Uniwersytet: Uniwersytet Śląski w Katowicach		Wydział: Wydział Nauk Ścisłych I Technicznych			
		Instytut: Fizyki			
		Rok:	2	Semestr:	4
Kierunek:	Informatyka Stosowana				
Przedmiot:	Systemy wbudowane				
Prowadzący:	Przemysław Raczyński				
Tytuł ćwiczenia:	Klaster obliczeniowy przy użyciu Raspberry Pi 2			Nr ćwiczenia	-
Sprawozdanie wykonał:	Błażej Pietryja, Tymon Pawełczyk, Marcin Partyka				

1. Wstęp teoretyczny

1.1 Klastry obliczeniowe

Klastry obliczeniowe można zastosować na kilka różnych sposobów. Jedne z podstawowych klas budowy klastrów to:

- Klastry wydajnościowe: pracujące jako zespół komputerów, z których każdy wykonuje własne zadania obliczeniowe. Celem ich budowy jest zwiększenie mocy obliczeniowej, w sytuacji, kiedy różne komputery w klastrze pracują nad odrębnymi podzadaniami pojedynczego dużego zadania obliczeniowego. Wiele obecnych superkomputerów działa na tej zasadzie.

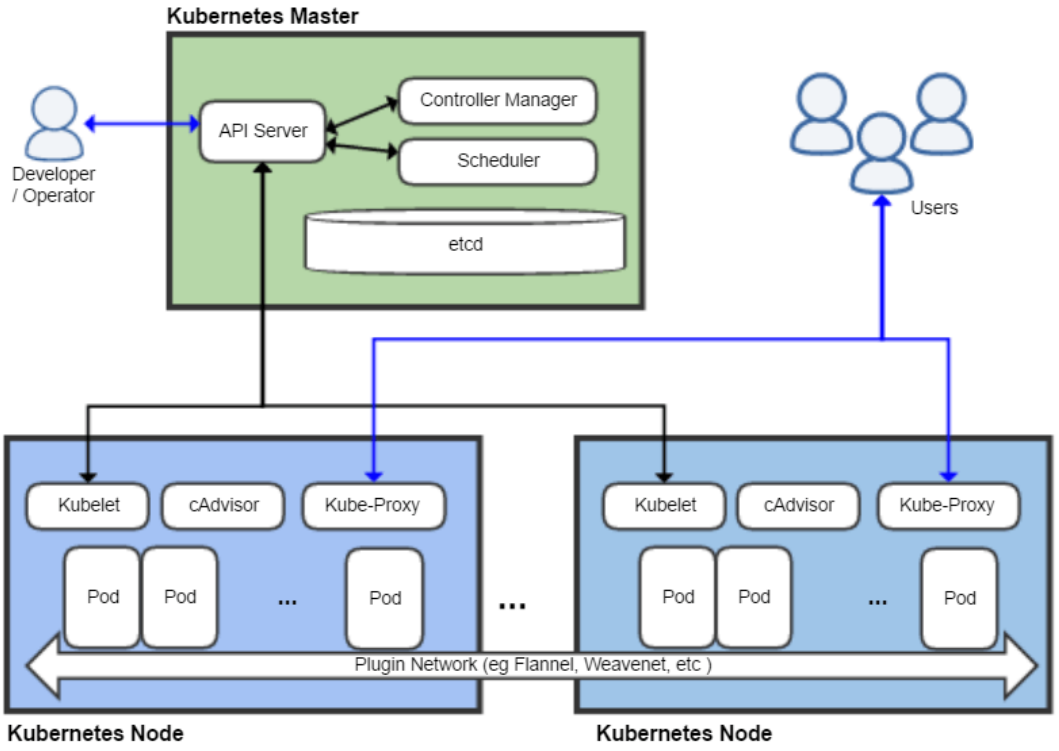
- Klastry niezawodnościowe: pracujące jako zespół komputerów wykonujące każdy swoje zadanie. W razie awarii jednego z węzłów, następuje automatyczne przejęcie jego funkcji przez inne węzły.

- Klastry równoważenia obciążenia: pracujące jako zespół komputerów, z których każdy wykonuje własne zadanie z puli zadań skierowanych do całego klastra. W takiej sytuacji pojedynczy komputer może wykonywać niezależne zadanie lub współpracować z kilkoma innymi węzłami klastra wykonując podzadanie większego zadania obliczeniowego. Klastry równoważenia obciążenia mogą być traktowane jako pierwowzór, a obecnie także jako częsty element składowy, systemów gridowych.

