Содержание

ВВЕДЕНИЕ

Актуальность темы. Вычислительные мощности многих компаний мира в настоящее либо организованы в частные облака либо арендованы на публичных облачных платформах [11] [15]. Это обусловлено скоростью внедрения такой инфраструктуры, а также её дешевизной в обслуживании и аренде (если оборудование арендовано) [14]. Кроме того, публичные облачные платформы позволяют малым и средним предприятиям арендовать дорогостоящее оборудование, оплачивая лишь те ресурсы, которые были действительно использованы [13].

Несмотря на перечисленные достоинства, целиком полагаться на одну облачную платформу нельзя, так как она может выйти из строя в результате сбоя [12]. Когда такое случается, компания может потерять доступ ко всей своей инфраструктуре, в результате теряя деньги. В связи с этим, сервисы, позволяющие организовать систему из нескольких вычислительных облачных платформ (для резервирования от непредвиденных сбоев или для скорости доступа из разных точек мира), в настоящее время активно исследуются и даже патентуются различными организациями и исследователями [18].

Кроме этого, у таких компаний может возникнуть необходимость мигрировать всю инфраструктуру с одной облачной платформы на другую [21]. Такая необходимость может быть вызвана рядом причин, среди которых выделяют:

- ожидаемое снижение стоимости облачной инфраструктуры;
- возросшие потребности в вычислительных мощностях, которые не могут быть удовлетворены текущими решениями.

Для таких случаев активно исследуются и разрабатываются программные модули, применение которых не привязывает компанию к конкретной реализации облачной платформы, а позволяет миграцию инфраструктуры в другое облако без доработок программного обеспечения.

С учётом тенденций, описанных выше, сервис управления виртуализированной инфраструктурой является актуальным и востребованным реше-

нием.

Цель и задачи исследования. Целью работы является упрощение реализации компонентов управления ресурсами систем с различными технологиями виртуализации инфраструктуры. Для достижения поставленной цели необходимо решить ряд **задач**.

1. Проанализировать предметную область и определить типовые подходы к управлению виртуализированной инфраструктурой.
2. Выявить недостатки существующих решений; сформулировать новый подход к управлению виртуализированной инфраструктурой.
3. На основе сформулированного подхода разработать сервис управления ресурсами систем с различными технологиями виртуализации инфраструктуры.

Область исследования. Проведённое исследование виртуализированной инфраструктуры полностью соответствует специальности «Программная инженерия», а содержание выпускной квалификационной работы — техническому заданию.

Объектом исследования являются платформы виртуализации инфраструктуры.

Предметом исследования является управление виртуализированной инфраструктурой с поддержкой нескольких существующих реализаций платформ виртуализации инфраструктуры.

Теоретическую основу исследования составляют научные труды отечественных и зарубежных авторов по компьютерным технологиям.

Методологическую основу исследования составляют метод обобщения, общенаучный метод абстрагирования, а так же эксперимент.

Практическая значимость исследования. Разработанный сервис управления может быть использован как в новых, так и в существующих облачных системах. Такой сервис позволяет осуществлять мониторинг и контроль разнородных платформ облачных вычислений при использовании единого

унифицированного программного интерфейса (API). Это позволяет использовать одни и те же системы мониторинга и управления с разными платформами облачных вычислений, а так же позволяет заменять одну виртуализированную инфраструктуру на другую без доработок в существующих программных модулях.

1 УПРАВЛЕНИЕ ВИРТУАЛИЗИРОВАННОЙ ИНФРАСТРУКТУРОЙ

1.1 Обзор предметной области

Для организации своей вычислительной инфраструктуры компании в настоящее время всё чаще выбирают решения, основанные на так называемых облачных платформах [16]. Такие решения имеют ряд достоинств, среди которых для данной работы наиболее важно следующее: загрузка аппаратного обеспечения может быть выше при организации инфраструктуры в облачное решение, чем при традиционной организации [10]. Загрузка аппаратного обеспечения показывает какое количество аппаратных ресурсов используется относительно общего их количества [17]. Чем выше загрузка, тем меньше аппаратных ресурсов работает вхолостую или простаивает. Как холостая работа, так и простой ведут к прямым убыткам, в связи с чем более высокая загрузка обеспечивает финансовую выгоду [19].

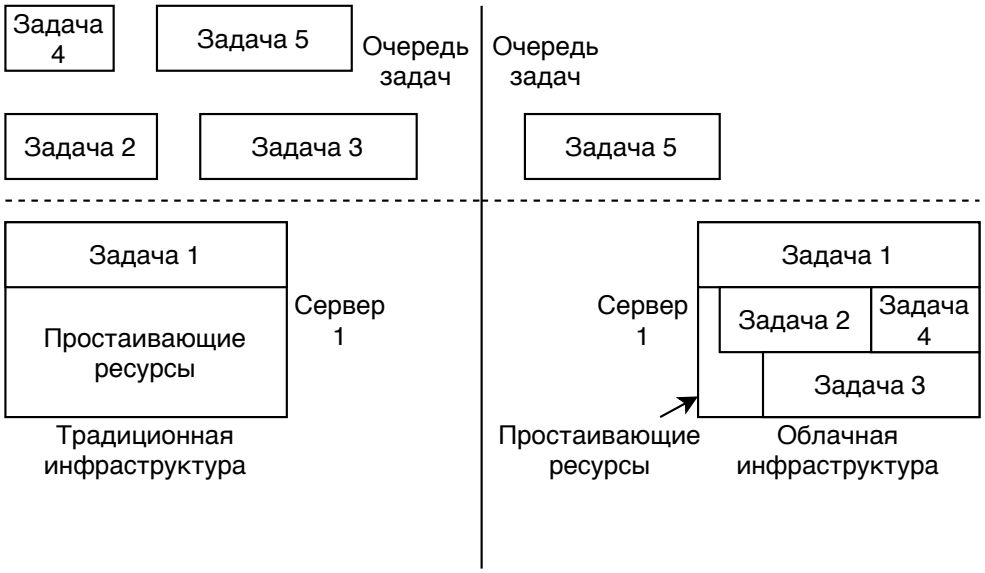
В облачных инфраструктурах повышение загрузки достигается с помощью разделения вычислительных ресурсов одного и того же сервера между несколькими задачами [20]. В традиционной инфраструктуре такое разделение не применяется по причинам безопасности, а также из-за возможного захвата всех аппаратных ресурсов одной задачей и последующим простаиванием других [22]. Таким образом, в традиционной архитектуре для выполнения того же объёма работы нужно больше аппаратных ресурсов, чем в облачной, как проиллюстрировано на рис. 1 на стр. 9.

В облачных решениях захват всех аппаратных ресурсов невозможен вследствие ограниченного выделения ресурсов каждой из задач [23]. В общем случае, это способы ограничения ресурсов делятся на два типа [9]:

- решения на базе виртуальных машин;
- решения на базе контейнеров.

В обоих подходах количество ресурсов, доступных задаче, управляются с помощью:

Рис. 1: Сравнение облачной и традиционной организации инфраструктуры



- количества таких виртуальных машин или контейнеров;

  Этот способ называется ”масштабирование”.

- количества ресурсов, доступных одной виртуальной машине или одному контейнеру.

Масштабирование в настоящее время применяется чаще [24] по ряду причин, среди которых:

- возможность выделения на одно приложение (задачу) несколько серверов за счёт запуска виртуальных машин или контейнеров на каждом из них [28].

- Более высокая загрузка оборудования, так как в случае отсутствия нагрузки на приложение, будет простаивать только то количество аппаратных ресурсов, которое выделено одной виртуальной машине или контейнеру приложения, другие ресурсы при этом будут высвобождены [25]. В случае масштабирования есть возможность выделить одной виртуальной машине или контейнеру минимально необходимое количество ресурсов для обработки минимальной нагрузки на приложение, в то время как при выделении ресурсов, достаточных для обработки пиковой нагрузки, будет наблюдаться простой в обычном режиме работы [26].

Таким образом, существует задача масштабирования приложений. Для решения этой задачи существуют технологии, имеющие общее название ”автомасштабирование” [4]. При помощи таких технологий осуществляется автоматическое масштабирование в зависимости от текущей нагрузки на приложение или других факторов.

Во многие платформы облачных вычислений сервисы автомасштабирования уже встроены [32], однако существуют сервисы автомасштабирования, являющиеся внешними по отношению к платформе. Причины появления таких сервисов могут быть разными:

- отсутствие решений автомасштабирования в используемой облачной платформе [27];
- более эффективная реализация автомасштабирования у внешнего решения, чем у встроенного. Это может быть обусловлено как спецификой конкретного приложения, на которое платформа не рассчитана, так и недостаточной эффективностью встроенного решения [29].

На практике зачастую внешние сервисы автомасштабирования спроектированы и разработаны под одну конкретную платформу облачных вычислений и не могут взаимодействовать с другими [30].

В случае, если система состоит из нескольких разных облачных платформ или в системе осуществляется миграция с одной облачной платформы на другую, появляется необходимость в существенной доработке сервиса автомасштабирования, если он был спроектирован лишь под одну платформу [31]. Далее в этой главе будет представлен обзор научных работ, статей и патентов, посвящённых решению описанной проблемы.

Программируемый инфраструктурный шлюз для обеспечения работы гибридных облачных сервисов

Программируемый инфраструктурный шлюз, используемый для обеспечения работы гибридных облачных сервисов в сетевом окружении[1] разработан с целью обеспечения организации взаимодействия частей гибридного облачного окружения.

Под гибридным облачным окружением в патенте понимается облачное окружение, где часть вычислительных ресурсов являются внутренними, а управление ими осуществляется самой организацией (или частным лицом). При этом другая часть вычислительных ресурсов управляется сторонней организацией (или сторонним частным лицом) и является внешней. В качестве примера приводится использование публичного облачного сервиса Amazon S3™ для хранения архивных данных компании и внутренней инфраструктуры организации для хранения корпоративных данных по текущим операциям.

Инфраструктурный шлюз, как и сервис управления, должен решать задачу предоставления унифицированного интерфейса к различным реализациям облачных окружений. Отсюда следует, что разработка описываемого инфраструктурного шлюза, как и сервиса управления виртуализированной инфраструктурой, предполагает разработку так называемых облачных адаптеров. Под облачным адаптером здесь понимается программный модуль, управляющий передачей данных и команд из частного облака (внутренних ресурсов) в публичную (внешнюю) инфраструктуру.

Авторы программируемого инфраструктурного шлюза не решали задачу мониторинга используемых приложениями ресурсов в реальном времени. Сервис управления, в свою очередь, должен решать такую задачу.

Сервис управления виртуализированной инфраструктурой, в отличие от описываемого шлюза, должен работать не только с гибридными облачными сервисами, но и в инфраструктуре, состоящей только из частного облака или только из публичного. При этом сервис управления не позволяет передавать данные между двумя частями инфраструктуры.

Метод оркестровки гибридными облачными сервисами

Метод оркестровки гибридными облачными сервисами[2] позволяет при помощи одного унифицированного интерфейса взаимодействовать с гибридным облачным окружением, состоящем из нескольких разнородных платформ облачных вычислений. Таким образом, этот метод, как и сервис управления виртуализированной инфраструктурой, предоставляет единый унифи-

цированный интерфейс, но в случае сервиса управления интерфейс скрывает всегда одну платформу облачных вычислений. Метод оркестровки предполагает создание интерфейса, скрывающего несколько разнородных реализаций виртуализированной инфраструктуры, работающих одновременно и формирующих таким образом единое гибридное облачное окружение.

С учётом вышесказанного, можно заметить, что и метод оркестровки, и сервис управления подразумевают возможность работы с различными реализациями виртуализированной инфраструктуры. Сюда включаются сразу две задачи:

1. Сбор статистики использования ресурсов.
2. Изменение количества выделенных приложению ресурсов.

При этом метод оркестровки не ставит перед собой именно эти задачи, он ставит перед собой задачу предоставления ”всеобъемлющего” API, который, будет решать эти задачи, если их решает каждая из платформ в гибридном облачном окружении.

