DevOps Orchestration

Objective

Update the backend and orchestrate migrating the 2 apps and script to Kubernetes Clusters following best practices using the technologies in the instructions.

Tasks and Solutions

Task 1

Add a new backend api:

- `/download_external_logs` makes a call to external service's api.
- The external download API is dummy api, *you may leave it blank,* however it requires $EXTERNAL_INTGERATION_KEY to authenticate
- the external api has multiple environments so the integration key varies by environment

Solution :

- In the `backend_api/app.py` file, a new api called download_external_logs is added. It makes an authentication using the $EXTERNAL_INTGERATION_KEY.
- The authentication key can also be passed via environment variables but here we have passed it using the $EXTERNAL_INTGERATION_KEY variable since it has only one environment. We can also import the key using os.environ.get() object.

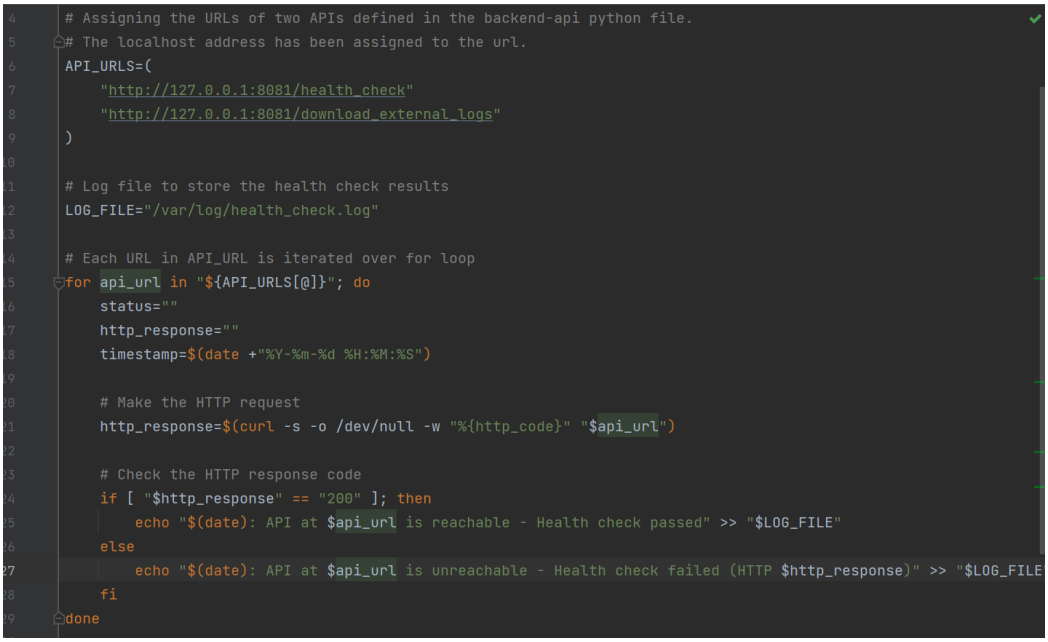
```
26
27  @app.route('/download_external_logs', methods=['GET'])
28  def download_external_logs():
29      #  Download External logs API : Authenticates with $EXTERNAL_INTEGRATION_KEY
30      #  Here the variable external_integration_key stores the authenticate key abd is passed into headers.
31      #  In case of multiple environments
32      #  we can import the EXTERNAL_INTEGRATION_KEY from the environment variable set using os.environ.get()
33      #  or we can also import it from the Vault 3rd party applications and define another API and assigning the value.
34      external_api_url = "https://sampleexternallink/auth/me"
35      external_integration_key = "BuZz9zaXplPTUweDNzc1NbVvvGtNjJ9"
36      headers = {"Authorization": "Bearer {}".format(external_integration_key)}
37
38      response = requests.get(external_api_url, headers=headers)
39
40      if response.status_code == 200:
41          return jsonify({"message": "External logs downloaded successfully"}), 200
42      else:
43          return jsonify({"error": "Failed to download external logs"}), response.status_code
44
```

