**Описание и функциональные возможности программы управления кластерами RabbitMQ «Мазай».**

В статье **Эксплуатация кроликов в режиме «Выжить любой ценой»** на habr.com я вкратце рассказал о существующей распределённой структуре кластеров RabbitMQ. Теперь подробно рассмотрим разработанное ПО для управления всем этим высоконагруженным зоопарком.

Программа не претендует на покрытие всех возможных эксплуатационных сценариев, но подавляющее большинство эксплуатационных задач позволяет решить достаточно просто. Весь основной функционал построен на использовании API самого кролика.

Как обычно договариваемся об используемых терминах:

«**Zone**», «**Зона**» - совокупность всех кластеров RabbitMQ, объединённых в единое информационное пространство. В нашем случае есть «**Рабочая Зона**» и одна или несколько «**Тестовых зон**».

В каждой из зон есть произвольное количество «**Contur**», «**Контур**» - совокупность одного или нескольких кластеров RabbitMQ объединённых по принадлежности к одному логическому узлу транспорта. Основной и произвольное количество идентичных ему резервных кластеров: «**Levels**». Такая структура необходима для автоматического переключения на резерв и обратно (подробнее см. вышеназванную статью).

Все кролики, которыми планируется управлять, для начала необходимо описать.

Для этого предназначен файл **rabbit\_servers.json**

Структура файла: **“zones”** – массив из списка всех зон, управление которыми планируется

“**zones**”.“**name**” – имя зоны

“**zones**”.“**conturs**” – массив всех контуров зоны

“**zones**”.“**conturs**”.”**name**” – имя контура

“**zones**”.“**conturs**”.”**statistic\_point**” – если используется балансировщик haproxy – url для получения статистики.

“**zones**”.“**conturs**”.”**source\_alias**” – адрес хоста в «src-uri» настройках шовелов, который будет заменён на “**zones**”.“**conturs**”.”**destination\_alias**” – используется при клонировании Рабочей зоны в Тестовую.

“**zones**”.“**conturs**”.”**levels**” – массив, содержащий все кластера однго узла – основной и резервные. Резервные узлы могут и отсутствовать, как например на простой тестовой зоне.

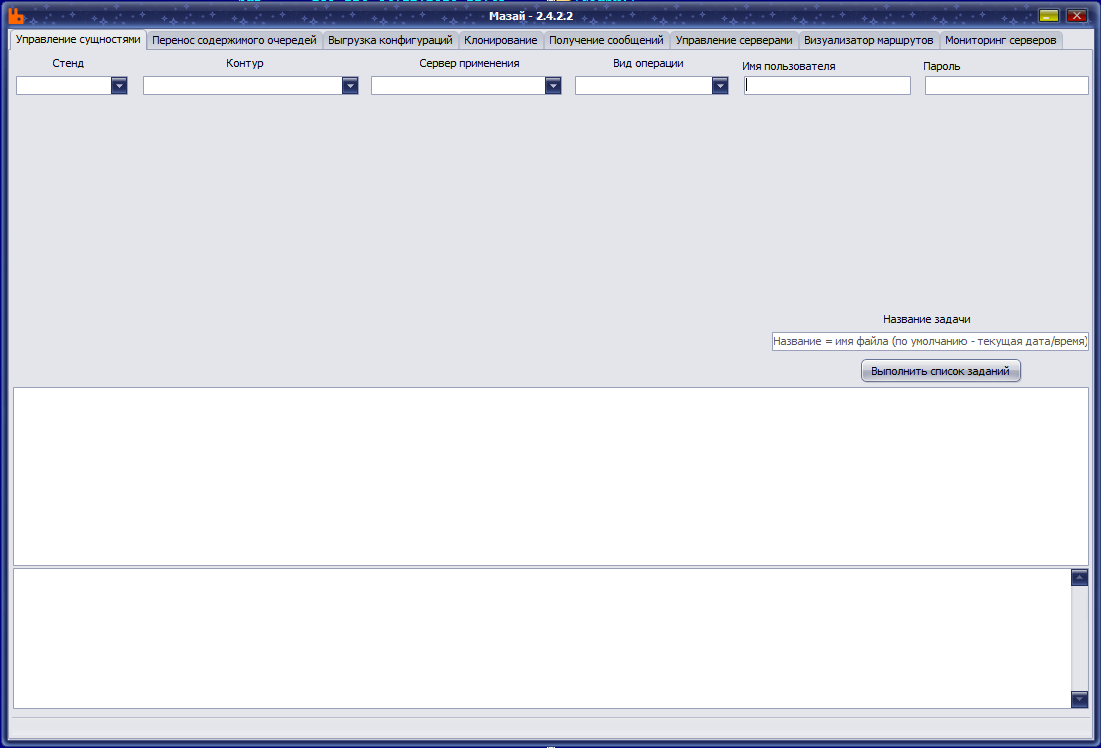
“**zones**”.“**conturs**”.”**levels**”.”**nodes**” – список нод соответствующего кластера. (В текущей версии 2.4.2 используется только первая нода)

“**zones**”.“**conturs**”.”**levels**”.”**nodes**”.”**name**” – имя сервера RabbitMQ или его IP адрес – используется для подключения к серверам.

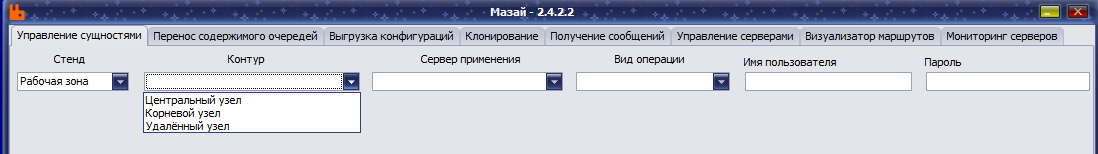
“**zones**”.“**conturs**”.”**levels**”.”**db**”, “**zones**”.“**conturs**”.”**levels**”.”**monitoring\_bank**” – настройки для использования совместно с сбором метрик в базу Influx DB.

Описав в таком файле всю используемую структуру RabbitMQ, переходим непосредственно к самой программе.

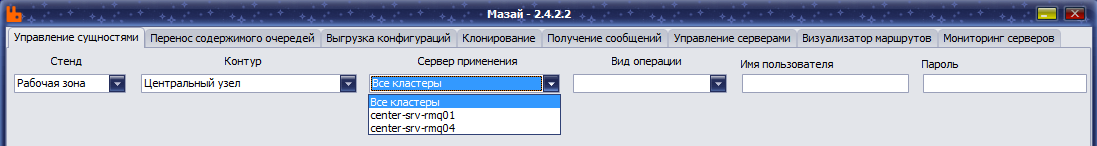
После запуска отображается основное окно программы и активируется первая закладка «**Управление сущностями**».



«Стенд» - выбор зоны, пока не выбрана зона выбор контура не доступен. Он будет заполнен после выбора зоны только контурами, принадлежащими выбранной зоне.

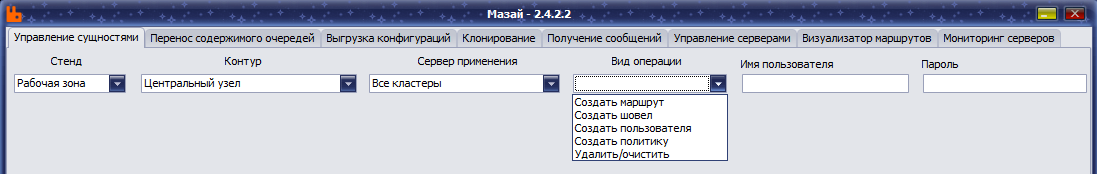


Аналогичным образом выбор сервера применения будет доступен после выбора контура и заполнен только ПЕРВЫМИ нодами кластеров выбранного контура

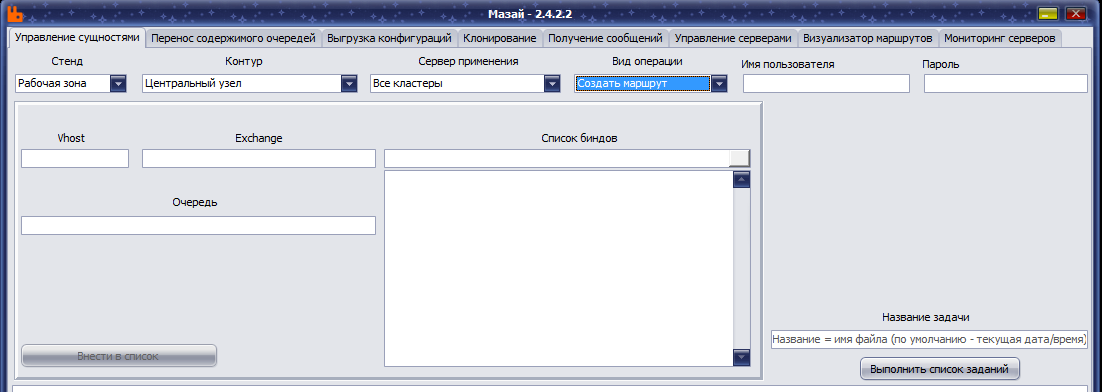


Первым пунктом автоматически добавляется «Все кластеры» - используется для генерации одинаковых команд на все кластеры. В противном случае будет сгенерирована команда на применение только на выбранном кластере

