Министерство образования и науки Российской Федерации

Федеральное государственное образовательное учреждение высшего образования
”Алтайский государственный технический университет им. И. И. Ползунова”

Факультет информационных технологий
наименование подразделения

Кафедра прикладной математики
наименование кафедры

Отчет защищен с оценкой_____
“____”_____2017 г.
Руководитель от вуза
_____ / Старолетов С.М.
подпись *Ф. И. О.*

ОТЧЕТ

О научно-исследовательской работе
Разработка горизонтально масштабируемой системы легковесных виртуальных машин
общая формулировка задания

в Soft-logic
наименование организации

Студент гр. 8ПИ-51_____ Борисов В.В.____
индекс группы *подпись* *Ф. И. О.*

Руководитель от организации _____ _____
подпись *Ф. И. О.*

Руководитель от университета _____ Старолетов С.М.
подпись *Ф. И. О.*

2017 г.

Задание и календарный план практики (научно-исследовательской работы)

ФГБОУ ВО “Алтайский государственный технический университет им. И. И. Ползунова”
Кафедра прикладной математики

УТВЕРЖДАЮ
Зав. кафедрой_____С.А. Кантор
“____”_____2017 г.

ЗАДАНИЕ

По научно-исследовательской работе

студенту группы 8ПИ-51_____Борисова В.В._____
фамилия, имя, отчество

__09.04.04 «Программная инженерия»
код и наименование направления

База практики(научно-исследовательской работы)_____Soft-logic___
наименование организации

Способ проведения практики _____*стационарная* _____
стационарная, выездная, другие

Срок практики **с 09.01.17 г. по 19.03.17**

Разработка микросервиса под управлением операционной системы IncludeOS
общая формулировка задания

Календарный план практики

Наименование задач (мероприятий), составляющих задание	Дата выполнения задачи (мероприятия)	Подпись руководителя практики от организации
Постановка задачи	09.01.17 г.	
Обзор предметной области	09.01.17 г. - 19.01.17 г.	
Обзор имеющихся разработок по теме, их преимуществ и недостатков	20.01.17 г. - 03.02.17 г.	
Разработка web сервиса	04.02.17 г. - 19.03.17 г.	

Срок представления работы к защите_____**20.03.17 г.**_____

Руководитель практики от вуза
_____Старолетов С.М., доцент каф. ПМ

Содержание

Введение

Местом для прохождения практики была выбрана компания «Soft-logic»[1]. Компания Soft-logic основана в 2008 году. Специализация компании - разработка ПО для платежных систем, банков и расчетных центров.

Миссия компании - внедрение инновационных платежных решений в различных отраслях промышленности и финансового сектора, разработка и внедрение комплексных решений: "Процессинговый центр Pay-logic", SmartKeeper, "Расчетный центр Pay-logic", Фискальный сервер, а также решений для оплаты товаров и услуг в сети Интернет: Платформа мобильной коммерции, мобильные клиенты Android и IOS.

В настоящее время на решениях Soft-logic работает 54 коммерческих проекта в России, а также странах ближнего и дальнего зарубежья.

Флагманским продуктом компании является платежная платформа «Процессинговый центр Pay-logic». Данный программный комплекс предназначен для организации приема платежей в пользу любых поставщиков услуг через платежные терминалы, персональный компьютер, pos-терминал, мобильные устройства и интернет.

Сегодня Soft-logic — это самостоятельная, динамично развивающаяся компания, специализирующаяся на разработке комплексных платежных решений.

Большинство специалистов работают в этой сфере с начала 2000 годов, с момента зарождения терминального бизнеса в РФ.

Накопленные знания и опыт в области платежных решений, штат из высококвалифицированных специалистов, а также репутация разработчика качественных программных решений позволяют говорить о Soft-logic как о надежном партнере, предоставляющем качественные решения в финансовой сфере.

Во время прохождения практики была поставлена задача о разработке

горизонтально масштабируемой системы легковесных виртуальных машин, а также о запуске и интегрировании в тестовое окружение web сервисов под управлением одномодульной операционных системы IncludeOS [2]. Сервер в горизонтально масштабируемой системе должен принимать get, set и put запросы от клиентов мобильного платёжного приложения под управлением Android и Ios, передавать полученные данные в одномодульные операционные системы IncludeOS, которые, в свою очередь, будут производить обработку полученных данных и возвращать результат через сервер клиентам.

IncludeOS предоставляет минимально необходимое, самодостаточное окружение, которое взаимодействует непосредственно с гипервизором и предоставляет загрузчик, ядро, минимальный набор библиотек и модулей, достаточный для выполнения кода на языке C++, написанный с использованием стандартной библиотеки классов. Окружение компонуется с предназначенным для выполнения приложением и оформляется в виде загрузочного образа виртуальной машины, образуя готовый облачный сервис.

1 Описание предметной области

1.1 Микросервис

Во многих IT компаниях для обработки однотипной информации в больших объёмах используют микросервисы [3]. Архитектурный стиль микросервисов — это подход, при котором единое приложение строится как набор небольших сервисов, каждый из которых работает в собственном процессе и коммуницирует с остальными, используя легковесные механизмы, как правило HTTP. Эти сервисы построены вокруг бизнес-потребностей и развертываются независимо с использованием полностью автоматизированной среды. Существует абсолютный минимум централизованного управления этими сервисами [4]. Сами по себе эти сервисы могут быть написаны на разных языках и использовать разные технологии хранения данных.

Сервис [5] представляет из себя программный продукт, выполняющий узконаправленные действия, такие как: конвертирование аудио файла из одного формата в другой, поиск на изображении лиц, построение панорамного изображения из нескольких, выполнение математических расчётов и т.д. Сервис запускается в реальной операционной системе (Windows или Unix подобных) на виртуальной машине. Для запуска, контролирования виртуальных машин, а также масштабирования всей системы используют гипервизор.

1.2 Гипервизор

Гипервизор [6] или монитор виртуальных машин — это, как правило, комплекс программ, обеспечивающий параллельное, одновременное выполнение нескольких операционных систем на одном и том же хост – компьютере.

Гипервизор в планируемой системе будет отвечать за запуск, остановку и контроль нагрузки виртуальных машин, например: если нагрузка падает и несколько виртуальных машин простаивают, то их можно отключить, и наоборот, если нагрузка возрастает и данное количество виртуальных машин не справляется, то запускаются ещё n-ое количество необходимых сервисов.

Сервисы на виртуальных машинах запускаются, как правило, под управлением операционных систем семейства Windows или Unix-подобных. Windows или Unix-подобные операционные системы на виртуальных машинах занимают достаточно большое количество ресурсов реального хост – компьютера, часто превосходящие затраты на выполнение самого сервиса, т. к. необходимо поддерживать работу самой операционной системы, в которой запущен сервис, а именно, поддерживать графический интерфейс [7] (для Windows), поддерживать работу неиспользуемых драйверов (видеокарты, драйвер принтера, звуковой карты и т. д.).

1.3 IncludeOS

В 2016 году на конференции CppCon [8] был представлен рабочий alpha прототип проекта одномодульной микро операционной системы IncludeOS [9]. IncludeOS предоставляет минимально необходимое, самодостаточное окружение, которое взаимодействует непосредственно с гипервизором и предоставляет загрузчик, ядро системы, минимальный набор библиотек, модулей и драйверов, достаточный для выполнения кода на языке C++, написанный с использованием стандартной библиотеки классов [10]. Окружение компонуется с предназначенным для выполнения приложением и оформляется в виде загрузочного образа виртуальной машины, образуя готовый облачный сервис. Из систем виртуализации, в которых могут работать подобные окружения, поддерживаются KVM/Linux, VirtualBox и QEMU. Схематичное представление построения загрузочного образа виртуальной машины с готовым микросервисом изображён на рисунке 1.

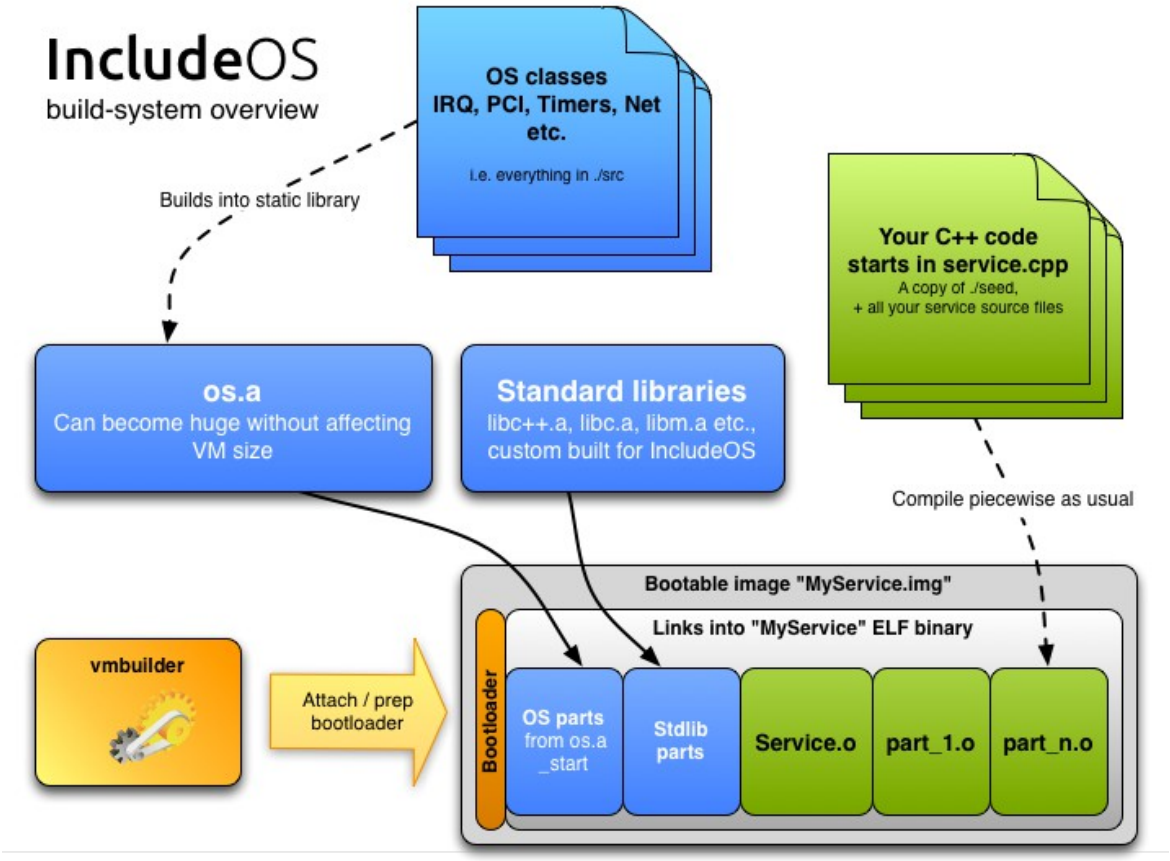

Рисунок 1.1 - Построение образа виртуальной машины

По затратам машинных ресурсов IncludeOS можно сравнить с встраиваемой операционной системой.

