Инструкция по установке Monq. Версия 8.0.0.

MDL

2024-11-21

Содержание

Введение

Цель данной инструкции предоставить пользователю возможность самостоятельно выполнить установку Monq включая все компоненты инфраструктуры в требуемой конфигурации.

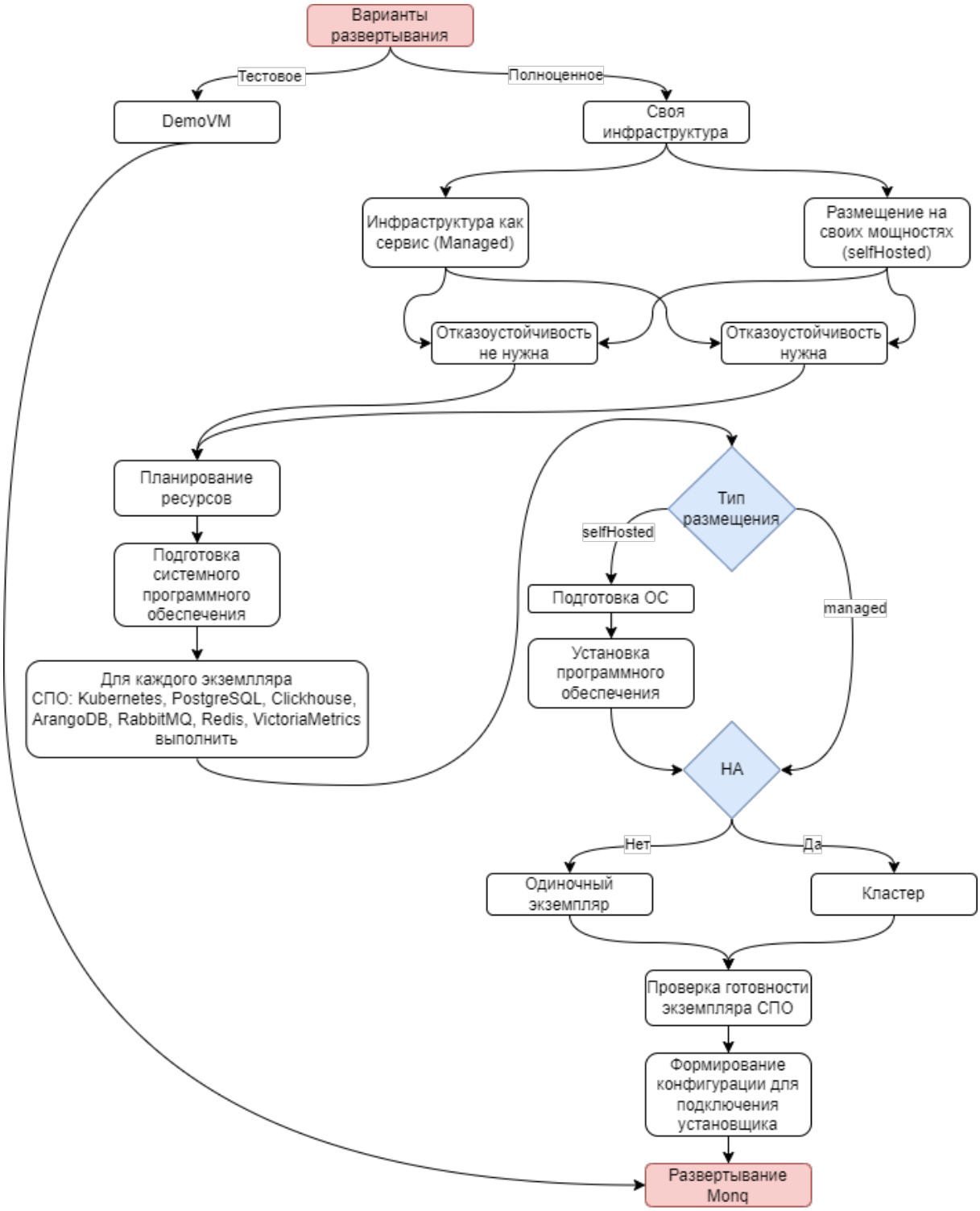
Инструкция описывает предполагаемый путь по развертыванию системы и носит рекомендательный характер, каждая установка уникальна по своему, частные утвержденные особенности операционной системы, отдельные наборы прав и ограничений.

Документ предполагает развертывания системы на базовых образах операционных систем без дополнительных изменений.

Для администрирования Monq и работы с данным документом пользователю нужно иметь опыт по работе с:

- инструментарием запуска контейнеров: containerd, docker
- оркестратором: kubernetes, в частности понимать как работать с логами
- субд: postgresql, clickhouse, redis, arangodb, victoriametrics
- брокером сообщений: rabbitmq
- кеш сервером: redis
- веб сервером: nginx
- сертификатами ssl

Схематический план развертывания Monq



Проектирование решения

Перед тем как начать проектирование ресурсов для Monq необходимо ознакомиться с общими требованиями и рекомендациями для запуска системы.

Список необходимых компонентов:

Название	Версия
postgresql	12.15
victoria-metrics	1.91.3
arangodb	3.11.2
redis	7.0.11
rabbitmq	3.11.18
clickhouse	23.3.8
cilium	1.13.3
consul	1.8.0
kubernetes	1.26.15
nginx-ingress-controller	1.8.0
container registry*	-
CSI совместимый storage (с поддержкой RWX)**	-

> * container registry должен быть совместим образами в docker формате
> ** Требуется RWX(readWriteMany) хранилище, для подключения данного хранилища в несколько сервисов (реплик). Список совместимых решений можно посмотреть в официальной документации.

Рекомендации:

- Для построения дисковой системы рекомендуется использовать SSD диски.

- Весь объем диска должен быть смонтирован в `/`.
- Использование файловых систем `ext4` или `xfs`.
- Не требуется установка Desktop Environment.
- Для запуска arangodb требуется процессор старше 2011 года выпуска(требуется поддержка процессором. 256 bit avx инструкций: ссылка на требования).

Данное руководство не учитывает расход ресурсов операционной системы и дополнительных служб(агентов мониторинга, резервного копирования и т.д.). Для наиболее точного планирования ресурсов необходимо понимание нагрузки и требований к отказоустойчивости выбранного решения.

Далее в разделе будут представлены наиболее распространенные конфигурации установки системы. Указанные системные требования рассчитаны на основе пользовательского опыта и могут отличаться в зависимости от характера или интенсивности нагрузки.

Типовые схемы размещения компонентов

Demo образ

Одна Виртуальная машина (далее ВМ) с предустановленной инфраструктурой. Все микросервисы запущены в единственном экземпляре, системное программное обеспечение сконфигурировано на минимальное использование ресурсов.

> Внимание! Образ ВМ предоставляется в ознакомительных целях, для возможности наиболее быстрой установки системы. Данное решение не поддерживает горизонтальное масштабирование.

Минимальные ресурсы:

VM	CPU	RAM	Disk
Demo-VM	8	24	60

Для выполнения установки необходимо скачать образ ВМ с официального сайта в формате OVF по ссылке https://monq.ru/download, импортировать в среду виртуализации и установить Monq согласно инструкции.

Минимальная конфигурация

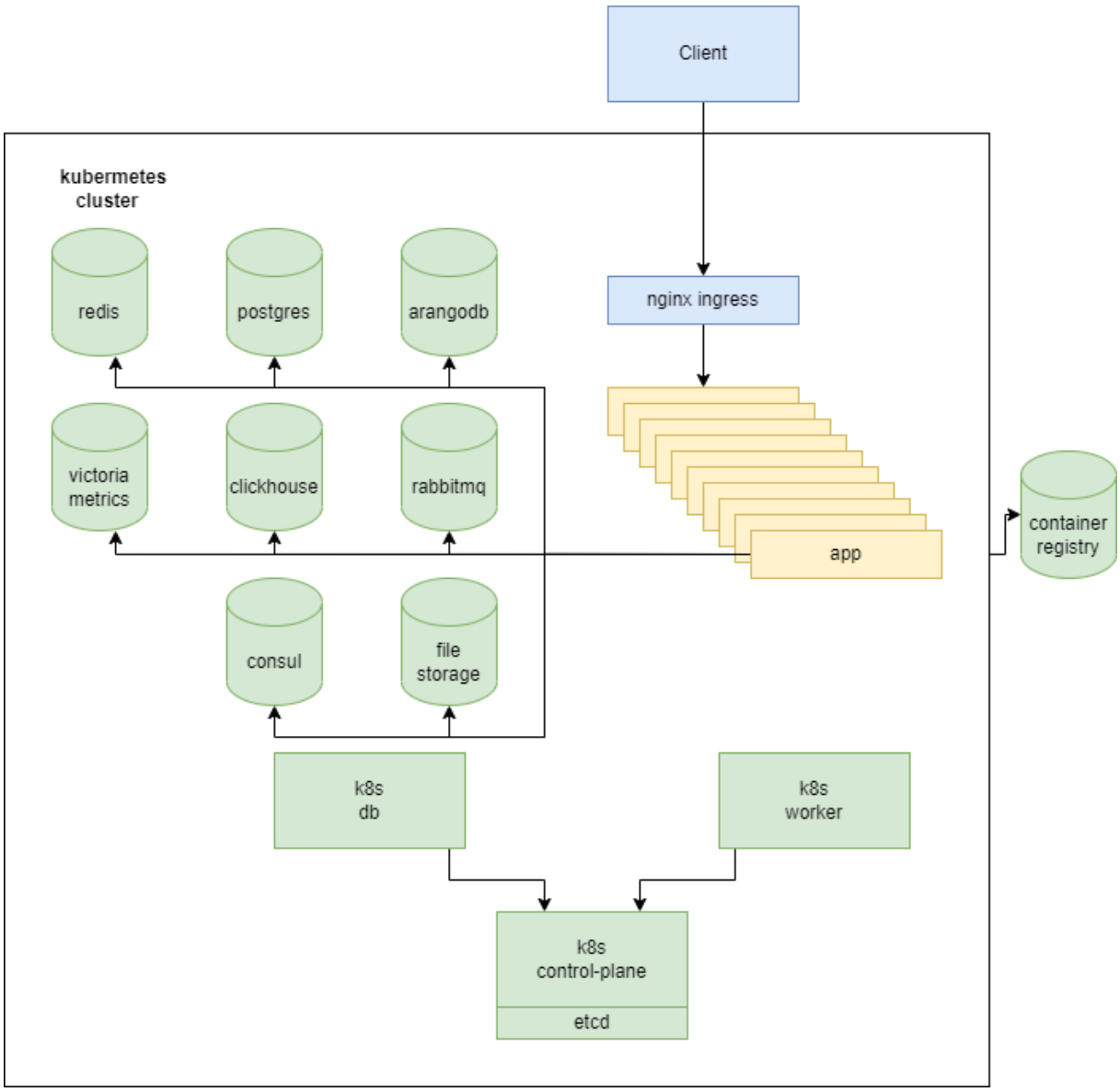
Минимально три сервера(ВМ), ручная подготовка инфраструктуры. Данная конфигурация позволит горизонтально масштабировать инфраструктуру при увеличении нагрузки путем добавления новых нод в kubernetes и перераспределения нагрузки, но не обеспечит отказоустойчивость решения. Все машины являются нодами kubernetes, СУБД запущены с помощью statefulSet.

Минимальные ресурсы:

Сервер(VM)	Hostname	CPU	RAM	Disk
Kubernetes control-plane	master	2	4	30
Kubernetes worker	worker	8	24	80
Kubernetes database	db	8	14	200

> Внимание! Hostname серверов указаны для примера, и будут использованы далее в пошаговой инструкции. Ограничений на Hostname нет, но изменения следует учитывать при выполнении команд.

Схема расположения компонентов представлена на рисунке:



Отказоустойчивая конфигурация

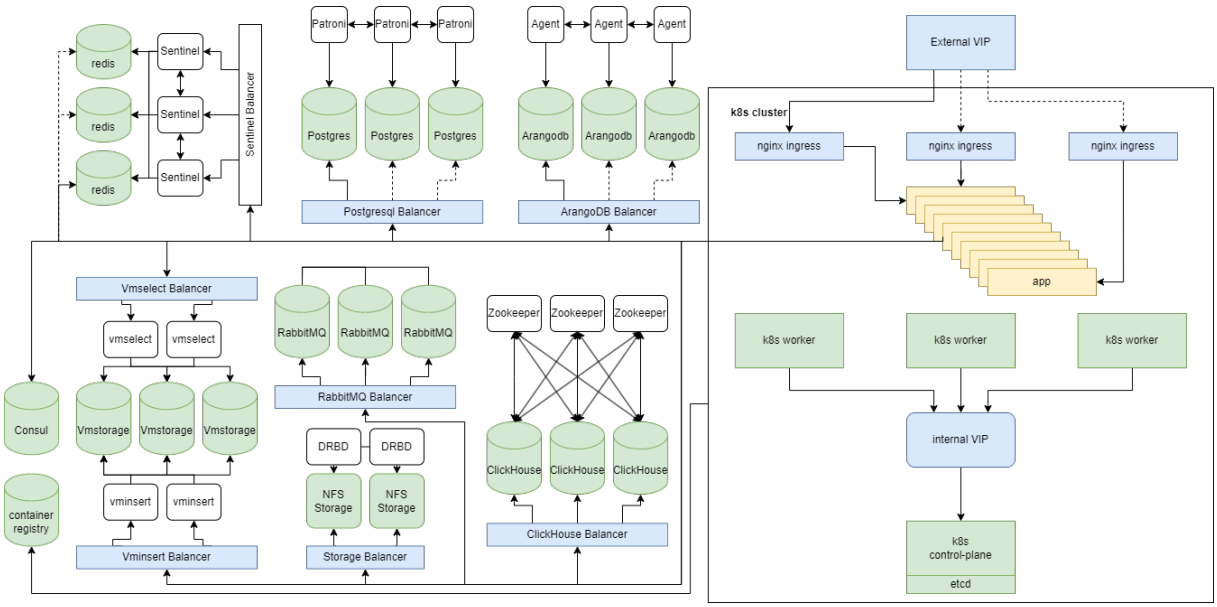
Все компоненты СПО запускаются в режиме отказоустойчивом режиме (далее HA), имеется возможность масштабирования количества микросервисов, участвующих в обработке данных. СУБД запущены в kubernetes с помощью операторов.

VM	Hostname	CPU	RAM	Disk
Kubernetes control-plane	master-1	2	4	30

VM	Hostname	CPU	RAM	Disk
Kubernetes control-plane	master-2	2	4	30
Kubernetes control-plane	master-3	2	4	30
Kubernetes worker	worker-1	12	24	120
Kubernetes worker	worker-2	12	24	120
Kubernetes worker	worker-3	12	24	120
ArangoDB	arangodb-1	8	32	60
ArangoDB	arangodb-2	8	32	60
ArangoDB	arangodb-3	8	32	60
Clickhouse	clickhouse-1	8	24	400
Clickhouse	clickhouse-2	8	24	400
Clickhouse	clickhouse-3	8	24	400
Postgresql	postgresql-1	6	16	200
Postgresql	postgresql-2	6	16	200
Postgresql	postgresql-3	6	16	200
RabbitMQ	rabbitmq-1	4	6	60
RabbitMQ	rabbitmq-2	4	6	60
RabbitMQ	rabbitmq-3	4	6	60
Redis	redis-1	4	6	40
Redis	redis-2	4	6	40
Redis	redis-3	4	6	40
VictoriaMetrics	vm-1	8	16	300
VictoriaMetrics	vm-2	8	16	300
VictoriaMetrics	vm-3	8	16	300

> Внимание! Hostname серверов указаны для примера, и будут использованы далее в пошаговой инструкции. Ограничений на Hostname нет, но изменения следует учитывать при выполнении команд.

