# 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. Сбор информации о кол-ве свободной RAM 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 | Возможности 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.

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 каждым сервисом (наверное, тогда общее занятое кол-во и использование нашими сервисами).

Также, пользователю необходимо хранить информацию о развернутых приложениях. Приложение хранится в системе Jenkins в виде архива, содержащего файлы этого приложения. Каждое приложение можно рассматривать, как программный компонент, имеющий имя и версию. Хранить такую информацию 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.

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:

1. Для того, чтобы пользователь имел возможность выбора серверов, на которые будет установлено необходимое программное обеспечение, данный перечень пользователю необходимо создавать.
2. Пользователь должен знать, какие программные компоненты он имеет возможность установить на сервера, поэтому в системе должны храниться имена этих компонентов.
3. 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.
4. Инфраструктура центра обработки данных, на работу в которой ориентировано данное приложение, претерпевает изменения относительно нечасто, к примеру, новые виртуальные сервера появляются в среднем каждые сколько? Когда пользователь знает, на какой группе серверов необходимо совершить установку необходимых релизов, а на каких удаление или обновление, он создает 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.
5. Пользователь по желанию должен иметь возможность воспользоваться возможностями программы, позволяющими определить размещение программных компонентов на серверах с целью максимально возможного использования ресурсов серверов.
6. Развертывание очередного релиза, таким образом, будет происходить следующим образом: пользователь выбирает необходимую среду развертывания и релиз с нужной версией (или релиз по умолчанию последний?). Путем сравнения версий в системе автоматически будет определено, какие релизы необходимо установить на сервер, а какие обновить, или удалить.
7. При развертывании приложения необходима также его настройка его конфигурации. Данные по настройке конфигурации необходимо также создавать и хранить в виде пар ключ-значение.
8. В первой версии программы не будет разделения на администраторов и пользователей, пользователю регистрируется в качестве администратора.

Исходя из вышеописанных требований данное приложение подразумевает использование нескольких программных компонентов, а именно: (Почему мы будем реализовывать именно веб-api?).

* 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 Структура фрагмента информационной сети предприятия

При проектировании системы развертывания приложений необходимо также выявить особенности сетевой инфраструктуры, где будет работать данное приложение. На рис. 3.1 представлена локальная вычислительная сеть. Пользователи данной подсети работают в одном домене. На один из серверов центра обработки данных будет установлено и разрабатываемое программное обеспечение. Про пользователей написать.

Топология сети представлена на рисунке 3.1



Рисунок 3.1. Топология фрагмента сети предприятия.

На рисунке 3.1 представлена топология фрагмента локальной вычислительной сети, в которой будет запущена система автоматического развертывания приложений. Инфраструктура сети, представленная на рисунке 3.1 включает в себя центр обработки данных и локальную сеть офиса. Центр обработки данных состоит из 4 физических серверов, на которых, свою очередь, развернуты виртуальные сервера L2 и L3 уровня под управлением системы аппаратной виртуализации Microsoft Hyper V. На виртуальных серверах запущены операционные системы Linux и Windows. Подсеть серверов центра обработки данных отделена от локальной вычислительной сети офиса межсетевым экраном. 10 Гбит Пропускная способность сети для передачи файлов приложений.

## 3.3 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.

Наиболее распространенной технологией разработки веб-приложений, отвечающей требованиям, перечисленным в пункте 3.1, является использующаяся на предприятии платформа .Net Core. 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.

.NET [CLR](https://docs.microsoft.com/ru-ru/dotnet/standard/clr) — это кроссплатформенная среда выполнения, которая включает поддержку Windows, macOS и Linux. Среда CLR обрабатывает выделение памяти и управление ей. Среда CLR также является виртуальной машиной, которая не только выполняет приложения, но и создает, а также компилирует код с помощью JIT-компилятора. Таким образом достигается платформонезависимость. На платформе .NET создается веб-приложение, выполняющееся на сервере и обрабатывающее запросы пользователя. При помощи данной платформы будет создан главный серверный компонент, отвечающий за логику выполнения команд пользователя.

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 предполагается разработать микросервисное приложение, его компоненты будут коммуницировать между собой с использованием протокола HTTP через вызовы API микросервисов и по протоколу AMQP. Также, вызовы API микросервисов будут осуществляться пользователем с помощью клиентского приложения через шлюз.

Выбранная реализация серверного приложения с использованием Rest Api 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. Для реализации клиентского приложения в данной работе будет использована технология создания клиентского приложения, как desktop-приложения с использованием паттерна MVVM.

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, 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, поддерживающий множество современных систем управления базами данных.

В качестве системы управления базами данных были рассмотрены 3 наиболее используемых варианта:

|  |  |  |  |
| --- | --- | --- | --- |
| 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 (а ничего, что это ПО работает на предприятии?) поэтому остановим выбор именно на СУБД PostgreSQL.