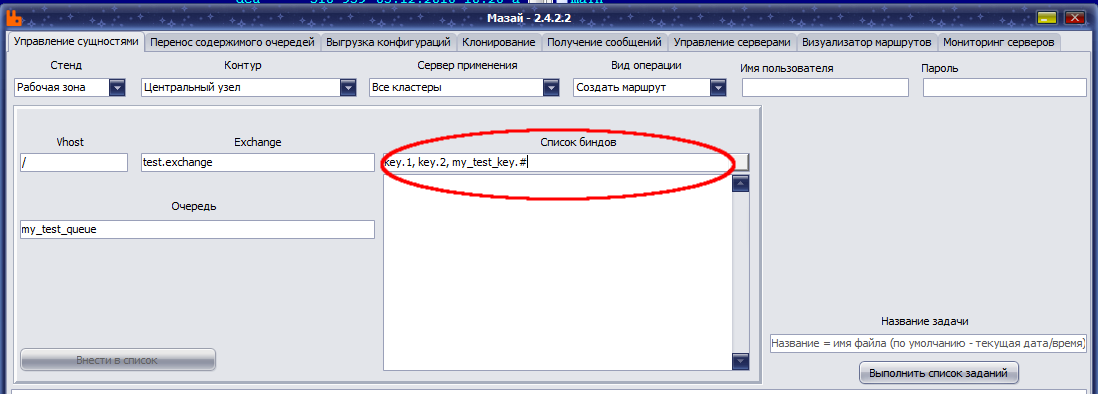
Возможные варианты видов операций

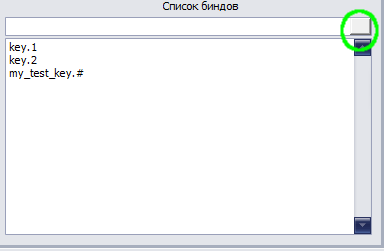


**Создать маршрут**



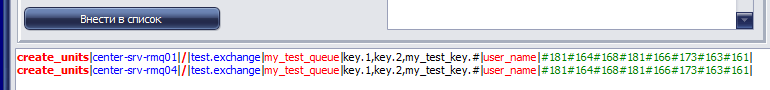
«Список биндов» - позволяет добавлять как по одному значению, так и списком (разделители элементов – перевод строки или «,» ) Ввод осуществляется в поле ввода (обведено красным). Добавление в список – нажатие на выделенную зеленым кнопку или нажатие на клавишу “Enter”

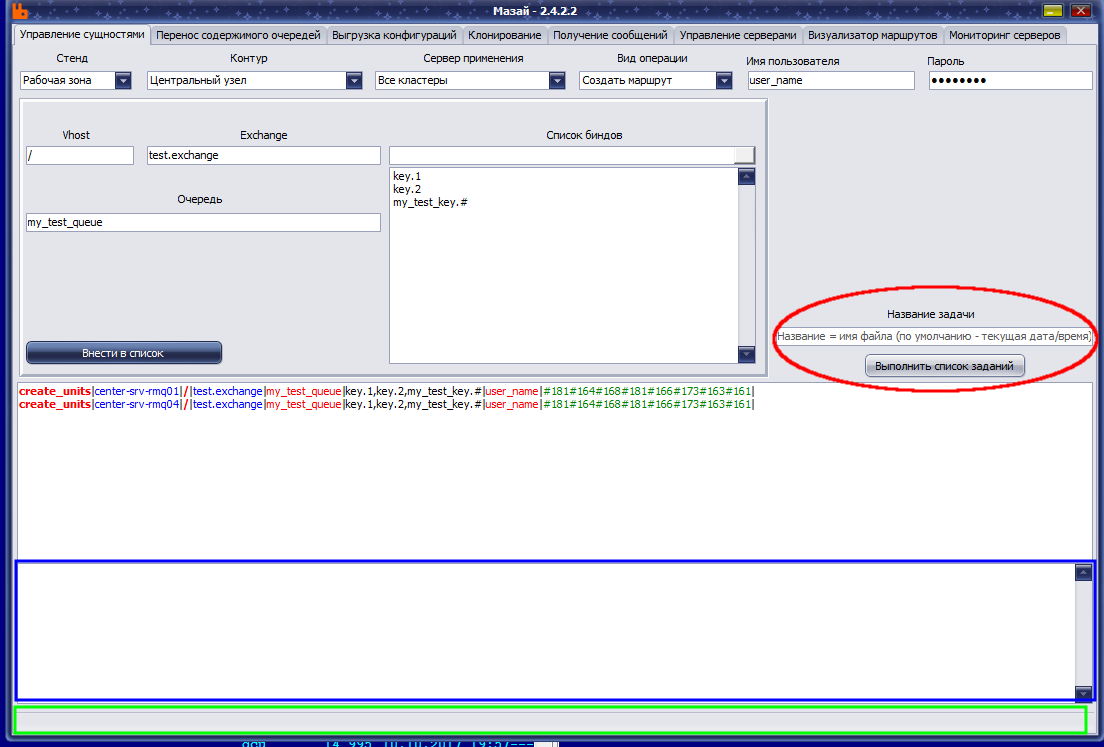


Результат: 

Все начальные и конечные пробелы и спец.символы в ключах будут удалены.

Пока не будут заполнены все необходимые поля, кнопка «Внести в список» будет недоступна. Не забываем заполнить имя пользователя и пароль – учётная запись у которой есть права администратора на WEB интерфейсе. После заполнения всех полей и нажатия на «Внести в список», на указанный кластер (его первая нода) или на все кластеры будет сгенерирована команда вида:





Осталось заполнить (при желании) имя файла в который будут записаны выполненные команды (выделено красным), в папке с программой будет создана папка **history**, и нажать на «Выполнить список заданий». В строке состояния (выделено зелёным) будет отображаться текущий статус выполнения команды, а в поле лога (выделено синим) будет записан пошаговый результат выполнения команд. Пароль учётной записи под которой выполняются команды всегда используется в шифрованном виде и дешифруется только непосредственно при вызове функций API кролика.

Особенности:

1. Эксченджи создаются только типа TOPIC.
2. Очереди только Durable, без дополнительных флагов (все флаги «навешиваются» через политики).
3. Можно создать только маршруты вида Эксчендж -> Очередь. Маршруты Эксчендж -> не создаются.

Каждая команда парсится на отдельные шаги:

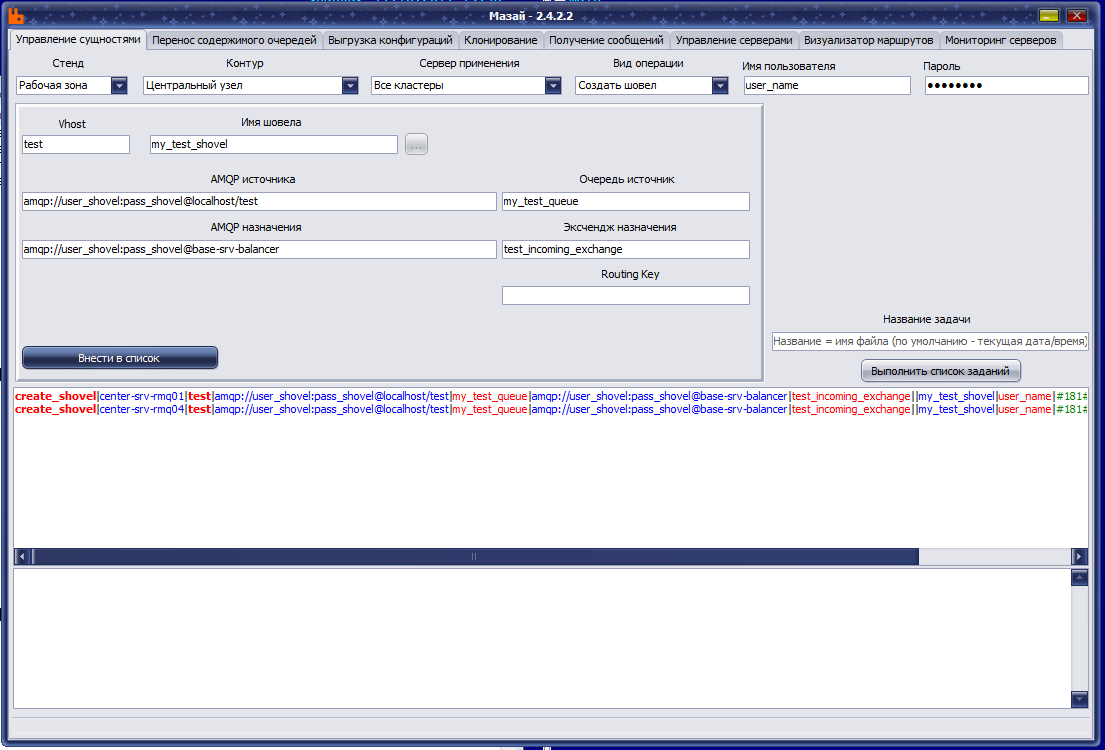
«create\_units» будет выполнять следующие действия:

1. Проверка на существование эксченджа.
2. Если его нет – будет создан эксчендж **type: topic**
3. Проверка существования очереди
4. Если очереди нет – будет создана очередь **durable=true, auto\_delete=false**
5. Если указаны несколько ключей, то по каждому из них будет вызвана API функция на создание (без проверки существования)

Все результаты будут выведены в поле лога.

