DOI https://doi.org/10.15407/usim.2019.02.070
УДК 004.4

Г.В. ХОДЯКОВА, канд. пед. наук, доцент кафедры компьютерных наук и прикладной математики, Николаевский национальный ун-т имени В.А. Сухомлинского, Николаев, ул. Шнеерсона, 11, кв. 7, khodiakovagalina@gmail.com
В.А. ПОЗДЕЕВ, д-р физ.-мат. наук, зав. кафедрой компьютерных наук и прикладной математики, Николаевский национальный ун-т имени В.А. Сухомлинского,Николаев, ул. Никольская, кв. 24, 5403 pozdeevval@gmail.com
Н.В. ХОДЯКОВА, ведущий разработчик, Ray Sono AG, Брудерхофштрассе 3, 81371, Мюнхен, Германия, Nathalie.mk.ua@gmail.com

АНАЛИЗ ТЕХНОЛОГИЙ СОЗДАНИЯ ИНФРАСТРУКТУРЫ ДЛЯ РАЗРАБОТКИ И РАЗВЕРТЫВАНИЯ ПРОГРАММНЫХ ПРИЛОЖЕНИЙ

Рассмотрена тенденция развития инфраструктуры для создания и развертывания программных приложений и технологии использования виртуальных машин, контейнеров, платформ для развертывания приложений, оркестровщиков контейнеров, а также классификация перечисленных технологий. Приведен пример установки и использования Docker и два варианта создания Docker-контейнеров.

Ключевые слова: *среда разработки, виртуальные машины, Docker-контейнеры, платформы для развертывания приложений, оркестровщик контейнеров, микросервисы.*

Введение

В процессе создания приложений можно выделить сам процесс разработки и развертывание приложений, которое всегда было большой проблемой у разработчиков и системных администраторов. Практически во всех случаях среда разработки значительно отличается от окружения, в котором созданное приложение реально будет работать. Очевидно, что различные компьютеры будут сконфигурированы поразному, но приложение должно успешно стартовать на каждом. В среду выполнения входит операционная система, а также программные компоненты и сервисы, необходимые для запуска приложения, такие как библиотеки, сервер, базы данных и т.д. Известны ситуации, когда приложение работает на одном компьютере, но отказывается включаться на устройстве другого пользователя. Опишем подробнее проблемы, возникающие в процессе разработки и развертывания приложений.

Пример 1. В 90-х годах проблема могла быть вызвана тем, что *Microsoft Visual C++* версии 2.0 отмечает загрузочные модули создаваемых приложений как предназначенные для работы в среде *Windows NT*, в то время как приложения *Microsoft Windows* 95 должны быть отмечены по-другому. В результате окна приложения не получают некоторые извещения. Возникают также проблемы и с диалоговыми панелями.

Выход заключался в ручном редактировании параметров для каждого вновь создаваемого приложения.

Пример 2. Различия в версиях какого-либо программного обеспечения, используемого приложением (*web server, Java*), или различия при использовании протокола шифрованной передачи данных *TLS* также приводили к конфликтам. Например, версия 1.2 поддерживается еще не всеми продуктами, и поэтому одна сторона шифрованного сообщения пытается использовать *TLS v*1.2, а вторая — его не поддерживает и не может прочитать сообщение.

Пример 3. Разработчик работает параллельно над тремя проектами, каждый из которых требует разные версии базы данных. Установить все три на одном компьютере не представляется возможным. Либо нужно перед каждой сменой проекта удалять одну версию и устанавливать другую, либо пытаться установить все три одновременно, но в этом случае нет никакой гарантии, что установленные версии базы данных не будут конфликтовать между собой. Часто каждый из проектов требует много зависимостей (базы данных, другие приложения), установить все на один компьютер и запускать каждый раз всю инфраструктуру слишком затратно. Кроме того, зависимости должны стартовать в определенном порядке, объединяться в сеть, подключаться друг к другу.

