**Введение**

Использовать готовый хостинг как GitHub Pages — удобно, но не всегда эффективно. Хотя бы потому, что репозиторий с исходным кодом необходимо делать публичным — тогда и задеплоенное на Pages приложение будет доступно всем пользователям. Ещё один важный недостаток — нельзя сделать модное доменное имя. Поэтому в этой теме разберём, как настроить арендованный удалённый сервер и разместить на нём код вашего приложения Stellar Burger.

Мы затронем базовые концепции работы с удалённым сервером: настройку удалённого сервера, размещение на нём фронтенд-части приложения и прикрепление доменного имени. А ещё рассмотрим темы SSL-сертификатов и nginx. Эти знания пригодятся вам в будущем и помогут при выполнении проектной работы.

# Создаём удалённый сервер

Существует несколько вариантов, где разместить код приложения и сделать его доступным другим пользователям. Первый — купить железо и поставить сервер у себя в гараже. Второй — использовать бесплатный сервис, например, GitHub Pages (о нём вы узнали в предыдущей теме) или Heroku.



Гилфойл бы одобрил

Третья опция — аренда облачного сервера. Это готовый сервер, который можно создать и сконфигурировать под себя за несколько минут. При этом его поддержкой, настройкой и безопасностью занимается компания, которая сдаёт этот сервер в аренду. Вот несколько популярных решений облачных серверов:

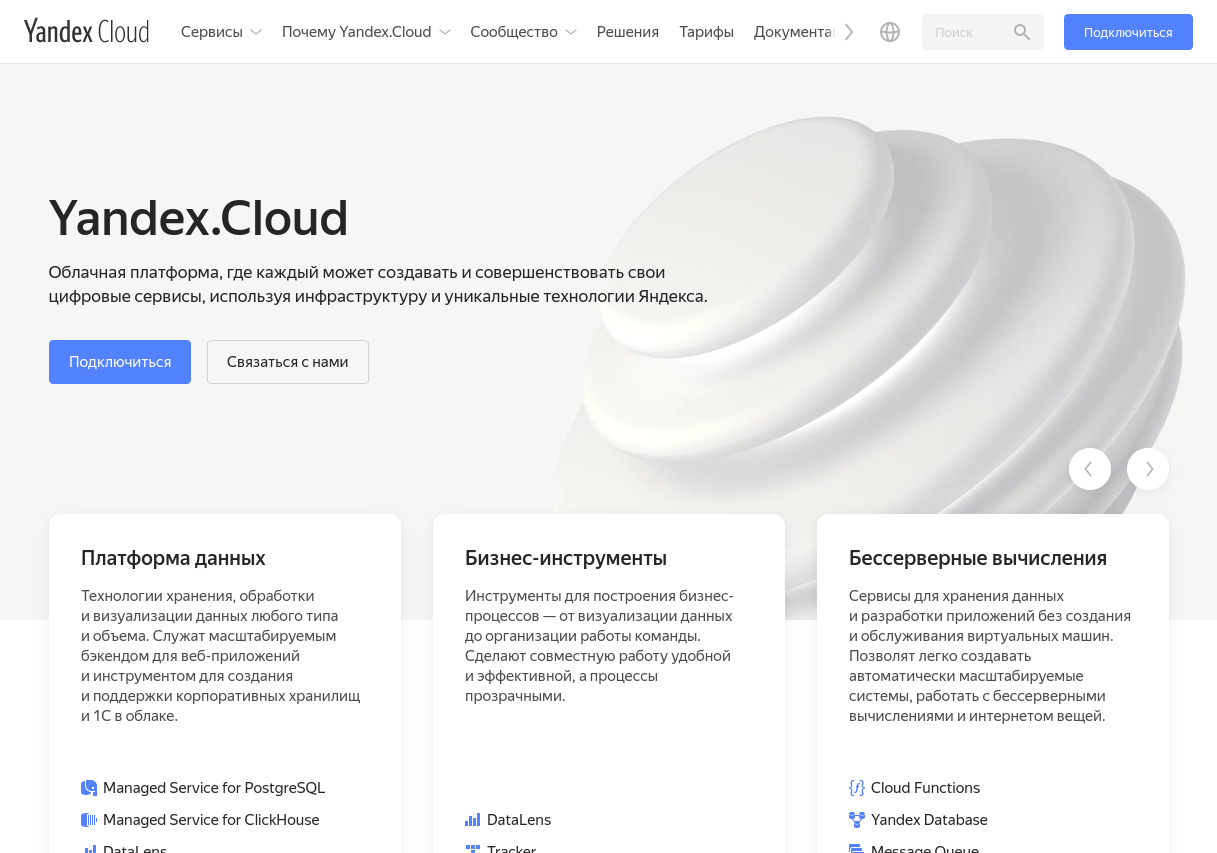
* Amazon Web Services,
* Google Compute Cloud,
* Яндекс.Облако.

Мы воспользуемся последним по двум причинам:

* Яндекс.Облако предоставляет грант на бесплатное использование сервиса. Этот грант полностью покроет нужды нашего обучения.
* Сервера Яндекс.Облака соответствуют [закону о персональных данных](http://www.consultant.ru/document/cons_doc_LAW_61801/), и нам не придётся об этом думать.

## Создайте аккаунт в Яндекс.Облаке

Зайдите на [https://cloud.yandex.ru](https://cloud.yandex.ru/) и залогиньтесь через аккаунт Яндекса. Затем [войдите в консоль](https://console.cloud.yandex.ru/):



После регистрации вы можете активировать пробный период. На это время Яндекс.Облако предоставляет вам грант на 4000 рублей. Он действует 60 дней и разбит на 2 части: 1000 и 3000 рублей. 1000 рублей можно потратить на создание виртуальных машин, а остальные 3000 — на другие сервисы облака.

Для деплоя будем использовать виртуальные машины Яндекс.Облака, то есть на наши нужды есть 1000 рублей стартового гранта. Остальным можете распоряжаться на ваше усмотрение.

Подробная информация о предоставлении гранта: <https://cloud.yandex.ru/docs/free-trial/concepts/usage-grant>.

Что делать, если грант закончился, а обучение ещё продолжается?

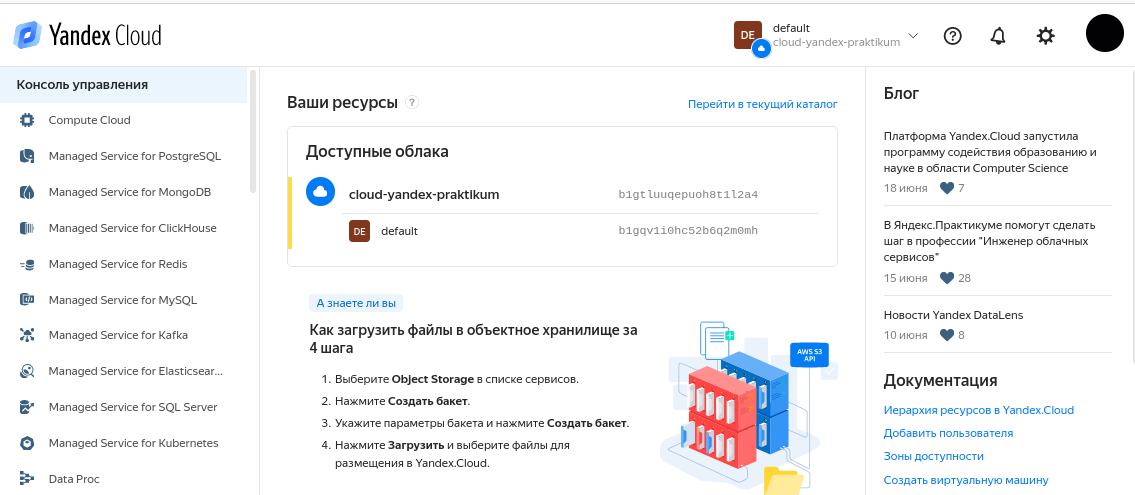
## Создайте платёжный аккаунт

Чтобы пользоваться Яндекс.Облаком, нужно создать платёжный аккаунт. Для этого перейдите [в раздел «Billing»](https://console.cloud.yandex.ru/billing) и нажмите «Создать аккаунт». В открывшемся окне введите информацию о себе, данные карты и нажмите «Активировать».

