Инструкция по установке Monq. Версия 7.11.0.

MDL

2023-09-19

Contents

Пошаговая инструкция по запуску Monq **1**
Схематический план развертывания Monq . . . 2

Проектирование решения **3**
Demo образ . . . 3
Самостоятельное развертывание Monq . . . 3
Self-Hosted или Managed сервисы . . . 3
Отказоустойчивость и балансировка нагрузки . . . 5
Кластеризация . . . 6
Рекомендуемые конфигурации развертывания . . . 6
Заключение . . . 7

Планирование ресурсов **8**
Минимальные требования для Demo-VM . . . 8
Минимальные требования . . . 9
Предпродуктивный контур . . . 9
Продуктивный контур . . . 9

Подготовка СПО **11**
Kubernetes . . . 13
Managed kubernetes . . . 13
SelfHosted kubernetes . . . 14
Последующая настройка кластера . . . 14
Container registry. . . . 15
Consul. . . . 16
Кластерный DNS. . . . 16
СУБД, кеш и диспетчер сообщений . . . 17
PostgreSQL . . . 18
Clickhouse . . . 19
ArangoDB . . . 19
VictoriaMetrics . . . 20

Redis . . . 21
RabbitMQ . . . 22
Заключение . . . 22

Приложение A1. Инструкция по наполнению authfile. 24
Описание полей файла авторизации в компонентах СПО. . . . 24
Пример файла system_auth.json . . . 28

Приложение A2. Подготовка серверов. 29
Debian 11 . . . 29
Настройка сервера для роли control plane/master . . . 29
Настройка сервера worker . . . 31

Приложение A3. Запуск Kubernetes для Monq. 33
Режим SingleMaster. . . . 33
Режим MultiMaster. . . . 34
Подключение рабочих нод . . . 35
Запуск CNI . . . 36
Cilium. . . . 36
Flannel . . . 37
Запуск ingress-nginx controller. . . . 37
Создание пространств. . . . 38
Настройка хранилища файлов на базе NFS. . . . 38
Выпуск токена для установки monq. . . . 40

Приложение A4. Запуск репозитория контейнеров на базе Docker Registry. 42

Приложение A5. Consul. 46

Приложение A6. DNS в работе Monq. 52
Список используемых доменных имен . . . 52
Специализированная настройка внутрикластерного DNS. . . . 53

Приложение A7. PostgreSQL 55
Standalone PostgreSQL . . . 55
HA cluster PostgreSQL . . . 60

Приложение A8. Clickhouse. 64
Standalone Clickhouse . . . 64

HA cluster Clickhouse 69
Zookeeper 69
Clickhouse 70

Приложение A9. ArangoDB. 74
Standalone ArangoDB 74
HA cluster ArangoDB 79

Приложение A10. VictoriaMetrics. 82
Standalone VictoriaMetrics 82
HA cluster VictoriaMetrics 86

Приложение A11. Redis. 91
Standalone Redis. 91
HA cluster Redis. 96
Проверка соединения с redis в режиме HA. 98

Приложение A12. RabbitMQ. 101
Standalone RabbitMQ. 101
HA cluster RabbitMQ. 105

Приложение A13. Установка helm. 109

Приложение A14. Список системного программного обеспечения. 110

Установка Monq 111
Подготовка к установке 111
Запуск сценария установки 112
Вне кластера k8s 112
Внутри кластера k8s 113
Запуск сценария удаления 114
Заключение 114

Приложение B1. Импорт образа установщика. 115
Импорт образа из репозитория разработчика 115
Экспорт образа в личный репозиторий контейнеров 115

Приложение B2. Импорт образов Monq из репозитория разработчика. 117

Приложение B3. Список изменяемых переменных сценария установщика. 120

Пошаговая инструкция по запуску Monq

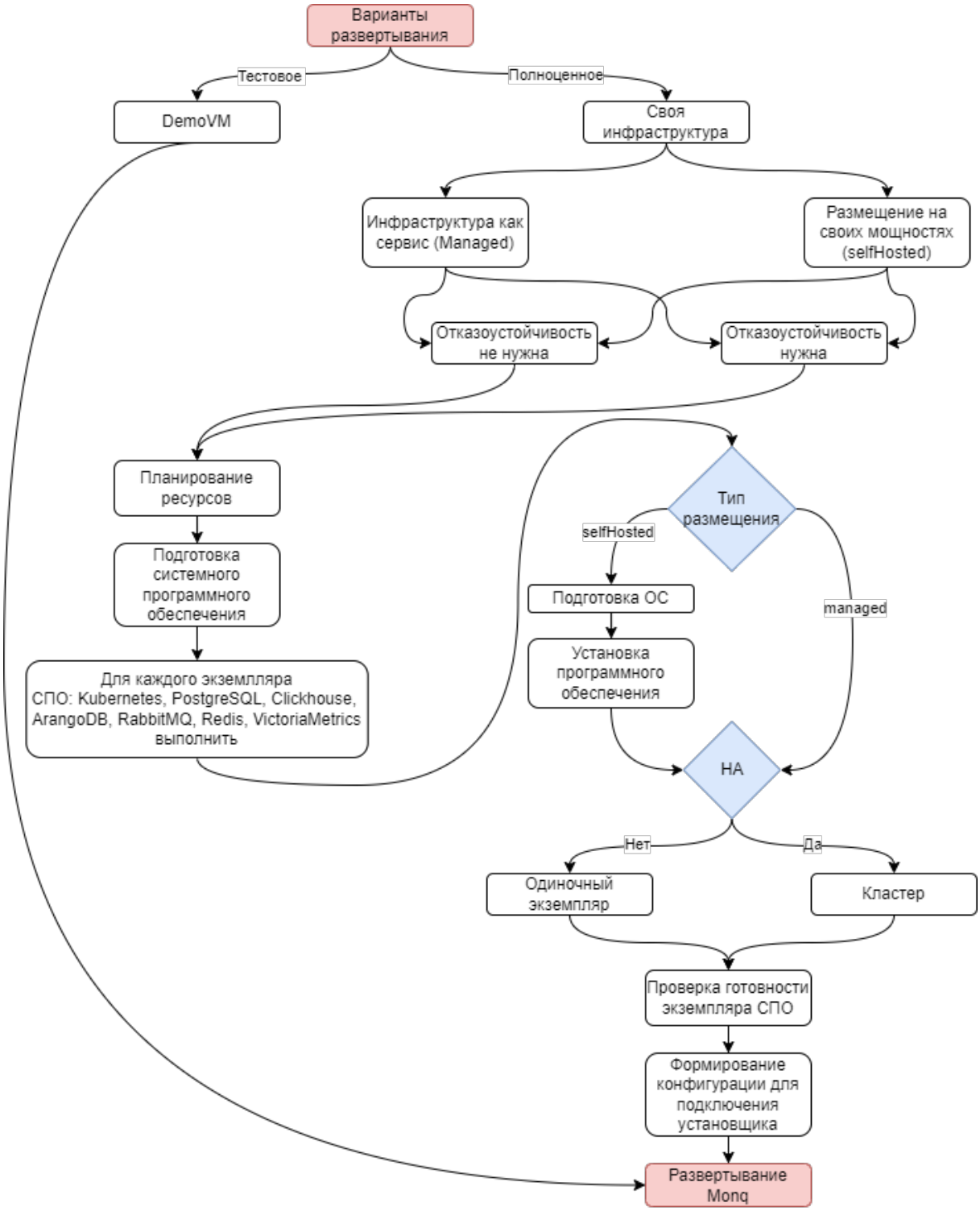
Данная инструкция содержит информацию, необходимую для подготовки инфраструктуры и последующего запуска Monq.

- 1. Проектирование решения;
- 2. Планирование ресурсов;
- 3. Подготовка СПО;
- 4. Развертывание ППО.

Дополнительные материалы

- Общий план развертывания Monq.

Схематический план развертывания Monq



Проектирование решения

Demo образ

Одна Виртуальная машина (далее ВМ) с предустановленной инфраструктурой.

В случае выбора данного варианта необходимо скачать образ ВМ в формате OVF с сайта по ссылке https://monq.ru/download, импортировать в свою виртуальную среду и установить Monq согласно инструкции.

> Внимание! Образ ВМ предоставляется в ознакомительных целях, при его сборке не закладывалась возможность масштабирования.

Самостоятельное развертывание Monq

В данном случае подразумевается самостоятельное планирование и подготовка инфраструктуры для запуска Monq.

Для реализации потребуется:

- ознакомиться с инструкцией по установке;
- подготовить одну или несколько ВМ с операционной системой поддерживаемого типа согласно инструкции;
- выполнить установку и настройку инфраструктуры;
- выполнить импорт контейнеров в локальный репозиторий;
- скачать установочный пакет;
- установить Monq.

Self-Hosted или Managed сервисы

Monq базируется на следующих компонентах системного программного обеспечения:

- Kubernetes - оркестратор контейнеров;

- PostgreSQL - СУБД для хранения системной информации;
- Clickhouse - СУБД для хранения событий;
- ArangoDB - СУБД для хранения информации о PCM и связях;
- RabbitMQ - брокер сообщений для межмикросервисного взаимодействия;
- Redis - кэширующий сервер;
- VictoriaMetrics - СУБД для хранения метрик;
- Consul - хранилище конфигураций микросервисов.

Каждый из компонентов может предоставляться как Self-Hosted решение (развернуто на своих серверах) или как Managed сервис (IaaS).

Self-Hosted

В данном случае необходимо развернуть инфраструктурные компоненты на своих мощностях: подготовить серверы, установить системное программное обеспечение (далее СПО) согласно списку.

При этом различные компоненты СПО могут быть как Self-Hosted, так и Managed.

Со стороны Monq нет явных ограничений, в каком виде они будут развернуты - это может быть установка на голом железе, в виде отдельного контейнера или внутри кластера Kubernetes с помощью оператора. Главное: чтобы все точки подключения к компонентам СПО были доступны из кластера.

Managed сервис

Следует учитывать, что Monq может работать не со всеми IaaS решениями, т.к. в некоторых случаях Monq создает подчиненные сущности в инфраструктурных объектах, такие как:

- пользователи;
- сущности для хранения информации, базы данных, очереди;
- права на доступ к сущностям.

Не все провайдеры предоставляют доступ к управлению IaaS в необходимом виде (например, Yandex Cloud пока такой возможности не дает).

Поэтому при выборе Managed решений необходимо проверить совместимость:

- Kubernetes - возможность создания пользователя k8s с правами на добавление/просмотр/удаление всех сущностей в namespace через API.
- PostgreSQL - возможность создания пользователя для выполнения запросов в СУБД:

  - создание/удаление БД;
  - создание/удаление пользователей;
  - назначение владельцев для созданных БД;
  - назначение прав для пользователей в БД.

- Clickhouse - возможность создания пользователя для выполнения запросов в СУБД:
  - назначение пользователю прав `access_management`;
  - создание/удаление БД;
  - создание/удаление пользователей;
  - назначение прав для пользователей в БД.

- ArangoDB - возможность создания пользователя для выполнения запросов в СУБД:
  - создание/удаление БД;
  - создание/удаление пользователей;
  - назначение прав для пользователей в БД.

- RabbitMQ - возможность создания пользователя для выполнения запросов через API:
  - создание/удаление пользователей;
  - назначение ролей пользователям;
  - назначение прав для пользователей в очередях.

- Redis
  - доступ к базам данных DB 0-4;
  - авторизация без ACL, только по паролю (Legacy Authentication Method).

- VictoriaMetrics
  - возможность чтения/записи в БД.

Отказоустойчивость и балансировка нагрузки

В текущей реализации Monq нет встроенной поддержки работы с СПО в режимах балансировки нагрузки.

При работе СПО в режиме высокой доступности (HA) балансировка запросов и проверка жизнеспособности экземпляра СПО осуществляется сторонними средствами, например HA-Proxy или service Kubernetes (если СПО запущено с помощью операторов).

Monq поддерживает работу в режиме высокой доступности со следующими компонентами СПО:

- Kubernetes - master-master;
- PostgreSQL - master-slave;
- Clickhouse - master-master;
- ArangoDB - active-failover (master-slave);
- RabbitMQ - master-master;
- Redis - master-slave;
- VictoriaMetrics - master-master.

Для каждого из компонентов определение работоспособности и балансировка запросов к работоспособному экземпляру осуществляется сторонними средствами, за исключением Redis: для него балансировка осуществляется с помощью sentinel.

Пример запуска Monq с СПО в режиме HA будет рассмотрен более подробно в документации по развертыванию.

Кластеризация

Под кластеризацией или шардированием понимается частичное разнесение информации между компонентами СПО.

В настоящий момент Monq не поддерживает кластеризацию данных, при проектировании решения это необходимо учитывать.

Рекомендуемые конфигурации развертывания

- Пробная инсталляция: предполагает запуск на одной ВМ. Подготовленный образ ВМ можно скачать с нашего сайта https://monq.ru/download;
- Предпродуктивный контур: рекомендуется запуск на трех ВМ. Данная конфигурация позже позволит масштабировать предпродуктивный контур в продуктивный.
  - сервер для хостинга k8s control-plane;
  - сервер для СПО;
  - сервер для прикладного программного обеспечения (далее ППО) Monq.
- Продуктивный контур:
  - один или несколько серверов для хостинга k8s control-plane;
  - три и более сервера для запуска ППО и балансировщиков веб-запросов;
  - 6 (без HA) - 18 (в HA режиме) серверов для запуска СПО (БД, кэш, диспетчер сообщений).

Заключение

На этапе проектирования решения следует выбрать способ развертывания того или иного компонента:

- тип инсталляции: пробная, предпродуктивный контур, продуктивный контур;
- Self-Hosted или Managed;
- HA или Standalone.

Выбранная архитектура решения может быть изменена и в процессе эксплуатации, но потребует проведения большого количества ручных операций.

Планирование ресурсов

Monq состоит из набора микросервисов. Для каждого микросервиса заданы лимиты на использование определенного количества ресурсов. Часть микросервисов использует настройки лимитов по-умолчанию, так называемые: `LimitRange` в рамках namespace в Kubernetes.

Компонентам СПО также требуется набор ресурсов.

В данной главе собраны базовые требования (в процессе выхода обновлений они могут меняться).

Для наиболее точного планирования ресурсов необходимо понимание нагрузки и требований к отказоустойчивости выбранного решения.

Данное руководство не учитывает расход ресурсов операционной системы и дополнительных служб (агентов мониторинга, резервного копирования и т.д.). При расчете ресурсов используются такие понятия как количество ресурсов для микросервиса, рекомендуемое количество микросервисов, рекомендуемые лимиты на СПО.

Все требования рассчитаны на основе пользовательского опыта и, в зависимости от характера и интенсивности нагрузки, могут отличаться.

Минимальные требования для Demo-VM

Рассматривается образ с предустановленной инфраструктурой.

Все микросервисы запущены в единственном экземпляре, системное программное обеспечение сконфигурировано на минимальное использование ресурсов.

Требуемые ресурсы:

VM	CPU	RAM	Disk
Demo-VM	8	24	60

Минимальные требования

В данном случае рассматривается самостоятельная установка на трех ВМ:

1. сервер для хостинга k8s control-plane;
2. сервер СУБД;
3. сервер для запуска микросервисов ППО.

Все микросервисы при этом запущены в единичном экземпляре, HA не применяется.

VM	CPU	RAM	Disk
Kubernetes control-plane	2	4	30
Kubernetes worker	8	24	80
DB server	8	14	200

Предпродуктивный контур

Считается стандартной инсталляцией, без применения HA. Может использоваться для тестирования функционала и скорости обработки данных.

Микросервисы запускаются в единичном экземпляре с возможностью масштабирования.

Данных ресурсов достаточно для обработки 300 событий в секунду.

VM	CPU	RAM	Disk
Kubernetes control-plane	2	4	30
Kubernetes worker	8	16	80
Kubernetes worker	8	16	80
Kubernetes worker	8	16	80
DB server	8	20	200

Продуктивный контур

Все компоненты СПО запускаются в режиме HA, имеется возможность масштабирования количества микросервисов, участвующих в обработке данных.

VM	CPU	RAM	Disk
Kubernetes control-plane	2	4	30
Kubernetes control-plane	2	4	30
Kubernetes control-plane	2	4	30
Kubernetes worker	12	24	120
Kubernetes worker	12	24	120
Kubernetes worker	12	24	120
ArangoDB	8	32	200
ArangoDB	8	32	200
ArangoDB	8	32	200
Clickhouse	8	24	400
Clickhouse	8	24	400
Clickhouse	8	24	400
Postgresql	6	16	200
Postgresql	6	16	200
Postgresql	6	16	200
RabbitMQ	4	6	40
RabbitMQ	4	6	40
RabbitMQ	4	6	40
Redis	4	6	40
Redis	4	6	40
Redis	4	6	40
VictoriaMetrics	8	16	300
VictoriaMetrics	8	16	300
VictoriaMetrics	8	16	300

Подготовка СПО

В соответствии с проектированием решения и планированием ресурсов потребуется подготовить итоговый стенд для развертывания.

По тексту будут ссылки на работу с файлом authfile, в целях исключения многократного повторения ссылка размещена в начале документа: см. Приложение А1. Инструкция по наполнению authfile.

Перед началом работ следует выбрать доменные имена для компонентов СПО.

В рамках демонстрационной VM используется зона `in.monq.local`. Каждому компоненту СПО присваивается отдельное доменное имя, например `postgresql.in.monq.local`.

Далее в документе для унификации будет использоваться зона `*.in.monq.local`, но может использоваться другая зона или отдельные имена для компонентов СПО.

Список переменных, используемых по тексту:

- `${infra_domain}` - dns зона для инфраструктурных компонентов
- `${global_domain}` - основное доменное имя развертываемого приложения

При описании способа запуска компонентов СПО в единичном экземпляре рассматривалась схема предпродуктивного контура:

- сервер для хостинга k8s control-plane;
- рабочая нода kubernetes - worker;
- рабочая нода kubernetes для хостинга СУБД.

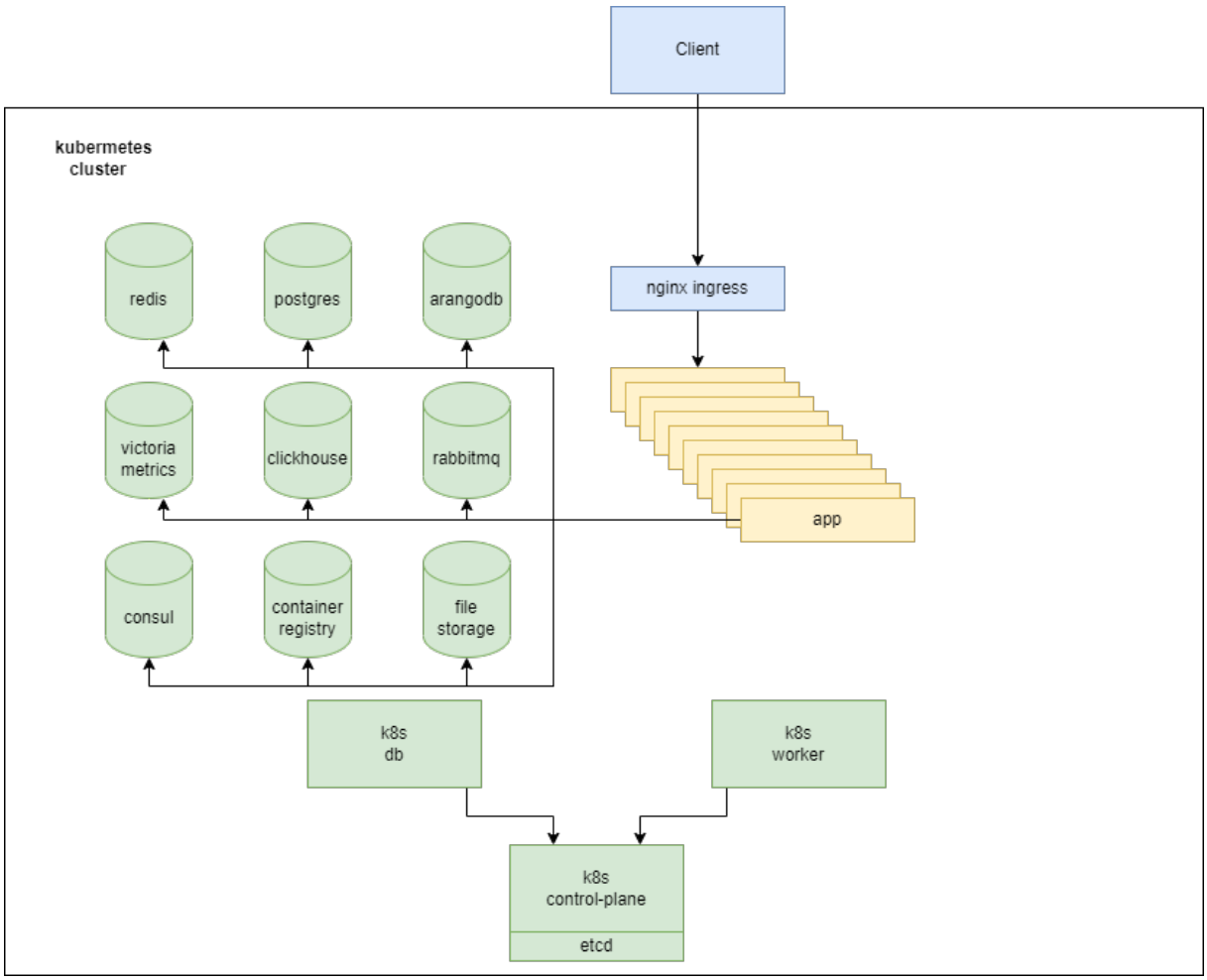


Схема минимального контура

При описании способа запуска компонентов СПО в режиме высокой доступности рассматривалась схема продуктивного контура с kubernetes в режиме singleMaster:

- сервер для хостинга k8s control-plane;
- несколько рабочих нод kubernetes - worker-1,2,3;
- для СУБД, кеша и диспетчеров сообщений выделены по три сервера на каждый сервис соответственно.

