# Arenadata™ Hadoop

*Версия - v2.1.3*

## Руководство пользователя по работе с кластером ADH

*2021 Arenadata™*

# Оглавление


**1 Руководство пользователя по работе с HDFS** **3**
- 1.1 Общий обзор . . . 3
- 1.2 Команды Shell . . . 4
- 1.3 Secondary NameNode . . . 8
- 1.4 Checkpoint Node . . . 8
- 1.5 Backup Node . . . 10
- 1.6 Import Checkpoint . . . 10
- 1.7 Balancer . . . 11
- 1.8 Rack Awareness . . . 12
- 1.9 Безопасный режим . . . 12
- 1.10 Команда fsck . . . 12
- 1.11 Команда fetchdt . . . 13
- 1.12 Recovery Mode . . . 14
- 1.13 Upgrade и Rollback . . . 14
- 1.14 DataNode Hot Swap Drive . . . 15
- 1.15 Права доступа к файлам и безопасность . . . 15
- 1.16 Масштабируемость . . . 15

**2 Руководство пользователя по работе с YARN** **16**

**3 Руководство по работе с Apache Zeppelin** **18**
- 3.1 Общий обзор . . . 18
- 3.2 Установка интерпретатора . . . 21
- 3.3 Конфигурация Apache Zeppelin . . . 23
- 3.4 Аутентификация Apache Shiro для Apache Zeppelin . . . 30
- 3.5 Интерфейс Apache Zeppelin . . . 35
- 3.6 Интерпретаторы в Apache Zeppelin . . . 41
- 3.7 Python 2 & 3 Интерпретатор для Apache Zeppelin . . . 45

**Important:** Контактная информация службы поддержки – e-mail: info@arenadata.io

# Глава 1

# Руководство пользователя по работе с HDFS

Руководство является отправной точкой для пользователей, работающих с распределенной файловой системой **HDFS** (Hadoop Distributed File System) либо в составе кластера **Hadoop**, либо в качестве автономной распределенной файловой системы общего назначения. Несмотря на то, что **HDFS** предназначена для работы в разных средах без особой настройки, практические знания о файловой системе могут значительно улучшить эксплуатационные свойства и диагностику конкретного кластера.

Руководство может быть полезно администраторам, программистам, разработчикам и сотрудникам подразделений информационных технологий, осуществляющих внедрение и эксплуатацию кластера.

**Important:** Контактная информация службы поддержки – e-mail: info@arenadata.io

## 1.1 Общий обзор

**HDFS** – это основное распределенное хранилище, используемое приложениями **Hadoop**. Кластер **HDFS** в основном состоит из NameNode, который управляет метаданными файловой системы, и DataNode, в которых хранятся фактические данные. Диаграмма архитектуры **HDFS** отображает основные взаимодействия между NameNode, DataNodes и клиентами: клиенты связываются с NameNode для изменения метаданных файла и выполняют фактический ввод-вывод файла непосредственно с помощью DataNodes.

Далее приведены некоторые из основных функций, которые могут представлять интерес для пользователей:

- Hadoop, включая HDFS, хорошо подходит для распределенного хранения и распределенной обработки: он отказоустойчив, масштабируем и чрезвычайно прост в расширении. MapReduce, хорошо известный своей простотой и применимостью для распределенных приложений, является неотъемлемой частью Hadoop.
- HDFS легко настраивается с конфигурацией по умолчанию, которая подходит для многих установок. Но при этом в большинстве случаев в зависимости от специфики решаемых задач конфигурацию необходимо настраивать.
- Hadoop написан на Java и поддерживается на всех основных платформах.
- Hadoop поддерживает shell-подобные команды для непосредственного взаимодействия с HDFS.
- NameNode и Datanodes имеют встроенные веб-интерфейсы, которые позволяют легко проверять текущее состояние кластера.

- Новые функции и улучшения регулярно внедряются в HDFS. Ниже приведен перечень полезных функций HDFS:
  - Права доступа к файлам и аутентификация;
  - Rack awareness: учет физического местоположения узла при планировании задач и выделении хранилища;
  - Safemode: административный режим для сопровождения;
  - fsck: утилита для диагностики работоспособности файловой системы, поиска отсутствующих файлов или блоков;
  - fetchdt: утилита для извлечения DelegationToken и сохранения его в файле в локальной системе;
  - Balancer: инструмент для балансировки кластера, когда данные неравномерно распределены между DataNodes;
  - Upgrade и rollback: после обновления программного обеспечения можно выполнить откат до состояния HDFS перед обновлением в случае непредвиденных проблем;
  - Secondary NameNode: выполняет периодические checkpoints пространства имен и помогает поддерживать размер файла, содержащего журнал изменений HDFS, в определенных пределах в NameNode;
  - Checkpoint node: выполняет периодические checkpoints пространства имен и помогает минимизировать размер журнала с изменениями HDFS, хранящийся в NameNode. NameNode позволяет использовать несколько узлов Checkpoint одновременно, если в системе не зарегистрированы резервные узлы;
  - Backup node: расширение узла Checkpoint. В дополнение к checkpointing он также получает поток изменений от NameNode и поддерживает свою собственную копию пространства имен в памяти, которая всегда синхронизируется с активным состоянием пространства имен NameNode. Только один резервный узел может быть зарегистрирован с помощью NameNode одновременно.

NameNode и DataNode запускают внутренний веб-сервер для отображения базовой информации о текущем состоянии кластера. В конфигурации по умолчанию главная страница NameNode находится по адресу http://namenode-name:9870/. На ней перечислены узлы данных в кластере и основные статистические данные кластера. Веб-интерфейс также можно использовать для просмотра файловой системы, используя ссылку “Browse the file system” на главной странице NameNode.

Права доступа к файлам в системе схожи с правами доступа на других известных платформах, таких как Linux. В настоящее время безопасность ограничивается простыми разрешениями на доступ к файлам. Пользователь, запускающий NameNode, рассматривается как суперпользователь для **HDFS**, однако, будущие версии **HDFS** будут поддерживать сетевые протоколы аутентификации, такие как **Kerberos**, для аутентификации пользователей и шифрования передачи данных.

**HDFS** имеет один NameNode для каждого кластера. В настоящее время общий объем памяти, доступный на NameNode, является основным ограничением масштабируемости. В очень больших кластерах увеличение среднего размера файлов, хранящихся в **HDFS**, помогает увеличить размер кластера без увеличения требований к памяти для NameNode.

## 1.2 Команды Shell

**Hadoop** включает в себя различные shell-подобные команды, напрямую взаимодействующие с **HDFS** и другими файловыми системами, которые поддерживает **Hadoop**. Команда `bin/hdfs dfs -help` выводит список команд, поддерживаемых оболочкой **Hadoop**, а `bin/hdfs dfs -help command-name` отображает более подробную справку. Команды поддерживают большинство обычных операций файловой системы, таких как копирование файлов, изменение прав доступа к файлам и т.д. Но также поддерживаются некоторые специфические операции **HDFS**, например, изменение репликации файлов.

Команда `bin/hdfs dfsadmin` поддерживает операции, связанные с администрированием **HDFS**, а `bin/hdfs dfsadmin -help` выводит список всех поддерживаемых в данный момент команд, например:

- `-report` – сообщает основную статистику HDFS. Некоторые из этих сведений также доступны на главной странице NameNode;
- `-safemode` – хотя обычно это не требуется, администратор может вручную войти или покинуть режим Safemode;
- `-finalizeUpgrade` – удаляет предыдущую резервную копию кластера, созданную во время последнего обновления;
- `-refreshNodes` – обновляет интерфейс NameNode с набором узлов DataNode, разрешенным подключаться к NameNode. По умолчанию NameNodes повторно считывает имена узлов DataNode в файле, определенном dfs.hosts и dfs.hosts.exclude. Хосты в dfs.hosts – это узлы данных, которые являются частью кластера. Если в dfs.hosts есть записи, то только хосты в нем могут регистрироваться с помощью NameNode. Записи в dfs.hosts.exclude – это узлы данных, которые должны быть выведены из эксплуатации. Альтернативно, если для *dfs.namenode.hosts.provider.classname* задано значение *org.apache.hadoop.hdfs.server.blockmanagement.CombinedHostFileManager*, все хосты include и exclude указываются в файле JSON, заданном dfs.hosts. Вывод из эксплуатации завершает DataNodes, когда все их записи среплицированы в другие DataNodes, при этом списанные узлы не выключаются автоматически и не выбираются для записи новых реплик;
- `-printTopology` – вывод топологии кластера. Отображение дерева стоек и узлов данных, прикрепленных к трекам, как показано в NameNode.

Пример использования:

```
hdfs dfsadmin [-report [-live] [-dead] [-decommissioning] [-enteringmaintenance] [-inmaintenance]]
hdfs dfsadmin [-safemode enter | leave | get | wait | forceExit]
hdfs dfsadmin [-saveNamespace [-beforeShutdown]]
hdfs dfsadmin [-rollEdits]
hdfs dfsadmin [-restoreFailedStorage true |false |check]
hdfs dfsadmin [-refreshNodes]
hdfs dfsadmin [-setQuota <quota> <dirname>...<dirname>]
hdfs dfsadmin [-clrQuota <dirname>...<dirname>]
hdfs dfsadmin [-setSpaceQuota <quota> [-storageType <storagetype>] <dirname>...<dirname>]
hdfs dfsadmin [-clrSpaceQuota [-storageType <storagetype>] <dirname>...<dirname>]
hdfs dfsadmin [-finalizeUpgrade]
hdfs dfsadmin [-rollingUpgrade [<query> |<prepare> |<finalize>]]
hdfs dfsadmin [-upgrade [query | finalize]
hdfs dfsadmin [-refreshServiceAcl]
hdfs dfsadmin [-refreshUserToGroupsMappings]
hdfs dfsadmin [-refreshSuperUserGroupsConfiguration]
hdfs dfsadmin [-refreshCallQueue]
hdfs dfsadmin [-refresh <host:ipc_port> <key> [arg1..argn]]
hdfs dfsadmin [-reconfig <namenode|datanode> <host:ipc_port> <start |status |properties>]
hdfs dfsadmin [-printTopology]
hdfs dfsadmin [-refreshNamenodes datanodehost:port]
hdfs dfsadmin [-getVolumeReport datanodehost:port]
hdfs dfsadmin [-deleteBlockPool datanode-host:port blockpoolId [force]]
hdfs dfsadmin [-setBalancerBandwidth <bandwidth in bytes per second>]
hdfs dfsadmin [-getBalancerBandwidth <datanode_host:ipc_port>]
hdfs dfsadmin [-fetchImage <local directory>]
hdfs dfsadmin [-allowSnapshot <snapshotDir>]
hdfs dfsadmin [-disallowSnapshot <snapshotDir>]
hdfs dfsadmin [-shutdownDatanode <datanode_host:ipc_port> [upgrade]]
hdfs dfsadmin [-evictWriters <datanode_host:ipc_port>]
hdfs dfsadmin [-getDatanodeInfo <datanode_host:ipc_port>]
hdfs dfsadmin [-metasave filename]
```

```
hdfs dfsadmin [-triggerBlockReport [-incremental] <datanode_host:ipc_port>]
hdfs dfsadmin [-listOpenFiles [-blockingDecommission] [-path <path>]]
hdfs dfsadmin [-help [cmd]]
```

- `-report [-live] [-dead] [-decommissioning] [-enteringmaintenance] [-inmaintenance]` – сообщает основную информацию и статистику файловой системы. Использование *dfs* может отличаться от *du*, поскольку оно измеряет необработанное пространство, используемое репликацией, контрольными суммами, снапшотами и т.д. на всех DN. Дополнительные флаги можно применять для фильтрации списка отображаемых DataNodes;
- `-safemode enter|leave|get|wait|forceExit` – команда обслуживания безопасного режима Safe mode – состояние Namenode, в котором он:
  - Не принимает изменения в пространстве имен (только для чтения);
  - Не копирует и не удаляет блоки.

  Безопасный режим вводится автоматически при запуске Namenode и автоматически выключается, когда настроенный минимальный процент блоков удовлетворяет условию минимальной репликации. Если Namenode обнаруживает какую-либо аномалию, он остается в безопасном режиме, пока проблема не будет решена. Если эта аномалия является следствием преднамеренного действия, администратор может использовать команду `-safemode forceExit` для принудительного выхода из безопасного режима. Случаи, когда это может потребоваться:
  - Метаданные Namenode не согласованы. Если Namenode обнаруживает, что метаданные были изменены вне диапазона и могут привести к потере данных, то Namenode переходит в состояние *forceExit*. В этот момент пользователь может либо перезапустить Namenode с правильными файлами метаданных, либо использовать *forceExit* (если потеря данных допустима);
  - Откат приводит к тому, что метаданные заменяются, и в редких случаях он может вызвать режим принудительного выхода из безопасного режима в Namenode.

  Безопасный режим можно ввести вручную, но тогда отключить его можно будет тоже только вручную;
- `-saveNamespace [-beforeShutdown]` – сохранение текущего пространства имен в каталогах хранилища и сброс журнала изменений. Требуется безопасный режим. Если задана опция *beforeShutdown*, NameNode делает контрольную точку тогда и только тогда, когда в течение временного окна контрольная точка не была установлена (настраиваемое количество периодов контрольных точек). Обычно функция используется перед закрытием NameNode для предотвращения возможного повреждения fsimage или журнала изменений;
- `-rollEdits` – откат журнала редактирования на активном NameNode;
- `-restoreFailedStorage true|false|check` – включение/выключение автоматической попытки восстановления неудачных реплик хранилища. Если сбойное хранилище снова станет доступным, система попытается восстановить журнал изменений и/или fsimage во время контрольной точки. Опция *check* возвращает текущую настройку;
- `-refreshNodes` – повторное чтение хостов и исключение файлов для обновления набора Datanodes, которым разрешено подключаться к Namenode, и тех, которые должны быть выведены из эксплуатации или повторно введены;
- `-finalizeUpgrade` – завершение обновления HDFS. Datanodes удаляют свои рабочие каталоги предыдущей версии, после чего Namenode делает то же самое. На этом процесс обновления завершается;
- `-upgrade query|finalize` – query: запрос текущего состояния обновления; finalize: завершить обновление HDFS (эквивалентно *finalizeUpgrade*);
- `-refreshServiceAcl` – перезагрузка файла политики авторизации на уровне сервиса;
- `-refreshUserToGroupsMappings` – обновить сопоставления пользователей и групп;

- `-refreshSuperUserGroupsConfiguration` – обновить сопоставления proxy-групп суперпользователя;
- `-refreshCallQueue` – перезагрузить очередь вызовов из конфига;
- `-refresh <host:ipc_port> <key> [arg1..argn]` – запуск обновления во время выполнения ресурса, указанного <key> на <host: ipc_port>. Все остальные аргументы после отправляются на хост;
- `-reconfig <datanode |namenode> <host:ipc_port> <start|status|properties>` – запуск реконфигурации, либо получение статуса текущей реконфигурации, либо получение списка реконфигурируемых свойств. Второй параметр указывает тип узла;
- `-printTopology` – отобразить дерево стоек и их узлов, как передается в Namenode;
- `-refreshNamenodes datanodehost:port` – перезагрузка файлов конфигурации для указанного datanode, прекращение обслуживания удаленных пулов блоков и старт обслуживания новых пулов блоков;
- `-getVolumeReport datanodehost:port` – получить отчет об объеме для указанного datanode;
- `-deleteBlockPool datanode-host:port blockpoolId [force]` – при принудительном вводе каталог пула блоков для указанного идентификатора блока данных на указанном datanode удаляется вместе с его содержимым, в противном случае каталог удаляется, только если он пуст. Команда не будет выполнена, если datanode все еще обслуживает пул блоков. Для выключения сервиса пула блоков на datanode использовать *refreshNamenodes*;
- `-setBalancerBandwidth <bandwidth in bytes per second>` – изменение пропускной способности сети, используемой каждым datanode во время балансировки блоков HDFS. <bandwidth> – максимальное число байтов в секунду, которое будет использоваться каждой datanode. Это значение переопределяет параметр *dfs.datanode.balance.bandwidthPerSec*. При этом новое значение не является постоянным в узле DataNode;
- `-getBalancerBandwidth <datanode_host:ipc_port>` – получение пропускной способности сети (в байтах в секунду) для указанного datanode. Это максимальная пропускная способность сети, используемая datanode при балансировке блоков HDFS;
- `-fetchImage <local directory>` – загрузка последнего fsimage из NameNode и сохранение его в указанном локальном каталоге;
- `-allowSnapshot <snapshotDir>` – разрешение на создание снапшотов каталога. Если операция завершается успешно, каталог становится моментальным снимком;
- `-disallowSnapshot <snapshotDir>` – запрет на создание снапшотов каталога, который будет создан. Все снимки каталога должны быть удалены перед включением функции;
- `-shutdownDatanode <datanode_host:ipc_port> [upgrade]` – отправить запрос на отключение для указанного datanode;
- `-evictWriters <datanode_host:ipc_port>` – заставляет datanode выселить всех клиентов, которые пишут блок. Функция полезна, когда эксплуатация приостановлена из-за медленных писателей;
- `-getDatanodeInfo <datanode_host:ipc_port>` – получение информации о указанном datanode;
- `-metasave filename` – сохранение основных структур данных Namenode в *filename* в каталоге, указанном свойством *hadoop.log.dir*. Файл *filename* перезаписывается, если уже существует. При этом *filename* содержит одну строку для каждого из следующих:
  - Сообщения heartbeat узлов Datanodes с Namenode;
  - Ожидающие репликации блоки;
  - Копируемые в настоящее время блоки;
  - Ожидающие удаления блоки.
- `-triggerBlockReport [-incremental] <datanode_host:ipc_port>` – запуск отчета о блокировке для указанного datanode. Если указано значение *incremental*, то отчет о полном блоке;

- `-listOpenFiles [-blockingDecommission] [-path <path>]` – список всех открытых файлов, которыми в данный момент управляет NameNode, а также имя клиента и клиентский компьютер, к которому они обращаются. Список открытых файлов фильтруется по заданному типу и пути;
- `-help [cmd]` – справка для указанной команды или для всех команд, если ни одна не указана.