Задача предоставления возможности адаптации дополнительной реализации виртуализированной инфраструктуры к единому унифицированному API ставится как перед методом оркестровки, так и перед сервисом управления виртуализированной инфраструктурой.

Программная архитектура сервиса мониторинга гибридных платформ облачных вычислений

Программная архитектура сервиса мониторинга гибридных платформ облачных вычислений [3] была спроектирована для решения двух основных задач:

- Мониторинг в режиме реального времени. Это может быть использовано в целях так называемого превентивного или планового обслуживания, то есть для проведения ряда мероприятий, необходимого для избежания неожиданных сбоев.
- Постфактум мониторинг состояния системы. С помощью такого мониторинга можно обнаружить проблемы в работе системы или даже про-

блемы, связанные с безопасностью системы.

Гибридность целевой платформы (платформы под мониторингом) заключается в том, что она может быть заменена на другую без необходимости вносить изменения в сам сервис мониторинга. Таким образом, задача предоставления унифицированного интерфейса решается как сервисом мониторинга, так и сервисом управления.

Сервис управления виртуализированной инфраструктурой, как и сервис мониторинга, решает задачу сбора статистики в реальном времени. При этом сервис мониторинга решает дополнительную задачу постфактум мониторинга, но такой задачи не стоит перед сервисом управления.

Перед сервисом мониторинга не стоит задачи предоставления интерфейса, который позволяет изменять количество выделенных каждому из приложений ресурсов. Сервис управления, в свою очередь, должен решать такую задачу и позволять не только узнать сколько ресурсов используется сейчас, но и высвобождать неиспользуемые ресурсы, а также выделять дополнительные.

Портативный сервис автомасштабирования для управления различными масштабируемыми облачными платоформами

Портативный сервис автомасштабирования для управления различными масштабируемыми облачными платоформами[4] был разработан для автоматического масштабирования разнородных реализаций виртуализированной инфраструктуры. Этот сервис не предоставляет внешний программный интерфейс, только интерфейс для администратора системы с доступом к параметрам сервиса, а так же собранной статистике.

Рассматриваемый сервис автомасштабирования является примером прикладного назначения сервиса управления, потому что сервис управления решает задачи двух модулей сервиса автомасштабирования:

1. Provision Manager - модуль, позволяющий изменять количество доступных каждому запущенному приложению ресурсов.

2. Monitoring Engine - модуль, предоставляющий статистику использования ресурсов, собранную платформой облачных вычислений.

Таким образом, сервис автомасштабирования решает задачи, поставленные перед сервисом управления, но не предоставляет внешний программный интерфейс. Напротив, сервис автомасштабирования использует унифицированный интерфейс самостоятельно и не позволяет применять другие алгоритмы автомасштабирования, кроме заложенных в него.

Федерация облачных сервисов с использованием прокси-слоя виртуального API в распределённом облачном окружении

Известен способ объединения одной из реализаций виртуализированной инфраструктуры OpenStack в федерацию с другими реализациями с помощью прокси-слоя виртуального API, который представляет другие облака как псевдо-зону доступности в OpenStack[5]. Используя этот способ, можно создать систему из нескольких облачных платформ, объединённую в OpenStack. Это позволяет использовать модули управления, мониторинга и так далее из OpenStack с другими платформами без необходимости их изменения.

Таким образом, эта работа решает все задачи сервиса управления виртуализированной инфраструктурой:

1. задача предоставления единого интерфейса решена предоставлением интерфейса OpenStack;
2. задача предоставления возможности добавлять дополнительные платформы облачных вычислений решена при помощи псевдо-зон доступности в OpenStack, куда можно добавить необходимую облачную платформу;
3. задача сбора статистики решена с помощью сбора статистики в OpenStack;
4. задача изменения количества выделенных приложению ресурсов так же решается с помощью модуля управления OpenStack.

Как видно из этого перечисления, все задачи решены, но решены при помощи использования дополнительного облачного окружения, которое будет

избыточно в той прикладной задаче, для которой предлагается использовать сервис управления виртуализированной инфраструктуры. Дополнительная облачная платформа внесёт существенные задержки и сложность в итоговую систему, а так же создаст дополнительную нагрузку на вычислительные мощности. Иными словами, этот способ рекомендуется использовать только в ситуациях, когда OpenStack уже является частью системы, но добавлять его только ради получения единого унифицированного интерфейса будет избыточно.

Программа для управления гибридными облачными сервисами и облачными хранилищами ”ACCENTOS”

Программа с коммерческим названием ”AccentOS”, запатентованная в Российской Федерации[6], предназначена для управления гибридными облачными сервисами и облачными хранилищами. Согласно описания патента, в данной программе реализовано:

- управление физическими серверами, входящими в облачную инфраструктуру;
- мониторинг состояния серверов в реальном времени; автоматическое восстановление облачных сервисов в случае отказа или аварии;
- унификацию интерфейсов управления для публичных, частных и смешанных облачных сервисов и облачных хранилищ;
- получение статистических данных об основных характеристиках производительности серверов;
- интефацию с системами мониторинга и резервного копирования сторонних производителей;
- предоставление расширенной поддержки драйверов для взаимодействия с серверами в облачной инфраструктуре;
- предоставление расширенной поддержки динамического управления серверами в облачной инфраструктуре.

Перечень не является исчерпывающим и полным и содержит лишь те пункты, которые являются важными в данной работе.

Основываясь на этом перечне, можно сделать вывод о том, что в данной программе реализованы все требования к сервису управления виртуализированной инфраструктурой, кроме одного: предоставление унифицированного программного интерфейса (API). Действительно, согласно патенту, программа ”AccentOS” предоставляет интерфейс администратору, но не предусматривает возможности интеграции внешних систем. Таким образом, данную программу нельзя считать сервисом управления. Наоборот, эта программа могла бы быть реализована с использованием такого сервиса, но, в отсутствие общедоступной реализации, программа содержит аналог такого сервиса как один из модулей.

Поддержка мульти-облачных систем в нативных облачных приложениях используя платформы эластичных контейнеров

Существует решение[7], позволяющее организовать облачную систему на основе разнородных поставщиков услуг облачных сервисов и разнородных платформ эластичных контейнеров. В качестве примера организации такой облачной системы и доказательств работоспособности решения в статье приведена система следующей конфигурации:

- Поставщики облачной инфраструктуры:
  - Google Cloud Engine
  - Amazon AWS
  - OpenStack (в датацентре университета исследователя)
- Платформы эластичных контейнеров:
  - Kubernetes
  - Docker Swarm

Само решение состоит из утилиты командной строки, принимающей на вход *желаемое* и *текущее* состояния всей облачной системы. Затем утилита

производит необходимые для достижения *желаемого* состояния мероприятия. Таким образом, утилита решает задачи сервиса управления виртуализированной инфраструктуры:

1. предоставление унифицированного программного интерфейса (API);
2. предоставить возможность изменять количество выделенных каждому из приложений ресурсов.

В работе так же освещена проблема интеграции дополнительных реализаций виртуализированной инфраструктуры. В частности, для интеграции дополнительной платформы облачных вычислений или для интеграции дополнительного поставщика облачной инфраструктуры необходимо реализовать адаптер API дополнительно интегрируемого модуля к API всей системы. Таким образом, задача предоставления возможности адаптации новых модулей решена.

В работе не освещён вопрос мониторинга *текущего* состояния системы и, как следствие, задача сбора статистики использования приложениями вычислительных ресурсов, не решена.

Поддержка программируемых правил автомасштабирования для контейнеров и виртуальных машин на облачных платформах

Policy Keeper[8], осуществляющий автомасштабирование контейнеров и виртуальных машин на различных облачных платформах, был спроектирован с поддержкой раздельного автомасштабирования узлов (виртуальных машин) системы и контейнеров системы. Иными словами, Policy Keeper позволяет изменять количество контейнеров приложения без изменения количества узлов приложения и наоборот. В круг задач, решаемых таким сервисом автомасштабирования, входят:

1. осуществлять мониторинг набора показателей использования вычислительных ресурсов по каждому из приложений;
2. позволять администратору изменять набор показателей, по которым собирается статистика, для каждого приложения;

3. осуществлять автомасштабирование каждого из приложений по заданному администратором набору правил;
4. позволять создавать условия на основе статистики использования вычислительных ресурсов в каждом из правил автомасштабирования;
5. позволять адаптировать сервис автомасштабирования к облачным системам, построенным на базе других реализаций виртуализированной инфраструктуры.

Из перечисленных выше требований к Policy Keeper, можно сделать следующие выводы:

1. требование к сервису управления виртуализированной инфраструктурой об адаптации к новым облачным платформам здесь реализовано в полной мере;
2. требование о предоставлении унифицированного интерфейса здесь не реализовано, так как Policy Keeper не предоставляет доступ к собранной им информации, он позволяет только создавать правила по которым он осуществляет автомасштабирование системы;
3. требование о сборе статистики здесь реализовано и Policy Keeper использует эту статистику для принятия решений по заданным правилам;
4. требование об изменении количества выделенных ресурсов здесь так же реализовано, Policy Keeper умеет включать и отключать дополнительные узлы и контейнеры в системе.

Таким образом, Policy Keeper реализует часть требований к сервису управления, но не все из них. Более того, нужно отметить, что сервис управления мог бы выступать как часть (модуль) системы автомасштабирования по программируемым правилам.

1.2 Постановка задачи

В результате обзора предметной области было выявлено, что в настоящее время не существует решений, удовлетворяющих всем предъявляемым тре-

бованиям. В связи с этим, в данной работе будет предложено решение на базе введения в систему промежуточного сервиса, называемого сервисом управления виртуализированной инфраструктурой.

”Виртуализированная инфраструктура” — специальный обобщающий термин, введённый в этой работе для объединения двух способов организации облачной инфраструктуры:

- способы на базе контейнерной организации приложений;
- способы с использованием виртуальных машин для организации приложений.

Виртуализированной инфраструктурой в данной работе называется такая инфраструктура, которая:

1. Управляет вычислительными мощностями.
2. Предоставляет возможность запускать программные приложения на вычислительных мощностях этой инфраструктуры.
3. Позволяет выделять каждому из запущенных программных приложений строго определённое количество ресурсов. Количество ресурсов при этом необязательно кратно количеству аппаратных ресурсов в этой инфраструктуре.
4. Предоставляет возможность изменять количество выделенных ресурсов в случае, если потребности приложения изменились.
5. Позволяет запускать дополнительные программные приложения на свободных вычислительных мощностях.

Сервис управления виртуализированной инфраструктурой будет решать задачу взаимодействия с виртуализированной инфраструктурой, в то время как сервисы автомасштабирования будут принимать решения о необходимости осуществления взаимодействия. Таким образом, сервис автомасштабирования получает возможность управлять любой виртуализированной инфраструктурой без каких-либо доработок, потому что задача адаптации новой платформы облачных вычислений будет решена сервисом управления.

Кроме управления, для осуществления автомасштабирования необходимо знать какое количество ресурсов сейчас простаивает[32]. Для этого сервис управления должен предоставлять возможность сбора статистики использования вычислительных мощностей каждым из приложений[33]. На основе этой статистики и будет осуществляться автомасштабирование.

Более формально, задача управления виртуализированной инфраструктурой включает в себя две основных подзадачи:

1. Предоставление доступа к статистике использования вычислительных мощностей каждым из приложений (виртуальных машин).
2. Предоставление возможности изменять количество выделенных вычислительных мощностей каждому из приложений (виртуальных машин).

