СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Облачные услуги — это способ предоставления, потребления и управления технологией. Они выводят гибкость и эффективность на революционный уровень, путем эволюции способов управления, таких как резервирование, самообслуживание, безопасность и непрерывность, которые соединяют физическую и виртуальную среду. Следовательно, возрастает потребность в качественно продуманной архитектуре, позволяющей правильно и надежно организовать облачную инфраструктуру.

Для эффективной облачной инфраструктуры требуется эффективная структура и организация. Определение и использование стандартов на каждом этапе работы, при размещении в стойках от отдельных компьютеров до отдельных кабелей и от рядовых операций до безопасности, позволяет сэкономить значительное время и правильно организовать процессы.

Чтобы спланировать и выполнить план, не нужны гигантские усилия. Небольшая команда из специалистов и бизнес-пользователей может создать обоснованный план и организовать свою работу в облачной инфраструктуре. Эта выделенная группа может намного эффективнее построить и управлять нестандартной облачной инфраструктурой, чем если компании будут просто продолжать добавлять дополнительные серверы и сервисы для поддержки центра обработки данных.

IaaS (Infrastructure as a Service) — это предоставление пользователю компьютерной и сетевой инфраструктуры и их обслуживание как услуги в форме виртуализации, то есть виртуальной инфраструктуры. Другими словами, на базе физической инфраструктуры дата-центров (ДЦ) провайдер создает виртуальную инфраструктуру, которую предоставляет пользователям как сервис.

Технология виртуализации ресурсов позволяет физическое оборудование (серверы, хранилища данных, сети передачи данных) разделить между пользователями на несколько частей, которые используются ими для выполнения текущих задач. Например, на одном физическом сервере можно запустить сотни виртуальных серверов, а пользователю для решения задач выделить время доступа к ним.

# 1 ПОСТАНОВКА ЗАДАЧИ

Конечная цель проектирования — разработка виртуальной инфраструктуры для реализации облачных услуг.

Виртуальная инфраструктура должна обладать следующими характеристиками:

- Использование, по возможности, продуктов, распространяющихся под свободной лицензией (GNU GPL) для организации инфраструктуры;
- Устранение единой точки отказа при проектировании инфраструктуры;
- Защита от распределенных атак на отказ (DDoS);
- Использование инфраструктуры в бизнесе для предоставления облачных услуг клиентам.

Аппаратное обеспечение должно базироваться в дата-центре ориентированным на требования стандарта Tier III, состоящего из информационной, телекоммуникационной и инженерной инфраструктуры, с возможностью аренды выделенных серверов и аппаратной защиты от DDoS-атак, а также поддержкой аппаратного RAID. Рекомендуемые характеристики выделенных серверов виртуализации:

- 2xIntel® Xeon E5430 @ 2.66GHz;
- Минимальный объем ОЗУ 32 Гб;
- Минимум 960 Гб места на жестком диске (SSD);
- Поддержка аппаратного RAID;
- Операционная система не ниже Debian 7 GNU/Linux или CentOS 6.5;
- Интернет-канал с пропускной способностью не менее 100 Мб/с;
- Возможность добавления дополнительных IP-адресов к серверам;
- Возможность удаленного доступа к серверам посредством IPMI (Intelligent Platform Management Interface).

Следует предусмотреть возможность развертывания дополнительных виртуальных серверов, для организации DNS-серверов, системы мониторинга и платежной системы. Также необходимо предусмотреть наличие системы хранения данных (СХД) для резервных копий, а также расширение дискового пространства на выделенных серверах.

Для разработки виртуальной инфраструктуры необходимо реализовать следующие этапы:

- Выбор технологий виртуализации;
- Выбор физических серверов на основе услуги IaaS в ДЦ;
- Приобретение лицензий на программное обеспечение;
- Приобретение подсетей IP-адресов;
- Создание и подключение инфраструктуры мониторинга, платежной системы (биллинга), резервного копирования;
- Реализация аппаратной защиты от DDoS-атак;
- Выбор перечня PaaS-услуг;
- Внедрение перечня предоставляемых PaaS-услуг;
- Создание руководства администратора по виртуализации;
- Создание руководства для клиентов по часто задаваемым вопросам;
- Внедрение инфраструктуры на рынке PaaS-услуг.

# 2 ОБЗОР СОВРЕМЕННЫХ МЕТОДОВ И ТЕХНОЛОГИЙ СЕРВЕРНОЙ ВИРТУАЛИЗАЦИИ

Термин «облачные вычисления» сегодня уже достаточно хорошо известен и в информационных технологиях (ИТ), и в бизнес-кругах. Почти каждую неделю появляются новые статьи, книги, презентации об облачных вычислениях — новой сервисной модели предоставления вычислительных услуг.

За время существования информационных технологий сменилось несколько моделей построения информационных систем. Все начиналось с монолитной архитектуры (mainframe), когда и база данных, и приложения работали на одном большом компьютере, а пользователи сидели у «тонких» терминалов, которые только отображали информацию. У такой архитектуры было много недостатков, и ее сменила более перспективная архитектура «клиент-сервер». В ней был свой выделенный сервер баз данных (БД) и пользователи на «толстых» клиентах, которые разгружали сервер БД. Затем появилась еще более современная архитектура — многоуровневая (или трехуровневая), где логика приложений была вынесена на отдельный компьютер, называемый сервером приложений, а пользователи работали на «тонких» клиентах через веб-браузеры. Большинство приложений сегодня выполнено именно в этой архитектуре. Она подразумевает развертывание всей ИТ-инфраструктуры на территории заказчика [1].

Облачные вычисления — это следующий шаг в эволюции архитектуры построения информационных систем. Благодаря огромным преимуществам этого подхода очевидно, что многие информационные системы в ближайшее время будут перенесены в облако. Этот процесс уже начался и его игнорирование или недооценка может привести к поражению в конкурентной борьбе на рынке. Имеется ввиду не только отставание ИТ, или неоправданные затраты на него, но и отставание в развитии основного бизнеса компании, зависящего от гибкости ИТ-инфраструктуры и скорости вывода новых сервисов и продуктов на рынок.

ИТ-директор американского правительства Вивек Кундра, в феврале 2011 года опубликовал стратегию стратегию переноса части информационных систем в облако. Документ под названием «Federal Cloud Computing Strategy» четко описы-

вает порядок и сроки переноса. Цель работ — уменьшение сложности и повышение управляемости ИТ, увеличение нагрузки оборудования до 70-80%, уменьшение количества центров обработки данных.

Основным требованием, предъявляемым к центрам обработки данных является отказоустойчивость. При этом подразумевается отключение ЦОД как на время планово-предупредительных работ и профилактики оборудования, так и внеплановых аварийных ситуаций.

Классификация Tier описывает надежность функционирования ЦОД и является необходимой для компаний, как желающих построить свой ЦОД, так и для арендующих чужие вычислительные мощности. В зависимости от критичности бизнеса, в зависимости от потерь, которые понесет компания в случае остановки бизнес-процессов выбирается тот или иной уровень надежности. В свою очередь, высокий уровень надежности требует высоких материальных и эксплуатационных затрат, поэтому и стоимость вычислительных мощностей зависит от уровня надежности ЦОД [2].