Встраиваемые операционные системы [11] – операционные системы (ОС), предназначенные не для запуска на виртуальных машинах, а для управления специализированными устройствами (промышленные контроллеры, радиоаппаратура и т.д.) и, вследствие этого, способные работать в условиях ограниченных ресурсов (малые объёмы памяти, недостаток вычислительных мощностей и т.п.) и в необслуживаемом режиме. Характерными особенностями встраиваемых ОС являются модульная структура, компактность, производительность, масштабируемость и повышенная отказоустойчивость.

Основным отличаем IncludeOS от встраиваемой операционной системы является то, что IncludeOS не предназначена для управления специализированными устройствами, хотя в её ядре есть модули, позволяющие взаимодействовать с USB и COM портами.

IncludeOS, в основном, разрабатывается для выполнение web сервисов. Суммарный размер библиотек и компонентов скомпилированной операционной системы составляет около 1 Мб [12]. В связи с тем что IncludeOS в большей степени не является встраиваемой операционной системой, и с тем, что гипервизоры в подавляющем большинстве случаев используют виртуальные машины под управлением операционных систем семейства Windows или Unix-подобных, проведём сравнение использования машинных ресурсов IncludeOS и операционной системы Ubuntu. На тестах с операционной системой Ubuntu, IncludeOS показал значительно меньшее потребление ресурсов реального хост – компьютера, а именно: занимаемое пространство на жёстком диске меньше в 300 раз; использование оперативной памяти меньше в 280 раз [13]; загрузка системы происходит менее чем за 0,5 секунды, для сравнения Ubuntu загружается в среднем за 10 секунд; нагрузка на CPU при работе экспериментального DNS - сервера, построенного на базе IncludeOS, оценивается в 5-20% по сравнению с

запуском того же исполняемого файла в Ubuntu, всё это объясняется тем, что в IncludeOS есть только необходимые элементы операционной системы для работы сервиса. Сравнительная диаграмма используемых ресурсов хост – компьютера для Ubuntu и IncludeOS изображена на рисунке 2.

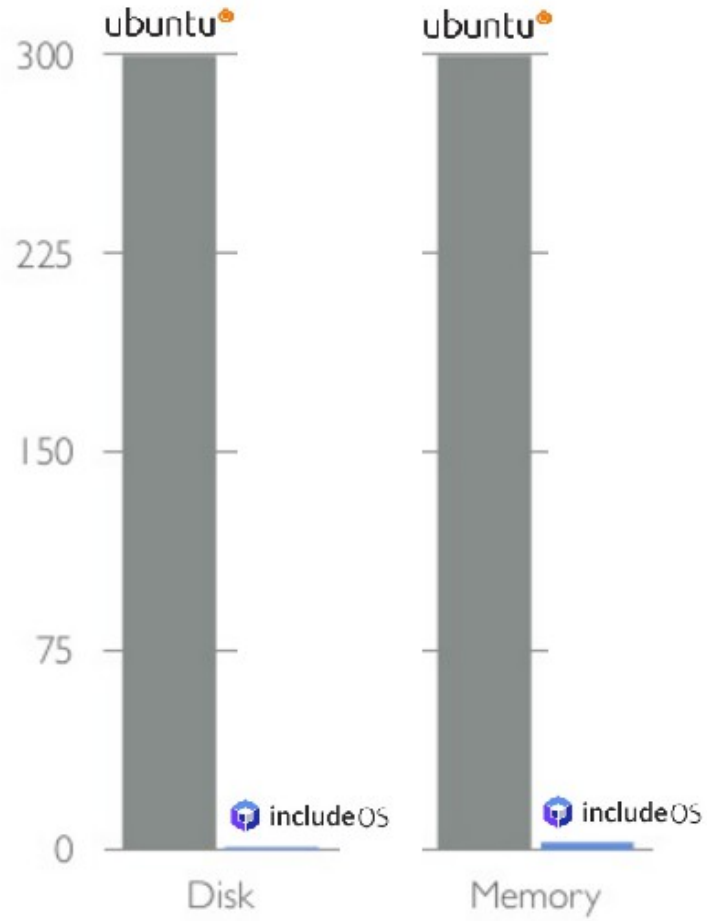

Рисунок 1.2 - Используемые ресурсы Ubuntu и IncludeOS

В связи с этим использование IncludeOS в качестве операционной системы для вируальной машины в настоящее время в разы эффективней, чем с любой другой операционной системой, т.к. на одних и тех-же аппаратных мощностях можно запускать в разы больше виртуальных машин.

Так как IncludeOS относительно новый проект и на данный момент находится в состоянии разработки, то поддерживаются только следующие системы виртуализации: KVM/Linux, VirtualBox и QEMU, а это значит, что запустить виртуальные машины под управлением данной операционной систеой, для работы сервисов на наиболее известных гипервизорах, таких как VMware ESX Server [14], XenServer [15], Citrix [16] и других не получится.

Исходя из изложенного, отказываться от преимуществ IncludeOS будет ошибкой, в связи с этим было принято решение о разработке собственного менеджера виртуальных машин с возможностью запуска, остановкой, просмотра состояния нагрузки виртуальных машин и т. д.

Планируется разработка программного комплекса, в который будет входить: клиентский компонент системы для формирования данных для обработки сервисами (передача данных для обработки и получение результата обработки); сервер для мониторинга виртуальных машин, который будет в зависимости от загруженности виртуальных машин принимать решения о запуске или остановки сервисов, а также будет посредником для передачи данных для обработки от клиента сервисам; контроллер будет установлен на хост-машину и предназначен для запуска, остановки, сбора информации о загрузки виртуальной машины; консоль администратора будет устанавливаться параллельно с клиентским компонентом для просмотра состояния загруженности виртуальных машин.

2 Обзор конкурирующих систем.

2.1 Виртуализация на основе продуктов VMware.

ESX и ESXi являются встроенными гипервизорами и ставятся непосредственно «на железо» [17], то есть при установке не требуют наличия на машине установленной операционной системы.

ESX Server представляет собой гипервизор типа 1 [18], который создает логические пулы системных ресурсов, позволяя множеству виртуальных машин разделять одни и те же физические ресурсы.

ESX Server - это операционная система, которая функционирует как гипервизор и работает непосредственно на оборудовании сервера. ESX Server добавляет уровень виртуализации между аппаратной частью системы и виртуальными машинами, превращая оборудование системы в пул логических вычислительных ресурсов, которые ESX Server может динамически выделять любой гостевой операционной системе. Операционные системы, работающие в виртуальных машинах, взаимодействуют с виртуальными ресурсами, как если бы это были физические ресурсы.

Ключевые компоненты архитектуры ESX Server:

1. Уровень виртуализации ESX Server: отделяет основные физические ресурсы от виртуальных машин
2. Менеджер ресурсов: создает виртуальные машины и обеспечивает их ресурсами процессора, памяти, сети и дисковой подсистемы
3. Служебная консоль: управляет установкой, настройкой, администрированием, устранением неисправностей и техническим обслуживанием ESX Server
4. Компоненты аппаратного интерфейса, в том числе драйверы устройств.

Возможности [19]:

1. ESX Server использует механизм пропорционального распределения

ресурсов процессоров, памяти и дисков, когда несколько виртуальных машин претендуют на одни и те же ресурсы

2. ESX Server может выделять емкость процессора на основе разделения времени, предотвращая возможность монополизации ресурсов процессора какой-либо виртуальной машиной
3. ESX Server выделяет память в зависимости от нагрузки виртуальной машины и заданного минимума. Например, если виртуальной машине недостаточно памяти, ESX Server может занять память у одной виртуальной машины, передать ее другой виртуальной машине, и в случае необходимости, вернуть эту память первоначальной виртуальной машине
4. ESX Server управляет пропускной способностью сети с формированием сетевого трафика. Разделение ресурсов сети осуществляется с помощью выделения маркеров или в зависимости от потребления, исходя из средней или максимальной потребности в пропускной способности виртуальной машины.

Плюсы [20]:

1. С технической точки зрения самый продвинутый гипервизор
2. Бесплатен (можно скачать с сайта Vmware)
3. Поддерживает множество ОС внутри себя (Windows, Linux, BSD, Solaris, и т.д.)
4. Легко установить и настроить.

Минусы:

1. Продвинутые инструменты администрирования платные
2. Может установиться только на ограниченное количество серверов
3. Занимает большое количество аппаратных ресурсов
4. Не поддерживает IncludeOS.

2.2 Виртуализация на основе продуктов Xen.

Xen [21] представляет собой гипервизор типа 1, который создает логические пулы системных ресурсов, позволяя множеству виртуальных машин разделять одни и те же физические ресурсы.

Xen – это гипервизор, который работает непосредственно на оборудовании системы. Xen добавляет уровень виртуализации между аппаратной частью системы и виртуальными машинами, превращая оборудование системы в пул логических вычислительных ресурсов, которые Xen может динамически выделять любой гостевой операционной системе. Операционные системы, работающие в виртуальных машинах, взаимодействуют с виртуальными ресурсами, как если бы это были физические ресурсы.

Xen основан на паравиртуализации [22]; он требует внесения в гостевые операционные системы изменений, направленных на поддержку операционной среды Xen. Тем не менее, пользовательские приложения и библиотеки никакой модификации не требуют.

Внесение изменений в операционную систему необходимо для того, чтобы:

1. Xen мог заменить операционную систему в качестве наиболее привилегированной программы
2. Xen мог использовать более эффективные интерфейсы (например, виртуальные блочные устройства и виртуальные сетевые интерфейсы) для эмуляции устройств - это повышает производительность.

Xen (с помощью стека управления) поддерживает миграцию гостевых виртуальных машин по сети. Миграция паравиртуальных машин поддерживается с версии Xen 2, а HVM – с версии 3. Миграция может происходить с выключением гостевой системы, или прямо в процессе работы, так называемая «живая» миграция без потери доступности.

Необходимо, чтобы оба физические сервера Xen видели одно и то же хранилище, на котором находятся данные виртуальной машины. Общее

хранилище может быть организовано на основе различных технологий SAN или NAS, например Fibre Channel, iSCSI или DRBD.

Четвертое поколение продуктов виртуализации компании XenSource [23] на основе гипервизора Xen включает в себя три версии платформы:

XenExpress - бесплатное стартовое издание системы виртуализации, включающее в себя возможность размещения нескольких виртуальных машин в пределах одного физического сервера. Эта версия является скорее ознакомительной и хорошо подходит для домашних пользователей и небольших компаний, планирующих внедрение виртуализации.

XenServer - издание платформы для сектора SMB (Small and Medium Business), обеспечивающее решение основных задач по консолидации виртуальных серверов на нескольких физических хостах.

XenEnterprise - наиболее полная версия платформы виртуализации, включающая в себя, помимо возможностей XenServer, также и необходимые средства по агрегации ресурсов, распределению нагрузки, живой миграции и обеспечению высокой доступности.

Плюсы:

1. Поддерживает множество ОС внутри себя
2. Бесплатен
3. Поддерживает достаточно большое количество серверов.

Минусы:

1. Продвинутые инструменты администрирования платные
2. Занимает большое количество аппаратных ресурсов
3. Может установиться только на ограниченное количество серверов
4. Не поддерживает IncludeOS.

2.3 Виртуализация на основе продуктов Hyper-V.

