ИТМО

И.В. Ананченко, Т.В. Зудилова, С.Е. Иванов

КОНТЕЙНЕРИЗАТОР ПРИЛОЖЕНИЙ DOCKER — УСТАНОВКА, НАСТРОЙКА, ОСНОВЫ УПРАВЛЕНИЯ ПРИЛОЖЕНИЯМИ В СРЕДАХ С ПОДДЕРЖКОЙ КОНТЕЙНЕРИЗАЦИИ

Санкт-Петербург
2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

И.В. Ананченко, Т.В. Зудилова, С.Е. Иванов

КОНТЕЙНЕРИЗАТОР ПРИЛОЖЕНИЙ DOCKER — УСТАНОВКА, НАСТРОЙКА, ОСНОВЫ УПРАВЛЕНИЯ ПРИЛОЖЕНИЯМИ В СРЕДАХ С ПОДДЕРЖКОЙ КОНТЕЙНЕРИЗАЦИИ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлению подготовки 11.03.02 Инфокоммуникационные технологии и системы связи
в качестве Учебно-методическое пособие для реализации основных профессиональных образовательных программ высшего образования бакалавриата

ИТМО

Санкт-Петербург
2023

Ананченко И.В., Зудилова Т.В., Иванов С.Е., Контейнеризатор приложений docker — установка, настройка, основы управления приложениями в средах с поддержкой контейнеризации– СПб: Университет ИТМО, 2023. – 54 с.

Рецензент(ы):
Чумаков Сергей Иванович, к.т.н., доцент, доцент кафедры системного анализа и информационных технологий, ФГБОУ ВО "СПбГТИ(ТУ)";

Методическое пособие по работе с ПО Docker, программным обеспечением для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации, предназначено для студентов очной формы обучения, осваивающих профессиональную образовательную программу «Программирование в инфокоммуникационных системах» направления подготовки 11.03.02 Инфокоммуникационные технологии и системы связи. ОГНП «Трансляционные информационные технологии».
В пособии содержатся базовые инструкции и рекомендации по установке, настройке и основам управления приложениями в средах с поддержкой контейнеризации на примере использования ПО Docker.

Содержание

1. Введение. Знакомство с Docker

Контейнеризация — это отличная альтернатива аппаратной виртуализации. Все процессы в ней протекают на уровне операционной системы, это позволяет существенно экономить ресурсы и повышать эффективность работы с приложениями. Docker является одним из наиболее популярных инструментов для программной виртуализации. Docker — автоматизированное средство управления виртуальными контейнерами. Docker — это платформа с открытым исходным кодом, которая позволяет разработчикам создавать, развертывать, запускать, обновлять и управлять контейнерами - стандартизированными исполняемыми компонентами, объединяющими исходный код приложения с библиотеками операционной системы (ОС) и зависимостями, необходимыми для запуска этого кода в любой среде. Контейнеры упрощают разработку и доставку распределенных приложений. Они становятся все более популярными по мере того, как организации переходят на облачную разработку и гибридные мультиоблачные среды.

1.1 Реестры

Реестр Docker — это интерактивная библиотека, в которой хранятся образы контейнеров для развертывания и запуска в Docker. Одним из основных общедоступных реестров Docker является Docker Hub, где отдельные пользователи могут хранить, загружать изображения контейнеров и обмениваться ими.

1.2 Изображения контейнеров

Образ контейнера — это шаблон, доступный только для чтения, с инструкциями по созданию контейнера Docker. В контейнере Docker приложение запускается в среде выполнения с указанными настройками.

1.3 Контейнеры

Контейнер Docker — это экземпляр образа контейнера Docker во время выполнения, который изначально запускается в Linux и совместно использует ядро хост-машины с другими контейнерами Docker. Контейнер Docker запускает отдельный процесс, независимый от других контейнеров, используя не больше памяти, чем любой другой исполняемый файл, что делает его использование памяти легким. По сравнению с виртуальными машинами, которые запускают приложения на виртуальном оборудовании, работающем от операционной системы хоста сервера, контейнеры требуют меньше вычислительных ресурсов и наилучшим образом используют вычислительную мощность.

Контейнеры предлагают логический механизм упаковки, в котором приложения могут быть абстрагированы от среды, в которой они на самом деле работают. Таким образом, контейнерные приложения могут быть легко и последовательно развернуты в различных средах, таких как частный центр обработки данных, общедоступное облако или даже персональный ноутбук разработчика. Это позволяет разработчикам создавать предсказуемые среды, которые изолированы от других приложений и могут быть запущены в любом месте.

1.4 Контрольные вопросы

Что представляет собой контейнеризация?
Что такое Docker?
Что является реестром Docker?
Для чего необходим образ контейнера?
Что представляет собой контейнер Docker?
В каких средах можно использовать контейнерные приложения?

2. Использование Docker в качестве легкой виртуальной машины

Виртуальные машины (VM) стали повсеместными в разработке и развертывании программного обеспечения с начала века. Абстракция машин к программному обеспечению сделала перемещение и управление программным обеспечением и службами в эпоху интернета проще и дешевле.

Виртуальная машина – это приложение, которое эмулирует компьютер обычно для того, чтобы запустить операционную систему и приложения. Его можно разместить на любых (совместимых) доступных физических ресурсах.

Docker не является VM-технологией. Он не моделирует аппаратное обеспечение компьютера и не включает в себя операционную систему. Контейнер Docker по умолчанию не ограничен конкретными аппаратными ограничениями. Если Docker виртуализирует что-либо, он виртуализирует среду, в которой запускаются службы, а не компьютер. Более того, Docker не может с легкостью запускать программное обеспечение Windows (или даже ПО, написанное для других операционных систем Unix). Однако с некоторых точек зрения Docker можно использовать как виртуальную машину. Для разработчиков и тестировщиков в эпоху интернета тот факт отсутствия процесса инициализации или прямого взаимодействия с оборудованием обычно не имеет большого значения. И есть существенные общие черты, такие как его изоляция от окружающего оборудования и доступность для более тонких подходов к доставке программного обеспечения.

Вот некоторые различия между виртуальными машинами и контейнерами Docker:

- Docker ориентирован на приложения, в то время как виртуальные машины на операционные системы;
- контейнеры Docker совместно используют операционную систему вместе с другими контейнерами Docker. Напротив, у каждой виртуальной машины есть собственная операционная система, управляемая гипервизором;
- контейнеры Docker предназначены для запуска одного основного процесса, а не для управления несколькими наборами процессов.

Для того чтобы запустить виртуальную машину через докер на Windows, вам необходимо иметь один из следующих драйверов:

- Amazon Web Services
- Microsoft Azure
- DigitalOcean
- Exoscale
- Google Compute Engine
- Microsoft Hyper-V
- OpenStack
- Rackspace
- IBM Softlayer
- Oracle VirtualBox
- VMware vCloud Air
- VMware Fusion
- VMware vSphere

Мы будем использовать на примере таких драйверов – Oracle VirtualBox и Microsoft Hyper-V. Так же нам понадобится Docker-machine.

Загрузите двоичный файл Docker Machine и извлеките его в свою папку PATH.

Если используете Windows с Git BASH

base=https://github.com/docker/machine/releases/download/v0.16.0 &&

mkdir -p "$HOME/bin" &&

curl -L $base/docker-machine-Windows-x86_64.exe > "$HOME/bin/docker-machine.exe" &&

chmod +x "$HOME/bin/docker-machine.exe"

В противном случае загрузите один из выпусков напрямую со страницы выпуска докер-машины:

https://github.com/docker/machine/releases/download/v0.16.2/docker-machine-Windows-x86_64.exe для Windows или https://github.com/docker/machine/releases/ для выбора нужной версии.

Docker для создания стандартной виртуальной машины использует так называемую boot2docker систему. Boot2Docker — это облегченный дистрибутив Linux, созданный специально для запуска контейнеров Docker. Он полностью работает из ОЗУ, и весит порядка 45 МБ. Boot2Docker используется через Docker Machine (установленную как часть Docker Toolbox), которая использует VirtualBox для инициализации, запуска, остановки и удаления виртуальной машины прямо из командной строки.

2.1 Oracle VirtualBox

После скачивания откройте командную строку и перейдите в папку с файлом docker-machine.exe (если не используется Git BASH).

1. Для того чтобы запустить стандартный образ boot2docker введите команду в командную строку:
   docker-machine create --driver virtualbox default
   где
   driver – указание используемого драйвера (в нашем случае это virtualbox),
   default – название вашей виртуальной машины

Пометка:

Если у вас выдает ошибку «This computer doesn't have VT-X/AMD-v enabled. Enabling it in the BIOS is mandatory» и вы включили средство виртуализации в BIOS, то используйте настройку «--virtualbox-no-vtx-check», т. е. команда будет выглядеть так:

docker-machine create --driver virtualbox --virtualbox-no-vtx-check default

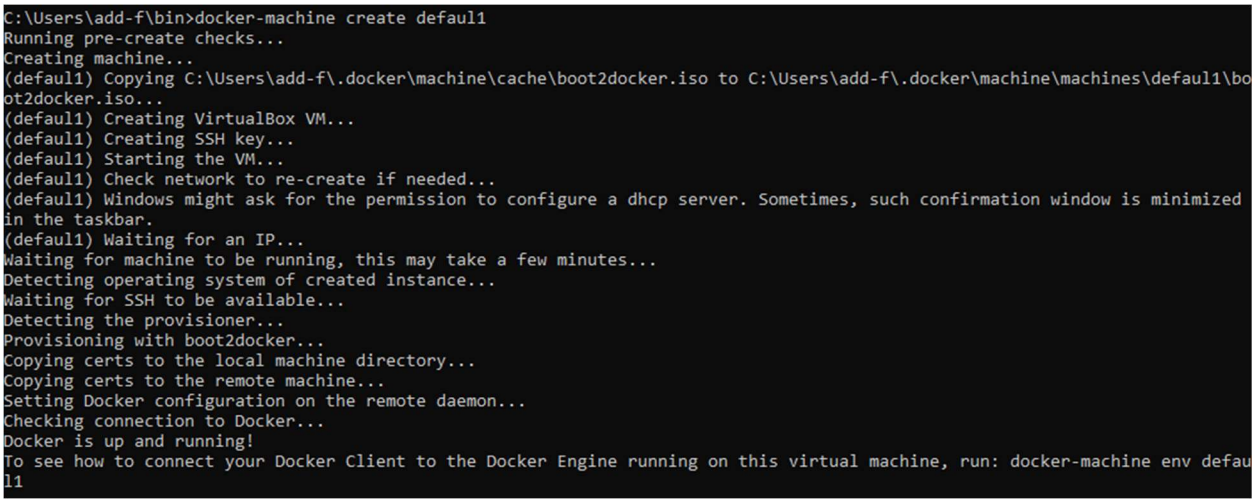

Рисунок 1 – Результат выполнения команды

Виртуальная машина успешно создана и запущена.

2. Как указано в выводе команды, вам нужно сообщить Docker, чтобы он связался с новой машиной. Сделать это можно с помощью docker-machine env:
   docker-machine env default

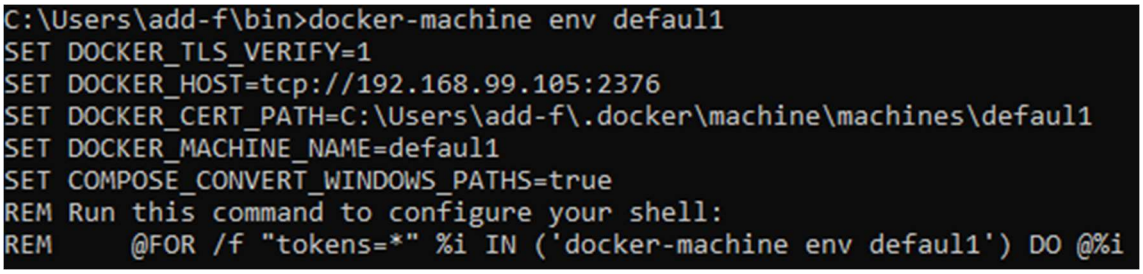

Рисунок 2 – Результат выполнения команды

3. Подключите вашу оболочку к новой машине командой:
   @FOR /f "tokens=*" %i IN ('docker-machine env default') DO @%i

Отлично, машина создана и настроена. Теперь для подключения к ней через командную строку введите:

docker-machine ssh default

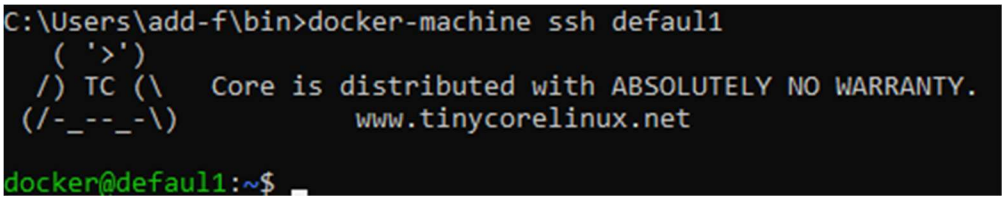

Рисунок 3 – Результат выполнения команды

Boot2Docker разработан и настроен для разработки. Использование его для любых производственных рабочих нагрузок крайне не рекомендуется.

Помимо провайдера, у вас есть возможность указать базовую операционную систему, поскольку Docker Machine имеет значения по умолчанию как для локальных, так и для удаленных провайдеров. Для локальных поставщиков, таких как VirtualBox, Hyper-V и т. д., базовой операционной системой по умолчанию является Boot2Docker. Для облачных провайдеров базовой операционной системой является последняя версия Ubuntu LTS, поддерживаемая провайдером.

Для использования другой операционной системы используется опция **--virtualbox-boot2docker-url**. Рассмотрим на примере RancherOS.

RancherOS — это самый компактный и простой способ запуска Docker в производственной среде. Каждый процесс в RancherOS — это контейнер, управляемый Docker. Сюда входят системные службы, такие как udev и syslog. Поскольку RancherOS включает только службы, необходимые для запуска Docker, она значительно меньше, чем большинство традиционных операционных систем. Благодаря удалению ненужных библиотек и служб также снижаются требования к исправлениям безопасности и другому обслуживанию. Это возможно, потому что с Docker пользователи обычно упаковывают все необходимые библиотеки в свои контейнеры. Все в RancherOS представляет собой контейнер Docker. Это достигается запуском двух экземпляров Docker. Один из них называется System Docker и является первым процессом в системе. Все остальные системные службы, такие как ntpd, syslog и console, работают в контейнерах Docker. System Docker заменяет традиционные системы инициализации systemd и используется для запуска дополнительных системных служб. System Docker запускает специальный контейнер под названием Docker, который является еще одним демоном Docker, отвечающим за управление всеми контейнерами пользователя. Любые контейнеры, которые вы запускаете как пользователь из консоли, будут работать внутри этого Docker. Это создает изоляцию от контейнеров System Docker и гарантирует, что обычные пользовательские команды не влияют на системные службы.

