Руководство системного администратора

Системы «Учет Средств Индивидуальной Защиты и Спецодежды (СИЗ и СО) (промышленный экземпляр)

АО «Компания»

Автор: Пазилбек А.

Дата создания: 7 ноября 2024 г.

Изменен: 7 ноября 2024 г

Версия: 1.0

**Утверждено:**

|  |  |
| --- | --- |
| Руководитель проекта от  Заказчика |  |
| Руководитель проекта от  Исполнителя |  |

## Контроль над документом

### Записи изменений

| Дата | Автор | Версия | Что изменено |
| --- | --- | --- | --- |
|  |  |  |  |
| 7 ноября 2024 г. | Пазилбек А. | 1.0 | Новый документ |
|  |  |  |  |
|  |  |  |  |

### Рецензенты

| ФИО | Должность | Подпись |
| --- | --- | --- |
|  |  |  |
|  |  |  |

### Рассылка

| № | ФИО | Расположение |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Заметки получателю: Если** Вы получили напечатанный экземпляр этого документа или если Вы распечатали его электронную копию, пожалуйста, напишите свое имя на заглавной странице. Это необходимо для контроля над документами.

Содержание

Контроль над документом ii

Записи изменений ii

Рецензенты ii

Рассылка ii

Введение 1

Назначение 1

Описание установленного программного обеспечения 2

Уровень Базы Данных PostgreSQL 6

Общие сведения 6

Конфигурация PostgreSQL 9

Конфигурация Host Based Access Control (pg\_hba.conf) 14

Запуск, останов сервисов баз данных 16

Резервное копирование 17

Восстановление из резервной копии 25

Мониторинг работы сервера базы данных PostgreSQL 27

Служебные скрипты PostgreSQL 28

Автоматические задачи планировщика операционной системы (cron) 29

Уровень сервера приложений Apache Tomcat 30

Общие сведения 30

Конфигурация сервера Apache Tomcat 33

Свойства приложений 42

Обновление приложеня 52

Добавление нового сервиса Apache Tomcat уровня Web Client 52

Добавление нового сервиса Apache Tomcat уровня Middleware 53

Мониторинг сервисов Apache Tomcat 54

Мониторинг пользовательских сессий 55

Настройка журнала изменения сущностей 56

Просмотр журнала изменения сущностей 57

Очистка логов приложения 57

Резервное копирование 57

Команды управления Apache Tomcat 57

Описание скриптов обслуживания 58

Автоматические задачи планировщика операционной системы (cron) 59

Порядок запуска/останова всей системы 60

Запуск системы 60

Останов системы 60

Изменение учетных записей 61

Изменение учетной записи пользователя tal базы данных PostgreSQL 61

Клонирование экземпляра 62

Подготовка нового сервера и программного обеспечения 62

Создание пустой базы данных 62

Перенос базы данных с продуктивного сервера 62

Настройка сервисов tomcat 63

Настройка параметров приложений Cuba 63

Настройка systemd сервисов 64

Пример ручного клонирования системы с продуктивной на тестовую среду 64

Доступ к приложению CUBA 66

Общие сведения 66

Уровень сервиса HAproxy 67

Конфигурация 67

Просмотр логов HAproxy 69

Мониторинг сервиса HAproxy 69

Параметры cookies HAproxy 69

Команды управления сервиса HAproxy 70

Доступ к приложению Apache Tomcat 71

Открытые и Закрытые Вопросы 72

Открытые Вопросы 72

Закрытые Вопросы 72

1. To update the table of contents, put the cursor anywhere in the table and press [F9]. To change the number of levels displayed, select the menu option Insert‑>Index and Tables, make sure the Table of Contents tab is active, and change the Number of Levels to a new value.

## Введение

1. It is very important that you use timely information that is pertinent to the release of the system components you are implementing.  
     
   Update this deliverable to correspond to the particular release of all aspects of the applications environment (Database Tier, Application Tier, and Client Tier).   
     
   The procedures included are provided as examples only and are very much subject to being unaligned with the current releases you are running.

### Назначение

Целью документа **Руководство системного администратора** является описание процедур и инструментов, которые используются для того, чтобы облегчить работникам департамента архитектуры ИТ и департамента эксплуатации поддержку системного окружения приложения. Этот документ будет являться главным опорным документом при создании справочника решения технических проблем по поддержке системы.

1. For mid to large size projects, this document will be an input to Publish System Management Guide (DO.090). For a smaller project, this document may become the System Management Guide.

Данный документ содержит следующие разделы:

1. The following section should be updated to reflect the organization structure that will support the network, servers, client devices and applications.

* **Уровень Сервера Базы Данных PostgreSQL** – Процедуры и инструменты, которые требуются для управления данными и серверами на уровне базы данных. Конфигурация, управление резервными копиями.
* **Уровень Сервера Приложений Apache Tomcat** – Процедуры и инструменты, необходимые для работы серверов на уровне приложений.
* **Уровень балансировщика нагрузки HAproxy** – Процедуры и инструменты, необходимые для обеспечения эффективного распределения запросов и стабильной работы системы приложений.
* **Изменение учетных записей базы данных** – Процедуры и инструменты, необходимые для изменения паролей ключевых пользователей базы данных и поддержания доступа к системе.
* **Порядок запуска/останова всей системы** – Процедуры, необходимые для пошагового запуска/останова всех программных составляющих системы.

### Описание установленного программного обеспечения

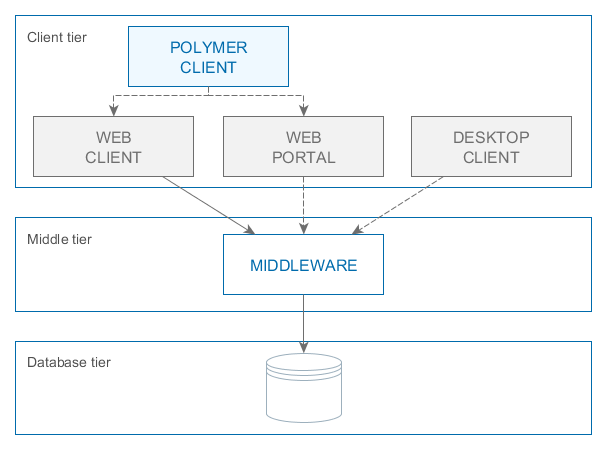
Этот документ описывает процедуры управления и обслуживания следующих системам, приложений и технической инфраструктуры:

1. База данных PostgreSQL (версия 16.4)
2. Сервер приложений Apache Tomcat (версия 9.0.91)
3. Балансировщик HAproxy (версия 2.6.12)
4. Open JDK Runtime Environment Amazon Corretto (версия 8.422.05.1)
5. Платформа CUBA (версия 7.2)

Приложения написаны с использованием платформы CUBA для написания web приложений на JAVA.

Платформа позволяет строить приложения по классической трехуровневой схеме: клиентский уровень, средний слой, база данных. Уровень отражает степень "удаленности" от хранимых данных.

На каждом уровне возможно создание одного или нескольких блоков (units) приложения. Блок представляет собой обособленную исполняемую программу, взаимодействующую с другими блоками приложения. Средства платформы CUBA позволяют создавать блоки в виде веб-приложений и десктопных приложений.



*Middleware*

Средний слой, содержащий основную бизнес-логику приложения и выполняющий обращения к базе данных. Представляет собой отдельное веб-приложение под управлением стандартного контейнера Java EE Web Profile.

*Web Client*

Основной блок клиентского уровня. Содержит интерфейс, предназначенный для внутренних пользователей организации. Представляет собой отдельное веб-приложение под управлением стандартного контейнера Java EE Web Profile. Реализация пользовательского интерфейса основана на фреймворке Vaadin.

*Desktop Client*

Дополнительный блок клиентского уровня. Содержит интерфейс, предназначенный для внутренних пользователей организации. Представляет собой десктопное Java-приложение, реализация пользовательского интерфейса основана на фреймворке Java Swing.

*Web Portal*

Дополнительный блок клиентского уровня. Может содержать интерфейс для внешних пользователей и средства интеграции с мобильными устройствами и сторонними приложениями. Представляет собой отдельное веб-приложение под управлением стандартного контейнера Java EE Web Profile. Реализация пользовательского интерфейса основана на фреймворке Spring MVC.

*Polymer Client*

Дополнительный клиентский блок на чистом JavaScript, предоставляющий интерфейс для внешних пользователей. Основан на фреймворке Google Polymer, работает со средним слоем через REST API, запущенный в блоке Web Client или Web Portal.

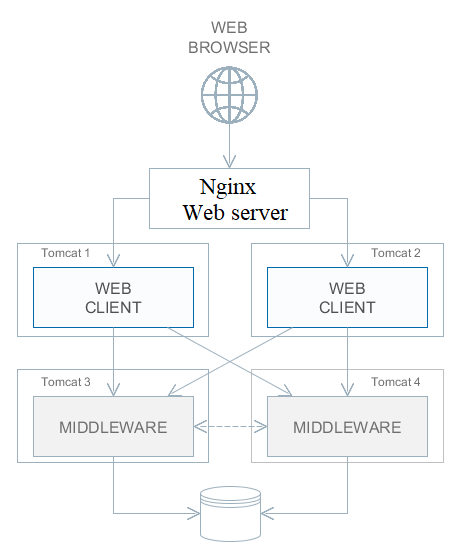
Обязательным блоком любого приложения является средний слой - Middleware. Для реализации пользовательского интерфейса на продуктивном сервере используется шесть клиентских блоков Web Client.

Все основанные на Java клиентские блоки взаимодействуют со средним слоем одинаковым образом посредством протокола HTTP, что позволяет размещать средний слой произвольным образом, в том числе за сетевым экраном.

На продуктивном экземпляре используется способ масштабирования CUBA-приложения, состоящего из блоков Middleware и Web Client, для поддерживания нагрузки и ужесточении требований к отказоустойчивости.

Кластер серверов Web Client работает с кластером серверов Middleware.

Это максимальный вариант масштабирования, обеспечивающий отказоустойчивость и балансировку нагрузки для Middleware и Web Client.



Подключение пользователей к серверам Web Client производится через Load Balancer. Серверы WebClient работают с кластером серверов Middleware. Для этого им не требуется дополнительный Load Balancer - достаточно определить список URL серверов Middleware в свойстве cuba.connectionUrlList.

В кластере серверов Middleware организуется взаимодействие для обмена информацией о пользовательских сессиях, блокировках и пр. При этом обеспечивается полная отказоустойчивость блока Middleware - при отключении одного из серверов выполнение запросов от клиентских блоков продолжается на доступном сервере прозрачно для пользователей.

*Общая, детальная архитектура всех серверов приложений и баз данных приведена на схеме:*



## Уровень Базы Данных PostgreSQL

1. Give an overview of database management considerations and issues.

### Общие сведения

#### Корпоративная среда размещения

База данных PostgreSQL 16.4 установлена в не отказоустойчивой конфигурации и расположена на одном сервере

|  |  |  |
| --- | --- | --- |
| Сервер | IP адрес | Роль сервера |
|  | 192.168.1.2 | Primary |

#### Параметры операционной системы

Для базы данных PostgreSQL 16 выделен сервер со следующими характеристиками:

* Debian 12
* ОЗУ 62 Гб
* 10 ядер - Intel(R) Xeon(R) Gold 6258R CPU @ 2.70GHz

#### Файловая система

На сервере установлены следующие файловые системы:

|  |  |  |  |
| --- | --- | --- | --- |
| Устройство | Точка монтирования | Размер | Файловая система |
| /dev/sda1 | /boot/efi | 511M | vfat |
| /dev/sda2 | /boot | 456M | Ext2 |
| /dev/mapper/DBServer-01--vg-root | / | 96G | Ext4 |
| /dev/mapper/vg01-data | /data | 511G | Ext4 |
| /swapfile | [SWAP] | 1G | swap |

#### Пользователь операционной системы

Владельцем установки кластера базы данных PostgreSQL является пользователь postgres. Пользователь postgres добавлен в /etc/sudoers для возможности выполнять команды запуска/останова/перезапуска сервисов, с правами привилегированного пользователя используя команды sudo.

Добавленные записи в /etc/sudoers

|  |
| --- |
| Описание |
| Cmnd\_Alias SERVICES = /usr/bin/systemctl start [A-z0-9]\*, /usr/bin/systemctl stop [0-9A-z]\*, /usr/bin/systemctl reload [A-z0-9]\*, /usr/bin/systemctl restart [A-z0-9]\*, /usr/bin/systemctl status [A-z0-9]\* |
| postgres ALL = NOPASSWD:SERVICES |

#### Переменные окружения пользователя postgres

В файле .profile прописаны следующие переменные окружения пользователя tomcat

|  |  |
| --- | --- |
| Переменная | Значение |
| PGDATA | /data |
| PGLOG | $PGDATA/log |
| LD\_LIBRARY\_PATH | /opt/postgres/16/lib:$LD\_LIBRARY\_PATH |
| PATH | /opt/postgres/16/bin:$PATH:$HOME/bin |

#### Основные директории PostgreSQL

Директории, необходимые для работы PostgreSQL

|  |  |
| --- | --- |
| Директория | Описание |
| /opt/postgres/16 | Бинарные (исполняемые, библиотеки, расширения) файлы |
| /data/ | Директория данных кластера PostgreSQL |
| /data/log | Директория log файлов работы PostgreSQL |
| /data/pg\_wal | Директория хранения WAL сегментов |
| /backup/postgres/pg\_probackup/wal | Директория хранения архивных WAL сегментов |
| /backup/postgres/pg\_probackup/backups | Используется для хранения резервных копий |
| /backup/postgres/log | Директория логов выполнения резервной копии |

#### Базы данных PostgreSQL

В PostgreSQL созданы следующие базы данных

|  |  |  |
| --- | --- | --- |
| База данных | Владелец | Описание |
|  | tal | Продуктивная база данных приложений CUBA |
| postgres | postgres | Служебная база данных |
| template0 template1 | postgres | Шаблоны базы данных |

#### Установленные расширения базы данных

В базе данных установлены следующие расширения:

|  |  |  |
| --- | --- | --- |
| Расширения | Версия | URL разработчика |
| plpgsql | 1.0 | Стандартное |

