Министерство образования и молодежной политики Свердловской области

государственное автономное профессиональное

образовательное учреждение Свердловской области

«Уральский радиотехнический колледж им. А.С. Попова»

ДОПУСТИТЬ К ЗАЩИТЕ

Зав. отделением

­\_\_\_\_\_\_\_\_\_\_\_О. В. Алферьева

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

РАЗРАБОТКА МАКЕТА IT-ИНФРАСТРУКТУРЫ

СОВРЕМЕННОЙ КОМПАНИИ

Пояснительная записка к дипломному проекту

РК 09.02.06 401 10 ПЗ

Рецензент Руководитель

Д. С. Апататьев Д. С. Апататьев

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2025 г. «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2025 г.

Консультант Разработчик

Д. С. Апататьев Т. И. Куваев

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2025 г. «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2025 г.

Нормоконтролер

Д. С. Апататьев

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2025 г.

2025

**Содержание**

[**Введение** 4](#_Toc196405350)

[**1 Теоретическая часть** 5](#_Toc196405351)

[1.1 Значение IT-инфраструктуры в современном мире 5](#_Toc196405352)

[1.2 Компоненты и составляющие IT-инфраструктуры 6](#_Toc196405353)

[1.3 Основные принципы проектирования 10](#_Toc196405354)

[1.4 Безопасность инфраструктуры 21](#_Toc196405355)

[1.5 Требования для построения инфраструктуры 21](#_Toc196405356)

[1.6 Выбор лабораторной среды для создания макета 21](#_Toc196405357)

[**2 Практическая часть** 22](#_Toc196405358)

[2.1 Определение ПО, сервисов и производителей, подходящих под требования 22](#_Toc196405359)

[2.2 Планирование и подготовка к реализации будущей инфраструктуры 22](#_Toc196405360)

[2.3 Настройка сетевого оборудования 22](#_Toc196405361)

[2.4 Настройка серверных ОС 22](#_Toc196405362)

[2.5 Развертывание сервисов 22](#_Toc196405363)

[2.6 Настройка пользовательских ОС 22](#_Toc196405364)

[2.7 Проверка соответствия представленным и законодательным требованиям 22](#_Toc196405365)

[3 Охрана труда и производственная санитария 23](#_Toc196405366)

[**Заключение** 24](#_Toc196405367)

[**Перечень сокращений и условных обозначений** 25](#_Toc196405368)

[**Список используемых источников** 26](#_Toc196405369)

[Приложение А 27](#_Toc196405370)

# **Введение**

Основные параметры документа согласно ГОСТ Р 2.105-2019:

* + текст шрифт Times New Roman, 12пт;
  + межстрочный интервал 1.5;
  + поля: слева 25 мм, справа 10 мм, сверху 15 мм, снизу 30 мм;
  + абзацный отступ 15 мм;
  + названия разделов выделять увеличенным размером шрифта (14пт), полужирным шрифтом, без точки в конце;
  + расстояние от заголовка раздела и подраздела до текста и от текста до заголовка должно быть равно 3 интервалам.

# **1 Теоретическая часть**

# 1.1 Значение IT-инфраструктуры в современном мире

ИТ-инфраструктура — это взаимосвязанные компоненты, из которых состоит информационная структура организации. В состав такой инфраструктуры входит техническое оборудование для выхода в интернет, выполнения рабочих задач и комфортной работы в офисе.

По мере распространения интернета и его доступности стали создаваться единые цифровые системы. Они необходимы для бесперебойной работы организации на всех уровнях.

Главная задача ИТ-инфраструктуры — обеспечить стабильную и безопасную работу организации, а значит, она решает множество актуальных вопросов. В первую очередь, важнейшим аспектом является защита информации. Сегодня, когда данные становятся наиболее ценным активом, грамотное управление ими критично важно. Организации разрабатывают системы, которые позволяют защищать конфиденциальную информацию от кибер-атак и утечек.

Еще одна важнейшая задача ИТ-инфраструктуры — это обеспечение отказоустойчивости. Никакой бизнес не может позволить себе длительные простои, и здесь на помощь приходят резервирование и автоматические решения для восстановления. Когда случаются сбои, возможность быстро вернуть работу системы в нормальный режим становится критически важной.

Оптимизация рабочих процессов — это значительное преимущество, которое дает современная ИТ-инфраструктура. Интеграция различных программных решений позволяет работникам получать доступ к нужной информации в любое время, что значительно облегчает выполнение задач. Поработав в такой системе, сотрудники быстро замечают, как снижается время, затрачиваемое на рутинные операции, и как повышается общая продуктивность.

По мере роста бизнеса возникает необходимость в гибкости и способности адаптироваться. ИТ-инфраструктура предоставляет возможность масштабировать ресурсы в зависимости от текущих потребностей. Компании могут увеличивать мощность своих систем, чтобы справляться с пиком загрузки, и, наоборот, снижать её в спокойные времена, что снижает затраты организации.

Кроме того, сегодня все чаще обсуждается такая тема, как мобильность и удаленная работа. ИТ-инфраструктура должна отвечать требованиям этого нового времени, позволяя сотрудникам иметь доступ к данным и приложениям независимо от места их нахождения.

Наконец, ИТ-инфраструктура не стоит на месте. Она требует постоянного обновления и адаптации к новым технологиям. Внедрение облачных сервисов, инструментов для анализа данных и средств, которые помогают обеспечить безопасность и конфиденциальность информации становится необходимостью.

# 1.2 Компоненты и составляющие IT-инфраструктуры

Любая ИТ-инфраструктура — это комплекс взаимосвязанных компонентов, где каждый элемент выполняет строго определенную роли. Без их совместной работы невозможны ни обработка данных, ни коммуникация, ни даже базовое функционирование организации. Условно компоненты делятся на несколько типов:

* + аппаратные;
  + программные;
  + сетевые.

Аппаратные компоненты можно подразделить на несколько подтипов:

* + системы хранения данных — обязательный элемент относительно продвинутых организаций и выше. От обычного сервера с жесткими дисками этот элемент отличается наличием специализированных решений, по типу NAS для файлового доступа или SAN, где данные передаются блоками через iSCSI/Fibre Channel. RAID-массивы в хранении данных являются абсолютным стандартом. Если один диск «умрет», данные не потеряются;
  + автоматизированное рабочее место — это всё, чем пользуются сотрудники: офисные ПК, инженерные станции, принтеры, телефоны. Главное правило — соответствие задачам. Для бухгалтерии хватит слабого ПК с офисным пакетом ПО, для специфических задач с обработкой графики — рабочая станция с GPU невероятной мощности;
  + сервер. В любой организации подобное оборудование можно грубо назвать «моторами» инфраструктуры. Бывают стоечные, блейд-сервера или Tower-сервера (по виду – обычный ПК). Их задача — не выключаться, поэтому всё идет с запасом: блоки питания, диски с горячей заменой и много других функций.

Аппаратные компоненты в инфраструктуре лишь некоторая часть. Поверх них работают программные компоненты, которые выполняют большую часть функций, необходимых организации.

Операционная система — фундамент, который превращает набор плат в компьютере в рабочую среду. Без неё сервера, ПК и смартфоны останутся бесполезными устройствами. ОС управляет ресурсами, распределяет память между программами, выделяет процессорное время, обеспечивает доступ к дискам и сети. Даже встроенные устройства (роутеры, например) имеют свою операционную систему — обычно сильно урезанные дистрибутивы Linux. Ключевая задача любой ОС — быть посредником, то есть принимать команды от программ, преобразовывать их в инструкции для процессора и гарантировать, что сбой одного приложения не сможет помешать остальным.

Системное программное обеспечение — это инструментарий для управления инфраструктурой. Оно не решает бизнес-задачи напрямую, но обеспечивает стабильность и контроль над аппаратными ресурсами. Если брать в пример гипервизоры, то увидим, что позволяют запускать на одном сервере десятки виртуальных машин, изолируя их друг от друга. Утилиты резервного копирования создают копии данных даже во время работы систем. Инструменты мониторинга следят за нагрузкой на процессор, диски и сеть, отправляя предупреждения при критических значениях. Отдельная категория — драйвера. Микропрограммы, которые «объясняют» аппаратной части, как взаимодействовать с ОС. Без них видеокарта не будет выводить изображение. Системное ПО работает в фоне, но его отсутствие парализует инфраструктуру, примерно, как отключение электричества на предприятии.

Прикладное программное обеспечение — это программы, которые непосредственно решают задачи бизнеса или пользователя. Например, офисные пакеты программ (Р7, Мой Офис) позволяет бухгалтеру формировать отчеты, CAD-программы помогают инженеру проектировать детали, а Paint дает возможность дизайнеру редактировать изображения. В корпоративной среде востребованы CRM-системы (Bitrix24, Salesforce) для учета клиентов, ERP-платформы (1С, SAP) для управления финансами и логистикой, ну и конечно, специализированные решения вроде САПР для машиностроения или медицинских информационных систем для больниц. Даже веб-браузер — часть прикладного ПО: без него сотрудник не откроет корпоративный портал. Эти программы зависят от системного ПО и ОС, но именно они создают ценность для бизнеса.

Ограничивается программное обеспечение последним типом – промежуточным ПО. Промежуточное программное обеспечение — это что-то вроде клея, который соединяет приложения друг с другом и с инфраструктурой, но при этом может выступать и в роли прикладного ПО, конкретное определение зависит от контекста эксплуатации. Используется подобное как правило в серверной части инфраструктуры, где имеется какое-то нагромождение сервисов. В качестве примера можно привести СУБД (PostgreSQL, MySQL и прочие), которая хранит данные для интернет-магазина, веб-сервер (Nginx, Apache) принимает запросы от покупателей и может передавать какую-то информацию дальше по системе. Без промежуточного ПО приложения не смогут взаимодействовать.