1.2 Kubernetes

Kubernetes to platforma do zarządzania, automatyzacji i skalowania aplikacji kontenerowych. Dzięki tej technologii możemy połączyć benefity trzech powyższych klas klastrów i korzystać z nich w wygodny i łatwo dostępny sposób.

W naszym projekcie używamy odmiany kubernetesa pod tytułem K3s. Jest to lekka wersja kubernetesa stworzona przez Rancher Labs. Cechuje się małym rozmiarem oraz niewielkimi wymaganiami sprzętowymi, co jest idealne dla naszego zastosowania.
Nasze rozwiązanie stosuje klasyczny model z użyciem master noda. Wszystkie zapytania oraz komendy przesyłam do mastera, który zarządza nodami pracowników.



2. Przedstawienie projektu oraz kod.

Do mastera łączę się za pomocą ssh:

```
PS C:\Users\MSI> ssh pi@192.168.18.215 -p 2219
pi@192.168.18.215's password:
Linux raspberrypi 5.15.84-v7+ #1613 SMP Thu Jan 5 11:59:48 GMT 2023 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Jun 17 17:25:39 2023 from 192.168.18.2
pi@raspberrypi:~ $
```

Do zarządzania kubernetesem używam polecenia kubectl (kubernetes control)

Żeby zobaczyć dostępne nody w klastrze: kubectl get nodes

```
root@raspberrypi:/home/pi# kubectl get nodes
NAME          STATUS   ROLES                  AGE   VERSION
raspberrypi   Ready    control-plane,master   66d   v1.26.3+k3s1
worker1       Ready    <none>                 39d   v1.26.4+k3s1
worker2       Ready    <none>                 39d   v1.26.4+k3s1
worker3       Ready    <none>                 39d   v1.26.4+k3s1
root@raspberrypi:/home/pi#
```

2.1 Kubernetes Job i paralelizacja zadań

Głównym celem naszego projektu jest zrównoleglenie zadań i pokazanie jak kubernetes rozdziela zasoby wewnątrz systemu.

Jako system rozdzielania zasobów wybraliśmy wbudowany system kubernetesa, ponieważ jest on dobrze zoptymalizowany i w pełni spełnia nasze wymagania. System obserwuje aktualne zużycie zasobów poszczególnych nodów i rozdziela zadania tym które są aktualnie najmniej “zajęte”.

Przedstawienie optymalizacji i paralelizacji zadań wykonam używając Kubernetes Job, oraz problemu obliczania liczby Pi metodą monte carlo.
Job jest to proces w strukturze kubernetesa który wykonuje dane zadanie i kończy się.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: python-monte
  labels:
    jobgroup: monte
spec:
  completions: 20
  parallelism: 3
  template:
    metadata:
      name: monte
      labels:
        jobgroup: monte
    spec:
      containers:
      - name: c
        image: nioxsh/cluster:apka
        command: ["python3", "-c", "import random; print(4*sum(1 for i in range(1000000) if random.random()**2+random.random()**2<=1) / 1000000)"]
      restartPolicy: Never
```

Job możemy wykonać tworząc plik manifestu z rozszerzeniem .yaml
Najważniejsze elementy pliku to:

kind: Job - ustalenie typu deploymentu kubernetesa
completions: 20 - liczba wykonań programu
parallelism: 3 - ile wykona się równolegle, mamy 3 nody pracowników więc dałem 3
command: tutaj mamy skrypt w pythonie obliczający liczbę Pi

print(4*sum(1 for i in range(1000000) if random.random()**2+random.random()**2<=1) / 1000000)

Poleceniem kubectl get pods -o wide, można zobaczyć pracujące joby i które nody się nimi zajmują.