2 ТРЕБОВАНИЯ К РЕАЛИЗАЦИИ СЕРВИСА УПРАВЛЕНИЯ

К разрабатываемому сервису управления виртуализированной инфраструктурой предъявлен следующий список функциональных и нефункциональных требований:

1. Интегрируемость с Kubernetes и OpenNebula. Требование интегрируемости именно с этими реализациями виртуализированной инфраструктуры обусловлено существенной разницей в их организации и, соответственно, программных интерфейсах (API), которые они предоставляют. Такая существенная разница позволит продемонстрировать преимущества использования сервиса управления наиболее полно.
2. Адаптируемость к дополнительным реализациям виртуализированной инфраструктуры с использованием программных адаптеров. Это нефункциональное требование объясняется необходимостью поддержки различных платформ облачных вычислений, в том числе не существующих на сегодняшний день, в связи с чем должна быть предусмотрена возможность адаптировать сервис к дополнительным платформам.
3. Возможность описать и сохранить параметры подключения к виртуализированной инфраструктуре, включая параметры конкретного приложения. Это функциональное требование возникает вследствие необходимости поддержки гибридных облачных окружений, состоящих из нескольких облачных платформ и нескольких приложений.
4. Возможность запросить список текущих приложений, то есть список параметров подключения к каждому из приложений.
5. Возможность обновить параметры подключения для каждого из приложений.
6. Возможность удалить параметры подключения для каждого из приложений.
7. Возможность запросить информацию по приложению, включающую в

себя:

- текущее количество ВМ (виртуальных машин) или контейнеров, выделенное данному приложению в данной платформе облачных вычислений;
- имя каждого из контейнеров или ВМ в платформе облачных вычислений;
- текущую загрузку центрального процессора в каждом из контейнеров или ВМ;
- текущее количество используемой каждой ВМ или контейнером оперативной памяти (RAM).

Это требование является частью задачи управления, решаемой сервисом управления, а именно подзадача сбора статистики использования вычислительных ресурсов (мониторинга).

8. Возможность осуществления масштабирования, то есть возможность изменять текущее количество выделенных контейнеров или ВМ для каждого из приложений. Это требование является частью задачи управления, решаемой сервисом управления, а именно подзадача изменения количества выделенных вычислительных мощностей.
9. Предоставление единого унифицированного программного интерфейса (API), не зависящего от типа управляемых виртуализированных инфраструктур, а так же их количества.

3 РАЗРАБОТКА СЕРВИСА УПРАВЛЕНИЯ ВИРТУАЛИЗИРОВАННОЙ ИНФРАСТРУКТУРОЙ

3.1 Описание предложенного решения

Для решения задачи, поставленной в разделе 1.2 на стр. 18, с учётом требований в разделе 2 на стр. 21 была спроектирована архитектура решения, состоящая из следующих компонентов:

1. СУБД — конфигурацию приложений необходимо надёжно хранить для того, чтобы у администраторов системы не было необходимости заново конфигурировать сервис управления в случае аварий.
2. Веб-сервис — сервис, осуществляющий взаимодействие с внешними компонентами, в частности с сервисами автомасштабирования. Он необходим для предоставления API этим сервисам, а так же для взаимодействия с СУБД.
3. Клиентская библиотека — программный модуль (библиотека), который будет подключаться к веб-сервису. Данная библиотека будет содержать API для взаимодействия с каждой из поддерживаемых платформами облачных вычислений, а также предоставлять унифицированный интерфейс для взаимодействия, который будет использоваться веб-сервисом для обработки запросов.

Пример взаимодействия компонентов представлен в виде диаграммы последовательности на рис. 2 на стр. 24

Приложение и внешний компонент не являются частью разрабатываемого сервиса. Напротив, сервис служит промежуточным звеном между ними для организации независимого взаимодействия.

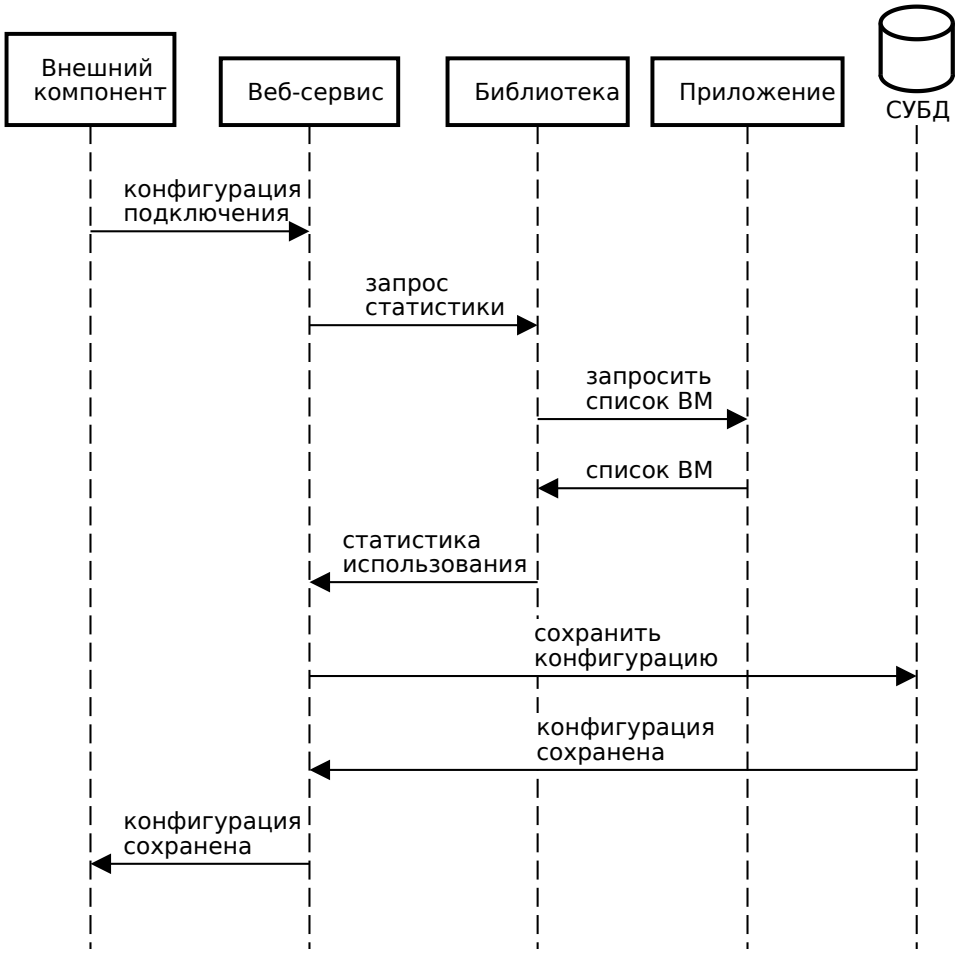


Рис. 2: Диаграмма последовательности взаимодействия компонентов

3.2 Описание схемы БД

Организация базы данных, используемой сервисом, представлена на рис. 3 на стр. 25.

Таблица ”application_entity” содержит записи с информацией об известных системе приложениях. Описание полей сущности представлено в табл. 1 на стр. 26.

Таблица ”kubernetes_config_entity” содержит записи с информацией о параметрах подключения к платформе ”Kubernetes”, а так же о конкретном приложении под управлением данной платформы. Описание полей сущности представлено в табл. 2 на стр. 26.

Таблица ”open_nebula_config_entity” содержит записи с информацией о параметрах подключения к платформе ”OpenNebula”, а так же о конкретном приложении под управлением данной платформы. Описание полей сущности представлено в табл. 3 на стр. 27.

Рис. 3: Схема базы данных

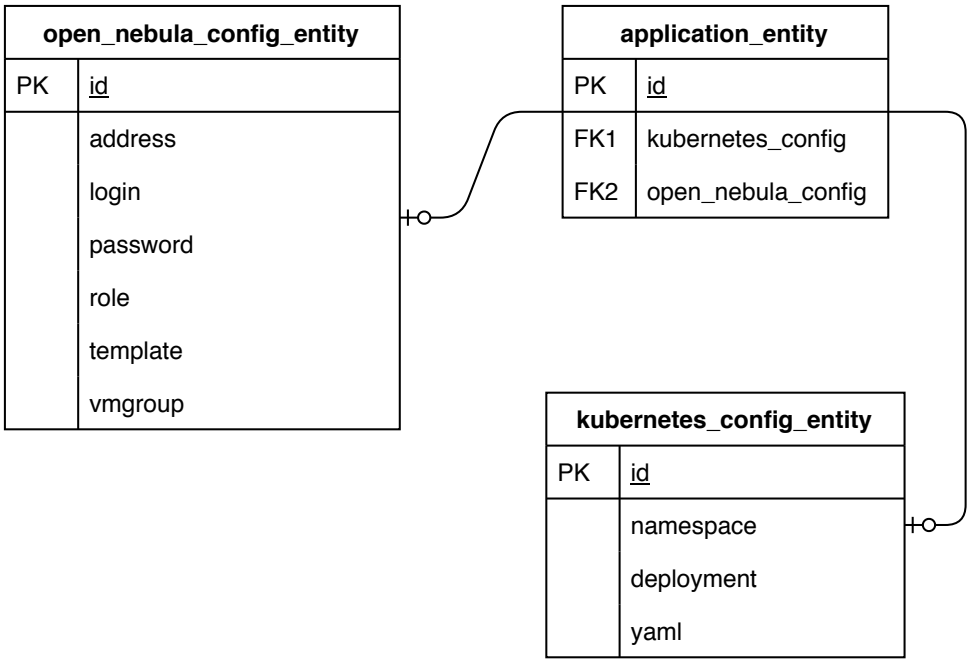


Таблица 1: Описание полей сущности ”application_entity”

Имя	**Описание**
id	Первичный ключ сущности, уникальное число для идентификации записи.
kubernetes_config	Первичный ключ сущности ”kubernetes_config_entity” для связи приложения и его конфигурации в системе ”Kubernetes”. Может отсутствовать, если это приложение сконфигурировано в ”OpenNebula”.
open_nebula_config	Первичный ключ сущности ”open_nebula_config_entity” для связи приложения и его конфигурации в системе ”OpenNebula”. Может отсутствовать, если это приложение сконфигурировано в ”Kubernetes”.

Таблица 2: Описание полей сущности ”kubernetes_config_entity”

Имя	**Описание**
id	Первичный ключ сущности, уникальное число для идентификации записи.
namespace	Имя того namespace ”Kubernetes”, в котором развёрнуто управляемое приложение.
deployment	Имя того deployment ”Kubernetes”, к которому относится управляемое приложение.
yaml	Конфигурация подключения к API ”Kubernetes” в формате ”YAML”, эту конфигурацию ещё называют ”kube config”.

Таблица 3: Описание полей сущности ”open_nebula_config_entity”

Имя	**Описание**
id	Первичный ключ сущности, уникальное число для идентификации записи.
address	URL, по которому расположен XML RPC API платформы ”OpenNebula”.
login	Логин пользователя.
password	Пароль пользователя.
role	Идентификатор роли внутри группы ВМ, к которой относится управляемое приложение.
template	Идентификатор шаблона VM, согласно которому будут создаваться новые ВМ управляемого приложения.
vmgroup	Идентификатор группы ВМ, к которой относится управляемое приложение.

3.3 Описание REST API веб-сервиса

В этом разделе содержится перечисление всех вызовов HTTP REST API, которые поддерживает веб-сервис, а так же описание запросов и ответов.

3.3.1 Запросы

Передать для сохранения параметры подключения к приложению под управлением ”Kubernetes” можно с помощью запроса, описание которого приведено в табл. 4 на стр. 28.

Передать для сохранения параметры подключения к приложению под управлением ”OpenNebula” можно с помощью запроса, описание которого приведено в табл. 10 на стр. 31.

Обновить параметры ранее созданного подключения к приложению под управлением ”OpenNebula” можно с помощью запроса, описание которого приведено в табл. 11 на стр. 32.

Обновить параметры ранее созданного подключения к приложению под управлением ”Kubernetes” можно с помощью запроса, описание которого при-

ведено в табл. 5 на стр. 29.

Запросить список приложений, параметры подключения к которым сохранены в СУБД, можно с помощью запроса, описание которого приведено в табл. 6 на стр. 29.