### Конфигурация PostgreSQL

Параметры конфигурации базы данных в файле /data/prod/postgresql.conf

|  |  |  |
| --- | --- | --- |
| Наменование параметра | Значение |  |
| listen\_addresses | '\*' | Задаёт адреса TCP/IP, по которым сервер будет принимать подключения клиентских приложений |
| port |  | TCP-порт, открываемый сервером; по умолчанию, 5432 |
| max\_connections | 100 | Определяет максимальное число одновременных подключений к серверу БД |
| superuser\_reserved\_connections | 3 | Определяет количество «слотов» подключений, которые PostgreSQL будет резервировать для суперпользователей |
| shared\_buffers | 4GB | Задаёт объём памяти, который будет использовать сервер баз данных для буферов в разделяемой памяти |
| huge\_pages | Try | Включает/отключает использование огромных страниц памяти. Допустимые значения: try (попытаться, по умолчанию), on (вкл.) и off (выкл.) |
| work\_mem | 82MB | Задаёт объём памяти, который будет использоваться для внутренних операций сортировки и хеш-таблиц, прежде чем будут задействованы временные файлы на диске. Значение по умолчанию — четыре мегабайта (4MB). |
| maintenance\_work\_mem | 819MB | Задаёт максимальный объём памяти для операций обслуживания БД, в частности VACUUM, CREATE INDEX и ALTER TABLE ADD FOREIGN KEY. По умолчанию его значение — 64 мегабайта (64MB). |
| autovacuum\_work\_mem | -1 | Задаёт максимальный объём памяти, который будет использовать каждый рабочий процесс автоочистки. По умолчанию равен -1, что означает, что этот объём определяется значением maintenance\_work\_mem. Этот параметр не влияет на поведение команды VACUUM, выполняемой в других контекстах. |
| wal\_level | replica | Параметр wal\_level определяет, как много информации записывается в WAL. Со значением replica (по умолчанию) в журнал записываются данные, необходимые для поддержки архивирования WAL и репликации, включая запросы только на чтение на ведомом сервере. Вариант minimal оставляет только информацию, необходимую для восстановления после сбоя или аварийного отключения. Наконец, logical добавляет информацию, требующуюся для поддержки логического декодирования |
| fsync | on | Если этот параметр установлен, сервер PostgreSQL старается добиться, чтобы изменения были записаны на диск физически, выполняя системные вызовы fsync() или другими подобными методами. Это даёт гарантию, что кластер баз данных сможет вернуться в согласованное состояние после сбоя оборудования или операционной системы. |
| full\_page\_writes | # | Когда этот параметр включён, сервер Postgres Pro записывает в WAL всё содержимое каждой страницы при первом изменении этой страницы после контрольной точки. Это необходимо, потому что запись страницы, прерванная при сбое операционной системы, может выполниться частично, и на диске окажется страница, содержащая смесь старых данных с новыми. При этом информации об изменениях на уровне строк, которая обычно сохраняется в WAL, будет недостаточно для получения согласованного содержимого такой страницы при восстановлении после сбоя. Сохранение образа всей страницы гарантирует, что страницу можно восстановить корректно, ценой увеличения объёма данных, которые будут записываться в WAL. |
| wal\_compression | on | Когда этот параметр имеет значение on, сервер PostgreSQL сжимает образ полной страницы, записываемый в WAL, когда включён режим full\_page\_writes или при создании базовой копии. Сжатый образ страницы будет развёрнут при воспроизведении WAL. Значение по умолчанию — off. Изменить этот параметр могут только суперпользователи.  Этот параметр позволяет без дополнительных рисков повреждения данных уменьшить объём WAL, ценой дополнительной нагрузки на процессор, связанной со сжатием данных при записи в WAL и разворачиванием их при воспроизведении WAL. |
| max\_wal\_size | 1GB | Максимальный размер, до которого может вырастать WAL во время автоматических контрольных точек. Это мягкий предел; размер WAL может превышать max\_wal\_size при особых обстоятельствах, например, при высокой нагрузке, сбое в archive\_command или при большом значении wal\_keep\_segments. Значение по умолчанию — 1 ГБ. Увеличение этого параметра может привести к увеличению времени, которое потребуется для восстановления после сбоя. |
| min\_wal\_size | 80MB | Пока WAL занимает на диске меньше этого объёма, старые файлы WAL в контрольных точках всегда перерабатываются, а не удаляются. Это позволяет зарезервировать достаточно места для WAL, чтобы справиться с резкими скачками использования WAL, например, при выполнении больших пакетных заданий. Значение по умолчанию — 80 МБ. |
| archive\_mode | on | Когда параметр archive\_mode включён, полные сегменты WAL передаются в хранилище архива командой archive\_command |
| archive\_command | '/opt/postgres/16/bin/pg\_probackup archive-push -B /backup/postgres/pg\_probackup --compress-algorithm=zlib --compress-level=9 --instance=main\_v16 --wal-file-path=%p --wal-file-name=%f' | Команда локальной оболочки, которая будет выполняться для архивации завершённого сегмента WAL. Любое вхождение %p в этой строке заменяется путём архивируемого файла, а вхождение %f заменяется только его именем. (Путь задаётся относительно рабочего каталога сервера, то есть каталога данных кластера.) Чтобы вставить в команду символ %, его нужно записать как %%. Важно, чтобы команда возвращала нулевой код, только если она завершается успешно. |
| restore\_command | '/opt/postgres/16/bin/pg\_probackup archive-get -B /backup/postgres/pg\_probackup --instance=main\_v16 --wal-file-path=%p --wal-file-name=%f' | Команда оболочки ОС, которая выполняется для извлечения архивного сегмента файлов WAL. Этот параметр требуется для восстановления из архива, но необязателен для потоковой репликации. Любое вхождение %f в строке заменяется именем извлекаемого из архива файла, а %p заменяется на путь назначения при копировании на сервере. |
| recovery\_target\_action | 'promote' | Указывает, какое действие должен предпринять сервер после достижения цели восстановления. Вариант по умолчанию — pause, что означает приостановку восстановления. Второй вариант, promote, означает, что процесс восстановления завершится и сервер начнёт принимать подключения. Наконец, с вариантом shutdown сервер остановится, как только цель восстановления будет достигнута. |
| log\_destination | 'stderr' | Postgres поддерживает несколько методов протоколирования сообщений сервера: stderr, csvlog и syslog. На Windows также поддерживается eventlog. В качестве значения log\_destination указывается один или несколько методов протоколирования, разделённых запятыми. По умолчанию используется stderr. Параметр можно задать только в конфигурационных файлах или в командной строке при запуске сервера. |
| logging\_collector | on | Параметр включает сборщик сообщений (logging collector). Это фоновый процесс, который собирает отправленные в stderr сообщения и перенаправляет их в журнальные файлы. |
| log\_directory | 'log' | При включённом logging\_collector, определяет каталог, в котором создаются журнальные файлы. Можно задавать как абсолютный путь, так и относительный от каталога данных кластера. Параметр можно задать только в конфигурационных файлах или в командной строке при запуске сервера |
| log\_filename | 'postgresql-%Y-%m-%d\_%H%M%S.log' | При включённом logging\_collector задаёт имена журнальных файлов. Значение трактуется как строка формата в функции strftime, поэтому в ней можно использовать спецификаторы % для включения в имена файлов информации о дате и времени. |
| log\_rotation\_age | 1d | Определяет максимальное время жизни отдельного журнального файла, при включённом logging\_collector. После того как прошло заданное количество минут, создаётся новый журнальный файл. |
| log\_rotation\_size | 10MB | Определяет максимальный размер отдельного журнального файла, при включённом logging\_collector. После того как заданное количество килобайт записано в текущий файл, создаётся новый журнальный файл. |
| log\_min\_messages | error | Управляет минимальным уровнем сообщений, записываемых в журнал сервера. Допустимые значения DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, INFO, NOTICE, WARNING, ERROR, LOG, FATAL и PANIC. |
| log\_min\_duration\_statement | 10000 | Записывает в журнал продолжительность выполнения всех команд, время работы которых равно или превышает указанное количество миллисекунд. Значение 0 (ноль) заставляет записывать продолжительность работы всех команд. Значение -1 (по умолчанию) запрещает регистрировать продолжительность выполнения операторов |
| log\_line\_prefix | '%t [%p]: pid=%p,user=%u,db=%d,app=%a,client=%h ' | Строка, в стиле функции printf, которая выводится в начале каждой строки журнала сообщений. С символов % начинаются управляющие последовательности, которые заменяются статусной информацией, описанной ниже. Неизвестные управляющие последовательности игнорируются. Все остальные символы напрямую копируются в журнальную строку. |
| log\_lock\_waits | on | Определяет, нужно ли фиксировать в журнале события, когда сеанс ожидает получения блокировки дольше, чем указано в deadlock\_timeout. Это позволяет выяснить, не связана ли низкая производительность с ожиданием блокировок. |
| log\_timezone | 'Asia/Almaty' | Устанавливает часовой пояс для штампов времени при записи в журнал сервера. |
| track\_activities | on | Включает сбор сведений о текущих командах, выполняющихся во всех сеансах (в частности, отслеживается время запуска команды). По умолчанию этот параметр включён. |
| track\_counts | on | Включает сбор статистики активности в базе данных. Этот параметр по умолчанию включён, так как собранная информация требуется демону автоочистки. |
| track\_io\_timing | on | Включает замер времени операций ввода/вывода. Этот параметр по умолчанию отключён, так как для этого требуется постоянно запрашивать текущее время у операционной системы, что может значительно замедлить работу на некоторых платформах. Для оценивания издержек замера времени на вашей платформе можно воспользоваться утилитой pg\_test\_timing. Статистику ввода/вывода можно получить через представление pg\_stat\_database, в выводе EXPLAIN (когда используется параметр BUFFERS) и через представление pg\_stat\_statements. |
| track\_functions | all | Включает подсчёт вызовов функций и времени их выполнения. Значение pl включает отслеживание только функций на процедурном языке, а all — также функций на языках SQL и C. Значение по умолчанию — none, то есть сбор статистики по функциям отключён. |
| datestyle | 'iso, mdy' | Задаёт формат вывода значений даты и времени, а также правила интерпретации неоднозначных значений даты |
| timezone | 'Asia/Almaty' | Задаёт часовой пояс для вывода и ввода значений времени |
| lc\_messages | 'en\_US.utf8' | Устанавливает язык выводимых сообщений |
| lc\_monetary | 'en\_US.utf8' | Устанавливает локаль для форматирования денежных сумм |
| lc\_numeric | 'en\_US.utf8' | Устанавливает локаль для форматирования чисел |
| lc\_time | 'en\_US.utf8' | Устанавливает локаль для форматирования даты и времени |
| default\_text\_search\_config | 'pg\_catalog.english' | Выбирает конфигурацию текстового поиска для тех функций текстового поиска, которым не передаётся аргумент, явно указывающий конфигурацию |

### Конфигурация Host Based Access Control (pg\_hba.conf)

Аутентификация клиентов управляется конфигурационным файлом, который традиционно называется pg\_hba.conf и расположен в каталоге с данными кластера базы данных. (HBA расшифровывается как host-based authentication — аутентификации по имени узла.) Файл pg\_hba.conf, со стандартным содержимым, создаётся командой initdb при инициализации каталога с данными. Обычный формат файла pg\_hba.conf представляет собой набор записей, по одной в строке. Пустые строки игнорируются, как и любой текст комментария после знака #. Каждая запись обозначает тип соединения, диапазон IP-адресов клиента (если он соотносится с типом соединения), имя базы данных, имя пользователя, и способ аутентификации, который будет использован для соединения в соответствии с этими параметрами.

#### Содержимое файла pg\_hba.conf

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TYPE | DATABASE | USER | ADDRESS | METHOD |
| local | all | all |  | trust |
| host | all | all | 127.0.0.1/32 | trust |
| host | all | all | 192.168.1.0/24 | scram-sha-256 |
| host | all | all | 10.0.50.0/24 | scram-sha-256 |
| host | all | all | ::1/128 | trust |
| local | replication | all |  | trust |
| host | replication | all | 127.0.0.1/32 | trust |
| host | replication | all | ::1/128 | trust |

Значения полей описаны ниже:

*local*

Управляет подключениями через Unix-сокеты. Без подобной записи подключения через Unix-сокеты невозможны.

*host*

Управляет подключениями, устанавливаемыми по TCP/IP.

*database*

Определяет, каким именам баз данных соответствует эта запись. Значение all определяет, что подходят все базы данных. Значение replication показывает, что запись соответствует, если запрашивается подключение физической репликации (имейте в виду, что подключения репликации не определяют какую-то конкретную базу данных)

*user*

Указывает, какому имени (или именам) пользователя базы данных соответствует эта запись. Значение all показывает, что это подходит всем пользователям.

*address*

Указывает адрес (или адреса) клиентской машины, которым соответствует данная запись. Это поле может содержать или имя компьютера, или диапазон IP-адресов, или одно из нижеупомянутых ключевых слов.

Диапазон IP-адресов указывается в виде начального адреса диапазона, дополненного косой чертой (/) и длиной маски CIDR. Длина маски задаёт количество старших битов клиентского IP-адреса, которые должны совпадать с битами IP-адреса диапазона. Биты, находящиеся правее, в указанном IP-адресе должны быть нулевыми. Между IP-адресом, знаком / и длиной маски CIDR не должно быть пробельных символов.

Типичные примеры диапазонов адресов IPv4, указанных таким образом: 172.20.143.89/32 для одного компьютера, 172.20.143.0/24 для небольшой и 10.6.0.0/16 для крупной сети. Диапазон адресов IPv6 может выглядеть как ::1/128 для одного компьютера (это адрес замыкания IPv6) или как fe80::7a31:c1ff:0000:0000/96 для небольшой сети. 0.0.0.0/0 представляет все адреса IPv4, а ::0/0 — все адреса IPv6. Чтобы указать один компьютер, используйте длину маски 32 для IPv4 или 128 для IPv6. Опускать замыкающие нули в сетевом адресе нельзя.

*method*

Указывает метод аутентификации, когда подключение соответствует этой записи. Варианты выбора приводятся ниже; подробности в Разделе 19.3.  
  
 trust

Разрешает безусловное подключение. Этот метод позволяет тому, кто может подключиться к серверу с базой данных Postgres, войти под любым желаемым пользователем Postgres без введения пароля и без какой-либо другой аутентификации.

md5

Проверяет пароль пользователя, производя аутентификацию SCRAM-SHA-256 или MD5.

Файл pg\_hba.conf прочитывается во время запуска и в момент получения основным процессом сервера сигнала SIGHUP. Если вы редактируете файл во время работы системы, необходимо послать сигнал процессу postmaster (используя pg\_ctl reload или kill -HUP), чтобы он прочел обновленный файл.

Для применения параметров измененного файла pg\_hba.conf испльзуйте команду операционой системы:

sudo systemctl reload postgresql

### Запуск, останов сервисов баз данных

#### Скрипт systemd для запуска/останова сервисов PostgreSQL

На сервере DBServer-01.company.com настроен скрипт запуска systemd, который хранится в директории /usr/lib/systemd/system

Содержимое файла /usr/lib/systemd/system/postgresql-16.service:

----------------------------------------------------------------------------

[Unit]

Description=PostgreSQL 16.4 database server

After=syslog.target network.target

[Service]

Type=forking

TimeoutSec=120

User=postgres

Environment=PGDATA=/data/

PIDFILE=/data/postmaster.pid

ExecStart=/opt/postgres/16/bin/pg\_ctl start -w -D "/data" -l "/data/log/postgresql.log"

ExecStop=/opt/postgres/16/bin/pg\_ctl stop -m fast -w -D "/data"

ExecReload=/opt/postgres/16/bin/pg\_ctl reload -D "/data"

[Install]

WantedBy=multi-user.target

----------------------------------------------------------------------------

Запуск/останов кластера выполняется пользователем postgres

|  |  |
| --- | --- |
| Наменование | Значение |
| sudo systemctl start | запуск кластера PostgreSQL |
| sudo systemctl stop | останов кластера PostgreSQL |
| sudo systemctl reload | считывание и применение параметров кластера PostgreSQL без перезагрузки |
| sudo systemctl status | проверка статуса запущенного кластера PostgreSQL |

### Резервное копирование

В кластере PostgreSQL настроена резервирование с помощью сторонней утилиты pg\_probackup (<https://postgrespro.ru/docs/postgrespro/16/app-pgprobackup>).

pg\_probackup — утилита для управления резервным копированием и восстановлением кластеров баз данных PostgreSQL, позволяющее создавать полные и инкрементальные резервные копии, управлять политиками устаревания, а также получать список резервных копий и моментов времени, доступных для восстановления.

По сравнению с другими средствами резервного копирования pg\_probackup имеет следующие преимущества, полезные для реализации различных стратегий резервного копирования и работы с базами данных большого объёма:

* Инкрементальное копирование: выбирая один из трёх режимов инкрементального копирования, вы можете реализовать стратегию резервного копирования, соответствующую вашему профилю транзакционной нагрузки. Это позволяет сэкономить место на диске и создавать копии быстрее, чем при полном копировании. Восстановление инкрементальных копий также осуществляется быстрее, чем воспроизведение файлов WAL.
* Инкрементальное восстановление: ускорение восстановления из копии благодаря повторному использованию неизменённых страниц, имеющихся в PGDATA.
* Проверка: автоматический контроль целостности данных и проверка резервных копий без восстановления данных кластера.
* Контроль целостности: выполняемая по запросу проверка экземпляра Postgres с помощью команды checkdb.
* Политика хранения: управление архивами WAL и резервными копиями в соответствии с установленными правилами их хранения. Вы можете ограничить хранение резервных копий по времени или их количеству, а также переопределить время жизни (TTL) для избранных копий. Потерявшие актуальность резервные копии могут объединяться или удаляться.
* Параллельное выполнение: выполнение внутренних процессов команд backup, restore, merge, delete, validate и checkdb в несколько параллельных потоков.
* Сжатие: хранение копируемых данных в сжатом состоянии для экономии дискового пространства.
* Исключение дублирования: экономия дискового пространства за счёт фильтрации при инкрементальном копировании файлов, не содержащих непосредственно данные (например, файлов \_vm или \_fsm), если эти файлы не изменялись с момента создания предыдущей копии в цепочке инкрементальных копий.
* Удалённый режим работы: выполнение резервного копирования экземпляра Postgres, находящегося в удалённой системе, и удалённое восстановление.
* Получение резервной копии с ведомого: исключение дополнительной нагрузки на ведущий сервер.
* Архивирование внешних каталогов: резервное копирование файлов и каталогов, расположенных вне каталога данных Postgres (PGDATA), например скриптов, файлов конфигурации, журналов или SQL-дампов.
* Каталогизация резервных копий: получение списка резервных копий и соответствующей метаинформации в виде простого текста или JSON.
* Каталогизация архивов WAL: получение списка всех линий времени в WAL и соответствующей метаинформации в виде простого текста или JSON.
* Частичное восстановление: восстановление только избранных баз данных.

Для управления резервными копиями pg\_probackup создаёт каталог резервных копий. В этом каталоге сохраняются все файлы резервных копий с дополнительной метаинформацией, а также архивы WAL, необходимые для восстановления на момент времени. Вы можете хранить резервные копии разных экземпляров в отдельных подкаталогах одного каталога копий.

