**РЕФЕРАТ**

Выпускная квалификационная работа магистра (магистерская диссертация) содержит 68 страниц, 14 рисунков, 1 таблицу, 16 использованных источников.

БАЛАНСИРОВКА РЕСУРСОВ, ВИРТУАЛИЗАЦИЯ, ВИРТУАЛЬНАЯ МАШИНА, ГИПЕРВИЗОР, КЛАСТЕР, РАЗВЁРТЫВАНИЕ ВИРТУАЛЬНОЙ МАШИНЫ, OPENNEBULA

Выпускная квалификационная работа магистра посвящена исследованию и модернизации механизмов развёртывания виртуальных машин в виртуальной инфраструктуре с удалёнными хостами с целью повышения скорости развёртывания, а также обеспечения безопасности виртуальной инфраструктуры.

В теоретической части были рассмотрены основные виды развёртывания виртуальных машин и основные проблемы, которые могут возникать во время развёртывания на хосте, были рассмотрены различные платформы виртуализации, были выявлены их сильные и слабые стороны. Также был проведён анализ общих проблем безопасности и возможный путей их решения.

В практической части была изучен исходной код OpenNebula, после чего на основе полученной теоретической информации для системы виртуализации OpenNebula было написано и протестировано решение, позволяющее выбирать оптимальный хост для развёртывания виртуальной машины.

**СОДЕРЖАНИЕ**

[ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ 5](#_Toc1)

[ВВЕДЕНИЕ 7](#_Toc2)

[1 ОСНОВНЫЕ ПРИНЦИПЫ РАБОТЫ ВИРТУАЛИЗАЦИИ И МЕХАНИЗМЫ ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ В LINUX 10](#_Toc3)

[1.1 Технологии виртуализации 10](#_Toc4)

[1.2 Обеспечение безопасности в Linux 11](#_Toc5)

[2 ОСНОВНЫЕ ПРИНЦИПЫ ХРАНЕНИЯ И УПРАВЛЕНИЯ СЕКРЕТАМИ 26](#_Toc6)

[2.1 Секрет и методы его защиты 27](#_Toc7)

[2.2 Hashicorp Vault 29](#_Toc8)

[2.3 Keycloak 32](#_Toc9)

[3 ИМЕЮЩИЕСЯ ПЛАТФОРМЫ ВИРТУАЛИЗАЦИИ 36](#_Toc10)

[3.1 VMware 36](#_Toc11)

[3.2 OpenStack 38](#_Toc12)

[3.3 OpenNebula 41](#_Toc13)

[4 ПРИНЦИПЫ ФУНКЦИОНИРОВАНИЯ OPENNEBULA 45](#_Toc14)

[4.1 Планировщик 46](#_Toc15)

[4.2 Сервис мониторинга 48](#_Toc16)

[4.3 OneFlow 49](#_Toc17)

[5 КОНФИГУРАЦИЯ СЕТЕЙ И СИСТЕМ ХРАНЕНИЯ ДАННЫХ В OPENNEBULA 51](#_Toc18)

[5.1 Конфигурация сетей 51](#_Toc19)

[5.2 Конфигурация хранилищ 52](#_Toc20)

[6 РАЗВЕРТЫВАНИЕ И ПРОВЕРКА ПОЛУЧЕННОГО РЕШЕНИЯ 59](#_Toc21)

[6.1 Развертывание решения 59](#_Toc22)

[6.2 Проверка полученного решения 63](#_Toc23)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 66](#_Toc24)

[ПРИЛОЖЕНИЕ А Исходный код 68](#_Toc25)

# **ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ**

В настоящей выпускной квалификационная работе магистра (магистерской диссертации) применяют следующие термины с соответствующими определениями:

|  |  |
| --- | --- |
| Балансировка ресурсов | метод равномерного распределения ресурсов физических хостов на все виртуальные машины для эффективности их работы. |
| Виртуализация | технология, которая позволяет формировать виртуальные копии компьютерных ресурсов, таких как аппаратура, операционные системы, средства хранения данных, сетевые интерфейсы и другие элементы. |
| Виртуальная инфраструктура | вычислительные ресурсы, такие как процессорные мощности, оперативная память, устройства хранения данных, сетевые интерфейсы и другие, которые распределяются автоматически между виртуальными машинами и другими виртуальными ресурсами кластера для проведения сложных облачных вычислений и поддержания распределения нагрузки на всей физической инфраструктуре кластера. |
| Виртуальная машина | программная эмуляция физического компьютера, включая его аппаратные компоненты, операционную систему и приложения |
| Высокая доступность | свойство, предполагающее набор специальных алгоритмов и проектировки системы, которые обеспечивают непрерывную работу и доступность значительной части ресурсов, сервисов и приложений, с минимальными простоями. |
| Гипервизор | программное обеспечение, которое можно использовать для запуска множества виртуальных машин на одной физической машине. |
| Кластер | это некоторое множество хостов (чаще всего физических серверов), которые объединены в единую систему с возможностью последующего расширения. Такая система предназначена для достижения общей цели, требующей значительного использования ресурсов и активной балансировки нагрузки. |

# ВВЕДЕНИЕ

История виртуализации берет свое начало с 1960-х годов, когда IBM разработала технологию виртуализации. Виртуализация была создана, чтобы максимизировать использование аппаратного обеспечения и упростить управление им, тем самым извлекая максимум пользы из имеющегося оборудования.

В 1970-х годах компания DEC выпустила первую коммерческую виртуальную машину VAX/VMS, которая позволяла запускать несколько операционных систем на одном физическом сервере. Концепция виртуализации продолжала развиваться вплоть до начала 2000-х годов, когда были созданы такие технологии, как VMware ESX Server и Xen.

Сегодня виртуализация широко используется в корпоративных ИТ-системах для оптимизации использования ресурсов, повышения гибкости и надежности системы и сильно представлена в облачных вычислениях, для которых она является центральной технологией. В настоящее время использование облачных технологий активно распространяется в различных сферах деятельности благодаря возможностям, предоставляемым виртуализацией.

Виртуализация позволяет создавать и запускать несколько виртуальных машин на одном физическом сервере, что способствует более эффективному использованию вычислительных ресурсов. Ранее для выполнения различных задач были выделены отдельные серверы, например, для почтового сервиса, базы данных и приложений. Однако с использованием виртуализации достаточно иметь один сервер с гипервизором, на котором будут развернуты все три виртуальные машины, каждая из которых будет выполнять предназначенную ей функцию. При этом каждая виртуальная машина будет находиться в изолированном окружении, будто бы находится на отдельном физическом сервере, что значительно упрощает процесс развертывания и поддержки системы, так как не требуется учитывать возможные конфликты между различными компонентами системы при работе с ними.

К тому же использование виртуальных машин позволяет повысить надёжность при хранении данных, ведь для неё можно использовать различные распределенные системы хранения данных, такие как Сeph, что позволяет делать репликацию информации в автоматическом режиме. Это обеспечивает возможность легкой миграции виртуальной машины на другой хост в случае, если с оригинальным хостом что-то произошло, а также настройку такой миграции в автоматическом режиме в случае аварии.

Также есть виртуализация с помощью контейнеров для приложений (таких как docker), которая обеспечивает меньшую изоляцию с системой, однако более легковесная чем виртуализация с полноценными виртуальными машинами, а также контейнеры гораздо быстрее разворачиваются и удаляются, чем виртуальные машины и у них есть специальные инструменты для оркестрации и автоматического развертывания. Тем не менее их функциональность более ограничена по сравнению с полноценной виртуальной машиной, в следствии чего их применение не всегда оправдано. Но есть и системные контейнеры (LXD), которые позволяют эмулировать полноценную виртуальную машину, за счёт того, что используется ядро операционной системы хоста.

Для того чтобы добиться лучшего контроля за созданием и удалением виртуальных машин, потребляемыми ими ресурсами, а также для мониторинга того, что они работают и их показателей были разработаны специальные инструменты – платформы виртуализации. Платформа виртуализации представляет собой инструментарий дающий полный контроль над использованием инфраструктуры организации для её виртуализации, позволяя выделять ресурсы как для группы пользователей, так и для какого-то конкретного пользователя, тем самым предоставляя возможность оптимизировать использование вычислительных ресурсов и балансировать нагрузку.

На текущий момент есть три популярные платформы виртуализации: VMware vSphere, OpenStack и OpenNebula. У каждой из них есть свои достоинства и недостатки, отличающих от других платформ, тем самым делая их уникальными, со своими областями применения. Например, VMware vSphere является стандартным выбором большинства организаций, которым нужна возможность создавать и удалять по требованию виртуальные машины для их сотрудников, из-за простоты работы, поддержки, а также большого количества специалистов разбирающихся в VMware. Однако из-за необходимости приобретать лицензию, а также из-за закрытости продуктов VMware, организация может отказаться от этого выбора в пользу более открытой платформы.

В контексте обеспечения безопасности платформ, можно отметить, что представленные решения не обладают оптимальным уровнем защиты. Для того чтобы его поднять можно использовать как встроенными средствами Linux, такими как SELinux, AppArmor, файрволл, так и внешними, например программами для контроля за секретами, такими как Hashicorp Vault и Keycloak, тем самым повысив безопасность инфраструктуры и снизив вероятность её взлома в будущем.

# 

# 1 ОСНОВНЫЕ ПРИНЦИПЫ РАБОТЫ ВИРТУАЛИЗАЦИИ И МЕХАНИЗМЫ ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ В LINUX

В современных вычислениях виртуализация и контейнеризация являются ключевыми технологиями, обеспечивающей экономию средств, эффективное использование ресурсов и гибкость. Технология виртуализации позволяет создавать уровень абстракции между программным обеспечением и физическим оборудованием, что позволяет нескольким виртуальным машинам работать на одной физической машине [1].

### 1.1 Технологии виртуализации

Виртуальные машины изолированы друг от друга, обеспечивая высокий уровень безопасности и отказоустойчивости [2]. Контейнеры в отличии от виртуальных машин обеспечивают меньший уровень изоляции, но за счёт скорости развёртывания гораздо лучше для развёртывания приложений, не требующих полноценной операционной системы. Однако также есть и системные контейнеры (LXD), позволяющие эмулировать полноценную виртуальную машину, за счёт использованием контейнером ядра операционной системы хоста [3].

Для виртуализации используются гипервизоры. Существуют гипервизоры первого и второго типа. Гипервизор ставящийся непосредственно на аппаратную часть называется гипервизором первого типа. Гипервизор, ставящийся поверх уже установленной операционной системы, называют гипервизором второго типа [4].

Гипервизор позволяет виртуальным машинам работать независимо друг от друга, обеспечивая их изоляцию. Также они легко создаются, изменяются и удаляются, что делает их гибкими и простыми в управлении.

Различают несколько типов виртуализации, каждый из которых предназначен для определенных аспектов вычислительных ресурсов. Ниже приведены основные типы виртуализации:

1. аппаратная виртуализация. Этот тип, который также называется виртуализацией платформы, предполагает создание виртуальных машин, которые эмулируют полную аппаратную систему, включая процессор, память, хранилище и сетевые устройства. Аппаратная виртуализация позволяет нескольким операционным системам работать одновременно на одном физическом сервере;
2. виртуализация операционной системы. Этот тип предполагает создание нескольких изолированных виртуальных сред, известных как контейнеры, в рамках одной операционной системы. Каждый контейнер использует ядро основной операционной системы, но запускает свои приложения и процессы изолированно. Этот подход обеспечивает более эффективное использование ресурсов и быстрое время запуска по сравнению с аппаратной виртуализацией;
3. виртуализация приложений. Этот тип относится к инкапсуляции приложения и его зависимостей в портативное автономное устройство, которое может работать на любой совместимой системе без необходимости установки;
4. виртуализация хранилища. Этот тип включает в себя объединение физических ресурсов хранения с нескольких устройств и представление их в хост-системе как единый логический блок;
5. виртуализация сети. Этот тип относится к абстрагированию физических сетевых ресурсов, таких как коммутаторы, маршрутизаторы и брандмауэры, в виртуальные ресурсы, которые можно выделять и управлять ими независимо.

### 1.2 Обеспечение безопасности в Linux

Одним из главных принципов безопасности в Linux является минимизация количества уязвимостей. Для этого используется механизм привилегий, который позволяет разграничить права доступа к ресурсам системы между пользователями и процессами.

У процессов в Linux может быть два вида привилегий, по которым их разделяют на: пользовательские (user) и системные (system) процессы. Пользовательские процессы дают доступ к ряду ресурсов, например, к файлам или сетевым портам, а также позволяют выполнять определенные действия, например, создавать процессы. Системные привилегии позволяют выполнять действия, которые могут иметь серьезные последствия для системы, например, изменять настройки ядра.

Также в Linux есть дополнительные механизмы защиты, позволяющие повысить безопасность Linux путем ограничения прав доступа к ресурсам системы, контроля доступа приложений к ресурсам, обнаружения и предотвращения атак, шифрования данных.

*SELinux*

SELinux (Security-Enhanced Linux) — это система безопасности для операционной системы Linux, разработанная Национальным институтом стандартов и технологий США в рамках проекта Flask. SELinux предоставляет многоуровневую защиту от угроз безопасности, обеспечивая контроль доступа к файлам, портам сети, процессам и другим ресурсам системы как показано на рисунке 1.