Hyper-V существует в двух вариантах [24]:

1. Как отдельный продукт Microsoft Hyper-V Server
2. Как роль Windows Server 2012, Windows Server 2008 R2, Windows Server 2008 и x64 битная версия Windows 8 Pro.

Отдельная версия Hyper-V Server является бесплатной. Является базовым («Server Core») вариантом Windows Server 2008, то есть включает в себя полную функциональность Hyper-V; прочие роли Windows 2008 Server отключены, также лимитированы службы Windows. Бесплатная 64-битная Core-версия Hyper-V ограничена интерфейсом командной строки (CLI PowerShell), где конфигурация текущей ОС, физического аппаратного и программного оборудования выполняется при помощи команд оболочки. Новое меню интерфейса управления позволяет выполнить простую первичную конфигурацию, а некоторые свободно распространяемые скрипты расширяют данную концепцию.

Администрирование и конфигурирование виртуального сервера осуществляется при помощи ПО, установленного на ПК под управлением Windows Vista, Windows 7 или Windows 2008 Server с установленным дополнением для администрирования Hyper-V из MMC [25]. Другим вариантом администрирования/конфигурирования сервера Windows 2008 Core является использование удаленной Windows или Windows Server при перенаправлении (некоторой) консоли управления (MMC), указывающей на Core Server.

В родительском разделе в пространстве пользователя имеются:

1. WMI-провайдеры – управление виртуальными машинами локально и удаленно
2. Сервис управления виртуальными машинами (VMMS) – управляет виртуальными машинами
3. Рабочие процессы виртуальных машин (VMWP) – процессы, в которых

выполняются все действия виртуальных машин – обращение к виртуальным процессорам, устройствам, и т.д

В пространстве ядра родительского раздела выполняются:

1. Драйвер виртуальной инфраструктуры (VID) – осуществляет управление разделами, а так же процессорами и памятью виртуальных машин
2. Провайдер сервисов виртуализации (VSP) – предоставляет специфические функции виртуальных устройств (так называемые "синтетические" устройства) посредством VMBus и при наличии интеграционных компонентов на стороне гостевой ОС
3. Шина виртуальных устройств (VMBus) – осуществляет обмен информацией между виртуальными устройствами внутри дочерних разделов и родительским разделом
4. Драйверы устройств – только родительский раздел имеет прямой доступ к аппаратным устройствам, и потому именно внутри него работают все драйверы
5. Windows Kernel – собственно ядро хостовой ОС.

Плюсы:

1. Бесплатный
2. Хорошо подходит для виртуализации ОС от Microsoft
3. Большинство продуктов Microsoft поддерживают работу в виртуальной среде Hyper-V
4. Может установиться на любой сервер.

Минусы:

1. Плохо подходит для виртуализации ОС не от Microsoft (т.е. не Windows)
2. Продвинутые инструменты администрирования платные
3. Занимает большое количество аппаратных ресурсов
4. Не поддерживает IncludeOS.

3 Архитектура IncludeOS

3.1 Принцип нулевых издержек

Для любого сервиса, предназначенного для масштабного развертывания на нескольких виртуальных машинах, важно, чтобы каждая такая машина использовала минимальное количество ресурсов [26].

В отличие от классических операционных систем, которые включают в себя как можно больше функций, IncludeOS стремится к минимальному набору используемых функций по умолчанию в том смысле, что служба не должна быть включена по умолчанию, если она явно не используется. Это соответствует принципу нулевых издержек, например, C++: «Вы не должны платить за то, что не используете» [27].

3.2 Статически линкующиеся библиотеки и GCC-toolchain

Статически линкующиеся библиотеки - это архивы объектных файлов, которые подключаются к программе во время линковки. Статические библиотеки делают программу более автономной: программа, скомпонованная со статической библиотекой, может запускаться на любом компьютере, не требуя наличия этой библиотеки (она уже включена в бинарные файлы) [28].

GCC toolchain — набор созданных в рамках проекта GNU пакетов программ, необходимых для компиляции и генерации выполняемого кода из исходных текстов. Являются стандартным средством разработки программ и ядра ОС Linux [29].

Используется механизм для извлечения только тех библиотек, которые требуются от операционной системы. Данный механизм установлен по умолчанию во всех современные линкерах. Каждая часть операционной системы компилируется в объектный файл, например, ip4.o, udp.o, pci_device.o и т.д., которые затем объединяются в виде статической

библиотеки os.a. Линковщик автоматически выбирает необходимые библиотеки и только они пакуются в итоговый бинарный файл. Чтобы облегчить этот процесс был создан кастомный GCC toolchain.

IncludeOS не имеет программного загрузчика, поэтому нет классической main-функции с параметрами и возвращаемого значения. Вместо этого используется класс Service, а пользователь должен реализовать функцию Service::start, которая будет вызвана после завершения инициализации операционной системы [26].

3.3 Стандарт библиотек

Новые библиотеки от RedHat были выбраны в качестве стандарта реализации библиотек на языке C прежде всего потому, что они имеют достаточно малый размер, и разработаны чтобы полагаться только на несколько системных вызовов, а также они компилируются в статически линкуемые библиотеки. Таким образом, компоновщик будет включать в итоговый бинарный файл только те части стандартной библиотеки, которые будут использоваться или операционный системой или выполняемым сервисом.

Стандартная библиотека шаблонов (STL) C++ является довольно крупной и сложной. Так как контейнеры STL используют исключения, в IncludeOS они не используются внутри ядра, вместо этого используется библиотека шаблонов от Electronic Arts, а именно — EASTL [30]. Хоть данная реализация и содержит наиболее важные части STL, такие как string, streams, vector и map, этого не достаточно, так что некоторые компоненты были реализованы, а некоторые все еще находятся в стадии разработки. IncludeOS версия этой библиотеки будет доступна для исполняемых сервисов (т.е. доступна в пространстве сервиса). Будущие реализации IncludeOS будут включать в себя порт полнофункциональной реализации, такой как GCC libstdc++ [26].

3.4 Сетевой драйвер Virtio

Сетевой драйвер Virtio — это технология, улучшающая производительность виртуальных машин под управлением KVM. Данные драйвера реализуют «паравиртуализацию». В этом режиме работа устройства не полностью эмулируется гипервизором. Драйвер устройства в виртуальной машине знает о том, что он работает не с настоящим устройством, и взаимодействует с гипервизором, что обеспечивает большую производительность [31].

IncludeOS имеет только один драйвер устройства, а именно драйвер VirtioNet. Virtio 1.0 является стандартом Oasis [32], но ни один из гипервизоров, используемых во время разработки IncludeOS, не поддерживал ни одной из новых функций. Поэтому драйвер в настоящее время реализует только устаревшую функциональность Virtio, но разработка была сделана с будущей поддержкой Virtio 1.0. Текущая реализация использует шину PCI и не включила MSI-x (Message signaled interrupts) [26].

3.5 Асинхронный ввод-вывод и IRQ

В настоящее время все обработчики запросов на прерывание (IRQ) в IncludeOS только обновляют счетчик и откладывают дальнейшее обращение к основной работе приложения, когда есть время. Это избавляет от необходимости переключения контекста, также устраняя связанные с параллелизмом проблемы, такие как гонка данных. Процессор занят, поскольку все операции ввода-вывода асинхронны, поэтому блокирование не происходит. Это поощряет модель программирования на основе обратного вызова (Callback) [26].

Callback в программировании — передача исполняемого кода в качестве одного из параметров другого кода. Обратный вызов позволяет в функции исполнять код, который задаётся в аргументах при её вызове [33].

4 Постановка задачи

Целью данной работы является разработка программного комплекса, в который будут входить:

1. клиентский компонент системы для формирования запросов с данными для обработки.
2. сервер для распределения нагрузки между виртуальными машинами;
3. контроллер для управления виртуальнми машинами на хост-компьютере;
4. консоль администратора для просмотра информации о количестве и загруженности виртуальных машин;
5. дополнения сервиса виртуальной машины.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Проектирование механизмов взаимодействия компонентов системы.
   1.1. Проектирование протокола запроса на исполнение задачи.
   1.2. Проектирование протокола запроса на выдачу информации о состоянии виртуальной машины.
   1.3. Проектирование протокола управления состоянием виртуальной машины.
   1.4. Проектирование протокола запуска и остановки экземпляров виртуальных машин на хост-компьютере.
2. Разработка серверного компонента.
   2.1. Разработка подсистемы перенаправления запросов на исполнение задачи.
   2.2. Разработка подсистемы сбора данных о состояниях виртуальных машин.
   2.3. Разработка подсистемы принятия решений о горизонтальном масштабировании.

2.4. Разработка подсистемы отображения информации о текущей загрузке системы.

3. Разработка компонента виртуальных машин.
   3.1. Разработка веб-сервиса, исполняемого в виртуальной машине.
   3.2. Разработка дополнения веб-сервиса, исполняемого в виртуальной машине, обеспечивающее предоставление информации о загруженности системы.
4. Разработка контроллера виртуальной машины.
   4.1. Разработка внутреннего обработчика контроллера, для обработки вывода виртуальной машины.
   4.2. Разработка механизма перенаправления вывода консоли виртуальной машины во внутренний обработчик контроллера.
   4.3. Разработка механизма запуска, остановки и получения состояния о загруженности виртуальных машин.
5. Разработка клиентского компонента.
6. Разработка консоли администратора.

5 Подход к решению задачи

Запросы на обработку формируются в клиентском компоненте, после чего данные передаются на сервер. Сервер представляет из себя посредника между клиентским компонентом и виртуальными машинами, также сервер собирает информацию о загруженности виртуальных машин для их горизонтального масштабирования. Сервер выбирает наименее загруженную виртуальную машину и передаёт на неё данные для обработки.

За работу виртуальных машин отвечает контроллер, его задача — производить запуск, остановку виртуальных машин, а также сбор информации о их загруженности и передачи данной информации на сервер. Для корректной работы системы необходимо, чтобы постоянно была запущенна хотя бы одна виртуальная машина, в связи с этим контроллер при старте запускает первую виртуальную машину.

Для просмотра количества запущенных виртуальных машин и загруженности каждой виртуальной машины разработана консоль администратора.

Общая структура системы изображена на рисунке 5.1, подробное описание каждого модуля системы представлено ниже.