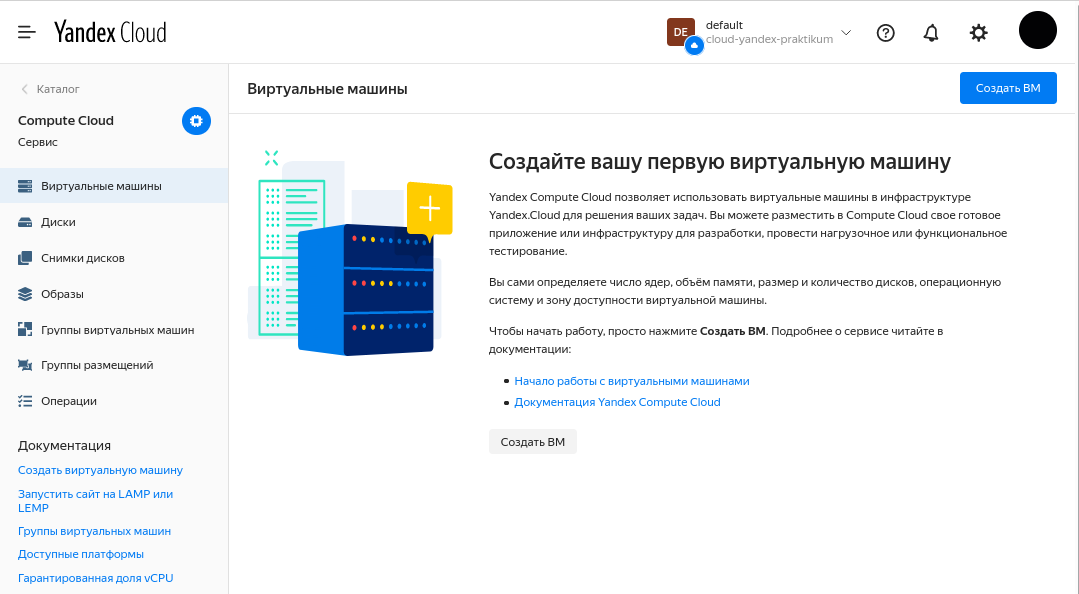
Мы понимаем скепсис по поводу ввода данных карты. Но для нас это необходимо, чтобы создать сервер и задеплоить на него приложение. Стартового гранта в 1000 рублей хватит примерно на 2 месяца, этого более чем достаточно. Дальше вы можете остаться в Облаке или выбрать другую платформу.

## Создайте виртуальную машину

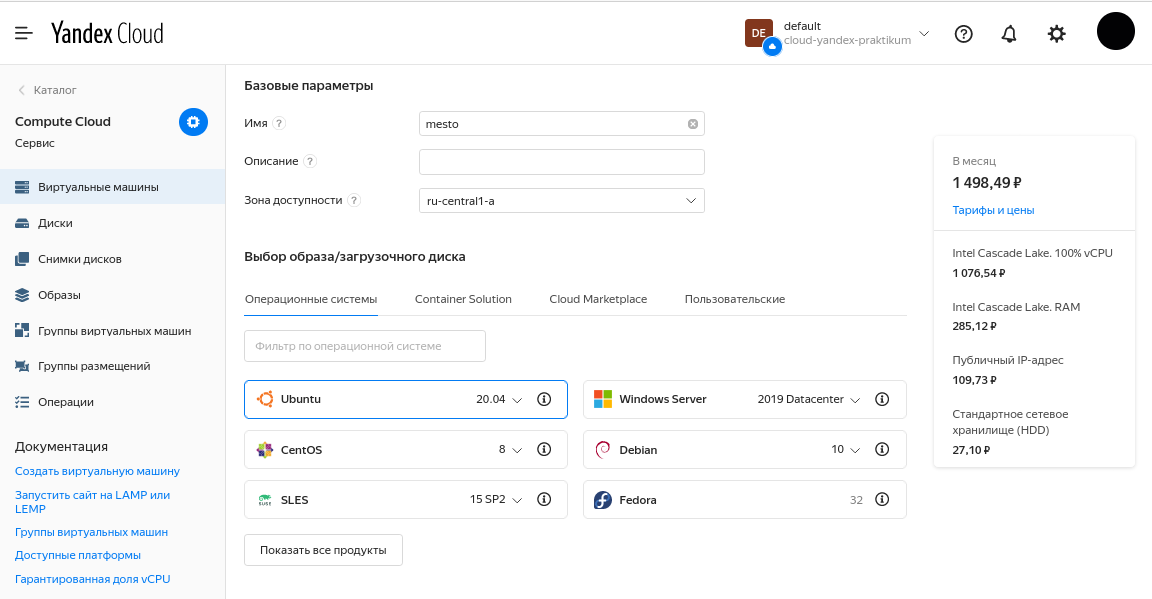
Чтобы начать создавать виртуальную машину, зайдите [в консоль Яндекс.Облака](https://console.cloud.yandex.ru/). Вам понадобится сервис Compute Cloud. В левом меню кликните на него:



Теперь нужно создать виртуальную машину. Кликните на «Создать ВМ»:



Введите название проекта на ваш выбор. В качестве публичного образа выберите Ubuntu 20.04:

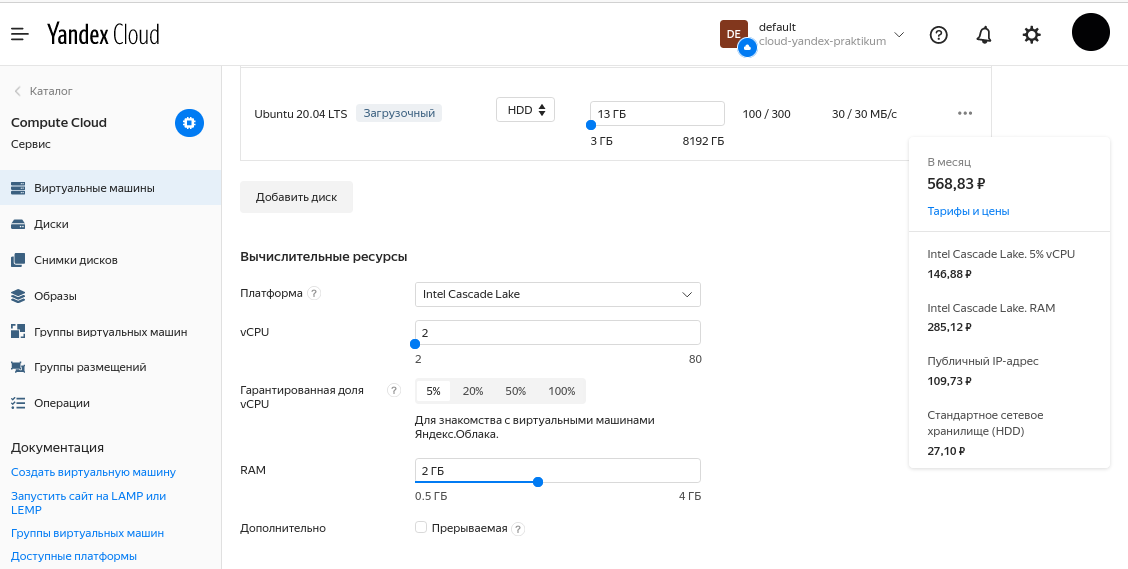


Дискам и вычислительным ресурсам оставьте настройки по умолчанию, для начала этого хватит. А вот для параметра «Гарантированная доля vCPU» необходимо проставить значение 5%.

vCPU — облачный процессор вашего сервера, а его гарантированная доля — минимальный процент мощности. Это не значит, что у сервера всегда будет мощность в 5% от полной — нередко она будет близка к максимальной по тарифу. Показатель говорит лишь о том, что в момент, когда ресурсы нашего сервера не используются на полную, Яндекс.Облако может предоставить их другому клиенту.

Это бывает нежелательно при использовании в продакшене: нагрузка на сервис может резко возрасти, и если Яндекс.Облако предоставит ресурсы с задержкой, сервис начнёт тормозить у клиентов. Но для учебных целей и экспериментальных проектов такой подход удобен: большую часть времени сервер будет использоваться только нами и можно заметно сэкономить, если не требовать у Облака полностью держать ресурсы под сервер.

После этого изменения стоимость аренды сервиса заметно снизится:



Было почти 1500 рублей, стало — чуть больше 550 рублей.

Ваши настройки должны совпадать с теми, которые указаны на скриншоте. Если что-то отличается, приведите их в соответствие c ним.

