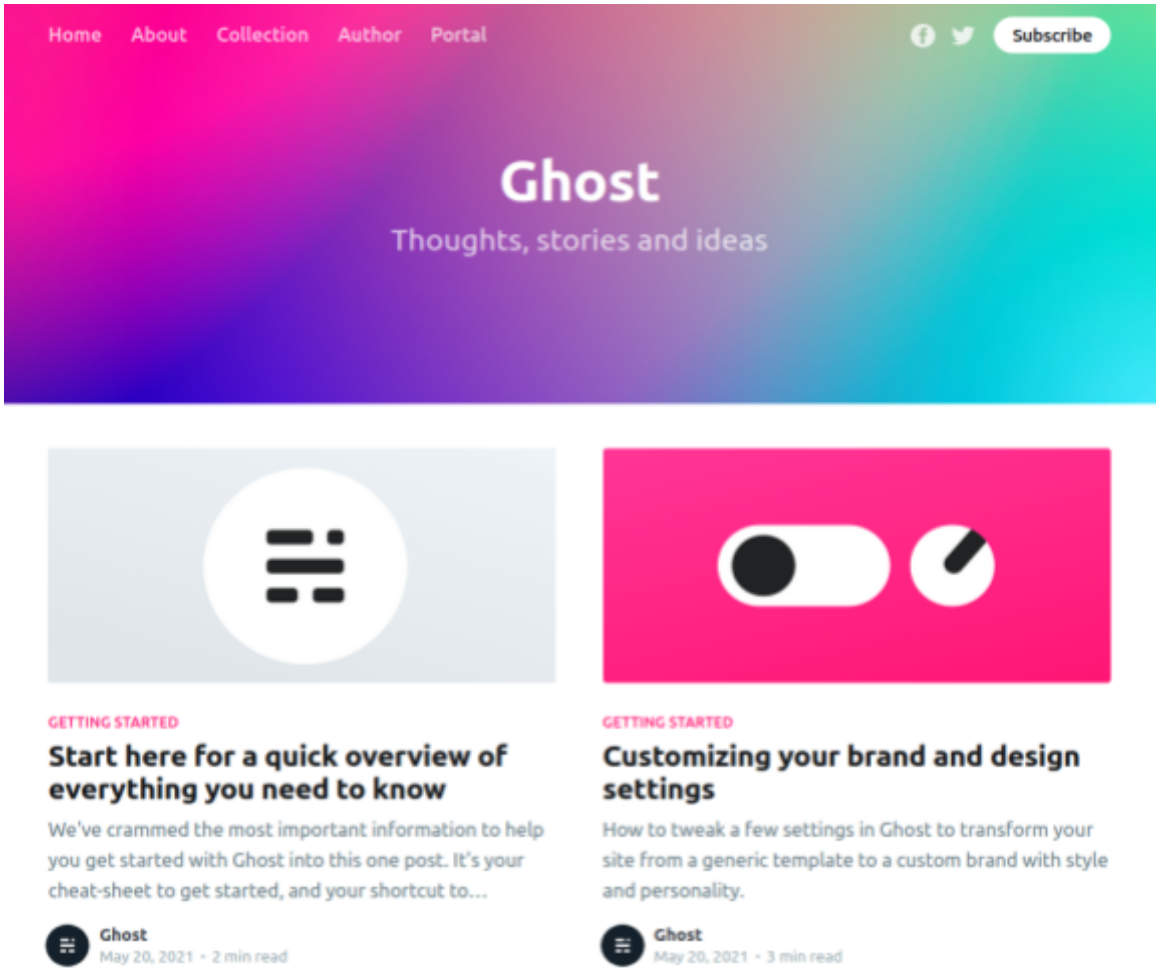
Aplikacje i usługi internetu

Kompozycja skalowalnej usługi w środowisku kontenerowym (Kubernetes, Helm)

Młyniec Paweł, Górski Michał, Wrzesień Wojciech
Prowadzący: mgr inż. Maciej Sosnowski

1. Wstęp

W ramach projektu zrealizowano przykładowego bloga (rysunek 1) z użyciem systemu zarządzania treścią Ghost. W realizacji aplikacji wykorzystano system orkiestracji Kubernetes oraz jego menadżer pakietów Helm. W celu monitoringu systemu skorzystano z aplikacji internetowej Grafana, do której metryki były zbierane za pomocą Prometheusa.



Rysunek 1. Widok głównej strony blogu.

Kod źródłowy wraz z krótkim opisem użycia dostępny jest na repozytorium..
Całość prac można podzielić na kilka etapów:

- Uruchomienie lokalnego klastra minikube
- Deployment blogu
- Uruchomienie narzędzia monitorującego Prometheus
- Test autoskalowania

2. Klaster minikube

Pierwszym etapem jest instalacja Virtual Box'a, na którym powstanie nasz klaster minikube'a. W przypadku kiedy chcemy sprecyzować na jakiej maszynie wirtualnej ma powstać klaster dodajemy opcję --vm-driver :

```
minikube start --vm-driver=virtualbox
```

Jeśli nie dodamy tej opcji klaster powstanie na pierwszej maszynie z dostępnych. Przy pierwszym uruchomieniu potrzebne będzie włączenie dodatków: ingress, metric-server. Robimy to poniższymi komendami:

```
minikube addons enable ingress
minikube addons enable metrics-server
```

Po pierwszym włączeniu opcje te już będą włączone na danym klastrze przy ponownym uruchomieniu.
Kolejnym krokiem jest sprawdzenie adresu ip naszego klastra minikube:

```
minikube ip
```

Uzyskane IP minikube'a oraz zdefiniowaną w pliku "values.yaml" nazwę domeny należy zapisać w pliku hosts, aby na domenę można było się dostać po zdefiniowanej nazwie domeny "ghost.info". Można to wykonać komendą (należy podać, aby podać tu adres ip zwrócony w pierwszej komendzie).

```
echo '192.168.99.101 ghost.info' | sudo tee -a /etc/hosts
```

W tej chwili nasz klaster jest przygotowany aby dokonać deploymentu wybranej aplikacji.

3. Deployment blogu

Aby dokonać deploymentu należy najpierw skopiować z GitHub'a repozytorium. Można to zrobić komendą:

```
git clone git@github.com:wwrzesien/ghost-helm.git
```

Po przejściu do folderu z skopiowanym repozytorium dokonujemy deploymentu poniższą komendą.

```
helm install ghost-blog ghost-blog
```

Po tej operacji aplikacja powinnna być dostępna pod adresem http://ghost.info . Można się na niego dostać również przez wpisanie w okno przeglądarki adresu ip minikube'a oraz portu 2368.

4. Uruchomienie Prometheus'a

Prometheus jest systemem monitorowania serwisów i usług. Pozwala na wyświetlenie stanu uruchomionych usług w miłej dla oka szacie graficznej wykresów. Jest to dużo przyjemniejsze niż korzystanie z dashboardu minikube'a. W naszym przypadku będziemy obserwować zachowanie klastra podczas testu autoskalowania.

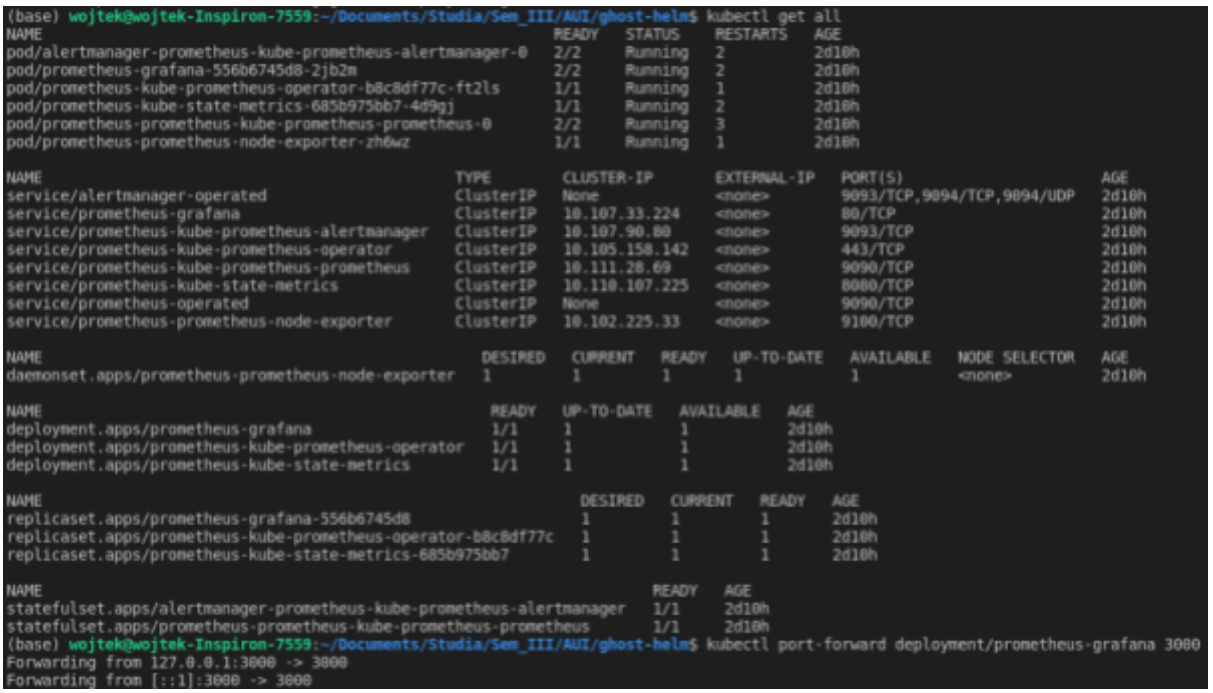
Pierwszym krokiem jest instalacja prometheus-operator:

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo add stable https://kubernetes-charts.storage.googleapis.com/
helm repo update
```

Następnie należy w oddzielnej przestrzeni uruchomić Prometheus'a oraz otworzyć pod na zewnątrz poprzez przekierowanie portu na 3000.

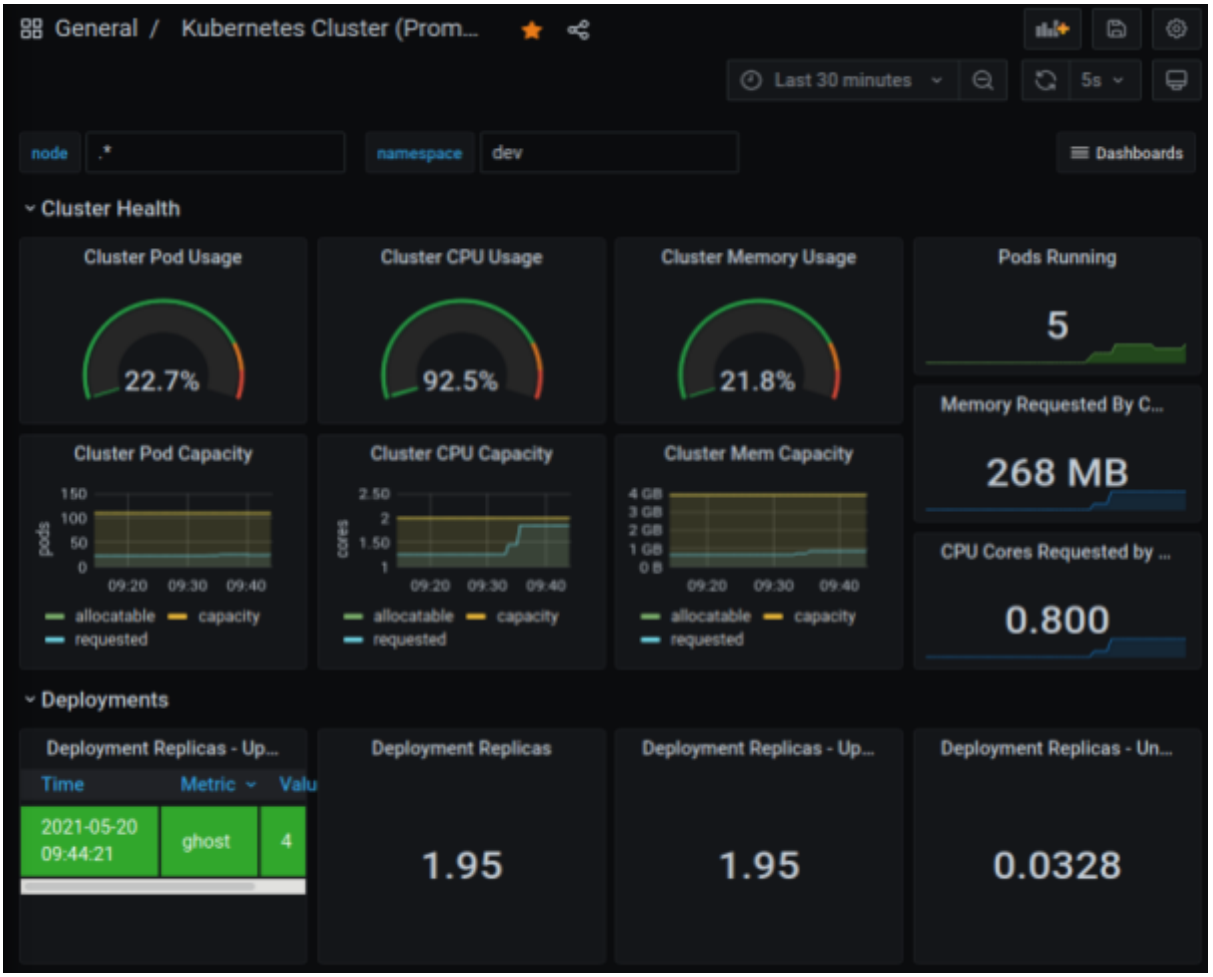
```
kubectl create namespace prometheus
helm install prometheus prometheus-community/kube-prometheus-stack
kubectl port-forward deployment/prometheus-grafana 3000
```

Na rysunku 2 zaprezentowano wyniki dwóch pierwszych poleceń w przestrzeni prometheus.



Rysunek 2. Zasoby działające na klastrze, przekierowanie Grafany.

W obecnej chwili Prometheus jest dostępny pod adresem http://localhost:3000. Należy podać login: "admin" oraz hasło: "prom-operator". W widocznym dashboardzie po prawej stronie wybieramy ikonę "plus" a z niej opcję "Import" i wskazujemy link do dashboardu, który chcemy zaimportować - https://grafana.com/grafana/dashboards/6417. Po potwierdzeniu operacji powinniśmy ujrzeć dashboard przypominający ten na rysunku 3.



Rysunek 3. Grafana dashboard.

5. Test autoskalowania

W pliku “hpa.yaml” są ustawione opcje autoskalowania. Zakładamy że usługa może mieć od 1 do 10 replik. Przy czym nowa replika powstaje przy obciążeniu CPU przekraczającym 50%.

Aby uruchomić test musimy uruchomić generator obciążenia komendą:

```
kubectl run -i --tty load-generator --rm --image=busybox --restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://ghost.info; done"
```

W możemy monitorować status usługi poprzez uruchomienie komend:

```
kubectl get hpa -w
kubectl get deployments -w
```

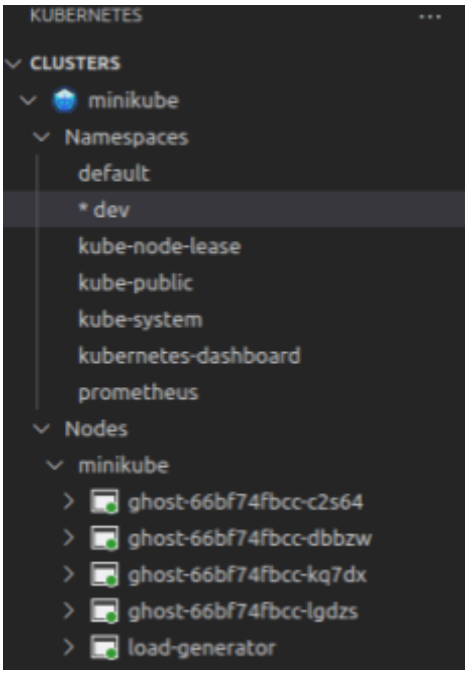
Na rysunku 4 pokazano przykładowy widok użycia pierwszej komendy. Widać jak ze wzrostem obciążenia przy osiągnięciu progu 50% uruchamiane są kolejne repliki serwisu “ghost”.

```
(base) wojtek@wojtek-Inspiron-7559:~/Documents/Studia/Sem_III/AUI/ghost-helm$ kubectl get hpa -w
NAME    REFERENCE          TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
ghost   Deployment/ghost   0%/50%    1         10        1          2d11h
ghost   Deployment/ghost   109%/50%  1         10        1          2d11h
ghost   Deployment/ghost   109%/50%  1         10        3          2d11h
ghost   Deployment/ghost   84%/50%   1         10        3          2d11h
ghost   Deployment/ghost   61%/50%   1         10        3          2d11h
ghost   Deployment/ghost   61%/50%   1         10        4          2d11h
ghost   Deployment/ghost   46%/50%   1         10        4          2d11h
```

Rysunek 4. Zmiana liczby replik serwisu “ghost” podczas testu obciążenia.

6. Podsumowanie

Stworzony blog spełnił swoje zadanie - wraz ze wzrostem obciążenia odpalane są kolejne repliki w celu odciążenia strony. Dzięki zastosowaniu menadżera pakietów Kubernetes’a (Helm), przy użyciu pojedynczej komendy, możliwe jest szybkie oraz łatwe zarządzanie wersją na serwerze (m.in. deployment nowej wersji, deployment nowej aplikacji, powrót do poprzedniej wersji aplikacji). Zastosowanie serwisów PersistentVolume i PersistentClaim umożliwia zdefiniowanie ilości pamięci dostępnej dla aplikacji oraz zapewnia zachowanie plików w przypadku awarii podów. Przydatne okazało się rozszerzenie “Kubernetes” w VSCode. Pozwoliło ono na szybkie monitorowanie klastra i wykonywanie operacji z poziomu GUI bez konieczności znajomości komend konsolowych. Przykładowy podgląd klastra za pomocą tego rozszerzenia widoczny jest na rysunku 5.



Rysunek 5. Rozszerzenie Kubernetes w VSC.