pytest-4.6.9, py-1.8.1, pluggy-0.13.0**

**rootdir: /home/stage-srv/Titanic/tests**

**collected 1 item**

**Titanic/tests/test1.py F [100%]**

**=================================== FAILURES ===================================**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ test1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**def test1():**

**with open("/home/stage-srv/Titanic/models/model.pkl", "rb") as fd:**

**clf = pickle.load(fd)**

**score = clf.score(X, y)**

**> assert score > 0.75**

**E assert 0.5977443609022557 > 0.75**

**Titanic/tests/test1.py:16: AssertionError**

**=============================== warnings summary ===============================**

**test1.py::test1**

**/home/stage-srv/.local/lib/python3.8/site-packages/sklearn/base.py:493: FutureWarning: The feature names should match those that were passed during fit. Starting version 1.2, an error will be raised.**

**Feature names unseen at fit time:**

**- 1**

**- 24.0**

**Feature names seen at fit time, yet now missing:**

**- 1.2**

**- 35.0**

**warnings.warn(message, FutureWarning)**

**-- Docs:** [**https://docs.pytest.org/en/latest/warnings.html**](https://docs.pytest.org/en/latest/warnings.html)

**===================== 1 failed, 1 warnings in 0.91 seconds =====================**

**Build step 'Execute shell' marked build as failure**

**Finished: FAILURE**

Из анализа вывода на консоль вы можете видеть, что проблема тестирования заключается в том, что построенная модель на тестовой выборке показывает результат 0.59, что меньше ожидаемой точности 0.75.

**> assert score > 0.75**

**E assert 0.5977443609022557 > 0.75**

Таким образом, тестирование с pytest работает, для решения проблемы надо улучшать качество работы модели.

## Тест

1. Какую утилиту мы использовали для автоматизации тестирования? (0.25)
   1. **pytest**
   2. unittest
   3. tensorflow
   4. autotest
2. Какое условие проверялось при тестировании? (0.25)
   1. **Скор модели не меньше 0.75**
   2. Скорость инференса модели быстрее 1 сек.
   3. Потребление оперативной памяти не превышает 1Мб
   4. Модель обучается не более 5 секунд
3. Какой флаг в pytest задает корневую директорию для работы утилиты? (0.25)
   1. --root
   2. **--rootdir**
   3. --dir
   4. --flag
4. Какие shell команды использовались при автоматизации? (0.25)
   1. **scp**
   2. **ssh**
   3. make
   4. init

## Итоги/выводы

В этом юните с использованием созданной инфраструктуры вы создали инструмент для автоматизированного тестирования модели машинного обучения перед ее выводом в эксплуатацию.

# Модуль 2. Юнит 8. Настройка сервера эксплуатации с ansible и автоматизированное развертывание с Jenkins.

## Введение:

Итогом разработки в проекте машинного обучения является выведенная в эксплуатацию модель. Как вы уже знаете, для успешной эксплуатации модели машинного обучения необходима точность в создании и настройке инфраструктуры, в которой будет работать модель. Ошибки в настройке могут привести к серьезным проблемам. Поэтому целесообразно применять автоматизированные процедуры для настройки инфраструктуры и установки правильной конфигурации программного обеспечения, в соответствии с идеологией Infrastructure-as-a-Code (IaaC), которая является ключевой в DevOps и MLOps. Утилита ansible предназначена для автоматизированной настройки инфраструктуры. С ее помощью системные администраторы, DevOps/MLOps инженеры автоматизируют рутинные операции по установке и настройке оборудования и программного обеспечения, запуску программ. В этом юните используем ansible для автоматического конфигурирования сервера эксплуатации и запуска на нем web сервиса, использующего обученную и протестированную на предыдущих шагах модель машинного обучения.

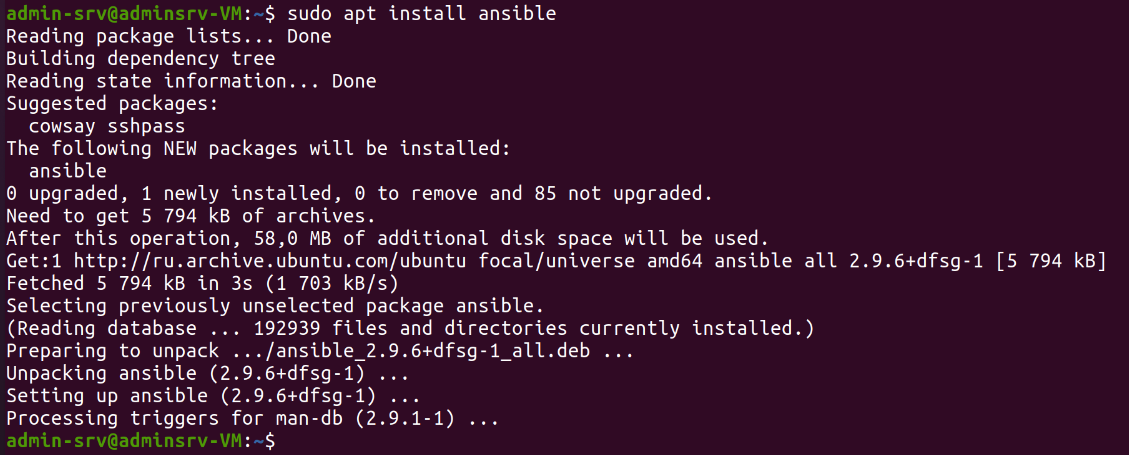
## Содержание:

Для эксплуатации модели машинного обучения вы создали ранее виртуальный сервер prod-srv, теперь на нем необходимо создать правильную конфигурацию данных и библиотек для обеспечения повторяемости результата. Для выполнения всех этих настроек используем ansible скрипты, которые вызываются из Jenkins.

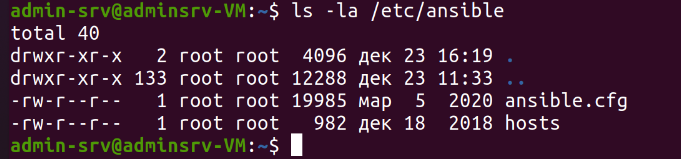
Утилита ansible является очень неприхотливой в части технических требований к инфраструктуре и используемому программному обеспечению. По сути, для ее нормальной работы достаточно установленного на центральном сервере администрирования программного обеспечения ansible, настроенного ssh доступа до всех управляемых машин и наличия интерпретатора python3 на управляемых машинах, причем последнее требование для некоторых действий с ansible даже не обязательно.

Сначала на сервере администратора admin-srv установите пакет Ansible с помощью команды

**sudo apt install ansible**

****

После установки вам надо отредактировать служебный файл ansible для описания инфраструктуры его работы, который находится по адреску /etc/ansible/hosts.

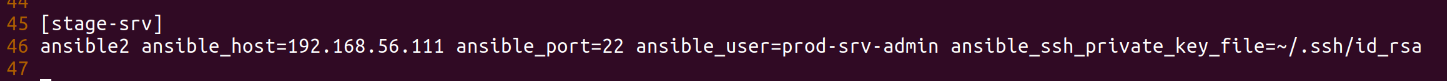


Внутри этого файла укажем название виртуального сервера prod-srv и данные для подключения к нему. Текст, который необходимо добавить в файл hosts для создания подключения к удаленному серверу prod-srv:

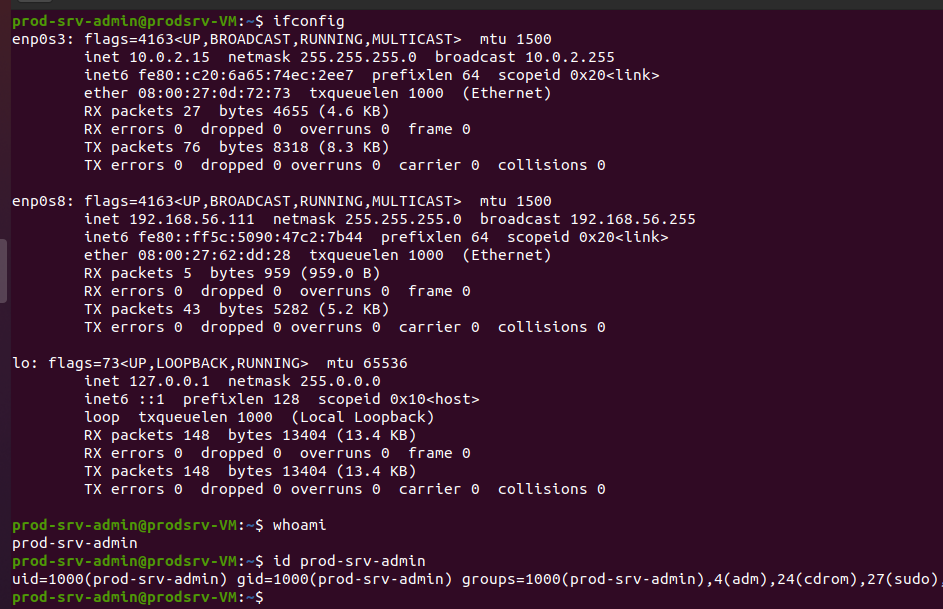
***[prod-srv]***

***ansible2 ansible\_host=192.168.56.111 ansible\_port=22 ansible\_user=prod-srv-admin ansible\_ssh\_private\_key\_file=~/.ssh/id\_rsa***

