Arenadata™ Database

Версия - v6.21.0_arenadata35

Руководство администратора по работе с кластером ADB

2022 Arenadata™

Оглавление

1 Контроль целостности в ADB (изоляция транзакций) 3
- 1.1 Снапшоты . . . 3
- 1.2 Зацикливание ID Транзакций . . . 4
- 1.3 Режимы изоляции транзакций . . . 5
- 1.4 Удаление неиспользуемых строк из таблиц . . . 6

2 Параллельная загрузка данных 8

3 Поддерживаемые клиентские приложения 9
- 3.1 Клиентские приложения ADB (Client Tools) . . . 9
- 3.2 Клиентские утилиты загрузки данных в ADB (Loader Tools) . . . 10
- 3.3 Подключение с помощью psql . . . 10
- 3.4 Интерфейсы приложений баз данных . . . 11
- 3.5 Инструменты сторонних клиентов . . . 11

4 Шифрование SSL 12
- 4.1 Шифрование соединений Клиент/Сервер . . . 12
- 4.2 Настройка аутентификации SSL . . . 12

5 Инструменты управления 17
- 5.1 Проверка состояния кластера . . . 17
- 5.2 Запуск и остановка ADB . . . 18
- 5.3 Управление файловыми пространствами . . . 22
- 5.4 Установа и удаление gpperfmon . . . 25
- 5.5 Расширение кластера без ADCM . . . 26
- 5.6 Расширение кластера средствами ADCM . . . 40
- 5.7 Добавление и удаление резервного мастера . . . 42
- 5.8 Активация Standby Master . . . 42
- 5.9 Управление сервисом *ADCC* . . . 43
- 5.10 Управление сервисом *Monitoring Clients* . . . 43
- 5.11 Инструментарий Arenadata_toolkit . . . 44

В руководстве приведены сведения для администраторов системы по работе с кластером – по выполнению регламентных операций в ADB, параллельной загрузке данных, по поддерживаемым клиентским приложениям и расширению кластера.

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

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

Глава 1

Контроль целостности в ADB (изоляция транзакций)

Контроль целостности позволяет согласовать конкурентные запросы с правильными результатами, обеспечивая целостность базы данных. Традиционные базы данных используют протокол двухфазной блокировки, что предотвращает транзакцию от изменения данных, которые были прочитаны другой параллельной транзакцией, и не позволяет любой параллельной транзакции считывать или записывать данные, обновленные другой транзакцией. Блокировки, необходимые для координации транзакций, добавляют нагрузку на базу данных, уменьшая общую пропускную способность.

В **ADB** используется **PostgreSQL Multiversion Concurrency Control** (**MVCC**) для управления параллельными транзакциями heap-таблиц. С **MVCC** каждый запрос работает со снапшотом базы данных, создаваемым в тот момент, когда запрос объявляется. Пока выполняется снапшот, запрос не может видеть сделанные другими параллельными транзакциями изменения. Это гарантирует, что запрос видит последовательное представление базы данных. Запросы, которые читают строки, никогда не блокируют транзакции, пишущие строки. И наоборот, запросы, которые записывают строки, не могут быть заблокированы транзакциями, которые читают строки. Это позволяет значительно увеличить параллелизм в сравнении с традиционными базами данных, в которых используется блокировка для координации доступа между транзакциями, которые считывают и записывают данные.

Important: Append optimized таблицы управляются с помощью иной модели управления параллелизмом, нежели чем модель MVCC, которая обсуждалась в данном разделе. Они предназначены для приложений *write-once, read-many*, что никогда (или только очень редко) подразумевает обновления на уровне строк

1.1 Снапшоты

Модель **MVCC** основывается на способности системы управлять несколькими версиями строк данных. Запрос работает со снапшотом базы данных. Снапшот – это набор строк, которые видны в момент транзакции. Снапшот гарантирует, что запрос имеет действительный и корректный снимок базы данных на время выполнения запроса или транзакции.

Каждой транзакции присваивается уникальный идентификатор (*XID*), *32-битное* значение. Оператор **SQL**, не являющийся частью транзакции, обрабатывается как одиночная транзакция (с добавлением *BEGIN* и *COMMIT*). Это похоже на *autocommit*, используемый в некоторых базах данных.

Когда транзакция вставляет строку, *XID* сохраняется со строкой в столбце *xmin*. Когда транзакция удаляет строку, *XID* сохраняется в столбце системы *xmax*. Обновление строки рассматривается как *delete* и

insert, поэтому *XID* сохраняется в *xmax* текущей строки и *xmin* только что вставленного ряда. Столбцы *xmin* и *xmax* вместе со статусом завершения транзакции определяют диапазон транзакций, для которых видна текущая версия строки. Транзакция видит результаты работы всех транзакций меньше, чем *xmin*, но не может видеть результаты работы любой транзакции больше или равной *xmax*.

Операции с несколькими действиями также должны записывать, какая команда в транзакции вставляла строку (*cmin*) или удаляла строку (*cmax*), чтобы транзакция могла видеть изменения, сделанные предыдущими командами в транзакции. Последовательность команд имеет смысл только во время выполнения транзакции, поэтому она сбрасывается до *0* в самом начале.

Каждый сегмент **ADB** имеет свою собственную последовательность *XID*, которая не может сравниваться с *XID* других инстансов. Мастер координирует распределенные транзакции с сегментами, использующими идентификационный номер сеанса, называемый *gp_session_id*. Сегменты сопоставляют идентификаторы распределенных транзакций с их локальными *XID*. Мастер координирует распределенные транзакции с помощью двухфазового протокола фиксации. Если транзакция не выполняется на одном из сегментов, транзакция прекращается на всех сегментах, и данные возвращаются в первоначальный вид.

Столбцы *xmin*, *xmax*, *cmin* и *cmax* для любой строки можно увидеть при помощи оператора *SELECT*:

```
SELECT xmin, xmax, cmin, cmax, * FROM tablename;
```

Поскольку команда *SELECT* выполняется на мастере, *XID* являются идентификаторами распределенных транзакций. В случае если команда выполняется в отдельной базе данных сегментов, значения *xmin* и *xmax* соответствуют локальным значениям *XID*.

1.2 Зацикливание ID Транзакций

Модель **MVCC** использует идентификаторы транзакций (*XID*), чтобы определить, какие строки видны в начале запроса или транзакции. *XID* – это 32-битное значение, поэтому база данных может теоретически выполнить более четырех миллиардов транзакций до того, как значение переполнится и обнулится. Тем не менее, в базе данных **ADB** используется арифметика по модулю *2^32*, что позволяет *XID* зацикливаться. Для любого *XID* может быть около двух миллиардов предыдущих и новых *XID*. Это работает до тех пор, пока текущая версия строки примерно через два миллиарда транзакций неожиданно не станет новой строкой. Чтобы предотвратить это, **ADB** имеет специальный *XID*, называемый *FrozenXID*, который всегда считается старше обычного *XID*, с которым он сравнивается. *Xmin* строки должен быть заменен на *FrozenXID* в течение двух миллиардов транзакций, и это одна из функций, выполняемых командой *VACUUM*.

Очистка (vacuuming) как минимум раз в два миллиарда транзакций предотвращает зацикливание *XID*. База данных **ADB** отслеживает *XID* и предупреждает, когда требуется произвести очистку (операция *VACUUM*).

Когда значительная часть идентификаторов больше недоступна и до того, как происходит зацикливание *XID*, выдается предупреждение:

```
WARNING: database "database_name"must be vacuumed within number_of_transactions

Transactions
```

Если операция *VACUUM* не выполняется, база данных **ADB** перестает создавать транзакции, чтобы избежать возможной потери данных. При достижении лимита, система сообщает об ошибке:

```
FATAL: database is not accepting commands to avoid wraparound data loss in database

"database_name"
```

Параметры конфигурации сервера *xid_warn_limit* и *xid_stop_limit* управляют отображением предупреждений и ошибок. Параметр *xid_warn_limit* показывает количество идентификаторов транзакций перед значением *xid_stop_limit*. А параметр *xid_stop_limit* показывает количество *XID* перед тем, как происходит зацикливание и выдается ошибка.

1.3 Режимы изоляции транзакций

Стандарт **SQL** описывает три явления, которые могут возникать при запуске транзакций базы данных одновременно:

- *Грязное чтение* – явление, которое возникает, когда транзакция считывает незафиксированные данные из другой параллельной транзакции;
- *Неповторяющееся чтение* – ситуация, когда при повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются измененными;
- *Чтение фантомов* – ситуация, когда при повторном чтении в рамках одной транзакции одна и та же выборка дает разные множества строк.

Стандарт **SQL** определяет четыре режима изоляции транзакций, которые должны поддерживать системы баз данных:

Команды **SQL** базы данных **ADB** позволяют запросить *READ UNCOMMITTED*, *READ COMMITTED*, или *SERIALIZABLE*. База данных **ADB** рассматривает *READ UNCOMMITTED* так же, как *READ COMMITTED*. Запрос *REPEATABLE READ* вызывает ошибку; вместо этого необходимо использовать *SERIALIZABLE*. Режим изоляции по умолчанию – *READ COMMITTED*.

Разница между *READ COMMITTED* и *SERIALIZABLE* заключается в том, что в режиме *READ COMMITTED* каждый оператор в транзакции видит только строки, обновленные до начала выполнения оператора, в то время как в режиме *SERIALIZABLE mode* все операторы транзакции видят строки, обновленные до начала транзакции.

Режим изоляции *READ COMMITTED* обеспечивает лучшее распараллеливание и лучшую производительность, чем режим *SERIALIZABLE*. Он допускает неповторяющиеся чтения, где значения в строке, полученные дважды в транзакции, могут отличаться, поскольку другая параллельная транзакция совершила изменения. Режим *READ COMMITTED* также позволяет делать фантомные чтения, где запрос, выполненный дважды в одной и той же транзакции, может возвращать два разных набора строк.

Режим изоляции *SERIALIZABLE* предотвращает как неповторяющиеся чтения, так и фантомные чтения, но за счет ухудшения распараллеливания и производительности. Все параллельные транзакции имеют одинаковый снимок базы данных, полученный перед началом транзакций. Параллельная транзакция, которая пытается изменить данные, уже измененные другой транзакцией, получает запрет на это действие. Приложения, выполняющие транзакции в режиме *SERIALIZABLE*, должны быть подготовлены к обработке транзакций, не выполняющихся из-за ошибок сериализации. Если режим изоляции *SERIALIZABLE* жестко не требуется, лучше использовать режим *READ COMMITTED*.

Стандарт **SQL** подразумевает, что параллельные упорядоченные транзакции приводят базу данных в одно состояние в независимости от типа их обработки (т.е. в параллели или друг за другом). Модель изоляции снапшотов **MVCC** предотвращает грязные чтения, неповторяющиеся чтения и фантомные чтения, не используя блокировку. Однако, иные взаимодействия, которые могут возникнуть между некоторыми транзакциями *SERIALIZABLE* в базе данных **ADB**, не позволяют им называться полностью упорядоченными. Эти аномалии можно отнести к тому факту, что база данных **ADB** не выполняет блокировку предикатов, а это означает, что запись одной транзакции может повлиять на результат предыдущего чтения в другой параллельной транзакции.

Транзакции, которые выполняются одновременно, должны быть проверены на предмет взаимодействий, которые не могут быть предотвращены путем запрета параллельных обновлений одних и тех же данных. Определенные проблемы можно предотвратить, используя блокировки таблиц или потребовав, чтобы конфликтующие транзакции обновили фиктивную строку, введенную для представления конфликта.

Оператор **SQL** *SET TRANSACTION ISOLATION LEVEL* устанавливает режим изоляции для текущей транзакции. Режим должен быть установлен перед любыми операциями *SELECT, INSERT, DELETE, UPDATE* или *COPY*:

```
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
...
COMMIT;
```

Режим изоляции также может быть указан как часть инструкции *BEGIN*:

```
BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Режим изоляции транзакции по умолчанию можно изменить с помощью свойства *default_transaction_isolation*.

1.4 Удаление неиспользуемых строк из таблиц

Обновление или удаление строки оставляет предыдущую версию строки в таблице. Если строка с истекшим сроком действия указана в активных транзакциях, ее можно удалить, а занимаемое ею пространство повторно использовать. За это действие отвечает команда *VACUUM*.

Когда недействительные строки накапливаются в таблице, дисковые файлы должны быть расширены для размещения новых строк. Увеличенная нагрузка на ввод/вывод дисков, используемых для обработки запросов, негативно влияет на производительность. Эта ситуация называется раздутие (bloat), и её следует контролировать с помощью регулярной чистки.

Команда *VACUUM* может работать одновременно с другими запросами. Она удаляет неиспользуемые строки со страниц и переупаковывает оставшиеся строки для консолидации свободного места. Если после очистки осталось много свободного пространства, в таблицу свободного пространства добавляется страница. Позднее, когда базе данных требуется пространство для новых строк, она обращается к карте свободного пространства таблицы. Если ни одна страница не найдена, добавляются новые страницы.

VACUUM не консолидирует страницы и не уменьшает размер таблицы на диске. Пространство, которое эта команда восстанавливает, доступно только через карту свободного пространства. Чтобы размер файлов на диске не рос, важно хотя бы один раз в день запускать *VACUUM*. Также важно запускать *VACUUM* после выполнения транзакции, которая обновляет или удаляет большое количество строк.

Команда *VACUUM FULL* перезаписывает таблицу без неиспользуемых строк, сводя ее к минимальному размеру. Для создания новой таблицы должно быть достаточно места на диске. При этом таблица блокируется до тех пор, пока команда *VACUUM FULL* не завершится. Это очень ресурсоемко по сравнению с обычной командой *VACUUM*, и ее можно избежать или отложить путем регулярной чистки. Лучше всего запускать *VACUUM FULL* в течение периода технического обслуживания. Альтернативой *VACUUM FULL* является воссоздание таблицы с помощью инструкции *CREATE TABLE AS* с последующим удалением старой таблицы.

Карта свободного пространства находится в общей памяти и отслеживает свободное пространство для всех таблиц и индексов. Каждая таблица или индекс использует около *60 байт* памяти, и каждая страница со свободным пространством занимает *6 байт*. Два параметра конфигурации системы определяют размер карты свободного пространства *max_fsm_pages* и *max_fsm_relations*.

- `max_fsm_pages`

  Данный параметр устанавливает максимальное количество дисковых страниц, которые могут быть добавлены в общую карту свободного пространства. Для каждого слота страницы потребляется *6 байт* общей памяти. Значение по умолчанию - *200000*. Этот параметр должен быть установлен как минимум в *16* раз больше значения *max_fsm_relations*.

- `max_fsm_relations`

  Параметр устанавливает максимальное количество отношений, которые отслеживаются в карте свободного пространства. Этот параметр должен быть установлен больше, чем общее количество *таблиц + индексов + системных таблиц*. Значение по умолчанию - *1000*. Для каждого отношения

к каждому сегменту потребляется около *60 байт* памяти. Рекомендуется устанавливать параметр на более высокое значение.

Если карта свободного пространства слишком маленькая, дисковые страницы с доступным пространством не добавляются на карту, и это пространство не может быть повторно использовано до тех пор, пока не будет запущена, по меньшей мере, следующая команда *VACUUM*. Это приводит к росту файлов.

Можно запустить *VACUUM VERBOSE tablename*, чтобы получить отчет по сегменту о количестве удаленных строк, количестве затронутых страниц и количестве страниц с полезным свободным пространством.

Чтобы узнать, сколько страниц таблица использует во всех сегментах, необходимо запросить системную таблицу *pg_class*. Чтобы получить точные данные обязательно следует сначала выполнить *ANALYZE* для таблицы.

