

Севастопольский государственный университет

Методические указания
к выполнению лабораторных работ
по дисциплине «Основы облачных
технологий»

А.Р. Умеров¹

Е.Н. Мащенко²

26 июня 2016 г.³

¹admin@amet13.name

²elmachenko@mail.ru

³Дата последней правки документа

Содержание

1	ЛР №1. Знакомство с виртуализацией, VirtualBox	3
1.1	Теоретические основы виртуализации	3
1.2	Порядок выполнения работы	7
1.3	Контрольные вопросы	7
2	ЛР №2. Модель обслуживания SaaS, ownCloud	8
2.1	Теоретические основы облачных вычислений	8
2.2	Порядок выполнения работы	10
2.3	Контрольные вопросы	10
3	ЛР №3. Модель обслуживания PaaS, Heroku	11
3.1	Краткие сведения о методологиях разработки ПО	11
3.2	Порядок выполнения работы	12
3.3	Контрольные вопросы	12
4	ЛР №4. Модель обслуживания IaaS, OpenStack	13
4.1	Краткие сведения об OpenStack	13
4.2	Порядок выполнения работы	14
4.3	Контрольные вопросы	14
A	Создание виртуальной машины в Oracle VM VirtualBox	15
B	Установка Debian GNU/Linux в VirtualBox	22
C	Установка ownCloud Server в Debian GNU/Linux	34
D	Настройка подключения ownCloud Client к серверу	38
E	Деплой приложения на Heroku	41
F	Создание инстанса в OpenStack	44

1 Лабораторная работа №1.

Знакомство с виртуализацией. Система виртуализации Oracle VM VirtualBox

Цель работы: ознакомиться с основными понятиями виртуализации, системой виртуализации VirtualBox, научиться настраивать виртуальную машину (ВМ), совершать простейшие операции с ней, устанавливать операционную систему на ВМ.

1.1 Теоретические основы виртуализации

Виртуализация — абстракция вычислительных ресурсов и предоставление пользователю системы, которая инкапсулирует (скрывает в себе) собственную реализацию.

Виртуализацию можно использовать в:

- консолидации серверов (позволяет мигрировать с физических серверов на виртуальные, тем самым увеличивается коэффициент использования аппаратуры, что позволяет существенно сэкономить на аппаратуре, электроэнергии и обслуживании);
- разработке и тестировании приложений (возможность одновременно запускать несколько различных ОС, это удобно при разработке кроссплатформенного ПО, тем самым значительно повышается качество, скорость разработки и тестирования приложений);
- бизнесе (использование виртуализации в бизнесе растет с каждым днем и постоянно находятся новые способы применения этой технологии, например, возможность безболезненно сделать снапшот¹ и быстро восстановить систему в случае сбоя);
- организации виртуальных рабочих станций (так называемых «тонких клиентов»²).

Общая схема взаимодействия виртуализации с аппаратурой и программным обеспечением (ПО) представлена на рис. 1.

Взаимодействие приложений и операционной системы (ОС) с аппаратным обеспечением осуществляется через абстрагированный слой виртуализации.

Существует несколько подходов организации виртуализации:

- эмуляция оборудования (QEMU, Bochs, Dynamips);
- полная виртуализация (KVM, HyperV, VirtualBox);
- паравиртуализация (Xen, L4, Trango);
- виртуализация уровня ОС (LXC, OpenVZ, Jails, Solaris Zones).

¹Снапшот (англ. snapshot) — снимок состояния ВМ в определенный момент времени. Сюда входят настройки ВМ, содержимое памяти и дисков

²Тонкий клиент (англ. thin client) — бездисковый компьютер-клиент в сетях с клиент-серверной или терминальной архитектурой, который переносит все или большую часть задач по обработке информации на сервер

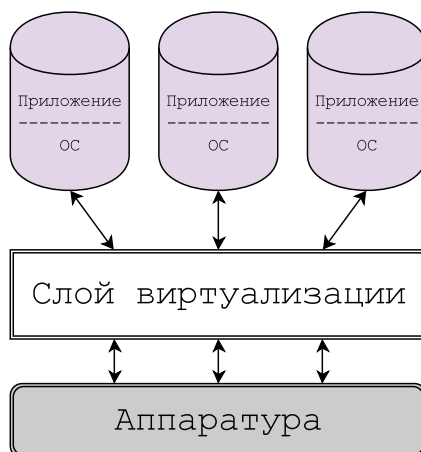


Рис. 1: Схема взаимодействия виртуализации с аппаратурой и ПО

Эмуляция аппаратных средств является одним из самых сложных методов виртуализации (рис. 2). В то же время главной проблемой при эмуляции аппаратных средств является низкая скорость работы, в связи с тем, что каждая команда моделируется на основных аппаратных средствах. В эмуляции оборудования используется механизм динамической трансляции, то есть каждая из инструкций эмулируемой платформы заменяется на заранее подготовленный фрагмент инструкций физического процессора.

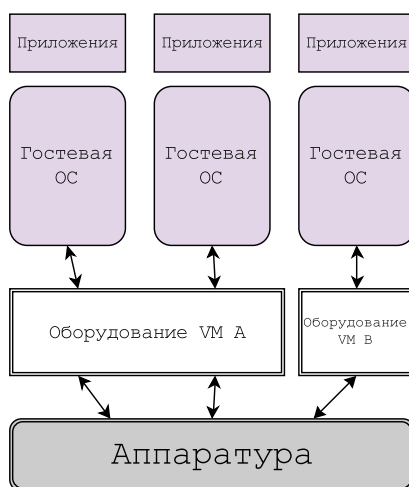


Рис. 2: Эмуляция оборудования моделирует аппаратные средства

В случае полной виртуализации поверх уже установленной ОС, устанавливается программа-гипервизор¹, которая осуществляет взаимосвязь между гостевыми ОС и хост-компьютером (рис. 3).

Преимуществом технологии полной виртуализации является установка различных ОС, а недостатком — меньшая производительность, за счет накладных расходов на гипервизор, а также понижение скорости работы с подсистемой ввода/вывода из-за необходимости изоляции.

¹Гипервизор (англ. hypervisor)— программа или аппаратная схема, позволяющая одновременное, параллельное выполнение нескольких ОС на одном и том же компьютере, обеспечивает изоляцию операционных систем друг от друга

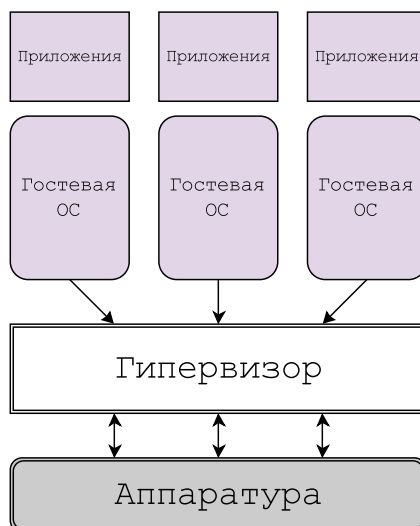


Рис. 3: Полная виртуализация использует гипервизор

Паравиртуализация имеет некоторые сходства с полной виртуализацией. В данном методе также используется гипервизор для разделения доступа к аппаратуре, но объединяется код, касающийся виртуализации, в ОС (рис. 4).

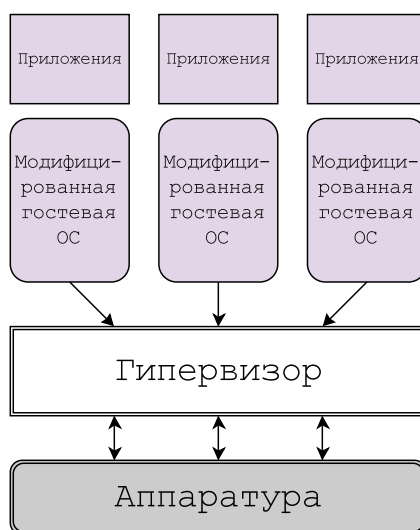


Рис. 4: Паравиртуализация разделяет процесс с гостевой ОС

Недостатком паравиртуализации является необходимость изменения гостевой ОС для гипервизора, однако таким образом гораздо увеличивается производительность.

Виртуализация уровня операционной системы не нуждается в гипервизоре. Для ее работы необходимо модифицированное ядро на хост-системе с набором патчей и утилит для управления контейнерами¹ (рис. 5).

За счет того, что контейнер напрямую взаимодействует с ядром, а не через гипервизор, обеспечивается максимальное быстродействие. Но, так как для всех контейнеров используется общее ядро, то нет возможности использовать разные ОС в контейнерах.

¹ Контейнер или VPS/VDS (англ. Virtual Private/Dedicated Server) — виртуальный выделенный сервер, эмулирует работу физического сервера

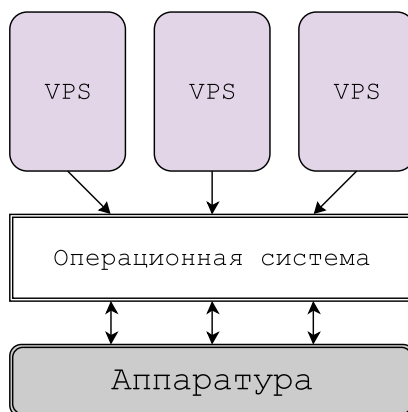


Рис. 5: Виртуализация уровня ОС изолирует серверы

Oracle VM VirtualBox — система виртуализации, разработанная компанией Innotek в 2007 году, позже приобретена компанией Sun Microsystems. Ключевыми возможностями системы является кроссплатформенность, наличие графического интерфейса, локализация, поддержка аппаратной виртуализации, экспериментальное 3D-ускорение, поддержка различных образов жестких дисков, возможность установки дополнений гостевой ОС, например для корректной работы проброшенных USB-устройств или возможности изменения разрешения рабочего стола гостевой ОС.

На рис. 6 изображен пример одновременной работы двух виртуальных машин (Windows 7 и CentOS 7).

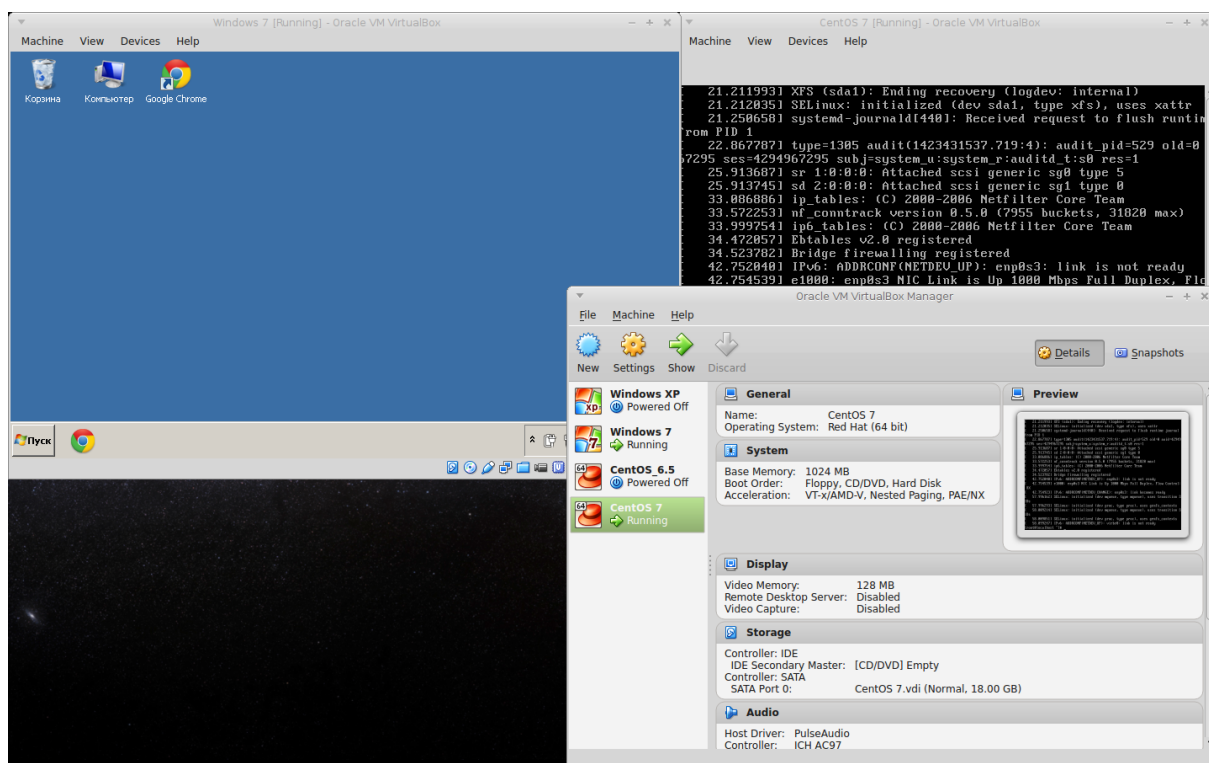


Рис. 6: Пример одновременной работы Windows 7 и CentOS 7

1.2 Порядок выполнения работы

1. Скачать¹ и установить Oracle VM VirtualBox;
2. Изучить интерфейс программы и создать² первую виртуальную машину на основе образа Debian GNU/Linux³;
3. Подключить к виртуальной машине ранее скачанный образ Debian GNU/Linux;
4. Загрузиться с подключенного образа и установить⁴ дистрибутив на виртуальный жесткий диск;
5. Во время установки дистрибутива необходимо будет задать пароль суперпользователя, пароль: `toor`, пароль для локального пользователя можно задать произвольный, но лучше его запомнить;
6. С помощью команд `free`, `df`⁵, `cat /proc/cpuinfo`, `ifconfig`, посмотреть параметры виртуальной машины;
7. С помощью команды `ping` проверить доступность виртуальной машины в сети как с хост-ноды, так и с других компьютеров сети;
8. С помощью SSH-клиента (например Putty⁶) подключиться по SSH к виртуальной машине.

1.3 Контрольные вопросы

1. Какие типы виртуализации Вы знаете? В чем между ними различия?
2. К какому типу виртуализации относится система Oracle VM VirtualBox?
3. Как может использовать виртуализацию в работе веб-программист, системный администратор, сетевой инженер?
4. В чем состоит основное преимущество и недостаток полной виртуализации от контейнерной?
5. В чем различие между режимами сети NAT и сетевой мост (Network Bridge)?

¹<https://www.virtualbox.org/wiki/Downloads>

²Пример создания виртуальной машины описан в прил. А

³<https://www.debian.org/distrib/>

⁴Процесс установки Debian GNU/Linux описан в прил. В

⁵Команды `free` и `df` используют ключ `-h` для показа информации в удобном для человека виде

⁶<http://www.chiark.greenend.org.uk/~sgtatham/putty/>

2 Лабораторная работа №2.

Модель обслуживания SaaS на примере развертывания облачного хранилища ownCloud

Цель работы: ознакомиться с основными моделями представления облачных услуг (SaaS, PaaS, IaaS), развернуть собственное облачное хранилище, доступное пользователям в пределах локальной сети.

2.1 Теоретические основы облачных вычислений

Облачные вычисления («облака», Cloud computing) — модель предоставления вычислительных ресурсов, охватывающая все, начиная от приложений и до центров обработки данных (ЦОД), через Интернет при условии оплаты за фактическое использование.

Одной из первых, кто стал внедрять услугу облачных вычислений стала компания Amazon, в то время (2002 г.) она еще являлась книжным Интернет-магазином, который впоследствии перерос, благодаря этим внедрениям, в одну из мощнейших технологических компаний. Уже в 2006 г. был запущен проект под названием Computing Cloud (Amazon EC2), после этого в 2009 г. компания Google представила Google Apps. После этих событий были сформированы общие понятия об облачных вычислениях, в частности выделены наиболее важные модели обслуживания и модели развертывания.

Облачные вычисления являются следующим шагом в эволюции архитектуры построения информационных систем. Благодаря преимуществам данного подхода вполне очевидно, что многие информационные системы в ближайшее время перенесутся или будут перенесены в облако. Процесс уже идет полным ходом и его игнорирование или недооценка может привести к поражению в конкурентной борьбе на рынке. В данном случае имеется в виду не только отставание ИТ или неоправданные затраты на него, но и отставание в развитии бизнеса компании, которая зависит от гибкости информационной инфраструктуры и скорости вывода новых сервисов и продуктов на рынок.

Различают три основные модели обслуживания:

1. программное обеспечение как услуга (Software-as-a-Service, SaaS);
2. платформа как услуга (Platform-as-a-Service, PaaS);
3. инфраструктура как услуга (Infrastructure-as-a-Service, IaaS).

SaaS — модель, при которой не требуется приобретать, устанавливать, обновлять и поддерживать ПО, эту задачу берет на себя поставщик услуги. Кроме того, осуществить регистрацию и использование облачных приложений можно немедленно, приложения и данные приложений доступны с любого устройства, подключенного к Интернету. При поломке устройства данные не теряются, они хранятся в облаке, пользователю, как правило, доступны локальные настройки конфигурации приложения. Примерами моделей SaaS могут служить: почтовая служба Gmail, Skype, CRM (система управления взаимоотношения с клиентами) и ERP (планирование ресурсов предприятия) системы и другие.

РaaS — модель, при которой пользователю предоставляется возможность использования облачной инфраструктуры для размещения базового ПО. В таком случае конфигурирование программного обеспечения целиком ложится на пользователя, предоставляется только платформа для развертывания ПО. Примером модели РaaS является предоставление услуги развертывания собственного ПО в рамках облачной инфраструктуры. Например Heroku, OpenShift.

IaaS — модель, при которой пользователю, доступно полное управление облаком в рамках операционной системы. Потребитель обладает контролем над операционными системами, сетевыми сервисами. Данная модель подходит задачам, для которых характерно быстрое изменение нагрузки. Пользователю предоставляется виртуализированное окружение, как правило с «чистой» операционной системой, пригодной для развертывания любого приложения. Примеры: Digital Ocean, Microsoft Azure, Google App Engine и т.д;

Модели развертывания облака можно разделить на четыре вида:

1. частное облако (private cloud);
2. публичное облако (public cloud);
3. общественное облако (community cloud);
4. гибридное облако (hybrid cloud).

