МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский ядерный университет «МИФИ»
(НИЯУ МИФИ)

Чуркин Кирилл Юрьевич

Разработка персональной распределённой файловой системы

Москва 2023

Оглавление

1 Обзор существующих распределенных файловых систем хранения

1.1 NFS (Network File System)

Сетевая файловая система позволяет пользователям получать доступ к файлам, хранящимся на удаленных серверах. Кроме того, пользователи не осознают, что файлы находятся на удаленном сервере, и обращаются к ним как к локальным файлам. Кроме того, NFS также обеспечивает безопасность и масштабируемость.

Сетевая файловая система реализована с использованием клиент-серверной архитектуры. В этой архитектуре один компьютер выступает в качестве сервера, а другие компьютеры - в качестве клиентов. Клиентский компьютер выдает запросы на данные, которые должен выполнить серверный компьютер. Существует два типа запросов, выдаваемых клиентскими компьютерами: запросы на чтение и запись.

Клиентский компьютер отправляет запрос на чтение серверу для чтения данных. Кроме того, клиентский компьютер может также отправить запрос на запись серверному компьютеру для записи данных. Запросы на чтение и запись реализуются с использованием стандартных операций чтения/записи. Компьютер-сервер выполняет запрос, используя соответствующий протокол. Затем он возвращает данные на компьютер клиента.

Преимущества NFS:

- NFS проста в использовании и не требует никакой настройки.
- Обеспечивает безопасность своих пользователей. Протокол использует строгую аутентификацию для защиты от несанкционированного доступа к данным.
- NFS обладает высокой масштабируемостью и может интегрировать данные из удаленных местоположений в локальное хранилище.
- Включает аварийное восстановление.

Недостатки NFS:

- NFS не обеспечивает никакой синхронизации между клиентом и сервером. Это означает, что данные на удаленном компьютере не синхронизированы с данными на компьютере клиента.
- Время доступа к файлу зависит от скорости сети, т.к. когда пользователь хочет получить доступ к файлу на сервере, файл должен пройти по сети на его компьютер.
- Если сервер выходит из строя, доступ к файлам невозможен.
- NFS поддерживает только один хост, что не позволяет обмениваться файлами между разными серверами, поскольку структура сети NFS допускает только один сервер.
- Не поддерживает иерархическое управление хранилищем. Если пользователь хочет хранить данные централизованно, он не сможет сделать это с помощью NFS.

1.2 ZFS (Zettabyte File System)

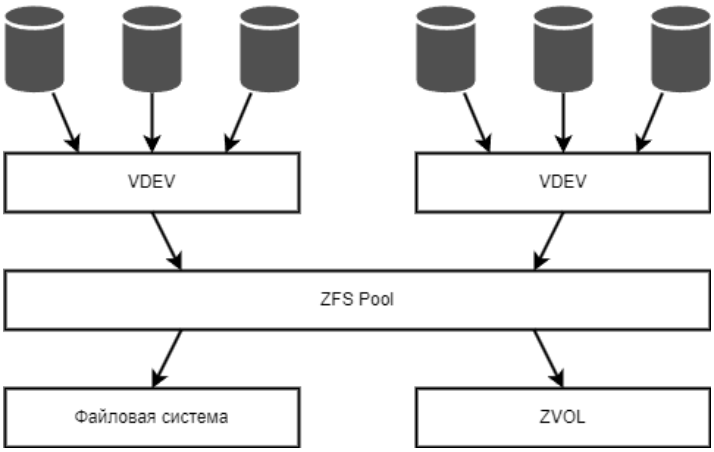
Эта система проектировалась с очень большим запасом по параметрам, на основе совершенно справедливого прогноза огромного роста данных, подлежащих хранению в распределенных системах в будущем.

Функции ZFS:

- Разделение системы хранения на пулы (Pooled storage).
- Копирование при записи (Copy-on-write).
- Снапшоты (Snapshots) системы.
- Верификация целостности данных и автоматическое исправление данных.
- Автоматическая замена на запасной диск (Hot spare).
- Максимальный размер файла 16 эксабайт.
- Объем хранения 256 квадриллионов зеттабайт.

1.2.1 Хранение в пулах (Pooled storage)

В отличие от большинства файловых систем, ZFS объединяет функции файловой системы и менеджера томов (volume manager). Это означает, что ZFS может создавать файловую систему, которая будет простираться по многим группам накопителей ZVOL или пулам. Более того, можно добавлять емкость в пул простым добавлением нового накопителя. ZFS сама выполнит партицию и форматирование нового накопителя.



