# 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. Развертывание очередного релиза, таким образом, будет происходить следующим образом: пользователь выбирает необходимую среду развертывания и релиз с нужной версией. Путем сравнения версий в системе автоматически будет определено, какие релизы необходимо установить на сервер, а какие обновить, или удалить.
6. При развертывании приложения необходима также его настройка его конфигурации. Данные по настройке конфигурации необходимо также создавать и хранить в виде пар ключ-значение.
7. 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 конструктивные особенности сетевой инфраструктуры, где будет работать данное приложение. В ГДЕ? в основу имеющейся кабельной системы положена смешанная топология. Функции узлов структуры выполняет коммутационное оборудование, такое как коммутаторы и маршрутизаторы. Коммутационное оборудование соединено между собой с использованием топологии «звезда». Все компоненты вместе со смешанной топологией в части, касающейся структурированной кабельной сети, обеспечивают гибкость и надёжность СКС, а также возможность простой переконфигурации под особые проекты. 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 гипервизора Hiper V V Sphere. 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.

## 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 непрерывную 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.

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 |

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.

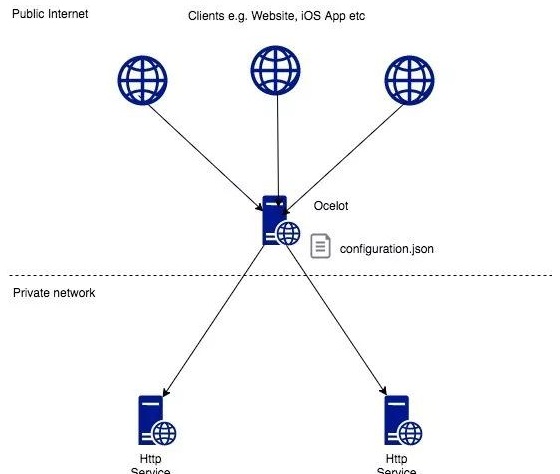


Рисунок 3.4. Функциональная схема шлюза



* 1. Функциональная схема системы развертывания приложений.

Ограничительные контексты определены и приложение разделено на микросервисы. Теперь нужно разработать домен каждой предметной области. Представим диаграммы классов с помощью унифицированного языка моделирования UML.

Разработать домен каждой предметной области.

Установить связи между разными контекстами и предотвратить перетекания контекстов друг в друга.

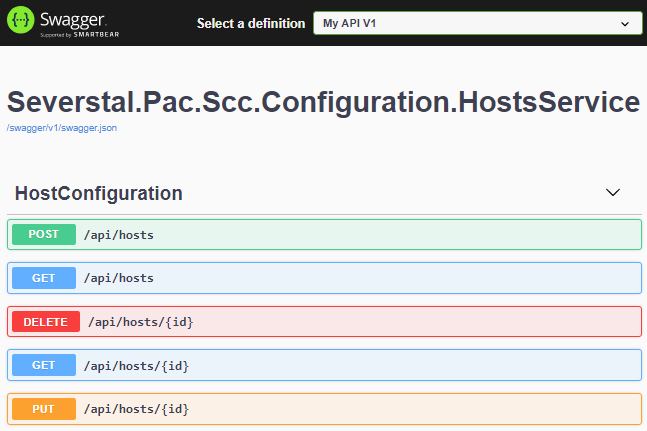
Рисовать диаграмму классов каждого сервиса и общую диаграмму

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.



