Приложение № 4

к конкурсной документации на реализацию среднесрочной ведомственной целевой программы «Создание единой системы межведомственного и внутриведомственного обмена электронными документами».

Требования к архитектуре программных решений.

Оглавление

Введение.

В рамках этого документа рассматриваются программы, программные комплексы, программные архитектуры.

В документе даются обязательные требования к элементам Технического задания (далее ТЗ) и требования к прототипам.

ТЗ должно давать исчисляемые по времени и объему характеристики сбоя, отсутствия сервиса или недоступности.

Численные и структурные характеристики объекта автоматизации, определяют выбор архитектуры программных решений. Численные и структурные характеристики объекта даны в Приложении № 5.

Аппаратное обеспечение рассматривается в той части, в которой оно напрямую влияет на доставку сервиса (на уровне прототипов). Случаи и проблемы производительности проектируемой системы, которые вызваны зависимостями и не находится в пределах специальных решений должны быть исключены.

В отдельном, закрытом разделе ТЗ, должны быть даны программные решения, обеспечивающие криптографию, реализацию и защиту закрытого информационного контура.

1. Требования технологии программной архитектуры.

В требованиях к архитектуре программных решений (базовых технологий) ТЗ определены два уровня:

A) технологии программной архитектуры,

B) системное программное обеспечение - программы, решающие задачи общевычислительного характера, выделения и разделения ресурсов, доступа к устройствам, обеспечивающие среды для разработки, запуска и выполнения других программ.

C) Слой ППО. Этот слой рассматривается не на уровне визуализации и конкретных пользовательских «дизайнов», которые должны быть установлены специалистами и писателями UI, а как как слой.

Требование к СПО – в первую очередь - требования к СУБД, операционным системам, контейнеризации, hiload, серриализации процедур о объектов и тп.

Рисунок №1.

Технологии программной архитектуры – это не программные технологии, не цифровые и не ИТ технологии, это стратегии строительства информационных систем, которые каждая на своем логическом участке корпуса конечной системы в целом и в частях закладывает основные концепты ИС, жизненного цикла, концепты безопасности системы, и в то же время определяет ее способности функционировать в предельных и экстремальных режимах.

Проектируемая система – система экстремальных размеров и экстремальных транзакционных параметров, технологии (стратегии) программной архитектуры важный элемент проектирования, так как позволяют добиться исчисляемых заданных параметров эффективности и безопасности, максимальных при заданных параметрах бюджета.

В целом технологии программной архитектуры не описывать экосистему СЭД, они дают основу экосистемы СЭД.

На Рисунке №1 приведена архитектура проектируемой системы

В конкретное проектирование ТЗ должен выполнить архитектор системы уровня Solution architect**,** или Enterprise architect.

В долгосрочной перспективе и в пересчете на базу (единицу) затрат технологии программной архитектуры закладывают основу экономической целесообразности внедрения системы и самые важные, наиболее критические, а также общие и существенные аспекты архитектуры безопасности и стоимости владения ИТ инфраструктуры конечной системы.

Программное обеспечение и аппаратура криптографической защиты каналов связи, специальные средства и аппаратура локализации пакетов и маршрутизации должны быть включены в проект. Это важные элементы безопасности системы.

Важнейшие аспекты.

А) Технологии программной архитектуры должны определять базовую безопасность.

В) Технологии программной архитектуры должны определять базовую стоимость в пересчете на единицу хранимой информации.

C) Технологии программной архитектуры должны определять предел достижимой масштабируемости без потери заданных параметров безопасности, целостности (контролируемости) и управляемости системы в границах заданных функционалов.

Методология проектирования схем данных должна следовать 12 правилам Кодда, СУБД должна обеспечивать функциональные характеристики требуемые от ИС, без риска трансформации в эксплуатационные и операционные угрозы.

Характеристика C) должна показывать стабильность системы через применимые отраслевые процедуры самооценки, процедура публичных стандартов и рекомендаций по информационной безопасности.
Основное «дизайн» требование к технологиям программной архитектуры – любое решение любого уровня не должна влиять на базовую безопасность.

A) СУБД

Главнейшим вопросом является выбор системы управления баз данных и архитектуры - СУБД.
Это становится критическим выбором, за который ответственность несет непосредственно Заказчик, его назначенный Enterprise (Senior) Project Lead, or Head of Supervisors. Этот выбор становится экстремальным, если конечная система является экстремальной по объемам, экстремальной по структуре, экстремальной по ответственности.

Для обработки метаданных (об объектах) потребуется быстрая, реляционная, OLAP (с аналитической вычислительной подсистемой)/OLTP СУБД. Такая СУБД умеет хорошо и главное быстро искать информацию о документах, где они хранятся, или связях между документами, или различные тэги (исторические, цветные, иерархические, численные, индексные и т.п.). Но для хранения самих документов, статического, или медленно изменяемого содержимого документа будет достаточно не очень быстрой, но с достаточной отгрузкой, документно-ориентированной СУБД – такая СУБД умеет правильно и хорошо хранить и упаковывать тело документов и выгружать содержимое пользователю.

Требование к квалификации *- работу или консультирование выполняет специалист уровня Database Advanced Manager имеющий компетенции Concepts of Relational Database Design & Emerging Database Models, Technologies and Applications, а также персоны, которые имеют многолетний успешный опыт реализации крупных и ответственных проектов.*

Есть много систем СЭД, которые хорошо реализованы на верхнем UI уровне, но, не верный выбор СУБД моментально снижает надежность и безопасность системы.

Выбор СУБД в любом случае исключает миграцию в будущем, а если выбрана не соответствующая уровню решаемой задачи, переход никогда не сможет гарантировать актуальность и безопасность данных. Обычно переходы в исторической перспективе невозможны. Только в рамках версий одного вендора. Для крупных систем такая миграция тем более невозможна, и при необходимости замены СУБД придется менять все компоненты.

Самая главная характеристика объекта автоматизации –масштаб, выражающийся в количестве и сложности структуры. В Приложении №5 показано, что предмет автоматизации - а(э)пи (υπερ) над-организационная система разнородных и разнонаправленных процессов и объектов. Система максимального масштаба – система уровня страны. Это экстремальная система с экстремальными характеристиками объемов, пользователей, доменов организационных структур.

Мы не будем здесь указывать весь спектр влияющих на задачу данных важных публичных обследований в отрасли. Но укажем наиболее важные.

В предыдущий период было проведено несколько анкетировании текущего документооборота на предмет структуры и количества документов. И все эти обследования не включали наиболее важнейшие аспекты, которые должны были быть обязательно включены, как например, фактическое количество документов.

Количественные оценки.

В анкете обследования РедXXX указано максимальная цифра возможных документов в планируемой системе, равная 20 000 (двадцать тысяч) документов.

Во втором независимом анкетировании была также указана цифра 20 тысяч документов.

Есть указания на то, что было проведено еще одно независимое обследование, где указывалось 40 тысяч документов, генерируемых государственными органами.

Фактический обзор, проведенный ГОСКОМИТЕТОМ ПО СВЯЗИ, МАССОВЫМ КОММУНИКАЦИЯМ И ЦИФРОВОМУ РАЗВИТИЮ в 2015 году показывает цифру 120 000 (точнее, 128 тысяч) фактических документов. К указанной цифре необходимо добавить 15% рост и мы получаем цифру 140 тысяч документов на текущий момент.

Необходимо отметить, что данные по фактическим исходящим и входящим документам предоставили всего 28 структур и организаций (Таблица №2), а по данным текущего обследования таких структур, попадающих в объект автоматизации (Таблица №4, Приложение №5) – 84 организаций первого уровня, без включения подуровней. Причем в списке отсутствуют крупные и системные государственные институты. Полученную цифру следует скорректировать, 140 000 * 84/28 ~ 420 000 тысяч документов.

Эта цифра указана без организаций нижнего уровня – организационно правовая форма которых государственная – таких организаций (по данным ГОСКОМСТАТА РА) 763, и они не включаются в подсчет.

Следует учесть общие информационные законы, подтверждающие, что “программы становятся медленнее значительно быстрее, чем компьютеры”, об экспотенциональном росте количества информации в информационной системе после ее инкарнации - парадокс Джевонса – резкое повышение потребности в вычислительной мощности по мере предоставления доступности.

При необходимости интеграции проектируемой ЭДО с ведомственными системами, такими, у которых есть на текущий момент свой специфичный, но некий ЭДО, эта цифра будет включать дополнительные значения, возможно, экспоненциальные значения, помимо того, что интерфейсы взаимодействия с

внешними системами также должны предусматривать экстремальные нагрузки:

Количество документов **≥ 420 000 (четыреста двадцать тысяч).**

Количество министерств комитетов и управлений – 84.

В связи с указанными цифрами необходимо также принять решение, включать ли в объект автоматизации структуры среды образования, медицины и документов гражданина и в какой форме.

Прогноз количество работающих ~ неизвестно, но не мнее 2000.

Решение должно предусматривать вертикальные (структурные) основы такого расширения в будущем.