Писать про ORM – entity, mapper – automapper, container – autofac

При проектировании программного обеспечения будем следовать концепции проблемно-ориентированного подхода (англ. domain driven design, DDD). 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: в пункте 3.4 перечислить как я следую данным принципам

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 The Command and Query Responsibility Segregation (CQRS) - шаблон разделения ответственности на команды и запросы. 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.

## 3.4 Проектирование структуры программного обеспечения

Вот откуда можно списать <https://habr.com/ru/company/microsoft/blog/346746/>

Веб апи <https://docs.microsoft.com/ru-ru/aspnet/core/tutorials/first-web-api?view=aspnetcore-5.0&tabs=visual-studio>

https://russianblogs.com/article/92241588473/

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.

На рисунке 3.2 представлена структурная схема системы развертывания приложений, на которой показаны основные связи между компонентами системы, расположенными на различных машинах в локальной вычислительной сети. Сервер баз данных, веб сервер и шлюз расположены на выделенных виртуальных серверах, агенты установлены на каждый виртуальный сервер центра обработки данных, пользователи имеют доступ по сети к ресурсам центра обработки данных.



Рисунок 3.2. Структурная схема системы развертывания приложений.

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 приведен перечень необходимых пользователю функций для работы с системой распределения приложений. По данному перечню мы можем выделить ограниченные контексты, представленные в таблице 3.2. В таблице 3.2 представлен вариант архитектуры с несколькими автономными микросервисами, каждый из которых располагает собственными данными и базой данных. В таблице 3.2 представлен вариант разбиения системы на микросервисы согласно их 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 |
| Развертывание приложения на сервера | Микросервис развертывания |
| Создание и хранение параметров развертывания | Микросервис параметров развертывания |
| Аутентификация пользователей | Микросервис аутентификации пользователей |
| 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 3.3 представлена функциональная схема микросервисного приложения. В качестве протокола связи между клиентскими приложениями 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.

Писать про контракты AMQP

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. Упрощенная функциональная схема работы шлюза представлена на рисунке 3.4.

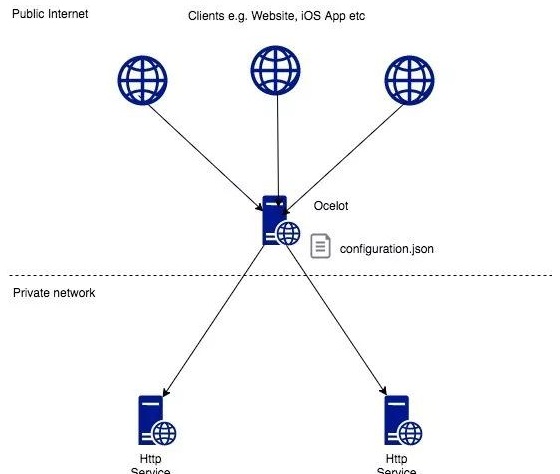


Рисунок 3.4. Функциональная схема шлюза



* 1. Функциональная схема системы развертывания приложений.

