Fundament

Węzeł k3s bpm01 z dyskiem /k8s i PostgreSQL z operatora CloudNativePG.

- Węzeł k3s dla aplikacji: bpm01 z dyskiem /k8s, local-path z Retain i próbny nginx
- PostgreSQL w Kubernetesie: operator CloudNativePG, kopie, odtworzenie i aktualizacja

Węzeł k3s dla aplikacji: bpm01 z dyskiem /k8s, local-path z Retain i próbny nginx

Poziom: średnie

Dotyczy: k3s v1.36.5+k3s1 (Traefik 3.7, local-path-provisioner v0.0.37), Rocky Linux 9, bpm01 (10.20.10.130), restic na bkp01

Czas: ~90 min

Stawiamy jednowęzłowy k3s pod aplikacje, które mają przetrwać wymianę maszyny: wszystko, co należy do klastra (stan, obrazy, wolumeny i pliki wartości Helm), leży na osobnym dysku `/k8s`. Robimy własną klasę magazynu, która nie kasuje danych po usunięciu wolumenu, sprawdzamy całość kolorowym nginksem i ustawiamy kopię stanu na `bkp01`. Na tym węźle stanie Camunda 8, a wracamy tu przy każdej nowej aplikacji na k3s i przy przenosinach maszyny.

ZAŁOŻENIA SCENARIUSZA

`bpm01` to maszyna wirtualna na klastrze `hvclu01` (`hv01` i `hv02`) w VLAN 10, z jednym węzłem k3s, który jest serwerem i wykonuje pody. Docelowo klaster rośnie do trzech serwerów według strony k3s w wysokiej dostępności: migawki etcd, wymiana węzła, odtworzenie i aktualizacja, dlatego od początku włączamy wbudowany etcd. Laboratorium `k3s01` zostaje osobno i nic z niego nie przenosimy.

Skróty użyte na tej stronie (14)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
DNS	Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP
EPEL	Extra Packages for Enterprise Linux: dodatkowe repozytorium pakietów dla RHEL i klonów
etcd	etc distributed: rozproszony magazyn par klucz-wartość, w którym Kubernetes trzyma cały stan klastra, a zmiany uzgadnia protokołem Raft

Skrót	Co znaczy
PV	Physical Volume: dysk lub partycja oddana pod LVM; w Kubernetesie PersistentVolume, trwały wolumen
PVC	PersistentVolumeClaim: zamówienie trwałego dysku przez aplikację w Kubernetesie
README	Read Me: plik tekstowy z opisem projektu, wykresu lub szablonu, czytany jako pierwszy
REST	sposób budowania API przez zwykłe żądania HTTP (GET, POST) zamiast osobnego protokołu
SQLite	mała baza danych zapisana w jednym pliku; Home Assistant trzyma w niej historię w pliku home-assistant_v2.db
SUSE	nazwa niemieckiej firmy i jej dystrybucji Linuksa, od niemieckiego skrótu Software und System Entwicklung
VHDX	Virtual Hard Disk v2: format pliku dysku wirtualnego maszyny Hyper-V, następca starszego VHD, obsługuje większe rozmiary i jest odporniejszy na uszkodzenie przy nagłym zaniku zasilania
VLAN	Virtual LAN: wydzielona logicznie sieć na tym samym sprzęcie
WWW	World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS

Fiszka

Komendy wykonujemy z CLI bash jako użytkownik z uprawnieniami sudo na `bpm01`, a `kubectl` jako zwykły użytkownik z plikiem `~/.kube/config` z kroku 4. Bloki „Przykład” pokazują kształt wyniku, a nie zapis z maszyn Zielonej.

□□ ZMIENIA SYSTEM

Instalacja k3s w przypiętej wersji: INSTALL_K3S_VERSION

```
curl -sfL https://get.k3s.io | INSTALL_K3S_VERSION='WersjaK3s' sh -s - server
```

> (gdzie `WersjaK3s` to na przykład `v1.36.5+k3s1`; instalator czyta `/etc/rancher/k3s/config.yaml`, więc katalog danych i ścieżkę wolumenów ustawiamy w pliku przed instalacją)

Instaluje binarkę, usługę systemd `k3s` i uruchamia serwer z Traefikiem, CoreDNS i local-path-provisioner.

□□ **curl** = client URL: pobieranie i wysyłanie przez HTTP z terminala

TYLKO ODCZYT

Sprawdzenie, gdzie k3s trzyma dane: ls /k8s/k3s

```
sudo ls /k8s/k3s /k8s/k3s/server/db
```

Po instalacji z `data-dir: /k8s/k3s` stan klastra, token i obrazy kontenerów leżą na dysku danych, a nie w `/var/lib/rancher/k3s`.

Przykład:

```
/k8s/k3s:
agent  data  server

/k8s/k3s/server/db:
etcd  snapshots
```

ls = list: wypisz zawartość katalogu

TYLKO ODCZYT

Lista klas magazynu: kubectl get storageclass

```
kubectl get storageclass
```

Pokazuje wbudowaną klasę `local-path` (domyślną, z polityką `Delete`) i naszą `local-path-retain` z polityką `Retain`.

Przykład:

```
NAME                   PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE
AGE
local-path (default)   rancher.io/local-path   Delete          WaitForFirstConsumer
12m
local-path-retain      rancher.io/local-path   Retain          WaitForFirstConsumer   2m
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Lista wolumenów trwałych z polityką: kubectl get pv

```
kubectl get pv
```

Wolumen w stanie `Released` to dane, które zostały na dysku po usunięciu PVC. Przy polityce `Retain` nie znikają same i czekają na naszą decyzję.

Przykład:

```
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS     CLAIM                 STORAGECLASS
pvc-3f0c8a52-1b7e-4d0a-9c61-2a4e7f1d9b30   1Gi        RWO            Retain
Released   kolory/test-retain    local-path-retain
```

PVC = PersistentVolumeClaim: zamówienie trwałego dysku przez aplikację w Kubernetesie · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

ZMIENIA SYSTEM

Zapis migawki etcd na żądanie: k3s etcd-snapshot save

```
sudo k3s etcd-snapshot save --data-dir /k8s/k3s --name NazwaMigawki
```

Zapisuje migawkę stanu klastra w `/k8s/k3s/server/db/snapshots`. Opcję `--data-dir` podajemy jawnie, bo katalog danych nie jest domyślny.

etcd = etc distributed: rozproszony magazyn par klucz-wartość, w którym Kubernetes trzyma cały stan klastra, a zmiany uzgadnia protokołem Raft · **k3s** = lekka dystrybucja Kubernetesa (5 liter mniej niż k8s)

TYLKO ODCZYT

Lista migawek: k3s etcd-snapshot ls

```
sudo k3s etcd-snapshot ls --data-dir /k8s/k3s
```

Pokazuje migawki zaplanowane (co sześć godzin) i zrobione ręcznie, z rozmiarem i datą.

k3s = lekka dystrybucja Kubernetesa (5 liter mniej niż k8s)

ZMIENIA SYSTEM

Uruchomienie kopii stanu na bkp01: systemctl start kopia-stanu-k3s

```
sudo systemctl start kopia-stanu-k3s.service
```

Wysyła migawki, token, plik konfiguracji k3s i katalog `/k8s/helm` do repozytorium restic `orkiestracja` na `bkp01`. Zwykle robi to zegar z kroku 8, ręcznie uruchamiamy kopię przed zmianą.

systemctl = system control: sterowanie usługami systemd

OSTROŻNIE · NIEODWRACALNE

Usunięcie wolumenu w stanie Released: kubectl delete pv

