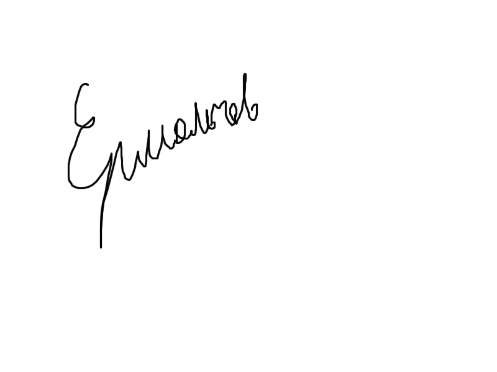
КУРСОВАЯ РАБОТА

*по теме*

**Организация CI/CD с помощью Jenkins и Docker**

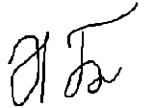
*по дисциплине* «Управление конфигурацией программного обеспечения для обработки больших объемов данных»



Выполнил:

студент гр.13544/4 Ф.А. Ермольчев

Руководитель:

ст. преподаватель  А.В.Баранов

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2018 г.

ЗАДАНИЕ

На выполнение курсового проекта

(курсовой работы)

Студенту группы 13544/4 Ермольчеву Ф.А.

*(номер группы) (фамилия, имя, отчество)*

*1. Тема проекта (работы):* Организация CI/CD с помощью Jenkins и Docker

*2. Срок сдачи студентом законченного проекта (работы) 28 декабря 2018*

*3. Исходные данные к проекту (работе)*: *Материалы лекций, документация и статьи для Docker, Jenkins, Slack, Docker-Compose, Makefile.*

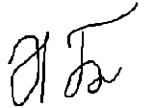
*4. Содержание пояснительной записки* (перечень подлежащих разработке вопросов): введение, основная часть (раскрывается структура основной части), заключение, список использованных источников, приложения.

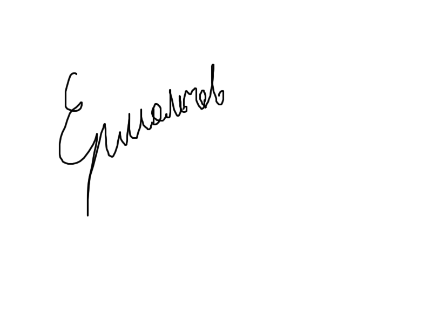
Примерный объем пояснительной записки *одна страница* машинописного текста.

*5. Перечень графического материала* (с указанием обязательных чертежей и плакатов): *иллюстрации и скриншоты работающего приложения*

*6. Консультанты* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*7. Дата получения задания*: «18» октября 2018г.

Руководитель  *Баранов А.В.*

*(подпись) (инициалы, фамилия)*

Задание принял к исполнению Ермольчев Ф.А.

*(подпись студента) (инициалы, фамилия)*

*(дата)* \_\_\_\_\_\_\_\_\_\_\_\_

**Оглавление**

[**Введение** 4](#_Toc535566417)

[**Цели и задачи** 5](#_Toc535566418)

[**Ход работы** 6](#_Toc535566419)

[**Заключение** 21](#_Toc535566420)

[**Литература** 22](#_Toc535566421)

[**Приложение (листинги)** 22](#_Toc535566422)

# **Введение**

CI и CD становятся все более популярными темами среди современных групп разработчиков. Вместе они позволяют команде создавать и тестировать код при любом коммите. Основным преимуществом этих подходов является возможность чаще выпускать более качественный код с помощью автоматизированных конвейеров.

В настоящее время используется Docker, который позволяет создать изолированный контейнер для приложения, включающий все его зависимости и позволяющий относительно просто его масштабировать и развертывать на совершенно разных машинах. В данной работе простое React приложение будет собрано и протестировано в эфемерных Docker-контейнерах.

Для непрерывной интеграции, которая не только упрощает жизнь программиста, но позволяет избежать ошибок на стадии разработки, будет использован Jenkins. В работе будет показана его настройка.

Для проверки деплоя, будет использоваться локальный docker-repository, а для уведомления о результатах – Slack.

# **Цели и задачи**

Основная цель: построение автоматизированного процесса CI/CD, для чего нужно выполнить следующие задачи:

1. Создать необходимые докер-контейнеры
2. Упаковать приложение в docker-контейнер
3. Создать пайплайн Jenkins для автоматической сборки, тестирования и упаковки приложения;
4. Деплоить в docker репозиторий для тестирования
5. Запустить приложение для проверки работоспособности CI/CD конвеера

# **Ход работы**

Пошагово опишем весь процесс:

1. Установка Docker

Настройка проводилась для компьютера с ОС Windows 7 x64. Docker Desktop for Windows – новая версия программы Docker. Однако, так как используется Windows 7, то был установлен Docker Toolbox – устаревшее ПО, которое работает на Win7.

<https://docs.docker.com/toolbox/toolbox_install_windows/>

1. Установка Jenkins

Было решено использовать Jenkins в Docker-контейнере. Для создания Докер-образа был написан Dockerfile. Это необходимо, если нужна более тонкая настройка контейнера. И в принципе для понимания, как собирать контейнер и из чего он состоит.

{Листинг 1}

1. Написание Dockerfile-ов: обратный прокси, докер-прокси и build-agent

Для обратного прокси используется nginx. Ретранслирует запросы на Jenkins. Использует порт 80.

{Листинг 2}

Docker-proxy – небольшой контейнер, который позволяет безопасно общаться Jenkins с build-agent контейнерами через порт 2375.

{Листинг 3}

Build-agent – эфемерный контейнер, в котором и будет собираться docker-контейнер с приложением. Сам же агент сборки динамический, то есть Jenkins соединяется с запускаемым Docker-контейнером и использует его как агент сборки.

{Листинг 4}

1. Настройка Docker

Необходимо создать docker-network. Для этого нужно запустить команду: docker network create --driver bridge jenkins-net

Нужно pull локальный docker registry для проверки деплоя: docker pull registry Для запуска: docker run -d -p 5000:5000 --restart=always --name registry registry

1. Написание Makefile-ов (либо Docker-compose.yml)

Создаются файлы, которые помогают легко запускать (собирать и т.д.) сразу несколько контейнеров с необходимыми параметрами. Можно использовать как Makefile-ы (общий Makefile запускает Makefile для каждого контейнера, в которых содержатся команды docker run, docker build и т.д.), так и более специализированный Docker-compose.

{Листинг 5}

1. Создание тоннеля к localhost с помощью ngrok

Это необходимо для работы Github Webhooks и др. То есть ngrok позволяет открыть компьютер для доступа через интернет. Для этого нужно скачать программу, запустить её и выполнить команду:

ngrok http 192.168.99.100:80

Где:

192.168.99.100 – это IP-адрес docker-machine. Чтобы его узнать, необходимо запустить команду: docker-machine ip.

:80 – порт, который использует обратный прокси nginx.

Теперь, чтобы взаимодействовать с компьютером через интернет, нужно будет использовать сгенерированный программой адрес. Например: <https://8an4dff8.ngrok.io> Теперь Jenkins доступен по этому адресу.

1. Настройка Jenkins, Github, Slack

На этом шаге производится настройка Jenkins. Это включает в себя: установку локали (языка) для Jenkins, Github Webhooks, Build slave, Slack integration, создание проекта.

После поднятия контейнера будет доступен на 192.168.99.100:80. Если это первый запуск Jenkins-контейнера, то нужно произвести предварительную настройку.

Для изменения настроек идем в *Manage Jenkins* – *Configure System*.

* Установка локали

*Locale* – *Default Language*. Вписываем тот, который нужен (en). Если нужно, то ставим галочку в опции: *Ignore browser preference and force this language to all users*

* Настройка Github Webhooks
  + Jenkins

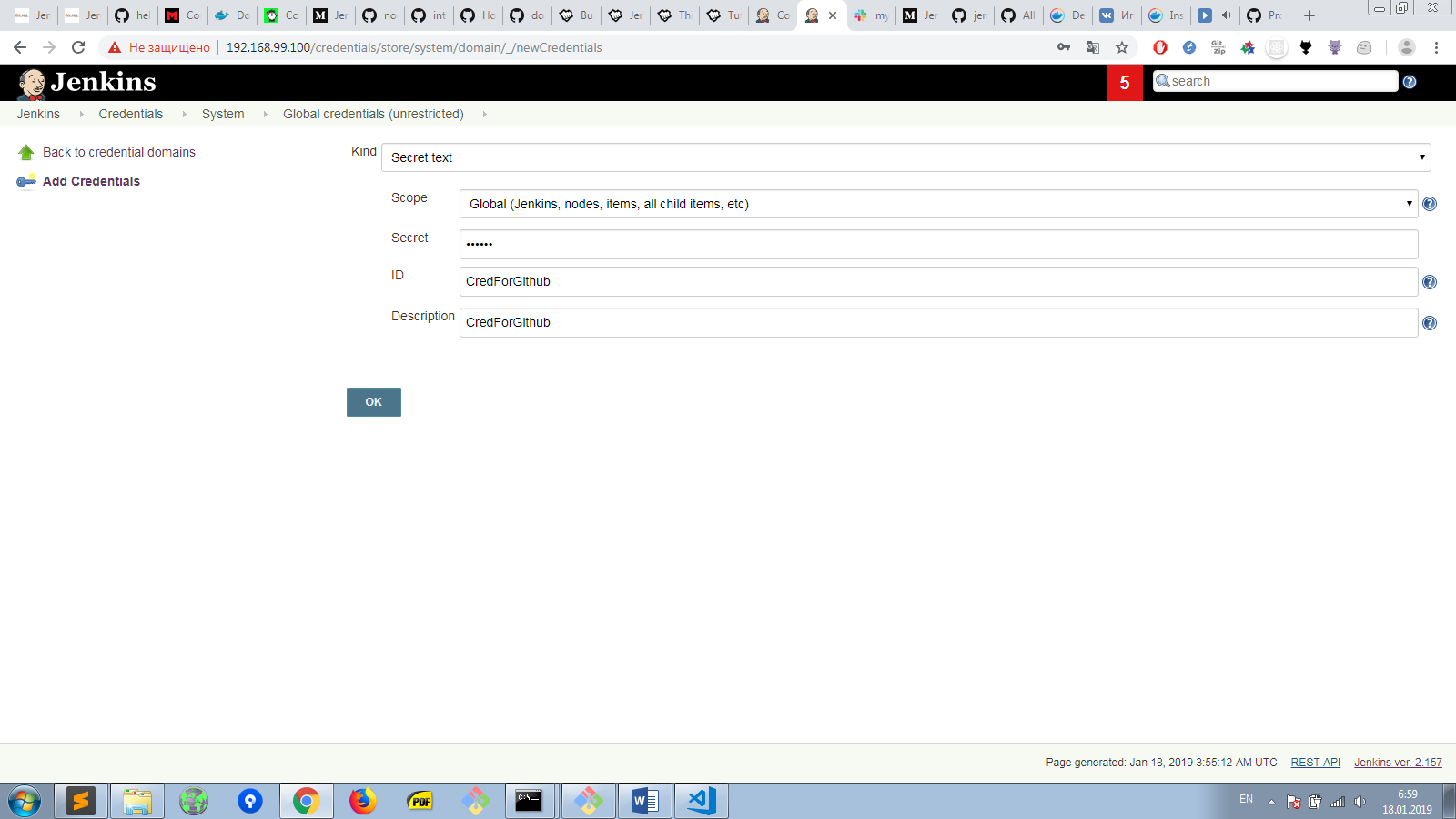
Создаем секретный токен для Github от Jenkins. Это должен быть секретный текст. Для этого идем в: *Credentials* – (*global*) – *Add* *Credentials*.

Заполняем поля:

Kind – Secret text

Secret – ваш секретный текст. Например, qwerty

ID, Description – любой понятный вам. Например, CredForGithub.



* + Github

Если нет токена (маркер доступа), то создаем его в настройках аккаунта Github. Инструкция: <https://help.github.com/articles/creating-a-personal-access-token-for-the-command-line/>

Далее, идем в репозиторий, где лежит код приложения, на котором будет испытываться CI/CD конвейер: *Settings* – *Webhooks* – *Add* *Webhook*

