ВВЕДЕНИЕ

На сегодняшний день в мире информационных технологий, который развивается с невероятной скоростью и который подталкивает нас к необходимости поддерживать высокие стандарты качества и возможности для масштабирования наших программ. В этом контексте особенно важно выделить идею о том, как можно упаковать приложения в так называемые контейнеры, отсюда мы получаем такое понятие контейнеризации приложений, которая позволяет разработчикам создавать и распространять свои программы в виде стандартизированных исполняемых единиц кода, содержащих все необходимые компоненты для работы.

Целью данного дипломного проекта является разработка и реализация системы управления контейнеризированными приложениями в распределенной среде. По сути, это проект о создании инструмента для управления процессами развертывания, масштабирования и обслуживания контейнеров, которая позволит повысить гибкость и эффективность их использования.

В процессе разработки были использованы самые современные технологии, касающиеся контейнеров и архитектуры микросервисов, такие как Docker и системы для их оркестровки. Весь проект был реализован на языке программирования Python, при этом активно использовались различные библиотеки для работы с Docker API.

Результатом работы является платформа, которая предоставляет лёгкий в использовании интерфейс для управления жизненным циклом контейнеров, включая возможности мониторинга и автоматического масштабирования под нагрузкой. Это решение значительно упрощает процесс управления контейнеризированными приложениями и повышает эффективность эксплуатации в условиях распределенной среды.

Применение разработанной системы может найти широкое применение в области облачных вычислений, DevOps и Continuous Integration / Continuous Deployment (CI/CD), способствуя ускорению разработки продуктов и их более быстрому выходу на рынок. Но не только технические аспекты делают проект важным, он также играет значимую социальную роль, делая разработку современного программного обеспечения доступнее и удобнее для большего числа специалистов.

В процессе работы над проектом было решено достаточно много задач, связанных с автоматическим масштабированием, сбором метрик работы контейнеров и оптимизацией распределения ресурсов между ними. Всё это в совокупности представляет значительный вклад в развитие инструментов управления программными приложениями в IT-секторе.

# Обзор литературы

## **Введение в контейнеризацию и системы управления**

## Контейнеризация представляет собой метод виртуализации на уровне операционной системы, который позволяет запускать и управлять приложениями и их зависимостями в процессах, изолированных от остальной системы. Этот подход позволяет легко развертывать и масштабировать приложения в любом окружении, поддерживая при этом их работоспособность и совместимость. Наглядная демонстрация такого подхода представлена на рисунке 1.1, которая иллюстрирует концепцию контейнеризации, показывая, как приложения, упакованные в контейнеры, изолируются друг от друга и одновременно работают на одной и той же хост-системе.

## Рисунок 1.1 – схема контейнеризации

## История контейнеризации начинается с появления изоляции пространства имен в ядре Linux, которая позволила разграничивать ресурсы системы между разными процессами. Основной вехой в развитии контейнеризации является появление Docker в 2013 году, которое было представлено Solomon Hykes. Docker предложил легкую и эффективную модель контейнеризации, в которой каждый контейнер работает независимо, обладает своим стеком технологий и может быть легко транспортирован между разными средами. Постепенно область управления контейнерами расширилась до необходимости их оркестрации — автоматизированного развертывания, масштабирования и управления контейнерными приложениями. Системы оркестрации, такие как Kubernetes, разработанный Google, стали ключевыми инструментами в управлении контейнеризированными приложениями в распределенных системах. Эти системы позволяют не только эффективно распределять нагрузку между контейнерами, но и обеспечивать их взаимодействие в рамках микросервисной архитектуры, упрощая разработку и поддержку сложных приложений. Такие инструменты как Docker и Kubernetes стали фундаментальными в мире современной разработки программного обеспечения и IT-инфраструктур. Внедрение контейнеризации позволяет ускорять процессы разработки, тестирования и развертывания, а также повышать надежность и удобство масштабирования приложений. Важным аспектом контейнеризации является также повышение уровня безопасности приложений, поскольку каждый контейнер является изолированным окружением, что снижает риски проникновения и распространения вредоносного кода. Научное сообщество активно развивает и изучает данные технологии, что находит отражение в большом количестве академических работ и публикаций, посвященных вопросам оптимизации, безопасности и управления ресурсами в контейнеризированных системах. В рамках этих исследований выявляются новые методы повышения эффективности работы контейнеров, разрабатываются рекомендации по оптимизации их развертывания и эксплуатации, а также изучаются вопросы взаимодействия контейнеризированных приложений с облачными сервисами и инфраструктурой. Таким образом, контейнеризация и системы управления контейнерами стали неотъемлемой частью современной ИТ-индустрии, предоставляя разработчикам и инженерам мощные инструменты для создания и поддержки сложных и масштабируемых приложений.

