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 5](#_Toc166820388)

[1.1 NFS (Network File System) 5](#_Toc166820389)

[1.2 ZFS (Zettabyte File System) 6](#_Toc166820390)

[1.2.1 Хранение в пулах (Pooled storage) 6](#_Toc166820391)

[1.2.2 Копирование при записи (Copy-on-write) 6](#_Toc166820392)

[1.2.3 Снапшоты (Snapshots) 6](#_Toc166820393)

[1.3 GFS 7](#_Toc166820394)

[1.4 HDFS 8](#_Toc166820395)

[1.5 Ceph 10](#_Toc166820396)

[1.6 Lustre 11](#_Toc166820397)

[1.7 Сравнение 12](#_Toc166820398)

[2 Концепция приложения 13](#_Toc166820399)

[2.1 Требования 13](#_Toc166820400)

[2.2 Особенности реализации и ограничения 13](#_Toc166820401)

[2.3 Архитектура 14](#_Toc166820402)

[2.4 Безопасность 14](#_Toc166820403)

[2.5 Запуск ролей 15](#_Toc166820404)

[2.6 Функциональность 15](#_Toc166820405)

[2.6.1 Просмотр файловой системы 15](#_Toc166820406)

[2.6.2 Изменения файловой системы 15](#_Toc166820407)

[2.6.3 Фильтрация показываемых файлов 15](#_Toc166820408)

[2.6.4 Репликация 16](#_Toc166820409)

[2.6.5 Открытие файлов и работа с ними 16](#_Toc166820410)

[2.7 Флаги запуска 16](#_Toc166820411)

[3 Реализация 17](#_Toc166820412)

[3.1 Сообщения 17](#_Toc166820413)

[3.1.1 Оповещение (Notify) 17](#_Toc166820414)

[3.1.2 Топология (Topology) 17](#_Toc166820415)

[3.1.3 Информация о файле (FileInfo) 17](#_Toc166820416)

[3.1.4 Директория файловой системы (FilesystemDirectory) 17](#_Toc166820417)

[3.1.5 Запрос аутентификации клиента (ClientAuthRequest) 17](#_Toc166820418)

[3.1.6 Запрос вставки (InsertRequest) 18](#_Toc166820419)

[3.1.7 Запрос переименования (RenameRequest) 18](#_Toc166820420)

[3.1.8 Запрос копирования/перемещения (CopyRequest/MoveRequest) 18](#_Toc166820421)

[3.1.9 Запрос открытия/запуска файла (OpenRequest) 18](#_Toc166820422)

[3.1.10 Ответ в строке состояния (StatusBarResponse) 18](#_Toc166820423)

[3.1.11 Ответ открытия/запуска файла (OpenResponse) 18](#_Toc166820424)

[3.1.12 Координаты и дельта координат (Coords/CoordsDelta) 19](#_Toc166820425)

[3.1.13 Событие мыши (MouseEvent) 19](#_Toc166820426)

[3.1.14 Событие клавиатуры (KeyboardEvent) 19](#_Toc166820427)

[3.2 Сервер 20](#_Toc166820428)

[3.3 Роль Storage 21](#_Toc166820429)

[3.3.1 Уведомление роутера 21](#_Toc166820430)

[3.3.2 Сбор данных о файловой системе 22](#_Toc166820431)

[3.3.3 Обработчик добавления файла 22](#_Toc166820432)

[3.3.4 Обработчик получения файла 23](#_Toc166820433)

[3.3.5 Обработчик удаления файла 24](#_Toc166820434)

[3.3.6 Обработчик копирования файла 25](#_Toc166820435)

[3.3.7 Обработчик перемещения файла 26](#_Toc166820436)

[3.3.8 Обработчик переименования файла 26](#_Toc166820437)

[3.4 Роль Runner 27](#_Toc166820438)

[3.4.1 Уведомление роутера 27](#_Toc166820439)

[3.4.2 Обработчик открытия/запуска файлов 27](#_Toc166820440)

[3.4.3 Обработчик получения изображения приложения 27](#_Toc166820441)

[3.4.4 Обработчик события мыши 27](#_Toc166820442)

[3.4.5 Обработчик события клавиатуры 27](#_Toc166820443)

[3.4.6 Обработчик создания прямого стрима приложения 27](#_Toc166820444)

[3.4.7 Обработчик создания стрима приложения 27](#_Toc166820445)

[3.5 Роль Router 27](#_Toc166820446)

[3.5.1 Сбор данных о файловой системе 27](#_Toc166820447)

[3.5.2 Отслеживание устаревших инстансов 27](#_Toc166820448)

[3.5.3 Репликация 27](#_Toc166820449)

[3.5.4 Фильтрация файлов 27](#_Toc166820450)

[3.5.5 Обработчик уведомлений 27](#_Toc166820451)

[3.5.6 Обработчик открытия/запуска файлов 27](#_Toc166820452)

[3.6 Аутентификации (Веб) 27](#_Toc166820453)

[3.7 Домашняя страница (Веб) 27](#_Toc166820454)

[3.8 Файловая система (Веб) 27](#_Toc166820455)

[3.8.1 Сбор данных о файловой системе 27](#_Toc166820456)

[3.9 Настройки (Веб) 27](#_Toc166820457)

[3.9.1 Фильтрация 27](#_Toc166820458)

[3.9.2 Репликация 27](#_Toc166820459)

[4 Разработка методики тестирования 27](#_Toc166820460)

# Обзор существующих распределенных файловых систем хранения

## 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](https://translated.turbopages.org/proxy_u/en-ru.ru.b9241a76-656c9ce8-43a5a94f-74722d776562/https/www.baeldung.com/cs/client-vs-server-terminology)**.** 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.

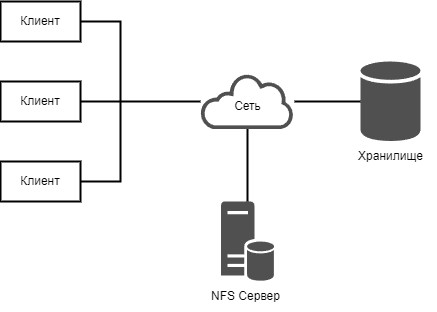


Рис.1.1.1 – Архитектура NFS

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](https://translated.turbopages.org/proxy_u/en-ru.ru.b9241a76-656c9ce8-43a5a94f-74722d776562/https/www.baeldung.com/cs/authentication-vs-authorization) 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](https://translated.turbopages.org/proxy_u/en-ru.ru.b9241a76-656c9ce8-43a5a94f-74722d776562/https/en.wikipedia.org/wiki/Hierarchical_storage_management)**.** Если пользователь хочет хранить данные централизованно, он не сможет сделать это с помощью NFS.