Используя pg\_probackup, вы можете выполнять полное или инкрементальное резервное копирование:

* Полные резервные копии содержат все файлы данных, необходимые для восстановления кластера баз данных с нуля.
* Инкрементальные копии создаются на уровне страниц и включают только те данные, которые изменились со времени последнего копирования. Это позволяет сэкономить место на диске и создавать копии быстрее, чем при полном копировании. Восстановление инкрементальных копий также осуществляется быстрее, чем воспроизведение файлов WAL. pg\_probackup использует на сервере режим инкрементального копирования Page. В режиме PAGE pg\_probackup сканирует все файлы WAL в архиве с момента создания предыдущей полной или инкрементальной копии. Новая резервная копия будет содержать только страницы, фигурирующие в записях WAL. При этом необходимо, чтобы в архиве WAL сохранялись все файлы WAL, записанные после предыдущей копии. Если размер этих файлов сравним с общим размером файлов базы данных, ускорение будет менее значительным, но размер копии будет всё же меньше.

#### Инициализация каталога резервных копий

pg\_probackup сохраняет все файлы копируемых данных и WAL в соответствущих подкаталогах каталога резервных копий.

Для инициализации каталога резервных копий выполнена команда:

pg\_probackup init -B /backup/postgres/pg\_probackup

Параметр –B указывает на каталог, предназначенный для резервных копий. Если каталог\_копий уже существует, он должен быть пустым. В противном случае pg\_probackup выдаст ошибку.

Пользователь, запускающий pg\_probackup, должен иметь полный доступ к каталогу\_копий.

pg\_probackup создаёт каталог\_копий со следующими подкаталогами:

* wal/ — каталог для файлов WAL.
* backups/ — каталог для файлов резервных копий.

Проинициализировав каталог резервных копий, вы можете добавить определение копируемого экземпляра.

#### Инициализация каталога резервных копий

Определение копируемого экземпляра

pg\_probackup может сохранять резервные копии разных кластеров баз данных в одном каталоге резервных копий. Для создания необходимых подкаталогов вы должны определить копируемый экземпляр в каталоге копий для каждого кластера баз данных, копию которого вы будете делать.

Для определения копируемого экземпляра выполнена команда:

pg\_probackup add-instance -B /backup/postgres/pg\_probackup -D /data/prod --instance main\_v16

Здесь:

-B - указывает на каталог, предназначенный для хранения резервных копий

-D - каталог, содержащий данные кластера, копию которого вы хотите сделать. Для подготовки и использования pg\_probackup необходимо иметь право записи в этот каталог.

--instance - имя подкаталогов, в которых будут храниться файлы копируемых данных и WAL для этого кластера.

Так же дополнительно можно указать параметры удалённого режима, если каталог данных располагается удалённо.

pg\_probackup создаёт подкаталоги имя\_экземпляра в каталогах backups/ и wal/ каталога резервных копий. Каталог backups/имя\_экземпляра содержит файл конфигурации pg\_probackup.conf с параметрами pg\_probackup, относящимися к данному экземпляру копии. Если этой команде передать параметры\_удалённого\_режима, они будут добавлены в pg\_probackup.conf.

Настройки управления резервными копиями pg\_probackup хранятся в файле /backup/postgres/pg\_probackup/backups/main\_v16/pg\_probackup.conf

|  |  |  |
| --- | --- | --- |
| Ключ | Значение | Примечание |
| pgdata | /data | Каталог кластера данных PostgreSQL |
| system-identifier | 7415185176128914551 |  |
| xlog-seg-size | 16777216 |  |
| retention-redundancy | 2 | Избыточность хранения резервных копий, достаточных для восстановления |

#### Настройка непрерывного архивирования WAL

Для выполнения копирования в режиме PAGE, восстановления на момент времени, создания резервных копий с использованием режима доставки WAL ARCHIVE и восстановления инкрементальной копии после смены линии времени, должно осуществляться непрерывное архивирование WAL. Чтобы настроить непрерывное архивирование, выполните следующие действия:

* Задать для параметра wal\_level значение replica.
* Задать для параметра archive\_mode значение on.
* Установить параметр archive\_command:

archive\_command = '/opt/postgres/16/bin/pg\_probackup archive-push -B /backup/postgres/pg\_probackup --compress-algorithm=zlib --compress-level=9 --instance=main\_v16 --wal-file-path=%p --wal-file-name=%f'

Здесь параметры:

-B - указывает на каталог, предназначенный для хранения резервных копий

--compress-algorithm – указывает на алгоритм сжатия WAL.

--compress-level уровень сжатия от 0 до 9

--instance - имя подкаталогов, в которых будут храниться файлы копируемых данных и WAL для этого кластера.

--wal-file-path указывает на диреткорию с WAL сегментами, используется переменная %p указывающая на каталог pg\_wal

--wal-file-name переменная %f указывает на текущий WAL сегмент

#### Настройки параметров восстановления

Для восстановления из резевной копии необходимо выставить параметры в файле postgresql.conf:

* restore\_command = '/opt/postgres/16/bin/pg\_probackup archive-get -B /backup/postgres/pg\_probackup --instance=main\_v16 --wal-file-path=%p --wal-file-name=%f'

Здесь параметры:

-B - указывает на каталог, предназначенный для хранения резервных копий

--compress-algorithm – указывает на алгоритм сжатия WAL.

--compress-level уровень сжатия от 0 до 9

--instance - имя подкаталогов, в которых будут храниться файлы копируемых данных и WAL для этого кластера.

--wal-file-path указывает на директорию с WAL сегментами, используется переменная %p указывающая на каталог архивными сегментами WAL

--wal-file-name переменная %f указывает на текущий WAL сегмент

* recovery\_target\_action = 'promote'

Указывает, какое действие должен предпринять сервер после достижения цели восстановления. Вариант по умолчанию — pause, что означает приостановку восстановления. Второй вариант, promote, означает, что процесс восстановления завершится и сервер начнёт принимать подключения. Наконец, с вариантом shutdown сервер остановится, как только цель восстановления будет достигнута.

#### О резервных копиях

Для создания резервной копии необходимо выполнить следующую команду:

pg\_probackup backup -B каталог\_копий --instance имя\_экземпляра -b режим\_копирования

Здесь режим\_копирования может быть следующим:

* FULL — создаётся полная резервная копия, содержащая все файлы данных кластера, необходимые для его восстановления.
* PAGE — создаётся инкрементальная резервная копия с файлами WAL, записанными после предыдущей полной или инкрементальной копии. Из файлов данных при этом считываются только изменённые страницы.

При восстановлении кластера из инкрементальной копии pg\_probackup использует родительскую полную копию и все инкрементальные копии между ними, которые в совокупности образуют «цепочку копий». Таким образом, прежде чем делать инкрементальные копии, необходимо сделать как минимум одну полную.

#### Создание полной резервной копии

Полная резервная копия содержит все файлы данных кластера, необходимые для его восстановления. Для выполнения полного резервного копирования используется команда:

*pg\_probackup backup -B /backup/postgres/pg\_probackup -b FULL -j 4 --stream --compress-algorithm=zlib --compress-level=9 -E /home/postgres/bin --delete-expired --delete-wal --log-level-file=info --log-directory=/backup/postgres/log --log-filename=main\_v16\_full\_$(date +%Y%m%d%H%M).log --instance main\_v16*

Здесь параметры:

-B - указывает на каталог, предназначенный для хранения резервных копий

-b указывает тип резервной копии. Для полной резервной копии значение FULL

-j количество потоков, создающих резервную копию

--stream создаёт потоковую резервную копию, включая в неё все необходимые файлы WAL, получаемые от сервера по протоколу репликации

--compress-algorithm – указывает на алгоритм сжатия WAL.

--compress-level уровень сжатия от 0 до 9

-E =путь\_внешнего\_каталога - включает в создаваемую копию указанный каталог, рекурсивно копируя его содержимое в отдельный подкаталог каталога резервной копии. Этот параметр полезен для архивирования скриптов, SQL-дампов и файлов конфигурации, расположенных вне каталога данных. Если вы хотите архивировать несколько внешних каталогов, их пути нужно разделять двоеточием

-- delete-expired удаляет резервные копии, не удовлетворяющие политике сохранения, определённой в файле конфигурации pg\_probackup.conf.

--delete-wal удаляет файлы WAL, которые не являются необходимыми для восстановления кластера из имеющихся резервных копий.

--log-level-file управляет уровнем сообщений, которые будут выводиться в файл журнала. Допустимые уровни: verbose, log, info, warning, error и off. Каждый уровень включает все последующие, и с каждым последующим уровнем объём сообщений уменьшается. Вариант off отключает вывод в файл журнала.

--log-directory определяет каталог, в котором будут создаваться файлы журналов. Вы должны задать в этом параметре абсолютный путь. Этот каталог создаётся только при необходимости, когда в журнал выводится первое сообщение. По умолчанию: $BACKUP\_PATH/log/

--log-filename определяет имена только для файлов журналов ошибок. Имена файлов обрабатываются по шаблону strftime, так что вы можете использовать спецкоды с % для выбора имён файлов, зависящих от времени.

--instance - имя подкаталогов, в которых будут храниться файлы копируемых данных и WAL для этого кластера.

#### Создание инкрементальной резервной копии

При создании инрементальной резервной копии, создаётся копия с файлами WAL, записанными после предыдущей полной или инкрементальной копии. Из файлов данных при этом считываются только изменённые страницы

Для выполнения полного резервного копирования используется команда:

*pg\_probackup backup -B /backup/postgres/pg\_probackup -b PAGE -j 4 --stream --compress-algorithm=zlib --compress-level=9 -E /home/postgres/bin --delete-wal --log-level-file=info --log-directory=/backup/postgres/log --log-filename=main\_v16\_full\_$(date+%Y%m%d%H%M).log --instance main\_v16*

Здесь параметры:

-B - указывает на каталог, предназначенный для хранения резервных копий

-b указывает тип резервной копии. Для инкрементальной резервной копии значение PAGE

-j количество потоков, создающих резервную копию

--stream создаёт потоковую резервную копию, включая в неё все необходимые файлы WAL, получаемые от сервера по протоколу репликации

--compress-algorithm – указывает на алгоритм сжатия WAL.

--compress-level уровень сжатия от 0 до 9

--delete-wal удаляет файлы WAL, которые не являются необходимыми для восстановления кластера из имеющихся резервных копий.

--log-level-file управляет уровнем сообщений, которые будут выводиться в файл журнала. Допустимые уровни: verbose, log, info, warning, error и off. Каждый уровень включает все последующие, и с каждым последующим уровнем объём сообщений уменьшается. Вариант off отключает вывод в файл журнала.

--log-directory определяет каталог, в котором будут создаваться файлы журналов. Вы должны задать в этом параметре абсолютный путь. Этот каталог создаётся только при необходимости, когда в журнал выводится первое сообщение. По умолчанию: $BACKUP\_PATH/log/

--log-filename определяет имена только для файлов журналов ошибок. Имена файлов обрабатываются по шаблону strftime, так что вы можете использовать спецкоды с % для выбора имён файлов, зависящих от времени.

--instance - имя подкаталогов, в которых будут храниться файлы копируемых данных и WAL для этого кластера.

#### Проверка резервных копий

pg\_probackup вычисляет контрольные суммы для всех файлов копии в ходе резервного копирования. Процесс проверки контрольных сумм файлов называется проверкой целостности копии. По умолчанию проверка выполняется сразу после создания резервной копии и непосредственно перед восстановлением для выявления возможных повреждений резервных копий.

Чтобы убедиться, что все необходимые файлы резервных копий имеются в наличии и что, используя их, можно восстановить кластер баз данных, вы можете запустить команду validate с теми же параметрами точки восстановления, с которыми вы будете производить восстановление.

Например, чтобы убедиться, что вы можете восстановить кластер баз данных из резервной копии, остановившись на транзакции с идентификатором 4242, выполните команду:

pg\_probackup validate -B каталог\_копий --instance имя\_экземпляра --recovery-target-xid=4242

Если проверка проходит успешно, pg\_probackup выдаёт сообщение об этом. В случае же неудачи вы получите сообщение об ошибке с указанием точного времени, идентификатора транзакции и значения LSN, до которого возможно восстановление.

#### Просмотр информации о резервных копиях

Чтобы просмотреть список существующих копий для каждого экземпляра, выполните команду:

pg\_probackup show -B каталог\_копий

команда выводит список всех имеющихся резервных копий. Например:

pg\_probackup show -B /backup/postgres/pg\_probackup

BACKUP INSTANCE 'main\_v16'

====================================================================================================================================

Instance Version ID Recovery Time Mode WAL Mode TLI Time Data WAL Zratio Start LSN Stop LSN Status

====================================================================================================================================

main\_v16 16 SIZ9H1 2024-08-29 16:34:17+05 FULL STREAM 1/0 12s 57MB 32MB 3.48 0/24000028 0/24000168 OK

main\_v16 16 SHQTSP 2024-08-05 16:41:15+05 FULL STREAM 1/0 11s 7477kB 16MB 3.05 0/3000028 0/30001A0 OK

Для каждой копии выдаются следующие сведения:

* Instance — имя экземпляра.
* Version — базовая версия Postgres Pro.
* ID — идентификатор резервной копии.
* Recovery time — самое ранее время, на которое можно восстановить кластер из данной копии.
* Mode — режим, в котором была сделана копия. Возможные значения: FULL (полная), PAGE (страничная).
* WAL Mode — режим доставки WAL. Возможные значения: STREAM (потоковый) и ARCHIVE (архивный).
* TLI — идентификаторы линии времени текущей копии и её родителя.
* Time — время, за которое была выполнена данная копия.
* Data — объём файлов данных в этой копии. Это значение не включает в себя объём файлов WAL. Для копий, сделанных в режиме STREAM, общий размер можно рассчитать, сложив значения Data и WAL.
* WAL — размер несжатых файлов WAL, которые должны быть применены в процессе восстановления копии для достижения согласованного состояния.
* Zratio — коэффициент сжатия, вычисленный как отношение «uncompressed-bytes» (объём несжатых данных в байтах) к «data-bytes» (итоговый объём данных).
* Start LSN — последовательный номер в журнале WAL, соответствующий началу процесса копирования. С этой позиции накатываются изменения (REDO) в процессе восстановления Postgres Pro.
* Stop LSN — последовательный номер в журнале WAL, соответствующий окончанию процесса копирования. Это позиция точки согласованности при восстановлении Postgres Pro.
* Status — состояние резервной копии. Возможные варианты:
* OK — резервная копия сделана и пригодна к использованию.
* DONE — резервная копия сделана, но не проверена.
* RUNNING — резервное копирование выполняется.
* MERGING — резервная копия объединяется.
* MERGED — файлы резервной копии были успешно обработаны в процессе объединения копий, но её метаданные ещё изменяются. Это состояние могут иметь только полные резервные копии.
* DELETING — файлы резервной копии удаляются.
* CORRUPT — некоторые файлы резервной копии повреждены.
* ERROR — резервное копирование было прервано из-за неожиданной ошибки.
* ORPHAN — резервная копия непригодна к использованию, так как её родительская копия испорчена или отсутствует.

Восстановить кластер из копии можно только для копий с состоянием OK или DONE.

### Восстановление из резервной копии

Чтобы восстановить кластер баз данных из резервной копии, выполните команду restore как минимум со следующими параметрами:

*pg\_probackup restore -B каталог\_копий --instance имя\_экземпляра -i ид\_резервной\_копии*

Здесь:

* каталог\_копий — каталог, в котором хранятся все файлы резервных копий и метаданные.
* имя\_экземпляра — имя экземпляра резервной копии кластера, которая будет восстановлена.
* ид\_резервной\_копии определяет, из какой резервной копии будет восстановлен кластер. Если этот параметр опускается, pg\_probackup использует последнюю подходящую копию для заданного экземпляра. Если вы выбираете для восстановления инкрементальную копию, pg\_probackup автоматически восстанавливает нижележащую полную копию и затем последовательно применяет все необходимые добавления.

#### Выполнение восстановления на момент времени (PITR)

Если вы настраивали непрерывное архивирование WAL до создания резервных копий, вы можете восстановить состояние кластера на любой момент времени (до заданной точки восстановления), используя с командой restore параметры точки восстановления.

Для восстановления на момент времени может использоваться копия типа STREAM или ARCHIVE, но при этом обязательно наличие архива WAL с момента создания копии или более раннего. Если параметр -i/--backup-id не задан, pg\_probackup автоматически выбирает резервную копию, ближайшую к заданной цели восстановления, и начинает процесс восстановления. В противном случае pg\_probackup попытается восстановить до заданной цели восстановления именно копию с заданным ид\_резервной\_копии.