Программные компоненты обеспечивают логику работы инфраструктуры, но без сети, в большинстве случаев – мало полезны. Сервер не передаст данные в СУБД на другом сервере, сотрудник не отправит отчет коллеге. Сетевые компоненты — это что-то вроде кровеносной системы ИТ-инфраструктуры, которая соединяет устройства, маршрутизирует трафик.

Физический уровень — основа любой сети, по которой движутся данные: кабели, разъемы, сетевые карты и оборудование, преобразующее сигналы в электрические импульсы или световые сигналы. Самый распространенный вариант — Ethernet. Витая пара CAT5E дешевая, проста в монтаже и подходят для офисов, где расстояние между устройствами не превышает 100 метров. Для больших дистанций или высокой скорости используют витую пару с категорией выше или оптоволокно — световые сигналы в таких кабелях почти не затухают, всегда используется в глобальных магистралях.

Канальный уровень обеспечивает передачу данных между устройствами в рамках локальной сети, организуя их в структурированные блоки — кадры. На этом уровне к исходной информации добавляется служебная информация, MAC-адреса отправителя и получателя, контрольные суммы для проверки целостности и управляющие флаги. Без этой структуризации данные не могут быть корректно интерпретированы принимающей стороной.

Основная задача уровня — точная доставка кадров, где эту задачу решает MAC-адресация. Например, при отправке данных с компьютера на принтер коммутатор анализирует MAC-адрес получателя и перенаправляет фрейм только в соответствующий порт. Это исключает избыточную рассылку трафика, характерную для устаревших сетевых концентраторов.

Канальный уровень также отвечает за обнаружение и коррекцию ошибок передачи. Контрольная сумма (CRC), включенная в каждый фрейм выявляет искажения данных из-за помехам или сбоев. При обнаружении ошибки фрейм отклоняется, что инициирует повторную передачу со стороны отправителя.

На канальном уровне реализованы:

* + Ethernet (IEEE 802.3) — стандарт для проводных сетей, использующий MAC-адресацию и фреймы до 1500 байт;
  + Wi-Fi (IEEE 802.11) — беспроводная передача данных с механизмами минимизации помех;
  + VLAN (IEEE 802.1Q) — логическое разделение физической сети на изолированные сегменты (например, для отделов компании);
  + PPPoE — инкапсуляция данных для широкополосных подключений;
  + LLDP — обмен служебной информацией между сетевыми устройствами.

Сетевой уровень обеспечивает передачу данных между устройствами, находящимися в разных сетях, определяя оптимальные маршруты для их доставки. Его ключевая задача — преодолеть ограничения локальной коммуникации, организовав взаимодействие в масштабах глобальных сетей, включая глобальные сети. На этом уровне кадры инкапсулируются в пакеты, содержащие IP-адреса отправителя и получателя и прочую служебную информацию для управления маршрутизации.

Основной функцией уровня является маршрутизация — процесс выбора пути для пакетов через маршрутизаторы. Для этого нужны таблицы маршрутизации, где указаны направления к сетевым сегментам. Например, пакет, отправленный из офиса в Москве в филиал в Берлине, проходит через несколько маршрутизаторов, каждый из которых анализирует IP-адрес назначения и перенаправляет его дальше по оптимальному маршруту.

Сетевой уровень также отвечает за:

* + фрагментацию пакетов — разделение данных на части, если их размер превышает MTU (максимальный размер блока, поддерживаемый канальным уровнем);
  + логическую адресацию — использование IP-адресов (IPv4/IPv6) вместо MAC-адресов, что позволяет работать в гетерогенных сетях (Ethernet, Wi-Fi, спутниковая связь);
  + обработку ошибок — протоколы вроде ICMP отправляют уведомления о проблемах доставки (например, если узел недоступен).

На сетевом уровне работают:

* + IP (Internet Protocol) — основа интернета, обеспечивающая адресацию и маршрутизацию;
  + ICMP (Internet Control Message Protocol) — диагностика и уведомления об ошибках (например, ping);
  + NAT (Network Address Translation) — преобразование приватных IP-адресов в публичные для выхода в интернет;
  + VPN (Virtual Private Network) — создание защищенных туннелей между сетями;
  + OSPF, BGP — протоколы динамической маршрутизации.

# 1.3 Основные принципы проектирования

Проектирование ИТ-инфраструктуры – комплексный процесс, направленный на создание технологической основы для поддержки бизнес-процессов организации. Современные принципы проектирования информационных систем учитывают не только текущие потребности, но и обеспечивают возможность масштабирования и отказоустойчивости.

Современная инфраструктура должна проектироваться с учетом концепции "пирамиды надежности", где каждый уровень (аппаратный, программный, сетевой) обеспечивает резервирование и отказоустойчивость. Рассматривая этот момент чуть подробнее, стоит отметить – полная отказоустойчивость в большинстве случаев требует достаточно больших затрат, ведь сделать резерв всего иногда попросту невозможно, поэтому иногда приходится где-то сэкономить, опираясь на надежность имеющихся аппаратных и программных средств.

Выделяют несколько основных типов, которые выбираются в зависимости от задач и внешних требований организации:

* + традиционная;
  + облачная;
  + гибридная.

Традиционная ИТ-инфраструктура, основанная на локальных дата-центрах и физических серверах, на протяжении десятилетий оставалась преобладающей моделью для большинства предприятий. Несмотря на растущие тенденции к облачным решениям и виртуализации, эта модель по-прежнему сохраняет свою актуальность и востребованность в ряде секторов.

В основе традиционной инфраструктуры лежит концепция полного контроля организации над собственными информационными ресурсами. Выделенные серверные помещения, оснащенные надежными системами электропитания, охлаждения и физической безопасности, выступают фундаментом этой архитектуры. Оборудование приобретается в собственность компании и размещается на ее территории. За счет этого обеспечивается прямое управление аппаратными мощностями и данными. Данный подход обусловлен жесткими требованиями к безопасности информации, которые характерны, например, для сфер государственного управления, финансов, энергетики и прочих отраслей, которые могут содержать в себе объекты КИИ.

При проектировании традиционной инфраструктуры ключевыми аспектами являются:

* + обеспечение избыточности и отказоустойчивости с помощью кластерных решений, резервного оборудования и систем бесперебойного питания;
  + иерархическая структура сетевой инфраструктуры с ядром, распределительными коммутаторами и точками подключения клиентских устройств;
  + интеграция с унаследованными системами и обеспечение совместимости с существующими ИТ-ресурсами.

Классическая сетевая инфраструктура опирается на жесткую иерархию физических компонентов с жестко закрепленными функциями для каждого элемента. Основу формируют проводные соединения, создающие фиксированную топологию с централизованными управляющими узлами. Коммутационные устройства, размещенные в защищенных серверных помещениях, выполняют роль системного каркаса, обеспечивая взаимодействие между серверами, рабочими станциями и периферийными устройствами. Архитектура подразумевает строгое разделение на зоны: внутренние сегменты сети изолированы от внешних подключений физическими барьерами, а межзональный доступ регулируется через централизованные контрольные точки.

В отличие от гибких современных решений, управление трафиком здесь базируется на заранее заданных правилах, жестко прописанных в конфигурациях оборудования. Данные передаются по статическим маршрутам, оптимизированным для стандартных нагрузочных сценариев. Подобная схема гарантирует стабильность работы, однако плохо адаптируется к динамичным изменениям — резким скачкам числа подключений или неравномерному распределению запросов.

Система безопасности реализуется через многоуровневую изоляцию. Внешний периметр контролируется аппаратными фильтрами, блокирующими несанкционированные подключения на уровне сетевых пакетов. Внутренние сегменты дополнительно разделяются по функциональному принципу для ограничения горизонтального перемещения угроз. Серверы баз данных, к примеру, могут располагаться в отдельном физическом контуре с ограниченным доступом для определенных групп пользователей. Централизованная реализация аутентификации упрощает администрирование, одновременно создавая риски единых точек отказа.

Масштабирование инфраструктуры связано с физическим расширением — установкой новых коммутаторов, прокладкой кабельных трасс, настройкой дополнительных портов. Подобные операции усложняют топологию и повышают эксплуатационные расходы. Интеграция новых узлов требует проверки совместимости оборудования, обновления таблиц маршрутизации и тестирования на предмет возникновения узких мест. Подключение облачных сервисов осуществляется через выделенные шлюзы, что увеличивает задержки передачи данных.

Надежность сети обеспечивается дублированием критических элементов: резервных линий связи, избыточных источников питания и зеркальных устройств. Несмотря на резервирование, часть мощностей часто остается невостребованной в штатных режимах, а перенастройка под меняющиеся условия требует остановки работы для физического переподключения компонентов.

Облачная инфраструктура представляет собой современный подход к организации вычислительных ресурсов, где все компоненты – серверы, хранилища и сети, предоставляются как сервис через интернет. В отличие от традиционных локальных решений, она исключает необходимость закупки и обслуживания собственного оборудования, перекладывая эти задачи на облачного провайдера.

Современные облачные системы строятся на принципе абстракции аппаратных ресурсов, преобразуя их в гибкие логические сервисы. Фундаментом этой модели выступает гипервизор — технология, разделяющая вычислительные мощности сервера на изолированные виртуальные среды. Каждая виртуальная машина функционирует независимо, получая выделенные параметры CPU, RAM и хранилища, что устраняет зависимость между ОС и физическим «железом». В отличие от традиционных дата-центров с фиксированным распределением ресурсов, облака позволяют динамически перераспределять мощности: незадействованная память одной ВМ может быть мгновенно передана соседнему экземпляру без прерывания работы сервисов.