На сегодняшний день существует четыре уровня надежности ЦОД названные Tier I, Tier II, Tier III и Tier IV, которые были введены организацией Uptime Institute (Институт бесперебойных процессов, США):

- Tier I: время простоя 28,8 часов в год, коэффициент отказоустойчивости 99,671%;
- Tier II: 22,0 часа в год, 99,749%;
- Tier III: 1,6 часа в год, 99,982%;
- Tier IV: 0,4 часа в год, 99,995%.

ЦОД уровня Tier I (базовый уровень) подвержен нарушениями работы как от плановых, так и от внеплановых действий. Применение фальшпола, источников бесперебойного питания (ИБП), дизель-генераторных установок (ДГУ) не обязательно. Если ИБП и ДГУ используются, то выбираются более простые модели, без резерва, с множеством точек отказа. Возможны самопроизвольные отказы оборудования. К простою ЦОД также приведут ошибки в действиях обслуживающего персонала. В таких ЦОД отсутствует защита от случайных и намеренных событий, обусловленных действиями человека.

В ЦОД уровня Tier II (с резервированными компонентами) время простоя возможно в связи с плановыми и внеплановыми работами, а также аварийными ситуациями, однако оно сокращено благодаря внедрению одной резервной единицы оборудования в каждой системе. Таким образом, системы кондиционирования, ИБП и ДГУ имеют одну резервную единицу, тем не менее, профилактические работы требуют отключения ЦОД. В центрах обработки данных с резервированными компонентами требуется наличие минимальных защитных мер от влияния человека.

Третий уровень надежности (уровень с возможность параллельного проведения ремонтных работ) требует осуществления любой плановой деятельности без остановки ЦОД. Под плановыми работами подразумевается профилактическое и программируемое техническое обслуживание, ремонт и замена компонентов, добавления или удаление компонентов, а также их тестирование. В таком случае необходимо иметь резерв, благодаря которому можно пустить всю нагрузку по другому пути, во время работ на первом. Для реализации Tier III необходима схема резервирования блоков схем кондиционирования, ИБП, ДГУ N+1, также требуется наличие двух комплектов трубопроводов для системы кондиционирования, построенной на основе чиллера (холодильной машины). Строительные требования обязывают сохранять работоспособность ЦОД при большинстве случаев намеренных и случайных вмешательств человека. Также следует предусмотреть резервные входы, дублирующие подъездные пути, контроль доступа, отсутствие окон, защиту от электромагнитного изучения.

Четвертый уровень надежности ЦОД (отказоустойчивый) характеризуется безостановочной работой при проведении плановых мероприятий и способен выдержать один серьезный отказ без последствий для критически важной нагрузки. Необходим дублированный подвод питания, резервирование системы кондиционирования и ИБП по схеме 2(N+1). Для дизель-генераторных установок необходима отдельная площадка с зоной хранения топлива. Tier IV требует защиту от всех потенциальных проблем в связи с человеческим фактором. Регламентированы избыточные средства защиты от намеренных или случайных действий человека. Учтено влияние сейсмоявлений, потопов, пожаров, ураганов, штормов, терроризма.

Дата-центры по виду использования подразделяют на корпоративные и коммерческие (аутсорсинговые). Корпоративные ДЦ предназначены для обслуживания конкретной компании, коммерческие, в свою очередь, предоставляют услуги всем желающим.

Некоторые ДЦ предлагают клиентам дополнительные услуги по использованию оборудования, по автоматическому уходу от различных видов атак. Команда специалистов круглосуточно производит мониторинг серверов. Для обеспечения сохранности данных используются системы резервного копирования. Для предотвращения кражи данных, в дата-центрах используются различные системы ограничения физического доступа, системы видеонаблюдения.

Дата-центры предоставляют несколько основных типов услуг, среди которых:

- Виртуальный хостинг (shared hosting);
- Виртуальный сервер (virtual private/dedicated server);
- Выделенный сервер (dedicated server);
- Размещение сервера (colocation);
- Выделенная зона (dedicated area).

Виртуальный хостинг используется для размещения большого количества сайтов на одном веб-сервере. В основном для построения веб-сервра используется типичный стек технологий LAMP, где в качестве операционной системы выступает GNU/Linux, http-сервер Apache (зачастую в связке с nginx), сервер баз данных MySQL, интерпретируемые скриптовые языки PHP, Perl, Python. Существует решение на базе ОС Windows Server, где в качестве http-сервера используется IIS, в качестве СУБД выступает MS SQL, а также существует поддержка платформы ASP.NET. Разделение ресурсов на виртуальном хостинге основывается на ограничении дискового пространства, сетевого трафика, количества используемых доменов, почтовых ящиков, баз данных, FTP-аккаунтов, ограничение на использование процессорного времени, памяти для PHP-скриптов и так далее.

Виртуальный выделенный сервер эмулирует работу отдельного физического сервера. На одной физической машине может быть запущено несколько виртуальных серверов, при этом каждый виртуальный сервер имеет свои процессы, ресурсы и отдельное администрирование. Для реализации виртуальных машин использу-

ются технологии виртуализации, как системы с открытым исходным кодом, так и коммерческие.

В случае выделенного сервера, клиенту целиком предоставляется отдельная физическая машина. Владелец сервера имеет возможность смены конфигурации оборудования, установки любой операционной системы. Такой тип хостинга подходит для высоконагруженных проектов.

Размещение сервера отличается от услуги предоставления выделенного сервера тем, что ДЦ размещает у себя сервер, который заранее подготовил клиент. Дата-центр подключает его в общую инфраструктуру ЦОДа, обеспечивает бесперебойное электропитание, охлаждение, доступ к сетевому каналу, удаленный доступ к серверу, охрану, мониторинг и другие услуги.

Выделенная зона предоставляется в основном для специальных клиентов, имеющих строгие нормы безопасности. В этом случае дата-центр предоставляет выделенную зону, обеспеченную электроснабжением, холодоснабжением и системами безопасности, а клиент сам создает свой дата-центр внутри этого пространства.

Также можно выделить такую услугу, как аренда телекоммуникационных стоек, которая является частным случаем размещения сервера, с отличием в том, что арендаторами в основном являются юридические лица.

При построении облачной инфраструктуры важную роль играет виртуализация.

Виртуализация — абстракция вычислительных ресурсов и предоставление пользователю системы, которая инкапсулирует (скрывает в себе) собственную реализацию. Термин «виртуализация» появился в шестидесятых годах XX века, а в девяностых — стали очевидны перспективы подхода: с ростом аппаратных мощностей, как персональных компьютеров, так и серверных решений, вскоре представится возможность использовать несколько виртуальных машин на одной физической платформе.

