Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

“Московский физико-технический институт
(государственный университет)”

Факультет управления и прикладной математики

Кафедра теоретической и прикладной информатики

“Разработка подсистемы оптимизации стоимости владения AWS”

Выпускная квалификационная работа

(бакалаврская работа)

Направление подготовки: 03.03.01 Прикладные математика и физика

Выполнил:

студент 376 группы _____ Самара О. С.

Научный руководитель: _____ Дяйкин А. Д.
Научный консультант: _____ Кулага А. А.

Москва 2017

Содержание

1. Введение

1.1. Облачные вычисления

Облачные вычисления (cloud computing) — это модель обеспечения повсеместного и удобного сетевого доступа по требованию к общему пулу вычислительных ресурсов (такие как сети передачи данных, серверы, хранилища данных, приложения и сервисы), которые могут быть быстро выделены и освобождены с минимальными усилиями или обращениями к поставщику [1].

В модели облачных вычислений выделяют 5 ключевых характеристик:

- **Самообслуживание по требованию.** Потребитель может в одностороннем порядке обеспечивать изменение объема предоставляемых облачных ресурсов, выделение новых облачных ресурсов, освобождение занятых облачных ресурсов без непосредственного взаимодействия с представителем поставщика услуг;
- **Обеспечение сетевого доступа.** К облачные ресурсам возможно получить доступ по сети с использованием стандартных механизмов с произвольных устройств (мобильные телефоны, планшеты, ноутбуки или рабочие станции);
- **Объединение ресурсов в коллекции.** Вычислительные ресурсы поставщика объединяются для обслуживания большого числа потребителей, динамически распределяя и перераспределяя физические и виртуальные ресурсы согласно потребностям потребителей. При этом клиент не имеет контроля над фактическим распределением ресурсов, имея возможность задавать только более высокий уровень абстракции, например географическое местоположение виртуальной машины

(страна, город, дата-центр). В качестве примера ресурсов можно привести размер хранилища данных, вычислительные мощности, память и канал сетевого доступа;

- **Эластичность.** Предоставляемые ресурсы могут быть получены или освобождены в соответствии с уровнем потребления потребителя, как правило, в автоматическом режиме;
- **Учет потребления.** Облачные системы автоматически контролируют и оптимизируют использованные ресурсы в различных уровнях абстракции (например, хранилища данных, вычислительные мощности, сетевой канал). Использование ресурсов может быть отслежено, обеспечивая прозрачность как для поставщика услуг, так и для клиента использующего облачные ресурсы. Эта характеристика получила название “Платеж по мере использования” (Pay-As-You-Go).

В модели облачных вычислений выделяют 3 способа предоставления облачных услуг:

- **Software as a Service (SaaS).** Потребителю предоставляется доступ к программным продуктам разработчика, функционирующих в облачной инфраструктуре. Доступ к такому программному обеспечению может осуществляться как с помощью веб-браузера, так и с помощью другого заранее оговоренного программного интерфейса. В такой модели распространения услуг потребитель не имеет контроля над обслуживающей сервис облачной инфраструктурой, включая сеть, сервера и операционные системы. Также потребитель не имеет контроля и над обслуживающим его программным продуктом — то есть, версия продукта, его функционал и интерфейс фиксируется поставщиком облачных услуг;

- **Platform as a Service (PaaS).** Потребителю предоставляется возможность развернуть на заданной облачной инфраструктуре собственные или полученные от третьих лиц программные продукты, используя программные языки, библиотеки, сервисы и инструменты, предоставляемые поставщиком облачных услуг. Потребитель не имеет контроля над обслуживающей его инфраструктурой, но может контролировать развертываемые приложения и конфигурационные настройки обслуживающего приложение окружения;
- **Infrastructure as a Service (IaaS).** Потребителю предоставляется возможность распределять и перераспределять вычислительные возможности, дисковое хранилище, возможности сети передачи данных и другие фундаментальные вычислительные ресурсы. Потребитель имеет возможность устанавливать и запускать произвольное программное обеспечение, включая операционные системы. Пользователь не управляет обслуживающей инфраструктурой напрямую, а лишь заказывает определенные ее характеристики производительности.

1.2. Преимущества облачных вычислений

Технологии облачных вычислений имеют ряд преимуществ перед традиционным хостингом (utility computing):

- Облачные вычисления позволяют более эффективно использовать ресурсы как провайдеру, так и потребителям облачных инфраструктур. Таким образом, потребитель, использовав некоторое количество часов реального вычислительного времени, платит только за него и ему не приходится заботится о простое вычислительных ресурсов и оплате простаивающей инфраструктуры, которое имеет место в случае с

традиционным хостингом (в случае с традиционным хостингом стоимость поддержки простаивающих ресурсов целиком лежит на потребителе). С другой стороны, провайдер может выделять облачные ресурсы потребителям по требованию, позволяя более эффективно использовать имеющееся оборудование — то есть, в случае простоя зарезервированных одним из потребителей облачных ресурсов ресурсов провайдер может просто перераспределить ресурсы, избавляя себя от проблемы простоя и соответствующих не окупающих себя расходов;

- Расположение виртуальных машин в облаке позволяет максимально быстро расширять объем предоставляемых ресурсов при пиковых нагрузках, позволяя сохранить работоспособность и доступность потребительских виртуальных машин. Кроме того, это расширение может происходить в автоматическом режиме, то есть, без участия провайдера. В то же время, в случае предоставления физического хостинга, расширение ресурсов занимает значительно большее время и необходимость явного участия в этом сотрудников на стороне провайдера. К тому же, в силу скачкообразности расширения инфраструкту, непосредственно после расширения инфраструктуры в случае хостинга будут следовать простои оборудования, что негативно сказывается на бюджете потребителя.

1.3. Стоимость облачных вычислений

Обычно предоставление облачных услуг производится по заранее объявленным провайдером облачных сервисов тарифам и тарифным планам.

Под этим подразумевается, что провайдер обязуется предоставлять в любой время потребителю заданный объем оперативной памяти,

процессорной мощности, дискового хранилища и сетевого канала за определенную оговариваемую тарифом почасовую плату, которая может как зависеть от степени утилизации предоставляемых ресурсов потребителем, так и не зависеть от нее.

Также возможна услуга резервации, когда потребитель платит не за фактическое потребление, а резервирует облачные мощности для последующего использования. По сути, в таком сценарии пользователь платит наперед. Наличие такого варианта обосновано тем, что провайдеру облачных услуг значительно удобнее предлагать свои услуги в таком виде. Именно поэтому тарифы резервации ощутимо ниже, чем тарифы потребления в реальном времени.

2. Задача оптимизации стоимости владения

2.1. Описание задачи

Среди задач облачных вычислений можно выделить ряд оптимизационных задач; среди них оказывается задача оптимизации стоимости владения, которую можно описать следующим образом.