## ZFS ([Zettabyte](https://ru.wikipedia.org/wiki/%D0%97%D0%B5%D1%82%D1%82%D0%B0%D0%B1%D0%B0%D0%B9%D1%82) 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.

### 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.

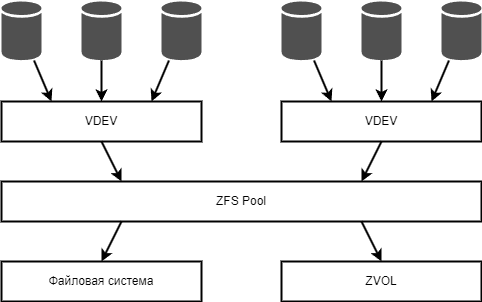


Рис.1.2.1 – Архитектура ZFS

### ****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.

### 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.

## 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*

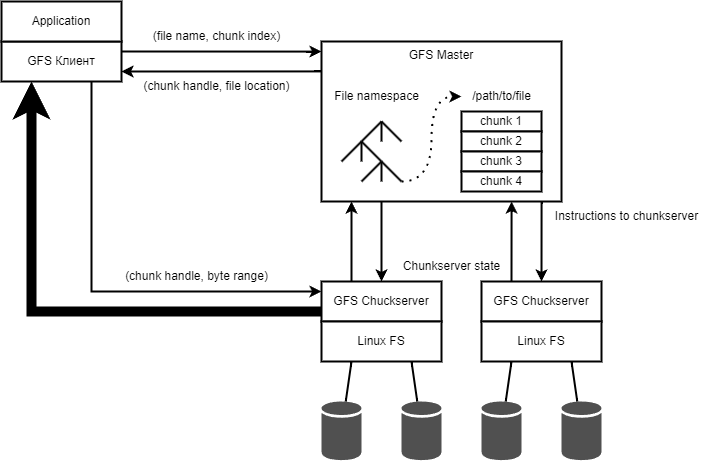


Рис.1.3.1 – Архитектура GFS

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.

## HDFS

**HDFS (**[Hadoop](https://bigdataschool.ru/wiki/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*

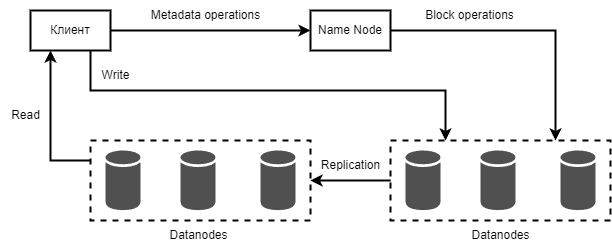


Рис.1.4.1 – Архитектура HDFS

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 (Сlient) – 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](https://bigdataschool.ru/bigdata/hdfs-%d0%b7%d0%b0%d0%bf%d0%b8%d1%81%d1%8c-%d1%84%d0%b0%d0%b9%d0%bb%d0%be%d0%b2-big-data.html)**,** что исключает внесение в них любых произвольных изменений.
* **принцип WORM (Write-once and read-many**, [один раз записать](https://bigdataschool.ru/bigdata/hdfs-%d0%b7%d0%b0%d0%bf%d0%b8%d1%81%d1%8c-%d1%84%d0%b0%d0%b9%d0%bb%d0%be%d0%b2-big-data.html) – 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.

## 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.

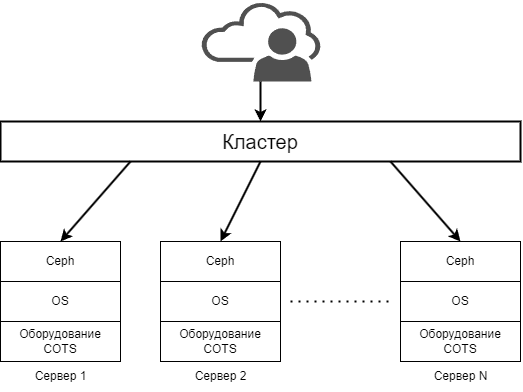


Рис.1.5.1 – Архитектура Ceph

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.

## 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.

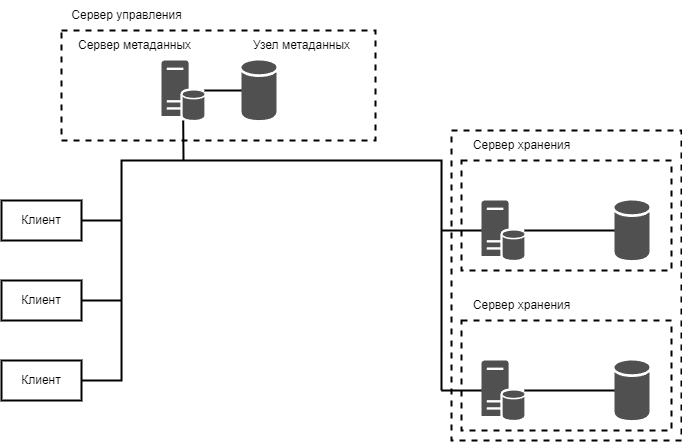


Рис.1.6.1 – Архитектура Lustre

*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.

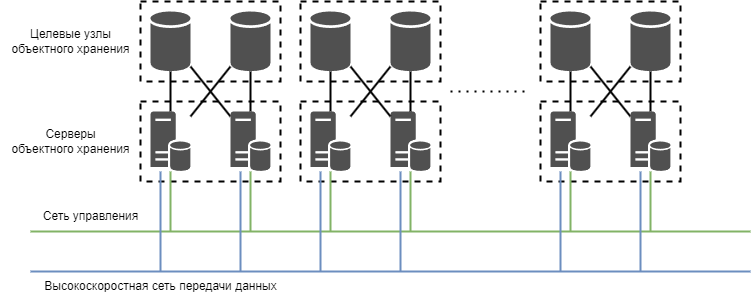


Рис.1.6.2 – Архитектура серверов хранения Lustre

## 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.

# Концепция приложения

## 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.

## 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 на уровне ОС так же ложится на **пользователя операционной системы,** под которым запущен инстанс приложения, позволяя гибко настроить доступ к данным на каждом устройстве.