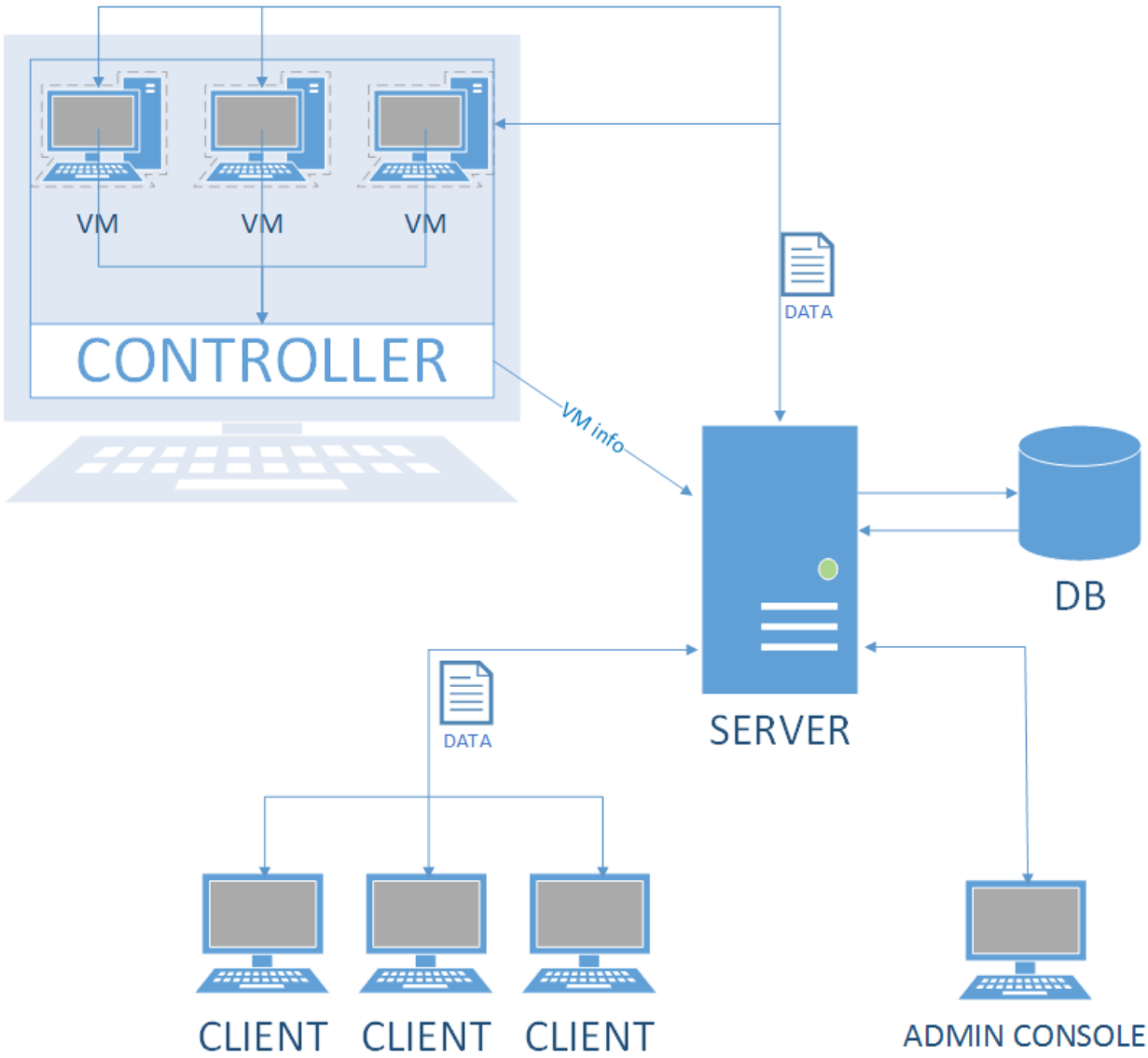

Рисунок 5.1 — Общая структура системы

5.1 Клиент

Для взаимодействия клиента с системой реализован «клиентский компонент» системы. С помощью данного компонента пользователь сможет предоставлять исходные данные для обработки сервисами, а также получать и просматривать обработанные данные. Данный компонент будет написан на языке программирования Java.

В связи с вышесказанным возникает ряд вопросов:

1. Как передавать данные для обработки и получать ответ?
2. Куда передавать данные для обработки?

Решением первого вопроса обмена данными между клиентским приложением и сервером был принят формат обмена данными json, а именно, json запросы на сервер и json ответы сервера.

Json [34] — простой текстовый формат обмена данными, основанный на JavaScript. Json базируется на двух основных концепциях:

1. Коллекция пар ключ/значение, практически во всех языках программирования реализованы данные типы данных
2. Упорядоченный список значений.

Основываясь на данных концепциях, такой формат данных независим от языка программирования.

Решением вопроса о передачи данных для обработки стала разработка серверного компонента системы, о котором будет написано ниже.

5.2 Сервер

Сервер будет использоваться для следующих возможностей:

1. Для принятия решений о запуске и остановки виртуальных машин на хост — компьютере
2. Для передачи исходных данных от клиентского компонента системы для обработки сервисами
3. Для получения обработанных данных от виртуальных машин и передачи этих данных клиентскому компоненту
4. Для сбора информации о состоянии загруженности виртуальных машин.

Сервер будет представлять из себя Java сервлет, принимающий запросы и возвращающий ответы для клиентского компонента, консоли администратора, контроллера и виртуальных машин.

В связи с вышесказанным возникает ряд вопросов:

1. Где будет запускается сервер?
2. В каком формате получать и передавать данные как от клиента, консоли администратора, так и от виртуальных машин?

3. Как следить за состоянием виртуальных машин?

Решением первого вопроса о запуске сервера было принять решение, что сервлет будет разворачиваться на GlassFish, так как GlassFish легко устанавливается и вместе с ним идёт web консоль администратора для настройки.

GlassFish [35] — сервер приложений, распространяемый под лицензией CDDL, с открытым исходным кодом, разработанный Sun Microsystems. В настоящее время спонсируется корпорацией Oracle. Актуальная версия платформы называется Oracle GlassFish Server.

В основу GlassFish легли части кода Java System Application Server [36] компании Sun и ORM TopLink. В качестве сервлет-контейнера в нём используется модифицированный Apache Tomcat, дополненный компонентом Grizzly, использующим технологию Java NIO.

Решением второго вопроса обмена данными между клиентским приложением и виртуальных машинами, как уже было описано в клиентском приложении, был принят формат обмена данными json.

Решением третьего вопроса о мониторинге состояний виртуальных машин было принято решение о хранении всей информации о состоянии виртуальных машин в базе данных. В качестве базы данных была выбрана база данных PostgreSQL [37].

PostgreSQL — свободно распространяемая объектно — реляционная система управления базами данных (СУБД).

Существует в реализациях для множества UNIX-подобных платформ, включая AIX, различные BSD-системы, PostgreSQL базируется на языке SQL и поддерживает многие из возможностей стандарта SQL:2011

Сильными сторонами PostgreSQL считаются:

1. Высокопроизводительные и надёжные механизмы транзакций и репликации
2. Расширяемая система встроенных языков программирования
3. Наследование

4. Легкая расширяемость.

Основные возможности:

1. Функции являются блоками кода, исполняемыми на сервере, а не на клиенте БД
2. Триггеры определяются как функции, инициируемые операциями вставки, удаления или изменения данных
3. Механизм правил представляет собой механизм создания пользовательских обработчиков не только операций вставки, удаления или изменения данных, но и операции выборки
4. Имеется поддержка индексов следующих типов: B-дерево, хэш, R-дерево, GiST, GIN
5. Поддерживается одновременная модификация БД несколькими пользователями с помощью механизма Multiversion Concurrency Control
6. Поддерживается большой набор встроенных типов данных

5.3 Контроллер

Запуск контроллера виртуальных машин планируется на Unix подобной операционной системе, так как сборку образа IncludeOS с необходимым сервисом можно проводить через командную оболочку, а именно запуск всех скриптов по сборке и запуску виртуальных машин. Контроллер будет представлять из себя программное обеспечение, написанное на языке программирования java, установленное на хост – компьютере. Контроллер будет взаимодействовать с виртуальными машинами следующим образом:

1. При инициализации контроллера на хост машине автоматически запускается первая виртуальная машина
2. Через определённые промежутки времени виртуальная машина будет передавать контроллеру информацию о загруженности процессора и используемой оперативной памяти
3. Контроллер через перенаправление вывода консоли виртуальной машины считывает данные

4. При получении новых данных контроллер обрабатывает их и совершает одно из трёх действий:
   4.1. Запустить ещё n образов виртуальных машин с новыми данными
   4.2. Остановить n образов виртуальных машин
   4.3. Отправить информацию об используемых ресурсах виртуальной машины
   4.4. Ничего не делать.

Для выполнения поставленной задачи нужно решить вопрос:

1. Каким образом взаимодействовать с виртуальными машинами?

Решением данного вопроса взаимодействия с виртуальными машинами было принято использование скриптов, написанных для выполнения в BASH [38], для компилирования образа виртуальной машины с необходимыми настройками dhcp (ip адреса, маски подсети и т. д.), запуска созданной виртуальной машины QEMU и перенаправления вывода запущенной виртуальной машины. Например, ниже представлен скрипт компилирования и запуска виртуальной машины под управлением IncludeOS с тестовым сервисом.

```
pushd examples/demo_service
mkdir -p build
pushd build
cmake ..
make
echo -e "Build complete \n"
echo -e "Starting VM with Qemu. "
echo -e "You should now get a boot message from the virtual machine:"
../run.sh build/IncludeOS_example.img
echo -e "\nTest complete.\n"
popd
trap — EXIT
```

Текст скрипта компилирования и запуска тестового сервиса test.sh

Bash — усовершенствованная и модернизированная вариация

командной оболочки Bourne shell. Одна из наиболее популярных современных разновидностей командной оболочки UNIX.

Bash — это командный процессор, работающий, как правило, в интерактивном режиме в текстовом окне. Bash также может читать команды из файла, который называется скриптом (или сценарием). Как и все Unix-оболочки, он поддерживает автодополнение имён файлов и директорий, подстановку вывода результата команд, переменные, контроль за порядком выполнения, операторы ветвления и цикла. Ключевые слова, синтаксис и другие основные особенности языка были заимствованы из sh.

5.4 Консоль администратора

Консоль администратора разрабатывается для мониторинга состояния виртуальных машин и отображения графиков их состояний. Данный компонент будет написан на языке программирования Java.

В связи с вышесказанным возникает ряд вопросов:

1. Какой формат запросов использовать для запроса состояний виртуальных машин?
2. Как отображать графики состояний виртуальных машин?

Решением первого вопроса обмена данными между консолью администратора и сервером, как уже было описано в клиентском приложении, был принят формат обмена данными json.

Решением второго вопроса о построении графиков состояний виртуальных машин стал JfreeChart [39] – библиотека с открытым исходным кодом на Java, распространяемая по лицензии LGPL и используемая для создания широкого спектра графиков. Используя JFreeChart, мы можем создать все основные типы 2D и 3D графики, таких как круговой диаграммы, гистограммы, линейный график, XY графика и 3D графики.

Причины выбора JfreeChart:

1. Бесплатный проект с открытым исходным кодом
2. Поставляется с хорошо документированных API

3. Поддерживает широкий спектр типов диаграмм, таких как круговая диаграмма, линия диаграммы, гистограммы, диаграммы Area и 3D графики
4. JFreeChart легко расширяемый.

5.5 Виртуальная машина

Сервис каждой виртуальной машины будет дополнен модулем, который выполняет следующие действия:

1. При старте считывает из параметров запуска ip адрес, на котором будет расположен сервис
2. В определённый промежуток времени выводит в консоль контроллера информацию о себе, а именно:
   2.1. Использование вычислительной мощности виртуального процессора, занимаемое виртуальной машиной
   2.2. Занимаемый объём оперативной памяти.

Для выполнения поставленных задач нужно решить следующие вопросы:

1. Как конфигурировать сетевую подсистему виртуальных машин для доступности сервиса?
2. Какое программное обеспечение использовать для запуска виртуальных машин?