* Чтобы восстановить состояние кластера на определённый момент времени, укажите это время в параметре --recovery-target-time, в формате timestamp. Например:

*pg\_probackup restore -B каталог\_копий --instance имя\_экземпляра --recovery-target-time='2017-05-18 14:18:11+03'*

* Чтобы восстановить состояние кластера до определённой транзакции, воспользуйтесь ключом --recovery-target-xid:

*pg\_probackup restore -B каталог\_копий --instance имя\_экземпляра --recovery-target-xid=687*

* Чтобы восстановить состояние кластера до определённой позиции в журнале (LSN), воспользуйтесь ключом --recovery-target-lsn:

*pg\_probackup restore -B каталог\_копий --instance имя\_экземпляра --recovery-target-lsn=16/B374D848*

* Чтобы восстановить состояние кластера до заданной именованной точки восстановления, воспользуйтесь ключом --recovery-target-name:

*pg\_probackup restore -B каталог\_копий --instance имя\_экземпляра --recovery-target-name='before\_app\_upgrade'*

* Чтобы восстановить последнее возможное состояние, исходя из содержимого архива WAL, передайте в параметре --recovery-target значение latest:

*pg\_probackup restore -B каталог\_копий --instance имя\_экземпляра --recovery-target='latest'*

* Чтобы восстановить самое раннее из возможных согласованное состояние кластера, передайте в параметре --recovery-target значение immediate:

*pg\_probackup restore -B каталог\_копий --instance имя\_экземпляра --recovery-target='immediate'*

#### Процедура восстановления

Для восстановления резервной копии требуется выполнить следующую последовательность действий

1. Остановить сервер баз данных, если он запущен, с помощью команды:

sudo systemctl stop postgresql-16

2. Если поволяет, скопировать весь текущий каталог кластера баз данных и все табличные пространства во временный каталог на случай, если они понадобятся. Следует учесть, что эта мера предосторожности требует, чтобы свободного места на диске было достаточно для размещения двух копий существующих данных. Если места недостаточно, необходимо сохранить как минимум содержимое подкаталога pg\_wal каталога кластера, так как он может содержать журналы, не попавшие в архив перед остановкой системы.

3. Удалить все существующие файлы и подкаталоги из каталога кластера и из корневых каталогов используемых табличных пространств.

4. Выполнить необходимую команду pg\_probackup c опцией restore (см. описание выше в «Выполнение восстановления на момент времени (PITR)»)

5. Если на шаге 2 были сохранены незаархивированные файлы с сегментами WAL, скопировать их в pg\_wal/. (Лучше всего именно копировать, а не перемещать их, чтобы остались неизменённые файлы на случай, если возникнет проблема и всё придётся начинать сначала.)

6. Запустить сервер, если параметр recovery\_target не был вставлен в promote, с помощью команды:

sudo systemctl start postgresql-16

7. Просмотреть содержимое базы данных, чтобы убедиться, что она вернулась к желаемому состоянию. Если это не так, вернуться к шагу 1. Если всё хорошо, разрешить пользователям подключаться к серверу.

### Мониторинг работы сервера базы данных PostgreSQL

Для мониторинга работоспособности базы данных PostgreSQL можно использовать следующие способы:

#### Стандартные инструменты Linux

Проверку состояния сервиса PostgreSQL, можно просмотреть используя утилиту systemctl:

sudo systemctl status postgresql-16

Команда Linux «ps» позволяет получить список процессов с параметрами запуска/выполнения. Пример использования команды:

**ps -ef|grep postgres**

postgres@DBServer-01:/data$ ps -ef |grep postgres

postgres 93776 1 0 Aug09 ? 00:05:03 /opt/postgres/16/bin/postgres -D /data

postgres 93777 93776 0 Aug09 ? 00:05:26 postgres: logger

postgres 93778 93776 0 Aug09 ? 00:03:25 postgres: checkpointer

postgres 93779 93776 0 Aug09 ? 00:00:31 postgres: background writer

postgres 93781 93776 0 Aug09 ? 00:04:53 postgres: walwriter

postgres 93782 93776 0 Aug09 ? 00:01:12 postgres: autovacuum launcher

postgres 93783 93776 0 Aug09 ? 00:00:01 postgres: logical replication launcher

postgres 93817 93776 0 Aug09 ? 00:00:33 postgres: tal appName\_preprod 192.168.1.1(37562) idle

В данном примере процесс 93776 Postmaster является родительским для всех процессов PostgreSQL, так же указывает параметры запуска, такие, как директорию данных кластера.

Процессы с 93776 93783 фоновые процессы PostgreSQL

Процессы с 93817 93776 процессы клиентских подключений к базе

#### Представления PostgreSQL для просмотра состояния

Следующие представления позволяют получить информацию о статусе базы данных PostgreSQL

|  |  |
| --- | --- |
| Представление | Описание |
| pg\_stat\_activity | Одна строка для каждого серверного процесса c информацией по текущей активности процесса, такой как состояние и текущий запрос |
| pg\_stat\_replication | По одной строке для каждого процесса-передатчика WAL со статистикой по репликации на ведомом сервере, к которому подключён этот процесс. Выполняется на Primary сервере |
| pg\_stat\_wal\_receiver | Возвращается только одна строка со статистикой приёмника WAL, полученной с сервера, на котором работает приёмник. Выполняется на Standby экземпляре |
| pg\_locks | Представление содержит одну строку для каждого активного блокируемого объекта, запрошенного режима блокировки и блокирующего процесса. |
| pg\_stat\_archiver | Только одна строка со статистикой о работе активности процесса архивации WAL |
| pg\_stat\_bgwriter | Только одна строка со статистикой о работе фонового процесса записи |
| pg\_stat\_database | Одна строка для каждой базы данных со статистикой на уровне базы |
| pg\_stat\_all\_tables | По одной строке на каждую таблицу в текущей базе данных со статистикой по обращениям к этой таблице |
| pg\_stat\_all\_indexes | По одной строке для каждого индекса в текущей базе данных со статистикой по обращениям к этому индексу |

### Служебные скрипты PostgreSQL

В домашней директории пользователя postgres /home/postgres/bin находятся различные скрипты для обслуживания экземпляра.

|  |  |
| --- | --- |
| Скрипт | Примечание |
| /home/postgres/bin/probackup\_main\_v16\_full.sh | Выполнение полной резервной копии pg\_probackup в 23:00 каждый понедельник, четверг |
| /home/postgres/bin/probackup\_main\_v16\_page.sh | Выполнение инкрементальной резервной копии pg\_probackup в 23:00 каждый кроме понедельника, четверга |

Содержимое скрипта probackup\_main\_v16\_full.sh

------------------------------------------------------------------------------------------------------------------------------

#!/bin/bash

PGDATA=/data

BACKUP\_PATH=/backup/postgres/pg\_probackup

CURRENT\_TIME="$(date +%Y%m%d%H%M)"

PGLOG=/backup/postgres/log

LD\_LIBRARY\_PATH=/opt/postgres/16/lib:$LD\_LIBRARY\_PATH

PATH=/opt/postgres/16/bin:$PATH:$HOME/bin

pg\_probackup backup -B $BACKUP\_PATH -b FULL -j 4 --stream --compress-algorithm=zlib --compress-level=9 -E /home/postgres/bin --delete-expired --delete-wal --log-level-file=info --log-directory=$PGLOG --log-filename=main\_v16\_full\_$CURRENT\_TIME.log --instance main\_v16

pg\_probackup show -B $BACKUP\_PATH >> $PGLOG/main\_v14\_full\_$CURRENT\_TIME.log

find $PGLOG -mtime +10 -exec rm -f {} \;

-----------------------------------------------------------------------------------------------------------------------

Содержимое скрипта probackup\_main\_v16\_page.sh

--------------------------------------------------------------------------------------------------------------------

#!/bin/bash

PGDATA=/data

BACKUP\_PATH=/backup/postgres/pg\_probackup

CURRENT\_TIME="$(date +%Y%m%d%H%M)"

PGLOG=/backup/postgres/log

LD\_LIBRARY\_PATH=/opt/postgres/16/lib:$LD\_LIBRARY\_PATH

PATH=/opt/postgres/16/bin:$PATH:$HOME/bin

pg\_probackup backup -B $BACKUP\_PATH -b PAGE -j 4 --stream --compress-algorithm=zlib --compress-level=9 -E /home/postgres/bin --delete-wal --log-level-file=info --log-directory=$PGLOG --log-filename=main\_v16\_page\_$CURRENT\_TIME.log --instance main\_v16

pg\_probackup show -B $BACKUP\_PATH >> $PGLOG/main\_v14\_page\_$CURRENT\_TIME.log

find $PGLOG -mtime +10 -exec rm -f {} \;

-------------------------------------------------------------------------------------------------------------------------------

### Автоматические задачи планировщика операционной системы (cron)

Запланированные задачи в cron пользователя postgres.

|  |  |  |
| --- | --- | --- |
| Расписание | Задача | Примечание |
| 0 23 \* \* 1,4 | /home/postgres/bin/probackup\_main\_v16\_full.sh | Выполнение полной резервной копии pg\_probackup в 23:00 каждый понедельник, четверг |
| 0 23 \* \* 2,3,5,6,7 | /home/postgres/bin/probackup\_main\_v16\_page.sh | Выполнение инкрементальной резервной копии pg\_probackup в 23:00 каждый кроме понедельника, четверга |
| 0 22 \* \* \* | /usr/bin/find /data/log/\* -mtime +10 -exec rm -f {} \; | Ежедневная очистка устаревших логов PostgreSQL в 22:00 |

## Уровень сервера приложений Apache Tomcat

### Общие сведения

#### Корпоративная среда размещения

Сервер приложений Apache Tomcat 9.0.67 установлен в не отказоустойчивой конфигурации на одном сервере, с использованием двух сервисов Tomcat, на который будет осуществляться перенаправление при помощи HAproxy. Сервер Tomcat установлен на одном сервере с базой данных PostgreSQL.

|  |  |  |
| --- | --- | --- |
| Сервер | IP адрес | Роль сервера |
|  |  | Active |

#### Параметры операционной системы

Для Apache Tomcat 9.0.67 выделен сервер со следующими характеристиками:

* Debian 12
* ОЗУ 17 Гб
* 8 ядер - Intel(R) Xeon(R) Gold 6258R CPU @2.70GHz

#### Файловая система

На сервере установлены следующие файловые системы:

|  |  |  |  |
| --- | --- | --- | --- |
| Устройство | Точка монтирования | Размер | Файловая система |
| /dev/sda1 | /boot/efi | 511M | vfat |
| /dev/sda2 | /boot | 456M | Ext2 |
| /dev/mapper/AppServer-01--vg-root | / | 96G | Ext4 |
| /swapfile | [SWAP] | 1G | swap |

#### Пользователь операционной системы

Владельцем установки Apache Tomcat является пользователь tomcat. Пользователь tomcat добавлен в /etc/sudoers для возможности выполнять команды запуска/останова/перезапуска сервисов, с правами привилегированного пользователя используя команды sudo.

Добавленные записи в /etc/sudoers

|  |
| --- |
| Описание |
| Cmnd\_Alias SERVICES = /usr/bin/systemctl start [A-z0-9]\*, /usr/bin/systemctl stop [0-9A-z]\*, /usr/bin/systemctl reload [A-z0-9]\*, /usr/bin/systemctl restart [A-z0-9]\*, /usr/bin/systemctl status [A-z0-9]\* |
| tomcat ALL = NOPASSWD:SERVICES |

#### Переменные окружения пользователя tomcat

В файле .profile и в tomcat1.service прописаны следующие переменные окружения пользователя tomcat

|  |  |
| --- | --- |
| Переменная | Значение |
| CATALINA\_HOME | /opt/tomcat/apache-tomcat-9 |

#### Основные директории Apache Tomcat

Директории, необходимые для работы серверов приложений CUBA

|  |  |
| --- | --- |
| Директория | Описание |
| /opt/tomcat | Корневая установка Apache Tomcat |
| /opt/tomcat/apache-tomcat-9 | Бинарные файлы Apache Tomcat (библиотеки, исполняемые файлы, конфигурация). Переменная окружения $CATALINA\_HOME |
| /opt/tomcat/apache-tomcat-9/lib | Директория библиотек Apache Tomcat. |
| /opt/tomcat/tomcat1/filestorage | Директоия для хранения подгружаемых файлов. |
| /opt/tomcat/ | Директории для запуска виртуальной машины JAVA (сервер Tomcat) и опубликованного приложения в данной виртуальной машине. На этих серверах разворачивается уровень Web Client и Middleware приложений, взаимодействующих с пользователями. Далее для каждой директории используется переменная $CATALINA\_BASE |
| $CATALINA\_BASE/conf | Директория конфигурации сервера Tomcat |
| $CATALINA\_BASE/fonts | Директория дополнительных шрифтов сервера Tomcat |
| $CATALINA\_BASE/logs | Директория хранения логов сервера Tomcat |
| $CATALINA\_BASE/webapps | Директория приложений сервера Tomcat |

#### Директория JDK, используемые Apache Tomcat

Apache Tomcat использует открытую Java от Amazon

|  |  |
| --- | --- |
| Директория | Версия |
| /opt/java/amazon-corretto-8 | amazon-corretto-8.422.05.1 |

#### Скрипты systemd для запуска/останова сервисов Apache Tomcat

На серверах настроены скрипты запуска systemd, которые хранятся в директории /usr/lib/systemd/system/

|  |  |
| --- | --- |
| Скрипт systemd | Описание |
| tomcat1.service | Скрипт запуска/останова сервисов уровня Web Client и Middleware |

#### Содержимое файла /usr/lib/systemd/system/.service

[Unit]

Description=Apache Tomcat Web Application Container

After=syslog.target network.target postgresql.service

[Service]

Type=forking

Environment=CATALINA\_BASE=/opt/tomcat/tomcat1

Environment=CATALINA\_HOME=/opt/tomcat/apache-tomcat-9

Environment=CATALINA\_PID=$CATALINA\_BASE/temp/tomcat.pid

Environment=JAVA\_HOME=/opt/java/amazon-corretto-8

Environment='CATALINA\_OPTS=-Xms512M -Xmx1024M -server -XX:+UseParallelGC -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=10001 -Djava.rmi.server.hostname=localhost -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false -Djavax.xml.parsers.DocumentBuilderFactory=com.sun.org.apache.xerces.internal.jaxp.DocumentBuilderFactoryImpl -Djavax.xml.transform.TransformerFactory=com.sun.org.apache.xalan.internal.xsltc.trax.TransformerFactoryImpl -Dorg.xml.sax.parser=com.sun.org.apache.xerces.internal.parsers.SAXParser -Djavax.xml.parsers.SAXParserFactory=com.sun.org.apache.xerces.internal.jaxp.SAXParserFactoryImpl -agentlib:jdwp=transport=dt\_socket,server=y,suspend=n,address=11001'

Environment='JAVA\_OPTS=-Dapp.home=$CATALINA\_BASE/webapps -Djava.awt.headless=true -Djava.security.egd=file:/dev/./urandom -Dlogback.configurationFile=$CATALINA\_BASE/conf/logback.xml -Djava.util.logging.config.file=$CATALINA\_BASE/conf/logging.properties -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -Dfile.encoding=UTF-8'

ExecStartPre=/bin/sleep 10

ExecStart=/opt/tomcat/apache-tomcat-9/bin/catalina.sh start

ExecStop=/opt/tomcat/apache-tomcat-9/bin/catalina.sh stop

User=tomcat

Group=tomcat

UMask=0007

[Install]

WantedBy=multi-user.target

----------------------------------------------------------------------------

#### Используемые порты Apache Tomcat

Во время запуска сервиса Apache Tomcat, автоматически резервируются порты, указанные в systemd скриптах, в параметрах запуска java CATALINA\_OPTS, так и указанные в файле $CATALINA\_BASE/conf/server.xml

*Порты сервисов Tomcat:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Сервис | HTTP | Shutdown | JMX | Debug |
| tomcat1.service |  |  |  |  |

#### Установленные приложения Apache Tomcat

Приложения установлены на сервере AppServer-01.company.com. На каждом сервисе Tomcat одно приложение. Путь приложений - CATALINA\_BASE/webapps

*Установленные приложения:*

|  |  |  |
| --- | --- | --- |
| Сервис | Уровень приложения | WAR file |
|  | Web Client  Middleware | .war .war |

### Конфигурация сервера Apache Tomcat

#### Дополнительно установленные библиотеки Apache Tomcat

На сервере в директорию $CATALINA\_HOME/lib добавлены следующие библиотеки:

h2-1.2.131.jar  
hsqldb-2.3.2.jar  
mysql-connector-java-5.1.17.jar  
postgresql-42.7.4.jar  
jcifs.jar

#### Файл web.xml уровня Apache Tomcat

Файл web.xml является стандартным дескриптором веб-приложения Java EE, и должен быть создан для блоков Middleware, Web Client.

