

Барнаул 2017г.

## Реферат

В данной работе была спроектирована и разработана система горизонтального масштабирования виртуальных машин и диспетчеризации запросов к исполняемым на них микросервисам.

Объём работы — 154 страницы, включающих 21 рисунок, 42 используемых источников.

Ключевые слова: микросервис, unikernel, виртуальная машина, горизонтальное масштабирование.

## The abstract

Within the scope of this work was done the design and the development of the system intended for a horizontal scaling of virtual machines and for a dispatching of requests for execution inside microservices.

The size of the work – 154 pages including 21 figures. 42 sources were used.

Keywords: microservices, unikernel, virtual machines, horizontal scaling.

Работа допущена к публикации с изъятием страниц с 91 по 154 в соответствии с п. 38 Приказа Минобрнауки от 29.06.2015г. № 636

						МД 09.04.04.01.000 ПЗ			
Изм.	Коп. уч.	Лист	№ док.	Подпись	Дата				
Разработал		Борисов В.В.				Разработка горизонтально масштабируемой системы легковесных виртуальных машин	Стадия	Лист	Листов
Рук. проекта		Казаков М.Г.					У	2	154
							АлтГТУ ФИТ 8ПИ-51		
Н. контролёр		Потупчик А.И.							
Зав. кафедрой		Кантор С.А.							

## Содержание

Введение.....	5
1 Описание предметной области.....	7
1.1 Микросервис.....	7
1.2 Гипервизор.....	8
1.3 IncludeOS.....	9
2 Обзор конкурирующих систем.....	13
2.1 Виртуализация на основе продуктов VMware.....	13
2.2 Виртуализация на основе продуктов Xen.....	15
2.3 Виртуализация на основе продуктов Hyper-V.....	17
3 Архитектура IncludeOS.....	19
3.1 Принцип нулевых издержек.....	19
3.2 Статически линкующиеся библиотеки и GCC-toolchain .....	19
3.3 Стандарт библиотек.....	20
3.4 Сетевой драйвер Virtio .....	21
3.5 Асинхронный ввод-вывод и IRQ.....	21
4 Постановка задачи.....	22
5 Подход к решению задачи.....	24
5.1 Клиент.....	25
5.2 Сервер.....	26
5.3 Контроллер.....	28
5.4 Консоль администратора.....	30
5.5 Виртуальная машина.....	31
5.6 База данных.....	33
6 Логика работы приложения.....	36
6.1 Бизнес-логика системы.....	36
6.2 Транспортный уровень системы.....	37
6.3 Обработка входящего запроса.....	38
6.4 Мониторинг виртуальных машин.....	39

7 Описание программного обеспечения.....	41
7.1 Классы реализующие клиентский компонент системы.....	41
7.2 Классы реализующие консоль администратора.....	47
7.3 Классы реализующие контроллер.....	53
7.4 Классы реализующие сервер.....	57
7.5 Классы реализующие модель данных.....	64
7.6 Классы реализующие web сервис.....	73
8 Результаты работы системы.....	74
Заключение.....	77
Список использованных источников.....	78
ПРИЛОЖЕНИЕ А ЗАДАНИЕ НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ.....	82
ПРИЛОЖЕНИЕ Б РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.....	84
Б.1 Установка и настройка сервера диспетчеризации.....	84
Б.2 Создание web сервиса на основе IncludeOS.....	87
Б.3 Установка и запуск контроллера виртуальных машин.....	87
Б.4 Клиентский компонент.....	88
Б.5 Консоль администратора.....	89
ПРИЛОЖЕНИЕ В ИСХОДНЫЙ КОД ПРОГРАММНОГО ПРОДУКТА.....	91
В.1 SQL скрипт создания базы данных.....	91
В.2 Сервер диспетчеризации.....	92
В.3 Контроллер виртуальных машин.....	108
В.4 Клиентский компонент системы.....	113
В.5 Консоль администратора.....	129
В.6 Модель данных.....	141
В.7 Web сервис.....	152

## Введение

Мир вокруг постоянно развивается, появляются всё новые технологии и чтобы оставаться «на плаву» даже самые маленькие компании (и не только из сферы IT) начинают использовать облачные технологии.

Облачные вычисления[1] - это технология распределённой обработки данных, в которой компьютерные ресурсы и мощности предоставляются пользователю как интернет-сервис. Т.е. здесь инфраструктура предоставляется как услуга. Соответственно потребителю (компании) предоставляются средства обработки данных, хранения, сетей и других базовых вычислительных ресурсов, на которых потребитель (компания) может развертывать и выполнять произвольное программное обеспечение, включая операционные системы, приложения, web-сервисы и т.д.

Преимущества облачных технологий:

- 1) Передача облачному провайдеру хранения постоянно увеличивающихся данных.
- 2) Более быстрое и простое разворачивание проекта и сайта и т.д.
- 3) Быстрая скорость реакции на изменение нагрузки на систему, т. е. в любой момент времени можно арендовать ещё вычислительные ресурсы при увеличении нагрузки на систему и наоборот, отказаться от некоторого объёма ресурсов при снижении нагрузки на систему.
- 4) Малая вероятность отказа системы из-за технической неисправности оборудования.
- 5) Практически полное отсутствие операционных затрат — затрат на текущее содержание оборудования.
- 6) Практически полное отсутствие капитальных затрат — единовременных изначальных затрат на покупку оборудования.

Для возможности эффективного использования облачных вычислений информационные системы должны обладать возможностью горизонтального

масштабирования. Применяемые ранее монолитные архитектуры довольно сложно эффективно масштабировать, поэтому в настоящее время принято разрабатывать системы на основе множества микросервисов.

Микросервис - это минимальная единица функциональности системы, слабо связанная с другими компонентами и обрабатывает однотипные запросы, обычно передаваемые по протоколу http. Микросервисы исполняются в виртуальной среде, как правило, для их запуска используются традиционные операционные системы, такие как: Windows, unix-подобные, FreeBSD и другие. Данные операционные системы, однако, не обладают архитектурой, ориентированной на такой режим исполнения и иногда могут накладывать дополнительные накладные расходы на вычислительные ресурсы, превосходящие затраты самого микросервиса.

В противовес традиционным операционным системам существует новый подход под названием Unikernel идеально подходящий для запуска горизонтально масштабируемых микросервисов. Unikernel система ориентирована на выполнение единственной задачи, поэтому лишены чрезмерных накладных расходов.

Данная работа является пояснительной запиской к магистерской диссертации, выполненной на кафедре «Прикладная математика» Алтайского государственного технического университета имени И. И. Ползунова. Целями данной работы является:

- 1) Проектирование системы горизонтального масштабирования виртуальных машин и диспетчеризации запросов к исполняемым на них микросервисам.
- 2) Реализация компонентов системы в соответствии с разработанной моделью взаимодействия.

# **1 Описание предметной области**

## **1.1 Микросервис**

Во многих IT компаниях для обработки однотипной информации в больших объёмах используют микросервисы [2]. 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 разворачиваются независимо с использованием полностью автоматизированной среды. Существует абсолютный минимум централизованного управления этими сервисами [3]. Сами по себе эти сервисы могут быть написаны на разных языках и использовать разные технологии хранения данных.

Сервис [4] 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 [5] 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 [6] (для Windows), поддерживать работу неиспользуемых драйверов (видеокарты, драйвер принтера, звуковой карты и т. д.).



### 1.3 IncludeOS

В 2016 году на конференции CppCon [7] был представлен рабочий alpha прототип проекта одномодульной микро операционной системы IncludeOS [8]. IncludeOS[9] 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.

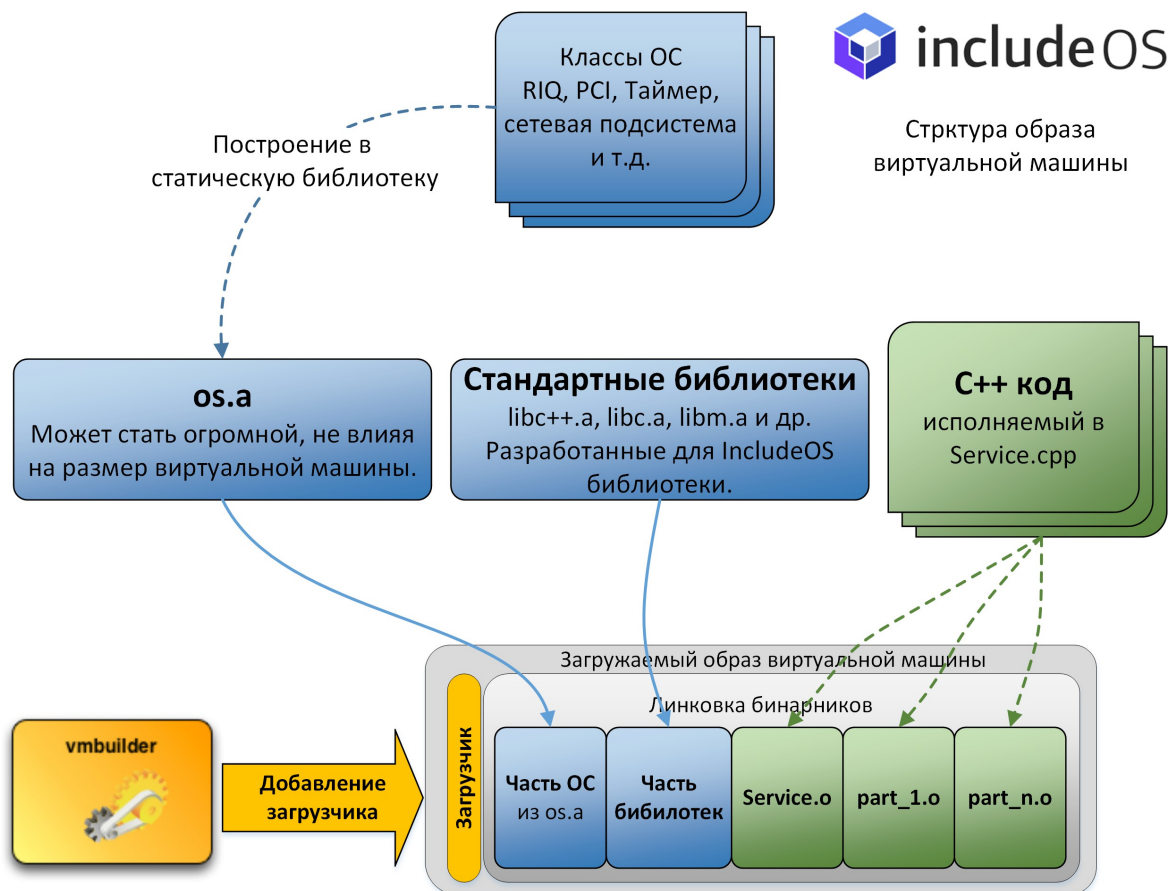


Рисунок 1.1 - Построение образа виртуальной машины

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 отличием 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 выполнения 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.

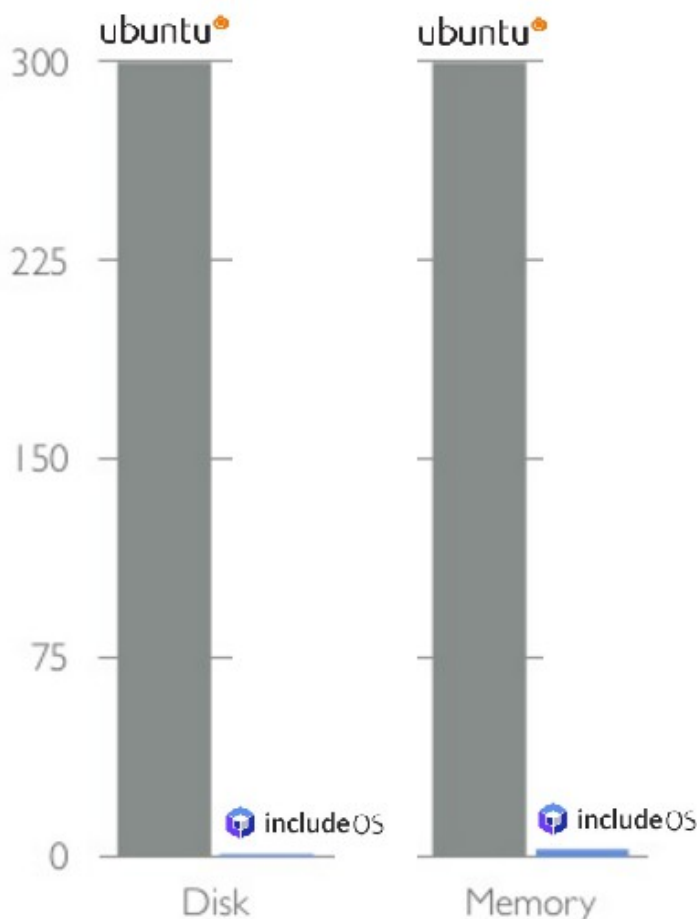


Рисунок 1.2 - Используемые ресурсы Ubuntu и IncludeOS

В связи с этим использование IncludeOS в качестве операционной системы для виртуальной 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 системой, 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 загрузке 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 Unikernel системы.

## **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 Unikernel системы.



## **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) Продвинутое инструменты администрирования платные.
- 3) Занимает большое количество аппаратных ресурсов.
- 4) Не поддерживает Unikernel системы.

## **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 современных 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 операционной 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]. Хотя 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-х (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 виртуальными 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.
  - проектирование протокола запроса на исполнение задачи;
  - проектирование протокола запроса на выдачу информации о состоянии виртуальной машины;
  - проектирование протокола управления состоянием виртуальной машины;
  - проектирование протокола запуска и остановки экземпляров виртуальных машин на хост-компьютере.
- 2) Разработка серверного компонента.
  - разработка подсистемы перенаправления запросов на исполнение задачи;
  - разработка подсистемы сбора данных о состояниях виртуальных машин;
  - разработка подсистемы принятия решений о горизонтальном масштабировании;

- разработка подсистемы отображения информации о текущей загрузке системы.
- 3) Разработка компонента виртуальных машин.
- разработка веб-сервиса, исполняемого в виртуальной машине;
  - разработка дополнения веб-сервиса, исполняемого в виртуальной машине, обеспечивающее предоставление информации о загрузке системы.
- 4) Разработка контроллера виртуальной машины.
- разработка внутреннего обработчика контроллера, для обработки вывода виртуальной машины;
  - разработка механизма перенаправления вывода консоли виртуальной машины во внутренний обработчик контроллера;
  - разработка механизма запуска, остановки и получения состояния о загрузке 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.



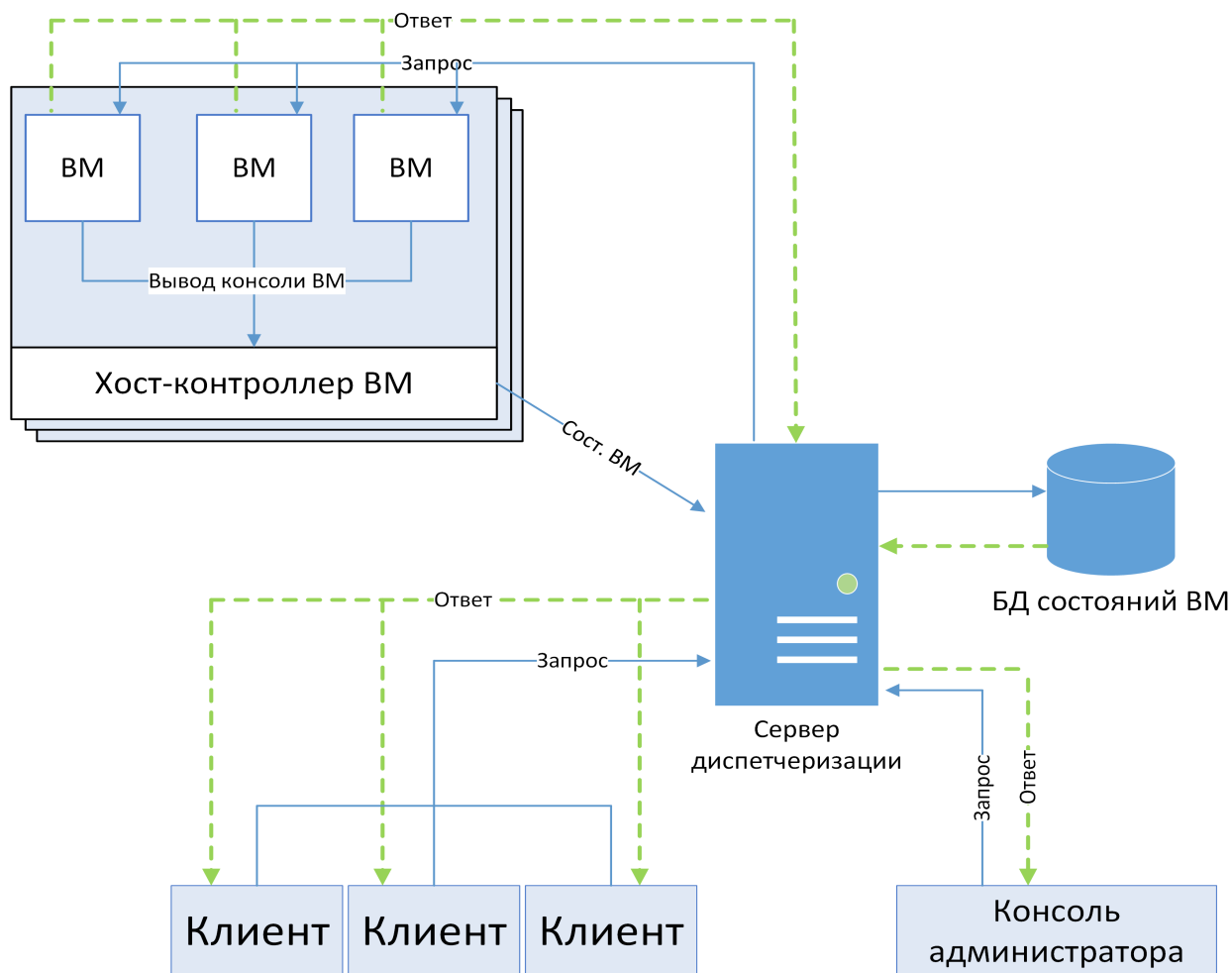


Рисунок 5.1 — Общая структура системы

## 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 запускаться 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: В-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 загрузке 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 `dhcpr` (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 загруженности. 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 `net4` [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.



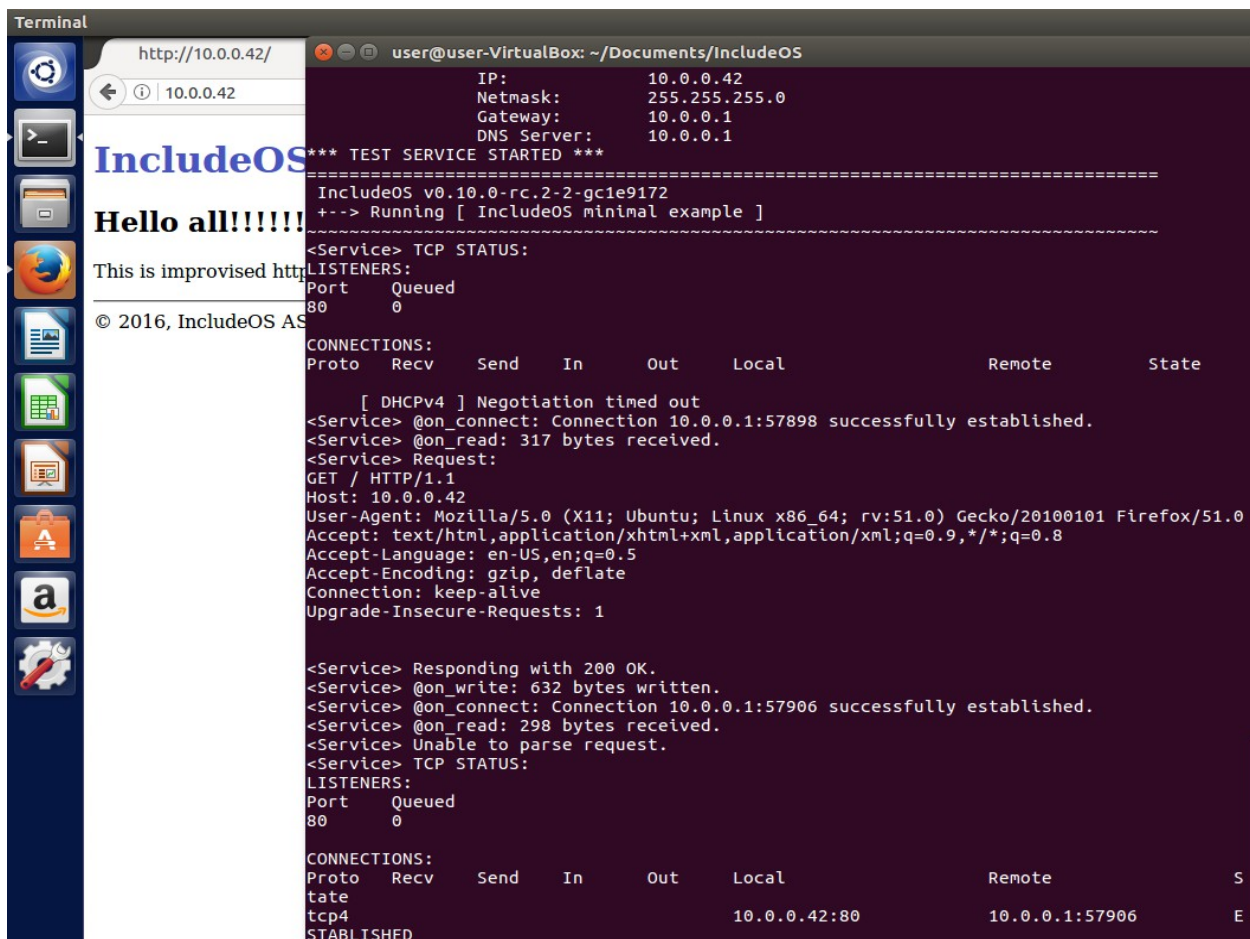


Рисунок 5.1 — Запущенный тестовый микросервис

## 5.6 База данных

Для хранения и обработки состояний системы используется реляционная база данных PostgreSQL, она включает в себя:

- 1) Реестр ip-адресов, зарезервированных для использования виртуальными машинами.
- 2) Пул исполняемых в текущий момент виртуальных машин.
- 3) Набор параметров, позволяющих настраивать поведения системы.

Реестр ip-адресов используется для масштабирования системы, а именно, в нём хранится набор ip адресов, выдаваемых виртуальным машинам при запуске. Всё последующее взаимодействие между сервером и виртуальными машинами происходит посредством отправления http запросов со стороны сервера на ip адреса, хранящиеся в базе данных.

Реестром ip-адресов в базе данных является таблица `ip_pool`.

Рассмотрим поля таблицы `ip_pool`:

- 1) `id_ip` — первичный ключ таблицы.
- 2) `ip` — адрес на котором будет доступен сервис.
- 3) `port` — порт на котором будет доступен сервис.
- 4) `reserve` — флаг резервирования записи, выставляется в `true`, если была команда о запуске новой виртуальной машины с этим адресом и выставляется в `false`, если была команда остановки машины.
- 5) `last_reserve` — время резервирования записи, если с момента резервирования прошло больше времени чем `ip_reserve_ms`, а внешний ключ `id_ip` в `vmachine` не проставился ни к одной записи, то флаг `reserve` снимается.

