Show Submission Credentials

P2. Containers: Docker and Kubernetes Linux Containers

Introduction

Information

Learning Objectives

At the end of the assignment, students should be able to:

1. Describe Docker and Kubernetes and how a developer uses them to implement a microservices architecture (MSA).
2. Configure and manage Kubernetes clusters.
3. Create and deploy Helm charts to manage Kubernetes applications.

4. Develop Dockerfiles, build Docker images and deploy Docker containers in order to containerize RESTful Spring applications.
5. Create container registries and push custom Docker images to the registries.
6. Deploy applications containers in Kubernetes clusters on multiple clouds.
7. Compare single Kubernetes cluster deployment and multi-cloud deployment.
8. Identify downstream service failures. Route traffic away from a failed cluster and scale up a service in the healthy cluster using autoscaling rules.
9. Develop and deploy real-time and scalable web services using the microservices pattern.
10. Explain the necessary steps to break up a monolithic application into microservices.
11. Explain how containerized services get registered and communicate.
12. Define, manage and monitor global routing based on performance and availability using Azure Front Door Service

Information

General Details

The following table contains some general information about this project:

Prerequisites	Java, Maven, REST
Primers	Intro to Containers and Docker Kubernetes and Container Orchestration
Applicable Languages	Java only Note: If you use Maven, the *Maven central repository* is the only remote repository you are allowed to use.
Applicable Cloud Platform	Google Cloud Platform (GCP), Azure
Total *Recommended* Budget	$20 on GCP, $20 on Azure
Tags Required	Key: `project` Value: `containers` for both GCP and Azure

Warning

Tagging and Budget Notes on GCP and Azure

GCP and Azure Tagging

Tag Azure and GCP resources using `project` : `containers` .

Azure Subscription

Be sure to use the correct subscription when you are using the Azure portal or Azure CLI. As you may have multiple subscriptions, make sure you are using the course subscription for this project. If you have multiple subscriptions, you can use the command below to change the default subscription.

```
az account list --output table --refresh
az account set --subscription <name or id>
```

Warning: your subscription will be disabled if you run out of your subscription budget. Please exercise caution to plan the budget.

Danger

Project Grading Penalties

The following table outlines the violations of the project rules and their corresponding grade penalties for this project.

Note that a penalty is the absolute value as per the table, not calculated by a percentage of your total score.

Violation	Penalty of the project grade
Incomplete submission of required files	-10%
Submitting your credentials, other secrets, or submission username in your code for grading	-100%
Submitting only executables (`.jar` , `.pyc` , etc.) without human-readable code (`.py` , `.java` , `.sh` , etc.)	-100%
Attempting to hack/tamper the grader	-100%
Cheating, plagiarism or unauthorized assistance (please refer to the university policy on academic integrity and our syllabus)	-200% or R in the course

Warning

Note

- It is **bad practice** in the industry to use **personally identifiable information (PII)** to name resources, in this project, please do **NOT** use your name or other PII to name GCP project, Azure resource group, Azure Container Registry hostname, etc. In this project, you will need to include these cloud resource IDs, and you do not want to submit your PII.
- Make sure you have finished the `Intro to Containers and Docker` and `Kubernetes and Container Orchestration` primers before starting this project. Otherwise, it will require much more time to complete this project.

Microservices and Overview

The Microservices Pattern

In this module, you will practice containerizing and deploying RESTful applications. During this process, you have to make decisions on how to organize the different components of our system. There are two common approaches to this problem. Combining all of the components to define a single deployable artifact (commonly referred to as monolithic applications) or decomposing our system into discrete components that are individually deployable (commonly referred to as microservices).

1. **Monolith** - A single logical application, under which a change to the system involves building and deploying a new version of the entire application.

2. **Microservices** - Loosely coupled applications, that generally communicate over a network and exist independently of each other.

The below table describes some of the common characteristics of `monoliths` and `microservices`:

Characteristic	**Monolith**	**Microservice**
Application Size	The size of the application may be large and will continue to increase over time as the functionalities grow.	The size of each service can stay minimal, because each component is standalone and its functionality is within a bounded scope.
Scalability	The entire application must be scaled at the same time, which results in increased cost and decreased utilization because not all the components of the application have the same scaling requirements.	Because the services are deployed independently, it's possible to scale an individual service separately. For example, if the login service becomes a bottleneck, we can scale that service only.
Modifiability	Each change to the application may require rebuilding and redeploying the whole service.	Microservices enable loose coupling between services. A service can get rebuilt and re-deployed independently.
Fault-tolerance (Availability)	Bugs or overload situations would affect the entire application.	Potential issues may only affect a subset of services.
Networking costs	Because monolithic applications usually run on a single resource, this can reduce the networking cost and reduce the time spent on service to service communication.	Microservices need to communicate using REST, SOAP, etc. The communication between services introduces additional network costs and latencies.

The WeChat Architecture

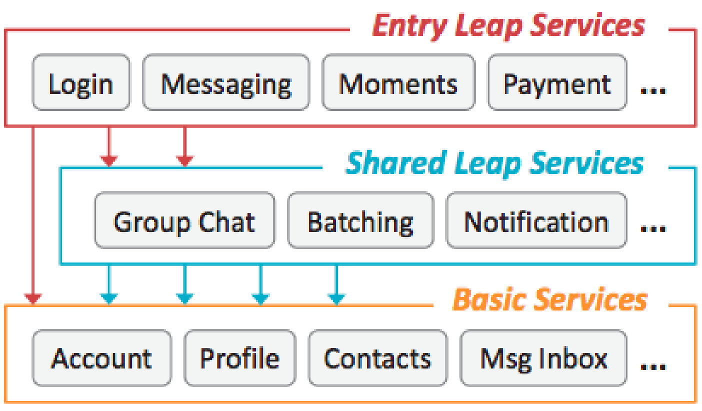


Figure 1: WeChat Microservices architecture [1 (https://www.cs.columbia.edu/~ruigu/papers/socc18-final100.pdf)]

Many businesses today are experiencing significant growth with respect to the number of users and the number of requests the users generate. The Overload Control for Scaling WeChat Microservices (https://www.cs.columbia.edu/~ruigu/papers/socc18-final100.pdf) white paper introduces the WeChat architecture, the techniques for handling request overload and the common approaches to building highly scalable services. As described in this whitepaper, WeChat is using the microservice pattern to develop a scalable and resilient application.

To describe the topology of microservices, WeChat refers to services as **nodes** and the connections between services as **edges**. This allows the WeChat designers to model the interaction between services as a directed acyclic graph (DAG (https://en.wikipedia.org/wiki/Directed_acyclic_graph)). For example, the group chat service will need to call the profile service to retrieve the user's profile information. By using this architectural pattern, it is possible to modularize each application and independently update, add features, etc. without having to change the design of any of the other microservices that have been identified as a bottleneck. Additionally, structuring applications in this manner aligns with agile methodology, where small teams can work on a specific service.

Converting a monolithic application into individual microservices does create additional complexity with respect to managing the microservices (i.e., handling separate deployments, monitoring, and API versions).