**Создать Шовел**

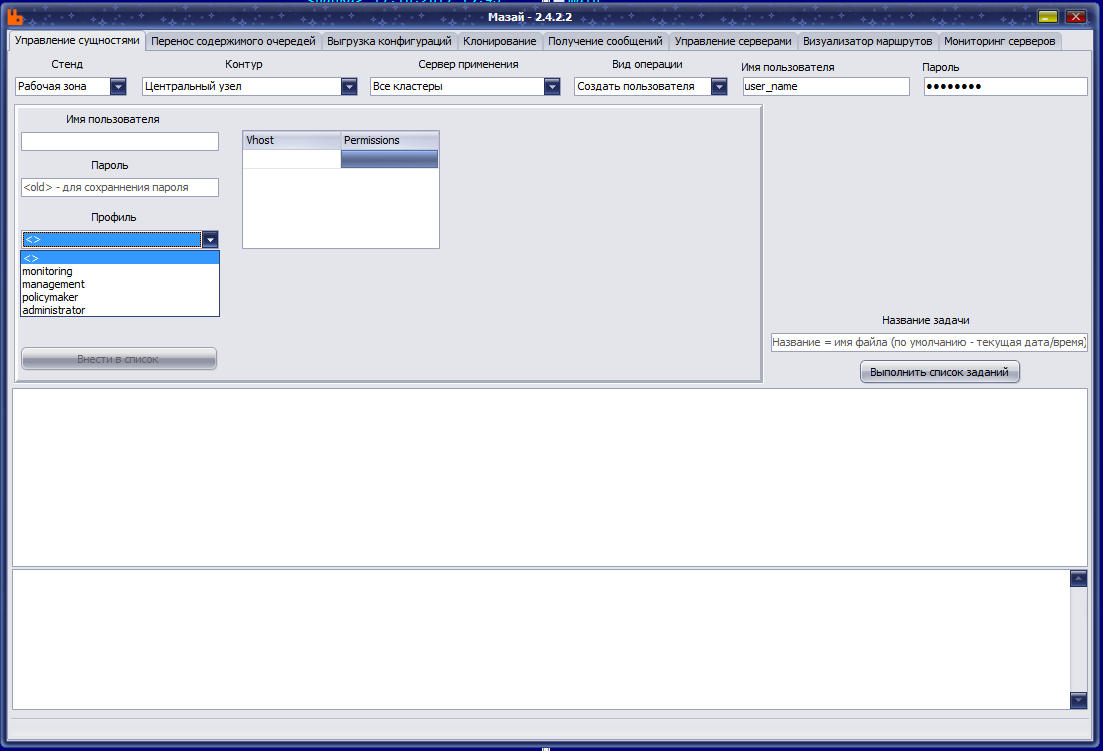
Для удобства работы со всем многообразием кроликов введён единый логин/пароль для работы всех шовелов, на всех кроликах. Значения логин/пароль указываются в коде (в последующих версиях, возможно, будет вынесено в конфигурацию для каждого кластера). Для создания доступны только шовелы с точкой входа Эксчендж. Шовелы «Очередь->Очередь» не создаются ввиду того, что даже такие шовелы на входе используют эксчендж, только временно сгенерированный.



После указания «VHOST» нажатие клавиши «Enter» приведёт к его добавлению в значение AMQP источника. Если указать значение Routing Key – то все сообщения, переданные этим шовелом из указанной очереди, будут соответствующим образом изменены.

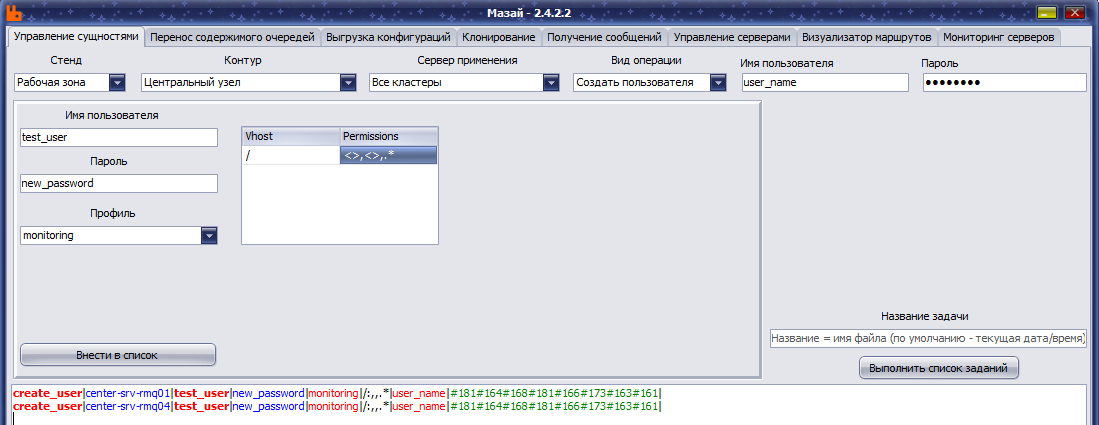
Данная команда не парсится на составляющие компоненты, а исполняется вызовом одной функции API.

**Создать пользователя**



Для VHOST возможен как ручной ввод прав, так и выбор из выпадающего спискаПрава: Первый элемент – Configure, второй – write, третий – read.

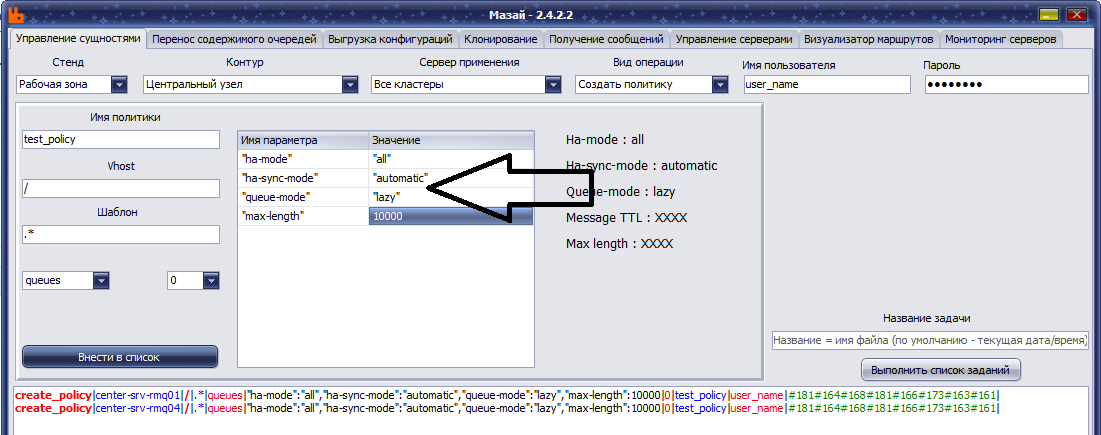
В отличии от WEB интерфейса самого кролика, в программе предусмотрен вариант изменения профиля пользователя без изменения (или указания) пароля. Сделано это через запрос и использование хэша пароля.



Выполнение данной команды пойдёт в несколько шагов:

1. Проверка существования пользователя (в этой же шаге запрашивается значение хэша пароля)
2. Если пользователя нет – создаётся
3. Если в поле пароля будет указано <old> - то будет использован хэш, полученный на первом шаге
4. Пользователю устанавливается указанный профиль и указанный пароль (или хэш текущего пароля). Аналогичное действие на WEB интерфейсе требует ввода пароля изменяемого пользователя.

**Создать политику**

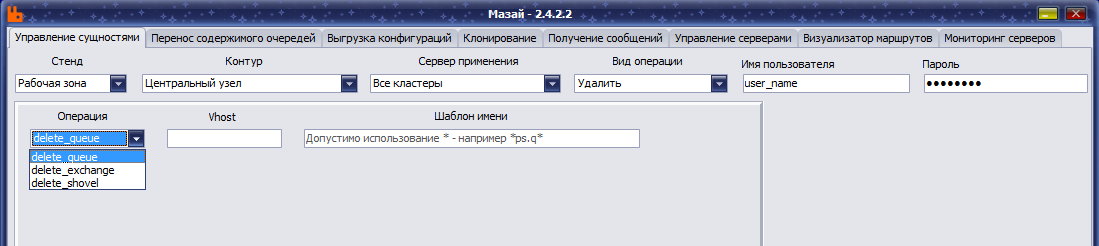


Реализована возможность “Drag And Drop” значений из перечня в применяемый список.

Функционал полностью идентичен управлению через WEB. Указывается имя политики, VHOST применения, шаблон (Reg Exp), Сущности для применения (queues/exchanges/all) и приоритет.