Пул исполняемых в текущий момент виртуальных машин используется для мониторинга количества запущенных виртуальных машин, а также для слежением за состоянием загруженности каждой виртуальной машины. Таблица реестра `ip`-адресов связана с пулом исполняемых в текущий момент виртуальных машин связью один к одному, так как одной виртуальной машине может быть присвоен только один `ip` адрес.

Пул виртуальных машин представлен в базе данных таблицей `vmachine`, содержащей следующие поля:

- 1) `id_vmachine` — первичный ключ таблицы.
- 2) `id_ip` — внешний ключ таблицы `ip_pool`.
- 3) `cpu` — загрузка процессора виртуальной машины в процентах.
- 4) `ram` — загрузка оперативной памяти в байтах.
- 5) `free_count` — значение, показывающее сколько раз подряд виртуальная машина отправляла состояние, указывающее, что она простаивает.
- 6) `last_connect` — время последнего обновления состояния.

Для настройки системы предусмотрена отдельная таблица базы данных — `system_params`, она имеет следующие поля:

Поля:

- 1) `key` — ключ.

2) value – значение.

Для корректной работы системы в таблице system\_params должны храниться следующие обязательные записи:

- 1) free\_count — максимальное количество простаивающих состояний виртуальной машины подряд.
- 2) min\_vm\_count — минимальное количество виртуальных машин, запущенное на контроллере.
- 3) last\_connect\_ms – время последнего обновления состояния виртуальной машины, после которого она считается выключенной.
- 4) ip\_reserve\_ms — максимальное время резервирования ip без проставления внешнего ключа id\_ip в vmachine.

Схематичное представление структуры базы данных предвидено на рисунке 5.2.

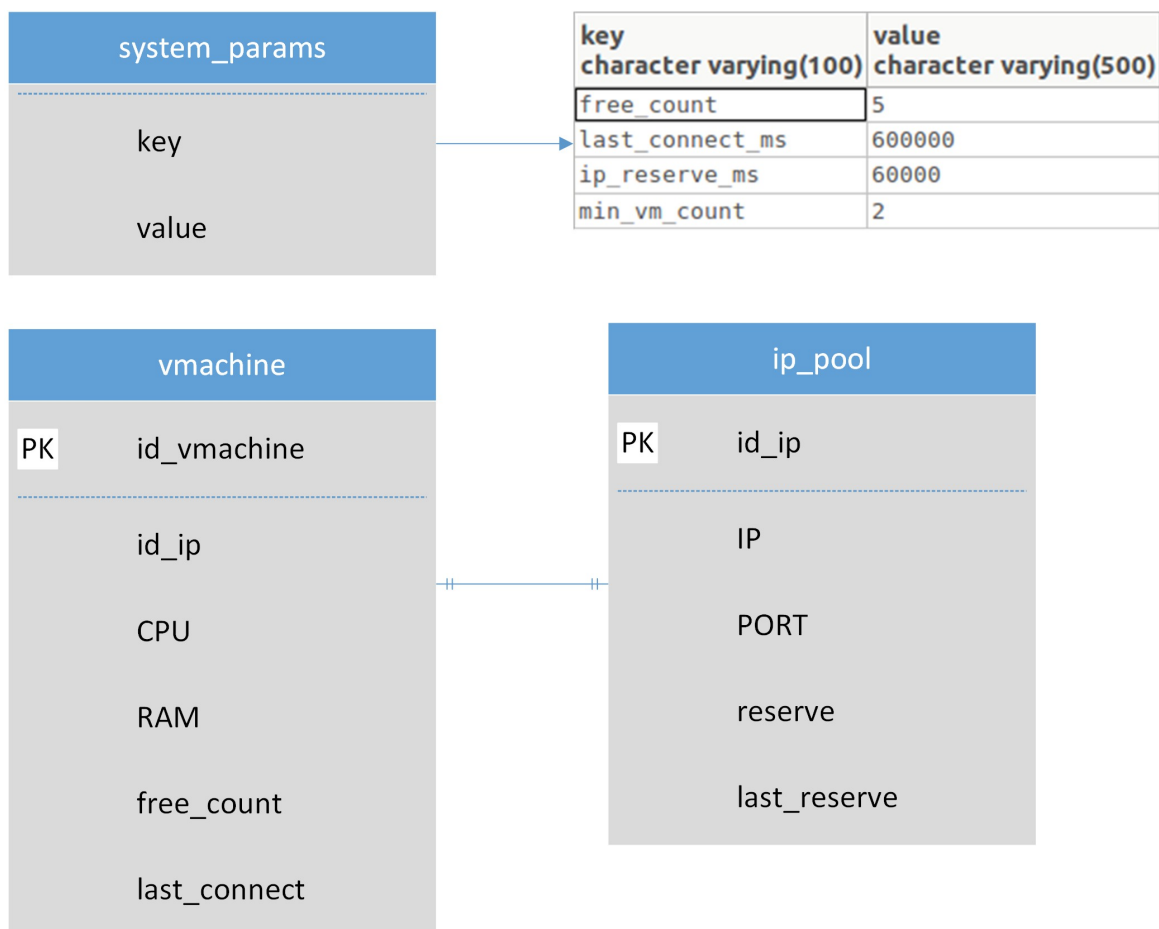


Рисунок 5.2 — Схематичное представление структуры базы данных

## 6 Логика работы приложения

### 6.1 Бизнес-логика системы

Бизнес-логика системы строится на обработке данных, полученных от клиента. В данной реализации в системе могут передаваться и обрабатываться только текстовые данные. Бизнес-логика приложения взаимодействует с двумя компонентами системы: с клиентским модулем и интерфейсом web-сервисов на уровне виртуальной машины. Клиентский модуль позволяет абстрагироваться от топологии инфраструктуры среды исполнения, его задача состоит в передаче формируемых на клиентской стороне запросов на, транспортный уровень. Микросервис абстрагирован от данных клиентской стороны и маршрута прохождения запроса.

Клиентских компонентов в системе может быть огромное количество, в разы превосходящее количество запущенных виртуальных машин, но эта проблема решается на уровне IncludeOS, а именно, очередью входных соединений.

Схематичное представление обработки запросов с точки зрения бизнес логики представлено на рисунке 6.1.

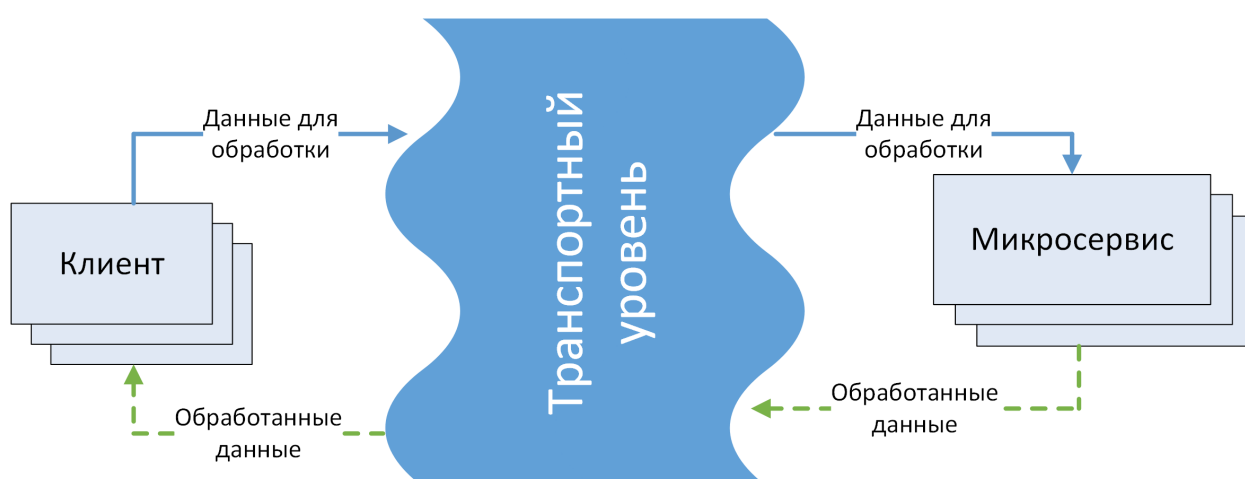


Рисунок 6.1 — Обработка запросов с точки зрения бизнес логики

## **6.2 Транспортный уровень системы**

Транспортный уровень реализует основную логику системы, а именно, горизонтально масштабирует легковесные виртуальные машины.

В транспортный уровень входят:

- 1) сервер диспетчеризации;
- 2) база данных;
- 3) контроллер;
- 4) виртуальные машины, с запущенными на них сервисами для обработки данных;
- 5) консоль администратора.

Сервер диспетчеризации реализуют диспетчеризацию запросов web-сервисов и горизонтальное масштабирование виртуальных машин. В зависимости от текущего состояния системы может быть принято решение о запуске дополнительных виртуальных машин и об остановке запущенных.

Входящий запрос на обработку передаётся одной из виртуальных машин, где обрабатывается Web сервисом на уровне прикладной бизнес-логики. Текущие сведения о состоянии системы можно получить в консоли администратора.

Схематичное представление транспортного уровня системы представлено на рисунке 6.2.

Описание каждого элемента разработанной системы, входящего в транспортный уровень, было представлено в пятой главе.

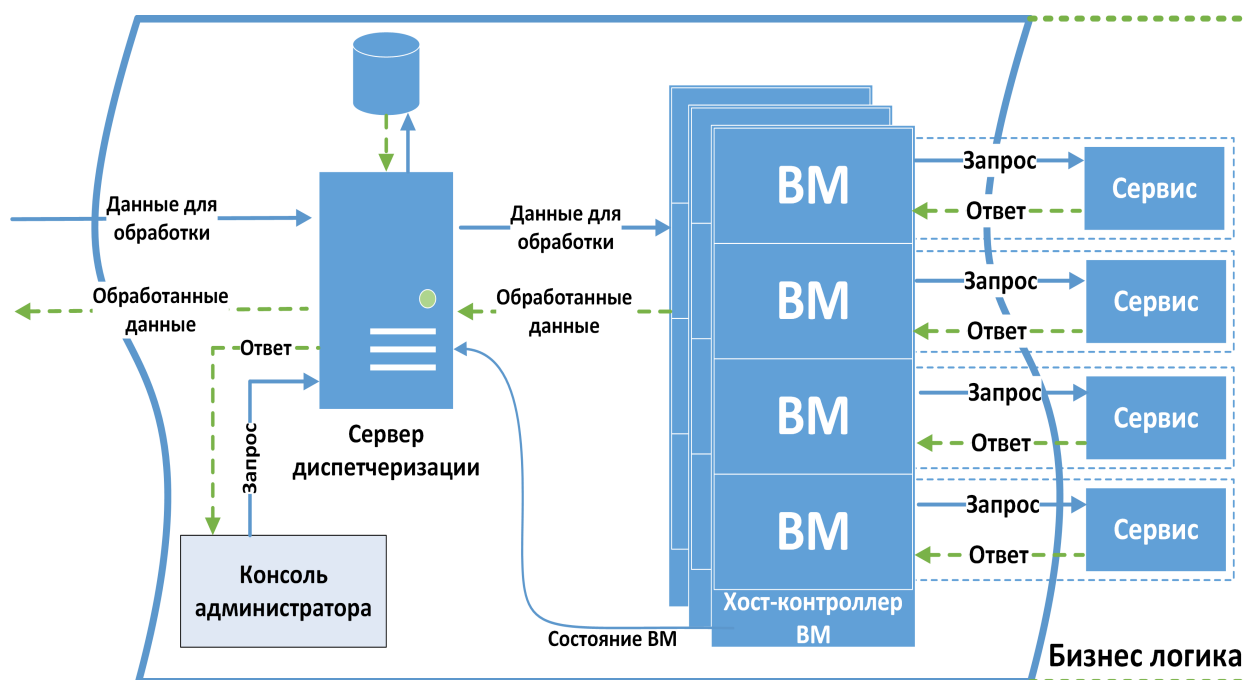


Рисунок 6.2 — Транспортный уровень системы

### 6.3 Обработка входящего запроса

Диаграмма процесса обработки входных запросов представлена на рисунке 6.3. При наличии простаивающих виртуальных машин, запрос на обработку отправляется одной из них, с одновременным сбросом простаивающих состояний, во избежания выключения виртуальной машины во время обработки запроса и соответственно потери данных клиента.

При отсутствии простаивающих виртуальных машин выбирается наименее загруженная, у неё проставляется максимальная степень загруженности (этот шаг не влияет на корректную работу системы, т. к. данные станут актуальными при следующей отправке виртуальной машиной своего состояния), чтобы при следующем запросе на обработку от клиента повторно не выбиралась эта же виртуальная машина. Если все виртуальные машины максимально загружены, то рандомно выбирается машина для обработки запроса. После получения ответа от выбранной виртуальной машины, обработанные данные пересылаются клиентскому модулю.

Исходя из текущей загруженности системы может быть принято решение о запуске еще одного экземпляра виртуальной машины. При

отсутствии свободных слотов виртуальной машины, выбирается для обработки запроса наименее загруженная.

Схематичное представление обработки входящего запроса представлено на рисунке 6.3.

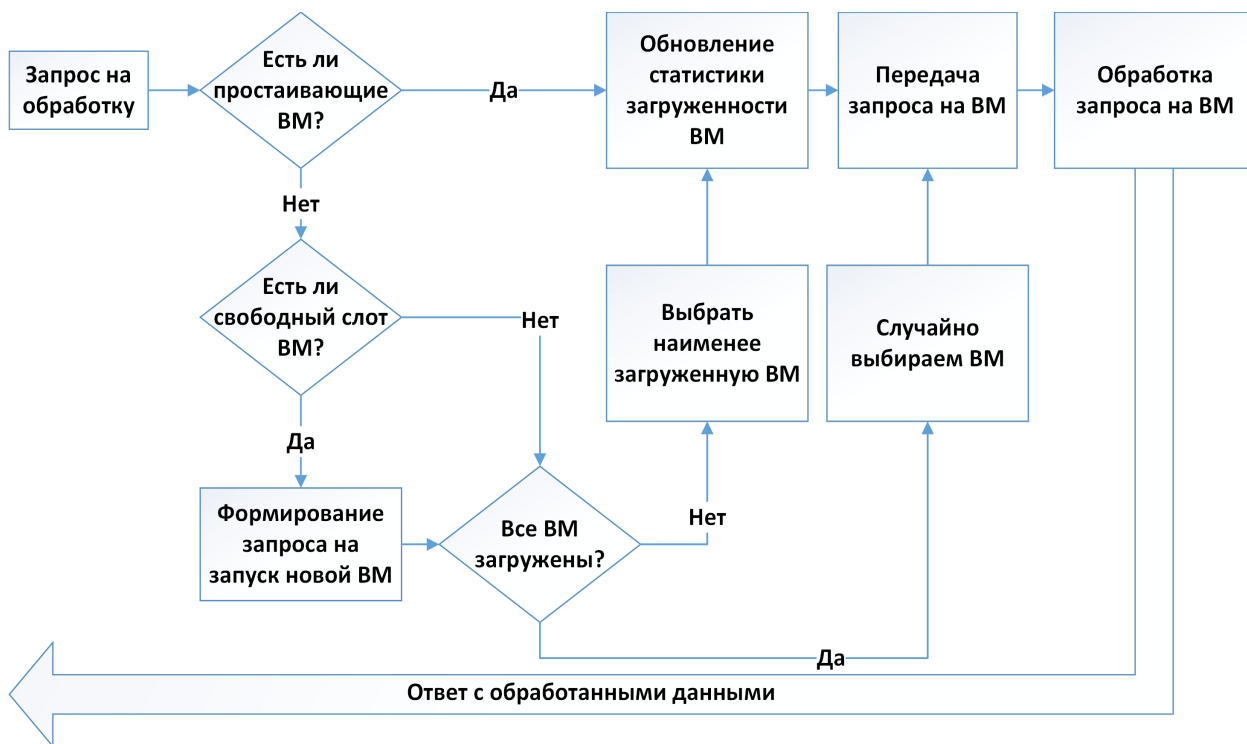


Рисунок 6.3 — Обработка входящего запроса

## 6.4 Мониторинг виртуальных машин

Для предоставления обратной связи, каждая виртуальная машина с заданной частотой выводит в выходной поток информацию о загрузке. Контроллер виртуальных машин считывает информацию, полученную от виртуальной машины и отправляет сведения на сервер диспетчеризации.

При получении сведений о состоянии виртуальной машины, сервер либо регистрирует в базе данных новую виртуальную машину, либо обновляет состояние уже имеющейся. Исходя из тренда загрузки виртуальной машины, при падении нагрузки ниже порогового значения сервер диспетчеризации принимает решение об её остановке и отправляет соответствующую команду.

Схематичное представление мониторинга виртуальных машин представлено на рисунке 6.4.

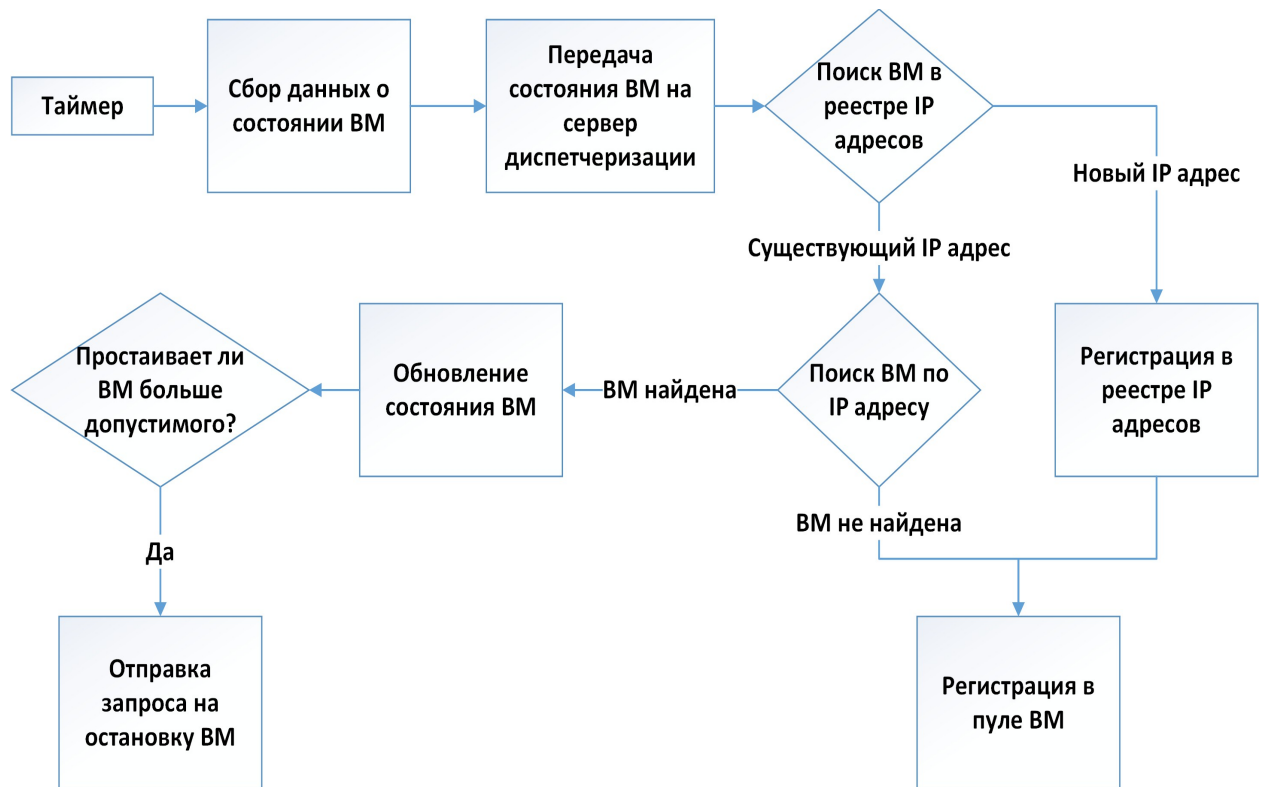


Рисунок 6.4 — Мониторинг виртуальных машин



## 7 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.

### 7.1 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:**

- Container container — контейнер элементов.
- boolean enable — флаг доступности.

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

Параметры:

- 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:

- 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:

- 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:

- `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:

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

Параметры:

- `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:

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

Параметры:

- `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:

- `String fileName` - название файла.

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

Параметры:

- Map<String, String> args - параметры для сохранения.
- 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:

- Data data — обработанные данные.
  - File file — файла для сохранения.
- 2) public static Data inputLoad(File file) — метод загрузки данных для обработки.

Параметры:

- File file — файла для загрузки.

## 7.2 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:**

- `Container container` — контейнер элементов.
  - `boolean enable` — флаг доступности.
- 7) `private void showError(String messege)` — метод отображения диалога ошибки.



Параметры:

- 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:

- 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:

- 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:

- `Response response` — ответ сервера.

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

Параметры:

- `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:

- `Response response` — ответ сервера.

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

Параметры:

- 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:

- String title — заголовок окна.
  - VirtualMachine vMachine — информация о виртуальной машине.
- 2) private static CategoryDataset createDataset(VirtualMachine vMachine) — метод подготовки данных для отображения.

Параметры:

- 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:

- String fileName - название файла.

2) public static boolean setConfig(Map<String, String> args, String fileName)

— сохранение конфигурационных параметров в файл.

Параметры:

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

### 7.3 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:

- String ip - ip виртуальной машины.
  - String sIp — ip сервера для отправки информации о загрузке виртуальной машины.
  - 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 загрузке 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:

- `Response response` — ответ сервера.

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

Параметры:

- `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.

## 7.4 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:

- ServletConfig config — конфигурация сервера.
- 2) protected void doPost(HttpServletRequest request, HttpServletResponse response) — получение post запроса.

Параметры:

- `HttpServletRequest request` — данные запроса на сервер.
- `HttpServletResponse response` — данные ответа сервера.

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

Параметры:

- `HttpServletRequest request` — данные запроса на сервер.
- `HttpServletResponse response` — данные ответа сервера.

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

Параметры:

- `HttpServletRequest request` — данные запроса на сервер.
- `HttpServletResponse response` — данные ответа сервера.

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

Параметры:

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

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

Параметры:

- `List<VirtualMachine> snapshot` - список работающих виртуальных

машин.

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

Параметры:

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

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

Параметры:

- ServerHandler sh — серверный обработчик данных.
- 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:

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

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

Параметры:

- Map<String, String[]> params — параметры системы.
- 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:

- `String url` — адрес виртуальной машины.
- `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:

- `String url` — адрес виртуальной машины.
- `String data` — данные для обработки.

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

Параметры:

- `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:

- `long maxVM` — максимальное время отсутствия обновлений для записей таблицы виртуальных машин.
  - `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:

- `String ip` — новый ip адрес.
  - `int port` — порт, соответствующий новому ip адресу.
- 6) `private void registerNewVM(VirtualMachine vm, int id_ip)` — регистрация новой виртуальной машины в базе данных.

### Параметры:

- `VirtualMachine vm` — информация о виртуальной машине.
- `int id_ip` — ip адрес новой виртуальной машины.

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

Параметры:

- `String ip` — ip адрес.
- `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:

- `int id_ip` — ключ ip адреса в базе данных.
- `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:

- `VirtualMachine vm` — информация о виртуальной машины.
- `int id_vm` — ключ виртуальной машины в базе.
- `int freeCount` — количество холостых состояний виртуальной машины подряд.
- `int id_ip` — ключ ip адреса в базе данных.

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

Параметры:

- `int id_vm` — ключ виртуальной машины в базе.

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

Параметры:

- `int id_vm` — ключ виртуальной машины в базе.