## 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.

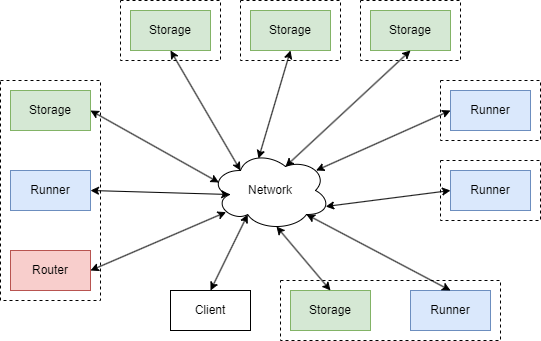


Рис.2.3.1 – Архитектура приложения

## 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.

## 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.

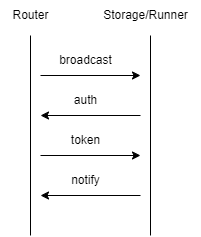


Рис.2.5.1 – Алгоритм запуска роли

При запросе оповещения storage/runner передают следующую информацию: тип роли, свой адрес, имя хоста, свой токен, название платформы ОС и текущую метку времени (таймстамп).

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

Важно отметить, что перед отправкой уведомления роутеру инстанс проверяет совпадение сертификатов. Так же и со стороны роутера проверяется совпадение сертификатов от инстанса. Таким образом, невозможно будет подделать инстанс приложения для получения каких-либо данных, так как запрос будет отклонён на https уровне.

## Функциональность

### 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, название хоста источника, название платформы источника, время последнего изменения и размер.

### 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.

В список изменений файловой системы входят: создание, копирование, перемещение, удаление и переименование файлов.

### Фильтрация показываемых файлов

Пользователь имеет возможность работать с чёрным и белым списками, в которых будут указаны пути директорий и файлов или шаблоны выражений (например, регулярные выражения).

### Репликация

Пользователь имеет возможность настраивать частную репликацию файлов, указывая периодичность, хост источник и целевой хост, а также соответствующие пути к этим файлам.

### Открытие файлов и работа с ними

Открытие файлов происходит на инстансе с ролью runner в соответствующем ему установленном в системе приложении.

Если инстанс имеет две роли storage и runner, то приложение будет приоритетно открыто на нём. В случае ошибки открытия или отсутствия роли runner, запрос на открытие будет отправлен другому хосту с соответствующей платформой, а файл будет скопирован для его локального открытия. После завершения работы устаревший файл на хосте источнике будет заменён файлом открытом на хосте с ролью runner, а временный рабочий файл удалён.

Так же для работы с файлом будет запущен стрим приложения, которое открыло этот файл, в отдельном веб-окне. Все нажатия мыши, клавиатуры, нажатия на экран мобильного устройства и прокрутка будут переданы в соответствующее приложение.

## Флаги запуска

* cert-crt – путь к файлу vss.crt (по умолчанию "certificates/vss.crt")
* cert-key – путь к файлу vss.key (default "certificates/vss.key")
* url – адрес сервера
* listen-port – порт для топологии (по умолчанию "3311")
* log-enable – включает логи
* log-level – уровень логирования (error, info, verbose, debug) (по умолчанию "info")
* log-path – путь к файлу логов (default "logs/log\_2024-05-14T07:06:16.txt")
* platform - платформа ('windows', 'linux', 'darwin', …) (по умолчанию "windows")
* storage – включает роль storage
* router – включает роль router
* runner – включает роль runner
* token – токен безопасности
* username – имя пользователя (по умолчанию "admin")
* password – пароль для аутентификации (по умолчанию "admin")

# Реализация

## Сообщения

Определение структуры сообщений для общения - самый важный этап. Для понимания приведённых далее алгоритмов, необходимо знать возможности, предоставляемые каждой структурой данных.

### Оповещение (Notify)

Содержит поля:

* Type – имя роли.
* Url – адрес хоста.
* Hostname – имя хоста.
* Token – токен безопасности.
* Platform – название платформы.
* Timestamp – метка времени в наносекундах.

Используется для оповещения роутера о появлении инстанса приложения в сети.

### Топология (Topology)

Содержит поля:

* Storages – массив Notify структур, описывающих приложения с ролью storage.
* Runners – массив Notify структур, описывающих приложения с ролью runner.

Используется для хранения информации о кластере, поиска данных в запросах и отображения информации о топологии на основной странице.

### Информация о файле (FileInfo)

Содержит поля:

* ModTime – время последней модификации.
* Size – размер.
* Url – адрес хранилища.
* Platform – название платформы.
* Hostname – имя хоста.

Используется для предоставления информации о файловой системе.

### Директория файловой системы (FilesystemDirectory)

Содержит поля:

* Directories – таблица ключ-значение, описывающих директории, где ключ – имя директории, а значение FileInfo структура.
* Files – таблица ключ-значений, описывающих файлы, где ключ – имя файла, а значение массив FileInfo структур, т.к. на разных хостах в одной директории могут храниться файлы с одинаковым именем.

Используется для предоставления информации о файловой системе.

### Запрос аутентификации клиента (ClientAuthRequest)

Содержит поля:

* Username – имя пользователя.
* Password – пароль пользователя.

Используется для аутентификации пользователя и ролей приложения.

### Запрос вставки (InsertRequest)

Содержит поля:

* Type – тип файла.
* Path – путь до директории.
* Name – имя файла.

Используется для добавления файла или директории.

### Запрос переименования (RenameRequest)

Содержит поля:

* Path – путь до директории.
* OldName – старое имя файла.
* NewName – новое имя файла.

Используется для переименования файлов и директорий.

### Запрос копирования/перемещения (CopyRequest/MoveRequest)

Содержит поля:

* SrcPath – путь к файлу на хосте источнике.
* DstPath – путь к файлу на целевом хосте.
* SrcUrl – адрес хоста источника.

Используется для копирования или перемещения файлов и директорий.

### Запрос открытия/запуска файла (OpenRequest)

Содержит поля:

* Platform – платформа ОС.
* Path – путь к файлу.
* SrcUrl – адрес хоста источника.
* Hostname – имя хоста источника.