Схема расположения компонентов представлена на рисунке:



Использование Managed сервисов

Приведенные выше решения подразумевают ручную установку СПО совместно с monq. Со стороны Monq нет явных ограничений, в каком виде они будут развернуты - это может быть установка ”на железе”, в виде отдельного контейнера или внутри кластера Kubernetes с помощью оператора. Главное: чтобы все точки подключения к компонентам СПО были доступны из кластера. Возможно использование Managed сервисов (корпоративных или предоставляемых облачными провайдерами) для следующих компонентов СПО:

- 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;
- VictoriaMetrics
  - возможность чтения/записи в БД.

Перед выбором managed сервиса для использования следует удостовериться в соответствии вышеуказанным требованиям.

Отказоустойчивость и балансировка нагрузки

В текущей реализации Monq нет встроенной поддержки работы с СПО в режимах балансировки нагрузки. Для каждого из компонентов определение работоспособности и балансировка запросов к работоспособному экземпляру осуществляется сторонними средствами (например HA-Proxy или service Kubernetes), за исключением Redis, для него реализован механизм поддерживающий работу с sentinel, который отвечает за определение мастера.

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.

Пример запуска Monq с СПО в режиме HA будет рассмотрен более подробно в приложениях к 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.

Заключение

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

- тип инсталляции: пробная, минимальная конфигурация, отказоустойчивая конфигурация или спроектированная самостоятельно в зависимости от требований;
- Self-Hosted или Managed;
- HA или Standalone.

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

Подготовка системного программного обеспечения

В соответствии с проектированием решения и планированием ресурсов потребуется подготовить итоговый стенд для развертывания. В данном разделе будет представлена пошаговая инструкция по подготовке минимального контура для запуска monq, которая подразумевает использование следующего набора ПО:

Название	Версия	Роль	Лицензия
debian	11.7	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
cilium	1.13.3	Плагин сети	Apache License 2.0
consul	1.8.0	Хранилище конфигураций	Mozilla Public License v2.0
kubernetes	1.26.9	Оркестратор контейнеров	Apache License 2.0
nginx-ingress-controller	1.8.0	Веб балансировщик	Apache License 2.0
registry	2.8.3	Репозиторий контейнеров	Apache License 2.0

Для запуска на дистрибутивах отличных от Debian 11.7 потребуется адаптировать команды приведенные в пошаговой инструкции.

По тексту будут ссылки на работу с файлом authfile, в целях исключения многократного

повторения ссылка размещена в начале документа: см. Приложение. Инструкция по наполнению authfile.

При использовании managed сервисов следует заполнить данные authfile и пропустить блоки инструкции описывающие установку данного сервиса.

Предпочтительный способ обращения микросервисов Monq к компонентам СПО с использованием доменных имен, без указания IP адресов. Перед началом работ следует выбрать доменные имена для компонентов СПО. В пошаговой инструкции будет использована зона `in.monq.local`. Каждому компоненту СПО присваивается отдельное доменное имя, например `postgresql.in.monq.local`. Рекомендуется использование внутреннего корпоративного DNS сервера для разрешения доменных имен компонентов СПО, перед тем как приступить к установке необходимо добавить соответствующие записи:

№	Сервис, назначение	Доменное имя по умолчанию	Расположение
1	Репозиторий контейнеров	registry.in.monq.local	db
2	СУБД postgresql*	postgresql.in.monq.local	db
3	СУБД clickhouse*	clickhouse.in.monq.local	db
4	СУБД arangodb*	arangodb.in.monq.local	db
5.a	СУБД victoriametrics single	victoriametrics.in.monq.local	db
5.b	СУБД victoriametrics, кластер	vminsert.in.monq.local	db
		vmselect.in.monq.local	db
6	Кеш сервер, Redis**	redis.in.monq.local	db
7	Брокер сообщений, Rabbitmq*	rabbitmq.in.monq.local	db
8	Адрес API сервера k8s*	k8s-api.in.monq.local	master
9	Адрес хранилища конфигураций	consul.in.monq.local	db
10	Доменные имена установки monq,	<доменное имя>	worker
	разрешаются в адрес ingress контроллера,	api.<доменное имя>	worker
	при HA в Loadbalancer	registry.api.<доменное имя>	worker

Таблица также отражает расположение компонентов по серверам, которое будет испрользовано в руководстве по установке.

* в случае использования HA, указать адрес балансировщика запросов;

** в случае использования HA, указать адрес sentinel.

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

В случае если нет возможности использовать корпоративный DNS, необходимо добавить вышеуказанные записи в hosts на каждом хосте до начала установки и выполнить настройку разрешения доменных имен на кластерном coredns в составе kubernetes после его инициализации, см. Приложение. Кластерный DNS.

В пошаговой инструкции отражен способ запуска СУБД с помощью StatefulSet Kubernetes. Для хранения данных используются hostPath volumes. Данное решение не рекомендовано к запуску в промышленной эксплутации. Вместо этого необходимо использовать CSI совместимый RWO storage. Список совместимых решений можно посмотреть в официальной документации.

Список переменных, используемых по тексту:

- `${infra_domain}` - dns зона для инфраструктурных компонентов
- `${global_domain}` - основное доменное имя развертываемого приложения
- `${registry_address}` - адрес репозитория контейнеров
- `${infra_namespace}` - kubernetes namespace для запуска инфраструктурных компонентов
- `${monq_namespace}` - kubernetes namespace для запуска monq

Инструкция составлена с учетом выполнения команд от пользователя `root`, поэтому перед выполнением команд необходимо перейти на использование root и установить переменные:

```
su -
infra_domain=in.monq.local
registry_address=registry.${infra_domain}:5000
infra_namespace=infra
monq_namespace=production
```

Платформа kubernetes

Установка ОС

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

Необходимо выполнить установку ос Debian 11.7 на все сервера/VM. Весь объем диска должен быть смонтирован в `/`, файловая система `ext4` или `xfs`. Раздел подкачки(swap) можно не

создавать, тк он не будет использован. На этапе выбора компонентов для установки следует выбрать стандартные системные утилиты и ssh сервер.

Выпуск CA сертификата

В данном разделе представлено руководство по выпуску самоподписного CA сертификата, который будет использован для шифрования соединений внутри системы.

Если планируется использовать существующие, заранее выпущенные, самоподписанные сертификаты, то см Приложение. Собственные сертификаты SSL.

Данным CA будет подписан сертификат для доменного имени устанавливаемой системы, а так же можно будет подписать сертификаты, выпускаемые для организации защищенного соединения между компонентами Monq и СПО, для этого из сертификатов надо создать секрет и указать реквизиты секрета в запуске соответствующего чарта СПО, см. Приложение. Описание типового helm chart. Актуально для standalone развертывания, для организации взаимодействия по SSL при развертывании в режиме HA необходимо обратиться к документации разработчика соответствующего решения.

> Внимание! Команды выполняются на сервере master.

Сформировать CA сертификат:

```
mkdir certs
openssl_conf=$(dpkg -L openssl | grep openssl.cnf | head -n 1)

openssl req -new -nodes -out certs/monq.ca.csr \
  -keyout certs/monq.ca.key -subj "/CN=monq"

openssl x509 -req -in certs/monq.ca.csr -days 3650 \
  -extfile ${openssl_conf} -extensions v3_ca \
  -signkey certs/monq.ca.key -out certs/monq.ca.crt
```

Добавить сертификат в список доверенных:

> Внимание! Данный шаг нужно выполнить на всех серверах, предварительно скопировав файл `certs/monq.ca.crt`

```
mkdir /usr/share/ca-certificates/monq
cp certs/monq.ca.crt /usr/share/ca-certificates/monq/monq.ca.crt
echo "monq/monq.ca.crt" >> /etc/ca-certificates.conf
update-ca-certificates
```

Выписать сертификат для registry:

```
mkdir certs/docker-registry
openssl req -new -nodes -out certs/docker-registry/registry.csr \
  -keyout certs/docker-registry/registry.key -subj "/CN=registry.${infra_domain}"
```

```
openssl x509 -req -in certs/docker-registry/registry.csr -days 3650 \
  -extfile <(printf "subjectAltName=DNS:registry.${infra_domain}") \
  -CA certs/monq.ca.crt -CAkey certs/monq.ca.key -CAcreateserial \
  -out certs/docker-registry/registry.crt
```

Файлы `certs/docker-registry/registry.key` и `certs/docker-registry/registry.crt` необходимо перенести на сервер db, для последующей настройки container registry.

Установка пакетов

> Внимание! Данный раздел написал для online установки. В случае если сервера не имеют доступа в интернет, необходимо выполнить заменить выполнение команд в данном разделе командами из Приложение. Установка пакетов Offline.

> Внимание! Команды из данного раздела необходимо выполнить на всех серверах!

Выполнить установку пакетов:

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

Выполнить установку 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
```

Выполнить установку kubernetes:

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | \
  gpg --dearmor -o /usr/share/keyrings/kubernetes-apt-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/kubernetes-apt-keyring.gpg] \
  https://pkgs.k8s.io/core:/stable:/v1.26/deb/ /" \
  | tee /etc/apt/sources.list.d/kubernetes.list

apt update
apt install -y kubelet=1.26.15-1.1 kubeadm=1.26.15-1.1 kubectl=1.26.15-1.1
apt-mark hold kubelet kubeadm kubectl
```

Выполнить установку helm:

```
helm_version="3.12.2"
wget https://get.helm.sh/helm-v${helm_version}-linux-amd64.tar.gz
tar -xf helm-v${helm_version}-linux-amd64.tar.gz linux-amd64/helm
mv linux-amd64/helm /usr/local/bin/
rm -r helm-v${helm_version}-linux-amd64.tar.gz linux-amd64/
```

Выполнить установку crane:

```
crane_version="0.16.1"
wget "https://github.com/google/go-containerregistry/releases/download\
/v${crane_version}/go-containerregistry_Linux_x86_64.tar.gz"
tar -xf go-containerregistry_Linux_x86_64.tar.gz -C /usr/local/bin/ crane
rm go-containerregistry_Linux_x86_64.tar.gz
```

Установка Container registry

Перед началом работ необходимо создать на внешнем DNS сервере запись для хоста registry вида `registry.${infra_domain}`, запись должна разрешаться в IP сервера на котором расположен container registry (в данном примере сервер db).

> Внимание! Команды выполняются на сервере db.

Выполнить установку container registry:

```
registry_version="2.8.3"
wget "https://github.com/distribution/distribution\
/releases/download/v${registry_version}/registry_${registry_version}_linux_amd64.tar.gz"
tar -xf registry_${registry_version}_linux_amd64.tar.gz -C /usr/local/bin/ registry
rm registry_${registry_version}_linux_amd64.tar.gz
```

Создать пользователя для registry:

```
useradd --no-create-home --shell /bin/false registry
```

Создать каталоги для работы приложения:

```
mkdir -p /storage/registry
chown -R registry /storage/registry
mkdir -p /etc/docker/registry
```

Сформировать конфигурационный файл:

```
cat <<EOF | tee /etc/docker/registry/config.yaml
version: 0.1
log:
  fields:
    service: registry
storage:
  cache:
    blobdescriptor: inmemory
  filesystem:
    rootdirectory: /storage/registry
http:
```

```
  addr: :5000
  tls:
    certificate: /etc/docker/registry/registry.crt
    key: /etc/docker/registry/registry.key
  headers:
    X-Content-Type-Options: [nosniff]
health:
  storagedriver:
    enabled: true
    interval: 10s
    threshold: 3
EOF
```

Создать unit для systemd:

```
cat <<EOF | tee /etc/systemd/system/registry.service
[Unit]
Description=docker private registry service

[Service]
ExecStart=/usr/local/bin/registry serve /etc/docker/registry/config.yaml
Restart=always
Type=simple
RestartSec=10s
User=registry

[Install]
WantedBy=multi-user.target
EOF
```

Скопировать сертификат registry выпущенный ранее на шаге Выпуск CA сертификата:

```
cp certs/docker-registry/registry.key certs/docker-registry/registry.crt /etc/docker/registry
chown -R registry /etc/docker/registry
```

Выполнить запуск container registry:

```
systemctl start registry
systemctl enable registry
```

Для наполнения authfile выставить значения:

- registry.host: `"registry.in.monq.local"`
- registry.port: `5000`
- registry.proto: `"https"`
- registry.auth_type: `"None"`

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

Установка сервера NFS

> Внимание! Команды выполняются на сервере db.

Выполнить установку nfs-server:

```
apt install -y nfs-kernel-server
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
```

Создание каталогов для хранения данных СУБД

> Внимание! Команды выполняются на сервере db, согласно таблице размещения компонентов.

Создать каталоги для хранения данных компонентов СПО:

```
mkdir -p /storage/consul
mkdir -p /storage/arangodb
mkdir -p /storage/clickhouse
mkdir -p /storage/postgresql
mkdir -p /storage/rabbitmq
mkdir -p /storage/redis
mkdir -p /storage/victoriametrics
```

Импорт контейнеров СПО

> Внимание! Команды выполняются на сервере где был установлен crane, обычно это master.

Выполнить импорт образов контейнеров платформы kubernetes в container registry:

```
crane copy registry.k8s.io/coredns/coredns:v1.9.3 ${registry_address}/coredns:v1.9.3
crane copy registry.k8s.io/kube-proxy:v1.26.15 ${registry_address}/kube-proxy:v1.26.15
crane copy registry.k8s.io/pause:3.6 ${registry_address}/pause:3.6
crane copy registry.k8s.io/pause:3.9 ${registry_address}/pause:3.9
crane copy registry.k8s.io/kube-apiserver:v1.26.15 \
  ${registry_address}/kube-apiserver:v1.26.15
crane copy registry.k8s.io/kube-controller-manager:v1.26.15 \
  ${registry_address}/kube-controller-manager:v1.26.15
crane copy registry.k8s.io/kube-scheduler:v1.26.15 \
  ${registry_address}/kube-scheduler:v1.26.15
crane copy registry.k8s.io/etcd:3.5.10-0 ${registry_address}/etcd:3.5.10-0
crane copy quay.io/cilium/cilium:v1.13.3 ${registry_address}/cilium/cilium:v1.13.3
crane copy quay.io/cilium/operator-generic:v1.13.3 \
  ${registry_address}/cilium/operator-generic:v1.13.3
crane copy registry.k8s.io/ingress-nginx/controller:v1.8.0 \
  ${registry_address}/ingress-nginx/controller:v1.8.0
crane copy registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20230407 \
  ${registry_address}/ingress-nginx/kube-webhook-certgen:v20230407
crane copy consul:1.8.0 ${registry_address}/consul:1.8.0
```

Выполнить импорт образов контейнеров используемого СПО в container registry:

```
crane copy arangodb:3.11.2 ${registry_address}/arangodb:3.11.2
crane copy clickhouse/clickhouse-server:23.3.8 \
  ${registry_address}/clickhouse/clickhouse-server:23.3.8
crane copy postgres:12.15 ${registry_address}/postgres:12.15
crane copy rabbitmq:3.11.18-management ${registry_address}/rabbitmq:3.11.18-management
crane copy redis:7.0.11 ${registry_address}/redis:7.0.11
crane copy victoriametrics/victoria-metrics:v1.91.3 \
  ${registry_address}/victoriametrics/victoria-metrics:v1.91.3
```

В случае если планируется HA инсталляция, импортировать дополнительные образы:

```
crane copy ghcr.io/kube-vip/kube-vip:v0.5.0 ${registry_address}/kube-vip/kube-vip:v0.5.0
crane copy registry.opensource.zalan.do/acid/postgres-operator:v1.10.1 \
  ${registry_address}/acid/postgres-operator:v1.10.1
crane copy ghcr.io/zalando/spilo-15:3.0-p1 ${registry_address}/zalando/spilo-15:3.0-p1
crane copy altinity/clickhouse-operator:0.21.3 \
  ${registry_address}/altinity/clickhouse-operator:0.21.3
crane copy pravega/zookeeper:0.2.15 ${registry_address}/pravega/zookeeper:0.2.15
crane copy pravega/zookeeper-operator:0.2.15 \
  ${registry_address}/pravega/zookeeper-operator:0.2.15
crane copy lachlanevenson/k8s-kubectl:v1.23.2 \
  ${registry_address}/lachlanevenson/k8s-kubectl:v1.23.2
crane copy arangodb/kube-arangodb:1.2.32 ${registry_address}/arangodb/kube-arangodb:1.2.32
crane copy alpine:3.11 ${registry_address}/alpine:3.11
crane copy victoriametrics/operator:v0.34.1 \
  ${registry_address}/victoriametrics/operator:v0.34.1
crane copy victoriametrics/vminsert:v1.91.3-cluster \
  ${registry_address}/victoriametrics/vminsert:v1.91.3-cluster
crane copy victoriametrics/vmselect:v1.91.3-cluster \
  ${registry_address}/victoriametrics/vmselect:v1.91.3-cluster
crane copy victoriametrics/vmstorage:v1.91.3-cluster \
  ${registry_address}/victoriametrics/vmstorage:v1.91.3-cluster
crane copy quay.io/spotahome/redis-operator:v1.2.4 \
  ${registry_address}/spotahome/redis-operator:v1.2.4
crane copy rabbitmqoperator/cluster-operator:2.3.0 \
  ${registry_address}/rabbitmqoperator/cluster-operator:2.3.0
```

Загрузка helm chart

> Внимание! Команды выполняются на сервере master.