## 1.3 Secondary NameNode

NameNode хранит изменения в файловой системе в виде журнала изменений, добавляемого к родному файлу. Когда NameNode запускается, сервис считывает состояние **HDFS** из файла образа, fsimage, а затем применяет изменения из файла журнала. После чего он записывает новое состояние **HDFS** в fsimage и начинает обычную работу с пустым файлом правок. Поскольку NameNode объединяет fsimage и редактирует файлы только во время запуска, файл журнала изменений может со временем стать очень большим. Другим побочным эффектом большего файла журнала изменений является то, что следующий перезапуск NameNode занимает больше времени.

В свою очередь Secondary NameNode время от времени объединяет файлы журнала изменений с fsimage и сохраняет размер журнала в пределах лимита. Поскольку требования к памяти у вторичного и основного NameNode одинаковы, Secondary NameNode обычно запускается на другом узле, чем NameNode.

Запуск процесса контрольной точки на Secondary NameNode управляется двумя параметрами конфигурации:

- `dfs.namenode.checkpoint.period` – по умолчанию установлено значение на 1 час; определяет максимальную задержку между двумя последовательными контрольными точками;
- `dfs.namenode.checkpoint.txns` – по умолчанию установлено значение на 1 миллион; определяет количество незарегистрированных транзакций на NameNode, которые будут вызывать следующую контрольную точку, даже если период ее не был достигнут.

Secondary NameNode хранит последнюю контрольную точку в каталоге, который структурирован таким же образом, как и каталог основного NameNode. Получается, образ с контрольной точки всегда готов к считыванию активным NameNode при необходимости.

Пример использования:

```
hdfs secondarynamenode [-checkpoint [force]] | [-format] | [-geteditsize]
```

- `-checkpoint [force]` – контрольные точки SecondaryNameNode, если *EditLog size*> = *fs.checkpoint.size*. Если используется принудительно, контрольная точка не зависит от размера EditLog;
- `-format` – форматирование локального хранилища во время запуска;
- `-geteditsize` – вывод количества незарегистрированных транзакций на NameNode.

## 1.4 Checkpoint Node

NameNode сохраняет свое пространство имен с помощью двух файлов: fsimage, который является последней контрольной точкой пространства имен, и журнал изменений (log) в пространстве имен с момента контрольной точки. Когда NameNode запускается, он объединяет fsimage и журнал изменений, чтобы обеспечить актуальное представление метаданных файловой системы. Затем NameNode перезаписывает fsimage с новым состоянием **HDFS** и начинает новый журнал изменений.

Checkpoint Node периодически создает контрольные точки пространства имен. Он загружает fsimage и изменения из активного NameNode, объединяет их локально и загружает новое изображение обратно в активный NameNode. Checkpoint Node обычно работает на компьютере, отличном от NameNode, поскольку их требования к памяти одинаковы. Запуск Checkpoint Node осуществляется командой `bin/hdfs namenode -checkpoint` на указанном в конфигурационном файле узле.

Расположение Checkpoint Node (или Backup) и сопровождающего его веб-интерфейса настраивается с помощью переменных конфигурации *dfs.namenode.backup.address* и *dfs.namenode.backup.http-address*.

Запуск процесса контрольной точки на Checkpoint Node контролируется двумя параметрами конфигурации:

- `dfs.namenode.checkpoint.period` – по умолчанию установлено значение на 1 час; определяет максимальную задержку между двумя последовательными контрольными точками;
- `dfs.namenode.checkpoint.txns` – по умолчанию установлено значение на 1 миллион; определяет количество незарегистрированных транзакций на NameNode, которые будут вызывать срочную контрольную точку, даже если период ее не был достигнут.

Checkpoint Node хранит последнюю контрольную точку в каталоге, который структурирован таким же образом, как и каталог NameNode. Таким образом контрольное изображение всегда доступно для чтения из NameNode при необходимости.

**Important:** В файле конфигурации кластера можно указать несколько Checkpoint Node

Пример использования:

```
hdfs namenode [-backup] |
      [-checkpoint] |
      [-format [-clusterid cid ] [-force] [-nonInteractive] ] |
      [-upgrade [-clusterid cid] [-renameReserved<k-v pairs>] ] |
      [-upgradeOnly [-clusterid cid] [-renameReserved<k-v pairs>] ] |
      [-rollback] |
      [-rollingUpgrade <rollback |started> ] |
      [-importCheckpoint] |
      [-initializeSharedEdits] |
      [-bootstrapStandby [-force] [-nonInteractive] [-skipSharedEditsCheck] ] |
      [-recover [-force] ] |
      [-metadataVersion ]
```

- `-backup` – запуск Backup Node;
- `-checkpoint` – запуск Checkpoint Node;
- `-format [-clusterid cid]` – форматирование указанного NameNode. Запускает NameNode, форматирует его и затем выключает. Будет генерировать NameNodeFormatException, если имя dir уже существует и если переформатирование для кластера отключено;
- `-upgrade [-clusterid cid] [-renameReserved <k-v pairs>]` – Namenode должен быть запущен с опцией обновления до новой версии Hadoop;
- `-upgradeOnly [-clusterid cid] [-renameReserved <k-v pairs>]` – обновляет и закрывает указанный NameNode;
- `-rollback` – откат NameNode до предыдущей версии. Следует использовать после остановки кластера и разворачивания старой версии Hadoop;
- `-rollingUpgrade <rollback|started>` – *query*: запрос текущего состояния обновления; *prepare*: подготовка нового обновления; *finalize*: завершение текущего обновления;
- `-importCheckpoint` – загрузка изображения из каталога контрольных точек и сохранение его в текущем. Контрольная точка dir читается из свойства *dfs.namenode.checkpoint.dir*;
- `-initializeSharedEdits` – форматирование нового общего каталога изменений и копирование его в достаточное количество сегментов журнала для запуска резервного NameNode;

- `-bootstrapStandby [-force] [-nonInteractive] [-skipSharedEditsCheck]` – позволяет загружать резервные каталоги хранилища NameNode путем копирования последнего снимка пространства имен из активного NameNode. Используется при первой настройке кластера высокой доступности. Опция *-force* или *-nonInteractive* имеет то же значение, что и *-format*. Опция *-skipSharedEditsCheck* пропускает проверку правок, которая гарантирует достаточное количество правок, уже находящихся в общем каталоге, для запуска с последней контрольной точки;
- `-recover [-force]` – восстановление потерянных метаданных в поврежденной файловой системе;
- `-metadataVersion` – печать версии метаданных программного обеспечения и образа (при условии наличия настроенных каталогов).

## 1.5 Backup Node

Backup Node обеспечивает ту же функциональность контрольных точек, что и Checkpoint Node, а также поддерживает в памяти обновленную копию пространства имен файловой системы, которая всегда синхронизируется с активным состоянием NameNode. Наряду с принятием потока изменений файловой системы журнала из NameNode и сохранением его на диске, Backup Node также применяет эти изменения к своей собственной копии пространства имен в памяти, создавая таким образом резервную копию пространства имен.

Backup Node не нужно загружать fsimage и редактировать файлы с активного NameNode для создания контрольной точки, как это требуется для Checkpoint Node или Secondary NameNode, поскольку Backup Node уже имеет актуальный статус состояния пространства имен в памяти. Процесс создания контрольной точки Backup узла является более эффективным, поскольку ему нужно только сохранить пространство имен в локальном файле fsimage и сбросить изменения.

Поскольку Backup Node поддерживает копию пространства имен в памяти, его требования к RAM такие же, как и у узла NameNode.

---

**Important:** NameNode поддерживает один Backup Node. Узлы Checkpoint не могут быть зарегистрированы, если используется Backup

---

Backup Node настраивается так же, как узел Checkpoint, начиная с `bin/hdfs namenode -backup`.

Расположение Backup Node (или Checkpoint) и его веб-интерфейса настраивается с помощью переменных конфигурации *dfs.namenode.backup.address* и *dfs.namenode.backup.http-address*.

Использование Backup Node обеспечивает возможность запуска NameNode без постоянного хранилища, делегируя всю ответственность за сохранение состояния пространства имен Backup Node. Для этого необходимо запустить NameNode с параметром `-importCheckpoint`, а также не указывать постоянные каталоги хранения типа edits *dfs.namenode.edits.dir* в конфигурации NameNode.

Пример использования приведен в главе *Checkpoint Node*.

## 1.6 Import Checkpoint

В случае потери всех копий образов и файлов транзакций в NameNode может быть импортирована последняя контрольная точка. Для этого необходимо:

- Создать пустой каталог, указанный в переменной конфигурации *dfs.namenode.name.dir*;
- Указать местоположение каталога контрольных точек в переменной конфигурации *dfs.namenode.checkpoint.dir*;
- Запустить NameNode с параметром `-importCheckpoint`.

При этом NameNode загружает контрольную точку из каталога *dfs.namenode.checkpoint.dir*, а затем сохраняет ее в каталог NameNode, заданный в *dfs.namenode.name.dir*. Если в *dfs.namenode.name.dir* содержится допустимый образ, NameNode выдает ошибку. NameNode проверяет консистентность образа в *dfs.namenode.checkpoint.dir*, но не изменяет его каким-либо образом.

Пример использования приведен в главе *Checkpoint Node*.

## 1.7 Balancer

Данные **HDFS** не всегда могут быть размещены равномерно по всем DataNode. Одной из распространенных причин является добавление новых DataNodes в существующий кластер. При размещении новых блоков (данные для файла хранятся в виде серии блоков) NameNode учитывает различные параметры, прежде чем выбирать узлы DataNodes для получения этих блоков. Некоторые из них:

- Политика для хранения одной из реплик блока на том же узле, что и узел, который записывает блок;
- Необходимо распределить различные реплики блока по стойкам, чтобы кластер мог пережить потерю всей стойки;
- Одна из реплик обычно размещается в той же стойке, что и узел, выполняющий запись в файл, что снижает количество операций ввода-вывода между стойками;
- Данные HDFS распределяются равномерно по узлам DataNodes в кластере.

Из-за множества конкурирующих соображений данные могут быть неравномерно размещены между узлами DataNodes. **HDFS** предоставляет инструмент для администраторов, который анализирует размещение блоков и осуществляет перебалансировку данных через DataNode.

Пример использования:

```
hdfs balancer
      [-policy <policy>]
      [-threshold <threshold>]
      [-exclude [-f <hosts-file> | <comma-separated list of hosts>]]
      [-include [-f <hosts-file> | <comma-separated list of hosts>]]
      [-source [-f <hosts-file> | <comma-separated list of hosts>]]
      [-blockpools <comma-separated list of blockpool ids>]
      [-idleiterations <idleiterations>]
      [-runDuringUpgrade]
```

- `-policy <policy>` – *datanode* (по умолчанию): кластер сбалансирован, если каждый datanode сбалансирован; *blockpool*: кластер сбалансирован, если каждый пул блоков в каждой datanode сбалансирован;
- `-threshold <threshold>` – процент емкости диска; перезапись порога по умолчанию;
- `-exclude -f <hosts-file> | <comma-separated list of hosts>` – исключение указанных datanode из балансировки;
- `-include -f <hosts-file> | <comma-separated list of hosts>` – включение только указанных datanode для балансировки;
- `-source -f <hosts-file> | <comma-separated list of hosts>` – выбор только указанных datanode в качестве исходных узлов;
- `-blockpools <comma-separated list of blockpool ids>` – работа балансировщика только на blockpools, включенных в этот список;
- `-idleiterations <iterations>` – максимальное количество холостых итераций перед выходом; перезапись количества итераций по умолчанию (*5*);

- `-runDuringUpgrade` – следует ли запускать балансировщик во время текущего обновления HDFS. Действие не влияет на используемое пространство на перегруженных машинах, поэтому нежелательно;
- `-h|--help` – показывает используемый инструмент и справочную информацию.

Для остановки процесса перебалансировки администратор может просто нажать комбинацию клавиш *Ctrl-C*.

**Important:** Политика blockpool является более строгой, чем политика datanode

Помимо указанных параметров введена функция закрепления (pinning feature) для предотвращения перемещения определенных реплик балансировщиком или mover. По умолчанию функция отключена и может быть включена с помощью свойства конфигурации *dfs.datanode.block-pinning.enabled*. В включенном состоянии она влияет только на блоки, которые записываются в указанные в вызове `create()` узлы. Функция полезна при необходимости локального сохранения данных для таких приложений, как **HBase regionserver**.

## 1.8 Rack Awareness

Кластер **HDFS** может распознавать топологию стоек, в которых размещены все узлы. Важно настроить эту топологию, чтобы оптимизировать емкость и использование данных.

Детальная информация приведена по ссылке rack awareness.

## 1.9 Безопасный режим

Во время запуска NameNode загружает состояние файловой системы из fsimage и файла журнала изменений. Затем чтобы не начать преждевременную репликацию блоков, он ожидает, когда DataNodes сообщат о своих блоках, не смотря на то, что в кластере уже существует достаточное количество реплик. В течение этого времени NameNode остается в безопасном режиме (Safemode). Безопасный режим для NameNode по существу является режимом только для чтения для кластера **HDFS**, где он не допускает каких-либо изменений в файловой системе или блоках. Обычно NameNode выходит из Safemode автоматически после того, как узлы DataNodes сообщают, что большинство блоков файловой системы доступно.

При необходимости система **HDFS** может быть размещена в безопасном режиме явно с помощью команды `bin/hdfs dfsadmin -safemode`. Включен ли режим Safemode отображается на первой странице NameNode.

## 1.10 Команда fsck

**HDFS** поддерживает команду **fsck** для проверки различных несоответствий. Команда предназначена для сообщения о проблемах с различными файлами, например, об отсутствующих блоках для файла или недостаточно реплицированных блоках. Но в отличие от традиционной утилиты **fsck** для собственных файловых систем, эта команда не исправляет обнаруженные ошибки.

Обычно NameNode автоматически исправляет большинство сбоев. По умолчанию **fsck** игнорирует открытые файлы, но предоставляет возможность выбора всех файлов во время создания отчетов. Команда **fsck** системы **HDFS** не является командой оболочки **Hadoop**, она может быть запущена путем `bin/hdfs fsck`.

Команда **fsck** может быть запущена по всей файловой системе или только на подмножестве файлов.

Пример использования:

```
hdfs fsck <path>
       [-list-corruptfileblocks |
       [-move | -delete | -openforwrite]
       [-files [-blocks [-locations | -racks | -replicaDetails | -upgradedomains]]]
```

```
      [-includeSnapshots] [-showprogress]
      [-storagepolicies] [-maintenance]
      [-blockId <blk_Id>]
```

- `path` – начало проверки с указанного пути;
- `-delete` – удаление поврежденных файлов;
- `-files` – отобразить проверяемые файлы;
- `-files -blocks` – отобразить отчет о блокировке;
- `-files -blocks -locations` – отобразить расположение каждого блока;
- `-files -blocks -racks` – отобразить топологию сети для расположения узлов данных;
- `-files -blocks -replicaDetails` – отобразить каждую копию реплики;
- `-files -blocks -upgradedomains` – отобразить обновления доменов для каждого блока;
- `-includeSnapshots` – включить данные снапшотов, если указанный путь указывает на каталог снапшотов или под ним есть каталоги снапшотов;
- `-list-corruptfileblocks` – отобразить список отсутствующих блоков и файлов, к которым они принадлежат;
- `-move` – переместить поврежденные файлы в */lost+found*;
- `-openforwrite` – отобразить файлы, открытые для записи;
- `-showprogress` – отобразить точки для прогресса в выводе; по умолчанию выключено (без прогресса);
- `-storagepolicies` – отобразить сводную политику хранения для блоков;
- `-maintenance` – отобразить детали узла;
- `-blockId` – отобразить информацию о блоке.

## 1.11 Команда fetchdt

Для получения Delegation Token и сохранения его в файле в локальной системе **HDFS** поддерживает команду **fetchdt**. Этот токен впоследствии можно использовать для доступа к защищенному серверу (например, NameNode) с незащищенного клиента. Утилита использует RPC или HTTPS (через **Kerberos**) для получения токена и, следовательно, требует наличия тикетов **Kerberos** перед запуском (необходимо запустить **kinit** для получения тикетов). В **HDFS** команда **fetchdt** не является командой оболочки **Hadoop** и может быть запущена как `bin/hdfs fetchdt DTfile`. После получения токена можно запустить команду **HDFS** без тикетов **Kerberos**, указав переменную среды `HADOOP_TOKEN_FILE_LOCATION` в файл токена делегирования.

Пример использования:

```
hdfs fetchdt <opts> <token_file_path>
```

- `--webservice NN_Url` – URL для связи с NN (начинается с *http* или *https*);
- `--renewer name` – обновленное название токена делегирования;
- `--cancel` – отменить токен делегирования;
- `--renew` – обновить токен делегирования. Токен делегирования должен быть получен с использованием параметра *-renewer name*;
- `--print` – отобразить токен делегирования;
- `token_file_path` – путь к файлу для хранения токена.

## 1.12 Recovery Mode

Как правило, обычно настраивается несколько мест хранения метаданных. И в случае, если хранилище повреждается, метаданные можно прочитать из других хранилищ. Однако, что делать, если единственное доступное хранилище повреждено? Для таких ситуаций существует специальный режим запуска NameNode, называемый режимом восстановления – Recovery Mode, который позволяет восстановить большую часть данных.

Запуск NameNode в режиме восстановления осуществляется следующим образом: `namenode -recover`.

В режиме восстановления NameNode в командной строке интерактивно запрашивает о возможных действиях, которые необходимо предпринять для восстановления данных. Для того, чтобы не получать эти запросы, можно указать опцию `-force`, которая заставляет режим Recovery Mode всегда выбирать первый вариант действий, обычно являющийся самым разумным.