```
SELECT relname, relpages, reltuples FROM pg_class WHERE relname = 'tablename';
```

Другим полезным инструментом является *gp_bloat_diag* в схеме *gp_toolkit*, который идентифицирует раздутие (bloat) в таблицах путем сравнения фактического количества используемых таблицей страниц с ожидаемым числом.

Глава 2

Параллельная загрузка данных

В разделе приводится краткое описание методов загрузки данных в **ADB**.

В крупномасштабном хранилище большие объемы данных должны загружаться за довольно короткий промежуток времени. **ADB** поддерживает быструю параллельную загрузку данных с помощью функции внешних таблиц. Администраторы также могут загружать внешние таблицы в режиме изоляции ошибочных строк, чтобы фильтровать ошибочные строки в отдельную таблицу, продолжая загрузку правильно отформатированных строк. Администраторы могут указать порог ошибок для операции загрузки, чтобы контролировать, какое количество неправильно отформатированных строк заставляет **ADB** прерывать операцию загрузки.

Используя внешние таблицы в сочетании с параллельным файловым сервером (**gpfdist**) можно достичь максимального распараллеливания и пропускной способности базы данных.

Другая утилита **ADB** – **gpload** – запускает задачу загрузки, указанную в управляемом файле в формате *YAML*. Необходимо описать местоположение исходных данных, формат, необходимые преобразования, участвующие хосты, адресаты баз данных и другие данные в файле управления, после чего **gpload** выполняет загрузку. Это позволяет описать сложную задачу и выполнить ее контролируемым системным образом.

Глава 3

Поддерживаемые клиентские приложения

Пользователи могут подключаться к базе данных **ADB** с помощью различных клиентских приложений:

- Ряд клиентских приложений **ADB** предоставляется с установщиком. Клиентское приложение *psql* предоставляет интерфейс командной строки;
- Используя стандартные интерфейсы приложений баз данных, такие как **ODBC** и **JDBC**, пользователи могут создавать свои собственные клиентские приложения, взаимодействующие с базой данных **ADB**. Поскольку СУБД **ADB** основана на **PostgreSQL**, она использует стандартные драйверы баз данных **PostgreSQL**.
  - с **ADB 5** совместимы **ODBC** и **JDBC** драйвера для PostgreSQL 8.3.0 или выше.
  - с **ADB 6** совместимы **ODBC** и **JDBC** драйвера для PostgreSQL 9.4.24 или выше.
- Большинство сторонних клиентских инструментов, которые используют стандартные интерфейсы базы данных, такие как **ODBC** и **JDBC**, могут быть настроены для подключения к **ADB**.

3.1 Клиентские приложения ADB (Client Tools)

База данных **ADB** поставляется с несколькими клиентскими приложениями, расположенными в *$GPHOME/bin* главной хост-системы. В таблице перечислены наиболее часто используемые клиентские приложения.

Таблица3.1.: Клиентские приложения

Имя	Применение
createdb	Создать новую базу данных
createlang	Определить новый процедурный язык
createuser	Определить новую роль базы данных
dropdb	Удалить базу данных
droplang	Удалить процедурный язык
dropuser	Удалить роль
PSQL	Интерактивный терминал *PostgreSQL*
reindexdb	Переиндексировать базу данных
vacuumdb	Сбор мусора и анализ базы данных

При использовании клиентских приложений следует подключиться к базе данных с помощью инстанса мастера **ADB**. Необходимо узнать имя целевой базы данных, имя хоста и номер порта мастера, а также имя пользователя базы данных для подключения. Эта информация может быть предоставлена в командной строке

с использованием опций `-d`, `-h`, `-p` и `-U` соответственно. Если находится аргумент, не принадлежащий ни одному из параметров, он интерпретируется как имя базы данных.

Все эти параметры имеют значения по умолчанию, которые используются, если опция не указана. По умолчанию, хостом является локальный хост. Номер порта – *5432*. Имя пользователя – это имя пользователя системы ОС, как и имя базы данных по умолчанию. Следует обратить внимание, что имена пользователей ОС и имена пользователей **ADB** необязательно должны быть одинаковы. Если значения по умолчанию неверны, можно установить переменные *PGDATABASE*, *PGHOST*, *PGPORT* и *PGUSER* на соответствующие значения или использовать файл *psql~/.pgpass* для хранения часто используемых паролей.

3.2 Клиентские утилиты загрузки данных в ADB (Loader Tools)

Для некоторых операционных систем, для которых не поставляется серверная часть ADB, есть возможность установить клиентские утилиты загрузки данных в ADB: утилиты *gpfdist* и *gpload*. В данный момент в поставке ADB дополнительно присутствуют клиентские утилиты загрузки данных из следующих операционных систем:

- Red Hat 8.5

Клиентская машина с приложениями для загрузки данных должна: + иметь сетевой доступ со всемт нодами Arenadata DB (мастер-нодой и сегмент-нодами). + иметь сетевой доступ к ETL-серверам, на которых лежат данные для загрузки.

Для установки клиентских утилит загрузки данных в ADB надо скачать rpm-пакет `gpdb-loaders-<Версия ADB>.rpm` на клиентскую машину и установить через пакетный менеджер операционнной системы. Утилиты будут доступны из директории `/usr/lib/gpdb/bin/`.

Также потребуется применить файл `/usr/lib/gpdb/greenplum_path.sh` перед вызовом утилит загруки. Например, на Red Hat нужно запустить:

```
$ source /usr/lib/gpdb/greenplum_path.sh
```

Для работы *gpload* дополнительно требуется Python 2 версии 2.7 и выше с модулями *pyyaml*, *pygresql*.

3.3 Подключение с помощью psql

В зависимости от используемых значений следующие примеры показывают, как получить доступ к базе данных через *psql*:

```
$ psql -d gpdatabase -h master_host -p 5432 -U gpadmin
$ psql gpdatabase
$ psql
```

Если пользовательская база данных еще не создана, можно получить доступ к системе, подключившись к базе данных *template1*. Например:

```
$ psql template1
```

После подключения к базе данных *psql* предоставляет приглашение с именем базы данных, к которой в настоящее время подключен *psql*, а затем строковой тип `=>` (или `= #` для суперпользователя базы данных). Например:

```
gpdatabase =>'
```

В командной строке можно вводить команды **SQL** для отправки на сервер и для выполнения, при этом каждая команда должна заканчиваться на знак `;`. Например:

```
=> SELECT * FROM mytable;
```

3.4 Интерфейсы приложений баз данных

Есть возможность разработки собственных клиентских приложений, взаимодействующих с **ADB**. **PostgreSQL** предоставляет ряд драйверов баз данных для наиболее часто используемых **API**, которые также могут использоваться с **ADB**. Эти драйверы не упакованы базовым дистрибутивом **ADB**. Каждый драйвер является независимым проектом разработки **PostgreSQL** и должен быть загружен, установлен и настроен для подключения к **ADB**. Доступные драйверы представлены в таблице.

Таблица3.2.: Интерфейсы базы данных

API	Драйвер PostgreSQL	Ссылка для скачивания
ODBC	pgodbc	https://odbc.postgresql.org/
JDBC	pgjdbc	https://jdbc.postgresql.org/
Perl DBI	pgperl	http://gborg.PostgreSQL.org/project/pgperl
Python DBI	pygresql	http://www.pygresql.org

Общие инструкции по доступу к **ADB** с помощью **API**:

1. Загрузить платформу и соответствующий **API** из источника. Например, можно получить Java-комплект разработчика (**JDK**) и **API JDBC** от **Sun**.
2. Записать клиентское приложение в соответствии со спецификациями **API**. При программировании приложения необходимо следить за тем, чтобы не был включен какой-либо неподдерживаемый синтаксис **SQL**.
3. Загрузить соответствующий драйвер **PostgreSQL** и настроить подключение к мастер-инстансу базы данных **ADB**. **ADB** предоставляет пакет клиентских инструментов, содержащий поддерживаемые драйверы.

3.5 Инструменты сторонних клиентов

В большинстве сторонних средств **ETL** и бизнес-аналитики **BI** используются стандартные интерфейсы баз данных, такие как **ODBC** и **JDBC**. Их можно настроить для подключения к **ADB**, которая работает со следующими инструментами:

- Business Objects;
- Microstrategy;
- Informatica Power Center;
- Microsoft SQL Server Integration Services (SSIS) and Reporting Services (SSRS);
- Ascential Datastage;
- SAS;
- Cognos.

Квалифицированные специалисты **Arenadata** могут помочь пользователям в настройке выбранного стороннего инструмента для его использования с базой данных.

Глава 4

Шифрование SSL

4.1 Шифрование соединений Клиент/Сервер

ADB обладает нативной поддержкой соединения **SSL** между клиентом и мастер-сервером. Соединение **SSL** предотвращает доступ к пакетам третьей стороны и предотвращает атаки посредника. **SSL** должно использоваться, когда соединение проходит по незащищенной ссылке, а также когда используется аутентификация сертификата клиента.

Для включения **SSL** требуется установленный **OpenSSL** как на клиенте, так и на мастер-сервере. **ADB** может запускаться с включенным **SSL** если настроить параметры конфигурации сервера ssl=on в мастере *postgresql.conf*. При запуске в режиме **SSL**, сервер будет искать файлы *server.key* (частный ключ сервера) и *server.crt* (сертификат сервера) в папке master data. Эти файлы должны быть настроены для запуска системы **ADB** с включенным **SSL**.

> **Warning:** Не используйте пароль для защиты частного ключа. Сервер не запрашивает пароля для частного ключа и БД не сможет запуститься, если должна затребовать его.

Для тестирования системы можно использовать самоподписанный сертификат, но в штатной работе необходим сертификат подписанный **certificate authority** (CA), чтобы клиент мог подтвердить сервер. CA может быть локальный и глобальный. Если все клиенты принадлежат локальной организации, рекомендуется использовать локальный CA.

4.2 Настройка аутентификации SSL

В процессе аутентификации **SSL** сравниваются *Common Name* (cn) сертификата, который предоставляет программа-клиент при подключении, и запрашиваемое имя пользователя Базы данных. Пользователь не должен существовать в БД. Для обеспечения соединения между системой и БД можно использовать map-файл.

4.2.1 Параметры аутентификации SSL

При аутентификации по сертификату доступны следующие опции:

- *Hostssl* - тип подключения должен быть *hostssl*.
- *map* - маппинг. Указывается в *pg_ident.conf*, или в файле, указанном в настройках сервера *ident_file*.

  Пример *pg_hba.conf*:

```
Hostssl testdb certuser 192.168.0.0/16 cert
Hostssl testdb all 192.168.0.0/16 cert map=gpuser
```

4.2.2 Конфигурация OpenSSL

ADB считывает файл конфигурации **OpenSSL**, указанный в *$GP_HOME/etc/openssl.cnf* по умолчанию. Для изменения настроек **OpenSSL** нужно внести изменения в конфигурационный файл и перезапустить сервер.

4.2.3 Пример создания самоподписанного сертификата

> **Warning:** Использовать самоподписанный сертификат рекомендуется только в целях тестирования системы! Для продуктивного окружения используйте проверенные удостоверяющие центры (CA) для подписания сертификатов.

Для создания самоподписанного сертификата необходимо:

1. Ввести команду openssl:

```
openssl req -new -text -out server.req
```

2. Ввести запрашиваемую информацию. Убедитесь, что вы указали локальный хост в качестве *Common Name*. Пароль можно оставить пустым.
3. Программа создает ключ, защищенный паролем, который должен быть не короче четырех символов. Его необходимо отключить для автоматического запуска, для чего ввести команду:

```
openssl rsa -in privkey.pem -out server.key
rm privkey.pem
```

4. Ввести старый пароль, чтобы разблокировать существующий ключ.
5. Выполнить команду:

```
openssl req -x509 -in server.req -text -key server.key -out server.crt
```

Данная команда превращает сертификат в самоподписанный и копирует ключ и сертификат туда, где сервер будет их искать.

6. Выполнить команду:

```
chmod og-rwx server.key
```

7. Скопировать доверенный сертификат *root.crt* на базе *server.crt*:

```
cp server.crt root.crt
```

8. Перенести *root.crt* на клиентскую машину (например через scp) для проверки работы **SSL** в различных режимах. Файл *root.crt* размещается на клиенте в домашнем каталоге пользователя (в **Microsoft Windows** эта директория называется *%APPDATA%postgresql*).

Более подробно о создании частного ключа и сертификата, см. документацию OpenSSL.

4.2.4 Конфигурация postgresql.conf для аутентификации SSL

Следующие настройки сервера должны быть указаны в конфигурационном файле *postgresql.conf*:

- *ssl [boolean]* - включить соединение **SSL**.
- *ssl_renegotiation_limit [integer]* - предел передачи данных до нового запроса ключа.
- *ssl_ciphers [string]* - список разрешенных шифрований **SSL**.

В папке *$MASTER_DATA_DIRECTORY* присутствуют следующие серверные файлы:

- *server.crt* - сертификат сервера.
- *server.key* - ключ сервера.
- *root.crt* - доверенные авторизации сертификата.
- *root.crl* - сертификаты, отозванные доверенными авторизациями.

Файлы *server.crt* и *server.key* сохраняются на сервере.

Файлы *root.crt* и *root.crl* сохраняются на клиенте в домашнем каталоге пользователя (в **Microsoft Windows** эта директория называется *%APPDATA%postgresql*). Это делается, чтобы клиент мог убедиться в том, что конечный сертификат сервера подписан центром сертификации, которому он доверяет. Если на клиенте существует файл *root.crl*, при проверке также учитывается содержащийся в нём список отозванных сертификатов **Certificate Revocation List** (**CRL**).

Более подробно о настройках сертификатов **SSL** на сервере см. документацию Postgres.

4.2.5 Конфигурация соединения SSL клиента

Доступны следующие опции **SSL**:

- *sslmode* - устанавливает уровень защиты (режимы подключения по **SSL**). Доступны следующие значения:

sslmode	Защита от прослушивания	Защита от MITM	Политика применения
disable	Нет	Нет	Мне не важна безопасность и я не приемлю издержки, связанные с шифрованием.
allow	Возможно	Нет	Мне не важна безопасность, но я приемлю издержки, связанные с шифрованием, если на этом настаивает сервер.
prefer	Возможно	Нет	Мне не важна безопасность, но я предпочитаю шифрование (и приемлю связанные издержки), если это поддерживает сервер.
require	Да	Нет	Я хочу, чтобы мои данные шифровались, и я приемлю сопутствующие издержки. Я доверяю сети в том, что она обеспечивает подключение к нужному серверу.
verify-ca	Да	Зависит от политики ЦС	Я хочу, чтобы мои данные шифровались, и я приемлю сопутствующие издержки. Мне нужна уверенность в том, что я подключаюсь к доверенному серверу.
verify-full	Да	Да	Я хочу, чтобы мои данные шифровались, и я приемлю сопутствующие издержки. Мне нужна уверенность в том, что я подключаюсь к доверенному серверу и это именно указанный мной сервер.

- *sslcert* - имя файла клиентского сертификата **SSL**. Расположение по умолчанию: *~/.postgresql/postgresql.crt*.
- *sslkey* - тайный ключ, используемый клиентским сертификатом. Расположение по умолчанию: *~/.postgresql/postgresql.key*.
- *sslrootcert* - имя файла, содержащего сертификат(ы) **SSL Certificate Authority**. Расположение по умолчанию: *~/.postgresql/root.crt*.
- *sslcrl* - имя списка отозванных сертификатов **SSL**. Расположение по умолчанию: *~/.postgresql/root.crl*.

Для настройки клиентских параметров можно использовать следующие переменные:

- *sslmode* – PGSSLMODE
- *sslkey* – PGSSLKEY
- *sslrootcert* – PGSSLROOTCERT
- *sslcert* – PGSSLCERT
- *sslcrl* – PGSSLCRL

Более подробно о настройках сертификатов **SSL** на сервере см. документацию Postgres.

4.2.6 Примеры подключения с клиента на сервер ADB в различных режимах SSL

В режиме *require* наличие *root.crt* сертификата на клиенте не требуется:

```
psql "sslmode=require host=test-ssl dbname=gpadmin user=gpadmin"
psql (9.4.24)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.
gpadmin=#
```

В режиме *verify-ca* отсутствие *root.crt* сертификата на клиенте приведет к отказу подключения:

```
psql "sslmode=verify-ca host=test-ssl dbname=gpadmin user=gpadmin"
psql: root certificate file "/home/gpadmin/.postgresql/root.crt" does not exist
Either provide the file or change sslmode to disable server certificate verification.
```

После добавления *root.crt* подключение станет возможным:

```
psql "sslmode=verify-ca host=test-ssl dbname=gpadmin user=gpadmin"
psql (9.4.24)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.
gpadmin=#
```

В режиме *verify-full* требуется наличие *root.crt* сертификата на клиенте и совпадение *hostname* и *server CN*, иначе в подключении будет отказано:

```
psql "sslmode=verify-full host=test-ssl dbname=gpadmin user=gpadmin"
psql: server common name "ssl-server" does not match host name "test-ssl"
```

Если пересоздать сертификаты, указав при создании сертификата *Server CN*, который будем использовать при подключении, станет возможным соединение по **SSL**:

```
psql "sslmode=verify-full host=test-ssl dbname=gpadmin user=gpadmin"
psql (9.4.24)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.
gpadmin=#
```

4.2.7 Проверка SSL соединения с помощью расширения sslinfo

Проверить, использует ли текущее соединение **SSL** можно с помощью функций расширения `sslinfo`, например:

```
gpadmin=# create extension sslinfo;
CREATE EXTENSION
gpadmin=# select ssl_is_used();
 ssl_is_used
-------------
 t
(1 row)
```

Более подробную информацию по расширению `sslinfo` см. документацию Postgres.

Глава 5

Инструменты управления

5.1 Проверка состояния кластера

Проверка состояния кластера **ADB** запускает утилиту **gpstate utility** и выполняет распределенный запрос по базе данных.

Для проверки результатов действий над кластером необходимо:

1. Открыть кластер *adb* в ADCM и нажать кнопку “Check” (Рис.5.1).

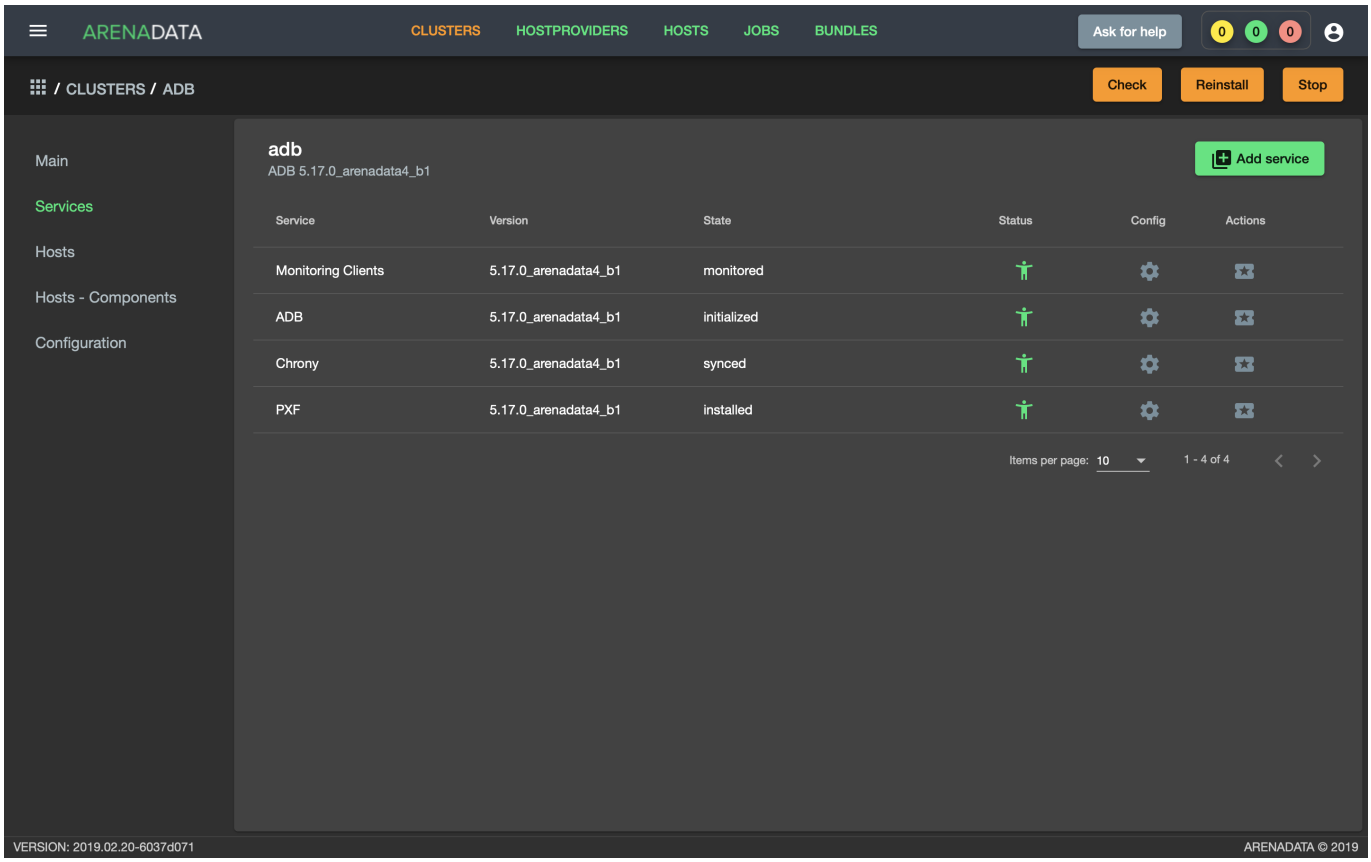


Рис.5.1.: Запуск проверки состояния кластера

2. Подтвердить действие в открывшемся диалоговом окне (Рис.5.2).

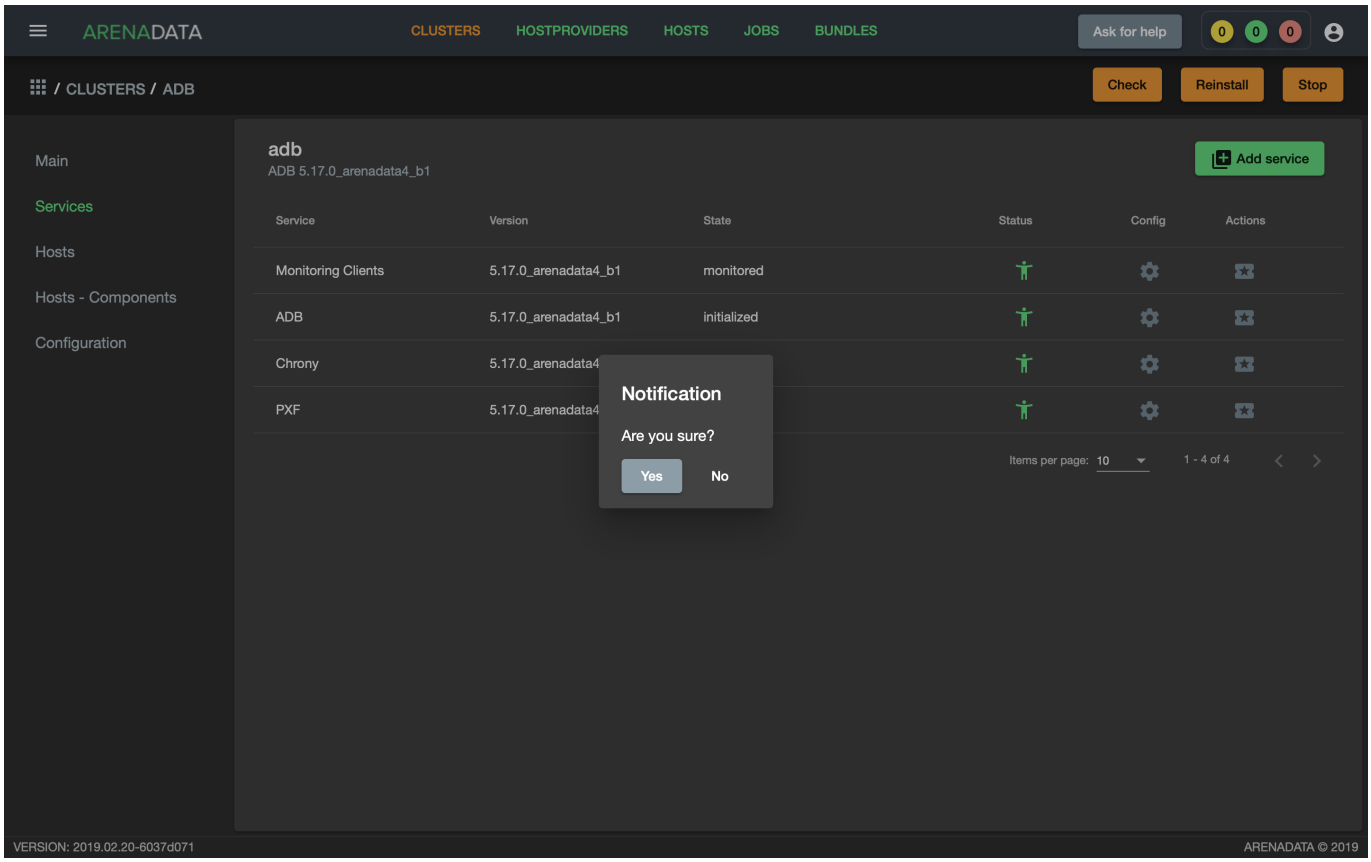


Рис.5.2.: Запрос на подтверждение действия

3. Открыть вкладку “JOBS” (Рис.5.3).
4. Выбрать последнее действие над кластером *adb* и в открывшемся окне проверить результаты (Рис.5.4).

5.2 Запуск и остановка ADB

Существует возможность запуска и остановки кластера **ADB** не только посредством консоли, но и через **ADCM**.

Для остановки кластера необходимо:

1. Перейти на любую вкладку кластера adb (в примере “Services”) и нажать кнопку “Stop” на верхней панели (Рис.5.5).
2. Выбрать режим остановки ADB.

В режиме *smart* остановка производится только в том случае, если отсутствуют клиентские соединения к базе данных, иначе процесс завершается с ошибкой. Чтобы прервать выполняющиеся транзакции, закрыть открытые соединения и принудительно остановить работу кластера необходимо использлвать режим *fast*. В режиме *immediate* принудительно завершаются процессы postgres, не давая корректно обработать транзакции кластеру. Этот режим не рекомендуется, так как в некоторых случаях может привести к повреждению базы данных. Для подтверждения действия необходимо нажать кнопку *Run* (Рис.5.6).

3. В результате успешной остановки кластера после обновления страницы пиктограммы статусов сервисов меняют цвет на оранжевый (Рис.5.7), а кластер меняет состояние с *running* на *stopped* (Рис.5.8).

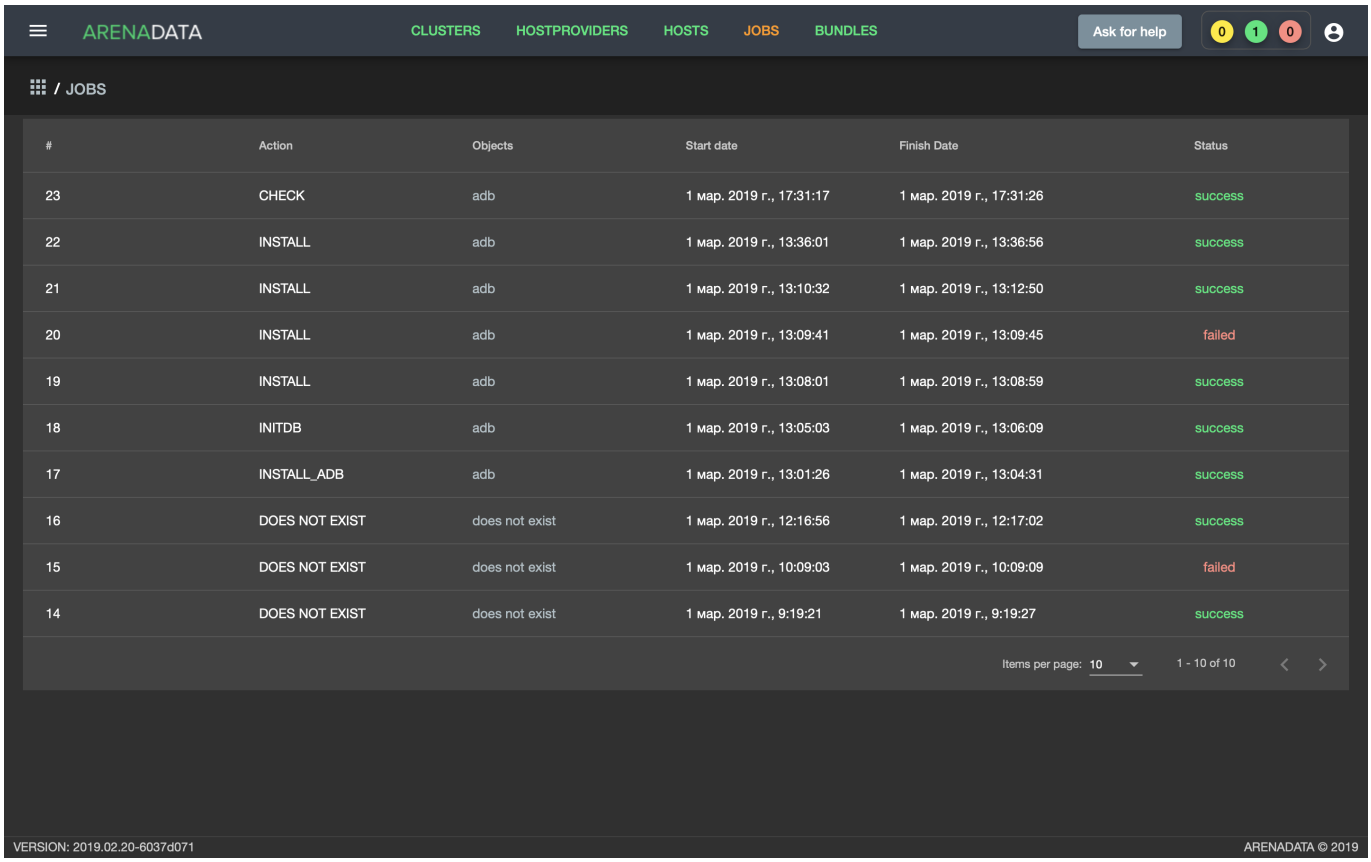


Рис.5.3.: Вкладка “JOBS”

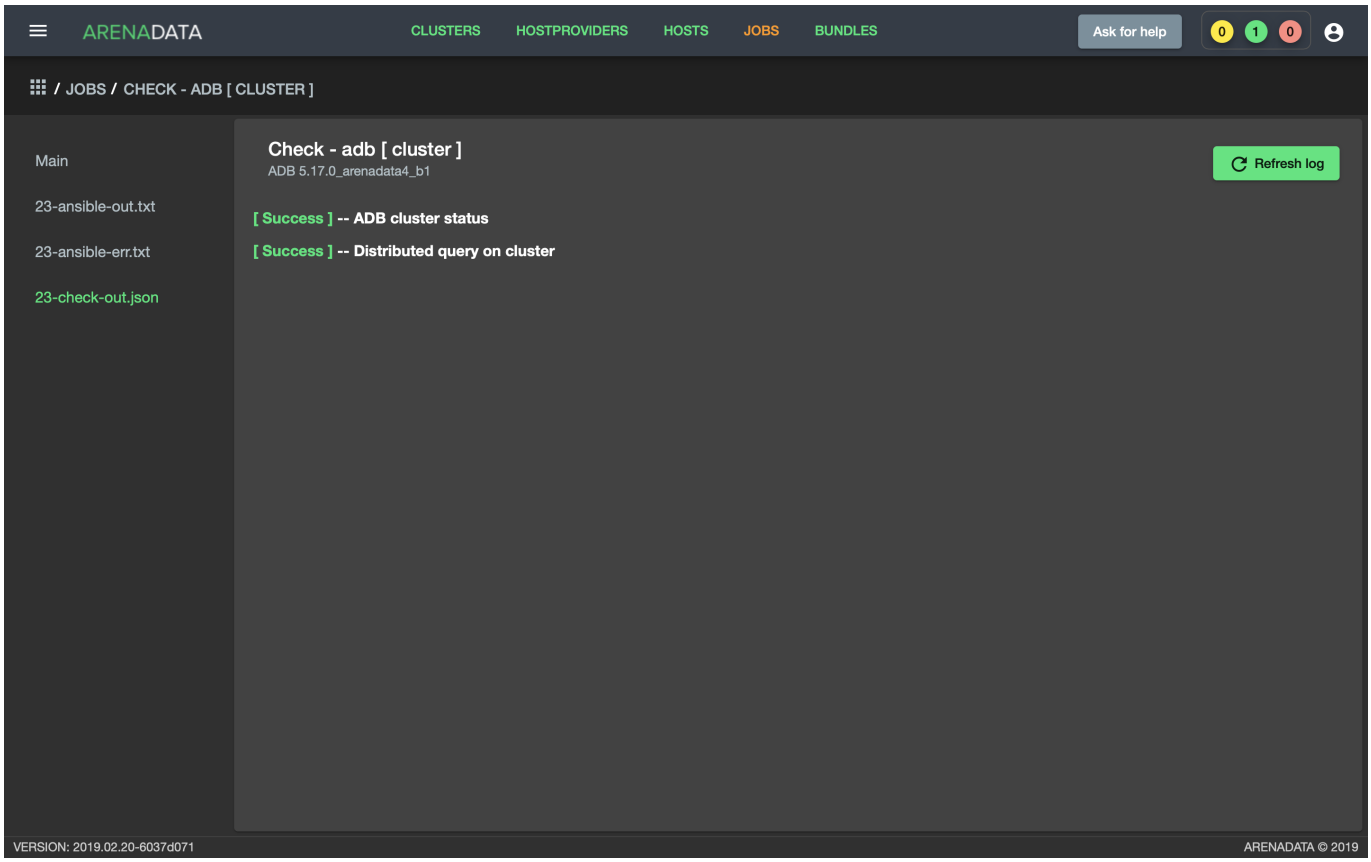


Рис.5.4.: Проверка состояния кластера

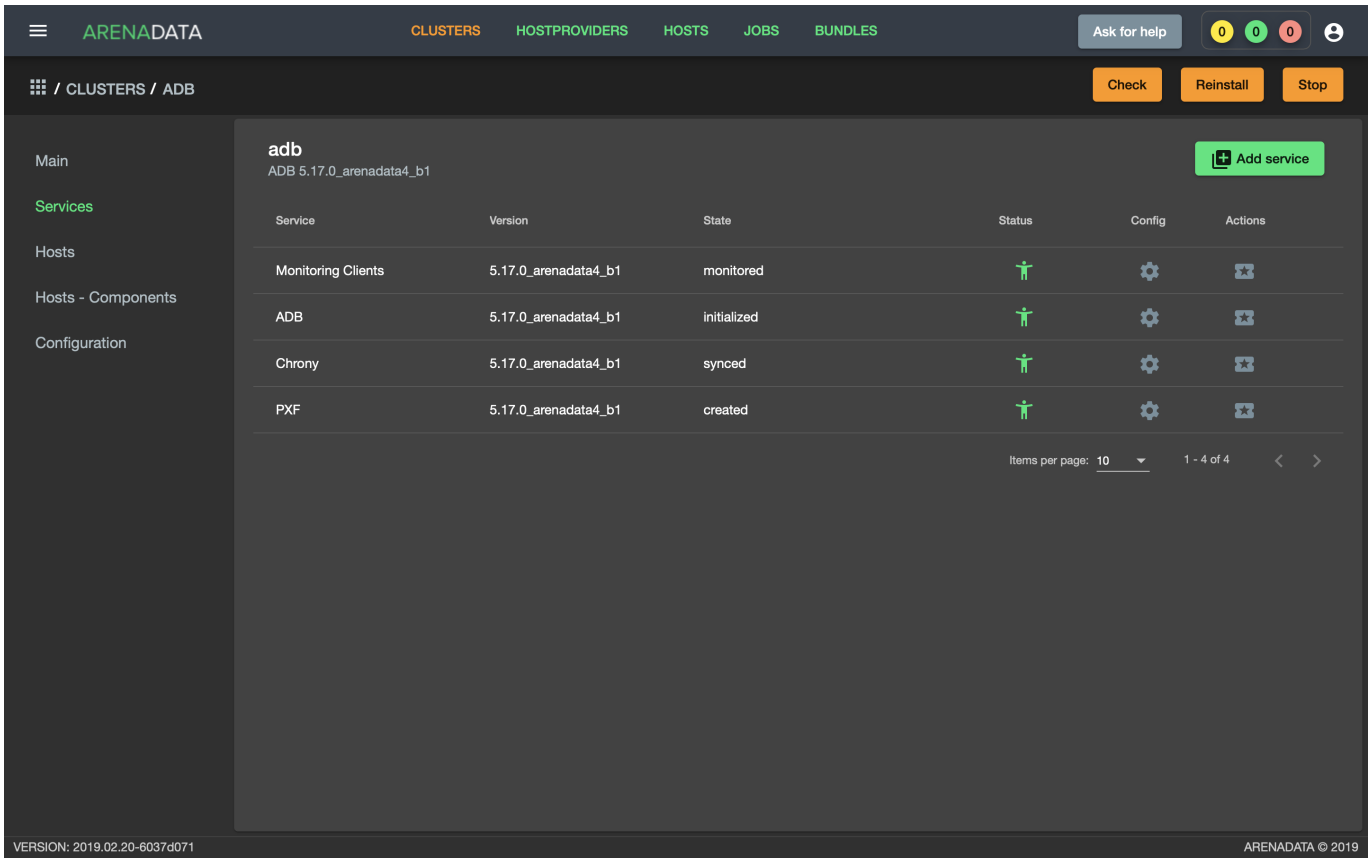


Рис.5.5.: Остановка кластера

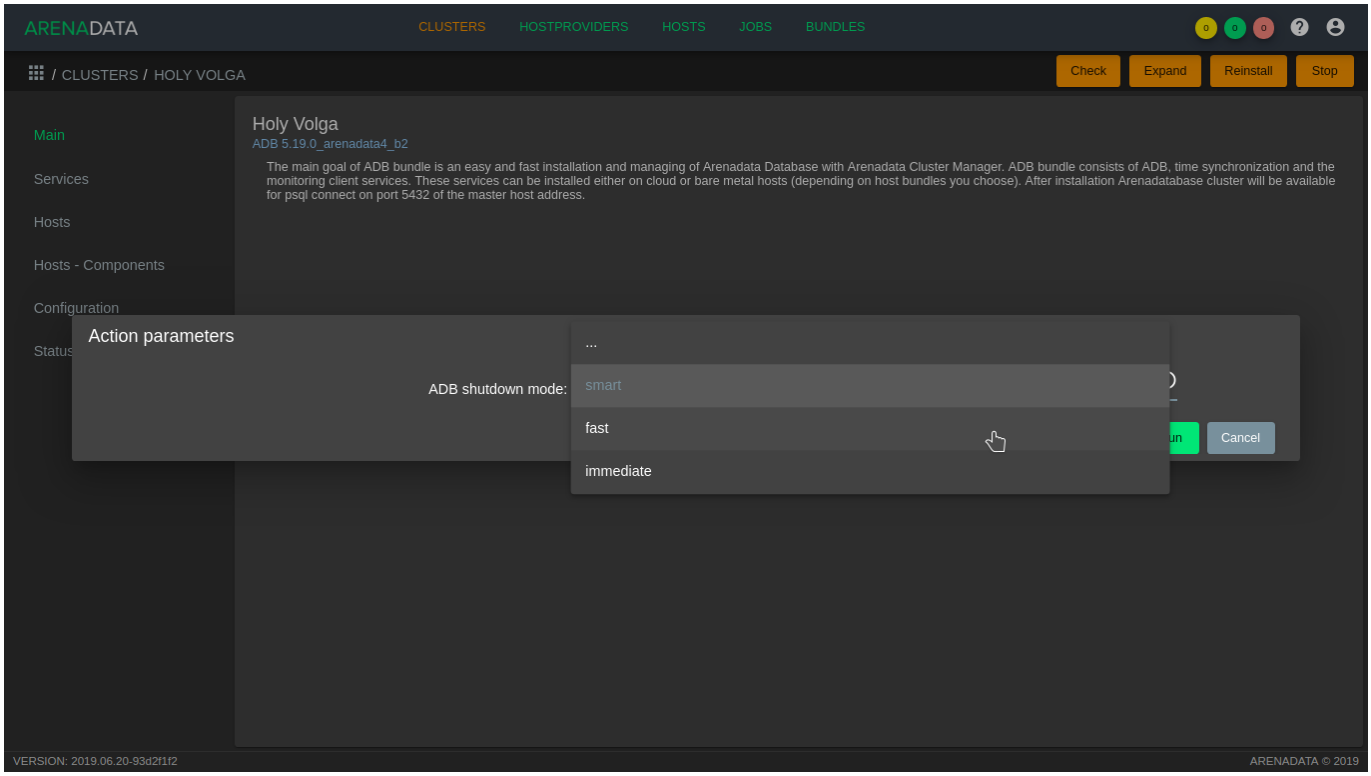


Рис.5.6.: Выбор режима остановки кластера

Important: В результате того или иного действия состояния сервисов не меняются на *stopped*, а сохраняют то же значение, что и после инсталляции. Это связано с опасениями инвалидации состояния в случаях, когда кластер *ADB* управляется через консоль в обход *ADCM*

Для запуска кластера необходимо:

1. Перейти на любую вкладку кластера adb и нажать кнопку “Start” на верхней панели.
2. Подтвердить действие в открывшемся диалоговом окне.
3. В результате успешного запуска кластера после обновления страницы пиктограммы статусов сервисов меняют цвет на зеленый, кластер меняет состояние с *stopped* на *running*.

5.3 Управление файловыми пространствами

Доступно с версии 5.19.0_arenadata4_b1

Для управления файловыми пространствами средствами ADCM может быть использовано действие “Manage filespace”. Действие доступно для проинициализированного кластера ADB:

- В выпадающем списке действий на странице со списком сервисов “Services” для сервиса *ADB*;
- В виде кнопки в верхней панели на страницы сервиса *ADB*.

Действие вызывает диалоговое окно управления файловым пространством (Рис.5.9).

В диалоговом окне доступны следующие поля:

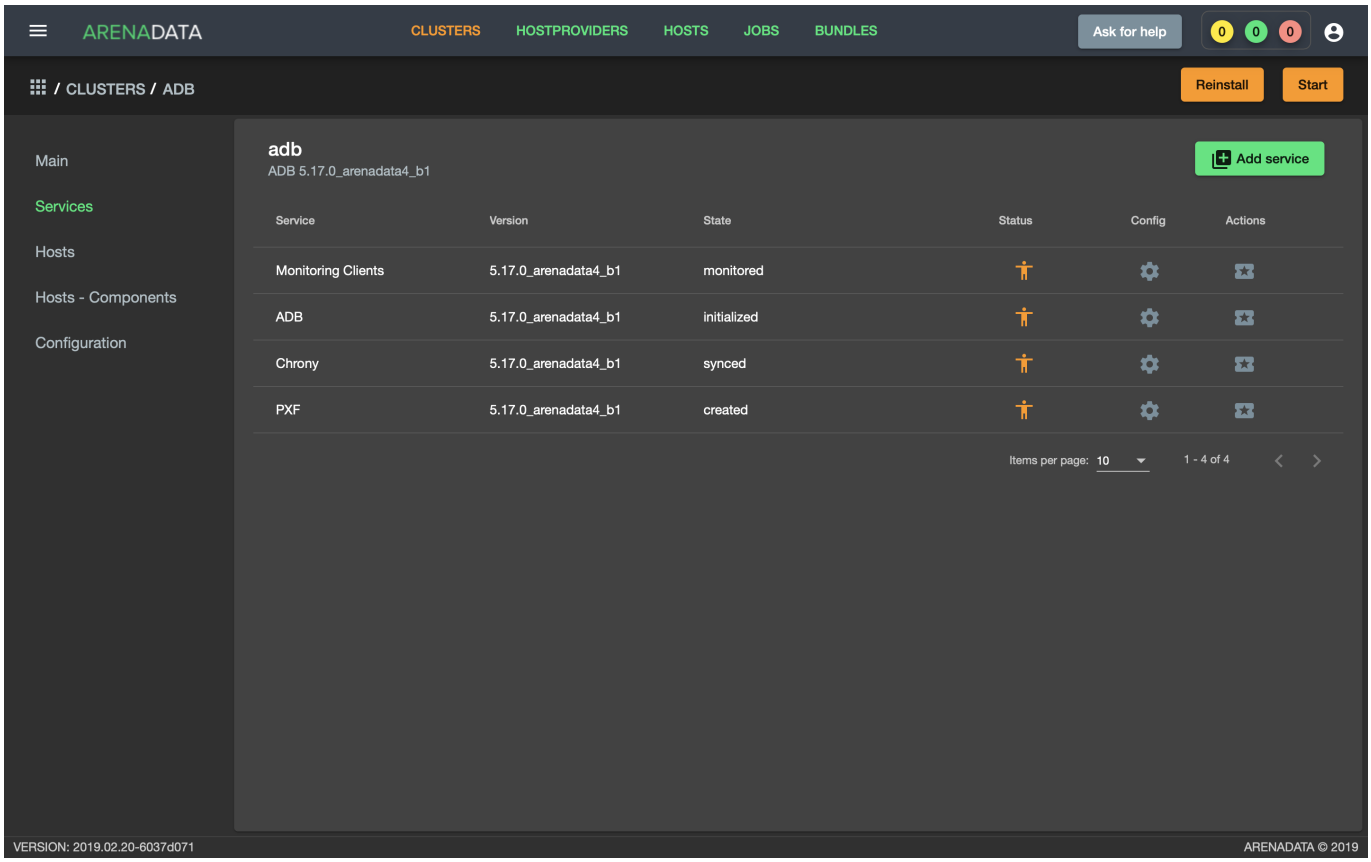


Рис.5.7.: Статусы сервисов

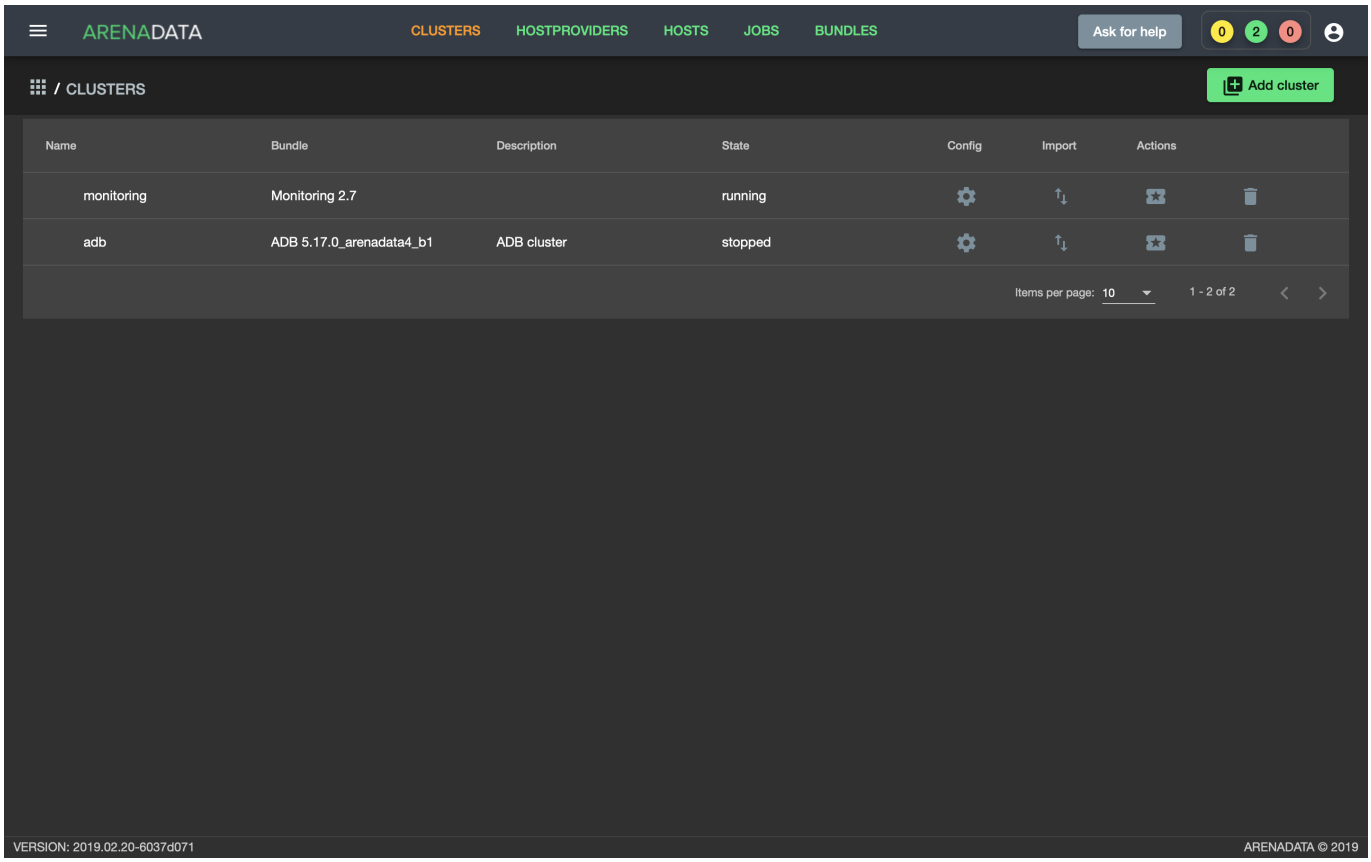


Рис.5.8.: Состояние кластера

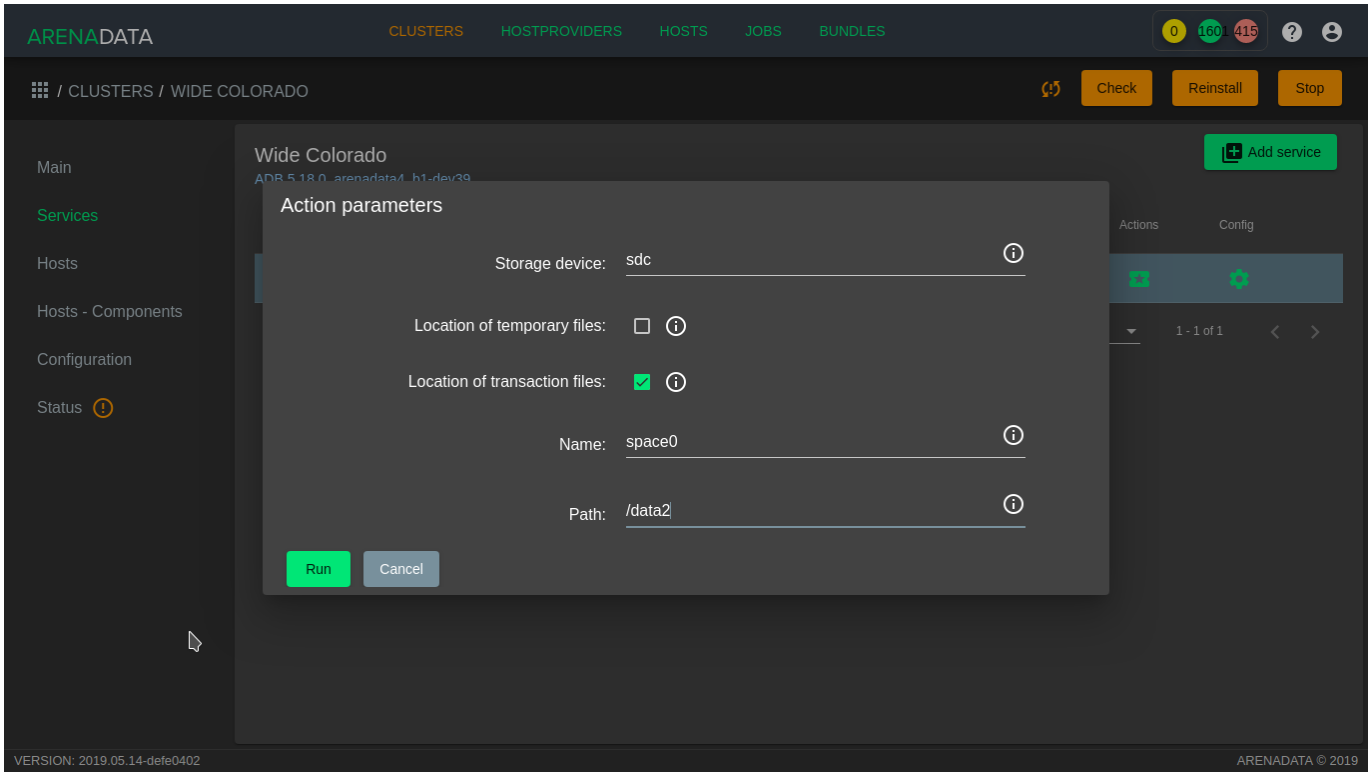


Рис.5.9.: Диалоговое окно управления файловым пространством

- **Name** – имя файлового пространства. Может быть указано имя уже существующего файлового пространства для совершения операций над ним;
- **Path** – путь, по которому должно быть расположено файловое пространство. Если не указано устройство, на котором должно быть расположено файловое пространство, путь должен существовать на всех хостах кластера и быть доступным на запись системному пользователю ADB;
- **Storage device** – имя устройства, которое необходимо использовать для создания файлового пространства (например, sdc). Для использования этой опции имя устройства должно совпадать на всех серверах кластера. На устройстве в процессе инициализации создается файловая система XFS, устройство монтируется в указанный в предыдущем параметре путь;
- **Location of temporary files** – переместить временные файлы в указанное файловое пространство;
- **Location of transaction files** – переместить файлы транзакций в указанное файловое пространство.

Important: Опции перемещения временных файлов и файлов транзакций в другое файловое пространство требуют остановки кластера. Активные соединения пользователей прерываются

5.4 Установа и удаление gpperfmon

Перед установкой кластера доступна возможность управления фактом установки на нем системы мониторинга gpperfmon при помощи соответствующего флажка в конфигурации сервиса *ADB*. После установки данный флажок недоступен для изменения, однако для сервиса *ADB* доступны два действия:

- Install GPperfmon – устанавливает необходимые пакеты на мастер и сегментные серверы, создает пользователя gpmon и предоставляет ему доступ к кластеру, генерирует конфигурационный файл и

инициализирует gpperfmon.

- Uninstall GPperfmon – выполняет удаление gpperfmon из кластера, включая пользователя gpmon, базу данных gpperfmon и файлы журналов, расположенные в дата-каталогах в каталоге gpperfmon. Также удаляется соответствующий пакет из системы.

5.5 Расширение кластера без ADCM

Увеличить производительность и емкость хранилища можно, развернув кластер **Arenadata DB**, добавив новые сегмент-сервера в массив.

Хранилища данных обычно растут по мере сбора дополнительных данных и увеличения сроков хранения существующих. Время от времени необходимо увеличивать емкость кластера для консолидации разных хранилищ в одной базе данных. Также может потребоваться дополнительная вычислительная мощность процессора для размещения недавно добавленных проектов аналитики. Хотя было бы разумно обеспечить потенциал для роста при изначальном определении системы, но, как правило, невозможно инвестировать в ресурсы задолго до того, как они потребуются. Поэтому следует периодически выполнять расширение базы данных.

При добавлении ресурсов в **ADB** по причине MPP-архитектуры ее емкость и производительность такие, как если бы система была первоначально реализована с добавленными ресурсами. В отличие от хранилищ данных, требующих значительного простоя для сброса и восстановления данных, в **ADB** этот показатель минимальный за счет поэтапного процесса расширения. Регулярные и специальные рабочие нагрузки могут продолжаться в ходе перераспределения данных с сохранением последовательности транзакций. Администратор может назначить активность распределения в соответствии с текущими операциями и при необходимости приостановить и возобновить его. А таблицы можно ранжировать таким образом, чтобы наборы данных перераспределялись в приоритетном порядке для скорейшего использования полученного расширенного объема критическими рабочими нагрузками или для освобождения необходимого дискового пространства для перераспределения очень больших таблиц.

В процессе расширения используются стандартные операции базы данных **ADB**. Зеркалирование сегментов и все механизмы репликации остаются активными, поэтому отказоустойчивость бескомпромиссна, а меры аварийного восстановления эффективны.

- *Планирование расширения кластера*
- *Подготовка и добавление узлов*
- *Добавление новых сегмент-серверов*
- *Перераспределение таблиц*
- *Удаление временной схемы расширения*

При расширении базы данных **ADB** следует ожидать следующие особенности:

- Масштабируемость и производительность – при добавлении ресурсов емкость и производительность кластера такие, как если бы система была первоначально реализована с добавленными ресурсами;
- Бесперебойная работа при расширении – регулярные рабочие нагрузки, как запланированные, так и специальные, не прерываются. Для инициализации новых сегмент-серверов требуется короткий запланированный период времени простоя, аналогичный необходимому для перезапуска системы. Продолжительность простоя не связана с размером кластера до или после расширения;
- Согласованность транзакций;
- Отказоустойчивость – во время расширения стандартные механизмы отказоустойчивости, такие как зеркальное отображение сегментов, остаются активными, последовательными и эффективными;

