1 ОБЗОР ЛИТЕРАТУРЫ

1.1 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.

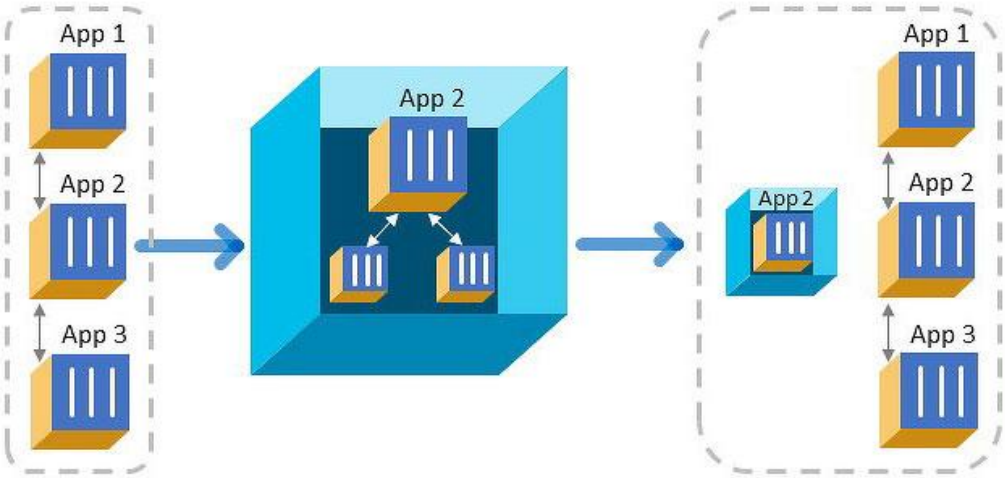

Рисунок 1.1 – Схема контейнеризации

История Docker [1] начинается с презентации Соломона Хайкса на PyCon в 2013 году, что стало значимым моментом в разработке программного обеспечения для приложений. Docker Inc. была основана Камелем Фунаиди, Соломоном Хайксом и Себастьеном Палем в рамках летней программы Y Combinator 2010 года, выросшая из dotCloud — компании, предоставляющей платформу как услугу. Эта компания стала истоком для Docker, который был выпущен как открытый проект в марте 2013 года, переходя от использования LXC к собственному компоненту, libcontainer, что улучшило независимость и эффективность программного обеспечении.

За годы своего развития Docker расширился за пределы своих первоначальных Linux x86 основ, поддерживая другие операционные системы и архитектуры, включая Windows и Arm. Это расширение, вместе с внедрением

оркестраторов, таких как Kubernetes от Google, значительно увеличило влияние Docker, облегчая разработку, обмен и развертывание приложений с невиданной ранее скоростью и эффективностью. Сообщество Docker значительно выросло, Docker Hub и Docker Desktop обслуживают миллионы активных пользователей.

Путь Хайкса, от его ранних дней программирования до управления Docker как смесью открытого исходного кода и бизнеса, подчеркивает трансформационную силу контейнеризации в IT-ландшафте. Успех Docker является примером безупречного сочетания технологии и предпринимательства. Эта философия продвинула Docker в качестве ведущей силы в технологическом мире, стимулируя движение контейнеризации и влияя на траекторию современной разработки программного обеспечения.

С развитием области контейнеризации появилась необходимость в их оркестрации, что представляет собой автоматизированное развертывание, масштабирование и управление контейнерными приложениями. Kubernetes [2], разработанный Google, стал одним из ключевых инструментов в этом процессе, обеспечивая не только распределение нагрузки между контейнерами, но и их взаимодействие в рамках микросервисной архитектуры. Это значительно упрощает разработку и поддержку сложных приложений, делая их более надежными и масштабируемыми.

Инструменты вроде Docker и Kubernetes легли в основу современного программного обеспечения и IT-инфраструктур. Внедрение контейнеризации способствует ускорению процессов разработки, тестирования и развертывания, повышает надежность приложений и упрощает их масштабирование. Одним из важных аспектов контейнеризации также является улучшение безопасности, так как каждый контейнер функционирует как изолированное окружение, снижая тем самым риски вторжения и распространения вредоносного код.

Научное сообщество активно исследует эти технологии, что отражается в большом количестве академических работ и публикаций. В рамках этих исследований выявляются новые методы увеличения эффективности контейнеров, разрабатываются рекомендации по их оптимизации и эксплуатации, а также изучается взаимодействие контейнеризированных приложений с облачными сервисами и инфраструктурой.