Понятие виртуализации можно условно разделить на две категории [3]:

– Виртуализация платформ, продуктом этого вида виртуализации являются виртуальные машины — некие программные абстракции, запускаемые на платформе реально аппаратно-программных систем;

– Виртуализация ресурсов преследует целью комбинирование или упрощение представления аппаратных ресурсов для пользователя и получение неких пользовательских абстракций оборудования, пространств имен, сетей.

Когда производится виртуализация, существует множество способов ее осуществления. Фактически есть несколько путей, с помощью которых достигаются одинаковые результаты через разные уровни абстракции [4]:

– Эмуляция аппаратных средств;
– Полная виртуализация;
– Паравиртуализация;
– Виртуализация уровня ОС.

Эмуляция аппаратных средств является одним из самых сложных методов виртуализации. В то же время, главной проблемой при эмуляции аппаратных средств является низкая скорость работы, в связи с тем, что каждая команда моделируется на основных аппаратных средствах. В эмуляции оборудования используется механизм динамической трансляции, то есть каждая из инструкций эмулируемой платформы заменяется на заранее подготовленный фрагмент инструкций физического процессора [5]. Архитектура процесса эмуляции представлена на рис. 2.1.

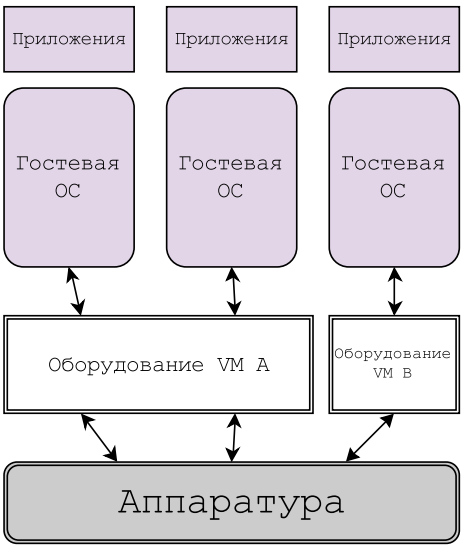


Рисунок 2.1 – Эмуляция аппаратных средств

Примерами виртуализации посредством эмуляции являются программные платформы QEMU и Bochs.

Система QEMU поддерживает два режима эмуляции: пользовательский и системный. Пользовательский режим эмуляции позволяет процессу, созданному на одном процессоре, работать на другом (выполняется динамический перевод инструкций для принимающего процессора и конвертация системных вызовов Linux). Системный режим позволяет эмулировать систему целиком, включая процессор и разнообразную периферию. Достоинством QEMU является его быстрый и компактный динамический транслятор. Динамический транслятор позволяет во время исполнения переводить инструкции целевого (гостевого) процессора в инструкции центрального процессора хоста для обеспечения эмуляции. QEMU обеспечивает динамическую трансляцию преобразованием целевой инструкции в микрооперации. Эти микрооперации представляют собой элементы С-кода, которые компилируются в объекты. Затем запускается основной транслятор, который отображает целевые инструкции на микрооперации для динамической трансляции. Такой подход не только эффективен, но и обеспечивает переносимость.

Использование QEMU в качестве эмулятора персонального компьютера обеспечивает поддержку разнообразных периферийных устройств. Сюда входят стандартные периферийные устройства — эмулятор аппаратного видеоадаптера (VGA), мыши и клавиатуры PS/2, интерфейс IDE для жестких дисков, интерфейс CD-ROM и эмуляция дисковода. Кроме того, QEMU имеет возможность эмуляции сетевых адаптеров NE2000 (PCI), последовательных портов, многочисленных звуковых плат и контроллера PCI Universal Host Controller Interface (UHCI), Universal Serial Bus (USB) (с виртуальным USB концентратором). Также поддерживается до 255 процессоров с поддержкой симметричной многопроцессорности (SMP).

Полная виртуализация использует гипервизор, который осуществляет связь между гостевой ОС и аппаратными средствами физического сервера. В связи с тем, что вся работа с гостевой операционной системой проходит через гипервизор, скорость работы данного типа виртуализации ниже чем в случае прямого взаимодействия с аппаратурой. Основным преимуществом является то, что в ОС не вносятся никакие изменения, единственное ограничение — операционная система должна поддерживать основные аппаратные средства. Архитектура полной виртуализации представлена на рис. 2.2.

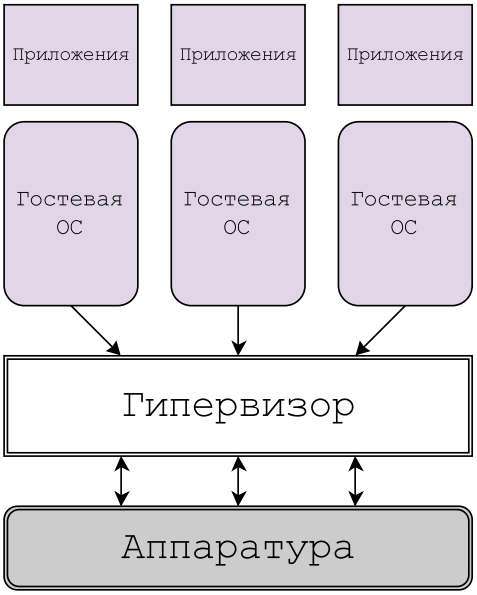


Рисунок 2.2 – Архитектура полной виртуализации

Полная виртуализация возможна исключительно при условии правильной комбинации оборудования и программного обеспечения. Например, она была невозможной ни в серии IBM System/360, за исключением IBM System/360-67, ни в ранних IBM System/370, пока IBM не добавила оборудование виртуальной памяти в своих System/370 в 1972 г. Аналогичная ситуация и с платформой x86: полная виртуализация была возможна не в полной мере, до добавления технологий AMD-V и Intel VT.

KVM (Kernel-based Virtual Machine) — программное решение, обеспечивающее виртуализацию в среде Linux, которая поддерживает аппаратную виртуализацию на базе Intel VT (Virtualization Technology) либо AMD SVM (Secure Virtual Machine) [6]. KVM не выполняет никакой самоэмуляции, вместо этого, программа, работающая в пользовательском пространстве, применяет интерфейс /dev/kvm для настройки адресного пространства гостевого виртуального сервера, берет его смоделированные ресурсы ввода/вывода и отображает его образ на образ хоста.

В архитектуре KVM, виртуальная машина выполняется как обычный Linux-процесс, запланированный стандартным планировщиком Linux. На самом деле виртуальный процессор представляется как обычный Linux-процесс, это позволяет KVM пользоваться всеми возможностями ядра Linux. Эмуляцией устройств управляет модифицированная версия QEMU, которая обеспечивает эмуляцию BIOS,

шины PCI, шины USB, а также стандартный набор устройств, таких как дисковые контроллеры IDE и SCSI, сетевые карты и другие.