**1.2 Анализ существующих систем управления контейнерами**

**1.2.1 Kubernetes (K8s)**

Kubernetes, часто сокращенно называемый K8s, представляет собой инструментарий для автоматизации жизненного цикла контейнеризированных приложений в различных средах развертывания, будь то облачные сервисы, собственные (on-premise) данные центры или гибридные облака. Созданный и первоначально разработанный инженерами Google на основе их опыта эксплуатации контейнеров в масштабе и далее поддерживаемый Cloud Native Computing Foundation, Kubernetes стал де-факто стандартом в области оркестрации контейнеров.

Сердцем Kubernetes является автоматизация развертывания приложений. Ключевой особенностью является способность не только упростить развертывание приложений в контейнерах, но и предоставить инструменты для их непрерывного управления и масштабирования. Это достигается за счет использования мощной абстракции, которая описывает желаемое состояние приложений и их окружения, а затем автоматически изменяет реальное состояние, чтобы соответствовать заданному.

Система автоматического масштабирования Kubernetes позволяет приложениям реагировать на изменения нагрузки путем добавления или удаления ресурсов. Это значит, что при увеличении нагрузки на приложение, система может автоматически добавить дополнительные контейнеры для обработки дополнительных запросов, а затем также автоматически их убрать, когда нагрузка снижается.

Сервисное обнаружение и балансировка нагрузки в Kubernetes обеспечивают, чтобы входящий трафик распределялся между контейнерами, что улучшает отказоустойчивость и общую производительность приложений. Управление конфигурациями и секретами помогает обеспечить безопасность приложений и данных, позволяя централизованно управлять конфигурационными данными и паролями, сертификатами или токенами, необходимыми для доступа к различным внешним ресурсам.

Кроме того, Kubernetes предлагает возможности по самоисцелению приложений, такие как автоматический перезапуск контейнеров, которые вышли из строя, замену и репликацию контейнеров, а также перераспределение ресурсов в случае выхода из строя узла.

Концепция кластера в Kubernetes строится на концепци, являющегося набором узлов, которые обеспечивают запуск контейнерных приложений. В контексте дипломного проекта, кластер Kubernetes будет изучаться как центральный элемент системы управления, способный обеспечивать высокую доступность и отказоустойчивость развертываемых приложений.

Компоненты кластера: кластер делится на главные узлы (Master Nodes) и рабочие узлы (Worker Nodes). Главные узлы координируют кластер и принимают решения о запуске и распределении приложений, а рабочие узлы непосредственно выполняют задачи по запуску контейнеров.

К основным компонентам рабочего узла можно отнести: Кubelet - это агент, работающий на каждом рабочем узле, отвечающий за запуск, остановку и поддержание работоспособности контейнеров в соответствии с указаниями API сервера, Кube-Proxy поддерживает сетевые правила на узлах, позволяя сетевому взаимодействию между контейнерными подами, Container Runtime отвечает за запуск контейнеров, используя Docker, containerd, CRI-O или любой другой совместимый с CRI (Container Runtime Interface).

Поды и сервисы: контейнеры группируются в поды — основные развертываемые единицы в Kubernetes, которые могут содержать один или несколько тесно связанных контейнеров. Поды в свою очередь управляются сервисами, предоставляющими постоянные точки доступа к наборам работающих подов.

Касаемо архитектуры Kubernetes следует сказать следующее,  
работающий кластер Kubernetes включает в себя агента, запущенного на нодах (kubelet) и компоненты мастера (APIs, scheduler, etc), поверх решения с распределённым хранилищем. Приведённая схема на рисунке 1.2 показывает желаемое, в конечном итоге, состояние.