На сервере AppServer-01.company.com

добавлена запись в $CATALINA\_HOME/conf/web.xml

<context-param>

<description>Vaadin production mode</description>

<param-name>productionMode</param-name>

<param-value>true</param-value>

</context-param>

позволяющая работать серверу в режиме продуктивного сервера.

#### Опции логирования сервисов Apache Tomcat

На сервере настроено логирование сервисов Apache Tomcat следующим образом. В файле $CATALINA\_BASE/conf/logging.properties, который является описателем конфигурации логирования самого сервера Tomcat, выставлены значения для событий логирования внутренних процессов Apache Tomcat.

#### Содержимое файла $CATALINA\_BASE/conf/logging.properties

handlers = 1catalina.org.apache.juli.FileHandler, 2localhost.org.apache.juli.FileHandler, java.util.logging.ConsoleHandler

.handlers = 1catalina.org.apache.juli.FileHandler, java.util.logging.ConsoleHandler

1catalina.org.apache.juli.FileHandler.level = FINE

1catalina.org.apache.juli.FileHandler.directory = ${catalina.base}/logs

1catalina.org.apache.juli.FileHandler.prefix = catalina.

2localhost.org.apache.juli.FileHandler.level = FINE

2localhost.org.apache.juli.FileHandler.directory = ${catalina.base}/logs

2localhost.org.apache.juli.FileHandler.prefix = localhost.

java.util.logging.ConsoleHandler.level = FINE

java.util.logging.ConsoleHandler.formatter = org.apache.juli.OneLineFormatter

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].level = INFO

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].handlers = 2localhost.org.apache.juli.FileHandler

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Использование этого файла указывается в параметрах запуска сервера CATALINA\_OPTS, см. раздел **Запуск/останов серверов Apache Tomcat**

Для логирования событий приложений Web Client, Web Portal и Middleware был создан файл $CATALINA\_HOME/conf/logback.xml в котором определяется конфигурация логирования.

Элементы appender файла logback.xml, задают "устройства вывода" логов. Основными аппендерами являются FILE и CONSOLE. В параметре level элемента filter можно задать порог уровня сообщения. По умолчанию порог для файла - DEBUG, для консоли - INFO. Это означает, что в файл выводятся сообщения с уровнями ERROR, WARN, INFO, DEBUG, а в консоль - с уровнями ERROR, WARN и INFO.

Для файлового аппендера в параметре file задается путь к файлу лога. По умолчанию это файл $CATALINA\_BASE/logs/app.log.

Элементы logger задают параметры логгеров, через которые производится посылка сообщений из кода программы. Имена логгеров иерархические, то есть например настройки для логгера com.company.sample влияют на логгеры com.company.sample.core.CustomerServiceBean, com.company.sample.web.CustomerBrowse, если для них явно не заданы собственные настройки.

Минимальный уровень указывается в атрибуте level. Например, если для логгера задан приоритет INFO, то сообщения с уровнями DEBUG и TRACE выводиться не будут. Следует иметь в виду, что на вывод сообщения также влияет порог уровня, заданный в аппендере.

Оперативно изменять уровни для логгеров и пороги аппендеров для работающего сервера можно с помощью экрана Administration > Server Log, доступного в веб-клиенте. Сделанные настройки логирования действуют только в текущем сеансе работы сервера и в файл не сохраняются.

#### Содержимое файла $CATALINA\_HOME/conf/logback.xml

<?xml version="1.0" encoding="UTF-8"?>

<configuration debug="false" packagingData="true">

<property name="logDir" value="${catalina.base}/logs"/>

<appender name="File" class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>${logDir}/app.log</file>

<filter class="ch.qos.logback.classic.filter.ThresholdFilter">

<level>DEBUG</level>

</filter>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<!-- daily rollover -->

<fileNamePattern>${logDir}/app.%d{yyyy-MM-dd}.log</fileNamePattern>

<!-- keep 30 days' worth of history -->

<maxHistory>30</maxHistory>

<cleanHistoryOnStart>true</cleanHistoryOnStart>

</rollingPolicy>

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level [%thread%X{cubaApp}%X{cubaUser}] %logger - %msg%n</pattern>

</encoder>

</appender>

<appender name="Console" class="ch.qos.logback.core.ConsoleAppender">

<filter class="ch.qos.logback.classic.filter.ThresholdFilter">

<level>INFO</level>

</filter>

<encoder>

<pattern>%d{HH:mm:ss.SSS} %-5level %-40logger{36}- %msg%n</pattern>

</encoder>

</appender>

<root>

<appender-ref ref="Console"/>

<appender-ref ref="File"/>

</root>

<!-- Begin CUBA -->

<logger name="com.haulmont.cuba" level="DEBUG"/>

<logger name="com.haulmont.cuba.core.sys" level="INFO"/>

<logger name="com.haulmont.cuba.core.sys.CubaDefaultListableBeanFactory" level="WARN"/>

<logger name="com.haulmont.cuba.core.app.scheduling" level="INFO"/>

<logger name="com.haulmont.cuba.web.sys" level="INFO"/>

<logger name="com.haulmont.cuba.portal" level="INFO"/>

<logger name="com.haulmont.restapi.sys" level="INFO"/>

<logger name="com.haulmont.cuba.core.app.LockManager" level="INFO"/>

<!-- End CUBA -->

<logger name="eclipselink" level="WARN"/>

<logger name="eclipselink.sql" level="INFO"/>

<logger name="org.springframework" level="WARN"/>

<logger name="com.vaadin" level="WARN"/>

<logger name="org.atmosphere" level="WARN"/>

<logger name="org.activiti" level="INFO"/>

<logger name="org.jgroups" level="INFO"/>

<logger name="freemarker" level="INFO"/>

<logger name="org.thymeleaf.TemplateEngine" level="INFO"/>

<logger name="org.docx4j" level="WARN"/>

<logger name="org.xlsx4j" level="WARN"/>

<logger name="org.apache.fop.apps.FOUserAgent" level="WARN"/>

<logger name="org.hibernate" level="WARN"/>

<logger name="sun" level="INFO"/>

<logger name="com.sun" level="INFO"/>

<logger name="javax" level="INFO"/>

<logger name="org.apache" level="INFO"/>

<logger name="org.eclipse.jetty" level="INFO"/>

<logger name="org.docx4j.utils.ResourceUtils" level="ERROR"/>

<logger name="org.docx4j.Docx4jProperties" level="ERROR"/>

<logger name="org.xlsx4j.jaxb.Context" level="ERROR"/>

<logger name="org.docx4j.utils.XSLTUtils" level="ERROR"/>

<logger name="org.docx4j.jaxb.JaxbValidationEventHandler" level="ERROR"/>

<logger name="org.docx4j.TraversalUtil" level="ERROR"/>

<logger name="org.docx4j.fonts" level="ERROR"/>

<!-- Begin Perf4J -->

<appender name="PerfStatFile" class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>${logDir}/perfstat.log</file>

<append>true</append>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>${logDir}/perfstat.%d{yyyy-MM-dd}.log</fileNamePattern>

<maxHistory>30</maxHistory>

<cleanHistoryOnStart>true</cleanHistoryOnStart>

</rollingPolicy>

<encoder>

<pattern>%msg%n</pattern>

</encoder>

</appender>

<appender name="CoalescingStatistics" class="org.perf4j.logback.AsyncCoalescingStatisticsAppender">

<param name="TimeSlice" value="60000"/>

<appender-ref ref="PerfStatFile"/>

</appender>

<appender name="UIPerfStatFile" class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>${logDir}/perfstat-ui.log</file>

<append>true</append>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>${logDir}/perfstat-ui.%d{yyyy-MM-dd}.log</fileNamePattern>

<maxHistory>30</maxHistory>

<cleanHistoryOnStart>true</cleanHistoryOnStart>

</rollingPolicy>

<encoder>

<pattern>%msg%n</pattern>

</encoder>

</appender>

<appender name="UICoalescingStatistics" class="org.perf4j.logback.AsyncCoalescingStatisticsAppender">

<param name="TimeSlice" value="120000"/>

<appender-ref ref="UIPerfStatFile"/>

</appender>

<logger name="org.perf4j.TimingLogger" additivity="false" level="INFO">

<appender-ref ref="CoalescingStatistics"/>

</logger>

<logger name="com.haulmont.cuba.gui.logging.UIPerformanceLogger" additivity="false" level="INFO">

<appender-ref ref="UICoalescingStatistics"/>

</logger>

<!-- End Perf4J -->

</configuration>

-----------------------------------------------------------------------------------------------------------

Использование этого файла указывается в параметрах запуска сервера CATALINA\_OPTS, см. раздел **Запуск/останов серверов Apache Tomcat**

#### Файл context.xml

Файл context.xml является дескриптором развертывания приложения на сервере Apache Tomcat. В развернутом приложении этот файл располагается в подкаталоге META-INF каталога веб-приложения или WAR-файла, например, $CATALINA\_BASE/webapps/app-core/META-INF/context.xml. В проекте файлы данного типа находятся в каталогах /web/META-INF модулей core, web, portal. Файл context.xml, расположенный в $CATALINA\_BASE/conf переопределяет файл приложения, расположенный в $CATALINA\_BASE/webapps/app-core/META-INF/.

Основное предназначение файла для блока Middleware - определить JDBC источник данных и поместить его в JNDI под именем, заданным свойством приложения cuba.dataSourceJndiName.

#### Содержимое файла context.xml

<?xml version='1.0' encoding='utf-8'?>

<Context>

<Resource driverClassName="org.postgresql.Driver"

maxIdle="2"

maxTotal="20"

maxWaitMillis="5000"

name="appNameDS"

password="<password>"

type="javax.sql.DataSource"

url="jdbc:postgresql:// 192.168.1.2: / "

username="tal"/>

<WatchedResource>WEB-INF/web.xml</WatchedResource>

<WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>

</Context>

--------------------------------------------------------------------------------------------------------------

#### Файл server.xml

Файл server.xml — главный файл настроек Tomcat. Элементы server.xml относятся к пяти базовым категориям:

Элементы верхнего уровня (Top Level Elements)

Соединители или коннекторы (Connectors)

Контейнеры (Containers)

Встраиваемые компоненты (Nested Components)

Глобальные настройки (Global Settings)

У всех элементов из этих категорий имеется множество атрибутов, которые позволяют точно определить функциональные возможности. Чаще всего если необходимо внести какие-то существенные изменения в установку Tomcat, как, например, изменить порт, на котором запускается служба Apache Tomcat, приходится редактировать файл server.xml.

*Server*

Этот элемент определяет отдельный сервер Tomcat и содержит элементы конфигурации Logger и ContextManager. К тому же, элемент Server поддерживает атрибуты “port”, “shutdown” и “className”. Атрибут порт используется для того, чтобы уточнить, через какой порт должны выполняться команды shutdown (отключения). Атрибут shutdown задает командную строку для отдельного порта, чтобы спровоцировать отключение. Атрибут className — реализацию класса Java, которая должна использоваться.

*Service*

Это элемент, который можно поместить в элемент Server; он содержит один или несколько компонентов Connector, у которых один общий компонент Engine. Главная функция этого компонента — задать эти компоненты как один сервис. Название сервиса, который будет появляться в логах, определяется с помощью атрибута “name” (элемент Service).

*Connectors*

Размещая один или несколько коннекторов (connector) в теге Service, вы тем самым позволяете системе перенаправить запросы из этих портов в один компонент Engine для обработки. Tomcat позволяет определить коннекторы HTTP и AJP.

*HTTP-* коннектор

Этот элемент представляет HTTP/1.1 Connector и обеспечивает Tomcat автономным функционалом веб-сервера. Это означает, что в дополнение к выполнению сервелатов и JSP -страниц, Tomcat способен прослушивать специфические TCP-порты для запросов.

*Context*

Этот элемент представляет определенное веб-приложение и содержит данные о пути, по которому определяются запросы для соответствующих ресурсов приложения. Tomcat получает запрос и пытается сопоставить самый длинный URI с контекстным путем определенного элемента Context до тех пор, пока не найдется корректный элемент, который бы обслуживал запрос.

*Engine*

Этот элемент используется в связке с одним или несколькими коннекторами, которые размещены в элементе Service. Элемент Engine может использоваться только в случае если он размещен в элементе Service, и только один элемент Engine разрешен в элементе Service. Атрибут jvmRoute элемента позволяет нестроить маршрутизацию для балансировки на уровне HAproxy .

*Host*

Элемент, который размещен в элементе Engine, и используется, чтобы связать названия серверной сети с серверами Tomcat. Этот элемент будет функционировать должным образом только если виртуальный хост был зарегистрирован в системе DNS соответствующего домена.

*Listeners*

Эти элементы можно поместить внутрь элементов Server, Engine, Host или Context. Они указывают на компонент, который производит определенное действие при специфическом событии.

У большинства компонентов есть атрибуты className, чтобы выбрать разные реализации элемента. Существует ряд дополнительных реализаций Listener, не только дефолтных. Все эти реализации требуют, чтобы элемент Listener размещался в определенном элементе Server.

*Global Naming Resources*

Этот элемент используется, чтобы определить ресурсы Java Naming and Directory Interface для специфического Server, отличного от любых контекстов веб-приложения JNDI. Если нужно, вы можете задать характеристики JNDI resource lookup для <resource-ref> и <resource-env-ref> в данном элементе, определив их и связав с помощью <ResourceLink>.

*Realm*

Этот элемент размещается в любом элементе Container и задает базу данных, содержащую имена пользователей, пароли и роли для Container. При размещении внутри элемента Host или Engine, характеристики, заданные в элементе Realm, передаются всем контейнерам нижнего уровня по умолчанию.

Важно корректно установить атрибут “className” этого элемента, поскольку существует множество реализаций. Эти реализации используются, чтобы сделать доступным Catalina другим системам управления безопасностью пользователей (например, JDBC , JNDI или DataSource).

*Resources*

У этого элемента только одно предназначение — направить Tomcat в статические ресурсы, которые используются вашими веб-приложениями. Эти ресурсы включают классы, HTML и JSP файлы. Использование этого элемента предоставляет Tomcat доступ к файлам, содержащимся в других местах, помимо файловой системы (filesystem), таким как ресурсы, которые содержатся в архивах WAR или базах данных JDBC.

*Valve*

Компоненты Valve размещаются внутри элементов Engine, Host и Context, с их помощью добавляются специальные функциональные возможности в конвейер, обрабатывающий запросы. На продуктивном сервере элемент используется для указания формата ведения логов.

#### Содержимое файла /opt/tomcat/tomcat1/conf/server.xml

<?xml version='1.0' encoding='utf-8'?>

<Server port="" shutdown="SHUTDOWN">

<Listener className="org.apache.catalina.startup.VersionLoggerListener" />

<Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />

<Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />

<Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

<Service name="Catalina">

<Connector port="" protocol="HTTP/1.1"

connectionTimeout="20000"

URIEncoding="UTF-8"

redirectPort="8444" />

<Engine name="Catalina" defaultHost="localhost">

<Host name="localhost" appBase="webapps"

unpackWARs="true" autoDeploy="true">

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"

prefix="localhost\_access\_log" suffix=".txt"

pattern="%h %l %u %t &quot;%m %U&quot; %s %b" />

</Host>

</Engine>

</Service>

</Server>

### Свойства приложений

#### Свойства приложения − именованные значения различных типов, определяющие всевозможные аспекты конфигурации и функционирования приложения. Свойства приложения широко используются в платформе, и могут применяться в приложении для решения аналогичных задач.

#### По назначению свойства приложения можно классифицировать следующим образом:

#### - Конфигурационные параметры - задают наборы конфигурационных файлов и некоторые параметры пользовательского интерфейса, т.е. определяют функциональность приложения. Значения конфигурационных параметров обычно задаются при разработке приложения.

#### *Например: cuba.springContextConfig.*

#### - Параметры развертывания - различные URL для соединения блоков приложения, тип используемой БД, настройки безопасности и т.д. Значения параметров развертывания обычно зависят от окружения, в котором устанавливается данный экземпляр приложения.

#### *Например: cuba.connectionUrlList, cuba.dbmsType, cuba.userSessionExpirationTimeoutSec.*

#### - Параметры времени выполнения - активность аудита, параметры отсылки email и т.д. Параметры времени выполнения могут быть изменены при необходимости во время работы приложения без его перезапуска.

#### *Например: cuba.entityLog.enabled, cuba.email.smtpHost.*

#### Значения свойств приложения могут быть заданы в базе данных, в файлах свойств, или через системные свойства Java. Кроме того, значение, заданное в файле, переопределяет одноименное значение, заданное в БД. Значение, заданное системным свойством Java, переопределяет одноименные значения из файлов и из БД.