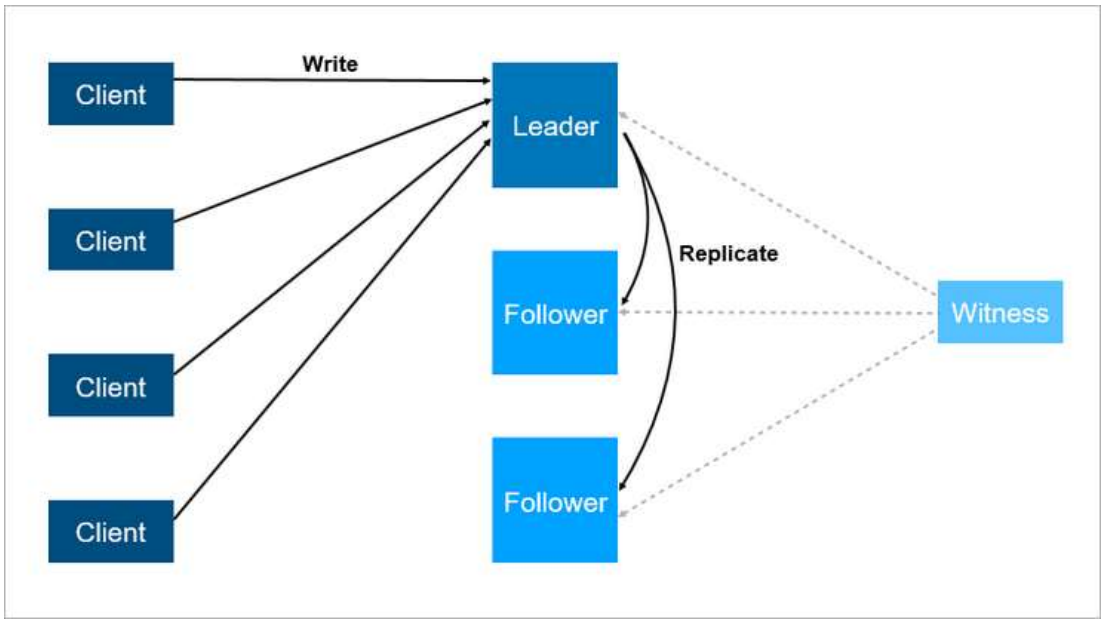
Применимые программная архитектура и технологии.

Дизайн устойчивости и способность решать задачи.

Задачи, требующие поддержу на уровне СУБД отнесены к критическим в разделе D.

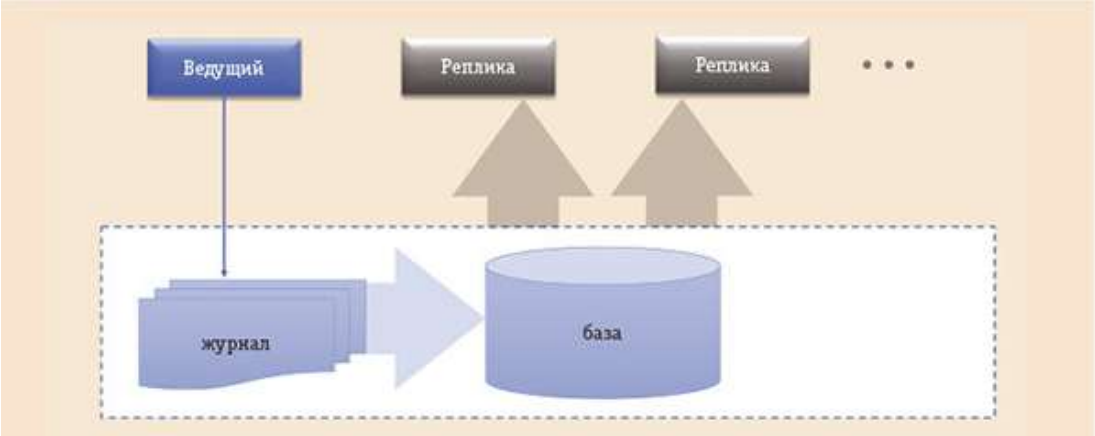
Инженерия уровня СУБД и применимые технологии обязаны поддерживать классические стандарты и проверенные временем передовые практики.

Дизайн устойчивой архитектуры. Вариант прототипа 1- простой фаловер:



Требование №1. СУБД должна поддерживать FOLLOWER стандартно.

Вариант 2 - архитектура с дезагрегированной кластеризацией пула нескольких DBMS, когда ведущий вычислительный экземпляр обрабатывает транзакции. Предусмотрен запуск множества экземпляров для синхронного чтения; журнал записывается в систему хранения, которая самостоятельно восстанавливает базу данных. Систему хранения - единая и доступная всем вычислительным экземплярам, что дает преимущество “быстрой отдачи” и гарантированного восстановления после сбоя.



Вариант прототипа 2.

Требование. СУБД должна поддерживать дезагрегацию и кластеризацию стандартно в рамках решения вендора.

Требования к планируемому “сбою” - программируемые события.

Готовность в процедурах гарантированного восстановления.

Это означает, что СУБД должна прозрачно поддерживать такие процедуры самостоятельно, без привлечения вспомогательного ПО (а не на уровне операционных систем). В СУБД должна быть реализована оперативная маршрутизация и горячая репликация на несколько экземпляров.

Например, **Oracle RAC** объединяет все узлы в один кластер, независимо от того, в каком физическом ЦОД они располагаются, и они все включены в активную работу, как и в случае локального кластера, при этом ПО верхнего уровня никак не будет “переживать” отключения или динамическую меж кластерную маршрутизацию и восстановление Oracle Recovery Manager, а пул соединений обеспечивает кластеризация **Oracle Resident Connection Pool. Все указанные системы являются резидентами экосистемы Oracle.**

Требование №2. СУБД должна поддерживать маршрутизацию и репликацию стандартно, в границах собственной экосистемы решений.

Это означает, что для обеспечения безопасности на базовом уровне требуется СУБД, обеспечивающая прозрачную инкапсуляцию этих технологий в рамках собственной архитектуры в сочетании с системной архитектурой.

Например, MSSQL никогда не предоставит гарантии восстановления ни на одной операционной системе, и в документации на эту СУБД вы не найдете упоминания об принятии ответственности.

Добиться безопасной масштабируемости БД для распределенных систем, каковой будет СЭДО на сегодняшний день, возможно двумя способами - **Вертикальное масштабирование – добавление** на сервер памяти и дисков. Это имеет свои пределы — по количеству ядер на процессор, количеству процессоров и объему памяти.

Горизонтальное масштабирование (или кластеризация – наборы таких машин называются кластерами) - используем много различных машин СУБД и распределение (маршрутизация) данных между ними. СУБД должна поддерживать шардрирование - — то есть для каждой записи определить, на каком конкретно сервере она будет размещена.

На первом этапе эксплуатации системы эти характеристики не будут существенно влиять на видимые эффекты производительности и безопасности. И это может вызвать к жизни призывы не учитывать это. Но на указанном объеме данных зависимость будет экспоненциальной по времени – то есть риски производительности могут наступить очень быстро.

Имеющийся опыт использования в СЭД масштабируемых реляционных БД говорит о том, что именно вертикальное масштабирование вносит основной сектор в стоимость данных и оказываются дорогими как раз за счет резкого увеличения стоимости такого масштабирования. Обычно это происходит на этапе перехода к интеллектуальным функциям. То есть технологии ИИ и стратегия добавления дисков на полку не совсем совместимы (по стоимости).

Требование №3. СУБД должна стандартно поддерживать расширенную безопасность и действительно быстрые протоколы аварийного восстановления, автоматическое резервное копирование в границах собственной экосистемы решений.

Использование NoSQL-решения будет без транзакций, но и без гарантий ACID, которые предоставляют реляционные СУБД. ACID для проектируемой ЭДО необходим, так как проектируется система нескольких организаций, и

придется искать много разных мета-данных в сложных аналитических запросах.

Требование №4. СУБД должна стандартно поддерживать средства анализа данных и бизнес-аналитики, обеспечивая надежный (без мутаций) аналитический отчет и запись его результатов.

Подтвержденный отраслевой опыт.

Распределенные системы поддерживают сценарии, когда данные “исчезают” в силу гарантированного риска в гетерогенных реляционных масштабированных системах. Обследование показывает, что кластер Redis теряет 45% сохраненных данных, кластер RabbitMQ — 35% сообщений, MongoDB — 9% записей, Postgress – 7%, BigDAWG - до 6.1%, Cassandra — до 5%, Oracle до 2,9 %, FoundationDB, 1,3%. Речь идет о потере данных после того, как кластер сообщил “клиенту” об успешном сохранении. Этот показатель называется транзакционной энтропией.

В целом это означает, что данные в проектируемой системе в соответствии с нулевой гипотезой Колмоговора-Смирнова, с вероятностью 97% нестабильности данных в системе станет равным указанным показателям СУБД, по гамма-функции - $\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt = \frac{e^{-\gamma z}}{z} \prod_{k=1}^{\infty} \left(1 + \frac{z}{k}\right)^{-1} e^{z/k}$, $\gamma \approx 0.577216$, где $\prod_{k=1}^{\infty}$ это объем данных (Таблица №1), а k, z унарные коэффициенты к исчислимому вполне времени dt равного пяти годам эксплуатации системы.

Требование №5. СУБД должна стандартно поддерживать целостность данных с помощью ограничений, триггеров и передовых методов управления параллелизмом в границах собственной экосистемы.

Техническое решение СУБД должно гарантировать жизнеспособность и историческую ценность данных на срок не менее 15 лет. Это законодательно установленная норма до списания стоимости внедрённой системы в соответствии с амортизационными нормами.

Расчеты относительно имеющихся с нашем распоряжении данных (таблица №1 и №2) показывают, что при ТЭ ≥ (5) достижимое время неактуальности и нестабильности данных системы равно минимум трем, максимум пяти годам – через пять лет система гарантировано

трансформируется в нестабильную систему, а к трем годам эволюции система будет близка к нестабильности.

Требование минимального срока жизни

15 лет распределенного функционирования, роста и усложнения структур, СУБД с показателем ТЭ ≤5.