Решением первого вопроса о конфигурировании сетевой подсистемы стало использование поставляемых вместе с IncludeOS библиотек, а именно niet4 [40]. Ниже приведён пример настройки сетевой подсистемы виртуальной машины.

```
#include <net/inet4>
void Service::start(const std::string&)
{
  auto& inet = net::Inet4::ifconfig(10.0);
  net::Inet4::ifconfig(
    { 10,0,0,42 },     // IP
    { 255,255,255,0 }, // Netmask
```

```
    { 10,0,0,1 },       // Gateway
    { 10,0,0,1 });      // DNS
}
```

Пример настройки сетевой подсистемы виртуальной машины.

Решением третьего вопроса о программном обеспечении для запуска виртуальных машин было принято использование QEMU.

QEMU [41] — свободно распространяемый программный продукт по лицензии GNU GPL 2 с открытым исходным кодом для эмуляции аппаратного обеспечения различных платформ.

Включает в себя эмуляцию процессоров Intel x86 и устройств ввода-вывода. Может эмулировать 80386, 80486, Pentium, Pentium Pro, AMD64 и другие x86-совместимые процессоры; PowerPC, ARM, MIPS, SPARC, SPARC64, m68k — лишь частично.

По умолчанию входит практически во все дистрибутивы Unix подобных операционных систем.

В настоящее время идёт разработка поддержки технологий аппаратной виртуализации (Intel VT и AMD SVM) на x86-совместимых процессорах Intel и AMD в QEMU. Первоначально разработка велась в рамках проекта Linux KVM (Kernel-based Virtual Machine), в котором помимо собственно KVM (поддержки технологий аппаратной виртуализации x86-совместимых процессоров на уровне ядра Linux) разрабатывались патчи для QEMU, позволяющие QEMU использовать функциональность KVM. Однако недавно разработчики QEMU в содружестве с разработчиками KVM приняли решение в ближайшем будущем интегрировать поддержку KVM в основную ветку QEMU (mainline).

На рисунке 3 изображён тестовый микросервис запущенный в операционной системе IncludeOS на QEMU.

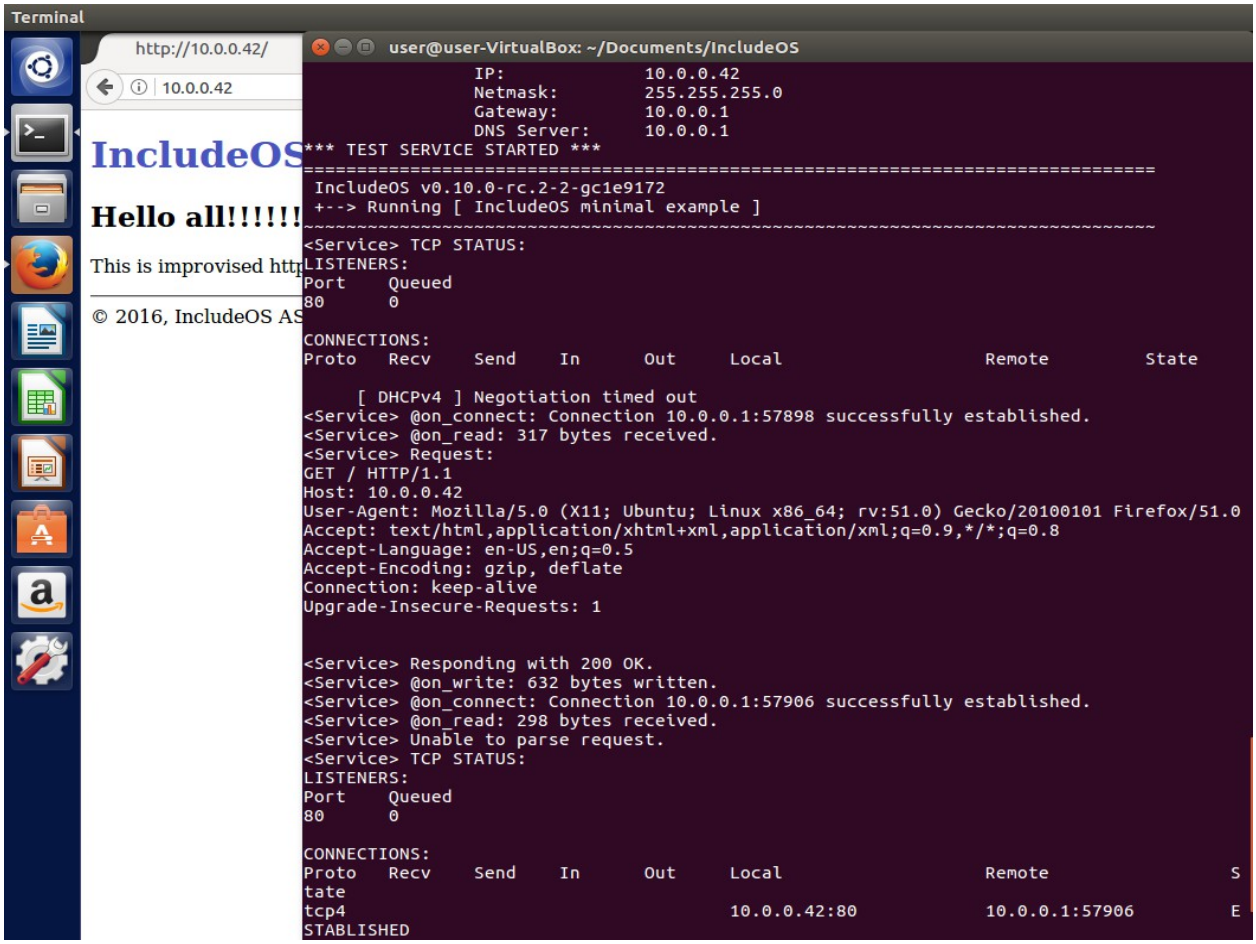

Рисунок 4.1 — Запущенный тестовый микросервис

6 Описание программного обеспечения

Следующие компоненты системы были написаны на языке программирования Java: консоль администратора, клиентский компонент системы, сервер, контроллер. На языке программирования C++ написано дополнение сервиса виртуальной машины.

Также были использованы библиотеки:

1) JfreeChart — для отображения графика загруженности виртуальной машины в консоли администратора.
2) HTTPclient и HTTPcore от Apache – для формирования http запросов от клиента, контроллера и консоли администратора на сервер и от сервера на виртуальные машины.
3) Gson – для сериализации объектов и передачи их через http запросы, а также десериализации объектов, полученных в качестве ответа на http запросы.
4) Log4j — для поддержки логирования сервера, запущенного на GlassFish.

6.1 Классы реализующие клиентский компонент системы

Класс: ClientForm

Описание: Класс, реализующий пользовательский интерфейс клиентского компонента.

Поля:

1) private String url - полный адрес сервера.
2) private final String TAG = "Client" — заголовок в системе логирования.
3) private final String CONFIG_NAME = "connect.cfg" — название конфигурационного файла, хранящего данные для

взаимодействия с сервером.

4) private String ip, port — ip и порт сервера.
5) private ClientHttp connect — http подключение к серверу.
6) private List<Job<Data, Data>> jobs — список отправленных запросов на сервер

Методы:

1) public ClientForm() - конструктор.
2) private void initComponents() - инициализации компонентов пользовательского интерфейса.
3) private void addJobButtonActionPerformed() - обработка клика пользователя по кнопке добавления задачи на обработку.
4) private void configConnectActionPerformed() - обработка клика пользователя по меню настройки подключения.
5) private void TestConnectActionPerformed() - обработка клика пользователя по кнопке проверки подключения.
6) private void CfgOkActionPerformed() - обработка клика пользователя по кнопке сохранения параметров подключения.
7) private void sendJobButtonActionPerformed() - обработка клика пользователя по кнопке отправки запроса на обработку.
8) private void saveJobButtonActionPerformed() - обработка клика пользователя по кнопке сохранения результатов обработки.
9) private void loadDataButtonActionPerformed() - обработка клика пользователя по кнопке загрузки данных для обработки.
10) private void enableComponents(Container container, boolean enable) — выставление флагов доступности компонентов пользовательского интерфейса.

Параметры:

1. Container container — контейнер элементов.
2. boolean enable — флаг доступности.

11) private File selectFile(String okButtomTest) — метод, возвращающий файл для записи/чтения.

Параметры:

1. String okButtomText — надпись на кнопке подтверждения действия.

12) private void loadAddJobDialog(Job<Data, Data> job) — метод, запускающий диалог просмотра ранее обработанной задачи, а также отправка нового запроса на обработку.

Параметры:

1. Job<Data, Data> job — данные ранее обработанной задачи для отображения, если job равен null, то создание новой задачи на обработку.

13) private boolean validateAddJob() — метод, проверки корректности заполненных данных для обработки.

14) private void showError(String messege) — метод, отображения диалога ошибки.

Параметры:

1. String messege — сообщение ошибки.

15) private void fillTableJob() — метод, заполнения таблицы обработанных запросов.

Класс: LocalMouseAdapter

Описание: Класс реализации абстрактного адаптера MouseAdapter для обработки клика мыши.

Поля: отсутствуют.

Методы:

1) public void mouseClicked(MouseEvent e) — метод, обрабатывающий клик правой кнопкой мыши по форме.

   Параметры:

   1. MouseEvent e — положение мыши в момент клика.

Класс: LocalAddJobListener

Описание: Класс реализации собственного интерфейса ConnectListener, который будет описан ниже, для обработки http запроса отправки данных на сервер. Реализация паттерна «Наблюдатель».

Поля: отсутствуют.

Методы:

1) public void onStart() — метод, сигнализирующий о старте запроса.
2) public void onResult(Response response) — метод, сигнализирующий о окончании запроса.

   Параметры:

   1. Response response — ответ сервера.
3) public void onError(Exception ex) — метод, сигнализирующий об ошибке во время запроса.

   Параметры:

   1. Exception ex — java класс исключения.

Класс: LocalPingListener

Описание: Класс реализации собственного интерфейса ConnectListener, который будет описан ниже, для обработки http запроса

проверки доступности сервера. Реализация паттерна «Наблюдатель».

Поля: отсутствуют.

Методы:

1) public void onStart() — метод, сигнализирующий о старте запроса.
2) public void onResult(Response response) — метод, сигнализирующий о окончании запроса.

   Параметры:

   1. Response response — ответ сервера.
3) public void onError(Exception ex) — метод, сигнализирующий об ошибке во время запроса.

   Параметры:

   1. Exception ex — java класс исключения.

Класс: JobTask

Описание: Класс, реализующий интерфейс Runnable. Задача на отправку данных для обработки.

Поля:

1) private ClientHttp.ConnectListener listener — наблюдатель выполнения запроса.
2) private ClientHttp client — http подключение к серверу.
3) private Job data — данные для обработки.

Методы:

1) public JobTask(ClientHttp.ConnectListener listener, ClientHttp client, Job data) — конструктор с параметрами, инициализирующими класс. Параметры совпадают с полями класса.

2) public void run() — метод, выполняющий запрос к серверу.

Класс: PingTask

Описание: Класс, реализующий интерфейс Runnable. Задача на проверку доступности сервера.

