# 3 ГЛАВА. РАЗРАБОТКА СИСТЕМЫ АВТОМАТИЧЕСКОГО РАЗВЕРТЫВАНИЯ ПРИЛОЖЕНИЙ В ОБЛАЧНОЙ ИНФРАСТРУКТУРЕ

## 3.1 Определение требований к системе автоматического развертывания приложений

Перед проектированием системы развертывания приложений в облачной инфраструктуре необходимо определить основной функционал. Представим данные требования в виде таблицы 3.1

Таблица 3.1. Необходимый функционал системы автоматического развертывания приложений в облачной инфраструктуре.

|  |  |
| --- | --- |
| Категория | Подробное описание |
| Основной функционал | 1. Развертывание приложений 2. Удаление приложений 3. Обновление приложений 4. Оптимизация серверной инфраструктуры 5. Сбор информации о кол-ве свободной ПЗУ и ОЗУ, запущенных процессах |
| Источники файлов приложений | Система непрерывной интеграции программного обеспечения Jenkins |
| Хранение данных | Обеспечение возможности управления версиями, отката и хранения истории операций. |
| Сетевой стек | Передача данных по HTTP |
| Поддерживаемые ОС | Windows, Linux |
| Пользовательский интерфейс | Возможность работы как веб-приложение или как десктоп приложение |

Рассмотрим требования, перечисленные в таблице 3.1 подробнее. Приложение предназначено для автоматического развертывания приложений в облачной инфраструктуре, соответственно, в качестве основного функционала рассматривается возможность:

* развертывания новых приложений,
* обновления уже установленных приложений,
* удаления приложений.

Оптимизация серверной инфраструктуры подразумевает под собой минимизацию количества задействованных серверов при развертывании, обновлении и удалении приложений.

Для использования данного функционала, пользователю необходимо получать следующую информацию о текущем состоянии инфраструктуры:

* запущено или остановлено развернутое ранее приложение;
* общее количество постоянной и оперативной памяти на сервере;
* занятое количество постоянной и оперативной памяти на сервере;
* использование CPU сервера.

Также, пользователю необходимо хранить информацию о развернутых приложениях. Сборка релизов осуществляется в системе непрерывной интеграции Jenkins, релиз хранится в виде архива. Каждое приложение можно рассматривать, как программный компонент, имеющий имя и версию. Хранить имя и версию релиза возможно в текстовом формате в базе данных. При таком формате хранения данных возможно отслеживать используемые в настоящий момент версии приложений, а также, историю обновления, установки или удаления приложений [3].

Работа приложения в локальной сети предприятия подразумевает под собой передачу данных по HTTP.

На виртуальных серверах центра обработки данных применяются операционные системы Windows и Linux, поэтому, также, необходимо обеспечить работы системы развертывания приложений в таких операционных системах.

Преобразуем вышеописанные требования в более формализованный вид (таблица 3.2):

Таблица 3.2. Описание функционала системы и перечень возможных действий пользователя.

|  |  |
| --- | --- |
| Описание функционала | Перечень действий пользователя |
| Пользователю нужно обеспечить возможность выбора серверов (хостов), на которые будет установлено необходимое программное обеспечение, данный перечень пользователю необходимо создавать | Создание, редактирование, удаление хостов. |
| Пользователь должен знать, какие программные компоненты он имеет возможность установить на сервера, поэтому в системе должны храниться имена этих компонентов. | Создание, редактирование, удаление компонентов. |
| Каждый выпуск новой версии программного обеспечения или релиза требует его обновления на серверах. Релиз создается на основе компонента и имеет название и версию. Группировка релизов предназначена для установки коллекции приложений на сервера «в один клик». Группа релизов имеет имя | Создание, редактирование, удаление релизов и их групп. |
| Пользователь должен иметь возможность хранить текущую конфигурацию серверов и устанавливаемых на них релизов, то есть создать среду развертывания, содержащую цель развертывания и компонент развертывания. Цель развертывания – целевой сервер, на котором планируется произвести установку приложения, а компонент развертывания – необходимый релиз. К примеру, целями развертывания будут сервера системы слежения, компонентами развертывания – устанавливаемые приложения, а вместе это будет называться средой развертывания | Создание, редактирование, удаление сред развертывания. |
| Пользователь по желанию должен иметь возможность воспользоваться возможностями программы, позволяющими определить размещение программных компонентов на серверах с целью минимизации числа задействованных при развертывании серверов. | Оптимизация среды развертывания. |
| Развертывание очередного релиза должно происходить по следующему сценарию: пользователь выбирает необходимую среду развертывания, путем сравнения версий, установленных компонентов и компонентов, находящихся в последнем релизе в системе автоматически определяется, какие компоненты необходимо установить на сервер, а какие обновить или удалить. | Разворачивание релизов на среде развертывания. |
| При развертывании приложения необходима настройка его конфигурации. Данные по настройке конфигурации пользователь должен иметь возможность создавать и хранить в виде пар ключ-значение. | Создание, редактирование, удаление параметров конфигурации. |

Исходя из вышеописанных требований данное приложение подразумевает внедрение нескольких программных модулей, а именно:

* компонента, являющего главным и отвечающего за логику выполнения команд пользователя;
* компонентов, запущенных на каждом виртуальном сервере, и выполняющих команды и запросы пользователя;
* клиентского приложения, отвечающего за взаимодействие с пользователем;
* компонента, выполняющего роль шлюза между клиентским и серверным компонентами.

В первой главе данной работы было показано, что при проектировании современных веб-приложений отдается предпочтение микросервисной архитектуре, поэтому выбор технологий для разработки системы развертывания приложений будем учитывать и данный критерий.

## 3.2 Выбор технологии проектирования

В пункте 3.1 были перечислены необходимые программные компоненты для разработки системы развертывания приложений, поэтому выбор технологий для их проектирования произведем в том же порядке.

Технологией разработки веб-приложений, отвечающей требованиям, перечисленным в пункте 3.1, является использующаяся на предприятии платформа .Net Core [4]. Приложения .NET написаны на языке программирования C# и запускают [управляемый код](https://docs.microsoft.com/ru-ru/dotnet/standard/managed-code) в среде выполнения, известной как среда CLR.

На каждом виртуальном сервере предполагается использовать программных агентов, выполняющих команды и запросы пользователя. Эти агенты также будут являться веб-приложениями, реализованными на платформе .Net core.

Так как разрабатываемое приложение будет иметь микросервисную архитектуру, его компоненты будут коммуницировать между собой с использованием протокола HTTP через вызовы API микросервисов и по протоколу AMQP [5]. Также, вызовы API микросервисов будут осуществляться пользователем с помощью клиентского приложения через шлюз.

Выбранная реализация серверного приложения с использованием Rest Api [9] расширяет возможности для реализации клиентского приложения. В этом случае является возможным создать как сетевое desktop-приложение, реализованное с использованием паттерна MVVM, так и веб-клиента, для работы с приложением из веб-браузера.

В качестве системы маршрутизации входящих запросов между клиентским приложением и микросервисами будет выступать шлюз API. В данном случае, ограничимся одним шлюзом API, который будет представлять единую точку входа в приложение. В качестве такого подходит простой легковесный шлюз Ocelot [6], предназначенный для работы с веб-приложениями на платформе .net core.

Для работы с базами даннных в приложениях .net core широко используется объектно-реляционный модуль сопоставления Entity Framework Core [7], поддерживающий множество современных систем управления базами данных.

В качестве системы управления базами данных были рассмотрены 3 наиболее используемых варианта (таблица 3.3):

Таблица 3.3. Наиболее распространенные СУБД.

|  |  |  |  |
| --- | --- | --- | --- |
| Название СУБД | Лицензия | Запуск образа в Docker | Совместимость с Entity Framework Core |
| Oracle Database | Коммерческое ПО | Да | Да |
| PostgreSQL | Открытое ПО | Да | Да |
| Microsoft SQL Server | Коммерческое ПО | Да | Да |

На данный момент все перечисленные СУБД имеют возможность запуска в контейнере Docker, однако PostgreSQL является свободно распространяемым ПО, поэтому остановим выбор именно на СУБД PostgreSQL.

При проектировании программного обеспечения будем следовать концепции проблемно-ориентированного подхода (DDD) [1]. Проблемно-ориентированный подход - это концепция, согласно которой структура и язык программного кода (имена классов, методы классов, переменные класса) должны соответствовать бизнес-области.

Проблемно-ориентированный подход преследует следующие цели:

* сосредоточение внимания проекта на основном [домене](https://en.wikipedia.org/wiki/Domain_(software_engineering)) и логике [предметной](https://en.wikipedia.org/wiki/Domain_(software_engineering)) области;
* построение комплексных проектов на модели предметной области;
* инициирование творческого сотрудничества между техническими [специалистами](https://en.wikipedia.org/wiki/Domain_expert) и [экспертами](https://en.wikipedia.org/wiki/Domain_expert) в [предметной области](https://en.wikipedia.org/wiki/Domain_expert) для итеративного уточнения концептуальной модели, направленной на решение конкретных проблем предметной области.

Концепция модели включает в себя:

* контекст, т.е. обстановку, в которой появляется слово или высказывание, определяющая его значение;
* домен - сферу знаний ([онтология](https://en.wikipedia.org/wiki/Ontology_(information_science))), влияния или деятельности. Предметную область, к которой пользователь применяет программу, является областью программного обеспечения;
* модель - систему абстракций, которая описывает выбранные аспекты домена и может использоваться для решения проблем, связанных с этим доменом;
* единый язык - язык, структурированный вокруг [модели предметной области](https://en.wikipedia.org/wiki/Domain_model) и используемый всеми членами группы для связи всех действий группы с программным обеспечением.

Для поддержания целостности модели необходимо придерживаться следующих принципов:

1. Принцип ограниченного контекста

В любом большом проекте задействовано несколько моделей. Тем не менее, когда код, основанный на разных моделях, объединяется, программное обеспечение становится неполноценным, ненадежным и трудным для понимания. Необходимо определить контекст, в котором применяется модель и соблюдать строгую согласованность модели в этих пределах.

