Министерство образования и науки Российской федерации

Алтайский государственный технический университет

им. И.И. Ползунова

Кафедра «Информационные системы в экономике»

Отчет

о курсовой работе

по теме: «Контейнеризация приложений: Docker, Kubernetes и их роль в современном ПО»

предмет: Современные технологии разработки программного обеспечения

Выполнил Студент группы 8ПИЭ-41

Хартов А.Е.

Проверил Степанов А.С.

Барнаул 2024

Содержание.

Цели и задачи
Цель: Анализ подходов к контейнеризации приложений с использованием Docker и Kubernetes, а также исследование их роли и влияния на разработку, развертывание и эксплуатацию современного программного обеспечения.

Задачи:

1. **Исследование теоретической базы:**

- Изучить основные концепции контейнеризации и различия между виртуализацией и контейнеризацией.
- Рассмотреть архитектуру Docker и Kubernetes, их основные компоненты и принципы работы.
- Изучить преимущества использования Docker и Kubernetes в разработке программного обеспечения.

2. **Обзор современных тенденций:**

- Анализ распространенности и областей применения Docker и Kubernetes.
- Рассмотреть роль контейнеризации в DevOps-практиках и CI/CD-процессах.
- Изучить примеры успешного использования Docker и Kubernetes в реальных проектах.
- Экономические аспекты

3. **Формирование выводов:**

- Проанализировать полученные результаты и сделать выводы о влиянии контейнеризации на разработку современного программного обеспечения.
- Сформулировать рекомендации по применению Docker и Kubernetes в зависимости от типов проектов и задач.

1. Исследование теоретической базы

1.1 Основные концепции контейнеризации

Контейнеризация — это подход к изоляции приложений и их окружения в компактных и автономных контейнерах. Этот подход основан на использовании общей операционной системы (ОС), что делает контейнеры легковесными и высокопроизводительными по сравнению с виртуальными машинами. Контейнеризация позволяет запускать несколько приложений на одном хосте с минимальными затратами ресурсов и при этом обеспечивать изоляцию между ними.

Основные принципы контейнеризации

1. Изоляция:
   Контейнеры предоставляют каждому приложению отдельную среду выполнения, включая файловую систему, сети и переменные окружения. Это предотвращает конфликты между приложениями и их зависимостями, что особенно важно для разработки и тестирования.

2. Легковесность:
   Контейнеры используют общее ядро операционной системы, что снижает затраты на ресурсы. В отличие от виртуальных машин, контейнеры не требуют полного образа операционной системы, что значительно уменьшает их размер.

3. Портативность:
   Контейнеры упаковывают приложение и все его зависимости в единую единицу, которая может быть запущена на любом сервере с установленным контейнерным движком (например, Docker Engine). Это позволяет легко переносить приложения между различными окружениями: локальной машиной разработчика, тестовым стендом или продакшен-сервером.

4. Управление зависимостями:
   Все библиотеки, конфигурационные файлы и зависимости упаковываются в контейнер. Это гарантирует, что приложение будет работать одинаково в любом окружении, независимо от установленных на хосте пакетов.

5. Масштабируемость и гибкость:
   Контейнеры могут быть легко масштабированы горизонтально (увеличение количества экземпляров) или вертикально (увеличение ресурсов для отдельных контейнеров). Оркестрация (например, с помощью Kubernetes) позволяет автоматизировать процесс масштабирования.

Основные термины и компоненты контейнеризации

1. Образы (Images):
   Контейнерные образы — это шаблоны, которые включают приложение, его зависимости и инструкции для запуска. Образы неизменяемы, что делает их удобными для создания предсказуемых сред. Docker Images, например, хранятся в реестрах, таких как Docker Hub.

2. Контейнеры:
   Контейнер — это запущенный экземпляр образа. Контейнеры изолированы друг от друга и от хост-системы, но могут взаимодействовать через определённые сетевые интерфейсы.