Удалить ранее сохранённые параметры подключения к приложению вне зависимости от платформы, под управлением которой оно находится, можно с помощью запроса, описание которого приведено в табл. 7 на стр. 29.

Таблица 4: Описание запроса сохранения параметров подключения к ”Kubernetes”-приложению

Метод	POST		
Адрес	/api/v1/kubernetes/{namespace}/{deployment}/		
Параметры запроса	**Имя**	**Тип**	**Описание**
	namespace	String	Namespace, к которому относится приложение.
	deployment	String	Deployment, к которому относится приложение.
Тело запроса	Конфигурация подключения к ”Kubernetes” в формате YAML.		

Запросить список ВМ или контейнеров относящихся к приложению, параметры подключения к которому ранее были сохранены, вне зависимости от платформы, под управлением которой оно находится, а так же статистическую информацию об использованных ими ресурсах, можно с помощью запроса, описание которого приведено в табл. 8 на стр. 30.

Осуществить масштабирование приложения, параметры подключения к которому ранее были сохранены, вне зависимости от платформы, под управлением которой оно находится, можно с помощью запроса, описание которого приведено в табл. 9 на стр. 30.

Таблица 5: Описание запроса обновления сохранённых параметров подключения к ”Kubernetes”-приложению

Метод	PUT		
Адрес	/api/v1/kubernetes/{namespace}/{deployment}/{id}/		
Параметры запроса	**Имя**	**Тип**	**Описание**
	namespace	String	Namespace, к которому относится приложение.
	deployment	String	Deployment, к которому относится приложение.
	id	int64	Идентификатор существующей конфигурации, которая должна быть обновлена.
Тело запроса	Конфигурация подключения к ”Kubernetes” в формате YAML.		

Таблица 6: Описание запроса списка сохранённых приложений

Метод	GET
Адрес	/api/v1/app

Таблица 7: Описание запроса удаления сохранённого приложения

Метод	DELETE		
Адрес	/api/v1/app/{id}		
Параметры запроса	**Имя**	**Тип**	**Описание**
	id	int64	Идентификатор удаляемого приложения.

Таблица 8: Описание запроса состояния сохранённого приложения

Метод	GET		
Адрес	/api/v1/app/{id}		
Параметры запроса	**Имя**	**Тип**	**Описание**
	id	int64	Идентификатор приложения, состояние которого должно быть возвращено.

Таблица 9: Описание запроса масштабирования приложения

Метод	PATCH		
Адрес	/api/v1/app/{id}		
Тело запроса	Тело запроса передаётся в формате JSON.		
	Имя	**Тип**	**Описание**
	incrementBy	int64	Количество ВМ или контейнеров, на которое должно быть увеличены доступные приложению ресурсы. В случае, если число отрицательное, такое количество ВМ или контейнеров приложения будет отключено.

Таблица 10: Описание запроса сохранения параметров подключения к ”OpenNebula”-приложению

Метод	POST		
Адрес	/api/v1/opennebula/		
Тело запроса	Тело запроса передаётся в формате JSON.		
	Имя	**Тип**	**Описание**
	address	String	URL, по которому находится XML RPC API OpenNebula, к которой осуществляется подключение.
	login	String	Логин пользователя, с которым осуществляется подключение.
	password	String	Пароль пользователя, с которым осуществляется подключение.
	password	String	Пароль пользователя, с которым осуществляется подключение.
	role	int64	Идентификатор роли, к которой относятся ВМ управляемого приложения.
	template	int64	Идентификатор шаблона, по которому создаются новые ВМ управляемого приложения.
	vmgroup	int64	Идентификатор группы ВМ, в которую входят ВМ управляемого приложения.

Таблица 11: Описание запроса обновления параметров подключения к ”OpenNebula”-приложению

Метод	PUT		
Адрес	/api/v1/opennebula/{id}/		
Тело запроса	Тело запроса передаётся в формате JSON.		
	Имя	**Тип**	**Описание**
	address	String	URL, по которому находится XML RPC API OpenNebula, к которой осуществляется подключение.
	login	String	Логин пользователя, с которым осуществляется подключение.
	password	String	Пароль пользователя, с которым осуществляется подключение.
	password	String	Пароль пользователя, с которым осуществляется подключение.
	role	int64	Идентификатор роли, к которой относятся ВМ управляемого приложения.
	template	int64	Идентификатор шаблона, по которому создаются новые ВМ управляемого приложения.
	vmgroup	int64	Идентификатор группы ВМ, в котороую входят ВМ управляемого приложения.
Параметры запроса	**Имя**	**Тип**	**Описание**
	id	int64	Идентификатор конфигурации OpenNebula приложения, которая должна быть обновлена.

3.3.2 Ответы

Ответ на любой запрос может содержать либо описание ошибки в формате, который является общим для всех запросов, либо содержать тело ответа в формате, зависящем от запроса, либо содержать только HTTP-код ответа без тела сообщения.

Описание общего формата ошибки приведено в табл. 12 на стр. 33.

Таблица 12: Описание общего для всех запросов формата описания ошибки

	Код	**Причина**
Код ответа	500	Внутренняя ошибка сервера, вызванная непредусмотренным набором параметров запроса.
	400	В полученном запросе обнаружена ошибка, запрос нужно сформировать заново.
	404	Полученный запрос был направлен на неизвестный серверу путь, запрос нужно сформировать заново.

Тело ответа	Тело ответа передаётся в формате JSON.		
	Имя	**Тип**	**Описание**
	timestamp	String	Время возникновения ошибки. Формат записи времени соответствует стандарту ISO 8601.
	status	int64	HTTP-код ответа.
	error	String	Текстовое описание HTTP-кода ответа на английском языке.
	message	String	Текстовое описание возникшей ошибки на английском языке.
	path	String	Путь на сервере, на который был получен запрос. Вычисляется относительно корневого пути, без адреса самого сервера.

Ответ на запрос сохранения параметров вне зависимости от платформы, параметры подключения к которой были сохранены, а так же ответ на запрос удаления приложения описан в табл. 13 на стр. 34.

Таблица 13: Описание ответа на запросы сохранения и удаления параметров подключения в случае успеха проведения операции

Код ответа	200		
Тело ответа	Тело ответа передаётся в формате JSON.		
	Имя	**Тип**	**Описание**
	appId	int64	Идентификатор приложения, которое только что было создано или удалено.

Ответ на запрос списка сохранённых приложений, то есть списка известных параметров подключения к приложениям, описан в табл. 15 на стр. 35.

Ответы на запрос состояния приложения и на запрос масштабирования приложения не зависят от платформы, под управлением которой находится само приложение. Формат ответа описан в табл. 16 на стр. 36.

Ответ на запрос обновления конфигурации так же не зависит от платформы облачных вычислений, к которой относится конфигурация. Формат ответа описан в табл. 14 на стр. 34.

Таблица 14: Описание ответа на запрос обновления конфигурации

Код ответа	200

Таблица 15: Описание ответа на запрос списка сохранённых приложений

Код ответа	200
Тело ответа	Тело ответа передаётся в формате JSON. **Имя** **Тип** openNebulaApps OpenNebulaApp[] kubernetesApps KubernetesApp[]
Тип данных OpenNebulaApp	Объекты этого типа используются для описания сохранённой конфигурации параметров подключения к приложению под управлением ”OpenNebula”. **Имя** **Тип** **Описание** appId int64 Идентификатор этого приложения. address String URL-адрес, по которому расположен XML RPC API системы ”OpenNebula”. role int64 Идентификатор роли, к которой относятся ВМ этого приложения. template int64 Идентификатор шаблона, по которому создаются ВМ этого приложения. template int64 Идентификатор группы ВМ, в которую входят ВМ этого приложения.
Тип данных KubernetesApp	Объекты этого типа используются для описания сохранённой конфигурации параметров подключения к приложению под управлением ”Kubernetes”. **Имя** **Тип** **Описание** appId int64 Идентификатор этого приложения. namespace String Namespace, в который входит это приложение. deployment String Deployment, в который входит это приложение.

Таблица 16: Описание ответа на запросы состояния и масштабирования приложения

Код ответа	200
Тело ответа	Тело ответа передаётся в формате JSON. Сам ответ является массивом объектов типа Instance.
Тип данных Instance	Объекты этого типа используются для описания состояния одного контейнера или ВМ приложения.

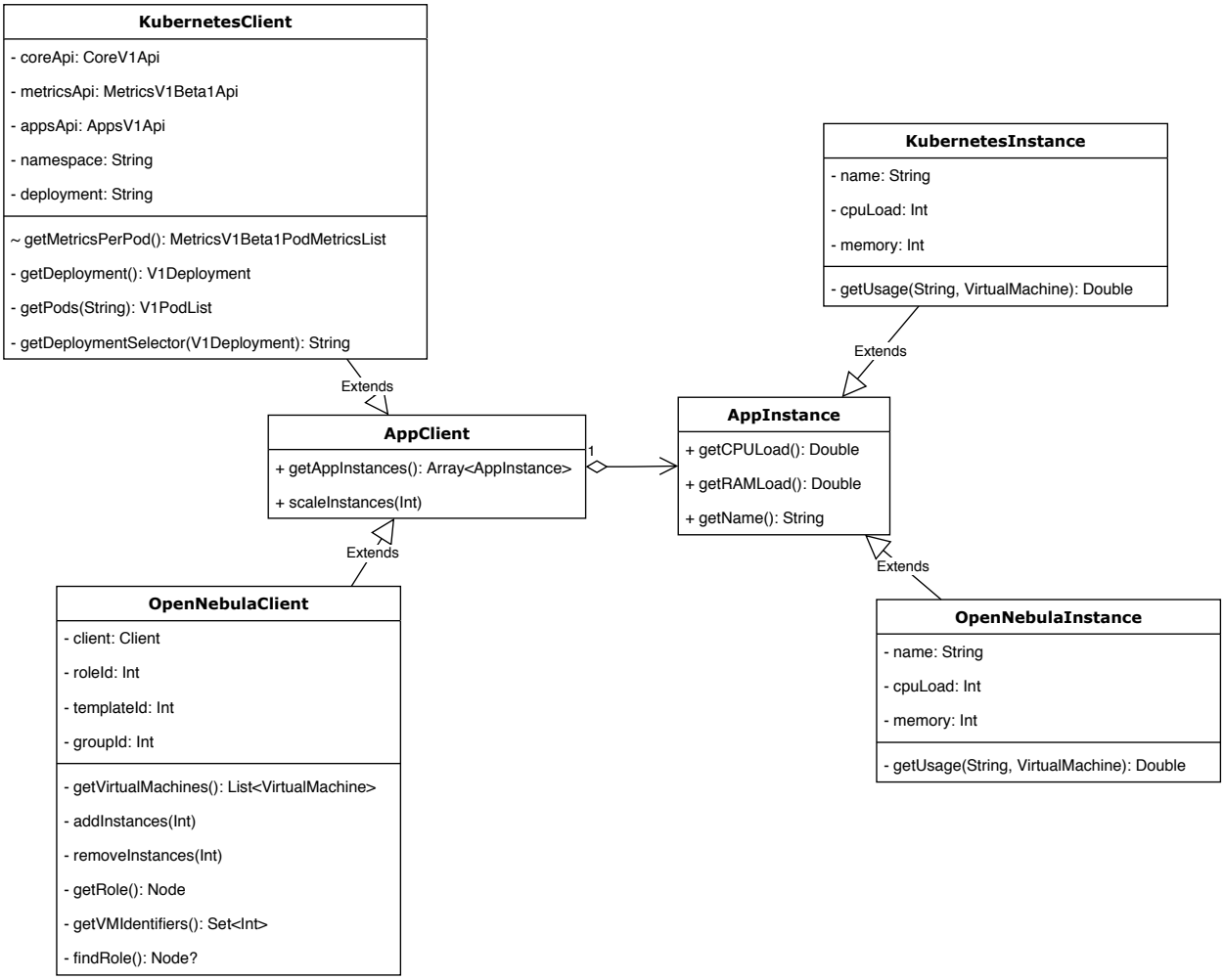
Имя	**Тип**	**Описание**
name	String	Имя контейнера или ВМ, по которому его можно идентифицировать внутри платформы облачных вычислений.
cpuload	float64	Загрузка ЦПУ (CPU). Это значение показывает какое количество вычислительных ядер использовалось в среднем последнюю минуту. Например, 2.3 означает, что 2 вычислительных ядра исполняло команды только этой задачи, а третье 70% времени исполняло команды других задач.
ramload	float64	Загрузка оперативной памяти (RAM). Это значение показывает какое количество байт памяти было занято данными этой задачи в последнюю минуту. Например, 1024 означает, что 1 КиБ памяти был использован этой задачей.