Поскольку выбор тарифного плана лежит на потребителе, перед ним встает оптимизационная задача: необходимо выбрать тарифный план для всех своих виртуальных машин таким образом, чтобы, с одной стороны, их мощность могла обеспечить достаточную производительность в плане количества проводимых бизнес-транзакций; при этом из всех удовлетворяющих критериям выше планов нужно выбрать такой, который обеспечит минимальную стоимость владения.

Отметим, что эта задача оптимизации может быть решена только при наличии на руках исторических данных мониторинга инфраструктуры; эта задача не допускает статического анализа — то есть, по заданному набору используемых технологий и по коду запускаемого потребительского приложение заранее невозможно определить, какое же количество ресурсов потребуется для поддержки отказоустойчивой, оперативной работы.

Более того, зачастую с помощью облачных технологий потребители облачных услуг решают задачи, связанные с обслуживанием своей клиентской базы — например, потребитель облачных услуг желает использовать облачные вычисления для обеспечения работы своего веб-сервера. В таких условиях оказывается принципиально невозможно по конфигурации сервера и сайта/приложения/сервиса предсказать, какая же будет нагрузка на инфраструктуру и какое же количество ресурсов потребуется для нормальной (без задержек и без отказов) работы

потребительского сервиса, ведь соответствующие вычислительные мощности сильно зависят от количества клиентов потребителя.

Более того, требуемая производительность инфраструктуры зависит не только лишь от количества клиентов потребителя, но и от сложности бизнес-транзакций, ассоциируемых с каждым клиентом. Под этим имеется ввиду случай, когда разные запросы клиента являются различными по природе или просто же требуют различного количества ресурсов для их обработки; хорошим примером является сравнение запроса страницы, при котором сервер потребителя возвращает заранее сгенерированную страницу, и запрос страницы, генерация которой требует обращения к базе данных или каких-либо вычислений.

Таким образом, становится очевидным невозможен статический подход к формулировке и решению данной задачи. Поэтому сформулируем задачу следующим образом.

2.2. Постановка задачи

В данной работе автор решает задачу создания программного комплекса, предоставляющего веб-интерфейс для мониторинга сервиса EC2 и получения рекомендаций по оптимизации стоимости его владения.

В данной работе в качестве основного ресурса будут рассматриваться процессорное время (в силу того, что без внедрения агентов в виртуальные машины потребителя является невозможным измерение утилизации памяти). Таким образом, каждой виртуальной машине поставим в соответствие функцию от времени, а именно — функцию зависимости средней нагрузки ЦПУ от времени.

В качестве решения задачи оптимизации будем рассматривать рекомендацию потребителю насчет оптимального тарифного плана для

данной виртуальной машины. В качестве опорных данных будем брать зависимость среднего процента утилизации ЦПУ от времени. На базе этой метрики будем выводить стоимость обладания данной машиной в данной конфигурации на данном отрезке времени. Здесь и далее будем считать, что стоимость работы машины пропорциональна затраченному процессорному времени с коэффициентом стоимости одного часа работы на данной конфигурации.

3. AWS

В качестве примера реализации концепции облачных вычислений будет рассмотрен комплекс проектов AWS — Amazon Web Services.

AWS — инфраструктура платформ облачных веб-сервисов, представленная компанией Amazon в начале 2006 года. В данной инфраструктуре представлено много сервисов для предоставления различных услуг, таких как: аренда виртуальных серверов (в дальнейшем будем их называть EC2 инстансы, или просто инстансы), предоставление вычислительных мощностей, хранение данных (файловый хостинг, распределенные хранилища данных, распределенные реляционные базы данных), DNS-услуги, создание изолированных сетей из ресурсов AWS, балансирование входящего трафика по нескольким инстансам.

Особо стоит выделить услуги-решения AWS, предоставляемые потребителям для мониторинга и управления предоставляемых облачных услуг, а именно IAM (Amazon Identity and Access Management) и CloudWatch.

3.1. EC2 (Amazon Elastic Compute Cloud)

Amazon Elastic Compute Cloud — веб-сервис, который предоставляет вычислительные мощности в облаке. Сервис входит в инфраструктуру Amazon Web Services.

С помощью EC2 можно:

- создать Amazon Machine Image (AMI), который будет содержать приложения, библиотеки, данные и связанные с ними конфигурационные параметры потребителя; возможно использование заранее настроенных шаблонов образов для работы;

- загрузить AMI в Amazon S3. Amazon EC2 предоставляет инструменты, для хранения AMI. Amazon S3 обеспечивает безопасное, надежное и быстрое хранилище для хранения образов;
- использовать Amazon EC2 веб-сервис для настройки безопасности и сетевого доступа;
- выбирать тип(ы) операционной системы, какой необходим потребителю, запускать, завершать, или контролировать несколько AMI по мере необходимости, используя API веб-сервиса, или различных инструментов управления, которые предусмотрены.

В дальнейшем AMI (Amazon Machine Image), запущенные на виртуальном сервере, будем именовать инстансами.

3.2. IAM (Amazon Identity and Access Management)

Amazon Identity and Access Management — веб-сервис, позволяющий контролировать доступ к AWS ресурсам. С помощью IAM возможно контролировать, кто имеет доступ к ресурсам (аутентификация) и к каким именно ресурсам будет иметь доступ этот человек (авторизация).

IAM позволяет потребителю возможность предоставлять другим пользователям права администрации и использования ресурсов, связанных с аккаунтом данного потребителя, не разглашая при этом аутентификационные данные аккаунта потребителя и избавляя пользователей от необходимости обретения AWS аккаунта; при этом все ассоциированные с деятельностью таких пользователей действия затраты оплачиваются владельцем аккаунта — в этой работе он именуется потребителем.

IAM оперирует понятиями account (аккаунт), users (пользователи), groups (группы пользователей), credentials (ключи доступа), permissions (разрешения или права) и policies (политики).

- Права — сущность, обладание которой обеспечивает доступ к функциям ресурсов, включая управляющие функции;
- Политика — множество прав; это множество прав можно присвоить пользователю, группе или роли;
- Аккаунт — под этой сущностью имеется ввиду аккаунт потребителя облачных услуг;
- Пользователь — сущность внутри аккаунта, ассоциируемая с человеком либо сервисом; владелец аккаунта либо обладатель соответствующих прав может создавать пользователей внутри своего аккаунта и присваивать им определенные права, назначать им роли, распределять их в группы и присваивать им ключи доступа;
- Группа — множество пользователей; упрощает присвоение группе пользователей определенных прав; одна группа может содержать в себе множество пользователей, равно как и один пользователь может состоять во множестве групп. В таком случае, пользователь наследует все права, которыми обладают группы, в которых он состоит;
- Ключи представляют собой пару [access key, secret access key]; пользователь-обладатель этой пары ключей получает возможность совершать программные вызовы, обладающие правами, закрепленными за этими ключами.

