Exercise 4: kubernetes & friends

Introduction

Please refer to the Exercise 4 FAQ for up-to-date answers on common problems. I will update them continually as new information rolls in.

By now, you should have a development environment on your laptop along with credit(s) for one or more cloud providers. (If you have not arranged for these credits, please do so now (for [AWS](), [Azure]() or [GCP]()). If you have problems applying for these, please contact your TA. You can also work with personal accounts but beware of the costs.

This exercise will revisit E3's application but you will instead use Kubernetes to orchestrate the containers:

- Assigning containers to the pods on which they will run
- Starting the containers
- Restarting them when they crash or fail health checks
- Establishing the network connections between services, both internal (inaccessible outside the cluster) and external (accessible outside the cluster

To manage cloud credits, you will run this exercise twice: once using a local cluster (Minikube) and a second time using a cloud provider of your choice (AWS, Azure or GCP).

Bear in mind that in all cases, the application uses DynamoDB key-value store which is running on AWS. The `cmpt756db` service will have to establish connections to Amazon to read and write the two tables.

A final word before I launch you into this exercise, this exercise is involved with many details. From prior cohort and TAs' experiences, it requires careful reading, attention to detail, follow-through and some initiative on your part. I suggest to read through the exercise *at least once* before starting anything. I have included a copious amount of links to fill in the background; follow and read them as required. Doing so will help you to orient to how the entire exercise fits together. Hopefully, Exercise 3 has prepared you by previewing some of the techniques and tools. Even though this is an individual exercise (submission and grading is per individual), you can definitely collaborate and discuss with your class mates as you work through it. Because of the importance and size of this exercise, this exercise is intentionally set for a longer period of time (two weeks instead of the usual one). But do not procrastinate as there is no substitute for time to read and absorb the material here.

0. Platform Considerations

This exercise was written for the Linux/MacOS environments. It will also work for Windows WSL2. Please follow the flow-chart below for specifics. **Note that the term 'native host' refers to the environment that your laptop/workstation boots into.**

0. If your native host has less than 16GB of memory, you may find it challenging to work through this exercise. If so, you can choose to use the CSIL/Blusson lab. (`.bash_profile` needs `EXPORT=/usr/shared/CMPT/scratch/$USERNAME`.) You will be using Linux with Docker as your CRI. Continue with step 2 below.

1. If your native host is MacOS, you can proceed with the exercise as is for both clusters (Minikube & cloud). The only choice (that is needed by everyone) is to choose a CRI for Minikube. Put simply, the CRI is the technology that the cluster will use for operating the containers. A good option is plain Docker. A slightly more realistic option is to choose a hypervisor: VirtualBox (free) or VMWare Fusion (paid/commercial).

2. If your native host is Linux, you too can proceed with the exercise as is. Again, the only choice you have is to choose a CRI for Minikube. Put simply, the CRI is the technology that the cluster will use for operating the containers. Again, a good option is plain Docker. A slightly more realistic option is a hypervisor. I have less experience with Linux virtualization but VirtualBox is a reasonable and free option.

3. If your native host is Windows, you have a more complex situation. Thankfully, Window 10's WSL2 makes it fairly easy. (Essentially, you will use Windows' built-in virtualization to install a Linux machine.) See here to setup WSL2. Note that you need WSL2, not the older WSL. Then install Ubuntu 20.04 LTS from the Microsoft store. Remember to change your default WSL2 distribution to Ubuntu 20.04 by following this.

   Now, you can access your Linux machine simply by running 'bash' from your Windows command-line. This Linux machine is useable for both Minikube and any of the cloud provider.

   You will use Docker as the CRI for Minikube.

