УДК 004.05
МРНТИ 20.51.19

https://doi.org/10.55452/1998-6688-2026-23-1-22-36

[1]*Кусепова Л.Т.,
ст. преподаватель, ORCID ID: 0000-0002-7972-3169,
e-mail: kussepova_lt@enu.kz
[1]Кусепова Г.Т.,
ст. преподаватель, PhD, ORCID ID: 0000-0001-9556-8763,
e-mail: kussepova_gt_2@enu.kz

[1]Евразийский национальный университет им. Л.Н. Гумилева,
г. Астана, Казахстан

АНАЛИЗ И СРАВНЕНИЕ ИНСТРУМЕНТОВ ОРКЕСТРАЦИИ КОНТЕЙНЕРОВ В ЛОКАЛЬНОЙ И ОБЛАЧНОЙ ИНФРАСТРУКТУРЕ

Аннотация

В крупных облачных средах виртуализация играет ключевую роль не только в развертывании приложений и распределении рабочих нагрузок, но и в эффективном управлении сервисами и ресурсами платформы. Использование технологии контейнеризации наряду с такими традиционными подходами, как виртуальные машины, становится все более популярным в современных облачных инфраструктурах. Контейнеры обеспечивают более простой и быстрый способ виртуализации по сравнению с виртуальными машинами, ускоряя процесс внедрения и управления приложениями в изолированной среде, тем самым позволяют им эффективно использовать ресурсы. Для разработчиков контейнеры представляют удобное решение для тестирования компонентов приложений и построения архитектур на основе микросервисов. В результате контейнеризация становится предпочтительным решением для современных облачных сред и актуальных бизнес-потребностей. Поэтому данная работа посвящена изучению современных и известных на сегодняшний день контейнерных технологий. В статье анализируются современные технологии контейнеризации и оркестровки с использованием таких параметров, как производительность контейнера, масштабируемость и управление ресурсами. Основное внимание уделено оркестраторам Docker Swarm, Kubernetes, Apache Mesos и множеству контейнерных решений, исследовались преимущества и недостатки каждой технологии. Данная работа будет полезна ИТ-специалистам при выборе наиболее подходящего инструмента для создания приложений и реализации их облачных стратегий.

Ключевые слова: контейнерные технологии, оркестрация контейнеров, облачные вычисления, сравнительный анализ.

Введение

В условиях увеличения объемов обрабатываемых данных и высокой вариативности рыночной среды технология контейнеризации способствует обеспечению изоляции приложений, гибкости их развертывания и масштабируемости. Контейнеризация является важнейшим компонентом облачных систем. Интеграция контейнерных технологий в облачные вычисления и распределенные системы позволяет адаптировать системы к динамически изменяющимся нагрузкам, а также оптимизировать использование вычислительных ресурсов. Высокая нагрузка и множественные запросы, поступающие за короткий промежуток времени порождает падение сервера, который в реальном производстве негативно влияет на экономические показатели компании, снижая эффективность бизнес-процессов и приводя к потерям. Данная технология позволяет организациям соответствовать необходимым требованиям по устойчивости систе-

мы, сокращению времени отклика, а также по возможности быстрого восстановления после сбоев. Для управления процессами развертывания и распределения ресурсов используются технологии оркестрации, которые обеспечивают автоматизацию, коррекцию распределения запущенных контейнеров и решение проблемы с управлением ресурсов. Координация контейнеров и их оркестрация осуществляются такими популярными платформами, как Amazon Web Services (AWS), Google Cloud Platform (GCP), Azure, и другими облачными сервисами. В работе Florian Hofer et al. (2021) и Rui Queiroz et al. (2023) утверждается, что контейнерные технологии упрощают процесс виртуализации за счет быстрого развертывания приложений из пакетов, доступных в репозиториях, а также легкого переноса приложений в различные среды с небольшими изменениями в их конфигурации.

В статье рассматриваются инструменты оркестрации контейнеров, а именно Docker Swarm, Kubernetes и Apache Mesos, а также контейнерные сервисы облачных систем. Исследование сосредоточено на анализе производительности этих инструментов, включая временные затраты на развертывание, использование вычислительных ресурсов и показатели времени отклика. Цель данной работы состоит в том, чтобы определить характеристики каждого инструмента и выявить наиболее эффективные решения для оркестрации контейнеров.

В данном исследовании были сформулированы следующие исследовательские вопросы, направленные на оценку эффективности инструментов оркестрации контейнеров:

1. Какие архитектурные шаблоны и стратегии используются при реализации контейнеризованных приложений?
2. Какова сравнительная скорость развертывания приложений при использовании Docker Swarm по сравнению с другими платформами оркестрации контейнеров?
3. Как Kubernetes соотносится с другими инструментами оркестрации с точки зрения рабочей нагрузки и масштабирования приложений?

Результаты исследования помогут пользователям и системным администраторам определить наиболее подходящие инструменты оркестрации контейнеров, а также систематизировать знания о контейнерных технологиях в облачных вычислениях, выявить преимущества и возможности различных инструментов и их эффективность в решении определенных задач.

С учетом постоянного развития методов контейнеризации возникает необходимость регулярного изучения и анализа наиболее эффективных решений в данной области. В связи с этим был проведен анализ научных статей, в которых были проведены исследования по производительности контейнерных систем.

Материалы и методы

В данной работе были применены количественные и качественные методы анализа, направленные на оценку эффективности инструментов оркестрации контейнеров в локальной и облачной инфраструктуре. Анализ охватывал современные контейнерные технологии, включая Docker Swarm, Kubernetes, Apache Mesos, Amazon Elastic Kubernetes Service, Azure Kubernetes Service, Google Kubernetes Engine. В ходе экспериментов оценивались время запуска приложений, потребление вычислительных ресурсов, устойчивость к нагрузке и масштабируемости. Исследование состоит из следующих основных этапов:

1. Анализ научной литературы. Были изучены существующие научные работы, практические кейсы с развертыванием и управлением контейнерных приложений.
2. Сравнение функциональных характеристик и тестирование инструментов оркестрации контейнеров.
3. Анализ результатов тестирования.

Для развертывания контейнерных оркестраторов применялась вычислительная среда со следующими техническими характеристиками (таблица 1).