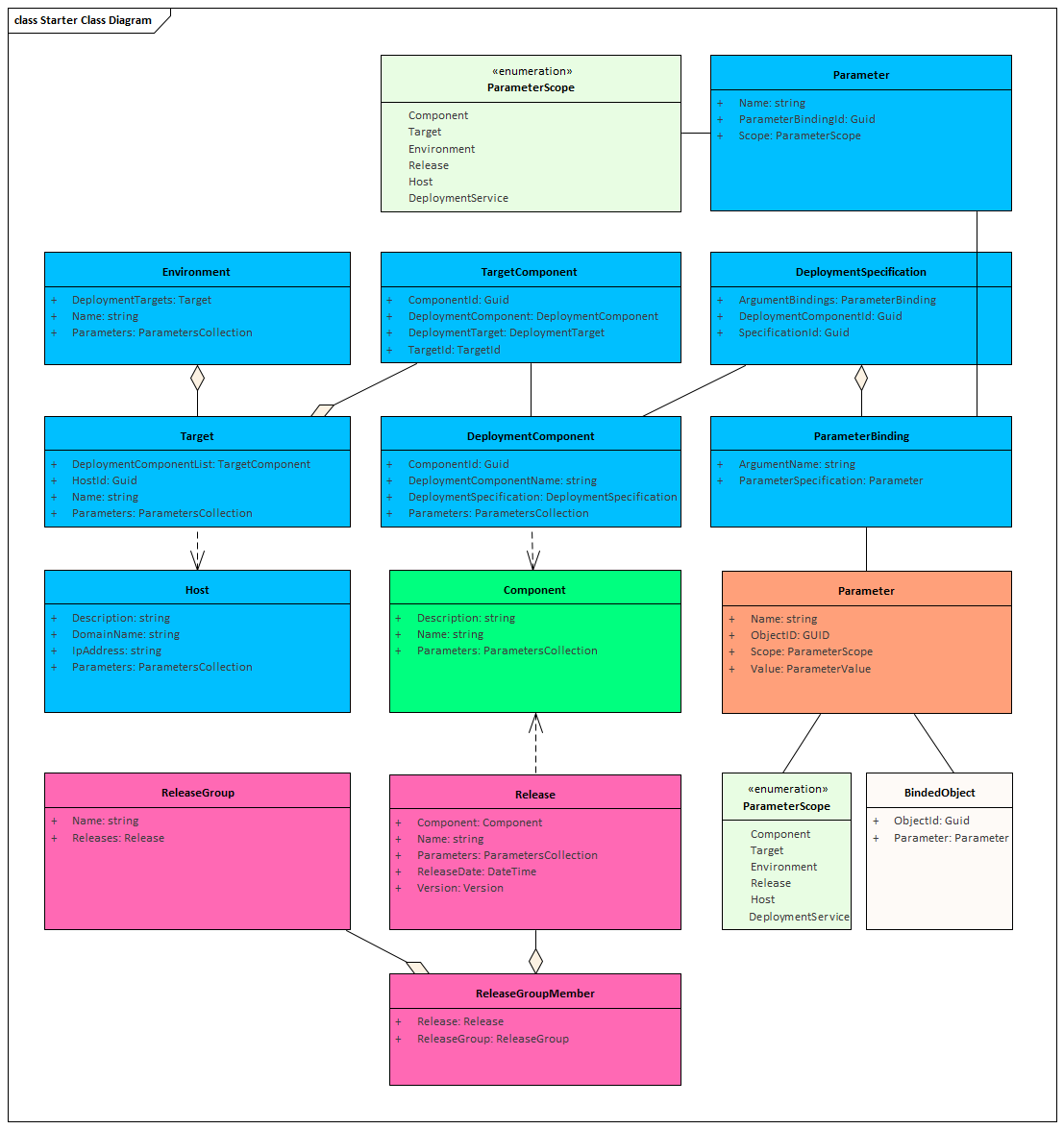
Ограничительные контексты определены и приложение разделено на микросервисы. Теперь нужно разработать домен каждой предметной области.

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 (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Representational State Transfer — «передача состояния представления»). 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. 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 3.5 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. Однако, не имеет смысла создавать разные среды развертывания, содержащие некоторые одинаковые цели развертывания. В данный момент мы можем создать новую среду и должны создать новую цель развертывания ту же самую, если нужно.



Можно блок-схему алгоритма создания среды, передачи ее сервису оптимизации или в обход сервиса оптимизации напрямую развертыванию