Одной из ключевых особенностей SELinux является использование политик безопасности для определения допустимых операций с ресурсами. Политика безопасности SELinux определяет права доступа для каждого процесса и объекта в системе. Это позволяет администраторам настроить свою систему максимально безопасно, ограничивая права доступа пользователей и процессов только необходимой минимальной функциональностью.

Рисунок 1 – Схематическое изображение работы SELinux



SELinux также использует концепцию мандатных меток для обеспечения безопасности. Каждый процесс, файл или другой ресурс системы имеет свою мандатную метку, которая определяет его уровень доступа и полномочия. Даже если злоумышленник получает доступ к системе, он не сможет выполнить недопустимые операции на ресурсах, так как уровень его доступа будет ограничен мандатной меткой.

SELinux является крайне гибкой системой безопасности. Она позволяет администраторам настроить свою систему с точностью до конкретного процесса или объекта, используя правила политики безопасности. Пользователи могут настраивать политические правила для своих приложений, чтобы гарантировать безопасность их работы.

Однако, настройка SELinux является нетривиальной задачей даже для опытных администраторов, при этом неправильная настройка политики безопасности может привести к блокировке доступа к ресурсам системы, а также перегрузке сервера, что будет негативно влиять на работу всей системы.

В целом, SELinux является мощной системой безопасности, которая обеспечивает высокий уровень обеспечения безопасности для операционных систем Linux. Однако, для её эффективного использования, необходимо настроить политики безопасности, на что потребуется время, а также знаний и опыт.

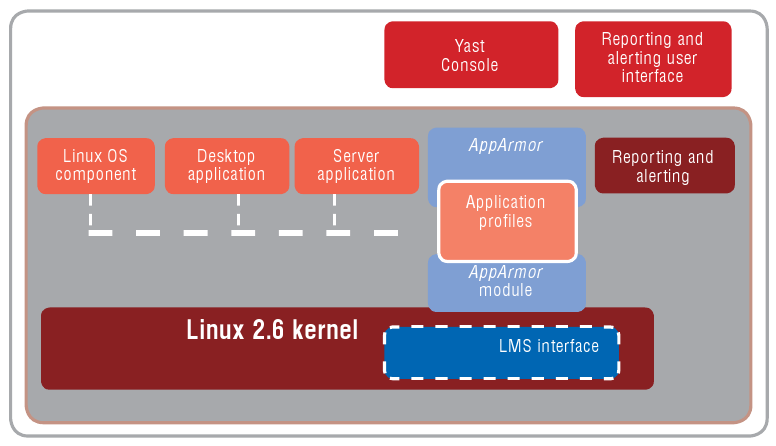
*AppArmor*

Для SELinux есть альтернатива в виде AppArmor. AppArmor также как и SELinux использует профили безопасности, чтобы определить, какие действия может выполнять каждое приложение в системе. Каждый профиль содержит набор правил, которые описывают, какие ресурсы могут быть доступны и какие операции разрешены на этих ресурсах. Приложения могут получить доступ только к тем ресурсам, которые разрешены в соответствующем профиле безопасности.

Профили безопасности могут быть созданы для конкретных приложений или групп приложений. Каждый профиль может быть связан с определенным пользователем или группой пользователей. Это позволяет администраторам определять, какие приложения могут запускаться от имени конкретного пользователя и какие действия они могут выполнять. Также система AppArmor поддерживает функцию аудита, которая записывает все попытки доступа к защищенным ресурсам. Это позволяет администраторам отслеживать активность в системе и обнаруживать возможные угрозы безопасности как показано на рисунке 2.

Благодаря этим функциям AppArmor можно использовать для защиты различных типов Linux-систем, включая серверы, рабочие станции и встроенные системы. Она также может быть интегрирована с другими системами безопасности, такими как межсетевой экран (firewall) или системой по обнаружению вторжений (intrusion detection system). А из-за своей простота в использовании и настройке, она становится особенно полезной для организаций, у которых нет специалистов по безопасности, за счёт того, что система может быть легко настроена для конкретных приложений и требований безопасности.

Рисунок 2 – Схематическое изображение работы AppArmor



В целом, AppArmor представляет собой мощную систему безопасности Linux, которая может быть использована для защиты различных типов систем и приложений. За счёт своей простоты использования и гибкости она является хорошим выбором для широкого круга пользователей, которым не нужны все функции SELinux.

*Файервол*

Firewall в Linux — это программное обеспечение, которое предназначено для фильтрации и трансляции сетевого трафика между различными сетями и устройствами. Оно работает на втором и третьем уровне сетевого протокола и может как блокировать, так и разрешать соединения в соответствии с используемыми правилами, с возможностью их гибкой настройки для контроля доступа к ресурсам сети. Эта гибкость позволяет настраивать правила для каждого приложения отдельно, что обеспечивает более точный контроль над доступом к ресурсам.

Это позволяет ограничить доступ к системе и защитить её от несанкционированного доступа, например, настроив правила для блокировки вредоносных запросов или защиты от DoS-атак. Также firewall может быть использован в сочетании с другими системами безопасности, что повышает эффективность защиты от внешних угроз. Например, он может быть интегрирован с системой по обнаружению вторжений для быстрого реагирования на новые виды атак. За счёт простоты настройки и использования, он легко вписывается в инфраструктуру организаций, тем самым обеспечивая значительный уровень безопасности в них.

Таким образом, использование firewall в Linux является необходимым элементом защиты сети от вредоносных программ и злоумышленников. Он предоставляет гибкие возможности контроля доступа, а также может быть легко интегрирован с другими системами безопасности для более эффективной защиты от угроз.

Для настройки файрволла в linux есть четыре распространённые утилиты:

1. ebtables – файервол 2-го уровня сетевого стека,
2. iptables – файервол 3-го уровня сетевого стека,
3. nftables – файервол 3-го уровня сетевого стека,
4. ufw - файервол 3-го уровня сетевого стека.

*ebtables*

Ebtables — это инструмент для фильтрации трафика в Linux, предназначенный для работы на уровне кадра Ethernet. В отличие от iptables, который работает на уровне сетевого стека, ebtables позволяет пользователю управлять трафиком на более низком уровне.

Одной из основных функций ebtables является возможность блокировки трафика на основе MAC-адресов. Это может быть использовано для предотвращения доступа к определенным устройствам в локальной сети или для блокировки широковещательного трафика, который может приводить к перегрузке сети. Схема настройки правил представлена на рисунке 3.

Кроме того, ebtables также позволяет пользователю создавать правила для маршрутизации трафика на основе MAC-адресов и VLAN-тегов. Это делает его полезным инструментом для управления трафиком в больших сетях.

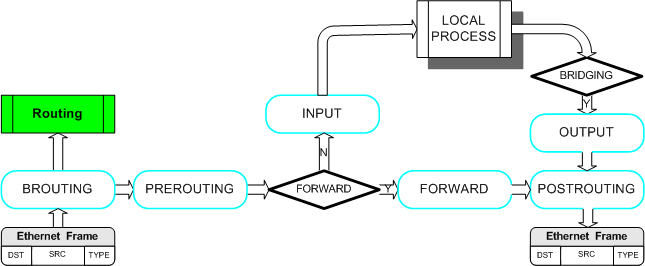


Рисунок 3 – Схематическое изображение работы ebtables

Ebtables также может быть использован совместно с другими инструментами для фильтрации трафика, такими как iptables и nftables. Это позволяет пользователям создавать сложные правила для управления трафиком на различных уровнях сетевого стека.

В целом, благодаря своей способности управлять трафиком на более низком уровне и возможности работать в сочетании с другими инструментами для фильтрации трафика, ebtables представляет собой мощный инструмент для управления трафиком в Linux. Он может быть особенно полезен для администраторов сетей, которые хотят более точно контролировать трафик в своих сетях.

*iptables*

iptables — это стандартный межсетевой экран в Linux, который используется для управления трафиком на уровне ядра. iptables работает в основном с пакетами IP, фильтруя их в зависимости от определенных правил. Это позволяет пользователям более детально контролировать, какие сетевые соединения разрешены или запрещены на их системах.

iptables использует цепочки правил для фильтрации входящего и исходящего трафика. Пользователи могут определить свои собственные правила для каждой цепочки, которые будут выполняться при определенных условиях. Например, они могут настроить правила для блокировки конкретных IP-адресов, портов или протоколов. Схема по которой определяется фильтрация пакета показана на рисунке 4.

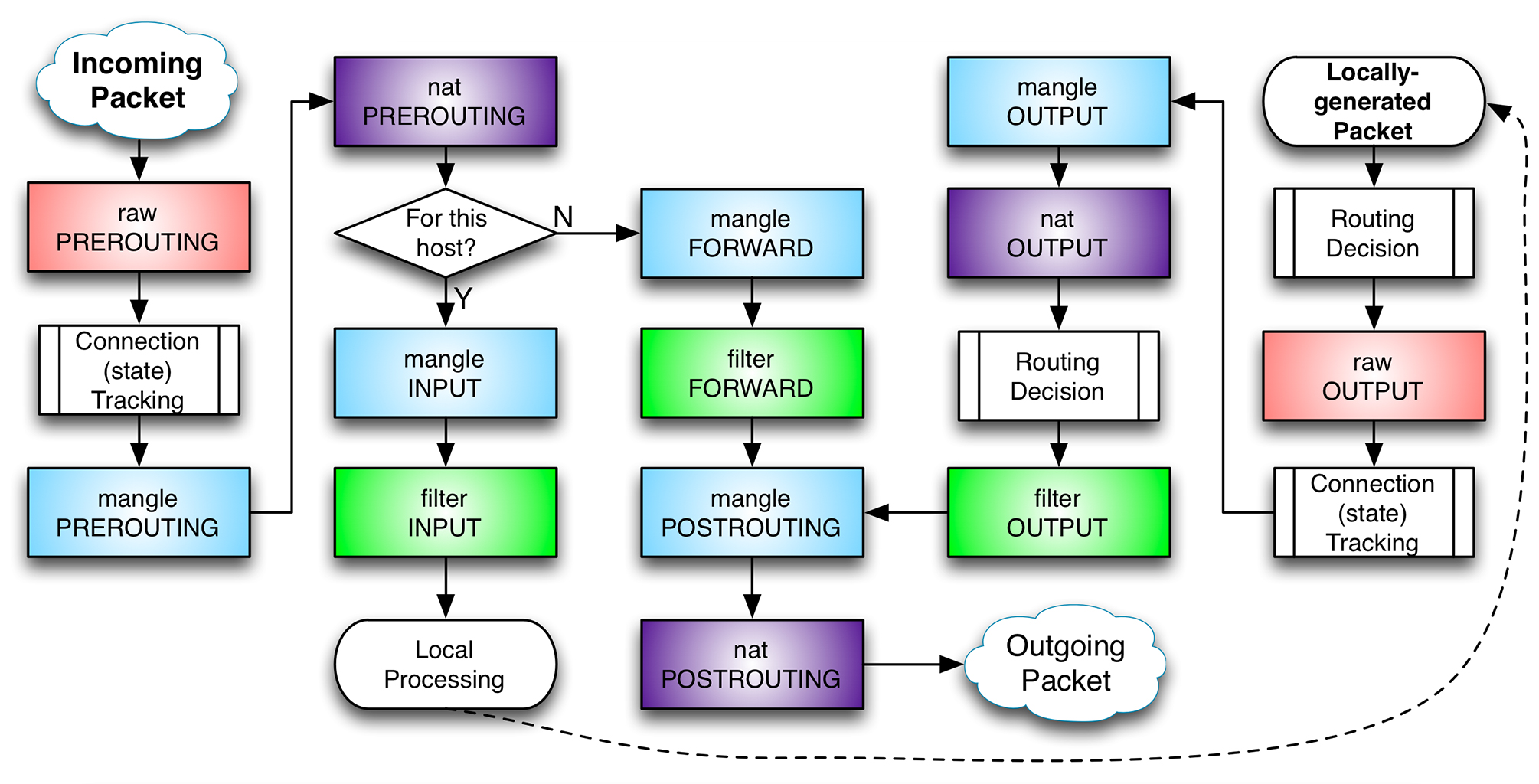


Рисунок 4 – Схематическое изображение работы iptables

Одним из ключевых преимуществ iptables является его гибкость и возможность настройки. Пользователи могут настроить правила в зависимости от своих потребностей без необходимости создания сложных списков. Кроме того, iptables может быть интегрирован с другими системами безопасности, такими как IDS (Intrusion Detection System), чтобы обеспечить еще большую защиту сетевого трафика.

По итогу, Iptables является мощный инструмент для управления сетевым трафиком на Linux-системах. Он позволяет пользователям определить свои собственные правила для фильтрации трафика, а также может быть интегрирован с другими системами безопасности для обеспечения еще большей защиты. Благодаря своей гибкости и простоте в использовании, iptables является особенно ценным инструментом для организаций, которые хотят обеспечить безопасность своих сетей.