Так, контейнеризация и системы оркестрации контейнеров стали неотъемлемой частью современной ИТ-индустрии, предоставляя разработчикам и инженерам эффективные инструменты для создания и поддержания сложных и масштабируемых приложений.

1.2 Анализ существующих систем управления контейнерами

1.2.1 Kubernetes (K8s)

Kubernetes, часто сокращаемый до K8s, является мощным

инструментом для автоматизации развертывания, масштабирования и управления контейнеризированными приложениями в различных средах, от облачных сервисов до частных и гибридных облаков. Созданный инженерами Google на основе их обширного опыта работы с контейнерами и поддерживаемый Cloud Native Computing Foundation, Kubernetes быстро стал 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. Это 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.

Структурно кластер Kubernetes состоит из главных (Master Nodes) и рабочих узлов (Worker Nodes), где главные узлы отвечают за координацию кластера и принятие решений о запуске и распределении приложений, а рабочие узлы — за непосредственное выполнение задач.

К основным компонентам рабочего узла можно отнести: Kubelet - это агент, работающий на каждом рабочем узле, отвечающий за запуск, остановку и поддержание работоспособности контейнеров в соответствии с указаниями API сервера, Kube-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.

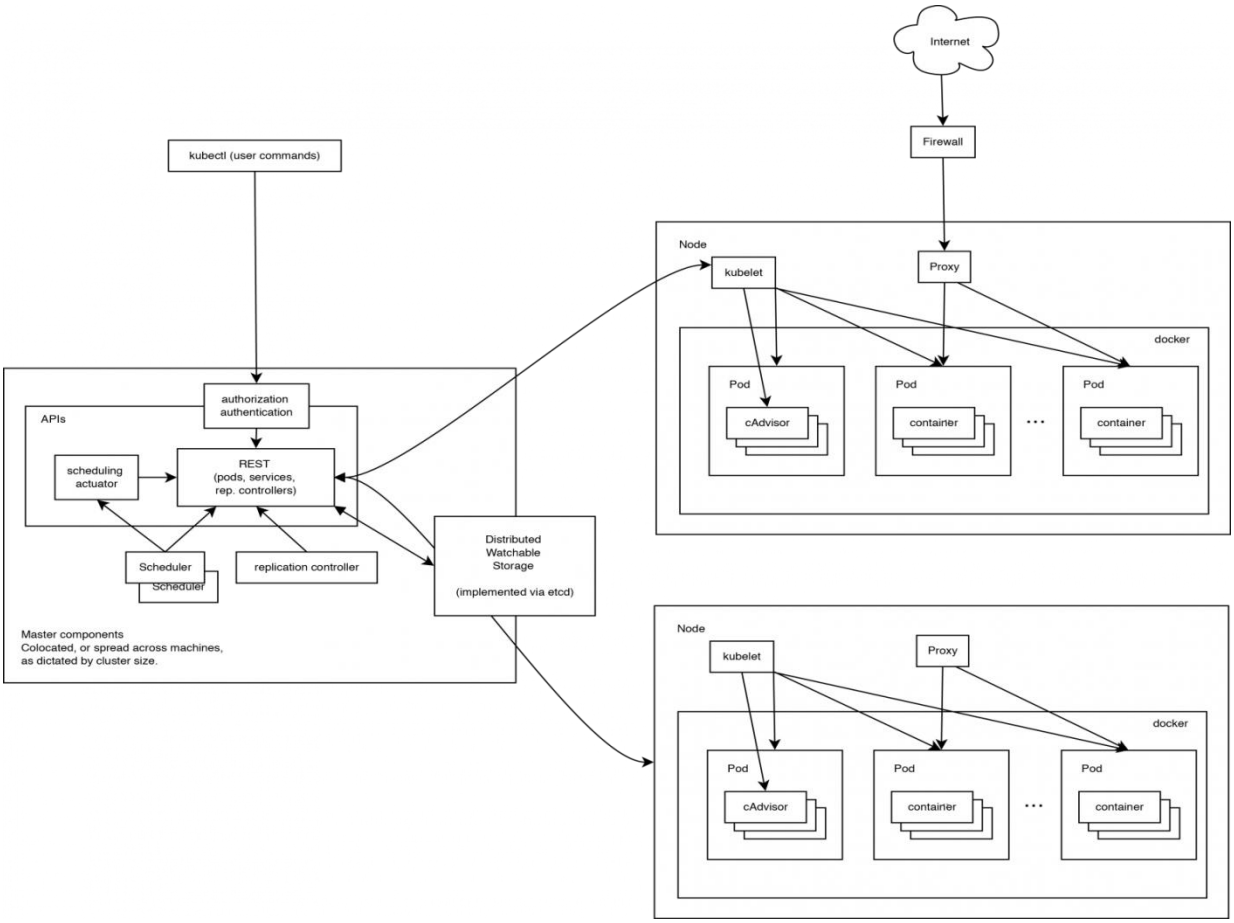

Рисунок 1.2 – Схема состояний Kubernetes

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

Управление контейнерами становится более сложным по мере увеличения их количества, а системы оркестрации, такие как Docker Swarm, предлагают решения для этой проблемы. Docker Swarm, встроенный в Docker Engine начиная с версии 1.12, объединяет несколько хостов Docker в единый кластер, упрощая развертывание и масштабирование контейнеров. Это делает Swarm идеальным для начинающих в области оркестрации контейнеров благодаря его простоте установки и управления.

Архитектурно Docker Swarm организован вокруг узлов управления и рабочих узлов. Узлы управления координируют кластер, принимая решения о распределении задач, в то время как рабочие узлы выполняют эти задачи, запуская контейнеры с приложениями. Архитектура Docker Swarm представлена на рисунке 1.3.

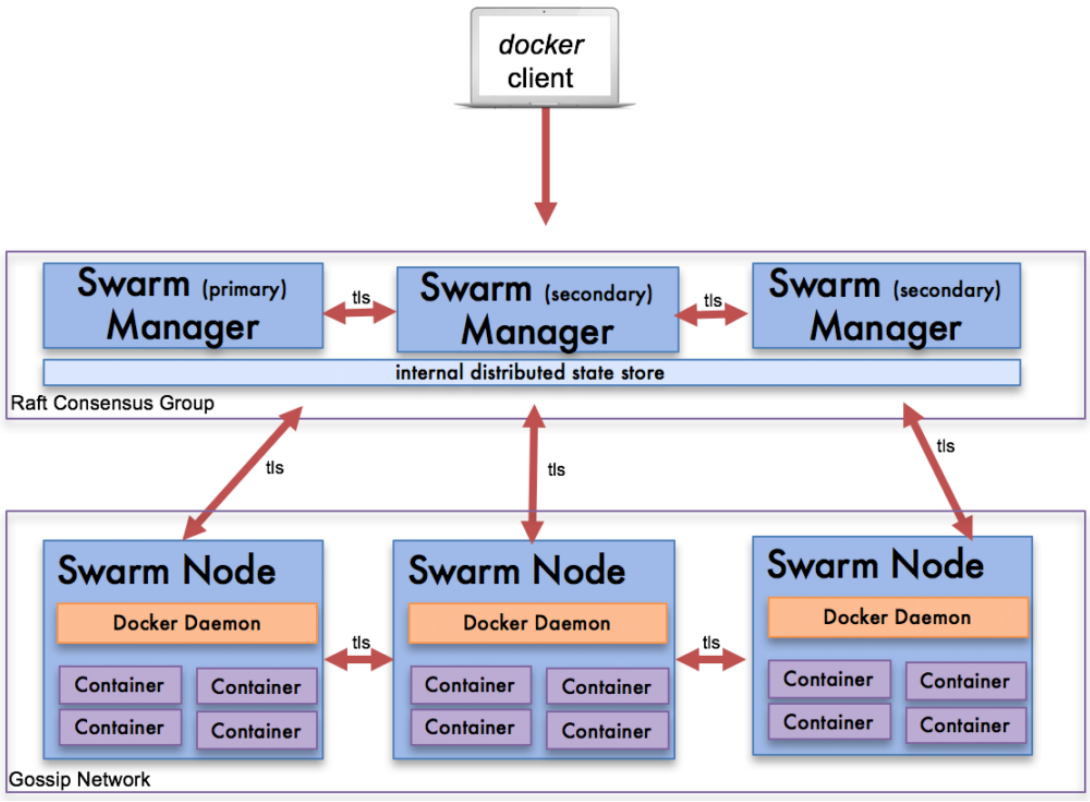

Рисунок 1.3 - Архитектура Docker Swarm

Хотя Docker Swarm отличается простотой и легкостью интеграции, его функционал может быть ограничен по сравнению с более масштабируемым и функциональным Kubernetes, что делает Swarm более подходящим для проектов меньшего масштаба.

