LAB 11 RESULTS AND COMMENTS: (26/01/26)

Objective:

Let's build our first Kubernetes application.

Steps:

1. Modify the code of the REST HTTP server so that now it will read some "preconfigured" resources from some file JSON file at boot. You can chose the location (e.g., /etc/my_res.json) and the internal structure of this file;
2. Re-build and ship the new version of the cointainer image;
3. Write the YAML files needed to deploy the HTTP server in the k0s cluster(i.e., one deployment file, one service file, and one ConfigMap file containing two "preconfigured" resources);
4. Deploy the application!;

N.B.

If you don't want to create a public account on Docker Hub you can run your local registry. You can find the container image for the registry here. Since the registry itsef it's a container you can run it inside your Kubernetes cluster and expose its port to up your container image there.

App.py:

```
  GNU nano 7.2
from flask import Flask, jsonify
import json
import os
import uuid

app = Flask(__name__)

# The exact directory path from your exercise requirements
CONFIG_PATH = "/etc/my_res.json"

# Global list to store the resources loaded at boot
generators = []

def load_boot_resources():
    """
    Reads the JSON file at /etc/my_res.json.
    Maps 'name' and 'value' from your ConfigMap into our internal list.
    """
    if os.path.exists(CONFIG_PATH):
        try:
            with open(CONFIG_PATH, 'r') as f:
                boot_data = json.load(f)

                if isinstance(boot_data, list):
                    for res in boot_data:
                        # Constructing the resource object using your JSON keys
                        resource = {
                            "id": str(uuid.uuid4()),  # Generates a unique ID for internal use
                            "name": res.get("name", "Unknown"),
                            "value": res.get("value", "No Value Provided"),
                            "params": {"fixed": 1} # Default wait time
                        }
                        generators.append(resource)
                    print(f"[*] Successfully loaded {len(generators)} resources.", flush=True)
                else:
                    print("[!] Error: JSON at /etc/my_res.json is not a list.", flush=True)
        except Exception as e:
            print(f"[!] Error reading config file: {e}", flush=True)
    else:
        print(f"[!] File {CONFIG_PATH} not found. Ensure ConfigMap is mounted.", flush=True)

# Load the Alpha and Beta resources before starting the server
load_boot_resources()

@app.route("/")
def welcome():
    return "<h1>REST Server: Alpha & Beta Resources Loaded</h1>\n"

@app.route('/response/v1/resources', methods=['GET'])
def get_all_resources():
    """Returns everything loaded from the ConfigMap."""
    return jsonify(generators), 200

@app.route('/response/v1/generate/<resource_name>', methods=['GET'])
def get_resource_by_name(resource_name):
    """
    Allows you to query a resource by name (e.g., /resource1).
    """
    # Find the resource where 'name' matches the URL parameter
```

```
    # Find the resource where 'name' matches the URL parameter
    res = next((g for g in generators if g["name"] == resource_name), None)

    if not res:
        return jsonify({
            "error": "Resource not found",
            "requested": resource_name,
            "available": [g["name"] for g in generators]
        }), 404

    return jsonify({
        "status": "success",
        "resource_name": res["name"],
        "resource_value": res["value"]
    })

if __name__ == "__main__":
    # Standard port 5000 as defined in your Service targetPort
    app.run(host="0.0.0.0", port=5000)
```

My-configmap.yaml:

```
  GNU nano 7.2
apiVersion: v1
kind: ConfigMap
metadata:
  name: rest-server-config
data:
  my_res.json: |
    [
      {
        "name": "resource1",
        "value": "Valore Alpha"
      },
      {
        "name": "resource2",
        "value": "Valore Beta"
      }
    ]
```

My-service.yaml

```
  GNU nano 7.2
apiVersion: v1
kind: Service
metadata:
  name: rest-server-svc
spec:
  type: NodePort
  selector:
    app: rest-server
  ports:
    - port: 80
      targetPort: 5000
      nodePort: 30080
```

My-deployment.yaml:

```
  GNU nano 7.2
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rest-server-dep
spec:
  replicas: 1
  selector:
    matchLabels:
      app: rest-server
  template:
    metadata:
      labels:
        app: rest-server
    spec:
      containers:
      - name: rest-server
        image: semplicementeria/rest-server:v2 # L'immagine creata al punto 2
        imagePullPolicy: Always
        ports:
        - containerPort: 5000
        volumeMounts:
        - name: config-volume
          mountPath: /etc/my_res.json # Percorso richiesto
          subPath: my_res.json
      volumes:
      - name: config-volume
        configMap:
          name: rest-server-config
```