*nftables*

Nftables - это современный инструмент для управления брандмауэром и фильтрации пакетов в Linux. Он был разработан для замены iptables, и предоставляет ряд преимуществ по сравнению с ним.

Одно из главных преимуществ nftables заключается в его гибкости и расширяемости. Настройка правил фильтрации осуществляется при помощи набора команд и выражений, которые могут быть легко изменены и дополнены. Это делает возможным создание сложных фильтрационных схем, которые могут быть адаптированы под конкретные потребности сетевой инфраструктуры.

Кроме того, nftables предоставляет множество опций для управления соединениями и обработки трафика в режиме реального времени. Его можно использовать для блокирования или разрешения конкретных типов трафика, а также для отслеживания соединений в режиме реального времени. Это позволяет операторам сетей легко настраивать правила безопасности для защиты от различных угроз, таких как атаки DDoS или вредоносные программы.

Таким образом можно выделить несколько ключевые отличия nftables от iptables. Во-первых, синтаксис nftables проще и более интуитивно понятен. В отличие от iptables, который использует множество команд, nftables основан на декларативном стиле, что делает его более простым в использовании. Во-вторых, nftables работает значительно быстрее, благодаря оптимизированному ядру. Это может быть особенно важным при обработке больших потоков данных. В-третьих, nftables лучше масштабируется по сравнению с iptables, особенно при больших таблицах правил. Он также имеет больше возможностей, таких как ограничение скорости, классификация трафика и target'ы статистики и accounting. Наконец, nftables основан на модульной архитектуре, что делает его более гибким и расширяемым.

Резюмируя, nftables является более современным, функциональным и удобным в использовании версией iptables, так что хотя iptables всё ещё широко используется, в будущем nftables его полностью заменит. Он обладает высокой эффективностью и множеством функций для обеспечения безопасности сетей на всех уровнях. Благодаря этим свойствам, он может быть использован как для защиты серверов, рабочих станций, так и для встроенных систем.

*ufw*

ufw, или Uncomplicated Firewall, является инструментом управления брандмауэром для ОС Linux, предоставляя возможность легко и гибко настраивать правила доступа к сети. ufw позволяет администраторам настроить различные политики безопасности, которые могут использоваться для защиты серверов, рабочих станций или встроенных систем.

Основным преимуществом ufw является его простота использования. Он имеет графический интерфейс, благодаря чему он может быть легко настроен даже теми, кто не является специалистами по безопасности, который показан на рисунке 5. Тем не менее, он также предоставляет широкий спектр функций, которые позволяют настраивать правила брандмауэра для различных типов соединений, включая TCP, UDP и ICMP.

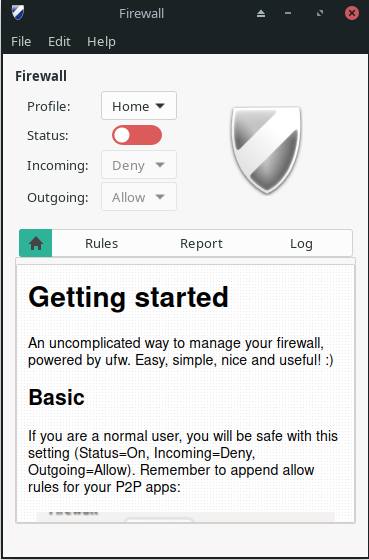


Рисунок 5 – Графический интерфейс ufw

Кроме того, ufw может быть легко интегрирован с другими системами безопасности, такими как межсетевой экран или система по обнаружению вторжений, что обеспечивает ещё большую защиту от потенциальных угроз. Это делает его особенно полезным для организаций, которым требуется надежная защита, но у которых нет специалистов по безопасности для настройки более сложных систем.

В целом, ufw представляет собой мощный инструмент управления брандмауэром, который обеспечивает высокий уровень безопасности и гибкость в настройке правил доступа к сети. Использование ufw может помочь организациям легко настроить необходимые политики безопасности и защитить свои системы от потенциальных угроз.

*Шифрование и пароли*

Шифрование и использование паролей в Linux являются критическими функциями для защиты системы от различных угроз, таких как взлом или несанкционированный доступ. Хорошо настроенная безопасность паролей и шифрования может защитить данные пользователей и конфиденциальную информацию от злоумышленников.

Linux предоставляет множество инструментов для защиты системы, включая использование шифрования для хранения данных и обмена ими. Эти инструменты включают в себя GPG (GNU Privacy Guard), OpenSSL и LUKS (Linux Unified Key Setup).

В дополнение к шифрованию, Linux также предоставляет возможности для использования паролей, которые могут защитить доступ к системе. Пароли могут быть использованы для аутентификации пользователя и контроля доступа к различным ресурсам на системе, таким как файлы и приложения.

Одним из популярных инструментов для управления паролями и шифрованием в Linux является PAM (Pluggable Authentication Modules). Он предоставляет гибкую систему аутентификации для различных приложений и сервисов, что позволяет использовать различные методы аутентификации и сохранять пароли в безопасном виде.

Кроме того, Linux также имеет возможность использовать различные алгоритмы шифрования, которые обеспечивают высокий уровень защиты данных. Эти алгоритмы включают AES (Advanced Encryption Standard), Blowfish и Twofish и многие другие.

Использование паролей и шифрования является критически важным для обеспечения безопасности системы Linux. При правильной настройке они могут гарантировать максимальную защиту конфиденциальной информации от несанкционированного доступа. Таким образом, обеспечение безопасности системы Linux может быть достигнуто путем использования правильных инструментов и методов, что сделает систему надежной и защищенной от угроз.

*Обновления и патчи*

Обновления и патчи в Linux являются неотъемлемой частью поддержки безопасности ОС. Они не только исправляют ошибки и уязвимости, но и обеспечивают свежие функции и повышенную производительность.

Один из наиболее важных аспектов обновлений и патчей в Linux — это защита от известных уязвимостей. После того, как новая уязвимость была обнаружена, разработчики выпускают соответствующий патч, который решает проблему. Без обновления системы, уязвимость остается открытой для злоумышленников, которые могут использовать ее для получения доступа к системе или выполнения других вредоносных действий.

Кроме того, обновления и патчи могут улучшить безопасность системы через введение новых функций. Например, SELinux - Security-Enhanced Linux - представляет собой расширение ядра Linux, которое добавляет дополнительные функции безопасности. SELinux может использоваться для ограничения прав доступа к файлам и процессам, что повышает защиту системы от потенциальных угроз.

Также, обновления и патчи могут обеспечить совместимость системы с другими безопасностями, такими как межсетевой экран или система по обнаружению вторжений. Например, некоторые патчи могут улучшить работу сетевых подключений, что может повысить эффективность межсетевого экрана.

Наконец, использование обновлений и патчей непосредственно связано с общей производительностью системы. Новые версии программного обеспечения, которые могут быть установлены через обновления, могут улучшить производительность приложений и снизить риск возникновения ошибок.

В целом, обновления и патчи в Linux являются необходимым компонентом безопасности ОС. Они помогают защитить систему от известных уязвимостей, добавляют дополнительные функции безопасности и улучшают производительность.

*Аудит безопасности*

Аудиты безопасности — это важный процесс, позволяющий оценить уровень безопасности системы, выявить потенциальные уязвимости и проблемы, которые могут быть использованы злоумышленниками для нарушения работы системы. Однако, безопасность не должна рассматриваться как статический процесс, а скорее, как постоянно изменяющуюся динамическую среду, где новые виды угроз появляются каждый день.

В связи с этим, необходимо постоянно улучшать безопасность системы, используя современные методы и технологии защиты. Например, можно использовать интеллектуальные системы, которые могут обнаруживать и предотвращать новые виды атак, основанные на машинном обучении и анализе больших данных. Такие системы могут улучшить эффективность защиты и снизить риски для организаций.

Кроме того, необходимо постоянно обновлять программное обеспечение системы и применять последние обновления безопасности. Это поможет устранить известные уязвимости и повысить уровень защиты системы. Также, следует контролировать доступ к системе и применять многофакторную аутентификацию, чтобы снизить риски несанкционированного доступа.

Наконец, важно обучать пользователей правильному использованию системы и осведомлять их о новых угрозах и методах защиты. Обучение пользователей может повысить уровень безопасности системы, поскольку они становятся более осведомленными и знают, как предотвратить потенциальные атаки.

Применение новых технологий и инструментов, обновление программного обеспечения и обучение пользователей могут помочь улучшить уровень безопасности и снизить риски для организаций.

# **2 ОСНОВНЫЕ ПРИНЦИПЫ ХРАНЕНИЯ И УПРАВЛЕНИЯ СЕКРЕТАМИ**

В настоящее время всё больше приложений способствуют хранению и переносу персональных данных пользователей, а через корпоративные системы проходит всё больше данных для исключительно внутреннего пользования. Причём в обоих случаях количество людей, пользующихся возможностями данных приложений и систем, только растёт. Из-за этого злоумышленники всё чаще пытаются находить уязвимости систем безопасности, для последующего проникновения и похищения информации. Одним из самых опасных способов проникновения считается получение данных для авторизации в системе (зачастую комбинации логин-пароль) через проникновение на личное менее защищённое устройство пользователя или перехват этих данных через слабо защищённый канал связи.

Ситуацию так же осложняет тот факт, что подобных систем становится всё больше, из-за чего людям становится сложнее ориентироваться в своих учётных записях, вследствие чего они пренебрегают правилами безопасности и создают похожие сочетания логинов-паролей или вовсе их повторяют. Некоторые приложения позволяют проводить аутентификацию и/или авторизацию на том или ином ресурсе без создания дополнительных пар логинов-паролей, что с одной стороны избавляет пользователей от нужды помнить или иметь записанными данные для авторизации, но с другой стороны получение злоумышленниками доступа к аккаунтам пользователя данных приложений делает его ещё более уязвимым. Это так же не всегда надёжно из-за того, что по результатам недавних исследований множество репозиториев Github не скрывает своих методов и ключей для хранения секретов. Однако взлом Electronic Arts 10 июня 2021 года показал, что частные корпоративные сети также не имеют идеальной защиты от злоумышленников, в первую очередь из-за халатного отношения к хранению секретов.

Из-за всех перечисленных обстоятельств вопрос хранения и управления секретами с каждым годом становится всё более актуальным.

### 2.1 Секрет и методы его защиты

Секрет - набор конфиденциальной информации, который необходимо надежно защитить. Такими данными могут быть пароли, базы данных с личной информацией клиентов, API токены, TLS сертификаты и прочее. При этом секреты, могут использоваться не только людьми, но и машинами.

Некоторое время назад секреты управлялись достаточно просто - администратор выдавал сочетание логина и пароля, и как клиент, так и администратор заносили его в свои базы данных. В современных реалиях этот метод применим только в случае наличия сотни пользователей для компании и наличии не более двух десятков различных учётных записей для пользователей. Впрочем, его модификация, где администраторы выдают пользователям некоторые сгенерированные данные для авторизации, широко применяется в современных корпоративных сетях. Данный метод хорошо себя показывает в случаях, когда по некоторым обстоятельствам пользователь потерял свои данные для авторизации, т.к. их можно легко и быстро восстановить. Весомым же недостатком такой системы является высокая уязвимость для атак социальной инженерии.

Одним из возможных решений данной проблемы является возложение задачи создания секрета на пользователя, что и делают многие компании в своих сервисах. Однако на уровне корпоративных сетей этот концепт не работает уже в теории, так как предполагается большое количество сотрудников, а многие люди склонны повторять как минимум паттерны создания своих секретов, как максимум полностью их копировать с других ресурсов, что сильно бьёт по безопасности сети. В случае, когда пользователю нужно авторизоваться в стороннем приложении через имеющееся, это просто бессмысленно. В добавок уже сейчас такой тип секретов как «приложение — приложение», использующихся исключительно машинами превышает количество пользовательских секретов и на их шифрование требуется всё больше ресурсов, которые они часто не получают из-за недооценки их важности. В таких условиях роста количества секретов контроль над их внутренним распространением в компании теряется и секреты начинают бесконтрольно применяться в рамках всей компании (так называемый secret sprawl). Восстановление контроля над управлением секретами в таких случаях требует больших денежных затрат и времени.

Данную проблему призваны решить различные средства автоматизации. Но, к сожалению, для большого количества программистов эти автоматизации остаются на уровне использования файлов с логинами и паролями и прямого встраивания секретов в код программ. Все данные проблемы призваны устранить системы управления секретами на подобии Hashicorp Vault и Keycloak.

В управлении секретами важна защищённость и возможность легкого контроля секретов на протяжении всего их жизненного цикла, которая достигается за счёт централизации процесса управления.

Система управления секретами — это технология для хранения, управления и предоставления секретов. Они позволяют безопасно предоставлять секреты не только бесчисленному количеству людей, но и во много раз большему количеству электронных устройств. Это осуществимо за счёт гибкости данной технологии - её можно применить для защиты различных типов секретов: секреты, которые требуются коду программы для доступа к различным ресурсам, благодаря чему секреты не приходится включать в сам код; секреты требующиеся для доступа к электронному устройству (серверу, смартфону, стационарному ПК); секреты, хранящиеся в базах данных и содержащие различные персональные данные и конфиденциальную информацию; секреты для доступа к облачным средам также будут защищены от компрометации. И на этом гибкость систем управления секретами не заканчивается - они могут быть встроены в уже созданную и набравшую большую пользовательскую базу систему приложений и при этом обеспечат этой системе единого SSO сервера с поддержкой множества идентификационных провайдеров и брокеров. Более того, подобные системы позволяют разделять секреты и хранить их отдельно друг от друга, что существенно снижает финансовые потери в случае утечки.