3. Контейнерный движок:
   Программное обеспечение, которое управляет контейнерами на уровне операционной системы. Docker Engine — один из самых популярных контейнерных движков.

4. Реестры (Registries):
   Это хранилища для контейнерных образов. Реестры бывают публичными (Docker Hub, Google Container Registry) и приватными (локальные или корпоративные).

5. Оркестрация:
   Это процесс автоматизации управления контейнерами в масштабах кластера. Kubernetes — наиболее популярная система оркестрации, обеспечивающая автоматическое масштабирование, самовосстановление и управление контейнерами.

Особенности и отличия от традиционных подходов

Контейнеризация решает множество проблем, которые были характерны для традиционных методов развертывания:

- Проблема несовместимости окружений:
  Контейнеризация устраняет ошибки, возникающие из-за различий между средами разработки, тестирования и продакшена. Фраза "на моей машине работает" больше не актуальна.

- Ускорение процессов:
  Развертывание контейнеров происходит в считанные секунды, так как они не требуют полной загрузки операционной системы, как виртуальные машины.

- Упрощённое управление:
  Благодаря инструментам оркестрации, как Kubernetes, контейнеры можно легко развертывать, обновлять и масштабировать.

Преимущества контейнеризации

1. Эффективное использование ресурсов:
   Контейнеры делят ресурсы хост-системы, что позволяет экономить вычислительные мощности и снижать расходы.

2. Быстрота и удобство развертывания:
   Контейнеры запускаются за секунды, что ускоряет разработку, тестирование и выпуск новых версий приложений.

3. Изоляция и безопасность:
   Каждый контейнер изолирован от других, что снижает риск вмешательства или конфликта между приложениями.

4. Упрощение DevOps-процессов:
   Контейнеризация интегрируется с CI/CD инструментами, облегчая автоматизацию разработки, тестирования и развертывания приложений.

1.2 Различия между виртуализацией и контейнеризацией
Виртуализация и контейнеризация — это две технологии, используемые для изоляции и управления вычислительными ресурсами. Несмотря на схожие цели, их подходы, архитектура и области применения значительно различаются. Рассмотрим их основные различия.

1. Архитектурный подход

- Виртуализация:
    - Виртуализация создаёт несколько виртуальных машин (Virtual Machines, VM) на одном физическом сервере с использованием гипервизора.
    - Каждая ВМ включает операционную систему (гостевую ОС), приложения и необходимые зависимости.
    - Гипервизор, такой как VMware, Hyper-V или VirtualBox, управляет распределением ресурсов между ВМ и обеспечивает их изоляцию.
- Контейнеризация:
    - Контейнеризация разделяет ресурсы одной операционной системы между изолированными контейнерами.
    - Контейнеры используют общее ядро хостовой ОС и включают только приложения с их зависимостями.
    - Движки контейнеров (например, Docker) управляют запуском, остановкой и изоляцией контейнеров.

2. Изоляция

- Виртуализация:
  Полная изоляция обеспечивается на уровне гипервизора, так как каждая ВМ работает с собственной копией операционной системы. Изоляция очень надёжна, но создаёт значительные накладные расходы на ресурсы.

- Контейнеризация:
  Изоляция реализована на уровне файловой системы и процессов внутри одного ядра операционной системы. Это даёт легковесность и скорость, но изоляция менее строгая по сравнению с виртуализацией.

3. Использование ресурсов

- Виртуализация:
  Каждая виртуальная машина требует значительных ресурсов (память, процессор, хранилище) для выполнения гостевой ОС, приложений и гипервизора. Это приводит к более высокой ресурсоёмкости.

- Контейнеризация:
  Контейнеры используют меньше ресурсов, так как не требуется гостевая

ОС. Это позволяет запускать больше контейнеров по сравнению с количеством ВМ на том же оборудовании.

4. Размер и скорость запуска

- Виртуализация:
  Виртуальные машины значительно крупнее, так как содержат образ ОС, обычно занимающий несколько гигабайтов. Запуск ВМ может занимать минуты, так как требуется загрузка операционной системы.