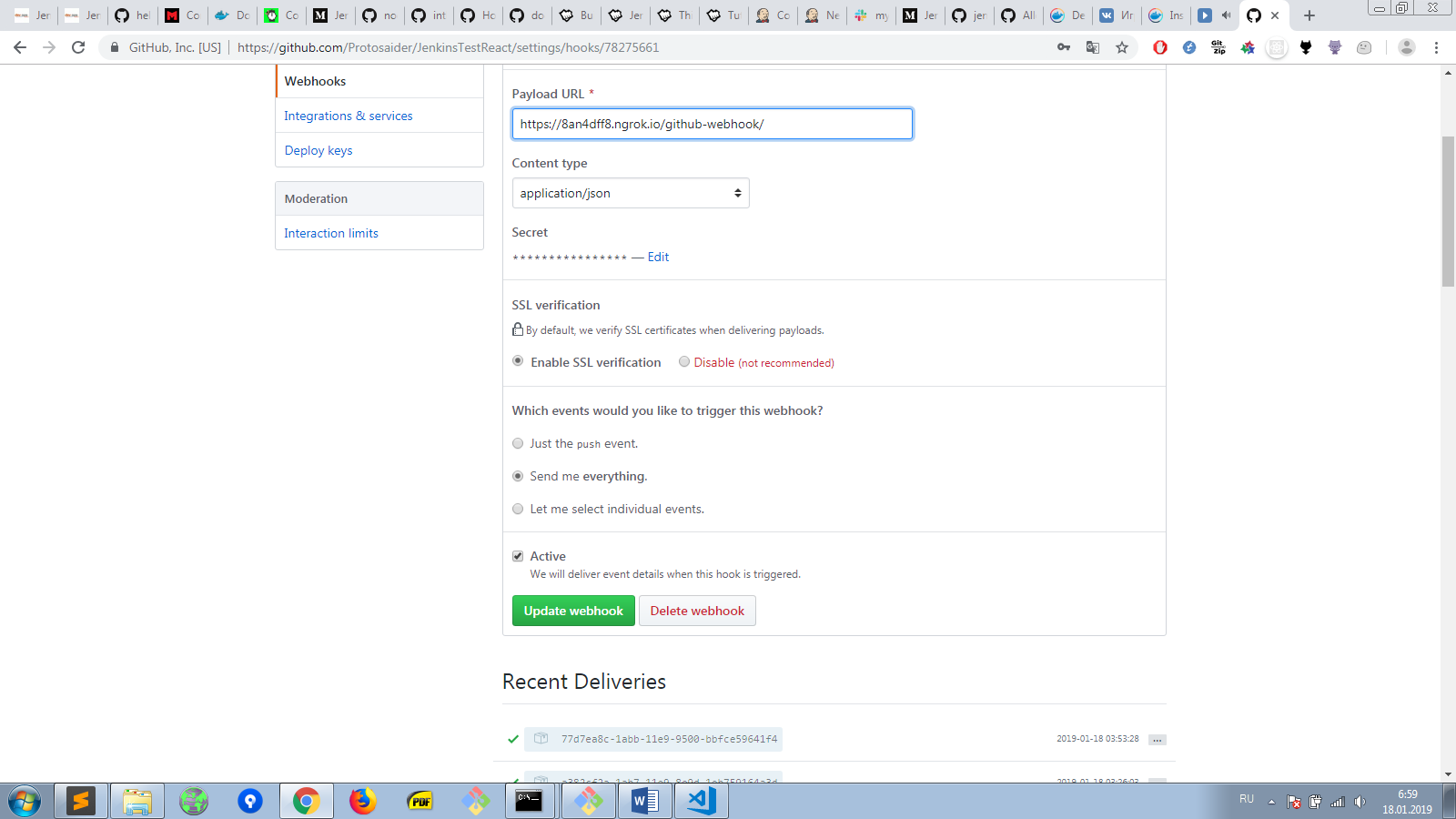
Заполняем данные:

Payload URL – используем адрес ngrok-тоннеля для Jenkins Webhooks. Пример: <https://8an4dff8.ngrok.io/github-webhook/>

Content type – application/json

Secret – используем текст из созданного в Jenkins секретного токена (CredForGithub)

SSL Verification, Which events would you like to trigger this webhook? – на ваше усмотрение.



* + Jenkins снова

Создаем секретный токен для Jenkins от Github. Это должен быть секретный текст. Для этого идем в: *Credentials* – (*global*) – *Add* *Credentials*. Заполняем поля:

Kind – Secret text

Secret – ваш секретный текст токена из Github.

ID, Description – любой понятный вам. Например, CredFromGithub.

Возвращаемся в *Manage Jenkins* – *Configure System*

*GitHub* – *Add* *Github* *Server*

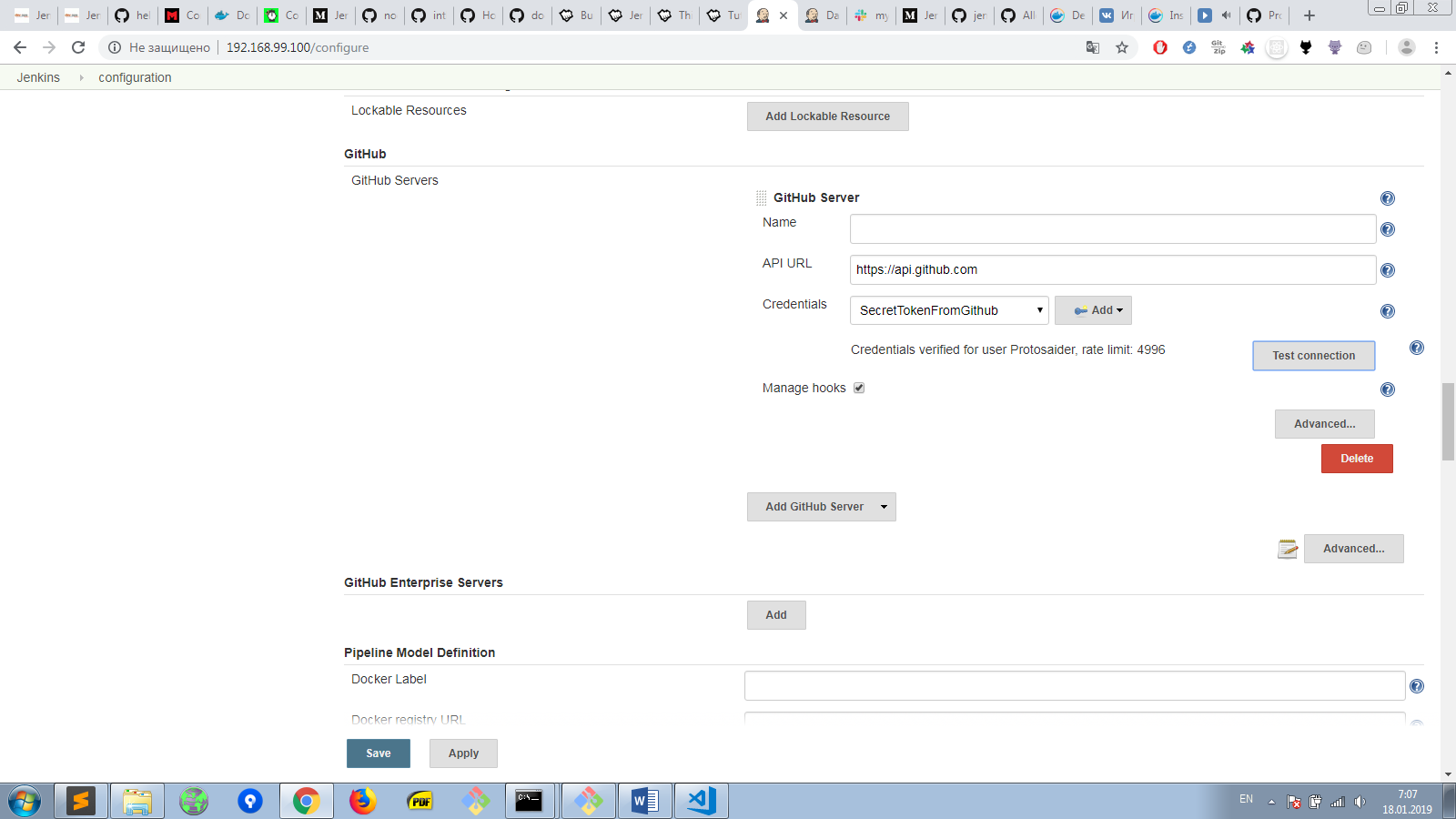
Заполняем:

Name, API URL – опционально, см. описание пунктов.

Credentials – выбираем токен из Github. (CredFromGithub)

Test Connection – при успехе ответ похож на этот: *Credentials verified for user Protosaider, rate limit: 4996*

Ставим галочку в *Manage hooks*



* Настройка Build Slave

*Manage Jenkins* – *Configure System – Cloud*

*Add new cloud*

Заполняем: Name – любое

*Docker cloud details…*

Заполняем:

Docker Host URI – используем docker alias – псевдоним для jenkins-net (созданный docker network): *tcp://proxy1:2375*

Галочка на Enabled

*Docker agent templates…*

Заполняем:

Labels – любое

Галочка на Enabled

Name – любое

Docker Image – название Docker образа для агента, например ephemeral-slave

*Container settings…*

Volumes – прописываем docker.sock (аналогично опции docker run –v /var/run…, то есть пробрасываем внутрь контейнера): /var/run/docker.sock:/var/run/docker.sock

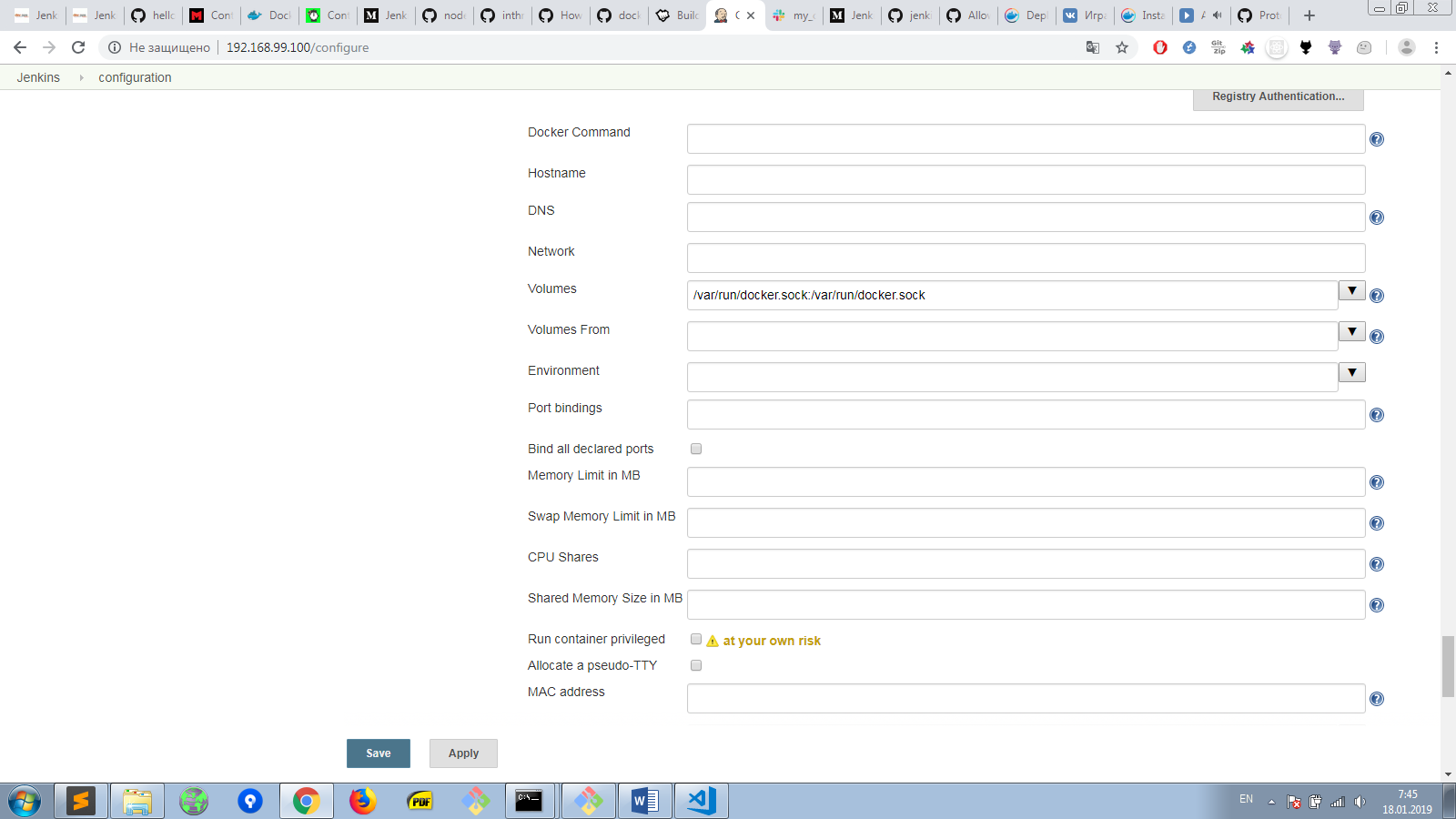
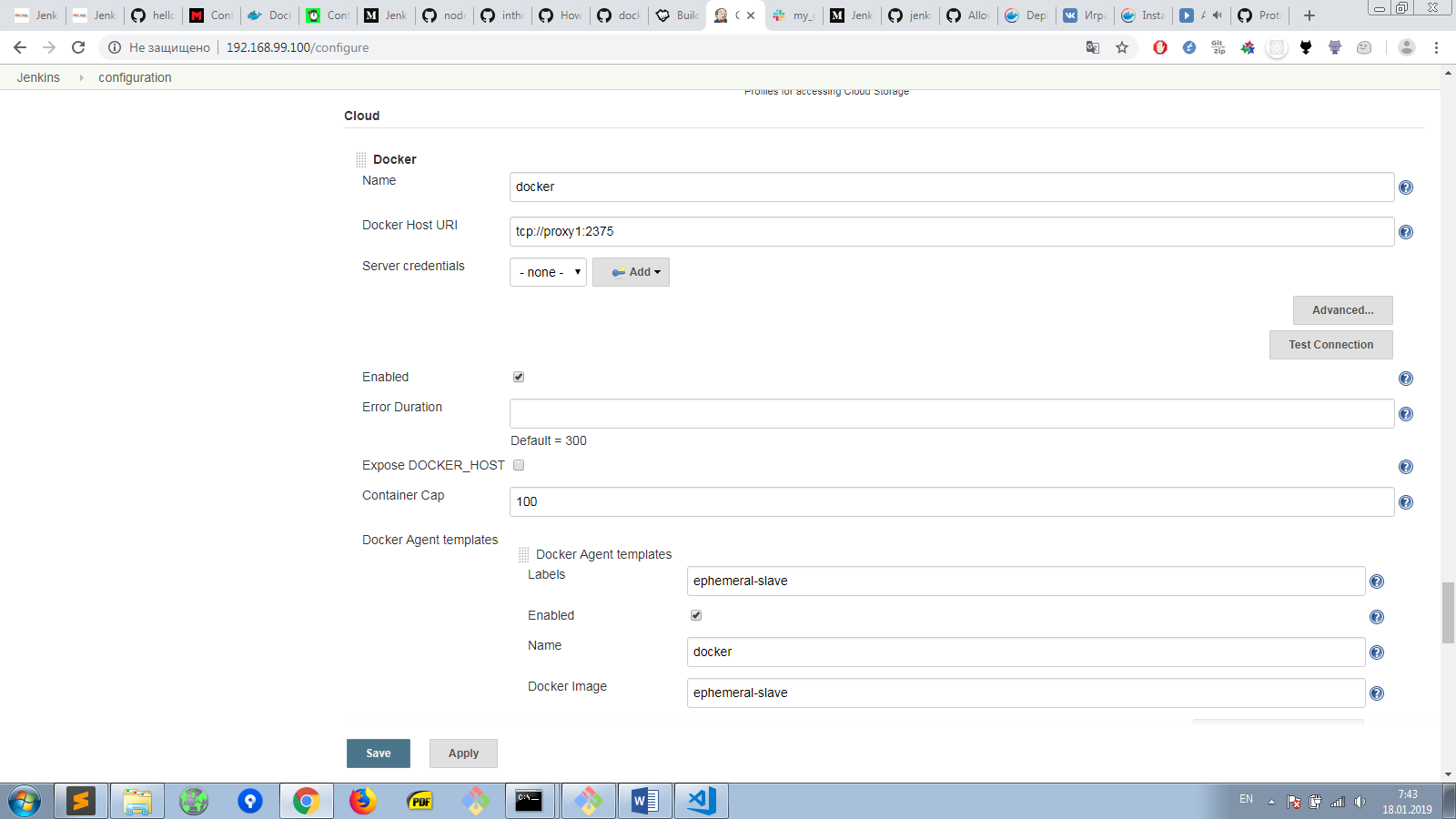
Remote File System Root – домашняя директория: /home/jenkins