1. Принцип непрерывной интеграции

Когда несколько людей работают в одном и том же ограниченном контексте, модель имеет сильную тенденцию к фрагментации. Чем больше команда, тем серьезнее проблема, но всего три-четыре человека могут столкнуться с серьезными проблемами. Однако при разбиении системы на все более мелкие контексты в конечном итоге теряется ценный уровень интеграции и согласованности. Необходимо организовать процесс слияния всего кода как можно чаще с автоматическими тестами для быстрого выявления фрагментации и использовать единый язык, чтобы выработать общий взгляд на модель по мере того, как концепции развиваются в головах разных людей.

1. Контекстная карта

Индивидуальный ограниченный контекст оставляет некоторые проблемы при отсутствии глобального обзора. Контекст других моделей все еще может быть расплывчатым и изменчивым. Люди в других командах не очень хорошо осведомлены о границах контекста и неосознанно вносят изменения, которые стирают границы или усложняют взаимосвязи. Когда необходимо установить связи между разными контекстами, они имеют тенденцию перетекать друг в друга. В этом случае нужно определить каждую модель, задействованную в проекте, и определить ее ограниченный контекст, назвать каждый ограниченный контекст и сделать имена частью повсеместного языка. Описать точки соприкосновения между моделями, выделив явный перевод для любого общения и выделив любое совместное использование.

Разрабатывая микросервисы, необходимо учитывать, какие образом будет построено их взаимодейтсвие. Здесь применим метод контрактноего программирования. Это метод проектирования [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), предполагающий, что проектировщик должен определить [формальные](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B), точные и верифицируемые спецификации интерфейсов для компонентов системы. Данные спецификации называются «контрактами» в соответствии с [концептуальной метафорой](https://ru.wikipedia.org/w/index.php?title=%D0%9A%D0%BE%D0%BD%D1%86%D0%B5%D0%BF%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B5%D1%82%D0%B0%D1%84%D0%BE%D1%80%D0%B0&action=edit&redlink=1) условий и ответственности в [гражданско-правовых договорах](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%B3%D0%BE%D0%B2%D0%BE%D1%80). Подключая такие контракты как динамические библиотеки можно быть уверенным, что при работе с данными используются одни и те же типа вне зависимости от микросервиса.

При разработке системы развертывания приложений будет применен шаблон проектирования микросервисов CQRS - шаблон разделения ответственности на команды и запросы [8]. Это архитектурный шаблон для разделения операции чтения и обновления для хранилища данных. Гибкость, создаваемая переходом на CQRS, позволяет системе лучше развиваться с течением времени и не позволяет командам обновления вызывать конфликты слияния на уровне домена. CQRS особенно хорошо вписывается в методологию [контрактного программирования](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), в которой используются [утверждения](https://ru.wikipedia.org/wiki/%D0%A3%D1%82%D0%B2%D0%B5%D1%80%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), встроенные в исходный код, описывающие состояние программы в определенные важные моменты. В контрактном программировании утверждения относятся к проектированию, а не к логике выполнения, поэтому их выполнение не должно оказывать влияния на состояние программы. CQRS выгоден для контрактного программирования, так как любой возвращающий значение метод (любой запрос) можно вызывать в утверждениях, не беспокоясь о возможном изменении состояния программы.

## 3.3 Проектирование структуры программного обеспечения

Представим в данном пункте следование принципам проблемно-ориентированного проектирования на практике:

* 1. Необходимо сосредоточить внимание проекта на основном [домене](https://en.wikipedia.org/wiki/Domain_(software_engineering)) и логике [предметной](https://en.wikipedia.org/wiki/Domain_(software_engineering)) области.
  2. Разработать домен каждой предметной области.
  3. Установить связи между разными контекстами и предотвратить перетекания контекстов друг в друга.
  4. В процессе разработки использовать непрерывное тестирование интеграцию кода.

Система автоматического развертывания приложений в облачной инфраструктуре работает следующим образом. На каждом виртуальном сервере уровня L2 и L3 расположены агенты, а на выделенном веб-сервере работает серверный компонент приложения. Агенты – программы-демоны, запущены на каждом сервере. Агенты выполняют команды и запросы, приходящие по локальной вычислительной сети от сервера. Такими запросами могут быть запросы типа операционной системы, кол-ва свободной оперативной и постоянной памяти, запущенных процессов, а командами – команды развертывания, удаления или обновления приложения на сервере.

На выделенном сервере расположена серверная часть системы, отвечающая за обработку управляющих воздействий пользователя, а также шлюз, выполняющие маршрутизацию запросов клиента. Для хранения данных, необходимых в процессе работы приложения используется сервер баз данных.

При разработке микросервиса размер не должен быть важным фактором. Главным должно быть создание слабо связанных служб, что позволяет добавиться автономности при разработке, развертывании и масштабировании каждой сервиса. Конечно же, при определении и проектировании микросервисов следует стремиться к тому, чтобы они были как можно меньше, если только они не имеют слишком много прямых зависимостей от других микросервисов. Внутренняя связанность микросервиса и его независимость от других сервисов важнее его размера.

Согласно принципу предметно-ориентированного проектирования, структура и язык программного кода должны соответствовать бизнес-области. Как было сказано ранее, создавать микрослужбы необходимо на основе ограниченного контекста (в рамках части предметной области). В некоторых случаях ограниченный контекст может состоять из нескольких физических служб, но не наоборот [2].

В пункте 3.1 в таблице 3.2 приведен перечень необходимых пользователю функций для работы с системой развертывания приложений. По данному перечню мы можем выделить ограниченные контексты, представленные в таблице 3.4. Здесь представлен вариант архитектуры с несколькими автономными микросервисами, каждый из которых располагает собственными данными и базой данных. В таблице 3.4 представлен вариант разбиения системы на микросервисы согласно их функционалу.

Таблица 3.4. Перечень микросервисов, соответствующих определенному функционалу.

|  |  |
| --- | --- |
| Функция | Микросервис |
| Создание перечня хостов, на которые будет произведена установка приложений | Микросервис хостов |
| Создание перечня релизов и их группировка | Микросервис релизных групп |
| Создание перечня программных компонентов (приложений), устанавливаемых на сервера | Микросервис компонентов |
| Создание сред развертывания | Микросервис сред развертывания |
| Создание и хранение параметров развертывания | Микросервис параметров развертывания |
| Формирование заданий на обновление, удаление или установку программных компонентов | Микросервис развертывания |
| Размещение программных компонентов на серверах с целью максимально возможного использования ресурсов серверов. | Микросервис оптимизации |

На рисунке 3.1 представлена диаграмма компонентов разрабатываемого приложения. В качестве протокола связи между клиентским приложением и микросервисами служит HTTP. Кроме того, поддерживается асинхронная связь для передачи обновленных данных нескольким службам на основе открытого [протокол](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)а для передачи сообщений между компонентами системы AMQP. Основная идея AMQP состоит в том, что отдельные подсистемы (или независимые приложения) могут обмениваться произвольным образом сообщениями через AMQP-брокер, который осуществляет [маршрутизацию](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), возможно гарантирует доставку, распределение потоков данных, подписку на нужные типы сообщений. В качестве брокера сообщений выступает RabbitMQ.

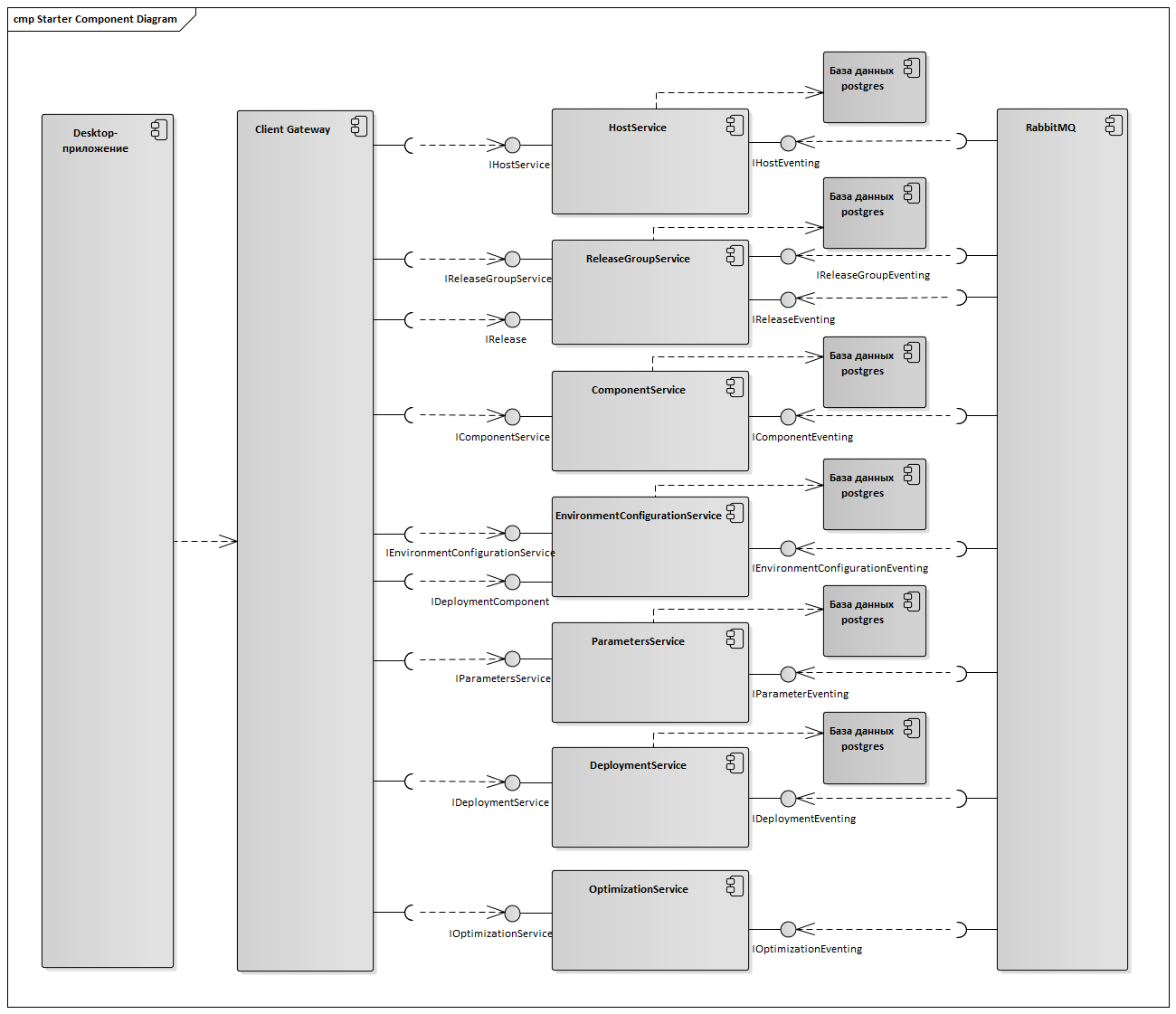


Рисунок 3.1. Диаграмма компонентов системы развертывания приложений.