   Now start up your Ubuntu machine by typing `bash` at any command. You will need to set a root password. Afterward, you will setup your environment. The default image is very up-to-date so only a handful of items are necessary.

```
$ cd ~
$ sudo apt update
$ sudo apt install make docker unzip
```

Now you can follow the instruction for Linux as in Exercise 3 to install the same toolset, being mindful to follow the Linux option:

1. awscli
2. eksctl
3. istio

1. Preparing Your Service

Start by pulling the code for this exercise and refer to the `e-k8s` subdirectory:

```
$ git clone https://github.com/scp-2021-jan-cmpt-756/sfu-cmpt756.211
```

Here's an abbreviated overview of the content:

directory/file	Note
e-k8s/s1/	the User service
e-k8s/s2/	the Music service
e-k8s/db/	the database writer service
e-k8s/postman/	the API definition collection
e-k8s/cluster/	material for working with a Kubernetes cluster
e-k8s/logs	all output files are collected here
e-k8s/tools	misc tools to prepare repo content (makefiles) for use
e-k8s/gatling	For Exercise 5 onward
e-k8s/{cmusic.sh, dmusic.sh, duser.sh}	scripts to drive the API
e-k8s/mk.mak	instantiated makefile to setup of a Minikube cluster
e-k8s/k8s.mak	instantiated makefile to operating a Kubernetes cluster
e-k8s/api.mak	instantiated makefile to operate the API
e-k8s/obs.mak	makefile to manage the observability components (Grafana, Prometheus, Kiali)
e-k8s/allclouds.mak	instantiated makefile to examine all cloud providers
e-k8s/az.mak	instantiated makefile for setup of an Azure AKS cluster
e-k8s/eks.mak	instantiated makefile for setup of an AWS EKS cluster
e-k8s/gcp.mak	instantiated makefile for setup of a GCP GKE cluster

Note that the exercise is organized as a collection of templates (files with `-tpl` suffix) that require instantiation. Do not fill in/use the `-tpl` files directly since you run the risk of leaking sensitive information should they be pushed (inadvertently or otherwise) to Github etc. (The `.gitignore` are setup to exclude your instantiated files.)

1.1 Fill in the required values in the template variable file

Copy the file `cluster/tpl-vars-blank.txt` to `cluster/tpl-vars.txt` and fill in all the required values in `tpl-vars.txt`. These include things like your AWS keys, your GitHub signon, and other identifying information. See the comments in that file for details.

Important: if you are using an AWS starter account, be sure to fill in the session token. The starter account's session token expires periodically (about an hour) so you will need to reinstantiate the templates periodically. If you are using a standard account, leave the field empty.

1.2 Instantiate the templates

Once you have filled in `tpl-vars.txt`, run

```
$ make -f k8s-tpl.mak templates
```

Whenever you update your session token, update `cluster/tpl-vars.txt` and re-run the command above to propagage the change into your environment.

This will check that all the programs you will need have been installed and are in the search path. If any program is missing, install it before proceeding.

The script will then generate the instantiated makefiles (as indicated above) that you can use to operate your environment.

Note: This is the *only* time you will call `k8s-tpl.mak` directly. Do not call/use the `-tpl.mak` files.

2. Cluster creation

You will now install Minikube to complete your toolset. Minikube is a single-node cluster environment for learning and experimentation. You will set this up on your laptop locally to provide a handy place to develop and work.

Tool	Location	Notes
`minikube`	local	single-node cluster suitable for learning and experimentation
`eksctl`	AWS	production-class cluster hosted on AWS
`az`	Microsoft Azure	production-class cluster hosted on Azure
`gcloud`	Google Cloud Platform	production-class cluster hosted on GCP

Once the cluster is created, you use the one common tool for all k8s clusters: `kubectl`.

In the same fashion, `istioctl` operates on a k8s cluster irrespective of how the cluster come to be.

2.1 Install `minikube`

Follow the instructions from kubernetes.io to install `minikube` as appropriate for your environment.

If you are using MacOS or a non-virtualized Linux (not WSL2), you can choose either Docker or a hypervisor for your CRI and Minikube will install the appropriate driver for it. `mk.mak` is set up by default for virtualbox already. (See the line `DRIVER=virtualbox`.)