Скачать чарты:

```
mkdir -p offline/charts/

helm repo add cilium https://helm.cilium.io/
helm pull cilium/cilium --version 1.13.3 -d offline/charts/

helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm pull ingress-nginx/ingress-nginx --version 4.7.0 -d offline/charts/

helm repo add monq https://helm.monq.ru/charts

helm pull monq/postgresql --version 1.0.1 -d offline/charts/
helm pull monq/clickhouse --version 1.0.1 -d offline/charts/
helm pull monq/arangodb --version 1.0.1 -d offline/charts/
helm pull monq/victoriametrics --version 1.0.1 -d offline/charts/
helm pull monq/redis --version 1.0.1 -d offline/charts/
```

```
helm pull monq/rabbitmq --version 1.0.1 -d offline/charts/
helm pull monq/consul --version 1.0.1 -d offline/charts/
```

В случае если планируется HA инсталляция, скачать дополнительные чарты:

```
helm repo add postgres-operator-charts \
  https://opensource.zalando.com/postgres-operator/charts/postgres-operator
helm pull postgres-operator-charts/postgres-operator --version 1.10.1 -d offline/charts/

helm repo add clickhouse-operator https://docs.altinity.com/clickhouse-operator/
helm pull clickhouse-operator/altinity-clickhouse-operator \
  --version 0.21.3 -d offline/charts/
helm repo add pravega https://charts.pravega.io
helm pull pravega/zookeeper-operator --version 0.2.15 -d offline/charts/

url="https://github.com/arangodb/kube-arangodb/releases/download/1.2.32"
helm pull ${url}/kube-arangodb-1.2.32.tgz -d offline/charts/

helm repo add vm https://victoriametrics.github.io/helm-charts/
helm pull vm/victoria-metrics-operator --version 0.23.1 -d offline/charts/

helm repo add redis-operator https://spotahome.github.io/redis-operator
helm pull redis-operator/redis-operator --version 3.2.8 -d offline/charts/
crane pull quay.io/spotahome/redis-operator:v1.2.4 ./offline/images/redis-operator.img

wget "https://github.com/rabbitmq/cluster-operator\
/releases/download/v2.3.0/cluster-operator.yml" -P offline/charts
```

Установка monqctl и импорт контейнеров ППО

> Внимание! Команды выполняются на сервере master.

Выполнить установку monqctl:

```
wget https://downloads.monq.ru/tools/monqctl/v1.13.0/linux-x64/monqctl.zip
unzip monqctl.zip
mv monqctl /usr/local/bin
rm monqctl.zip
```

Выполнить временную конфигурацию контекста monqctl:

```
token="< токен обновления полученный с сайта. выписывается вместе с лицензией >"
monqctl config set instance temp --server=http://registry.api.monq.local
monqctl config set credential temp --registry-token=000
monqctl config set releasehub monq-release-hub --token=${token}
monqctl config set context temp --instance=temp --credential=temp \
  --releasehub=monq-release-hub
monqctl config use-context temp
```

Выполнить импорт образов контейнеров Monq в container registry:

```
monqctl release use-version 8.0.0 --product=installer
monqctl release import-images --registryUri ${registry_address} --registryAuth=None
monqctl release use-version 3.13.8 --product=monq-registry
monqctl release import-images --registryUri ${registry_address} --registryAuth=None
monqctl release use-version 8.0.0 --product=monq
monqctl release import-images --registryUri ${registry_address} --registryAuth=None
```

Выполнить удаление временной конфигурации monqctl:

```
rm ~/.monq/config.yml
rm -rf /tmp/monqctl
```

Инициализация Control Plane

Дополнительные варианты развертывания:

- Приложение. Kubernetes режим MultiMaster.

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

> Внимание! Команды выполняются на сервере master.

> 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.

Задать нужное имя сервера, в примере используется master (имя сервера должно корректно разрешаться в ip адрес сервера):

```
hostnamectl set-hostname master
```

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
```

Выполнить конфигурацию containerd:

```
containerd config default > /etc/containerd/config.toml
sed -i 's/SystemdCgroup = false/SystemdCgroup = true/g' /etc/containerd/config.toml
sed -i "s|k8s.gcr.io/pause:3.6|${registry_address}/pause:3.6|g" /etc/containerd/config.toml
systemctl enable containerd
systemctl restart containerd
```

Настроить автодополнение команд:

```
mkdir -p /etc/bash_completion.d/
kubectl completion bash > /etc/bash_completion.d/kubectl
crictl completion bash > /etc/bash_completion.d/crictl
helm completion bash > /etc/bash_completion.d/helm
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
```

Провести инициализацию control plane:

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

systemctl enable kubelet.service
kubeadm init --control-plane-endpoint "${k8s_domain}:6443" \
  --skip-phases="addon/kube-proxy" \
  --upload-certs \
  --pod-network-cidr="10.244.0.0/16" \
  --service-cidr="10.16.0.0/16" \
  --image-repository="${registry_address}" \
  --kubernetes-version 1.26.15
```

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
```

В данном примере используется cilium в режиме kube-proxy replacement (kubernetes инициализирован без kube-proxy). Использование других CNI возможно, но не описывается в данной документации. При использовании других CNI следует изменить команду инициализации для запуска kube-proxy.

Выполнить установку cilium:

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

helm upgrade --install cilium ./offline/charts/cilium-1.13.3.tgz \
  --namespace kube-system \
  --set kubeProxyReplacement=strict \
  --set k8sServiceHost=${api_server_host} \
  --set k8sServicePort=${api_server_port} \
  --set operator.replicas=1 \
  --set ipam.mode=kubernetes \
  --set image.useDigest=false \
  --set image.repository="${registry_address}/cilium/cilium" \
  --set operator.image.useDigest=false \
  --set operator.image.repository="${registry_address}/cilium/operator" \
  --set hubble.enabled=false \
  --set l7Proxy=false
```

Убедиться что нода перешла в статус `Ready`:

```
kubectl get node -w
```

Включение нод в kubernetes

> Внимание! Команды выполняются на всех серверах worker/db

> 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.

Задать нужное имя сервера, в примере используется worker:

```
hostnamectl set-hostname worker
```

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
```

Выполнить конфигурацию containerd:

```
containerd config default > /etc/containerd/config.toml
sed -i 's/SystemdCgroup = false/SystemdCgroup = true/g' /etc/containerd/config.toml
sed -i "s|k8s.gcr.io/pause:3.6|${registry_address}/pause:3.6|g" /etc/containerd/config.toml
systemctl enable containerd
systemctl restart containerd
```

Настроить автодополнение команд:

```
mkdir -p /etc/bash_completion.d/
crictl completion bash > /etc/bash_completion.d/crictl
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
```

Импортировать CA сертификат выпущенный ранее на шаге Выпуск CA сертификата:

```
mkdir /usr/share/ca-certificates/monq
cp certs/monq.ca.crt /usr/share/ca-certificates/monq/monq.ca.crt
echo "monq/monq.ca.crt" >> /etc/ca-certificates.conf
update-ca-certificates
```

> 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 и container-registry в /etc/hosts.

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

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

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

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

```
kubectl get node -w
```

В течении двух минут статус должен измениться.

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

Последующая настройка кластера

> Внимание! Команды выполняются на сервере master.

Ingress controller

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.

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.

Добавить метку на ноду, определяющую её как разрешенную для запуска ingress-nginx controller:

```
server_name=worker
kubectl label node ${server_name} ingress=
```

Запустить контроллер с помощью helm chart:

```
helm upgrade --install ingress-nginx offline/charts/ingress-nginx-4.7.0.tgz \
  --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 \
  --set controller.image.registry=${registry_address} \
  --set controller.image.digest=null \
  --set controller.admissionWebhooks.patch.image.registry=${registry_address} \
  --set controller.admissionWebhooks.patch.image.digest=null
```

Проверить состояние контейнеров контроллера:

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

Пространства для запуска компонентов.

Создать namespace для запуска инфраструктурных компонентов и monq:

```
kubectl create namespace ${infra_namespace}
kubectl create namespace ${monq_namespace}
```

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

> Внимание! Данную команду необходимо повторить для всех серверов с ролью worker.

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

Задать лимиты 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: ${monq_namespace}
spec:
  limits:
  - default:
      cpu: 2
      memory: 2048Mi
    defaultRequest:
      cpu: 25m
      memory: 128Mi
    type: Container
EOF
```

Создать secret содержащий CA сертификат выпущенный ранее в разделе Выпуск CA сертификата для его переиспользования в остальных компонентах:

```
kubectl create secret generic -n ${monq_namespace} monq-ca \
  --from-file=./certs/monq.ca.key --from-file=./certs/monq.ca.crt

kubectl create secret generic -n ${monq_namespace} monq-ca-certificates \
  --from-file=./certs/monq.ca.crt
```

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.

Требуется RWX (readWriteMany) хранилище, так-как в Monq используется подключение данного хранилища в несколько сервисов (реплик). Список совместимых решений можно посмотреть в официальной документации.

В данной инструкции будет рассмотрено решение на базе NFS, так-как оно является самым распространенным.

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

Общий порядок установки и настройки:

В данном примере будет использован сервер db, на котором ранее был установлен NFS в разделе Установка сервера NFS.

Создать PV и PVC:

```
host_path="/storage/nfs"
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: ${monq_namespace}
    annotations:
      volume.beta.kubernetes.io/storage-class: nfs
  spec:
    accessModes:
      - ReadWriteMany
    resources:
      requests:
        storage: ${nfs_storage_size}
kind: List
metadata:
```

```
  resourceVersion: ""
  selfLink: ""
EOF
```

Авторизация в кластере kubernetes.

Выписать и зафиксировать токен и адрес подключения к api k8s в authfile.

Применить манифесты сервисной учетной записи:

```
cat <<EOF | kubectl create -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
```

Получить токен:

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

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}>";

Consul.

Хранилище конфигураций Monq.

Рассматривается SelfHosted, запуск в единичном экземпляре statefulSet с помощью helm chart. Без высокой доступности.

Перед началом работ необходимо создать на внешнем DNS сервере запись для хоста consul вида consul.${infra_domain}, запись должна разрешаться в IP сервера на котором расположен consul(в данном примере сервер db).

Выполнить назначение метки на выбранную ноду:

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

Выполнить запуск с помощью helm chart:

```
helm install consul offline/charts/consul-1.0.1.tgz \
  --namespace ${infra_namespace} --create-namespace \
  --set application.image.registry=${registry_address} \
  --set application.ssl.enable=false
```

> Устройство чарта описано в соответствующем приложении, см. Приложение. Описание типового helm chart.

Инициализировать consul:

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

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

Создать токен агента в consul:

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

Применить токен агента в consul:

```
curl -X PUT -H "X-Consul-Token: ${bootstrap_token}" \
consul.${infra_domain}:8500/v1/agent/token/acl_agent_token \
  -d '{ "Token": "'${agent_token}'" }'
```

Выполнить проверку:

1. Состояние запуска контейнера

   ```
   kubectl get po -n ${infra_namespace} consul-0 -o wide -w
   ```

2. Логи контейнера

   ```
   kubectl logs -n ${infra_namespace} consul-0 -f
   ```

3. Возможность подключения с авторизационными данными

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.

СУБД, кеш и диспетчер сообщений

Общие положения:

- В данном документе рассматривается решение по запуску всех компонентов внутри kubernetes;

- В случае требуется запустить компонент СПО другим методом, рекомендуем воспользоваться официальной инструкцией поставщика решения. Настройку необходимо будет произвести как managed сервис;
- В предложенных примерах в качестве хранилища данных для компонентов СУБД используется локальный каталог (hostPath volume) на ноде. Этот способ указан для примера, что бы программный комплекс можно было запустить вне зависимости от наличия того или иного хранилища, а следуя пошаговой инструкции. Не рекомендуется использовать данный тип хранилища для промышленной эксплуатации. 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.
- Для установки большинства компонентов требуются установленные Helm и Crane см. Установка пакетов;
- В примере будет рассмотрена установка компонентов без резервирования;
- Методы запуска HA и managed решений отражены в приложениях.

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.

Инструкция составлена с учетом выполнения команд от пользователя `root`, поэтому перед выполнением команд необходимо перейти на использование root и установить переменные:

```
su -
infra_domain=in.monq.local
registry_address=registry.${infra_domain}:5000
infra_namespace=infra
monq_namespace=production
```

PostgreSQL

Дополнительные варианты развертывания:

- Приложение. HA PostgreSQL.
- Приложение. Managed services, раздел PostgreSQL.

Перед началом работ необходимо создать на внешнем DNS сервере запись для хоста postgres вида `postgresql.${infra_domain}`, запись должна разрешаться в IP сервера на котором расположен postgres(в данном примере сервер db).

> Внимание! Команды выполняются на сервере master.

Выполнить назначение метки на выбранную ноду:

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

Выполнить запуск с помощью helm chart:

```
helm install postgresql offline/charts/postgresql-1.0.1.tgz \
  --namespace ${infra_namespace} --create-namespace \
  --set application.image.registry=${registry_address} \
  --set application.ssl.enable=false
```

> Устройство чарта описано в соответствующем приложении, см. Приложение. Описание типового helm chart.

Получить автоматически сгенерированный пароль пользователя `postgres`:

```
postgres_password=$(kubectl get secrets -n ${infra_namespace} postgresql-secret \
  -o jsonpath='{.data.POSTGRES_PASSWORD}' | base64 --decode)
echo "save it: ${postgres_password}"
```

Выполнить проверку:

1. Состояние запуска контейнера:

   ```
   kubectl get po -n ${infra_namespace} postgresql-0 -o wide
   ```

2. Логи контейнера:

   ```
   kubectl logs -n ${infra_namespace} postgresql-0 -f
   ```

3. Возможность подключения с авторизационными данными.