- Репликация и аварийное восстановление – все существующие механизмы репликации продолжают функционировать во время расширения. Необходимые в случае сбоя или аварии механизмы восстановления остаются эффективными;
- Прозрачность процесса – в процессе расширения используются стандартные механизмы, поэтому администраторы могут диагностировать и устранять любые проблемы;
- Настраиваемый процесс – расширение может быть длительным процессом, но его можно вписать в график текущих операций. Таблицы схемы расширения позволяют администраторам устанавливать приоритет порядка перераспределения таблиц, а активность расширения может быть приостановлена и возобновлена.

Планирование и физические аспекты проекта по расширению кластера составляют большую часть работы, чем само расширение. Для планирования и выполнения требуется многопрофильная команда – для локальных установок необходимо получить и подготовить пространство для новых сегмент-серверов; сервера должны быть выбраны, приобретены, установлены, подключены, настроены и протестированы. Для облачных развертываний должны быть сделаны аналогичные работы.

После подготовки новых платформ и конфигурации их сетей необходимо настроить операционные системы и выполнить тесты на производительность с помощью утилит **ADB**. ПО кластера включает утилиты, которые полезны для тестирования и записи новых сегмент-серверов перед началом расширения.

Сразу после установки и тестирования новых сегмент-серверов начинается программная фаза процесса расширения, спроектированная минимально разрушительной, транзакционно последовательной, надежной и гибкой.

Во время инициализации хостов новых сегмент-серверов и подготовки системы к процессу расширения есть короткий интервал простоя. Это время может быть запланировано на период низкой активности во избежание нарушения текущих бизнес-операций. В течение инициализации выполняются задачи:

- Устанавливается ПО;
- Создается база данных с объектами на хостах нового сегмент-сервера;
- Создается схема расширения в базе данных master для управления процессом расширения;
- Изменяется политика распределения на *DISTRIBUTED RANDOMLY* для каждой таблицы.

Затем система перезапускается, и приложения возобновляются:

- Добавленные сегмент-сервера сразу становятся доступными и участвуют в новых запросах и загрузке данных. Однако существующие данные искажаются, так как они сконцентрированы на исходных сегментах и должны быть перераспределены по их новому общему числу;
- По причине того, что таблицы теперь имеют рандомную политику распределения, оптимизатор создает планы запросов, которые не зависят от ключей распределения. Некоторые запросы становятся менее эффективными, поскольку необходимы дополнительные операторы данных.

Таблицы и партиции перераспределяются, используя таблицы контроля расширения (expansion control) в качестве инструкции, при этом:

- Для изменения политики распределения обратно к исходной используется оператор *ALTER TABLE* с параметром *REORGANIZE=TRUE*, в результате данные распределяются по всем серверам, старым и новым, в соответствии с первоначальной политикой;
- Статус таблиц обновляется в таблицах expansion control;
- Оптимизатор запросов создает более эффективные планы выполнения, опираясь на ключи распределения.

Расширение кластера **ADB** завершается по окончанию перераспределения всех таблиц.

Перераспределение данных – это длительный процесс, создающий большую сетевую и дисковую активность. Перераспределение некоторых очень больших баз данных может занять несколько дней. С целью свести к минимуму последствия повышенной активности для бизнес-операций системные администраторы могут

приостанавливать и возобновлять деятельность по расширению кластера на разовой основе или в соответствии с заранее заданным графиком. Наборам данных можно установить приоритетность для того, чтобы критически важные приложения в первую очередь получили пользу от расширения.

За весь процесс расширения утилита **gpexpand** запускается четыре раза с различными параметрами:

1. При создании настроечного файла:

   ```
   gpexpand -f hosts_file
   ```

2. При инициализации сегмент-серверов и создании схемы расширения:

   ```
   gpexpand -i input_file -D database_name
   ```

Утилита **gpexpand** создает каталог данных, копирует пользовательские таблицы из всех существующих баз данных в новые сегмент-сервера и записывает метаданные для каждой таблицы в схему расширения для отслеживания статуса. По завершении процесса операция расширения фиксируется и является необратимой.

3. При перераспределении таблиц:

   ```
   gpexpand -d duration
   ```

При инициализации **gpexpand** аннулирует политики распределения хеша в таблицах во всех существующих базах данных, за исключением родительских в секционированной таблице, и устанавливает рандомную политику распределения для всех таблиц.

Для завершения процесса расширения системы необходимо запустить утилиту **gpexpand** для перераспределения таблиц данных по добавленным сегмент-серверам. В зависимости от размера и масштаба системы перераспределение может быть выполнено за одну сессию или его можно разделить по этапам, но в течение более продолжительного периода. В процессе перераспределения таблицы и партиции недоступны для операций чтения и записи. Поскольку каждая таблица перераспределяется по новым сегмент-серверам, производительность базы данных постепенно улучшается и в итоге превышает тот уровень, который был до расширения.

В крупномасштабных системах, требующих нескольких сеансов перераспределения, может потребоваться неоднократный запуск **gpexpand**. Утилита может быть полезна при явном ранжировании перераспределенной таблицы.

По завершению инициализации и возвращению системы в оперативный режим пользователи получают доступ к базе данных, но возможно ухудшение производительности систем, которые в значительной степени зависят от хеш-распределения таблиц. Обычные операции, такие как задания ETL, пользовательские запросы и отчеты, продолжаются, хотя пользователи могут отмечать более медленное время отклика.

Когда таблица имеет рандомную политику распределения, база данных **ADB** не может применять уникальные ограничения (например, PRIMARY KEY), поскольку повторяющиеся строки не выдают ошибку “constraint violation”. Это может повлиять на ETL и процессы загрузки до тех пор, пока не завершится перераспределение таблицы.

4. При удалении схемы расширения:

   ```
   gpexpand -c
   ```

5.5.1 Планирование расширения кластера

Тщательное планирование обеспечивает успех проекта по расширению кластера **ADB**. Темы данного раздела помогают убедиться в готовности к процессу расширения системы:

- *Контрольный список расширения*
- *Планирование нового аппаратного оборудования*
- *Планирование инициализации нового сегмент-сервера*
- *Планирование таблицы перераспределения*

Контрольный список расширения

Контрольный список расширения – это список, включающий задачи для подготовки и выполнения процесса расширения базы данных **ADB**.

Онлайн-задачи перед процессом расширения (система работает и доступна):

- Разработать и выполнить план заказа, создания и подключения новых аппаратных платформ или подготовить облачные ресурсы;
- Разработать план расширения базы данных. Сопоставить количество сегментов на одном хосте, запланировать период простоя для тестирования производительности и создания схемы расширения, назначить интервалы для перераспределения таблиц;
- Выполнить полный сброс схемы;
- Установить двоичные файлы базы данных на новые хосты;
- Скопировать SSH-ключи на новые хосты (**gpssh-exkeys**);
- Проверить работоспособность операционной системы нового оборудования или облачных ресурсов (**gpcheck**);
- Проверить дисковый ввод-вывод и пропускную способность памяти нового оборудования или облачных ресурсов (**gpcheckperf**);
- Проверить, что в каталоге master data нет чрезвычайно больших файлов в директориях *pg_log* и *gpperfmon/data*;
- Проверить, что нет проблем с каталогом (**gpcheckcat**);
- Подготовить настроечный файл для расширения (**gpexpand**).

Офлайн-задачи расширения (система заблокирована и недоступна для всех действий пользователя во время данного процесса):

- Проверить среду операционной системы для объединения существующих и новых аппаратных платформ или облачных ресурсов (**gpcheck**);
- Проверить дисковый ввод-вывод и пропускную способность памяти для объединения существующих и новых аппаратных платформ или облачных ресурсов (**gpcheckperf**);
- Инициализировать новые сегмент-сервера в массиве и создать схему расширения (**gpexpand-i** *input_file*).

Онлайн-расширение и перераспределение таблиц (система работает и доступна):

- Остановить любые автоматизированные процессы создания снапшотов или другие процессы, занимающие дисковое пространство, перед началом перераспределения таблиц;
- Перераспределить таблицы с помощью расширенной системы (**gpexpand**);
- Удалить схему расширения (**gpexpand-c**);
- Запустить **analyze** для обновления статистики распределения. Во время расширения использовать **gpexpand -a**, после расширения – **analyze**.

Планирование нового аппаратного оборудования

Продуманный и тщательный подход к разворачиванию совместимого оборудования значительно сокращает возможные риски процесса расширения.

Аппаратные ресурсы и конфигурации для новых сегмент-серверов должны соответствовать ресурсам существующих хостов. Шаги по планированию и настройке новых аппаратных платформ различаются для каждого разворачивания. Некоторые общие рекомендации:

- Подготовить физическое пространство для нового оборудования, учесть охлаждение, источник питания и другие физические факторы;
- Определить физическую сеть и кабели для подключения нового и существующего оборудования;