## 

## Рисунок 1.2 – схема состояний Kubernetes

Благодаря своим расширенным возможностям, Kubernetes стал важным компонентом в стратегиях DevOps и CI/CD, облегчая непрерывную интеграцию и непрерывное развертывание программного обеспечения. Это позволяет организациям быстрее и более надежно выводить на рынок новые версии приложений, улучшая тем самым их конкурентоспособность.

**1.2.2 Docker Swarm**

Контейнеризация является одной из ключевых технологий в современной разработке программного обеспечения, позволяющей изолировать приложения и их зависимости в контейнеры. Эти контейнеры включают в себя всё необходимое для функционирования приложения, от системных библиотек до специфических настроек. Отличительная черта контейнеров — их изолированность, обеспечивающая стабильную работу приложений, независимо от внешних факторов.

Менеджмент контейнеров становится особенно сложным, когда их количество возрастает. Решением данной проблемы служат системы оркестрации, среди которых особняком стоят Docker Swarm и Kubernetes. Эти инструменты представляют собой самые распространённые решения в области контейнерной оркестрации. Далее приводится анализ этих систем, их достоинств и недостатков, а также проводится сравнение, чтобы упростить выбор подходящего инструмента.

Docker Swarm — это оркестратор, встроенный в Docker Engine. Он преобразует несколько Docker-хостов в один кластер, упрощая тем самым развертывание и масштабирование контейнеров. Docker Swarm предлагает легкую установку и управление, что делает его идеальным для начинающих в области оркестрации.

Архитектурно Swarm состоит из узлов управления, которые принимают решения и координируют кластер, и рабочих узлов, выполняющих контейнеры по заданным инструкциям. Несмотря на свою простоту и легкость интеграции, Docker Swarm обладает ограниченным функционалом по сравнению с Kubernetes и подходит преимущественно для более мелкомасштабных проектов. Архитектура Docker Swarm представлена на рисунке 1.3.