Системы управления секретами полезны не только с точки зрения удобного и безопасного хранения секретов в едином пространстве, быстрого поиска секретов по базе и получения их по требованию. С их помощью можно легко поддерживать каждый секрет в период его жизненного цикла. Создание секрета можно доверить как человеку, так и более надёжным алгоритмам компьютера и настроить политику их создания и использования. При использовании секрета можно настроить его автоматическую блокировку через определённый период времени для его регулярного изменения или задать период его автоматического изменения программой. Удалять секреты с помощью системы управления виду их дискредитации или ненужности так же просто как человеку, так и компьютеру [5].

Благодаря всему этому возможно быстро наладить централизованную политику управления секретами - ограничение минимальной длинны, обязательное использование заглавных букв, цифр и знаков препинания, и запретить неприемлемые пароли, просматривать логи, журналы и проводить регулярные мониторинги, и восстанавливать секреты без серьёзных денежных затрат.

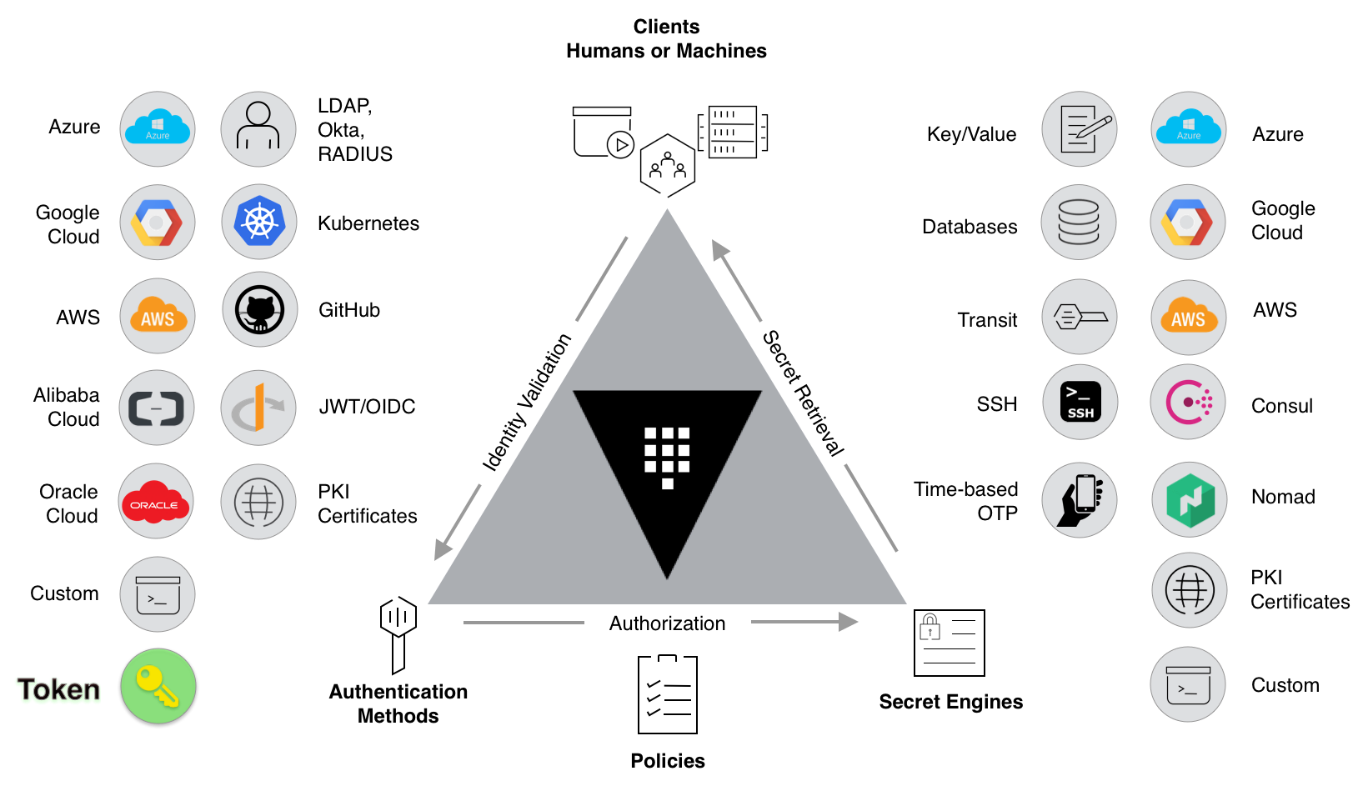
### 2.2 Hashicorp Vault

Hashicorp Vault — это программа с открытым исходным кодом, которая предоставляет надёжный и безопасный способ хранения секретов (API, токены доступа и пр.), и которую можно просто развернуть как у себя, так и в облаке [6]. Vault поддерживает командную строку и графическую оболочку, а также к ней можно делать запросы через HTTP. Vault централизованно хранит всю информации и поставляется с встроенными менеджерами секретов и методами аутентификации, которые можно добавлять и убирать как плагины, либо добавить свои собственные. Общая схема работы Hashicorp Vault представлена на рисунке 6.

Работа с Vault происходит по следующему сценарию:

1. клиент аутентифицируется с помощью одного из включённых способов аутентификации (через Github, aws, просто логин/пароль);
2. Происходит авторизация пользователя, с выдачей ему прав согласно настроенным политикам;
3. клиент получает доступ к нужному секрету (если у него есть к нему доступ, согласно выданным на 2 этапе разрешениям).

Рисунок 6 – Основные принципы работы Vault



Vault позволяет использовать как статические, так и динамические секреты. Динамические секреты не записаны в памяти, так что их нельзя украсть, и генерируются при доступе к ним, так что другой клиент получит другой секрет при генерации. По истечении срок действия, динамически созданный секрет будет автоматически отозван(удалён), однако Vault поддерживает возможность удаления секрет и до истечения срока действия, то есть можно отозвать секрет сразу после его использования. За счёт этого повышается безопасность, т.к. даже если секрет утечёт, он уже может быть отозван, а также на его использование накладывается временное ограничение, то есть вместо статичной цели, у нас получается динамическая цель (moving target).

Политики позволяют разграничить доступ к секретам и ресурсам как для индивидуальных пользователей, так и для группы. Например, можно создать политику, которая позволяет всем клиентам, аутентифицирующимся через Github, и состоящими в организации в определённой команде (на Github), получить доступ к динамической секретам определённой роли для сервисов aws, имеющая политику, которая позволяет пользователям такого динамического ключа иметь доступ к ограниченному кругу сервисов aws и действий на них.

При аутентификации, пользователю выдаётся токен, который и используется для доступа к Vault. Можно отозвать как этот индивидуальный токен, так и все токены полученные через определённый тип авторизации. При удалении любого типа авторизации, все токены полученные через него отзываются. Через политики, можно настроить TTL (Time-To-Live - время жизни) токенов, и т.к. политики гибко настраиваются для разных методов аутентификации, то соответственно можно настраивать сколько токен будет жить при том, или ином методе входа.

Ещё Vault предоставляет Encryption as a Service, таким образом клиенту не надо беспокоится об правильной реализации и интегрировании алгоритмов шифрования, т.к. всё происходит на стороне Vault с использованием Rest API, то есть клиент не имеет доступа к секретному ключу, что обеспечивает ещё большую сохранность секретов.

Vault может обеспечивать отказоустойчивость за счёт запуска несколько экземпляров (instances) Vault. Среди них выбирается один лидер, и все запросы к другим экземпляров перенаправляются на него. Если лидер умирает, то среди оставшихся instances автоматически выбирается новый лидер.

При инициализации Vault, создаётся заданное количество unseal ключей (Схема разделения секрета Шамира). Эти ключи нужны для того, чтобы “открыть” (unseal) Vault, при этом нужно использовать заданное заранее количество ключей (это число называется threshold). Например, есть 5 ключей, и из них 3 нужно для разблокировки. При этом все ключи равнозначны, то есть можно использовать любые 3 ключа, в любом порядке. В случае необходимости, любой пользователь с нужными правами может “запечатать” Vault обратно, в таком случае, получить любые данные из Vault будет невозможно до её разблокировки. В целях безопасности unseal ключи хранятся в разных местах, и, если один из владельцев ключей заметил подозрительную активность, он может использовать свой ключ и самостоятельно заблокировать Vault, и даже если у злоумышленника есть 1 или даже несколько unseal ключей, он не сможет разблокировать Vault, пока их меньше, чем threshold.

### 2.3 Keycloak

Keycloak - это инструмент хранения секретов с открытым исходным кодом. Имеет лицензию Apache и является основой для продуктов компаний, использующих SSO – RH-SSO.

Основной задачей данного инструмента является обеспечение защиты приложения и/или сервиса, обладающих слабым уровнем шифрования (в том числе и вовсе не обладающих им).

Благодаря простой интеграции Keycloak, уровень безопасности аутентификации приложения резко возрастёт, что позволит разработчикам освободить как трудовые, так и денежные ресурсы для других задач, в первую очередь для реализации различных функций приложения/сервиса.

Удобство использования Keycloak на прикладном уровне обусловлено совместимостью с контейнеризацией и возможностью изменения методов аутентификации без потребности в изменениях самого приложения. Это осуществляется за счёт использования различных адаптеров для различных языков программирования.

Повышение безопасности аутентификации осуществляется за счёт широкого спектра возможностей Keycloak:

1. через SSO, являющимся балансировщиком нагрузки, осуществляется централизованная аутентификация;
2. возможна аутентификация и синхронизация пользователей из различных (в том числе и внешних) идентификационных провайдеров (Active Directory, LDAP OpenID Connect и т.д.), без необходимости повторного ввода сочетания логин-пароль;
3. аутентификация может быть делегирована различным социальным сетям;
4. с помощью Google Authenticator или FreeOTP используется одноразовый код для аутентификации;
5. автоматическая аутентификация через Kerberos;
6. производится токенизация различных атрибутов пользователя.

Всё вышеперечисленное так же помогает упростить регистрацию пользователям, но, кроме этого, инструмент позволяет настроить создание учётных записей по заданной форме и проводить автоматические действия в случае утери пароля пользователем.

Так же Keycloak позволяет осуществлять удобную администрацию за счёт управления сессиями из единой точки, простую деактивацию аккаунтов, работу через командную строку, настройку интерфейса в стиле компании и в целом автоматизацию процессов управления для больших сетей. Так же инструмент не только имеет встроенную СУБД для хранения пользовательских данных, но и поддерживает большое количество существующих.

Для того чтобы позволить пользователям осуществлять аутентификацию в приложениях через Keycloak необходимо настроить серверную часть инструмента - создать на нём отдельные разделы с различными пользователями, их ролями и ключами. После этого пользователь делает запрос на аутентификацию и авторизацию. Внутри сервера происходит аутентификация и авторизация, после чего в зависимости от того, что попросил пользователь, возвращает либо код авторизации, либо токен доступа, вставляемый прямо в запрос авторизации как на рисунке 7.

В первом случае цепь запросов заканчивается. Во втором же случае после получения кода авторизации пользователь запрашивает у сервера токен и получает токен идентификации, содержащий сведения о пользователе, и уже упомянутый токен доступа, как показано на рисунке 8. Все токены подписаны RSA ключом и можно потребовать публичный ключ для проверки.

Рисунок 7 – Схема простой аутентификации

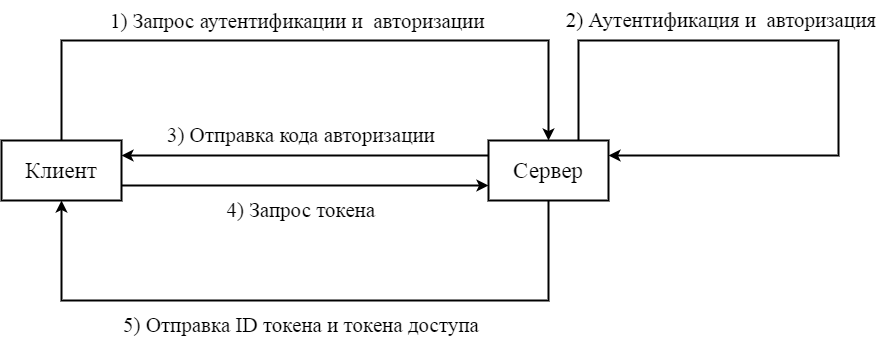
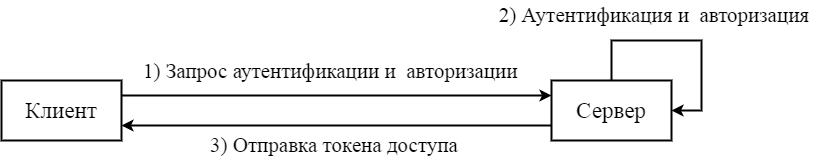


Рисунок 8 – Схема аутентификации с получением токена доступа и токена идентификации

# 3 ИМЕЮЩИЕСЯ ПЛАТФОРМЫ ВИРТУАЛИЗАЦИИ

Платформы виртуализации представляют собой программные решения для управления виртуальными машинами (VM) и ресурсами, такими как процессоры, оперативная память и хранилища данных. В данном тексте мы рассмотрим три наиболее популярные платформы виртуализации: VMWare, OpenNebula и OpenStack, каждая из которых имеет свои особенности и преимущества. Выбор конкретной платформы зависит от потребностей и возможностей организаций.

