Methods of Cloud Computing

Winter Term 2024/2025

Practical Assignment No. 4

Due: 03.02.2025 23:55

The goal of this final assignment is to get even more familiar with Kubernetes. Specifically, we investigate a use case for carbon-aware computing, where we place Kubernetes based on the carbon intensity of electricity.

0. Prepare Virtual Resources & Kubernetes Cluster

From the previous assignment, you should already have scripts for setting up 3 GCP virtual machines and deploying Kubernetes onto them via Kubespray. Either re-use your existing Kubernetes cluster or re-execute these steps. As an outcome and prerequisite for the following tasks, you should have a working Kubernetes cluster.

1. Prepare the Workload

Our use case is about scheduling workloads in a Kubernetes cluster. As a first step, we need a workload that we can use for such a simulation.

Write a YAML file `workload.yaml` which includes a Kubernetes pod specification for the workload. The details of the actual workload are not relevant, as long as its execution time can be controlled: You can either use a custom container image, or rely on something existing, e.g. sysbench cpu benchmarking.

Outputs:

- `workload.yaml`

2. Carbon-Aware Pod Placement

We want to schedule Pods to nodes that currently have access to low-carbon energy. In practice, this requires information about the workloads (e.g. priority, expected execution duration, interruptibility) as well as energy-related metrics (like carbon intensity) and accurate forecasts. For the scope of this assignment, we will follow a simplified approach: We always execute one workload at a time and assume access to the average carbon intensity at the node's location. For scheduling, we prioritize the node with the lowest carbon intensity (refresh your knowledge on affinity).

Write a Python service `scheduler.py` that periodically reads from the `workload.yaml` template, assigns a unique name, defines node affinity based on carbon intensity, and deploys the specification to the Kubernetes cluster. Afterwards, the service should monitor the actual placement of the Pod. Your service should fulfill the following requirements:

- Everything is written in a single Python file.
- The workload scheduling period is configurable via an environment variable, with a default value of 60 seconds.
- The workload itself has a defined execution time of 60-180 seconds; The value is randomly chosen for each workload execution.
- Use https://wj38sqbq69.execute-api.us-east-1.amazonaws.com/Prod/row to retrieve artificial average carbon intensity values of three regions (updated every minute). The data is generated by a hosted Vessim instance, a testbed for carbon-aware systems developed at our research group[1].
- Associate each of your nodes with one region and use this mapping when determining which node is most promising.
- Use the kopf framework to implement custom operator logic, which allows you to subscribe to particular events within Kubernetes and hence record / observe the actual Pod placement after submitting your adapted specification.
- Configure Python logging so that you log the following events / information:
  - New unique workload name, carbon intensity, recommended Pod placement
  - Actual Pod placement, as observed e.g. via kopf handlers

Hints:

- Building and testing such an application might be easier using a local Kubernetes setup (e.g. using kind)
- For now, make your service use your admin kubeconfig file so that it can interact with the Kubernetes API.

Outputs:

- `scheduler.py`

[1] If you want to support us, feel free to star the repository to help us gain visibility: https://github.com/dos-group/vessim

3. Experiments & Discussion

We now want to run our scheduler service within the cluster. Package the service as a docker image and deploy it to the cluster as a Pod. You will also need to give the service appropriate permissions to access and manipulate Kubernetes objects via a ServiceAccount. Have a look at this guide.

Once your service is successfully running within the cluster, it is time to execute the experiments. For this, you need to slightly adapt your `scheduler.py` code. Sequentially (every 60 seconds; via the scheduling period configuration), schedule 300 workloads to the cluster - once with your carbon-aware placement strategy and once without. Collect the results (log files).

Look at your results and answer the following questions:

- Think about suitable ways for interpreting the recorded information and comparing them against each other (e.g. rank nodes based on carbon intensity forecasts and compute a histogram; estimate the normalized savings in carbon intensity, etc.). How did the two strategies perform?
- Have there been situations where a Pod was not placed as intended? If so, what might be the reasons? If not, is this even possible?
- Think about applying this in practice: What might go wrong, what are things we did not take into consideration?

Outputs:

- `Dockerfile`
- `rbac.yaml`
- `deployment.yaml`
- `normal_strategy.log, carbonaware_strategy.log`
- `answers.txt`

4. Submission Deliverables

Submit your solution on the ISIS platform as individual files. Please submit ONLY the necessary files and use exactly the given file names!

Expected submission files:

- `workload.yaml`
  - **5 points**
- `scheduler.py`
  - **25 points**
- `Dockerfile`
  - **5 points**
- `rbac.yaml`
  - **5 points**
- `deployment.yaml`
  - **5 points**
- `normal_strategy.log, carbonaware_strategy.log`
  - **10 points; 5 points each**
- `answers.txt`
  - **15 points; 5 points each**

Total points: 70