Соотношение между группами, пользователями и ключами доступа иллюстрирует следующая схема:

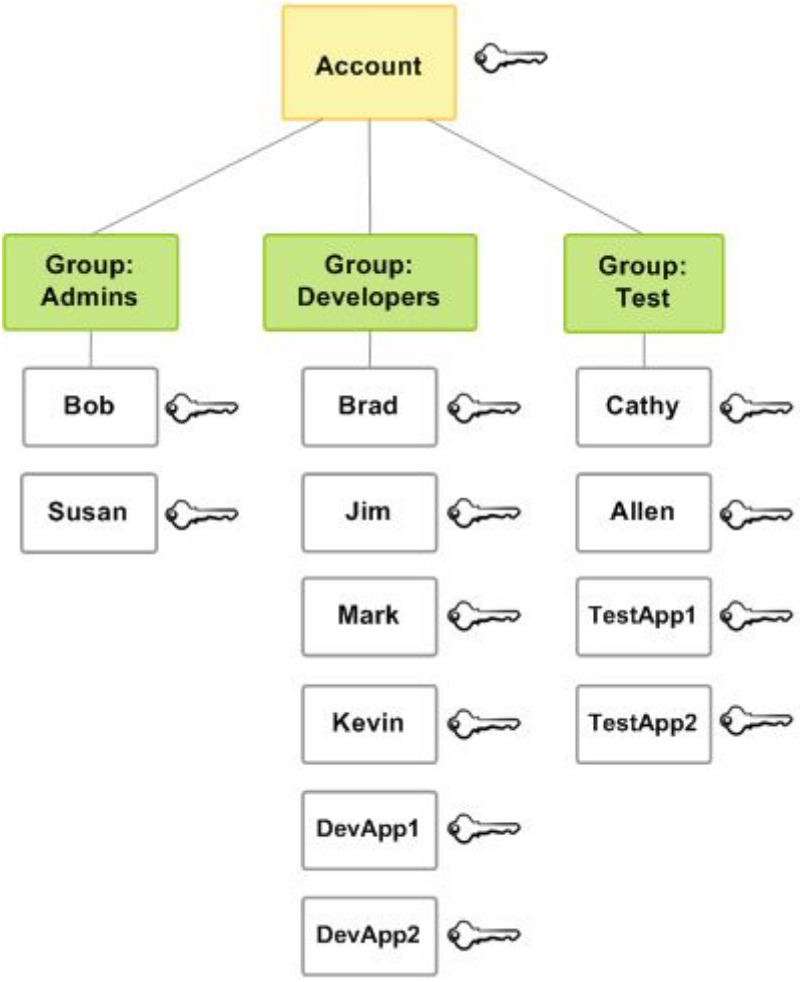

Рис. 1: Пример конфигурации групп, пользователей и ключей доступа.

3.3. CloudWatch

Amazon CloudWatch — веб-сервис, предоставляющий возможность мониторинга в реальном времени сервисов AWS, включая мониторинг EC2 инстансов пользователя путем предоставления метрик нагрузки ресурсов [7]. CloudWatch предоставляет информацию о загрузке ЦПУ, о количестве обращений к диску и о количестве переданной через сеть информации. CloudWatch не предоставляет мониторинг потребляемой памяти или наличного дискового пространства без установки дополнительного программного обеспечения внутри инстансов — так называемых агентов.

Также CloudWatch предоставляет услугу хранения вышеописанных метрик, то есть, запросами к CloudWatch можно извлечь не только метрики, связанные с текущим моментом времени, но и определенную историю метрик.

CloudWatch предоставляет REST API, обращение к которому идет посредством HTTP или HTTPS запросов. Также существуют AWS SDK для языков Java, JavaScript, PHP, Python, Ruby.

Одним из инструментов, используемых при реализации данного диплома, является как раз такой SDK для языка Python, который называется Boto.

4. Решение

4.1. Используемые инструменты и понятия

В этой части описан набор технологий, используемых для реализации дипломного проекта.

4.1.1. SQLite

SQLite — компактная встраиваемая реляционная база данных.

Слово «встраиваемый» означает, что SQLite не использует парадигму клиент-сервер, то есть движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а предоставляет библиотеку, с которой программа компонуется, и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа. Простота реализации достигается за счёт того, что перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется.

Именно возможность работы нескольких потоков с базой данных и простота реализации реализации такой работы послужили причинами выбора SQLite в качестве базы данных проекта.

4.1.2. Python 3

Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений и удобные высокоуровневые структуры данных. Код в Python организовывается в функции и классы, которые могут объединяться в модули (они в свою очередь могут быть объединены в пакеты).

Python был выбран для этой работы именно ввиду простоты написания на нем пользовательских приложений и краткости необходимого кода. Помимо этого для Python существует множество фреймворков и пакетов, делающих его самодостаточным для разработки этого проекта полностью на Python, что и будет показано в следующих частях этого раздела.

4.1.3. SQLAlchemy

SQLAlchemy — это программная библиотека на языке Python для работы с реляционными СУБД с применением технологии ORM. SQLAlchemy служит для синхронизации объектов Python и записей реляционной базы данных. SQLAlchemy позволяет описывать структуры баз данных и способы взаимодействия с ними на языке Python без использования SQL [9].

Именно простота, связанная с отсутствием необходимости явно писать запросы к SQL базе данных, послужила причиной выбора этой библиотеки в качестве связующего звена между основной логикой сервера и базой данных.

4.1.3. Boto 3

Boto — это AWS SDK для Python, позволяющий разработчикам писать программное обеспечение, которое взаимодействует с сервисами Amazon такими как EC2, S3 и Cloudwatch. Boto предоставляет как простое в использовании объектно-ориентированное API, так и низкоуровневый прямой доступ к сервисам [8].

Boto был выбран в силу того, что он является оберткой API AWS для языка Python и предоставляет необходимый для работы программного комплекса функционал.

В Boto нас интересуют следующие классы и вызовы:

1. class CloudWatch.Client
    a. get_metric_statistics()
    b. list_metrics()
2. class EC2.Client
    a. describe_instances()
3. class boto3.session.Session
    a. get_available_regions(service_name)

4.1.4. Flask

Flask — фреймворк для создания веб-приложений на языке программирования Python, использующий набор инструментов Werkzeug, а также шаблонизатор Jinja2. Flask относится к категории так называемых микрофреймворков — минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности [6].

Flask был выбран из-за предлагаемого упрощения написания веб-сервера приложения.