My terminal on which I built my image NO CACHE otherwise it copies the previous paths of the docker file and an error of CRASHLOOPBACKOFF would appear. And I have created the image as a second version, not the v1 to have a more refreshed environment.

```
lost-011:LAB11_assignment mariapiabuonomo$ docker build --no-cache -t semplicementeria/rest-server:v2 .
[+] Building 10.5s (10/10) FINISHED                          docker:desktop-linux
 => [internal] load build definition from Dockerfile                          0.0s
 => => transferring dockerfile: 137B                                          0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim            1.1s
 => [auth] library/python:pull token for registry-1.docker.io                 0.0s
 => [internal] load .dockerignore                                             0.0s
 => => transferring context: 2B                                               0.0s
 => [1/4] FROM docker.io/library/python:3.9-slim@sha256:2d97f6910b16bd338     0.0s
 => => resolve docker.io/library/python:3.9-slim@sha256:2d97f6910b16bd338     0.0s
 => [internal] load build context                                             0.0s
 => => transferring context: 28B                                              0.0s
 => CACHED [2/4] WORKDIR /app                                                 0.0s
 => [3/4] RUN pip install flask numpy                                         6.7s
 => [4/4] COPY app.py .                                                       0.0s
 => exporting to image                                                        2.6s
 => => exporting layers                                                       2.1s
 => => exporting manifest sha256:c6da0227e00b28a09a9beeb42e459fa821c1745e     0.0s
 => => exporting config sha256:5f7353395518856561302094ec82a441554c5a3a3e     0.0s
 => => exporting attestation manifest sha256:1a3af763be878a63f59c3967aa83     0.0s
 => => exporting manifest list sha256:e65f05297c92f37d9c5613bfbefd8b5d6aa     0.0s
 => => naming to docker.io/semplicementeria/rest-server:v2                    0.0s
 => => unpacking to docker.io/semplicementeria/rest-server:v2                 0.4s
lost-011:LAB11_assignment mariapiabuonomo$ docker push semplicementeria/rest-server:v2
he push refers to repository [docker.io/semplicementeria/rest-server]
168ef7bd0d5b: Pushed
223f4b503473: Layer already exists
db0074e141e: Pushed
fa61c17f1c67: Pushed
ebfe7d4fa0c5: Layer already exists
a16e55119267: Layer already exists
479b8ad8bcc3: Layer already exists
9780716eac04: Layer already exists
v2: digest: sha256:e65f05297c92f37d9c5613bfbefd8b5d6aabe7c46df89838e51996db394427e0 size: 856
lost-011:LAB11_assignment mariapiabuonomo$
```

To make the previous step in a clear way, I have also freed the memory from previous docker images by starting fresh with:

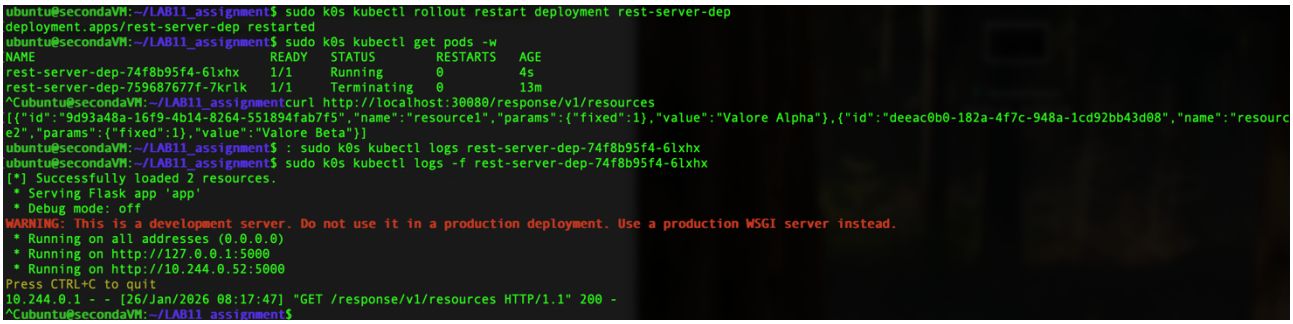
docker system prune -a (that eliminates: - all stopped containers
- all networks not used by at least one container
- all images without at least one container associated to them

- all build cache

After, I had applied eventual modifications to the configuration files (in this case were the same as before). And then I checked the status of the pods after a refresh of the rest server deployment. With a curl, I look for the resources to check if the system has taken the right ones.

In order to check the server logs:

sudo k0s kubectl logs -f *pod name* (with -f that stands for “follow”. This tag uploads the previous logs and closes them). To see the GET response, I open another terminal in order to perform anither time the curl so that I can see the “200” code on the server part.



FURTHER COMMENTS from previous tests:

- If nothing is installed (k0s): **curl -sSf https://get.k0s.sh | sudo sh**

  Then I initialize it and start

  sudo k0s install controller --single

  sudo k0s start

- Once I had a problem with taints: node-level configurations that "repel" pods, acting as restrictions to prevent specific pods from scheduling onto them, unlike labels which attract pods. So to eliminate this block I had to perform this:
  sudo k0s kubectl taint nodes --all node-role.kubernetes.io/control-plane-

- I had a lot of errors when the cache didn’t realize that I was using another app.py in another directory /app. For example, CrashLoopBack, ImagePullBackOff or

ErrImagePull. In order to analyze each error I had to perform a log on that specific "broken" pod**: sudo k0s kubectl logs *rest server dep name***
When I used a previous VM in which the memory (RAM) was too little for these operations with Kubernetes, the log gave me an error of **DISK PRESSURE** so I had to transfer all my files in a new VM ("secondaVM") that had more memory space. It also appeared another time but it was the problem of previous images that occupied space and further cache that I had to free.
Sometimes there was the problem of the exposed port too. For the logs, if the server Flask works properly, the port is seen in the terminal. So, if it's not coincident with the one specified in my-deployment and my-service YML files, it's never going to work.

- Always linked to the cache problem, I had to face an output of the curl that was an empty generator vector: **ubuntu@k8s-node:~$ curl http://localhost:30080/response/v1/resources**
  []

  By making some trials, I discovered that there was a sort of stale due to the subPath in the configuration file, it was like it didn't update it.