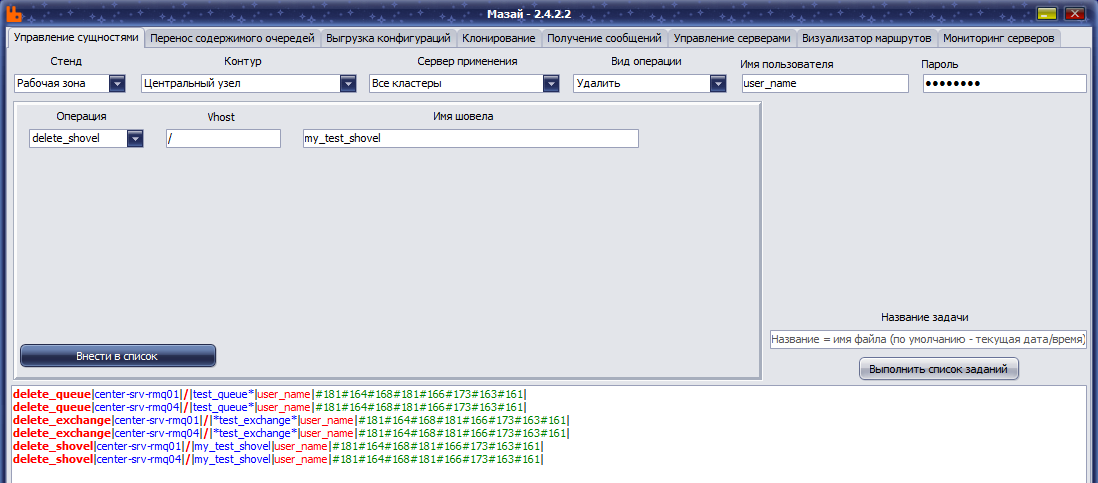
Заполняя список параметров следует помнить, что строковые параметры ОБЯЗАТЕЛЬНО указываются в двойных кавычках, цифровые значения указываются БЕЗ кавычек.

**Удалить**



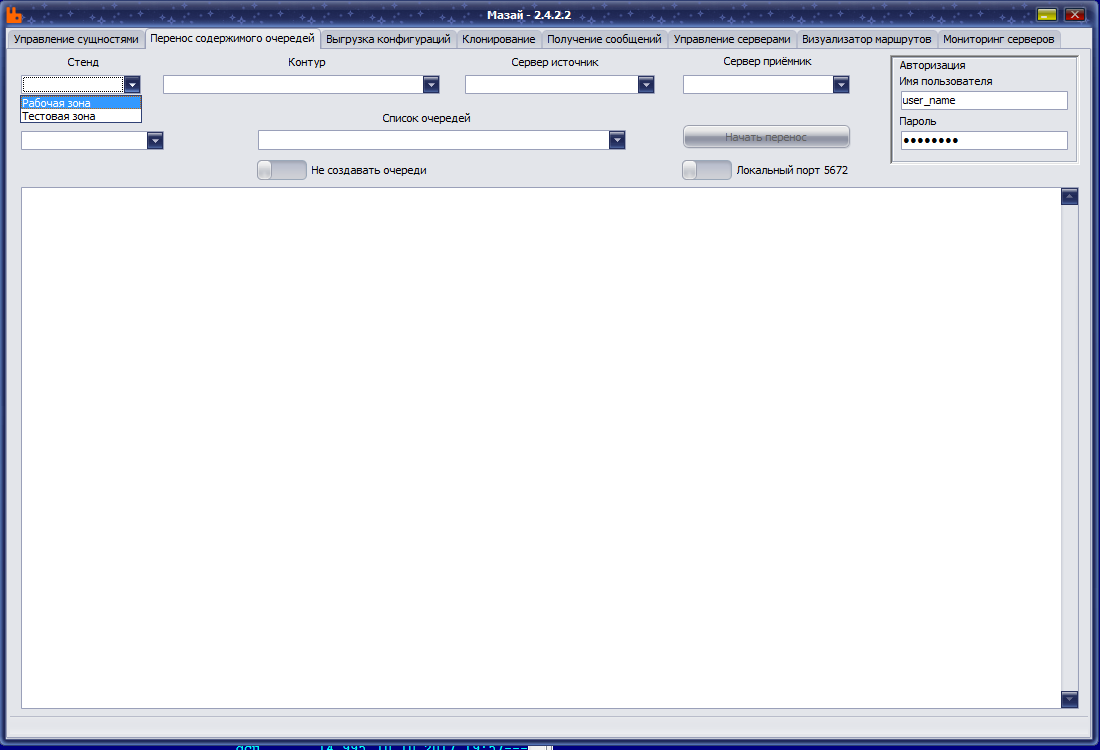
Доступны возможности удаления:

1. очередей и эксченджей по шаблону с использованием \* в начале и/или конце шаблона. Например \*test\_\* приведёт к удалению всех совпадений в которых встречаются “test\_”; шаблон test\_\* будет применим только к очередям/эксченджам начинающимся с test\_\*; шаблон \*test\_ - соответственно будут найдены только очереди/эксченджи заканчивающиеся на test\_. Более никакие спец. Символы не используются и их наличие будет рассматриваться как обычный символ. Так же не допускается использование \* в середине шаблона (возможно будет реализовано в последующих версиях программы).
2. Шовелов – по точному указанию имени



Данные команды будут выполнены для каждой найденной сущности. Ход выполнения будет записан в поле лога.

Вторая закладка «**Перенос содержимого очередей**»



Данная закладка предоставляет механизм переноса сообщений между кластерами одного контура. Применяется после переключения с основного на резервный или обратно, если на контуре остались невычитанные сообщения. Минимальное условие для работы функционала: Одинаковый набор VHOST на кластере источнике и приёмнике. Одинаковые учётные данные для обоих кластеров. Если не заполнены логин/пароль, список источников будет пустым, в таком случае необходимо указать учётные данные и повторить выбор контура. При выборе сервера источника выполняется запрос на этот сервер для получения списка всех доступных vhost. Первым пунктом в vhosts и в список очредей будет добавлена директива “All” – при выполнении с такой директивой будут последовательно проверены все VHOST и все очереди. Если выбран один VHOST и директива “All” указана в списке очередей, то будут проверены все очереди в указанном VHOST. В список попадают только очереди с ненулевым количеством сообщений. Запрос на очереди и количества выполняется при выборе VHOST.

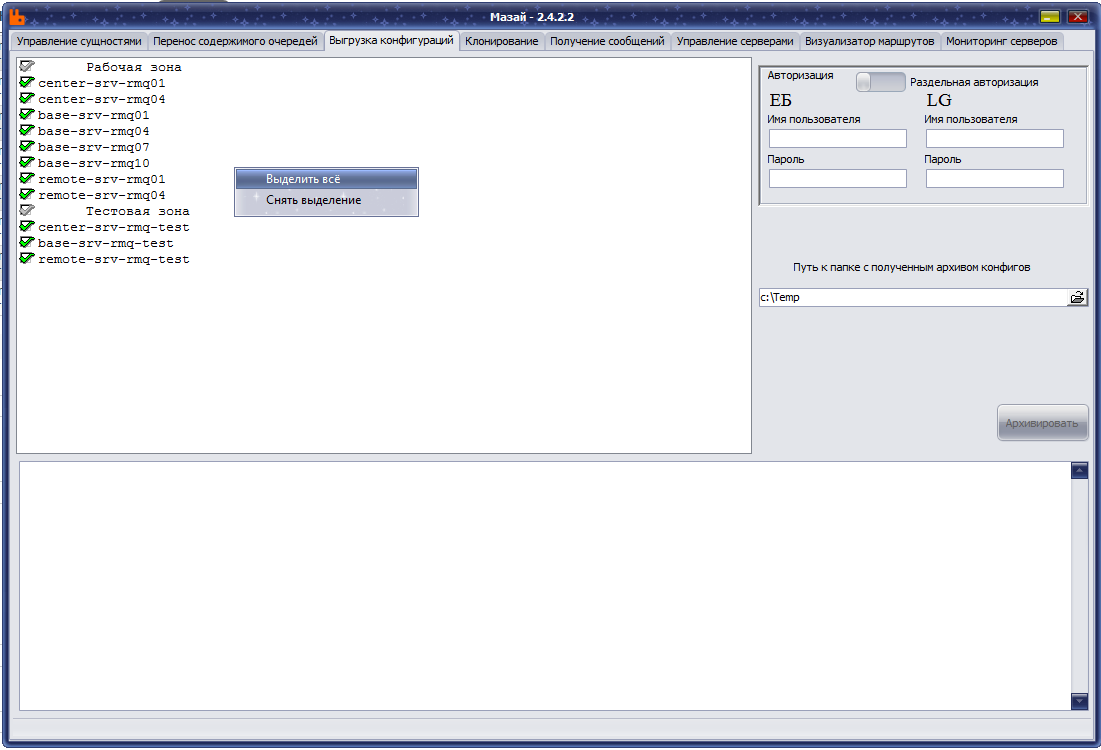
Дополнительно используются переключатели:

1. «Не создавать очереди/Создавать очереди в случае отсутствия» - позволяет на сервере приёмнике создать очередь с названием=очереди источника при её отсутствии. Если очереди нет и переключатель стоит в положении «Не создавать очреди» - перенос такой очереди будет пропущен.
2. «Локальный порт 5672/Локальный порт 5673» - используется при работе сервера источника на смещённом amqp порту. Данная ситуация реализуется при выводе основного кластера из под нагрузки – изменяются порты amqp на 5673, в результате шататный поток сообщений через балансировщики уходит на резервный кластер, при этом основной кластер остаётся доступен. Например подобное используется при восстановительных работах на основном кластере после падения.

Рассмотрим механизм переноса сообщений с включёнными директивами “All” и для VHOST и для Списка очередей.

