

Севастопольский государственный университет

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

А.Р. Умеров<sup>1</sup>  
Е.Н. Мащенко<sup>2</sup>  
24 сентября 2016 г.<sup>3</sup>

---

<sup>1</sup>admin@amet13.name

<sup>2</sup>elmachenko@mail.ru

<sup>3</sup>Дата последней правки <https://github.com/Amet13/metod-oblaka>

# Содержание

<b>1 ЛР №1. Знакомство с виртуализацией, VirtualBox</b>	<b>3</b>
1.1 Теоретические основы виртуализации . . . . .	3
1.2 Порядок выполнения работы . . . . .	6
1.3 Контрольные вопросы . . . . .	7
<b>2 ЛР №2. Модель обслуживания SaaS, ownCloud</b>	<b>8</b>
2.1 Теоретические основы облачных вычислений . . . . .	8
2.2 Порядок выполнения работы . . . . .	11
2.3 Контрольные вопросы . . . . .	11
<b>3 ЛР №3. Модель обслуживания PaaS, Heroku</b>	<b>12</b>
3.1 Краткие сведения о методологиях разработки ПО . . . . .	12
3.2 Порядок выполнения работы . . . . .	14
3.3 Контрольные вопросы . . . . .	14
<b>4 ЛР №4. Модель обслуживания IaaS, OpenStack</b>	<b>15</b>
4.1 Краткие сведения об OpenStack . . . . .	15
4.2 Порядок выполнения работы . . . . .	16
4.3 Контрольные вопросы . . . . .	16
<b>A Создание виртуальной машины в VirtualBox</b>	<b>17</b>
<b>B Установка Debian GNU/Linux в VirtualBox</b>	<b>24</b>
<b>C Установка ownCloud Server в Debian GNU/Linux</b>	<b>36</b>
<b>D Подключение ownCloud Client к серверу</b>	<b>39</b>
<b>E Деплой приложения на Heroku</b>	<b>42</b>
<b>F Создание инстанса в OpenStack</b>	<b>45</b>

# 1 Лабораторная работа №1.

## Знакомство с виртуализацией. Система виртуализации Oracle VM VirtualBox

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

### 1.1 Теоретические основы виртуализации

В вычислениях, виртуализация является процессом создания виртуальной (не физической) версии чего-либо, в том числе аппаратных платформ виртуального компьютера, операционных систем, устройств хранения данных и вычислительных ресурсов.

Виртуализация может быть предоставлена на различных аппаратных и программных уровнях, таких как центральный процессор, диск, память, файловые системы и прочее. Чаще всего виртуализация используется для создания виртуальных машин и эмуляции различного оборудования для последующей установки операционных систем (ОС) на них.

Виртуализацию можно использовать в:

- консолидации серверов (возможность мигрировать физические сервера на виртуальные, таким образом увеличивается коэффициент использования серверных мощностей, 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, таким образом 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 снапшот<sup>1</sup> и быстро восстановить систему в случае сбоя);
- организации виртуальных рабочих станций (так называемых «тонких клиентов»<sup>2</sup>).

Общая схема взаимодействия виртуализации с аппаратным 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).

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

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

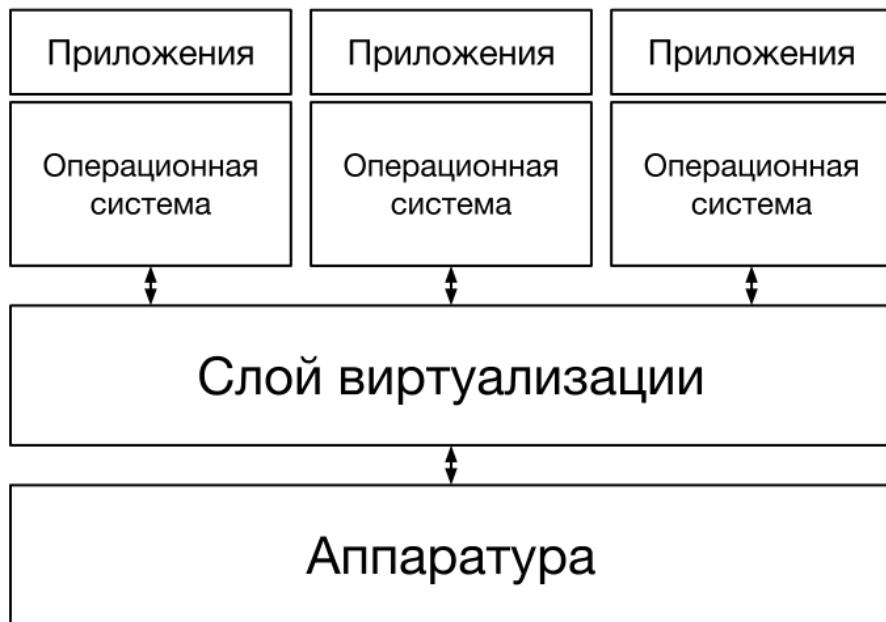


Рис. 1: Схема взаимодействия виртуализации с аппаратурой и ПО

Эмуляция аппаратных средств является одним из самых сложных методов виртуализации (рис. 2). В то же время, главной проблемой при эмуляции аппаратных средств остается низкая скорость работы, это вызвано тем, что каждая команда моделируется на основных аппаратных средствах.

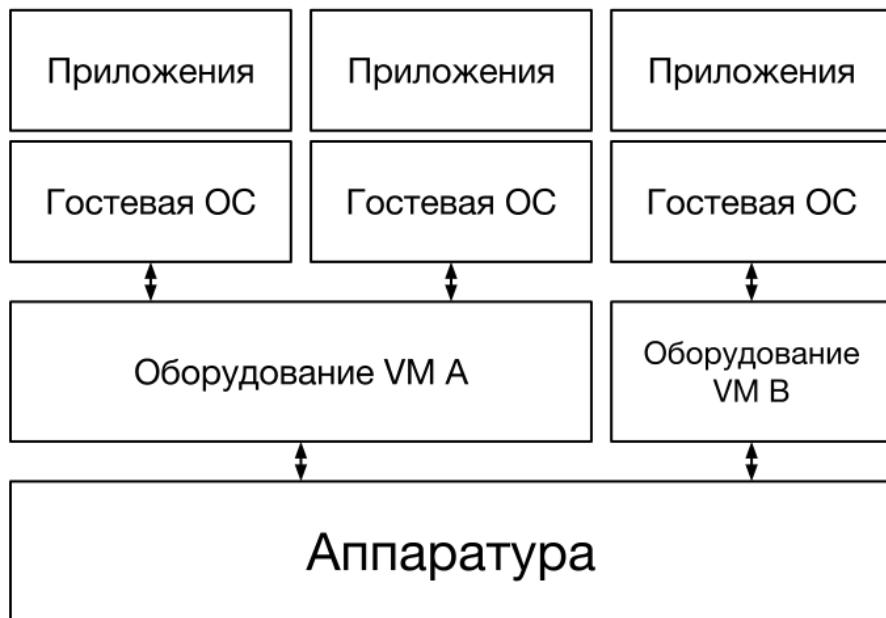


Рис. 2: Эмуляция оборудования моделирует аппаратные средства

В эмуляции оборудования используется механизм динамической трансляции, то есть каждая из инструкций эмулируемой платформы заменяется на заранее подготовленный фрагмент инструкций физического процессора.

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

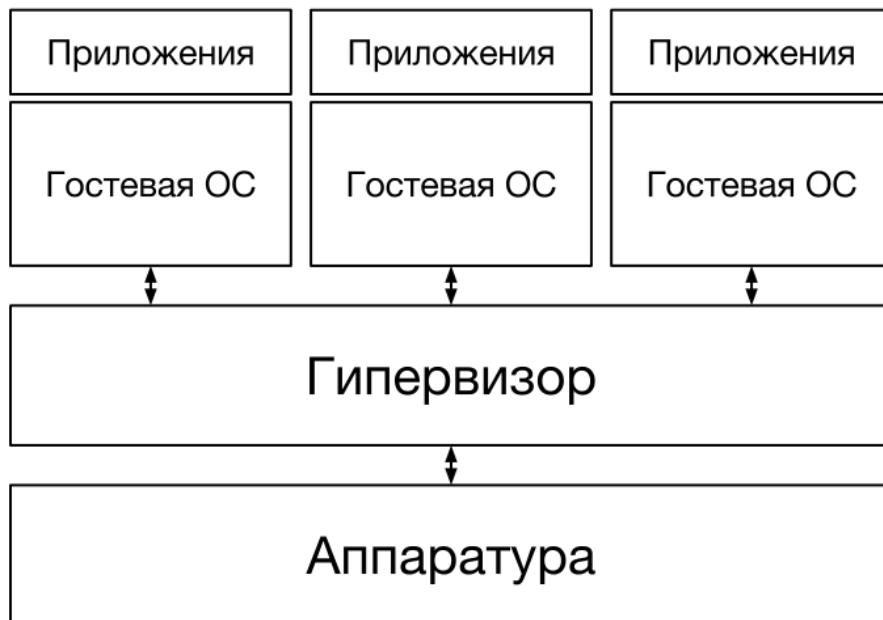


Рис. 3: Полная виртуализация использует гипервизор

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

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).

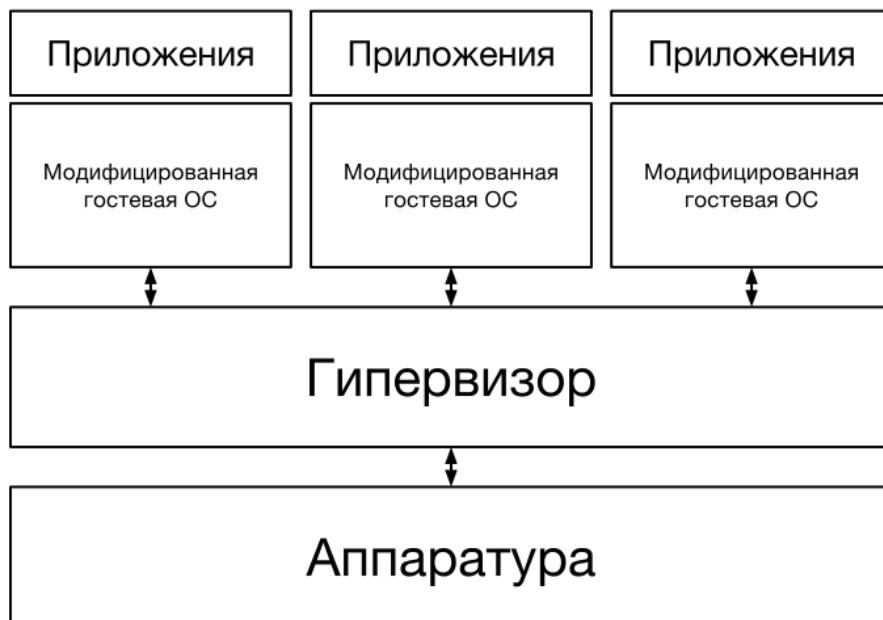


Рис. 4: Паравиртуализация разделяет процесс с гостевой ОС

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 контейнерами (рис. 5).

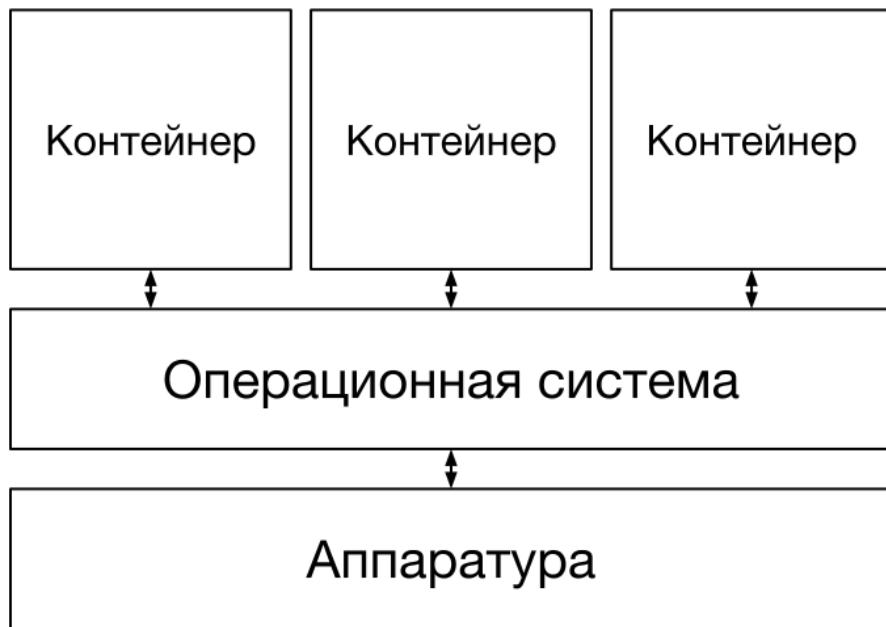


Рис. 5: Виртуализация уровня ОС изолирует серверы

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.

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).

## 1.2 Порядок выполнения работы

1. Скачать<sup>1</sup> и установить Oracle VM VirtualBox;
2. Изучить интерфейс программы и создать<sup>2</sup> первую виртуальную машину на основе образа Debian GNU/Linux<sup>3</sup>;
3. Подключить к виртуальной машине ранее скачанный образ Debian GNU/Linux;
4. Загрузиться с подключенного образа и установить<sup>4</sup> дистрибутив на виртуальный жесткий диск;

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

<sup>2</sup>Пример создания виртуальной машины описан в прил. А

<sup>3</sup><https://www.debian.org/distrib/>

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

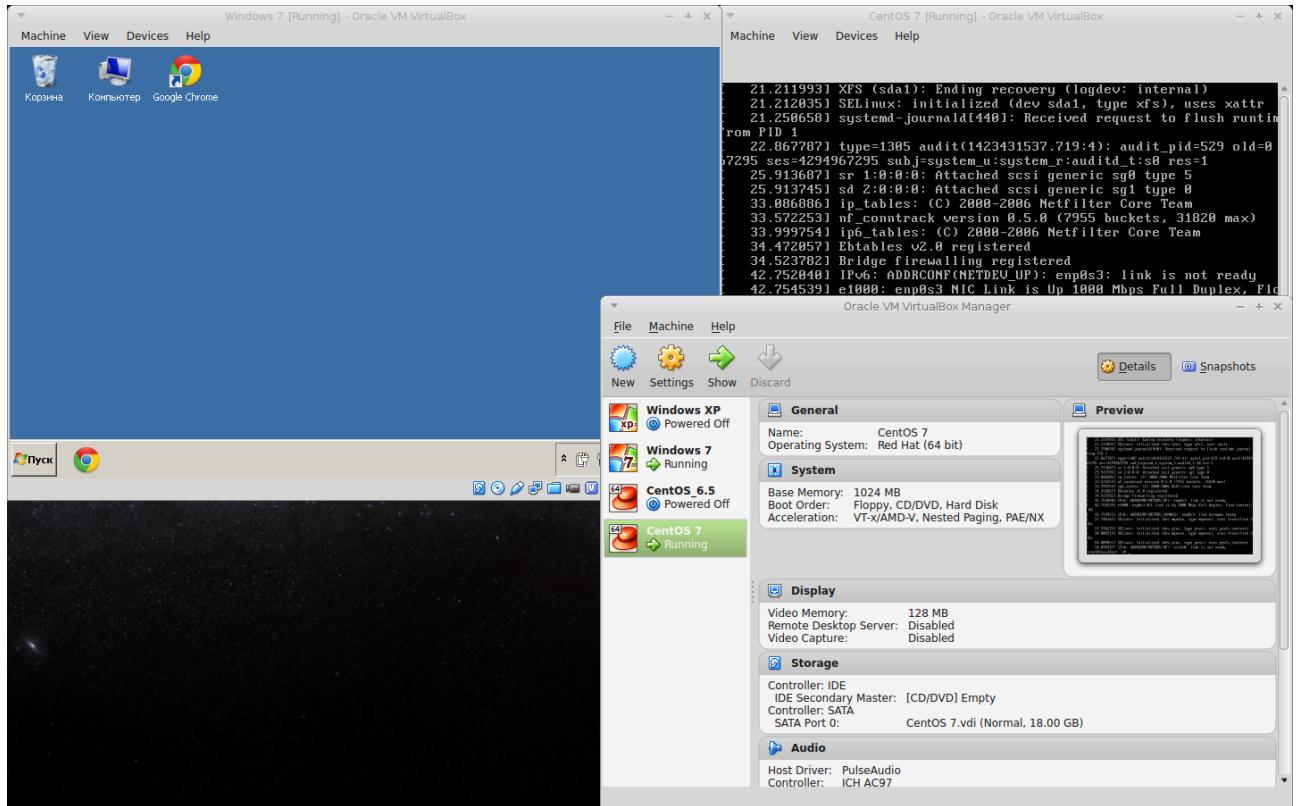


Рис. 6: Пример одновременной работы Windows 7 и CentOS 7

5. Во время установки дистрибутива необходимо будет задать пароль суперпользователя, пароль: `toor`, пароль для обычного пользователя можно задать произвольным;
6. С помощью команд `free`, `df`, `cat /proc/cpuinfo`, `ip`, посмотреть параметры виртуальной машины (RAM, HDD, CPU, сеть);
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 Putty<sup>1</sup>) подключиться по SSH к виртуальной машине<sup>2</sup>.

### 1.3 Контрольные вопросы

1. Назовите основные типы виртуализации, какие между ними сходства и различия?
2. К какому типу виртуализации относится система VirtualBox и почему?
3. Как может использовать виртуализацию в работе программист, системный администратор, сетевой инженер, студент?
4. Назовите основные недостатки и преимущества контейнерной виртуализации?
5. В чем различие между режимами сети NAT и «сетевой мост» (Network Bridge)?

<sup>1</sup><http://www.chiark.greenend.org.uk/~sgtatham/putty/>

<sup>2</sup>Если во время установки ОС был пропущен пункт с установкой SSH-сервера, это можно сделать командой `apt install openssh-server`

## 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, от приложений и до центров обработки данных (ЦОД), через Интернет при условии оплаты за фактическое использование.

Одной из первых, кто стала 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)<sup>1</sup>, после этого в 2009 г. компания Google представила Google Apps<sup>2</sup>.

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 переносятся 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 информационных технологий 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).

Такие модели обслуживания как Container as a Service (CaaS) и Database as a Service (DaaS), являются по больше мере частными случаями IaaS и SaaS соответственно.

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 модели SaaS могут служить: почтовая служба Gmail, Dropbox, Google Docs, Microsoft Office 365.

---

<sup>1</sup><https://aws.amazon.com/ru/ec2/>

<sup>2</sup><https://apps.google.com/>

PaaS — 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 PaaS 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 Compute Engine (GCE), Amazon Web Services (AWS).



Рис. 7: Представители различных моделей обслуживания

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, как правило оно находится в собственности самой организации. Публичное облако предназначено для широкой публики, как правило находится в собственности сторонних организаций. Общественное, как правило предназначено для сообщества или организации, а гибридное облако состоит из двух или более различных облачных инфраструктур.

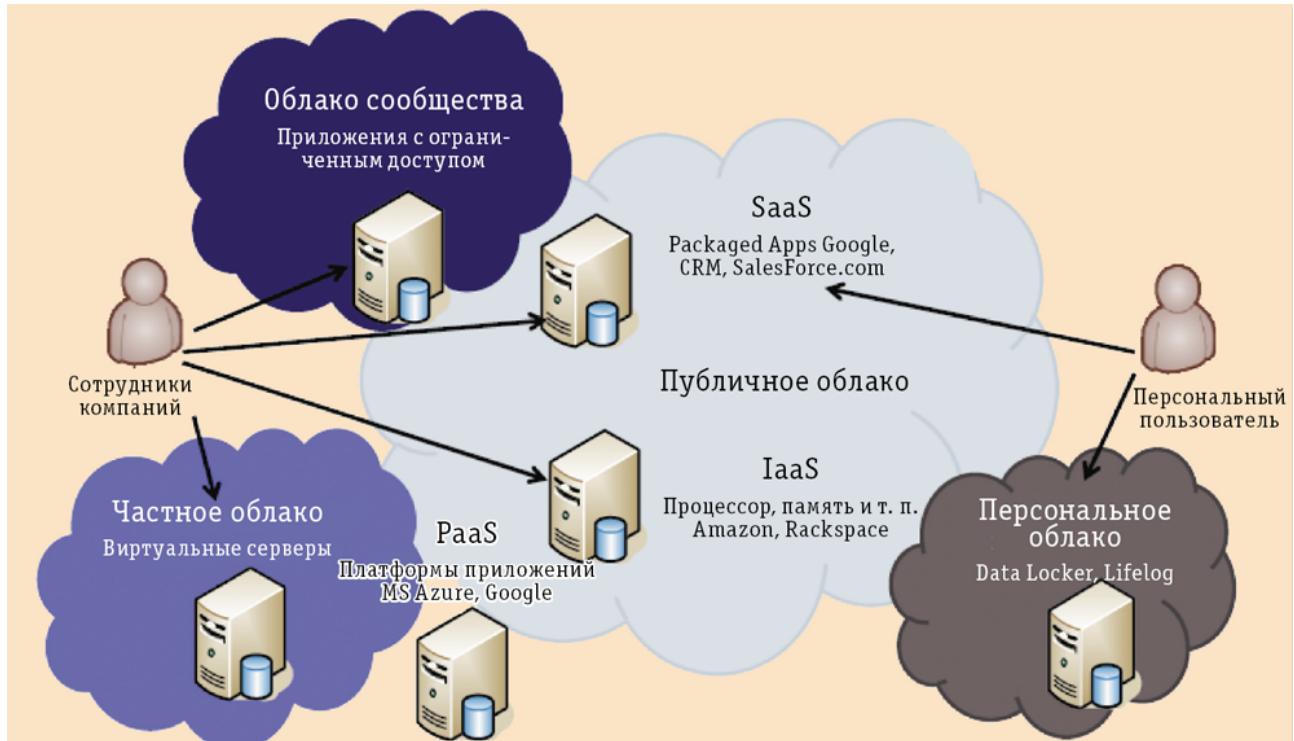


Рис. 8: Различия между моделями развертывания облака

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 код<sup>1</sup> 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.

<sup>1</sup><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 ранее установленным (в ЛР№1) дистрибутивом Debian. Виртуальная машина должна иметь доступ в сеть Интернет для скачивания нужных пакетов.

1. Установить<sup>1</sup> ownCloud Server в виртуальную машину;
2. На хост-машине или на другом компьютере сети скачать<sup>2</sup> и установить ownCloud Client;
3. Подключить ownCloud Client к серверу во время первого запуска клиента, а также синхронизировать все данные с сервера<sup>3</sup>;
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 преимущества ownCloud по сравнению с Dropbox или другими облачными хранилищами? Недостатки?
4. Можно ли Skype можно отнести к модели SaaS, почему?
5. Почему одни организации предпочитают использовать в своей инфраструктуре частные облака, а другие — публичные?

---

<sup>1</sup>Пример установки ownCloud Server представлен в прил. С

<sup>2</sup><https://owncloud.org/install/>

<sup>3</sup>Пример настройки ownCloud Client представлен в прил. D

### 3 Лабораторная работа №3.

## Модель обслуживания PaaS на примере деплоя приложения в Heroku

**Цель работы:** ознакомиться с моделью гибкой 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 вследствии 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.

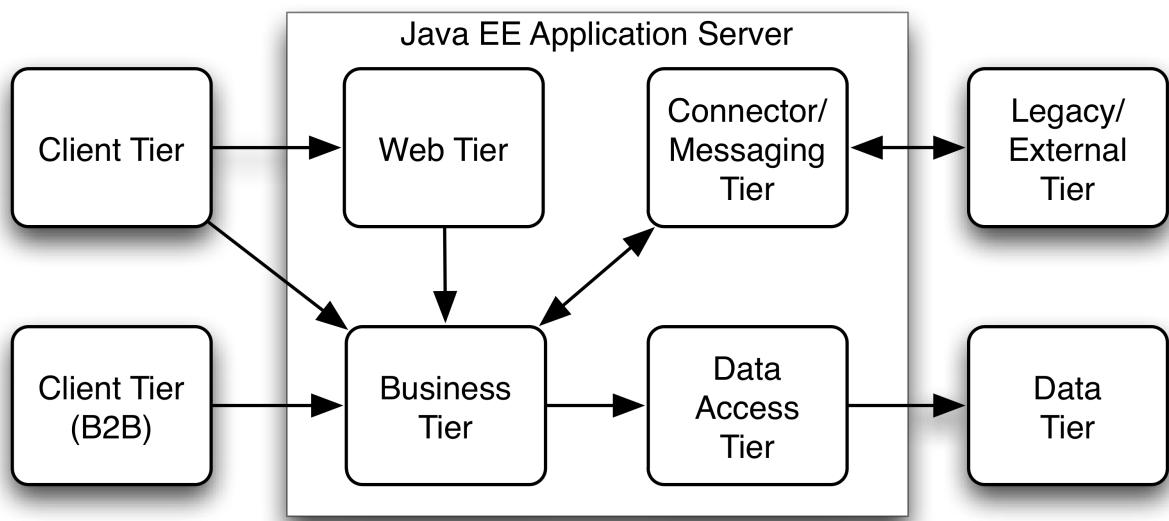


Рис. 9: Пример многоуровневой архитектуры

Организация процесса разработки ПО 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.