- Контейнеризация:
  Контейнеры меньше по размеру, часто занимая сотни мегабайтов или меньше. Запуск контейнера происходит за считаные секунды, так как нет необходимости загружать ОС.

5. Совместимость и портативность

- Виртуализация:
  Виртуальные машины менее портативны, так как их перенос требует совместимости гипервизоров и настройки среды для каждой ВМ.

- Контейнеризация:
  Контейнеры крайне портативны. Они могут быть запущены на любом сервере с установленным контейнерным движком, независимо от платформы (локальный сервер, облако, кластер).

6. Масштабируемость

- Виртуализация:
  Масштабирование виртуальных машин требует больше ресурсов и времени, так как каждая новая ВМ требует запуска отдельной копии ОС.

- Контейнеризация:
  Контейнеры легко масштабируются за счёт их легковесности. Системы оркестрации, такие как Kubernetes, позволяют автоматизировать масштабирование контейнеров в реальном времени.

7. Обновление и управление

- Виртуализация:
  Обновления требуют управления каждой ВМ отдельно, включая обновление гостевой ОС. Это увеличивает сложность администрирования.

- Контейнеризация:
  Обновления проще, так как изменения можно внести в образ контейнера и заново развернуть его без остановки всей системы.

8. Примеры использования

- Виртуализация:
  - Подходит для запуска приложений с разными операционными системами на одном сервере.
  - Используется для управления большими монолитными приложениями или в случаях, где требуется строгая изоляция (например, в банках).
  - Примеры: VMware, VirtualBox, Hyper-V.
- Контейнеризация:
  - Идеальна для микросервисной архитектуры и приложений, которые должны работать в распределённых системах.
  - Широко применяется в DevOps-практиках и CI/CD процессах.
  - Примеры: Docker, Kubernetes, Podman.

9. Безопасность

- Виртуализация:
  Каждая виртуальная машина полностью изолирована на уровне гипервизора, что минимизирует риски вмешательства одной ВМ в другую.

- Контейнеризация:
  Контейнеры изолированы на уровне процессов и файловой системы, но при эксплуатации общего ядра ОС существует риск, что уязвимость ядра повлияет на безопасность всех контейнеров.

10. Стоимость

- Виртуализация:
  Часто требует более мощного оборудования и лицензирования гипервизоров, что увеличивает общую стоимость эксплуатации.

- Контейнеризация:
  Обладает более низкой стоимостью, так как позволяет эффективнее использовать ресурсы и не требует дополнительных лицензий для хостовой ОС.

1.3 Архитектура Docker и Kubernetes

Docker

Docker — это платформа для контейнеризации, которая автоматизирует создание, развертывание и управление контейнерами. Архитектура Docker включает три ключевых компонента:

1. Docker Engine — серверная часть, отвечающая за управление контейнерами.
2. Docker Images — образы, содержащие приложения и их зависимости.
3. Docker Hub — репозиторий для хранения и распространения образов.

Основные преимущества Docker:

- Быстрое развертывание приложений.
- Поддержка микросервисной архитектуры.
- Простота управления зависимостями.

Kubernetes

Kubernetes (K8s) — это система оркестрации контейнеров, разработанная для автоматизации развертывания, масштабирования и управления контейнеризованными приложениями. Архитектура Kubernetes состоит из следующих компонентов:

1. Master Node — центральный узел, управляющий кластером.
2. Worker Nodes — узлы, выполняющие контейнеры (Pods).
3. Pods — минимальная единица Kubernetes, представляющая собой группу контейнеров.
4. Control Plane — набор компонентов (API Server, Scheduler, Controller Manager), отвечающих за управление кластером.

Принципы работы Kubernetes:

- Оркестрация контейнеров в кластерах.
- Автоматическое масштабирование приложений.
- Самовосстановление приложений при сбоях.