- Сопоставить существующее пространство IP-адресов и разработать сетевой план для расширенной системы;
- Узнать конфигурацию системы (пользователи, профили, сетевые карты и т.д.) по существующему оборудованию для использования в качестве подробного списка для нового оборудования;
- Создать план кастомной сборки для разворачивания аппаратного обеспечения с требуемой конфигурацией на конкретном месте и в конкретной среде.

После выбора и добавления нового аппаратного оборудования в сетевую среду убедиться, что успешно выполняются burn-in tasks.

Планирование инициализации нового сегмент-сервера

Расширение **ADB** требует ограниченного периода простоя системы. В течение этого времени необходимо запустить **gpexpand** для инициализации новых сегмент-серверов в массиве и создать схему расширения.

Период простоя системы зависит от количества объектов схемы в **ADB** и от других факторов, связанных с производительностью оборудования. В большинстве сред инициализация новых сегмент-серверов занимает менее тридцати минут в автономном режиме.

Important: После начала инициализации новых сегмент-серверов восстановление системы с помощью резервных файлов, созданных перед расширением, невозможно. При успешной инициализации расширение фиксируется и не может быть отменено

При наличии у массива зеркальных сегментов необходимо, чтобы новые сегмент-сервера имели сконфигурированное зеркалирование, иначе добавление зеркал к новым хостам с помощью утилиты **gpexpand** невозможно. Также необходимо убедиться, что добавлено достаточно новых хост-машин для размещения новых зеркальных сегментов. Количество новых хостов зависит от используемой стратегии зеркалирования:

- Spread Mirroring – добавить в массив хотя бы на один новый хост больше, чем количество сегментов на хост. Для обеспечения равномерного распределения количество отдельных хостов должно быть больше, чем количество инстансов сегмента на хост;
- Grouped Mirroring – добавить в массив не менее двух новых хостов, чтобы зеркала для первого хоста могли находиться на втором, а зеркала для второго хоста – на первом.

По умолчанию новые хосты инициализируются с таким же количеством основных сегментов, что и у существующих хостов. При этом есть возможность увеличить количество сегментов на хост или добавить новые сегмент-сервера к существующим хостам. Например, если существующие хосты в настоящее время имеют два сегмента на хост, можно использовать **gpexpand** для инициализации двух дополнительных сегментов на существующие хосты и в итоге получить четыре сегмента и по четыре новых сегмент-сервера на новых хостах.

Интерактивный процесс создания настроечного файла для расширения запрашивает данную опцию, директории новых сегмент-серверов можно вручную указать в файле конфигурации (*Создание настроечного файла для расширения кластера*).

При инициализации утилита **gpexpand** создает схему расширения. При этом если база данных не задана (**gpexpand -D**), то схема создается в БД, указанной в переменной среде *PGDATABASE*. Схема расширения хранит метаданные для каждой таблицы в системе, поэтому ее статус можно отслеживать на протяжении всего процесса. Схема состоит из двух таблиц и представления для отслеживания хода выполнения операции расширения:

- *gpexpand.status*

- *gpexpand.status_ detail*
- *gpexpand.expansion_ progress*

Процессом расширения можно управлять в *gpexpand.status_ detail*. Например, удаление записи из этой таблицы не позволяет системе развернуть таблицу по новым сегмент-серверам. Также можно задать порядок перераспределения таблиц путем назначения им ранга.

Планирование таблицы перераспределения

Перераспределение таблицы выполняется во время работы системы, и во многих случаях оно осуществляется за одну сессию **gpexpand** в период низкого использования. Но для крупных систем может потребоваться несколько сеансов работы утилиты и настройка порядка перераспределения таблицы для минимизации влияния на производительность.

Important: Для перераспределения таблиц на хостах сегмента должно быть достаточно места на диске для временного хранения копии самой большой таблицы. Все таблицы недоступны для операций чтения и записи во время перераспределения

Эффективность перераспределения таблицы зависит от ее размера, типа хранилища и структуры партицирования. Перераспределение любой таблицы с помощью **gpexpand** занимает столько же времени, сколько и операция *CREATE TABLE AS SELECT*. В случае перераспределения таблицы фактов терабайтного масштаба утилита расширения может использовать большую часть доступных системных ресурсов, что может повлиять на производительность запросов или другие рабочие нагрузки базы данных.

При наличии большого объема свободного места на диске есть возможность как можно скорее сосредоточиться на восстановлении оптимальной производительности запроса, перераспределив сначала самые важные таблицы. Для этого необходимо им назначить наивысший рейтинг и запланировать операции перераспределения на период низкого использования системы. После чего запустить процесс перераспределения таблицы.

Если существующие узлы имеют ограниченное дисковое пространство, то следует сначала перераспределить меньшие таблицы (например, таблицы измерений) с целью освобождения места на диске для хранения копии самой большой таблицы. Доступное дисковое пространство на исходных сегментах увеличивается по мере перераспределения каждой таблицы по расширенному массиву.

Утилита **gpexpand** перераспределяет таблицы типа append-optimized и compressed append-optimized с отличной скоростью от перераспределения таблицы heap. Необходимый для сжатия и распаковки данных объем процессора имеет тенденцию увеличивать влияние на производительность системы. Поэтому для таблиц аналогичных по размеру и данным можно найти общие различия, например:

- Несжатые таблицы типа append-optimized расширяются на *10%* быстрее, чем heap таблицы.
- zlib-сжатые таблицы типа append-optimized расширяются значительно медленнее, чем аналогичные несжатые (примерно на *80%*).
- Системы с сжатием данных, такие как **ZFS**/**LZJB**, требуют больше времени для перераспределения.

В период времени между инициализацией новых сегмент-серверов и успешным перераспределением таблицы ограничения первичного ключа не могут быть применены. Повторяющиеся данные, добавленные в таблицы в течение этого периода, не позволяют утилите расширения перераспределить затронутые таблицы. После перераспределения таблицы ограничение первичного ключа снова применяется надлежащим образом. В случае если процесс расширения нарушает ограничения, утилита регистрирует ошибки и отображает предупреждения. Для исправления нарушений ограничений следует выполнить одно из следующих действий:

- Очистить повторяющиеся данные в столбцах первичного ключа и повторно запустить **gpexpand**;
- Удалить ограничения первичного ключа и повторно запустить **gpexpand**.

Для перераспределения таблицы, содержащей столбцы пользовательских типов данных на удаление, необходимо сначала заново создать таблицу, используя *CREATE TABLE AS SELECT*. А после того, как процесс исключит столбцы, перераспределить таблицу с помощью **gpexpand**.

Поскольку утилита расширения может обрабатывать каждую отдельную партицию в большой таблице, эффективная конструкция партиций снижает влияние перераспределения таблиц на производительность. Политика случайного распределения применяется только к дочерним таблицам партицированной таблицы, и блокировка операций чтения и записи единовременно применяется для перераспределения только к одной дочерней таблице.

Системы с интенсивным индексированием имеют значительно более медленные темпы перераспределения таблиц, так как утилита **gpexpand** повторно индексирует каждую таблицу после ее перераспределения.

5.5.2 Подготовка и добавление узлов

Важно убедиться, что новые узлы готовы к интеграции в существующую систему **ADB**.

Для подготовки новых системных узлов к расширению следует установить двоичные файлы программного обеспечения базы данных **ADB**, заменить необходимые SSH-ключи и выполнить тесты на производительность.

Тесты необходимо выполнять сначала только на новых узлах, а затем на всех. Тесты на всех узлах важно проводить с системой в автономном режиме, чтобы активность пользователя не искажала результаты. Как правило, тесты на производительность следует выполнять, когда администратор изменяет сеть узлов или другие особые условия в системе. Например, если система работает на двух сетевых кластерах, тесты выполняются на каждом из них.

Обмен ключами SSH

Новые узлы должны обмениваться ключами SSH с существующими узлами, чтобы административные утилиты **ADB** могли подключаться ко всем сегментам без запроса пароля. Процесс обмена ключами необходимо выполнить дважды – сначала от *root* (для удобства администрирования), а затем в качестве пользователя *gpadmin* (для служебных программ управления). Следует выполнить следующие задачи по порядку:

- Обмен ключами SSH от root;
- Создание пользователя gpadmin;
- Обмен ключами SSH в качестве пользователя gpadmin.

Для обмена ключами SSH от пользователя *root* необходимо выполнить следующие действия:

1. Создать файл *host file* с существующими именами хостов в массиве и отдельный файл с именами новых хостов. Для существующих узлов можно использовать файл настройки SSH-ключей в системе. В файлах должны быть перечислены все хосты (мастер, резервный мастер и сегмент-сервер) с одним именем в строке и без дополнительных строк или пробелов. Обмен SSH-ключами при конфигурации с несколькими NIC использует настроенные имена хостов. В примере *mdw* сконфигурирован с одним NIC, а *sdw1*, *sdw2* и *sdw3* сконфигурированы с четырьмя сетевыми адаптерами:

```
mdw
sdw1-1
sdw1-2
sdw1-3
sdw1-4
sdw2-1
sdw2-2
sdw2-3
sdw2-4
sdw3-1
sdw3-2
```

```
sdw3-3
sdw3-4
```

2. Войти в систему как *root* на master-хосте и отправить файл *greenplum_path.sh* из инсталяции **ADB**:

```
$ su -
# source /usr/local/greenplum-db/greenplum_path.sh
```

3. Запустить утилиту **gpssh-exkeys**, ссылающуюся на файл списка хостов. Например:

```
# gpssh-exkeys -e /home/gpadmin/existing_hosts_file -x
/home/gpadmin/new_hosts_file
```

4. Утилита **gpssh-exkeys** проверяет удаленные хосты и выполняет обмен ключами между всеми узлами. При появлении запроса ввести пароль пользователя *root*.

```
***Enter password for root@hostname: <root_password>
```

Для создания пользователя *gpadmin* необходимо выполнить следующие действия:

1. Применить утилиту **gpssh** для создания пользователя *gpadmin* на всех хостах нового сегмент-сервера, используя созданный для обмена ключами список новых хостов. Например:

```
# gpssh -f new_hosts_file '/usr/sbin/useradd gpadmin -d
/home/gpadmin -s /bin/bash'
```

2. Задать пароль для нового пользователя *gpadmin*. В **Linux** это можно сделать на всех сегментах одновременно с помощью утилиты **gpssh**. Например:

```
# gpssh -f new_hosts_file 'echo gpadmin_password | passwd
gpadmin --stdin'
```

3. Путем поиска домашней директории убедиться, что пользователь *gpadmin* создан:

```
# gpssh -f new_hosts_file ls -l /home
```

Для обмена ключами SSH в качестве пользователя *gpadmin* необходимо выполнить следующие действия:

1. Войти в систему как *gpadmin* и запустить утилиту **gpssh-exkeys**, ссылаясь на файл списка хостов:

```
# gpssh-exkeys -e /home/gpadmin/existing_hosts_file -x
/home/gpadmin/new_hosts_file
```

2. Утилита **gpssh-exkeys** проверяет удаленные хосты и выполняет обмен ключами между всеми узлами. При появлении запроса ввести пароль пользователя *gpadmin*.

```
***Enter password for gpadmin@hostname: <gpadmin_password>
```

Проверка настроек ОС

Утилита **gpcheck** проверяет, что все новые хосты имеют правильные настройки ОС для запуска программного обеспечения базы данных **ADB**. Для запуска утилиты необходимо:

1. Войти на master-хост в качестве пользователя, который в дальнейшем будет запускать систему **ADB** (например, *gpadmin*):

```
$ su - gpadmin
```

2. Запустить утилиту **gpcheck** с помощью файла хоста для новых узлов. Например:

```
$ gpcheck -f new_hosts_file
```

Проверка дискового ввода-вывода и пропускной способности памяти

Для проверки дискового ввода-вывода и пропускной способности памяти системы **ADB** используется утилита **gpcheckperf**:

1. Запустить утилиту **gpcheckperf**, используя файл хоста для новых узлов. Для указания файловых систем, которые следует протестировать на каждом хосте, применяется параметр *-d*. При этом должен быть доступ на запись к данным каталогам.

