

ЗАЩИЩЕННАЯ СИСТЕМА УПРАВЛЕНИЯ
БАЗАМИ ДАННЫХ «JATOVA»

Руководство по настройке. Часть 35.
Настройка и использование отказоустойчивого кластера
СУБД «Jatoba» в среде «Kubernetes»

643.72410666.00067-07 98 01-35

Листов 80

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

АННОТАЦИЯ

В документе приведены сведения, описывающие работу оператора отказоустойчивого кластера CloudNativePG с СУБД «Jatoba» в кластере Kubernetes.

Степени важности примечаний, применяемые в документе:



Важная информация – указания, требующие особого внимания



Дополнительная информация – указания, позволяющие упростить работу с изделием



Для СУБД «Jatoba» версии ядра 6 используется версия — 6.10.1-57608-dfl.6.3

СОДЕРЖАНИЕ

1. Назначение компонента.....	6
1.1. Архитектура кластера CloudNativePG	8
2. Требования к ПО.....	9
3. Установка Kubernetes кластера на основе k3s.....	10
3.1. Требования к аппаратному обеспечению	10
3.2. Установка служебных программ для работы с кластером и чартами	10
3.3. Установка хоста с ролью «сервер» (мастер узел).....	10
3.4. Установка хоста с ролью «агент» (рабочий узел)	12
3.5. Проверка статуса корректной инсталляции кластера	13
4. Удаление Kubernetes кластера на основе k3s	15
5. Комплект оператора.....	16
6. Подготовка к установке оператора и кластера.....	17
6.1. Загрузка Docker-образов Jatoba CNPG в локальный docker registry на мастер и воркер узлах кластера k3s.....	17
6.2. Распаковка архива с helm-чартами	18
6.3. Установка дополнительных компонент	18
6.3.1. Установка MinIO сервиса для хранения бекапов	18
6.3.2. Установка SSL сертификатов для доступа к удаленному docker registry	19
6.3.3. Установка monitoring сервисов Prometheus и Grafana.....	20
6.3.4. Установка cert manager.....	21
7. Установка оператора.....	23
7.1. Параметры развертывания	23
7.2. Установка оператора	23
7.3. Установка кластера	25
8. Удаление оператора и кластера.....	26
9. Основные операции с кластером	27
9.1. Просмотр ресурсов кластера или оператора в кластере Kubernetes.	27
9.2. Просмотр журналов кластера и оператора	28
9.3. Подключение к кластеру напрямую (локально в поде СУБД)	28
9.4. Подключение к кластеру из клиента (отдельный под)	29
10. Изменение пользовательских настроек при установке кластера.....	31
10.1. Блок конфигурации bootstrap.....	31
10.2. Блок конфигурации backups.....	32
10.3. Блок конфигурации recovery.....	34
10.4. Блок конфигурации parameters	36
10.5. Блок конфигурации poolers	36
10.6. Блок конфигурации monitoring	37

10.7. Блок конфигурации ограничений ресурсов кластера.....	37
10.8. Блок конфигурации сертификатов.....	39
11. Дополнительные операции с кластером.....	41
11.1. Создание реплики кластера.....	41
11.2. Резервное копирование через VolumeSnapshot	42
11.2.1. Настройка механизма VolumeSnapshot	43
11.2.2. Настройка резервного копирования через VolumeSnapshot.....	44
11.2.3. Настройка восстановления кластера через VolumeSnapshot.....	46
11.3. Основные команды плагина CNPG для kubectl.....	47
11.4. Гибернация.....	48
11.4.1. Императивная гибернация.....	49
11.4.2. Декларативная гибернация.....	50
Приложение 1	51
Установка с предварительной загрузкой компонент K3s.....	51
Приложение 2	53
Файл конфигурации кластера ./values/jatoba-default-values.yaml	53
Приложение 3	62
Файл конфигурации кластера ./ values/values-opt/jatoba-bootstrap.yaml.....	62
Приложение 4	63
Файл конфигурации кластера ./ values/values-opt/jatoba-backup.yaml.....	63
Приложение 5	64
Файл конфигурации кластера ./ values/values-opt/jatoba-recovery.yaml.....	64
Приложение 6	65
Файл конфигурации кластера ./ values/values-opt/jatoba-parameters.yaml	65
Приложение 7	66
Файл конфигурации кластера ./ values/values-opt/jatoba-poolers.yaml.....	66
Приложение 8	67
Файл конфигурации кластера ./ values/values-opt/monitoring.yaml.....	67
Приложение 9	68
Файл конфигурации кластера jatoba-resources-minimal.yaml	68
Файл конфигурации кластера jatoba-resources-recommend.yaml	69
Файл конфигурации кластера jatoba-resources-large.yaml	70
Приложение 10	71
Файл конфигурации кластера jatoba-custom-certs.yaml	71
Файл конфигурации кластера jatoba-custom-certs-poolers.yaml.....	71
Приложение 11	73
Манифест для создания реплики на основе pg_basebackup.....	73
Приложение 12	75
Файл настройки ресурсов Longhorn csi provider backuptarget.yaml.....	75

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

Файл конфигурации создания бэкапа кластера с использованием механизма VolumeSnapshot cluster-backup.yaml	75
Файл конфигурации восстановления кластера из бэкапа, созданного с использованием механизма VolumeSnapshot cluster-recovery.yaml	77
Термины и определения	78
Перечень сокращений.....	79

1. НАЗНАЧЕНИЕ КОМПОНЕНТА

СУБД «Jatoba» может применяться в составе продукта CloudNativePG, позволяющего разворачивать предварительно сконфигурированный отказоустойчивый кластер в окружении Kubernetes. Текущая реализация кластера CNPG использует физическую потоковую репликацию.

В руководстве приведено описание настройки продукта CloudNativePG с использованием СУБД Jatoba для кластера Kubernetes.

В руководстве настройка и работа продукта CloudNativePG с СУБД Jatoba рассматривается на примере кластера Kubernetes на базе продукта K3s для минимизации конфигурации и возможности легкой настройки.



В данном руководстве не рассматриваются разворачивание и лучшие практики для настройки и разворачивания кластера Kubernetes

Кластер K3s состоит из двух узлов: один главный и один резервный. Количество узлов кластера не влияет на разворачиваемый продукт CloudNativePG, но следует учитывать, что для обеспечения минимальной отказоустойчивости необходимо не менее двух узлов.

Схематично физическая структура узлов кластера показана на рисунке 1.1.

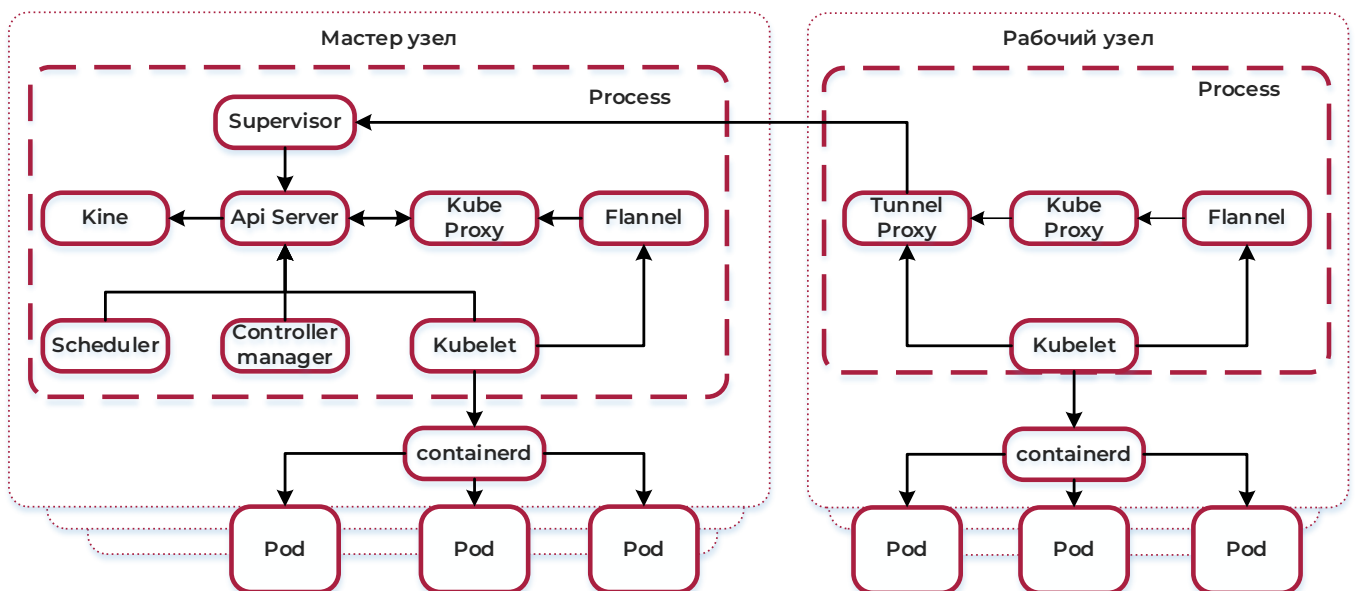


Рисунок 1.1 – Физическая структура узлов кластера K3s.

Основные понятия, используемые на рисунке 1.1:

Мастер узел	Основной узел кластера, к нему подключаются рабочие узлы, обычно не используется для размещения рабочей нагрузки
Рабочий узел	Рабочий узел кластера, развертываемые в кластере приложения размещаются на них
Supervisor	Основной элемент управления процессами в K3s, отвечает за поддержку и работоспособность остальных компонент кластера
Kine	Альтернативное хранилище для K3s, используется вместо etcd, транслируя API для etcd в виде данных из базы данных (sqlite/postgres/mysql).
Api Server	Компонент плоскости управления (control plane) предоставляет и взаимодействует с API Kubernetes.
Kube Proxy	Управляет сетевыми правилами на всех узлах кластера
Flannel	Компонент обеспечения сетевой доступности между узлами в кластере, управляет подсетями на узлах
Scheduler	Планировщик нагрузки на узлах, отвечает за оценку ресурсов на узлах, тем самым определяя где и какие сервисы будут запущены
Controller manager	Компонент управления контроллерами, отвечающими за внутренние процессы внутри кластера
Kubelet	Агент запускаемый на каждом узле кластера и обеспечивающий работоспособность контейнеров
Containerd	Среда запуска контейнеров в кластере, отвечает за жизненный цикл контейнеров (Pod) и хранение и загрузку образов на узлах
Tunnel proxy	Обеспечивает безопасное соединение между мастер узлом и рабочими узлами кластера

Данные компоненты характерны только для кластеров на базе K3s, основные отличия от K8s в том, что K3s использует упакованный набор тех же бинарных файлов, запускаемых как единое целое в виде одной службы на хосте.

1.1. Архитектура кластера CloudNativePG

Кластер СУБД «Jatoba» устанавливаемый средствами CloudNativePG имеет стандартные способы взаимодействия через строки подключения по имени хоста и порту. Порт по умолчанию 5432.

Управление кластером оператор осуществляет через связи с каждым отдельным экземпляром кластера – первичным узлом и репликами. Оператор получает сведения о состоянии экземпляра обращаясь на порт 8000, это основной порт взаимодействия с экземплярами кластера.

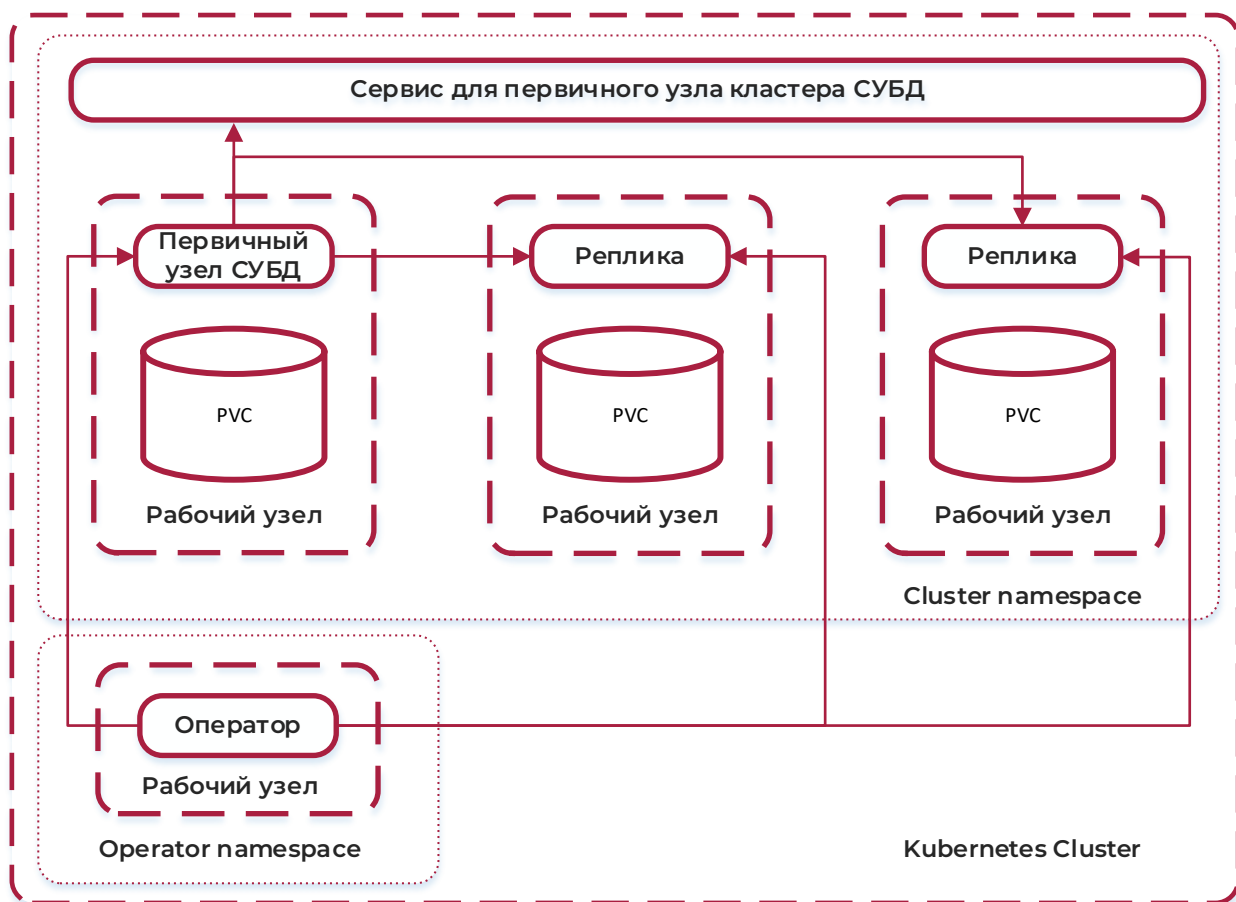


Рисунок 1.2 – Архитектура кластера, устанавливаемого с помощью CloudNativePG.

2. ТРЕБОВАНИЯ К ПО

Оператор отказоустойчивого кластера на основе CloudNativePG может быть развернут только в среде Kubernetes или аналогичных ей. Список версий оператора и поддерживаемых версий Kubernetes приведен в таблице 2.1.

Таблица 2.1 – Перечень соответствий поддерживаемых версий оператора с версиями Kubernetes.

Версия	Поддержка	Дата выпуска	Окончание поддержки	Поддерживаемые версии Kubernetes	Поддерживаемые версии Jatoba
1.27.x	Да	12.08.2025	февраль 2026	1.31, 1.32, 1.33	6.10.1
1.25.x	Да	23.12.2024	Май/Июнь 2025	1.29, 1.30, 1.31, 1.32	6.6.1, 6.8.1, 6.9.1, 6.10.1
main	Нет, только разработка	X	X	—	—

Docker-образ СУБД «Jatoba» создан на основе РЕД ОС 7.3.

Для использования оператора CloudNativePG необходимо иметь утилиты командной строки и развернутый кластер Kubernetes в соответствии с документацией на продукт. Альтернативным вариантом может служить развернутый кластер на K3s или аналогичных средах, менее требовательных к ресурсам и компонентам. Инструкции по разворачиванию можно найти на сайтах производителя продукта:

- Kubernetes — <https://kubernetes.io/docs/setup>,
- K3s — <https://docs.K3s.io/installation>,
- helm - <https://helm.sh/docs/intro/install>,
- kubectl - <https://kubernetes.io/docs/tasks/tools>.

Для разворачивания среды для тестирования и разработки можно развернуть K3s в минимальной конфигурации.

3. УСТАНОВКА KUBERNETES КЛАСТЕРА НА ОСНОВЕ K3S

Большая часть команд для установки должна выполняться с повышенными привилегиями, при необходимости переключитесь на пользователя root (sudo -i) или используйте sudo.

3.1. Требования к аппаратному обеспечению

Для запуска кластера Jatoba CNPG на основе кластера k3s необходимы 2 ВМ со следующими минимальными параметрами ([рекомендации](#)):

- 2 vCPU (master) / 1 vCPU (worker);
- 4 GB RAM (master) / 2 GB RAM (worker);
- 30 GB пространства на диске (на обеих машинах).

Данных ресурсов будет достаточно для запуска кластера в тестовом режиме, без нагрузки.



В документации приведен пример установки k3s кластера на Linux операционную систему – Ubuntu 22.04

3.2. Установка служебных программ для работы с кластером и чартами

Установка kubectl:

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"  
sudo cp kubectl /usr/local/bin/kubectl  
sudo chmod +x /usr/local/bin/kubectl
```

Установка helm:

```
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3  
chmod 700 get_helm.sh  
./get_helm.sh
```

Проверка установленных программ:

```
kubectl version  
helm version
```

3.3. Установка хоста с ролью «сервер» (мастер узел)

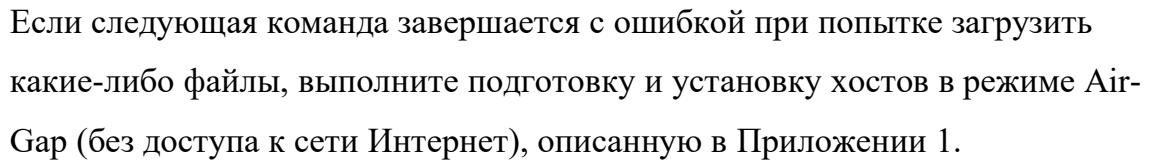
Необходимо сгенерировать токен для первой установки:

```
K3S_SERVER_TOKEN=$(head -n 5 /dev/urandom | base64 | tr -dc 'a-zA-Z0-9' | head -c 25)
```

```
juser@u602doc-k3s-01:~$ K3S_SERVER_TOKEN=$(head -5 /dev/urandom | base64 | tr -dc 'a-zA-Z0-9' | head -c 25)
```

Рисунок 3.1 – Генерация токена

Установка «сервера»:



```
curl -sfL https://get.k3s.io | INSTALL_K3S_VERSION=v1.32.5+k3s1 INSTALL_K3S_EXEC="server" K3S_TOKEN=$K3S_SERVER_TOKEN sh -s -
```

```
[user@u602doc-k3s-01:~$ curl -sfl https://get.k3s.io | INSTALL_K3S_EXEC="server" K3S_TOKEN=$K3S_SERVER_TOKEN sh -s -
[INFO] Finding release for channel stable
[INFO] Using v1.32.4+k3s1 as release
[INFO] Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.32.4+k3s1/sha256sum-amd64.txt
[INFO] Downloading binary https://github.com/k3s-io/k3s/releases/download/v1.32.4+k3s1/k3s
[INFO] Verifying binary download
[INFO] Installing k3s to /usr/local/bin/k3s
[INFO] Skipping installation of SELinux RPM
[INFO] Creating /usr/local/bin/kubectrl symlink to k3s
[INFO] Creating /usr/local/bin/crictl symlink to k3s
[INFO] Creating /usr/local/bin/ctr symlink to k3s
[INFO] Creating killall script /usr/local/bin/k3s-killall.sh
[INFO] Creating uninstall script /usr/local/bin/k3s-uninstall.sh
[INFO] env: Creating environment file /etc/systemd/system/k3s.service.env
[INFO] systemd: Creating service file /etc/systemd/system/k3s.service
[INFO] systemd: Enabling k3s unit
Created symlink /etc/systemd/system/multi-user.target.wants/k3s.service → /etc/systemd/system/k3s.service.
[INFO] systemd: Starting k3s
```

Рисунок 3.2 – Установка сервера

Проверить, что служба запустилась:

```
systemctl status k3s.service
```

```
juser@602doc-k3s-01:~$ systemctl status k3s.service
● k3s.service - Lightweight Kubernetes
   Loaded: loaded (/etc/systemd/system/k3s.service; enabled); vendor preset: enabled)
   Active: active (running) since Wed 2025-05-07 13:49:10 MSK; 3min 15s ago
     Docs: https://k3s.io
 Process: 503837 ExecStartPre=/bin/sh -xc ! /usr/bin/systemctl is-enabled --quiet nm-cloud-setup.service 2>/dev/null (code=exited, status=0/SUCCESS)
 Process: 503839 ExecStartPre=/sbin/modprobe br_netfilter (code=exited, status=0/SUCCESS)
 Process: 503840 ExecStartPre=/sbin/modprobe overlay (code=exited, status=0/SUCCESS)
 Main PID: 503841 (k3s-server)
    Tasks: 99
   Memory: 1.4G
      CPU: 50.145s
   CGroup: /system.slice/k3s.service
           └─503841 "/usr/local/bin/k3s server"
             └─503893 "containerd ..."
               └─504940 /var/lib/rancher/k3s/data/e1730ceee3d97d63f58b7ccd96fe08638e972abfd3b1ebdf497b52572f85b316/bin/containerd-sys
                 └─504944 /var/lib/rancher/k3s/data/e1730ceee3d97d63f58b7ccd96fe08638e972abfd3b1ebdf497b52572f85b316/bin/containerd-sys
                   └─504947 /var/lib/rancher/k3s/data/e1730ceee3d97d63f58b7ccd96fe08638e972abfd3b1ebdf497b52572f85b316/bin/containerd-sys
                     └─505918 /var/lib/rancher/k3s/data/e1730ceee3d97d63f58b7ccd96fe08638e972abfd3b1ebdf497b52572f85b316/bin/containerd-sys
                       └─506070 /var/lib/rancher/k3s/data/e1730ceee3d97d63f58b7ccd96fe08638e972abfd3b1ebdf497b52572f85b316/bin/containerd-sys
```

Lines 1-19/19 (END)

Рисунок 3.3 – Проверка статуса службы

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм.: _____
--------------------	--------------------------	---------------------------

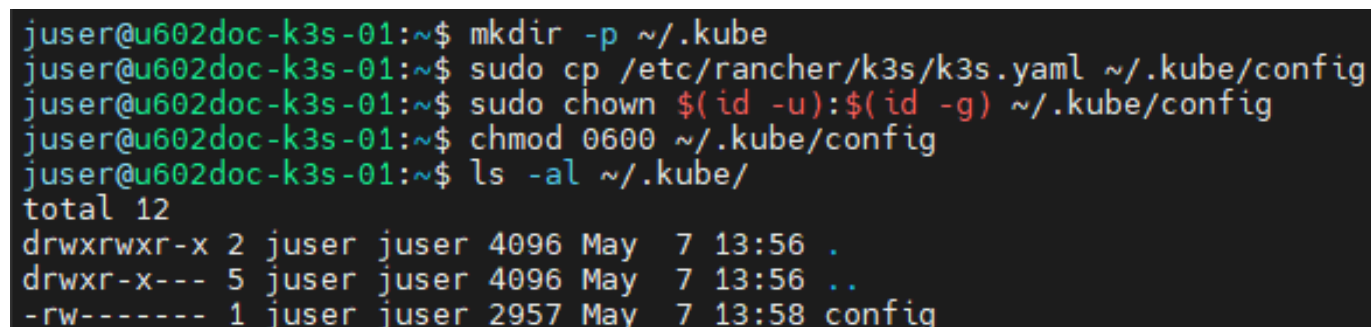
В результате будет получен файл конфигурации для пользователя, которому необходим доступ к развернутому кластеру. Команды выполняются в контексте выбранного вами пользователя через команду ОС «sudo» (требуется соответствующий доступ).

Скопировать файл конфигурации для доступа к кластеру в профиль, меняем владельца файла и добавляем необходимые разрешения:

```
mkdir -p ~/.kube
sudo cp /etc/rancher/k3s/k3s.yaml ~/.kube/config
sudo chown $(id -u):$(id -g) ~/.kube/config
chmod 0600 ~/.kube/config
```

Добавить экспорт переменной, указывающей на место хранения файла конфигурации для доступа к k3s в .bashrc пользователя, для того, чтобы при логине эта переменная автоматически подгружалась, и сразу же подгружаем файл .bashrc, чтобы изменения применились в текущей SSH-сессии:

```
echo "export KUBECONFIG=~/.kube/config" >> ~/.bashrc
source ~/.bashrc
```



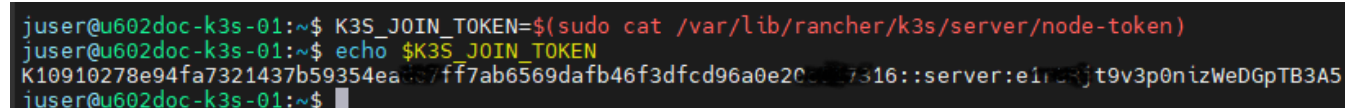
```
juser@u602doc-k3s-01:~$ mkdir -p ~/.kube
juser@u602doc-k3s-01:~$ sudo cp /etc/rancher/k3s/k3s.yaml ~/.kube/config
juser@u602doc-k3s-01:~$ sudo chown $(id -u):$(id -g) ~/.kube/config
juser@u602doc-k3s-01:~$ chmod 0600 ~/.kube/config
juser@u602doc-k3s-01:~$ ls -al ~/.kube/
total 12
drwxrwxr-x 2 juser juser 4096 May  7 13:56 .
drwxr-x--- 5 juser juser 4096 May  7 13:56 ..
-rw----- 1 juser juser 2957 May  7 13:58 config
```

Рисунок 3.4 – Экспорт переменной

3.4. Установка хоста с ролью «агент» (рабочий узел)

Получить на Мастер-узле значение переменной K3S_JOIN_TOKEN (токен для подключения рабочих узлов):

```
K3S_JOIN_TOKEN=$(sudo cat /var/lib/rancher/k3s/server/node-token)
echo $K3S_JOIN_TOKEN
```



```
juser@u602doc-k3s-01:~$ K3S_JOIN_TOKEN=$(sudo cat /var/lib/rancher/k3s/server/node-token)
juser@u602doc-k3s-01:~$ echo $K3S_JOIN_TOKEN
K10910278e94fa7321437b59354eae57ff7ab6569dafb46f3dfcd96a0e20a27316::server:e1nc3jt9v3p0nizWeDGPtB3A5
juser@u602doc-k3s-01:~$
```

Рисунок 3.5 – Получение переменной K3S_JOIN_TOKEN



```
export K3S_JOIN_TOKEN=<значение K3S_JOIN_TOKEN полученного на мастер уз  
ле>  
curl -sfL https://get.k3s.io | INSTALL_K3S_VERSION=v1.32.5+k3s1 INSTALL  
_K3S_EXEC="agent --server https://<IP_master_node>:6443" K3S_TOKEN=$K3S  
_JOIN_TOKEN sh -s -  
unset K3S JOIN TOKEN
```

```
juser@u602doc-k3s-02:~$ K3S_JOIN_TOKEN=K10910278e94fa7321437b5935...ff7ab6569dafb46f3dfcd96a0e20...16::server:ef...t9v3p0niz
WeDgP TB3A5
juser@u602doc-k3s-02:~$ curl -sfl https://get.k3s.io | INSTALL_K3S_EXEC="agent --server https://10.116.102.64:6443" K3S_TOKEN=$K3S_JOIN_TOKEN sh -s -
[INFO] Finding release for channel stable
[INFO] Using v1.32.4+k3s1 as release
[INFO] Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.32.4+k3s1/sha256sum-amd64.txt
[INFO] Downloading binary https://github.com/k3s-io/k3s/releases/download/v1.32.4+k3s1/k3s
[INFO] Verifying binary download
[INFO] Installing k3s to /usr/local/bin/k3s
[INFO] Skipping installation of SELinux RPM
[INFO] Creating /usr/local/bin/kubectrl symlink to k3s
[INFO] Creating /usr/local/bin/crictl symlink to k3s
[INFO] Creating /usr/local/bin/ctr symlink to k3s
[INFO] Creating killall script /usr/local/bin/k3s-killall.sh
[INFO] Creating uninstall script /usr/local/bin/k3s-agent-uninstall.sh
[INFO] env: Creating environment file /etc/systemd/system/k3s-agent.service.env
[INFO] systemd: Creating service file /etc/systemd/system/k3s-agent.service
[INFO] systemd: Enabling k3s-agent unit
Created symlink /etc/systemd/system/multi-user.target.wants/k3s-agent.service → /etc/systemd/system/k3s-agent.service.
[INFO] systemd: Starting k3s-agent
```

Рисунок 3.6 – Запуск скрипта инсталляции рабочего узла

Проверить, что служба k3s-agent запущена:

```
systemctl status k3s-agent.service
```

```
juser@u002doc-k3s-02:~$ systemctl status k3s-agent.service
● k3s-agent.service - Lightweight Kubernetes
   Loaded: loaded (/etc/systemd/system/k3s-agent.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2025-05-07 14:24:27 MSK; 1min 58s ago
     Docs: https://k3s.io
  Process: 1390817 ExecStartPre=/bin/sh -xc ! /usr/bin/systemctl is-enabled --quiet nm-cloud-setup.service 2>/dev/null (code=ex
  Process: 1390819 ExecStartPre=/sbin/modprobe br_netfilter (code=exited, status=0/SUCCESS)
  Process: 1390820 ExecStartPre=/sbin/modprobe overlay (code=exited, status=0/SUCCESS)
 Main PID: 1390821 (k3s-agent)
    Tasks: 36
   Memory: 328.5M
      CPU: 6.291s
  CGroup: /system.slice/k3s-agent.service
          └─1390821 "/usr/local/bin/k3s agent"
          └─1390865 "containerd"
          └─1391508 /var/lib/rancher/k3s/data/e1730ceee3d97d63f58b7ccd96fe08638e972abfd3b1ebdf497b52572f85b316/bin/containerd-
```

Рисунок 3.7— Проверка статуса службы k3s-agent

3.5. Проверка статуса корректной инсталляции кластера

На мастер узле, проверить статус кластера. Кластер должен состоять из двух узлов:

```
kubectl get nodes
```

```
juser@u602doc-k3s-01:~$ sudo kubectl get nodes
NAME                STATUS    ROLES                  AGE   VERSION
u602doc-k3s-01      Ready     control-plane,master   91m   v1.32.4+k3s1
u602doc-k3s-02      Ready     <none>                 56m   v1.32.4+k3s1
```

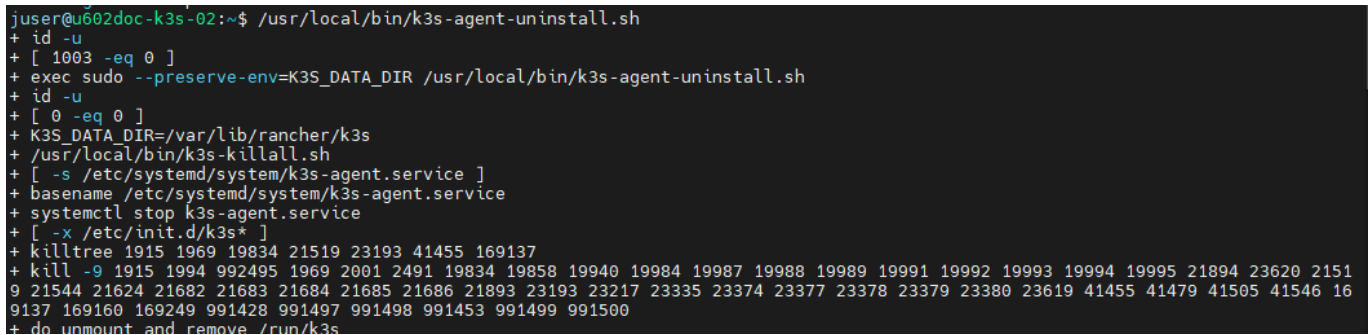
Рисунок 3.8 – Проверка состояния кластера

4. УДАЛЕНИЕ KUBERNETES КЛАСТЕРА НА ОСНОВЕ K3S

Для удаления кластера Kubernetes продукт k3s имеет готовые скрипты, включенные в поставку при установке. После их выполнения можно произвести «чистую» установку повторно.

На рабочем узле необходимо запустить с повышенными привилегиями или от пользователя «root» скрипт удаления агента k3s кластера:

```
sudo /usr/local/bin/k3s-agent-uninstall.sh
```

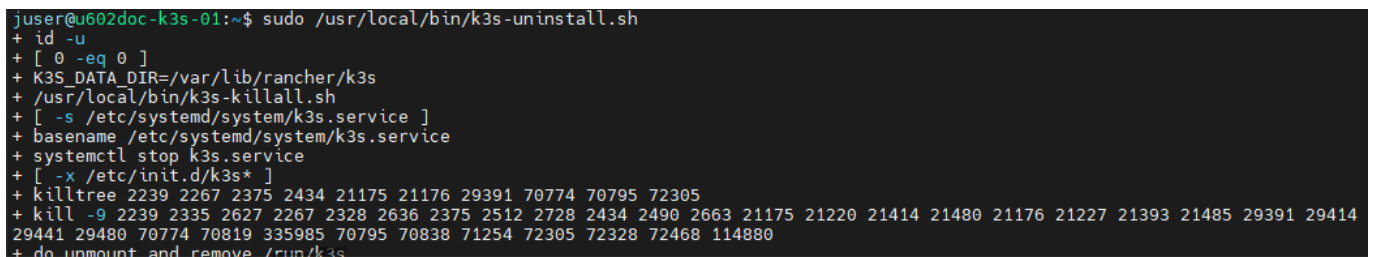


```
juser@u602doc-k3s-02:~$ /usr/local/bin/k3s-agent-uninstall.sh
+ id -u
+ [ 1003 -eq 0 ]
+ exec sudo --preserve-env=K3S_DATA_DIR /usr/local/bin/k3s-agent-uninstall.sh
+ id -u
+ [ 0 -eq 0 ]
+ K3S_DATA_DIR=/var/lib/rancher/k3s
+ /usr/local/bin/k3s-killall.sh
+ [ -s /etc/systemd/system/k3s-agent.service ]
+ basename /etc/systemd/system/k3s-agent.service
+ systemctl stop k3s-agent.service
+ [ -x /etc/init.d/k3s* ]
+ killtree 1915 1969 19834 21519 23193 41455 169137
+ kill -9 1915 1994 992495 1969 2001 2491 19834 19858 19940 19984 19987 19988 19989 19991 19992 19993 19994 19995 21894 23620 2151
9 21544 21624 21682 21683 21684 21685 21686 21893 23193 23217 23335 23374 23377 23378 23379 23380 23619 41455 41479 41505 41546 16
9137 169160 169249 991428 991497 991498 991453 991499 991500
+ do umount and remove /run/k3s
```

Рисунок 4.1 - Скрипт удаления агента k3s кластера

На мастер-узле необходимо запустить с повышенными привилегиями или от пользователя «root» скрипт удаления k3s кластера:

```
sudo /usr/local/bin/k3s-uninstall.sh
```



```
juser@u602doc-k3s-01:~$ sudo /usr/local/bin/k3s-uninstall.sh
+ id -u
+ [ 0 -eq 0 ]
+ K3S_DATA_DIR=/var/lib/rancher/k3s
+ /usr/local/bin/k3s-killall.sh
+ [ -s /etc/systemd/system/k3s.service ]
+ basename /etc/systemd/system/k3s.service
+ systemctl stop k3s.service
+ [ -x /etc/init.d/k3s* ]
+ killtree 2239 2267 2375 2434 21175 21176 29391 70774 70795 72305
+ kill -9 2239 2335 2627 2267 2328 2636 2375 2512 2728 2434 2490 2663 21175 21220 21414 21480 21176 21227 21393 21485 29391 29414
29441 29480 70774 70819 335985 70795 70838 71254 72305 72328 72468 114880
+ do umount and remove /run/k3s
```

Рисунок 4.2 - Скрипт удаления k3s кластера

5. КОМПЛЕКТ ОПЕРАТОРА

Оператор CloudNativePG с СУБД «Jatoba» распространяется в виде контейнера и поставляемых оператором helm-чартов для установки кластера.

Таблица 5.1 – Описание файлов комплекта контейнера

Название документа	Описание
jatoba-cnpg-ha-operator-1.25.0-ja-v1.0.0-ubi7.tar	образ контейнера jatoba cnpg оператора
jatoba-cnpg-ha-cluster-6.10.1-57608-dfl.6.3-redos7.3-comm.tar	образ контейнера jatoba cnpg кластера (контейнер Jatoba)
jatoba-cnpg-japooler-6.10.1-redos7.3.tar	образ контейнера jatoba cnpg japooler
jatoba-cnpg-operator0.23.0-app1.25.0-ja-v1.0.0-helm-charts.tar	Helm-чарты оператора
jatoba-cnpg-cluster0.2.1-app6.10.1-57608-dfl.6.3-helm-charts.tar	Helm-чарты кластера БД
kubectl-cnpg-v1.25.3.tar	kubectl-cnpg плагин

В образе контейнера Jatoba CNPG кластера содержится БД «Jatoba» и компоненты к ней (см. таблицу 5.2). Настройка компонентов описана отдельно в соответствующих документах.

Таблица 5.2 - Состав образа контейнера

№	Наименование	J6		Описание
		комм	серт	
1	ядро СУБД	X	—	Ядро СУБД
1.1	server	X	—	Сервер СУБД
1.2	client	X	—	Клиентские утилиты для СУБД
1.3	contrib	X	—	Дополнительные расширения для сервера СУБД
1.4	libs	X	—	Библиотеки для клиентских утилит СУБД
2	plpython3	X	—	Расширение для реализации хранимых процедур на языке Python 3
3	postgis	X	—	Расширение для реализации поддержки географических объектов
3.1	gdal	X	—	Библиотека, необходимая для работы модуля postgis raster
4	pgvector	X	—	Расширение для реализации поиск векторного сходства
5	pg_failover_slots	X	—	Расширение для реализации взаимодействия с логическими слотами репликации
6	pgaudit	X	—	Расширение для реализации дополнительных возможностей аудита для ведения журнала в PostgreSQL

6. ПОДГОТОВКА К УСТАНОВКЕ ОПЕРАТОРА И КЛАСТЕРА

6.1. Загрузка Docker-образов Jatoba CNPG в локальный docker registry на мастер и воркер узлах кластера k3s

На Мастер и Воркер узлы кластера k3s нужно скопировать *tar архивы с Docker-образами «Jatoba», входящими в комплект поставки. После этого необходимо через команду `ctr image import` загрузить данные образы в локальный docker registry. Команды выполняются с повышенными привилегиями или от пользователя «root». Команды выполняются как на Мастер-узле, так и на Воркер-узле кластера k3s.