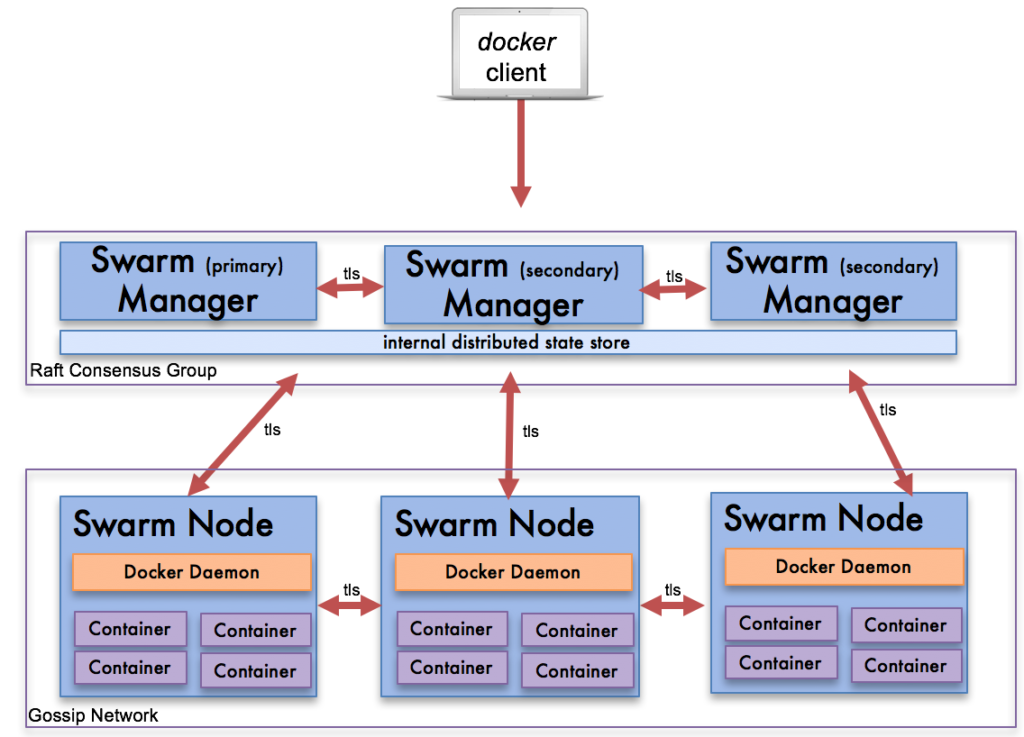


Рисунок 1.3 - архитектура Docker Swarm

Сравнивая Docker Swarm и Kubernetes, стоит отметить, что K8s предлагает значительно более расширенный набор функций и возможностей, что сопровождается повышенной сложностью установки и настройки. Тем не менее, этот оркестратор является предпочтительным выбором для крупномасштабных и сложных проектов, требующих автоматического масштабирования и управления.

Docker Swarm, с другой стороны, хоть и уступает в возможностях, но его простота и интеграция непосредственно с Docker делают его удобным для меньших по размеру проектов, где не требуются продвинутые функции управления.

Итак, выбор между Docker Swarm и Kubernetes во многом зависит от специфики проекта и его требований. Для небольших проектов с ограниченными требованиями Swarm может быть идеальным решением, тогда как для более масштабных и динамично развивающихся систем Kubernetes предложит необходимый уровень контроля и гибкости.

**1.2.3 Apache Mesos**

Apache Mesos представляет собой высокоуровневую абстракцию над кластерами машин, позволяя эффективно управлять ресурсами и распределять задачи среди большого количества серверов. Он предоставляет разработчикам программный интерфейс для управления ресурсами, что делает его особенно полезным в условиях распределённых вычислений и работы с Big Data.

Основная концепция, лежащая в фундаменте Mesos, заключается в его способности делегировать управление запуском задач двухуровневому планировщику. Это означает, что Mesos выступает в качестве "мастера", координирующего распределение ресурсов, в то время как фреймворки, такие как Marathon или Chronos, непосредственно управляют запуском приложений, учитывая предложенные ресурсы. На рисунке 1.4 изображена архитектура кластерного менеджера Apache Mesos.

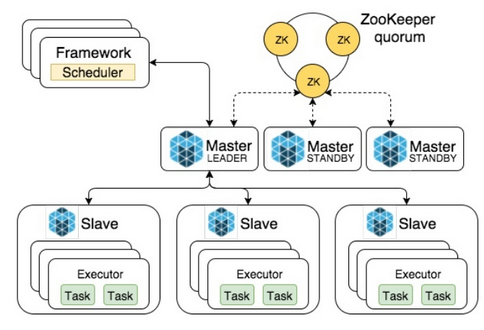


Рисунок 1.4 — архитектура Apache Mesos

В верхней части рисунка 1.4 находится планировщик фреймворка (Framework Scheduler), который взаимодействует с Mesos через ZooKeeper. ZooKeeper - это централизованная служба для обслуживания распределённых систем, которая используется для согласования кластера. Также на рисунке 1.4 показано, что планировщик фреймворка подключается к ZooKeeper, который состоит из трех узлов (ZK), обеспечивая отказоустойчивость и согласованность данных.

Ниже ZooKeeper располагаются узлы Mesos: один мастер-узел (Master Leader), который активен, и два мастер-узла в режиме ожидания (Master Standby), готовые взять на себя управление в случае сбоев.

Ещё ниже расположены рабочие узлы (Slave), которые выполняют задачи (Task). Каждый рабочий узел содержит исполнителя (Executor), который непосредственно запускает и управляет задачами. Executor может выполнять несколько задач одновременно. Эта модель позволяет Mesos эффективно распределять ресурсы и задачи по кластеру, повышая его производительность и масштабируемость.

Apache Mesos поддерживает контейнеризацию через Mesos Containerizer и интеграцию с Docker, что позволяет разработчикам запускать контейнеры в распределенных средах и масштабировать их согласно потребностям проекта. Это обеспечивает гибкость при работе с контейнеризированными приложениями, а также упрощает процесс масштабирования и управления приложениями в кластере.

Apache Mesos подходит для крупных развертываний, где требуется глубокий уровень контроля над ресурсами и где задачи имеют разнообразные требования к вычислительным ресурсам. Он широко используется в индустрии, в частности, для управления ресурсами в облачных средах и центрах обработки данных, где требуется высокая степень масштабируемости и надежности.