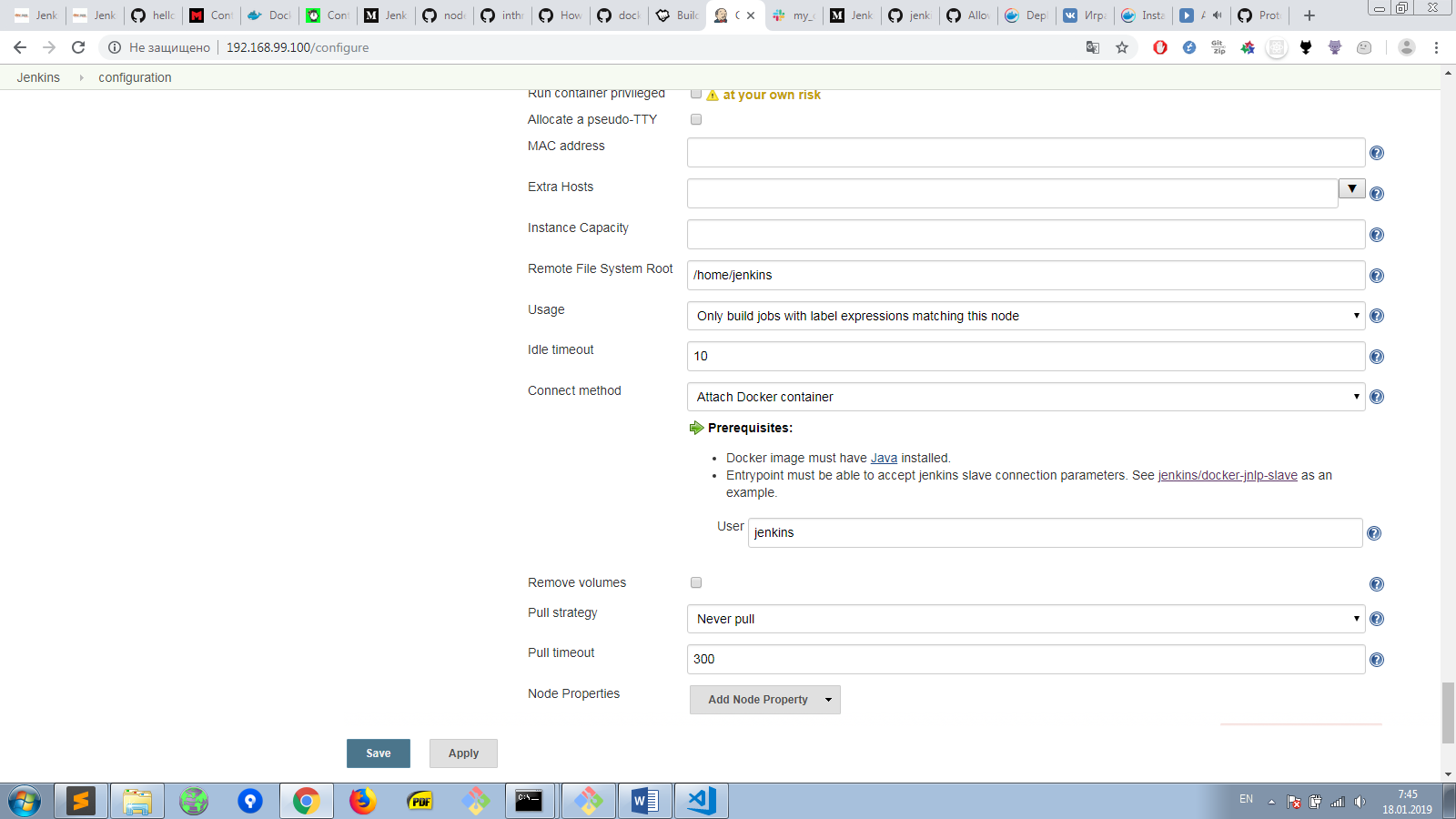
Usage - only build jobs with the label expressions matching this node

Connect Method – Attach Docker Container

User – jenkins

Pull strategy – Never pull





* Slack integration
  + Slack

Устанавливаем Jenkins-ci app:

<https://workinghandguard.slack.com/apps/A0F7VRFKN-jenkins-ci>

*Edit* *configuration*

Token – копируем оттуда текст

* + Jenkins

Создаем секретный токен для Jenkins от Slack. Это должен быть секретный текст. Для этого идем в: *Credentials* – (*global*) – *Add* *Credentials*.

Заполняем поля:

Kind – Secret text

Secret – ваш секретный текст токена из Slack.

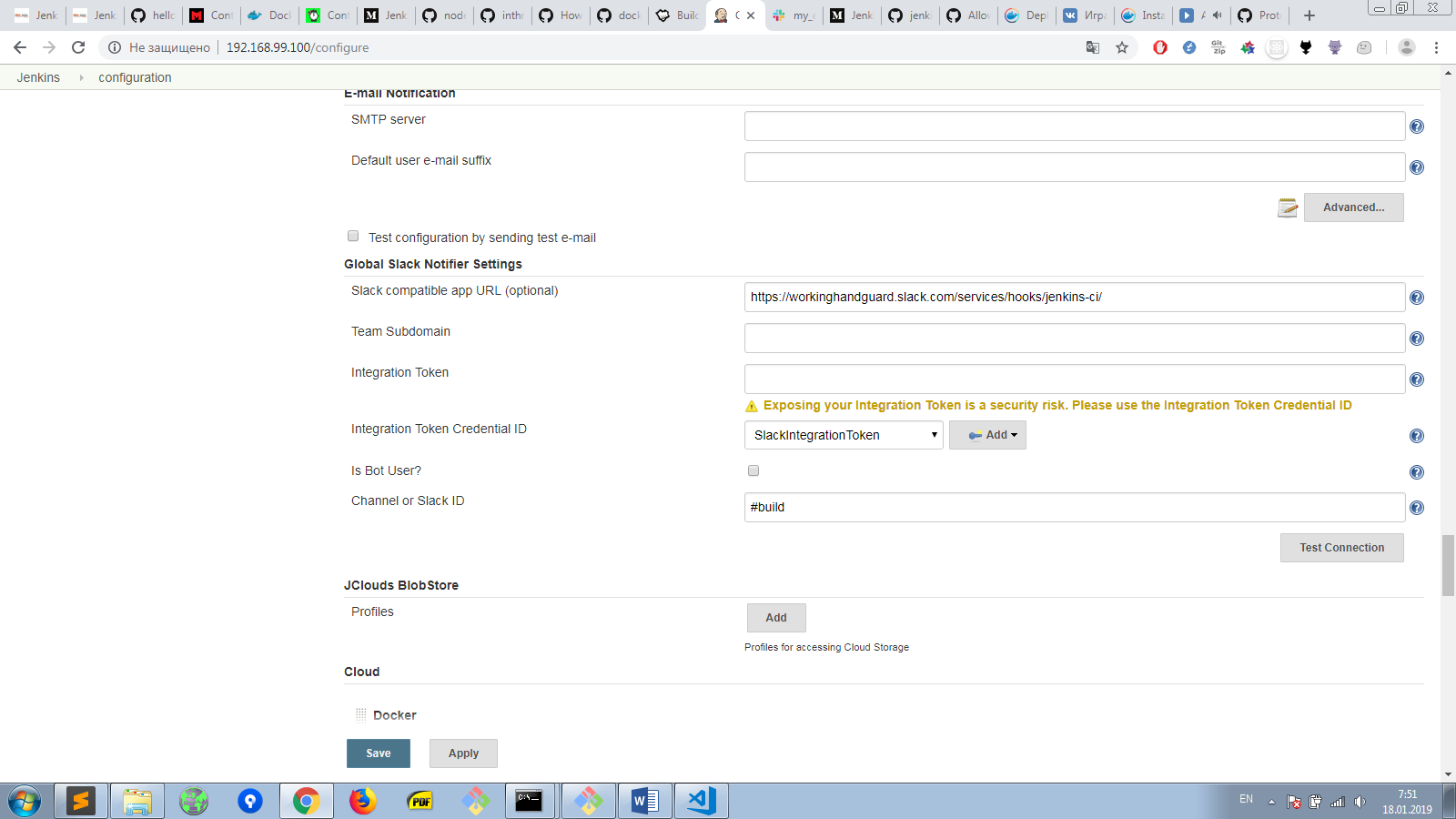
ID, Description – любой понятный вам. Например, TokenFromSlack.

Теперь идем в *Manage Jenkins* – *Configure System -- Global Slack Notifier Settings*

Заполняем:

Slack compatible app URL – адрес хуков вашего workspace: <https://youname.slack.com/services/hooks/jenkins-ci/>

Integration Token Credential ID – выбираем созданный токен (TokenFromSlack)



* Создание проекта

*New Item*

Заполняем:

Item name – любое (например, test)

Multibranch pipeline

*OK*

*Branch* *Sources* – *Add* *Sources* – *Github*

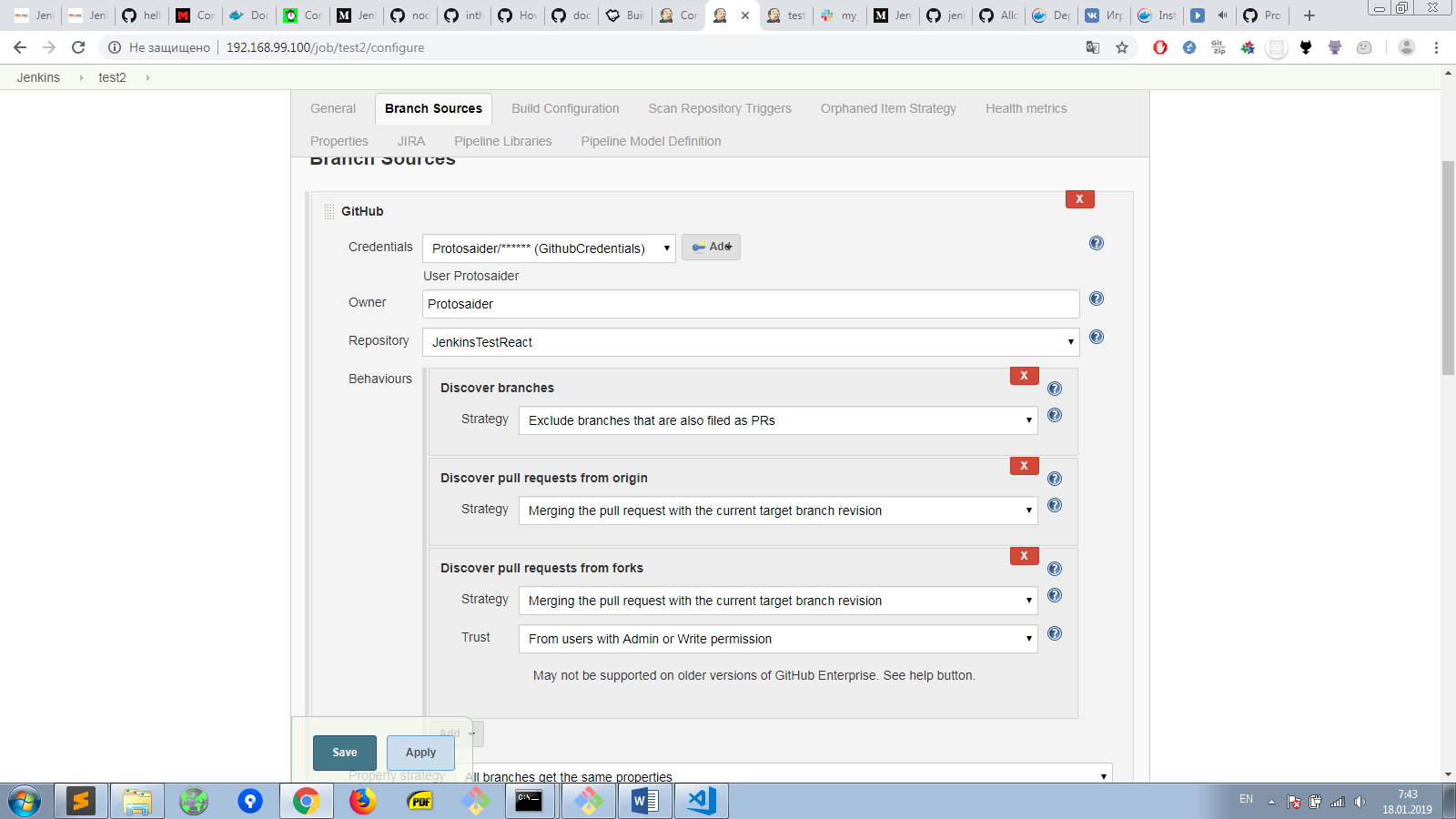
Заполняем:

Credentials – создаем секретный токен из логина и пароля от Github, либо используем ранее созданный токен для Jenkins от Github. (CredFromGithub)

Owner – имя аккаунта Github

Repository – выберите нужный репозиторий

*Save*



1. Написание Jenkinsfile, Dockerfile, Dockerfile.test

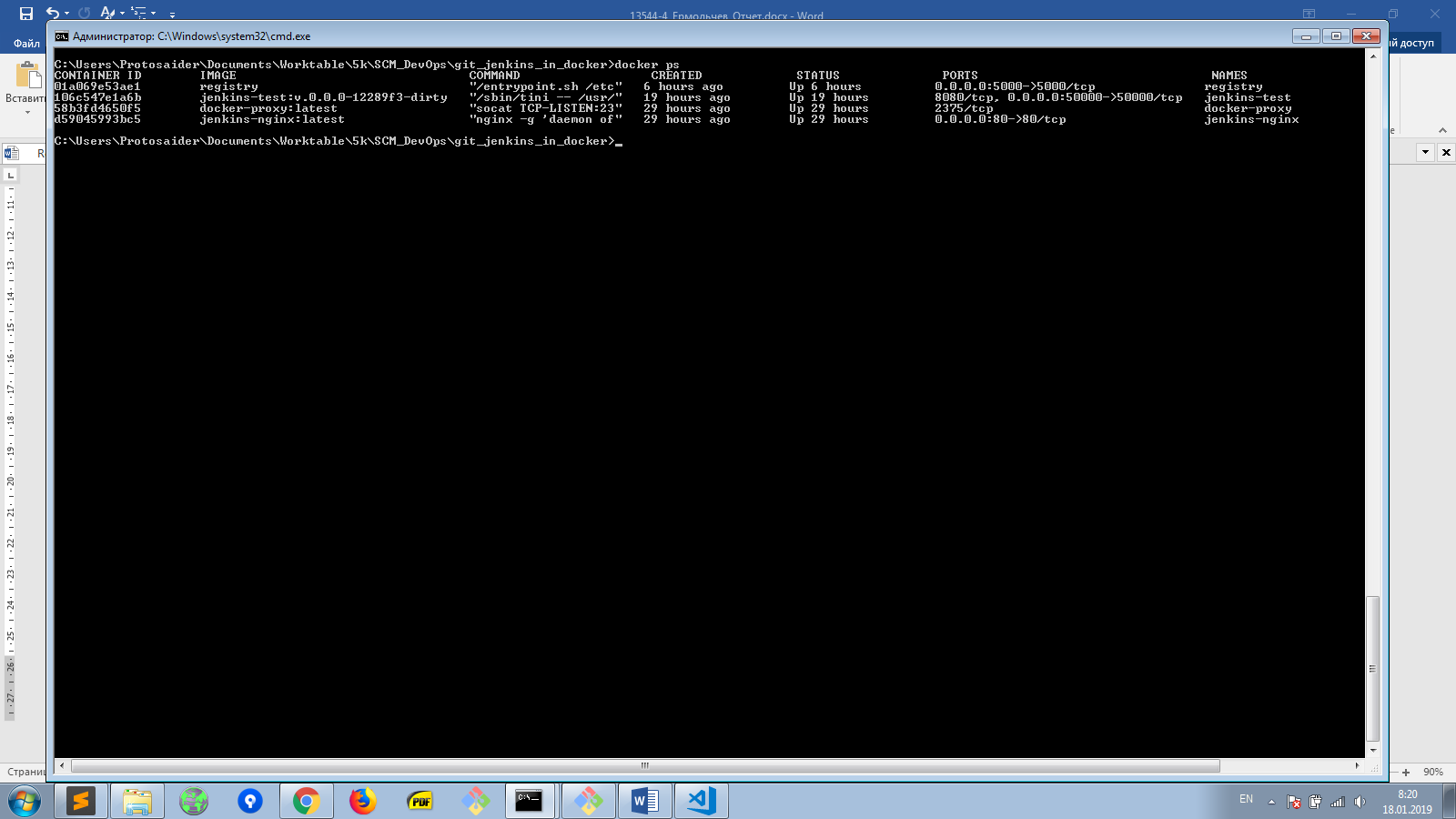
Этот файл предназначен для хранения Jenkins-pipeline, которая будет запущена внутри Jenkins после коммита в репозитории Github. Также необходимо написать Dockerfile-ы для контейнеров, в одном из которых будет проводится тестирование приложения, а второй – для деплоя.

{Листинги 6, 7, 8}

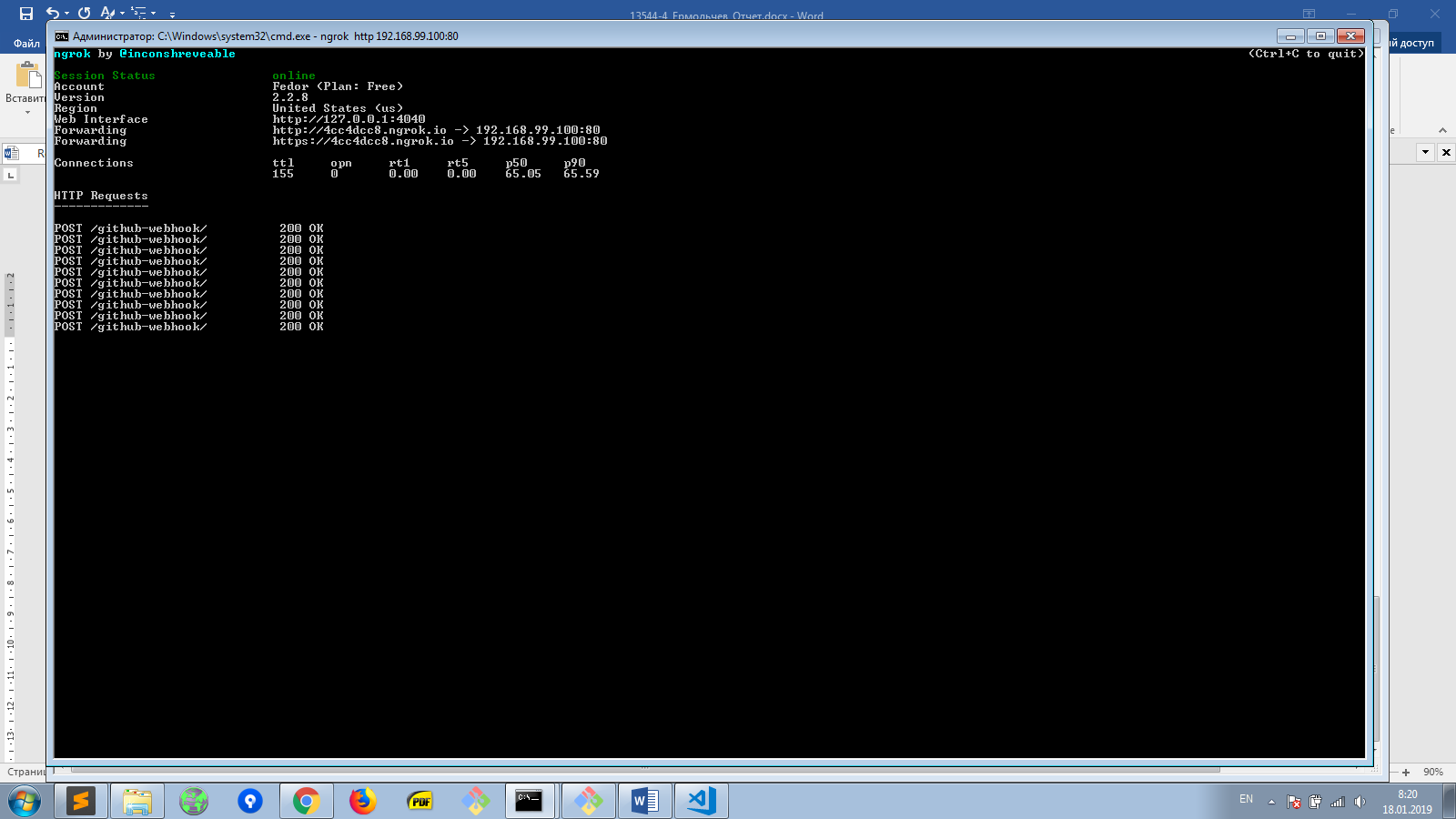
1. Проверка работоспособности

Заключительный этап для демонстрации работоспособности и использования созданного CI/CD конвейера:

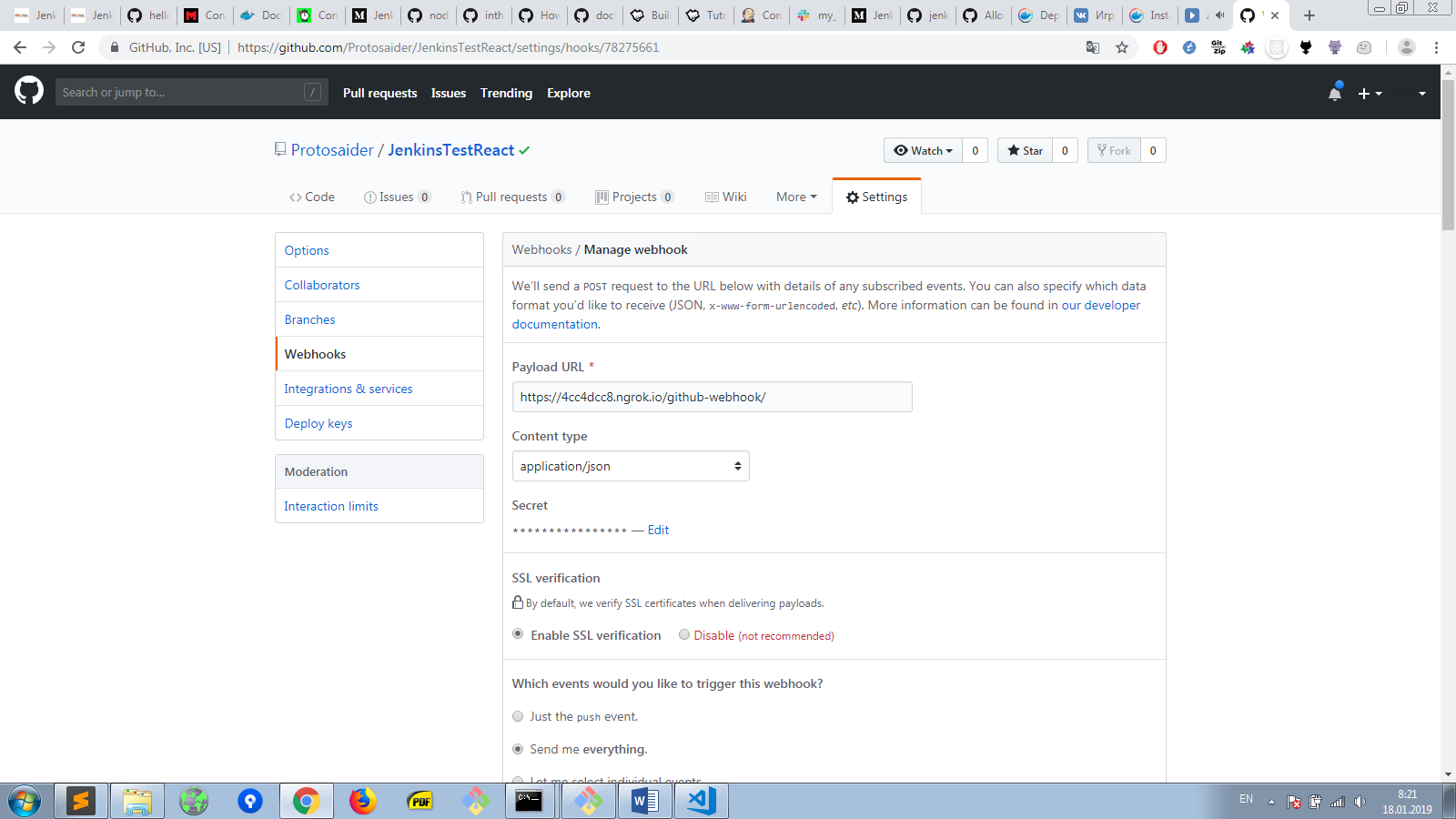
Запускаем контейнеры Jenkins-master, nginx-proxy, docker-proxy, docker registry (с помощью docker-compose или Makefile).