Используется для старта запуска выбранного файла.

### Ответ в строке состояния (StatusBarResponse)

Содержит поля:

* Status – статус (успех/провал).
* Text – комментарий.

Используется для передачи статуса выполнения запросов.

### Ответ открытия/запуска файла (OpenResponse)

Содержит поля:

* Pid – идентификатор процесса запустившего файл.
* RunnerUrl – адрес хоста, на котором был запущен файл.
* ClientUrl – адрес хоста клиента, который отправил запрос.
* Error – ошибка открытия.
* StatusBar – статут выполнения запроса.

Используется для ответа ролей runner и router, и дальнейшего получения информации и о открытом/запущенном файле.

### Координаты и дельта координат (Coords/CoordsDelta)

Содержит поля:

* X – абсолютное горизонтальное значение относительно нуля координат или изменение по сравнению с предыдущим значением.
* Y – абсолютное вертикальное значение относительно нуля координат или изменение по сравнению с предыдущим значением.

Используется для передачи координат событий мыши.

### Событие мыши (MouseEvent)

Содержит поля:

* Type – тип события (нажата или отпущена левая/правая клавиша, прокрутка)
* Coords – абсолютные координаты курсора мыши в момент события.
* Scroll – изменения координат при прокрутке.

Используется для передачи событий нажатия кнопки мыши или прокрутки.

### Событие клавиатуры (KeyboardEvent)

Содержит поля:

* Type – тип нажатия клавиши (нажата или отпущена).
* Keycode – код клавиши.

Используется для передачи события нажатия клавиши на клавиатуре.

## Сервер

Сервер – основа приложения, он занимается аутентификацией пользователя и других инстансов приложения и перенаправляет запросы в соответствующие обработчики.

Алгоритмы аутентификации и обработки запросов приведены на рисунках 3.2.1 и 3.2.2 соответственно.