Частное облако предназначено для использования одной организацией, публичное — для широкой публики, общественное, как правило для сообщества или организации, гибридное облако состоит из двух или более различных облачных инфраструктур.

Основными преимуществами использования облачных технологий являются:

- снижение расходов на закупку оборудования и построения центров обработки данных (ЦОД);
- удобство использования приложений с большого количества устройств, в том числе и мобильных;
- обеспечение надежности хранения данных, производительности приложений, за счет простоты использования ПО, мониторинга, балансировки нагрузки, миграции данных и т.д.

ownCloud — свободное веб-приложение, предназначенное для синхронизации данных между сервером и клиентами, как правило данными являются документы и медиаконтент. ownCloud является альтернативой таким облачным сервисам как Dropbox, Google Drive, Яндекс Диск, MEGA и др. Отличие состоит в том, что приложение можно развернуть на собственном сервере как для домашнего использования, так и для использования в организациях. Так как приложение распространяется бесплатно, имеет открытый исходный код¹ и периодически обрастает новыми функциями (планировщик задач, календарь, фотогалерея, просмотрщик документов, гибкая аутентификация пользователей и другие) проект обрел популярность как у пользователей, так и у Open-source разработчиков.

Установка ownCloud происходит в несколько простых шагов. ownCloud Server доступен для платформ Windows и Linux, ownCloud Client также доступен для Windows, Linux, а также Mac.

¹<https://github.com/owncloud/>

2.2 Порядок выполнения работы

Выполнять работу рекомендуется группами студентов по 3-5 человек. В качестве сервера может использоваться виртуальная машина с установленным дистрибутивом Debian, ранее установленная в лабораторной работе №1. Виртуальная машина должна иметь доступ в сеть Интернет для скачивания нужных пакетов.

1. Установить¹ ownCloud Server в виртуальную машину;
2. На хост-машине или на другом компьютере сети скачать² и установить ownCloud Client;
3. Подключить ownCloud Client к серверу во время первого запуска клиента, а также синхронизировать все данные с сервера³;
4. Проверить работу синхронизации данных между клиентом и сервером (создать, удалить, переместить файл или каталог);
5. Предоставить публичную ссылку на файл или каталог и проверить его доступность в пределах локальной сети;
6. Ознакомиться с дополнительными возможностями ownCloud.

2.3 Контрольные вопросы

1. Каковы основные преимущества использования облачных технологий?
2. В чем состоит отличие SaaS от PaaS и IaaS?
3. В чем состоят преимущества ownCloud по сравнению с Dropbox или другими облачными хранилищами? Недостатки?
4. Можно ли Skype отнести к модели SaaS, почему?

¹Пример установки ownCloud Server представлен в прил. С

²<https://owncloud.org/install/>

³Скриншоты настройки ownCloud Client представлены в прил. D

3 Лабораторная работа №3.

Модель обслуживания PaaS на примере деплоя приложения в Heroku

Цель работы: ознакомиться с моделью гибкой (agile) разработки программного обеспечения, развернуть приложение на Heroku, ознакомиться с системой контроля версий Git.

3.1 Краткие сведения о методологиях разработки ПО

За время существования информационных технологий создавались и изменялись подходы к построению информационных систем. Первой моделью информационной системы была монолитная архитектура. В данной модели на одном компьютере работали и приложения и база данных (БД), а пользователи сидели у «тонких» терминалов которые отображали информацию с компьютера.

У данной архитектуры было большое количество недостатков, поэтому впоследствии ее сменила более перспективная клиент-серверная архитектура. В этом случае на компьютере располагался выделенный сервер баз данных, а пользователи с «толстых клиентов» разгружали сервер БД.

Затем появилась более современная многоуровневая архитектура, у которой логика приложений вынесена на отдельный компьютер, который называется сервер приложений, а пользователи работали на «тонких» клиентах через веб-браузеры. В современном информационном мире большинство приложений выполнено именно в многоуровневой архитектуре. Она подразумевает развертывание всей ИТ-инфраструктуры на территории заказчика.

Организация процесса разработки программного обеспечения также претерпела много изменений с течением времени. Одна из самых старых методологий — каскадная (водопадная), подразумевает последовательное прохождение стадий, каждая из которых должна завершиться полностью до начала следующей. В этой модели легко управлять проектом. Благодаря ее жесткости, разработка проходит быстро, стоимость и срок заранее определены.

В «гибкой» (Agile) методологии разработки после каждой итерации заказчик может наблюдать результат и понимать, удовлетворяет он его или нет. Это одно из преимуществ гибкой модели. К ее недостаткам относят то, что из-за отсутствия конкретных формулировок результатов сложно оценить трудозатраты и стоимость, требуемые на разработку. Экстремальное программирование (XP) является одним из наиболее известных применений гибкой модели на практике.

В последнее время все большую популярность приобретает именно Agile-методология, в которой необходимо на каждой итерации разработки проводить развертку новой версии ПО, гораздо удобнее это делать с помощью PaaS-решений, таких как Heroku.

Heroku является облачной PaaS-платформой, поддерживающей ряд языков программирования, таких как Java, Node.js, Scala, Clojure, Python и PHP. В Heroku можно легко развертывать (деплоить) проекты, не беспокоясь о развертывании собственной инфраструктуры для одного приложения.

Приложения, работающие на Heroku, используют также DNS-сервер Heroku. Для каждого приложения выделяется несколько независимых виртуальных процессов,

которые называются «dynos». Они распределены по специальной виртуальной сетке «dynos grid», которая состоит из нескольких серверов. Heroku также поддерживает систему контроля версий Git, а также подключение к аккаунту GitHub.

3.2 Порядок выполнения работы

В качестве сервера может использоваться виртуальная машина с установленным дистрибутивом Debian, ранее установленная в лабораторной работе №1. Виртуальная машина должна иметь доступ в сеть Интернет для скачивания нужных пакетов и работы с Heroku.

1. Зарегистрироваться в Heroku¹;
2. Создать тестовое приложение на языке Python;
3. Задеплоить приложение на Heroku;
4. Внести изменения в исходный код приложения и снова задеплоить приложение;
5. Ознакомиться с дополнительными возможностями Heroku и Git.

3.3 Контрольные вопросы

1. Почему методология гибкой разработки ПО становится все более популярной?
2. В чем преимущество использования PaaS-решений по сравнению с IaaS для развертывания своего ПО?
3. Для чего необходима команда commit в Git?
4. В чем преимущество использования Heroku по сравнению с деплоем в локальном окружении?

¹Пример работы с Heroku представлен в прил. **Е**

4 Лабораторная работа №4.

Модель обслуживания IaaS на примере создания инстанса в OpenStack

Цель работы: ознакомиться с веб-интерфейсом OpenStack, научиться настраивать сеть, создавать инстансы и управлять ими в OpenStack.

4.1 Краткие сведения об OpenStack

Ранее в лабораторной работе №3, мы уже использовали готовые PaaS-решения от облачных провайдеров, однако если же мы хотим использовать более гибкие решения или же самим стать облачным провайдером, то необходимо использование IaaS-решений, таких как OpenStack.

С помощью OpenStack можно создавать платформы облачных вычислений для частных и публичных облаков. OpenStack был начат как совместный проект между Rackspace и NASA в 2010 году. С 2012 года им управляет некоммерческая организация OpenStack Foundation.

Теперь OpenStack поддерживают более чем 500 сторонних организаций. OpenStack является открытым проектом, исходные коды распространяются под лицензией Apache 2.0.

Помимо обеспечения IaaS-решений, OpenStack развивалась с течением времени, чтобы предоставлять другие услуги, как базы данных, системы хранения данных и прочее.

Благодаря модульной архитектуре OpenStack, любой желающий может добавить дополнительные компоненты, чтобы получить специфические особенности или функциональные возможности.

Некоторые из основных компонентов OpenStack:

- Keystone, инструментом идентификации пользователей;
- Nova, диспетчер облачного процесса вычислений;
- Horizon, панель управления для OpenStack;
- Neutron, реализация сети в качестве сервиса, обеспечение сетевых возможностей для различных компонентов;
- Glance, используется для управления образами ОС, которые требуются для работающих экземпляров (инстансов);
- Swift, распределенное хранилище высокой доступности;
- Cinder, блочное хранилище;
- Heat, инструмент оркестровки, позволяет запускать готовые облачные архитектуры из шаблонов, описанных текстом;
- Celiometer, инструмент для сбора различных статистических данных в облаке.

Каждый из компонентов OpenStack является также модульным. Например, с помощью Nova мы можем выбрать гипервизор в зависимости от требований, Libvirt (QEMU/KVM), Hyper-V, VMware, XenServer, Xen с Libvirt.

Преимущества использования OpenStack:

- решение с открытым исходным кодом;
- платформа облачных вычислений для публичных и частных облаков;
- предлагает гибкое и настраиваемое окружение;
- обеспечивает высокий уровень безопасности;
- облегчает автоматизацию на протяжении всех этапов жизненного цикла облака;
- за счет снижения затрат на управление системой и привязанности к вендору, это может быть экономически эффективным.

4.2 Порядок выполнения работы

Для выполнения данной лабораторной работы необходим аккаунт на Facebook. Пример работы с TryStack представлен в прил. F.

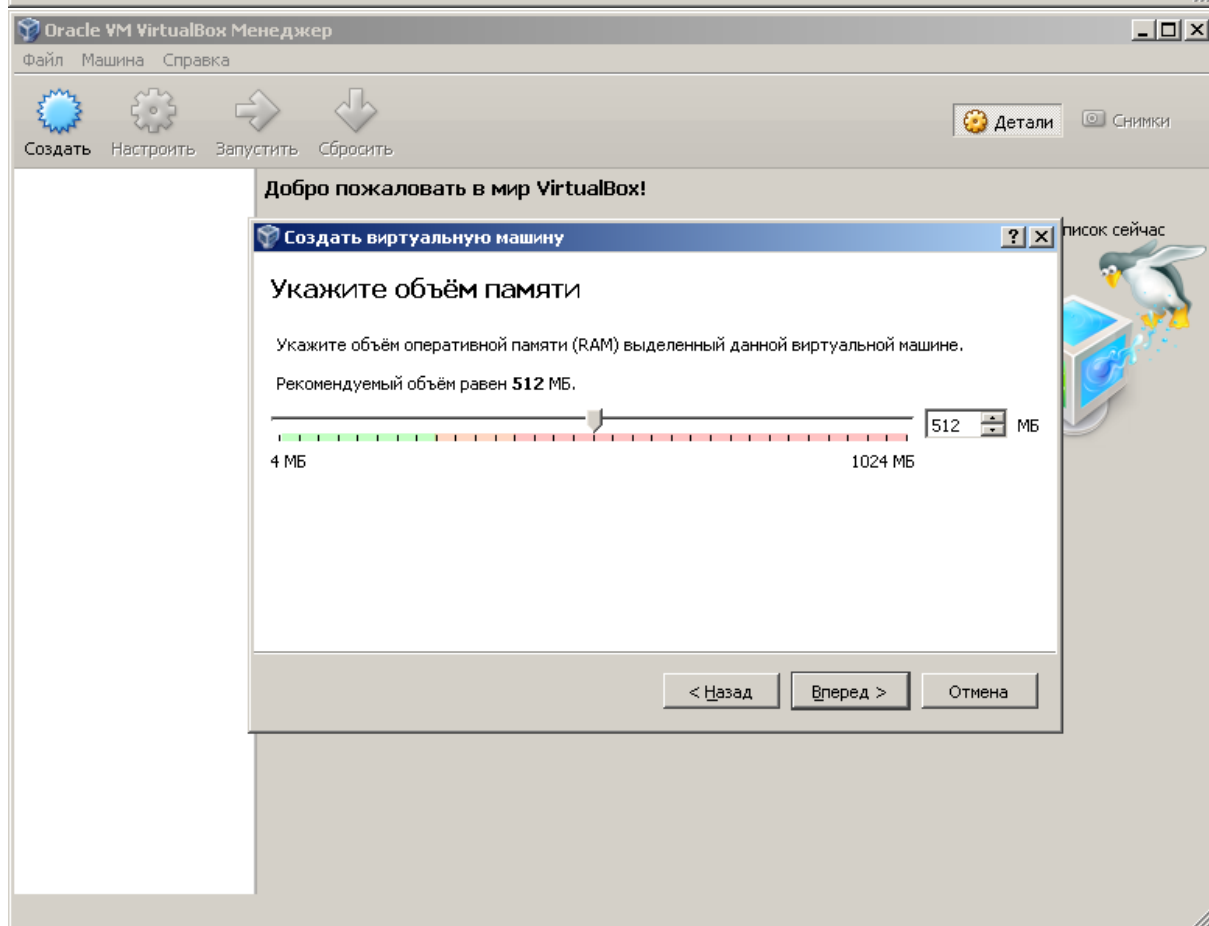
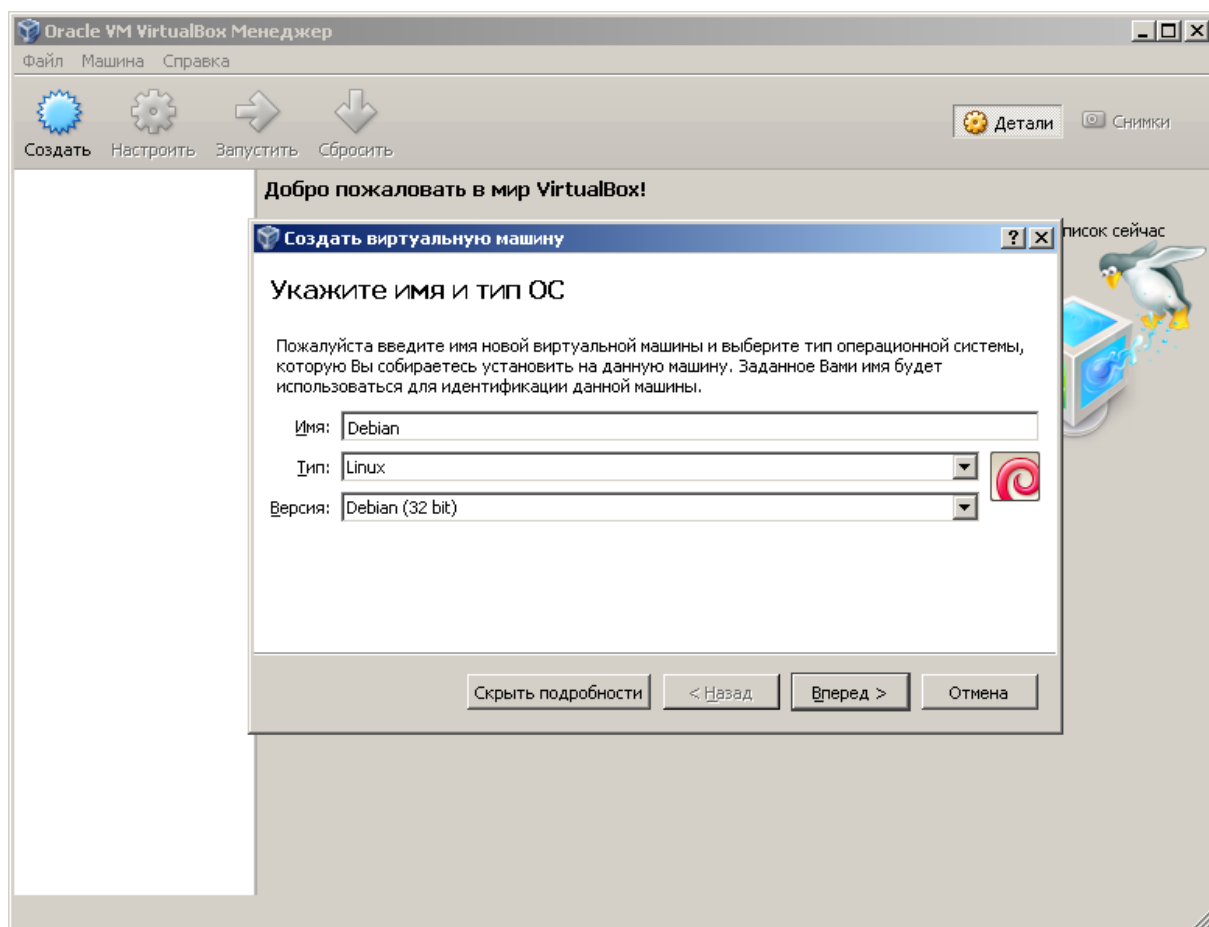
1. Вступить в группу TryStack на Facebook;
2. Авторизоваться на сервисе TryStack;
3. Ознакомиться с интерфейсом OpenStack;
4. Согласно прил. F настроить окружение для создания инстанса;
5. Создать тестовый инстанс и разместить любое приложение (по желанию);
6. Проверить работоспособность приложения в облаке.