Таблица 1 – Экспериментальная среда

Параметр	Физическая машина	Виртуальная машина
Процессор	AMD Ryzen7 5800H	
Число ядер	16	8
Оперативная память	32	20
Операционная система	Windows 11 PRO	Ubuntu 24.04
Диск	1 Тб SSD	25 Гб
Сетевая карта	Realtek RTL	Intel PRO

Технологии контейнеризации пользуются большим спросом в последнее десятилетие, что подтверждается данными из отчетов исследовательской и консалтинговой компании Gartner. В связи с этим на данную тематику был проведен анализ научной литературы, ниже приведены основные результаты этих трудов. К одним из таких исследований относится работа Malviya A. и др. [1], в которой отмечается, что методы контейнеризации в облачных вычислениях очень важны, но процесс оркестрации контейнеров представляет собой сложную задачу. В этой статье анализируются инструменты оркестрации, такие как Docker Swarm, Kubernetes, Mesos и Redhat OpenShift, с позиции безопасности, стабильности, масштабируемости, настройки кластера и внедрения. В результате анализа было установлено, что Docker Swarm показал хорошие результаты при низкой нагрузке, а также легко управляем. Kubernetes демонстрирует лучшие функции планирования, Mesos обеспечивает эффективное масштабирование, а OpenShift предоставляет высоконадежные инструменты оркестрации.

Для эффективного управления и контроля контейнерных приложений требуется интеграция различных компонентов контейнерной инфраструктуры. В Docker интеграция осуществляется за счет взаимодействия Docker daemon и Docker API server, а в Kubernetes – посредством основных компонентов kube-apiserver, kubelet и etcd. Взаимодействие между этими компонентами производится через HTTP-интерфейсы (например, REST API), обеспечивая масштабируемость и гибкость системы [2, 3].

Сравнительный анализ платформ оркестрации контейнеров был проведен в работе Pankowski A., Powroznik P. [4], в которой исследованы контейнерные оркестраторы Docker Swarm, Kubernetes и Apache Mesos. В рамках исследования были измерены ключевые параметры, такие как потребление памяти, загрузка центрального процессора, использование дисковых ресурсов, а также рассмотрены производительность и эффективность каждой технологии. В свою очередь, Acharya J.N. и Suthar A.C. [5] изучили инструменты оркестрации контейнеров для планирования и управления контейнерами в микросервисных приложениях.

Преимущества и недостатки системного и прикладного контейнера рассмотрели авторы Moravchik M. и другие [6], сравнительно исследовав технологии LXC и Docker. Putri A.R. и т.д. [7] изучили контейнерные решения Docker, LXC и LXD, ориентируясь на общие показатели производительности. Koziolek H. и Eskandani N. [8] рассмотрели облегченные дистрибутивы инструмента оркестрации контейнеров Kubernetes, MicroK8s, k3s, k0s и MicroShift, включая использование ресурсов, управление в стрессовых ситуациях и производительность пространства данных. Работа показывает, что в то время как k3s и k0s имеют высокую пропускную способность в пространстве управления, MicroShift обеспечивает высокую пропускную способность в пространстве данных с позиции таких факторов, как простота использования, безопасность и возможность восстановления в случае сбоя.

Mercl L. и Pavlik J. [9] предоставляют подробную информацию об оркестраторах контейнеров, таких как Docker Compose, Docker Swarm, Fleet, Kubernetes и Apache Mesos, анализи-

руя их масштабируемость, отказоустойчивость, высокую доступность и сетевые подключаемые модули. Особое внимание уделено управлению контейнерами в частных и публичных облачных кластерах. В последние годы облачные вычисления привлекли значительное внимание благодаря экономичности предоставляемых услуг. В свою очередь, Sen A. и соавторы [10] предложили офлайн-оценку рисков облачных сервисов, направленную на выявление потенциальных угроз. Кроме того, следующие авторы научных работ [11–13] представили подходы и методику оценки качества облачных точечных данных, основанные на многопроекционном анализе, который учитывает различные аспекты точности и надежности.

Интерес к контейнерным технологиям также наблюдается и в области высокопроизводительных вычислений (HPC). В работе Liu P. и Guitart J. [14] рассмотрены Docker и Singularity для быстрой и гибкой реализации своих рабочих нагрузок. Авторы установили, что использование нескольких контейнеров с полной настройкой повышает производительность приложений с низкоуровневым межпроцессным взаимодействием, тогда как технологии контейнеризации, устанавливающие изолированные сетевые пространства имен, демонстрируют снижение производительности приложений.

Аналогичные исследования были проведены в направлении высокопроизводительных вычислительных систем. Abraham S. и другие авторы [15] провели эмпирический анализ современных контейнерных решений, таких как Docker, Podman, Singularity и Charliecloud. В результате был сделан вывод, что Charliecloud превосходит другие контейнерные решения по времени запуска контейнера, а Charliecloud и Singularity эквивалентны по пропускной способности ввода-вывода. На основании этой работы известно, что повышение производительности контейнеров и приложений, использующих возможности машинного обучения и глубокого обучения, может быть достигнуто путем выявления и реализации стратегий оптимизации. Хотя контейнерные технологии упрощают разработку и управление, их открытость может привести к проблемам и породить уязвимость в безопасности, если не была обеспечена надлежащая защита.

В области обеспечения безопасности контейнеров Wong A.Y. и другие авторы (2021) [16] в своей работе предложили моделировать уязвимость с помощью метода STRIDE. Данное исследование считается одной из работ, в которой проведено комплексное исследование уязвимостей контейнеров и мер по организации действий против выявленных угроз. Допущенная ошибка в настройке конфигураций может привести к серьезным нарушениям безопасности контейнеров.

Parast F.K. и соавторы [17] провели всесторонний обзор вопросов безопасности облачных вычислений в рамках сервисно-ориентированной модели, рассмотрели ключевые угрозы и меры их предотвращения. В свою очередь, Zhang C. et al. [18] и Sen A. et al. [19] исследовали модели оценки рисков для облачных сетей Петри и проектирования приложений.

Архитектурные особенности контейнерных оркестраторов. На рисунке 1 представлена общая схема процесса развертывания систем Kubernetes, Docker Swarm и Apache Mesos. Процессы развертывания этих трех инструментов во многом схожи и включают этапы загрузки образа (Dockerfile, containerd), конфигурации файла развертывания и измерения производительности. Реализация данных процессов требует базовых знаний контейнерных технологий, основанных на Docker. По изображению можно увидеть отличие процессов Apache Mesos от Kubernetes и Docker Swarm, его развертывание требует использования Terraform в качестве внешнего инструмента для корректной настройки и управления инфраструктурой.