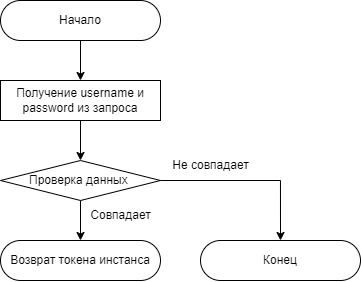


Рис.3.2.1 – Алгоритм аутентификации

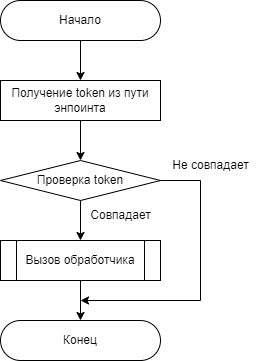


Рис.3.2.2 – Алгоритм обработки запросов

## Роль Storage

### Уведомление роутера

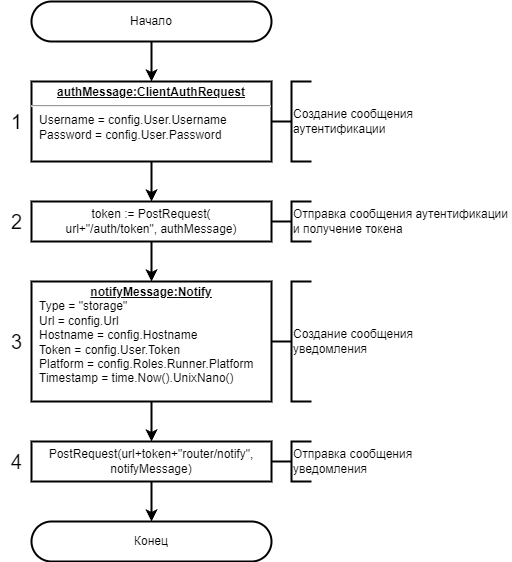


Рис.3.3.1.1 – Алгоритм уведомления роутера

### Сбор данных о файловой системе

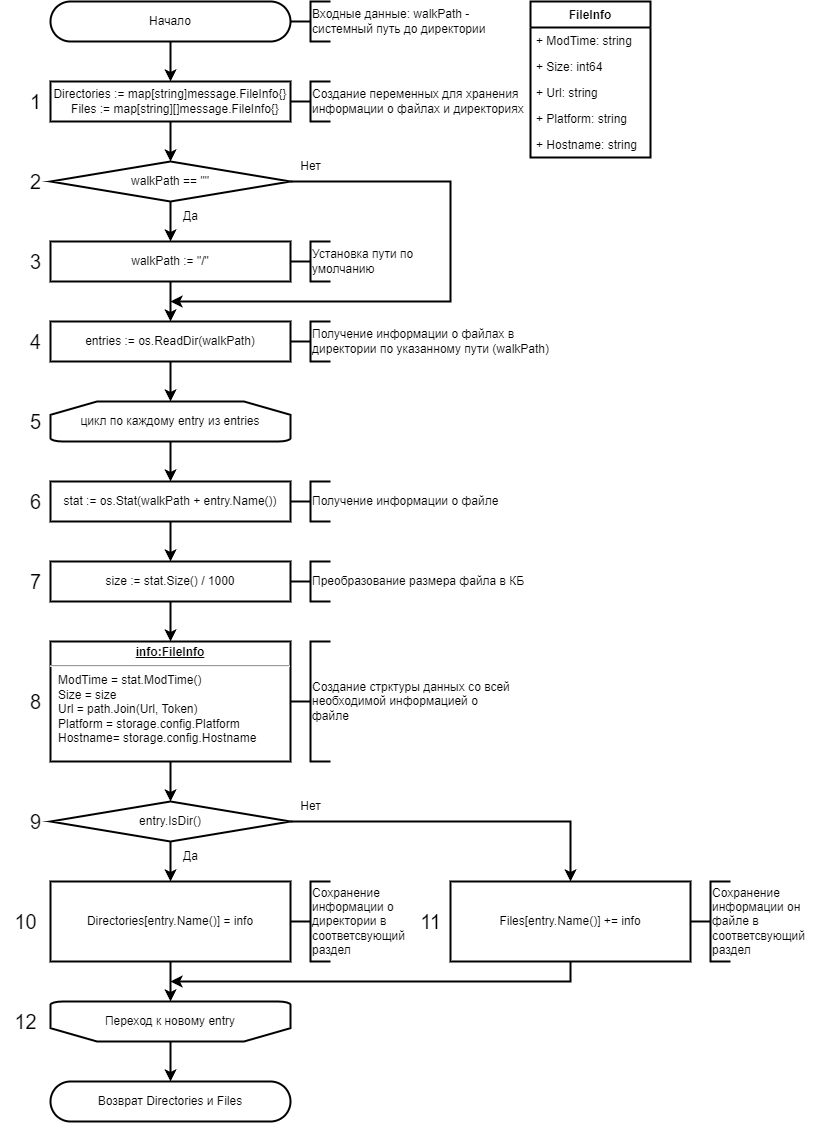


Рис.3.3.2.1 – Алгоритм сбора данных о файловой системе

### Обработчик добавления файла

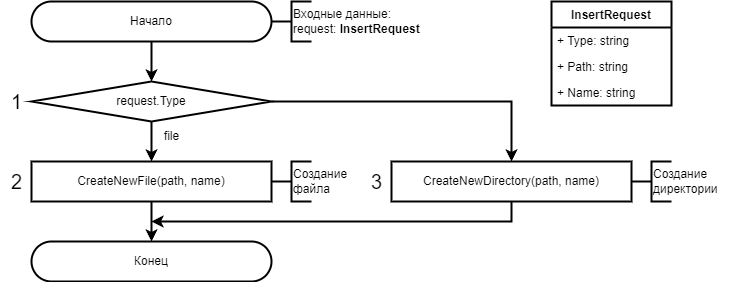


Рис.3.3.3.1 – Алгоритм добавления файла

### Обработчик удаления файла

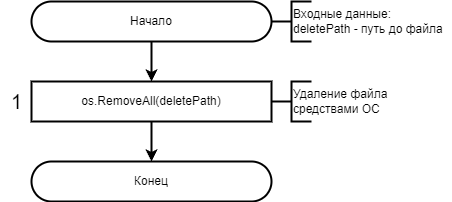


Рис.3.3.4.1 – Алгоритм удаления файла