```
root@raspberrypi:/home/pi/job_try# kubectl get pods -o wide
NAME                                    READY   STATUS      RESTARTS      AGE   IP            NODE      NOMINATED NODE   READINESS GATES
hello-virt-deployment-54dd69fdc-nmt45   1/1     Running     2 (80m ago)   45h   10.42.3.80    worker3   <none>           <none>
site-5cf586bf75-6jxn4                   1/1     Running     1 (80m ago)   16h   10.42.3.81    worker3   <none>           <none>
site-deployment-5484f69fdc-8l6x5        1/1     Running     1 (80m ago)   16h   10.42.3.82    worker3   <none>           <none>
montecarlo-deployment-6445f99768-6ndm9  1/1     Running     1 (80m ago)   14h   10.42.2.122   worker2   <none>           <none>
site-deployment-5484f69fdc-cx2ll        1/1     Running     1 (80m ago)   16h   10.42.2.124   worker2   <none>           <none>
site-5cf586bf75-pht2d                   1/1     Running     1 (80m ago)   16h   10.42.2.123   worker2   <none>           <none>
site-5cf586bf75-qp2sz                   1/1     Running     1 (80m ago)   16h   10.42.4.72    worker1   <none>           <none>
hello-virt-deployment-54dd69fdc-lq6rk   1/1     Running     2 (80m ago)   45h   10.42.4.73    worker1   <none>           <none>
montecarlo-deployment-6445f99768-psgd2  1/1     Running     1 (80m ago)   14h   10.42.4.71    worker1   <none>           <none>
site-deployment-5484f69fdc-h46cs        1/1     Running     1 (80m ago)   16h   10.42.4.70    worker1   <none>           <none>
python-monte-4slf7                      0/1     Completed   0             47s   10.42.2.126   worker2   <none>           <none>
python-monte-j5hrm                      0/1     Completed   0             47s   10.42.3.83    worker3   <none>           <none>
python-monte-w2v5s                      0/1     Completed   0             47s   10.42.2.125   worker2   <none>           <none>
python-monte-px4w5                      0/1     Completed   0             28s   10.42.2.127   worker2   <none>           <none>
python-monte-fmrbx                      0/1     Completed   0             28s   10.42.2.128   worker2   <none>           <none>
python-monte-qx7dv                      0/1     Completed   0             28s   10.42.3.84    worker3   <none>           <none>
python-monte-2xp7b                      1/1     Running     0             10s   10.42.2.129   worker2   <none>           <none>
python-monte-dvhgr                      1/1     Running     0             10s   10.42.4.76    worker1   <none>           <none>
python-monte-jq5rm                      1/1     Running     0             10s   10.42.3.85    worker3   <none>           <none>
root@raspberrypi:/home/pi/job_try#
```

Polecenie kubectl logs pokazuje wyniki każdego procesu

```
root@raspberrypi:/home/pi/job_try# kubectl logs -f -l jobgroup=monte --max-log-requests=20
I0618 13:01:37.238020    7535 request.go:690] Waited for 1.194592565s due to client-side throttling, not prior
0.1:6443/api/v1/namespaces/default/pods/python-monte-kskft/log?container=c&follow=true&tailLines=10
3.139896
3.142564
3.139732
3.140688
3.142936
3.14174
3.141856
3.139884
3.139904
3.140508
3.14098
3.141564
3.141208
3.14166
3.143292
3.141036
3.142784
3.140108
3.141756
3.140464
root@raspberrypi:/home/pi/job_try#
```

Wyniki przesyłam do skryptu który oblicza ich średnią.

```
#!/bin/bash
# skrypt do obliczania sredniej z danych z klastra
ilosc=20
suma=0
while read line
do
        suma=$(bc -l <<<"${suma}+${line}")
done
srednia=$(bc -l <<<"${suma}/${ilosc}")
echo $srednia
```

```
root@raspberrypi:/home/pi/job_try# kubectl logs -f -l jobgroup=monte --max-log-requests=20 | ./script.sh
I0618 13:14:23.592083    7985 request.go:690] Waited for 1.196658825s due to client-side throttling, not priority
0.1:6443/api/v1/namespaces/default/pods/python-monte-6bzbj/log?container=c&follow=true&tailLines=10
3.14122800000000000000
root@raspberrypi:/home/pi/job_try#
```

2.2 Kubernetes deployment i nginx website

Możemy teraz przejść do stworzenia strony internetowej
W kubernetesie aplikacje nazywane są deployment. Można je tworzyć na różne sposoby.

Jako pierwsze stworzę aplikację która stawia domyślną stronę html z użyciem silnika nginx.