1. Из списка берётся первый vhost, по нему выполняется запрос на сервер источник – получение всех очередей, у которых количество сообщений больше 0.
2. Последовательно для каждой из полученных очередей выполняются действия:
   1. На сервере приёмнике создаётся эксчендж **tmp\_transfer**
   2. При включенном переключателе «Создавать очереди в случае отсутствия», при отсутствии очереди, очередь, на сервере приёмнике, будет создана с именем равным очереди источника
   3. На сервере приёмнике вешается бинд по routing key “#” между tmp\_tranfser и целевой очередью
   4. На сервере источнике создаётся шовел от очереди источника до эксченджа tmp\_transfer на сервере приёмнике. На этом шаге обрабатывается переключатель «Локальный порт 5672/Локальный порт 5673» - у сервера источника указывается порт 5672 или 5673
   5. Раз в 5 секунд следует проверка очереди источника на оставшееся количество сообщений и вывод количества в поле лога
   6. Если сообщения ещё есть, следует очередное пятисекундное ожидание
   7. Так до тех пор, пока очередь источник не будет полностью передана на сервер приёмник. При работе на порту 5673 все остальные шовелы на сервере источнике будут находиться в состоянии Terminated, т.к. локальный порт 5672 не обслуживается
   8. После передачи всех сообщений – остаток сообщений 0, следует удаление шовела на сервере источнике и эксченджа на сервере приёмнике.
3. Дальше берётся следующий VHOST и повторяется шаг 2, до тех пор, пока не будет обработано всё содержимое сервера источника.

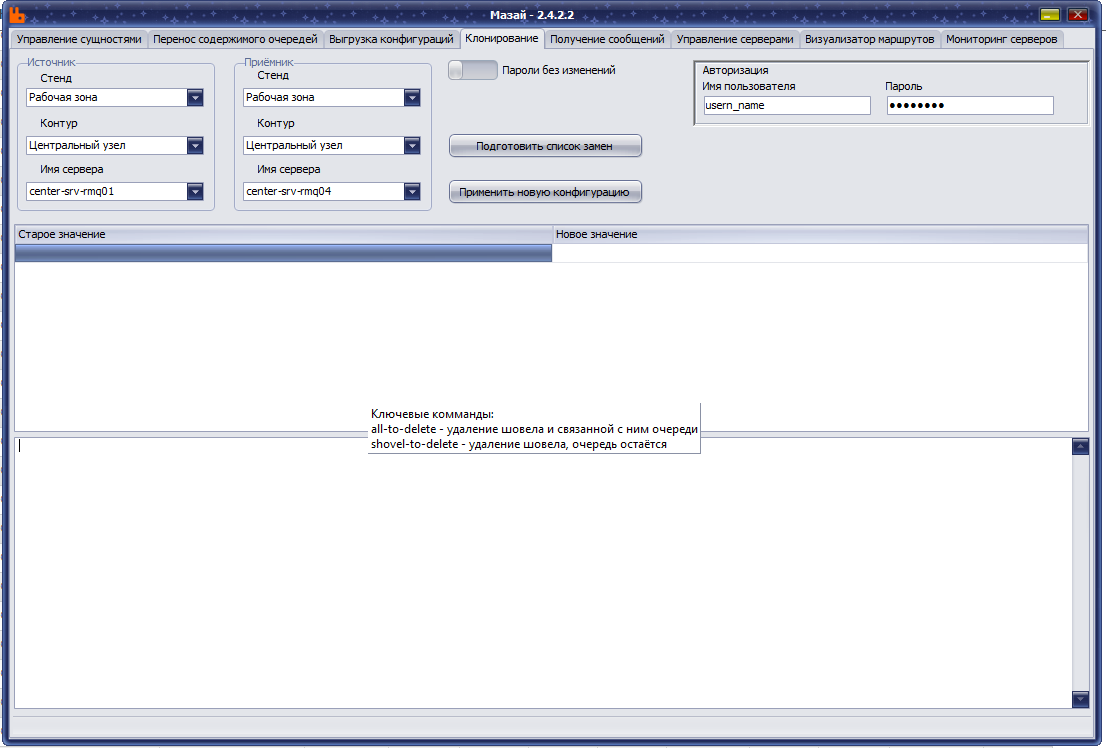
Закладка «**Выгрузка конфигураций**»



Предназначена для архивации выбранных серверов. Введена возможность раздельной авторизации на двух разных типах зон. Если имя зоны “LG”, то будет использована соответствующая пара логин/пароль. Во всех остальных случаях будет использоваться значение пары «ЕБ». (Данный функционал введён как временный, в последствии будет отключён). Контекстное меню позволяет выбрать или снять выделение со всех серверов списка. Список серверов загружается из файла конфигурации. Серой галочкой помечены названия зон. Двойной клик левой кнопкой на имени сервера приведёт к переходу на закладку «**Управление серверами**» и позиционированию на сервер в списке.

Механизм работы: Последовательно на каждый отмеченный сервер отправляется запрос (штатное API RabbtiMQ manager) на получения конфигурации в виде JSON, в указанной папке создаётся папка {ИмяКонтура} в которую сохраняется полученный конфиг в файл {имя сервера}.json. Затем происходит упаковка в ZIP архив с именем configs\_<CurrentDate>. На выходе – один архив со всеми конфигами всех кролей, управляемых программой.

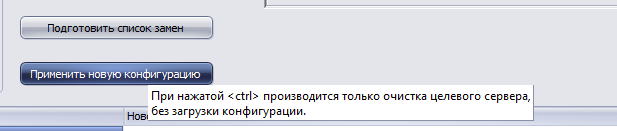
Закладка «**Клонирование**»



Данный функционал позволяет сделать точную копию указанного кластера.

Используется при периодической синхронизации настроек между кластерами одного контура. Например, какие-то изменения вносили на основной, а в это время резервный не работал и соответственно начал отличаться по настройкам. Можно повторить все изменения и на резервном кластере, воспользовавшись уже выполненными командами из истории. Но когда изменений много и какие-то ещё делались руками, то проще полностью перелить конфигурацию. Так же очень удобно клонировать настройки, например, на тестовый стенд, при этом срабатывает дополнительный функционал – из файла конфигурации срабатывают настройки source\_alias и destination\_alias.

**Как работает функционал.**

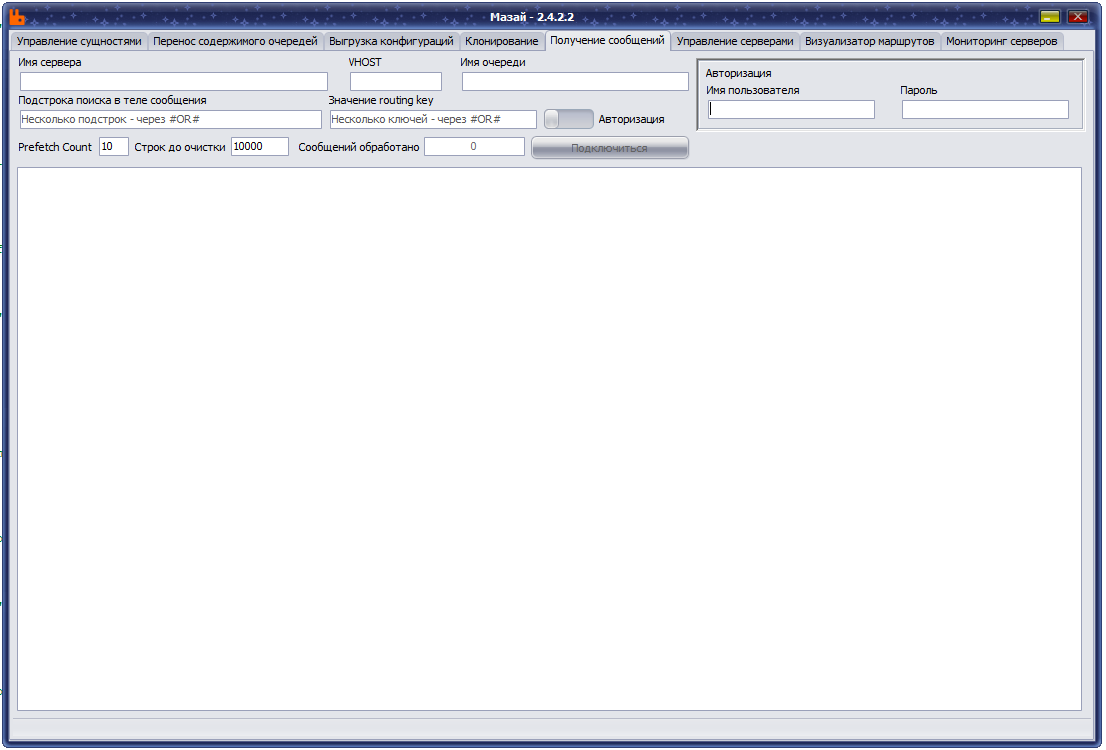
1. Нажатие на «Подготовить список замен» - вычитывает конфигурацию с сервера источника, вычитывает секцию шовелов и имена серверов из dest-uri прогружает в список замен в столбец «Старое значение».
2. Применяются настройки из файла конфигурации source\_alias – ищутся совпадения в списке «Старое значение» и при нахождении заполняется соответствующим значением destination\_alias и записывается в «Новое значение». Используется при клонировании настроек с промышленного стенда на тестовый – меняются названия серверов, использующихся в шовелах.
3. В столбце «Новое значение» доступны две директивы: **shovel-to-delete** – при использовании удалит все шовелы, со значением «Старое значение» в dest-uri. Используется при клонировании на тестовую зону, но на ней нет кролика, который бы соответствовал значению «Старое значение». **all-to-delete** при использовании удаляет не только шовел, но и очередь, из которой этот шовел вычитывает данные. Используется при клонировании для недопущения роста очередей, которые без вычитки шовелов будут расти. Или в случае, если на целевом кролике требуется только часть маршрутов, а остальные удаляются, чтобы не мешали.
4. Если поле «Новое значение» оставить незаполненным – оно будет приравнено к «Старому значению»
5. 