### Обработчик получения файла

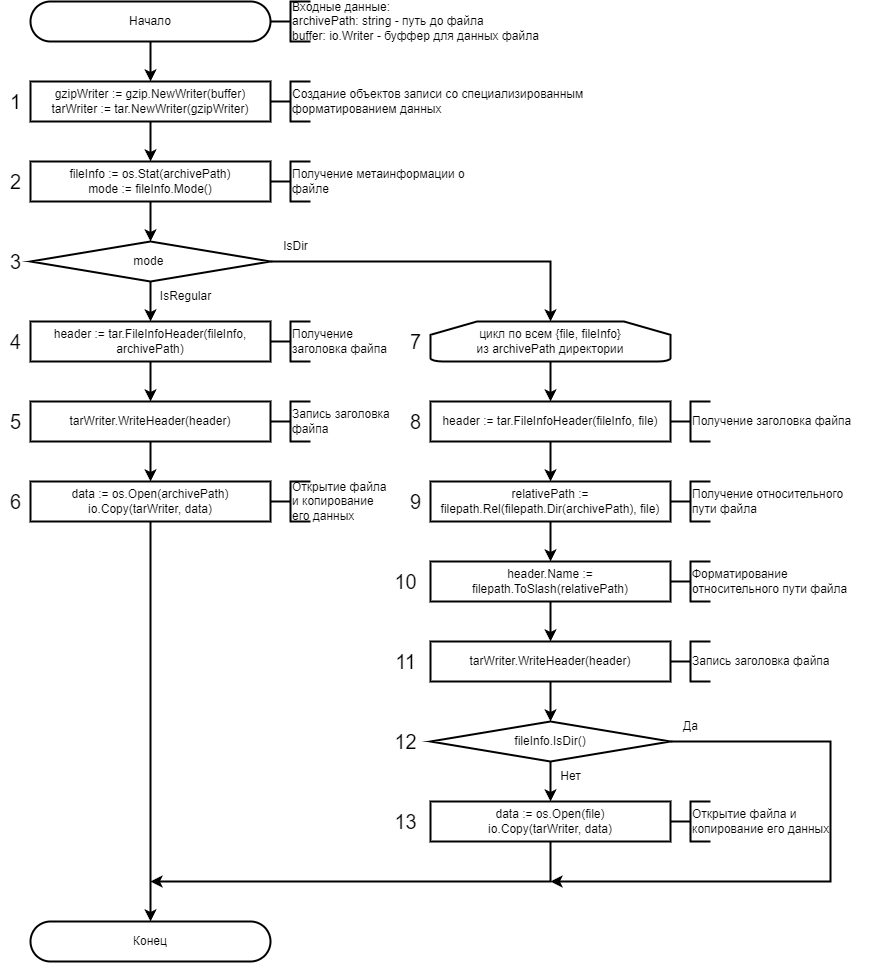


Рис.3.3.5.1 – Алгоритм получения файла

### Обработчик копирования файла

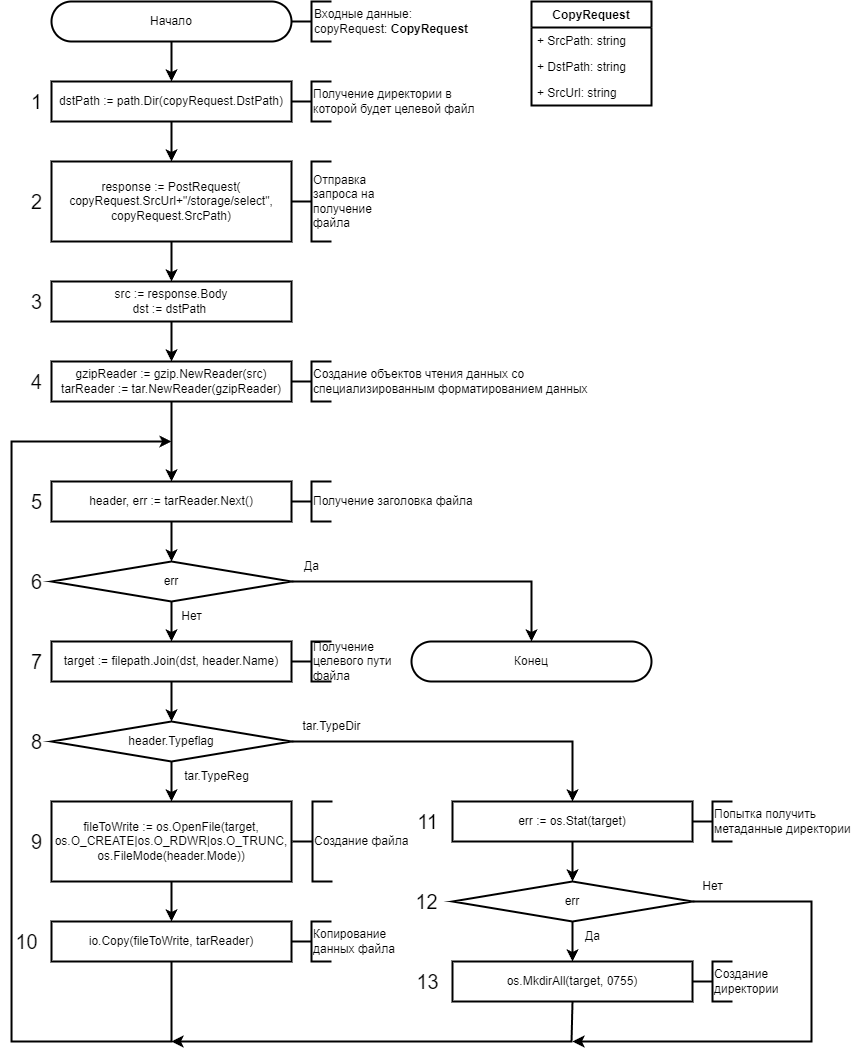


Рис.3.3.6.1 – Алгоритм копирования файла

### Обработчик перемещения файла

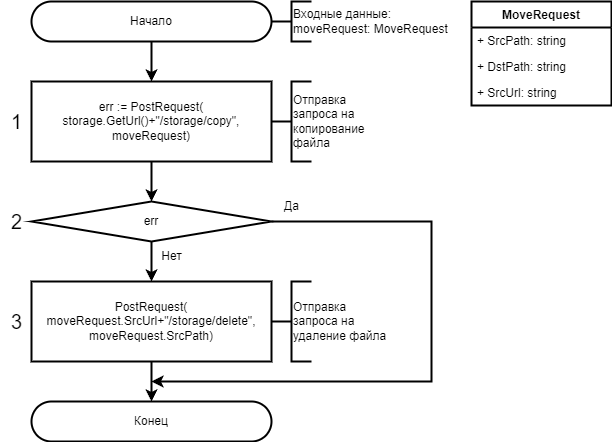


Рис.3.3.7.1 – Алгоритм перемещения файла

### Обработчик переименования файла

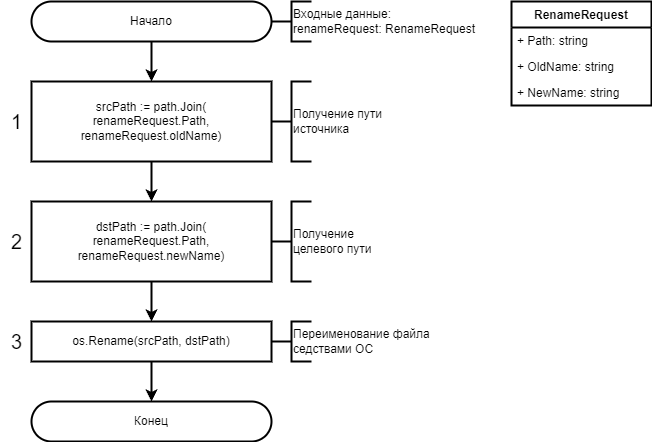


Рис.3.3.8.1 – Алгоритм переименования файла

## Роль Runner

### Уведомление роутера

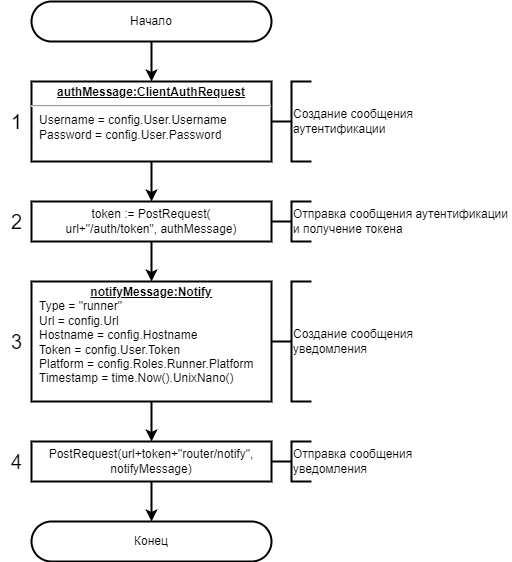


Рис.3.4.1.1 – Алгоритм уведомления роутера