**Important:** Поскольку режим Recovery Mode может привести к потере данных, всегда необходимо перед его использованием делать резервную копию журнала изменений и fsimage

## 1.13 Upgrade и Rollback

При обновлении **Hadoop** на существующем кластере, как и при любом обновлении программного обеспечения, возможны новые ошибки или несовместимые изменения, которые влияют на существующие приложения и не были обнаружены ранее. Система позволяет администраторам вернуться к более ранней версии **Hadoop** и откатить кластер до состояния, в котором он находился до обновления. Но **HDFS** может иметь одну такую резервную копию. Перед обновлением администраторам необходимо удалить существующую резервную копию с помощью команды `bin/hadoop dfsadmin -finalizeUpgrade`. Далее кратко описывается типичная процедура обновления:

- Перед обновлением программного обеспечения Hadoop завершить работу, если имеется существующая резервная копия.
- Остановить кластер и распространить новую версию Hadoop.
- Запустить новую версию с параметром *-upgrade* (`sbin/start-dfs.sh -upgrade`).
- Завершить обновление, как только новая HDFS считается работающей (возможно, после нескольких дней эксплуатации). Важно обратить внимание, что пока кластер не будет завершен, удаление файлов, существовавших до обновления, не освобождает реальное дисковое пространство на узлах данных DataNodes.
- При необходимости вернуться к старой версии:
  - Остановить кластер и распространить более раннюю версию Hadoop;
  - Выполнить команду отката в namenode (`bin/hdfs namenode -rollback`);
  - Запустить кластер с возможностью отката (`sbin/start-dfs.sh -rollback`).

При обновлении **HDFS** до новой версии необходимо переименовать или удалить все пути, зарезервированные в новой версии **HDFS**, в ином случае, когда NameNode встречает зарезервированный путь во время обновления, выдает ошибку:

```
/.reserved is a reserved path and .snapshot is a reserved path component in this version of HDFS. Please␣
↪rollback and delete or rename this path, or upgrade with the -renameReserved [key-value pairs] option to␣
↪automatically rename these paths during upgrade.
```

Указание `-upgrade -renameReserved [optional key-value pairs]` заставляет NameNode автоматически переименовывать любые зарезервированные пути, найденные во время запуска. Например, чтобы переименовать

все пути с именами *.snapshot* на *.my-snapshot* и *.reserved* на *.my-reserved*, необходимо указать `-upgrade -renameReserved .snapshot=.my-snapshot,.reserved=.my-reserved`.

Если с ключом *-renameReserved* не указаны пары ключ-значение, NameNode добавляет суффиксы зарезервированным путям в виде *.<LAYOUT-VERSION>.UPGRADE_RENAMED*, например, `.snapshot.-51.UPGRADE_RENAMED`.

Есть некоторые оговорки к этому процессу переименования и по возможности рекомендуется перед обновлением сначала выполнять `hdfs dfsadmin -saveNamespace`. Это связано с тем, что может возникнуть несогласованность данных, если операция журнала изменений ссылается на место назначения автоматически переименованного файла.

## 1.14 DataNode Hot Swap Drive

Datanode поддерживает возможность горячей замены дисков, поэтому пользователь может добавлять или заменять узлы DataNode в **HDFS**, не выключая их. Далее кратко описана типичная процедура замены Hot Swap Drive:

- При наличии новых каталогов хранения пользователь должен отформатировать их и подключить соответствующим образом;
- Пользователь обновляет конфигурацию DataNode *dfs.datanode.data.dir*, чтобы отразить каталоги томов данных, которые будут использоваться;
- Пользователь запускает `dfsadmin -reconfig datanode HOST:PORT start` для инициализации процесса реконфигурации. Пользователь может использовать `dfsadmin -reconfig datanode HOST:PORT status` для запроса текущего состояния задачи реконфигурации;
- После завершения задачи реконфигурации пользователь может безопасно размонтировать удаленные каталоги томов данных и физически удалить диски.

## 1.15 Права доступа к файлам и безопасность

Права доступа к файлам схожи с правами доступа к файлам на других известных платформах, таких как **Linux**. В настоящее время безопасность ограничена простыми правами доступа к файлам. Пользователь, запускающий NameNode, для **HDFS** рассматривается как суперпользователь. Будущие версии **HDFS** будут поддерживать сетевые протоколы аутентификации, такие как Kerberos, для аутентификации пользователей и шифрования передачи данных.

## 1.16 Масштабируемость

В настоящее время **Hadoop** может работать на кластерах с тысячами узлов, где **HDFS** имеет по одному NameNode для каждого кластера. Таким образом общий объем памяти, доступный на NameNode, является основным ограничением масштабируемости. В очень больших кластерах увеличение среднего размера файлов, хранящихся в **HDFS**, способствует увеличению размера кластера без повышения требований к памяти для NameNode.

**Important:** Конфигурация по умолчанию может не подходить для очень больших кластеров

# Глава 2

# Руководство пользователя по работе с YARN

Руководство может быть полезно администраторам, программистам, разработчикам и сотрудникам подразделений информационных технологий, осуществляющих внедрение и эксплуатацию кластера.

**Important:** Контактная информация службы поддержки – e-mail: info@arenadata.io

Основная идея **YARN** состоит в том, чтобы разделить функции управления ресурсами и планирования/мониторинга заданий на отдельные демоны. Суть заключается в том, чтобы иметь общий ResourceManager и ApplicationMaster для каждого приложения. Приложение – это отдельная работа или группа работ.

ResourceManager и NodeManager образуют среду для вычисления данных (Рис.2.1.). ResourceManager – это высший орган, который распределяет ресурсы между всеми приложениями в системе. NodeManager – это агент инфраструктуры для каждой машины, который отвечает за контейнеры, отслеживает использование их ресурсов (процессор, память, диск, сеть) и сообщает об этом в ResourceManager/Scheduler.

ApplicationMaster для каждого приложения, по сути, является библиотекой, специфичной для платформы, и на него возложена задача согласования ресурсов из ResourceManager и работа с NodeManager(-ами) для выполнения и мониторинга задач.

ResourceManager имеет два основных компонента: Scheduler и ApplicationsManager.

Scheduler отвечает за распределение ресурсов между различными запущенными приложениями с учетом известных ограничений емкости, очередей и т.д. Scheduler является чистым планировщиком в том смысле, что он не выполняет никакого мониторинга или отслеживания состояния приложения. Кроме того, он не дает никаких гарантий относительно перезапуска сбойных задач из-за ошибок приложения или оборудования. Планировщик выполняет свою функцию планирования на основе требований к ресурсам приложений на базе абстрактного понятия *resource Container*, включающего в себя такие элементы, как память, процессор, диск, сеть и т.д.

Планировщик имеет подключаемую политику, которая отвечает за распределение ресурсов кластера между различными очередями, приложениями и т.д. Текущие планировщики, такие как CapacityScheduler и FairScheduler, являются некоторыми примерами плагинов.

ApplicationsManager отвечает за прием заявок, согласование первого контейнера для выполнения приложения определенным ApplicationMaster и предоставляет сервис для перезапуска контейнера ApplicationMaster при сбое. ApplicationMaster для каждого приложения отвечает за согласование подходящих *resource Container* с планировщиком, отслеживание их состояния и мониторинг прогресса.

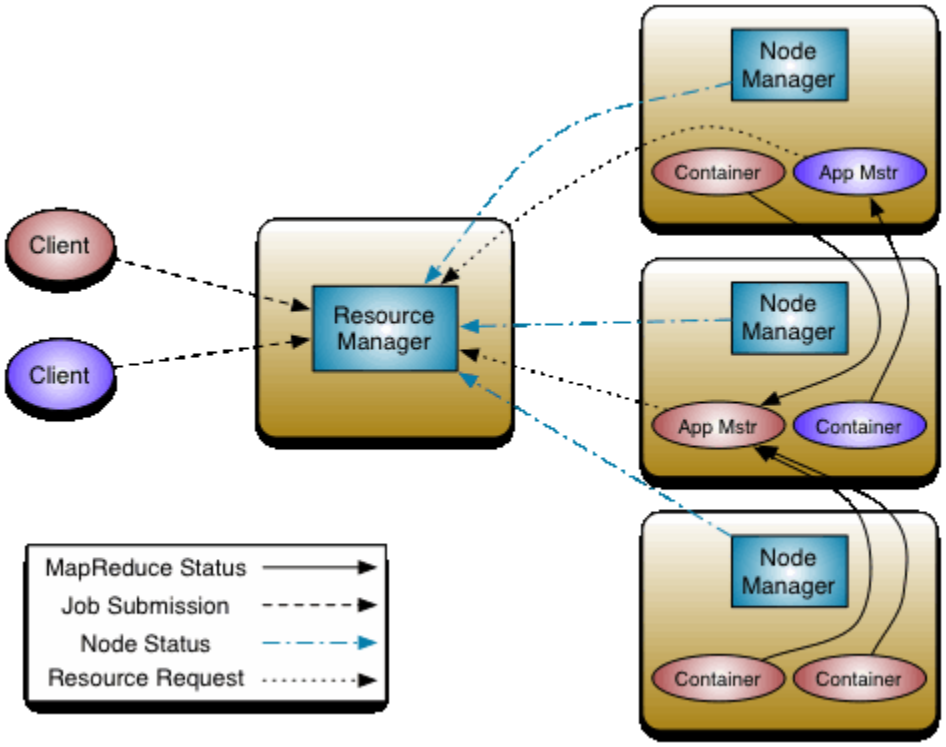



Рис.2.1.: Архитектура YARN

**MapReduce** в **hadoop-3.x** поддерживает API-совместимость с предыдущей стабильной версией (**hadoop-2.x**). Это означает, что все задания **MapReduce** по-прежнему выполняются поверх **YARN** только с учетом перекомпиляции.

**YARN** поддерживает понятие резервирования ресурсов через ReservationSystem – компонент, который позволяет пользователям определять профиль ресурсов over-time и задавать временные ограничения (сроки), а так же резервировать ресурсы для обеспечения предсказуемого выполнения важных заданий. ReservationSystem отслеживает ресурсы over-time, выполняет управление допуском для резервирования и динамично инструктирует базовый планировщик, чтобы гарантировать, что резервирование полностью выполнено.

С целью масштабирования **YARN** за пределы нескольких тысяч узлов поддерживается понятие Federation через функцию *YARN Federation*. Федерация позволяет прозрачно соединять несколько кластеров (подкластеров) *yarn* и делать их единым массивным кластером. Это применимо для достижения большего масштаба и/или для того, чтобы несколько независимых кластеров могли использоваться вместе для очень объемных заданий.

# Глава 3

# Руководство по работе с Apache Zeppelin

В документе приведено общее описание Apache Zeppelin, руководство по установке, конфигурации, использовании интерфейса и интерпретаторах.

Документ может быть полезен администраторам, программистам, разработчикам и сотрудникам подразделений информационных технологий, осуществляющих внедрение и сопровождение кластеров Arenadata Hadoop и Arenadata DB.

**Important:** Контактная информация службы поддержки – e-mail: info@arenadata.io

## 3.1 Общий обзор

**Открытый веб-блокнот для интерактивной аналитики данных**

**Apache Zeppelin** – это новый многофункциональный веб-блокнот, обеспечивающий считывание, анализ и визуализацию данных, их обмен и взаимодействие с **Hadoop** и **Spark** (Рис.3.1.).

Интерактивные браузерные блокноты позволяют инженерам данных, аналитикам и ученым в области данных более продуктивно выполнять работу, благодаря совместному использованию кода данных, его разработке, организации и выполнению, а также благодаря визуализации результатов без необходимости обращения к командной строке или к компонентам кластера. Блокноты обеспечивают пользователям не только выполнение задач, но и интерактивную работу с долго выполняющимися потоками операций.

**Apache Zeppelin** – это новый веб-блокнот, который предоставляет функции поиска, визуализации, совместного использования и функциональное взаимодействие с **Apache Spark**. В него встроена интеграция со **Spark**, что избавляет от необходимости создания отдельного модуля, плагина или библиотеки, и это дает следующие преимущества:

- Автоматическое создание *SparkContext* и *sqlcontext*;
- Загрузка jar-зависимостей из локальной файловой системы или репозитория *maven* во время выполнения задачи;
- Возможность отмены задания и отображение хода его выполнения.

**Apache Zeppelin** поддерживает **Python**, но при этом концепция интерпретатора блокнота позволяет подключать любой язык/фреймворк обработки данных в **Zeppelin**. В настоящее время **Zeppelin** поддерживает множество интерпретаторов, например, такие как **Scala**, **Hive**, **SparkSQL**, **Shell** и **Markdown** (Рис.3.2.).

Некоторые базовые диаграммы уже включены в **Apache Zeppelin**, но визуализация не ограничивается запросом **Spark SQL** и любой результат с любого языка может быть распознан и визуализирован (Рис.3.3.).

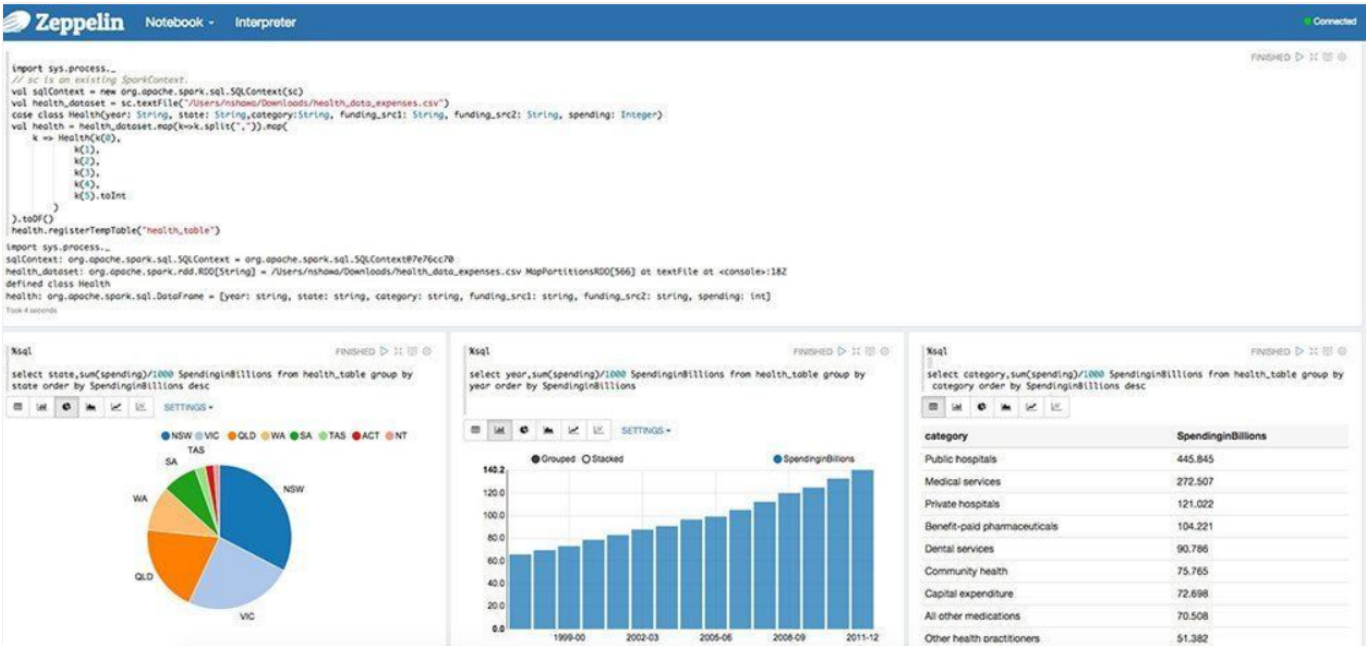



Рис.3.1.: Apache Zeppelin




Рис.3.2.: Интерпретаторы

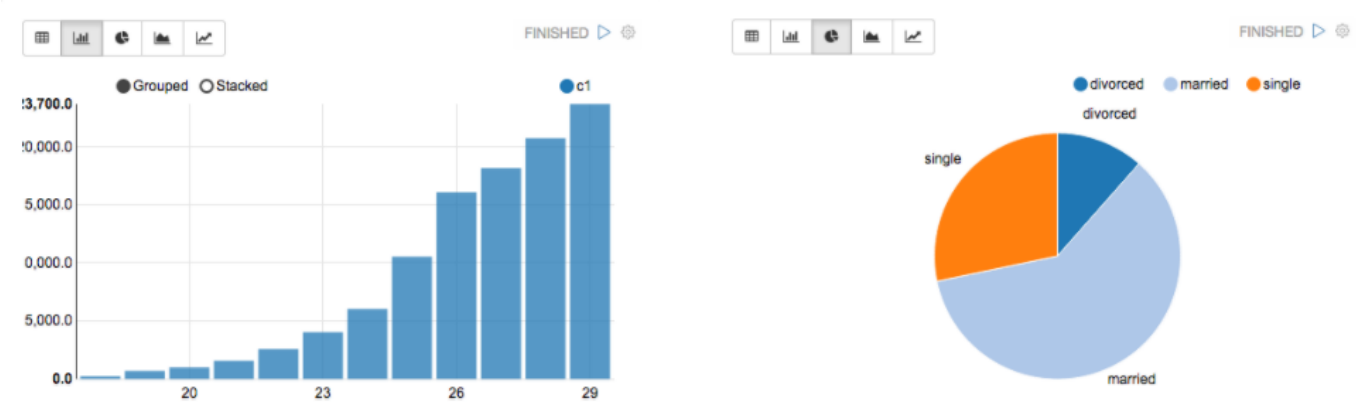



Рис.3.3.: Визуализация данных

**Apache Zeppelin** агрегирует значения и отображает их в сводной диаграмме с простым перемещением drag-and-drop. Можно легко создать диаграмму с несколькими агрегированными значениями, в том числе: сумма, количество, среднее, минимальное, максимальное (Рис.3.4.).

