

# **Dokumentacja TriggerMesh**

Michał Kiełkowski, Sebastian Misztal, Krzysztof Miśkiewicz, Paweł Steczkiewicz

## Serwer

Kod prostego serwera pythonowego, napisanego we frameworku Flask. Serwer udostępnia jeden endpoint zwracający jako odpowiedź string. Analogicznie dla tego serwera napisane są 2 inne (premium oraz legacy), które różnią się zwracanym stringiem oraz dependencjami.

```
# basic/app.py

from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello from the basic version!'

if __name__ == '__main__':
    app.run()
```

Plik requirements.txt przechowujący listę dependencji dla pojedynczego serwera.

```
Flask==3.0.3
```

## Docker

Plik Dockerfile odpowiadający za konteneryzację serwera. Obraz budowany jest na podstawie obrazu pythona (dla tego serwisu w wersji 3.12). Przy konteneryzacji instalowane są wszystkie biblioteki zdefiniowane w pliku requirements.txt. Następnie przy serwis jest uruchamiany na porcie 5000, który to jest eksponowany z obrazu na zewnątrz.

```
# basic/Dockerfile

FROM python:3.12

WORKDIR /basic-app
COPY . .
RUN pip install -r requirements.txt

ENV FLASK_RUN_PORT=5000
EXPOSE $FLASK_RUN_PORT

CMD flask run --host=0.0.0.0 --port=$FLASK_RUN_PORT
```

Dodatkowo zdefiniowany jest plik .dockerignore, który określa pliki oraz folder nie będące kopiowane do kontenera podczas jego tworzenia.

```
env
__pycache__
```

## Kubernetes

Następnie uruchamiany jest klaster minikube. Jest to klaster kubernetes, który umożliwia tworzenie komponentów kubernetes lokalnie.

```
trigger-mesh-components git:(app_versions) * minikube start
minikube v1.29.0 na Darwin 14.4.1
Using the docker driver based on existing profile
Starting control plane node minikube in cluster minikube
Pulling base image ...
Restarting existing docker container for "minikube" ...
Przygotowywanie Kubernetesa v1.26.1 na Docker 20.10.23...
Configuring bridge CNI (Container Networking Interface) ...
Verifying Kubernetes components...
Po włączeniu addona wykonaj komendę "minikube tunnel". Twoje zasoby będą dostępne pod adresem "127.0.0.1"
  ■ Using image gcr.io/k8s-minikube/storage-provisioner:v5
Po włączeniu addona wykonaj komendę "minikube tunnel". Twoje zasoby będą dostępne pod adresem "127.0.0.1"
  ■ Using image gcr.io/k8s-minikube/minikube-ingress-dns:0.0.2
  ■ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20220916-gd32f8c343
  ■ Using image registry.k8s.io/ingress-nginx/controller:v1.5.1
  ■ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20220916-gd32f8c343
Verifying ingress addon...
Enabled addons: storage-provisioner, default-storageclass, ingress-dns, ingress
! /usr/local/bin/kubectl jest w wersji 1.29.1, co może być niekompatybilne z Kubernetesem w wersji 1.26.1.
  ■ Want kubectl v1.26.1? Try 'minikube kubectl -- get pods -A'
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Jako że nasz klaster kubernetes będzie korzystał z komponentu ingress kontroler, należy dostarczyć jego konkretną implementację od jednego z zewnętrznych dostawców. W tym przypadku użyto implementacji dostarczonej przez nginx.

```
Trigger_Mesh git:(app_versions) * minikube addons enable ingress
ingress is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/master/OWNERS
Po włączeniu addona wykonaj komendę "minikube tunnel". Twoje zasoby będą dostępne pod adresem "127.0.0.1"
  ■ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20220916-gd32f8c343
  ■ Using image registry.k8s.io/ingress-nginx/controller:v1.5.1
  ■ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20220916-gd32f8c343
```

Aby udostępnić możliwość podłączenia się do klastra z zewnątrz, wykonane jest tunelowanie klastra.