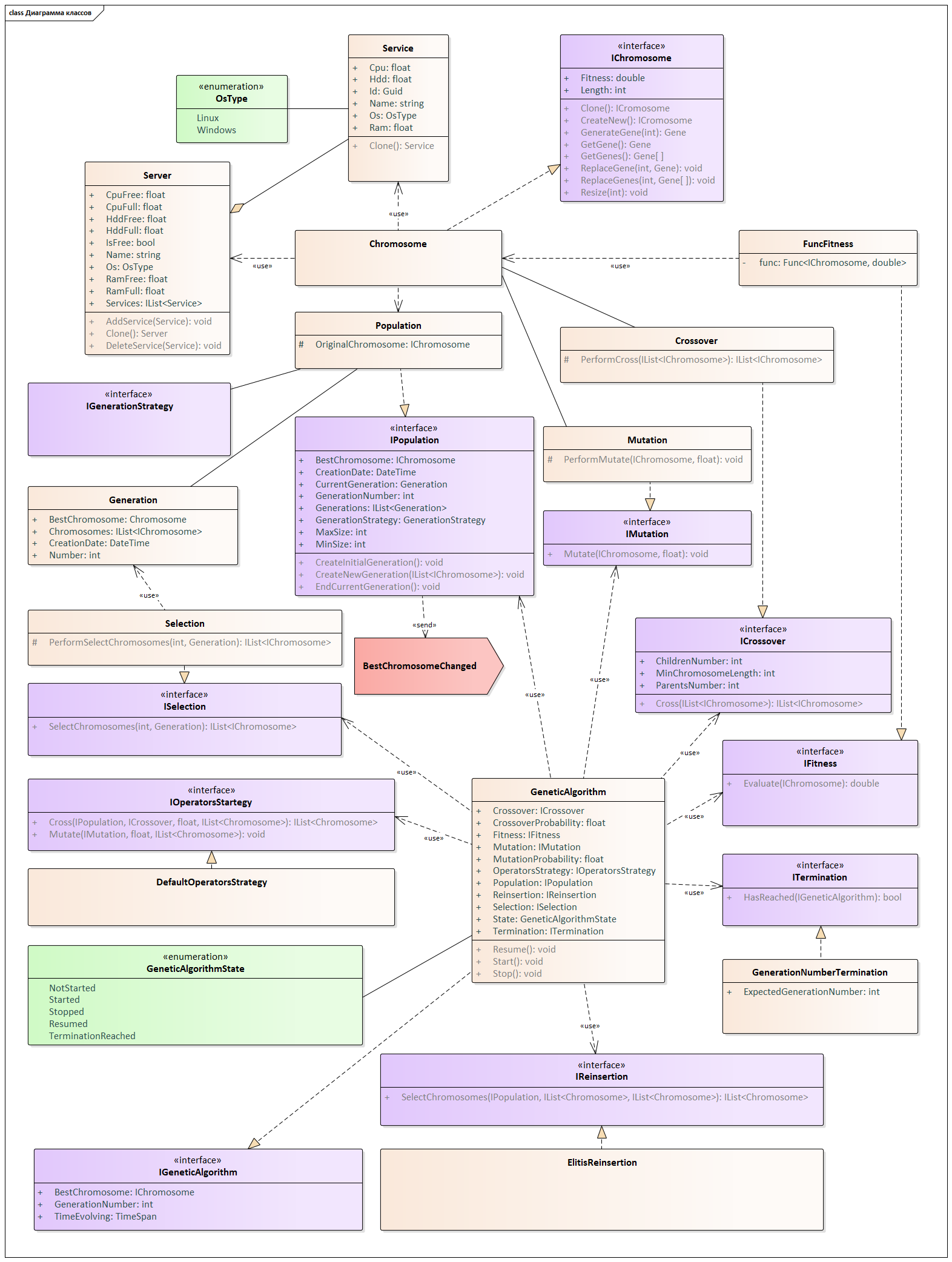
Рассмотрим связь между ограниченными контекстами сервисов сред, оптимизации и развертывания.

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.

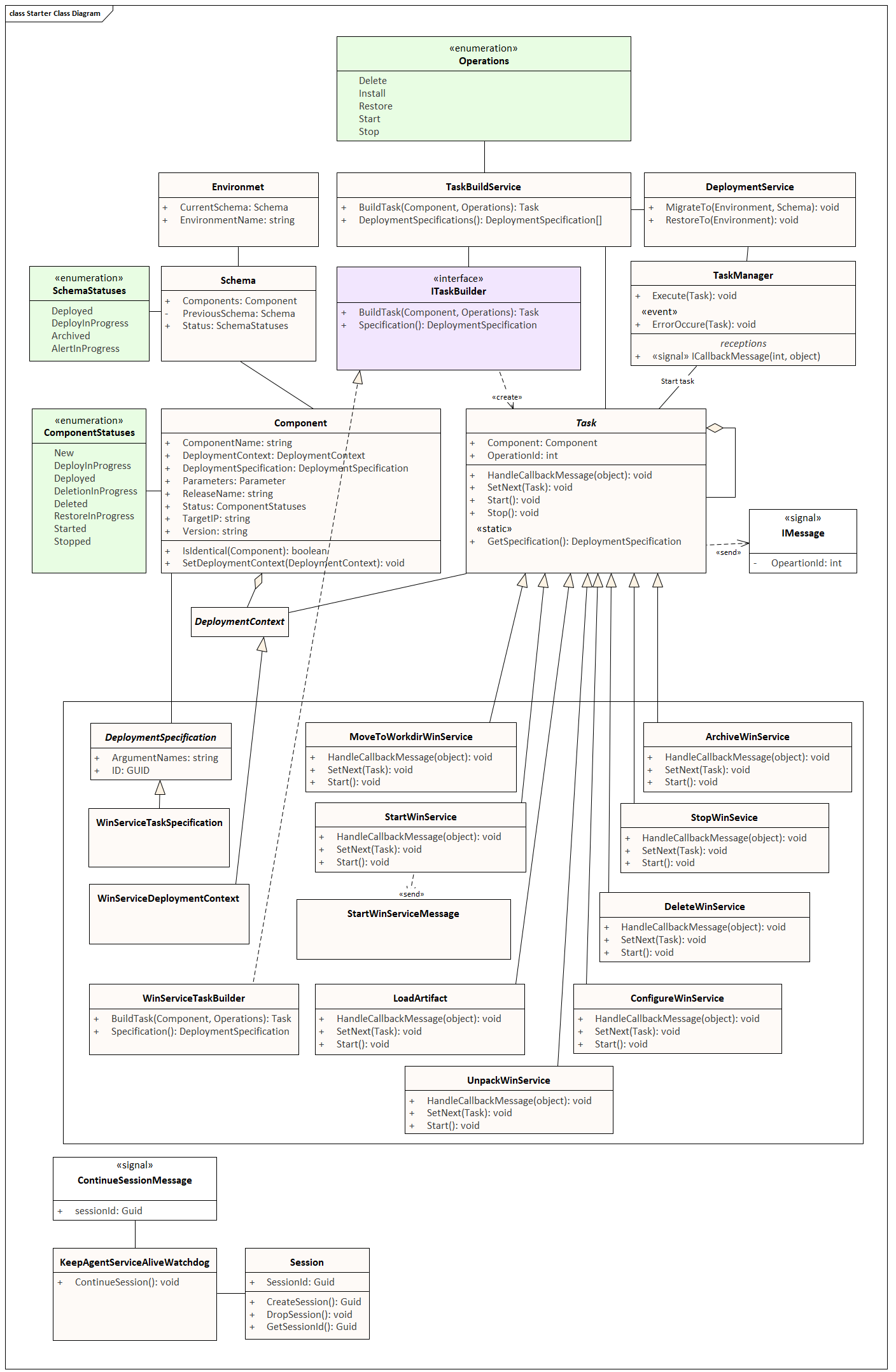
Контракты сервиса сред развертывания подключены в качестве .dll файлов в проектах сервисов оптимизации и развертывания. Сервис оптимизации получает запрос, содержащий сформированную пользователем среду развертывания. Доменные объекты сервиса оптимизации, используемые при работе с серверами и программными компонентами, представляют из себя объекты Server 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.