Пример 4. Инфраструктуру, необходимую для работы над проектом, установили на “чистую” операционную систему и создали ее образ (**.iso* файл). Затем создали виртуальную машину (например, в *VirtualBox*) и запустили в ней образ ОС. Каждый новый разработчик должен скачать 20-Гигабайтный файл и создать свою копию виртуальной машины. При внесении изменений в эту инфраструктуру, обновлении какого-то компонента необходимо распространить новый файл среди разработчиков и проконтролировать обновление у каждого. На практике это сделать очень трудно и поэтому возникают многочисленные конфликты в коде, вызванные одновременной разработкой в разных версиях среды. На первом этапе решение указанных проблем осуществлялось путем написания необходимой программы на системном языке программирования или через подключение необходимых библиотек; позже — через использование виртуальных машин. Сейчас появляются новые средства, известные, как контейнерные технологии.

Обзор исследований

Архитектуру, принципы работы и процесс установки виртуальных машин описывают Джеймс Смит, Рави Наир, Гультяев А. К., Меркулов Ю., Михирев Д. [1–5] и многие другие авторы.

О новых технологиях, обеспечивающих совместимость среды разработки и запуска приложений, таких, как контейнерные технологии, можно найти публикации в Интернете: *Will Wang, Jatin Aneja, Alexander Volirik*, Игорь Новиков, Мартин Фаулер, Джеймс Льюис, *Christian Abdelmassih* и др. [6, 7, 9, 10]. Большая часть этих публикаций — на английском языке. Интернет-источники содержат также документацию, общие сведения, историю появления этих технологий, некоторые аспекты их использования [11–12].

Публикаций, посвященных обобщению и классификации технологий, сопровождающих процесс создания и запуска приложений, авторы не встречали.

Постановка задачи

Ежедневно растущий поток новых технологий инициируется все новыми и новыми задачами, возникающими перед разработчиками программного обеспечения. Стала популярной коллективная разработка приложений, создание программных продуктов на основе микросервисов, использование систем контроля версий. Процесс разработки переносится в Интернет и использует возможности и ресурсы облачных систем. В связи с тем, что микросервисы могут быть написаны на разных языках и использовать разные технологии хранения данных, требуется централизованное управ-

ление ими и легко настраиваемая среда для их развертывания. Соответственно, появляются новые средства для создания и запуска приложений: контейнеры, платформы для развертывания приложений, оркестровщики контейнеров, балансировщики нагрузки и др.

Поэтому целью данной статьи является анализ технологий организации инфраструктуры для создания и развертывания программных приложений и их классификация. В статье приводятся также некоторые примеры практической реализации новых технологий: установка *Docker* на операционную систему *Linux Ubuntu* и создание контейнеров.

Анализ технологий создания инфраструктуры для разработки и развертывания приложений

Развитие инфраструктуры для разработки и развертывания приложений направлено на то, чтобы полностью абстрагировать функции администрирования и чтобы заказчик мог выполнять свой код, не беспокоясь о среде разработки.

В 80–90 гг. в случае использования интерпретируемых языков программирования проблемы по настройке среды разработки решались путем написания специальной программы на системном языке программирования. Для компилируемых языков взаимодействие с вычислительным окружением реализовывалось набором подключаемых разделяемых библиотек среды выполнения.

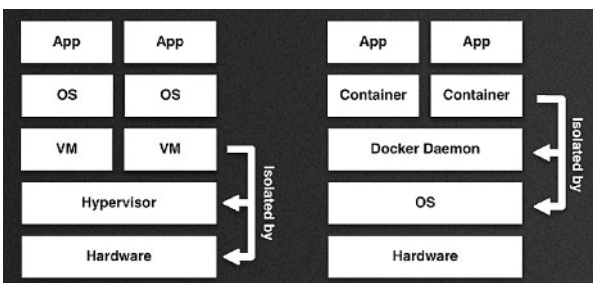

Рис. 1. Разделение ресурсов на разных уровнях инфраструктуры

В результате операционную систему нужно было настраивать, оптимизировать и интегрировать с одним-единственным приложением. В эти же годы появляются виртуальные машины (гипервизоры), которые, по сути, являются эмуляторами аппаратного обеспечения, а уже в конце 90-х годов начинает использоваться виртуализация приложений, как, например, *Java Virtual Machine или Parrot Virtual Machine*.