```
root@raspberrypi:/home/pi# kubectl create deployment site-deployment --image=nginx --replicas=3 --port=80
deployment.apps/site-deployment created
root@raspberrypi:/home/pi#
```

kubectl - prefix kubernetesa
create - komenda stworzenia
deployment - typ aplikacji
site-deployment - nazwa deployment'u
–image=nginx - obraz silnika nginx, który będzie pobrany ze strony Dockera
–replicas=3 - ustawiam liczbę replik które będą wystawione w klastrze
–port=80 - port aplikacji

```
root@raspberrypi:/home/pi# kubectl get deployments
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
hello-virt-deployment   2/2     2            2           26h
montecarlo-deployment   2/2     2            2           24h
site-deployment         3/3     3            3           3m33s
root@raspberrypi:/home/pi#
```

Komendą kubectl get deployments, możemy zobaczyć nasze aktywne deployment'y. Widać site-deployment, które właśnie stworzyłem oraz inne które stworzyłem wcześniej.

```
root@raspberrypi:/home/pi# kubectl get pods -o wide
NAME                                    READY   STATUS    RESTARTS      AGE     IP           NODE      NOMINATED NODE   READINESS GATES
hello-virt-deployment-54dd69fdc-lq6rk   1/1     Running   1 (58m ago)   26h     10.42.4.49   worker1   <none>           <none>
montecarlo-deployment-6445f99768-qscsv  1/1     Running   1 (58m ago)   24h     10.42.2.89   worker2   <none>           <none>
montecarlo-deployment-6445f99768-7h24k  1/1     Running   1 (58m ago)   24h     10.42.3.55   worker3   <none>           <none>
hello-virt-deployment-54dd69fdc-nmt45   1/1     Running   1 (58m ago)   26h     10.42.3.56   worker3   <none>           <none>
site-deployment-5465c9d6d7-97jpd        1/1     Running   0             6m11s   10.42.2.90   worker2   <none>           <none>
site-deployment-5465c9d6d7-vcm6r        1/1     Running   0             6m10s   10.42.4.50   worker1   <none>           <none>
site-deployment-5465c9d6d7-znq24        1/1     Running   0             6m10s   10.42.3.57   worker3   <none>           <none>
root@raspberrypi:/home/pi#
```

Komenda kubectl get pods -o wide, pokazuje rozdział replik oraz które nody się czym aktualnie zajmują.
Żeby mieć dostęp do deployment'u spoza sieci kubernetesa trzeba stworzyć serwis który otworzy deployment na świat

```
root@raspberrypi:/home/pi# kubectl expose deployment site-deployment --type="NodePort" --port=80
service/site-deployment exposed
root@raspberrypi:/home/pi#
```

```
root@raspberrypi:/home/pi# kubectl get services
NAME                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes            ClusterIP   10.43.0.1       <none>        443/TCP          66d
nginx-nodeport        NodePort    10.43.65.75     <none>        80:31111/TCP     38d
nginx-service         NodePort    10.43.38.135    <none>        80:30666/TCP     38d
hello-virt-exposed    ClusterIP   10.43.14.106    <none>        5000/TCP         17d
hello-virt-exposed2   ClusterIP   10.43.195.214   10.10.10.2    5000/TCP         17d
montecarlo-deployment NodePort    10.43.208.96    <none>        5000:31601/TCP   27h
hello-virt-service    NodePort    10.43.162.168   <none>        5000:32113/TCP   26h
montecarlo-service    NodePort    10.43.109.215   <none>        5000:32115/TCP   26h
montecarlo-out        NodePort    10.43.233.250   <none>        5000:31172/TCP   26h
site-deployment       NodePort    10.43.107.229   <none>        80:30293/TCP     3m52s
root@raspberrypi:/home/pi#
```

Można teraz zobaczyć stronę używając komendę curl 192.168.18.215:30293

```
root@raspberrypi:/home/pi# curl 192.168.18.215:30293
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
root@raspberrypi:/home/pi#
```

2.3 Python flask website

Następnym przykładem będzie strona internetowa w języku Python używająca biblioteki Flask.

Aplikacja Flask

```python
from flask import Flask
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

class HelloWorld(Resource):
    def get(self):
        return "Prosimy o 5"

api.add_resource(HelloWorld, '/')