Единой точкой входа для клиентских приложений является шлюз API. Шлюз API - это точка доступа, которую система предоставляет извне. Шлюз API инкапсулирует внутреннюю архитектуру системы и предоставляет индивидуальный API для каждого клиента. У него также могут быть другие обязанности, такие как аутентификация, мониторинг, балансировка нагрузки, кэширование, фрагментация запросов и управление, обработка статических ответов и т.д.

Ограничительные контексты определены и приложение разделено на микросервисы. Теперь нужно разработать домен каждой предметной области.

При разработке API микросервисов применим [архитектурный стиль](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) взаимодействия компонентов REST [9]. Требование архитектуры REST – идентификация сущностей. Все ресурсы идентифицируются в запросах, например, с использованием [URI](https://ru.wikipedia.org/wiki/URI) в интернет-системах. Ресурсы концептуально отделены от представлений, которые возвращаются клиентам. Например, [сервер](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)) может отсылать данные из [базы данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) в виде [HTML](https://ru.wikipedia.org/wiki/HTML), [XML](https://ru.wikipedia.org/wiki/XML) или [JSON](https://ru.wikipedia.org/wiki/JSON), ни один из которых не является типом хранения внутри сервера. Каждый представленный далее доменный объект является сущностью, имеющей идентификатор, т.е. каждый доменный объект это наследник базового класса, имеющего уникальный идентификатор. Этот функционал вынесен в общую библиотеку классов, подключаемую в каждом проекте при разработке микросервисов, поэтому не представлен на диаграммах классов доменов.

Представим диаграммы классов домена с помощью унифицированного языка моделирования UML.

На рисунке 3.2 представлена диаграмма классов CRUD-сервисов, а именно:

1. Сервиса хостов, домен которого содержит единственный объект Host, отражающий информацию об уникальном виртуальном сервере.
2. Сервиса релизных групп, в котором для организации связи многие-ко-многим создан класс ReleaseGroupMember.
3. Сервиса компонентов, содержащего единственный класс Component.
4. Сервиса параметров развертывания, отвечающего за работу с пользовательскими параметрами, необходимыми при развертывании приложений на серверах.
5. Сервиса сред развертывания. Как сказано в пункте 3.1, в том случае, когда пользователь знает, на какой группе серверов необходимо совершить установку необходимых релизов, а на каких удаление или обновление, он может воспользоваться заранее созданной средой развертывания, которая должна содержать список целей развертывания, а каждая цель развертывания должна содержать список компонентов развертывания. Один компонент развертывания может содержаться как в одной среде развертывания, так и в нескольких, поэтому, предусмотрен объект TargetComponent для организации связи многие-ко-многим.

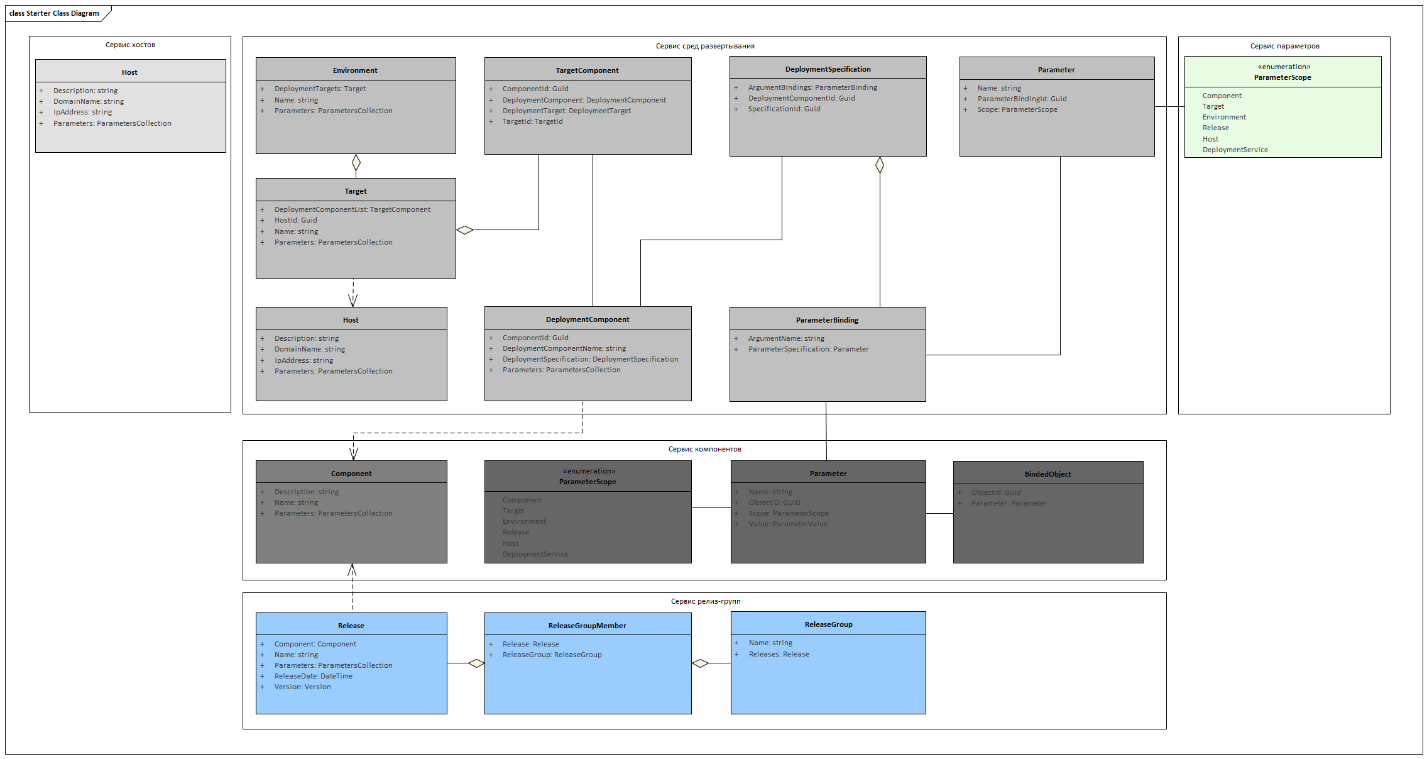


Рисунок 3.2. Диаграмма классов CRUD сервисов.

Рассмотрим связь между ограниченными контекстами сервисов сред, оптимизации и развертывания.

Сервис сред развертывания позволяет создать среду с целями развертывания, которые содержат сервисы, которые разрешено разместить на эти цели. То есть, иными словами, пользователь может создать цели, на которые он рассчитывает установить компоненты, но компоненты при создании среды могут быть размещены на одну цель развертывания, либо распределены случайным образом.

Контракты сервиса сред развертывания подключены в качестве .dll файлов в проектах сервисов оптимизации и развертывания. Сервис оптимизации получает запрос, содержащий сформированную пользователем среду развертывания. Доменные объекты сервиса оптимизации, используемые при работе с серверами и программными компонентами, представляют из себя объекты Server и Service. Объект Server используется для представления и работы с серверами, а Service для представления и работы с сервисами. Коллекции объектов Server содержат коллекции объектов Service. Для работы с доменными объектами сервису оптимизации необходимо произвести маппинг dto-объектов в доменные объекты и получить недостающую информацию от сервиса агентов. По IP-адресу цели развертывания делается запрос сервису агентов для получения таких параметров, как загрузка центрального процессора в процентах, количество свободного места на жестком диске в мегабайтах, количество свободной оперативной памяти, тип операционной системы. По идентификаторам компонентов развертывания также будут получены данные о необходимом проценте процессорного времени в процентах, количестве занимаемой памяти на жестком диске в мегабайтах, количестве необходимой оперативной памяти, типе операционной системы.

Когда все данные о компонентах и целях развертывания получены, создаются коллекции доменных объектов Server и Service сервиса оптимизации, которые будут переданы на вход алгоритма оптимизации. На этом маппинг считается завершенным.