С одной стороны, преимуществом Mesos является его мощная и гибкая модель управления ресурсами, что делает его идеальным для запуска и управления масштабируемыми приложениями. С другой стороны, высокий порог входа и сложность управления может оказаться барьером для новых пользователей и проектов с меньшими требованиями к инфраструктуре.

В целом, Apache Mesos представляет собой надежный и мощный инструмент для управления ресурсами в крупномасштабных вычислительных средах, способный обеспечить эффективное выполнение задач в разнообразных доменах, включая облачные вычисления, обработку данных и микросервисные архитектуры.

Apache Mesos представляет собой высокоуровневую абстракцию для управления кластером, оптимизируя работу с большими объемами данных и вычислениями. Это делает его особенно ценным для задач обработки больших данных. В отличие от Mesos, Kubernetes предлагает более широкий спектр возможностей для оркестрации контейнеризированных приложений, облегчая развертывание, масштабирование и управление, что делает его предпочтительным для общих случаев использования. Docker Swarm же выделяется своей простотой и удобством в управлении контейнерами для меньших или менее сложных проектов. Таким образом, выбор между этими технологиями зависит от конкретных требований проекта, где Mesos идеален для специализированных вычислительных задач, Kubernetes подходит для обширного управления приложениями, а Docker Swarm предлагает простоту и удобство для более мелких проектов.

**1.2.4 OpenShift**

OpenShift представляет собой платформу для разработки приложений, основанную на контейнеризации и автоматизированном управлении, разработанную компанией Red Hat. Эта платформа построена на основе Kubernetes и предлагает комплексное решение для разработки, развертывания и масштабирования приложений в контейнерах. OpenShift расширяет возможности Kubernetes, предоставляя дополнительные инструменты и сервисы, которые облегчают жизнь разработчикам и системным администраторам. На рисунке 1.5 представлена архитектура платформы контейнеризации OpenShift