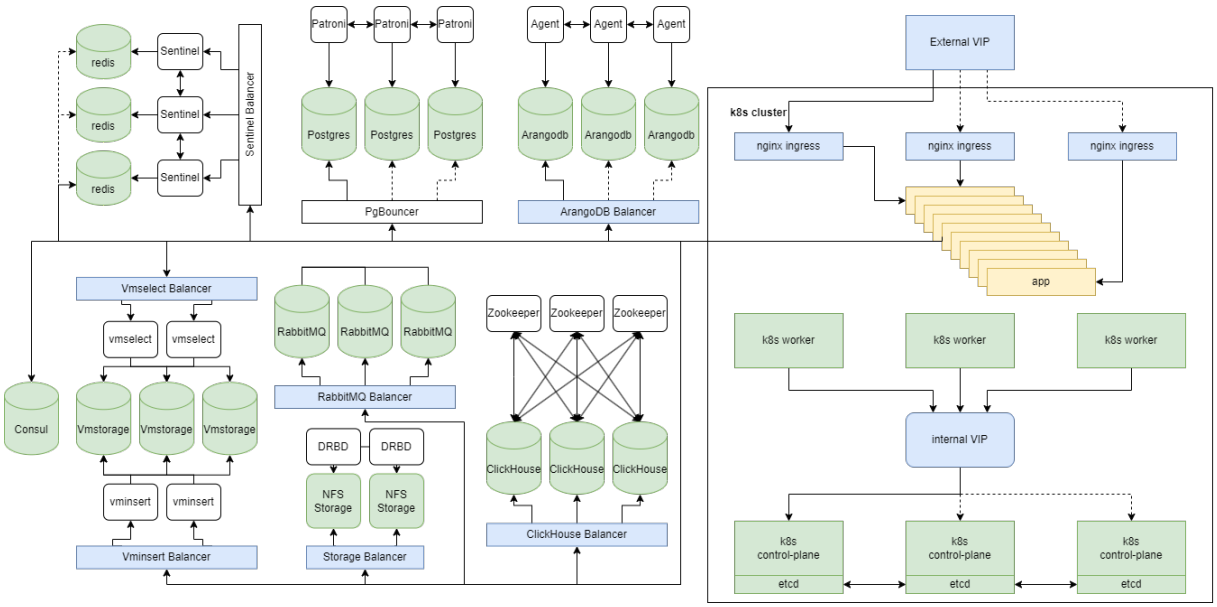


Схема продуктивного контура

В данном примере предполагается, что на все сервера будет установлен kuberenetes, под управлением которого будет запущено ППО и СПО.

В соответствии с примером так же будут обозначаться имена серверов:

- master, worker, db для минимального контура;
- master, worker-1,2,3, postgres-1,2,3, clickhouse-1,2,3 и тп для продуктивного контура.

Так как это примеры, в будущем пользователь может именовать сервера по своему.

Kubernetes

Managed kubernetes

Проверить возможность:

- создания ролей;
- создания сервисных аккаунтов;
- назначение роли сервисному аккаунту;
- запуск манифестов в рамках namespace;
- наличие или возможность подключения в кластер постоянных хранилищ в режиме read-WriteMany.

При использовании kubernetes в ”качестве сервиса”, пропускается блок ”SelfHosted kubernetes”, следует сразу перейти к пункту ”Последующая настройка кластера”.

SelfHosted kubernetes

Подготовка серверов

В данном пункте будет рассмотрена самостоятельная установка kubernetes. Подготовка операционных систем сводится к подготовке к запуску kubernetes.

Серверов может быть несколько, но не менее одного выделенного сервера под нужды Monq.

Рассматривается установка на Debian 11. Установка на другие дистрибутивы может быть произведена по аналогии.

На данном этапе выполняется установка операционных систем и первичная настройка.

см. Приложение А2. Подготовка серверов.

Запуск kubernetes

1. SingleMaster. Несколько серверов, сервер управления кластером запущен в единственном экземпляре, среда выполнения Monq запущена на нескольких серверах. Рекомендован для установок с некритичным уровнем отказоустойчивости, например предпродуктивный контур для отладки;
2. MultiMaster. Продуктивная среда, рекомендуется для установок с высокими требованиями по отказоустойчивости;
3. Подключение рабочих нод.

см. Приложение А3. Запуск Kubernetes для Monq.

Последующая настройка кластера

Необходимо настроить:

1. Балансировку http запросов внутри кластера. В качестве пограничного сервиса для перенаправления запросов к микросервисам служит ingress-controller на базе nginx.

   Nginx-ingress-controller отвечает за:

   - маршрутизацию запросов от клиентов к микросервисам, в том числе для самих микросервисов при внутреннем взаимодействии;

- балансировку запросов между микросервисами, когда задействованы механизмы горизонтального масштабирования;
- назначение дополнительных http заголовков, требуемых для прохождения данных в Monq.

  см. Запуск nginx ingress controller.

2. Пространства для запуска компонентов. Обычно это production и infra, для запуска ППО и СПО соответственно.

   см. Создание пространств.

3. Файловое хранилище. Требуется для хранения данных, которые по различным причинам не могут быть размещены в БД.

   - прикрепляемые файлы к КЕ;
   - результаты сборок автотестов;
   - фотографии в профиле пользователя;
   - плагины системных агентов.

   В Monq используется подключение единого хранилища к нескольким микросервисам.

   По этому для Monq в kubernetes драйвер хранилища должен иметь возможность подключение тома в режиме readWriteMany (список совместимых решений можно посмотреть в официальной документации).

   Самым простым и наиболее распространенным решением является NFS, по этому можно начать с него.

   см. Настройка хранилища файлов на базе NFS.

4. Зафиксировать токен и адрес подключения к api k8s, сформировать конфиг подключения в authfile, см инструкцию.

   Если токен администратора отсутствует, то его можно выписать самостоятельно по инструкции: см. Выпуск токена администратора.

Container registry.

Хранилище для образов микросервисов

Имеется возможность использовать собственное совместимое хранилище, для этого нужно указать данные для аутентификации в authfile. В процессе установки в это хранилище будут загружены образы микросервисов Monq для последующего запуска в kubernetes.

При развертывании отдельного хранилища образов необходимо:

- Выполнить установку согласно инструкции: см. Приложение А4. Запуск репозитория контейнеров на базе Docker Registry.;
- Сформировать конфигурацию подключения в authfile, см инструкцию.

Consul.

Хранилище конфигураций Monq.

Рассматривается SelfHosted

1. Запуск в единичном экземпляре statefulSet. Без высокой доступности.

   см. Приложение А5. Consul.

2. Формирование строки подключения к consul в authfile, см инструкцию.

Кластерный DNS.

> Внимание! Не рекомендуется использовать кластерный DNS kubernetes для разрешения доменных имен СПО.

Предпочтительный способ обращения микросервисов Monq к компонентам СПО с использованием доменных имен.

Рекомендуется использование внутреннего корпоративного DNS сервера для разрешения доменных имен компонентов СПО.

Рекомендуется не изменять настройки кластерного coredns, для обеспечения возможного последующего бесшовного обновления coredns.

См. Приложение А6. DNS в работе Monq.

Способы конфигурации DNS:

1. Имеется корпоративный DNS сервер, на котором имеется возможность создать DNS зоны с вышеуказанным набором доменных имен;
2. Доступ к DNS серверу ограничен или его нет, то в данном случае решение можно будет построить на кластерном coredns в составе kubernetes;

> Внимание! Это прямое вмешательство в конфигурацию coredns, при последующих обновлениях СПО следует помнить об этом.

см. Специализированная настройка внутрикластерного DNS.

3. В случае если развертывание компонентов СПО производится в kubernetes с помощью операторов, то в качестве точек подключения указываются соответствующие сервисы kubernetes.

СУБД, кеш и диспетчер сообщений

Общие положения:

- В данном документе рассматривается решение по запуску всех компонентов внутри kubernetes;
- Если пользователю требуется запустить компонент СПО по своему методу, рекомендуем воспользоваться официальной инструкцией поставщика решения. В таком случае запущенный экземпляр СПО нужно будет настраивать как managed сервис;
- В предложенных примерах в качестве хранилища данных для компонентов СУБД используется локальный каталог на ноде, этот способ указан для примера, что бы программный комплекс можно было запустить используя подход ”copy-past”. В случае если СПО будет запущено в kubernetes в продуктивном режиме, рекомендуется подключать хранилища в рекомендуемой конфигурации: см документацию kubernetes;
- В данном документе не рассматриваются лучшие практики по запуску кластерных решений.

  Выбор оптимального решения всегда обусловлен требованиями к отказоустойчивости и производительности конкретной инсталляции.

  Данное руководство демонстрирует базовый подход к запуску кластера высокой доступности и настройку Monq для работы с ним.

  В целях минимизации ручной настройки используются операторы k8s от производителя СПО.

Описательная часть будет разложена по следующему принципу:

1. Использование СПО, предоставляемого в качестве сервиса (managed).
2. SelfHosted решение:

    1. Запуск в единичном экземпляре statefulSet. Без высокой доступности;
    2. Запуск с помощью kubernetes operator. Обеспечение высокой доступности.

3. Заключительные мероприятия.

   При использовании СПО в ”качестве сервиса”, следует сразу перейти к пункту ”Последующая настройка кластера”.

PostgreSQL

Managed PostgreSQL.

1. Проверить наличие возможности создания пользователей и назначение ролей этим пользователям: CREATEDB, CREATEROLE через SQL запросы к БД. В данном случае указан минимальный набор прав, требуемый для установки. В комплекте со сценарием установки есть возможность очистки СПО от объектов Monq (бд, пользователи итп). Для его работы требуется роль SUPERUSER, без неё очистка БД от пользователей и ролей самостоятельно не возможна, потребуется привлекать адиминистраторов.
2. Проверить возможность подключения к порту СУБД с рабочих нод.
3. Список известных ограничений:

   - Yandex Managed Service for PostgreSQL не поддерживает создание ролей с помощью SQL.

Если предоставленное в качестве сервиса решение удовлетворяет вышеуказанным требованиям, можно перейти к пункту ”Последующая настройка”, в противном случае следует воспользоваться ”SelfHosted” решением.

SelfHosted PostgreSQL

1. Запуск в единичном экземпляре statefulSet. Без высокой доступности. См. Standalone PostgreSQL;
2. Запуск с помощью kubernetes operator. Обеспечение высокой доступности. См. HA cluster PostgreSQL.

Заключительные мероприятия

Формирование строки подключения к PostgreSQL в authfile, см инструкцию.

Clickhouse

Managed ClickHouse

Проверить возможность создания ролей, профилей, квот и пользователей со следующими параметрами:

1. назначение пользователю прав `access_management`;
2. Назначение прав для роли: `SELECT`,`INSERT`,`ALTER`,`CREATE TABLE`,`CREATE VIEW` ,`CREATE DICTIONARY`, `CREATE TEMPORARY TABLE`, `DROP TABLE`,`DROP VIEW`, `DROP`,`TRUNCATE`,`OPTIMIZE`,`SHOW`,`KILL QUERY`;
3. Назначение ограничений на роль: `max_memory_usage_for_user`, `max_memory_usage_for_all` , `max_execution_time`, `timeout_before_checking_execution_speed`;
4. Создание квоты и назначения её на роль: `FOR INTERVAL 60 MINUTE MAX queries 0, errors 0, result_rows 0, read_rows 0, execution_time 0`;
5. Создание пользователей и назначение роли с вышеуказанными параметрами;
6. Проверить возможность подключения к порту СУБД с рабочих нод.

Если предоставленное в качестве сервиса решение удовлетворяет вышеуказанным требованиям, можно перейти к пункту ”Последующая настройка”, в противном случае следует воспользоваться ”SelfHosted” решением.

SelfHosted ClickHouse

1. Запуск в единичном экземпляре statefulSet. Без высокой доступности. См. Standalone Clickhouse;
2. Запуск с помощью kubernetes operator. Обеспечение высокой доступности. См. HA cluster Clickhouse.

Заключительные мероприятия

Формирование строки подключения к ClickHouse в authfile, см инструкцию.

ArangoDB

Managed ArangoDB

Проверить возможность:

- создания пользователей с уровнем прав ”Administrative” для бд ”_system”;
- подключения к порту СУБД с рабочих нод.

Если предоставленное в качестве сервиса решение удовлетворяет вышеуказанным требованиям, можно перейти к пункту ”Последующая настройка”, в противном случае следует воспользоваться ”SelfHosted” решением.

SelfHosted ArangoDB

1. Запуск в единичном экземпляре statefulSet. Без высокой доступности. См. Standalone ArangoDB;
2. Запуск с помощью kubernetes operator. Обеспечение высокой доступности. См. HA cluster ArangoDB.

Заключительные мероприятия

Формирование строки подключения к ClickHouse в authfile, см инструкцию.

VictoriaMetrics

Managed

Проверить возможность:

- записи и чтения в бд;
- подключения к порту СУБД с рабочих нод.

VictoriaMetrics, запущенная в кластере может иметь две конфигурации с отдельными точками на запись чтение (select и insert) или за единым балансировщиком (vmauth).

Во всех случаях следует убедиться в доступности интерфейсов и возможности записи чтения.

Если предоставленное в качестве сервиса решение удовлетворяет вышеуказанным требованиям, можно перейти к пункту ”Последующая настройка”, в противном случае следует воспользоваться ”SelfHosted” решением.

SelfHosted

1. Запуск в единичном экземпляре statefulSet. Без высокой доступности. См. Standalone VictoriaMetrics;
2. Запуск с помощью kubernetes operator. Обеспечение высокой доступности. См. HA cluster VictoriaMetrics.

Заключительные мероприятия

Формирование строки подключения к VictoriaMetrics в authfile, см инструкцию.

Redis

Managed

1. StandAlone. Проверить доступность экземпляра СПО по заданному порту. Проверить возможность авторизации и выполнения основных команд;
2. HA-Cluster. Проверить доступность Sentinel, убедиться что Sentinel отдает список экземпляров с ролями master, slave. Проверить возможность подключения к экземплярам redis;
3. Подключения к порту Redis с рабочих нод.

Если предоставленное в качестве сервиса решение удовлетворяет вышеуказанным требованиям, перейти к пункту ”Последующая настройка”, в противном случае следует воспользоваться ”SelfHosted” решением.

SelfHosted

1. Запуск в единичном экземпляре statefulSet. Без высокой доступности. См. Standalone Redis;
2. Запуск с помощью kubernetes operator. Обеспечение высокой доступности. См. HA cluster Redis.

Заключительные мероприятия

Формирование строки подключения к Redis в authfile, см инструкцию.

RabbitMQ

Managed

1. StandAlone:
    1. Проверить доступность экземпляра СПО по заданному порту;
    2. Проверить возможность авторизации и выполнения основных операций: ”создание пользователя с ролью Administrator, создание очередей”.
2. HA-Cluster:
    1. Проверить доступность экземпляра СПО по заданному порту;
    2. Проверить возможность авторизации и выполнения основных команд: ”создание пользователя с ролью Administrator, создание очередей” на одном сервере, подключиться ко второму серверу, проверить распространение данных.
3. Подключения к порту RabbitMQ с рабочих нод.

Если предоставленное в качестве сервиса решение удовлетворяет вышеуказанным требованиям, можно перейти к пункту ”Последующая настройка”, в противном случае следует воспользоваться ”SelfHosted” решением.

SelfHosted

1. Запуск в единичном экземпляре statefulSet. Без высокой доступности. См. Standalone RabbitMQ;
2. Запуск с помощью kubernetes operator. Обеспечение высокой доступности. См. HA cluster RabbitMQ.

Заключительные мероприятия

Формирование строки подключения к RabbitMQ в authfile, см инструкцию.

Заключение

Результатом выполненных мероприятий можно считать:

- Рабочую инфраструктуру для запуска Monq;

- Заполненный authfile - system_auth.json, в котором заданы настройки для подключения ко всем компонентам СПО.

  Если было пропущено заполнение одного из блоков с настройками, необходимо его заполнить перед переходом к дальнейшей установке.

Приложение А1. Инструкция по наполнению authfile.

Authfile используется сценарием установки Monq для подключения к инфраструктурным объектам и создания объектов Monq:

- Манифесты kubernetes;
- Пользователи и базы данных;
- Прочие объекты требуемые для работы Monq.

В authfile помимо параметров подключения, указывается тип запуска системного программного обеспечения, это может быть cluster(ha) или standalone, и соответствующие ключи для конфигурации параметров.

> Внимание! Важно корректно заполнить параметры подключения к СПО, т.к. для некоторых компонентов взаимодействие с HA СПО отличается от режима standAlone.

В приложенных документах хранится модель данных authfile и описание переменных, необходимо внимательно изучить эти документы перед наполнением.

- Пример файла system_auth.json;
- Описание полей файла авторизации в компонентах СПО.

Описание полей файла авторизации в компонентах СПО.

Ключ	Тип	Описание
arangodb.host	string	hostname или ip адрес сервера arangodb
arangodb.port	int	порт сервера сервера arangodb
arangodb.proto	string	протокол сервера сервера arangodb (`http`,`https`)
arangodb.users.root_user.name	string	имя пользователя arangodb

Ключ	Тип	Описание
arangodb.users.root_user.password	string	пароль пользователя arangodb
clickhouse.host	string	hostname или ip адрес сервера clickhouse
clickhouse.port	int	порт сервера clickhouse
clickhouse.proto	string	протокол сервера clickhouse (`http`,`https`)
clickhouse.cluster	bool	Переменная должна быть **`true`** если используется cluster режим
clickhouse.cluster_name	string	Имя кластера в clickhouse. Если не используется кластер - задать любое значение, например `monq`
clickhouse.users.root_user.name	string	имя пользователя clickhouse
clickhouse.users.root_user.password	string	пароль пользователя clickhouse
consul.host	string	hostname или ip адрес consul
consul.port	int	порт consul
consul.proto	string	протокол consul (`http`,`https`)
consul.users.root_user.token	string	токен пользователя consul
k8s.host	string	hostname или ip адрес apiserver kubernetes
k8s.port	int	порт apiserver kubernetes
k8s.proto	string	протокол apiserver kubernetes (`http`,`https`)
k8s.users.root_user.token	string	токен авторизации в apiserver kubernetes
postgresql.host	string	hostname или ip адрес сервера postgresql
postgresql.port	int	порт сервера postgresql
postgresql.ssl	bool	используется ли ssl для подключению к серверу postgresql
postgresql.ssl_mode	string	Режим ssl для подключению к серверу postgresql. Переменная должна быть задана если используется ssl. Возможные значения: `Allow`, `Prefer`, `Require`. Если ssl не используется, указать любое из значений
postgresql.ssl_trust	bool	Доверять ssl сертификату сервера postgresql. Если ssl

Ключ	Тип	Описание
		не используется, указать **`false`**
postgresql.users.root_user.name	string	имя пользователя postgresql
postgresql.users.root_user.password	string	пароль пользователя postgresql
rabbitmq.host	string	hostname или ip адрес сервера rabbitmq
rabbitmq.port	int	порт сервера rabbitmq
rabbitmq.amqp_port	int	ampq порт сервера rabbitmq
rabbitmq.proto	string	протокол api сервера rabbitmq (`http`,`https`)
rabbitmq.quorum_queues	bool	использовать ли quorum очереди, необходимо для кластера
rabbitmq.virtual_host	string	vhost в rabbitmq, задать / если не используется кастомный
rabbitmq.users.root_user.name	string	имя пользователя rabbitmq
rabbitmq.users.root_user.password	string	пароль пользователя rabbitmq
redis.host	string	hostname или ip адрес сервера redis
redis.port	int	порт сервера redis
redis.sentinel	bool	используется ли sentinel
redis.service_name	string	имя сервиса в sentinel. Должна быть задана если используется sentinel. Если не используется, то задать `mymaster`
redis.users.root_user.password	string	пароль пользователя redis
registry.host	string	hostname или ip адрес имя пользователя
registry.port	int	порт docker registry
registry.proto	string	протокол docker registry (`http`,`https`)
registry.location	string	путь в registry до образов Monq (например если переменная задана и имеет значение `monq` при деплое image будет запрошен по `myregistry.ru/monq/myservice:mytag`)

Ключ	Тип	Описание
registry.user	string	опционально. имя пользователя docker registry
registry.password	string	опционально. пароль пользователя docker registry
victoriametrics.host	string	hostname или ip адрес сервера victoria metrics. Должен быть задан если cluster = false
victoriametrics.port	int	порт сервера victoria metrics. Должен быть задан если cluster = false
victoriametrics.proto	string	протокол сервера victoria metrics (`http`,`https`). Должен быть задан если cluster = false
victoriametrics.cluster	bool	используется ли кластер victoria metrics
victoriametrics.cluster_account	string	аккаунт в кластере victoria metrics. Должен быть задан если используется кластерная версия
victoriametrics.auth_type	string	тип авторизации в victoria metrics, возможные значения `BasicAuth`,`None`
victoriametrics.cluster_insert.host	string	hostname или ip адрес сервера vminsert. Должен быть задан если используется кластерная версия
victoriametrics.cluster_insert.port	int	опционально. порт сервера vminsert. Должен быть задан если используется кластерная версия
victoriametrics.cluster_insert.proto	string	опционально. протокол сервера vminsert (`http`,`https`). Должен быть задан если используется кластерная версия
victoriametrics.cluster_select.host	string	опционально. hostname или ip адрес сервера select. Должен быть задан если используется кластерная версия
victoriametrics.cluster_select.port	string	порт сервера vmselect. Должен быть задан если используется кластерная версия
victoriametrics.cluster_select.proto	string	протокол сервера vmselect (`http`,`https`). Должен быть задан если используется кластерная версия

Ключ	Тип	Описание
victoriametrics.users.root_user.name	int	имя пользователя victoria metrics, должен быть задан если auth_type = `BasicAuth`
victoriametrics.users.root_user.password	string	пароль пользователя victoria metrics, должен быть задан если auth_type = `BasicAuth`

Пример файла system_auth.json

```
{
    "arangodb": { "host": "arangodb.in.monq.local", "port": 8529, "proto": "http",
        "users": { "root_user": { "name": "root", "password": "**********" } }
    },
    "clickhouse": { "host": "clickhouse.in.monq.local", "port": 8123, "proto": "http",
        "cluster": false, "cluster_name": "monq",
        "users": { "root_user": { "name": "root_user", "password": "*********" } }
    },
    "consul": { "host": "consul.in.monq.local", "port": 8500, "proto": "http",
        "users": { "root_user": { "token": "**********" } }
    },
    "k8s": { "host": "k8s-api.in.monq.local", "port": 6443, "proto": "https",
        "users": { "root_user": { "token": "***********" } }
    },
    "postgresql": { "host": "postgresql.in.monq.local", "port": 5432, "ssl": false,
        "ssl_mode": "Require", "ssl_trust": true,
        "users": { "root_user": { "name": "postgres", "password": "********" } }
    },
    "rabbitmq": { "amqp_port": 5672, "host": "rabbitmq.in.monq.local", "port": 15672,
        "proto": "http", "quorum_queues": false, "virtual_host": "/",
        "users": { "root_user": { "name": "root", "password": "*********" } }
    },
    "redis": { "host": "redis.in.monq.local", "port": 6379, "sentinel": false,
        "service_name": "mymaster",
        "users": { "root_user": { "password": "********" } }
    },
    "registry": { "host": "registry.in.monq.local", "port": 5000, "proto": "http",
        "user": "registry", "password": "********"
    },
    "victoriametrics": { "host": "victoriametrics.in.monq.local", "port": 8428,
        "proto": "http", "cluster": false, "cluster_account": "0", "auth_type": "BasicAuth",
        "cluster_insert": { "host": "vminsert.in.monq.local", "port": 8480,
          "proto": "http" },
        "cluster_select": { "host": "vmselect.in.monq.local", "port": 8481,
          "proto": "http" },
        "users": { "root_user": { "name": "monq", "password": "******" } }
    }
}
```

Приложение А2. Подготовка серверов.