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

Параметры:

- `VirtualMachine vm` — информация о виртуальной машине.

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

Параметры:

- `VirtualMachine vm` — информация о виртуальной машине.

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

## 7.5 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 отправления 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:

- `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:

- `VirtualMachine vm` — информация о виртуальной машины.
- 6) `private String execute(List<NameValuePair> params)` — выполнение http запроса.

Параметры:

- `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:

- `String TAG` - заголовок в системе логирования.
- `String str` - сообщение.

- 2) `public static void e(String TAG, String str)` — сообщение об ошибке.

Параметры:

- `String TAG` - заголовок в системе логирования.
- `String str` - сообщение.

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

Параметры:

- `String TAG` - заголовок в системе логирования.
- `String str` - сообщение.
- `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 обработки.
- 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:

- `Data data` - данные для обработки.

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

Параметры:

- `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:

- `int errorCode` — код ответа.

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

Параметры:

- `Data data` - обработанные данные.

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

Параметры:

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

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

Параметры:

- `String data` - gson строка.

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

Параметры:

- `String data` - gson строка.

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

Параметры:

- `Response res` — ответ сервера.

**Класс:** `JsonUtils`

**Описание:** Класс для создания `Gson` объекта.

**Поля:** отсутствуют.

**Методы:**

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

**Класс:** `HttpMethods`

**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.

## 7.6 Классы реализующие web сервис

**Класс:** Service

**Описание:** Класс обработки запроса web сервисом

**Поля:** отсутствуют.

**Методы:**

- 1) `void Service::start(const std::string& s)` — инициализация web сервиса.  
Параметры:
  - `std::string& s` — параметры запуска web сервиса.
- 2) `void ping()` - вывод информации о загруженности web сервиса в консоль.
- 3) `double getActive()` - сбор информации о загруженности web сервиса.
- 4) `http::Response handle_request(const std::string& s)` — формирование ответа web сервиса.
  - `std::string& s` - запрос.
- 5) `std::string service_handler(const std::string& s)` — обработка данных запроса.
  - `std::string& s` - запрос.

## 8 Результаты работы системы

Так как web сервис в системе занимается обработкой пользовательских данных, то для проверки работоспособности системы не имеет значения каким образом web сервис будет обрабатывать запросы. Главным условием работы web сервиса было то, чтобы обработка занимала достаточно времени (примерно до 5-6 секунд).

Для тестирования системы был разработан web сервис, вычисляющий количество простых чисел до заданного числа  $N$ . Данная задача отлично подходит для тестирования работоспособности системы, так как при малых значениях  $N$ , в пределах 1 до 10 000 вычисления занимают доли секунды, но при значении  $N$  около 20 000, вычисления занимают до 5 секунд в зависимости от оборудования и загруженности хост-компьютера.

Тестирование, как показано на рисунке 8.1, проводилось на фиксированном количестве запущенных виртуальных машин. Сбор и анализ результатов тестирования производился следующим образом:

- 1) Запускаем  $M$  виртуальных машин на одном хост-компьютере.
- 2) Отправляем в пределах секунды  $K$  запросов (все запросы имеют примерно одинаковую сложность, значения  $N$  в пределах 12 000 — 15 000).
- 3) Подсчитываем среднее время обработки  $K$  запросов.
- 4) При достижении суммарного времени обработки  $K$  запросов в 20 секунд останавливаем эксперимент.

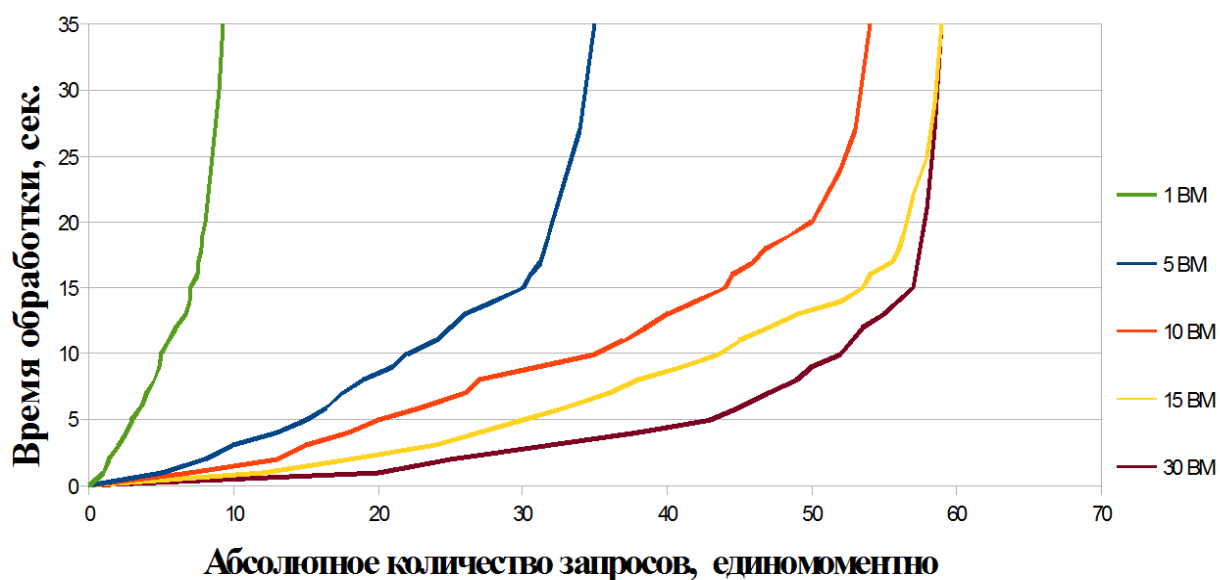


Рисунок 8.1 - Время обработки К запросов на различном количестве виртуальных машин

Также проводилось тестирование в режиме реальной работы системы на том же web сервисе, вычисляющим количество простых чисел до заданного числа N, когда сервер сам принимал решения о запуске, либо остановки виртуальных машин.

На рисунке 8.2 приведён скриншот работы системы. На рисунке видно, что запущено 5 виртуальных машин, состояние которых отображается в консоли администратора. В лог сервера выведена информация о том, что отправлена команда на запуск ещё одной виртуальной машины с ip адресом 10.0.0.46. В контроллере отображена информация о том, что была запущена новая виртуальная машина с ip адресом 10.0.0.46, но ещё не отправила сведения о загруженности, т. к. она не отмечена в консоли администратора. Также в клиентском приложении отображена информация о том, что первые 6 тестовых запроса выполнены успешно.

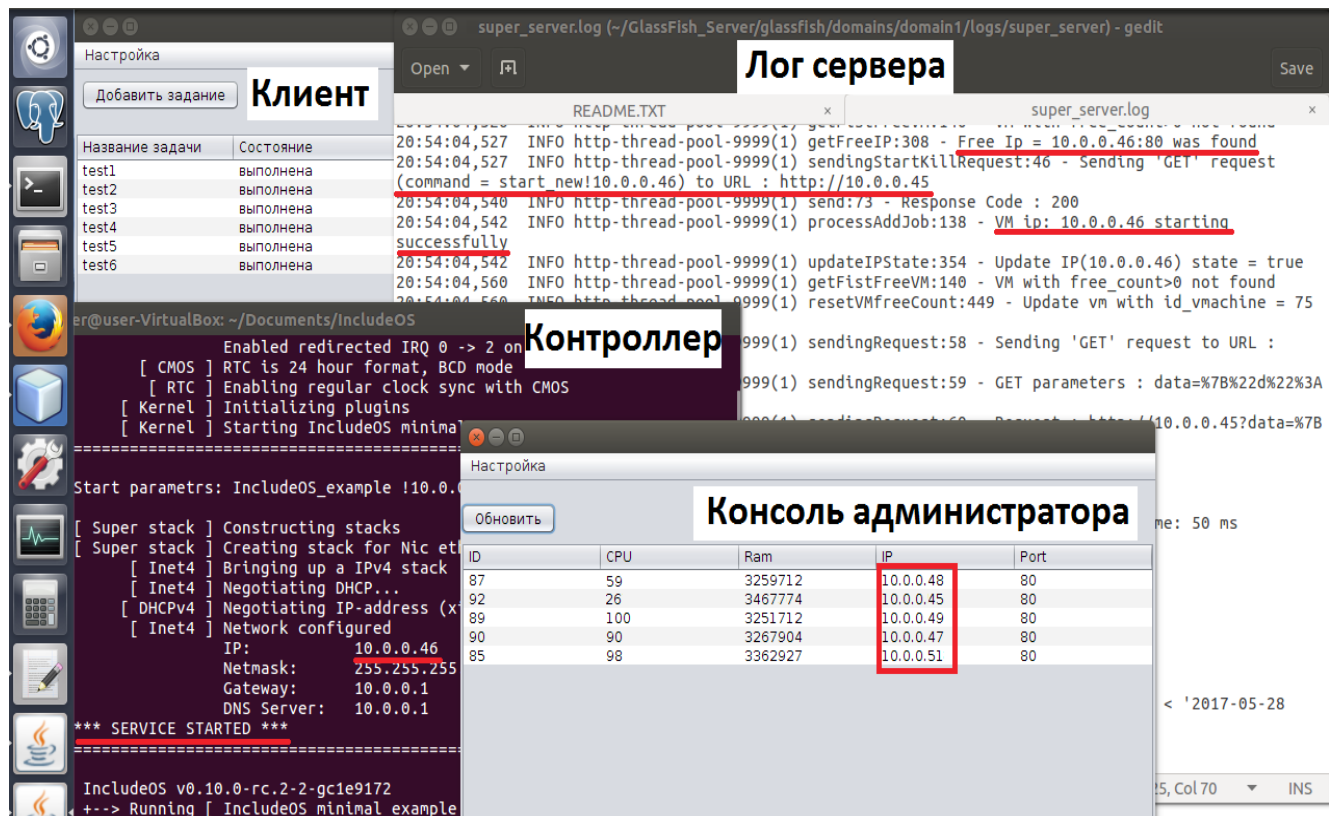


Рисунок 8.2 — Работа системы на хост-компьютере

## **Заключение**

В ходе данной работы была изучена и проанализирована предметная область, рассмотрены наиболее популярные системы масштабирования сервисов, их преимущества и недостатки. С учётом полученных знаний была спроектирована система горизонтального масштабирования виртуальных машин и диспетчеризации запросов к исполняемым на них микросервисам.

Работоспособность спроектированной системы была подтверждена практически. Результатом работы стало работоспособное ПО, обладающее необходимым функционалом и отвечающее постановке задачи и требованиям руководителя.

Планируется дальнейшая доработка системы, а именно:

- будет добавлена возможность передачи файлов для обработки микросервисами;
- будет добавлена возможность масштабирования виртуальных машин на нескольких хост-компьютерах;
- будут разработаны варианты клиентского компонента и консоли администратора для мобильных устройств под управлением ОС Android.

Данная работа была представлена на XIV Всероссийской научно-технической конференции студентов, аспирантов и молодых ученых «Наука и молодежь – 2017»[42]

## Список использованных источников

1. Облачные вычисления [Электронный ресурс]. - Режим доступа: <https://habrahabr.ru/post/111274/>
2. Микросервисы (Microservices) [Электронный ресурс]. - Режим доступа: <https://habrahabr.ru/post/249183/>
3. Преимущества и недостатки микросервисной архитектуры [Электронный ресурс]. - Режим доступа: <http://eax.me/micro-service-architecture/>
4. Микросервисы на практике - SmartMe University [Электронный ресурс]. - Режим доступа: <http://smartme.university/course/microservices-in-practice/>
5. Национальная библиотека им. Н.Э.Баумана Bauman National Library [Электронный ресурс]. - Режим доступа: <http://ru.bmstu.wiki/Гипервизор>
6. Графический интерфейс пользователя [Электронный ресурс]. - Режим доступа: <http://www.microchip.com.ru/Support/GUI.html>
7. CppCon 2016 [Электронный ресурс]. - Режим доступа: <https://cppcon2016.sched.com/>
8. 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>
9. Главная страница сайта IncludeOS [Электронный ресурс]. - Режим доступа: <http://www.includeos.org/>
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](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](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](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.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\\_\(Unix\\_shell\)](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/>
42. Борисов, В. В. Горизонтально масштабируемая система развёртывания сервисов, под управлением микрооперационной системы // Горизонты образования. Приложение. Сборник трудов Научно-техн. конф. "Наука и молодежь - 2017". Секция «Информационные технологии». Подсекция «Программная инженерия». / Алт. гос. техн. ун-т им. И.И.Ползунова. – Барнаул: изд-во АлтГТУ, 2017. – 13 с.

## ПРИЛОЖЕНИЕ А

### ЗАДАНИЕ НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ

Министерство образования и науки Российской Федерации  
ФГБОУ ВО  
«Алтайский государственный технический университет им. И.И. Ползунова»

#### ЗАДАНИЕ

##### на выполнение магистерской диссертации

по направлению 09.04.04 Программная инженерия  
магистранту Борисову Виктору Викторовичу  
(фамилия, имя, отчество)

**Тема:** Разработка горизонтально масштабируемой системы  
легковесных виртуальных машин

Утверждено приказом ректора от 30 декабря 2016г. № Л-4483

Срок исполнения работы 11 июня 2017г.

Задание принял к выполнению \_\_\_\_\_ В.В. Борисов  
(подпись) (инициалы, фамилия)

Барнаул 2017

## 1 Исходные данные

Задание на выполнение, техническая документация на языки программирования Java, C++, SQL, техническая документация на unikernel операционную систему IncludeOS, ресурсы internrt

## 2 Содержание разделов проекта

Наименование и содержание раздела проекта	Срок выполнения	Консультант (ФИО, подпись)
1. Изучение предметной области	20.02.2017г.	Казаков М.Г.
2. Разработка математической модели	24.03.2017г.	Казаков М.Г.
3. Разработка алгоритма	10.04.2017г.	Казаков М.Г.
4. Разработка программного обеспечения	30.05.2017г.	Казаков М.Г.
5. Написание отчёта	18.06.2017г.	Казаков М.Г.

## 3 Рекомендуемая литература

3.1 По научно-технической литературе просмотреть РЖ «Информатика», «Программное обеспечение» за последние 2 года и научно-технические журналы «Программирование», «Программная инженерия» за последние 2 года.

3.2 По нормативной литературе просмотреть указатели государственных и отраслевых стандартов за последний год.

3.3 Патентный поиск провести за 4 года по странам Россия

Научный руководитель \_\_\_\_\_  
(подпись)

М.Г. Казаков \_\_\_\_\_  
(инициалы, фамилия)

# ПРИЛОЖЕНИЕ Б

## РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

### Б.1 Установка и настройка сервера диспетчеризации

Для запуска сервера диспетчеризации необходимо установить и настроить GlassFish server 3.1, а также установить и настроить PostgreSQL и выполнить sql скрипт конфигурации базы данных из приложения В.

Настройка GlassFish server 3.1 производится через любой web браузер, необходимо зайти в Domain Admin Console GlassFish и добавить соответствующие настройки подключения к PostgreSQL в JDBC connection pools как показано на рисунке Б.1 и JDBC resources как показано на рисунке Б.2, с одним исключением: адрес PostgreSQL сервера, имя пользователя и пароль указываются соответственно настройке Вашего PostgreSQL сервера.

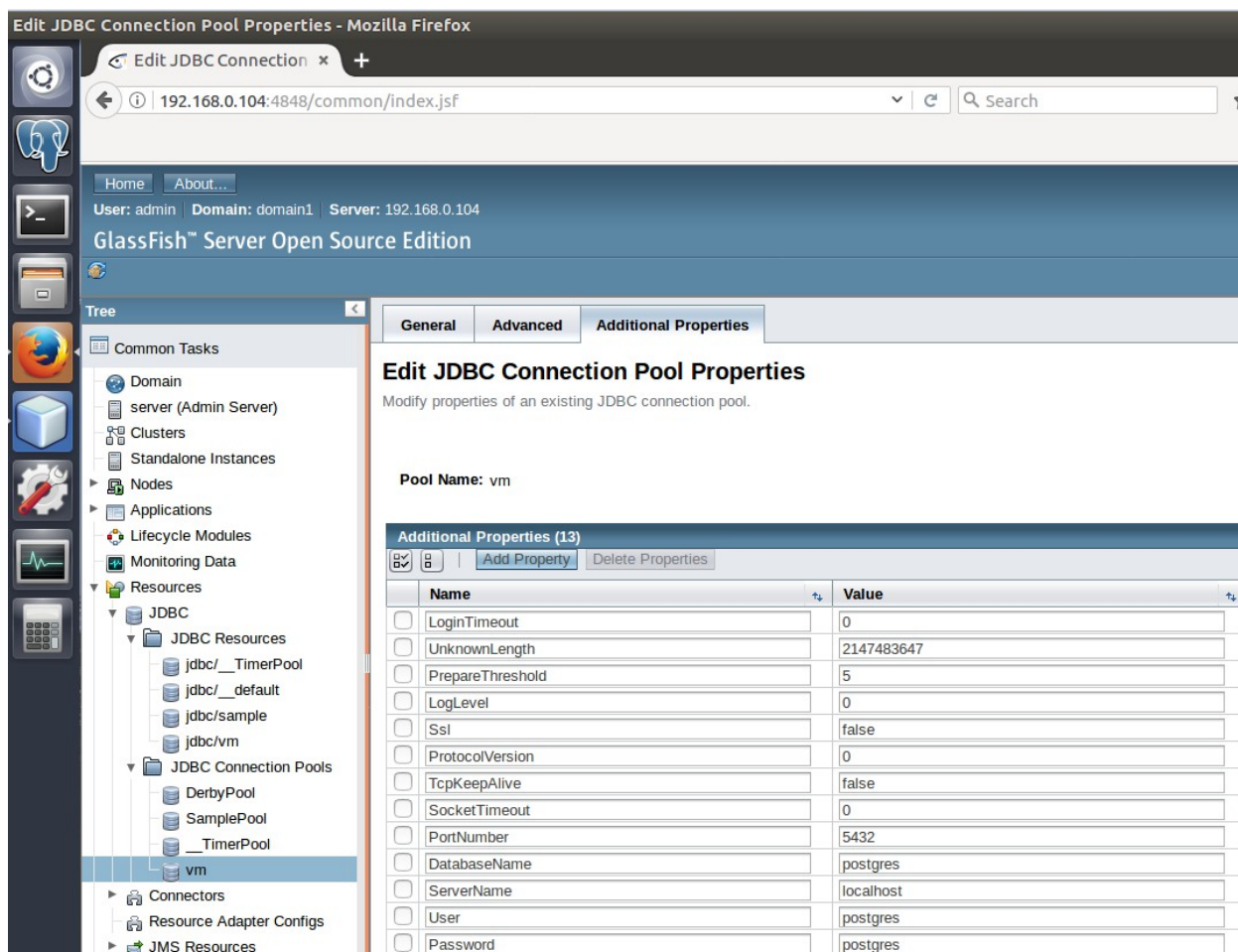


Рисунок Б.1 — Настройка JDBC connection pools

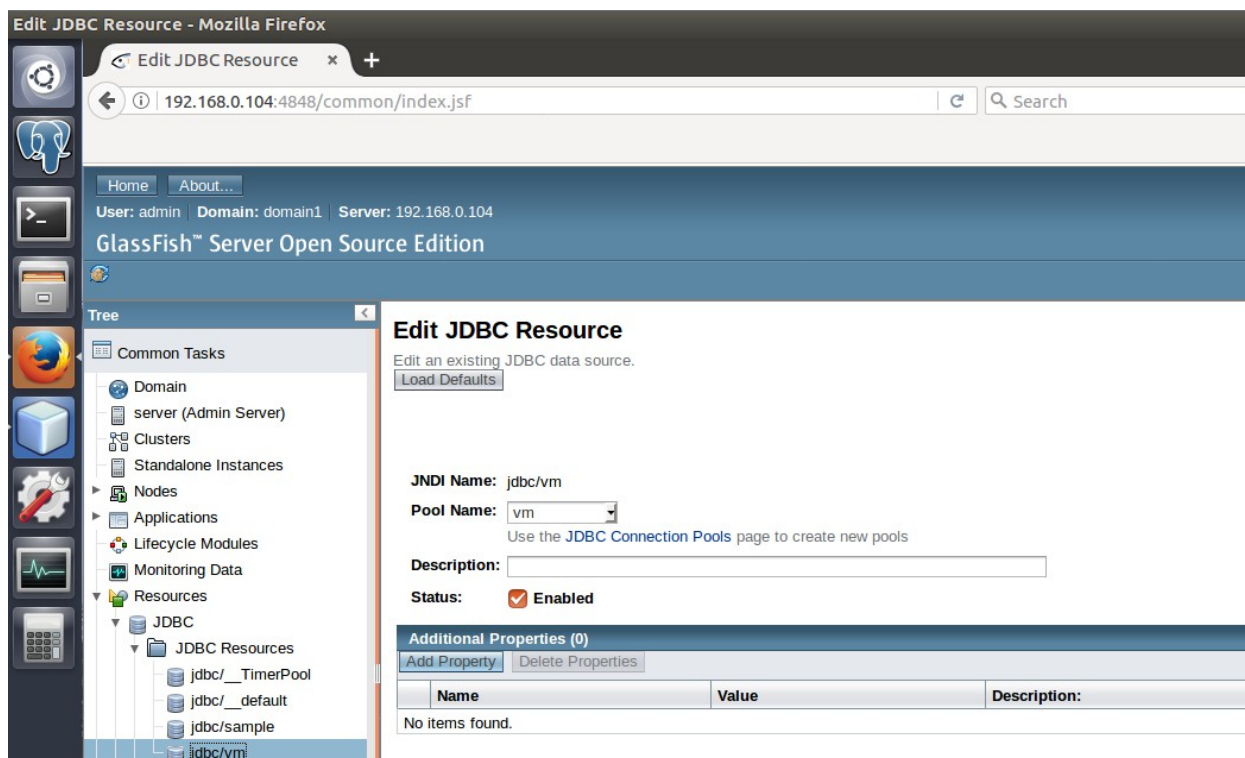


Рисунок Б.2 — Настройка JDBC resources

Проверить правильность настройки JDBC connection pools можно на вкладке General, нажав на кнопку Ping. Сообщение о корректной настройке подключения представлено на рисунке Б.3.