Эластичность инфраструктуры усиливается за счет программно-определяемых сетей (SDN). Их ключевая инновация — декомпозиция управления: контроллер верхнего уровня (control plane) централизованно задает правила маршрутизации, тогда как коммутаторы (data plane) выполняют только транспортировку пакетов. Используя открытые протоколы вроде OpenFlow, система адаптирует топологию под текущие задачи — от автоматического создания изолированных сегментов для клиентов до глобальной балансировки нагрузки между дата-центрами. Такой подход не только минимизирует зависимость от вендорного оборудования, но и позволяет мгновенно блокировать кибератаки на уровне всей сети.

Следующий уровень абстракции — виртуализация хранилищ. Технологии распределенных файловых систем (Ceph, GlusterFS) объединяют дисковые массивы серверов в единый пул с автоматическим восстановлением данных. Информация дробится на блоки, которые реплицируются между узлами, а метаданные управляются кластерными службами. При отказе диска система перенаправляет запросы к другим копиям, сохраняя доступность данных.

Сегодня существует два основных варианта построения облачной инфраструктуры:

* + частные облака — выделенные ресурсы для одной организации;
  + публичные облака — общие ресурсы у провайдера.

Частное облако - это модель локальных облачных вычислений, которая предоставляет выделенные ресурсы, включая вычислительную мощность, хранилище и сеть, ограниченному числу пользователей в рамках одной организации. Большинство компаний выбирают частное облако в тех случаях, когда они хотят иметь огромный контроль над своей информацией и быть в безопасности. Это могут быть виртуальные облака, такие как VMware Cloud и OpenStack, а также облачные решения, предоставляемые поставщиками частных платформ.

Публичное облако – тип облака, которое провайдер предоставляет в аренду на время. Используется для созданий пул ресурсов, выделенного под конкретный проект или задачу. Можно создавать несколько виртуальных серверов и управлять ими, однако физического доступа к оборудованию у компании не будет. Оставшиеся ресурсы, достаточные, чтобы выделить железо, будут использоваться уже другими компаниями. То есть, вашими виртуальными соседями будут сервисы других организаций.

Ресурсы в публичном облаке будут использоваться по модели pay-as-you-go — по мере потребления. В случае пиковых нагрузок (акции, распродажи) облако будет расходовать больше ресурсов и плата за них возрастает. Но когда нагрузка вновь стабилизируется, облако продолжит работать на обычных мощностях и плата снизится. Публичные облака выросли из частных облаков провайдеров, которые накопили экспертизу и поняли, что ее можно предлагать в форме продукта. Так же, как и частное облако, публичное представляет собой инфраструктуру по типу IaaS.

Рассматривая российский рынок облачных услуг, самыми крупными провайдерами являются:

- Timeweb;

- VK Cloud;

- Yandex Cloud;

- MTS Web Services;

- Cloud.ru.

Чтобы детальнее понять разницу между этими двумя типами, нужно чуть углубиться в документации различных провайдеров.

Термин «частное облако» был введен, чтобы провести различие между этими внутренними облачными средами и публичными облачными сервисами сторонних производителей. Пользователи как публичных, так и частных облачных сервисов имеют определенные сходства.

И публичные, и частные облака абстрагируют и совместно используют вычислительные ресурсы, такие как аппаратное обеспечение, сети, программное обеспечение, обычные серверы и серверы хранения данных. Пользователи могут выделять и освобождать ресурсы по мере необходимости и управлять конфигурацией инфраструктуры.

Оба типа облачных сред используют схожие базовые технологии. Они используют виртуализацию для абстрагирования базового оборудования и его предоставления через API. Они также обеспечивают автоматическое масштабирование, автоматическую оркестровку, отказоустойчивость и улучшенные системы резервного копирования.

Как публичные, так и частные облака обеспечивают операционную эффективность ИТ-инфраструктуры компании. Компании могут сократить расходы за счет централизованного управления инфраструктурой. Быстрее масштабируются и быстрее выводят на рынок новые продукты. Существующие мощности используются более эффективно, а затраты снижаются.

Опыт малых и средних предприятий показывает, что во многих случаях публичные облака значительно эффективнее частных.

Гибридное облако — смесь публичного и частного. Кратко говоря – берутся две инфраструктуры, связываются через VPN или прямые каналы связи (что не сильно распространено) и организация по итогу этих действий получает фиксированные мощности с возможностью расширения.

При таком подходе также важно учесть совместимость API, одинаковые сетевые настройки. Иногда ставят шлюзы для синхронизации — чтобы приложения с обоих сторон видели друг друга как одну систему. Также играет роль финансовая составляющая – локальная инфраструктура все также требует вложений, но в меньшем количестве, но затраты на облачную инфраструктуру заметно сокращаются из-за модели оплаты по факту использования, вплоть до почти нулевых значений, когда нет сильной нагрузки и локальная инфраструктура справляется сама.

Если рассматривать такой подход с точки зрения безопасности, то появляется возможность обрабатывать критичные данные — внутри, а различные функции и сервисы, не требующие специфических условий – перенести в публичное облако. Аудит доступа в таких случаях проводится везде, даже если части облака физически в разных местах.

Как выбирать какой-то определенный тип?

Итоговое сравнение всех моделей приведено в таблице 1.1.

Таблица. 1.1 – Сравнение моделей

|  |  |  |  |
| --- | --- | --- | --- |
| Параметр | Частная | Публичная | Гибридная |
| Уровень контроля | Высокий | Ограниченный | Высокий |
| Гибкость и масштабируемость | Ограниченная | Высокая | Очень высокая |
| Теоретическая стоимость инфраструктуры | Высокая | Низкая | Высокая |
| Уровень безопасности данных | Высокий | Зависит от поставщика. Потенциально – низкий | Зависит от реализации и поставщика. Потенциально – высокий |
| Возможность соблюдения требований регуляторов | Высокая | Зависит от поставщика. Потенциально – низкая | Зависит от реализации и поставщика. Потенциально – высокая |
| Производительность | Низкая | Количество возможных ресурсов зависит от поставщика. Потенциально – очень высокая | Очень высокая |

Развивая тему облачных моделей, стоит отметить модель управления инфраструктурой под названием IaC (инфраструктура как код). Условно говоря, это подход, с помощью которого инфраструктура управляется через конфигурационные файлы и код. Вместо ручной настройки серверов, виртуальных машин и прочих необходимых сервисов, системный администратор может написать несколько файлов, в которых опишет что, в каком формате и на каких устройствах хочет получить. Почти та же ручная настройка – но фактически не выходя из консоли.

К самым распространенным средствам эксплуатации такого подхода относят:

* + Ansible. При привычном подходе нужно вручную прописывать каждую команду или скрипт и по кругу запускать их на серверах. Когда серверов много, это процесс становится сложным и трудоёмким, а вот уже с помощью Ansible всю настройку можно прописать в одном конфигурационном файле, который программа разошлёт на большое количество машин. Пример файла конфигурации можно увидеть на рисунке №.
  + Terraform. Этот продукт уже не занимается как таковой настройкой инфраструктуры, он ее только разворачивает в определенной среде. Если у организации имеется какое-то облако или несколько серверов, выделенных под виртуализацию – развернуть и настроить все виртуальные машины можно ровно также через несколько файлов.

Есть еще много другого ПО, которое позволяет автоматизировать управление инфраструктурой организации: Puppet, Chef, Vagrant, SaltStack и многие другие. Но как правило, все они используются в связке с первыми двумя средствами, чтобы как можно меньше лезть в сами сервера и минимизировать ручную настройку.

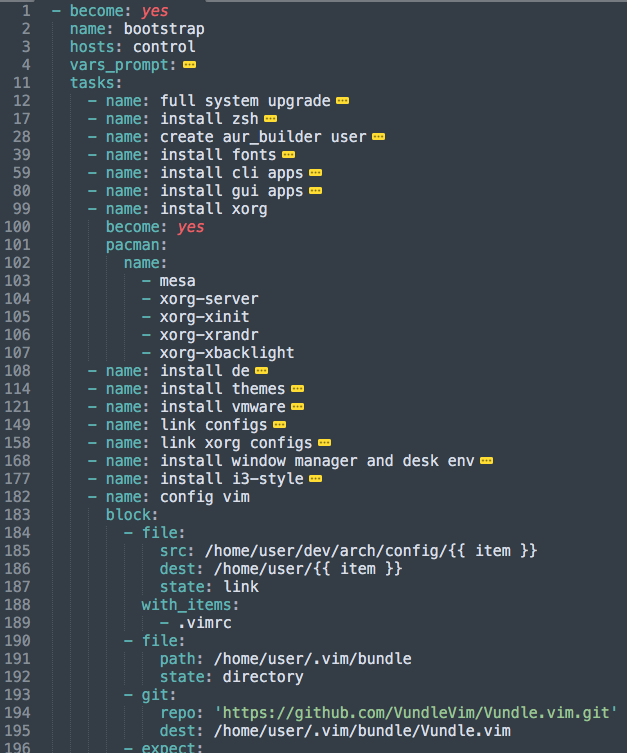


Рисунок № – пример Ansible-плейбука

Самым важным моментом при проектировании инфраструктуры являются сервисы, ПО и задачи, которые они будут выполнять. Эффективность итоговой инфраструктуры в большей части определяется не новизной и новыми технологиями, а наличием отказоустойчивых базовых элементов, которые обеспечивают жизнеспособность всего ПО и иногда аппаратной части инфраструктуры. Ключевая особенность такой архитектуры заключается в способности обеспечивать непрерывность процессов при сохранении гибкости для адаптации к изменяющимся условиям.