```
kubectl delete pv NazwaWolumenu
```

Usuwa sam obiekt PV, a katalog z danymi na `/k8s/apps` zostaje, bo polityka `Retain` nie kasuje plików. Katalog usuwamy osobno, dopiero gdy wiemy, że dane nie są potrzebne.

PV = Physical Volume: dysk lub partycja oddana pod LVM; w Kubernetesie PersistentVolume, trwały wolumen · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

Z życia wzięte

Z ŻYCIA WZIĘTE

Środa, 10:30. Tomek usuwa „stary” wolumen próbny na laboratorium `k3s01` i dopiero po chwili orientuje się, że PVC o tej samej nazwie trzymało też bazę testowej Camundy, którą Ola stawiała w poniedziałek. Klasa `local-path` ma politykę `Delete`, więc katalog zniknął z dysku razem z obiektem. Na `bpm01` robimy inaczej: osobny dysk, klasa z polityką `Retain` i katalogi nazwane od przestrzeni nazw i PVC, żeby w `ls /k8s/apps` od razu było widać, czyje to dane.

Jak to działa

Rozwiń: jak to działa pod spodem

Dlaczego osobny dysk /k8s

Na `docker01` wszystko, co należy do aplikacji, leży na wolumenie `/docker`, oddzielnym od systemu. Na `bpm01` robimy to samo dla Kubernetesa: drugi dysk maszyny wirtualnej staje się grupą `vgK8s` z wolumenem `lvK8s`, zamontowanym jako `/k8s`. Pełny dysk aplikacji nie zatrzymuje wtedy systemu, a sam dysk (plik VHDX na `hvclu01`) da się odpiąć i podpiąć do nowej maszyny.



Rysunek 1. Układ dysków bpm01: na pierwszym dysku system, binarka k3s, config.yaml i /var/lib/kubelet; na drugim /k8s z katalogami k3s (stan i obrazy), apps (wolumeny PVC), helm (pliki wartości i manifesty) i kopie, oraz odpowiedniki katalogów z docker01 (kliknięcie otwiera pełny rozmiar)

Pojęcie: katalog danych k3s (`data-dir`). Miejsce, w którym k3s trzyma wszystko, co tworzy sam: bazę stanu (u nas etcd), certyfikaty, token klastra i obrazy kontenerów. Domyślnie to `/var/lib/rancher/k3s`, a opcja `--data-dir` przenosi go gdzie indziej.

Pojęcie: local-path-provisioner. Dodatek wbudowany w k3s, który na żądanie PVC zakłada zwykły katalog na dysku węzła i podaje go podowi jako wolumen. Nie powiela danych na inne węzły, więc wolumen jest przypięty do jednej maszyny. Katalog bazowy ustawia opcja `--default-local-storage-path`.

> **Pojęcie: polityka odzyskiwania (reclaim policy).** Co ma się stać z danymi wolumenu, gdy ktoś usunie PVC. `Delete` kasuje katalog razem z obiektem, `Retain` zostawia dane, a wolumen przechodzi w stan `Released` i czeka na administratora.
>
> Szczegóły PV, PVC i klas są na stronie Wolumeny, PV, PVC i StorageClass.

Katalogi jak przy Dockerze

docker01	bpm01	Co tam leży
`/docker/apps/<aplikacja>`	`/k8s/apps/<przestrzeń>/<pvc>`	dane aplikacji: bazy, dzienniki Zeebe, pliki
`/docker/compose/<aplikacja>`	`/k8s/helm/<aplikacja>`	`values.yaml`, manifesty, notatki o wersji
`/docker/lib`	`/k8s/k3s`	dane samego silnika: obrazy, stan
`/docker/kopie`	`/k8s/kopie`	zrzuty przed wysłaniem na `bkp01`

Nazwy katalogów w `/k8s/apps` ustawia parametr `pathPattern` naszej klasy magazynu. Bez niego local-path nazywa katalog od identyfikatora wolumenu (`pvc-3f0c..._kolory_test-retain`), z nim powstaje `/k8s/apps/kolory/test-retain/`. Ta wygoda ma cenę: nowe PVC o tej samej nazwie dostaje stary katalog z danymi, co przy przenosinach jest zaletą, a przy próbach może zaskoczyć.

Dlaczego od razu etcd, a nie SQLite

Jednowęzłowy k3s domyślnie trzyma stan w SQLite. Opcja `cluster-init: true` zakłada od początku wbudowany etcd, więc drugi i trzeci serwer dołączymy później bez przebudowy. Dostajemy też migawki etcd z harmonogramem, które kopiujemy na `bkp01` tak samo jak na `k3s11` do `k3s13`.

Dlaczego k3s 1.36, a nie 1.37

Najnowszy k3s to gałąź 1.37, ale operator CloudNativePG 1.30, na którym postawimy PostgreSQL, wspiera według swoich informacji o wydaniu Kubernetesa 1.34, 1.35 i 1.36. Wybieramy więc najnowsze wydanie gałęzi 1.36, czyli `v1.36.5+k3s1` z 30 września 2026 roku. Na 1.37 przejdziemy, gdy CloudNativePG ją obejmie.

Rancher: kiedy warto

Rancher to aplikacja SUSE do zarządzania wieloma klastrami Kubernetesa z przeglądarki: zakładanie klastrów, użytkownicy, katalog aplikacji Helm, podgląd podów. Najnowsza seria na stronie macierzy wsparcia SUSE to 2.15 (wydanie 2.15.2). Ma sens, gdy klastrów jest kilka, a z panelu korzysta więcej osób niż dwóch administratorów.

Na `bpm01` go nie stawiamy. Instalujemy aplikacje poleceniem `helm` z plikami wartości w `/k8s/helm`, które trzymamy też w Gitea, bo to daje powtarzalny zapis, co i z jakimi ustawieniami zainstalowaliśmy. Instalacja tej samej aplikacji klikaniem w katalogu Ranchera chowa wartości w obiekcie klastra i trudniej je potem porównać albo odtworzyć.

Przebieg

Zanim zaczniemy

Co	Wartość	Uwagi
Maszyna	`bpm01`, Rocky Linux 9 według strony o instalacji z LVM, 8 vCPU, 24 GiB pamięci	Camunda z PostgreSQL potrzebuje około 8 GiB na start, reszta to zapas na Hub, Identity i Keycloak
Dysk systemowy	40 GB, `vgOS/lvOS` i `vgVAR/lvVAR`	jak na każdym linuksie Zielonej
Dysk danych	200 GB, osobny plik VHDX, `/dev/sdb`	rozmiar do decyzji autora
Adres	10.20.10.130	sprawdziliśmy grep w całej bazie: nieużywany
DNS	`bpm01.ad.zielona.example` i `bpm.zielona.example` na 10.20.10.130	rekordy z `dc01`, niżej

System instalujemy według strony Instalacja Rocky Linux z własnym nazewnictwem VG i LV. Rekordy DNS zakładamy w PowerShellu na `dc01`:

```
Add-DnsServerResourceRecordA -ZoneName "ad.zielona.example" -Name "bpm01" -IPv4Address
"10.20.10.130" -CreatePtr
Add-DnsServerResourceRecordA -ZoneName "zielona.example" -Name "bpm" -IPv4Address
"10.20.10.130"
```

Pierwszy rekord to nazwa maszyny, drugi to adres aplikacji w wewnętrznej strefie `zielona.example`, jak przy innych usługach wewnętrznych (DNS w praktyce: strefy w AD, przekazywanie warunkowe,

scavenging i split-brain).

1. Dysk /k8s na LVM

Komendy można wykonywać całościowo jako blok tekstu. Przed pierwszą sprawdzamy `lsblk`, czy `/dev/sdb` to na pewno pusty dysk danych.

```
# Dysk danych bpm01 jako vgK8s/lvK8s, montowany w /k8s
sudo pvcreate /dev/sdb
sudo vgcreate vgK8s /dev/sdb
sudo lvcreate -l 100%FREE -n lvK8s vgK8s
sudo mkfs.xfs /dev/mapper/vgK8s-lvK8s
sudo mkdir -p /k8s
echo '/dev/mapper/vgK8s-lvK8s /k8s xfs defaults 0 0' | sudo tee -a /etc/fstab
sudo systemctl daemon-reload
sudo mount /k8s
sudo mkdir -p /k8s/k3s /k8s/apps /k8s/helm /k8s/kopie
df -h /k8s
```

> (gdzie `/dev/sdb` to dysk danych, nie systemowy: bardzo istotne, bo `pvcreate` na złym dysku niszczy system!)

2. Zapora

Na jednym węźle wystarczy API, ruch WWW do Traefika i zaufanie dla sieci podów i usług. Porty etcd i flannel dopiszemy przy dołączaniu drugiego serwera, jak na stronie o k3s w wysokiej dostępności.