   ```
   kubectl run utils -n ${monq_namespace} \
     --image="${registry_address}/utils:3.0" -- \
     /bin/bash -c -- "trap : TERM INT; sleep infinity & wait"

   kubectl exec -it -n ${monq_namespace} utils \
     -- psql -c "SHOW server_version;" \
     postgresql://postgres:${postgres_password}@postgresql.${infra_domain}

   kubectl delete pod -n ${monq_namespace} utils
   ```

Для наполнения authfile выставить значения:

- postgresql.host: `"postgresql.in.monq.local"`;
- postgresql.port: 5432;
- postgresql.ssl: **`false`**;
- postgresql.ssl_mode: `"Require"`;
- postgresql.ssl_trust: **`true`**;

- postgresql.users.root_user.name: `"postgres"`;
- postgresql.users.root_user.password: `"< check ${postgres_password}" >`.

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

RabbitMQ

Дополнительные варианты развертывания:

- Приложение. HA RabbitMQ.
- Приложение. Managed services, раздел RabbitMQ.

Перед началом работ необходимо создать на внешнем DNS сервере запись для хоста consul вида `rabbitmq.${infra_domain}`, запись должна разрешаться в IP сервера на котором расположен rabbitmq(в данном примере сервер db).

> Внимание! Команды выполняются на сервере master.

Выполнить назначение метки на выбранную ноду:

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

Выполнить запуск с помощью helm chart:

```
helm install rabbitmq offline/charts/rabbitmq-1.0.1.tgz \
  --namespace ${infra_namespace} --create-namespace \
  --set application.image.registry=${registry_address} \
  --set application.ssl.enable=false
```

> Устройство чарта описано в соответствующем приложении, см. Приложение. Описание типового helm chart.

Выполнить проверку:

1. Состояние запуска контейнера:

   ```
   kubectl get po -n ${infra_namespace} rabbitmq-0 -o wide
   ```

2. Логи контейнера:

   ```
   kubectl logs -n ${infra_namespace} rabbitmq-0 -f
   ```

Создать пользователя с административными правами:

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

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

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

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

Для наполнения authfile выставить значения:

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

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

Arangodb

Дополнительные варианты развертывания:

- Приложение. HA Arangodb.
- Приложение. Managed services, раздел Arangodb.

Перед началом работ необходимо создать на внешнем DNS сервере запись для хоста postgres вида `arangodb.${infra_domain}`, запись должна разрешаться в IP сервера на котором расположен arangodb(в данном примере сервер db).

> Внимание! Команды выполняются на сервере master.

Выполнить назначение метки на выбранную ноду:

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

Выполнить запуск с помощью helm chart:

```
helm install arangodb offline/charts/arangodb-1.0.1.tgz \
  --namespace ${infra_namespace} --create-namespace \
  --set application.image.registry=${registry_address} \
  --set application.ssl.enable=false
```

> Устройство чарта описано в соответствующем приложении, см. Приложение. Описание типового helm chart.

Получить автоматически сгенерированный пароль пользователя `root`:

```
arangodb_password=$(kubectl get secrets -n ${infra_namespace} arangodb-secret \
  -o jsonpath='{.data.ARANGO_ROOT_PASSWORD}' | base64 --decode)
echo "save it: ${arangodb_password}"
```

Выполнить проверку:

1. Состояние запуска контейнера:

   ```
   kubectl get po -n ${infra_namespace} arangodb-0 -o wide -w
   ```

2. Логи контейнера:

   ```
   kubectl logs -n ${infra_namespace} arangodb-0 -f
   ```

3. Возможность подключения с авторизационными данными.

   ```
   kubectl run utils -n ${monq_namespace} \
     --image="${registry_address}/utils:3.0" -- \
     /bin/bash -c -- "trap : TERM INT; sleep infinity & wait"

   kubectl exec -it -n ${monq_namespace} utils \
     -- curl -X POST -u root:${arangodb_password} \
     arangodb.${infra_domain}:8529/_db/_system/_admin/echo \
     -d "string" | jq

   kubectl delete pod -n ${monq_namespace} utils
   ```

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.

Clickhouse

Дополнительные варианты развертывания:

- Приложение. HA Clickhouse.
- Приложение. Managed services, раздел Clickhouse.

Перед началом работ необходимо создать на внешнем DNS сервере запись для хоста postgres вида `clickhouse.${infra_domain}`, запись должна разрешаться в IP сервера на котором расположен clickhouse(в данном примере сервер db).

> Внимание! Команды выполняются на сервере master.

Выполнить назначение метки на выбранную ноду:

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

Выполнить запуск с помощью helm chart:

```
helm install clickhouse offline/charts/clickhouse-1.0.1.tgz \
  --namespace ${infra_namespace} --create-namespace \
  --set application.image.registry=${registry_address} \
  --set application.ssl.enable=false
```

> Устройство чарта описано в соответствующем приложении, см. Приложение. Описание типового helm chart.

Получить автоматически сгенерированный пароль пользователя `admin`:

```
clickhouse_password=$(kubectl get secrets -n ${infra_namespace} clickhouse-secret \
  -o jsonpath='{.data.CLICKHOUSE_ADMIN_PASSWORD}' | base64 --decode)
echo "save it: ${clickhouse_password}"
```

Выполнить проверку:

1. Состояние запуска контейнера:

   ```
   kubectl get po -n ${infra_namespace} clickhouse-0 -o wide
   ```

2. Логи контейнера:

   ```
   kubectl logs -n ${infra_namespace} clickhouse-0 -f
   ```

3. Возможность подключения с авторизационными данными:

   ```
   kubectl run utils -n ${monq_namespace} \
     --image="${registry_address}/utils:3.0" -- \
     /bin/bash -c -- "trap : TERM INT; sleep infinity & wait"

   kubectl exec -it -n ${monq_namespace} utils \
     -- curl --get --data-urlencode "query=SELECT version()" \
     -u root_user:${clickhouse_password} clickhouse.${infra_domain}:8123

   kubectl delete pod -n ${monq_namespace} utils
   ```

Для наполнения authfile выставить значения:

- clickhouse.host: `"clickhouse.in.monq.local"`;
- clickhouse.port: 8123;

- clickhouse.proto: `"http"`;
- clickhouse.cluster: **`false`**;
- clickhouse.cluster_name: `"monq"`;
- clickhouse.users.root_user.name: `"root_user"`;
- clickhouse.users.root_user.password: `"<check ${clickhouse_password}>"`.

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

Redis

Дополнительные варианты развертывания:

- Приложение. HA Redis.
- Приложение. Managed services, раздел Redis.

Перед началом работ необходимо создать на внешнем DNS сервере запись для хоста postgres вида `redis.${infra_domain}`, запись должна разрешаться в IP сервера на котором расположен redis(в данном примере сервер db).

> Внимание! Команды выполняются на сервере master.

Выполнить назначение метки на выбранную ноду:

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

Выполнить запуск с помощью helm chart:

```
helm install redis offline/charts/redis-1.0.1.tgz \
  --namespace ${infra_namespace} --create-namespace \
  --set application.image.registry=${registry_address} \
  --set application.ssl.enable=false
```

> Устройство чарта описано в соответствующем приложении, см. Приложение. Описание типового helm chart.

Получить автоматически сгенерированный пароль пользователя **`default`**:

```
redis_password=$(kubectl get secrets -n ${infra_namespace} redis-secret \
  -o jsonpath='{.data.REDIS_PASSWORD}' | base64 --decode)
echo "save it: ${redis_password}"
```

Выполнить проверку:

1. Состояние запуска контейнера:

   ```
   kubectl get po -n ${infra_namespace} redis-0 -o wide
   ```

2. Логи контейнера:

```
kubectl logs -n ${infra_namespace} redis-0 -f
```

3. Возможность подключения с авторизационными данными:

```
kubectl run utils -n ${monq_namespace} \
  --image="${registry_address}/utils:3.0" -- \
  /bin/bash -c -- "trap : TERM INT; sleep infinity & wait"

kubectl exec -it -n ${monq_namespace} utils \
 -- redis-cli -h redis.${infra_domain} -a "${redis_password}" PING

kubectl delete pod -n ${monq_namespace} utils
```

Для наполнения authfile выставить значения:

- redis.host: "redis.in.monq.local";
- redis.port: 6379;
- redis.ssl: **false**;
- redis.sentinel: **false**;
- redis.service_name: "mymaster";
- redis.users.root_user.password: "<check ${redis_password}">.

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

VictoriaMetrics

Дополнительные варианты развертывания:

- Приложение. HA VictoriaMetrics.
- Приложение. Managed services, раздел Victoria metrics.

Перед началом работ необходимо создать на внешнем DNS сервере запись для хоста postgres вида victoriametrics.${infra_domain}, запись должна разрешаться в IP сервера на котором расположен victoriametrics(в данном примере сервер db).

> Внимание! Команды выполняются на сервере master.

Выполнить назначение метки на выбранную ноду:

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

Выполнить запуск с помощью helm chart:

```
helm install victoriametrics offline/charts/victoriametrics-1.0.1.tgz \
  --namespace ${infra_namespace} --create-namespace --version 1.0.1 \
  --set application.image.registry=${registry_address} \
  --set application.ssl.enable=false
```

> Устройство чарта описано в соответствующем приложении, см. Приложение. Описание типового helm chart.

Получить автоматически сгенерированный пароль пользователя monq:

```
vm_password=$(kubectl get secrets -n ${infra_namespace} victoriametrics-secret \
  -o jsonpath='{.data.VM_AUTH_PASSWORD}' | base64 --decode)
echo "save it: ${vm_password}"
```

Выполнить проверку

1. Состояние запуска контейнера:

   ```
   kubectl get po -n ${infra_namespace} victoriametrics-0 -o wide
   ```

2. Логи контейнера:

   ```
   kubectl logs -n ${infra_namespace} victoriametrics-0 -f
   ```

3. Возможность подключения с авторизационными данными.

   ```
   kubectl run utils -n ${monq_namespace} \
     --image="${registry_address}/utils:3.0" -- \
     /bin/bash -c -- "trap : TERM INT; sleep infinity & wait"

   kubectl exec -it -n ${monq_namespace} utils \
     -- curl -u monq:${vm_password} \
     victoriametrics.${infra_domain}:8428/prometheus/api/v1/status/tsdb

   kubectl delete pod -n ${monq_namespace} utils
   ```

4. 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_password}">.

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

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.

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.

Список переменных, используемых по тексту:

- `${global_domain}` - основное доменное имя развертываемого приложения
- `${registry_address}` - адрес репозитория контейнеров, если используется развернутый по данной инструкции, то http://registry.in.monq.local:5000

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. Выбрать доменное имя для установки Monq, далее будет обозначено как `${global_domain}`;
4. Обеспечить разрешение доменных имен `${global_domain}`, `api.${global_domain}`, `registry.api.${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;

Запуск сценария установки

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

В данном примере рассматривается запуск установки с помощью контейнера, запущенного вне кластера kubernetes, для запуска установки внутри кластера kubernetes, см. Приложение. Установка внутри кластера kubernetes.

Сценарий установки запускается за пределами инфраструктуры развертываемой системы, например с машины системного администратора. Настроен вышестоящий DNS, все требуемые доменные имена (список доменных имен) корректно разрешаются в IP адреса соответствующих сервисов.

Создать каталог для артефактов установки, разместить в данном каталоге authfile.

```
installer_files="< каталог с артефактами установщика и файлом авторизации СПО, по умолчанию /
    opt/monq >"
mkdir -p ${installer_files}

cp `authfile` ${installer_files}/system_auth.json
```

Запустить контейнер со сценарием установки:

```
installer_files="< каталог с артефактами установщика и файлом авторизации СПО, по умолчанию /
    opt/monq >"
global_domain="доменное< имя>"
registry_address="адрес< репозитория контейнеров>"

crictl pull ${registry_address}/installer:8.0.0
ctr -n k8s.io run --rm --net-host \
  --mount type=bind,src=${installer_files},dst=/opt/monq,options=rbind:rw \
  ${registry_address}/installer:8.0.0 monq-installer-temp \
  ansible-playbook -e "global_domain='${global_domain}'"\
  monq/monq.yaml
```

В случае возникновения ошибок они будут отражены в файле `errors.json`, расположенном в каталоге `${installer_files}`.

После установки система готова к использованию, интерфейс доступен по адресу `${global_domain}`.

Авторизационные данные по-умолчанию:

- логин: `admin@${global_domain}`
- пароль: `monq_admin`

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.

> Внимание! Удаление должно запускаться с теми же переменными, что и установка

Запустить контейнер со сценарием удаления:

```
installer_files="< каталог с артефактами установщика и файлом авторизации СПО, по умолчанию /
    opt/monq >"

ctr -n k8s.io run --rm --net-host \
  --mount type=bind,src=${installer_files},dst=/opt/monq,options=rbind:rw \
  ${registry_address}/installer:8.0.0 monq-installer-temp \
  ansible-playbook -e "global_domain='${global_domain}'"\
  monq/eraser.yaml
```

Приложение. 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.amqp_ssl	bool	использовать ssl для amqp
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.ssl	bool	использовать ssl для redis
redis.ssl_host	bool	опционально, CN в сертификате 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.name	string	имя пользователя redis, если включен ACL
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`)

Ключ	Тип	Описание
registry.auth_type	string	тип авторизации в docker registry (`None`, `BasicAuth`, `Token`)
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`)
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

Ключ	Тип	Описание
		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
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.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","amqp_ssl":false,"quorum_queues":false,"virtual_host":"/",
  "users":{"root_user":{"name":"root","password":"*********"}}
 },
 "redis":{"host":"redis.in.monq.local","port":6379,"ssl":false,"sentinel":false,
  "service_name":"mymaster",
  "users":{"root_user":{"password":"********"}}
 },
 "registry":{"host":"registry.in.monq.local","port":5000,"proto":"https",
  "auth_type":"None","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. Kubernetes режим MultiMaster.

> Внимание! Перед началом работ по данной главе предварительно должны быть установлены необходимые пакеты, см. Установка пакетов

Продуктивная среда, рекомендуется для установок с высокими требованиями по отказоустойчивости.

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.

> Внимание! Команды выполняются на всех серверах master.

Задать нужное имя сервера, в примере используется `master-<n>`, где `<n>` - порядковый номер сервера:

```
hostnamectl set-hostname master-<n>
```

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
```

Выполнить конфигурацию containerd:

```
containerd config default > /etc/containerd/config.toml
sed -i 's/SystemdCgroup = false/SystemdCgroup = true/g' /etc/containerd/config.toml
sed -i "s|k8s.gcr.io/pause:3.6|${registry_address}/pause:3.6|g" /etc/containerd/config.toml
systemctl enable containerd
systemctl restart containerd
```

Настроить автодополнение команд:

```
mkdir -p /etc/bash_completion.d/
kubectl completion bash > /etc/bash_completion.d/kubectl
crictl completion bash > /etc/bash_completion.d/crictl
helm completion bash > /etc/bash_completion.d/helm
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
```

> Внимание! Команды выполняются на первом master сервере.

Провести инициализацию control plane на первом master:

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

systemctl enable kubelet.service
kubeadm init --control-plane-endpoint "${k8s_domain}:6443" \
  --skip-phases="addon/kube-proxy" \
  --upload-certs \
  --pod-network-cidr="10.244.0.0/16" \
  --service-cidr="10.16.0.0/16" \
  --image-repository="${registry_address}" \
  --kubernetes-version 1.26.15
```

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
```

В данном примере используется cilium в режиме kube-proxy replacement (kubernetes инициализирован без kube-proxy). Использование других CNI возможно, но не описывается в данной документации. При использовании других CNI следует изменить команду инициализации для запуска kube-proxy.

Выполнить установку cilium:

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

helm upgrade --install cilium ./offline/charts/cilium-1.13.3.tgz \
  --namespace kube-system \
  --set kubeProxyReplacement=strict \
  --set k8sServiceHost=${api_server_host} \
  --set k8sServicePort=${api_server_port} \
  --set operator.replicas=1 \
  --set ipam.mode=kubernetes \
  --set image.useDigest=false \
  --set image.repository="${registry_address}/cilium/cilium" \
  --set operator.image.useDigest=false \
  --set operator.image.repository="${registry_address}/cilium/operator" \
  --set hubble.enabled=false \
  --set l7Proxy=false
```

Убедиться что нода перешла в статус Ready:

```
kubectl get node
```

> Внимание! Команды выполняются на оставшихся master серверах.