Поля:

1) private ClientHttp.ConnectListener listener — наблюдатель выполнения запроса.
2) private ClientHttp client — http подключение к серверу.

Методы:

1) public PingTask(ClientHttp.ConnectListener listener, ClientHttp client) — конструктор с параметрами, инициализирующими класс. Параметры совпадают с полями класса.
2) public void run() — метод, выполняющий запрос к серверу.

Класс: ConfigEditor

Описание: Класс — утилита для загрузки и сохранения конфигурационного файла.

Поля: отсутствуют.

Методы:

1) public static Map<String, String> getConfig(String fileName) — загрузка конфигурационных параметров из файла.

   Параметры:

   1. String fileName - название файла.

2) public static boolean setConfig(Map<String, String> args, String fileName) — сохранение конфигурационных параметров в файл.

Параметры:

1. Map<String, String> args - параметры для сохранения.
2. String fileName - название файла.

Класс: FileHelper

Описание: Класс — утилита для загрузки данных для обработки из файла и сохранения обработанных данных в файл.

Поля: отсутствуют.

Методы:

1) public static void saveResultJob(Data data, File file) — метод сохранения обработанных данных в файл.

   Параметры:

   1. Data data — обработанные данные.
   2. File file — файла для сохранения.

2) public static Data inputLoad(File file) — метод загрузки данных для обработки.

   Параметры:

   1. File file — файла для загрузки.

6.2 Классы реализующие консоль администратора

Класс: AdminForm

Описание: Класс, реализующий пользовательский интерфейс консоли администратора.

Поля:

1) private String url - полный адрес сервера.
2) private final String TAG = "Admin" — заголовок в системе

логирования.

3) private final String CONFIG_NAME = "connect.cfg" — название конфигурационного файла, хранящего данные для взаимодействия с сервером.
4) private String ip, port — ip и порт сервера.
5) private ClientHttp connect — http подключение к серверу.
6) private List<VirtualMachine> vm — список запущенных виртуальных машин.

Методы:

1) public AdminForm() - конструктор.
2) private void initComponents() - инициализации компонентов пользовательского интерфейса.
3) private void configConnectActionPerformed() - обработка клика пользователя по меню настройки подключения.
4) private void TestConnectActionPerformed() - обработка клика пользователя по кнопке проверки подключения.
5) private void CfgOkActionPerformed() - обработка клика пользователя по кнопке сохранения параметров подключения.
6) private void enableComponents(Container container, boolean enable) — выставление флагов доступности компонентов пользовательского интерфейса.

   Параметры:

   1. Container container — контейнер элементов.
   2. boolean enable — флаг доступности.

7) private void showError(String messege) — метод отображения диалога ошибки.

Параметры:

1. String messege — сообщение ошибки.

8) private void fillVMTable() — метод заполнения таблицы информации о виртуальных машинах.

9) private void openVMStateDialog(VirtualMachine vm) — отображения окна с подробной информацией о виртуальной машине.

Параметры:

1. VirtualMachine vm — информация о виртуальной машине.

Класс: LocalMouseAdapter

Описание: Класс, реализации абстрактного адаптера MouseAdapter, для обработки клика мыши.

Поля: отсутствуют.

Методы:

1) public void mouseClicked(MouseEvent e) — метод, обрабатывающий клик правой кнопкой мыши по форме.

Параметры:

1. MouseEvent e — положение мыши в момент клика.

Класс: LocalPingListener

Описание: Класс реализации собственного интерфейса ConnectListener, который будет описан ниже, для обработки http запроса проверки доступности сервера. Реализация паттерна «Наблюдатель».

Поля: отсутствуют.

Методы:

1) public void onStart() — метод, сигнализирующий о старте запроса.
2) public void onResult(Response response) — метод, сигнализирующий о окончании запроса.

   Параметры:

   1. Response response — ответ сервера.
3) public void onError(Exception ex) — метод, сигнализирующий об ошибке во время запроса.

   Параметры:

   1. Exception ex — java класс исключения.

Класс: LocalVMStateLoadListener

Описание: Класс реализации собственного интерфейса ConnectListener, который будет описан ниже, для обработки http запроса информации о состоянии всех запущенных виртуальных машин. Реализация паттерна «Наблюдатель».

Поля: отсутствуют.

Методы:

1) public void onStart() — метод, сигнализирующий о старте запроса.
2) public void onResult(Response response) — метод, сигнализирующий о окончании запроса.

   Параметры:

   1. Response response — ответ сервера.
3) public void onError(Exception ex) — метод, сигнализирующий об ошибке во время запроса.

   Параметры:

1. Exception ex — java класс исключения.

Класс: VMStateChartPanel

Описание: Класс, реализующий интерфейс StandardDialog из библиотеке JFreeChart, для отображения подробной информации об виртуальной машине в виде графика.

Поля: отсутствуют.

Методы:

1) public VMStateChartPanel(String title, VirtualMachine vMachine) — конструктор.

   Параметры:

   1. String title — заголовок окна.
   2. VirtualMachine vMachine — информация о виртуальной машине.

2) private static CategoryDataset createDataset(VirtualMachine vMachine) — метод подготовки данных для отображения.

   Параметры:

   1. VirtualMachine vMachine — информация о виртуальной машине.

Класс: VMStateTask

Описание: Класс, реализующий интерфейс Runnable. Задача на запрос состояний виртуальных машин.

Поля:

1) private ClientHttp.ConnectListener listener — наблюдатель выполнения запроса.
2) private ClientHttp client — http подключение к серверу.

Методы:

1) public VMStateTask(ClientHttp.ConnectListener listener, ClientHttp client) — конструктор с параметрами, инициализирующими класс. Параметры совпадают с полями класса.
2) public void run() — метод, выполняющий запрос к серверу.

Класс: PingTask

Описание: Класс, реализующий интерфейс Runnable. Задача на проверку доступности сервера.

Поля:

1) private ClientHttp.ConnectListener listener — наблюдатель выполнения запроса.
2) private ClientHttp client — http подключение к серверу.

Методы:

1) public PingTask(ClientHttp.ConnectListener listener, ClientHttp client) — конструктор с параметрами, инициализирующими класс. Параметры совпадают с полями класса.
2) public void run() — метод, выполняющий запрос к серверу.

Класс: ConfigEditor

Описание: Класс — утилита для загрузки и сохранения конфигурационного файла.

Поля: отсутствуют.

Методы:

1) public static Map<String, String> getConfig(String fileName) — загрузка конфигурационных параметров из файла.

Параметры:

1. String fileName - название файла.

2) public static boolean setConfig(Map<String, String> args, String fileName) — сохранение конфигурационных параметров в файл.

Параметры:

1. Map<String, String> args - параметры для сохранения.
2. String fileName - название файла.

6.3 Классы реализующие контроллер

Класс: Controller

Описание: Класс инициализирующий работу контроллера.

Поля: отсутствуют.

Методы:

1) public static void main() — метод, запускающий главный поток контроллера.

Параметры: отсутствуют.

Класс: MainThread

Описание: Класс, реализующий главный поток контроллера. Реализует java класс Runnable.

Поля:

1) private String sudo — пароль суперпользователя Unix системы.
2) private String serverIp — ip сервера.
3) private String serverPort — порт сервера.
4) private BlockingQueue<String> vmQueue — очередь ip адресов для

запуска виртуальных машин.

5) private String firstIp — ip первой виртуальной машины, стартующей при запуске контроллера.

Методы:

1) public MainThread(String sudo, String serverIp, String serverPort, String firstIp) - конструктор с параметрами, инициализирующими класс. Параметры совпадают с полями класса.
2) public void run() — метод, проверяющий наличие ip адресов в очереди и производящий запуск виртуальных машин при обнаружении нового ip в очереди.

Интерфейс: VMThreadListener

Описание: Интерфейс наблюдателя работы виртуальной машины .

Поля: отсутствуют.

Методы:

1) void onStart() - событие на запуск наблюдателя.
2) void setData(String data) - главный обработчик, получает сообщения от виртуальной машины и обрабатывает их.
3) void onDestroy() - событие на остановку наблюдателя.

Класс: LocalVMThreadListener

Описание: Класс, реализующий интерфейс наблюдателя работы виртуальной машины - VMThreadListener.

Поля:

1) private int pid — идентификатор процесса, который обрабатывает наблюдатель.

2) private String ip — ip виртуальной машины, запущенной в данном процессе.
3) private ClientHttp client — http подключение к серверу.

Методы:

1) public LocalVMThreadListener(String ip, String sIp, String sPort) - конструктор с параметрами, инициализирующими класс.

   Параметры:

   1. String ip - ip виртуальной машины.
   2. String sIp — ip сервера для отправки информации о загруженности виртуальной машины.
   3. String sPort — порт сервера.

2) void onStart() - событие на запуск наблюдателя.
3) void setData(String data) - главный обработчик, получает сообщения от виртуальной машины и обрабатывает их.
4) void onDestroy() - событие на остановку наблюдателя.
5) private int killProces(int pid) — метод, останавливающий процесс с запущенной виртуальной машиной.

Класс: LocalStateListener

Описание: Класс реализации собственного интерфейса ConnectListener, который будет описан ниже, для отправки http запроса с информацией о загруженности запущенной виртуальной машины. Реализация паттерна «Наблюдатель».

Поля: отсутствуют.

Методы:

1) public void onStart() — метод, сигнализирующий о старте запроса.

2) public void onResult(Response response) — метод, сигнализирующий об окончании запроса.

   Параметры:

   1. Response response — ответ сервера.

3) public void onError(Exception ex) — метод, сигнализирующий об ошибке во время запроса.

   Параметры:

   1. Exception ex — java класс исключения.

Класс: VMThread

Описание: Класс, реализующий поток работы процесса виртуальной машины. Реализует java класс Runnable.

Поля:

1) private final VMThreadListener listener — слушатель работы виртуальной машины.
2) private final String[] cmd — unix команда на запуск виртуальной машины с ip адресом в качестве параметра.

Методы:

1) public VMThread(VMThreadListener listener, String[] cmd) - конструктор с параметрами, инициализирующими класс. Параметры совпадают с полями класса.
2) public void run() — метод, реализующий запуск виртуальной машины и чтение выходного потока.

Класс: SendVMStateTask

Описание: Класс, реализующий интерфейс Runnable. Задача на

отправку состояния виртуальной машины.

Поля:

1) private ClientHttp.ConnectListener listener — наблюдатель выполнения запроса.
2) private ClientHttp client — http подключение к серверу.
3) private VirtualMachine vm — информация о виртуальной машине.

Методы:

1) public SendVMStateTask(ClientHttp.ConnectListener listener, ClientHttp client, VirtualMachine vm) — конструктор с параметрами, инициализирующими класс. Параметры совпадают с полями класса.
2) public void run() — метод, выполняющий запрос к серверу.

6.4 Классы реализующие сервер

Класс: ServerServlet

Описание: Главный класс сервера, расширяющий класс HttpServlet. ServerServlet обрабатывает все get и post запросы.

Поля:

1) private final Logger log = Logger.getLogger("super_server") — инициализация системы логирования сервера.
2) private DataSource ds — коннект к базе данных.