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». 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.



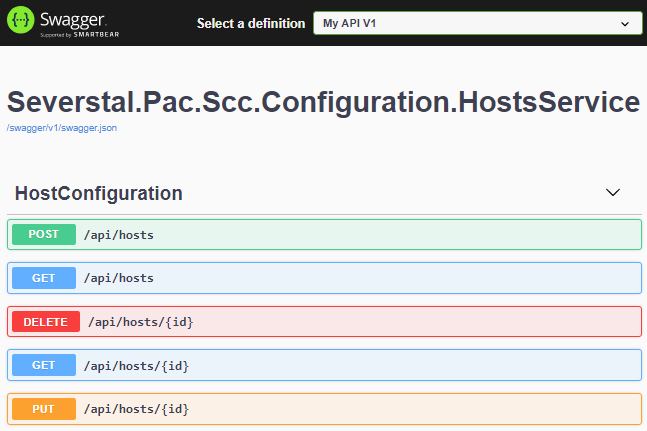
Согласно рисунку 3.6 можно судить, что границы и размеры всех ограниченных контекстов и моделей предметной области выбраны правильно, так как между этими моделями существуют несколько прочных связей, и не обязательно объединять информацию из нескольких моделей предметной области при выполнении типичных 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). Объект DTO не содержит какого-либо поведения и является лишь объектом, содержащим данные. Схема использования шаблона Data Transfer Object представлена на рисунке 3.3.



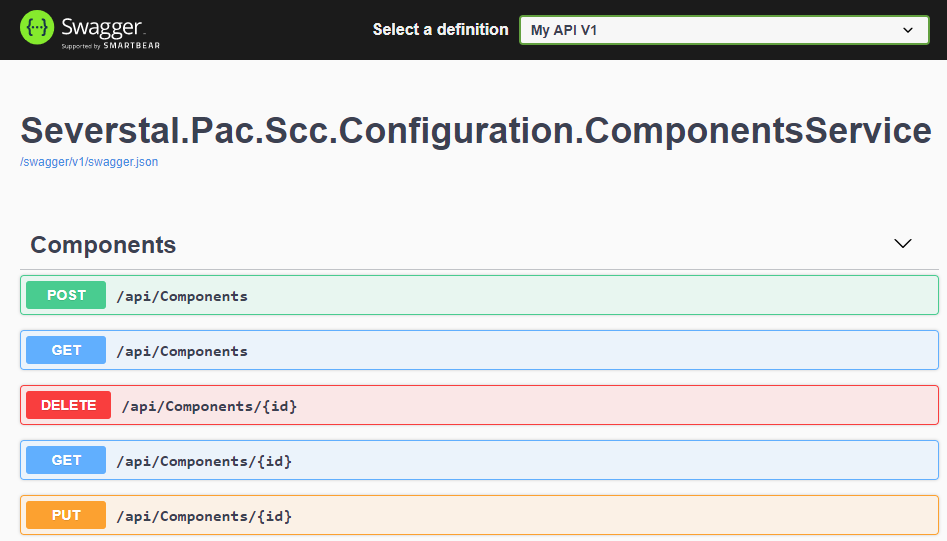
Документация по API сгенерирована с использованием фреймворка Swagger UI, позволяющего создать веб-страницу с интерактивной документацией. Скриншоты веб-страниц с документацией представлены ниже. В приложении В приведена документация по API с использованием спецификации OpenAPI, содержащую, также, описание 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 позволяет записать информацию о хосте, отредактировать, удалить хост с необходимым идентификатором, а также, получить коллекцию хостов или единственный хост по идентификатору.