Выполнить подключение в кластер оставшихся master нод:

```
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 *******
```

Убедиться что ноды перешли в статус `Ready`:

```
kubectl get node
```

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

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

Выполнить запуск kube-vip для организации virtual ip между master нодами.

> Внимание! Команды выполняются на всех master серверах.

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

ctr image pull ${registry_address}/kube-vip/kube-vip:v0.5.0
ctr run --rm --net-host ${registry_address}/kube-vip/kube-vip:v0.5.0 vip /kube-vip \
  manifest pod --interface $interface --address $VIP --controlplane \
  --arp --prometheusHTTPServer 127.0.0.1:2112 --leaderElection \
  | sed "s|ghcr.io/|${registry_address}/|g" | tee /etc/kubernetes/manifests/kube-vip.yaml
```

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

После выполнения описанных в разделе команд необходимо продолжить установку с раздела Включение нод в kubernetes

Приложение. Установка пакетов Offline.

Подготовка установочного пакета

> Внимание! Команды из данного раздела необходимо выполнить на сервере имеющем доступ в интернет!

> Внимание! Для обеспечения совместимости с Debian 11.7 подготовка установочного пакета так же должна производиться на Debian 11.7.

Пакеты системного программного обеспечения

Подготовить OS для скачивания пакетов:

```
su
apt install -y apt-rdepends gpg curl ca-certificates wget unzip

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

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | \
gpg --dearmor -o /usr/share/keyrings/kubernetes-apt-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/kubernetes-apt-keyring.gpg] \
https://pkgs.k8s.io/core:/stable:/v1.26/deb/ /" \
| tee /etc/apt/sources.list.d/kubernetes.list

apt update

mkdir -p offline/packages/{common,kubernetes,nfs,registry,tools}
chown -R _apt:root offline/packages
```

Скачать базовый набор пакетов:

```
cd offline/packages/common
apt download $(apt-rdepends sudo gpg curl wget dnsutils vim telnet unzip bash-completion \
 ca-certificates jq libicu67 nfs-common mount | grep -v "^ " \
 | sed 's/debconf-2.0/debconf/g;/^host/d'); cd -
```

Скачать набор пакетов платформы kubernetes:

```
cd offline/packages/kubernetes
apt download containerd.io=1.6.6-1 kubelet=1.26.15-1.1 kubeadm=1.26.15-1.1 \
 kubectl=1.26.15-1.1 \
 $(apt-rdepends conntrack ebtables ethtool iproute2 iptables kubernetes-cni socat \
 util-linux cri-tools | grep -v "^ " | sed 's/debconf-2.0/debconf/g;/^host/d') ; cd -
```

Скачать пакеты nfs сервера:

```
cd offline/packages/nfs
apt download $(apt-rdepends nfs-kernel-server  | grep -v "^ " \
 | sed 's/debconf-2.0/debconf/g;/^host/d'); cd -
```

Скачать архивы с утилитами:

```
mkdir offline/tools
cd offline/tools

helm_version="3.12.2"
wget https://get.helm.sh/helm-v${helm_version}-linux-amd64.tar.gz
wget https://downloads.monq.ru/tools/monqctl/v1.13.0/linux-x64/monqctl.zip

crane_version="0.16.1"
wget "https://github.com/google/go-containerregistry/releases/download\
/v${crane_version}/go-containerregistry_Linux_x86_64.tar.gz"

registry_version="2.8.3"
wget "https://github.com/distribution/distribution\
/releases/download/v${registry_version}/registry_${registry_version}_linux_amd64.tar.gz"

cd -
```

Образы контейнеров платформы kubernetes

Подготовить OS для скачивания образов:

```
mkdir -p offline/images
tar -xf offline/tools/go-containerregistry_Linux_x86_64.tar.gz -C /usr/local/bin/ crane
cd offline/images
```

Скачать образы контейнеров платформы kuberentes:

```
crane pull registry.k8s.io/coredns/coredns:v1.9.3 coredns.tar
crane pull registry.k8s.io/kube-proxy:v1.26.15 kube-proxy.tar
crane pull registry.k8s.io/pause:3.6 pause3.6.tar
crane pull registry.k8s.io/pause:3.9 pause3.9.tar
crane pull registry.k8s.io/kube-apiserver:v1.26.15 kube-apiserver.tar
crane pull registry.k8s.io/kube-controller-manager:v1.26.15 kube-controller-manager.tar
crane pull registry.k8s.io/kube-scheduler:v1.26.15 kube-scheduler.tar
crane pull registry.k8s.io/etcd:3.5.10-0 etcd.tar
crane pull quay.io/cilium/cilium:v1.13.3 cilium.tar
crane pull quay.io/cilium/operator-generic:v1.13.3 cilium-operator.tar
crane pull registry.k8s.io/ingress-nginx/controller:v1.8.0 ingress-nginx.tar
crane pull registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20230407 \
  kube-webhook-certgen.tar
```

Скачать образы контейнеров используемого СПО:

```
crane pull consul:1.8.0 consul.tar
crane pull arangodb:3.11.2 arangodb.tar
```

```
crane pull clickhouse/clickhouse-server:23.3.8 clickhouse.tar
crane pull postgres:12.15 postgres.tar
crane pull rabbitmq:3.11.18-management rabbitmq.tar
crane pull redis:7.0.11 redis.tar
crane pull victoriametrics/victoria-metrics:v1.91.3 victoria-metrics.tar
cd -
```

В случае если планируется HA инсталляция, скачать дополнительные образы:

```
cd offline/images
crane pull ghcr.io/kube-vip/kube-vip:v0.5.0 kube-vip.tar
crane pull registry.opensource.zalan.do/acid/postgres-operator:v1.10.1 postgres-operator.tar
crane pull ghcr.io/zalando/spilo-15:3.0-p1 spilo.tar
crane pull altinity/clickhouse-operator:0.21.3 clickhouse-operator.tar
crane pull pravega/zookeeper:0.2.15 zookeeper.tar
crane pull pravega/zookeeper-operator:0.2.15 zookeper-operator.tar
crane pull lachlanevenson/k8s-kubectl:v1.23.2 k8s-kubectl.tar
crane pull arangodb/kube-arangodb:1.2.32 kube-arangodb.tar
crane pull alpine:3.11 alpine.tar
crane pull victoriametrics/operator:v0.34.1 vmoperator.tar
crane pull victoriametrics/vminsert:v1.91.3-cluster vminsert.tar
crane pull victoriametrics/vmselect:v1.91.3-cluster vmselect.tar
crane pull victoriametrics/vmstorage:v1.91.3-cluster vmstorage.tar
crane pull quay.io/spotahome/redis-operator:v1.2.4 redis-operator.tar
crane pull rabbitmqoperator/cluster-operator:2.3.0 rabbitmq-operator.tar
cd -
```

Helm chart для платформы kubernetes

Подготовить OS для скачивания чартов:

```
helm_version="3.12.2"
tar -xf offline/tools/helm-v${helm_version}-linux-amd64.tar.gz linux-amd64/helm
mv linux-amd64/helm /usr/bin/
rm -r linux-amd64/

mkdir -p offline/charts
cd offline/charts
```

Скачать чарты:

```
helm repo add cilium https://helm.cilium.io/
helm pull cilium/cilium --version 1.13.3

helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm pull ingress-nginx/ingress-nginx --version 4.7.0

helm repo add monq https://helm.monq.ru/charts
helm pull monq/postgresql --version 1.0.1
helm pull monq/clickhouse --version 1.0.1
helm pull monq/arangodb --version 1.0.1
helm pull monq/victoriametrics --version 1.0.1
helm pull monq/redis --version 1.0.1
helm pull monq/rabbitmq --version 1.0.1
helm pull monq/consul --version 1.0.1
cd -
```

В случае если планируется HA инсталляция, скачать дополнительные чарты:

```
cd offline/charts
helm repo add postgres-operator-charts \
```

```
https://opensource.zalando.com/postgres-operator/charts/postgres-operator
helm pull postgres-operator-charts/postgres-operator --version 1.10.1

helm repo add clickhouse-operator https://docs.altinity.com/clickhouse-operator/
helm pull clickhouse-operator/altinity-clickhouse-operator --version 0.21.3
helm repo add pravega https://charts.pravega.io
helm pull pravega/zookeeper-operator --version 0.2.15

url="https://github.com/arangodb/kube-arangodb/releases/download/1.2.32"
helm pull ${url}/kube-arangodb-1.2.32.tgz

helm repo add vm https://victoriametrics.github.io/helm-charts/
helm pull vm/victoria-metrics-operator --version 0.23.1

helm repo add redis-operator https://spotahome.github.io/redis-operator
helm pull redis-operator/redis-operator --version 3.2.8

wget "https://github.com/rabbitmq/cluster-operator\
/releases/download/v2.3.0/cluster-operator.yml"
cd -
```

Образы контейнеров Monq

Подготовить OS для скачивания образов:

```
token="< токен обновления полученный с сайта. выписывается вместе с лицензией >"

unzip offline/tools/monqctl.zip
mv monqctl /usr/local/bin/

monqctl config set instance temp --server=http://registry.api.monq.local
monqctl config set credential temp --registry-token=000
monqctl config set releasehub monq-release-hub --token=${token}
monqctl config set context temp --instance=temp --credential=temp \
--releasehub=monq-release-hub
monqctl config use-context temp
```

Скачать образы Monq:

```
mkdir -p offline/monq/{installer,registry,release}

monqctl release version export 8.0.0 --product=installer \
 --dest=offline/monq/installer --full
monqctl release version export 3.13.8 --product=monq-registry \
 --dest=offline/monq/registry --full
monqctl release version export 8.0.0 --product=monq \
 --dest=offline/monq/release --full
```

Сборка итогового архива

```
tar -zcf offline.tar.gz offline/
```

Установка из каталога.

Перед началом установки необходимо перенести сформированный архив на все сервера и выполнить его распаковку:

```
su -
tar -xf ./offline.tar.gz
rm ./offline.tar.gz
```

Установка пакетов

> Внимание! Команды из данного раздела необходимо выполнить на всех серверах!

Выполнить установку пакетов:

```
dpkg -i ./offline/packages/common/*
dpkg -i ./offline/packages/kubernetes/*

helm_version="3.12.2"
tar -xf ./offline/tools/helm-v${helm_version}-linux-amd64.tar.gz linux-amd64/helm
mv linux-amd64/helm /usr/local/bin/
rm -r linux-amd64/

crane_version="0.16.1"
tar -xf ./offline/tools/go-containerregistry_Linux_x86_64.tar.gz -C /usr/local/bin/ crane

unzip ./offline/tools/monqctl.zip
mv monqctl /usr/local/bin
```

Установка Container registry

Перед началом работ необходимо создать на внешнем DNS сервере запись для хоста registry вида `registry.${infra_domain}`, запись должна разрешаться в IP сервера на котором расположен arangodb(в данном примере сервер db).

> Внимание! Команды выполняются на сервере db.

Выполнить установку container registry:

```
registry_version="2.8.3"

tar -xf ./offline/tools/registry_${registry_version}_linux_amd64.tar.gz \
 -C /usr/local/bin/ registry
```

Создать пользователя для registry:

```
useradd --no-create-home --shell /bin/false registry
```

Создать каталоги для работы приложения:

```
mkdir -p /storage/registry
chown -R registry /storage/registry
mkdir -p /etc/docker/registry
```

Сформировать конфигурационный файл:

```
cat <<EOF | tee /etc/docker/registry/config.yaml
version: 0.1
log:
```

```
  fields:
    service: registry
storage:
  cache:
    blobdescriptor: inmemory
  filesystem:
    rootdirectory: /storage/registry
http:
  addr: :5000
  tls:
    certificate: /etc/docker/registry/registry.crt
    key: /etc/docker/registry/registry.key
  headers:
    X-Content-Type-Options: [nosniff]
health:
  storagedriver:
    enabled: true
    interval: 10s
    threshold: 3
EOF
```

Создать unit для systemd:

```
cat <<EOF | tee /etc/systemd/system/registry.service
[Unit]
Description=docker private registry service

[Service]
ExecStart=/usr/local/bin/registry serve /etc/docker/registry/config.yaml
Restart=always
Type=simple
RestartSec=10s
User=registry

[Install]
WantedBy=multi-user.target
EOF
```

Скопировать сертификат registry выпущенный ранее на шаге Выпуск CA сертификата:

```
cp certs/docker-registry/registry.key certs/docker-registry/registry.crt /etc/docker/registry
chown -R registry /etc/docker/registry
```

Выполнить запуск container registry:

```
systemctl start registry
systemctl enable registry
```

Для наполнения authfile выставить значения:

- registry.host: `"registry.in.monq.local"`
- registry.port: `5000`
- registry.proto: `"https"`
- registry.auth_type: `"None"`

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

Установка сервера NFS

> Внимание! Команды выполняются на сервере db.