Виртуальные машины

Новая технология позволяла иметь на одном компьютере несколько виртуальных машин с совершенно разными операционными системами и их состояниями. Создание и настройка виртуальной машины (*VM*) требовали, учитывая параметры вычислительной системы, выполнить целый ряд шагов: указать имя *VM*, выбрать тип ОС, создать виртуальный жесткий диск, задать его объем, выбрать тип и формат хранения, настроить *VM*, выбрать общий буфер обмена, установить дополнения для операционной системы *VM*. Настройку среды разработки можно уже было делать на уровне «работы с меню», а не на уровне «команд». Однако, использование *VM* выявило их следующие недостатки [1]:

- создание *VM* — достаточно длительный процесс;
- возможная плата за предоставление дополнительного пространства;
- не все виртуальные машины поддерживают совместимое использование;
- *VM* занимает значительные ресурсы, а в большинстве случаев на сервер устанавливается несколько *VM*, которые занимают еще больше места.

VM по-прежнему активно используются в процессе разработки и запуска приложений, но становятся все более популярными контейнерные технологии.

Контейнеры

Контейнеры, как наследники виртуальных машин, в течение последнего 10-летия ис-

пользуются для развертывания приложений. Они напоминают *VM*, но реализованы на более высоком уровне и с меньшей изоляцией, их можно перенести в другую локальную среду или облако. Появление контейнеров сделало решение задачи по настройке среды разработки еще более простым.

В [10] приводится рисунок, на котором видно разделение ресурсов в случае использования виртуальных машин и контейнеров. Компьютер, на котором можно осуществлять основную обработку информации и/или который предоставляет сервисы типа «клиент-сервер» в режиме сервера, называется *host*-компьютером. Для каждой виртуальной машины используется собственная ОС, которую называют гостевой ОС, а для всех контейнеров применяется ядро одной операционной системы *host*-компьютера (рис. 1). Если для виртуальных машин изоляция нужна на уровне аппаратного обеспечения, то для контейнеров изоляция осуществляется в пределах операционной системы. При этом сохраняется изоляция и по отношению к другим контейнерам. Запуск контейнера выполняется в той же операционной среде, в которой работают остальные приложения.

Первой реализацией были контейнеры для *Linux*. Для создания виртуальной среды, с разделением процессов и сетевого пространства, взяли за основу преимущества *cgroups* и изоляцию пространств имен. "*control groups*" — это функция ядра *Linux*, которая изолирует и контролирует использование ресурсов для пользовательских процессов.

В статье, опубликованной в *The Register*, контейнер называют преемником виртуализации. Там же приводятся данные, что уже в 2014 г. *Google* поддерживал больше двух миллиардов контейнеров в неделю [9].

Docker — платформа для развертывания и запуска контейнерных приложений

Наиболее распространенная технология контейнеров — это *Docker* — среда для создания и запуска контейнеров. Это не единственная подобная платформа, но, бесспорно, одна из самых популярных и востребованных. Существуют и другие *open source* технологии контейнеров, например *rkt* от *CoreOS*. Крупные компании создают собственные продукты, например *lmctfy* от *Google*. *Docker* стал стандартом в этой области. Он был выпущен для *Linux* в 2013 г., поэтому она является рекомендуемой операционной системой для его установки. В 2016 г. *Docker* был портирован на *Windows Server*, затем на 64-битную версию *Windows* 10 и *macOS*.

Docker-контейнер состоит из слоев образов, при этом бинарные файлы упакованы в один пакет. В базовом образе содержится операционная система, она может отличаться от ОС *host*-компьютера и содержит файловую систему и бинарные файлы, в то время как в полноценной ОС, помимо этого, есть еще и ядро. Поверх базового образа лежит еще несколько образов, каждый из них является частью контейнера. Например, следующим после базового образа, может быть образ, который содержит команды установки зависимостей (например, с помощью *apt-get*). Следующим может быть образ с бинарными файлами приложения и т.д.