1.2.2 Копирование при записи (Copy-on-write)

В большинстве файловых систем при перезаписи данных на то же физическое место носителя ранее записанные там данные теряются навсегда. В ZFS новая информация пишется в новый блок. После окончания записи метаданные в файловой системе обновляются, указывая на местоположение нового блока. При этом, если в процессе записи информации с системой что-то происходит, старые данные будут сохранены. Это означает, что не нужно запускать проверку системы после аварии.

1.2.3 Снапшоты (Snapshots)

Copy-on-write закладывает основу для другой функции ZFS: моментальных снимков системы (снапшотов). ZFS использует для отслеживания изменений в системе.

Снапшот содержит оригинальную версию файловой системы, и в «живой» файловой системе присутствуют только изменения, которые были сделаны с момента последнего снапшота. Никакого дополнительного пространства не используется. Когда новые данные записываются в «живую» систему, выделяются новые блоки для сохранения этих данных.

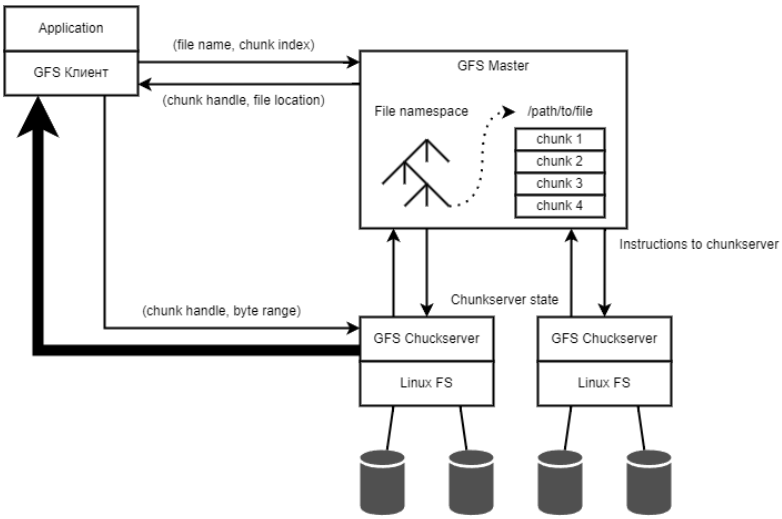
Если файл удаляется, то ссылка на него в снапшоте тоже удаляется. Поэтому снапшоты в основном предназначены для отслеживания изменений в файлах, а не для добавления или создания файлов.

1.3 GFS

GFS является наиболее, наверное, известной распределенной файловой системой. Надежное масштабируемое хранение данных крайне необходимо для любого приложения, работающего с таким большим массивом данных, как все документы в интернете. GFS является основной платформой хранения информации в Google. GFS — большая распределенная файловая система, способная хранить и обрабатывать огромные объемы информации.

Файлы в GFS организованы иерархически, при помощи каталогов, как и в любой другой файловой системе, и идентифицируются своим путем. С файлами в GFS можно выполнять обычные операции: создание, удаление, открытие, закрытие, чтение и запись. Более того, GFS поддерживает резервные копии, или снимки (snapshot). Можно создавать такие резервные копии для файлов или дерева директорий, причем с небольшими затратами.

Архитектура GFS



В системе существуют мастер-сервера и чанк-сервера, собственно, хранящие данные. GFS кластер состоит из одной главной машины мастера (master) и множества машин, хранящих фрагменты файлов чанк-серверы (chunkservers). Клиенты имеют доступ ко всем этим машинам. Файлы в GFS разбиваются на куски — чанки (chunk). Чанк имеет фиксированный размер, который может настраиваться. Каждый такой чанк имеет уникальный и глобальный 64 — битный ключ, который выдается мастером при создании чанка. Чанк-серверы хранят чанки, как обычные Linux файлы, на локальном жестком диске. Для надежности каждый чанк может реплицироваться на другие чанк-серверы.

Мастер отвечает за работу с метаданными всей файловой системы. Метаданные включают в себя пространства имен, информацию о контроле доступа к данным, отображение файлов в чанки, и текущее положение чанков. Также мастер контролирует всю глобальную деятельность системы такую, как управление свободными чанками, сборка мусора и перемещение чанков между чанк-серверами. Мастер постоянно обменивается сообщениями с чанк-серверами, чтобы отдать инструкции, и определить их состояние.