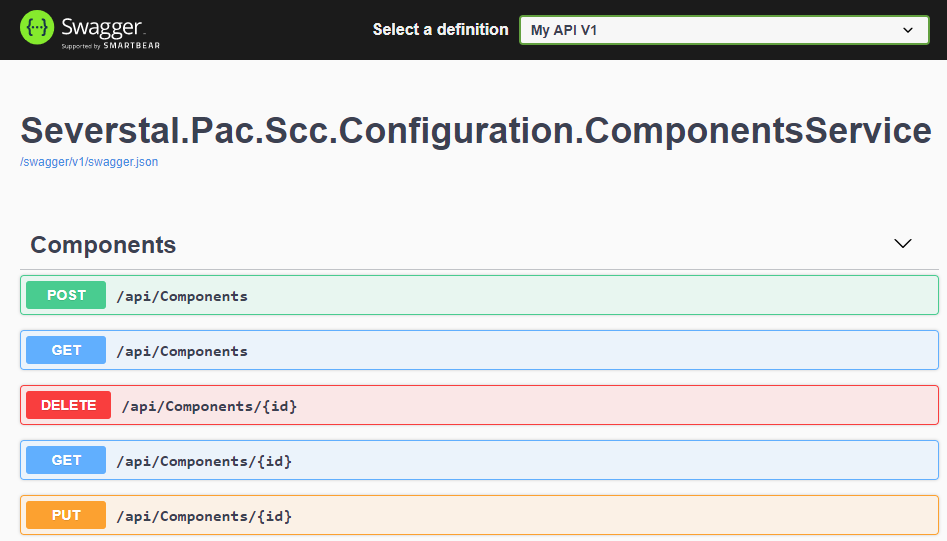
Документация по API сгенерирована с использованием фреймворка Swagger UI, позволяющего создать веб-страницу с интерактивной документацией. Скриншоты веб-страниц с документацией представлены ниже. В приложении В приведена документация по API с использованием спецификации OpenAPI, содержащую, также, описание 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.

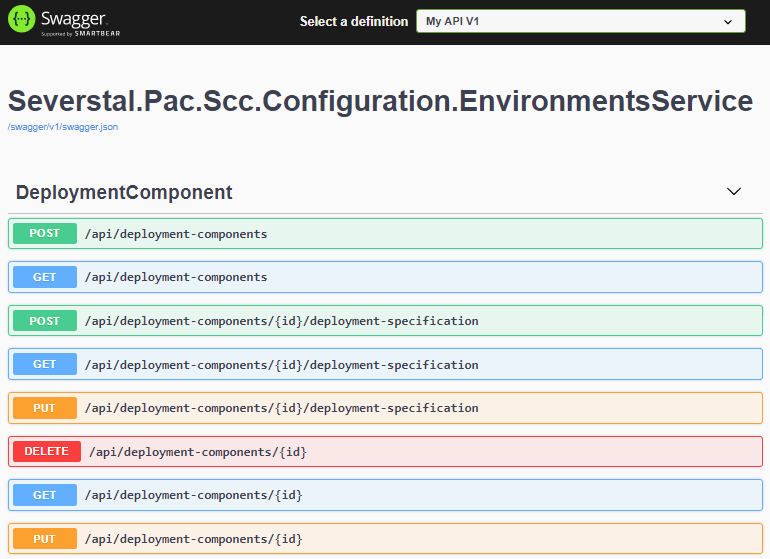


Написать это в описании среды развертыванияСреда развертывания, как было описано выше, является объектом, содержащим коллекцию целей развертывания, т.е. виртуальных серверов, на которых предполагается установка, обновление или удаление программных компонентов, и вложенные коллекции компонентов развертывания,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

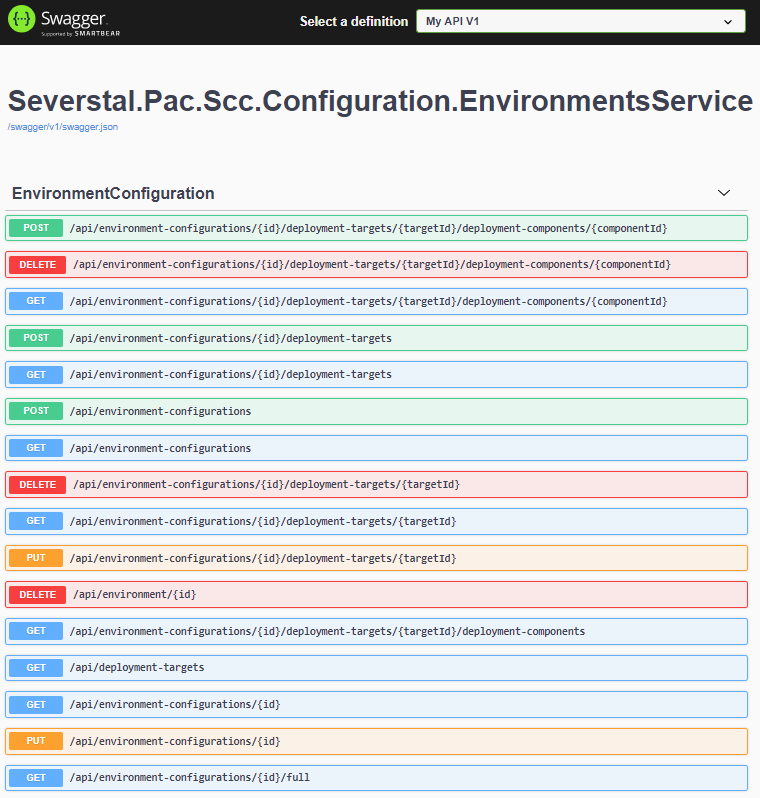
Сервис сред развертывания имеет более расширенный API и разделен на две категории:

* компонентов развертывания;
* сред развертывания.

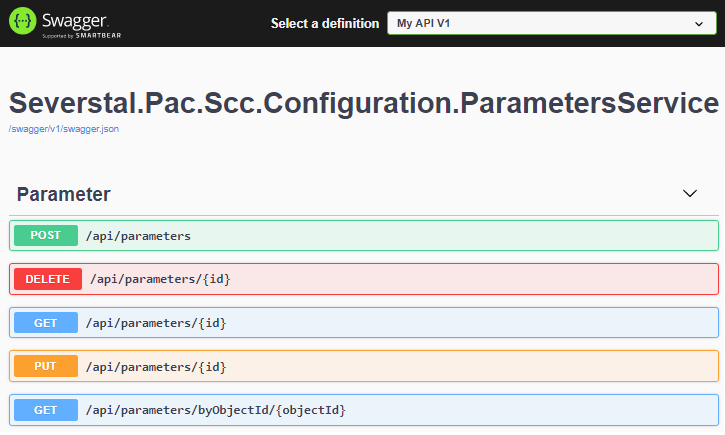
API компонентов развертывания позволяет осуществлять такие же базовые операции, как и вышеописанные сервисы, т.е. запись информации о компоненте развертывания, редактирование, удаление компонента развертывания с необходимым идентификатором, а также, получение коллекции компонентов или единственного компонента 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.



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.



Рассмотрим более подробно, какие шаблоны проектирования микросервисов применить при разработке системы развертывания приложений.

Выделим ограниченный контекст и сделаем микросервис

(CQRS). На какие слои разделено приложение – бизнес логики, доступа к данным.) ДИАГРАММЫ КОМПОНЕНТОВ, КЛАССОВ каждого микросервиса

[**Разделение ответственности за запросы команд**](https://en.wikipedia.org/wiki/Command_Query_Responsibility_Segregation)**(CQRS)**

CQRS - это архитектурный шаблон для разделения операций чтения и записи, где первое - это запрос, а второе - это команда. Команды изменяют состояние и, следовательно, приблизительно эквивалентны вызову метода для совокупных корней / объектов. Запросы читают состояние, но не изменяют его. CQRS - это производный архитектурный шаблон от шаблона проектирования, называемого разделением команд и запросов (CQS), который был придуман [Бертраном Мейером.](https://en.wikipedia.org/wiki/Bertrand_Meyer). В то время как CQRS не требует DDD, доменная структура делает различие между командами и запросами явным, исходя из концепции совокупного корня. Идея состоит в том, что данный агрегированный корень имеет метод, соответствующий команде, а обработчик команд вызывает метод для агрегированного корня. Совокупный корень отвечает за выполнение логики операции и выдачу либо количества событий, либо отказа (исключение или перечисление / количество результатов выполнения) ИЛИ (если источник событий (ES) не используется), просто изменяя его состояние для постоянная реализация, такая как ORM для записи в хранилище данных, в то время как обработчик команд отвечает за решение проблем инфраструктуры, связанных с сохранением состояния или событий совокупного корня и созданием необходимых контекстов (например, транзакций).

## 3.5 Разработка БД

Схема БД сервиса релиз-групп



Сервиса параметров



## 3.6 Разработка интерфейса

– MVVM, backend for frontend. Можно стырить схемы из книги про микросервисы

# 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*..