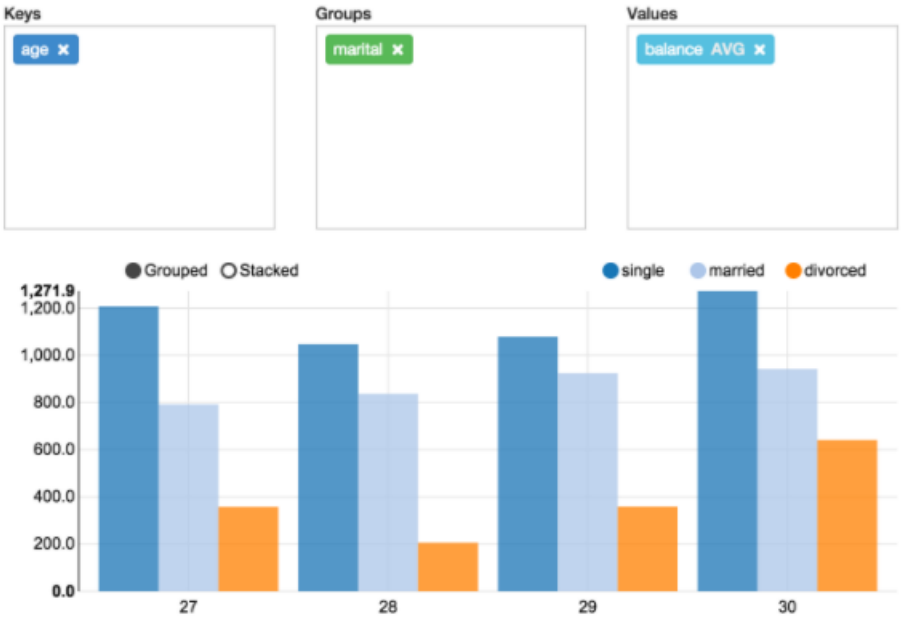



Рис.3.4.: Сводная диаграмма

Также **Apache Zeppelin** может динамически создавать некоторые формы ввода в блокноте пользователя (Рис.3.5.).

Поиск данных, их анализ, отчетность и визуализация являются ключевыми компонентами рабочего процесса в области данных. **Zeppelin** предоставляет “Modern Data Science Studio” (“Современную научную студию данных”), которая поддерживает **Spark** и **Hive** из коробки. Фактически **Zeppelin** поддерживает несколько языков, которые в свою очередь имеют поддержку растущей экосистемы источников данных. Блокноты **Zeppelin** позволяют ученым в области данных в реальном времени создавать и выполнять небольшие фрагменты кода.

URL-адресом блокнота можно поделиться между сотрудниками. В таком случае **Apache Zeppelin**

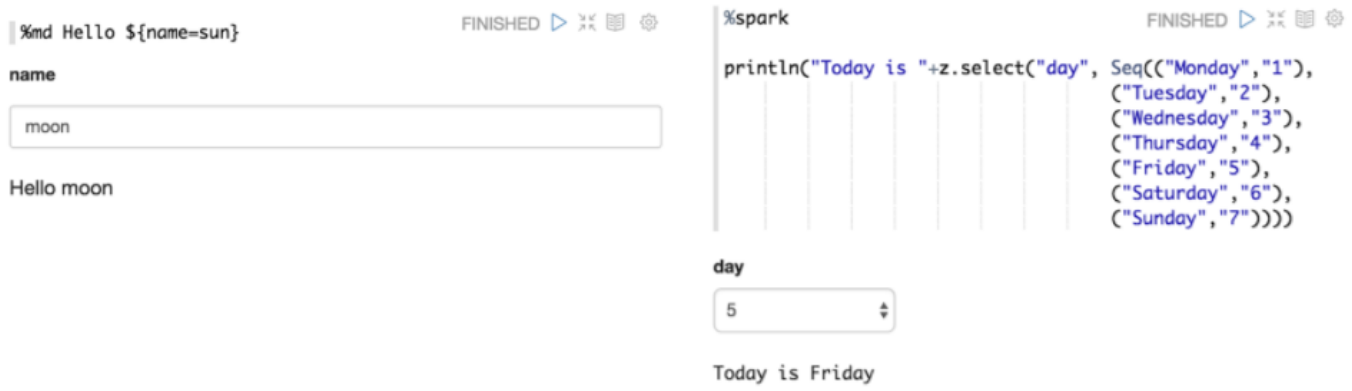



Рис.3.5.: Формы ввода в блокноте

транслирует любые изменения в реальном времени точно так же, как при работе в **Google docs**. Но данный URL-адрес отображает только результат, страница не содержит никаких меню и кнопок для редактирования. Кроме того, при завершении работы с блокнотом можно создать отчет и при необходимости распечатать его или экспортировать (Рис.3.6.).

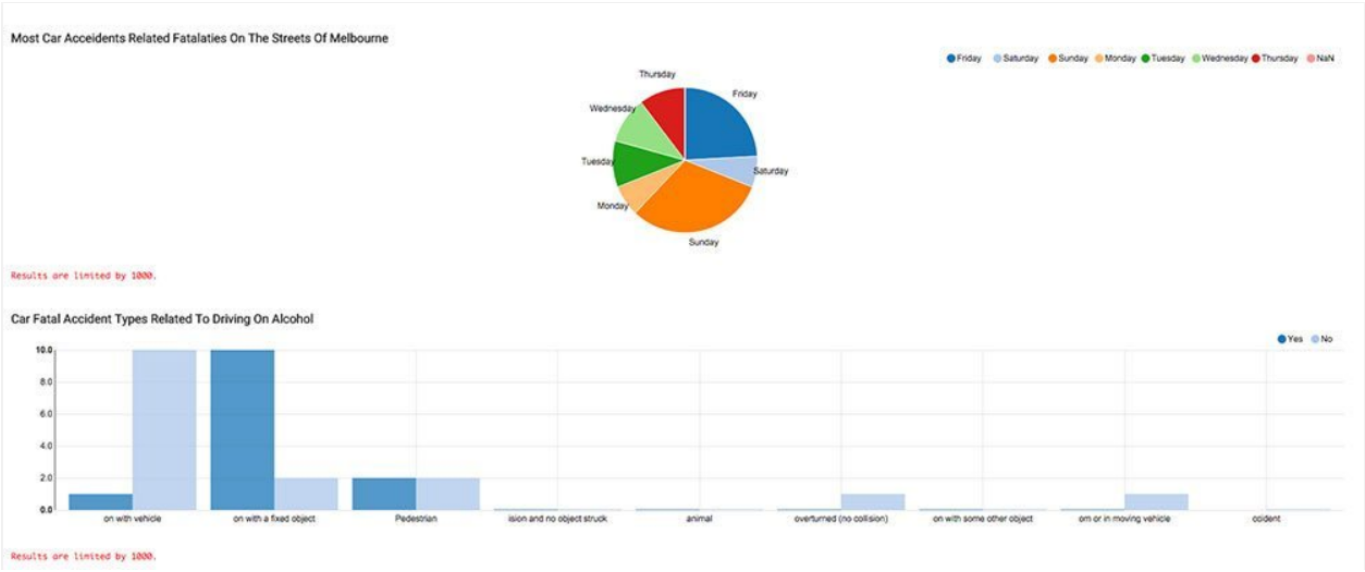



Рис.3.6.: Отчет о работе в Apache Zeppelin

В **Arenadata** мы считаем, что **Spark** и **Hadoop** идеально сочетаются. И что **Zeppelin** является ключевым компонентом для ускорения решений в области науки о данных.

## 3.2 Установка интерпретатора

**Apache Zeppelin** обеспечивает механизм установки интерпретатора, благодаря загруженному с **Zeppelin** бинарному пакету *netinst*, также можно установить другие сторонние интерпретаторы.

- *Установка интерпретаторов, разрабатываемых сообществом*
- *Сторонние интерпретаторы*
- *Доступные интерпретаторы, разрабатываемые сообществом*

**Important:** После установки интерпретаторов необходимо перезапустить Apache Zeppelin, выполнить настройку интерпретатора и привязать его к блокноту (Интерпретаторы в Apache Zeppelin)

### 3.2.1 Установка интерпретаторов, разрабатываемых сообществом

**Apache Zeppelin** позволяет управлять несколькими интерпретаторами одновременно, объединяя их в группы, перечень которых представлен в разделе *Доступные интерпретаторы, разрабатываемые сообществом*. Если бинарный пакет *netinst* уже загружен, следует в зависимости от необходимых интерпретаторов выполнить соответствующие команды.

- **Установка предоставляемых сообществом интерпретаторов**

Для установки всех интерпретаторов, которые предоставляются сообществом, необходимо выполнить следующую команду:

```
./bin/install-interpreter.sh --all
```

- **Установка выборочных интерпретаторов**

Для установки отдельно выбранных интерпретаторов необходимо воспользоваться следующей командой:

```
./bin/install-interpreter.sh --name md,shell,jdbc,python
```

Для получения полного списка интерпретаторов, разрабатываемых сообществом, следует выполнить команду:

```
./bin/install-interpreter.sh --list
```

- **Установка интерпретатора с версией языка Scala 2.10**

**Zeppelin** поддерживает **Scala 2.10** и **2.11** для нескольких интерпретаторов, параметры которых приведены в таблице.

Таблица3.1.: Параметры интерпретаторов для Scala

| Параметр –name | Параметр –artifact для Scala 2.10 | Параметр –artifact для Scala 2.11 |
|---|---|---|
| cassandra | org.apache.zeppelin:zeppelin-cassandra_2.10:0.7.3 | org.apache.zeppelin:zeppelin-cassandra_2.11:0.7.3 |
| flink | org.apache.zeppelin:zeppelin-flink_2.10:0.7.3 | org.apache.zeppelin:zeppelin-flink_2.11:0.7.3 |
| ignite | org.apache.zeppelin:zeppelin-ignite_2.10:0.7.3 | org.apache.zeppelin:zeppelin-ignite_2.11:0.7.3 |
| scio | org.apache.zeppelin:zeppelin-scio_2.10:0.7.3 | org.apache.zeppelin:zeppelin-scio_2.11:0.7.3 |
| spark | org.apache.zeppelin:zeppelin-spark_2.10:0.7.3 | org.apache.zeppelin:zeppelin-spark_2.11:0.7.3 |

При установке интерпретатора только с параметром *–name*, программа установки загружает по умолчанию интерпретатор с поддержкой версии языка **Scala 2.11**. Для указания иной версии **Scala** следует добавить параметр *–artifact*. Далее приведен пример установки интерпретатора *flink* с версией языка **Scala 2.10**:

```
./bin/install-interpreter.sh --name flink --artifact org.apache.zeppelin:zeppelin-flink_2.10:0.7.3
```

- **Установка интерпретатора Spark, поддерживающего версию языка Scala 2.10**

Дистрибутив **Spark** до версии *1.6.2* поддерживает **Scala 2.10**. Если *SPARK_HOME* указывает на версию **Spark** ниже *2.0.0*, необходимо скачать интерпретатор **Spark** с версией языка **Scala 2.10**. Для этого следует выполнить команду:

```
rm -rf ./interpreter/spark
./bin/install-interpreter.sh --name spark --artifact org.apache.zeppelin:zeppelin-spark_2.10:0.7.3
```

### 3.2.2 Сторонние интерпретаторы

Сторонние интерпретаторы из репозитория **maven** можно установить при помощи следующей команды:

```
./bin/install-interpreter.sh --name interpreter1 --artifact groupId1:artifact1:version1
```

Данная команда загружает артефакт **maven** *groupId1:artifact1:version1* и все его зависимости в каталог *interpreter/interpreter1*.

Установка нескольких сторонних интерпретаторов осуществляется командой, где аргументы *–name* и *–artifact* указываются списком через запятую:

```
./bin/install-interpreter.sh --name interpreter1,interpreter2 --artifact␣
↪groupId1:artifact1:version1,groupId2:artifact2:version2
```

### 3.2.3 Доступные интерпретаторы, разрабатываемые сообществом

Список интерпретаторов, предоставляемых сообществом, приведен в таблице. Также данную информацию можно найти в файле *conf/interpreter-list*.

Таблица3.2.: Предоставляемые сообществом интерпретаторы

| Параметр –name | Maven Artifact | Описание |
|---|---|---|
| alluxio | org.apache.zeppelin:zeppelin-alluxio:0.7.3 | Интерпретатор Alluxio |
| angular | org.apache.zeppelin:zeppelin-angular:0.7.3 | Просмотр HTML и AngularJS |
| beam | org.apache.zeppelin:zeppelin-beam:0.7.3 | Интерпретатор Beam |
| bigquery | org.apache.zeppelin:zeppelin-bigquery:0.7.3 | Интерпретатор BigQuery |
| cassandra | org.apache.zeppelin:zeppelin-cassandra_2.11:0.7.3 | Интерпретатор Cassandra, построенный с помощью Scala 2.11 |
| elasticsearch | org.apache.zeppelin:zeppelin-elasticsearch:0.7.3 | Интерпретатор Elasticsearch |
| file | org.apache.zeppelin:zeppelin-file:0.7.3 | Интерпретатор файлов HDFS |
| flink | org.apache.zeppelin:zeppelin-flink_2.11:0.7.3 | Интерпретатор Flink, построенный с помощью Scala 2.11 |
| hbase | org.apache.zeppelin:zeppelin-hbase:0.7.3 | Интерпретатор Hbase |
| ignite | org.apache.zeppelin:zeppelin-ignite_2.11:0.7.3 | Интерпретатор Ignite, построенный с помощью Scala 2.11 |
| jdbc | org.apache.zeppelin:zeppelin-jdbc:0.7.3 | Интерпретатор Jdbc |
| kylin | org.apache.zeppelin:zeppelin-kylin:0.7.3 | Интерпретатор Kylin |
| lens | org.apache.zeppelin:zeppelin-lens:0.7.3 | Интерпретатор Lens |
| livy | org.apache.zeppelin:zeppelin-livy:0.7.3 | Интерпретатор Livy |
| md | org.apache.zeppelin:zeppelin-markdown:0.7.3 | Поддержка Markdown |
| pig | org.apache.zeppelin:zeppelin-pig:0.7.3 | Интерпретатор Pig |
| postgresql | org.apache.zeppelin:zeppelin-postgresql:0.7.3 | Интерпретатор Postgresql |
| python | org.apache.zeppelin:zeppelin-python:0.7.3 | Интерпретатор Python |
| scio | org.apache.zeppelin:zeppelin-scio_2.11:0.7.3 | Интерпретатор Scio, построенный с помощью Scala 2.11 |
| shell | org.apache.zeppelin:zeppelin-shell:0.7.3 | Команда Shell |

## 3.3 Конфигурация Apache Zeppelin

- *Свойства Apache Zeppelin*
- *Конфигурация SSL*
  - *Создание и настройка сертификатов*

- *Настройка SSL на стороне сервера*
- *Включение проверки подлинности сертификата на стороне клиента*
- *Скрытие паролей с помощью Jetty Password Tool*

### 3.3.1 Свойства Apache Zeppelin

Существует две локации, в которых можно настроить **Apache Zeppelin**:

- Переменные окружения могут быть заданы в *conf/zeppelin-env.sh* (*confzeppelin-env.cmd* для ОС Windows);
- Свойства Java могут быть определены в *conf/zeppelin-site.xml*.

В случае если настроены обе локации, переменные окружения будут приоритетными.

Таблица3.3.: Параметры Zeppelin, их значения и описание

| Параметр Zeppelin | Значение и описание |
|---|---|
| • zeppelin-env.sh: ZEPPELIN_PORT<br>• zeppelin-site.xml: zeppelin.server.port | *8080*<br>Порт сервера Zeppelin. Примечание: необходимо убедиться, что не используется тот же порт, что для разработки веб-приложений Zeppelin (по умолчанию: 9000) |
| • zeppelin-env.sh: ZEPPELIN_SSL_PORT<br>• zeppelin-site.xml: zeppelin.server.ssl.port | *8443*<br>ssl порт сервера Zeppelin (используется, когда значение ssl свойства установлено true) |
| • zeppelin-env.sh: ZEPPELIN_MEM<br>• zeppelin-site.xml: N/A | *-Xmx1024m -XX:MaxPermSize=512m*<br>Параметры JVM mem |
| • zeppelin-env.sh: ZEPPELIN_INTP_MEM<br>• zeppelin-site.xml: N/A | *ZEPPELIN_MEM*<br>Параметры JVM mem для процесса интерпретатора |
| • zeppelin-env.sh: ZEPPELIN_JAVA_OPTS<br>• zeppelin-site.xml: N/A | Параметры JVM |
| • zeppelin-env.sh: ZEPPELIN_ALLOWED_ORIGINS<br>• zeppelin-site.xml: zeppelin.server.allowed.origins | *символ “*”*<br>Позволяет разделяя знаком запятой перечислить разрешенные источники для REST и websockets подключений. Например, http://localhost:8080 |
| • zeppelin-env.sh: N/A<br>• zeppelin-site.xml: zeppelin.anonymous.allowed | *true*<br>Вход для неавторизованного пользователя разрешен по умолчанию |

| Параметр Zeppelin | Значение и описание |
|---|---|
| • zeppelin-env.sh: ZEPPELIN_SERVER_CONTEXT_PATH<br>• zeppelin-site.xml: zeppelin.server.context.path | *символ "/"*<br>Относительный путь веб-приложения |
| • zeppelin-env.sh: ZEPPELIN_SSL<br>• zeppelin-site.xml: zeppelin.ssl | *false* |
| • zeppelin-env.sh: ZEPPELIN_SSL_CLIENT_AUTH<br>• zeppelin-site.xml: zeppelin.ssl.client.auth | *false* |
| • zeppelin-env.sh: ZEPPELIN_SSL_KEYSTORE_PATH<br>• zeppelin-site.xml: zeppelin.ssl.keystore.path | *keystore* |
| • zeppelin-env.sh: ZEPPELIN_SSL_KEYSTORE_TYPE<br>• zeppelin-site.xml: zeppelin.ssl.keystore.type | *JKS* |
| • zeppelin-env.sh: ZEPPELIN_SSL_KEYSTORE_PASSWORD<br>• zeppelin-site.xml: zeppelin.ssl.keystore.password | |
| • zeppelin-env.sh: ZEPPELIN_SSL_KEY_MANAGER _PASSWORD<br>• zeppelin-site.xml: zeppelin.ssl.key.manager.password | |
| • zeppelin-env.sh: ZEPPELIN_SSL_TRUSTSTORE_PATH<br>• zeppelin-site.xml: zeppelin.ssl.truststore.path | |
| • zeppelin-env.sh: ZEPPELIN_SSL_TRUSTSTORE_TYPE<br>• zeppelin-site.xml: zeppelin.ssl.truststore.type | |
| • zeppelin-env.sh: ZEPPELIN_SSL_TRUSTSTORE _PASSWORD<br>• zeppelin-site.xml: zeppelin.ssl.truststore.password | |