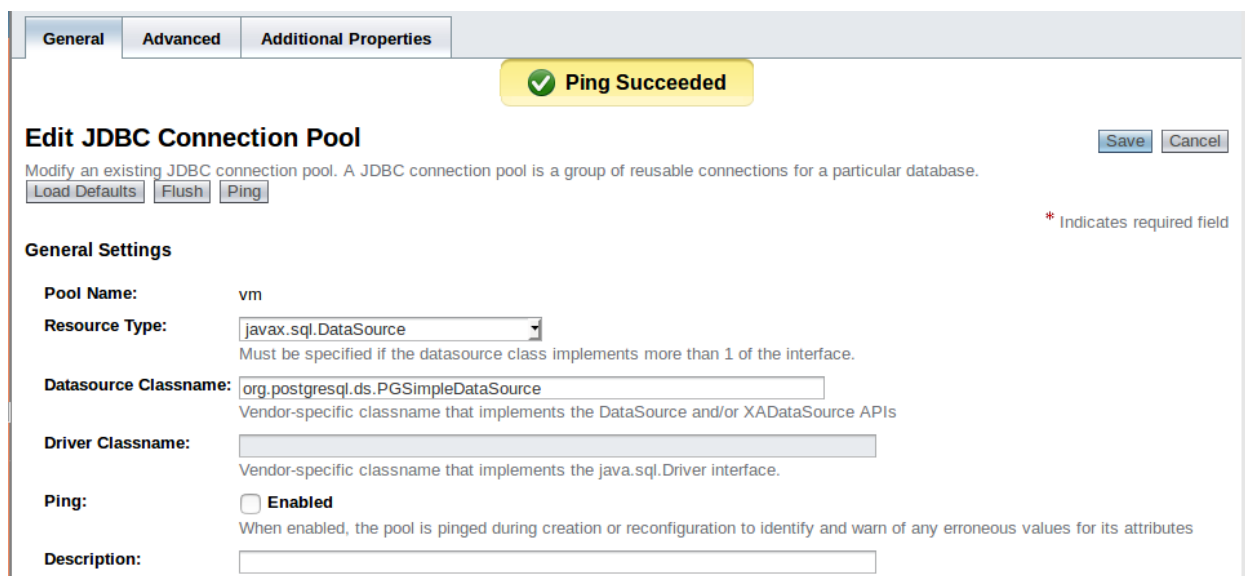


Рисунок Б.3 — Сообщение о корректной настройке JDBC connection pools

Для развёртывания сервер диспетчеризации на GlassFish сервере необходимо в Domain Admin Console GlassFish выбрать пункт меню Applications и кликнуть по кнопке Deploy... после выбрать файл Server.war и

кликнуть по кнопке ОК. Развёрнутый на GlassFish сервере сервер диспетчеризации будет выглядеть как на рисунке Б.4.

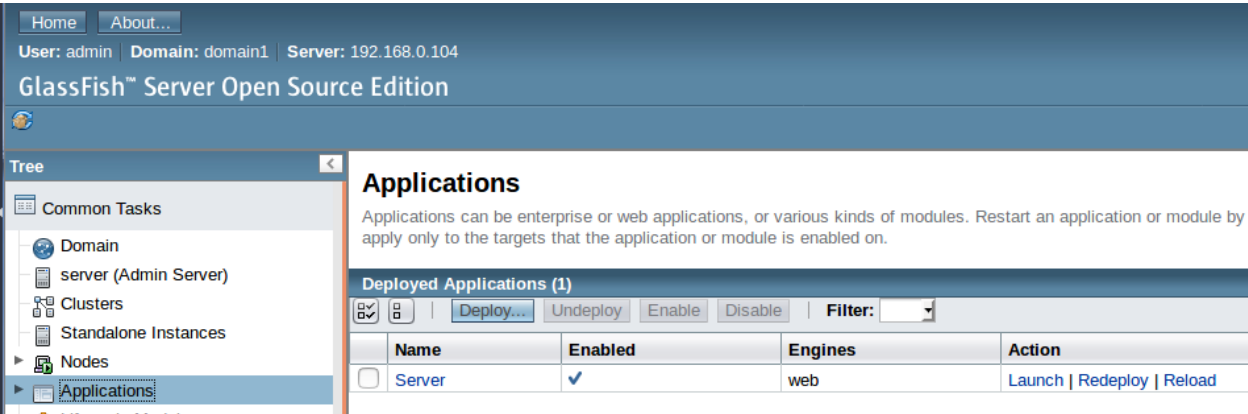


Рисунок Б.4 — Развёрнутый на GlassFish сервере сервер диспетчеризации

После создания базы данных из sql скрипта представленного в приложении В и добавления IP адресов в таблицу ip\_pool базы данных postgresSQL, сервер диспетчеризации можно запускать.

Примерное содержание таблицы ip\_pool базы данных postgresSQL представлено на рисунке Б.5.

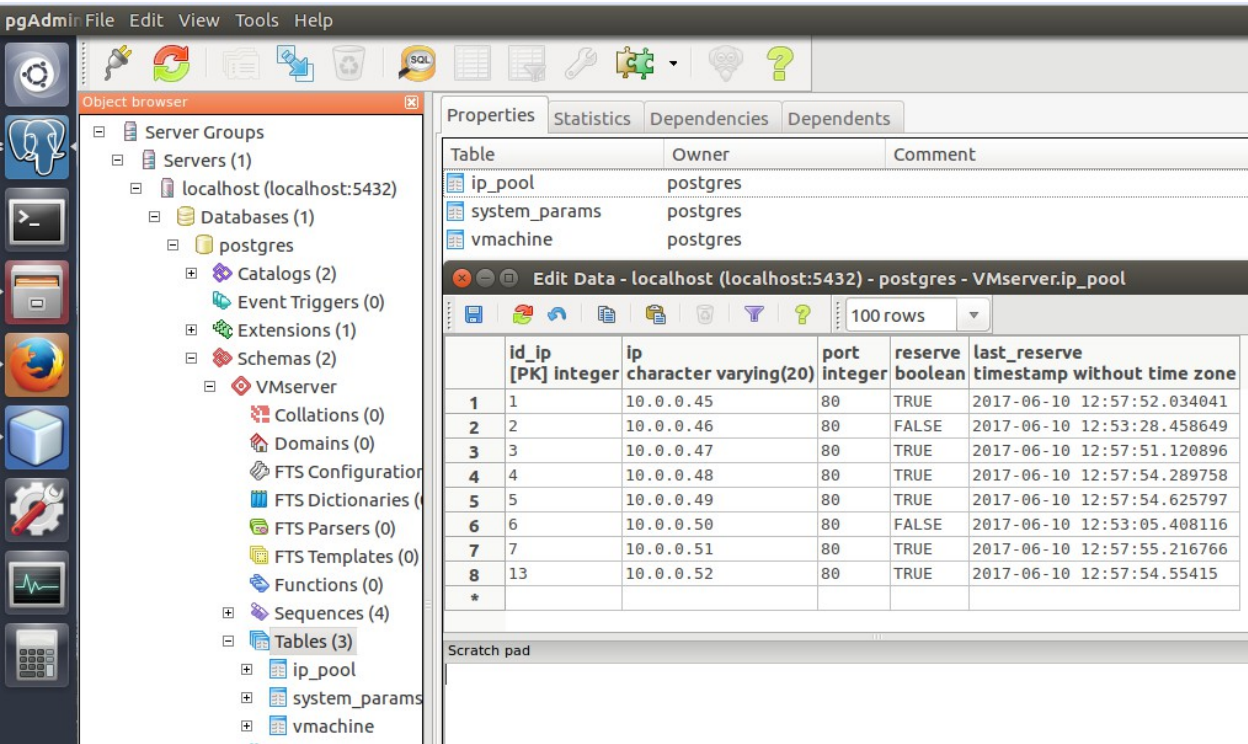


Рисунок Б.5 — Пример заполнения таблицы ip\_pool

Проверить корректность установки сервера диспетчеризации в

GlassFish сервере можно перейдя по ссылке:

<Адресс GlassFish сервера>:<Порт GlassFish сервера>/Server

При успешной установки сервера диспетчеризации будет отображена web страница представленная на рисунке Б.6.

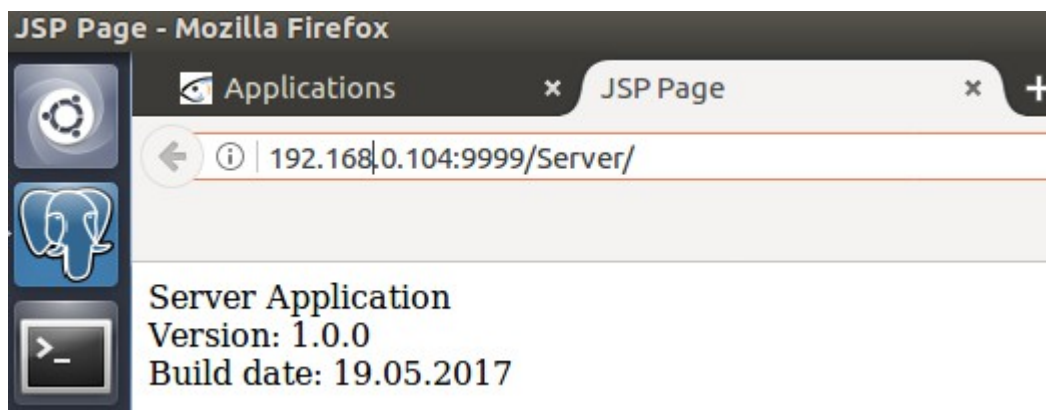


Рисунок Б.6 — Web страница успешной установки сервера диспетчеризации

## Б.2 Создание web сервиса на основе IncludeOS

Для создания web сервиса необходимо развернуть проект IncludeOS из исходных кодов, расположенных на GitHub. Все инструкции по развертыванию проекта IncludeOS из исходных кодов, а также о подробном создании web сервиса расположены на официальном сайте IncludeOS.

## Б.3 Установка и запуск контроллера виртуальных машин

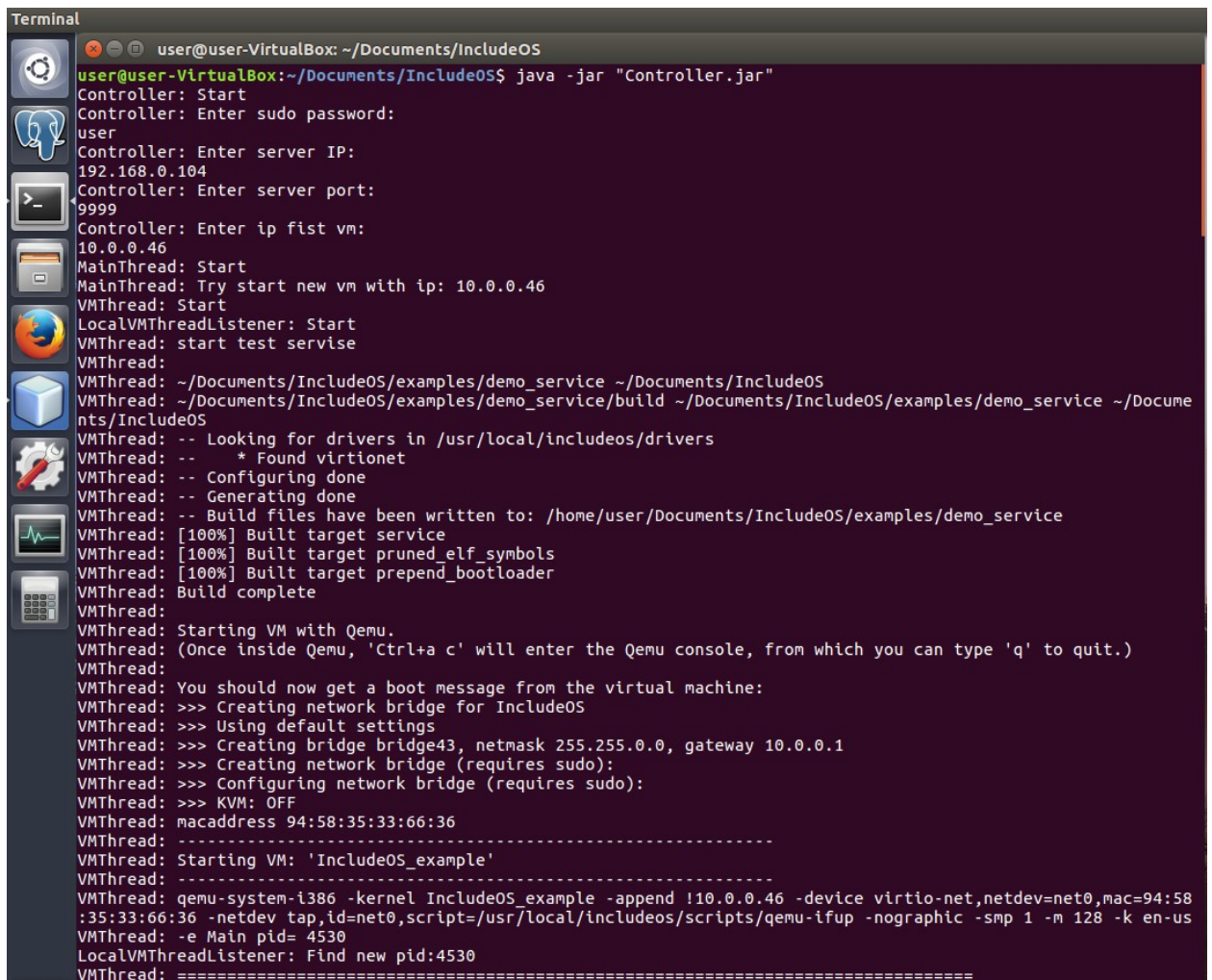
Для установке контроллера виртуальных машин необходимо поместить файл Controller.jar в папку установки IncludeOS.

Запуск контроллера происходит из терминала операционной системы следующей командой: `java -jar "Controller.jar"`.

После запуска контроллера виртуальных машин необходимо указать пароль root пользователя, ip адрес и порт сервера диспетчеризации, а также ip адрес первой запускаемой виртуальной машины.

Пример успешного запуска контроллера виртуальных машин представлен на рисунке Б.7.





```
user@user-VirtualBox: ~/Documents/IncludeOS
user@user-VirtualBox:~/Documents/IncludeOS$ java -jar "Controller.jar"
Controller: Start
Controller: Enter sudo password:
user
Controller: Enter server IP:
192.168.0.104
Controller: Enter server port:
9999
Controller: Enter ip first vm:
10.0.0.46
MainThread: Start
MainThread: Try start new vm with ip: 10.0.0.46
VMThread: Start
LocalVMThreadListener: Start
VMThread: start test service
VMThread:
VMThread: ~/Documents/IncludeOS/examples/demo_service ~/Documents/IncludeOS
VMThread: ~/Documents/IncludeOS/examples/demo_service/build ~/Documents/IncludeOS/examples/demo_service ~/Documents/IncludeOS
VMThread: -- Looking for drivers in /usr/local/includeos/drivers
VMThread: -- * Found virtio-net
VMThread: -- Configuring done
VMThread: -- Generating done
VMThread: -- Build files have been written to: /home/user/Documents/IncludeOS/examples/demo_service
VMThread: [100%] Built target service
VMThread: [100%] Built target pruned_elf_symbols
VMThread: [100%] Built target prepend_bootloader
VMThread: Build complete
VMThread:
VMThread: Starting VM with Qemu.
VMThread: (Once inside Qemu, 'Ctrl+a c' will enter the Qemu console, from which you can type 'q' to quit.)
VMThread:
VMThread: You should now get a boot message from the virtual machine:
VMThread: >>> Creating network bridge for IncludeOS
VMThread: >>> Using default settings
VMThread: >>> Creating bridge bridge43, netmask 255.255.0.0, gateway 10.0.0.1
VMThread: >>> Creating network bridge (requires sudo):
VMThread: >>> Configuring network bridge (requires sudo):
VMThread: >>> KVM: OFF
VMThread: macaddress 94:58:35:33:66:36
VMThread: -----
VMThread: Starting VM: 'IncludeOS_example'
VMThread: -----
VMThread: qemu-system-i386 -kernel IncludeOS_example -append !10.0.0.46 -device virtio-net,netdev=net0,mac=94:58:35:33:66:36 -netdev tap,id=net0,script=/usr/local/includeos/scripts/qemu-ifup -nographic -smp 1 -m 128 -k en-us
VMThread: -e Main pid= 4530
LocalVMThreadListener: Find new pid:4530
VMThread: =====
```

Рисунок Б.7 - Успешный запуск контроллера виртуальных машин

## Б.4 Клиентский компонент

Для запуска клиентского компонента необходимо запустить файл Client.jar. При первом запуске необходимо настроить подключение к серверу диспетчеризации, для этого необходимо перейти в Настройка, Настройка подключения и указать адрес сервера диспетчеризации.

Пример успешного подключения к серверу диспетчеризации представлен на рисунке Б.8.

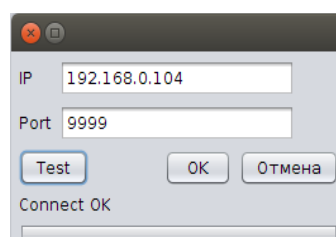


Рисунок Б.8 — Успешное подключение к серверу диспетчеризации



Для отправки данных на обработку необходимо кликнуть на кнопку «Добавить задание», после чего выбрать файл с данными и кликнуть по кнопке «Отправить». Обработанные данные можно сохранить в файл.

Результат обработки данных web сервисом, вычисляющим количество простых чисел до заданного числа  $N$  представлен на рисунке Б.9.

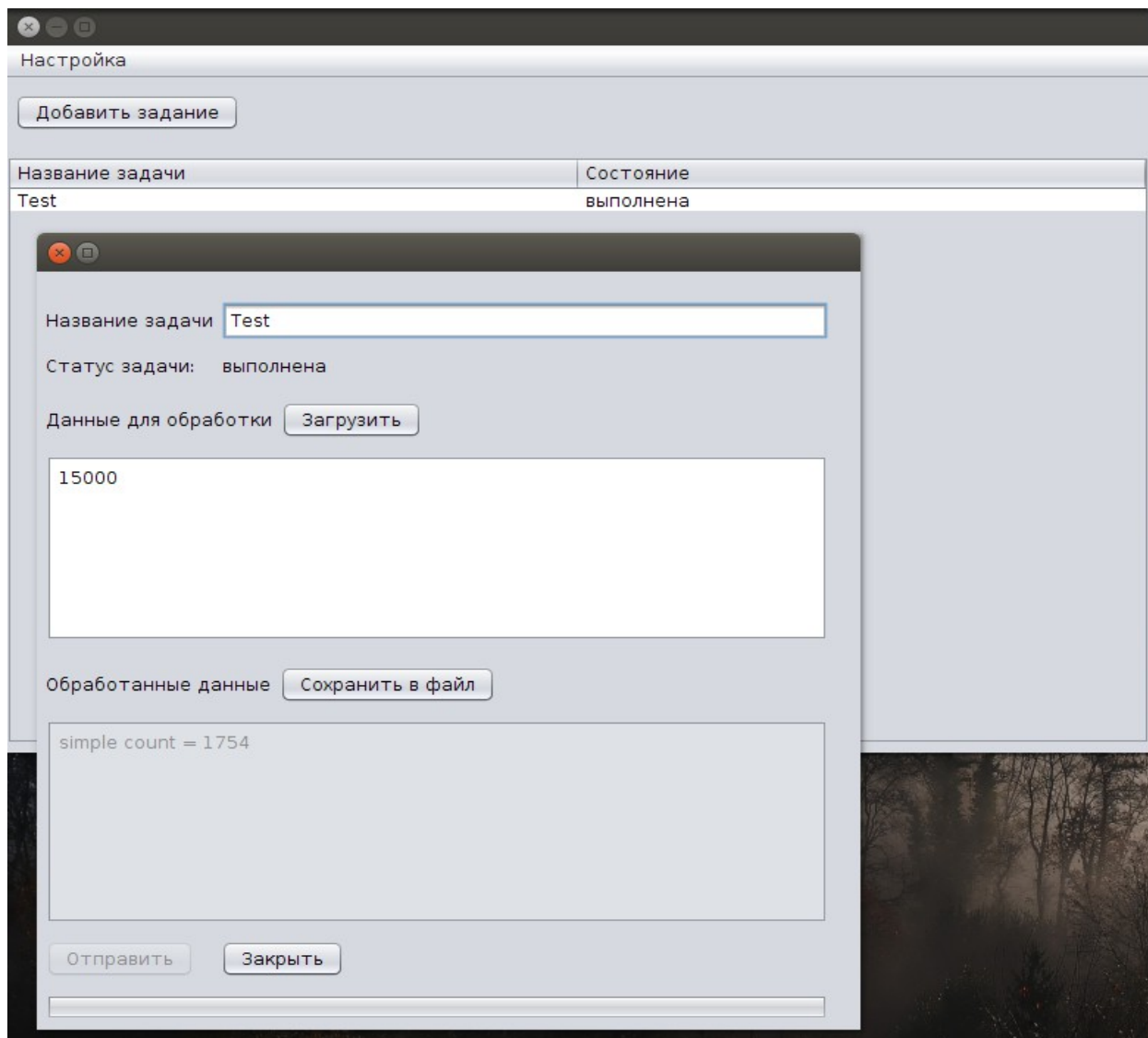


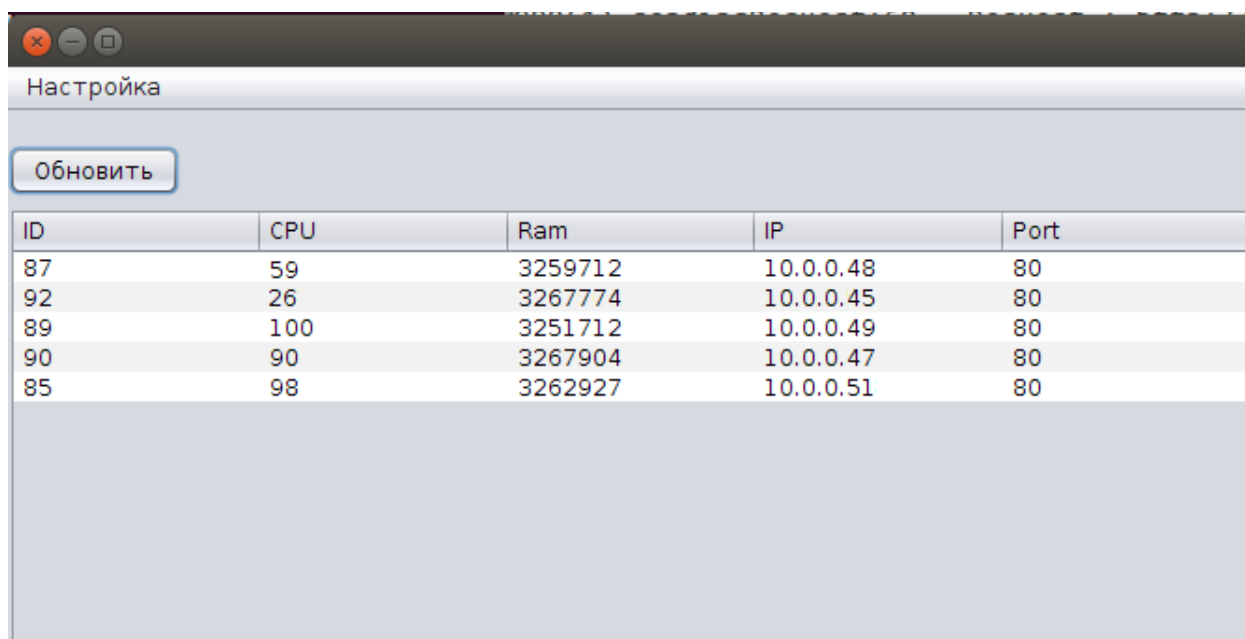
Рисунок Б.9 — Результат обработки запроса

### Б.5 Консоль администратора

Для запуска консоли администратора необходимо запустить файл Admin console.jar. При первом запуске необходимо настроить подключение к серверу диспетчеризации аналогично как в клиентском компоненте.

Главное окно консоли администратора с отображением информации о

запущенных виртуальных машинах и их загрузки представлено на рисунке Б.10.