```
sudo firewall-cmd --permanent --add-rich-rule='rule family="ipv4" source
address="10.20.10.0/24" port port="6443" protocol="tcp" accept'
sudo firewall-cmd --permanent --add-service=http --add-service=https
sudo firewall-cmd --permanent --zone=trusted --add-source=10.42.0.0/16
sudo firewall-cmd --permanent --zone=trusted --add-source=10.43.0.0/16
sudo firewall-cmd --reload
```

Zakresy 10.42.0.0/16 (pody) i 10.43.0.0/16 (usługi) to domyślne sieci k3s. Bez nich zapora może odrzucać ruch między podami, a objaw wygląda potem jak błąd aplikacji (strona o częstych problemach).

3. Plik konfiguracji k3s

Plik `/etc/rancher/k3s/config.yaml` zakładamy przed instalacją:

```
data-dir: /k8s/k3s
default-local-storage-path: /k8s/apps
cluster-init: true
node-ip: 10.20.10.130
tls-san:
  - 10.20.10.130
  - bpm01.ad.zielona.example
write-kubeconfig-mode: "0600"
etcd-snapshot-schedule-cron: "0 */6 * * *"
etcd-snapshot-retention: 28
```

Klucze odpowiadają opcjom `--data-dir`, `--default-local-storage-path`, `--cluster-init`, `--node-ip`, `--tls-san` i `--write-kubeconfig-mode` z dokumentacji k3s. Migawki co sześć godzin z 28 zachowanymi to siedem dni na węźle, tak jak na `k3s11` do `k3s13`. Kopię pliku trzymamy też w `/k8s/helm/k3s/config.yaml`, bo sam plik leży na dysku systemowym.

```
sudo install -d -m 755 /etc/rancher/k3s /k8s/helm/k3s
sudo vim /etc/rancher/k3s/config.yaml
sudo cp /etc/rancher/k3s/config.yaml /k8s/helm/k3s/config.yaml
```

4. Instalacja k3s i dostęp kubectl

```
curl -sfL https://get.k3s.io | INSTALL_K3S_VERSION='v1.36.5+k3s1' sh -s - server
sudo k3s kubectl get nodes -o wide
mkdir -p ~/.kube
sudo cat /etc/rancher/k3s/k3s.yaml > ~/.kube/config
chmod 600 ~/.kube/config
kubectl get pods -A
```

Plik `k3s.yaml` zostaje w `/etc/rancher/k3s` mimo innego katalogu danych. Po kilkudziesięciu sekundach w `kube-system` działają `coredns`, `local-path-provisioner`, `metrics-server` i `traefik`, a karta „Sprawdzenie, gdzie k3s trzyma dane” z Fiszki pokazuje katalogi na `/k8s`.

5. Klasa magazynu local-path-retain

Wbudowanej klasy `local-path` nie zmieniamy, bo k3s nadpisuje swoje manifesty przy starcie. Zakładamy obok własną klasę, zapisaną jako `/k8s/helm/k3s/storageclass-retain.yaml`:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-path-retain
provisioner: rancher.io/local-path
parameters:
  pathPattern: "{{ .PVC.Namespace }}/{{ .PVC.Name }}/"
reclaimPolicy: Retain
volumeBindingMode: WaitForFirstConsumer
```

```
kubectl apply -f /k8s/helm/k3s/storageclass-retain.yaml
kubectl get storageclass
```

Klasy nie robimy domyślną: każda aplikacja na `bpm01` podaje ją jawnie (w Camundzie `pvcStorageClassName`, w CloudNativePG `storageClass`). Dwie klasy domyślne naraz to proszenie się o PVC na złej klasie. Parametr `pathPattern` opisuje README local-path-provisioner; po pierwszym PVC sprawdzamy, czy wersja wbudowana w k3s go respektuje (krok 6).

6. Próbny nginx i test wolumenu

Kolorowy nginx ze strony k3s w laboratorium: pierwszy deploy, nginx w kolorach sprawdza Traefika, Service i DNS w klastrze. Zapisujemy manifest z tamtej strony jako `/k8s/helm/kolory/kolorowy-nginx.yaml`, zmieniamy host na `kolory-bpm.zielona.example` i stosujemy:

```
sed -i 's/kolory.zielona.example/kolory-bpm.zielona.example/' /k8s/helm/kolory/kolorowy-
nginx.yaml
kubectl apply -f /k8s/helm/kolory/kolorowy-nginx.yaml
for i in $(seq 1 6); do curl -s --resolve kolory-bpm.zielona.example:80:10.20.10.130
http://kolory-bpm.zielona.example/ | grep "Kolor:"; done
```

Mieszane kolory w wyniku znaczą, że Traefik na `bpm01` kieruje ruch do trzech podów. Teraz test wolumenu, plik `/k8s/helm/kolory/test-retain.yaml`:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: test-retain
```

```
  namespace: kolory
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: local-path-retain
  resources:
    requests:
      storage: 1Gi
---
apiVersion: v1
kind: Pod
metadata:
  name: test-retain
  namespace: kolory
spec:
  containers:
    - name: zapis
      image: busybox:1.37.0
      command: ["sh", "-c", "date >> /dane/test.txt; sleep 3600"]
      volumeMounts:
        - name: dane
          mountPath: /dane
  volumes:
    - name: dane
      persistentVolumeClaim:
        claimName: test-retain
```

```
kubectl apply -f /k8s/helm/kolory/test-retain.yaml
kubectl -n kolory get pvc test-retain
sudo ls -R /k8s/apps/kolory
kubectl -n kolory delete pod test-retain
kubectl -n kolory delete pvc test-retain
kubectl get pv
sudo cat /k8s/apps/kolory/test-retain/test.txt
```

Oczekujemy PVC `Bound`, katalogu `/k8s/apps/kolory/test-retain/` z plikiem `test.txt`, a po usunięciu PVC wolumenu `Released` i pliku nadal na dysku. Jeśli katalog nazywa się `pvc-..._kolory_test-retain`, wbudowana wersja local-path nie obsługuje `pathPattern` i zostajemy przy domyślnych nazwach. Na koniec sprzątamy: karta „Usunięcie wolumenu w stanie Released”, `sudo rm -rf /k8s/apps/kolory/test-retain` i `kubectl delete namespace kolory`.

7. Repozytorium restic dla stanu klastra

Stan `bpm01` trafia do istniejącego repozytorium `orkiestracja` na `bkp01` (poziom 90 dni, decyzja autora z 18 października 2026), tym samym kontem REST co laboratoria. Instalujemy `restic` z EPEL i zakładamy dwa pliki z hasłami według strony k3s w wysokiej dostępności: migawki etcd, wymiana węzła, odtworzenie i aktualizacja: `/etc/restic/haslo-orkiestracja` i `/etc/restic/rest-haslo-orkiestracja`, oba z prawami `0600`.

Skrypt `/usr/local/sbin/kopia-stanu-k3s.sh` (prawa `0750`) różni się od tamtego ścieżkami i tagiem:

```
#!/usr/bin/env bash
# /usr/local/sbin/kopia-stanu-k3s.sh na bpm01
# Wysyla migawki etcd, token, config.yaml i pliki wartosci Helm do repozytorium orkiestracja.
set -euo pipefail
export RESTIC_REPOSITORY=rest:https://bkp01.ad.zielona.example:8000/orkiestracja
export RESTIC_REST_USERNAME=orkiestracja
RESTIC_REST_PASSWORD="$(cat /etc/restic/rest-haslo-orkiestracja)"
export RESTIC_REST_PASSWORD
export RESTIC_PASSWORD_FILE=/etc/restic/haslo-orkiestracja
restic backup --tag k3s --tag bpm01 \
  /k8s/k3s/server/db/snapshots \
  /k8s/k3s/server/token \
  /etc/rancher/k3s/config.yaml \
  /k8s/helm
```

Jednostkę `kopia-stanu-k3s.service` i zegar `kopia-stanu-k3s.timer` (20 minut po każdej migawce) przepisujemy bez zmian z tamtej strony i włączamy:

```
sudo systemctl daemon-reload
sudo systemctl enable --now kopia-stanu-k3s.timer
sudo systemctl start kopia-stanu-k3s.service
sudo journalctl -u kopia-stanu-k3s.service -n 20
```

Ta kopia obejmuje stan klastra i konfigurację, a nie dane aplikacji. Bazy PostgreSQL kopiuje operator (strona o CloudNativePG), a dziennik Zeebe ma własną procedurę kopii opisaną na stronie o instalacji Camundy. Katalogu `/k8s/apps` nie kopiujemy restikiem na żywo, bo pliki działających baz skopiowane w trakcie zapisu są niespójne.

8. Przeniesienie dysku /k8s na nową maszynę

To jest powód, dla którego dysk danych jest osobny. Przebieg poniżej opieramy na tym, że k3s czyta cały stan z `data-dir`, a wolumeny local-path są przypięte do nazwy węzła; całość przećwiczymy w laboratorium przed pierwszą prawdziwą potrzebą.

1. Na starej maszynie: migawka (`k3s etcd-snapshot save`), kopia z kroku 7, potem `sudo systemctl stop k3s` i `sudo /usr/local/bin/k3s-killall.sh`.
2. `sudo umount /k8s`, wyłączenie maszyny i odpięcie pliku VHDX w Hyper-V.
3. Nowa maszyna z tą samą nazwą `bpm01` i adresem 10.20.10.130, Rocky Linux 9, zapora z kroku 2.
4. Podpięcie VHDX, `sudo vgchange -ay vgK8s`, wpis w `/etc/fstab` i `mount /k8s`.
5. `config.yaml` z `/k8s/helm/k3s/config.yaml` do `/etc/rancher/k3s/` i instalacja tej samej wersji k3s co przed przenosinami.
6. Kontrola: `kubectl get nodes`, `kubectl get pods -A`, `kubectl get pv` (wszystkie `Bound`).

Inna nazwa węzła to inna sytuacja: stary węzeł zostaje w klastrze jako `NotReady`, a wolumeny local-path wskazują jego nazwę. README local-path opisuje parametr `nodeAffinityKey` na taki przypadek, ale w tej książce trzymamy się zasady „ta sama nazwa”.

9. Droga do trzech serwerów

Gdy pilotaż przejdzie w produkcję, dokładamy `bpm02` i `bpm03` (adresy do przydzielenia) według strony o k3s w wysokiej dostępności: porty etcd i flannel w zaporze, adres wirtualny API na keepalived, dopisanie go do `tls-san` na `bpm01` z restartem usługi, potem dołączenie nowych serwerów przez adres wirtualny i token. Local-path nie przenosi danych między węzłami, więc wtedy aplikacje z danymi przechodzą na magazyn replikowany (Longhorn) albo na powielanie w samej aplikacji (CloudNativePG z kilkoma instancjami, Zeebe z trzema brokerami).

Weryfikacja

```
df -h /k8s
kubectl get nodes -o wide
kubectl get storageclass
sudo k3s etcd-snapshot ls --data-dir /k8s/k3s
systemctl list-timers kopia-stanu-k3s.timer
```

Oczekujemy `/k8s` zamontowanego z `vgK8s-lvK8s`, węzła `bpm01` w stanie `Ready` z wersją `v1.36.5+k3s1` i adresem 10.20.10.130, klasy `local-path-retain` z polityką `Retain`, migawek co sześć godzin i zegara kopii z najbliższym uruchomieniem. Na `bkp01` polecenie `restic snapshots --tag bpm01` pokazuje kopie stanu.

Wycofanie zmian

Zegar kopii wyłączamy (`sudo systemctl disable --now kopia-stanu-k3s.timer`), usuwamy jednostki i skrypt. k3s odinstalowujemy skryptem `/usr/local/bin/k3s-uninstall.sh`, ale wcześniej czytamy go, żeby wiedzieć, czy przy innym `data-dir` usuwa katalog `/k8s/k3s` (do sprawdzenia w naszej wersji). Dysk `/k8s` zostawiamy albo usuwamy świadomie: wpis w `/etc/fstab`, `umount`, `lvremove`, `vgremove`, `pvremove`. Rekordy `bpm` i `bpm01` w DNS usuwamy na końcu.

Gdy coś nie zadziała

Objaw	Przyczyna	Co robimy
Po instalacji stan leży w `/var/lib/rancher/k3s`	`config.yaml` założony po instalacji albo z literówką w kluczu	odinstalowanie, poprawny plik, instalacja od nowa (klaster jest pusty)
`k3s etcd-snapshot ls` nie pokazuje migawek	polecenie szuka w domyślnym katalogu	dopisujemy `--data-dir /k8s/k3s`
PVC wisi w `Pending`	klasa `WaitForFirstConsumer` czeka na poda albo zła nazwa klasy	to normalne do startu poda; potem `kubectl describe pvc`
Katalog wolumenu nazywa się `pvc-..._przestrzeń_nazwa`	wbudowana wersja local-path nie obsługuje `pathPattern`	zostajemy przy domyślnych nazwach albo podnosimy k3s
Pody nie widzą się nawzajem, `curl` do ClusterIP z węzła działa	zapora odrzuca ruch z sieci podów	strefa `trusted` dla 10.42.0.0/16 i 10.43.0.0/16 (krok 2)
Po przeniesieniu dysku pody z wolumenami nie startują	inna nazwa węzła niż w przypięciu wolumenów	ta sama nazwa maszyny albo ręczna zmiana PV
Kopia kończy się błędem 401	złe hasło konta REST `orkiestracja`	plik `rest-haslo-orkiestracja` z tym samym hasłem co na `bkp01`

□□ Ćwiczenia

1. Dlaczego nie zmieniamy po prostu polityki wbudowanej klasy `local-path` na `Retain`?

Odpowiedź

Bo k3s przy każdym starcie zastępuje swoje wbudowane manifesty wersją domyślną, więc nasza zmiana by zniknęła. Własna klasa obok jest trwała i jawna: w każdym pliku wartości widać, której klasy używa aplikacja.

1. Pan Zbigniew pyta, czy skoro dane leżą na osobnym dysku, kopia na `bkp01` jest jeszcze potrzebna. Co odpowiadamy?

Odpowiedź

Tak. Osobny dysk chroni przed awarią systemu i ułatwia przenosiny, ale nie przed skasowaniem przestrzeni nazw, uszkodzeniem pliku VHDX, awarią magazynu klastra Hyper-V ani atakiem szyfrującym. Kopia na `bkp01` i krzyżowa na `bkp02` to osobne kopie w innym miejscu.

1. Dlaczego na `bpm01` od razu włączamy `cluster-init: true`, skoro węzeł jest jeden?

Odpowiedź

Żeby stan od początku był w etcd: dostajemy migawki z harmonogramem i możemy później dołączyć drugi i trzeci serwer bez przebudowy. Przy SQLite migawki etcd nie działają, a przejście na etcd jest osobną operacją.

1. Co się stanie, gdy po teście usuniemy PV, ale nie usuniemy katalogu, a potem znowu utworzymy PVC `test-retain` w przestrzeni `kolory`?

Odpowiedź

Przy `pathPattern` bez nazwy wolumenu nowy PV dostanie ten sam katalog `/k8s/apps/kolory/test-retain/`, więc pod zobaczy stary plik `test.txt`. To wygodne przy przenosinach, ale przy testach może zaskoczyć, dlatego po próbach usuwamy katalog.

Powiązane: k3s w laboratorium: pierwszy deploy, nginx w kolorach · Wolumeny, PV, PVC i StorageClass · k3s w wysokiej dostępności: migawki etcd, wymiana węzła, odtworzenie i aktualizacja · Jak trzymamy aplikacje w Dockerze: konwencje Zielonej · Strategia kopii zapasowych Zielonej: RPO, RTO, reguła 3-2-1-1-0 i dwie lokalizacje · Serwery kopii bkp01 i bkp02: restic, kopia krzyżowa i niezmienność · PostgreSQL w Kubernetesie: operator CloudNativePG, kopie, odtworzenie i aktualizacja

Źródła (odczyt 1 listopada 2026): docs.k3s.io, k3s server (`--data-dir`, `--default-local-storage-path`, `--cluster-init`, `--tls-san`, `--write-kubeconfig-mode`, `--node-ip`, `--disable`); docs.k3s.io,

v1.36.X release notes (v1.36.5+k3s1 z 30 września 2026, Traefik v3.7.13, local-path-provisioner v0.0.37, etcd v3.6.14); docs.k3s.io, Networking Services (Traefik v3, ServiceLB, HelmChartConfig, nadpisywanie manifestów); github.com/rancher/local-path-provisioner, README (`pathPattern`, `nodePath`, `nodeAffinityKey`); cloudnative-pg.io, Release notes v1.30 (Kubernetes 1.34 do 1.36); suse.com, macierz wsparcia Rancher (seria 2.15, wydanie 2.15.2).

PostgreSQL w Kubernetesie: operator CloudNativePG, kopie, odtworzenie i aktualizacja

Poziom: średnie

Dotyczy: CloudNativePG 1.30.1, wtyczka Barman Cloud 0.15.1, cert-manager v1.21.3, PostgreSQL 17.11, k3s v1.36 na bpm01

Czas: ~75 min

Operator CloudNativePG zakłada i pilnuje baz PostgreSQL w Kubernetesie: tworzy pody, użytkowników, bazy, kopie do magazynu obiektowego i odtwarza klaster na wskazaną chwilę. Ta strona pokazuje go na `bpm01` jako jeden klaster `pg-bpm` z trzema bazami dla Camundy, ale tak samo posłuży każdej innej aplikacji na k3s. Wracamy tu przy nowej bazie w klastrze, przed aktualizacją PostgreSQL i w dniu, w którym trzeba coś odtworzyć.

ZAŁOŻENIA SCENARIUSZA

kopie fizyczne idą do magazynu zgodnego z S3 na `bkp01`. Przyjmujemy jako propozycję Garage (otwarty magazyn obiektowy, w Arch Linux wersja 2.5.0 z 27 października 2026) z kubełkiem `cnpg-bpm` pod adresem `https://bkp01.ad.zielona.example:3900`; MinIO odpada, bo jego repozytorium jest zarchiwizowane od 25 kwietnia 2026, jak opisuje strona Perforce i Lore: porównanie dróg do wysokiej dostępności na Windowsie. Instalację Garage na `bkp01` autor zatwierdzi osobno, a do tego czasu ta sama konfiguracja działa z dowolnym magazynem S3, także z Amazon S3.

Skróty użyte na tej stronie (10)

Skrót	Co znaczy
CNCF	Cloud Native Computing Foundation: fundacja pod Linux Foundation, formalny właściciel projektu Kubernetes od lipca 2015 roku.

Skrót	Co znaczy
CRD	Custom Resource Definition: własny typ obiektu dodany do Kubernetesa
GLPI	Gestionnaire Libre de Parc Informatique: otwarty system do zgłoszeń, inwentarza i bazy wiedzy IT; w Zielonej działa w kontenerze na docker01
MinIO	program do własnego magazynu obiektowego zgodnego z S3; wersja społecznościowa jest od 2026 roku zarchiwizowana i dostępna tylko jako kod źródłowy
PEM	Privacy-Enhanced Mail: tekstowy format certyfikatu zaczynający się od -----BEGIN
PVC	PersistentVolumeClaim: zamówienie trwałego dysku przez aplikację w Kubernetesie
README	Read Me: plik tekstowy z opisem projektu, wykresu lub szablonu, czytany jako pierwszy
REST	sposób budowania API przez zwykłe żądania HTTP (GET, POST) zamiast osobnego protokołu
S3	Simple Storage Service: magazyn plików w chmurze AWS
WAL	Write-Ahead Log: dziennik PostgreSQL, zapis zmian przed zapisem danych

Fiszka

Komendy `kubectl` wykonujemy jako zwykły użytkownik z plikiem `~/.kube/config` na `bpm01` albo na stacji administratora. Wtyczkę `kubectl cnpg` instalujemy w kroku 2. W przykładach klaster nazywa się `pg-bpm` i stoi w przestrzeni nazw `camunda`.

□□ ZMIENIA SYSTEM

Instalacja operatora CloudNativePG: kubectl apply --server-side