Мы не будем рассматривать такие характеристики СУБД, как используемая модель данных, реализация языка, триггеры или хранимые процедуры.

Но это также означает, что если мы будем использовать для разных логик разные DBMS, мы сможем использовать плюсы OLTP & ACID там, где он нужен и критичен, и OLAP и объектность, там где нужны именно они.

Например, MongoDB как объектная база, умеющая хранить JSON объекты (другими словами – объекты документов) не умеет делать то, что может Oracle Database Sharding для поддержки десятков тысяч транзакций при построении метаданных. Возможна архитектура, когда для хранения графов (строительства быстрой математики связей и вершин) используется графовая БД. Имея в виду, что проектируемая система, это система «тысячи подключений, тысячи операций в секунду, тысячи гигабайт», на которых такие системы как MySql, MS-Sql или PostgreSQL быстро исчерпают ресурсы управляемости и ресурсы вычислимости.

Важнейшие требования к СУБД,

влияющие на конечные характеристики проектируемой системы, мы также относим:

“истинный” параллелизм вычислений

пуллирование ресурсов памяти

встроенные механизмы поддержки пула ресурсов соединений,

низкая расчетная стоимость выделения пула соединения на клиента, на транзакцию,

встроенная гибридизация и мультимастеринг

балансировщик пула соединений на уровне ядра машины СУБД

избыточность встроенных механизмов отказоустойчивости

поддержка и классической и неклассической реализации режима консенсуса

режим сегментирования кластеров

виртуализация программных и табличных сущностей

перекрестные множества кластеров данных для различных сегментов конечных потребителей (верхних и горизонтальных подпрограмм и программ)

прозрачная и предустановленная поддержка маршрутизации данных

включении групп отказоустойчивости

истинная транзакционность и поддержка изоляции

исключение неразборчивого сегментирования

исключение мутирования данных

поддержка различных механизмов серриализации реплик (синхронная и асинхронная)

реализация агрегатной модели сегментов транзакций (агрегируют широковещательные таблицы – “медленно меняющиеся таблицы” – 99% такого набора локализованы внутри сегментов не требуют двухфазной фиксации и более сложных протоколов – “Sharding”)

поддержка изолированных транзакций в драйвере пула соединений на уровне промышленных стандартов во все стороны: на границе перед серверами приложений,

кластеризация пула драйверов соединений

обеспечение неизбежной конфликтности и исключительной минимальной достаточности данных в модели данных.

Требование №6. Соответствие стандартам HIPAA и PCI-DSS. Это стандарты безопасности и соответствия уровню решаемых задач.

На этом технические требования к СУБД завершены.

На выбор СУБД в ТЗ должны влиять.

- ***Качество и полнота документации.*** К сожалению, не все системы имеют полную и подробную документацию.

- ***Локализация.*** Возможность использования национального языка (языков) не во всех системах реализована полностью, или не все производители поддерживают ISO локализации.
- ***Модель формирования стоимости***. Как правило, производители СУБД используют определенные модели формирования стоимости. Например, стоимость одного и того же продукта может существенно изменяться в зависимости от того, сколько пользователей будет с ним работать, или сколько одновременных соединений. Например, Oracle остается “свободным”, пока вы остаетесь в границах пула соединений. Но стоимость DBA может быть дорогой. С другой стороны, PostgressPro может оказаться значительно дороже из-за корпоративных битв. Специалисты Cassandra окажутся за пределами доступной стоимости.
- *Стабильность производителя и история – важнейший фактор. Не следует доверять недавно появившимся (системам, которые моложе 10 лет), хоть и хорошо капитализированным брендам.*

B) Требование к операционным системам и эластичности.

Операционные системы. Требования.

Используемые операционные системы - реального времени (real-time operating system, *RTOS*) — тип специализированной операционной системы, основное назначение которой — предоставление необходимого и достаточного набора функций для проектирования, разработки и функционирования систем реального времени на конкретном аппаратном оборудовании.

ОС или или инфраструктура должна предоставлять прозрачно за счет встроенных средств или специально комбинированный подход к балансировке нагрузки вычислителя и подключения к пулу портов: Round Robin для равномерного распределения общей нагрузки между серверами, Least Connections для обеспечения более эффективной балансировки внутри кластера серверов с разной производительностью и IP Hash для обеспечения сохранения состояния между клиентом и сервером, когда это необходимо.

Комбинированный подход позволяет адаптировать балансировку нагрузки к конкретным потребностям приложения и инфраструктуры.

Масштабируемость инфраструктуры ОС должна также масштабировать алгоритм балансировки нагрузки. Выбранный алгоритм должен поддерживать рост приложения.

Балансировщики должны автоматически масштабировать и перераспределять запросы на серверы. Это существенно упрощает управление нагрузкой и обеспечивает отказоустойчивость.

Окончательный выбор конкретного алгоритма балансировки нагрузки зависит от множества факторов, включая тип приложения, производительность аппаратуры серверов, требования к сохранению состояния и устойчивость к сбоям.

Мы специально не формируем требования к верхнему уровню – UserLayer предоставления сервиса и услуг, ППО должен предоставить разработчик Технического задания.

Так как мы сознательно принимаем необходимость “мягкой связи” между тем, какие аспекты мы контролируем жестко для того, чтобы мы могли отдать варианты реализации уровня ППО расширенно, но, контролируемо. То есть мы не будем определять фреймворк, стеке языков и технологий уровня ППО конечной ГИС, ориентируясь на то, что мы делаем выбор OPENSOURCE решения. В рамках этой парадигмы и методики, ППО которое накладывается поверх СУБД и ОС, вторично, и не становится важной и критической частью.

В связи с специфичностью ГИС, мы можем на этапе согласования технического задания потребовать числа СУБД.

В документе может быть приведен исключительно эскизный дизайн требований, относительно которых будет осуществляться выбор.

C) Слой ППО.

1) Прикладное программное обеспечение – общие требования.

A) Дизайн требований верхнего уровня ППО.

Звенность - client-server архитектуры

Версионность – неотказуемость на любой версии патча.

Контейнерность – поставка решений и модулей системы в упакованных оркестрированных технологиях в целях автоматизации их развёртывания, масштабирования и координации в условиях кластеров (kuber rkt Docker)

Кластеризация доменов приложений и данных.

Разнесенная интеграция приложений.

Событийность

Мульти-сервисность.

API.

2) Дизайн ППО .

Контейнеры с nginx (openresty) и UI статикой (js + css).

Реестры приложений, Spring Cloud, PAYARA, APACHE, GLASSFISH.

Микросервисы API шлюз взаимодействия от клиента к серверу.

Микросервисы приложений, отвечающие за доставку приложений к целевым сервисам.

Микросервисы отправки уведомлений (email, push-нотификации и др.).

Микросервисы моделей. Отвечает за информацию о типах, шаблонах нумерации и о матрицах прав.

Микросервис для хранения истории. Подписан на события в системе и сохраняет информацию о них в БД.

Микросервис для управления процессами.

Приложение Keycloak для аутентификации в системе.

Выделенная система индексации метаданных и контента документов.

Микросервис UI конфигураций. Отвечает за формы, журналы, UI действия, темы, дашборды, локализацию, иконки, конфигурацию меню.

Микросервис для интеграции с внешними системами (SAP, 1C, Rabbit MQ и тд.).

Микросервис для преобразования (трансформации) контента.

Микросервис для обеспечения хранения файлов в системе в определенное файловое хранилище.

Распределенное key-value хранилище для координации приложений Citeck между собой.

Приложения обмена сообщениями между микросервисами.

Хранение данных/детализация.

НЕИЗМЕННОСТЬ ИСТОРИИ – гарантии принципа однократной записи

ГАРАНТИИ временных меток записи и их неизменности

Ключевание записей, от которых требуется поддержки вещественности

Процедурность СУБД.

Документированность.

Основная используемая реляционная база данных – **ORACLE DBMS**

Хранение метаданных поддерживается в любой системе через адаптеры в: PostgreSQL, TARANTOOL**,** Oracle DB, MS SQL, Mongo DB, Alfresco ECM, SAP HANA.

Хранение документов в документно-ориентированной **DBMS**

Хранение событий в граф-ориентированной БД – **TARANTOOL** или **ORACLE BIGDATA**

Для хранения документов может быть использована БД **Mongo DB, PostgreSQL, Alfresco ECM, S3** - совместимое хранилище или внешняя ECM система через адаптер (например, разработан адаптер к системе OpenText).

Помимо баз данных используется также прямая запись в файловую систему для приложений Alfresco (Content Store), Zookeeper, Rabbit MQ и Solr**.**

API/Общее требования.

API для организации простого и масштабируемого общения между потребителем информации и источником данных.

Классический REST API и оптимизированный типизированный подход GraphQL сервер отдает только те данные, которые нужны клиенту с предсказуемой типизацией.

API для доступа к данным в системе для всех потребителей (Браузер, Мобильное приложение, Система построения отчетов, Индексирование данных, микросервисы, интеграция и т.д.).

Поддержка загрузки данных из связанных сущностей. Например, если документ ссылается на поручение, то, имея идентификатор поручения, мы можем получить любой атрибут связанного с поручением события, документа и т.п.

Оптимальность. Загружаются и вычисляются только те атрибуты, которые нужны потребителю в данный момент.

Простота в разработке — разработчик источника данных описывает все атрибуты, которые могут запросить потребители вне зависимости от сложности их вычисления. Потребитель в запросе указывает только те атрибуты, в которых он заинтересован.

Простота поддержки —API структурно предусматривает отсутствие версионности, т.к. в любой момент можно добавлять новые атрибуты, не трогая старые атрибуты API.