WeCloud Chat Scenario

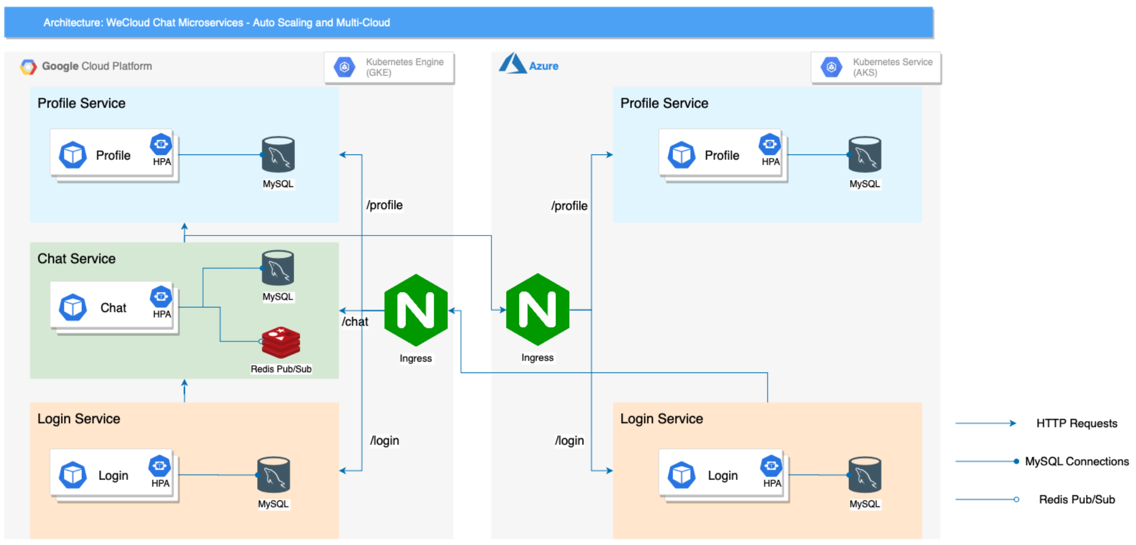


Figure 2: WeCloud Chat architecture

As shown in the diagram above, the WeChat architecture has many interconnected components; designing and implementing such a system would require a large number of teams to coordinate. In this project, to simplify the scenario, you are provided with the Java code that implements the login service, the group chat service, and the profile service. Together these services compose the WeCloud Chat application. The provided code is not ready to be deployed as Docker images. Your task is to complete the missing parts of these three services, package them as Docker images and deploy them to a Kubernetes cluster to form a scalable, fault-tolerant application.

The tools used in this project include the Java Spring Suite (Spring Boot, Spring Websocket, Spring Security, Spring JPA, Spring Cloud Ribbon), Maven, STOMP [1 (https://en.wikipedia.org/wiki/Streaming_Text_Oriented_Messaging_Protocol)] [2 (http://jmesnil.net/stomp-websocket/doc/)] (a sub-protocol over Websocket), Redis Pub/Sub (https://redis.io/topics/pubsub) (a messaging service feature from Redis), MySQL, Docker, Kubernetes, and Helm. You are free to explore the code implementations provided, but you will primarily focus on Docker files, Kubernetes YAML files, and Spring related configurations.

After you complete the project, you will achieve a microservices architecture as defined in the image above. We will deploy three services - the login, chat, and profile services - that form a simple DAG (`[LOGIN] -> [GROUP CHAT] -> [PROFILE]`). The profile service and login service will be replicated across multiple clouds to achieve fault tolerance.

Containerizing the Profile Service

Containerizing the Profile Service

Description

The profile service is a simple REST application that handles `GET` requests to fetch profile data from the database and responds in the JSON format. Each user profile object contains a username, name, gender, and age. The initial implementation of the profile service uses H2, which is an embedded database (https://en.wikipedia.org/wiki/Embedded_database), to reduce the number of components that need to be deployed. In the subsequent tasks, you will need to modify the Spring configuration and the source code to replace the embedded database(H2) with MySQL.

Tasks to Complete

Orchestrate and manage the architecture with Terraform

Although you are allowed to use the Web UI to provision your resources in this project, we strongly recommend that you use Terraform. We provide you with this prepared template that enables you to provision the required resources in this project.

```
wget https://clouddeveloper.blob.core.windows.net/s22-cloud-developer/docker-kubernetes/terraform/terraform.tgz && tar -xvf terraform.tgz
```

In order to manage GCP resources with Terraform, you must first set up Google Cloud SDK for authentication and create a GCP project to work with.

Setup Google Cloud SDK

1. Install Cloud SDK (https://cloud.google.com/sdk/docs/downloads-interactive).
2. Initialize Cloud SDK (https://cloud.google.com/sdk/docs/initializing#run_gcloud_init).

```
gcloud init
```

3. Create a GCP project. You may use an existing project, but make sure to follow the other steps.

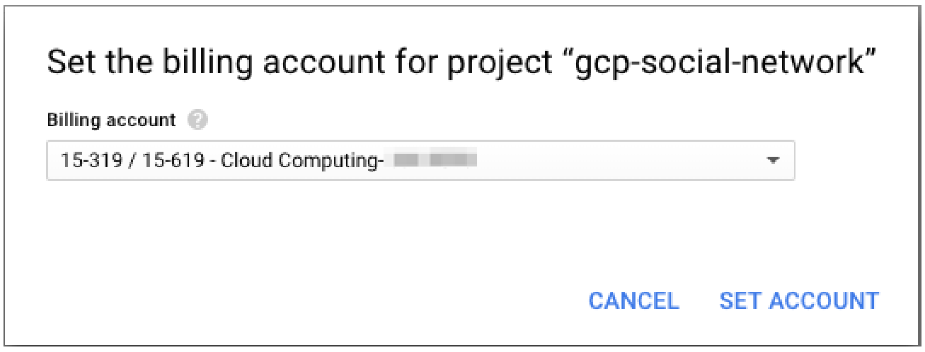
```
$ gcloud projects create --name gcp-docker-kubernetes
No project id provided.

Use [gcp-docker-kubernetes-xxxxxx] as project id (Y/n)?  Y
```

4. Note down the project ID as you will use the ID very often later, please note the difference between project ID `gcp-docker-kubernetes-xxxxxx` (e.g., `gcp-docker-kubernetes-123456`) and project name (`gcp-docker-kubernetes`). Project ID is a globally unique identifier on GCP.