Паравиртуализация имеет некоторые сходства с полной виртуализацией. Этот метод использует гипервизор для разделения доступа к основным аппаратным средствам, но объединяет код, касающийся виртуализации, в непосредственно операционную систему, поэтому недостатком метода является то, что гостевая ОС должна быть изменена для гипервизора. Но паравиртуализация существенно быстрее полной виртуализации, скорость работы виртуальной машины приближена к скорости реальной, это осуществляется за счет отсутствия эмуляции аппаратуры и учета существования гипервизора при выполнении системных вызовов в коде ядра. Вместо привилегированных операций совершаются гипервызовы обращения ядра гостевой ОС к гипервизору с просьбой о выполнении операции. Архитектура паравиртуализации представлена на рис. 2.3.

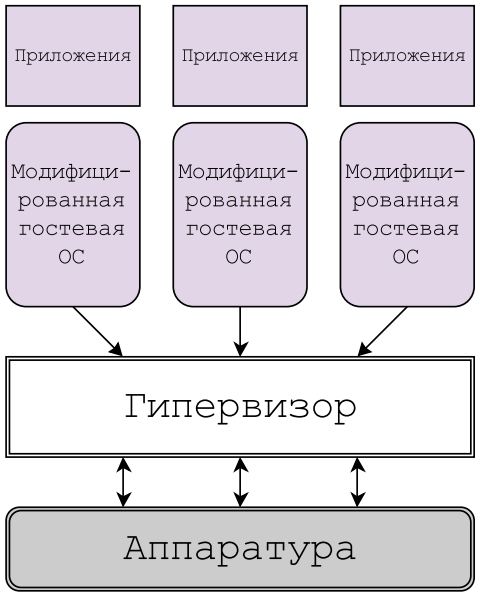


Рисунок 2.3 – Архитектура паравиртуализации

Для организации паравиртуализации используется программный продукт Xen.

Xen — это монитор виртуальных машин (VMM, Virtual Machine Monitor) или гипервизор с поддержкой паравиртуализации для процессоров x86 архитектуры, распространяющийся с открытым исходным кодом. Xen может организовать совместное безопасное исполнение нескольких виртуальных машин на одной физи-

ческой системе, с производительностью близкой к непосредственной. Он перекладывает большинство задач по поддержке аппаратуры на гостевую операционную систему, работающую в управляющей виртуальной машине, также известной как домен 0 (dom0) [7]. Сам Xen содержит только код, необходимый для обнаружения и запуска остальных процессоров системы, настройки обработки прерываний и нумерации PCI шины. Драйверы устройств работают внутри привилегированной гостевой операционной системы, а не в самом Xen. Такой подход обеспечивает совместимость с большинством устройств, поддерживаемых Linux. Сборка XenLinux по умолчанию содержит поддержку большинства серверного сетевого и дискового оборудования, но при необходимости можно добавить поддержку других устройств, переконфигурировав Linux-ядро стандартным способом.

В паравиртуальном режиме (PV) оборудование не эмулируется, и гостевая ОС должна быть специальным образом модифицирована, чтобы работать в таком окружении. Начиная с версии 3.0, ядро Linux поддерживает запуск в паравиртуальном режиме без перекомпиляции со сторонними патчами. Преимущество режима паравиртуализации состоит в том, что он не требует поддержки аппаратной виртуализации со стороны процессора, а также не тратит вычислительные ресурсы для эмуляции оборудования на шине PCI. Режим аппаратной виртуализации (HVM) появился в Xen, начиная с версии 3.0 гипервизора, и требует поддержки со стороны оборудования. В этом режиме для эмуляции виртуальных устройств используется QEMU, который «неповоротлив» даже с паравиртуальными драйверами. Однако со временем поддержка аппаратной виртуализации в оборудовании получила настолько широкое распространение, что используется даже в процессорах лэптопов. Поэтому у разработчиков возникло желание использовать быстрое переключение контекста исполнения между гипервизором и гостевой ОС и в паравиртуальном режиме, используя возможности оборудования. Так появился новый режим — аппаратная паравиртуализация (PVH), который доступен в Xen с версии 4.4.

Виртуализация уровня операционной системы отличается от других. Она использует технику, при которой сервера виртуализируются непосредственно над ОС. Недостатком метода является то, что поддерживается одна единственная операционная система на физическом сервере, которая изолирует контейнеры друг

от друга. Преимуществом виртуализации уровня ОС является «родная» производительность. Виртуализация уровня ОС — метод виртуализации, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя, вместо одного. Эти экземпляры с точки зрения пользователя полностью идентичны реальному серверу. Для систем на базе UNIX, эта технология может рассматриваться как улучшенная реализация механизма chroot. Ядро обеспечивает полную изолированность контейнеров, поэтому программы из разных контейнеров не могут воздействовать друг на друга. Архитектура виртуализации уровня ОС представлена на рис. 2.4.

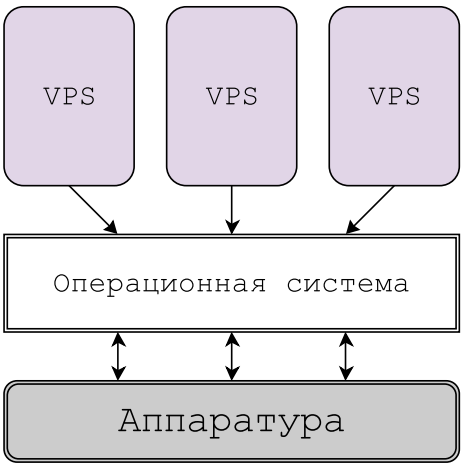


Рисунок 2.4 – Архитектура виртуализации уровня ОС

Для реализации виртуализации уровня операционной системы часто используется продукт OpenVZ.

OpenVZ разрабатывается как патч к исходным текстам ядра Linux. В модифицированном ядре добавлен массив дополнительных сущностей — виртуальных окружений (VE) или контейнеров (CT), а для всех имеющихся объектов (процессы, сокеты и прочие) введены дополнительные поля — номер контейнера, к которому этот объект относится, и номер объекта внутри контейнера. Каждое виртуальное окружение имеет собственный набор квот на потребление системных ресурсов и отдельный каталог для использования в качестве корневой файловой системы. Дополнительные модули ядра — vzdev, vzmon и прочие, отвечают за работу ограничений, мониторинг, эмуляцию сети в контейнере, сохранение и восстановление текущего состояния запущенных контейнеров. К преимуществам OpenVZ, по сравнению с

более универсальными инструментами виртуализации, такими как Xen и KVM, относят прозрачный доступ из внешней системы к процессам, файлам и прочим ресурсам в контейнере.

OpenVZ разрабатывается фирмой Parallels как часть более крупного коммерческого продукта под названием Parallels Virtuozzo Containers (PVC) [8]. В число преимуществ Virtuozzo, по сравнению с OpenVZ, входят:

- Файловая система VZFS;
- Управление через графическую консоль и веб-интерфейс;
- Программный интерфейс на базе XML для создания собственных инструментов управления и контроля;
- Средства миграции с физическом системы в контейнер и обратно;
- Средства контроля за полосой и суммарным потреблением трафика;
- Интеграция с Plesk, коммерческой панелью управления хостингом;
- Круглосуточная техническая поддержка.