ID	CPU	Ram	IP	Port
87	59	3259712	10.0.0.48	80
92	26	3267774	10.0.0.45	80
89	100	3251712	10.0.0.49	80
90	90	3267904	10.0.0.47	80
85	98	3262927	10.0.0.51	80

Рисунок Б.10 — Главное окно консоли администратора

## ПРИЛОЖЕНИЕ В

### ИСХОДНЫЙ КОД ПРОГРАММНОГО ПРОДУКТА

#### В.1 SQL скрипт создания базы данных

```
-- Schema: VMserver

-- DROP SCHEMA "VMserver";

CREATE SCHEMA "VMserver"
AUTHORIZATION postgres;

-- Table: "VMserver".ip_pool

-- DROP TABLE "VMserver".ip_pool;

CREATE TABLE "VMserver".ip_pool
(
    id_ip serial NOT NULL,
    ip character varying(20),
    port integer DEFAULT 80,
    reserve boolean DEFAULT false,
    last_reserve timestamp without time zone,
    CONSTRAINT id_ip PRIMARY KEY (id_ip)
)
WITH (
    OIDS=FALSE
);
ALTER TABLE "VMserver".ip_pool
OWNER TO postgres;

-- Table: "VMserver".system_params

-- DROP TABLE "VMserver".system_params;

CREATE TABLE "VMserver".system_params
(
    key character varying(100),
    value character varying(500)
)
WITH (
    OIDS=FALSE
);
ALTER TABLE "VMserver".system_params
OWNER TO postgres;

-- Table: "VMserver".vmachine

-- DROP TABLE "VMserver".vmachine;

CREATE TABLE "VMserver".vmachine
(
    id_vmachine serial NOT NULL,
    cpu integer,
    ram bigint,
```

```

free_count integer DEFAULT 0,
id_ip integer,
last_connect timestamp without time zone,
CONSTRAINT vmachine_pkey PRIMARY KEY (id_vmachine),
CONSTRAINT id_ip FOREIGN KEY (id_ip)
    REFERENCES "VMserver".ip_pool (id_ip) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
    OIDS=FALSE
);
ALTER TABLE "VMserver".vmachine
    OWNER TO postgres;

-- Index: "VMserver".fki_id_ip

-- DROP INDEX "VMserver".fki_id_ip;

CREATE INDEX fki_id_ip
ON "VMserver".vmachine
USING btree
(id_ip);

```

## В.2 Сервер диспетчеризации

```

package server.app.expection;
public class NoArgumentException extends Exception {

    public NoArgumentException() {
    }

    public NoArgumentException(String message) {
        super(message);
    }

    public NoArgumentException(String message, Throwable cause) {
        super(message, cause);
    }

    public NoArgumentException(Throwable cause) {
        super(cause);
    }
}

package server.app.expection;
public class NoFreeVMException extends Exception {

    public NoFreeVMException() {
    }

    public NoFreeVMException(String message) {
        super(message);
    }

    public NoFreeVMException(String message, Throwable cause) {
        super(message, cause);
    }

    public NoFreeVMException(Throwable cause) {
        super(cause);
    }
}

```

```

    }
}

package server.app;

import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.List;
import java.util.Map;
import java.util.logging.Level;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;
import model.data.Data;
import model.data.HttpMethods;
import model.data.Response;
import model.data.SystemParams;
import model.data.VMCommand;
import model.data.VirtualMachine;
import model.io.ServerHandler;
import org.apache.commons.dbutils.DbUtils;
import org.apache.log4j.Logger;
import server.app.expection.NoArgumentException;
import server.app.expection.NoFreeVMException;
import server.db.DBManager;
import server.web.HttpRequestHelper;

/**
 *
 * @author user
 */
public class ServerServlet extends HttpServlet {

    private final Logger log = Logger.getLogger("super_server");
    private DataSource ds;

    @Override
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        log.info("Server init");
        try {
            ds = (DataSource) new InitialContext().lookup("jdbc/vm");
        } catch (NamingException ex) {
            log.error(ex, ex);
            throw new ServletException(ex);
        }
    }

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json;charset=UTF-8");
        long start = System.currentTimeMillis();
        Map<String, String[]> params = request.getParameterMap();
        ServerHandler sh = new ServerHandler(response.getWriter());
    }
}

```

```

Connection conn = null;
DBManager db = null;
log.info("New Request");
log.info("Params: " + params);
try {
    conn = ds.getConnection();
    db = new DBManager(conn);
    db.clearDB(
        Long.parseLong(getSysParam(db.getSystemParams(), SystemParams.LAST_CONNECT)),
        Long.parseLong(getSysParam(db.getSystemParams(), SystemParams.IP_RESERVE)));
    String method = getParam(params, HttpMethods.METHOD);
    log.info("Execute method: " + method);
    switch (method) {
        case HttpMethods.SEND_JOB:
            processAddJob(params, sh, db);
            break;
        case HttpMethods.PING:
            sh.process(Response.OK);
            break;
        case HttpMethods.VM_STATE:
            processVMState(sh, db);
            break;
        case HttpMethods.SEND_STATE:
            processVMStateUpdate(params, sh, db);
            break;
        default:
            throw new NoArgumentException("Method \"" + method + "\" not supported!");
    }
} catch (NoArgumentException ex) {
    log.error(ex);
    sh.process(Response.WRONG_METHOD);
} catch (NoFreeVMException ex) {
    log.error(ex);
    sh.process(Response.VM_NOT_FOUND);
} catch (SQLException ex) {
    log.error("DB error", ex);
    sh.process(Response.DB_ERROR);
} catch (Exception ex) {
    log.error("Server exception", ex);
    sh.process(Response.INTERNAL_ERROR);
} finally {
    log.info("Done, process time: " + (System.currentTimeMillis() - start) + " ms\n\n");
    response.getWriter().close();
    DbUtils.closeQuietly(conn);
}

}

/**
 * новая задача на обработку
 *
 * @param params
 * @param sh
 */
private void processAddJob(Map<String, String[]> params, ServerHandler sh, DBManager db) throws
NoArgumentException, SQLException, IOException, NoFreeVMException {
    String data = getParam(params, HttpMethods.DATA);
    log.info("New job: " + data);

    VirtualMachine vm = db.getFistFreeVM();
    if (vm != null) {
        //если есть свободна VM

```

```

        sh.process(sh.getProcessedData(HttpRequestHelper.sendingRequest(vm.getIp(), data)));
    } else {

        vm = getMinVM(db.getVMSnapshot());

        if (vm == null) {
            throw new NoFreeVMException("There are no running virtual machines");
        }

        String ip = db.getFreeIP();
        if (!"".equals(ip)) {
            //есть свободный ip
            HttpRequestHelper.sendingStartKillRequest(vm.getIp(), VMCommand.START + "!" + ip);
            log.info("VM ip: " + ip + " starting successfully");
            db.updateIPState(ip, true);
        }
        //нет свободного ip
        vm = db.getFistFreeVM();
        if (vm != null) {
            //если есть свободна VM
            db.resetVMfreeCount(vm.getId());
            sh.process(sh.getProcessedData(HttpRequestHelper.sendingRequest(vm.getIp(), data)));
        } else {
            vm = getMinVM(db.getVMSnapshot());

            if (vm == null) {
                throw new NoFreeVMException("There are no running virtual machines");
            }

            db.resetVMfreeCount(vm.getId());
            sh.process(sh.getProcessedData(HttpRequestHelper.sendingRequest(vm.getIp(), data)));
        }
    }
}

private VirtualMachine getMinVM(List<VirtualMachine> snapshot) {
    long minRam = Long.MAX_VALUE;
    int minCpu = Integer.MAX_VALUE;

    VirtualMachine minVM = null;

    for (VirtualMachine vm : snapshot) {
        if (vm.getCpu() <= minCpu && vm.getRam() <= minRam) {
            minVM = vm;
        }
    }

    return minVM;
}

/**
 * обновление сост VM
 *
 * @param params
 * @param sh
 * @param db
 * @throws NoArgumentException
 * @throws SQLException
 */
private void processVMStateUpdate(Map<String, String[]> params, ServerHandler sh, DBManager db) throws
NoArgumentException, SQLException {
    String vmStr = getParam(params, HttpMethods.VM);

```

```

log.info("New VM state: " + vmStr);
VirtualMachine vm = sh.getVirtualMachine(vmStr);

int freeCount = db.updateVmState(vm);
sh.process(Response.OK);

Map<String, String> sParams = db.getSystemParams();

int maxFreeCount = Integer.parseInt(getSysParam(sParams, SystemParams.FREE_COUNT));
int minVMCount = Integer.parseInt(getSysParam(sParams, SystemParams.MIN_VM_COUNT));

if (db.getVMSnapshot().size() > minVMCount) {
    if (freeCount > maxFreeCount) {
        log.info("Try to kill VM: " + vmStr);
        try {
            db.deleteVM(vm);
            HttpRequestHelper.sendingStartKillRequest(vm.getIp(), VMCommand.KILL);
            log.info("VM: " + vmStr + " killing successfully");
        } catch (IOException ex) {
            log.error("Kill VM error: " + ex, ex);
        }
    }
}

/**
 * запрос снимшота VM
 *
 * @param sh
 */
private void processVMState(ServerHandler sh, DBManager db) throws SQLException {
    log.info("Create vm state list");
    sh.process(db.getVMSnapshot());
}

private String getParam(Map<String, String[]> params, String name) throws NoArgumentException {
    String[] data = params.get(name);
    if (data == null || data.length != 1) {
        throw new NoArgumentException("Param \"" + name + "\" is not found");
    }
    String res = data[0];
    if (res == null || res.isEmpty()) {
        throw new NoArgumentException("Param \"" + name + "\" is empty");
    }
    return res;
}

private String getSysParam(Map<String, String> params, String name) throws NoArgumentException {
    String data = params.get(name);
    if (data == null) {
        throw new NoArgumentException("Param \"" + name + "\" is not found");
    }
    if (data.isEmpty()) {
        throw new NoArgumentException("Param \"" + name + "\" is empty");
    }
    return data;
}

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    log.info("Do Get");
}

```



```

        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        log.info("Do Post");
        processRequest(request, response);
    }

    @Override
    public String getServletInfo() {
        return "Server component";
    }
}

```

```
package server.db;
```

```

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import model.data.VirtualMachine;
import org.apache.commons.dbutils.DbUtils;
import org.apache.log4j.Logger;

/**
 *
 * @author user
 */
public class DBManager {

    private static final Logger log = Logger.getLogger("super_server");

    private Connection conn;

    public DBManager(Connection conn) {
        if (conn == null) {
            throw new NullPointerException("Connection can not be null.");
        }
        this.conn = conn;
    }

    /**
     * получить состояния виртуальных машин
     *
     * @return
     * @throws SQLException
     */
    public List<VirtualMachine> getVMSnapshot() throws SQLException {
        PreparedStatement stmt = null;
        ResultSet rs = null;
    }
}

```

```

try {
    stmt = conn.prepareStatement(
        "SELECT                                \n"
        + "mv.id_vmachine id,                  \n"
        + "mv.cpu cpu,                          \n"
        + "mv.ram ram,                          \n"
        + "ip.ip ip,                            \n"
        + "ip.port port                        \n"
        + " FROM \"VMserver\".ip_pool ip          \n"
        + " JOIN \"VMserver\".vmachine mv using (id_ip)"
    );

    List<VirtualMachine> lvm = new ArrayList<>();

    rs = stmt.executeQuery();
    while (rs.next()) {
        VirtualMachine vm = new VirtualMachine(
            rs.getInt("id"),
            rs.getLong("ram"),
            rs.getInt("cpu"),
            rs.getString("ip"),
            rs.getInt("port"));

        lvm.add(vm);
    }
    return lvm;
} finally {
    DbUtils.closeQuietly(rs);
    DbUtils.closeQuietly(stmt);
}
}

/**
 * очистить БД от умерших машин
 *
 * @param maxVM
 * @param maxIP
 * @throws SQLException
 */
public void clearDB(long maxVM, long maxIP) throws SQLException {

    log.info("Clear DB ...");
    PreparedStatement stmt = null;

    try {
        stmt = conn.prepareStatement(
            "DELETE FROM                        \n"
            + "\"VMserver\".vmachine vm          \n"
            + " WHERE                            \n"
            + "   createCondition(\"vm.last_connect\", maxVM)
        );

        log.info(stmt.executeUpdate() + " VM was removed");

        freeIPs(maxIP);

        log.info("... DB Clear OK");
    } finally {
        DbUtils.closeQuietly(stmt);
    }
}
}

```

```

private static String createCondition(String row, long max){
    DateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.000");
    StringBuilder sb = new StringBuilder();
    sb.append(row).append(" < ").append(df.format(new Date().getTime() - max)).append("");
    log.info(sb);
    return sb.toString();
}

```

```

public VirtualMachine getFistFreeVM() throws SQLException{
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try {
        pstmt = conn.prepareStatement(
            "SELECT                                \n"
            + "mv.id_vmachine id,                  \n"
            + "mv.cpu cpu,                          \n"
            + "mv.ram ram,                          \n"
            + "ip.ip ip,                            \n"
            + "ip.port port                        \n"
            + " FROM \"VMserver\".ip_pool ip          \n"
            + " JOIN \"VMserver\".vmachine mv using (id_ip) \n"
            + " WHERE mv.free_count>0"
        );

        rs = pstmt.executeQuery();
        if (rs.next()) {
            log.info("VM with free_count>0 id = " + rs.getInt("id") + " was found");
            return new VirtualMachine(
                rs.getInt("id"),
                rs.getLong("ram"),
                rs.getInt("cpu"),
                rs.getString("ip"),
                rs.getInt("port"));
        } else {
            log.info("VM with free_count>0 not found");
            return null;
        }
    } finally {
        DbUtils.closeQuietly(rs);
        DbUtils.closeQuietly(pstmt);
    }
}

```

```

/**
 * регистрация нового ip в БД и возвращение его id в базе
 *
 * @param ip
 * @param port
 * @return
 * @throws SQLException
 */
public int registerNewIP(String ip, int port) throws SQLException {
    log.info("Register new ip = " + ip + ":" + port);
    PreparedStatement stmt = null;
    ResultSet rs = null;
    try {
        stmt = conn.prepareStatement(
            "insert into "
            + "\"VMserver\".ip_pool("
            + " ip,"
            + " port) "
            + "values "

```

```

        + "(?,?)"
        + " returning id_ip");

stmt.setString(1, ip);
stmt.setInt(2, port);

stmt.execute();

rs = stmt.getResultSet();
rs.next();
return rs.getInt("id_ip");
} finally {
    DbUtils.closeQuietly(rs);
    DbUtils.closeQuietly(stmt);
}
}

/**
 * освобождение ip по max времени
 *
 * @param max
 * @throws SQLException
 */
private void freeIPs(long max) throws SQLException {
    log.info("Free IP ...");
    PreparedStatement stmt = null;
    try {
        stmt = conn.prepareStatement(
            "UPDATE "
            + "\"VMserver\".ip_pool ip\n"
            + " SET "
            + "reserve=FALSE, "
            + "last_reserve=now()\n"
            + " WHERE "
            + "ip.reserve=TRUE and "
            + "ip.id_ip not in (SELECT id_ip FROM \"VMserver\".vmachine vm) "
            + "and " + createCondition("ip.last_reserve", max));

        log.info(stmt.executeUpdate() + " IP was free");

        log.info("... IP free OK");

    } finally {
        DbUtils.closeQuietly(stmt);
    }
}

/**
 * зарегистрировать новую VM
 * @param vm
 * @param id_ip
 */
private void registerNewVM(VirtualMachine vm, int id_ip) throws SQLException {
    log.info("Register new vm = " + vm + ", with id_ip = " + id_ip);
    PreparedStatement stmt = null;
    try {
        stmt = conn.prepareStatement(
            "INSERT INTO          \n"
            + "\"VMserver\".vmachine( \n"
            + "    cpu,          \n"
            + "    ram,           \n"
            + "    free_count,    \n"

```

```

        + " id_ip,          \n"
        + " last_connect)  \n"
+ " VALUES (?, ?, 1, ?, now())");

stmt.setInt(1, vm.getCpu());
stmt.setLong(2, vm.getRam());
stmt.setInt(3, id_ip);

stmt.execute();

updateIPState(id_ip, true);

} finally {
    DbUtils.closeQuietly(stmt);
}
}

/**
 * найти id_ip для ip:port в базе
 * @param ip
 * @param port
 * @return id_ip
 * @throws SQLException
 */
private int checkIP(String ip, int port) throws SQLException {
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try {
        pstmt = conn.prepareStatement(
            "SELECT          \n"
            + "id_ip          \n"
            + "FROM            \n"
            + "\"VMserver\".ip_pool \n"
            + "where              \n"
            + "ip=? and            \n"
            + "port=?              "
        );

        pstmt.setString(1, ip);
        pstmt.setInt(2, port);

        rs = pstmt.executeQuery();
        if (rs.next()) {
            log.info("Ip = " + ip + ":" + port + " was found");
            return rs.getInt("id_ip");
        } else {
            log.info("Ip = " + ip + ":" + port + " not found");
            return -1;
        }
    } finally {
        DbUtils.closeQuietly(rs);
        DbUtils.closeQuietly(pstmt);
    }
}

/**
 * получить первый свободный ip
 * @return
 * @throws SQLException
 */
public String getFreeIP() throws SQLException {
    PreparedStatement pstmt = null;

```

```

ResultSet rs = null;
try {
    pstmt = conn.prepareStatement(
        "SELECT          \n"
        + "id_ip,        \n"
        + "ip,            \n"
        + "port,          \n"
        + "last_reserve

        \n"
        + " FROM          \n"
        + "\"VMserver\".ip_pool ip \n"
        + " WHERE          \n"
        + "ip.reserve=FALSE"
    );

    rs = pstmt.executeQuery();
    if (rs.next()) {
        log.info("Free Ip = " + rs.getString("ip") + ":" + rs.getInt("port") + " was found");
        return rs.getString("ip");
    } else {
        log.info("Free Ip not found");
        return "";
    }
} finally {
    DbUtils.closeQuietly(rs);
    DbUtils.closeQuietly(pstmt);
}
}

/**
 * обновить состояние блокировки для ip
 * @param id_ip
 * @param newState
 * @throws SQLException
 */
private void updateIPState(int id_ip, boolean newState) throws SQLException {
    log.info("Update IP(" + id_ip + ") state = " + newState);
    PreparedStatement stmt = null;
    try {
        stmt = conn.prepareStatement(
            "UPDATE          \n"
            + "\"VMserver\".ip_pool \n"
            + " SET          \n"
            + " reserve=?,      \n"
            + " last_reserve=now() \n"
            + "WHERE          \n"
            + " id_ip=" + id_ip);

        stmt.setBoolean(1, newState);

        stmt.execute();
    } finally {
        DbUtils.closeQuietly(stmt);
    }
}

/**
 * обновить состояние блокировки для ip
 * @param ip
 * @param newState
 * @throws SQLException

```

```

*/
public void updateIPState(String ip, boolean newState) throws SQLException {
    log.info("Update IP(" + ip + ") state = " + newState);
    PreparedStatement stmt = null;
    try {
        stmt = conn.prepareStatement(
            "UPDATE          \n"
            + "\"VMserver\".ip_pool  \n"
            + " SET          \n"
            + " reserve=?,          \n"
            + " last_reserve=now()  \n"
            + " WHERE          \n"
            + " ip=?");

        stmt.setBoolean(1, newState);
        stmt.setString(2, ip);

        stmt.execute();
    } finally {
        DbUtils.closeQuietly(stmt);
    }
}

/**
 * найти VM для id_ip в базе
 * @param id_ip
 * @return id_vmachine
 * @throws SQLException
 */
private int checkVM(int id_ip) throws SQLException {
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try {
        pstmt = conn.prepareStatement(
            "SELECT          \n"
            + "id_vmachine          \n"
            + "FROM          \n"
            + "\"VMserver\".vmachine  \n"
            + "where          \n"
            + "id_ip=?          "
        );

        pstmt.setInt(1, id_ip);

        rs = pstmt.executeQuery();
        if (rs.next()) {
            log.info("VM with id_ip = " + id_ip + " was found");
            return rs.getInt("id_vmachine");
        } else {
            log.info("VM with id_ip = " + id_ip + " not found");
            return -1;
        }
    } finally {
        DbUtils.closeQuietly(rs);
        DbUtils.closeQuietly(pstmt);
    }
}

/**
 * обновить инфу о VM
 * @param vm
 * @param id_vm

```

```

    * @return
    */
private void updateVM(VirtualMachine vm, int id_vm, int freeCount, int id_ip) throws SQLException{

    log.info("Update vm = " + vm + ", with id_vmachine = " + id_vm + " and freeCount = " + freeCount);
    PreparedStatement stmt = null;
    try {
        stmt = conn.prepareStatement(
            "UPDATE \"VMserver\".vmachine \n"
            + "SET \n"
            + "cpu=?, \n"
            + "ram=?, \n"
            + "free_count=?, \n"
            + "last_connect=now() \n"
            + " WHERE id_vmachine=" + id_vm);

        stmt.setInt(1, vm.getCpu());
        stmt.setLong(2, vm.getRam());
        stmt.setInt(3, freeCount);

        stmt.execute();

        updateIPState(id_ip, true);
    } finally {
        DbUtils.closeQuietly(stmt);
    }
}

/**
 * обновить инфу о VM
 * @param vm
 * @param id_vm
 * @return
 */
public void resetVMfreeCount(int id_vm) throws SQLException{
    log.info("Update vm with id_vmachine = " + id_vm + " and reset freeCount");
    PreparedStatement stmt = null;
    try {
        stmt = conn.prepareStatement(
            "UPDATE \"VMserver\".vmachine \n"
            + "SET \n"
            + "free_count=0 \n"
            + " WHERE id_vmachine=" + id_vm);

        stmt.execute();
    } finally {
        DbUtils.closeQuietly(stmt);
    }
}

private int getFreeCount(int id_vm) throws SQLException{
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try {
        pstmt = conn.prepareStatement(
            "SELECT \n"
            + "free_count \n"
            + "FROM \n"
            + "\"VMserver\".vmachine \n"
            + "where \n"
            + "id_vmachine=? "
        );
    }

```



```

pstmt.setInt(1, id_vm);

rs = pstmt.executeQuery();
if (rs.next()) {
    log.info("VM with id_vmachine(free_count) = " + id_vm + " was found");
    return rs.getInt("free_count");
} else {
    log.info("VM with id_vmachine(free_count) = " + id_vm + " not found");
    return -1;
}
} finally {
    DbUtils.closeQuietly(rs);
    DbUtils.closeQuietly(pstmt);
}
}

```

```

/**
 * обновление сосотояния VM
 * 1-проверить ip в базе
 * 2-если нет - добавить ip и машину
 * 3-если есть - вытащить id_ip
 * 4-если нет машины с таким id_ip - добавить (новая)
 * 5-если есть - провеить загруженность
 * 6-если загружена - обновить
 * 7-если нет - вернуть free_count
 *
 * @param vm
 * @return free_count
 */
public int updateVmState(VirtualMachine vm) throws SQLException {

    //(1)
    int id_ip = checkIP(vm.getIp(), vm.getPort());

    if (id_ip > 0) {
        //(3)
        int id_vm = checkVM(id_ip);
        if (id_vm > 0) {
            //(5)
            if (vm.isFree()) {
                //(7)
                int freeCount = getFreeCount(id_vm);
                updateVM(vm, id_vm, freeCount + 1, id_ip);
                return freeCount + 1;
            } else {
                //(6)
                updateVM(vm, id_vm, 0, id_ip);
                return 0;
            }
        } else {
            //(4)
            registerNewVM(vm, id_ip);
            return 0;
        }
    } else {
        //(2)
        id_ip = registerNewIP(vm.getIp(), vm.getPort());
        registerNewVM(vm, id_ip);
        return 0;
    }
}