Кроме того, контейнеры – это отличный инструмент для реализации архитектуры микросервисов. И каждый микросервис представляет собой комплекс взаимодействующих контейнеров. Например, микросервис *Redis*, можно реализовать с одним *master*-контейнером и несколькими *slave*-контейнерами.

Docker обеспечивает использование ресурсов *host*-машины динамически, он не резервирует мощности, не бронирует ресурсы, а распределяет их. *Docker* подключается к ОС и разделяет ресурсы на все *Docker*-контейнеры, работающие как отдельные процессы. Вместо упаковки всей операционной системы со всем необходимым программным обеспечением, контейнеры упаковывают только приложение и его прямые зависимости. Это основная концепция, реализованная *Docker*. В одном контейнере *Docker* мы получаем доступ ко всем необходимым ресурсам: исходному коду, зави-

симостям и среде выполнения. Формат *Docker* похож на обычный *package*, за исключением того, что пакет является автономным, и несколько копий может быть запущено на одном или нескольких компьютерах. Такой подход максимально эффективно использует ресурсы, повышает производительность и снижает размер приложения. При этом *Docker* обеспечивает высокую степень изоляции, ограничивая проблемы приложения внутри самого контейнера, без ущерба для вычислительной системы.

До появления *Docker* при развертывании систем возникало множество неприятных и сложных вопросов, связанных с конфликтом версий и непонятными зависимостями. *Docker* позволяет упаковать образ состояния системы, развернуть его в производственной среде и там запустить. К преимуществам *Docker* можно отнести:

- стандартизированный механизм создания среды; ускоренный процесс разработки.
- нет необходимости устанавливать вспомогательные инструменты вроде *PostgreSQL*, *Redis, Elasticsearch*: их можно запускать в контейнерах;
- удобная инкапсуляция приложений;
- защита от потери информации, надежность;
- уменьшение времени ответа;
- понятный мониторинг;

Таблица 1. **Уровни абстрагирования от инфраструктуры в порядке возрастания абстракции**

Уровень	Технология
1	прямой доступ к аппаратным ресурсам (программа на *C*)
2	виртуальные машины (*Oracle Virtual Box*)
3	контейнеры (*Docker*)
4	оркестровщики контейнеров (*Kubernetes*)

- простое масштабирование;
- версия *Docker EE* (2017) работает в облачных средах *AWS* и *Azure*.

Docker отлично подходит для работы с контейнерами на одном *host*-компьютере. Но использование распределенных сервисов требует уже управления ресурсами и рабочими нагрузками на разных серверах и в сложных инфраструктурах.

Оркестровщики контейнеров

Процесс создания программных приложений в настоящее время требует от разработчика не только знания популярных языков программирования (*C++*, *Java, Python, PHP, Kotlin* и др.), но также умений по настройке *web*-серверов, серверов приложений, управления *docker*-контейнерами.

Управлять контейнерами, запущенными в окружениях со множеством узлов, можно, конечно, непосредственно, но это превращается в довольно трудоемкий процесс.

Поэтому все чаще в практике работы используют оркестровщики контейнеров, при этом предпочтение отдают *Kubernetes*. Он управляет и запускает контейнеры *Docker* на большом количестве компьютеров, обеспечивает совместное размещение, репликацию (возобновление) контейнеров, масштабирование, логическое группирование контейнеров, а также позволяет проверять «здоровье» контейнеров.

Проект был начат *Google* и теперь поддерживается многими компаниями, среди которых *Microsoft, RedHat, IBM* и *Docker*. В число предоставляемых *Kubernetes*-сервисов входят также балансировщики нагрузки. Они уменьшают нагрузку на отдельный сервер, распределяя трафик на нескольких серверах.

В процессе анализа технологий, позволяющих создать инфраструктуру для разработки и развертывания приложений, авторы решили классифицировать технологии по уровню абстрагирования от инфраструктуры, доступной разработчикам.