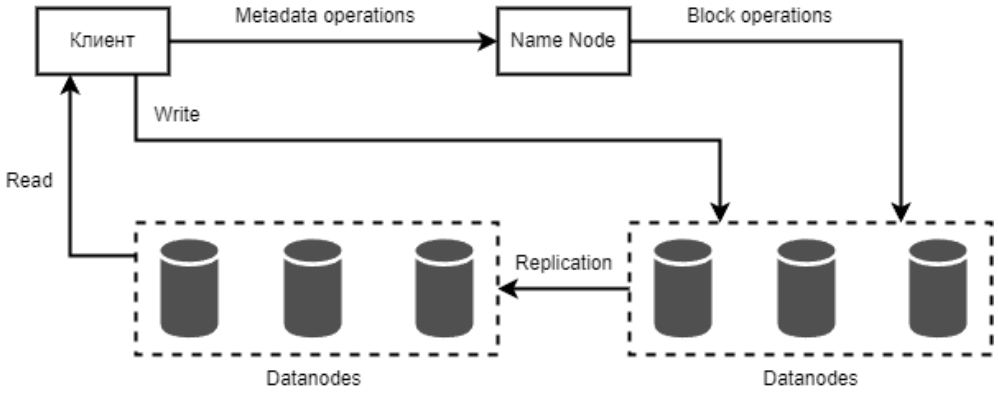
Клиент взаимодействует с мастером только для выполнения операций, связанных с метаданными.

Использование одного мастера существенно упрощает архитектуру системы. Позволяет производить сложные перемещения чанков, организовывать репликации, используя глобальные данные. Казалось бы, что наличие только одного мастера должно являться узким местом системы, но это не так. Клиенты никогда не читают и не пишут данные через мастера. Вместо этого они спрашивают у мастера, с каким чанк-сервером они должны контактировать, а далее они общаются с чанк-серверами напрямую.

1.4 HDFS

HDFS (Hadoop Distributed File System) — распределенная файловая система Hadoop для хранения файлов больших размеров с возможностью потокового доступа к информации, поблочно-распределённой по узлам вычислительного кластера, который может состоять из произвольного аппаратного обеспечения. Hadoop Distributed File System, как и любая файловая система – это иерархия каталогов с вложенными в них подкаталогами и файлами.

Архитектура HDFS



Управляющий узел (NameNode) – отдельный, единственный в кластере, сервер для управления пространством имен файловой системы, хранящий дерево файлов, а также метаданные файлов и каталогов. NameNode – обязательный компонент кластера HDFS, который отвечает за открытие и закрытие файлов, создание и удаление каталогов, управление доступом со стороны внешних клиентов и соответствие между файлами и блоками, дублированными на узлах данных.

Secondary NameNode — вторичный узел имен, отдельный сервер, единственный в кластере, который копирует образ HDFS и лог транзакций операций с файловыми блоками во временную папку, применяет изменения, накопленные в логе транзакций к образу HDFS, а также записывает его на узел NameNode и очищает лог транзакций. Необходим для быстрого ручного восстановления NameNode в случае его выхода из строя.

Узел или сервер данных (DataNode, Node) – один их множества серверов, отвечающих за файловые операции и работу с блоками данных. Данные проходят с остальных узлов кластера к клиенту мимо узла NameNode.

Клиент (Client) – пользователь или приложение, взаимодействующий через специальный интерфейс с распределенной файловой системой. Создавая файл, клиент может явно указать размер блока файла (по умолчанию 64 Мб) и количество создаваемых реплик (по умолчанию значение равно 3-ем).

Особенности:

- большой размер блока по сравнению с другими файловыми системами (>64MB), поскольку HDFS предназначена для хранения большого количества огромных (>10GB) файлов.
- ориентация на недорогие и, поэтому не самые надежные сервера – отказоустойчивость всего кластера обеспечивается за счет репликации данных;
- репликация происходит в асинхронном режиме – информация распределяется по нескольким серверам прямо во время загрузки, поэтому выход из строя отдельных узлов данных не повлечет за собой полную пропажу данных.
- HDFS оптимизирована для потоковых считываний файлов, поэтому применять ее для нерегулярных и произвольных считываний нецелесообразно.