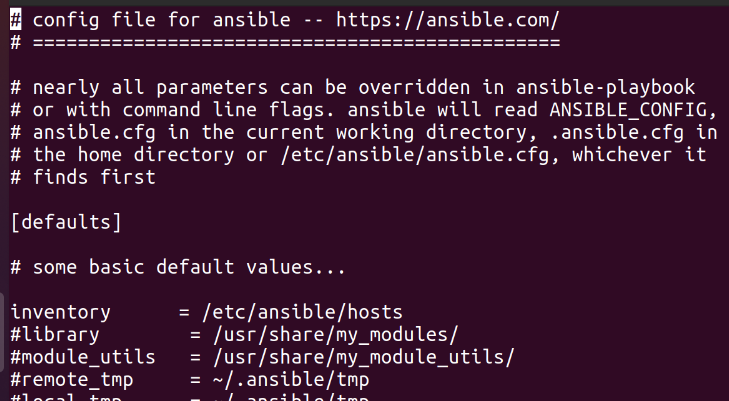
Вот часть файла /etc/ansible/hosts, открытая в редакторе vi, для нумерации строк слева надо нажать “:” и набрать команду set nu.



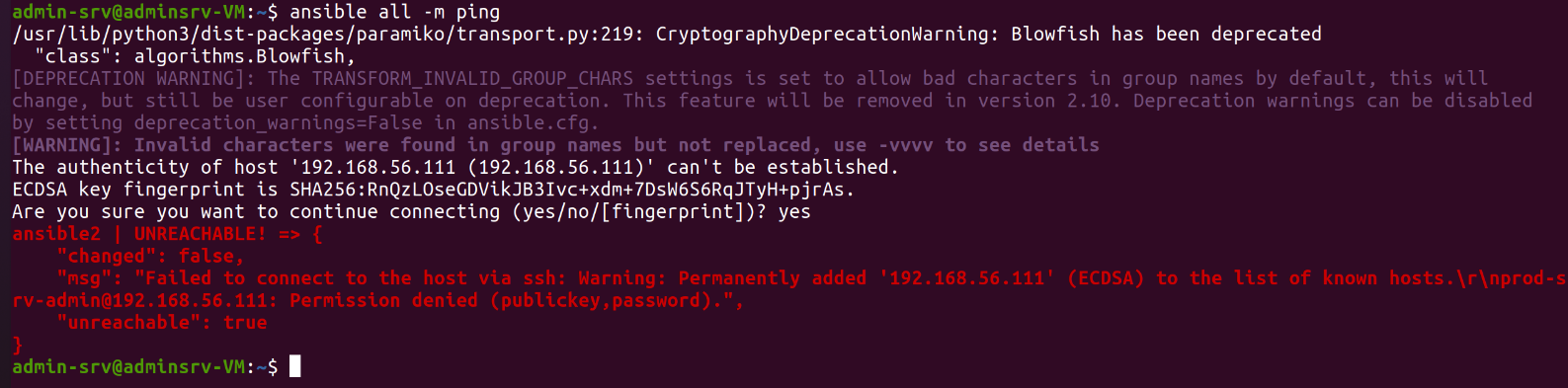
В качестве параметра ansible\_host необходимо указать открытый IP адрес сервера prod-srv, на котором будет эксплуатироваться модель, а ansible\_user должен быть пользователь prod-srv с правами sudo



Далее изменим конфигурационный файл /etc/ansible/ansible.cfg, для подключения ранее созданного файла. В конфигурационном файле необходимо раскоментировать строку «inventory» и указать путь до созданного файла. Из даного файла, будут браться устройства к которым можно будет подключаться.