Для установки RancherOS введите следующую команду в командную строку:

docker-machine create -d virtualbox --virtualbox-boot2docker-url “https://releases.rancher.com/os/latest/rancheros.iso” --virtualbox-memory “2048” testrancher

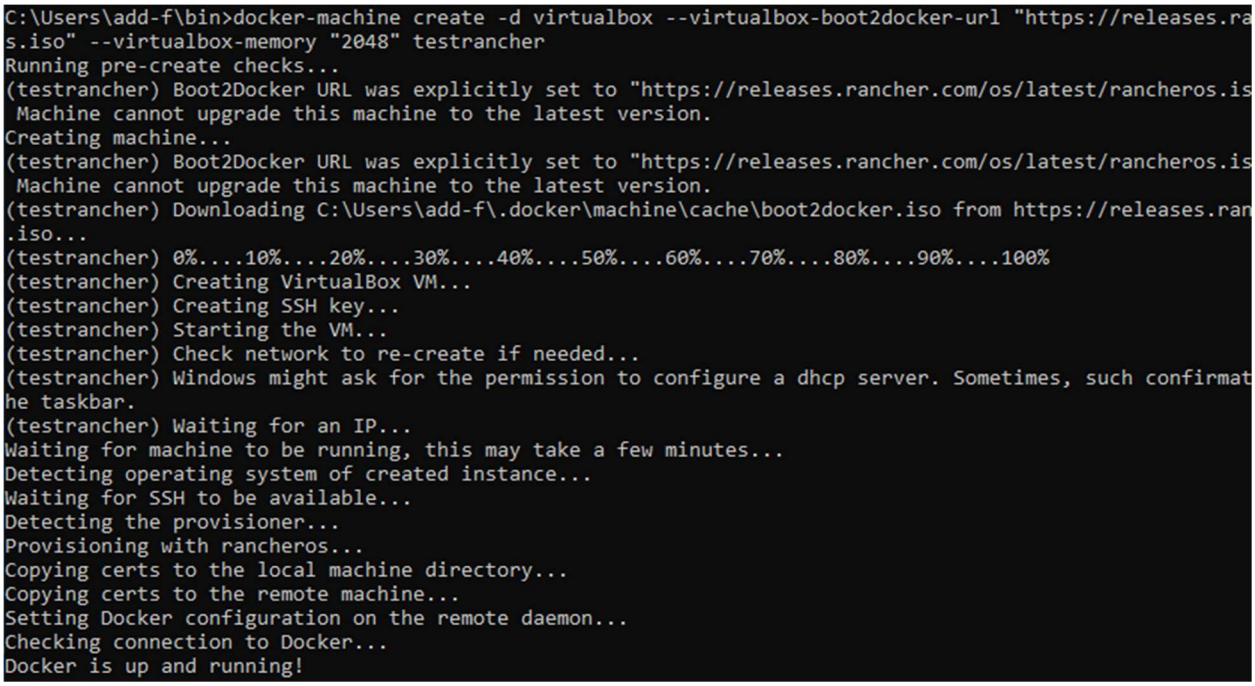

Рисунок 4 – Результат выполнения команды

Далее те же команды, что использовали для создания стандартного образа boot2docker:

docker-machine env testrancher

@FOR /f "tokens=*" %i IN ('docker-machine env testrancher') DO @%i

docker-machine ssh testrancher

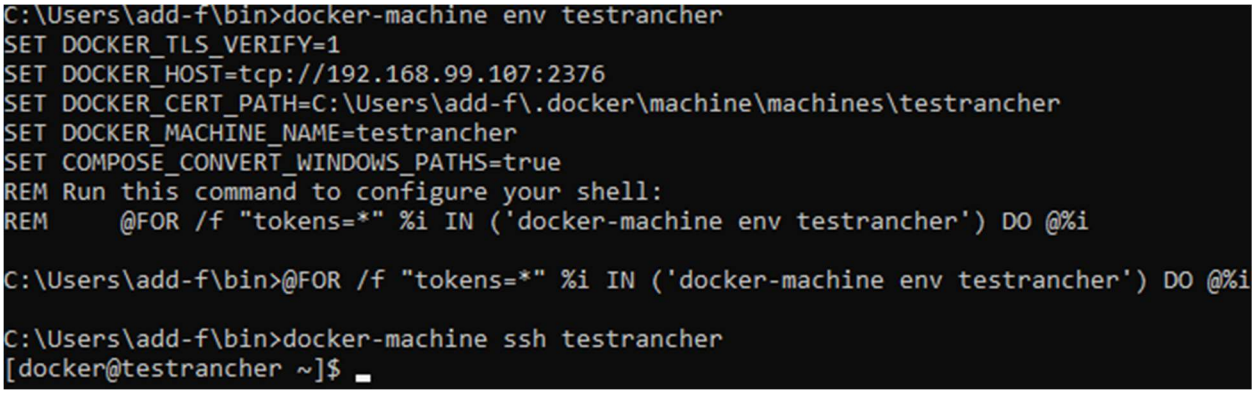

Рисунок 5 – Результат выполнения команды

При первом запуске RancherOS в демоне Docker не запущены контейнеры. Однако, если вы запустите ту же команду в System Docker, увидите ряд системных служб, поставляемых с RancherOS:

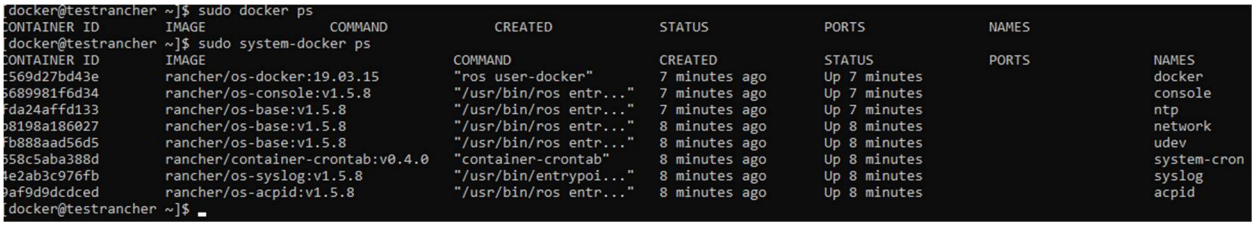

Рисунок 6 – Системные службы, запущенные RancherOS

Ниже приведен простой контейнер Docker для настройки Linux-dash, который представляет собой веб-панель с минимальными накладными расходами для мониторинга серверов Linux. Чтобы запустить этот контейнер в System Docker, используйте следующую команду:

sudo system-docker run -d --net=host --name busydash husseingalal/busydash

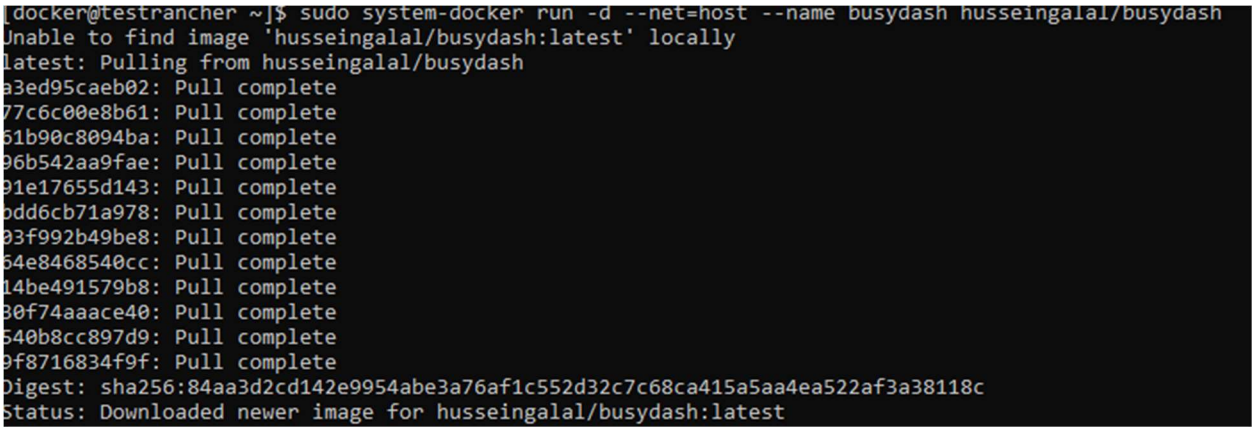

Рисунок 7 – Результат выполнения команды

В команде **--net=host** указываем System Docker не контейнеризовать сеть контейнера, а вместо этого использовать сеть хоста. После запуска контейнера можно увидеть сервер мониторинга, перейдя по ip вашего сервера. Для того, чтобы узнать ip введите

ip config

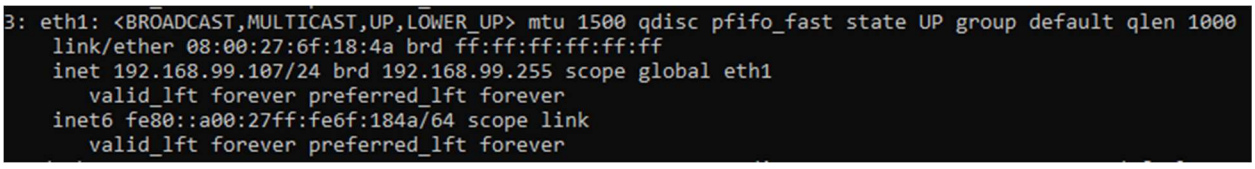

Рисунок 8 – Данные о сети, в которой находится пользователь

В нашем случае это 192.168.99.107

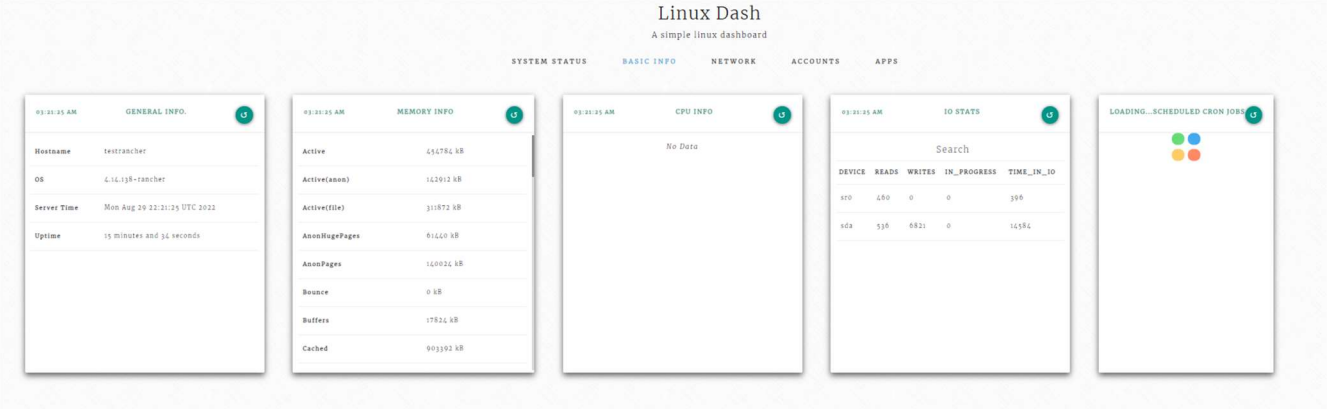

Рисунок 9 – Веб-панель с мониторингом ресурсов машины

Чтобы контейнер выжил во время перезагрузки, можно создать **/opt/rancher/bin/start.sh** скрипт и добавить стартовую строку Docker для запуска Docker при каждой загрузке системы. Введите следующие команды:

sudo mkdir -p /opt/rancher/bin

echo "sudo system-docker start busydash" | sudo tee -a /opt/rancher/bin/start.sh

sudo chmod 755 /opt/rancher/bin/start.sh

sudo reboot

и после перезагрузки (придется немного подождать) вновь команду:

docker-machine ssh testrancher

Вновь перейдя по нашему ip, можно увидеть ту же самую картину. Скрипт сработал успешно.

Вы также можете подключить другие операционные системы и настроить их с помощью Docker. Для этого замените ссылку для скачивания вашего образа ОС в опции **--virtualbox-boot2docker-url** или введите локальный путь на вашей хост-машине. Например, для установки Ubuntu 14.04 команда будет выглядеть следующим образом:

docker-machine create -d virtualbox --virtualbox-boot2docker-url “https://releases.ubuntu.com/14.04/ubuntu-14.04.6-desktop-amd64.iso” --virtualbox-memory “2048” --virtualbox-ui-type “gui” <Название вашей машины>

Опция **--virtualbox-ui-type** отвечает за тип пользовательского интерфейса. По умолчанию используется флаг “headless”. Возможны такие ключи, как gui, sdl, headless, separate.

2.2 Microsoft Hyper-V

1. НАСТРОЙТЕ НОВЫЙ ВНЕШНИЙ СЕТЕВОЙ КОММУТАТОР
   1.1 Откройте диспетчер Hyper-V.
   1.2 Выберите Диспетчер виртуальных коммутаторов на правой панели Действия.