```
sudo k3s ctr image import jatoba-cnpg-ha-operator-1.25.0-ja-v1.0.0-ubi7.tar
sudo k3s ctr image import jatoba-cnpg-ha-cluster-6.10.1-57608-df1.6.3-redos7.3-comm.tar
sudo k3s ctr image import jatoba-cnpg-japooler-6.10.1-redos7.3.tar
```

```
luser@k3s01:/tmp/cnpg$ ls -al
total 3304164
drwxrwxr-x 2 luser luser 4096 Oct 6 15:31 .
drwxrwxrwt 14 root root 4096 Oct 6 15:30 ..
-rw-rw-r-- 1 luser luser 430080 Oct 6 15:31 jatoba-cnpg-cluster0.2.1-app6.9.1-57496-df1.6.2-helm-charts.tar
-rw-rw-r-- 1 luser luser 2600252928 Oct 6 15:31 jatoba-cnpg-ha-cluster-6.9.1-57496-df1.6.2-redos7.3-comm.tar
-rw-rw-r-- 1 luser luser 131999744 Oct 6 15:31 jatoba-cnpg-ha-operator-1.25.0-ja-v1.0.0-ubi7.tar
-rw-rw-r-- 1 luser luser 649579008 Oct 6 15:31 jatoba-cnpg-japooler-6.9.1-redos7.3.tar
-rw-rw-r-- 1 luser luser 1177600 Oct 6 15:31 jatoba-cnpg-operator0.23.1-app1.25.0-ja-v1.0.0-helm-charts.tar
luser@k3s01:/tmp/cnpg$ sudo ctr -n=k8s.io image import jatoba-cnpg-ha-operator-1.25.0-ja-v1.0.0-ubi7.tar
docker.io/jatoba/cloudnative pg/operator saved
application/vnd.oci.image.manifest.v1+json sha256:6fea74f492f3edd0ce373612587c3e2025cd78710127b79638551edc49a8b75c
Importing elapsed: 5.3 s total: 0.0 B (0.0 B/s)
luser@k3s01:/tmp/cnpg$ sudo ctr -n=k8s.io image import jatoba-cnpg-ha-cluster-6.9.1-57496-df1.6.2-redos7.3-comm.tar
docker.io/jatoba/cloudnative pg/cluster: saved
application/vnd.oci.image.manifest.v1+json sha256:b742443f64a9ce0e5d731216b392abc09933734763c0d8b7e61068b8df7287c8
Importing elapsed: 107.9s total: 0.0 B (0.0 B/s)
luser@k3s01:/tmp/cnpg$ sudo ctr -n=k8s.io image import jatoba-cnpg-japooler-6.9.1-redos7.3.tar
docker.io/jatoba/cloudnative pg/japooler saved
application/vnd.oci.image.manifest.v1+json sha256:a83f4761f710091158cc0e642df73ff9c3691ee6272a566c3ec1ce76a4146e56
Importing elapsed: 24.5s total: 0.0 B (0.0 B/s)
```

Рисунок 6.1 – Загрузка образов

Проверить наличие образов в локальном docker registry можно следующей командой:

```
sudo ctrctl images list
```

```
luser@k3s01:/tmp/cnpg$ sudo crictl images
IMAGE TAG IMAGE ID SIZE
docker.io/jatoba/cloudnative-pg/cluster 6.9.1-57496-df1.6.2-redos7.3-comm 0434d1bc78374 2.6GB
docker.io/jatoba/cloudnative-pg/japooler 6.9.1-redos7.3 73fd183259e02 650MB
docker.io/jatoba/cloudnative-pg/operator 1.25.0-ja-v1.0.0-ubi7 d9e90ca1487a2 132MB
docker.io/rancher/klipper-helm v0.9.8-build20250709 180d1ef27ac95 76.8MB
docker.io/rancher/klipper-lb v0.4.13 f7415d0003cb6 5.02MB
docker.io/rancher/local-path-provisioner v0.0.31 8309ed19e06b9 20.7MB
docker.io/rancher/mirrored-coredns-coredns 1.12.3 0392ee0389032 22.4MB
docker.io/rancher/mirrored-library-traefik 3.3.6 3a1e150bf4c56 58.3MB
docker.io/rancher/mirrored-metrics-server v0.8.0 b9e1e3849e070 22.5MB
docker.io/rancher/mirrored-pause 3.6 6270bb605e12e 301kB
```

Рисунок 6.2 - Проверка наличия образов

6.2. Распаковка архива с helm-чартами

Распаковываем архивы с helm-чартами на мастер-узле k3s кластера:

```
tar xvf jatoba-cnpg-cluster0.2.1-app6.10.1-57608-df1.6.3-helm-charts.tar
tar xvf jatoba-cnpg-operator0.23.1-app1.25.0-ja-v1.0.0-helm-charts.tar
```

6.3. Установка дополнительных компонент

Для корректной работы отказоустойчивого кластера СУБД «Jatoba» в среде Kubernetes требуется установка дополнительных сервисов.

Дополнительные сервисы могут быть установлены через утилиты «kubectl» и «helm», установленные на мастер-узле.

6.3.1. Установка MinIO сервиса для хранения бекапов

Для хранения резервных копий кластера «Jatoba» CNPG необходимо установить локальное s3-хранилище на базе сервиса MinIO. Установка производится из helm репозитория yandex-bitnami:

```
# Добавляем репозиторий helm
helm repo add yandex-bitnami https://mirror.yandex.ru/helm/charts.bitnami.com

# Устанавливаем MinIO
helm upgrade --install minio -n minio --create-namespace \
  --version 16.0.11 --set mode=standalone \
  --set global.security.allowInsecureImages=true \
  --set image.repository=bitnamilegacy/minio \
  --set image.tag=2025.4.22-debian-12-r1 \
  --set statefulset.replicaCount=1 \
  --set provisioning.enabled=false \
  --set resourcesPreset=micro \
  --set ingress.enabled=true \
  --set ingress.ingressClassName=traefik \
  --set ingress.hostname=minio.jatoba.local \
  --set networkPolicy.enabled=false \
  --set persistence.size=2Gi \
yandex-bitnami/minio
```

```

root@k3s01:/tmp/cnpg# helm repo add yandex-bitnami https://mirror.yandex.ru/helm/charts/bitnami.com
"yandex-bitnami" has been added to your repositories
root@k3s01:/tmp/cnpg# helm upgrade --install minio -n minio --create-namespace \
--version 16.0.11 --set mode=standalone \
--set global.security.allowInsecureImages=true \
--set image.repository=bitnamilegacy/minio \
--set image.tag=2025.4.22-debian-12-r1 \
--set statefulset.replicaCount=1 \
--set provisioning.enabled=false \
--set resourcesPreset=micro \
--set ingress.enabled=true \
--set ingress.ingressClassName=traefik \
--set ingress.hostname=minio.jatoba.local \
--set networkPolicy.enabled=false \
--set persistence.size=2Gi \
yandex-bitnami/minio
Release "minio" does not exist. Installing it now.
NAME: minio
LAST DEPLOYED: Mon Oct 6 15:44:34 2025
NAMESPACE: minio
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: minio
CHART VERSION: 16.0.11
APP VERSION: 2025.4.22

Did you know there are enterprise versions of the Bitnami catalog? For enhanced secure software supply chain features, unlimited pulls
Premium or Tanzu Application Catalog. See https://www.arrow.com/globalecs/na/vendors/bitnami for more information.

```

Рисунок 6.3 Установка репозитория yandex-bitnami

Проверить, что под с minio работает:

```
kubectl -n minio get pod
```

```

juser@u602doc-k3s-01:~$ sudo kubectl -n minio get pod
NAME                                READY   STATUS    RESTARTS   AGE
minio-7c55cc88d8-wc97c             1/1     Running   0           2m48s

```

Рисунок 6.4 – Проверка статуса

Чтобы получить доступ к веб-интерфейсу MinIO, необходимо в файл hosts на компьютере, с которого производится управление кластером, добавить запись (<MASTER_NODE_IP> - IP-адрес мастер-ноды кластера k3s):

```
<MASTER_NODE_IP> minio.jatoba.local
```

Войти в веб-интерфейс можно по ссылке <http://minio.jatoba.local>, с логином admin, и паролем, полученным в результате выполнения следующей команды:

```

kubectl get secret -n minio minio \
-o jsonpath='{.data.root-password}' | base64 --decode

```

6.3.2. Установка SSL сертификатов для доступа к удаленному docker registry

Выполнить на хосте с ролью сервер» (мастер узел). Установка сертификатов производится на примере удаленного docker registry nexus.da.lan.

Добавить сертификат с хоста nexus.da.lan:443:

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

```
sudo sh -c 'openssl s_client -connect nexus.da.lan:443 -showcerts </dev/null 2>/dev/null | openssl x509 -outform PEM > /etc/ssl/certs/nexus_da_lan.pem'
```

Перезапустить службу k3s:

```
sudo systemctl restart k3s.service
```

```
juser@u602doc-k3s-01:~$ sudo sh -c 'openssl s_client -connect nexus.da.lan:443 -showcerts </dev/null 2>/dev/null | openssl x509 -outform PEM > /etc/ssl/certs/nexus_da_lan.pem'
juser@u602doc-k3s-01:~$ ls -al /etc/ssl/certs/ | grep nexus.da.lan
-rw-r--r-- 1 root root 2098 May 13 16:03 nexus_da_lan.pem
juser@u602doc-k3s-01:~$
```

Рисунок 6.5 – Перезапуск службы k3s

Дополнительно убедиться, что служба корректно запустилась и работает после перезапуска:

```
systemctl status k3s.service
```

6.3.3. Установка monitoring сервисов Prometheus и Grafana

Для отображения метрик работы кластера необходимо развернуть сервисы Prometheus и Grafana на Воркер-узлы. Установка производится из helm репозитория prometheus-community, используется чарт kube-prometheus-stack. Рекомендованный файл values (values/values-prom-stack.yaml) находится в архиве с чартами.

Необходимо выполнить следующие команды:

```
# Добавляем репозиторий helm
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
# Устанавливаем сервисы мониторинга
helm upgrade --install \
  --create-namespace \
  -n monitoring \
  -f values/values-prom-stack.yaml \
  prometheus-community \
  prometheus-community/kube-prometheus-stack
```

Проверить, что все установилось корректно и работает (релиз в статусе «deployed», поды в статусе «Running»):

```
helm list -n monitoring
```

user@u604ops-k3s-test-01:~/cnpg/charts/manifests/client\$ helm list -n monitoring					
NAME	NAMESPACE	REVISION	UPDATED	STATUS	CHART
prometheus-community	monitoring	4	2025-05-21 17:30:05.310805929 +0300 MSK	deployed	kube-prometheus-stack-72.5.1
				APP VERSION	v0.82.2

Рисунок 6.6 – Вывод статуса

```
kubectl get pod -n monitoring
```

user@u604ops-k3s-test-01:~/cnpg/charts/manifests/client\$ kubectl get po -n monitoring					
NAME	READY	STATUS	RESTARTS	AGE	
prometheus-community-grafana-6c59465696-5rd7g	3/3	Running	4 (2d6h ago)	7d1h	
prometheus-community-kube-operator-559545c8cb-9djqs	1/1	Running	1 (7d21h ago)	7d22h	
prometheus-community-kube-state-metrics-5b4f99697c-r7zjj	1/1	Running	2 (2d6h ago)	7d22h	
prometheus-prometheus-community-kube-prometheus-0	2/2	Running	2 (7d21h ago)	7d22h	

Рисунок 6.7 – Вывод статуса

В файл hosts на компьютере, с которого производится управление кластером, добавить запись (<MASTER_NODE_IP> - IP-адрес мастер-ноды кластера k3s), либо заранее поменяйте доменное имя на нужное отредактировав файл values-prom-stack.yaml:

```
<MASTER_NODE_IP> monitor.jatoba.local
```

Чтобы система мониторинга отображала данные, необходимо подключить ее к кластеру. Данная процедура описана в п. 10.6 документации. Войти в веб-интерфейс можно по ссылке <http://monitor.jatoba.local/d/cloudnative-pg/cloudnativepg> и войти с логином «admin» и паролем, полученным в результате выполнения следующей команды:

```
kubectl get secret -n monitoring prometheus-community-grafana \
-o jsonpath='{.data.admin-password}' | base64 --decode
```

Откроется дашборд с данными о кластере. После подключения кластера CNPG к системе мониторинга, необходимо подождать 5-10 минут, чтобы накопился объем данных, достаточный для корректной визуализации.

6.3.4. Установка cert manager

Установка компонента для управления сертификатами в кластере K3s выполняется на хосте с ролью сервер» (мастер узел).

Необходимо выполнить следующие команды:

- Указать версию Cert-manager в переменной:

```
CERTM_VERSION=1.18.2
```

- Создать пространство имен для «cert-manager» с именем «cert-manager»:

```
kubectl create namespace cert-manager
```

- Применить манифест добавляющий новые ресурсы в API Kubernetes (CRD) относящиеся к «cert-manager»:

```
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/  
download/v${CERTM_VERSION}/cert-manager.crd.yaml  
helm repo add jetstack https://charts.jetstack.io  
helm repo update  
helm install cert-manager jetstack/cert-manager \\  
  --namespace cert-manager \\  
  --version v${CERTM_VERSION} \\  
  --create-namespace
```



Рекомендуется использовать в качестве репозитория yandex mirror

- Добавить репозиторий с чартами, обновить список репозитория и выполнить установку «cert-manager» нужной версии с параметрами по умолчанию:

Расскоментируйте в блоке команды. Команда должна выглядеть следующим образом:

```
helm repo add yandex-jetstack https://mirror.yandex.ru/helm/charts.jets  
tack.io  
helm repo update  
helm install cert-manager yandex-jetstack/cert-manager \\  
  --namespace cert-manager \\  
  --version v${CERTM_VERSION} \\  
  --create-namespace
```

7. УСТАНОВКА ОПЕРАТОРА

Предварительные требования для установки:

- Наличие кластера Kubernetes или аналогичной среды;
- Наличие доступа к кластеру Kubernetes (файл конфигурации с учетными данными и адресом кластера);
- Наличие утилит командной строки - kubectl, helm - на устройстве, с которого производится установка;
- Загруженный в удаленный docker registry образ с Jatoba для CloudNativePG, образ менеджера соединений для CloudNativePG (pgbouncer) и образ оператора CloudNativePG (альтернативой реестру образов могут выступать образы, заранее загруженные в локальный docker registry узлов кластера вручную, через docker pull или ctr image import, в зависимости от используемого кластером CRI). Далее в установке оператора будет рассмотрен вариант загрузки образов через локальный docker registry k3s кластера. Инструкция по загрузке в локальный docker registry приведена в п 6.1.

7.1. Параметры развертывания

Конфигурация, используемая для установки в кластер Kubernetes для оператора и кластера СУБД «Jatoba», определяется через файл с параметрами по умолчанию – jatoba-default-values.yaml. Файлы конфигурации приведены в Приложении 2 и выше.

Параметры конфигурации можно изменять как в самом файле jatoba-default-values.yaml, так и создавать отдельные файлы конфигураций и указывать их в команде установки.

При установке могут требоваться параметры, которые меняются в зависимости от окружения, такие параметры можно задать через ключи утилиты командной строки

```
--set <имя_параметра>=<значение_параметра>
```

7.2. Установка оператора

Команды развертывания оператора CloudNativePG. Данные команды выполняются на Мастер-узле k3s кластера.

Чарт оператора может содержать зависимости, которые требуются для установки, первым шагом требуется выполнить команду проверки и обновления зависимостей. Если вывод команды отсутствует, значит зависимости не обнаружены и можно приступить к следующему шагу.

Перейти в директорию с чартами и обновляем зависимости чарта:

```
helm dependency update ./charts/jatoba-cnpg-operator/
```

```
juser@u602doc-k3s-01:/tmp/cnpg$ helm dependency update ./charts/cloudnative-pg/
Getting updates for unmanaged Helm repositories...
...Successfully got an update from the "https://cloudnative-pg.github.io/grafana-dashboards" chart repository
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "yandex-bitnami" chart repository
Update Complete. *Happy Helming!*
Saving 1 charts
Downloading cluster from repo https://cloudnative-pg.github.io/grafana-dashboards
Deleting outdated charts
juser@u602doc-k3s-01:/tmp/cnpg$
```

Рисунок 7.1 – Обновление зависимостей чарта

Запустить команду установки оператора, в команде указываем image.repository и image.tag, значения которых соответствуют имени и тегу выгруженному на шаге 6.1 архива с Jatoba CNPG оператором (jatoba-cnpg-ha-operator-1.25.0-ja-v1.0.0-ubi7.tar):

```
helm upgrade --install cnpg-operator \
  -n cnpg-system \
  --create-namespace \
  --set monitoring.grafanaDashboard.create=false \
  --set image.repository=jatoba/cloudnative-pg/operator \
  --set image.tag=1.25.0-ja-v1.0.0-ubi7 \
  ./charts/jatoba-cnpg-operator/
```

Проверить работу оператора можно следующими командами:

```
helm list -n cnpg-system
kubectl get pod -n cnpg-system
```

Результатом вывода команд будет информация о версии оператора и о запущенном поде с оператором:

```
luser@k3s01:/tmp/cnpg$ helm list -n cnpg-system
NAME      NAMESPACE    REVISION    UPDATED                               STATUS          CHART              APP VERSION
cnpg-operator  cnpg-system    1           2025-10-07 05:46:49.387493684 +0000 UTC deployed      jatoba-cnpg-operator-0.23.1  1.25.0-ja-v1.0.0

luser@k3s01:/tmp/cnpg$ kubectl get pod -n cnpg-system
NAME                                READY    STATUS    RESTARTS   AGE
cnpg-operator-cloudnative-pg-5c66ffdfb4-s5rzk  1/1      Running   0          3m24s
```

Рисунок 7.2 - Вывод информации о версии оператора

7.3. Установка кластера

В данном разделе представлена инструкция для развертывания MVP (минимально жизнеспособный продукт) кластера. Способ указания значений – файл конфигурации jatoba-default-values.yaml.

```
export NSCLUSTER=jatoba-cnpg-cluster
helm upgrade --install cnpg-cluster -n $NSCLUSTER \
  --create-namespace \
  --values ./values/jatoba-default-values.yaml \
  ./charts/$NSCLUSTER/
```

Установка дополнительных опций описана в 11 разделе настоящего документа.

```
luser@debian:/tmp/cnpg$ export NSCLUSTER=jatoba-cnpg-cluster
luser@debian:/tmp/cnpg$ helm upgrade --install cnpg-cluster -n $NSCLUSTER \
> --create-namespace \
> --values ./values/jatoba-default-values.yaml \
> ./charts/$NSCLUSTER/
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /home/luser/.kube/config.jatoba-stage
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /home/luser/.kube/config.jatoba-stage
Release "cnpg-cluster" does not exist. Installing it now.
NAME: cnpg-cluster
LAST DEPLOYED: Fri Sep 12 09:10:05 2025
NAMESPACE: jatoba-cnpg-cluster
STATUS: deployed
REVISION: 1
NOTES:
The cnpg-cluster-jatoba-cnpg-cluster has been installed successfully.
```

CloudNativePG

Cheatsheet

```
Run Helm Tests:
helm test --namespace jatoba-cnpg-cluster cnpg-cluster

Get a list of all base backups:
kubectl --namespace jatoba-cnpg-cluster get backups --selector cnpg.io/cluster=cnpg-cluster-jatoba-cnpg-cluster

Connect to the cluster's primary instance:
kubectl --namespace jatoba-cnpg-cluster exec --stdin --tty services/cnpg-cluster-jatoba-cnpg-cluster-rw -- bash
```

Configuration

Configuration	Value
Cluster mode	standalone
Type	postgresql
Image	ImageCatalog: cnpg-cluster-jatoba-cnpg-cluster(16)
Instances	3
Backups	Disabled
Storage	2Gi
Storage class	Default
PGBouncer	Disabled
Monitoring	Disabled

Warning! Backups not enabled. Recovery will not be possible! Do not use this configuration in production.

Рисунок 7.3 – Установка кластера

Проверить наличие релиза кластера возможно командой:

```
helm list -n $NSCLUSTER
```

luser@k3s01:/tmp/cnpg\$ helm list -n \$NSCLUSTER						
NAME	NAMESPACE	REVISION	UPDATED	STATUS	CHART	APP VERSION
cnpg-cluster	jatoba-cnpg-cluster	1	2025-10-07 05:51:50.21989325 +0000 UTC	deployed	jatoba-cnpg-cluster-0.2.1	6.9.1-57496-df1.6.2

Рисунок 7.4 – Проверка релиза

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

8. УДАЛЕНИЕ ОПЕРАТОРА И КЛАСТЕРА

Для удаления ранее установленных оператора и кластеров CloudNativePG необходимо удалить релизы, устанавливаемые через helm.

Оператор расширяет конфигурацию API кластера Kubernetes, внося дополнительные ресурсы (CRD). Их требуется удалить отдельно.

Удаление кластера СУБД Jatoba:

```
helm uninstall cnpg-cluster -n jatoba-cnpg-cluster
```

Удаление оператора CloudNativePG:

```
helm uninstall cnpg-operator -n cnpg-system
```

Удаление пространств имен в кластере Kubernetes используемых при установке оператора и кластера:

```
kubectl delete ns jatoba-cnpg-cluster  
kubectl delete ns cnpg-system
```

Удаление Custom Resource Definition (CRD) устанавливаемых с оператором CloudNativePG:

```
kubectl delete crd backups.postgresql.cnpg.io  
kubectl delete crd clusterimagecatalogs.postgresql.cnpg.io  
kubectl delete crd clusters.postgresql.cnpg.io  
kubectl delete crd databases.postgresql.cnpg.io  
kubectl delete crd imagecatalogs.postgresql.cnpg.io  
kubectl delete crd poolers.postgresql.cnpg.io  
kubectl delete crd publications.postgresql.cnpg.io  
kubectl delete crd scheduledbackups.postgresql.cnpg.io  
kubectl delete crd subscriptions.postgresql.cnpg.io
```

9. ОСНОВНЫЕ ОПЕРАЦИИ С КЛАСТЕРОМ

Большая часть возможностей по конфигурированию вносится через параметры при разворачивании кластера. Тем не менее, с уже развернутым кластером можно выполнять определенные действия.

9.1. Просмотр ресурсов кластера или оператора в кластере Kubernetes.

Ресурсы, создаваемые в кластере Kubernetes, можно получать и просматривать через команды `get` и `describe` утилиты `kubectl`.

Полный список ресурсов кластера Kubernetes можно получить командой, для каждого из них возможно выполнить команду `get`:

```
kubectl api-resources --verbs=list
```

Пример для вывода всех узлов кластера СУБД Jatoba:

```
kubectl -n jatoba-cnpg-cluster get pod
```

Ресурсы, относящиеся к установке кластера и оператора:

- Ресурс типа `ConfigMap` (файлы конфигурации, монтируемые в контейнеры);
- Ресурс типа `Secret` (чувствительные данные, необходимые для работы контейнеров - пароли, строки подключения и т.п.);
- Ресурс типа `ServiceAccount` (сервисные учетные записи для обращения приложений к API кластера);
- Ресурс типа `Service` (сервисы, отвечающие за сетевое взаимодействие между приложениями);
- Ресурс типа `Endpoints` (точки подключения к сервисам);
- Ресурс типа `Deployment` (параметры развернутого приложения);
- Ресурс типа `ReplicaSet` (конфигурация кол-ва копий сервиса на узлах);
- Ресурс типа `Pod` (контейнеры приложения);

Ресурсы, создаваемые только при разворачивании кластера:

- Ресурс типа `Cluster` (описывает развернутый кластер и его параметры);

- Ресурс типа Backup (описывает существующие задания резервного копирования, если они выполнялись);
- Ресурс типа ScheduledBackups (запланированные задания резервного копирования);
- Ресурс типа PersistentVolumeClaim (тома, подключенные к контейнерам приложения);
- Ресурс типа Pooler (описывает специализированные сервисы подключения к кластеру);

Ресурсы, относящиеся только к оператору — это ресурсы Custom Resource Definition (CRD), которые расширяют API Kubernetes, внося дополнительные объекты для управления.

Запросить ресурсы такого типа можно следующей командой:

```
kubectl get crds | grep cnpg
```

В данной команде используется дополнительная фильтрация вывода через утилиту командной строки `grep` выделяющую объекты в выводе по ключевому слову «`cnpg`».

9.2. Просмотр журналов кластера и оператора

Для просмотра журналов работы развернутого ранее кластера можно использовать специализированные инструменты или утилиту командной строки `kubectl`.

Просмотр журналов для первого узла кластера. Предварительно запрашивается список подов и на основе него берется имя первого пода (в примере это `cnpg-cluster-jatoba-cnpg-cluster-1`):

```
kubectl -n jatoba-cnpg-cluster logs cnpg-cluster-jatoba-cnpg-cluster-1
```

9.3. Подключение к кластеру напрямую (локально в поде СУБД)

В некоторых случаях (например, для отладки или администрирования), возникает необходимость подключиться к СУБД напрямую, без промежуточных звеньев.

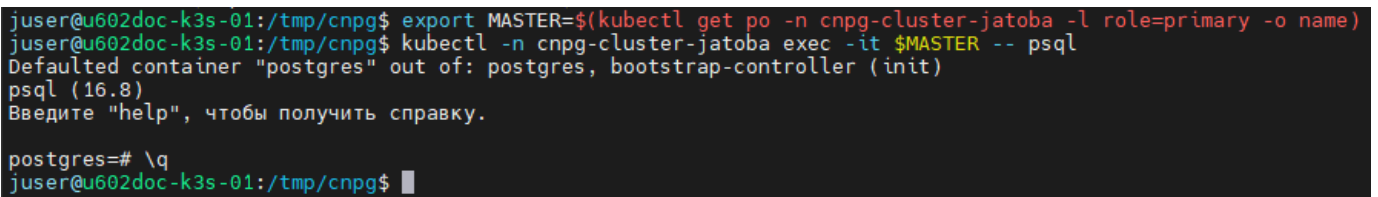
Данный способ нельзя использовать для работы с прикладным ПО, так как он требует доступа к API Kubernetes, что открывает большую поверхность атаки на инфраструктуру, плохо поддается автоматизации, и открывает доступ к БД от суперпользователя.

Для подключения к поду кластера необходимо получить имя пода, являющегося мастером:

```
export MASTER=$(kubectl get pod -n jatoba-cnpg-cluster -l role=primary -o name)
```

И выполнить следующую команду:

```
kubectl -n jatoba-cnpg-cluster exec -it $MASTER -- psql
```



```
juser@u602doc-k3s-01:/tmp/cnpg$ export MASTER=$(kubectl get po -n cnpg-cluster-jatoba -l role=primary -o name)
juser@u602doc-k3s-01:/tmp/cnpg$ kubectl -n cnpg-cluster-jatoba exec -it $MASTER -- psql
Defaulted container "postgres" out of: postgres, bootstrap-controller (init)
psql (16.8)
Введите "help", чтобы получить справку.

postgres=# \q
juser@u602doc-k3s-01:/tmp/cnpg$
```

Рисунок 9.1 - Подключение к поду кластера

9.4. Подключение к кластеру из клиента (отдельный под)

В данном варианте подключение к СУБД происходит без балансировщика соединений (пуллера), но по сети, с использованием DNS k3s-кластера.



Обязательно удаляйте под с клиентом и создавайте его заново, особенно при пересоздании кластера или изменении логина и пароля

Этот вариант подходит для тестирования подключения к СУБД.

В репозиторий с чартами входит манифест с конфигурацией пода клиента psql, из которого можно взять шаблон конфигурации переменных для подключения прикладного ПО, развёрнутого в кластере k3s к СУБД (в разделе env).

Для подключения необходимо создать под с клиентом psql из приложенного манифеста:

```
kubectl -n jatoba-cnpg-cluster apply -f manifests/client/psql-client.yaml
# Удаление пода
kubectl -n jatoba-cnpg-cluster delete -f manifests/client/psql-client.yaml
```

И выполнить следующую команду:

```
kubectl -n jatoba-cnpg-cluster exec -it pg-client-ssl -- /bin/sh \
-c '/usr/bin/psql $PSQL_CONNECTION_STRING'
```

```
juser@u602doc-k3s-01:/tmp/cnpg$ kubectl -n cnpg-cluster-jatoba apply -f manifests/client/psql-client.yaml
pod/pg-client-ssl configured
juser@u602doc-k3s-01:/tmp/cnpg$ kubectl -n cnpg-cluster-jatoba exec -it pg-client-ssl -- /bin/sh -c '/usr/bin/psql -At $PSQL_CONNECTION_STRING'
psql (17.4, server 16.8)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off, ALPN: none)
Type "help" for help.

app=>
\q
juser@u602doc-k3s-01:/tmp/cnpg$
```

Рисунок 9.2 - Подключение к кластеру из клиента

10. ИЗМЕНЕНИЕ ПОЛЬЗОВАТЕЛЬСКИХ НАСТРОЕК ПРИ УСТАНОВКЕ КЛАСТЕРА

Все настройки для разворачиваемого кластера и оператора хранятся в файле `./values/jatoba-default-values.yaml`. Для внесения изменений можно редактировать уже имеющийся файл или подключать дополнительные параметры (опции) через отдельные файлы с конфигурациями (блоками конфигураций). Файлы конфигурации с опциями расположены в директории `values/values-opt`. Примеры файлов и конфигураций опций представлены в Приложениях документации.

10.1. Блок конфигурации bootstrap

В этом блоке описаны конфигурации, скрипты инициализации БД и т.д., используемые **при разворачивании кластера**. Параметры блока конфигурации bootstrap представлены в файле `values/values-opt/jatoba-bootstrap.yaml` (см. Приложение 3).

Предусловие: Т.к. bootstrap обрабатывает только при инициализации кластера, то в разворачиваемом namespace не должно быть уже развернутого кластера.

В файле конфигурации чарта кластера `values/values-opt/jatoba-bootstrap.yaml` существует раздел `postgresql.initdb`, в котором задаются параметры создания БД в кластере. В подразделе `postInitApplicationSQL` можно задать набор произвольных SQL-запросов, который будет выполнен после создания БД приложения. Значение по умолчанию для данного набора в файлах конфигурации чарта кластер Jatoba задано следующим:

```
- create extension if not exists dblink;
- create extension if not exists ltree;
- create extension if not exists pgcrypto;
- create extension if not exists postgres_fdw;
- create extension if not exists fuzzystmatch;
- create extension if not exists lo;
- create extension if not exists pg_trgm;
- create extension if not exists tablefunc;
- create extension if not exists xml2;
- create extension if not exists adminpack;
- create extension if not exists plpython3u;
- create extension if not exists postgis;
- create extension if not exists postgis_tiger_geocoder cascade;
- create extension if not exists postgis_topology cascade;
- create extension if not exists postgis_raster cascade;
- create extension if not exists vector;
```

То есть, создаются все указанные в запросе расширения. Данный запрос можно отредактировать, убрав ненужные расширения, или удалить полностью, если в БД не нужно устанавливать расширения автоматически.

В рассматриваемом примере создается база данных `app`, с кодировкой UTF-8, заполняется тестовыми данными и создается расширение. Для запуска создания кластера с измененными параметрами инициализации БД необходимо выполнить следующие команды:

```
helm upgrade --install cnpg-cluster -n $NSCLUSTER --create-namespace \
--values ./values/jatoba-default-values.yaml \
--values ./values/values-opt/jatoba-bootstrap.yaml \ ./charts/$NSCLUSTER/
```

Проверить доступные расширения можно подключившись к кластеру напрямую или из пода клиента (см п 7.4 и 7.5 настоящей инструкции) выполнив следующую команду:

```
SELECT * from pg_available_extensions;
```

Для получения списка установленных расширений необходимо выполнить команду:

```
SELECT extname,extversion from pg_extension;
```

Альтернативный вариант команды вывода расширений:

```
\dx;
```

10.2. Блок конфигурации backups

В этом блоке описаны правила создания бекапов для кластера. Параметры блока конфигурации кластера `backups` представлены в файле `values/values-opt/jatoba-backup.yaml` (см. Приложение 4).

Предусловие: необходимо иметь заранее созданное `s3` хранилища. Создание и настройка `s3 minio` сервиса описана в разделе 6.3.1 документации, также необходимо чтобы в БД были минимальные данные, например тестовая таблица создаваемая через `bootstrap` (раздел 10.1).

Для запуска резервных копий необходимо выполнить следующие команды:

```
# Получаем переменные для доступа к s3 хранилищу (на примере s3 minio описанного в разделе 4.2)
```

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------


```
export S3_ACCESS_KEY=$(kubectl get secret -n minio minio \
-o jsonpath='{.data.root-user}' | base64 --decode)
export S3_SECRET_KEY=$(kubectl get secret -n minio minio \
-o jsonpath='{.data.root-password}' | base64 --decode)

# Указываем файлы конфигураций резервного копирования и учетные данные
для подключения к S3 и запускаем обновление кластера
helm upgrade --install cnpg-cluster -n $NSCLUSTER --create-namespace \
--values ./values/jatoba-default-values.yaml \
--values ./values/values-opt/jatoba-bootstrap.yaml \
--values ./values/values-opt/jatoba-backup.yaml \
--set backups.s3.accessKey=$S3_ACCESS_KEY \
--set backups.s3.secretKey=$S3_SECRET_KEY \
./charts/$NSCLUSTER/

# Проверить состояние выполнения бекапа можно с использованием команд
kubectl -n $NSCLUSTER describe scheduledbackups.postgresql.cnpg.io
kubectl -n $NSCLUSTER get backups.postgresql.cnpg.io

# Проверить наличие бекапа, его размер и wals в minio можно следующими
командами
kubectl -n minio exec $(kubectl -n minio get pods -o name) \
-- du -sh /bitnami/minio/data/cnpg/cnpg-default/cnpg-cluster-jatoba-c
npg-cluster
kubectl -n minio exec $(kubectl -n minio get pods -o name) \
-- ls -al /bitnami/minio/data/cnpg/cnpg-default/cnpg-cluster-jatoba-c
npg-cluster/base
kubectl -n minio exec $(kubectl -n minio get pods -o name) \
-- ls -al /bitnami/minio/data/cnpg/cnpg-default/cnpg-cluster-jatoba-c
npg-cluster/wals/0000000100000000
```

```

user@k3s01:/tmp/cnpg$ kubectl -n minio exec $(kubectl -n minio get pods -o name) \
-- du -sh /bitnami/minio/data/cnpg/cnpg-default/cnpg-cluster-jatoba-cnpg-cluster
16M      /bitnami/minio/data/cnpg/cnpg-default/cnpg-cluster-jatoba-cnpg-cluster
user@k3s01:/tmp/cnpg$ kubectl -n minio exec $(kubectl -n minio get pods -o name) \
-- ls -al /bitnami/minio/data/cnpg/cnpg-default/cnpg-cluster-jatoba-cnpg-cluster/base
total 12
drwxr-sr-x 3 1001 1001 4096 Oct  7 08:45 .
drwxr-sr-x 4 1001 1001 4096 Oct  7 08:45 ..
drwxr-sr-x 4 1001 1001 4096 Oct  7 08:45 20251007T084520
user@k3s01:/tmp/cnpg$ kubectl -n minio exec $(kubectl -n minio get pods -o name) \
-- ls -al /bitnami/minio/data/cnpg/cnpg-default/cnpg-cluster-jatoba-cnpg-cluster/wals/0000000100000000
total 64
drwxr-sr-x 16 1001 1001 4096 Oct  7 08:50 .
drwxr-sr-x  3 1001 1001 4096 Oct  7 08:44 ..
drwxr-sr-x  3 1001 1001 4096 Oct  7 08:44 00000001000000000000000000000001.gz
drwxr-sr-x  3 1001 1001 4096 Oct  7 08:44 00000001000000000000000000000002.gz
drwxr-sr-x  3 1001 1001 4096 Oct  7 08:44 00000001000000000000000000000003.gz
drwxr-sr-x  2 1001 1001 4096 Oct  7 08:44 00000001000000000000000000000004.gz
drwxr-sr-x  2 1001 1001 4096 Oct  7 08:44 00000001000000000000000000000005.00000028.backup.gz
drwxr-sr-x  2 1001 1001 4096 Oct  7 08:44 00000001000000000000000000000005.gz
drwxr-sr-x  3 1001 1001 4096 Oct  7 08:45 00000001000000000000000000000006.gz
drwxr-sr-x  2 1001 1001 4096 Oct  7 08:45 00000001000000000000000000000007.00000028.backup.gz
drwxr-sr-x  2 1001 1001 4096 Oct  7 08:45 00000001000000000000000000000007.gz
drwxr-sr-x  2 1001 1001 4096 Oct  7 08:45 00000001000000000000000000000008.00000108.backup.gz
drwxr-sr-x  2 1001 1001 4096 Oct  7 08:45 00000001000000000000000000000008.gz
drwxr-sr-x  2 1001 1001 4096 Oct  7 08:45 00000001000000000000000000000009.gz
drwxr-sr-x  2 1001 1001 4096 Oct  7 08:45 0000000100000000000000000000000A.gz
drwxr-sr-x  2 1001 1001 4096 Oct  7 08:50 0000000100000000000000000000000B.gz

```

Рисунок 10.1 – Создание резервных копий

Ручной (принудительный) запуск резервных копий запускается применением манифеста manifests/backup/jatoba-backup-manual.yaml. Для этого необходимо выполнить следующую команду:

```
kubectl -n $NSCLUSTER apply -f manifests/backup/jatoba-backup-manual.yaml
```

```

user@k3s-n1:~$ kubectl -n $NSCLUSTER get backups.postgresql.cnpg.io
NAME                                     AGE    CLUSTER                                     METHOD                PHASE
cnpg-cluster-jatoba-cnpg-cluster-backup-manual  2m53s  cnpg-cluster-jatoba-cnpg-cluster          barmanObjectStore     completed
cnpg-cluster-jatoba-cnpg-cluster-daily-backup-20251003143503  4m32s  cnpg-cluster-jatoba-cnpg-cluster          barmanObjectStore     completed

```

Рисунок 10.2 - Ручной (принудительный) запуск создания резервных копий

10.3. Блок конфигурации recovery

В этом блоке описаны правила восстановления из бекапов для кластера. Параметры блока конфигурации кластера recovery представлены в файле values/values-opt/jatoba-recovery.yaml (см. Приложение 5).

Предусловие:

— необходимо иметь заранее созданное s3 хранилище. В s3 хранилище должна находиться заранее созданная резервная копия. S3 резервная копия должна содержать как каталог резервной копии «**base**», так и каталог «**wals**». (Предварительно создать резервную можно с помощью bootstrap, как описано в п.п. 10.1)

— в тестовом окружении предварительно следует удалить ранее развернутый кластер с помощью команд из р. 8, либо изменить имя кластера восстанавливаемого из резервной копии, например заменить «cnpg-cluster» на «cnpg-cluster-recovery».

Восстановление из резервной копии происходит путем создания нового кластера в режиме «recovery».

Для запуска восстановления из резервной копии необходимо выполнить следующие команды:

```
# Получаем переменные для доступа к s3 хранилищу
export S3_ACCESS_KEY=$(kubectl get secret -n minio minio \
  -o jsonpath='{.data.root-user}' | base64 --decode)
export S3_SECRET_KEY=$(kubectl get secret -n minio minio \
  -o jsonpath='{.data.root-password}' | base64 --decode)
# Указываем файлы конфигураций восстановления из бекапа и учетные данны
# е для подключения к S3 и запускаем установку кластера из бекапа.
# recovery.destinationPath - путь к s3 bucket. Должен начинаться с s3:/
# recovery.clusterName - имя bucket (содержит в себе директорию бекапа
# (base) и директорию с wal файлами (wals))

# Итоговая команда:
helm upgrade --install cnpg-cluster -n $NSCLUSTER --create-namespace \
  --values ./values/jatoba-default-values.yaml \
  --values ./values/values-opt/jatoba-recovery.yaml \
  --set recovery.s3.accessKey=$S3_ACCESS_KEY \
  --set recovery.s3.secretKey=$S3_SECRET_KEY \
  --set recovery.destinationPath=s3://cnpg/cnpg-default \
  --set recovery.clusterName=cnpg-cluster-jatoba-cnpg-cluster \
  ./charts/jatoba-cnpg-cluster/

# проверить состояние кластера, восстановленного из бекапа
kubectl -n $NSCLUSTER get pods
```

Пример вывода инициализации кластера, восстанавливаемого из резервной копии:

```
user@k3s-n1:~$ kubectl -n $NSCLUSTER get pods
```

NAME	READY	STATUS	RESTARTS	AGE
cnpg-cluster-jatoba-cnpg-cluster-1	1/1	Running	0	32s
cnpg-cluster-jatoba-cnpg-cluster-1-full-recovery-7nz2n	0/1	Completed	0	54s
cnpg-cluster-jatoba-cnpg-cluster-2	1/1	Running	0	13s
cnpg-cluster-jatoba-cnpg-cluster-2-join-m6kkl	0/1	Completed	0	21s
cnpg-cluster-jatoba-cnpg-cluster-3-join-hpgm8	0/1	Pending	0	2s

Рисунок 10.3 – Вывод инициализации кластера

10.4. Блок конфигурации parameters

В этом блоке описаны конфигурации для изменения параметров postgresql.conf, pg_hba, pg_ident файлов кластера. Параметры блока конфигурации parameters представлены в файле values/values-opt/jatoba-parameters.yaml (см. Приложение 6).

Предусловие: нет.

Для запуска изменения параметров postgresql.conf необходимо выполнить следующие команды:

```
helm upgrade cnpg-cluster -n $NSCLUSTER --create-namespace \
  --values ./values/jatoba-default-values.yaml \
  --values ./values/values-opt/jatoba-parameters.yaml \
  ./charts/jatoba-cnpg-cluster/
```

После запуска этой команды, по очереди будут обновлены поды текущего кластера.

10.5. Блок конфигурации poolers

В этом блоке описаны конфигурации для подключения менеджеров соединений jarooler к БД кластера. Параметры блока poolers представлены в файле values/values-opt/jatoba-poolers.yaml (см. Приложение 7).

Предусловие: нет.

Для запуска создания балансировщиков соединений (пуллеров) необходимо выполнить следующие команды:

```
helm upgrade cnpg-cluster -n $NSCLUSTER --create-namespace \
  --values ./values/jatoba-default-values.yaml \
  --values ./values/values-opt/jatoba-poolers.yaml \
  ./charts/jatoba-cnpg-cluster/
```

Для подключения к кластеру из клиента через балансировщик соединений (пуллер) соединений необходимо:

```
# получить имя сервиса rw для jarooler - точка входа, доменное имя пуллера в кластере
export SVC_POOLER_NAME="$(kubectl -n $NSCLUSTER get svc -o name | grep pooler-rw | cut -d"/" -f 2)}.${NSCLUSTER}"

# Создать под с клиентом psql приложенной командой (хост СУБД для переменной PSQL_CONNECTION_STRING_POOLER динамически заменяется значением, полученным из предыдущей команды)
# Предварительно удалите ранее созданный клиент, если таковой был
```

```
# Удаление пода клиента
kubectl -n $NSCLUSTER delete -f manifests/client/psql-client.yaml --for
ce
# Запуск нового
envsubst '$SVC_POOLER_NAME' < manifests/client/psql-client.yaml | kubec
tl apply -f - -n $NSCLUSTER

# подключиться через под клиента
kubectl -n $NSCLUSTER exec -it pg-client-ssl -- /bin/sh \
  -c '/usr/bin/psql $PSQL_CONNECTION_STRING_POOLER'
# Для подтверждения наличия соединения через пуллер также можно просмот
реть лог самого пуллера
kubectl -n $NSCLUSTER logs --tail=10 $(kubectl -n $NSCLUSTER get pods -
o name | grep pooler-rw) | grep "login attempt"
```

10.6. Блок конфигурации monitoring

В этом блоке описаны конфигурации для подключения метрик мониторинга кластера. Параметры блока monitoring представлены в файле values/values-opt/jatoba-monitoring.yaml (см. Приложение 8).

Предусловие:

- подключение блока poolers (если необходимо мониторить поды с пуллерами), см. п.п. 10.5;
- подключение стека сервисов Prometheus и Grafana для отображения метрик, см п.п. 6.3.3;

Для запуска мониторинга необходимо выполнить следующие команды:

```
# Важно подключать jatoba-poolers.yaml после jatoba-monitoring.yaml для
корректного ппереопределения параметров запуска
helm upgrade --install cnpg-cluster -n $NSCLUSTER --create-namespace \
  --values ./values/jatoba-default-values.yaml \
  --values ./values/values-opt/jatoba-monitoring.yaml \
  --values ./values/values-opt/jatoba-poolers.yaml \
  ./charts/jatoba-cnpg-cluster/
```

Для того, чтобы выключить поддержку мониторинга в кластере Jatoba, необходимо выполнить аналогичную команду обновления кластера с параметрами, но без включения файла ./values/values-opt/jatoba-monitoring.yaml.

10.7. Блок конфигурации ограничений ресурсов кластера

В этом блоке описаны конфигурации для ограничения потребляемых ресурсов кластером.

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

Requests - Минимальный объем ресурсов, который гарантированно выделяется поду.

Limits - Максимальный объем ресурсов, который контейнер может потреблять.

Параметры блока представлены в файлах находящихся в директории ./values/values-opt/resources-preset:

- jatoba-resources-minimal.yaml ;
- jatoba-resources-large.yaml;
- jatoba-resources-recommend.yaml;

Ограничения кластера «Jatoba» настраиваются от доступных ресурсов самого k8s кластера.

Так же в конфигурациях присутствуют оптимальные для этих параметров настройки postgresql.conf.

Запустить jatoba кластер можно в минимальном, оптимальном (рекомендуемом) и большом кластере k8s.

Значения ограничений и параметров кластера k8s более подробно описаны в Приложении 9, настоящего документа.

Для применения ограничений к кластеру необходимо выполнить следующую команду:

```
helm upgrade --install cnpg-cluster -n $NSCLUSTER --create-namespace \
  --values ./values/jatoba-default-values.yaml \
  --values ./values/values-opt/resources-preset/jatoba-resources-minimal.yaml \
  ./charts/jatoba-cnpg-cluster/
```

Проверка заданных ресурсов и вывод ресурсов развернутого кластера можно получить, выполнив следующую команду:

```
cat ./values/values-opt/resources-preset/jatoba-resources-minimal.yaml
| grep -A 6 resources
kubectl -n $NSCLUSTER describe $(kubectl -n $NSCLUSTER get cluster -o name | head -1) | grep -A 6 Resources
```

10.8. Блок конфигурации сертификатов

В этом блоке описаны конфигурации для создания ssl сертификатов и кластера на основе этих сертификатов.

Предусловие:

Установленный «Cert-manager». (<https://cert-manager.io/docs/installation/helm/>)

Настройка cert-manager описана в п.п. 6.3.4.

По умолчанию оператор CNPG генерирует свой набор сертификатов, но ему также можно передать сертификаты, созданные заранее.

Параметры блока представлены в файлах находящихся в директории ./values/values-opt/custom-cert:

- jatoba-custom-certs.yaml;
- jatoba-custom-certs-poolers.yaml.

Конфигурации подключения сертификатов представлены в Приложении 10 текущей документации.

Для запуска кластера с специализированными сертификатами необходимо:

- 1) Создать сертификаты и ключи содержащие сертификаты.

Для этого используется следующий манифест manifests/ssl/gen-cnpg-certs.yaml

Создать «namespace» для применения манифестов сертификатов, если ранее такое не создавалось (например при установке другого кластера).

```
kubectl create ns $NSCLUSTER
```

Применить манифест в созданном ранее «namespace»

```
kubectl -n $NSCLUSTER apply -f manifests/ssl/gen-cnpg-certs.yaml
```

Проверить, что сертификаты созданы командой:

```
kubectl -n $NSCLUSTER get issuers.cert-manager.io  
kubectl -n $NSCLUSTER get certificates.cert-manager.io
```

Проверить, что ключи созданы командой:

```
kubectl -n $NSCLUSTER get secret | grep k8s-dev
```

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

2) Устанавливаем кластер с созданными ранее сертификатами.

```
helm upgrade --install cnpg-cluster -n $NSCLUSTER --create-namespace \
  --values ./values/jatoba-default-values.yaml \
  --values ./values/values-opt/custom-cert/jatoba-custom-certs.yaml \
  ./charts/jatoba-cnpg-cluster/
```

Установка кластера с созданными вручную сертификатами кластера с балансировщиком соединений (пуллером).

```
helm upgrade --install cnpg-cluster -n $NSCLUSTER --create-namespace \
  --values ./values/jatoba-default-values.yaml \
  --values ./values/values-opt/custom-cert/jatoba-custom-certs-poolers.yaml \
  --values ./values/values-opt/custom-cert/jatoba-custom-certs.yaml \
  ./charts/jatoba-cnpg-cluster/
```

3) Проверка конфигурации кластера на наличие и использование сгенерированных через «cert-manager» сертификатов.

```
kubectl -n $NSCLUSTER describe $(kubectl -n $NSCLUSTER get clusters -o name) | grep -A 20 Certificates
```



Предполагается установка нового кластера. Обновление существующего кластера на специализированные сертификаты не предусмотрено.

11. ДОПОЛНИТЕЛЬНЫЕ ОПЕРАЦИИ С КЛАСТЕРОМ

11.1. Создание реплики кластера

В этом разделе описано создание кластера реплики.

Предварительное условие:

— наличие уже ранее развернутого кластера.

Цель:

1) Создать шаблон (резервную копию) с тестовыми данными. И использовать этот резервную копию в тестовой инфраструктуре.

Используйте кластер как шаблон. Создайте новые кластера (реплики) прямо из шаблона кластера. При этом не тратится время на создание и обновление резервных копий. Все возьмет на себя оператор в k8s.

2) Создать реплику, но с задержкой по времени репликации, скажем в пределах рабочего дня.

Таким образом если случится какой-либо сбой и повреждение данных, по причине работы сервисов, возможно иметь кластер, который будет иметь полную копию данных для расследования инцидента, и при этом также иметь возможность быстро откатиться на рабочую конфигурацию, до которой проблемные изменения ещё не дошли, т.к. настроена задержка репликации. Продвиньте реплику и переключитесь на нее.

3) Иметь резервные копии в виде работающего кластера в другом филиале или резервном ЦОД.

Это повышает безопасность и сохранение важных данных.

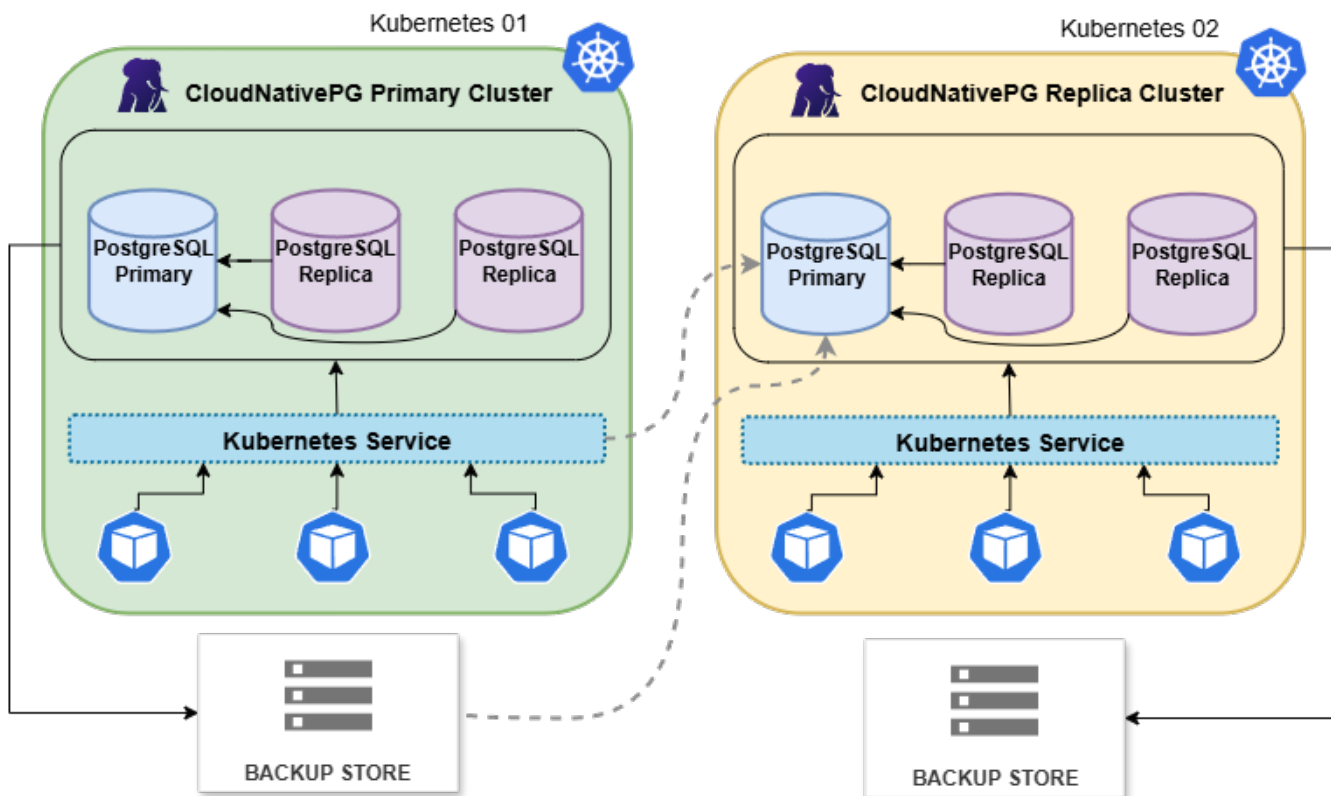


Рисунок 11.1 – Схема работы кластера реплики

Для создания кластер реплики необходимо применить манифест, описанный в Приложении 11 настоящего документа:

```
kubectl -n $NSCLUSTER apply -f ./manifests/clusters/replica/replica-delay-cluster.yaml
```



Создание кластера реплики возможно в рамках namespace созданного ранее кластера.

Требуется предварительно настроить «storageClass». В примере используется «storageClass longhorn».

При необходимости реплики кластера из другого namespace или хоста – укажите соответствующие параметры в манифесте, на основе пояснений в Приложении 11.

11.2. Резервное копирование через VolumeSnapshot

В этом разделе описаны конфигурации, необходимые для создания резервных копий через механизм VolumeSnapshot и восстановления кластера из ранее созданных снимков резервных копий.

Механизм снятия копий БД через снимки томов позволяет восстанавливать данные от гигабайт до нескольких терабайт с минимальными временными затратами.

В данном разделе описаны конфигурации для «longhorn csi driver».

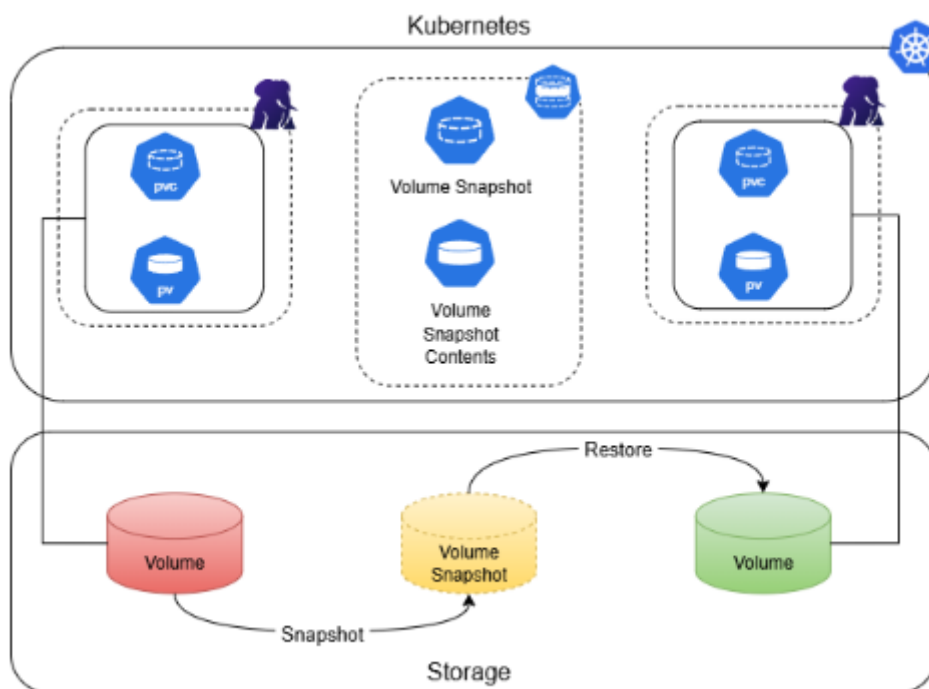


Рисунок 11.2 – Схема создания резервных копий через механизм VolumeSnapshot

11.2.1. Настройка механизма VolumeSnapshot

В данном разделе документации приведена настройка «csi provider» на базе Longhorn. Для других «csi providers» настройка будет отличной от текущей.

Конфигурации настройки Longhorn csi provider находятся в директории manifests/longhorn/backuptarget.yaml и представлены в Приложении 12 настоящего документа.

Механизм резервных копий VolumeSnapshot на базе Longhorn в приведенных примерах предполагает работу с S3-хранилищем. Скорость работы механизма создания снимков с сохранением в S3 не имеет больших преимуществ по скорости работы и приведена в качестве примера, рекомендуется использовать NFS, CIFS или другие аналоги.

Для настройки «csi provider» необходимо:

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

Установить параметры для подключения к S3 для Longhorn в файле конфигурации `./manifests/longhorn/backuptarget.yaml`.



Для доступа к s3 используется tls соединение. Требуется предварительно создать secret с ssl сертификатом в default namespace.

Секрет должен содержать `tls.key` и `tls.crt`.

Значения параметров берется из ресурсов действующего кластера.

```
sed -i "s/<cert_base64>/$(kubectl get secret -n default <имя секрета с  
ssl сертификатом> -o jsonpath='{.data['tls\\.crt']}'})/" ./manifests/lo  
nghorn/backuptarget.yaml  
sed -i "s/<private_key_base64>/$(kubectl get secret -n default <имя сек  
рета с ssl сертификатом> -o jsonpath='{.data['tls\\.key']}'})/" ./manif  
ests/longhorn/backuptarget.yaml
```

Применить манифесты для Longhorn BackupTarget.

```
kubectl apply -f ./manifests/longhorn/backuptarget.yaml  
kubectl apply -f ./manifests/k8s/longhorn-volumesnapshot-class.yaml
```

Проверить созданные ресурсы можно выполнив команду:

```
kubectl get volumesnapshotclasses.snapshot.storage.k8s.io  
kubectl -n longhorn-system get backuptargets.longhorn.io
```

```
user@ubuntu24:~/cnpg/charts$ kubectl get volumesnapshotclasses.snapshot.storage.k8s.io  
kubectl -n longhorn-system get backuptargets.longhorn.io  
NAME          DRIVER          DELETIONPOLICY  AGE  
longhorn       driver.longhorn.io  Delete          4m20s  
longhorn-vs-delete driver.longhorn.io  Delete          4m20s  
longhorn-vs-retain driver.longhorn.io  Retain          4m20s  
NAME          URL              CREDENTIAL          LASTBACKUPAT  AVAILABLE  
default       s3://cnpg@us-east-1/volumesnapshot minio-secret-cert  5m0s          true
```

Рисунок 11.2 – Проверка созданных ресурсов

11.2.2. Настройка резервного копирования через VolumeSnapshot

В данном разделе документации описано создание бэкапа через VolumeSnapshot для csi provider на базе Longhorn. Для других csi providers настройка будет отличной от текущей.

Предварительное условие: настроенный csi provider. см. п.п. 11.2.1.

Конфигурации настройки находятся в манифесте manifests/clusters/volumesnapshots/cluster-backup.yaml. Манифест представлен в Приложении 12 настоящей документации.

В манифесте в качестве примера в bootstrap приведено создание тестовой таблицы.

Для создания резервной копии необходимо:

Отредактировать файл манифеста кластера, указав имя класса CSI используемого для VolumeSnapshot (пример для Longhorn), следующими командами:

```
export CSICLASSNAME=longhorn
sed -i "s/<csi-class-name>/${CSICLASSNAME}/" ./manifests/clusters/volumesnapshots/cluster-backup.yaml
cat ./manifests/clusters/volumesnapshots/cluster-backup.yaml
```

Запустить кластер командой:

```
kubectl -n $NSCLUSTER apply -f ./manifests/clusters/volumesnapshots/cluster-backup.yaml
```

Проверить создание резервной копии можно следующими командами:

```
# Предварительно необходимо убедиться, что все экземпляры кластера запущены
kubectl -n $NSCLUSTER get pods
kubectl -n $NSCLUSTER get backups.postgresql.cnpg.io
kubectl -n $NSCLUSTER get volumesnapshots.snapshot.storage.k8s.io
kubectl -n $NSCLUSTER get volumesnapshotcontents.snapshot.storage.k8s.io

# Проверка контента в S3
# Запрашиваем PVC и выделяем идентификаторы томов реплик
kubectl -n $NSCLUSTER get pvc
# Просматриваем бакет добавляя новый уровень каталога
kubectl -n minio exec $(kubectl -n minio get pods -o name) -- ls -al /bitnami/minio/data/cnpg/volumesnapshot/backupstore/volumes/
```

```
user@ubuntu24:~/cnpg/charts$ kubectl -n $NSCLUSTER get pvc
NAME                                STATUS    VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
cluster-replica-1                  Bound     pvc-ecf3ef0a-1863-4ebe-b6ba-f11023dfc8d9   2Gi        RWO            longhorn       <unset>                  5h37m
cnpg-cluster-jatoba-cnpg-cluster-1 Bound     pvc-838d69e4-ed89-4ef3-94b2-17d9c42ecd85   2Gi        RWO            longhorn       <unset>                  5h45m
cnpg-cluster-jatoba-cnpg-cluster-2 Bound     pvc-f673071e-95a5-4032-acda-498117867c69   2Gi        RWO            longhorn       <unset>                  5h45m
cnpg-cluster-jatoba-cnpg-cluster-3 Bound     pvc-72808062-276c-456b-ale6-b4fa4c9aa9ed   2Gi        RWO            longhorn       <unset>                  5h43m
cnpg-cluster-vs-1                  Bound     pvc-27230c35-cfde-4353-8957-12c45c6cd340   2Gi        RWO            longhorn       <unset>                  13m
cnpg-cluster-vs-2                  Bound     pvc-da660036-22d0-425c-ac7e-05a8eae0dab    2Gi        RWO            longhorn       <unset>                  13m
cnpg-cluster-vs-3                  Bound     pvc-3c9953c1-31e3-416b-a7bf-c193212301b9   2Gi        RWO            longhorn       <unset>                  12m
user@ubuntu24:~/cnpg/charts$ kubectl -n minio exec $(kubectl -n minio get pods -o name) -- ls -al /bitnami/minio/data/cnpg/volumesnapshot/backupstore/volumes/
total 16
drwxr-sr-x 4 1001 1001 4096 Oct 24 12:53 .
drwxr-sr-x 3 1001 1001 4096 Oct 24 12:37 ..
drwxr-sr-x 3 1001 1001 4096 Oct 24 12:37 84
drwxr-sr-x 3 1001 1001 4096 Oct 24 12:53 c0
user@ubuntu24:~/cnpg/charts$ kubectl -n minio exec $(kubectl -n minio get pods -o name) -- ls -al /bitnami/minio/data/cnpg/volumesnapshot/backupstore/volumes/c0
total 12
drwxr-sr-x 3 1001 1001 4096 Oct 24 12:53 .
drwxr-sr-x 4 1001 1001 4096 Oct 24 12:53 ..
drwxr-sr-x 3 1001 1001 4096 Oct 24 12:53 70
user@ubuntu24:~/cnpg/charts$ kubectl -n minio exec $(kubectl -n minio get pods -o name) -- ls -al /bitnami/minio/data/cnpg/volumesnapshot/backupstore/volumes/c0/70
total 12
drwxr-sr-x 3 1001 1001 4096 Oct 24 12:53 .
drwxr-sr-x 3 1001 1001 4096 Oct 24 12:53 ..
drwxr-sr-x 5 1001 1001 4096 Oct 24 12:53 pvc-da660036-22d0-425c-ac7e-05a8eae0dab
```

Рисунок 11.3 – Вывод идентификаторов томов реплик

Для удаления кластера выполните следующую команду:

```
kubectl -n $NSCLUSTER delete -f ./manifests/clusters/volumesnapshots/cluster-backup.yaml
```

Если резервная копия VolumeSnapshot более не нужна и не используется, удалить её можно с помощью следующей команды

```
kubectl -n $NSCLUSTER delete $(kubectl -n $NSCLUSTER get volumesnapshots.snapshot.storage.k8s.io -o name)
```

При необходимости удаления отдельных резервных копий укажите их имен а вручную получив список копий и удалив их по имени через команду kubectl delete

```
kubectl -n $NSCLUSTER get volumesnapshots.snapshot.storage.k8s.io
```

11.2.3. Настройка восстановление кластера через VolumeSnapshot

В данном разделе документации описано восстановление из снимка резервной копии через VolumeSnapshot для csi provider на базе Longhorn. Для других «csi providers» настройка будет отличной от текущей.

Предварительное условие: наличие снимка резервной копии через VolumeSnapshot для csi provider. см. п.п. 11.2.2.

Конфигурации настройки находятся в манифесте manifests/clusters/volumesnapshots/cluster-recovery.yaml. Манифест представлен в Приложении 12 настоящего документа.

Для восстановления из снимка резервной копии необходимо:

Получить последнюю резервную копию VolumeSnapshot и исправить манифест cluster-recovery.yaml

```
export LASTVS=$(kubectl -n $NSCLUSTER get volumesnapshot -o name | cut -d"/" -f 2)
sed -i "s/<VOLUME SNAPSHOT OBJECT NAME>/${LASTVS}/" ./manifests/clusters/volumesnapshots/cluster-recovery.yaml
```

Запустить восстановление кластера из резервной копии VolumeSnapshot следующей командой:

```
kubectl -n $NSCLUSTER apply -f ./manifests/clusters/volumesnapshots/cluster-recovery.yaml
kubectl -n $NSCLUSTER get pod
```

```
user@ubuntu24:~/cnpg/charts$ kubectl -n $NSCLUSTER apply -f ./manifests/clusters/volumesnapshots/cluster-recovery.yaml
cluster.postgresql.cnpg.io/cluster-restore-vs created
user@ubuntu24:~/cnpg/charts$ kubectl -n $NSCLUSTER get pod
NAME                                READY   STATUS    RESTARTS   AGE
cluster-restore-vs-1-snapshot-recovery-4j8r4  0/1     Init:0/1   0          15s
user@ubuntu24:~/cnpg/charts$ kubectl -n $NSCLUSTER get pod
NAME                                READY   STATUS    RESTARTS   AGE
cluster-restore-vs-1                0/1     Init:0/1   0          9s
cluster-restore-vs-1-snapshot-recovery-4j8r4  0/1     Completed  0          85s
user@ubuntu24:~/cnpg/charts$ kubectl -n $NSCLUSTER get pod
NAME                                READY   STATUS    RESTARTS   AGE
cluster-restore-vs-1                1/1     Running   0          51s
```

Рисунок 11.4 - Восстановление кластера из резервной копии VolumeSnapshot

Для удаления восстановленного кластера выполните следующую команду:

```
kubectl -n $NSCLUSTER delete -f ./manifests/clusters/volumesnapshots/cluster-recovery.yaml
```

11.3. Основные команды плагина CNPG для kubectl

В этом разделе описаны команды работы с kubectl-cnpg plugin.

Даже если вы не работали с k3s, то через плагин в любой момент возможно:

- собрать информацию о кластере и сохранить ее в отчет;
- подключиться через клиента в кластер;
- приостановить работу кластера;
- создать резервную копию;

В состав дистрибутива входит «tar» архив с бинарным файлом kubectl-cnpg. Для подключения его к системе необходимо выполнить следующие команды:

```
tar xvf kubect1-cnpg-v1.25.3.tar
sudo mv ./kubect1-cnpg /usr/local/bin/kubect1-cnpg
sudo chmod +x /usr/local/bin/kubect1-cnpg
```

Проверить установку плагина возможно командой:

```
kubect1 cnpg version
```

Для примера можно выполнить команды для работы с jatoba кластером:

```
# Статус кластера
kubect1-cnpg -n $NSCLUSTER status cnpg-cluster-jatoba-cnpg-cluster

# Подключение к кластеру через клиент
kubect1-cnpg -n $NSCLUSTER psql cnpg-cluster-jatoba-cnpg-cluster

# Получение отчета по кластеру
kubect1-cnpg -n $NSCLUSTER report cluster cnpg-cluster-jatoba-cnpg-cluster
```

11.4. Гибернация

Гибернация позволяет временно отключить работающий кластер без потери конфигурации и данных.

Гибернация позволяет экономить ресурсы k3s кластера за счет безопасного выключения jatoba cnpg кластера.

После выключения jatoba cnpg кластера остаются только минимально необходимые ресурсы - тома с базой данных. Остальные ресурсы выключаются и включаются после вывода jatoba cnpg кластера из гибернации.

Поддерживается 2 режима гибернации:

— Императивная гибернация, когда остается только один том с данными (максимальная экономия ресурсов, но более длительный старт);

— Декларативная гибернация, когда остается несколько томов с данными, в соответствии с количеством экземпляров БД (экономия ресурсов меньше, но более быстрый старт).

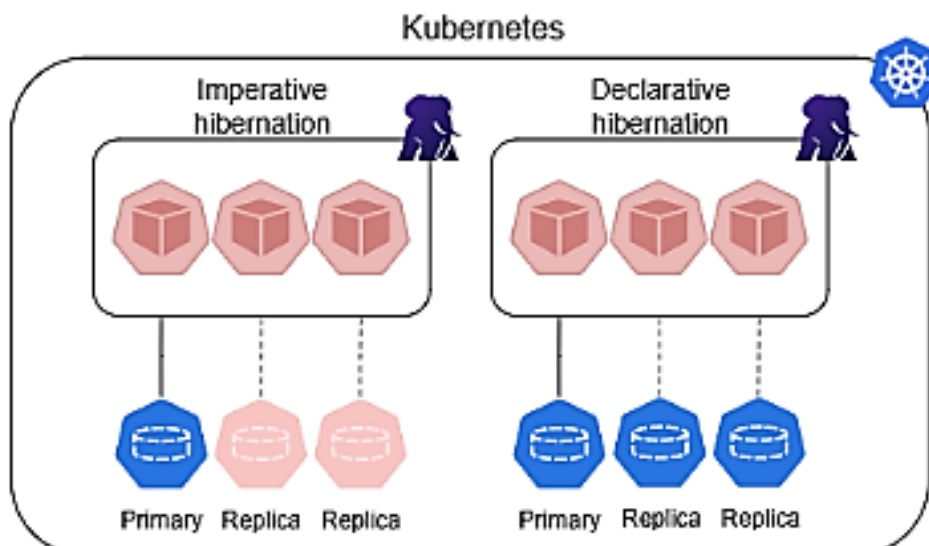


Рисунок 11.5 - Гибернация

11.4.1. Императивная гибернация.

Для работы с этим типом гибернации используется `kubectl-cnpg` plugin (описан в разделе 11.3).

После ввода кластера в гибернацию остается только один том с данными. Это максимальная экономия ресурсов, но для включения кластера потребуется более длительный период времени.

Для ввода в гибернацию необходимо выполнить следующие команды:

```
# Статус кластера
kubectl-cnpg -n $NSCLUSTER status cnpg-cluster-jatoba-cnpg-cluster

# Перевод кластера в режим гибернации
kubectl-cnpg -n $NSCLUSTER hibernate on cnpg-cluster-jatoba-cnpg-cluster

# Проверка статуса кластера в гибернации (обычный статус не работает в данной ситуации)
kubectl-cnpg -n $NSCLUSTER hibernate status cnpg-cluster-jatoba-cnpg-cluster
```

Для вывода кластера из гибернации необходимо выполнить следующие команды:

```
# Вывод из гибернации (+-60сек)
kubectl-cnpg -n $NSCLUSTER hibernate off cnpg-cluster-jatoba-cnpg-cluster

# Статус кластера
kubectl-cnpg -n $NSCLUSTER status cnpg-cluster-jatoba-cnpg-cluster
```

```
# Проверка что поды создаются и происходит инициализация (до нескольки  
х минут, в зависимости от объемов БД кластера)  
kubectl -n $NSCLUSTER get pods
```

11.4.2. Декларативная гибернация.

Для работы с этим типом гибернации используются k8s labels.

После ввода кластера в гибернацию остается тома с данными не только лидера, но и реплик. Экономия ресурсов меньше чем в императивной гибернации, но включение кластера происходит быстрее.

Для ввода в гибернацию необходимо выполнить следующие команды.

```
# Статус кластера  
kubectl-cnpg -n $NSCLUSTER status cnpg-cluster-jatoba-cnpg-cluster  
  
# Режим гибернации в данном случае определяется меткой кластера (label  
) cnpg.io/hibernation  
kubectl -n $NSCLUSTER annotate cluster cnpg-cluster-jatoba-cnpg-cluste  
r --overwrite cnpg.io/hibernation=on  
  
# Статус по метке - в гибернации  
kubectl -n $NSCLUSTER get cluster cnpg-cluster-jatoba-cnpg-cluster -o  
"jsonpath={.status.conditions[?(.type=="cnpg.io/hibernation\")]}"
```

Для вывода кластера из гибернации необходимо выполнить следующие команды:

```
# Переводим метку в режим "off"  
kubectl -n $NSCLUSTER annotate cluster cnpg-cluster-jatoba-cnpg-cluste  
r --overwrite cnpg.io/hibernation=off  
  
# Либо удалением метку гибернации  
kubectl -n $NSCLUSTER annotate cluster cnpg-cluster-jatoba-cnpg-cluste  
r cnpg.io/hibernation-  
  
# Проверка что поды создаются и происходит инициализация (до нескольки  
х минут, в зависимости от объемов БД кластера)  
kubectl -n $NSCLUSTER get pods
```

ПРИЛОЖЕНИЕ 1

Установка с предварительной загрузкой компонент K3s

Загрузка компонент для установки без доступа к сети Интернет (в примере используется версия v1.32.5-k3s1). Загрузка представлена в виде двух способов – загрузка через веб-браузер и загрузка с помощью команд из терминала.

1. Загрузка через веб-браузер

Предполагается что загрузка производится через веб-браузер от пользователя и по умолчанию все файлы загружены в каталог в домашней директории данного пользователя ~/Downloads.

Загрузка образов:

```
https://github.com/k3s-io/k3s/releases/download/v1.32.5%2Bk3s1/k3s-airgap-images-amd64.tar.zst
```

Загрузка бинарного файла K3s:

```
https://github.com/k3s-io/k3s/releases/download/v1.32.5%2Bk3s1/k3s
```

Загрузка скрипта установки K3s:

```
# Скопируйте текст файла и вставьте в файл с именем install.sh через любой удобный редактор текстовых файлов
https://raw.githubusercontent.com/k3s-io/k3s/refs/heads/release-1.32/install.sh
```

После загрузки всех файлов выполните следующие действия в терминале:

```
# Команды выполняются из терминала запущенного от обычного пользователя
sudo cp ~/Downloads/k3s-airgap-images-amd64.tar.zst /var/lib/rancher/k3s/agent/images/
sudo cp ~/Downloads/k3s /usr/local/bin/
sudo chmod +x /usr/local/bin/k3s
sudo chmod +x install.sh
```

2. Загрузка из терминала.

На каждом из узлов будущего кластера (серверы и агенты) выполните следующие команды:

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

```
export K3S_VER=1.32.5
export K3S_REV=k3s1
sudo mkdir -p /var/lib/rancher/k3s/agent/images/
sudo curl -L -o /var/lib/rancher/k3s/agent/images/k3s-airgap-images-amd64.tar.zst https://github.com/k3s-io/k3s/releases/download/v${K3S_VER}%2B${K3S_REV}/k3s-airgap-images-amd64.tar.zst
sudo curl -L -o /usr/local/bin/k3s https://github.com/k3s-io/k3s/releases/download/v${K3S_VER}%2B${K3S_REV}/k3s
sudo chmod +x /usr/local/bin/k3s
curl -L https://get.k3s.io/ > install.sh
chmod +x install.sh
```

Команды установки.

После загрузки одним из перечисленных выше способов компонент K3s выполните следующие команды установки кластера K3s.

На узле сервера выполните команды:

```
# Генерация токена сервера
K3S_SERVER_TOKEN=$(head -5 /dev/urandom | base64 | tr -dc 'a-zA-Z0-9' | head -c 25)
# Запуск скрипта установки в режиме «без загрузки»
INSTALL_K3S_SKIP_DOWNLOAD=true INSTALL_K3S_EXEC="server" K3S_TOKEN=$K3S_SERVER_TOKEN ./install.sh
# Получение токена подключения агентов
K3S_JOIN_TOKEN=$(sudo cat /var/lib/rancher/k3s/server/node-token)
# Вывод токена подключения агентов для копирования на агенты
echo $K3S_JOIN_TOKEN
# Проверка службы сервера
systemctl status k3s.service
```

На агентах выполните следующие команды:

```
# Экспорт значения токена сервера на узле агента
export K3S_JOIN_TOKEN=<value_from_server>
# Запуск скрипта установки в режиме «без загрузки»
INSTALL_K3S_SKIP_DOWNLOAD=true INSTALL_K3S_EXEC="agent --server https://<IP_master_node>:6443" K3S_TOKEN=$K3S_JOIN_TOKEN ./install.sh
# Проверка службы агента
systemctl status k3s-agent.service
```

После завершения всех команд и проверки служб перейдите к разделу 4.2 данного руководства.

ПРИЛОЖЕНИЕ 2

Файл конфигурации кластера ./values/jatoba-default-values.yaml

Параметры кластера, используемые по умолчанию.

```
# -- Override the name of the chart
nameOverride: ""
# -- Override the full name of the chart
fullnameOverride: ""
# -- Override the namespace of the chart
namespaceOverride: ""

###
# -- Type of the CNPG database. Available types:
# * `postgresql`
# * `postgis`
# * `timescaledb`
type: postgresql

version:
# -- PostgreSQL major version to use
postgresql: "16"
# -- If using TimescaleDB, specify the version
timescaledb: "2.15"
# -- If using PostGIS, specify the version
postgis: "3.4"

###
# -- Cluster mode of operation. Available modes:
# * `standalone` - default mode. Creates new or updates an existing CNPG cluster.
# * `replica` - Creates a replica cluster from an existing CNPG cluster. # TODO
# * `recovery` - Same as standalone but creates a cluster from a backup, object store or via pg_basebackup.
mode: standalone

recovery:
##
# -- Available recovery methods:
# * `backup` - Recovers a CNPG cluster from a CNPG backup (PITR supported) Needs to be on the same cluster in the same namespace.
# * `object_store` - Recovers a CNPG cluster from a barman object store (PITR supported).
# * `pg_basebackup` - Recovers a CNPG cluster via streaming replication protocol. Useful if you want to migrate databases to CloudNativePG, even from outside Kubernetes. # TODO
method: backup

## -- Point in time recovery target. Specify one of the following:
pitrTarget:
# -- Time in RFC3339 format
time: ""

##
# -- Backup Recovery Method
backupName: "" # Name of the backup to recover from. Required if method is `backup`.

##
# -- The original cluster name when used in backups. Also known as serverName.
clusterName: ""
# -- Overrides the provider specific default endpoint. Defaults to:
# S3: https://s3.<region>.amazonaws.com"
# Leave empty if using the default S3 endpoint
endpointURL: ""
# -- Specifies a CA bundle to validate a privately signed certificate.
endpointCA:
# -- Creates a secret with the given value if true, otherwise uses an existing secret.
```

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

```

create: false
name: ""
key: ""
value: ""
# -- Overrides the provider specific default path. Defaults to:
# S3: s3://<bucket><path>
# Azure: https://<storageAccount>.<serviceName>.core.windows.net/<containerName><path>
# Google: gs://<bucket><path>
destinationPath: ""
# -- One of `s3`, `azure` or `google`
provider: s3
s3:
  region: ""
  bucket: ""
  path: "/"
  accessKey: ""
  secretKey: ""
  # -- Use the role based authentication without providing explicitly the keys
  inheritFromIAMRole: false
azure:
  path: "/"
  connectionString: ""
  storageAccount: ""
  storageKey: ""
  storageSasToken: ""
  containerName: ""
  serviceName: blob
  inheritFromAzureAD: false
google:
  path: "/"
  bucket: ""
  gkeEnvironment: false
  applicationCredentials: ""
secret:
  # -- Whether to create a secret for the backup credentials
  create: true
  # -- Name of the backup credentials secret
  name: ""

# See https://cloudnative-pg.io/documentation/1.22/bootstrap/#bootstrap-from-a-live-cluster-
pg_basebackup
pgBaseBackup:
  # -- Name of the database used by the application. Default: `app`.
  database: app
  # -- Name of the owner of the database in the instance to be used by applications. Default
s to the value of the `database` key.
  secret: ""
  # -- Name of the secret containing the initial credentials for the owner of the user datab
ase. If empty a new secret will be created from scratch
  owner: ""
  source:
    host: ""
    port: 5432
    username: ""
    database: "app"
    sslMode: "verify-full"
  passwordSecret:
    # -- Whether to create a secret for the password
    create: false
    # -- Name of the secret containing the password
    name: ""
    # -- The key in the secret containing the password
    key: "password"
    # -- The password value to use when creating the secret
    value: ""
  sslKeySecret:
    name: ""
    key: ""
  sslCertSecret:

```

```

    name: ""
    key: ""
    sslRootCertSecret:
      name: ""
      key: ""

cluster:
  # -- Number of instances
  instances: 3

  # -- Name of the container image, supporting both tags (<image>:<tag>) and digests for deterministic and repeatable deployments:
  # <image>:<tag>@sha256:<digestValue>
  imageName: "" # Default value depends on type (postgresql/postgis/timescaledb)

  # -- Reference to `ImageCatalog` of `ClusterImageCatalog`, if specified takes precedence over `cluster.imageName`
  imageCatalogRef: {}
    # kind: ImageCatalog
    # name: postgresql

  # -- Image pull policy. One of Always, Never or IfNotPresent. If not defined, it defaults to IfNotPresent. Cannot be updated.
  # More info: https://kubernetes.io/docs/concepts/containers/images#updating-images
  imagePullPolicy: IfNotPresent

  # -- The list of pull secrets to be used to pull the images.
  # See: https://cloudnative-pg.io/documentation/current/cloudnative-pg.v1/#postgresql-cnpg-io-v1-LocalObjectReference
  imagePullSecrets: []

  storage:
    size: 2Gi
    storageClass: ""

  walStorage:
    enabled: false
    size: 1Gi
    storageClass: ""

  # -- The UID of the postgres user inside the image, defaults to 26
  postgresUID: -1

  # -- The GID of the postgres user inside the image, defaults to 26
  postgresGID: -1

  # -- Customization of service definitions. Please refer to https://cloudnative-pg.io/documentation/1.24/service_management/
  services: {}