```
trigger-mesh-components git:(app_versions) * minikube tunnel
Tunnel successfully started
NOTE: Please do not close this terminal as this process must stay alive for the tunnel to be accessible ...
! The service/ingress app-ingress requires privileged ports to be exposed: [80 443]
sudo permission will be asked for it.
Starting tunnel for service app-ingress.
Password:
Sorry, try again.
Password:
```

Kolejnym krokiem jest zdefiniowanie komponentów kubernetes w plikach yaml. Obejmować one będą deployment (dla wszystkich 3 aplikacji), service (także dla 3 aplikacji) oraz ingress (jeden plik, udostępniający pod wskazanym adresem wszystkie serwisy).

Deployment definiuje przede wszystkim nazwę poda (basic-app), liczbę jego replik (3), obraz na bazie którego jest on definiowany (basic-app), a także port kontenera, na który wysyłane są zapytania.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: basic-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: basic-app
  template:
    metadata:
      labels:
        app: basic-app
    spec:
      containers:
      - name: basic-app
        image: ugodowy518wachta/trigger-mesh:basic-app
        ports:
        - containerPort: 5000

```

Serwis odpowiada za przydzielenie stałego adresu IP do poda, nawet w przypadku jego śmierci. Należy podać pod na podstawie którego będzie on działać, a także mapowanie portów w nim zachodzące.

```

apiVersion: v1
kind: Service
metadata:
  name: basic-app-service
spec:
  selector:
    app: basic-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000

```

Na koniec należy napisać definicję komponentu ingress. Jest on gatewayem do klastra. Definiuje adresy url, na których nasłuchują konkretne serwisy. W przypadku ich dopasowania, zapytania HTTP są przekierowywane bezpośrednio do odpowiednich serwisów. Określa on także nazwę domenową aplikacji (app.com), dzięki czemu można połączyć się z nią z wykorzystaniem przyjaznej domeny zamiast adresu IP.

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
  name: app-ingress
spec:
  rules:
    - host: app.com
      http:
        paths:
          - path: /basic
            pathType: ImplementationSpecific
            backend:
              service:
                name: basic-app-service
                port:
                  number: 80
          - path: /premium
            pathType: ImplementationSpecific
            backend:
              service:
                name: premium-app-service
                port:
                  number: 80
          - path: /legacy
            pathType: ImplementationSpecific
            backend:
              service:
                name: legacy-app-service
                port:
                  number: 80

```

Aby wszystkie komponenty zdefiniowane w plikach yaml zostały wdrożone do klastra, dla każdego z nich wywoływana jest komenda **kubectl apply -f <file\_name>**, jak poniżej.

```

➔ basic git:(app_versions) ✖ kubectl apply -f basic-app-service.yaml && kubectl apply -f legacy-app-service.yaml&& kubectl apply -f premium-app-service.yaml

```

Aby zweryfikować poprawność działania wszystkich podów można użyć komendy **kubectl get pods**, która powinna zwracać odpowiedź jak poniżej:

NAME	READY	STATUS	RESTARTS	AGE
basic-app-6ddc966796-pcbjb	1/1	Running	5 (11h ago)	16d
basic-app-6ddc966796-vkicz	1/1	Running	5 (11h ago)	16d
basic-app-6ddc966796-zcrtp	1/1	Running	5 (11h ago)	16d
legacy-app-b79d8577f-npjwn	1/1	Running	5 (11h ago)	16d
legacy-app-b79d8577f-pt5z5	1/1	Running	5 (11h ago)	16d
legacy-app-b79d8577f-q2vpz	1/1	Running	5 (11h ago)	16d
nginx-deployment-6b7f675859-j42bk	1/1	Running	4	3d1h
nginx-deployment-6b7f675859-kt2zs	1/1	Running	4 (11h ago)	3d1h
nginx-deployment-6b7f675859-zqfrv	1/1	Running	4	3d1h
premium-app-859597bff5-jwpjq	1/1	Running	5 (11h ago)	16d
premium-app-859597bff5-q7glj	1/1	Running	5 (11h ago)	16d
premium-app-859597bff5-st5bs	1/1	Running	5 (11h ago)	16d

Aby żądania HTTP było poprawnie przekierowywane przez przeglądarkę, w pliku /etc/hosts dodana zostaje linijka definiującą mapowanie domeny app.com na adres IP 127.0.0.1.

```

127.0.0.1 app.com

```

Teraz kiedy wszystkie komponenty w klastrze działają poprawnie, a także zostało zdefiniowane odpowiednie mapowanie strony domenowej na adres IP, można użyć narzędzia curl wysłania zapytania HTTP na jeden z serwisów.