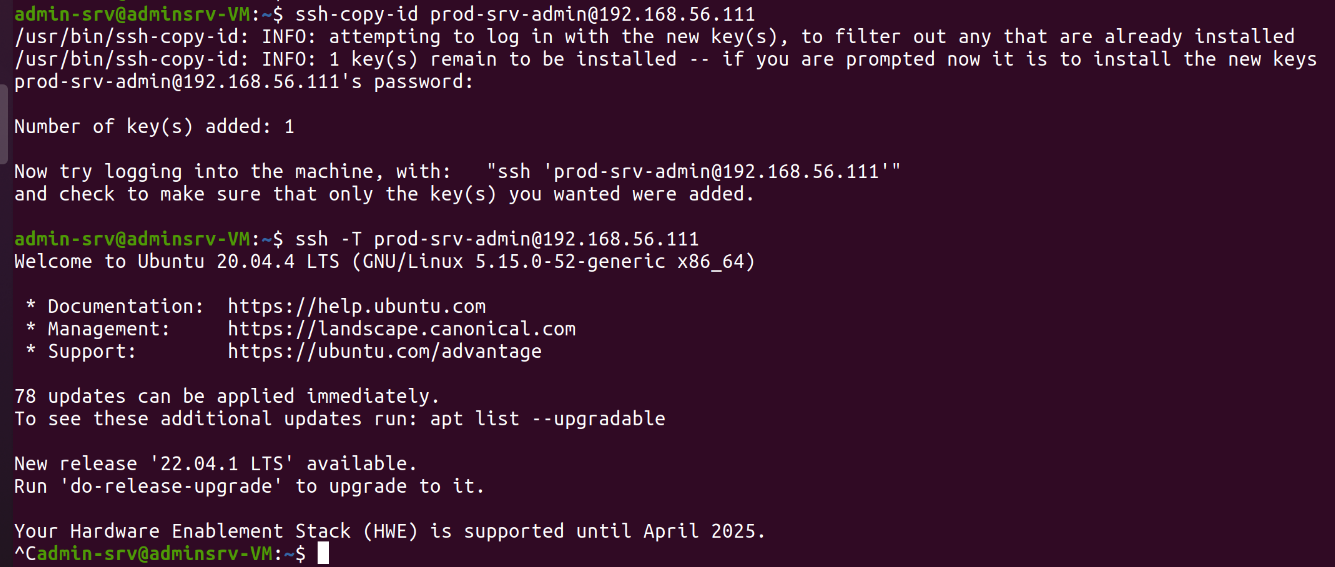


После сделанных изменений необходимо проверить, что все настройки сделаны корректно с использованием команды ansible all -m ping. Если вы еще не настроили ssh между admin-srv и prod-srv, то возможна вот такая ошибка

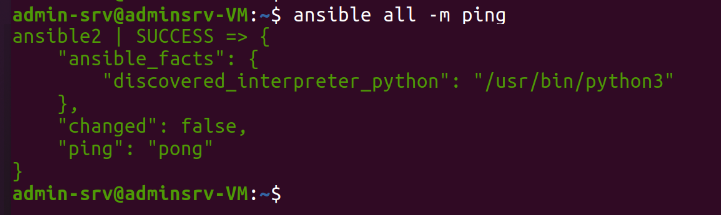


Это происходит потому, что соединение по ssh между admin-srv и prod-srv осуществляется без использования ключей, по паролю. Чтобы решить эту проблему необходимо передать публичный ключ с сервера admin-srv на сервер prod-srv командой

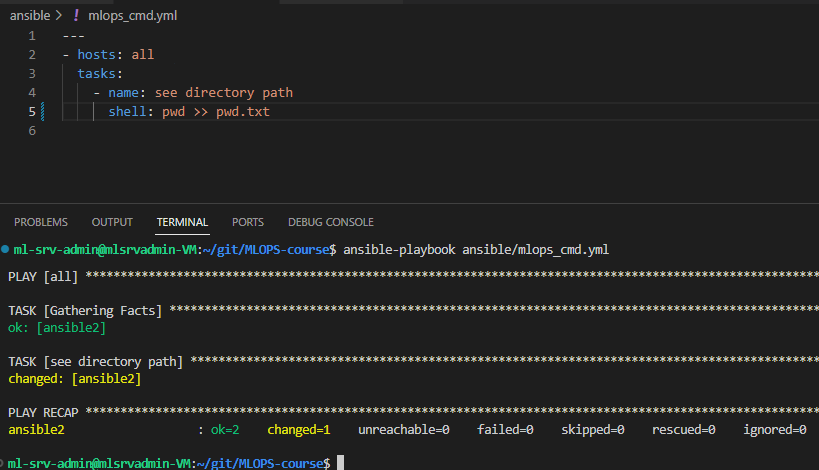
**ssh-copy-id prod-srv-admin@192.168.56.111**



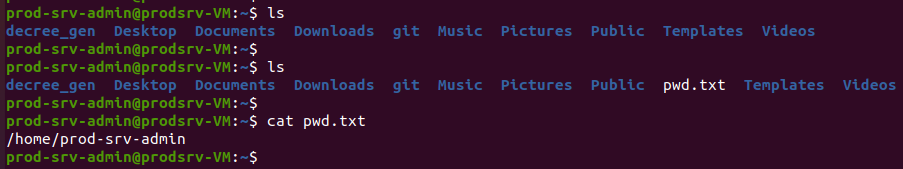
После этого проверка настройки ansible проходит успешно



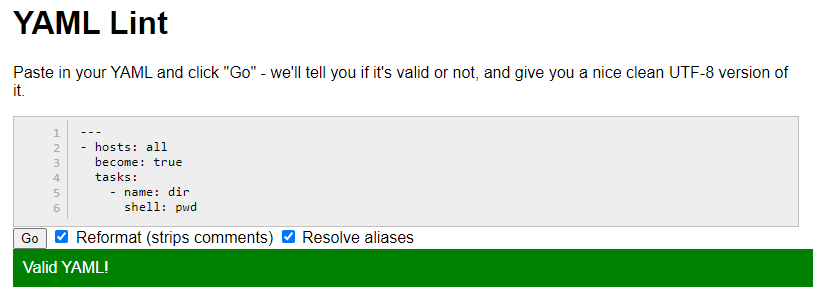
Далее создадим файл ansible плейбука mlops\_cmd.yml, в котором укажем шаги, которые будут исполняться на удаленном сервере, например, давайте выведем результат выполнения команды pwd в текстовый файл pwd.txt с помощью вот такого плейбука:



В результате выполнения плейбука в рабочей директории пользователя prod-srv-admin на сервере prod-srv появится текстовый файл pwd.txt



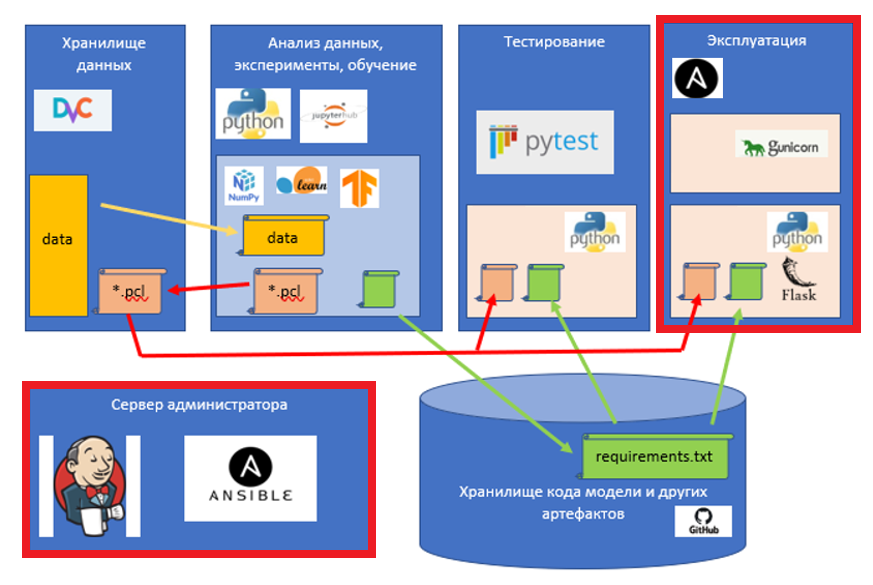
Плейбук ansible описывается в формате yaml (yet another markup language), в котором можно совершить ошибки, если недостаточно опыта работы с этим форматом. Однако существует достаточно много инструментов как в виде специальных утилит, так и облачных сервисов, которые позволяют проверить синтаксис файла плейбука, минимизируя потери времени на поиск и исправление ошибок. Один из таких сервисов доступен по адресу <https://www.yamllint.com>,



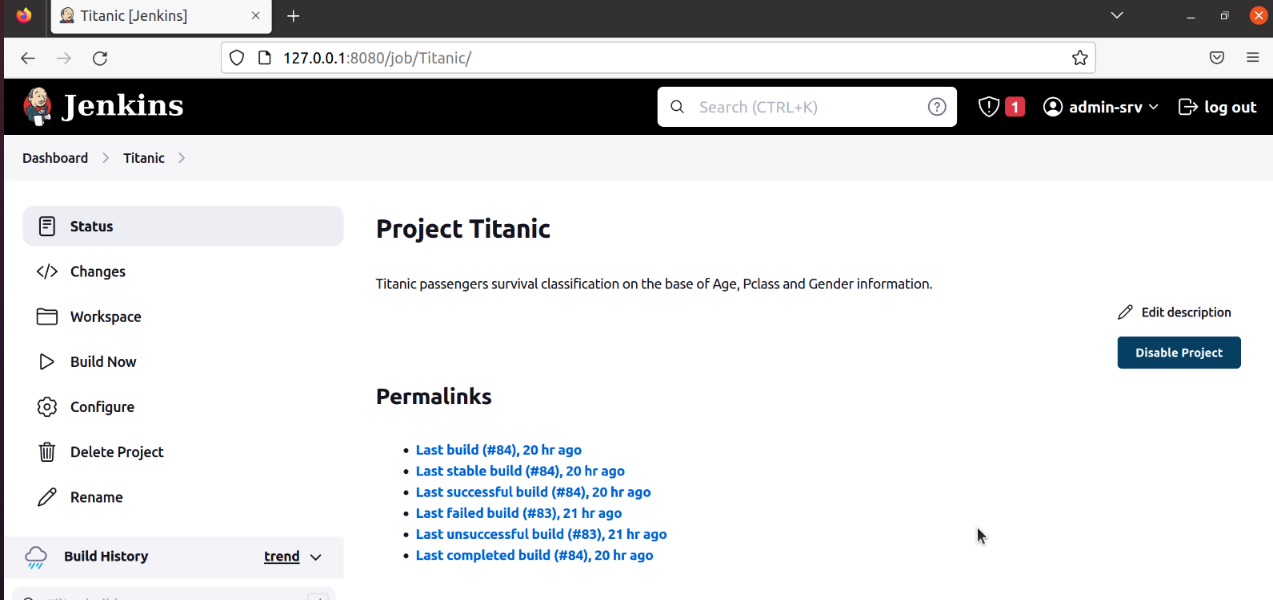
Итак, вы настроили ansible на сервере администрирования admin-srv, настроили соединения ssh между сервером администрирования admin-srv и управляемым удаленным сервером prod-srv, на котором будет эксплуатироваться модель машинного обучения, создали учебный ansible плейбук, запускаемый на выполнение на сервере администрирования admin-srv командой