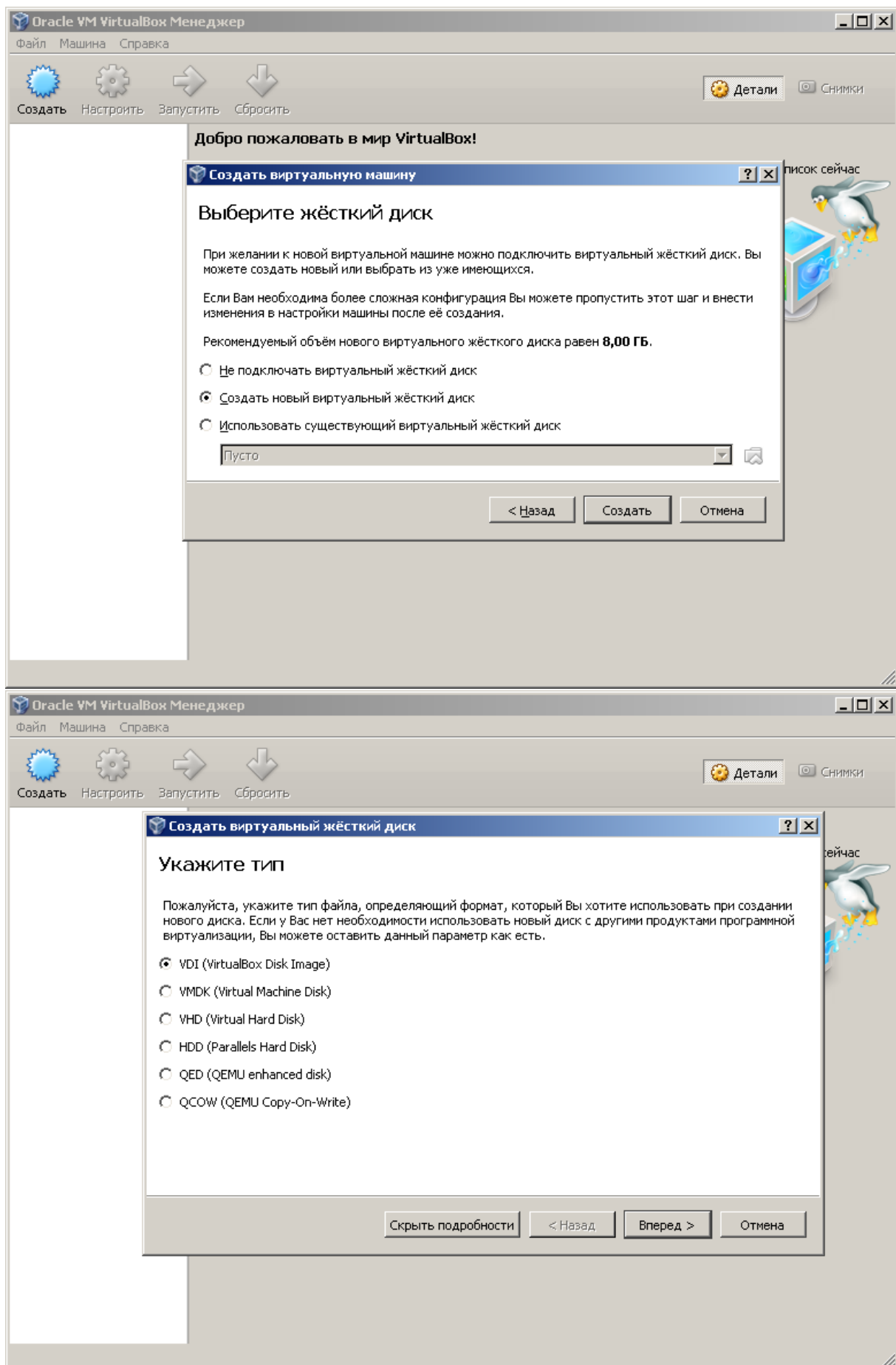
4.3 Контрольные вопросы

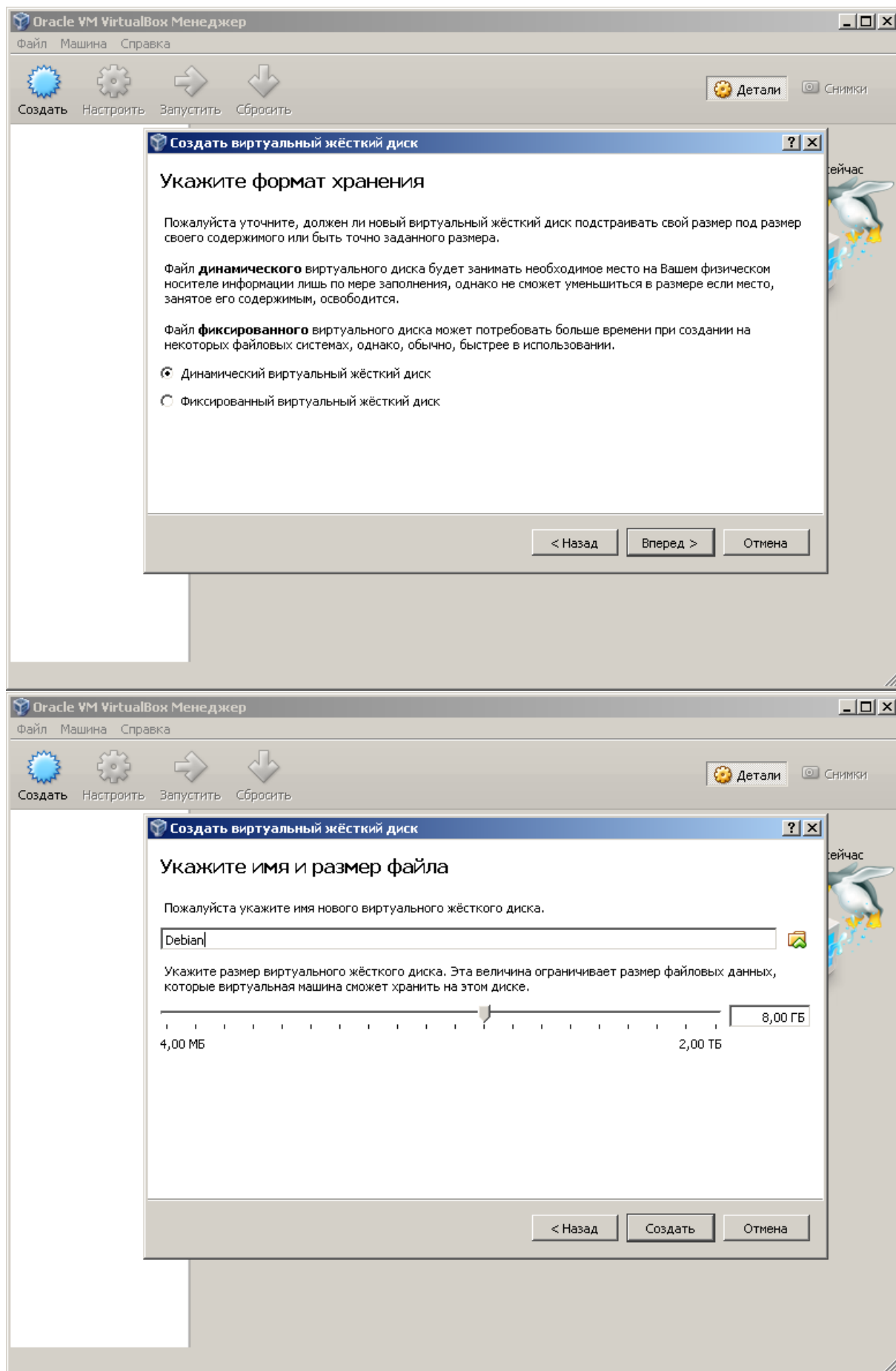
1. В чем преимущество и недостатки использования IaaS перед PaaS?
2. Назовите несколько примеров облачных поставщиков IaaS?
3. Почему использование SSH-ключей безопаснее чем парольная аутентификация?
4. В чем преимущество модульной архитектуры перед монолитной?

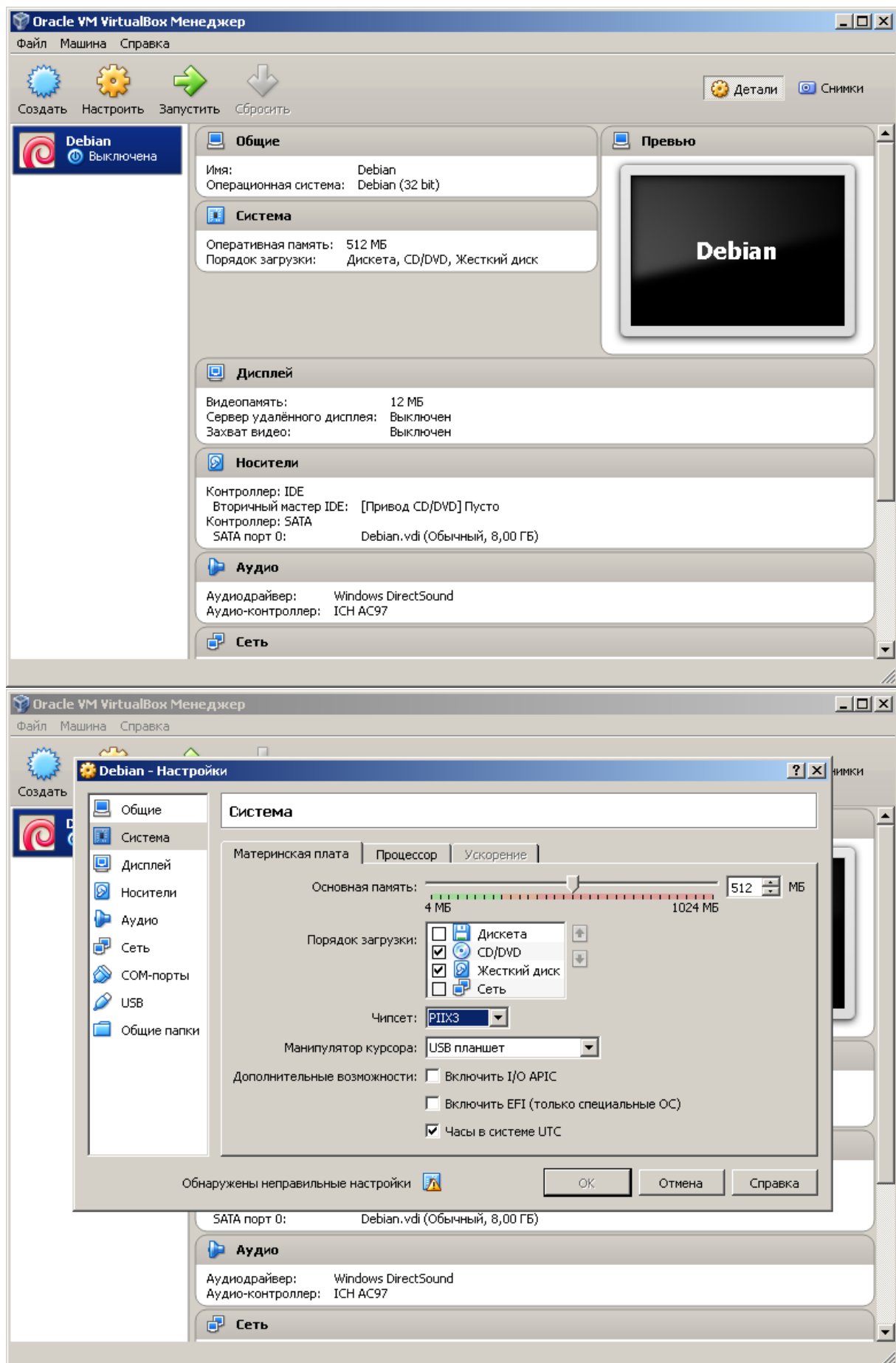
А Создание виртуальной машины в Oracle VM VirtualBox