4.1.5. REST API

REST (Representational State Transfer — “передача состояния представления”) — архитектурный стиль взаимодействия компонентов распределенного приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределенной гипермедиа-системы. Вызов удаленной процедуры может представлять собой обычный HTTP-запрос (обычно “GET” или “POST”; такой запрос называют “REST-запрос”), а необходимые данные передаются в качестве параметров запроса [2].

В ходе работы над проектом, разрабатываемого в течении этой дипломной работы, было разработано REST API, обеспечивающее взаимодействие отдаваемой пользователю страницы с сервером.

4.2. Схема работы

В этой части опишем принципиальную схему работы приложения как на уровне взаимодействия с пользователем и AWS, так и на уровне внутреннего взаимодействия частей системы.

4.2.1. Внешнее устройство

В этом пункте опишем протокол, согласно которому работает программный комплекс, а именно — порядок событий, которые происходят от начала работы пользователя с системой (тот момент, когда пользователь решает воспользоваться комплексом для своих нужд) до момента получения пользователем ожидаемого от программного комплекса результата. Для этого

пункта ограничимся операциями, наблюдаемыми снаружи — то есть, опустим внутреннее устройство описываемого программного комплекса.

Протокол кратко описывается следующей схемой:

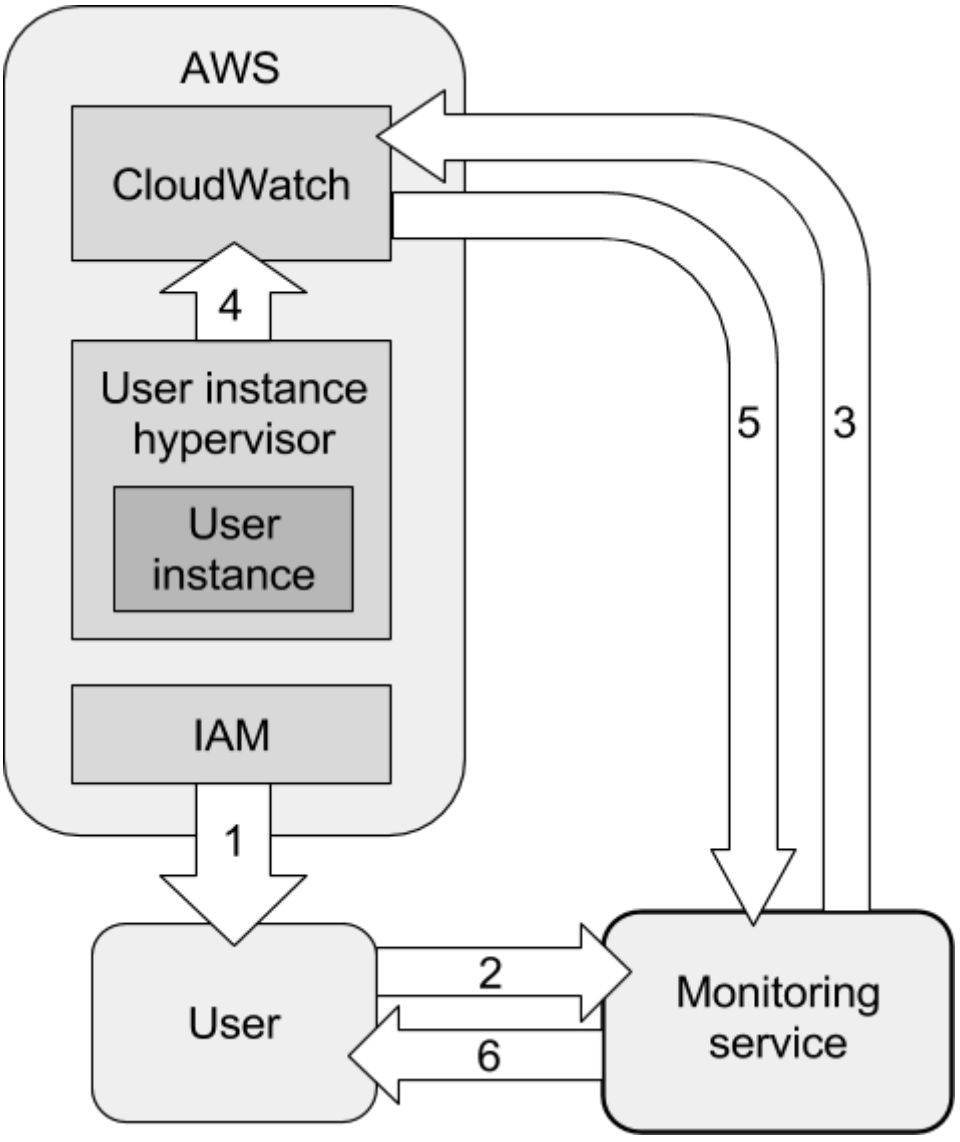

Рис. 2: Организация общения описываемого в работе программного комплекса (monitoring service) с потребителем (user) и AWS

Поясним эту схему:

1. У пользователя возникает потребность в мониторинге производительности и стоимости его инстансов. Он обращается к AWS IAM, чтобы получить ключи доступа с правами достаточными для сбора метрик из CloudWatch;

2. Пользователь передает полученные от IAM ключи доступа программному комплексу;
3. Программный комплекс с помощью Boto делает запрос на метрики производительности к CloudWatch, пользуясь предоставленными пользователем ключами доступа;
4. CloudWatch собирает запрашиваемые метрики — на самом деле, CloudWatch собирает метрики не только лишь в момент запроса к нему, а постоянно; затем он по запросу их отдает. Но для данной схемы это не имеет никакого значения;
5. CloudWatch передает метрики производительности программному комплексу;
6. Программный комплекс обрабатывает метрики производительности, обсчитывает метрики стоимости и оптимизационные рекомендации, компонует требуемую пользователем информацию на странице и отдает полученную страницу пользователю.

4.2.2. Внутреннее устройство

В этом пункте опишем части программного комплекса и механизм их взаимодействия.

Программный комплекс состоит из сервера на Flask, базы данных, модуля управления стоимостью, модуля сбора метрик производительности и модуля отрисовки страниц. Автор этой работы разработал модуль управления стоимостью и принимал участие в разработке сервера и базы данных.