Все фреймворки оркестрации контейнеров функционируют на разных платформах, но их поддержка осуществляется только на Linux, при этом они могут быть использованы для тестирования и разработки приложений в операционных системах Windows или macOS. Как в локальной, так и в облачной инфраструктуре контейнеризированное приложение можно развернуть, создав образ с помощью Dockerfile файла (рисунок 2) и создать deployment.yaml файл для настройки контейнеров и служб с определенным количеством реплик.

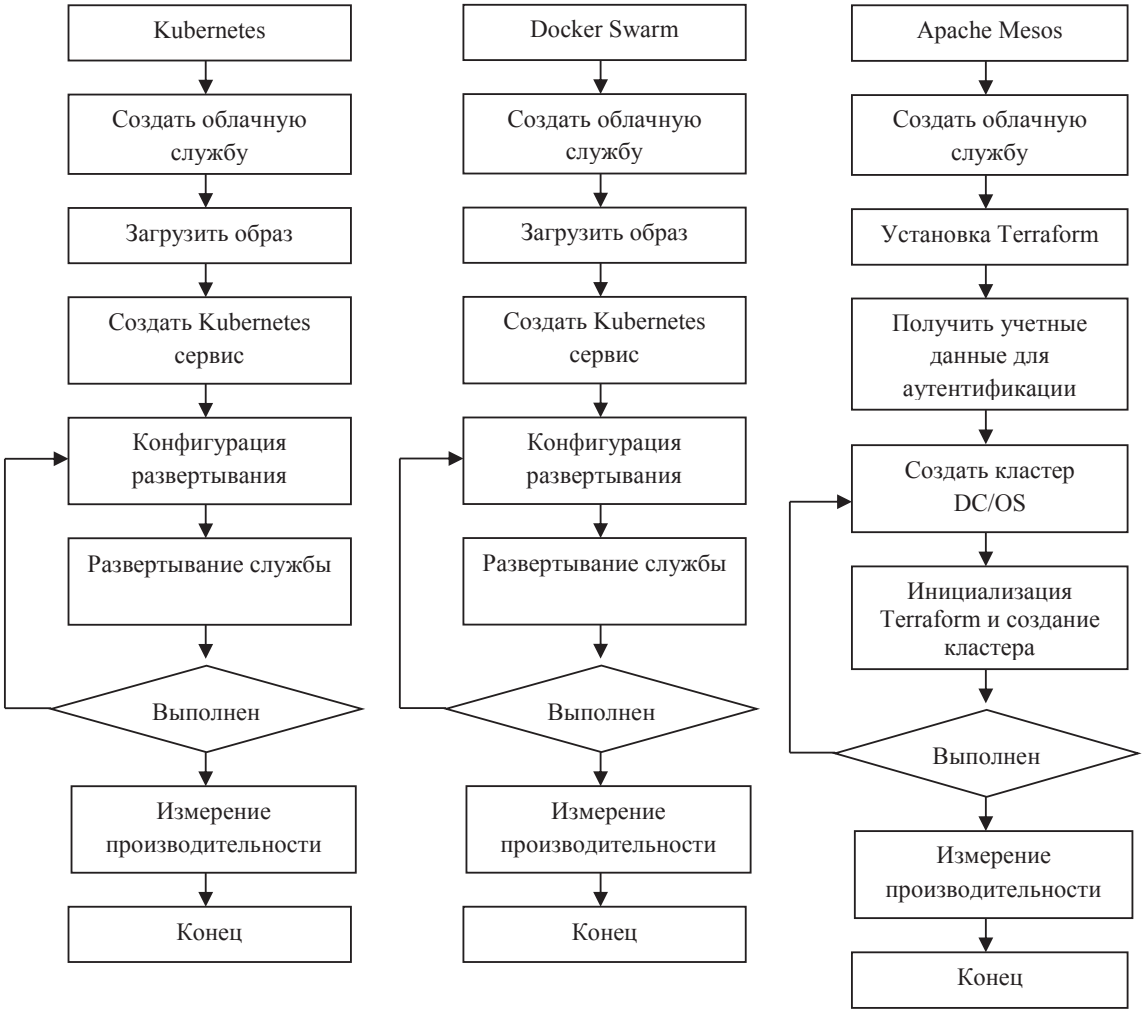

Рисунок 1 – Процесс развертывания и измерения производительности оркестраторов контейнера

```
FROM openjdk:11-jre-slim

WORKDIR /app

COPY target/english-learning-1.0-SNAPSHOT.jar app.jar

ENTRYPOINT ["java", "-jar", "app.jar"]
```

Рисунок 2 – Dockerfile

Kubernetes является одним из наиболее популярных и динамично развивающихся проектов в области контейнеризации и оркестрации контейнеров. Он представляет собой распределенную систему управления контейнеризованными приложениями, архитектура которой включает два компонента: главный узел и рабочий узел. По умолчанию кластер содержит один главный узел, который выполняет функции контроля для других подчиненных узлов. В каждом рабочем узле находятся контейнеризованные приложения, которые развернуты и инкапсулированы в pod. Главный узел состоит из kube apiserver, kube controller manager, cloud controller manager и kube scheduler. Функционал главного узла заключается в непрерывном мониторинге состояния кластера и управлении ресурсами. Kubernetes разработан на Go и использует YAML в качестве языка сериализации для создания deployment файла (представлен

на рисунке 3). Проект получает широкую поддержку со стороны ведущих мировых компаний, таких как Google, IBM, Amazon, Microsoft, Cisco, Red Hat, которые в свою очередь способствуют развитию облачных сервисов и интеграции с определенными инфраструктурными платформами.