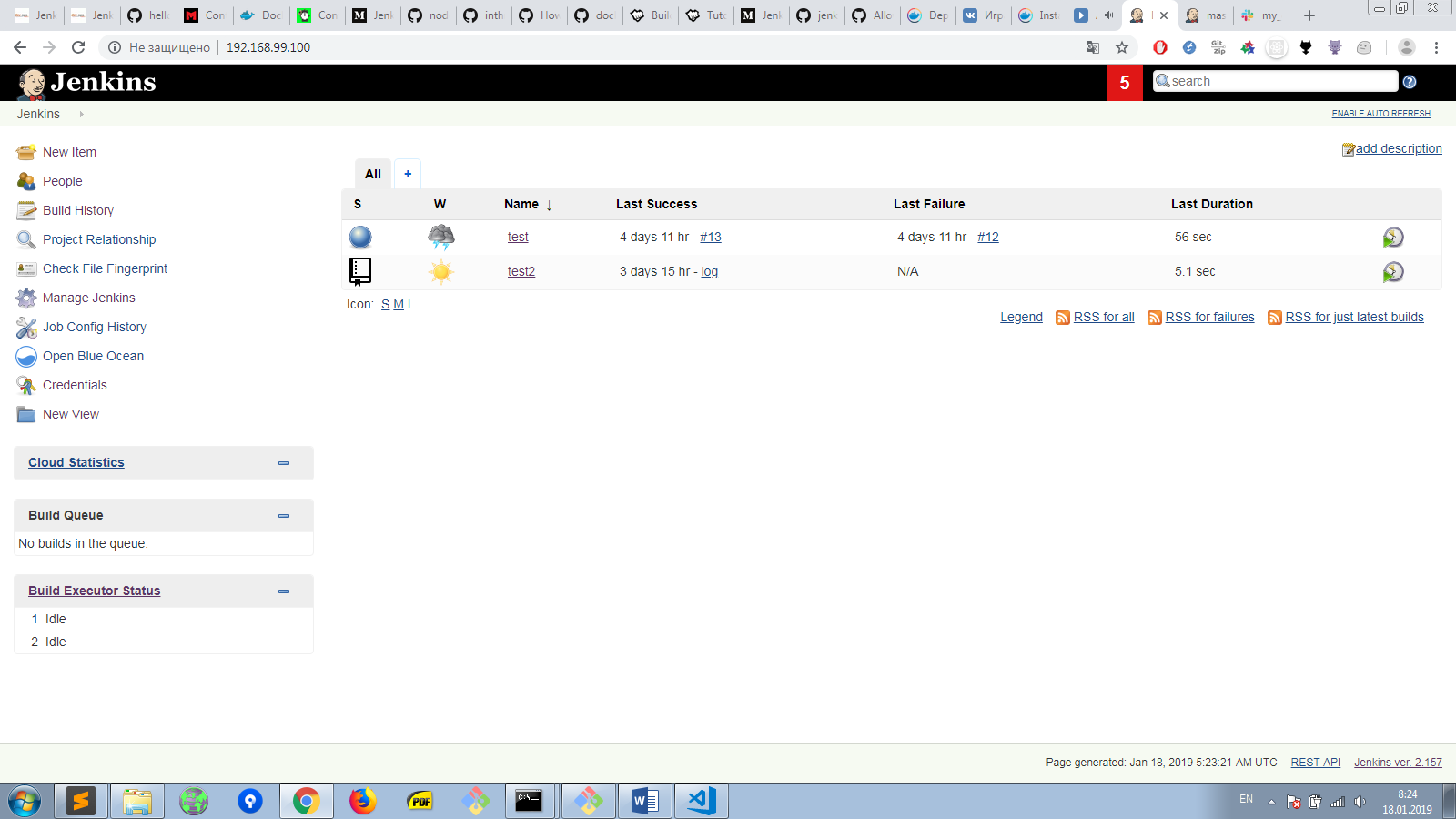
Запускаем ngrok.



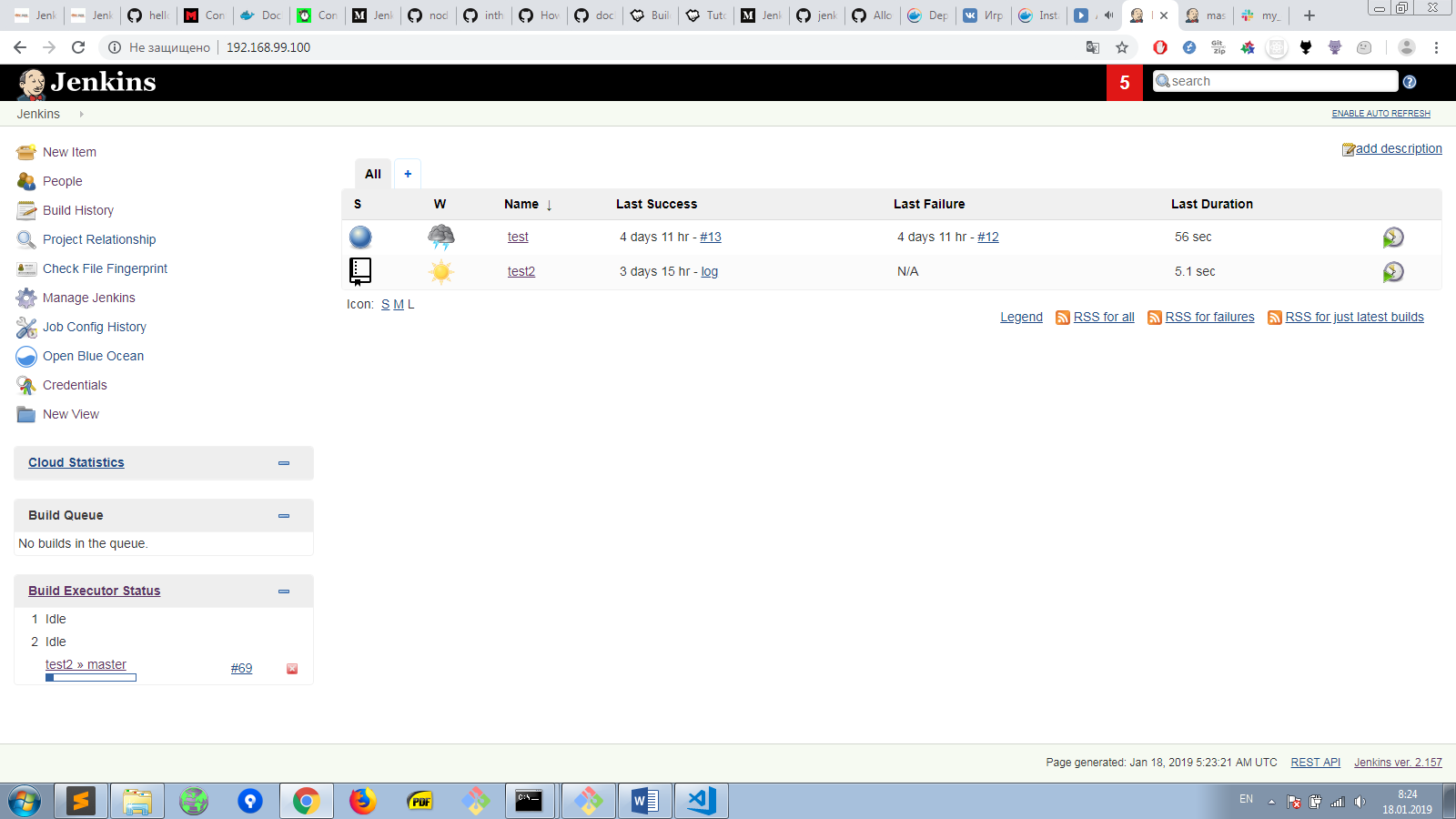
Проверяем, что адрес в Github Webhooks соответствует адресу тоннеля ngrok.



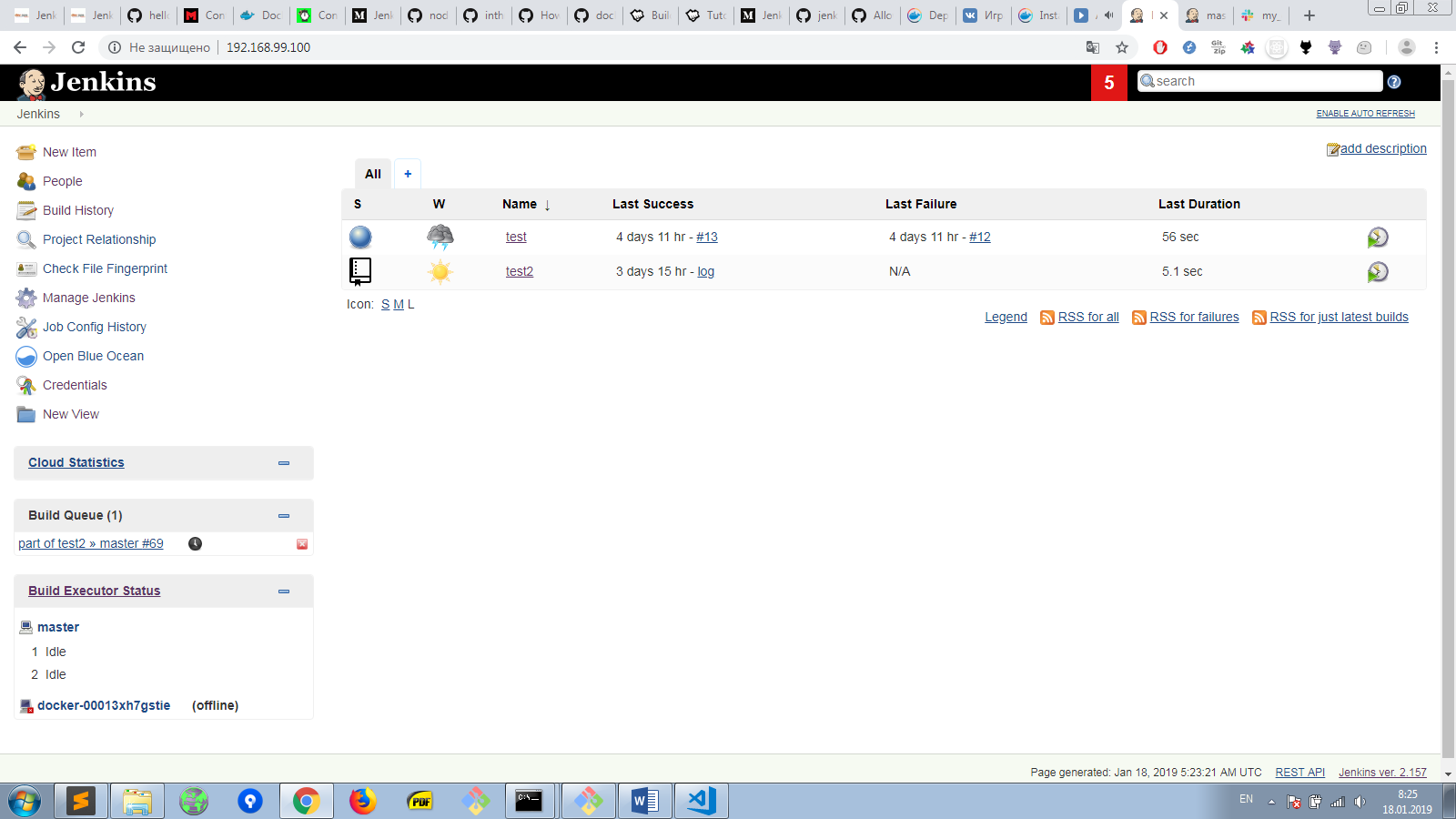
Идем в Jenkins.



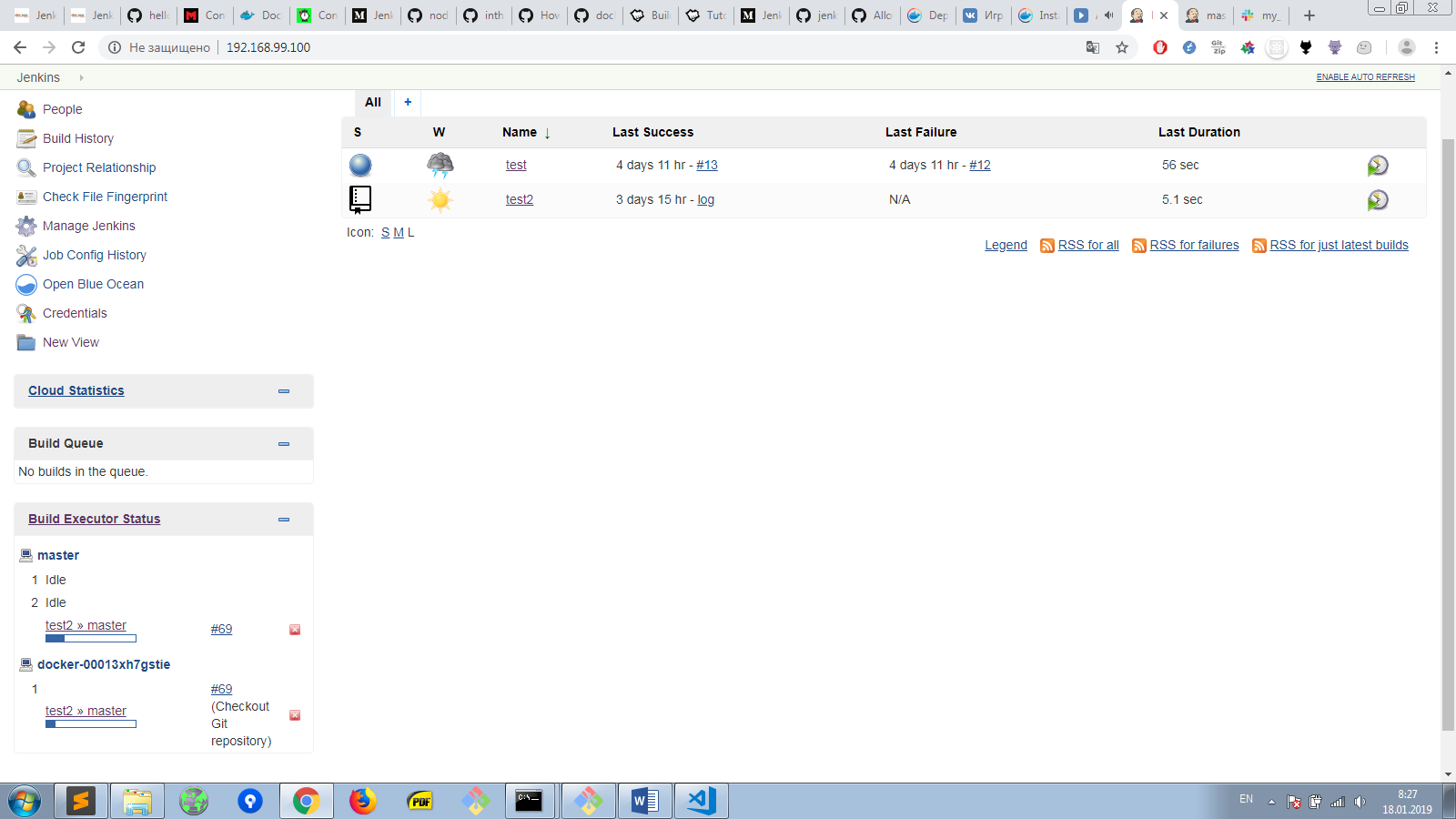
Пушим изменения в репозиторий с программой, либо вручную запускаем сборку.