```

➔ basic git:(app_versions) ✖ curl http://app.com/basic
Hello from the basic version!%

```

Widać że zwrócona została odpowiedź zdefiniowana w kodzie aplikacji, co potwierdza że architektura serwisów działa poprawnie.

## Próba uruchomienia TriggerMesh

Użyte komendy z **Getting Started**:

<https://docs.triggermesh.io/1.27/get-started/quickstart/>

1. `brew install triggermesh/cli/tmctl`
2. `tmctl create broker foo`
3. `tmctl watch` (on second terminal)
4. `tmctl create source httppoller \`

```
--endpoint https://corporatebs-generator.sameerkumar.website/ \
```

```
--eventType buzzword.phrase \
```

```
--interval 20s \
```

```
--method GET
```

5. `tmctl create target \`

```
--name console \
```

```
--from-image gcr.io/triggermesh/triggermesh-console:v0.0.1 \
```

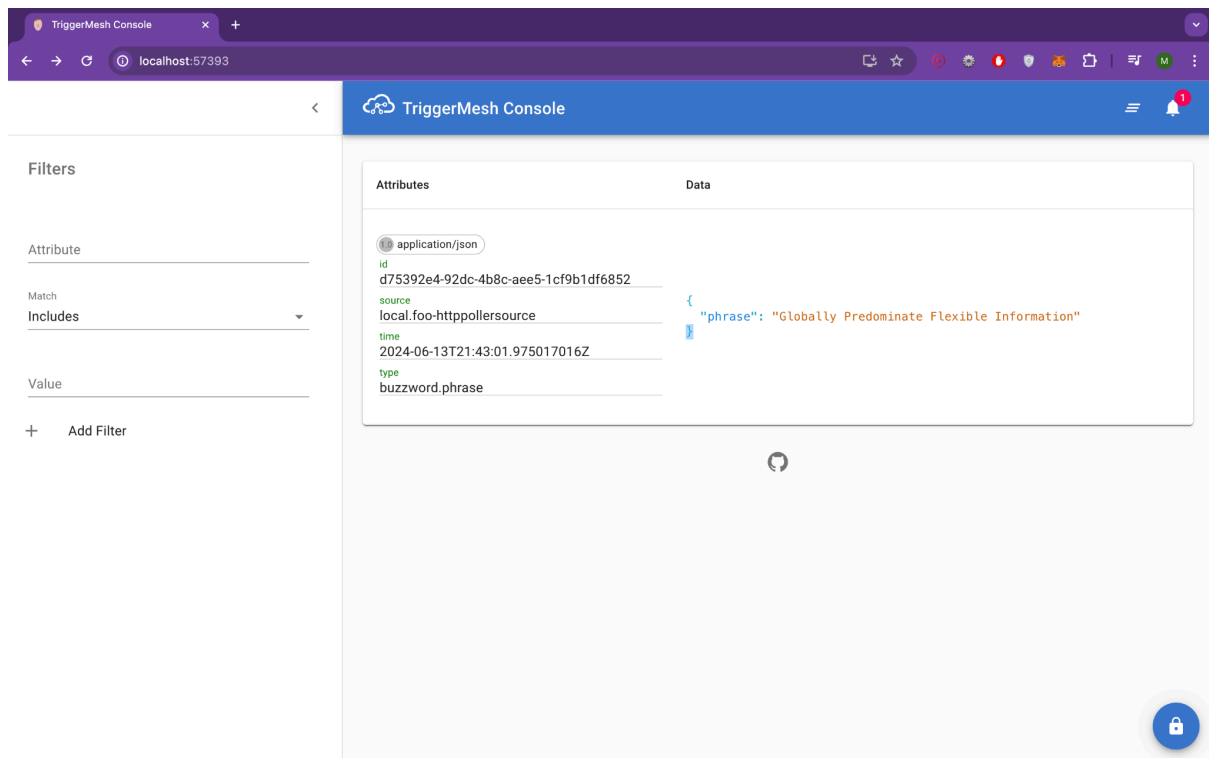
```
--source foo-httppollersource
```

## Output:

```
michalkielkowski ~ -zsh — 80x24
--version string TriggerMesh components version. (default "v1.23.0")
2024/06/13 23:40:24 source initialization: CR validation: validation failure lis
t:
.eventType in body is required
.method in body is required
.endpoint in body is required
.interval in body is required
zsh: command not found: --endpoint
zsh: command not found: --eventType
zsh: command not found: --interval
zsh: command not found: --method
(base) michalkielkowski@Michas-MacBook-Pro-2 ~ % tmctl create source httppoller
\
  --endpoint https://corporatebs-generator.sameerkumar.website/ \
  --eventType buzzword.phrase \
  --interval 20s \
  --method GET
2024/06/13 23:40:37 foo | Updating manifest
2024/06/13 23:40:37 foo | Starting container
Created object name: foo-httpollersource
Component produces: buzzword.phrase
(base) michalkielkowski@Michas-MacBook-Pro-2 ~ %

michalkielkowski ~ tmctl watch — 80x24
Context Attributes,
specversion: 1.0
type: buzzword.phrase
source: local.foo-httpollersource
id: 56faf07e-0332-42b2-afb6-3c29f6408e40
time: 2024-06-13T21:40:41.633282876Z
datacontenttype: application/json
Data,
{
  "phrase": "Objectively Implement Sticky Functionalities"
}
cloudevents.Event
Context Attributes,
specversion: 1.0
type: buzzword.phrase
source: local.foo-httpollersource
id: bd80017d-2b25-431c-ae42-1eccdc97de34
time: 2024-06-13T21:41:01.782637018Z
datacontenttype: application/json
Data,
{
  "phrase": "Distinctively Promote Optimal Virtualization"
}
```

```
michalkielkowski ~ -zsh — 80x24
zsh: command not found: --method
(base) michalkielkowski@Michas-MacBook-Pro-2 ~ % tmctl create source httppoller
\
  --endpoint https://corporatebs-generator.sameerkumar.website/ \
  --eventType buzzword.phrase \
  --interval 20s \
  --method GET
2024/06/13 23:40:37 foo | Updating manifest
2024/06/13 23:40:37 foo | Starting container
Created object name: foo-httpollersource
Component produces: buzzword.phrase
(base) michalkielkowski@Michas-MacBook-Pro-2 ~ % tmctl create target \
  --name console \
  --from-image gcr.io/triggermesh/triggermesh-console:v0.0.1 \
  --source foo-httpollersource
2024/06/13 23:41:58 foo | Updating manifest
2024/06/13 23:41:58 foo | Starting container
Created object name: console
Subscribed to: foo-httpollersource(buzzword.phrase)
Listening on: http://localhost:57393
(base) michalkielkowski@Michas-MacBook-Pro-2 ~ %
```



Test dla innego endpointa (<https://jsonplaceholder.typicode.com/todos/1>)

Użyte komendy:

1. `tmctl create source httppoller \`

`--endpoint https://https://jsonplaceholder.typicode.com/todos/1/ \`

`--eventType test_endpoint \`

`--interval 5s \`

`--method GET`



Output:

```
michalkielkowski - zsh - 80x24
Component produces: buzzword.phrase
(base) michalkielkowski@Michas-MacBook-Pro-2 ~ % tmctl create target \
  --name console \
  --from-image gcr.io/triggernesh/triggernesh-console:v0.0.1 \
  --source foo-httpollersource
2024/06/13 23:41:58 foo | Updating manifest
2024/06/13 23:41:58 foo | Starting container
-----
Created object name: console
Subscribed to: foo-httpollersource(buzzword.phrase)
listening on: http://localhost:57393
(base) michalkielkowski@Michas-MacBook-Pro-2 ~ % tmctl create source httppoller \
  --endpoint https://jsonplaceholder.typicode.com/todos/1/ \
  --eventType test_endpoint \
  --interval 5s \
  --method GET
2024/06/13 23:47:32 foo | Updating manifest
2024/06/13 23:47:32 foo | Starting container
-----
Created object name: foo-httppollersource
Component produces: test_endpoint
(base) michalkielkowski@Michas-MacBook-Pro-2 ~ %
```

```
michalkielkowski - tmctl watch - 80x24
datacontenttype: application/json
Data,
{
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
}
}
cloudvents.Event
Context Attributes,
specversion: 1.0
type: test_endpoint
source: local.foo-httppollersource
id: 879a1d06-e3be-4ef8-9a0c-d439c1116ce7
time: 2024-06-13T21:47:44.63982304Z
datacontenttype: application/json
Data,
{
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
}
}
```

Teraz sprawdzimy czy połączenie do naszego endpointu zadziała  
(<http://app.com/basic>)

Output naszego endpointu:

```
app.com/basic
Not Secure app.com/basic
Hello from the basic version!
```

Użyte komendy:

1. `tmctl create source httppoller \`

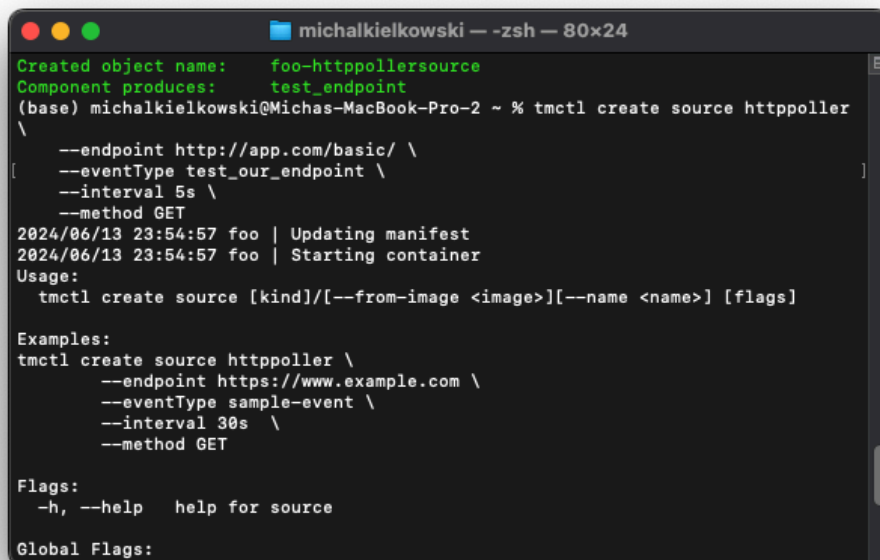
`--endpoint http://app.com/basic/ \`

`--eventType test_our_endpoint \`

`--interval 5s \`

`--method GET`

Output:

A terminal window titled "michalkielkowski - zsh - 80x24" showing the output of the command "tmctl create source httppoller \ --endpoint http://app.com/basic/ \ --eventType test\_our\_endpoint \ --interval 5s \ --method GET". The output shows the creation of an object named "foo-httppollersource" which produces the "test\_endpoint" component. It also shows the manifest being updated and the container starting. The terminal includes usage instructions, examples, and flags for the command.

```
michalkielkowski - zsh - 80x24
Created object name:   foo-httppollersource
Component produces:   test_endpoint
(base) michalkielkowski@Michas-MacBook-Pro-2 ~ % tmctl create source httppoller \
--endpoint http://app.com/basic/ \
--eventType test_our_endpoint \
--interval 5s \
--method GET
2024/06/13 23:54:57 foo | Updating manifest
2024/06/13 23:54:57 foo | Starting container
Usage:
  tmctl create source [kind]/[--from-image <image>][--name <name>] [flags]

Examples:
tmctl create source httppoller \
  --endpoint https://www.example.com \
  --eventType sample-event \
  --interval 30s \
  --method GET

Flags:
  -h, --help    help for source

Global Flags:
```



## Próba na innym komputerze

Aby HTTP Polar mógł dobrze wykorzystać i się poprawnie łączyć z naszym endpoint powinien on być w naszym dockerowym obrazie a nie wywoływany poprzez **tmctl** command line. Wtedy posiadalibyśmy dostęp do naszego ingressu i powinien on działać dobrze jednak komenda **dump** w połączeniu z **kubectl apply** nie działa.

Memory broker implementacja:

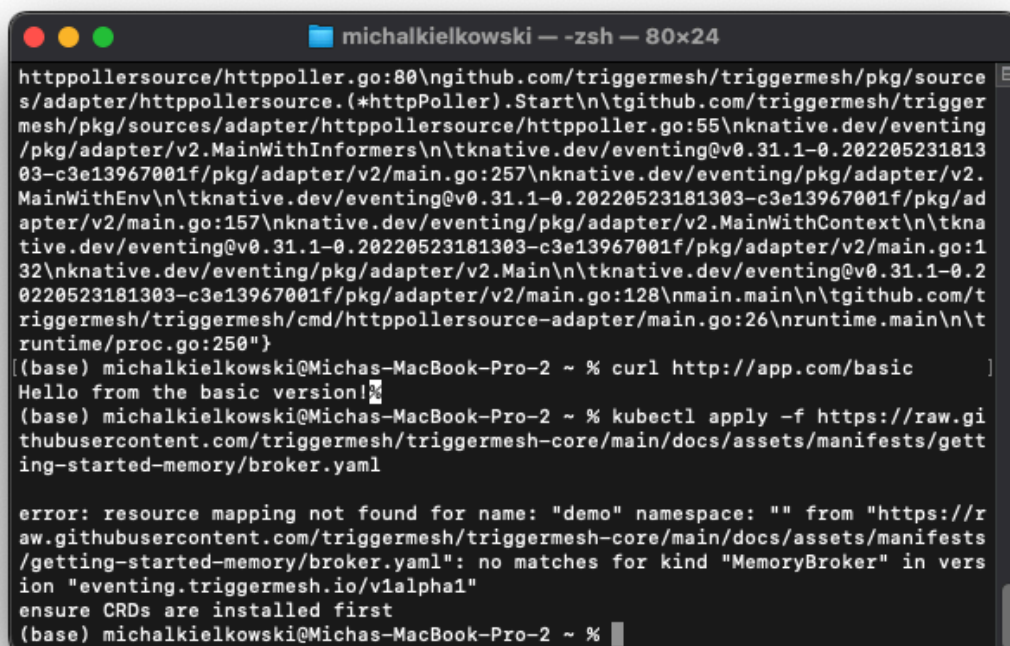
(<https://docs.triggermesh.io/1.27/brokers/memorybroker/>)

Użyte komendy:

1. **kubectl apply -f**

<https://raw.githubusercontent.com/triggermesh/triggermesh-core/main/docs/assets/manifests/getting-started-memory/broker.yaml>

Output:



```
michalkielkowski — zsh — 80x24
httpollersource/httppoller.go:80\ngithub.com/triggermesh/triggermesh/pkg/source
s/adapters/httpollersource.(*httpPoller).Start\n\tgithub.com/triggermesh/trigger
mesh/pkg/sources/adapters/httpollersource/httppoller.go:55\nknative.dev/eventing
/pkg/adapters/v2.MainWithInformer\n\tknative.dev/eventing@v0.31.1-0.202205231813
03-c3e13967001f/pkg/adapters/v2/main.go:257\nknative.dev/eventing/pkg/adapters/v2.
MainWithEnv\n\tknative.dev/eventing@v0.31.1-0.20220523181303-c3e13967001f/pkg/ad
apters/v2/main.go:157\nknative.dev/eventing/pkg/adapters/v2.MainWithContext\n\tkna
tive.dev/eventing@v0.31.1-0.20220523181303-c3e13967001f/pkg/adapters/v2/main.go:1
32\nknative.dev/eventing/pkg/adapters/v2.Main\n\tknative.dev/eventing@v0.31.1-0.2
0220523181303-c3e13967001f/pkg/adapters/v2/main.go:128\nmain.main\n\tgithub.com/t
riggermesh/triggermesh/cmd/httpollersource-adapters/main.go:26\nruntime.main\n\t
runtime/proc.go:250"}
(base) michalkielkowski@Michas-MacBook-Pro-2 ~ % curl http://app.com/basic
Hello from the basic version!
(base) michalkielkowski@Michas-MacBook-Pro-2 ~ % kubectl apply -f https://raw.gi
thubusercontent.com/triggermesh/triggermesh-core/main/docs/assets/manifests/gett
ing-started-memory/broker.yaml

error: resource mapping not found for name: "demo" namespace: "" from "https://r
aw.githubusercontent.com/triggermesh/triggermesh-core/main/docs/assets/manifests
/getting-started-memory/broker.yaml": no matches for kind "MemoryBroker" in vers
ion "eventing.triggermesh.io/v1alpha1"
ensure CRDs are installed first
(base) michalkielkowski@Michas-MacBook-Pro-2 ~ %
```

Z tmctl to Kubernetes

(<https://docs.triggermesh.io/1.27/get-started/moving-from-dev-to-K8s/>)

Użyte komendy:

```
trigger_brooker — zsh — 80x24

ref:
  apiVersion: eventing.triggermesh.io/v1alpha1
  kind: RedisBroker
  name: foo
---
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  labels:
    triggermesh.io/context: foo
    triggermesh.io/role: target
  name: console
spec:
  template:
    spec:
      containers:
      - env: []
        image: gcr.io/triggermesh/triggermesh-console:v0.0.1
        name: user-container
---
apiVersion: eventing.triggermesh.io/v1alpha1
kind: Trigger
metadata:
  labels:
```

```
trigger_brooker — zsh — 80x24

[(base) michalkielkowski@Michas-MacBook-Pro-2 trigger_brooker % tmctl dump > foo.]
yaml
[(base) michalkielkowski@Michas-MacBook-Pro-2 trigger_brooker % cat foo.yaml ]