VZFS позволяет совмещать файловые системы контейнеров, при этом базовый образ используется всеми контейнерами, а изменения в нем для каждого контейнера сохраняются раздельно. Преимущества такого подхода:

- Место, занимаемое программами на диске, становится фиксированным и не зависит от количества контейнеров, в которые эти программы установлены;
- Уменьшается расход оперативной памяти, так как код нескольких экземпляров программы или библиотеки, запущенной из одного и того же исполняемого файла, размещается в памяти в единственном экземпляре;
- Обновление программного обеспечения в группе контейнеров выполняется одной командой.

LXC (Linux Containers) — система виртуализации на уровне операционной системы. Данная система сходна с OpenVZ и Linux-VServer для Linux, а также FreeBSD jail и Solaris Containers. LXC основана на технологии cgroups, входящей в ядро Linux, начиная с версии 2.6.29. Ее нельзя рассматривать как законченный продукт, фактически это набор из нескольких совершенно самостоятельных функций ядра Linux и пользовательских утилит, которые позволяют удобно создавать и

управлять изолированными контейнерами [9]. Практически вся функциональность LXC представления известными механизмами ядра cgroups и namespaces:

cgroups (Control Groups) — позволяет ограничить аппаратные ресурсы некоторого набора процессов. Под аппаратными ресурсами подразумеваются: процессорное время, память, дисковая и сетевая подсистемы. Набор или группа процессов могут быть определены различными критериями. Например, это может быть целая иерархия процессов, получающая все лимиты родительского процесса. Кроме этого, возможен подсчет расходуемых группой ресурсов, заморозка (freezing) групп, создание контрольных точке (checkpointing) и их перезагрузка. Для управления этим полезным механизмом существует специальная библиотека libcgroups, в состав которой входят такие утилиты, как cgcreate, cgexec и некоторые другие.

namespaces — пространства имен. Это механизм ядра, который позволяет изолировать процессы друг от друга. Изоляция может быть выполнена в шести контекстах (пространствах имен):

- Mount — предоставляет процессам собственную иерархию файловой системы и изолирует ее от других таких же иерархий, по аналогии с chroot;
- PID — изолирует идентификаторы процессов (PID) одного пространства имен от процессов с такими же идентификаторами другого пространства;
- Network — предоставляет отдельным процессам логически изолированный от других стек сетевых протоколов, сетевой интерфейс, IP-адрес, таблицу маршрутизации, ARP и прочие реквизиты;
- IPC — обеспечивает разделяемую память и взаимодействие между процессами;
- UTS — изоляция идентификаторов узла, такого как имя хоста (hostname) и домена (domainname);
- User — позволяет иметь один и тот же набор пользователей и групп в рамках разных пространств имен, в каждом контейнере могут быть свой root и любые другие пользователи и группы.

Одно из главных преимуществ LXC — это присутствие его базовых блоков (cgroups и namespaces) во всех современных ядрах Linux. Это означает, что нет необходимости что-то компилировать или использовать стороннее ядро, как в

случае с OpenVZ. Единственное, что необходимо установить, это пакет утилит управления (vzctl, Docker, libvirt, systemd).

К системам управления можно отнести Docker.

Docker — программное обеспечение для автоматизации развертывания и управления приложениями в среде виртуализации на уровне операционной системы, например LXC. Docker позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесен на любой Linux-системе с поддержкой cgroups в ядре, а также предоставляет среду по управлению контейнерами.

Для экономии дискового пространства проект использует файловую систему Aufs с поддержкой каскадно-объединенного монтирования: контейнеры используют образ базовой операционной системы, а изменения записываются в отдельную область. Также поддерживается размещение контейнеров в файловой системе Btrfs с включенным режимом копирования при записи. В состав программных средств входит демон — сервер контейнеров, клиентские средства, позволяющие из интерфейса командной строки управлять образами и контейнерами, а также API, позволяющий в стиле REST управлять контейнерами программно. Демон обеспечивает полную изоляцию запускаемых на узле контейнеров на уровне файловой системы (у каждого контейнера собственная файловая система), на уровне процессов (процессы имеют доступ только к собственной файловой системе контейнера, а ресурсы разделены средствами LXC), на уровне сети (каждый контейнер имеет доступ только к привязанному к нему сетевому пространству имен и соответствующим виртуальным сетевым интерфейсам).

Набор клиентских средств позволяет запускать процессы в новых контейнерах, останавливать и запускать контейнеры, приостанавливать и возобновлять процессы в контейнерах. Серия команд позволяет осуществлять мониторинг запущенных процессов (по аналогии с ps, top). Новые образы возможно создавать из специального сценарного файла (dockerfile), возможно записать все изменения, сделанные в контейнере в новый образ. Все команды могут работать как с docker-демоном локальной системы, так и с любым сервером docker, доступным по сети. Кроме того, в интерфейсе командной строки встроены возможности по взаимодей-

ствию с публичным репозиторием Docker Hub, в котором размещены предварительно собранные образы контейнеров, образы можно скачивать в локальную систему, возможно также отправить локально собранные образы в Docker Hub.

# 3 СИСТЕМНЫЙ АНАЛИЗ ВИРТУАЛЬНОЙ ИНФРАСТРУКТУРЫ

На данный момент при анализе и синтезе сложных программных и аппаратных систем все чаще используется системный подход. Важным моментом для системного подхода является определение структуры системы — совокупности связей между элементами системы, отражающих их взаимодействие. Совокупность элементов и связей между ними позволяет судить о структуре системы.

Принципы системного анализа — это некоторые положения общего характера, являющиеся обобщением опыта работы человека со сложными системами. Общепринятых формулировок на настоящее время нет, но не все формулировки так или иначе описывают одни и те же понятия. Пренебрежение принципами при проектировании любой нетривиальной технической системы, непременно приводит к потерям того или иного характера, от увеличения затрат в процессе проектирования до снижения качества и эффективности конечного продукта.

Системный анализ выполнен в соответствии с [10].

## 3.1 Принцип конечной цели

Принцип конечной цели — это абсолютный приоритет конечной цели, он имеет несколько правил:

- Для проведения системного анализа необходимо, в первую очередь, сформировать цель исследования, так как не полностью определенные цели влекут за собой неверные выводы;
- Анализ следует вести на базе уяснения основной цели, что позволит определить ее существенные свойства показателей качества и критериев оценки;
- При синтезе систем любая попытка изменения или совершенствования должна оцениваться относительно конечной цели;
- Цель функционирования искусственной системы задается, как правило, системой, в которой исследуемая система является составной частью.