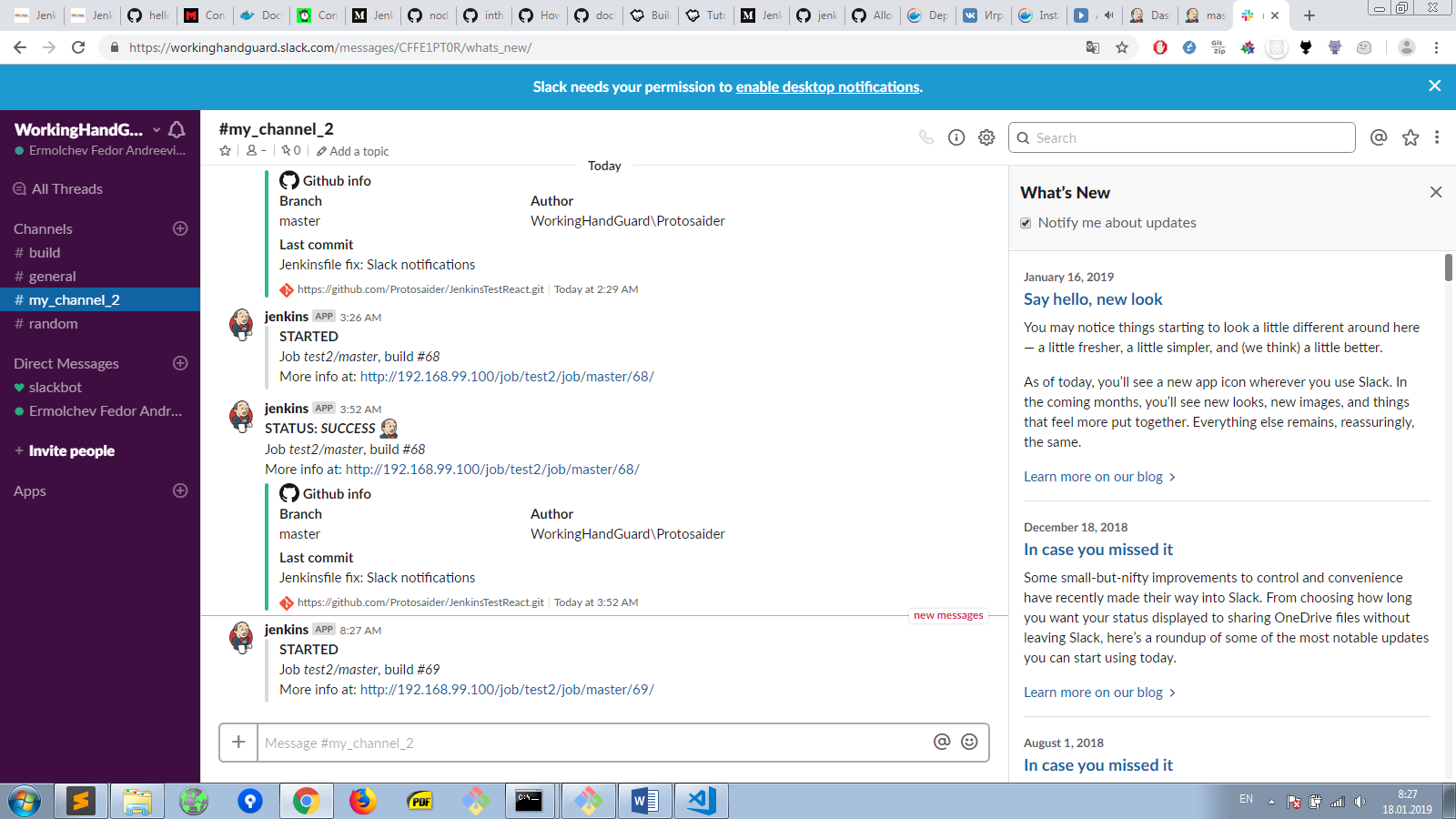
Смотрим, как динамически создается build-slave.



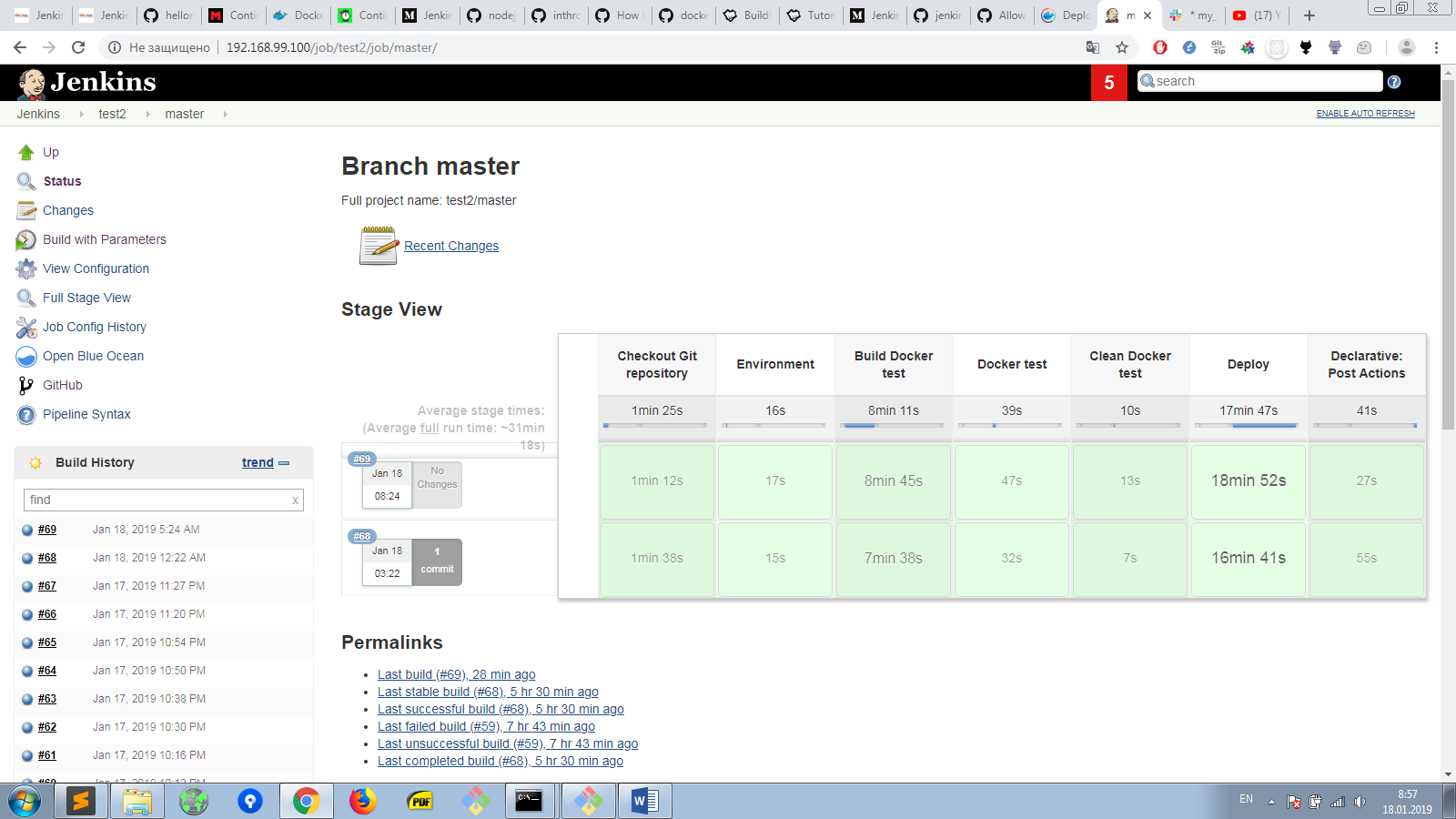
Видим, как он начал свою работу.

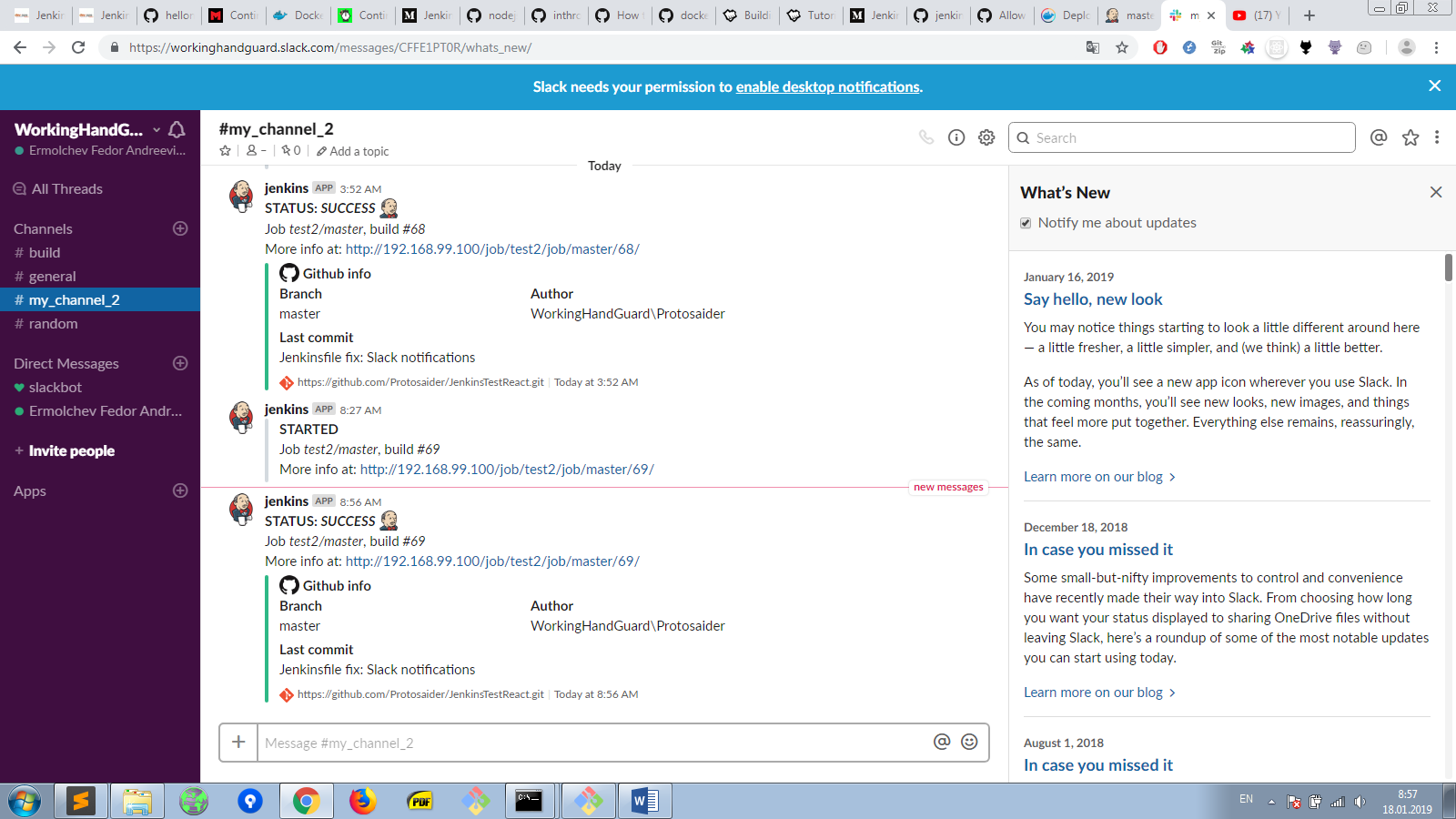


Получаем уведомление в Slack.