- Debian 11.

Debian 11

Данный пример описывает подготовку VM Debian 11 для запуска kubernetes.

Настройка сервера для роли control plane/master

Задать нужное имя сервера, в примере используется master

```
hostnamectl set-hostname master
```

Установить пакеты необходимые для запуска и эксплуатации Monq:

```
apt update
apt install -y gpg curl wget dnsutils vim telnet unzip bash-completion ca-certificates jq \
  libicu67
```

Выполнить настройку синхронизации времени, и убедиться что время синхронизировано:

```
timedatectl status
```

В случае если необходимо изменить адреса NTP серверов надо отредактировать файл и выполнить перезапуск сервиса

```
nano /etc/systemd/timesyncd.conf

systemctl enable systemd-timesyncd
systemctl restart systemd-timesyncd
```

Отключить файл подкачки - swap:

```
swapoff -a
sed -i '/ swap / s/^/#/' /etc/fstab
```

Загрузить необходимые модули ядра:

```
cat <<EOF | tee /etc/modules-load.d/containerd.conf
overlay
br_netfilter
EOF
modprobe br_netfilter
modprobe overlay
```

Добавить опции ядра:

```
cat <<EOF | tee /etc/sysctl.d/99-kubernetes-cri.conf
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
fs.inotify.max_user_instances = 524288
EOF

cat <<EOF | tee /etc/sysctl.d/70-disable-ipv6.conf
net.ipv6.conf.all.disable_ipv6 = 1
EOF

sysctl --system
```

Установить containerd:

```
mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg \
  | gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/debian $(lsb_release -cs) stable" \
  | tee /etc/apt/sources.list.d/docker.list > /dev/null

apt update
apt install containerd.io=1.6.6-1

containerd config default > /etc/containerd/config.toml
sed -i 's/SystemdCgroup = false/SystemdCgroup = true/g' /etc/containerd/config.toml
systemctl enable containerd
systemctl restart containerd
```

Установить kubernetes:

```
curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | \
  gpg --dearmor -o /usr/share/keyrings/kubernetes-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] \
  https://apt.kubernetes.io/ kubernetes-xenial main" \
  | tee /etc/apt/sources.list.d/kubernetes.list

apt update
apt install -y kubelet=1.26.6-00 kubeadm=1.26.6-00 kubectl=1.26.6-00
apt-mark hold kubelet kubeadm kubectl
systemctl enable kubelet
```

Настроить автодополнение команд:

```
mkdir -p /etc/bash_completion.d/
kubectl completion bash | tee /etc/bash_completion.d/kubectl > /dev/null
crictl completion bash | tee /etc/bash_completion.d/crictl > /dev/null
```

```
cat > /etc/crictl.yaml << EOF
runtime-endpoint: unix:///run/containerd/containerd.sock
image-endpoint: unix:///run/containerd/containerd.sock
timeout: 2
debug: false
pull-image-on-create: true
EOF
```

Активировать автодополнение команд для текущего пользователя:

```
cat >> ~/.bashrc << EOF
if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
    . /etc/bash_completion
fi
EOF
```

Настройка сервера worker

Задать нужное имя сервера, в примере используется worker.

```
hostnamectl set-hostname worker
```

Установить необходимые пакеты:

```
apt update
apt install -y gpg curl
```

Выполнить настройку синхронизации времени, и убедиться что время синхронизировано:

```
timedatectl status
```

В случае если необходимо изменить адреса NTP серверов надо отредактировать файл и выполнить перезапуск сервиса

```
nano /etc/systemd/timesyncd.conf

systemctl enable systemd-timesyncd
systemctl restart systemd-timesyncd
```

Отключить swap:

```
swapoff -a
sed -i '/ swap / s/^/#/' /etc/fstab
```

Загрузить необходимые модули ядра:

```
cat <<EOF | tee /etc/modules-load.d/containerd.conf
overlay
br_netfilter
EOF
modprobe br_netfilter
modprobe overlay
```

Добавить опции ядра:

```
cat <<EOF | tee /etc/sysctl.d/99-kubernetes-cri.conf
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
fs.inotify.max_user_instances = 524288
EOF

cat <<EOF | tee /etc/sysctl.d/70-disable-ipv6.conf
net.ipv6.conf.all.disable_ipv6 = 1
EOF

sysctl --system
```

Установить containerd:

```
mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg \
  | gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/debian $(lsb_release -cs) stable" \
  | tee /etc/apt/sources.list.d/docker.list > /dev/null

apt update
apt install containerd.io=1.6.6-1

containerd config default > /etc/containerd/config.toml
sed -i 's/SystemdCgroup = false/SystemdCgroup = true/g' /etc/containerd/config.toml
systemctl enable containerd
systemctl restart containerd
```

Установить kubernetes:

```
curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | \
  gpg --dearmor -o /usr/share/keyrings/kubernetes-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] \
  https://apt.kubernetes.io/ kubernetes-xenial main" \
  | tee /etc/apt/sources.list.d/kubernetes.list

apt update
apt install -y kubelet=1.26.6-00 kubeadm=1.26.6-00
apt-mark hold kubelet kubeadm
systemctl enable kubelet
```

Приложение А3. Запуск Kubernetes для Monq.

- Режим SingleMaster;
- Режим MultiMaster;
- Подключение рабочих нод.

Режим SingleMaster.

Перед началом работ необходимо создать на внешнем DNS сервере запись для хоста apiserver вида `k8s-api.${infra_domain}`, запись должна разрешаться в IP master сервера.

Провести инициализацию кластера на master сервере:

```
infra_domain="in.monq.local"
k8s_domain="k8s-api.${infra_domain}"

systemctl enable kubelet.service
kubeadm init --control-plane-endpoint "${k8s_domain}:6443" \
  --skip-phases="addon/kube-proxy" \
  --upload-certs \
  --pod-network-cidr="10.244.0.0/16" \
  --service-cidr="10.16.0.0/16"
```

В случае если сети pod и service пересекаются с текущими сетями есть возможность задать другие сети, рекомендуется использовать /16.

По завершении команды будет выдано сообщение с командой которую необходимо выполнить на остальных серверах для join, ее следует скопировать для дальнейшего использования.

Выполнить конфигурацию kubectl:

```
mkdir -p $HOME/.kube
cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
chown $(id -u):$(id -g) $HOME/.kube/config
```

Выполнить настройки опций логирования:

```
kubectl get cm -n kube-system kubelet-config -o json | sed \
's|cluster.local\\n|cluster.local\\ncontainerLogMaxFiles: 2\\n|g' \
| kubectl replace -f -

kubectl get cm -n kube-system kubelet-config -o json | sed \
's|cluster.local\\n|cluster.local\\ncontainerLogMaxSize: 10Mi\\n|g' \
```

```
| kubectl replace -f -

echo "containerLogMaxFiles: 2" >> /var/lib/kubelet/config.yaml
echo "containerLogMaxSize: 10Mi" >> /var/lib/kubelet/config.yaml
systemctl restart kubelet
```

Выполнить запуск CNI: см Запуск CNI.

Убедиться что нода перешла в статус `Ready`:

```
kubectl get node
```

Режим MultiMaster.

Необходимо обязательно создать на внешнем DNS сервере запись для хоста apiserver вида `k8s-api.${infra_domain}`, на первоначальном этапе запись должна разрешаться в IP первого master сервера.

Провести инициализацию кластера на первом master сервере:

```
k8s_domain="k8s-api.${infra_domain}"

systemctl enable kubelet.service
kubeadm init --control-plane-endpoint "${k8s_domain}:6443" \
  --skip-phases="addon/kube-proxy" \
  --upload-certs \
  --pod-network-cidr="10.244.0.0/16" \
  --service-cidr="10.16.0.0/16"
```

В случае если сети pod и service пересекаются с текущими сетями есть возможность задать другие сети, рекомендуется использовать /16.

По завершении команды будет выдано сообщение с командой которую необходимо выполнить на остальных серверах для join, ее следует скопировать для дальнейшего использования.

Выполнить конфигурацию kubectl:

```
mkdir -p $HOME/.kube
cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
chown $(id -u):$(id -g) $HOME/.kube/config
```

Выполнить настройки опций логирования:

```
kubectl get cm -n kube-system kubelet-config -o json | sed \
's|cluster.local\\n|cluster.local\\ncontainerLogMaxFiles: 2\\n|g' \
| kubectl replace -f -

kubectl get cm -n kube-system kubelet-config -o json | sed \
's|cluster.local\\n|cluster.local\\ncontainerLogMaxSize: 10Mi\\n|g' \
| kubectl replace -f -

echo "containerLogMaxFiles: 2" >> /var/lib/kubelet/config.yaml
echo "containerLogMaxSize: 10Mi" >> /var/lib/kubelet/config.yaml
```

```
systemctl restart kubelet
```

Выполнить подключение в кластер оставшихся master нод (команды выполнить на всех master нодах):

```
k8s_domain="k8s-api.${infra_domain}"
systemctl enable kubelet
kubeadm join ${k8s_domain}:6443 --token ************* \
--discovery-token-ca-cert-hash sha256:******** \
--control-plane --certificate-key *******
```

После включения нод в кластер скопировать конфигурацию kubectl в локальный каталог:

```
mkdir ~/.kube/
cp /etc/kubernetes/admin.conf ~/.kube/config
```

Выполнить запуск CNI: см Запуск CNI

Выполнить запуск kube-vip для организации virtual ip между master нодами. Для этого на всех master серверах выполнить команды:

> Внимание! Необходимо задать имя интерфейса и IP адрес используемый для VIP!

```
export VIP=10.10.0.11
export interface=eth0
export kube_vip_version=v0.5.0

ctr image pull ghcr.io/kube-vip/kube-vip:$kube_vip_version
ctr run --rm --net-host ghcr.io/kube-vip/kube-vip:$kube_vip_version vip /kube-vip manifest
    pod \
  --interface $interface \
  --address $VIP \
  --controlplane \
  --arp \
  --prometheusHTTPServer 127.0.0.1:2112 \
  --leaderElection | tee /etc/kubernetes/manifests/kube-vip.yaml
```

После запуска заменить адрес для DNS записи `k8s-api.${infra_domain}`, на IP адрес используемый для VIP.

Убедиться что ноды перешли в статус `Ready`:

```
kubectl get node
```

Подключение рабочих нод

Если токен был утерян, срок действия истек, можно получить новый, для этого на master сервере надо выполнить команду:

```
kubeadm token create --print-join-command
```

Вывод команды сохранить и применить на рабочей ноде для подключения в существующий кластер, примерное содержание команды.

> Внимание! Если не было настроено разрешение доменных имен на вышестоящем DNS сервере, то добавить запись с указанием доменного имени и ip адреса сервера хостинга k8s control-plane в /etc/hosts.

```
infra_domain="in.monq.local"
k8s_domain="k8s-api.${infra_domain}"
kubeadm join ${k8s_domain}:6443 --token ************* --discovery-token-ca-cert-hash sha256
    :*************
```

Убедиться что ноды перешли в статус `Ready`, для этого на сервере k8s control-plane выполнить команду:

```
kubectl get node
```

В течении двух минут статус должен измениться.

Если нода так и не перешла в статус готовности, проверить логи kubelet.

Запуск CNI

- Cilium (Рекомендуемое решение)
- Flannel

Cilium.

> Внимание! Для установки требуется Helm, см. # Приложение A13. Установка helm.

Добавить helm репозиторий и установить cilium

```
infra_domain="in.monq.local"
api_server_host="k8s-api.${infra_domain}"
api_server_port="6443"

helm repo add cilium https://helm.cilium.io/

helm upgrade --install cilium cilium/cilium --version 1.13.3 \
    --namespace kube-system \
    --set kubeProxyReplacement=strict \
    --set k8sServiceHost=${api_server_host} \
    --set k8sServicePort=${api_server_port} \
    --set operator.replicas=1 \
    --set ipam.mode=kubernetes
```

Flannel

> Внимание! Изменить переменную `podCidr`, если была изменена при инициализации kubernetes.

> Внимание! Kubernetes установлен с аргументом --skip-phases="addon/kube-proxy", рекомендуется выполнить реинициализацию кластера без данного аргумента или установить kube-proxy самостоятельно.

Добавить helm репозиторий и установить flannel

```
helm repo add flannel https://flannel-io.github.io/flannel/

helm upgrade --install flannel flannel/flannel --version v0.22.0 \
--namespace kube-system \
--set podCidr="10.244.0.0/16"
```

Запуск ingress-nginx controller.

Добавить метку на ноду, определяющую её как разрешенную для запуска ingress-nginx controller. В данном примере балансировщик будет запущен на ноде worker, но если планируется несколько нод в качестве балансировщиков, например для организации HA, то можно поставить метки на все ноды.

```
server_name=worker
kubectl label no ${server_name} ingress=
```

Добавить helm репозиторий и устанавливить чарт:

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

helm upgrade --install ingress-nginx ingress-nginx/ingress-nginx --version 4.7.0 \
--namespace ingress-nginx --create-namespace \
--set controller.kind=DaemonSet \
--set controller.hostPort.enabled=true \
--set controller.nodeSelector.ingress="" \
--set controller.service.enabled=true \
--set controller.config.body-size=50m \
--set controller.config.hsts=false \
--set controller.config.large-client-header-buffers="4 32k" \
--set controller.config.proxy-body-size=50m \
--set controller.config.proxy-buffer-size=128k \
--set controller.config.proxy-buffers="4 256k" \
--set controller.config.proxy-busy-buffers-size=256k \
--set controller.config.proxy-connect-timeout="15" \
--set controller.config.proxy-read-timeout="300" \
--set controller.config.proxy-send-timeout="300" \
--set controller.config.server-name-hash-bucket-size="256" \
--set controller.config.worker-shutdown-timeout=10s
```

Проверить:

1. Состояние запуска контейнера:

```
kubectl get po -n ingress-nginx -o wide -w
```

2. Возможность подключения:

```
curl "< ${server_name} ip >"
# должны получить ответом 404
```

Создание пространств.

1. Создание namespace production и infra

```
kubectl create namespace infra
kubectl create namespace production
```

2. Ноды, предназначенные для запуска микросервисов Monq должны быть отмечены меткой ”worker”.

   Например, в кластере только одна нода ”worker”:

```
kubectl label node worker function=worker
```

   Если таких нод несколько, то на каждую ноду необходимо добавить соответствующую метку.

3. Создание лимитов по умолчанию для namespace production

   > Внимание! Не рекомендуется уменьшать лимиты по умолчанию.

```
cat <<EOF | kubectl create -f -
apiVersion: v1
kind: LimitRange
metadata:
  name: limit-range
  namespace: production
spec:
  limits:
  - default:
      cpu: 500m
      memory: 1024Mi
    defaultRequest:
      cpu: 25m
      memory: 128Mi
    type: Container
EOF
```

Настройка хранилища файлов на базе NFS.

> Внимание! Если NFS предоставляется как сервис, то зафиксировать адрес подключения и перейти пункту создания PV и PVC.

Общий порядок установки и настройки:

1. Выбрать сервер, установить на него NFS server.

   В данном примере будет использован сервер db, на него и будет установлен NFS server.

   Установить сервис и создать монтируемый каталог:

   > Внимание! Выполняется на сервере выделенным под NFS сервер.

   ```
   apt install nfs-kernel-server -y
   host_path="/storage/nfs"
   mkdir -p $host_path
   echo "$host_path *(rw,sync,no_root_squash,no_all_squash,no_subtree_check)" >> \
    /etc/exports

   systemctl restart nfs-server
   systemctl enable nfs-server
   ```

2. На серверах где будут запущены kubernetes ноды надо установить клиент nfs:

   > Внимание! Выполняется на рабочей ноде kubernetes - worker.

   ```
   apt install nfs-common
   ```

3. Создать PV и PVC:

   > Внимание! Выполняется на сервере для хостинга k8s control-plane.

   ```
   host_path="/storage/nfs"
   product_namespace="production"
   nfs_server_address="< ip address nfs server >"
   nfs_storage_size="20Gi"

   cat <<EOF | kubectl create -f -
   apiVersion: v1
   items:
   - apiVersion: v1
     kind: PersistentVolume
     metadata:
       name: pv-monq
       annotations:
         volume.beta.kubernetes.io/storage-class: nfs
     spec:
       capacity:
         storage: ${nfs_storage_size}
       accessModes:
         - ReadWriteMany
       persistentVolumeReclaimPolicy: Retain
       nfs:
         server: "${nfs_server_address}"
   ```

```
      path: "${host_path}"
- kind: PersistentVolumeClaim
  apiVersion: v1
  metadata:
    name: pvc-monq
    namespace: ${product_namespace}
    annotations:
      volume.beta.kubernetes.io/storage-class: nfs
  spec:
    accessModes:
      - ReadWriteMany
    resources:
      requests:
        storage: ${nfs_storage_size}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
EOF
```

Выпуск токена для установки monq.

1. Создать манифесты сервисной учетной записи

```
monq_namespace="production"

cat <<EOF | kubectl apply -f -
apiVersion: v1
items:
- apiVersion: v1
  kind: ServiceAccount
  metadata:
    name: "installer"
    namespace: "${monq_namespace}"
  secrets:
    - name: "installer-token"
- apiVersion: v1
  kind: Secret
  metadata:
    name: "installer-token"
    namespace: "${monq_namespace}"
    annotations:
      kubernetes.io/service-account.name: "installer"
  type: kubernetes.io/service-account-token
- apiVersion: rbac.authorization.k8s.io/v1
  kind: Role
  metadata:
    name: "installer"
    namespace: "${monq_namespace}"
  rules:
  - apiGroups: [""]
    resources:
      - services
      - pods
      - configmaps
      - secrets
      - serviceaccounts
    verbs: ['*']
  - apiGroups: ['apps']
    resources:
```

```
      - deployments
    verbs: ['*']
  - apiGroups: [""]
    resources:
      - namespaces
      - persistentvolumeclaims
    verbs: ['get']
  - apiGroups: ["networking.k8s.io"]
    resources:
      - ingresses
    verbs: ['*']
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources:
      - roles
      - rolebindings
    verbs: ['*']
- apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    name: "installer"
    namespace: "${monq_namespace}"
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: Role
    name: "installer"
  subjects:
  - kind: ServiceAccount
    name: "installer"
    namespace: "${monq_namespace}"
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
EOF
```

2. Получить токен

```
installer_token=$(kubectl get secret -n ${monq_namespace} installer-token \
 -o jsonpath='{.data.token}' | base64 --decode)
echo save it: ${installer_token}
```

3. Для наполнения authfile выставить значения:

- k8s.host: "k8s-api.in.monq.local";
- k8s.port: 6443;
- k8s.proto: https;
- k8s.users.root_user.token: "<check ${installer_token}>";

Приложение А4. Запуск репозитория контейнеров на базе Docker Registry.

Порядок запуска:

1. Выбрать сервер, на котором будет запущен сервис.

   > Внимание! В данном примере используется hostpath для хранения файлов контейнера. По этому размещение контейнера должно быть явно назначено на указанный сервер.

   > Внимание! В данном случае используется каталог на сервере, данное решение не рекомендовано к запуску в промышленной среде.

   Выполнить назначение метки на выбранную ноду:

   ```
   server_name="db"
   kubectl label no ${server_name} registry=
   ```

2. Создать каталог для хранения данных.

   1. На сервере, где будет размещен контейнер создать каталог для хранения данных

      > Внимание! Эта операция выполняется не на сервере controlplane, а на сервере, где будет запущен контейнер.

      ```
      mkdir -p /storage/registry
      ```

   2. На сервере controlplane создать манифест хранилища данных.

      ```
      infra_domain="in.monq.local"
      host_path="/storage"
      storage_size="30Gi"

      cat <<EOF | kubectl create -f -
      apiVersion: v1
      kind: PersistentVolume
      metadata:
        name: registry-pv
        labels:
      ```

```
    app: registry
    instance: registry.${infra_domain}
spec:
  capacity:
    storage: ${storage_size}
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: ${host_path}/registry
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: registry
          operator: Exists
EOF
```

3. Применить манифест.

