УДК 004.9

ИССЛЕДОВАНИЕ МЕТОДОЛОГИИ DEVOPS ДЛЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Нажимова Н.А., Вдовин А.А.

ФГБОУ ВО «Нижегородский государственный технический университет им. Р.Е. Алексеева», Нижний Новгород, e-mail: bahmetova@list.ru

В настоящее время крупные компании при разработке программного обеспечения стремятся переходить на микросервисную и бессерверную архитектуры. Использование микросервисов позволяет разработчикам создавать сложные системы в виде ограниченного количества небольших приложений. Процессы разработки, тестирования и развертывания при этом для каждого приложения могут проходить независимо. Это позволяет значительно увеличить стабильность работы системы в целом и снизить время, требуемое для ее создания. Бессерверная архитектура, в свою очередь, предполагает использование облачных сервисов, что позволяет команде разработчика использовать все свои ресурсы для написания кода приложений, не занимаясь при этом созданием и обслуживанием требуемой для информационной системы инфраструктуры. При этом для ускорения процесса разработки требуется максимальная оптимизация и автоматизация доставки разрабатываемого программного обеспечения на сторону заказчика. Современная методология DevOps позволяет успешно решать подобный спектр задач, обеспечивая полностью автоматизированный процесс сборки, настройки и развертывания разрабатываемых приложений. В работе рассмотрены основные инструменты, применяемые в методологии DevOps и практиках CI-CD: GitLab CI, Ansible, Docker, Terraform, Jenkins, Kubernetes. Данный набор не является обязательным и может быть изменен в зависимости от задач, которые необходимо решить.

Ключевые слова: DevOps, CI-CD, Docker, Kubernetes, Gitlab-CI, Jenkins, Ansible, Terraform

RESEARCH OF DEVOPS METHODOLOGY FOR SOFTWARE DEVELOPMENT

Nazhimova N.A., Vdovin A.A.

R.E. Alekseev Nizhny Novgorod State Technical University, Nizhny Novgorod, e-mail: bahmetova@list.ru

Nowadays, large companies tend to switch to microservice and serverless architectures when developing software. The use of microservices allows developers to create complex systems in the form of a limited number of small applications. At the same time, the development, testing and deployment processes for each application can take place independently. This allows you to significantly increase the stability of the system as a whole and reduce the time required to create it. The serverless architecture, in turn, involves the use of cloud services, which allows the developer team to use all its resources to write application code without having to create and maintain the infrastructure required for the information system. At the same time, to speed up the development process, maximum optimization and automation of the delivery of the developed software to the customer is required. Modern DevOps methodology allows you to successfully solve such a range of tasks, providing a fully automated process of building, configuring and deploying applications under development. The paper discusses the main tools used in the DevOps methodology and CI-CD practices: GitLab CI, Ansible, Docker, Terraform, Jenkins, Kubernetes. This set is optional and can be changed, depending on the tasks that need to be solved.

Keywords: DevOps, CI-CD, Docker, Kubernetes, Gitlab-CI, Jenkins, Ansible, Terraform

DevOps объединяет процессы разработки и эксплуатации программного обеспечения для повышения эффективности, скорости и безопасности создания информационных систем по сравнению с традиционными методами. Более быстрый жизненный цикл разработки программного обеспечения обеспечивает конкурентное преимущество компаниям разработчиков и их клиентам. Инструменты DevOps тесно входят в нашу жизнь, а специалисты, владеющие ими, наиболее востребованы на рынке труда.

Цель исследования – рассмотреть наиболее важные аспекты методологии DevOps для их максимально эффективного внедрения в процессы разработки программного обеспечения.

Материалы и методы исследования

Методы исследования включают в себя анализ и обобщение исходной информации из литературных источников авторитетных авторов в области современных методов проектирования информационных систем.