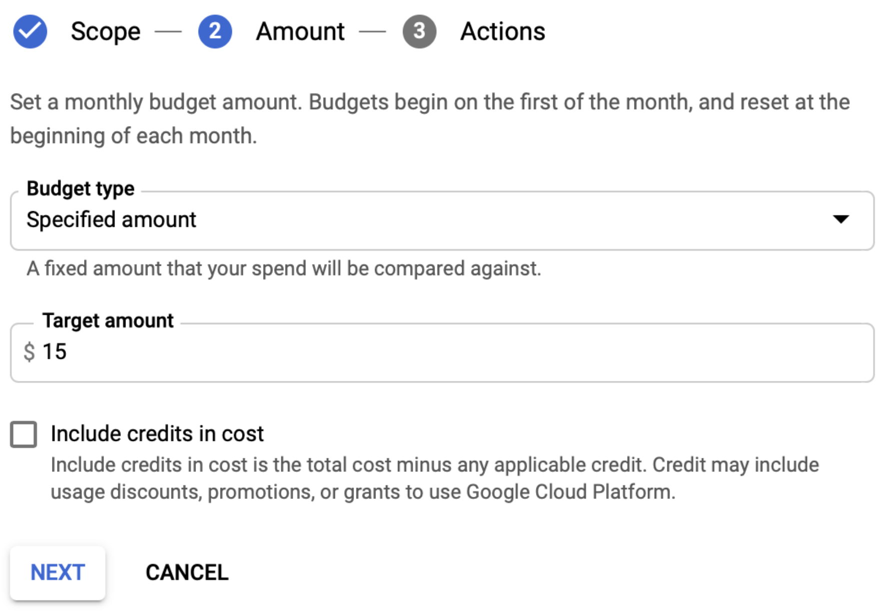
```
$ gcloud projects list
PROJECT_ID                    NAME                   PROJECT_NUMBER
gcp-docker-kubernetes-xxxxxx  gcp-docker-kubernetes  ...
```

5. Go to https://console.cloud.google.com/billing/projects (https://console.cloud.google.com/billing/projects). Click the "Actions" - "Change Billing" from the dropdown list. Select the billing account that belongs to this course (the account name may be different from the one in the picture) and click "SET ACCOUNT".


Set the billing account for project “gcp-social-network”

Billing account: 15-319 / 15-619 - Cloud Computing- [redacted]

6. From our previous experience, learners are likely to **exceed** their budget limits in this project. Therefore, we strongly recommend that you estimate and plan your resource usage before getting started. Budgets track expenses within a Google Cloud Platform project or billing account. Your budget can be a specified amount or based on previous spending. At the billing page, select the billing account name you enabled, and search and click "Budgets & alerts" - "CREATE BUDGET".
7. In the Scope section, choose your GCP project for "Projects" and "All services" for "Services". **In the Credits section, you must uncheck all the checkboxes to deselect all the options (e.g., "Discounts", "Promotions and others").** Budget tracks the total cost minus any applicable selected credits. Therefore, in order to calculate and monitor your actual spend before any credits are applied, do not select any credit options.
8. In the Amount section, choose "Specified amount" for "Budget type", and set the target amount as $15 (or lower).


✓ Scope — 2 Amount — 3 Actions

Set a monthly budget amount. Budgets begin on the first of the month, and reset at the beginning of each month.

Budget type: Specified amount

A fixed amount that your spend will be compared against.

Target amount: $ 15

☐ Include credits in cost

Include credits in cost is the total cost minus any applicable credit. Credit may include usage discounts, promotions, or grants to use Google Cloud Platform.

9. In the Actions section, set the alert threshold rules (e.g., 50%, 90%, 100%). Check "Email alerts to billing admins and users" so that you will receive emails when you exceed certain thresholds. Finally, click "Finish".
10. Configure gcloud and set the default region, zone, and project:

```
gcloud config set project gcp-docker-kubernetes-xxxxxx
gcloud config set compute/region us-east1
gcloud config set compute/zone us-east1-b
```

11. Enable the APIs for the GCP project: Google Compute Engine API (https://console.cloud.google.com/apis/library/compute.googleapis.com), Google Container Registry API (https://console.cloud.google.com/apis/library/containerregistry.googleapis.com) and Google Kubernetes Engine API (https://console.cloud.google.com/apis/library/container.googleapis.com). Please ensure that you are enabling APIs for the correct GCP project.

12. Configure Application Default Credentials (ADC) to allow the Google Auth library to view and manage your data across Google Cloud Platform services.

```
gcloud auth application-default login
```

13. Create a file named `terraform.tfvars` and set the values of the variables defined in `variables.tf`:

```
project = "gcp-docker-kubernetes-xxxxxx" # your gcp-docker-kubernetes project ID
```

Information

- **Always destroy the resources managed by terraform through `terraform destroy`.**
- Be careful when you use `terraform destroy`. It may destroy all your resources previously provisioned by your script, not only the latest one! To destroy a particular resource, you can use `terraform destroy --target=RESOURCE_TYPE.NAME`.
- When working on GCP, if you get any errors related to credentials or permissions, it is likely that you omitted some authentication or configuration step, e.g.

```
gcloud auth application-default login
gcloud config set project gcp-docker-kubernetes-xxxxxx
gcloud config set compute/region us-east1
gcloud config set compute/zone us-east1-b
```

Setting up the Student Instance

1. Launch Student Instance with Terraform.

```
terraform init # initialize a Terraform working directory if you have not done so
terraform apply
```

2. When `terraform apply` completes, Terraform will print the HTTP endpoint of the instance and the command to SSH into the instance. **Note that after `terraform apply` completes, it may take 1-2 minutes before you can access port 80.** Please be patient. You should be able to see the following:

```
student_instance_guide = Please open http://<student-vm-external-ip> in your web browser a
nd select Containers: Docker and Kubernetes.
Note that it may take 1-2 minutes before you can access it.
After the installation finishes, SSH into the instance using:
gcloud compute --project gcp-docker-kubernetes-xxxxxx ssh --zone us-east1-b clouduser@stud
ent-vm
```

3. Go to `http://<student-vm-external-ip>` and log into your student instance. Choose **Containers: Docker and Kubernetes**, this may take several minutes.

4. Use the `gcloud compute --project gcp-docker-kubernetes-xxxxxx ssh --zone us-east1-b clouduser@student-vm` command shared earlier to connect to the instance. You may be prompted to create a new SSH key if this is the first time you run this command. **Please do not use the GCP console or other commands to ssh into the instance as you may face issues of not seeing the project files or permission issues to run docker.**

5. Get familiar with the provided Maven project, you may import it into an IDE such as Intellij (https://www.jetbrains.com/help/idea/maven-support.html#maven_import_project_start) to view the source code and comments. Review the project files - including pom.xml, application.properties, ProfileController.java, Profile.java, ProfileRepository.java, ProfileApplication.java - before you move on to the next step.

6. Create a directory named `docker` in `profile-service-embedded-db/src/main/` and develop a `Dockerfile` in it to containerize the Profile service. The Dockerfile should **copy the source code into the container, compile the application in the container** and run the `JAR` file of the profile service. You may want to refer to the example in the `Intro to Containers and Docker` primer.

7. Build a docker image. You can use `docker build -f $DOCKERFILE_PATH -t $TAG $PROJECT_PATH` to build a docker image.

8. Start a container based on the image. You will need to use `docker run -p` to map port `8000` of localhost to the correct port of the container. The application server port is defined in `resources/application.properties`. The docker run command takes in arguments in the order `hostPort:containerPort`.

9. Once the application is containerized and a container is launched, you will be able to use cURL to send a GET request inside the VM via `http://localhost:8000/profile?username=$USERNAME`. For example, http://localhost:8000/profile?username=majd (http://localhost:8000/profile?username=majd). The

service will be available via `VM_IP:8000/profile?username=$USERNAME` , or you may visit `http://VM_IP:8000/profile?username=$USERNAME` in the browser. You can find multiple usernames for testing in `ProfileApplication.java` .