```

```

    }

    public void deleteVM(VirtualMachine vm) throws SQLException{
        log.info("Delete vm = " + vm);
        int id_ip = checkIP(vm.getIp(), vm.getPort());

        PreparedStatement stmt = null;
        try {
            stmt = conn.prepareStatement(
                "DELETE FROM \"VMserver\".vmachine WHERE id_ip=" + id_ip
            );
            stmt.execute();

            updateIPState(id_ip, false);
        } finally {
            DbUtils.closeQuietly(stmt);
        }
    }

    public Map<String, String> getSystemParams() throws SQLException {
        Map<String, String> sParams = new HashMap<>();

        PreparedStatement pstmt = null;
        ResultSet rs = null;
        try {
            pstmt = conn.prepareStatement(
                "SELECT key, value FROM \"VMserver\".system_params;"
            );

            rs = pstmt.executeQuery();

            while (rs.next()) {
                sParams.put(rs.getString("key"), rs.getString("value"));
            }
        } finally {
            DbUtils.closeQuietly(rs);
            DbUtils.closeQuietly(pstmt);
        }

        return sParams;
    }
}

```

```

package server.web;

```

```

import com.google.gson.Gson;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;
import model.data.Data;
import model.data.HttpMethods;
import model.data.VMCommand;
import model.data.VirtualMachine;
import model.io.JsonUtils;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;

```

```

import org.apache.http.client.utils.URLEncodedUtils;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.params.BasicHttpParams;
import org.apache.http.params.HttpConnectionParams;
import org.apache.http.params.HttpParams;
import org.apache.log4j.Logger;

/**
 *
 * @author user
 */
public class HttpRequestHelper {

    private static final Logger log = Logger.getLogger("super_server");

    public static String sendingStartKillRequest(String url, String command) throws IOException {
        String sendingUrl = "http://" + url;

        HttpGet request = new HttpGet(sendingUrl);
        request.addHeader(VMCommand.COMMAND, command);

        log.info("Sending 'GET' request (command = " + command + ") to URL : " + sendingUrl);

        return send(request);
    }

    public static String sendingRequest(String url, String data) throws IOException {

        List<NameValuePair> params = new ArrayList<>();
        params.add(new BasicNameValuePair(HttpMethods.DATA, data));

        String sendingUrl = "http://" + url;

        log.info("Sending 'GET' request to URL : " + sendingUrl);
        log.info("GET parameters : " + URLEncodedUtils.format(params, "utf-8"));
        log.info("Request : " + sendingUrl + "?" + URLEncodedUtils.format(params, "utf-8"));

        HttpGet request = new HttpGet(sendingUrl + "?" + URLEncodedUtils.format(params, "utf-8"));
        request.setHeader("charset", "UTF-8");
        request.setHeader("Content-Type", "application/json");

        return send(request);
    }

    private static String send(HttpGet request) throws IOException {
        HttpClient client = getHttpClient();
        HttpResponse response = client.execute(request);

        log.info("Response Code : "
            + response.getStatusLine().getStatusCode());
        BufferedReader rd = new BufferedReader(
            new InputStreamReader(response.getEntity().getContent()));
        StringBuilder result = new StringBuilder();
        String line;

        // !!!!!без head'еров в ответе от VM повисает
        while ((line = rd.readLine()) != null) {
            log.info("line: " + line);
            result.append(line);
        }
    }
}

```

```

        log.info("Response: " + result.toString());

        return result.toString();
    }

    private static DefaultHttpClient getHttpClient() {
        HttpParams httpParameters = new BasicHttpParams();
        int timeoutConnection = 3000;
        HttpConnectionParams.setConnectionTimeout(httpParameters, timeoutConnection);
        int timeoutSocket = 5000;
        HttpConnectionParams.setSocketTimeout(httpParameters, timeoutSocket);

        return new DefaultHttpClient(httpParameters);
    }
}

```

### **В.3 Контроллер виртуальных машин**

```

package controller;

import controller.threads.MainThread;
import java.util.Scanner;
import model.utils.Logger;

public class Controller {

    private static final String TAG = "Controller";

    public static void main(String[] args) {
        Logger.i(TAG, "Start");

        Logger.i(TAG, "Enter sudo password:");
        String sudo = new Scanner(System.in).nextLine();

        Logger.i(TAG, "Enter server IP:");
        String sIp = new Scanner(System.in).nextLine();

        Logger.i(TAG, "Enter server port:");
        String sPort = new Scanner(System.in).nextLine();

        Logger.i(TAG, "Enter ip first vm:");
        String ip = new Scanner(System.in).nextLine();

        new Thread(new MainThread(sudo, sIp, sPort, ip)).start();
    }
}

```

```

package controller.tasks;

import model.data.VirtualMachine;
import model.web.ClientHttp;

public class SendVMStateTask implements Runnable{

    private VirtualMachine vm;
    private ClientHttp.ConnectListener listener;
    private ClientHttp client;
}

```

```

    public SendVMStateTask(ClientHttp.ConnectListener listener, ClientHttp client, VirtualMachine vm) {
        this.listener = listener;
        this.client = client;
        this.vm = vm;
    }

    @Override
    public void run() {
        listener.onStart();
        try {
            listener.onResult(client.sendVMState(vm));
        } catch (Exception ex) {
            listener.onError(ex);
        }
    }
}

```

```

package controller.threads;

```

```

import controller.tasks.SendVMStateTask;
import controller.threads.VMThread.VMThreadListener;
import java.io.IOException;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingDeque;
import java.util.concurrent.TimeUnit;
import java.util.logging.Level;
import model.data.Response;
import model.data.VMCommand;
import model.data.VirtualMachine;
import model.utils.Logger;
import model.web.ClientHttp;

```

```

/**

```

```

 * главный поток контроллера

```

```

 *

```

```

 * @author user

```

```

 */

```

```

public class MainThread implements Runnable {

```

```

    private static String TAG = "MainThread";

```

```

    private String sudo;

```

```

    private BlockingQueue<String> vmQueue; //очередь машин на запуск, хранит ip

```

```

    private String serverIp;

```

```

    private String serverPort;

```

```

    public MainThread(String sudo, String serverIp, String serverPort, String firstIp) {

```

```

        this.sudo = sudo;

```

```

        this.vmQueue = new LinkedBlockingDeque<>();

```

```

        vmQueue.add(firstIp); //первая дефолтная машина

```

```

        this.serverIp = serverIp;

```

```

        this.serverPort = serverPort;

```

```

    }

```

```

    @Override

```

```

    public void run() {

```

```

        Logger.i(TAG, "Start");

```

```

        String[] cmd = {"bin/bash", "-c", "echo " + sudo + "| sudo -S ./test.sh"};

```

```

        String ip;

```

```

while (true) {
    try {

        ip = vmQueue.poll(5, TimeUnit.SECONDS);

        if (ip != null) {
            if (!ip.equals("")) {
                Logger.i(TAG, "Try start new vm with ip: " + ip);
                new Thread(
                    new VMThread(
                        new LocalVMThreadListener(ip, serverIp, serverPort),
                        new String[]{cmd[0], cmd[1], cmd[2] + " " + ip}
                    )
                ).start();
            }
        }

    } catch (Exception e) {
        Logger.e(TAG, e.getMessage(), e);
    }
}

private class LocalVMThreadListener implements VMThreadListener {

    private String TAG = "LocalVMThreadListener";
    private final String PID = "Main pid";
    private final String PING = "Go ping server";
    private int pid = -1;
    private ClientHttp client = null;
    private String ip;

    public LocalVMThreadListener(String ip, String sIp, String sPort) {
        client = new ClientHttp("http://" + sIp + ":" + sPort + "/Server/ServerServlet");
        this.ip = ip;
    }

    @Override
    public void onStart() {
        Logger.i(TAG, "Start");
    }

    @Override
    public void setData(String data) {
        if (data.equals(VMCommand.COMMAND + ": " + VMCommand.KILL)) {
            try {
                Logger.i(TAG, "Stop the servise");
                Thread.sleep(1000);
                if (killProces(pid) == 0) {
                    Logger.i(TAG, "VM kill success");
                } else {
                    Logger.i(TAG, "Error on VM kill");
                }
            } catch (InterruptedException ex) {
                java.util.logging.Logger.getLogger(MainThread.class.getName()).log(Level.SEVERE, null, ex);
            }
        } else if (data.contains(VMCommand.COMMAND + ": " + VMCommand.START)) {
            String[] cmd = data.split("!");
            Logger.i(TAG, "Start new servise:" + cmd[1]);
            vmQueue.add(cmd[1]);
        } else if (data.contains(PID)) {
            //костыль, но работает идеально

```

```

        String[] cmd = data.split("=");
        pid = Integer.parseInt(cmd[1].replace(" ", ""));
        Logger.i(TAG, "Find new pid:" + pid);
    } else if (data.contains(PING)) {
        String[] state = data.split(":");
        VirtualMachine vm = new VirtualMachine(Long.parseLong(state[2]), Integer.parseInt(state[1]), ip);
        new Thread(new SendVMStateTask(new LocalStateListener(), client, vm)).start();
    }
}

@Override
public void onDestroy() {
    Logger.i(TAG, "Destroy");
}

private int killProces(int pid) {
    String[] cmd = {"/bin/bash", "-c", "echo " + sudo + "| sudo -S kill " + pid};
    try {
        Process p = new ProcessBuilder(cmd).start();
        return p.waitFor();
    } catch (IOException | InterruptedException e) {
        Logger.e(TAG, e.getMessage(), e);
        return -1;
    }
}

private class LocalStateListener implements ClientHttp.ConnectListener {

    @Override
    public void onStart() {
        Logger.i(TAG, "Send state");
    }

    @Override
    public void onResult(Response response) {
        Logger.i(TAG, "Send state error code:" + response.getResultCode());
    }

    @Override
    public void onError(Exception ex) {
        Logger.e(TAG, ex.getMessage());
    }
}
}
}

```

```
package controller.threads;
```

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import model.utils.Logger;
```

```
/**
 * поток работы виртуальной машины
 *
 * @author user
 */
public class VMThread implements Runnable {

    /**

```

```

    * слушатель работы виртуальной машины
    */
    public interface VMThreadListener {

        /**
         * Событие на запуск
         */
        void onStart();

        /**
         * главный обработчик: получает сообщения от vm и обрабатывает их
         *
         * @param data
         * @param p
         */
        void setData(String data);

        /**
         * событие на остановку
         */
        void onDestroy();
    }

    private final VMThreadListener listener;
    private static final String TAG = "VMThread";
    private final String[] cmd;

    public VMThread(VMThreadListener listener, String[] cmd) {
        this.listener = listener;
        this.cmd = cmd;
    }

    @Override
    public void run() {

        Logger.i(TAG, "Start");
        listener.onStart();

        try {
            Process p = new ProcessBuilder(cmd).start();

            BufferedReader reader = new BufferedReader(new InputStreamReader(p.getInputStream()));

            String line;
            while ((line = reader.readLine()) != null) {
                Logger.i(TAG, line);
                listener.setData(line);
            }

        } catch (Exception e) {
            Logger.e(TAG, e.getMessage(), e);
        }

        Logger.i(TAG, "Finish");
        listener.onDestroy();
    }
}

```



## В.4 Клиентский компонент системы

```
package client.gui;

import client.tasks.JobTask;
import client.tasks.PingTask;
import client.utils.ConfigEditor;
import client.utils.FileHelper;
import java.awt.Component;
import java.awt.Container;
import java.awt.Point;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import model.data.Data;
import model.data.Job;
import model.data.Response;
import model.utils.Logger;
import model.web.ClientHttp;
import model.web.ClientHttp.ConnectListener;

/**
 *
 * @author user
 */
public class MainForm extends javax.swing.JFrame {

    private String url = "";
    private final String TAG = "Client";
    private final String CONFIG_NAME = "connect.cfg";
    private String ip, port;
    private final String NEW = "новая";
    private final String ERROR = "ошибка обработки";
    private final String OK = "выполнена";
    private ClientHttp connect = null;
    private List<Job<Data, Data>> jobs = null;

    /**
     * Creates new form MainForm
     */
    public MainForm() {
        initComponents();
        Map<String, String> map = new HashMap();

        map = ConfigEditor.getConfig(CONFIG_NAME);
        if (map != null) {
            ip = map.get(ConfigEditor.IP);
            port = map.get(ConfigEditor.PORT);
            url = "http://" + ip + ":" + port + "/Server/ServerServlet";
            connect = new ClientHttp(url);
            Logger.i(TAG, "Cfg url = " + url);
            jobs = new ArrayList<>();
            fillTableJob();
        }
    }
}
```

```

        jobTable.addMouseListener(new LocalMouseAdapter());
    } else {
        enableComponents(PanelContent, false);
    }
    enableComponents(jPanel1, false);
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code"> //GEN-BEGIN: initComponents
private void initComponents() {

    connectDialog = new javax.swing.JDialog();
    TestConnect = new javax.swing.JButton();
    jLabel1 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    IpField = new javax.swing.JTextField();
    PortField = new javax.swing.JTextField();
    jPanel1 = new javax.swing.JPanel();
    CfgOk = new javax.swing.JButton();
    CfgCancel = new javax.swing.JButton();
    PingProgress = new javax.swing.JProgressBar();
    resultText = new javax.swing.JLabel();
    addJobDialog = new javax.swing.JDialog();
    addJobPanel = new javax.swing.JPanel();
    jLabel3 = new javax.swing.JLabel();
    jobNameField = new javax.swing.JTextField();
    jLabel4 = new javax.swing.JLabel();
    jobStatusLabel = new javax.swing.JLabel();
    jLabel6 = new javax.swing.JLabel();
    loadDataButton = new javax.swing.JButton();
    jScrollPane2 = new javax.swing.JScrollPane();
    jobText = new javax.swing.JTextArea();
    jLabel7 = new javax.swing.JLabel();
    jScrollPane3 = new javax.swing.JScrollPane();
    processedText = new javax.swing.JTextArea();
    addJobButton = new javax.swing.JButton();
    closeJobButton = new javax.swing.JButton();
    addJobProgress = new javax.swing.JProgressBar();
    saveJobButton = new javax.swing.JButton();
    PanelContent = new javax.swing.JPanel();
    jScrollPane1 = new javax.swing.JScrollPane();
    jobTable = new javax.swing.JTable();
    jButton1 = new javax.swing.JButton();
    jMenuBar1 = new javax.swing.JMenuBar();
    jMenu1 = new javax.swing.JMenu();
    jMenu2 = new javax.swing.JMenu();
    jMenuItem1 = new javax.swing.JMenuItem();

    connectDialog.setAlwaysOnTop(true);
    connectDialog.setMinimumSize(new java.awt.Dimension(265, 140));

    TestConnect.setText("Test");
    TestConnect.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            TestConnectActionPerformed(evt);
        }
    });
}

```



```

        .addComponent(jLabel1))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(connectDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
        .addComponent(IpField)
        .addComponent(PortField, javax.swing.GroupLayout.DEFAULT_SIZE, 186,
Short.MAX_VALUE))))))
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
connectDialogLayout.createSequentialGroup())
        .addComponent(PingProgress, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        .addContainerGap()
        .addGroup(connectDialogLayout.createSequentialGroup())
        .addComponent(resultText)
        .addGap(0, 0, Short.MAX_VALUE))))
);
connectDialogLayout.setVerticalGroup(
connectDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
.addGroup(connectDialogLayout.createSequentialGroup()
.addContainerGap()
.addGroup(connectDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
.addComponent(jLabel1)
.addComponent(IpField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addGroup(connectDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
.addComponent(jLabel2)
.addComponent(PortField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addGroup(connectDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)
.addComponent(TestConnect, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(resultText)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 27, Short.MAX_VALUE)
.addComponent(PingProgress, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
.addContainerGap())
);

addJobDialog.setAlwaysOnTop(true);
addJobDialog.setMinimumSize(new java.awt.Dimension(560, 550));

jLabel3.setText("Название задачи");

jLabel4.setText("Статус задачи.");

jobStateLabel.setText("новая");

jLabel6.setText("Данные для обработки");

loadDataButton.setText("Загрузить");
loadDataButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        loadDataButtonActionPerformed(evt);
    }
}

```

```

});

jobText.setColumns(20);
jobText.setRows(5);
jScrollPane2.setViewportViewView(jobText);

jLabel7.setText("Обработанные данные");

processedText.setColumns(20);
processedText.setRows(5);
processedText.setEnabled(false);
jScrollPane3.setViewportViewView(processedText);

addJobButton.setText("Отправить");
addJobButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        addJobButtonActionPerformed(evt);
    }
});

closeJobButton.setText("Закрыть");
closeJobButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        closeJobButtonActionPerformed(evt);
    }
});

saveJobButton.setText("Сохранить в файл");
saveJobButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        saveJobButtonActionPerformed(evt);
    }
});

javax.swing.GroupLayout addJobPanelLayout = new javax.swing.GroupLayout(addJobPanel);
addJobPanel.setLayout(addJobPanelLayout);
addJobPanelLayout.setHorizontalGroup(
    addJobPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(addJobPanelLayout.createSequentialGroup()
            .addGap(10, 10, 10)
            .addGroup(addJobPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jScrollPane3, javax.swing.GroupLayout.Alignment.TRAILING)
                .addComponent(jScrollPane2, javax.swing.GroupLayout.Alignment.TRAILING)
                .addGroup(addJobPanelLayout.createSequentialGroup()
                    .addComponent(jLabel6)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(loadDataButton))
                .addGroup(addJobPanelLayout.createSequentialGroup()
                    .addComponent(jLabel7)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(saveJobButton))
                .addGroup(addJobPanelLayout.createSequentialGroup()
                    .addComponent(addJobButton)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                    .addComponent(closeJobButton))
                .addComponent(addJobProgress, javax.swing.GroupLayout.PREFERRED_SIZE, 532,
                    javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGroup(addJobPanelLayout.createSequentialGroup()
                    .addComponent(jLabel3)

```

```

        .addComponent(jLabel4))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

    .addGroup(addJobPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jobStatusLabel)
        .addComponent(jobNameField))))
        .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
    );
    addJobPanelLayout.setVerticalGroup(
        addJobPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, addJobPanelLayout.createSequentialGroup())
        .addContainerGap(22, Short.MAX_VALUE)
        .addGroup(addJobPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel3)
            .addComponent(jobNameField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
        .addGroup(addJobPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel4)
            .addComponent(jobStatusLabel))
        .addGap(18, 18, 18)
        .addGroup(addJobPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel6)
            .addComponent(loadDataButton))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
        .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED_SIZE, 135,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addGroup(addJobPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel7)
            .addComponent(saveJobButton))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
        .addComponent(jScrollPane3, javax.swing.GroupLayout.PREFERRED_SIZE, 148,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
        .addGroup(addJobPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(addJobButton)
            .addComponent(closeJobButton))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
        .addComponent(addJobProgress, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addContainerGap())
    );

    javax.swing.GroupLayout addJobDialogLayout = new
    javax.swing.GroupLayout(addJobDialog.getContentPane());
    addJobDialog.getContentPane().setLayout(addJobDialogLayout);
    addJobDialogLayout.setHorizontalGroup(
        addJobDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(addJobDialogLayout.createSequentialGroup())
        .addComponent(addJobPanel, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(0, 4, Short.MAX_VALUE))
    );
    addJobDialogLayout.setVerticalGroup(
        addJobDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(addJobPanel, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
    );

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

```

```

jobTable.setModel(new javax.swing.table.DefaultTableModel(
    new Object [][] {

        },
        new String [] {
            "Название задачи", "Состояние"
        }
    ) {
        boolean[] canEdit = new boolean [] {
            false, false
        };

        public boolean isCellEditable(int rowIndex, int columnIndex) {
            return canEdit [columnIndex];
        }
    });
jobTable.setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
jScrollPane1.setViewportView(jobTable);

jButton1.setLabel("Добавить задание");
jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton1ActionPerformed(evt);
    }
});

javax.swing.GroupLayout PanelContentLayout = new javax.swing.GroupLayout(PanelContent);
PanelContent.setLayout(PanelContentLayout);
PanelContentLayout.setHorizontalGroup(
    PanelContentLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(PanelContentLayout.createSequentialGroup()
            .addGap(10, 10, 10)
            .addComponent(jButton1)
            .addGap(10, 10, 10)
            .addComponent(jScrollPane1, javax.swing.GroupLayout.Alignment.TRAILING,
                javax.swing.GroupLayout.DEFAULT_SIZE, 774, Short.MAX_VALUE)
            .addContainerGap())
        .addGroup(PanelContentLayout.createSequentialGroup()
            .addGap(10, 10, 10)
            .addComponent(jButton1)
            .addGap(10, 10, 10)
            .addComponent(jScrollPane1)
            .addContainerGap())
);

jButton1.getAccessibleContext().setAccessibleName("AddButton");

jMenu1.setText("File");
jMenuBar1.add(jMenu1);

jMenu2.setText("Настройка");

jMenuItem1.setText("Настройка подключения");
jMenuItem1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jMenuItem1ActionPerformed(evt);
    }
});
jMenu2.add(jMenuItem1);

```

```

jMenuBar1.add(jMenu2);

setJMenuBar(jMenuBar1);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(0, 0, Short.MAX_VALUE))
        );
layout.setVerticalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGap(0, 12, Short.MAX_VALUE)
            .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
        );

pack();
} // </editor-fold> // GEN-END: initComponents

//<editor-fold defaultstate="collapsed" desc="Настройка подключения">
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_jButton1ActionPerformed
    // TODO add your handling code here:
    loadAddJobDialog(null);
} //GEN-LAST:event_jButton1ActionPerformed

private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_jMenuItem1ActionPerformed
    connectDialog.setVisible(true);
    IpField.setText(ip);
    PortField.setText(port);
} //GEN-LAST:event_jMenuItem1ActionPerformed

private void TestConnectActionPerformed(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_TestConnectActionPerformed
    // Тест подключения
    ip = IpField.getText();
    port = PortField.getText();
    String localUrl = "http://" + ip + ":" + port + "/Server/ServerServlet";
    connect = new ClientHttp(localUrl);
    new Thread(new PingTask(new LocalPingListener(), connect)).start();
} //GEN-LAST:event_TestConnectActionPerformed

private void CfgOkActionPerformed(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_CfgOkActionPerformed
    Map<String, String> map = new HashMap();
    map.put(ConfigEditor.IP, ip);
    map.put(ConfigEditor.PORT, port);
    if (ConfigEditor.setConfig(map, CONFIG_NAME)) {
        connectDialog.setVisible(false);
        enableComponents(jPanel1, true);
    }
} //GEN-LAST:event_CfgOkActionPerformed

private void CfgCancelActionPerformed(java.awt.event.ActionEvent evt) { //GEN-

```



```

FIRST:event_CfgCancelActionPerformed
    connectDialog.setVisible(false);
} //GEN-LAST:event_CfgCancelActionPerformed
//</editor-fold>

    private void addJobButtonActionPerformed(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_addJobButtonActionPerformed
    // TODO add your handling code here:
    if (validateAddJob()) {
        if (connect != null) {
            Job<Data, Data> job = new Job<>(new Data(jobText.getText(), null, jobNameField.getText(),
Response.PROCESS_ERROR);
            new Thread(new JobTask(new LocalAddJobListener(), connect, job)).start();
        } else {
            showError("Не настроено соединение с сервером!");
        }
    }
} //GEN-LAST:event_addJobButtonActionPerformed

    private void closeJobButtonActionPerformed(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_closeJobButtonActionPerformed
    // TODO add your handling code here:
    addJobDialog.setVisible(false);
} //GEN-LAST:event_closeJobButtonActionPerformed

    private void saveJobButtonActionPerformed(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_saveJobButtonActionPerformed
    File file = selectFile("Выбрать");
    FileHelper.saveResultJob(new Data(processedText.getText(), file);
} //GEN-LAST:event_saveJobButtonActionPerformed

    private void loadDataButtonActionPerformed(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_loadDataButtonActionPerformed
    File file = selectFile("Открыть файл");
    try {
        jobText.setText(FileHelper.inputLoad(file).getData());
    } catch (IOException ex) {
        showError(ex.getMessage());
    }
} //GEN-LAST:event_loadDataButtonActionPerformed