При использовании данного принципа, разрабатываемая виртуальная инфраструктура будет рассматриваться в виде «черного ящика», функционирование которого описывается формулой (3.1):

$$Y=F(X, Z) \tag{3.1}$$

где Y — выходной вектор системы, который функционально зависит от входного вектора X и вектора внутреннего состояния системы Z (рис. 3.1). Входными данными (вектор X) будут являться запросы пользователей на обработку инцидентов и запросов на обслуживание. Внутреннее состояние системы (вектор Z) представляет собой создание услуг по требованию клиента. Выходными данными (вектор Y) будут являться предоставление доступа к услугам клиентам.

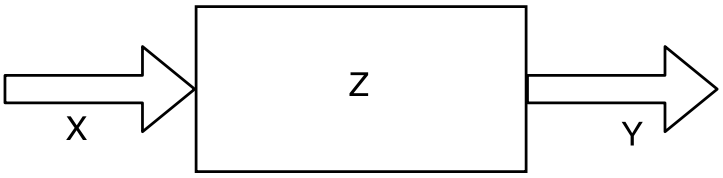


Рисунок 3.1 – Проектируемая система в виде черного ящика

В соответствии с данным принципом должна быть четко сформулирована конечная цель — назначение проектируемой системы и сформирован список функций, которые должна выполнять система. Кроме того, необходимо определить перечень входных воздействий, на которые реагирует система.

Цель проектирования — разработка виртуальной инфраструктуры для реализации облачных услуг. Список функций проектируемой системы:

- Ф1 — прием и регистрация обращений пользователей;
- Ф2 — идентификация и обработка инцидентов и запросов на обслуживание;
- Ф3 — создание, смена, обновление и удаление услуг по требованию;
- Ф4 — предоставление доступа к услугам;
- Ф5 — мониторинг состояния инфраструктуры.

Перечень входных воздействий на систему:

- Внесение изменений данных о заявке;
- Мониторинг хода решения.

## 3.2 Принцип единства

Принцип единства — это совместное рассмотрение системы как целого и как совокупности частей. Принцип ориентирован на декомпозицию с сохранением целостных представлений о системе.

На основании функций проектируемой системы, представленных выше, в ней можно выделить следующие подсистемы:

1. Подсистема взаимодействия с пользователем;
2. Подсистема управления услугами;
3. Подсистема управления инфраструктурой;
4. Подсистема защиты и обеспечения целостности данных.

На рис. 3.2 представлена схема взаимодействия между подсистемами.

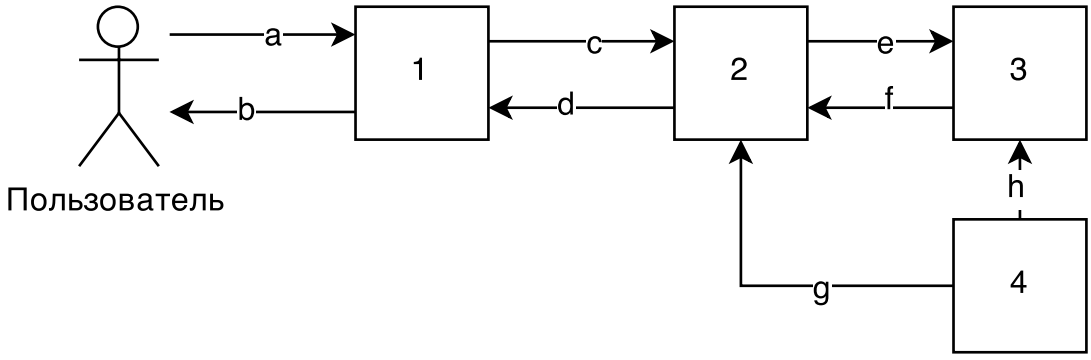


Рисунок 3.2 – Взаимодействие между подсистемами и их связь с окружающей средой

Обозначения, приведенные на рис. 3.2 требуют пояснения:

a — информация, предоставляемая пользователем, передается на сервер;

b — выходная информация (результат выполнения);

c — проверка корректности переданных данных;

d — в случае неправильно введенных данных, возвращается управление к подсистеме взаимодействия с пользователем;

e — создание услуги;

f — возврат информации о созданной услуге;

g — обеспечение целостности информации об услуге;

h — обеспечение целостности данных пользователя.

На рис. 3.3 представлена структура проектируемой системы в виде ориентированного графа.

Приняты те же обозначения, что приведены выше.

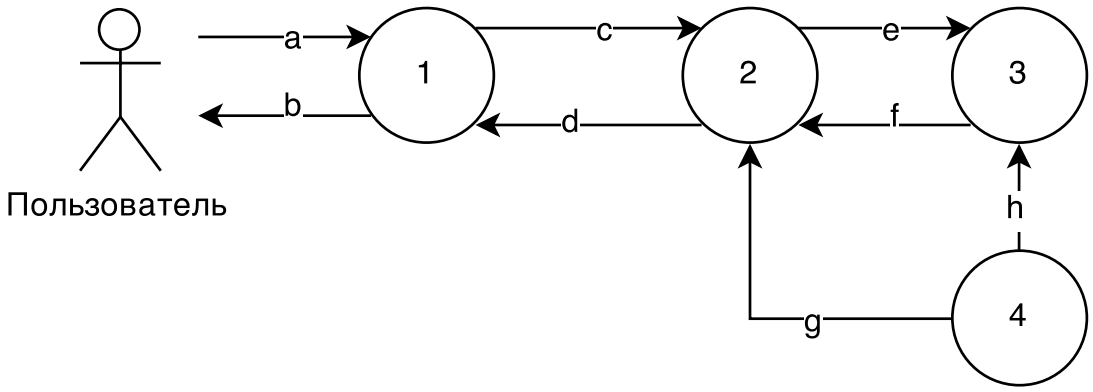


Рисунок 3.3 – Структура проектируемой системы, представленная в виде ориентированного графа

3.3 Определение взаимосвязей между подсистемами на основе принципа связности

Рассмотрение любой части совместно с ее окружением подразумевает проведение процедуры выявления связей между элементами системы и выявление связей со средой. В соответствии с этим принципом систему в первую очередь следует рассматривать как часть другой системы, называемой суперсистемой или старшей системой.

Подсистема взаимодействия с пользователем представлена на рис. 3.4.


Рисунок 3.4 – Внутренние и внешние связи подсистемы взаимодействия с пользователем

Подсистема управления услугами представлена на рис. 3.5.

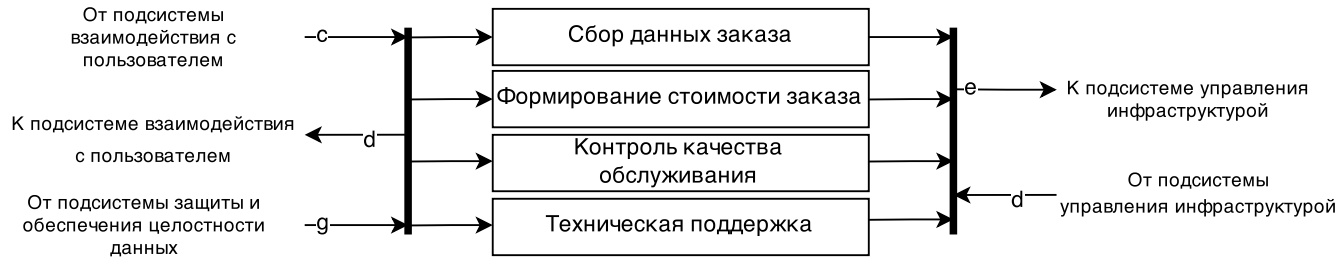


Рисунок 3.5 – Внутренние и внешние связи подсистемы управления услугами

Подсистема управления инфраструктурой представлена на рис. 3.6.