If you are working in Windows WSL2 with Ubuntu (see section 0), you must use Docker as your CRI for Minikube. Change the line to `DRIVER=docker`

Cluster start up is dead easy:

```
minikube start
```

The cluster will start up within a minute or so. Verify that the cluster is operational:

```
minikube status
```

Once you cluster is up and running, you switch over to `kubectl` to work with it. For example, to examine your kubeconfig:

```
kubectl config get-clusters
kubectl config get-contexts
```

The first command will show one cluster named `minikube` while the second will show an additional context (also named) `minikube` and marked with an asterisk indicating it is currently active:

```
CURRENT   NAME                 CLUSTER          AUTHINFO         NAMESPACE
          az756
          docker-desktop       docker-desktop   docker-desktop
          docker-for-desktop   docker-desktop   docker-desktop
*         minikube             minikube         minikube
```

Save the output of this command to your submission document.

2.1.1 kubeconfig

`kubectl` operates from a so-called kubeconfig file which contains the definitions of clusters and contexts. There is no file with such a name. Rather, the file is `~/.kube/config` (typical for MacOS/Linux/WSL2; native Windows will be similar) which holds the clusters and contexts set up.

Examine your kubeconfig file now.

One obvious but subtle point about kubeconfig is that its content (each entry being a context comprising a cluster and a set of credential) exists independently of the resources that it refers to. The makefiles for the course's set of exercises tries to keep the two in sync: when a new cluster is created, a corresponding entry is added to the kubeconfig. And similarly when you delete a cluster (its kubeconfig entry is

removed). But it is possible for the two to diverge. For example, if you create a cluster via the makefile/command-line and use the cloud provider's web portal to delete this same cluster.

There is an included `allclouds.mak` to fetch the status across all your environments:

```
$ make -f allclouds.mak
gkyc@fontina.lan:~/...somepath/sfu-cmpt756.211/e-k8s$ make -f allclouds.mak
kubectl config get-contexts
CURRENT   NAME       CLUSTER    AUTHINFO                   NAMESPACE
*         az756      az756      clusterUser_c756ns_az756
          minikube   minikube   minikube

Azure (az.mak):
az aks list -o table
Name    Location       ResourceGroup    KubernetesVersion    ProvisioningState    Fqdn
------  -------------  ---------------  -------------------  -------------------  ------------------------------------
az756   canadacentral  c756ns           1.19.3               Succeeded            az756-c756ns-4a1b66-5e84ff59.hcp.cana

AWS (eks.mak):
eksctl get cluster --region us-west-2
No clusters found

GCP (gcp.mak):
gcloud container clusters --zone us-west1-c list

Updates are available for some Cloud SDK components.  To install them,
please run:
  $ gcloud components update

DynamoDB tables, read units, and write units
Music   5       5
User    5       5
User.restore    5       5

Run the following command to list any background Gatling jobs for this process
jobs | grep gatling
```

2.1.2 make & makefile

The exercises for 756 is organized around a set of makefile to simplify and to introduce the various tools. If you aren't familiar with makefile or the make tool, refer here.

There are 7 files that you will interact with frequently:

file	Purpose
mk.mak	instantiated makefile to setup of a Minikube cluster
k8s.mak	makefile for operating k8s
api.mak	instantiated makefile to operate the API
allclouds.mak	instantiated makefile to examine all cloud providers
az.mak	instantiated makefile for setup of an Azure AKS cluster
eks.mak	instantiated makefile for setup of an AWS EKS cluster
gcp.mak	instantiated makefile for setup of a GCP GKE cluster

The core idea of a makefile is to automate the execution of commands to reproduce artifacts in a chained fashion. It is typically used to compile program source into executables but it is applicable anytime you have outputs that are dependent on a chain of upstream inputs.