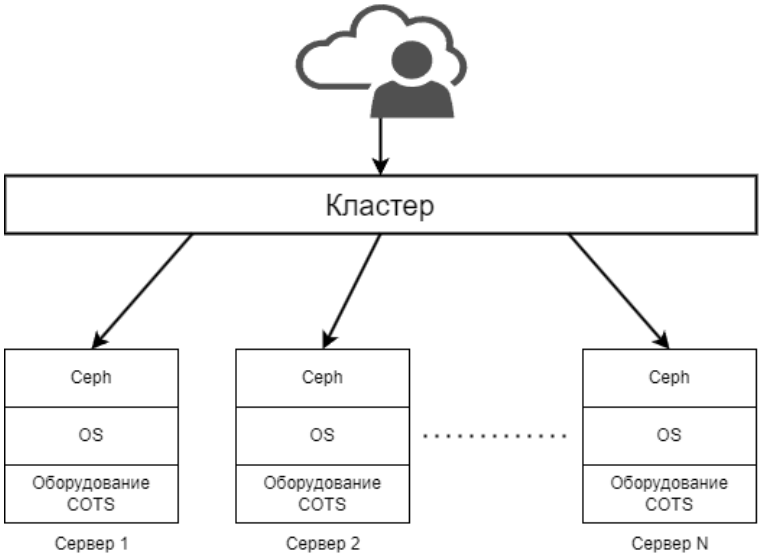
- файлы пишутся однократно, что исключает внесение в них любых произвольных изменений.
- принцип WORM (Write-once and read-many, один раз записать – много раз прочитать) полностью освобождает систему от блокировок типа «запись-чтение». Запись в файл в одно время доступен только одному процессу, что исключает конфликты множественной записи.
- сжатие данных и рациональное использование дискового пространства позволило снизить нагрузку на каналы передачи данных, которые чаще всего являются узким местом в распределенных средах.
- самодиагностика — каждый узел данных через определенные интервалы времени отправляет диагностические сообщения узлу имен, который записывает логи операций над файлами в специальный журнал.

Недостатки:

- сервер имен является центральной точкой всего кластера и его отказ повлечет сбой системы целиком.
- отсутствие полноценной репликации Secondary NameNode.
- отсутствие возможности дописывать или оставить открытым для записи файлы в HDFS, за счет чего в классическом дистрибутиве Apache Hadoop невозможно обновлять блоки уже записанных данных.
- отсутствие инструментов для поддержки ссылочной целостности данных, что не гарантирует идентичность реплик. HDFS перекладывает проверку целостности данных на клиентов. При создании файла клиент рассчитывает контрольные суммы каждые 512 байт, которые в последующем сохраняются на сервере имен. При считывании файла клиент обращается к данным и контрольным суммам. В случае их несоответствия происходит обращение к другой реплике.

1.5 Ceph

Ceph – распределенная масштабируемая система хранения с дизайном без единой точки отказа. Система разрабатывалась как открытый проект с файловой системой высокой масштабируемости до уровня эксабайт и рассчитанная на работу на стандартном коммерчески доступном серверном оборудовании COTS (Commercial Off The Shelf). Ceph приобретает все большую популярность в облачных системах хранения данных.



Основные принципы построения Ceph:

- Масштабируемость всех компонентов.
- Отсутствие единой точки отказа.
- Использование открытого ПО (Linux).
- Использование стандартного оборудования COTS (Commercial Off The Shelf).
- Автономность в управлении компонентами.

Ceph для клиента выглядит как обычная файловая система с папками и файлами, организованными по принципу иерархии.

Ceph реплицирует данные, за счет чего файловая система устойчива к отказам при использовании обычного стандартного компьютерного оборудования COTS, которое не требует специального обслуживания или администрирования. Поэтому она хорошо подходит для организаций, которые не обладают экспертизой в IT, и в которых недостаточно средств для содержания штата IT-специалистов.

1.6 Lustre

Lustre – это кластерная архитектура для СХД, центральным элементом которой является файловая система Lustre, работающая на ОС Linux и обеспечивающая POSIX-интерфейс для файловой системы UNIX.

Lustre способна масштабировать емкость и производительность практически до любых требуемых величин, устраняя надобность в развертывании многих отдельных файловых систем в каждом компьютерном кластере. Кроме агрегирования емкости хранения многих серверов, пропускная способность ввода-вывода также агрегируется и масштабируется при добавлении серверов – в том числе в динамическом режиме.