if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

Deployment aplikacji

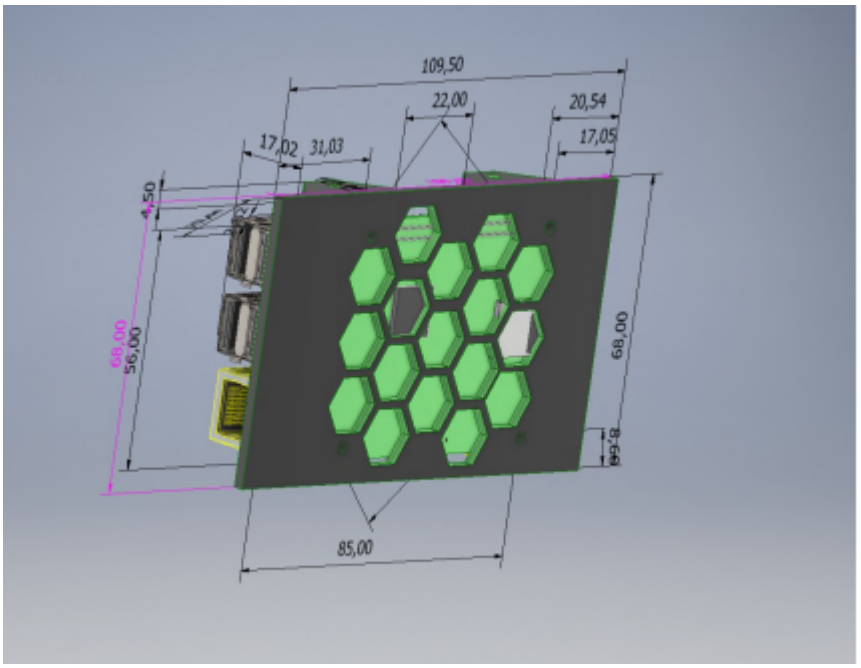
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-virt-deployment
  labels:
    app: hello-virt
spec:
  replicas: 2
  selector:
    matchLabels:
      app: hello-virt-label
  template:
    metadata:
      labels:
        app: hello-virt-label
    spec:
      containers:
      - name: hello-virt-container
        image: nioxsh/cluster:hello2
        ports:
        - containerPort: 5000
```

Serwis aplikacji

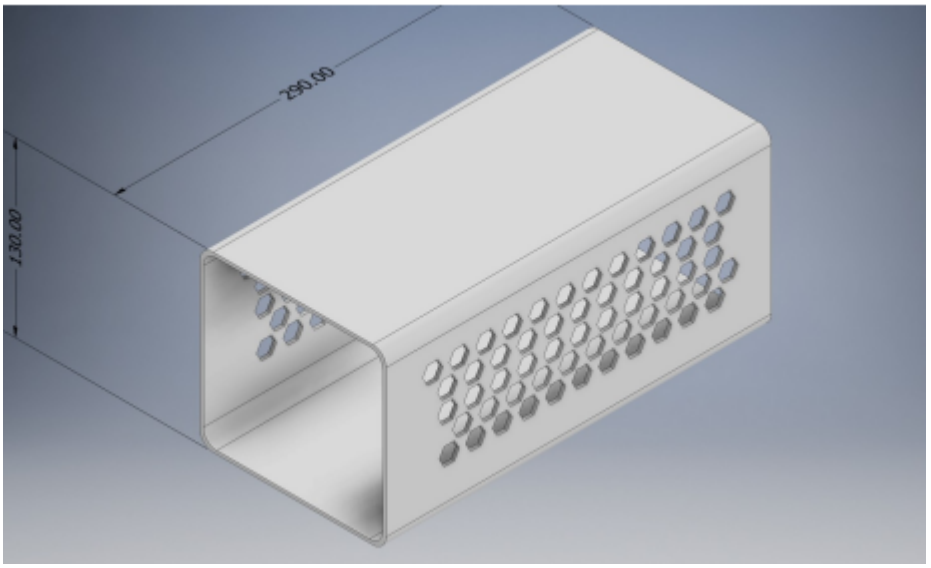
```
apiVersion: v1
kind: Service
metadata:
  name: hello-virt-service
  labels:
    app: hello-virt-label
spec:
  type: NodePort
  ports:
  - port: 5000
    targetPort: 5000
    nodePort: 32113
    protocol: TCP
  selector:
    app: hello-virt-label
```

3.Obudowa

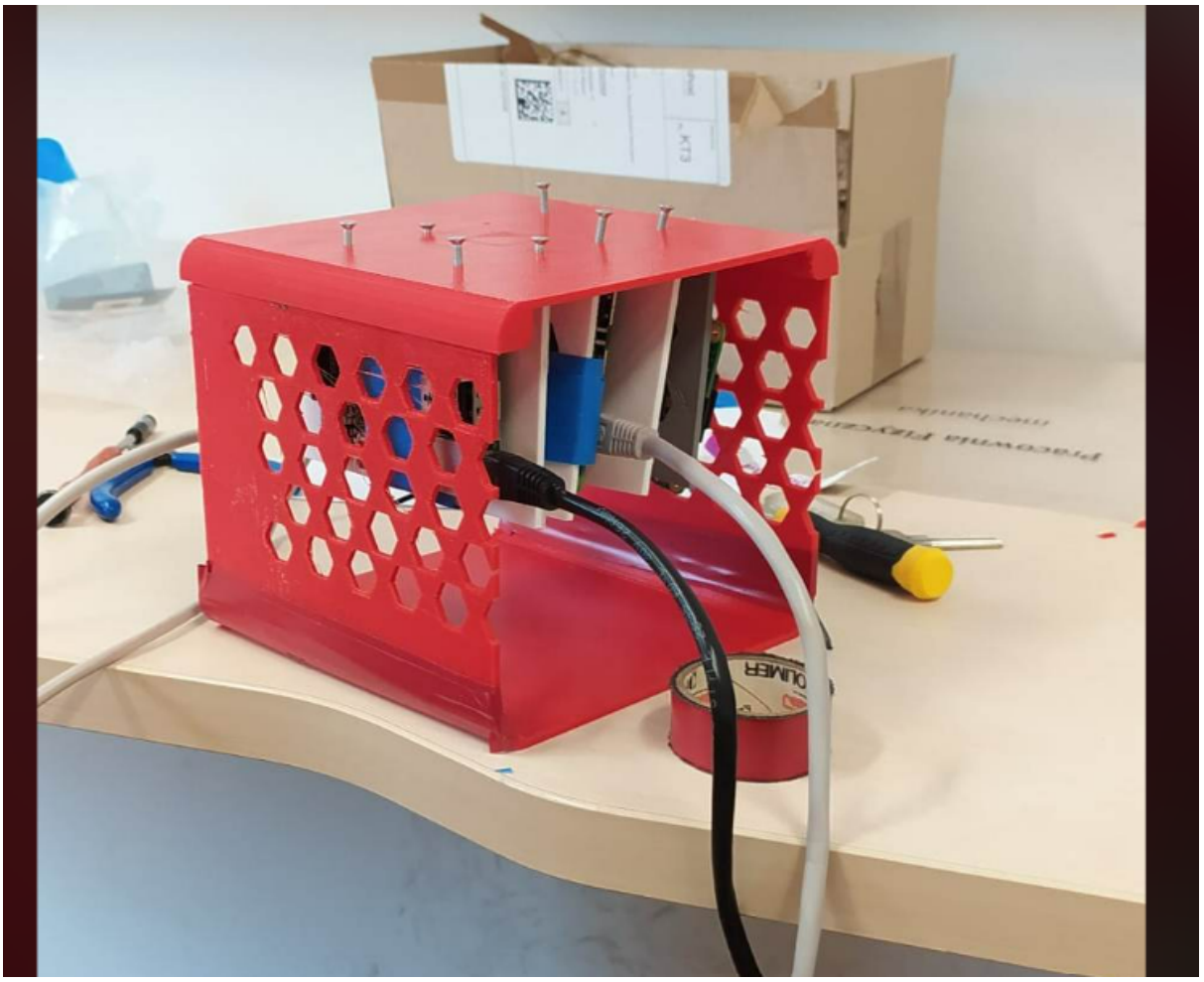
Każde Raspberry zamontowane jest do obudowy na płytce:



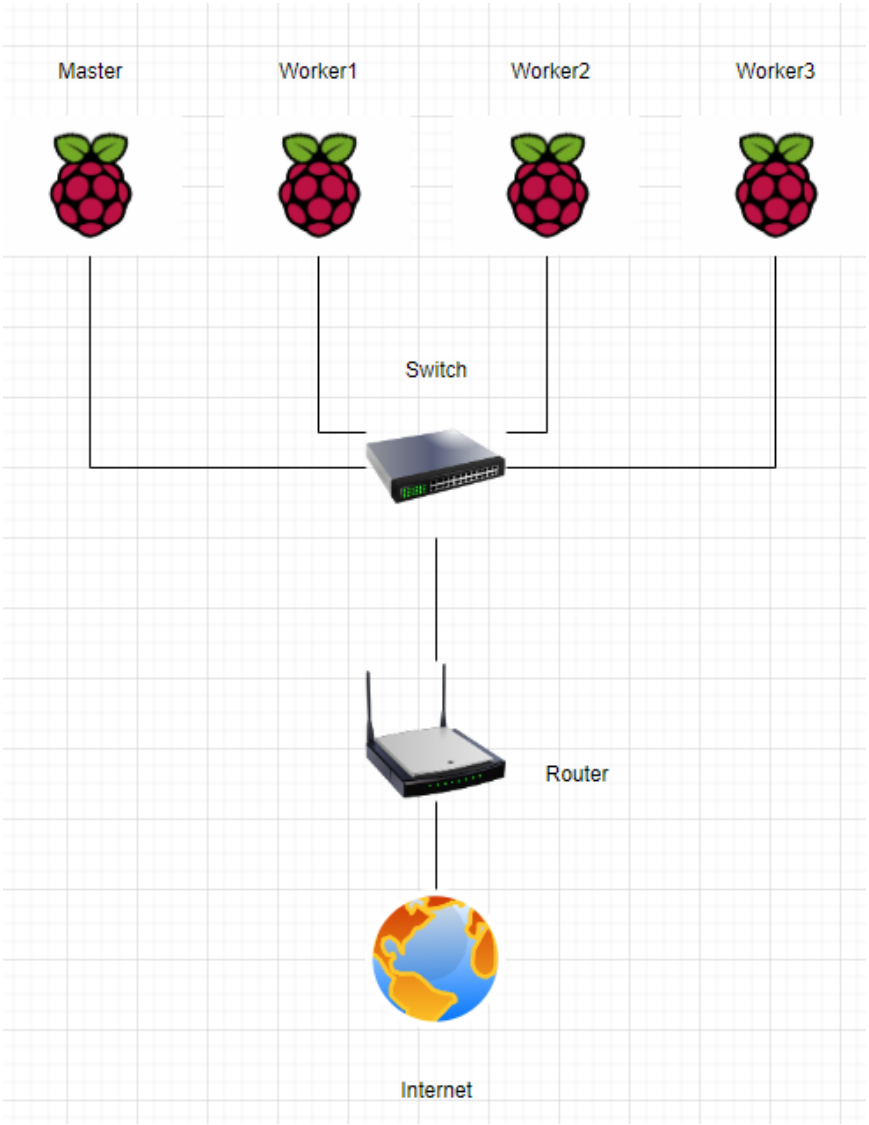
Wieczko z obudowy jest zdejmowane co ułatwia montaż oraz ewentualne prace serwisowe. Raspberry przymocowane są do wieczka za pomocą płytek montażowych. Półotwarta konstrukcja oraz ażurowe ścianki zapewniają dobry przepływ powietrza.



Modele 3d zaprojektowane w programie AutoDesk Inventor, sliceowane przez program Cura.
Elementy zostały wydrukowane na drukarce Creality Ender 3 z filamentu PLA. Grubość warstwy 0,2 mm, infill 50% pattern grid. Temperatura stołu 50ºC, temperatura ekstrudera 225ºC



4.Schemat fizycznych połączeń



Użyte elementy:
4x - Raspberry Pi 2B v1.1
Switch Tenda 8
Router Planet
Kable sieciowe

4. Wnioski i krytyczne podejście

Możliwość paralelizacji zadań zwiększa wydajność obliczeń w znaczący sposób. Według naszych obserwacji, wydajność klastra, jest niewiele gorsza niż obliczenia na laptopie średniej półki. Laptop wykonał program w około 6 sekund dla 1 miliona próbek, a nasz klaster, dla tej samej liczby próbek, kończy zadanie w około 8 sekund. Uwzględniając parametry raspberry pi 2b, jest to dobry wynik.

Co nam się nie udało to wystawienie interaktywnej strony internetowej która od razu przy starcie strony, wykonuje podobne obliczenia. Moim zdaniem algorytm trwa na tyle długo, że klaster jest traktowany jako nieaktywny i nie wyświetla strony.

Podsumowując, klastry obliczeniowe, nawet w tak małej skali, są w stanie wykonywać istotne obliczenia. System optymalizacji zasobów i równoważenia obciążenia platformy Kubernetes daje nowe dla nas możliwości pod względem wydajnego przetwarzania danych i/lub obliczania skomplikowanych problemów matematyczno fizycznych.

Oczywiście nasz klaster nie zbliża się nawet do możliwości obliczeniowych klastrów używanych w firmach lub placówkach badawczych jednak koncept zrównoleglania pozostaje ten sam.