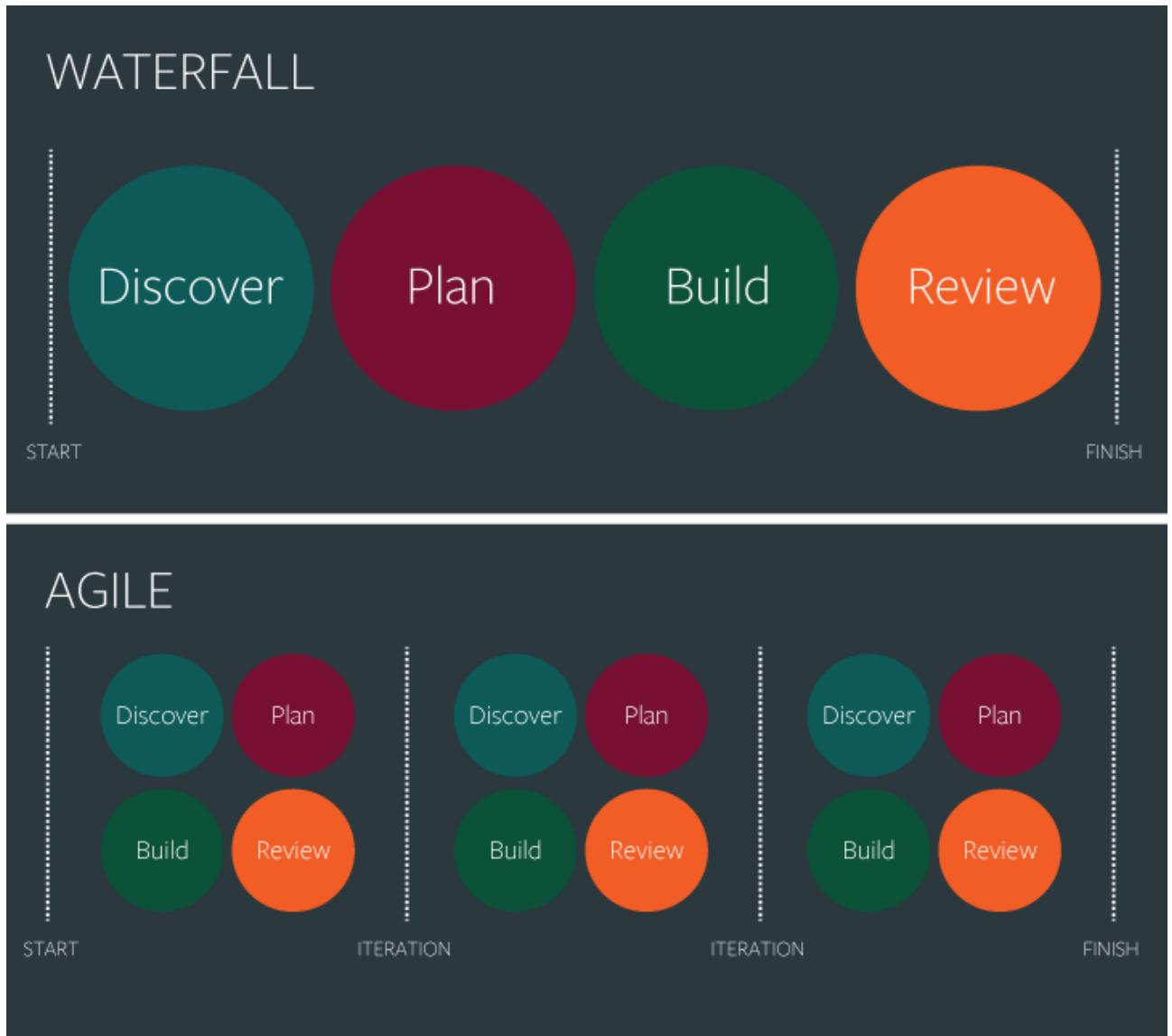


Рис. 10: Каскадная и Agile-методологии

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 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 Heroku<sup>1</sup>;
2. Создать тестовое приложение на языке Python (или любом другом языке);
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 необходимы команды `add`, `commit`, `push` в Git?
4. В чем преимущество использования Heroku по сравнению с локальным окружением?

---

<sup>1</sup>Пример работы с Heroku представлен в прил. E

## 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, ознакомиться с преимуществами использования IaaS-решений по сравнению с PaaS.

#### 4.1 Краткие сведения об OpenStack

Ранее в лабораторной работе №3, уже использовалось готовое PaaS-решение от облачного провайдера Heroku. Однако если же нужно использовать более гибкие IaaS-решения или же есть желание самому стать облачным провайдером, то необходимо использование инструментов, 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 можно выбрать гипервизор в зависимости от требований, libvirt (QEMU/KVM), Hyper-V, VMWare, XenServer, Xen с libvirt.

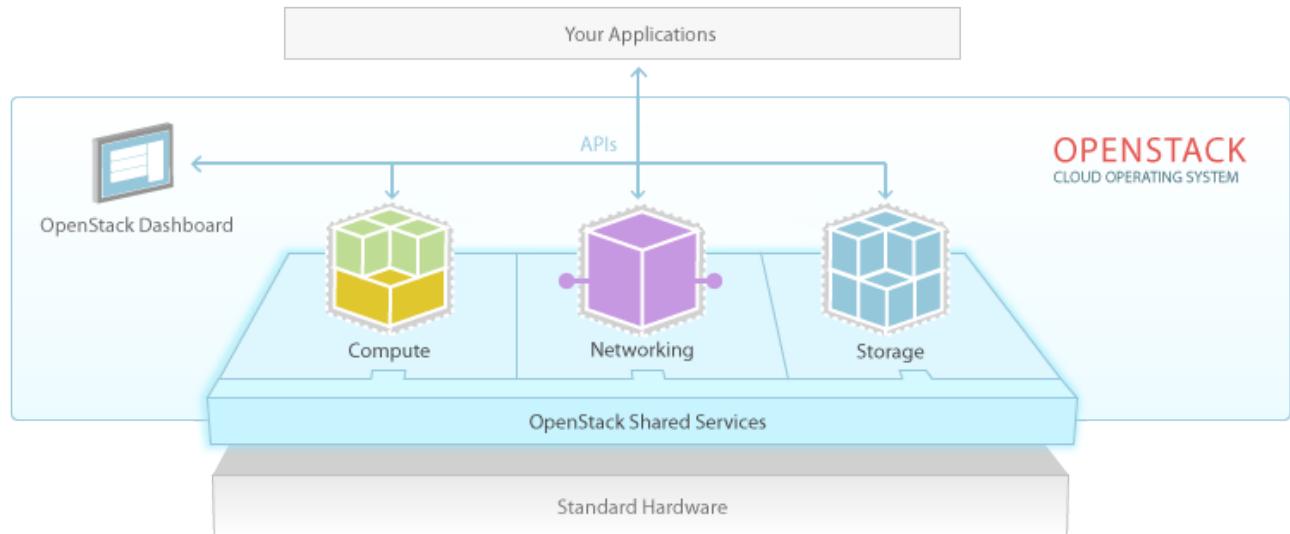


Рис. 11: Общая схема OpenStack

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 отсутствия привязанности к вендору, это может быть экономически эффективным.

## 4.2 Порядок выполнения работы

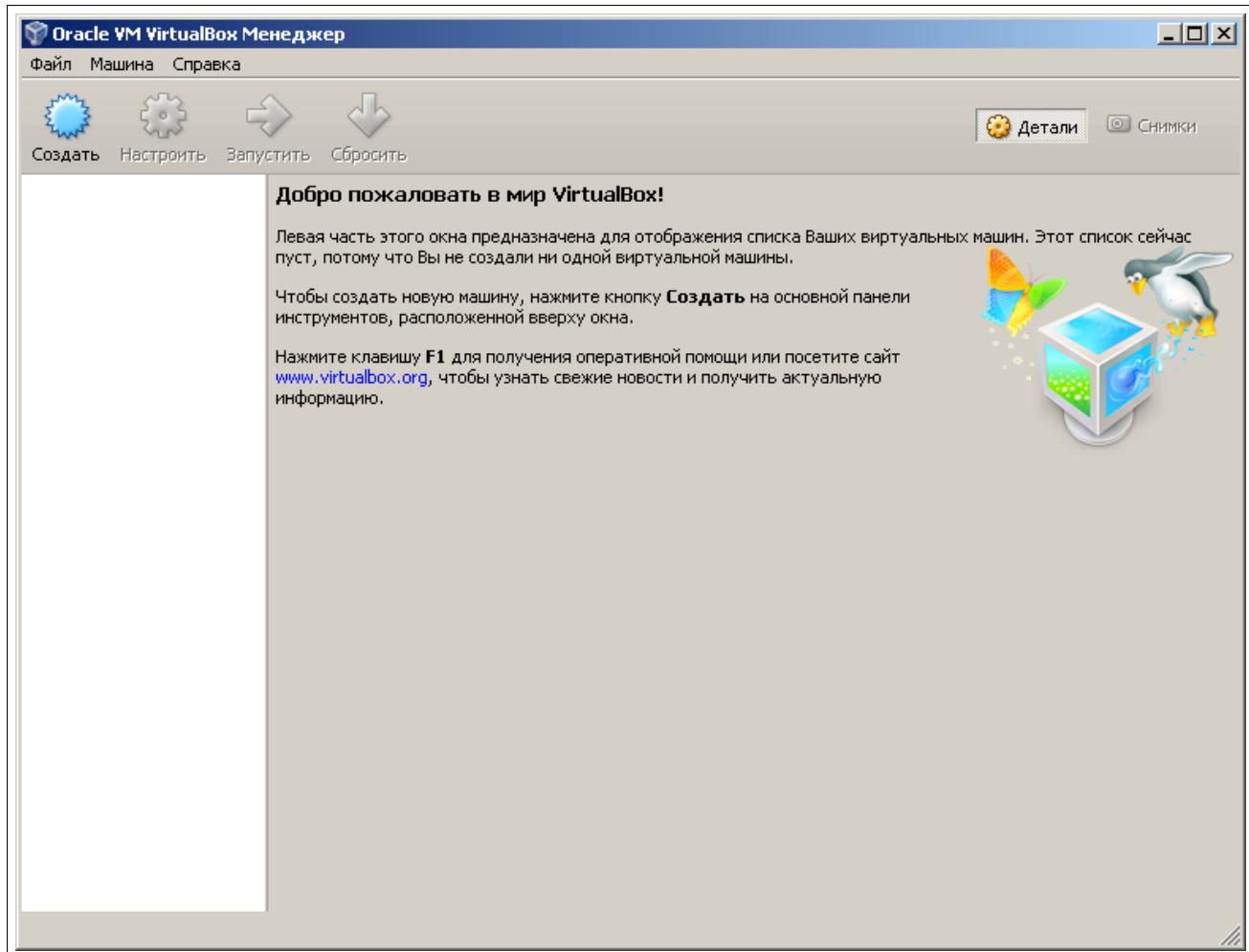
Для выполнения данной лабораторной работы необходим аккаунт на Facebook. Пример работы с TryStack представлен в прил. F.

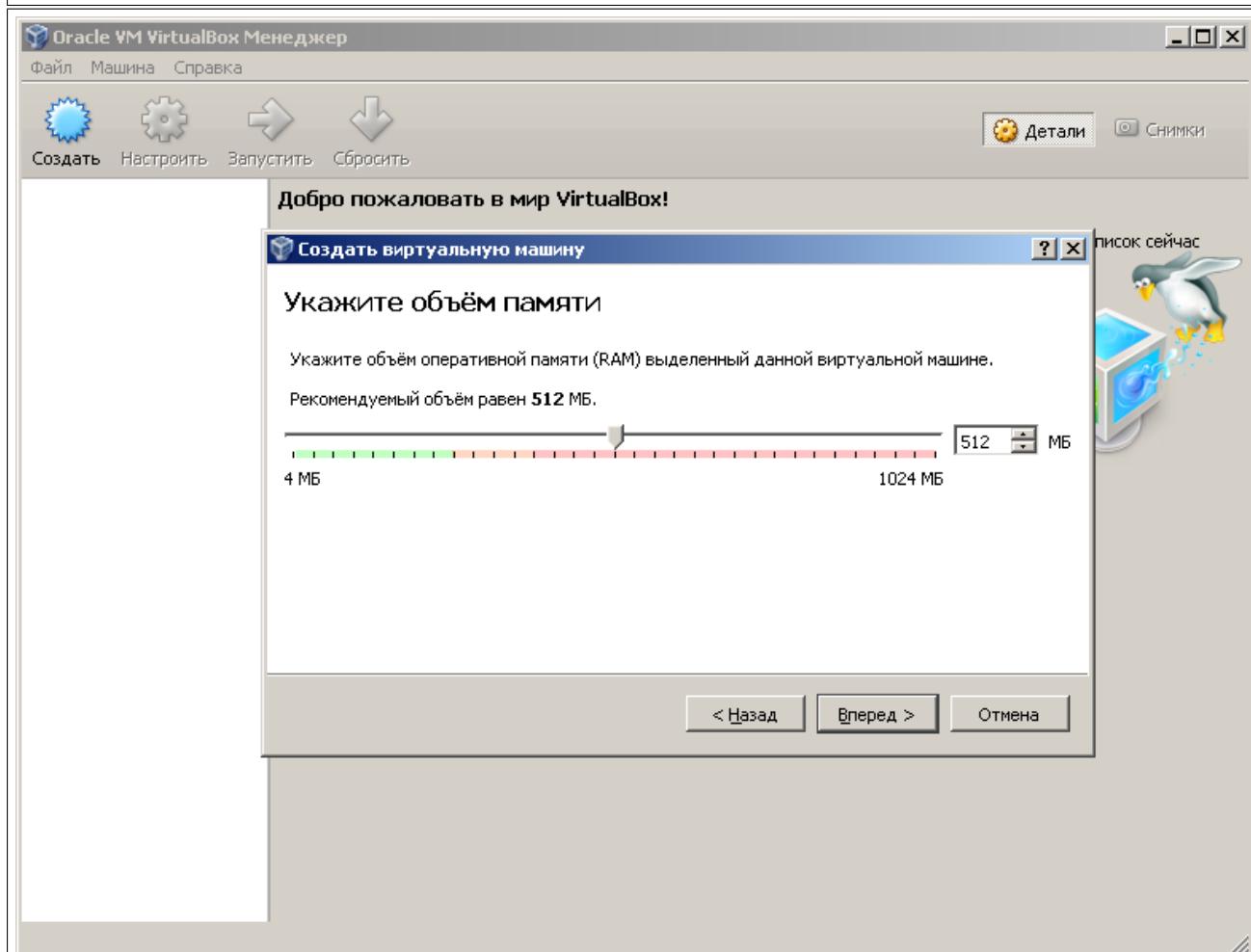
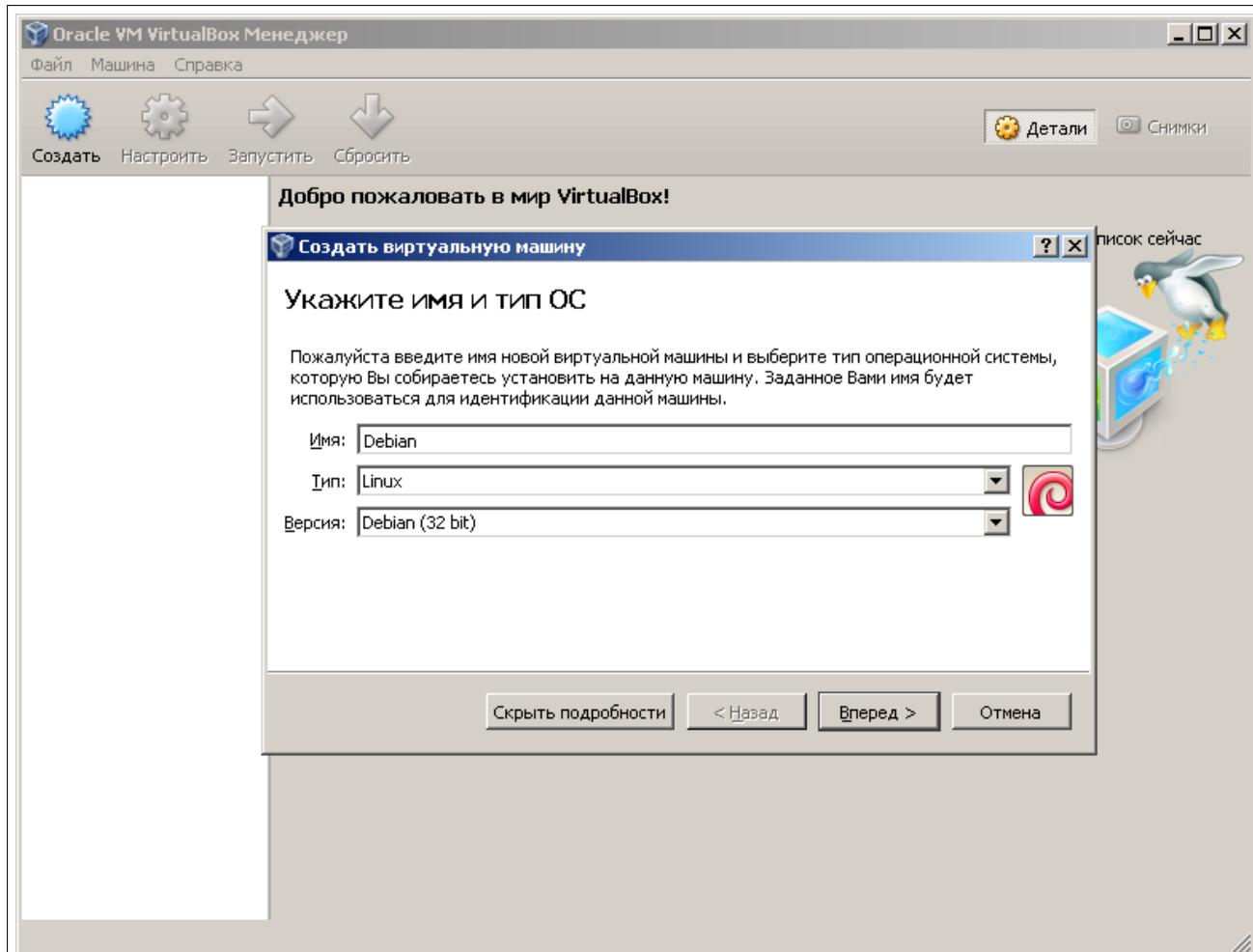
1. Вступить в группу TryStack на Facebook;
2. Авторизоваться на сервисе TryStack и ознакомиться с интерфейсом OpenStack;
3. Согласно прил. F настроить окружение для создания инстанса;
4. Создать тестовый инстанс и разместить любое приложение (по желанию);
5. Проверить работоспособность приложения в облаке.

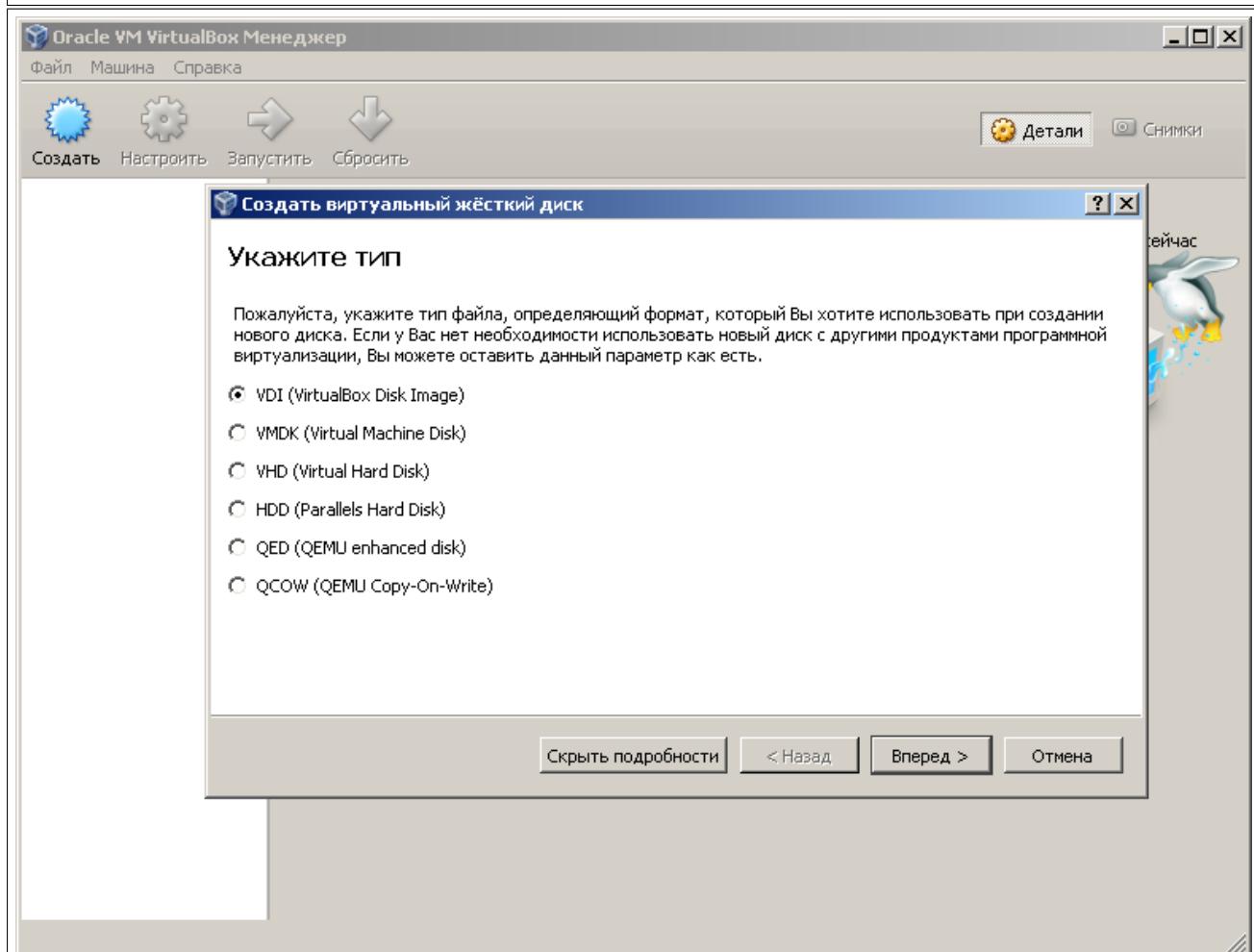
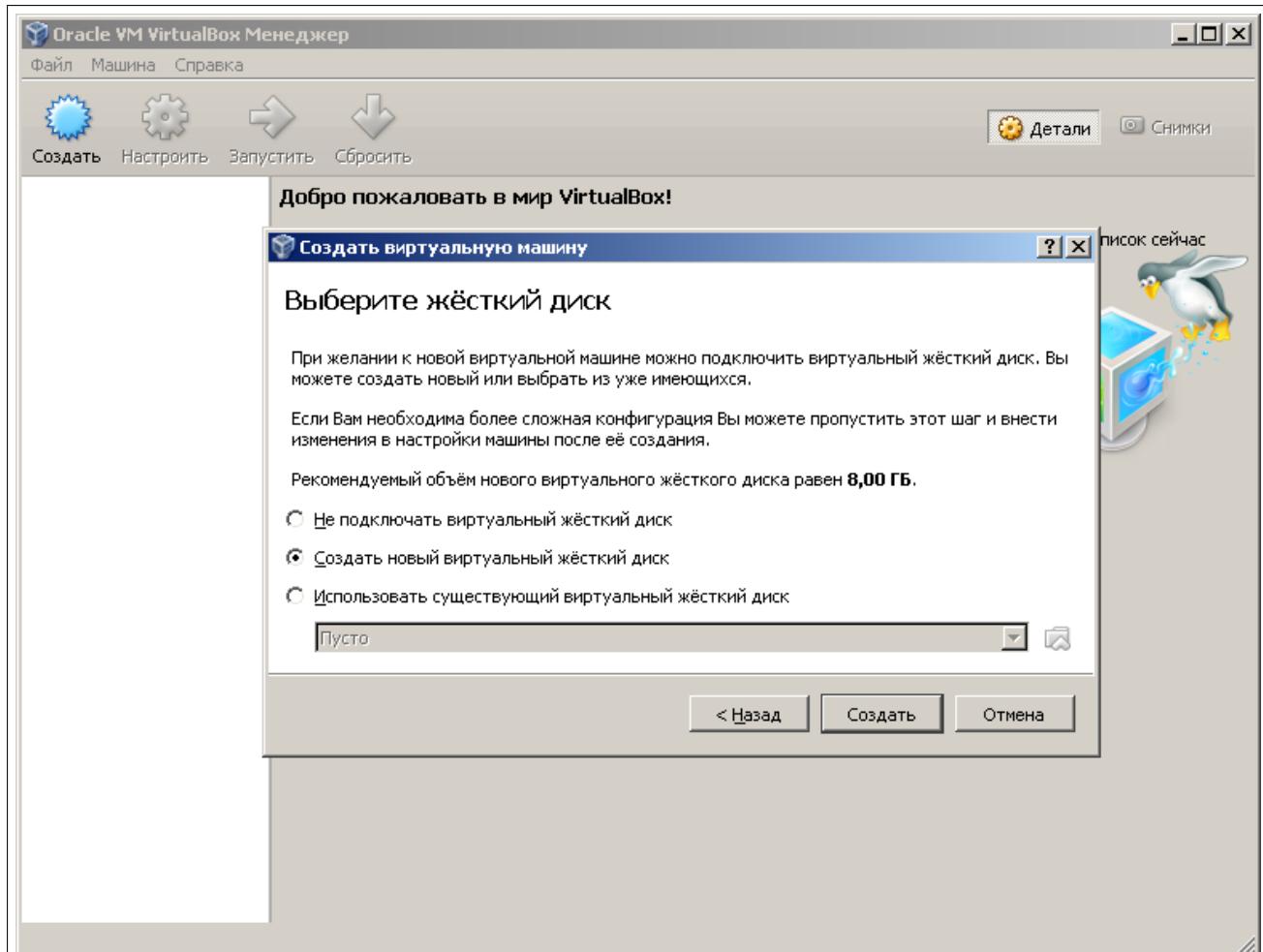
## 4.3 Контрольные вопросы