```
$ gpcheckperf -f new_hosts_file -d /data1 -d /data2 -v
```

2. Утилита может занять много времени для выполнения тестов, так как она копирует очень большие файлы между хостами. При завершении процесса отображаются итоговые результаты тестов *Disk Write*, *Disk Read* и *Stream*.

Для сети, разделенной на подсети, необходимо повторить процедуру с отдельным файлом хоста для каждой подсети.

5.5.3 Добавление новых сегмент-серверов

Для инициализации новых сегмент-серверов, создания схемы расширения и установки общесистемной политики случайного распределения используется утилита **gpexpand**.

При первом запуске утилиты **gpexpand** с настроечным файлом она создает схему расширения и устанавливает политику распределения для всех таблиц в *DISTRIBUTED RANDOMLY*. Затем при последующем запуске **gpexpand** определяет, была ли создана схема расширения и, если да, выполняет перераспределение таблиц.

- *Создание настроечного файла для расширения кластера*
- *Запуск gpexpand для инициализации новых сегмент-серверов*
- *Откат неудачной установки*

Создание настроечного файла для расширения кластера

Утилите **gpexpand** требуется настроечный файл с информацией о новых сегментах и хостах. При запуске **gpexpand** без указания настроечного файла утилита отображает интерактивное интервью, которое собирает необходимую информацию и автоматически создает настроечный файл. В таком случае можно указать файл со списком хостов расширения.

Создание настроечного файла в интерактивном режиме

Для создания настроечного файла в интерактивном режиме необходимо перед запуском **gpexpand** убедиться, что известно:

- Количество новых хостов (или файл *hosts*);
- Новые имена хостов (или файл *hosts*);
- Стратегия зеркалирования, используемая в существующих хостах (если имеется);
- Количество сегмент-серверов для добавления на хост (если имеются).

Утилита автоматически создает настроечный файл на основе указанной информации, *dbid*, *content ID* и каталога данных, хранящихся в *gp_segment_configuration*, и сохраняет файл в текущем каталоге.

Создание настроечного файла в интерактивном режиме:

1. Выполнить вход на мастер хост базы данных пользователем, который будет запускать **ADB**, например, *gpadmin*.
2. Запустить **gpexpand**. Утилита отображает сообщение о том, как подготовиться к операции расширения, и предлагает выйти или продолжить. При необходимости указать файл *hosts*, содержащий перечень новых хостов, с помощью опции *-f*. Например:

```
$ gpexpand -f /home/gpadmin/new_hosts_file
```

3. Для продолжения в командной строке выбрать *Y*.
4. Если на втором шаге файл *hosts* не был указан, то необходимо ввести список имен хостов новых расширений, разделяя их запятыми и не включая имена дополнительных хостов. Например:

```
> sdw4, sdw5, sdw6, sdw7
```

   Для добавления логических сегментов только на существующих серверах (без добавления новых серверов) ввести пустую строку в запрос, не указывая *localhost* и имени действующего узла.

5. Ввести стратегию зеркалирования, используемую в кластере (если имеется), с параметром *spread*, *grouped* или *none*. Значение по умолчанию *spread*. И убедиться, что имеется достаточно хостов для выбранной стратегии.
6. Ввести количество добавляемых основных сегментов (если они есть). По умолчанию новые хосты инициализируются с таким же количеством основных сегментов, что и существующие хосты. Можно увеличить количество сегментов на каждом хосте. Указанное число будет количеством дополнительных сегментов, инициализированных на всех хостах. Например, если существующие хосты в настоящее время имеют по два сегмента каждый, ввод значения *2* инициализирует еще два сегмента на существующих хостах и четыре сегмента на новых хостах.
7. При добавлении новых основных сегментов ввести новый корневой каталог первичных данных для новых сегментов. Не указывать фактическое имя каталога данных, оно создается автоматически с помощью **gpexpand** на основе существующих имен каталога данных. Например, если существующие каталоги данных выглядят следующим образом:

```
/gpdata/primary/gp0
/gpdata/primary/gp1
```

   тогда, чтобы указать каталоги данных для двух новых основных сегментов, ввести следующее (по одному в каждом запросе):

```
/gpdata/primary
/gpdata/primary
```

   При запуске инициализации утилита создает новые каталоги *gp2* и *gp3* в каталоге */gpdata/primary*.

8. При добавлении новых зеркальных сегментов ввести новый корневой каталог зеркальных данных для новых сегментов. Не указывать имя каталога данных, он создается автоматически с помощью **gpexpand** на основе существующих имен каталога данных. Например, если существующие каталоги данных выглядят следующим образом:

```
/gpdata/mirror/gp0
/gpdata/mirror/gp1
```

   тогда, чтобы указать каталоги данных для двух новых зеркальных сегментов, ввести следующее (по одному в каждом запросе):

```
/gpdata/mirror
/gpdata/mirror
```

При запуске инициализации утилита создает новые каталоги *gp2* и *gp3* в каталоге */gpdata/mirror*.

Основные и зеркальные каталоги для новых сегмент-серверов должны существовать на хостах, а пользователь, запускающий **gpexpand**, должен иметь разрешения на создание каталогов в них.

После ввода всей необходимой информации утилита генерирует настроечный файл и сохраняет его в текущем каталоге. Например:

```
gpexpand_inputfile_yyyymmdd_145134
```

Формат настроечного файла расширения

Следует использовать интерактивный режим для создания собственного настроечного файла, если сценарий расширения не имеет нетипичных потребностей.

Формат настроечных файлов расширения:

```
hostname:address:port:fselocation:dbid:content:preferred_role:replication_port
```

Например:

```
sdw5:sdw5-1:50011:/gpdata/primary/gp9:11:9:p:53011
sdw5:sdw5-2:50012:/gpdata/primary/gp10:12:10:p:53011
sdw5:sdw5-2:60011:/gpdata/mirror/gp9:13:9:m:63011
sdw5:sdw5-1:60012:/gpdata/mirror/gp10:14:10:m:63011
```

Для каждого нового сегмент-сервера требуется формат настроечного файла расширения с параметрами, приведенными в таблице.

Таблица5.1.: Данные для файла конфигурации расширения

Параметр	Допустимые значения	Описание
hostname	Hostname	Имя хоста для хоста сегмент-сервера
port	Номер порта	Прослушивающий порт базы данных для сегмент-сервера, увеличенный на количество существующих сегментов базового порта
fselocation	Имя каталога	Каталог данных (файловое пространство) для сегмент-сервера в соответствии с системным каталогом *pg_filespace_entry*
dbid	Целое число (Integer). Не должен конфликтовать с существующими значениями *dbid*	Идентификатор базы данных для сегмент-сервера. Вводимые значения должны последовательно увеличиваться из существующих значений *dbid*, указанных в системном каталоге *gp_segment_configuration*. Например, чтобы добавить четыре узла к существующему массиву из десяти сегментов со значениями *dbid 1-10*, перечислить новые значения *dbid 11*, *12*, *13* и *14*
content	Целое число (Integer). Не должен конфликтовать с существующими значениями *content*	content ID сегмент-сервера. Основной сегмент и его зеркало должны иметь один и тот же идентификатор *content*, последовательно увеличиваемый от существующих значений
preferred_role	*p* | *m*	Определяет, является сегмент основным или зеркальным: *p* – основной, *m* – зеркальный
replication_port	Номер порта	Порт репликации файлов для сегмент-сервера, увеличенный на существующий базовый номер сегмента *replication_port*

Запуск gpexpand для инициализации новых сегмент-серверов

После создания настроечного файла необходимо запустить **gpexpand**. Утилита автоматически останавливает инициализацию сегмент-серверов базы данных **ADB** и перезапускает систему после завершения процесса.

Запуск **gpexpand** с настроечным файлом:

1. Выполнить вход на мастер хост базы данных пользователем, который будет запускать **ADB**, например, *gpadmin*.
2. Запустить утилиту **gpexpand**, указав настроечный файл с опцией *-i*. Для указания базы данных, в которой будет создана схема расширения, использовать *-D*. Например:

```
$ gpexpand -i input_file -D database1
```

Утилита определяет наличие схемы расширения для кластера **ADB**. Если схема существует, необходимо удалить ее с помощью опции *-c* перед началом новой операции расширения (*Удаление временной схемы расширения*).

3. Утилита выводит сообщение об успешном завершении инициализации новых сегмент-серверов и создании схемы расширения и завершает работу.

После завершения процесса инициализации можно подключиться к **ADB** и увидеть схему расширения. Она находится в базе данных, указанной при помощи опции *-D* или переменной среды *PGDATABASE*.

Откат неудачной установки

Откат операции настройки расширения возможен только в случае сбоя операции.

В случае если на этапе инициализации происходит сбой расширения, а база данных не работает, необходимо сначала перезапустить базу данных в режиме master-only, выполнив команду *gpstart -m*. Откат неудачного расширения выполняется с помощью следующей команды с указанием базы данных, содержащей схему расширения:

```
gpexpand --rollback -D database_name
```

5.5.4 Перераспределение таблиц

Перераспределение таблиц проводится для балансировки существующих данных по недавно расширенному кластеру.

После создания схемы расширения можно привести базу данных **ADB** в онлайн и перераспределить таблицы по всему массиву с помощью **gpexpand**. Рекомендуется запускать утилиту в период низкой нагрузки на базу, когда использование центрального процессора и блокировки таблиц имеют минимальное влияние на операции в кластере. При этом рекомендуется ранжировать таблицы для первоочередного перераспределения крупнейших и наиболее важных элементов.

Important: При перераспределении данных ADB должна работать в продуктивном режиме. Она не должна быть ограничена или находиться в режиме master. Опции *-R* или *-m* команды *gpstart* не могут быть указаны для запуска базы данных

В процессе перераспределения таблиц любые новые созданные таблицы или секции распределяются по всем сегментам точно так же, как и при нормальных условиях работы. Запросы могут обращаться ко всем сегментам до того момента, как соответствующие данные будут перераспределены в таблицы в новых сегмент-серверах. Во время перераспределения таблица или раздел блокируются и недоступны для операций чтения и записи. По завершению перераспределения возможность операций возобновляется.

- *Ранжирование таблиц для перераспределения*
- *Перераспределение таблиц с помощью gpexpand*
- *Мониторинг перераспределения таблиц*

Ранжирование таблиц для перераспределения

Порядком перераспределения таблиц можно управлять. Для этого необходимо скорректировать значения рангов таблиц в схеме расширения для определения приоритетов часто используемых таблиц и минимизации влияния на производительность. Доступное свободное место на диске может сказываться на ранжировании таблиц.

Для ранжирования таблиц с целью их последующего перераспределения необходимо подключиться к базе данных **ADB** с помощью **psql** или другого поддерживаемого клиента и обновить файл *gpexpand.status_detail* с помощью следующих команд:

```
=> UPDATE gpexpand.status_detail SET rank=10;
```

```
=> UPDATE gpexpand.status_detail SET rank=1 WHERE fq_name = 'public.lineitem';
=> UPDATE gpexpand.status_detail SET rank=2 WHERE fq_name = 'public.orders';
```

В примере команды понижают приоритет всех таблиц до *10*, а затем присваивают ранг *1* для *lineitem* и ранг *2* для *orders*. В результате сначала перераспределяется *lineitem*, затем *orders*, а далее все другие таблицы из файла *gpexpand.status_detail*.

Для исключения таблицы из перераспределения следует удалить ее из файла *gpexpand.status_detail*.

Перераспределение таблиц с помощью gpexpand

Перераспределение таблиц с помощью утилиты **gpexpand** осуществляется следующим образом:

1. Выполнить вход на мастер хост базы данных пользователем, который будет запускать **ADB**, например, *gpadmin*.
2. Запустить утилиту **gpexpand**. Можно использовать опцию *-d* или *-e* для определения периода времени сеанса расширения. Например, для запуска утилиты на срок до *60* часов:

```
$ gpexpand -d 60:00:00
```

Утилита перераспределяет таблицы до тех пор, пока последняя таблица в схеме не завершится, или пока не будет достигнут период указанной длительности (или не будет достигнуто установленное время окончания). Утилита **gpexpand** обновляет статус и время сессии в файле *gpexpand.status* при запуске и завершении перераспределения.

Мониторинг перераспределения таблиц

Во время процесса перераспределения таблицы можно запросить схему расширения. В представлении *gpexpand.expansion_progress* содержится текущая сводка хода выполнения, включая предполагаемую скорость перераспределения таблицы и расчетное время до завершения операции. Для информации о статусе таблицы необходимо запросить *gpexpand.status_detail*.

После завершения перераспределения первой таблицы *gpexpand.expansion_progress* выполняет предварительный подсчет и обновляет оценку по скорости перераспределения всех таблиц. Расчеты перезапускаются каждый раз при инициализации сеанса перераспределения таблицы с помощью **gpexpand**. Для мониторинга прогресса необходимо подключиться к базе данных **postgres** с помощью **psql** или другого поддерживаемого клиента и выполнить запрос *gpexpand.expansion_progress* с помощью следующей команды:

```
=# SELECT * FROM gpexpand.expansion_progress;
             name              |         value
-------------------------------+-----------------------
 Bytes Left                    | 5534842880
 Bytes Done                    | 142475264
 Estimated Expansion Rate      | 680.75667095996092 MB/s
 Estimated Time to Completion  | 00:01:01.008047
 Tables Expanded               | 4
 Tables Left                   | 4
(6 rows)
```

В таблице *gpexpand.status_detail* хранится информация о каждой таблице в схеме: статусы, время последнего обновления и дополнительные сведения. Для просмотра статуса таблицы необходимо подключиться к базе данных **postgres** с помощью **psql** или другого поддерживаемого клиента и выполнить запрос *gpexpand.status_detail*:

```
=> SELECT status, expansion_started, source_bytes FROM
gpexpand.status_detail WHERE fq_name = 'public.sales';
  status    |     expansion_started      | source_bytes
```

```
-----------+---------------------------+--------------
 COMPLETED | 2017-02-20 10:54:10.043869 |   4929748992
(1 row)
```

5.5.5 Удаление временной схемы расширения

Important: Для выполнения очередной операции расширения в системе ADB сначала необходимо удалить существующую схему расширения

После завершения и проверки операции расширения схему расширения можно безопасно удалить. Для этого необходимо выполнить следующий порядок действий:

1. Выполнить вход на мастер хост базы данных пользователем, который будет запускать **ADB**, например, *gpadmin*.
2. Запустить утилиту **gpexpand** с параметром *-c*. Например:

```
$ gpexpand -c
$
```

При этом в некоторых системах требуется дважды нажать клавишу *Enter*.

5.6 Расширение кластера средствами ADCM

Если кластер **ADB** разворачивается с помощью **ADCM**, часть действий по расширению кластера выполняется автоматически. После выполнения планирования нового аппаратного обеспечения необходимо добавить записи для новых хостов в выбранный кластер в интерфейсе **ADCM**, используя кнопку *Add hosts* на вкладке “Hosts”. Кроме того, необходимо выполнить инициализацию каждого хоста, если того требует провайдер хостов.

Когда хосты будут доступны для подключения по ssh для менеджера кластеров, необходимо запустить действие *Expand* кластера. В появившемся диалоге (Рис.5.10) необходимо указать следующие параметры:

- *Reboot new servers after installation* – перезагрузка хостов с новыми компонентами. Перезагрузка требуется для применения значения некоторых параметров, изменяемых в процессе установки. Сервера необходимо перезагрузить позднее вручную, если это невозможно сделать в процессе расширения;
- *Additional primary segments count* – количество сегментов, которые необходимо добавить на все хосты в кластере. Например, если в исходной конфигурации указано два сегмента на хост и в данном параметре задано два дополнительных сегмента, в результате на уже существующие в кластере хосты добавляется по два сегмента, на новые – четыре. Если увеличение количества сегментов не требуется, следует оставить значение по умолчанию – 0;
- *Attach new data catalog (optional)* – создание дополнительных сегментов в указанном каталоге. Все используемые каталоги должны содержать одинаковое количество сегментов во избежание неравномерной нагрузки на дисковую подсистему. В ином случае подготовка к расширению завершается ошибкой. Если поле не заполнено (по умолчанию), сегменты равномерно распределяются по существующим каталогам;
- *Storage for data catalog (optional)* – отформатировать и смонтировать указанное устройство (напрмер, sdc) в каталог, заданный в предыдущей опции;
- *Specify path for every non-default tablespace* – если параметр “Attach new data catalog” задан, то для каждого табличного пространства кластера необходимо указать каталог для хранения данных пространства сегментами. В левом поле следует указать имя табличного пространства, в правом – путь;

- *Check array* – проверить, что все сегмент-хосты имеют одинаковое количество основных сегментов и сегмент-зеркал, одинаковые суффиксы имен сегментов "-<n>" (-1, -2, -3. . .), ;
- *Directory for gpexpand tar file (optional)* – полный путь к директории на сегмент-хостах, куда утилита gpexpand копирует временный tar-файл. Файл содержит базу данных для создания сегмента. По умолчанию используется домашняя директория пользователя.

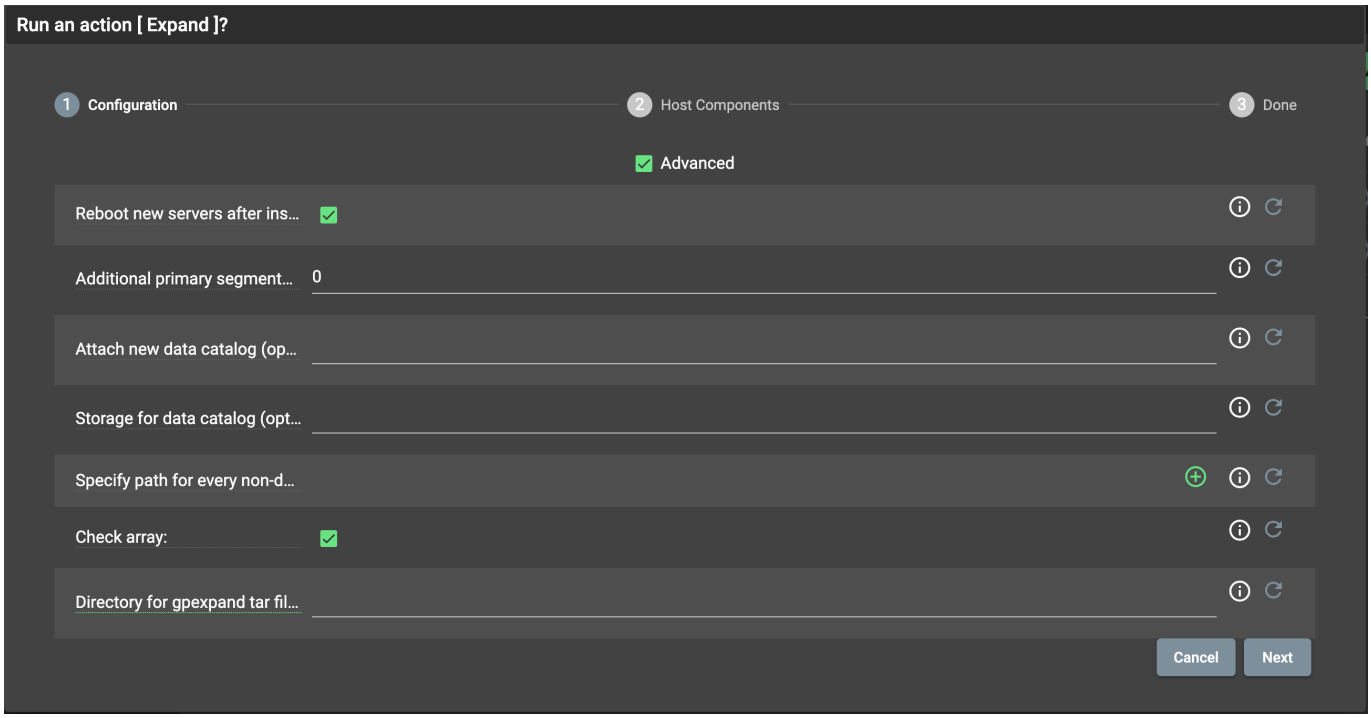


Рис.5.10.: Параметры расширения кластера

Для перехода к следующей странице конфигурации нажать кнопку *Next* и распределить компонент *ADB Segment* сервиса *ADB* по добавляемым хостам (Рис.5.11). Если используются сервисы *PXF*, *Chrony*, *Monitoring Clinets*, их компоненты также необходимо разместить на добавляемых хостах для корректного функционирования. Затем необходимо запустить расширение кластера кнопкой *Run*.

Important: В процессе расширения кластера новые хосты инициализируются в соответствии с настройками сервиса ADB аналогично уже существующим хостам. Если при создании кластера запрашивается монтирование блочных устройств для создания каталогов с данными, на хостах должны присутствовать дисковые устройства с такими же именами

На первом шаге между новыми узлами и уже существующими производится обмен ключами ssh, на добавленные хосты устанавливаются необходимые пакеты и производится их настройка. Затем генерируется схема для расширения кластера в зависимости от количества добавляемых хостов и использования зеркалирования. Если количество новых хостов в кластере больше числа сегментов на хост в используемой конфигурации, применяется *spread*-зеркалирование, иначе – *group*. На основе созданного файла схемы производится подготовка кластера к расширению. В случае успешного завершения подготовки, кластер переводится в состояние *expanding*, и возврат предыдущей конфигурации становится невозможен.

На этом этапе возможна настройка порядка перераспределения таблиц, как указано в разделе *Ранжирование таблиц для перераспределения*.

В состоянии *expanding* для кластера становится доступно действие *Redistribute*, в процессе выполнения

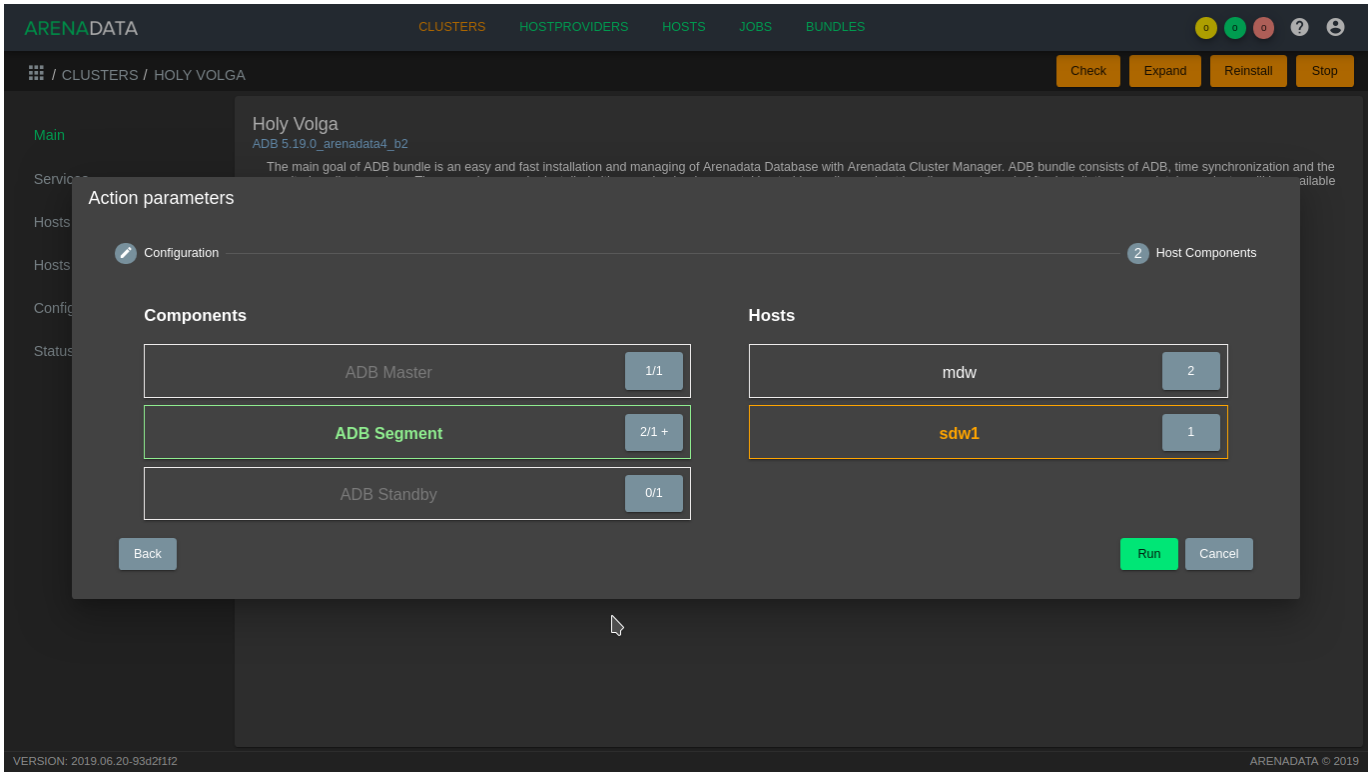


Рис.5.11.: Распределение компонентов сервиса *ADB* по новым хостам

которого производится перераспределение таблиц. Для действия необходимо указать длительность сеанса в формате *ЧЧ:ММ:СС*. В случае если перераспределение успешно завершается до истечения указанного времени, схема расширения очищается и кластер переводится в состоянии *running*. Иначе становится доступен повторный запуск действия *Redistribute* для продолжения прерванного процесса.

5.7 Добавление и удаление резервного мастера

Доступно с версии 5.19.0_arenadata4_b2

Для добавления или удаления резервного мастера предназначено действие *Init Standby Master* кластера *ADB*. Аналогично процессу расширения кластера необходимо указать, перезагружать ли добавляемый в кластер хост в процессе инициализации. Затем необходимо разместить компонент резервного мастера на хосте для его инициализации или убрать для удаления. Для запуска действия необходимо нажать кнопку *Run*.

5.8 Активация Standby Master

Доступно с версии 6.15.0_arenadata16

Для переключения на резервный мастер предназначено действие *Activate Standby* кластера *ADB*.

Important: Активация резервного мастера возможна только при неработающем мастере

Important: Для переключения мастера по желанию (не во время сбоя) необходимо сначала остановить сервис

ADB

Доступны следующие опции:

- *force activate* – для принудительного переключения, когда хост действующего мастера доступен, но процесс мастера внутри операционной системы остановлен;
- *run analyze* – для запуска analyze после переключения на всех базах кроме `template0`, `template1` и `postgres`.

Для запуска действия необходимо нажать *Run*.

После переключения мастера на резервный мастер старый мастер перестает участвовать в работе кластера *ADB*. При этом резервного мастера больше нет, так как он стал основным. Для создания резервного мастера необходимо воспользоваться *Init Standby Master*.

После включения хоста старого мастера (или сразу после переключения, если хост остается доступен) на нем необходимо:

- удалить задачи cron из под администратора ADB (gpadmin по умолчанию) для того, чтобы:
  - не было попыток обращений к незапущенному процессу устаревшего мастера;
  - не отсылать ложные метрики в мониторинг;
- переименовать (или удалить) каталог мастера (*/data1/master* по умолчанию) для того, чтобы:
  - случайно не запустить устаревший мастер;
  - иметь возможность расположить на этом хосте новый резервный мастер.

5.9 Управление сервисом *ADCC*

Для установленного сервиса *ADCC* доступны следующие действия:

- *Uninstall* – удалить компоненты *ADCC* из кластера *ADB*. Для удаления необходимо установить в диалоговом окне флаг *Erase data*. Позволяет удалить с файловой системы логи и исторические данные, хранящиеся в базе данных *ADCC*;
- *Start* – запустить web-сервер *ADCC*;
- *Stop* – остановить web-сервер *ADCC*;
- *Reconfigure & Restart* – сгенерировать конфигурационные файлы и перезапустить компоненты *ADCC*. Действие необходимо выполнить для применения изменений, внесенных в конфигурацию сервиса в интерфейсе *ADCM*.

5.10 Управление сервисом *Monitoring Clients*

Для не установленного сервиса *Monitoring Clients* доступны следующие действия:

- *Install* – установить *Monitoring Clients* на кластер *ADB*. Перед установкой необходимо импортировать адрес graphite и grafana в меню кластера *ADB* разел *Import*;
- *Delete* – удалить *Monitoring Clients* из кластера *ADB*, без установки;

Для установленного сервиса *Monitoring Clients* доступны следующие действия:

- *Uninstall* – удалить сервис *Monitoring Clients* из кластера *ADB*;
- *Reinstall* – переустановить сервис *Monitoring Clients* на кластере *ADB*;

5.11 Инструментарий Arenadata_toolkit

Инструментарий **Arenadata_toolkit** предназначен для автоматизированного управления клиентом Arenadata DB и сбору информации по его работе. **Arenadata_toolkit** представляет из себя наборы таблиц и скриптов.

5.11.1 Таблицы Arenadata_toolkit

- arenadata_toolkit.daily_operations
- arenadata_toolkit.db_files_current
- arenadata_toolkit.db_files_history
- arenadata_toolkit.operations_exclude

daily_operations

Таблица *daily_operations* описывает проведенные сервисные операции над таблицами, такие как vacuum и analyze.

Таблица5.2.: Структура таблицы *daily_operations*

Название поля	Тип данных	Описание
schema_name	text	Имя схемы
table_name	text	Имя таблицы
action	text	Действие
status	text	Статус операции
time	bigint	Длительность операции
processed_dttm	timestamp	Дата и время операции

db_files_current

Таблица *db_files_current* содержит актуальную информацию по всем файлам на всех сегментах с привязкой к таблицам, индексам и другим объектам БД при такой возможности. Она пересоздается ежедневно в 21:00 при исполнении скрипта `collect_table_stats`.

Таблица5.3.: Структура таблицы *db_files_current*

Название поля	Тип данных	Описание
oid	bigint	Внутренний id таблицы
table_name	text	Имя таблицы
table_schema	text	Имя схемы
type	char(1)	Тип таблицы. Значения аналогичны полю relkind таблицы pg_class
storage	char(1)	Тип хранения. Значения аналогичны полю relstorage таблицы pg_class
table_parent_table	text	Имя родительской таблицы (или null). Актуально для партиций
table_parent_schema	text	Схема родительской таблицы (или null). Актуально для партиций
table_database	text	Имя БД таблицы
table_tablespace	text	Имя табличного пространства таблицы
content	integer	Номер сегмента, где физически расположен файл
segment_preferred_role	char(1)	Роль сегмента
hostname	text	Имя хоста, где расположен данный файл
address	text	Адрес либо FQDN хоста
file	text	Полный путь к файлу
modifiedtime	timestamp	Время изменения файла
file_size	bigint	Размер файла в байтах

db_files_history

Таблица *db_files_history* содержит результаты всех измерений за прошедшее время. В неё копируются данные из *db_files_current* в конце скрипта `collect_table_stats`. Она позволяет наблюдать за динамикой роста той или иной таблицы или схемы. Структура таблицы схожа с предыдущей, но содержит дополнительное поле `collecttime`, отображающее время сбора информации. Таблица не очищается.

operations_exclude

Таблица *operations_exclude* содержит информацию о схемах, где не требуется проводить операции vacuum и analyze. Структура таблицы содержит в себе единственное поле `schema_name`, отображающее имена схем.

5.11.2 Скрипты

Скрипты **Arenadata_toolkit** являются наборами команд sql и запускаются системным сервисом cron.

В инструментарий входят следующие скрипты:

Подробные описания

operation.sh

Алгоритм работы скрипта:

- Выгружает все схемы обрабатываемой базы данных.
- Определяет дату последнего vacuum для каждого представления в схеме на основе `pg_stat_operations`. Если дата не находится - устанавливает минимальную.
- Сортирует представления по дате операции.

- Выполняет скрипт analyze или vacuum для каждого представления в списке, пока не закончится время работы.

  Каждая операция записывается в таблицу *arenadata_toolkit.daily_operation*.

  Скрипт запускает операцию vacuum ежедневно с 06:00 до 10:00.

  Скрипт запускает операцию analyze ежедневно с 11:00 до 15:00.

collect_table_stats.sql

Алгоритм работы скрипта:

- Собирает информацию о расположении всех таблиц.
- Собирает данные по файлам на сегментах, чьи названия соответствуют OID.
- Объединяет собранные данные с данными из pg_class обрабатываемой базы данных.
- Записывает получившиеся данные в *arenadata_toolkit.db_files_current*.
- Создает копию данных в *arenadata_toolkit.db_files_history*.

Следующие поля остаются пустыми у файлов, для которых не найдено соответствие объектам обрабатываемой БД: oid, table_name, table_schema, type, storage , table_parent_table, table_parent_schema.

Запускается вместе с *run_sql.sh*.

По умолчанию скрипт запускается ежедневно в 21:00.

vacuum_catalog_tables.sql

Скрипт ставит все таблицы из схемы pg_catalog обрабатываемой базы данных в очередь и запускает последовательные операции analyze и vacuum для каждой.

Запускается вместе с *run_sql.sh*.

По умолчанию скрипт запускается ежедневно в 00:04, 08:04 и 16:04.

gzip_pg_log.sql

Скрипт собирает директории данных по каждому сегменту и выполняет bash команду, которая архивирует логи старше 3+1 дней.

Запускается вместе с *run_sql_to_gpssh.sh*.

Скрипт запускается ежедневно в 01:00.

system_db.sh

Скрипт выполняет vacuum freeze для баз данных template1, postgres и gpperfmon (если есть).

Скрипт запускается ежедневно в 02:00.

5.11.3 Манипуляции со скриптами

Хотя сами скрипты не рекомендуется модифицировать, так как они работают полностью автоматически, следующие операции доступны пользователю для работы с ними.

Исключения таблиц

Пользователь может исключить целиковые схемы, добавляя их в *arenadata_toolkit.operation_exclude*.

Исключать отдельные таблицы не представляется возможным.

Управление временем запуска

Время запуска и время остановки можно прописывать вручную в следующих файлах:

`/home/gpadmin/arenadata_configs/crontab.txt` - время запуска.

`/home/gpadmin/arenadata_configs/operations.sh` - время остановки.

Проверка запуска

При необходимости, пользователь может проверить, когда последний раз запускался скрипт, при помощи команды (пример для *collect_table_stats.sql*):

```
crontab -l | grep /home/gpadmin/arenadata_configs/collect_table_stats.sql
```