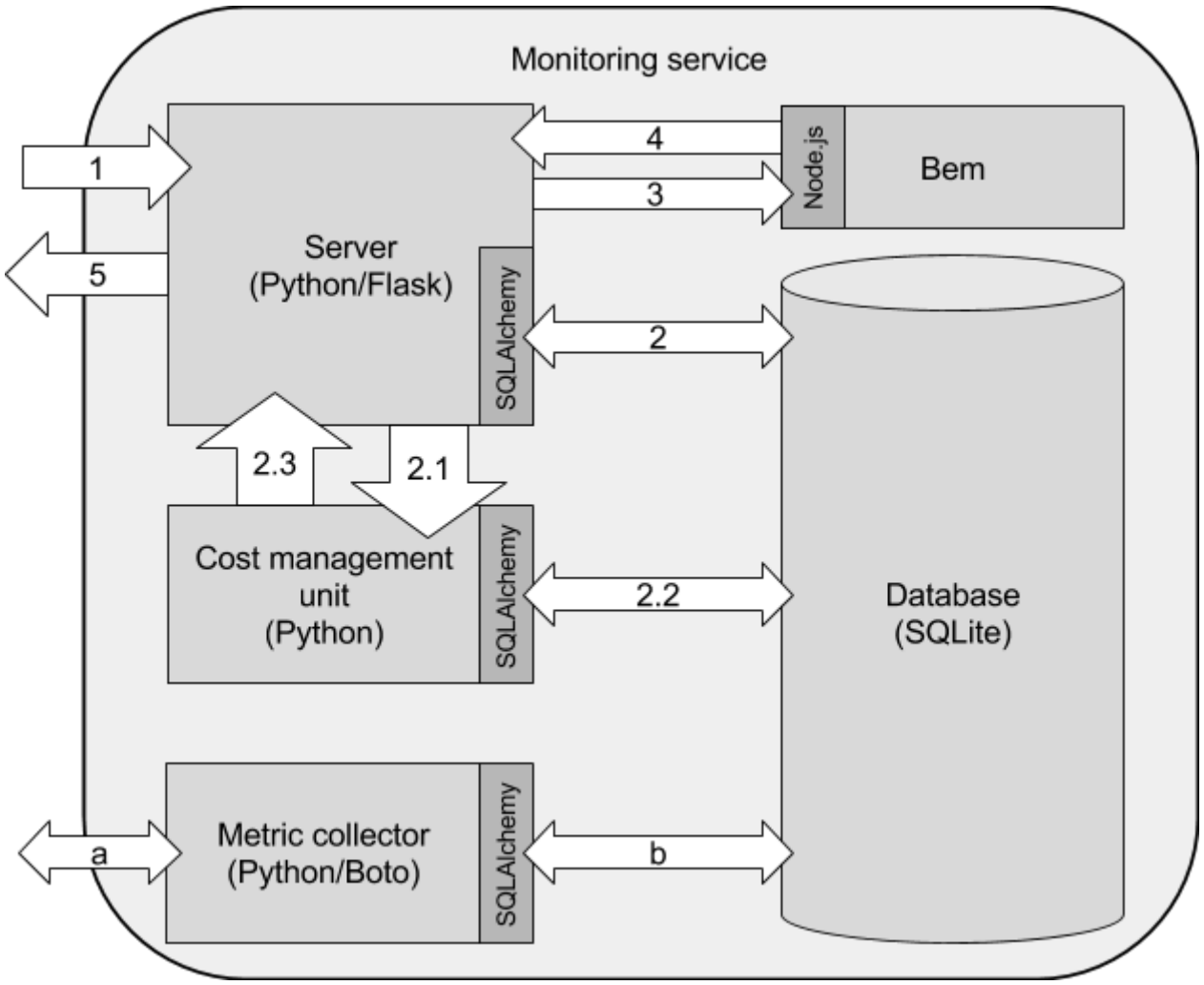

Рис. 3: Внутреннее устройство программного комплекса

Программный комплекс имеет две невзаимодействующие непосредственно части: веб-сервис и сервис сбора метрик AWS. Опишем сначала цикл работы веб-сервиса [3]:

1. От потребителя приходит запрос;
2. Из базы данных посредством SQLAlchemy достаются релевантные для пользователя и запроса данные, необходимые для заполнения страницы — например, список инстансов пользователя, доступные метрики и тарифные планы. Если же пользователь запрашивает график метрики, то в зависимости от типа этого графика происходит следующее: Если запрашивался график метрики производительности, сервер посредством SQLAlchemy запрашивает из базы данных

соответствующую метрику для соответствующего инстанса на соответствующем временном интервале;

2.1. Если же запрашивался график метрики стоимости, сервер запрашивает у модуля управления стоимостью соответствующую метрику стоимости;

2.2. Модуль управления стоимостью делает запрос к базе данных посредством SQLAlchemy, запрашивая метрику утилизации ЦПУ для соответствующего инстанса на соответствующем временном интервале. Также модуль управления стоимостью вычисляет оптимальный план на данном интервале для данного инстанса;

2.3. Модуль управления стоимостью возвращает серверу вычисленную метрику стоимостью с информацией о рекомендации насчет оптимального плана эксплуатации инстанса;

3. Обладая всеми необходимыми данными для генерации страницы, сервер обращается к генератору страниц;
4. Сервер получает готовую HTML-страницу;
5. Сервер отдает страницу пользователю.

Модуль сбора метрик при запуске сервера инициализирует нити исполнения, каждая из которых отвечает за сбор всех метрик для фиксированного для нее инстанса [4]. Цикл работы такой нити исполнения:

a. Нить делает запрос к Cloudwatch с соответствующими ключами доступа и для соответствующего ей инстанса, в ответ получая метрики производительности;
b. Нить сохраняет полученные метрики производительности в базу данных посредством SQLAlchemy.

4.2.3. Устройство базы данных и работа с ней

В этом пункте будет описана схема устройства таблиц в базе данных сервера и связи между ними.

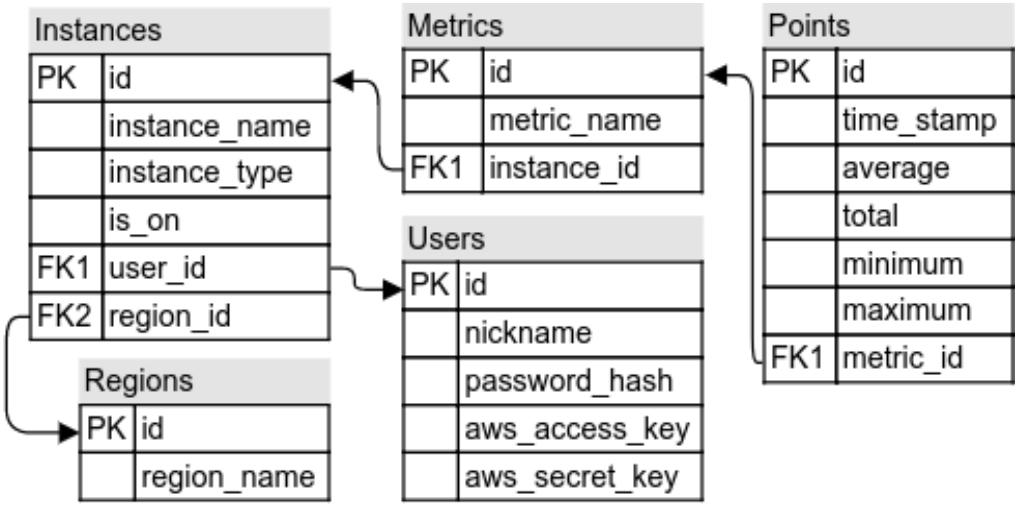

Рис. 4: Схема устройства базы данных

В базе данных сервера существует 5 таблиц:

1. Instances — таблица, в которой перечислены все инстансы. Ее поля:
    - instance_name — имя инстанса;
    - instance_type — тип текущего тарифного плана, или, другими словами, тип инстанса;
    - is_on — включен или выключен мониторинг данного инстанса в данный момент времени (программный комплекс поддерживает включение/выключение мониторинга инстансов);
    - user_id — идентификатор пользователя-владельца инстанса. Является ссылкой на таблицу users;
    - region_id — идентификатор региона видимости инстанса. Данное поле используется для запросов к CloudWatch;
2. Regions — таблица регионов видимости. Суть в том, что инстансы обладают некоторым регионом видимости, который нужно упоминать

при запросах к CloudWatch. Таблица содержит лишь одно поле — имя региона.

3. Users — таблица пользователей, зарегистрированных на сервере. Ее поля:
    - nickname — имя, под которым авторизуется пользователь на сервере;
    - password_hash — хеш пароля пользователя; хранится именно хеш пароля, ведь его достаточно для авторизации, но при этом устраняется уязвимость хранения паролей в базе данных в открытом виде;
    - aws_access_key — первая часть ключа доступа, используемого для авторизации запросов к CloudWatch;
    - aws_secret_key — вторая часть ключа доступа, используемого для авторизации запросов к CloudWatch.
4. Metrics — таблица метрик инстансов, собираемых модулем сбора метрик и хранимых в базе данных. Поля таблицы:
    - metric_name — имя метрики;
    - instance_id — идентификатор инстанса, к которому относится данная метрика.
5. Points — таблица точек метрики. Суть в том, что метрика описывает зависимость некоторой функции от времени. Мы же эту функцию приближаем ломаной, задавая лишь конечное число точек на отрезке. Соответственно, каждая строка этой таблицы репрезентует одну из точек какой-либо ломаной, соответствующей какой-либо метрике. Поля таблицы:
    - time_stamp — момент во времени, соответствующий данной точке;

- average — поскольку мы пытаемся приблизить непрерывную зависимость ломаной, получаем, что каждая точка ломаной репрезентует некоторый отрезок значений репрезентуемой функции. Данное поле говорит нам, каким было бы значение ломаной в этой точки, если бы мы приближали ломаную на отрезке ее средним значением;
- total — величина, актуальная для “квантуемых” метрик, например, для числа обращений к диску. В случае метрики числа обращений к диску именно в этой величине будет записано суммарное число обращений к диску на отрезке времени;
- minimum — результат приближения метрики на отрезке ее минимальным значением на отрезке;
- maximum — результат приближения метрики на отрезке ее максимальным значением;
- metric_id — идентификатор метрики, которой принадлежит данная точка данных. Напомним, что точка связывается с инстансом, на котором она измерялась, именно посредством объекта метрики, для которой определена принадлежность инстансу.

4.2.4. Устройство модуля управления стоимостью

Остановимся более подробно на модуле управления стоимостью. Основную часть этого модуля составляет следующий класс (приведен его псевдокод):

```
Class CostManager:
   fetch_metrics(instance_name, metric_name, time_period)
   calc_cost_metric(instance_name, instance_type, time_period)
   optimize(instance_name, time_period)
```

Рассмотрим действие этих методов и разберем поподробнее их устройство:

1. `fetch_metrics` — этот метод вызывается, когда модулю необходимо извлечь данные о производительности какого-либо инстанса в каком-либо периоде. Именно этот метод работает с базой данных. Действует он следующим образом:
   1.1. Вначале метод осуществляет join таблиц Points и Metrics по признаку Points.metric_id = Metric.id;
   1.2. Затем метод осуществляет join полученной таблицы с таблицей Instances по признаку Metric.instance_id = Instance.id. В итоге получается таблица точек метрик, про каждую из которых известна принадлежность метрике и инстансу;
   1.3. После этого из полученной таблицы отбираются точки, соответствующие нужной метрике, нужному инстансу и лежащие в нужном временном интервале;
   1.4. Полученный результат возвращается.
2. `calc_cost_metric` — этот метод производит расчет метрики стоимости для данного инстанса на данном промежутке времени и для данного тарифного плана. Именно этот метод вызывается, когда пользователь запрашивает график зависимости стоимости эксплуатации от времени. При таком способе использования сервер либо указывает в качестве тарифного плана либо текущий план этого инстанса, либо выбранный пользователем план для гипотетических расчетов. Действует он следующим образом:
   2.1. В начале своей работы этот метод вызывает `fetch_metrics` для соответствующего инстанса, метрики и временного интервала;
   2.2. Затем метод сортирует полученные точки по времени в порядке возрастания;

2.3. Метод вычисляет поправочный множитель стоимости, связанный с тем, что имея на одном тарифном плане один процент оккупации ресурсов, на другом плане при той же загрузке процент оккупации будет другим. Именно этот поправочный множитель и называется поправочным множителем стоимости;

2.4. Затем метод обходит точки метрики в порядке возрастания временной отметки. Имея новую точку метрики производительности, он генерирует новую точку метрики стоимости, прибавляя к предыдущей точке метрики стоимости произведение средней нагрузки ЦПУ, отрезка времени, который аппроксимируется точкой, стоимости часа полной утилизации ЦПУ и поправочного множителя стоимости;

2.5. После завершения обхода точек метрики производительности полученная метрика стоимости возвращается.

3. `optimize` — этот метод вызывает сервер для того, чтобы определить из всех наличных планов для данного инстанса на данном промежутке времени оптимальный. Действует он следующим образом:

3.1. Для каждого доступного тарифного плана определяется, является ли предлагаемая им производительность достаточной для поддержки функционирования в текущем режиме данного инстанса;

3.2. Для каждого допустимого тарифного плана метод вычисляет стоимость эксплуатации в данном режиме с данным планом. Это происходит аналогично работе функции `calc_cost_metric`;

3.3. Из всех результатов метод фиксирует минимальный, и возвращает соответствующий минимизирующий план стоимости.

5. Эксперимент

5.1. Описание стенда

Для проведения эксперимента, были задействованы следующие объекты:

1. Локальная машина (характеристики: intel core i7 3537U, 6 gb RAM);
2. EC2 инстанс планом t2.micro.

