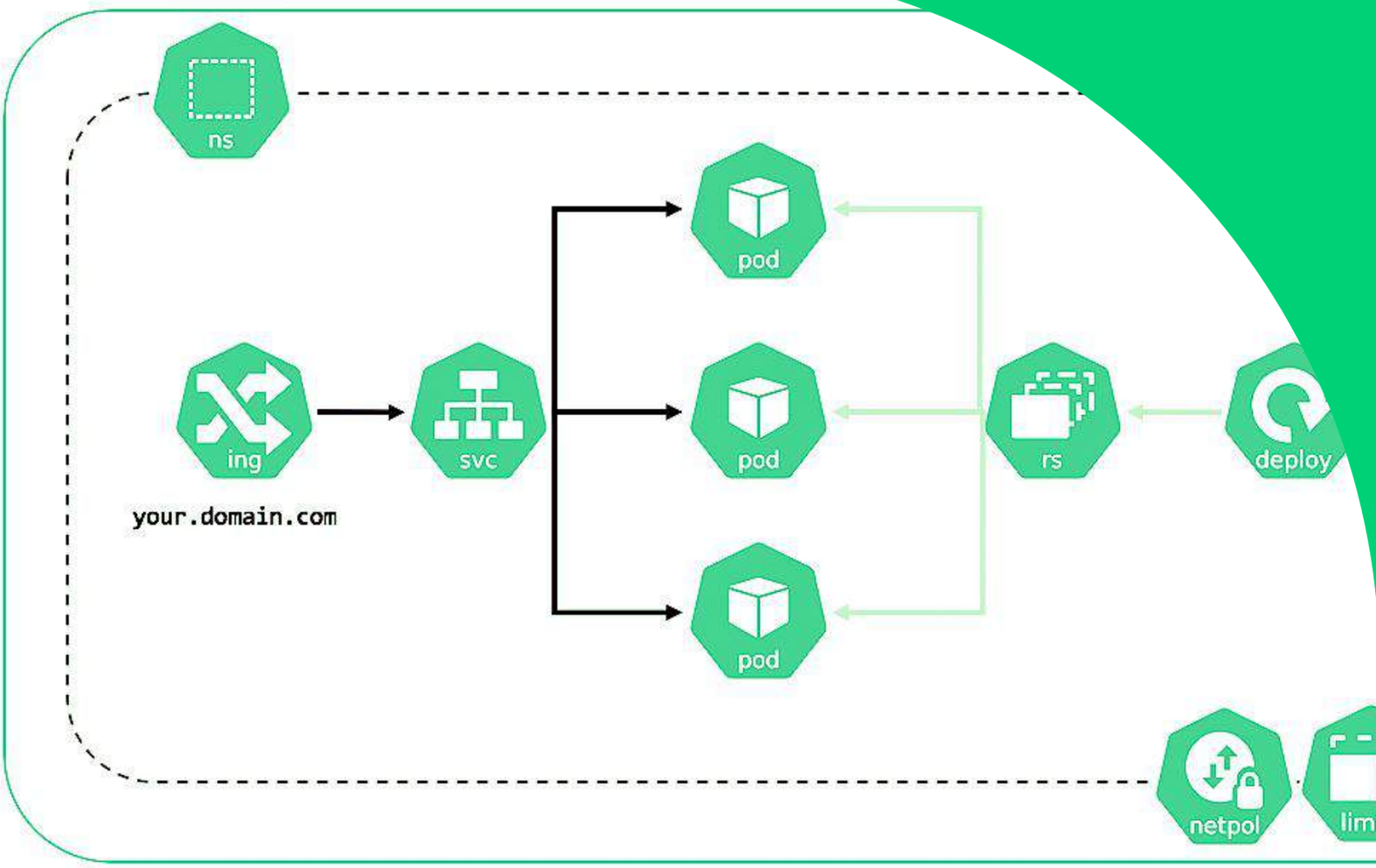
core • logic

Raport

Przewodnik po aktualizacji klastra Kubernetes

Migracja z wersji 1.25 do wersji 1.26
Pod Security Policy → Pod Security Admission



Spis treści [1/2]

1. Aktualizacja usług korzystających z PSP 5
2. Wykonanie pełnej kopii zapasowej klastra 6
3. ArgoCD 7
4. Działania wstępne przed wdrożeniem MetalLb w AgroCD 10
5. Deklaracja manifestów, configmag 12
6. Przyrównanie Agro CD do środowiska 15
7. Aktualizacja MetalLB 17
8. Portworx 18
9. Przygotowanie manifestu Storage Cluster 19
10. Aktualizacja manifestu usługi 20
11. Przygotowanie portworx-operator 21
12. Przygotowanie manifestu Storage Class 22
13. Przygotowanie manifestów 23
14. Modyfikacja manifestów 25
15. Aktualizacja Portworx 26
16 Prometheus Stack 27
17. Przygotowanie skryptów oraz values.yml 28

Spis treści [2/2]

18. Przygotowanie punktu odzyskiwania ____ 29
19. Surowy manifest dla aktualnej wersji dostępnej na klastrze ____ 31
20. Przyrównanie ArgoCD do klastra ____ 33
21. Aktualizacja Prometheus Stack ____ 36
22. Blackbox Exporter ____ 37
23. Przygotowanie ArgoCD do klastra ____ 38
24. Kubernetes - aktualizacja Nodów Control-Plane ____ 39
25. Aktualizacja klastra Kubernetes ____ 40

Wstęp

Przedstawiamy Państwu raport "Przewodnik po aktualizacji klastra Kubernetes - migracja z wersji 1.25 do wersji 1.26 - Pod Security Policy → Pod Security Admission”

Każda ewolucja technologii przynosi ze sobą wyzwania, ale także nowe możliwości. W najnowszej wersji klastra Kubernetes, 1.25, odchodzi PodSecurityPolicy (PSP) na rzeczbardziej elastycznej i nowoczesnej metody, zwanej Pod Security Admission (PSA). W tym raporcie omówiono kluczowe kroki, które należy podjąć przed aktualizacją klastra, skupiając się na usługach, które do tej pory korzystały z PSP.

Konrad Matyas

Wiceprezes Core Logic

1. Aktualizacja usług korzystających z PSP

Na początku omówione zostaną kluczowe elementy zarządzania oraz monitorowania klastra w kontekście niezbędnych aktualizacji i zgodności z najnowszymi standardami bezpieczeństwa. W każdym przypadku istotne jest posiadanie najnowszych wersji narzędzi, zoptymalizowanych pod kątem współpracy z aktualnymi politykami bezpieczeństwa oraz zapewnienia stabilności i niezawodności infrastruktury klastra.

1. Metallb - zarządzanie LoadBalancer:

Metallb odgrywa kluczową rolę w zarządzaniu usługami LoadBalancer w klastrze. Przed migracją trzeba posiadać najnowszą wersję Metallb, zoptymalizowaną pod kątem współpracy z PodSecurity Admission.

2. Portworx - zarządzanie storage:

W przypadku korzystania z Portworx do zarządzania storage, należy posiadać najnowszą wersję, w pełni zgodną z nową polityką bezpieczeństwa.

3. Prometheus Stack -monitorowanie klastra:

Jeśli używane są narzędzia z pakietu Prometheus do monitorowania klastra, trzeba zaktualizować prometheus-stack do najnowszej wersji, aby uniknąć problemów związanych z bezpieczeństwem.

4. Blackbox Exporter -monitorowanie zdalnych usług

Należy upewnić się, że blackbox-exporter, odpowiedzialny za monitorowanie zdalnych usług, jest zgodny z nowymi standardami bezpieczeństwa.
Listę koniecznych do aktualizacji zasobów można wyświetlić za pomocą polecania:

```
kubectl get podsecuritypolicies.policy -A
```

```
Warning: policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25+
NAME                                              PRIV
blackbox-exporter-prometheus-blackbox-exporter-psp   false
controller                                        false
monitoring-grafana                                false
monitoring-grafana-test                           false
monitoring-kube-prometheus-alertmanager           false
monitoring-kube-prometheus-operator               false
monitoring-kube-prometheus-prometheus             false
monitoring-kube-state-metrics                     false
monitoring-prometheus-node-exporter               false
px-operator                                       false
speaker                                           true
```

Gdzie:

- px-operator = portworx,
- speaker, controller = metallb
- monitoring = prometheus
- blackbox-exporter = blackbox-exporter

2. Wykonanie pełnej kopii zapasowej klastra

Procedura backup jest analogiczna do procesu przygotowań do aktualizacji Kubernetes (k8s) do wersji1.24.

Poniżej przedstawiono instrukcje dotyczące wykonywania kopii zapasowej klastra Kubernetes, obejmujące kopiowanie certyfikatów, manifestów oraz danych ETCD.