К обязательным компонентам любой инфраструктуры в первую очередь относят, конечно, IP-адресацию и организацию сети. Обосновывать такое, в целом, бессмысленно. Без нее ничего в ИТ-инфраструктуре работать никогда не будет.

Организация сети начинается с проектирования её логической структуры. Основой служит разделение на подсети — либо по VLAN, либо через физическую изоляцию. Например, отдельная подсеть для серверов, клиентских устройств и управляющего оборудования. Маршрутизация между ними настраивается на L3-коммутаторах или маршрутизаторах, где прописываются правила доступа (ACL). Для защиты периметра используются межсетевые экраны, блокирующие нежелательный трафик по портам или IP-адресам. Внутри сети часто применяется 802.1X для аутентификации устройств: если компьютер не прошел проверку — доступ в VLAN ограничивается. Отдельное внимание уделяется резервированию: два независимых маршрутизатора с VRRP или HSRP гарантируют, что выход одного из строя не парализует инфраструктуру. Для Wi-Fi сетей аналогично — несколько точек доступа с общим SSID и быстрым роумингом. Всё это требует строгой документации: схемы адресации, таблицы VLAN, политики безопасности. Без этого даже незначительное изменение конфигурации может привести к коллизиям или уязвимостям.

Продолжая тему IP-адресации и организации сети, нельзя упустить DHCP – протокол динамической настройки хостов. Позволяет автоматически присваивать настройки сети хостам, которые отправляют запросы на получение адреса. В основном используются для устройств, которые по большей части принимают трафик, а не получают. Обуславливается это тем, что подобным устройствам в сети нужно чего-то получать, делать что-то и после куда-то опять отсылать данные. Под такую роль подходят именно различные клиентские ПК. Также для связи с каким-то устройством надо знать его IP-адрес (или доменное имя, которое тоже преобразуется в IP-адрес), что при получении адреса через DHCP достаточно затруднительно. Эту проблему могут решать DDNS и статические адреса, выдаваемые по MAC-адресам.

На этом моменте инфраструктура, пускай и откровенно плохая – но будет работать. Хорошим тоном всегда будет настроить систему доменных имен. Задача простая – переводить доменное имя в IP-адрес, чтобы никаких лишних чисел с точками работникам запоминать не приходилось. Есть много других функций, по типу DNSSEC, TXT-записей, которые передают метаданные, но они играют роль обеспечения работоспособности других сервисов, например – почтовых серверов (MX и SPF-записи). Как правило, имена присваиваются только сервисам и серверам, к которым часто приходится обращаться – локальное ПО для конференций, сервисы, взаимодействие с которыми происходит через веб-интерфейс. Для обычных рабочих станций такое обычно не настраивается, они только получают различные записи с DNS-серверов. Но в случае если такое вдруг понадобится – DDNS снова в помощь. Это функция, которая позволяет DHCP серверам при очередной аренде вносить на один или несколько DNS-серверов изменения. Составляются они из имени устройства и обслуживаемой зоны.

Иногда администратору не приходится вручную настраивать DNS – этим занимается контроллер домена. Это как раз тот сервис, который не может работать без доменных имен. Добавляются служебные SRV записи, которые по большей части необходимы для Kerberos-аутентификации. Сам домен строится в большинстве случаев из LDAP (места, где хранятся все объекты домена), DNS и Kerberos. В случае использования AD-совместимого домена, все это разворачивается автоматически. И записи для всех клиентов, и в том числе, что немаловажно – централизованная аутентификация. Существуют и другие реализации: FreeIPA и ее производные (ALD PRO). Эти системы в основном ориентированы на Linux и Unix, но и Windows поддерживают. Также контроллеры домена реализуют общие сетевые папки (netlogon, например), политики доступа к ПК, сетевым папкам и почти всему, что есть в используемом домене. Доменов может быть несколько – корневой, дочерний, несколько дочерних, у которых в свою очередь еще пару дочерних доменов. Задачу аутентификации в таких решают доверительные отношения. Администратор вручную указывает, какие домены могут передавать данные от клиента на другие контроллеры домена.

Вслед за доменом и сетевыми папками приходит проблема износа комплектующих в серверах и в особенности – накопителей. Эту проблему успешно решают RAID-массивы и резервные копии на другие сервера. Проблему, когда из строя выходит сервер с резервными копиями, решают незамысловато - третьим сервером. Продуктов, которые предоставляют функции резервного копирования – несметное множество. От rsync до проприетарного ПО с немыслимым функционалом.

С вышеописанным функционалом работает огромное большинство организаций, будь то облачная инфраструктура или традиционная. Все что выстраивается дополнительно необходимо для обеспечения работы ПО специфичного для определенной отрасли организации или для ускорения каких-то процессов (любое ПО для этого и служит). Но даже резервные копии и RAID не спасут от проблем, которые не видны.

Решения вроде Zabbix или связки Prometheus c Grafana – обязательный минимум для организаций покрупнее. Они следят за нагрузкой на процессор, диски, сеть, предупреждают о критических сбоях или аномалиях. За логами тоже следят – через Elasticsearch или Graylog. Без этого админ узнает о проблеме только когда сотрудники начнут негодовать.

Следующий шаг для оптимизации инфраструктуры – уменьшить количество физических серверов. Не полностью, но большую часть можно заменить виртуальными машинами. Виртуализация дает несколько плюсов: можно быстро масштабировать ресурсы, мигрировать ВМ между хостами, изолировать сервисы. Да и с точки зрения энергопотребления такой подход гораздо выгоднее.

Если уж говорить о виртуализации – нельзя обойти гипервизоры. Proxmox здесь вне конкуренции для небольших и средних инфраструктур. Весь функционал – live-миграция, распределенное хранилище (CEPH), HA-кластеры и прочее идут в составе по умолчанию. Достаточно настроить ноды, прописать общее хранилище – и падение одного хоста останется незамеченным для пользователей. Также имеется встроенная сетевая организация: мосты, VLAN, SDN. Конечно, SDN в Proxmox не может конкурировать с Cisco ACI, но для базовых сценариев функционала хватает. Например, отделить трафик виртуальных машин от управляющего трафика гипервизоров или создать изолированные подсети для тестовых стендов.

Когда инфраструктура начинает достаточно разрастаться, появляются проблемы масштабирования и организации сети. Обычно эти проблемы решаются самыми первыми – еще на этапе создания примерного макета сети, путем выбора одной из распространенных сетевых архитектур.

Стандартная трехуровневая архитектура (также известная как иерархическая модель) — базовая модель для большинства корпоративных сетей. Выделяют уровни ядра, распределения и доступа. Ядро отвечает за маршрутизацию между подсетями, распределение — за агрегацию трафика и применение политик (ACL, QoS), доступ — за подключение конечных устройств. Такая архитектура масштабируется вертикально: добавление новых коммутаторов на уровне доступа не затрагивает ядро. Однако при росте устройств возникают узкие места — например, STP-блокировки или перегрузка L3-интерфейсов. Пример такой архитектуры представлен на рисунке №

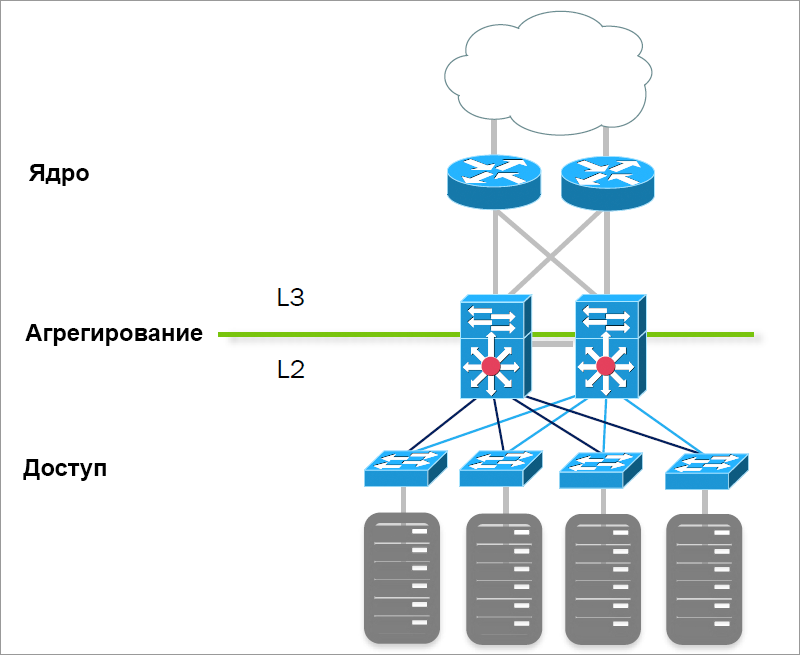


Рисунок № – Иерархическая модель сети

Leaf-spine — архитектура для ЦОД и высоконагруженных сред. Все leaf-коммутаторы (доступ) соединены с каждым spine-коммутатором (ядро), образуя сетку с фиксированной задержкой. Маршрутизация — только L3, что исключает петли и STP. Плюсы: горизонтальное масштабирование, балансировка нагрузки через ECMP, отказоустойчивость (выход из строя spine-узла не влияет на связность). Из мнусов такого подхода выделяют стоимость каждого порта на коммутаторах и невероятную сложность миграции с устаревших сетей. Подходит для кластеров и распределенных СХД. Пример такой архитектуры представлен на рисунке №.

