Methods of Cloud Computing

Winter Term 2019/2020

Practical Assignment No. 3

Due: 26.01.2020 23:59

In this assignment you will get familiar with container orchestration and the Infrastructure-as-code paradigm. Two major projects in this field are Kubernetes and Ansible, which you will use to set up and evaluate a small distributed infrastructure of two interdependent HTTP services.

0. Prepare Virtual Resources

In this assignment you will set up a Kubernetes cluster consisting of 3 machines. Prepare 3 virtual machines for this task. The preferred method is to use one of the public cloud platforms you worked with in the previous assignments. If you don't have enough credits left for this, work with virtual machines on your personal computer. Please use an Ubuntu Server 18.04 image for the VMs.

When using VMs in a public cloud, we suggest you open all TCP and ICMP traffic to avoid errors due to firewall settings. If you prefer a restrictive firewall setup, you will have to open ports successively when you run into connection problems. At least ports 22 (SSH) and 2379/2380 (etcd) are required.

Make sure you have password-less SSH access to the VMs and the SSH user has a password-less sudo prompt. You will need this to properly work with Ansible in the next steps. You should also install the `python-pip` package on your nodes to avoid errors later in the assignment.

1. Set up the Kubernetes Cluster

Deploy a Kubernetes cluster using Ansible. For this task, it is important to get familiar with Ansible, especially how to use the command line tool **`ansible-playbook`** and Ansible inventory files.

Follow these general steps:

- Clone the Kubespray repository on your local machine and checkout the commit tagged as 'v2.12.0' (makes sure you use a stable version).
- Create an Ansible inventory file named `hosts.yml` or `hosts.ini`. Use the Ansible documentation and follow the hints in the README file of the Kubespray repository.
  - Make sure all three of your worker VMs are used as both Kubernetes master and worker nodes
  - Tip: set the "`ansible_become: yes`" attribute on all worker nodes, but not in the "`all`" group to avoid become root on your local host.
- Run the main Kubespray playbook `cluster.yml` with your inventory file
  - Expect this to take at least 15-20 minutes
  - If the playbook keeps failing or timing out, try to completely re-create your VMs for a fresh start
- When the playbook succeeds, evaluate your Kubernetes cluster by logging into one of your VMs, becoming root, and running the following commands:
  - `kubectl cluster-info`
  - `kubectl get node`
    - Store the output of both commands in a file named `cluster-info.txt`
  - Note: with the default Kubespray deployment, you can only access your Kubernetes cluster as the root user from within your VMs. Accessing the cluster directly from your personal computer would require a more advanced configuration.

Outputs:

- `hosts.ini` or `hosts.yml`
- `cluster-info.txt`

2. Prepare application containers

Now that the Kubernetes cluster is running, it is time to prepare the Docker images that will make up our web services.
Write two Dockerfiles called `frontend.Dockerfile` and `backend.Dockerfile`. Both resulting images should be based on the latest official `nginx` Docker image from Dockerhub and use the files frontend.nginx.conf and backend.nginx.conf files, respectively, as the Nginx configuration. These files are provided on the ISIS platform for the course.

The frontend Nginx server will be the entrypoint for our webservice. It plays the role of a reverse proxy and forwards requests to the root path (‘`/`’) to a backend server. Besides this, it attaches the header `CC-Frontend-Server` to the HTTP reply, using the hostname as value. The backend Nginx server returns a simple Hello World text message and attaches the HTTP header `CC-Backend-Server` similar to the frontend. Using these two headers we can determine the path that incoming requests take through our deployment. Look at the provided `*.nginx.conf` files to understand their function in more detail.

After locally creating the Docker images, push the images to a public Dockerhub repository. You can use an existing repository, or create a new one.

List and comment the commands that you used for creating and uploading the Docker images in a file called `commands.txt`.

Outputs:

- `frontend.Dockerfile`
- `backend.Dockerfile`
- `commands.txt`

3. Deploy the application

The next task is to deploy the web service containers to Kubernetes. To make our deployment easily updateable and usable from the outside, we will use the Ansible module k8s (short for Kubernetes).

Write one single, self-contained Ansible playbook named `cc-webapp.yml` to deploy the entire web application. Do not use any Ansible roles or external playbooks. Make sure the playbook works with the `hosts.ini` (or `hosts.yml`) inventory file that you created earlier. The web application deployment must fulfill the following requirements:

- Run in its own Kubernetes namespace (not the default namespace)
- Both the backend and the frontend service consist of one Kubernetes Deployment object and one Service object, each.
- The frontend is replicated 4 times, the backend 6 times.
- Both Deployments should include a readiness probe and a liveness probe that probe the HTTP path "`/ready`".
- The Service object for the backend should be of type ClusterIP, while the Service object for the frontend should be of type NodePort. This ensures that the frontend service can be reached from the outside.

Hints:

- The tasks in the playbook should be executed only on the first Kubernetes master node (not all of them)
- To work with the k8s module, you need to install the Python module `openshift` on the target nodes. You can do this in your playbook using the Ansible pip module.
- The backend Service must be named in a way that the frontend service can reach it via DNS. This must match the hostname used in line 23 of the provided frontend.nginx.conf file. If you name your Service differently, you can modify the conf file and re-create your containers.

Execute your playbook with the `ansible-playbook` command. Document this command inside the commands.txt file from the previous tasks.

You can verify and debug your deployment and service objects with the `kubectl` tool (e.g. `kubectl get deployment`). Remember to use the `--namespace` parameter. You can also query your web services with commands like `curl` and `wget`.

Find out the node port chosen by Kubernetes to export your frontend services on the VMs. Document the command used for this in the file `commands.txt`.

Finally, test your deployment with the provided Python script `test-deployment.py`. The script should not be modified by you, and invoked with 3 parameters, which are the `host:port` pairs of your VMs, where the port is the node port chosen by Kubernetes to export your service. Document the command for starting the test script in the file `commands.txt`. When the tests succeed, store the output of the script in the file `test-output.txt`.

Outputs:

- `cc-webapp.yml`
- `commands.txt` file (extended from previous tasks)
- `test-output.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:

- `hosts.ini` or `hosts.yml`
  - **3 points**
- `cluster-info.txt`
  - **2 points**
- `frontend.Dockerfile`
  - **2 points**
- `backend.Dockerfile`
  - **2 points**
- `commands.txt`
  - **7 points**
- `cc-webapp.yml`
  - **15 points**
- `test-output.txt`
  - **3 points**

Total points: 34