| Параметр Zeppelin | Значение и описание |
|---|---|
| • zeppelin-env.sh: ZEPPELIN_NOTEBOOK_HOMESCREEN<br>• zeppelin-site.xml: zeppelin.notebook.homescreen | Отображение идентификаторов блокнотов на рабочем столе Apache Zeppelin. Например, 2A94M5J1Z |
| • zeppelin-env.sh: ZEPPELIN_NOTEBOOK_HOMESCREEN _HIDE<br>• zeppelin-site.xml: zeppelin.notebook.homescreen.hide | *false*<br>Скрытие идентификатора блокнота, установленного в ZEPPELIN_NOTEBOOK_HOMESCREEN на рабочем столе Apache Zeppelin |
| • zeppelin-env.sh: ZEPPELIN_WAR_TEMPDIR<br>• zeppelin-site.xml: zeppelin.war.tempdir | *webapps*<br>Расположение временного каталога jetty |
| • zeppelin-env.sh: ZEPPELIN_NOTEBOOK_DIR<br>• zeppelin-site.xml: zeppelin.notebook.dir | *notebook*<br>Каталог root, в котором хранятся каталоги блокнотов |
| • zeppelin-env.sh: ZEPPELIN_NOTEBOOK_STORAGE<br>• zeppelin-site.xml: zeppelin.notebook.storage | *org.apache.zeppelin.notebook.repo. GitNotebookRepo*<br>Список мест хранения блокнотов, перечисленный через запятую |
| • zeppelin-env.sh: ZEPPELIN_NOTEBOOK_ONE_WAY_SYNC<br>• zeppelin-site.xml: zeppelin.notebook.one.way.sync | *false*<br>Если есть несколько мест для хранения блокнотов, следует ли рассматривать первое как единственное? |
| • zeppelin-env.sh: ZEPPELIN_NOTEBOOK_PUBLIC<br>• zeppelin-site.xml: zeppelin.notebook.public | *true*<br>Сделать блокнот общедоступным по умолчанию при создании или импорте (при указании только владельцев). Если установлено значение false, необходимо добавить пользователей, для которых доступно чтение и редактирование, таким образом делая блокнот недоступным и невидимым для пользователей, не имеющих прав на него |
| • zeppelin-env.sh: ZEPPELIN_INTERPRETERS<br>• zeppelin-site.xml: zeppelin.interpreters | *org.apache.zeppelin.spark.SparkInterpreter, org.apache.zeppelin.spark.PySparkInterpreter, org.apache.zeppelin.spark.SparkSqlInterpreter, org.apache.zeppelin.spark.DepInterpreter, org.apache.zeppelin.markdown.Markdown, org.apache.zeppelin.shell.ShellInterpreter, . . .*<br>Конфигурации (классы) интерпретатора с разделителями-запятыми. Примечание: это свойство устарело с Zeppelin-0.6.0 и не будет поддерживаться Zeppelin-0.7.0 |

| Параметр Zeppelin | Значение и описание |
|---|---|
| • zeppelin-env.sh: ZEPPELIN_INTERPRETER_DIR<br>• zeppelin-site.xml: zeppelin.interpreter.dir | *interpreter*<br>Каталог интерпретатора |
| • zeppelin-env.sh: ZEPPELIN_INTERPRETER_DEP _MVNREPO<br>• zeppelin-site.xml: zeppelin.interpreter.dep.mvnRepo | *http://repo1.maven.org/maven2/*<br>Удаленный основной репозиторий для загрузки дополнительных зависимостей интерпретатора |
| • zeppelin-env.sh: ZEPPELIN_DEP_LOCALREPO<br>• zeppelin-site.xml: zeppelin.dep.localrepo | *local-repo*<br>Локальный репозиторий для загрузки зависимостей. Модули npm |
| • zeppelin-env.sh: ZEPPELIN_HELIUM_NPM_REGISTRY<br>• zeppelin-site.xml: zeppelin.helium.npm.registry | *http://registry.npmjs.org/*<br>Удаленный реестр Npm для загрузки зависимостей Helium |
| • zeppelin-env.sh: ZEPPELIN_INTERPRETER_OUTPUT _LIMIT<br>• zeppelin-site.xml: zeppelin.interpreter.output.limit | *102400*<br>Размер выходного сообщения от интерпретатора. Если сообщение превышает заданный размер, то она урезается до заданного значения. |
| • zeppelin-env.sh: ZEPPELIN_WEBSOCKET_MAX_TEXT _MESSAGE_SIZE<br>• zeppelin-site.xml: zeppelin.websocket.max.text.message.size | *1024000*<br>Размер (в символах) максимального текстового сообщения, которое может быть получено от websocket |
| • zeppelin-env.sh: ZEPPELIN_SERVER_DEFAULT_DIR _ALLOWED<br>• zeppelin-site.xml: zeppelin.server.default.dir.allowed | *false*<br>Доступность каталогов на сервере |

### 3.3.2 Конфигурация SSL

Включение **SSL** требует некоторых изменений конфигурации – следует создать сертификаты, а затем обновить необходимые настройки для подключения проверки подлинности **SSL** со стороны сервера и/или клиентской стороны.

#### Создание и настройка сертификатов

Информацию о создании сертификатов и хранилище ключей можно найти по ссылке. Краткий пример можно найти в верхнем ответе на вопрос StackOverflow.

Хранилище ключей **keystore** содержит закрытый ключ и сертификат на сервере, а хранилище **trustore** содержит сертификаты доверенных клиентов. Необходимо убедиться, что путь и пароль для этих двух хранилищ правильно настроены в полях пароля, приведенных ниже. Они могут быть скрыты с помощью **Jetty Password Tool**. **Maven** подтягивает все зависимости для сборки **Zeppelin**, один из jar-файлов **Jetty** содержит инструмент

**Jetty Password Tool**. Необходимо вызвать команду из каталога сборки *Zeppelin Home* с соответствующей версией, пользователем и паролем:

```
java -cp ./zeppelin-server/target/lib/jetty-all-server-<version>.jar org.eclipse.jetty.util.
↪security.Password <user> <password>
```

Если используется самоподписанный сертификат, сертификат, подписанный недоверенным центром сертификации, или если включена аутентификация клиента, то у клиента в браузере должно появиться исключение как в случае https-порта, так и websocket-порта. Это можно проверить, установив HTTPS соединение по обоими портами в браузере (например, если порты *443* и *8443*, при переходе на *https://127.0.0.1:443* и *https://127.0.0.1:8443*). Данный шаг может быть пропущен, если сертификат сервера подписан доверенным центром сертификации и аутентификация клиента отключена.

### Настройка SSL на стороне сервера

Для включения **SSL** на стороне сервера необходимо отредактировать в *zeppelin-site.xml* следующие свойства:

```
<property>
  <name>zeppelin.server.ssl.port</name>
  <value>8443</value>
  <description>Server ssl port. (used when ssl property is set to true)</description>
</property>

<property>
  <name>zeppelin.ssl</name>
  <value>true</value>
  <description>Should SSL be used by the servers?</description>
</property>

<property>
  <name>zeppelin.ssl.keystore.path</name>
  <value>keystore</value>
  <description>Path to keystore relative to Zeppelin configuration directory</description>
</property>

<property>
  <name>zeppelin.ssl.keystore.type</name>
  <value>JKS</value>
  <description>The format of the given keystore (e.g. JKS or PKCS12)</description>
</property>

<property>
  <name>zeppelin.ssl.keystore.password</name>
  <value>change me</value>
  <description>Keystore password. Can be obfuscated by the Jetty Password tool</description>
</property>

<property>
  <name>zeppelin.ssl.key.manager.password</name>
  <value>change me</value>
  <description>Key Manager password. Defaults to keystore password. Can be obfuscated.</
↪description>
</property>
```

### Включение проверки подлинности сертификата на стороне клиента

Для включения аутентификации по сертификату на стороне клиента необходимо отредактировать в *zeppelin-site.xml* следующие свойства:

```
<property>
  <name>zeppelin.server.ssl.port</name>
  <value>8443</value>
  <description>Server ssl port. (used when ssl property is set to true)</description>
</property>

<property>
  <name>zeppelin.ssl.client.auth</name>
  <value>true</value>
  <description>Should client authentication be used for SSL connections?</description>
</property>

<property>
  <name>zeppelin.ssl.truststore.path</name>
  <value>truststore</value>
  <description>Path to truststore relative to Zeppelin configuration directory. Defaults to the␣
↪keystore path</description>
</property>

<property>
  <name>zeppelin.ssl.truststore.type</name>
  <value>JKS</value>
  <description>The format of the given truststore (e.g. JKS or PKCS12). Defaults to the same type␣
↪as the keystore type</description>
</property>

<property>
  <name>zeppelin.ssl.truststore.password</name>
  <value>change me</value>
  <description>Truststore password. Can be obfuscated by the Jetty Password tool. Defaults to the␣
↪keystore password</description>
</property>
```

### Скрытие паролей с помощью Jetty Password Tool

Лучшие практики по безопасности рекомендуют не использовать текстовые пароли, а с помощью утилиты **Jetty Password Tool** (см. документацию) можно усложнить пароли, используемые для доступа к **keystore** и **trustore**.

После установки **Jetty Password Tool**:

```
java -cp $ZEPPELIN_HOME/zeppelin-server/target/lib/jetty-util-9.2.15.v20160210.jar \
         org.eclipse.jetty.util.security.Password  \
         password

2016-12-15 10:46:47.931:INFO::main: Logging initialized @101ms
password
OBF:1v2j1uum1xtv1zej1zer1xtn1uvk1v1v
MD5:5f4dcc3b5aa765d61d8327deb882cf99
```

Затем необходимо обновить конфигурацию с паролем:

```
<property>
  <name>zeppelin.ssl.keystore.password</name>
```

```
  <value>OBF:1v2j1uum1xtv1zej1zer1xtn1uvk1v1v</value>
  <description>Keystore password. Can be obfuscated by the Jetty Password tool</description>
</property>
```

**Important:** После обновления настроек сервер Zeppelin необходимо перезапустить

## 3.4 Аутентификация Apache Shiro для Apache Zeppelin


- *Обзор*
- *Настройка безопасности*
- *Группы и разрешения (опционально)*
- *Настройка Realm (опционально)*
  - *Active Directory*
  - *LDAP*
  - *PAM*
  - ZeppelinHub
- *Безопасность Cookie в сессиях Zeppelin (опционально)*
- *Защита информации Zeppelin (опционально)*
- *Альтернативные методы аутентификации*


### 3.4.1 Обзор

Apache Shiro – это мощный и простой в использовании **Java** фреймворк безопасности, позволяющий выполнить аутентификацию, авторизацию, криптографию и управление сессиями. В данном разделе объясняется, как **Shiro** работает для аутентификации на сервер **Zeppelin**.

При подключении к **Apache Zeppelin** необходимо ввести учетные данные. После входа в систему появляется доступ ко всем блокнотам, включая блокноты других пользователей.

### 3.4.2 Настройка безопасности

Настройка аутентификации на сервер **Zeppelin** осуществляется несколькими простыми шагами:

**1. Права Shiro**

По умолчанию в *conf* находится файл *shiro.ini.template*. Данный файл используется в качестве примера, и настоятельно рекомендуется создать файл *shiro.ini*, выполнив следующую команду:

```
cp conf/shiro.ini.template conf/shiro.ini
```

Дополнительная информация о формате файла *shiro.ini* находится по ссылке Shiro Configuration.

**2. Безопасность канала Websocket**

Для свойства *zeppelin.anonymous.allowed* необходимо установить значение *false* в *conf/zeppelin-site.xml*. В случае если данного файла нет, следует просто скопировать файл *conf/zeppelin-site.xml.template* в *conf/zeppelin-site.xml*.

**3. Запуск Zeppelin**

Для запуска **Zeppelin** необходимо выполнить команду:

`bin/zeppelin-daemon.sh start (or restart)`

После чего можно обратиться к **Zeppelin** по адресу *http://localhost:8080*.

**4. Авторизация**

Теперь можно войти в систему, используя комбинацию имени и пароля пользователя (Рис.3.7.).

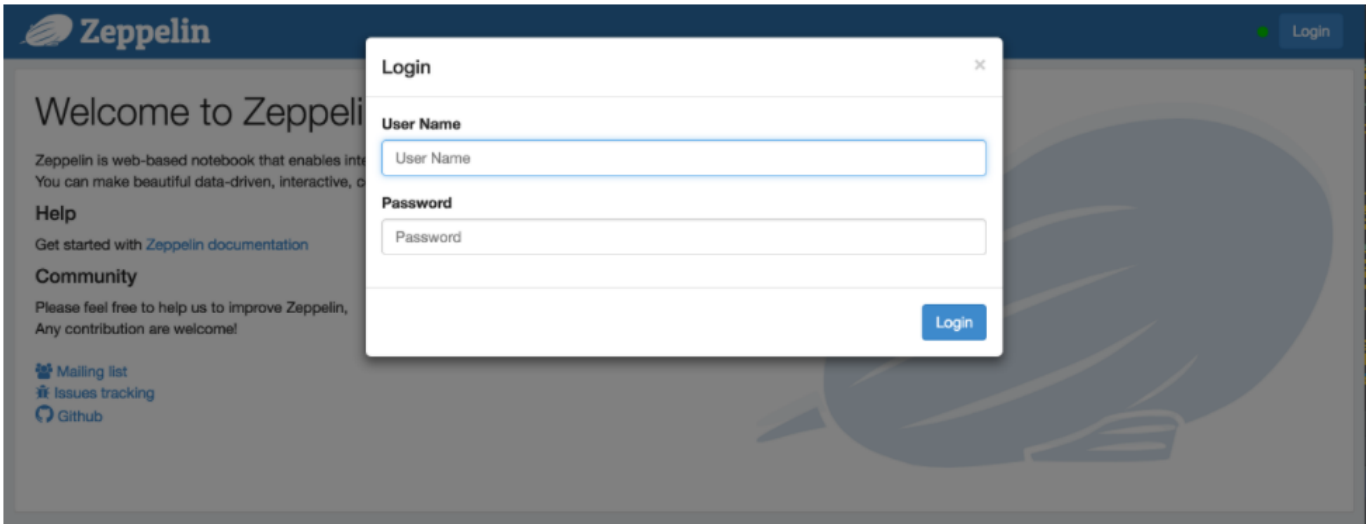



Рис.3.7.: Авторизация в Apache Zeppelin

После пароля через запятую можно указать роли для каждого пользователя:

```
[users]

admin = password1, admin
user1 = password2, role1, role2
user2 = password3, role3
user3 = password4, role2
```

### 3.4.3 Группы и разрешения (опционально)

Для использования групп пользователей и прав для них необходимо применить одну из нижеприведенных конфигураций для **LDAP** или **AD** в разделе *[main]* в файле *shiro.ini*:

```
activeDirectoryRealm = org.apache.zeppelin.realm.ActiveDirectoryGroupRealm
activeDirectoryRealm.systemUsername = userNameA
activeDirectoryRealm.systemPassword = passwordA
activeDirectoryRealm.searchBase = CN=Users,DC=SOME_GROUP,DC=COMPANY,DC=COM
activeDirectoryRealm.url = ldap://ldap.test.com:389
activeDirectoryRealm.groupRolesMap = "CN=aGroupName,OU=groups,DC=SOME_GROUP,DC=COMPANY,DC=COM":
↪"group1"
activeDirectoryRealm.authorizationCachingEnabled = false
activeDirectoryRealm.principalSuffix = @corp.company.net

ldapRealm = org.apache.zeppelin.server.LdapGroupRealm
# search base for ldap groups (only relevant for LdapGroupRealm):
ldapRealm.contextFactory.environment[ldap.searchBase] = dc=COMPANY,dc=COM
ldapRealm.contextFactory.url = ldap://ldap.test.com:389
ldapRealm.userDnTemplate = uid={0},ou=Users,dc=COMPANY,dc=COM
ldapRealm.contextFactory.authenticationMechanism = simple
```

И определить роли/группы, которые необходимо иметь в системе, например:

```
[roles]
admin = *
hr = *
finance = *
group1 = *
```

### 3.4.4 Настройка Realm (опционально)

**Realms** отвечают за аутентификацию и авторизацию в **Apache Zeppelin**. По умолчанию **Apache Zeppelin** использует IniRealm (пользователи и группы настраиваются в файле *conf/shiro.ini* в разделах *[user]* и *[group]*). Также можно использовать **Shiro Realms**, такие как JndiLdapRealm, JdbcRealm или создать собственный. Подробная документация о **Apache Shiro Realm** представлена по ссылке.