5.2. Проведение эксперимента

Порядок действий был следующим:

1. Из репозитория на локальную машину был клонирован и развернут проект;
2. Были запущены обе части проекта — как сервер, так и генератор страниц;
3. С помощью AWS EC2 был создан t2.micro инстанс;
4. На инстансе была размещена тестовая нагрузка — расчет числа Пи в течении отрезка времени;
5. Был создан веб-интерфейс включения и выключения такой нагрузки;
6. С помощью IAM были созданы ключи доступа, обладающие административными правами;
7. Пользуясь веб-интерфейсом разработанного приложения, был зарегистрирован пользователь;
8. С помощью веб-интерфейса этому пользователю были добавлены ключи доступа, выданные IAM на шаге 6;

9. Через веб-интерфейс был запущен мониторинг этого инстанса (после добавления ключей инстанс автоматически опросил AWS и получил информацию о доступных для мониторинга с помощью данного ключа доступа инстансах);
10. На тестовом инстансе была спустя какое-то время запущена тестовая нагрузка;
11. С помощью веб-интерфейса программного комплекса запрашивалось состояние метрик связанных с этим инстансом.

5.2. Результаты

Результаты работы программного комплекса можно представить снимками экрана и сравнительным графиком стоимостей владения.

Соответственно, как видно по приведенным снимкам, комплекс корректно распознал скачки в использовании ЦПУ во время как пуска нагрузки, так и ее выключения. Также заметно, что зависимость суммарной стоимости владения более стремительно возрастала на этом отрезке.

Можно сделать заключение о корректной работе модуля оптимизации в силу фактической оптимальности предложенного модулем плана.

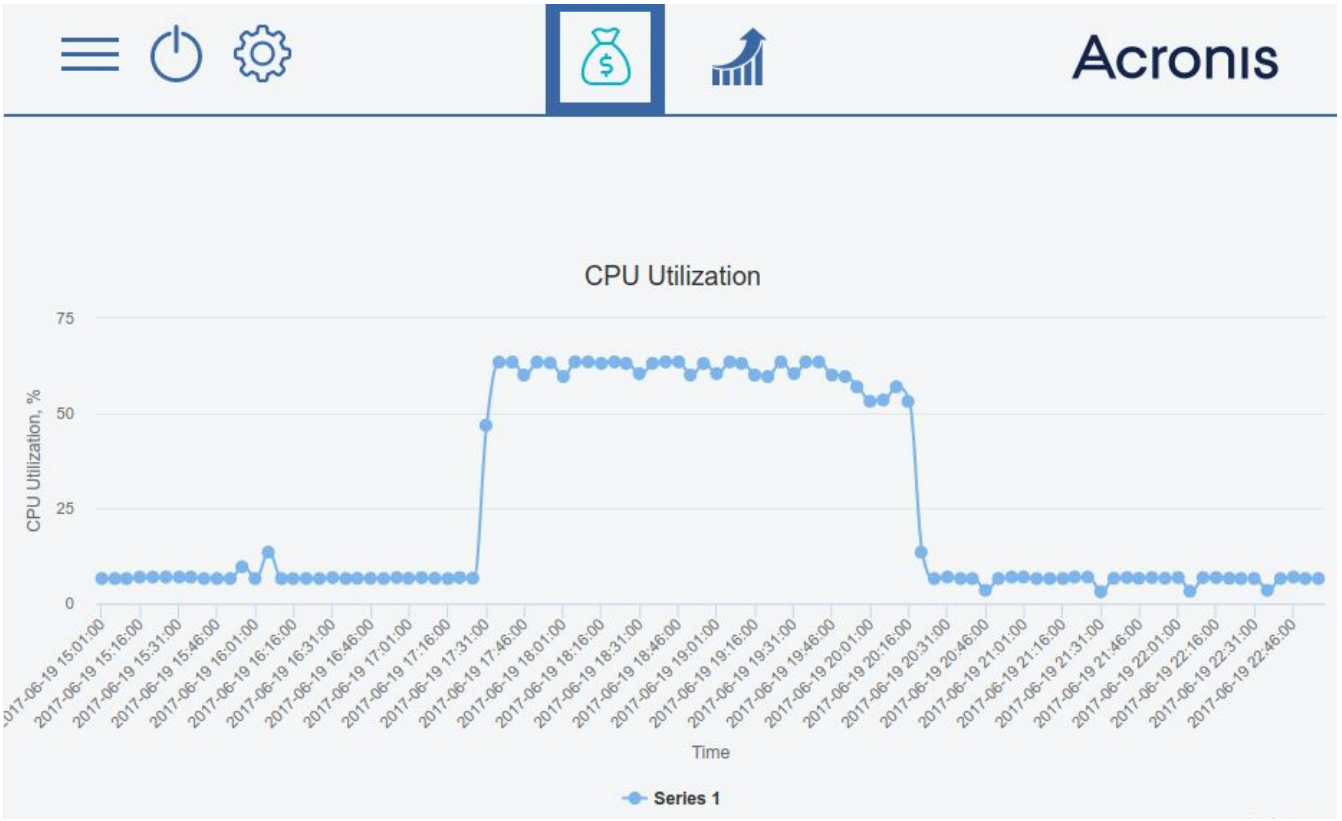

Рис. 5: Результат измерения нагруженности ЦПУ с помощью программного комплекса в ходе эксперимента с включением и выключением нагрузки

Рис. 6: Результат работы модуля управления стоимостью — график стоимости обладания облачным ресурсом