DevOps (development & operations) – это методология автоматизация процессов сборки, настройки и развертывания (установки) приложений [1]. Данная методология подразумевает, что DevOps инженер (специалист) будет работать совместно с разработчиком программного обеспечения (ПО) для автоматизации сборки и установки ПО. DevOps неразрывно связан с описанием CI-CD процесса. CI-CD – Continuous integration and continuous delivery (непре-

рывная интеграция и непрерывная доставка). Непрерывная интеграция – это интеграция отдельных частей кода программного обеспечения, которые в процессе сборки соединяются между собой. Это позволяет чаще производить итерации тестирования и сборки, тем самым ускоряется и делается более гибким процесс создания ПО. Непрерывная доставка – это подход к разработке программного обеспечения, при котором ПО автоматически передается и разворачивается на рабочих серверах (станциях).

Методология DevOps представляет собой обширную тему. Это значительно более глубокое понятие, чем настройка и установка программного обеспечения. DevOps затрагивает абсолютно разные инструменты и языки программирования для реализации принципов данной методологии. Рассмотрим CI-CD процесс в его наиболее предпочтительной форме (рис. 1).

Данный процесс идет непрерывно во времени, и каждая его стадия сменяет предыдущую. CI (Continuous integration) процесс состоит из следующих основных этапов: Plan, Code, Build, Test. **Plan** – этап планирования. На данном этапе идет обсуждение и постановка целей и задач на ближайшее время. **Code** – написание кода. На данном этапе разработчики пишут код в соответствии с поставленными задачами с использованием системы контроля версий. **Build** – сборка. На данном этапе части кода, написанные участниками команды разработчика, собираются в единый код приложения. **Test** – тестирование. На данном этапе происходит тестирование собранного кода, в том числе средствами автоматизированного тестирования.

CD (Continuous delivery) процесс состоит из следующих основных этапов: Release, Deploy, Operate, Monitor [2]. **Release** – этап выпуска программного продукта. На данном этапе принимается решение о выпуске соответствующей версии разрабатываемого ПО или отдельного микросервиса. **Deploy** – этап развертывания. На данном этапе происходит развертывание разрабатываемого ПО или отдельного микросервиса на серверах или рабочих станциях в автоматизированном режиме. **Operate** – процесс эксплуатации ПО. На данном этапе разрабатываемое ПО или приложения на базе микросервисной архитектуры эксплуатируются непосредственно клиентами. **Monitor** – процесс сбора данных об эксплуатируемом программном продукте. На данном этапе специалисты DevOps производят оценку текущих релизов с целью формирования заданий для будущего развития проекта. Наличие обратной связи позволяет построить процесс разработки информационных систем с непрерывным повышением качества.

Следует отметить, что DevOps инструменты в своем составе содержат [3]:

- GitLab CI;
- Ansible;
- Docker;
- Terraform;
- Jenkins;
- Kubernetes.

Данный набор не является обязательным и может быть изменен в зависимости от задач, которые необходимо решить.

Одним из основных инструментов для разработки по методологии DevOps является Docker (Докер). Docker – это популярная технология контейнеризации [4]. Появилась она в 2013 г. Идея контейнеризации преследует цель изолировать процессы, выполняющиеся внутри контейнера, от внешней операционной системы и сопутствующих зависимостей.

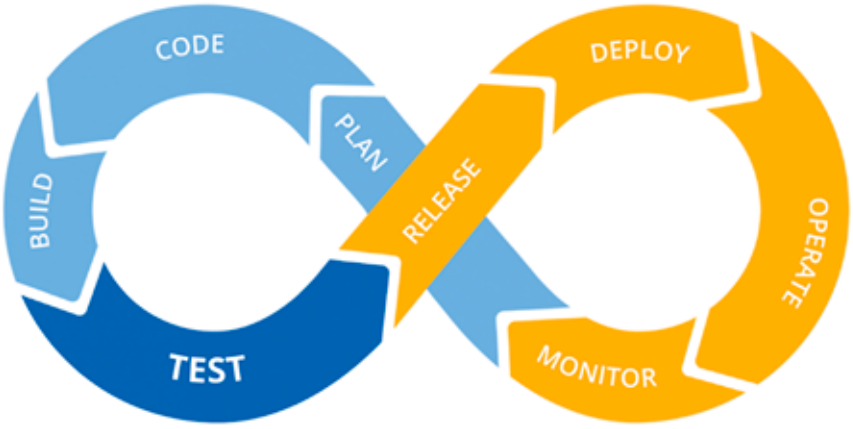

Рис. 1. Иллюстрация CI-CD процесса

Рис. 2. Пример gitlab-ci пайплайна

С помощью контейнеров очень удобно разделять и запускать ПО модульно. Кроме того, системы контейнеризации позволяют управлять группами контейнеров, что является одной из основных причин их популярности в современных средствах разработки. Мы не будем углубляться в отличие виртуализации от контейнеризации, это отдельная достаточно хорошо описанная тема [5].