Hints

1. In order to be ready for the hands-on tasks in this project, make sure that you complete the Docker and Kubernetes primers and practice developing Dockerfiles and using the `docker` and `kubectl` CLI.
2. This Java application requires Java 8, do not use any other Java version during your development and testing. The student virtual machine uses Java 8.
3. If you want to use scp or third-party IDE to transfer files between the student-vm and your local machine, you can locate the SSH key files in the following locations (https://cloud.google.com/compute/docs/instances/adding-removing-ssh-keys#locatesshkeys):

   ```
   Linux and macOS
   Public key: $HOME/.ssh/google_compute_engine.pub
   Private key: $HOME/.ssh/google_compute_engine
   Windows:
   Public key: C:\Users\[USERNAME]\.ssh\google_compute_engine.pub
   Private key: C:\Users\[USERNAME]\.ssh\google_compute_engine
   ```

   where [USERNAME] is your username on your local workstation.
4. Postman, cURL, or `httpie` are tools that can be used to invoke REST endpoints and you should be familiar with at least one of them.
5. Spring guides (https://spring.io/guides) provide fully working solutions. You may consider referring to them throughout the project.
   - https://spring.io/guides/gs/spring-boot/ (https://spring.io/guides/gs/spring-boot/)
   - https://spring.io/guides/gs/accessing-data-jpa/ (https://spring.io/guides/gs/accessing-data-jpa/)
   - https://spring.io/guides/gs/spring-boot-docker/ (https://spring.io/guides/gs/spring-boot-docker/)
   - https://spring.io/guides/gs/messaging-stomp-websocket/ (https://spring.io/guides/gs/messaging-stomp-websocket/)
   - https://spring.io/guides/gs/client-side-load-balancing/ (https://spring.io/guides/gs/client-side-load-balancing/)
6. Consider `maven:3.8.3-jdk-8-slim` as the base image when developing your Dockerfiles. Also, write your own sh script to pack and run the application in the container.

What to Submit

Warning

In manual grading, we will grade your **last submission of each task separately**. Therefore, you should have your latest code ready for **your last submission** of each task rather than putting them all in the last task. It is okay if you forget to include all the files in previous submissions as long as you include all the files in the last submission of each task. It is not acceptable to submit the code of all tasks in the last task and expect us to manually grade them.

When submitting, please make sure the following files are in your project folder:

- the Dockerfile under `~/Project_Containers/task1/profile-service-embedded-db/src/main/docker`
- the citation file references under `~/Project_Containers/task1/` to include all the links that you referred to for completing this task.

How to Submit

1. Before you submit, deploy the profile service on the student-vm by creating a Docker container. You should test that the profile service accepts GET requests:

```
$ curl VM_IP:8000/profile?username=$USERNAME
```

2. Export your credentials as follows:

```
export SUBMISSION_USERNAME="your_submission_username"
export SUBMISSION_PASSWORD="your_submission_pwd"
```

3. Under the `task1` directory, run the submitter with `./submitter`.

Deploy the Profile Service with GCR and GKE

Using GCR and GKE to Deploy the Profile Service

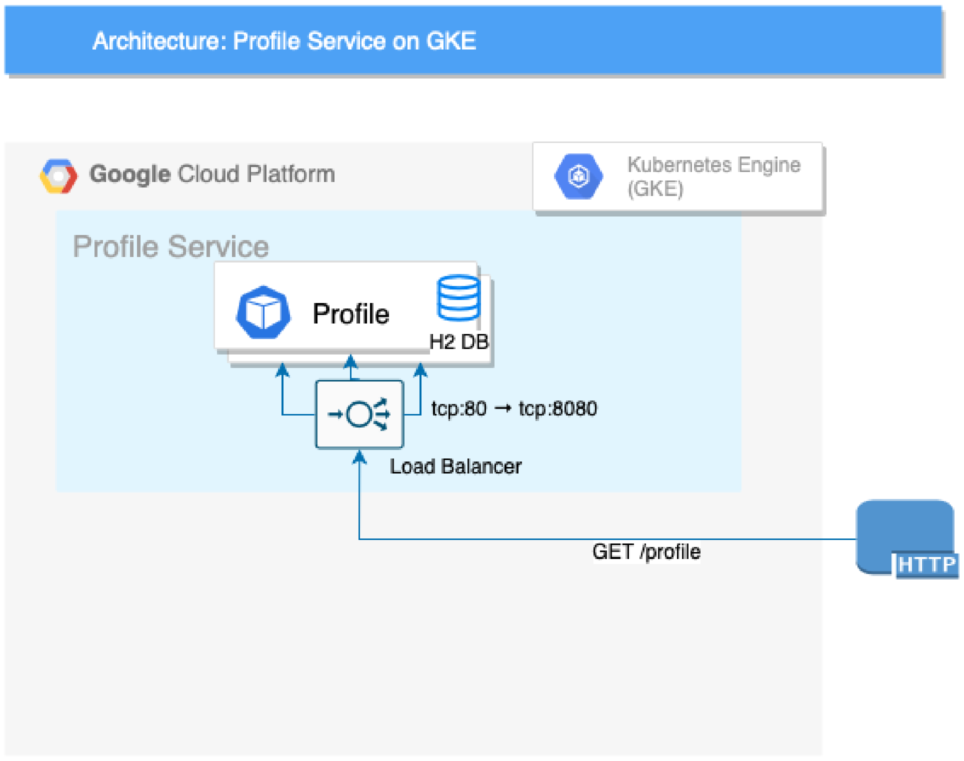


Figure 3: Profile Service deployed to GKE

Now that you have packaged the Profile service as a Docker image, you will deploy the service on a Kubernetes cluster in the cloud and make the service available to external clients. You will push the Docker images to the Google Container Registry (GCR (https://cloud.google.com/container-registry/)) and deploy the applications to the Google Kubernetes Engine (GKE (https://cloud.google.com/kubernetes-engine/)).

Tasks to Complete

1. Copy the `Dockerfile` you developed in Task 1 into `task2/profile-service-embedded-db/src/main/docker`. You might want to create the `docker` directory first.
2. Define the cluster state with YAML files in the `task2/profile-service-embedded-db/src/main/k8s` folder. The YAML files should define **three replicas** of the profile service and a load balancer service.

   You will need at minimum 2 YAML files - deployment.yaml (https://kubernetes.io/docs/concepts/workloads/controllers/deployment/) and service.yaml (https://kubernetes.io/docs/concepts/services-networking/service/).

The **load balancer** service must be named **spring-profile-service** and the deployment must be named **spring-profile-deployment**. The service port will be port `80` and the service should forward port `80` to port `8080` of the pod.