3.4 Описание клиентской библиотеки

Клиентская библиотека — программный модуль в составе сервиса управления виртуализированной инфраструктурой, который отвечает за взаимодействие с каждой из поддерживаемых платформ облачных вычислений, а так же предоставляет унифицированный программный интерфейс (API), не зависящий от конкретной платформы.

На рис. 4 на стр. 37 приведена диаграмма классов клиентской библиотеки. В центре диаграммы находятся два интерфейса ”AppClient” и ”AppInstance”, которые служат для абстрагирования пользователей библиотеки от такого с какой именно платформой облачных вычислений они работают.

Рис. 4: Диаграмма классов



В частности, ”AppClient” предоставляет интерфейс взаимодействия с приложением, а именно два метода:

1. getAppInstances — метод для получения списка ВМ или контейнеров, выделенных этому приложению;
2. scaleInstances — метод для изменения количества выделенных приложению ВМ или контейнеров.

Выделенные приложению ВМ или контейнеры (в зависимости от платформы) представлены интерфейсом ”AppInstance”, который позволяет узнать своё имя, заданное на платформе облачных вычислений, а так же собранную статистическую информацию по использованным ресурсам. В частности, ”getCPULoad” возвращает загрузку вычислительных ядер в течение последней минуты, а ”getRAMLoad” возвращает количество используемых приложением байт оперативной памяти.

Остальные классы на данной диаграмме являются реализациями двух описанных выше интерфейсов под конкретные виртуализированные инфраструктуры. Как видно из набора полей и методов этих классов, внутренне они существенно различаются, но, несмотря на это, пользователи библиотеки могут взаимодействовать с ними через один и тот же интерфейс, что ускоряет и упрощает разработку новых модулей облачных систем.

ЗАКЛЮЧЕНИЕ

В выпускной квалификационной работе были получены следующие результаты.

1. Проведён обзор предметной области, в результате которого были выявлены преимущества и недостатки существующих решений управления виртуализированной инфраструктурой, а также типовые подходы к управлению такой инфраструктурой.
2. Было установлено, что в настоящее время в публичном доступе не существует сервиса управления виртуализированной инфраструктурой, который удовлетворял бы всем требованиям, выдвинутым в данной работе к такому сервису.
3. В результате этого было принято решение о разработке нового подхода. Для этого была поставлена задача, а затем сформулированы технические требования к его реализации в соответствии с поставленной задачей.
4. Был разработан сервис управления виртуализированной инфраструктурой, удовлетворяющий всем требованиям, сформулированным в работе и задании.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] US 9 755 858 B2, 2017.

[2] WO 2014/021849 Al

[3] Aktas, MS. Hybrid cloud computing monitoring software architecture. Concurrency Computat Pract Exper. 2018; 30:e4694. https://doi.org/10.1002/cpe.4694

[4] Y. Wadia, R. Gaonkar and J. Namjoshi, ”Portable Autoscaler for Managing Multi-cloud Elasticity,”2013 International Conference on Cloud & Ubiquitous Computing & Emerging Technologies, Pune, 2013, pp. 48-51.

[5] M. M. Shreyas, ”Federated Cloud Services using Virtual API Proxy Layer in a Distributed Cloud Environment,”2017 Ninth International Conference on Advanced Computing (ICoAC), Chennai, 2017, pp. 134-141.

[6] RU 2020612651, 2020.

[7] Kratzke, Nane. ”Smuggling Multi-cloud Support into Cloud-native Applications using Elastic Container Platforms.”CLOSER. 2017.

[8] Kovács, J. (2019). Supporting Programmable Autoscaling Rules for Containers and Virtual Machines on Clouds. Journal of Grid Computing, 17(4), 813-829.

[9] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu and W. Zhou, ”A Comparative Study of Containers and Virtual Machines in Big Data Environment,”2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, 2018, pp. 178-185, doi: 10.1109/CLOUD.2018.00030.

[10] Naresh Kumar Sehgal, Pramod Chandra P. Bhatt. Cloud Computing: Concepts and Practices — Springer, 2018. — 269 c.

[11] Petcu, D., Macariu, G., Panica, S., & Crăciun, C. (2013). Portable cloud applications — from theory to practice. Future Generation Computer Systems, 29(6), 1417-1430.

[12] Мартынчук И. Г., Жмылёв С. А. Сравнительный анализ систем для

организации облачных вычислений // Сборник трудов VIII научно-практической конференции молодых ученых «Вычислительные системы и сети (Майоровские чтения)» – 2017.

[13] Singh J. et al. Data flow management and compliance in cloud computing //IEEE Cloud Computing. – 2015. – Т. 2. – No. 4. – С. 24-32.

[14] Мартынчук И. Г., Жмылёв С. А. Архитектура и организация сервисов автомасштабирования в облачных системах // Альманах научных работ молодых ученых Университета ИТМО – 2017.

[15] Matthew Portnoy. Virtualization Essentials, 2nd Edition. – New York: Wiley / Sybex, 2016. – 336 с.

[16] Жмылёв С.А., Мартынчук И.Г., Киреев В.Ю. Оценка длины периода нестационарных процессов в облачных системах // VI научно-практическая конференция с международным участием «Наука настоящего и будущего» для студентов, аспирантов и молодых ученых. 2018. С. 41–43.

[17] Bogatyrev V., Vinokurova M. Control and safety of operation of duplicated computer systems // International Conference on Distributed Computer and Communication Networks / Springer. 2017. С. 331–342.

[18] Жмылёв С.А., Киреев В.Ю., Мартынчук И.Г. Исследование систем с нестационарными процессами // Сборник тезисов докладов конгресса молодых ученых. – СПб., 2018.

[19] Мартынчук И.Г., Жмылёв С.А. Модели и методы композиции нестационарных распределений // Сборник тезисов докладов конгресса молодых ученых. – СПб., 2019.

[20] Мартынчук И.Г. Разработка программного комплекса для автоматического управления ресурсами облачной вычислительной системы // Аннотированный сборник научно-исследовательских выпускных квалификационных работ бакалавров Университета ИТМО. 2017. С. 44–46.

[21] Mr. Ray J Rafaels. Cloud Computing: From Beginning to End. – CreateSpace Independent Publishing Platform, 2015. – 152 c.

[22] Mao M., Humphrey M. A performance study on the vm startup time in the cloud // Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on. – IEEE, 2012. – C. 423-430.

[23] Chhibber A., Batra S. Security analysis of cloud computing //International Journal of Advanced Research in Engineering and Applied Sciences. – 2013. – T. 2. – No. 3. – C. 2278-6252.

[24] Rodriguez I., Llana L., Rabanal P. A General Testability Theory: Classes, properties, complexity, and testing reductions //IEEE Transactions on Software Engineering. – 2014. – T. 40. – No. 9. – C. 862-894.

[25] He S. et al. Elastic application container: A lightweight approach for cloud resource provisioning //Advanced information networking and applications (aina), 2012 ieee 26th international conference on. – IEEE, 2012. – C. 15-22.

[26] Optimal cloud resource auto-scaling for web applications / Jiang J., Jie Lu, Guangquan Zhang [и др.] // 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. IEEE, 2013. C. 58–65.

[27] I. Bermudez, S. Traverso, M. Mellia. Exploring the cloud from passive measurements: The Amazon AWS case // 2013 Proceedings IEEE INFOCOM. IEEE, 2013. C. 230–234.

[28] Медведев Алексей. Облачные технологии: тенденции развития, примеры исполнения // Современные технологии автоматизации. 2013. Т. 2. С. 6–9.

[29] Analytical methods of nonstationary processes modeling / Sergei Zhmylev, Ilya Martynchuk, Valeriy Kireev [и др.] // CEUR Workshop Proceedings. 2019.

[30] Using machine learning for black-box autoscaling / Muhammad Wajahat, Anshul Gandhi, Alexei Karve [и др.] // 2016 Seventh International Green and Sustainable Computing Conference (IGSC) / IEEE. 2016. C. 1–8.

[31] Comparison of open-source cloud management platforms: OpenStack and OpenNebula / Xiaolong Wen, Genqiang Gu, Qingchun Li [и др.] // 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery / IEEE. 2012. С. 2457–2461.

[32] Barr Jeff, Narin Attila, Varia Jinesh. Building fault-tolerant applications on aws // Amazon Web Services. 2011. С. 1–15.

[33] Lorido-Botrán Tania, Miguel-Alonso José, Lozano Jose Antonio. Auto-scaling techniques for elastic applications in cloud environments // Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09. 2012. Т. 12. С. 2012.