#### Active Directory

```
activeDirectoryRealm = org.apache.zeppelin.realm.ActiveDirectoryGroupRealm
activeDirectoryRealm.systemUsername = userNameA
activeDirectoryRealm.systemPassword = passwordA
activeDirectoryRealm.hadoopSecurityCredentialPath = jceks://file/user/zeppelin/conf/zeppelin.jceks
activeDirectoryRealm.searchBase = CN=Users,DC=SOME_GROUP,DC=COMPANY,DC=COM
activeDirectoryRealm.url = ldap://ldap.test.com:389
activeDirectoryRealm.groupRolesMap = "CN=aGroupName,OU=groups,DC=SOME_GROUP,DC=COMPANY,DC=COM":
↪"group1"
activeDirectoryRealm.authorizationCachingEnabled = false
activeDirectoryRealm.principalSuffix = @corp.company.net
```

Кроме того, вместо указания *systemPassword* в виде текста в *shiro.ini* администратор может указать то же самое, что и в *hadoop credential*. Необходимо создать keystore-файл, используя командную строку *hadoop credential*, для этого *hadoop* должен быть прописан в *classpath*:

```
hadoop credential create activeDirectoryRealm.systempassword -provider jceks://file/user/zeppelin/
↪conf/zeppelin.jceks
```

Далее следует изменить следующие значения в файле *Shiro.ini* и раскомментировать строку:

```
activeDirectoryRealm.hadoopSecurityCredentialPath = jceks://file/user/zeppelin/conf/zeppelin.jceks
```

#### LDAP

Для настройки **LDAP Realm** существует два способа. Проще использовать **LdapGroupRealm**. Однако, он менее гибкий при настройке соответствий между группами **LDAP** и пользователями, а также для авторизации групп пользователей. Далее приведен пример файла с соответствующими настройками:

```
ldapRealm = org.apache.zeppelin.realm.LdapGroupRealm
# search base for ldap groups (only relevant for LdapGroupRealm):
ldapRealm.contextFactory.environment[ldap.searchBase] = dc=COMPANY,dc=COM
ldapRealm.contextFactory.url = ldap://ldap.test.com:389
ldapRealm.userDnTemplate = uid={0},ou=Users,dc=COMPANY,dc=COM
ldapRealm.contextFactory.authenticationMechanism = simple
```

Другим более гибким способом является использование **LdapRealm**. Он позволяет сопоставлять *ldapgroups* с ролями, а также допускает проверку подлинности на основе ролей/групп на сервере *zeppelin*. Пример конфигурации приведен ниже:

```
ldapRealm=org.apache.zeppelin.realm.LdapRealm

ldapRealm.contextFactory.authenticationMechanism=simple ldapRealm.contextFactory.url=ldap://
↪localhost:33389  ldapRealm.userDnTemplate=uid={0},ou=people,dc=hadoop,dc=apache,dc=org
```

**Возможность задать параметр ldap paging. Размер по умолчанию - 100**

```
ldapRealm.pagingSize = 200 ldapRealm.authorizationEnabled=true ldapRealm.contextFactory.
↪systemAuthenticationMechanism=simple ldapRealm.searchBase=dc=hadoop,dc=apache,dc=org ldapRealm.
↪userSearchBase = dc=hadoop,dc=apache,dc=org ldapRealm.groupSearchBase = ou=groups,dc=hadoop,
↪dc=apache,dc=org ldapRealm.groupObjectClass=groupofnames
```

**Возможность настройки параметра userSearchAttribute**

```
ldapRealm.userSearchAttributeName = sAMAccountName ldapRealm.memberAttribute=member
```

**Возврат имен пользователей из ldap в нижнем регистре для использования в AD**

```
ldapRealm.userLowerCase = true
```

**Возможность установить параметр searchScopes в одно из трех значений: subtree (по умолчанию), one или base**

```
ldapRealm.userSearchScope = subtree; ldapRealm.groupSearchScope = subtree; ldapRealm.
↪memberAttributeValueTemplate=cn={0},ou=people,dc=hadoop,dc=apache,dc=org ldapRealm.
↪contextFactory.systemUsername=uid=guest,ou=people,dc=hadoop,dc=apache,dc=org ldapRealm.
↪contextFactory.systemPassword=S{ALIAS=ldcSystemPassword}
```

**Включение поддержки вложенных групп при помощи оператора LDAPMATCHINGRULEINCHAIN**

```
ldapRealm.groupSearchEnableMatchingRuleInChain = true
```

**Дополнительная настройка соответствий между физическими группами и логическими ролями приложений**

```
ldapRealm.rolesByGroup = LDNUSERS: userrole, NYKUSERS: userrole, HKGUSERS: userrole, GLOBALADMIN:␣
↪adminrole
```

**Дополнительный список ролей, которым разрешена аутентификация. В случае если список не представлен, всем ролям разрешается аутентификация (вход)**

**Данные изменения не влияют на специфические права url. Для url будут работать те права, которые указаны в разделе [urls]**

```
ldapRealm.allowedRolesForAuthentication = adminrole,userrole ldapRealm.permissionsByRole=␣
↪userrole = :ToDoItemsJdo::, *:ToDoItem::*; adminrole = * securityManager.sessionManager =
↪$sessionManager securityManager.realms = $ldapRealm ```
```

### PAM

Поддержка аутентификации с помощью PAM позволяет повторно использовать существующие модули аутентификации в узле, где запущен **Zeppelin**. В типичных системных модулях, например, *sshd*, *passwd* и других сервис настраивается в */etc/pam.d/*. Можно повторно использовать один из этих сервисов или создать свой собственный для **Zeppelin**. Для активации аутентификации **PAM** требуется два параметра: 1 – realm: использование **Shiro realm**; 2 – service: настроенный в */etc/pam.d/* сервис. Название должно совпадать с именем файла в */etc/pam.d/*.

```
[main]
 pamRealm=org.apache.zeppelin.realm.PamRealm
 pamRealm.service=sshd
```

#### ZeppelinHub

ZeppelinHub – это сервис, синхронизурующий блокноты **Apache Zeppelin** и обеспечивающий легкое взаимодействие с ними. Для подключения **ZeppelinHub** необходимо применить следующее изменение в *conf/shiro.ini* в разделе *[main]*:

```
### A sample for configuring ZeppelinHub Realm
zeppelinHubRealm = org.apache.zeppelin.realm.ZeppelinHubRealm
## Url of ZeppelinHub
zeppelinHubRealm.zeppelinhubUrl = https://www.zeppelinhub.com
securityManager.realms = $zeppelinHubRealm
```

**Important:** ZeppelinHub не относится к проекту Apache Zeppelin

### 3.4.5 Безопасность Cookie в сессиях Zeppelin (опционально)

**Zeppelin** может быть настроен выставлением флага **HttpOnly** в настройка **cookie** для сессии. С такой конфигурацией cookie-файлы **Zeppelin** не могут быть доступны через скрипты на стороне клиента, тем самым предотвращая большинство атак типа **Cross-Site scripting** (**XSS**).

Чтобы включить безопасную поддержку файлов **cookie** через **Shiro**, необходимо добавить следующие строки в *conf/shiro.ini* в раздел *[main]*, а затем задать *sessionManager*:

```
cookie = org.apache.shiro.web.servlet.SimpleCookie
cookie.name = JSESSIONID
cookie.secure = true
cookie.httpOnly = true
sessionManager.sessionIdCookie = $cookie
```

### 3.4.6 Защита информации Zeppelin (опционально)

По умолчанию любой пользователь, определенный в *[users]*, может видеть информацию об интерпретаторах, учетных данных и настройках в **Apache Zeppelin**. В случае если данную информацию необходимо скрыть, поскольку **Shiro** обеспечивает защиту на уровне url, следует закомментировать или раскомментировать приведенные ниже строки в *conf/shiro.ini*:

```
[urls]

/api/interpreter/** = authc, roles[admin]
/api/configurations/** = authc, roles[admin]
/api/credential/** = authc, roles[admin]
```

В таком случае информацию об интерпретаторах, учетных данных и настройках в **Apache Zeppelin** могут видеть только пользователи с ролью *admin*. При необходимости предоставления прав другим пользователям следует изменить роли в разделе *[users]*.

### 3.4.7 Альтернативные методы аутентификации

HTTP аутентификация с помощью NGINX

## 3.5 Интерфейс Apache Zeppelin

- *Главная страница*
- *Меню*
- *Набор инструментов*

### 3.5.1 Главная страница

При первом обращении к серверу **Zeppelin** открывается главная страница, представленная на Рис.3.8.

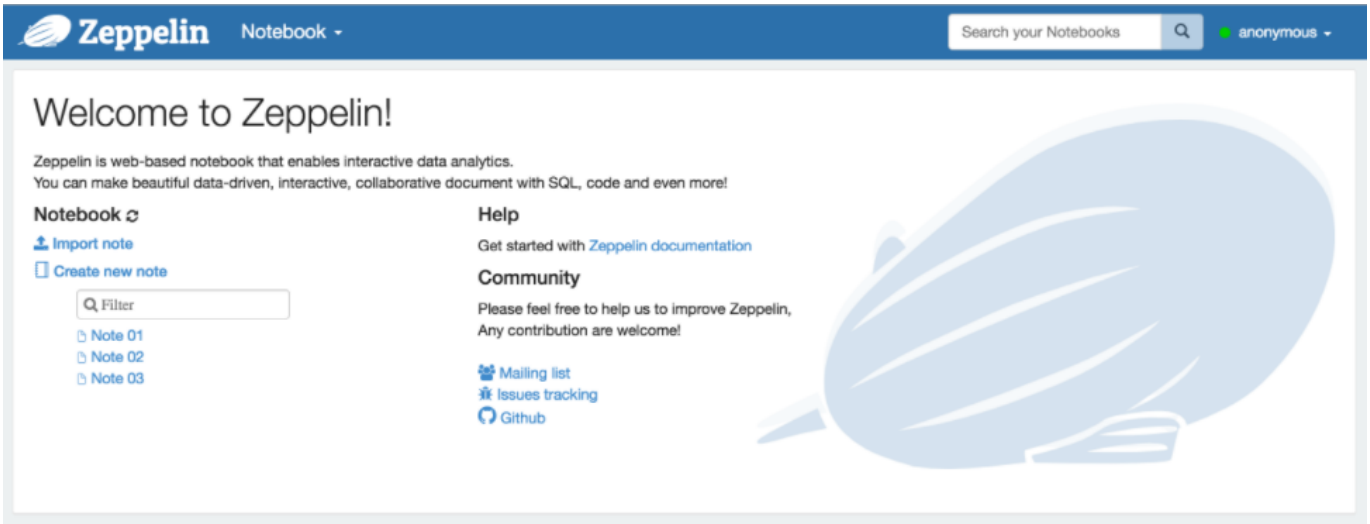



Рис.3.8.: Главная страница Apache Zeppelin

В левой части страницы перечислены все существующие блокноты. По умолчанию блокноты сохраняются в папке *$ZEPPELIN_HOME/notebook*.

Есть возможность фильтровать блокноты по имени, используя форму ввода текста. Также можно создавать новые, обновлять список существующих блокнотов (в случае если они вручную скопированы в папку *$ZEPPELIN_HOME/notebook*) и импортировать их.

При нажатии кнопки “Import Note” открывается новое диалоговое окно (Рис.3.9.). Блокнот можно импортировать с локального диска или из удаленного места по указанному URL-адресу. По умолчанию название импортируемого блокнота остается оригинальным, но при желании его можно изменить.

### 3.5.2 Меню

Панель меню “Notebook” предлагает почти те же функции, что и раздел управления блокнотами на главной странице (Рис.3.8.). В раскрывающемся контекстном меню можно выбрать (Рис.3.10.):

- Открыть выбранный блокнот;
- Фильтр блокнотов по имени;
- Создать новый блокнот.

Меню настроек дает доступ к конфигурации и отображает информацию о **Zeppelin** (Рис.3.11.). При использовании настроек по умолчанию имя пользователя задается как *anonymous*. Настройка аутентификации описана в разделе Аутентификация Apache Shiro для Apache Zeppelin.

По ссылке “About Zeppelin” (см. Рис.3.11.) можно получить информацию об установленной версии **Apache Zeppelin** (Рис.3.12.).

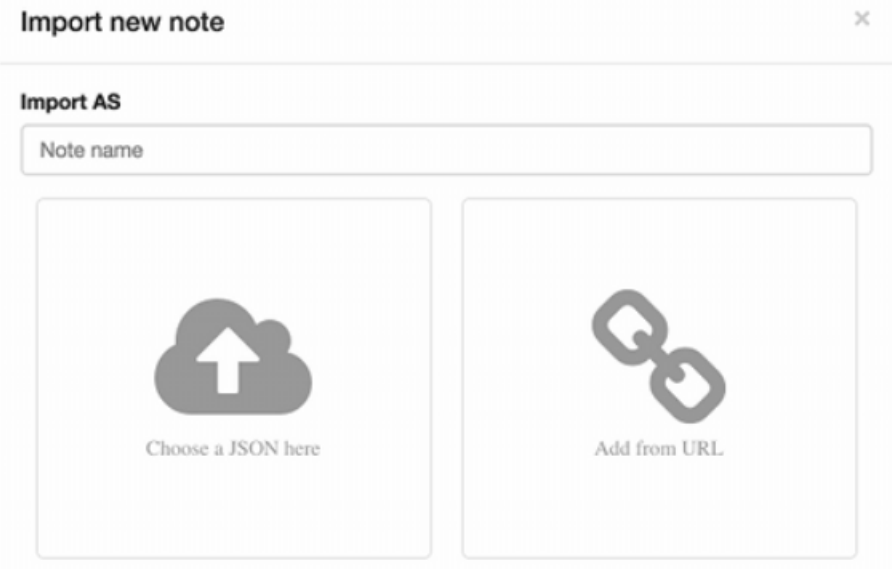



Рис.3.9.: Импорт новой заметки

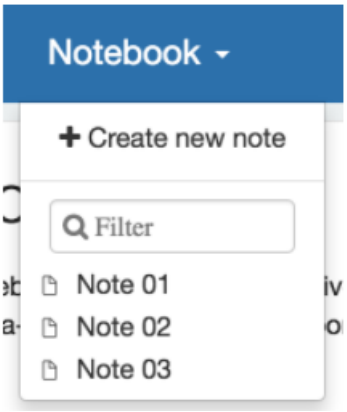



Рис.3.10.: Панель меню Notebook

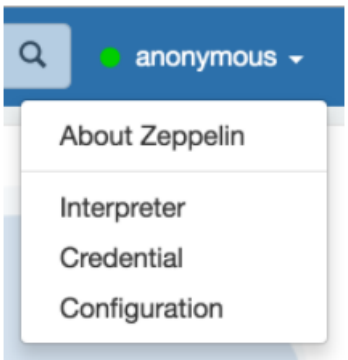



Рис.3.11.: Меню настроек

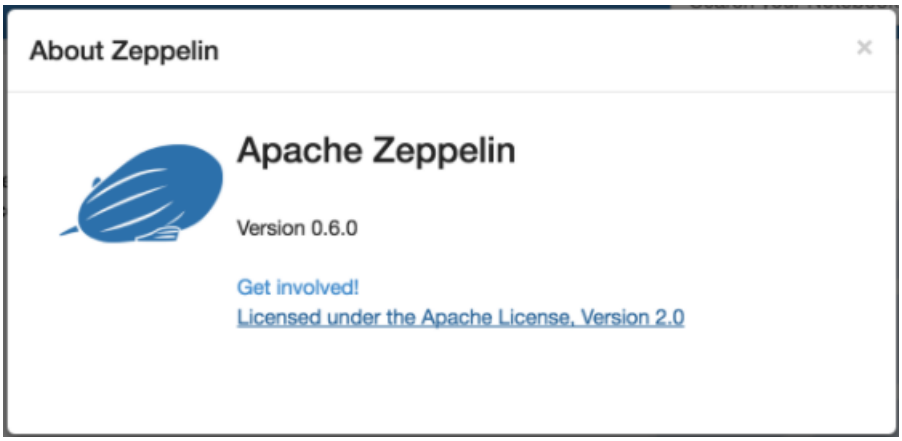



Рис.3.12.: About Zeppelin

Перейдя по ссылке “Interpreter” (см. Рис.3.11.), можно выполнить следующие функции (Рис.3.13.):

- Настроить существующий интерпретатор;
- Добавить/удалить интерпретатор.

Ссылка “Credential” (см. Рис.3.11.) позволяет для источников данных сохранять учетные данные, которые передаются интерпретаторам (Рис.3.14.).

Ссылка “Configuration” (см. Рис.3.11.) отображает все настройки **Apache Zeppelin**, которые заданы в файле конфигурации *$ZEPPELIN_HOME/conf/zeppelin-site.xml* (Рис.3.15.).

### 3.5.3 Набор инструментов

Каждый блокнот **Apache Zeppelin** состоит из нескольких параграфов (Рис.3.16.). Блокнот можно рассматривать как контейнер параграфов.

Каждый параграф состоит из двух разделов: *code section*, в который помещается исходный код, и *result section*, где можно увидеть результат выполнения кода (Рис.3.17.).

В правом верхнем углу каждого параграфа есть несколько команд:

- Выполнить код параграфа;
- Скрыть/показать *code section*;
- Скрыть/показать *result section*;
- Настроить параграф.

Для перехода к настройкам параграфа необходимо нажать на значок шестеренки, при этом открывается контекстное меню (Рис.3.18.).