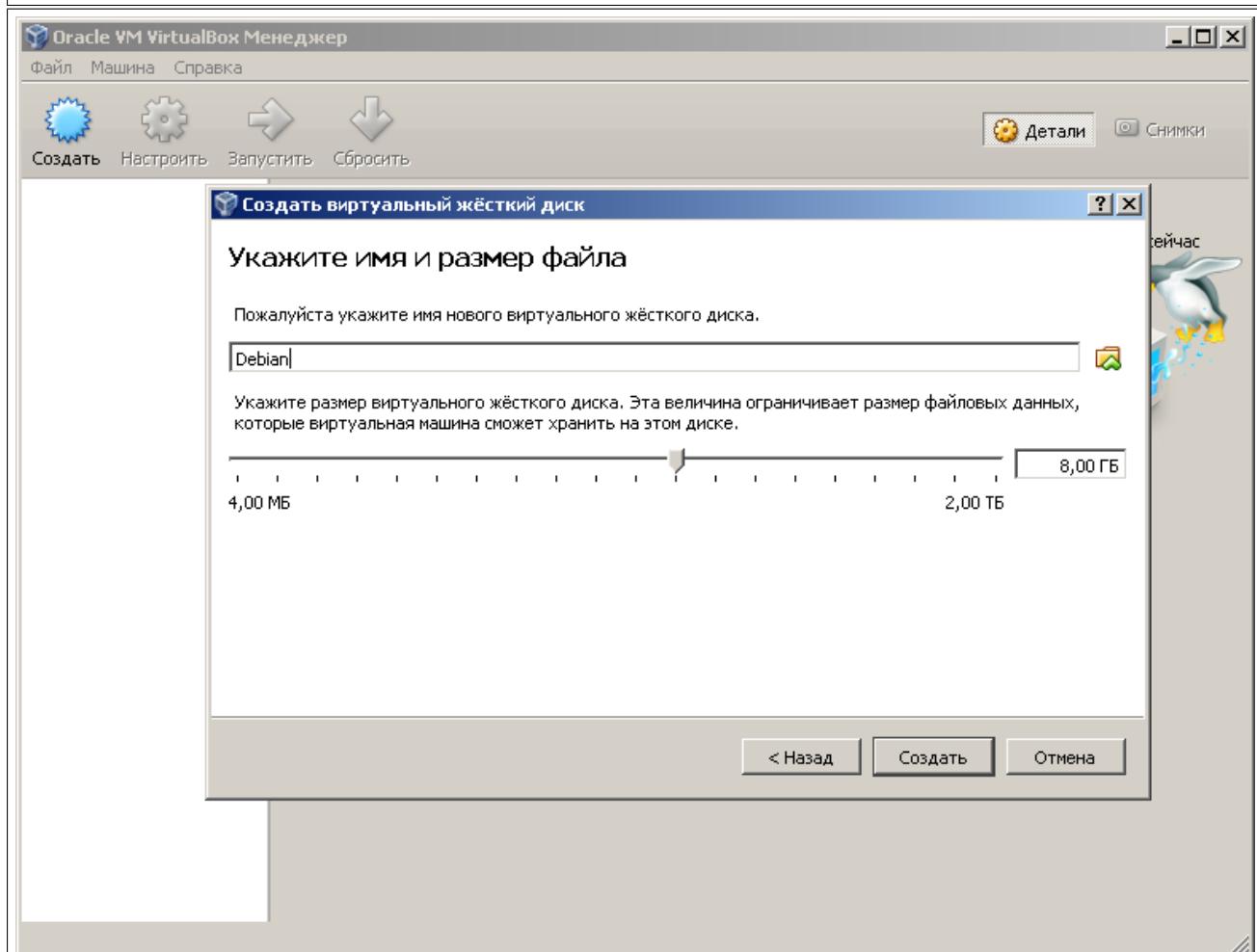
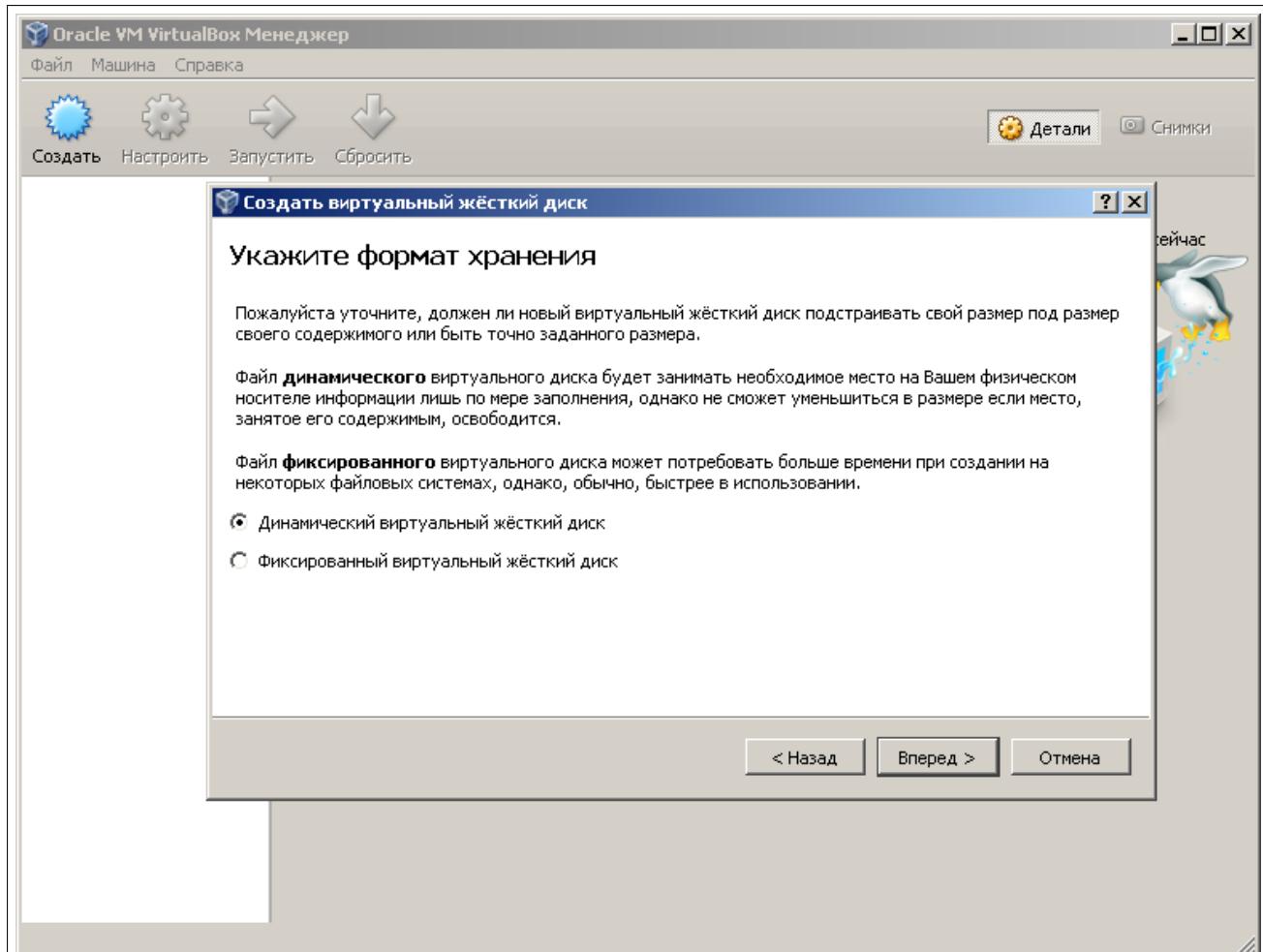
1. В чем преимущество и недостатки использования IaaS перед PaaS?
2. Назовите несколько примеров облачных поставщиков IaaS?
3. Почему использование SSH-ключей безопаснее чем парольная аутентификация?

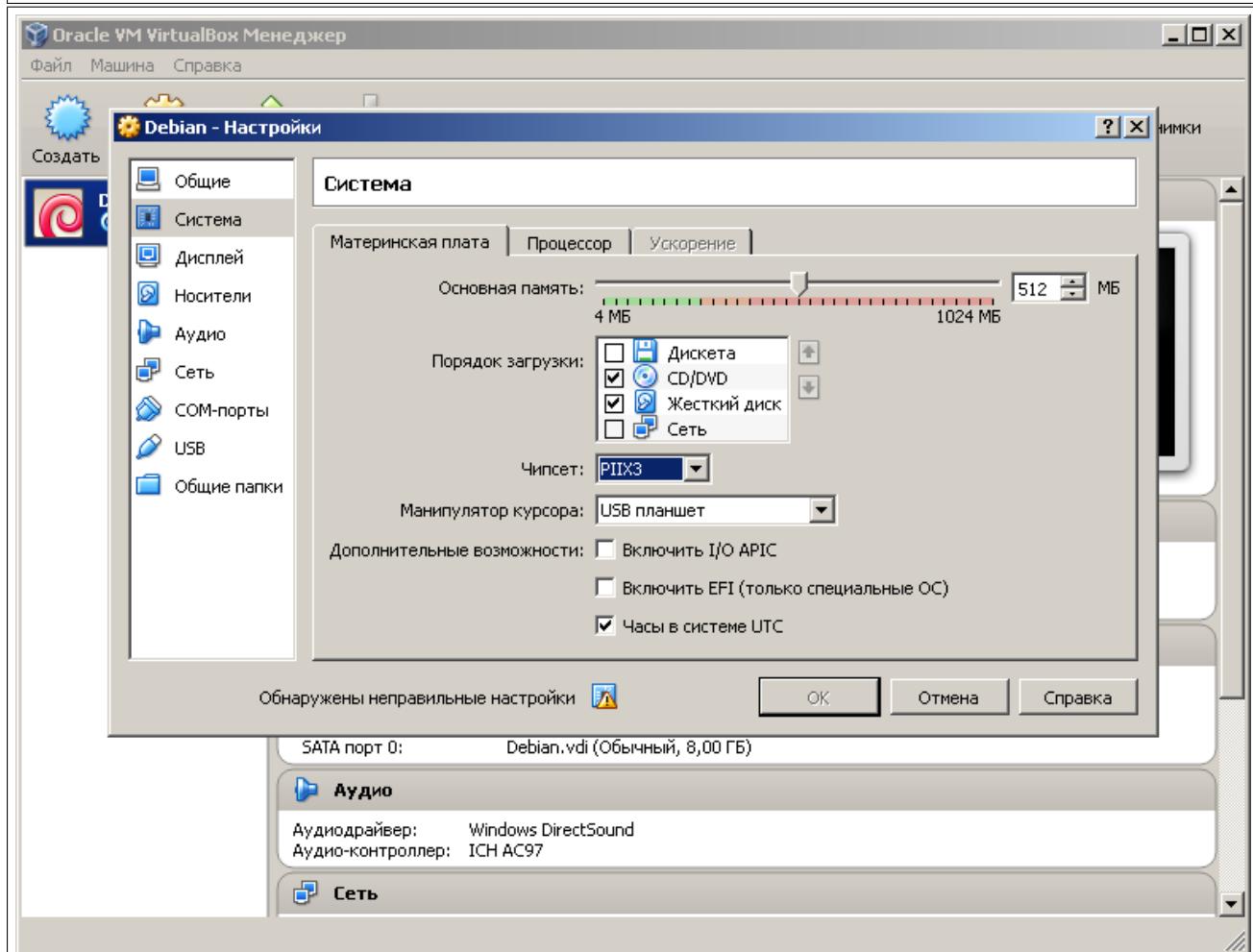
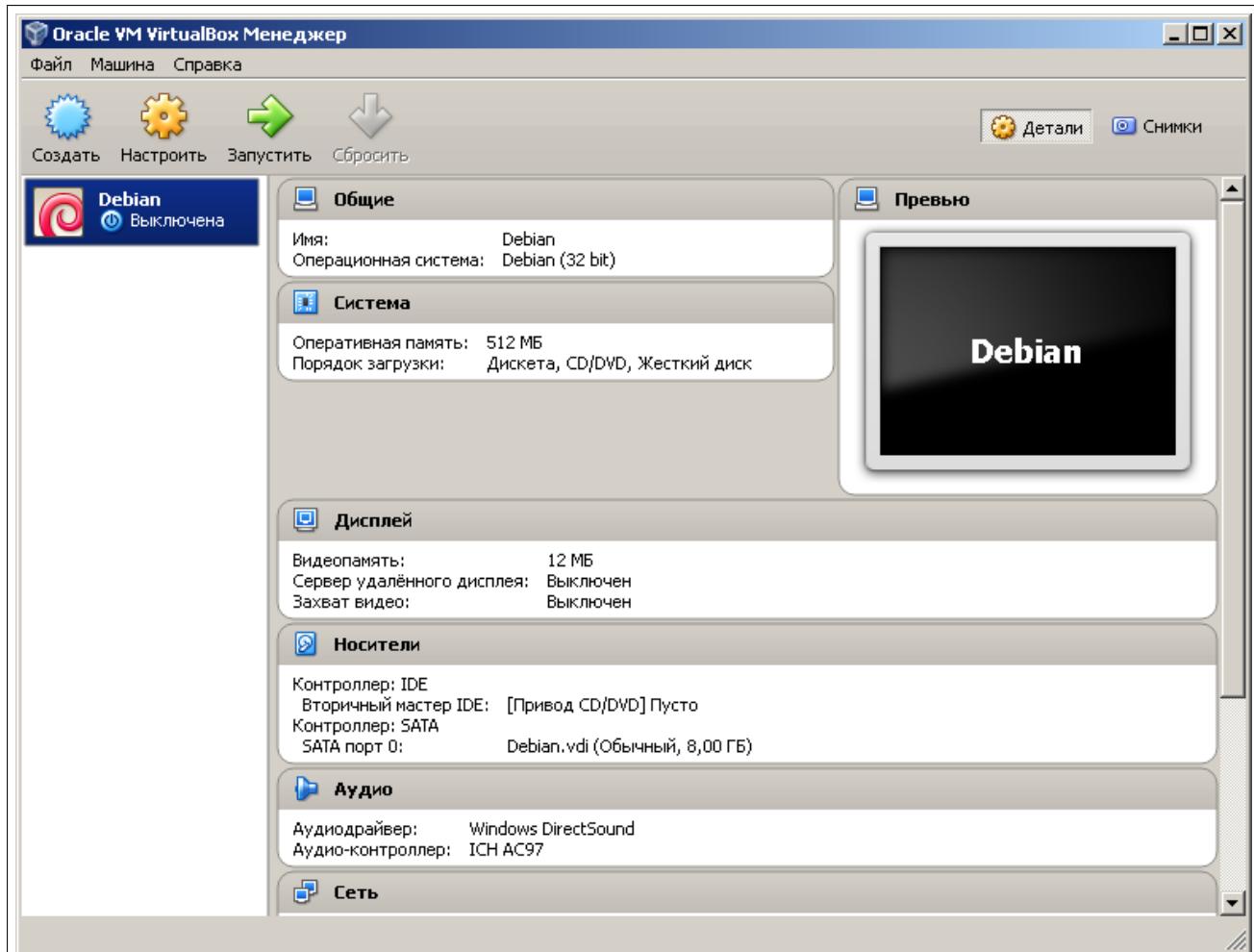
## A Создание виртуальной машины в VirtualBox

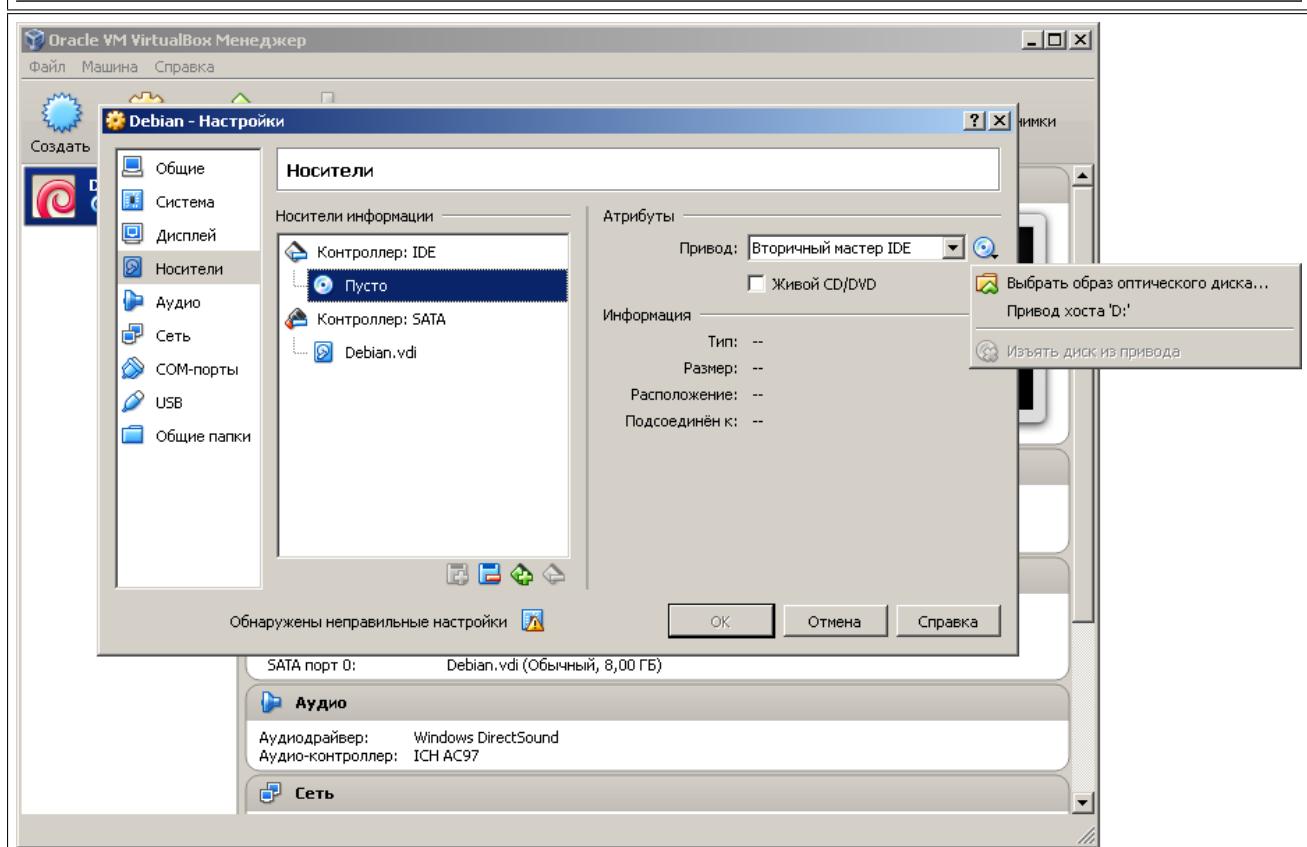
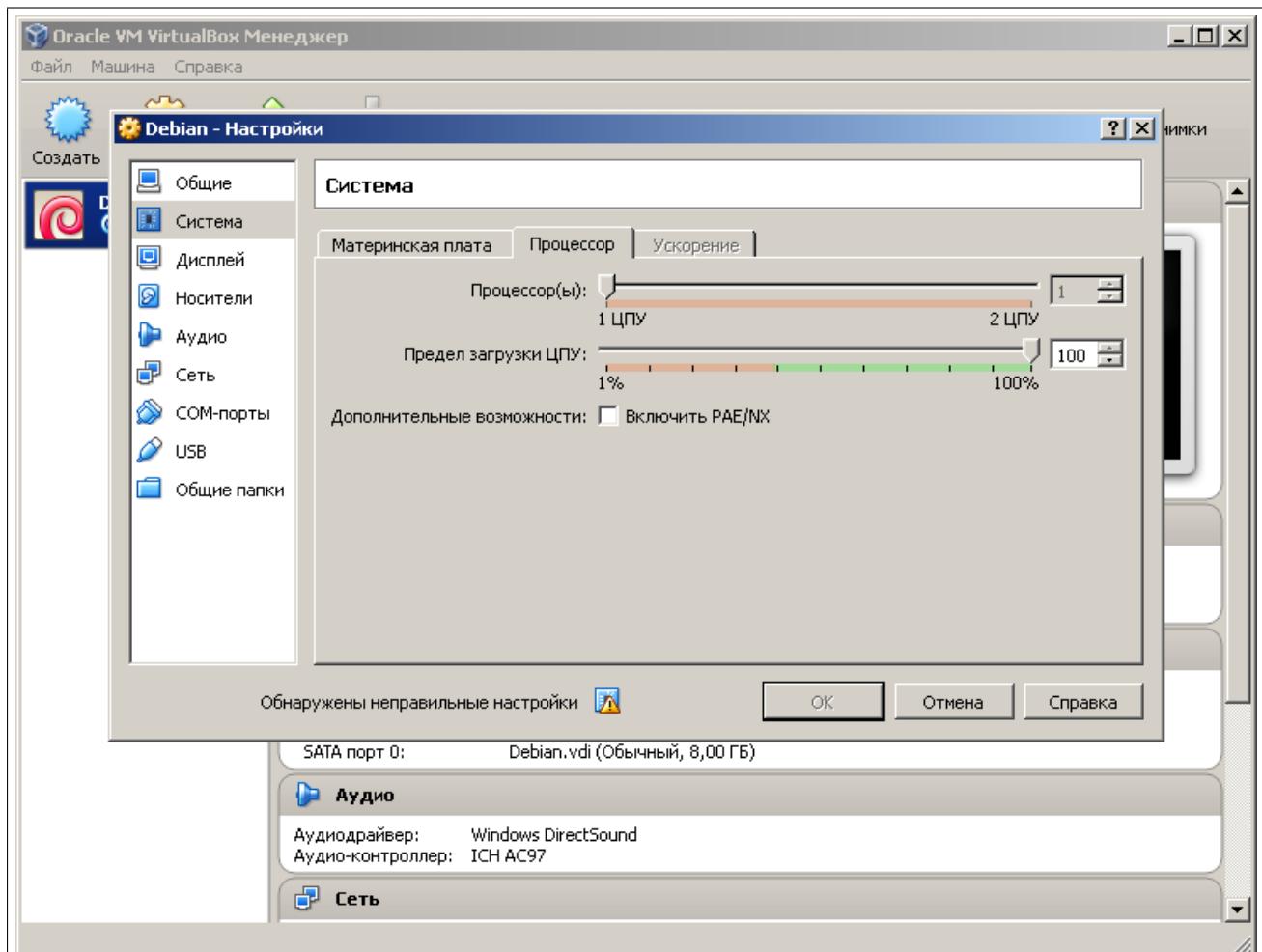


