Bitcoin Monitor

About the project:

Project Bitcoin Monitor is a microservices implementation of an application that can provide a real time monitor for the Bitcoin values. It is implemented in Golang using the **net/http** library for implementing the web server connectivity. By using a Go routine that is running in parallel with the main function, the Bitcoin value is automatically update at specific time interval, default of 10 seconds, and based on the iteration value (default is 3 times), it calculate the average value. The application is deployed as a container, during the build process the code is compiled and the tagged image is pushed in a private ACR Registry. Also, the image will automatically expose the port 8080.

Prerequisites:

- Active Azure Subscription for using the managed AKS
- Azure CLI to interact with the Subscription resources
- Kubectl – used for deploying the Kubernetes object
- Docker – For building and pushing the application to public/private registry

Installation:

We deploy the Azure Infrastructure with Terraform. Using the latest version of AKS cluster with a single Node of type System and a VM SKU of Standard DS2_V2. On the networking layer we choose the Azure CNI, a System Assigned Identity and integration with Azure Entra for access to resources.

An Output block will provide access to the kubeconfig configuration in order to interact with with the managed AKS cluster through kubectl binary.

With Azure CLI we'll attach the Azure Container Registry to our AKS Cluster. This command will actually provide the acr_pull Role to the Identity of the cluster over the Container Registry, hence the authentication of the containerd process over the ACR will be done directly over the Managed Identity.

For public exposing the services, we will use a Ingress Controller, in current setup we'll deploy an Nginx Ingress Controller. This will assure the routing of the external requests to internal resources (Pods) inside the cluster. Installation of Nginx Ingress Controller is made with Helm, a cloud native package manager.

As we use the community version of Nginx, we need to add the repository that host the components as follows:

helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

Installation of the Nginx IC is created in minimal way, we only add the annotation for changing the Health **Endopoint** of the Load Balancer to **/healthz**. This is because of an architectural change that modified the health probe from **TCP** to **HTTP**.

Once we installed the Nginx Ingress Controller, it will automatically create a Load Balancer rules for inbound traffic, also it will automatically use the Frontend IP address of the underlay Load Balancer service.

We deployed controllers in different namespaces as follows:

- Ingress, Deployment and ClusterIP ServiceA in **servicea** namespace.
- Ingress, Deployment and ClusterIP ServiceB in **serviceb** namespace.

```
servicea      service-a                          1/1     1            1           28m
serviceb      service-b                          1/1     1            1           72m
ovidiu@SandboxHost-638365986924924226:~/bitcoin$ k get ingress -A
NAMESPACE   NAME        CLASS   HOSTS              ADDRESS           PORTS   AGE
servicea    bitcoin     nginx   azbastion.cloud    20.123.228.223    80      26m
serviceb    serviceb    nginx   azbastion.cloud    20.123.228.223    80      63m
servicea      service-a                  ClusterIP      10.0.150.22     <none>          8080/TCP                 29m
serviceb      service-b                  ClusterIP      10.0.152.163    <none>          8080/TCP                 73m
```

Our Service A deployment consist of a single replica of our application (btcmon) and exposed with a Cluster IP service. This Service name will be used in the Ingress **configuration.**

kubectl create deployment service-a --image=sslovidiu.azurecr.io/btcmon --port=8080 --replicas=1 -n servicea

kubectl expose deployment service-a --type=ClusterIP --port=8080 --target-port=8080 -n servicea

We used a custom domain configured in Ingress Host section and also created as a zone in Azure DNS.

We will deploy the Service B in a different namespace (serviceb)

kubectl create deployment service-b --image=sslovidiu.azurecr.io/btcmon --replicas=1 --port=8080 -n serviceb

kubectl expose deployment service-b --type=ClusterIP --port=8080 --target-port=8080 -n serviceb

Will add the new endpoint in the Ingress configuration file with a different path (/health). It will use as an endpoint the new Service-B deployment.

```
PS C:\dev\bitcoin> kubectl get pods
NAME                                        READY   STATUS    RESTARTS   AGE
ingress-nginx-controller-cd7b768cc-w9lk8    1/1     Running   0          13h
nginx                                       1/1     Running   0          3h4m
service-a-7fbf6bb868-l4chd                  1/1     Running   0          13h
service-b-685c745b87-hvzfv                  1/1     Running   0          166m
```

Securing the traffic between services:

```
NAME                                        READY   STATUS    RESTARTS   AGE    IP            NODE                                NOMINATED NODE   READINESS GATES
ingress-nginx-controller-cd7b768cc-w9lk8    1/1     Running   0          13h    10.224.0.27   aks-system-35669538-vmss000001   <none>           <none>
nginx                                       1/1     Running   0          3h6m   10.224.0.23   aks-system-35669538-vmss000001   <none>           <none>
service-a-7fbf6bb868-l4chd                  1/1     Running   0          13h    10.224.0.14   aks-system-35669538-vmss000001   <none>           <none>
service-b-685c745b87-hvzfv                  1/1     Running   0          168m   10.224.0.29   aks-system-35669538-vmss000001   <none>           <none>
```

First, we'll test the functionality of the Pod's endpoint by opening a shell session on Service A Pod:

kubectl exec -it service-a-7fbf6bb868-l4chd – bash

A curl on the Service B Pod's IP address will show that the application is reachable. Also the other way around, from Service B towards Service A.

```
root@service-a-7fbf6bb868-l4chd:/app# curl 10.224.0.29:8080
<html>
    <meta http-equiv="refresh" content="10">
    <head>
        <title>Bitcoin Monitor </title>
    </head>
    <body>
            Local Time:         2023-11-26 11:48:00.691178521 +0000 UTC
        Bitcoin Realtime Value       37341.37               Average value:         37410.24        </body>        </html>root@service-a-7fbf6bb868-l4chd:/app#
```

To isolate the traffic between two namespaces (servicea and serviceb), we apply the following Network Policy in **default** namespace

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-egress
spec:
  podSelector: {}
  policyTypes:
  - Egress
  - Ingress
```

To allow the traffic from Ingress controller in the **default** namespace towards the backend in **servicea** and **serviceb** namespaces we will apply the following network policy:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: access-nginx
spec:
  podSelector: {}
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          app: "default"
```

Ingress configuration:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: bitcoin
  namespace: servicea
spec:
  ingressClassName: nginx
  rules:
  - host: azbastion.cloud
    http:
      paths:
      - backend:
          service:
            name: service-a
            port:
              number: 8080
        path: /
        pathType: Prefix
```

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: serviceb
  namespace: serviceb
spec:
  ingressClassName: nginx
  rules:
  - host: azbastion.cloud
    http:
      paths:
      - backend:
          service:
            name: service-b
            port:
              number: 8080
        path: /health
        pathType: Prefix
```