### 3.1 VMware

VMware является пионером в области виртуализации и облачных вычислений. Продукты VMWare ESXi и VMWare vSphere являются двумя флагманскими продуктами VMware.

VMware ESXi (ранее известный как VMware ESX) является гипервизором 1 типа, то есть он работает напрямую на аппаратном уровне без установки поверх операционной системы [7]. Одной из основных характеристик VMware ESXi является его удобство использования. Управление и настройка гипервизора может быть выполнено с помощью простого веб-интерфейса или через командную строку. Кроме того, VMware ESXi имеет широкий спектр функций, таких как миграция виртуальных машин, отказоустойчивость и динамическое выделение ресурсов.

VMware ESXi позволяет запускать несколько виртуальных машин на одном сервере, за счёт чего значительно повышает производительность и эффективность вычислительного оборудования, обеспечивая при этом высокий уровень безопасности, защищая данные и предотвращая несанкционированный доступ благодаря строгой изоляции между виртуальными машинами.

VMware vSphere — это платформа для виртуализации инфраструктуры, разработаная компанией VMware. Она даёт возможность создания и управления виртуальными машинами на сервере, а также создания кластера из несколько физических серверов для их объединения в единую инфраструктуру [8].

История развития VMware vSphere началась в 2002 году, когда VMware выпустила свой первый продукт - VMware Virtual Platform. В 2003 году вышла новая версия этого продукта, который уже назывался VMware ESX Server. С тех пор VMware ESX Server постоянно развивался и усовершенствовался. В 2006 году VMware выпустила первую версию платформы VMware Infrastructure, которая объединила в себе несколько продуктов, включая VMware ESX Server, VMware Virtual Center и VMware Converter.

В 2009 году VMware представила новую версию платформы - VMware vSphere 4.0, которая стала основой для всех последующих версий. Она включала в себя ряд новых функций и возможностей, таких как Fault Tolerance, которая обеспечивает непрерывность работы ВМ, даже если один из серверов выходит из строя.

В последующие годы VMware продолжала развивать и улучшать платформу vSphere, выпуская новые версии с улучшенными функциями и возможностями. В 2015 году была представлена версия VMware vSphere 6.0, которая стала первой версией, работающей с контейнерами. В 2018 году компания выпустила версию VMware vSphere 6.7, которая включает в себя ряд новых функций, таких как защита от угроз безопасности и улучшенные возможности управления. На текущий момент VMware vSphere получила широкую популярность благодаря своей возможности организации вычислительного центра, обеспечивающего надежность и масштабируемость, оставаясь при этом удобной в работе и поддержке.

Однако, из-за того, что вся продукция VMware является проприетарной и не имеет открытого кода, самостоятельное развитие и расширение её продукции невозможно, также за использование продукции VMware нужно платить, соответственно требуются дополнительные ресурсы на покупку лицензии. В связи с этим VMWare является плохим выбором, поэтому следует обратить внимание на продукты с открытым исходным кодом, который можно самостоятельно модифицировать.

### 3.2 OpenStack

OpenStack — это открытая платформа для создания облачной инфраструктуры, которая объединяет в себе различные технологии и сервисы, необходимые для создания, управления и масштабирования облачных приложений. Система OpenStack была создана в 2010 году как совместный проект компаний Rackspace и NASA [9], и на данный момент она является одной из самых популярных платформ для облачных вычислений.

Основным принципом работы OpenStack является использование модульной архитектуры, которая позволяет комбинировать различные компоненты системы в соответствии с конкретными потребностями пользователя. В рамках платформы OpenStack предоставляются такие базовые сервисы, как управление виртуальными машинами, хранение данных, сетевые сервисы и авторизация, и аутентификация пользователей. Кроме того, в состав платформы OpenStack входят дополнительные сервисы, такие как мониторинг, логирование, балансировка нагрузки и многое другое. Благодаря использованию OpenStack открытых стандартов и протоколов, таких как REST API, обеспечивается максимальная совместимость и интеграция с другими системами, расширяя возможности пользователей и позволяя легко интегрировать платформу OpenStack со своими приложениями и сервисами.

Другим преимуществом платформы OpenStack является ее гибкость и расширяемость. Пользователи могут выбирать только те компоненты системы, которые им необходимы, и настраивать их в соответствии с конкретными требованиями. Кроме того, благодаря активному сообществу разработчиков, платформа OpenStack получает постоянные обновления и улучшения, что позволяет ей оставаться актуальной и конкурентоспособной.

*Compute (Nova)*

OpenStack Compute, также известный как Nova, является одним из основных компонентов платформы OpenStack. Этот сервис предоставляет виртуальные машины (VMs) и другие ресурсы для обработки вычислений в облачных средах. Он предоставляет пользователю возможность создавать и управлять VMs, администраторам - возможность управлять физическими серверами и хранением.

Основная функциональность Nova заключается в создании и управление виртуальными машинами, а также масштабировании, обработки запросов на создание и удаление экземпляров виртуальных машин [10]. Nova состоит из нескольких модулей, каждый из которых выполняет свою конкретную задачу. Каждый модуль может быть запущен на отдельном сервере или на нескольких серверах для обеспечения отказоустойчивости. Nova написан на языке Python и использует множество внешних библиотек Python, таких как Eventlet (конкурентная сетевая библиотека), Kombu (фреймворк AMQP-сообщений) и SQLAlchemy (SQL-инструментарий и объектно-реляционный отображатель). Nova разработана для горизонтального масштабирования, то есть для расширения, используется больше серверов, а не более мощный сервер, и на этих серверах устанавливаются идентично сконфигурированные сервисы.

В связи с широкой интеграцией в инфраструктуры предприятий мониторинг производительности OpenStack в целом и производительности Nova в частности, масштабирование стало все более важной проблемой. Для мониторинга производительности от начала до конца необходимо отслеживать метрики от Nova, Keystone, Neutron, Cinder, Swift и других сервисов, а также мониторить RabbitMQ, который используется сервисами OpenStack для передачи сообщений. Все эти сервисы генерируют свои собственные файлы журналов, которые, особенно в инфраструктурах предприятий, также должны быть контролируемы.

*Image (Glance)*

OpenStack Image(Glance) позволяет создавать, хранить и управлять образами виртуальных машин [11]. Он поддерживает различные форматы образов. С помощью Glance пользователи могут загружать, сохранять и распространять свои образы виртуальных машин в облаке.

*Networking (Neutron)*

OpenStack Networking (Neutron) - это модуль для управления сетевыми ресурсами в платформе OpenStack. Он предоставляет программный интерфейс для создания и управления виртуальными сетями, подсетями, маршрутизацией и другими сетевыми функциями [12].

Основной целью Neutron является обеспечение гибкости и масштабируемости виртуальных сетей. На уровне архитектуры Neutron состоит из нескольких сервисов, каждый из которых выполняет определенную функцию. К примеру, сервис L3 Agent отвечает за управление маршрутизацией между виртуальными сетями и внешними сетями, а сервис DHCP Agent – за автоматическую настройку IP-адресов. Neutron использует концепцию плагинов, которые могут быть использованы для интеграции с различными типами сетевых устройств и технологий. Это позволяет расширять возможности Neutron и интегрировать его с существующей инфраструктурой.

Важной частью Neutron является механизм управления политиками безопасности. Он основан на использовании Security Group, который позволяет определять правила доступа к виртуальным машинам на основе портов (TCP/UDP) и IP-адресов. Кроме того, Neutron поддерживает различные типы туннелирования, такие как VXLAN, GRE и Geneve, что обеспечивает возможность создания виртуальных сетей в различных физических сегментах сети.

В целом, можно сказать, что OpenStack — это мощная и гибкая платформа для создания облачной инфраструктуры, которая позволяет пользователям создавать, управлять и масштабировать облачные приложения и сервисы. Система OpenStack использует открытые стандарты и протоколы, обладает модульной архитектурой и постоянно обновляется, и улучшается благодаря активному сообществу разработчиков. Однако из-за своей громоздкости (на текущей момент в проекте больше миллиона строк кода) OpenStack довольно сложен как для модификаций, так и для поддержки, для которой требуется отдельная команда специалистов. В связи с этим стоит рассмотреть следующего кандидата - OpenNebula.

### 3.3 OpenNebula

OpenNebula — это свободный программный инструмент имеющий открытый исходный код, который используется для создания и управления частными, общественными и гибридными облачными вычислительными системами. Он был разработан в Испании в 2005 году как проект исследовательской группы OpenNebula Project при Университете Компьютерных наук Мадрида [13].

В начале своей истории OpenNebula был основан как простой менеджер виртуальных машин, который позволял пользователям создавать виртуальные машины и динамически управлять ими. Но со временем он быстро развивался и стал полноценной платформой облачных вычислений. Одной из ключевых черт OpenNebula является его открытость и гибкость. Он предлагает модульную архитектуру, которая позволяет разработчикам расширять и дополнять систему новыми функциями, не изменяя ее основного кода.

OpenNebula также отличается тем, что поддерживает несколько гипервизоров, включая KVM, VMware и Xen, что позволяет пользователю выбрать наиболее подходящий для своих нужд гипервизор, а также Firecracker который позволяет пользователям создавать mini-vm, которые сочетают в себе легковесность контейнеров с изоляцией и безопасностью обеспечивающуюся виртуальными машинами.

На сегодняшний день OpenNebula является одним из наиболее популярных и надёжных инструментов для создания и управления облачными вычислительными средами, что подтверждается его использованием в таких организациях, как NASA, Unity, Иви, QT и более сотни других компаний по всему миру [14]. У неё есть множества преимуществ:

1. открытость. У OpenNebula открытый исходный код, за счёт чего каждый желающий может реализовать собственные модификации;
2. управление ресурсами. OpenNebula имеет встроенный модуль для мониторинга виртуальных машин, а также планировщик OneFlow, позволяющий оптимизировать использование ресурсов за счёт распределения на разные узлы;
3. отказоустойчивость. OneFlow позволяет отслеживать состояния хостов, и в случае их выхода из строя переносить виртуальные машины на другой хост;
4. гибкость виртуализации. Поддержка широкого спектра гипервизоров, включая KVM, VMware и Xen, а также возможность создания mini-vm с помощью Firecraker делает платформу гибкой, позволяя удолетворить самые разные потребности пользователей и организаций при решении их задач;
5. совместимость. OpenNebula можно легко интегрировать с другими инструментами и платформами управления виртуализацией, такими как Kubernetes, Terraform, Ansible. Это возможно из-за поддержки OpenNebula стандартных API и протоколов;
6. масштабируемость. OpenNebula может легко масштабироваться для крупных развертываний, поддерживая тысячи физических и виртуальных ресурсов. Эта масштабируемость позволяет расширять инфраструктуру по мере необходимости, не беспокоясь об ограничениях, налагаемых пTwofish;
7. легкость поддержки. OpenNebula простота в использовании и поддержке, за счёт использования интуитивно понятного веб интерфейса, хорошей документации и активному сообществу, которому можно задать дополнительные вопросы. Благодаря этому, для поддержки Open Nebula достаточно одного человека;
8. хранилища данных. OpenNebula имеет широкий перечень поддерживаемых типов хранилищ и файловых систем: NAS, SAN, LUNs, NFS, GlusterFS, Ceph, LVM, iSCSI, DRBD, LINSTOR [15]. Благодаря этому организация может выбрать систему хранения данных подходящую конкретно под свои нужды, с последующим её подключением к OpenNebula;
9. двухфакторная аутентификация. Поддержка OpenNebula двухфакторной аутентификации позволяет повысить безопасность при доступе к ресурсам инфраструктуры.

Таким образом из всех рассмотренных платформ виртуализации OpenNebula лучше всего подходит под поставленные задачи за счёт своей гибкости, надёжности, безопасности, удобства и открытости к модификациям, с возможностью добавления функциональности к платформе как за счёт использования расширенного XML API для интеграции с существующими инструментами и процессами, так и за счёт модификации исходного кода, с последующим развёртывания полученного решения.