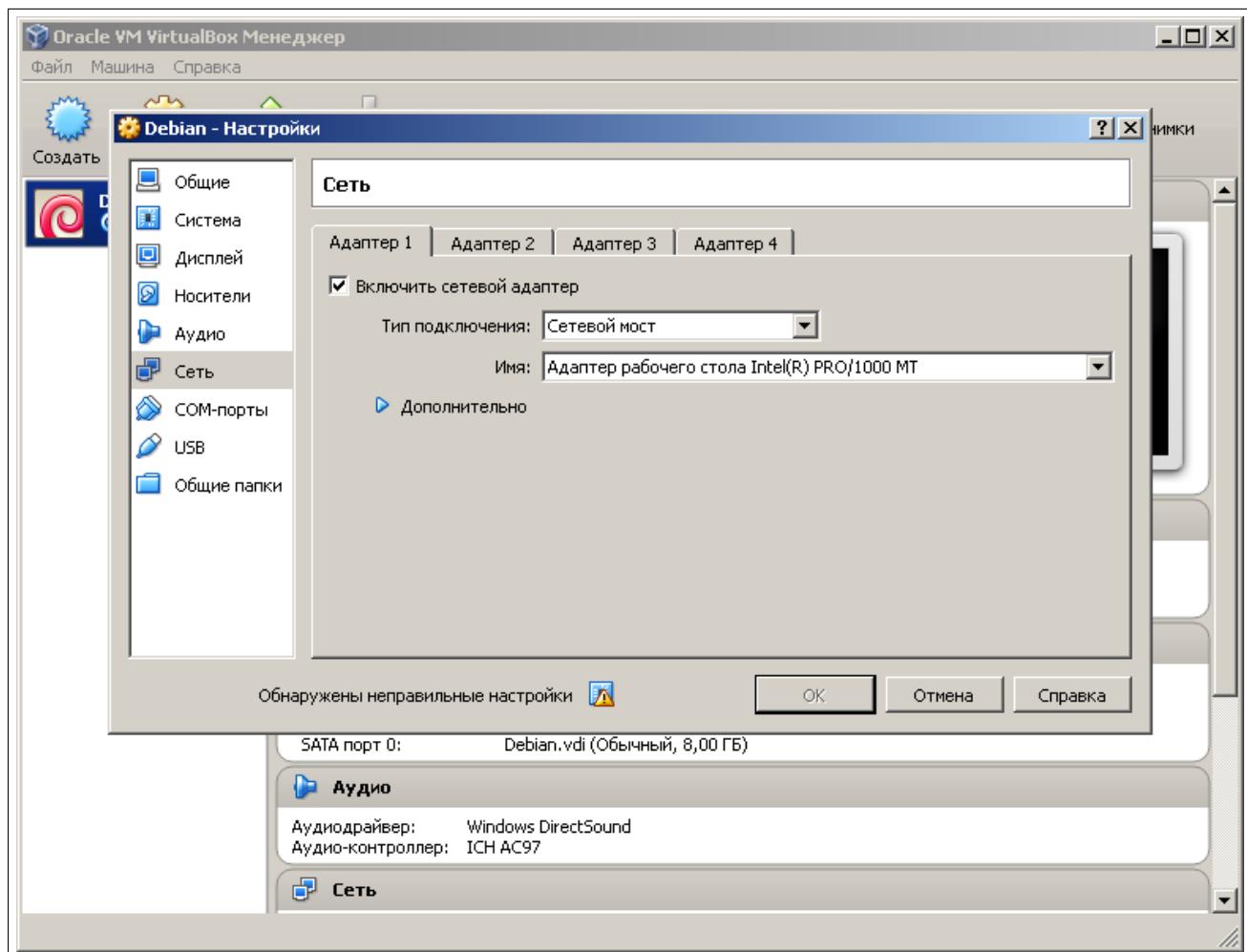




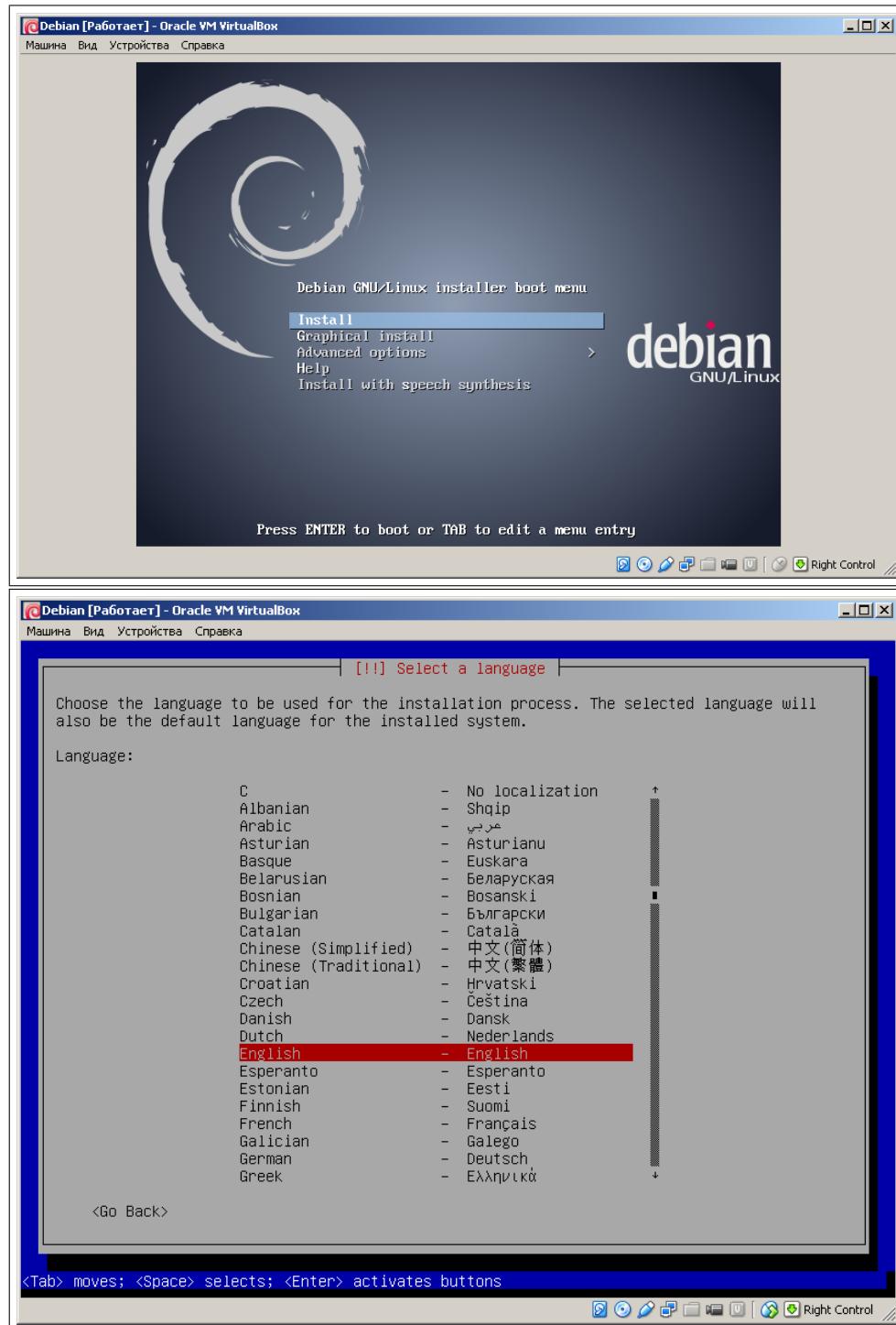


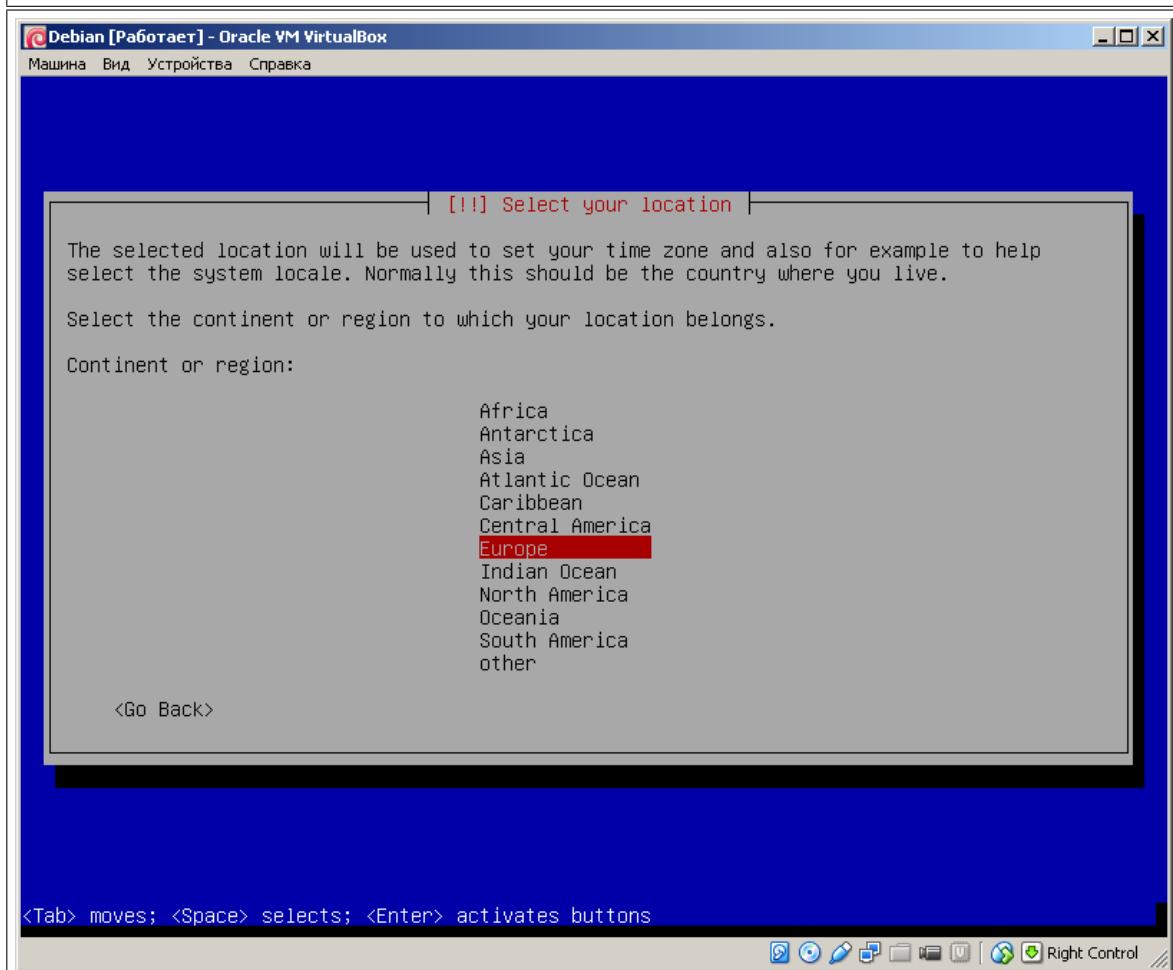
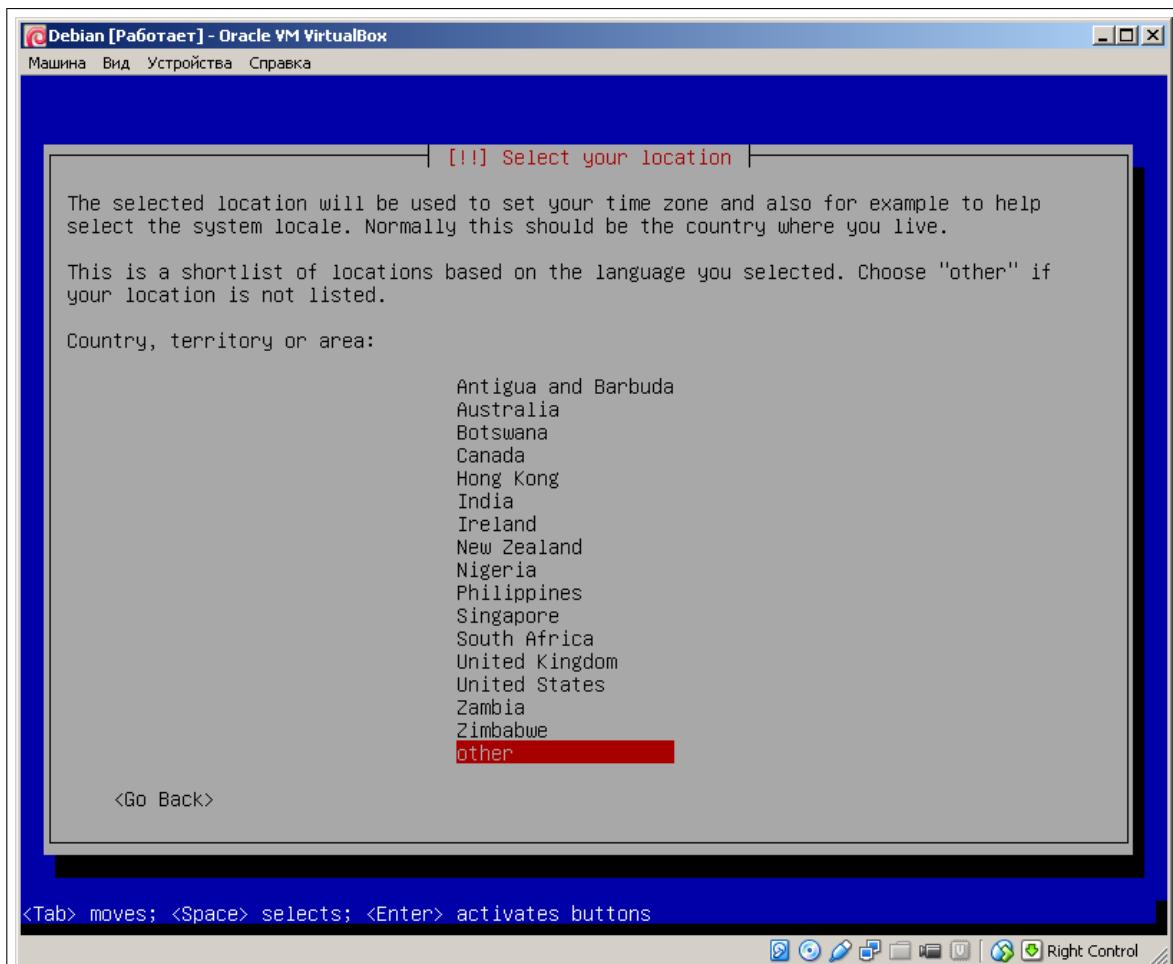


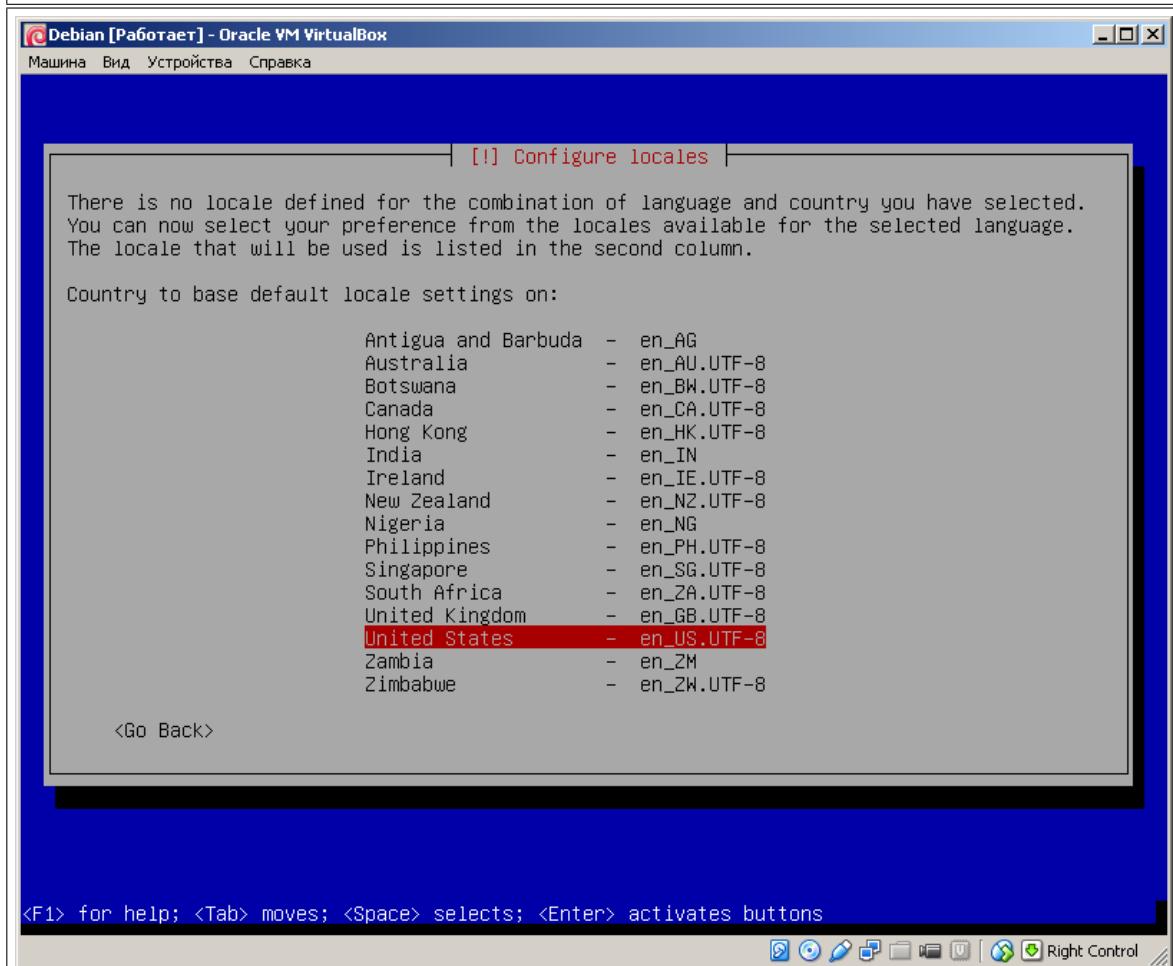
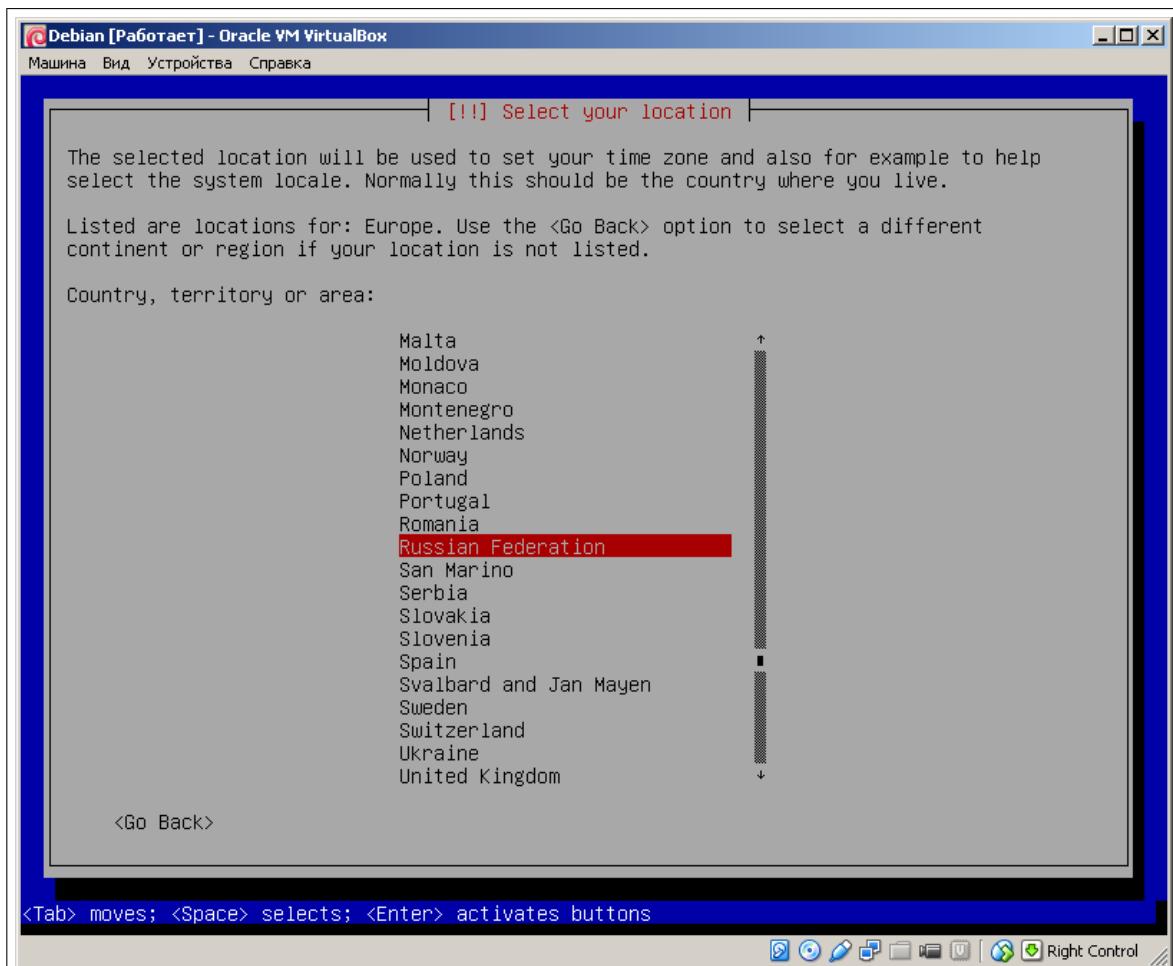


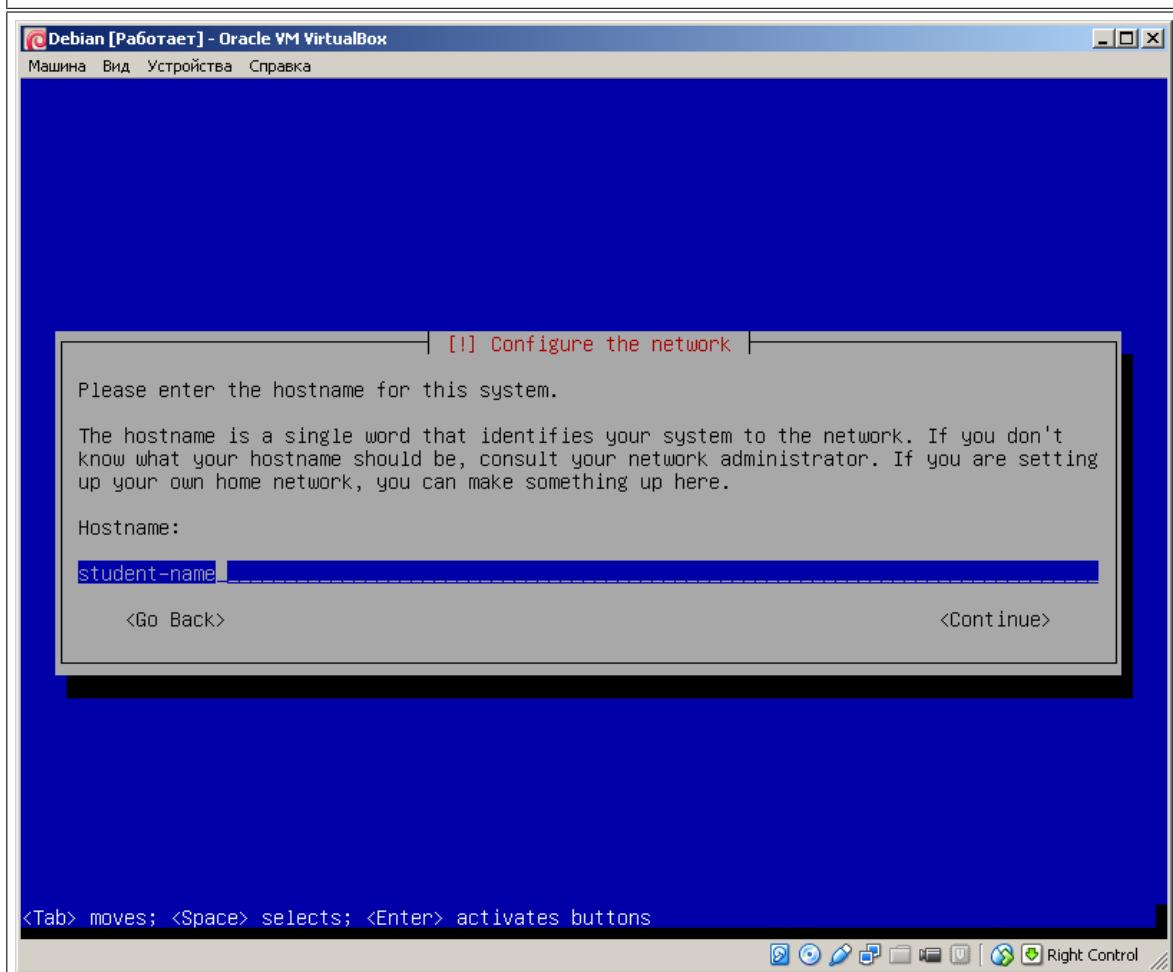
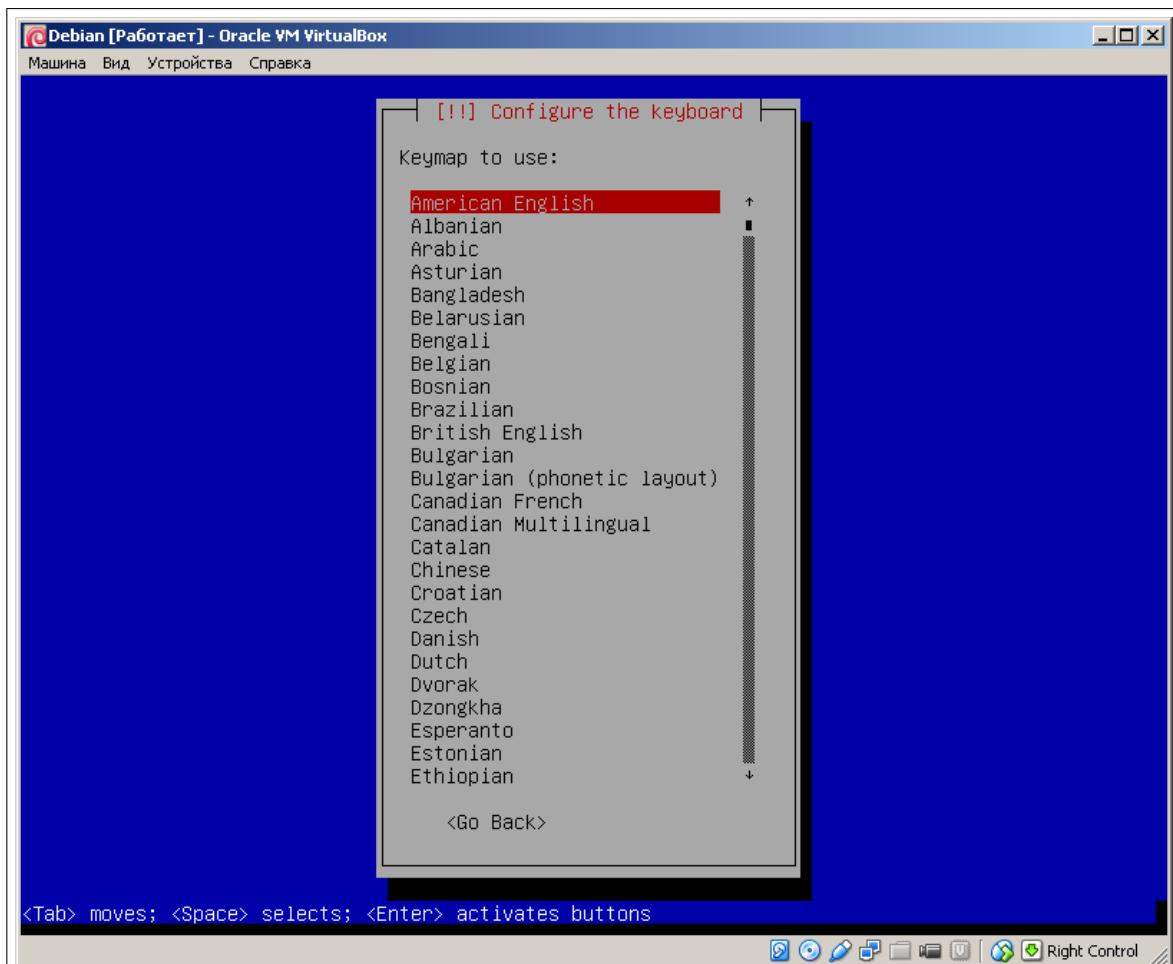