Релиз-группы!

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

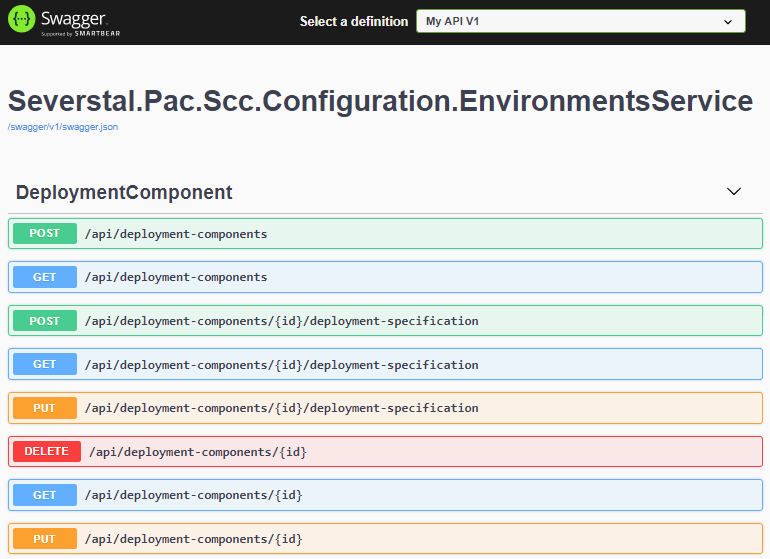


Среда развертывания, как было описано в пункте 3.1, является объектом, содержащим коллекцию целей развертывания, т.е. виртуальных серверов, на которых предполагается установка, обновление или удаление программных компонентов, и вложенные коллекции компонентов развертывания, содержащихся на каждой цели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

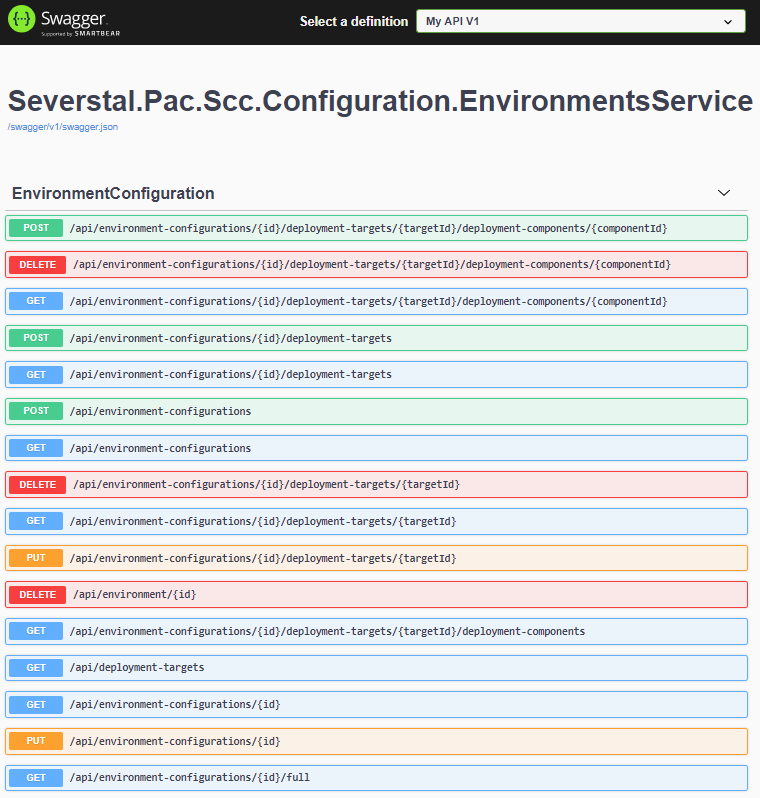
Сервис сред развертывания имеет более расширенный API и разделен на две категории:

* компонентов развертывания;
* сред развертывания.