Тип получаемых данных полностью описывается запросом. Из источника данных мы возвращаем атрибуты с любым типом, а Records API приводит их к нужному для потребителя.

Вычисляемые атрибуты. Возможность добавлять атрибуты, которые не хранятся в БД или любом другом хранилище, а вычисляются на основе существующих.

Поддержка объединения атрибутов из разных источников. Например, можно написать источник данных, который часть атрибутов будет брать, например, из alfresco, а часть – из внешней БД, объединяя их по идентификатору.

Команды консоли — декларативное описание действия, которое нужно сделать на удаленном сервисе или локально.

Кластеризация — разворачивание нескольких инстансов приложения для обработки большой нагрузки и повышения отказоустойчивости системы.

Логически система работает одинаково вне зависимости от количества инстансов.

Инстансы приложения в кластере, как правило, работают с одними и теми же хранилищами данных (БД, файловая система).

Кластеризация нужна для отказоустойчивости и распределения нагрузки по CPU, RAM и сети.

Для разворачивания кластера микросервисов несколько инстансов приложения.

При старте все приложения регистрируются, указывая при этом свой **IP, HOST** и **PORT**.

Пример - Балансировка нагрузки на порту - **gateway** - приходит запрос от пользователя за некоторым ресурсом, **gateway** по информации в **registry** определяет список инстансов нужного приложения. После этого запрос уходит на один из инстансов по алгоритму, например **round-robin, registry** регулярно проверяет регистрацию стека приложений. Если приложение перестало отвечать или отсутствует, оно должно быть перегужено контейнером перегрузки.

Таймеры позволяют отложить выполнение любых действий во времени.

Любой сервис в системе отправляет в процессы команду **«E0»**, указав время срабатывания таймера и команду, которая должна при этом выполниться.

Когда наступает время срабатывания таймера, микросервис или приложение отправляет зарегистрированную в п.1 **команду на целевой сервис**. Целью команды может быть любой микросервис или приложение.

Примеры команд: «Отправить email», «Выполнить скрипт», «Завершить этап/задачу в процессе» и др.

2. Архитектуры

Отказоустойчивость. При выходе из строя любого узла системы работоспособность должна сохраняться.

Сохранность данных. При полной или частичной потере данных на одном из узлов хранилища данные в системе не должны быть потеряны.

Горизонтальное масштабирование. При росте количества процессов должна быть возможность горизонтального расширения за счет увеличения количества узлов в кластере, чтобы избежать деградации времени выполнения запросов с увеличением времени жизни системы. Старые процессы, которые уже давно завершились, не должны оказывать негативное влияние на активные.

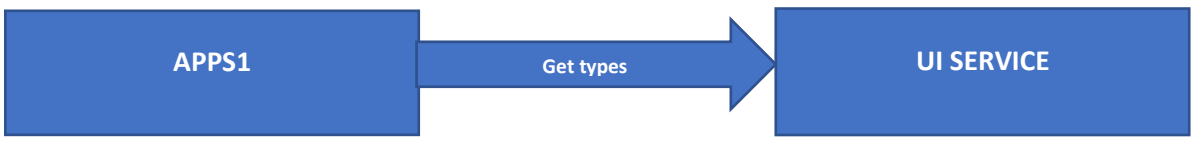
ПРОЦЕССЫ.

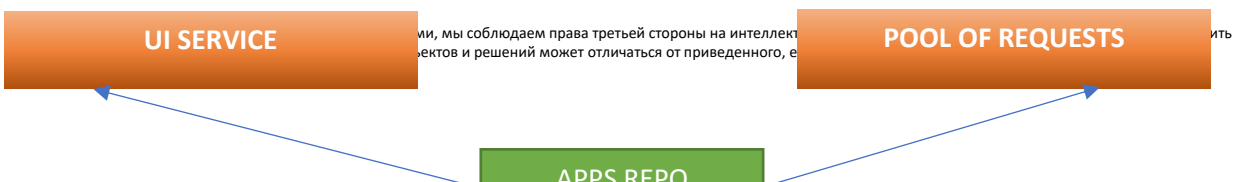
В качестве движка для процессов применяется open-source решение **Camunda (или подобное)**.

В качестве редактора для создания и редактирования процессов наличие low-code редактор (пример) на основе библиотеки bpmn-js; интеграция с экосистемой ролей, форм, статусов и др. - для разработки стандартных процессов не требуется участие программистов - Графический редактор процессов.

Приложения, Артефакты и События.

Приложения, Артефакты и События реализованы на микросервисной оркестризации - “артефактами управляют микросервисы, микросервисами управляют артефакты, события управляют трубой вызовов”.





GET SERVICE

СХЕМА №1. Оркестризация приложений.

Примерная логика.

Доставка артефактов при старте системы происходит в несколько этапов:

1. Микросервис **apps№1**, «увидев» новый микросервис в сети, загружает из него список типов, в которых он потребовал.
2. Получив типы, **apps №1** рассылает на все остальные микросервисы запрос на получение артефактов с данными типами.
3. Получив нужные артефакты со всех микросервисов, **apps№1** проверяет, изменился ли их контент с прошлого деплоя. Если изменений нет, то алгоритм заканчивает свою работу. Если изменения есть, то происходит redeploy новых данных в целевой микросервис.

Приложения выгружают из системы артефакты “по требованию” в формате **deployment архива war, jar, zip, exploded** и «на горячую» систему уровня предоставления сервиса.

Артефактами являются формы, журналы, типы, матрицы прав, действия, описания процессов и многие другие сущности в системе – единица расширения данных.

Микросервис – единица расширения приложения.

Реестр версионности микросервисов, они доставляются до целевого порта по запросу. Контекст, изменяемый в зависимости от запроса артефактов, в системе неизменяемый, и при любом изменении артефакта всегда создается новая версия, а старая сохраняется в списке версий.

Зависимости.

Центральной частью системы должны быть абстракции **<DATA SOURCE>**, в качестве которого может выступать любой источник данных в любом из микросервисов.

Для добавления новых источников достаточно реализовать определенный интерфейс и данные из этого источника могут быть свободно интегрированы со всей экосистемой ППО (их можно отображать в журнале, редактировать и просматривать через формы, отправлять по ним уведомления, запускать по ним процессы и т. д.).

Любой **<DATA SOURCE>** в общем случае может общаться со следующими сервисами:

model для автонумерации, делегирования полномочий и получения индивидуальных настроек прав;

content для работы с контентом;

apache zookeeper для работы с реестрами артефактов.

Общение с источниками данных построено на базе универсального API

Зависимости от **<DATA SOURCE>** по микросервисам:

uiserv загружает атрибуты для фильтрации UI действий по заданным в конфигурации условиям;

notifications загружает атрибуты для заполнения шаблона уведомления;

history загружает атрибуты для сохранения записи в истории;

process загружает и меняет атрибуты в ходе выполнения BPMN процессов.

2. Почти все микросервисы работают с **Rabbit MQ** (события и команды) и с **Zookeeper** (события, конфигурация ППО, реестры типов, аспектов, настроек прав, шаблонов нумерации, распределенные блокировки, внешние миксины);

UI (мобильный и браузерный) зависят от **gateway** (шлюз для доступа в систему) и от **uiserv** (микросервис с UI конфигурациями);

gateway зависит от **model** для получения информации по пользователям и группах, в которых они состоят. Эта информация используется для формирования JWT-токена с последующей отправкой его в остальные микросервисы для аутентификации и авторизации;

integrations зависит от внешних систем, с которыми настроена интеграция.

content зависит от места хранения контента (Alfresco или S3, 1С, и другими).

Apache Solr зависит от источников данных для индексации контента и атрибутов.

Система ППО построена на основе микросервисной событийно-ориентированной архитектуры. В качестве основного способа взаимодействия микросервисов используется обмен сообщениями через очередь сообщений (MQ).

Платформа поддерживает работу с различными источниками данных без необходимости копирования данных во внутренний репозиторий.

Для работы с данными используется должен использоваться документированный собственный язык запросов Records API. Перечень источников данных должен быть легко расширен.

Основные компоненты микросервисной системы. Примерная логика.

- **DAO** – сервисы работы с контентом и метаданными. В качестве источников данных могут использоваться Базы данных, Alfresco Content Service, 1C, SAP и другие;
- **Process services** – сервисы управления бизнес-процессами, поддерживаются нотации моделирования бизнес-процессов BPMN и CMMN;
- **Application services** – сервисы управления приложениями, их версионностью и деплойментом;
- **Data services** – сервисы работы с данными, в том числе валидации данных и их индексации;
- **Integration services** – интеграционные сервисы, включая юридически значимый документооборот;
- **API gateway** – API шлюз, используется в том числе для запросов от пользовательского интерфейса (WEB и мобильного);
- **Business logic services** – сервисы бизнес-логики и конфигурации.