3. Build and tag the image with the example commands below. Please refer to the GKE Pushing and Pulling documentation (https://cloud.google.com/container-registry/docs/pushing-and-pulling) to decide the hostname to use for data centers in the United States.

```
docker build -f $DOCKERFILE_PATH -t $TAG $PROJECT_PATH
docker tag   [SOURCE_IMAGE]:[TAG]     [HOSTNAME]/[GCP_PROJECT_ID]/[IMAGE]:[TAG]
```

4. Push the profile service image built in the previous step to GCR. Refer to the Pushing and Pulling documentation (https://cloud.google.com/container-registry/docs/pushing-and-pulling) for further information.

```
gcloud auth login

gcloud auth configure-docker

docker push [HOSTNAME]/[GCP_PROJECT_ID]/[IMAGE]
```

5. Create a Kubernetes cluster in GCP. We will use the Googe Cloud SDK to create the cluster and retrieve the cluster credentials so you can access the cluster via kubectl.

```
$ gcloud auth application-default login
$ gcloud config set project $GCP_PROJECT_ID

# Show the available Kubernetes versions
$ gcloud container get-server-config --zone=us-east1-d

# You can modify the cluster name, machine type and the zone
$ CLUSTER_NAME="wecloudchatcluster"
$ gcloud container clusters create $CLUSTER_NAME --zone=us-east1-d --num-nodes=1 --machine
-type=custom-4-12288

$ gcloud container clusters get-credentials $CLUSTER_NAME --zone=us-east1-d
```

6. Navigate to the `k8s/` directory and use `kubectl` to deploy the profile service to the cluster with the YAML definitions:

```
# Using apply (as shown below) or create
$ kubectl apply -f .
```

7. Use the following commands to check the deployment results, including viewing the deployed Kubernetes objects and pod logs.

```
# List all services in the default namespace
$ kubectl get services

# List all pods in the default namespace
$ kubectl get pods

# Retrieve the logs for a specific pod
$ kubectl logs $POD_NAME

# The exec command will give you the terminal access inside of the pod
$ kubectl exec -it $POD_NAME -- /bin/sh
```

8. Similar to the previous task, test if the profile service is available. However, unlike the previous task, you will access the service via the load balancer.

```
$ curl http://$LOAD_BALANCER_EXTERNAL_IP/profile?username=$USERNAME
```

What to Submit

When submitting, please make sure the following files are in your project folder:

- the Dockerfile under `~/Project_Containers/task2/profile-service-embedded-db/src/main/docker`
- the K8s YAML files under `~/Project_Containers/task2/profile-service-embedded-db/src/main/k8s`
- the citation file references under `~/Project_Containers/task2/` to include all the links that you referred to for completing this task.

How to Submit

1. You will submit from your student VM on GCP.
2. Export your submission username and submission password:

```
$ export SUBMISSION_USERNAME="your_submission_username"
$ export SUBMISSION_PASSWORD="your_submission_pwd"
```

3. Your submission will not be successful if the expected service and deployment names are not returned by kubectl. Before running the submitter you should validate that you have the expected deployments and services by running the respective `kubectl` commands:

Kubernetes API objects on the GKE cluster	Names
Deployments (`kubectl get deployment`)	spring-profile-deployment
Service (`kubectl get svc`)	spring-profile-service kubernetes (the default service)

4. Use `kubectl` to validate expected deployment and service and cURL to check that you can access the profile service via the load balancer.
5. In the `task2` directory, execute the submitter `./submitter`.
6. In the next tasks, you will **NOT** need the profile deployment and the load balancer service created by kubectl, and failing to remove them can cause the submissions to fail in the next tasks. Once you complete Task 2, please delete the profile deployment and the load balancer service from the GKE cluster you created in this task.

```
$ cd ~/Project_Containers/task2/profile-service-embedded-db/src/main/k8s
$ kubectl delete -f .
```

Hints

1. For the Intellij users, the Kubernetes plugin (https://www.jetbrains.com/help/idea/kubernetes.html) will be very helpful in developing the YAML files.
2. You may refer to GKE's https://github.com/GoogleCloudPlatform/kubernetes-engine-samples/tree/master/hello-app/manifests (https://github.com/GoogleCloudPlatform/kubernetes-engine-samples/tree/master/hello-app/manifests) as a example of YAML definitions of service and deployment.

Using Helm Charts and Migrating to MySQL

Using Helm Charts and Migrating to MySQL

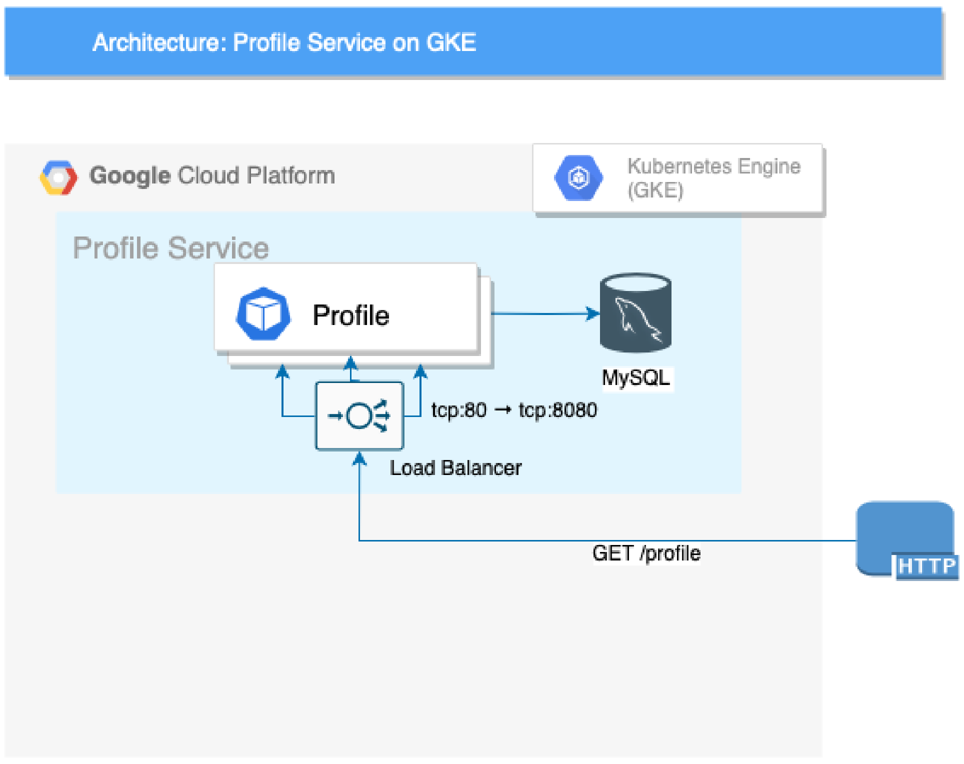


Figure 4: Profile Service with MySQL database

As mentioned earlier, the profile service was using an embedded database for simplicity. To make the profile service production-ready, we will migrate from an embedded database to a remote MySQL database. One microservice relying on additional services introduces another level of complexity, fortunately, Helm can help us deploy and configure applications with ease.

We use Kubernetes as a tool to manage and orchestrate containerized applications. As the complexity of your application increases, it is advantageous to use tools that help manage this complexity. Helm is a tool that maintains the relationship of Kubernetes objects within a cluster. Helm helps manage Kubernetes applications. Helm Charts are the primary means of organizing these relationships. Helm Charts helps you define, install, and upgrade even the most complex Kubernetes application. Helm Charts are composed of template files that can be parameterized to deploy uniquely configured versions of an application. Please read the Kubernetes primer to learn more about Helm and how to transform Kubernetes YAML defintions into Helm Charts before you continue.

The deployment architecture of the profile service is the same as task 2. At the end of this task, you will have a profile service, a profile deployment with three replicas, a MySQL service, and a MySQL deployment.