```
infra_domain="in.monq.local"
registry_port="5000"
infra_namespace="infra"
storage_size="30Gi"

cat <<EOF | kubectl create -f -
apiVersion: v1
items:
- kind: StatefulSet
  apiVersion: apps/v1
  metadata:
    labels:
      app: registry
      instance: registry.${infra_domain}
    name: registry
    namespace: ${infra_namespace}
  spec:
    replicas: 1
    serviceName: statefulset-registry
    selector:
      matchLabels:
        app: registry
        instance: registry.${infra_domain}
    template:
      metadata:
        labels:
          app: registry
          instance: registry.${infra_domain}
      spec:
        containers:
        - name: registry
          image: registry:2.8.2
          imagePullPolicy: IfNotPresent
          ports:
            - name: http
              containerPort: ${registry_port}
              hostPort: ${registry_port}
              protocol: TCP
          env:
            - name: REGISTRY_HTTP_ADDR
              value: "0.0.0.0:${registry_port}"
```

```
            - name: REGISTRY_STORAGE_DELETE_ENABLED
              value: "true"
          volumeMounts:
            - mountPath: /var/lib/registry
              name: host-volume
          livenessProbe:
            httpGet:
              scheme: HTTP
              path: /
              port: ${registry_port}
            initialDelaySeconds: 30
            timeoutSeconds: 30
        nodeSelector:
          registry: ""
    volumeClaimTemplates:
    - metadata:
        name: host-volume
        annotations:
          volume.beta.kubernetes.io/storage-class: local-storage
      spec:
        selector:
          matchLabels:
            app: registry
            instance: registry.${infra_domain}
        accessModes: ["ReadWriteOnce"]
        resources:
          requests:
            storage: ${storage_size}
- apiVersion: v1
  kind: Service
  metadata:
    name: registry
    namespace: infra
    labels:
      app: registry
      instance: registry.${infra_domain}
  spec:
    ports:
    - name: http
      port: ${registry_port}
      protocol: TCP
      targetPort: ${registry_port}
    selector:
      app: registry
      instance: registry.${infra_domain}
    sessionAffinity: None
    type: ClusterIP
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
EOF
```

4. Добавить insecure registry в конфигурацию containerd.

> Внимание! Выполняется на каждой ноде, которая будет взаимодействовать с данным registry.

В файле конфигурации `/etc/containerd/config.toml` после строки `[plugins."io.containerd.grpc.v1.cri".registry.mirrors]`, добавить запись (**учесть**

пробелы):

```
        [plugins."io.containerd.grpc.v1.cri".registry.mirrors."registry.in.monq.local
    :5000"]
          endpoint = ["http://registry.in.monq.local:5000"]
```

Или с помощью команды:

```
infra_domain="in.monq.local"
registry_port="5000"
registry_address="registry.${infra_domain}:${registry_port}"

sed -i '/plugins."io.containerd.grpc.v1.cri".registry.mirrors.$/a\ \
     [plugins."io.containerd.grpc.v1.cri".registry.mirrors."'${registry_address}'"] \
       endpoint = ["http://'${registry_address}'"]' /etc/containerd/config.toml
```

После перезагрузить containerd, что бы он применил новую конфигурацию:

```
systemctl restart containerd
```

5. Добавить DNS запись `registry.${infra_domain} < ip address db server >` :
    1. Запись на вышестоящих DNS серверах (рекомендуемый метод);
    2. В файле /etc/hosts (НЕ рекомендуемый метод):

    ```
    echo "<ip address db server> registry.${infra_domain}" >> /etc/hosts
    ```

6. Проверить:
    1. Состояние запуска контейнера:

    ```
    kubectl get po -n infra registry-0 -o wide -w
    ```

    2. Логи контейнера:

    ```
    kubectl logs -n infra registry-0 -f
    ```

    3. Возможность подключения с авторизационными данными.

    ```
    curl registry.${infra_domain}:${registry_port}/v2/_catalog
    ```

Приложение А5. Consul.

Порядок запуска:

1. Выбрать сервер, на котором будет запущен сервис.

   > Внимание! В данном примере используется hostpath для хранения файлов контейнера. По этому размещение контейнера должно быть явно назначено на указанный сервер.

   > Внимание! В данном случае используется каталог на сервере, данное решение не рекомендовано к запуску в промышленной среде.

   Выполнить назначение метки на выбранную ноду:

   ```
   server_name="db"
   kubectl label no ${server_name} consul=
   ```

2. Создать каталог для хранения данных.

   1. На сервере, где будет размещен контейнер создать каталог для хранения данных:

      > Внимание! Эта операция выполняется не на сервере controlplane, а на сервере, где будет запущен контейнер.

      ```
      mkdir -p /storage/consul
      ```

   2. На сервере controlplane создать манифест хранилища данных:

      ```
      infra_domain="in.monq.local"
      host_path="/storage"
      storage_size="1Gi"

      cat <<EOF | kubectl create -f -
      apiVersion: v1
      kind: PersistentVolume
      metadata:
        name: consul-pv
        labels:
          app: consul
          instance: consul.${infra_domain}
      spec:
      ```

```
  capacity:
    storage: ${storage_size}
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: ${host_path}/consul
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: consul
          operator: Exists
EOF
```

3. Применить манифест в kubernetes. В данном манифесте задана базовая конфигурация сервиса, можно её изменить исходя из своих потребностей.

```
infra_domain="in.monq.local"
infra_namespace="infra"

cat <<EOF | kubectl create -f -
kind: ConfigMap
apiVersion: v1
metadata:
  name: consul-configuration
  namespace: ${infra_namespace}
  labels:
    app: consul
    instance: consul.${infra_domain}
data:
  consul.json: |
    {
      "bind_addr": "0.0.0.0",
      "client_addr": "0.0.0.0",
      "datacenter": "monq",
      "data_dir":"/var/lib/consul",
      "enable_script_checks":true,
      "enable_syslog":false,
      "log_level":"info",
      "server":true,
      "ui":true,
      "acl": {
        "enabled": true,
        "default_policy": "deny",
        "down_policy": "extend-cache"
      },
      "bootstrap_expect": 1,
      "telemetry" : {
        "disable_hostname":true,
        "prometheus_retention_time": "2h"
      }
    }
EOF
```

4. Применить манифест.

```
infra_domain="in.monq.local"
infra_namespace="infra"
storage_size="1Gi"
```

```
cat <<EOF | kubectl create -f -
apiVersion: v1
items:
- kind: StatefulSet
  apiVersion: apps/v1
  metadata:
    labels:
      app: consul
      instance: consul.${infra_domain}
    name: consul
    namespace: ${infra_namespace}
  spec:
    replicas: 1
    serviceName: statefulset-consul
    selector:
      matchLabels:
        app: consul
        instance: consul.${infra_domain}
    template:
      metadata:
        labels:
          app: consul
          instance: consul.${infra_domain}
      spec:
        containers:
          - name: consul
            image: consul:1.8.0
            imagePullPolicy: IfNotPresent
            args:
              - agent
              - -config-file=/etc/consul/consul.json
              - -dns-port=0
              - -serf-wan-port=0
            ports:
              - name: http
                containerPort: 8500
                hostPort: 8500
                protocol: TCP
            volumeMounts:
              - name: host-volume
                mountPath: /consul/data/
                subPath: consul/data
              - mountPath: /etc/consul/consul.json
                name: configuration
                subPath: consul.json
            livenessProbe:
              httpGet:
                path: /
                port: 8500
                scheme: HTTP
              initialDelaySeconds: 30
              timeoutSeconds: 30
        volumes:
          - name: configuration
            configMap:
              name: consul-configuration
        nodeSelector:
          consul: ""
    volumeClaimTemplates:
    - metadata:
        name: host-volume
        annotations:
          volume.beta.kubernetes.io/storage-class: local-storage
```

```
      spec:
        selector:
          matchLabels:
            app: consul
            instance: consul.${infra_domain}
        accessModes: ["ReadWriteOnce"]
        resources:
          requests:
            storage: ${storage_size}
- apiVersion: v1
  kind: Service
  metadata:
    labels:
      app: consul
      instance: consul.${infra_domain}
    name: consul
    namespace: ${infra_namespace}
  spec:
    ports:
    - name: http
      port: 8500
      protocol: TCP
      targetPort: 8500
    type: ClusterIP
    selector:
      app: consul
      instance: consul.${infra_domain}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
EOF
```

5. Инициировать consul.

   Данная операция выполняется через API consul. В предложенном примере сервис использует порт сервера 8500.

```
bootstrap_token=$(curl -s -X PUT consul.in.monq.local:8500/v1/acl/bootstrap | jq -r '.ID
    ')
echo "save it: "${bootstrap_token}
```

   Ответ содержит токен доступа с максимальными правами, следует его сохранить.

   Создать и применить токен агента:

```
agent_token=$(curl -s -X PUT -H "X-Consul-Token: ${bootstrap_token}" \
consul.in.monq.local:8500/v1/acl/create \
-d '{"Name": "Agent Token",
     "Type": "client",
     "Rules":"node \"\" { policy = \"write\" } service \"\" { policy = \"read\" }"}' \
| jq -r '.ID')
```

   Обновить манифест конфигурации consul:

```
infra_domain="in.monq.local"
infra_namespace="infra"
# в данном манифесте используется переменная ${agent_token}, определенная в предыдущем
    шаге

cat <<EOF | kubectl apply -f -
```

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: consul-configuration
  namespace: ${infra_namespace}
  labels:
    app: consul
    instance: consul.${infra_domain}
data:
  consul.json: |
    {
      "bind_addr": "0.0.0.0",
      "client_addr": "0.0.0.0",
      "datacenter": "monq",
      "data_dir":"/var/lib/consul",
      "enable_script_checks":true,
      "enable_syslog":false,
      "log_level":"info",
      "server":true,
      "ui":true,
      "acl": {
        "enabled": true,
        "default_policy": "deny",
        "tokens": {
         "agent": "${agent_token}"
        },
        "down_policy": "extend-cache"
      },
      "bootstrap_expect": 1,
      "telemetry" : {
        "disable_hostname":true,
        "prometheus_retention_time": "2h"
      }
    }
EOF
```

Перезагрузить контейнер

```
kubectl delete po -n ${infra_namespace} consul-0
```

6. Проверить:

    1. Состояние запуска контейнера

        ```
        kubectl get po -n infra consul-0 -o wide -w
        ```

    2. Логи контейнера

        ```
        kubectl logs -n infra consul-0 -f
        ```

    3. Возможность подключения с авторизационными данными

7. Для наполнения authfile выставить значения:

    - consul.host: `"consul.in.monq.local"`;
    - consul.port: `8500`;
    - consul.proto: `http`;
    - consul.users.root_user.token: `"<check ${bootstrap_token}>"`;

> Внимание! Значение вышеуказанных переменных заполнены с учетом текущего примера.

Приложение А6. DNS в работе Monq.

Список используемых доменных имен

№	Сервис, назначение	Доменное имя по умолчанию
1	Репозиторий контейнеров	registry.in.monq.local
2	СУБД postgresql*	postgresql.in.monq.local
3	СУБД clickhouse*	clickhouse.in.monq.local
4	СУБД arangodb*	arangodb.in.monq.local
5.a	СУБД victoriametrics, одиночный инстанс или кластер с vmauth	victoriametrics.in.monq.local
5.b	СУБД victoriametrics, кластер	vminsert.in.monq.local vmselect.in.monq.local
6	Кеш сервер, Redis**	redis.in.monq.local
7	Брокер сообщений, Rabbitmq*	rabbitmq.in.monq.local
8	Адрес API сервера k8s*	k8s-api.in.monq.local
9	Адрес хранилища конфигураций	consul.in.monq.local
10	Доменные имена установки monq, разрешаются в адрес ingress контроллера, в случае использования нескольких контроллеров, указывается адрес точки балансировки	<доменное имя> api.<доменное имя> registry.api.<доменное имя>

* в случае использования HA, указать адрес балансировщика запросов; ** в случае использования HA, указать адрес sentinel.

Указанные доменные имена участвуют в установке и эксплуатации системы и должны корректно разрешаться внутри kubernetes и с машины на которой будет производиться запуск сценария установки monq.

Специализированная настройка внутрикластерного DNS.

1. Подключить в конфигурационном файле чтение файлов по маске из каталога.

   Для этого открыть на редактирование основной конфигурационный файл coredns и добавить настройку импорта файлов зон в конец конфигурации (после loadbalance):

```
kubectl edit cm -n kube-system coredns

...
    loadbalance
    auto {
      directory /etc/coredns/ (.*)\.db {1}
    }
...
```

2. Применить ручную конфигурацию разрешения имен. Перед применением рекомендуется ознакомиться с официальной документации coredns.

> Внимание! В примере используется CNAME на k8s сервис, это применимо в том случае, если СПО развертывалось внутри kubernetes, иначе следует написать свою конфигурацию кластерного разрешения имен.

```
infra_domain="in.monq.local"
infra_namespace="infra"
k8s_api_ip="<ip kubernetes api server>"
global_domain="<monq domain name>"

cat <<EOF | kubectl create -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns-custom
  namespace: kube-system
data:
  ${infra_domain}.db: |
    ${infra_domain}. IN SOA ns1.${infra_domain}. info.${infra_domain}. (
    2023052200 7200 3600 1209600 3600)
    ${infra_domain}.                IN      NS      ns1.${infra_domain}.
    ;;; apps ;;;
    arangodb.${infra_domain}.     IN  CNAME   arangodb.infra.svc.cluster.local.
    clickhouse.${infra_domain}.   IN  CNAME   clickhouse.infra.svc.cluster.local.
    postgresql.${infra_domain}.   IN  CNAME   postgresql.infra.svc.cluster.local.
    rabbitmq.${infra_domain}.     IN  CNAME   rabbitmq.infra.svc.cluster.local.
    redis.${infra_domain}.        IN  CNAME   redis.infra.svc.cluster.local.
    consul.${infra_domain}.       IN  CNAME   consul.infra.svc.cluster.local.
    victoriametrics.${infra_domain}. IN  CNAME victoriametrics.infra.svc.cluster.local.
    registry.${infra_domain}.     IN  CNAME   registry.infra.svc.cluster.local.
    k8s-api.${infra_domain}.      IN  A       ${k8s_api_ip}
  ${global_domain}.db: |
    ${global_domain}.  IN   SOA   ns1.${global_domain}. info.${global_domain}. (
    2023052200 7200 3600 1209600 3600)
    ${global_domain}.             IN      NS       ns1.${global_domain}.
    ;;; apps ;;;
    ${global_domain}. IN CNAME ingress-nginx-controller.ingress-nginx.svc.cluster.local.
    ;;; CNAME ;;;
    api.${global_domain}.         IN  CNAME   ${global_domain}.
```

```
    *.api.${global_domain}.       IN  CNAME   ${global_domain}.
EOF
```

3. Включить config-map в манифест и выполнить перезагрузку coredns

```
kubectl -n kube-system patch deploy coredns -p \
 '{"spec":{"template":{"spec":{"volumes":[{"configMap":
   {"defaultMode": 420,"name": "coredns-custom","optional": true},
   "name": "custom-config-volume"}]}}}}'

kubectl -n kube-system patch deploy coredns -p \
 '{"spec":{"template":{"spec":{"containers":[{"name":"coredns","volumeMounts":
   [{"name":"custom-config-volume","mountPath":"/etc/coredns/custom"}]}]}}}}'

kubectl delete po -n kube-system -l k8s-app=kube-dns
```

Приложение А7. PostgreSQL

Standalone PostgreSQL

Порядок запуска:

1. Выбрать сервер, на котором будет запущен сервис.

   > Внимание! В данном примере используется hostpath для хранения файлов контейнера. По этому размещение контейнера должно быть явно назначено на указанный сервер.

   > Внимание! В данном случае используется каталог на сервере, данное решение не рекомендовано к запуску в промышленной среде.

   Выполнить назначение метки на выбранную ноду:

   ```
   server_name="db"
   kubectl label no ${server_name} postgresql=
   ```

2. Создать каталог для хранения данных.

   1. На сервере, где будет размещен контейнер создать каталог для хранения данных:

      > Внимание! Эта операция выполняется не на сервере controlplane, а на сервере, где будет запущен контейнер.

      ```
      mkdir -p /storage/postgresql
      ```

   2. На сервере controlplane создать манифест хранилища данных, в данном случае используется каталог на сервере, данное решение не рекомендовано к запуску в промышленной среде

      ```
      infra_domain="in.monq.local"
      host_path="/storage/postgresql"
      storage_size="30Gi"

      cat <<EOF | kubectl create -f -
      apiVersion: v1
      ```

```
kind: PersistentVolume
metadata:
  name: postgresql-pv
  labels:
    app: postgresql
    instance: postgresql.${infra_domain}
spec:
  capacity:
    storage: ${storage_size}
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: ${host_path}
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: postgresql
          operator: Exists
EOF
```

3. Создать конфигурационный файл в kubernetes. В данном конфигурационном файле предложена примерная конфигурация СУБД, пользователь может изменить её исходя из своих потребностей.

```
infra_domain="in.monq.local"
infra_namespace="infra"

cat <<EOF | kubectl create -f -
kind: ConfigMap
apiVersion: v1
metadata:
  name: postgresql-configuration
  namespace: ${infra_namespace}
  labels:
    app: postgresql
    instance: postgresql.${infra_domain}
data:
  postgresql.conf: |
    data_directory = '/var/lib/postgresql/data'
    hba_file = '/var/lib/postgresql/data/pg_hba.conf'
    ident_file = '/var/lib/postgresql/data/pg_ident.conf'
    external_pid_file = '/var/lib/postgresql/data/12-main.pid'
    listen_addresses = '*'
    port = 5432
    max_connections = 200
    unix_socket_directories = '/var/run/postgresql'
    ssl = true
    ssl_cert_file = '/etc/ssl/certs/ssl-cert-snakeoil.pem'
    ssl_key_file = '/etc/ssl/private/ssl-cert-snakeoil.key'
    shared_buffers = 768MB
    work_mem = 1966kB
    maintenance_work_mem = 192MB
    dynamic_shared_memory_type = posix
    effective_io_concurrency = 200
    max_worker_processes = 2
    wal_buffers = 16MB
    max_wal_size = 4GB
    min_wal_size = 1GB
```

```
    checkpoint_completion_target = 0.9
    random_page_cost = 1.1
    effective_cache_size = 2304MB
    default_statistics_target = 100
    log_directory = '/var/log/postgresql'
    log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'
    log_rotation_size = 100MB
    log_line_prefix = '%t [%p-%l] %q%u@%d '
    log_timezone = 'W-SU'
    cluster_name = '9.5/main'
    stats_temp_directory = '/var/lib/postgresql/data/pg_stat_tmp'
    datestyle = 'iso, mdy'
    timezone = 'W-SU'
    lc_messages = 'en_US.UTF-8'
    lc_monetary = 'en_US.UTF-8'
    lc_numeric = 'en_US.UTF-8'
    lc_time = 'en_US.UTF-8'
    default_text_search_config = 'pg_catalog.english'
    max_parallel_workers_per_gather = 1
    max_parallel_workers = 2
    max_parallel_maintenance_workers = 1
EOF
```

4. Задать авторизационные данные учетной записи

```
infra_domain="in.monq.local"
infra_namespace="infra"

pg_root_password=$(openssl rand -base64 16)

cat <<EOF | kubectl create -f -
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: postgresql-secret
  namespace: ${infra_namespace}
  labels:
    app: postgresql
    instance: postgresql.${infra_domain}
data:
  POSTGRES_PASSWORD: $(echo $pg_root_password | base64)
  POSTGRES_HOST_AUTH_METHOD: $(echo md5 | base64)
EOF
```

Впоследствии пароль можно будет посмотреть с помощью команды:

```
infra_namespace="infra"

kubectl get secrets -n ${infra_namespace} postgresql-secret \
  -o jsonpath='{.data.POSTGRES_PASSWORD}' | base64 --decode
```

5. Применить манифест.

```
infra_domain="in.monq.local"
infra_namespace="infra"
storage_size="30Gi"

cat <<EOF | kubectl create -f -
apiVersion: v1
items:
  - kind: StatefulSet
```

```
apiVersion: apps/v1
metadata:
  name: postgresql
  namespace: ${infra_namespace}
  labels:
    app: postgresql
    instance: postgresql.${infra_domain}
spec:
  replicas: 1
  serviceName: statefulset-postgresql
  selector:
    matchLabels:
      app: postgresql
      instance: postgresql.${infra_domain}
  template:
    metadata:
      labels:
        app: postgresql
        instance: postgresql.${infra_domain}
    spec:
      containers:
        - name: postgresql
          image: postgres:12.15
          imagePullPolicy: IfNotPresent
          args:
            - "-c"
            - "config_file=/etc/postgresql/postgresql.conf"
          envFrom:
            - secretRef:
                name: postgresql-secret
          ports:
            - name: postgresql
              containerPort: 5432
              hostPort: 5432
              protocol: TCP
          volumeMounts:
            - mountPath: /var/lib/postgresql/data
              name: host-volume
              subPath: var/lib/postgresql/data
            - mountPath: /etc/postgresql/
              name: configuration
          livenessProbe:
            tcpSocket:
              port: 5432
            initialDelaySeconds: 30
            timeoutSeconds: 30
          resources:
            limits:
              cpu: 2
              memory: 4G
            requests:
              cpu: 250m
              memory: 512Mi
      volumes:
        - configMap:
            name: postgresql-configuration
          name: configuration
      nodeSelector:
        postgresql: ""
  volumeClaimTemplates:
    - metadata:
        name: host-volume
        annotations:
          volume.beta.kubernetes.io/storage-class: local-storage
```

```
          spec:
            selector:
              matchLabels:
                app: postgresql
                instance: postgresql.${infra_domain}
            accessModes: [ "ReadWriteOnce" ]
            resources:
              requests:
                storage: ${storage_size}
  - apiVersion: v1
    kind: Service
    metadata:
      name: postgresql
      namespace: ${infra_namespace}
      labels:
        app: postgresql
        instance: postgresql.${infra_domain}
    spec:
      ports:
        - name: postgresql
          port: 5432
          protocol: TCP
          targetPort: 5432
      type: ClusterIP
      selector:
        app: postgresql
        instance: postgresql.${infra_domain}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
EOF
```

6. Проверить:

    1. Состояние запуска контейнера

    ```
    kubectl get po -n infra postgresql-0 -o wide -w
    ```

    2. Логи контейнера

    ```
    kubectl logs -n infra postgresql-0 -f
    ```

    3. Возможность подключения с авторизационными данными

7. Для наполнения authfile выставить значения:

    - postgresql.host: `"postgresql.in.monq.local"`;
    - postgresql.port: 5432;
    - postgresql.users.root_user.name: `"postgres"`;
    - postgresql.users.root_user.password: `"< check ${pg_root_password}" >`.