Рисунок 3.6 – Внутренние и внешние связи подсистемы управления инфраструктурой

Подсистема защиты и обеспечения целостности данных представлена на рис. 3.7.

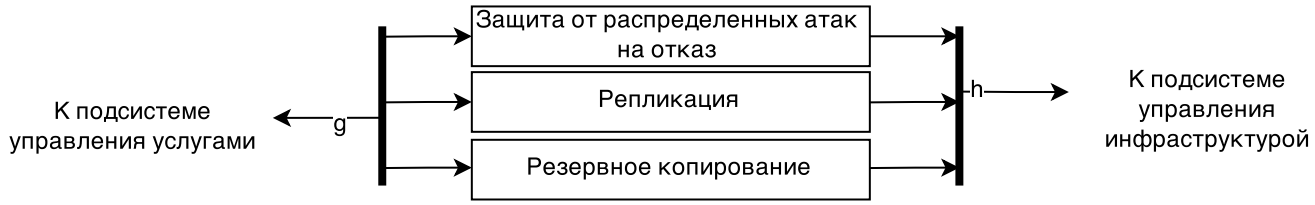


Рисунок 3.7 – Внутренние и внешние связи подсистемы защиты и обеспечения целостности данных

## 3.4 Принцип модульности

Полезно выделение модулей в системе и рассмотрение ее как совокупности модулей. Принцип указывает на возможность вместо части системы исследовать совокупность ее входных и выходных воздействий (абстрагирование от излишней детализации).

Принцип модульности для разрабатываемой системы поясняется с помощью рис. 3.8, описывающего разбиение на модули системы взаимодействия с пользователем.


Рисунок 3.8 – Принцип модульности на примере подсистемы взаимодействия с пользователем

Излишняя детализация не требуется, поэтому остальные системы на модули принято решение не разбивать.

## 3.5 Принцип функциональности

Принцип утверждает, что любая структура тесно связана с функцией системы и ее частей. В случае придания системе новых функций полезно пересматривать ее структуру, а не пытаться втиснуть новую функцию в старую схему. Поскольку выполняемые функции составляют процессы, то целесообразно рассматривать отдельно процессы, функции, структуры. В свою очередь, процессы сводятся к анализу потоков различных видов:

- Материальный поток;
- Поток энергии;
- Поток информации;
- Смена состояний.

С этой точки зрения структура есть множество ограничений на потоки в пространстве и времени.

Функции подсистем приведены в п. 3.1.

Матрица инциденций функций системы и функций назначения подсистем приведена в табл. 3.1.

Таблица 3.1 – Матрица инциденций

| Функции | Подсистемы | | | | Виртуальная инфраструктура |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | |
| Ф1 | + | | | | + |
| Ф2 | + | | | | + |
| Ф3 | | + | | | + |
| Ф4 | | + | | | + |
| Ф5 | | | + | + | + |

В матрице инциденций знаком «+» обозначены функции, которые реализуются для каждой из подсистем.

Детализация функций подсистемы на примере подсистемы взаимодействия с пользователем:

1. Регистрация и авторизация пользователя в платежной системе;
2. Обеспечение перечня услуг;
3. Обеспечение возможных способов оплаты услуг;
4. Возможность уточнения или изменения услуги.

Входными данными для подсистемы является информация о пользователе, а выходными — подтвержденный заказ услуг.

### 3.6 Принцип иерархии

Полезно введение иерархии частей и их ранжирование, что упрощает разработку системы и устанавливает порядок рассмотрения частей.

Выполнение принципа иерархичности для разрабатываемой системы на примере подсистемы защиты и обеспечения целостности данных проиллюстрировано на рис. 3.9.

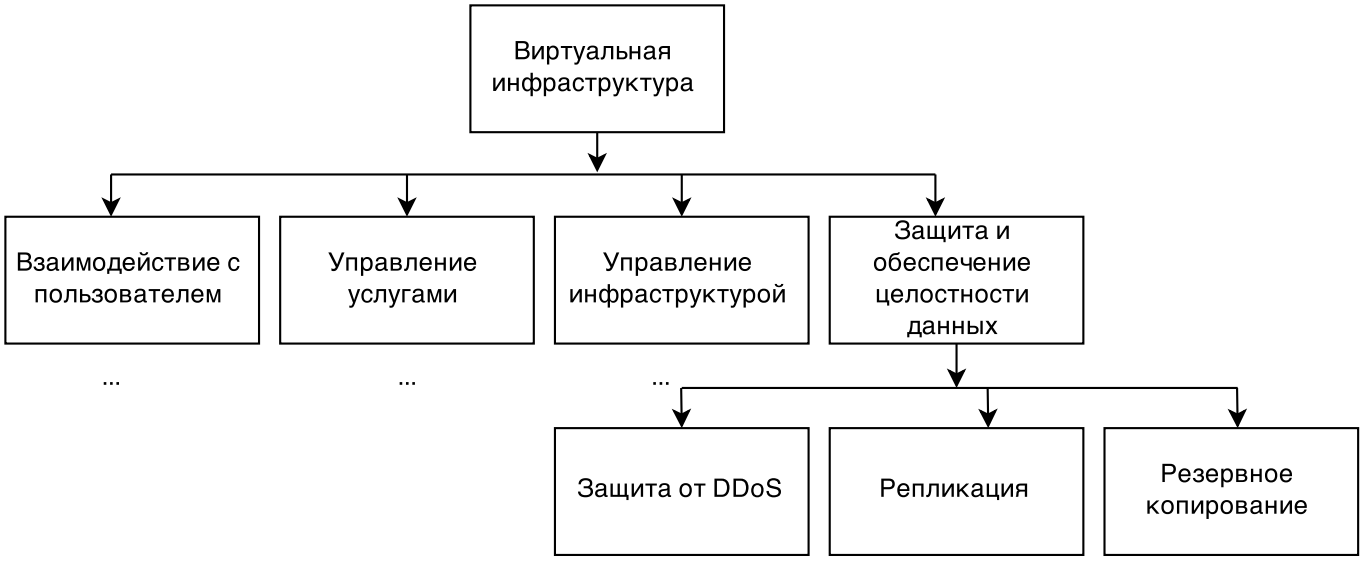


Рисунок 3.9 – Принцип иерархии на примере подсистемы защиты и обеспечения целостности данных

## 3.7 Принцип сочетания централизации и децентрализации

Степень централизации должна быть минимальной, обеспечивающей выполнение поставленной цели. Соотношение централизации и децентрализации определяется уровнями, на которых вырабатываются и принимаются управленческие решения.