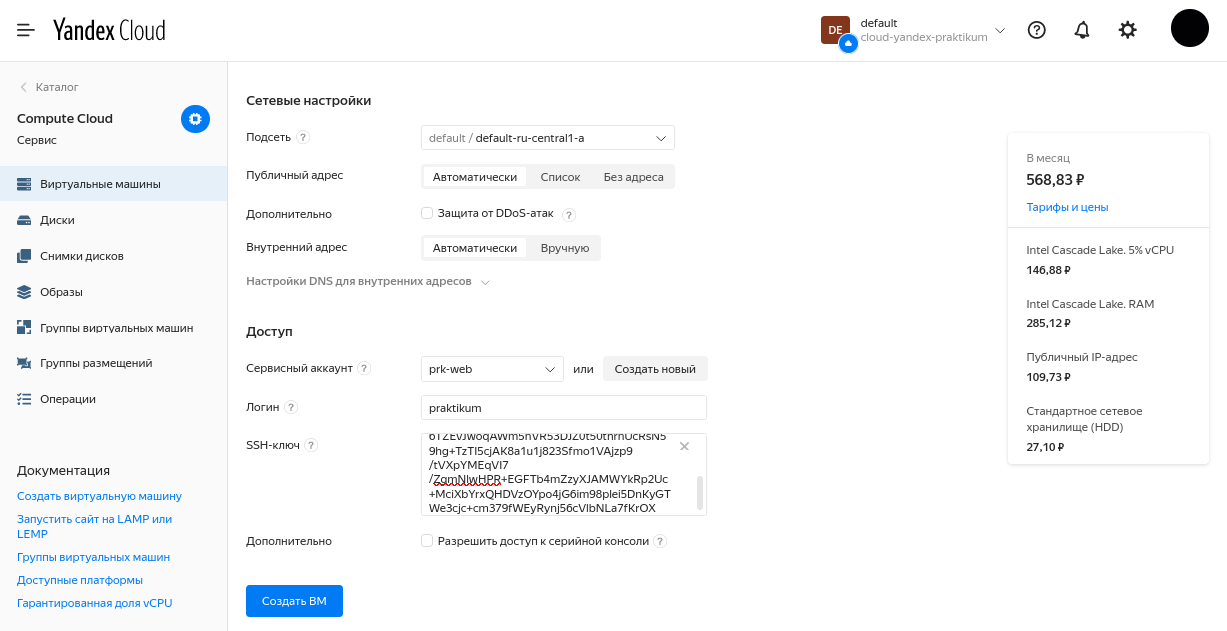
В блоке «Доступ» в поле «Логин» введите имя пользователя, по которому будете заходить на сервер. Оно может быть произвольным, но должно соответствовать правилам, которые можно прочитать, если навести курсор на знак вопроса слева от поля.

У сервера может быть несколько пользователей с разными паролями, ключами доступа и правами. Когда добавляется новый пользователь, для него автоматически создаётся специальная домашняя директория, которая предназначена для хранения личных файлов и конфигов.

Обычно она располагается по пути /home/<имя\_пользователя>. Для домашней директории также придумали специальное сокращение — ~ (тильда), которое можно использовать вместо полного пути.

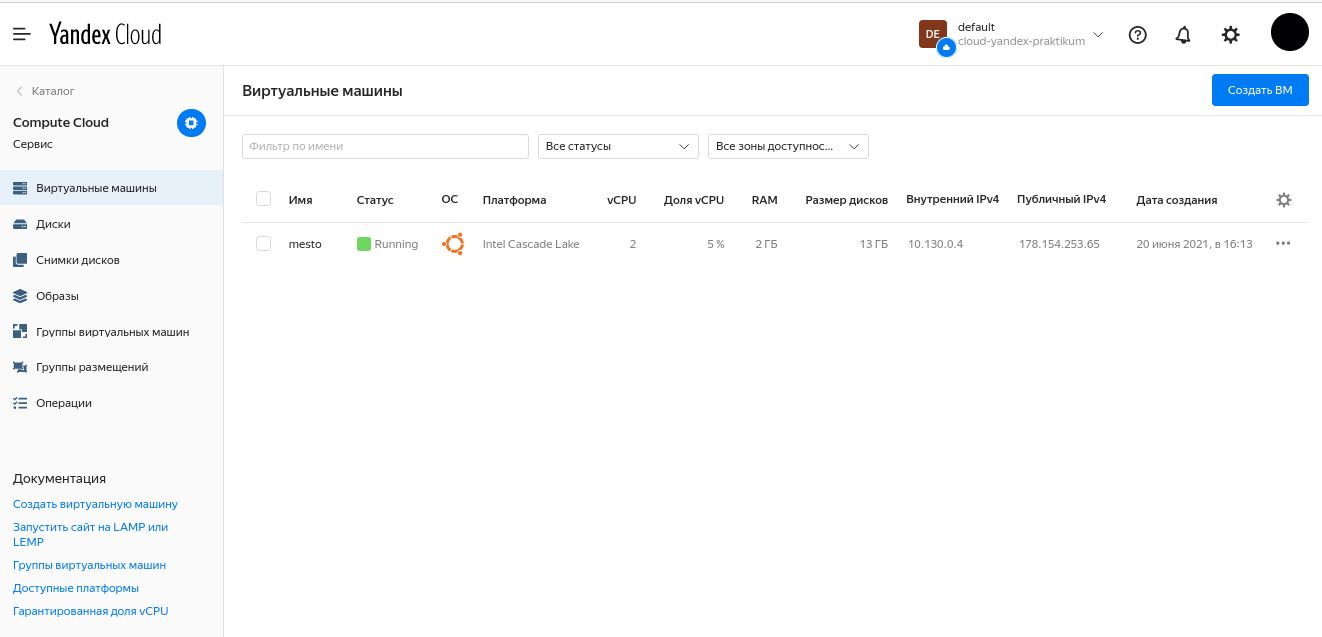
Во время выполнения проектной работы вам потребуется разместить код в домашней папке, поэтому мы ещё вернёмся к этой теме.

В поле «SSH-ключ» скопируйте ваш SSH-ключ, он понадобится для подключения к серверу:



Важно: SSH-ключ, который вы указываете в Яндекс.Облаке, должен совпадать с ключом доступа в GitHub-репозиторий с проектом.

После того как всё сделали, нажмите «Создать BM». Облаку понадобится какое-то время, чтобы создать машину. Примерно через минуту статус изменится на “Running” — это значит, что машина создана:



## Подключение к серверу

Чтобы подключиться к серверу, понадобится публичный IP и логин, который мы указали при создании виртуальной машины.

Скопируйте публичный IP, зайдите в командную строку и введите команду (замените praktikum и 84.201.130.70 на свои логин и IP):

Скопировать кодBASH

ssh praktikum@84.201.130.70

Если это первое подключение к серверу, вы увидите фразу:

Скопировать кодBASH

The authenticity of host '84.201.130.70 (84.201.130.70)' can't be established.

ECDSA key fingerprint is SHA256:gGz1AULJpNptRRaqLz2FQTDf/IRxSGPA0vvmmXWy/6I.

Are you sure you want to continue connecting (yes/no)?

Введите “yes” и нажмите Enter. Вам будет предложено ввести пароль от приватного ключа. Введите его и снова нажмите Enter. Если пароль правильный, вы окажетесь на удалённом сервере:



Отлично, вы на сервере! Освойтесь: введите несколько команд. А в следующем уроке мы установим всё необходимое для работы нашего API. Чтобы выйти из сервера, воспользуйтесь сочетанием клавиш Ctrl+D.

# Как предоставить SSH-доступ другому разработчику

Мы часто сталкиваемся с необходимостью предоставить доступ к нашему серверу другому разработчику. Например, когда работаем в команде и хотим, чтобы каждый из её участников мог управлять приложением, читать логи или конфигурировать сервер.

Как вы знаете из предыдущих уроков, для удалённого управления сервером разработчики используют протокол SSH с аутентификацией при помощи ключей. В этом уроке научимся управлять SSH-ключами и разберёмся, как реализуется аутентификация с их использованием.

## Узнаём публичный SSH-ключ

Для начала следует узнать публичный SSH-ключ пользователя, которому мы предоставляем доступ. Ключи обычно хранятся в домашней папке пользователя директории .ssh. Для публичных ключей используются файлы с расширением .pub (по умолчанию это id\_rsa.pub).

Если владелец не менял стандартное имя ключа, его можно узнать с помощью такой команды:

Скопировать кодJSX

cat ~/.ssh/id\_rsa.pub

После этого ключ следует сохранить в удобное место в файл your-new-key.pub.

Второй шаг — прописать SSH-ключ на удалённом сервере. Это можно сделать двумя способами: при помощи утилиты ssh-copy-id или вручную, путём прописывания ключа в конфиге.