```
dpkg -i ./offline/packages/nfs/*
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
```

Создать каталоги для хранения данных компонентов СПО:

```
mkdir -p /storage/consul
mkdir -p /storage/arangodb
mkdir -p /storage/clickhouse
mkdir -p /storage/postgresql
mkdir -p /storage/rabbitmq
mkdir -p /storage/redis
mkdir -p /storage/victoriametrics
```

Импорт контейнеров СПО

Импортировать образы контейнеров используемого СПО:

```
crane push ./offline/images/coredns.tar ${registry_address}/coredns:v1.9.3
crane push ./offline/images/kube-proxy.tar ${registry_address}/kube-proxy:v1.26.15
crane push ./offline/images/pause3.6.tar ${registry_address}/pause:3.6
crane push ./offline/images/pause3.9.tar ${registry_address}/pause:3.9
crane push ./offline/images/kube-apiserver.tar ${registry_address}/kube-apiserver:v1.26.15
crane push ./offline/images/kube-controller-manager.tar \
  ${registry_address}/kube-controller-manager:v1.26.15
crane push ./offline/images/kube-scheduler.tar ${registry_address}/kube-scheduler:v1.26.15
crane push ./offline/images/etcd.tar ${registry_address}/etcd:3.5.10-0
crane push ./offline/images/cilium.tar ${registry_address}/cilium/cilium:v1.13.3
crane push ./offline/images/cilium-operator.tar \
  ${registry_address}/cilium/operator-generic:v1.13.3
crane push ./offline/images/ingress-nginx.tar \
  ${registry_address}/ingress-nginx/controller:v1.8.0
crane push ./offline/images/kube-webhook-certgen.tar \
  ${registry_address}/ingress-nginx/kube-webhook-certgen:v20230407
crane push ./offline/images/consul.tar ${registry_address}/consul:1.8.0
crane push ./offline/images/arangodb.tar ${registry_address}/arangodb:3.11.2
crane push ./offline/images/clickhouse.tar \
  ${registry_address}/clickhouse/clickhouse-server:23.3.8
crane push ./offline/images/postgres.tar ${registry_address}/postgres:12.15
crane push ./offline/images/rabbitmq.tar ${registry_address}/rabbitmq:3.11.18-management
crane push ./offline/images/redis.tar ${registry_address}/redis:7.0.11
crane push ./offline/images/victoria-metrics.tar \
  ${registry_address}/victoriametrics/victoria-metrics:v1.91.3
```

В случае если планируется HA инсталляция, импортировать дополнительные образы:

```
crane push ./offline/images/kube-vip.tar ${registry_address}/kube-vip/kube-vip:v0.5.0
```

```
crane push ./offline/images/postgres-operator.tar \
  ${registry_address}/acid/postgres-operator:v1.10.1
crane push ./offline/images/spilo.tar ${registry_address}/zalando/spilo-15:3.0-p1
crane push ./offline/images/clickhouse-operator.tar \
  ${registry_address}/altinity/clickhouse-operator:0.21.3
crane push ./offline/images/zookeeper.tar ${registry_address}/pravega/zookeeper:0.2.15
crane push ./offline/images/zookeper-operator.tar \
  ${registry_address}/pravega/zookeeper-operator:0.2.15
crane push ./offline/images/k8s-kubectl.tar \
  ${registry_address}/lachlanevenson/k8s-kubectl:v1.23.2
crane push ./offline/images/kube-arangodb.tar \
  ${registry_address}/arangodb/kube-arangodb:1.2.32
crane push ./offline/images/alpine.tar ${registry_address}/alpine:3.11
crane push ./offline/images/vmoperator.tar \
  ${registry_address}/victoriametrics/operator:v0.34.1
crane push ./offline/images/vminsert.tar \
  ${registry_address}/victoriametrics/vminsert:v1.91.3-cluster
crane push ./offline/images/vmselect.tar \
  ${registry_address}/victoriametrics/vmselect:v1.91.3-cluster
crane push ./offline/images/vmstorage.tar \
  ${registry_address}/victoriametrics/vmstorage:v1.91.3-cluster
crane push ./offline/images/redis-operator.tar \
  ${registry_address}/spotahome/redis-operator:v1.2.4
crane push ./offline/images/rabbitmq-operator.tar \
  ${registry_address}/rabbitmqoperator/cluster-operator:2.3.0
```

Импорт образов Monq

Выполнить временную конфигурацию контекста monqctl:

```
monqctl config set instance temp --server=http://registry.api.monq.local
monqctl config set credential temp --registry-token=000
monqctl config set releasehub monq-release-hub --token=000
monqctl config set context temp --instance=temp --credential=temp \
--releasehub=monq-release-hub
monqctl config use-context temp
```

Выполнить импорт образов контейнеров Monq в container registry:

```
monqctl release use-version 8.0.0 --product=installer \
--sourceDir=./offline/monq/installer/
monqctl release import-images --registryUri ${registry_address} \
--registryAuth=None
monqctl release use-version 3.13.8 --product=monq-registry \
--sourceDir=./offline/monq/registry/
monqctl release import-images --registryUri ${registry_address} \
--registryAuth=None
monqctl release use-version 8.0.0 --product=monq \
--sourceDir=./offline/monq/release/
monqctl release import-images --registryUri ${registry_address} \
--registryAuth=None
```

Выполнить удаление временной конфигурации monqctl:

```
rm ~/.monq/config.yml
rm -rf /tmp/monqctl
```

Последующую установку продолжить с раздела Инициализация Control Plane

Приложение. HA PostgreSQL.

Запуск с помощью kubernetes operator postgres-operator от zalando.. Обеспечение высокой доступности.

Выполнить установку оператора:

```
helm install postgres-operator offline/charts/postgres-operator-1.10.1.tgz \
  --namespace postgres-operator --create-namespace \
  --set image.registry=${registry_address}
```

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
```

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
```

Выполнить 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"
  dockerImage: ${registry_address}/zalando/spilo-15:3.0-p1
  volume:
    size: ${storage_size}
    storageClass: local-storage
    selector:
      matchLabels:
        service: postgres
  enableShmVolume: true
  numberOfInstances: 3
  postgresql:
    version: "12"
EOF
```

Увеличение кластера выполняется через редактирование CRD postgresql, поле `spec.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;

Получить автоматически сгенерированный пароль пользователя `postgres`:

```
postgres_password=$(kubectl get secret -n postgres-operator \
  postgres.monq-pg-cluster.credentials.postgresql.acid.zalan.do \
  -o 'jsonpath={.data.password}' | base64 -d)
echo "save it: ${postgres_password}"
```

Выполнить проверку:

1. Состояние запуска контейнеров:

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

2. Логи контейнеров на предмет ошибок:

   ```
   kubectl logs -n postgres-operator -l cluster-name=monq-pg-cluster
   ```

3. Возможность подключения с авторизационными данными:

   ```
   kubectl run utils -n ${monq_namespace} \
     --image="${registry_address}/utils:3.0" -- \
     /bin/bash -c -- "trap : TERM INT; sleep infinity & wait"

   kubectl exec -it -n ${monq_namespace} utils \
     -- psql -c "SHOW server_version;" \
     postgresql://postgres:${postgres_password}@monq-pg-cluster.postgres-operator
   ```

4. Распространение данных в кластере, чтение и запись. Создать тестовую таблицу и выполнить вставку:

   ```
   kubectl exec -it -n ${monq_namespace} utils \
     -- psql -c "CREATE TABLE public.test_replication (id int8);
     INSERT INTO public.test_replication (id) VALUES(1);" \
     postgresql://postgres:${postgres_password}@monq-pg-cluster.postgres-operator
   ```

   Выполнить запрос данных с реплики:

   ```
   kubectl exec -it -n ${monq_namespace} utils \
     -- psql -c "SELECT * FROM public.test_replication;" \
     postgresql://postgres:${postgres_password}@monq-pg-cluster-repl.postgres-operator
   ```

   Удалить тестовые данные:

   ```
   kubectl exec -it -n ${monq_namespace} utils \
     -- psql -c "DROP TABLE public.test_replication;" \
     postgresql://postgres:${postgres_password}@monq-pg-cluster.postgres-operator

   kubectl delete pod -n ${monq_namespace} utils
   ```

Для наполнения authfile выставить значения:

- postgresql.host: `"monq-pg-cluster.postgres-operator"`;
- 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 ${postgres_password}"` >.

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

Приложение. HA RabbitMQ.

Запуск с помощью kubernetes operator rabbitmq-operator. Обеспечение высокой доступности.

Выполнить установку оператора:

```
kubectl create -f offline/charts/cluster-operator.yml

kubectl set image deployment -n rabbitmq-system rabbitmq-cluster-operator \
 operator=${registry_address}/rabbitmqoperator/cluster-operator:2.3.0
```

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
```

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
```

Выполнить запуск кластера:

```
storage_size="2Gi"

cat <<EOF | kubectl apply -f -
apiVersion: rabbitmq.com/v1beta1
kind: RabbitmqCluster
metadata:
  name: monq-rabbitmq
  namespace: rabbitmq-system
spec:
  replicas: 3
  image: ${registry_address}/rabbitmq:3.11.18-management
  resources:
    requests:
      cpu: 500m
      memory: 200Mi
    limits:
      cpu: 1
      memory: 800Mi
  rabbitmq:
    additionalConfig: |
      vm_memory_high_watermark.relative = 0.9
  persistence:
    storageClassName: local-storage
    storage: "${storage_size}"
EOF
```

Кластер остается работоспособным при отказе одной ноды из трех;

Получить автоматически сгенерированные данные авторизации:

```
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)
echo "save username: ${rabbit_username}"
echo "save password: ${rabbit_password}"
```

Выполнить проверку:

1. Состояние запуска контейнеров:

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

2. Логи контейнеров на предмет ошибок:

```
kubectl logs -n rabbitmq-system -l app.kubernetes.io/name=monq-rabbitmq
```

3. Возможность подключения с авторизационными данными:

```
kubectl run utils -n ${monq_namespace} \
  --image="${registry_address}/utils:3.0" -- \
  /bin/bash -c -- "trap : TERM INT; sleep infinity & wait"

kubectl exec -it -n ${monq_namespace} utils \
  -- curl -u ${rabbit_username}:${rabbit_password} \
  monq-rabbitmq.rabbitmq-system:15672/api/overview | jq
```

4. Распространение данных в кластере: Создать очередь с типом quorum:

```
kubectl exec -it -n ${monq_namespace} utils \
  -- curl -X PUT -u ${rabbit_username}:${rabbit_password} \
  -H "Content-Type: application/json" \
  "monq-rabbitmq.rabbitmq-system:15672/api/queues/%2F/test_replication" \
  -d '{"auto_delete":false,"durable":true,"arguments":{"x-queue-type": "quorum"}}'
```

Проверить наличие очереди на всех репликах:

```
rabbit_ips=$(kubectl get po -n rabbitmq-system -l app.kubernetes.io/name=monq-rabbitmq \
 -o=jsonpath='{.items[*].status.podIP}')

for rabbit_ip in ${rabbit_ips}; do
  kubectl exec -it -n ${monq_namespace} utils \
    -- curl -u ${rabbit_username}:${rabbit_password} \
    "${rabbit_ip}:15672/api/queues/%2F" | jq
done
```

Удалить тестовые данные:

```
kubectl exec -it -n ${monq_namespace} utils \
  -- curl -X DELETE -u ${rabbit_username}:${rabbit_password} \
  "monq-rabbitmq.rabbitmq-system:15672/api/queues/%2F/test_replication" -v
kubectl delete pod -n ${monq_namespace} utils
```

Для наполнения authfile выставить значения:

- rabbitmq.host: `"monq-rabbitmq.rabbitmq-system"`;
- rabbitmq.port: 15672;
- rabbitmq.amqp_port: 5672;
- rabbitmq.amqp_ssl: **`false`**;
- 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. HA Arangodb.

Запуск с помощью kubernetes operator kube-arangodb. Обеспечение высокой доступности.

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.

Выполнить установку оператора:

```
helm install kube-arango offline/charts/kube-arangodb-1.2.32.tgz \
  --set "operator.features.storage=true" \
  --set "operator.replicaCount=1" \
  --create-namespace \
  --namespace arangodb-operator \
  --set operator.image=${registry_address}/arangodb/kube-arangodb:1.2.32 \
  --set operator.images.base=${registry_address}/alpine:3.11
```

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

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=
```

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
```

Выполнить запуск в ActiveFailover режиме:

```
arango_storage_size="3Gi"
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: ${arango_storage_size}
    storageClassName: arango-storage
    nodeSelector:
      arangodb: ""
  image: "${registry_address}/arangodb:3.11.2"
EOF
```

Проверить состояние кластера (запуск занимает некоторое время):

```
kubectl get po -n arangodb-operator
```

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

Сменить пароль пользователя с административными правами:

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

arangodb_svc_ip=$(kubectl get svc -n arangodb-operator monq-arangodb-cluster-ea \
  -o=jsonpath='{.spec.clusterIP}')

curl -X PATCH -H "Content-Type: application/json" -u "root:" \
  http://${arangodb_svc_ip}:8529/_api/user/root -d \
  "{\"passwd\":\"${arangodb_password}\"}"
```

Для определения точки подключения извне к web 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.

Выполнить проверку:

1. Состояние запуска контейнеров:

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

2. Логи контейнеров на предмет ошибок:

```
kubectl logs -n arangodb-operator -l arango_deployment=monq-arangodb-cluster
```

3. Возможность подключения с авторизационными данными:

```
kubectl run utils -n ${monq_namespace} \
  --image="${registry_address}/utils:3.0" -- \
  /bin/bash -c -- "trap : TERM INT; sleep infinity & wait"

kubectl exec -it -n ${monq_namespace} utils \
  -- curl -X POST -u root:${arangodb_password} \
  monq-arangodb-cluster.arangodb-operator:8529/_db/_system/_admin/echo \
  -d "string" | jq
```

4. Статус кластера:

```
kubectl exec -it -n ${monq_namespace} utils \
  -- curl -X GET -u root:${arangodb_password} \
  monq-arangodb-cluster.arangodb-operator:8529/_db/_system/_admin/cluster/health | jq

kubectl delete pod -n ${monq_namespace} utils
```

Для наполнения authfile выставить значения:

- arangodb.host: `"monq-arangodb-cluster.arangodb-operator"`;
- 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. HA Clickhouse.

Запуск с помощью kubernetes operator clickhouse-operator от altinity. Обеспечение высокой доступности. Для работы в режиме кластера clickhouse требует наличие zookeper.

Zookeeper

Запуск осуществляется с помощью оператора zookeeper-operator от pravega.

Выполнить установку оператора:

```
helm install zookeeper-operator offline/charts/zookeeper-operator-0.2.15.tgz \
 --namespace zookeeper-operator --create-namespace \
 --set image.repository=${registry_address}/pravega/zookeeper-operator \
 --set hooks.image.repository=${registry_address}/lachlanevenson/k8s-kubectl
```

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
```

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
```

Выполнить запуск zookeeper:

```
zk_storage_size="1Gi"

cat <<EOF | kubectl create -f -
apiVersion: "zookeeper.pravega.io/v1beta1"
kind: "ZookeeperCluster"
metadata:
  name: "zookeeper"
  namespace: "zookeeper-operator"
spec:
  image:
    repository: ${registry_address}/pravega/zookeeper
    tag: 0.2.15
  replicas: 3
  persistence:
    spec:
      storageClassName: local-storage
      resources:
        requests:
          storage: ${zk_storage_size}
EOF
```

Clickhouse

Выполнить установку оператора:

```
helm install altinity-clickhouse-operator \
  offline/charts/altinity-clickhouse-operator-0.21.3.tgz \
   --namespace clickhouse-operator --create-namespace \
   --set operator.image.repository=${registry_address}/altinity/clickhouse-operator \
   --set metrics.enabled=false
```

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
```

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
```

Выполнить запуск кластера.

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

```
ch_storage_size="5Gi"
clickhouse_password=$(openssl rand -base64 16)
clickhouse_password_hash=$(echo -n "${clickhouse_password}" | sha256sum | sed s/\ .*$//g)
echo save it: ${clickhouse_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: ${clickhouse_password_hash}
      admin/profile: default
      admin/quota: default
      admin/networks/ip: "0.0.0.0/0"
    settings:
      logger/level: warning
```

```
    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: ${registry_address}/clickhouse/clickhouse-server:23.3.8
EOF
```

Кластер остается работоспособным при одной ноде ch и двух zookeper.

Выполнить проверку:

1. Состояние запуска контейнеров:

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

2. Логи контейнеров на предмет ошибок:

```
kubectl logs -n clickhouse-operator -l clickhouse.altinity.com/chi=monq-clickhouse
```

3. Возможность подключения с авторизационными данными.