Когда все сервисы распределены на нужные сервера при помощи алгоритма оптимизации, обратное преобразование в контракты сервиса сред происходит следующим образом: создаются dto-объект среды развертывания с вложенными коллекциями целей и компонентов, расположенных в нужном месте. Диаграмма классов сервиса оптимизации представлена на рисунке 3.3. Сделаю в А3 с ортогональными стрелками

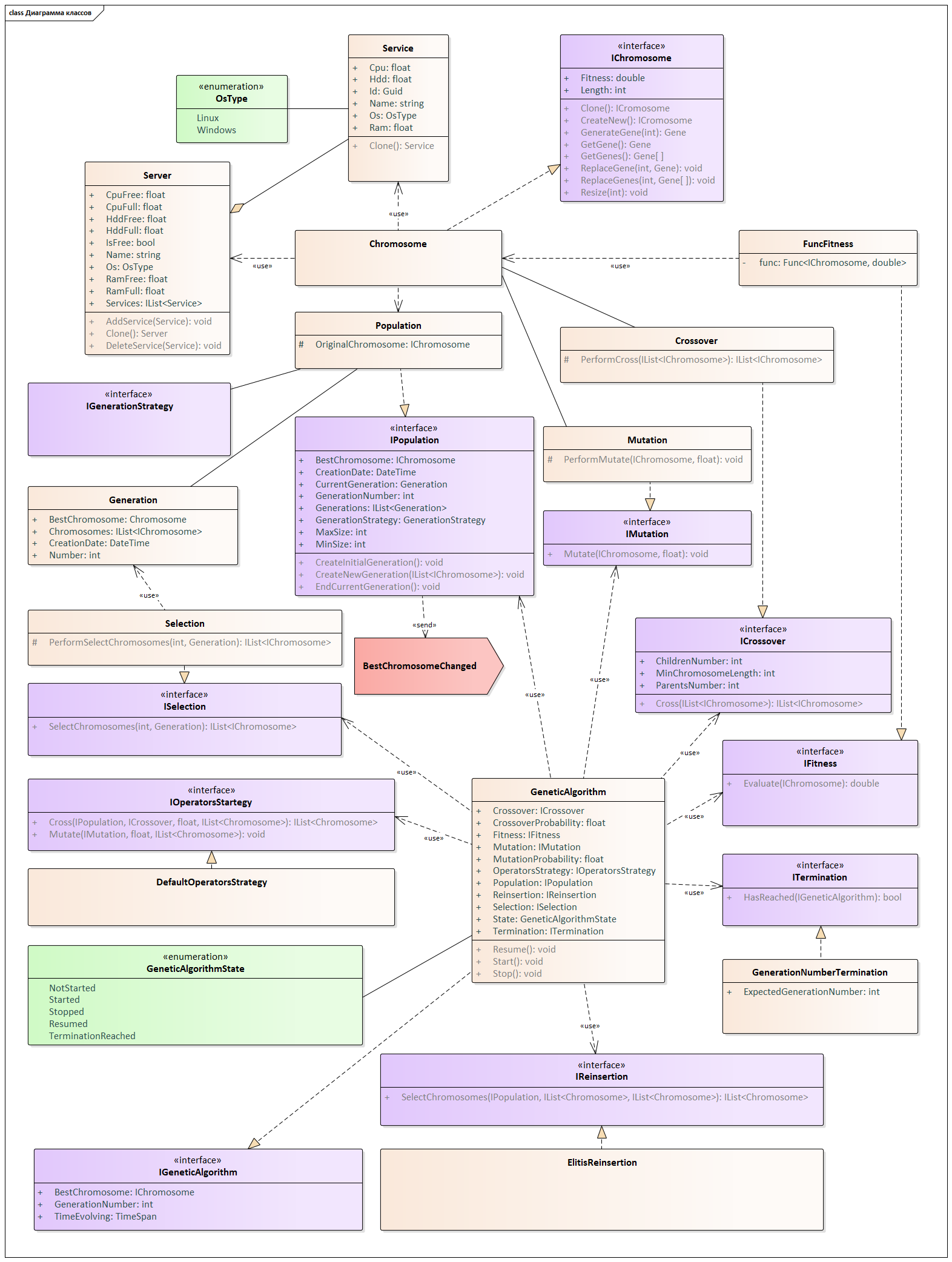


Рисунок 3.3. Диаграмма классов сервиса оптимизации.

Среда развертывания принимается на входе сервиса развертывания. Сервис развертывания несет ответственность за сравнение переданной среды с предыдущей развернутой средой. В зависимости от версии релизов программных компонентов и наличия их в среде развертывания сервис принимает решение об обновлении, удалении или развертывании приложения на сервере. Результатом работы сервиса является объект типа Task, созданный с применением паттерна «Цепочка обязанностей» [14]. При вызове метода Start у данного объекта будут созданы события в брокер сообщений, указывающие агентам, расположенным на серверах, какие операции совершать с программными компонентами. Только при завершении очередного задания будет выслано следующее. Метод Start вызывается в отдельном потоке и не блокирует основной. Ожидание выполнения задания агентами на сервисе развертывания реализовано с помощью объекта AutoResetEvent, который применяется для синхронизации потоков. Диаграмма классов сервиса развертывания представлена на рисунке 3.4.

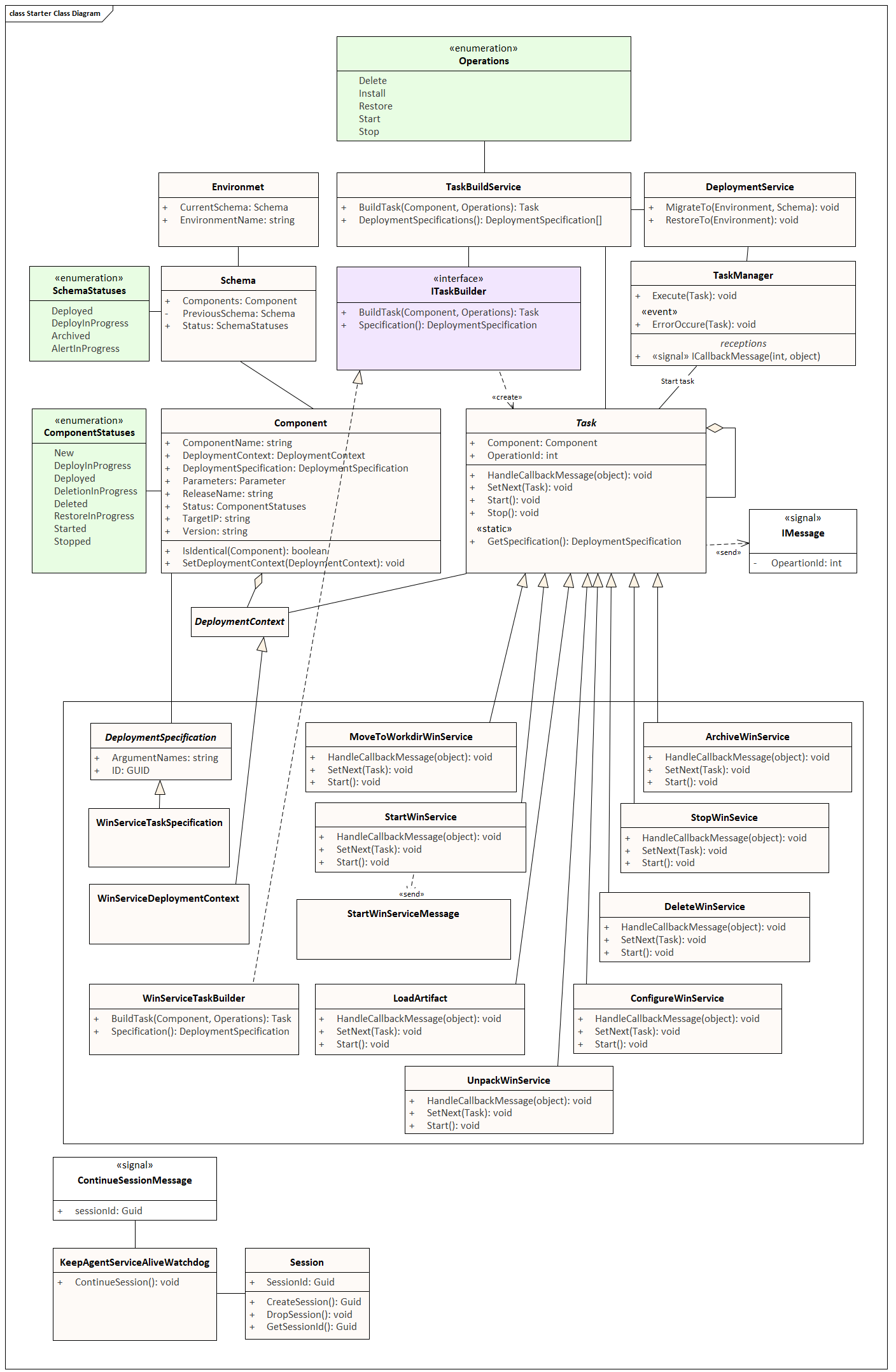


Рисунок 3.4. Диаграмма классов сервиса развертывания.

Согласно рисункам 3.2 - 3.4 можно судить, что границы и размеры всех ограниченных контекстов и моделей предметной области выбраны правильно, так как между этими моделями существуют несколько прочных связей, и не обязательно объединять информацию из нескольких моделей предметной области при выполнении типичных операций в приложении.

Программный интерфейс приложения (API) — описание способов (набор [классов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [процедур](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [функций](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [структур](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) или [констант](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D1%82%D0%B0%D0%BD%D1%82%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))), которыми одна компьютерная программа может взаимодействовать с другой программой. API определяет функциональность, которую предоставляет программа ([модуль](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [библиотека](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))), при этом API позволяет абстрагироваться от того, как именно эта функциональность реализована. Для передачи информации, необходимой для обмена сервисами по HTTP, а также возврата ее конечному пользователю, используется шаблон проектирования Data Transfer Object (DTO) [13]. Объект DTO не содержит какого-либо поведения и является лишь объектом, содержащим данные.

Документация по API сгенерирована с использованием фреймворка Swagger UI [10], позволяющего создать веб-страницу с интерактивной документацией. Скриншоты веб-страниц с документацией представлены ниже. В приложении В приведена документация по API с использованием спецификации OpenAPI, содержащую, также, описание используемых DTO. Сервисы, предназначенные для выполнения вспомогательных функций, такие как сервис хостов, релизных групп, компонентов и параметров развертывания являются CRUD-приложениями, т.е. приложениями, реализующими четыре базовые функции, используемые при работе с базами данных: создание (англ. create), чтение (read), модификация (update), удаление (delete). Опишем программные интерфейсы каждого микросервиса.

API сервиса хостов (рисунок 3.5) позволяет записать информацию о хосте, отредактировать, удалить хост с необходимым идентификатором, а также, получить коллекцию хостов или единственный хост по идентификатору.