### 

### Обработчик открытия/запуска файлов

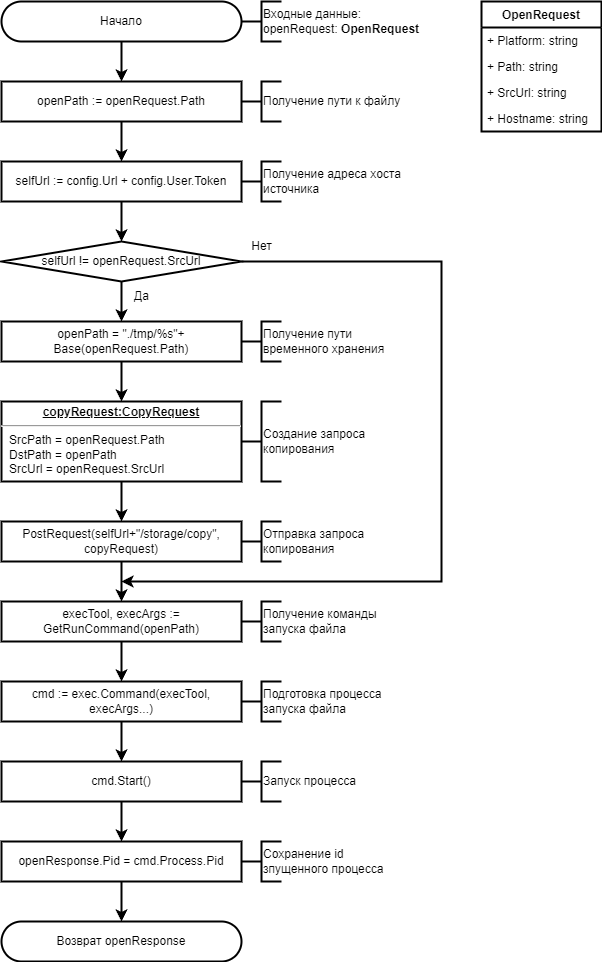


Рис.3.4.2.1 – Алгоритм открытия и запуска файлов

### Обработчик получения изображения приложения

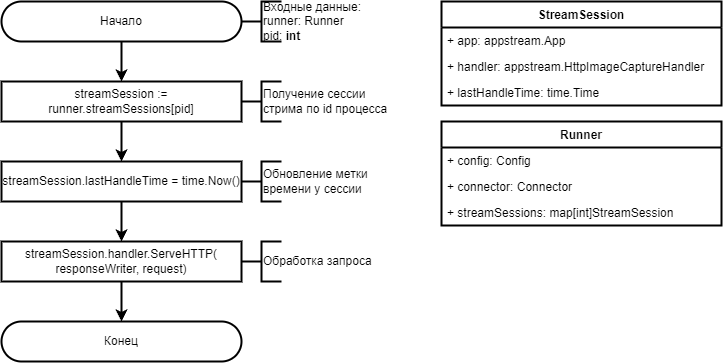


Рис.3.4.3.1 – Алгоритм получения изображения приложения

### Обработчик события клавиатуры

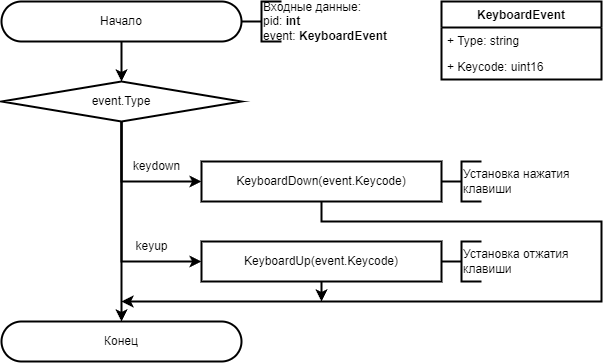


Рис.3.4.4.1 – Алгоритм обработки события клавиатуры

### Обработчик события мыши

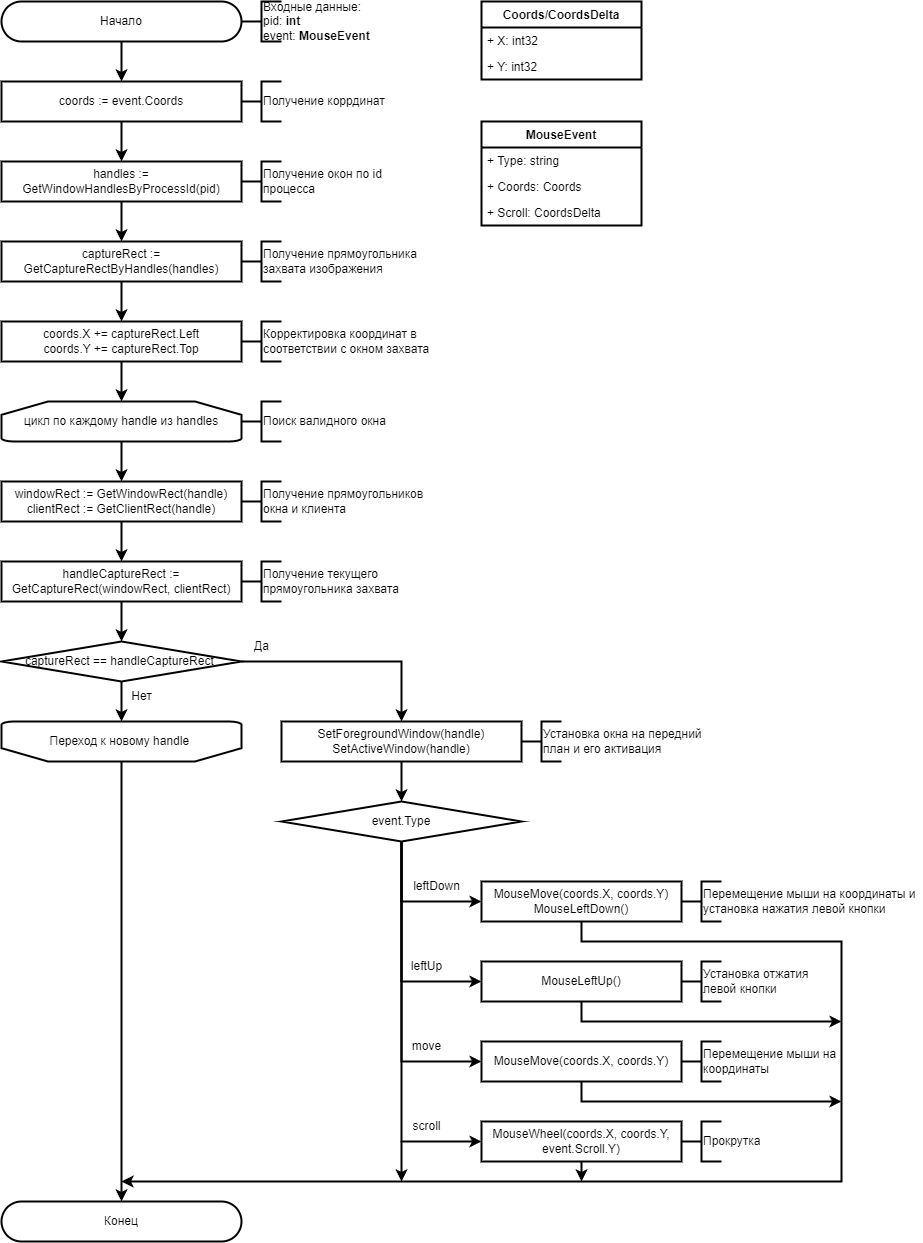


Рис.3.4.5.1 – Алгоритм обработки события мыши

### Обработчик создания прямого стрима приложения

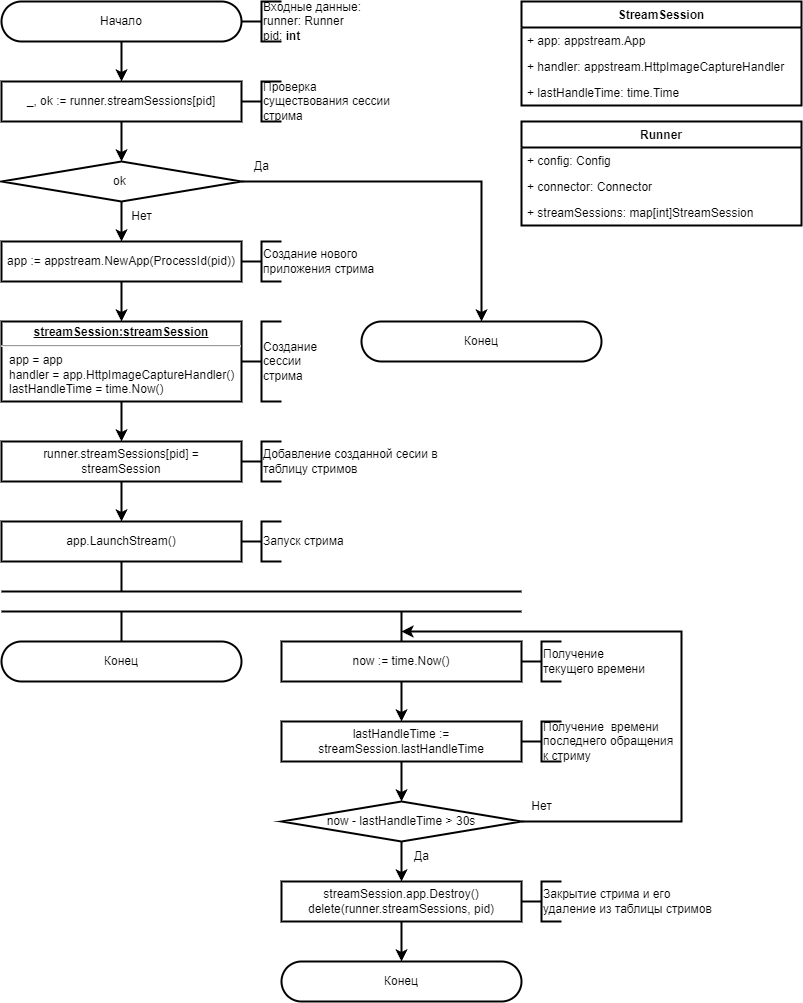


Рис.3.4.6.1 – Алгоритм создания прямого стрима приложения