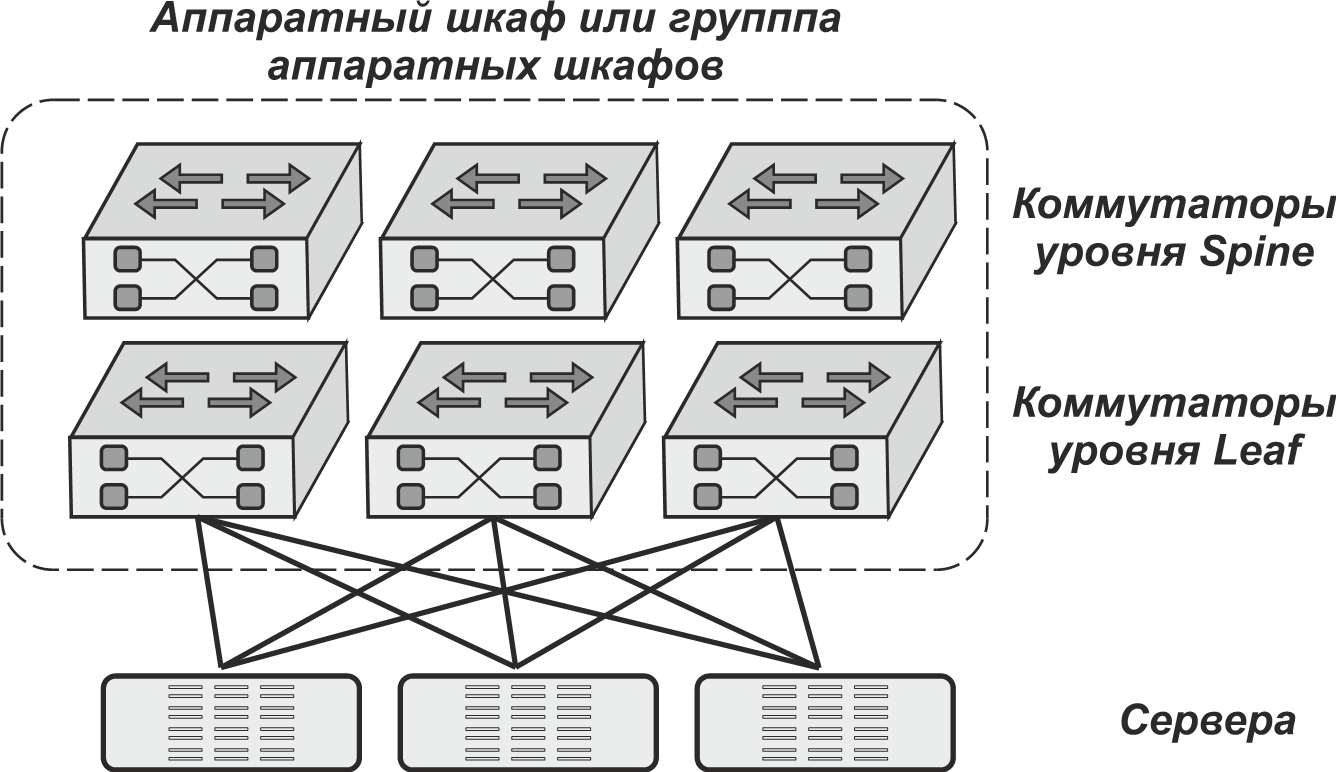


Рисунок № – Архитектура leaf-spine

Гибридные модели — комбинации подходов. Например, трехуровневая архитектура в офисе и leaf-spine в ЦОД. Или spine-уровень на основе VXLAN, что позволяет расширить L2-домены поверх L3-сети. Пример гибридной модели представлен на рисунке №

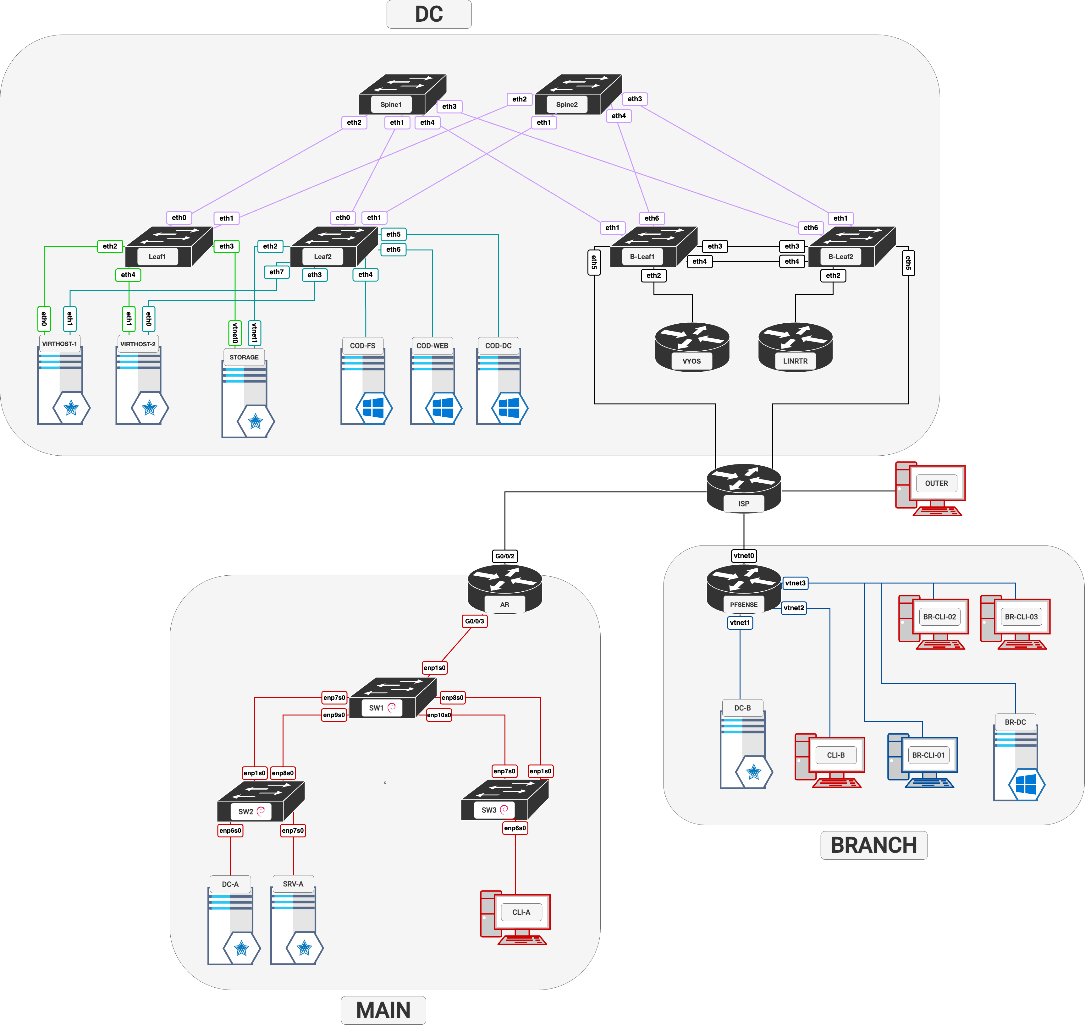


Рисунок № – Гибридная архитектура

Такие решения часто встречаются в гибридных облаках, где часть инфраструктуры размещена локально, часть — у облачного провайдера.

# 

# 1.4 Безопасность инфраструктуры

Здесь описать основные угрозы, как от них защититься, описать всякие законодательные штучки

# 1.5 Требования для построения инфраструктуры

Главным специалистом отдела аналитики и архитектуры государственной единой облачной платформы и государственной информационной системы мне были предоставлены требования, которые необходимо учесть при проектировании и разработке макета.

# 1.6 Выбор лабораторной среды для создания макета

Для создания и тестирования сетевых макетов в рамках проекта необходима лабораторная среда, обеспечивающая реалистичную эмуляцию или симуляцию сетевой инфраструктуры. Выбор подходящей платформы зависит от множества факторов, включая функциональные возможности, требования к ресурсам, совместимость с реальными устройствами, уровень сложности настройки и поддержки, наличие встроенных инструментов для анализа сетевого трафика и диагностики. Основные инструменты, примерно подходящие под требования описаны в этом разделе.

Cisco Packet Tracer — Это симулятор, разработанный Cisco для обучения базовым сетевым концепциям. Он имитирует поведение устройств Cisco (роутеры, коммутаторы) через упрощенные алгоритмы. Внутренняя архитектура основана на визуальном моделировании логических и физических слоев сети, включая автоматическую генерацию кабелей и упрощенную эмуляцию протоколов (RIP, OSPF, DHCP и прочие протоколы). Каждое устройство — программная модель с ограниченным набором команд, которые не включают полный функционал аналогичного физического устройства.

Из явных плюсов выделяются:

* + относительная простота использования;
  + поддержка лабораторных работ внутри интерфейса;
  + не требователен к ресурсам;

Минусов слегка больше:

* + изолированность в экосистеме Cisco;
  + отсутствие возможности взаимодействия со сторонними программами;
  + ограничения на количество узлов в одном проекте;
  + не бесплатен;
  + только достаточно устаревшие модели оборудования;

Переходя к следующим симуляторам, можно упомянуть малоизвестный Mininet. Это эмулятор сетей, использующий встроенные средства Linux и OpenFlow для создания виртуальных устройств. На всех устройствах в рамках проекта можно полноценно пользоваться утилитами из ядра.

С основной задачей в построении сети этот инструмент справится, но с нюансами. Для сетевых устройств используются OpenFlow-контроллеры, что встречается только в SDN и особой популярностью не пользуются и получается, что и пользы в реализации данного проекта – не несет.

Следующий инструмент имеет подобный функционал и гораздо большее применение. ContainerLab – инструмент, который также используется для создания и тестирования сетей. Топология строится не в графическом интерфейсе, а описывается в yml файле, почти идентичному файлу docker-compose, а после управляется через терминал.