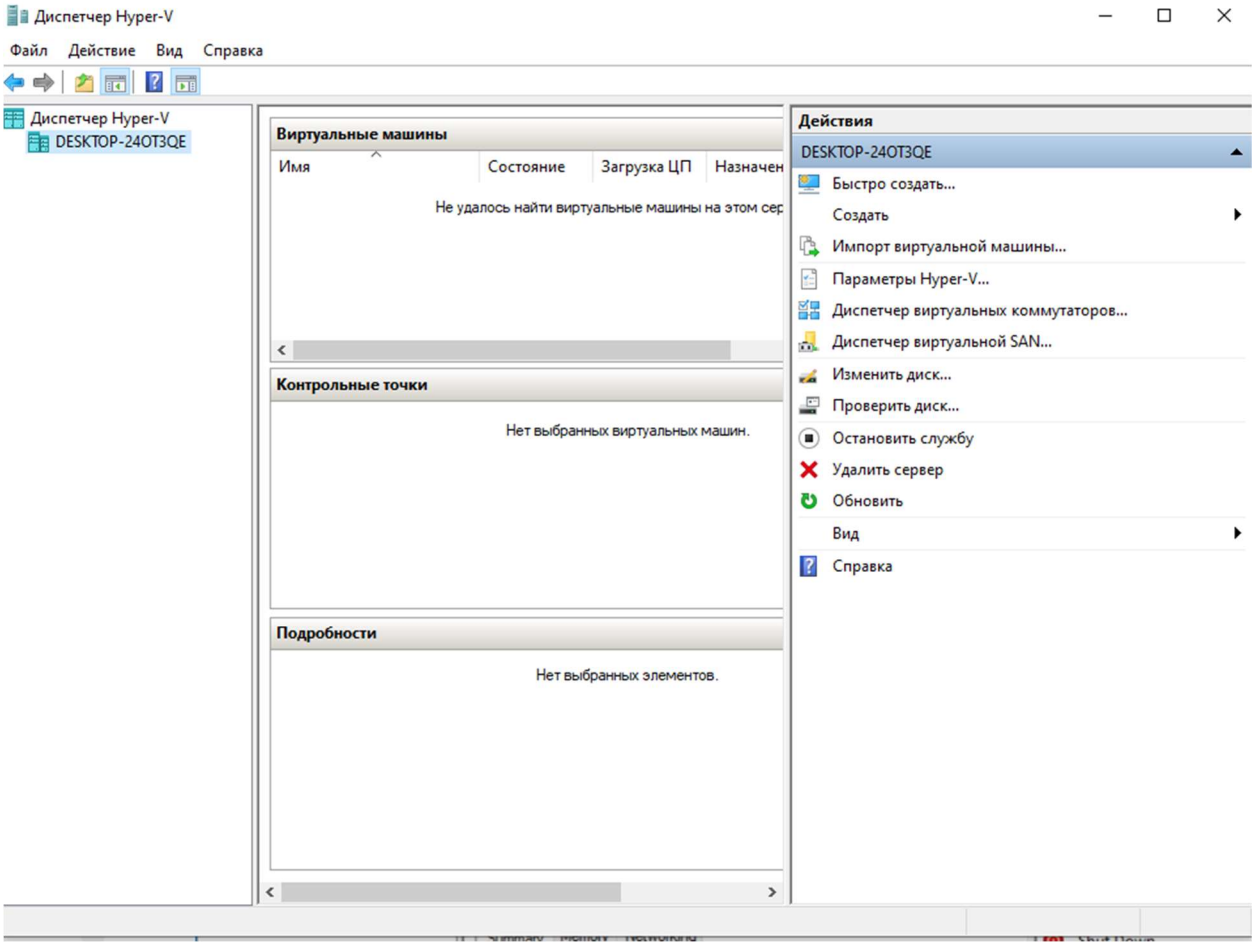

Рисунок 10 – Окно диспетчера Hyper-V

1.3 Создайте внешний виртуальный коммутатор и настройте его

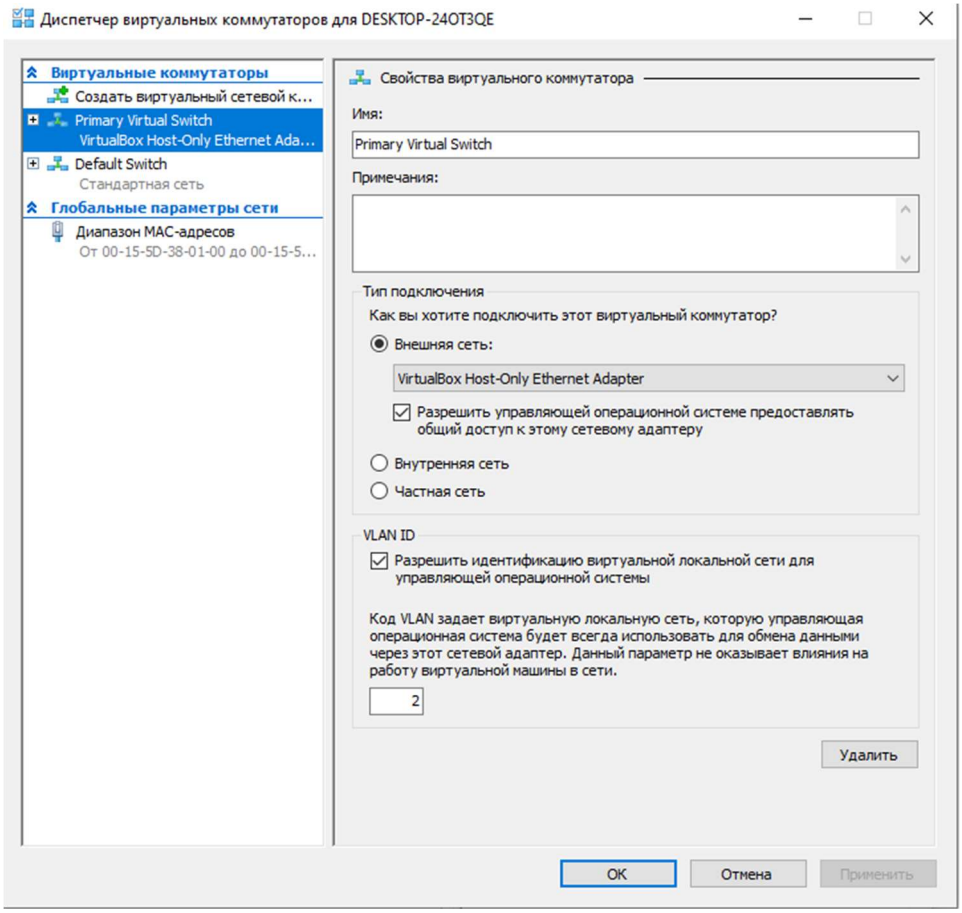

Рисунок 11 – Окно создания и настройки виртуальных коммутаторов

1.4. Перезагрузите устройство

2. Создайте машину с помощью docker-machine и драйвера hyper-v, с помощью следующей команды:

docker-machine create -d hyperv --hyperv-virtual-switch <НазваниеВашегоКоммутатора> <НазваниеМашины>

Если все правильно сделано, выведется следующее:

C:\Users\add-f\bin>docker-machine create -d hyperv --hyperv-virtual-switch "Primary Virtual Switch" manager
Running pre-create checks...
Creating machine...
(manager) Copying C:\Users\add-f\.docker\machine\cache\boot2docker.iso to C:\Users\add-f\.docker\machine\machines\manager\boot2docker.iso...
(manager) Creating SSH key...
(manager) Creating VM...
(manager) Using switch "Primary Virtual Switch"
(manager) Creating VHD
(manager) Starting VM...

(manager) Waiting for host to start...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: C:\Program Files\Doc
ker\Docker\Resources\bin\docker-machine.exe env manager1
PS C:\WINDOWS\system32>

Если не произошло автоматическое открытия терминала boot2docker, то введите следующее:

docker-machine ssh manager

Так же, как и для драйвера Oracle VirtualBox, у Hyper-V есть определенные опции, но их гораздо меньше:

--hyperv-boot2docker-url: URL-адрес ISO-образа ОС.

--hyperv-virtual-switch: имя используемого виртуального коммутатора.

--hyperv-disk-size: Размер диска в МБ.

--hyperv-memory: Размер памяти в МБ.

--hyperv-cpu-count: количество процессоров для ВМ.

Одно из ключевых отличий между контейнером Docker и виртуальной машиной заключается в том, что контейнер предназначен для запуска одного процесса. Когда этот процесс заканчивается, контейнер завершает работу. Это отличается от виртуальной машины Linux (или любой операционной системы) тем, что в ней нет процесса инициализации. Именно поэтому контейнеры docker не используются как полноценные виртуальные машины. Docker позволяет легко и быстро настроить и развернуть ОС, используя предустановленные драйверы для этого. На практике использование докера в качестве VM применяется лишь для оценивания поведения разрабатываемого ПО в другой системе, выполнения дальнейших

итераций с ним или извлечения выгоды из экосистемы Docker, используя инструменты и связанные с этим технологии.

2.3 Контрольные вопросы

Что такое виртуальная машина?

Является ли Docker VM-технологией?

Какме различия между виртуальными машинами и контейнерами Docker?

Какие драйвера нужны, чтобы запустить виртуальную машину через докер на Windows?

Какую систему использует Docker для создания стандартной виртуальной машины?

Для чего применяется RancherOS?

Как создать внешний виртуальный коммутатор и настройте его?

Какое ключевое отличие между контейнером Docker и виртуальной машиной?

3. Работа с контейнерами и изображениями

3.1 Зачем использовать контейнеры?

Контейнеры предлагают способ упаковки, с помощью которого приложения могут быть ограждены от среды, в которой они фактически выполняются. Такое разделение позволяет легко и последовательно развертывать приложения на основе контейнеров, независимо от того, является ли целевая среда ЦОД, облаком или даже личным ноутбуком. Это дает разработчикам возможность создавать предсказуемые среды, которые изолированы от остальных приложений и могут быть запущены в любом месте.

С точки зрения операций, помимо переносимости, контейнеры также обеспечивают более детальный контроль над ресурсами, что может привести к лучшему использованию ваших вычислительных ресурсов.

3.2 Практическое задание «Игра с Busybox»

Запустите контейнер Busybox, для этого введите в терминал

$ docker pull busybox

Примечание: В зависимости от того, как был установлен docker в системе, можно увидеть «в разрешении отказано» после выполнения приведенной выше

команды. Если используется компьютер Mac, убедитесь, что движок Docker запущен. Если используется Linux, то начните ввод команды с sudo.

Команда **pull** позволяет загрузить изображение из реестра Docker и сохранить на ПК. Посмотреть информацию об изображениях можно с помощью команды

$ docker images

Теперь запустите контейнер Docker из этого изображения. Для этого используется docker run команда.

$ docker run busybox

Можно заметить, что никаких видимых изменений не произошло. Однако это не так. На самом деле в фоновом режиме были выполнены следующие действия: когда используется команда run, клиент Docker находит изображение (в данном случае busybox), загружает контейнер и затем запускает команду в этом контейнере. Когда была вызвана команда docker run busybox, никакая команда для исполнения внутри контейнера не была предоставлена, поэтому контейнер загрузился, выполнил пустую команду и затем вышел. Для того, чтобы увидеть, что контейнер действительно запускается, добавим какое-нибудь действие. Пусть это будет вывод «Hello, world!». Для этого используется команда

docker run busybox echo “Hello, world!”

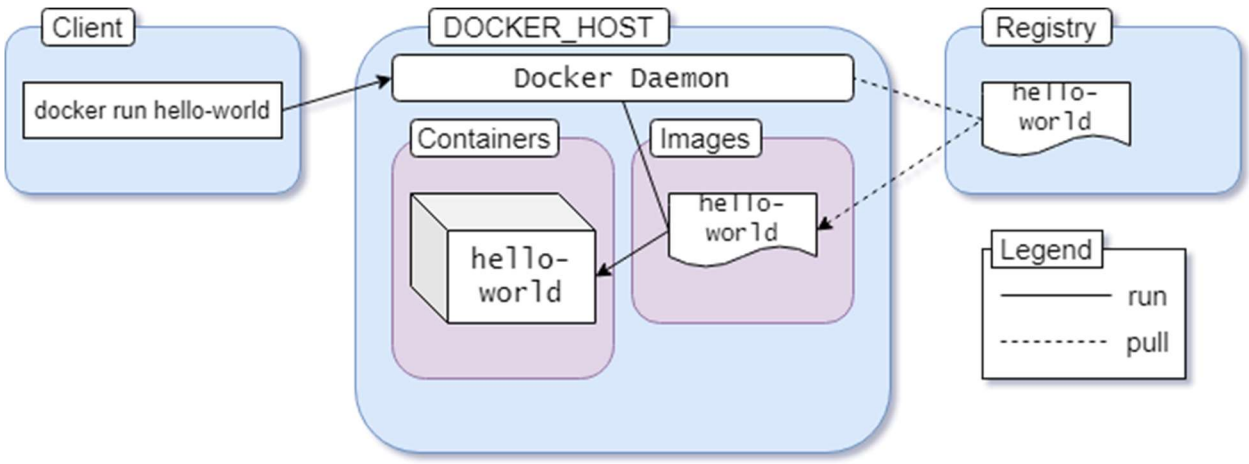

Рисунок 12 – Схема запуска и работы контейнера

Наконец-то можно обнаружить какой-то результат. В этом случае клиент Docker запустил echo команду в контейнере busybox, а затем вышел из него. Все это произошло довольно быстро. Аналогичный процесс, проделанный с помощью

виртуальной машины, занял бы куда больше времени. Именно поэтому говорят, что контейнеры — это быстро.

Команда **$ docker ps** показывает список всех запущенных. Поскольку контейнеры не запущены, то данная команда возвращает пустую строку. Для того, чтобы посмотреть полный список даже не запущенных контейнеров, используется команда: **$ docker ps -a**

Итак, то, что возвращает данная команда, — это список всех контейнеров, которые были запущены. Обратите внимание, что столбец Status показывает, что эти контейнеры завершены несколько минут назад.

С помощью команды **$ docker run -it busybox sh** можно выполнять несколько команд в одном контейнере. Для того, чтобы покинуть контейнер, нужно ввести команду exit.

Для того, чтобы удалить контейнер, необходимо использовать команду **$ docker rm ID,** где ID – это уникальный идентификатор контейнера. Чтобы удалить все завершенные контейнеры, нужно выполнить команду **$docker rm $(docker ps -a -q -f status=exited)**. В последних версиях Docker команду **docker container prune** можно использовать для достижения того же эффекта.

Запуск контейнеров

В качестве примера будем использовать готовый образ fhsinchy/hello-dock из репозитория Docker. Этот образ содержит простое приложение Vue.js , которое работает на порту 80 внутри контейнера. Чтобы запустить контейнер с использованием этого образа, необходимо выполнить следующую команду в терминале:

docker container run --publish 8080:80 fhsinchy/hello-dock

Контейнеры — это изолированные среды. Хост-система не может получить информацию о том, что происходит внутри контейнера. Следовательно, все, что запущено внутри контейнера, не может быть получено за пределами этого контейнера. Но чтобы разрешить доступ извне контейнера, необходимо объявить соответствующий порт внутри контейнера для локальной сети. Общий синтаксис для параметра –publish или -p выглядит следующим образом:

--publish <порт хоста>:<порт контейнера>

Команда --publish 8080:80 означает, что любой запрос, отправленный на порт 8080 хост-системы, будет перенаправлен на порт 80 внутри контейнера.

Теперь, чтобы получить доступ к приложению в браузере, необходимо ввести в адресную строку http://127.0.0.1:8080.

Контейнер можно остановить, просто нажав ctrl + c, когда окно терминала находится в фокусе, или полностью закрыв окно терминала.

Важным параметром команды run является параметр –detach или -d. В обычном режиме для работы контейнера необходимо держать окно терминала открытым. Это связано с тем, что по умолчанию контейнеры запускаются на переднем плане и присоединяются к терминалу, как и любая другая обычная программа, вызываемая из терминала. Чтобы изменить это поведение и сохранить работу контейнера в фоновом режиме, используется –detach параметр:

docker container run --detach --publish 8080:80 fhsinchy/hello-dock

В результате выполнения команды будет возвращен идентификатор только что созданного контейнера.

Порядок передачи параметров не имеет значения. Если поместить --publish перед --detach, команда будет работать точно так же. Одна вещь, которую необходимо знать при использовании команды run - имя изображения должно стоять последним. Если поместить что-либо после имени изображения, это будет передано в качестве аргумента точке входа контейнера и может привести к непредвиденным ситуациям.

3.3 Переименование контейнеров

По умолчанию каждый контейнер имеет два идентификатора.

- CONTAINER ID - случайная строка длиной 64 символа.
- NAME - комбинация двух случайных слов, соединенных знаком подчеркивания.

Ссылаться на контейнер на основе этих двух случайных идентификаторов довольно неудобно. Было бы здорово, если бы на контейнеры можно было ссылаться, используя имя, определенное пользователем.

Имя контейнера может быть задано с помощью параметра --name. Чтобы запустить другой контейнер, используя fhsinchy/hello-dock образ с именем hello-dock-container, необходимо выполнить следующую команду:

docker container run --detach --publish 8888:80 --name hello-dock-container fhsinchy/hello-dock

Порт 8080 в локальной сети занят другим контейнером (контейнер, созданный в предыдущем подразделе), поэтому придется использовать другой номер порта, например 8888. Теперь для проверки выполните container ls (аналог команды ps):

docker container ls

Новый контейнер с именем hello-dock-container был запущен.

Также можно переименовать старые контейнеры с помощью команды container rename. Синтаксис команды следующий:

docker container rename <идентификатор контейнера> <новое имя>

Команда не дает никакого вывода, но можно убедиться, что изменения произошли с помощью container ls команды. Команда rename работает для контейнеров как в запущенном, так и в остановленном состоянии.

3.4 Остановка контейнеров

Контейнеры, работающие на переднем плане, можно остановить, просто закрыв окно терминала или нажав комбинацию клавиш ctrl + c.