Таблица 2. **Установка *Docker* с помощью репозитория и его настройка**

№	Команда	Комментарий
1	*sudo apt-get update*	Обновление списка доступных для установки пакетов. *apt (Advanced Packaging Tool)* — это утилита в *Debian*-подобных системах
2	*sudo apt-get install*	Установка пакетов
3	*apt-transport-https*	Пакет, позволяющий *apt* использовать протокол *HTTPS* (для доступа к репозиторию *Docker*)
4	*ca-certificates*	*CA*-сертификаты
5	*curl*	*client URL* — это *URL*-адрес клиента, который позволяет вам подключаться к другим *URL*-адресам и получать ответы
6	*software-properties-common*	Пакет управления репозиториями для установки программ (общие файлы, драйверы и т.д.)
7	*curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -*	*apt-key* — утилита управления ключами *apt* эта команда скачивает ключ к репозиторию *docker* и добавляет его в *apt*
8	*sudo add-apt-repository*	Добавить репозиторий *docker* в *apt*
	*"deb [arch=amd64] https://download.docker.com/linux/ubuntu \ $(lsb_release -cs) *	*docker* для *debian [arch=amd64] (ubuntu)* *lsb* (*Linux Standard Base*) — определение дистрибутива *linux*
	stable"	*stable*" — версия ядра
9	*sudo apt-get update*	Обновление списка доступных для установки пакетов
10	*sudo apt-get install docker-ce*	Установить *Docker CE*
11	*sudo docker run hello-world*	Если установка *Docker* прошла успешно, протестировать его работу

Установка Docker на операционную систему Ubuntu

Приведем пример практической реализации процесса установки *Docker*-среды для создания и доставки пакетов образов контейнеров. Основой для выбора необходимых команд нам послужила документация по *Docker* [8]. Определение дистрибутива *Linux* происходит в ходе установки, поэтому для описанной процедуры установки *Docker* подходит любая версия *Ubuntu*. Установка занимает несколько минут.

```
sudo apt-get update
sudo apt-get install \
apt-transport-https \
```

```
ca-certificates \
curl \
software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"
sudo apt-get update
sudo apt-get install docker-ce
sudo docker run hello-world
```

Комментарии к командам установки

Docker CE можно установить по-разному, в зависимости от потребностей. Большинство пользователей настраивают репозитории *Docker* и устанавливают их, чтобы упростить задачи установки и обновления.

Это рекомендуемый подход. Некоторые пользователи скачивают пакет *DEB* и устанавливают его вручную, а также полностью управляют обновлениями. Это полезно в таких ситуациях, как установка *Docker* в системах *air-gap* — без доступа к Интернету.

Существует общедоступный репозиторий образов — *Docker hub* по адресу *https://hub.docker.com*. В этом репозитории представлены широко используемые образы, такие как: *redis, ubuntu, mysql, mongo, nginx* и т.д., на базе которых можно создавать контейнеры. Всего в репозитории находятся более 100 000 образов, предоставленных как официальными разработчиками продуктов, так и загруженных пользователями.

Примеры создания Docker-контейнеров

Пример 1. Создание контейнера из общедоступного образа

Ш а г 1. Локальный запуск *mongodb*

Ш а г 2. *Docker run* -p 27017:27017 *mongo*

Docker найдет последнюю версию образа *mongo* в репозитории *Docker hub*, загрузит файлы в локальный репозиторий и создаст контейнер на основании образа *mongo*, связывая порт 27017 контейнера (стандартный порт *MongoDB*) с портом 27017 *host*-машины. После запуска контейнера, подключиться к *mongodb* с *host*-машины можно по адресу *localhost*:27017. Запуск *MongoDB* с помощью *Docker* требует выполнение всего одной команды вместо установки полновесного сервера базы данных.

Пример 2. Создание собственного образа и контейнера на его основе.

В этом примере авторы рассматривают создание контейнера для запуска *Spring Boot* приложения. *Spring Boot — Java* фреймворк для разработки веб-приложений. Его преимуществом является поставка встроенного *Jetty*-сервера. После сборки приложения в *jar*- файл и его запуска, пользователь получает запущенный веб-сервер с работающим внутри приложением. *Docker* образ описывается в конфигурационном файле *Dockerfile*.