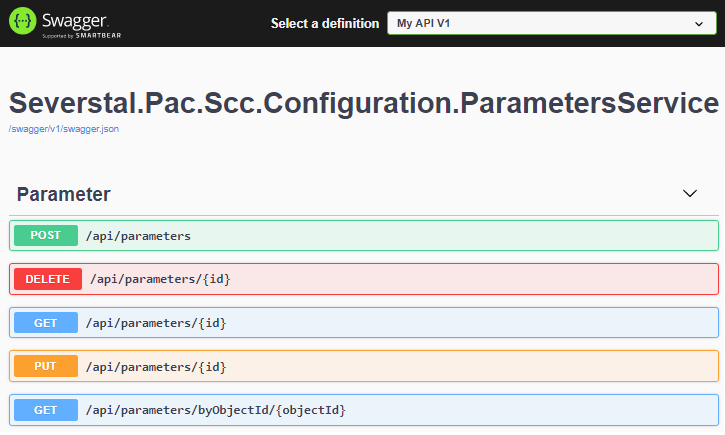
API компонентов развертывания позволяет осуществлять такие же базовые операции, как и вышеописанные сервисы, т.е. запись информации о компоненте развертывания, редактирование, удаление компонента развертывания с необходимым идентификатором, а также, получение коллекции компонентов или единственного компонента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



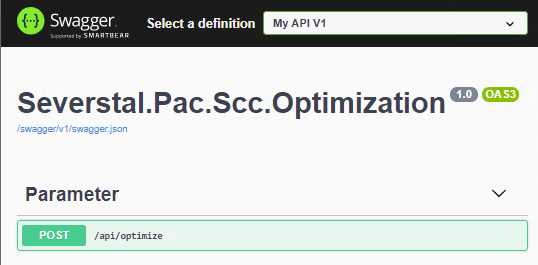
API сред развертывания, кроме базовых CRUD операций со средой развертывания, позволяет добавлять, получать, редактировать и удалять цели развертывания и компоненты развертывания, назначенные на эти цели. Также, реализована возможность прямого запроса коллекции созданных целей развертывания.



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



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



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.



## Разработка БД

В пункте 3.3 данной главы путем сравнения наиболее популярных СУБД выбрана PostgreSQL. В процессе проектирования базы данных будет применен метод нормальных форм, который состоит в сборе информации о объектах решения задачи в рамках одного отношения и последующей декомпозиции этого отношения на несколько взаимосвязанных отношений на основе процедур нормализации отношений. Процесс проектирования базы данных с использованием метода нормальных форм является итерационным и заключается в последовательном переводе отношения из первой нормальной формы в нормальную форму более высокого порядка по определенным правилам. Каждая следующая нормальная форма ограничивается определенным типом функциональных зависимостей и устранением соответствующих аномалий при выполнении операций над отношениями базой данных, а также сохранении свойств предшествующих нормальных форм. База данных будет нормализована до третьей нормальной формы. Почему? Кратко перечислим необходимые для этого требования:

1. Отношение находится в первой нормальной форме, если все его атрибуты являются простыми, все используемые домены должны содержать только скалярные значения. Не должно быть повторений строк в таблице.
2. Отношение находится во второй нормальной форме, если оно находится в первой нормальной форме и каждый не ключевой атрибут неприводимо зависит от первичного ключа.
3. Отношение находится в третьей нормальной форме, когда находится во второй нормальной форме и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа. Иначе говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы.

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 (Globally Unique Identifier) — [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 |
| Микросервис аутентификации пользователей | &&& |

Модели данных представлены на рисунках ниже. Сервисы релиз-групп, хостов, компонентов, параметров, представляющие из себя простые CRUD-сервисы используют упрощенные схемы для хранения данных. 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 3.10. Схемы баз данных CRUD-сервисов.

В микросервисе развертывания хранение данных реализовано с целью обеспечения отслеживания версий установленного программного обеспечения и возможности возврата к предыдущей версии.



Рисунок 3.10. Схема БД 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.

backend for frontend писать в шлюз. Интерфейс построен с применением библиотеки стилей AdonisUI. Ниже представлены скриншоты пользовательского интерфейса различных модулей. Пользователь может создать релиз-группу, компоненты, цели и среду развертывания. На рисунке 3.2 представлен скриншот модуля компонентов с двумя добавленными программными компонентами.