Одним из основных понятий, которые используются при работе с системой контейнеризации Docker, является «docker image». Docker image (образ) – неизменяемый файл, из которого создаются контейнеры. Все приложения упаковываются в образы. Чтобы в контейнерах могло что-либо запуститься, образы основываются на какой-либо «легкой» операционной системе ОС, например дистрибутиве Linux Ubuntu. Можно построить на основе этого образа свой, дополненный образ, где будут установлены только те пакеты, которые нужны для данной части сервиса (приложения).

Как было отмечено выше, DevOps предполагает автоматизированный процесс сборки, тестирования и установки ПО. Для начала необходимо понять, на каком этапе требуется собирать разрабатываемое ПО. Допустим это будет отведения тега в репозитории – тогда мы сталкиваемся с первым инструментом GitLab-CI.

GitLab-CI – это встроенный в GitLab инструмент, который позволяет автоматизировать процессы сборки. У GitLab-CI есть несколько основных понятий – это pipeline (пайплайн), job'a (джоба), и stage (этап). Пайплайн – это набор джоб и этапов. Каждая джоба привязывается к своему этапу, для определения последовательности выполнения джоб. Джоба выполняется в отдельном докер-контейнере, на основе docker image, который задается при конфигурации пайплайна и джоб. В джобе описываются bash скрипты, которые будут выполняться на данном этапе. Это может быть вызов любых скриптов, основное условие, чтобы это было установлено в docker image. GitLab-CI пайплайн может быть вызван на любое событие, которые будет прописано в конфигурации. Событиями может быть отведение тега, название ветки, в которой был сделан коммит, и многое другое (рис. 2). Мы не будем углубляться в особенности этого процесса, поскольку они обычно определяются спецификой разрабатываемого ПО.

Для того чтобы построить процесс сборки, необходимо протестировать все обновления, которые появились в этом теге. Для самого начального уровня тестирования можно использовать линтеры, которые будут проверять синтаксис кода, это будет как первый этап в пайплайне, назовем его linters. После этого следует собрать написанный разработчиками код в patch/build. Так же можно описать это в джобе, вызвав bash/python скрипт.

В действительности build может быть описан в любом удобном виде, например в виде готовых docker-образов. Тогда необходимо описать команды, для создания этих докер-образов. Важно понимать, что для хранения докер-образов требуется использовать специальный репозиторий. Это может быть любой docker registry, например docker hub, где лежит большинство Open Source докер-образов.

Следующий этап – это развертывание докер-образов на каком-либо сервере, но, чтобы этого достичь, необходимо добиться того, чтобы этот сервер был сконфигурирован. Тут мы переходим к такому инструменту, как Terraform. Он позволит нам развернуть сервер в облачной инфраструктуре.

Terraform – это инструмент для управления инфраструктурой. В данном случае инфраструктура – это инстансы, сети и т.д. Инстанс EC2 – это ресурс с такими атрибутами, как тип машины, загрузочный образ, группа безопасности и зона доступности. Если простыми словами, Terraform позволяет развернуть сервер(а) по описанной конфигурации в файле, на таких облачных инфраструктурах, как Amazon и др. (рис. 3).