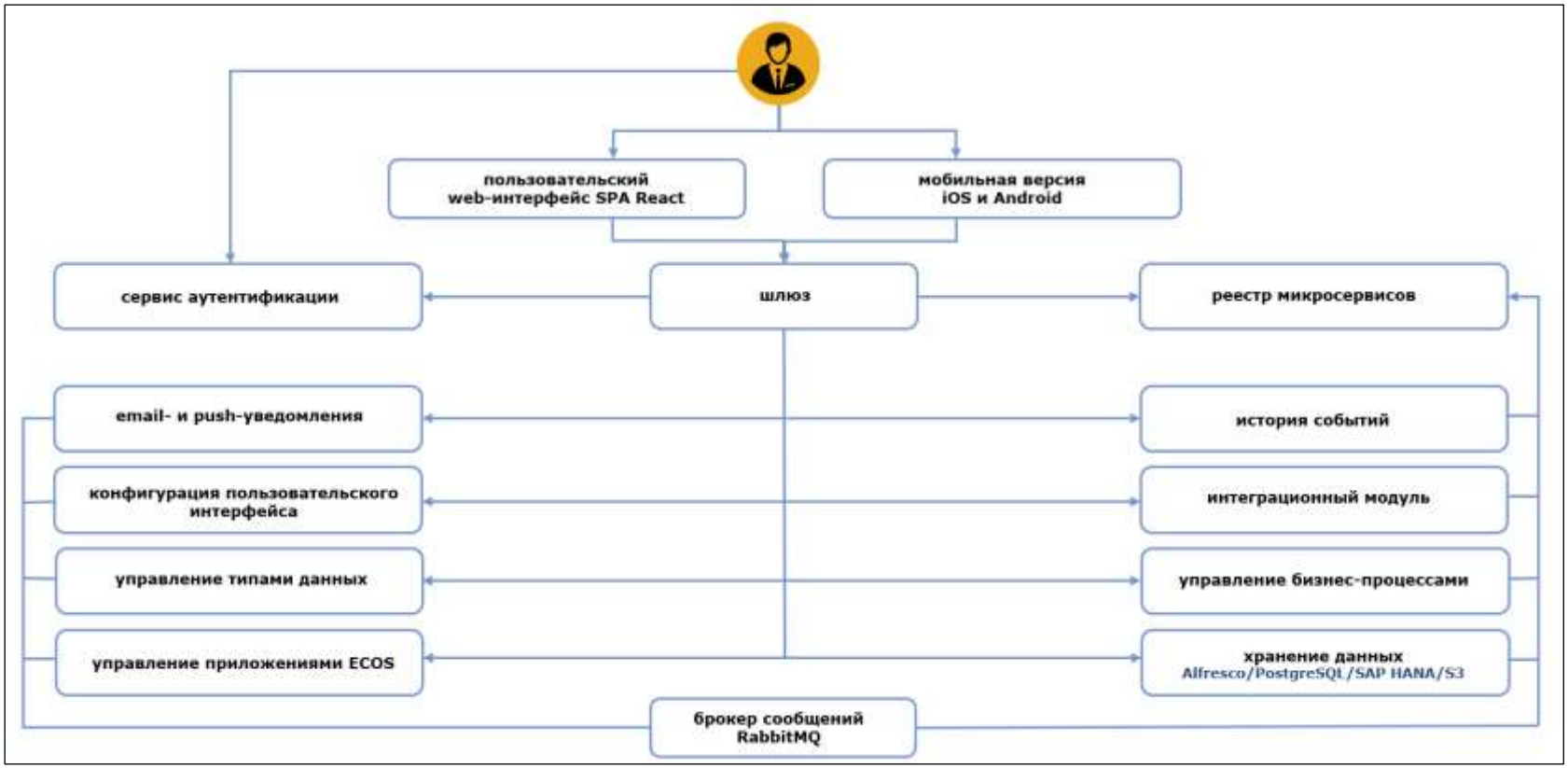
Интерфейс системы на базе фреймворков – **React, Vue, Node,** которые раздаются через NGINX и работают под управлением веб-серверов – **full stateless web apps (с сохранением состояния).**

Мобильный интерфейс на базе фреймворка **React Native**, **Vue** и пр., поддерживается адаптивная трансформация для типа мобильных браузеров.

Запросы от пользовательских интерфейсов маршрутизируются через **NGINX** и **API шлюз**.

В запросе к **NGINX отправляются идентификаторы приоритета на обслуживание очередей, полосы канала на ответ, идентификаторы приложений.**

На уровне предоставления порта пользователю развертывается инфраструктура контейнеризации **Docker / Kubernetes** – СХЕМА №2.

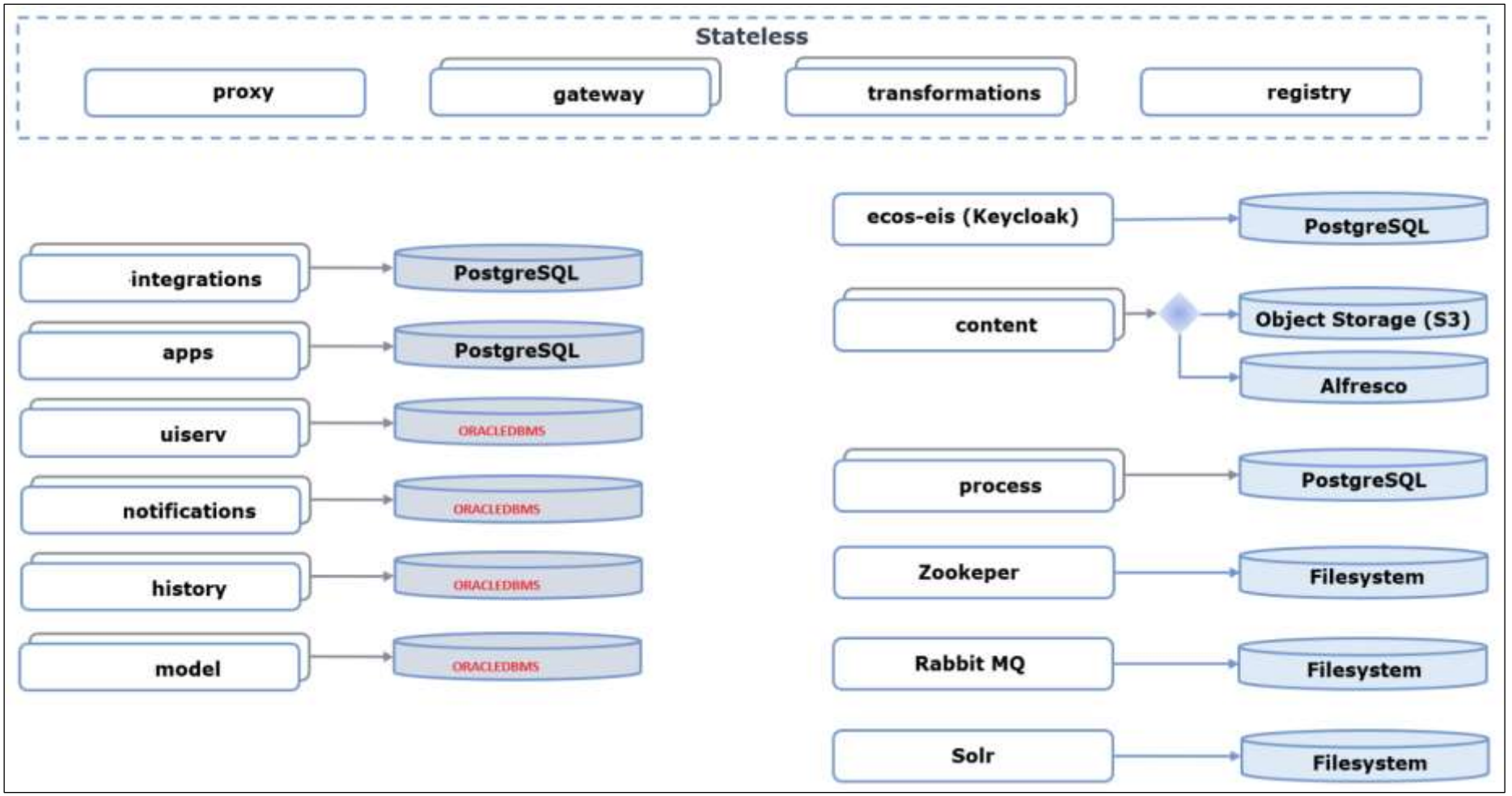


Взаимодействие клиента с сервером.

Примерная логика.

При поступлении запроса от клиента **nginx**, квалифицирует токен доступа на **Keycloak** для аутентификации через протокол **OpenID Connect**. **Keycloak** может предложить окно ввода логина/пароля или сразу выдать пользователю токен, с помощью которого он сможет зайти в систему (SSO). После успешной аутентификации пользователь перенаправляется на страницу, с которой его отправили в keycloak.
После того, как запрос прошел дальше, **gateway (шлюз)** смотрит на URL запроса и по нему решает, какой именно микросервис должен его обработать (например, запрос **/eodel/api/records/query** должен уйти в **model**).

Для получения IP адреса и порта целевого микросервиса **gateway** обращается в **registry** за нужной информацией и, получив её, отправляет запрос дальше.



МИКРОСЕРВИСЫ.

Перечень микросервисов.

Компонент	Описание
proxy	Контейнер с nginx (openresty) и UI статикой (js + css).
registry	Реестр приложений и сервер Spring Cloud конфигурации.
gateway	Микросервис реализует API шлюз взаимодействия от клиента к серверу.
serv-apps	Микросервис приложений Citeck, отвечающий за доставку приложений Citeck к целевым сервисам.
serv-notifications	Микросервис отправки уведомлений (email, push-нотификации и др.).
serv-model	Микросервис моделей. Отвечает за информацию о типах, шаблонах нумерации и о матрицах прав.
serv-history	Микросервис для хранения истории. Подписан на события в системе и сохраняет информацию о них в БД.
serv-process	Микросервис для управления BPMN процессами.
serv-eis	Приложение Keycloak для аутентификации в системе.
alfresco	Open-source ECM система, которая может использоваться для хранения контента и метаданных документов в системе (один из вариантов реализации).
Apache Solr	Система индексации метаданных и контента документов.
serv-uiserv	Микросервис UI конфигураций. Отвечает за формы, журналы, UI действия, темы, дашборды, локализацию, иконки, конфигурацию меню.
serv-integrations	Микросервис для интеграции с внешними системами (SAP, 1C, Rabbit MQ и тд.).
serv-transformations	Микросервис для преобразования (трансформации) контента.
serv-content	Микросервис для обеспечения хранения файлов в системе в определенное файловое хранилище.
zookeeper	Распределенное key-value хранилище для координации приложений Citeck между собой.
Rabbit MQ	Приложение для обмена сообщениями между микросервисами.