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    //<editor-fold defaultstate="collapsed" desc="Look and feel setting code (optional) ">
    /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
     * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
     */
    try {
        for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
            if ("Nimbus".equals(info.getName())) {
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (ClassNotFoundException ex) {
        java.util.logging.Logger.getLogger(MainForm.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
    } catch (InstantiationException ex) {
        java.util.logging.Logger.getLogger(MainForm.class.getName()).log(java.util.logging.Level.SEVERE, null,

```

```

ex);
    } catch (IllegalAccessException ex) {
        java.util.logging.Logger.getLogger(MainForm.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
        java.util.logging.Logger.getLogger(MainForm.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
    }
}
//</editor-fold>

/* Create and display the form */
java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new MainForm().setVisible(true);
    }
});
}

private void enableComponents(Container container, boolean enable) {
    Component[] components = container.getComponents();
    for (Component component : components) {
        component.setEnabled(enable);
        if (component instanceof Container) {
            enableComponents((Container) component, enable);
        }
    }
}

private File selectFile(String okButtonText){
    JFileChooser fileopen = new JFileChooser();
    int ret = fileopen.showDialog(null, okButtonText);
    if (ret == JFileChooser.APPROVE_OPTION) {
        return fileopen.getSelectedFile();
    }
    return null;
}

private void loadAddJobDialog(Job<Data, Data> job) {
    Logger.i(TAG, "Dialog: " + job);
    if (job == null) {
        jobNameField.setText("");
        jobStateLabel.setText(NEW);
        jobText.setText("");
        processedText.setText("");
        processedText.setEnabled(false);
        addJobButton.setEnabled(true);

        addJobDialog.setVisible(true);
    } else {
        jobNameField.setText(job.getName());
        switch (job.getState()) {
            case Response.OK:
                jobStateLabel.setText(OK);
                processedText.setText(job.getOutputData().getData());
                break;
            default:
                jobStateLabel.setText(ERROR);
                break;
        }
        jobText.setText(job.getInputData().getData());
    }
}

```

```

        processedText.setEnabled(false);
        addJobButton.setEnabled(false);

        addJobDialog.setVisible(true);
    }
}

private boolean validateAddJob() {
    if (jobNameField.getText().equals("")) {
        showError("Необходимо указать название задания!");
        return false;
    }

    if (!isUniqueName(jobNameField.getText())) {
        showError("Название с таким именем уже существует!");
        return false;
    }

    if (jobText.getText().equals("")) {
        showError("Необходимо указать данные задания!");
        return false;
    }

    return true;
}

private boolean isUniqueName(String name) {
    if (jobs != null) {
        for (Job job : jobs) {
            if (job.getName().equals(name)) {
                return false;
            }
        }
    }
    return true;
}

private Job findJobByName(String name) {
    if (jobs != null) {
        for (Job job : jobs) {
            if (job.getName().equals(name)) {
                return job;
            }
        }
    }
    return null;
}

private void showError(String message) {
    JOptionPane.showMessageDialog(null, message);
}

private void fillTableJob() {
    if (jobs != null) {
        DefaultTableModel model = (DefaultTableModel) jobTable.getModel();
        while (model.getRowCount() > 0) {
            model.removeRow(0);
        }
        Object rowData[] = new Object[2];
        for (int i = 0; i < jobs.size(); i++) {
            rowData[0] = jobs.get(i).getName();
            rowData[1] = jobs.get(i).getState() == Response.OK ? OK : ERROR;
        }
    }
}

```

```

        model.addRow(rowData);
    }
}
}

```

```
private class LocalMouseAdapter extends MouseAdapter {
```

```

    @Override
    public void mouseClicked(MouseEvent e) {
        Point p = e.getPoint();
        int row = jobTable.rowAtPoint(p);
        int column = jobTable.columnAtPoint(p);
        DefaultTableModel model = (DefaultTableModel) jobTable.getModel();
        System.out.println("Click on " + column + " column, " + row + " row; " + model.getValueAt(row, column));
        String name = (String) model.getValueAt(row, 0);
        loadAddJobDialog(findJobByName(name));
    }
}

```

```

/**
 * Слушатель отправки задачи
 */

```

```
private class LocalAddJobListener implements ConnectListener {
```

```

    @Override
    public void onStart() {
        addJobProgress.setIndeterminate(true);
        enableComponents(addJobPanel, false);
    }

    @Override
    public void onResult(Response response) {
        addJobProgress.setIndeterminate(false);
        enableComponents(addJobPanel, true);
        Job<Data, Data> job = (Job<Data, Data>) response.getData();
        loadAddJobDialog(job);

        jobs.add(job);
        fillTableJob();
    }
}

```

```

    @Override
    public void onError(Exception ex) {
        addJobProgress.setIndeterminate(false);
        enableComponents(addJobPanel, true);
        Logger.e(TAG, ex + "", ex);
        showError(ex + "");
        processedText.setEnabled(false);
        addJobButton.setEnabled(false);

        fillTableJob();
    }
}

```

```

/**
 * Слушатель подключения
 */

```

```
private class LocalPingListener implements ConnectListener {
```

```

    @Override

```

```

    public void onStart() {
        PingProgress.setIndeterminate(true);
    }

    @Override
    public void onResult(Response response) {
        PingProgress.setIndeterminate(false);
        if (response.getResultCode() == Response.OK) {
            enableComponents(jPanel1, true);
            resultText.setText("Connect OK");
        } else {
            resultText.setText("Connect failed");
        }
    }

    @Override
    public void onError(Exception ex) {
        PingProgress.setIndeterminate(false);
        Logger.e(TAG, ex.getMessage());
        resultText.setText("Connect failed");
    }
}

// Variables declaration - do not modify//GEN-BEGIN:variables
private javax.swing.JButton CfgCancel;
private javax.swing.JButton CfgOk;
private javax.swing.JTextField IpField;
private javax.swing.JPanel PanelContent;
private javax.swing.JProgressBar PingProgress;
private javax.swing.JTextField PortField;
private javax.swing.JButton TestConnect;
private javax.swing.JButton addJobButton;
private javax.swing.JDialog addJobDialog;
private javax.swing.JPanel addJobPanel;
private javax.swing.JProgressBar addJobProgress;
private javax.swing.JButton closeJobButton;
private javax.swing.JDialog connectDialog;
private javax.swing.JButton jButton1;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JMenu jMenu1;
private javax.swing.JMenu jMenu2;
private javax.swing.JMenuBar jMenuBar1;
private javax.swing.JMenuItem jMenuItem1;
private javax.swing.JPanel jPanel1;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JTextField jobNameField;
private javax.swing.JLabel jobStateLabel;
private javax.swing.JTable jobTable;
private javax.swing.JTextArea jobText;
private javax.swing.JButton loadDataButton;
private javax.swing.JTextArea processedText;
private javax.swing.JLabel resultText;
private javax.swing.JButton saveJobButton;
// End of variables declaration//GEN-END:variables
}

```

```

package client.tasks;

import model.data.Job;
import model.web.ClientHttp;

/**
 *
 * @author user
 */
public class JobTask implements Runnable{

    private ClientHttp.ConnectListener listener;
    private ClientHttp client;
    private Job data;

    public JobTask(ClientHttp.ConnectListener listener, ClientHttp client, Job data) {
        this.listener = listener;
        this.client = client;
        this.data = data;
    }

    @Override
    public void run() {
        listener.onStart();
        try {
            listener.onResult(client.addJob(data));
        } catch (Exception ex) {
            listener.onError(ex);
        }
    }
}

```

```

package client.tasks;

import model.web.ClientHttp;
import model.web.ClientHttp.ConnectListener;

/**
 *
 * @author user
 */
public class PingTask implements Runnable{

    private ConnectListener listener;
    private ClientHttp client;

    public PingTask(ConnectListener listener, ClientHttp client) {
        this.listener = listener;
        this.client = client;
    }

    @Override
    public void run() {
        listener.onStart();
        try {
            listener.onResult(client.ping());
        } catch (Exception ex) {
            listener.onError(ex);
        }
    }
}

```

```
}
```

```
package client.utils;
```

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import model.utils.Logger;
```

```
/**
```

```
 *
```

```
 * @author user
```

```
 */
```

```
public class ConfigEditor {
```

```
    private static final String TAG = "ConfigEditor";
```

```
    public static final String IP = "ip";
```

```
    public static final String PORT = "port";
```

```
    public static Map<String, String> getConfig(String fileName) {
```

```
        String ip, port;
```

```
        Map<String, String> map = new HashMap();
```

```
        try (FileReader file = new FileReader(fileName)) {
```

```
            BufferedReader reader = new BufferedReader(file);
```

```
            ip = reader.readLine();
```

```
            port = reader.readLine();
```

```
            map.put(IP, ip);
```

```
            map.put(PORT, port);
```

```
            file.close();
```

```
            Logger.i(TAG, "Config load success!");
```

```
        } catch (IOException ex) {
```

```
            Logger.e(TAG, ex.toString());
```

```
            return null;
```

```
        }
```

```
    }

    return map;
```

```
}
```

```
    public static boolean setConfig(Map<String, String> args, String fileName) {
```

```
        try (FileWriter writer = new FileWriter(fileName, false)) {
```

```
            writer.write(args.get(IP) + "\n");
```

```
            writer.write(args.get(PORT));
```

```
            writer.flush();
```

```
            writer.close();
```

```
            Logger.i(TAG, "Config update success!");
```

```
            return true;
```

```
        } catch (IOException ex) {
```

```
            Logger.e(TAG, ex.toString());
```

```
            return false;
```

```
        }
```

```
    }
```

```
}
```

```

package client.utils;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import model.data.Data;
import model.data.Job;
import model.utils.Logger;

/**
 *
 * @author user
 */
public class FileHelper {

    private static final String TAG = "FileHelper";

    public static void saveResultJob(Data data, File file) {

        try (FileWriter writer = new FileWriter(file)) {
            if(data != null){
                writer.write(data.getData() + "\n");
            } else {
                writer.write("null\n");
            }
            writer.flush();
            writer.close();
            Logger.i(TAG, "File \"" + file.getAbsolutePath() + file.getName() + "\" save success!");
        } catch (IOException ex) {
            Logger.e(TAG, ex.toString());
        }
    }

    public static Data inputLoad(File file) throws IOException {
        try (FileReader fileR = new FileReader(file)) {
            Data data = new Data(fileR);
            Logger.i(TAG, file.getAbsolutePath() + " load success!");
            return data;
        } catch (IOException ex) {
            Logger.e(TAG, ex.toString());
            return null;
        }
    }
}

```



## V.5 Консоль администратора

```
package admin.gui;

import admin.tasks.PingTask;
import admin.tasks.VMStateTask;
import admin.utils.ConfigEditor;
import java.awt.Component;
import java.awt.Container;
import java.awt.Point;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import model.data.Response;
import model.data.VirtualMachine;
import model.utils.Logger;
import model.web.ClientHttp;
import model.web.ClientHttp.ConnectListener;

/**
 *
 * @author user
 */
public class Admin extends javax.swing.JFrame {

    private String url = "";
    private final String TAG = "Client";
    private final String CONFIG_NAME = "connect.cfg";
    private String ip, port;
    private ClientHttp connect = null;
    private List<VirtualMachine> vm = null;

    /**
     * Creates new form Admin
     */
    public Admin() {
        initComponents();
        Map<String, String> map = new HashMap();

        map = ConfigEditor.getConfig(CONFIG_NAME);
        if (map != null) {
            ip = map.get(ConfigEditor.IP);
            port = map.get(ConfigEditor.PORT);
            url = "http://" + ip + ":" + port + "/Server/ServerServlet";
            connect = new ClientHttp(url);
            Logger.i(TAG, "Cfg url = " + url);

            vmTable.addMouseListener(new LocalMouseListener());
            new Thread(new VMStateTask(new LocalVMStateLoadListener(), connect)).start();
        } else {
            enableComponents(adminPanel, false);
        }
        enableComponents(jPanel1, false);
    }

    /**
```

```

* This method is called from within the constructor to initialize the form.
* WARNING: Do NOT modify this code. The content of this method is always
* regenerated by the Form Editor.
*/
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code"> //GEN-BEGIN: initComponents
private void initComponents() {

    connectDialog = new javax.swing.JDialog();
    TestConnect = new javax.swing.JButton();
    jLabel1 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    IpField = new javax.swing.JTextField();
    PortField = new javax.swing.JTextField();
    jPanel1 = new javax.swing.JPanel();
    CfgOk = new javax.swing.JButton();
    CfgCancel = new javax.swing.JButton();
    PingProgress = new javax.swing.JProgressBar();
    resultText = new javax.swing.JLabel();
    vmStateDialog = new javax.swing.JDialog();
    vmInfoPanel = new javax.swing.JPanel();
    adminPanel = new javax.swing.JPanel();
    jScrollPane1 = new javax.swing.JScrollPane();
    vmTable = new javax.swing.JTable();
    updateButton = new javax.swing.JButton();
    vmLoadProgress = new javax.swing.JProgressBar();
    jMenuBar1 = new javax.swing.JMenuBar();
    jMenu1 = new javax.swing.JMenu();
    jMenu2 = new javax.swing.JMenu();
    connectMenu = new javax.swing.JMenuItem();

    connectDialog.setMinimumSize(new java.awt.Dimension(265, 140));

    TestConnect.setText("Test");
    TestConnect.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            TestConnectActionPerformed(evt);
        }
    });

    jLabel1.setText("IP");

    jLabel2.setText("Port");

    jPanel1.setEnabled(false);

    CfgOk.setText("OK");
    CfgOk.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            CfgOkActionPerformed(evt);
        }
    });

    CfgCancel.setText("Отмена");
    CfgCancel.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            CfgCancelActionPerformed(evt);
        }
    });

    javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
    jPanel1.setLayout(jPanel1Layout);

```

```

jPanel1Layout.setHorizontalGroup(
    jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel1Layout.createSequentialGroup()
            .addGap(54, 54, 54)
            .addComponent(CfgOk, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(CfgCancel)
            .addContainerGap())
        );
jPanel1Layout.setVerticalGroup(
    jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel1Layout.createSequentialGroup()
            .addGap(0, 0, Short.MAX_VALUE)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(CfgOk)
                .addComponent(CfgCancel)))
        );

javax.swing.GroupLayout connectDialogLayout = new
javax.swing.GroupLayout(connectDialog.getContentPane());
connectDialog.getContentPane().setLayout(connectDialogLayout);
connectDialogLayout.setHorizontalGroup(
    connectDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(connectDialogLayout.createSequentialGroup()
            .addContainerGap()
            .addGroup(connectDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(connectDialogLayout.createSequentialGroup()
                    .addGap(0, 0, Short.MAX_VALUE)
                    .addGroup(connectDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addGroup(connectDialogLayout.createSequentialGroup()
                            .addComponent(TestConnect)
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                            .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addGroup(connectDialogLayout.createSequentialGroup()
                            .addGroup(connectDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                .addComponent(jLabel2)
                                .addComponent(jLabel1))
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addGroup(connectDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                                    .addComponent(ipField)
                                    .addComponent(portField, javax.swing.GroupLayout.DEFAULT_SIZE, 186,
Short.MAX_VALUE))))))
                            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
connectDialogLayout.createSequentialGroup()
                                .addComponent(pingProgress, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                .addContainerGap()
                                .addGroup(connectDialogLayout.createSequentialGroup()
                                    .addComponent(resultText)
                                    .addGap(0, 0, Short.MAX_VALUE))))
                            );
connectDialogLayout.setVerticalGroup(
    connectDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(connectDialogLayout.createSequentialGroup()
            .addContainerGap()
            .addGroup(connectDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel1)
                .addComponent(ipField, javax.swing.GroupLayout.PREFERRED_SIZE,

```

```

javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
    .addGroup(connectDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(jLabel2)
        .addComponent(PortField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
    .addGroup(connectDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)
        .addComponent(TestConnect, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
    .addComponent(resultText)
    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 27, Short.MAX_VALUE)
    .addComponent(PingProgress, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
    .addContainerGap())
);

vmStateDialog.setMinimumSize(new java.awt.Dimension(400, 300));

javax.swing.GroupLayout vmInfoPanelLayout = new javax.swing.GroupLayout(vmInfoPanel);
vmInfoPanel.setLayout(vmInfoPanelLayout);
vmInfoPanelLayout.setHorizontalGroup(
    vmInfoPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 400, Short.MAX_VALUE)
);
vmInfoPanelLayout.setVerticalGroup(

    vmInfoPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 300, Short.MAX_VALUE)
);

javax.swing.GroupLayout vmStateDialogLayout = new
javax.swing.GroupLayout(vmStateDialog.getContentPane());
vmStateDialog.getContentPane().setLayout(vmStateDialogLayout);
vmStateDialogLayout.setHorizontalGroup(
    vmStateDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(vmInfoPanel, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
);
vmStateDialogLayout.setVerticalGroup(
    vmStateDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(vmInfoPanel, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
);

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

vmTable.setModel(new javax.swing.table.DefaultTableModel(
    new Object [][] {

    },
    new String [] {
        "ID", "CPU", "Ram", "IP", "Port"
    }
) {
    boolean[] canEdit = new boolean [] {
        false, false, false, false, false
    }

```

```

    };

    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return canEdit [columnIndex];
    }
});
jScrollPane1.setViewportView(vmTable);

updateButton.setText("Обновить");
updateButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        updateButtonActionPerformed(evt);
    }
});

javax.swing.GroupLayout adminPanelLayout = new javax.swing.GroupLayout(adminPanel);
adminPanel.setLayout(adminPanelLayout);
adminPanelLayout.setHorizontalGroup(
    adminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 668, Short.MAX_VALUE)
        .addGroup(adminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(updateButton)
            .addGap(0, 0, Short.MAX_VALUE))
        .addComponent(vmLoadProgress, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
    );
adminPanelLayout.setVerticalGroup(
    adminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, adminPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 19, Short.MAX_VALUE)
            .addComponent(updateButton)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 449,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(vmLoadProgress, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(6, 6, 6))
    );

jMenu1.setText("File");
jMenuBar1.add(jMenu1);

jMenu2.setText("Настройка");

connectMenu.setText("Настройка подключения");
connectMenu.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        connectMenuActionPerformed(evt);
    }
});
jMenu2.add(connectMenu);

jMenuBar1.add(jMenu2);

setJMenuBar(jMenuBar1);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(adminPanel, javax.swing.GroupLayout.DEFAULT_SIZE,

```

```

javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(adminPanel, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
    );

    pack();
} // </editor-fold> // GEN-END: initComponents

private void TestConnectActionPerformed(java.awt.event.ActionEvent evt) { // GEN-
FIRST:event_TestConnectActionPerformed
    // Тест подключения
    ip = IpField.getText();
    port = PortField.getText();
    String localUrl = "http://" + ip + ":" + port + "/Server/ServerServlet";
    connect = new ClientHttp(localUrl);
    new Thread(new PingTask(new LocalPingListener(), connect)).start();
} // GEN-LAST:event_TestConnectActionPerformed

private void CfgOkActionPerformed(java.awt.event.ActionEvent evt) { // GEN-
FIRST:event_CfgOkActionPerformed
    Map<String, String> map = new HashMap();
    map.put(ConfigEditor.IP, ip);
    map.put(ConfigEditor.PORT, port);
    if (ConfigEditor.setConfig(map, CONFIG_NAME)) {
        connectDialog.setVisible(false);
        enableComponents(adminPanel, true);
    }
} // GEN-LAST:event_CfgOkActionPerformed

private void CfgCancelActionPerformed(java.awt.event.ActionEvent evt) { // GEN-
FIRST:event_CfgCancelActionPerformed
    connectDialog.setVisible(false);
} // GEN-LAST:event_CfgCancelActionPerformed

private void connectMenuActionPerformed(java.awt.event.ActionEvent evt) { // GEN-
FIRST:event_connectMenuActionPerformed
    // TODO add your handling code here:
    connectDialog.setVisible(true);
    IpField.setText(ip);
    PortField.setText(port);
} // GEN-LAST:event_connectMenuActionPerformed

private void updateButtonActionPerformed(java.awt.event.ActionEvent evt) { // GEN-
FIRST:event_updateButtonActionPerformed
    // TODO add your handling code here:
    new Thread(new VMStateTask(new LocalVMStateLoadListener(), connect)).start();
} // GEN-LAST:event_updateButtonActionPerformed

private void enableComponents(Container container, boolean enable) {
    Component[] components = container.getComponents();
    for (Component component : components) {
        component.setEnabled(enable);
        if (component instanceof Container) {
            enableComponents((Container) component, enable);
        }
    }
}

private void fillVMTable(){

```

```

        if (vm != null) {
            DefaultTableModel model = (DefaultTableModel) vmTable.getModel();
            while (model.getRowCount() > 0) {
                model.removeRow(0);
            }
            Object rowData[] = new Object[5];
            for (int i = 0; i < vm.size(); i++) {
                rowData[0] = vm.get(i).getId();
                rowData[1] = vm.get(i).getCpu();
                rowData[2] = vm.get(i).getRam();
                rowData[3] = vm.get(i).getIp();
                rowData[4] = vm.get(i).getPort();
                model.addRow(rowData);
            }
        }
    }

    private void showError(String message) {
        JOptionPane.showMessageDialog(null, message);
    }

    private void openVMStateDialog(VirtualMachine vm){
        VMStateChartPanel cp = new VMStateChartPanel(this, "", false, vm);
        cp.setVisible(true);
    }

    private class LocalMouseAdapter extends MouseAdapter {

        @Override
        public void mouseClicked(MouseEvent e) {
            Point p = e.getPoint();
            int row = vmTable.rowAtPoint(p);
            int column = vmTable.columnAtPoint(p);
            DefaultTableModel model = (DefaultTableModel) vmTable.getModel();
            System.out.println("Click on " + column + " column, " + row + " row; " + model.getValueAt(row, column));
            int id = (int) model.getValueAt(row, 0);
            int cpu = (int) model.getValueAt(row, 1);
            long ram = (long) model.getValueAt(row, 2);
            String ip = (String) model.getValueAt(row, 3);
            int port = (int) model.getValueAt(row, 4);
            VirtualMachine vm = new VirtualMachine(id, ram, cpu, ip, port);
            openVMStateDialog(vm);
        }
    }

    /**
     * Слушатель подключения
     */
    private class LocalPingListener implements ConnectListener {

        @Override
        public void onStart() {
            PingProgress.setIndeterminate(true);
        }

        @Override
        public void onResult(Response response) {
            PingProgress.setIndeterminate(false);
            if (response.getResultCode() == Response.OK) {