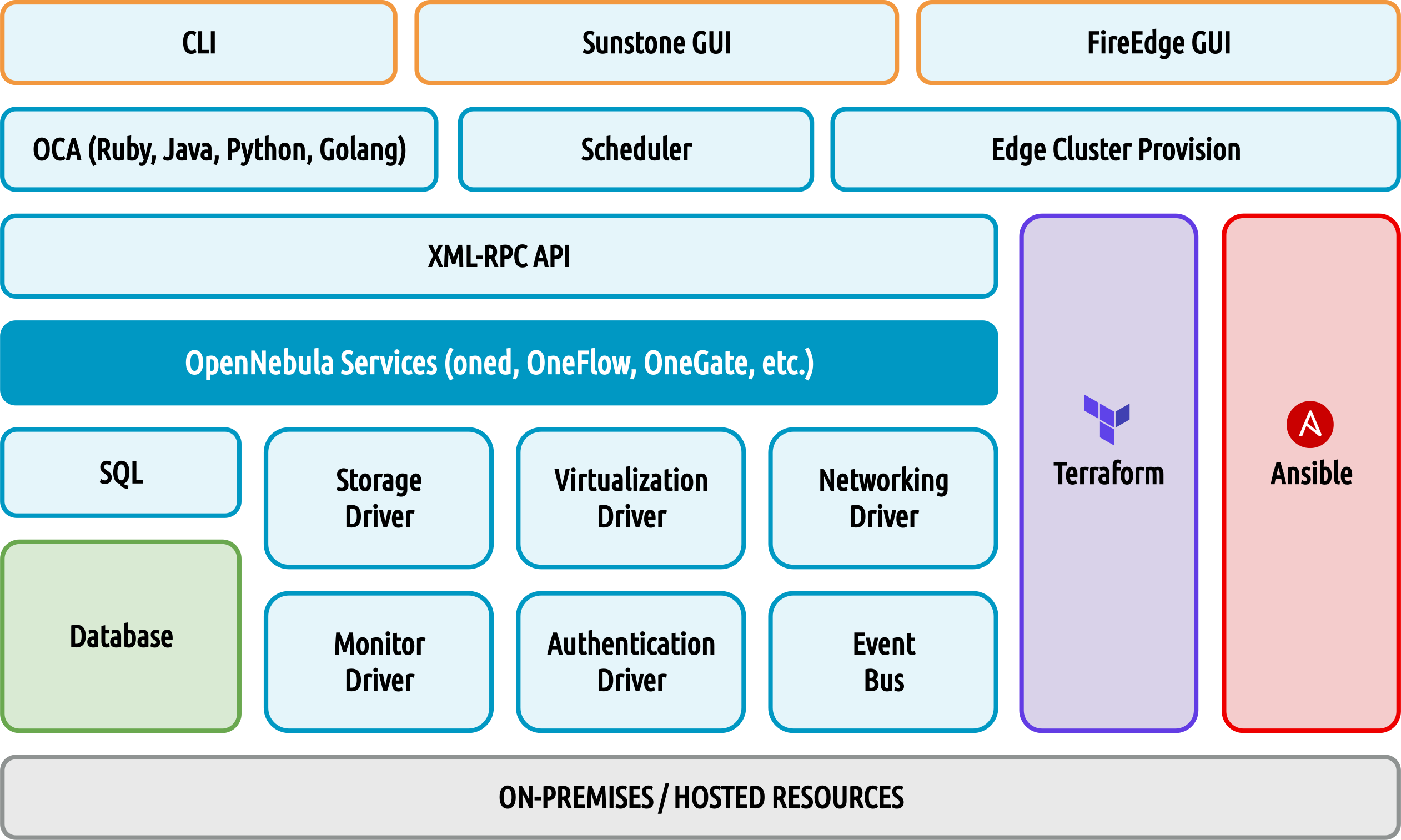


Рисунок 9 – Схема с основными компонентами OpenNebula

Тем не менее, для правильной реализации всего необходимого функционала, сначала нужно подробнее разобраться во внутреннем устройстве OpenNebula, что поможет достичь наилучших результатов при работе с ней в будущем.

# 4 ПРИНЦИПЫ ФУНКЦИОНИРОВАНИЯ OPENNEBULA

OpenNebula имеет модульную архитектуру и состоит из нескольких основных компонентов:

1. демон OpenNebula (oned). Основной сервис платформы OpenNebula для управления узлами кластера, виртуальными сетями и хранилищами, группами, пользователями, их виртуальными машинами. Этот демон предоставляет API XML-RPC для других сервисов и конечных пользователей;
2. база данных. OpenNebula сохраняет состояние облака в SQL-базу данных. Это ключевой компонент, который нужно мониторить и настраивать для достижения наилучшей производительности администраторами облака, следуя лучшим практикам выбранной базы данных;
3. планировщик (Scheduler). Отвечает за планирование размещения ожидающих виртуальных машин на доступных узлах гипервизоров. Это отдельный демон, устанавливаемый вместе с демоном OpenNebula, но может быть развернут независимо от него на другой машине;
4. сервис мониторинга. Подсистема мониторинга представлена выделенным демоном, который работает как часть демона OpenNebula. Он собирает информацию об состоянии узлов и виртуальных машин, например, статус хоста, основные показатели производительности, статус виртуальной машины и потребление ресурсов;
5. OneFlow. OneFlow оркестрирует многокомпонентные многовиртуальные сервисы, определяет зависимости и политики автомасштабирования для компонентов приложения, взаимодействует с демоном OpenNebula для управления виртуальными машинами (запуск, остановка) и может контролироваться через Sunstone GUI или по интерфейсу командной строки. Это отдельный демон, который устанавливается по умолчанию при установке, но может быть развернут независимо на другой машине;
6. OneGate. OneGate позволяет виртуальным машинам получать и отправлять информацию в OpenNebula, чтобы пользователи и администраторы могли использовать ее для сбора метрик, обнаружения проблем в своих приложениях и запуска правил OneFlow изнутри виртуальных машин. Он может использоваться со всеми поддерживаемыми гипервизоров (KVM, LXC, Firecracker и vCenter), если гостевая операционная система заранее установила специальный пакет для контекстной настройки OpenNebula. Это отдельный демон, который устанавливается по умолчанию при установке, но может быть развернут независимо на другой машине.

Остановимся на каждом сервисе отдельно, чтобы подробнее рассмотреть их особенности и функциональность.

### 4.1 Планировщик

Планировщик OpenNebula отвечает за планирование ожидающих виртуальных машин на доступных гипервизорах. Планировщик распространяется как пакет операционной системы opennebula с системным сервисом opennebula-scheduler.

OpenNebula поставляется с планировщиком подбора пар (match-making scheduler), который реализует политику ранжирования. Цель этой политики - приоретизировать наиболее подходящие ресурсы для виртуальной машины.

Алгоритм подбора пар работает следующим образом: каждый диск работающей ВМ использует хранилище из хранилища данных образа. Виртуальные машины, которым требуется больше места для хранения, чем доступно в настоящее время, отфильтровываются и остаются в состоянии ожидания.

Хосты, которые не соответствуют требованиям к ВМ (SCHED\_RANK) или не имеют достаточных ресурсов (доступного ЦП и памяти) для запуска ВМ, отфильтровываются.

То же самое происходит и с системными хранилищами данных: отфильтровываются те, которые не соответствуют требованиям DS (SCHED\_DS\_RANK) или не имеют достаточного количества свободного места.

Наконец, если виртуальная машина использует автоматический выбор сети, отфильтровываются виртуальные сети, которые не соответствуют требованиям к сетевым адаптерам.

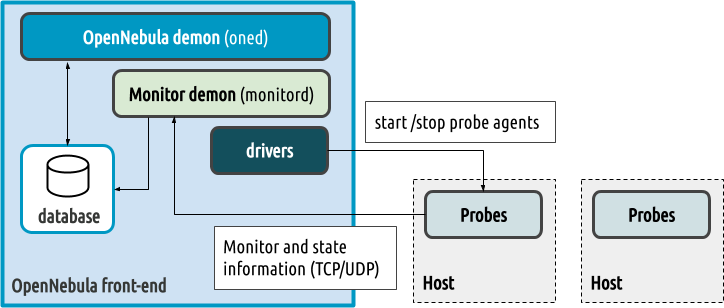
Выражения SCHED\_RANK и SCHED\_DS\_RANK оцениваются по списку узлов и хранилищ данных с использованием информации, собранной драйверами монитора. Кроме того, выражения NIC/SCHED\_RANK оцениваются по списку сетей с использованием информации из шаблона виртуальной сети. Любая переменная, сообщаемая драйвером монитора (или заданная вручную в шаблоне хоста, хранилища данных или сети), может быть включена в выражения ранжирования. Ресурсы с более высоким рангом используются в первую очередь для размещения виртуальных машин. Алгоритм планирования позволяет легко реализовать несколько эвристик размещения в зависимости от выбранных выражений RANK. Политику, используемую для размещения ВМ, можно настроить в двух местах: глобально для всех ВМ в конфигурационном файле /etc/one/sched.conf или для каждой виртуальной машины, как определено общими параметрами SCHED\_RANK и SCHED\_DS\_RANK, а также SCHED\_RANK, специфичным для сетевой карты, в шаблоне виртуальной машины.

Также пользователи могут запланировать выполнение одного или нескольких действий виртуальной машины на определенную дату и время. Команда расписания onevm добавит новый атрибут SCHED\_ACTION в редактируемый шаблон виртуальной машины.

### 4.2 Сервис мониторинга

Подсистема мониторинга представлена отдельным демоном (onemonitord), работающим как часть демона OpenNebula (oned), который собирает информацию, относящуюся к хостам и виртуальным машинам, такие как состояние хоста, основные индикаторы производительности, состояние виртуальной машины и занимаемое место. Эта информация собирается путем выполнения набора специальных программ(зондов), предоставляемых OpenNebula. Результат выполнения этих программ отправляется в OpenNebula с помощью механизма push. Это часть основного пакета для работы системы opennebula. Схема работы демона монитора представлена на рисунке 10.

Рисунок 10 – Схема работы демона монитора



Каждый хост периодически отправляет данные мониторинга по сети на сервер, который их собирает и обрабатывает в специальном модуле. Эта распределенная система мониторинга напоминает архитектуру выделенных систем мониторинга, использующих упрощенный протокол связи и модель push-уведомлений. Как часть обычного процесса запуска, OpenNebula запускает демон onemonitord, работающий во внешнем интерфейсе, который прослушивает сетевые подключения на порту 4124 (как UDP, так и TCP). Первоначально OpenNebula подключается к хостам с помощью SSH и запускает легковесный агент, который выполняет тестовые сценарии для сбора и отправки данных демону onemonitord во внешнем интерфейсе. Зонды структурированы по информационным категориям для информации о хосте и виртуальной машине. Через равные промежутки времени (в секундах) (настраиваемые для каждой категории в файле monitord.conf) данные передаются, поэтому подсистеме мониторинга не нужно устанавливать дополнительные соединения для их сбора.

Также этот сервис предоставляет возможность для шифрования этих сообщений для повышения безопасности, а также настройку для работы в кластерах высокой доступности и возможность самостоятельного задания интервалов мониторинга.

### 4.3 OneFlo**w**

OneFlow позволяет пользователям и администраторам определять, выполнять и управлять многоуровневыми приложениями, которые мы называем Службами, состоящими из взаимосвязанных виртуальных машин с зависимостями развертывания между ними. Каждая группа виртуальных машин развертывается и управляется как единое целое и полностью интегрирована с расширенным управлением пользователями и группами OpenNebula.

На следующей диаграмме представлено многоуровневое приложение. Каждый узел представляет роль и ее кардинальность (количество виртуальных машин, которые будут развернуты). Стрелки указывают зависимости развертывания: виртуальные машины каждой роли развертываются только тогда, когда работают все родительские виртуальные машины.

Также OneFlow позволяет администраторам и пользователям OpenNebula регистрировать шаблоны сервисов в OpenNebula, для того чтобы позже из них можно было создавать собственные экземпляры сервисов. Эти шаблоны можно переиспользовать, а также можно делиться ими с другими пользователями. Пользователи могут управлять шаблонами служб с помощью команды oneflow-template или Sunstone.

# 5 КОНФИГУРАЦИЯ СЕТЕЙ И СИСТЕМ ХРАНЕНИЯ ДАННЫХ В OPENNEBULA

### 5.1 Конфигурация сетей

Важной частью по обеспечению виртуализации инфраструктуры является конфигурация сетей и систем хранения данных. Хорошая конфигурация позволяет оптимально использовать ресурсы инфраструктуры, давая доступ к сети виртуальным машинам и обеспечивая репликацию данных с дальнейшей возможностью использования возможностей OpenNebula по обеспечению высокой доступности.

Виртуализация сетей — это технология, которая позволяет создавать виртуальные сетевые адаптеры, виртуальные сети и виртуальные маршрутизаторы на физическом сервере, что позволяет использовать несколько виртуальных машин на одном физическом сервере.

Существует несколько типов виртуальных сетевых адаптеров, таких как e1000 и virtio. Адаптер e1000 является эмуляцией сетевого интерфейса Intel PRO/1000, который поддерживает протокол Ethernet. Адаптер virtio — это драйвер, разработанный для работы в виртуализированной среде, и используется для обеспечения более высокой эффективности передачи данных.

VLAN (Virtual Local Area Network) позволяет разделить одну физическую сеть на несколько логических виртуальных сетей. При этом каждая VLAN имеет свой уникальный номер, что позволяет изолировать трафик между подсетями.