Aby zabezpieczyć certyfikaty oraz manifesty klastra Kubernetes, można skorzystać z narzędzia rsync:

```
rsync -r --exclude=tmp user@host:/etc/kubernetes/ /home/user/k8s-backup/
```

Narzędzie rsync umożliwia skopiowanie folderów/etc/kubernetes/pki/ oraz /etc/kubernetes/manifests/ zachowując strukturę katalogów. Dodatkowo, można wykluczyć zbędne foldery, na przykład tmp.

Kopia ETCD

Przed wykonaniem kopii zapasowej ETCD, należy potwierdzić lokalizację certyfikatów oraz endpointu za pomocą polecenia:

```
kubectl -n kube-system describe pod etcd-tst-traficar-cloud-master | grep -E 'listen-client-urls|cert-file|key-file|trusted-ca-file'
```

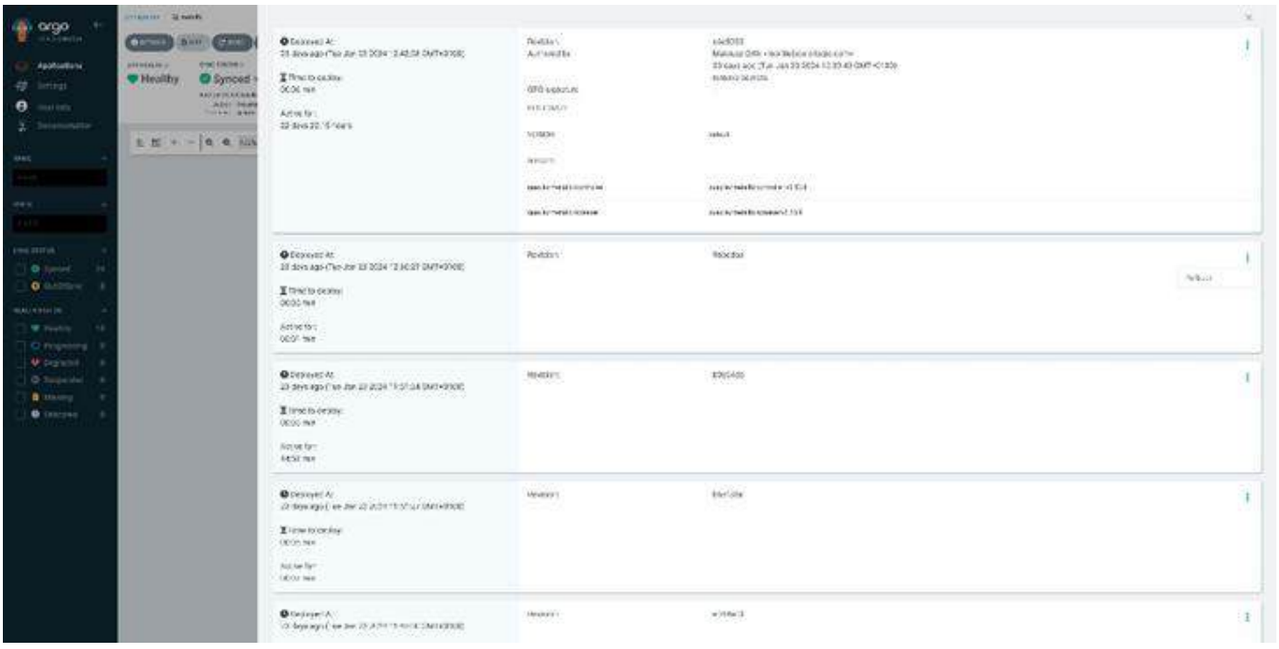
Następnie można wykonać kopię zapasową ETCD, używając poniższego polecenia:

```
ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 \
--cacert=/etc/kubernetes/pki/etcd/ca.crt \
--cert=/etc/kubernetes/pki/etcd/server.crt \
--key=/etc/kubernetes/pki/etcd/server.key \
snapshot save /opt/snapshot-pre-boot.db
```

Powyższe polecenie tworzy kopię zapasową ETCD w pliku snapshot-pre-boot.db.

3. ArgoCD

ArgoCD to niezbędne narzędzie do bezpiecznej i kontrolowanej aktualizacji serwisów w klastrze Kubernetes. Dzięki ArgoCD istnieje możliwość wizualnej analizy różnic pomiędzy przygotowanym manifestem a obecnym stanem w środowisku. Jednak największą korzyścią jest elastyczna opcja szybkiego cofnięcia (rollback) do poprzedniej wersji, co czyni ArgoCD niezastąpionym narzędziem przy aktualizacji praktycznie każdego obiektu w klastrze.



Przygotowanie aplikacji w ArgoCD

W celu skonfigurowania aplikacji w ArgoCD, warto postępować deklaratywnie, opierając się na zdefiniowanym w repozytorium cloud-gitops zestawie manifestów. Choć panel ArgoCD umożliwia bezpośrednie tworzenie aplikacji, w artykule skupiono się na podejściu opartym na deklaracjach w repozytorium, co umożliwia spójne zarządzanie aplikacjami w różnych systemach.

Struktura manifestu aplikacji

Deklaracja aplikacji powinna być dostosowana do ogólnej struktury manifestu, co umożliwi łatwą adaptację do konkretnych usług. W przypadku tego opisu, przyjęto, że metodologia używana w niniejszym przypadku wykorzystuje repozytorium cloud-gitops.

3. ArgoCD

Sama deklaracja opiera się na stworzeniu nowego pliku\manifestu w ścieżce

```
argocd/{{target_env}}/{{target_app_name}}.yml
```

Manifest powinien zawierać następujący wpis:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: {{target_app_name}}
  namespace: argocd
spec:
  destination:
    name: ''
    namespace: default
    server: 'https://kubernetes.default.svc'
  source:
    path: {{target_app_path}}
    repoURL: {{target_app_repo}}
    targetRevision: {{target_app_repo_branch}}
  project: default

# {{target_app_name}}               - Nazwa aplikacji
# {{target_app_path}}               - Ścieżka do deklaracji / manifestu
# {{target_app_repo}}               - Link do repozytorium Git
# {{target_app_repo_branch}}        - Branch z powyższego repo
```

Tak wygląda przykładowa deklaracja dla Metallb prod:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: metallb
  namespace: argocd
spec:
  destination:
    name: ''
    namespace: default
    server: 'https://kubernetes.default.svc'
  source:
    path: 'kustomize/metallb/overlays/prod'
    repoURL: 'https://repository.domain.com/project/gitops'
    targetRevision: master
  project: default
```

3. ArgoCD

Warto tworzyć nowe aplikacje na osobnych branchach w repozytorium kodu GIT.
To podejście pozwala zachować bezpieczeństwo pozostałych elementów klastra,
gdy wprowadzane są zmiany w danym projekcie. Każda aplikacja powinna mieć swoją
dedykowaną przestrzeń roboczą, co ułatwia kontrolę i monitorowanie zmian.

W przypadku wcześniej stworzonych aplikacji, które są już dostępne w klastrze, zaleca się
wyłączenie funkcji automatycznej synchronizacji (Auto-Sync). Warto także upewnić się,
że Auto-Sync jest wyłączone lub w razie potrzeby, wyłączyć je ręcznie na stworzonym branchu.
Można to zrobić poprzez modyfikację wartości {{target_app_repo_branch}} w definicji aplikacji.

Konieczne jest wówczas usunięcie wpisu:

```
syncPolicy:
   automated:
     prune: false
     selfHeal: true
```

Po wykonaniu koniecznych operacji można zsynchronizować główną aplikację ArgoCD.

kzadlo@core-logic.com
www.core-logic.com

4. Działania wstępne przed wdrożeniem MetalLB w ArgoCD

Aktualizacja MetalLB w kontekście ArgoCD wymaga odpowiedniego przygotowania klastra, szczególnie jeśli chodzi o przypisanie i zarządzanie adresami IP. Poniżej opisano kluczowe kroki oraz ważne uwagi dotyczące prawidłowego przypisywania adresów IP przed operacją wdrożenia Metallb.

Prawidłowe Przypisanie Adresów IP

Przypisanie adresów IP dla usług w klastrze, szczególnie w kontekście ArgoCD, nie jest wymagane. Jednakże, zaleca się utrzymanie sztywnej konfiguracji adresów IP dla obiektów, które tego wymagają. Warto weryfikować, które usługi wymagają stałego przypisania adresów IP, a które mogą korzystać z dynamicznie przydzielonych adresacji. Przed dalszymi krokami należy się upewnićczy klaster jest gotowy do aktualizacji. W przypadku MetalLB konieczne jest zadeklarowanie adresów IP dla serwisów, zwłaszcza jeśli są one wymagane nastałe.