Ш а г 1. *FROM openjdk:8-jdk-alpine*
Ш а г 2. *ARG JAR_FILE*
Ш а г 3. *COPY ${JAR_FILE} app.jar*
Ш а г 4. *ENTRYPOINT* ["*java*","*-jar*","*/app.jar*"]

Базовым образом для нового контейнера выбран образ *openjdk* с версией 8-*jdkalpine*. Детали образа можно найти на *Docker hub*. Это легковесная *Linux*-система с предустановленной *JDK* 1.8.

Инструкция *ARG* определяет переменную, значение которой может быть передано сборщику во время создания образа. В этой команде ожидают, что аргумент с названием *JAR_FILE* будет предоставлен на момент сборки образа.

Копируем файл, путь к которому указан в переменной *JAR_FILE* на *host*-машине, в рабочую директорию образа под именем *app.jar*.

Рабочая директория образа по умолчанию — корневой каталог.

ENTRYPOINT позволяет создать исполняемый контейнер, при его запуске будет выполнена команда *java -jar/app.jar*. Затем, в

директории, в которой находится *Dockerfile*, исполним команду:

Шаг 1. *docker build -build-arg JAR_FILE=/path/to/myapp.jar -t test/myjavaapp*.

Docker образ *test/myjavaapp* создан в локальном репозитории.

Публикация образа в репозиторий (*Docker hub* или другой сконфигурированный репозиторий) осуществляется командой:

Шаг 2. *docker push test/myjavaapp*

Для запуска контейнера из образа, выполним команду:

Шаг 3. *docker run* -p 8080:8080 *test/myjavaapp*

Порт 8080 контейнера (стандартный порт сервера *Jetty*) связывается с портом 8080 *host*-машины. Веб-сервер с приложением доступен по адресу *localhost*:8080.

Список запущенных контейнеров выводится командой *docker ps*. Первая колонка вывода команды — идентификатор контейнера. Остановить контейнер можно командой *docker stop CONTAINTER_ID*, а снова запустить уже созданный — с помощью *docker start CONTAINTER_ID*.

Заключение

В данной статье описаны технологии, обеспечивающие совместимость среды разработки и запуска приложений, а также, в каком направлении в последнее время развивается инфраструктура создания и развертывания программных приложений. Это контейнеры, платформы для развертывания приложений, оркестровщики контейнеров. Рассмотренные технологии авторы классифицировали по уровню абстрагирования от инфраструктуры, дос-тупной разработчикам.

Приводится также пример практической реализации на компьютере процесса установки *Docker* и два варианта создания *Docker*-контейнеров. Статья будет полезна для разработчиков программного обеспечения, начинающих осваивать контейнерные технологии.

Описание других разрабатываемых разными корпорациями программных продуктов, поддерживающих названные технологии, сравнительный анализ их функциональных возможностей и популярность у разработчиков могут явиться предметом отдельного исследования.

ЛИТЕРАТУРА

1. *Гультяев А.К.* Виртуальные машины: несколько компьютеров в одном. Спб: Питер, 2006. 224 с.
2. *Джеймс Смит, Рави Наир*. Архитектура виртуальных машин. Открытые системы. 2005. https://www.osp.ru/os/2005/05-06/185586/
3. *Зосимов В.В.* Планировщики задач операционной системи Linux и перспективи их развития для эффективного ведення научных расчетов. Проблеми Моделювання. Збірник наук. праць. Вип. 70. Київ: Інститут проблем моделювання в енергетиці, 2013. С. 73–78.
4. *Меркулов Ю.* Виртуальная среда. CHIP, 2010. № 1 (130). С. 106–109.
5. *Михирев Д.* Второе лицо. ComputerBild, 2011. № 6 (129). С. 52–57.
6. *Новиков И.* Контейнеры или виртуальные машины, что выбрать? itWeek. https://www.itweek.ru/infrastructure/article/detail.php?ID=188866 (14 дек.2018).
7. *Jatin Aneja*. Container Technologies Overview. Dzone. https://dzone.com/articles/container-technologies-overview (20 янв. 2019).
8. *Will Wang.* Demystifying containers 101: a deep dive into container technology for beginners. freeCodeCamp. https://medium.freecodecamp.org/demystifying-containers-101-a-deep-dive-into-container-technology-for-beginners-d7b60d8511c1 (19 нояб. 2018).
9. *Docker Documentation*. docker docs. https://docs.docker.com/ (24 нояб. 2018).
10. *Jack Clark*. Google: 'EVERYTHING at Google runs in a container'. The register. https://www.theregister.co.uk/2014/05/23/google_containerization_two_billion/ (5 нояб. 2018).