Однако контейнеры, работающие в фоновом режиме, не могут быть остановлены таким же образом.

Есть две команды, которые решают эту задачу. Первая команда – container stop. Общий синтаксис команды следующий:

docker container stop <идентификатор контейнера>, где container identifier может быть идентификатором или именем контейнера. Чтобы остановить контейнер, запущенный в предыдущем примере, выполните команду:

docker container stop hello-dock-container

Если используется имя в качестве идентификатора, то в качестве вывода будет получено это имя. Команда stop «аккуратно» закрывает контейнер, отправляя SIGTERM сигнал. Если контейнер не останавливается в течение определенного времени, посылается SIGKILL сигнал, который немедленно останавливает контейнер.

В случаях, когда нужно послать SIGKILL сигнал вместо SIGTERM сигнала, можно использовать команду container kill вместо stop. Команда container kill следует тому же синтаксису, что и stop команда.

docker container kill hello-dock-container-2

Перезапуск контейнеров

- Перезапуск контейнера, который ранее был остановлен или убит.
- Перезагрузка работающего контейнера.

Команду container start можно использовать для запуска любого остановленного или убитого контейнера. Синтаксис команды следующий:

docker container start <идентификатор контейнера>

Теперь в сценариях, где нужно перезагрузить работающий контейнер, можно использовать команду container restart. Команда container restart следует точному синтаксису container start.

docker container restart <идентификатор контейнера>

Основное различие между двумя командами заключается в том, что container restart останавливает целевой контейнер, а затем снова запускает его, тогда как команда container start просто запускает уже остановленный контейнер.

3.5 Создание контейнеров без запуска

До сих пор в этом разделе контейнеры запускали с помощью container run, которая на самом деле представляет собой комбинацию двух отдельных команд. Эти команды следующие:

- container create - команда создает контейнер из заданного образа.
- container start - команда запускает уже созданный контейнер.

С помощью этих двух команд можно сделать что-то вроде следующего:

docker container create --publish 8080:80 fhsinchy/hello-dock

docker container start hello-dock

Если после каждой команды запустить ls -all, то можно увидеть, как меняется статус с Created на Up.

3.6 Запуск контейнеров в интерактивном режиме

На данный момент были запущены только контейнеры, созданные либо из образа hello-world, либо из образа fhsinchy/hello-dock. Эти образы созданы для выполнения простых программ, которые не являются интерактивными.

Но не все изображения так просты. Образы могут инкапсулировать в себе весь дистрибутив Linux. Популярные дистрибутивы, такие как Ubuntu, Fedora, Debian, имеют официальные образы Docker, доступные в хабе. Языки программирования, такие как python, php, go или среды выполнения, такие как node, deno имеют свои официальные образы.

Эти образы не просто запускают какую-то заранее настроенную программу. Вместо этого они настроены на запуск оболочки по умолчанию. В случае образов операционной системы это может быть что-то вроде sh или, bash, а в случае языков программирования или среды выполнения — это обычно их языковая оболочка по умолчанию.

Изображения, сконфигурированные для запуска какой-либо оболочки, являются интерактивными изображениями. Эти изображения требуют специальной -it опции для передачи в container run команду.

Например, если запустить контейнер с использованием ubuntu образа, с помощью команды docker container run ubuntu, ничего не произойдет. Но если выполнить ту же команду с -it опцией, то станет доступен bash внутри контейнера Ubuntu.

docker container run --rm -it ubuntu

Этот -it параметр позволяет взаимодействовать с любой интерактивной программой внутри контейнера. Этот вариант на самом деле представляет собой два отдельных параметра, объединенных вместе.

- Параметр -i или –interactive подключает к входному потоку контейнера, чтобы можно было отправлять входные данные в bash.
- Параметр -t или –tty гарантирует, что встроенный в оболочку родной терминал, например, за счет выделения псевдотерминала.

Нужно использовать эту -it опцию всякий раз, когда необходимо запустить контейнер в интерактивном режиме. Другим примером может быть запуск node образа следующим образом:

docker container run -it node

Вместо того, чтобы писать -it, можно написать --interactive --tty отдельно.

3.7 Основы работы с изображениями

Рекомендуется установить Visual Studio Code с официальным расширением Docker из магазина.

3.7.1 Создание изображений

Образы — это многоуровневые автономные файлы, которые действуют как шаблон для создания контейнеров Docker. Они похожи на замороженную копию контейнера, доступную только для чтения.

Чтобы создать изображение с помощью одной из ваших программ, вы должны иметь четкое представление о том, что хотите от изображения. Возьмем, к примеру, официальный образ nginx. Можно запустить контейнер, используя этот образ, просто выполнив следующую команду:

docker container run --rm --detach --name default-nginx --publish 8080:80 nginx

Теперь по адресу http://127.0.0.1:8080 будет страница ответа по умолчанию.

Также можно создать собственный образ NGINX, который работает точно так же, как официальный. Чтобы создать собственный образ NGINX, нужно иметь четкое представление о том, каким будет конечное состояние образа. Изображение должно быть следующим:

- В образе должен быть предустановлен NGINX, что можно сделать с помощью менеджера пакетов или собрать из исходного кода.
- Образ должен запускать NGINX автоматически при запуске.

3.7.2 Практическое задание «создание образа NGINX»

Если был клонирован репозиторий проекта, зайдите в корень проекта и найдите там каталог с указанным именем custom-nginx. Создайте новый файл с именем Dockerfile внутри этого каталога. Dockerfile — это набор инструкций, которые после обработки демоном превращаются в образ. Содержание для этого Dockerfile следующее:

```
FROM ubuntu:latest
EXPOSE 80
RUN apt-get update && \
apt-get install nginx -y && \
apt-get clean && rm -rf /var/lib/apt/lists/*
CMD ["nginx", "-g", "daemon off;"]
```

Изображения представляют собой многослойные файлы, и в этом файле каждая написанная строка (известная как инструкции) создает слой для изображения.

- Каждый Dockerfile начинается с FROM инструкции. Эта инструкция устанавливает базовое изображение для результирующего изображения.

Установив ubuntu:latest, будут получены все преимущества Ubuntu, уже доступные в пользовательском образе, поэтому можно использовать такие вещи, как apt-get команда для простой установки пакета.

- Инструкция EXPOSE используется для указания порта, который необходимо опубликовать. Использование этой инструкции не означает, что не понадобится –publish порт. Все равно нужно будет явно использовать эту опцию. Эта инструкция работает как документация для тех, кто пытается запустить контейнер, используя ваш образ.
- Инструкция RUN в Dockerfile выполняет команду внутри оболочки контейнера. Команда *apt-get update && apt-get install nginx -y* проверяет наличие обновленных версий пакетов и устанавливает NGINX. Команда apt-get clean && rm -rf /var/lib/apt/lists/* используется для очистки кеша пакетов, потому что не нужно, чтобы в изображении был ненужный набор информации. Эти две команды — простые вещи Ubuntu, ничего особенного. Инструкции RUN здесь, написаны в shell форме. Они также могут быть записаны в exec форме.
- Наконец, CMD инструкция устанавливает команду по умолчанию для изображения. Эта инструкция написана в exec форме, состоящей из трех отдельных частей. Здесь nginx относится к исполняемому файлу NGINX. И -g -daemon off — это параметры для NGINX. Запуск NGINX как отдельного процесса внутри контейнеров считается лучшей практикой, поэтому используется этот параметр. Инструкция CMD также может быть написана в shell форме. Более подробно об отличиях можно узнать из официальной документации.

Теперь, когда был создан рабочий Dockerfile, можно создать из него изображение. Как и команды, связанные с контейнером, команды, связанные с изображением, могут быть выполнены с использованием следующего синтаксиса:

docker image <command> <options>

Чтобы создать образ с помощью Dockerfile, откройте терминал внутри custom-nginx каталога и выполните следующую команду:

docker image build .

Чтобы выполнить сборку образа, демону нужны два параметра. Это имя Dockerfile и каталог сборки. В команде выше:

- docker image build – это команда для создания образа. Демон находит любой файл с именем Dockerfile в каталоге.

- В конце задается каталог для этой сборки. В данном примере используется специальный символ «.», который означает использование нынешнего каталога

Теперь, чтобы запустить контейнер с использованием этого образа, можно использовать container run в сочетании с идентификатором образа, который был получен в результате процесса сборки. Чтобы проверить результат, посетите http://127.0.0.1:8080

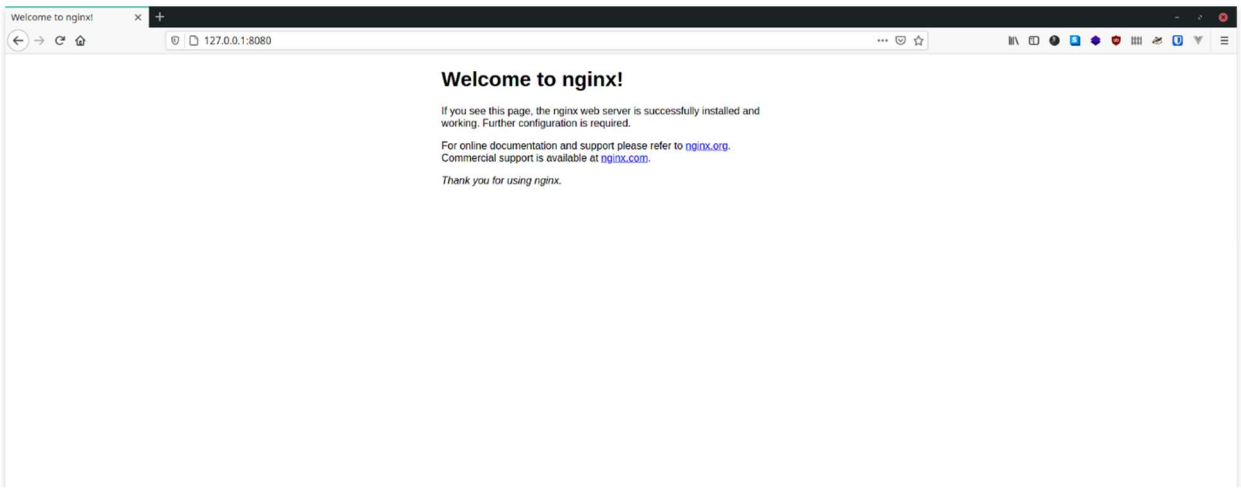

Рисунок 13 – Начальная страница сервиса Nginx

3.7.3 Пользовательские теги для изображений

Как и в случае с контейнерами, можно назначать изображениям пользовательские идентификаторы вместо того, чтобы полагаться на случайно сгенерированный идентификатор. В случае изображения это называется меткой вместо имени. В таких случаях используется параметр –tag или -t Общий синтаксис выглядит следующим образом:

--tag <репозиторий изображений>:<тег изображения>

Репозиторий обычно известен как имя образа, а тег указывает на определенную сборку или версию. Возьмем, к примеру, официальный образ mysql. Если хотите запустить контейнер, используя определенную версию MySQL, например, 5.7, можно выполнить, docker container run mysql:5.7, где mysql – имя, а 5.7 - тег.

Чтобы пометить свой собственный образ NGINX custom-nginx:packaged, можно выполнить следующую команду:

docker image build --tag custom-nginx:packaged .

Ничего не изменится, кроме того факта, что теперь обращаться к этому изображению можно как custom-nginx:packaged.

В тех случаях, когда забыли пометить изображение во время сборки или, может быть, хотите изменить тег, можете использовать image tag команду, для этого используется:

docker image tag <image id> <image repository>:<image tag>

или

docker image tag <image repository>:<image tag> <new image repository>:<new image tag>

3.7.4 Список и удаление изображений

Как и в случае с container lsкомандой, можно использовать image ls для вывода списка всех образов в локальной системе.

Перечисленные здесь изображения можно удалить с помощью команды image rm. Общий синтаксис выглядит следующим образом:

docker image rm <идентификатор образа>

Чтобы удалить изображение custom-nginx:packaged, необходимо выполнить следующую команду:

docker image rm custom-nginx:packaged

Аналогично используется команда image prune для очистки всех непомеченных висячих изображений следующим образом:

docker image prune –force

Параметр –force или -f пропускает все вопросы подтверждения. Также можно использовать параметр –all или -a для удаления всех кэшированных изображений в локальном реестре.

3.8 Контрольные вопросы

Зачем использовать контейнеры?
Что выдаст команда $ docker ps ?
Сколько идентификаторов по умолчанию имеет каждый контейнер?
Как задается имя контейнера?
Для чего нужна команда container start?
Как создать контейнер без запуска?
Что такое запуск контейнеров в интерактивном режиме?

Что представляет собой образ?
Как создать образ NGINX?

4. Непрерывная интеграция

Рассмотрим метод, который будет использовать Docker для активизации и улучшения усилий по непрерывной интеграции. К настоящему времени должны понимать, насколько хорошо Docker подходит для автоматизации. Его легкий характер и мощь, которую он дает для переноса сред из одного места в другое, могут сделать его ключевым фактором непрерывной интеграции. К концу главы вы поймете, как Docker может сделать этот процесс более быстрым, стабильным и воспроизводимым. Используя инструменты тестирования и расширяя возможности сборки с помощью плагина Jenkins Swarm увидите, как Docker может помочь вам получить еще больше от процесса непрерывной интеграции.

Непрерывная интеграция – это стратегия жизненного цикла программного обеспечения, используемая для ускорения процесса разработки. Автоматически перезапуская тесты каждый раз, когда в кодовую базу вносятся существенные изменения, вы получаете более быструю и стабильную поставку, поскольку в поставляемом программном обеспечении есть базовый уровень стабильности.

4.1 Установка Jenkins

Проект Jenkins предоставляет образ контейнера Linux, а не образ контейнера Windows. Убедитесь, что ваша установка Docker для Windows настроена на запуск Linux Containers, а не на Windows Containers. При установке Docker по стандарту настроен на запуск Linux Containers.

Откройте окно командной строки и выполните следующие действия:

Установка Jenkins.