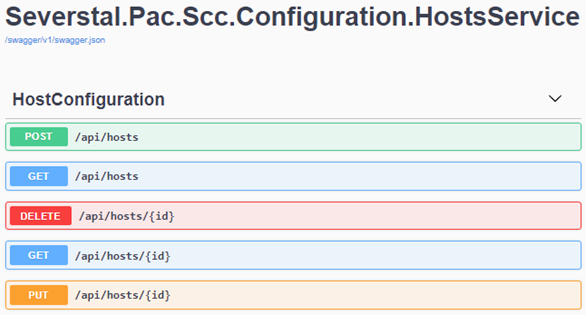


Рисунок 3.5. API сервиса хостов.

API сервиса релиз-групп (рисунок 3.6) позволяет записать информацию о релизе, отредактировать, удалить релиз с необходимым идентификатором, а также, получить коллекцию релизов или единственный релиз по идентификатору.

Из коллекции релизов можно создать группу релизов, добавить/удалить необходимый релиз по идентификатору.

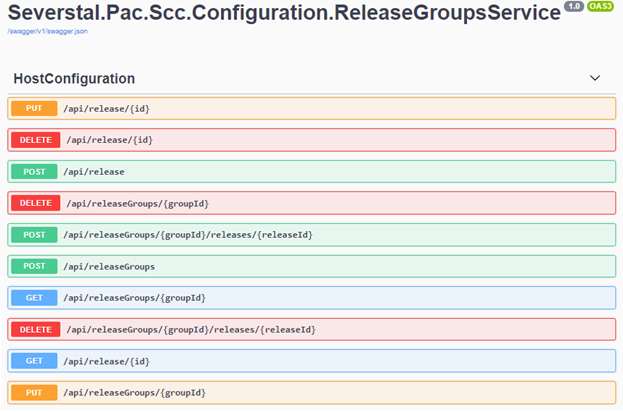


Рисунок 3.6. API сервиса релиз-групп.

API сервиса компонентов (рисунок 3.7) позволяет записать информацию о компоненте, отредактировать, удалить компонент с необходимым идентификатором, а также, получить коллекцию компонентов или единственный компонент по идентификатору.

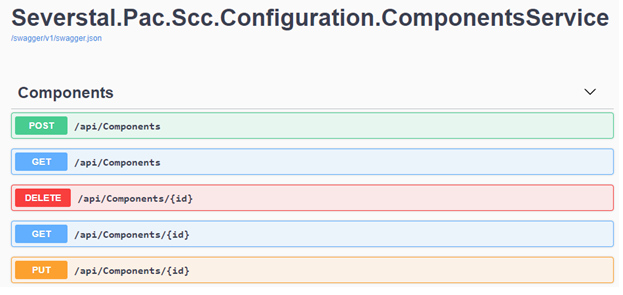


Рисунок 3.7. API сервиса компонентов.

Среда развертывания, как было описано в пункте 3.1, является объектом, содержащим коллекцию целей развертывания, т.е. виртуальных серверов, на которых предполагается установка, обновление или удаление программных компонентов, и вложенные коллекции компонентов развертывания, содержащихся на каждой цели развертывания. С точки зрения API, работа с данным функционалом реализована следующим образом.

Сервис сред развертывания имеет более расширенный API и разделен на две категории:

* компонентов развертывания;
* сред развертывания.

API компонентов развертывания (рисунок 3.8) позволяет осуществлять такие же базовые операции, как и вышеописанные сервисы, т.е. запись информации о компоненте развертывания, редактирование, удаление компонента развертывания с необходимым идентификатором, а также, получение коллекции компонентов или единственного компонента развертывания по идентификатору, а также, имеет функционал для записи, получения и редактирования спецификации развертывания каждого компонента.

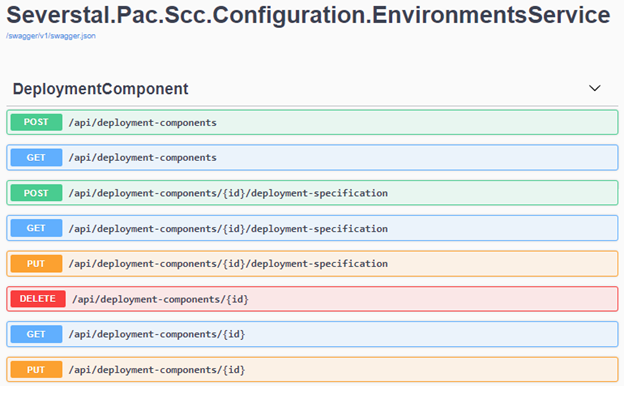


Рисунок 3.8. API сервиса компонентов.

API сред развертывания (рисунок 3.9), кроме базовых CRUD операций со средой развертывания, позволяет добавлять, получать, редактировать и удалять цели развертывания и компоненты развертывания, назначенные на эти цели. Также, реализована возможность прямого запроса коллекции созданных целей развертывания.

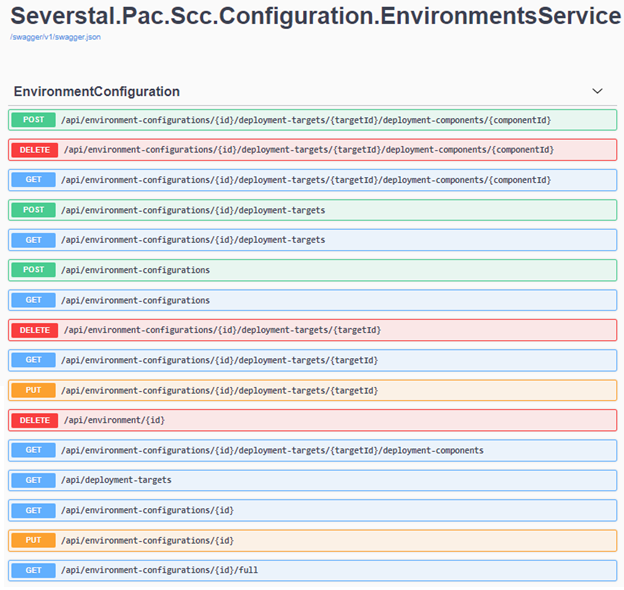


Рисунок 3.9. API сервиса сред развертывания.

API сервиса параметров (рисунок 3.10) предоставляет также типовой функционал, позволяющие производить CRUD операции с параметрами развертывания.

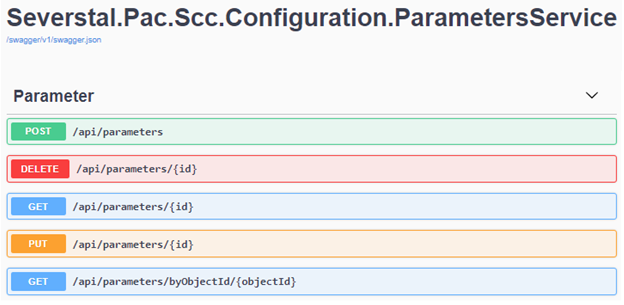


Рисунок 3.10. API сервиса параметров.

Среда развертывания в части распределения программных компонентов может быть оптимизирована при помощи вызова метода optimize API сервиса оптимизации (рисунок 3.11). Метод Post возвращает оптимизированную среду развертывания.

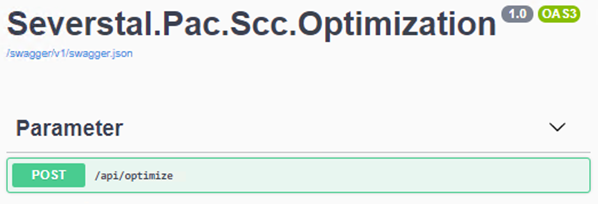


Рисунок 3.11. API сервиса компонентов.

Информация, подготовленная ранее, в частности, среда развертывания и группа релизов, используется при развертывании приложений. API сервиса развертывания представляет собой один метод, принимающий идентификатор среды развертывания и идентификатор группы релизов (рисунок 3.12).



Рисунок 3.12. API сервиса компонентов.

## Разработка БД

В пункте 3.3 данной главы путем сравнения наиболее популярных СУБД выбрана PostgreSQL. При разработке данной системы применен ORM Entity Framework, управляющий созданием и изменением базы данных. База данных создается на основе доменной модели приложения.

Структура базы данных, созданной в СУБД postgres, представляет из себя, собственно базу данных с именем sccdb, и схемы, созданные в данной базе для всех микросервисов. Схема представляет собой пространство имён: она содержит именованные объекты (таблицы, типы данных, функции и операторы), имена которых могут совпадать с именами других объектов, существующих в других схемах. Для обращения к объекту нужно либо дополнить его имя именем схемы в виде префикса, либо установить путь поиска, включающий требуемую схему. Для уникальной идентификации сущностей используется тип идентификатора GUID — [статистически](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0) уникальный 128-[битный](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82) [идентификатор](https://ru.wikipedia.org/wiki/%D0%98%D0%B4%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%82%D0%BE%D1%80).

Имена схем представлены в таблице 3.5.

Таблица 3.5. Схемы базы данных системы развертывания приложений.

|  |  |
| --- | --- |
| Микросервис | Схема базы данных |
| Микросервис хостов | scc\_host |
| Микросервис релизных групп | scc\_release |
| Микросервис компонентов | scc\_component |
| Микросервис сред развертывания | scc\_environment |
| Микросервис развертывания | scc\_deployment |
| Микросервис параметров развертывания | scc\_parameter |

Модели данных представлены на рисунках ниже. Сервисы релиз-групп, хостов, компонентов, параметров, представляющие из себя простые CRUD-сервисы используют упрощенные схемы для хранения данных (рисунки 3.13 – 3.14). В микросервисе сред развертывания реализована возможность создания среды развертывания, добавления целей развертывания в среду развертывания, а также, добавления одинаковых компонентов развертывания на различные цели развертывания. Компонент развертывания содержит, также, информацию о сопоставлении переменных, используемых при развертывании с пользовательскими переменными.