  # -- Resources requirements of every generated Pod.
  # Please refer to https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ for more information.
  # We strongly advise you use the same setting for limits and requests so that your cluster pods are given a Guaranteed QoS.
  # See: https://kubernetes.io/docs/concepts/workloads/pods/pod-qos/
  resources: {}
    # limits:
    #   cpu: 2000m
    #   memory: 8Gi
    # requests:
    #   cpu: 2000m
    #   memory: 8Gi

  priorityClassName: ""

  # -- Method to follow to upgrade the primary server during a rolling update procedure, after all replicas have been

```

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

```
# successfully updated. It can be switchover (default) or restart.
primaryUpdateMethod: switchover

# -- Strategy to follow to upgrade the primary server during a rolling update procedure, after all replicas have been
# successfully updated: it can be automated (unsupervised - default) or manual (supervised)
primaryUpdateStrategy: unsupervised

# -- The instances' log level, one of the following values: error, warning, info (default), debug, trace
logLevel: "info"

# -- Affinity/Anti-affinity rules for Pods.
# See: https://cloudnative-pg.io/documentation/current/cloudnative-pg.v1/#postgresql-cnpg-io-v1-AffinityConfiguration
affinity:
  topologyKey: topology.kubernetes.io/zone

# -- The configuration for the CA and related certificates.
# See: https://cloudnative-pg.io/documentation/current/cloudnative-pg.v1/#postgresql-cnpg-io-v1-CertificatesConfiguration
certificates: {}

# -- When this option is enabled, the operator will use the SuperuserSecret to update the postgres user password.
# If the secret is not present, the operator will automatically create one.
# When this option is disabled, the operator will ignore the SuperuserSecret content, delete it when automatically created,
# and then blank the password of the postgres user by setting it to NULL.
enableSuperuserAccess: true
superuserSecret: ""

# -- Allow to disable PDB, mainly useful for upgrade of single-instance clusters or development purposes
# See: https://cloudnative-pg.io/documentation/current/kubernetes_upgrade/#pod-disruption-budgets
enablePDB: true

# -- This feature enables declarative management of existing roles, as well as the creation of new roles if they are not
# already present in the database.
# See: https://cloudnative-pg.io/documentation/current/declarative_role_management/
roles: []
# - name: dante
#   ensure: present
#   comment: Dante Alighieri
#   login: true
#   superuser: false
#   inRoles:
#     - pg_monitor
#     - pg_signal_backend

monitoring:
# -- Whether to enable monitoring
enabled: false
podMonitor:
# -- Whether to enable the PodMonitor
enabled: true
# --The list of relabelings for the PodMonitor.
# Applied to samples before scraping.
relabelings: []
# -- The list of metric relabelings for the PodMonitor.
# Applied to samples before ingestion.
metricRelabelings: []
prometheusRule:
# -- Whether to enable the PrometheusRule automated alerts
enabled: true
# -- Exclude specified rules
excludeRules: []
```



```
# - CNPGClusterZoneSpreadWarning
# -- Whether the default queries should be injected.
# Set it to true if you don't want to inject default queries into the cluster.
disableDefaultQueries: false
# -- Custom Prometheus metrics
# Will be stored in the ConfigMap
customQueries: []
# - name: "pg cache hit ratio"
#   query: "SELECT current_database() as datname, sum(heap_blks_hit) / (sum(heap_blks_hit)
) + sum(heap_blks_read)) as ratio FROM pg_statio_user_tables;"
#   metrics:
#     - datname:
#       usage: "LABEL"
#       description: "Name of the database"
#     - ratio:
#       usage: GAUGE
#       description: "Cache hit ratio"
# -- The list of secrets containing the custom queries
customQueriesSecret: []
# - name: custom-queries-secret
#   key: custom-queries

postgresql:
# -- PostgreSQL configuration options (postgresql.conf)
parameters:
# enable pgaudit
pgaudit.log: "none"
pgaudit.log_catalog: "off"
pgaudit.log_parameter: "on"
pgaudit.log_relation: "on"
# enable pg_stat statements
pg_stat_statements.max: "1000"
pg_stat_statements.track: all
# default Jatoba values
archive_mode: 'on'
archive_timeout: 5min
dynamic_shared_memory_type: posix
full_page_writes: 'on'
log_destination: csvlog
log_directory: /controller/log
log_filename: postgres
log_rotation_age: '0'
log_rotation_size: '0'
log_truncate_on_rotation: 'false'
logging_collector: 'on'
max_parallel_workers: '32'
max_replication_slots: '32'
max_worker_processes: '32'
shared_memory_type: mmap
ssl_max_protocol_version: TLSv1.3
ssl_min_protocol_version: TLSv1.3
wal_keep_size: 512MB
wal_level: logical
wal_log_hints: 'on'
wal_receiver_timeout: 5s
wal_sender_timeout: 5s
max_connections: '300'
log_min_messages: 'warning'
log_min_error_statement: 'error'
log_checkpoints: 'True'
log_connections: 'True'
log_disconnections: 'True'
log_duration: 'False'
log_hostname: 'True'
log_statement: 'mod'
log_lock_waits: 'False'
log_error_verbosity: verbose
log_temp_files: '-1'
log_replication_commands: 'False'
```

```

log_timezone: 'Europe/Moscow'
log_line_prefix: '|%m|%d|%u|%a|%r|%h|%p|%e|%c|%x|'
wal_sender_check_crc: 'off'
wal_sender_panic_on_crc_error: 'off'
std_fuzz_factor: '1.01'
log_mask_password: 'on'
autovacuum_freeze_max_age: '10000000000'
autovacuum_multixact_freeze_max_age: '20000000000'
default_with_oids: 'off' # not set in pg_settings, only "show default_with_oids"
#unix_socket_directories: /controller/run
synchronous: {}
# method: any
# number: 1
# -- PostgreSQL Host Based Authentication rules (lines to be appended to the pg_hba.conf f
ile)
pg_hba: []
# - host all all 10.244.0.0/16 md5
# -- PostgreSQL User Name Maps rules (lines to be appended to the pg_ident.conf file)
pg_ident: []
# - mymap /^(.*)@mydomain\.com$ \1
# -- Lists of shared preload libraries to add to the default ones
# pgaudit and pg_stat_statements - operator managed extensions, auto create if exist para
meters in settings
# https://cloudnative-pg.io/documentation/1.25/postgresql_conf/#managed-extensions
shared_preload_libraries:
- vector

# -- BootstrapInitDB is the configuration of the bootstrap process when initdb is used.
# See: https://cloudnative-pg.io/documentation/current/bootstrap/
# See: https://cloudnative-pg.io/documentation/current/cloudnative-pg.v1/#postgresql-cnpg-io
-v1-bootstrapinitdb
initdb:
database: app
owner: "" # Defaults to the database name
secret:
name: "" # Name of the secret containing the initial credentials for the owner of the us
er database. If empty a new secret will be created from scratch
options: []
encoding: UTF8
localeCollate: en_US.UTF-8
localeCTYPE: en_US.UTF-8
postInitSQL: []
# pgaudit and pg_stat_statements - operator managed extensions, auto create if exist para
meters in settings
# https://cloudnative-pg.io/documentation/1.25/postgresql_conf/#managed-extensions
postInitApplicationSQL:
- create extension if not exists dblink;
- create extension if not exists ltree;
- create extension if not exists pgcrypto;
- create extension if not exists postgres_fdw;
- create extension if not exists fuzzystrmatch;
- create extension if not exists lo;
- create extension if not exists pg_trgm;
- create extension if not exists tablefunc;
- create extension if not exists xml2;
- create extension if not exists adminpack;
- create extension if not exists plpython3u;
- create extension if not exists postgis;
- create extension if not exists postgis_tiger_geocoder cascade;
- create extension if not exists postgis_topology cascade;
- create extension if not exists postgis_raster cascade;
- create extension if not exists vector;
postInitTemplateSQL: []

# -- Configure the metadata of the generated service account
serviceAccountTemplate: {}

additionalLabels: {}
annotations: {}

```

```
backups:
# -- You need to configure backups manually, so backups are disabled by default.
enabled: false

# -- Overrides the provider specific default endpoint. Defaults to:
# S3: https://s3.<region>.amazonaws.com"
endpointURL: "http://minio.minio.svc.cluster.local:9000" # Leave empty if using the default
S3 endpoint
# -- Specifies a CA bundle to validate a privately signed certificate.
endpointCA:
# -- Creates a secret with the given value if true, otherwise uses an existing secret.
create: false
name: ""
key: ""
value: ""

# -- Overrides the provider specific default path. Defaults to:
# S3: s3://<bucket><path>
# Azure: https://<storageAccount>.<serviceName>.core.windows.net/<containerName><path>
# Google: gs://<bucket><path>
destinationPath: "s3://cnpg/cnpg-default"
# -- One of `s3`, `azure` or `google`
provider: s3
s3:
  region: ""
  bucket: "cnpg"
  path: "/"
  accessKey: ""
  secretKey: ""
  # -- Use the role based authentication without providing explicitly the keys
  inheritFromIAMRole: false
azure:
  path: "/"
  connectionString: ""
  storageAccount: ""
  storageKey: ""
  storageSasToken: ""
  containerName: ""
  serviceName: blob
  inheritFromAzureAD: false
google:
  path: "/"
  bucket: ""
  gkeEnvironment: false
  applicationCredentials: ""
secret:
# -- Whether to create a secret for the backup credentials
create: true
# -- Name of the backup credentials secret
name: "cnpg-cluster-backup-secret"

wal:
# -- WAL compression method. One of `` (for no compression), `gzip`, `bzip2` or `snappy`.
compression: gzip
# -- Whether to instruct the storage provider to encrypt WAL files. One of `` (use the sto
rage container default), `AES256` or `aws:kms`.
encryption: ""
# -- Number of WAL files to be archived or restored in parallel.
maxParallel: 1
data:
# -- Data compression method. One of `` (for no compression), `gzip`, `bzip2` or `snappy`.
compression: gzip
# -- Whether to instruct the storage provider to encrypt data files. One of `` (use the st
orage container default), `AES256` or `aws:kms`.
encryption: ""
# -- Number of data files to be archived or restored in parallel.
jobs: 2
```

```
scheduledBackups:
-
  # -- Scheduled backup name
  name: daily-backup
  # -- Schedule in cron format
  schedule: "0 0 0 * * *"
  # -- Backup owner reference
  backupOwnerReference: self
  # -- Backup method, can be `barmanObjectStore` (default) or `volumeSnapshot`
  method: barmanObjectStore

# -- Retention policy for backups
retentionPolicy: "30d"

imageCatalog:
  # -- Whether to provision an image catalog. If imageCatalog.images is empty this option will
  # be ignored.
  create: true
  # -- List of images to be provisioned in an image catalog.
  images:
    - image: jatoba/cloudnative-pg/jatoba:6.10.1-57608-df1.6.3-redos7.3-comm
      major: 16

# -- List of PgBouncer poolers
poolers: []
# -
# # -- Pooler name
# name: rw
# # -- PgBouncer type of service to forward traffic to.
# type: rw
# # -- PgBouncer pooling mode
# poolMode: transaction
# # -- Number of PgBouncer instances
# instances: 1
# # -- PgBouncer configuration parameters
# parameters:
#   max_client_conn: "50"
#   default_pool_size: "10"
# monitoring:
#   # -- Whether to enable monitoring
#   enabled: false
#   podMonitor:
#     # -- Whether to enable the PodMonitor
#     enabled: false
# # -- Custom PgBouncer deployment template.
# # Use to override image, specify resources, etc.
# template:
#   spec:
#     containers:
#       - image: jatoba/cloudnative-pg/japooler:6.10.1-redos7.3
#         name: pgbouncer
# -
# # -- Pooler name
# name: ro
# # -- PgBouncer type of service to forward traffic to.
# type: ro
# # -- PgBouncer pooling mode
# poolMode: transaction
# # -- Number of PgBouncer instances
# instances: 1
# # -- PgBouncer configuration parameters
# parameters:
#   max_client_conn: "50"
#   default_pool_size: "10"
# monitoring:
#   # -- Whether to enable monitoring
#   enabled: false
#   podMonitor:
```

```
#      # -- Whether to enable the PodMonitor
#      enabled: false
#      # -- Custom PgBouncer deployment template.
#      # Use to override image, specify resources, etc.
#      template:
#      spec:
#      containers:
#      - image: jatoba/cloudnative-pg/japooler:6.10.1-redos7.3
#      name: pgbouncer
```

ПРИЛОЖЕНИЕ 3

Файл конфигурации кластера ./values/values-opt/jatoba-bootstrap.yaml

Параметры кластера, используемые для конфигурирования базы во время создания кластера.

```
---
cluster:
  # Инициализация новой БД и её параметры - имя, владелец, секрет из которого будет взят пароль
  # для пользователя (если секрет заранее создан)
  initdb:
    database: app
    owner: ""
    secret:
      name: ""
  # Параметры кодировки и сортировки кластера БД по умолчанию
  encoding: UTF8
  localeCollate: en_US.UTF-8
  localeCTYPE: en_US.UTF-8
  # Скрипты выполняемые после создания БД от имени суперпользователя, не переключают контекст
  postInitSQL: []
  # Скрипты выполняемые после создания БД от имени пользователя, относятся к создаваемой БД
  postInitApplicationSQL:
    - create extension if not exists dblink;
    - create extension if not exists ltree;
    - create extension if not exists pgcrypto;
    - create extension if not exists postgres_fdw;
    - create extension if not exists fuzzystrmatch;
    - create extension if not exists lo;
    - create extension if not exists pg_trgm;
    - create extension if not exists tablefunc;
    - create extension if not exists xml2;
    - create extension if not exists adminpack;
    - create extension if not exists plpython3u;
    - create extension if not exists postgis;
    - create extension if not exists postgis_tiger_geocoder cascade;
    - create extension if not exists postgis_topology cascade;
    - create extension if not exists postgis_raster cascade;
    - create extension if not exists vector;
    - create table test (
      id text primary key,
      column1 text NOT NULL,
      column2 text NOT NULL,
      column3 text NOT NULL
    );
    - insert into test (
      id, column1, column2, column3
    )
      select
        left(md5(i::text), 10),
        md5(random()::text),
        md5(random()::text),
        left(md5(random()::text), 4)
      from generate_series(1, 10000) s(i);
    - grant all privileges on table test to app;
  postInitTemplateSQL: []
```

ПРИЛОЖЕНИЕ 4

Файл конфигурации кластера ./values/values-opt/jatoba-backup.yaml

Параметры кластера, используемые для создания бекапов.

```
---
# Конфигурация резервного копирования кластера
backups:
  # Включение конфигурации
  enabled: true
  # URL S3 к которому будет подключаться механизм резервных копий и сохранять там журналы и копии базы
  endpointURL: "http://minio.minio.svc.cluster.local:9000"
  # Задать сертификат для S3, если используется https с самоподписанным или сертификатом внутреннего PKI
  endpointCA:
    create: false
    name: ""
    key: ""
    value: ""
  # Путь по которому будут сохраняться копии, состоит из типа провайдера, имени бакета и каталога
  destinationPath: "s3://cnpg/cnpg-default"
  # Определение провайдера хранилища резервных копий (s3, google, azue)
  provider: s3
  # Конфигурация под указанного провайдера - s3
  s3:
    region: ""
    bucket: "cnpg"
    path: "/"
    # Ключ доступа/имя пользователя
    # рекомендуем accessKey задавать через --set backups.s3.accessKey
    accessKey: ""
    # Секретный ключ/пароль
    # рекомендуем secretKey задавать через --set backups.s3.secretKey
    secretKey: ""
    inheritFromIAMRole: false
  # Создаваемый секрет для сохранения учетных данных подключения к провайдеру хранилища
  secret:
    create: true
    name: "cnpg-cluster-backup-secret"
  # Указание параметров резервного копирования WAL
  wal:
    compression: gzip
    encryption: ""
    maxParallel: 1
  # Указание параметров резервного копирования DATA файлов
  data:
    compression: gzip
    encryption: ""
    jobs: 2

  # Расписание резервного копирования
  # Задача запускается сразу, это не изменяемый параметр,
  # в случае инициализации пустой БД может выдать ошибку, после наполнения БД, пройдет штатно по расписани
ю
  scheduledBackups:
    -
      name: daily-backup
      # cron-формат, но первое число - секунды
      schedule: "0 0 0 * * *"
      backupOwnerReference: self
      # Тип резервной копии по умолчанию, возможно использовать VolumeSnapshot, но требуется поддержка и н
астройка со стороны CSI
      # VolumeSnapshot - incremental and differential block level backup (требуется поддержка и настройка
со стороны CSI)
      # barmanObjectStore - online/hot backup с использование инструмента Barman (требуется обязательное п
одключение к хранилищу резервных копий. См. provider)
      method: barmanObjectStore
      # Период хранения резервных копий
      retentionPolicy: "7d"
```

ПРИЛОЖЕНИЕ 5

Файл конфигурации кластера ./values/values-opt/jatoba-recovery.yaml

Параметры кластера, используемые для восстановления из бекапов.

```
---
# При восстановлении не используется initdb (bootstrap), поэтому важно указать режим кластера "recovery"
mode: recovery

recovery:
  # Метод восстановления из заранее созданной резервной копии (поддерживается режим восстановления из друго
  # го кластера методом репликации "pg basebackup")
  method: object_store
  # Восстановление на конкретную точку времени
  pitrTarget:
    time: ""
  # Имя кластера в резервной копии из которой восстанавливается текущий кластер
  clusterName: "cnpg-cluster"
  # URL S3 из которого будет взята резервная копия (параметры endpointCA аналогичны jatoba-backup.yaml)
  endpointURL: "http://minio.minio.svc.cluster.local:9000"
  # Путь по которому расположена резервная копия необходимая для восстановления
  destinationPath: "s3://cnpg/cnpg-default"
  # Провайдер и параметры подключения к эндпоинту
  provider: s3
  # Конфигурация под указанного провайдера - s3
  s3:
    region: ""
    bucket: "cnpg"
    path: "/"
    # Ключ доступа/имя пользователя
    # рекомендуем accessKey задавать через --set backups.s3.accessKey
    accessKey: ""
    # Секретный ключ/пароль
    # рекомендуем secretKey задавать через --set backups.s3.secretKey
    secretKey: ""
    inheritFromIAMRole: false
  secret:
    create: true
    name: ""

# Параметры для репликации существующего кластера (перенос данных в CNPG) если метод "method: pg_basebac
kup"
pgBaseBackup:
  # Имя создаваемой базы в CNPG
  database: app
  # Секрет содержащий данные подключения к БД
  secret: ""
  # Владелец будущей базы
  owner: ""
  # Источник к которому будет произведено подключения для переноса данных из источника в CNPG
  source:
    host: ""
    port: 5432
    username: ""
    database: "app"
    sslMode: "disable"
  # Заранее созданный секрет с паролем для подключения к источнику
  passwordSecret:
    create: false
    name: ""
    key: "password"
    value: ""
```


ПРИЛОЖЕНИЕ 6

Файл конфигурации кластера ./ values/values-opt/jatoba-parameters.yaml

Параметры кластера, используемые для конфигурирования postgresql.conf, pg_hba, pg_ident файлов кластера.

```
---
cluster:
  postgresql:
    # Задать значения для postgresql.conf
    # Фиксированные параметры менять нельзя, список фиксированных параметров - https://cloudnative-pg.io/documentation/1.
    25/postgresql conf/#fixed-parameters
    parameters:
      # enable pgaudit
      pgaudit.log: "none"
      pgaudit.log_catalog: "off"
      pgaudit.log_parameter: "on"
      pgaudit.log_relation: "on"
      # enable pg_stat_statements
      pg_stat_statements.max: "1000"
      pg_stat_statements.track: all
      # default Jatoba values
      archive mode: 'on'
      archive timeout: 5min
      dynamic_shared_memory_type: posix
      full_page_writes: 'on'
      log_destination: csvlog
      log_directory: /controller/log
      log_filename: postgres
      log rotation age: '0'
      log rotation size: '0'
      log truncate on rotation: 'false'
      logging_collector: 'on'
      max_parallel_workers: '32'
      max_replication_slots: '32'
      max_worker_processes: '32'
      shared_memory_type: mmap
      ssl max protocol version: TLSv1.3
      ssl min protocol version: TLSv1.3
      wal_keep_size: 512MB
      wal_level: logical
      wal_log_hints: 'on'
      wal_receiver_timeout: 5s
      wal_sender_timeout: 5s
      max_connections: '300'
      log_min_messages: 'warning'
      log_min_error_statement: 'error'
      log_checkpoints: 'True'
      log_connections: 'True'
      log_disconnections: 'True'
      log_duration: 'False'
      log_hostname: 'True'
      log_statement: 'mod'
      log_lock_waits: 'False'
      log_error_verbosity: verbose
      log temp files: '-1'
      log replication commands: 'False'
      log_timezone: 'Europe/Moscow'
      log_line_prefix: '%m%d%u%a%r%h%p%e%c|x|'
      wal_sender_check_crc: 'off'
      wal_sender_panic_on_crc_error: 'off'
      std_fuzz_factor: '1.01'
      log_mask_password: 'on'
      autovacuum freeze max age: '10000000000'
      autovacuum multixact freeze max age: '20000000000'
      default with oids: 'off' # not set in pg settings, only "show default with oids"
      #unix_socket_directories: /controller/run
    # Определение доступов к БД на основе pg_hba.conf
    pg_hba: []
    # Определение сопоставление пользователей, например для доменной авторизации
    pg_ident: []
    # Указание общих библиотек расширений
    shared_preload_libraries:
      - vector
```

ПРИЛОЖЕНИЕ 7

Файл конфигурации кластера ./values/values-opt/jatoba-poolers.yaml

Параметры кластера, используемые для подключения дополнительных подов кластера – менеджеров соединений - japooleer.