ПРИЛОЖЕНИЕ 1. ЛИСТИНГИ РАЗРАБОТАННЫХ ПРОГРАММНЫХ МОДУЛЕЙ

```
1  package ru.ifmo.kirmanak.manager.controllers
2
3  import org.slf4j.LoggerFactory
4  import org.springframework.beans.factory.annotation.Autowired
5  import org.springframework.dao.DataIntegrityViolationException
6  import org.springframework.data.repository.findByIdOrNull
7  import org.springframework.web.bind.annotation.*
8  import ru.ifmo.kirmanak.elasticappclient.AppClient
9  import ru.ifmo.kirmanak.elasticappclient.AppClientException
10 import ru.ifmo.kirmanak.elasticappclient.AppInstance
11 import ru.ifmo.kirmanak.manager.models.exceptions.ExistingApplicationException
12 import ru.ifmo.kirmanak.manager.models.exceptions.InvalidAppConfigException
13 import ru.ifmo.kirmanak.manager.models.exceptions.NoAppConnectionException
14 import ru.ifmo.kirmanak.manager.models.exceptions.NoSuchApplicationException
15 import ru.ifmo.kirmanak.manager.models.requests.OpenNebulaRequest
16 import ru.ifmo.kirmanak.manager.models.requests.ScaleRequest
17 import ru.ifmo.kirmanak.manager.models.responses.*
18 import ru.ifmo.kirmanak.manager.storage.entities.AppConfiguration
19 import ru.ifmo.kirmanak.manager.storage.entities.ApplicationEntity
20 import ru.ifmo.kirmanak.manager.storage.entities.KubernetesConfigEntity
21 import ru.ifmo.kirmanak.manager.storage.entities.OpenNebulaConfigEntity
22
23 @RestController
24 class ApiController {
25     private val logger = LoggerFactory.getLogger(javaClass)
26
27     @Autowired
28     private lateinit var appRepository: ApplicationRepository
29
30     @Autowired
31     private lateinit var kubernetesConfigRepo: KubernetesConfigRepository
32
33     @Autowired
34     private lateinit var openNebulaConfigRepo: OpenNebulaConfigRepository
35
36     @PostMapping("/api/v1/kubernetes/{namespace}/{deployment}")
37     fun createKubernetes(
38             @RequestBody yaml: String,
39             @PathVariable("namespace") namespace: String,
40             @PathVariable("deployment") deployment: String
```

```
41      ): AppIdResponse {
42          logger.info("createKubernetes(namespace: \"$namespace\", deployment:
            ↪ \"$deployment\", yaml: \"$yaml\")")
43
44          val config = KubernetesConfigEntity(deployment, namespace, yaml)
45          checkAppConfig(config)
46          val id = saveConfiguration(config)
47
48          return result("createKubernetes", AppIdResponse(id))
49      }
50
51      @PostMapping("/api/v1/opennebula")
52      fun createOpenNebula(
53              @RequestBody request: OpenNebulaRequest
54      ): AppIdResponse {
55          logger.info("createOpenNebula(request = $request)")
56
57          val config = OpenNebulaConfigEntity(request.address, request.login,
            ↪ request.password, request.role, request.template, request.vmgroup)
58          checkAppConfig(config)
59          val id = saveConfiguration(config)
60
61          return result("createOpenNebula", AppIdResponse(id))
62      }
63
64      @GetMapping("/api/v1/app/{id}")
65      fun getApplicationInfo(@PathVariable("id") id: Long): Array<AppInstanceResponse> {
66          logger.info("getApplicationInfo(id: \"$id\")")
67
68          val client = getApp(id).getAppClient()
69
70          val result = getAppInfo(client)
71
72          return result("getApplicationInfo", result)
73      }
74
75      @GetMapping("/api/v1/app")
76      fun getAppConfigurations(): AppConfigResponse {
77          logger.info("getAppConfigurations()")
78
79          val apps = appRepository.findAll()
80          val kubernetesApps = apps.mapNotNull {
81              KubernetesConfigResponse(
82                      namespace = it.kubernetesConfig?.namespace ?: return@mapNotNull null,
83                      deployment = it.kubernetesConfig.deployment,
84                      appId = it.id ?: throw IllegalStateException("Id must not be null if
                        ↪ data is retrieved from DB")
```

```
85              )
86          }
87
88          val openNebulaApps = apps.mapNotNull {
89              OpenNebulaConfigResponse(
90                      address = it.openNebulaConfig?.address ?: return@mapNotNull null,
91                      appId = it.id ?: throw IllegalStateException("Id must not be null if
                        ↪  data is retrieved from DB"),
92                      role = it.openNebulaConfig.role,
93                      template = it.openNebulaConfig.template,
94                      vmgroup = it.openNebulaConfig.vmgroup
95              )
96          }
97
98          return result("getAppConfigurations", AppConfigResponse(openNebulaApps,
            ↪  kubernetesApps))
99      }
100
101     @DeleteMapping("/api/v1/app/{id}")
102     fun removeApplication(@PathVariable("id") id: Long): AppIdResponse {
103         logger.info("removeApplication(id: \"$id\")")
104
105         if (appRepository.existsById(id))
106             appRepository.deleteById(id)
107         else
108             throw NoSuchApplicationException(id)
109
110         return result("removeApplication", AppIdResponse(id))
111     }
112
113     @PutMapping("/api/v1/kubernetes/{namespace}/{deployment}/{id}")
114     fun updateKubernetes(
115             @RequestBody yaml: String,
116             @PathVariable("namespace") namespace: String,
117             @PathVariable("deployment") deployment: String,
118             @PathVariable("id") id: Long
119     ) {
120         logger.info("updateKubernetes(namespace: \"$namespace\", deployment:
            ↪  \"$deployment\", id = $id, yaml: \"$yaml\")")
121
122         val app = getApp(id)
123         val currentConfig = app.kubernetesConfig ?: throw NoSuchApplicationException(id)
124         val updated = KubernetesConfigEntity(deployment, namespace, yaml, app,
            ↪  currentConfig.id)
125
126         checkAppConfig(updated)
```

```
127         saveConfiguration(updated, app)
128     }
129
130     @PutMapping("/api/v1/opennebula/{id}")
131     fun updateOpenNebula(
132             @RequestBody request: OpenNebulaRequest,
133             @PathVariable("id") id: Long
134     ) {
135         logger.info("updateOpenNebula(request = $request, id = $id)")
136
137         val app = getApp(id)
138         val currentConfig = app.openNebulaConfig ?: throw NoSuchApplicationException(id)
139         val config = OpenNebulaConfigEntity(
140                 request.address, request.login, request.password, request.role,
141                 request.template, request.vmgroup, app, currentConfig.id
142         )
143
144         checkAppConfig(config)
145         saveConfiguration(config, app)
146     }
147
148     @PatchMapping("/api/v1/app/{id}")
149     fun scaleApplication(@PathVariable("id") id: Long, @RequestBody request:
    ↪ ScaleRequest): Array<AppInstanceResponse> {
150         logger.info("scaleApplication(id = $id, request = $request)")
151
152         val client = getApp(id).getAppClient()
153
154         client.scaleInstances(request.incrementBy)
155
156         val result = getAppInfo(client)
157
158         return result("scaleApplication", result)
159     }
160
161     private fun checkAppConfig(config: AppConfiguration) {
162         logger.info("checkAppConfig(config = $config)")
163
164         val instances: Array<AppInstance>
165         try {
166             instances = config.getAppClient().getAppInstances()
167         } catch (e: AppClientException) {
168             logger.error("checkAppConfig: no connection", e)
169             throw NoAppConnectionException(e)
170         }
171
```

```
172             logger.info("checkAppConfig: instances count = ${instances.size}")
173             for (instance in instances) {
174                 try {
175                     logger.info("checkAppConfig: instance(name = \"${instance.getName()}\",
                        ↪  CPU = ${instance.getCPULoad()}), RAM = ${instance.getRAMLoad()}")
176                 } catch (e: AppClientException) {
177                     logger.error("checkAppConfig: unable to get instance info", e)
178                     throw InvalidAppConfigException(e)
179                 }
180             }
181         }
182
183         private fun saveConfiguration(config: KubernetesConfigEntity, oldApp:
            ↪  ApplicationEntity? = null): Long {
184             val saved: KubernetesConfigEntity
185
186             try {
187                 saved = kubernetesConfigRepo.save(config)
188             } catch (e: DataIntegrityViolationException) {
189                 val existing = kubernetesConfigRepo.findByNamespaceAndDeploymentAndYaml(
190                         config.namespace, config.deployment, config.yaml
191                 ) ?: throw e
192                 val id = existing.application?.id ?: throw e
193                 throw ExistingApplicationException(id)
194             }
195
196             val app = ApplicationEntity(kubernetesConfig = saved, id = oldApp?.id)
197             return saveApplication(app)
198         }
199
200         private fun saveConfiguration(config: OpenNebulaConfigEntity, oldApp:
            ↪  ApplicationEntity? = null): Long {
201             val saved: OpenNebulaConfigEntity
202
203             try {
204                 saved = openNebulaConfigRepo.save(config)
205             } catch (e: DataIntegrityViolationException) {
206                 val existing =
                    ↪  openNebulaConfigRepo.findByAddressAndLoginAndPasswordAndRoleAndTemplateAndVmgroup(
207                         config.address, config.login, config.password, config.role,
                            ↪  config.template, config.vmgroup
208                 ) ?: throw e
209                 val id = existing.application?.id ?: throw e
210                 throw ExistingApplicationException(id)
211             }
212
```

```
213             val app = ApplicationEntity(openNebulaConfig = saved, id = oldApp?.id)
214             return saveApplication(app)
215         }
216 
217         private fun saveApplication(app: ApplicationEntity): Long {
218             return appRepository.save(app).id
219                     ?: throw IllegalStateException("Id for a new application was not
                    ↪  generated")
220         }
221 
222         private fun <Result> result(method: String, result: Result): Result {
223             logger.info("$method result: $result")
224             return result
225         }
226 
227         private fun <Result> result(method: String, result: Array<Result>): Array<Result> {
228             logger.info("$method result: ${result.contentToString()}")
229             return result
230         }
231 
232         private fun getApp(id: Long) = appRepository.findByIdOrNull(id) ?: throw
        ↪  NoSuchApplicationException(id)
233 
234         private fun getAppInfo(client: AppClient) = client.getAppInstances().map {
235             try {
236                 AppInstanceResponse(it.getCPULoad(), it.getRAMLoad(), it.getName())
237             } catch (e: AppClientException) {
238                 logger.error("getAppInfo: unable to get instance info", e)
239                 throw e
240             }
241         }.toTypedArray()
242     }
```

```
1   package ru.ifmo.kirmanak.elasticappclient
2   
3   class AppClientException(override val message: String) : Exception(message) {
4       constructor(throwable: Throwable) : this(throwable.toString()) {
5           addSuppressed(throwable)
6       }
7   }
```

```
1   package ru.ifmo.kirmanak.elasticappclient
2   
3   import io.kubernetes.client.openapi.ApiClient
4   import org.opennebula.client.Client
5   import ru.ifmo.kirmanak.elasticappclient.kubernetes.KubernetesClient
6   import ru.ifmo.kirmanak.elasticappclient.opennebula.OpenNebulaClient
```

```
7
8  open class AppClientFactory {
9      companion object {
10         @JvmStatic
11         fun getClient(kubeClient: ApiClient, namespace: String, deployment: String):
           ↪ AppClient {
12             return KubernetesClient(kubeClient, namespace, deployment)
13         }
14
15         @JvmStatic
16         fun getClient(openNebulaClient: Client, groupID: Int, roleId: Int, templateId:
           ↪ Int): AppClient {
17             return OpenNebulaClient(openNebulaClient, groupID, roleId, templateId)
18         }
19     }
20 }
```

```
1  package ru.ifmo.kirmanak.elasticappclient
2
3  /**
4   * Interface to access elastic application working in a virtualized infrastructure.
5   */
6  interface AppClient {
7      /**
8       * Requests information about currently working application instances from
   ↪   infrastructure provider.
9       */
10     @Throws(AppClientException::class)
11     fun getAppInstances(): Array<AppInstance>
12
13     /**
14      * Scales application instances count by {@param count}.
15      * Instances are added if {@param count} is positive and removed if it is negative.
16      * If {@param count} is zero nothing will be done.
17      */
18     @Throws(AppClientException::class)
19     fun scaleInstances(count: Int)
20 }
```

```
1  package ru.ifmo.kirmanak.manager.models.responses
2
3  data class AppConfigResponse(
4          val openNebulaApps: List<OpenNebulaConfigResponse>,
5          val kubernetesApps: List<KubernetesConfigResponse>
6  )
```

```
1  package ru.ifmo.kirmanak.manager.storage.entities
2
```

```
import ru.ifmo.kirmanak.elasticappclient.AppClient

/**
 * Represents configuration of connection to an application hosted in a virtualized
 ↪ infrastructure
 */
interface AppConfiguration {
    /**
     * Creates elastic application client out of the configuration
     */
    fun getAppClient(): AppClient
}
```

```
package ru.ifmo.kirmanak.manager.models.responses

data class AppIdResponse(val appId: Long)
```

```
package ru.ifmo.kirmanak.elasticappclient

/**
 * Interface to access elastic application instance information.
 * It is either a container or a virtual machine.
 *
 * Be aware: all of the information is cached in object's fields.
 * Instance might not exist by the time data is requested.
 */
interface AppInstance {
    /**
     * CPU load is represented in CPU core usage in the last minute.
     * For instance, if this value is 0.5, then half of a CPU core was used in the minute
     ↪ before data was requested.
     * CPU usage is always absolute value, it is the same on 12-core machine and 36-core
     ↪ machine.
     */
    fun getCPULoad(): Double

    /**
     * RAM usage is represented in bytes of RAM used in the last minute.
     * For example, if this value is 1024 then 1 KiB of RAM was used by this instance in
     ↪ the minute before data was requested.
     * RAM usage is always absolute value, it is the same both on 16 GiB RAM machine and
     ↪ 1 GiB RAM machine.
     */
    fun getRAMLoad(): Double

    /**
     * Name of the instance in terms of virtualized infrastructure provider.
```

```
     */
    fun getName(): String
}
```

```
package ru.ifmo.kirmanak.manager.models.responses

import ru.ifmo.kirmanak.elasticappclient.AppInstance

data class AppInstanceResponse(
        private val CPULoad: Double,
        private val RAMLoad: Double,
        private val name: String
) : AppInstance {

    override fun getCPULoad(): Double = CPULoad

    override fun getName(): String = name

    override fun getRAMLoad(): Double = RAMLoad
}
```

```
package ru.ifmo.kirmanak.manager.storage.entities

import ru.ifmo.kirmanak.elasticappclient.AppClient
import ru.ifmo.kirmanak.manager.storage.constraints.OnlyOneSetConstraint
import javax.persistence.*

@Entity
@Table(uniqueConstraints = [UniqueConstraint(columnNames = ["open_nebula_config",
↪  "kubernetes_config"])])
@OnlyOneSetConstraint(fields = ["openNebulaConfig", "kubernetesConfig"])
data class ApplicationEntity(
        @OneToOne(optional = true, orphanRemoval = true)
        @JoinColumn(name = "kubernetes_config")
        val kubernetesConfig: KubernetesConfigEntity? = null,

        @OneToOne(optional = true, orphanRemoval = true)
        @JoinColumn(name = "open_nebula_config")
        val openNebulaConfig: OpenNebulaConfigEntity? = null,

        @Id
        @GeneratedValue
        val id: Long? = null
) : AppConfiguration {
    override fun getAppClient(): AppClient {
        val configuration = arrayOf(kubernetesConfig, openNebulaConfig).first { it !=
        ↪  null }
```

```
                ?: throw IllegalStateException("Exactly one configuration must not be
                ↪  null")

        return configuration.getAppClient()
    }
}
```

```
package ru.ifmo.kirmanak.manager.controllers

import org.springframework.data.repository.CrudRepository
import ru.ifmo.kirmanak.manager.storage.entities.ApplicationEntity

interface ApplicationRepository : CrudRepository<ApplicationEntity, Long>
```

```
package ru.ifmo.kirmanak.manager.models.exceptions

import org.springframework.http.HttpStatus
import org.springframework.web.bind.annotation.ResponseStatus

@ResponseStatus(code = HttpStatus.BAD_REQUEST)
class ExistingApplicationException(existingId: Long) : Exception("Such application has id
↪  = $existingId")
```

```
package ru.ifmo.kirmanak.manager.models.exceptions

import org.springframework.http.HttpStatus
import org.springframework.web.bind.annotation.ResponseStatus

@ResponseStatus(code = HttpStatus.BAD_REQUEST, reason = "Application configuration is
↪  invalid")
class InvalidAppConfigException(override val message: String) : Exception(message) {
    constructor(e: Throwable) : this(e.message ?: e.toString()) {
        addSuppressed(e)
    }
}
```

```
package ru.ifmo.kirmanak.elasticappclient.kubernetes

import io.kubernetes.client.openapi.ApiClient
import io.kubernetes.client.openapi.ApiException
import io.kubernetes.client.openapi.apis.AppsV1Api
import io.kubernetes.client.openapi.apis.CoreV1Api
import io.kubernetes.client.openapi.models.V1Deployment
import io.kubernetes.client.openapi.models.V1Scale
import ru.ifmo.kirmanak.elasticappclient.AppClient
import ru.ifmo.kirmanak.elasticappclient.AppClientException
import ru.ifmo.kirmanak.elasticappclient.AppInstance