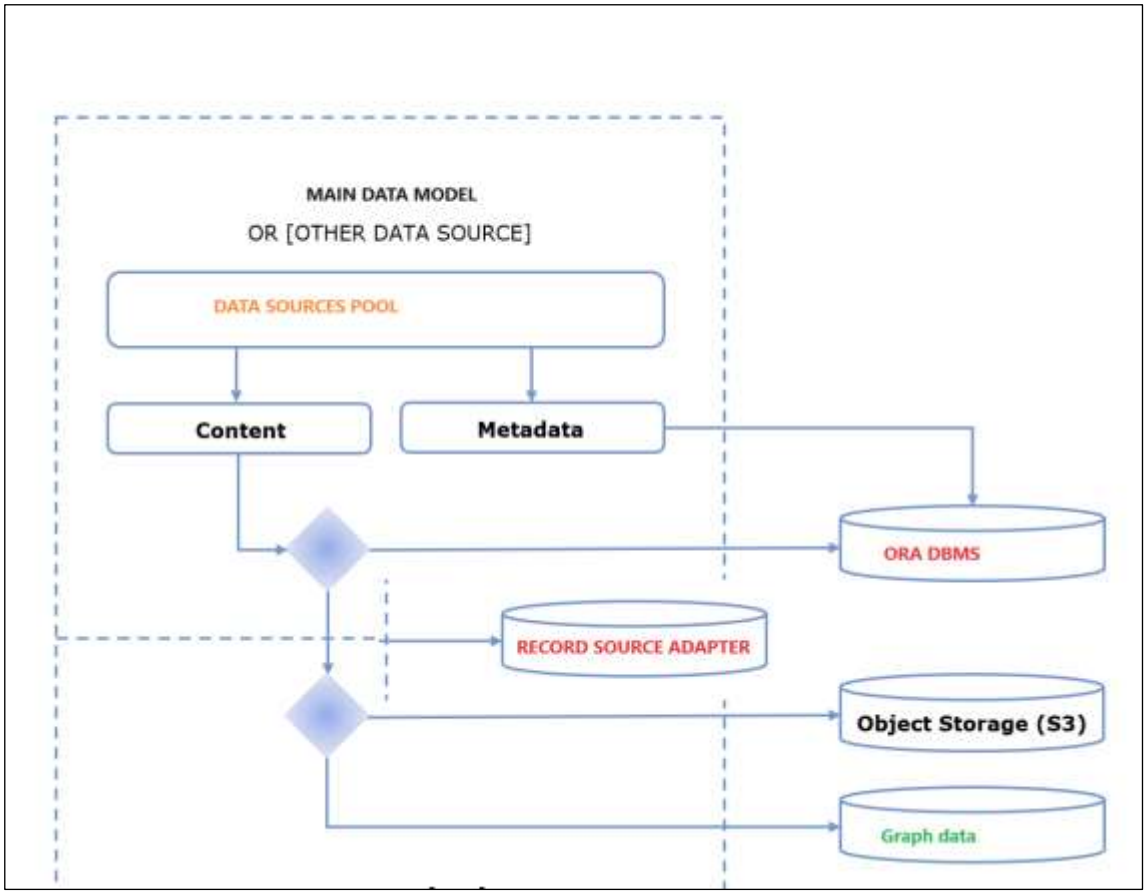


СХЕМА №4. МОДЕЛЬ ХРАНЕНИЯ И ПРЕДОСТАВЛЕНИЯ ДАННЫХ.

1. Основные используемые реляционные базы данных – **ORACLE**, **PostgreSQL**.
2. Хранение метаданных и их перераспределения по хранилищам поддерживается в **ORACLE или иной** любой системе через адаптер (record source). Минимальные адаптеры: **PostgreSQL, Oracle DB, MS SQL, Mongo DB, Alfresco ECM, SAP HANA.**
3. Для хранения документов может быть использована БД **PostgreSQL, Alfresco ECM, S3, MongoDB** -совместимое хранилище или внешняя ECM система через адаптер (например, разработан адаптер к системе OpenText).
4. Для хранення графов используется **TARANTOOL** или **ORACLE BIGDATA**
5. Помимо баз данных используется (также) прямая запись в файловую систему для приложений **Alfresco (Content Store), Zookeeper, Rabbit MQ и Apache Solr**.

Команды - декларативное описание действия, которое нужно сделать на удаленном сервисе или локально.

Пример команды для выполнения задачи:

```
{
    "id": "123e4567-e89b-12d3-a456-426655448474",
    "time": "2019-01-01T01:01:01.952Z",
    "target": "1proc",
```

```
    "actor": "system",
    "source": "alfresco:a8aae115-e2c5-418c-a261-61ed4ce94ba8",
    "type": "activity.complete",
    "config": {
        "activityId": "2143",
        "processId": "cmmn$c7a57bf4-43b8-4c78-a154-7551aac0152d",
        "attributes": {
            "outcome": "Done"
        }
    }
}
```

Команды в качестве транспорта используют очереди RabbitMQ. Использование команд возможно как в синхронном, так и в асинхронном режиме.

Целью команд могут быть:

1. Тип сервиса (proccess, apps, uiserv, alfresco и др.). Команду исполняет один из инстансов сервиса.
2. Инстанс сервиса (у каждого типа сервиса может быть много инстансов).
3. Все типы сервисов supports single or широковещательные команды. Сервис-источник команды отправляет широковещательную команду в RabbitMQ и её обрабатывают все сервисы, которые в данный момент активны. Сервис-источник команды отправляет singletype команду в RabbitMQ и её обрабатывают те сервисы, которые в данный момент на нее подписаны активны.

ШАРДИНГ и ЗАПИСИ.

Записи.

Примерная логика.

Пользователь формирует атрибуты документа и отправляет результат на сервер.

1. Данные с формы принимает **gateway** и на основе правил, которые настроил администратор, выбирает **id приложения**, которому нужно делегировать запрос создания.
2. После того, как целевой сервис успешно выполнил операцию, **gateway** возвращает клиенту **ID новой сущности**, в котором содержится **ID приложения**, которое было выбрано согласно правилам в п.2. Записи создаются в UNC/ Пример:

fresc2/nodes@workspace://SpacesStore/123e4567-e89b-12d3-a456-426655440000

apps3/nodes@admidnuser://SpacesStore/123e4567-e89b-12d3-a456-668782GHJVSv

Сквозной поиск

1. Получив запрос на поиск данных в системе, **gateway** на основе располагаемых данных отправляет N запросов на поиск по разным шардам.
2. Получив результаты запросов, **gateway** объединяет их результаты и отдает получившийся список клиенту. Клиентом могут быть apps, artifacts, services

Возможный вид списка сущностей, которые вернутся в результате поискового запроса:

```
Apps1/nodes@news://SpacesStore/2650bee2-43e9-4768-8e99-e1f86cb56151
Apps3/nodes@WS://SpacesStore/84e5ead8-f26f-48ed-ab7e-34f7d52db6e2
Serv2/nodes@datastore1://SpacesStore/2ceb8a4f-583a-458e-a94e-e17ae003ca3c
Timer2/admin@mashine002://SpacesStore/cb31f7ae-c27f-42f5-8887-81f337663686
```

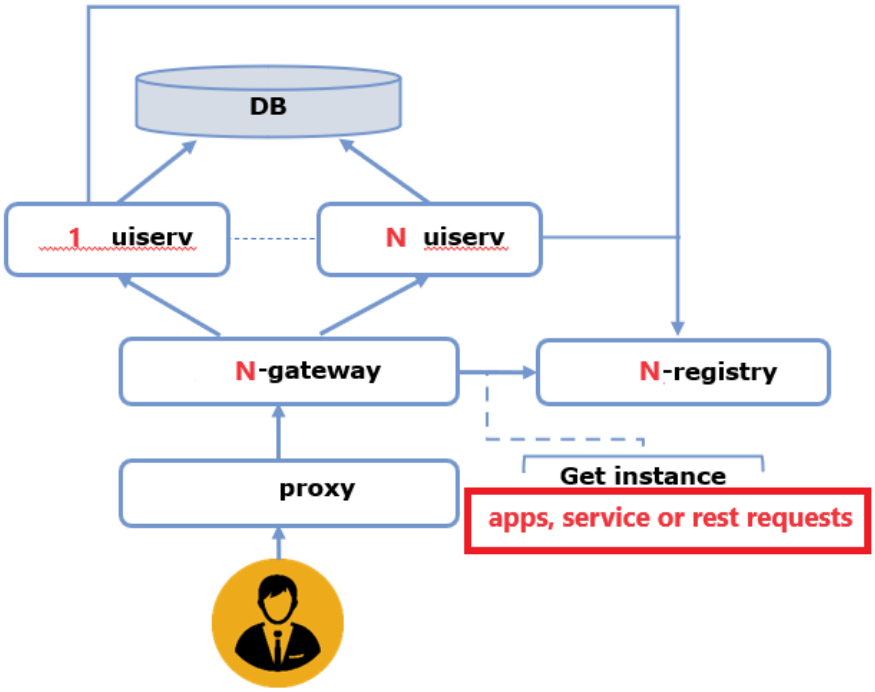
Кластеризация — разворачивание нескольких инстансов приложения для обработки большой нагрузки и повышения отказоустойчивости системы.