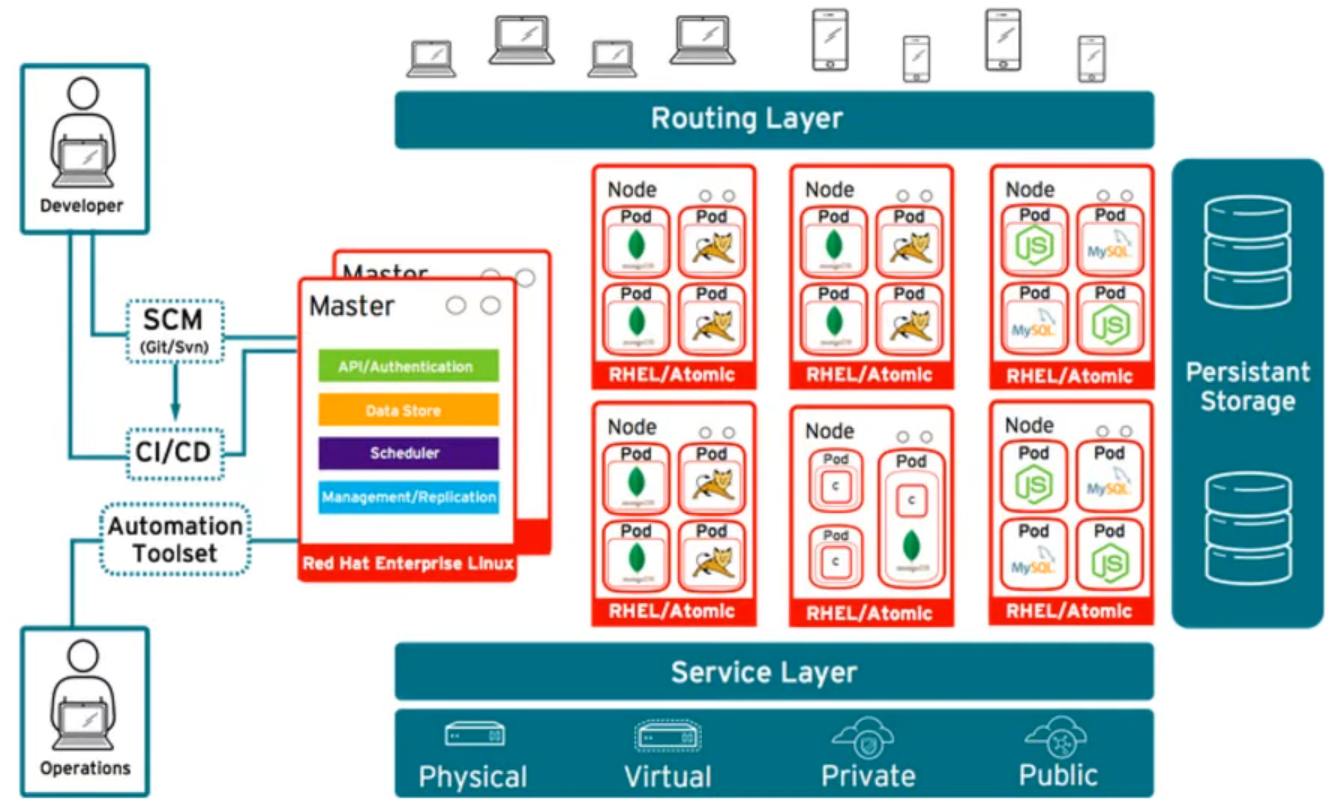


Рисунок 1.5 - архитектура OpenShift

В левой части изображения разработчики и операционные специалисты работают с системами управления версиями (SCM), такими как Git или SVN, и инструментами непрерывной интеграции и непрерывной доставки (CI/CD).

Центральная часть рисунка демонстрирует "Мастер" (Master), который является управляющим узлом в OpenShift. В нём реализованы следующие компоненты:

- API/Authentication: интерфейс API и система аутентификации;

- Data Store: база данных для хранения состояния и конфигураций;

- Scheduler: планировщик для распределения задач по нодам;

- Management/Replication: управление кластером и репликацией ресурсов.

Справа от "Мастера" находится область постоянного хранения данных (Persistent Storage), которая необходима для хранения данных, которые должны сохраняться и быть доступными, даже когда контейнеры пересоздаются или перемещаются по кластеру.

В верхней центральной части диаграммы расположен маршрутизационный слой (Routing Layer), который управляет сетевым трафиком и маршрутами к приложениям, работающим внутри кластера.

Под маршрутизационным слоем располагается сервисный слой (Service Layer), который может быть развернут на физических, виртуальных или облачных (частных и публичных) платформах.

Центральная часть изображения представляет собой узлы (Nodes), на каждом из которых запущены поды (Pods). Поды являются базовыми единицами развертывания в OpenShift и могут содержать один или несколько контейнеров. На рисунке каждый под маркирован логотипом технологии или сервиса, которые в нем работают (например, база данных MySQL или приложение на Node.js). Узлы основаны на операционной системе Red Hat Enterprise Linux (RHEL) или RHEL Atomic, которая оптимизирована для работы с контейнерами.

В целом, диаграмма иллюстрирует, как в OpenShift организована работа с контейнерами, обеспечивается их развертывание и управление, а также как взаимодействуют друг с другом различные компоненты платформы, обеспечивая создание масштабируемой и управляемой среды для приложений.

Основные характеристики OpenShift:

1) Полная интеграция с Docker и Kubernetes: OpenShift использует Docker для управления контейнерами и Kubernetes для оркестрации, добавляя свои уникальные функции и инструменты для упрощения разработки и управления.

2) Разработка и развертывание приложений: OpenShift предлагает мощные средства для CI/CD (непрерывной интеграции и доставки), позволяя разработчикам быстро и эффективно создавать, тестировать и развертывать приложения.

3) Управление жизненным циклом приложений: Платформа обеспечивает автоматизацию многих аспектов жизненного цикла приложений, включая масштабирование, обновления и восстановление после сбоев.

4) Безопасность и соответствие стандартам: OpenShift включает в себя расширенные функции безопасности и управления доступом, что позволяет строго контролировать доступ к ресурсам и данным приложений.

5) Расширяемость и гибкость: Благодаря поддержке широкого спектра языков программирования, фреймворков и баз данных, OpenShift может быть использован для разработки и развертывания самых разнообразных приложений.