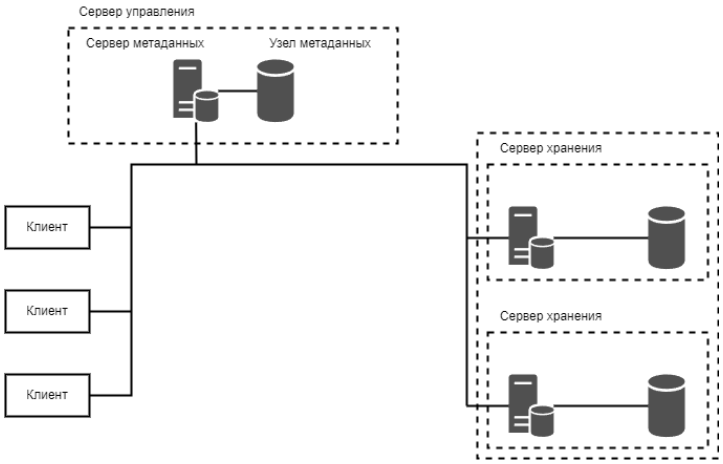
Lustre также не очень подходит для сетевых моделей peer-to-peer, где клиенты и серверы работают на одном узле хранения, и каждый занимает небольшую область емкости, из-за недостаточной репликации на уровне ПО Lustre. При этом, если один клиент или сервер отказывает, то данные этого узла будут недоступны, пока узел не перезапустится.

Архитектура Lustre

Сервер метаданных MDS (Metadata Server) – обеспечивает доступ к метаданным файловой системе и управляет запросами клиентов.

Узел метаданных – хранит метаданные файловой системы.

Сервер хранения – хранит данные.

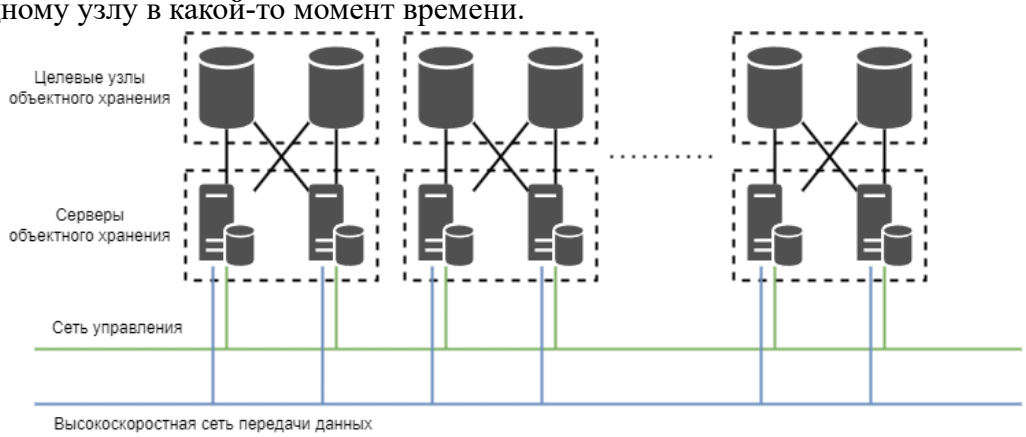


Архитектура серверов хранения

Сервер объектного хранения – обрабатывает запросы данных.

Целевой узел объектного хранения – диск с данными.

Любой узел может быть доступен для многих серверов, но сервер может иметь доступ только к одному узлу в какой-то момент времени.



1.7 Сравнение

Самым главным отличительным плюсом всех рассмотренных архитектур является практически бесконечная расширяемость хранилищ данных. При этом все файловые системы используют сеть для соединения хранилищ между собой.

Ниже приведена таблица сравнения рассмотренных систем, основанная на удобстве эксплуатации.

	Наличие единой точки отказа	**Требование специального оборудования**	**Зависимость от платформы**
NFS	+		
ZFS	-		
GFS	+		
HDFS	-		
Ceph	-	+	+
Lustre	+		+

Так же стоит отметить, что большинство файловых систем поддерживают механизм репликации, который повышает вероятность сохранения данных при физических поломках оборудования и в некоторых случаях снижает нагрузку на основное хранилище данных.

Стоит отметить, что в текущем контексте отсутствие плюса в пункте “Наличие единой точки отказа” подразумевает, что каждое хранилище является отдельным модулем и при его отказе пользователь не сможет получить доступ только к данным из этого хранилища. Сохранить доступ к данным при отказе хранилища без настроенного механизма репликации не получится.