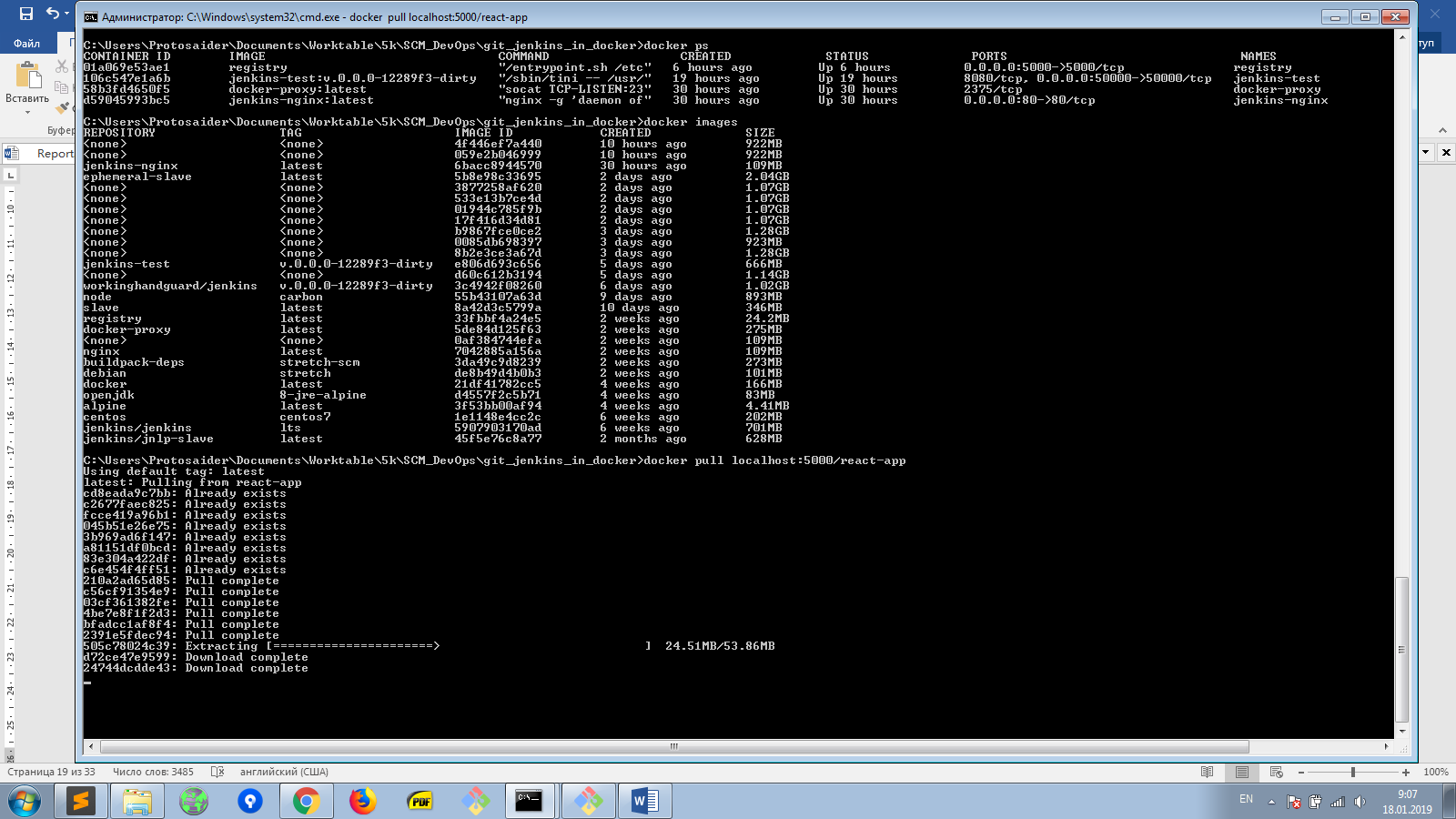
Получаем уведомление о завершенной работе.





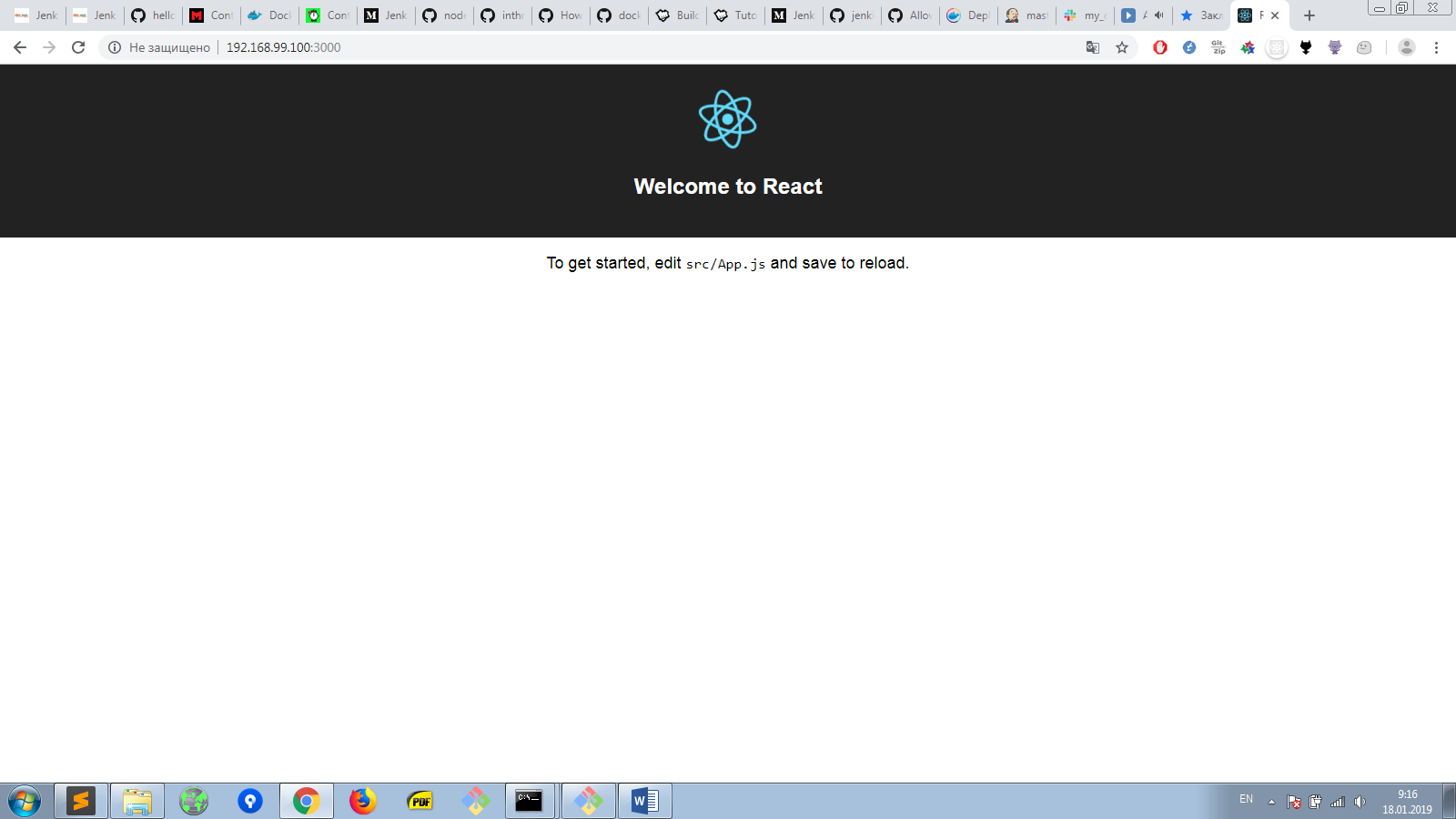
Теперь pull-им собранный контейнер из local docker registry:

docker pull localhost:5000/react-app



Запускаем контейнер и проверяем, что приложение работает:

docker run -d -p 3000:3000 --name react localhost:5000/react-app:latest



# **Заключение**

В итоге, была выполнена основная цель работы – построен CI/CD проект, в ходе работы над которым были рассмотрены такие инструменты как Docker и Jenkins.

# **Литература**

1. Using Docker-in-Docker for your CI environment? Think twice. [Электронный ресурс] – Ссылка: <https://jpetazzo.github.io/2015/09/03/do-not-use-docker-in-docker-for-ci/>
2. THINKING INSIDE THE CONTAINER  
   [Электронный ресурс] – Ссылка: <https://engineering.riotgames.com/news/thinking-inside-container>
3. Docker Documentation [Электронный ресурс]. – Ссылка: https://docs.docker.com/
4. Docker Compose [Электронный ресурс]. – Ссылка: https://docs.docker.com/compose/
5. Docker Hub [Электронный ресурс]. – Ссылка: https://hub.docker.com/
6. Jenkins User Documentation [Электронный ресурс]. – Ссылка: <https://jenkins.io/doc/>
7. Jenkins Pipeline: Send Slack Notifications using Shared Library  
   [Электронный ресурс] – Ссылка: https://medium.com/@lvthillo/send-slack-notifications-in-jenkins-pipelines-using-a-shared-library-873ca876f72c

# **Приложение (листинги)**

{Листинг 1: Dockerfile для Jenkins-master}

FROM debian:stretch

ENV LANG C.UTF-8

# see https://bugs.debian.org/775775

# and https://github.com/docker-library/java/issues/19#issuecomment-70546872

ENV CA\_CERTIFICATES\_JAVA\_VERSION 20170531+nmu1

RUN apt-get update \

&& apt-get install -y --no-install-recommends \

wget \

curl \

ca-certificates \

zip \

openssh-client \

unzip \

openjdk-8-jdk \

ca-certificates-java \

git \

&& rm -rf /var/lib/apt/lists/\*

RUN /var/lib/dpkg/info/ca-certificates-java.postinst configure

# Change uid gid to 1001 to resolve problems with Docker (security implications)

ARG user=jenkins

ARG group=jenkins

ARG uid=1001

ARG gid=1001

ARG http\_port=8080

ARG agent\_port=50000

ARG JENKINS\_VERSION=2.157

ARG TINI\_VERSION=v0.18.0

# jenkins.war checksum, download will be validated using it

ARG JENKINS\_SHA=8b03ec1c74325df030c65d4f5347191efb9f3a6ee84a27fb3050a18513d937b6

# Can be used to customize where jenkins.war get downloaded from

ARG JENKINS\_URL=http://mirrors.jenkins.io/war/${JENKINS\_VERSION}/jenkins.war

ENV JENKINS\_VERSION ${JENKINS\_VERSION}

ENV JENKINS\_HOME /var/jenkins\_home

ENV JENKINS\_SLAVE\_AGENT\_PORT ${agent\_port}

ENV JENKINS\_UC https://updates.jenkins.io

ENV JENKINS\_UC\_EXPERIMENTAL=https://updates.jenkins.io/experimental

ENV JAVA\_OPTS="-Xmx8192m -Djenkins.install.runSetupWizard=false"

ENV JENKINS\_OPTS="--handlerCountMax=300 --logfile=/var/log/jenkins/jenkins.log --webroot=/var/cache/jenkins/war"

ENV COPY\_REFERENCE\_FILE\_LOG $JENKINS\_HOME/copy\_reference\_file.log

# Use tini as subreaper in Docker container to adopt zombie processes

RUN curl -fsSL https://github.com/krallin/tini/releases/download/${TINI\_VERSION}/tini-static-$(dpkg --print-architecture) -o /sbin/tini \

&& chmod +x /sbin/tini

# Jenkins is run with user `jenkins`, uid = 1001

# If you bind mount a volume from the host or a data container,

# ensure you use the same uid

RUN groupadd -g ${gid} ${group} \

&& useradd -d "$JENKINS\_HOME" -u ${uid} -g ${gid} -m -s /bin/bash ${user}

# Jenkins home directory is a volume, so configuration and build history

# can be persisted and survive image upgrades

VOLUME /var/jenkins\_home

# `/usr/share/jenkins/ref/` contains all reference configuration we want

# to set on a fresh new installation. Use it to bundle additional plugins

# or config file with your custom jenkins Docker image.

RUN mkdir -p /usr/share/jenkins/ref/init.groovy.d

# could use ADD but this one does not check Last-Modified header neither does it allow to control checksum

# see https://github.com/docker/docker/issues/8331

RUN curl -fsSL ${JENKINS\_URL} -o /usr/share/jenkins/jenkins.war \

&& echo "${JENKINS\_SHA} /usr/share/jenkins/jenkins.war" | sha256sum -c -

RUN chown -R ${user} "$JENKINS\_HOME" /usr/share/jenkins/ref

RUN mkdir /var/log/jenkins

RUN mkdir /var/cache/jenkins

RUN chown -R ${user}:${user} /var/log/jenkins

RUN chown -R ${user}:${user} /var/cache/jenkins

# for main web and slave agents:

EXPOSE ${http\_port}

EXPOSE ${agent\_port}

# Copy in local config files

COPY init.groovy /usr/share/jenkins/ref/init.groovy.d/tcp-slave-agent-port.groovy

COPY jenkins-support /usr/local/bin/jenkins-support

COPY plugins.sh /usr/local/bin/plugins.sh

COPY jenkins.sh /usr/local/bin/jenkins.sh

COPY install-plugins.sh /usr/local/bin/install-plugins.sh

RUN chmod +x /usr/share/jenkins/ref/init.groovy.d/tcp-slave-agent-port.groovy \

&& chmod +x /usr/local/bin/jenkins-support \

&& chmod +x /usr/local/bin/plugins.sh \

&& chmod +x /usr/local/bin/jenkins.sh \

&& chmod +x /usr/local/bin/install-plugins.sh

COPY plugins.txt /usr/share/jenkins/ref/plugins.txt

RUN /usr/local/bin/install-plugins.sh < /usr/share/jenkins/ref/plugins.txt

USER ${user}

ARG BUILD\_DATE

ARG VCS\_REF

ARG VCS\_URL

ARG VERSION=0

LABEL \

# This label contains the Date/Time the image was built. The value SHOULD be formatted according to RFC 3339.

org.label-schema.build-date=$BUILD\_DATE \

#How to run a container based on the image under the Docker runtime.

org.label-schema.docker.cmd="docker run -d -p 8080:8080 -v \"$$(pwd)/jenkins-home:/var/jenkins\_home\" -v /var/run/docker.sock:/var/run/docker.sock workinghandguard/jenkins" \

#Text description of the image. May contain up to 300 characters.

org.label-schema.description="Jenkins with docker support, Jenkins ${JENKINS\_VER}, Docker ${DOCKER\_VER}" \

#A human friendly name for the image. For example, this could be the name of a microservice in a microservice architecture.

org.label-schema.name="workinghandguard/jenkins" \

#This label SHOULD be present to indicate the version of Label Schema in use.

org.label-schema.schema-version="1.0" \

#URL of website with more information about the product or service provided by the container.

org.label-schema.url="https://github.com/Protosaider/" \

#Identifier for the version of the source code from which this image was built. For example if the version control system is git this is the SHA.

org.label-schema.vcs-ref=$VCS\_REF \

#URL for the source code under version control from which this container image was built.

org.label-schema.vcs-url="https://github.com/Protosaider/" \

#The organization that produces this image.

org.label-schema.vendor="Fedor Ermolchev" \

#Release identifier for the contents of the image. This is entirely up to the user and could be a numeric version number like 1.2.3, or a text label.