For example, `mk.mak` has been setup to start, stop, and check on the status of your Minikube cluster as follows:

command	Purpose
`make -f mk.mak start`	start your Minikube cluster
`make -f mk.mak stop`	stop your Minikube cluster
`make -f mk.mak delete`	delete the local VM serving your Minikube cluster
`make -f mk.mak status`	check on the status of your Minikube cluster
`make -f mk.mak ls`	examine the components inside the Minikube cluster

command	Purpose
`make -f mk.mak dashboard`	start up the dashboard for your Minikube cluster
`make -f mk.mak lb`	start up a loadbalancer to access your Minikube cluster

It may seem dubious value to introduce this extra machinery when Minikube is already easy and direct to start, stop and check on. But one benefit of using make is to wrap a *series* of commands and associate it with a pseudo-target (e.g., the `start`, `stop`, or `ls` in the above) for automation. (The `ls` pseudo-target is an example that will display all deployments, pods and services inside your cluster.) More crucially, adopting a set of consistent pseudo-targets across a number of makefiles allows us to homogenize the creation of clusters in multiple contexts--across the a set of public cloud providers.

Thus,

command	Purpose
`make -f az.mak start`	create an AKS cluster
`make -f eks.mak start`	create an AWS EKS cluster
`make -f gcp.mak start`	create a GCP GKE cluster
`make -f allclouds.mak` | a convenient `ls -a` across all three clouds	

2.2 Start up clusters

To help with learning k8s, this exercise is built around a set of makefiles that wrap many long and verbose commands into convenient short snippets. Please study the makefiles (*.mak) and examine the commands therein to work out what's happening. You will need to understand these commands to effectively use Kubernetes here and in the term project.

2.2.1 minikube

Start up your Minkube cluster as follows:

```
$ make -f mk.mak start
```

As this is a local operation and a very small (single-node) cluster, it will complete very quickly.

2.2.2 Cloud cluster

Start up a cloud cluster as follows:

```
$ make -f VENDOR.mak start
```

where VENDOR is one of eks, az, or gcp.

This is by contrast a slow operation. If you review `eks.mak`, this is the command that was used:

```
eksctl create cluster --name aws756 --version 1.17 --region us-west-2 --nodegroup-name worker-nodes --node-type t3.smal
```

You can see where parameters have been used to allow for easy tailoring. At the completion of this command (typically 10-15 minutes), you will have a barebone k8s cluster comprising some master nodes and 2 worker nodes (each of which is an Azure instances). (The nodes are specified by the tail `--node-type t3.small --nodes 2 --nodes-min 2 --nodes-max 2`.)

(AKS and GCP are similar. Please examine the makefile accordingly.)

2.2.1 kubeconfig

Upon completion of the creation of one or both of your clusters, you can see a summary of your current environment (also known as kubeconfig) by:

```
$ kubectl get-contexts
```

```
CURRENT   NAME                                          CLUSTER                      AUTHINFO
          iam-root-account@aws756.us-west-2.eksctl.io   aws756.us-west-2.eksctl.io   iam-root-account@aws756.us-west-2.
*         minikube                                      minikube                     minikube
```

Read up on k8s kubeconfig contexts. This is a vital concept for working and managing k8s clusters.

Observe that upon creation of each cluster via the corresponding VENDOR.mak, your kubeconfig gains an additional entry indicating that there is an additional cluster available to you. As you work with `kubectl`, you choose which cluster to operate on. Strictly speaking, each entry returned by `kubectl get-contexts` is a context which comprise a cluster with a specific set of credentials (the `AUTHINFO`). For convenience, there is a 'current context' (indicated by the asterisk * above) which will be the default if you do not specify a context.

Minikube creates both a context and a cluster with the same name.

EKS/AKS/GKS does similar though the cloud vendors use rather verbose name for the context. To save your fingers, you can shorten the context name via the following (substituting your specific context name as appropriate):

```
$ kubectl config rename-context iam-root-account@aws756.us-west-2.eksctl.io aws756
```

The makefiles included with this exercise relies upon the following context names for each cloud vendor. Always remember to rename the context after you create it.

Cloud	Context Name
AWS EKS	aws756
Azure AKS	az756
GCP GKE	gcp756

While you're here organizing your context, create a `c756ns` namespace inside each cluster and set each context to use this:

```
$ kubectl config use-context minikube
$ kubectl create ns c756ns
$ kubectl config set-context minikube --namespace=c756ns
```

```
# for AWS EKS; similar for AKS/GKE
$ kubectl config use-context aws756
$ kubectl create ns c756ns
$ kubectl config set-context aws756 --namespace=c756ns
```

From this point on, you can use the shorter:

```
# for Minikube
$ make -f mk.mak cd
# for AKS/EKS/GKE
$ make -f VENDOR.mak cd
```

to switch between contexts/clusters.