Применение конфигурации проходит через несколько шагов:

1. Пустые значение в «Новое значение» заполняются данными из «Старое значение»
2. Формируется json конфигурация, он выводится в поле лога (если его загрузка приводит к ошибке этот json можно загрузить руками, например, через Postman и найти, где ошибка)
3. Происходит последовательное удаление vhost и пользователей на целевом сервере. Этот шаг можно выполнить изолированно: на «Применить новую конфигурацию» левая кнопка мыши с нажатой клавиша <ctrl>.
4. На сервер приёмник загружается полученный json кофигурации (используется штатное API кролика)
5. Если используется переключатель «Пароли без изменений/<Пароль> = <Логин>123», то по всем пользователям (кроме пользователя для системы мониторинга – в нашем случае snmp\_int\_user) последовательно будет произведена замена пароля – он станет равным логину с добавлением 123. Используется при клонировании на тестовые зоны – исключает возможность подключения к тестовым кроликам с промышленными настройками подсистем и наоборот.
6. Обрабатываются директивы **shovel-to-delete** и **all-to-delete** сначала удаляются шовелы, затем очереди.

Результаты выполнения всех шагов выводятся в окно лога.

Закладка «**Получение сообщений**»

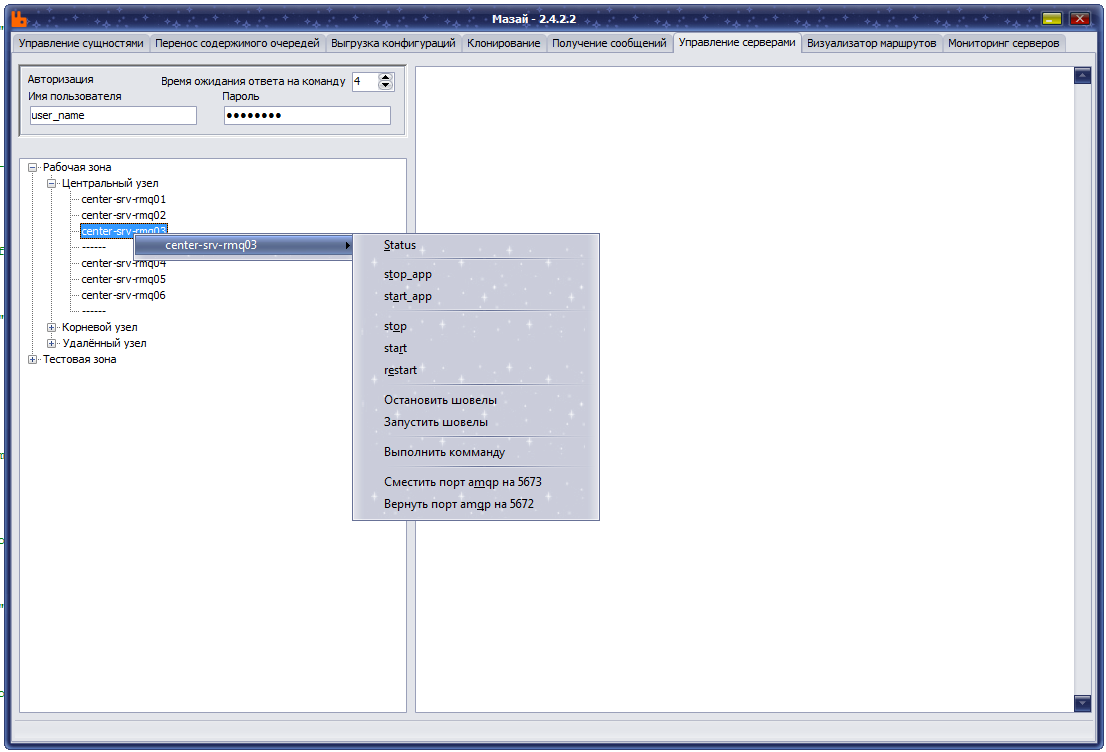


Данный функционал использует библиотеку rabbitmq-c для подключения по amqp к произвольному серверу. Переключатель «Авторизация/Вычитка очереди»: Проверка авторизации под указанными учётными данными или полноценное подключение к очереди по протоколу amqp и получение сообщений. Данный функционал позволяет производить в вычитываемых сообщениях поиск по совпадениям значений как в теле сообщений, так и в значениях routing key. Если в имени сервера не указан порт, то подключение будет произведено по стандартному порту 5672. Допускается использование нескольких значений, разделённых логическим оператором OR. Например, “text\_message 1#OR#msg\_2”, в поиске по телу сообщения, будут выведены только сообщения в теле которых найдено или text\_message 1 или msg\_2. Остальные сообщения будут удалены. При заполнении обоих поисковых полей, поиск происходит по формуле: <совпадение в теле сообщения> И <совпадения в ключе маршрутизации>. При этом в каждом поисковом поле допускается использовать #OR# для нескольких значений. При необходимости значение Prefetch Count можно изменить для более оптимальной скорости обработки. Поле «Строк до очистки» - значение срок в поле вывода, по достижении которого произойдёт очистка поля вывода, останется последняя 1000 строк и происходит освобождение оперативной памяти, занимаемой программой. При большом количестве строк в поле вывода программа может упасть по переполнению памяти. Для 32 разрядной версии приложения 1700 Мб будет пороговым значением.

Использование: Поиск редких сообщений в большом общем потоке. Реализация такой задачи примерно следующая:

1. Создаётся копия входящего потока в отдельную очередь. Не лишним будет «повесить» на эту очередь ограничение например по количеству сообщений.
2. Настраивается подключение к этой очереди и заполняются искомые значения.
3. Производится подключение и вычитка сообщений.
4. После нахождения искомого, вычитку останавливаем и не забываем удалить временную очередь, если её использование больше не требуется.

Закладка «**Управление серверами**»



Данная закладка работает при подключении через SSH с использованием библиотеки Chilkat. Требуется использование учётных данных, обладающих возможностью подключения к серверам кроликов по ssh. Реализуемый функционал делится на две части:

1. Функции, исполняемые под учеткой rabbitmq (Учетная запись по умолчанию, создаваемая при установке RabbitMQ)
2. Функции, исполняемые под учетной запись с которой подключились. (В скриншоте – user\_name)

Блок функций исполняемый под “sudo - iu rabbitmq”

1. Status – выполнение и вывод на экран результата запуска «rabbitmqctl status»
2. stop\_app и start\_app – результат выполнения rabbitmqctl stop\_app / rabbitmqctl start\_app
3. stop/start/restart – результат выполнения скрипта rabbitmq-server stop/start/restart из init.d
4. Остановить/Запустить шовелы – выполнение rabbitmqctl eval 'application:stop(rabbitmq\_shovel).' И rabbitmqctl eval 'application:start(rabbitmq\_shovel).' соответственно. Данный функционал позволяет остановить/запустить плагин шовелов, не прибегая к рестарту приложения.
5. «Сместить порт amqp на 5673» и «Вернуть порт amqp на 5672» - выполнение последовательности команд, приводящих к добавлению параметра RABBITMQ\_NODE\_PORT=5673/ RABBITMQ\_NODE\_PORT=5672 в файле скрипта **rabbitmq-env** – срабатывает при запуске/перезапуске кролика

Блок функций, исполняемых от имени пользователя:

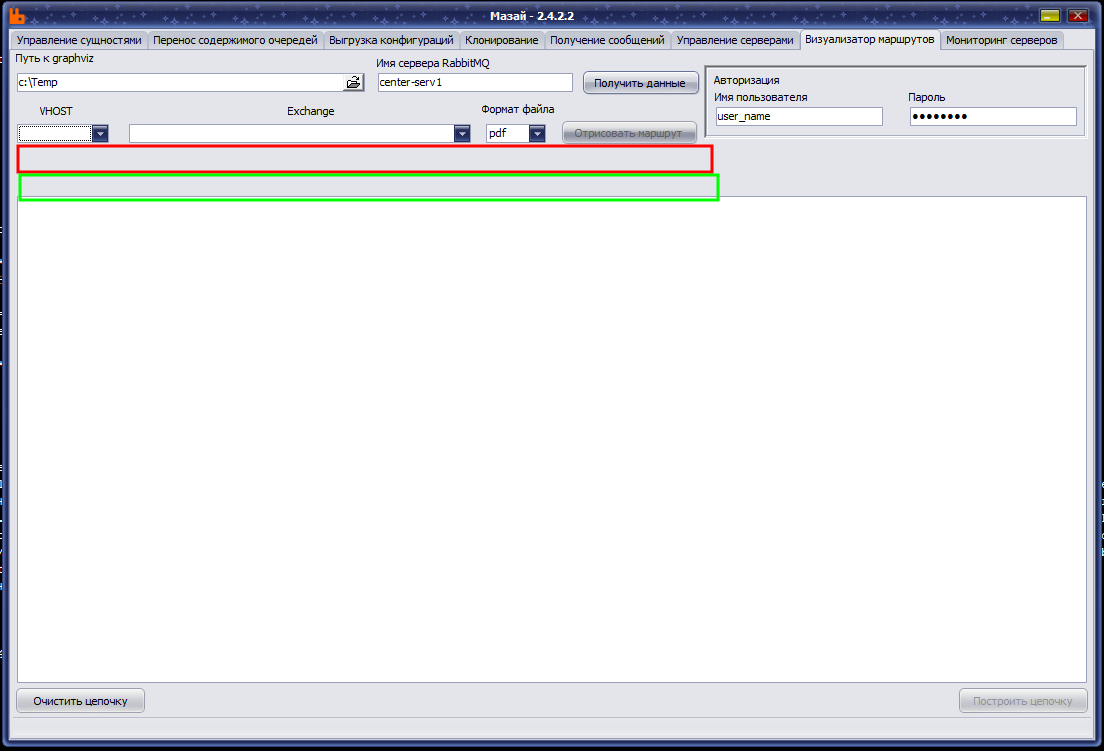
Фактически функция одна: выполнение одной или нескольких произвольных команд, разделённых «;». Сделана отдельная обработка результата работы команды telnet, поскольку она ожидает ввода данных.