Виртуализация сетей может быть полезна в случаях, когда требуется запустить несколько виртуальных машин на одном физическом сервере, но при этом необходимо изолировать трафик между этими машинами или разрешить общение только по определенным протоколам. Кроме того, виртуализация сетей может значительно упростить настройку и управление сетью.

### 5.2 Конфигурация хранилищ

Виртуализация хранилища — это технология, которая позволяет создать виртуальное хранилище на основе физического устройства. Она используется для обеспечения более гибкой и эффективной работы с данными в виртуальных машинах позволяя пользователям иметь множество собственных виртуальных дисков.

Для работы с виртуальным хранилищем в гипервизорах используются различные форматы файлов, такие как raw и qcow2. Формат RAW представляет собой простой бинарный файл, содержащий данные в необработанном виде. Этот формат хорошо подходит для простого использования с виртуальными машинами, хоть и имеет ограниченные возможности по управлению и защите данных. Благодаря своей простоте может обеспечивать хорошую производительность, также можно легко примонтировать данный диск на хост системе при необходимости, например для просмотра логов или загрузки файла непосредственно на диск.

Формат QCOW2 является более продвинутым форматом файлов виртуального диска, который поддерживает функции сжатия, шифрования и кэширования данных. Этот формат также позволяет создавать "тонкие" диски, которые не занимают выделенное место на жестком диске хост-системы до тех пор, пока в них не будут записаны данные, таким образом даже позволяя выделять дискам больше места чем есть на самом деле. Минусом такого подхода является сниженная производительность из-за накладных расходов необходимых для работы дисков.

Для организации систем хранения данных обычно используются следующие технологии:

1. диск;
2. DRBD;
3. NFS;
4. SCSI/ISCSI;
5. CEPH.

*Диск*

Используется диск, имеющийся на сервере. Плюсами такого подхода является простота и отсутствие дополнительных накладных расходов для функционирования и работы. Минусами является отсутствии репликации и соответственно потеря надёжности при хранении данных, отсутствие возможности обеспечения высокой доступности для виртуальной машины, необходимость вручную копировать диск на другой хост при миграции виртуальной машины.

*DRBTwofishD*

DRBD (Distributed Replicated Block Device) — это технология, которая позволяет создавать реплицированные блочные устройства для хранения данных на нескольких серверах [16]. То есть несколько дисков на разных хостах работают как один общий диск для всех этих хостов, как показано на рисунке 11.

Плюсами является обеспечение отказоустойчивость и высокой доступности данных в распределенной среде, путем автоматической синхронизации данных между узлами, простота настройки и работы благодаря возможности использования утилиты высокого уровня drbdadm.

Минусами являются дополнительные расходы на диски, необходимость синхронизации для работы дисков, настройка правильного доступа к дискам, настройка правил для правильной работы для ситуации со split-brain, то есть, когда несколько хостов считают себя главными(primary) и могут записать данные в один и тот же сектор диск, тем самым приводя к появлению артефактов и порче данных.

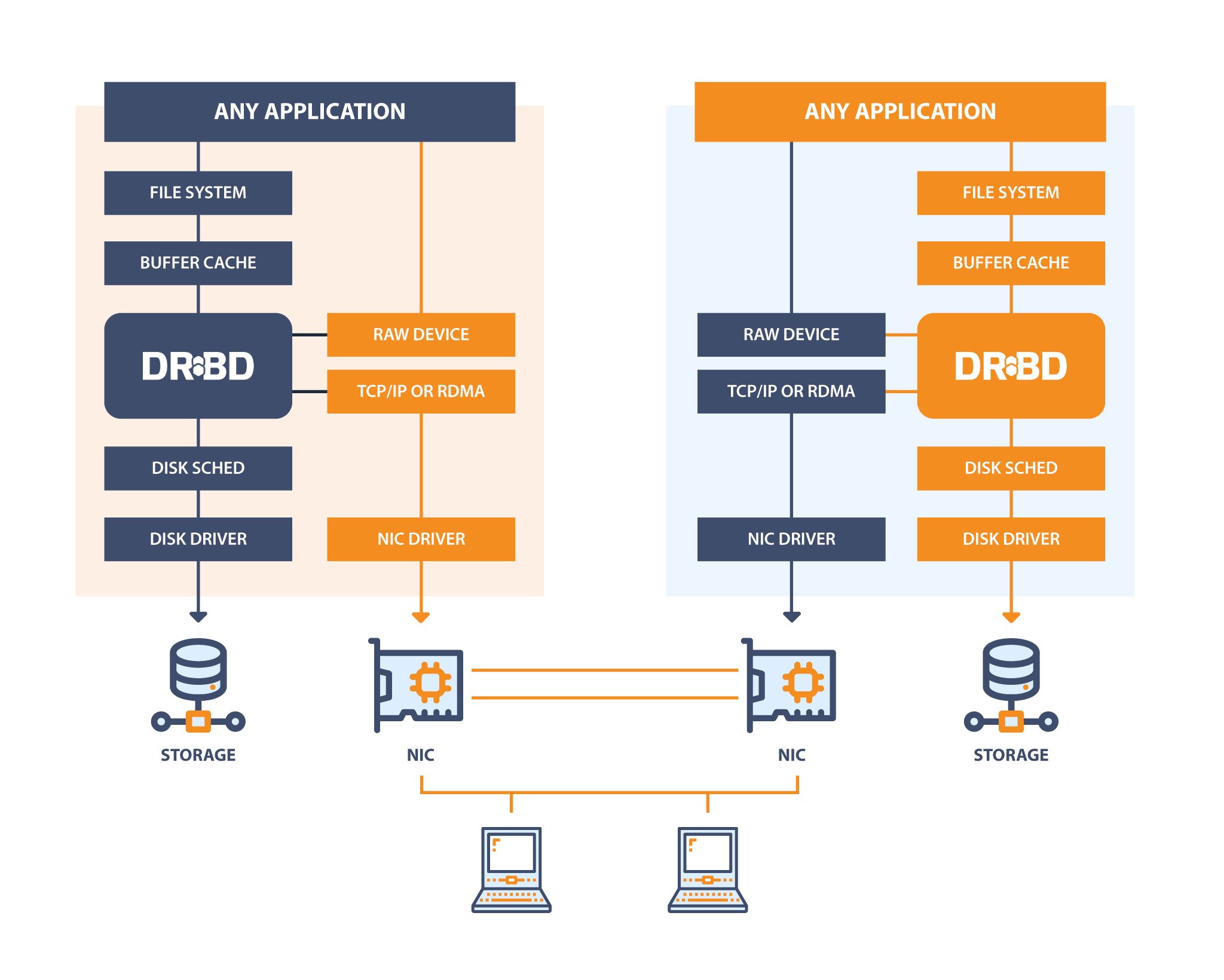


Рисунок 11 – Схема работы DRBD

*NFS*

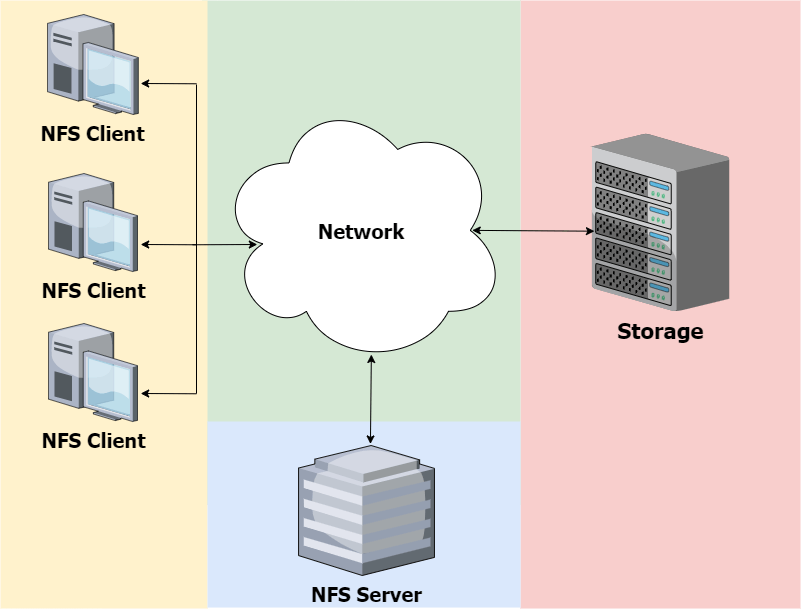
NFS (Network File System) — это протокол сетевого доступа к файловым системам, который позволяет удаленным пользователям получать доступ к файлам и каталогам на сервере в сети. Он был создан для работы в UNIX-подобных операционных системах, хотя теперь он также поддерживается и в других операционных системах.

NFS разработан как открытый стандарт и состоит из двух основных компонентов: клиентского и серверного. Серверный компонент предоставляет доступ к файловой системе и управляет ею, а клиентский компонент позволяет удаленному пользователю получать доступ к этой файловой системе, как показано на рисунке 12.

Одной из главных проблем при использовании NFS является его низкая производительность в условиях высоких нагрузок. Это связано с тем, что NFS не является протоколом, оптимизированным для работы с большим объемом данных или большим количеством одновременных запросов.

Другой проблемой является безопасность. При использовании NFS, данные передаются открытым текстом, что делает их уязвимыми к перехвату и изменению.

Рисунок 12 – Схема работы NFS хранилища

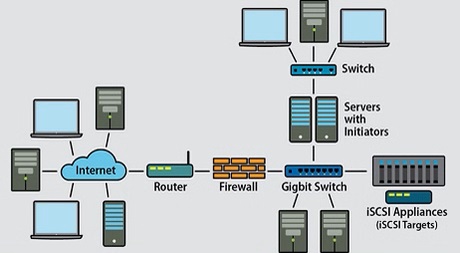


Тем не менее, NFS продолжает быть популярным протоколом для обмена файлами в локальных сетях, благодаря своей простоте и низкой стоимости внедрения. Также в версии NFS 4 были добавлены новые функции, такие как шифрование данных и возможность работать с файлами большого размера, что решает эту проблемы, но тем не менее NFS часто является не лучшим выбором как системы хранения данных для платформы по развёртыванию виртуальных машин.

*SCSI/ISCSI*

SCSI (Small Computer System Interface) и ISCSI (Internet Small Computer System Interface) являются протоколами передачи данных для подключения устройств хранения данных к компьютерам. SCSI — это стандартный интерфейс для подключения устройств хранения данных, таких как жесткие диски, к компьютерам. Он позволяет нескольким хостам использовать один диск, тем самым давая возможность создавать виртуальные машины с высокой доступностью. ISCSI — это стандартный протокол передачи данных, который позволяет использовать SCSI устройства через Интернет или другую сеть, таким образом сильно облегчая задачу администратора по созданию и управлению инфраструктурой. Схема работы iscsi показана на рисунке 13.

Рисунок 13 – Схема работы ISCSI хранилища

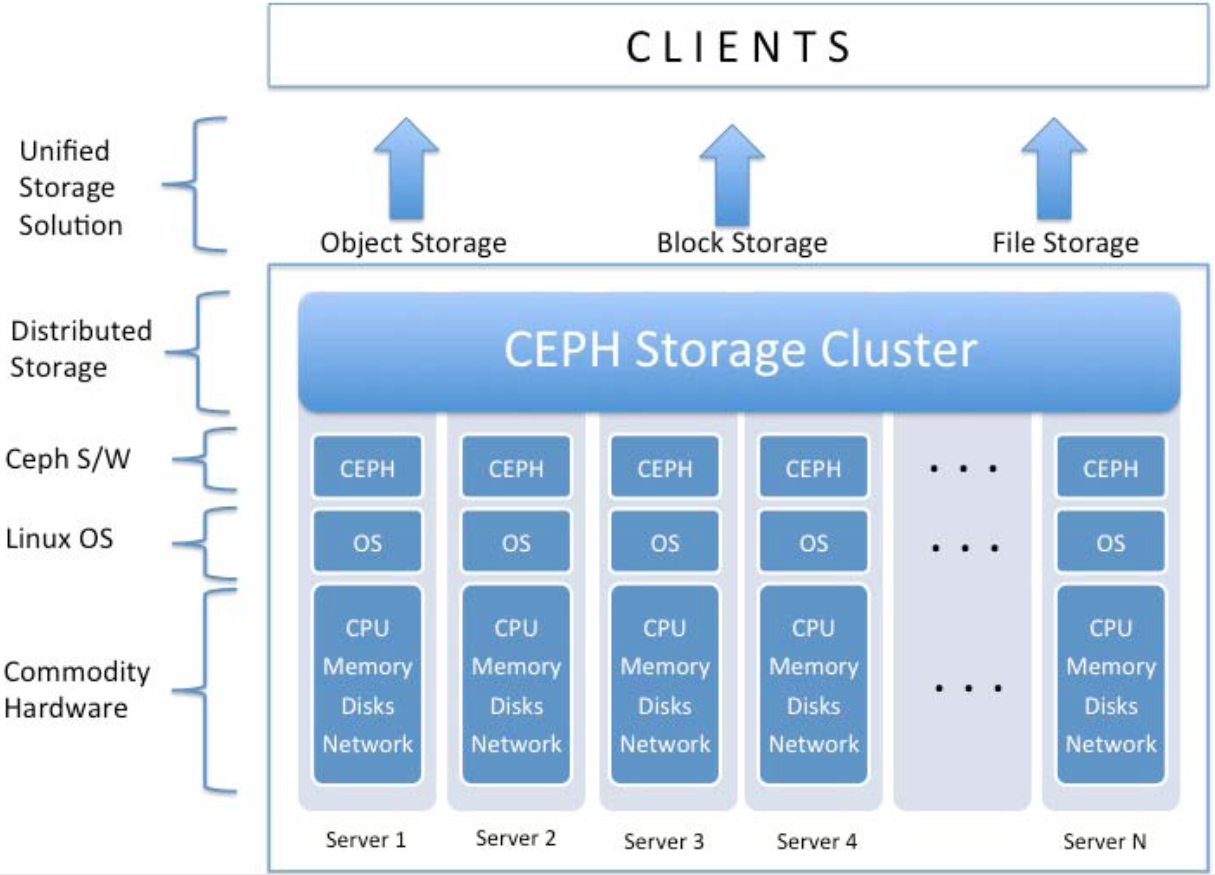


*CEPH*

Ceph — это распределенная система хранения данных с открытым исходным кодом, которая предоставляет высокую доступность, отказоустойчивость и масштабируемость для хранения больших объемов данных. Ceph использует распределенную архитектуру, состоящую из нескольких узлов, которые работают вместе для обработки и хранения данных.