#### Некоторые свойства не поддерживают установку свойств в базе данных по причине того, что их значения требуются еще то того, как БД становится доступной приложению. Это параметры конфигурации и развертывания. Поэтому их можно устанавливать только в файлах свойств или через системные свойства Java. Параметры времени выполнения всегда могут быть установлены в базе данных (и, возможно, переопределены в файле или системными свойствами).

#### Как правило, некоторое свойство используется только в одном или нескольких блоках приложения. Например, cuba.persistenceConfig необходимо только для Middleware, cuba.web.appWindowMode − только для Web Client, а cuba.springContextConfig − для всех блоков. Это означает, что если нужно задать значение некоторому свойству, это необходимо сделать во всех блоках, в которых данное свойство используется. Свойства, хранящиеся в БД, доступны всем блокам, поэтому они устанавливаются в одном месте (в таблице базы данных), независимо от того, в каких блоках они используются. Более того, платформа предоставляет экран Administration > Application Properties для управления свойствами, хранящимися в БД. Свойства, хранящиеся в файлах, должны быть установлены одновременно в соответствующих файлах блоков приложения.

#### Хранение свойств в файлах

Свойства, определяющие конфигурацию и параметры развертывания, задаются в специальных файлах свойств, имеющих имя вида \*app.properties. Каждый блок приложения имеет набор таких файлов, который определяется следующим образом:

Для блоков, являющихся веб-приложениями (Middleware, Web Client) набор файлов свойств задается в web.xml в параметре appPropertiesConfig.

Например, набор файлов свойств блока Middleware задается в файле web/WEB-INF/web.xml модуля core, и выглядит следующим образом:

<context-param>

<param-name>appPropertiesConfig</param-name>

<param-value>

classpath:com/company/sample/app.properties

/WEB-INF/local.app.properties

"file:${catalina.base}/conf/app-core/local.app.properties"

</param-value>

</context-param>

Здесь префикс classpath: означает, что данный файл нужно искать в Java classpath, префикс file: − в файловой системе. Путь без такого префикса означает путь внутри веб-приложения относительно его корня. Возможно использование системных свойств Java, в данном случае это catalina.home − путь к каталогу установки Tomcat.

Порядок перечисления файлов важен, так как значения, указанные в каждом последующем файле, заменяют значения одноименных свойств, заданные в предыдущих файлах.

Последний файл в приведенном наборе − local.app.properties. Он может использоваться для переопределения свойств приложения при развертывании. Если этого файла нет, он игнорируется. Если же во время инсталляции системы требуется переопределение некоторых параметров (как правило, различных URL), достаточно создать этот файл и поместить в него переопределяемые свойства. При последующих обновлениях системы такой файл с локальными настройками легко сохранить. В разделе Использование Tomcat при эксплуатации приложения приведен пример использования файла local.app.properties.

Правила задания информации в файлах \*.properties:

- Кодировка файла - UTF-8

- Ключ может состоять из латинских букв, цифр, точек и знаков подчеркивания

- Значение пишется после знака равно (=)

- Значение не нужно брать в кавычки " или '

- Файловые пути записываются либо в UNIX-виде (/opt/haulmont/), либо в Windows-виде (c:\\haulmont\\)

- Возможно использование кодов \n \t \r. Символ \ является зарезервированным, для вставки в значение экранируется сам собой (\\). Подробнее см.: http://docs.oracle.com/javase/tutorial/java/data/characters.html

- Для ввода значения в нескольких строках файла используйте символ \ в конце строки, для того чтобы данное значение продолжалось на следующей строке.

#### Параметры файла local.app.properties:

*cuba.logDir*

Конфигурационный параметр, задающий расположение каталога журналов данного блока приложения.

Значение по умолчанию для WAR: ${app.home}/logs, что означает расположение в подкаталоге logs домашнего каталога приложения.

Интерфейс: GlobalConfig

Используется во всех стандартных блоках.

*cuba.confDir*

Конфигурационный параметр, задающий расположение каталога конфигурации данного блока приложения.

Значение по умолчанию для WAR: ${app.home}/${cuba.webContextName}/conf, что означает расположение в подкаталоге домашнего каталога приложения.

Интерфейс: GlobalConfig

Используется во всех стандартных блоках.

*cuba.tempDir*

Конфигурационный параметр, задающий расположение временного каталога данного блока приложения.

Значение по умолчанию для WAR: ${app.home}/${cuba.webContextName}/temp, что означает расположение в подкаталоге домашнего каталога приложения.

Интерфейс: GlobalConfig

Используется во всех стандартных блоках.

*cuba.dataDir*

Конфигурационный параметр, задающий расположение рабочего каталога данного блока приложения.

Значение по умолчанию для WAR: ${app.home}/${cuba.webContextName}/work, что означает расположение в подкаталоге домашнего каталога приложения.

Интерфейс: GlobalConfig

Используется во всех стандартных блоках.

*cuba.download.directories*

Задает список каталогов, из которых можно загружать с Middleware файлы через com.haulmont.cuba.core.controllers.FileDownloadController. Загрузка файлов используется в частности механизмом отображения журналов сервера, доступным через экран Администрирование → Журнал сервера веб-клиента.

Список задается через ";".

Значение по умолчанию: ${cuba.tempDir};${cuba.logDir}, означает что файлы можно загружать из временного каталога и каталога логов.

Используется в блоке Middleware.

*cuba.useLocalServiceInvocation*

При установке данного свойства в true блоки Web Client вызывают сервисы Middleware в обход сетевого стека, что положительно сказывается на производительности системы. Это возможно в случае быстрого развертывания в Tomcat, а также для единого WAR. В других вариантах развертывания данное свойство необходимо установить в false.

Значение по умолчанию: true

Используется в блоках Web Client

*cuba.connectionUrlList*

Задает список URL для подключения клиентских блоков к серверам Middleware.

Значением свойства должен быть один или несколько разделенных запятой URL вида http[s]://host[:port]/app-core, где host - имя сервера, port - порт сервера, app-core - имя веб-приложения, реализующего блок Middleware. Например:

cuba.connectionUrlList=http://localhost:8000/app-core

В случае использования кластера серверов Middleware, для обеспечения отказоустойчивости и балансировки нагрузки необходимо перечислить их адреса через запятую:

cuba.connectionUrlList=http://server1:8000/app-core,http://server2:8000/app-core

Интерфейс: ClientConfig

Используется в блоках Web Client.

*cuba.webHostName*

Конфигурационный параметр, задающий имя хоста, на котором запущен данный блок приложения.

Значение по умолчанию: localhost

Интерфейс: GlobalConfig

Используется в блоках Middleware, Web Client.

Например, для блока Middleware, доступного по URL http://192.168.1.1:8001 /app-core данное свойство должно быть задано следующим образом:

*cuba.webHostName=*192.168.1.1

*cuba.webPort*

Конфигурационный параметр, задающий имя порта, на котором запущен данный блок приложения.

Значение по умолчанию: 8000

Интерфейс: GlobalConfig

Используется в блоках Middleware, Web Client.

Например, для блока Middleware, доступного по URL http://192.168.1.1:8001 /app-core данное свойство должно быть задано следующим образом:

*cuba.webPort=8001*

*cuba.webContextName*

Конфигурационный параметр, задающий имя контекста веб-приложения. Как правило, эквивалентен имени каталога или WAR-файла, содержащего данный блок приложения.

Интерфейс: GlobalConfig

Используется в блоках Middleware, Web Client.

Например, для блока Web Client, расположенного в каталоге $CATALINA\_BASE/webapps/appName, и доступного по URL http://192.168.1.1:8001/appName данное свойство должно быть задано следующим образом:

*cuba.webContextName=appName*

*cuba.webAppUrl*

URL, по которому доступен Web Client приложения.

Используется, в частности, для формирования ссылок на экраны приложения извне, а также классом ScreenHistorySupport.

Значение по умолчанию: http://localhost:8000/app

Хранится в базе данных.

Интерфейс: GlobalConfig

Может использоваться во всех стандартных блоках.

*cuba.restApiUrl*

URL, по которому доступен REST API приложения.

Значение по умолчанию: http://localhost:8000/app-portal/api

Хранится в базе данных.

Интерфейс: GlobalConfig

Может использоваться во всех стандартных блоках.

*cuba.frontAppUrl*

URL, по которому доступен портал приложения приложения.

Значение по умолчанию: http://localhost:8000/app-front

Хранится в базе данных.

Интерфейс: GlobalConfig

Может использоваться во всех стандартных блоках.

*cuba.web.loginDialogDefaultUser*

Задает имя пользователя по умолчанию. Оно будет автоматически подставляться в экране входа в систему, что удобно в процессе разработки приложения. В режиме эксплуатации приложения в данном свойстве необходимо задать значение <disabled>.

Значение по умолчанию: admin

Интерфейс: WebConfig

Используется в блоке Web Client.

*cuba.web.loginDialogDefaultPassword*

Задает пароль пользователя по умолчанию. Он будет автоматически подставляться в экране входа в систему, что удобно в процессе разработки приложения. В режиме эксплуатации приложения в данном свойстве необходимо задать значение ` <disabled>`.

Значение по умолчанию: admin

Интерфейс: WebConfig

Используется в блоке Web Client.

*cuba.web.productionMode*

Позволяет полностью запретить консоль разработчика Vaadin в браузере, доступную через добавление ?debug к адресу приложения, тем самым, отключает доступ к возможностям отладки JavaScript и сокращает количество информации о сервере, выдаваемой браузеру.

Значение по умолчанию: false

Интерфейс: WebConfig

Используется в блоке Web Client.

*cuba.dataSourceJndiName*

Задает JNDI имя источника данных javax.sql.DataSource, через который производится обращение к базе данных приложения. Это имя должно соотвествовать Datasource.name в файле context.xml

Значение по умолчанию: java:comp/env/jdbc/CubaDS

Используется в блоке Middleware.

*cuba.dbDir*

Конфигурационный параметр, задающий расположение каталога скриптов базы данных.

Значение по умолчанию для WAR : web-inf:db, что означает расположение в подкаталоге WEB-INF/db внутри WAR.

Интерфейс: ServerConfig

Используется в блоке Middleware.

*cuba.automaticDatabaseUpdate*

Включает режим выполнения скриптов БД сервером на старте приложения.

Значение по умолчанию: false

Хранится в базе данных.

Интерфейс: ServerConfig

Используется в блоке Middleware.

*cuba.cluster.jgroupsConfig*

Путь к конфигурационному файлу JGroups. Файл загружается с помощью интерфейса Resources, поэтому может быть расположен в classpath или в конфигурационном каталоге.

Пример:

*cuba.cluster.jgroupsConfig = jgroups\_tcp.xml*

Значение по умолчанию: jgroups.xml

Используется в блоке Middleware.

*cuba.trustedClientPermittedIpList*

Список IP адресов, с которых возможен вызов метода LoginService.loginTrusted().

Значение по умолчанию: 127.0.0.1

Интерфейс: ServerConfig

Используется в блоке Middleware.

*cuba.cluster.enabled*

Включает взаимодействие серверов Middleware в кластере. Подробнее см. Настройка взаимодействия серверов Middleware.

Значение по умолчанию: false

Используется в блоке Middleware.

*reporting.openoffice.path*

Задает путь к установленному пакету OpenOffice.

Значение по умолчанию: /

Используется в блоке Middleware.

*reporting.openoffice.docx.useOfficeForDocumentConversion*

Включает использование OpenOffice для вывода отчета с DOCX шаблоном в HTML/PDF, что значительно увеличивает точность конвертации.

Значение по умолчанию: false

Используется в блоке Middleware.

*reporting.openoffice.docFormatterTimeout*

Указаывает максимальное время выполнения коныертации отчета из формата docx в секундах.

Используется в блоке Middleware.

*cuba.syncNewUserSessionReplication*

Включает синхронную отправку в кластер новых пользовательских сессий после логина в систему.

Значение по умолчанию: false

Интерфейс: ServerConfig

Используется в блоке Middleware.

*cuba.rest.syncTokenReplication*

Включает синхронную репликацию созданных токенов между кластерами Middleware. По умолчанию токены отправляются в кластер асинхронно.

Хранится в свойствах приложения.

Интерфейс: RestConfig

Значение по умолчанию: false

Используется в блоке Middleware.

#### Файл local.app.properties для Web Client

Файл local.app.properties для каждого сервиса уровня Web client, расположен в $CATALINA\_BASE/conf/appName и одинаковый для всех сервисов уровня Web client за исключением свойств cuba.webPort, где для каждого cuba.webPort указан http порт (см. таблицу портов), а в случае, если используется несколько серверов приложений, для каждого cuba.webHostName указан соотвествующий сервер либо виртуальное имя сервера, указывающее на несколько серверов.

#### Содержимое файла local.app.properties для tomcat1

cuba.logDir = ${catalina.base}/logs

cuba.confDir = ${catalina.base}/conf/${cuba.webContextName}

cuba.tempDir = ${catalina.base}/temp/${cuba.webContextName}

cuba.dataDir = /opt/tomcat/tomcat1/filestorage

cuba.webPort=

cuba.webHostName =

cuba.webContextName =

cuba.webAppUrl = https://cuba.company.com/${cuba.webContextName}

cuba.restApiUrl = http://${cuba.webHostName}:${cuba.webPort}/${cuba.webContextName}-portal/api

cuba.connectionUrlList = http://:${cuba.webPort}/${cuba.webContextName}-core

cuba.useLocalServiceInvocation = false

cuba.web.ldap.enabled = false

cuba.web.ldap.urls = ldap://192.168.1.20:3268

cuba.web.ldap.base = DC=company,DC=com

cuba.web.ldap.user = svc\_uco

cuba.web.ldap.password = <password>

cuba.web.loginDialogDefaultUser = <disabled>

cuba.web.loginDialogDefaultPassword = <disabled>

cuba.rest.ldap.enabled = false

cuba.rest.ldap.urls = ldap://192.168.1.20:3268

cuba.rest.ldap.base = DC=company,DC=com

cuba.rest.ldap.user = svc\_uco

cuba.rest.ldap.password = <password>

-------------------------------------------------------------------------------------------------------------

#### Файл local.app.properties для Middleware

Файл local.app.properties для сервиса уровня Middleware расположен в $CATALINA\_BASE/conf/ и одинаковый для всех сервисов уровня Middleware за исключением свойств cuba.webPort и cuba.webHostName, где для каждого cuba.webPort указан http порт (см. таблицу портов).

#### Содержимое файла local.app.properties для .

cuba.logDir = ${catalina.base}/logs

cuba.confDir = ${catalina.base}/conf/${cuba.webContextName}

cuba.tempDir = ${catalina.base}/temp/${cuba.webContextName}

cuba.dataDir = /opt/tomcat/tomcat1/filestorage

cuba.webPort=

cuba.webHostName =

cuba.webContextName =

cuba.dataSourceJndiName = appNameDS

cuba.download.directories = ${cuba.tempDir};${cuba.logDir}

cuba.dbDir = web-inf:db

cuba.automaticDatabaseUpdate = true

cuba.webAppUrl = https://ppe.company.com/appName

cuba.restApiUrl = http://${cuba.webHostName}:${cuba.webPort}/appName-portal/api

cuba.connectionUrlList = http://:${cuba.webPort}/${cuba.webContextName}ermittedIpList = 127.0.0.1,

reporting.openoffice.path = /usr/lib/libreoffice/program

reporting.openoffice.docx.useOfficeForDocumentConversion = true

reporting.openoffice.docFormatterTimeout=1800

----------------------------------------------------------------------------------

#### Хранение свойств в базе данных

Свойства приложения, представляющие собой параметры времени выполнения, хранятся в таблице SYS\_CONFIG базы данных.

Такие свойства имеют следующие особенности:

Так как значение свойства хранится в базе данных, оно задается в одном месте, независимо от того, в каких блоках приложения оно используется.

Значение может быть изменено и сохранено во время работы приложения следующими способами:

- Через экран Administration > Application Properties.

- Через JMX бин ConfigStorageMBean.

Если конфигурационный интерфейс, содержащий это свойство, имеет соответствующий setter, то свойство может изменено кодом приложения.

Значение свойства может быть переопределено для конкретного блока приложения в его файле app.properties или одноименным системным свойством Java.

Следует иметь в виду, что на клиентском уровне чтение свойства, хранящегося в БД, приводит к запросу к Middleware, что менее эффективно, чем чтение свойства из локального файла app.properties. Для уменьшения количества таких запросов клиент кэширует все свойства, хранящиеся в БД, на время жизни экземпляра реализации конфигурационного интерфейса. Поэтому если, например, в некотором экране UI необходимо несколько раз обратиться к свойствам одного конфигурационного интерфейса, лучше получить ссылку на него при инициализации экрана, и сохранить в поле для последующих обращений к одному и тому же экземпляру.