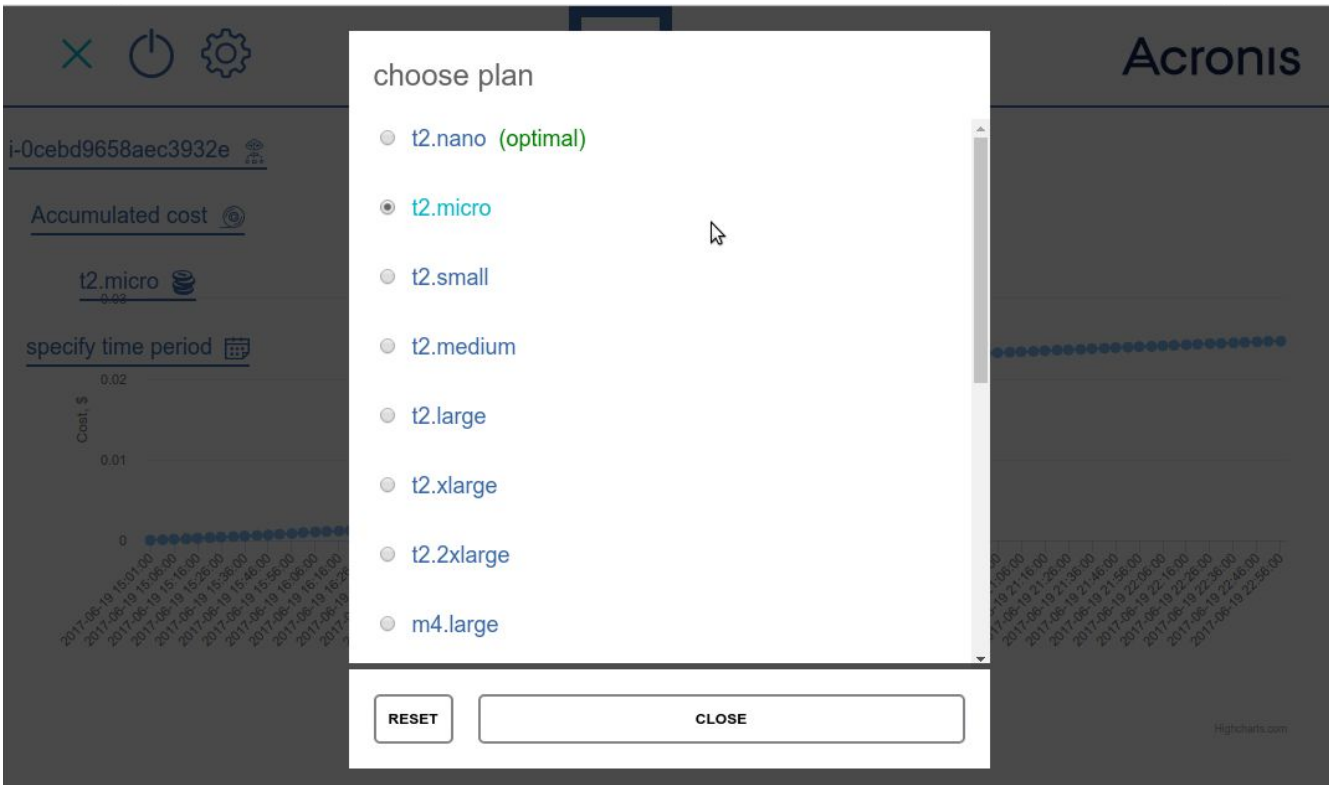

Рис. 7: Результат работы модуля управления стоимостью — рекомендация по оптимизации тарифного плана

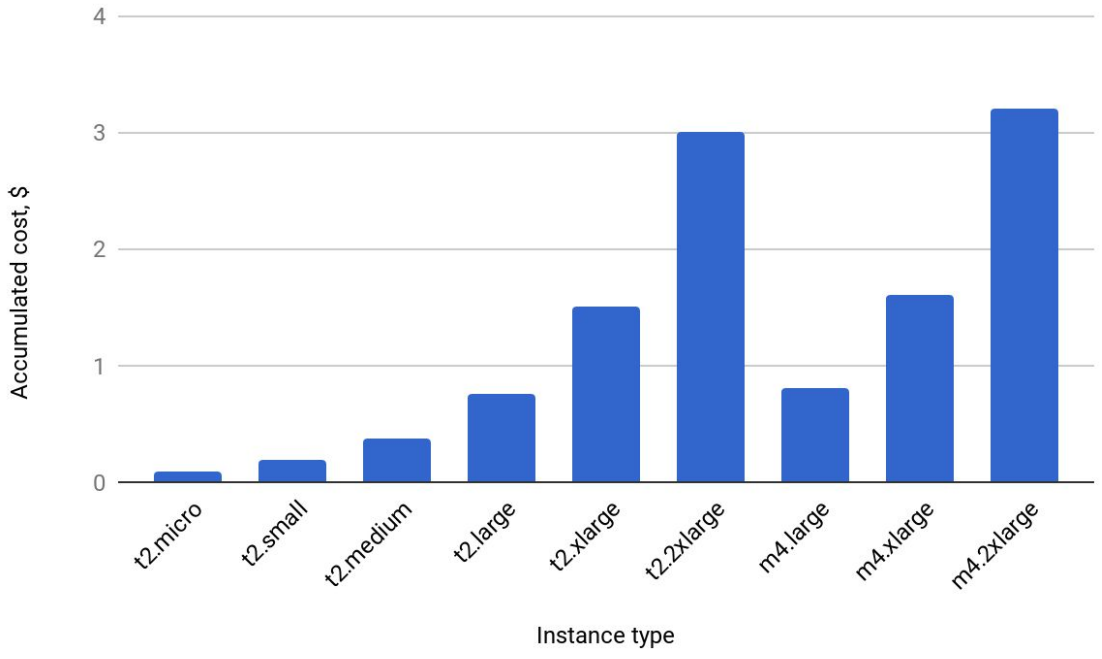

Рис. 8: Сравнение стоимости владения при разных тарифных планах; по графику видно оптимальность плана t2.micro

6. Выводы

В ходе дипломной работы был исследован рынок продуктов, предоставляющих услуги мониторинга производительности облачных инфраструктур.

Во время выполнения выпускной квалификационной работы была сформулирована и рассмотрена задача оптимизации стоимости владения облачными сервисами.

Был изучен круг технологий, используемых для управления и мониторинга облачными сервисами. Кроме того, был изучен и применен на практике набор технологий, используемых для работы веб-сервисов и их взаимодействия с облачными сервисами. Была разработана архитектура веб-сервиса мониторинга и затем она была реализована. В результате был получен веб-сервис, осуществляющий мониторинг облачной инфраструктуры.

Был предложен алгоритм, оптимизирующий стоимость владения облачными сервисами. Данный алгоритм был реализован в рамках дипломного проекта и протестирован на экспериментальном стенде. В результате был получен веб-сервис, способный осуществлять автоматический мониторинг и предоставлять комплекс советов по оптимизации стоимости владения облачными сервисами.

При помощи тестирования была выявлена работоспособность и эффективность как предложенного алгоритма, так и системы мониторинга и оптимизации стоимости владения в целом. Кроме того, была подтверждена теоретическая модель системы мониторинга.

Полученный в результате выполнения дипломного проекта прототип может являться основой для расширения продуктовой линейки компании Acronis.

Список источников

1. *Peter Mell, Timothy Grance*, The NIST Definition of Cloud Computing
2. *Roy Thomas Fielding*, Architectural Styles and the Design of Network-based Software Architectures
3. *Byron Ellis*, Real-Time Analytics
4. *Sam Newman*, Building Microservices
5. *Miguel Grinberg*, Flask Web Development: Developing Web Applications with Python
6. Официальная документация Flask http://flask.pocoo.org/docs/0.12/
7. Официальная документация CloudWatch API http://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/
8. Официальная документация Boto 3 https://boto3.readthedocs.io/
9. Официальная документация SQLAlchemy http://docs.sqlalchemy.org/