> Внимание! Значение вышеуказанных переменных заполнены с учетом текущего примера.

HA cluster PostgreSQL

Запуск осуществляется с помощью оператора postgres-operator от zalando.

> Внимание! Для установки требуется Helm, см. # Приложение A13. Установка helm.

1. Установить оператор:

```
helm repo add postgres-operator-charts \
 https://opensource.zalando.com/postgres-operator/charts/postgres-operator
helm install postgres-operator postgres-operator-charts/postgres-operator \
  --namespace postgres-operator --create-namespace
```

2. На нодах, где будет запушен postgresql создать каталоги для хранения данных

> Внимание! Эта операция выполняется не на сервере controlplane, а на сервере, где будет запущен postgresql.

```
mkdir -p /storage/postgresql
```

3. Создать storage-class и pv, так как csi driver hostPath не поддерживает provisioning:

> Внимание! Обратить внимание на привязку pv к ноде, указаны имена серверов из примера.

> Внимание! В данном примере создание PV выполняется в цикле, для трех серверов.

```
host_path="/storage/postgresql"
storage_size="30Gi"

cat <<EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
EOF

for server_number in {1..3}; do
cat <<EOF | kubectl create -f -
apiVersion: v1
kind: PersistentVolume
metadata:
  name: postgres-pv-w${server_number}
  labels:
    service: postgres
spec:
  capacity:
    storage: ${storage_size}
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
```

```
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: ${host_path}
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - postgres-${server_number}
EOF
done
```

4. Запуск кластера

```
storage_size="30Gi"

cat <<EOF | kubectl create -f -
apiVersion: "acid.zalan.do/v1"
kind: postgresql
metadata:
  name: monq-pg-cluster
  namespace: postgres-operator
spec:
  teamId: "monq"
  volume:
    size: ${storage_size}
    storageClass: local-storage
    selector:
      matchLabels:
        service: postgres
  additionalVolumes:
    - name: empty
      mountPath: /opt/empty
      targetContainers:
        - all
      volumeSource:
        emptyDir: {}
  enableShmVolume: true
  numberOfInstances: 3
  postgresql:
    version: "12"
EOF
```

Увеличение кластера выполняется через редактирование CRD postgres, поле `numberOfInstances` ;

При уменьшении количества реплик всегда удаляется последняя(старший номер в statefulSet);

Кластер остается работоспособным при работоспособности одной реплики;

Подключение осуществляется только с опцией ssl=required;

Два сервиса позволяют обращаться либо к мастеру, либо к репликам, но в режиме только чтения;

5. Получить пароль пользователя postgres:

```
infra_namespace="infra"

kubectl get secret -n ${infra_namespace} \
  postgres.monq-pg-cluster.credentials.postgresql.acid.zalan.do \
  -o 'jsonpath={.data.password}' | base64 -d
```

6. Проверить:

    1. Состояние запуска контейнеров:

        ```
        kubectl get po -n postgres-operator -w
        ```

    2. Логи контейнеров на предмет ошибок;

    3. Возможность подключения с авторизационными данными:

        - Получить пароль:

            ```
            kubectl -n postgres-operator get secret \
              postgres.monq-pg-cluster.credentials.postgresql.acid.zalan.do \
              -o 'jsonpath={.data.password}' | base64 -d
            ```

        - Настроить ***временное*** соединение с БД.

            Создать configmap с конфигурацией проксирования:

            ```
            cat <<EOF | kubectl create -f -
            apiVersion: v1
            kind: ConfigMap
            metadata:
              name: tcp-services
              namespace: ingress-nginx
            data:
              6432: "postgres-operator/monq-pg-cluster:5432"
            EOF
            ```

            Добавить проброс порта в сервис ingress:

            ```
            kubectl edit -n ingress-nginx svc/ingress-nginx-controller
            ```

            ```
            ...
                - name: proxied-tcp-6432
                  port: 6432
                  targetPort: 6432
                  protocol: TCP
            ...
            ```

            В ingress nginx добавить аргумент для включения проксирования:

            ```
            ...
                - --tcp-services-configmap=ingress-nginx/tcp-services
            ...
            ```

            Выполнить подключение удаленным клиентом.

    4. Распространение данных в кластере, чтение и запись.

Создать тестовую таблицу:

```
CREATE TABLE public."__EFMigrationsHistory" (
  "MigrationId" varchar(150) NOT NULL,
  "ProductVersion" varchar(32) NOT NULL,
  CONSTRAINT "PK___EFMigrationsHistory" PRIMARY KEY ("MigrationId")
);
```

Сменить активного мастера путем перезагрузки текущего:

```
pg_master=$(kubectl -n postgres-operator get pods \
  -o jsonpath={.items..metadata.name} \
  -l application=spilo,cluster-name=monq-pg-cluster,spilo-role=master)
kubectl delete po -n postgres-operator $pg_master
```

Убедиться, что роль мастера перешла другому POD:

```
kubectl -n postgres-operator get pods -o jsonpath={.items..metadata.name} \
  -l application=spilo,cluster-name=monq-pg-cluster,spilo-role=master
```

Переподключиться и проверить наличие данных;

Удалить тестовую таблицу и временное соединение, см ”Настроить ***временное*** соединение с БД”.

7. Для наполнения authfile выставить значения:

   - postgresql.host: `"monq-pg-cluster.postgres-operator.svc.cluster.local"`;
   - postgresql.port: 5432;
   - postgresql.ssl: **`true`**;
   - postgresql.ssl_mode: `"Require"`;
   - postgresql.ssl_trust: **`true`**;
   - postgresql.users.root_user.name: `"postgres"`;
   - postgresql.users.root_user.password: `"< check ${postgresPass}"` >.

> Внимание! Значение вышеуказанных переменных заполнены с учетом текущего примера.

Приложение А8. Clickhouse.

Standalone Clickhouse

Порядок запуска:

1. Выбрать сервер, на котором будет запущен сервис.

   > Внимание! В данном примере используется hostpath для хранения файлов контейнера. По этому размещение контейнера должно быть явно назначено на указанный сервер.

   > Внимание! В данном случае используется каталог на сервере, данное решение не рекомендовано к запуску в промышленной среде.

   Выполнить назначение метки на выбранную ноду:

   ```
   server_name="db"
   kubectl label no ${server_name} clickhouse=
   ```

2. Создать каталог для хранения данных.

   1. На сервере, где будет размещен контейнер создать каталог для хранения данных:

      > Внимание! Эта операция выполняется не на сервере controlplane, а на сервере, где будет запущен контейнер.

      ```
      mkdir -p /storage/clickhouse
      ```

   2. На сервере controlplane создать манифест хранилища данных, в данном случае используется каталог на сервере hostPath, данное решение не рекомендовано к запуску в промышленной среде

      ```
      infra_domain="in.monq.local"
      host_path="/storage/clickhouse"
      storage_size="30Gi"

      cat <<EOF | kubectl create -f -
      apiVersion: v1
      ```

```
kind: PersistentVolume
metadata:
  name: clickhouse-pv
  labels:
    app: clickhouse
    instance: clickhouse.${infra_domain}
spec:
  capacity:
    storage: ${storage_size}
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: ${host_path}
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: clickhouse
          operator: Exists
EOF
```

3. Задать авторизационные данные пользователя root_user, запомнить пароль, он понадобиться в дальнейшем при наполнении authfile.

```
ch_root_password=$(openssl rand -base64 16)
echo save it: ${ch_root_password}
```

4. Создать конфигурационный файл в kubernetes. В данном конфигурационном файле предложена примерная конфигурация СУБД, пользователь может изменить её исходя из своих потребностей.

```
infra_domain="in.monq.local"
infra_namespace="infra"
ch_password_hash=$(echo -n "${ch_root_password}" | sha256sum | sed s/\ .*$//g)

cat <<EOF | kubectl create -f -
kind: ConfigMap
apiVersion: v1
metadata:
  name: clickhouse-configuration
  namespace: ${infra_namespace}
  labels:
    app: clickhouse
    instance: clickhouse.${infra_domain}
data:
  users.xml: |
    <yandex>
        <profiles>
            <default>
                <use_uncompressed_cache>0</use_uncompressed_cache>
                <load_balancing>random</load_balancing>
            </default>
            <readonly>
                <readonly>1</readonly>
            </readonly>
        </profiles>
        <users>
```

```
            <root_user>
                <password_sha256_hex>${ch_password_hash}</password_sha256_hex>
                <networks>
                    <ip>::/0</ip>
                </networks>
                <profile>default</profile>
                <access_management>1</access_management>
                <quota>default</quota>
            </root_user>
        </users>
        <quotas>
            <default>
                <interval>
                    <duration>3600</duration>
                    <queries>0</queries>
                    <errors>0</errors>
                    <result_rows>0</result_rows>
                    <read_rows>0</read_rows>
                    <execution_time>0</execution_time>
                </interval>
            </default>
        </quotas>
    </yandex>
  config.xml: |
    <?xml version="1.0"?>
    <yandex>
        <logger>
            <level>warning</level>
            <console>1</console>
        </logger>
        <http_port>8123</http_port>
        <tcp_port>9000</tcp_port>
        <interserver_http_port>9009</interserver_http_port>
        <listen_host>0.0.0.0</listen_host>
        <listen_try>1</listen_try>
        <max_connections>4096</max_connections>
        <keep_alive_timeout>3</keep_alive_timeout>
        <max_concurrent_queries>100</max_concurrent_queries>
        <uncompressed_cache_size>8589934592</uncompressed_cache_size>
        <mark_cache_size>5368709120</mark_cache_size>
        <path>/clickhouse/</path>
        <tmp_path>/clickhouse/tmp/</tmp_path>
        <user_files_path>/var/lib/clickhouse/user_files/</user_files_path>
        <users_config>users.xml</users_config>
        <default_profile>default</default_profile>
        <default_database>default</default_database>
        <timezone>Europe/Moscow</timezone>
        <!-- <umask>022</umask> -->
        <mlock_executable>false</mlock_executable>
        <zookeeper incl="zookeeper-servers" optional="true" />
        <macros incl="macros" optional="true" />
        <builtin_dictionaries_reload_interval>360</builtin_dictionaries_reload_interval>
        <max_session_timeout>3600</max_session_timeout>
        <default_session_timeout>60</default_session_timeout>
        <query_log>
            <database>system</database>
            <table>query_log</table>
            <partition_by>toYYYYMM(event_date)</partition_by>
            <ttl>event_date + INTERVAL 7 DAY DELETE</ttl>
            <flush_interval_milliseconds>7500</flush_interval_milliseconds>
        </query_log>
        <dictionaries_config>*_dictionary.xml</dictionaries_config>
        <compression></compression>
        <distributed_ddl>
```

```
            <path>/clickhouse/task_queue/ddl</path>
        </distributed_ddl>
        <merge_tree>
            <max_suspicious_broken_parts>5</max_suspicious_broken_parts>
            <enable_mixed_granularity_parts>1</enable_mixed_granularity_parts>
        </merge_tree>
        <format_schema_path>/var/lib/clickhouse/format_schemas/</format_schema_path>
        <access_control_path>/clickhouse/access/</access_control_path>
    </yandex>
EOF
```

5. Применить манифест.

```
infra_domain="in.monq.local"
infra_namespace="infra"
storage_size="30Gi"

cat <<EOF | kubectl create -f -
apiVersion: v1
items:
  - kind: StatefulSet
    apiVersion: apps/v1
    metadata:
      labels:
        app: clickhouse
        instance: clickhouse.${infra_domain}
      name: clickhouse
      namespace: ${infra_namespace}
    spec:
      replicas: 1
      serviceName: statefulset-clickhouse
      selector:
        matchLabels:
          app: clickhouse
          instance: clickhouse.${infra_domain}
      template:
        metadata:
          labels:
            app: clickhouse
            instance: clickhouse.${infra_domain}
        spec:
          containers:
            - name: clickhouse
              image: clickhouse/clickhouse-server:23.3.8
              imagePullPolicy: IfNotPresent
              ports:
                - name: http
                  containerPort: 8123
                  hostPort: 8123
                  protocol: TCP
              volumeMounts:
                - mountPath: /clickhouse
                  name: host-volume
                - mountPath: /etc/clickhouse-server/
                  name: configuration
              livenessProbe:
                httpGet:
                  path: /ping
                  port: 8123
                  scheme: HTTP
                initialDelaySeconds: 30
                timeoutSeconds: 30
              resources:
                limits:
```

```
                cpu: 3
                memory: 4G
              requests:
                cpu: 500m
                memory: 1G
        volumes:
          - configMap:
              name: clickhouse-configuration
            name: configuration
        nodeSelector:
          clickhouse: ""
    volumeClaimTemplates:
      - metadata:
          name: host-volume
          annotations:
            volume.beta.kubernetes.io/storage-class: local-storage
        spec:
          selector:
            matchLabels:
              app: clickhouse
              instance: clickhouse.${infra_domain}
          accessModes: [ "ReadWriteOnce" ]
          resources:
            requests:
              storage: ${storage_size}
- apiVersion: v1
  kind: Service
  metadata:
    labels:
      app: clickhouse
      instance: clickhouse.${infra_domain}
    name: clickhouse
    namespace: ${infra_namespace}
  spec:
    ports:
      - name: http
        port: 8123
        protocol: TCP
        targetPort: 8123
    type: ClusterIP
    selector:
      app: clickhouse
      instance: clickhouse.${infra_domain}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
EOF
```

6. Проверить:

    1. Состояние запуска контейнера

    ```
    kubectl get po -n infra clickhouse-0 -o wide -w
    ```

    2. Логи контейнера

    ```
    kubectl logs -n infra clickhouse-0 -f
    ```

    3. Возможность подключения с авторизационными данными

7. Для наполнения authfile выставить значения:

- clickhouse.host: "clickhouse.in.monq.local";
- clickhouse.port: 8123;
- clickhouse.proto: http;
- clickhouse.users.root_user.name: "admin";
- clickhouse.users.root_user.password: "<check ${ch_root_password}>".

> Внимание! Значение вышеуказанных переменных заполнены с учетом текущего примера.

HA cluster Clickhouse

Zookeeper

Запуск осуществляется с помощью оператора zookeeper-operator от pravega.

> Внимание! Для установки требуется Helm, см. # Приложение A13. Установка helm.

1. Установить оператор:

```
helm repo add pravega https://charts.pravega.io
helm install zookeeper-operator pravega/zookeeper-operator \
 --version 0.2.15 --namespace zookeeper-operator --create-namespace
```

2. На нодах, где будет запушен zookeeper создать каталоги для хранения данных:

> Внимание! Эта операция выполняется не на сервере controlplane, а на сервере, где будет запущен zookeeper.

```
mkdir -p /storage/zookeeper
```

3. Создать PV zookeeper:

```
zk_storage_size="5Gi"
host_path="/storage/zookeeper"

for server_number in {1..3}; do
cat <<EOF | kubectl create -f -
apiVersion: v1
kind: PersistentVolume
metadata:
  name: zookeeper-pv-w${server_number}
  labels:
    service: zookeeper
spec:
  capacity:
    storage: ${zk_storage_size}
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
```

```
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: ${host_path}
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - clickhouse-${server_number}
EOF
done
```

4. Запустить zookeeper:

```
zk_storage_size="1Gi"

cat <<EOF | kubectl create -f -
apiVersion: "zookeeper.pravega.io/v1beta1"
kind: "ZookeeperCluster"
metadata:
  name: "zookeeper"
  namespace: "zookeeper-operator"
spec:
  replicas: 3
  persistence:
    spec:
      storageClassName: local-storage
      resources:
        requests:
          storage: ${zk_storage_size}
EOF
```

Clickhouse

Запуск осуществляется с помощью оператора clickhouse-operator от altinity.

> Внимание! Для установки требуется Helm, см. # Приложение A13. Установка helm.

1. Установить оператор

```
helm repo add clickhouse-operator https://docs.altinity.com/clickhouse-operator/
helm install clickhouse-operator clickhouse-operator/altinity-clickhouse-operator \
--version 0.21.3 --namespace clickhouse-operator --create-namespace
```

2. На нодах, где будет запушен clickhouse создать каталоги для хранения данных

> Внимание! Эта операция выполняется не на сервере controlplane, а на сервере, где будет запущен clickhouse.

```
mkdir -p /storage/clickhouse
```

3. Создать storage-class (и pv в случае с hostPath тк этот csi driver не поддерживает provisioning)

```
host_path="/storage/clickhouse"
ch_storage_size="30Gi"

cat <<EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
EOF

for server_number in {1..3}; do
cat <<EOF | kubectl create -f -
apiVersion: v1
kind: PersistentVolume
metadata:
  name: clickhouse-pv-w${server_number}
  labels:
    service: clickhouse
spec:
  capacity:
    storage: ${ch_storage_size}
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: ${host_path}
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - clickhouse-${server_number}
EOF
done
```

4. Запуск кластера.

> Внимание! Запомнить пароль clickhouse, для последующего наполнения authfile.

```
ch_storage_size="5Gi"
ch_root_password=$(openssl rand -base64 16)
ch_root_password_hash=$(echo -n "${ch_root_password}" | sha256sum | sed s/\ .*$//g)
echo save it: ${ch_root_password}

cat <<EOF | kubectl create -f -
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "monq-clickhouse"
  namespace: clickhouse-operator
spec:
  defaults:
    templates:
```

```
      dataVolumeClaimTemplate: default
      podTemplate: clickhouse:23.3.8
  configuration:
    users:
      admin/access_management: "1"
      admin/password_sha256_hex: ${ch_root_password_hash}
      admin/profile: default
      admin/quota: default
      admin/networks/ip: "0.0.0.0/0"
    zookeeper:
      nodes:
        - host: zookeeper-client.zookeeper-operator
    clusters:
      - name: monqch
        layout:
          shardsCount: 1
          replicasCount: 3
  templates:
    volumeClaimTemplates:
      - name: default
        spec:
          storageClassName: local-storage
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: ${ch_storage_size}
          selector:
            matchLabels:
              service: clickhouse
    podTemplates:
      - name: clickhouse:23.3.8
        spec:
          containers:
            - name: clickhouse-pod
              image: clickhouse/clickhouse-server:23.3.8
EOF
```

Кластер остается работоспособным при одной ноде ch и двух zookeper.

5. Проверить:

    1. Состояние запуска контейнеров:

        ```
        kubectl get po -n clickhouse-operator -w
        ```

    2. Логи контейнеров на предмет ошибок;

    3. Возможность подключения с авторизационными данными.