Tasks to Complete

1. Helm is installed on the student VM. You can validate the helm with the command:

```
$ helm version
```

2. Get the latest Chart information from chart repositories, similar to the `apt-get update` command in Linux.

```
$ helm repo add stable https://charts.helm.sh/stable

$ helm repo update
```

3. Use Helm to deploy the MySQL backend to the GKE cluster:

```
# Before you run the command, set the values of the environment variables $mysqlRootPasswo
rd, $mysqlUser and $mysqlPassword using `export`.

# Please avoid using digits only in these variables or the value will be recognized as int
eger instead of string which may cause error.

$ export mysqlRootPassword=...
$ export mysqlUser=...
$ export mysqlPassword=...

$ helm install mysql-profile --set mysqlRootPassword=${mysqlRootPassword},mysqlUser=${mysq
lUser},mysqlPassword=${mysqlPassword},mysqlDatabase=test stable/mysql
```

4. Validate the helm installation by running the following command. You should receive no error messages if the above steps were successful:

```
$ helm list
```

5. Update the profile service to use the MySQL backend instead of the embedded database.
    1. Update the `~/Project_Containers/task3/profile-service/src/main/resources/application.properties` and `pom.xml` by removing the properties for H2 and adding properties for MySQL. Note that the folder was named `profile-service-embedded-db` in Task 1 and Task 2, and the folder in Task 3 is named `profile-service`, e.g., do not directly drag and drop the Task 1 or Task 2 folder into `~/Project_Containers/task3/`.
    2. Remove the dependency on `com.h2database` and add a dependency of the `MySQL connector` in `~/Project_Containers/task3/profile-service/pom.xml`.

```
    <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.19</version>
    </dependency>
```

6. You will not need to modify any of the Java code, but you will have to **rebuild the Docker image and push it to GCR using a new tag!** Make sure you have copied the `/docker` folder from the previous tasks into the `/main` directory.
7. You can make use of the YAML files developed in the previous task as the starting point, and develop the profile Helm chart under the `~/Project_Containers/task3/profile-service/src/main/helm/profile/templates` directory. This time you **must use configMap (https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/#define-container-environment-variables-using-configmap-data)** for environment variables, failing to do so will result in score deduction during the manual grading.
8. Install the `profile` chart you developed. You must name the release exactly as `profile`:

```
$ helm install profile profile-service/src/main/helm/profile/
```

9. Check the name of the profile pods.

```
$ kubectl get pods
```

10. Check the logs to see if you have deployed the profile service successfully.

```
$ kubectl logs $POD_NAME
```

11. You can use `kubectl` to retrieve services, pods and application logs similar to what you did in the previous task.
12. After you confirm that the deployment is successful, make a GET request to verify the profile service:

```
$ curl http://LOAD_BALANCER_EXTERNAL_IP/profile?username=USERNAME
```

What to Submit

When submitting, please make sure the following files are in your project folder:

- the Dockerfile under `~/Project_Containers/task3/profile-service/src/main/docker`
- the Helm Chart files under `~/Project_Containers/task3/profile-service/src/main/helm/profile/templates`
- the `~/Project_Containers/task3/profile-service/src/main/resources/application.properties` and `~/Project_Containers/task3/profile-service/pom.xml` which include the MySQL dependency
- the citation file references under `~/Project_Containers/task3/` to include all the links that you referred to for completing this task.

How to Submit

1. You will submit from your student VM on GCP.
2. Export your submission username and submission password:

```
$ export SUBMISSION_USERNAME="your_submission_username"
$ export SUBMISSION_PASSWORD="your_submission_pwd"
```

3. Before running the submitter, validate that the required helm releases and kubernetes objects exist with the required names.

Helm objects on the GKE cluster	**Names**
Helm Releases (as returned by `helm list`)	mysql-profile profile

Kubernetes API objects on the GKE cluster	**Names**
Deployments (as returned by `kubectl get deployment`)	spring-profile-deployment mysql-profile
Services (as returned by `kubectl get services`)	mysql-profile spring-profile-service kubernetes (which is the default service)

4. Similar to the previous task, use kubectl and cURL, to verify that your profile service deployment is able to reply to requests sent to the load balancer.
5. In the `task3` directory, execute the submitter `./submitter`.
6. Once you complete Task 3, please delete the profile release:

```
$ helm uninstall profile
```

This ensures that the K8s cluster is at a clean state that is ready for the next tasks.

Troubleshooting

1. If you encounter the **ImagePullBackOff** error, please check if the image name and tag in the deployment YAML configuration match the ones pushed in the Google Container Registry.
2. If you encounter the **CrashLoopBackOff** error, please use `kubectl get pods` to get all pods and `kubectl logs POD_NAME` to log the output of specific pods to find error messages.

WeCloud Chat Microservices

WeCloud Chat Microservices - Putting it Together

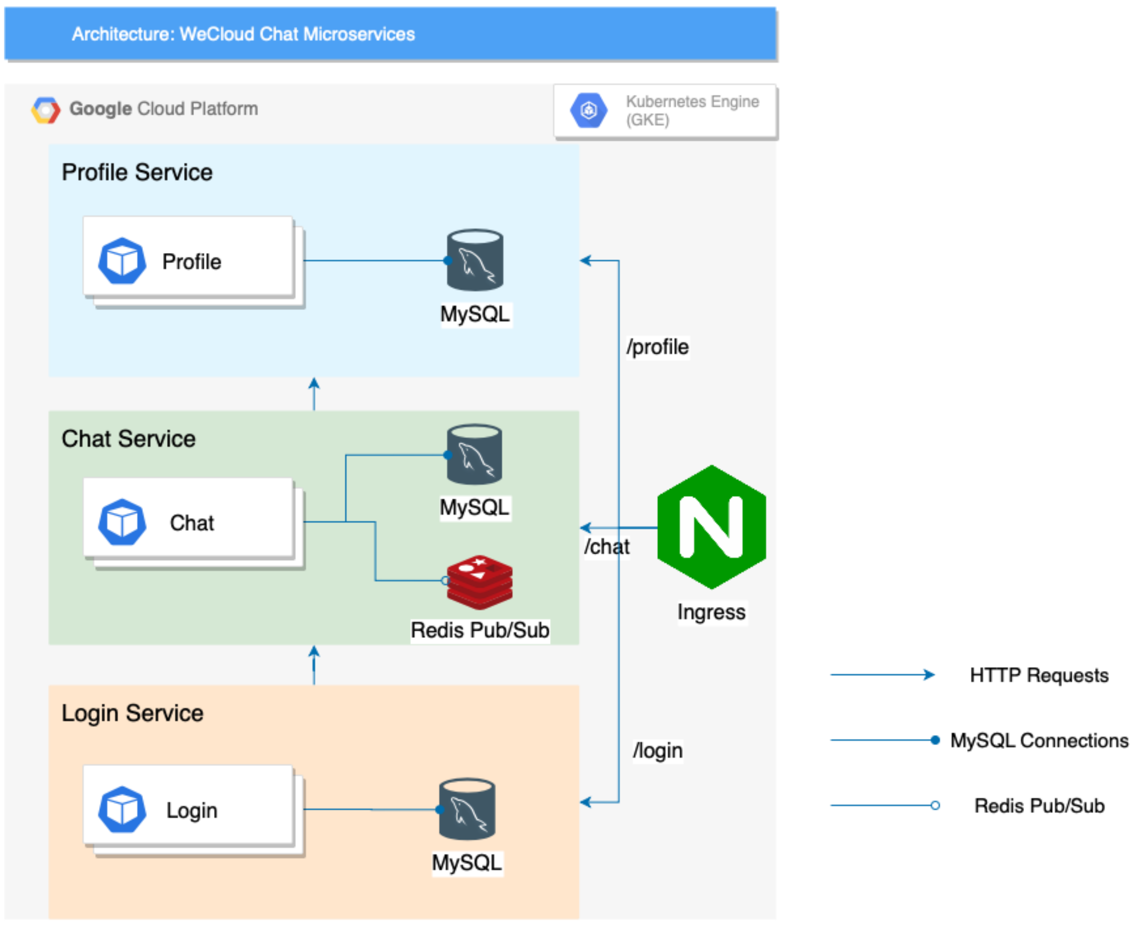


Figure 5: Chat, Login and Profile Services deployed to GKE

Congratulations! At this point, you have containerized an application as well as deployed it to a Kubernetes cluster running in the cloud. In this task, you will containerize the login service and the chat service similarly and integrate them into the full architecture.