### Обработчик создания стрима приложения

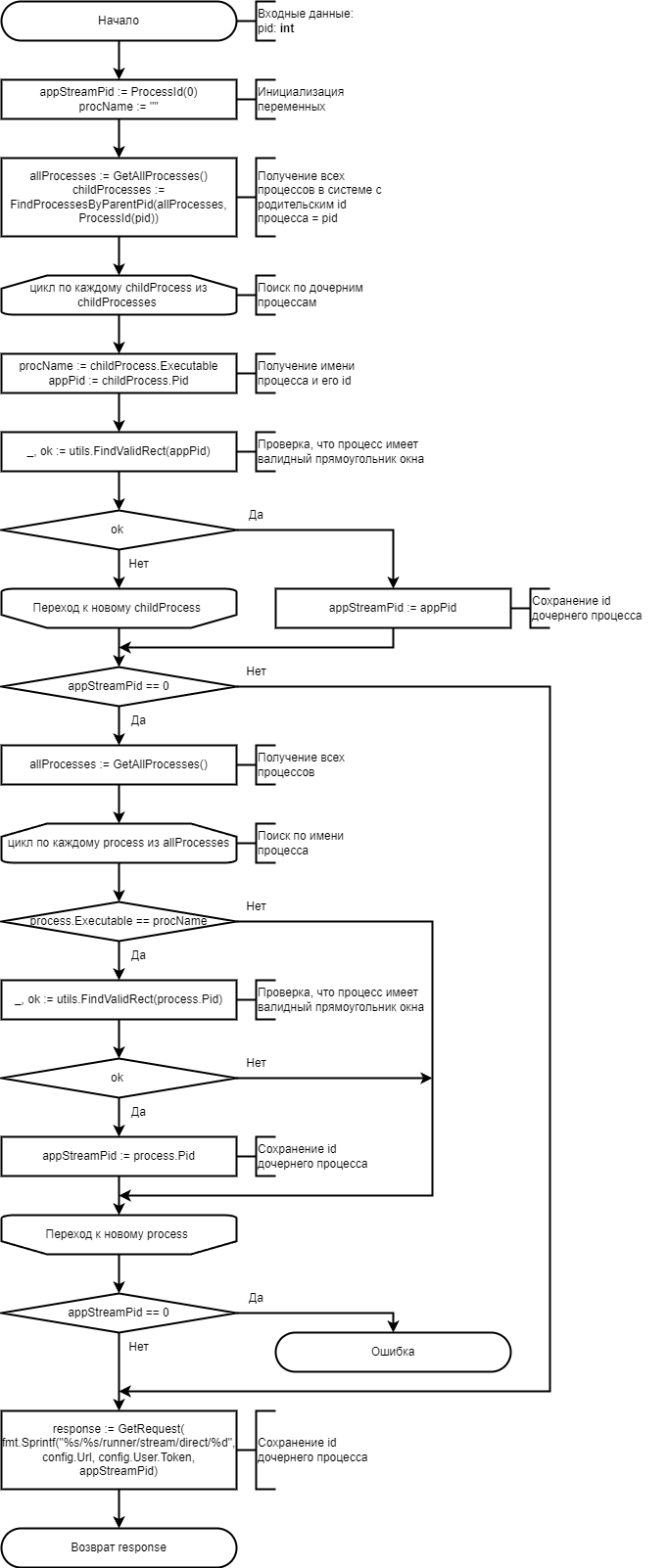


Рис.3.4.7.1 – Алгоритм создания стрима приложения

## Роль Router

### Сбор данных о файловой системе

### Отслеживание устаревших инстансов

### Репликация

#### Добавление задач репликации

#### Удаление задач репликации

### Фильтрация файлов

#### Добавление фильтра

#### Удаление фильтра

### Обработчик уведомлений

### Обработчик открытия/запуска файлов

## Аутентификации (Веб)

## Домашняя страница (Веб)

## Файловая система (Веб)

### Сбор данных о файловой системе

## Настройки (Веб)

### Фильтрация

### Репликация

# Разработка методики тестирования