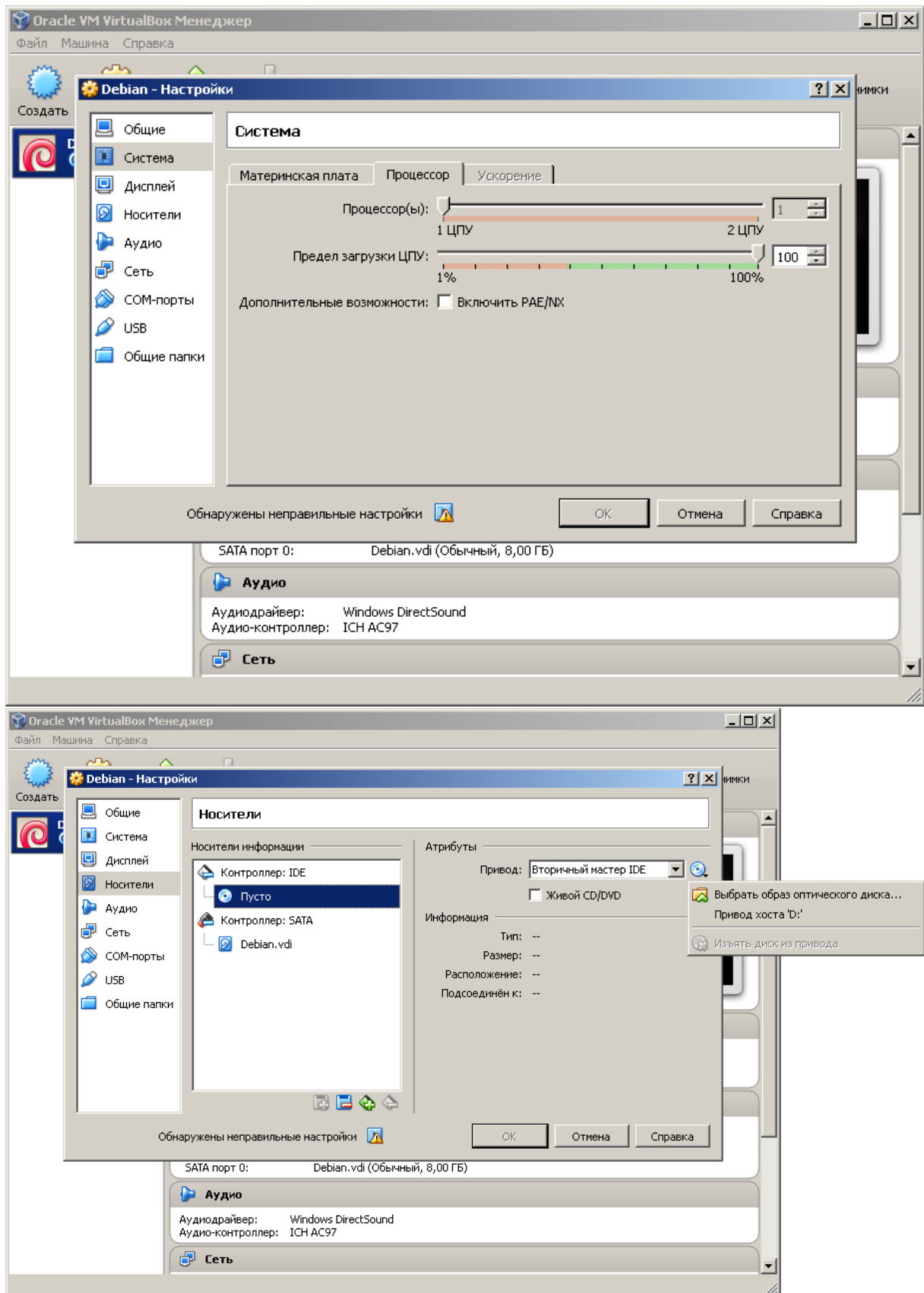


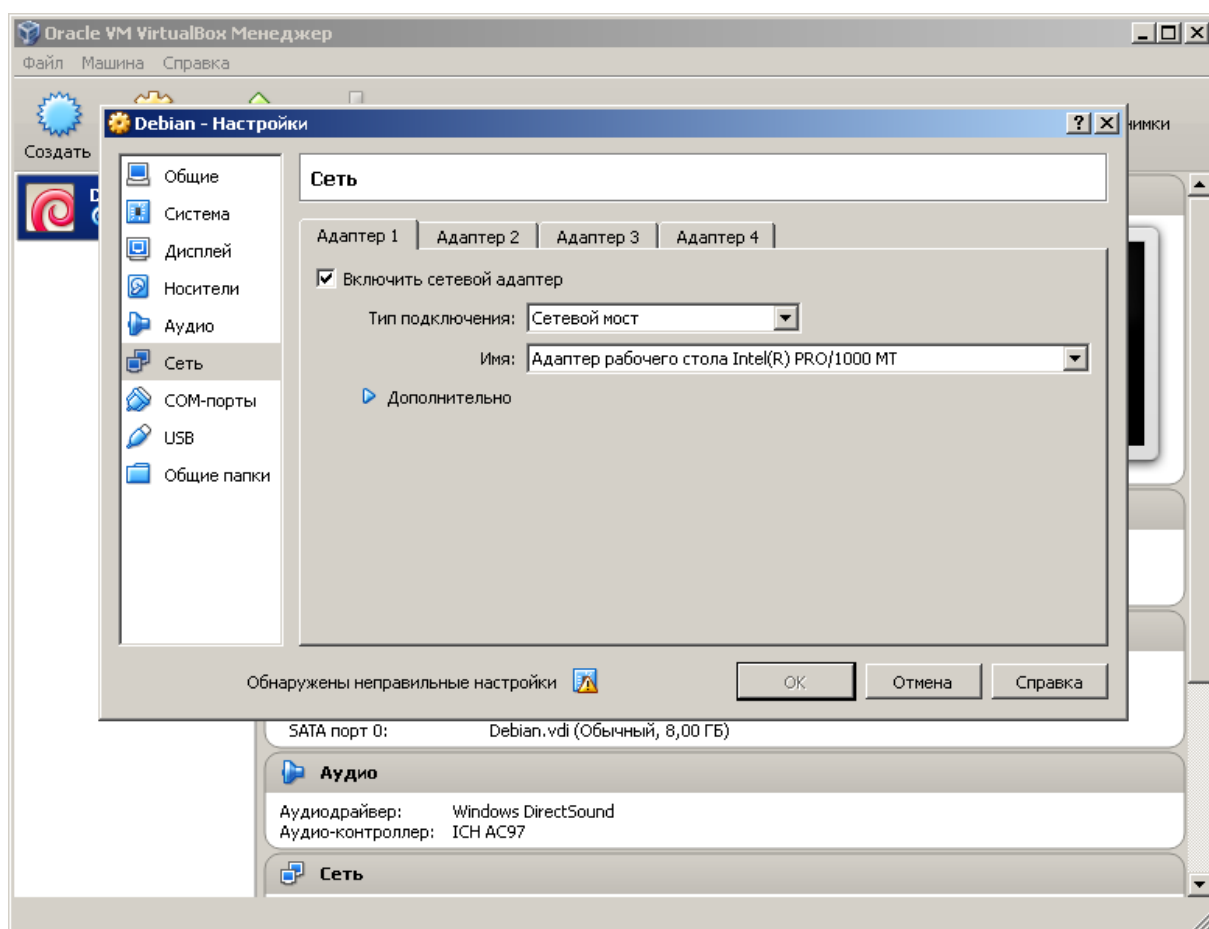




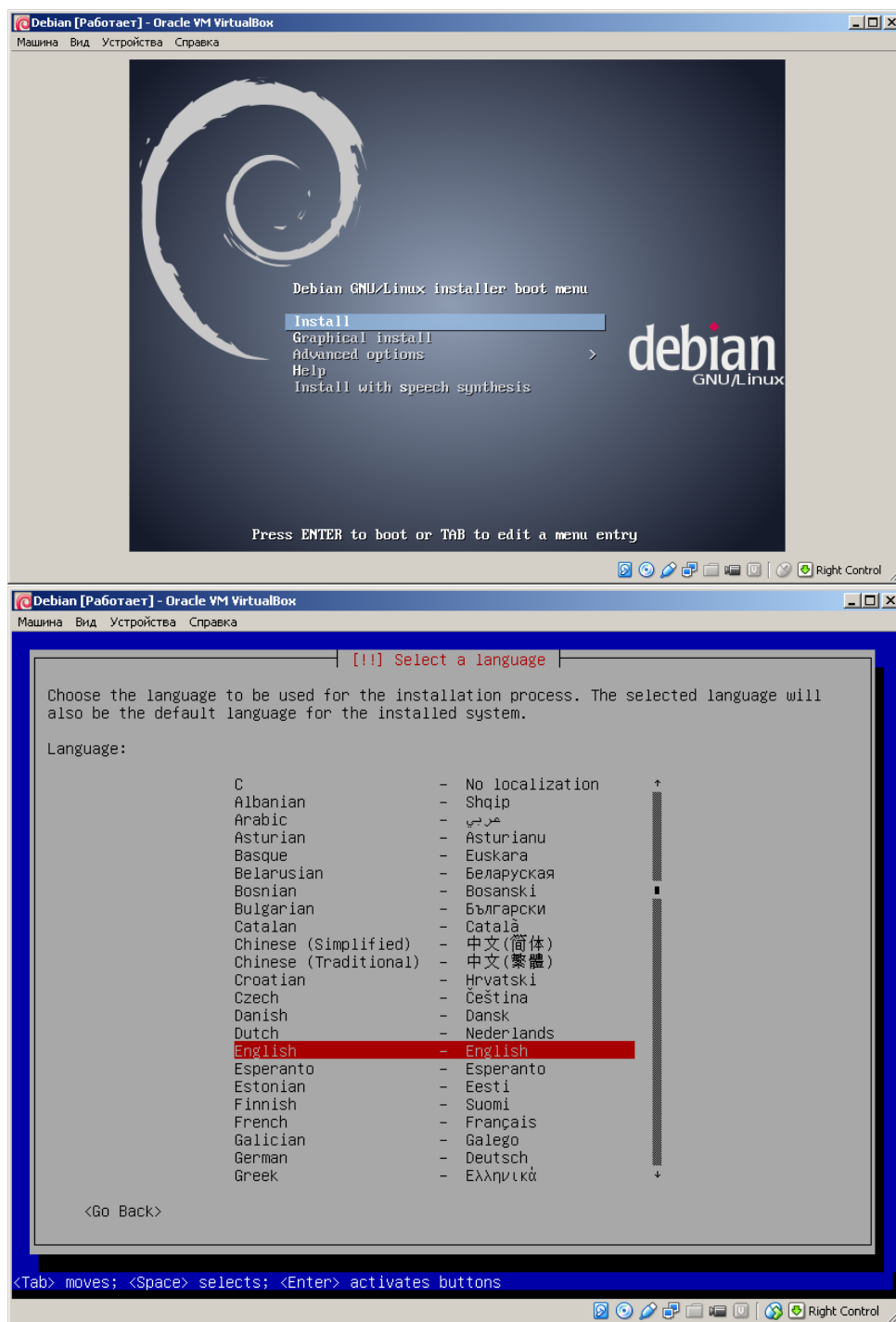


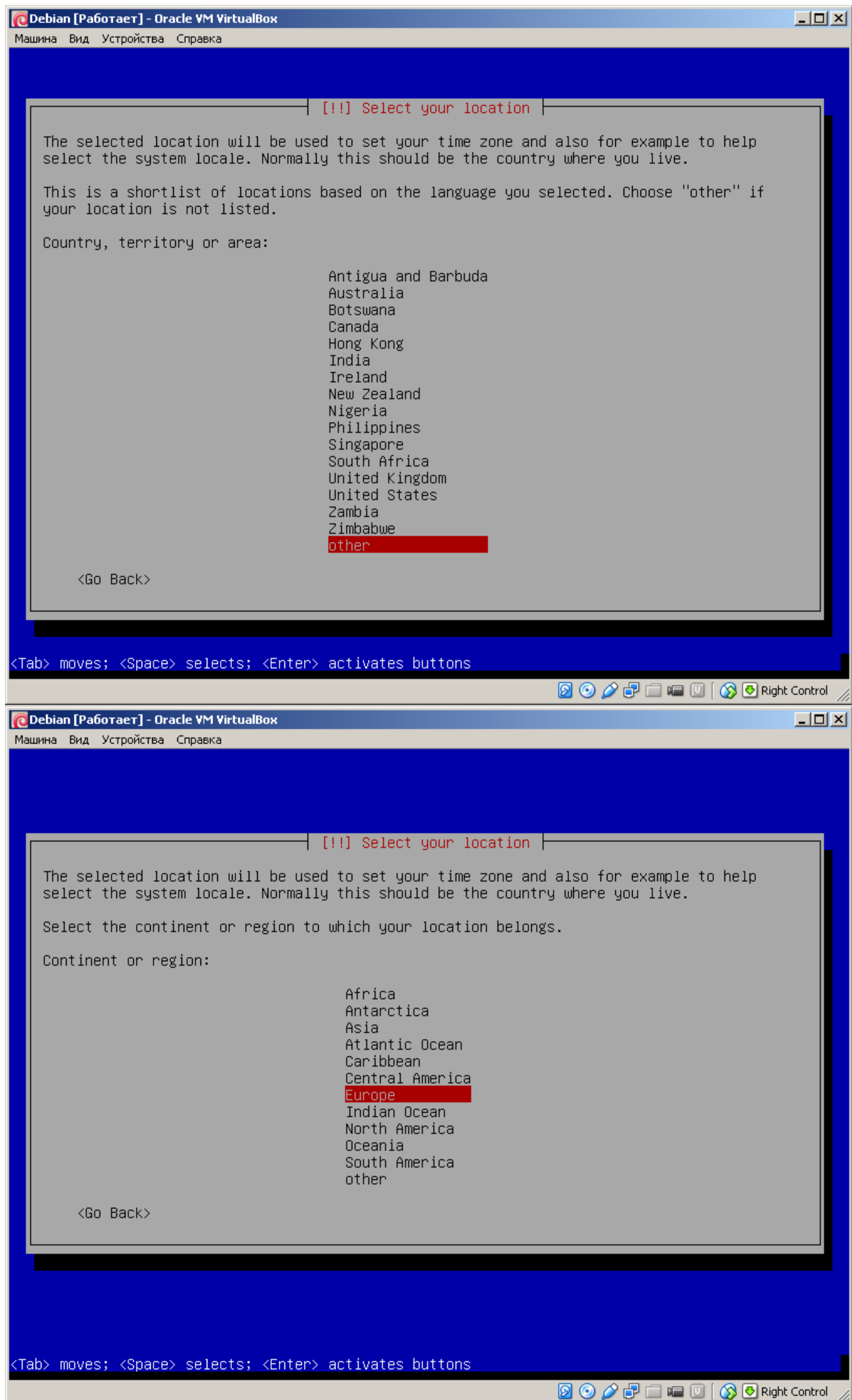


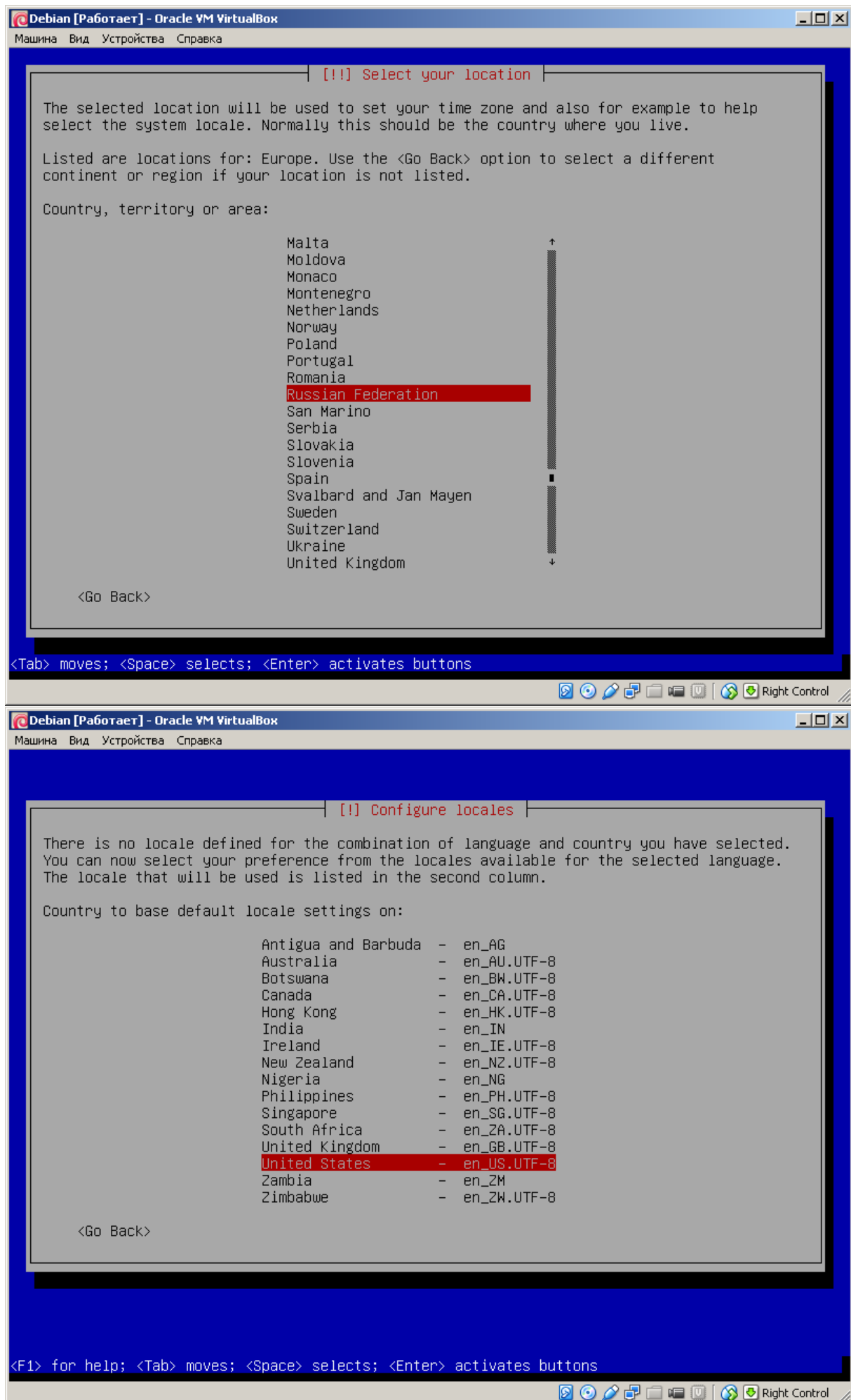


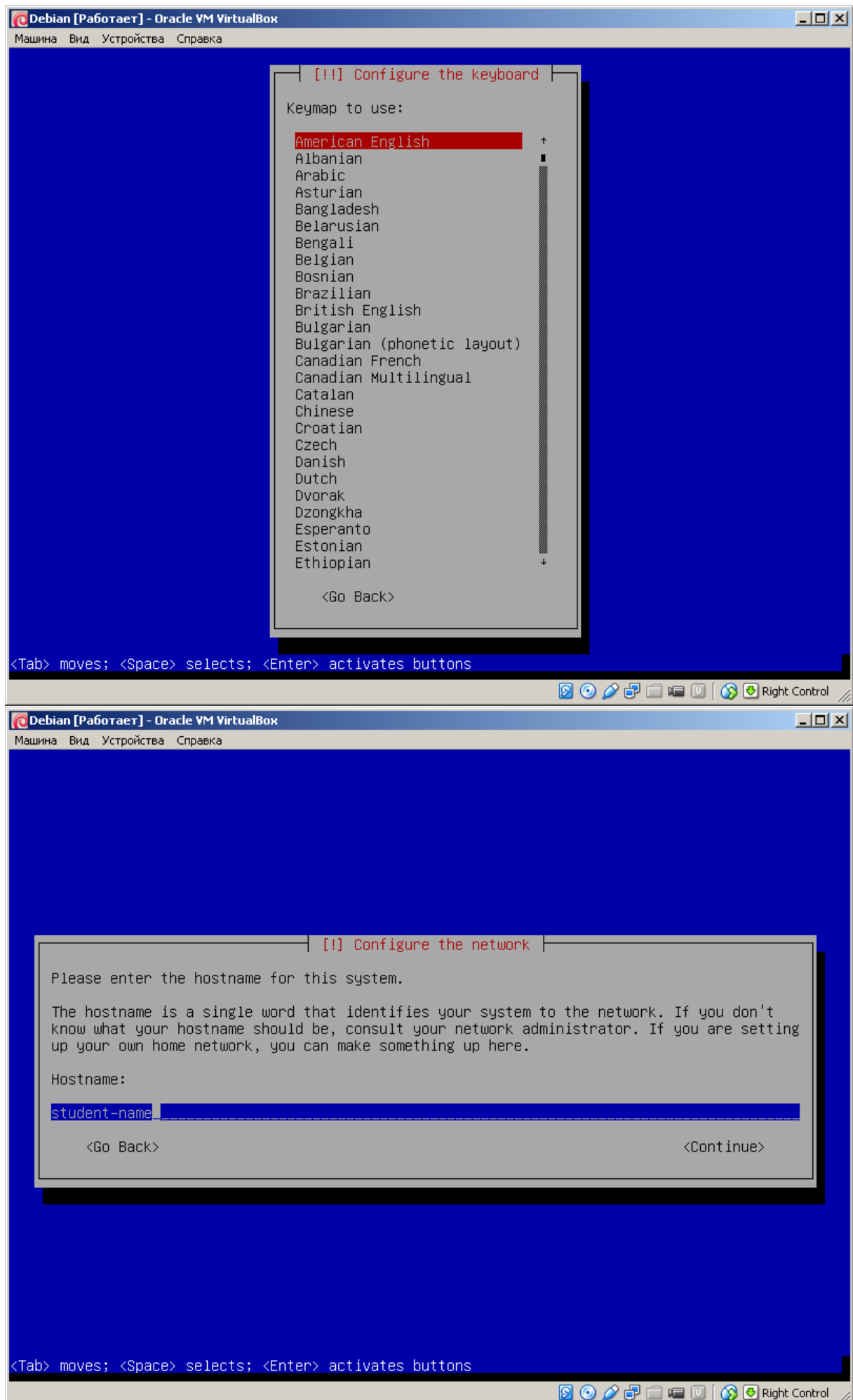


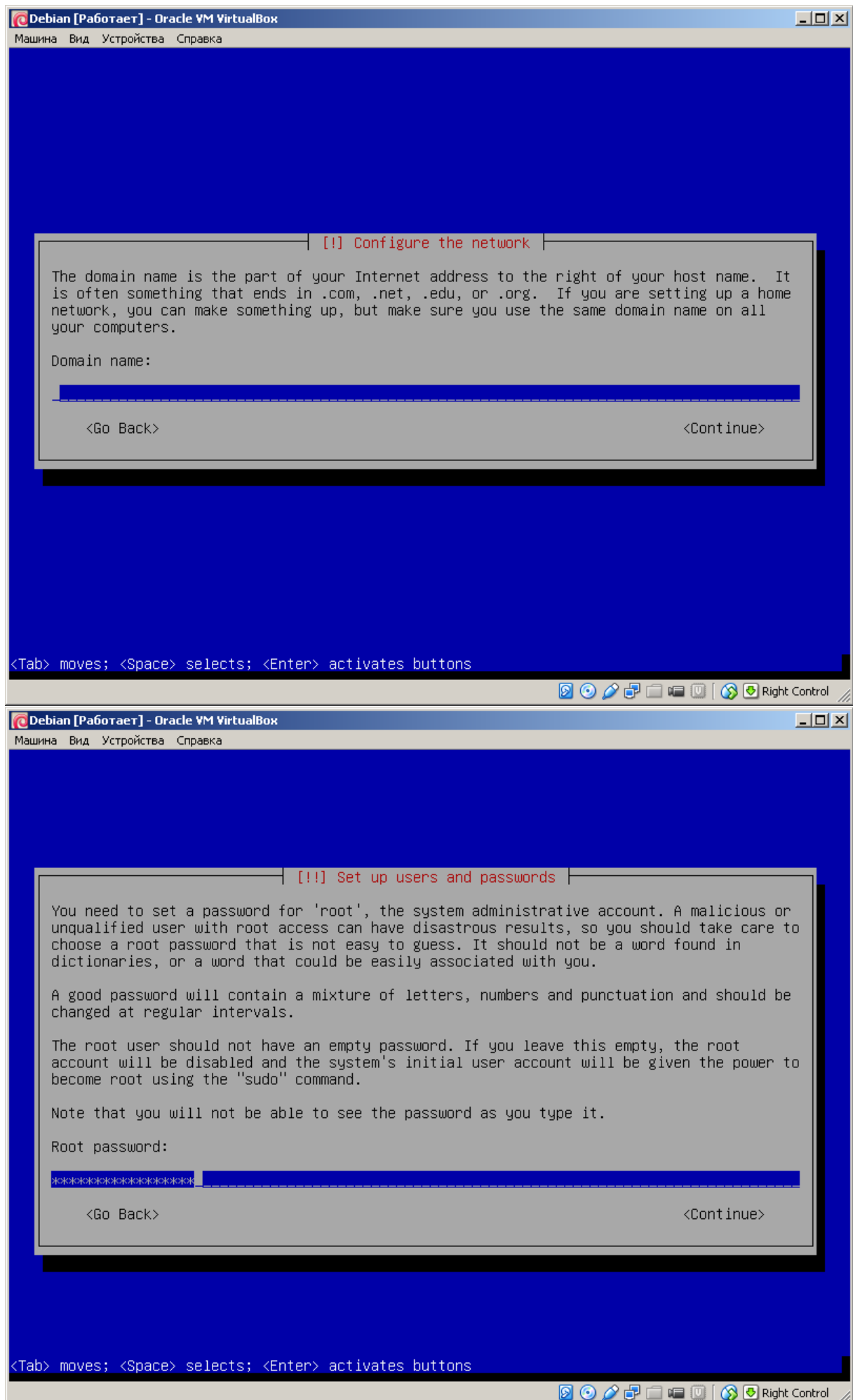
В Установка Debian GNU/Linux в VirtualBox