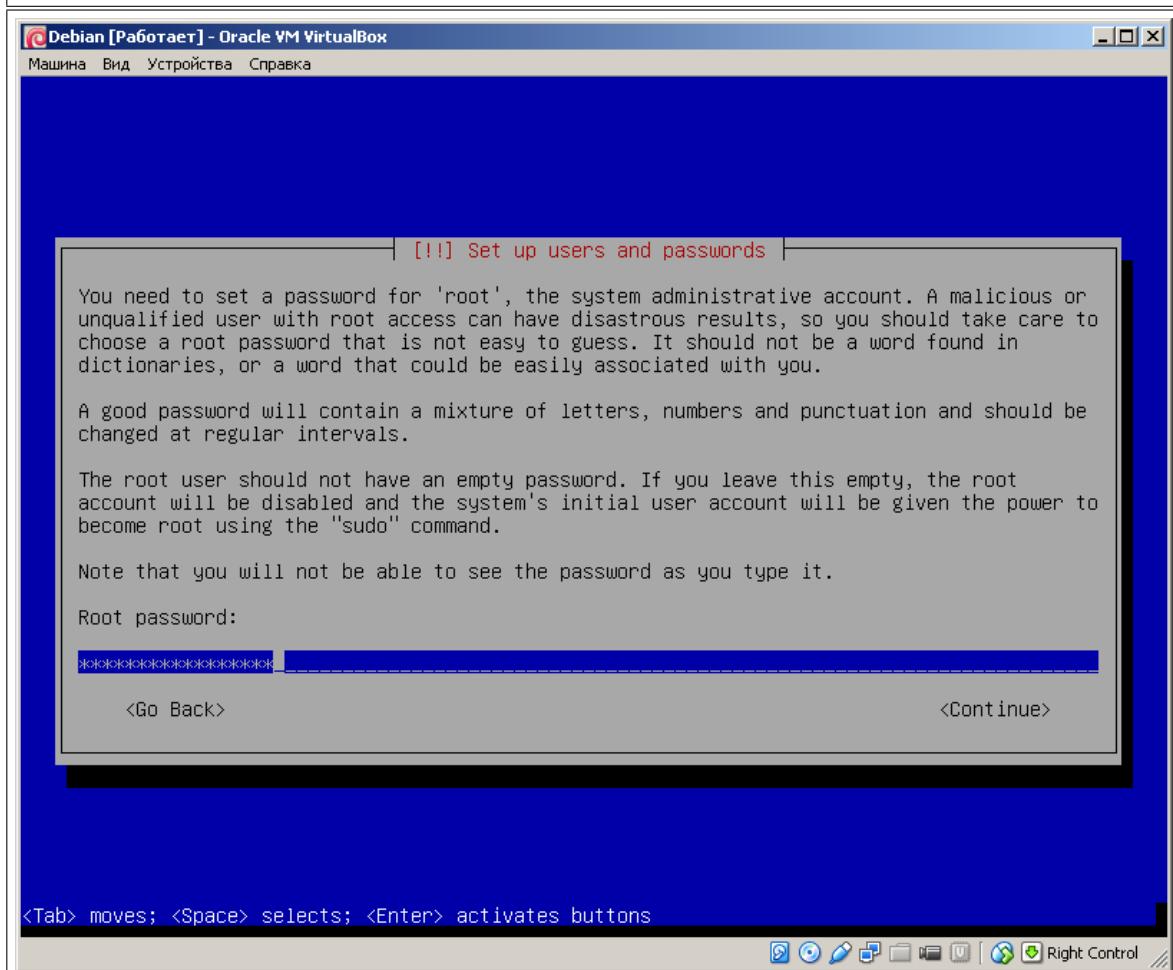
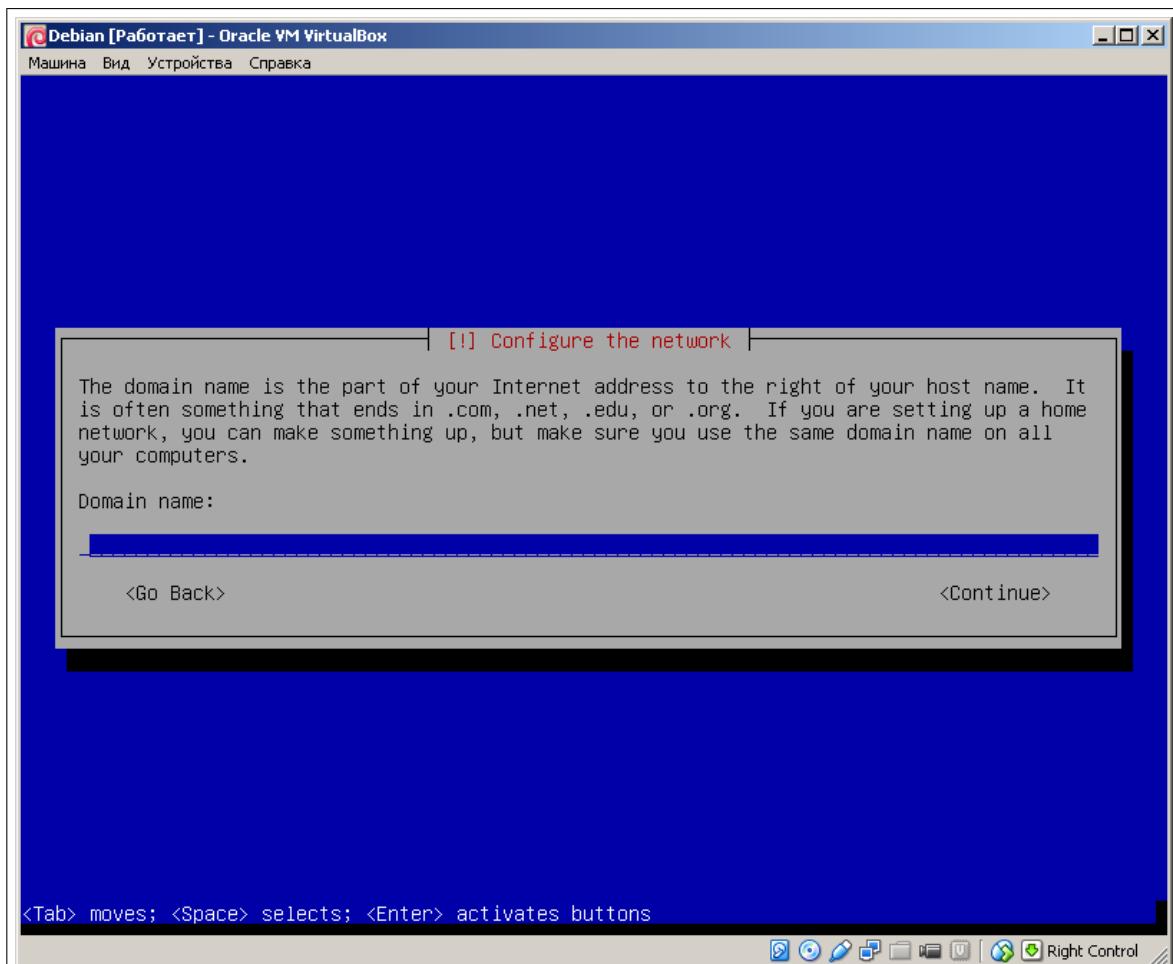
## B Установка Debian GNU/Linux в VirtualBox

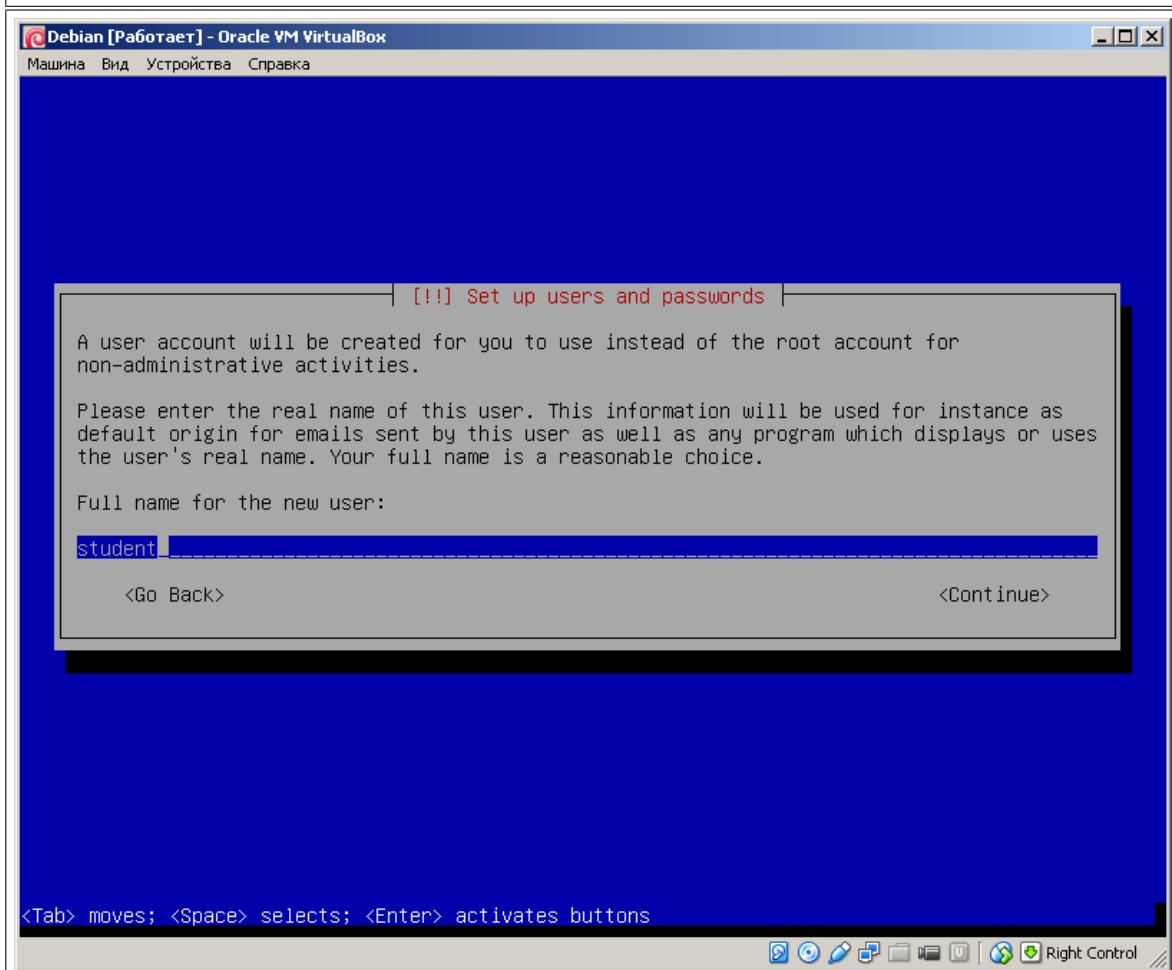
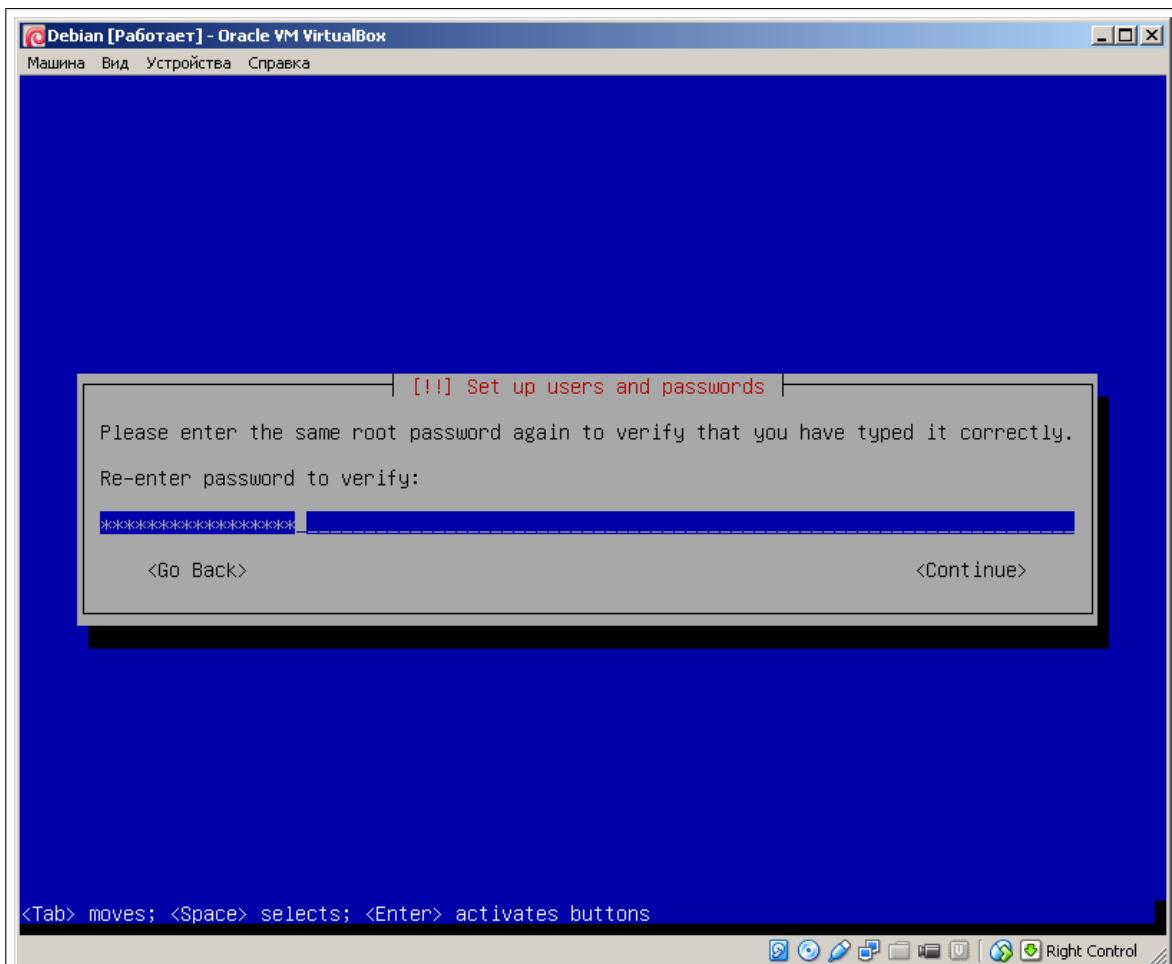


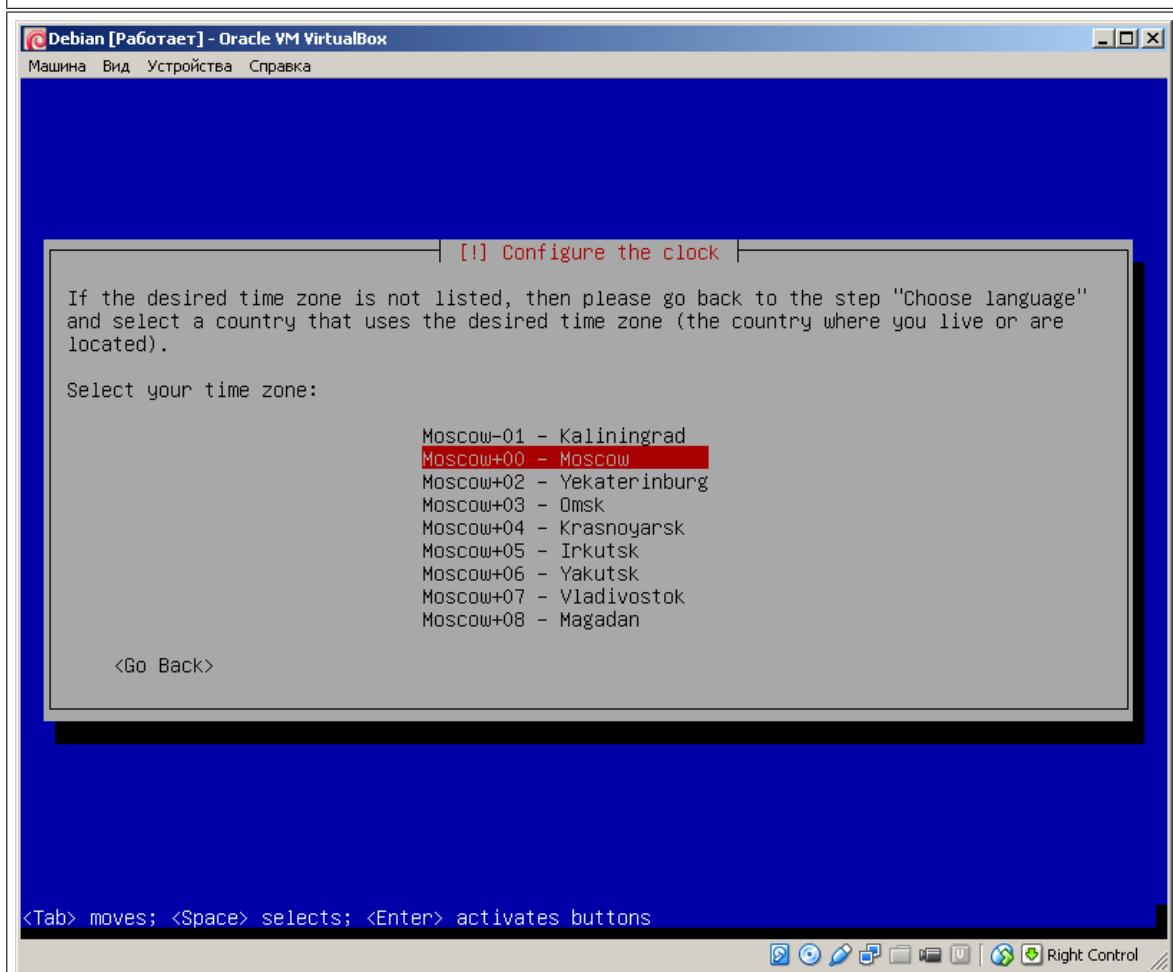
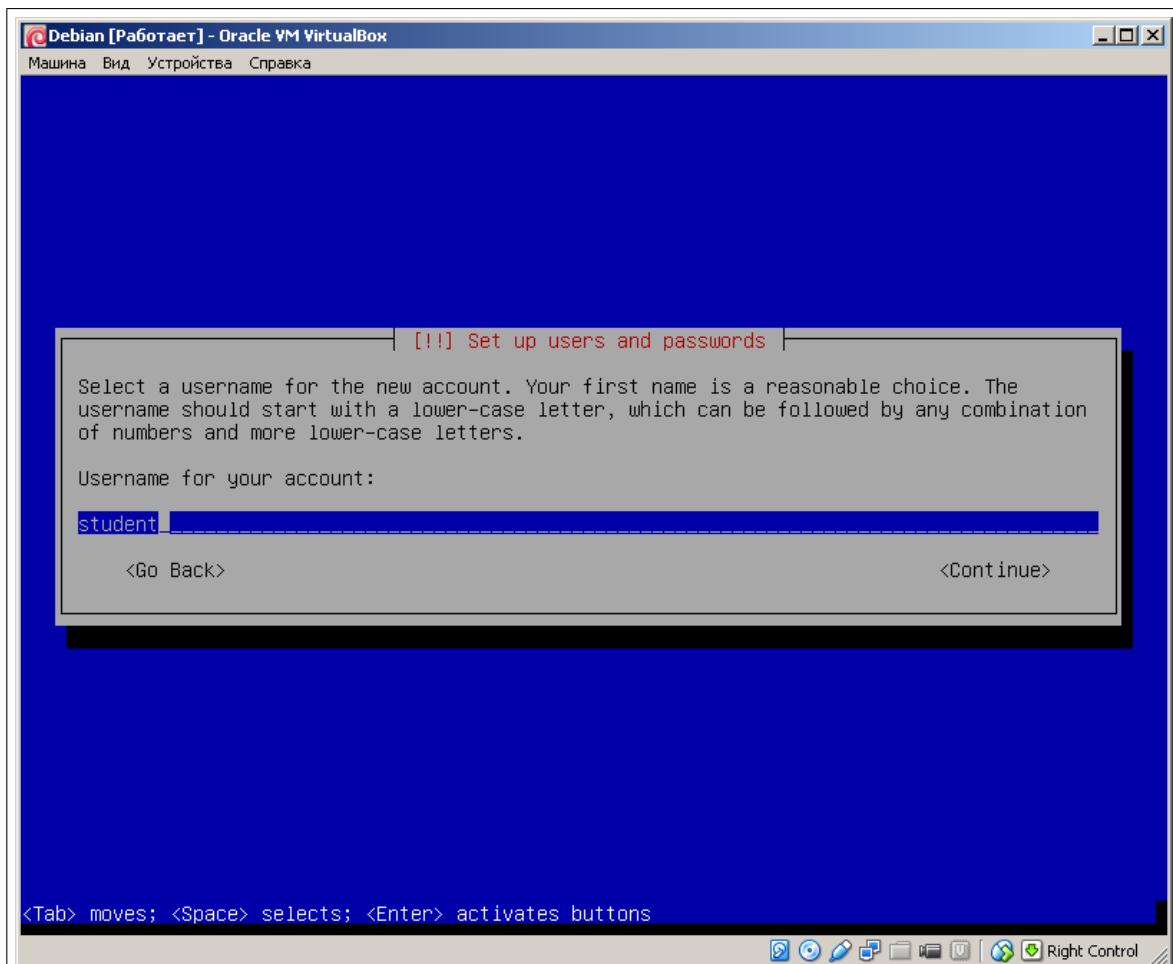