Confirm that your kubeconfig has c756ns as the default namespace for all contexts.

2.3 Stop the clusters

2.3.1 Minikube

You can stop your Minkube cluster as follows:

```
$ make -f mk.mak stop
```

Stopping the Minikube cluster stops the VM behind the scene.

To delete the VM that is underlying your Minikube cluster:

```
$ make -f mk.mak delete
```

Practice starting, stopping and deleting your Minikube cluster so you are comfortable with the impact on your system. These are fast (and cheap) operations so practice away. You won't be able to do this as readily once you move to the cloud.

2.3.1 Cloud cluster

For the cloud vendors, the following will stop the cluster:

```
$ make -f VENDOR.mak stop
```

In the public cloud, stopping a machine is tantamount to deleting a machine since your hold on the resource ends when you stop using (and paying for) it.

As with startup, this command will take longer to complete in the cloud than for Minikube. This command will complete at this time because your cluster was freshly initialized with no resources (beyond the barebone nodes). In general, cloud vendors will not allow you to delete a cluster unless all resources within it have been cleaned up appropriately. More on this later.

With your cloud cluster, practice at least stopping it once. Once you are comfortable, restart the cluster. Beware that one stop-and-start cycle is easily 15-30 minutes so manage your time. When you are comfortable, leave your cloud cluster up to continue with the exercise. (But we will stop the cluster at the end of this exercise.)

2.3.1 Managing cloud costs

Two additional sub-commands have been added to `VENDOR.mak` to manage costs within your cloud.

As the course's education credit is limited (<= $100), you will want to pay attention to your spending. For example with EKS, there are two costs to watch out for: NAT gateway (which allow your cluster to communicate with the public Internet) are charged at $0.045/h while the t3.small EC2 instances (which provides the compute resources for your containers) are charged at $0.0208/h. If you leave these up continuously, you will deplete your credit prematurely. ($100/($0.045+$0.0208)=1519h or about 9 weeks. And there are other costs beyond these 2 pieces.)

To avoid burning precious credit, remember to dispose of the compute/node group whenever you do not need them (overnight or just away for an extended period (2h+)).

Delete your node group will not harm the cluster as that exists independently of the resources within it. There just would be no resources to serve your application.

To delete the node-group of your cloud cluster:

```
$ make -f VENDOR.mak down
```

To recreate the node-group of your cloud cluster:

```
$ make -f VENDOR.mak up
```

Either of these command will take a relatively long time (10+ min) to complete. In fact, the time of the initial creation of your cluster is largely composed of the time for creating the node group.

This `down` command will again complete readily at this time because there are no services running on them.

The `ls` command will determine what services are running on your cluster (the nodes). You will need to stop any service that exposes an external IP before you can delete the cluster.

```
$ make -f az.mak ls
gkyc@fontina.lan:~/newroot/GitHub.nosync/sfu-cmpt756.211/e-k8s$ make -f az.mak ls
kubectl config get-contexts
CURRENT   NAME       CLUSTER    AUTHINFO                   NAMESPACE
*         az756      az756      clusterUser_c756ns_az756
          minikube   minikube   minikube
az aks list -o table
Name    Location       ResourceGroup    KubernetesVersion    ProvisioningState    Fqdn
------  -------------  ---------------  -------------------  -------------------  ------------------------------------
az756   canadacentral  c756ns           1.19.3               Succeeded            az756-c756ns-4a1b66-5e84ff59.hcp.cana
gkyc@fontina.lan:~/newroot/GitHub.nosync/sfu-cmpt756.211/e-k8s$
```

2.4 istio

`istio` is a service mesh that was conceived concurrently with k8s. But for various reasons, it was ultimately pulled out of k8s and developed as an independent project.