а)

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: english-learning-app
spec:
    replicas: 5
    selector:
        matchLables:
            app: english-learning
    template:
        metadata:
            labels:
                app: english-learning
        spec:
            containers:
            - name: english-learning-app
              image: gcr.io/spring-
project-id-1/english-learning-app
              ports:
              - containerPort: 8086
---
apiVersion: v1
kind: Service
metadata:
    name: english-learning-service
spec:
    type: LoadBalancer
    ports:
        - port: 80
          targetPort: 8086
    selector:
        app: english-learning
```

б)

```
apiVersion: apps/v1
kind: Deployment
metadata:
    annotations:
        deployment.kubernetes.io/revision: “3”
        kubectl.kubernetes.io/last-applied-configuration: [
            {“apiVersion”: “apps/v1”, “kind”: “Deployment”, “metadata”:
        {“annotations”: {}, “name”: “english-learning-app”, “namespace”: “default”},
        “spec”:{“replica”: 5, “selector”: {“matchLabels”: {“app”: “english-learning”}},
        “template”: {“metadata”:{“labels”: {“app”: “english-learning”}}, “spec”:
        {“containers”: [{“image”: “gcr.io/spring-project-id-1/english-learning-app”,
        “name”: “english-learning-app”, “ports”: [{“containerPort”: 8086}]}]}}}}
    creationTimestamp: “2024-10-30T17:09:21Z”
    generation: 3
    managedFields:
    -    apiVersion: apps/v1
        fieldsType: FieldsV1
        fieldsV1:
            f: metadata:
                f:annotations:
                    .: {}
                    f: kubectl.kubernetes.io/last-applied-configuration: {}
            f: spec:
                f: progressDeadlineSeconds: {}
                f: replicas: {}
                f: revisionHistoryLimit: {}
                f: selector: {}
                f: strategy:
                    f: rollingUpdate:
                        .: {}
                        f: maxSurge: {}
                        f: maxUnavailable: {}
                    f: type: {}
            f: template:
                f: metadata:
                    f: labels
                        .: {}
                        f: app: {}
            f: spec:
                f: containers:
                    k: {“name”: “english-learning-app”}:
                        .: {}
                        f: image: {}
                        f: imagePullPolicy: {}
                        f: name: {}
                        f: ports:
                            .: {}
                            k: {“containerPort”: 8086, “protocol”: “TCP”}:
                                .: {}
                                f: containerPort: {}
                                f: protocol: {}
                        f: resources: {}
                        f: terminationMessagePath: {}
```

Рисунок 3 – а) Deployment.yaml, созданный в локальной инфраструктуре,
б) Фрагмент файла Deployment.yaml, созданный в облачной инфраструктуре

Наиболее распространенным оркестратором контейнеров в облачных системах на сегодняшний день является Kubernetes. Данный факт подтверждается данными, представленными в таблице 2, в которой перечислены контейнерные технологии, используемые различными платформами и облачными провайдерами.

Таблица 2 – Контейнерные технологии, используемые платформами/провайдерами

Платформа/провайдер	Контейнерные технологии
Amazon Elastic Container Service	Docker
Amazon Elastic Kubernetes Service	Kubernetes
VMWare Tanzu	Kubernetes
Google Kubernetes Engine	Kubernetes
Azure Kubernetes Service	Kubernetes
Red Hat Openshift Kubernetes Engine	Kubernetes
Red Hat Openshift	Docker, Kubernetes

Продолжение таблицы 2

Tencent Kubernetes Engine	Kubernetes
Alibaba Cloud Container Service for Kubernetes	Kubernetes
Azure Red Hat Openshift	Docker, Kubernetes
IBM Red Hat Openshift on IBM Cloud	Docker, Kubernetes
IBM Cloud Kubernetes Service	Kubernetes
Huawei Cloud Container Engine	Kubernetes
Hetzner Cloud	Kubernetes

Docker Swarm является платформой оркестрации контейнеров с открытым исходным кодом, она позволяет объединять несколько экземпляров Docker в единый виртуальный хост и обеспечивает централизованное управление контейнерами, адаптируясь к потребностям пользователей, кроме этого автоматизирует процессы балансировки нагрузки в кластерах. Архитектура кластера Docker Swarm включает следующие ключевые компоненты: узлы, службы и задачи. Docker Swarm не поддерживает автоматическое масштабирование, изменение количества реплик служб осуществляется только вручную с использованием соответствующих команд. Процесс управления развертыванием включает три основные стороны взаимодействия: администратор, оркестратор, диспетчер. Администратор конфигурирует параметры развертывания, включая выбор базового образа контейнера. Оркестратор выполняет развертывание контейнеров и управление кластером. Диспетчер осуществляет координацию узлов и распределение задач между рабочими узлами, а также обеспечивает развертывание сервисов на основе заданных параметров, например, создавая экземпляры веб-сервера Nginx в рамках кластера. Ключевой особенностью данного оркестратора является его способность к автоматическому восстановлению после сбоев за счет использования нескольких узлов управления. Благодаря простоте настройки и управления Docker Swarm становится особенно привлекательным для малого и среднего бизнеса, в котором сложность экосистемы не требует специализированных решений.

Apache Mesos представляет платформу оркестрации вычислительных ресурсов в распределенной среде, который обеспечивает гибкое управление ресурсами, позволяя эффективно осуществлять оптимальное распределение ресурсов внутри кластера. Все запущенные задачи в Apache Mesos выполняются в контейнерах, причем контейнеры могут быть запущены путем загрузки образов из репозитория Docker, например с Docker Hub. Архитектура Mesos позволяет использовать несколько главных узлов, ориентированных на управление рабочих узлов, а также узлы-агенты, выполняющие развертывание образов Docker. Для организации главных узлов применяется Zookeeper, который выполняет функции управления кластером, аналогично планировщикам Hadoop или MPI. Apache Mesos широко применяется в таких крупных компаниях, как Twitter, Netflix, Airbnb и Apple, которые используют его в качестве инструмента для эффективного управления и масштабирования вычислительных ресурсов.