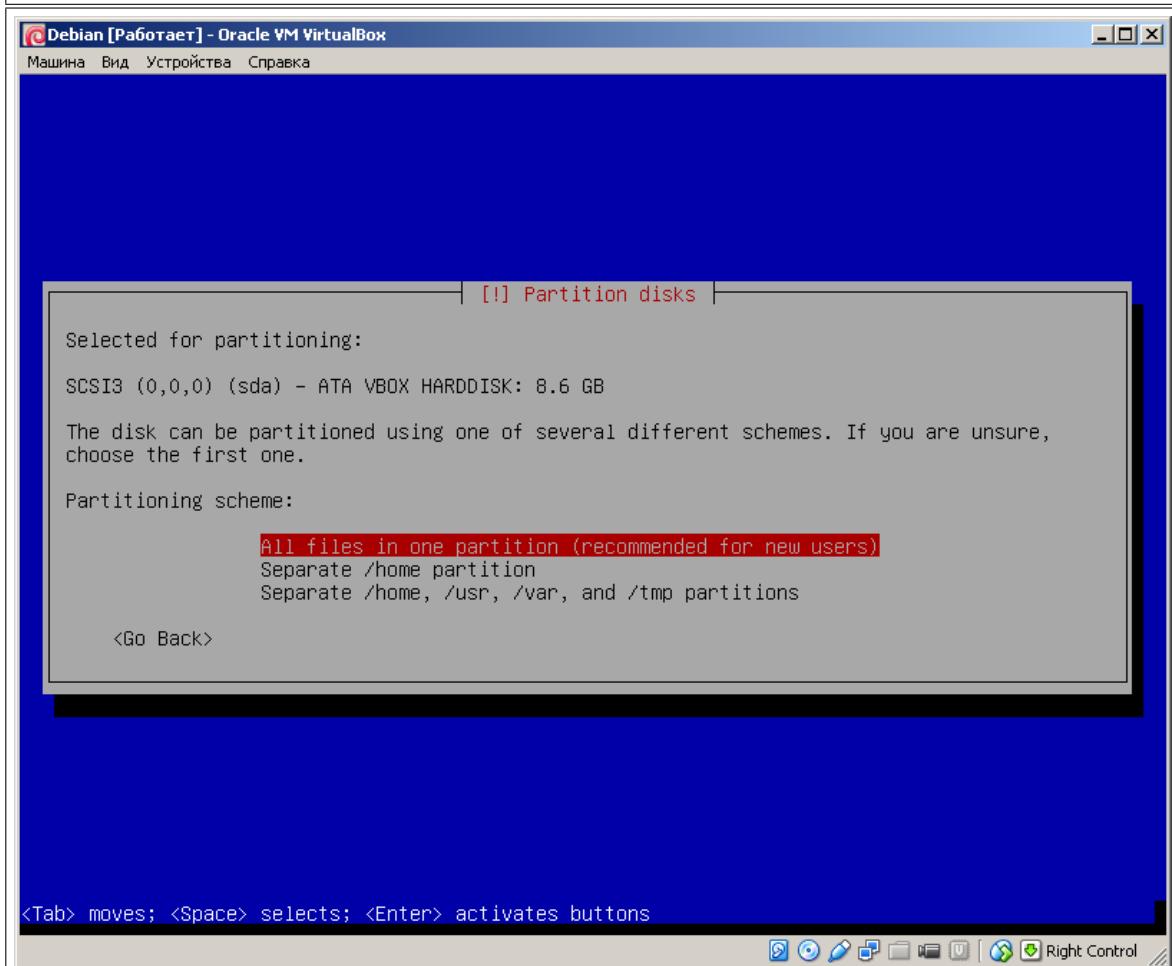
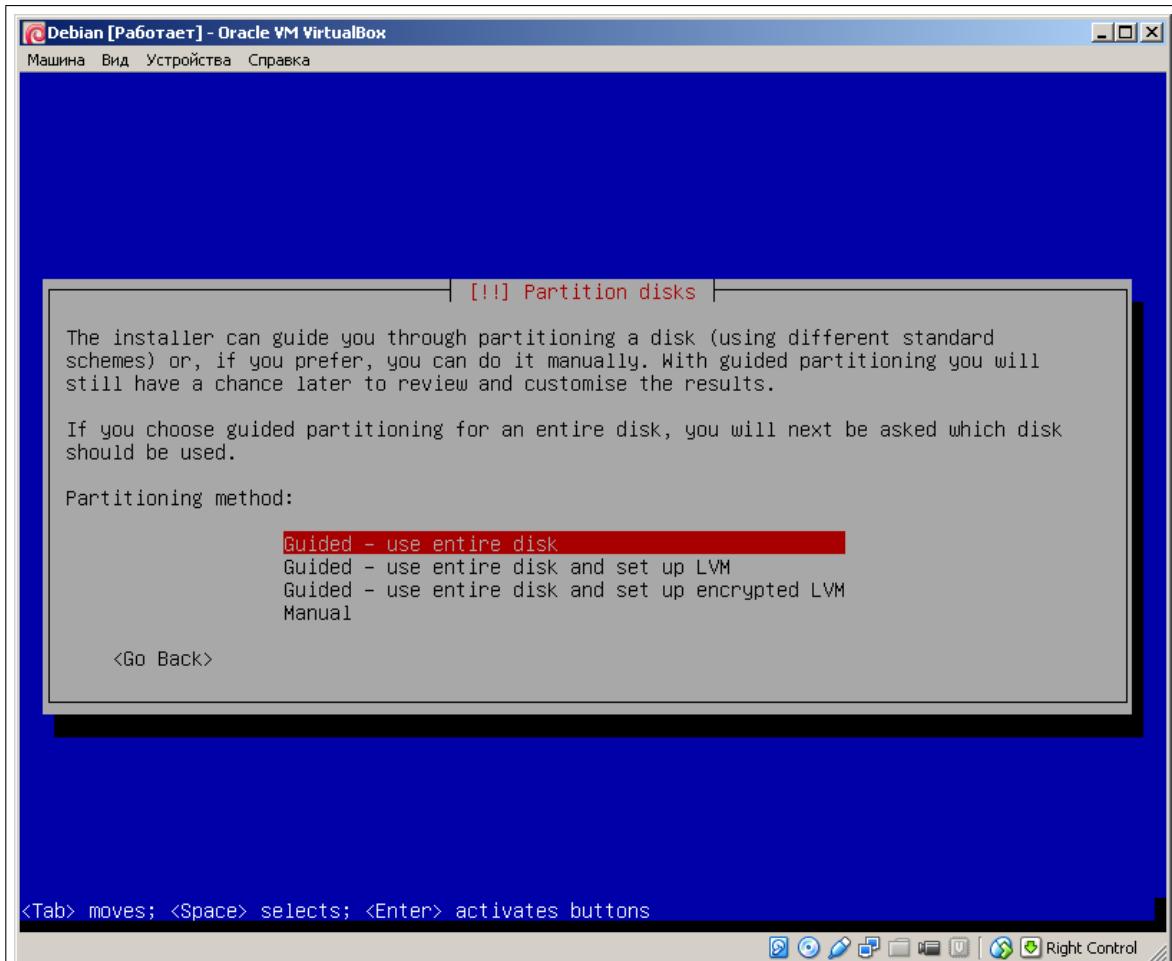


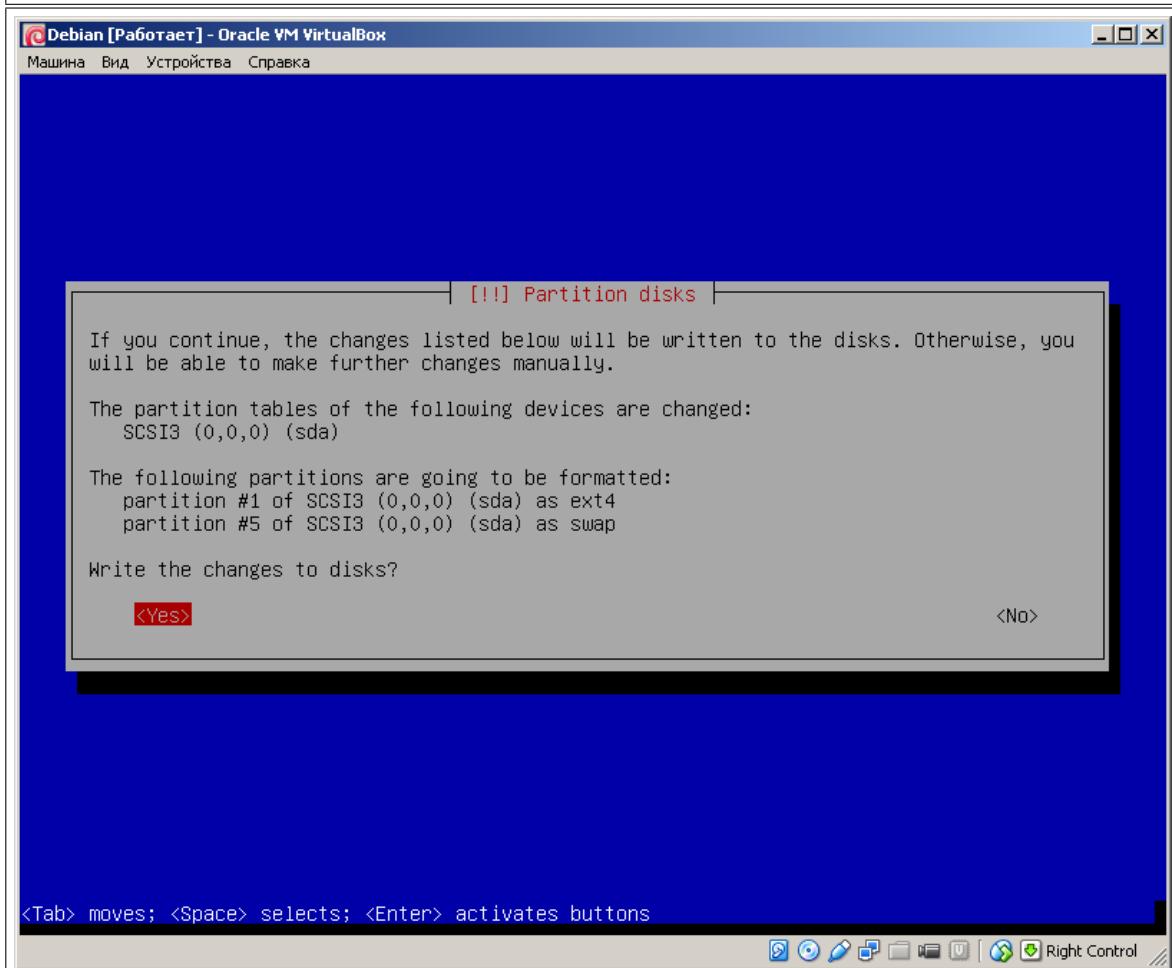
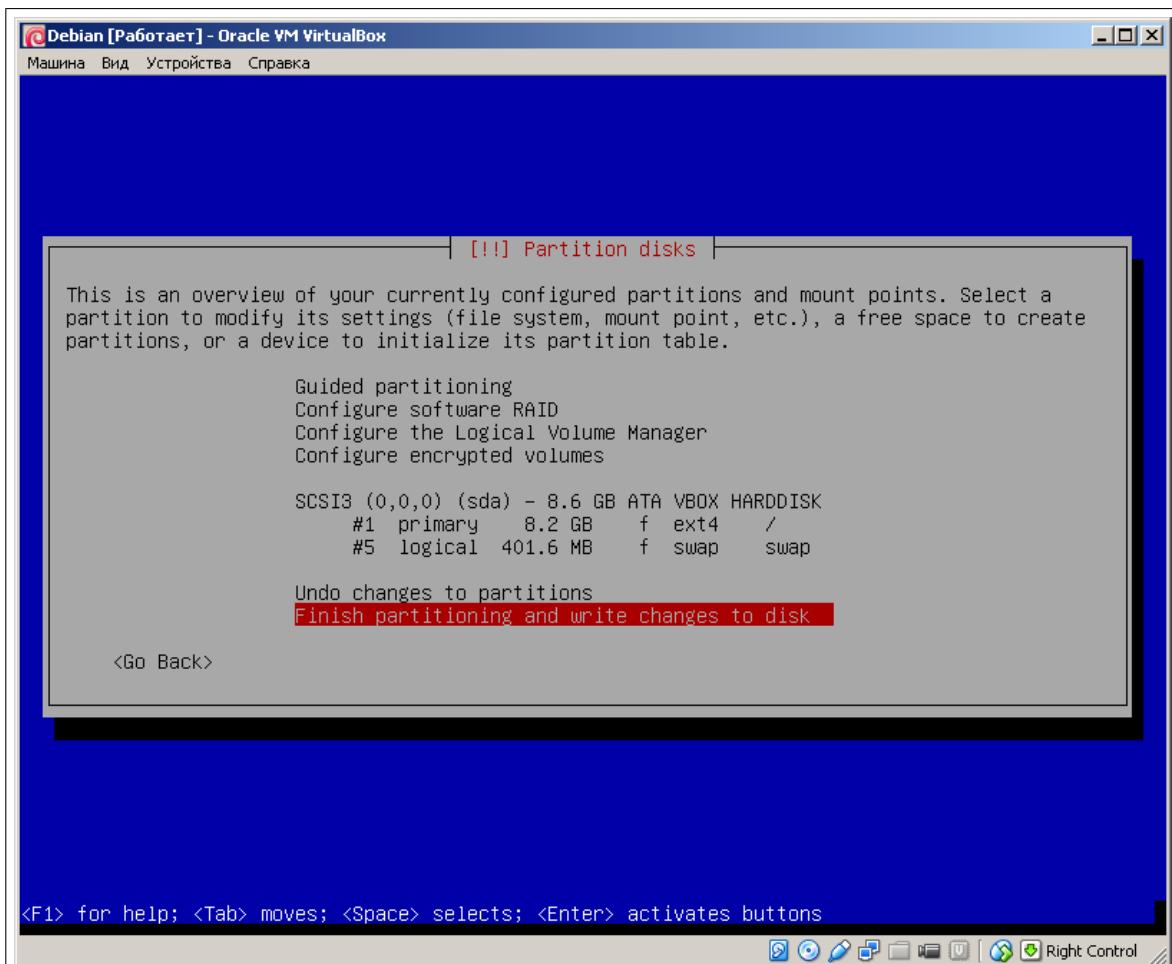


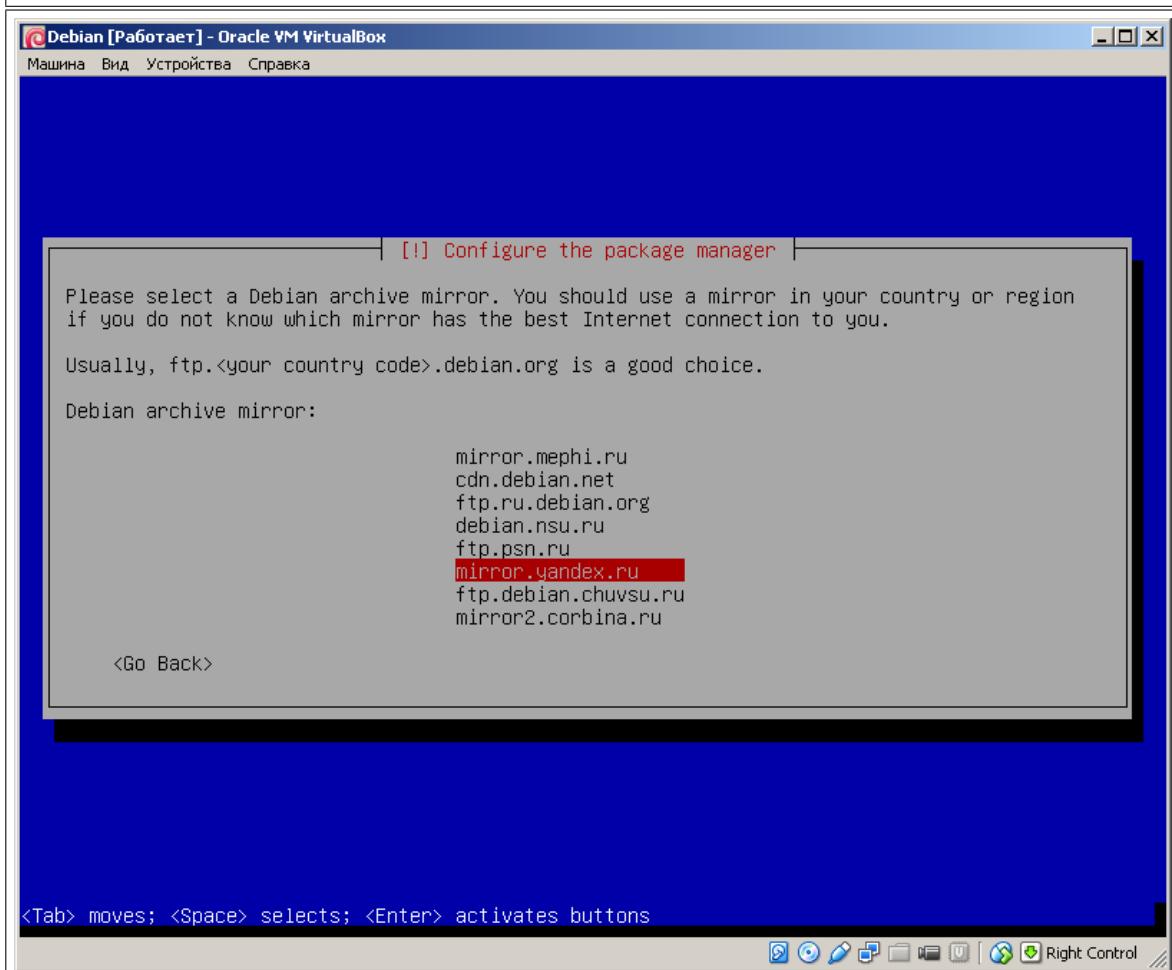
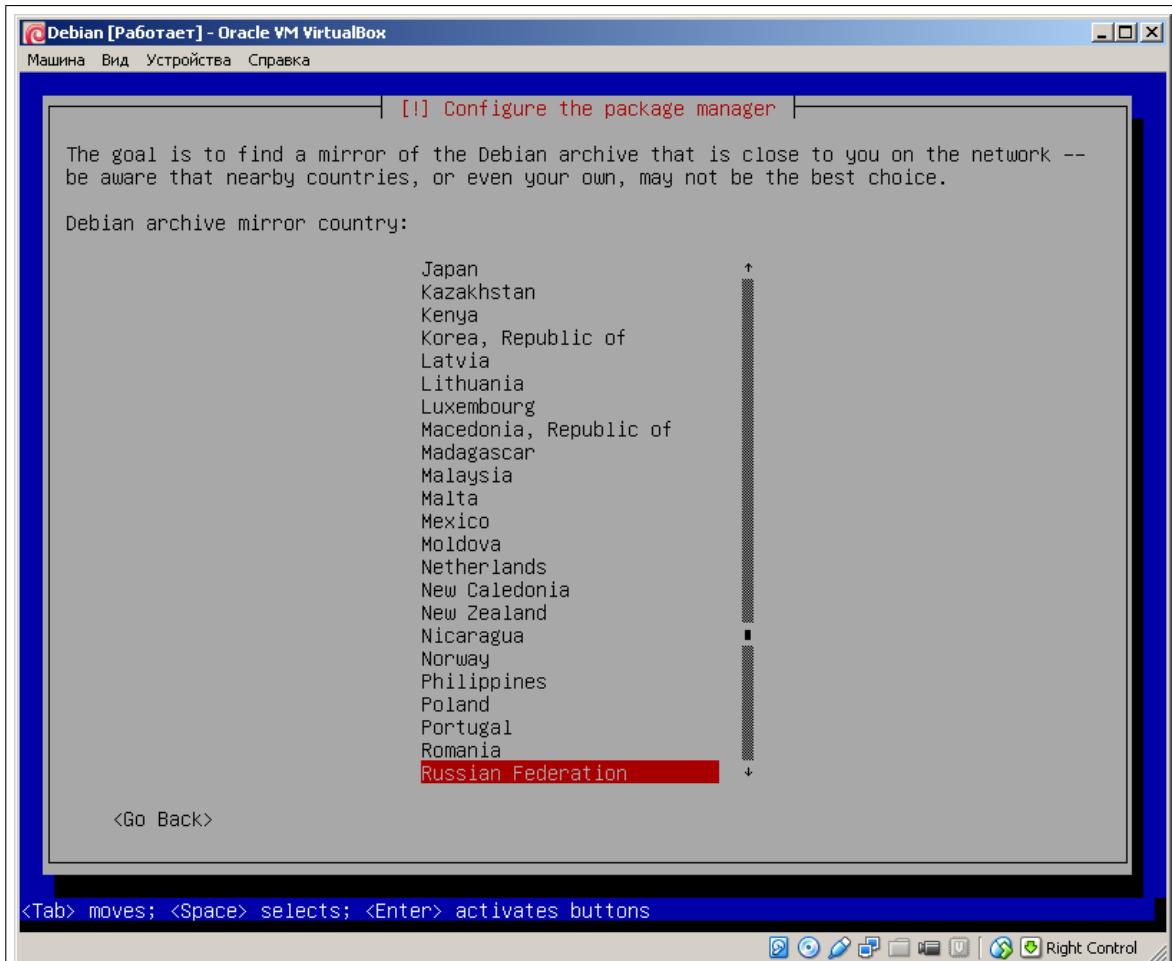