Utrzymywanie Stabilnych Adresów IP

Warto upewnić się, że usługi mają przypisane na stałe adresy IP, szczególnie w przypadku serwisów korzystających z DNS lub posiadających zadeklarowane stałe adresy IP w innych serwisach.
W przypadku klastra, dla którego opisujemy proces aktualizacji, nie wszystkie usługi miały przypisane adresy IP przed aktualizacją, co skutkowało przypisaniem losowych adresów z puli. To może prowadzić do nieoczekiwanych zmian w przypadku usług opierających się na określonych adresach IP.

Możemy sprawdzić to za pomocą polecenia:

```
kubectl get svc -A -o custom-columns="NAME":.metadata.name,"TYPE":.spec.type,"CURRENT_IP":.status.loadBalancer
.ingress[0].ip,"REQUESTED_IP":.spec.loadBalancerIP | grep LoadBalancer
```

4. Działania wstępne przed wdrożeniemMetalLB w ArgoCD

W przypadku braku adresu należy zdefiniować go manifeście, np.:

```
apiVersion: v1
kind: Service
metadata:
  name: eaw-documentation
  namespace: default
spec:
  type: LoadBalancer
```

Manifest zmieniamy na:

```
apiVersion: v1
kind: Service
metadata:
  name: eaw-documentation
  namespace: default
spec:
  type: LoadBalancer
  loadBalancerIP: 10.220.15.166
```

W przypadku usługi ArgoCD adres należy przypisać ręcznie edytując manifest bezpośrednio na klastrze

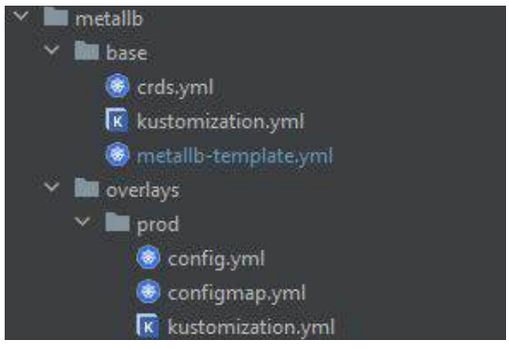
```
kubectl edit svc -n argocd argocd-server-external
```

5. Deklaracja manifestów, configmap oraz IPAddressPool

Na początku konieczne jest stworzenie ścieżki zadeklarowanej w samej aplikacji ArgoCD, w tym przypadku stworzono strukturę folderów.

W kontekście opisywanego klastra korzystanie z narzędzia Kustomize przynosi dodatkową wartość, choć nie jest ono wymagane do poprawnej konfiguracji. Użycie Kustomize może znacząco zwiększyć przejrzystość i ułatwić pracę nad konfiguracją klastra.

Tak wygląda gotowa struktura plików:



W opisywanym klastrze dotychczas MetalLB nie był wdrożony za pomocą ArgoCD, co wymaga teraz pełnej odbudowy konfiguracji oraz skonfrontowania jej z aktualnym stanem zarządzanym przez ArgoCD.

Przygotowanie configmap oraz IPAddressPool

Aby przygotować klaster do aktualizacji, istnieją pewne niezbędne kroki dotyczące konfiguracji, takie jak przygotowanie configmap oraz IPAddressPool. Ważne jest, że są one niezbędne do aktualizacji klastra, nie trzeba ich wykonywać podczas porównywania ArgoCD do k8s.

W nowej wersji jednak ta konfiguracja uległa zmianie. Teraz nowa metoda przypisywania adresów wymaga przygotowania configmap oraz IPAddressPool. Aby dostosować się do tych zmian, niezbędne jest zaznajomienie się z nowymi procedurami zarządzania adresacją. Warto podkreślić, że te kroki są konieczne do prawidłowej aktualizacji klastra, ale nie są wymagane podczas operacji porównywania ArgoCD do k8s.

W kontekście zmian w sposobie przypisywania i zarządzania adresacją po aktualizacji, warto zaznaczyć, że w poprzedniej wersji opierało się to na configmapie, którą można było przejrzeć za pomocą polecenia:

```
kubectl get configmaps -n metallb-system config -o yaml
```

5. Deklaracja manifestów, configmap oraz IPAddressPool

Prezentuje się ona następująco:

```
apiVersion: v1
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
        - 10.10.105.100-10.10.105.200
kind: ConfigMap
metadata:
  name: config
  namespace: metallb-system
```

W wyniku wprowadzonych zmian konieczne jest zastosowanie dwóch dodatkowych manifestów, które modyfikują sposób zarządzania adresami. W opisywanym przypadku konieczne było otwarcie dodatkowego portu, bez którego jeden z webhooków nie będzie działał. Opisywana powyżej operacja była wymagana tylko w przypadku opisywanego klastra i wynika bezpośrednio z jego specyfiki oraz funkcjonujących usług.

```
apiVersion: crd.projectcalico.org/v1
kind: NetworkPolicy
metadata:
  name: default.allow-metallb-controller-ingress
  namespace: metallb-system
spec:
  ingress:
  - action: Allow
    destination:
      ports:
      - 9443
    protocol: TCP
    source:
      namespaceSelector: global()
      selector: has(node-role.kubernetes.io/control-plane)
  selector: component == 'controller'
  types:
  - Ingress
```

W celu stworzono dwa dodatkowe manifesty w ścieżce /metallb/overlays/prod/configmap.yml zawierający kopię ConigMapy:

```
apiVersion: v1
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
        - 10.220.21.91-10.220.21.99
        - 10.220.21.50-10.220.21.59
kind: ConfigMap
metadata:
  name: config
  namespace: metallb-system
```

5. Deklaracja manifestów, configmap oraz IPAddressPool

Oraz manifest zawierający nowe deklaracje i adresację conifg.yml:

```yaml
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: first-pool
  namespace: metallb-system
spec:
  addresses:
  - 10.220.21.91-10.220.21.99
  - 10.220.21.50-10.220.21.59
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: default-advertisement
  namespace: metallb-system
```

kind: L2 Advertisement wymaga wcześniejszego wdrożenia pliku /metallb/base/crds.yml. Bez wcześniejszej deklaracji CustomResourceDefinition zawierającej ten wpis nie będzie on działał.
Jak zostało wspomniane powyżej w przypadku opisywanego klastra konieczna była deklaracja manifestu otwierająca dodatkowy port 9443 bez którego nie zadziałałby jeden z WebHooków Przykład deklaracji w poniższym pliku config.yml

```yaml
apiVersion: crd.projectcalico.org/v1
kind: NetworkPolicy
metadata:
  name: default.allow-metallb-controller-ingress
  namespace: metallb-system
spec:
  ingress:
  - action: Allow
    destination:
      ports:
      - 9443
    protocol: TCP
    source:
      namespaceSelector: global()
      selector: has(node-role.kubernetes.io/control-plane)
  selector: component == 'controller'
  types:
  - Ingress
```

Ostatecznie konieczne jest zadeklarowanie nowych plików w /metallb/overlays/prod/kustomization.yml

```yaml
resources:
  - ../../base
  - configmap.yml
  - config.yml
```


kzadlo@core-logic.com
www.core-logic.com

6. Przyrównanie ArgoCD do środowiska

Pierwszym krokiem jest konieczność przygotowania manifestu dla aktualnej wersji MetalLB. Jest to niezbędne, ponieważ w opisywanym przypadku brakowało manifestu dla tej aplikacji po stronie Repozytorium jak i ArgoCD. Bez tego nie będziemy w stanie bezpiecznie i sprawnie cofnąć operacji czyli przeprowadzić operacji „rollback".

W ArgoCD rollback polegana przekształceniu do poprzedniego, stabilnego stanu z ostatniego commitu wskazanego repozytorium.

Najskuteczniejszym sposobem na porównanie środowisk jest wygenerowanie szablonu dla konkretnej aplikacji za pomocą Helma. Należy określić aktualną wersję MetalLB i wygenerować szablon za pomocą:

```
helm template --versoion={{target_version}} --include-crds=true --namespace=metallb-system metallb metallb/metallb
> metallb-template.yml
```

Ten proces umożliwi dokładne określenie różnic między obecnym stanem a wcześniejszą wersją aplikacji MetalLB, co jest kluczowe w celu ewentualnego wykonania skutecznej i bezpiecznej operacji rollback.