#You SHOULD omit the version label, or use a marker like “dirty” or “test” to indicate when a container image does not match a labelled / tagged version of the code.

org.label-schema.version="${JENKINS\_NS}/${JENKINS\_REPO}:${JENKINS\_VER}-${VERSION}"

ENTRYPOINT ["/sbin/tini", "--", "/usr/local/bin/jenkins.sh"]

{Листинг 2: Dockerfile для Jenkins-nginx-proxy}

FROM nginx:latest

# Cleanup the default NGINX configuration file we don’t need

RUN rm /etc/nginx/conf.d/default.conf

# Copy the configuration file from the current directory and paste

# it inside the container to use it as Nginx's default config.

COPY jenkins.conf /etc/nginx/conf.d/jenkins.conf

COPY nginx.conf /etc/nginx/nginx.conf

# Port 80 of the container will be exposed and then mapped to port

# 8080 of our host machine via Compose. This way we'll be able to

# access the server via localhost:80 on our host.

# Listen on Port 80

EXPOSE 80

# same as -p 8080:80

# docker run --name some-nginx -d -p 8080:80 some-content-nginx

# Remove write access from config files to protect from accidental damage

RUN chmod -v 0444 /etc/nginx/conf.d/jenkins.conf && chmod -v 0444 /etc/nginx/nginx.conf

# Create volume to persist SSL data

VOLUME /etc/ssl/certs/nginx

# If you add a custom CMD in the Dockerfile, be sure to include -g daemon off; in the CMD in order for nginx to stay in the foreground, so that Docker can track the process properly (otherwise your container will stop immediately after starting)!

# Start Nginx when the container has provisioned.

CMD ["nginx", "-g", "daemon off;"]

{Листинг 3: Dockerfile для docker-proxy}

FROM buildpack-deps:stretch-scm

# Socat is a simple linux utility for transporting data between two byte streams.

RUN apt-get update \

    && DEBIAN\_FRONTEND=noninteractive apt-get install -y --no-install-recommends socat \

    && apt-get clean \

    && rm -rf /var/lib/apt/lists/\*

# We’re making a simple docker image to put our docker.sock from our desktop on one end and TCP port 2375 on the other.

# This is why the docker images is mounting a volume that contains the socket file, bridging desktop with docker network.

VOLUME /var/run/docker.sock

# The goal here is to take your docker.sock file and expose it on port 2375 securely and only to Jenkins.

# We need to do this because the Jenkins Docker plugin expects to talk over TCP/IP or HTTP to a port.

# In a production environment this would be some kind of Docker Swarm end point, but here on our local setup it’s just your desktop.

#

# We don’t want to expose that port on your desktop to your network. So once we have an image, we’re going to have it join our docker-network for Jenkins where it can keep that port private.

#

# docker tcp port

EXPOSE 2375

ENTRYPOINT ["socat", "TCP-LISTEN:2375,reuseaddr,fork", "UNIX-CLIENT:/var/run/docker.sock"]

{Листинг 4: Dockerfile для ephemeral-build-slave}

FROM jenkins/jnlp-slave:latest

USER root

RUN apt-get update \

    && apt-get install -y --no-install-recommends ca-certificates curl apt-transport-https openssh-client apt-transport-https software-properties-common apt-utils gnupg2

RUN apt-get update \

    && apt-get install -y --no-install-recommends python-pip python

RUN apt-get update \

    && DEBIAN\_FRONTEND=noninteractive apt-get install -y --no-install-recommends git \

wget \

sudo \

    && apt-get clean \

    && rm -rf /var/lib/apt/lists/\*

ENV DOCKER\_USER=docker

ENV DOCKER\_GROUP=docker

ARG uid=1000

ARG gid=100

RUN groupadd -g ${gid} ${DOCKER\_GROUP} -o

RUN adduser --uid ${uid} --gid ${gid} ${DOCKER\_USER}

RUN curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -

RUN add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/debian \

$(lsb\_release -cs) \

stable"

RUN apt-get update

RUN apt-get install -y docker-ce

# Define env variables and arguments

ENV JENKINS\_HOME=/home/jenkins

ENV JENKINS\_USER=jenkins

ARG uid=1001

ARG gid=1001

ARG shell=/bin/sh

RUN mkdir -p /home/jenkins

RUN adduser -h $JENKINS\_HOME -u ${uid} -G ${JENKINS\_USER} -s ${shell} -D ${JENKINS\_USER}

# Add the jenkins user to sudoers

RUN echo "${JENKINS\_USER} ALL=(ALL) ALL" >> /etc/sudoers

# Sometimes Docker containers struggled to route or resolve DNS names correctly, so I’ve taken to making sure the proper DNS servers are added to my build slaves.

# Set Name Servers

COPY resolv.conf /etc/resolv.conf

RUN chown -R $JENKINS\_USER $JENKINS\_HOME ${trustStore}

RUN usermod -a -G docker Jenkins

# GCLOUD

ARG CLOUD\_SDK\_VERSION=229.0.0

ENV CLOUD\_SDK\_VERSION=$CLOUD\_SDK\_VERSION

RUN apt-get -qqy update && apt-get install -qqy \

gcc \

python-dev \

python-setuptools \

apt-transport-https \

lsb-release \

gnupg \

&& easy\_install -U pip && \

pip install -U crcmod && \

export CLOUD\_SDK\_REPO="cloud-sdk-$(lsb\_release -c -s)" && \

echo "deb https://packages.cloud.google.com/apt $CLOUD\_SDK\_REPO main" > /etc/apt/sources.list.d/google-cloud-sdk.list && \

curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add - && \

apt-get update && \

apt-get install -y google-cloud-sdk=${CLOUD\_SDK\_VERSION}-0 \

google-cloud-sdk-app-engine-python=${CLOUD\_SDK\_VERSION}-0 \

google-cloud-sdk-app-engine-python-extras=${CLOUD\_SDK\_VERSION}-0 \

google-cloud-sdk-app-engine-java=${CLOUD\_SDK\_VERSION}-0 \

google-cloud-sdk-app-engine-go=${CLOUD\_SDK\_VERSION}-0 \

google-cloud-sdk-datalab=${CLOUD\_SDK\_VERSION}-0 \

google-cloud-sdk-datastore-emulator=${CLOUD\_SDK\_VERSION}-0 \

google-cloud-sdk-pubsub-emulator=${CLOUD\_SDK\_VERSION}-0 \

google-cloud-sdk-bigtable-emulator=${CLOUD\_SDK\_VERSION}-0 \

google-cloud-sdk-cbt=${CLOUD\_SDK\_VERSION}-0 \

kubectl && \

gcloud config set core/disable\_usage\_reporting true && \

gcloud config set component\_manager/disable\_update\_check true && \

gcloud config set metrics/environment github\_docker\_image && \

gcloud --version && \

docker --version && kubectl version --client

VOLUME ["/root/.config"]

# Switch to the jenkins user

USER ${JENKINS\_USER}

{Листинг 5: docker-compose.yml}

version: "3"

services:

jenkins-master:

build:

./jenkins-master

ports:

-"50000:50000"

volumes:

-jenkins-data:/var/jenkins\_home

-jenkins-log:/var/log/jenkins

networks:

-jenkins-net

restart:

unless-stopped

nginx-proxy:

build:

./nginx-proxy

ports:

-"80:80"

networks:

-jenkins-net

docker-proxy:

build:

./docker-proxy

volumes:

-/var/run/docker.sock:/var/run/docker.sock

networks:

-jenkins-net

aliases:

-proxy1

ephemeral-slave:

build:

./ephemeral-slave

volumes:

jenkins-data:

jenkins-log:

networks:

jenkins-net:

{Листинг 6: Jenkinsfile}

import net.sf.json.JSONArray;

import net.sf.json.JSONObject;

// @Library('jenkins-shared-library') \_

pipeline {

    agent {

        node {

            label 'ephemeral-slave'

        }

    }

    options {

        skipDefaultCheckout()

    //  disableConcurrentBuilds()

    //  timeout(time: 10, unit: 'MINUTES')

    //  buildDiscarder(logRotator(numToKeepStr: '10'))

        timestamps()

    }

    parameters {

        string(name: 'SLACK\_CHANNEL\_1',

                description: 'Default Slack channel to send messages to',

                defaultValue: '#build')

        string(name: 'SLACK\_CHANNEL\_2',

                description: 'Default Slack channel to send messages to',

                defaultValue: '#my\_channel\_2')

    }

    environment {

        CI = true

    }

    stages {

        stage('Checkout Git repository') {

            steps {

                checkout scm

                script {

                    def url = sh(returnStdout: true, script: "git remote get-url origin").trim()

                    def commit = sh(returnStdout: true, script: 'git rev-parse HEAD')

                def author = sh(returnStdout: true, script: "git --no-pager show -s --format='%an' ${commit}").trim()

                 def lastCommitMessage = sh(returnStdout: true, script: 'git log -1 --pretty=%B').trim()

                  long epoch = System.currentTimeMillis()/1000

                  String buildStatus = 'STARTED'

                  def color = '#D4DADF'

                  def subject = "\*${buildStatus}\*\nJob \_${env.JOB\_NAME}\_, build \_#${env.BUILD\_NUMBER}\_"

                  def msg = "${subject}\n More info at: ${env.BUILD\_URL}"

                    slackSend(color: color, message: msg, channel: "${params.SLACK\_CHANNEL\_2}")

                }

            }

        }

        stage('Environment') {

            steps {

                sh 'git --version'

                echo "Branch: ${env.BRANCH\_NAME}"

                sh 'docker -v'

                sh 'printenv|sort'

                sh 'env|sort'

            }

        }

        stage('Build Docker test') {

            steps {

                sh 'docker build --tag react-test --file Dockerfile.test --no-cache .'

            }

        }

        stage('Docker test') {

            steps {

                sh 'docker run --rm react-test'

            }

        }

        stage('Clean Docker test') {

            steps {

                sh 'docker rmi react-test'

            }

        }

        stage('Deploy') {

            when {

                branch 'master'

            }

            steps {

                sh 'docker build -t react-app --no-cache .'

                sh 'docker tag react-app localhost:5000/react-app'

                sh 'docker push localhost:5000/react-app'

                sh 'docker rmi -f react-app localhost:5000/react-app'

            }

        }

    }

    post {

        always {

            echo "Sending message to Slack"

            script {

                def url = sh(returnStdout: true, script: "git remote get-url origin").trim()

                def commit = sh(returnStdout: true, script: 'git rev-parse HEAD')

                def author = sh(returnStdout: true, script: "git --no-pager show -s --format='%an' ${commit}").trim()

               def lastCommitMessage = sh(returnStdout: true, script: 'git log -1 --pretty=%B').trim()

                long epoch = System.currentTimeMillis()/1000

                String buildStatus = currentBuild.result

                buildStatus = buildStatus ?: 'SUCCESS'

                def iconEmoji = ':jenkins\_ci:'

                def color

                if (buildStatus == 'STARTED' || buildStatus == 'ABORTED') {

                 color = '#D4DADF' // Grey

                } else if (buildStatus == 'SUCCESS') {

                 color = 'good'

                } else if (buildStatus == 'UNSTABLE' || buildStatus == 'INPUT REQUIRED' || buildStatus == 'NOT\_BUILT') {

                 color = 'warning'

                } else if (buildResult == 'FAILURE') {

                 color = 'danger'

                 iconEmoji = ':jenkins\_devil:'

                }

                else {

                 color = '#FF9FA1'

                 iconEmoji = ':jenkins\_devil:'

                }

                def subject = "\*STATUS: \_${buildStatus}\_\* ${iconEmoji}\nJob \_${env.JOB\_NAME}\_, build \_#${env.BUILD\_NUMBER}\_"

                def msg = "${subject}\n More info at: ${env.BUILD\_URL}"

                JSONArray attachments = new JSONArray();

                JSONObject attachment = new JSONObject();

                attachment.put('fallback', subject);

                attachment.put('title', ':github: Github info');

                attachment.put('color', color);

                JSONObject fieldBranch = new JSONObject();

                fieldBranch.put('title', 'Branch');

                fieldBranch.put('value', env.BRANCH\_NAME);

                fieldBranch.put('short', 'true');

                JSONObject fieldGitAuthor = new JSONObject();

                fieldGitAuthor.put('title', 'Author');

                fieldGitAuthor.put('value', author);

                fieldGitAuthor.put('short', 'true');

                JSONObject fieldLastCommitMessage = new JSONObject();

                fieldLastCommitMessage.put('title', 'Last commit');

                fieldLastCommitMessage.put('value', lastCommitMessage);

                fieldLastCommitMessage.put('short', 'true');

                JSONArray fields = new JSONObject();

                fields.add(fieldBranch);

                fields.add(fieldGitAuthor);

                fields.add(fieldLastCommitMessage);

                attachment.put('fields', fields);

                attachment.put('footer', url);

                attachment.put('footer\_icon', 'https://emojis.slackmojis.com/emojis/images/1501021339/341/git.png?1501021339');

                attachment.put('ts', epoch);

                attachments.add(attachment);

                slackSend(color: color, message: msg, channel: "${params.SLACK\_CHANNEL\_2}", attachments: attachments.toString())

            }

        }

    }

}

{Листинг 7: Dockerfile.test}

# Extending image

FROM node:carbon

RUN apt-get update

RUN apt-get upgrade -y

RUN apt-get -y install autoconf automake libtool nasm make pkg-config git apt-utils

# Create app directory

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

# Versions

RUN npm -v

RUN node -v

# Install app dependencies

COPY package.json /usr/src/app/

COPY package-lock.json /usr/src/app/

RUN npm install

# Bundle app source

COPY . /usr/src/app

# Main command

CMD [ "npm", "run", "test:ci" ]

{Листинг 8: Dockerfile}

## Extending image

FROM node:carbon

RUN apt-get update

RUN apt-get upgrade -y

RUN apt-get -y install autoconf automake libtool nasm make pkg-config git apt-utils

# Create app directory

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

# Versions

RUN npm -v

RUN node -v

# Install app dependencies

COPY package.json /usr/src/app/

COPY package-lock.json /usr/src/app/

RUN npm install

# Bundle app source

COPY . /usr/src/app

# Port to listener

EXPOSE 3000

RUN npm run build

# Main command

CMD [ "npm", "run", "start" ]