**ansible-playbook mlops\_cmd.yml**

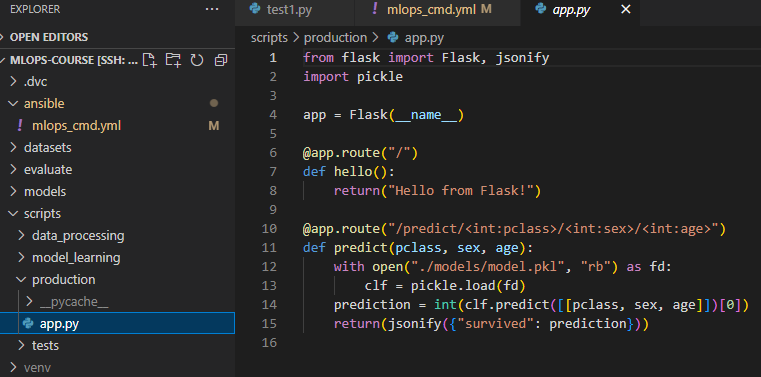
Теперь все готово для того, чтобы создать плейбук ansible, запускающий выполнение разработанной системы машинного обучения в виде web сервиса на удаленном сервере эксплуатации prod-srv, в соответствии с целевой схемой создаваемой вами инфраструктуры



Соответствующие изменения надо внести в созданную в предыдущем юните задачу (job), для этого в главном меню Jenkins надо выбрать соответствующую задачу, в нашем случае это задача “Titanic”, и открыть графический интерфейс для ее редактирования.

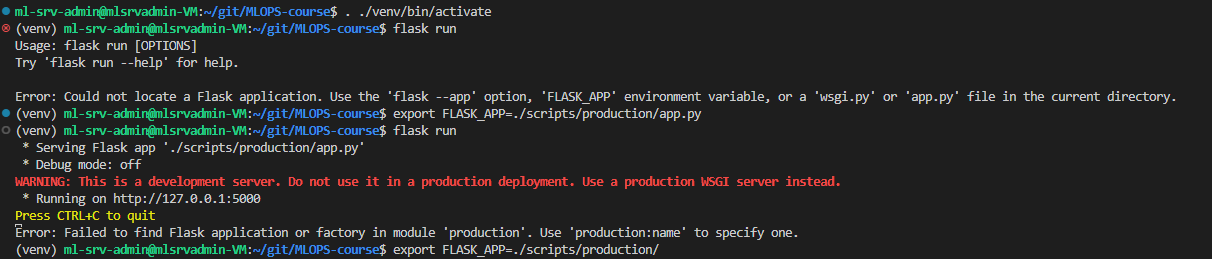


Ранее в юните 2 текущего модуля вы создали скрипт /scripts/production.app.py для работы flask сервера, запускающего web сервис.