Инструмент воссоздает нужную сетевую инфраструктуру используя docker-контейнеры и стандартные средства по типу виртуальных пар интерфейсов и мостов в Linux. Так как в docker’е можно запустить практически все что угодно, за исключением специфических устройств, то и в этом инструменте их тоже можно использовать. В поддержке имеются все самые используемые linux-дистрибутивы, также есть возможность запуска сетевого оборудования Cisco, Huawei (сильно урезанные подобия VRP), MikroTik, Arista и многих других. Пример использования представлен на рисунке 1.1

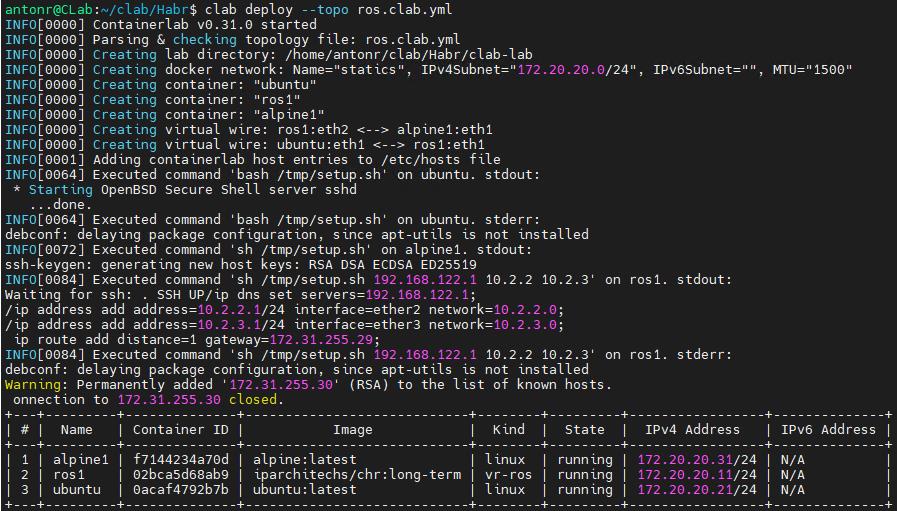


Рисунок № – пример использования ContainerLab

Из интересных моментов также выделяется возможность запуска Windows, но только сильно странных редакций для интернета вещей. Ресурсов оно расходует очень мало, так как не требуется полная виртуализация устройств, они работают просто как отдельный процесс в системе. В рамках проекта следующий пункт роли не играет, но также имеется возможность интеграции в процесс автоматического развертывания при разработке какого-либо продукта в команде.

Подводя итог, инструмент имеет потенциал, в виде:

* + быстрого развертывания проекта;
  + отличной интеграции с облачными технологиями;
  + ресурсоемкости;
  + большой поддержки числа производителей.

Минусов слегка больше:

* + непрактичное управление;
  + сложность управления и построения больших сетей;
  + малая функциональность в сравнении с другими продуктами.

Следующий эмулятор – EVE-NG. Представляет из себя набор инструментов для работы с виртуальными устройствами, построением сетей, коммутацией с реальным оборудованием. Возможности данного продукта позволяют легко использовать, управлять, коммутировать моделируемое сетевое оборудование. Имеет две редакции – бесплатную, с ограничениями в виде 64 узлов в проекте (что в целом и немало) и платную – без ограничений, с поддержкой контейнеров в одном проекте с виртуальными машинами. В отличии от прошлых инструментов – используется полная виртуализация используемых устройств, что позволяет производителям в образах устройств вместить все функции, которые используются в реальном оборудовании. Пример топологии представлен на рисунке №.

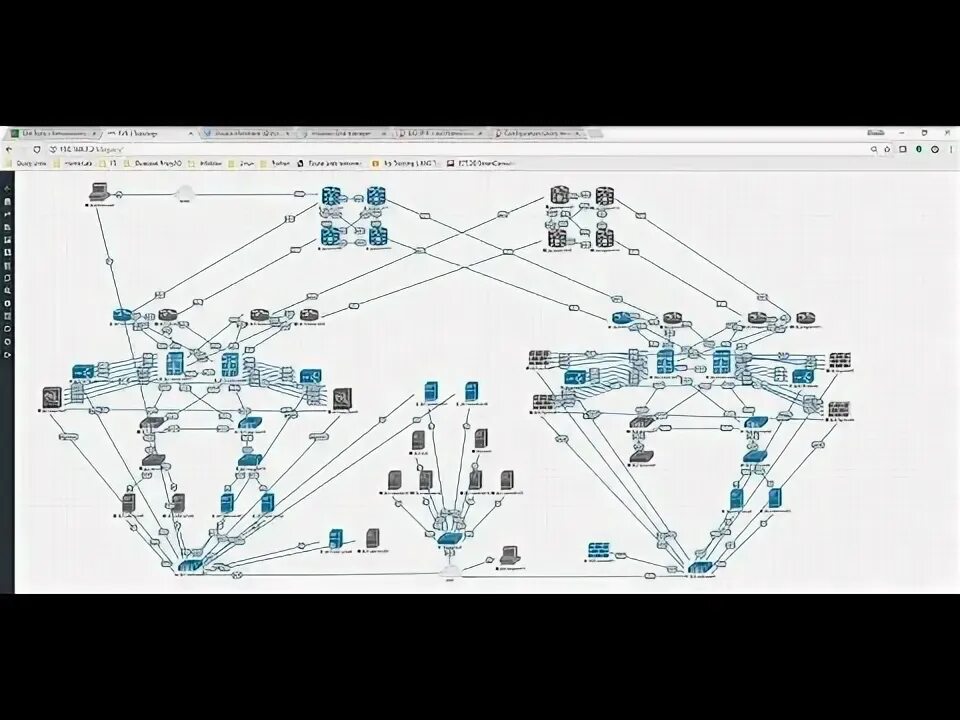


Рисунок № – пример топологии в EVE-NG

Продукт сильно серьезный, так как до определенного момента использовался в исследованиях ФСТЭК в рамках исследований и тестирований уязвимостей. Имеет поддержку вообще всех производителей, которые представляют программные образы своих устройств, используется через веб-интерфейс, а управляется – через терминал. Также позволяет прослушивать весь трафик на абсолютно всех соединениях, которые имеются в проекте, через WireShark. Интересный нюанс - при должном понимании работы EVE-NG можно прекрасно интегрировать проект с физической инфраструктурой, даже в случае, когда просто требуется проверить совместимость. Развертывается данный инструмент может в качестве виртуальной машины или сразу на физическое оборудование и поставляется в виде установщика, по типу обычной ОС.

Последний инструмент, и по мнению сотен тысяч людей – самый перспективный из всех вышеописанных – GNS3. Если дословно – графический симулятор сети, но говоря правильно – эмулятор. Инструмент абсолютно бесплатен, не имеет ограничений в сравнении с прошлым инструментом. Так как проект открытый, то и технической поддержки не предоставляется, что заставляет различные организации посмотреть в сторону EVE-NG.

Развертываться он может в качестве виртуальной машины и также в качестве хостовой ОС, но первое, что его отличает от остальных инструментов – это возможность запуска еще и на любой ОС, без необходимости развертывания виртуальной машины. Имеет полную поддержку всех функций остальных инструментов. Также выделяется возможность запуска машин в проекте на разных серверах, чем не может порадовать EVE-NG. Получается что-то вроде объединения мощностей серверов как в облачной инфраструктуре, но только для запуска ВМ в рамках проекта. Среди всех этих преимуществ есть и минус – для построения топологий больше чем пара устройств, потребуется неплохое количество ресурсов.

GNS3 имеет веб-интерфейс, представленный на рисунке №, но достаточно скудный.