```

```

        enableComponents(jPanel1, true);
        resultText.setText("Connect OK");
    } else {
        resultText.setText("Connect failed");
    }
}

@Override
public void onError(Exception ex) {
    PingProgress.setIndeterminate(false);
    Logger.e(TAG, ex.getMessage());
    resultText.setText("Connect failed");
}
}

/**
 * Слушатель отправки задачи
 */
private class LocalVMStateLoadListener implements ConnectListener {

    @Override
    public void onStart() {
        vmLoadProgress.setIndeterminate(true);
        enableComponents(adminPanel, false);
    }

    @Override
    public void onResult(Response response) {
        vmLoadProgress.setIndeterminate(false);
        enableComponents(adminPanel, true);

        vm = (List<VirtualMachine>) response.getData();
        fillVMTable();
    }

    @Override
    public void onError(Exception ex) {
        vmLoadProgress.setIndeterminate(false);

        enableComponents(adminPanel, true);
        Logger.e(TAG, ex + "");

        showError(ex + "");
    }
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
    /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
     * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
     */
    try {
        for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
            if ("Nimbus".equals(info.getName())) {
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
            }
        }
    } catch (ClassNotFoundException ex) {
        Logger.e(TAG, ex.getMessage());
    } catch (InstantiationException ex) {
        Logger.e(TAG, ex.getMessage());
    } catch (IllegalAccessException ex) {
        Logger.e(TAG, ex.getMessage());
    }
}

```



```

        break;
    }
}
} catch (ClassNotFoundException ex) {
    java.util.logging.Logger.getLogger(Admin.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
} catch (InstantiationException ex) {
    java.util.logging.Logger.getLogger(Admin.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
} catch (IllegalAccessException ex) {
    java.util.logging.Logger.getLogger(Admin.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
} catch (javax.swing.UnsupportedLookAndFeelException ex) {
    java.util.logging.Logger.getLogger(Admin.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
}
}
//</editor-fold>

/* Create and display the form */
java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new Admin().setVisible(true);
    }
});
}

// Variables declaration - do not modify//GEN-BEGIN:variables
private javax.swing.JButton CfgCancel;
private javax.swing.JButton CfgOk;
private javax.swing.JTextField IpField;
private javax.swing.JProgressBar PingProgress;
private javax.swing.JTextField PortField;
private javax.swing.JButton TestConnect;
private javax.swing.JPanel adminPanel;
private javax.swing.JDialog connectDialog;
private javax.swing.JMenuItem connectMenu;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JMenu jMenu1;
private javax.swing.JMenu jMenu2;
private javax.swing.JMenuBar jMenuBar1;
private javax.swing.JPanel jPanel1;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JLabel resultText;
private javax.swing.JButton updateButton;
private javax.swing.JPanel vmInfoPanel;
private javax.swing.JProgressBar vmLoadProgress;
private javax.swing.JDialog vmStateDialog;
private javax.swing.JTable vmTable;
// End of variables declaration//GEN-END:variables
}

package admin.gui;

import java.awt.Frame;
import model.data.VirtualMachine;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.ui.StandardDialog;

/**

```

```

*
* @author user
*/
public class VMStateChartPanel extends StandardDialog {

    public VMStateChartPanel(Frame owner, String title, boolean modal, VirtualMachine vMachine) {
        super(owner, title, modal);
        JFreeChart barChart = ChartFactory.createBarChart(
            "Статистика",
            "",
            "Загруженность",
            createDataset(vMachine),
            PlotOrientation.VERTICAL,
            true, true, false);

        this.setMinimumSize(new java.awt.Dimension(400, 300));
        ChartPanel cp = new ChartPanel(barChart);
        setContentPane( cp );
    }

    private static CategoryDataset createDataset(VirtualMachine vMachine) {
        final String id = "Machine # " + vMachine.getId();
        final String ram = "bt";
        final String cpu = "%";

        final DefaultCategoryDataset dataset
            = new DefaultCategoryDataset();

        dataset.addValue(vMachine.getRam(), id, ram);
        dataset.addValue(vMachine.getCpu(), id, cpu);

        return dataset;
    }
}

package admin.tasks;

import model.web.ClientHttp;

/**
 *
 * @author user
 */
public class VMStateTask implements Runnable{

    private ClientHttp.ConnectListener listener;
    private ClientHttp client;

    public VMStateTask(ClientHttp.ConnectListener listener, ClientHttp client) {
        this.listener = listener;
        this.client = client;
    }

    @Override
    public void run() {
        listener.onStart();
        try {

```

```

        listener.onResult(client.getVMState());
    } catch (Exception ex) {
        listener.onError(ex);
    }
}
}

```

```
package admin.tasks;
```

```
import model.web.ClientHttp;
import model.web.ClientHttp.ConnectListener;
```

```

/**
 *
 * @author user
 */
public class PingTask implements Runnable{

    private ConnectListener listener;
    private ClientHttp client;

    public PingTask(ConnectListener listener, ClientHttp client) {
        this.listener = listener;
        this.client = client;
    }

    @Override
    public void run() {
        listener.onStart();
        try {
            listener.onResult(client.ping());
        } catch (Exception ex) {
            listener.onError(ex);
        }
    }
}

```

```
package admin.utils;
```

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import model.utils.Logger;
```

```

/**
 *
 * @author user
 */
public class ConfigEditor {

    private static final String TAG = "ConfigEditor";

    public static final String IP = "ip";
    public static final String PORT = "port";

```

```

public static Map<String, String> getConfig(String fileName) {
    String ip, port;
    Map<String, String> map = new HashMap();
    try (FileReader file = new FileReader(fileName)) {
        BufferedReader reader = new BufferedReader(file);
        ip = reader.readLine();
        port = reader.readLine();
        map.put(IP, ip);
        map.put(PORT, port);
        file.close();
        Logger.i(TAG, "Config load success!");
    } catch (IOException ex) {
        Logger.e(TAG, ex.toString());
        return null;
    }

    return map;
}

public static boolean setConfig(Map<String, String> args, String fileName) {
    try (FileWriter writer = new FileWriter(fileName, false)) {
        writer.write(args.get(IP) + "\n");
        writer.write(args.get(PORT));
        writer.flush();
        writer.close();
        Logger.i(TAG, "Config update success!");
        return true;
    } catch (IOException ex) {
        Logger.e(TAG, ex.toString());
        return false;
    }
}
}

```

## В.6 Модель данных

```
package model.data;

import com.google.sljson.annotations.SerializedName;

/**
 * Задача на обработку
 * @author user
 */
public class Job<T,M> {
    static final long serialVersionUID = 1L;

    @SerializedName("id")
    private T inputData;

    @SerializedName("od")
    private T outputData;

    @SerializedName("n")
    private String name;

    @SerializedName("s")
    private int state;

    public Job(T inputData, T outputData, String name, int state) {
        this.inputData = inputData;
        this.outputData = outputData;
        this.name = name;
        this.state = state;
    }

    public T getInputData() {
        return inputData;
    }

    public void setInputData(T inputData) {
        this.inputData = inputData;
    }

    public T getOutputData() {
        return outputData;
    }

    public void setOutputData(T outputData) {
        this.outputData = outputData;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getState() {
        return state;
    }

    public void setState(int state) {
```

```

        this.state = state;
    }

    @Override
    public String toString() {
        return "Job{" + "inputData=" + inputData + ", outputData=" + outputData + ", name=" + name + ", state=" +
state + '}';
    }
}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package model.data;

import com.google.gson.annotations.SerializedName;
import java.io.Serializable;

/**
 *
 * @author user
 * ответ на запрос серверу
 */
public class Response<T> implements Serializable{
    static final long serialVersionUID = 1L;

    public static final int OK = 0;          //успех
    public static final int IN_PROCESS = 1;  //данные в обработке
    public static final int WRONG_METHOD = 2; //неизвестный метод
    public static final int INTERNAL_ERROR = 3; //внутренняя ошибка сервера
    public static final int DB_ERROR = 4;    //ошибка БД
    public static final int WRONG_DATA = 5;  //неверные данные в запросе
    public static final int PROCESS_ERROR = 6; //ошибка обработки данных
    public static final int VM_NOT_FOUND = 7; //нет доступных VM

    @SerializedName("r")
    private final int resultCode;
    @SerializedName("d")
    private T data;

    public Response(int resultCode, T data) {
        this.resultCode = resultCode;
        this.data = data;
    }

    public Response(int resultCode) {
        this.resultCode = resultCode;
    }

    public T getData() {
        return data;
    }

    public void setData(T data) {
        this.data = data;
    }
}

```

```

    public int getResultCode() {
        return resultCode;
    }

    @Override
    public String toString() {
        return "Response{" + "resultCode=" + resultCode + ", data=" + data + '}';
    }
}
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package model.data;

import com.google.gson.annotations.SerializedName;
import java.io.FileReader;
import java.io.IOException;
import java.io.Serializable;
import model.utils.Logger;

/**
 *
 * @author user
 */
public class Data implements Serializable {

    static final long serialVersionUID = 1L;

    @SerializedName("d")
    private String data = "";

    public Data(String data) {
        this.data = data;
    }

    public Data(FileReader file) {
        int c;
        try {
            while ((c = file.read()) != -1) {
                data += ((char) c);
            }
        } catch (IOException ex) {
            Logger.e(Data.class.getName(), ex.toString());
        }
    }

    public String getData() {
        return data;
    }
}
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package model.data;

/**

```

```

*
* @author user
*/
public class HttpMethods {
    //параметры
    public static final String DATA = "data";
    public static final String METHOD = "method";
    public static final String VM = "virtualmachine";

    //типы запросов
    public static final String SEND_JOB = "sendJob";
    public static final String PING = "ping";
    public static final String VM_STATE = "vmstate";
    public static final String SEND_STATE = "sendvmstate";

}
/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
package model.data;

/**
 * системные параметры
 */
* @author user
*/
public class SystemParams {

    /**
     * кол-во холостых состояний подряд
     */
    public static final String FREE_COUNT = "free_count";

    /**
     * время с полседнего коннекта vm, после которого её можно считать мёртвой
     * (в мс)
     */
    public static final String LAST_CONNECT = "last_connect_ms";

    /**
     * время резерва ip без привязки в vm, после которого ip можно использовать
     * снова (в мс)
     */
    public static final String IP_RESERVE = "ip_reserve_ms";

    /**
     * max кол-во памяти, после которого машина считается загруженной
     */
    public static final String MAX_RAM = "max_ram";

    /**
     * max загруженность процессора, после которого машина считается загруженной
     */
    public static final String MAX_CPU = "max_cpu";

    /**
     * min кол-во запущенных vm
     */
    public static final String MIN_VM_COUNT = "min_vm_count";/*

```



```

* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/

```

```
package model.data;
```

```
import com.google.gson.annotations.SerializedName;
```

```
/**
```

```
*
```

```
* @author user
```

```
*/
```

```
public class VirtualMachine {
    static final long serialVersionUID = 1L;
```

```
    @SerializedName("id")
```

```
    private int id = 0;
```

```
    @SerializedName("ram")
```

```
    private long ram;
```

```
    @SerializedName("cpu")
```

```
    private int cpu;
```

```
    @SerializedName("ip")
```

```
    private String ip;
```

```
    @SerializedName("port")
```

```
    private int port;
```

```
    public VirtualMachine(int id, long ram, int cpu, String ip, int port) {
```

```
        this(ram, cpu, ip, port);
```

```
        this.id = id;
```

```
    }
```

```
    public VirtualMachine(int id, long ram, int cpu, String ip) {
```

```
        this(ram, cpu, ip);
```

```
        this.id = id;
```

```
    }
```

```
    public VirtualMachine( long ram, int cpu, String ip, int port) {
```

```
        this.ram = ram;
```

```
        this.cpu = cpu;
```

```
        this.ip = ip;
```

```
        this.port = port;
```

```
    }
```

```
    public VirtualMachine( long ram, int cpu, String ip) {
```

```
        this.ram = ram;
```

```
        this.cpu = cpu;
```

```
        this.ip = ip;
```

```
        this.port = 80;
```

```
    }
```

```
    public boolean isFree(){
```

```
        return cpu < 20;
```

```
    }
```

```
    public long getRam() {
```

```
        return ram;
```

```
    }
```

```

    public int getCpu() {
        return cpu;
    }

    public String getIp() {
        return ip;
    }

    public int getPort() {
        return port;
    }

    public int getId() {
        return id;
    }

    @Override
    public String toString() {
        return "VirtualMachine{" + " ram=" + ram + ", cpu=" + cpu + ", ip=" + ip + ", port=" + port + '}';
    }
}

}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package model.data;

/**
 *
 * @author user
 */
public class VMCommand {
    public static final String COMMAND = "Command";
    public static final String KILL = "kill_me";
    public static final String START = "start_new";
}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package model.io;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

/**
 *
 * @author user
 * билдер json'a
 */
public class JsonUtils {
    public static Gson createGson(){
        return new GsonBuilder().create();
    }
}

```

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package model.io;

import com.google.gson.Gson;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import model.data.Data;
import model.data.Response;
import model.data.VirtualMachine;

/**
 * объекты -> gson -> отправляем ответ
 */
public class ServerHandler {

    private Appendable appendable;
    private Gson gson;

    public ServerHandler(Appendable appendable) {
        this.appendable = appendable;
        this.gson = JsonUtils.createGson();
    }

    /**
     * отправить код обработки без данных
     *
     * @param errorCode
     */
    public void process(int errorCode) {
        Response res = new Response(errorCode);
        dataSend(res);
    }

    /**
     * отправить обработанные данные
     * @param data обработанные данные
     */
    public void process(Data data) {
        if (data == null) {
            throw new NullPointerException("ProcessedData is not set");
        } else {
            Response res = new Response(Response.OK, data);
            dataSend(res);
        }
    }

    /**
     * отправить состояние VM
     * @param data обработанные данные
     */
    public void process(List<VirtualMachine> data) {
        if (data == null) {
            throw new NullPointerException("List<VirtualMachine> is not set");
        } else {
            Response res = new Response(Response.OK, data);
            dataSend(res);
        }
    }
}

```

```

    }
}

public Data getProcessedData(String data){
    return gson.fromJson(data, Data.class);
}

public VirtualMachine getVirtualMachine(String data){
    return gson.fromJson(data, VirtualMachine.class);
}

/**
 * парсим Response в gson для ответа сервера
 *
 * @param res
 */
private void dataSend(Response res) {
    try {
        String data = gson.toJson(res);
        this.appendable.append(data);
    } catch (IOException ex) {
        Logger.getLogger(ServerHandler.class.getName()).log(Level.SEVERE, null, ex);
    }
}

}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package model.io;

import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
import java.lang.reflect.Type;
import java.util.List;
import model.data.Data;
import model.data.Response;
import model.data.VirtualMachine;

/**
 *
 * для парсинга ответа сервера
 */
public class ClientHandler {

    private final Gson gson;

    public ClientHandler() {
        this.gson = JsonUtils.createGson();
    }

    public String data(Data data) {
        return gson.toJson(data);
    }

    public Response<Data> getData(String str) {
        Type type = new TypeToken<Response<Data>>().
        }.getType();

```

```

        return getResponse(str, type);
    }

    public Response getResponse(String str) {
        Type type = new TypeToken<Response>() {
        }.getType();
        return getResponse(str, type);
    }

    public Response<List<VirtualMachine>> getVMState(String str) {
        Type type = new TypeToken<Response<List<VirtualMachine>>>() {
        }.getType();
        return getResponse(str, type);
    }

    private Response getResponse(String data, Type type) {
        Response res = gson.fromJson(data, type);
        return res;
    }
}

package model.utils;

/**
 *
 * @author user
 */
public class Logger {
    public static void i(String TAG, String str){
        System.out.println(TAG + ": " + str);
    }

    public static void e(String TAG, String str){
        System.err.println(TAG + ": " + str);
    }

    public static void e(String TAG, String str, Exception ex){
        System.err.println(TAG + ": " + str + "\n" + ex);
        ex.printStackTrace();
    }
}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package model.web;

import com.google.gson.Gson;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;
import model.data.Data;
import model.data.HttpMethods;
import model.data.Job;
import model.data.Response;
import model.data.VirtualMachine;
import model.io.ClientHandler;
import model.io.JsonUtils;

```

```

import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.utils.URLEncodedUtils;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;

/**
 *
 * @author user взаимодействие с сервером
 */
public class ClientHttp {

    public interface ConnectListener {
        void onStart();
        void onResult(Response response);
        void onError(Exception ex);
    }

    private final Gson gson;
    private String url;

    public ClientHttp(String url) {
        this.url = url;
        this.gson = JsonUtils.createGson();
    }

    public Response ping() throws Exception {
        ArrayList<NameValuePair> params = new ArrayList<>();
        params.add(new BasicNameValuePair(HttpMetods.METHOD, HttpMetods.PING));
        return new ClientHandler().getResponse(execute(params));

        //return new ClientHandler().getResponse(executeGet(params));
    }

    public Response<Job<Data, Data>> addJob(Job job) throws Exception {
        List<NameValuePair> params = new ArrayList<>();
        params.add(new BasicNameValuePair(HttpMetods.DATA, gson.toJson(job.getInputData())));
        params.add(new BasicNameValuePair(HttpMetods.METHOD, HttpMetods.SEND_JOB));

        Response<Data> rData = new ClientHandler().getData(execute(params));
        job.setOutputData(rData.getData());
        job.setState(rData.getResultCode());
        return new Response(rData.getResultCode(), job);
    }

    public Response<List<VirtualMachine>> getVMState() throws Exception {
        List<NameValuePair> params = new ArrayList<>();
        params.add(new BasicNameValuePair(HttpMetods.METHOD, HttpMetods.VM_STATE));

        return new ClientHandler().getVMState(execute(params));
    }

    public Response sendVMState(VirtualMachine vm) throws Exception {
        ArrayList<NameValuePair> params = new ArrayList<>();
        params.add(new BasicNameValuePair(HttpMetods.METHOD, HttpMetods.SEND_STATE));
        params.add(new BasicNameValuePair(HttpMetods.VM, gson.toJson(vm)));

        return new ClientHandler().getResponse(execute(params));
    }

```

```

        //return new ClientHandler().getResponse(executeGet(params));
    }

    private String execute(List<NameValuePair> params) throws Exception {

        System.out.println("\nSending 'POST' request to URL : " + url);
        System.out.println("Post parameters : " + URLEncodedUtils.format(params, "utf-8"));

        HttpClient client = new DefaultHttpClient();
        HttpPost post = new HttpPost(url + "?" + URLEncodedUtils.format(params, "utf-8"));

        // add header
        post.setHeader("charset", "UTF-8");
        post.setHeader("Content-Type", "application/json");

        HttpResponse response = client.execute(post);

        System.out.println("Response Code : "
            + response.getStatusLine().getStatusCode());

        BufferedReader rd = new BufferedReader(
            new InputStreamReader(response.getEntity().getContent()));

        StringBuffer result = new StringBuffer();
        String line = "";
        while ((line = rd.readLine()) != null) {
            result.append(line);
        }

        System.out.println("result: " + result.toString());
        return result.toString();
    }

    private String executeGet(List<NameValuePair> params) throws Exception {

        System.out.println("\nSending 'GET' request to URL : " + url);
        System.out.println("GET parameters : " + params);

        HttpClient client = new DefaultHttpClient();
       HttpGet get = new HttpGet(url + "?" + URLEncodedUtils.format(params, "utf-8"));

        // add header
        get.setHeader("charset", "UTF-8");
        get.setHeader("Content-Type", "application/json");

        HttpResponse response = client.execute(get);

        System.out.println("Response Code : "
            + response.getStatusLine().getStatusCode());

        BufferedReader rd = new BufferedReader(
            new InputStreamReader(response.getEntity().getContent()));
        StringBuffer result = new StringBuffer();
        String line = "";
        while ((line = rd.readLine()) != null) {
            result.append(line);
        }
        System.out.println("result: " + result.toString());
        return result.toString();
    }
}

```

## B.7 Web сервис

```
#include <cmath>
#include <sstream>
#include <string>

#include <os>
#include <net/inet4>
#include <timers>
#include <net/http/request.hpp>
#include <net/http/response.hpp>

using namespace std::chrono;

uint64_t old_halt_ = 0;
uint64_t new_halt_ = 0;
uint64_t old_total_ = 0;
uint64_t new_total_ = 0;

uint64_t serialized_halt_ = 0;
uint64_t serialized_total_ = 0;

std::string service_handler(const std::string& s) {
    int sp = s.find("/?data=") + 6;
    int fp = s.find(" HTTP");

    std::string number = s.substr(sp, fp - sp);
    long myLong = std::stol( number );

    int count = 0;
    for (long i = 0; i <= myLong; i++) {
        int divCount = 0;
        for (long j = 1; j <= i; j++) {
            if(i%j == 0){
                divCount++;
            }
        }
        if(divCount == 2){
            count++;
        }
    }

    std::stringstream stream;
    stream << "{"d\":"simple count = " + std::to_string(count) + "\"}";

    return stream.str();
}

http::Response handle_request(const std::string& s) {
    http::Response res;

    auto& header = res.header();

    header.set_field(http::header::Server, "IncludeOS/0.10");

    res.add_body(service_handler(s));
    header.set_field(http::header::Content_Type, "text/html; charset=UTF-8");
    header.set_field(http::header::Content_Length, std::to_string(res.body().to_string().size()));

    header.set_field(http::header::Connection, "close");
}
```



```

    return res;
}

double getActive() {
    uint64_t temp_halt = new_halt_;
    uint64_t temp_total = new_total_;

    new_halt_ = OS::get_cycles_halt();
    new_total_ = OS::get_cycles_total();
    old_halt_ = temp_halt;
    old_total_ = temp_total;

    if (new_halt_ > old_halt_)
        serialized_halt_ = new_halt_ - old_halt_;

    if (new_total_ > old_total_)
        serialized_total_ = new_total_ - old_total_;

    if (serialized_total_ > serialized_halt_) {
        return serialized_total_ - serialized_halt_;
    } else {
        return 0;
    }
}

void ping() {
    printf("\nGo ping server:");

    int proc = (getActive() * 100.0) / ((OS::cpu_freq().count() * 1000) * 1000);

    if (proc < 100) {
        printf("%d:", proc);
    } else {
        printf("100:");
    }

    printf("%d\n", OS::heap_usage());
}

void Service::start(const std::string& s) {
    printf("Start params: %s\n\n", s.c_str());
    //get ip from start params
    std::size_t ipPos = s.find("!");
    // DHCP on interface 0
    auto& inet = net::Inet4::ifconfig(10.0);
    // static IP in case DHCP fails

    net::Inet4::ifconfig(
        net::ip4::Addr(s.substr(ipPos + 1)), // IP
        {
            255, 255, 255, 0
        }, // Netmask
        {
            10, 0, 0, 1 }, // Gateway
        {
            10, 0, 0, 1 }); // DNS

    // Print some useful netstats every 30 secs
    //TODO: тут будет отправка своего состояния
    Timers::periodic(5s, 5s,
        [&inet, &s] (uint32_t) {
            printf("<Service(%s)> TCP STATUS:\n%s\n", s.substr(s.find("!") + 1).c_str(), inet.tcp().status().c_str());
        });
}

```

```

        ping();
    });

// Set up a TCP server on port 80
auto& server = inet.tcp().bind(80);

// Add a TCP connection handler - here a hardcoded HTTP-service
server.on_connect(
    [] (auto conn) {
        printf("<Service> @on_connect: Connection %s successfully established.\n",
            conn->remote().to_string().c_str());
        // read async with a buffer size of 1024 bytes
        // define what to do when data is read
        conn->on_read(1024,
            [conn] (auto buf, size_t n) {
                printf("<Service> @on_read: %u bytes received.\n", n);
                printf("\nbuf: %s\n", buf.get());
                auto res = handle_request(std::string(reinterpret_cast<char*> (buf.get())));
                printf("<Service> Responding with %u %s.\n",
                    res.status_code(), http::code_description(res.status_code()).to_string().c_str());

                conn->write(res, [] (size_t written) {
                    printf("<Service> @on_write: %u bytes written.\n", written);
                });
            });
    });

    printf("*** SERVICE STARTED ***\n");
}

```