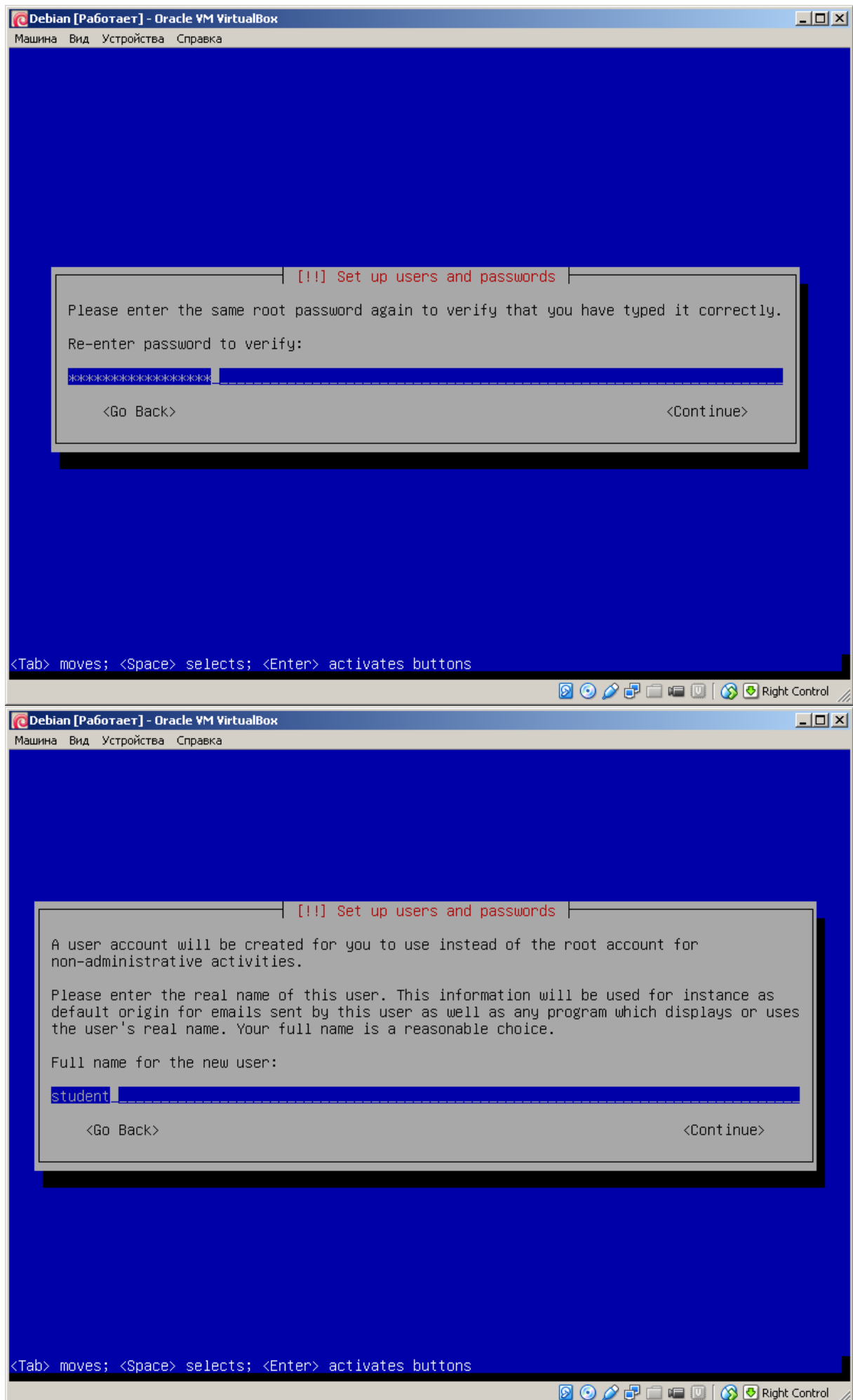


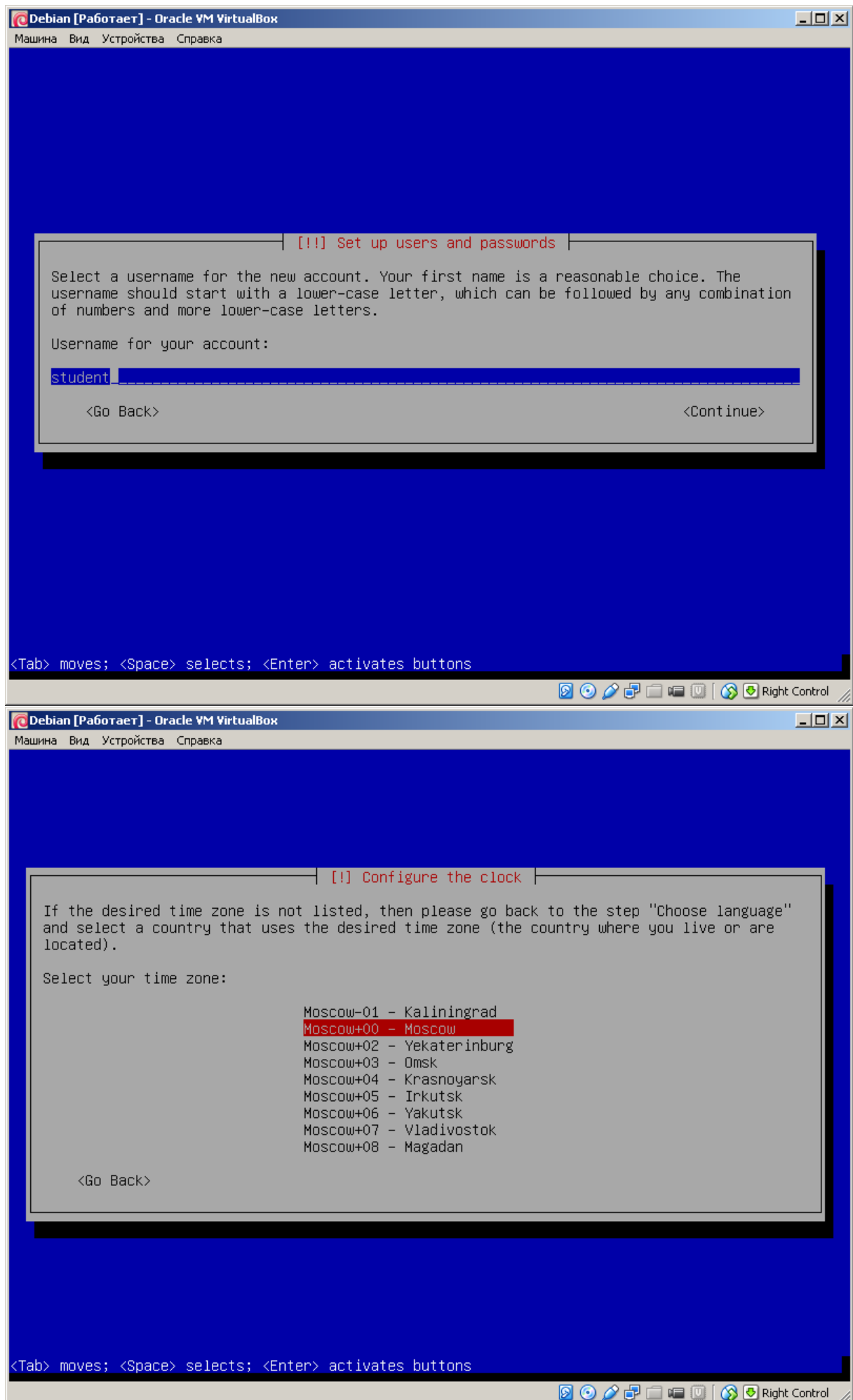


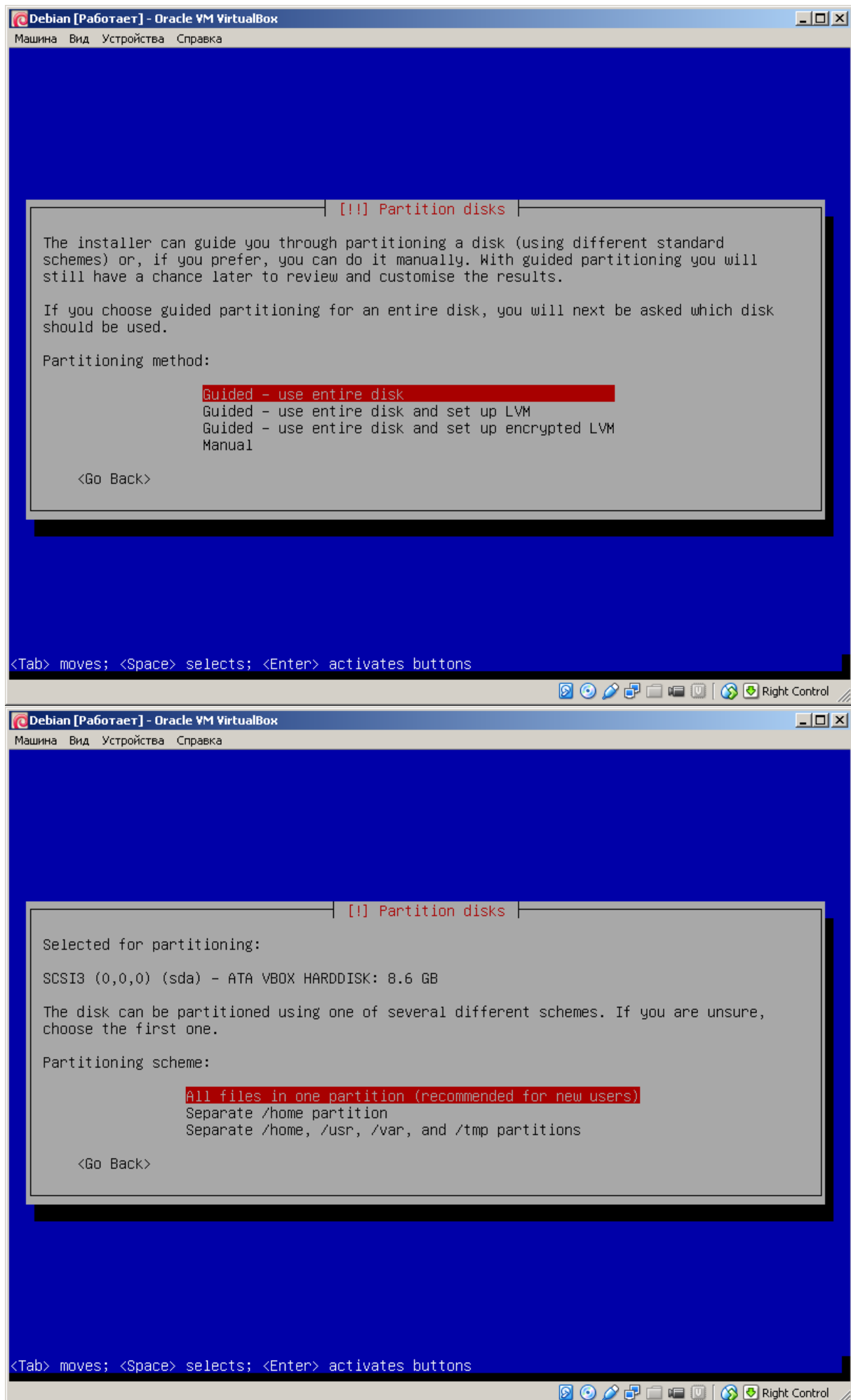


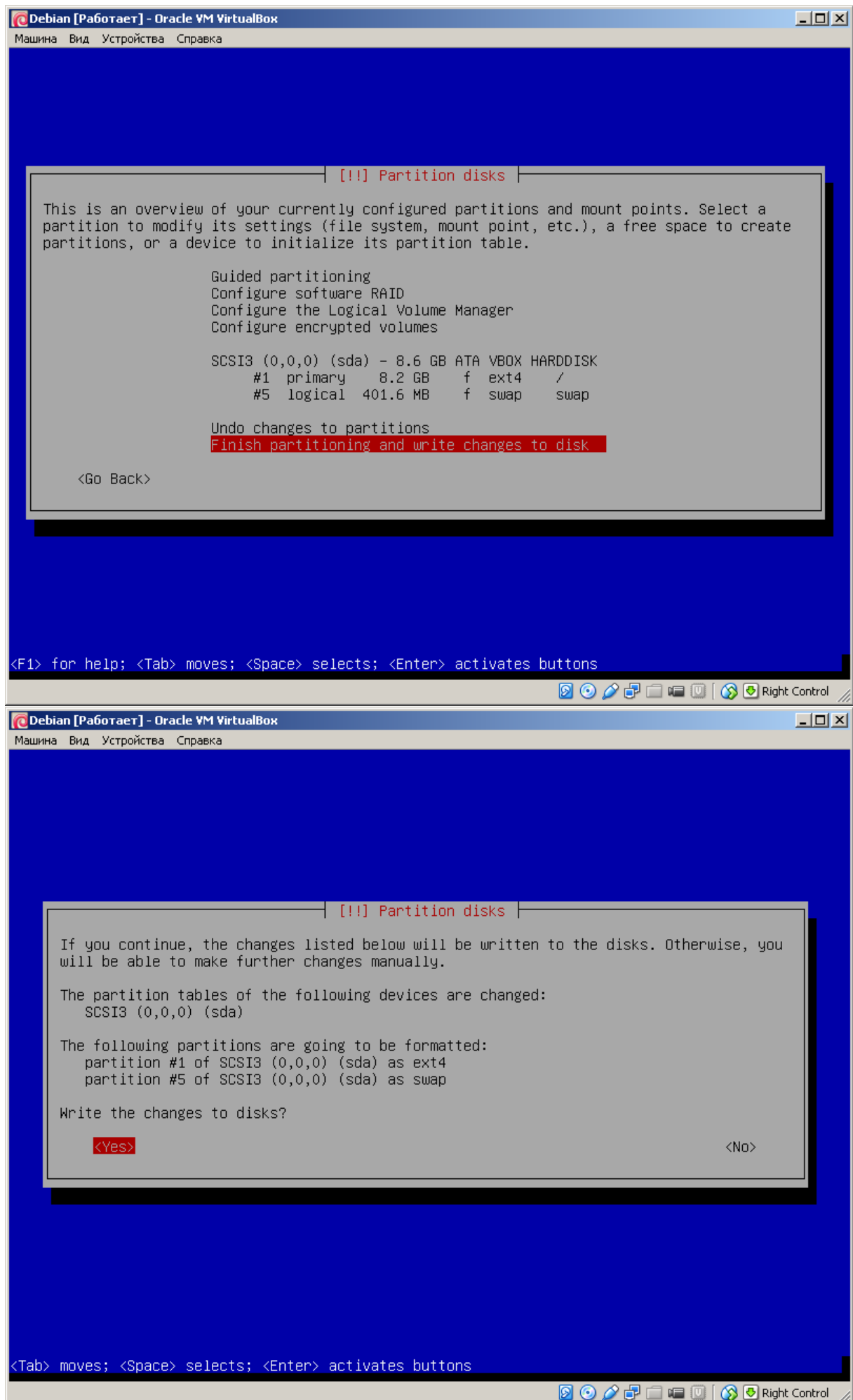


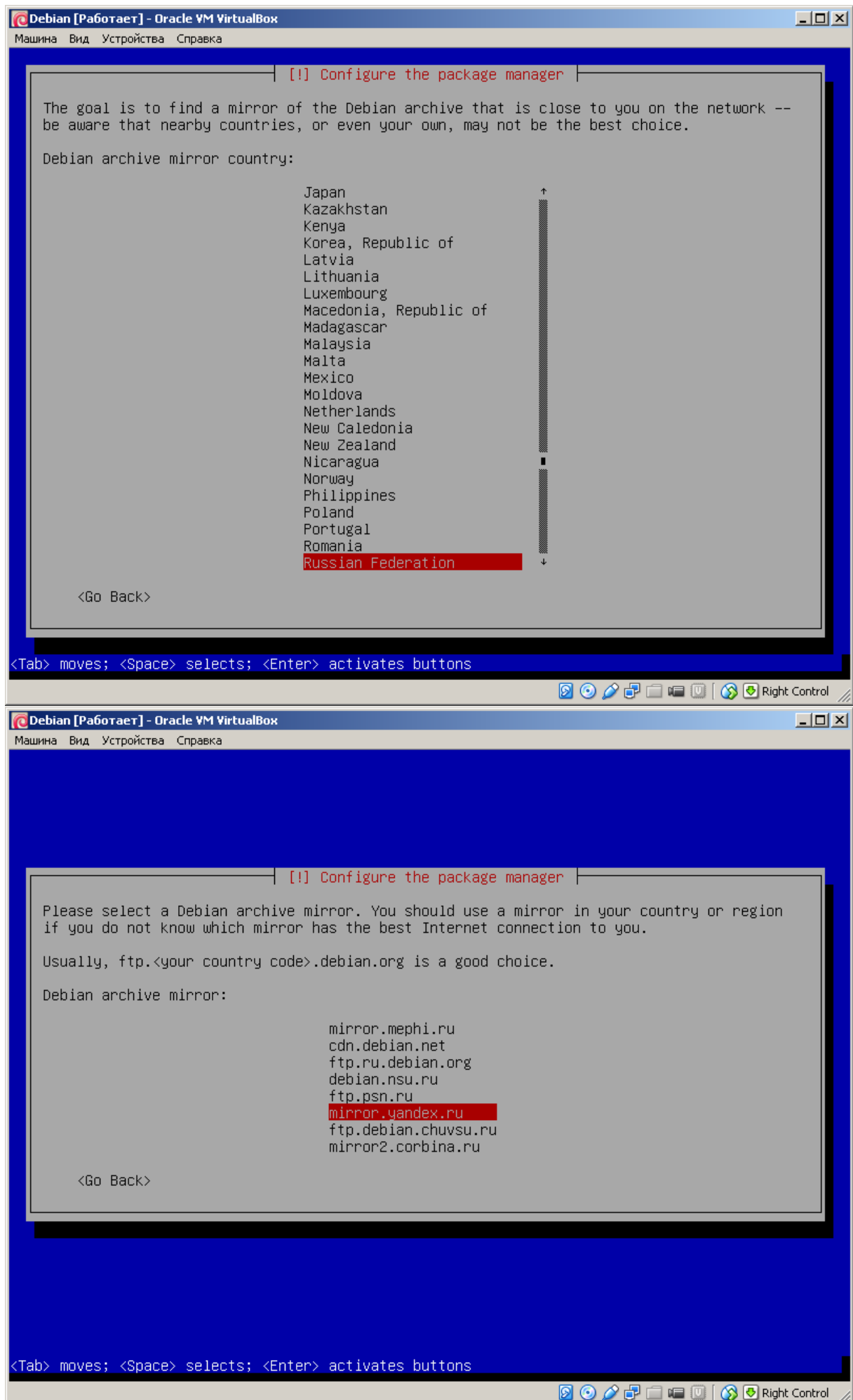


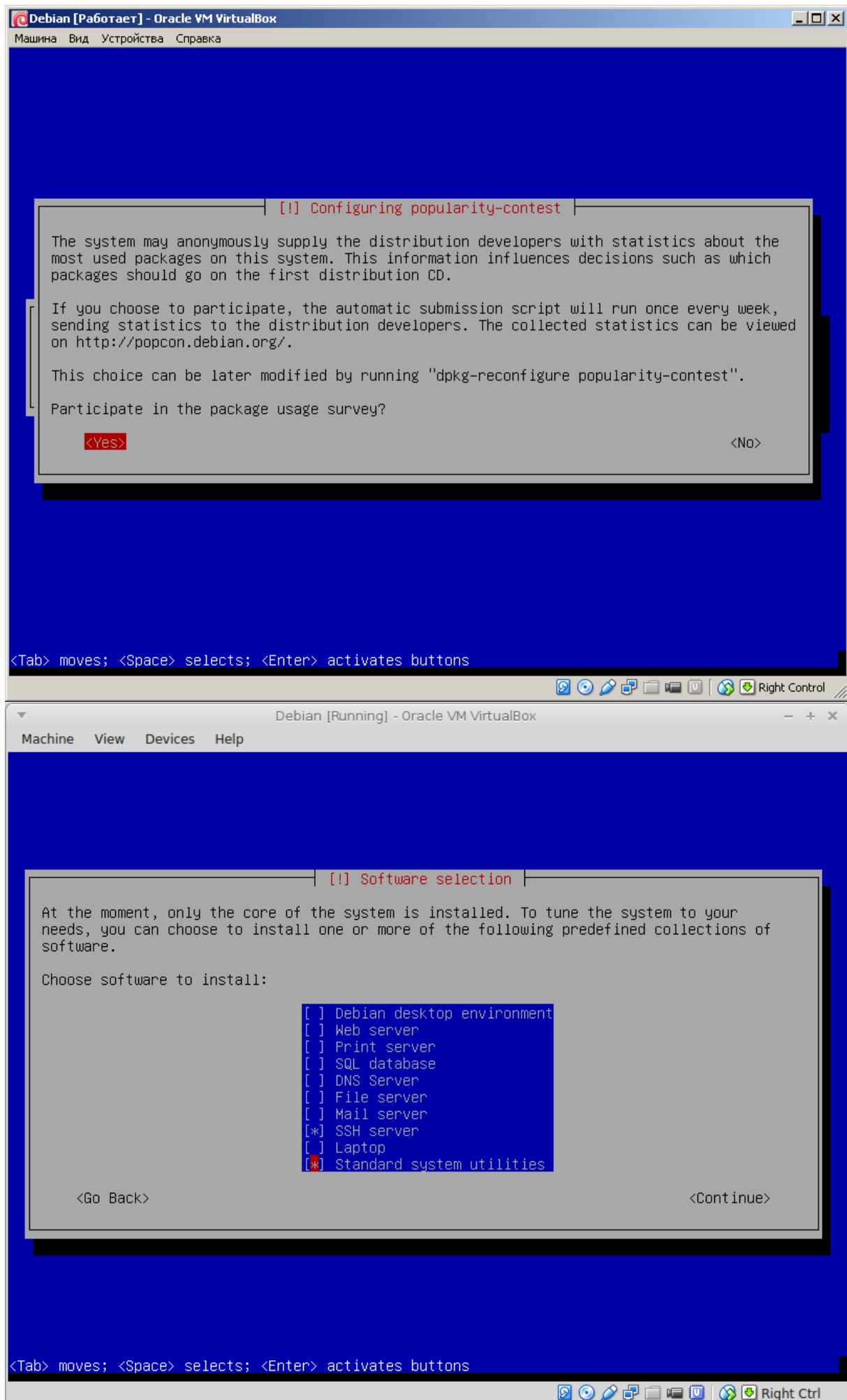














С Установка ownCloud Server в Debian GNU/Linux

После успешного входа в систему, в первую очередь необходимо получить права суперпользователя¹:

```
$ su -  
Password: toor  
# apt-get install sudo  
# usermod -a -G sudo student
```

Добавление репозитория и ключа для ownCloud Server:

```
# wget -nv https://goo.gl/mmV2ga -O Release.key  
# apt-key add - < Release.key  
OK  
# echo deb http://download.owncloud.org\  
> /download/repositories/stable/Debian_8.0/ / > \  
> /etc/apt/sources.list.d/owncloud.list
```

После добавления репозитория необходимо обновить список доступного в репозиториях ПО и запустить установку ownCloud Server (во время установки MySQL, установщик запросит пароль суперпользователя MySQL, он не обязательно должен совпадать с паролем пользователя root):

```
# apt-get update  
# apt-get install owncloud owncloud-files  
...  
New password for the MySQL "root" user: toor-mysql  
Repeat password for the MySQL "root" user: toor-mysql
```

После того, как установщик скачал и установил все необходимые пакеты, можно проверить корректность установки (рис. 7), зайдя по адресу <http://192.168.0.102/owncloud/>, где 192.168.0.102 — IP-адрес сервера (виртуальной машины).

Приложение предлагает использовать базу данных SQLite по умолчанию, мы же будем использовать MySQL:

```
# mysql -uroot -ptoor-mysql  
mysql> CREATE DATABASE owncloud_DB;  
mysql> CREATE USER "owncloud-web"@"localhost" \  
-> IDENTIFIED BY "owncloud-passwd";  
mysql> GRANT ALL PRIVILEGES ON owncloud_DB.* \  
-> TO "owncloud-web"@"localhost";  
mysql> FLUSH PRIVILEGES;  
mysql> quit
```

После создания базы данных, необходимо обновить страницу приложения, настроить параметры базы данных и установить данные аккаунта администратора ownCloud (рис. 8).

После нажатия клавиши «Finish Setup» база данных и пользовательские настройки успешно подключаются к ownCloud (рис. 9).