11. *Zosimov V. Khrystodorov O., Bulgakova O.* Dynamically changing user interfaces: software solutions based on automatically collected user information. Programming and Computer Software, vol 44 (6), 2018. P. 492–498.
12. *Zosimov V., Khrystodorov O., Bulgakova O.* Technology of web applications based on the cyber-entities identification. sist.ma , 2018, Issue 3 (275), P. 51–59.

Поступила 19.02.2019

REFERENCES

1. *Gultyaev, A.K.*, 2006. Virtualnyie mashinyi: neskolko kompyuterov v odnom. Spb:Piter, 224 p. (In Russian).
2. *Smit, Dzh., Nair, R.*, 2005. Arhitektura virtualnyih mashin. Otkryityie sistemyi. 2005. [online] Available at: <https://www.osp.ru/os/2005/05-06/185586/> [Accessed 1 Dec. 2018]. (In Russian).
3. *Zosimov, V.V.*, 2013. "Planirovshhiki zadach operaczionnoj sistemi Linux i perspektivi ikh razvitiya dlya e`ffektivnogo vedennya nauchny`kh raschetov". Problemi Modelyuvannya. Zbi`rnik naukovikh pracz`, 70. Kiyiv: I`nstitut problem modelyuvannya v energeticzi, pp. 73–78. (In Ukrainian).
4. *Merkulov, Yu.*, 2010. "Virtualnaya sreda". CHIP, 1 (130), pp. 106–109. (In Russian).
5. *Mikhirev, D.*, 2011. "Vtoroe litso". ComputerBild, 6 (129), pp. 52–57. (In Germany).
6. *Novikov, I.* Konteyneryi ili virtualnyie mashinyi, chto vyibrat? itWeek. [online] Available at: <https://www.itweek.ru/infrastructure/ article/detail.php?ID=188866> [Accessed 14 Dec. 2018].
7. *Aneja, J.*. Container Technologies Overview. Dzone[online] Available at: <https://dzone.com/articles/container-technologies-overview> [Accessed 20 Jan. 2019].
8. *Wang, W.* Demystifying containers 101: a deep dive into container technology for beginners. freeCodeCamp. [online] Available at: <https://medium.freecodecamp.org/demystifying-containers-101-a-deep-dive-into-container-technology-for-beginners-d7b60d8511c1> [Accessed 19 Nov. 2018].
9. *Docker Documentation*. docker docs. [online] Available at: <https://docs.docker.com/> [Accessed 24 Nov. 2018].
10. *Clark, J.* Google: 'EVERYTHING at Google runs in a container'. The register. [online] Available at: <https://www.theregister.co.uk/2014/ 05/23/google_containerization_two_billion/> [Accessed 5 Nov. 2018].
11. *Zosimov, V. Khrystodorov,O., Bulgakova, O.*, 2018. "Dynamically changing user interfaces: software solutions based on automatically collected user information". Programming and Computer Software, 44 (6), pp. 492–498.
12. *Zosimov, V., Khrystodorov, O., Bulgakova, O.*, 2018. " Technology of web applications based on the cyber-entities identification.". Upravl ie sistemy i ma iny, 3 (275), pp. 51–59. (In Ukrainian).