1. Создайте мостовую сеть в Docker:
   docker network create jenkins
2. Запустите docker:dind Docker image:
   docker run --name jenkins-docker --rm --detach ^
   --privileged --network jenkins --network-alias docker ^
   --env DOCKER_TLS_CERTDIR=/certs ^
   --volume jenkins-docker-certs:/certs/client ^
   --volume jenkins-data:/var/jenkins_home ^
   --publish 2376:2376 ^
   docker:dind
3. Настройте официальный образ Jenkins Docker, выполнив следующие два шага:

1. Создайте Dockerfile со следующим содержимым:

```
FROM jenkins/jenkins:2.346.3-jdk11
USER root
RUN apt-get update && apt-get install -y lsb-release
RUN curl -fsSLo /usr/share/keyrings/docker-archive-keyring.asc \
  https://download.docker.com/linux/debian/gpg
RUN echo "deb [arch=$(dpkg --print-architecture) \
  signed-by=/usr/share/keyrings/docker-archive-keyring.asc] \
  https://download.docker.com/linux/debian \
  $(lsb_release -cs) stable" > /etc/apt/sources.list.d/docker.list
RUN apt-get update && apt-get install -y docker-ce-cli
USER jenkins
RUN jenkins-plugin-cli --plugins "blueocean:1.25.6 docker-workflow:1.29"
```

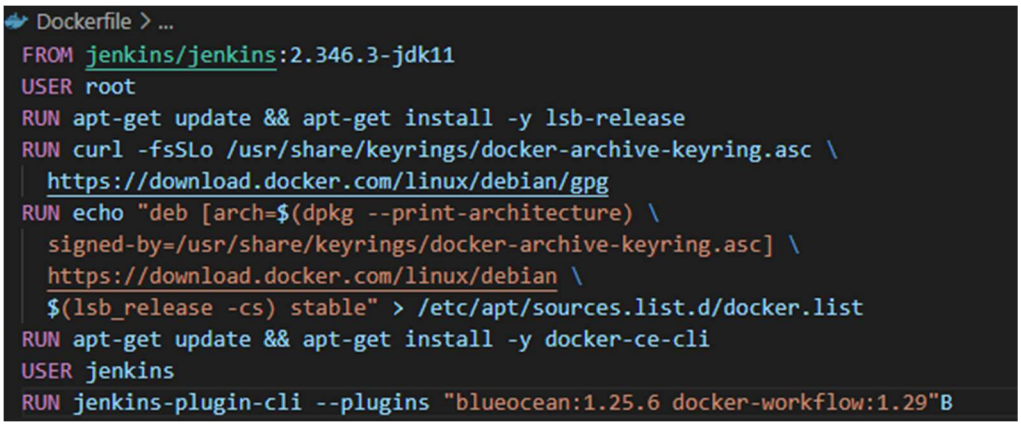

Рисунок 14 – Команды содержащиеся в Dockerfile

2. Создайте новый образ Docker из этого Dockerfile и назначьте образу значимое имя, например, «myjenkins-blueocean: 2.346.3-1»:

```
docker build -t myjenkins-blueocean:2.346.3-1 .
```

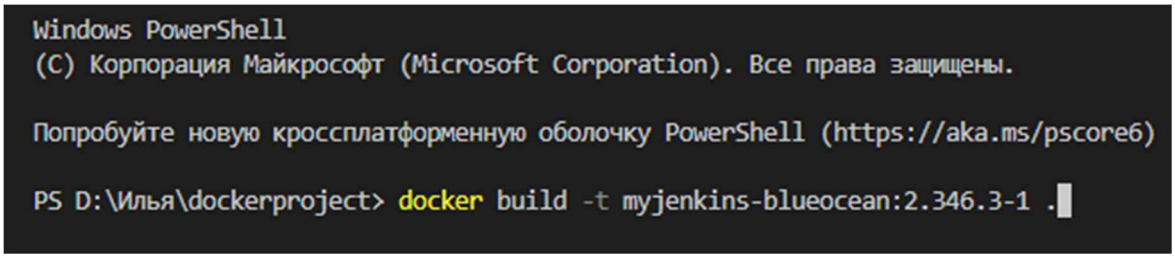

Рисунок 15 – Выполнение команды в терминале

4. Запустите свой собственный myjenkins-blueocean:2.346.3-1 образ как контейнер в Docker, используя следующую docker run команду:

```
docker run --name jenkins-blueocean --restart=on-failure --detach ^
  --network jenkins --env DOCKER_HOST=tcp://docker:2376 ^
```

--env DOCKER_CERT_PATH=/certs/client --env DOCKER_TLS_VERIFY=1 ^
--volume jenkins-data:/var/jenkins_home ^
--volume jenkins-docker-certs:/certs/client:ro ^
--publish 8080:8080 --publish 50000:50000 myjenkins-blueocean:2.346.3-1

Можно проверить состояние контейнеров в десктопной версии Docker или прописав в консоле:

docker ps

```
C:\Users\Kapra>docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS
NAMES
0cdd4ee7a49d   docker:dind    "dockerd-entrypoint.…"   5 minutes ago   Up 5 minutes   2375/tcp, 0.0.0.0:2376->2376/tcp
jenkins-docker
```

Рисунок 16 – Состояние контейнеров Docker

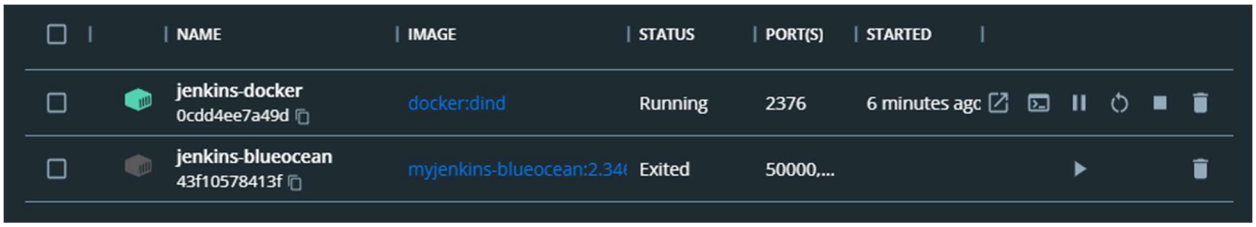

Рисунок 17 – Состояние контейнеров Docker

А так же проверить работу Jenkins перейдя на localhost:8080:

Рисунок 18 – Начальная страница Jenkins

4.2 Настройка Jenkins

1. Перейдите по http://localhost:8080 (или любой другой порт, который настроили для Jenkins при его установке) и подождите, пока не появится страница Unlock Jenkins .

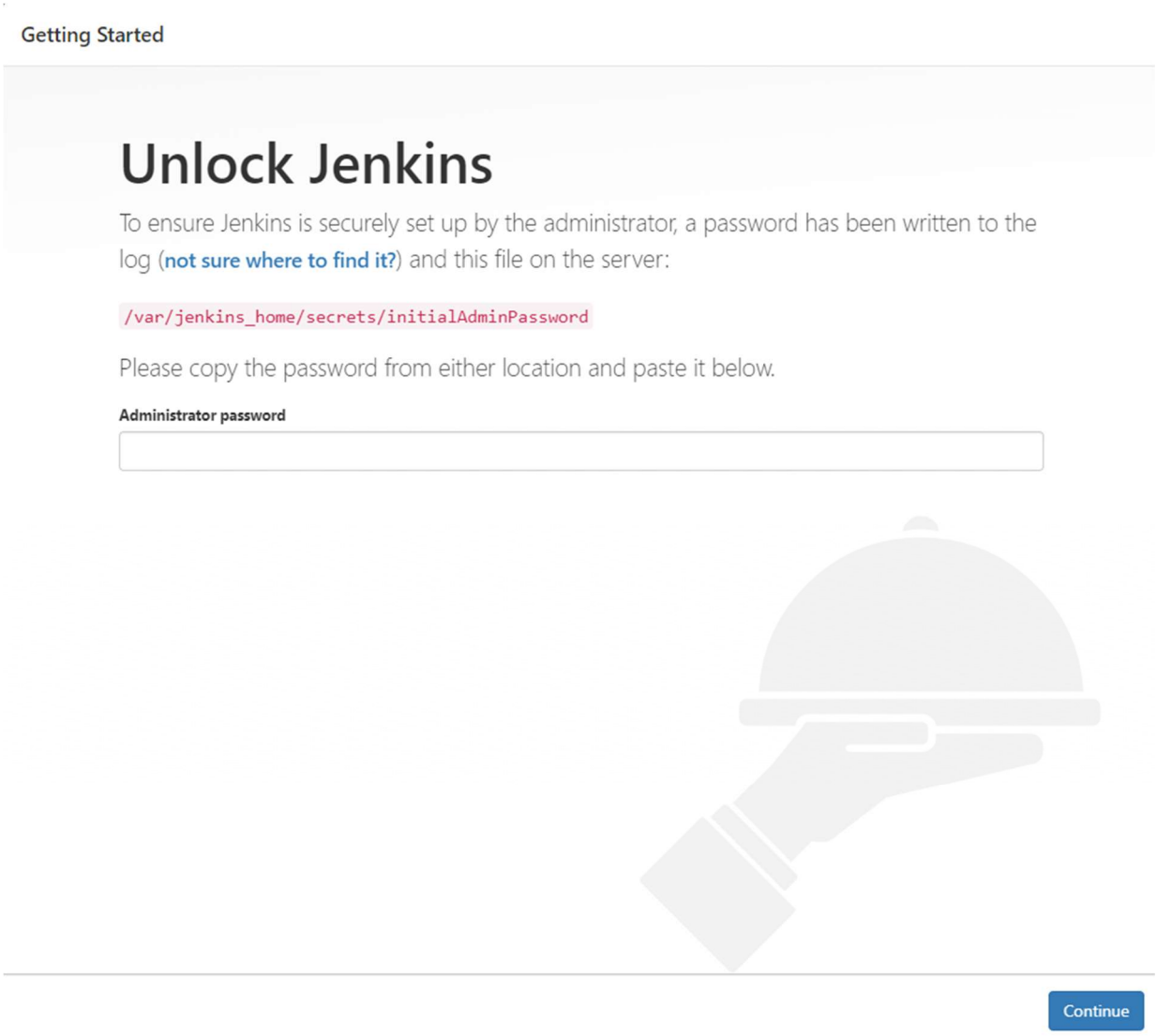

Рисунок 19 – Начальная страница Jenkins

2. Из вывода журнала консоли Jenkins скопируйте автоматически сгенерированный буквенно-цифровой пароль (между двумя наборами звездочек).

Для этого введите в консоль:

docker logs [ID container]

```
C:\Users\Kapra>docker ps
CONTAINER ID   IMAGE
951752057bef   myjenkins-blueocean:2.346.3-1
799c821c62e2   docker:dind

C:\Users\Kapra>docker logs 951752057bef
```

Рисунок 20 – Журнал консоли

Либо посмотрите логи в десктопной версии Docker

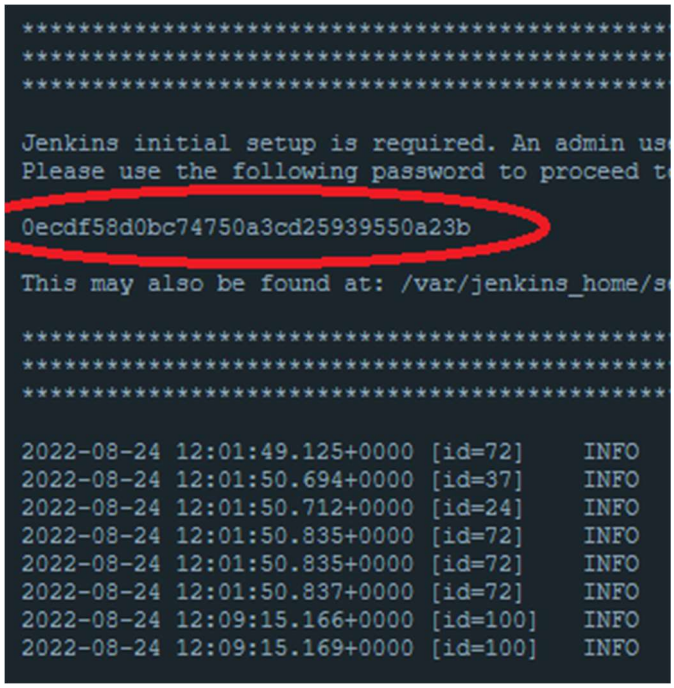

Рисунок 21 – Журнал консоли

3. На странице «Разблокировать Jenkins» вставьте этот пароль в поле «Пароль администратора» и нажмите «Продолжить».
4. После разблокировки Jenkins появится страница Customize Jenkins. Здесь можно установить любое количество полезных плагинов в рамках первоначальной настройки.
   Щелкните один из двух показанных вариантов:

   Install suggested plugins — чтобы установить рекомендуемый набор плагинов, основанный на наиболее распространенных вариантах использования.

   Select plugins to install — выбор набора плагинов для первоначальной установки. При первом доступе к странице выбора плагинов предлагаемые плагины выбираются по умолчанию.

   Если вы не уверены, какие плагины нужны, выберите «Установить предлагаемые плагины». Можно установить (или удалить) дополнительные подключаемые модули Jenkins позднее через страницу «Управление Jenkins» > « Управление подключаемыми модулями» в Jenkins.

   Мастер установки показывает процесс настройки Jenkins и установки выбранного набора подключаемых модулей Jenkins. Этот процесс может занять несколько минут.

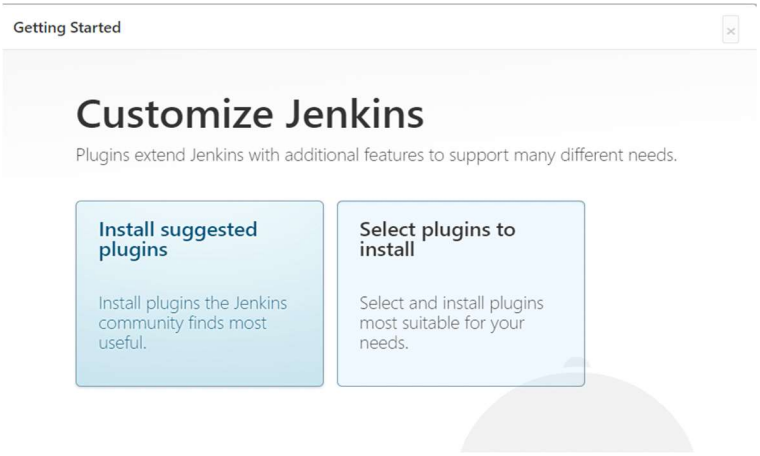

Рисунок 22 – Выбор установки плагинов

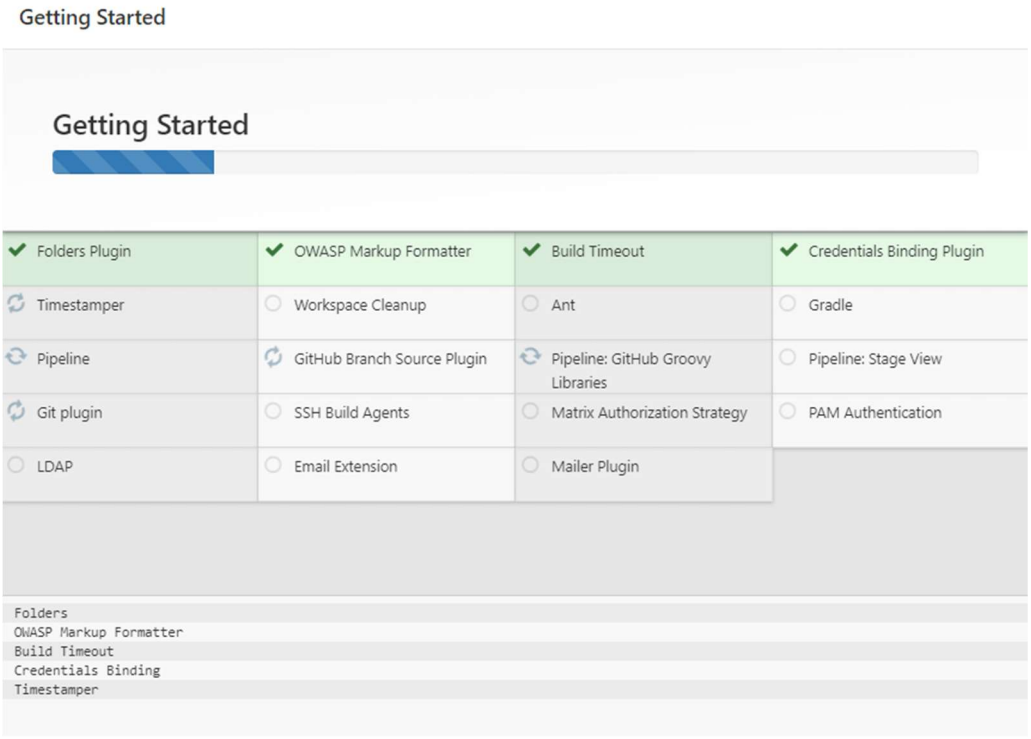

Рисунок 23 – Установка плагинов

5. Создание первого пользователя-администратора

Наконец, после настройки Jenkins с помощью плагинов Jenkins попросит вас создать первого пользователя-администратора. Когда появится страница «Создать первого пользователя-администратора», укажите сведения о вашем пользователе-администраторе в соответствующих полях и нажмите «Сохранить и завершить». Когда появится страница Jenkins is ready, нажмите Start using Jenkins.