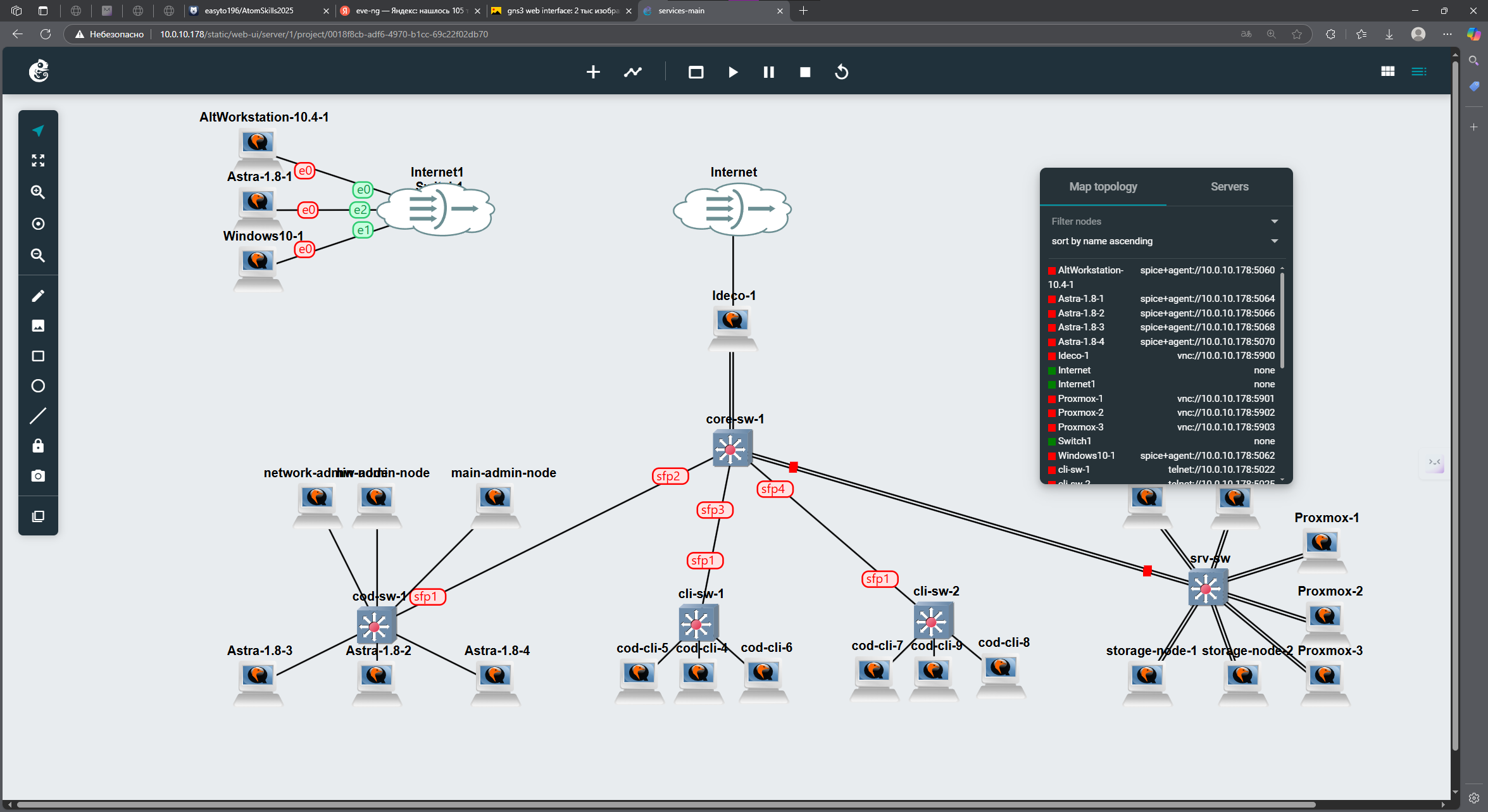


Рисунок № – внешний вид веб-интерфейса

Лучшим вариантом использования является клиент, который включает в себя сразу и сервер. Внешний вид клиента представлен на рисунке №. В одном проекте возможна совместная работа сразу нескольких людей, все изменения в проекте отображаются с минимальной задержкой.

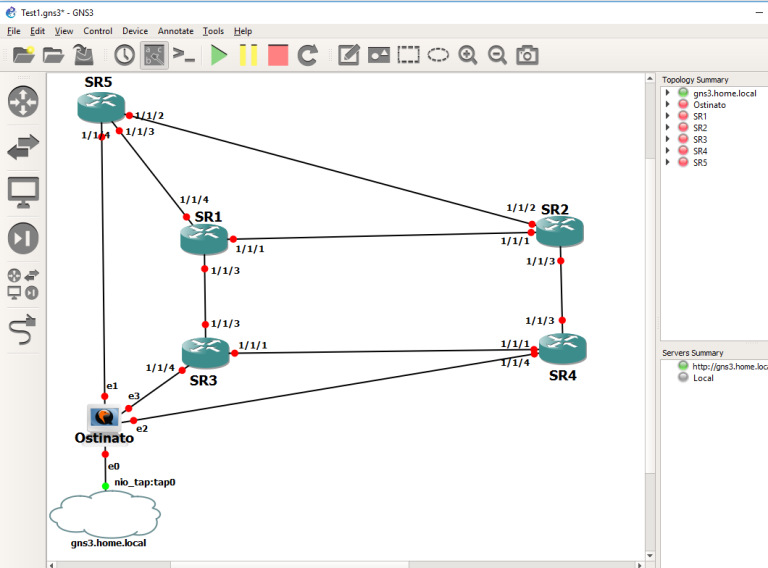


Рисунок № – внешний вид клиента GNS3

Подводя итог – GNS3 показал существенные преимущества, что и делает его фаворитом во многих сферах. каждый инструмент имеет свои уникальные особенности, подходящие для разных целей и условий работы. Однако, исходя из потребностей, предъявляемых к будущему проекту и характеристик рассматриваемых инструментов, можно сделать вывод, что GNS3 является наиболее подходящей платформой для создания сложных и детализированных сетевых топологий. Множество функциональных возможностей, поддержка полноценных ОС и безумная возможность настройки делают его идеальной лабораторной средой.

# **2 Практическая часть**

# 2.1 Определение ПО, сервисов и производителей, подходящих под требования

1) пару предложений введения

2) на каждый пункт, который отвечает за вендоров/ОС делаем какие-то выводы, сравниваем

3) можно представить табличку с итоговой конфигурацией стенда, какие оси роутеры коммутаторы

# 2.2 Планирование и подготовка к реализации будущей инфраструктуры

Здесь описать установку темплейтов, настройку гнс, само проектирование

При создании макета с виртуальными машинами, сначала всегда требуется создать шаблоны ВМ, из которых все будет разворачиваться. Первым делом будет создан шаблон Astra Linux обеих версий (1.7 и 1.8). Конфигурации шаблонов показаны на рисунках №. 4 ГБ и 4 vCPU были выбраны, так как данный объем считается избыточным для макета и тестирования инфраструктуры. Также играют роль минимальные требования и клиентских ПК, и серверов, ПО которых должно запустится и функционировать.

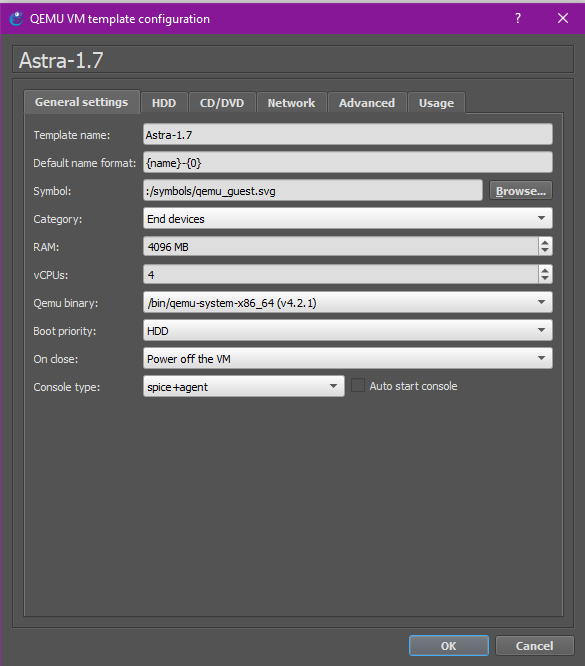


Рисунок № – Настройки шаблона Astra Linux 1.7

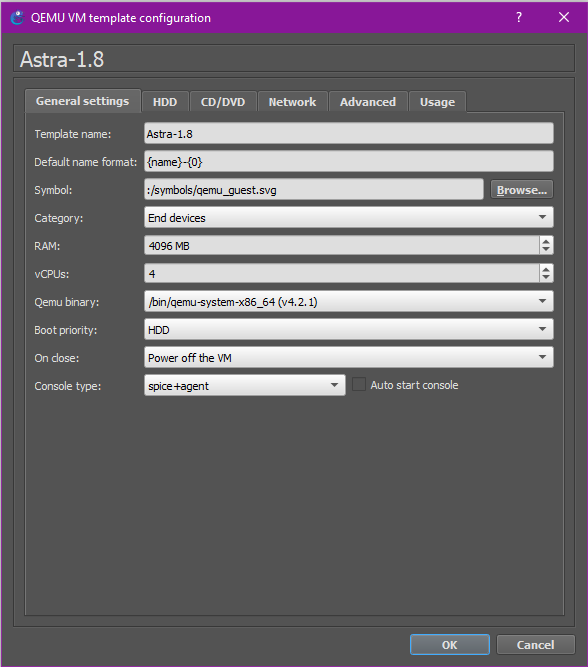


Рисунок № – Настройки шаблона Astra Linux 1.8

Далее для шаблонов необходимо создать диск и установить саму операционную систему. Ключевые моменты процесса установки показан на рисунках №. Необходимо придумать имя для учетной записи администратора:

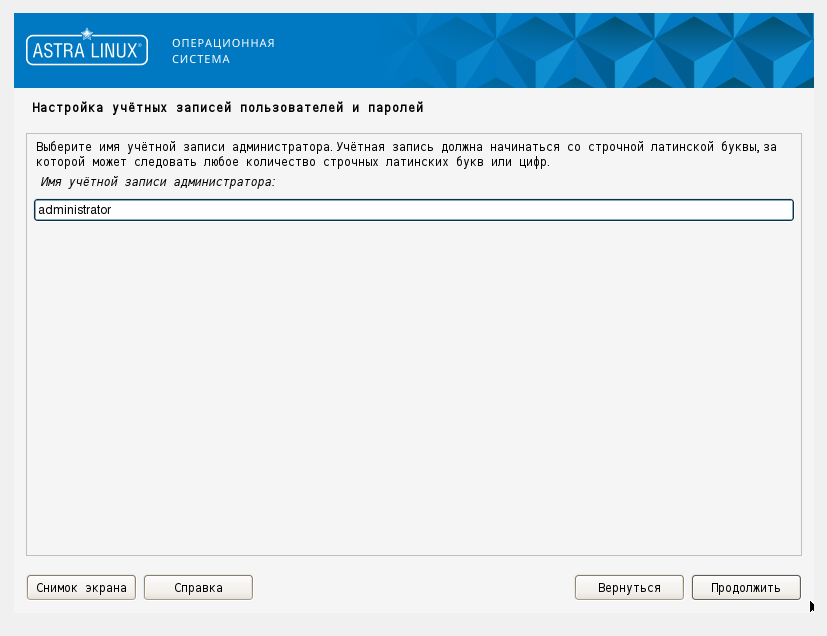


Рисунок № – Создание аккаунта администратора

Дальше необходимо два раза повторить пароль для нашего пользователя. Пароль в рамках макета выставляется символически, ради ускорения развертывания. После ввода пароля и нескольких шагов по настройке системы появится этап разметки диска. Ввиду отсутствия требований можно выбрать «Авто», что автоматически создаст загрузочный раздел и раздел подкачки.