        Выполнить тестовый запрос к ноде, где запущен экземпляр clickhouse, например с помощью curl:

        ```
        ch_port=$(kubectl get svc -n clickhouse-operator clickhouse-monq-clickhouse \
          -o jsonpath='{.spec.ports[?(@.name=="http")].nodePort}')
        ch_ip=$(kubectl get no clickhouse-1 \
          -o jsonpath='{.status.addresses[?(@.type=="InternalIP")].address}')

        ch_root_password=< пароль сгенерированный ранее >
        ch_user_name=< имя пользователя, обычно admin >
        ```

```
echo 'SELECT * FROM system.settings' \
  | curl "http://${ch_user_name}:${ch_root_password}@${ch_ip}:${ch_port}/" \
  --data-binary @-
```

4. Распространение данных в кластере, чтение и запись.

   Подключиться к одному из серверов clickhouse создать тестовую бд и таблицу:

```
CREATE DATABASE test_cl_stream_data_service ON CLUSTER monqch ENGINE = Ordinary;

CREATE TABLE test_cl_stream_data_service.stream1 ON CLUSTER monqch
(
    `_id` UUID,
    `_rawId` UUID,
    `_stream.id` Int64,
    `_stream.name` String,
    `_date` Date,
    `_userspaceId` Int64
)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/stream1', '{replica}')
PARTITION BY _date
ORDER BY (_date)
SETTINGS enable_mixed_granularity_parts = 1,
index_granularity = 8192;
```

   Подключиться к другому серверу clickhouse в кластере и проверить наличие тестовой бд и таблицы. Адрес другого сервера можно выяснить вот так:

```
kubectl get no clickhouse-2 \
  -o jsonpath='{.status.addresses[?(@.type=="InternalIP")].address}'
```

6. Для наполнения authfile выставить значения:

   - clickhouse.host: `"clickhouse-monq-clickhouse.clickhouse-operator.svc.cluster.local"`;
   - clickhouse.port: 8123;
   - clickhouse.cluster: **true**;
   - clickhouse.cluster_name: `"monqch"`;
   - clickhouse.users.root_user.name: `"admin"`;
   - clickhouse.users.root_user.password: `"<check ${ch_root_password}>"`.

> Внимание! Значение вышеуказанных переменных заполнены с учетом текущего примера.

Приложение A9. ArangoDB.

Standalone ArangoDB

1. Выбрать сервер, на котором будет запущен сервис.

   > Внимание! В данном примере используется hostpath для хранения файлов контейнера. По этому размещение контейнера должно быть явно назначено на указанный сервер.

   > Внимание! В данном случае используется каталог на сервере, данное решение не рекомендовано к запуску в промышленной среде.

   Выполнить назначение метки на выбранную ноду:

   ```
   server_name="db"
   kubectl label no ${server_name} arangodb=
   ```

2. Создать каталог для хранения данных.

   1. На сервере, где будет размещен контейнер создать каталог для хранения данных:

      > Внимание! Эта операция выполняется не на сервере controlplane, а на сервере, где будет запущен контейнер.

      ```
      mkdir -p /storage/arangodb
      ```

   2. На сервере controlplane создать манифест хранилища данных, в данном случае используется каталог на сервере hostPath, данное решение не рекомендовано к запуску в промышленной среде

      ```
      infra_domain="in.monq.local"
      host_path="/storage/arangodb"
      storage_size="30Gi"

      cat <<EOF | kubectl create -f -
      apiVersion: v1
      kind: PersistentVolume
      metadata:
      ```

```
  name: arangodb-pv
  labels:
    app: arangodb
    instance: arangodb.${infra_domain}
spec:
  capacity:
    storage: ${storage_size}
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: ${host_path}
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: arangodb
              operator: Exists
EOF
```

3. Создать конфигурационный файл в kubernetes. В данном конфигурационном файле предложена примерная конфигурация СУБД, пользователь может изменить её исходя из своих потребностей.

```
infra_domain="in.monq.local"
infra_namespace="infra"

cat <<EOF | kubectl create -f -
kind: ConfigMap
apiVersion: v1
metadata:
  name: arangodb-configuration
  namespace: ${infra_namespace}
  labels:
    app: arangodb
    instance: arangodb.${infra_domain}
data:
  arangod.conf: |
    [database]
    directory = /var/lib/arangodb3

    [server]
    endpoint = tcp://0.0.0.0:8529
    storage-engine = rocksdb
    authentication = true
    statistics = false

    [javascript]
    startup-directory = /usr/share/arangodb3/js
    app-path = /var/lib/arangodb3-apps
    enabled = true
    v8-contexts = 10
    v8-max-heap = 512

    [foxx]
    queues = false

    [log]
    level = info
    file = -
```

```
    [cluster]

    [rocksdb]
    total-write-buffer-size = 100MiB
    write-buffer-size = 50MiB
    max-total-wal-size = 100MiB
    block-cache-size = 256MiB
    enforce-block-cache-size-limit = true

    [cache]
    size = 256MiB
EOF
```

4. Задать авторизационные данные пользователя root

```
infra_domain="in.monq.local"
infra_namespace="infra"

arangodb_root_password=$(openssl rand -base64 16)

cat <<EOF | kubectl create -f -
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: arangodb-secret
  namespace: ${infra_namespace}
  labels:
    app: arangodb
    instance: arangodb.${infra_domain}
data:
  ARANGO_ROOT_PASSWORD: $(echo -n "$arangodb_root_password" | base64)
EOF
```

Впоследствии пароль можно будет посмотреть с помощью команды:

```
infra_namespace="infra"

kubectl get secrets -n ${infra_namespace} arangodb-secret \
  -o jsonpath='{.data.ARANGO_ROOT_PASSWORD}' | base64 --decode
```

5. Применить манифест.

```
infra_domain="in.monq.local"
infra_namespace="infra"
storage_size="30Gi"

cat <<EOF | kubectl create -f -
apiVersion: v1
items:
- kind: StatefulSet
  apiVersion: apps/v1
  metadata:
    labels:
      app: arangodb
      instance: arangodb.${infra_domain}
    name: arangodb
    namespace: ${infra_namespace}
  spec:
    replicas: 1
    serviceName: statefulset-arangodb
```

```
    selector:
      matchLabels:
        app: arangodb
        instance: arangodb.${infra_domain}
    template:
      metadata:
        labels:
          app: arangodb
          instance: arangodb.${infra_domain}
      spec:
        containers:
          - name: arangodb
            image: arangodb:3.11.2
            imagePullPolicy: IfNotPresent
            env:
              - name: ARANGO_STORAGE_ENGINE
                value: rocksdb
            envFrom:
              - secretRef:
                  name: arangodb-secret
            ports:
              - name: http
                containerPort: 8529
                hostPort: 8529
                protocol: TCP
            volumeMounts:
              - name: host-volume
                mountPath: /var/lib/arangodb3
                subPath: var/lib/arangodb3
              - name: host-volume
                mountPath: /var/lib/arangodb3-apps
                subPath: var/lib/arangodb3-apps
              - mountPath: /etc/arangodb3/arangod.conf
                name: configuration
                subPath: arangod.conf
            livenessProbe:
              httpGet:
                path: /
                port: 8529
                scheme: HTTP
              initialDelaySeconds: 30
              timeoutSeconds: 30
            resources:
              limits:
                cpu: 2
                memory: 2048Mi
              requests:
                cpu: 250m
                memory: 256Mi
        volumes:
          - configMap:
              name: arangodb-configuration
            name: configuration
        nodeSelector:
          arangodb: ""
    volumeClaimTemplates:
    - metadata:
        name: host-volume
        annotations:
          volume.beta.kubernetes.io/storage-class: local-storage
      spec:
        selector:
          matchLabels:
            app: arangodb
```

```
            instance: arangodb.${infra_domain}
        accessModes: ["ReadWriteOnce"]
        resources:
          requests:
            storage: ${storage_size}
- apiVersion: v1
  kind: Service
  metadata:
    labels:
      app: arangodb
      instance: arangodb.${infra_domain}
    name: arangodb
    namespace: ${infra_namespace}
  spec:
    ports:
    - name: http
      port: 8529
      protocol: TCP
      targetPort: 8529
    type: ClusterIP
    selector:
      app: arangodb
      instance: arangodb.${infra_domain}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
EOF
```

6. Проверить:

    1. Состояние запуска контейнера

    ```
    kubectl get po -n infra arangodb-0 -o wide -w
    ```

    2. Логи контейнера

    ```
    kubectl logs -n infra arangodb-0 -f
    ```

    3. Возможность подключения с авторизационными данными

7. Для наполнения authfile выставить значения:

    - arangodb.host: `arangodb.in.monq.local`;
    - arangodb.port: `8529`;
    - arangodb.proto: `http`;
    - arangodb.users.root_user.name: `root`;
    - arangodb.users.root_user.password: `< check arangodb password >`.

    > Внимание! Значение вышеуказанных переменных заполнены с учетом текущего примера.

HA cluster ArangoDB

Запуск осуществляется с помощью оператора kube-arangodb

> Внимание! Для установки требуется Helm, см. # Приложение A13. Установка helm.

Может быть запущен в двух режимах:

- cluster - кластерная конфигурация master-master с шардированием данных;
- activeFailover - master-slave.

В инструкции рассматривается только activefailover, т.к. monq не работает с шаридированием данных.

1. Установить оператор:

```
git_url="https://github.com/arangodb/kube-arangodb/releases/download/1.2.32"

helm install kube-arango-crd ${git_url}/kube-arangodb-crd-1.2.32.tgz \
  --namespace arangodb-operator --create-namespace
helm install kube-arango ${git_url}/kube-arangodb-1.2.32.tgz \
  --set "operator.features.storage=true" \
  --set "operator.replicaCount=1" \
  --namespace arangodb-operator
```

   Опция operator.features.storage=true используется в случае если нет быстрого storage. В этом случае будет поднят сервис arango-storage на каждой ноде.

2. Назначить сервера на которых будет запущена arangodb в режиме HA. Для примера будут использованы сервера arangodb-1, arangodb-2, arangodb-3. Данные сервера должны быть уже настроены и включены в кластер kubernetes.

   Выставить метки на ноды:

```
kubectl label node arangodb-1 arangodb-2 arangodb-3 arangodb=
```

3. Для запуска arango-storage необходимо создать ресурс:

```
host_path="/storage/arangodb"

cat <<EOF | kubectl create -f -
apiVersion: "storage.arangodb.com/v1alpha"
kind: "ArangoLocalStorage"
metadata:
  name: "arangodb-storage"
spec:
  storageClass:
    name: arango-storage
  localPath:
  - ${host_path}
  nodeSelector:
    arangodb: ""
EOF
```

4. Запуск в ActiveFailover режиме:

```
arango_storage_size="30Gi"
agent_storage_size="1Gi"

cat <<EOF | kubectl create -f -
apiVersion: "database.arangodb.com/v1"
kind: "ArangoDeployment"
metadata:
  name: "monq-arangodb-cluster"
  namespace: arangodb-operator
spec:
  mode: ActiveFailover
  environment: Production
  disableIPv6: true
  tls:
    caSecretName: None
  agents:
    count: 3
    resources:
      requests:
        storage: ${agent_storage_size}
    storageClassName: arango-storage
    nodeSelector:
      arangodb: ""
  single:
    count: 3
    resources:
      requests:
        storage: ${arango_storage_size}
    storageClassName: arango-storage
    nodeSelector:
      arangodb: ""
  image: "arangodb/arangodb:3.11.2"
EOF
```

   Кластер остается работоспособным если доступны: 1 single и 2 agent.

5. Перейти в веб интерфейс, задать пользователю `root` пароль, по умолчанию используется пустой пароль.

   Для определения точки подключения извне к веб интерфейсу нужно узнать адрес порта у сервиса `monq-arangodb-cluster-ea`. После подключиться к кластеру через <ip ноды:найденный порт>, более подробно см документация arangodb.

6. Проверить:

   1. Состояние запуска контейнеров:

      ```
      kubectl get po -n arangodb-operator -o wide -w
      ```

   2. Логи контейнеров на предмет ошибок;
   3. Возможность подключения с авторизационными данными;
   4. Распространение данных в кластере, чтение и запись.

7. Для наполнения authfile выставить значения:

- arangodb.host: `monq-arangodb-cluster.arangodb-operator.svc.cluster.local`;
- arangodb.port: 8529;
- arangodb.proto: `http`;
- arangodb.users.root_user.name: `root`;
- arangodb.users.root_user.password: < `check arangodb password` >.

> Внимание! Значение вышеуказанных переменных заполнены с учетом текущего примера.

> Внимание! Если запускать ArangoDB с помощью данного оператора, установщик должен быть запущен внутри кластера k8s.

Приложение A10. VictoriaMetrics.

Standalone VictoriaMetrics

Порядок запуска:

1. Выбрать сервер, на котором будет запущен сервис.

   > Внимание! В данном примере используется hostpath для хранения файлов контейнера. По этому размещение контейнера должно быть явно назначено на указанный сервер.

   > Внимание! В данном случае используется каталог на сервере, данное решение не рекомендовано к запуску в промышленной среде.

   Выполнить назначение метки на выбранную ноду:

   ```
   server_name="db"
   kubectl label no ${server_name} victoriametrics=
   ```

2. Создать каталог для хранения данных.

   1. На сервере, где будет размещен контейнер создать каталог для хранения данных:

      > Внимание! Эта операция выполняется не на сервере controlplane, а на сервере, где будет запущен контейнер.

      ```
      mkdir -p /storage/victoriametrics
      ```

   2. На сервере controlplane создать манифест хранилища данных, в данном случае используется каталог на сервере, данное решение не рекомендовано к запуску в промышленной среде

      ```
      infra_domain="in.monq.local"
      host_path="/storage/victoriametrics"
      storage_size="30Gi"

      cat <<EOF | kubectl create -f -
      apiVersion: v1
      ```

```
kind: PersistentVolume
metadata:
  name: victoriametrics-pv
  labels:
    app: victoriametrics
    instance: victoriametrics.${infra_domain}
spec:
  capacity:
    storage: ${storage_size}
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: ${host_path}
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: victoriametrics
          operator: Exists
EOF
```

3. Задать авторизационные данные пользователя root

```
infra_domain="in.monq.local"
infra_namespace="infra"

vm_root_password=$(openssl rand -base64 16)

cat <<EOF | kubectl create -f -
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: victoriametrics-secret
  namespace: ${infra_namespace}
  labels:
    app: victoriametrics
    instance: victoriametrics.${infra_domain}
data:
  VM_AUTH_USER: $(echo -n monq | base64)
  VM_AUTH_PASSWORD: $(echo -n $vm_root_password | base64)
EOF
```

Впоследствии пароль можно будет посмотреть с помощью команды:

```
infra_namespace="infra"

kubectl get secrets -n ${infra_namespace} victoriametrics-secret \
  -o jsonpath='{.data.VM_AUTH_PASSWORD}' | base64 --decode
```

4. Применить манифест.

```
infra_domain="in.monq.local"
infra_namespace="infra"
storage_size="30Gi"

cat <<EOF | kubectl create -f -
apiVersion: v1
items:
```

```
- kind: StatefulSet
  apiVersion: apps/v1
  metadata:
    labels:
      app: victoriametrics
      instance: victoriametrics.${infra_domain}
    name: victoriametrics
    namespace: ${infra_namespace}
  spec:
    replicas: 1
    serviceName: statefulset-victoriametrics
    selector:
      matchLabels:
        app: victoriametrics
        instance: victoriametrics.${infra_domain}
    template:
      metadata:
        labels:
          app: victoriametrics
          instance: victoriametrics.${infra_domain}
      spec:
        containers:
          - name: victoriametrics
            image: victoriametrics/victoria-metrics:v1.91.3
            imagePullPolicy: IfNotPresent
            args:
              - "-http.disableResponseCompression=true"
              - "-retentionPeriod=3"
              - "-memory.allowedBytes=2GB"
              - "-denyQueryTracing=true"
              - "-httpAuth.username=%{VM_AUTH_USER}"
              - "-httpAuth.password=%{VM_AUTH_PASSWORD}"
              - "-flagsAuthKey=%{VM_AUTH_PASSWORD}"
              - "-pprofAuthKey=%{VM_AUTH_PASSWORD}"
            envFrom:
              - secretRef:
                  name: victoriametrics-secret
            ports:
              - name: http
                containerPort: 8428
                hostPort: 8428
                protocol: TCP
            volumeMounts:
              - mountPath: /victoria-metrics-data
                name: host-volume
                subPath: victoria-metrics-data
            livenessProbe:
              httpGet:
                path: /health
                port: 8428
                scheme: HTTP
              initialDelaySeconds: 30
              timeoutSeconds: 30
            resources:
              limits:
                cpu: 2
                memory: 2Gi
              requests:
                cpu: 250m
                memory: 256Mi
        nodeSelector:
          victoriametrics: ""
    volumeClaimTemplates:
    - metadata:
```

```
        name: host-volume
        annotations:
          volume.beta.kubernetes.io/storage-class: local-storage
      spec:
        selector:
          matchLabels:
            app: victoriametrics
            instance: victoriametrics.${infra_domain}
        accessModes: ["ReadWriteOnce"]
        resources:
          requests:
            storage: ${storage_size}
- apiVersion: v1
  kind: Service
  metadata:
    name: victoriametrics
    namespace: ${infra_namespace}
    labels:
      app: victoriametrics
      instance: victoriametrics.${infra_domain}
  spec:
    ports:
    - name: http
      port: 8428
      protocol: TCP
      targetPort: 8428
    type: ClusterIP
    selector:
      app: victoriametrics
      instance: victoriametrics.${infra_domain}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
EOF
```

5. Проверить:

    1. Состояние запуска контейнера

    ```
    kubectl get po -n infra victoriametrics-0 -o wide -w
    ```

    2. Логи контейнера

    ```
    kubectl logs -n infra victoriametrics-0 -f
    ```

    3. Возможность подключения с авторизационными данными

6. Для наполнения authfile выставить значения:

    - victoriametrics.host: `victoriametrics.in.monq.local`;
    - victoriametrics.port: `8428`;
    - victoriametrics.proto: `http`;
    - victoriametrics.auth_type: `BasicAuth`;
    - victoriametrics.users.root_user.name: `monq`;
    - victoriametrics.users.root_user.password: `"<check ${vm_root_password}"`>.

> Внимание! Значение вышеуказанных переменных заполнены с учетом текущего примера.

HA cluster VictoriaMetrics

Запуск осуществляется с помощью оператора victoria-metrics-operator.

> Внимание! Для установки требуется Helm, см. # Приложение A13. Установка helm.

Может быть запущен в двух режимах:

- прямое обращение в vminsert и vmselect (в данном режиме запросы на чтение и вставку должны осуществляться к разным хостам, отсутствует авторизация, должны быть использованы кластерные url);
- обращение к компонентам через vmauth (в данном случае есть возможность установки авторизации, обращение осуществляется к одному хосту, но присутствует дополнительный узел).

1. Установить оператор:

```
helm repo add vm https://victoriametrics.github.io/helm-charts/
helm install vmoperator vm/victoria-metrics-operator -n victoria --create-namespace
```

2. На нодах, где будет запушена victoriametrics создать каталоги для хранения данных

> Внимание! Эта операция выполняется не на сервере controlplane, а на сервере, где будет запущена victoriametrics.

```
mkdir -p /storage/victoriametrics
```

3. Создать storage-class(и pv в случае с hostPath тк этот csi driver не поддерживает provisioning):

> Внимание! Обратить внимание на привязку pv к ноде, указаны имена серверов из примера, заменить на свои.

```
host_path="/storage/victoriametrics"
storage_size="30Gi"

cat <<EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-storage
provisioner: kubernetes.io/no-provisioner
```

```
volumeBindingMode: WaitForFirstConsumer
EOF

for server_number in {1..3}; do
cat <<EOF | kubectl create -f -
apiVersion: v1
kind: PersistentVolume
metadata:
  name: victoria-pv-w${server_number}
  labels:
    service: victoria
spec:
  capacity:
    storage: ${storage_size}
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: ${host_path}
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - victoria-${server_number}
EOF
done
```

4. Задать авторизационные данные пользователя

```
vm_user="monq"
vm_password=$(openssl rand -base64 16)

cat <<EOF | kubectl create -f -
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: victoria-metrics-auth
  namespace: victoria
data:
  VM_AUTH_USER: $(echo -n $vm_user | base64)
  VM_AUTH_PASSWORD: $(echo -n $vm_password | base64)
EOF
```

Впоследствии пароль можно будет посмотреть с помощью команды:

```
kubectl get secrets -n victoria victoria-metrics-auth \
  -o jsonpath='{.data.VM_AUTH_PASSWORD}' | base64 --decode
```

5. Запустить кластер

```
storage_size="30Gi"

cat <<EOF | kubectl create -f -
apiVersion: operator.victoriametrics.com/v1beta1
kind: VMCluster
metadata:
```

```
  name: monq
  namespace: victoria
spec:
  retentionPeriod: "1"
  replicationFactor: 2
  clusterVersion: v1.91.3-cluster
  vmstorage:
    replicaCount: 3
    storageDataPath: "/vm-data"
    affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: "app.kubernetes.io/name"
                operator: In
                values:
                - "vmstorage"
            topologyKey: "kubernetes.io/hostname"
    storage:
      volumeClaimTemplate:
        spec:
          storageClassName: local-storage
          selector:
            matchLabels:
              service: victoria
          resources:
            requests:
              storage: ${storage_size}
    resources:
      limits:
        cpu: "1"
        memory: 2048Mi
  vmselect:
    extraArgs:
      httpAuth.username: "%{VM_AUTH_USER}"
      httpAuth.password: "%{VM_AUTH_PASSWORD}"
    extraEnvs:
      - name: VM_AUTH_USER
        valueFrom:
          secretKeyRef:
            name: victoria-metrics-auth
            key: VM_AUTH_USER
      - name: VM_AUTH_PASSWORD
        valueFrom:
          secretKeyRef:
            name: victoria-metrics-auth
            key: VM_AUTH_PASSWORD
    replicaCount: 2
    cacheMountPath: "/select-cache"
    affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: "app.kubernetes.io/name"
                operator: In
                values:
                - "vmselect"
            topologyKey: "kubernetes.io/hostname"
  vminsert:
    extraArgs:
      httpAuth.username: "%{VM_AUTH_USER}"
      httpAuth.password: "%{VM_AUTH_PASSWORD}"
```

```
    extraEnvs:
      - name: VM_AUTH_USER
        valueFrom:
          secretKeyRef:
            name: victoria-metrics-auth
            key: VM_AUTH_USER
      - name: VM_AUTH_PASSWORD
        valueFrom:
          secretKeyRef:
            name: victoria-metrics-auth
            key: VM_AUTH_PASSWORD
    replicaCount: 2
    affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: "app.kubernetes.io/name"
                operator: In
                values:
                - "vminsert"
            topologyKey: "kubernetes.io/hostname"
EOF
```

Будет создано по два экземпляра vminsert, vmselect и три vmstorage. Все компоненты независимы, при вставке на любой из vminsert происходит запись в каждый доступный vmstorage.