При необходимости войдите в Jenkins с учетными данными только что созданного пользователя, можно начать использовать Jenkins.

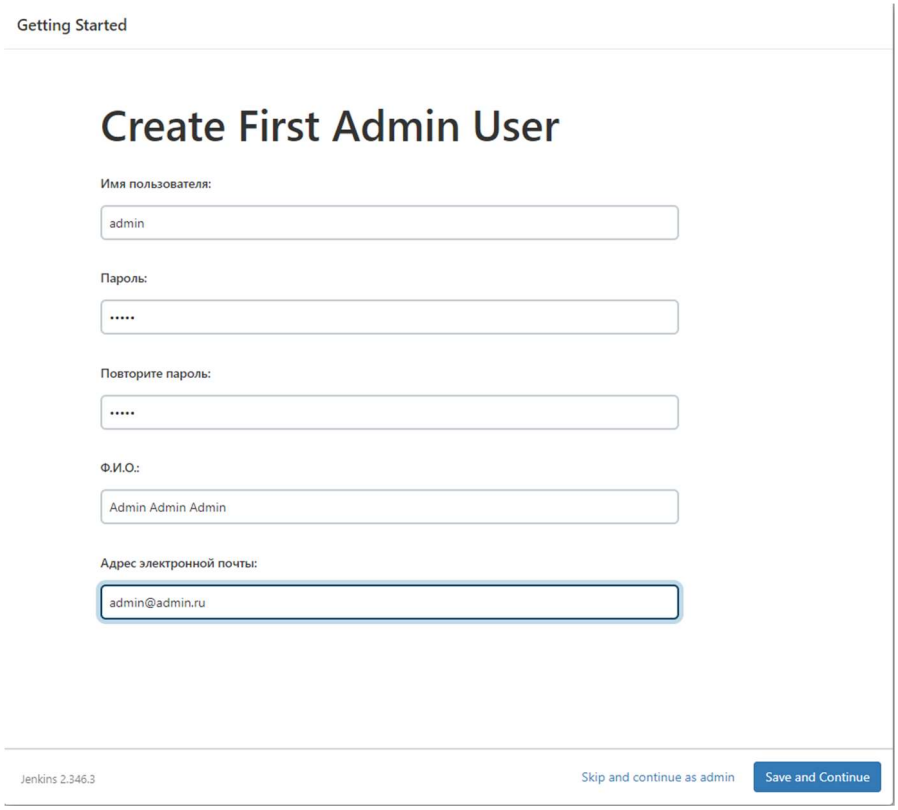

Рисунок 24 – Создание первого администратора

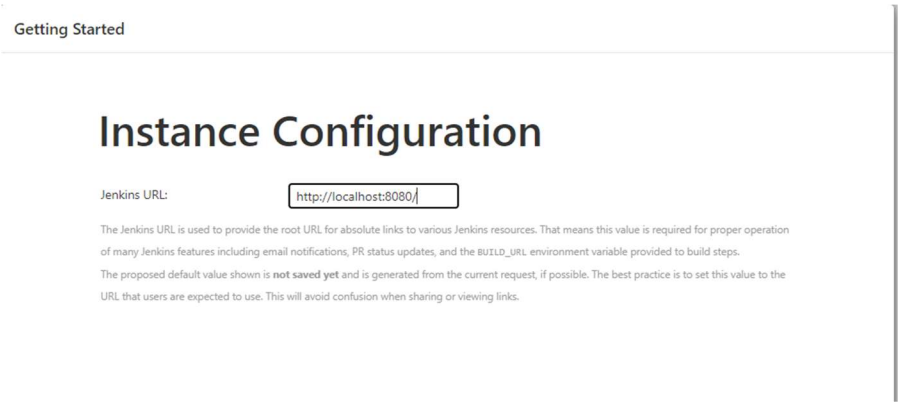

Рисунок 25 – Создание первого администратора

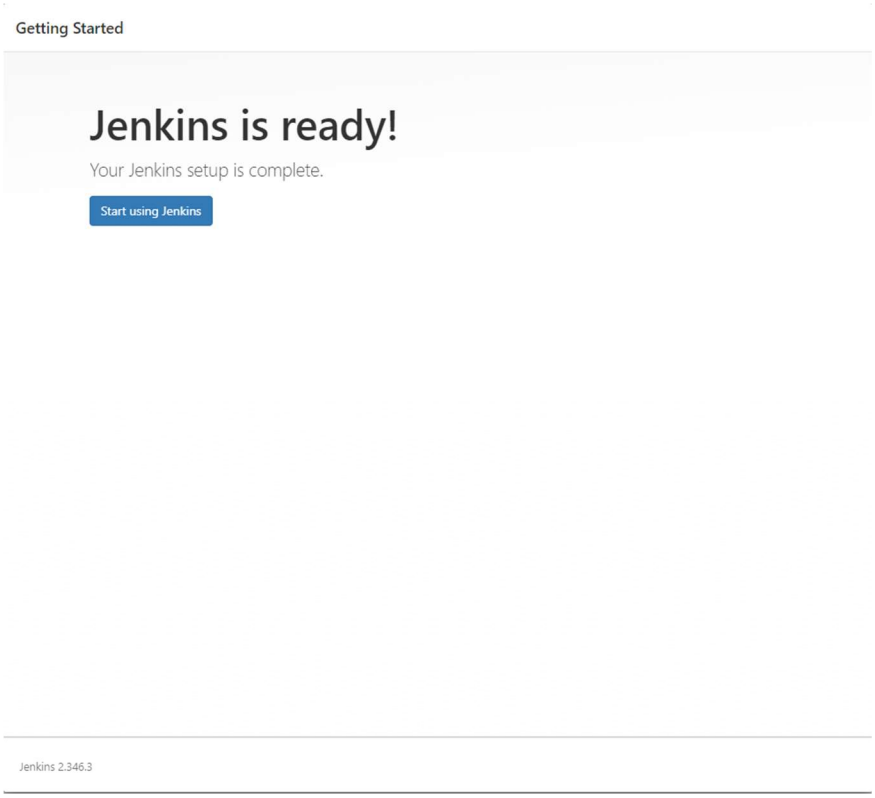

Рисунок 26 – Создание первого администратора

6. Далее нужно докачать оставшиеся плагины, а именно docker plugin. Для это перейдите «Настроить Jenkins» > «Управление плагинами» > «Доступные», в поиске введите Docker, выберите соответствующие плагины и нажмите «Download now and install after restart»

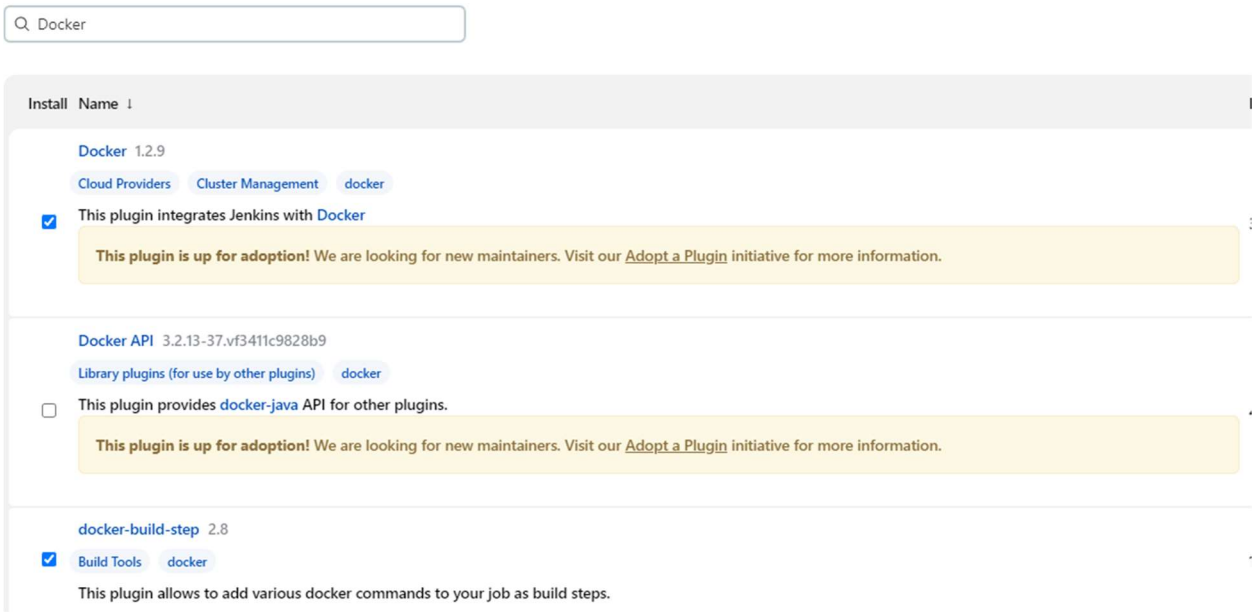

Рисунок 27 – Поиск необходимых плагинов

Рисунок 28 – Установка необходимых плагинов

7. Вернемся на домашнюю страницу, там можно увидеть предложение настроить облако «Configure a cloud», перейдем и выберем Docker

Мы успешно настроили и готовы использовать Jenkins для работы и непрерывной интеграции.

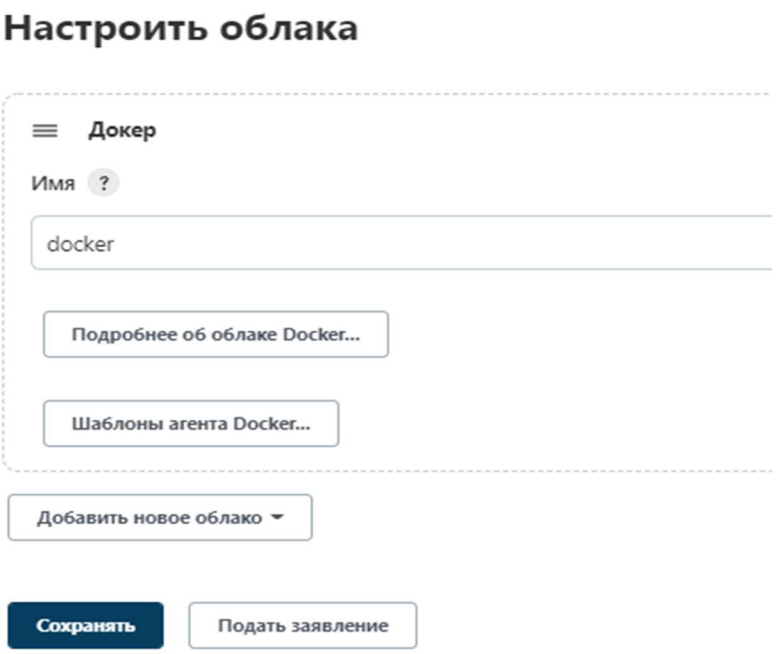

Рисунок 29 – Настройка облака Docker

Давайте создадим простую задачу и соберем её. Для этого нажмите «Создать Item», выберите «Pipeline» и введите название, например «docker-ci-cd». Здесь можно настроить свою задачу, в зависимости от ваших требований и условий. Пролистайте вниз до вкладки «Pipeline» и выберите предложенный образец «Hello World», нажмите Сохранить.

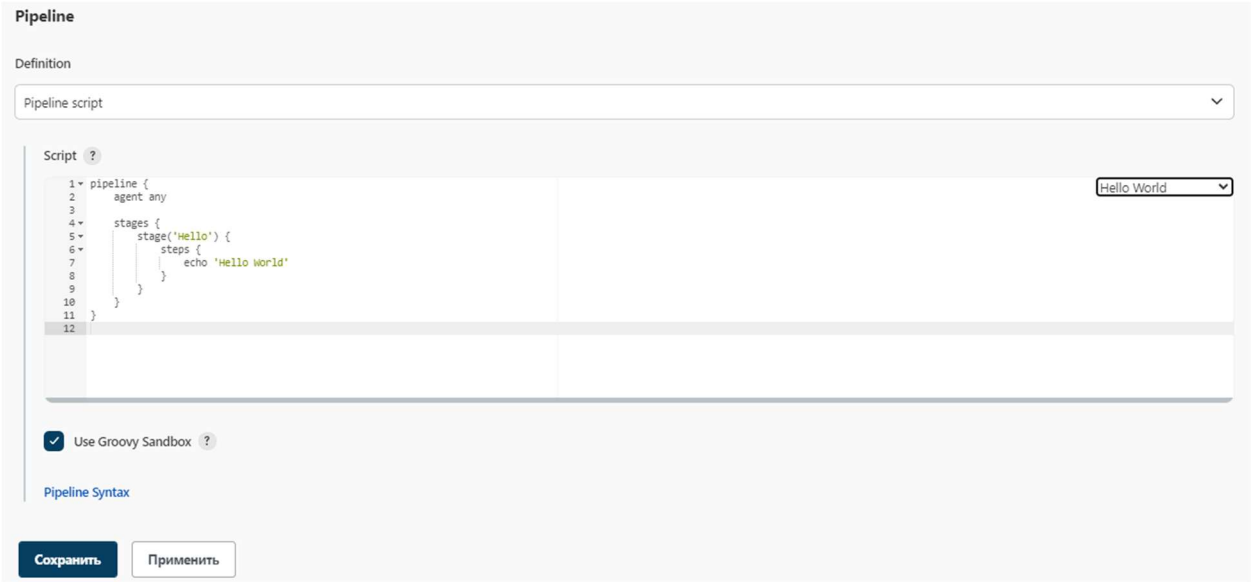

Рисунок 30 – Образец задачи

Нажмите «Собрать сейчас» и соберите вашу сборку. Перейдите к сборке, кликнув на «#1» и зайдите в «Console Output». Как видите, сборка прошла успешно и в консоль выведено «Hello world».