Вспомним, что для запуска web сервиса необходимо выполнить команду

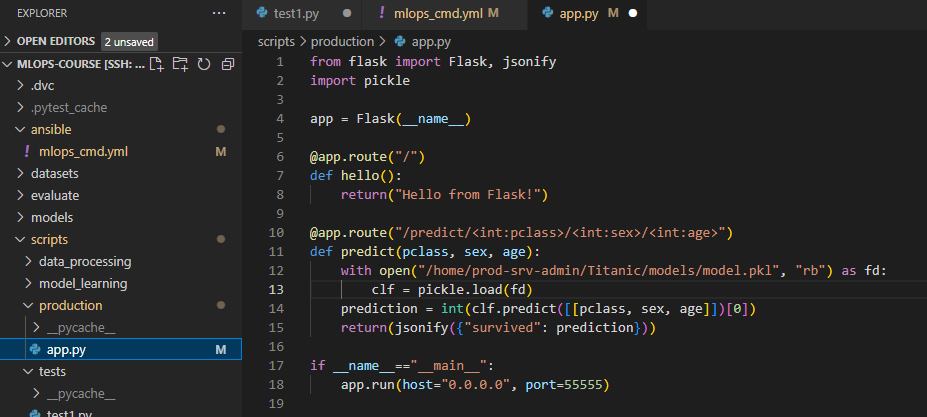
**flask run**



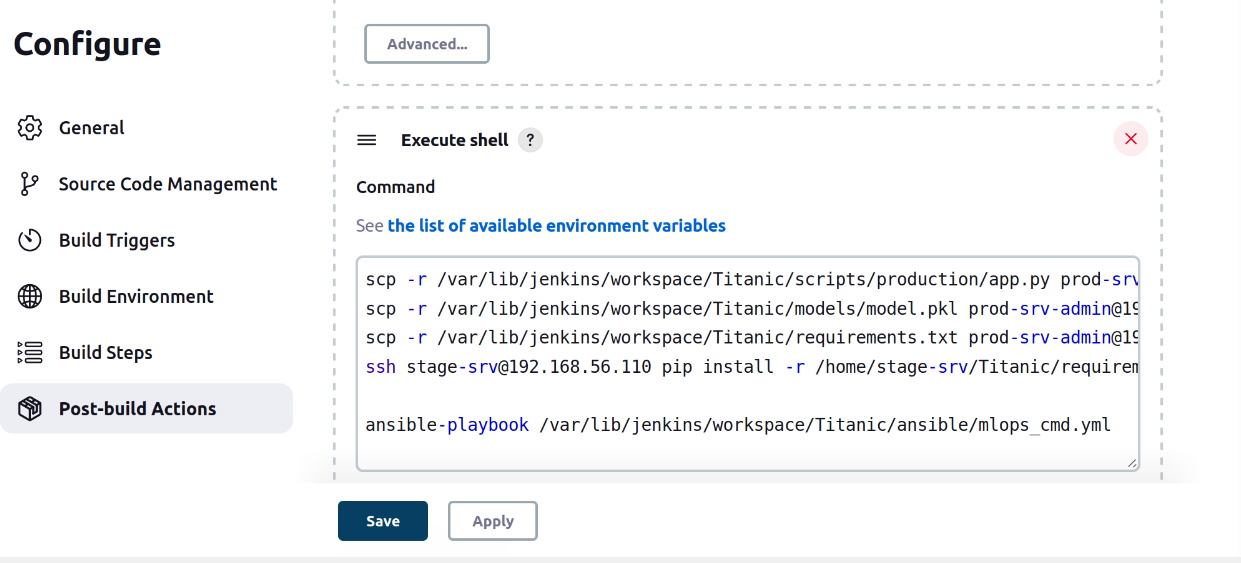
При этом надо установить значение переменной окружения FLASK\_APP. Также можно указать на каком конкретном порте будет запущен web сервис, значение по умолчанию 5000

**flask run -h 0.0.0.0 -p 11111**

Другая возможность заключается в том, что можно настройки flask устанавливать непосредственно при запуске python скрипта, написав соответствующий код в скрипте. Далее будет использоваться app.py скрипт следующего вида



В Jenkins необходимо добавить код, который переносит все необходимые файлы на сервер prod-srv и запускает выполнение web сервера flask.



Т.е. последним этапом в сборке необходимо добавить следующий код

**scp -r /var/lib/jenkins/workspace/Titanic/scripts/production/app.py prod-srv-admin@192.168.56.111:~/Titanic/scripts/production/app.py**

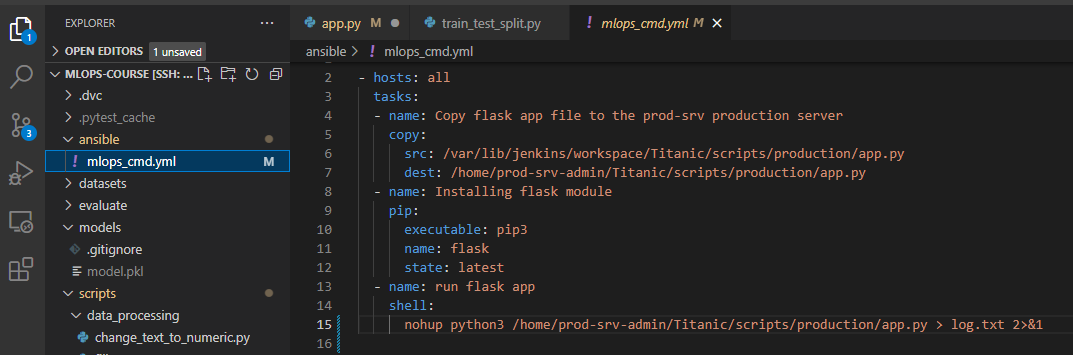
**scp -r /var/lib/jenkins/workspace/Titanic/models/model.pkl prod-srv-admin@192.168.56.111:~/Titanic/models/model.pkl**

**scp -r /var/lib/jenkins/workspace/Titanic/requirements.txt prod-srv-admin@192.168.56.111:~/Titanic/requirements.txt**

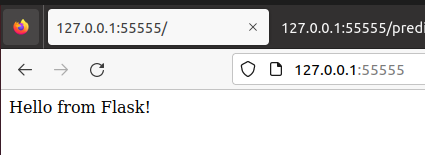
**ssh stage-srv@192.168.56.111 pip install -r /home/stage-srv/Titanic/requirements.txt**