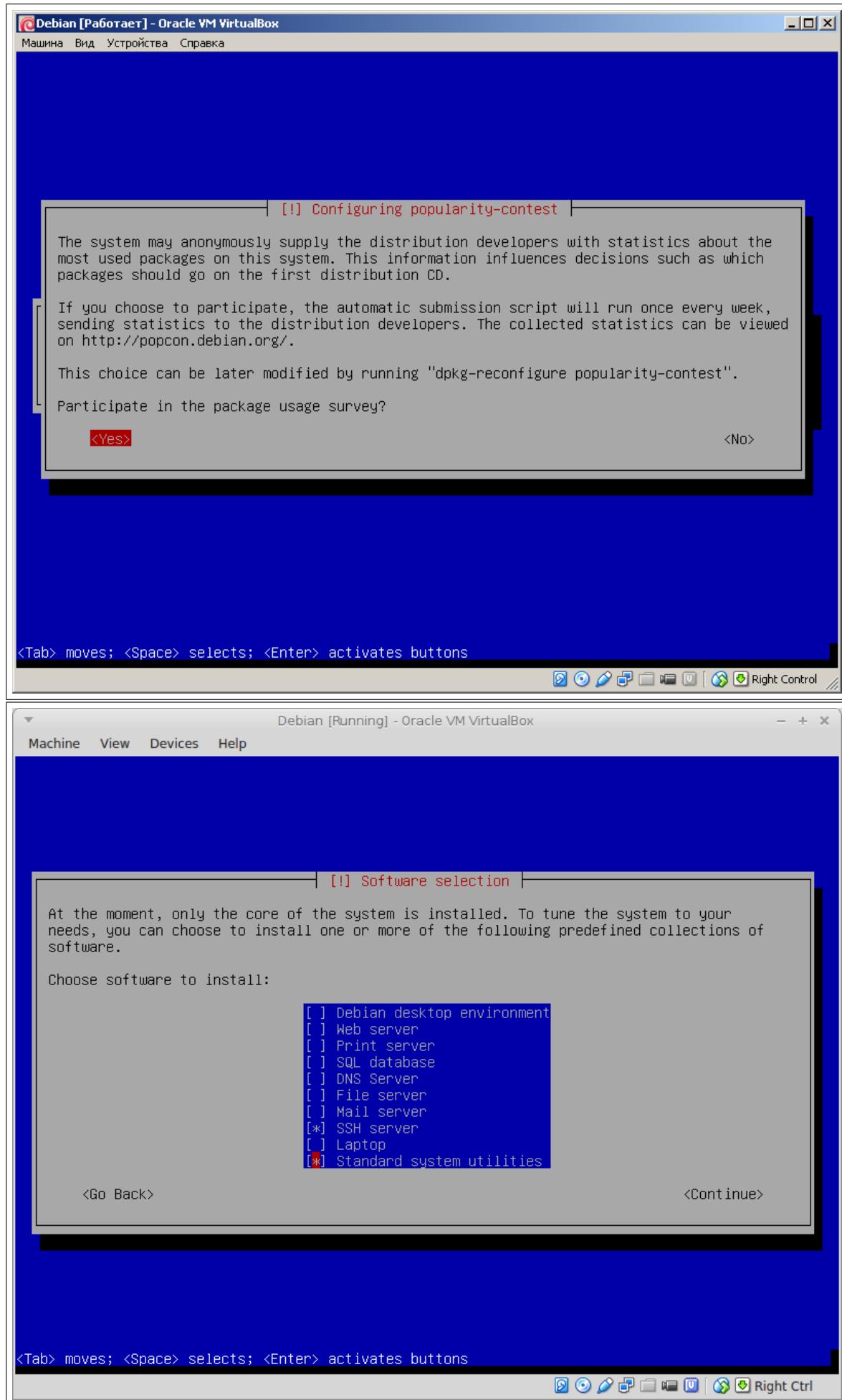


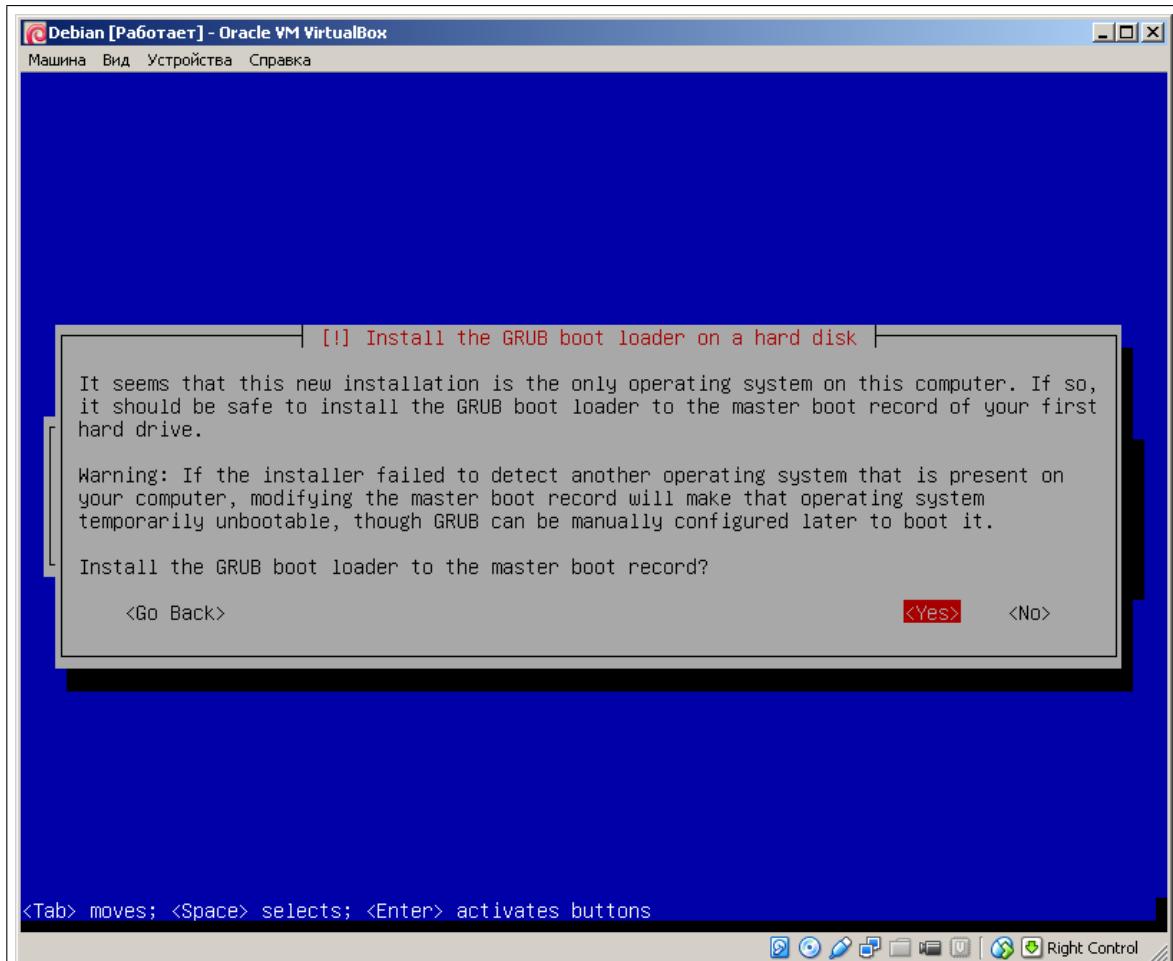






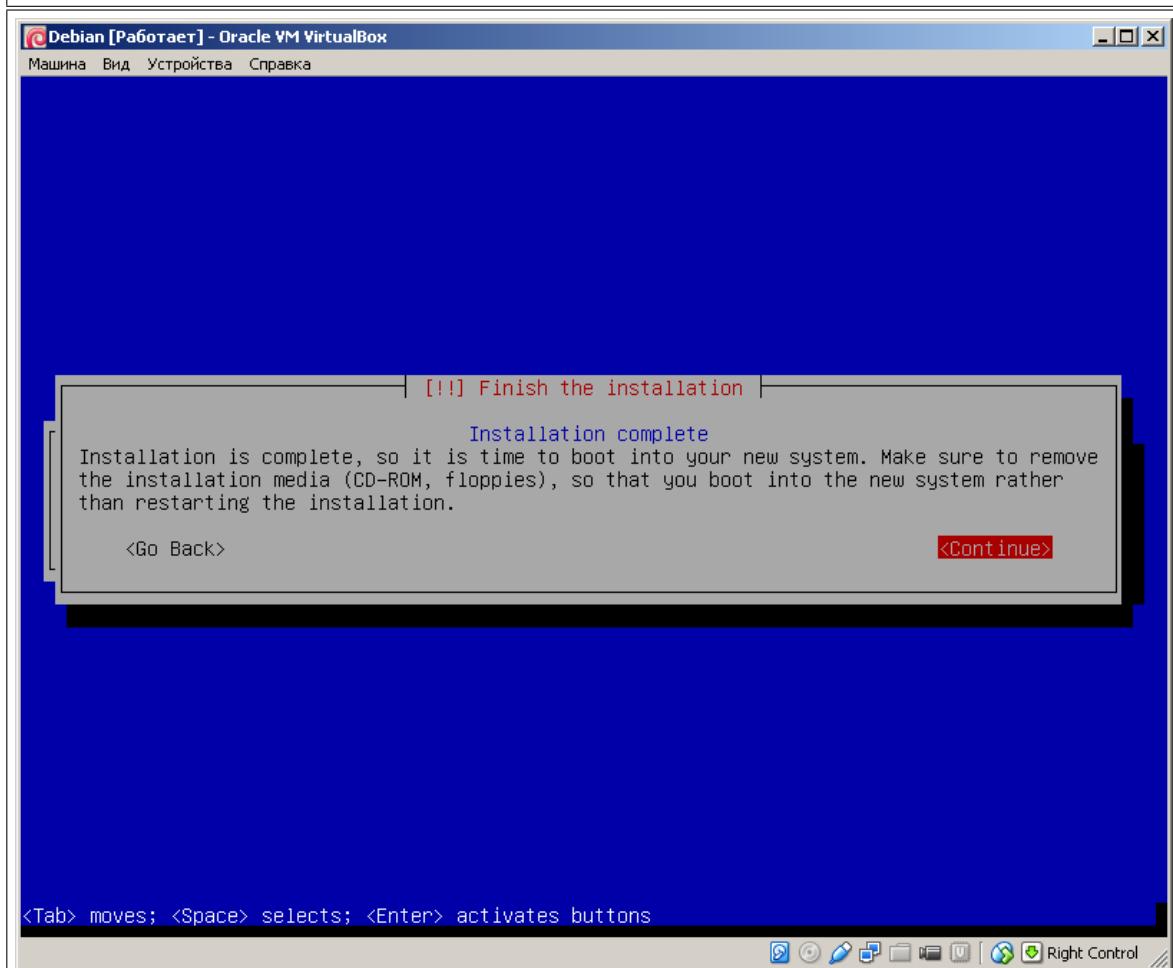






&lt;Go Back&gt;

&lt;Yes&gt; &lt;No&gt;



&lt;Go Back&gt;

&lt;Continue&gt;

&lt;Tab&gt; moves; &lt;Space&gt; selects; &lt;Enter&gt; activates buttons

Right Control

## C Установка ownCloud Server в Debian GNU/Linux

После успешного входа в систему, в первую очередь необходимо получить права суперпользователя<sup>1</sup>:

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

Добавление репозитория и ключа для ownCloud Server:

```
# wget -nv https://goo.gl/mmmV2ga -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 update
# apt 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, 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<sup>2</sup> по умолчанию, мы 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. 12).

После нажатия клавиши «Finish Setup» база данных и пользовательские настройки успешно подключаются к ownCloud (рис. 13).

<sup>1</sup>Пароль пользователя не отображается на экране во время набора

<sup>2</sup>При использовании SQLite вместо MySQL, следующий шаг можно пропустить

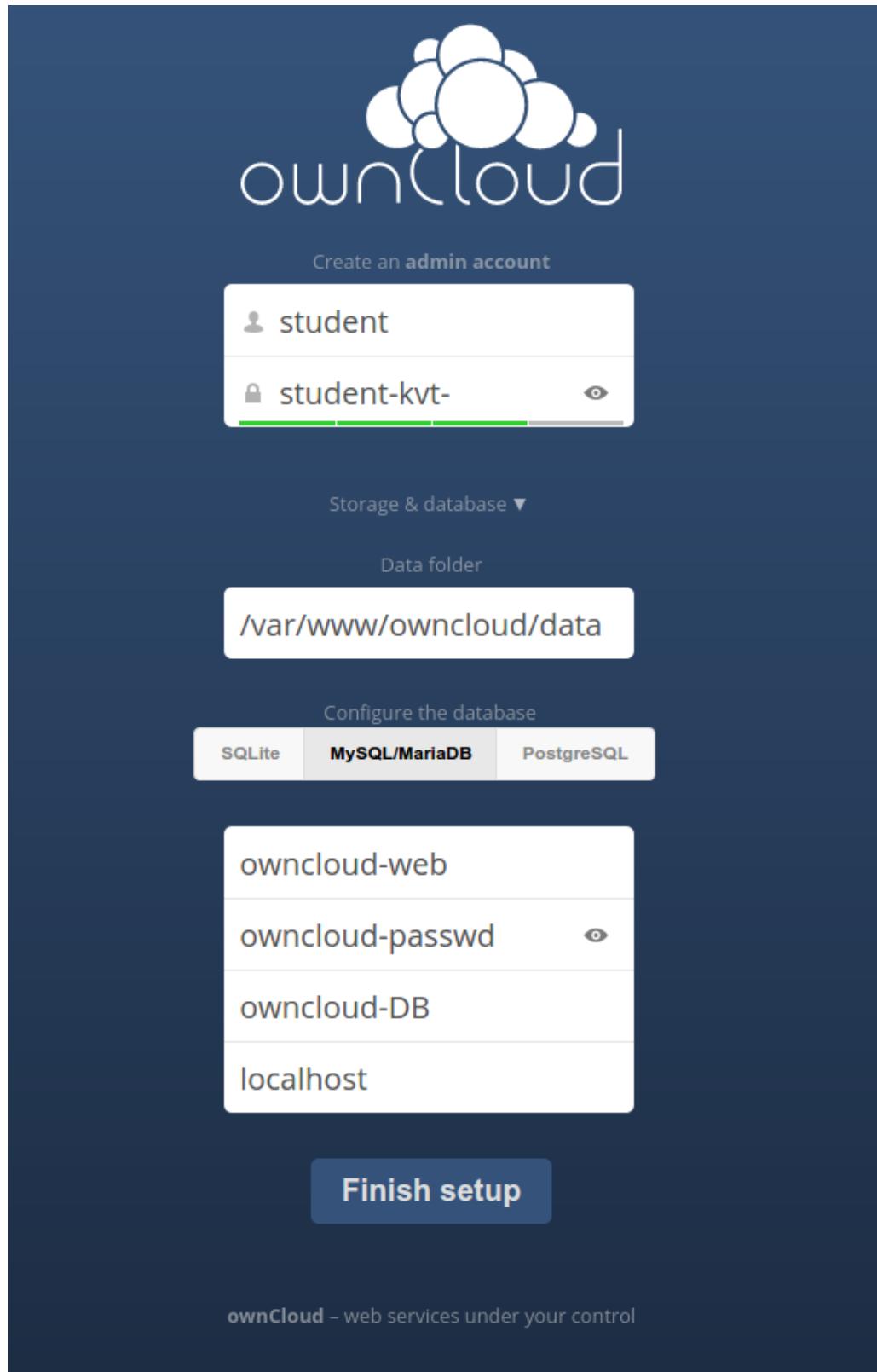


Рис. 12: Параметры БД и аккаунта администратора

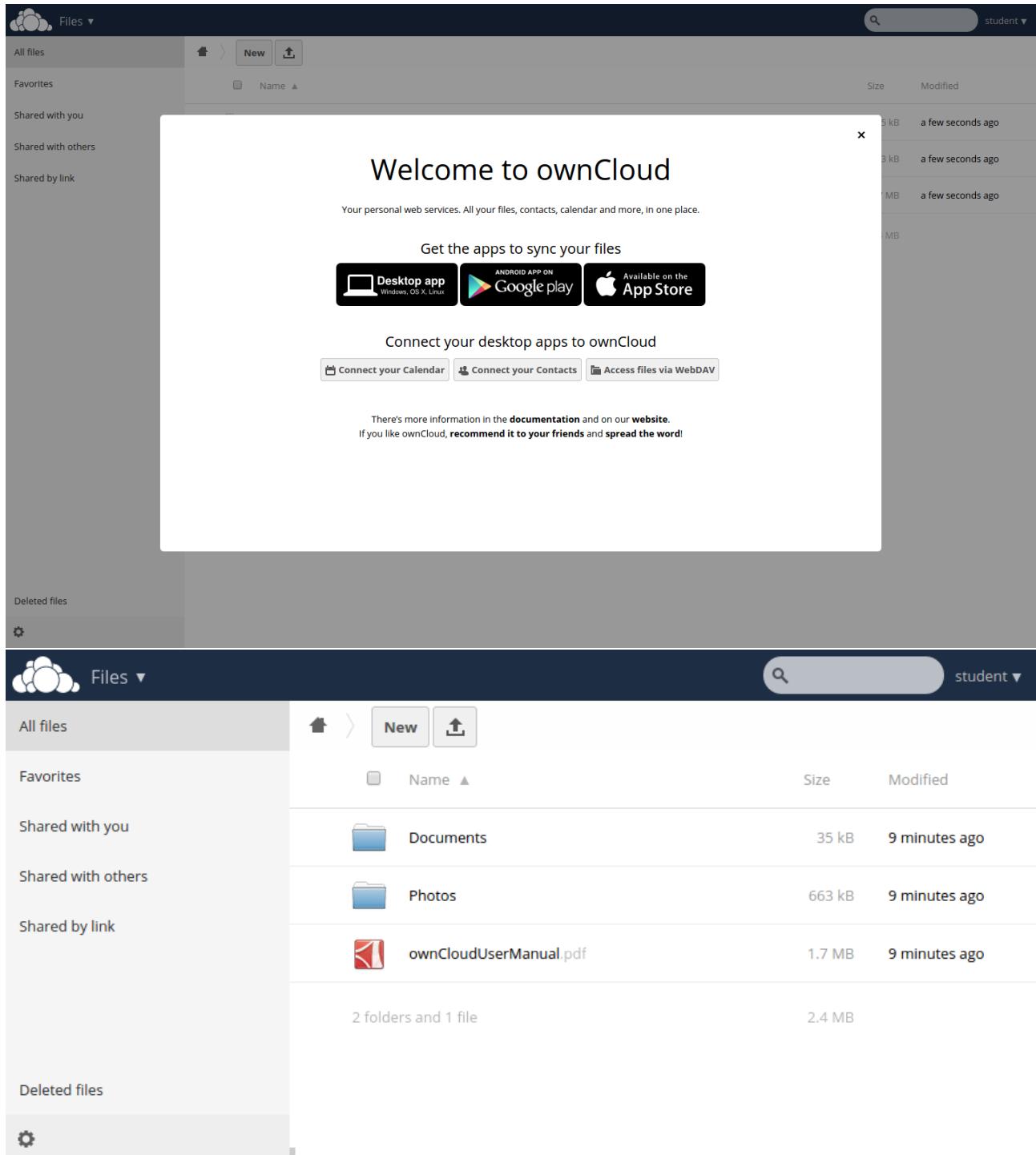
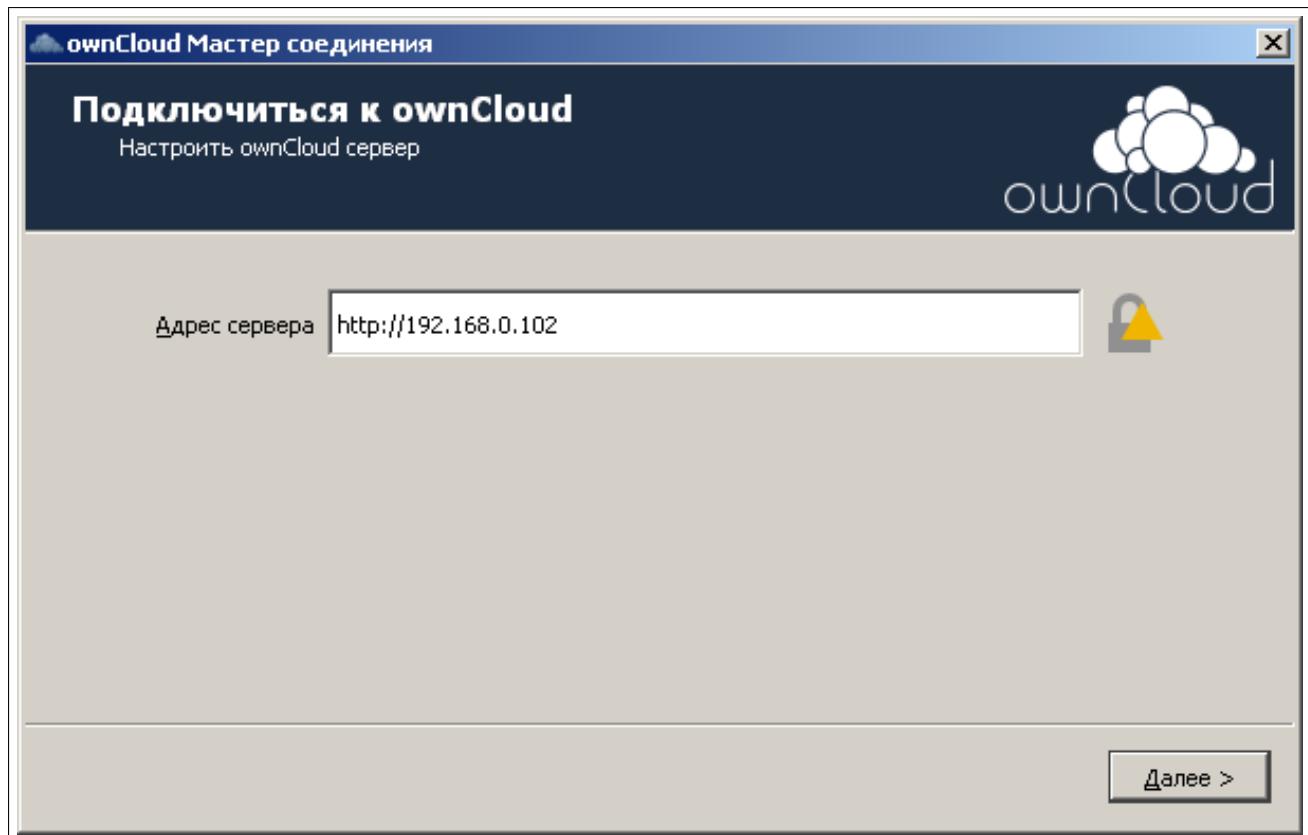
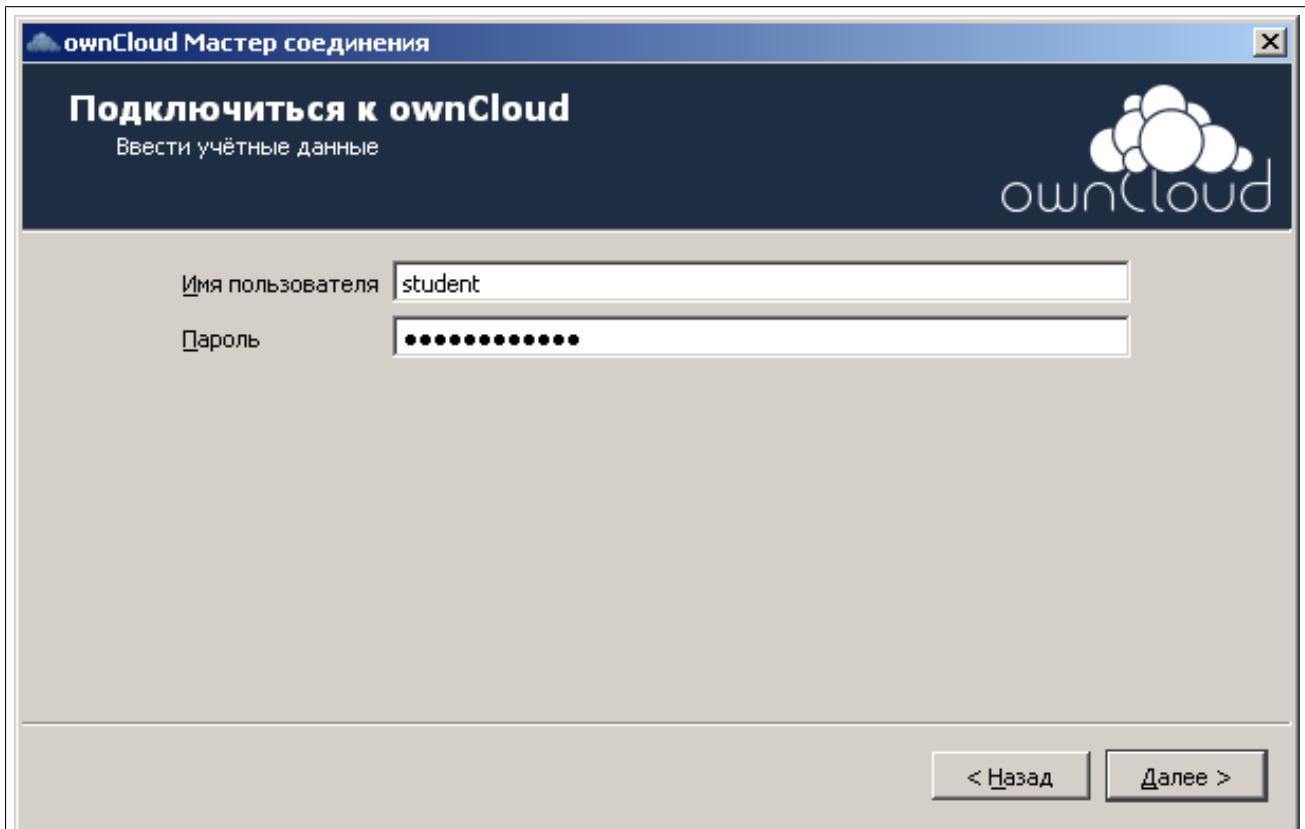
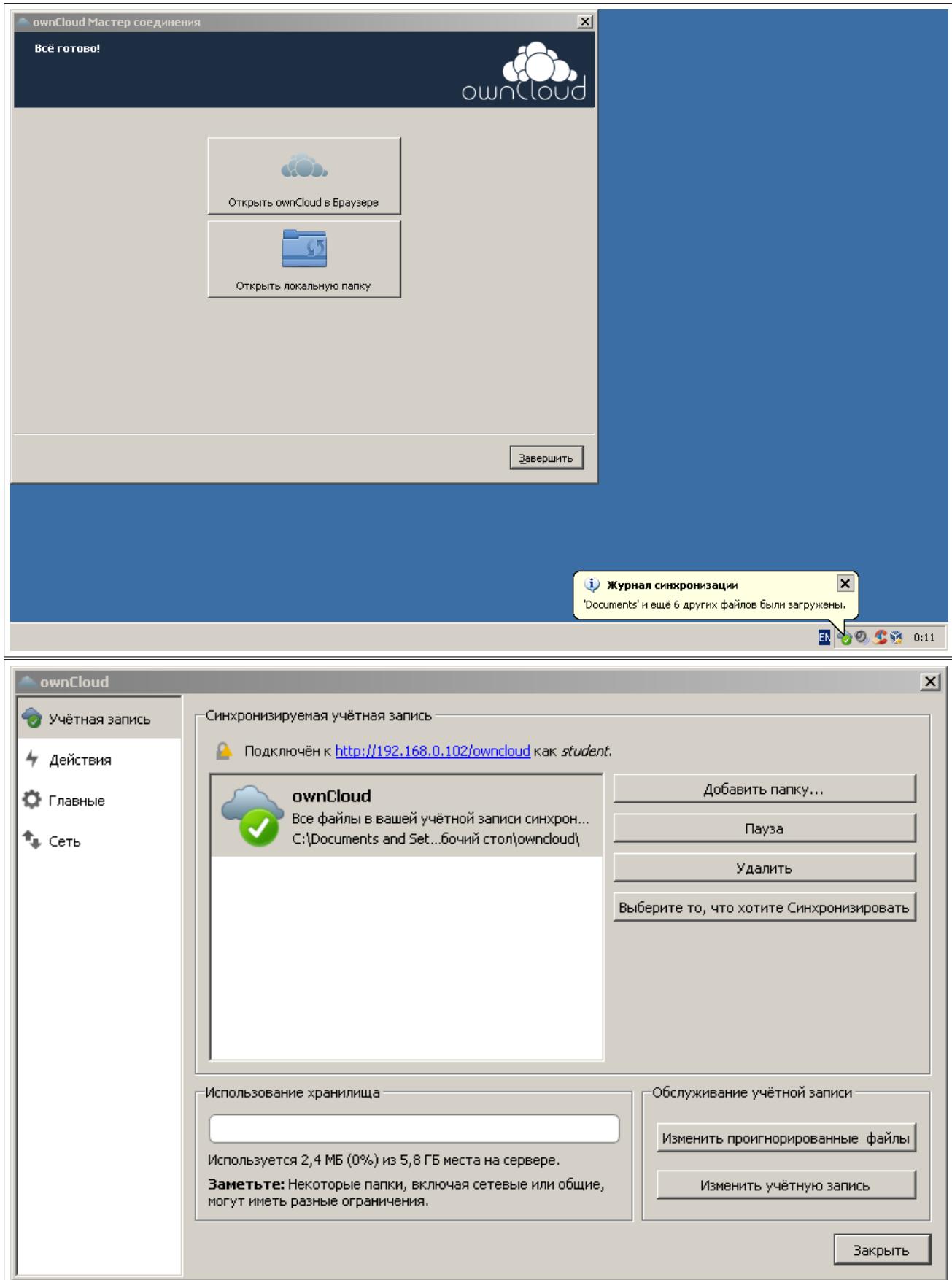


Рис. 13: Первый вход в ownCloud и интерфейс приложения

## D Подключение ownCloud Client к серверу







## E Деплой приложения на Heroku

Для регистрации в Heroku<sup>1</sup> необходима только электронная почта (рис. 14).

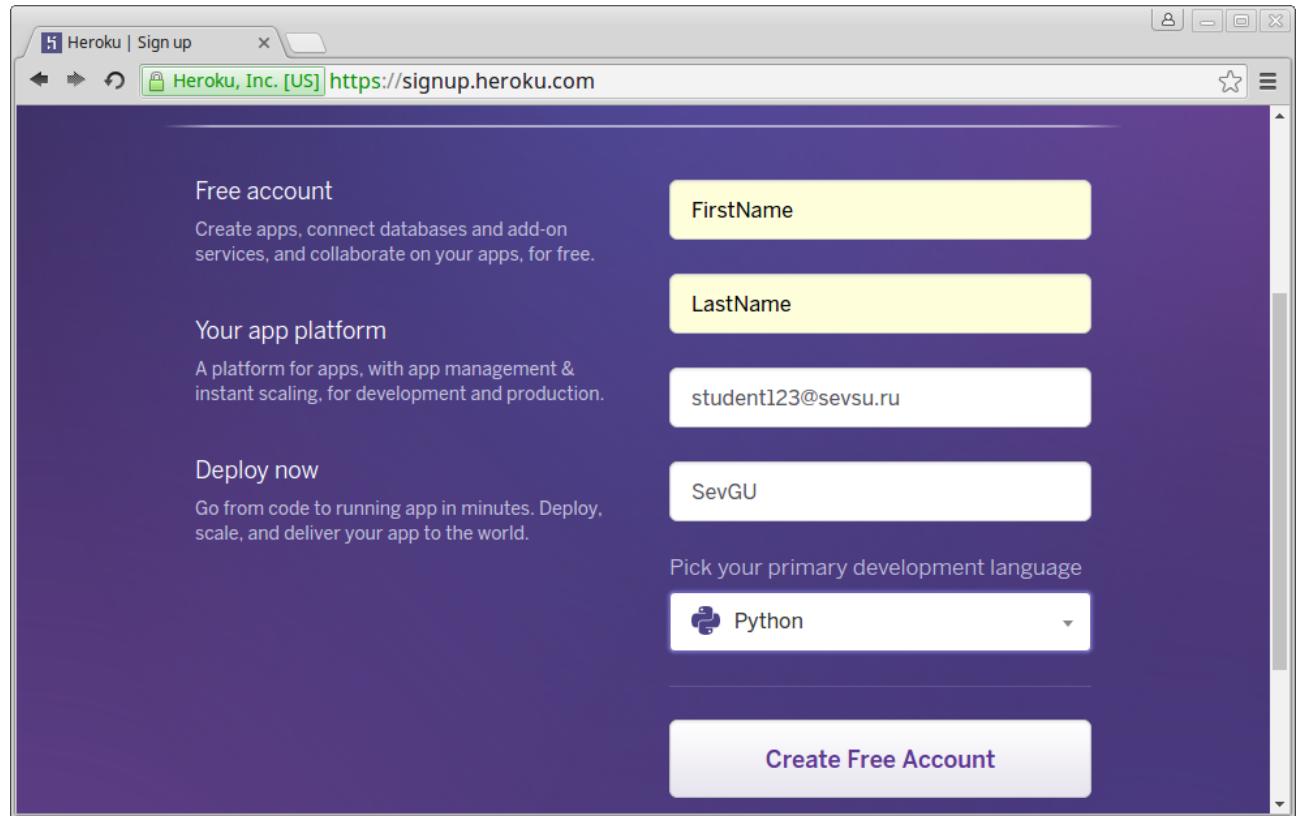


Рис. 14: Регистрация в Heroku

После регистрации, на указанный почтовый ящик приходит письмо с подтверждением, 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. 15).

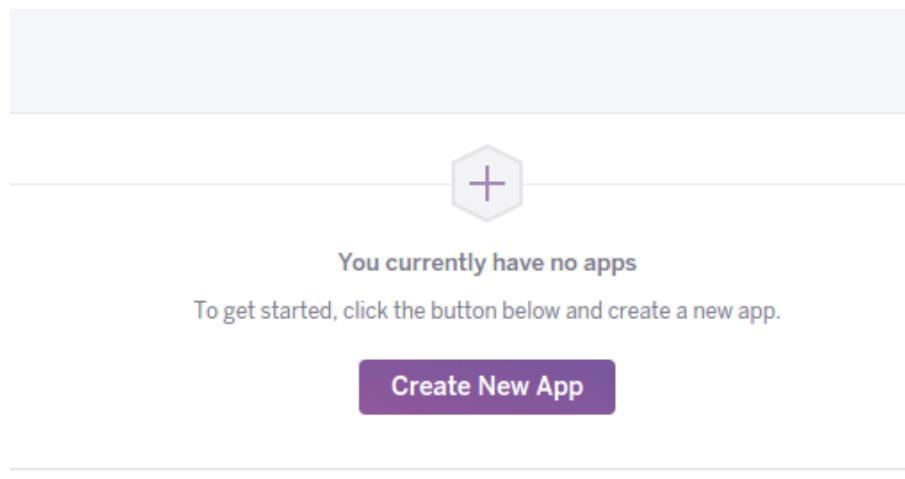


Рис. 15: Список созданных приложений в Heroku

Пусть имя приложения будет «helloivt» (рис. 16).

<sup>1</sup><https://signup.heroku.com/login>

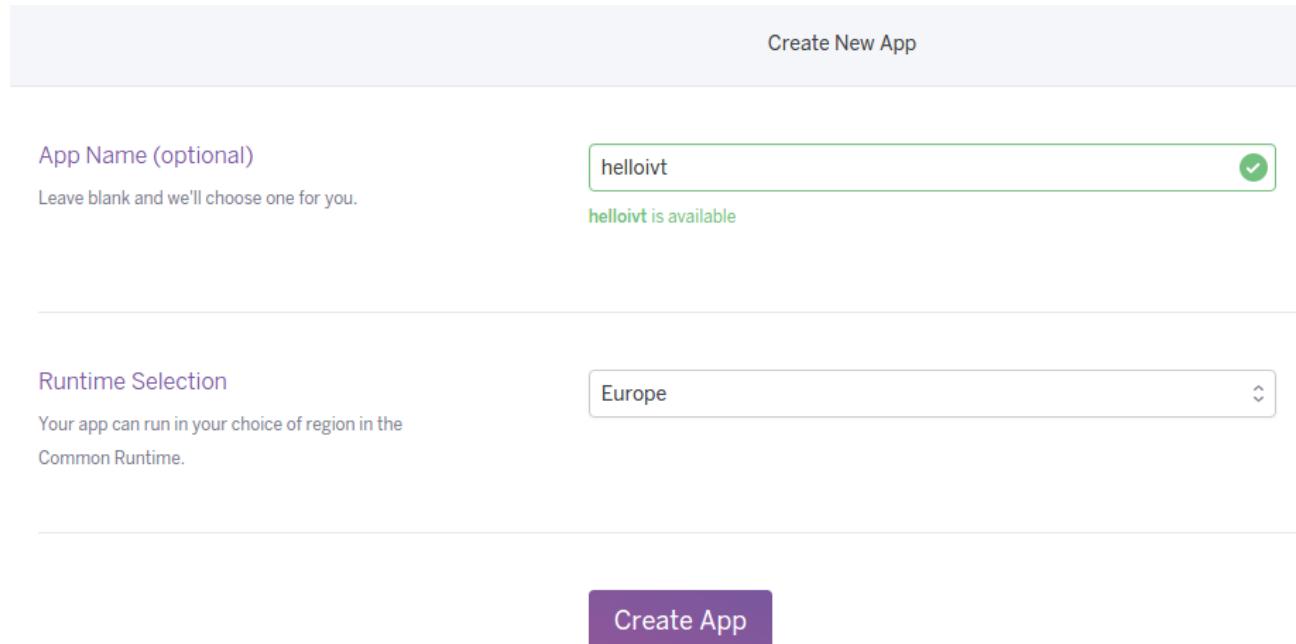


Рис. 16: Создание нового приложения

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 приложения<sup>1</sup> и проверяем, страница должна выводить надпись «Hello from Python!» (рис. 17).

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

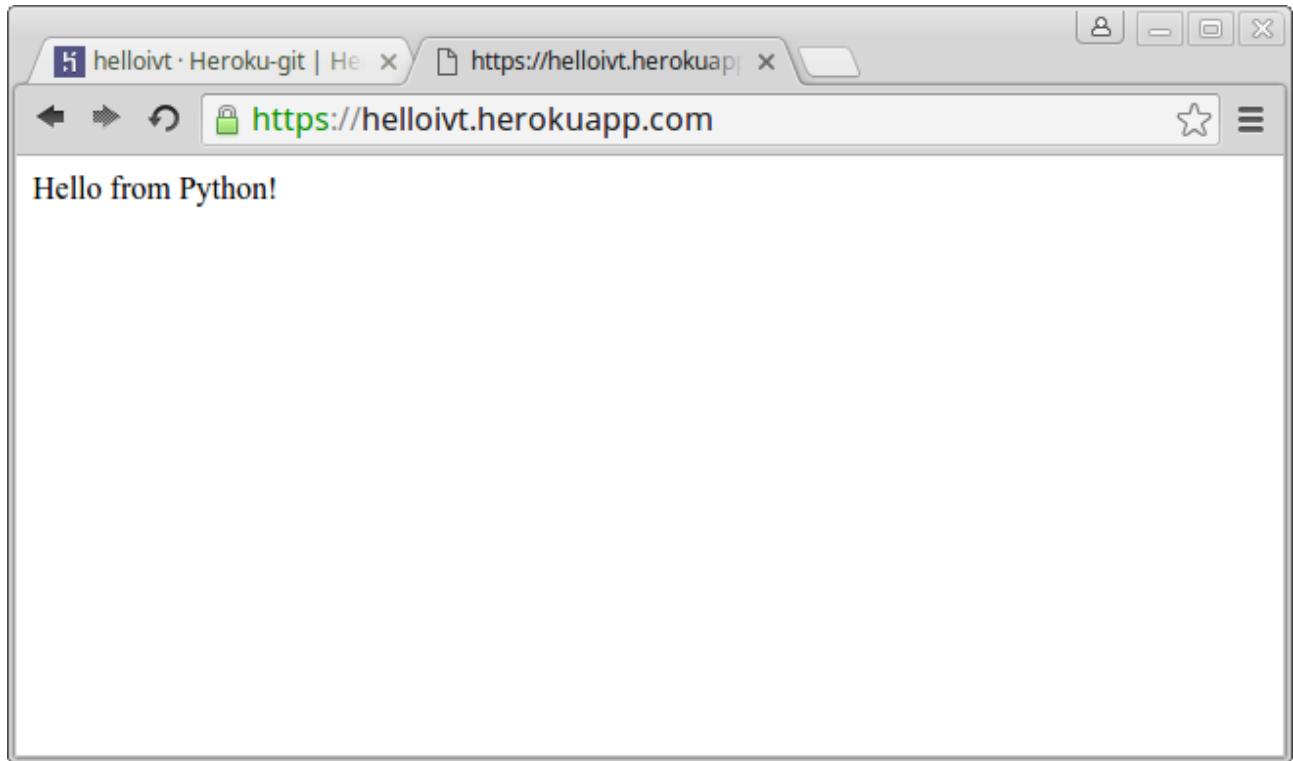


Рис. 17: Создание нового приложения

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. 18).

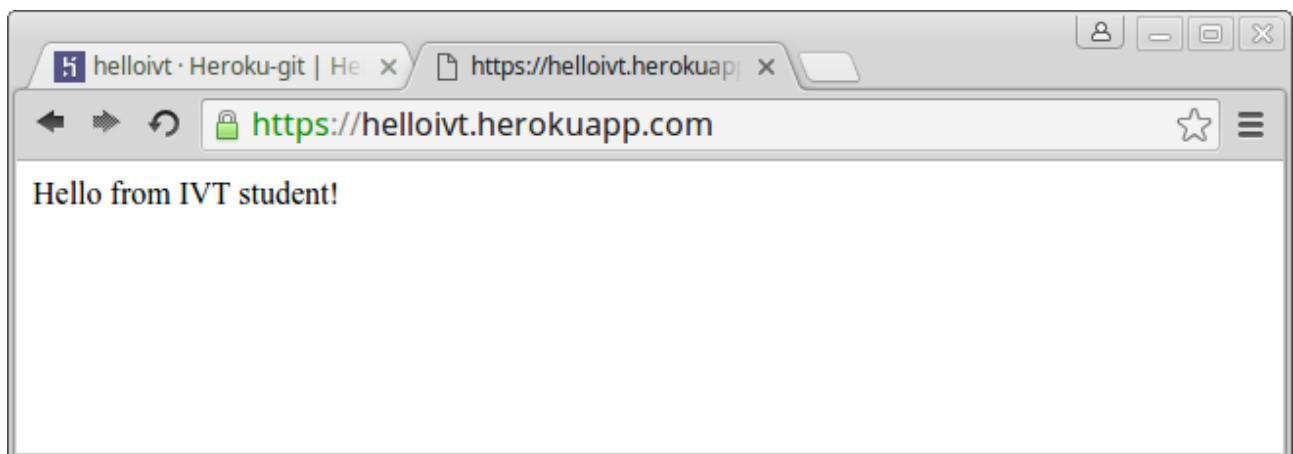


Рис. 18: Изменения в приложении вступили в силу

## F Создание инстанса в OpenStack

Сервис TryStack<sup>1</sup> позволяет использовать готовое окружение OpenStack без развертывания сложной инфраструктуры на локальных серверах.

Для регистрации на сервисе TryStack необходимо иметь аккаунт в Facebook и вступить в группу TryStack<sup>2</sup>.

После отправки заявки на вступление в закрытую группу, возможно, придется подождать несколько дней для подтверждения заявки. Когда заявка подтверждена, то можно авторизоваться в панели<sup>3</sup> TryStack с помощью аккаунта Facebook.

После авторизации можно посмотреть интерфейс управления OpenStack (рис. 19).

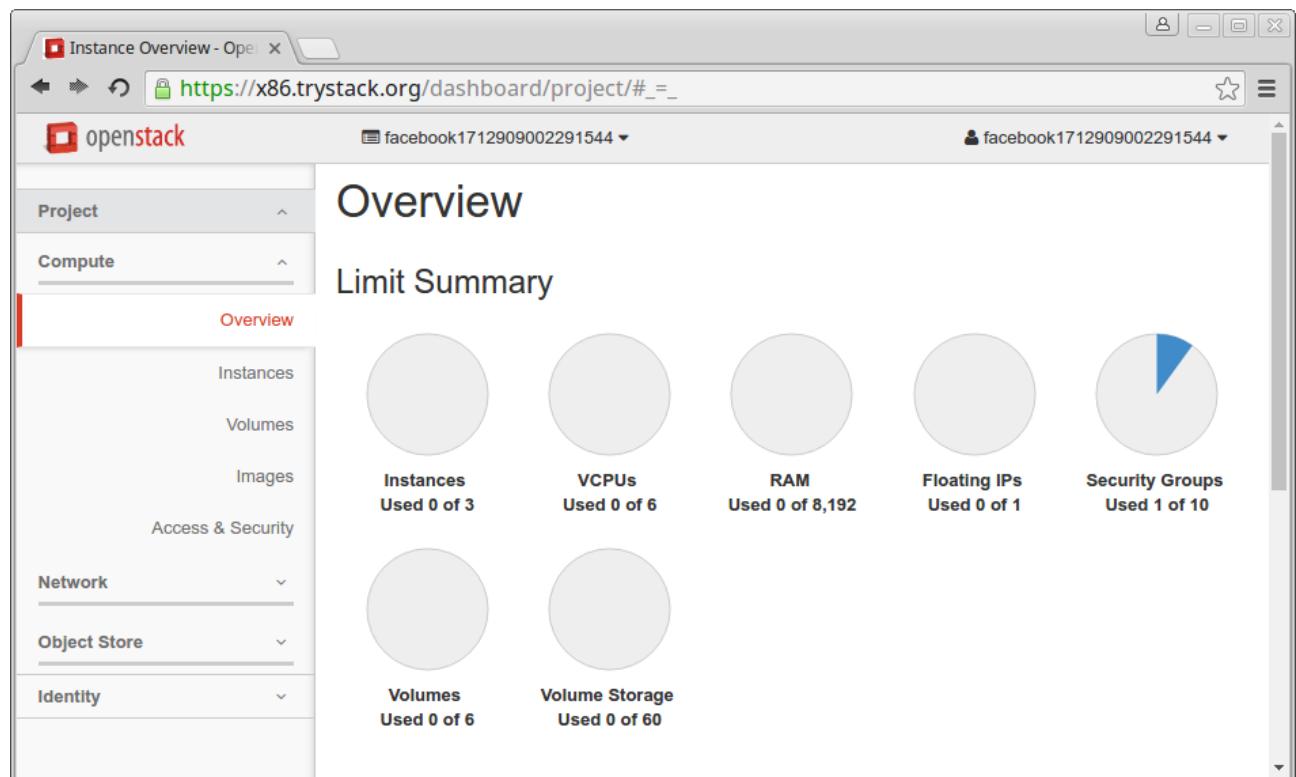


Рис. 19: Интерфейс управления OpenStack

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

Для настройки OpenStack, в первую очередь, необходимо настроить сеть.

Создадим в разделе **Network - Networks - Create Network** (рис. 20):

- 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).

<sup>1</sup><https://trystack.openstack.org>

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

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

**Create Network**

Network Subnet Subnet Details

**Network Name**  
student-net

**Admin State** UP

Create Subnet

**Create Network**

Network Subnet Subnet Details

**Subnet Name**  
student-subnet

**Network Address** 192.168.0.0/24

**IP Version** IPv4

**Gateway IP** 192.168.0.1

Disable Gateway

**Create Network**

Network Subnet Subnet Details

Enable DHCP

Specify additional attributes for the subnet.

**Allocation Pools**  
192.168.0.2,192.168.0.10

**DNS Name Servers**  
8.8.8.8  
8.8.4.4

Рис. 20: 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. 21).

**Create Router**

**Router Name \***  
student-router

**Description:**  
Creates a router with specified parameters.

**Admin State**  
UP

**External Network**  
public

Рис. 21: Создание маршрутизатора

Текущую топологию сети можно увидеть в разделе **Network - Network Topology** (рис. 22).

На схеме видно, что маршрутизатор (student-router) связан с внешней сетью (public), но не с внутренней (student-net).

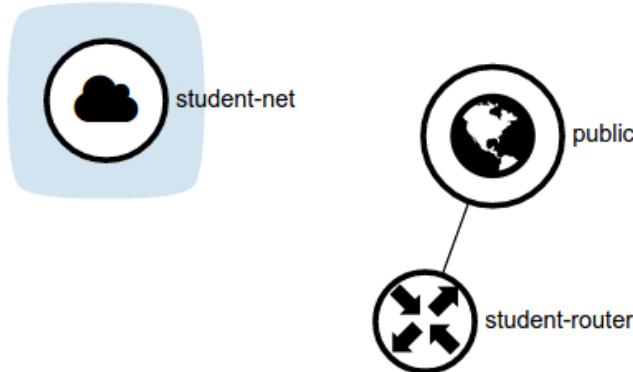
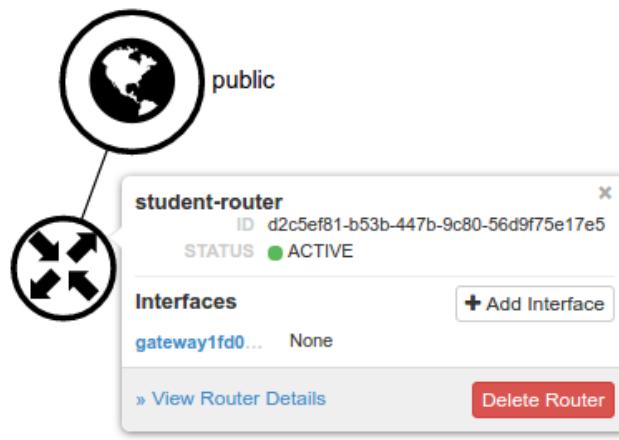


Рис. 22: Текущая топология сети

Соединяем маршрутизатор с внутренней сетью. Щелкаем мышью на маршрутизатор и выбираем пункт **Add Interface**, добавляем внутреннюю сеть (student-net) к маршрутизатору, в качестве шлюза указываем адрес 192.168.0.1 (рис. 23).

После этого можно посмотреть на новую топологию сети (рис. 24). На этой топологии видно, что 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. 23: Подключение внутренней сети к маршрутизатору

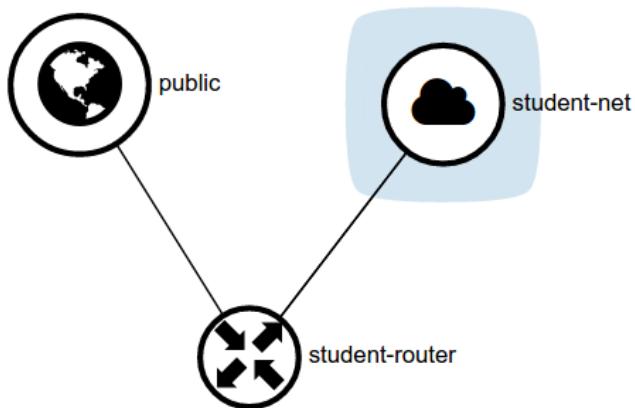


Рис. 24: Маршрутизатор соединяет внешнюю и внутреннюю сети

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 файл student123.pem, 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 удается, в таком случае необходимо попробовать запросить адрес позже, предварительно обновив веб-страницу, это связано с ограниченным пулем IP-адресов, поэтому для всех пользователей TryStack таких адресов может не хватать.

После получения внешнего адреса, его можно увидеть в списке (рис. 25).

## Access & Security

The screenshot shows the 'Access & Security' interface with the 'Floating IPs' tab selected. A message at the top right says 'Allocate IP To Project (Quota exceeded)'. Below is a table with one item:

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

Displaying 1 item

Рис. 25: Список выделенных внешних IP-адресов

Далее необходимо создать группу безопасности, в которой можно настроить правила фильтрации трафика для инстансов. 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. 26).

Аналогично создадим разрешающие правила для TCP и UDP (рис. 27).

## 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. 26: Разрешающее правило для работы ICMP

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

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

Displaying 5 items

Рис. 27: 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 (2048МВ RAM, 20GB HDD). Имя инстанса student-instance (рис. 28).

В разделе **Access & Security** укажем пару ключей student123 и группу безопасности my-secgroup. В разделе **Networking** выберем сеть student-net.

Рис. 28: Создание нового инстанса

После этого можно видеть запущенный инстанс в общем списке (рис. 29), ему присвоен внутренний адрес 192.168.0.3.

## Instances

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

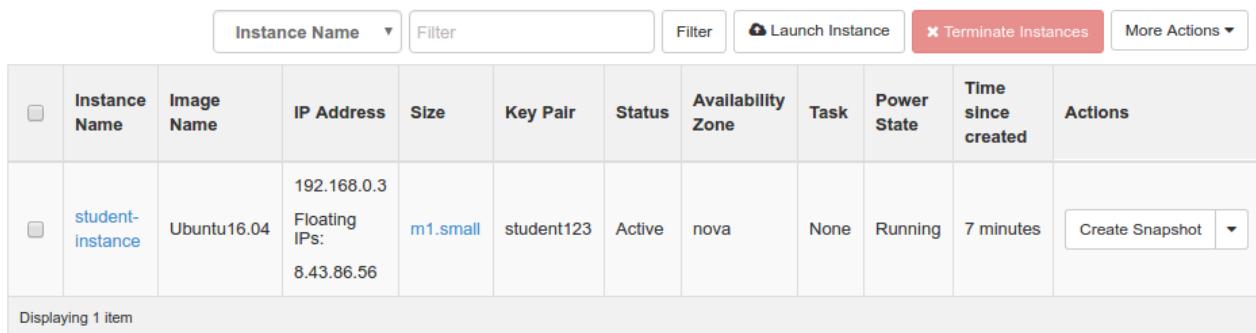
Displaying 1 item

Рис. 29: 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. 30).

## Instances



	Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
<input checked="" 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	<button>Create Snapshot</button>

Displaying 1 item

Рис. 30: 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.

```
$ sudo apt update && sudo apt install nginx
$ sudo echo "<h1>Hello, this instance created by OpenStack.</h1>" > \
> /var/www/html/index.nginx-debian.html
$ sudo 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. 31).

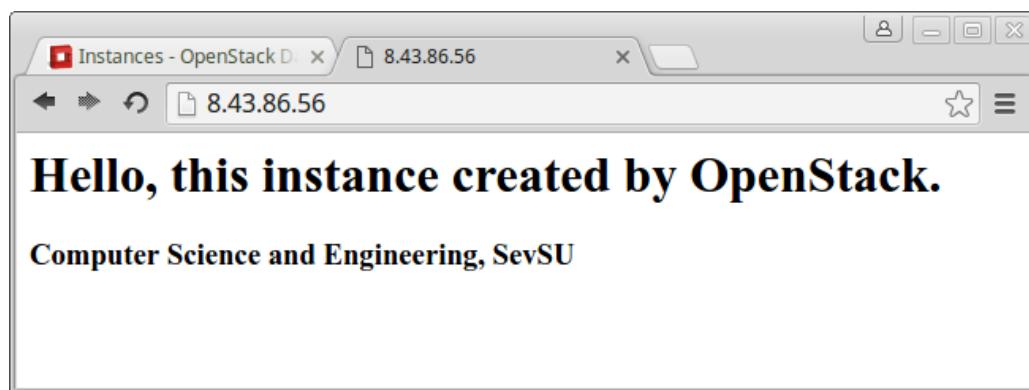


Рис. 31: Демонстрация работы веб-сервера в облаке