Результаты и обсуждение

1. Сравнение функциональных характеристик инструментов оркестрации контейнеров. Сравнение инструментов оркестрации контейнеров проводится на основании ключевых параметров системы, а информация о характеристиках и их особенностях представлены в таблице 3.

Docker Compose характеризуется простотой в использовании и настройке, но его функциональные возможности ориентированы на разработку и тестирование приложений, а не на эксплуатацию в производственных средах. В случае необходимости дополнительных функций для пользователей, им следует перейти на Docker Swarm, который может использовать текущую конфигурацию Docker Compose. Однако две среды оркестровки Docker Compose и

Docker Swarm работают исключительно с контейнерами Docker, а для использования альтернативных контейнерных платформ требуется применение других оркестраторов.

Таблица 3 – Функциональное сравнение оркестраторов контейнеров и технологий виртуализации

Описание	Compose	Swarm	Kubernetes	OpenShift	Mesos	Fleet	OpenVZ	Rancher	Nomad
Поддерживаемые контейнеры	Docker	Docker	Docker, rkt, CRI-O, OCI, Windows Containers	Docker, CRI-O	Docker, rkt	Docker, rkt	LXC	Docker, CRI-O	Docker, CRI-O
Масштабирование	Есть	Есть	Есть	Есть	Есть	Есть	Есть	Есть	Есть
Отказоустойчивость	Нет	Есть	Есть	Есть	Есть	Есть	Есть	Есть	Есть
Сетевые плагины	Нет	Есть	Есть	Есть	Есть	Нет	Есть	Есть	Есть
Интеграция с облачными платформами	Ограниченный	Ограниченный	Есть	Есть	Есть	Есть	Ограниченный	Есть	Есть
Интеграция инструментов развертывания	Нет	Swarm CLI	Helm, Apollo, Kube-spray	Helm, Open-shift Pipelines	Marathon, Terraform	Systemd	LXC templates	Helm, Fleet	Terraform, Packer
Интеграция инструментов мониторинга	Нет	Prometheus	cAdvisor	Prometheus, Thanos	Prometheus	Fleetctl	LXC Monitor	Prometheus	Nomad UI
Интеграция инструментов безопасности	Нет	Docker Security	Twistlock, App-Armor, Falco, Aqua	RedHat Advanced Cluster Security	Vault, RBAC	Нет		CIS Benchmarks	Sentinel

Развертывание контейнерного приложения на Kubernetes является довольно сложным для начинающих, оно требует настройки с помощью файлов конфигурации и не имеет простого веб-интерфейса. Реализация сети Kubernetes в локальной инфраструктуре требует установки внешних сетевых плагинов, таких как Flannel или Calico, а также поддерживает Weave Net, Cilium и Multus. Nomad и Rancher используют сетевые плагины Flannel и Calico, тогда как Docker Swarm использует встроенные сетевые плагины, такие как overlay network, macvlan и bridge. Apache Mesos является самым сложным для настройки кластера, поскольку требуются внешние фреймворки, такие как Marathon или Terraforms.

Несмотря на удобство использования Docker Compose в локальных компьютерах, его архитектура не включает встроенные средства безопасности, такие как шифрование данных и сетевую изоляцию, а также не интегрируется напрямую с облачными платформами. Docker Swarm не может автоматически перезапускать контейнеры в случае сбоя на уровне узла и не может динамически управлять ресурсами, а также существуют риски для TLS и конфиденциальной информации. Неправильная настройка управления доступом на основе ролей (RBAC) Kubernetes может привести к повышению привилегий и поставить под угрозу безопасность из-за уязвимостей в плагинах и внешних компонентах. Сложность системы OpenShift увеличивает вероятность ошибок конфигурации и сбоев защиты данных из-за неправильных настроек списка контроля доступа (ACL). В свою очередь, OpenVZ использует встроенные сетевые механизмы, он не поддерживает плагины CNI, такие как Calico или Weave, и зависит от механизмов безопасности хоста.

2. Производительность контейнеризированных приложений. Первый этап исследования был проведен на основании мониторинга процесса создания контейнерных приложений и измерения времени от запуска процесса до полной готовности приложения к работе. Для обеспечения точности измерений был рассмотрен анализ журналов каждого контейнера, фиксирующие инициализацию контейнера, загрузку образа приложения, конфигурацию среды и других ключевых этапов запуска. Время запуска контейнера определялось как интервал между началом его создания и полной готовностью приложения. Второй этап направлен на оценку загрузки процессора и использования оперативной памяти при работе инструментов оркестрации. В качестве инструмента сбора данных применялось Docker Desktop. Аналогичный анализ для Apache Mesos выполнен с использованием Mesos Web UI, тем самым позволяя в реальном времени отслеживать нагрузку и собирать данные.

Таблица 4 содержит результаты тестирования эксплуатационных характеристик инструментов оркестрации контейнеров Docker Swarm, Kubernetes и Apache Mesos в локальной инфраструктуре, а также развертывания приложений в популярных облачных системах Amazon Web Services, Azure и Google.

Таблица 4 – Сравнительный анализ эксплуатационных характеристик систем оркестрации контейнеров

Эксплуатационные характеристики	Docker Swarm (v24.0.5)	Kubernetes (v1.30.1)	Apache Mesos (1.11.0)	Amazon Web Services EKS	Azure AKS	Google GKE
Среднее время развертывания приложений	90–100 сек.	180–200 сек.	90–100 сек.	20–30 сек.	33–56 сек.	39–61 сек.
Среднее время запуска приложения	9,9 сек.	11,2 сек.	10,7 сек.	В зависимости от множеств факторов: пропускной способности сети и рабочей нагрузки	В зависимости от множеств факторов: пропускной способности сети и рабочей нагрузки	В зависимости от множеств факторов: пропускной способности сети и рабочей нагрузки
Загрузка ЦП	3,88 Мгц	5,03 МГц	4,71 МГц	150-200 сек. (Y-cruncher) 100000-120000 (количество событий (Sysbench))	350-400 сек. (Y-cruncher) 80000-100000 (количество событий (Sysbench))	250-300 сек. (Y-cruncher) 90000-100000 (количество событий (Sysbench))
Результаты теста производительности диска	В зависимости от ресурсов виртуальной машины	В зависимости от ресурсов виртуальной машины	В зависимости от ресурсов виртуальной машины	100-150 Мб/с (Bonnie++) 14-16 Мб/с (read) 10-12 Мб/с (write) (Sysbench)	0-50 Мб/с (Bonnie++) 4-6 Мб/с (read) 2-4 Мб/с (write) (Sysbench)	300-350 Мб/с (Bonnie++) 4-6 Мб/с (read) 2-4 Мб/с (write) (Sysbench)