1.4 Преимущества использования Docker и Kubernetes в разработке ПО

1. **Портативность:**
   Контейнеры работают одинаково на любом сервере, будь то локальная машина или облако.

2. **Масштабируемость:**
   Kubernetes позволяет легко масштабировать приложения в зависимости от нагрузки.

3. **Эффективность:**
   Контейнеры потребляют меньше ресурсов по сравнению с виртуальными машинами, что снижает затраты на инфраструктуру.

4. **Автоматизация:**
   Docker и Kubernetes упрощают процессы развертывания, обновления и управления приложениями.

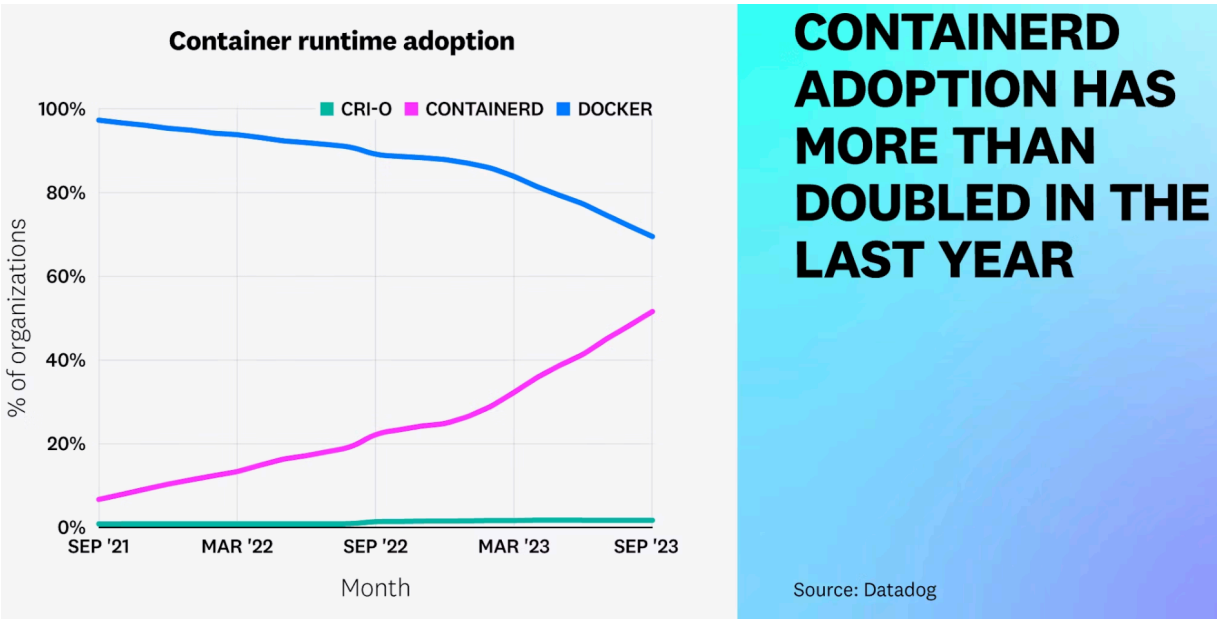
5. **Поддержка микросервисов:**
   Контейнеризация идеально подходит для микросервисной архитектуры, так как позволяет изолировать и управлять каждым сервисом независимо.

6. **Быстрое восстановление:**
   Kubernetes автоматически перезапускает упавшие контейнеры, обеспечивая высокую доступность приложений.

2. Обзор современных тенденций

2.1 Анализ распространённости и областей применения Docker и Kubernetes

На сегодняшний день контейнеризация с использованием Docker и Kubernetes стала стандартом в разработке и эксплуатации программного обеспечения. Обе технологии нашли широкое применение благодаря своим преимуществам в области производительности, масштабируемости и гибкости.



