



**Politechnika
Śląska**

Dokumentacja projektowa

2023/2024

Zarządzanie systemami informatycznymi

Konteneryzacja i implementacja kubernetes: utworzenie klastra w
usłudze Microsoft Azure

Kierunek: Informatyka

Członkowie zespołu:

Dawid Gala

Hubert Bojda

Gliwice, 2023/2024

Spis treści

1	Wprowadzenie	2
1.1	Cel projektu	2
2	Założenia projektowe	3
2.1	Założenia techniczne i nietechniczne	3
2.2	Stos technologiczny	4
2.3	Oczekiwane rezultaty projektu	5
3	Realizacja projektu	7
3.1	Docker	7
3.2	Kubernetes	10
3.3	Azure AKS	13
3.4	Testy	15
4	Wnioski	18
4.1	Spostrzeżenia	18
4.2	Osiągnięcia	18
4.3	Potencjał rozwoju	19

1 Wprowadzenie

1.1 Cel projektu

Naszym celem jest stworzenie skonteneryzowanej aplikacji, którą następnie chcemy wdrożyć w środowisku chmurowym. Ponieważ wcześniej realizowaliśmy już podobne projekty, tym razem chcemy poszerzyć naszą wiedzę i umiejętności, wdrażając naszą aplikację w Kubernetesie. Chcemy nauczyć się zarówno podstawowych, jak i zaawansowanych aspektów tej technologii oraz zrozumieć, jak można efektywnie zarządzać skonteneryzowanymi aplikacjami w skalowalnym i elastycznym środowisku klastrowym.

2 Założenia projektowe

Plan działań obejmuje kilka kluczowych kroków:

- Stworzenie aplikacji: Zaczniemy od napisania aplikacji, której poszczególne komponenty zostaną skonteneryzowane za pomocą narzędzi takich jak Docker.
- Tworzenie obrazów Docker: Dla każdego komponentu aplikacji utworzymy obrazy Docker, które będą zawierały wszystkie niezbędne zależności i konfiguracje.
- Konfiguracja Kubernetes: Zainstalujemy i skonfigurujemy klaster Kubernetes, który pozwoli nam zarządzać wdrożeniem naszych kontenerów. Na początku to wszystko będzie testowane na środowisku lokalnym - za pomocą Minikube. Poznamy podstawowe elementy, takie jak Pody, Deployments, Services.
- Przygotujemy pliki YAML, które zdefiniują zasoby Kubernetes, takie jak Deployment i Service. Następnie wdrożymy naszą aplikację w klastrze Kubernetes.
- Integracja z chmurą: Gdy już wszystko lokalnie będzie działać tak jak się spodziewamy, skonfigurujemy nasz klaster Kubernetes w środowisku chmurowym Azure Kubernetes Service (AKS), aby zyskać dodatkowe możliwości skalowania i zarządzania infrastrukturą.

2.1 Założenia techniczne i nietechniczne

Założenia techniczne :

- Miejsce w przestrzeni dyskowej na obrazy lokalne i wirtualizacje minikube.
- Subskrypcja w usłudze Microsoft Azure.
- Globalna baza danych - MongoDB udziela darmowej wersji swojej bazy w celach naukowych.

Założenia nietechniczne:

- Termin: Ukończenie do 5 czerwca 2024r, aby zaprezentować na wykładzie.
- Budżet jak najniższy, najlepiej zerowy, subskrypcja studencka usługi Microsoft Azure w tym bardzo nam pomoże.

2.2 Stos technologiczny

Narzędzia i systemy informatyczne związane z projektem:

- **Docker:**
 - Docker to platforma do tworzenia, wdrażania i uruchamiania aplikacji w kontenerach. Umożliwia izolację aplikacji oraz jej zależności w środowisku, co zapewnia bezpieczeństwo środowiska.
- **Docker Hub:**
 - Docker Hub to publiczna rejestracja obrazów kontenerów, gdzie użytkownicy mogą przechowywać, udostępniać i pobierać obrazy Dockera. Umożliwia łatwe publikowanie i dystrybucję kontenerów zarówno w środowiskach deweloperskich, jak i produkcyjnych. W naszym przypadku pozwoli kubernetesowi pobierać najnowsze obrazy, na podstawie których utworzy kontenery w klastrach.
- **Kubernetes:**
 - Kubernetes to system typu Open Source, stworzony przez Google do orkiestracji kontenerów, który automatyzuje wdrażanie, skalowanie i zarządzanie aplikacjami kontenerowymi. Umożliwia zarządzanie klastrami kontenerów i zapewnia ich niezawodność oraz skalowalność.
- **Kubectl:**
 - kubectl to narzędzie wiersza poleceń do interakcji z klastrami Kubernetes. Umożliwia zarządzanie aplikacjami, inspekcję zasobów, wdrażanie i rozwiązywanie problemów w środowisku Kubernetes.
- **Minikube:**
 - Minikube to narzędzie, które pozwala na lokalne uruchamianie klastra Kubernetes na komputerze deweloperskim. Umożliwia testowanie i rozwijanie aplikacji w Kubernetes bez potrzeby używania pełnowymiarowego klastra.
- **Azure:**
 - Microsoft Azure to chmurowa platforma obliczeniowa oferująca różne usługi, w tym wirtualne maszyny, bazy danych, usługi kontenerowe (Azure Kubernetes Service - AKS) oraz narzędzia do zarządzania i monitorowania aplikacji.

- **Visual Studio Code (VS Code):**

- Visual Studio Code to lekki edytor kodu źródłowego, który wspiera wiele języków programowania i narzędzi developerskich. Oferuje funkcje takie jak debugowanie, integracja z Git, oraz liczne rozszerzenia, które ułatwią nam pracę.

- **Postman:**

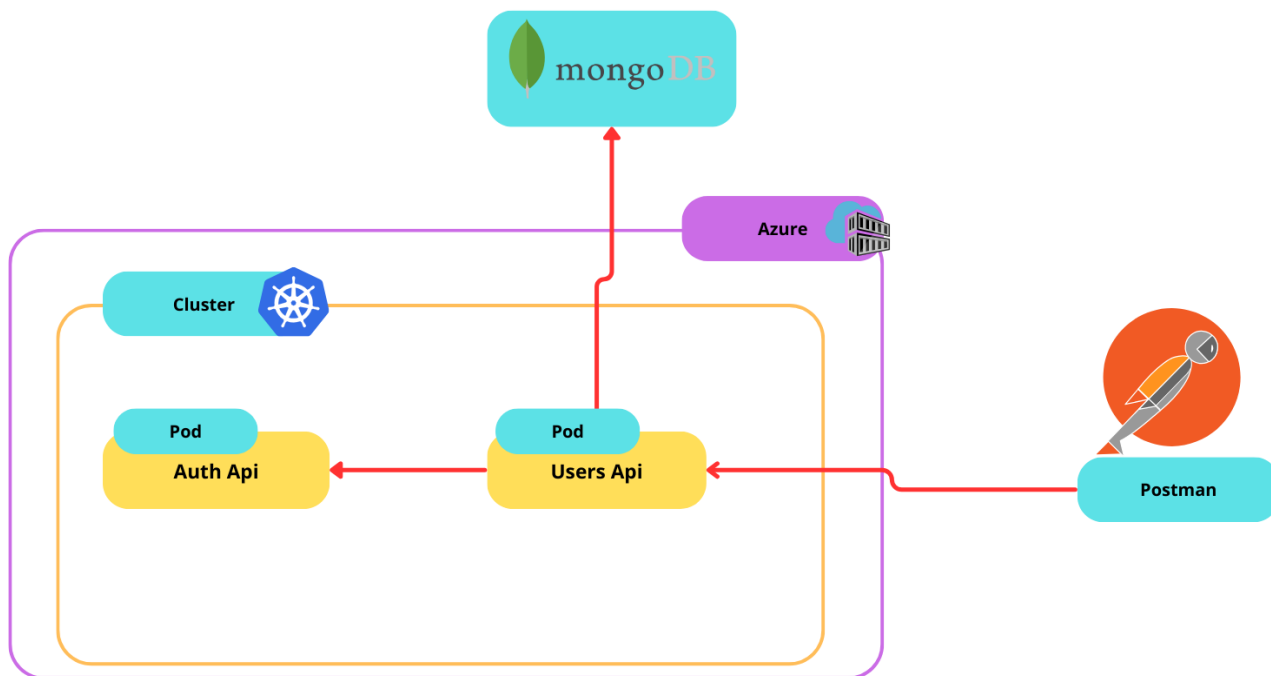
- Postman to narzędzie do testowania API, które umożliwia wysyłanie zapytań HTTP i analizowanie odpowiedzi. Jest używane do tworzenia i testowania interfejsów API, automatyzacji testów oraz współpracy w zespole przy dokumentowaniu API.

- **MongoDB:**

- MongoDB to nierelacyjna baza danych typu NoSQL, która przechowuje dane w formacie dokumentów JSON. Jest skalowalna, elastyczna i przede wszystkim umożliwia nam darmową opcję do testowania rozwiązania.

2.3 Oczekiwane rezultaty projektu

Naszymi oczekiwaniami obejmują skuteczne skonteneryzowanie aplikacji, wdrożenie jej w klastrze Kubernetes i jej dostępność w chmurze Azure. Jednocześnie zależy nam, aby zdolność zapisu do bazy danych MongoDB nie uległa zmianie. Poprzez konteneryzację aplikacji, ta będzie spakowana w kontener Dockerowy i wypchnięta do rejestru Docker Hub i gotowa do wdrożenia w klastrze Kubernetes. Po pomyślnym wdrożeniu, aplikacja będzie dostępna w chmurze Azure, co zapewni elastyczność i skalowalność jej działania. Współpraca z bazą danych MongoDB umożliwi aplikacji przechowywanie i odczytywanie danych logowania użytkownika.



Rysunek 1: Prosty schemat, ukazujący nasze założenia

Aby spełnić wymagania, należy odpowiednio zabezpieczyć obie aplikacje oraz zapewnić komunikację tylko między nimi. Aplikacja "Users Api" będzie wystawiona publicznie, co pozwoli użytkownikom na rejestrację i logowanie. Po poprawnym uwierzytelnieniu, "Users Api" będzie komunikować się z "Auth Api", który będzie odpowiedzialny za generowanie tokenów uwierzytelniających. W ten sposób dane uwierzytelniające będą przechowywane w ukrytej części architektury, niedostępnej dla świata zewnętrznego. W celu sprawdzenia poprawności działania aplikacji, możemy użyć narzędzia takiego jak Postman, które pozwoli na wykonywanie żądań HTTP i sprawdzanie odpowiedzi serwera. Dzięki temu będziemy mogli przetestować poprawność działania naszej aplikacji, jednocześnie zachowując jej bezpieczeństwo. Dzięki Kubernetes, nawet w przypadku awarii naszej aplikacji - którą celowo uwzględniliśmy w kodzie w celach demonstracyjnych - nasza usługa pozostanie nadal dostępna. Jest to efekt orkiestracji i zarządzania kontenerami przez Kubernetes.

Dzięki elastycznemu skalowaniu i automatycznemu przywracaniu usług, Kubernetes zapewnia nieprzerwane działanie naszej aplikacji, nawet w obliczu incydentów. To sprawia, że nasza aplikacja jest bardziej niezawodna i odporna na potencjalne zakłócenia, co przekłada się na lepsze doświadczenie użytkownika i większą pewność siebie w zakresie działania naszej usługi.

3 Realizacja projektu

3.1 Docker

Na początku zajmijmy się konteneryzacją naszych obu aplikacji. Najpierw zajmijmy się aplikacją "Users Api". Przed konteneryzacją warto spojrzeć w kod i zastanowić się co tak naprawdę potrzebujemy. Przy konteneryzacji aplikacji warto spojrzeć na port, na którym ona nasłuchuje. Jednocześnie, dla naszej wygody i ewentualnej łatwej zmiany bazy danych. Wyodrębnimy link do zmiennej środowiskowej.

```
mongoose.connect(  
  process.env.MONGODB_CONNECTION_URI,  
  { useNewUrlParser: true },  
  (err) => {  
    if (err) {  
      console.log('COULD NOT CONNECT TO MONGODB!');  
    } else {  
      app.listen(3000);  
    }  
  }  
);
```

Rysunek 2: Podłączenie się pod MongoDB

Teraz, musimy pamiętać o przypisaniu tej zmiennej, przy uruchomianiu kontenera, co pokażemy później.

Poniżej znajduje się przedstawienie pliku Dockerfile dla utworzenia obrazów obu aplikacji:

```
users-api > Dockerfile > ...
1 FROM node:14-alpine
2
3 WORKDIR /app
4
5 COPY package.json .
6
7 RUN npm install
8
9 COPY . .
10
11 EXPOSE 3000
12
13 CMD [ "node", "users-app.js" ]
```

Rysunek 3: Dockerfile dla aplikacji "Users Api"

```
auth-api > Dockerfile > FROM
1 FROM node:14-alpine
2
3 WORKDIR /app
4
5 COPY package.json .
6
7 RUN npm install
8
9 COPY . .
10
11 EXPOSE 3000
12
13 CMD [ "node", "auth-app.js" ]
```

Rysunek 4: Dockerfile dla aplikacji "Auth Api"

Teraz, gdy już mamy nasz schemat do utworzenia obrazów naszych kontenerów, utworzymy plik "docker-compose.yaml", który nam określi schemat utworzenia kontenerów:

Jak widać na poniższym zdjęciu, do zmiennej środowiskowej:

MONGODB_CONNECTION_URI przypisaliśmy link do naszej bazy.

```
docker-compose.yml > {} services > {} users > {} environment > AUTH_API_ADDRESSS
docker-compose.yml - The Compose specification establishes a standard for the definition of multi-container platform-agnostic applications (compose-spec.json)
1 version: "3"
2 services:
3   auth:
4     build: ./auth-api
5     ports:
6       - '8000:3000'
7     environment:
8       TOKEN_KEY: 'shouldbeverysecure'
9   users:
10    build: ./users-api
11    ports:
12      - '8080:3000'
13    environment:
14      MONGODB_CONNECTION_URI: 'mongodb+srv://dawid:                @kubernetes-azure.dxmak13.mongodb.net/?
15      retryWrites=true&w=majority&appName=kubernetes-azure'
16    AUTH_API_ADDRESSS: 'auth:3000'
```

Rysunek 5: DockerFile, dzięki któremu docker utworzy kontenery

Uruchommy konsolę w ścieżce, w której znajduje się nasz plik "docker-compose.yml" i wywołamy komendę:

docker-compose up -d Komenda ta tworzy kontenery na podstawie zawartego w pliku docker-compose, który znajduje się w naszej ścieżce. Parametr d pozwala nam "odłączyć się" od kontenera.

Po udanych testach za pomocą POSTMAN, możemy przejść do dalszej części.

Teraz zależy nam na umieszczeniu tych obrazów w repozytorium. Umożliwi to późniejsze pobieranie obrazów przez kubernetes, gdy będzie tworzył swoje pody. W każdym katalogu, który zawiera plik dockerfile wywołajmy komendę, która zbuduje nam obraz z określoną nazwą:

docker build . -t baitazar/kubernetes-azure-users

Następnie:

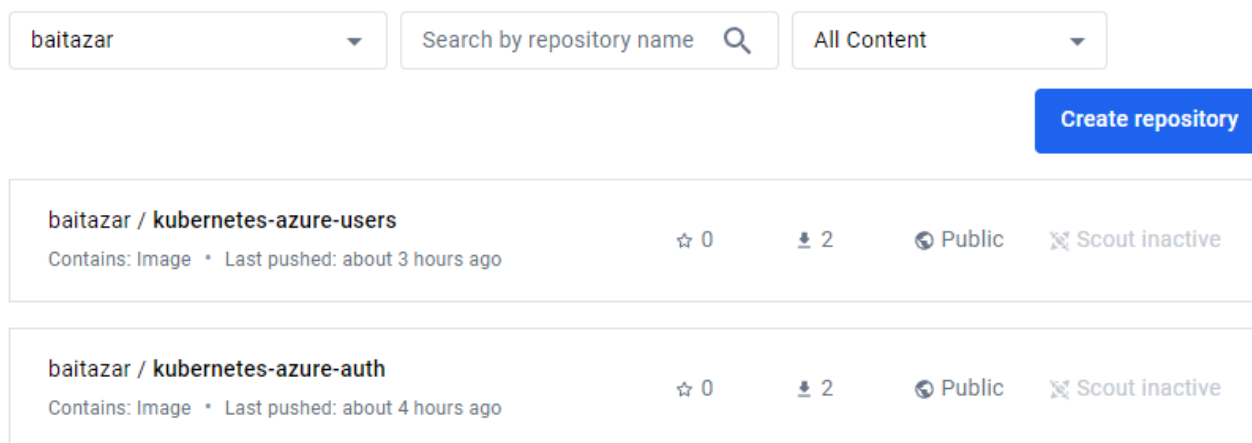
docker push baitazar/kubernetes-azure-users

docker build . -t baitazar/kubernetes-azure-auth

Następnie:

docker push baitazar/kubernetes-azure-auth

Spowoduje to utworzenie obrazu o określonej nazwie, która musi być taka sama jak wcześniej utworzone repozytorium. Następnie za pomocą komendy "docker push" wypychamy ten obraz do repozytorium.



Rysunek 6: Widok naszych obrazów w Docker Hub

3.2 Kubernetes

Zaraz zajmiemy się tworzeniem plików yaml, które zdefiniują nam wszystko co potrzeba do naszego kubernetesa. Ale zanim to, wyjaśnijmy sobie kilka pojęć, które będziemy wykorzystywać:

- **Deployment:** Jest to obiekt Kubernetes, który definiuje sposób wdrażania aplikacji lub mikroservisów. Deployment kontroluje wdrożenie aplikacji poprzez zarządzanie replikami podów. Pozwala to na skalowanie aplikacji w górę lub w dół, automatyczne naprawianie awarii oraz aktualizację aplikacji na żywo bez przerywania dostępności.
- **Pod:** Jest to najmniejsza jednostka wdrażania w Kubernetes. Pod jest instancją pojedynczego kontenera, który zawiera aplikację lub serwis. Może również zawierać współdzielone zasoby, takie jak wolumeny. Pod jest zarządzany przez Kubernetes i może być automatycznie uruchamiany, skalowany i restartowany.
- **Service:** W Kubernetes serwis to abstrakcja, która definiuje politykę dostępu do podów. Serwis zapewnia stały adres IP i nazwę DNS, które mogą być wykorzystane przez inne aplikacje w klastrze do komunikacji z usługą, niezależnie od tego, jak wiele replik podów jest za nią ukrytych. Jest to sposób zapewnienia stałej dostępności usług w klastrze, nawet gdy podstawowe pody są skalowane lub aktualizowane.

```

kubernetes > ! auth.yaml > {} spec > {} selector > {} matchLabels
io.k8s.api.apps.v1.Deployment (v1@deployment.json) | io.k8s.api.core.v1.Service (v1@service.json)
1  ✓ apiVersion: v1
2  ✓ kind: Service
3  ✓ metadata:
4    | name: auth-service
5  ✓ spec:
6    | selector:
7      | app: auth
8    | type: ClusterIP
9    | ports:
10   | - protocol: TCP
11     |   port: 3000
12     |   targetPort: 3000
13   ---
14  ✓ apiVersion: apps/v1
15  ✓ kind: Deployment
16  ✓ metadata:
17    | name: auth-deployment
18  ✓ spec:
19    | replicas: 1
20    | selector:
21      | matchLabels:
22        | app: auth
23    | template:
24      | metadata:
25        | labels:
26          | app: auth
27      | spec:
28        | containers:
29          | - name: auth-api
30            | image: baitazar/kubernetes-azure-auth:latest
31            | env:
32              | - name: TOKEN_KEY
33                | value: 'shouldbeverysecure'
34            | resources:
35              | limits:
36                | memory: "128Mi"
37                | cpu: "500m"
38

```

Rysunek 7: Plik konfiguracyjny dla service i deployment aplikacji auth-api

```

kubernetes > ! users.yaml > {} spec > # replicas
io.k8s.api.apps.v1.Deployment (v1@deployment.json) | io.k8s.api.core.v1.Service (v1@service.json)
1  ∨ apiVersion: v1
2  ∨ kind: Service
3  ∨ metadata:
4  |   name: users-service
5  ∨ spec:
6  |   selector:
7  |     app: users
8  |   type: LoadBalancer
9  |   ports:
10 |     - protocol: TCP
11 |       port: 80
12 |       targetPort: 3000
13 ---
14 ∨ apiVersion: apps/v1
15 ∨ kind: Deployment
16 ∨ metadata:
17 |   name: users-deployment
18 ∨ spec:
19 |   replicas: 1
20 |   selector:
21 |     matchLabels:
22 |       app: users
23 |   template:
24 |     metadata:
25 |       labels:
26 |         app: users
27 |     spec:
28 |       containers:
29 |         - name: users-api
30 |           image: baitazar/kubernetes-azure-users:latest
31 |           env:
32 |             - name: MONGODB_CONNECTION_URI
33 |               value: 'mongodb+srv://dawid:12345678@kubernetes-azure.d
34 |             - name: AUTH_API_ADDRESSS
35 |               value: 'auth-service.default:3000'
36 |       resources:
37 |         limits:
38 |           memory: "128Mi"
39 |           cpu: "500m"
40

```

Rysunek 8: Plik konfiguracyjny dla service i deployment aplikacji user-api

Po takiej implementacji, możemy naszą konfigurację przetestować lokalnie. Jednak my przejdziemy już bezpośrednio do implementacji tego w chmurze.

3.3 Azure AKS

Azure Kubernetes Service (AKS) to usługa zarządzania kontenerami w chmurze Microsoft Azure. Zapewnia zarządzane środowisko do uruchamiania, skalowania i zarządzania aplikacjami opartymi na kontenerach, wykorzystując Kubernetes, popularny system open-source do automatyzacji wdrażania, skalowania i zarządzania aplikacjami kontenerowymi. Dzięki Azure Kubernetes Service organizacje mogą łatwo wdrażać, zarządzać i skalować aplikacje kontenerowe w chmurze Microsoft Azure, korzystając z elastyczności i wydajności środowiska Kubernetes, przy jednoczesnym zapewnieniu zaawansowanych funkcji i integracji z ekosystemem Azure. Dzięki darmowej subskrypcji studenckiej, którą oferuje nam Politechnika Śląska, możemy bez obaw o nasz budżet uruchomić w tym nasze środowisko. Microsoft Azure można zarządzać graficznie, jak i konsolowo. My będziemy używać naprzemiennie obu możliwości. Przejdźmy teraz do konfiguracji.

Microsoft Azure

Home > Kubernetes services >

Create Kubernetes cluster

Basics Node pools Networking Integrations Monitoring Advanced Tags Review + create

[View automation template](#)

Basics

Subscription	Azure for Students
Resource group	Kubernetes-Azure
Region	Poland Central
Kubernetes cluster name	Kubernetes-Project
Kubernetes version	1.28.9
Automatic upgrade	patch
Automatic upgrade scheduler	Every week on Sunday (recommended)
Node security channel type	NodeImage
Security channel scheduler	Every week on Sunday (recommended)

Node pools

Node pools	1
Enable virtual nodes	Disabled

Access

Resource identity	System-assigned managed identity
Local accounts	Enabled
Authentication and Authorization	Local accounts with Kubernetes RBAC
Encryption type	(Default) Encryption at-rest with a platform-managed key

Networking

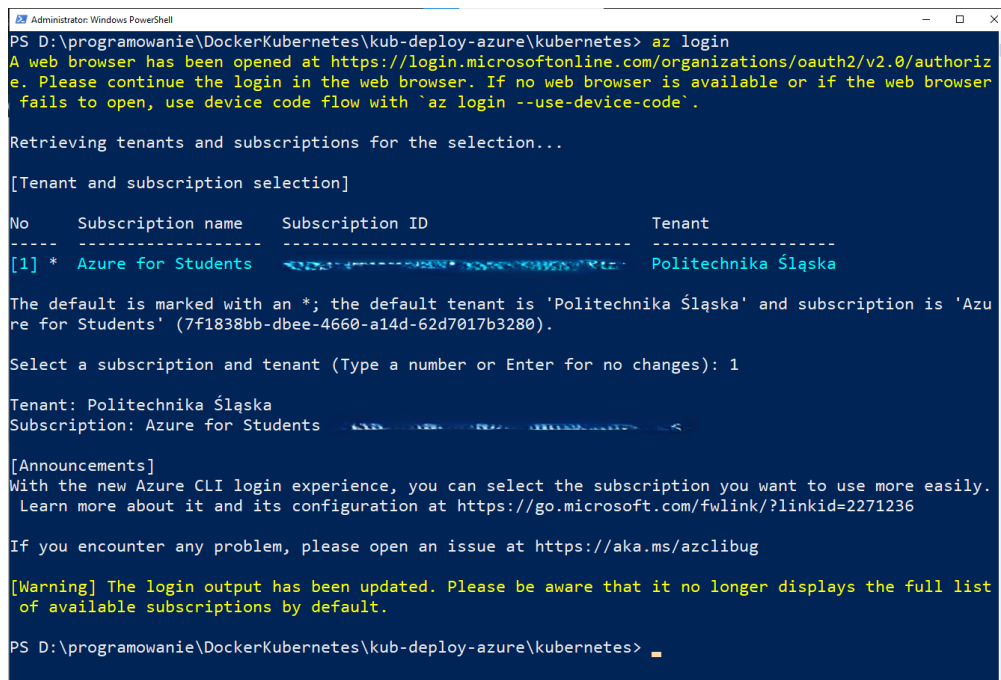
Private cluster	Disabled
Authorized IP ranges	Disabled

[Previous](#) [Next](#) [Create](#)

<https://portal.azure.com/#>

Rysunek 9: Panel tworzenia AKS

Tworzymy naszą usługę AKS tak jak na powyższym. Większość pozostaje domyślna. Jedyne co zmieniamy to ilość podów, bo nam tak dużo nie jest potrzebne. Resztę będziemy konfigurować z terminala z zainstalowanym narzędziem Azure CLI.



```
PS D:\programowanie\DockerKubernetes\kub-deploy-azure\kubernetes> az login
A web browser has been opened at https://login.microsoftonline.com/organizations/oauth2/v2.0/authorize. Please continue the login in the web browser. If no web browser is available or if the web browser fails to open, use device code flow with 'az login --use-device-code'.

Retrieving tenants and subscriptions for the selection...

[Tenant and subscription selection]

No      Subscription name      Subscription ID      Tenant
-----
[1] *    Azure for Students  7f1838bb-dbee-4660-a14d-62d7017b3280  Politechnika Śląska

The default is marked with an *; the default tenant is 'Politechnika Śląska' and subscription is 'Azure for Students' (7f1838bb-dbee-4660-a14d-62d7017b3280).

Select a subscription and tenant (Type a number or Enter for no changes): 1

Tenant: Politechnika Śląska
Subscription: Azure for Students

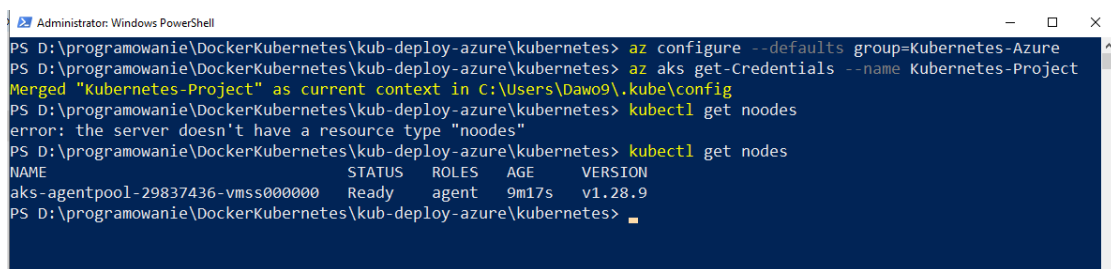
[Announcements]
With the new Azure CLI login experience, you can select the subscription you want to use more easily.
Learn more about it and its configuration at https://go.microsoft.com/fwlink/?linkid=2271236

If you encounter any problem, please open an issue at https://aka.ms/azclibug

[Warning] The login output has been updated. Please be aware that it no longer displays the full list of available subscriptions by default.

PS D:\programowanie\DockerKubernetes\kub-deploy-azure\kubernetes>
```

Rysunek 10: Zalogowanie się do usługi Azure z poziomu konsoli



```
PS D:\programowanie\DockerKubernetes\kub-deploy-azure\kubernetes> az configure --defaults group=Kubernetes-Azure
PS D:\programowanie\DockerKubernetes\kub-deploy-azure\kubernetes> az aks get-credentials --name Kubernetes-Project
Merged "Kubernetes-Project" as current context in C:\Users\Dawo9\.kube\config
PS D:\programowanie\DockerKubernetes\kub-deploy-azure\kubernetes> kubectl get nodes
error: the server doesn't have a resource type "noodes"
PS D:\programowanie\DockerKubernetes\kub-deploy-azure\kubernetes> kubectl get nodes
NAME                                STATUS    ROLES    AGE    VERSION
aks-agentpool-29837436-vmss000000  Ready    agent    9m17s  v1.28.9
PS D:\programowanie\DockerKubernetes\kub-deploy-azure\kubernetes>
```

Rysunek 11: Ustawienie domyślnej grupy zasobów i utworzenie kontekstu dla narzędzia kubectl

```
Administrator: Windows PowerShell
PS D:\programowanie\DockeKubernetes\kub-deploy-azure\kubernetes> kubectl apply -f .\auth.yaml,.\users.yaml
service/auth-service created
deployment.apps/auth-deployment created
service/users-service created
deployment.apps/users-deployment created
PS D:\programowanie\DockeKubernetes\kub-deploy-azure\kubernetes> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
auth-deployment-749857c95c-8rxlh    1/1     Running   0           19s
users-deployment-56d9dc8f48-9http    1/1     Running   0           19s
PS D:\programowanie\DockeKubernetes\kub-deploy-azure\kubernetes> kubectl get service
NAME            TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
auth-service    ClusterIP     10.0.232.174  <none>         3000/TCP         28s
kubernetes      ClusterIP     10.0.0.1      <none>         443/TCP          5m32s
users-service   LoadBalancer 10.0.53.129   20.215.88.123  80:31253/TCP     27s
PS D:\programowanie\DockeKubernetes\kub-deploy-azure\kubernetes>
```

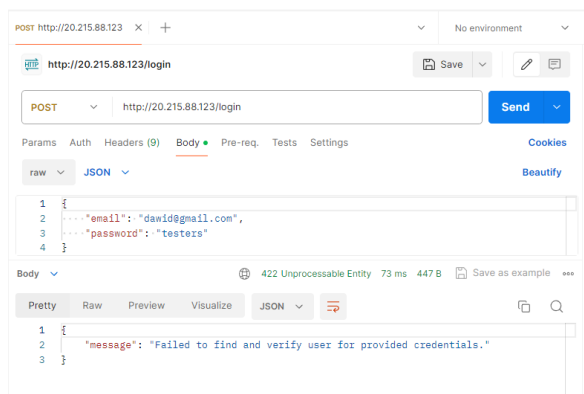
Rysunek 12: Przesłanie konfiguracji do naszego klastra

Po wykonaniu powyższych poleceń, możemy pozyskać adres IP, dzięki któremu możemy komunikować się z naszymi podami.

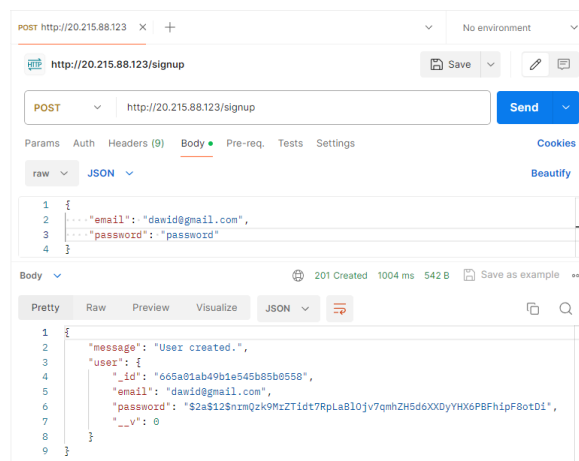
```
kubectl get service
```

Dzięki tej komendzie możemy pobrać IP, w naszym przypadku jest to: 20.215.88.123
Przejdźmy teraz do Postmana. Przypomnijmy sobie na czym polega nasza aplikacja. Nasza aplikacja zawiera dwie usługi: User Api - odpowiedzialna za rejestrację i logowanie oraz Auth Api - która polega na autoryzacji użytkownika.

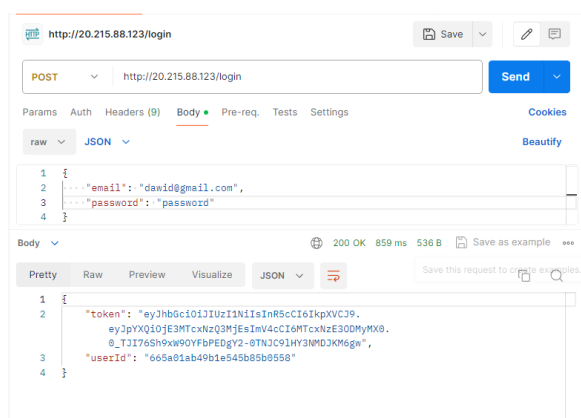
3.4 Testy



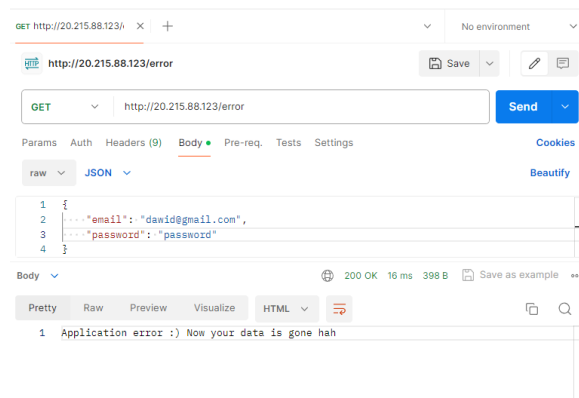
Rysunek 13: Logowanie na nieistniejącego usera



Rysunek 14: Rejestracja



Rysunek 15: Logowanie na istniejącego usera



Rysunek 16: Wywołajnie crasha aplikacji

QUERY RESULTS: 1-3 OF 3	
	<pre> _id: ObjectId('6659913c86bd4529d2a97c85') email: "dawo@onet.pl" password: "\$2a\$12\$p49p45vXCa7CNaxc1KnL300.0bz.M9vhJ3TcinH8aRNP3X4jYdcAq" __v: 0 </pre>
	<pre> _id: ObjectId('6659a4ca4dac5ac4ff334a21') email: "hubert@gmail.pl" password: "\$2a\$12\$6TybJ2.PxXvE6gCw4Q8KGusD6ax789sSblhHcbhs.wHqMb.asv/tW" __v: 0 </pre>
	<pre> _id: ObjectId('665a01ab49b1e545b85b0558') email: "dawid@gmail.com" password: "\$2a\$12\$nrnQzk9MrZTidt7RplabL0jv7qmhZH5d6XXDyYHX6PBfhipF8otDi" __v: 0 </pre>

Rysunek 19: Widok z bazy danych

4 Wnioski

4.1 Spostrzeżenia

- Konteneryzacja to technologia, która znacząco zmienia sposób, w jaki tworzymy, wdrażamy i zarządzamy aplikacjami webowymi. Wykorzystanie kontenerów nie tylko ułatwia procesy związane z tworzeniem aplikacji, ale również przyczynia się do ich bezpieczeństwa.
- Kubernetes to potężny system, który rzeczywiście wymaga znacznej wiedzy i umiejętności do efektywnej obsługi. Jego złożoność wynika z bogactwa funkcji i możliwości, które oferuje, aby zarządzać aplikacjami kontenerowymi na dużą skalę.
- Azure Kubernetes Service (AKS) umożliwia łatwe skalowanie aplikacji zarówno w poziomie (dodawanie nowych instancji), jak i w pionie (przydzielanie większej ilości zasobów do istniejących instancji), co pozwala na dostosowanie się do zmieniających się wymagań obciążeniowych.

4.2 Osiągnięcia

- Utworzenie kontenerów na podstawie kodu źródłowego.
- Podłączenie aplikacji do bazy danych.
- Utworzenie AKS .

- Poprawne ustawienie klastra w usłudze AKS z naszymi kontenerami.
- Poprawne działanie całej aplikacji, nawet po crashu.

4.3 Potencjał rozwoju

- Utworzenie stałego DNS dla adresu IP serwisu. Obecnie, gdy edytujemy serwis w Kubernetes, adres IP serwisu może się zmieniać.
- Rozszerzenie logiki aplikacji poprzez rozwijanie funkcji na oddzielnych gałęziach kodu.
- Wdrożenie strategii Continuous Integration/Continuous Deployment CI/CD automatyzuje procesy budowania, testowania i wdrażania aplikacji, co zwiększa efektywność i niezawodność.
- Zwiększenie bezpieczeństwa aplikacji za pomocą wdrożenia lepszych podsieci.