## Добавляем SSH-ключ при помощи ssh-copy-id

ssh-copy-id — небольшая утилита, единственное назначение которой — прописывать ключ на удалённом сервере. Добавить ключ с её использованием можно так:

Скопировать кодJSX

ssh-copy-id -i <путь-до-ключа>/your-new-key.pub <имя\_пользователя>@<ip\_вашего сервера>

Здесь мы указываем путь до файла с ключом your-new-key.pub, имя пользователя и IP удалённого сервера.

Но у этой утилиты есть недостаток: чтобы добавить новый ключ к серверу, нужно подключиться с использованием уже добавленного на него ключа. Если это ваш первый ключ, следует воспользоваться вторым решением — добавить его вручную.

## Добавляем SSH-ключ вручную

Это может быть полезно в случаях, когда невозможно авторизоваться под пользователем, для которого добавляется ключ. Например, если вы только что добавили нового пользователя на сервере.

Перед началом операции скопируйте публичный ключ в буфер обмена, он нам ещё пригодится.

Чтобы добавить SSH-ключ на удалённый сервер, проделаем такие операции:

1. Подключимся к удалённому серверу при помощи SSH:

Скопировать кодJSX

ssh praktikum@<ip\_сервера>

1. Перейдём в домашнюю директорию пользователя, для которого необходимо добавить ключ. По умолчанию это /home/something, где something — имя учётной записи.

Если нужно добавить ключ текущему пользователю, можно использовать в пути символ ~ (тильда), который укажет на домашнюю директорию текущего пользователя. Перейдём в неё при помощи такой команды:

Скопировать кодJSX

cd ~

1. В случае, если мы добавляем первый ключ для данной учётной записи, нам потребуется создать соответствующие конфиги. Это можно сделать при помощи таких команд:

Скопировать кодJSX

# создадим директорию для конфига ssh

sudo mkdir -p -m 700 .ssh

# создадим конфиг с ключами, для которых разрешено подключение

sudo touch .ssh/authorized\_keys

# проставим для него требуемые права доступа

# заменим something на имя пользователя, для которого добавляем ключ

sudo chmod 664 .ssh/authorized\_keys

sudo chown -R something:something .ssh

Если для пользователя уже был задан SSH-ключ, этот этап можно пропустить и перейти к следующему.

1. Откроем конфигурацию SSH в редакторе при помощи команды:

Скопировать кодJSX

sudo nano ./.ssh/authorized\_keys



Интерфейс редактора nano

Вставим публичный ключ из буфера обмена и сохраним файл конфигурации нажатием комбинации Ctrl+O. Затем можно закрыть редактор, нажав Ctrl+X.

Если потребуется отозвать доступ для какого-то ключа, можно просто стереть соответствующую строчку из этого файла и снова его сохранить.

Всё готово — можно подключаться. Теперь разберёмся, каким образом работает аутентификация по ключу и как сервер нас узнаёт.

## **Как работает аутентификация по ключу**

Для управления сервером мы используем протокол SSH с аутентификацией по ключу. Этот механизм требует наличия пары ключей: приватного (privatekey) и публичного (publickey).

Приватный ключ сохраняется на стороне пользователя и не должен быть доступен кому-то ещё. Его утечка позволит злоумышленнику войти на сервер и выполнить действия от имени владельца ключа, поэтому при обращении с ним нужно быть очень осторожным.

Приватный ключ выглядит так:

Скопировать кодJSX

----BEGIN OPENSSH PRIVATE KEY-----

xxxx

-----END OPENSSH PRIVATE KEY-----

Он состоит из разделителей, которые помечают начало и конец ключа, и ключа шифрования между ними.

Публичный, или открытый ключ используется для шифрования сообщений, которые можно расшифровать только приватным ключом. Он генерируются автоматически при создании приватного.

Публичный ключ выглядит так:

Скопировать кодJSX

ssh-rsa xxxx denis@your-pc-name

Сначала указывается алгоритм шифрования, в нашем случае — RSA. Затем — сама ключевая последовательность. В конце записывается текстовый комментарий, который по умолчанию содержит имя пользователя и компьютера, на котором был сгенерирован этот ключ. Части в описании публичного ключа разделяются пробелом.

Запись приватного и публичного ключей выглядит по-разному, что помогает наглядно их различать. Так у разработчика меньше шансов ошибиться и скинуть не тот ключ.

Теперь, когда мы знаем какой ключ за что отвечает, разберёмся, как это работает на практике. Когда клиент пытается подключиться к серверу, последний отправляет клиенту сообщение, зашифрованное с помощью открытого ключа. Если клиент сможет расшифровать его и вернуть правильный ответ, то есть исходное сообщение, аутентификация пройдена.

Главное преимущество этого подхода — высокая безопасность. Приватный ключ не передаётся по сети и не покидает пределы компьютера пользователя, поэтому злоумышленник не может перехватить его в процессе подключения. А публичный ключ не позволяет подключиться к серверу, поэтому он бесполезен для злоумышленника.

Стойкость ключа, как и обычного пароля, зависит от длины и набора символов. Для SSH обычно используются ключи длиною в несколько тысяч символов (размер ключа в битах вы можете выбрать при его генерации), что делает его перебор «в лоб» практически бессмысленным.

## Добавляем ключ команды Практикума

Теперь, когда вы знаете, как управлять SSH-ключами, самое время закрепить это на практике — добавить ключ команды Практикума. Скопируйте ключ из блока ниже и добавьте его на сервер для вашего пользователя любым изученным способом:

Скопировать кодJSON

ssh-rsa  console@console

С помощью этого ключа вы предоставите код-ревьюеру и наставнику доступ к своей работе. Код-ревьюер проверит, что сервер настроен корректно, а наставник поможет, если возникнут трудности.

В следующем уроке настроим сервер. Поехали!

# Устанавливаем всё необходимое на сервер

В одном из предыдущих уроков вы подключились к удалённому серверу по SSH. Теперь необходимо установить на него всё, что нужно для работы API:

* Node.js, чтобы устанавливать дополнительные пакеты,
* git, чтобы загружать код на сервер.

## Установка Node.js

Сначала проверьте версию Node.js, которая установлена у вас локально. Для этого введите в командную строку команду:

Скопировать кодBASH

node -v

В командной строке появится номер версии — три числа, разделённые точками. Первое число — мажорная версия. Версия Node, которую мы установим на сервер, должна иметь такую же мажорную версию. Так инфраструктура нашего сервера будет близка к той, в которой мы разрабатывали приложение, а значит, оно будет работать без сюрпризов.

Теперь зайдите на сервер, который создали в предыдущем уроке и по очереди введите две следующие команды. В первой команде замените 14 на мажорную версию, которую узнали только что:

Скопировать кодBASH

*# в этой команде замените 14 на ту мажорную*

*# версию Node, которая стоит у вас локально*

curl -sL https://deb.nodesource.com/setup\_14.x | sudo -E bash -

*# установите Node.js*

sudo apt install -y nodejs

Первая команда добавляет официальный дистрибутив Node.js к нам в систему, а вторая его устанавливает. После того как вторая команда закончит установку, проверьте, что нужная версия Node установлена на сервере:

Скопировать кодBASH

node -v

## Установка git

Осталось установить git, он позволит легко обновлять код на сервере. Для этого выполните следующую команду:

Скопировать кодBASH

sudo apt install -y git

После установки проверьте, всё ли в порядке, с помощью команды:

Скопировать кодBASH

git --version

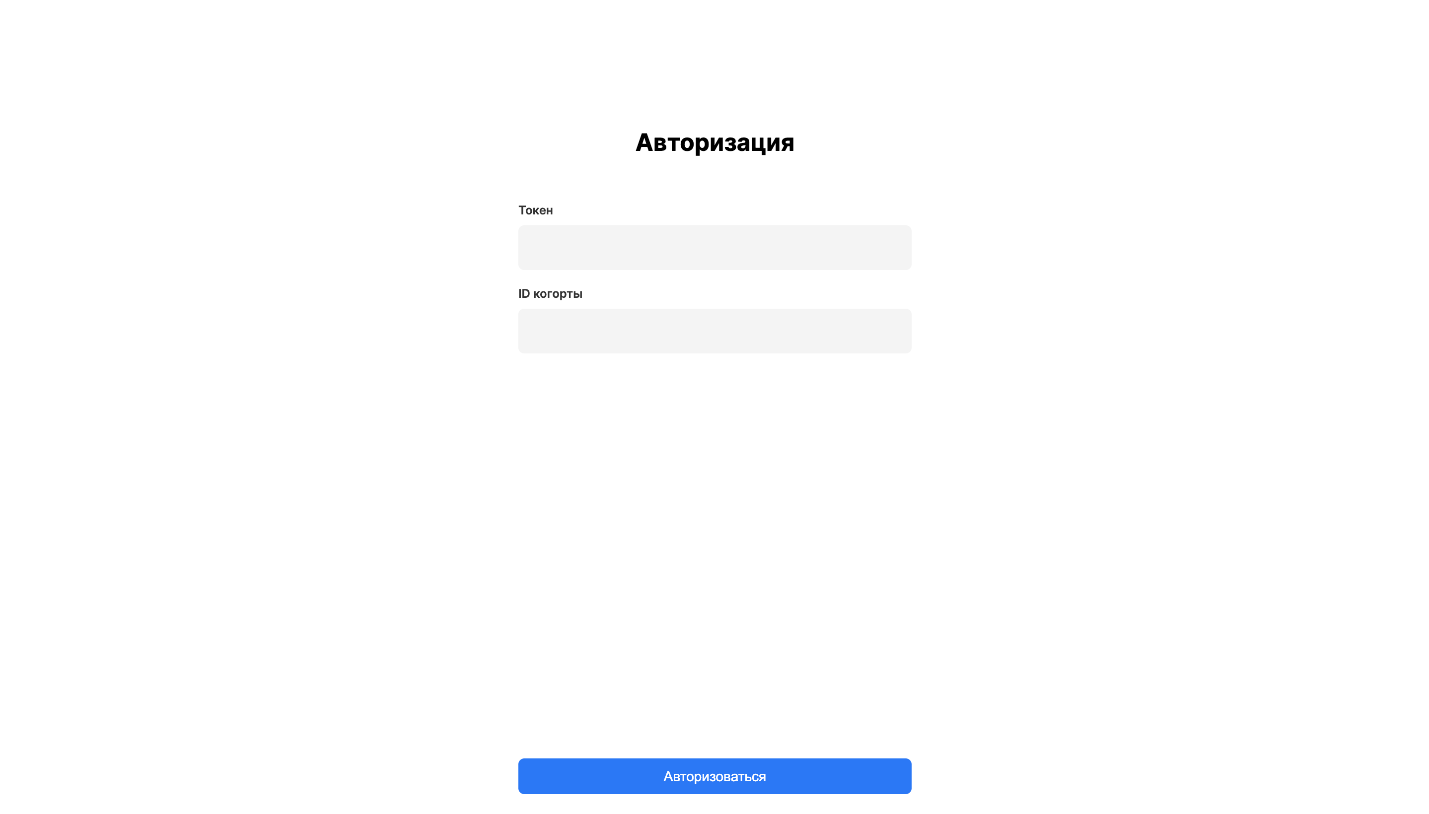
Она должна вывести на экран установленную версию git.

В следующем уроке разберёмся, как сделать так, чтобы при выходе приложение продолжало работать.

# Прикрепляем доменное имя

Пока мы можем обратиться к серверу только по IP-адресу, но в реальном мире обычно используется доменное имя. В этом уроке вы создадите его и прикрепите к IP-адресу сервера.

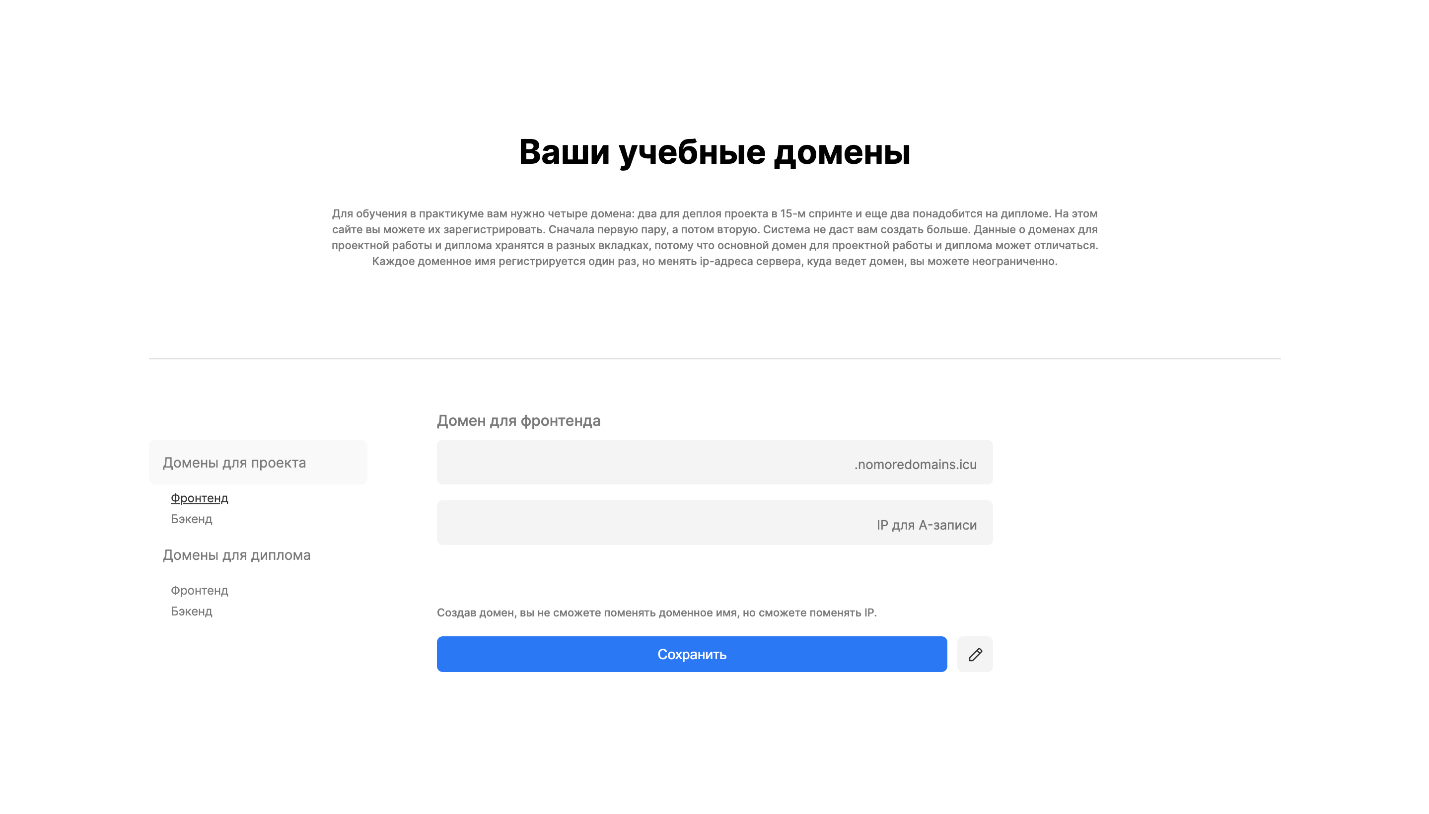
Обычно создание домена — платная услуга. А пока учитесь — [воспользуйтесь нашим сервисом по ссылке бесплатно](https://domain.nomoreparties.site/). Для начала авторизуйтесь по токену и номеру когорты:



Интерфейс авторизации перед созданием доменов

Всю необходимую информацию пришлёт вам куратор.

После успешной авторизации откроется интерфейс с доменами:



Интерфейс со списком доменов

Сначала нужно указать доменное имя, которое вы хотели бы получить. Будем регистрировать домен для фронтенда, поэтому выберите вкладку «Фронтенд» в «Домены для проекта». В первое полу введите желаемое имя. Домен может иметь такой вид:

Скопировать код

domainname.students

another.domainname.students

Домен .nomoredomains.icu со скриншота выше автоматически добавится к имени домена. Другими словами, если вы введёте в поле domainname.students , будет создан такой домен:

Скопировать кодBASH

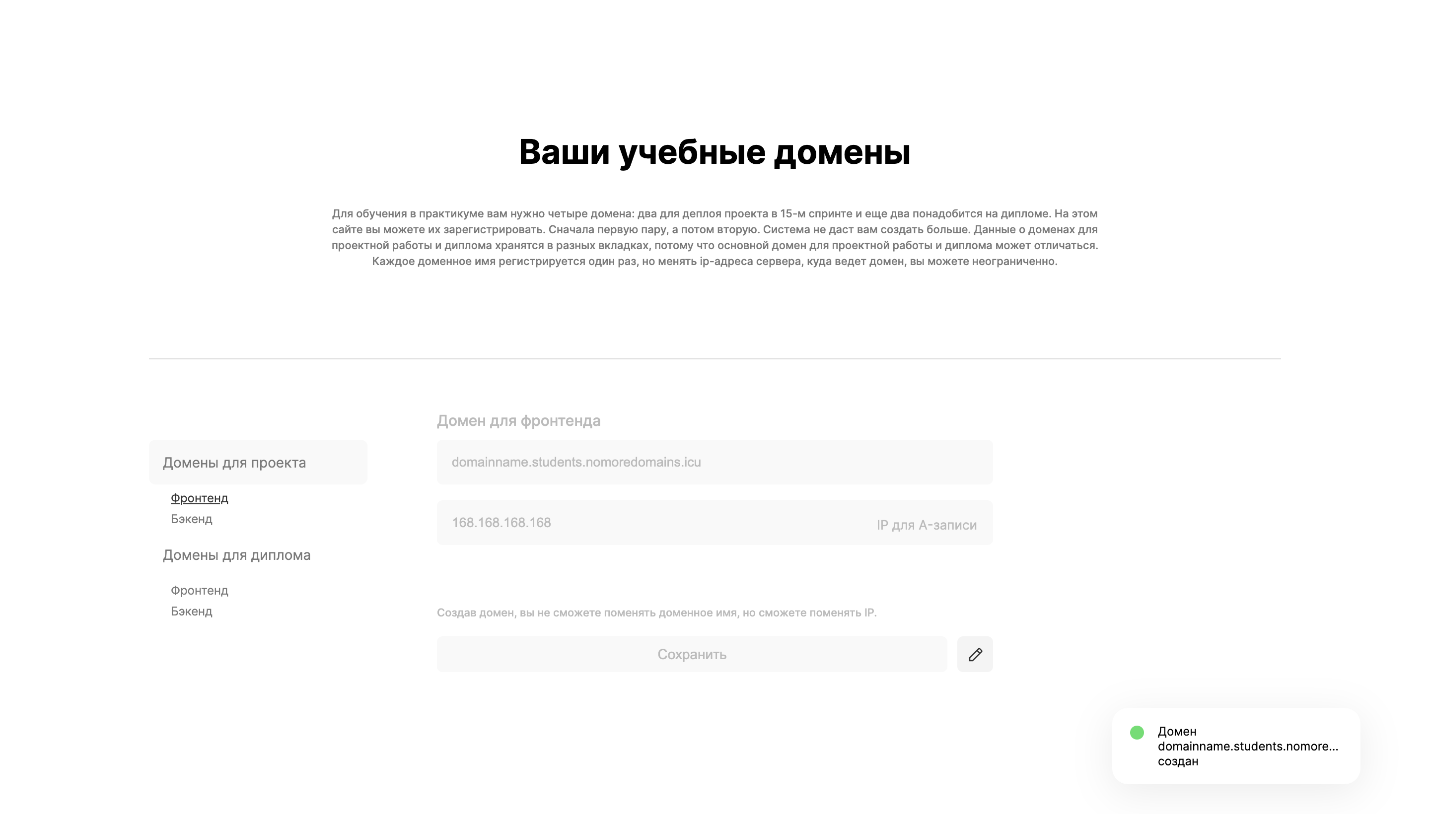
domainname.students.nomoredomains.icu

Следующим шагом нужно создать привязку домена и вашего сервера в Яндекс.Облаке. Это можно сделать с помощью A-записи. Она связывает домен и сервер по публичному IP-адресу. Введите публичный IP-адрес сервера в Яндекс.Облаке во второе поле. Используйте такой формат записи:

Скопировать код

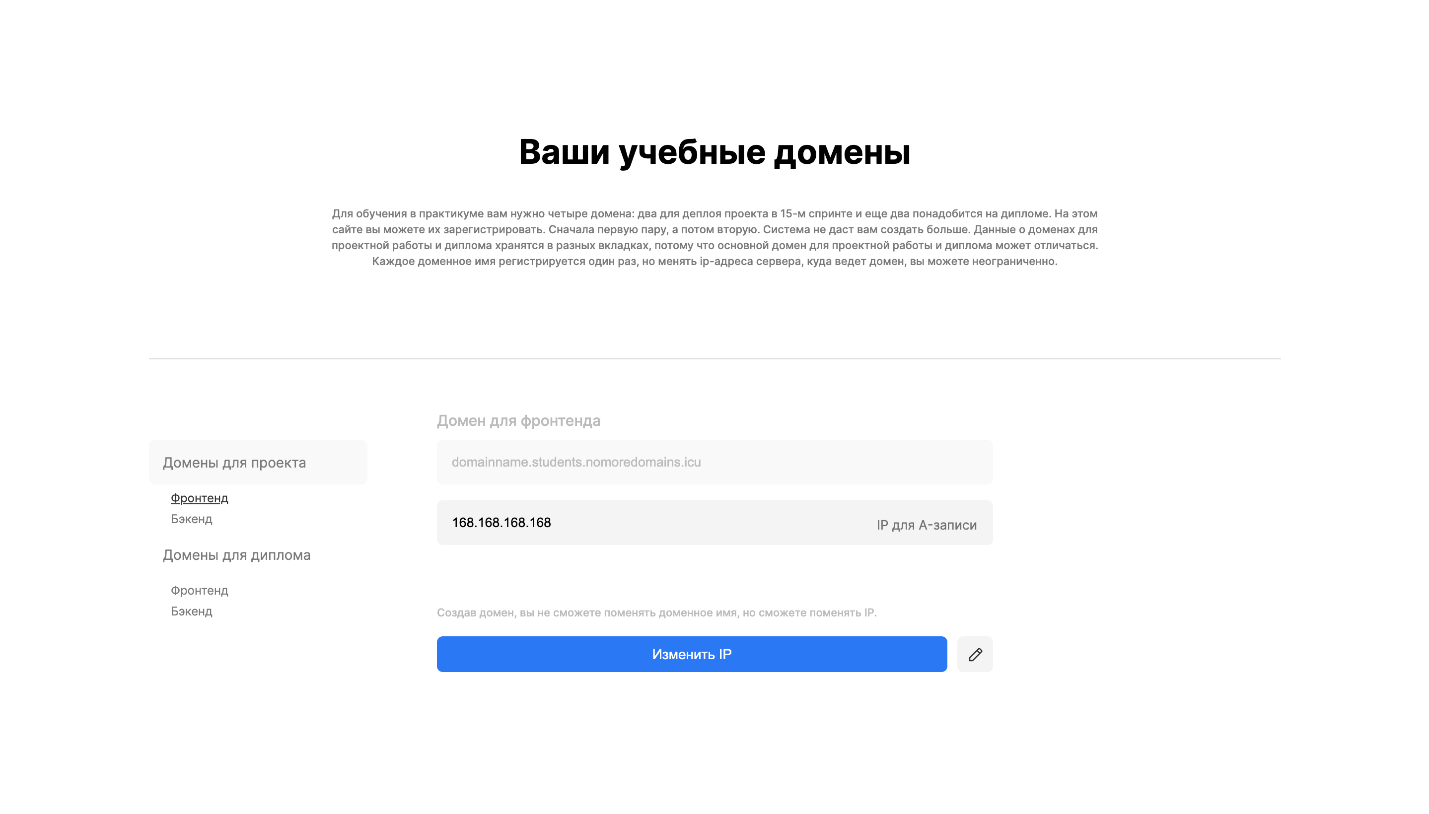
000.000.000.000

Как только все данные будут введены, нажмите «Сохранить». Если запрос на создание домена прошёл успешно, появится соответствующее сообщение:



Успешное создание домена

В дальнейшем, если публичный IP-адрес сервера в Яндекс.Облаке изменится, вы сможете отредактировать его нажатием на иконку редактирования. Изменить доменное имя нельзя.



Интерфейс редактирования IP-адреса домена

Теперь DNS-службе нужно время, чтобы обновить данные о домене. Обычно это занимает до 10 минут. В следующем уроке установим nginx. Нам понадобится nginx, чтобы раздавать статические файлы — фронтенд-часть приложения Stellar Burger.

# HTTP-сервер nginx

Nginx — это HTTP-сервер. Он умеет раздавать статические файлы, перенаправлять запросы, кешировать результат, и делает всё это очень быстро. Но на nginx нельзя создать сервер с какой-то сложной логикой. Нам сложная логика и не нужна, ведь мы просто собираемся раздавать статичные страницы фронтенд-части приложения, поэтому нам полностью подходит nginx.

Мы используем nginx как обратный прокси-сервер. Обратный прокси-сервер принимает запросы извне и перенаправляет их во внутреннюю сеть. Через неё мы будем раздавать статические файлы приложения.