`istio` uses the side-car pattern to manage network traffic for an application within a specified namespace. Refer here for details.)

`istio` is installed into each cluster only once but it will only intervene for specific namespaces that you choose within the cluster. A k8s namespace is a cluster-level construct that organizes the resources within your cluster. You can liken it to the way a filesystem have folders though namespaces are only one-level deep. (You can't nest namespaces as you can with file system directories.)

To use `istio` with an application, you create a namespace for your application, install the components of your application into this namespace and mark the namespace for `istio` to 'inject' itself.

2.4.1 Minikube & tunnel

For Minikube:

```
# switch to the Minikube context
$ kubectl config use-context minikube
$ istioctl install --set profile=demo --set hub=gcr.io/istio-release
$ kubectl label namespace c756ns istio-injection=enabled
```

Now, start up a side terminal shell and run the following:

```
$ minikube tunnel
# or
$ make -f mk.mak lb
```

See here for more details on how the tunnel works.

As this invocation is a network tunnel, this is a blocking call; it will direct traffic into your Minikube cluster while it is running. When you are done with the tunnel, press ctrl-C to terminate the tunnel.

On MacOS (and likely Linux), you will need to supply the root password for your laptop.

On MacOS the tunnel will report the following continuously:

```
Status:
    machine: minikube
    pid: 86020
    route: 10.96.0.0/12 -> 172.16.199.128
    minikube: Running
    services: [istio-ingressgateway]
    errors:
        minikube: no errors
        router: no errors
        loadbalancer emulator: no errors
```

With Windows WSL2, the tunnel is silent (no output).

2.4.2 Tunneling into your cluster in the cloud

For EKS:

```
# switch to the EKS (or whichever cloud) context
$ kubectl config use-context aws756
$ istioctl install --set profile=demo --set hub=gcr.io/istio-release
$ kubectl label namespace c756ns istio-injection=enabled
```

The steps above will trigger istio to set up an AWS ELB to act as an ingress gateway.

(The comparable operation on Azure or GCP will do the same.)

Now, use `kubectl get svc --all-namespaces` (or the `lsa` pseudo-target) to verify that you have 3 new services under a new `istio-system` namespace: `istio-egressgateway`, `istio-ingressgateway`, & `istiod`.

Finally, you will need to determine how to access your cluster.

Examine your system:

```
$ kubectl -n istio-system get service istio-ingressgateway
```

```
NAME                   TYPE           CLUSTER-IP      EXTERNAL-IP
istio-ingressgateway   LoadBalancer   10.100.108.96   a67ee95372ccc43189c3b4d7ebe3ad3f-2110992951.us-west-2.elb.amazona
```

Note the entry under `EXTERNAL-IP` which is the entry point to your cluster. The example above (for EKS) shows the rather ungainly DNS name (`a67ee95372ccc43189c3b4d7ebe3ad3f-2110992951.us-west-2.elb.amazonaws.com`). With Minikube, this will be a raw IP address (typically, 10.x.y.z).

Whenever there is no way into your cluster, you will have a `<pending>` as your EXTERNAL-IP.

2.5 dashboard

Kubernetes provides a standard dashboard to examine its internal states. Accessing it though varies according to how the cluster is set up.

Spend some time to explore each of these to get a feel for the capabilities and flexibility of Kubernetes.

There isn't much happening just yet in your cluster but keep these in mind for the future.

2.5.1 Minikube dashboard (easy!)

Minikube offers the standard Kubernetes web dashboard through its direct 'dashboard' sub-command which is implemented directly in `mk.mak`:

```
$ make -f mk.mak dashboard
```

This will setup the process and also navigate your default browser to the dashboard. (You will need to Ctrl-C and kill the dashboard command in the terminal when you are done.)

2.5.2 Dashboard into EKS cluster (a bit more involved)

The dashboard into your EKS cluster is more involved.

Follow Amazon's own tutorial to set up the appropriate bits inside your cluster. You only need to do this once per cluster. (Unfortunately, this means you will re-do this after you stop your cluster (to save credits!).)