Параметры, хранящиеся в базе данных для администрирования экземпляра

|  |  |
| --- | --- |
| Наименование параметра | Значение параметра |
| hse.organization.code | 2790 |

Значения свойств приложения могут быть заданы в базе данных, в файлах свойств, через системные свойства Java или переменные окружения ОС. Если свойство с некоторым именем задано в нескольких источниках, его значение определяется в следующем приоритете:

1. Системное свойство Java (высший приоритет)
2. Переменная окружения ОС
3. Файл свойств
4. База данных (низший приоритет)

Например, значение, заданное в файле, переопределяет одноименное значение, заданное в БД.

Некоторые свойства не поддерживают установку свойств в базе данных по причине того, что их значения требуются еще до того, как БД становится доступной приложению. Это параметры конфигурации и развертывания. Поэтому их можно устанавливать только в файлах свойств, через системные свойства Java или переменные окружения ОС.

### Обновление приложеня

Для обновления приложения, следует обновить war файлы на сервере

AppServer-01.company.com. Файл для обновления Web Client называется

appName.war, файл для обновления Middleware называется appName-core.war.

*Обновление приложений:*

|  |  |
| --- | --- |
| WAR файл | Путь |
| .war | /opt/tomcat//webapps |
| .war | /opt/tomcat//webapps |

При обновлении приложений, необходимо придерживаться следующего порядка:

- Остановить сервисы приложений

- Очистить содержимое директории $CATALINA\_BASE/webapps

- Скопировать соответствующий файл в директорию $CATALINA\_BASE/webapps

- Запустить сервисы приложений

Для удобства обновления приложений на сервере AppServer-01.company.com создан скрипт /home/tomcat1/bin/deploy.sh, который обновляет приложения war файлами из директории /home/tomcat/deploy

### Добавление нового сервиса Apache Tomcat уровня Web Client

Для добавления нового сервиса Apache Tomcat, следует выполнить следующие действия на сервере приложений. Ниже приведен пример создание сервиса tomcat2

1) скопировать в директории $CATALINA\_BASE с большим номером и прирастить на единицу номер сервиса,

cd /opt/tomcat

cp –rp tomcat4

2) выполнить очистку логов

rm –f tomcat4/logs

3) в файле tomcat2/conf/server.xml прирастить на единицу значения портов для

Server port= +1  
 Connector port= +1 protocol="HTTP/1.1"

4) изменить в файле tomcat4/conf//local.app.properties порт, прирастив единицу

cuba.webPort =+1

5) создать скрипт запуска systemctl в /usr/lib/systemd/system

sudo cp /usr/lib/systemd/system/tomcat1.service /usr/lib/systemd/system/tomcat4.service

6) отредактировать скрипт запуска tomcat4.service для systemctl

sudo vi /usr/lib/systemd/system/tomcat4.service

изменить параметр

Environment=CATALINA\_BASE=/opt/tomcat/tomcat4

в параметре Environment=CATALINA\_OPTS прирастить на единицу

-Dcom.sun.management.jmxremote.port= +1

7) включить скрипт запуска и запустить сервис

sudo systemctl enable tomcat4.service

sudo systemctl start tomcat4

8) добавить новый web port в firewall

9) добавить новый сервис в балансировку HAproxy

в файле /etc/haproxy/haproxy.conf дописать в раздел bacend server значение аттрибута Connector port из tomcat4/conf/server.xml

server cuba\_srv4 192.168.1.1:8004 check cookie S1

10) перезапустить HAproxy

sudo systemctl stop haproxy  
 sudo systemctl start haproxy

11) Отредактировать скрипты обслуживания в директори /home/tomcat/bin, добавив новый сервис (см. раздел «Описание скриптов обслуживания»)

«Описание скриптов обслуживания»)

### Добавление нового сервиса Apache Tomcat уровня Middleware

Для добавления нового сервиса Apache Tomcat, следует выполнить следующие действия на сервере приложений. Ниже приведен пример создание сервиса tomcat32

1) скопировать в директории /opt/tomcat $CATALINA\_BASE с большим номером и прирастить на единицу номер сервиса,

cd /opt/tomcat

cp –rp tomcat3

2) выполнить очистку логов

rm –f tomcat3/logs

3) в файле tomcat4/conf/server.xml прирастить на единицу значения портов для

Server port= +1  
 Connector port= +1 protocol="HTTP/1.1"

4) изменить в файле tomcat3/conf//local.app.properties порт, прирастив едниницу

cuba.webPort =+1

5) создать скрипт запуска systemctl в /usr/lib/systemd/system

sudo cp /usr/lib/systemd/system/tomcat1.service /usr/lib/systemd/system/tomcat3.service

6) отредактировать скрипт запуска tomcat3.service для systemctl

sudo vi /usr/lib/systemd/system/tomcat3.service

изменить параметр

Environment=CATALINA\_BASE=/opt/tomcat/tomcat3

в параметре Environment=CATALINA\_OPTS прирастить на единицу

-Dcom.sun.management.jmxremote.port= +1

7) включить скрипт запуска и запустить сервис

sudo systemctl enable tomcat3.service

sudo systemctl start tomcat3

8) добавить новый webport в Firewall

9) добавить новый сервис в балансировку сервисов уровня Web Client в каждый файл $CATALINA\_BASE/conf//local.app.properies и Web Portal в каждый файл $CATALINA\_BASE/conf//local.app.properies

cuba.connectionUrlList = + **Ошибка! Недопустимый объект гиперссылки.** server>:<new middleware\_web\_port> /

10) перезапустить все Apache Tomcat сервисы уровня Web Client

sudo systemctl stop tomcat\*  
 sudo systemctl start tomcat\*

11) Отредактировать скрипты обслуживания в директории /home/tomcat/bin, добавив новый сервис (см. раздел «Описание скриптов обслуживания»)

### Мониторинг сервисов Apache Tomcat

Для просмотра состояния сервисов Apache Tomcat использовать:

1) вывести информацию с использованием утилиты systemctl для всех сервисов

sudo systemctl status tomcat\*

2) просмотром лога catalina.out в директории $CATALINA\_BASE/logs

3) графической утилитой jvisualvm, подключившись к рабочему столу сервера пользователем tomcat и выполнив команду

jvisualvm

Утилита позволяет просмотреть состояние используемой памяти, загрузки CPU, работу сборщика мусора, а также вручную вызывать сборку мусора.

### Мониторинг пользовательских сессий

Приложение позволяет просматривать различные уровни состояния системы, сессий и сущностей:

#### Просмотр текущих пользовательских соединений

Зайти в меню Администрирование -> Пользовательские сессии

На форме отображается информация о текущих подключениях и отображается следующая информация

- ID сессии пользователя

- логин пользователя

- имя пользователя

- IP-адрес, с которого пришёл запрос на вход в систему

- Информация о клиенте (Web, Rest, ID сервера (например,192.168.1.1:8001/appName, операционная система, веб-браузер и т.д.)

- дата и время начала сессии

- последняя активность сессии

#### Журнал пользовательских сессий

Механизм журналирования предназначен для отслеживания факта входа пользователей в систему. В журнале администратор системы может найти информацию, кто и когда вошёл в систему и вышел из неё. Механизм основан на отслеживании пользовательских сессий. При каждом создании объекта UserSession в базу данных сохраняется следующая информация:

- ID сессии пользователя

- ID пользователя

- ID замещаемого пользователя

- последнее действие пользователя (логин / выход / истечение срока сессии / сессия прервана)

- удаленный IP-адрес, с которого пришёл запрос на вход в систему

- тип клиента (web, desktop, portal)

- ID сервера (например, localhost:8000/app-core)

- дата и время начала сессии

- дата и время окончания сессии

- информация о клиенте (окружение сессии: операционная система, веб-браузер и т.д.)

По умолчанию записи о пользовательских сессиях не сохраняются. Простейший способ активировать журналирование - установить в true значение свойства приложения cuba.UserSessionLogEnabled и перезапустить сервер.

При необходимости можно создать отчёт для сущности sec$SessionLogEntry.

Для просмотра необходимо зайти на форму Администрирование-->Журнал изменения сущностей, выбрать в поле "Сущность" значение "Лог пользовательских сессий (sec$SessionLogEntry)", нажать на кнопку просмотра (...) поля "Экземпляр сущности". При необходимости добавить в фильтр интересующие даты для просмотра сессий за интересущющий период.

### Настройка журнала изменения сущностей

Механизм журналирования предназначен для отслеживания изменений атрибутов произвольных сущностей в процессе работы приложения. Измененные значения сохраняются в специальной таблице базы данных, список изменений для конкретного экземпляра сущности может быть отображен в пользовательском интерфейсе.

Данный механизм перехватывает сохранение сущностей в БД на уровне Entity Listeners, т.е. гарантированно отслеживаются все изменения, проходящие через персистентный контекст EntityManager. Непосредственное изменение сущностей в базе данных с помощью SQL, в том числе изнутри системы через NativeQuery и QueryRunner, в журнал не попадает.

Журнал содержит информацию о том, кто и когда изменил данный экземпляр, а также новые значения измененных атрибутов. Записи журнала сохраняются в таблице SEC\_ENTITY\_LOG базы данных, соответствующей сущности EntityLogItem. Измененные значения атрибутов хранятся в этой же таблице в колонке CHANGES, а при чтении на Middleware преобразуются в экземпляры сущности EntityLogAttr.

Аудит настраивается при помощи сущностей LoggedEntity и LoggedAttribute (соответствующих таблицам SEC\_LOGGED\_ENTITY и SEC\_LOGGED\_ATTR).

LoggedEntity описывает тип сущности, изменения которой необходимо журналировать. Атрибуты LoggedEntity:

name (колонка NAME) - тип сущности в виде имени мета-класса, например, sales$Customer.

auto (колонка AUTO) - нужно ли журналировать изменения при вызове EntityLogAPI с параметром auto = true (т.е. из Entity Listeners).

manual (колонка MANUAL) - нужно ли журналировать изменения при вызове EntityLogAPI с параметром auto = false.

LoggedAttribute описывает журналируемый атрибут сущности и содержит ссылку на LoggedEntity и имя атрибута.

Если вы хотите его отключить, необходимо установить в false атрибут Enabled JMX-бина app-core.cuba:type=EntityLog и вызвать его операцию invalidateCache(). В качестве альтернативы можно установить в false значение свойства приложения cuba.entityLog.enabled и рестартовать сервер.

Для настройки журнала – перейти в меню Администрирование --> Журнал изменения сущностей --> Настройка. Нажать кнопку «Создать». В форме редактирования, которая отображается с правой стороны выбрать наименование сущности, которую необходимо отслеживать. Поставить галочки «Авто» и/или «Вручную» и выбрать аттрибуты, по которым хотите отслеживать изменения. После выбора аттрибутов, нажать кнопку «Сохранить». На вкладке Настройка, формы «Журнал изменения сущностей», нажать кнопку «Применить изменения».

### Просмотр журнала изменения сущностей

Для просмотра журнала – перейти в меню Администрирование --> Журнал изменения сущностей.

Выбрать в фильтре необходимые критерии, такие как: Сущность, Пользователь, Тип изменения, Период и нажать кнопку «Поиск». Если по данной сущности настроен журнал изменения, то отобразится список сделанных изменений по выбранной сущности. В правой части формы отображаются изменения, сделанные пользователем: аттрибут, старое значение, новое значение.

### Очистка логов приложения

Очистка логов приложения производится либо в автоматическом режиме, сохраняя логи за последние 4 дня, либо в ручном режиме, очистка полностью, с помощью скрипта /home/tomcat1/bin/clear\_logs.sh.

### Резервное копирование

На сервере настроено резервное копирование конфигурационных файлов Tomcat и \*.war файлов. Резервное копирование осуществляется скриптом /home/tomcat/bin/backup.sh, которое хранит 5 последних резервных копии в директории /backup/tomcat/$HOSTAME. Для изменения параметров хранения количества резервных копий, изменить в скрипте backup.sh переменную $BACKUPS\_TO\_KEEP.

### Команды управления Apache Tomcat

Для остановки/запуска сервисов Apache Tomcat использовать следующие команды

|  |  |
| --- | --- |
| Команда | Примечание |
| sudo systemctl start tomcat\* | Запуск сервисов уровня Web Client и Middleware |
| sudo systemctl stop tomcat\* | Останов сервисов уровня Web Client и Middleware |

### Описание скриптов обслуживания

На серверах в директории /home/tomcat/bin находятся скрипты по обслуживанию приложений и сервисов Tomcat

.

|  |  |
| --- | --- |
| Скрипт | Примечание |
| backup.sh | Скрипт резервирования конфигурационных файлов и \*.war файлов. Хранит 5 последние версии резервных копий. |
| clear\_logs.sh | Выполняет полную очистку логов tomcat в директориях $CATALINA\_BASE/log |
| deploy.sh | Копирует ранее собранный файлы из директории /home/tomcat/deploy и устанавливает его на продуктивный экземпляр. |
| start\_tomcat.sh | Выполняет запуск всех сервисов Tomcat |
| stop\_tomcat.sh | Выполняет останов всех сервисов Tomcat |

Содержимое скрипта backup.sh

#!/bin/bash

BACKUP\_DIR="/backup/tomcat/$HOSTNAME"

CURRENT\_TIME="$(date +%Y%m%d%H)"

LOG="/backup/tomcat/log/$CURRENT\_TIME.log"

SAVE\_BACKUP\_DIR="$BACKUP\_DIR/$CURRENT\_TIME"

mkdir -p $SAVE\_BACKUP\_DIR

cd $SAVE\_BACKUP\_DIR

cp -rp /opt/tomcat .

cp -rp /opt/java .

mkdir -p bin

cp -rp /home/tomcat/bin/\* bin/

cp /lib/systemd/system/tomcat\*.service .

cp /etc/haproxy/haproxy.cfg .

cd ..

tar czf "$CURRENT\_TIME.tar.gz" $CURRENT\_TIME

rm -rf $CURRENT\_TIME

BACKUPS\_TO\_KEEP="2"

if [ $BACKUPS\_TO\_KEEP -ne "0" ]; then

for backup\_name in `ls -dtr $BACKUP\_DIR/\* `

do

arr[i]=$backup\_name

echo "\*\*\* Backup name: ${arr[$i]}" >> $LOG

let "i += 1"

done

echo "" >> $LOG

echo "\*\*\* Number of exports to keep is $BACKUPS\_TO\_KEEP." >> $LOG

echo "\*\*\* Number of detected backups is $i." >> $LOG

if [ "$i" -gt "$BACKUPS\_TO\_KEEP" ]; then

let "i = i - $BACKUPS\_TO\_KEEP"

echo "\*\*\* $i obsoleted exports will be deleted." >> $LOG

echo "" >> $LOG

for (( x=0; x < $i; x++ ))

do

rm -rf ${arr[$x]} 2> /dev/null

if [ $? -eq "0" ]; then

echo "\*\*\* Deleted export: ${arr[$x]}" >> $LOG

fi

done

else

echo "\*\*\* No exports for delete." >> $LOG

fi

fi

find /backup/tomcat/log -mtime +14 -exec rm -f {} \;

### Автоматические задачи планировщика операционной системы (cron)

Запланированные задачи в cron пользователя tomcat.

|  |  |  |
| --- | --- | --- |
| Расписание | Задача | Примечание |
| 0 0 \* \* \* | /home/tomcat/bin/backup.sh | Ежедневное резервирование конфигурационных файлов tomcat |
| 55 23 \* \* \* | /bin/find /opt/tomcat/tomcat1/logs -mtime +4 -ecec rm -f {} \; | Ежедневная очистка логов, старше 4х дней |

## Порядок запуска/останова всей системы

### Запуск системы

На серверах настроен автоматический запуск всех приложений, но для обслуживания системы, например, таких как, обновления ядра, перемонтирование дисков и т.п. может потребоваться останов сервисов. При запуске/останове сервисов важно соблюдать порядок.

Порядок запуска всей системы отображена в таблице

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Команда | Сервер | Пользователь | Описание |
| 1 | sudo systemctl start | DBServer-01.company.com | postgres | Запуск кластера PostgreSQL |
| 2 | sudo systemctl start |  | tomcat | Запуск сервисов Tomcat уровня Web Сlient и Middleware |
| 3 | sudo systemctl start haproxy |  | tomcat | Запуск HAproxy |

### Останов системы

Порядок останова всей системы отображена в таблице

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Команда | Сервер | Пользователь | Описание |
| 1 | sudo systemctl stop haproxy |  | tomcat | Останов HAproxy |
| 2 | sudo systemctl stop |  | tomcat | Останов сервисов Tomcat уровня Middleware и Web client |
| 3 | sudo systemctl stop | DBServer-01.company.com | postgres | Останов кластера PostgreSQL |

## Изменение учетных записей

### Изменение учетной записи пользователя tal базы данных PostgreSQL

Процедура изменения пароля учетной записи пользователя tal базы данных PostgreSQL:

1. Изменить пароль пользователя tal на сервере DBServer-01.company.com

подключиться через ssh пользователем postgres

* в командной строке набрать команду psql
* в командной строке psql выполнить команду

alter user tal with password ‘newpassword’;

2. На сервере приложений , поменять атрибут password в конфигурационном файле $CATALINA\_BASE/conf/context.xml для всех сервисов Apache Tomcat.