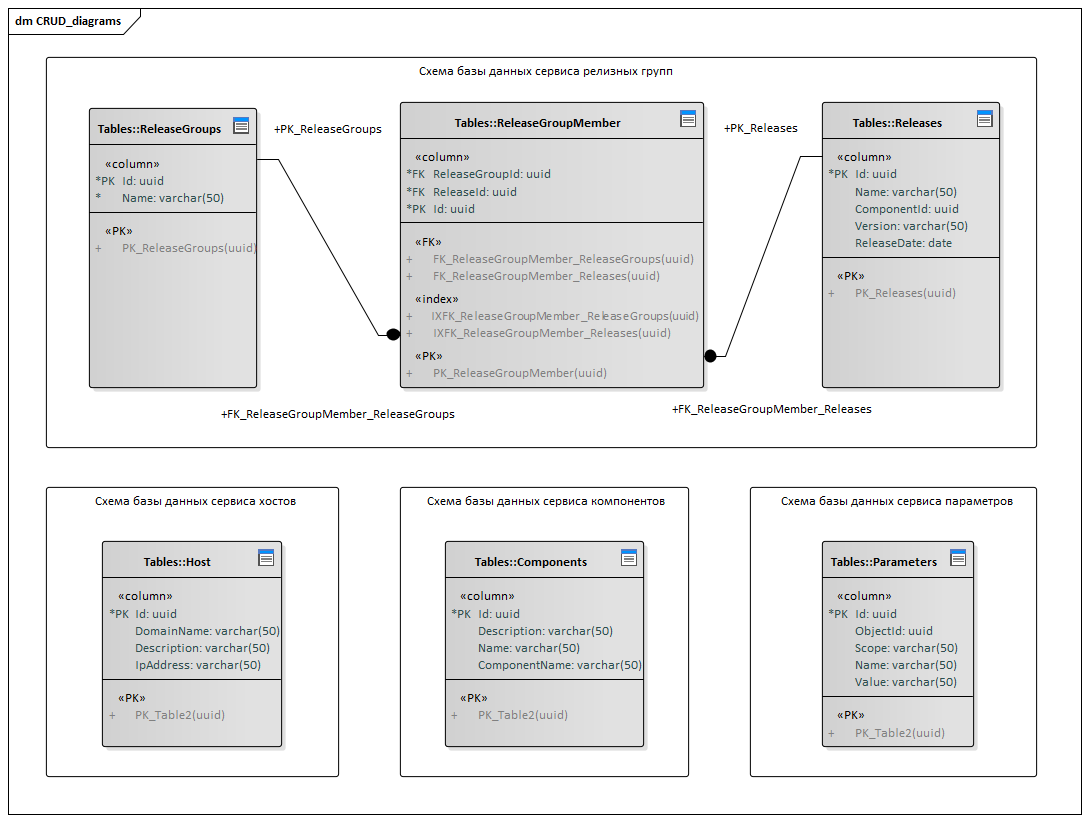


Рисунок 3.13. Схема баз данных CRUD-сервисов.

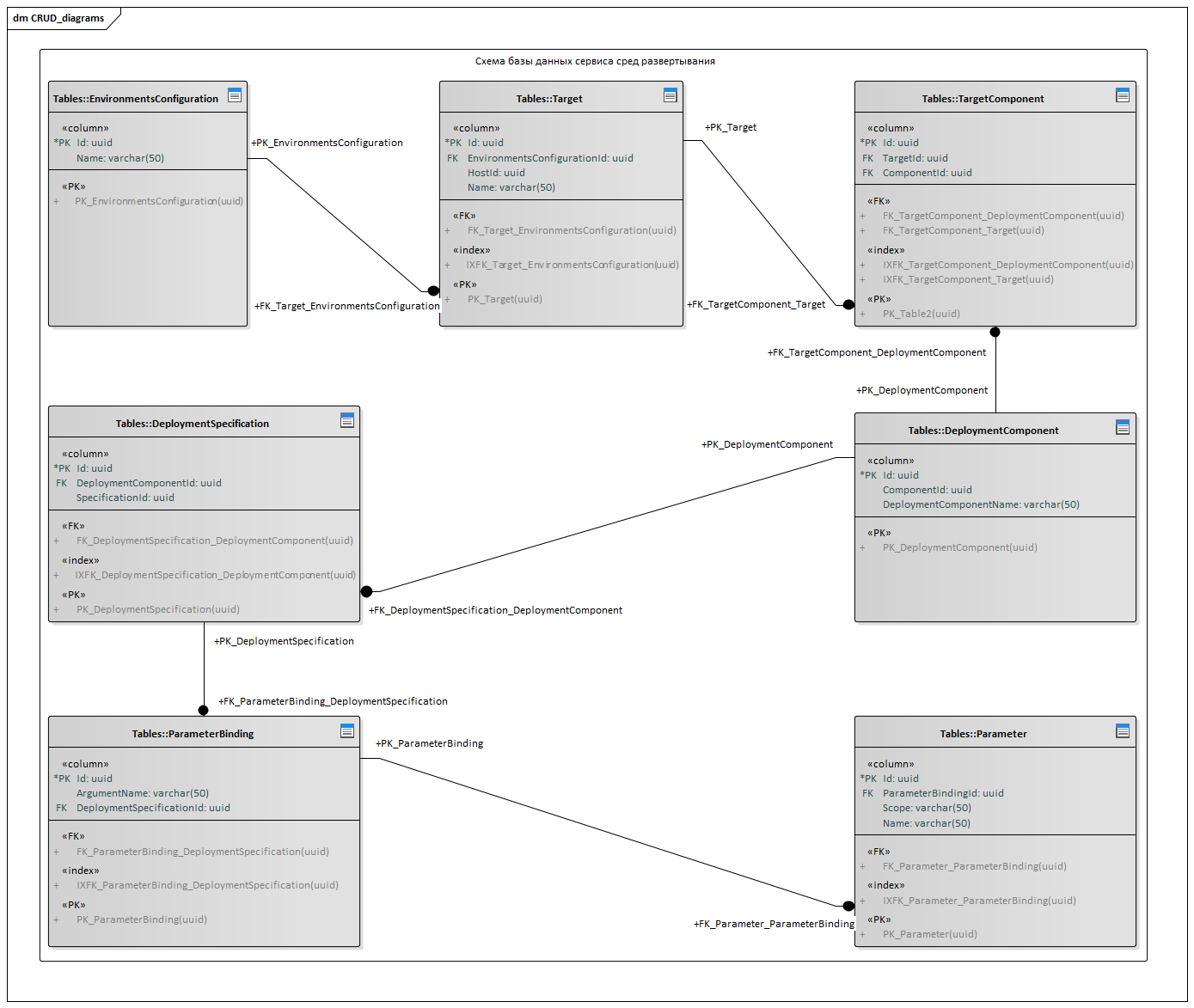


Рисунок 3.14. Схема базы данных сервиса сред развертывания.

В микросервисе развертывания хранение данных реализовано с целью обеспечения отслеживания версий установленного программного обеспечения и возможности возврата к предыдущей версии. Схема базы данных сервиса развертывания представлена на рисунке 3.15.

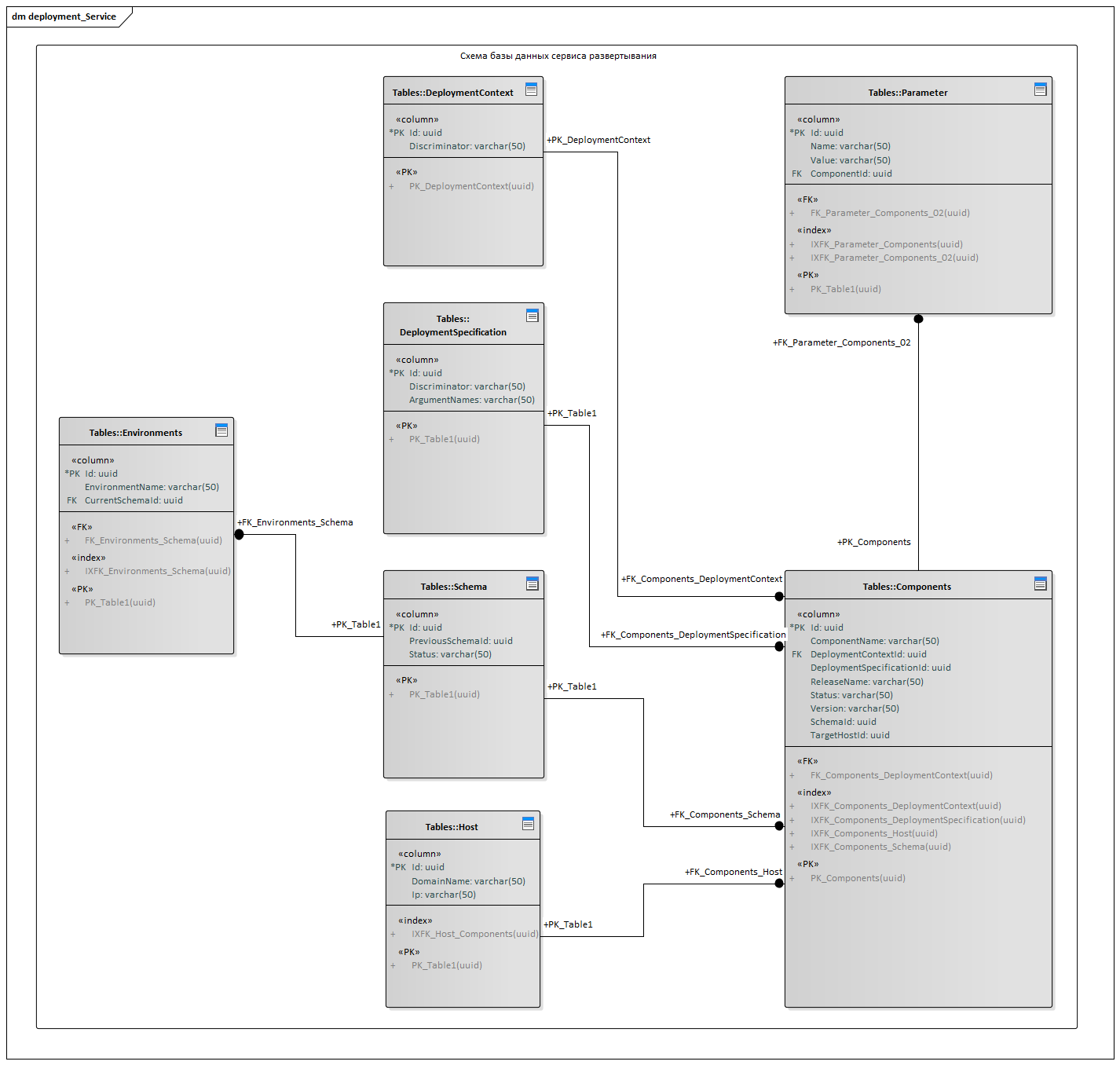


Рисунок 3.15. Схема базы данных сервиса развертывания.