Результаты теста производительности памяти	3,58 Гб	4,12 Гб	3,83 Гб	8-10 Гб (STREAM)	8-10 Гб (STREAM)	8-10 Гб (STREAM)
Пропускная способность сети	2027,36 запросов/сек	1420,7 запросов/сек	1824,65 запросов/сек	500-1000 Мб/с (Nuttcp)	500-1000 Мб/с (Nuttcp)	3500-4000 Мб/с (Nuttcp)
Стоимость управления кластером	бесплатно	бесплатно	бесплатно	0,10 долларов США в час	бесплатно	0,10 долларов США в час

Полученные результаты основаны на данных, полученных в ходе экспериментов по развертыванию и эксплуатации контейнерных технологий, а также на анализе научной литературы. Данные, относящиеся к облачным контейнерным сервисам, основаны на работе [20]. Исследование показало, что Amazon EKS подходит для высоконагруженных систем с требованиями к быстрому внедрению, тогда как Azure AKS и Google GKE подходят для средних и крупных проектов. Docker Swarm хорошо масштабируется благодаря простоте настройки на виртуальных машинах. GKE обеспечивает высокую пропускную способность сети, подходящую для интенсивного сетевого подключения. Представленная здесь информация показывает, что выбор инструмента/платформы оркестровки контейнеров должен основываться на конкретных требованиях проекта:

- Amazon EKS рекомендуется для экономии времени и быстрого внедрения;
- GKE целесообразно выбирать при высоких требованиях к сети и пропускной способности;
- Azure AKS предпочтителен для сложных вычислительных задач.
- Docker Swarm является лучшим выбором для небольших проектов, следует выбирать из-за минимальных затрат на управление, но обладает ограниченными возможностями по автоматическому управлению ресурсами;
- Kubernetes подходит для корпоративных и облачных систем, имеет более сложную конфигурацию и требует больше вычислительных ресурсов;
- Apache Mesos предпочтителен для крупных распределенных систем и требует интеграции с дополнительными инструментами.

Производительность любого программного приложения определяется совокупностью программных и аппаратных компонентов, на которые может влиять интенсивность рабочей нагрузки и поддерживаемых сервисов. При неправильном выборе рабочей нагрузки становится сложно достичь ожидаемого результата, решение этой проблемы можно найти путем проведения стресс-теста или нагрузочного теста и масштабирования ресурсов по вертикали или горизонтали. Важным аспектом считается влияние количества пользователей и объема запросов на стабильность системы, который, в свою очередь, особенно критичен в масштабируемых средах. Кроме того, по мере увеличения числа пользователей система может оказаться не в состоянии выделять ресурсы каждому пользователю, что приведет к задержкам в сети и увеличению времени отклика. Решением этой ситуации является поддержка кластеризации системы или микросервисной архитектуры, а также динамическое масштабирование ресурсов.

В рамках данного исследования для тестирования отправки HTTP-запросов была выбрана платформа Apache JMeter. Основной целью тестирования являлось определение предельной нагрузки, которую способны обработать Kubernetes, Docker Swarm и Apache Mesos. Базовый рабочий процесс JMeter включает этапы формирования запроса к целевому серверу, сбору и обработке статистических данных, а также генерации отчета, представляемых в виде графики, таблицы, деревьев или файловых журналов. Главным параметром тестирования является настройка количества потоков, моделирующих пользователей, отправляющих HTTP-запросы. В рамках тестирования были инициированы 2000 потоков (пользователей), каждый из которых выполнял 30 итераций с 10-секундным интервалом между запусками. Данное тестирование моделирует условия высокой нагрузки на систему, позволяя оценить ее поведение при интенсивном пользовательском трафике.

Представленный график (рисунок 4) демонстрирует результат четырех метрик времени отклика (в миллисекундах) для оркестрации контейнеров: Docker Swarm, Kubernetes, Apache Mesos. Показатели включают среднее, медианное, минимальное и максимальное время отклика, который позволяет оценить не только производительность, но и стабильность каждого решения.

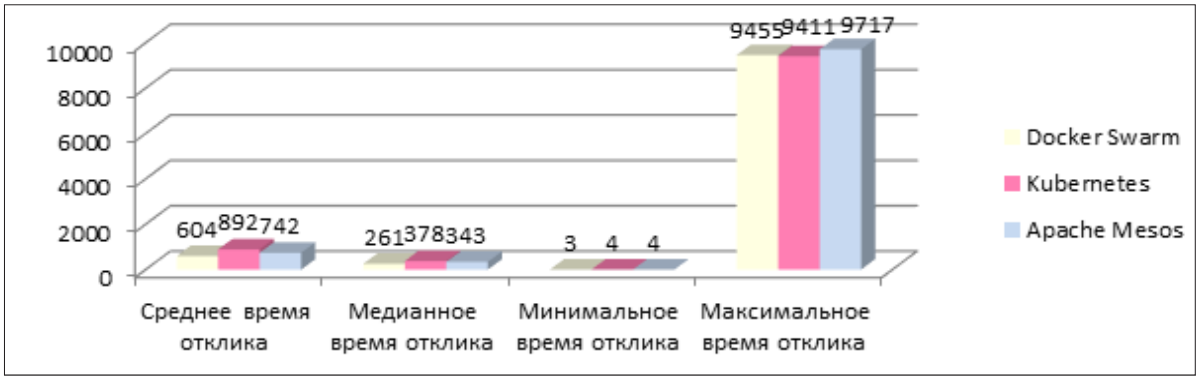

Рисунок 4 – Время отклика контейнерных оркестраторов

По всем параметрам Docker Swarm имеет наименьшее значение, свидетельствующее о его высокой эффективности в условиях интенсивной нагрузки. Если важна минимальная задержка, лучшим выбором является Docker Swarm. Kubernetes показывает высокие средние и медианные задержки, но при этом демонстрирует стабильные результаты по сравнению с Apache Mesos. Apache Mesos характеризуется наибольшим максимальным временем отклика, что может указывать на потенциальные проблемы с управлением нагрузкой.

В таблице 5 находятся обобщенные результаты анализа пропускной способности оркестраторов контейнеров, включая объем полученной и отправленной информации, а также уровень ошибок, позволяющий оценить эффективность и надежность каждого решения.

Таблица 5 – Анализ передачи данных и устойчивости к ошибкам

	Kubernetes	Docker Swarm	Apache Mesos
Пропускная способность сети (запросов/сек)	1420,7	2027,36	1824,65
Полученные данные (KB/s)	31299,83	43280,67	38532,74
Отправленные данные (KB/s)	161,66	226,43	170,75
Ошибки	0,41	1,41%	0,71%

Анализ представленных данных характеризует то, что наибольший объем передаваемой информации в расчете на единицу времени демонстрирует оркестратор Docker Swarm, достигая показателя 43 280,67 KB/s. Однако данный инструмент имеет наибольшую долю ошибок, свидетельствуя о потенциальных недостатках его механизма управления распределенными средами. Kubernetes показывает минимальный уровень ошибок (0,41%), при этом указывая на свою высокую устойчивость и надежность. Apache Mesos обеспечивает сбалансированное соотношение между объемом переданных данных и стабильностью системы, что, в свою очередь, делает его оптимальным выбором, когда требуется компромисс между производительностью и надежностью.

3. Проблемы реализации, развертывания, обеспечения безопасности и мониторинга контейнерных приложений в локальных инфраструктурах и облачных системах. На основе рассмотренных выше вопросов, связанных с компонентами оркестрации контейнеров, была создана таблица 6, в которой описаны проблемы внедрения, развертывания, безопасности и мониторинга в локальной и облачной инфраструктуре.