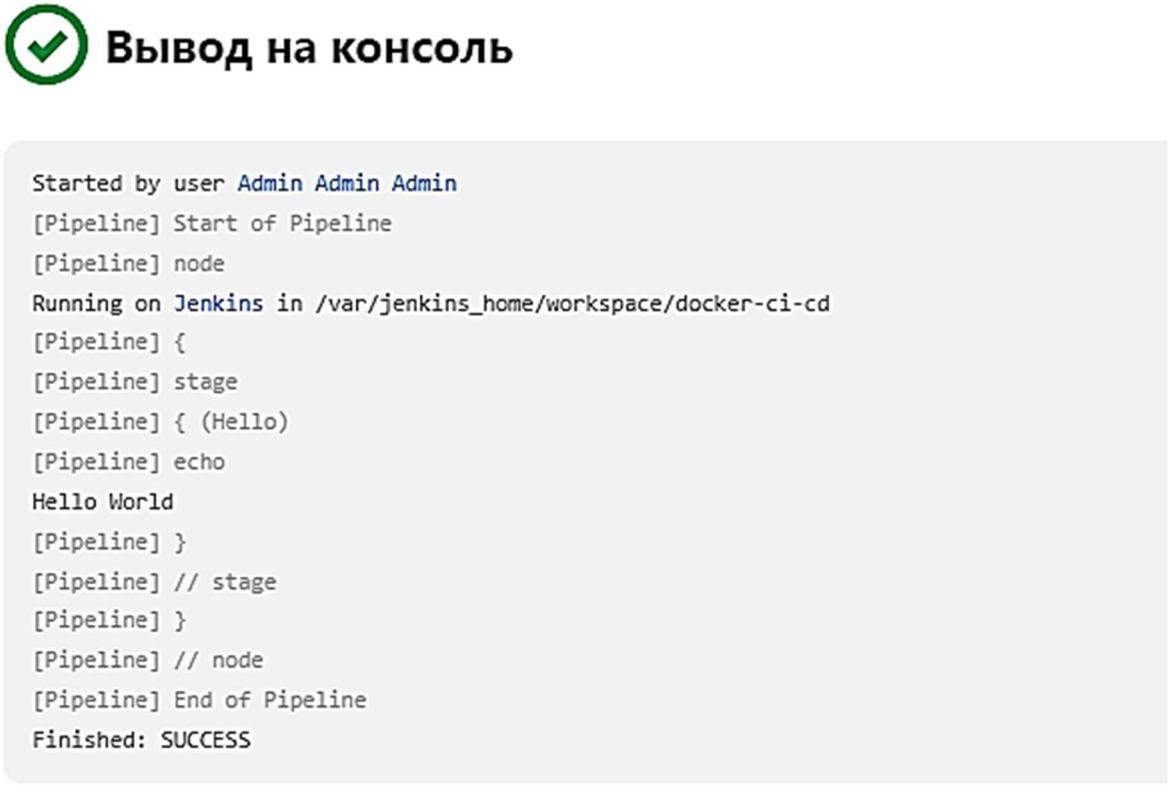

Рисунок 31 – Результат выполнения задачи

Для более наглядного примера непрерывной интеграции возьмем любой репозиторий на github.

Для нашего примера необходимы плагины для проекта Java. В разделе «Управление плагинами» скачиваем плагины «Pipeline Maven Integration», «AdoptOpenJDK» и «Maven Integration». Для настройки плагинов перейдите в «Настроить Jenkins» > «Конфигурация глобальных инструментов», долистайте до «Maven» и кликните «Добавить Maven». Укажите имя Maven и версию. Нажмите «сохранить».

Итак, нужно создать конвейер между Build-Test-Package

Сначала создадим независимые файлы для сборки, тестирования и упаковки, а затем объединим все три вместе, создав таким образом конвейер(pipeline).

1. Build:
   Кликните на «Создать Item» и выберите «Создать проект maven», дав название «build1».
   В Управление исходным кодом укажите:

- Git
- Ссылку на свой репозиторий
- Используемый логин и пароль (в нашем случае admin/admin)

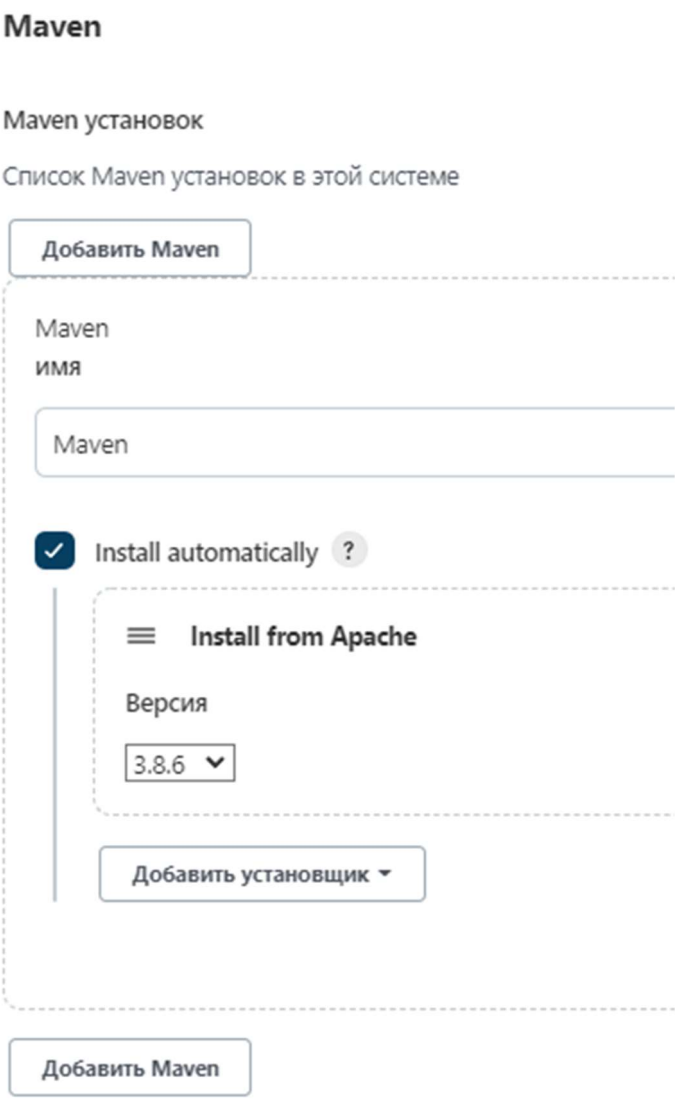

Рисунок 32 – Настройка Maven

В «Сборка» укажите:

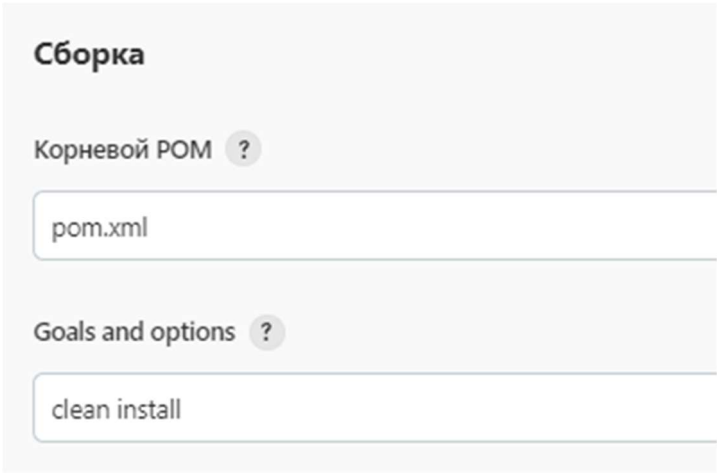

Рисунок 33 – Настройка сборки Build

Нажмите сохранить и после «Собрать сейчас». Если сборка завершилась успехом, то можно двигаться далее.

2. Test:
   Абсолютна та же процедура, что и для Build, за исключением названия и раздела «Сборка».
   Кликните на «Создать Item» и выберите «Создать проект maven», дав название «test1».
   В Управление исходным кодом укажите:

- Git
- Ссылку на свой репозиторий
- Используемый логин и пароль(в нашем случае admin/admin)

В «Сборка» укажите:

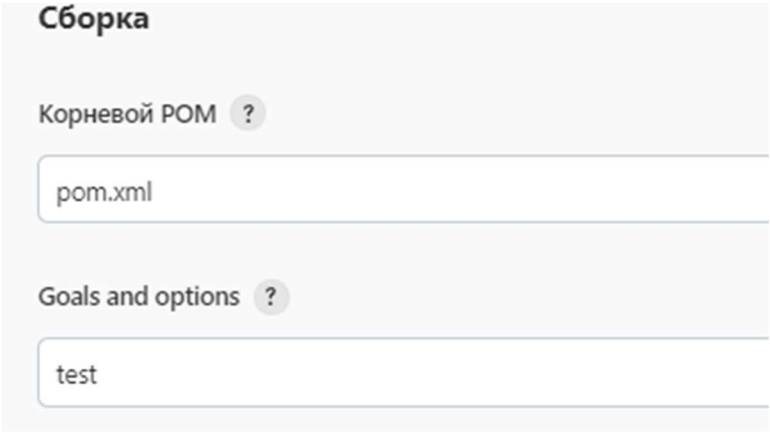

Рисунок 34 – Настройка сборки Test

Нажмите «сохранить» и после «Собрать сейчас». Если сборка завершилась успехом, то можно двигаться далее.

3. Package:
   Кликните на «Создать Item» и выберите «Создать проект maven», дав название «package1».
   В Управление исходным кодом укажите:

- Git
- Ссылку на свой репозиторий
- Используемый логин и пароль (в нашем случае admin/admin)
- В «Послесборочные операции» выберите «Заархивировать артефакты» и укажите местоположение, в котором будет храниться файл пакета, например: «**/target/*.jar»

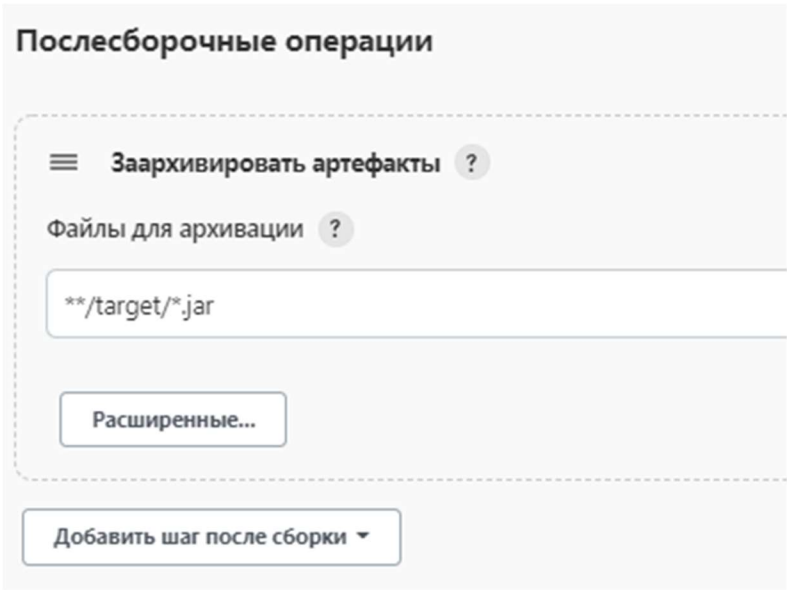

Рисунок 35 – Настройка послесборочных операций

В «Сборка» укажите:

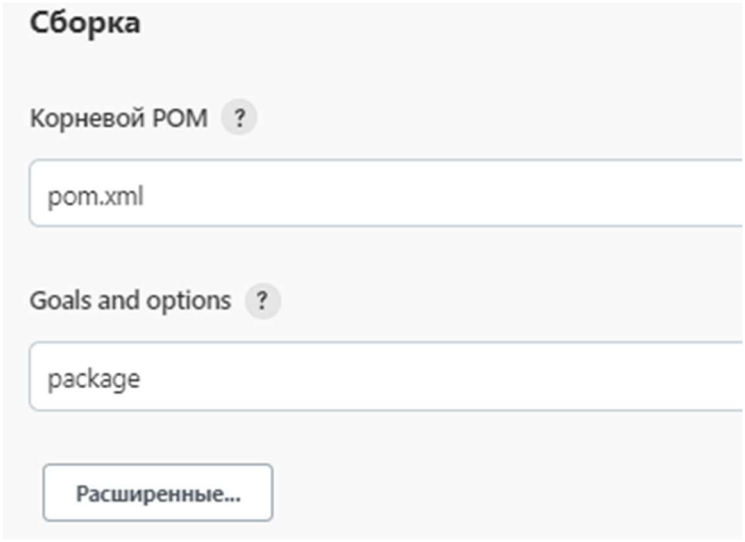

Рисунок 36 – Настройка сборки Package

Нажмите «сохранить» и после «Собрать сейчас». Если сборка завершилась успехом, то можно двигаться далее.

4. Интеграция “Build-Test-Package”:

   На данный момент все эти 3 файла независимы, нам нужно их интегрировать.

   1. Интеграция файла “build” в файл “test”

В настройках сборки «build1» > Послесборочные операции выберите Собрать другой проект и укажите название «test1»:

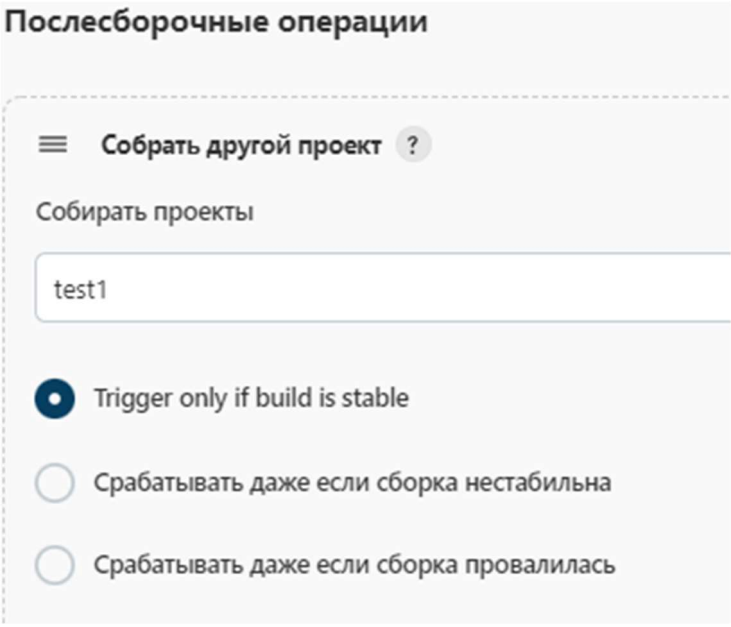

Рисунок 37 – Настройка послесборочных операций

2. Интеграция файла “test” в файл “package”

В настройках сборки «test1» > «Послесборочные операции» выберите «Собрать другой проект» и укажите название «package1»:

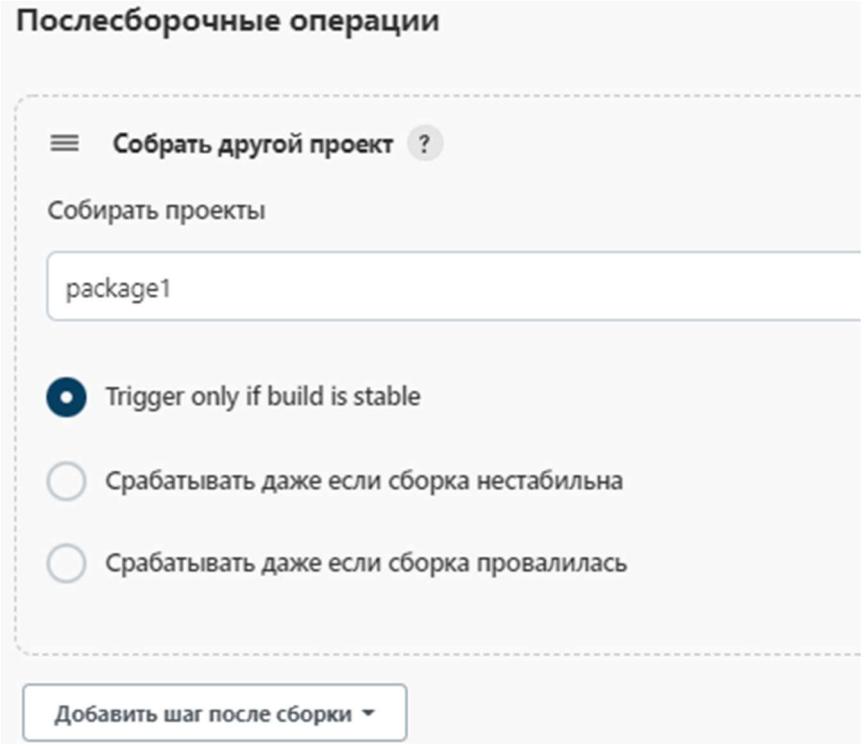

Рисунок 38 – Настройка послесборочных операций

Теперь всякий раз, когда в проекте происходит изменение / редактирование, это запускает фазу сборки. Фаза сборки запустит фазу тестирования, а фаза тестирования запустит фазу пакета. Следовательно, создается непрерывный интеграционный конвейер.