Для начала нужно установить Nginx. Зайдите на удалённый сервер и выполните ряд команд:

Скопировать кодBASH

sudo apt update *# обновляем список пакетов (программ), доступных для установки*

sudo apt install -y nginx *# устанавливаем nginx*

После установки нужно настроить файрвол — прослойку между сервером и внешним миром, которая решает, какие запросы пускать дальше, а какие отклонять.

В операционной системе Ubuntu по умолчанию установлен файрвол — ufw. Сконфигурируем его, разрешив некоторые виды трафика. Для этого используйте команды:

Скопировать кодBASH

sudo ufw allow 'Nginx Full'

sudo ufw allow OpenSSH

sudo ufw allow 'Nginx Full' открывает на текущей машине два порта: 80 и 443, на них приходят http и https запросы к серверу. OpenSSH открывает порт 22 — соединение по ssh, чтобы мы могли подключаться к серверу через консоль и при работающем файрволе.

После этого нужно включить файрвол:

Скопировать кодBASH

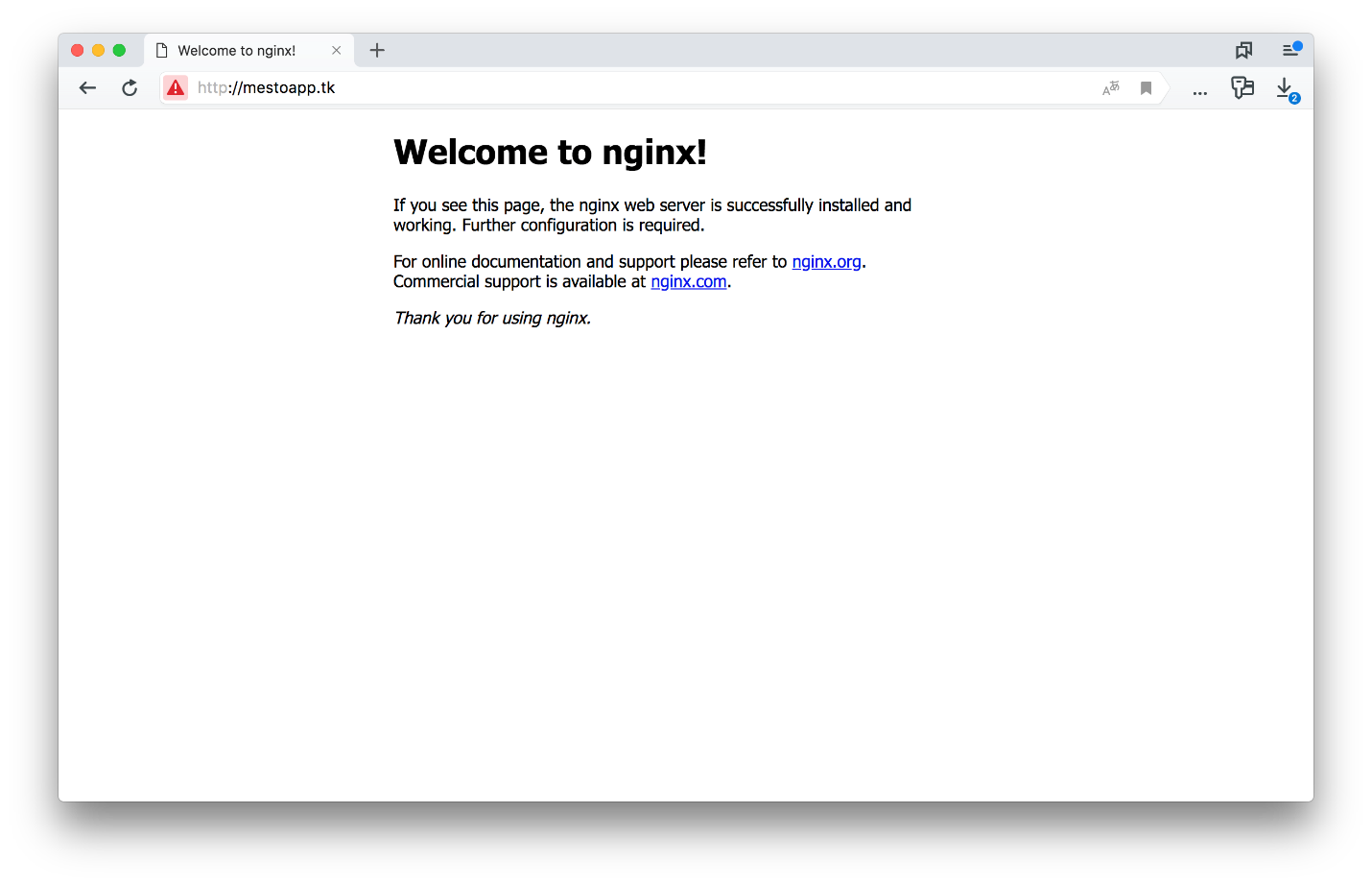
sudo ufw enable

Осталось запустить nginx и добавить его в автозапуск. Каждый раз при запуске сервера будет подниматься сервер nginx. Это можно сделать за один шаг при помощи следущей команды :

Скопировать кодBASH

sudo systemctl enable --now nginx

Откройте в браузере адрес вашего сервера. Вы должны увидеть:



Это значит, что nginx запустился и работает на 80 порту. Но сейчас он просто отдаёт стандартную html-страницу, а мы хотим, чтобы запросы перенаправлялись. Для этого нужно изменить конфигурационный файл nginx.

В nginx есть две директории для хранения конфигураций: sites-available/ и sites-enabled/. В первой хранятся конфигурации всех хостов, а во второй — ссылки только на активные. Перед редактированием конфигурации нужно создать символическую ссылку между sites-available и sites-enabled:

Скопировать кодBASH

sudo ln -s /etc/nginx/sites-available/<имя\_конфига> /etc/nginx/sites-enabled/<имя\_конфига>

Для конфигурации по умолчанию (default) символическая ссылка уже создана, так что этот этап можно пропустить.

Теперь переходим к редактированию файла конфигурации. В командной строке введите:

Скопировать кодBASH

sudo nano /etc/nginx/sites-available/default

*# sudo значит запустить команду как суперпользователь*

*# nano — это текстовый редактор*

*# /etc/nginx/sites-available/default — путь к конфигурационному файлу nginx*

Удалите содержимое этого файла и вставьте туда такую конфигурацию:

Скопировать кодBASH

server {

listen 80;

server\_name domainname.students.nomoreparties.co;

location / {

}

}

Этой конфигурацией мы настраиваем сервер Nginx следующим образом: слушаем 80 порт домена [domainname.students.nomoreparties.co](http://domainname.students.nomoreparties.co/) в блоке server\_name. В нём укажите то доменное имя, которое зарегистрированное ранее. Тогда такая конфигурация будет отдавать статику на все запросы к [http://domainname.students.nomoreparties.co](http://domainname.students.nomoreparties.co/)

Запоминать эти команды необязательно, их всегда можно подсмотреть в документации.

После того как всё сделаете, нажмите Ctrl+X. Nano спросит у вас, хотите ли вы сохранить изменения, нажимайте y и Enter.

После обновления конфигурации nginx нужно перезапустить. Но если в конфигурации окажутся ошибки, то в лучшем случае работа приложения будет остановлена. Nginx позволяет изменять и тестировать конфигурацию «на лету». Для этого до перезапуска nginx введите команду:

Скопировать кодBASH

sudo nginx -t

Она позволит проверить конфигурацию на наличие ошибок в синтаксисе команд. Если всё прошло успешно, вы получите такой ответ:

Скопировать кодBASH

nginx: the configuration file /etc/nginx/nginx.conf syntax is ok

nginx: configuration file /etc/nginx/nginx.conf test is successful

Уже после этого можно перезагрузить конфигурацию nginx:

Скопировать кодBASH

sudo systemctl reload nginx

В следующем уроке поговорим про CORS. И уже через урок загрузим фронтенд часть приложения на сервер.

# Размещаем фронтенд на сервере

Вы уже умеете деплоить фронтенд на Github Pages, но фронтенд реальных проектов обычно хранят не на Github, а на каком-нибудь сервере с красивым доменным именем. Нужно научиться делать так, чтобы при заходе на сайт с использованием доменного имени пользователь видел фронтенд сайта, и при этом сохранялась возможность делать запросы к API. Приступим.

## Заливаем код фронтенда на сервер

Залить код фронтенда можно клонирование git-репозитория. После того как репозиторий клонирован, нужно запустить на сервере процесс сборки, и когда он закончится, в папке build будут файлы, готовые к раздаче. Для этого в предыдущих уроках мы установили Node.js и git на сервер.