Task 2

Update the health check to fit the new architecture

Solution

- We declare an array API_URLs and assign two values as seen in line number 6 below. These two values are the URLs to the APIs declared in the `backend-api/app.py` file.
- The URL is modified to match the localhost.
- We iterate through each value of the API_URLs and make a HTTP request.
- If the response is OK, i.e "200" then we echo the success message as reachable and health check passed and write it into the $LOG_FILE
- If the response is NOT OK, then we echo the failure message as unreachable, and the http_response and write it into the $LOG_FILE



```
# Assigning the URLs of two APIs defined in the backend-api python file.
# The localhost address has been assigned to the url.
API_URLS=(
    "http://127.0.0.1:8081/health_check"
    "http://127.0.0.1:8081/download_external_logs"
)

# Log file to store the health check results
LOG_FILE="/var/log/health_check.log"

# Each URL in API_URL is iterated over for loop
for api_url in "${API_URLS[@]}"; do
    status=""
    http_response=""
    timestamp=$(date +"%Y-%m-%d %H:%M:%S")

    # Make the HTTP request
    http_response=$(curl -s -o /dev/null -w "%{http_code}" "$api_url")

    # Check the HTTP response code
    if [ "$http_response" == "200" ]; then
        echo "$(date): API at $api_url is reachable - Health check passed" >> "$LOG_FILE"
    else
        echo "$(date): API at $api_url is unreachable - Health check failed (HTTP $http_response)" >> "$LOG_FILE
    fi
done
```

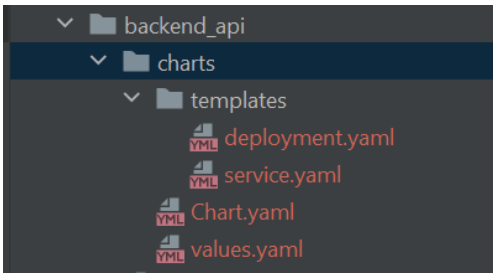
Task 3

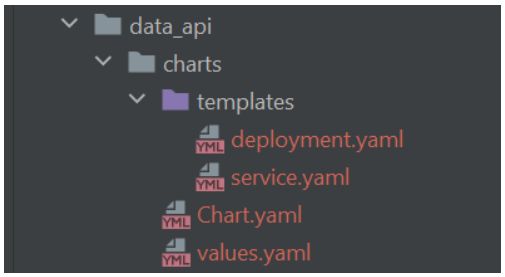
Create Helm chart for the stack

Solution

The below Helm Chart has been created for the project.

- The Kubernetes manifest files are inside the `templates` directory namely `deployment.yaml` and `service.yaml`
- The basic information about the chart is stored in `Chart.yaml` file.
- The values for the Kubernetes manifest files are stored in `values.yaml` file.





Task 4

Deployment via Ansible

Solution

- The Deployment of the application with Ansible is done using the `deployment.yml` file.
- We first build the two docker images with the corresponding Dockerfile located in backend_api and data_api directories respectively.
- We then install these images to the Kubernetes cluster using Helm Charts.
- We run the health_check.sh script to perform the health check.

```
---
- name: Deployment of the DevOps Orchestration project using Ansible
  hosts: localhost
  tasks:
    - name: Build backend api Docker image
      command: docker build -t backend-api ./backend_api
    - name: Build data api Docker image
      command: docker build -t data-api ./data_api
    - name: Install backend api to the Kubernetes cluster
      shell: helm install backend-api ./backend_api/charts
    - name: Install data api to the Kubernetes cluster
      shell: helm install data-api ./data_api/charts
    - name: Run health_check.sh script
      script: ./health_check.sh

```

Task 5

Monitoring Kubernetes Applications - Demonstrate how to monitor the node and Pod and container's resource utilization

Solution

- The Kubernetes node's resource utilization can be monitored using the command `kubectl top node`
- The Kubernetes pod's resource utilization can be monitored using the command `kubectl top pod`
- The Kubernetes container's resource utilization can be monitored using the command `kubectl top pod --containers`

Task 6

How to display only resource utilization for Pods with specific label (k8s-app=kube-Devops)

Solution

- To display only resource utilization for pods with specific label, we can use the command `kubectl top pod -l kube-Devops`