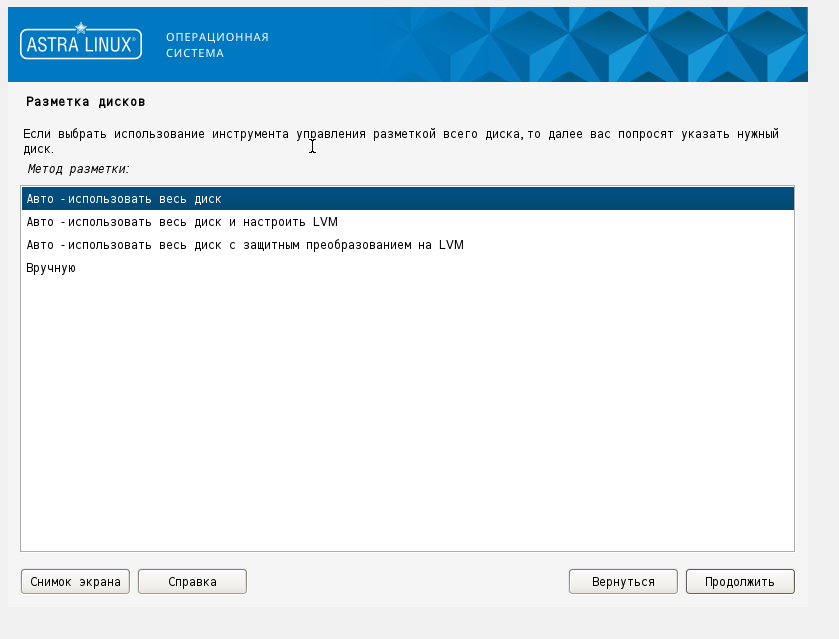


Рисунок № – Выбор метода разметки

Далее, если был выбран метод «Авто», появится окно выбора схемы. Необходимо просто выбрать «Все файлы в одном разделе», так как второй в данном случае никакой роли не играет.

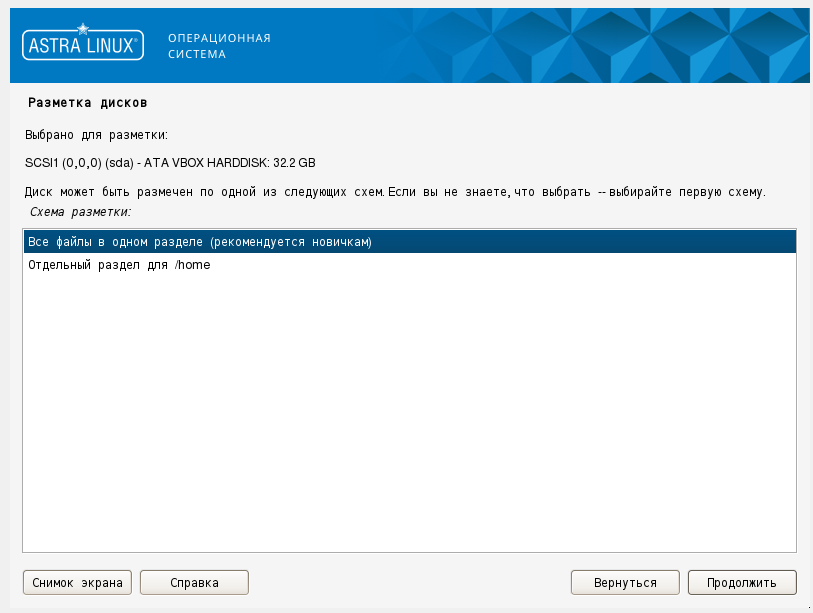


Рисунок № – Выбор метода разметки

После разметки идет следующий важный шаг в виде выбора ПО, которое будет установлено сразу же, без ручного вмешательства. Для полноты и упрощения последующей работы необходимо выбрать пункты:

- Графический интерфейс

- Средства работы с Интернет

- Консольные утилиты

- Средства удаленного подключения SSH

Пункт с ядром не выбирается ввиду отсутствия совместимости многого ПО с ядром «hardened».

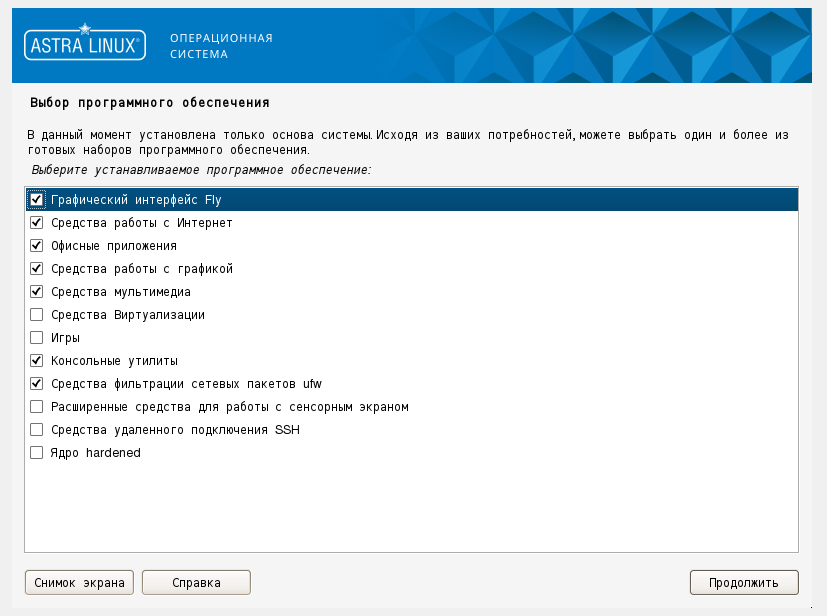


Рисунок № – Выбор программного обеспечения

После идет завершающий этап основных настроек – выбор режима работы ОС. Режимы отличаются функциональностью, которая отвечает за безопасность информации.

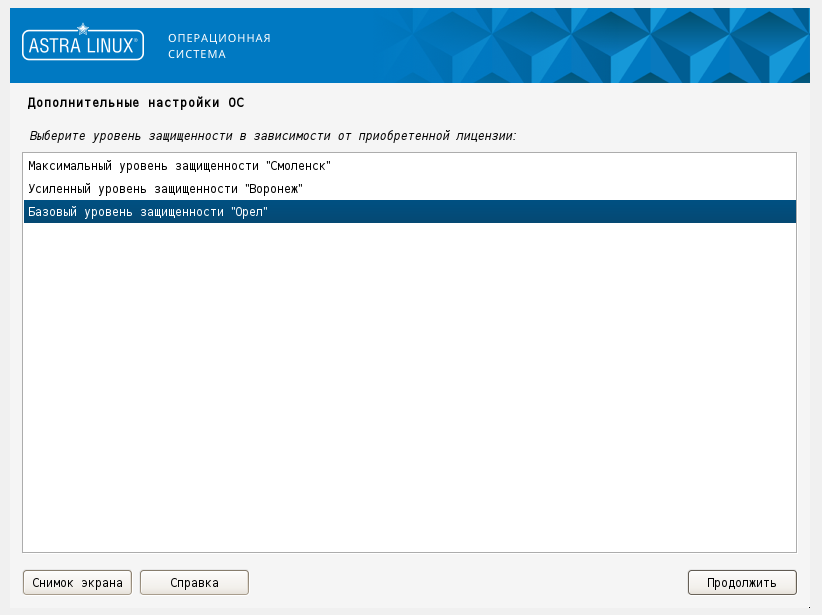


Рисунок № – Выбор режима работы ОС Astra Linux

Режим выбирается в соответствии с приобретенной лицензией, но в данном случае нужно выбрать базовый уровень защищенности, чтобы без особых проблем с функциями безопасность базово настроить ОС. После установки режим работы также можно сменить.

# 2.3 Настройка сетевого оборудования

Текст

# 2.4 Настройка серверных ОС

Текст

# 2.5 Развертывание сервисов

Текст

# 2.6 Настройка пользовательских ОС

Текст

# 2.7 Проверка соответствия представленным и законодательным требованиям

Текст

### 3 Охрана труда и производственная санитария

Текст

# **Заключение**

Текст

# **Перечень сокращений и условных обозначений**

ГБ – гигабайт

ГГц - гигагерц

Ссылаюсь только на ГОСТ 2.32-2017 п. 6.15 «Перечень сокращений, условных обозначений, символов, единиц физических величин и определений должен располагаться столбцом без знаков препинания в конце строки. Слева без абзацного отступа в алфавитном порядке приводятся сокращения, условные обозначения, символы, единицы физических величин, а справа через тире - их детальная расшифровка.»

# **Список используемых источников**

Приведенный ниже список используемых источников выполнен исходя из ГОСТа Р.105-2019 п. 6.4.2 «Элемент «Библиография» размещают перед листом регистрации изменений. Выполнение элемента и ссылки на него в тексте — согласно требованиям к элементу «Список использованных источников» по ГОСТ 7.32.»

ГОСТ 7.32 п. 6.16 «Сведения об источниках следует располагать в порядке появления ссылок на источники в тексте отчета и нумеровать арабскими цифрами с точкой и печатать с абзацного отступа.»

# Приложение А

(обязательное)

**Прайс-лист с ценами**