---
apiVersion: eventing.triggermesh.io/v1alpha1
kind: RedisBroker
metadata:
  labels:
    triggermesh.io/context: foo
  name: foo
---
apiVersion: sources.triggermesh.io/v1alpha1
kind: HTTPPollerSource
metadata:
  labels:
    triggermesh.io/context: foo
  name: foo-httppollersource
spec:
  endpoint: http://app.com/basic/
  eventType: test_our_endpoint
  interval: 5s
  method: GET
  sink:
    ref:
```

```
trigger_broker — zsh — 80x24

labels:
  triggermesh.io/context: foo
  name: foo-trigger-a73dcb65
spec:
  broker:
    group: eventing.triggermesh.io
    kind: RedisBroker
    name: foo
  filters:
  - exact:
    type: buzzword.phrase
  target:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: console

[(base) michalkielkowski@Michas-MacBook-Pro-2 trigger_broker % kubectl apply -f ]
foo.yaml
httpollersource.sources.triggermesh.io/foo-httpollersource created
resource mapping not found for name: "foo" namespace: "" from "foo.yaml": no mat
ches for kind "RedisBroker" in version "eventing.triggermesh.io/v1alpha1"
ensure CRDs are installed first
resource mapping not found for name: "console" namespace: "" from "foo.yaml": no
```

```
trigger_broker — zsh — 80x24

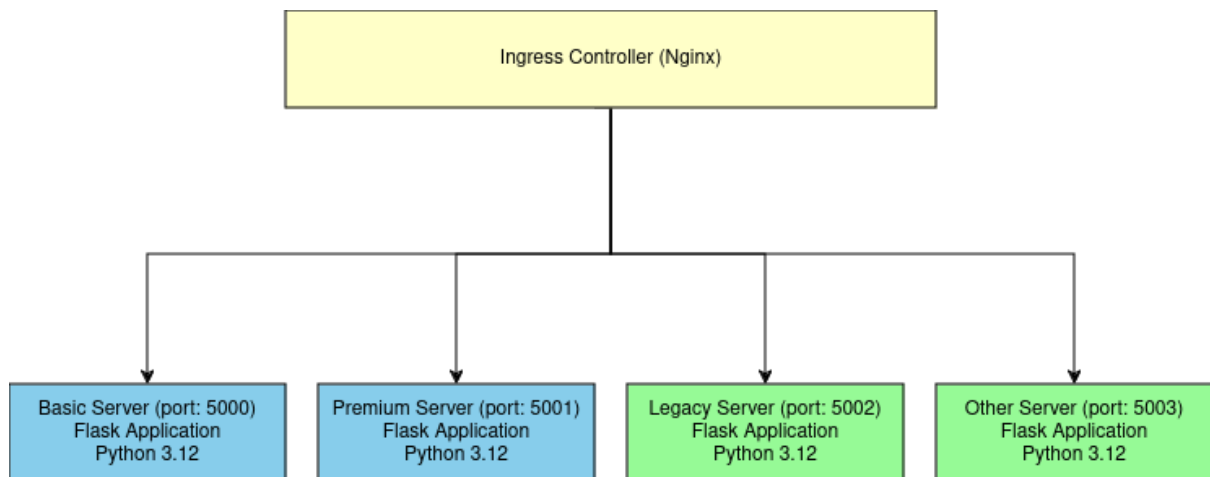
name: foo
filters:
  - exact:
    type: buzzword.phrase
  target:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: console