```
kubectl apply --server-side -f https://raw.githubusercontent.com/cloudnative-pg/cloudnative-pg/release-1.30/releases/cnpg-WersjaCnpg.yaml
```

> (gdzie `WersjaCnpg` to na przykład `1.30.1`; adres manifestu ma w nazwie gałąź `release-1.30`)

Zakłada przestrzeń nazw `cnpg-system`, definicje zasobów (CRD) i wdrożenie `cnpg-controller-manager`. Opcja `--server-side` jest w poleceniu z dokumentacji, bo definicje są zbyt duże na zwykłe `apply`.

□□ **CRD** = Custom Resource Definition: własny typ obiektu dodany do Kubernetesa · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Sprawdzenie, czy operator działa: kubectl rollout status

```
kubectl rollout status deployment -n cnpg-system cnpg-controller-manager
```

Przykład:

```
deployment "cnpg-controller-manager" successfully rolled out
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Stan klastra PostgreSQL: kubectl cnpg status

```
kubectl cnpg status NazwaKlastra -n NazwaPrzestrzeni
```

Pokazuje instancje, która jest główna, stan archiwizacji WAL, ostatnią kopię i certyfikaty. To pierwsza komenda przy każdym pytaniu „czy baza żyje”.

Przykład (skrócony):

```
Cluster Summary
Name                 camunda/pg-bpm
PostgreSQL Image:    ghcr.io/cloudnative-pg/postgresql:17.11-standard-trixie
Primary instance:    pg-bpm-1
Status:              Cluster in healthy state
Instances:           1
Ready instances:     1
```

WAL = Write-Ahead Log: dziennik PostgreSQL, zapis zmian przed zapisem danych · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Połączenie psql z bazą w klastrze: kubectl cnpg psql

```
kubectl cnpg psql NazwaKlastra -n NazwaPrzestrzeni -- -d NazwaBazy
```

Otwiera `psql` w podzie głównej instancji jako superużytkownik `postgres`. Opcje po `--` idą wprost do `psql`.

kubectl = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Lista kopii klastra: kubectl get backups

```
kubectl get backups -n NazwaPrzestrzeni
```

Przykład:

```
NAME                            AGE   CLUSTER   METHOD   PHASE       ERROR
pg-bpm-codziennie-20261102013000  7h    pg-bpm    plugin   completed
pg-bpm-przed-zmiana             5m    pg-bpm    plugin   completed
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

ZMIENIA SYSTEM

Zrzut logiczny bazy na dysk węzła: pg_dump przez kubectl exec

```
kubectl exec -n NazwaPrzestrzeni NazwaPoda -c postgres -- pg_dump -Fc -d NazwaBazy > /k8s/kopie/NazwaBazy.dump
```

Zrzut w formacie własnym `pg_dump` (opis na stronie pg_dump, pg_dumpall, pg_restore) trafia do `/k8s/kopie`, skąd zabiera go restic. To uzupełnienie kopii fizycznej, a nie jej zastępstwo.

kubectl = Kubernetes control: główne narzędzie Kubernetesa

ZMIENIA SYSTEM

Restart instancji klastra: kubectl cnpg restart

```
kubectl cnpg restart NazwaKlastra -n NazwaPrzestrzeni
```

Restartuje instancje po kolei. Przy jednej instancji oznacza to krótką przerwę dla aplikacji.

kubectl = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Raport do zgłoszenia: kubectl cnpg report cluster