На каждом узле есть диски и демоны, такие как OSD демон, демон монитора и демон менеджмента, которые отвечают за корректную работу eph. OSD демон отвечает за чтение и запись данных, а также их репликацию, тем самым обеспечивая отказоустойчивость и надёжность их хранения. Если некоторые хосты уходят в офлайн и вместе с этим реплик данных становится меньше заданного порога, то запись данных становиться невозможна, до тех пор, пока не возобновят работу хосты, тем самым гарантируя корректность данных. Слежение за хостами и OSD хранилищами лежит на демонах мониторов, которые проверяют heartbeat (онлайн он или нет) для каждого узла, помечая его как офлайн, перенося данные с него на другие узлы. Демон менеджмента не является обязательным узлом, но позволяет собирать метрики с узлов, а также отвечает за изменения в конфигурации. Ceph также поддерживает различные протоколы передачи данных, включая блочный, файловый и объектный доступ к данным, как показано на рисунке 14, что делает его универсальным и удобным в использовании решением для хранения данных.

Рисунок 14 – Доступ к разным типам хранлищ в Ceph



Для работы виртуальных машин с виртуальным хранилищем используются различные протоколы передачи данных, например virtio позволяет ускорить передачу данных и снизить нагрузку на процессор хост-системы.

# 6 РАЗВЕРТЫВАНИЕ И ПРОВЕРКА ПОЛУЧЕННОГО РЕШЕНИЯ

Для тестирования полученного решения, было решено развернуть три виртуальные машины на ноутбуке Dell со следующими характеристиками:

* операционная система Arch Linux;
* процессор Intel Core i7-7700;
* оперативная память 24 ГБ.Twofish

На данном ноутбуке был установлен гипервизор 2 типа qemu/kvm. С его помощью было создано 3 виртуальных машины со следующими конфигурациями:

* операционная система Gentoo;
* процессор 4 ядра, включен Nested VT-x;
* оперативная память 4 ГБ;
* хранилище 64 ГБ, raw.

Для того, чтобы обеспечить максимальный uptime в OpenNebula, при развёртывании было решено использовать pacemaker/corosync для создания кластера высокой доступности с использованием ceph для повышения надёжности хранения данных и их автоматической репликации. Для установки ceph был использован ansible playbook, который развернул ceph на всех узлах.

### 6.1 Развертывание решения

Далее для доступа к ceph из libvirt нужно определить секрет, по котором libvirt и будет получать доступ к пулам из ceph.

Для этого на узлах, с хранилищем нужно зайти в */etc/ceph* и сгенерировать ключ для подключения libvirt с помощьою команды *ceph auth get-key client.libvirt*:

*cd /etc/ceph/*

*ceph auth get-key client.libvirt | tee client.libvirt.key*

На всех узлах нужно создать xml файл, в котором с помощью uuidgen будет сгенерированон UUID секрета для libvirt:

*cat > secret.xml <<EOF*

*<secret ephemeral='no' private='no'>*

*<uuid>$(uuidgen) </uuid>*

*<usage type='ceph'>*

*<name>client.libvirt secret</name>*

*</usage>*

*</secret>*

*EOF*

Далее нужно под пользователем oneadmin на все гипервизоры скопировать secret.xml и client.libvirt.key в домашний каталог пользователя oneadmin:

*sshpass -p "horizon" scp client.libvirt.key secret.xml oneadmin@server:*

На каждом гипервизоре добавить секрет под oneadmin:

*su - oneadmin*

*virsh -c qemu:///system secret-define secret.xml*

Скопировать UUID из secret.xml и ввести команду

*virsh -c qemu:///system secret-set-value --secret UUID --base64 $(cat client.libvirt.key)*

Затем удаляем ключ, чтобы предотвратить его возможную утечку:

*rm client.libvirt.key*

На всех гипервизорах создаем файл */etc/corosync/corosync.conf* с следующим содержимым:

*# Totem Protocol Configuration*

*totem {*

*version: 2*

*cluster\_name: TEST\_CLUSTER*

*secauth: off*

*bindnetaddr: 10.10.101.39*

*transport: udpu*

*}*

*# Nodelist - Server List*

*nodelist {*

*node {*

*ring0\_addr: h30*

*nodeid: 1*

*}*

*node {*

*ring0\_addr: h31*

*nodeid: 2*

*}*

*node {*

*ring0\_addr: h32*

*nodeid: 3*

*}*

*}*

*# Quorum configuration*

*quorum {*

*provider: corosync\_votequorum*

*two\_node: 0*

*}*

*# Corosync Log configuration*

*logging {*

*to\_logfile: yes*

*logfile: /var/log/corosync/corosync.log*

*to\_syslog: yes*

*debug: on*

*timestamp: on*

*}*

*service {*

*}*

Затем нужно запустить corosync и pacemaker:

*rc-service corosync start*

*rc-service pacemaker start*

После этого нужно проверить что всё было настроено правильно, и они работают. Для этого можно использовать команду crm status. Затем можно приступить к непосредственной настройке HA. Для этого нужно добавить fencing device, который будет выключать хост в случае, если она уходит в офлайн, котролируя доступ к ресурсам кластера хостами.

*crm*

*crm(live)# configure*

*crm(live)configure# primitive st-null stonith:null params hostlist=h30 h31 h32*

*crm(live)configure# clone fencing st-null*

Затем активируем логический том системы LVM, на котором будут хранится все данные, которые должны быть разделены между хостами:

*crm(live)configure# primitive hcs-ha LVM params volgrpname=TAM-VG0 op monitor timeout=30 interval=10*

Далее примонтируем файловую систему этого логического тома. Мною была использована файловая система xfs.

*crm(live)configure# primitive fs\_work Filesystem params device=/dev/TAM-VG0/hcs-ha directory=/data/ fstype=xfs op monitor interval=10s*

Запускаем контейнер с OpenNebula.

*crm(live)configure# primitive docker\_start docker params image=one:latest name=one run\_opts=--network=host --restart=always --security-opt=seccomp:unconfined -v /var/tmp:/var/tmp -v /data/onevol:/onevol --stop-timeout=90 meta target-role=Started*

Далее назначаем плавающий IP адрес на ранее созданый при помощи ovs-vsctl порт в бридже sw0:

*crm(live)configure# primitive vip\_sgu IPaddr2 params ip=10.10.101.39 cidr\_netmask=24 nic=ha-port op monitor interval=10 timeout=20*

Добавляем все ранее созданные примитивы в группу, чтобы они все запускались на одном хосте, а не на разных:

*crm(live)configure# group one one\_lvm fs\_work docker\_start vip\_sgu*

Делаем комит нашей конфигурации:

*crm(live)configure# commit*

Таким образом получаем рабочее HA решение, готовое к дальнейшему тестированию.

### 6.2 Проверка полученного решения

Тесты проводились как для всех встроенных в OpenNebula алгоритмов планирования, таких как Packing, Striping и Load-aware, так и собственного решения на основе рангов (Ranks). Для проверки работы был написан скрипт, который создавал и удалял виртуальные машины, тем самым эмулируя настоющую нагрузку. Полученные в результаты отображены в таблице 1.

Таблица 1 – Данные об времени создания и удаления ВМ

|  |  |  |
| --- | --- | --- |
| Планировщик | Среднее время создания (с) | Среднее время удаления (с) |
| Packing | 26,21 | 15,88 |
| Striping | 20,21 | 13,42 |
| Load-balance | 17,33 | 12,40 |
| Ranks | 16,97 | 12,03 |

Видно, что полученный алгоритм на основе рангов имеет меньшее время создания и удаления виртуальных машин, тем самым уменьшив время простоя для виртуальных машин и повысив общую производительность системы. Таким образом использование рангов позволило оптимизировать диспетчеризацию ресурсов и уменьшить время нахождения виртуальных машин в очереди ожидания.

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения выпускной квалификационной работы магистра (магистерской диссертации) работы были рассмотрены и внедрены различные методы обеспечения безопасности инфраструктуры, для снижения риска их взлома, а также были проанализированы различные системы хранения данных и организация кластеров высокой доступности с их использованием.

Были проанализированы различные платформы виртуализации и среди них была выбрана платформа OpenNebula, на которой был реализован собственный алгоритм планирования развёртывания виртуальных машин с использованием рангов, а также было произведено его сравнение с встроенными алгоритмами планирования, показывающие повышение общей производительности системы. Благодаря тому, что при реализации алгоритма были использованы стандартный набор инструментов OpenNebula, данный алгоритм можно использовать в любом центр обработки данных, за счёт того, что OpenNebula является открытой системой, а также из-за простоты её установки, настройки и поддержки.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Graziano, Charles. A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds Project // Iowa State University – 2011. – P. 4
2. Virtualization: Benefits, drawbacks and defining features [Электронный ресурс]. - URL: <https://www.techrepublic.com/article/virtualization-benefits-drawbacks-defining-features/> (дата обращения 03.12.2022)
3. What is LXD? [Электронный ресурс]. - URL: <https://linuxcontainers.org/lxd/> (дата обращения 21.04.2023)
4. Что такое гипервизор? [Электронный ресурс]. - URL: <https://aws.amazon.com/ru/what-is/hypervisor/> (дата обращения 21.04.2023)
5. СИСТЕМЫ УПРАВЛЕНИЕ СЕКРЕТАМИ [Электронный ресурс]. - URL: <https://cyberleninka.ru/article/n/sistemy-upravlenie-sekretami/viewer> (дата обращения 08.12.2021)
6. Manage Secrets & Protect Sensitive Data [Электронный ресурс]. - URL: https://www.vaultproject.io/ (дата обращения 23.12.2021)
7. VMware ESXi [Электронный ресурс]. - URL: https://www.vmware.com/products/esxi-and-esx.html (дата обращения 02.12.2021)
8. Enterprise Workload Platform vSphere [Электронный ресурс]. - URL: https://www.vmware.com/products/vsphere.html (дата обращения 03.02.2023)
9. [OpenStack Launches as Independent Foundation, Begins Work Protecting, Empowering and Promoting OpenStack [Электронный ресурс]. - URL: https://www.businesswire.com/news/home/20120919005997/en/OpenStack-Launches-Independent-Foundation-Begins-Work-Protecting](https://www.businesswire.com/news/home/20120919005997/en/OpenStack-Launches-Independent-Foundation-Begins-Work-Protecting) (дата обращения 02.12.2022)
10. OpenStack Compute (nova) [Электронный ресурс]. - URL: https://docs.openstack.org/nova/latest/(дата обращения 02.12.2022)
11. Glance Administration Guide [Электронный ресурс]. - URL: <https://docs.openstack.org/glance/latest/admin/index.html> (дата обращения 02.12.2022)
12. Neutron’s documentation [Электронный ресурс]. - URL: <https://docs.openstack.org/neutron/latest/> (дата обращения 01.02.2022)
13. About OpenNebula [Электронный ресурс]. - URL: https://opennebula.io/about-opennebula/ (дата обращения 25.01.2022)
14. Облачная инфраструктура [Электронный ресурс]. - URL: https://www.basealt.ru/infrastructure (дата обращения 21.03.2023)
15. OpenNebula 4.8 Design and Installation Guide // OpenNebula Project - 2014. – P.1
16. The DRBD9 User’s Guide [Электронный ресурс]. - URL: https://linbit.com/drbd-user-guide/drbd-guide-9\_0-en/ (дата обращения 11.02.2023)

# **ПРИЛОЖЕНИЕ А**

**Исходный код**

На рисунке А.1 изображен QR-код, который позволит изучить исходный код выпускной квалификационной работы магистра (магистерской диссертации).



Рисунок A.1 – QR-код на репозиторий работы в GitHub