1. Распространённость Docker и Kubernetes:

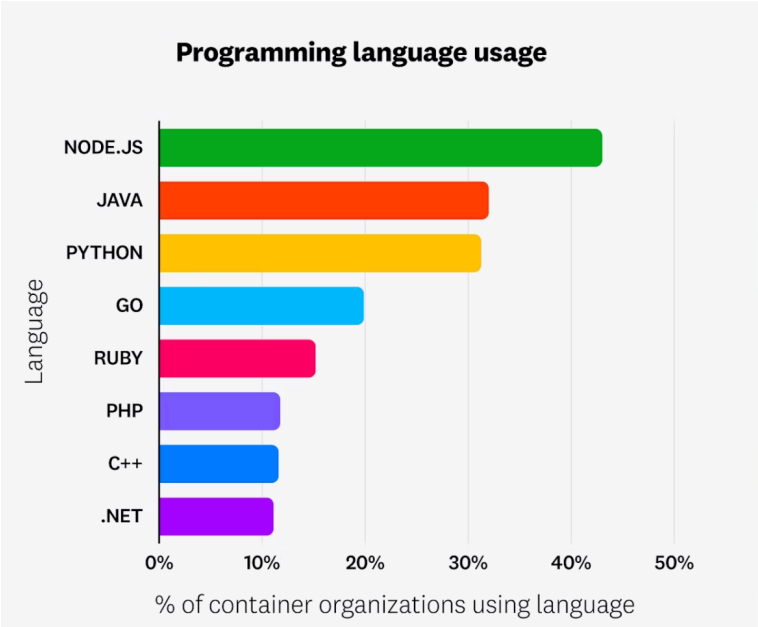
- Docker стал основным инструментом для создания и управления контейнерами. Согласно исследованиям аналитических компаний, таких как Gartner и IDC, более 70% организаций, занимающихся разработкой ПО, используют Docker.
- Kubernetes, как платформа оркестрации контейнеров, также завоевал популярность. На рынке оркестрации Kubernetes занимает около 85%, оставив позади такие решения, как Docker Swarm и Apache Mesos.

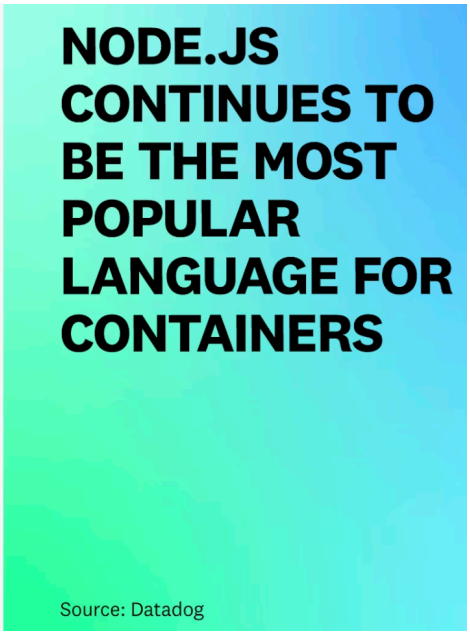
2. Области применения:

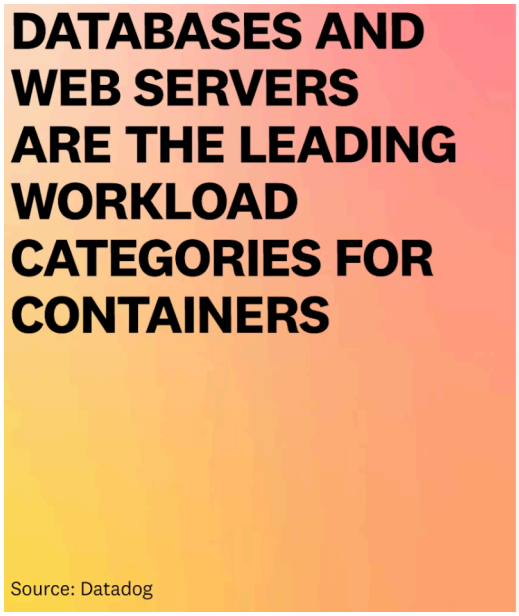
Docker и Kubernetes применяются в различных сферах, включая:

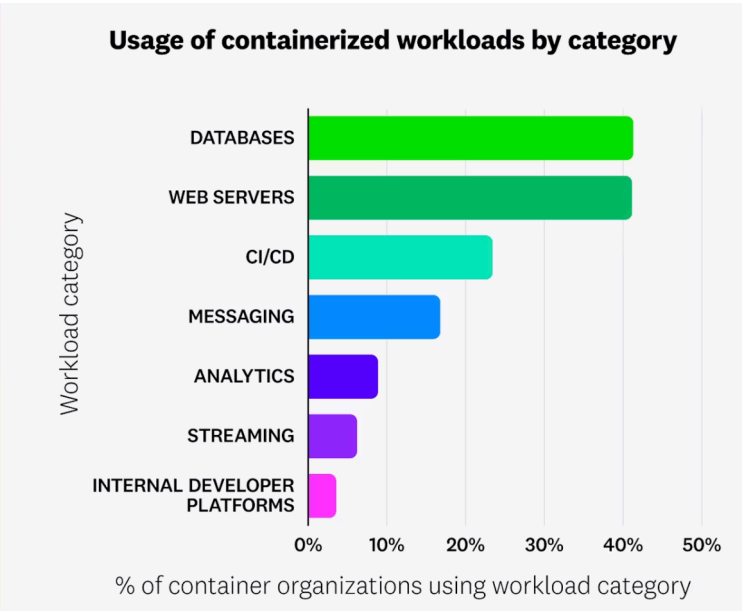
- **Разработка ПО:** Упрощение настройки окружения для разработки, обеспечение единства среды разработки, тестирования и продакшена.
- **Микросервисная архитектура:** Контейнеризация позволяет разрабатывать и развёртывать микросервисы независимо друг от друга.
- **Большие данные и искусственный интеллект:** Развёртывание контейнеров для обработки данных, машинного обучения и анализа данных.

- **Облачные вычисления:** Поддержка контейнеров во всех крупных облачных платформах (AWS, Google Cloud, Microsoft Azure).
- **Электронная коммерция:** Обеспечение масштабируемости и высокой доступности для платформ интернет-магазинов.









2.2 Роль контейнеризации в DevOps-практиках и CI/CD-процессах
Контейнеризация играет ключевую роль в DevOps, объединяя процессы разработки (Dev) и эксплуатации (Ops) через автоматизацию и стандартизацию рабочих процессов.

1. Контейнеризация и DevOps:
Docker и Kubernetes значительно упрощают работу DevOps-инженеров:

- **Автоматизация развертывания:** Возможность развёртывать приложения с помощью контейнеров без ручной настройки окружения.
- **Единое окружение:** Контейнеры гарантируют, что приложение работает одинаково в любой среде, исключая конфликты между разработкой и продакшеном.
- **Масштабируемость:** Kubernetes автоматизирует масштабирование в зависимости от нагрузки, обеспечивая стабильную работу системы.

2. Контейнеризация в CI/CD:
Контейнеры интегрируются с системами непрерывной интеграции и доставки (Continuous Integration/Continuous Delivery, CI/CD):

- **Быстрое тестирование:** Тестовые контейнеры можно быстро создавать, запускать и уничтожать после проверки.
- **Изоляция процессов:** Каждый этап CI/CD (сборка, тестирование, развертывание) может выполняться в отдельных контейнерах, что повышает надёжность процесса.
- **Автоматизация обновлений:** Оркестрация с помощью Kubernetes позволяет обновлять приложения без остановки сервиса (Rolling Updates).

Популярные CI/CD инструменты, такие как Jenkins, GitLab CI, CircleCI, предлагают готовую интеграцию с Docker и Kubernetes, что делает их использование ещё проще.