В диалоговом окне отображается следующая информация и возможные действия:

- Идентификатор параграфа (в данном примере *20150924-163507_134879501*);
- Ширина параграфа. Поскольку Zeppelin использует сетчатую систему Twitter Bootstrap, ширина каждого параграфа может быть изменена от *1* до *12*;
- Переместить параграф на 1 уровень вверх;
- Переместить параграф на 1 уровень вниз;
- Создать новый параграф;

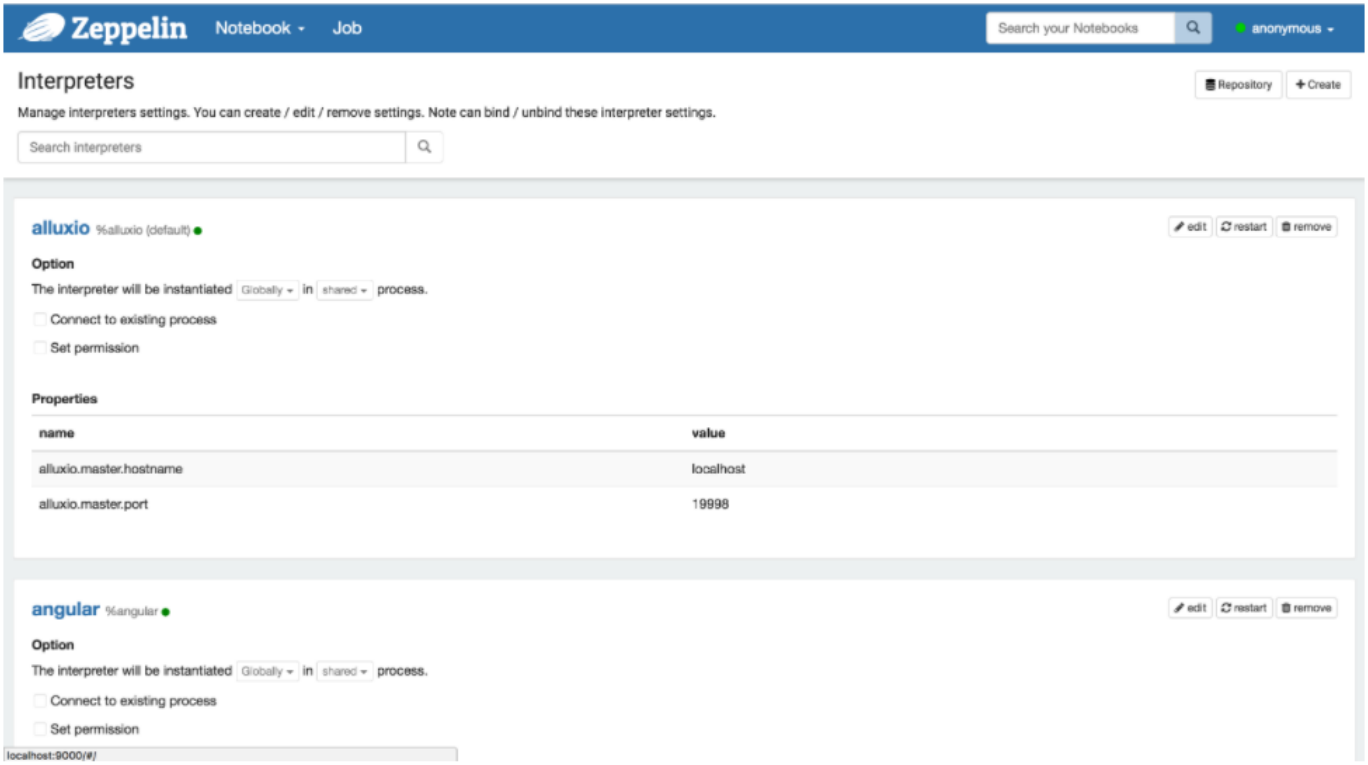



Рис.3.13.: Управление интерпретаторами

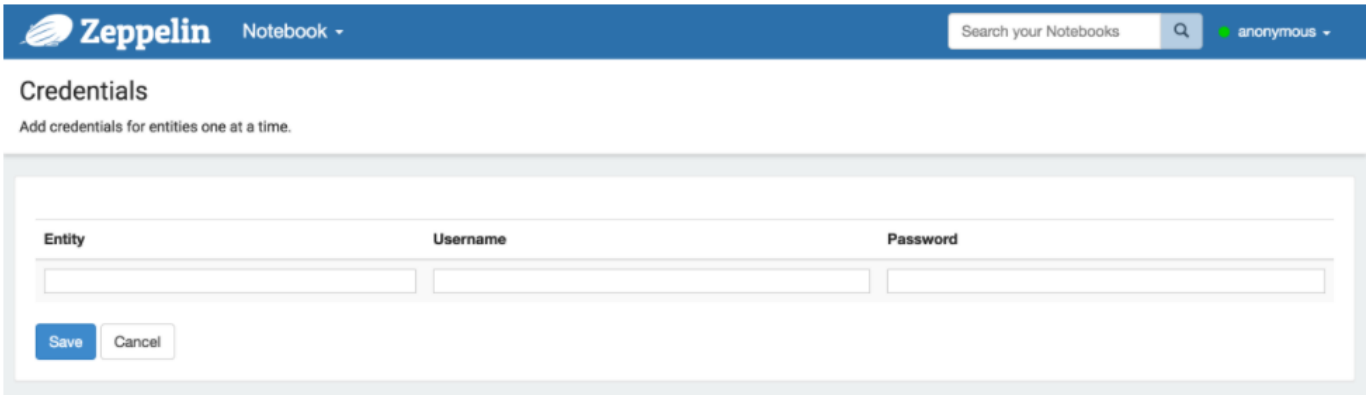



Рис.3.14.: Учетные данные

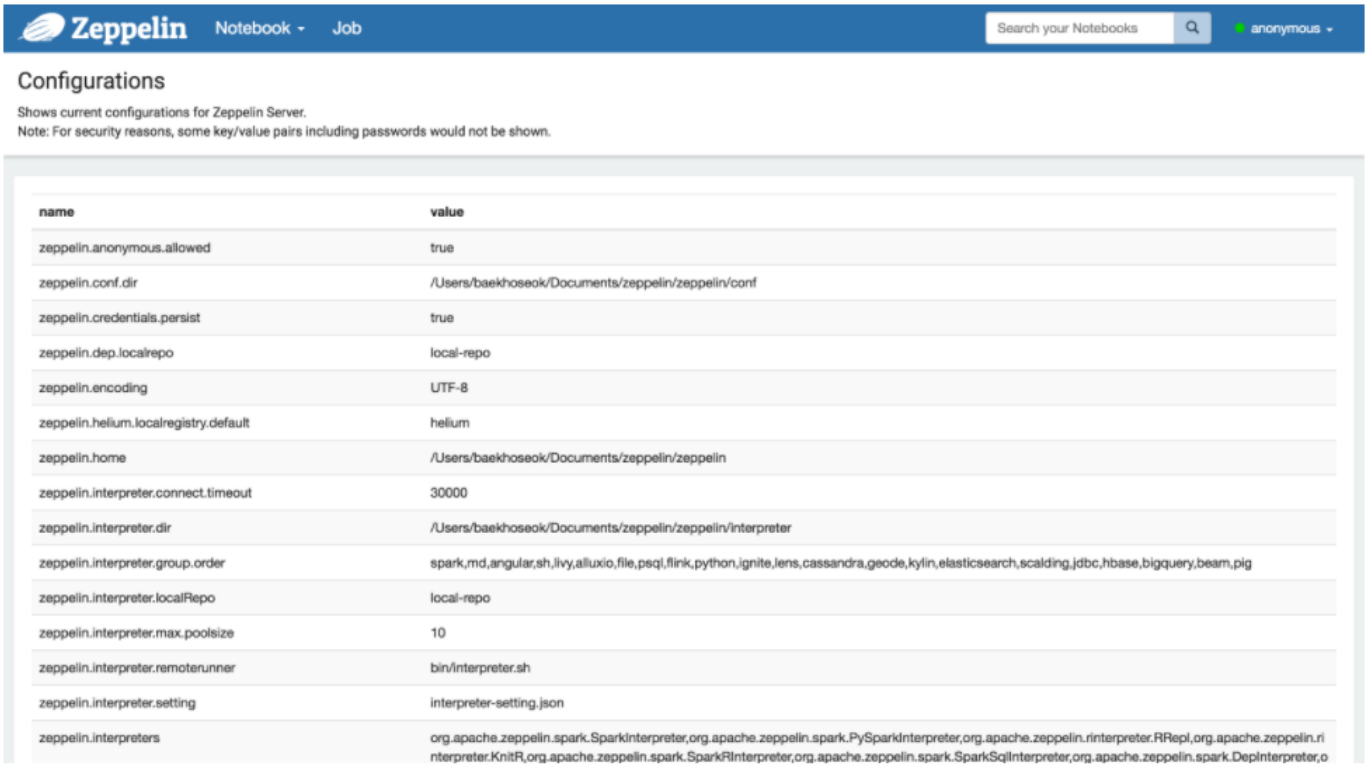



Рис.3.15.: Конфигурация Apache Zeppelin

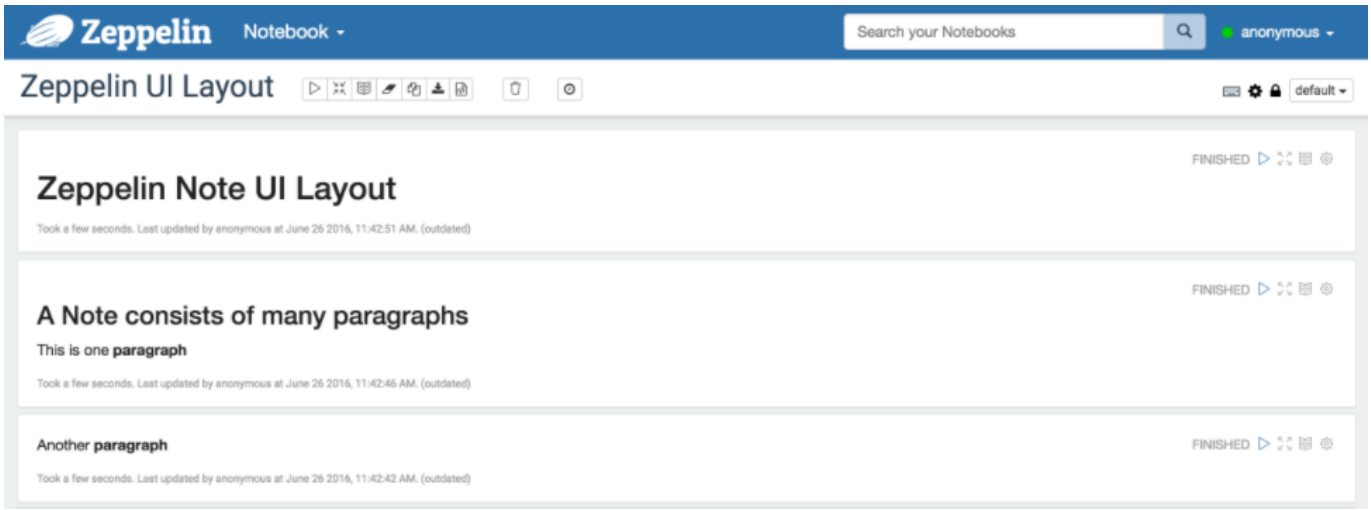



Рис.3.16.: Шаблон блокнота

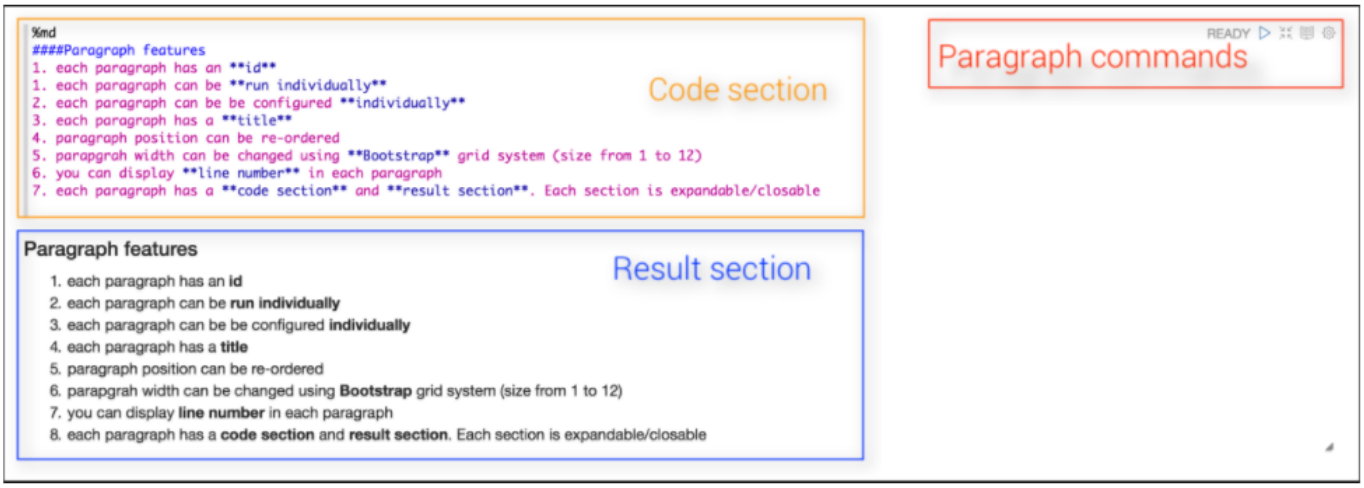



Рис.3.17.: Разделы параграфа

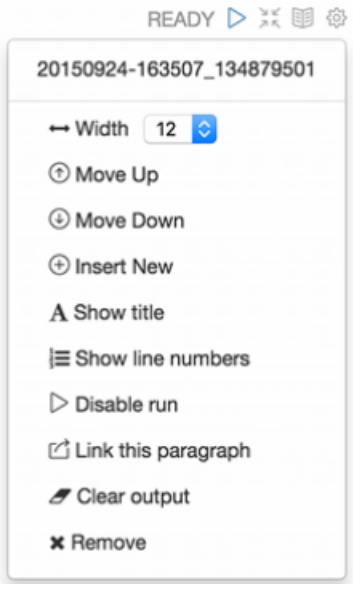



Рис.3.18.: Контекстное меню настроек параграфа

- Изменить название параграфа;
- Показать/скрыть номера строк в *code section*;
- Отключить кнопку запуска для параграфа;
- Экспортировать текущий параграф как *iframe* и открыть *iframe* в новом окне;
- Очистить *result section*;
- Удалить текущий параграф.

### Панель инструментов блокнота

В верхней части экранной формы блокнота **Apache Zeppelin** находится панель инструментов, которая представляет собой командные кнопки настройки, безопасности и отображения (см. Рис.3.16.).

В левом углу панели инструментов отображается название блокнота, необходимо нажать на него, чтобы открыть форму ввода и обновить его. По центру панели находятся следующие кнопки:

- Выполнить все параграфы последовательно в порядке их отображения;
- Скрыть/показать *code section* всех параграфов;
- Скрыть/показать *result section* всех параграфов;
- Очистить *result section* всех параграфов;
- Копировать текущий блокнот;
- Экспортировать текущий блокнот в файл *JSON*. При этом *code section* и *result section* всех параграфов будут экспортированы. При наличии тяжелых данных в *result section*, рекомендуется их очистить перед экспортом;
- Зафиксировать текущее содержимое блокнота;
- Удалить блокнот;
- Запланировать выполнение всех параграфов, используя синтаксис *CRON*.

Справа от панели инструментов блокнота располагаются кнопки конфигурации:

- Отобразить все сочетания клавиш клавиатуры;
- Настроить интерпретаторы, привязанные к текущему блокноту;
- Настроить права для блокнота;
- Переключить режим отображения блокнота на *default*, *simple* или *report*.

## 3.6 Интерпретаторы в Apache Zeppelin


- *Обзор*
- *Интерпретатор Zeppelin*
- *Настройка интерпретатора*
- *Группа интерпретаторов*
- *Режим привязки интерпретатора*
- *Подключение к существующему удаленному интерпретатору*

### 3.6.1 Обзор

В разделе рассказывается об интерпретаторах, их группах и настройках в **Apache Zeppelin**. Концепция интерпретатора **Zeppelin** позволяет подключать любой язык/фреймворк обработки данных. В настоящее время **Zeppelin** поддерживает множество интерпретаторов, таких как **Scala** (с **Apache Spark**), **Python** (с **Apache Spark**), **Spark SQL**, **JDBC**, **Markdown**, **Shell** и другие.

### 3.6.2 Интерпретатор Zeppelin

Интерпретатор **Zeppelin** – это плагин, который позволяет пользователям **Apache Zeppelin** использовать определенный язык/фреймворк обработки данных. Например, чтобы использовать Scala-код в **Zeppelin**, понадобится *%spark* интерпретатор.

На странице интерпретатора при нажатии кнопки “+Create” в открывшемся диалоговом окне в выпадающем списке поля “Interpreter” отображаются все доступные интерпретаторы на сервере (Рис.3.19.).

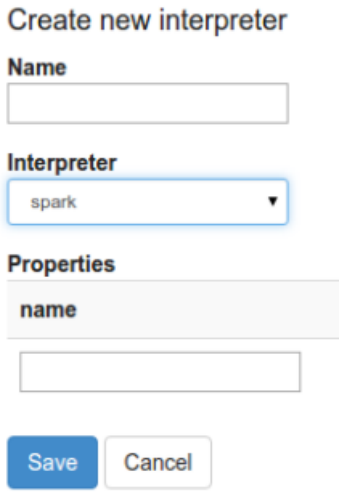



Рис.3.19.: Создать новый интерпретатор

### 3.6.3 Настройка интерпретатора

Настройка интерпретатора **Zeppelin** – это конфигурация данного интерпретатора на сервере **Zeppelin**. Например, ниже приведены свойства, необходимые для подключения интерпретатора **hive JDBC** к **Hive** серверу (Рис.3.20.).

Для экспорта свойств в качестве переменной среды окружения необходимо, чтобы имя свойства состояло из символов верхнего регистра, цифр и подчеркивания (*[A-Z_ 0-9]*). В противном случае свойства задаются как свойство **JVM**.

Каждый блокнот может быть связан с несколькими конфигурациями одного интерпретатора, для этого следует использовать значок настройки в правом верхнем углу блокнота (Рис.3.21.).

### 3.6.4 Группа интерпретаторов

Каждый интерпретатор принадлежит к группе интерпретаторов. **Группа интерпретаторов (Interpreter Group)** – это единый блок, позволяющий одновременно управлять (start/stop) несколькими интерпретаторами. По умолчанию каждый интерпретатор принадлежит к одной группе, но группа может содержать больше интерпретаторов. Например, группа интерпретаторов **Spark** включает поддержку **Spark**, **pySpark**, **Spark SQL** и загрузчик зависимостей *%dep*.