For your convenience, an `eks-admin-service-account.yaml` for Step 3.1 is included with this exercise at `e-k8s\cluster\eks-admin-service-account.yaml`

And as with Minikube, you need to do a similar `pkill kubectl` to terminate the proxy after you are done with the dashboard.

2.5.3 Dashboard into AKS cluster (easy!)

Microsoft Azure has added a dashboard into its Azure portal for examining and working with your Kubernetes cluster. You will find this in the Azure portal under "Kubernetes services". (Find the purple icon with 7 pips.) This is not exactly the Kubernetes project's dashboard but it has similar functionality.

2.5.3 Dashboard into GKE cluster (easy!)

Refer to Google's documentation on their GKE dashboards

2.6 Restarts

Whenever you create (not restart after a stop) a cluster, remember the following:

1. Re-install istio into the cluster
2. Recreate the namespace for your work
3. Label this namespace for istio injection
4. Set the default namespace for the context to this new namespace (for convenience)
5. Confirm the expected namespaces and services are present

The `reinstate` target does this:

```
# for Minikube
$ make -f mk.mak reinstate
# for EKS
$ make -f eks.mak reinstate
```

3. Deploying your Services

3.1 Building your images

As in Exercise 3, you will need to build the containers and set them up in the container registry.

Kubernetes uses CoreDNS to handle discovery and routing of the service calls.

The difference between the service that you ran in Exercise 3 and what you will run in Exercise 4 is exactly in how services are referenced. You can see this clearly when you compare the code for Docker (`s1/appd.py`) with that for Kubernetes (`s1/app.py`):

```
gkyc@fontina.lan:~/...mypath/sfu-cmpt756.211/e-k8s$ diff s1/app.py s1/appd.py
35c35
<     "name": "http://cmpt756db:30002/api/v1/datastore",
---
>     "name": "http://host.docker.internal:30002/api/v1/datastore",
```

Build your images with (`cri` short for container registry images):

```
$ make -f k8s.mak cri
```

Finally, there is one manual step left before the system can come up auto-magically: to open up access of your container repositories to allow public access. This is reasonable within the context of this exercise because the work involved to set up authentication from the cloud provider (AWS, Azure or GCP) back to `ghcr.io` is more than I can bear at this time.

Refer to GitHub's documentation to set public access on your container repositories

3.2 Kubernetes operation

Kubernetes operates by way of a declaration specified via a manifest file. There are two formats supported for a manifest file--JSON & YAML--with YAML the preferred format. See here. A manifest file contains one or more declarations of resource for k8s to set up.

This exercise contain 3 services: s1, s2 and db. s1 and db are comparable in that they are both unversioned. (There is only one version of either.) In contrast, s2 is provided with 2 versions. Note that the YAML files in `cluster` are similarly templatized (have a `-tpl` suffix) as the makefiles. As before, only use the files without the `-tpl` suffix. You will find all manifests in the `cluster` directory.

Our usage of `istio` necessitates a manifest (`cluster/service-gateway.yaml`) to declare the ingress gateway to allow traffic into the cluster.

Each resource in a manifest file starts off with 4 common element: `apiVersion`, `kind`, `metadata` and `spec`. The first three are standardized while the last (`spec`) varies according to the resource. Refer to the documentation on k8s objects.

We will use a combination of a `Deployment` resource to declare the desired resources to power your container and a `Service` resource to expose the containers' capabilities to the world. (Refer to the documentation for Deployment and Service)

Refer to `cluster/s1.yaml` and look for the Deployment section which looks similar to:

```
spec:
      serviceAccountName: svc-s1
      containers:
      - name: service1
        image: docker.io/<YOURNAME>/cmpt756s1:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 5000
```

Note that your manifest contains your Github id in place of `<YOURNAME>`. (This was performed during the instantiation process at the start of this exercise.)