2.3 Экономические аспекты

1 **Снижение затрат на оборудование**

- **Сравнение:** Контейнеризация позволяет запускать больше приложений на одном сервере благодаря уменьшению накладных расходов (нет необходимости запускать гостевые ОС, как в случае виртуализации).
- **Экономия:** Оптимизация использования ресурсов позволяет снизить потребность в дополнительном оборудовании и уменьшить расходы на инфраструктуру.

2 **Сокращение времени разработки и развертывания**

- **Преимущества:** Быстрое создание контейнеров, идентичных в тестовой и продакшен-средах, сокращает цикл разработки и развертывания.
- **Влияние:** Это позволяет компании быстрее выпускать новые версии продуктов, повышая конкурентоспособность.

3 **Снижение эксплуатационных затрат**

- **Фактор:** Kubernetes автоматизирует управление кластером (например, распределение нагрузки, обновление приложений, масштабирование), что уменьшает затраты на поддержку инфраструктуры.
- **Результат:** Уменьшается потребность в ручной работе администраторов и вероятность человеческих ошибок.

4 **Экономия на лицензиях**

- **Сравнение:** Использование Docker и Kubernetes, являющихся open-source решениями, снижает затраты на программное обеспечение по сравнению с проприетарными аналогами.

2.4 Примеры успешного использования Docker и Kubernetes

1. Netflix:
Netflix использует Docker и Kubernetes для управления своей микросервисной архитектурой, состоящей из тысяч отдельных сервисов. Эти технологии помогают Netflix автоматически масштабировать приложения в зависимости от количества пользователей и нагрузки.

2. Spotify:
Spotify применяет контейнеризацию для разработки и развертывания своих сервисов. Docker используется для тестирования новых функций, а Kubernetes обеспечивает автоматическое управление контейнерами в продакшен-среде.

3. Pinterest:
Pinterest развернул Kubernetes для управления миллиардами изображений и постов, предоставляя высокую доступность и стабильность. Kubernetes позволяет масштабировать систему при росте числа пользователей, одновременно снижая затраты на инфраструктуру.

4. eBay:
eBay применяет контейнеризацию для ускорения разработки и повышения производительности. С помощью Docker и Kubernetes они смогли автоматизировать управление серверами и упростить CI/CD процессы.

5. Cloud-платформы:
Крупные облачные провайдеры, такие как Google Cloud, AWS и Microsoft Azure, активно используют Docker и Kubernetes для предоставления контейнерных решений в своих сервисах. Например, Google Kubernetes Engine (GKE) позволяет пользователям быстро запускать и управлять кластерами Kubernetes.

Docker и Kubernetes стали неотъемлемой частью современных подходов к разработке и эксплуатации программного обеспечения. Они используются во многих сферах, от облачных вычислений до электронных коммерческих платформ, и играют ключевую роль в автоматизации процессов DevOps и CI/CD. Реальные примеры из практики крупных компаний, таких как Netflix и Spotify, показывают, что использование этих технологий помогает ускорить разработку, повысить стабильность систем и снизить затраты на инфраструктуру.

3. Формирование выводов

3.1 Анализ влияния контейнеризации на разработку современного программного обеспечения

Контейнеризация внесла существенные изменения в процесс разработки и эксплуатации программного обеспечения, став важным компонентом современных подходов, таких как DevOps и CI/CD. На основе изученных материалов и примеров из практики можно выделить несколько ключевых влияний:

1. **Ускорение разработки и развертывания:**
   Контейнеры позволяют быстро создавать изолированные среды разработки, тестирования и продакшена. Это уменьшает временные затраты на настройку инфраструктуры и снижает вероятность ошибок, связанных с несовместимостью окружений.

2. **Повышение гибкости и масштабируемости:**
   Docker и Kubernetes позволяют масштабировать приложения горизонтально и вертикально, автоматически адаптируя систему к изменению нагрузки. Это особенно важно для проектов с высоким уровнем пользовательской активности.