In addition, you need to enable external access to the chat service and the login service. In Task 3, you used a Kubernetes service with the service `type` as `LoadBalancer`, which creates an external load balancer that points to a Kubernetes service in the cluster. Although it is possible to create more load balancers similarly for the chat service and the login service, having multiple external load balancers can be inefficient. Additional public IP addresses on managed K8s clusters can incur extra cost. Besides, having multiple public IPs complicates the usage and the maintenance of the application compared to sharing the same external IP for the application in a holistic approach.

Kubernetes supports a high-level abstraction called Ingress (https://kubernetes.io/docs/concepts/services-networking/ingress/), which allows simple traffic routing, e.g., based on URL or hostname. Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource. An ingress is a core concept of Kubernetes but is always implemented by a third party proxy. These implementations are known as ingress controllers. Ingress controller is responsible for reading the Ingress Resource information and processing that data accordingly. The ingress controller you will use in this project is Ingress controllers (https://kubernetes.github.io/ingress-nginx/deploy/).

Below there is a brief introduction of the Chat service and the Login service to help you understand the components of each service. There are no actions you need to take other than reading for the "Chat Service" and "Login Service" sections. The task for you to complete will be specified in the "Tasks to Complete" section.

Information

(Optional Reading) Implementation of a Real-time Distributed Chat Service

The group chat service uses primarily Stomp and WebSockets to enable real-time communications. WebSockets are better than raw HTTP connections for chat rooms as they provide a two-way full-duplex connection between the client and server. Additionally, WebSockets can reduce the amount of network traffic. HTTP headers are typically around 700 bytes, while WebSocket headers are around 3 bytes. Google Docs and online multiplayer games are other good applications of WebSockets (https://www.infoworld.com/article/2609720/application-development/9-killer-uses-for-websockets.html).

If too many users access the group chat service at the same time, the service will become overloaded and prone to failure. Hence, scalability is a must for this real-time chat service.

Because the number of connections each instance of the application can handle concurrently is limited, it is better to design a service that can be scaled horizontally.If the group chat service is scaled horizontally, all the replicas must synchronize the data. A common solution to this problem is to use the PubSub pattern, in which there is a messaging service or an in-memory database to synchronize the data among replicas.

Several open-source tools support PubSub semantics, including Redis, RabbitMQ, and Kafka. There are differences between the available options in terms of ease of use, reliability and performance which should be considered when choosing a solution. The following selection of articles discusses some of these considerations.

1. https://tech.trello.com/why-we-chose-kafka/ (https://tech.trello.com/why-we-chose-kafka/)
2. https://blog.tuleap.org/how-we-replaced-rabbitmq-redis (https://blog.tuleap.org/how-we-replaced-rabbitmq-redis)
3. https://www.ibm.com/cloud/learn/message-queues (https://www.ibm.com/cloud/learn/message-queues).

(Reading Only) Chat Service

In this project , the provided application uses Redis' PubSub to synchronize the chat messages among replicas as it can be easily deployed and managed in a Kubernetes cluster. Besides, the provided application uses MySQL to persist the chat messages, so that users don't lose their chat history.

(Reading Only) Login Service

Once you have containerized and deployed the group chat service, you will containerize and deploy the login service similarly. Information security is one of the most important aspects of the login service and it's important to protect the identity information of your users. Common attacks on software systems include SQL injection, Cross-site scripting (XSS), Cross-site request forgery (CSRF), etc. Adopting an authentication and authorization framework can help address many security vulnerabilities. The provided application adopts Spring Security.

Tasks to Complete

Create the Ingress resource

1. Create an NGINX ingress controller by running the following command

```
$ helm install my-nginx stable/nginx-ingress
```

2. Create an Ingress resource file named as **ingress.yaml** under `~/Project_Containers/task4/Ingress` . Please refer to the K8s Ingress Resource documentation (https://kubernetes.io/docs/concepts/services-networking/ingress/#the-ingress-resource) to develop ingress.yaml. You should use **networking.k8s.io/v1** as apiVersion. In the Ingress Resource documentation, you may notice usage of Kubernetes annotations (`metadata.annotations`) in the example YAML snippet to customize the behavior of the Ingress resource. Note that in this project you do **NOT** need to use any annotations for the Ingress resource. You need to develop the `spec.rules` section per the requirements in the table below:

path	serviceName	servicePort
/chat	spring-chat-service	80
/login	spring-login-service	80

/profile	spring-profile-service	80

3. Create an Ingress resource by running the following command

```
$ kubectl create -f ingress.yaml
```

4. Check the state of the Ingress you just added with

```
$ kubectl get ingress
```

Create the profile service

1. Reuse the files in `profile-service` developed in the previous tasks. You will need to copy the profile-service folder to the task4 folder, e.g.,

```
$ cp -r ~/Project_Containers/task3/profile-service ~/Project_Containers/task4
```

2. **Delete the profile service you installed with Helm in Task 3 if you haven't.**
3. In this task, you will use Ingress to handle external traffic, so there is no more need for an external load balancer exclusively for the profile service. Therefore, if you reuse the profile-service solution in the previous tasks, you need to change the **type** in **service.yaml** from **LoadBalancer** to **NodePort**. NodePort enables you to set up your own load balancing solution.
4. Install the profile service with the updated **service.yaml** file.

```
$ helm install profile profile-service/src/main/helm/profile/
```

Create the chat service

1. (Optional) If you are interested in the implementation of the chat Spring application, review the application code as listed below. You do not need to make any changes to these files to complete this task:
   - `group-chat-service/src/main/java`
   - `group-chat-service/src/main/resources`

   Java code contains comments that will help you understand the functionality of the chat service.
2. Deploy the chat service. Use the names **mysql-chat** for the MySQL helm chart, **chat** for the name of the helm chart, **spring-chat-deployment** for the deployment, and **spring-chat-service** for the service. The deployments should use three replicas and the service should be a **NodePort** service. Expose port **80** for the service, similar to the profile service.
3. As mentioned before, the chat service uses Redis to synchronize the data among replicas. For the Redis deployment, the YAML file (`group-chat-service/src/main/helm/chat/templates/redis.yaml`) is provided for you and you do not need to make any changes to it.
4. A configMap.yaml is also provided in the same folder as `redis.yaml`. You can use this as a starting point and you may add more environment variables.
5. The MySQL data source configuration is provided in `group-chat-service/src/main/resources/application.properties`. You do not need to change `application.properties`. However, you should refer to this file to decide what the K8s environment variables you need to add into `configMap.yaml`.
6. Install the helm chart of the chat service by running the following command. You must use **chat** as the name of the release:

```
    $ helm install chat group-chat-service/src/main/helm/chat/
```

7. After deploying the chat service, it can be accessed via `http://NGINX_INGRESS_CONTROLLER_LOAD_BALANCER_EXTERNAL_IP/chat`.