На все команды действует устанавливаемый в секундах параметр: 

Если какая-то команда планируется выполняться дольше – следует увеличить значение таймаута с запасом.

В пункте «Выполнить команду» будет происходить запись последней выполненной команды. Полезно, когда требуется выполнять одни и те же действия на многих серверах кроликов.

Закладка «**Визуализатор маршрутов**»



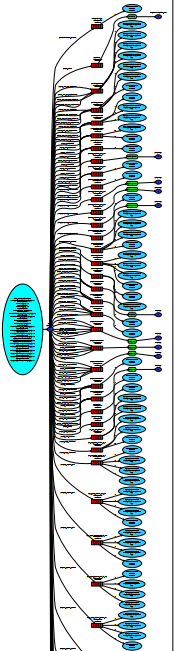
Данный функционал генерирует файл .dot для дальнейшего преобразования с помощью программы graphvis. На выходе получится PDF с графически отрисованным маршрутом сообщений. Маршруты строятся от эксченжей. Отображаются шовелы. Так же есть возможность построить «Цепочку» - полный маршрут прохождения сообщений через шовелы на разных кроликах. Таким образом от точки публикации можно визуально увидеть все возможные пути сообщений. Для работы требуется учетная запись с правами администратора RabbitMQ, т.к. используется штатное API кролика.

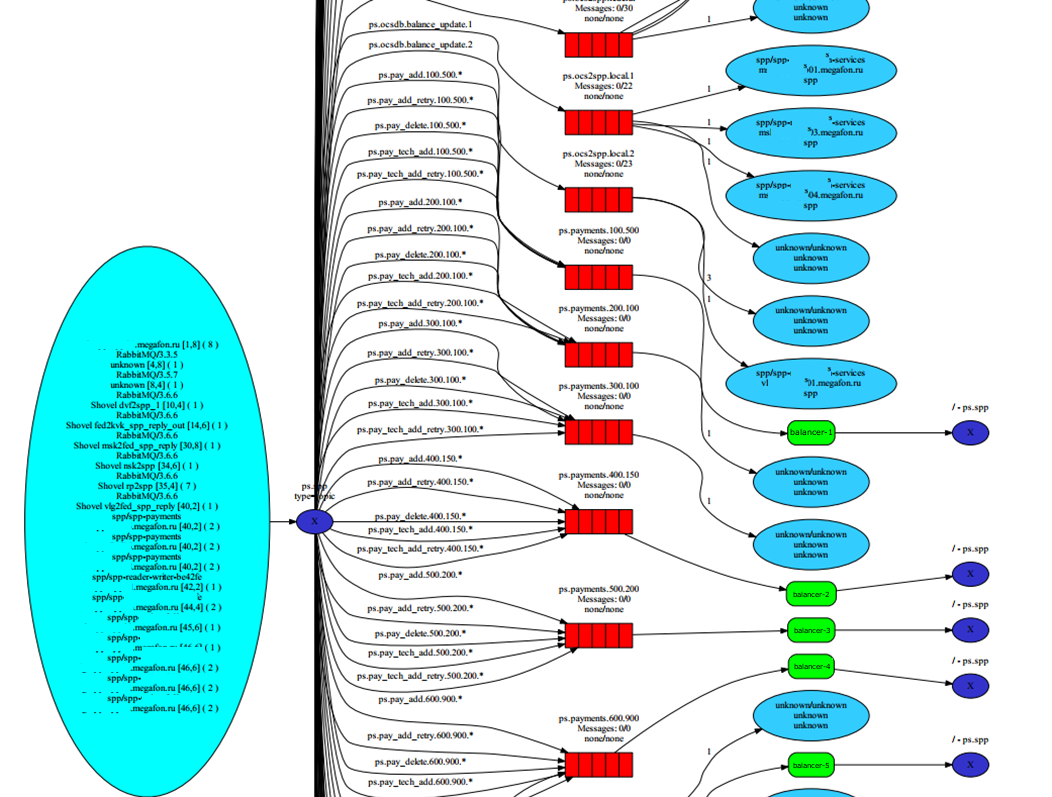
**Описание работы функционала**:

После нажатия на «Получить данные» список VHOST будет заполнен всеми доступными VHOST, полученными с указанного сервера. Первым пунктом автоматически добавляется: «Все VHOSTs». Выбор этого пункта приведёт к последовательному построению маршрутов от всех эксченджей всех VHOST выбранного кролика. При выборе любого из доступных vhost из списка, загружается список всех эксченджей этого vhost-а. Первым пунктом автоматически добавляется вариант «Все эксченджи». Выбор этого пункта приведёт к последовательному построению всех маршрутов от всех эксченджей выбранного VHOST.

Путь к сгенерированному файлу .dot будет отображаться в верхней строчке (выделено красным). Путь к полученному файлу (pdf/svg/ps2) отображается во второй строке (выделено зелёным). При двойном клике на этой строке полученный файл будет открыт связанным приложением.

Для построения используется следующий алгоритм:

1. При выборе всех «Все VHOSTs» следующие шаги делаются для каждого VHOST последовательно
2. При выборе «Все эксченджи» следующие шаги будут выполнены для каждого последовательно.
3. На указанный кролик отправляется запрос на получение всех биндов, указанных у эксченджа
4. Если бинд связан с очередью, срабатывает функционал интеграции с базой мониторинга (при использовании и указании в файле конфигурации) и по этой очереди запрашиваются данные по скорости публикации и скорости вычитки за последние 4 минуты с поминутной детализацией.
5. Определяется есть ли у очереди подписчики.
6. Поскольку при проектировании архитектуры кроликов все шовелы работают в режиме отправки сообщений и src-uri у всех шовелов – localhost, то в списке подписчиков их легко определить – IP адрес 127.0.0.1.
7. Если подписчик шовел – то он соответствующим образом помечается (зелёным цветом в полученном файле) и отображается VHOST и экчендж/очередь в которую этот шовел отправляет сообщения. В качестве имени отображается имя сервера, куда этот шовел «ведёт».
8. Если подписчик какая-либо подсистема, то из данных по коннекту пробуем получить информацию из Client properties. (Для всех подсистем, работающих с кроликом настоятельно рекомендуем эти данные заполнять – сильно облегчает работу и поиск сбоев). Если подписчик один, но работает в несколько потоков, то данные суммируются
9. Полученные данные фиксируются.
10. При включённом режиме detailed (в файле конфигурации самого RabbitMQ), получаются данные так же по всем, кто публикует в данный эксчендж. Если вместо адреса публикатора, по каким либо причинам отображается пустое значение, то происходит запрос всех коннектов и всех каналов на кролике и в полученных данных производится поиск искомых публикаторов. Если число коннектов и каналов велико, то их опрос может занять достаточно продолжительное время, однако, это только при первом построении. Пока не произведено заново «Получение данных» - список хранится в памяти и по следующим эксченджам запрос к кролику не отправляется, а используется уже внутренние закэшированные данные, соответственно скорость обработки очень существенно вырастает.
11. По публикаторам так же идёт попытка получения данных их Client properties.
12. Данные по скорости публикации так же фиксируются. Если публикация идёт с одного коннекта, но в несколько потоков, данные суммируются и отображается количество публикаторов и суммарная интенсивность публикации
13. Все собранные данные записываются в файл формата .dot
14. Производится вызов GraphViz на вход которой и подаётся полученный файл. В результате работы получаем файл, например, PDF. Выглядит он примерно так:

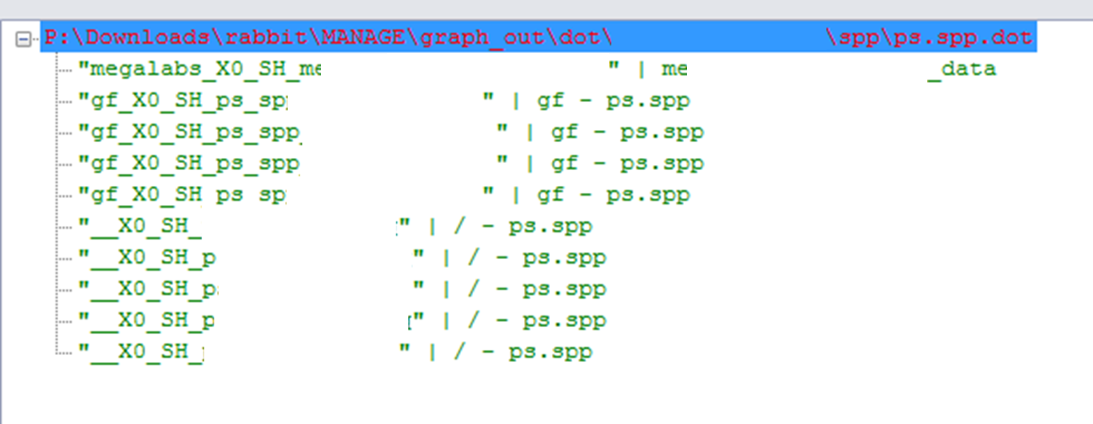


Таким образом даже сложные маршруты визуально становятся вполне понятными.