2 Концепция приложения

2.1 Требования

Из обзорной части очевидно, что такие решения рассчитаны на уровень использования компаниями и для развёртывания такой системы необходимо закупать дополнительное оборудование. Однако все это не подходит для обычного пользователя, имеющего в своем распоряжении всего лишь несколько персональных устройств, что заставляет его использовать одновременно несколько устройств и передавать информацию через различные буферы в виде жестких накопителей информации. Поэтому первое и основное требование к приложению – **не использовать/требовать дополнительное оборудование, кроме персональных устройств самого пользователя**.

Конечно же пользователь может иметь в своём распоряжении и дополнительное оборудование, например, различные накопители и сервера, не подходящее для полноценного использования, но все еще имеющее возможности хранить большое количество данных или имеющее малое количество памяти, но достаточно мощные характеристики железа для обработки данных или не имеющее ни того, ни другого, но при этом очень портативное, чтобы было удобно иметь доступ откуда угодно. Поэтому стоит определить некоторые **роли запущенного приложения на определенном оборудовании**, а именно:

- **Storage** – устройство, обладающее только возможностью хранить данные.
- **Runner** – устройство, позволяющее запускать приложения без сильных потерь производительности. Имеет только копию данных, связанных с запущенным приложением.
- **Router** – определяет устройство, имеющее интерфейс доступа к данным. Выполняет сбор информации о файловой системе со стораджей и распределяет нагрузку на раннеры.

Таким образом, одно устройство может иметь от 1 до 3 ролей одновременно, так как они никак не конфликтуют друг с другом по функциональности.

Очевидно, что пользователь может обладать устройствами с различной операционной системой (Windows, MacOS, Linux, Android и другие Unix системы), поэтому следующим требованием является **независимость от операционной системы**.

Обычный пользователь с большой вероятностью не будет обладать достаточной экспертизой в настройке таких систем, следовательно **настройка должна происходить автоматически и с минимальным количеством параметров**.

Для сохранения данных с неустойчивых хранилищ **необходимо предоставить гибкий и точечный механизм репликации**, так как пользователь с наибольшей вероятностью не обладает достаточно большим количеством памяти, чтобы иметь реплику всей файловой системы.

2.2 Особенности реализации и ограничения

Исходя из универсальности использования самым логичным будет **предоставить пользователю веб-интерфейс** для взаимодействия с файловой системой, переложив системные особенности взаимодействия с файлами на саму операционную систему.

Так же основываясь на автоматизации настройки приложения необходимо ввести ограничение на **использование только локальной сети** для существующий устройств пользователя, хорошим примером здесь будет использование роутера, который предоставляет пользователю как доступ к глобальной, так и локальной сетям одновременно.

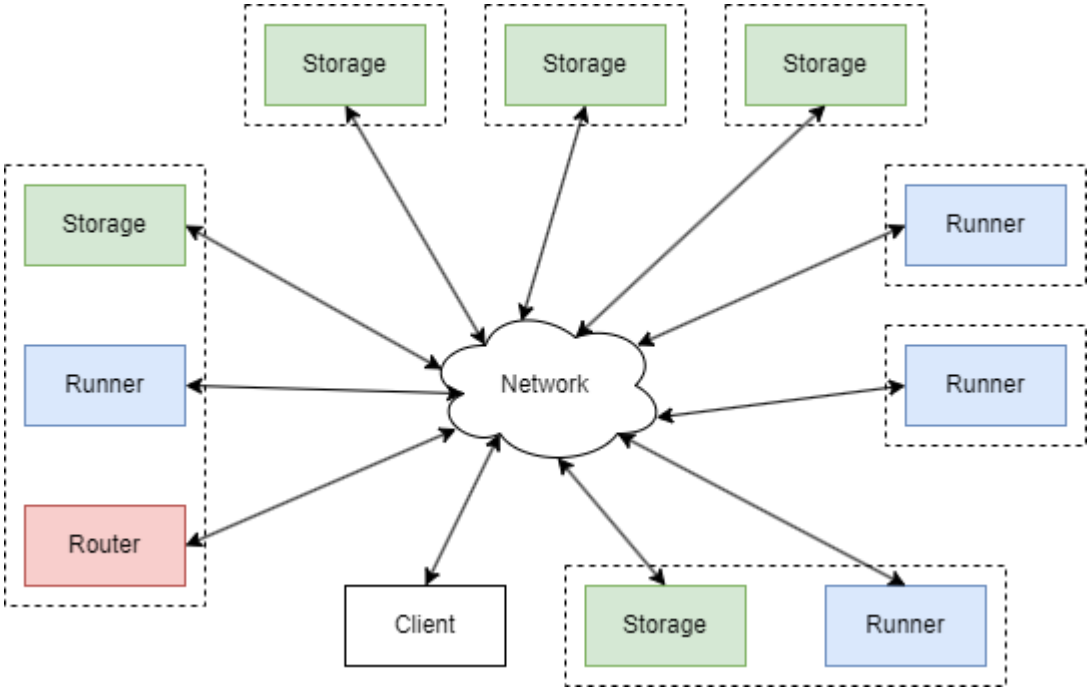
Безопасность выполнения операций так же ложится на **пользователя операционной системы,** под которым запущен инстанс приложения, позволяя гибко настроить доступ к данным на каждом устройстве.