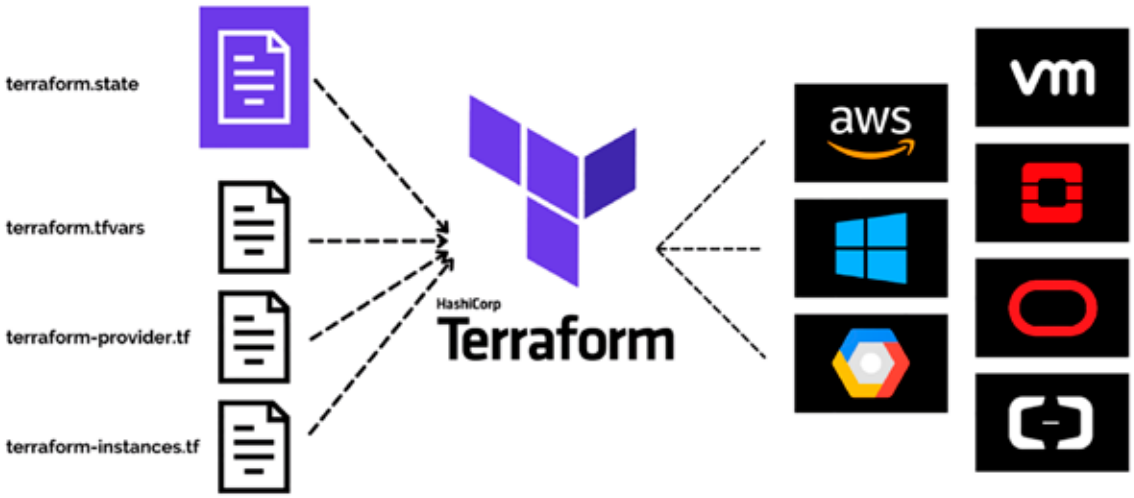

Рис. 3. Схема работы Terraform

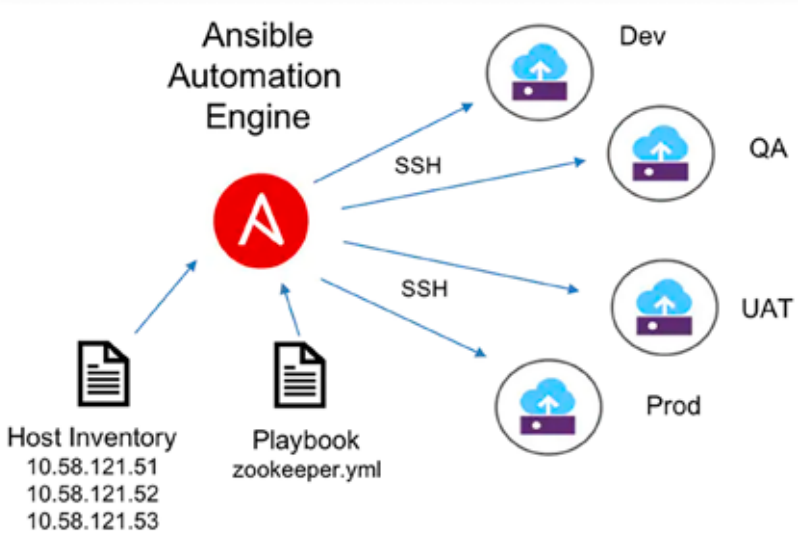

Рис. 4. Схема работы Ansible

Terraform может справиться с разворотом виртуальной машины и настройкой сети. Однако поставка необходимых third-party по типу protobuf или python3 не входит в возможности Terraform. Для этого следует использовать такой инструмент, как Ansible.

Ansible – система управления конфигурациями, написанная на языке программирования Python (рис. 4).

Данный инструмент позволяет нам описать порядок действий для установки и выполнения различных команд. Конфигурацию следует прописывать для каждого отдельного хоста. В ansible терминах конфигурация называется – inventory. Она содержит в себе информацию о хостах и переменных, которые будут распространяться на данный хост. Например, если необходимо установить ряд пакетов с помощью пакетного менеджера yum, следует использовать стандартный ansible модуль yum. С помощью данного инструмента мы можем выстраивать логику установки и ее конфигурацию для каждой рабочей станции (сервера) отдельно.

После развертывания виртуальной машины и установки на нее всего необходимого с помощью ansible следует переходить к этапу разворачивания docker-image (докер образов). Здесь обычно применяется инструмент Kubernetes [6]. Он служит для автоматического развертывания собранных docker-образов.

Kubernetes – это портативная расширяемая платформа, которая позволяет управлять контейнерами и конфигурировать рабочие нагрузки и сервисы (рис. 5).