[(base) michalkielkowski@Michas-MacBook-Pro-2 trigger_broker % kubectl apply -f ]
foo.yaml
httpollersource.sources.triggermesh.io/foo-httpollersource created
resource mapping not found for name: "foo" namespace: "" from "foo.yaml": no mat
ches for kind "RedisBroker" in version "eventing.triggermesh.io/v1alpha1"
ensure CRDs are installed first
resource mapping not found for name: "console" namespace: "" from "foo.yaml": no
matches for kind "Service" in version "serving.knative.dev/v1"
ensure CRDs are installed first
resource mapping not found for name: "foo-trigger-a73dcb65" namespace: "" from "
foo.yaml": no matches for kind "Trigger" in version "eventing.triggermesh.io/v1a
lpha1"
ensure CRDs are installed first
(base) michalkielkowski@Michas-MacBook-Pro-2 trigger_broker %
```

## Schematy

Połączenia między serwisami a Ingress Controller (Nginx):

- Każdy serwer (Basic Server, Premium Server, Legacy Server, Other Server) nasłuchuje na określonym porcie i jest skonfigurowany do odbierania żądań HTTP.
- Ingress Controller (Nginx) działa jako punkt wejścia dla żądań HTTP, mapując URL-e na odpowiednie serwisy. Dlatego połączenia są skierowane od Ingress Controller do serwisów.

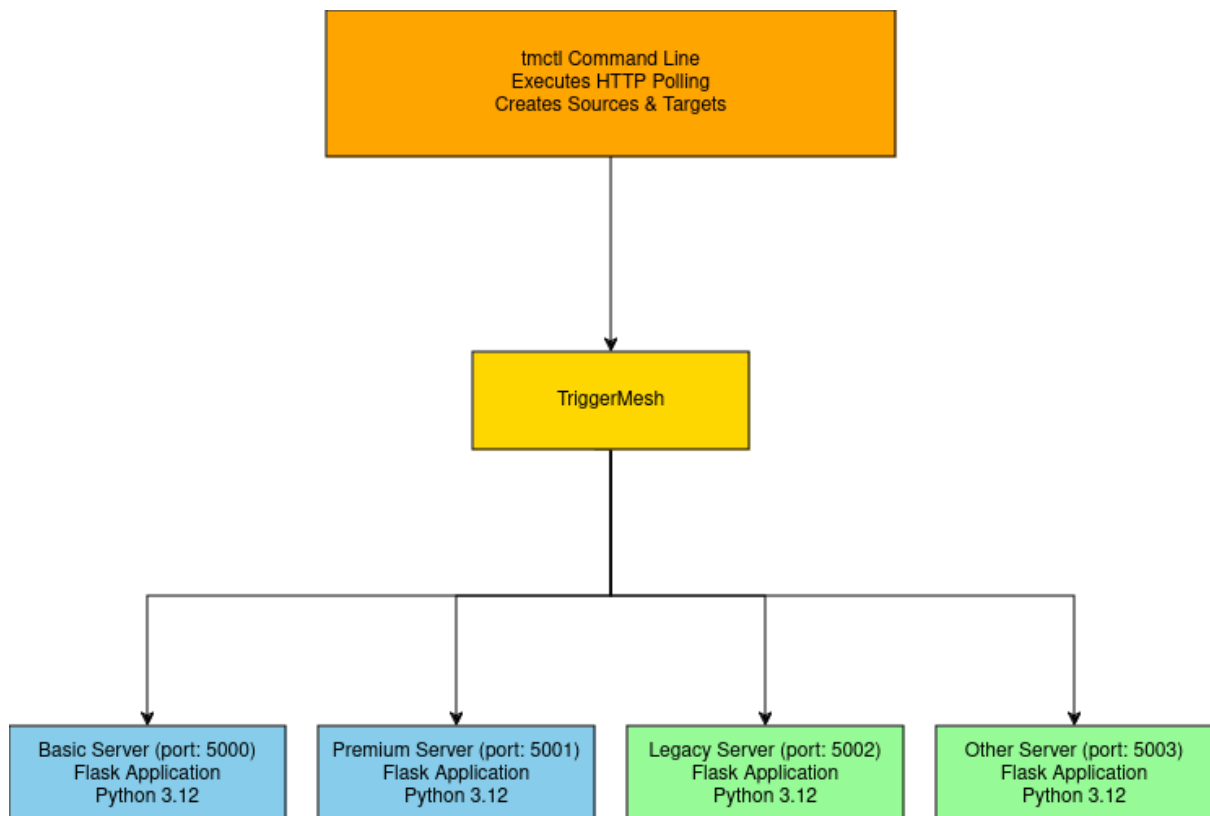


Połączenia między **tmctl Command Line** a TriggerMesh:

- **tmctl Command Line** jest narzędziem wiersza poleceń używanym do konfigurowania TriggerMesh, tworzenia źródeł i celów. Dlatego połączenia są skierowane od **tmctl Command Line** do TriggerMesh.

Połączenia między TriggerMesh a serwisami:

- TriggerMesh zarządza przepływem zdarzeń między różnymi serwisami i systemami. Dlatego połączenia są skierowane od TriggerMesh do serwisów, aby umożliwić przetwarzanie i routowanie zdarzeń.





Połączenia między Kubernetes Cluster a serwisami:

- Kubernetes Cluster zarządza wdrażaniem serwisów, zarządzaniem podami i skalowaniem. Dlatego połączenia są skierowane od Kubernetes Cluster do serwisów, aby zapewnić zarządzanie i skalowalność.