Recall that you have already created a namespace `c756ns`. You will now run your container in your cluster by `APPLY`ing this manifest. For Minikube, this would be:

```
$ make -f mk.mak cd
#or
$ kubectl config use-context minikube
```

And for EKS:

```
$ make -f eks.mak cd
#or
$ kubectl config use-context aws756
```

Finally, follow up with the pseudo-target s1. (You are using a pseudo-target instead of just applying the single s1.yaml because there is more machinery involved that I won't get into presently.)

```
$ make -f k8s.mak s1
```

Verify that the service is operating with:

```
$ make -f k8s.mak ls
```

To shutdown the service

```
$ kubectl -n c756ns delete svc service1
```

You can deploy the other services via this commandline:

```
$ make -f k8s.mak s2 db gw
```

3.3 Getting into your cluster

Review section 2.4.2 to look up the EXTERNAL-IP for your cluster in turn. (This exercise is setup to use port 80.)

Fill in the IP/DNS-port combo into `api.mak` as the IGW variable.

Study the following pseudo-targets inside `api.mak` which will call the API. This exercise only uses `curl`. Pay attention to the variables used to pass the various parameters to `curl`.

`api.mak` psuedo-target	operation
cuser	create a user
apilogin	login as a specified user
apilogoff	logoff as a specified user
uuser	update a user
duser	delete a user
cmusic	create of a song
dmusic	delete of a song

Fill in the variables inside `api.mak` as appropriate.

For each cluster, perform the following:

- 1. Create 2 users.
- 2. Delete one of the 2 users.
- 3. Update the remaining user.
- 4. Login as the remaining user.
- 3. Create 2 songs as the user that you've log in as.
- 6. Delete one of the 2 songs.
- 7. Logoff from the previously log-in user.

Save the output of each operation. With the creation of user and music, add a serial no to distinguish the various runs (`mv logs/cuser.out logs/cuser1.out` etc). You will have 9 files in total.

4. Cleaning up your cluster

Remember to shutdown your clusters after this exercise.

Your Minikube cluster is backed by either Docker or a virtual machine. Minikube cleans up after itself with `minikube delete`.

You need to be particularly careful about your cloud-side clusters as they are expensive to keep over longer periods of time (overnight). Remember that you can use `allclouds.mak` to help with this.

5. Submission

Work through section 3.3 above for Minikube and one cloud cluster (one of AWS, Azure or GCP). For clarity of understanding, you may choose to run thru 2 or more cloud clusters. However, this exercise submission requires the submission of exactly two clusters: Minikube and one cloud cluster of your choosing.

Collect the Material

Here's a checklist of the items to collect:

output file	Note				
logs/mk-start.log	output file for creation of Minikube cluster				
logs/mk-stop.log	output file for deletion of Minikube cluster				
{eks	aks	gks}-start.log	output file for creation of {EKS	AKS	GKE} cluster
{eks	aks	gks}-stop.log	output file for deletion of {EKS	AKS	GKE} cluster

For each cluster:

output file	Note
logs/cuser1.out	output file for creation of user 1
logs/cuser2.out	output file for creation of user 2
logs/duser.out	output file for deletion of a user

output file	Note
logs/uuser.out	output file for update of a user
logs/apilogin.out	output file for login as a user
logs/cmusic1.out	output file for creation of song 1
logs/cmusic2.out	output file for creation of song 2
logs/dmusic.out	output file for deletion of a song
logs/apilogoff.out	output file for logoff as a user

Paste the content of each file above into the document below.

Create a PDF

Open the submission template(GDoc format).

Fill in:

a. The header box at the top of the document.

b. Content from the steps above.

c. Answer the additional questions.

When complete, generate a PDF.

You must name your PDF according to the pattern: **SFU-student-no**`-e4-submission.pdf` where **SFU-student-no** is the numeric id (typicall 30...) assigned to you upon entering SFU. Unfortunately, you will be penalized for incorrect filename because of cascading dependencies for a large class.

A penalty will be assessed for failure to name your submission appropriately.

Canvas submission

Navigate to this exercise and upload your PDF.

Reference

Introduction to Kubernetes A tight 15 min introduction to Kubernetes.

TechWorld with Nana YouTube Channel with tutorials for many distributed systems topics including Docker and Kubernetes.