Методы:

1) public void init(ServletConfig config) — инициализация сервера.

   Параметры:

   1. ServletConfig config — конфигурация сервера.

2) protected void doPost(HttpServletRequest request, HttpServletResponse response) — получение post запроса.

Параметры:

1. HttpServletRequest request — данные запроса на сервер.
2. HttpServletResponse response — данные ответа сервера.

3) protected void doGet(HttpServletRequest request, HttpServletResponse response) — получение get запроса.

Параметры:

1. HttpServletRequest request — данные запроса на сервер.
2. HttpServletResponse response — данные ответа сервера.

4) protected void processRequest(HttpServletRequest request, HttpServletResponse response) – обработка get и post запросов.

Параметры:

1. HttpServletRequest request — данные запроса на сервер.
2. HttpServletResponse response — данные ответа сервера.

5) private void processAddJob(Map<String, String[]> params, ServerHandler sh, DBManager db) — обработка новой задачи.

Параметры:

1. Map<String, String[]> params — параметры запроса.
2. ServerHandler sh — серверный обработчик данных.
3. DBManager db — менеджер по работе с базой данных.

6) private VirtualMachine getMinVM(List<VirtualMachine> snapshot) — получение из списка работающих виртуальных машин наименее загруженную.

Параметры:

1. List<VirtualMachine> snapshot - список работающих виртуальных машин.

7) private void processVMStateUpdate(Map<String, String[]> params, ServerHandler sh, DBManager db) — обновление информации виртуальной машины.

   Параметры:

   1. Map<String, String[]> params — параметры запроса.
   2. ServerHandler sh — серверный обработчик данных.
   3. DBManager db — менеджер по работе с базой данных.

8) private void processVMState(ServerHandler sh, DBManager db) — обработка запроса списка работающих виртуальных машин.

   Параметры:

   1. ServerHandler sh — серверный обработчик данных.
   2. DBManager db — менеджер по работе с базой данных.

9) private String getParam(Map<String, String[]> params, String name) — получение из параметров запроса определённый параметр.

   Параметры:

   1. Map<String, String[]> params — параметры запроса.
   2. String name — название определённого параметра.

10) private String getSysParam(Map<String, String> params, String name) - получение из параметров системы определённый параметр.

    Параметры:

    1. Map<String, String[]> params — параметры системы.
    2. String name — название определённого параметра.

Класс: NoArgumentException

Описание: Класс, расширяющий стандартный класс java исключений

Exception. Исключение, генерируемое при отсутствии искомого параметра запроса или системного параметра.

Поля: отсутствуют.

Методы:

1) public NoArgumentException() — конструктор по умолчанию.
2) public NoArgumentException(String message) — конструктор с текстом исключения.

Класс: NoFreeVMException

Описание: Класс, расширяющий стандартный класс java исключений Exception. Исключение, генерируемое при отсутствии информации о запущенных виртуальных машинах.

Поля: отсутствуют.

Методы:

1) public NoFreeVMException() — конструктор по умолчанию.
2) public NoFreeVMException(String message) — конструктор с текстом исключения.

Класс: HttpRequestHelper

Описание: Класс утилита, реализующий запросы от сервера к виртуальным машинам.

Поля:

1) private final Logger log = Logger.getLogger("super_server") — инициализация системы логирования сервера.

Методы:

1) public static String sendingStartKillRequest(String url, String command) —

формирования http запроса на запуск/остановку виртуальной машины.

Параметры:

1. String url — адрес виртуальной машины.
2. String command — команда (запуск/остановка виртуальной машины).

2) public static String sendingRequest(String url, String data) - формирования http запроса на обработку данных виртуальной машины.

Параметры:

1. String url — адрес виртуальной машины.
2. String data — данные для обработки.

3) private static String send(HttpGet request) — метод выполнения http запроса.

Параметры:

1. HttpGet request — параметры запроса, сформированные в функциях sendingRequest и sendingStartKillRequest.

4) private static DefaultHttpClient getHttpClient() - метод создания параметров http запроса.

Класс: DBManager

Описание: Класс, реализующий менеджера по работе с базой данных.

Поля:

1) private final Logger log = Logger.getLogger("super_server") — инициализация системы логирования сервера.
2) private Connection conn — подключение к базе данных.

Методы:

1) public DBManager(Connection conn) — конструктор с параметрами, инициализирующими класс. Параметры совпадают с полями класса.

2) public List<VirtualMachine> getVMSnapshot() - получить состояния всех виртуальных машин.

3) public void clearDB(long maxVM, long maxIP) — очистить таблицы виртуальных машин и списка ip адресов базы данных от записей, которые не обновлялись определённое время.

   Параметры:

   1. long maxVM — максимальное время отсутствия обновлений для записей таблицы виртуальных машин.
   2. long maxIP — максимальное время отсутствия обновлений для записей таблицы ip адресов.

4) public VirtualMachine getFistFreeVM() - получить первую простаивающую виртуальную машину.

5) public int registerNewIP(String ip, int port) - регистрация нового ip в базе данных и возвращение его id в базе данных.

   Параметры:

   1. String ip — новый ip адрес.
   2. int port — порт, соответствующий новому ip адресу.

6) private void registerNewVM(VirtualMachine vm, int id_ip) — регистрация новой виртуальной машины в базе данных.

   Параметры:

   1. VirtualMachine vm — информация о виртуальной машине.
   2. int id_ip — ip адрес новой виртуальной машины.

7) private int checkIP(String ip, int port) — проверка наличия ip адреса в базе.

Параметры:

1. String ip — ip адрес.
2. int port — порт, соответствующий ip адресу.

8) public String getFreeIP() - получить первый свободный ip адрес.

9) private void updateIPState(int id_ip, boolean newState) — обновить состояние блокировки для ip.

Параметры:

1. int id_ip — ключ ip адреса в базе данных.
2. boolean newState — новое состояние блокировки.

10) private void updateVM(VirtualMachine vm, int id_vm, int freeCount, int id_ip) — обновить информацию о виртуальной машине.

Параметры:

1. VirtualMachine vm — информация о виртуальной машины.
2. int id_vm — ключ виртуальной машины в базе.
3. int freeCount — количество холостых состояний виртуальной машины подряд.
4. int id_ip — ключ ip адреса в базе данных.

11) public void resetVMfreeCount(int id_vm) — метод сброса холостых состояний виртуальной машины подряд.

Параметры:

1. int id_vm — ключ виртуальной машины в базе.

12) private int getFreeCount(int id_vm) — получить количество холостых состояний виртуальной машины подряд.

Параметры:

1. int id_vm — ключ виртуальной машины в базе.

13) public int updateVmState(VirtualMachine vm) — обновление состояния виртуальной машины.

Параметры:

1. VirtualMachine vm — информация о виртуальной машине.

14) public void deleteVM(VirtualMachine vm) — удалить виртуальную машину из базы данных.

Параметры:

1. VirtualMachine vm — информация о виртуальной машине.

15) public Map<String, String> getSystemParams() - получить парамерты системы.

6.5 Классы реализующие модель данных

Модель данных — компонент системы, реализующий общие классы данных для других компонентов системы. Например, общими классами являются VirtualMachine — информация о виртуальной машины, Job – задача на обработку и т. д. Модель данных является неким утилитным компонентом системы, например, в ней реализована работа по сериализации и десериализации данных для всех компонентов системы и т. д.

Класс: ClientHttp

Описание: Класс для формирования и отправления http запросов.

Поля:

1) private final Gson gson — объект, предоставляющий функционал для сериализации и десериализации данных в Gson формат.
2) private String url — адрес отравления http запросов.

Методы:

1) public ClientHttp(String url) — конструктор с параметрами,

инициализирующими класс. Параметры совпадают с полями класса.

2) public Response ping() - формирование запроса на доступность удалённого узла.
3) public Response<Job<Data, Data>> addJob(Job job) — формирование запроса на отправку данных для обработки.

   Параметры:

   1. Job job — данные для обработки.

4) public Response<List<VirtualMachine>> getVMState() - формирование запроса на получение информации обо всех запущенных виртуальных машинах.
5) public Response sendVMState(VirtualMachine vm) — формирование запроса на отправку информации о виртуальной машине.

   Параметры:

   1. VirtualMachine vm — информация о виртуальной машины.

6) private String execute(List<NameValuePair> params) — выполнение http запроса.

   Параметры:

   1. List<NameValuePair> params — список параметров http запроса.

Класс: Logger

Описание: Утилитный класс для вывода системной информации в консоль.

Поля: отсутствуют.

Методы:

1) public static void i(String TAG, String str) — информационное сообщение.

Параметры:

1. String TAG - заголовок в системе логирования.
2. String str - сообщение.

2) public static void e(String TAG, String str) — сообщение об ошибки.

Параметры:

1. String TAG - заголовок в системе логирования.
2. String str - сообщение.

3) public static void e(String TAG, String str, Exception ex) — сообщение об ошибке.

Параметры:

1. String TAG - заголовок в системе логирования.
2. String str - сообщение.
3. Exception ex – исключение.

Класс: Data

Описание: Класс, реализующий класс Serializable, представляющий данные для обработки.

Поля:

1) private String data — данные для обработки.

Методы:

1) public Data(String data) — конструктор с параметрами, инициализирующими класс. Параметры совпадают с полями класса.
2) public String getData() - возвращает данные для обработки.

Класс: Job<T,M>

Описание: Класс, реализующий класс Serializable, представляющий задачу для обработки.

Поля:

1) private T inputData — входные данные для обработки.
2) private M outputData — выходные данные для обработки.
3) private String name — название работы.
4) private int state — состояние задачи.

Методы:

1) public Job(T inputData, T outputData, String name, int state) — конструктор с параметрами, инициализирующими класс. Параметры совпадают с полями класса.

Класс: Response<T>

Описание: Класс, реализующий класс Serializable, представляющий ответ на запрос серверу.

Поля:

1) private T data — данные.
2) private final int resultCode — код ошибки.
3) public static final int OK = 0 — успех оработки.
4) public static final int IN_PROCESS = 1 — данные в обработке.
5) public static final int WRONG_METHOD = 2 — ошибка, неизвестный метод.
6) public static final int INTERNAL_ERROR = 3 — внутренняя ошибка сервера.
7) public static final int DB_ERROR = 4 — ошибка базы данных.

8) public static final int WRONG_DATA = 5 — ошибка, неверные данные в запросе.

9) public static final int PROCESS_ERROR = 6 — ошибка обработки данных.

10) public static final int VM_NOT_FOUND = 7 — ошибка, нет доступных виртуальных машин.

Методы:

1) public Response(int resultCode, T data) — конструктор с параметрами, инициализирующими класс. Параметры совпадают с полями класса.
2) public T getData() - получить данные.
3) public int getResultCode() - получить код обработки.

Класс: VirtualMachine

Описание: Класс, реализующий класс Serializable, представляющий информацию о виртуальной машине.

Поля:

1) private int id — идентификатор машины в базе.
2) private long ram — используемый объём оперативной памяти.
3) private int cpu — загрузка процессора (в процентах).
4) private String ip — ip адрес виртуальной машины.
5) private int port — порт виртуальной машины.