Недостаток децентрализованного управления — увеличение времени адаптации системы. Он существенно влияет на функционирование системы в быстро меняющихся средах. То, что в централизованных системах можно сделать за короткое время, в децентрализованной системе будет осуществляться весьма медленно. Данный недостаток нивелируется налаживанием горизонтальных связей.

Недостатком централизованного управления является сложность управления из-за огромного потока информации, подлежащей переработке в старшей системе управления. Поэтому в сложной системе обычно присутствуют два уровня управления. В медленно меняющейся обстановке децентрализованная часть системы успешно справляется с адаптацией поведения системы к среде и с достижением глобальной цели системы за счет оперативного управления, а при резких изменениях среды осуществляется централизованное управление по переводу системы в новое состояние.

Например, можно выполнить декомпозицию подсистемы взаимодействия с пользователем таким образом:

1. {Подсистема работы с клиентом};
2. {Подсистема работы клиентам с услугой};
3. {Подсистема учета финансов}.

Такое разбиение позволит реализовать полученные подмножества в виде отдельных модулей.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Каталог программных продуктов семейства Oracle [Электронный ресурс]. – Электрон. текстовые данные (11126315 bytes) – Режим доступа: http://oracle.ocs.ru/files/catalog_Oracle_Database_12C.pdf

2. Tier: уровни надежности ЦОД и что из этого следует [Электронный ресурс]. – Электрон. текстовые данные (38916 bytes) – Режим доступа: http://www.aboutdc.ru/page/390.php

3. Виртуальный Linux. Обзор методов виртуализации, архитектур и реализаций [Электронный ресурс]. – Электрон. текстовые данные (109051 bytes) – Режим доступа: http://www.ibm.com/developerworks/ru/library/l-linuxvirt/index.html

4. Руководство по созданию и управлению контейнерами на базе OpenVZ [Электронный ресурс]. – Электрон. текстовые данные (55237 bytes) – Режим доступа: https://github.com/Amet13/openvz-tutorial/blob/master/main.pdf

5. Эмуляция систем с помощью QEMU [Электронный ресурс]. – Электрон. текстовые данные (84008 bytes) – Режим доступа: http://www.ibm.com/developerworks/ru/library/l-qemu/

6. Гипервизоры, виртуализация и облако: Анализ гипервизора KVM [Электронный ресурс]. – Электрон. текстовые данные (79196 bytes) – Режим доступа: http://www.ibm.com/developerworks/ru/library/cl-hypervisorcompare-kvm/

7. Руководство пользователя Xen v3.0 [Электронный ресурс]. – Электрон. текстовые данные (272945 bytes) – Режим доступа: http://xgu.ru/xen/manual/

8. Контейнеризация на Linux в деталях – LXC и OpenVZ Часть 2 [Электронный ресурс]. – Электрон. текстовые данные (186899 bytes) – Режим доступа: http://habrahabr.ru/company/FastVPS/blog/209084/

9. Виртуализация на уровне ОС: теория и практика LXC [Электронный ресурс]. – Электрон. текстовые данные (118259 bytes) – Режим доступа: http://creativeyp.com/568-virtualizaciya-na-urovne-os-teoriya-i-praktika-lxc.html

10. Сергеев Г.Г., Скатков А.В., Мащенко Е.Н. Методические указания «Процедура системного анализа при проектировании программных систем» для студентов-дипломников дневной и заочной формы обучения специальности 7.091501 / Сост.: Сергеев Г.Г., Скатков А.В., Мащенко Е.Н. – Севастополь: Изд-во СевНТУ, 2005. – 32с.

11. Лукъянчук А.Г., Антонова Г.З., Заика Е.В. Положение об организации дипломного проектирования в СевНТУ / Лукъянчук А.Г., Антонова Г.З., Заика Е.В. – Севастополь: Изд-во СевНТУ, 2012. – 30 с.

12. Балакирева И.А., Скатков А.В. Методические указания для выполнения типовой выпускной квалификационной работы бакалавра по тематике анализа эффективности компьютерных систем обработки данных для студентов направления 6.050102 – «Компьютерная инженерия» дневной и заочной форм обучения / Сост.: И.А. Балакирева, А.В. Скатков – Севастополь: Изд-во СевНТУ, 2012. – 32 с.

13. ГОСТ 19.701-90, ЕСПД «Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения». Дата введения: 01.01.1992. – М: Изд-во стандартов, 1991. – 26 с.

14. ДСТУ 3008-95 «Документация. Отчеты в сфере науки и техники. Структура и правила оформления». Дата введения: 01.01.1996. – К.: Госстандарт Украины, 1995. – 37 с.

15. Mark Furman. OpenVZ Essentials / Mark Furman – Великобритания: Изд-во Packt Publishing Ltd., 2014. – 110 с.

16. Михаил Михеев. Администрирование VMware vSphere 5 / Михаил Михеев – Москва: Изд-во ДМК Пресс, 2012. – 504 с.

17. Performance Evaluation of Virtualization Technologies for Server Consolidation [Электронный ресурс]. – Электрон. текстовые данные (428865 bytes) – Режим доступа: http://www.hpl.hp.com/techreports/2007/HPL-2007-59R1.pdf

18. Виртуальная реальность по-русски: Осваиваем виртуализацию уровня ОС на примере OpenVZ [Электронный ресурс]. – Электрон. текстовые данные (131599 bytes) – Режим доступа: https://xakep.ru/2011/08/27/56244/

19. Виртуализация: новый подход к построению IT-инфраструктуры [Электронный ресурс]. – Электрон. текстовые данные (160658 bytes) – Режим доступа: http://www.ixbt.com/cm/virtualization.shtml

20. Технологии виртуализации: вчера, сегодня, завтра [Электронный ресурс]. – Электрон. текстовые данные (111519 bytes) – Режим доступа: http://citforum.ru/operating_systems/virtualization/

21. Руководство по созданию виртуальных выделенных серверов на базе Virtuozzo [Электронный ресурс]. – Электрон. текстовые данные (839355 bytes) – Режим доступа: http://www.opennet.ru/docs/RUS/virtuozzo/virtuozzo-linux.html.gz

22. Гипервизоры, виртуализация и облако: О гипервизорах, виртуализации систем и о том, как это работает в облачной среде [Электронный ресурс]. – Электрон. текстовые данные (87662 bytes) – Режим доступа: http://www.ibm.com/developerworks/ru/library/cl-hypervisorcompare/

23. Обзор достижений контейнерной изоляции за последние два года [Электронный ресурс]. – Электрон. текстовые данные (222074 bytes) – Режим доступа: http://www.opennet.ru/opennews/art.shtml?num=40126

24. Performance – OpenVZ Linux Containers Wiki [Электронный ресурс]. – Электрон. текстовые данные (37869 bytes) – Режим доступа: http://openvz.org/Performance