Для обеспечения консистентности данных необходимо включать replicationFactor=N, который заставляет vmInsert производить вставку минимум на N экземпляров vmstorage. При этом количество экземпляров vmstrorage должно быть >=N+1, таким образом кластер останется в работе при N Доступных экземплярах vmStorage.

После запуска будут созданы сервисы для обращения к vminsert и vmselect.

При необходимости доступа извне их можно опубликовать через ingress или loadBalancer.

В данном примере кластер запущен без авторизации. Для ограничения доступа рекомендуется ознакомиться с официальной документацией.

6. Проверить, в данном примере мы рассматриваем конфигурацию без vmauth, только vmselect и vm insert:

    1. Состояние запуска контейнеров:

    ```
    kubectl get po -n victoria -w
    ```

    2. Логи контейнеров на предмет ошибок;

    3. Возможность подключения и распространение данных в кластере. Получить список параметров для подключения:

    ```
    vm_insert_host=$(kubectl get ep -n victoria vminsert-monq \
      -o jsonpath='{.subsets[].addresses[*].ip}')
    ```

```
vm_select_host=$(kubectl get ep -n victoria vmselect-monq \
  -o jsonpath='{.subsets[].addresses[*].ip}')
vm_user=$(kubectl get secrets -n victoria victoria-metrics-auth \
 -o jsonpath='{.data.VM_AUTH_USER}' | base64 --decode)
vm_password=$(kubectl get secrets -n victoria victoria-metrics-auth \
 -o jsonpath='{.data.VM_AUTH_PASSWORD}' | base64 --decode)
```

Выполнить запросы на запись в vminsert:

```
for instance in ${vm_insert_host}; do
  curl -d 'metric_name{foo="bar"} 123' --user ${vm_user}:${vm_password} \
    -X POST http://${instance}:8480/insert/0/prometheus/api/v1/import/prometheus
done
```

Выполнить запросы на чтение с vmselect:

```
for instance in ${vm_select_host}; do
  curl http://${instance}:8481/select/0/prometheus/api/v1/query \
    --user ${vm_user}:${vm_password} \
    -d 'query=metric_name{foo="bar"}' && echo ""
done
```

Для проверки можно поочередно перезагружать сервера повторяя запросы на запись чтение.

7. Для наполнения authfile выставить значения:

   - victoriametrics.cluster: **true**;
   - victoriametrics.cluster_account: 0;
   - victoriametrics.auth_type: BasicAuth;
   - victoriametrics.cluster_insert.host: vminsert-monq.victoria.svc.cluster.local;
   - victoriametrics.cluster_insert.port: 8480;
   - victoriametrics.cluster_insert.proto: http;
   - victoriametrics.cluster_select.host: vmselect-monq.victoria.svc.cluster.local;
   - victoriametrics.cluster_select.port: 8481;
   - victoriametrics.cluster_select.proto: http;
   - victoriametrics.users.root_user.name: "<check ${vm_user}">;
   - victoriametrics.users.root_user.password: "<check ${vm_password}">.

> Внимание! Значение вышеуказанных переменных заполнены с учетом текущего примера.

Приложение A11. Redis.

Standalone Redis.

Порядок запуска:

1. Выбрать сервер, на котором будет запущен сервис.

   > Внимание! В данном примере используется hostpath для хранения файлов контейнера. По этому размещение контейнера должно быть явно назначено на указанный сервер.

   > Внимание! В данном случае используется каталог на сервере, данное решение не рекомендовано к запуску в промышленной среде.

   Выполнить назначение метки на выбранную ноду:

   ```
   server_name="db"
   kubectl label no ${server_name} redis=
   ```

2. Создать каталог для хранения данных.

   1. На сервере, где будет размещен контейнер создать каталог для хранения данных:

      > Внимание! Эта операция выполняется не на сервере controlplane, а на сервере, где будет запущен контейнер.

      ```
      mkdir -p /storage/redis
      ```

   2. На сервере controlplane создать манифест хранилища данных, в данном случае используется каталог на сервере, данное решение не рекомендовано к запуску в промышленной среде

      ```
      infra_domain="in.monq.local"
      host_path="/storage/redis"
      storage_size="5Gi"

      cat <<EOF | kubectl create -f -
      apiVersion: v1
      ```

```
kind: PersistentVolume
metadata:
  name: redis-pv
  labels:
    app: redis
    instance: redis.${infra_domain}
spec:
  capacity:
    storage: ${storage_size}
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: ${host_path}
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: redis
          operator: Exists
EOF
```

3. Создать конфигурационный файл в kubernetes. В данном конфигурационном файле предложена примерная конфигурация СУБД, пользователь может изменить её исходя из своих потребностей.

```
infra_domain="in.monq.local"
infra_namespace="infra"

cat <<EOF | kubectl create -f -
kind: ConfigMap
apiVersion: v1
metadata:
  name: redis-configuration
  namespace: ${infra_namespace}
  labels:
    app: redis
    instance: redis.${infra_domain}
data:
  redis.conf: |
    protected-mode no
    port 6379
    timeout 0
    tcp-keepalive 300
    daemonize no
    supervised no
    pidfile /var/run/redis_6379.pid
    loglevel notice
    logfile ""
    databases 16
    always-show-logo yes
    save 900 1
    save 300 10
    save 60 1000
    stop-writes-on-bgsave-error yes
    rdbcompression yes
    rdbchecksum yes
    dbfilename dump.rdb
    dir /data/
    slave-serve-stale-data yes
```

```
    slave-read-only yes
    repl-diskless-sync no
    repl-diskless-sync-delay 5
    repl-disable-tcp-nodelay no
    slave-priority 100
    maxmemory 512mb
    maxmemory-policy allkeys-lru
    lazyfree-lazy-eviction yes
    lazyfree-lazy-expire yes
    lazyfree-lazy-server-del yes
    slave-lazy-flush yes
    appendonly yes
    appendfilename "appendonly.aof"
    appendfsync everysec
    no-appendfsync-on-rewrite no
    auto-aof-rewrite-percentage 10
    auto-aof-rewrite-min-size 64mb
    aof-load-truncated yes
    aof-use-rdb-preamble no
    lua-time-limit 5000
    slowlog-log-slower-than 10000
    slowlog-max-len 128
    latency-monitor-threshold 0
    notify-keyspace-events ""
    hash-max-ziplist-entries 512
    hash-max-ziplist-value 64
    list-max-ziplist-size -2
    list-compress-depth 0
    set-max-intset-entries 512
    zset-max-ziplist-entries 128
    zset-max-ziplist-value 64
    hll-sparse-max-bytes 3000
    activerehashing yes
    client-output-buffer-limit normal 0 0 0
    client-output-buffer-limit slave 256mb 64mb 60
    client-output-buffer-limit pubsub 32mb 8mb 60
    hz 10
    aof-rewrite-incremental-fsync yes
EOF
```

4. Задать авторизационные данные пользователя

```
infra_domain="in.monq.local"
infra_namespace="infra"

redis_password=$(openssl rand -base64 16)

cat <<EOF | kubectl create -f -
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: redis-secret
  namespace: ${infra_namespace}
  labels:
    app: redis
    instance: redis.${infra_domain}
data:
  REDIS_PASSWORD: $(echo -n ${redis_password} | base64)
EOF
```

Впоследствии пароль можно будет посмотреть с помощью команды:

```
infra_namespace="infra"

kubectl get secrets -n ${infra_namespace} redis-secret \
  -o jsonpath='{.data.REDIS_PASSWORD}' | base64 --decode
```

5. Применить манифест.

```
infra_domain="in.monq.local"
infra_namespace="infra"
storage_size="5Gi"

cat <<EOF | kubectl create -f -
apiVersion: v1
items:
- kind: StatefulSet
  apiVersion: apps/v1
  metadata:
    labels:
      app: redis
      instance: redis.${infra_domain}
    name: redis
    namespace: ${infra_namespace}
  spec:
    replicas: 1
    serviceName: statefulset-redis
    selector:
      matchLabels:
        app: redis
        instance: redis.${infra_domain}
    template:
      metadata:
        labels:
          app: redis
          instance: redis.${infra_domain}
      spec:
        containers:
          - name: redis
            image: redis:7.0.11
            imagePullPolicy: IfNotPresent
            args:
              - "/etc/redis/redis.conf"
              - "--requirepass"
              - "\$(REDIS_PASSWORD)"
            envFrom:
              - secretRef:
                  name: redis-secret
            ports:
              - name: redis
                containerPort: 6379
                hostPort: 6379
                protocol: TCP
            volumeMounts:
              - mountPath: /data
                name: host-volume
                subPath: data
              - mountPath: /etc/redis/redis.conf
                name: configuration
                subPath: redis.conf
            livenessProbe:
              tcpSocket:
                port: 6379
              initialDelaySeconds: 30
              timeoutSeconds: 30
```

```
            resources:
              limits:
                cpu: 1
                memory: 600Mi
              requests:
                cpu: 500m
                memory: 256Mi
        volumes:
          - configMap:
              name: redis-configuration
            name: configuration
        nodeSelector:
          redis: ""
    volumeClaimTemplates:
    - metadata:
        name: host-volume
        annotations:
          volume.beta.kubernetes.io/storage-class: local-storage
      spec:
        selector:
          matchLabels:
            app: redis
            instance: redis.${infra_domain}
        accessModes: ["ReadWriteOnce"]
        resources:
          requests:
            storage: ${storage_size}
- apiVersion: v1
  kind: Service
  metadata:
    name: redis
    namespace: ${infra_namespace}
    labels:
      app: redis
      instance: redis.${infra_domain}
  spec:
    ports:
    - name: redis
      port: 6379
      protocol: TCP
      targetPort: 6379
    type: ClusterIP
    selector:
      app: redis
      instance: redis.${infra_domain}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
EOF
```

6. Проверить:

    1. Состояние запуска контейнера

    ```
    kubectl get po -n infra redis-0 -o wide -w
    ```

    2. Логи контейнера

    ```
    kubectl logs -n infra redis-0 -f
    ```

    3. Возможность подключения с авторизационными данными

7. Для наполнения authfile выставить значения:

    - redis.host: "redis.in.monq.local";
    - redis.port: 6379;
    - redis.users.root_user.password: "<check ${redis_password}">.

> Внимание! Значение вышеуказанных переменных заполнены с учетом текущего примера.

HA cluster Redis.

Запуск осуществляется с помощью оператора redis-operator от spotahome.

> Внимание! Для установки требуется Helm, см. # Приложение A13. Установка helm.

1. Установить оператор:

```
helm repo add redis-operator https://spotahome.github.io/redis-operator
helm install redis-operator redis-operator/redis-operator \
  --version 3.2.8 --namespace redis-operator --create-namespace
```

2. На нодах, где будет запушен redis создать каталоги для хранения данных

> Внимание! Эта операция выполняется не на сервере controlplane, а на сервере, где будет запущен redis.

```
mkdir -p /storage/redis
```

3. Создать storage-class и pv, так как csi driver hostPath не поддерживает provisioning.

> Внимание! Обратить внимание на привязку pv к ноде, указаны имена серверов из примера.

> Внимание! В данном примере создание PV выполняется в цикле, для трех серверов.

```
host_path="/storage/redis"
storage_size="5Gi"

cat <<EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
EOF
```

```
for server_number in {1..3}; do
cat <<EOF | kubectl create -f -
apiVersion: v1
kind: PersistentVolume
metadata:
  name: redis-pv-w${server_number}
  labels:
    service: redis
spec:
  capacity:
    storage: ${storage_size}
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: ${host_path}
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - redis-${server_number}
EOF
done
```

4. Создать секрет с данными для авторизации:

```
kubectl create secret generic monq-redis-auth \
  -n redis-operator --from-literal=password=someSecretPass
redis_password=$(kubectl get secret -n redis-operator \
  monq-redis-auth -o jsonpath='{.data.password}' | base64 -d)
```

5. Запуск кластера:

```
storage_size="5Gi"

cat <<EOF | kubectl create -f -
apiVersion: databases.spotahome.com/v1
kind: RedisFailover
metadata:
  name: monq-redis
  namespace: redis-operator
spec:
  auth:
    secretPath: monq-redis-auth
  sentinel:
    image: redis:7.0.11
    replicas: 3
  redis:
    image: redis:7.0.11
    replicas: 3
    storage:
      keepAfterDeletion: true
      persistentVolumeClaim:
        metadata:
          name: redisfailover-persistent-keep-data
        spec:
          accessModes:
            - ReadWriteOnce
```

```
          resources:
            requests:
              storage: ${storage_size}
          selector:
            matchLabels:
              service: redis
          storageClassName: local-storage
EOF
```

Подключение к redis осуществляется с помощью sentinel.

Подключение возможно только из k8s так как сервис мастера не доступен вне кластера.

Кластер остается работоспособным при двух работоспособных экземплярах.

6. Проверить:

    1. Состояние запуска контейнеров:

        ```
        kubectl get po -n redis-operator -w
        ```

    2. Логи контейнеров на предмет ошибок
    3. Возможность подключения с авторизационными данными и распространение данных в кластере в режиме HA, чтение и запись. Из-за большого количества шагов вынесено в отдельный документ Инструкция.

7. Для наполнения authfile выставить значения:

    - redis.host: `"rfs-monq-redis.redis-operator.svc.cluster.local"`;
    - redis.port: 26379;
    - redis.sentinel: **`true`**;
    - redis.service_name: `"mymaster"`;
    - redis.users.root_user.password: `"<check ${redis_password}"`>.

    > Внимание! Значение вышеуказанных переменных заполнены с учетом текущего примера.

Проверка соединения с redis в режиме HA.

Запустить отладочный контейнер:

```
infra_namespace="infra"

cat <<EOF | kubectl create -f -
apiVersion: v1
kind: Pod
metadata:
  name: redis-cli
  namespace: ${infra_namespace}
```

```
  labels:
    app: redis-cli
spec:
  containers:
  - name: redis-cli
    image: redis:7.0.11
    command: [ "/bin/bash", "-c", "--" ]
    args: [ "trap : TERM INT; sleep infinity & wait" ]
    resources:
      limits:
        cpu: 500m
        memory: 256Mi
      requests:
        cpu: 25m
        memory: 128Mi
EOF
```

Определить точку подключения:

```
kubectl get svc -n redis-operator rfs-monq-redis
```

Перейти внутрь отладочного контейнера для определения текущего мастера:

```
kubectl exec -ti -n infra redis-cli -- bash
redis-cli -h rfs-monq-redis.redis-operator.svc.cluster.local -p 26379
SENTINEL get-master-addr-by-name mymaster
# Ответом будет
1) "< ip текущего мастера >"
2) "< порт текущего мастера >"
exit
```

Подключиться к мастеру, создать тестовый ключ и выйти:

```
redis-cli -h < ip текущего мастера > -p < порт текущего мастера >
auth < пароль, см "Создать секрет с данными для авторизации" >
SET mykey "Hello"
GET mykey
exit
exit
```

Перезагрузить текущий мастер:

```
kubectl delete po -n redis-operator -l redisfailovers-role=master
```

Перейти внутрь отладочного контейнера для определения текущего мастера:

```
kubectl exec -ti -n infra redis-cli -- bash
redis-cli -h rfs-monq-redis.redis-operator.svc.cluster.local -p 26379
SENTINEL get-master-addr-by-name mymaster
# Ответом будет
1) "<ip текущего мастера>"
2) "порт< текущего мастера>"
exit
```

Подключиться к мастеру, получить тестовый ключ и выйти:

```
redis-cli -h <ip текущего мастера> -p порт< текущего мастера>
auth пароль<, см "Создать секрет с данными для авторизации">
GET mykey
```

```
# Ответом должен быть ключ, заданный в п. Подключиться" к мастеру, создать тестовый ключ и
    выйти"
"Hello"
exit
exit
```

Приложение A12. RabbitMQ.

Standalone RabbitMQ.

Порядок запуска:

1. Выбрать сервер, на котором будет запущен сервис.

   > Внимание! В данном примере используется hostpath для хранения файлов контейнера. По этому размещение контейнера должно быть явно назначено на указанный сервер.

   > Внимание! В данном случае используется каталог на сервере, данное решение не рекомендовано к запуску в промышленной среде.

   Выполнить назначение метки на выбранную ноду:

   ```
   server_name="db"
   kubectl label no ${server_name} rabbitmq=
   ```

2. Создать каталог для хранения данных.

   1. На сервере, где будет размещен контейнер создать каталог для хранения данных:

      > Внимание! Эта операция выполняется не на сервере controlplane, а на сервере, где будет запущен контейнер.

      ```
      mkdir -p /storage/rabbitmq
      ```

   2. На сервере controlplane создать манифест хранилища данных, в данном случае используется каталог на сервере, данное решение не рекомендовано к запуску в промышленной среде

      ```
      infra_domain="in.monq.local"
      host_path="/storage/rabbitmq"
      storage_size="5Gi"

      cat <<EOF | kubectl create -f -
      apiVersion: v1
      ```

```
kind: PersistentVolume
metadata:
  name: rabbitmq-pv
  labels:
    app: rabbitmq
    instance: rabbitmq.${infra_domain}
spec:
  capacity:
    storage: ${storage_size}
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: ${host_path}
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: rabbitmq
          operator: Exists
EOF
```

3. Создать конфигурационный файл в kubernetes. В данном конфигурационном файле предложена примерная конфигурация СУБД, пользователь может изменить её исходя из своих потребностей.

```
infra_domain="in.monq.local"
infra_namespace="infra"

cat <<EOF | kubectl create -f -
kind: ConfigMap
apiVersion: v1
metadata:
  name: rabbitmq-configuration
  namespace: ${infra_namespace}
  labels:
    app: rabbitmq
    instance: rabbitmq.${infra_domain}
data:
  rabbitmq.conf: |
    vm_memory_high_watermark.absolute = 600MiB
EOF
```

4. Применить манифест.

```
infra_domain="in.monq.local"
infra_namespace="infra"
storage_size="5Gi"

cat <<EOF | kubectl create -f -
apiVersion: v1
items:
- kind: StatefulSet
  apiVersion: apps/v1
  metadata:
    labels:
      app: rabbitmq
      instance: rabbitmq.${infra_domain}
    name: rabbitmq
```

```
    namespace: ${infra_namespace}
  spec:
    replicas: 1
    serviceName: statefulset-rabbitmq
    selector:
      matchLabels:
        app: rabbitmq
        instance: rabbitmq.${infra_domain}
    template:
      metadata:
        labels:
          app: rabbitmq
          instance: rabbitmq.${infra_domain}
      spec:
        containers:
          - name: rabbitmq
            image: rabbitmq:3.11.18-management
            imagePullPolicy: IfNotPresent
            env:
              - name: RABBITMQ_LOGS
                value: "-"
              - name: HOME
                value: "/var/lib/rabbitmq"
              - name: RABBITMQ_CONFIG_FILE
                value: "/opt/rabbitmq/etc/rabbitmq/rabbitmq.conf"
            ports:
              - name: amqp
                containerPort: 5672
                hostPort: 5672
                protocol: TCP
              - name: http
                containerPort: 15672
                hostPort: 15672
                protocol: TCP
            volumeMounts:
              - mountPath: /var/lib/rabbitmq
                name: host-volume
                subPath: var/lib/rabbitmq
              - mountPath: /opt/rabbitmq/etc/rabbitmq/
                name: configuration
            livenessProbe:
              httpGet:
                path: /
                port: 15672
                scheme: HTTP
              initialDelaySeconds: 30
              timeoutSeconds: 30
            resources:
              limits:
                cpu: 1
                memory: 800Mi
              requests:
                cpu: 500m
                memory: 256Mi
        volumes:
          - configMap:
              name: rabbitmq-configuration
            name: configuration
        nodeSelector:
          rabbitmq: ""
    volumeClaimTemplates:
    - metadata:
        name: host-volume
        annotations:
```

```
          volume.beta.kubernetes.io/storage-class: local-storage
      spec:
        selector:
          matchLabels:
            app: rabbitmq
            instance: rabbitmq.${infra_domain}
        accessModes: ["ReadWriteOnce"]
        resources:
          requests:
            storage: ${storage_size}
- apiVersion: v1
  kind: Service
  metadata:
    name: rabbitmq
    namespace: ${infra_namespace}
    labels:
      app: rabbitmq
      instance: rabbitmq.${infra_domain}
  spec:
    ports:
    - name: http
      port: 15672
      protocol: TCP
      targetPort: 15672
    - name: amqp
      port: 5672
      protocol: TCP
      targetPort: 5672
    type: ClusterIP
    selector:
      app: rabbitmq
      instance: rabbitmq.${infra_domain}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
EOF
```

5. Проверить:

    1. Состояние запуска контейнера

    ```
    kubectl get po -n infra rabbitmq-0 -o wide -w
    ```

    2. Логи контейнера

    ```
    kubectl logs -n infra rabbitmq-0 -f
    ```

    3. Возможность подключения с авторизационными данными

6. Создать пользователя с адмнистративными правами:

    1. Подключиться к веб интерфейсу rabbitmq логин/пароль - guest/guest;
    2. Создать пользователя root с правами Administrator, запомнить пароль;
    3. Удалить пользователя guest.