```
kubectl run utils -n ${monq_namespace} \
  --image="${registry_address}/utils:3.0" -- \
  /bin/bash -c -- "trap : TERM INT; sleep infinity & wait"

kubectl exec -it -n ${monq_namespace} utils \
  -- curl --get --data-urlencode "query=SELECT version()" \
  -u admin:${clickhouse_password} clickhouse-monq-clickhouse.clickhouse-operator:8123
```

4. Распространение данных в кластере, чтение и запись. Создать тестовую таблицу на первой реплике и выполнить вставку:

```
query="CREATE DATABASE test_replication ON CLUSTER monqch;"
kubectl exec -it -n ${monq_namespace} utils \
  -- curl -X POST -d "${query}" -u admin:${clickhouse_password} \
  chi-monq-clickhouse-monqch-0-0.clickhouse-operator:8123
```

```
query="CREATE TABLE test_replication.test ON CLUSTER monqch
(\`_id\` Int64, \`_date\` Date)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/test_replication/test',
'{replica}')
PARTITION BY _date ORDER BY (_date)
SETTINGS enable_mixed_granularity_parts = 1, index_granularity = 8192;"
kubectl exec -it -n ${monq_namespace} utils \
  -- curl -X POST -d "${query}" -u admin:${clickhouse_password} \
  chi-monq-clickhouse-monqch-0-0.clickhouse-operator:8123

query="INSERT INTO test_replication.test (\`_id\`, \`_date\`) VALUES(1, now());"
kubectl exec -it -n ${monq_namespace} utils \
  -- curl -X POST -d "${query}" -u admin:${clickhouse_password} \
  chi-monq-clickhouse-monqch-0-0.clickhouse-operator:8123
```

Выполнить запрос данных с других реплик:

```
query="SELECT * FROM test_replication.test;"
kubectl exec -it -n ${monq_namespace} utils \
  -- curl -X POST -d "${query}" -u admin:${clickhouse_password} \
  chi-monq-clickhouse-monqch-0-1.clickhouse-operator:8123

kubectl exec -it -n ${monq_namespace} utils \
  -- curl -X POST -d "${query}" -u admin:${clickhouse_password} \
  chi-monq-clickhouse-monqch-0-2.clickhouse-operator:8123
```

Удалить тестовые данные:

```
query="DROP DATABASE test_replication ON CLUSTER monqch;"
kubectl exec -it -n ${monq_namespace} utils \
  -- curl -X POST -d "${query}" -u admin:${clickhouse_password} \
  clickhouse-monq-clickhouse.clickhouse-operator:8123

kubectl delete pod -n ${monq_namespace} utils
```

Для наполнения authfile выставить значения:

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

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

Приложение. HA Redis.

Запуск с помощью kubernetes operator redis-operator от spotahome.. Обеспечение высокой доступности.

Выполнить установку оператора:

```
helm install redis-operator offline/charts/redis-operator-3.2.8.tgz \
--namespace redis-operator --create-namespace \
--set image.repository=${registry_address}/spotahome/redis-operator
```

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
```

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
```

Создать секрет с данными для авторизации:

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

kubectl create secret generic monq-redis-auth \
  -n redis-operator --from-literal=password="${redis_password}"
```

Выполнить 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: ${registry_address}/redis:7.0.11
    replicas: 3
  redis:
    image: ${registry_address}/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
```

Обнаружение master 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.

Выполнить проверку:

1. Состояние запуска контейнеров:

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

2. Логи контейнеров на предмет ошибок:

   ```
   kubectl logs -n redis-operator -l app.kubernetes.io/name=monq-redis
   ```

3. Возможность подключения с авторизационными данными и распространение данных в кластере:

   Записать значение:

   ```
   kubectl run utils -n ${monq_namespace} \
     --image="${registry_address}/utils:3.0" -- \
     /bin/bash -c -- "trap : TERM INT; sleep infinity & wait"

   master=$(kubectl exec -it -n ${monq_namespace} utils \
     -- redis-cli -h rfs-monq-redis.redis-operator -p 26379 --raw -d ' -p ' \
     SENTINEL get-master-addr-by-name mymaster)

   kubectl exec -it -n ${monq_namespace} utils \
     -- redis-cli -h ${master} -a "${redis_password}" \
     SET replicationTest "Works"
   ```

   Выполнить переключение master и прочитать значение:

   ```
   kubectl exec -it -n ${monq_namespace} utils \
     -- redis-cli -h rfs-monq-redis.redis-operator -p 26379 \
     SENTINEL FAILOVER mymaster

   master=$(kubectl exec -it -n ${monq_namespace} utils \
     -- redis-cli -h rfs-monq-redis.redis-operator -p 26379 --raw -d ' -p ' \
     SENTINEL get-master-addr-by-name mymaster)

   kubectl exec -it -n ${monq_namespace} utils \
     -- redis-cli -h ${master} -a "${redis_password}" \
     GET replicationTest
   ```

   Удалить тестовые данные:

   ```
   kubectl exec -it -n ${monq_namespace} utils \
     -- redis-cli -h ${master} -a "${redis_password}" \
     DEL replicationTest
   kubectl delete pod -n ${monq_namespace} utils
   ```

Для наполнения authfile выставить значения:

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

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

Приложение. HA VictoriaMetrics.

Запуск с помощью kubernetes operator victoria-metrics-operator.. Обеспечение высокой доступности.

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).

Оба режима поддерживаются в monq. В пошаговой инструкции будет рассмотрен режим с прямым обращением к `vminsert` и `vmselect`.

Выполнить установку оператора:

```
helm install vmoperator offline/charts/victoria-metrics-operator-0.23.1.tgz \
  -n victoria --create-namespace \
  --set image.repository=${registry_address}/victoriametrics/operator
```

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
```

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
```

Задать авторизационные данные пользователя

```
vm_user="monq"
vm_password=$(openssl rand -base64 16)
echo "save it: "${vm_password}

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
```

Выполнить запуск кластера:

```
storage_size="30Gi"

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

```
  namespace: victoria
spec:
  retentionPeriod: "1"
  replicationFactor: 3
  clusterVersion: v1.91.3-cluster
  vmstorage:
    image:
      repository: ${registry_address}/victoriametrics/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:
    image:
      repository: ${registry_address}/victoriametrics/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:
```

```
    image:
      repository: ${registry_address}/victoriametrics/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.

Выполнить проверку:

1. Состояние запуска контейнеров:

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

2. Логи контейнеров на предмет ошибок:

   ```
   kubectl logs -n victoria -l app.kubernetes.io/instance=monq
   ```

3. Возможность подключения и распространение данных в кластере. Выполнить запись данных:

   ```
   kubectl run utils -n ${monq_namespace} \
   ```

```
  --image="${registry_address}/utils:3.0" -- \
  /bin/bash -c -- "trap : TERM INT; sleep infinity & wait"

kubectl exec -it -n ${monq_namespace} utils \
  -- curl -d 'metric_name{foo="bar"} 123' -u ${vm_user}:${vm_password} \
  -X POST vminsert-monq.victoria:8480/insert/0/prometheus/api/v1/import/prometheus -v
```

Проверить кол-во datapoints записанных в каждый vmstorage:

```
vmstorage_ips=$(kubectl get po -n victoria -l app.kubernetes.io/name=vmstorage \
  -o=jsonpath='{.items[*].status.podIP}')
for vmstorage_ip in ${vmstorage_ips}; do
  curl -s ${vmstorage_ip}:8482/metrics | grep 'vm_rows_added_to_storage_total'
done
```

Выполнить запросы на чтение с vmselect:

```
kubectl delete pod -n ${monq_namespace} utils
```

Для наполнения authfile выставить значения:

- victoriametrics.cluster: `true`;
- victoriametrics.cluster_account: `"0"`;
- victoriametrics.auth_type: `"BasicAuth"`;
- victoriametrics.cluster_insert.host: `"vminsert-monq.victoria"`;
- victoriametrics.cluster_insert.port: `8480`;
- victoriametrics.cluster_insert.proto: `"http"`;
- victoriametrics.cluster_select.host: `"vmselect-monq.victoria"`;
- 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. Managed services.

В данном приложении отражен список требований для использования внешних сервисов для monq.

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 readWriteMany.

Если предоставленное в качестве сервиса решение удовлетворяет вышеуказанным требованиям, можно перейти к настройке кластера с последующим наполнением authfile, в противном случае следует воспользоваться ”SelfHosted” решением.

PostgreSQL

Список известных ограничений:

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

Проверить возможность:

- 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 администраторов;

- Подключения к порту СУБД с рабочих нод.

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

RabbitMQ

Выполнить проверки:

- Проверить доступность экземпляра СПО по заданному порту;
- Проверить возможность авторизации и выполнения основных операций: ”создание пользователя с ролью Administrator, создание очередей”;
- Подключения к порту RabbitMQ с рабочих нод.

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

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 наполнению authfile, в противном случае следует воспользоваться ”SelfHosted” решением.

Clickhouse

Clickhouse как сервис.

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

- Назначение пользователю прав `access_management`;
- 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`;

- Назначение ограничений на роль: `max_memory_usage_for_user, max_memory_usage_for_all, max_execution_time, timeout_before_checking_execution_speed`;
- 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`;
- Создание пользователей и назначение роли с вышеуказанными параметрами;
- Проверить возможность подключения к порту СУБД с рабочих нод.

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

Redis

Выполнить проверки:

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

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

Victoria metrics

Проверить возможность:

- Записи и чтения в бд;
- Подключения к порту СУБД с рабочих нод. VictoriaMetrics, запущенная в режиме HA 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 наполнению authfile, в противном случае следует воспользоваться ”SelfHosted” решением.

Приложение. Описание типового helm chart.

В целях упрощения запуска и унификации операций процесс запуска системного программного обеспечения реализован с помощью `helm`.

Пример установки postgres с помощью чарта:

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

helm install postgresql offline/charts/postgresql-1.0.1.tgz \
 --namespace ${infra_namespace} --create-namespace \
 --set application.image.registry=${registry_address} \
 --set application.ssl.enable=false
```

Список наиболее важных аргументов:

- --set application.image.registry=${registry_address} - выполнить установку из локального registry;
- --set application.ssl.enable=false - включить ssl в приложении;
- --set application.ssl.generateCert=false - сгенерировать новый сертификат с помощью CA из `application.ssl.ca.secret`;
- --set application.ssl.cert.secret.name - имя секрета содержащего сертификат, ключ и корневой сертификат;
- --set application.ssl.cert.secret.certName - ключ содержащий сертификат;
- --set application.ssl.cert.secret.keyName - ключ содержащий приватный ключ;
- --set application.ssl.ca.secret.name=monq-ca - имя секрета содержащего существующий CA сертификат;
- --set application.ssl.ca.secret.namespace=production - namespace секрета;
- --set application.ssl.ca.secret.certName=monq.ca.crt.pem - ключ с открытой частью сертификата;
- --set application.ssl.ca.secret.keyName=monq.ca.key.pem - ключ с закрытой частью сертификата;
- --set volume.localVolume.enable=true - при запуске приложения создать локальное хранилище, к которому в последствии будет примонтирован pvc;
- --set monitoring.enable=false - запустить компоненты для сбора метрик с приложения.

Список всех возможных переменных можно получить командой:

```
helm show values offline/charts/postgresql-1.0.1.tgz
```

Если планируется использовать SSL, то нужно учесть следующие моменты при наполнении `authfile` и проведении проверок:

- в consul меняется порт с 8500 на 8501 и протокол соответственно;
- в clickhouse меняется порт с 8123 на 8443 и протокол соответственно;
- в rabbitmq меняются порты: с 5672, 15672 на 5671 и 15671 и протокол соответственно;
- для остальных программ меняется протокол, см соответствующие значения ключей в `authfile`.

Приложение. Установка внутри кластера kubernetes.

Если компоненты СПО доступны только изнутри кластера kubernetes, например при развертывании операторами, то установка должна быть запущена внутри среды, из которой есть доступ ко всем компонентам СПО.

Это не совсем стандартный сценарий и его можно выполнить следующим образом: Запустить pod с установщиком внутри кластера:

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

cat <<EOF | kubectl create -f -
apiVersion: v1
kind: Pod
metadata:
  name: installer
  namespace: ${monq_namespace}
  labels:
    app: installer
spec:
  containers:
  - name: installer
    image: ${registry_address}/installer:8.0.0
    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 ${monq_namespace} -ti installer -- mkdir -p /opt/monq/
kubectl -n ${monq_namespace} cp /tmp/system_auth.json installer:/opt/monq/
```

Запустить установку:

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

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

Посмотреть ошибки, если таковые возникли:

```
kubectl exec -n ${monq_namespace} -ti installer -- cat /opt/monq/errors.json
```

Скопировать артефакты на локальную машину:

```
kubectl -n ${monq_namespace} cp installer:/opt/monq /tmp/monq
```

После окончания установки удалить pod:

```
kubectl delete po -n ${monq_namespace} installer
```

После установки система готова к использованию, интерфейс доступен по адресу `${global_domain}`.

Авторизационные данные по-умолчанию:

- логин: `admin@${global_domain}`
- пароль: `monq_admin`

Приложение. 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;
- api_domain - домен для внутреннего api, по умолчанию: `api.{{ global_domain }}`;
- registry_domain - домен для реестра микросервисов, по умолчанию: `registry.api.{{ global_domain }}`;
- 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: `ru-RU`;
- 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`**;
- generate_self_signed_cert - флаг генерации самоподписных сертификатов для Monq, по умолчанию: **`true`**;

- registry_secret_name - имя секрета .dockerconfig в namespace {{ monq_namespace }}, по умолчанию: `regsecret-monq`;
- create_management_users - флаг создания management пользователей, по умолчанию: **`true`**;
- no_log - флаг отключающий расширенное логирование во время установки, по умолчанию: **`false`**.

Приложение. Список используемых портов и протоколов.

Без HA

SRC	DST	Proto	Port	Description
All Servers	Ntp server	UDP	123	Синхронизация времени на хостах
All Servers	DNS server	UDP	53	Разрешение доменных имен на хостах
All k8s nodes	K8s Control plane	TCP	6443	Kubernetes API server
K8s Control plane	All k8s nodes	TCP	10250	Kubelet API
All k8s nodes	All k8s nodes	UDP	8472	Cilium/Flannel VXLAN overlay
All k8s nodes	All k8s nodes	TCP	4240	Cilium health checks
All k8s nodes	All k8s nodes	ICMP		Cilium health checks
All k8s nodes	All k8s nodes	UDP	8285	Flannel
All k8s nodes	DB Server	TCP	5000	container registry
All Servers	K8s worker	TCP	80,443	ingress-nginx-controller
K8s worker	DB Server	TCP	111,2049	NFS PV
K8s worker	DB Server	UDP	111,2049	NFS PV
Administrator PC	DB Server	TCP	8500	Consul HTTP Api
Administrator PC	K8s worker	TCP	80,443	ingress-nginx-controller
Administrator PC	DB Server	TCP	5432	Postgresql
Administrator PC	DB Server	TCP	8123	Clickhouse HTTP interface
Administrator PC	DB Server	TCP	8529	ArangoDB HTTP Api
Administrator PC	DB Server	TCP	8428	Victoria metrics HTTP Api
Administrator PC	DB Server	TCP	6379	Redis

SRC	DST	Proto	Port	Description
Administrator PC	DB Server	TCP	15672	RabbitMQ HTTP Api
Administrator PC	K8s Control plane	TCP	6443	Kubernetes API server

С HA

SRC	DST	Proto	Port	Description
All Servers	Ntp server	UDP	123	Синхронизация времени на хостах
All Servers	DNS server	UDP	53	Разрешение доменных имен на хостах
All k8s nodes	K8s Control plane	TCP	6443	Kubernetes API server
All k8s nodes	K8s Control plane VIP	TCP	6443	Kubernetes API server
K8s Control plane	K8s Control plane	TCP	2379-2380	Etcd server client API
K8s Control plane	All k8s nodes	TCP	10250	Kubelet API
All k8s nodes	All k8s nodes	UDP	8472	Cilium/Flannel VXLAN overlay
All k8s nodes	All k8s nodes	TCP	4240	Cilium health checks
All k8s nodes	All k8s nodes	ICMP		Cilium health checks
All k8s nodes	All k8s nodes	UDP	8285	Flannel
All k8s nodes	DB Server	TCP	5000	container registry
All Servers	K8s worker	TCP	80,443	ingress-nginx-controller
All Servers	Ingress VIP	TCP	80,443	ingress-nginx-controller
K8s worker	DB Server	TCP	111,2049	NFS PV
K8s worker	DB Server	UDP	111,2049	NFS PV
Administrator PC	DB Server	TCP	8500	Consul HTTP Api
Administrator PC	K8s worker	TCP	80,443	ingress-nginx-controller
Administrator PC	DB Server	TCP	5432	Postgresql
Administrator PC	DB Server	TCP	8123	Clickhouse HTTP interface
Administrator PC	DB Server	TCP	8529	ArangoDB HTTP Api
Administrator PC	DB Server	TCP	8480	VM insert HTTP Api

SRC	DST	Proto	Port	Description
Administrator PC	DB Server	TCP	8481	VM Select HTTP Api
Administrator PC	DB Server	TCP	6379	Redis
Administrator PC	DB Server	TCP	15672	RabbitMQ HTTP Api
Administrator PC	K8s Control plane	TCP	6443	Kubernetes API server

При запуске СУБД вне k8s(дополнительно):

SRC	DST	Proto	Port	Description
K8s worker	Postgresql Server	TCP	5432	Postgresql
K8s worker	Clickhouse Server	TCP	8123	Clickhouse HTTP interface
K8s worker	Arangodb Server	TCP	8529	ArangoDB HTTP Api
K8s worker	VictoriaMetrics Standalone Server	TCP	8428	Victoria metrics HTTP Api
K8s worker	VM Insert Server	TCP	8480	VM insert HTTP Api
K8s worker	VM Select Server	TCP	8481	VM select HTTP Api
K8s worker	Redis Server	TCP	6379	Redis
K8s worker	Redis Sentinel Server	TCP	26379	Redis Sentinel
K8s worker	Rabbitmq Server	TCP	15672	RabbitMQ HTTP Api
K8s worker	Rabbitmq Server	TCP	5672	RabbitMQ AMQP
Administrator PC	Postgresql Server	TCP	5432	Postgresql
Administrator PC	Clickhouse Server	TCP	8123	Clickhouse HTTP interface
Administrator PC	Arangodb Server	TCP	8529	ArangoDB HTTP Api
Administrator PC	VictoriaMetrics Standalone Server	TCP	8428	Victoria metrics HTTP Api
Administrator PC	VM Insert Server	TCP	8480	VM insert HTTP Api
Administrator PC	VM Select Server	TCP	8481	VM select HTTP Api
Administrator PC	Redis Server	TCP	6379	Redis
Administrator PC	Redis Sentinel Server	TCP	26379	Redis Sentinel
Administrator PC	Rabbitmq Server	TCP	15672	RabbitMQ HTTP Api

SRC	DST	Proto	Port	Description
Administrator PC	Rabbitmq Server	TCP	5672	RabbitMQ AMQP

Приложение. Кластерный DNS.

В случае отсутствия возможности конфигурации внешнего dns сервера для разрешения необходимых DNS имен имеется возможность настройки внутрикластерного DNS сервера - coredns.

> Внимание! Внесение изменений в конфигурацию coredns может повлечь за собой ошибки при обновлении kubernetes. Поэтому необходимо учитывать внесенные изменения при обновлении kubernetes Мы не рекомендуем использовать данный подход, однако в некоторых случаях он может быть единственным.

Разрешение реализовано с учетом развертывания согласно пошаговой инструкции, все компоненты развернуты в kubernetes. При использовании иной конфигурации размещения компонентов СПО, необходимо внести соответсвующие изменения. Поддерживаемые опции конфигурации описаны в официальной документации coredns.

> Внимание! Если развертывание компонентов СПО производится в kubernetes с помощью операторов, то в качестве точек подключения указываются соответствующие сервисы kubernetes.

Создать configmap `coredns-custom`:

```
infra_domain="in.monq.local"
infra_namespace="infra"
k8s_api_ip="<ip address kubernetes api server>"
global_domain="<monq domain name>"
svc_domain="${infra_namespace}.svc.cluster.local"

cat <<EOF | kubectl create -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns-custom
  namespace: kube-system
data:
  monq.server: |
    ${infra_domain}:53 {
      file /etc/coredns/custom/${infra_domain}.db
      kubernetes cluster.local {
        noendpoints
        namespaces default ${infra_namespace}
      }
      cache 30
```

```
    }
    ${global_domain}:53 {
      file /etc/coredns/custom/${global_domain}.db
      kubernetes cluster.local {
        noendpoints
        namespaces default ${infra_namespace}
      }
      cache 30
    }
  ${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.${infra_domain}.       IN  CNAME   arangodb.${svc_domain}.
    clickhouse.${infra_domain}.     IN  CNAME   clickhouse.${svc_domain}.
    postgresql.${infra_domain}.     IN  CNAME   postgresql.${svc_domain}.
    rabbitmq.${infra_domain}.       IN  CNAME   rabbitmq.${svc_domain}.
    redis.${infra_domain}.          IN  CNAME   redis.${svc_domain}.
    consul.${infra_domain}.         IN  CNAME   consul.${svc_domain}.
    victoriametrics.${infra_domain}. IN  CNAME   victoriametrics.${svc_domain}.
    registry.${infra_domain}.       IN  CNAME   registry.${svc_domain}.
    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
```

Добавить `coredns-custom` в основную конфигурацию coredns:

```
kubectl get cm -n kube-system coredns -o=jsonpath='{.data.Corefile}' > Corefile
echo "import /etc/coredns/custom/*.server" >> Corefile
kubectl create cm -n kube-system coredns --from-file=./Corefile -o yaml --dry-run=client \
| kubectl replace -f -
kubectl get deploy -n kube-system coredns -o json \
| jq '.spec.template.spec.volumes |= . + [{"configMap":{"defaultMode":420,
"name":"coredns-custom", "optional":true},"name":"custom-config-volume"}]' \
| jq '.spec.template.spec.containers[0].volumeMounts |= .
+ [{"mountPath":"/etc/coredns/custom",
"name":"custom-config-volume","readOnly":true}]' \
| kubectl replace -f -
kubectl rollout restart -n kube-system deploy coredns
```

Приложение. Собственные сертификаты SSL.

Список переменных, используемых по тексту:

- ${global_domain} - основное доменное имя развертываемого приложения
- ${infra_domain} - dns зона для инфраструктурных компонентов
- ${monq_namespace} - kubernetes namespace для запуска monq
- ${infra_namespace} - kubernetes namespace для запуска инфраструктурных компонентов
- ${ca_cert_name} - имя файла сертификата CA
- ${ca_key_name} - имя файла ключа сертификата CA

Данный раздел описывает следующие ситуации:

Собственный CA, генерация сертфикатов в установщике

В наличии только собственный CA сертификат, в составе открытого и закрытого ключа. Данным CA должны быть подписаны сертификаты для организации защищенного взаимодействия между компонентами ППО и СПО, клиентом и интерфейсом monq, а так же внутреннего API при межмикросервисном взаимодействии.

Сертификаты для container registry и непосредственно Monq будут сгенерированы и подписаны вручную с помощью openssl, а сертификаты для СПО выпускаются при деплое helm chart.

Предполагается, что CA сертификаты размещены в каталоге certs:

```
mkdir -p certs
cp ${ca_cert_name} ${ca_key_name} certs/
```

Добавить сертификат в список доверенных на хосте, данную операцию нужно выполнить на всех серверах:

```
mkdir -p /usr/share/ca-certificates/monq
cp certs/${ca_cert_name} /usr/share/ca-certificates/monq/${ca_cert_name}
echo "monq/${ca_cert_name}" >> /etc/ca-certificates.conf
update-ca-certificates
```

Сгенерировать сертификата для container registry:

```
mkdir certs/docker-registry
openssl req -new -nodes -out certs/docker-registry/registry.csr \
  -keyout certs/docker-registry/registry.key -subj "/CN=registry.${infra_domain}"
openssl x509 -req -in certs/docker-registry/registry.csr -days 3650 \
  -extfile <(printf "subjectAltName=DNS:registry.${infra_domain}") \
  -CA certs/${ca_cert_name} -CAkey certs/${ca_key_name} -CAcreateserial \
  -out certs/docker-registry/registry.crt
```

> Файлы certs/docker-registry/registry.key и certs/docker-registry/registry.crt необходимо перенести на сервер db, для последующей настройки container registry

Создать секрет содержащий CA сертификаты для переиспользования в остальных компонентах:

```
kubectl create secret generic -n ${monq_namespace} monq-ca-exist \
  --from-file=./certs/${ca_key_name} --from-file=./certs/${ca_cert_name}
```

Создать секрет содержащий CA сертификат для установки доверия к выписанным сертификатам:

```
kubectl create secret generic -n ${monq_namespace} monq-ca-certificates \
  --from-file=./certs/${ca_cert_name}
```

При развертывании СПО с помощью helm chart установить следующие значения переменных:

- `--set application.ssl.enable=true` - включить ssl в приложении;
- `--set application.ssl.generateCert=true` - сгенерировать новый сертификат с помощью CA из `application.ssl.ca.secret`;
- `--set application.ssl.ca.secret.name=monq-ca-exist` - имя секрета k8s содержащего существующий CA сертификат;
- `--set application.ssl.ca.secret.namespace=production` - namespace k8s в котором создан секрет;
- `--set application.ssl.ca.secret.certName=${ca_cert_name}` - ключ секрета с открытой частью сертификата;
- `--set application.ssl.ca.secret.keyName=${ca_key_name}` - ключ секрета с закрытой частью сертификата;

Сгенерировать сертификаты и секреты для интерфейсов Monq:

```
global_domain="< основное доменное имя развертываемого приложения >"

domain_list="${global_domain} api.${global_domain} registry.api.${global_domain}"

for domain in ${domain_list}; do
  mkdir certs/${domain}
  openssl req -new -nodes -out certs/${domain}/${domain}.csr \
    -keyout certs/${domain}/${domain}.key -subj "/CN=${domain}"

  openssl x509 -req -in certs/${domain}/${domain}.csr -days 3650 \
```

```
    -extfile <(printf "subjectAltName=DNS:${domain}") \
    -CA certs/${ca_cert_name} -CAkey certs/${ca_key_name} -CAcreateserial \
    -out certs/${domain}/${domain}.crt

  kubectl create secret tls -n ${monq_namespace} ${domain}-tls \
    --cert=certs/${domain}/${domain}.crt --key=certs/${domain}/${domain}.key
done
```

Выполнять запуск сценария установки с установленным флагом генерации самоподписных сертификатов для Monq в значении **`false`**:

```
generate_self_signed_cert=false
```

Собственный CA, сертификаты сгенерированы заранее

В наличии открытая часть собственного CA сертификата. Данным CA подписаны имеющиеся сертификаты для организации защищенного взаимодействия между компонентами ППО и СПО, клиентом и интерфейсом monq, а так же внутреннего API при межмикросервисном взаимодействии.

В составе:

- `exist.ca.crt` - открытая часть сертификата CA
- `registry.${infra_domain}` - открытая и закрытая часть сертификата для container registry
- `consul.${infra_domain}` - открытая и закрытая часть сертификата для consul
- `postgresql.${infra_domain}` - открытая и закрытая часть сертификата для postgresql
- `arangodb.${infra_domain}` - открытая и закрытая часть сертификата для arangodb
- `clickhouse.${infra_domain}` - открытая и закрытая часть сертификата для clickhouse
- `redis.${infra_domain}` - открытая и закрытая часть сертификата для redis
- `rabbitmq.${infra_domain}` - открытая и закрытая часть сертификата для rabbitmq
- `victoriametrics.${infra_domain}` - открытая и закрытая часть сертификата для victoriametrics
- `${global_domain}` - открытая и закрытая часть сертификата для основного домена
- `api.${global_domain}` - открытая и закрытая часть сертификата для внутреннего домена
- `registry.api.${global_domain}` - открытая и закрытая часть сертификата для реестра микросервисов

Предполагается, что все сертификаты размещены в каталоге certs, в соответствующих каталогах по имени сервиса, с учетом этого написаны дальнейшие команды.

Добавить сертификат в список доверенных на хосте, данную операцию нужно выполнить на всех серверах:

```
mkdir -p /usr/share/ca-certificates/monq
cp certs/${ca_cert_name} /usr/share/ca-certificates/monq/${ca_cert_name}
echo "monq/${ca_cert_name}" >> /etc/ca-certificates.conf
update-ca-certificates
```

Файлы `certs/docker-registry/registry.key` и `certs/docker-registry/registry.crt` необходимо перенести на сервер db, для последующей настройки container registry

Создать секрет содержащий CA сертификат для установки доверия к выписанным сертификатам:

```
kubectl create secret generic -n ${monq_namespace} monq-ca-certificates \
  --from-file=./certs/${ca_cert_name}
```

Создать секреты из сертификатов для СПО:

- Секрет для consul.${infra_domain}:

```
cp ./certs/${ca_cert_name} ./certs/consul/ca.crt
kubectl create secret generic -n ${infra_namespace} consul-certificates \
  --from-file=./certs/consul/
```

- Секрет для postgresql.${infra_domain}:

```
kubectl create secret generic -n ${infra_namespace} postgresql-certificates \
  --from-file=./certs/postgresql/
```

- Секрет для arangodb.${infra_domain}:

```
cat certs/arangodb/arangodb.crt certs/arangodb/arangodb.key > \
  certs/arangodb/arangodb.pem
kubectl create secret generic -n ${infra_namespace} arangodb-certificates \
  --from-file=certs/arangodb/arangodb.pem
```

- Секрет для clickhouse.${infra_domain}:

```
cp ./certs/${ca_cert_name} ./certs/clickhouse/ca.crt
kubectl create secret generic -n ${infra_namespace} clickhouse-certificates \
  --from-file=./certs/clickhouse/
```

- Секрет для redis.${infra_domain}:

```
cp ./certs/${ca_cert_name} ./certs/redis/ca.crt
kubectl create secret generic -n ${infra_namespace} redis-certificates \
  --from-file=./certs/redis/
```

- Секрет для rabbitmq.${infra_domain}:

```
cp ./certs/${ca_cert_name} ./certs/rabbitmq/ca.crt
kubectl create secret generic -n ${infra_namespace} rabbitmq-certificates \
  --from-file=./certs/rabbitmq/
```

- Секрет для victoriametrics.${infra_domain}:

```
kubectl create secret generic -n ${infra_namespace} victoriametrics-certificates \
  --from-file=./certs/victoriametrics/
```

При развертывании СПО с помощью helm chart установить следующие значения переменных:

- `--set application.ssl.enable=true` - включить ssl в приложении;
- `--set application.ssl.generateCert=false` - сгенерировать новый сертификат с помощью CA из `application.ssl.ca.secret`;
- `--set application.ssl.cert.secret.name` - имя секрета содержащего набор сертификатов;
- `--set application.ssl.cert.secret.certName` - ключ содержащий сертификат;
- `--set application.ssl.cert.secret.keyName` - ключ содержащий приватный ключ;
- `--set application.ssl.cert.secret.keyName` - ключ содержащий сертификат CA;

Для некоторых программ сертификаты в секретах могут скомбинированы или в конфигурации задействован сертификат CA, в таком случае набор переменных может отличаться, перед установкой helm chart следует уточнить набор переменных, для этого надо выполнить команду:

```
helm show values offline/charts/< название чарта >
```

Создать секреты с сертификатами с типом tls для интерфейсов Monq:

```
global_domain="< основное доменное имя развертываемого приложения >"

domain_list="${global_domain} api.${global_domain} registry.api.${global_domain}"

for domain in ${domain_list}; do
  kubectl create secret tls -n ${monq_namespace} ${domain}-tls \
    --cert=certs/${domain}/${domain}.crt --key=certs/${domain}/${domain}.key
done
```

Выполнять запуск сценария установки с установленным флагом генерации самоподписных сертификатов для Monq в значении **`false`**:

```
generate_self_signed_cert=false
```