Aktualną wersję można określić opisując pod controller lub speaker:

```
kubectl describe pod controller-7cf77c64fb-pr8t7 -n metallb-system | grep Image:
    Image:         quay.io/metallb/controller:v0.11.0
```

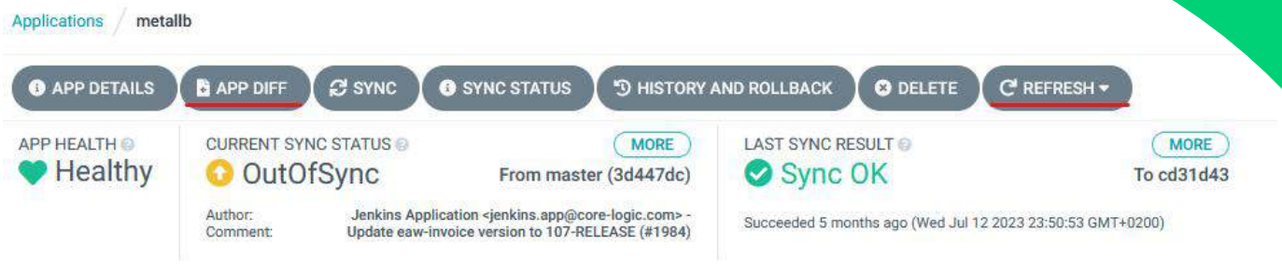
Praktyką zalecaną jest podzielenie manifestu na dwa osobne pliki. Jeden z nich powinien zawierać wszystkie wpisy dotyczące CustomResourceDefinition, a drugi - pozostałe obiekty. Ostatnim etapem jest dodanie odpowiednich wpisów do pliku/metallb/base/kustomization.yml.

```
resources:
  - crds.yml
  - metallb-template.yml
```

Uwaga!

Niezwykle istotne jest, aby w kolejnych etapach nie dokonywać synchronizacji aplikacji po stronie ArgoCD. Obecnie naszym celem jest jedynie weryfikacja różnic i porównanie manifestu w ArgoCD z rzeczywistym stanem w klastrze. Po stronie ArgoCD ograniczono się do korzystania jedynie z dwóch przycisków: "Refresh" oraz "App Diff".

6. Przyrównanie ArgoCD do środowiska



Przycisk "AppDiff" jest kluczowy do porównania zmian między ArgoCD a Klastrem. Zaleca się skorzystanie z opcji "Compact diff", dostępnej w górnym prawym rogu, aby uprościć analizę różnic.

Naszym zadaniem jest eliminacja wszelkich różnic poprzez modyfikacje bezpośrednio w manifeście, a następnie wykonanie commita i pusha do repozytorium. Warto również regularnie porównywać zmiany za pomocą "App Diff".
W przypadku MetalLB można również wykorzystać plik values.yaml, który może być używany podczas generowania szablonu, dodając flagę -f values.yaml do polecenia. Warto jednak zauważyć, że funkcjonalność ta może być ograniczona w przypadku MetalLB, a plik values.yaml będzie bardziej użyteczny przy aktualizacji prometheus-stack.

Po skutecznym porównaniu ArgoCD z klastrze i upewnieniu się, że "App Diff" nie wykazuje znaczących zmian, można przystąpić do synchronizacji za pomocą przycisku "Sync". Dzięki temu procesowi zyskujemy punkty przywracania, a samo wdrożenie MetalLB przez ArgoCD zostaje zakończone sukcesem.

Przygotowanie manifestu

Procedura ta przeważnie nie różni się znacząco od porównywania ArgoCD do środowiska. Kluczową modyfikacją jest przede wszystkim generacja pliku szablonu z najnowszą wersją MetalLB za pomocą polecenia:

```
helm template --include-crds=true --namespace=metallb-system metallb metallb/metallb  > metallb-template.yml
```

Należy pamiętać o konieczności wykonania pozostałych operacji opisanych w poprzednim etapie. Warto jednak zauważyć, że ze względu na aktualizację wersji mogą wystąpić istotne zmiany, co będzie widoczne w wynikach narzędzia App Diff. Istotne jest, aby kluczowe elementy, takie jak 'name' czy 'namespace', zachowały swoje pierwotne wartości. W praktyce oznacza to, że nawet w przypadku znacznych modyfikacji wersji, te kluczowe atrybuty powinny pozostać niezmienione. Pamiętajmy, że różnice w App Diff mogą być dość znaczące, dlatego istnieje potrzeba świadomego monitorowania tych zmian i ewentualnej ręcznej korekty, aby zapewnić spójność i poprawność konfiguracji środowiska.

7. Aktualizacja MetalLB

Po wygenerowaniu manifestu dla nowej wersji, przystępujemy ostatecznie do synchronizacji aplikacji w ArgoCD.

Warto pamiętać, że przed wprowadzeniem aktualizacji konieczne jest przygotowanie konfiguracji dla configmap oraz IPAddressPool. Zaleca się rozpoczęcie od synchronizacji obiektów CRD, ponieważ są one niezbędne do konfiguracji i warto je wdrożyć przed aktualizacją.

CRD często są kompatybilne wstecz, co oznacza, że ich aktualizacja nie powinna wpływać negatywnie na działanie klastra czy usług. Jednakże, w przypadku CRD, istnieje konieczność wykonania synchronizacji z opcją przełącznika "Replace". Pominięcie tego kroku może prowadzić do błędów synchronizacji w ArgoCD.

Po zsynchronizowaniu obiektów CRD, można przejść do synchronizacji pozostałych elementów. Po zakończeniu tego procesu w ArgoCD, zaleca się monitorowanie stanu podów w namespace metallb-system za pomocą polecenia:

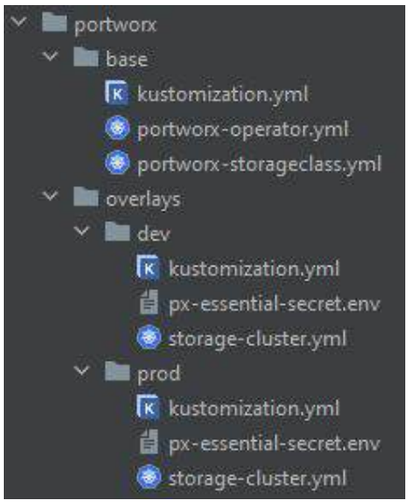
```
watch kubectl get pod -o wide -n metallb-system
```

Każdy z podów powinien przejść przez proces restartu i ostatecznie osiągnąć status "Ready1/1" oraz "Status Running". Monitorowanie tego procesu pomaga upewnić się, że aktualizacja została pomyślnie wdrożona, a wszystkie podsystemy stabilizują się poprawnie.

8. Portworx

Przed rozpoczęciem implementacji Portworx w klastrze Kubernetes z wykorzystaniem ArgoCD, istnieje kilka kroków wstępnych, które warto przeprowadzić.

Aby skutecznie zarządzać Portworx za pomocą ArgoCD, zaleca się stworzenie osobnej aplikacji dedykowanej tej usłudze. W tym celu warto utworzyć strukturę plików zgodną ze ścieżką zadeklarowaną w konfiguracji aplikacji ArgoCD.



Podczas organizowania struktury plików dla Portworx, należy sprawdzić, czy ścieżka jest zgodna z tą zadeklarowaną w konfiguracji aplikacji ArgoCD. W powyższym przykładzie stworzono folder portworx, zawierający plik kustomization.yaml oraz strukturę dla ewentualnych overlay'ów. Plik portworx.yaml będzie zawierał specyficzne konfiguracje dla Portworx.

Przyrównanie ArgoCD do klastra

Jeśli Portworx wcześniej nie był zarządzany przez ArgoCD, konieczne jest odtworzenie aktualnego środowiska. Podobnie, jak w przypadku MetalLB, ten krok pozwoli nam utworzyć punkt przywracania. W przypadku istnienia wcześniejszej konfiguracji, wskazane jest przeprowadzenie niezbędnej czynności w celu przywrócenia środowiska.

W następnym kroku trzeba uruchomić proces synchronizacji aplikacji w ArgoCD, aby wprowadzić zmiany zdefiniowane w plikach konfiguracyjnych.

9. Przygotowanie manifestu StorageCluster

Jeśli proces wdrożenia oraz potencjalne wcześniejsze aktualizacje zostały poprawnie wykonane, manifest dla StorageCluster powinien być dostępny do pobrania z panelu Portworx. Poniżej znajdują się niezbędne czynności do wykonania manifestu oraz ewentualnej aktualizacji usługi.

W panelu Portworx, w zakładce "Install and Run", warto skorzystać z opcji "Download", dostępnej poprzez kliknięcie przycisku "Actions".

Po pobraniu manifestu, trzeba upewnić się, czy wartość dla wpisu portworx.io/install-source ma zgodne wartości dla kluczy user= oraz c=, a następnie, czy wpis name: jest zbieżny z portworx.io/install-source c=

```
portworx.io/install-source: "https://install.portworx.com/?operator=true&mc=false&kbver=1.25.12&oem=esse&user=190e038c-f42f-
-f42f-11eb-a2c5-eb-a2c5-c24e499c7467&b=true&j=auto&c=px-cluster-2c55c9b4-b839-470b-9ad9-7a46d3b07cf2&stork=false&csi=
true&mon=true&tel=true&st=k8s&promop=true"
```

```
name: px-cluster-2c55c9b4-b839-470b-9ad9-7a46d3b07cf2
```

W przypadku różnic, zaleca się pobranie danych bezpośrednio z klastra i nadpisanie ich w manifeście. Gotowy manifest należy zapisać w pliku /portwrox/overlays/dev/storage-cluster.yml.