Примерная логика.

Логически система работает одинаково вне зависимости от количества инстансов приложения.

Инстансы приложения в кластере работают с одними и теми же хранилищами данных (БД, файловая система).

Кластеризация нужна для отказоустойчивости и распределения нагрузки по CPU, RAM, СХД и сети.



1. Для разворачивания кластера микросервисов загружается несколько инстансов приложения.
2. При старте все приложения регистрируются в **registry**, указывая при этом свой **IP, HOST** и **PORT**.
3. Балансировкой нагрузки занимается **gateway**. Когда приходит запрос от пользователя за некоторым ресурсом, **gateway** по информации в **registry** определяет список инстансов нужного приложения. После этого запрос уходит на один из инстансов по алгоритму **round-robin или иным косинусным алгоритмам**.

4. **registry** регулярно проверяет health-check **и перегружает их**.
5. **Регламентная перегрузка приложений и очистка “песочницы кучи”.**

Криптография, шифрование каналов и применение ЭЦП.

ТЗ должно поставлять описания и процедуры криптографии, на всех участках решения в соответствии документаций применимого крипто провайдера.

Применение крипто-провайдеров для формирования закрытого контура сегмента ГИС должно осуществляться без применения VPN.

Эти решения приводятся в специальном не публичном разделе ТЗ.

3. Требования прототипам аппаратуры вычислительной среды (серверов).

Таблица 3.1 Терминология.

Платформа	Платформой в документе называется совокупность программных компонентов гриддов, включающих серверы сети, Prods Servers или другие компоненты (как: серверы сообщения, брокеры сообщений и т.д.), вместе обеспечивающие функциональность серверной части системы.
Вычислительная инфраструктура	В рамках документа под вычислительной инфраструктурой (или просто «инфраструктурой») подразумевается совокупность серверных аппаратно-программных средств, выделяемых каналов в системах передачи данных, выделяемых для работы системы
Инсталляция	Инсталляция серверов Prod, приложения и проч.
КУ	Конечное устройство, подключаемое к систем
Сервер сети	Сервера в составе платформы осуществляющие обмен данными оконечных устройств на сервера ProdServer
Виртуальная машина	Создаваемое гипервизором изолированное программное окружение, эмулирующее аппаратную среду ЭВМ.
Гипервизор	Программная система управления виртуальными машинами. Эмулирует несколько аппаратных компьютеров (guest-платформ) на базе одного физического компьютера (host-платформы).

RDBMS	Реляционная система управления базами данных
Программный брокер сообщений **Kafka**	Программный брокер сообщений
Redis	Remote Dictionary Server – хранилище ключ-значение, располагаемое в памяти
RAM	Оперативная память, ОЗУ
ROM	Постоянное запоминающее устройство, ПЗУ, дисковое хранение данных

Состав вычислительных задач платформы

Нагрузка, которую создают приведенные в таблице 3.2 сервисы и приложения зависят от интенсивности нагрузки со стороны конечных устройств и подключенных систем. В техническом задании объем необходимых вычислительных ресурсов должен планироваться в зависимости от количества конечных устройств, осуществляющих передачу данных в зависимости от структуры инфор-трафика и учитывать его. В таблице 3.2 приведен пример прототипирования представления сервисов и приложений, платформы (состав сервисов поставляется разработчиком (исполнителем) Технического задания в зависимости от типа планируемой платформы и, кроме того, зависит от задач, решаемых в конкретной инсталляции.

	Таблица 3.2. Прототипирование состава прикладных вычислительных задач на платформу. Примерны план.			
№	Узел (ВМ)	Роли в кластере Kubernetes	Роли в платформе	Состав сервисов, приложений
1	Мастер-узел	Master-node	Сервер сети Сервисы сервера сети	Задачи управления кластером Kubernetes: Kubernetes Master, Kubernetes API Server, coredns
2	Рабочий узел	Worker Node		Сервисы сервера сети1: auth-srv-core, devreg-srv-core, firmware-srv-core, Prod-adpt-core, mpu-srv-core, nwk-srv-core, routing-srv-core, semtech-gtw-core, public-rest-core, support-rest-core, Prod-console, support-tools-web, tafel-dummy-srv, userspace
3	Сервисный узел	Service node		Серверные приложения, Oracle, PostgreSQL, Kafka, Redis
4	Prod	Сервера логгирования «Prod Server»	Сервисы Prod Server: IEC104-Proxy, HDLC-Proxy-Core, HDLC-Proxy-Connector, HDLC_Processor, Discovery, Diagnostic, ControllableLoad, InstantValue,	

			TimeSync, JournalEventProcessor, JournalEventCore, DataIntegrity, Серверное приложение Redis.
5	Master (Prod) DataBase	Master DB	Серверное приложение DBMS
6	Monitoring	Monitoring	Серверные приложения мониторинга: Promettheus, Loki, Grafana
7	MASTER Decoder MASTER Encoder		Сервера wrap

The HSDLC IP core implements a controller for the High-Level Data Link Control (HDLC) and the Synchronous Data Link Control (SDLC) protocols.

В техническом задании нагрузки на платформу должны фиксироваться на тех инсталляциях, где разворачивается платформа.

Платформа должна включать в себя в качестве главных компонент: (1) сервера сети и (2) сервер обработки, сервера идентификации, сервера шифрования/дешифрования данных.

Программное обеспечение инсталляции может устанавливаться как на одиночном аппаратном сервере, так и на виртуальных машинах.

Прототипирование требований к инфраструктурам платформы

Для обеспечения заданного уровня качества сервиса сбора и обработки информационных запросов при развертывании платформы разработчиком должен быть разработан набор требований к производительности серверного оборудования, к параметрам каналов передачи данных и настойкам безопасности сетевого доступа.

Требования к настройкам доступа отличаются в зависимости от стадии и вида работ: установка и настройка ПО, эксплуатация платформы, сопровождение и техническая поддержка, обновление ПО **(описаны в следующей части документа**).
Для определения конфигурации серверного оборудования, требуемого для развертывания платформы, требования к производительности в Техническом Задании должны быть даны в терминологии IOPS, FLOPS.

Прототипирование производительности

Пример.

Дисковая подсистема платформы должна обеспечивать производительность выполнения операций чтения/записи согласно таблице 3.3.

Таблица 3.3. Прототипирование требования к производительности операций ввода/вывода систем постоянного {ранения Нагрузка	Производительность операций записи/чтения
Сервисный узел	5 IOPS/Гб
Узел MASTER DB	9 IOPS/Гб

Узел мониторинга	0,5 IOPS/Гб
Остальные нагрузки	2 IOPS/Гб
Дисковая подсистема платформы должна обеспечивать производительность выполнения операций чтения/записи согласно указанному в таблице.	

Прототипирование определений производительности

Таблица 3.4. Параметрами, которые приняты для определения способности серверного оборудования к обработке потоков данных, являются	Краткое обозначение	Единица измерения
Число виртуальных процессоров	vCPU	шт.
Объем оперативной памяти	RAM	Гбайт
Дисковое пространство	ROM	Гбайт
Производительность системы хранения	IOPS	Операций/сек.
Пропускная способность сетевых соединений		Гбит/с

Прототипы планирования требования к пропускной способности и резервированию сети.
Пример прототипа. При разнесении узлов платформы на разные физические машины пропускная способность соединений между ними должна быть не менее 10 Гбит/с.
Параметра значения 750 мс - снижение емкости сети, более 2400 мс – нарушение работы сети.

- скорость соединения: не менее 100 Мбит/с

- время кругового отклика (ping): не более 100 мс.

Дополнительные требования при развертывании сети.
Аппаратные интерфейсы и организация подсети - При установке платформы на отдельный сервер, на сетевом оборудовании и сервере необходимо предусмотреть три порта Ethernet с пропускной способностью не менее 1 Гбит/сек.:

1	Порт 1GE сервера	Основной порт
2	Порт 2GE сервера	Резервный порт
3	IPMI	Контроллер удаленного доступа

3.2. Требования к планированию вычислительных ресурсов.

Прототипирование требований к планированию вычислительных ресурсов.
Пример прототипа плана нагрузки.
Максимальные планируемые нагрузки на аппаратное обеспечение серверов, при котором обеспечивается заданный уровень качества обслуживания платформы является:
 Регулярная средняя загрузка CPU 80% утилизации процессорного времени;
 Достижение утилизации 60% оперативной памяти.

Дополнительные сведения о планировании вычислительных ресурсов.

Прототипирование среды виртуализации и сред вычислений

Требования представлению вычислительных ресурсов

Требования к числу виртуальных процессоров, памяти, постоянному хранению, требования к вычислительным ресурсам при развертывании в среде виртуализации вычислений при раскрываются как данные таблицы должны отражать примерный объем ресурсов, при котором обеспечивается загрузка CPU и RAM на 80 и 60%.
При развертывании платформы на виртуальных машинах предъявляются требования к прототипу вычислительной производительности и прототипам объемов хранения данных. В табличных представлениях и схемах должны быть приведены параметры исходя из опыта эксплуатации подобных платформ и соответствующих загрузке процессоров и памяти на 50%.
Пример.

Таблица 1.1 Требования к ресурсам виртуальных машин Количество КУ	Количество инсталляционных доменов	Вид ресурса	Объем	Загрузка ЦП / ОЗУ %
до yyy	*до zzz*	vCPU RAM ROM	XXX ядер XXX Гб XXX Гб	FFF% / VVV%
Примечания: 1) Соответствие дано примерно, для целей прототипа. Точное число определяется при планировании в ТЗ. 2) Должно быть передано как общее число виртуальных процессоров распределяется между виртуальными машинами (ВМ) в зависимости от типа трафика и вычислительной нагрузки.				