```
kubectl cnpg report cluster NazwaKlastra -n NazwaPrzestrzeni -f raport-pg.zip --logs
```

Zbiera definicje, zdarzenia i dzienniki w jedno archiwum, które dołączamy do zgłoszenia w GLPI albo na forum projektu.

GLPI = Gestionnaire Libre de Parc Informatique: otwarty system do zgłoszeń, inwentarza i bazy wiedzy IT; w Zielonej działa w kontenerze na docker01 · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

Z życia wzięte

Z ŻYCIA WZIĘTE

Poniedziałek, 7:50. Ola pisze, że w piątek po południu ktoś z jej zespołu uruchomił skrypt testowy na bazie `hub` w laboratorium i skasował połowę projektów w Web Modelerze. Zamiast szukać zrzutu sprzed tygodnia, zakładamy obok nowy klaster `pg-bpm-odtw` z kopii w kubełku na `bkp01`, z czasem docelowym „piątek, 15:30”. Po dwudziestu minutach porównujemy listę projektów, przełączamy Hub na odtworzoną bazę i Tomek zapisuje w GLPI, że skrypty testowe dostają odtąd osobny klaster.

Jak to działa

Rozwiń: jak to działa pod spodem

Operator zamiast ręcznego PostgreSQL

> **Pojęcie: operator Kubernetesa.** Program działający w klastrze, który zna się na jednej aplikacji i wykonuje za administratora jej rutynę. Administrator zapisuje stan pożądany w nowym rodzaju obiektu (tu `Cluster`), a operator tworzy pody, wolumeny, usługi i hasła, a potem pilnuje ich stale, tak jak Deployment pilnuje podów.

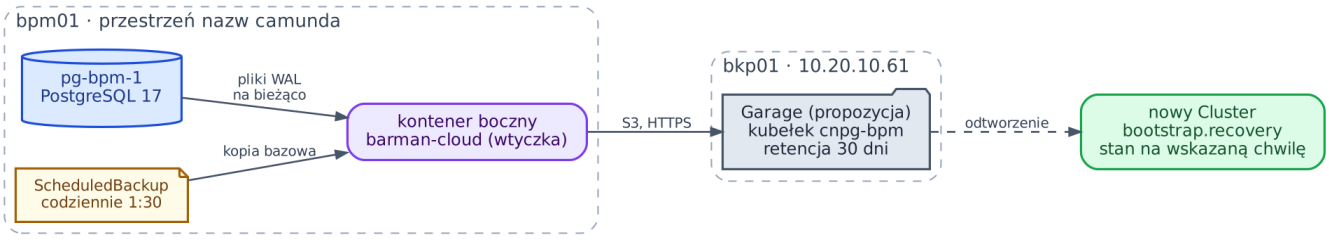
CloudNativePG to projekt CNCF, który dokumentacja Camundy wskazuje jako zalecany sposób trzymania PostgreSQL dla Management Identity, Keycloaka i Camunda Hub. Operator nie używa StatefulSetów: każda instancja to osobny pod z własnym PVC, a o tym, która jest główna, decyduje operator.

Obiekt	Co opisuje	U nas
`Cluster`	instancje PostgreSQL, obraz, wolumeny, role, wtyczki	`pg-bpm`, jedna instancja
`Database`	baza w klastrze z właścicielem	`identity`, `hub` (`camunda` zakłada `initdb`)
`ObjectStore` (wtyczka Barman Cloud)	magazyn kopii: adres S3, klucze, retencja	`bkp01-s3`
`ScheduledBackup`	harmonogram kopii bazowych	codziennie o 1:30
`Backup`	jedna kopia na żądanie	przed każdą zmianą

Operator zakłada też usługi: `pg-bpm-rw` (zawsze instancja główna, do zapisu), `pg-bpm-ro` (repliki) i `pg-bpm-r` (dowolna instancja). Aplikacje łączymy z `pg-bpm-rw`.

Kopie: WAL na bieżąco i kopia bazowa raz dziennie

Fizyczna kopia PostgreSQL to dwie rzeczy: kopia bazowa (zdjęcie plików bazy) i ciąg plików WAL od tej chwili. Razem pozwalają odtworzyć stan z dowolnej sekundy po kopii bazowej. Teorię opisuje strona Kopie logów transakcyjnych i odtwarzanie do punktu w czasie (PITR).



Rysunek 1. Kopie CloudNativePG na bpm01: instancja pg-bpm-1 oddaje pliki WAL na bieżąco do kontenera bocznego barman-cloud, ScheduledBackup codziennie o 1:30 robi kopię bazową, oba trafiają przez S3 do kubełka cnpg-bpm na bkp01, a odtworzenie zakłada nowy Cluster na wskazaną chwilę (kliknięcie otwiera pełny rozmiar)

> **Pojęcie: wtyczka Barman Cloud.** Dodatek do CloudNativePG, który wysyła WAL i kopie bazowe do magazynu obiektowego narzędziami `barman-cloud`. Działa jako kontener boczny w podzie bazy. Zastępuje kopie wbudowane w operatora (`.spec.backup.barmanObjectStore`), które w 1.30 są przestarzałe i według informacji o wydaniu znikną w 1.31.0.

> **Pojęcie: magazyn obiektowy zgodny z S3.** Serwer, który przechowuje pliki jako obiekty w kubełkach i rozmawia protokołem Amazon S3. Ten sam klient działa wtedy z Amazon S3, z Garage na własnym serwerze i z innymi zgodnymi magazynami.

Wtyczka nie umie pisać do repozytorium restic. Dlatego kopie fizyczne idą do magazynu S3 na `bkp01`, a restic dostaje tylko zrzuty logiczne z `/k8s/kopie`, jeżeli autor zdecyduje się je robić (sekcja o retencji niżej).

Retencja a polityka kopii Zielonej

Tabela poziomów na stronie Strategia kopii zapasowych Zielonej: RPO, RTO, reguła 3-2-1-1-0 i dwie lokalizacje daje bazom aplikacji (repozytorium `bazy`) poziom 180 dni. Kubełek z WAL i kopiami bazowymi przez 180 dni zajmuje dużo miejsca, bo trzyma każdy WAL.

Wariant	Co trzymamy	Koszt	Uwagi
A	`retentionPolicy: "30d"` w kubełku i codzienny zrzut logiczny do restic `bazy` (180 dni)	mało miejsca na Garage, dodatkowy skrypt	odtworzenie do sekundy przez 30 dni, starsze tylko do dnia zrzutu
B	`retentionPolicy: "180d"` w kubełku	dużo miejsca, jeden mechanizm	odtworzenie do sekundy przez 180 dni

W przykładach ustawiamy wariant A, a wybór zostawiamy autorowi.

Dlaczego PostgreSQL 17, a nie 18

Domyślny obraz CloudNativePG 1.30.1 to PostgreSQL 18.6, a Orchestration Cluster Camundy 8.10 wspiera 15 do 18. Management Identity według tabeli obsługiwanych środowisk Camundy 8.10 obsługuje jednak PostgreSQL 14 do 17, a stawiamy jeden klaster dla wszystkich baz. Przypinamy więc obraz `17.11-standard-trixie` z katalogu obrazów CloudNativePG z 26 października 2026 roku. Obrazy `standard` są przeznaczone do pracy z wtyczkami kopii, a starsze `system` są przestarzałe.

Przebieg

Zanim zaczniemy

Potrzebujemy węzła `bpm01` z klasą `local-path-retain` (Węzeł k3s dla aplikacji: bpm01 z dyskiem /k8s, local-path z Retain i próbny nginx), kubełka `cnpg-bpm` z kluczem dostępu na magazynie S3 i certyfikatu głównego urzędu Zielonej w pliku PEM (`zielona-ca.crt`), bo Garage na `bkp01` ma certyfikat z wewnętrznego urzędu. Pliki manifestów zapisujemy w `/k8s/helm/pg-bpm/`.

1. cert-manager i operator

Wtyczka Barman Cloud korzysta z cert-managera do certyfikatów między sobą a operatorem.