В данной главе был рассмотрен пример развертывания сервиса CI/CD в Docker. Был настроен сервис Jenkins для непрерывной интеграции и последовательного запуска каждого этапа тестового приложения.

4.3 Контрольные вопросы

Что такое непрерывная интеграция?
Что такое проект Jenkins?
Как создать пользователя-администратора в Jenkins?
Какую фазу запускает фаза тестирования?

5. Составление проектов с помощью Docker-Compose

Наиболее простым способом управления многоконтейнерным проектом является Docker Compose.

Согласно документации Docker, «Compose — это инструмент для определения и запуска многоконтейнерных приложений Docker. С помощью Compose используете файл YAML для настройки служб вашего приложения. Затем с помощью одной команды создаете и запускаете все службы из вашей конфигурации."

Хотя Compose работает во всех средах, он больше ориентирован на разработку и тестирование. Использование Compose в производственной среде не рекомендуется.

Еще одна замечательная особенность Compose — поддержка запуска модульных и E2E-тестов быстрым и повторяемым образом, помещая их в свои собственные среды. Это означает, что вместо тестирования приложения на вашей локальной/хост-ОС можете запустить среду, которая очень похожа на производственные условия.

Compose использует имена проектов для изоляции сред друг от друга, что дает следующие преимущества:

- Возможность запускать несколько копий одной и той же среды на одном компьютере.
- Это предотвращает взаимодействие разных проектов и сервисов друг с другом.

5.1 Структура файла Compose

Compose позволяет легко обрабатывать несколько контейнеров Docker одновременно. Есть возможность использовать сразу несколько правил, для этого их необходимо указать в файле docker-compose.yml.

Такой файл состоит из записей, разделенных табуляцией. В большинстве случаев файл включает в себя 4 основных пункта:

- Версия компоновочного файла
- Услуги, которые будут построены
- Все использованные тома
- Сети, которые соединяют различные услуги

Пример файла может выглядеть так:

```
version: '3.3'

services:

  db:

    image: mysql:5.7

    volumes:

      - db_data:/var/lib/mysql

    restart: always

    environment:

      MYSQL_ROOT_PASSWORD: somewordpress

      MYSQL_DATABASE: wordpress

      MYSQL_USER: wordpress

      MYSQL_PASSWORD: wordpress

  wordpress:

    depends_on:

      - db

    image: wordpress:latest

    ports:

      - "8000:80"
```

```
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
volumes:
  db_data: {}
```

Данный файл описывает приложение Wordpress с базой MySQL. Особенностью такого запуска является то, что каждая служба запускается отдельно, что позволяет заменить любой компонент при необходимости.

5.2 Services

Services – тег, который объединяет все контейнеры, используемые в проекте Compose.

```
services:
  proxy:
    build: ./proxy
  app:
    build: ./app
  db:
    image: postgres
```

5.3 Базовое изображение:

Работа с изображениями в Docker Compose аналогична работе в Docker. Для развертывания проекта можно использовать уже готовые образы из DockerHub, либо же создать собственный образ.

Вот несколько основных примеров:

version: '3.3'

services:

alpine:

image: alpine:latest

stdin_open: true

tty: true

command: sh

Здесь используется предопределенный образ из DockerHub.

Другой вариант определения базового образа — использовать файл Dockerfile с произвольным именем.

build:

context: ./dir

dockerfile: Dockerfile.dev

5.4 Порты

Предоставление портов в Compose работает так же, как и в Dockerfile. Различают два разных метода раскрытия порта:

5.4.1 Предоставление порта связанным службам:

expose:

- "3000"

- "8000"

Здесь используются порты для связанных служб контейнера, а не для хост-системы.

5.4.2 Предоставление порта хост-системе:

```
ports:
  - "8000:80"  # host:container
```

В этом примере определяется, какой порт открыть, и порт хоста, которому он должен быть предоставлен.

Также можно определить протокол порта, который может быть UDP или TCP:

```
ports:
 - "8000:80/udp"
```

5.5 Команды

Для автоматизации процесса работы контейнера в Docker Compose созданы команды. По сути, данная функция позволяет выполнять команды в терминале из файла проекта compose. Чаще всего используется именно команда запуска. Например, для запуска веб-сайта с помощью команды CLI указывается команда npm run start в поле command файла проекта.

```
app:
    container_name: website
    restart: always
    build: ./
    ports:
      - '3000:3000'
    command:
      - 'npm run start'
```

Эта команда будет выполнена после запуска контейнера, а затем запустится веб-сайт.

5.6 Тома

Использование томов – это основной способ хранения данных Docker. Преимуществом томов является тот факт, что они полностью управляются Docker. С помощью томов можно легко обмениваться информацией между контейнером и хост-системой.

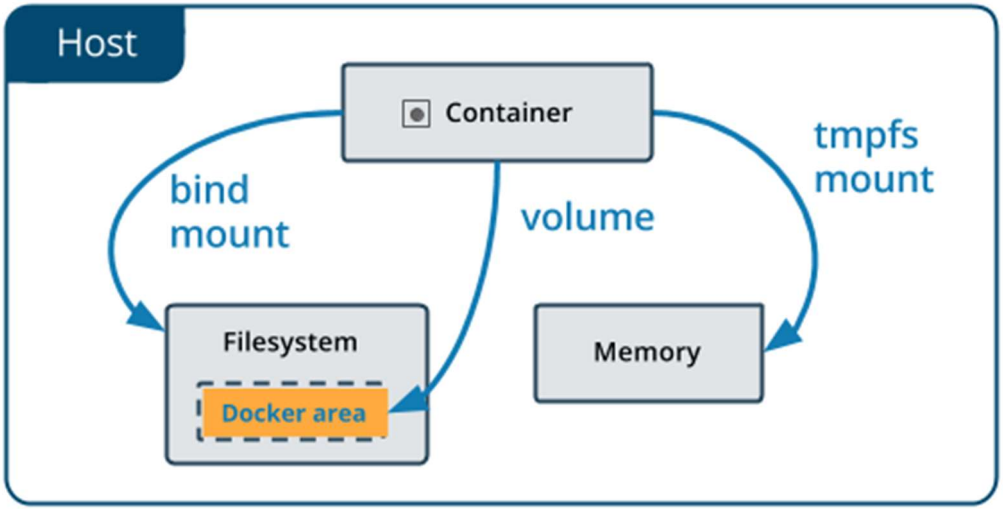

Рисунок 40 – Схема взаимодействия контейнера с внешней средой.

В Docker можно использовать несколько типов томов. Все они могут быть определены с помощью ключевого слова volumes, но имеют некоторые незначительные отличия.

5.6.1 Нормальный том

Обычный способ использования томов — просто определить конкретный путь и позволить движку создать для него том. Это можно сделать следующим образом:

```
volumes:

  # Just specify a path and let the Engine create a volume

  - /var/lib/mysql
```

5.6.2 Отображение пути

Можно определить абсолютное сопоставление путей для своих томов, указав путь в хост-системе и сопоставив его с местом назначения контейнера с помощью оператора:

```
volumes:
```

- /opt/data:/var/lib/mysql

Определяется путь хост-системы, за которым следует путь контейнера.

5.6.3 Именованный том

Другой тип тома — это именованный том, который похож на другие тома, но имеет свое собственное имя, которое упрощает его использование в нескольких контейнерах. Вот почему его часто используют для обмена данными между несколькими контейнерами и службами.

volumes:

- datavolume:/var/lib/mysql

5.7 Переменные среды

Переменные среды используются для переноса данных конфигурации в приложения. Это часто используется, если есть некоторые конфигурации, которые зависят от операционной системы хоста, или какие-то другие переменные.

5.7.1 Установка переменной окружения

Можно установить переменные среды в контейнере, используя ключевое слово «environment».

web:

environment:

- NODE_ENV=production

Устанавливается переменная среды, предоставляя ключ и значение для этого ключа.

5.7.2 Передача переменной окружения

Можно передавать переменные среды из оболочки прямо в контейнер, просто определяя ключ среды в файле Compose и не присваивая ему значения.

web:

environment:

- NODE_ENV

Здесь значение *NODE_ENV* берется из значения той же переменной в оболочке, которая запускает файл Compose.

5.7.3 Использование файла .env

Иногда нескольких переменных среды недостаточно, и управление ими в файле Compose может стать довольно запутанным. Вот для чего нужны файлы .env. Они содержат все переменные среды для вашего контейнера и могут быть добавлены с помощью одной строки в файле Compose.

web:

env_file:

- variables.env

5.8 Сеть

Сети определяют правила связи между контейнерами, а также между контейнерами и хост-системой. Их можно настроить для обеспечения полной изоляции контейнеров, что позволяет создавать приложения, которые безопасно работают вместе.

По умолчанию Compose устанавливает одну сеть для каждого контейнера. Каждый контейнер автоматически присоединяется к сети по умолчанию, что делает их доступными как для других контейнеров в сети, так и для обнаружения по имени хоста, указанному в файле Compose.

5.8.1 Использование пользовательских сетей

Вместо использования сети по умолчанию также можно указать собственные сети в ключе сетей верхнего уровня, что позволяет создавать более сложные топологии и указывать сетевые драйверы и параметры.

networks:

frontend:

backend:

driver: custom-driver

```
    driver_opts:
      foo: "1"
```

Каждый контейнер может указать, к каким сетям подключаться, с помощью ключевого слова «networks».

```
services:
  proxy:
    build: ./proxy
    networks:
      - frontend
  app:
    build: ./app
    networks:
      - frontend
      - backend
  db:
    image: postgres
    networks:
      - backend
```

Также можно указать собственное имя для сети (начиная с версии 3.5):

```
version: "3.5"
networks:
  webapp:
    name: website
```

driver: website-driver

5.8.2 Внешние (ранее существовавшие) сети

Можно использовать существующие сети с Docker Compose, применяя внешний вариант.

networks:

default:

external:

name: pre-existing-network

В этом примере Docker никогда не создает сеть по умолчанию, а просто использует существующую сеть, определенную во внешнем теге.

5.9 CLI

Важным этапом работы с Docker Compose является процесс управления проектом. Для этого используется определенный набор команд, который очень похож на набор команд для стандартного Docker.

build Создать или пересоздать проект

help Получить помощь по команде

kill Убить контейнер

logs Просмотр выходных данных из контейнеров

port Выведите общедоступный порт для привязки порта

ps Список контейнеров

pull Извлекает изображения сервиса

rm Удаляет остановленный контейнер

run Выполнение одной команды

scale Установите количество контейнеров для службы

start Запуск службы

stop Остановка службы

restart Перезапуск службы

up Создание и запуск контейнера

down Остановка и удаление контейнера

Они не только похожи, но и ведут себя как их аналоги из Docker. Единственное отличие состоит в том, что они влияют на всю многоконтейнерную архитектуру, определенную в файле docker-compose.yml, а не на один контейнер.

5.10 Compose в работе

Docker Compose значительно упрощает работу с большими проектами, так как позволяет развернуть связанные службы на одном сервере.

Есть вещи, которые, вероятно, нужно изменить перед развертыванием конфигурации приложения в рабочей среде. Эти изменения включают в себя:

- Привязку разных портов к хосту
- Указание политики перезапуска, чтобы избежать простоя вашего контейнера.
- Добавление дополнительных сервисов
- Удаление любых ненужных привязок томов для кода приложения

5.11 Контрольные вопросы

Для чего необходим Docker Compose?
Какая структура файла Compose?
Что такое Services?
Какие существуют методы раскрытия порта?
Что такое предоставление порта связанным службам?
Для чего используются тома?
Для чего применяют переменные среды?
Как определяют правила связи между контейнерами?

Заключение

Стоит отметить, что технология виртуализации с использованием программного обеспечения Docker уже широко используется или интенсивно

внедряется в производство многими компаниями. Применение Docker позволяет легко развертывать свои приложения в облаке и управлять многоконтейнерным проектом посредством Docker Compose. Данная технология позволяет сэкономить большое количество ресурсов и повысить производительность разворачиваемого ПО. Пособие поможет обучающимся начать работу с Docker, улучшить процесс разработки с применением современных технологий виртуализации.

Список источников

1 Документация Docker: сайт. – URL: https://docs.docker.com/get-started/overview/ (дата обращения: 09.08.2022)
2 Docker for begginers: сайт. – URL: https://docker-curriculum.com/ (дата обращения: 09.08.2022)
3 Docker tutorial: сайт. – URL: Docker Tutorial (дата обращения: 09.08.2022)
4 Милл И. Docker на практике / И. Милл, Э. Х. Сейерс. – Москва: ДМК Пресс, 2020. – 516 с.
5 Learning Docker: сайт. – URL: https://riptutorial.com/Download/docker.pdf (дата обращения: 09.08.2022)
6 Potdar A. M. et al. Performance evaluation of docker container and virtual machine //Procedia Computer Science. – 2020. – Т. 171. – С. 1419-1428.
7 Lingayat A., Badre R. R., Gupta A. K. Performance evaluation for deploying docker containers on baremetal and virtual machine //2018 3rd International Conference on Communication and Electronics Systems (ICCES). – IEEE, 2018. – С. 1019-1023.
8 Wan X. et al. Application deployment using Microservice and Docker containers: Framework and optimization //Journal of Network and Computer Applications. – 2018. – Т. 119. – С. 97-109.
9 De Benedictis M., Lioy A. Integrity verification of Docker containers for a lightweight cloud environment //Future generation computer systems. – 2019. – Т. 97. – С. 236-246.
10 Brady K. et al. Docker container security in cloud computing //2020 10th Annual Computing and Communication Workshop and Conference (CCWC). – IEEE, 2020. – С. 0975-0980.
11 Saha P. et al. Evaluation of docker containers for scientific workloads in the cloud //Proceedings of the Practice and Experience on Advanced Research Computing. – 2018. – С. 1-8.
12 Yadav A. K. et al. Docker containers versus virtual machine-based virtualization //Emerging Technologies in Data Mining and Information Security. – Springer, Singapore, 2019. – С. 141-150.

Зудилова Татьяна Викторовна
Ананченко Игорь Викторович
Иванов Сергей Евгеньевич

Контейнеризатор приложений docker — установка, настройка, основы управления приложениями в средах с поддержкой контейнеризации

Учебно-методическое пособие

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49, литер А