1. Add additional Failure Scenarios as required.

## Клонирование экземпляра

### Подготовка нового сервера и программного обеспечения

Самым простым способом подготовить новый сервер для разворачивания клона – сделать копию виртуальной машины, выполнить настройки среды CUBA, перенаправить подключения к новой базе данных.

Возможен вариант установки «с нуля» программного обеспечения Java, Apache Tomcat, PostgreSQL, HAproxy, в соответствии с описанным в документе версиям и конфигурациям программного обеспечения.

Следующее описание работ предполагает, что вышеуказанные работы по настройке сервера и установке программного обеспечения, либо клонирование виртуальной машины выполнено.

### Создание пустой базы данных

В некоторых случаях требуется создать пустую базу данных со всеми необходимыми объектами и частично проинициализированную. Объекты и инициализация данных в таком случае происходит при первом запуске приложения, если свойство cuba.automaticDatabaseUpdate выставлено в true, в файле local.app.properties

Если база данных создается на сервере, на котором было «с нуля» установлено программное обеспечение PostgreSQL, то так же необходимо создать пользователя – владельца базы данных:

create user tal password ‘<password>’;

Далее необходимо создать пустую базу данных

create database <database\_name> owner tal;

Изменить строку подключения к базе данных в контекстном файле $CATALINA\_BASE/conf/context.xml для каждого $CATALINA\_BASE. Указать новый сервер, имя пользователя, пароль, базу данных.

### Перенос базы данных с продуктивного сервера

В случае, если надо просто обновить базу данных переносом с продуктивного сервера, необходимо выполнить следующие действия:

На продуктивном сервере базы данных выполнить экспорт данных пользователем postgres командой

pg\_dump –f appName\_prod.sql appName\_prod

Эта команда выгрузит базу данных appName\_prod в текстовый файл appName\_prod.sql в текущей директории. Для выполнения данной команды, необходимо убедиться, что у вас есть необходимый размер дискового пространства в текущей директории, либо переназначить путь, указав раздел с достаточным количеством свободного пространства.

Создать новую базу, при необходимости удалить старую базу данных.

Команда

drop database <database\_name’;

удаляет базу данных, если есть в этом необходимость.

Команда

create database <database\_name> owner tal;

создаст новую базу данных, владельцем которой является пользователь tal.

Выполнить импорт данных в базу данных на клонированном сервере пользователем postgres.

psql –f appName\_prod.sql appName\_prod\_clon

Данная команды выполнит загрузу данных из файла appName\_prod.sql в базу appName\_prod\_clon. База данных должна существовать перед выполнением этой команды.

### Настройка сервисов tomcat

Если настраивается новая среда клонирования, то необходимо указать в файлах $CATALINA\_BASE/conf/server.xml корректные порты для данного экземпляра. Требуется так же убедиться, что указанные порты не заняты другими приложениями, и порты доступа и отладки открыты в брандмауэре сервера. Подробнее см. описание раздела «Файл server.xml»

### Настройка параметров приложений Cuba

Если сервер был клонирован путем копирования виртуальной машины, то необходимо изменить существующие файлы $CATALINA\_BASE/conf/context.xml, $CATALINA\_BASE/conf/appName/local.app.properties и $CATALINA\_BSE/conf/appName-core/local.app.properties. Если выполняется просто обновление базы, то нет необходимости менять конфигурацию файлов.

В файле context.xml указать параметры подключения к новой базе, а именно: имя сервера, порт, имя пользователя, пароль, база данных. Подробнее см. описание раздела «Файл context.xml»

В файлах $CATALINA\_BASE/conf/appName/local.app.properties и $CATALINA\_BASE/conf/appName-core/local.app.properties

необходимо правильно указать веб порт приложения, hostname и наименование подключения в context.xml

Подробнее см. раздел «Параметры файла local.app.properties»

### Настройка systemd сервисов

Для каждого сервиса tomcat в системе создан systemd файл запуска сервисов tomcat, haproxy, postgresql. См. документ «Техническая архитектура» с описанием каждого файла systemd.

Для запуска сервисов tomcat указываются debug порт в переменной $CATALINA\_OPTS, параметр address. Этот порт так же необходимо открыть в брендмауре сервера

ера.

Важно после каждого изменения systemd скрипта, которые находятся в директории /usr/lib/systemd/system выполнять команду systemctl daemon-reload

### Пример ручного клонирования системы с продуктивной на тестовую среду

1. пользователем postgres на сервере AppServer-01.company.com выгрузить базу

cd /opt/postgres/16

pg\_dump -f appName\_prod.sql

2. скопировать файл с сервера AppServer-01.company.com на тестовый сервер 192.168.1.2 в /opt/postgres/16

scp appName\_prod.sql postgres@192.168.1.2:/opt/postgres/16

3. на сервере 192.168.1.2 создать и загрузить базу пользователем postgres

psql

postgres=# create database appName\_prod\_clon owner tal;

postgres=# grant all on database appName\_prod\_clon to tal;

postgres=# \q

cd /opt/postgres/16

psql -f .sql \_test

4. проверить номер последнего созданного tomcatN на сервере 192.168.1.2 в директории /opt/tomcat

ls -la tomcat\*

5. пользователем tomcat на сервере AppServer-01.company.com заархивировать /opt/tomcat/tomcat3 (резервная копия)

cd /opt/tomcat

tar czf tomcat3.tar.gz tomcat3

6. Создать папку в /opt/tomcat/tomcatN, где N — это номер последнего tomcat+1 на сервере AppServer-01.company.com. Скажем последний tomcat4, тогда команда будет следующая

cd /opt/tomcat

mkdir tomcat5

7. заархивировать tomcat3 на сервере AppServer-01.company.com и скопировать на сервер AppServer-01.company.com

cd /opt/tomcat

tar czf tomcat3.tar.gz tomcat3

scp tomcat3.tar.gz tomcat@AppServer-01.company.com:/opt/tomcat/tomcat5

8. распаковать tomcat3 на сервере AppServer-01.company.com

cd /opt/tomcat/tomcat5

tar xzf tomcat3.tar.gz

mv -fR tomcat3/\* /opt/tomcat/tomcat5

rm -fR tomcat3 tomcat3.tar.gz

9. выполнить очистку ненужных файлов (логи, распакованные диретории) на сервере AppServer-01.company.com, сохранив WAR файлы, filestorage и конфигурационные файлы

cd /opt/tomcat

rm -rf tomcat5/logs/\*

rm -rf tomcat5/temp/\*

rm -rf tomcat5/webapps/appName

rm -rf tomcat5/webapps/appName-core

10. отредактировать файл tomcat5/conf/context.xml. Изменить аттрибуты подключения к БД

password ← указать пароль пользователя БД tal для сервера 192.168.1.2

url ← изменить имя БД с appName\_prod на имя, созданное на шаге 3 данной инструкции

11. отредактировать файл tomcat5/conf/server.xml. Изменить порты

<Server port="N" shutdown="SHUTDOWN"> , где N = 9000 + номер tomcat. В данном примере будет 9005

<Connector port="N" protocol="HTTP/1.1", где N = 8000 + номер tomcat. В данном примере будет 8005

12. отредактировать файл tomcat5/conf/appName/local.app.properties. Изменить свойства

cuba.webPort= указать порт из server.xml

cuba.webHostName=<сюда dns имя для адреса AppServer-01.company.com >

13. отредактировать файл tomcat5/conf/appName-core/local.app.properties. Изменить свойства

cuba.webPort= <порт из server.xml>

cuba.webHostName=<сюда dns имя для адреса AppServer-01.company.com >

cuba.trustedClientPermittedIpList=127.0.0.1, <ip адрес сервера сервера AppServer-01.company.com >>

reporting.openoffice.path = /usr/lib64/libreoffice/program

14. пользователем root на сервере AppServer-01.company.com создать скрипт запуска systemd нового экземпляра tomcat5

cd /usr/lib/systemd/system

cp tomcat1.service tomcat5.service

16. отредактировать файл tomcat5.service, изменить

Environment=CATALINA\_BASE= ← указать новое расположение

Environment='CATALINA\_OPTS ← изменить 2 порта

-Dcom.sun.management.jmxremote.port=10000+<номер tomcat> , в данном примере 10005

-agentlib:jdwp=transport=dt\_socket,server=y,suspend=n,address=11000+<номер tomcat>, в даном случае 11005

16. включить автоматическую загрузку сервиса и запустить сервис

systemctl enable tomcat5

systemctl start tomcat5

17. проверить вход в приложении по следующему URL(страница логина должна открыться без ошибок)

http://AppServer-01.company.com:8005/appName

П 1-3 выполнять пользователем postgres

П 4-13 выполнять пользователем tomcat

П 14-16 выполнять пользователем root

При возникновении ошибок (не открывается страница логина) смотреть в tomcat32/logs/catalina.out или tomcat5/logs/localhost.yyyy-mm-dd.log

## Доступ к приложению CUBA

1. Add open issues that you identify while writing or reviewing this document to the open issues section. As you resolve issues, move them to the closed issues section and keep the issue ID the same. Include an explanation of the resolution.  
     
   When this deliverable is complete, any open issues should be transferred to the project- or process-level Risk and Issue Log (PJM.CR.040) and managed using a project level Risk and Issue Form (PJM.CR.040). In addition, the open items should remain in the open issues section of this deliverable, but flagged in the resolution column as being transferred.

### Общие сведения

Система CUBA построена по трехуровневой архитектуре и состоит из клиента, серверного слоя и базы данных. HAproxy выполняет роль прокси-сервера, обеспечивая балансировку нагрузки между компонентами системы. Балансировка также выполняется HAProxy, что повышает стабильность и отказоустойчивость системы.

Балансировщик HAproxy установлен на сервере организации, где настроен обратный прокси сервер, переправляющий запросы пользователей на сервер CUBA - AppServer-01.company.com

#### Корпоративная среда размещения

|  |  |  |
| --- | --- | --- |
| Сервер | IP адрес | Роль сервера |
|  |  | Active | |

## Уровень сервиса HAproxy

HAproxy выполняет балансировку нагрузки между клиентами и backend-сервером приложений. Для управления сессиями используется механизм сессий через куки. Доступна статистика производительности через веб-интерфейс по адресу: http://192.168.1.1:8404/stats.

#### Корпоративная среда размещения

Балансировщик HAproxy установлен на том же сервере, где установлены Apache Tomcat.

|  |  |  |
| --- | --- | --- |
| Сервер | IP адрес | Роль сервера |
|  |  | Active | |

#### Основные директории HAproxy Load Balancer

Директории, необходимые для работы балансировщика HAproxy

|  |  |
| --- | --- |
| Директория | Описание |
| /etc/haproxy | Корневая установка HAproxy Load Balancer |
| /usr/sbin | Бинарные файлы балансировщика HAProxy |

### Конфигурация

Основной конфигурационный файл HAProxy находится по пути /etc/haproxy/haproxy.cfg. Он содержит настройки для работы балансировщика и управления соединениями и логирования. Также для настройки логов HAProxy используется файл /etc/rsyslog.d/49-haproxy.conf, который задаёт параметры отправки логов в /var/log/haproxy.log.

#### Конфигурационные файлы HAproxy

Для настройки HAproxy используются следующие конфигурационные файлы:

|  |  |  |
| --- | --- | --- |
| Файл | Путь | Описание |
| haproxy.cfg | /etc/haproxy | Основной файл конфигурации HAproxy |

#### Конфигурационный файл /etc/haproxy/haproxy.cfg

global

log /dev/log local0

log /dev/log local1 notice

chroot /var/lib/haproxy

stats socket /run/haproxy/admin.sock mode 660 level admin

stats timeout 30s

user haproxy

group haproxy

daemon

# Default SSL material locations

ca-base /etc/ssl/certs

crt-base /etc/ssl/private

# See: https://ssl-config.mozilla.org/#server=haproxy&server-version=2.0.3&config=intermediate

ssl-default-bind-ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384

ssl-default-bind-ciphersuites TLS\_AES\_128\_GCM\_SHA256:TLS\_AES\_256\_GCM\_SHA384:TLS\_CHACHA20\_POLY1305\_SHA256

ssl-default-bind-options ssl-min-ver TLSv1.2 no-tls-tickets

defaults

log global

mode http

option httplog

option dontlognull

timeout connect 5000

timeout client 50000

timeout server 50000

errorfile 400 /etc/haproxy/errors/400.http

errorfile 403 /etc/haproxy/errors/403.http

errorfile 408 /etc/haproxy/errors/408.http

errorfile 500 /etc/haproxy/errors/500.http

errorfile 502 /etc/haproxy/errors/502.http

errorfile 503 /etc/haproxy/errors/503.http

errorfile 504 /etc/haproxy/errors/504.http

listen cuba

bind :80

mode http

option forwardfor

option httpclose

fullconn 20000

redirect code 302 location /appName if { path / }

cookie SERVERID insert indirect nocache maxidle 30m maxlife 8h

server cuba\_srv 192.168.1.1:8001 check cookie S1

listen stats

bind :8404

mode http

stats enable

# stats realm LBS

# stats scope fr1

# stats auth admin:!admin

stats uri /stats

-----------------------------------------------------------------------------------------------------

### Просмотр логов HAproxy

На сервере HAProxy просмотр логов в реальном времени осуществялется через команду:

sudo journalctl -xeu haproxy.service

### Мониторинг сервиса HAproxy

Доступна статистика производительности через веб-интерфейс по URL: [http://192.168.1.1:8404/stats](http://172.16.32.131:8404/stats)

Параметры веб-интерфейса указаны в конфигурационном файле haproxy.cfg:

listen stats

bind :8404

mode http

stats enable

# stats realm LBS

# stats scope fr1

# stats auth admin:!admin

stats uri /stats

### Параметры cookies HAproxy

HAProxy выдаёт Cookie SERVERID в ответ клиента.

Строка из конфигурационного файла haproxy.cfg содержит параметры cookie SERVERID:

insert indirect nocache maxidle 30m maxlife 8h

|  |  |  |
| --- | --- | --- |
| Наименование параметра | Значение | Описание |
| insert | - | Добавляет информацию об идентификаторе в Cookie. |
| indirect | - | Использует значение, связанное с пользователем (например, IP адрес клиента). |
| nocache | - | Не кэшировать Cookie, чтобы сохранять актуальность идентификатора. |
| maxidle | 30m | Устанавливает время, через которое Cookie считается устаревшим (если клиент не взаимодействует более 30 минут). |
| maxlife 8h | 8h | Ограничивает срок жизни Cookie до 8 часов. |

### Команды управления сервиса HAproxy

Для остановки/запуска/считывания конфигурации HAproxy, использовать следующие команды

|  |  |
| --- | --- |
| Наменование | Значение |
| sudo systemctl start haproxy | Запуск HAProxy Load Balancer |
| sudo systemctl stop haproy | Останов HAProxy Load Balancer |
| sudo systemctl reload haproxy | Перечитывание конфигурационных файлов HAProxy Load Balancer, без потери соединений |
| haproxy -c -f /etc/haproxy/haproxy.cfg | Валидация конфигурационного файла haproxy.cfg (требуется выполнять после каждого изменения) |

## Доступ к приложению Apache Tomcat

#### Корпоративная среда размещения

Балансировщик Nginx установлен на сервере организации anotherexample02.company.com (192.168.1.x) где настроен обратный прокси сервер переправляющий запросы пользователей на сервер приложения Cuba - AppServer-01.company.com по URL в браузере [https://ppe.company.com/](https://ppe.altynalmas.kz/)

|  |  |  |
| --- | --- | --- |
| Сервер | IP адрес | Роль сервера |
|  |  | Active | |

## Открытые и Закрытые Вопросы

1. Add open issues that you identify while writing or reviewing this document to the open issues section. As you resolve issues, move them to the closed issues section and keep the issue ID the same. Include an explanation of the resolution.  
     
   When this deliverable is complete, any open issues should be transferred to the project- or process-level Risk and Issue Log (PJM.CR.040) and managed using a project level Risk and Issue Form (PJM.CR.040). In addition, the open items should remain in the open issues section of this deliverable, but flagged in the resolution column as being transferred.

### Открытые Вопросы

| Номер | Проблема | Решение | Ответственный | Решить до | Критичная дата |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

### Закрытые Вопросы

| Номер | Проблема | Решение | Ответственный | Решить до | Критичная дата |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |