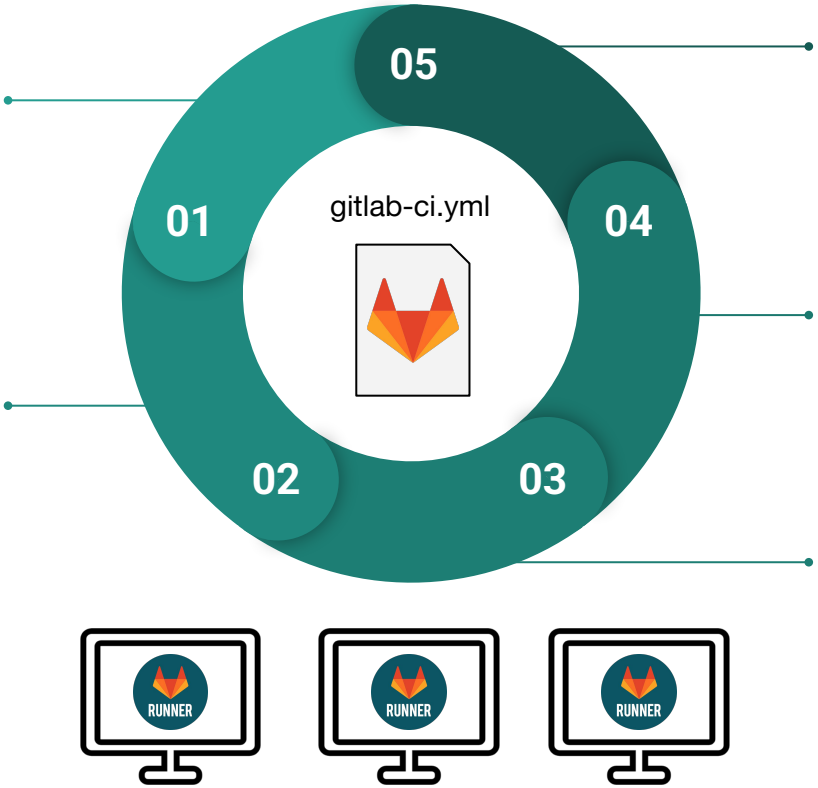
Wosk AutoSerwis

Miłosz Kutyła, Jakub Ossowski,
Kacper Szaruch, Jan Wojciechowski

Diagram faz CI/CD

Kontrola wersji
I etap continous integration
Wykorzystanie narzędzia wersjonowania do rozwoju systemu
`git push`

Build
II etap continous integration
Budowanie rozwiązania .NET
`dotnet build`

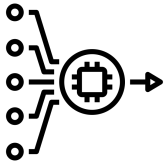


Deploy
Wdrożenie systemu w docelowym środowisku wdrożeniowym wykorzystując narzędzie orkiestracji
`kubectl apply`
`kubectl rollout`

Delivery
Zbudowanie obrazu kontenera zawierającego moduł i opublikowanie go w repozytorium obrazów
`docker build`
`docker push`

Test
III etap continous integration
Przeprowadzenie testów jednostkowych rozwiązania
`dotnet test`

Fazy CI/CD: build



Schemat fazy build w pliku **gitlab-ci.yml**

```
build:
  stage: build
  before_script:
    - cd <ścieżka do folderu z implementacją dotnet>
  script:
    - dotnet restore
    - dotnet build -c Debug
    - dotnet build -c Release
  tags:
    - build
```

Zakres: budowa rozwijanego projektu. Ma na celu sprawdzenie, czy wprowadzone zmiany nie powodują błędów kompilacji.

Fazy CI/CD: test

Schemat fazy test w pliku **gitlab-ci.yml**

```
test:
  stage: test
  before_script:
    - cd <ścieżka do folderu z implementacją testów>
  script:
    - dotnet restore
    - dotnet build -c Debug
    - dotnet build -c Release
    - dotnet test
  tags:
    - test
```

Zakres: testowanie. Ma na celu sprawdzenie, czy wprowadzone zmiany nie zmieniają zaprojektowanego działania modułu.

Fazy CI/CD: delivery

Schemat fazy delivery w pliku **gitlab-ci.yml**

```
delivery:
  stage: delivery
  before_script:
    - cd Docker
    - docker rm --force $CONTAINER_NAME
    - docker rmi --force $DOCKERHUB_IMAGE_PATH
  script:
    - echo $DOCKER_TOKEN | docker login -u wosk --password-stdin
    - docker build -f ../X/Dockerfile.prod -t $DOCKERHUB_IMAGE_PATH ..
    - docker image prune --filter label=stage=X-rest_build--force
    - docker push $DOCKERHUB_IMAGE_PATH
    - docker logout
  tags:
    - delivery
```

Wzorzec pliku **Dockerfile**

```
FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
LABEL stage=Y_build

WORKDIR /src
COPY . .

WORKDIR "/src/X"
RUN dotnet restore
RUN dotnet build -c Release -o /app/build
RUN dotnet publish -c Release -o /app/publish

FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS runtime
WORKDIR /app
EXPOSE 80
EXPOSE 443
COPY --from=build /app/publish .
ENTRYPOINT ["dotnet", "Wosk.AutoSerwis.X.dll"]
```

Zakres: usunięcie lokalnego kontenera i obrazu, zalogowanie w usłudze Docker, zbudowanie nowego obrazu, usunięcie kopii obrazu, opublikowanie nowego obrazu w serwisie Docker Hub

gdzie **X** to nazwa konteneryzowanego projektu,
Y to nazwa związanego z nim etapu np.:
X=Wosk.AutoSerwis.WebMechanicApp.BlazorServer
Y=wosk-autoserwis-webmechanicapp

Zmienne środowiskowe:

- $CONTAINER_NAME — nazwa kontenera np. mechanicappusvc
- $DOCKERHUB_IMAGE_PATH — wskaźnik na odpowiedni obraz w serwisie Docker Hub np. wosk/wosk.autoserwis:mechanicappusvc-rest
- $DOCKER_TOKEN — token do uwierzytelnienia w serwisie Docker Hub

Fazy CI/CD: deploy

Schemat fazy deploy w pliku **gitlab-ci.yml**

```
deploy:
  stage: deploy
  before_script:
    - cd Kubernetes
  script:
    - kubectl config set-cluster docker-desktop-cluster
--server=https://127.0.0.1:6443 --insecure-skip-tls-verify
    - kubectl config set-credentials admin --token=$KUBERNETES_TOKEN
    - kubectl config set-context docker-desktop
--cluster=docker-desktop-cluster --namespace=default --user=admin
    - kubectl config use-context docker-desktop
    - kubectl apply -f X-deployment.yaml, X-service.yaml
    - kubectl rollout restart deployment/x
  after_script:
    - docker container prune -f
  tags:
    - deploy
```

gdzie **X** to nazwa wdrażanego kontenera np.: mechanicapp, clientsdatausvc

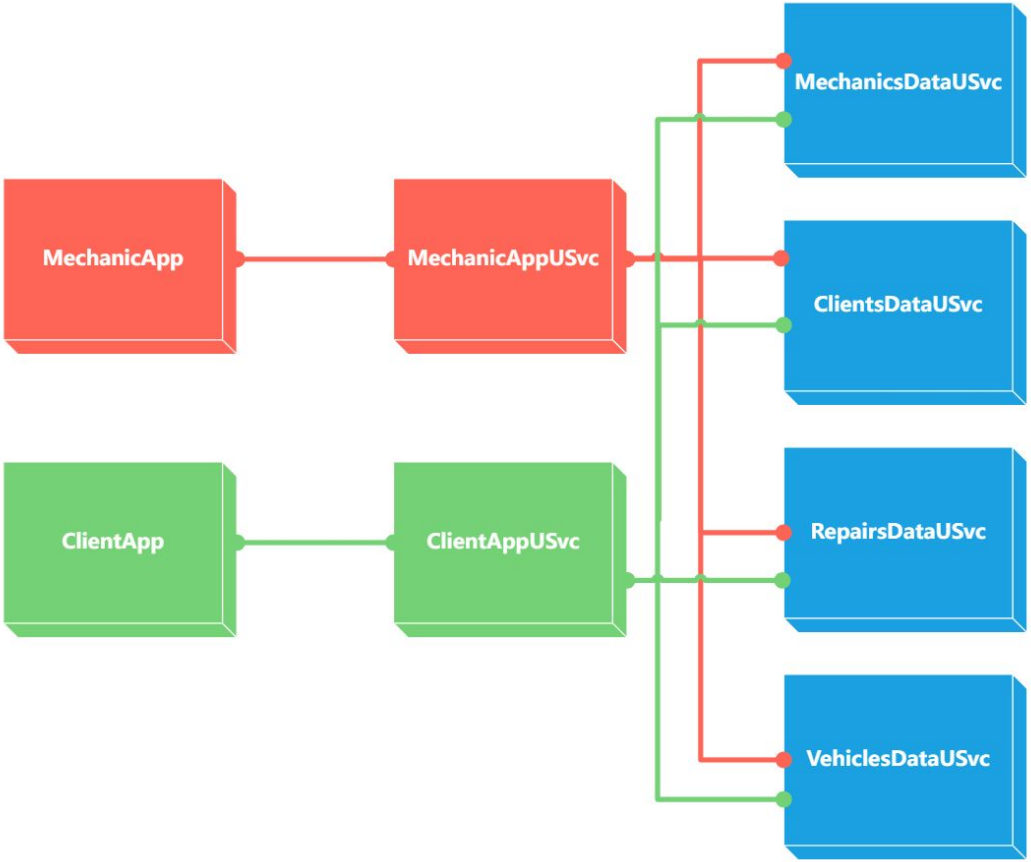
Zmienne środowiskowe:

- $KUBERNETES_TOKEN token do uwierzytelnienia w k8s

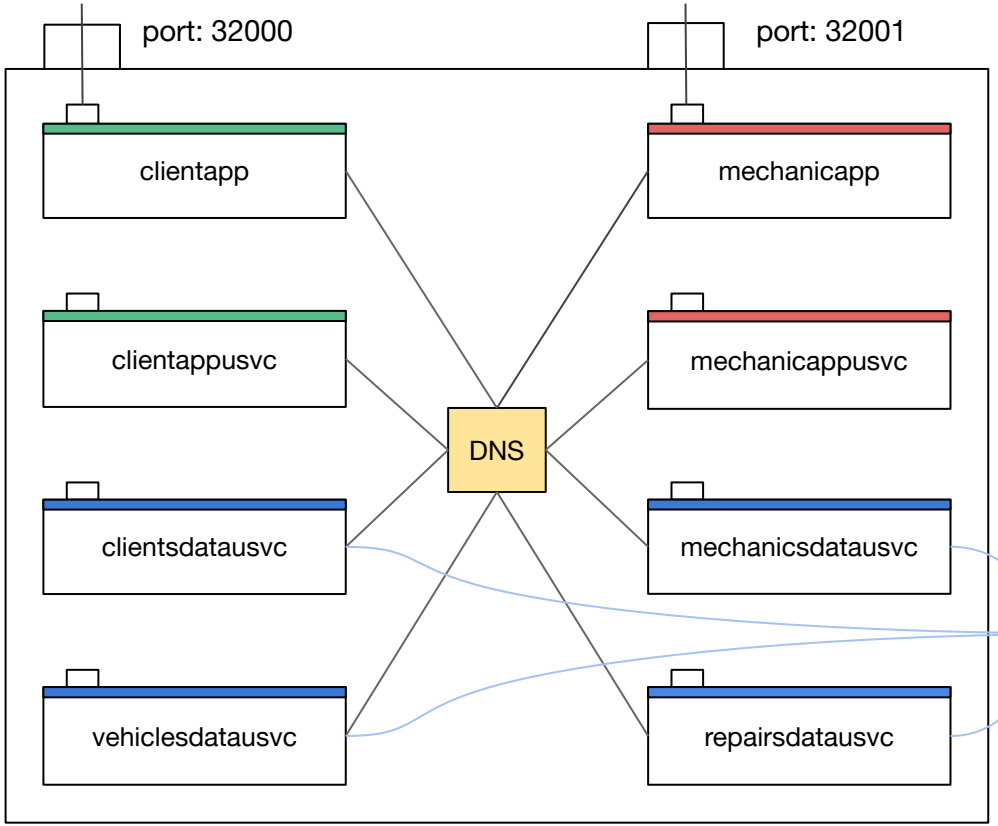
Zakres:

- Konfiguracja klastra Kubernetes, ustawienie adresu serwera,
- Ustawienie poświadczenia dla użytkownika "admin" na podstawie dostarczonego tokena,
- Ustawienie kontekstu, definicja klastra, przestrzeni nazw (namespace) i użytkownika,
- Użycie wcześniej ustawionego kontekstu "`docker-desktop`",
- Wdrożenie plików YAML zawierających definicje zasobów, takich jak deployment i usługa specyficznych dla danej mikrousługi,
- Restart wdrożenia, który wymusza restart kontenerów z nim związanych
- Usunięcie nieużywanych kontenerów

Schemat konfiguracji wdrożeniowej systemu

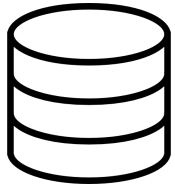


Docker Compose: kompozycja systemu



Interakcja z systemem i między kontenerami:

- aplikacja Klienta jest dostępna pod adresem http://*localhost:32000*
- aplikacja Mechanika jest dostępna pod adresem http://*localhost:32001*
- każda usługa udostępniana przez kontenery wystawiona jest na porcie 80, przy czym nazwa hosta jest tożsama z nazwą kontenera



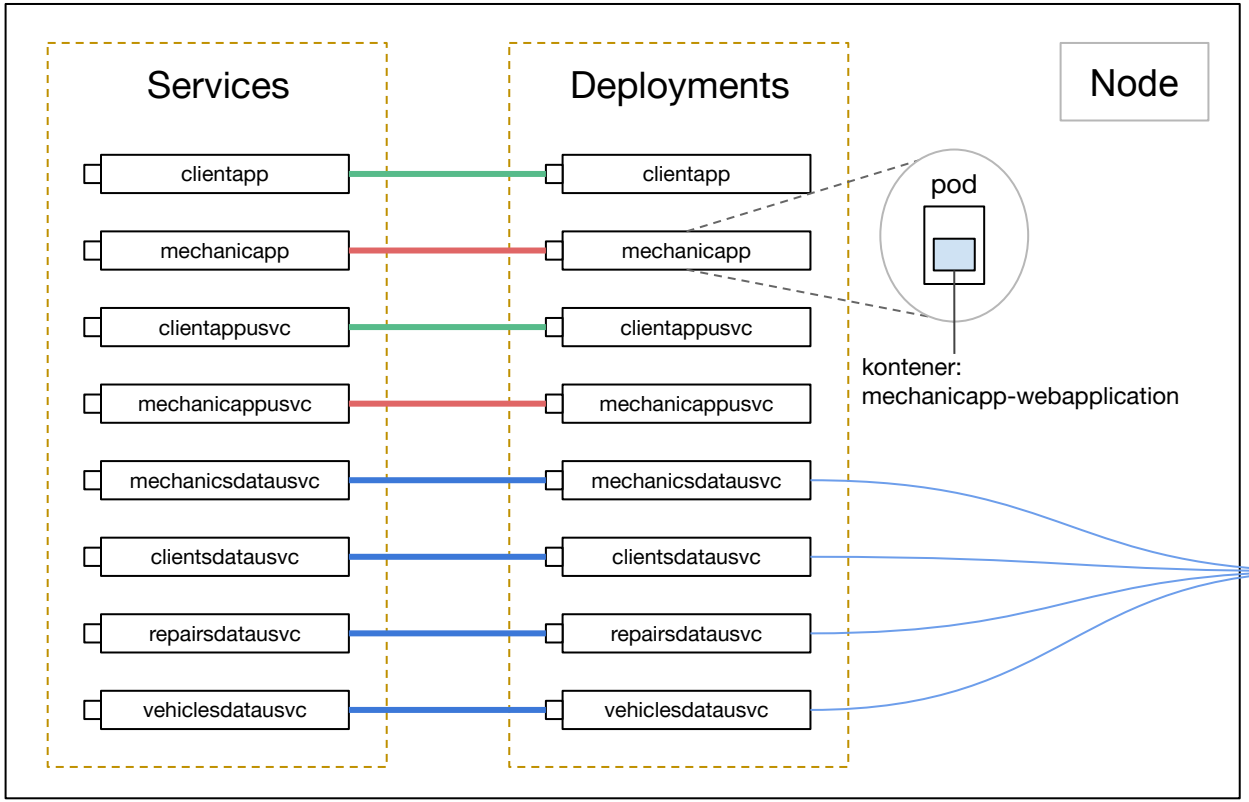
local volume

Struktura magazynu:

4 pliki **X**.json dla odpowiadających mikrousług danych **X**USvc

gdzie **X** to:
clients, mechanics, repairs lub vehicles

Kubernetes: kompozycja systemu



Interakcja z systemem i między kontenerami oraz struktura magazynu jest identyczna jak na slajdzie dot. kompozycji systemu na platformie Docker Compose