¹ Пароль пользователя не отображается на экране во время набора

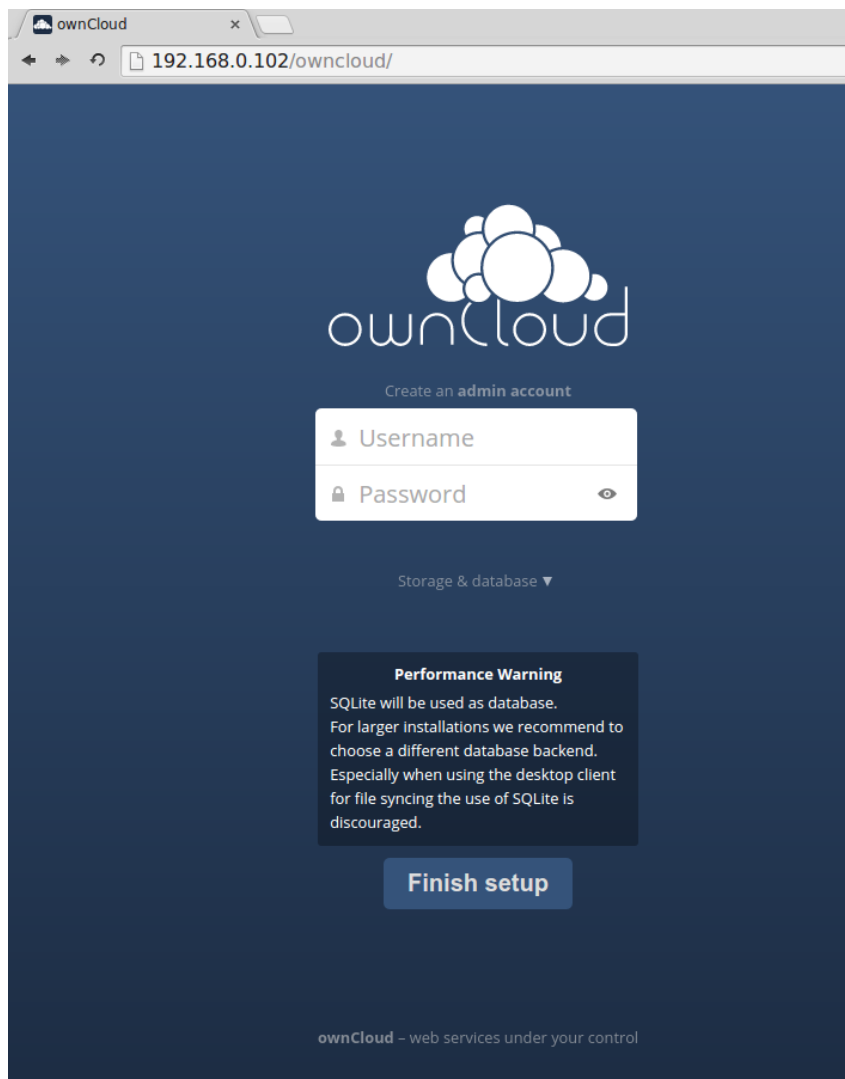



Рис. 7: Первый запуск приложения



Create an admin account

student

student-kvt-

Storage & database ▼

Data folder

/var/www/owncloud/data

Configure the database

SQLite MySQL/MariaDB PostgreSQL

owncloud-web

owncloud-passwd

owncloud-DB

localhost

Finish setup

ownCloud – web services under your control

Рис. 8: Параметры БД и аккаунта администратора

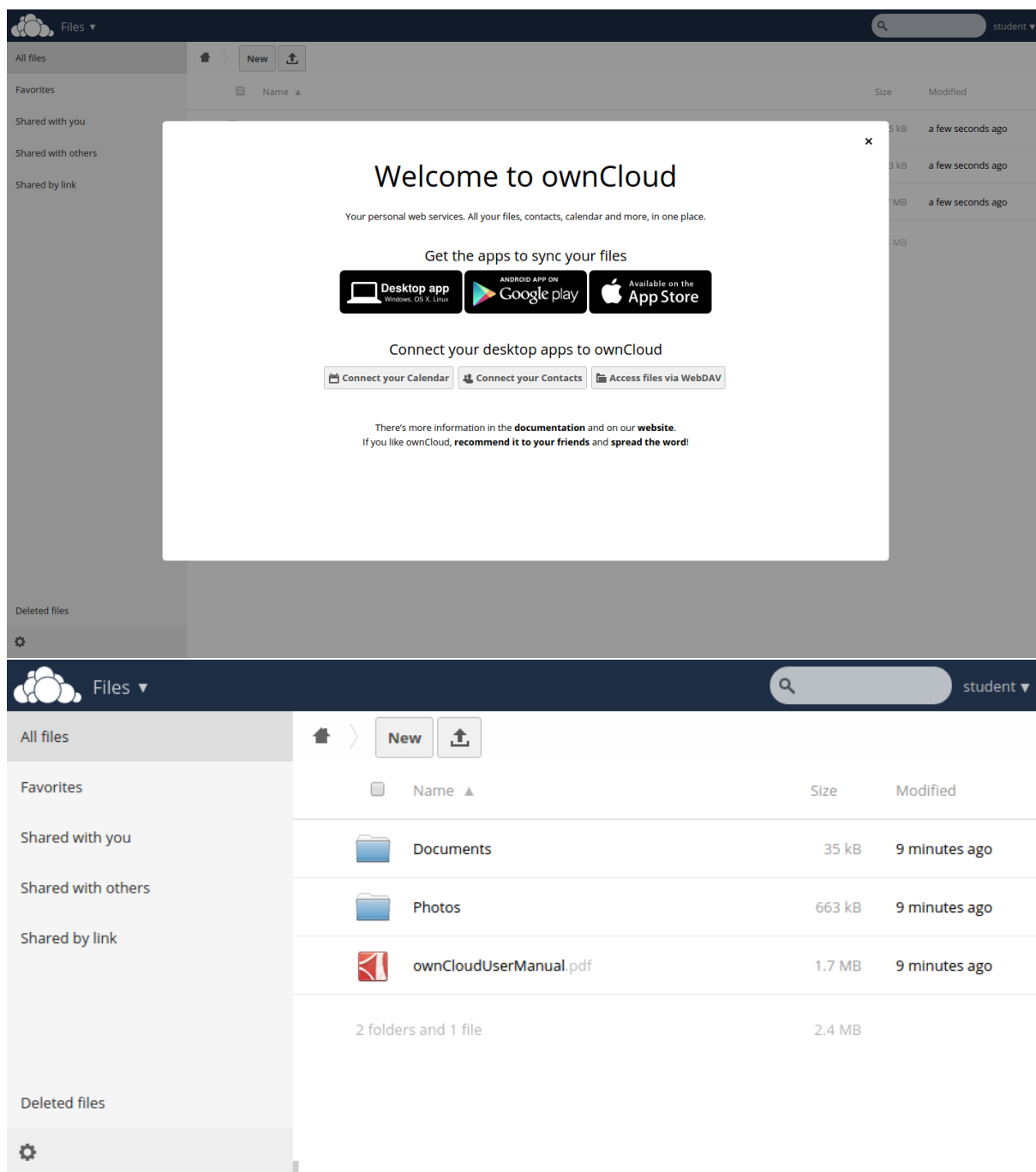
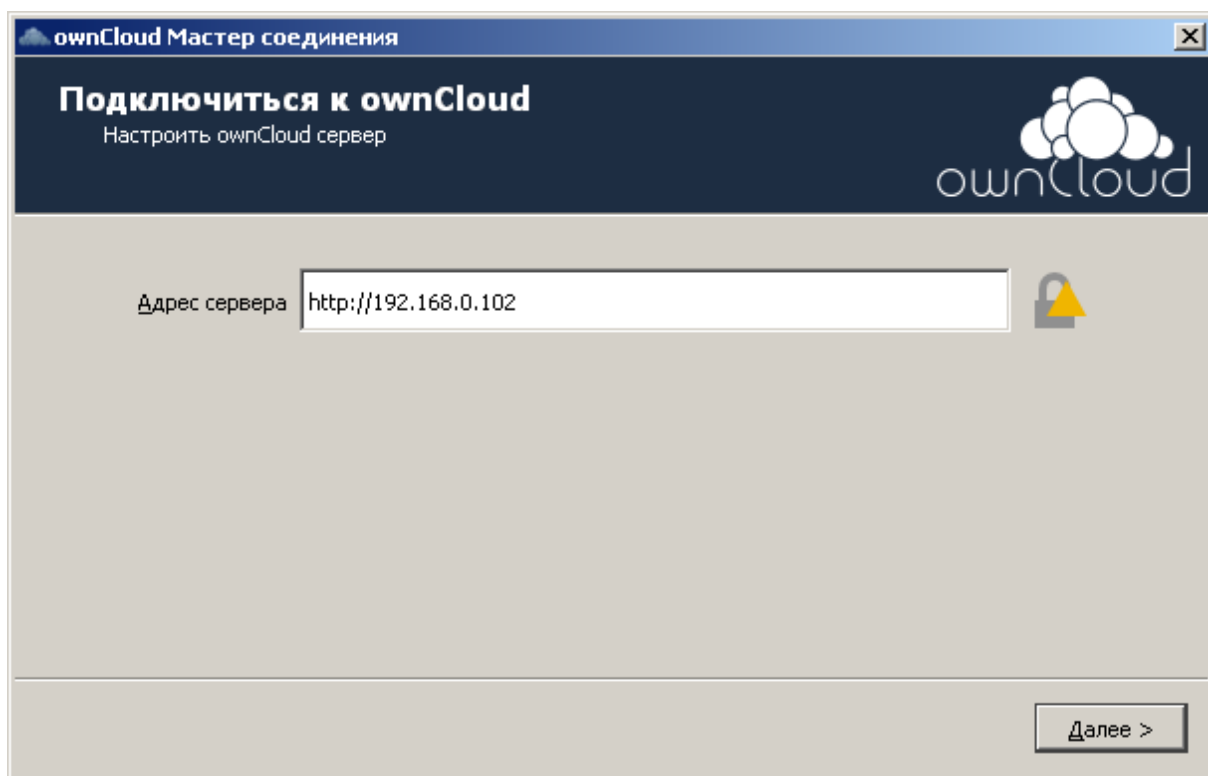
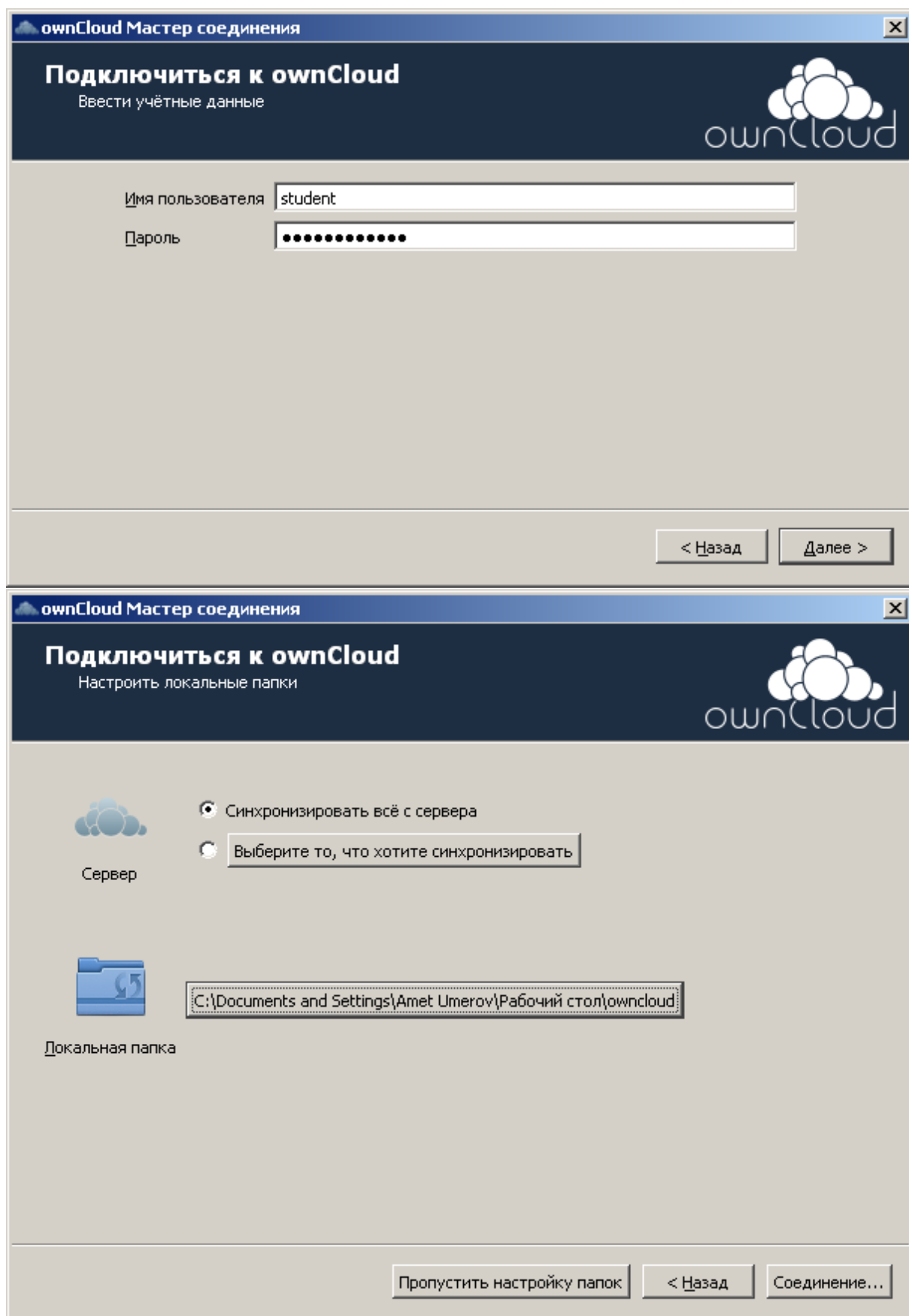
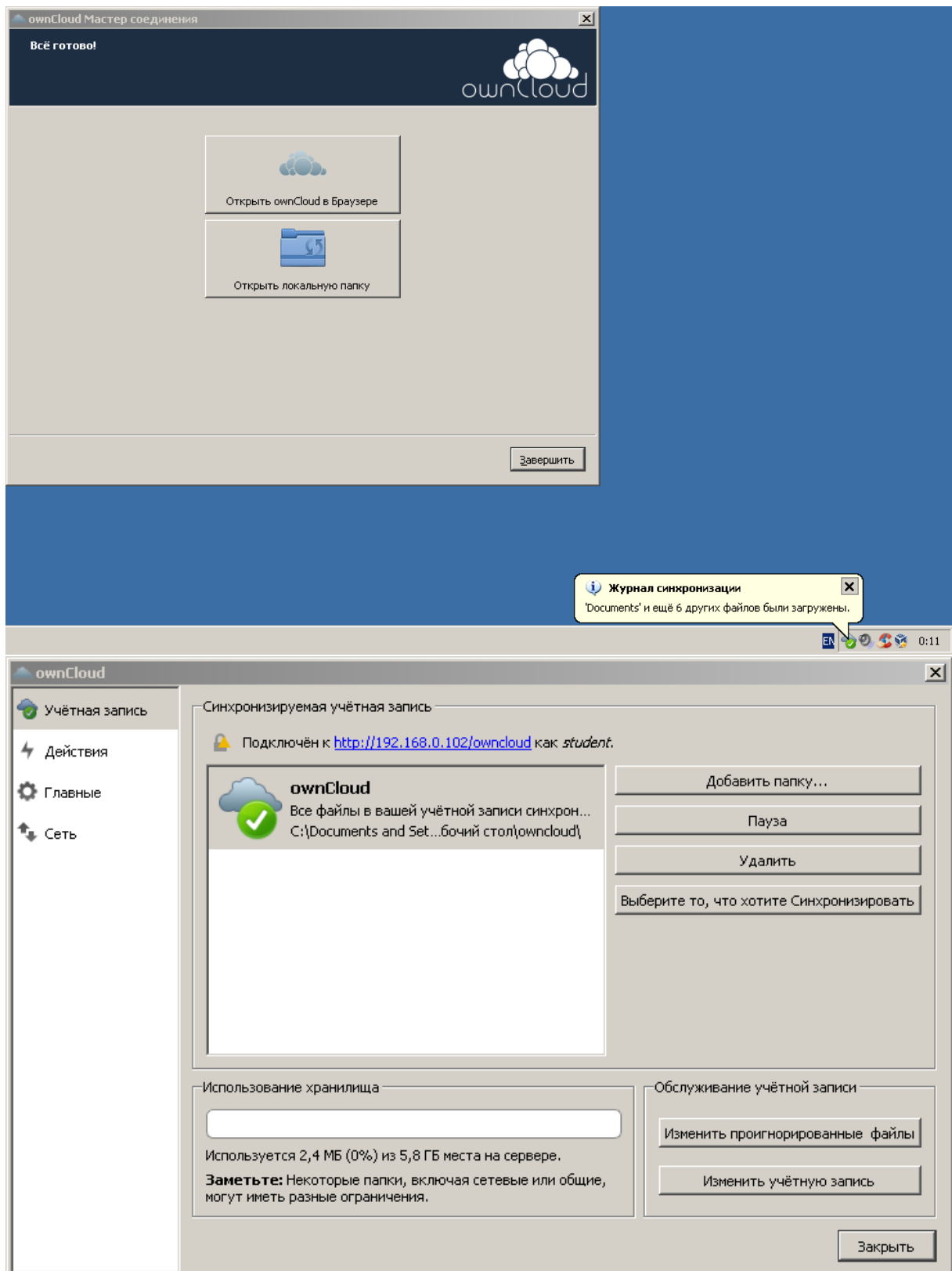


Рис. 9: Первый вход в ownCloud и интерфейс приложения

D Настройка подключения ownCloud Client к серверу







Е Деплой приложения на Heroku

Для регистрации в Heroku¹ необходима только электронная почта (рис. 10).

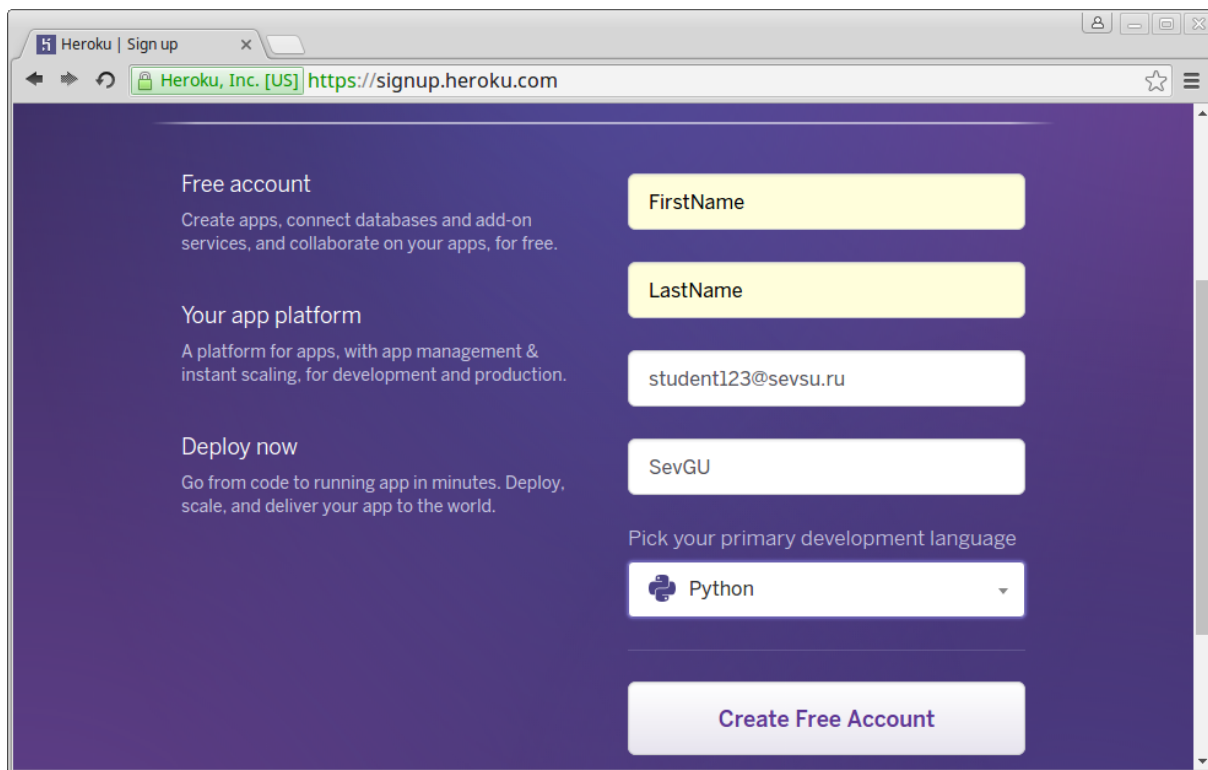


Рис. 10: Регистрация в Heroku

После регистрации, на указанный почтовый ящик приходит письмо с подтверждением регистрации, необходимо перейти по ссылке из письма и подтвердить регистрацию.

После подтверждения регистрации можно войти в свою учетную запись и ознакомиться с интерфейсом платформы.

Создадим новое приложение в разделе «Personal Apps» (рис. 11).

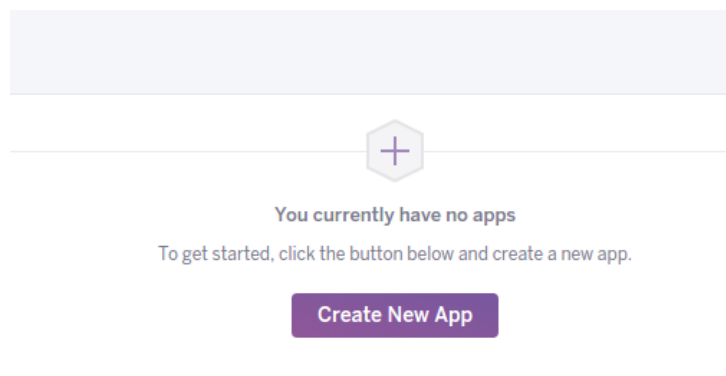


Рис. 11: Список созданных приложений в Heroku

Пусть имя приложения будет «helloivt» (рис. 14).

¹<https://signup.heroku.com/login>

Create New App

App Name (optional)
Leave blank and we'll choose one for you.

helloivt ✓
helloivt is available

Runtime Selection
Your app can run in your choice of region in the Common Runtime.