```

```
13  private const val DEPLOYMENT_SELECTOR = "app"
14
15  open class KubernetesClient(
16      apiClient: ApiClient, private val namespace: String, private val deployment: String
17  ) : AppClient {
18      private val coreApi = CoreV1Api(apiClient)
19      private val metricsApi = MetricsV1Beta1Api(apiClient)
20      private val appsApi = AppsV1Api(apiClient)
21
22      override fun getAppInstances(): Array<AppInstance> {
23          val dep = getDeployment()
24
25          val selector = getDeploymentSelector(dep)
26          val labelSelector = "$DEPLOYMENT_SELECTOR=$selector"
27
28          val pods = getPods(labelSelector)
29
30          return pods.items.mapNotNull { KubernetesInstance.create(it, this)
            ↪  }.toTypedArray()
31      }
32
33      override fun scaleInstances(count: Int) {
34          if (count == 0) return
35
36          val scale: V1Scale
37
38          try {
39              scale = appsApi.readNamespacedDeploymentScale(deployment, namespace,
                ↪  false.toString())
40          } catch (e: ApiException) {
41              throw AppClientException(e)
42          }
43
44          val currentCount = scale.spec?.replicas
45              ?: throw AppClientException("Unable to get replica count for deployment
                ↪  \"$deployment\"")
46          scale.spec?.replicas = currentCount + count
47
48          try {
49              appsApi.replaceNamespacedDeploymentScale(deployment, namespace, scale, null,
                ↪  null, null)
50          } catch (e: ApiException) {
51              throw AppClientException(e)
52          }
53      }
54
```

```
55      internal fun getMetricsPerPod() = metricsApi.getPodMetrics(namespace)
56
57      private fun getDeployment() =
58          try {
59              appsApi.readNamespacedDeployment(deployment, namespace, null, null, null)
60                  ?: throw AppClientException("Deployment \"$deployment\" was not found in
                    ↪  namespace \"$namespace\"")
61          } catch (e: ApiException) {
62              throw AppClientException(e)
63          }
64
65
66      private fun getPods(labelSelector: String) =
67          try {
68              coreApi.listNamespacedPod(namespace, null, null, null, null, labelSelector,
                ↪  null, null, null, null)
69                  ?: throw AppClientException("Pods list of \"$deployment\" was not found
                    ↪  in namespace \"$namespace\"")
70          } catch (e: ApiException) {
71              throw AppClientException(e)
72          }
73
74      private fun getDeploymentSelector(dep: V1Deployment) =
        ↪  dep.spec?.selector?.matchLabels?.get(DEPLOYMENT_SELECTOR)
75          ?: throw AppClientException("Deployment \"$deployment\" has no
            ↪  \"$DEPLOYMENT_SELECTOR\" selector")
76  }
```

```
1   package ru.ifmo.kirmanak.manager.storage.entities
2
3   import io.kubernetes.client.util.ClientBuilder
4   import io.kubernetes.client.util.KubeConfig
5   import ru.ifmo.kirmanak.elasticappclient.AppClient
6   import ru.ifmo.kirmanak.elasticappclient.AppClientFactory
7   import java.io.StringReader
8   import javax.persistence.*
9
10  @Entity
11  @Table(uniqueConstraints = [UniqueConstraint(columnNames = ["deployment", "namespace",
    ↪  "yaml"])])
12  data class KubernetesConfigEntity(
13          @Column(nullable = false)
14          val deployment: String,
15
16          @Column(nullable = false)
17          val namespace: String,
18
```

```
        @Column(nullable = false, length = 4096)
        val yaml: String,

        @OneToOne(optional = false, mappedBy = "kubernetesConfig")
        val application: ApplicationEntity? = null,

        @Id
        @GeneratedValue
        val id: Long? = null
) : AppConfiguration {

    override fun getAppClient(): AppClient {
        val configReader = StringReader(yaml)
        val client =
        ↪ ClientBuilder.kubeconfig(KubeConfig.loadKubeConfig(configReader)).build()
        return AppClientFactory.getClient(client, namespace, deployment)
    }

    override fun toString(): String {
        return "KubernetesConfigEntity(deployment='$deployment', namespace='$namespace',
        ↪ yaml='$yaml', application=${application?.id}, id=$id)"
    }
}
```

```
package ru.ifmo.kirmanak.manager.controllers

import org.springframework.data.repository.CrudRepository
import ru.ifmo.kirmanak.manager.storage.entities.KubernetesConfigEntity

interface KubernetesConfigRepository : CrudRepository<KubernetesConfigEntity, Long> {
    fun findByNamespaceAndDeploymentAndYaml(namespace: String, deployment: String, yaml:
    ↪ String): KubernetesConfigEntity?
}
```

```
package ru.ifmo.kirmanak.manager.models.responses

data class KubernetesConfigResponse(
        val appId: Long,
        val namespace: String,
        val deployment: String
)
```

```
package ru.ifmo.kirmanak.elasticappclient.kubernetes

import io.kubernetes.client.openapi.models.V1Pod
import ru.ifmo.kirmanak.elasticappclient.AppClientException
import ru.ifmo.kirmanak.elasticappclient.AppInstance
import ru.ifmo.kirmanak.elasticappclient.kubernetes.models.MetricsV1Beta1PodMetrics
```

```kotlin
import java.util.concurrent.atomic.DoubleAdder

internal data class KubernetesInstance(
    private val name: String,
    private val CPULoad: Double,
    private val RAMLoad: Double
) : AppInstance {

    companion object {
        fun create(pod: V1Pod, client: KubernetesClient): KubernetesInstance? {
            val name = pod.metadata?.name ?: throw AppClientException("Pod name or whole
            ↪  metadata is unknown")
            val cpuLoad = getUsage(name, "cpu", client) ?: return null
            val ramLoad = getUsage(name, "memory", client) ?: return null

            return KubernetesInstance(name, cpuLoad, ramLoad)
        }

        private fun getUsage(podName: String, metricName: String, client:
        ↪  KubernetesClient): Double? {
            val podContainers = getPodMetrics(podName, client)?.containers

            return podContainers?.fold(DoubleAdder()) { acc, container ->
                val usage = container.usage?.get(metricName)?.number
                    ?: throw AppClientException("Usage of \"$metricName\" was not found
                    ↪  for container \"${container.name}\"")
                acc.add(usage.toDouble())
                acc
            }?.toDouble()
        }

        private fun getPodMetrics(podName: String, client: KubernetesClient):
        ↪  MetricsV1Beta1PodMetrics? {
            val allMetrics = client.getMetricsPerPod().items ?: return null

            for (item in allMetrics) {
                if (item.metadata?.name == podName)
                    return item
            }

            return null
        }
    }

    override fun getCPULoad() = CPULoad

```

```
    override fun getRAMLoad() = RAMLoad

    override fun getName() = name
}
```

```
package ru.ifmo.kirmanak.manager

import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.boot.runApplication

@SpringBootApplication
class ManagerApplication

fun main(args: Array<String>) {
        runApplication<ManagerApplication>(*args)
}
```

```
package ru.ifmo.kirmanak.elasticappclient.kubernetes

import com.google.gson.reflect.TypeToken
import io.kubernetes.client.openapi.ApiClient
import io.kubernetes.client.openapi.ApiResponse
import okhttp3.Call
import ru.ifmo.kirmanak.elasticappclient.AppClientException
import ru.ifmo.kirmanak.elasticappclient.kubernetes.models.MetricsV1Beta1PodMetricsList

private const val API_PATH = "/apis/metrics.k8s.io/v1beta1"
private val AVAILABLE_ACCEPT_HEADERS = arrayOf(
    "application/json",
    "application/yaml",
    "application/vnd.kubernetes.protobuf",
    "application/json;stream=watch",
    "application/vnd.kubernetes.protobuf;stream=watch"
)

internal class MetricsV1Beta1Api(private var api: ApiClient) {

    fun getPodMetrics(namespace: String? = null): MetricsV1Beta1PodMetricsList {
        val path = if (namespace === null) {
            "pods"
        } else {
            "namespaces/$namespace/pods"
        }

        val call = buildGET("$API_PATH/$path")
        val type = object : TypeToken<MetricsV1Beta1PodMetricsList>() {}