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.

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.

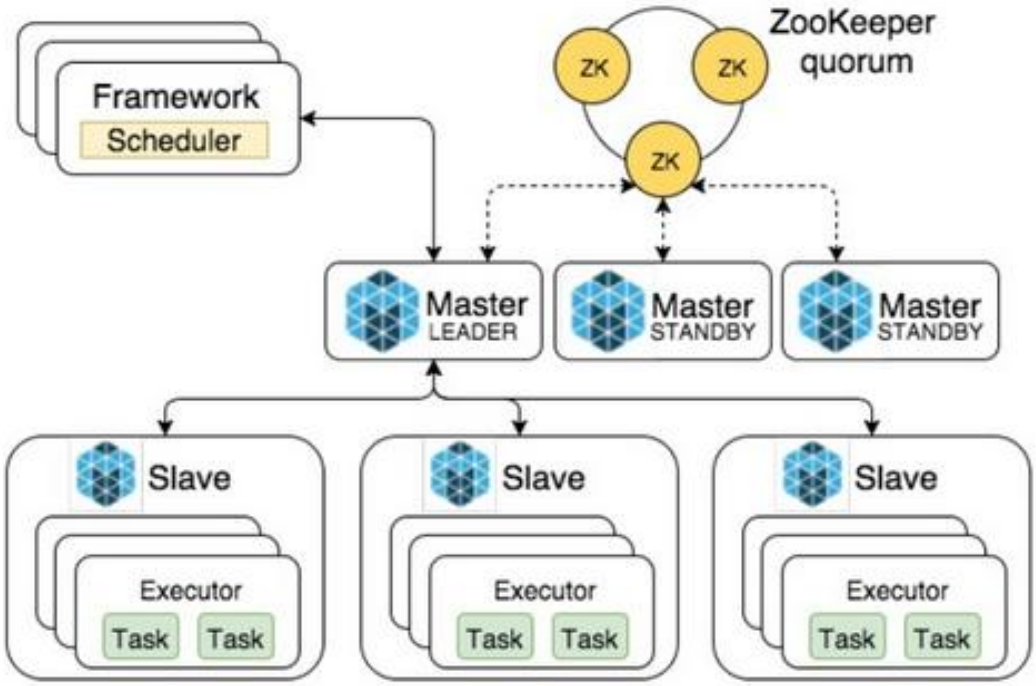

Рисунок 1.4 — Архитектура Apache Mesos

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.

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.

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.

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.

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.

1.5 Выбор фреймворка для разработки

Выбор подходящего фреймворка и архитектурного паттерна для разработки дипломного проекта является ключевым решением, которое определяет не только производительность и масштабируемость конечного продукта, но и его гибкость в долгосрочной перспективе. В этом контексте, интеграция микросервисной архитектуры с использованием FastAPI в качестве основного фреймворка для разработки, а также Docker для контейнеризации приложений, представляет собой передовой подход, поддерживаемый современной средой разработки PyCharm. Эта интеграция не только обеспечивает мощную платформу для создания высокопроизводительных веб-приложений и сервисов, но также значительно упрощает процесс разработки и обеспечивает удобство поддержки проекта.

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, зарекомендовавший себя как передовой и высокопроизводительный веб-фреймворк для создания API в экосистеме Python 3.6 и выше, выделяется среди прочих решений благодаря своему уникальному подходу к разработке. Он применяет стандартные типы Python для объявления переменных, что позволяет разработчикам пользоваться мощью статической типизации, обеспечивая при этом автоматическую валидацию данных и сериализацию и десериализацию без необходимости писать обширный код для этих целей.

Одним из ключевых преимуществ FastAPI является его выдающаяся производительность. Благодаря асинхронной поддержке, веб-фреймворк способен обрабатывать огромное количество запросов параллельно, не жертвуя при этом скоростью ответа. Это ставит FastAPI в один ряд с самыми производительными веб-фреймворками на Python, делая его особенно привлекательным для разработки высоконагруженных приложений и сервисов.

Асинхронная поддержка в FastAPI не просто является дополнительной функцией — она лежит в основе его дизайна. Это позволяет разработчикам эффективно использовать асинхронные запросы и обрабатывать I/O-операции, такие как обращения к базам данных или вызовы внешних API, без блокировки основного потока выполнения. Такой подход максимизирует производительность и эффективность приложений, разработанных на FastAPI.

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.