Europe ⌵

Create App

Рис. 12: Создание нового приложения

Авторизуемся на сервере с Debian, устанавливаем `heroku-cli` и авторизуемся в Heroku:

```
$ wget -O- https://toolbelt.heroku.com/install-ubuntu.sh | sh
$ heroku login
heroku-cli: Installing CLI... 21.83MB/21.83MB
Enter your Heroku credentials.
Email: student123@sevsu.ru
Password (typing will be hidden):
Logged in as student123@sevsu.ru
```

Добавляем контактные данные разработчика:

```
$ git config --global --add user.email "student123@sevsu.ru"
$ git config --global --add user.name "Name Surname"
```

Клонируем репозиторий с тестовым приложением на Flask:

```
$ git clone https://github.com/craigkerstiens/flask-helloworld
$ cd flask-helloworld/
```

Реинициализируем репозиторий:

```
$ git init
Reinitialized existing Git repository in /home/student/flask-
helloworld/.git/
$ heroku git:remote -a helloivt
set git remote heroku to https://git.heroku.com/helloivt.git
```

Деплоим приложение в Heroku:

```
$ git push heroku master
```

Переходим на страницу приложения¹ и проверяем, страница должна выводить надпись «Hello from Python!» (рис. 13).

¹<https://helloivt.herokuapp.com/>, где «helloivt» — это имя приложения, заданное в веб-интерфейсе Heroku

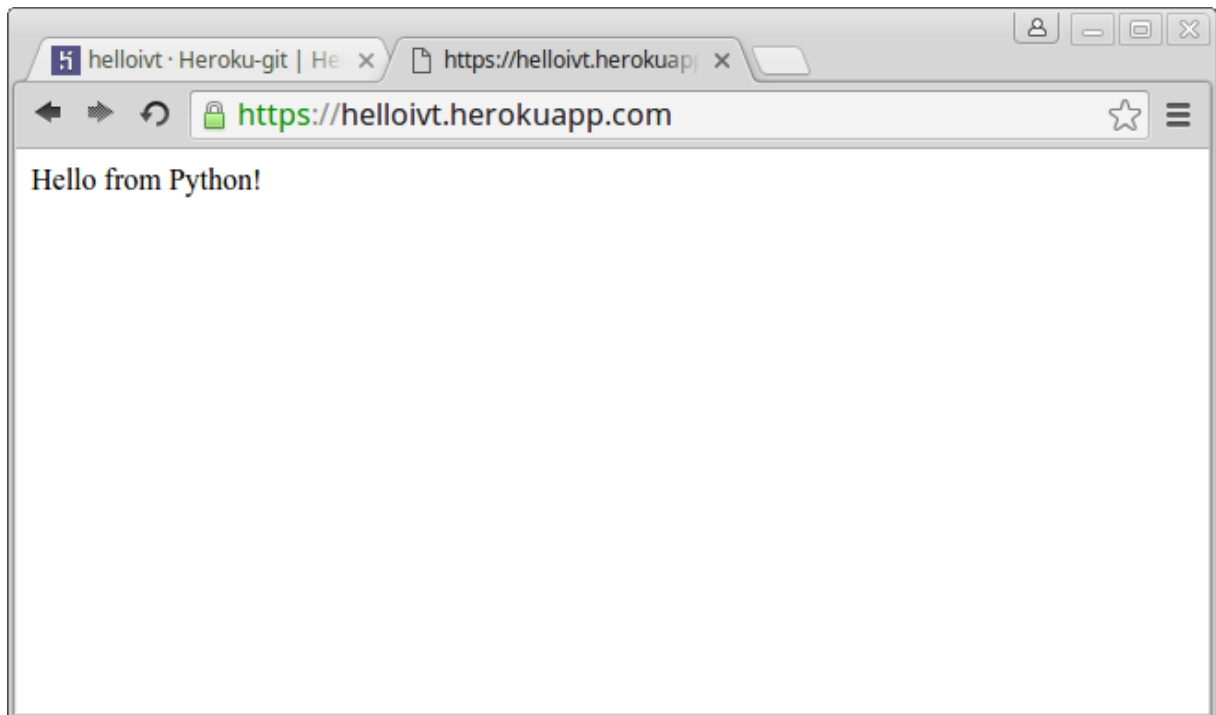


Рис. 13: Создание нового приложения

Внесем изменения в приложение, отредактировав приветствие:

```
$ nano app.py
    return "Hello from IVT student!"
$ git commit -am "First commit!"
[master 898f7d8] First commit!
1 file changed, 1 insertion(+), 1 deletion(-)
$ git push heroku master
```

Обновляем страницу с приложением (рис. 14).