6) Интеграция с облачными сервисами: OpenShift легко интегрируется с облачными сервисами и инфраструктурой, что позволяет использовать платформу как в частных, так и в публичных облаках.

Преимущества OpenShift перед аналогами:

1) Полный стек для разработки и эксплуатации: В отличие от чистого Kubernetes, OpenShift предоставляет интегрированный набор инструментов для разработки, мониторинга и управления приложениями.

2) Высокая степень автоматизации: OpenShift автоматизирует множество задач, связанных с управлением контейнерами и кластерами, сокращая тем самым нагрузку на операционные команды.

3) Безопасность "из коробки": OpenShift включает расширенные функции безопасности, такие как политики безопасности по умолчанию, управление секретами и автоматическое сканирование образов на наличие уязвимостей.

4) Масштабируемость и производительность: Основываясь на мощной основе Kubernetes, OpenShift позволяет масштабировать приложения от малых до очень крупных нагрузок, при этом обеспечивая высокую производительность и доступность.

OpenShift, Kubernetes, Docker Swarm, и Apache Mesos представляют собой различные подходы к управлению контейнеризированными приложениями, каждый со своими уникальными преимуществами и недостатками. OpenShift выделяется своей мощностью и гибкостью, предлагая обширную поддержку и интеграцию для корпоративных приложений с акцентом на безопасность и автоматизацию. Kubernetes — это де-факто стандарт оркестрации контейнеров, обеспечивающий универсальность и масштабируемость для управления контейнерами в разнообразных средах. Docker Swarm привлекает своей простотой и интеграцией с Docker, что делает его идеальным для меньших проектов и команд, ищущих легкость в управлении контейнерами. Apache Mesos предлагает высокоуровневую абстракцию для эффективного распределения ресурсов и задач в крупномасштабных вычислительных средах, особенно полезную для специализированных вычислительных задач и обработки больших данных.

Эти технологии в совокупности покрывают широкий спектр потребностей в индустрии разработки программного обеспечения, от простых и небольших проектов до сложных корпоративных приложений, требующих значительных вычислительных ресурсов и высокой степени масштабируемости. Важно провести детальное исследование каждого из этих инструментов в контексте конкретных требований проекта, чтобы определить наиболее подходящий инструмент, который будет способствовать улучшению процессов разработки, развертывания и управления приложениями.

**1.3 Практическое применение контейнеризации**

Приложения и сервисы в современной информационной среде обязаны быть адаптируемыми к быстрым изменениям, обеспечивать безотказную работу и быть готовыми к масштабированию в соответствии с потребностями пользователей. Технология контейнеризации отвечает этим требованиям и предлагает средства для оптимизации всего жизненного цикла приложений. Ниже описаны ключевые аспекты практического применения контейнеризации, актуальные для сферы информационных технологий и за пределами её.

Эффективность разработки и эксплуатации: Контейнеры гарантируют однородность среды на всех этапах разработки, тестирования и развертывания, что упрощает интеграцию и непрерывную доставку (CI/CD) и сокращает "время до рынка" (time-to-market) для новых версий приложений.

Гибкость масштабирования: Системы управления контейнерами, такие как Kubernetes, предоставляют инструменты для автоматического масштабирования приложений в ответ на изменение нагрузки, тем самым оптимизируя использование ресурсов и обеспечивая требуемый уровень сервиса.

Безопасность и изоляция: Контейнеры изолируют приложения друг от друга, уменьшая риск системных уязвимостей. Изоляция также способствует соблюдению стандартов безопасности и соответствия регуляторным требованиям.

Поддержка микросервисной архитектуры: Микросервисы часто реализуются с помощью контейнеризации, что позволяет разрабатывать и развертывать независимые компоненты сложных приложений, ускоряя обновления и улучшения.

Портативность приложений: Контейнеры обеспечивают независимость приложений от инфраструктуры, что позволяет переносить их между локальными серверами, частными и общедоступными облачными средами без изменения кода.

Оптимизация затрат: Контейнеризация позволяет уменьшить затраты на инфраструктуру за счёт повышенной плотности развертывания и улучшенного управления ресурсами, снижая общую стоимость владения (TCO).

Модернизация и интеграция: Технология контейнеризации способствует модернизации существующих приложений и их интеграции с современными облачными сервисами, а также облегчает миграцию легаси-систем.