```
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.21.3/cert-manager.yaml
kubectl -n cert-manager rollout status deploy/cert-manager-webhook
kubectl apply --server-side -f https://raw.githubusercontent.com/cloudnative-pg/cloudnative-pg/release-1.30/releases/cnpg-1.30.1.yaml
kubectl rollout status deployment -n cnpg-system cnpg-controller-manager
kubectl apply -f https://github.com/cloudnative-pg/plugin-barman-cloud/releases/download/v0.15.1/manifest.yaml
kubectl -n cnpg-system rollout status deploy/barman-cloud
```

Wtyczka musi stać w tej samej przestrzeni co operator (`cnpg-system`), a jej manifest to zapewnia.

2. Wtyczka kubectl cnpg

```
curl -sSfL https://github.com/cloudnative-pg/cloudnative-pg/raw/main/hack/install-cnpg-plugin.sh | sudo sh -s -- -b /usr/local/bin
kubectl cnpg version
```

Skrypt pochodzi z gałęzi `main`, więc nie przypina wersji. Kto woli pakiet, bierze `.rpm` z wydania 1.30.1 na GitHubie.

3. Przestrzeń nazw i sekrety

Hasła generujemy i od razu zapisujemy w Vaultwardenie. Sekrety mają typ `kubernetes.io/basic-auth`, bo tego wymaga zarządzanie rolami w CloudNativePG, a etykieta `cnpg.io/reload` sprawia, że zmiana hasła trafia do bazy od razu.

```
kubectl create namespace camunda
for r in camunda identity hub; do
  kubectl -n camunda create secret generic pg-bpm-$r --type=kubernetes.io/basic-auth \
    --from-literal=username=$r --from-literal=password="$(openssl rand -base64 24)"
  kubectl -n camunda label secret pg-bpm-$r cnpg.io/reload=true
done
kubectl -n camunda create secret generic bkp01-s3-klucz \
  --from-literal=ACCESS_KEY_ID=IdentyfikatorKlucza --from-literal=ACCESS_SECRET_KEY=TajnyKlucz
kubectl -n camunda create secret generic zielona-ca --from-file=ca.crt=zielona-ca.crt
```

> (gdzie `IdentyfikatorKlucza` i `TajnyKlucz` to klucz kubełka `cnpg-bpm` z magazynu S3)

4. Magazyn kopii: ObjectStore

Plik `/k8s/helm/pg-bpm/objectstore.yaml`:

```
apiVersion: barmancloud.cnpg.io/v1
kind: ObjectStore
metadata:
  name: bkp01-s3
  namespace: camunda
spec:
  retentionPolicy: "30d"
  configuration:
    destinationPath: s3://cnpg-bpm/
    endpointURL: https://bkp01.ad.zielona.example:3900
    endpointCA:
      name: zielona-ca
      key: ca.crt
    s3Credentials:
      accessKeyId:
        name: bkp01-s3-klucz
        key: ACCESS_KEY_ID
```

```
      secretAccessKey:
        name: bkp01-s3-klucz
        key: ACCESS_SECRET_KEY
    wal:
      compression: gzip
    data:
      compression: gzip
```

Pola `destinationPath`, `endpointURL`, `endpointCA`, `s3Credentials`, `wal` i `data` pochodzą z definicji zasobu `objectstores.barmancloud.cnpg.io` w manifeście wtyczki 0.15.1, a `retentionPolicy` z jej dokumentacji. Region i styl adresowania kubełka (ścieżkowy albo w nazwie hosta) sprawdzamy w konfiguracji Garage, gdy pierwsza kopia zgłosi błąd adresu.

5. Klaster pg-bpm i bazy

Plik `/k8s/helm/pg-bpm/cluster.yaml`:

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: pg-bpm
  namespace: camunda
spec:
  instances: 1
  imageName: ghcr.io/cloudnative-pg/postgresql:17.11-standard-trixie
  storage:
    storageClass: local-path-retain
    size: 20Gi
  resources:
    requests:
      cpu: 500m
      memory: 1Gi
    limits:
      memory: 2Gi
  bootstrap:
    initdb:
      database: camunda
      owner: camunda
      secret:
        name: pg-bpm-camunda
```

```
  managed:
    roles:
      - name: identity
        ensure: present
        login: true
        passwordSecret:
          name: pg-bpm-identity
      - name: hub
        ensure: present
        login: true
        passwordSecret:
          name: pg-bpm-hub
  plugins:
    - name: barman-cloud.cloudnative-pg.io
      isWALArchiver: true
      parameters:
        barmanObjectName: bkp01-s3
---
apiVersion: postgresql.cnpg.io/v1
kind: Database
metadata:
  name: pg-bpm-identity
  namespace: camunda
spec:
  name: identity
  owner: identity
  cluster:
    name: pg-bpm
---
apiVersion: postgresql.cnpg.io/v1
kind: Database
metadata:
  name: pg-bpm-hub
  namespace: camunda
spec:
  name: hub
  owner: hub
  cluster:
    name: pg-bpm
```

```
kubectl apply -f /k8s/helm/pg-bpm/objectstore.yaml -f /k8s/helm/pg-bpm/cluster.yaml
kubectl -n camunda get cluster pg-bpm -w
kubectl cnpg status pg-bpm -n camunda
```

Jedna instancja wystarcza na pilotaż. Dokumentacja Camundy zakłada w swoim wzorcu osobne klastry dla każdej aplikacji (`pg-identity`, `pg-keycloak`, `pg-hub`, `pg-camunda`), po dwie instancje na różnych węzłach i osobny wolumen na WAL. Przy przejściu do produkcji na trzech węzłach rozdzielamy więc bazy i podnosimy `instances` do 2 albo 3 (decyzja autora).

6. Harmonogram kopii i pierwsza kopia

Plik `/k8s/helm/pg-bpm/kopie.yaml`:

```
apiVersion: postgresql.cnpg.io/v1
kind: ScheduledBackup
metadata:
  name: pg-bpm-codziennie
  namespace: camunda
spec:
  schedule: "0 30 1 * * *"
  backupOwnerReference: self
  cluster:
    name: pg-bpm
  method: plugin
  pluginConfiguration:
    name: barman-cloud.cloudnative-pg.io
---
apiVersion: postgresql.cnpg.io/v1
kind: Backup
metadata:
  name: pg-bpm-pierwsza
  namespace: camunda
spec:
  cluster:
    name: pg-bpm
  method: plugin
  pluginConfiguration:
    name: barman-cloud.cloudnative-pg.io
```

```
kubectl apply -f /k8s/helm/pg-bpm/kopie.yaml
kubectl get backups -n camunda -w
kubectl cnpg status pg-bpm -n camunda
```

Harmonogram CloudNativePG ma sześć pól, pierwsze to sekundy, więc `0 30 1 * * *` znaczy 1:30:00 każdego dnia. W `kubectl cnpg status` sprawdzamy, że archiwizacja WAL działa i że widać pierwszą kopię. Przed każdą zmianą (aktualizacja Camundy, operatora, PostgreSQL) robimy obiekt `Backup` z nazwą mówiącą, przed czym.

Wariant A z retencji dokłada codzienny zrzut logiczny: karta „Zrzut logiczny bazy na dysk węzła” dla baz `camunda`, `identity` i `hub` w zegarze systemd i `restic backup /k8s/kopie` do repozytorium `bazy` na `bkp01`. Skrypt piszemy według wzoru z kroku 7 strony o `bpm01`, z kontem REST `bazy`.