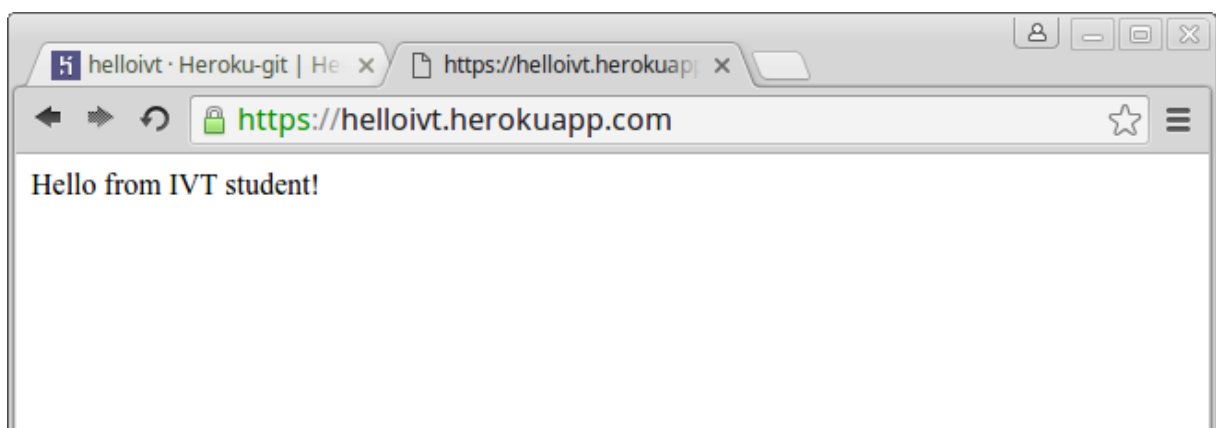


Рис. 14: Изменения в приложении вступили в силу

F Создание инстанса в OpenStack

Сервис TryStack¹ позволяет использовать уже готовое окружение OpenStack без развертывания сложной инфраструктуры на локальных серверах.

Для регистрации на сервисе необходимо иметь аккаунт в Facebook и вступить в группу TryStack². Ведется разработка OpenStackID для авторизации на сервисе с помощью OAuth2, однако регистрация возможна только через Facebook.

После отправки заявки на вступление в закрытую группу возможно придется подождать несколько дней подтверждения заявки. Когда заявка подтверждена, то можно авторизоваться в панели³ TryStack с помощью аккаунта Facebook.

После авторизации можно наблюдать интерфейс управления OpenStack (рис. 15).

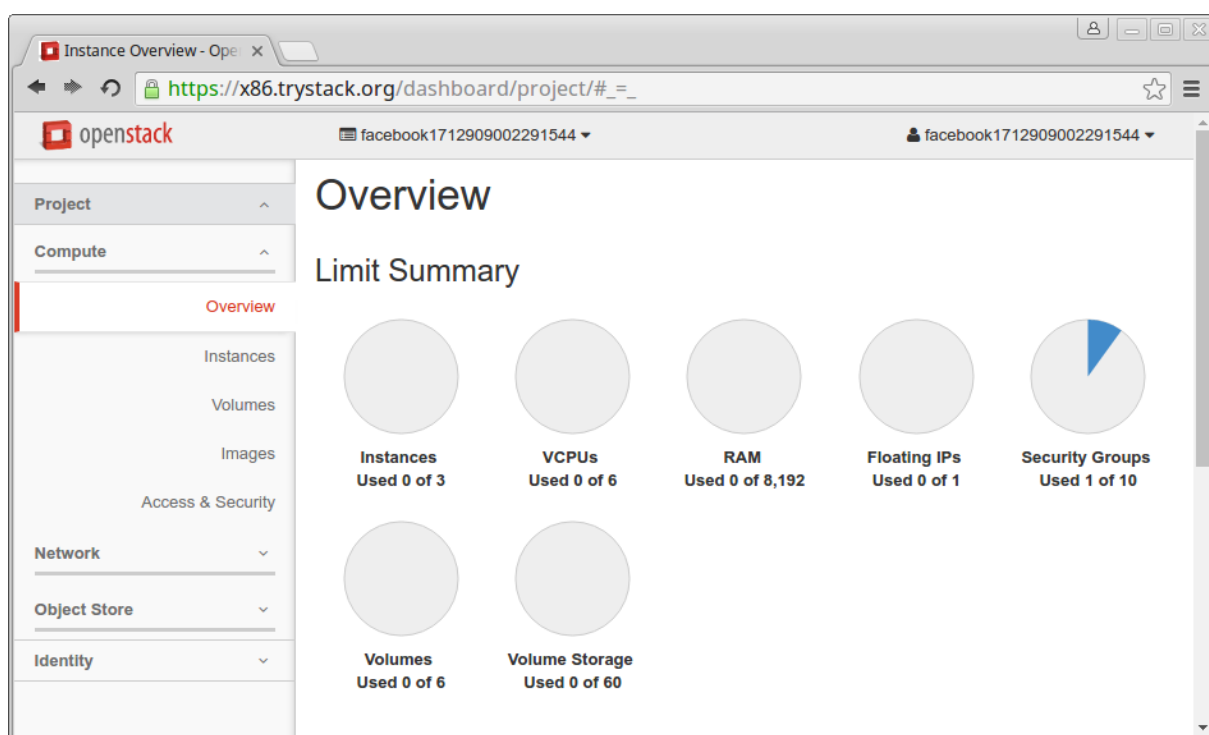


Рис. 15: Интерфейс управления OpenStack

OpenStack позволяет создавать множество инстансов из готовых шаблонов ОС, также существует возможность создавать свои образы. Доступные шаблоны ОС можно наблюдать в разделе **Compute - Images**.

Для настройки OpenStack в первую очередь необходимо настроить сеть.

Создадим внутреннюю сеть с именем student-net, подсеть student-subnet (192.168.0.0/24), укажем пул используемых IP-адресов (192.168.0.2 – 192.168.0.10) и укажем DNS-сервера (8.8.8.8, 8.8.4.4) в разделе **Network - Networks - Create Network** (рис. 16).

¹<https://trystack.openstack.org>

²<https://www.facebook.com/groups/269238013145112>

³<https://x86.trystack.org/dashboard/auth/login/?next=/dashboard/>

Create Network ✕

Network

Subnet

Subnet Details

Network Name

Admin State ?

UP

☒ Create Subnet

Create a new network. In addition, a subnet associated with the network can be created in the next panel.

Create Network ✕

Network

Subnet

Subnet Details

Subnet Name

Network Address ?

IP Version

IPv4

Gateway IP ?

☐ Disable Gateway

Create a subnet associated with the network. Advanced configuration is available by clicking on the "Subnet Details" tab.

Create Network ✕

Network

Subnet

Subnet Details

☒ Enable DHCP

Allocation Pools ?

DNS Name Servers ?

Specify additional attributes for the subnet.

Рис. 16: Настройка внутренней сети

После создания внутренней сети, необходимо создать маршрутизатор, который соединит внешнюю сеть (public) с внутренней (student-net), сделать это можно в разделе **Network - Routers - Create Router** (рис. 17).

Create Router ✕

Router Name *

Description:
Creates a router with specified parameters.

Admin State

UP ▼

External Network

public ▼

Рис. 17: Создание маршрутизатора

Текущую топологию сети можно увидеть в разделе **Network - Network Topology** (рис. 18). На схеме видно, что маршрутизатор (student-router) связан с внешней сетью (public), но не с внутренней (student-net).

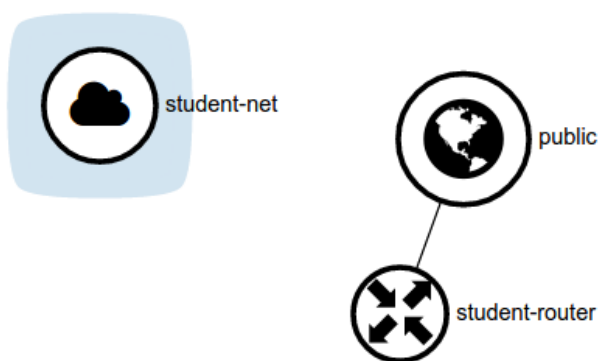
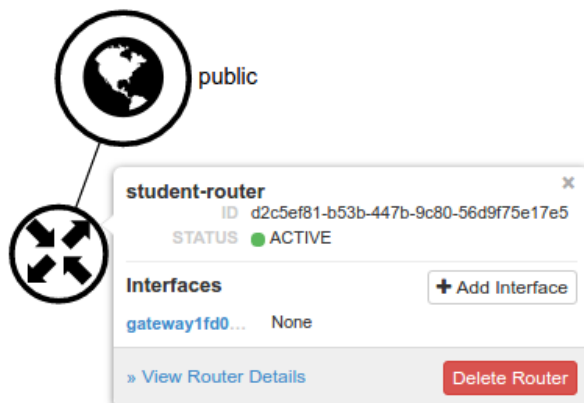


Рис. 18: Текущая топология сети

Соединяем маршрутизатор с внутренней сетью. Щелкаем на маршрутизатор и выбираем пункт **Add Interface**, добавляем внутреннюю сеть (student-net) к маршрутизатору, в качестве шлюза указываем адрес 192.168.0.1 (рис. 19).

После этого можно посмотреть на новую топологию сети (рис. 20). Видно что теперь маршрутизатор соединяет внутреннюю и внешнюю сети.

На этом настройка сети для OpenStack окончена.



Add Interface

Subnet *

student-net: 192.168.0.0/24 (student-subnet) ▼

IP Address (optional) ⓘ

192.168.0.1

Router Name *

student-router

Router ID *

d2c5ef81-b53b-447b-9c80-56d9f75e17e5

Description:

You can connect a specified subnet to the router.

The default IP address of the interface created is a gateway of the selected subnet. You can specify another IP address of the interface here. You must select a subnet to which the specified IP address belongs to from the above list.

Рис. 19: Подключение внутренней сети к маршрутизатору

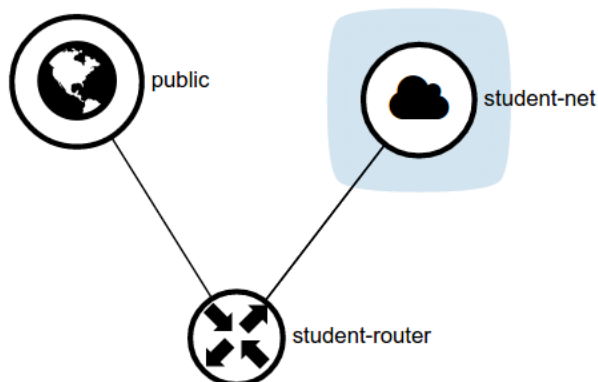


Рис. 20: Маршрутизатор соединяет внешнюю и внутреннюю сеть

По умолчанию для подключения к инстансам используется подключение по SSH с помощью шифрованных ключей. Поэтому необходимо создать пару ключей, сделать это можно в разделе **Access & Security - Key Pairs - Create Key Pair**. Пусть это будет пара ключей с именем student123.

После создания пары ключей, автоматически скачается .pem файл, который нужно будет использовать для соединения к инстансам.

Для соединения к инстансу с внешнего мира, необходимо иметь выделенный внешний IP-адрес. В TryStack можно получить только один такой адрес, сделать это можно в разделе **Compute - Access & Security - Floating IPs - Allocate IP To Project**. В некоторых случаях возможно, что выделить внешний IP-адрес не удастся, в таком случае необходимо попробовать позже, предварительно обновив веб-страницу, это связано с ограниченным пулом IP-адресов, поэтому для всех пользователей TryStack таких адресов может не хватать.

После получения внешнего адреса, можно увидеть в списке (рис. 21).

Access & Security

Security Groups

Key Pairs

Floating IPs

API Access

🔗 Allocate IP To Project (Quota exceeded)

🔗 Release Floating IPs

<input type="checkbox"/>	IP Address	Mapped Fixed IP Address	Pool	Status	Actions
<input type="checkbox"/>	8.43.86.56	-	public	Down	Associate ▼

Displaying 1 item

Рис. 21: Список выделенных внешних IP-адресов

Далее необходимо создать группу безопасности, в которой можно настроить правила фильтрации трафика для инстанса. В разделе **Compute - Access & Security - Security Groups - Create Security Group** создадим группу my-secgroup.

После создания группы, в общем списке выбираем пункт **Manage Rules** для my-secgroup. Там можем видеть, что по умолчанию для данной группы открыты все исходящие соединения для IPv4 и IPv6. Создадим разрешающее правило для входящих соединений по протоколу ICMP (рис. 22).

Аналогично создадим разрешающие правила для TCP и UDP (рис. 23).

Add Rule

Rule *

ALL ICMP

Direction

Ingress

Remote * ?

CIDR

CIDR ?

0.0.0.0/0

Description:

Rules define which traffic is allowed to instances assigned to the security group. A security group rule consists of three main parts:

Rule: You can specify the desired rule template or use custom rules, the options are Custom TCP Rule, Custom UDP Rule, or Custom ICMP Rule.

Open Port/Port Range: For TCP and UDP rules you may choose to open either a single port or a range of ports. Selecting the "Port Range" option will provide you with space to provide both the starting and ending ports for the range. For ICMP rules you instead specify an ICMP type and code in the spaces provided.

Remote: You must specify the source of the traffic to be allowed via this rule. You may do so either in the form of an IP address block (CIDR) or via a source group (Security Group). Selecting a security group as the source will allow any other instance in that security group access to any other instance via this rule.

Рис. 22: Разрешающее правило для работы ICMP

Manage Security Group Rules: my-secgroup (b679d00a-1db9-4b6b-b548-2eb526f7402d)

+ Add Rule

x Delete Rules

<input type="checkbox"/>	Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Actions
<input type="checkbox"/>	Egress	IPv6	Any	Any	::/0	-	<div>Delete Rule</div>
<input type="checkbox"/>	Egress	IPv4	Any	Any	0.0.0.0/0	-	<div>Delete Rule</div>
<input type="checkbox"/>	Ingress	IPv4	ICMP	Any	0.0.0.0/0	-	<div>Delete Rule</div>
<input type="checkbox"/>	Ingress	IPv4	TCP	1 - 65535	0.0.0.0/0	-	<div>Delete Rule</div>
<input type="checkbox"/>	Ingress	IPv4	UDP	1 - 65535	0.0.0.0/0	-	<div>Delete Rule</div>

Displaying 5 items

Рис. 23: Разрешающие правила для группы безопасности my-secgroup

Далее мы можем создать первый инстанс (**Compute - Instances - Launch Instance - Details**). Создадим его на базе ОС Ubuntu 16.04, размер инстанса m1.small (2048MB RAM, 20GB HDD). Имя инстанса student-instance (рис. 24).
В разделе **Access & Security** укажем пару ключей student123 и группу безопасности my-secgroup. В разделе **Networking** выберем сеть student-net.

Launch Instance

Details *

Access & Security

Networking *

Post-Creation

Advanced Options

Availability Zone

nova

Instance Name *

student-instance

Flavor * ?

m1.small

Instance Count * ?

1

Instance Boot Source * ?

Boot from image

Image Name *

Ubuntu16.04 (289.3 MB)

Specify the details for launching an instance.

The chart below shows the resources used by this project in relation to the project's quotas.

Flavor Details

Name	m1.small
VCPUs	1
Root Disk	20 GB
Ephemeral Disk	0 GB
Total Disk	20 GB
RAM	2,048 MB

Project Limits

Number of Instances

0 of 3 Used

Number of VCPUs

0 of 6 Used

Total RAM

0 of 8,192 MB Used

Рис. 24: Создание нового инстанса

После этого можно видеть запущенный инстанс в общем списке (рис. 25), ему присвоился внутренний адрес 192.168.0.3.

Instances

Instance Name

Filter

Filter

Launch Instance

Terminate Instances

More Actions

	Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
<input type="checkbox"/>	student-instance	Ubuntu16.04	192.168.0.3	m1.small	student123	Active	nova	None	Running	0 minutes	Create Snapshot

Displaying 1 item

Рис. 25: Список созданных инстансов

Для того, чтобы получить доступ к инстансу с внешней сети, необходимо привязать к инстансу внешний IP-адрес. В разделе **Compute - Access & Security - Floating IPs** напротив IP-адреса нажимаем кнопку **Associate**, затем указываем, что ассоциируем адрес с инстансом student-instance.

После этого в списке инстансов видим, что student-instance присвоен внешний IP-адрес 8.43.86.56 (рис. 26).

Instances

<div> <div>Instance Name ▾</div> <div>Filter</div> <div>Filter</div> <div>Launch Instance</div> <div>✖ Terminate Instances</div> <div>More Actions ▾</div> </div>											
<input type="checkbox"/>	Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
<input type="checkbox"/>	student-instance	Ubuntu16.04	192.168.0.3 Floating IPs: 8.43.86.56	m1.small	student123	Active	nova	None	Running	7 minutes	Create Snapshot ▾
Displaying 1 item											

Рис. 26: Подключение внешнего IP-адреса к инстансу

После того, как к инстансу привязан внешний адрес, можно проверить его доступность, соединившись по SSH:

```
$ ssh ubuntu@8.43.86.56 -i Downloads/student123.pem
```

Инстансы созданные в TryStack доступны на протяжении 24 часов, так как сервис бесплатный, он позволяет только взглянуть на то, как работает OpenStack, поэтому он не годится для размещения своих проектов в облаке.

Установим для примера веб-сервер Nginx и разместим для общего доступа HTML-страничку.

```
# apt-get update && apt-get install nginx
# echo "<h1>Hello, this instance created by OpenStack.</h1>" > \
> /var/www/html/index.nginx-debian.html
# echo "<h3>Computer Science and Engineering, SevSU</h3>" >> \
> /var/www/html/index.nginx-debian.html
```

Заходим из веб-браузера по IP-адресу инстанса и видим результат работы веб-сервера (рис. 27).



Рис. 27: Демонстрация работы веб-сервера в облаке