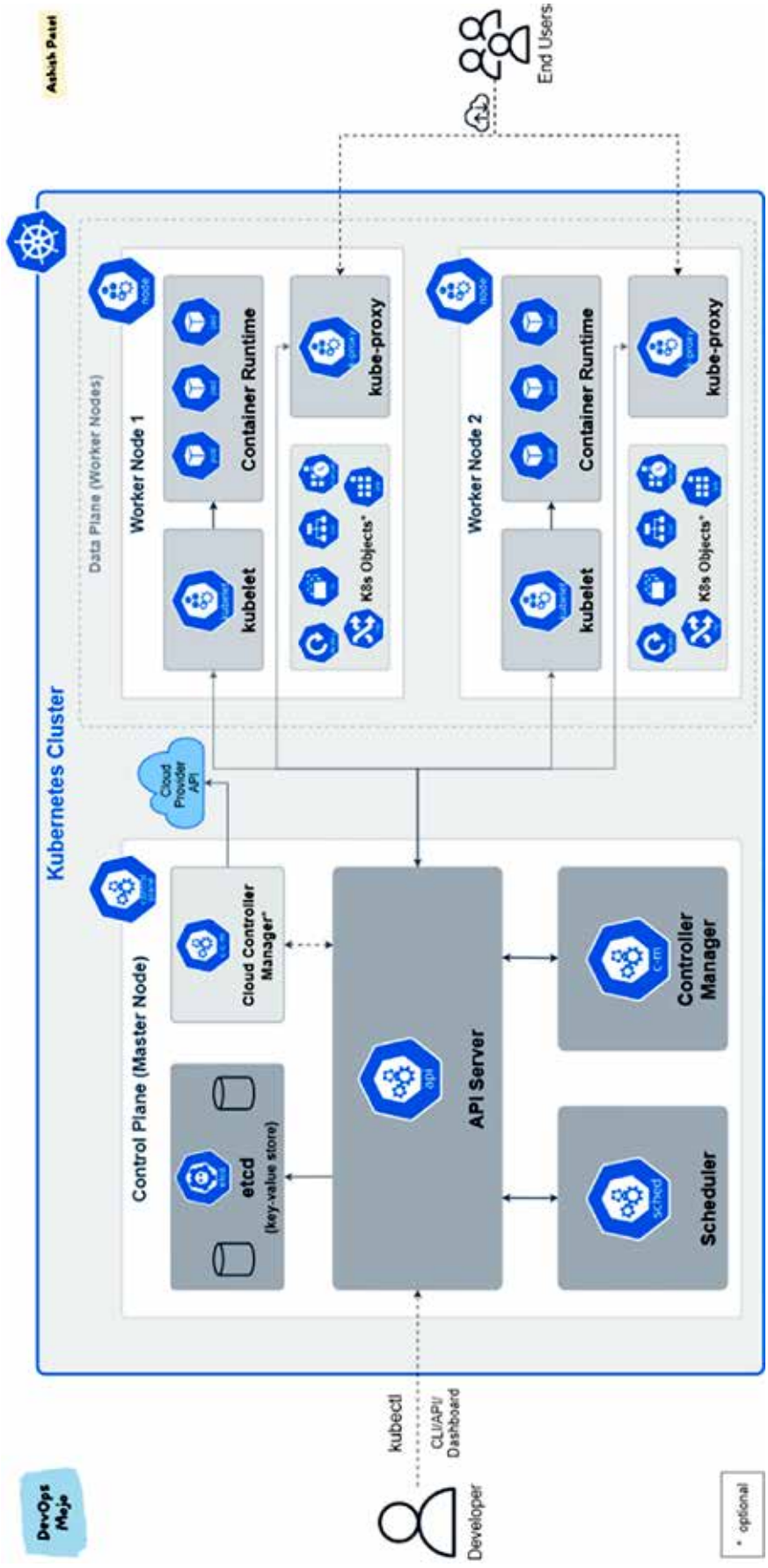

Рис. 5. Схема работы Kubernetes

В общем случае Kubernetes – это оркестратор для docker-контейнеров. Кроме того, он позволяет масштабировать приложение, производить мониторинг и распределение нагрузки между сервисами, а также осуществлять оркестрирование хранилища данных и пр.