В заключение, применение контейнеризации в индустрии ИТ не только способствует повышению эффективности разработки и эксплуатации приложений, но и оказывает весомое влияние на ускорение цифровой трансформации предприятий. Оно является критически важным элементом в стратегии повышения агильности бизнеса, обеспечивая необходимую гибкость, скорость и стабильность в изменяющемся цифровом ландшафте.

**1.4 Современные исследования и разработки**

Современные исследования и разработки в области контейнеризации акцентируют внимание на интеграции с облачными хранилищами, обеспечивая масштабируемость и упрощение процессов обновления и резервного копирования приложений. Использование образов контейнеров позволяет ускорить развертывание новых версий приложений и гарантирует их непрерывную работу. Кроме того, облачные хранилища обладают рядом преимуществ, включая гибкость, доступность, безопасность и удобство использования, а также возможность синхронизации данных между устройствами [1].

Принципы работы облачных хранилищ и контейнеризации способствуют оптимизации и эффективности развертывания и масштабирования приложений, предоставляя решения вроде Docker и Kubernetes для автоматизации этих процессов. Это способствует не только повышению производительности и надежности, но и облегчает передачу и развертывание приложений на различных платформах [2].

С другой стороны, концепция нативной облачной архитектуры подчеркивает значимость надежности, самовосстановления, масштабируемости и экономичности, а также обеспечивает простоту сопровождения и повышенный уровень безопасности. Облачные решения предоставляют возможности для автоматизированного масштабирования приложений и оптимизации использования инфраструктурных ресурсов, что способствует снижению расходов и ускорению процессов разработки и отгрузки приложений. Подход cloud-native также предлагает независимость от конкретных вендоров, давая возможность запуска приложений в разнообразных облачных средах [3], [4], [5].

## **1.5** **Выбор фреймворка для разработки**

В выборе фреймворка и архитектуры для дипломного проекта рассматривается интеграция архитектурного паттерна микросервисов с использованием FastAPI как основы для разработки и Docker для контейнеризации приложений. Этот подход поддерживается средой разработки PyCharm, которая предлагает обширные инструменты для работы с Python и фреймворками на его основе.

Архитектурный паттерн микросервисов выбран за его гибкость, масштабируемость и возможность независимой разработки и развертывания каждого сервиса. Принципы микросервисов позволяют разделять функционал проекта на мелкие, легко управляемые части, что упрощает тестирование, поддержку и развитие каждого компонента системы. Для упрощения развертывания и обеспечения согласованности окружений на всех этапах жизненного цикла приложения используется Docker, обеспечивающий контейнеризацию сервисов.

FastAPI выбран как основной фреймворк для разработки благодаря его асинхронной природе, высокой производительности и легкости в использовании. Он поддерживает автоматическую генерацию документации API, валидацию данных и предоставляет мощные инструменты для создания RESTful API. FastAPI также легко интегрируется с различными асинхронными библиотеками и системами управления базами данных, что делает его идеальным выбором для микросервисной архитектуры.

В качестве среды разработки (IDE) используется PyCharm, которая предоставляет расширенную поддержку для Python и его фреймворков, включая FastAPI, упрощает работу с Docker и облегчает разработку микросервисов благодаря интегрированным инструментам для дебаггинга, тестирования и развертывания приложений. PyCharm также поддерживает виртуальные окружения, что позволяет легко управлять зависимостями проекта и обеспечивать его портативность.

Таким образом, интеграция микросервисной архитектуры с FastAPI в контейнерах Docker, разрабатываемых в PyCharm, обеспечивает эффективную и гибкую платформу для разработки современных веб-приложений и микросервисов, соответствующих текущим требованиям к производительности, масштабируемости и удобству разработки.

[1] - https://www.ibm.com/topics/containerization

[2] -

[3] - https://memorygid.com/oblachnye-hranilishha-i-tehnologii-kontejnerizatsii-sovremennye-podhody-k-razvertyvaniyu-prilozhenij

[4] -

[5] - https://memorygid.com/oblachnye-hranilishha-i-tehnologii-kontejnerizatsii-sovremennye-podhody-k-razvertyvaniyu-prilozhenij