Но можно сделать проще. В случае фронтенда на сервере нам нужна только готовая сборка, а репозиторий с исходным кодом понадобится только для разработки. Поэтому процесс может быть таким:

* осуществить сборку локально;
* скопировать папку build на удалённый сервер.

Собирать фронтенд вы уже умеете, осталось придумать, где на сервере расположить папку с ним, и скопировать её туда. Для этого:

* на удалённом сервере создайте папку для фронтенда. Назовите её, например, stellar-frontend;
* соберите файлы локально;
* скопируйте файлы с локального компьютера на удалённый. Это можно сделать командой scp (замените praktikum и 84.201.172.240 на свои логин и A-запись):  
  bash scp -r ./build/\* praktikum@84.201.172.240:/home/praktikum/stellar-frontend

В таком виде команда scp принимает одну опцию и несколько аргументов. Опция -r говорит, что папки нужно копировать рекурсивно. Значит, если в папке dist есть вложенные папки, их содержимое тоже необходимо скопировать. Аргументы, которые принимает scp:

* ./build/\* — путь к локальным файлам или папкам, которые нужно скопировать. Build/\* означает, что нужно взять все файлы из папки build.
* praktikum@84.201.172.240:/home/praktikum/stellar-frontend — адрес удалённого сервера и папки на нём.

Для удобства можно добавить скрипт для деплоя в package.json проекта с фронтендом:

Скопировать кодJSON

"deploy-script": "npm run build && scp -r ./build/\* praktikum@84.201.172.240:/home/praktikum/mesto-frontend"

Такой скрипт сначала соберёт файлы, а затем скопирует их на сервер.

Осталось научить сервер их раздавать.

## Расширяем конфигурацию Nginx

С раздачей статичных файлов лучше справляется nginx. Он не только делает это быстро, но и позволяет относительно несложно реализовать сжатие этих файлов. Статические файлы мы загрузили в предыдущем шаге, а ещё ранее установили Nginx и даже добавили базовую конфигурацию.

Сконфигурируем nginx так, чтобы он отвечал за раздачу статики. Для этого нужно отредактировать созданный nginx конфигурационный файл. Зайдите на сервер и введите команду:

Скопировать кодBASH

sudo nano /etc/nginx/sites-available/default

В текстовом редакторе nano откроется файл конфигурации nginx. Его содержимое должно быть таким (вы указывали эту конфигурацию в [уроке про nginx](https://practicum.yandex.ru/trainer/react/lesson/28f5e1ee-fc66-4d26-9d18-12a8ef19d12f/)):

Скопировать код

server {

listen 80;

server\_name domainname.students.nomoreparties.co;

location / {

}

}

Теперь нужно сделать так, чтобы раздавался фронтенд Stellar Burger. Для этого в конфигурацию nginx следует добавить директиву root, передав ей путь к папке со статическими файлами:

Скопировать кодJSX

root /home/praktikum/stellar-frontend;

Добавить её можно где-то в блоке server, например, после директивы server\_name:

Скопировать код

server {

listen 80;

server\_name domainname.students.nomoreparties.co;

root /home/praktikum/stellar-frontend;

location / {

}

}

Всё, что остаётся сделать после этого — перезапустить nginx. Перед этим обязательно проверьте изменённую конфигурацию.

Скопировать кодBASH

sudo nginx -t

*# Если ошибок не появилось:*

sudo systemctl restart nginx

## Добавляем поддержку роутинга на фронтенде

По умолчанию nginx считает, что запрос пользователя обращён к какому-либо файлу. Если этот файл не обнаружен, сервер возвращает ошибку 404. Это приводит к тому, что nginx воспринимает маршруты на фронтенде как обращения к файлам, и роутинг перестаёт работать корректно. Исправим это.

В блок location / {...} конфига nginx и добавьте директиву try\_files $uri $uri/ /index.html; в конец:

Скопировать кодJSON

location / {

try\_files $uri $uri/ /index.html;

}

Эта директива сообщает nginx, что при обработке запроса сначала следует интерпретировать его как путь к файлу, затем к папке, а если ни того, ни другого не найдено, передавать его фронтенду в index.html. Теперь всё работает, как нужно: если по пути есть файл, nginx его отдаёт, в противном случае считает запрос маршрутом на фронтенде.

# Шифрование данных. Протокол HTTPS

Между клиентом и сервером происходит обмен информацией. Часто она конфиденциальна и нужно сделать всё, чтобы у злоумышленника не было к ней доступа. Для этого информацию между клиентом и сервером передают в зашифрованном виде. Шифрование осуществляется при помощи SSL-сертификата. Разберёмся, что это.

## Что такое сертификат

SSL расшифровывается как Secure Socket Layer (англ. «уровень защищённых сокетов»).

С помощью SSL-сертификата информация, передаваемая между клиентом и сервером, шифруется. Теперь, даже если злоумышленнику удастся её перехватить, он не сможет извлечь из этого пользы, так как информация зашифрована.

SSL-сертификат выдаётся так называемой Certificate Authority или CA (англ. «центр сертификации»). CA — третья сторона, которой клиент и сервер могут доверять. Она выдаёт сертификат на основе доменного имени и информации о сервере. Сертификат выпускается на доменное имя и его нужно создавать, зайдя на сервер. Проверив доменное имя и IP-адрес сервера, куда оно указывает, CA убедится, что сертификат выпускает именно владелец домена, и выдаст его.

После этого вся передаваемая информация может быть зашифрована и подписана с использованием выпущенного сертификата. А клиент сможет убедиться, что подписана информация именно сертификатом сервера и никаким другим. Таким образом, клиент и сервер смогут шифровать информацию и проверять, кем она зашифрована, это исключает изменение и подлог данных.

Некоторые CA выпускают сертификаты за деньги, но есть и бесплатные. Одна из самых популярных — Let's Encrypt. Её сертификатом мы и воспользуемся.

## Выпускаем сертификат

SSL-сертификат от Letsencrypt можно выпустить специальной программой — certbot.

[На сайте certbot](https://certbot.eff.org/lets-encrypt/ubuntubionic-nginx) есть описание процесса, как выпустить сертификат и сконфигурировать nginx, чтобы он его использовал. Можете сделать это самостоятельно или воспользоваться дальнейшей инструкцией.

Зайдите на сервер и по очереди введите команды:

Скопировать кодBASH

sudo apt update

sudo apt install -y certbot python3-certbot-nginx

Теперь certbot установлен.

## Подключаем сертификат

Чтобы подключить выпущенный сертификат, введите команду:

Скопировать кодBASH

sudo certbot --nginx

В процессе исполнения вам нужно несколько вопросов:

* Enter email address (англ. «введите почту»). Почта нужна для предупреждений, что сертификат пора обновить.
* Please read the Terms of Service; (A)gree/(C)ancel: (англ. «прочитайте правила сервиса; принять/отклонить»). Прочитайте правила по ссылке и введите a. Затем нажмите Enter.
* Would you be willing to share your email address with the Electronic Frontier Foundation? (англ. «хотите ли вы поделиться своей почтой с Фондом электронных рубежей»). Отметьте на своё усмотрение y (да) или n (нет) и нажмите Enter.
* Which names would you like to activate HTTPS for? (англ. «для каких доменных имён вы хотите включить https?»). Вам будет предложен вариант: domainname.students.nomoreparties.co. Это доменные имена, которые мы добавили в поле server\_name конфигурации nginx в предыдущих уроках. Ничего не вводите, просто нажмите Enter. Тогда https будет включён для всех доменов.
* Please choose whether or not to redirect HTTP traffic to HTTPS? 1: No redirect, 2: Redirect (англ. «нужно ли перенаправлять http траффик на https, 1: не перенаправлять, 2: перенаправлять»). Выберите 1 и нажмите Enter.

В итоге сертификаты будут выпущены. Также эта команда отредактирует конфигурацию nginx: добавит в неё нужные настройки и пропишет пути к сертификату.

Не забудьте перезапустить nginx:

Скопировать кодBASH

sudo systemctl reload nginx

Теперь весь трафик, который приходит на порт 443 (порт для https-запросов и ответов), будет шифроваться.

Откройте Postman и попробуйте отправить https-запрос — всё должно получиться. Это значит, что сертификат успешно подключён и информация между клиентом и сервером передаётся в зашифрованном виде.

## Обновление сертификата

Выпущенный сертификат нужно обновлять минимум раз в 3 месяца. Когда придёт время, воспользуйтесь командой:

Скопировать кодBASH

sudo certbot renew --pre-hook "service nginx stop" --post-hook "service nginx start"

Она обновит сертификат и перезапустит nginx.