## Разработка интерфейса

При разработке интерфейса используется система построения клиентских приложений WPF, а также, шаблон проектирования MVVM – модель, представление, модель представления.

Данный подход позволяет разделить приложение на три функциональные части:

* **модель** — основная логика программы (работа с данными, вычисления, запросы и так далее);
* **представление** — вид или представление (пользовательский интерфейс);
* модель представления — модель представления, которая служит прослойкой между View и Model.

Такое разделение позволяет ускорить разработку и поддерживаемость программы — можно менять один компонент, не затрагивая код другого.

Как правило, требования к приложению могут меняться с течением времени. Могут появиться новые бизнес-возможности и проблемы, могут стать доступными новые технологии, или даже отзывы клиентов во время цикла разработки могут существенно повлиять на требования к приложению. Поэтому важно писать приложения так, чтобы они были гибкими и могли быть легко изменены или расширены в будущем. Это может потребовать архитектуру, которая позволит отдельным частям приложения разрабатываться и тестироваться независимо, и которые могут быть изменены или обновлены позже, в изоляции, без ущерба для остального приложения. Для упрощения расширяемости и поддержки проекта в будущем будет использован фреймворк Prizm [11], который является эффективным средством для решения этих проблем. Приложение разделяется на некоторое количество дискретных, слабо связанных, полунезависимых модулей, которые затем могут быть легко интегрированы в приложение «оболочку» для формирования цельного решения. Интерфейс построен с применением библиотеки стилей AdonisUI. Ниже представлены скриншоты пользовательского интерфейса различных модулей. Пользователь может создать релиз-группу, компоненты, цели и среду развертывания.

Пользователь создает коллекцию релизов (рисунок 3.16).

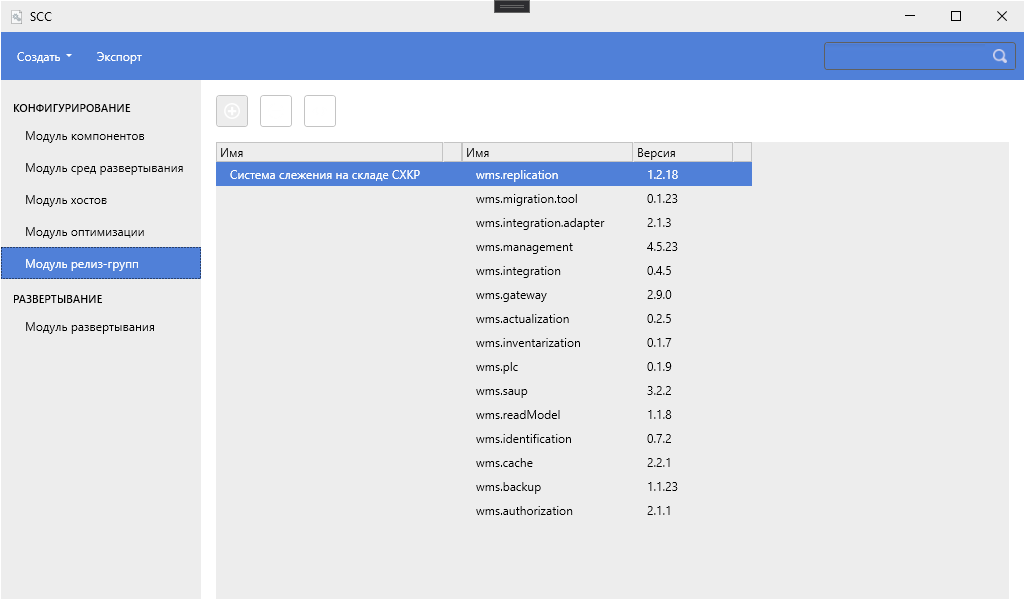


Рисунок 3.16. Модуль релизных групп.

После создания коллекции релизов необходимо добавить нужные программные компоненты в модуле компонентов (рисунок 3.17).

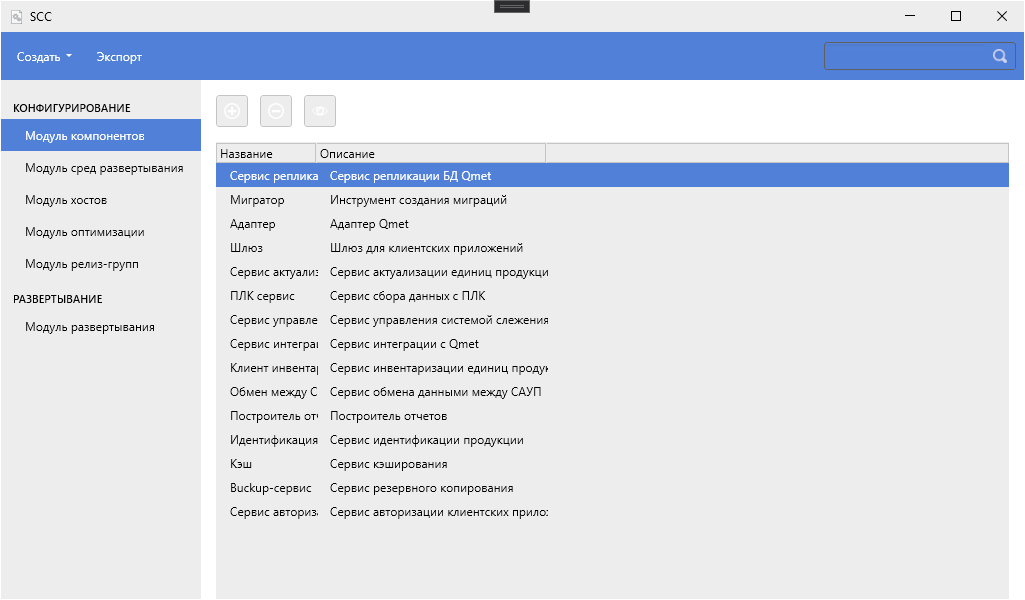


Рисунок 3.17. Модуль компонентов.

Когда созданы программные компоненты, нужно добавить целевые сервера, на которые необходимо установить программные компоненты (рисунок 3.18).

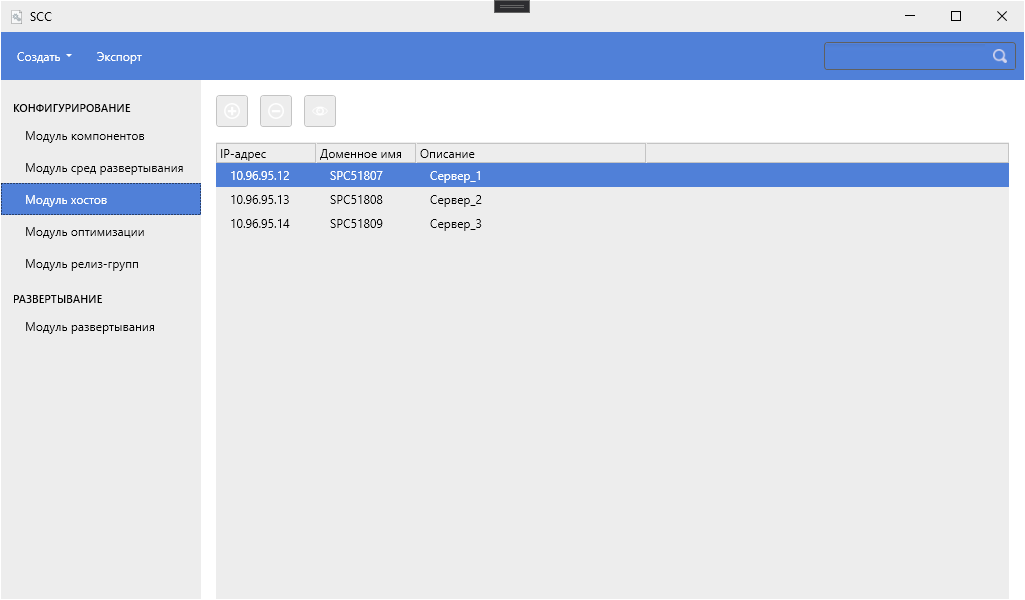


Рисунок 3.18. Модуль хостов.

Среда развертывания создается путем выбора целевых серверов и компонентов, предназначенных для установки на них (рисунок 3.19). Пользователь может назначить установку программных компонентов как вручную, так и автоматически. В последнем случае достаточно добавить все компоненты на один выбранные сервер и перейти в модуль оптимизации.