```

```
        return execute(call, type).data
    }

    private fun buildGET(url: String): Call {
        val headers: MutableMap<String, String> = HashMap()

        api.selectHeaderAccept(AVAILABLE_ACCEPT_HEADERS)?.let { headers["Accept"] = it }
        api.selectHeaderContentType(emptyArray())?.let { headers["Content-Type"] = it }

        val authNames = arrayOf("BearerToken")
        return api.buildCall(
            url, "GET", emptyList(), emptyList(), null, headers,
            emptyMap(), emptyMap(), authNames, null
        )
    }

    private fun <T> execute(call: Call, typeToken: TypeToken<T>): ApiResponse<T> {
        val type = typeToken.type

        try {
            return api.execute(call, type)
        } catch (e: Exception) {
            throw AppClientException(e)
        }
    }
}
```

```
package ru.ifmo.kirmanak.elasticappclient.kubernetes.models

import io.kubernetes.client.custom.Quantity

/*
{
    "name": "nginx",
    "usage": {
        "cpu": "0",
        "memory": "2392Ki"
    }
}
*/

data class MetricsV1Beta1Container(
    val name: String?,
    val usage: Map<String, Quantity>?
)
```

```
package ru.ifmo.kirmanak.elasticappclient.kubernetes.models
```

```

import io.kubernetes.client.openapi.models.V1ObjectMeta

/*
{
    "metadata": {
        "name": "nginx-deployment-54f57cf6bf-vrf9h",
        "namespace": "default",
        "selfLink":
↪   "/apis/metrics.k8s.io/v1beta1/namespaces/default/pods/nginx-deployment-54f57cf6bf-vrf9h",
        "creationTimestamp": "2020-01-08T12:35:52Z"
    },
    "timestamp": "2020-01-08T12:35:00Z",
    "window": "1m0s",
    "containers": []
}
*/
data class MetricsV1Beta1PodMetrics(
    val metadata: V1ObjectMeta?,
    val containers: List<MetricsV1Beta1Container>?
)
```

```
package ru.ifmo.kirmanak.elasticappclient.kubernetes.models

import io.kubernetes.client.openapi.models.V1ObjectMeta

/*
{
  "kind": "PodMetricsList",
  "apiVersion": "metrics.k8s.io/v1beta1",
  "metadata": {
    "selfLink": "/apis/metrics.k8s.io/v1beta1/pods"
  },
  "items": []
*/
data class MetricsV1Beta1PodMetricsList(
    val kind: String?,
    val apiVersion: String?,
    val metadata: V1ObjectMeta?,
    val items: List<MetricsV1Beta1PodMetrics>?
)
```

```
package ru.ifmo.kirmanak.manager.models.exceptions

import org.springframework.http.HttpStatus
import org.springframework.web.bind.annotation.ResponseStatus

```

```
@ResponseStatus(code = HttpStatus.BAD_REQUEST, reason = "Can not establish connection to
↪ the requested application")
class NoAppConnectionException(override val message: String) : Exception(message) {
    constructor(e: Throwable) : this(e.message ?: e.toString()) {
        addSuppressed(e)
    }
}
```

```
package ru.ifmo.kirmanak.manager.models.exceptions

import org.springframework.http.HttpStatus
import org.springframework.web.bind.annotation.ResponseStatus

@ResponseStatus(code = HttpStatus.BAD_REQUEST)
class NoSuchApplicationException(id: Long) : Exception("Application with id = $id was not
↪ found")
```

```
package ru.ifmo.kirmanak.manager.storage.constraints

import javax.validation.Constraint
import javax.validation.Payload
import kotlin.reflect.KClass

@Constraint(validatedBy = [OnlyOneSetValidator::class])
@Target(AnnotationTarget.CLASS)
@Retention(AnnotationRetention.RUNTIME)
annotation class OnlyOneSetConstraint(
        val groups: Array<KClass<in Any>> = [],
        val payload: Array<KClass<in Payload>> = [],
        val fields: Array<String> = [],
        val message: String = "Only one field must be not-null"
)
```

```
package ru.ifmo.kirmanak.manager.storage.constraints

import org.springframework.beans.BeanWrapperImpl
import javax.validation.ConstraintValidator
import javax.validation.ConstraintValidatorContext

class OnlyOneSetValidator : ConstraintValidator<OnlyOneSetConstraint, Any> {
    private lateinit var columns: Array<String>

    override fun initialize(constraintAnnotation: OnlyOneSetConstraint) {
        super.initialize(constraintAnnotation)
        columns = constraintAnnotation.fields
    }

    override fun isValid(value: Any, context: ConstraintValidatorContext): Boolean {
```

```
16          val wrapper = BeanWrapperImpl(value)
17
18          return columns.map { wrapper.getPropertyValue(it) }.count { it != null } == 1
19      }
20
21  }
```

```
1   package ru.ifmo.kirmanak.elasticappclient.opennebula
2
3   import org.opennebula.client.Client
4   import org.opennebula.client.template.Template
5   import org.opennebula.client.vm.VirtualMachine
6   import org.opennebula.client.vm.VirtualMachinePool
7   import org.opennebula.client.vmgroup.VMGroup
8   import org.w3c.dom.Node
9   import ru.ifmo.kirmanak.elasticappclient.AppClient
10  import ru.ifmo.kirmanak.elasticappclient.AppClientException
11  import ru.ifmo.kirmanak.elasticappclient.AppInstance
12  import kotlin.math.min
13
14  open class OpenNebulaClient(
15      private val client: Client, private val groupId: Int, private val roleId: Int,
        ↪ private val templateId: Int
16  ) : AppClient {
17
18      override fun getAppInstances(): Array<AppInstance> =
19          getVirtualMachines().map { OpenNebulaInstance(it) }.toTypedArray()
20
21      override fun scaleInstances(count: Int) {
22          if (count > 0) addInstances(count)
23          else if (count < 0) removeInstances(-count)
24      }
25
26      private fun getVirtualMachines(): List<VirtualMachine> {
27          val pool = VirtualMachinePool(client)
28          throwIfError(pool.info())
29          val vmIDs = getVMIdentifiers()
30
31          return pool.filter { vmIDs.contains(it.id()) }
32      }
33
34      private fun addInstances(count: Int) {
35          val template = Template(templateId, client)
36          val roleName = getString(getRole(), "NAME")
37          val groupTemplate = "VMGROUP = [ VMGROUP_ID = \"$groupId\", ROLE = \"$roleName\"
            ↪ ]"
38          for (i in 0 until count) {
```

```
            val response = template.instantiate("", false, groupTemplate, false)
            throwIfError(response)
        }
    }

    private fun removeInstances(count: Int) {
        val list = getVirtualMachines()

        val toRemove = min(count, list.size)
        if (toRemove == 0) return

        list.take(toRemove).forEach { throwIfError(it.terminate()) }
    }

    private fun getRole(): Node =
        findRole() ?: throw AppClientException("Failed to find role with id $roleId in VM
        ↪ group $groupId")

    private fun getVMIdentifiers(): Set<Int> {
        val role = getRole()
        val vmList = getString(role, "VMS")
        return vmList.split(",").filter { it.isNotBlank() }.map { it.toInt()
        ↪ }.toSortedSet()
    }

    private fun findRole(): Node? {
        val group = VMGroup(groupId, client)

        val root = getRootElement(group.info())
        val roleList = getNodeList(root, "ROLES/ROLE")
        for (i in 0 until roleList.length) {
            val item = roleList.item(i)
            val id = getNumber(item, "ID").toInt()
            if (id == roleId)
                return item
        }

        return null
    }
}
```

```
package ru.ifmo.kirmanak.manager.storage.entities

import org.opennebula.client.Client
import ru.ifmo.kirmanak.elasticappclient.AppClient
import ru.ifmo.kirmanak.elasticappclient.AppClientFactory
import javax.persistence.*
```

```
7
8  @Entity
9  @Table(uniqueConstraints = [UniqueConstraint(columnNames = ["address", "login",
   ↪  "password", "role", "template", "vmgroup"])])
10 data class OpenNebulaConfigEntity(
11         @Column(nullable = false)
12         val address: String,
13
14         @Column(nullable = false)
15         val login: String,
16
17         @Column(nullable = false)
18         val password: String,
19
20         @Column(nullable = false)
21         val role: Int,
22
23         @Column(nullable = false)
24         val template: Int,
25
26         @Column(nullable = false)
27         val vmgroup: Int,
28
29         @OneToOne(optional = false, mappedBy = "openNebulaConfig")
30         val application: ApplicationEntity? = null,
31
32         @Id
33         @GeneratedValue
34         val id: Long? = null
35 ) : AppConfiguration {
36
37     override fun getAppClient(): AppClient {
38         val client = Client("$login:$password", address)
39         return AppClientFactory.getClient(client, vmgroup, role, template)
40     }
41
42     override fun toString(): String {
43         return "OpenNebulaConfigEntity(address='$address', login='$login',
           ↪  password='$password', role=$role, template=$template, vmgroup=$vmgroup,
           ↪  application=${application?.id}, id=$id)"
44     }
45 }
```

```
1  package ru.ifmo.kirmanak.manager.controllers
2
3  import org.springframework.data.repository.CrudRepository
4  import ru.ifmo.kirmanak.manager.storage.entities.OpenNebulaConfigEntity
```

```

interface OpenNebulaConfigRepository : CrudRepository<OpenNebulaConfigEntity, Long> {
    fun findByAddressAndLoginAndPasswordAndRoleAndTemplateAndVmgroup(
            address: String, login: String, password: String, role: Int, template: Int,
            ↪ vmgroup: Int
    ): OpenNebulaConfigEntity?
}
```

```
package ru.ifmo.kirmanak.manager.models.responses

data class OpenNebulaConfigResponse(
        val appId: Long,
        val address: String,
        val role: Int,
        val template: Int,
        val vmgroup: Int
)
```

```
package ru.ifmo.kirmanak.elasticappclient.opennebula

import org.opennebula.client.vm.VirtualMachine
import ru.ifmo.kirmanak.elasticappclient.AppClientException
import ru.ifmo.kirmanak.elasticappclient.AppInstance

internal class OpenNebulaInstance(vm: VirtualMachine) : AppInstance {
    private val name: String = vm.name ?: throw AppClientException("Unknown vm name")
    private val cpuLoad = getUsage("CPU", vm) / 100
    private val memoryLoad = getUsage("MEMORY", vm) * 1024

    override fun getCPULoad() = cpuLoad

    override fun getRAMLoad() = memoryLoad

    override fun getName() = name

    private fun getUsage(resource: String, vm: VirtualMachine): Double {
        val root = getRootElement(vm.info())
        return getNumber(root, "MONITORING/$resource")
    }

    override fun toString(): String {
        return "OpenNebulaNode(name='$name', cpuLoad=$cpuLoad, memoryLoad=$memoryLoad)"
    }
}
```

```
package ru.ifmo.kirmanak.manager.models.requests

data class OpenNebulaRequest(
```

```
        val address: String,
        val login: String,
        val password: String,
        val role: Int,
        val template: Int,
        val vmgroup: Int
)
```

```
package ru.ifmo.kirmanak.manager.models.requests

data class ScaleRequest(val incrementBy: Int)
```

```
package ru.ifmo.kirmanak.elasticappclient.opennebula

import org.opennebula.client.OneResponse
import org.w3c.dom.Element
import org.w3c.dom.Node
import org.w3c.dom.NodeList
import ru.ifmo.kirmanak.elasticappclient.AppClientException
import java.io.ByteArrayInputStream
import javax.xml.parsers.DocumentBuilderFactory
import javax.xml.xpath.XPathConstants
import javax.xml.xpath.XPathFactory


internal val Any.xpath by lazy {
    XPathFactory.newInstance().newXPath()
}

internal val Any.docBuilder by lazy {
    DocumentBuilderFactory.newInstance().newDocumentBuilder()
}

@Throws(AppClientException::class)
internal fun throwIfError(response: OneResponse): OneResponse {
    if (response.isError)
        throw AppClientException(response.errorMessage)
    else
        return response
}

internal fun Any.getRootElement(response: OneResponse): Element {
    val message = throwIfError(response).message
    val stream = ByteArrayInputStream(message.toByteArray())
    return docBuilder.parse(stream).documentElement
}

```

```
36 internal fun Any.getNodeList(node: Node, expression: String): NodeList =
37     xpath.evaluate(expression, node, XPathConstants.NODESET) as NodeList
38
39 internal fun Any.getNumber(node: Node, expression: String): Double =
40     xpath.evaluate(expression, node, XPathConstants.NUMBER) as Double
41
42 internal fun Any.getString(node: Node, expression: String): String =
43     xpath.evaluate(expression, node, XPathConstants.STRING) as String
```