Таблица 6 – Проблемы реализации, развертывания, безопасности и мониторинга в локальной и облачной инфраструктуре

	Локальная инфраструктура	Облачная инфраструктура
Проблемы реализации	1. Низкая мощность сервера; 2. Настройка инструмента оркестрации, такого как Kubernetes, сложна; 3. Программное обеспечение сервера может быть несовместимо с контейнерами.	1. Процесс перехода к другому провайдеру сложен; 2. Трудно заранее планировать используемые ресурсы.
Проблемы развертывания	1. Развертывание контейнеров и создание их конфигураций требует времени и ресурсов; 2. Неспособность оперативного развертывания дополнительных серверов.	1. Конфиденциальные данные трудно защитить; 2. Рост трафика приводит к увеличению времени ожидания.
Проблемы обеспечения безопасности	1. Размещение контейнеров на одном хосте создает риск их взаимного влияния; 2. Задержка в установке обновлений безопасности приводит к возникновению уязвимостей.	1. Недостаточная защита API и конфиденциальной информации; 2. Необходимость обеспечения шифрования при хранении данных.
Проблемы мониторинга	Анализ системных журналов занимает много времени.	Большие объемы данных требуют дополнительной обработки.

Учет этих вопросов и принятие правильных решений повысит производительность и надежность контейнеризированных приложений.

Перспективными направлениями дальнейших исследований являются разработка стратегий масштабирования до сотен и тысяч контейнеров, изучение их производительности и влияния на стоимость для различных сценариев.

Заключение

В заключение стоит отметить, что выбор контейнерной технологии играет значительную роль в успешном развертывании и управлении приложениями в современной инфраструктуре информационных технологий. Постоянное развитие контейнерных технологий и расширение их функциональности предоставляют пользователям множество возможностей для разработки, внедрения и масштабирования современных приложений. В связи с этим критически необходимо выбирать наиболее подходящие инструменты, соответствующие требованиям проекта и особенностям среды применения.

Каждая из рассмотренных технологий имеет свои преимущества и недостатки, и понимание этих ситуаций является залогом успешного внедрения и принятия решений в проектах по локальной инфраструктуре и облачным вычислениям. Выбор инструмента или платформы оркестрации контейнеров должен основываться на конкретных требованиях проекта, таких как экономия времени, ускорение внедрения, снижение затрат на управление, высокие требования к сети, а также сложные вычислительные задачи.

Проведенный анализ показал, что Docker Swarm осуществляет быстрое развертывание, имеет низкое потребление центрального процессора и лучшую пропускную способность, он подходит для небольших проектов, поскольку не поддерживает автоматическое масштабирование и требует ручного управления нагрузкой. В свою очередь, данный инструмент обеспечивает наиболее быстрое развертывание контейнерных приложений и упрощает управление кластером.

Kubernetes обеспечивает высокую производительность памяти, предоставляет возможности автоматического масштабирования, подходит для корпоративных и облачных систем, имеет более сложную конфигурацию и требует больше вычислительных ресурсов. Следовательно, Amazon Elastic Kubernetes Service подходит для высоконагруженных систем с требованиями к быстрому внедрению, тогда как Azure Kubernetes Service и Google Kubernetes Engine подходят для средних и крупных проектов.

Архитектурные особенности Apache Mesos позволяют эффективно управлять масштабными кластерами за счет поддержки различных фреймворков и динамического распределения ресурсов, а также легко интегрируется с устаревшими системами. Данная платформа является предпочтительным решением для крупных корпоративных инфраструктур и распределенных систем.

ЛИТЕРАТУРА

1 Malviya, A., Dwivedi, R.K. A comparative analysis of container orchestration tools in cloud computing. 2022 9th International Conference on Computing for Sustainable Global Development (INDIACom) (IEEE, 2022), pp. 698–703. https://doi.org/10.23919/INDIACom54597.2022.9763171.

2 Kumar, R., Trivedi, M.C. Networking analysis and performance comparison of Kubernetes CNI plugins. Advances in Computer, Communication and Computational Sciences: Proceedings of IC4S 2019 (Springer Singapore, 2021), pp. 99–109.

3 Atlidakis, V., Godefroid, P., Polishchuk, M. Checking security properties of cloud service REST APIs. 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST) (IEEE, 2020), pp. 387–397.

4 Pankowski, A., Powroźnik, P. Comparison of application container orchestration platforms. Journal of Computer Sciences Institute, 29, 383–390 (2023). https://doi.org/10.35784/jcsi.3823.

5 Acharya J.N., Suthar A.C. Docker container orchestration management: A review. International Conference on Intelligent Vision and Computing. Cham: Springer International Publishing, 2021, pp. 140–153. https://doi.org/10.1007/978-3-030-97196-0_12.

6 Moravcik, M., Segec, P., Kontsek, M., Uramova, J., & Papan, J. Comparison of lxc and docker technologies. 2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA) (IEEE, 2020), pp. 481–486. https://doi.org/10.1109/ICETA51985.2020.9379212.

7 Putri, A.R., Munadi, R., Negara, R.M. Performance analysis of multi services on container Docker, LXC, and LXD. Bulletin of Electrical Engineering and Informatics, 9(5), 2008–2011. https://doi.org/10.11591/eei.v9i5.1953.

8 Koziolek, H., Eskandani, N. Lightweight kubernetes distributions: A performance comparison of microk8s, k3s, k0s, and microshift. Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering, 2023, pp. 17–29. https://doi.org/10.1145/3578244.3583737.

9 Mercl, L., Pavlik, J. The comparison of container orchestrators. Third International Congress on Information and Communication Technology: ICICT 2018, London (Springer Singapore, 2019), pp. 677–685. https://doi.org/10.1007/978-981-13-1165-9_62.

10 Sen, A., Madria, S. Analysis of a cloud migration framework for offline risk assessment of cloud service providers. Software: Practice and Experience, 50(6), 998–1021 (2020).