Методы:

1) public VirtualMachine(int id, long ram, int cpu, String ip, int port) — конструктор с параметрами, инициализирующими класс. Параметры совпадают с полями класса.

2) public VirtualMachine(int id, long ram, int cpu, String ip) — конструктор с параметрами, инициализирующими класс. Параметры совпадают с полями класса.
3) public VirtualMachine(long ram, int cpu, String ip, int port) — конструктор с параметрами, инициализирующими класс. Параметры совпадают с полями класса.
4) public VirtualMachine(long ram, int cpu, String ip) — конструктор с параметрами, инициализирующими класс. Параметры совпадают с полями класса.

Класс: ClientHandler

Описание: Класс для сериализации и десериализации объектов клиентского компонента.

Поля:

1) private final Gson gson — объект предоставляющий функционал для сериализации и десериализации данных в Gson формат.

Методы:

1) public ClientHandler() — конструктор.
2) public String data(Data data) — сериализации данных для обработки в gson строку.

   Параметры:

   1. Data data - данные для обработки.

3) public Response<Data> getData(String str) - десериализации ответа сервера на запрос обработки данных из gson строки.

   Параметры:

   1. String str - gson строка.

4) public Response<List<VirtualMachine>> getVMState(String str) - десериализации ответа сервера на запрос списка запущенных виртуальных машин из gson строки.

Класс: ServerHandler

Описание: Класс для сериализации и десериализации объектов сервера.

Поля:

1) private final Gson gson — объект предоставляющий функционал для сериализации и десериализации данных в Gson формат.
2) private Appendable appendable — ответ сервера.

Методы:

1) public ServerHandler(Appendable appendable) — конструктор с параметрами, инициализирующими класс. Параметры совпадают с полями класса.
2) public void process(int errorCode) - отправить код обработки без данных.

   Параметры:

   1. int errorCode — код ответа.

3) public void process(Data data) - отправить обработанные данные.

   Параметры:

   1. Data data - обработанные данные.

4) public void process(List<VirtualMachine> data) - отправить список запущенных виртуальных машин.

   Параметры:

   1. List<VirtualMachine> data - список запущенных виртуальных машин.

5) public Data getProcessedData(String data) — десериализация данных из gson строки.

   Параметры:

   1. String data - gson строка.

6) public VirtualMachine getVirtualMachine(String data) — десериализация информации о виртуальной машине из gson строки.

   Параметры:

   1. String data - gson строка.

7) private void dataSend(Response res) — отправка ответа.

   Параметры:

   1. Response res — ответ сервера.

Класс: JsonUtils
Описание: Класс для создания Gson объекта.

Поля: отсутствуют.

Методы:

1) public static Gson createGson() - создание Gson объекта.

Класс: HttpMetods
Описание: Список запросов, поддерживаемых сервером.

Поля:

1) public static final String SEND_JOB = "sendJob" — запрос на обработку данных.
2) public static final String PING = "ping" — проверка доступности удалённого узла.

3) public static final String VM_STATE = "vmstate" — запрос на получение списка запущенных виртуальных машин.
4) public static final String SEND_STATE = "sendvmstate" — запрос на обновление информации о виртуальной машине.

Методы: отсутствуют.

Класс: VMCommand

Описание: Список команд, поддерживаемых контроллером виртуальных машин.

Поля:

1) public static final String COMMAND = "Command" — заголовок команды.
2) public static final String KILL = "kill_me" — команда остановки виртуальной машины.
3) public static final String START = "start_new" — команда на запуск виртуальных машин.

Методы: отсутствуют.

Класс: SystemParams

Описание: Список системных параметров сервера.

Поля:

1) public static final String FREE_COUNT = "free_count" — количество обновлений простаивающих состояний виртуальной машины подряд.
2) public static final String LAST_CONNECT = "last_connect_ms" - время с последнего обновления состояния виртуальной машины, после которого её можно считать мёртвой (в мс).

3) public static final String IP_RESERVE = "ip_reserve_ms" - время резерва ip адреса без привязки к виртуальной машине, после которого ip можно использовать снова (в мс).

4) public static final String MIN_VM_COUNT = "min_vm_count" — минимальное количество запущенных виртуальных машин.

Методы: отсутствуют.

Заключение

После анализа литературы по заданной теме и изучения документации по проекту IncludeOS была разработана горизонтально масштабируемая системы легковесных виртуальных машин, а также написан тестовый web сервис на языке программирования C++ под управлением одномодульной операционной системы IncludeOS. Сервер в горизонтально масштабируемой системе принимает get, set и put запросы от клиентов мобильного платёжного приложения под управлением Android и Ios, передаёт полученные данные в одномодульные операционные системы IncludeOS, которые, в свою очередь, производят обработку полученных данных и возвращают результат на клиенты.

Разработано программное обеспечение, в который входит:

1) сервер для распределения нагрузки между виртуальными машинами;
2) контроллер для управления виртуальнми машинам на хост-компьютере;
3) консоль администратора для просмотра информации о количестве и загруженности виртуальных машин;
4) дополнения сервиса виртуальной машины.

Список использованных источников

1. Главная страница сайта ООО «Soft-logic» [Электронный ресурс]. - Режим доступа: https://soft-logic.ru/
2. Главная страница сайта IncludeOS [Электронный ресурс]. - Режим доступа: http://www.includeos.org/
3. Микросервисы (Microservices) [Электронный ресурс]. - Режим доступа: https://habrahabr.ru/post/249183/
4. Преимущества и недостатки микросервисной архитектуры [Электронный ресурс]. - Режим доступа: http://eax.me/micro-service-architecture/
5. Микросервисы на практике - SmartMe University [Электронный ресурс]. - Режим доступа: http://smartme.university/course/microservices-in-practice/
6. Национальная библиотека им. Н.Э.Баумана Bauman National Library [Электронный ресурс]. - Режим доступа: http://ru.bmstu.wiki/Гипервизор
7. Графический интерфейс пользователя [Электронный ресурс]. - Режим доступа: http://www.microchip.com.ru/Support/GUI.html
8. CppCon 2016 [Электронный ресурс]. - Режим доступа: https://cppcon2016.sched.com/
9. CppCon 2016: #Include <os>: from bootloader to REST API with the new C++ [Электронный ресурс]. - Режим доступа: https://cppcon2016.sched.com/event/7nLe/include-ltosgt-from-bootloader-to-rest-api-with-the-new-c
10. В рамках проекта IncludeOS развивается ядро для обособленного запуска C++ приложений [Электронный ресурс]. - Режим доступа: https://www.opennet.ru/opennews/art.shtml?num=43444
11. Встраиваемые операционные системы [Электронный ресурс]. - Режим доступа: http://www.itechno.ru/index.php& option=com_k2 &view=itemlist

&task=category &id=153:Встраиваемые операционные системы
12. IncludeOS [Hatred's Log Place] — Programming [Электронный ресурс]. - Режим доступа: https://htrd.su/wiki/zhurnal/2016/09/22/includeos
13. #Include os [Электронный ресурс]. - Режим доступа: https://www.slideshare.net/IncludeOS/include-ltos-from-bootloader-to-rest-api-with-the-new-c
14. VMware ESX [Электронный ресурс]. - Режим доступа: http://www.vmware.com/ru/products/esxi-and-esx.html
15. XenServer | Open Source Server Virtualization [Электронный ресурс]. - Режим доступа: https://xenserver.org/
16. Citrix [Электронный ресурс]. - Режим доступа: https://www.citrix.ru/
17. Виртуализация Vmware [Электронный ресурс]. - Режим доступа: http://www.vmware.com/ru.html
18. Гипервизоры, виртуализация и облако: О гипервизорах, виртуализации систем и о том, как это работает в облачной среде [Электронный ресурс]. - Режим доступа: http://www.ibm.com/developerworks/ru/library/cl-hypervisorcompare/
19. Базовая информация о VMWare vSphere [Электронный ресурс]. - Режим доступа: https://habrahabr.ru/post/226231/
20. Анализ гипервизора Vmware, его преимущества и недостатки [Электронный ресурс]. - Режим доступа: https://xakep.ru/2011/11/07/Vmware
21. Xen [Электронный ресурс]. - Режим доступа: http://xgu.ru/wiki/Xen
22. Национальная библиотека им. Н. Э. Баумана Bauman National Library / Паравиртуализация [Электронный ресурс]. - Режим доступа: http://ru.bmstu.wiki/ Паравиртуализация
23. XenServer - Server Virtualization and Consolidation – Citrix [Электронный ресурс]. - Режим доступа: https://www.citrix.com/products/xenserver/
24. Архитектура Hyper-V [Электронный ресурс]. - Режим доступа:

https://habrahabr.ru/post/98580/

25. Review Hyper-V [Электронный ресурс]. - Режим доступа: https://msdn.microsoft.com/library/hh831531(v=ws.11).aspx
26. IncludeOS: A minimal, resource efficient unikernel for cloud services [Электронный ресурс]. - Режим доступа: https://folk.uio.no/paalee/publications/2015-cloudcom.pdf
27. B. Stroustrup, The C++ Programming Language (4th. edition). Addison-Wesley, 2013. – Pp 66-67.
28. Статические библиотеки [Электронный ресурс]. - Режим доступа: https://www.opennet.ru/docs/RUS/zlp/003.html
29. GCC toolchain [Электронный ресурс]. - Режим доступа: https://ru.wikipedia.org/wiki/GNU_toolchain
30. P. Pedriana, “EASTL - Electronic Arts Standard Template library,” Open Standards, Tech. Rep., Apr. 2007. [Электронный ресурс]. - Режим доступа: http://www.open-std.org/jtc1/ sc22/wg21/docs/papers/2007/n2271.html
31. KVM virtio drivers [Электронный ресурс]. - Режим доступа: http://itman.in/kvm-virtio-drivers/
32. R. Russell, M. S. Tsirkin, C. Huck, and P. Moll, “Virtual I/O Device (VIRTIO) Version 1.0,” OASIS Standard, OASIS Committee Specification 02, January 2015. [Электронный ресурс]. - Режим доступа: http://docs.oasis-open.org/ virtio/virtio/v1.0/virtio-v1.0.html
33. Callback (программирование) [Электронный ресурс]. - Режим доступа: https://ru.wikipedia.org/wiki/Callback
34. Json [Электронный ресурс]. - Режим доступа: http://www.json.org/json-ru.html
35. GlassFish Server Documentation [Электронный ресурс]. - Режим доступа: https://glassfish.java.net/documentation.html
36. GlassFish Server [Электронный ресурс]. - Режим доступа: http://www.oracle.com/us/products/middleware/cloud-app-

foundation/glassfish-server/overview/index.html

37. PostgreSQL: The world's most advanced open source database [Электронный ресурс]. - Режим доступа: https://www.postgresql.org/
38. Bash (Unix shell) [Электронный ресурс]. - Режим доступа: https://en.wikipedia.org/wiki/Bash_(Unix_shell)
39. JFreeChart – Jfree.org [Электронный ресурс]. - Режим доступа: http:// www.jfree.org/jfreechart/
40. inet4.cpp [Электронный ресурс]. - Режим доступа: https://github.com /hioa-cs/IncludeOS/blob/master/src/net/inet4.cpp
41. QEMU [Электронный ресурс]. - Режим доступа: http://www.qemu-project.org/