3. **Снижение эксплуатационных затрат:**
   За счёт эффективного использования ресурсов и унифицированного подхода к развёртыванию контейнеризация помогает сократить расходы на оборудование и обслуживание.

4. **Улучшение управления зависимостями:**
   Благодаря упаковке приложения вместе с его зависимостями, контейнеризация устраняет конфликты между версиями библиотек и позволяет запускать приложения одинаково в любых средах.

5. **Поддержка микросервисной архитектуры:**
   Контейнеры идеально подходят для реализации микросервисов, так как позволяют изолировать каждый сервис и управлять его жизненным циклом независимо от других компонентов системы.

3.2 Рекомендации по применению Docker и Kubernetes в зависимости от типов проектов и задач

На основе анализа можно дать следующие рекомендации для эффективного применения Docker и Kubernetes:

1. **Небольшие проекты и стартапы:**
    - **Docker:**
      Подходит для разработки, тестирования и развертывания небольших приложений. Использование Docker позволит сократить время на настройку окружения и облегчить переносимость проекта.
    - **Kubernetes:**
      Может быть избыточным для небольших проектов. Альтернативой может стать Docker Compose или другие лёгкие инструменты управления контейнерами.
2. **Микросервисы и распределённые системы:**
    - **Docker:**
      Используется для упаковки каждого микросервиса в отдельный контейнер. Это упрощает управление зависимостями и обновление отдельных сервисов.
    - **Kubernetes:**
      Необходим для оркестрации большого числа микросервисов. Kubernetes автоматизирует задачи масштабирования, балансировки нагрузки и восстановления при сбоях.
3. **Проекты с высокой нагрузкой и требованиями к масштабируемости:**
    - **Docker:**
      Помогает создавать контейнеры с минимальными затратами на ресурсы. Использование Docker облегчает управление высоконагруженными сервисами.
    - **Kubernetes:**
      Рекомендуется для управления кластерами контейнеров. Возможности автоматического масштабирования и оркестрации делают Kubernetes идеальным для таких проектов.
4. **Тестовые и образовательные проекты:**
    - **Docker:**
      Отлично подходит для создания лабораторных сред, обучения и тестирования. Простота и доступность Docker делают его удобным для новичков.
    - **Kubernetes:**
      Используется для изучения систем оркестрации и работы с кластерными приложениями. Полезен в образовательных целях для понимания управления распределёнными системами.
5. **Корпоративные проекты и долгосрочная эксплуатация:**

- **Docker:**
  Подходит для унификации процессов разработки и упрощения CI/CD.
- **Kubernetes:**
  Является необходимым компонентом для управления большими проектами, где важны отказоустойчивость, гибкость и автоматизация.

3.3 Итоговые выводы
Контейнеризация с использованием Docker и Kubernetes значительно улучшает процессы разработки, тестирования и эксплуатации программного обеспечения. Эти технологии предоставляют инструменты для создания гибких, масштабируемых и надёжных систем, что особенно важно в условиях современной микросервисной и облачной архитектуры.

Тем не менее, выбор инструментов и подходов должен зависеть от конкретных требований проекта:

- Docker подходит для всех типов проектов благодаря своей универсальности и простоте.
- Kubernetes рекомендован для крупных проектов с высокими требованиями к масштабируемости и автоматизации.

Таким образом, внедрение контейнеризации позволяет оптимизировать рабочие процессы, повысить производительность и снизить эксплуатационные затраты, что делает её важной частью современной разработки программного обеспечения.

Список литературы:

1. 10 insights on real-world container use | Datadog https://www.datadoghq.com/container-report/
2. Какие известные компании используют Docker в production и для чего? https://habr.com/ru/companies/flant/articles/326784/
3. Контейнеризация приложений: преимущества Docker и Kubernetes https://moluch.ru/archive/550/120711/
4. Kubernetes https://ru.wikipedia.org/wiki/Kubernetes
5. Docker https://ru.wikipedia.org/wiki/Docker