**ansible-playbook /var/lib/jenkins/workspace/Titanic/ansible/mlops\_cmd.yml**

Плейбук ansible имеет вид



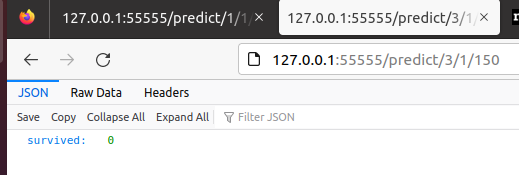
Выполнение ansible плейбука приводит к тому, что запускается web cервис на виртуальном сервере prod-srv на номере порта 55555. Можно зайти через браузер на этот порт и увидеть подтверждение работы flask сервера

****

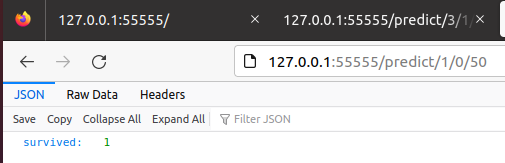
Теперь можно передать параметры Pclass, Sex, Age через url в таком виде

127.0.0.1:55555/predict/<Pclass>/<Sex>/<Age>

Например, вероятность выживания 150-летнего пассажира мужского пола, который путешествует третьим классом, очень низкая



А предсказание вероятности выживания 50-летней женщины из первого класса обслуживания – положительное.



Вы реализовали простой конвейер автоматизации работы с моделью машинного обучения, в том числе автоматизированное тестирование и запуск модели в эксплуатацию в виде flask сервера.

Тест

1. Что необходимо сделать, чтобы подключение ansible к удаленному администрируемому серверу по ssh осуществлялось без пароля? (0.25)
   1. **передать на удаленный сервер файл публичного ключа ssh, сгенерированный на сервере администрирования ansible**
   2. использовать при установлении соединения по ssh служебный параметр –without-passwd
   3. запускать выполнение ansible от пользователя root
   4. использовать только последние версии программного обеспечения ansible, ssh и python
2. Какие файлы служат для настройки параметров работы ansible? (0.25)
   1. **hosts**
   2. ansible.ini
   3. **ansible.cfg**
   4. readme
3. Как называется файл, который используется в ansible для описания последовательности команд, выполняемых на удаленных машинах? (0.25)
   1. скрипт
   2. **плейбук**
   3. сценарий
   4. инструкция
4. Какой фреймворк использовался для создания web сервиса? (0.25)
   1. **flask**
   2. django
   3. node.js
   4. fastapi

## Итоги/выводы

В этом юните с использованием средства автоматизации ansible осуществили настройку сервера эксплуатации и развертывание на нем web сервиса, использующего обученную вами модель машинного обучения.

# Итоги/выводы по модулю

В этом модуле мы решали задачу: создать инфраструктуру для небольшого проекта машинного обучения, включающую инструменты для работы с данными и проведения исследований, средства автоматизации, создание производственного окружения и запуск модели в эксплуатацию. Эта общая задача была решена с помощью отдельных частных подзадач:

1. развертывание виртуальных машин,
2. установка инструментов для участников команды
   1. python как основной язык программирования
   2. инструменты для работы с виртуальными окружениями
   3. VSCode для разработчиков
   4. Pytest для автоматизации тестирования
   5. Ansible для вывода модели в эксплуатацию
3. создание инфраструктуры для хранения и контроля версия для данных и программного кода, настройка git и dvc
4. организация управления операциями с использованием Jenkins
5. автоматизация тестирования с использованием pytest
6. автоматизация развертывания с использованием скриптов ansible

# Список источников

Образовательные и обзорные статьи, справочники, FAQ

1. «Как настроить Ansible в связке с Jenkins»

https://habr.com/ru/company/southbridge/blog/585776/

1. Статья о практическом использовании ansible https://habr.com/ru/post/488966
2. «Установка ssh в ubuntu» <https://losst.ru/ustanovka-ssh-ubuntu-16-04>
3. «Совместное использование данных и моделей с использованием dvc» (видеолекция) <https://www.youtube.com/watch?v=EE7Gk84OZY8>
4. «Git и Github для Data Science (основы для новичков) и DVC» (видеолекция)

<https://www.youtube.com/watch?v=6_hxth-sr9Q>

1. Описание формата yaml <https://ru.wikipedia.org/wiki/YAML>
2. Yaml валидатор <https://www.yamllint.com/>
3. https://medium.com/clarusway/running-flask-web-server-on-virtualbox-using-ansible-and-vagrant-d2b92c6d4075

Официальные страницы программных продуктов, инструментов, утилит

1. [https://www.virtualbox.org](https://www.virtualbox.org/wiki/Downloads)
2. <https://www.jenkins.io/>
3. https://www.ansible.com/
4. <https://dvc.org>
5. <https://python.org>
6. <https://ubuntu.com>
7. <https://www.docker.com>

Репозитории и источники, использованные в модуле для практической работы

1. пример репозитория проекта <https://github.com/CyberLympha>