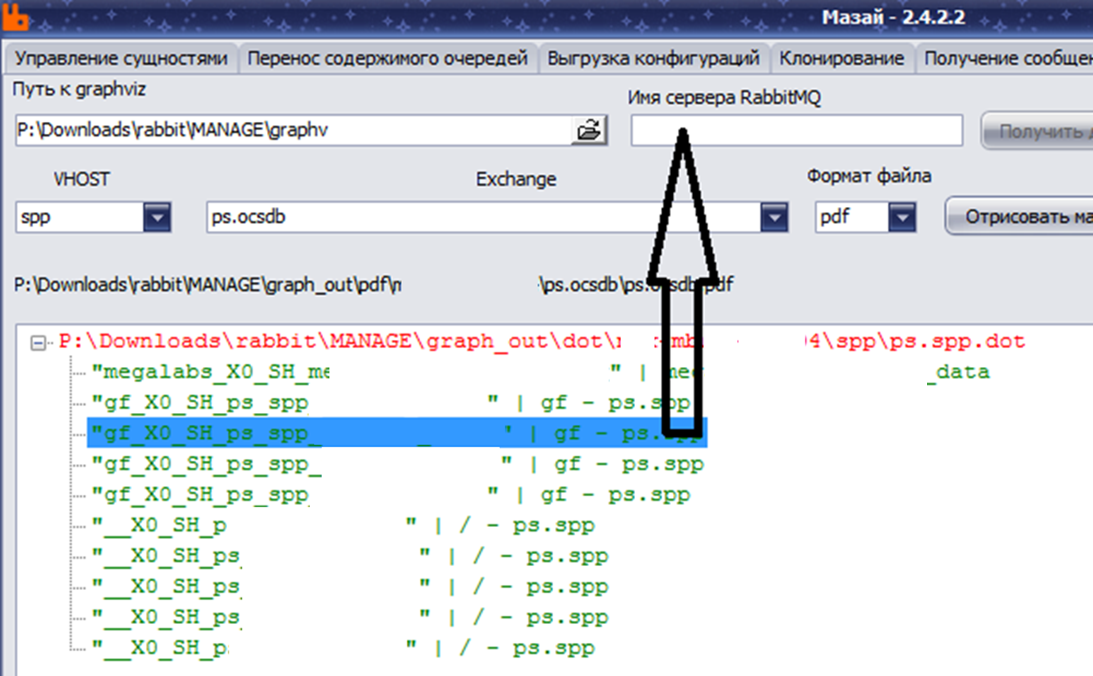
Но это отображает маршрут только в пределах одного кластера, теперь переходим к построению всей цепочки – полный маршрут через произвольное количество шовелов. Для этого использую Drag And Drop перетаскиваем dot файл в поле построения цепочки.



Если у нас есть сформированный ранее файл, то его можно перетащить в поле цепочки, путь к файлу и имя будут отображены всё в той же строке, после чего так же перетаскиваем из строки в поле цепочки наш файл. В контекстном меню выбираем «Определение исходящих шовелов». Будет проанализирован dot файл на наличие шовелов и все они будут добавлены в список в виде дерева.

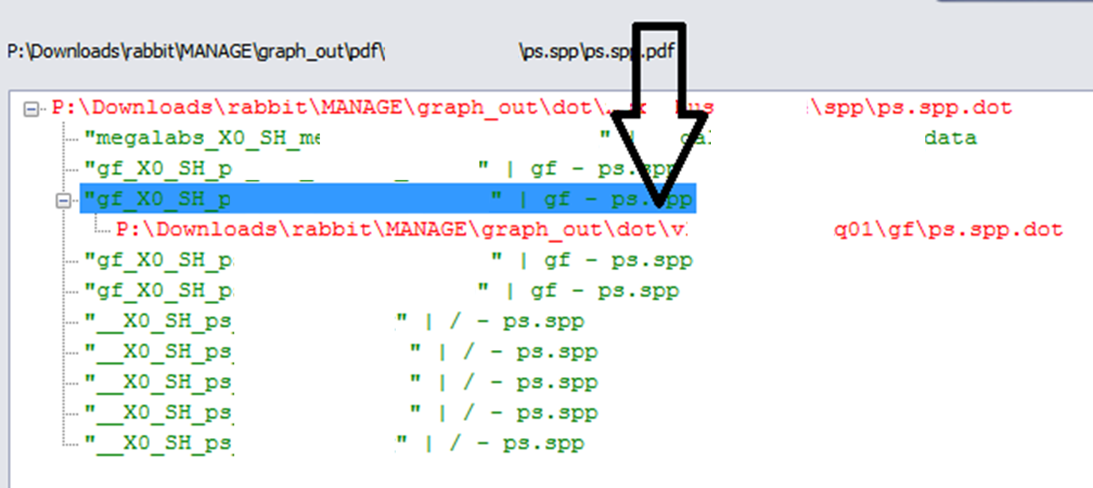


Следующий шаг – построить маршрут по каждому определившемуся серверу и эксченджу. Поскольку все сервера кроликов описаны в файле конфигурации программы, перетаскиваем соответствующую запись шовела в поле адреса:



Из файла конфигурации определяется сервер кролика к которому надо подключиться и полученное значение записывается в поле адреса.

Теперь просто повторяем шаги по построению маршрута и прицепляем полученный файл к ветке шовела. Если файлы уже ранее сформированы, можно их просто перетащить на форму и так же прицепить к шовелу. Глубина вложений не ограничена.

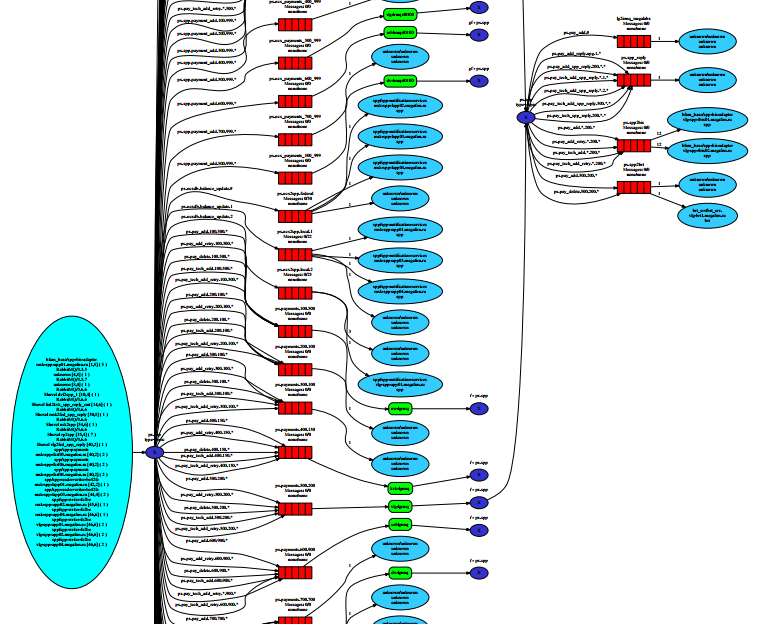


По прицепленному файлу так же можно запустить «Определение исходящих шовелов», так заполняются все ветки – подготавливается данные по построению всё цепочки. Если файл прицеплен ошибочно – его можно удалить: клавиша <Del>. При этом из списка будут так же удалены все прицепленные к нему файлы – все потомки в дереве.

Построить цепочку можно на любом этапе, даже не заполняя данные по всем шовелам. Нажатие на «Построить цепочку» приведёт к последовательному анализу всех используемых файлов dot и построению файла цепочки. Так же в файл будет сохранено получившееся дерево. Этот файл так же можно в последующем перетащить на форму и продолжить работу с ним.

Все сгенерированные файлы сохраняются в папке “graph\_out” в папке с программой. Папка “dot” содержит разложенные по серверам/vhost-ам файлы .dot. Папка “pdf”/ “svg” / “ps2” содержит построенные файлы маршрутов соответствующих форматов. Папка “chains” хранит построенные файлы цепочек .dot, .list – файл дерева, .pdf/.svg/.ps2 полученные файлы маршрутов.

Получиться может нечто такое:



Таким образом визуально можно не только определить какие маршруты проходят от или через начальных эксчендж, но и оценить интенсивность потока сообщений на момент построения маршрутов.

Закладка «**Мониторинг серверов**»

Данный функционал построен на опросе балансировщиков haproxy и отображению их состояния. Используется для получения мгновенного среза текущего состояния всех серверов кроликов, описанных в файле конфигурации. Нажатие на «Заполнить список» приведёт к последовательному опросу всех haproxy, данные по которым указаны в файле конфигурации - **statistic\_point**. Если данные по балансировщику не заполнены, то в список будет выведено только название зоны.



В список загружаются данные, полученные с haproxy, указанного в конфигурации каждого контура. Имена серверов и их состояние берётся не из файла конфигурации, а непосредственно с сервера haproxy. Таким образом, если имя сервера (в файле конфигурации) **central-srv-rmq01**, но в haproxy он прописан под именем **central-srv1**, то и в список он попадёт под именем **central-srv1**.