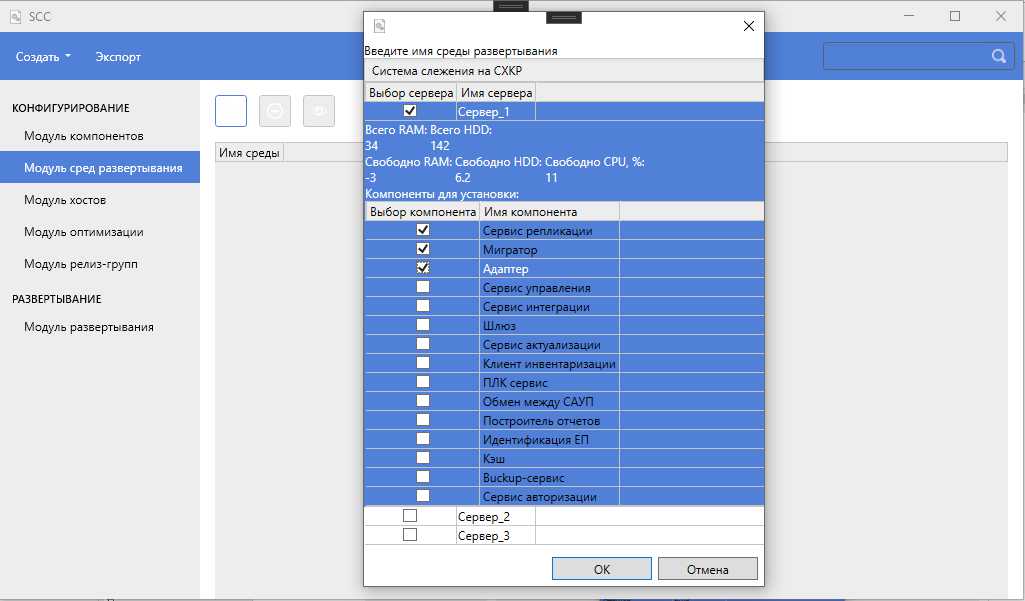


Рисунок 3.19. Модуль сред развертывания.

Когда пользователь создал среду развертывания, он может оптимизировать расположение программных компонентов на ней, запустив механизм оптимизации путем нажатия на кнопку «Оптимизировать». На рисунке 3.20 показан результат работы алгоритма оптимизации.

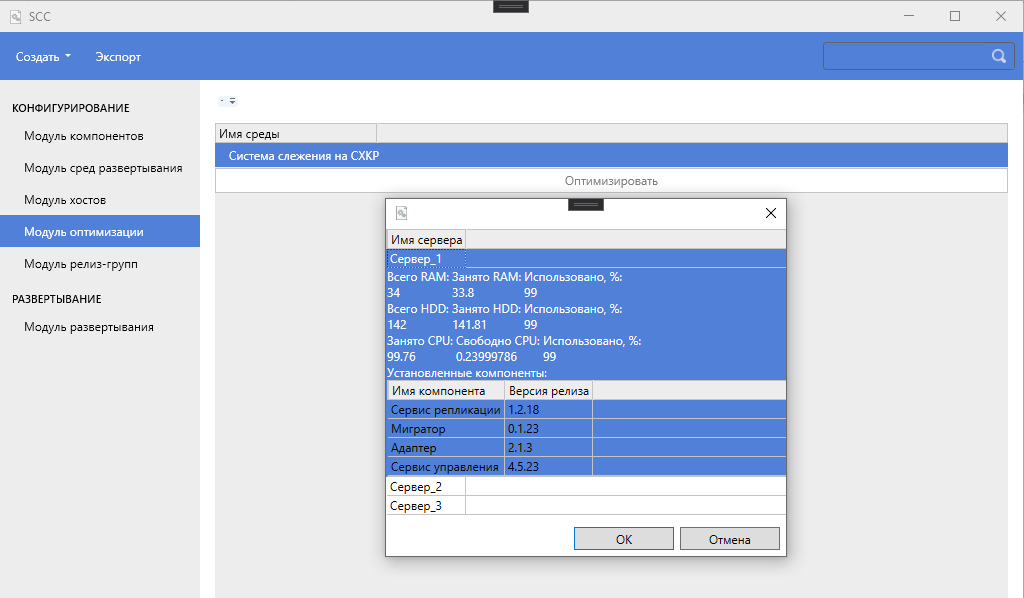


Рисунок 3.20. Модуль оптимизации.

Разворачивание программных компонентов на серверах происходит в автоматическом режиме с последующим оповещением пользователя об окончании работы, либо о том, что произошла ошибка и изменения отменены (рисунок 3.21).

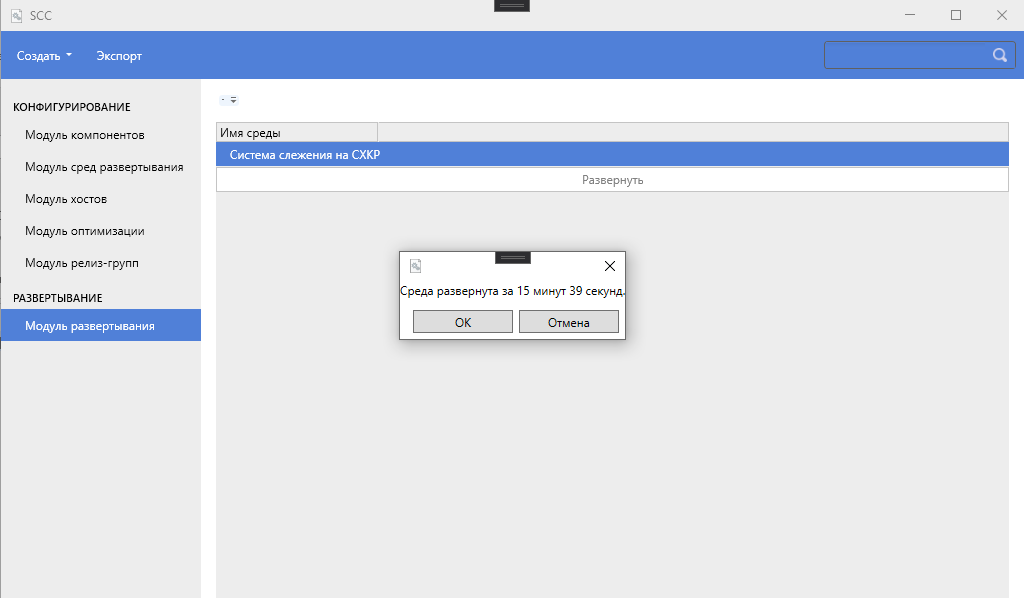


Рисунок 3.21. Модуль развертывания.

# Вывод по главе 3

Перед началом разработки системы автоматического развертывания приложений в облачной инфраструктуре в данной главе был определен список основных функциональных требований. Согласно данному списку был определен стек технологий, которые будут применены при разработке данной системы. Принято решение, что система будет реализовывать клиент-серверную архитектуру. В качестве клиента будет выступать приложение, созданное с использованием системы построения клиентских приложений WPF. На серверной стороне программное обеспечение представляет из себя микросервисное приложение на платформе .NET Core, предоставляющее API для клиентских программ и хранящее информацию в базе данных postgresql. Единой точкой доступа для клиентских программ служит шлюз Ocelot. Основным функционалом приложения является возможность автоматически оптимизировать расположение программных компонентов на виртуальных серверах и развернуть данные компоненты в облачной инфраструктуре.

# Список литературы:

* 1. *[Evans, Eric](https://en.wikipedia.org/w/index.php?title=Eric_Evans_(technologist)&action=edit&redlink=1" \o "Eric Evans (technologist) (page does not exist))* (2004). [*Domain-Driven Design: Tackling Complexity in the Heart of Software*](http://dddcommunity.org/book/evans_2003/). Addison-Wesley. [*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier)) [*978-032-112521-7*](https://en.wikipedia.org/wiki/Special:BookSources/978-032-112521-7)*. Retrieved 2012-08-12*..
  2. Building Microservices: Designing Fine-Grained Systems 1st Edition, Kindle Edition S[am Newman](https://www.amazon.com/Sam-Newman/e/B00LEP6IB0/ref=dp_byline_cont_ebooks_1)  2015 oreilly isbn 9781491950357
  3. Джин Ким, Патрик Дебуа, Джон Уиллис, Джез Хамбл Руководство по DevOps. Как добиться гибкости, надежности и безопасности мирового уровня в технологических компаниях, изд-во манн, иванов и фербер 2018

* 1. [C# для профессионалов. Тонкости программирования](https://www.labirint.ru/books/512895/" \o "Джон Скит - C# для профессионалов. Тонкости программирования)[2 рец.](https://www.labirint.ru/reviews/goods/512895/)  
     [Скит Джон](https://www.labirint.ru/authors/166622/), изд-во Вильямс 2014 608 стр.
  2. RabbitMQ in Depth Gavin M. Roy 2017 ISBN 9781617291005 264 pages Shelter Island, NY 11964
  3. Реализация шлюзов API с помощью Ocelot: сайт. – URL: <https://docs.microsoft.com/ru-ru/dotnet/architecture/microservices/multi-container-microservice-net-applications/implement-api-gateways-with-ocelot> (дата обращения: 14.04.2021). – Текст: электронный.
  4. Начало работы с EF Core в веб-приложении MVC ASP.NET: сайт. – URL: <https://docs.microsoft.com/ru-ru/aspnet/core/data/ef-mvc/intro?view=aspnetcore-5.0> (дата обращения: 11.04.2021). – Текст: электронный.
  5. CommandQuerySeparation: сайт. – URL:<https://martinfowler.com/bliki/CommandQuerySeparation.html>(дата обращения: 20.04.2021). – Текст: электронный.
  6. Руководство по использованию REST API: сайт. – URL: <https://api.mail.ru/docs/guides/restapi/> (дата обращения: 17.04.2021). – Текст: электронный.
  7. Руководство swagger ui: сайт. – URL: <https://starkovden.github.io/swagger-ui-tutorial.html> (дата обращения: 18.04.2021). – Текст: электронный.
  8. Introduction to Prism: сайт. – URL: <https://prismlibrary.com/docs/> (дата обращения: 17.04.2021). – Текст: электронный.
  9. Дейт К. Введение в системы баз данных.- К.; М.; СПб.: Изд.дом "Вильямс", 2000.- с.
  10. Software Engineering for Internet Applications Eve Anderrson, MIT Press 2006; ISBN 0262511916;
  11. Погружение в паттерны проектирования Александр Швец Интернет-издание 2018 год 306 страниц