7. Odtworzenie na wskazaną chwilę

Odtwarzamy zawsze do nowego klastra obok, nigdy na działający. Plik `/k8s/helm/pg-bpm/odtworzenie.yaml`:

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: pg-bpm-odtw
  namespace: camunda
spec:
  instances: 1
  imageName: ghcr.io/cloudnative-pg/postgresql:17.11-standard-trixie
  storage:
    storageClass: local-path-retain
    size: 20Gi
  bootstrap:
    recovery:
      source: pg-bpm-kopia
      recoveryTarget:
        targetTime: "2026-11-06 15:30:00+01"
  externalClusters:
    - name: pg-bpm-kopia
      plugin:
        name: barman-cloud.cloudnative-pg.io
        parameters:
          barmanObjectName: bkp01-s3
```

```
    serverName: pg-bpm
```

> (gdzie `targetTime` to chwila, do której odtwarzamy, a `serverName` to nazwa klastra, który robił kopie)

```
kubectl apply -f /k8s/helm/pg-bpm/odtworzenie.yaml
kubectl -n camunda get cluster pg-bpm-odtw -w
kubectl cnpg psql pg-bpm-odtw -n camunda -- -d hub -c 'select count(*) from pg_tables;'
```

Odtworzony klaster nie archiwizuje WAL, dopóki nie dopiszemy mu sekcji `plugins` z osobnym magazynem albo osobną ścieżką, żeby nie mieszał swoich plików z kopiami `pg-bpm`. Aplikację przełączamy na nowy klaster zmianą adresu w jej konfiguracji (`pg-bpm-odtw-rw`) albo po weryfikacji przenosimy dane zrzutem do starego klastra. Taki test robimy raz na kwartał według strony Testy odtwarzania: plik, maszyna, baza i cała lokalizacja.

8. Aktualizacja

Co	Jak	Przerwa
operator CloudNativePG	`kubectl apply --server-side` z nowym manifestem po przeczytaniu informacji o wydaniu; potem operator podmienia menedżera instancji w podach	według dokumentacji dwa kroki: kontroler, potem menedżer w każdym podzie
wtyczka Barman Cloud	`kubectl apply` manifestu nowej wersji w `cnpg-system`	kopie wstrzymane na czas restartu wtyczki
PostgreSQL w obrębie 17 (17.11 do 17.12)	zmiana `imageName` w `cluster.yaml` i `kubectl apply`	przy jednej instancji restart, czyli krótka przerwa
PostgreSQL 17 do 18	zmiana `imageName` na obraz 18 z tą samą dystrybucją (`trixie`); operator uruchamia `pg_upgrade`	cały klaster niedostępny na czas aktualizacji; potem nowa kopia bazowa, bo odtwarzanie do chwili nie przechodzi przez granicę wersji

Na PostgreSQL 18 przechodzimy dopiero wtedy, gdy Management Identity oficjalnie go obsłuży albo gdy jej baza trafi do osobnego klastra. Przed każdą aktualizacją: obiekt `Backup`, odczyt informacji o wydaniu i zapis w GLPI, a w okresie od 15 listopada do 6 stycznia tylko w awarii.

Częste błędy

Objaw	Przyczyna	Co robimy
`kubectl apply` operatora kończy się błędem o rozmiarze adnotacji	zwykłe `apply` zamiast `--server-side`	powtarzamy z `--server-side`
Klaster wisi w `Setting up primary`, PVC `Pending`	zła nazwa klasy w `storage.storageClass`	`kubectl describe pvc -n camunda`, poprawna klasa `local-path-retain`
`kubectl cnpg status` pokazuje błąd archiwizacji WAL	zły klucz, adres, certyfikat albo kubełek	`kubectl logs -n camunda pg-bpm-1 -c plugin-barman-cloud` (nazwę kontenera sprawdzamy w `kubectl describe pod`)
Błąd certyfikatu przy połączeniu z `bkp01`	brak `endpointCA` przy certyfikacie z wewnętrznego urzędu	sekret `zielona-ca` i pole `endpointCA`
Rola `identity` nie może się zalogować	sekret nie ma typu `kubernetes.io/basic-auth` albo `username` różni się od nazwy roli	sekret z kroku 3, ta sama nazwa w `username` i w `managed.roles`
Kopie się udają, ale kubełek rośnie bez końca	brak `retentionPolicy` w `ObjectStore`	ustawiamy retencję według wariantu z tabeli
Odtworzenie kończy się błędem „nie znaleziono kopii”	zła `serverName` albo `targetTime` sprzed pierwszej kopii bazowej	nazwa klastra źródłowego i czas po pierwszej kopii
Po aktualizacji do 18 Management Identity nie startuje	wersja spoza listy obsługiwanych przez Identity	odtworzenie ze świeżej kopii na 17 albo osobny klaster dla Identity

□□ Ćwiczenia

1. Dlaczego odtwarzamy do nowego klastra `pg-bpm-odtw`, a nie na `pg-bpm`?

Odpowiedź

Bo działający klaster ma dane, które mogą być jeszcze potrzebne (wszystko po błędzie), a odtworzenie na niego nadpisałoby je bez odwrotu. Nowy klaster obok pozwala porównać dane, przełączyć aplikację po sprawdzeniu i w razie pomyłki wrócić do starego.

1. Kubełek ma retencję 30 dni. Pani Halina pyta o stan bazy faktur sprzed czterech miesięcy. Co możemy zrobić w wariancie A?

Odpowiedź

Kopii fizycznej sprzed czterech miesięcy już nie ma, ale w repozytorium restic `bazy` jest codzienny zrzut logiczny z tamtego dnia (poziom 180 dni). Odtwarzamy go `pg_restore` do osobnej bazy i szukamy danych; dokładnej godziny nie odtworzymy, tylko stan z chwili zrzutu.

1. Dlaczego na razie nie przechodzimy na PostgreSQL 18, skoro jest domyślny w CloudNativePG 1.30.1?

Odpowiedź

Bo w jednym klastrze trzymamy też bazę Management Identity, a tabela obsługiwanych środowisk Camundy 8.10 wymienia dla niej PostgreSQL 14 do 17. Przejście na 18 ma sens, gdy Identity to obsłuży albo gdy jej baza będzie w osobnym klastrze.

1. Czym różni się usługa `pg-bpm-rw` od `pg-bpm-ro` i z którą łączymy Camundę?

Odpowiedź

`pg-bpm-rw` zawsze wskazuje instancję główną, która przyjmuje zapisy, a `pg-bpm-ro` tylko repliki do odczytu. Camunda zapisuje, więc łączymy ją z `pg-bpm-rw`; po przełączeniu instancji głównej operator sam przestawia tę usługę.

Powiązane: Węzeł k3s dla aplikacji: bpm01 z dyskiem /k8s, local-path z Retain i próbny nginx · Camunda 8 Self-Managed przez Helm: instalacja próbna na bpm01 · Architektura PostgreSQL: klaster, baza, schemat, procesy · Logi WAL: włączenie i archiwizacja · Upgrade silnika PostgreSQL · Kopie logów transakcyjnych i odtwarzanie do punktu w czasie (PITR) · Serwery kopii bkp01 i bkp02: restic, kopia krzyżowa i niezmienność

Źródła (odczyt 1 listopada 2026):

1. cloudnative-pg.io: Installation and upgrades 1.29 (polecenie `kubectl apply --server-side`, `cnpg-system`, aktualizacja w dwóch krokach) i Release notes v1.30 (1.30.0 z 29 czerwca 2026, 1.30.1 z 23 września 2026, Kubernetes 1.34 do 1.36, PostgreSQL 14 do 18, domyślny 18.6, wbudowane kopie Barman Cloud przestarzałe i do usunięcia w 1.31.0)
2. cloudnative-pg.io: Declarative database management, Declarative role management, kubectl plugin, PostgreSQL upgrades
3. cloudnative-pg.io: Barman Cloud Plugin (0.15.1, CloudNativePG 1.26 lub nowszy, cert-manager), Installation, Usage, Migration, Retention; github.com/cloudnative-pg/plugin-barman-cloud, `manifest.yaml` wydania v0.15.1 (pola `ObjectStore`)
4. github.com/cloudnative-pg/artifacts: `image-catalogs/catalog-standard-trixie.yaml` z 26 października 2026 (PostgreSQL 17.11, 18.6); github.com/cloudnative-pg/postgres-containers, README (obrazy `minimal`, `standard`, przestarzałe `system`)
5. cert-manager.io: Installation (v1.21.3)
6. docs.camunda.io: Supported environments (Management Identity: PostgreSQL 14 do 17) i Deploy required dependencies with Kubernetes operators (CloudNativePG 1.30, klastry

`pg-identity`, `pg-keycloak`, `pg-hub`, `pg-camunda`)
7. archlinux.org: pakiet `garage` 2.5.0 (27 października 2026, AGPL-3.0)