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.

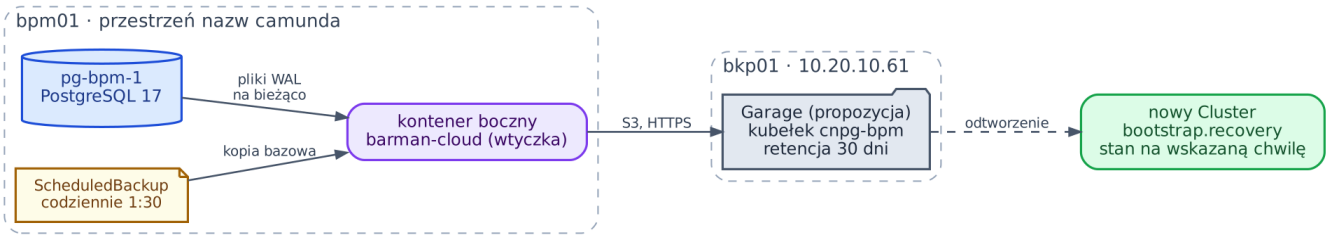
CloudNativePG to projekt CNCF, który dokumentacja Camundy wskazuje jako zalecany sposób trzymania PostgreSQL dla Management Identity, Keycloaka i Camunda Hub. Operator nie używa StatefulSetów: każda instancja to osobny pod z własnym PVC, a o tym, która jest główna, decyduje operator.

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)

Revision #2
Created 2026-11-01 18:02:08 UTC by Tomasz Konieczny
Updated 2026-11-01 18:03:33 UTC by Tomasz Konieczny