```
---
# Включение пуллеров в установку кластера
poolers:
  # Имя, тип пуллера, кол-во экземпляров, отдельные параметры (если требуются)
  # Read-Write пуллер
  - name: rw
    type: rw
    poolMode: transaction
    instances: 1
    parameters:
      max_client_conn: "50"
      default_pool_size: "10"
  # Образ для пуллера задается через шаблон спецификации манифеста, меняется только image если необходимо
  template:
    spec:
      containers:
        - image: jatoba/cloudnative-pg/japooleer:6.10.1-redos7.3
          name: pgbouncer
  # Read-Only пуллер
  - name: ro
    type: ro
    poolMode: transaction
    instances: 1
    parameters:
      max_client_conn: "50"
      default_pool_size: "10"
  # Образ для пуллера задается через шаблон спецификации манифеста, меняется только image если необходимо
  template:
    spec:
      containers:
        - image: jatoba/cloudnative-pg/japooleer:6.10.1-redos7.3
          name: pgbouncer
```

ПРИЛОЖЕНИЕ 8

Файл конфигурации кластера ./values/values-opt/monitoring.yaml

Параметры кластера, используемые для мониторинга кластера.

```
---
# Для визуализации метрик используйте Prometheus и Grafana (CloudNativePG dashboard ID - 20417).
# Инструкции и параметры по установке Prometheus и Grafana не входят в данный yaml.
# см. values-prom-stack.yaml

# Параметры мониторинга спрг задаваемые для экземпляров БД через podMonitor
cluster:
  monitoring:
    enabled: true
    podMonitor:
      enabled: true
    prometheusRule:
      enabled: true
      # Включение/отключение запросов по умолчанию к БД для формирования метрик (по умолчанию включено, пара
метр false)
      disableDefaultQueries: false
      # Заранее сформированный список запросов к БД если требуются дополнительные или более уникальные метри
ки
      customQueries: []

# Параметры мониторинга для пуллеров - вкл/выкл
# По умолчанию используется образ pgbouncer от CloudNativePG
# Образы указываются через template, или при подключении файл jatoba-poolers.yaml при установке кластера
poolers:
  - name: rw
    monitoring:
      enabled: true
    podMonitor:
      enabled: true
  - name: ro
    monitoring:
      enabled: true
    podMonitor:
      enabled: true
# Включение пуллеров в установку кластера
poolers:
  # Имя, тип пуллера, кол-во экземпляров, отдельные параметры (если требуются)
  # Read-Write пуллер
  - name: rw
    type: rw
    poolMode: transaction
    instances: 1
    parameters:
      max_client_conn: "50"
      default_pool_size: "10"
  # Образ для пуллера задается через шаблон спецификации манифеста, меняется только image если необходимо
  template:
    spec:
```

ПРИЛОЖЕНИЕ 9

Файл конфигурации кластера jatoba-resources-minimal.yaml

```
---
# Ресурсы для работы кластера Jatoba CNPG на минимальных требованиях

# Конечные ресурсы кластера Kubernetes следует рассчитывать исходя из компонентов, размещенных на хостах (мастер/воркер) и требований для размещения дополнительных сервисов
# Минимальные требования:
# - Компоненты Kubernetes (мастер/control plane): 2vCPU, 4Gb RAM
# - Компоненты Kubernetes (воркер): 2vCPU, 2Gb RAM (или из расчета 1vCPU = 2Gb RAM)
# Рекомендации для продуктовых сред:
# - 4vCPU 8Gb RAM для мастер узлов
# - 1vCPU на каждые 4GB памяти рабочих узлов (при размещении дополнительных сервисов)

# Итого под текущую конфигурацию БД, и кластера Kubernetes, не считая другой нагрузки:
# мастер узел - 6vCPU, 12Gb RAM
# рабочий узел - 4vCPU, 6Gb RAM
cluster:
  resources:
    limits:
      cpu: 2
      memory: 4Gi
    requests:
      cpu: 1
      memory: 2Gi
  postgresql:
    # Задать значения для postgresql.conf в зависимости от установленных лимитов CPU и memory
    # Фиксированные параметры менять нельзя, список фиксированных параметров - https://cloudnative-pg.io/documentation/1.25/postgresql_conf/#fixed-parameters
    parameters:
      # DB Version: 16
      # OS Type: linux
      # DB Type: mixed
      # Total Memory (RAM): 4 GB
      # CPUs num: 2
      # Connections num: 150
      # Data Storage: ssd
      max_connections: "100"
      shared_buffers: 1GB
      effective_cache_size: 3GB
      maintenance_work_mem: 256MB
      checkpoint_completion_target: "0.9"
      wal_buffers: 16MB
      default_statistics_target: "100"
      random_page_cost: "1.1"
      effective_io_concurrency: "200"
      work_mem: 3318kB
      huge_pages: "off"
      min_wal_size: 1GB
      max_wal_size: 4GB
```

№ изменения: _____

Подпись отв. лица: _____

Дата внесения изм: _____

Файл конфигурации кластера jatoba-resources-recommend.yaml

```
---
# Ресурсы для работы кластера Jatoba CNPG на рекомендуемых требованиях

# Конечные ресурсы кластера Kubernetes следует рассчитывать исходя из компонентов размещенных на хостах (мастер/воркер) и требований для размещения дополнительных сервисов
# Минимальные требования:
# - Компоненты Kubernetes (мастер/control plane): 2vCPU, 4Gb RAM
# - Компоненты Kubernetes (воркер): 2vCPU, 2Gb RAM (или из расчета 1vCPU = 2Gb RAM)
# Рекомендации для продуктовых сред:
# - 4vCPU 8Gb RAM для мастер узлов
# - 1vCPU на каждые 4GB памяти рабочих узлов (при размещении дополнительных сервисов)

# Итого под текущую конфигурацию БД, и кластера Kubernetes, не считая другой нагрузки
:
# мастер узел - 8vCPU, 24Gb RAM
# рабочий узел - 6vCPU, 18Gb RAM
cluster:
  resources:
    limits:
      cpu: 4
      memory: 16Gi
    requests:
      cpu: 4
      memory: 8Gi
  postgresql:
    # Задать значения для postgresql.conf в зависимости от установленных лимитов CPU
и memory
    # Фиксированные параметры менять нельзя, список фиксированных параметров - https://cloudnative-pg.io/documentation/1.25/postgresql\_conf/#fixed-parameters
    parameters:
      # DB Version: 16
      # OS Type: linux
      # DB Type: mixed
      # Total Memory (RAM): 16 GB
      # CPUs num: 4
      # Connections num: 300
      # Data Storage: ssd
      max_connections: "300"
      shared_buffers: 4GB
      effective_cache_size: 12GB
      maintenance_work_mem: 1GB
      checkpoint_completion_target: "0.9"
      wal_buffers: 16MB
      default_statistics_target: "100"
      random_page_cost: "1.1"
      effective_io_concurrency: "200"
      work_mem: 6898kB
      huge_pages: "off"
      min_wal_size: 1GB
      max_wal_size: 4GB
      max_worker_processes: "4"
      max_parallel_workers_per_gather: "2"
      max_parallel_workers: "4"
      max_parallel_maintenance_workers: "2"
```

Файл конфигурации кластера jatoba-resources-large.yaml

```
---
# Ресурсы для работы кластера Jatoba CNPG на больших мощностях

# Конечные ресурсы кластера Kubernetes следует рассчитывать исходя из компонентов размещенных на хостах (мастер/воркер) и требований для размещения дополнительных сервисов
# Минимальные требования:
# - Компоненты Kubernetes (мастер/control plane): 2vCPU, 4Gb RAM
# - Компоненты Kubernetes (воркер): 2vCPU, 2Gb RAM (или из расчета 1vCPU = 2Gb RAM)
# Рекомендации для продуктовых сред:
# - 4vCPU 8Gb RAM для мастер узлов
# - 1vCPU на каждые 4GB памяти рабочих узлов (при размещении дополнительных сервисов)

# Итого под текущую конфигурацию БД, и кластера Kubernetes, не считая другой нагрузки
:
# мастер узел - 20vCPU, 72Gb RAM
# рабочий узел - 18vCPU, 68Gb RAM
cluster:
  resources:
    limits:
      cpu: 16
      memory: 64Gi
    requests:
      cpu: 16
      memory: 64Gi
  postgresql:
    # Задать значения для postgresql.conf в зависимости от установленных лимитов CPU
и memory
    # Фиксированные параметры менять нельзя, список фиксированных параметров - https://cloudnative-pg.io/documentation/1.25/postgresql\_conf/#fixed-parameters
    parameters:
      # DB Version: 16
      # OS Type: linux
      # DB Type: mixed
      # Total Memory (RAM): 64 GB
      # CPUs num: 16
      # Connections num: 500
      # Data Storage: ssd
      max_connections: "500"
      shared_buffers: 16GB
      effective_cache_size: 48GB
      maintenance_work_mem: 2GB
      checkpoint_completion_target: "0.9"
      wal_buffers: 16MB
      default_statistics_target: "100"
      random_page_cost: "1.1"
      effective_io_concurrency: "200"
      work_mem: 16256kB
      huge_pages: "try"
      min_wal_size: 1GB
      max_wal_size: 4GB
      max_worker_processes: "16"
      max_parallel_workers_per_gather: "4"
      max_parallel_workers: "16"
      max_parallel_maintenance_workers: "4"
```

ПРИЛОЖЕНИЕ 10

Файл конфигурации кластера jatoba-custom-certs.yaml

```
---
# Перед использованием этой конфигурации требуется заранее создать секреты
# содержащие все необходимые сертификаты
# Пример файла генерирующего сертификаты до установки CNPG и их использования - manifests/gen-cnpg-certs.yaml
cluster:
  certificates:
    serverTLSSecret: k8s-dev-db-server-cert
    serverCASecret: k8s-dev-db-server-cert
    clientCASecret: k8s-dev-db-client-cert
    replicationTLSSecret: k8s-dev-db-client-cert
```

Файл конфигурации кластера jatoba-custom-certs-poolers.yaml

```
---
# Перед использованием этой конфигурации требуется заранее создать секреты содержащие все необходимые сертификаты
# Пример файла генерирующего сертификаты до установки CNPG и их использования - manifests/gen-cnpg-certs.yaml
# Эта конфигурация относится только к пулерам, при необходимости подключите также файл jatoba-custom-certs.yaml
poolers:
  # Параметры для пулера: имя, тип пулера, кол-во экземпляров, образ, сертификаты
  # Read-Write пулер
  - name: rw
    type: rw
    poolMode: transaction
    instances: 1
    # Образ для пулера задается через шаблон спецификации манифеста, меняется только image если необходимо
    template:
      spec:
        containers:
          - image: jatoba/cloudnative-pg/japooler:6.10.1-redos7.3
            name: pgbouncer
    # Заменяем имя секрета содержащего сертификаты для пулера, по умолчанию имя генерируется чартом на основе шаблона "release-name-pooler"
    authQuerySecret:
      name: k8s-dev-db-client-pooler-cert
    # Необходимо указывать при смене секрета, оставляем как есть
    authQuery: SELECT username, passwd FROM pg_catalog.pg_shadow WHERE username=$1

  # Read-Only пулер
  - name: ro
    type: ro
    poolMode: transaction
    instances: 1
    # Образ для пулера задается через шаблон спецификации манифеста, меняется только image если необходимо
    template:
      spec:
        containers:
```

```
- image: jatoba/cloudnative-pg/japooler:6.10.1-redos7.3
  name: pgbouncer
# Заменяем имя секрета содержащего сертификаты для пулера, по умолчанию имя генер
ируется чартом на основе шаблона "release-name-pooler"
authQuerySecret:
  name: k8s-dev-db-client-pooler-cert
# Необходимо указывать при смене секрета, оставляем как есть
authQuery: SELECT username, passwd FROM pg_catalog.pg_shadow WHERE username=$1
```


ПРИЛОЖЕНИЕ 11

Манифест для создания реплики на основе pg_basebackup

```
manifests/clusters/replica/replica-delay-cluster.yaml
```

Манифест для создания реплики на основе pg_basebackup из уже существующего кластера CloudNativePG Jatoba cnpg-cluster-jatoba-cnpg-cluster:

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-replica
spec:
  instances: 1
```

Инициализация кластера из кластера источника cnpg-cluster-jatoba-cnpg-cluster:

```
bootstrap:
  pg_basebackup:
    database: app
    owner: app
    source: cnpg-cluster-jatoba-cnpg-cluster
```

Указание, что кластер будет работать как реплика кластера источника:

```
replica:
  enabled: true
  source: cnpg-cluster-jatoba-cnpg-cluster
```

Параметр, отвечающий за задержку между репликацией источника и текущего кластера:

```
minApplyDelay: '5m'
# Указание образов с Jatoba
imageCatalogRef:
  apiGroup: postgresql.cnpg.io
  kind: ImageCatalog
  major: 16
  name: cnpg-cluster-jatoba-cnpg-cluster
```

Параметры для подключения к кластеру источнику:

```
externalClusters:
- name: cnpg-cluster-jatoba-cnpg-cluster
  # Параметры подключения
  connectionParameters:
```

```
c    host: cnpg-cluster-jatoba-cnpg-cluster-rw.cnpg-cluster-abcdef.sv
    user: streaming_replica
    sslmode: verify-full
    dbname: postgres
```

Ключи с сертификатами для подключения к кластеру источнику с помощью сертификатов:

```
sslKey:
  name: cnpg-cluster-jatoba-cnpg-cluster-replication
  key: tls.key
sslCert:
  name: cnpg-cluster-jatoba-cnpg-cluster-replication
  key: tls.crt
sslRootCert:
  name: cnpg-cluster-jatoba-cnpg-cluster-ca
  key: ca.crt
```

Объем хранилища для текущего кластера и класс CSI (убедитесь что оно не меньше чем у источника):

```
storage:
  storageClass: longhorn
  size: 2Gi
```

ПРИЛОЖЕНИЕ 12

Файл настройки ресурсов Longhorn csi provider backuptarget.yaml

```
---
apiVersion: v1
kind: Secret
metadata:
  name: minio-secret-cert
  namespace: longhorn-system
data:
  AWS_ACCESS_KEY_ID: <s3_access_key_base64>
  AWS_CERT: <cert_base64>
  AWS_ENDPOINTS: <endpoint_url_base64>
  AWS_KEY: <private_key_base64>
  AWS_SECRET_ACCESS_KEY: <s3_secret_key_base64>
  type: Opaque

# Указание параметров для объекта backupTarget
# Путь к бакету (обязателен регион, по умолчанию MinIO использует us-east-1)
# Именованное секрета с учетными данными для подключения к S3
# Интервал опроса таргета (значение по умолчанию)
---
apiVersion: longhorn.io/v1beta2
kind: BackupTarget
metadata:
  name: default
  namespace: longhorn-system
spec:
  backupTargetURL: s3://cnpg@us-east-1/volumesnapshot
  credentialSecret: minio-secret-cert
  pollInterval: 5m0s
```

Файл конфигурации создания бэкапа кластера с использованием механизма VolumeSnapshot cluster-backup.yaml

```
---
# Кластер CNPG с режимом резервного копирования через VolumeSnapshot
# Укажите имя класса провайдера для CSI выполнив: export CSI_CLASSNAME=longhorn
# Замените "заглушку" на имя класса командой: sed -i "s/<csi-class-name>/${CSI_CLASSNAME}/" manifests/clusters/volumesnapshots/cluster-backup.yaml
---
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cnpg-cluster-vs
spec:
  # Количество экземпляров кластера
  instances: 3
  # Указание образов с Jatoba
  imageCatalogRef:
    apiGroup: postgresql.cnpg.io
    kind: ImageCatalog
```

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

```
major: 16
name: cnpg-cluster-jatoba-cnpg-cluster
# Начальная инициализация кластера с тестовой таблицей на 9999 строк
bootstrap:
initdb:
database: app
owner: ""
secret:
name: ""
options: []
encoding: UTF8
postInitApplicationSQL:
- create table test (
id text primary key,
column1 text NOT NULL,
column2 text NOT NULL,
column3 text NOT NULL
);
- insert into test (
id, column1, column2, column3
)
select
left(md5(i::text), 10),
md5(random()::text),
md5(random()::text),
left(md5(random()::text), 4)
from generate_series(1, 9999) s(i);
- grant all privileges on table test to app;
# Объем хранилища для текущего кластера и класс CSI
storage:
size: 2Gi
# Конфигурация резервного копирования
backup:
# Включение режима резервных копий через механизм VolumeSnapshot
volumeSnapshot:
className: <csi-class-name>

# Расписание резервного копирования
---
apiVersion: postgresql.cnpg.io/v1
kind: ScheduledBackup
metadata:
name: cnpg-cluster-vs-backup
спес:
# Имя кластера для резервного копирования
cluster:
name: cnpg-cluster-vs
# Указание метода снятия резервной копии - VolumeSnapshot
method: volumeSnapshot
# Расписание (крон формат, но первая цифра - секунды)
schedule: '0 0 0 * * *'
backupOwnerReference: cluster
immediate: true
```

Файл конфигурации восстановления кластера из бэкапа, созданного с использованием механизма VolumeSnapshot cluster-recovery.yaml

```
--- # Манифест для восстановления кластера из объекта VolumeSnapshot
# Укажите имя VolumeSnapshot объекта из которого будет восстановлен кластер
# Взять имя последнего объекта: export LASTVS=$(kubectl -n $NSCLUSTER get volumesnapshot -o name | cut -d"/" -f 2)
# Заменить "заглушку" на имя объекта: sed -i "s/<VOLUME SNAPSHOT OBJECT NAME>/${LASTVS}/" manifests/clusters/volumesnapshots/cluster-recovery.yaml
---
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-restore-vs
spec:
  # Количество экземпляров кластера (для демо указан 1 экземпляр)
  instances: 1
  # Указание образов с Jatoba
  imageCatalogRef:
    apiGroup: postgresql.cnpg.io
    kind: ImageCatalog
    major: 16
    name: cnpg-cluster-jatoba-cnpg-cluster
  # Объем хранилища для текущего кластера и класс CSI (убедитесь что оно не меньше чем у источника резервной копии)
  storage:
    size: 2Gi
  # Начальная инициализация в режиме recovery
  # Восстановление предполагает указание конкретного объекта VolumeSnapshot из которого будут восстановлены данные на PVC
  bootstrap:
    recovery:
      volumeSnapshots:
        storage:
          # Замените имя-шаблон на конкретный объект VolumeSnapshot
          # Просмотреть все объекты можно командой: kubectl -n $NSCLUSTER get vs
          name: <VOLUME SNAPSHOT OBJECT NAME>
          kind: VolumeSnapshot
          apiGroup: snapshot.storage.k8s.io
```

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

CloudNativePG (CNPG) — это оператор с открытым исходным кодом для управления базами данных PostgreSQL в среде Kubernetes.

Container Runtime Interface (CRI) — это ключевой интерфейс взаимодействия между компонентом kubelet в Kubernetes и средой выполнения контейнеров. Он позволяет Kubernetes управлять контейнерами, не создавая их самостоятельно.

Под (Pod) — это базовая единица развертывания в Kubernetes, представляющая собой группу из одного или нескольких контейнеров, которые разделяют сетевые и файловые ресурсы.

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

ОС	–	Операционная система
БД	–	База данных
ПО	–	Программное обеспечение
СУБД	–	Система управления базами данных

Лист регистрации изменений

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------