После того, как специалисты DevOps (DevOps инженеры) развернули приложение, и оно работает в докер-контейнерах под управлением Kubernetes'a, следует начать тестирование приложения. Здесь обычно осуществляются request-запросы, какие-либо внутренние тесты и т.д., все это делается на уровне описания Gitlab-ci джобы.

Результаты исследования и их обсуждение

Результаты исследования можно представить в виде следующих основных стадий процесса разработки ПО в соответствии с методологией DevOps:

1. После многочисленных изменений и правок в коде отводится тег от главной ветки git репозитория (часто данный тег называется «release»).

2. С помощью GitLab-CI запускается пайплайн (по тегу), который несет в себе ряд джоб и этапов:

2.1. Этап проверки синтаксиса кода (linters) – проверяет код на синтаксические ошибки.

2.2. Тестирование ПО. Данный этап не был освещен подробно, однако это не снижает его важности в процессе разработки ПО. Есть разные методологии тестирования ПО. Предпочтение следует отдавать автоматизированным методам. Это позволит снизить риски ошибок на данном этапе и значительно ускорить процесс.

2.3. Этап сборки (build) – собираются докер-образы из кода, который находится в данной ветке (теге) и загружаются в docker registry.

2.4. Этап разворачивания сервера (infra_deploy) – в данном этапе используется Terraform для конфигурации машины в облачном сервисе, а после запускается Ansible скрипт, который позволит установить все необходимое для разворачивания docker-образов.

2.5. Этап установки (deploy) – на данном этапе устанавливаются (разворачиваются) докер-контейнеры на сервере с помощью оркестратора Kubernetes.

2.6. Этап тестирования (tests) – на данном этапе происходит тестирование приложения, это могут быть requests-запросы или же иные способы.

3. После полной отработки пайплайна получается готовая версия приложения на сервере, вместе с собранными образами, которые можно отправлять заказчику.

На этом процесс сборки и тестирования может считаться завершенным. Далее следует отправить готовый программный продукт на сторону заказчика. Для этого вместе gitlab-ci можно использовать Jenkins.

Jenkins – программная система с открытым исходным кодом на Java, предназначенная для обеспечения процесса непрерывной интеграции программного обеспечения. Основное отличие Jenkins'a от GitLab-CI в том, что gitlab-ci подходит больше для единой, не гибкой системы развертывания и тестирования. Jenkins же позволяет быть максимально гибким и безопасным. Он также имеет свои джобы и пайплайны, однако в данном случае не предусмотрено внесение изменений в репозиторий. При работе с Jenkins существует возможность выстроить аналогичный deploy процесс, как и в gitlab-ci, при этом указывая разные параметры и ссылки на конфигурации, что позволит правильно распределить сервера.

Заключение

В данном исследовании были рассмотрены основные аспекты работы DevOps-инженера и DevOps как методологии в целом. Кроме того, приведены основные этапы CI-CD процесса. Следует отметить, что методология DevOps на практике сталкивается с множеством тонкостей и проблем в своей работе, потому специалисту необходимо знать и понимать весь этот процесс, начиная от написания исходного кода и заканчивая установкой разрабатываемого ПО на сторону заказчика.

Список литературы

1. Дженнифер Д., Дэниел К. Философия DevOps. Искусство управления IT. СПб.: Питер, 2019. 600 с.

2. Вольф Э. Continuous delivery. Практика непрерывных апдейтов. СПб.: Питер, 2017. 320 с.

3. Ким Д., Дебуа П., Уиллис Д., Хамбл Д. Руководство по DevOps. М.: Манн, Иванов и Фербер (МИФ), 2018. 540 с.

4. Моуэт Э. Использование Docker. М.: ДМК-Пресс, 2017. 354 с.

5. Андреев Р. Виртуализация и контейнеризация: обзор технологий и в чем разница. [Электронный ресурс]. URL: https://timeweb.cloud/blog/virtualizaciya-i-kontejnerizaciya-raznica (дата обращения: 23.03.2023).

6. Основы Kubernetes. [Электронный ресурс]. URL: https://kubernetes.io/ru/docs/tutorials/kubernetes-basics/ (дата обращения: 23.03.2023).