11 Liu, Q. et al. PQA-Net: Deep no reference point cloud quality assessment via multi-view projection. IEEE transactions on circuits and systems for video technology, 31(12), 4645–4660 (2021).

12 Liu, Y. et al. Point cloud quality assessment: Dataset construction and learning-based no-reference metric. ACM Transactions on Multimedia Computing, Communications and Applications, 19(2s), 1–26 (2023).

13 Wang, S. et al. Non-local geometry and color gradient aggregation graph model for no-reference point cloud quality assessment. Proceedings of the 31st ACM International Conference on Multimedia, 2023, pp. 6803–6810.

14 Liu, P., Guitart, J. Performance comparison of multi-container deployment schemes for HPC workloads: an empirical study. The Journal of Supercomputing, 77(6), 6273–6312 (2021). https://doi.org/10.1007/s11227-020-03518-1.

15 Abraham, S., Paul, A.K., Khan, R.I.S., & Butt, A.R. On the use of containers in high performance computing environments. 2020 IEEE 13th International Conference on Cloud Computing (CLOUD) (IEEE, 2020), pp. 284–293. https://doi.org/10.1109/CLOUD49709.2020.00048.

16 Wong A.Y. et al. Threat modeling and security analysis of containers: A survey. arXiv preprint arXiv:2111.11475, 2021.

17 Parast F.K. et al. Cloud computing security: A survey of service-based models. Computers & Security, 114, 102580 (2022).

18 Zhang C. et al. Interval-valued intuitionistic uncertain linguistic cloud petri net and its application to risk assessment for subway fire accident. IEEE transactions on automation science and engineering, 19(1), 163–177 (2020).

19 Sen, A., Madria, S. Application design phase risk assessment framework using cloud security domains. Journal of Information Security and Applications, 55, 102617 (2020).

20 Ferreira, A.P., Sinnott, R. A performance evaluation of containers running on managed kubernetes services. 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom) (IEEE, 2019), pp. 199–208.

[1]*Кусепова Л.Т.,
аға оқытушы, ORCID ID: 0000-0002-7972-3169,
*e-mail: kussepova_lt@enu.kz
[1]Кусепова Г.Т.,
аға оқытушы, PhD, ORCID ID: 0000-0001-9556-8763,
e-mail: kussepova_gt_2@enu.kz

[1]Л.Н. Гумилев атындағы Еуразия ұлттық университеті,
Астана қ., Қазақстан

ЖЕРГІЛІКТІ ЖӘНЕ БҰЛТТЫ ИНФРАҚҰРЫЛЫМДА КОНТЕЙНЕРЛЕРДІ ОРКЕСТРЛЕУ ҚҰРАЛДАРЫН ТАЛДАУ ЖӘНЕ САЛЫСТЫРУ

Аңдатпа

Ауқымды бұлтты орталарда виртуализация тек қосымшаларды жүзеге асыруда және жұмыс жүктемелерін үлестіруде ғана емес, сондай-ақ платформаның сервистері мен ресурстарын тиімді басқаруда да маңызды рөл атқарады. Виртуалды машиналар сияқты дәстүрлі тәсілдермен қатар, контейнерлеу технологиясын қолдану заманауи бұлтты инфрақұрылымдарда барған сайын кеңінен таралуда. Контейнерлер виртуалды машиналарға қарағанда виртуализацияның қарапайым әрі жылдам тәсілін қамтамасыз етеді, оқшауланған ортада қосымшаларды жүзеге асыру және басқару процесін жеделдетеді, осылайша ресурстарды тиімді пайдалануға мүмкіндік береді. Бағдарламалық қамтамасыз етуді әзірлеушілер үшін контейнерлер қосымшалар компоненттерін тестілеу мен микросервистерге негізделген архитектураларды құру үшін қолайлы шешім болып табылады. Нәтижесінде контейнерлеу заманауи бұлтты орталар мен бизнес қажеттіліктері үшін таңдаулы шешімге айналуда. Сондықтан бұл жұмыс заманауи және қазіргі уақытта кеңінен қолданылатын контейнерлік технологияларды зерттеуге бағытталған. Мақалада контейнерлер жұмысының тиімділігі, масштабтау мүмкіндіктері, ресурстарды басқару сияқты параметрлер негізінде заманауи контейнерлік технологиялар мен контейнерлерді оркестрлеу құралдарына талдау жасалды. Docker Swarm, Kubernetes және Apache Mesos оркестраторларына, сондай-ақ бірқатар контейнерлік шешімдерге назар аударылып, әрбір технологияның артықшылықтары мен кемшіліктері зерттелді. Бұл жұмыс ақпараттық технологиялар саласының мамандары үшін қосымшаларды әзірлеу және олардың бұлттық стратегияларын жүзеге асыру барысында ең қолайлы құралды таңдауда пайдалы болады.

Тірек сөздер: контейнерлік технологиялар, контейнерлерді оркестрлеу, бұлтты есептеулер, салыстырмалы талдау.

[1]*Kussepova L.T.,
Senior lecturer, ORCID ID: 0000-0002-7972-3169,
e-mail: kussepova_lt@enu.kz
[1]Kussepova G.T.,
Senior lecturer, PhD, ORCID ID: 0000-0001-9556-8763,
e-mail: kussepova_gt_2@enu.kz

[1]L.N. Gumilyov Eurasian National University,
Astana, Kazakhstan

ANALYSIS AND COMPARISON OF CONTAINER ORCHESTRATION TOOLS IN ON-PREMISES AND CLOUD INFRASTRUCTURE

Abstract

In large-scale cloud environments, virtualization plays a key role not only in application deployment and workload distribution but also in the effective management of platform services and resources. The use of containerization technology, alongside traditional approaches such as virtual machines, is becoming increasingly popular in modern cloud infrastructures. Containers provide a simpler and faster method of virtualization compared to virtual machines, accelerating the deployment and management of applications in isolated environments and thereby enabling more efficient resource utilization. For developers, containers offer a convenient solution for testing application components and building microservice-based architectures. As a result, containerization is becoming the preferred solution for modern cloud environments and current business needs. Therefore, this study is dedicated to examining modern and widely recognized container technologies. The article analyzes contemporary containerization and orchestration technologies using parameters such as container performance, scalability, and resource management. Particular attention is given to orchestrators such as Docker Swarm, Kubernetes, and Apache Mesos, as well as a range of container-based solutions, with an evaluation of the advantages and disadvantages of each technology. This work may be useful for IT specialists in selecting the most appropriate tools for application development and the implementation of their cloud strategies.

Keywords: container technologies, container orchestration, cloud computing, comparative analysis.

Received: March 12, 2025; revised: December 10, 2025; accepted: January 13, 2026.