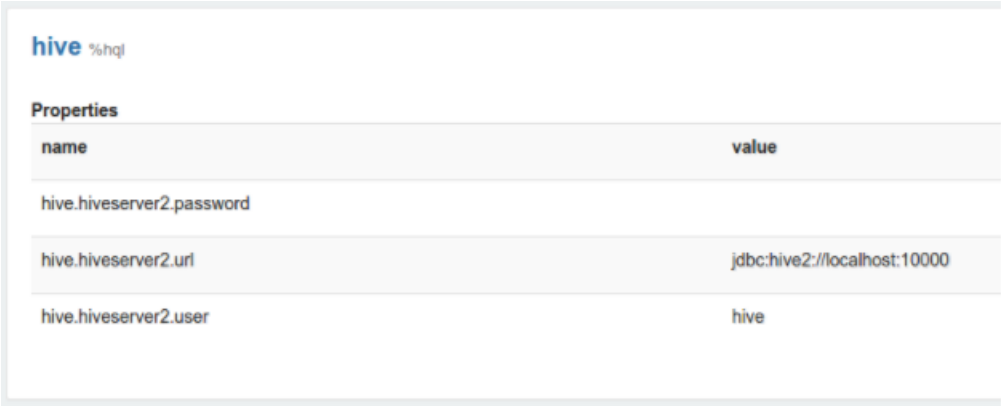



Рис.3.20.: Свойства интерпретатора

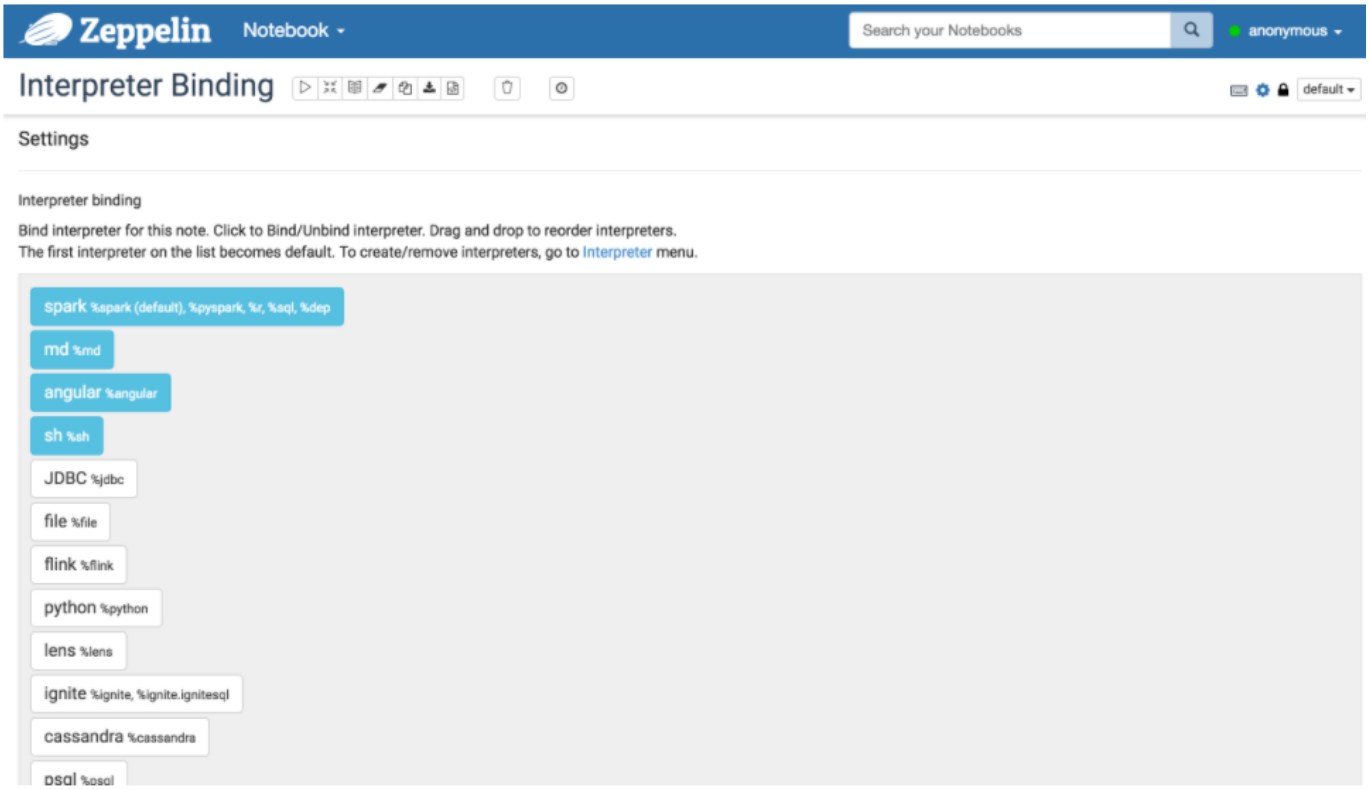



Рис.3.21.: Настройки интерпретатора

Технически, интерпретаторы **Zeppelin** одной группы запускаются в одной **JVM**. Каждый из интерпретаторов может относиться к одной группе. Все их свойства перечислены в настройках интерпретатора (Рис.3.22.).

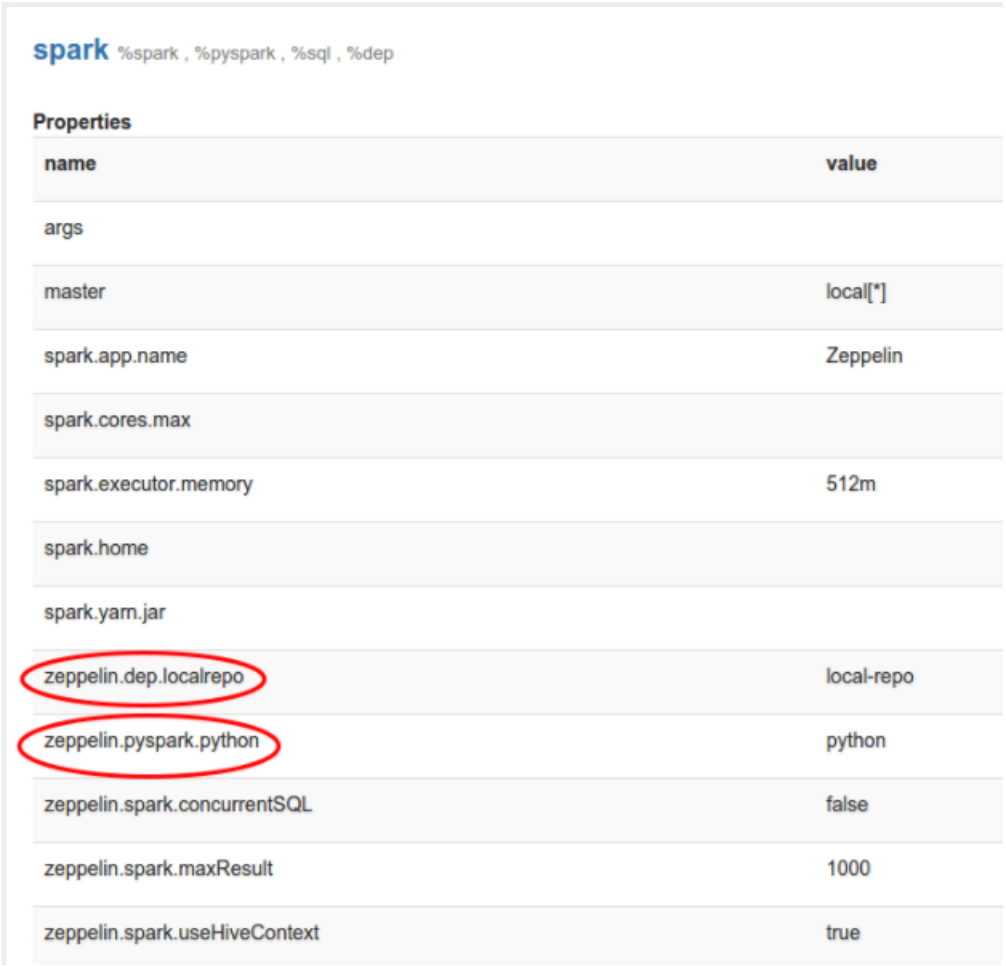



Рис.3.22.: Свойства групп интерпретатора

### 3.6.5 Режим привязки интерпретатора

Каждая конфигурация интерпретатора может быть сделана в одном из приведенных режимов привязки: *"shared"* – общедоступный, *"scoped"* – ограниченный, *"isolated"* – отдельный (Рис.3.23.). В режиме *"shared"* каждый блокнот, связанный с конфигурацией интерпретатора, совместно использует один экземпляр интерпретатора. В режиме *"scoped"* каждый блокнот создает новый экземпляр интерпретатора в том же процессе интерпретатора. В режиме *"isolated"* каждый блокнот создает новый процесс интерпретатора.

### 3.6.6 Подключение к существующему удаленному интерпретатору

Существует возможность запуска потока интерпретатора пользователем **Zeppelin** на удаленном узле. Для этого необходимо создать экземпляр *RemoteInterpreterServer* и запустить его следующим образом:

```
RemoteInterpreterServer interpreter=new RemoteInterpreterServer(3678);
// Here, 3678 is the port on which interpreter will listen.
interpreter.start()
```

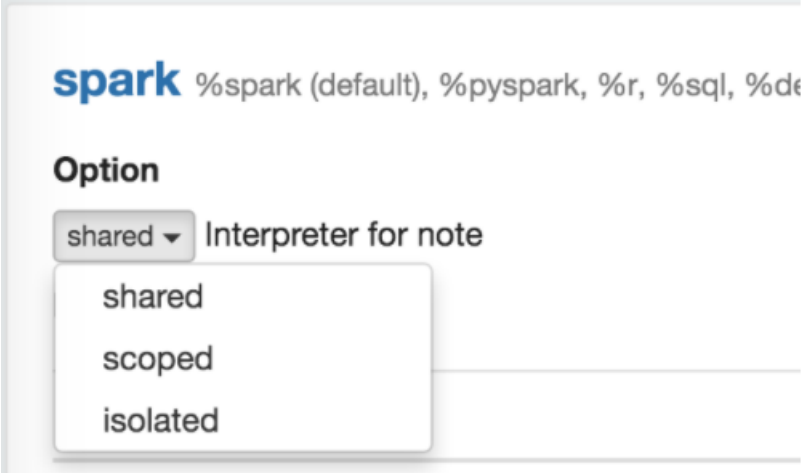



Рис.3.23.: Режимы привязки интерпретатора

Данный код запускает поток интерпретатора внутри процесса. После запуска интерпретатора можно настроить **Zeppelin** для подключения к *RemoteInterpreter*, установив флаг *"Connect to existing process"* и указав узел (*Host*) и порт (*Port*), который слушает процесс интерпретатора (Рис.3.24.).

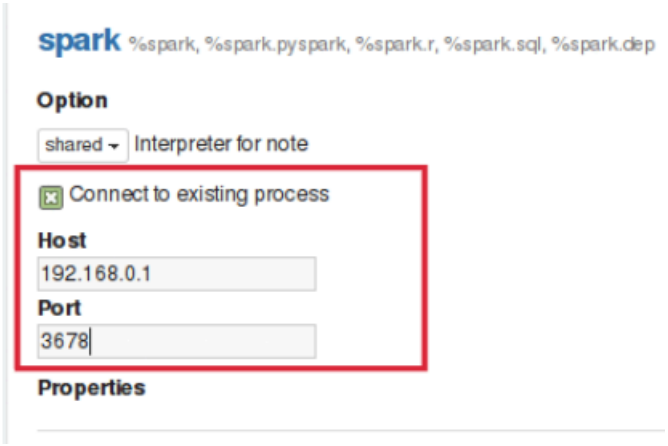



Рис.3.24.: Подключение к удаленному интерпретатору

## 3.7 Python 2 & 3 Интерпретатор для Apache Zeppelin

### 3.7.1 Конфигурирование

Для настройки **Python** для **Apache Zeppelin** необходимо задать параметры, представленные в таблице.

Таблица3.4.: Параметры конфигурации

| Параметр | Значение по умолчанию | Описание |
|---|---|---|
| zeppelin.python | python | Путь к уже установленному бинарному пакету Python (может быть *python2* или *python3*). Если *python* не прописан в *$PATH*, можно задать полный путь (например: */usr/bin/python*) |
| zeppelin.python.maxResult | 1000 | Максимальное количество отображаемых строк датафрейма |

### 3.7.2 Включение интерпретатора Python

Для включения интерпретатора **Python** в блокноте необходимо нажать значок “Gear” и выбрать “Python”.

### 3.7.3 Использование интерпретатора Python

Для выбора интерпретатора **Python** необходимо в параграфе указать *%python*, а затем ввести все команды. Интерпретатор может работать, только если уже установлен *python* (интерпретатор не предоставляет собственные бинарные файлы *python*). Получить справку можно вызовом команды *help()*.

### 3.7.4 Переменные окружения Python

- По умолчанию

По умолчанию *PythonInterpreter* использует команду *python*, определенную в свойстве *zeppelin.python* для запуска процесса *python*. Интерпретатор может использовать все установленные модули (с помощью *pip*, *easy_install* и других)

- Conda

**Conda** – это система управления пакетами и переменными окружения *python*. Интерпретатор *%python.conda* позволяет переключаться между переменными окружения.

Перечень переменных окружения:

```
%python.conda
```

Активация переменных окружения:

```
%python.conda activate [ENVIRONMENT_NAME]
```

Деактивация переменных окружения:

```
%python.conda deactivate
```

- Докер

Интерпретатор *%python.docker* позволяет *PythonInterpreter* создавать процесс *python* в указанном докер-контейнере.

Активация переменных окружения:

```
%python.docker activate [Repository]
%python.docker activate [Repository:Tag]
%python.docker activate [Image Id]
```

Деактивация переменных окружения:

```
%python.docker deactivate
```

Пример:

```
# activate latest tensorflow image as a python environment
%python.docker activate gcr.io/tensorflow/tensorflow:latest
```

### 3.7.5 Использование Zeppelin Dynamic Forms

Динамическую форму **Zeppelin** можно использовать внутри кода **Python** (см. подробное описание).

**Zeppelin Dynamic Form** может использоваться только в том случае, если в системе установлена *py4j Python library*. Библиотеку можно установить с помощью *pip install py4j*.

Пример:

```
%python
### Input form
print (z.input("f1","defaultValue"))

### Select form
print (z.select("f1",[("o1","1"),("o2","2")],"2"))

### Checkbox form
print("".join(z.checkbox("f3", [("o1","1"), ("o2","2")],["1"])))
```

### 3.7.6 Интеграция Matplotlib

Интерпретатор **Python** может автоматически отображать графики *matplotlib* с помощью встроенного модуля *pyplot*:

```
%python
import matplotlib.pyplot as plt
plt.plot([1, 2, 3])
```

Это рекомендуемый метод использования *matplotlib* из блокнота **Zeppelin**. Выходные данные команды по умолчанию преобразовываются в HTML, неявно используя *%html*. Дополнительные настройки можно выполнить с помощью встроенного метода *z.configure_mpl()*. Например:

```
z.configure_mpl(width=400, height=300, fmt='svg')
plt.plot([1, 2, 3])
```

В данном примере изображение задается в формате *SVG 400x300*, которое по умолчанию обычно представляется в формате *600x400* и *PNG* соответственно. В дальнейшем можно будет использовать другую библиотеку, *angular*, которая позволит обновлять график, созданный одним параграфом, непосредственно из другого (выходные данные в таком случае *%angular* вместо *%html*). Функция уже доступна в интерпретаторе **pyspark**.

Если **Zeppelin** не может найти файлы *matplotlib* (которые обычно находятся в *$ZEPPELIN_HOME/interpreter/lib/python*) в *PYTHONPATH*, то программа автоматически устанавливается в *agg* и нижеприведенные инструкции могут использоваться с ограничениями.

Если не удается загрузить встроенные модули, можно использовать *z.show(plt)*:

```
python %python import matplotlib.pyplot as plt plt.figure() (.. ..) z.show(plt)
plt.close()
```

Данная функция *z.show()* может принимать дополнительные параметры для адаптации размеров графика (ширина и высота), а также его выходной формат – *png* или опционально *svg* (Рис.3.25.):

```
%python
z.show(plt, width='50px')
z.show(plt, height='150px', fmt='svg')
```

```
%python
import matplotlib.pyplot as plt
plt.figure()
x = [1, 2, 3, 4, 5, 6, 7, 8]
y = [20, 21, 20.5, 20.81, 21.0, 21.48, 22.0, 21.89]

plt.plot(x, y, linestyle='dashed', marker='o', color='red')
zeppelin_show(plt,width='400px')
plt.close()
```

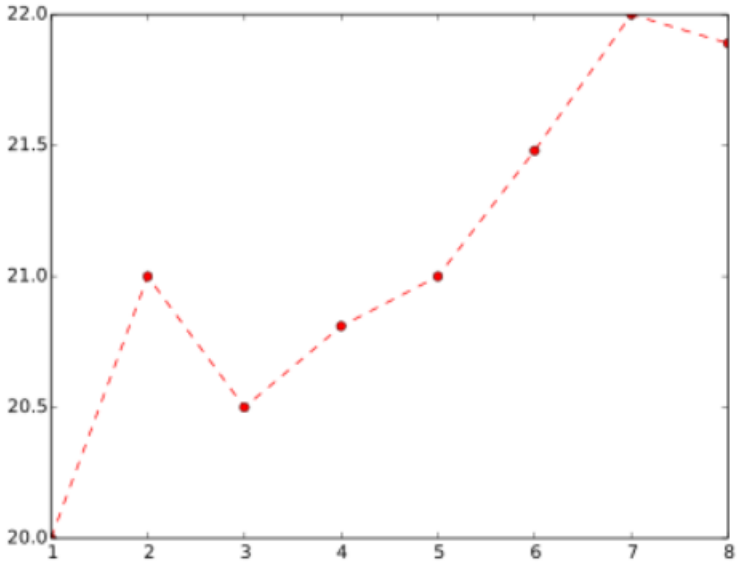



Рис.3.25.: Интеграция Matplotlib

### 3.7.7 Интеграция с Pandas

Система отображения таблиц **Apache Zeppelin** предоставляет встроенные возможности визуализации данных. Интерпретатор **Python** использует его для визуализации датафреймов *Pandas*, аналогично через API функции *z.show()* как в случае интеграции с библиотекой *matplotlib* (*Интеграция Matplotlib*). Например:

```
import pandas as pd
rates = pd.read_csv("bank.csv", sep=";")
z.show(rates)
```

### 3.7.8 SQL поверх датафреймов Pandas

Существует удобный интерпретатор *%python.sql*, который по своему использованию похож на **Apache Spark** в **Zeppelin** и позволяет использовать язык **SQL** для запроса к датафреймам **Pandas** и визуализации результатов через встроенную систему отображения таблиц Table Display System.

Предварительные настройки:

- Pandas *pip install pandas*
- PandaSQL *pip install -U pandasql*

В случае, если по умолчанию выбран интерпретатор **Python** (первый в списке интерпретаторов под значком шестеренки), можно его указывать как просто *%sql*:

- Первый параграф:

```
import pandas as pd
rates = pd.read_csv("bank.csv", sep=";")
```

- Следующий параграф:

```
%sql
SELECT * FROM rates WHERE age < 40
```

В противном случае – *%python.sql*.

### 3.7.9 Техническое описание

Подробные технические сведения о текущей реализации приведены по ссылке python/README.md.

Некоторые функции, еще не реализованные в интерпретаторе **Python**:

- Прерывание выполнения параграфа (способ *cancel ()*) в настоящее время поддерживается только в системах **Linux** и **MacOs**. Если интерпретатор запущен в другой ОС (например, в **MS Windows**), прерывание параграфа завершает работу всего процесса интерпретатора. **JIRA** ticket (*ZEPPELIN-893*) открыт для реализации этой функции в следующей версии интерпретатора;
- Строка состояния в web-интерфейсе (метод *getProgress()*) в настоящее время не реализована;
- Завершение кода в настоящее время не реализовано.