    Можно выполнить с помощью команд:

```
rabbitmq_root_password=$(openssl rand -base64 16)
```

```
echo "save it: "${rabbitmq_root_password}

curl -X PUT http://guest:guest@rabbitmq.in.monq.local:15672/api/users/root \
 -d '{"password": "'${rabbitmq_root_password}'", "tags":"administrator"}'

curl -X PUT http://guest:guest@rabbitmq.in.monq.local:15672/api/permissions/%2F/root \
 -d '{"configure":".*","write":".*","read":".*"}'

curl -X DELETE --user root:${rabbitmq_root_password} \
  http://rabbitmq.in.monq.local:15672/api/users/guest
```

7. Для наполнения authfile выставить значения:

   - rabbitmq.host: `"rabbitmq.in.monq.local"`;
   - rabbitmq.port: 15672;
   - rabbitmq.amqp_port: 5672;
   - rabbitmq.proto: `http`;
   - rabbitmq.virtual_host: /;
   - rabbitmq.users.root_user.name: `"root"`;
   - rabbitmq.users.root_user.password: `"<check ${rabbitmq_root_password}>"`.

> Внимание! Значение вышеуказанных переменных заполнены с учетом текущего примера.

HA cluster RabbitMQ.

Запуск осуществляется с помощью оператора rabbitmq-operator.

1. Установить оператор:

```
git_url="https://github.com/rabbitmq/cluster-operator/releases/download/v2.3.0"

kubectl apply -f \
  ${git_url}/cluster-operator.yml
```

2. На нодах, где будет запушен rabbitmq создать каталоги для хранения данных

> Внимание! Эта операция выполняется не на сервере controlplane, а на сервере, где будет запущен rabbitmq.

```
mkdir -p /storage/rabbitmq
```

3. Создать storage-class и pv, так как csi driver hostPath не поддерживает provisioning:

> Внимание! Обратить внимание на привязку pv к ноде, указаны имена серверов из примера.

> Внимание! В данном примере создание PV выполняется в цикле, для трех серверов.

```
host_path="/storage/rabbitmq"
storage_size="2Gi"

cat <<EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
EOF

for server_number in {1..3}; do
cat <<EOF | kubectl create -f -
apiVersion: v1
kind: PersistentVolume
metadata:
  name: rabbitmq-pv-w${server_number}
  labels:
    service: rabbitmq
spec:
  capacity:
    storage: ${storage_size}
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: ${host_path}
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - rabbitmq-${server_number}
EOF
done
```

4. Создать кластер rabbitmq

```
storage_size="2Gi"

cat <<EOF | kubectl apply -f -
apiVersion: rabbitmq.com/v1beta1
kind: RabbitmqCluster
metadata:
  name: monq-rabbitmq
  namespace: rabbitmq-system
spec:
  replicas: 3
  override:
    statefulSet:
```

```
      spec:
        template:
          spec:
            containers:
              - name: rabbitmq
                ports:
                  - containerPort: 15672
                    name: management
                    protocol: TCP
                    hostPort: 15672
  image: rabbitmq:3.11.18-management
  resources:
    requests:
      cpu: 500m
      memory: 200Mi
    limits:
      cpu: 1
      memory: 800Mi
  rabbitmq:
    additionalConfig: |
      vm_memory_high_watermark.relative = 0.9
  persistence:
    storageClassName: local-storage
    storage: "${storage_size}"
EOF
```

Кластер остается работоспособным при отказе одной ноды из трех;

5. Проверить:

    1. Состояние запуска контейнеров:

    ```
    kubectl get po -n rabbitmq-system -w
    ```

    2. Логи контейнеров на предмет ошибок;

    3. Возможность подключения с авторизационными данными:

    ```
    rabbit_ip=$(kubectl get no db-1 \
      -o jsonpath='{.status.addresses[?(@.type=="InternalIP")].address}')
    rabbit_username=$(kubectl get secrets -n rabbitmq-system \
      monq-rabbitmq-default-user -o jsonpath='{.data.username}' \
      | base64 -d)
    rabbit_password=$(kubectl get secrets -n rabbitmq-system \
      monq-rabbitmq-default-user -o jsonpath='{.data.password}' \
      | base64 -d)

    curl -s -H "content-type:application/json" \
      http://${rabbit_username}:${rabbit_password}@${rabbit_ip}:15672/api/overview \
      | jq
    ```

    4. Распространение данных в кластере, чтение и запись. В данном случае можно создать кворумную очередь и проверить наличие очереди и данные в ней на экземплярах rabbitmq.

6. Для наполнения authfile выставить значения:

    - rabbitmq.host: `"monq-rabbitmq.rabbitmq-system.svc.cluster.local"`;

- rabbitmq.port: 15672;
- rabbitmq.amqp_port: 5672;
- rabbitmq.proto: `http`;
- rabbitmq.quorum_queues: **`true`**;
- rabbitmq.virtual_host: /;
- rabbitmq.users.root_user.name: `"<check ${rabbit_username}>"`;
- rabbitmq.users.root_user.password: `"<check ${rabbit_password}>"`.

> Внимание! Значение вышеуказанных переменных заполнены с учетом текущего примера.

Приложение A13. Установка helm.

Официальная документация helm доступна по ссылке.

Установить helm

```
helm_version="3.12.2"
curl https://get.helm.sh/helm-v${helm_version}-linux-amd64.tar.gz \
  -o helm-v${helm_version}-linux-amd64.tar.gz
tar xvfz helm-v${helm_version}-linux-amd64.tar.gz
mv linux-amd64/helm /usr/bin/
rm -r helm-v${helm_version}-linux-amd64.tar.gz linux-amd64/
```

Добавить автодополнение команд

```
helm completion bash > /etc/bash_completion.d/helm
```

Приложение А14. Список системного программного обеспечения.

Название	Версия	Роль	Лицензия
debian	11.6	операционная система	GNU GPL
containerd	1.6.6	среда для запуска контейнеров	Apache License 2.0
postgresql	12.15	субд	PostgreSQL
victoria-metrics	1.91.3	субд	Apache License 2.0
arangodb	3.11.2	субд	Apache License 2.0
redis	7.0.11	кеш сервер	3-Clause-BSD
rabbitmq	3.11.18	брокер сообщений	Mozilla Public License
clickhouse	23.3.8	субд	Apache License 2.0
docker-registry	2.8.2	репозиторий образов	Apache License 2.0
cilium	1.13.3	плагин сети	Apache License 2.0
consul	1.8.0	Хранилище конфигураций	Mozilla Public License v2.0
kubernetes	1.26.6	Оркестратор контейнеро	Apache License 2.0
nginx-ingress-controller	1.8.0	Веб балансировщик	Apache License 2.0

Установка Monq

Установка выполняется с помощью сценария поставляемого в виде контейнера, содержащего все необходимые библиотеки и зависимости.

В поставке Monq идет два сценария:

- Сценарий установки Monq;
- Сценарий очистки СПО от объектов Monq.

Подготовка к установке

1. Подготовить файл авторизации СПО `system_auth.json`;
2. Проверить, что точки подключения к компонентам СПО доступны с той машины, откуда будет запущена установка;
3. Сконфигурировать контекст monqctl:
    1. Установить monqctl версии не ниже 1.12.0, инструкция по установке;
    2. Выполнить настройку monqctl, инструкция по настройке, при этом аргументы настройки контекста: ”--registry-token”, ”--server” могут быть заданы с ложными значениями, т.к. monq и реестр микросервисов еще не запущены;
4. Импортировать контейнер `installer` необходимой версии, см. Приложение B1. Импорт образа установщика;
5. Выбрать доменное имя для установки Monq, далее будет обозначено как `${global_domain}`;
6. Обеспечить разрешение доменных имен `${global_domain}`, `api.${global_domain}`, `*.api.${global_domain}` в адрес ingress-controller, установленный в kubernetes. Указанные доменные имена должны корректно резолвиться внутри kubernetes и с машины на которой будет производиться запуск сценария;
7. Импортировать образы микросервисов в локальный container registry, см. Приложение B2. Импорт образов Monq из репозитория разработчика.

Запуск сценария установки

В примерах запуска сценариев установки используется минимальный набор переменных, с полным списком переменных можно ознакомится в соответствующем документе, см. Приложение B3. Список изменяемых переменных сценария установщика.

Вне кластера k8s

В данном случае подразумевается, что сценарий установки запущен за пределами инфраструктуры развертываемой системы, например с машины системного администратора.

Возможны следующие варианты запуска:

1. Настроен вышестоящий DNS вне kubernetes, все требуемые доменные имена (см. Приложение A6. DNS в работе Monq) корректно разрешаются в IP адреса соответствующих сервисов. В таком случае команда запуска установки будет иметь следующую конструкцию.

```
installer_version="7.11.0"
installer_files="< каталог с артефактами установщика и файлом авторизации СПО, по
    умолчанию /opt/monq >"
global_domain="< доменное имя установки >"

docker run --rm \
   --name=monq-installer-temp \
   --network=host \
   -v ${installer_files}:/opt/monq \
   installer:${installer_version} \
   ansible-playbook -e "global_domain='${global_domain}'" monq/monq.yaml
```

2. Используется DNS внутри кластера kubernetes, доменные имена разрешаются с помощью coredns внутри кластера, установка запускается вне кластера.

```
installer_version="7.11.0"
installer_files="< каталог с артефактами установщика и файлом авторизации СПО, по
    умолчанию /opt/monq >"
global_domain="< доменное имя установки >"

docker run --rm \
--add-host=postgresql.in.monq.local:<IP адрес postgresql> \
--add-host=clickhouse.in.monq.local:<IP адрес clickhouse> \
--add-host=arangodb.in.monq.local:<IP адрес arangodb> \
--add-host=victoriametrics.in.monq.local:<IP адрес victoriametrics> \
--add-host=redis.in.monq.local:<IP адрес redis> \
--add-host=rabbitmq.in.monq.local:<IP адрес rabbitmq> \
--add-host=k8s-api.in.monq.local:<IP адрес k8s-api server> \
--add-host=registry.in.monq.local:<IP адрес docker registry> \
--add-host=consul.in.monq.local:<IP адрес consul> \
--add-host=api.${global_domain}:<IP адрес веб балансировщика> \
--add-host=${global_domain}:<IP адрес веб балансировщика> \
--add-host=registry.api.${global_domain}:<IP адрес веб балансировщика> \
--name=monq-installer-temp \
--network=host \
```

```
-v ${installer_files}:/opt/monq \
installer:${installer_version} \
ansible-playbook -e "global_domain='${global_domain}'" monq/monq.yaml
```

По окончанию процесса установки можно пройти авторизацию, перейдя в систему по адресу `${global_domain}` и начинать дальнейшую настройку.

Внутри кластера k8s

Когда компоненты СПО доступны только изнутри кластера kubernetes, например при развертывании операторами. Установка должна быть запущена внутри среды, из которой есть доступ ко всем компонентам СПО. Это не совсем стандартный сценарий, его можно выполнить следующим образом:

- Импортировать образ установщика в registry. См. Приложение B1. Импорт образа установщика;
- Запустить pod с установщиком внутри кластера:

```
product_namespace="production"
registry_address="< адрес репозитория контейнеров, если используется развернутый по
    данной инструкции, то registry.in.monq.local:5000 >"
installer_version="7.11.0"

cat <<EOF | kubectl create -f -
apiVersion: v1
kind: Pod
metadata:
  name: installer
  namespace: ${product_namespace}
  labels:
    app: installer
spec:
  containers:
  - name: installer
    image: ${registry_address}/installer:${installer_version}
    command: [ "/bin/bash", "-c", "--" ]
    args: [ "trap : TERM INT; sleep infinity & wait" ]
    resources:
      limits:
        cpu: 500m
        memory: 1024Mi
      requests:
        cpu: 25m
        memory: 128Mi
  imagePullSecrets:
  - name: regsecret-monq
EOF
```

- Скопировать внутрь контейнера файл с авторизационными данными:

```
kubectl exec -n production -ti installer -- mkdir -p /opt/monq/
kubectl -n production cp /tmp/system_auth.json installer:/opt/monq/
```

- Запустить установку:

```
global_domain="доменное< имя установки>"

kubectl exec -n production -ti installer -- ansible-playbook \
  -e "global_domain='${global_domain}'" monq/monq.yaml
```

- Посмотреть ошибки. Первым этапом установки выполняется проверка доступности инфраструктурных объектов, если что-то не доступно сценарий остановит свое выполнение, и в выделенный файл будет записана информация для отладки. Следующей командой можно посмотреть файл на предмет ошибок:

```
kubectl exec -n production -ti installer -- cat /opt/monq/errors.json
```

- Скопировать артефакты. После прохождения установки будет создан набор артефактов, файл настройки подключения, сертификаты, желательно эти файлы скопировать себе и сохранить. Скопировать можно следующей командой:

```
kubectl -n production cp installer:/opt/monq /tmp/monq
```

- После окончания установки удалить pod:

```
kubectl delete po -n production installer
```

После установки можно пройти авторизацию, перейдя в систему по адресу `${global_domain}`, и начинать дальнейшую настройку.

Запуск сценария удаления

В случае, если установка не была завершена успешно, следует использовать сценарий eraser, с помощью него очищаются все компоненты СПО от объектов monq. Данный сценарий использует аналогичный набор переменных и запускается следующим образом:

> Внимание! Удаление должно запускаться с теми же переменными, что и установка, только переменную `global_domain` можно исключить.

Команда на запуск сценария удаления (eraser), аналогичная команде установки, меняется только название сценария: `monq/monq.yaml` -> `monq/eraser.yaml`

Заключение

В данном примере рассмотрены основные способы запуска сценария установки и удаления monq.

Приложение В1. Импорт образа установщика.

Импорт образа из репозитория разработчика

Скачивание релиза с образом установщика:

```
installer_version="7.11.0"
token="< токен обновления полученный с сайта. выписывается вместе с лицензией >"
destination_dir="< целевой каталог, в который будет скачан релиз >"

monqctl installer version download ${installer_version} --token=${token} \
  --dest=${destination_dir}
```

Распаковка в локальное хранилище:

- Если используется docker

  ```
  docker load -i ${destination_dir}/installer*.img
  ```

- Если используется ctr

  ```
  ctr -n=k8s.io image import ${destination_dir}/installer*.img
  ```

Экспорт образа в личный репозиторий контейнеров

> Информация! Обязательно выполнить при запуске установки из kubernetes.

Экспорт образа в репозиторий контейнеров:

- Если используется docker

  ```
  installer_version="7.11.0"
  registry_address="< адрес репозитория контейнеров, если используется развернутый по
      данной инструкции, то registry.in.monq.local:5000 >"

  docker load -i ${destination_dir}/installer*.img
  docker tag installer:${installer_version} \
    ${registry_address}/installer:${installer_version}
  docker push ${registry_address}/installer:${installer_version}
  ```

- Если используется ctr

```
installer_version="7.11.0"
registry_address="< адрес репозитория контейнеров, если используется развернутый по
    данной инструкции, то registry.in.monq.local:5000 >"

ctr -n=k8s.io image import ${destination_dir}/installer*.img
ctr -n=k8s.io image tag \
  docker.io/library/installer:${installer_version} \
  ${registry_address}/installer:${installer_version}

# аргумент --plain-http задается при использовании insecure registry
ctr -n=k8s.io image push --plain-http \
  ${registry_address}/installer:${installer_version}
```

Приложение В2. Импорт образов Monq из репозитория разработчика.

Нужно выполнить импорт для следующих продуктов:

- продукт: `monq-registry`, версия: `3.13.0`;
- продукт: `monq`, версия: `7.11.0`.

> Внимание! Импорт должен быть выполнен для двух продуктов, `monq-registry` и `monq` соответственно.

Порядок действий для импорта образов в приватный репозиторий из репозитория разработчика.

С доступом в интернет:

1. Установить контекст релиза необходимой версии и продукта:

```
product="< название продукта >"
release_version="< версия продукта >"

monqctl release use-version ${release_version} --product=${product}
```

2. Выполнить команду импорта образов контейнеров в приватный репозиторий:

```
registry_address="< адрес репозитория контейнеров, если используется развернутый по
    данной инструкции, то http://registry.in.monq.local:5000 >"
monqctl release import-images --registryUri ${registry_address} --registryAuth=None
```

   Обязательные параметры:

   - ”--registryAuth” задает тип авторизации в приватном registry;
   - ”--registryUri” задает адрес приватного registry для импорта контейнеров.

   Опциональные параметры:

   - ”–-registryUsername”, ”–registryPassword” задают имя и пароль для basic авторизации в приватном registry, должны использоваться вместе;

- ”–-registryLocation” задает location в private registry, например если задать в значение monq, то в дальнейшем image будет доступен по адресу http://registry.in.monq.local:5000/monq/<monq_microservice>:.

Без доступа в интернет:

1. Сконфигурировать контекст monqctl:
    1. Установить monqctl версии не ниже 1.12.0, инструкция по установке;
    2. Выполнить настройку monqctl, инструкция по настройке, при этом аргументы настройки контекста: ”--registry-token”, ”--server” могут быть заданы с ложными значениями, т.к. monq и реестр микросервисов еще не запущены.
2. Выполнить экспорт релиза в каталог на диске:

```
product="< название продукта >"
release_version="< версия продукта >"
storage_path="< путь к каталогу >"

mkdir -p ${storage_path}
monqctl release version export ${release_version} --product=${product} \
 --dest=${storage_path} --full
```

3. Переместить каталог с образами на машину с которой есть доступ в личный репозиторий контейнеров;
4. Сконфигурировать контекст monqctl, пример см выше;
5. Установить контекст релиза необходимой версии и продукта, при этом в качестве источника указать каталог с образами:

```
product="< название продукта >"
release_version="< версия продукта >"
storage_path="< путь к каталогу >"

monqctl release use-version ${release_version} --product=${product} \
 --sourceDir=${storage_path}
```

6. Выполнить команду импорта образов контейнеров в приватный репозиторий:

```
registry_address="< адрес репозитория контейнеров, если используется развернутый по
    данной инструкции, то http://registry.in.monq.local:5000 >"
monqctl release import-images --registryUri ${registry_address} --registryAuth=None
```

Обязательные параметры:

- ”--registryAuth” задает тип авторизации в приватном registry;
- ”--registryUri” задает адрес приватного registry для импорта контейнеров.

Опциональные параметры:

- ”–-registryUsername”, ”–registryPassword” задают имя и пароль для basic авторизации в приватном registry, должны использоваться вместе;
- ”–-registryLocation” задает location в private registry, например если задать в значение `monq`, то в дальнейшем image будет доступен по адресу http://registry.in.monq.local:5000/monq/<monq_microservice>:.

Приложение В3. Список изменяемых переменных сценария установщика.

В большинстве случаев изменяется только переменная `global_domain`, но могут возникнуть ситуации, когда требуется дополнительная настройка установки.

- global_domain - доменное имя установки, обязательная к назначению;
- modules - список модулей для установки, по умолчанию: `registry,pl,cl,fm,sm,mcs,plugins`;
- prechecks - флаг запуска предварительных проверок, по умолчанию: **`true`**;
- monq_namespace - kubernetes namespace в который будет произведена установка Monq, по умолчанию: `production`;
- pvc_name - имя pvc в namespace {{ monq_namespace }}, который будет использовать микросервисы Monq, по умолчанию: `pvc-monq`;
- files_dir - каталог с генерируемыми файлами, по умолчанию: `/opt/monq`;
- system_auth_file - имя файла в папке {{ files_dir }} с которого будут считаны учетные данные для подключения к сервисам, по умолчанию: `system_auth.json`;
- node_selector - node selector для запуска deployment Monq, данная переменная задается отдельно от остальных переменных дополнительным аргументом `-e`, и обязательно в двойных кавычках, по умолчанию: `"{\"node_selector\":{\"foo\":\"bar\"}}"`
- ingress_class - ingress class, по умолчанию: `nginx`;
- defaultLocale - локаль по умолчанию, по умолчанию: `en-US`;
- admin_password - пароль пользователя admin@{{ global_domain }} в Monq, по умолчанию: `monq_admin`;
- create_registry_secret - флаг создания секрета .dockerconfig в namespace {{ monq_namespace }}, по умолчанию: **`true`**;
- registry_secret_name - имя секрета .dockerconfig в namespace {{ monq_namespace }}, по умолчанию: `regsecret-monq`;
- generate_self_signed_cert - флаг генерации само подписных сертификатов для Monq, по умолчанию: **`true`**;
- create_management_users - флаг создания management пользователей, по умолчанию: **`true`**.