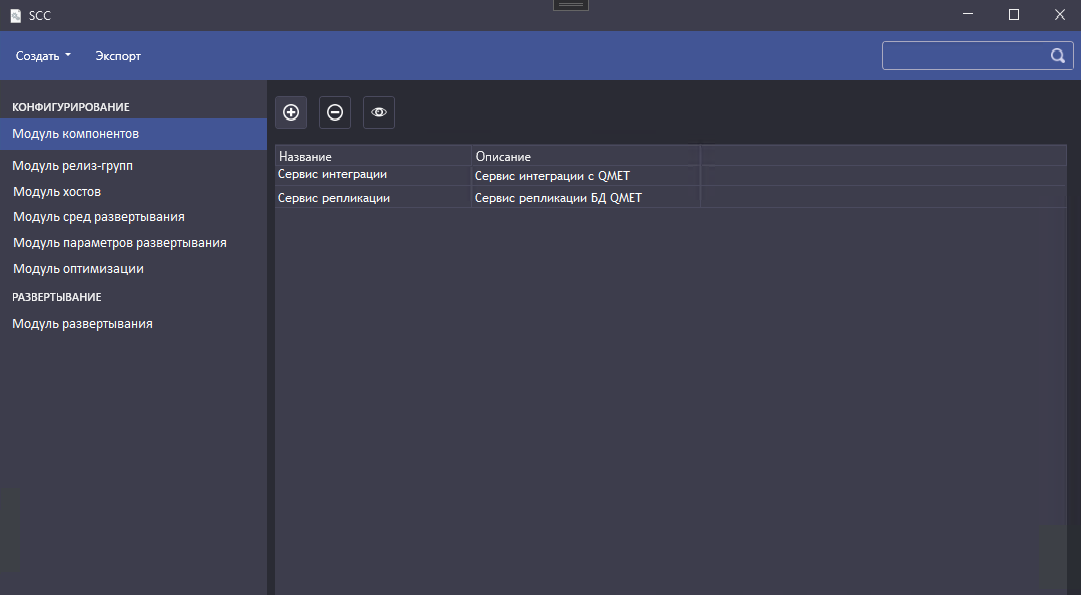


Рисунок 3.2. Модуль компонентов.

Когда пользователь создал среду развертывания, он может оптимизировать расположение программных компонентов на ней, запустив механизм оптимизации путем нажатия на кнопку «Оптимизировать». Пока скрин сделан в пейнте, сделаю для видео рабочую страницу

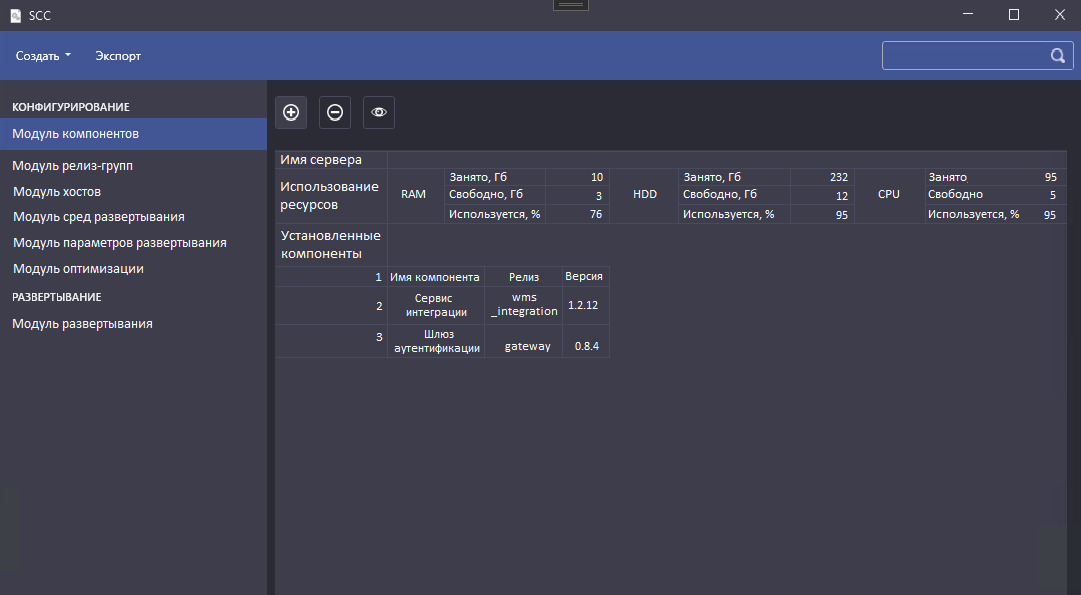


Рисунок 3.2. Модуль оптимизации.

Развернуть среду

1. Создать компоненты, хосты и т.п. - есть
2. Создать среду развертывания – открывается окно с именем среды и списком целей, которые можно отметить галочками. Редактируем цель и добавляем галочками компоненты. Все сделать просто текстом на фронтенде
3. Добавили среду. Теперь идем в сервис оптимизации. Там список сред, выбираем среду и нажимаем кнопку оптимизировать. Вылезает окно оптимизации. Можно его закрыть. Все сделать просто текстом на фронтенде
4. Идем в сервис развертывания. Там список сред. Берем среду и жмем кнопку «Оптимизировать». Все сделать просто текстом на фронтенде.

# 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) (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