2.3 Архитектура

Для минимально работающей системы необходимо, чтобы была запущена только одна роль – Router, т.к. она обеспечивает клиента интерфейсом. Но в таком случае никаких данных увидеть не получится.

Чтобы увидеть данные необходимо либо добавить роль Storage в уже запущенное приложение, либо запустить дополнительное с этой ролью. Теперь клиенту становится доступна возможность изменения состояния файловой системы устройства, на которой запущена эта роль. Если запустить ещё на одном устройстве приложение с этой ролью, то клиент сможет увидеть объединённую файловую систему двух хранилищ.

Однако, имея только вышеуказанные две роли, клиент не сможет изменять сами файлы. Для получения такой возможности необходимо добавить роль Runner. Она позволяет открывать файлы через предназначенные для них приложения и изменять их.



2.4 Безопасность

В данной концепции приложения вопрос безопасности данных очень важен. Любой человек подключившийся к сети, зная правильные запросы, может с лёгкостью удалить все существующие данные или занести различные вредоносные программы. Поэтому используется два механизма безопасности.

Все приложения используют протокол https для общения. При надобности пользователь так же можешь перегенерировать сертификаты и заменить их.

Роль роутера поддерживает механизм аутентификации. При запуске приложение случайно генерирует достаточно большой токен на основе логина и пароля, которые указаны в конфигурационном файле или флагах запуска приложения. Все эндпоинты приложения включают в себя этот токен, поэтому, не имея логина и пароля, становится невозможно отправить хотя бы какой-то запрос.

Таким образом каждый эндпоинт будет описываться шаблоном url/token/role/path.

2.5 Запросы

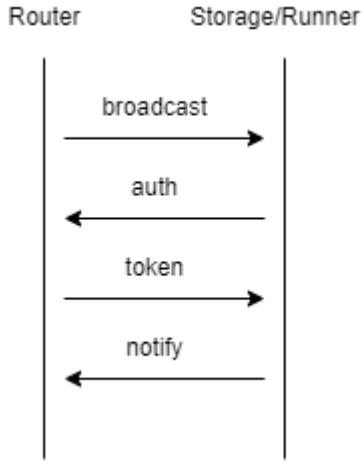
Далее будут рассматриваться модели запросов, выполняемые ролями. Все запросы выполняются через сеть, и никогда напрямую между ролями, поэтому далее сеть на рисунках упоминаться не будет.

2.5.1 Запуск ролей

Независимо от порядка запуска ролей каждая выполняет постоянные и периодические действия по поддержанию актуальной информации о кластере.

Роутер будет периодически отправлять широковещательные запросы в сеть.

Остальные роли при получении таких запросов будут пытаться пройти аутентификацию и получить токен, а после этого отправят оповещение на специальный эндпоинт.



При запросе оповещения storage/runner передают следующую информацию: тип роли, свой адрес, имя хоста, свой токен.

После этого роли считаются соединёнными, и каждая знает необходимые токены других ролей, поэтому все остальные запросы будут выполняться без прохождения аутентификации.

2.5.2 Просмотр файловой системы

При просмотре файловой системы роутер выполняет запрос информации на все известные хранилища. В ответ хранилища передают информацию о всех известных по указанному пути файлах и папках.

В передаваемую информацию входят имя, время последнего изменения и размер.

2.5.3 Изменения файловой системы

Все изменения производятся локально на хранилище, и запросы отправляются только к определённому хранилищу, основываясь либо на полученных метаданных файла, либо на выборе самого пользователя.