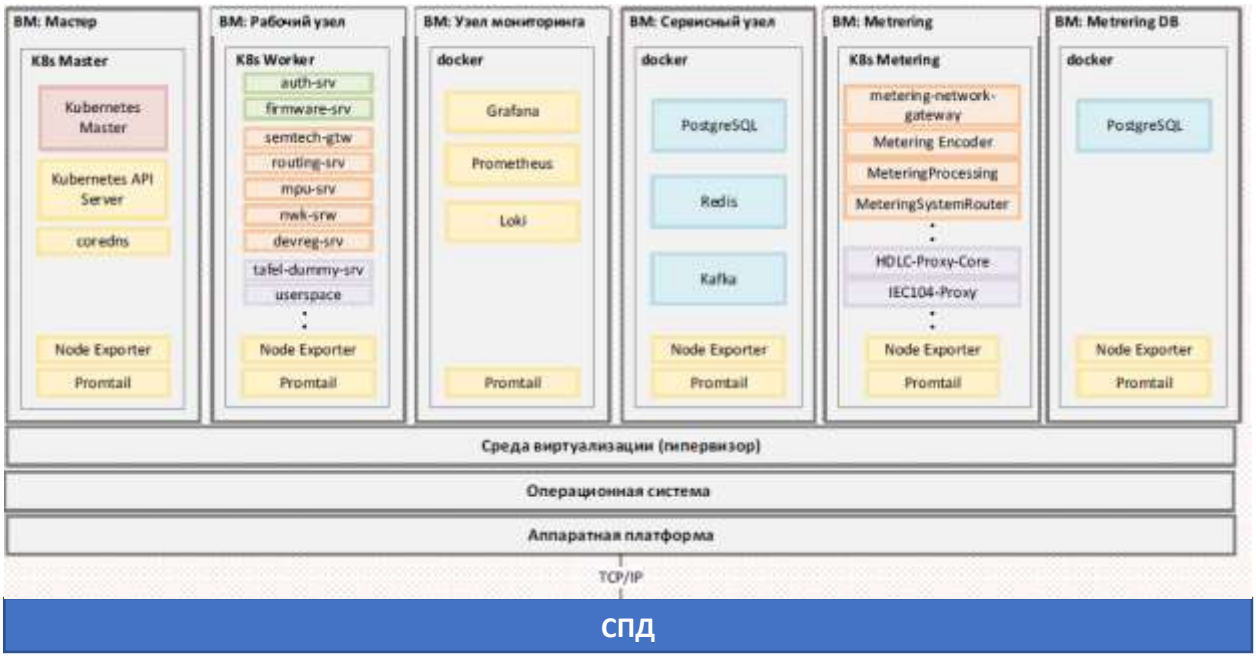
Прототипирование требований к ОЗУ

Таблица 2.3. Прототипирование требований к ОЗУ							
Число КУ	Мастер-узел	Рабочий узел	Сервисный узел	Prod	Prod DataBase	Monitoring	ИТОГО с запасом 10%
11	4	19	11	13	9	4	66
13	4	20	11	13	9	4	68

Прототипирование требований к числу виртуальных процессорных ядер

Число КУ	Мастер-узел	Рабочий узел	Сервисный узел	Prod	Prod DataBase	Monitoring	ИТОГО с запасом 10%
1000	2	8	7	7	5	4	37
5 000	3	9	8	7	5	4	40
10 000							

Схема №4. Прототипирование схем развертывания.



ID	SERVICE	**PLANNING**	
12005	Prod-data-integrity	**И|Э|П**	
12006	Prod-journal-events	**И|Э|П**	
12007	Prod-discovery	**И|Э|П**	
12008	Prod-hdlc-processor	**И|Э|П**	
12009	Планировщик	**И|Э|П**	
12011	Prod-diagnostic	**И|Э|П**	
12012	Prod-planner	**И|Э|П**	
12013	Prod-heat	**И|Э|П**	
12014	metring-gas	**И|Э|П**	
12015	water data integrity	**И|Э|П**	
12016	Iec 104 proxy	**И|Э|П**	
12017	Prod-file-export	**И|Э|П**	
12018	Prod-auth	**И|Э|П**	
12019	Prod-file-export	**И|Э|П**	
Мониторинг и журналирование			
12020	Grafana	**И|Э|П**	**Э**
12021	Grafana	**И|Э|П**	**Э**
12022	Prometheus	**И|Э|П**	**Э**
Взаимодействие с приложениями через HDLC proxy (Прод и т.д.)			
12050÷13000		**И|Э|П**	
Протоколы связи			
Примечание: * **Э** – этап эксплуатации; **И** – этап инсталляции (развертывания, установки) и при обновлении версий ПО; **П** – требуется для осуществления сопровождения в эксплуатации и технической поддержки			

** Список портов может меняться на разных инсталляциях и подлежит согласованию на этапе заключения договора

Прототипирование параметров инсталляции

№ п/п	Параметр	Значение	
1. Общие сведения			
1.1	Наименование инсталляции		**И**|**Э**|**П**
1.2	Расчетное количество КУ		**И**|**Э**|**П**
1.3	Тип сервера *Аппаратный / ВМ*		**И**|**Э**|**П**
2. Аппаратные ресурсы сервера			
2.1	Количество vCPU / ядер *Для ВМ – vCPU, для физ. – ядра*		**И**|**Э**|**П**
2.2	Объем оперативной памяти *Для ВМ – объем выделяемой для платформы RAM*		**И**|**Э**|**П**
2.3	Объем постоянного хранения (ROM) *Выделенный объем дискового хранения*		**И**|**Э**|**П**
2.4	ROM: вид резервирования *Схема применяемого RAID*		**И**|**Э**|**П**
2.5	ROM: Производительность *Скорость операций ввода/вывода*		**И**|**Э**|**П**
2.6	ROM: принятая стратегия резервного копирования		**И**|**Э**|**П**
2.7	Сетевые карты ethernet *Количество и скорость; IPMI, сведения о IPMI*		**И**|**Э**|**П**
3. Сведения о конфигурации			
Необходимо выделить подсеть не менее 16 адресов. Порты включить в одну VLAN. Каждой виртуальной машине назначается статический IP-адрес. *Переадресация (NAT)* Для порта IPMI организовать переадресацию с внешнего IP-адреса (любые порты по согласованию) на порты 80 и 443. Для виртуальных машин организовать переадресацию с внешнего IP (любые порты по согласованию) на порты 22.			
3.1	Состав виртуальных машин		**И**|**Э**|**П**
* **Э** – этап эксплуатации; **И** – этап инсталляции (развертывания, установки) и при обновлении версий ПО; **П** – требуется для осуществления сопровождения в эксплуатации и технической поддержки ** Список портов может меняться на разных инсталляциях и подлежит согласованию на этапе заключения договора			

Прототипы также должны быть даны в отношении к

Требований к сетевым соединениям

Требований к DNS именованиям

Требований к NTP.

Общие требования к Техническому заданию.

Техническое задание должно содержать разделы в соответствии с SyRS System Requirements Specification — общие требования к построению систем, их принципам и характеру взаимодействия пользователя с ними. (ГОСТ 34).

Требования к документированию и структуре Технического задания.

В ТЗ должны быть приведены:

Технические характеристики объектов автоматизации;

Состав и содержание работ по созданию системы;

Порядок контроля и приемки системы

Требования к документированию.

Техническое задание должно содержать требования к окружению безопасности, правилам, для приложений — требования к внешним интерфейсам, их функциям и юзабилити, ссылки на референсы документации вендоров;

Описание системных требований;

Обязательные элементы ТЗ.

Пояснительная записка к техническому (эскизному) проекту

Схема организационной структуры

Схема комплекса технических средств (КТС)

Схема функциональной структуры ИС

Схема автоматизации ИС

Перечень входных и выходных сигналов и данных ИС

Описание автоматизированных функций ИС.

Техническое задание должно давать пояснительные записки, частные технические решения проекта конечной системы.

Техническое задание должно содержать требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие.

В техническом задании должны быть приведены требования о распределении процессов и обязательным документам опытной, пред-промышленной и промышленной этапов эксплуатации, если они не могут быть указаны в другом документе.

Требования к минимальному составу и квалификации специалистов будут представлены Поставщиком решения, но в техническом задании должны быть указания на состав специалистов, если это требует проект конечной системы.

Условия использования материалов.

Интеллектуальная собственность

Мы сохраняем за собой права на интеллектуальную собственность - напр., авторское право на опубликованные, созданные нами объекты в наших документах. Копирование и/или обработка и/или распространение и/или другое (прочее) применение подобных объектов (например, имена, фотографии, рисунки, дизайны, таблицы, текстов) в других электронных или печатных публикациях со ссылкой на источник без нашего ясно выраженного разрешения запрещается. Если права на интеллектуальную собственность принадлежат третьей стороне, требуется письменное разрешение соответствующего правообладателя, они приводятся на условиях публичных лицензий их владельцев.

Если информация или какая-либо ее часть создана не нами, мы соблюдаем права третьей стороны на интеллектуальную собственность. В частности, содержания третьих сторон обозначены как таковые. Если Вы, тем не менее, заметили нарушение авторских прав на интеллектуальную собственность, мы просим сообщить нам об этом. Как только нам станет известно о нарушении закона, мы немедленно удалим эти содержания.

Если часть отдельных формулировок настоящих условий использования не соответствуют или не полностью соответствуют действующим юридическим положениям, остальная часть условий использования остаётся неизменной в отношении содержания и действительности.