Received 19.02.2019

G.V. Khodiakova, Doctor of Philosophy in Education, Associate Professor at the Department
of Computer Science and Applied Mathematics of the V.O. Sukhomlynsky Mykolaiv National University,
Mykolaiv, 11 Shneerson Str., Apt. 7,
khodiakovagalina@gmail.com
V.A. Pozdeev, Doctor of Phys.-Math. Sciences, Chief of Department for applied mathematics and
information computer technologies V.O. Sukhomlynsky Mykolaiv National University
Mykolaiv, Nikolskaya Str., 24, 54030
pozdeevval@gmail.com
N.V. Khodiakova, Senior Software Developer at Ray Sono AG,
Bruderhofstrasse 3, 81371, Munich, Germany,
Nathalie.mk.ua@gmail.com

TECHNOLOGY ANALYSIS OF SOFTWARE DEVELOPMENT AND OPERATIONS INFRASTRUCTBRE

Introduction. In software engineering the development environment almost always differs from the runtime environment. This problem is solved through creating the environment-independent applications.

Purpose. This article aims at considering evolution dynamics and classification of modern means and technologies that ensure compatibility of development environment and applications that are developed and executed in this environment.

Methods. In this article various technologies for solving problems of creating environment-independent software with the purpose of abstracting management tasks are analyzed, widely used virtual machines and their drawbacks are briefly overviewed. A summary and classification of other means accompanying software development and deployment are given.

Results. The evolution trends of modern means and technologies that ensure the compatibility between development environment and applications that are developed and executed in this en-vironment such as virtual machines, container technologies and their classification are considered. The trends of software creation and operation infrastructure are described, abstraction levels of in-frastructure available to developers are marked out. A practical example of Docker container management system installation and the usage is provided, as well as of a load balancer operation mechanism.

Conclusion. The description of the currently developed software products that support technologies mentioned above, the comparative research of their functional abilities and the usage popularity among software developers and system administrators can be a subject of a separate research.

Keywords*: *development environment, virtual machines, Docker-containers, application deployment platforms, container orchestration, micro services.

Г.В. Ходякова, канд. пед. наук, доцент кафедри комп'ютерних наук
і прикладної математики, Миколаївський національний ун-т
імені В.О. Сухомлинського, Миколаїв, вул. Шнеєрсона, 11, кв. 7,
khodiakovagalina@gmail.com
В.О. Поздєєв, д-р физ.-мат. наук, зав. кафедрою комп'ютерних наук
і прикладної математики, Миколаївський національний ун-т
імені В.О. Сухомлинського, Миколаїв, вул. Нікольська, кв. 24, 54030
pozdeevval@gmail.com
Н.В. Ходякова, ведучий розробник, Ray Sono AG,
Брудерхофштрассе 3, 81371, Мюнхен, Німеччина,
Nathalie.mk.ua@gmail.com

АНАЛІЗ ТЕХНОЛОГІЙ СТВОРЕННЯ ІНФРАСТРУКТУРИ ДЛЯ РОЗРОБКИ І РОЗГОРТАННЯ ПРОГРАМНИХ ДОДАТКІВ

Вступ. В процесі створення додатків середовище розробки майже завжди значно відрізняється від оточення, в якому створений додаток реально буде працювати. Ця проблема вирішується через створення незалежності програми від середовища виконання.

Мета статті. Аналіз технологій організації інфраструктури для створення і розгортання програмних додатків та їх класифікація.

Методи. У статті проводиться огляд різних технологій для вирішення завдань створення незалежності програми від середовища виконання з тим, щоб повністю абстрагувати функції адміністрування. У цьому напрямку розвивається інфраструктура створення і використання програмних продуктів.

Результати. Описано засоби і технології, що забезпечують сумісність середовища розробки та запуску додатків, а також, в якому напрямку останнім часом розвивається інфраструктура створення та розгортання програмних додатків. Це контейнерні технології, платформи для розгортання додатків, оркестровщик контейнерів. Розглянуті технології ми класифікували за рівнем абстрагування від інфраструктури, доступної розробникам. Наводиться також приклад практичної реалізації на комп'ютері процесу установки *Docker* і два варіанта створення *Docker*-контейнерів.

Висновки. Детальний опис інших розроблюваних різними корпораціями програмних продуктів, що підтримують названі технології, порівняльний аналіз їх функціональних можливостей і популярність у розробників і системних адміністраторів може стати предметом окремого дослідження.

Ключові слова*: *середовище розробки, віртуальні машини, Docker-контейнери, платформи для розгортання додатків, оркестровщик контейнерів, мікросервіси.