10. Aktualizacja manifestu usługi

W przypadku aktualizacji usługi Portworx, wystarczy dokonać zmian w odpowiednich sekcjach manifestu.

Zmiana wersji Kubernetes (kbver=):

- Znalezienie w pliku manifestu wpis portworx.io/install-source.
- Zaktualizowanie wartość kbver = na odpowiednią wersję Kubernetes, np. 1.25.11.

Weryfikacja wpisów (name:):

- Zweryfikowanie czy wartość pod name: jest zgodna z poprzednią wersją.
- Wprzypadku różnic, dostosowanie wartość name: zgodnie z poprzednią konfiguracją.

Weryfikacja sekcji (user=):

- Sprawdzenie czy wartość user = w sekcji portworx.io/install-source jest zgodna z poprzednią wersją.
- Upewnienie się, że klucz user = jest zgodny z wcześniejszym ustawieniem.

Po dokonaniu tych zmian, manifest usługi Portworx jest gotowy do zastosowania w klastrze Kubernetes. Trzeba pamiętać o sprawdzeniu poprawności konfiguracji i zastosowaniu zmian zgodnie z procedurami aktualizacyjnymi.

```
apiVersion: core.libopenstorage.org/v1
kind: StorageCluster
metadata:
  annotations:
# Różnica jedynie w wersji k8s. W razie konieczności podmienić c= na wartość z name:
    portworx.io/install-source: "https://install.portworx.com/?operator=true&mc=false&kbver=1.25.11&oem=esse&u
    ser=190e038c-f42f-11eb-a2c24e499c7467&b=true&j=auto&c=px-cluster-2c55c9b4-b839-470b-9ad9-7a46d3b07cf2&stor
    k=false&csi=true&mon=true&tel=true&st=k8s&promop=true"

    portworx.io/misc-args: "--oem esse"
  finalizers:
    - operator.libopenstorage.org/delete
# Wpis musi być jednakowy z poprzednią wersją. Przy generacji menifestu zmienia się każdorazowo.
  name: px-cluster-2c55c9b4-b839-470b-9ad9-7a46d3b07cf2
  namespace: kube-system
spec:
  image: portworx/oci-monitor:2.13.6
  imagePullPolicy: Always
```

11. Przygotowanie manifestu portworx-operator

W panelu Portworx należy przejść do zakładki "Install and Run" i kontynuować klikając przycisk "Next" aż do ostatniego kroku. W sekcji "Portworx Operator" należy kliknąć na ikonę oka obok manifestu operatora, skopiować manifest Portworx Operatora, przejść do katalogu /portworx/base/ w projekcie i utworzyć plik portworx-operator.yml. Następnie należy wkleić skopiowany manifest PortworxOperatora do nowego pliku.

Po wykonaniu tych kroków, manifest Portworx Operatora będzie dostępny w lokalnym repozytorium, gotowy do użycia w konfiguracji klastra Kubernetes. Trzeba sprawdzić poprawności skopiowanego manifestu oraz zastosowaniu go zgodnie z wymaganiami aplikacji.

12. Przygotowanie manifestu StorageClass

Aby uzyskać dostęp do manifestu StorageClass dla Portworx, należy uruchomić poniższą komendę w terminalu, aby otrzymać YAMLmanifestu StorageClass.

```
kubectl get storageclass px-storageclass -oyaml
```

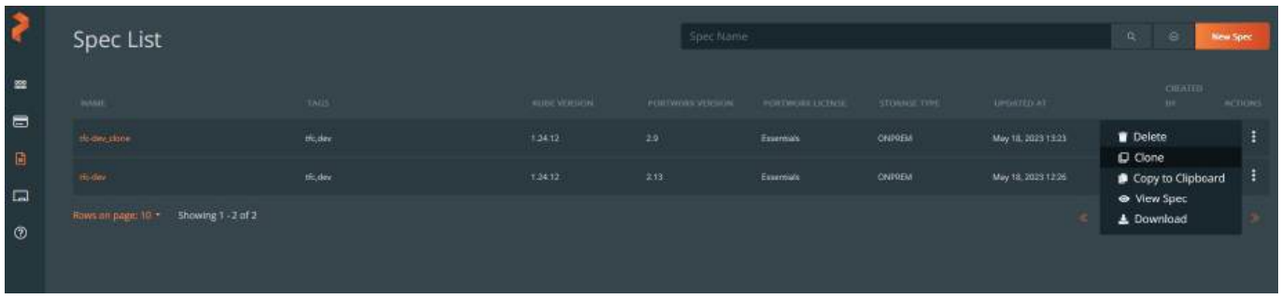
W wyniku tej operacji manifest zostanie wyświetlony w konsoli. Alternatywnie, można bezpośrednio zapisać manifest do pliku, wykonując komendę:

```
kubectl get storageclass px-storageclass -oyaml > portworx-storageclass.yml
```

Ten krok zapisze manifest StorageClass do pliku o nazwie portworx-storageclass.yml.
Po wykonaniu tych kroków, będzie dostępny manifest StorageClass dla Portworx, który można edytować lub użyć w innych procesach konfiguracyjnych. Niezbędne jest monitorowanie i weryfikacja zawartości manifestu przed jego zastosowaniem.

13. Przygotowanie manifestów

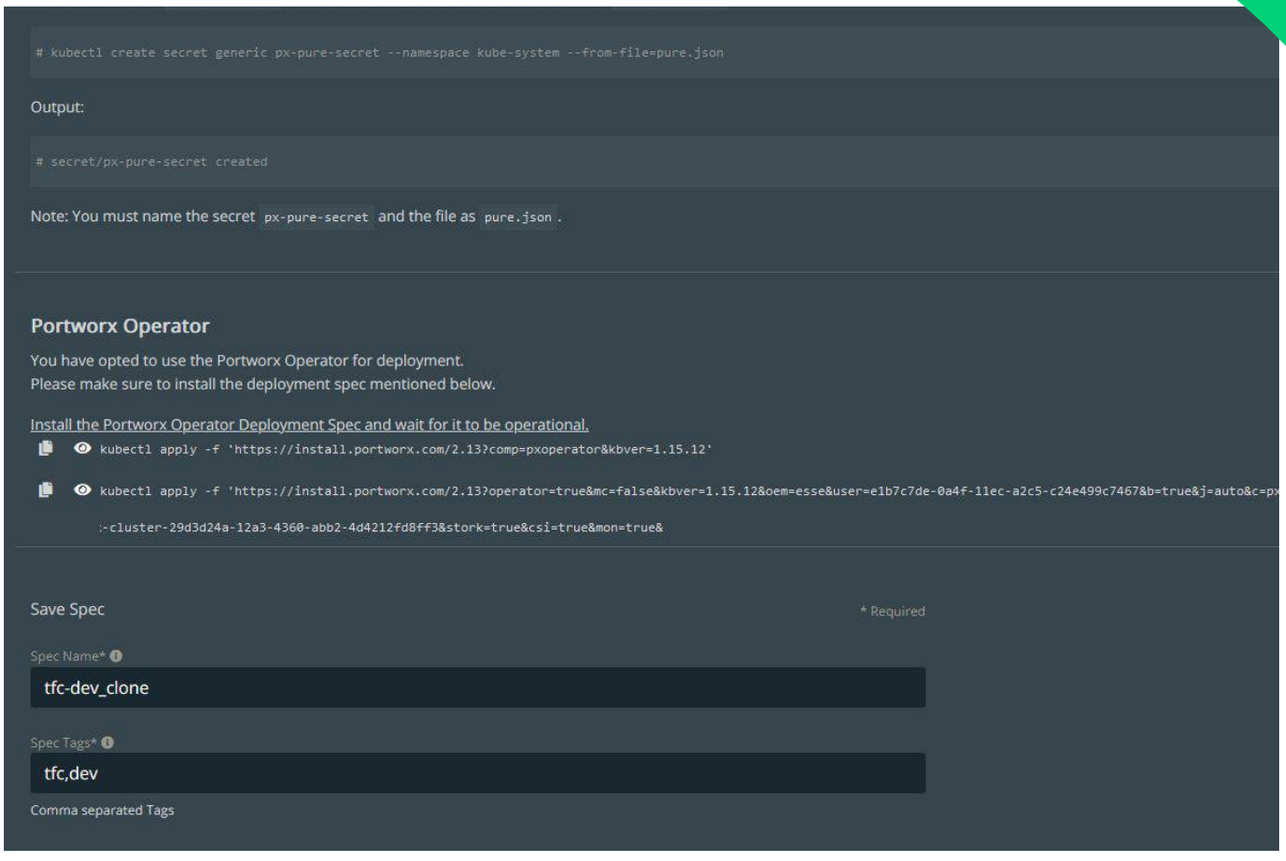
Aby zaktualizować manifesty za pomocą panelu Portworx, należy zalogować się do platformy i wybrać aktualną specyfikację w zakładce "Install and Run". Następnie sklonować ją, definiując nowe wersje dla Portworxa i klastra.



Konieczne jest zdefiniowane nowej wersji, zarówno Portworxa, jak i samego klastra. W dalszych krokach konieczne jest dostosowanie specyfikacji do naszych potrzeb.

Wynikiem końcowym jest znany widok, z którego można pobrać wszystkie potrzebne nam manifesty.

13. Przygotowanie manifestów



Po sklonowaniu, należy pobrać manifesty, a następnie umieścić manifest StorageCluster w dedykowanej ścieżce dla swojego środowiska, np./portworx/overlays/dev/. Na ostatnim ekranie podsumowującym znajduje się informacja dotyczącą utworzenia obiektu typu Secret.

W przypadku posiadania odpowiedniego obiektu Secret, ponowne jego tworzenie nie jest konieczne. Dzięki tym krokom można zaktualizować manifesty Portworx, dostosowując je do bieżących potrzeb z zachowaniem spójność konfiguracji klastra Kubernetes.

14. Modyfikacja manifestów

W tym etapie niezapomniane jest również przeprowadzenie weryfikacji oraz ewentualne dostosowanie zmian. Dane te są kluczowe, a w przypadku potrzeby, należy pamiętać o ich podmianie. Dodatkowo, konieczne jest utworzenie i dodanie odpowiedniego obiektu Secret, o którym wcześniej wspomniano.

Aby to zrealizować, można skopiować obiekt Secret bezpośrednio z klastra za pomocą polecenia:

```
kubectl get secret -n kube-system px-essential -oyaml
```

Następnie należy dodać skopiowane dane do pliku w dedykowanej ścieżce, na przykład /portworx/overlays/px-essential-secret.env. Warto pamiętać, aby nie umieszczać Secret w takiej formie w systemie kontroli wersji, takim jak GIT. To dotyczy wszystkich obiektów Secret, nie tylko tego konkretnego. Jeśli konieczne jest dodanie do repozytorium, zaleca się zaszyfrowanie go za pomocą klucza GPG.

15. Aktualizacja Portworx

Przed przystąpieniem do aktualizacji, konieczne jest przygotowanie plików kustomization.yml, które połączą wszystkie wcześniej utworzone manifesty. Każdy z folderów (/base/, /overlays/dev/, /overlays/prod) musi posiadać swój plik /portworx/base/kustomization.yml.

Folder /base/ nomen omen wykorzystywany jest jako baza. Manifesty zadeklarowane w kustomization.yml tego folderu są bazą dla manifestów z folderu /overlays/.

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
# Zdefiniowany dodany przez nas wcześniej portworx-operator oraz storageclass
  - portworx-operator.yml
  - portworx-storageclass.yml
```

/portworx/overlays/kustomization.yml

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

generatorOptions:
  disableNameSuffixHash: true

resources:
# Zdefiniowany folder /base/, o którym pisałem wcześniej
  - ../../base
# Zdefiniowany storage-clister który wygenerowaliśmy w poprzednich krokach
  - storage-cluster.yml

# W tej sekcji dodawany jest secret. W tym konretnym przypadku konieczne jest by secret był już zaszyfrowany przez klucz GPG
secretGenerator:
  - name: px-essential
    namespace: kube-system
    envs:
      - px-essential-secret.env
```

Po wykonaniu tych operacji można porównać manifest z klastrem. Porównanie to możemy dokonać poprzez ArgoCD po wcześniejszym przesłaniu zmian do repozytorium GIT. Zaleca się unikanie opcji Sync i ograniczanie się do korzystania jedynie z opcji Refresh oraz App Diff.
Jeśli ArgoCD nie wykazuje błędów lub zmian, które wpływają na działanie klastra, możemy przystąpić do synchronizacji aplikacji ArgoCD. Zmiany możemy obserwować po stronie klastra za pomocą polecenia:

```
watch kubectl get pods -o wide -n kube-system
```

Po ustabilizowaniu się stanu klastra, aktualizacja została zakończona.

16. Prometheus Stack

Aktualizacja prometheus-stack w klastrze może być jednym z bardziej czasochłonnych i wymagających procesów. Jest to głównie związane z kompleksowością samego prometheus-stack, który składa się z wielu komponentów, a także koniecznością jego odtworzenia poprzez ArgoCD od podstaw.

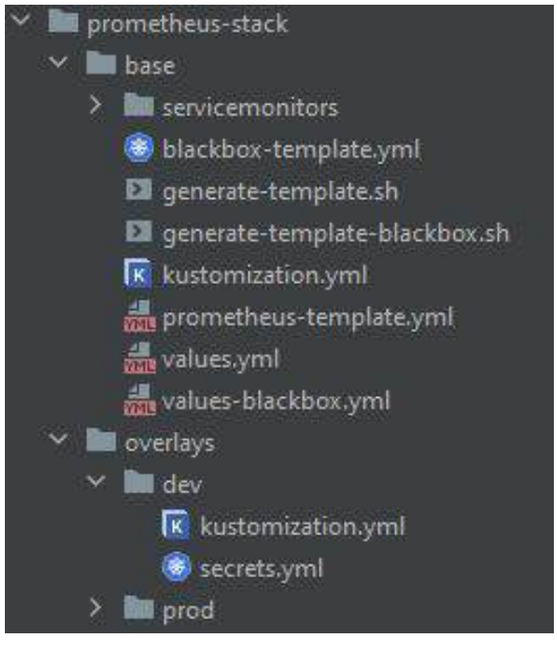
Mimo konieczności poświęcenia wielu godzin na wdrożenie prometheus-stack przez ArgoCD, jest to decyzja uzasadniona i wręcz niezbędna, szczególnie w kontekście długoterminowego rozwoju klastra oraz przyszłych aktualizacji.

Działania wstępne

Przed rozpoczęciem aktualizacji należy pamiętać o kilku kluczowych krokach. Można upewnić się, że stworzona aplikacja w ArgoCD nosi nazwę "monitoring", a definicja tej aplikacji znajduje się wpliku monitoring.yml dostępnym w repozytorium.

Warto również zapewnić odpowiednią strukturę plików zgodną z wymaganiami dla klastra oraz upewnić się, że wszystkie niezbędne plikii konfiguracje są dostępne przed przystąpieniem do aktualizacji.

Dokładając staranności na etapie przygotowań, można zminimalizować trudności i zapewnić płynne przejście przez proces aktualizacji prometheus stack w klastrze.



kzadlo@core-logic.com
www.core-logic.com

17. Przygotowanie skryptu oraz pliku values.yml

W pierwszym etapie należy przygotować skryptu łatwiający generację manifestu Prometheus-Stack, wykorzystując narzędzie Helm.

Przed generacją pierwszego manifestu warto zdefiniować wersję, która obecnie jest używana w klastrze. Dzięki temu otrzymuje się "czysty" manifest dla aktualnej wersji. Ze względu na złożoność i rozmiar prometheus-stack, korzystne (a czasem nawet konieczne) jest wykorzystanie pliku values.yml.

Poniżej znajduje się polecenie Helm, które generuje manifest, biorąc pod uwagę wartości z pliku values.yml oraz wersję docelową ({{target-version}}):

```
helm template -f values.yml --include-crds=true --namespace=monitoring --version={{target-version}} monitoring
prometheus-community/kube-prometheus-stack  > prometheus-template.yml
```

Przed generacją warto zapoznać się z dokumentacją pliku values.yml. Plik ten pozwoli dostosować manifest do specyfiki klastra i działających na nim usług, poprzez predefiniowanie części zmiennych. Dzięki temu proces generowania manifestu staje się bardziej elastyczny i dostosowany do indywidualnych potrzeb oraz specyfiki konkretnego klastra.

18. Przygotowanie punktu odzyskiwania

Z uwagi na duży rozmiar manifestu prometheus-stack, efektywnym podejściem jest lokalne porównywanie zmian. Realizacja tego zadania po stronie ArgoCD staje się trudna z powodu ilości danych, które trzeba przetworzyć i wyświetlić w przeglądarce.

W przypadku klastra, należy postępować inaczej, wykonując lokalne porównywanie zmian. Jednak wcześniej warto utworzyć punkt odzyskiwania, do którego porównamy nowo utworzone manifesty. Obejmuje to manualne przygotowanie lub pobranie manifestu prometheus-stack z klastra. Zadanie to warto przygotować od surowego manifestu dla aktualnej wersji na klastrze. Warto wykorzystać wcześniej przygotowany skrypt lub polecenie do generacji template. Jednakże, trzeba wykonać to polecenie z opcją --include-crds=false, co znacznie zmniejszy rozmiar pliku.

Teraz można przejść do dalszych kroków przygotowania punktu odzyskiwania i lokalnego porównywania zmian w manifestach dla prometheus-stack.

Przygotowanie surowego manifestu dla aktualnej wersji dostępnej na klastrze

Przygotowanie surowego manifestu dla aktualnej wersji dostępnej na klastrze może być zrealizowane za pomocą wcześniej przygotowanego skryptu lub polecenia do generacji template. Jednak wskazane jest wykonać to polecenie z opcją --include-crds=false, co znacząco zmniejszy rozmiar pliku.

Przygotowanie backupu obiektów na podstawie surowego manifestu

Gdy jest dostępny surowy manifest, warto przystąpić do tworzenia backupu. Zadanie to polega na stworzeniu pliku zawierającego te same obiekty, co surowy manifest. Innymi słowy, należy utworzyć kopię surowego manifestu, pobierając obiekty bezpośrednio z klastra.

18. Przygotowanie punktu odzyskiwania

Surowy manifest może zawierać różne obiekty, na przykład Service, który może wyglądać następująco:

```yaml
# Source: kube-prometheus-stack/templates/exporters/kube-controller-manager/service.yaml
apiVersion: v1
kind: Service
metadata:
  name: monitoring-kube-prometheus-kube-controller-manager
  labels:
    app: kube-prometheus-stack-kube-controller-manager
    jobLabel: kube-controller-manager

    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/instance: monitoring
    app.kubernetes.io/version: "28.0.0"
    app.kubernetes.io/part-of: kube-prometheus-stack
    chart: kube-prometheus-stack-28.0.0
    release: "monitoring"
    heritage: "Helm"
  namespace: kube-system
spec:
  clusterIP: None
  ports:
    - name: http-metrics
      port: 10252
      protocol: TCP
      targetPort: 10252
  selector:
    component: kube-controller-manager
  type: ClusterIP
```


kzadlo@core-logic.com
www.core-logic.com

19. Surowy manifest dla aktualnej wersji dostępnej na klastrze

Stworzenie backupu polega na skopiowaniu obiektów tego typu z klastra i zapisaniu ich w pliku, który będzie stanowił punkt odzyskiwania w przypadku potrzeby przywrócenia konkretnych konfiguracji.

Wskazane jest pobranie manifestu obiektu Service o nazwie monitoring-kube-prometheus-kube-controller-manager z klastra za pomocą polecenia:

```
kubectl get svc -n kube-system -o yaml monitoring-kube-prometheus-kube-controller-manager >> prometheus-templ
ate-old.yml && echo "---" >> prometheus-template-old.yml
```

Zaznaczone sekcje można sukcesywnie usuwać. Będą jedynie przeszkadzać w procesie porównywania plików

19. Surowy manifest dla aktualnej wersji dostępnej na klastrze

```yaml
apiVersion: v1
kind: Service
metadata:
# Sekcja annotatnios może zostać usunięta, będzie później przeszkadzać przy porównaniach
#  annotations:
#    kubectl.kubernetes.io/last-applied-configuration: |
#      {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"labels":{"app":"kube-prometheus-stack-kube-controller-manager","app.kubernet
        stack-kube-controller-manager","app.kubernetes.io/instance":"monitoring","app.kubernetes.io/managed-by":"Helm","app.kubernetes.io/part-of":"kube-
#    meta.helm.sh/release-name: monitoring
#    meta.helm.sh/release-namespace: monitoring
# creationTimestamp również może zostać usunięte
#  creationTimestamp: "2021-09-22T13:56:12Z"
  labels:
    app: kube-prometheus-stack-kube-controller-manager
    app.kubernetes.io/instance: monitoring
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/part-of: kube-prometheus-stack
    app.kubernetes.io/version: 28.0.0
    chart: kube-prometheus-stack-28.0.0
    heritage: Helm
    jobLabel: kube-controller-manager
    release: monitoring
  name: monitoring-kube-prometheus-kube-controller-manager
  namespace: kube-system
# resourceVersion oraz uid może zostać usunięte
#  resourceVersion: "522965913"
#  uid: e787cbf2-4c49-4282-9d12-d8276fdb636a
spec:
  clusterIP: None
  clusterIPs:
  - None
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http-metrics
    port: 10252
    protocol: TCP
    targetPort: 10252
  selector:
    component: kube-controller-manager
  sessionAffinity: None
  type: ClusterIP
# Sekcja status może zostać usunięta
#status:
#  loadBalancer: {}
```

Należy powtórzyć tę czynność dla każdego obiektu w surowym manifeście, zgodnie z ich kolejnością. Każdy obiekt z manifestu powinien być skopiowany z klastra i zapisany w odpowiednim pliku, tworząc tym samym punkt odzyskiwania dla konkretnych konfiguracji.

Po skopiowaniu wszystkich obiektów, trzeba przystąpić do całościowego porównania plików. Przydatnym rozwiązaniem ułatwiającym pracę jest porównywanie obiektów natychmiast po ich zapisaniu do pliku prometheus-stack-old.yml. Dzięki temu jest możliwość dokonywania zmian i poprawek w pliku values.yaml na bieżąco.

20. Przyrównanie ArgoCD do klastra

Podobnie, jak w przypadku poprzednich aplikacji, ten krok jest konieczny tylko wtedy, gdy prometheus-stack nie jest dostępny po stronie ArgoCD. W takiej sytuacji konieczne jest przyrównanie aplikacji do obecnego stanu w klastrze.

Dodatkowe definicje - patch.yml oraz secrets.yml
Niestety, nie wszystkie zmienne można zdefiniować w opisywanym wcześniej pliku values.yaml. Część ustawień nie znajduje tam swojego odpowiednika, co może wymagać alternatywnego podejścia do ich zdefiniowania.

W tej sytuacji sięgamy po narzędzie kustomize, które pozwala na elastyczne dodawanie dodatkowych konfiguracji, a nawet całych manifestów.

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - ../../base

patches:
  - path: secrets.yml

patchesStrategicMerge:
  - patch.yml
```

20. Przyrównanie ArgoCD do klastra

Mechanizm patchesStrategicMerge pozwala elastycznie nadpisywać tylko te zmienne lub konfiguracje, które są interesujące, pozostawiając resztę nienaruszoną.
Dzięki temu są aktualizowane wcześniejsze wpisy, zamiast całkowicie je zastępować.

Wpis w surowym manifeście:

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name:  monitoring-kube-prometheus-admission
  labels:
    app: kube-prometheus-stack-admission
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/instance: monitoring
    app.kubernetes.io/version: "28.0.0"
    app.kubernetes.io/part-of: kube-prometheus-stack
    chart: kube-prometheus-stack-28.0.0
    release: "monitoring"
    heritage: "Helm"
webhooks:
  - name: prometheusrulemutate.monitoring.coreos.com
    failurePolicy: Ignore
    rules:
      - apiGroups:
          - monitoring.coreos.com
        apiVersions:
          - "*"
        resources:
          - prometheusrules
        operations:
          - CREATE
          - UPDATE
    clientConfig:
      service:
        namespace: monitoring
        name: monitoring-kube-prometheus-operator
        path: /admission-prometheusrules/validate
    admissionReviewVersions: ["v1", "v1beta1"]
    sideEffects: None
```

20. Przyrównanie ArgoCD do klastra

Zawartość patch.yml

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: monitoring-kube-prometheus-admission
webhooks:
  - admissionReviewVersions:
      - v1
      - v1beta1
    failurePolicy: Fail
    matchPolicy: Equivalent
    name: prometheusrulemutate.monitoring.coreos.com
    rules:
      - apiGroups:
          - monitoring.coreos.com
        apiVersions:
          - '*'
        operations:
          - CREATE
          - UPDATE
        resources:
          - prometheusrules
        scope: '*'
    timeoutSeconds: 10
```

Wynik kustomize build:

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  labels:
    app: kube-prometheus-stack-admission
    app.kubernetes.io/instance: monitoring
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/part-of: kube-prometheus-stack
    app.kubernetes.io/version: 28.0.0
    chart: kube-prometheus-stack-28.0.0
    heritage: Helm
    release: monitoring
  name: monitoring-kube-prometheus-admission
webhooks:
- admissionReviewVersions:
  - v1
  - v1beta1
  clientConfig:
    service:
      name: monitoring-kube-prometheus-operator
      namespace: monitoring
      path: /admission-prometheusrules/validate
# Zmiana z Ignore na Failure
  failurePolicy: Fail
  matchPolicy: Equivalent
  name: prometheusrulemutate.monitoring.coreos.com
  rules:
  - apiGroups:
    - monitoring.coreos.com
    apiVersions:
    - '*'
    operations:
    - CREATE
    - UPDATE
    resources:
    - prometheusrules
# Dodano scope
    scope: '*'
  sideEffects: None
# Dodano timeoutSeconds
  timeoutSeconds: 10
```

Dodatkowo, konieczne jest dodanie obiektu typu Secret. Zadanie to należy zrealizować podobnie jak w przypadku "PortworxSecret".

21. Aktualizacja Prometheus Stack

Proces aktualizacji przebiega analogicznie, jak do poprzednich usług.
Aby zaktualizować, należy wygenerować manifest dla najnowszej wersji, uwzględniając pliki values.yml, patch.yml oraz secrets.yml.

Polecenie do wygenerowania manifestu prezentuje się następująco:

```
helm template -f values.yml --include-crds=true --namespace=monitoring --version=28.0.0
monitoring prometheus-community/kube-prometheus-stack  > prometheus-template.yml
```

W trakcie generowania manifestu dla nowej wersji, ważne jest zmienienie przełącznika --include-crds na true oraz rozdzielenie manifestów, zgodnie z dotychczasową praktyką.

Następnie przystąpiono do porównania zmian za pomocą narzędzia App Diff w ArgoCD. To pozwala na wyłapanie nawet drobnych zmian, które mogły zostać pominięte wcześniej. Po zakończeniu tego procesu i upewnieniu się, że nie ma kluczowych różnic ani błędów, należy przystąpić do synchronizacji aplikacji w ArgoCD.

Jeśli App Diff nie wykrywa istotnych różnic, można bezpiecznie kontynuować i rozpocząć synchronizację aplikacji, zapewniając spójność i zgodność konfiguracji z oczekiwaniami i wymaganiami dotyczącymi klastra.

22. Blackbox Exporter

W przypadku klastra, podniesienie wersji BlackboxExporter okazało się niezbędne dla zachowania stabilności i efektywności usług monitorujących. Proces aktualizacji Blackbox Exportera nie różni się znacząco od standardowej procedury aktualizacji aplikacji, jednak istotną różnicą jest fakt, że nie było konieczności budowy osobnej aplikacji w celu implementacji zmian.

W kontekście architektury monitoringu, BlackboxExporter pełni kluczową rolę, umożliwiając zbieranie i analizę danych dotyczących stanu oraz dostępności usług. Z tego powodu warto skorzystać z istniejącej aplikacji do monitoringu, która została zaprojektowana i dostosowana do potrzeb aktualizacji całego wachlarza narzędzi monitorujących, w tym również prometheus-stack.

Dzięki wykorzystaniu aplikacji do monitoringu, która została już skonfigurowana i zoptymalizowana pod kątem wymagań, proces aktualizacji Blackbox Exportera mógł być przeprowadzony sprawnie i zminimalizowano ryzyko wystąpienia ewentualnych komplikacji czy niezgodności w środowisku monitorującym. W ten sposób zapewniono ciągłość działania monitoringu oraz wysoką jakość usług dostarczanych przez klaster.

Przygotowanie manifestu

W celu przygotowania manifestu dla aktualizacji Blackbox Exportera, warto oprzeć się na pliku values.yaml oraz wykorzystaniu narzędzia Helm.

Aby wygenerować gotowy, pełny manifest, należy zastosować polecenie:

```
helm template -f values-blackbox.yml --include-crds=true --namespace=monitoring --version={{target-version}} blackbox-exporter prome
theus-blackbox-exporter > blackbox-template.yml
```

W powyższym poleceniu {{target-version}} odnosi się do aktualnej wersji Blackbox Exportera na klastrze, korzystając z pliku values-blackbox.yml.

Dzięki tej operacji, zgodnie z wymaganiami, osiągnięto gotowość do wdrożenia manifestu, który uwzględnia specyfikacje środowiska monitorującego i zapewnia zgodność z wymaganiami klastra.

23. Przygotowanie ArgoCD do klastra

Podobnie, jak w poprzednich przypadkach, konieczne jest dokładne dostosowanie pliku do wymagań i konfiguracji klastra. W celu identyfikacji niezbędnych zmian, ponownie warto skorzystać z narzędzia AppDiff dostępnego w ArgoCD, które umożliwia porównanie różnic między repozytorium, a aktualnym stanem klastra.

Celem jest porównanie manifestu ArgoCD z rzeczywistymstanem klastra, aby upewnić się, że konfiguracje są zgodne. Procedura ta nie różni się znacząco od wcześniejszych działań. Po zidentyfikowaniu różnic i wprowadzeniu niezbędnych zmian, nowo utworzony manifest należy dodać do pliku /prometheus-stack/base/kustomization.yml.

Ostateczna struktura pliku w folderze głównym prezentuje się następująco:

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - prometheus-template.yml
  - blackbox-template.yml
```

Po przeprowadzeniu porównania repozytorium z aktualnym stanem środowiska, możemy przystąpić do synchronizacji aplikacji za pośrednictwem ArgoCD. Dzięki temu uzyskano punkt odzyskiwania oraz zapewniono spójność i zgodność konfiguracji aplikacji z wymaganiami i oczekiwaniami dotyczącymi klastra.

24. Kubernetes - aktualizacja Nodów Control-Plane

Po zaktualizowaniu wszystkich komponentów, należy przystąpić do aktualizacji Kubernetesa. Pierwszym krokiem jest aktualizacja węzłów kontrolnych.

Aktualizacja narzędzia kubeadm

Aby przeprowadzić aktualizację, najpierw należy ustalić odpowiednią wersję, którą chcemy zainstalować. Następnie, po wybraniu odpowiedniej wersji, można zainstalować ją poprzez:

```
apt-mark unhold kubeadm && \
apt-get update && apt-get install -y kubeadm=1.25.14-00 && \
apt-mark hold kubeadm
```

Po zainstalowaniu odpowiedniej wersji narzędzia kubeadm, należy przygotować plan aktualizacji poprzez wykonanie:

```
kubeadm upgrade plan
```

25. Aktualizacja klastra Kubernetes

Następnie, trzeba wykonać aktualizację klastra Kubernetesa poprzez:

```
sudo kubeadm upgrade node
```

Po aktualizacji narzędzia kubeadm, trzeba zaktualizować również kubelet oraz kubectl. Przed aktualizacją warto wyciągnąćich blokadę, a po aktualizacji - ponownie ją nałożyć. Procedurę uaktualnienia kubelet i kubectl można przeprowadzić za pomocą następujących kroków:

Po wyznaczeniu węzła do aktualizacji, najpierw wyeliminowano go z zadań i zapewniono poprawność informacji.

```
kubectl drain <node-to-drain> --ignore-daemonsets
```

Następnie zaktualizowano kubelet oraz kubectl.

```
apt-mark unhold kubelet kubectl && \
apt-get update && apt-get install -y kubelet=1.25.14-00 kubectl=1.25.14-00 && \
apt-mark hold kubelet kubectl
```

Po zainstalowaniu nowych wersji, trzeba zrestartować usługę kubelet:

```
sudo systemctl daemon-reload
sudo systemctl restart kubelet
```

Na koniec, warto upewnić się, że węzeł został ponownie udostępniony do pracy poprzez wykonanie poniższego polecenia. Dodatkowo trzeba uzyskać zapewnienie, że <node-to-drain> to nazwa węzła, który został wcześniej zaktualizowany.

```
kubectl uncordon <node-to-drain>
```

Nody worker

Aktualizacja węzłów roboczych odbywa się za pomocą playbooka Ansible,
który został podzielony na dwa osobne pliki:

- update-node-k8s.yml- odpowiedzialny za aktualizację Kubernetesa.
- update-node-os.yml- odpowiedzialny za aktualizację systemu operacyjnego oraz restart serwera.

Należy pamiętać, że operacje należy wykonywać pojedynczo, węzeł po węźle, aby uniknąć przerw w dostępności usług.

Zakończenie

To już koniec przewodnika po aktualizacji klastra Kubernetes z wersji 1.25 do wersji 1.26. W najnowszej wersji klastra Kubernetes, dokonano przejścia z mechanizmu Pod Security Policy (PSP) na bardziej elastyczną i nowoczesną metodę, zwaną Pod Security Admission (PSA). To istotna zmiana, która wymaga starannego przygotowania i ścisłego przestrzegania procedur podczas implementacji i aktualizacji narzędzi w klastrze. Artykuł omawiał kluczowe kroki przed aktualizacją klastra, skupiając się na usługach, które wcześniej korzystały z PSP.

Podsumowując:

Raport stanowi kompleksowe omówienie kluczowych aspektów związanych z aktualizacją klastra Kubernetes, podkreślając konieczność uwzględnienia zmian w mechanizmie zabezpieczeń oraz przestrzegania procedur aktualizacji dla każdego z komponentów klastra.

Konrad Matyas

Wiceprezes Core Logic

core • logic

Konrad Matyas
Wiceprezes Core Logic

Napisz do nas

Podzielimy się naszą wiedzą i doświadczeniem,
aby pomóc Ci rozwiązać twoje problemy
i stworzyć produkt, który zawsze chciałeś.

Web:
www.core-logic.com

Feliksa Radwańskiego 15/1,
30-065 Kraków, Polska

Nr Telefonu & Whatsapp:
+48 606 524 052

E-mail:
kzadlo@core-logic.com