Create the login service

1. (Optional) Similar to the chat and profile services, you can review the implementation of the login Spring application:
    - `login-service/src/main/java`
    - `login-service/src/main/resources`
2. Deploy the login service. Use the names **mysql-login** for the MySQL helm chart, **login** for the name of the helm chart, **spring-login-deployment** for the deployment, and **spring-login-service** for the service. The deployments should use three replicas and the service should be a **NodePort** service. Expose port **80** for the service, similar to the profile service. Upon a successful login, you should redirect the user to the Ingress controller load balancer's external IP.
3. Install the Helm chart of the login service via Helm. You must use **login** as the name of the release:

```
  $ helm install login login-service/src/main/helm/login/
```

4. Verify the login service. The login Spring application uses Spring Security which adds a CSRF token to the login page to prevent CSRF (https://en.wikipedia.org/wiki/Cross-site_request_forgery) attack; the CSRF token makes it more difficult to test the requests using tools like cURL or Postman. We suggest that you directly visit the login page via the browser. `LoginApplication.java` contains the details of the test users that you can use to log into the UI for testing.

What to Submit

When submitting, please make sure the following files are in your project folder:

- the Dockerfiles under
    - ~/Project_Containers/task4/profile-service/src/main/docker
    - ~/Project_Containers/task4/group-chat-service/src/main/docker
    - ~/Project_Containers/task4/login-service/src/main/docker
- the Helm Chart files under
    - ~/Project_Containers/task4/profile-service/src/main/helm/profile/templates
    - ~/Project_Containers/task4/group-chat-service/src/main/helm/chat/templates
    - ~/Project_Containers/task4/login-service/src/main/helm/login/templates
- the citation file references under `~/Project_Containers/task4/` to include all the links that you referred to for completing this task.

How to Submit

1. You will submit from your student VM on GCP.
2. Export your submission username and submission password:

```
$ export SUBMISSION_USERNAME="your_submission_username"
$ export SUBMISSION_PASSWORD="your_submission_pwd"
```

3. Before running the submitter, validate that the required Helm releases and Kubernetes objects exist with the required names.

Helm objects on the GKE cluster	Names
Helm Releases (`helm list`)	chat login my-nginx mysql-chat mysql-login mysql-profile profile

Kubernetes API objects on the GKE cluster	Names

Deployments (`kubectl get deployment`)	my-nginx-nginx-ingress-controller my-nginx-nginx-ingress-default-backend mysql-chat mysql-login mysql-profile redis-deployment spring-chat-deployment spring-login-deployment spring-profile-deployment
Services (`kubectl get services`)	kubernetes (the default service) my-nginx-nginx-ingress-controller my-nginx-nginx-ingress-default-backend mysql-chat mysql-login mysql-profile redis-service spring-chat-service spring-profile-service spring-login-service

4. Using kubectl, cURL and/or the browser to verify that your service deployments are able to reply to requests via the load balancer and all the services can communicate with one another.
5. In the `task4` directory, execute the submitter `./submitter`.

Hints

1. Initialization of the services will take time subject to the application size and the cluster configuration (e.g., CPU, memory). Validate that your services are running and are able to handle web requests before you submit.
2. You can use `kubectl` to get the logs of the pods to identify issues.
3. Pressing the show details button on the group chat service UI will invoke the call to the profile service; you should test this via the UI to confirm that the chat service can make requests to the profile service.
4. The provided configMap.yaml for the chat service can be a starting point for you to decide all the Kubernetes environment variables in this task.
5. It may take around 1 minute for the submitter to finish, please wait patiently. If any service is not working, the submission may take longer because of the request timeout.

Auto-scaling, Multiple Cloud Deployment and Fault-tolerance

Auto-scaling, Multiple Cloud Deployment and Fault-tolerance

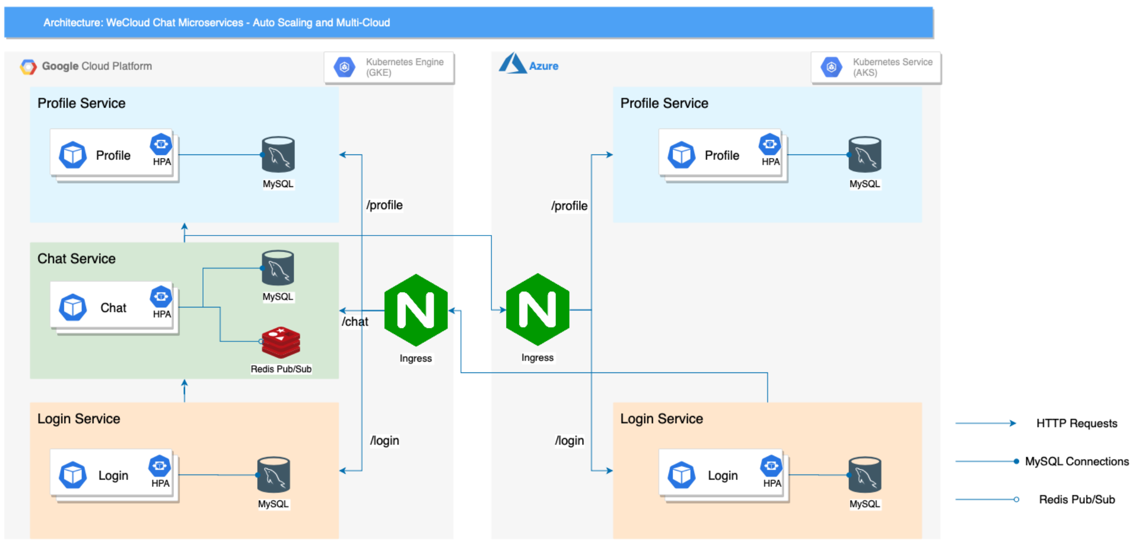


Figure 6: Login and Profile Service replicated to AKS

You have successfully built and deployed the WeCloud Chat microservices that collectively works as a holistic application.

In this task we will take advantage of Kubernetes features that make it easy to horizontally scale your microservices based on pod metrics. The Horizontal Pod Autoscaler (HPA) handles the complexities of triggering auto-scaling for deployments:

1. Horizontal Pod Autoscaler (https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/)
2. Autoscaling in Kubernetes (https://kubernetes.io/blog/2016/07/autoscaling-in-kubernetes/)

Once you have created the HPAs for each service, we will introduce the idea of multi-cloud deployments and fault-tolerance. While cloud platforms like AWS, Azure and GCP are very reliable and do not encounter long term outages frequently, it's possible that one cloud platform could suffer from a temporary or transient outage in a specific zone or region. To mitigate the impact of such an outage in one cloud service provider, we can deploy our application to multiple clouds. This will achieve fault-tolerance and potentially improved performance (if the traffic is routed between the different cloud deployments based on some heuristic).

We will experiment with multi-cloud deployments. Specifically, we will replicate the login and profile services to Azure. Replicating the chat service across multiple clouds would require mechanisms to share the Redis PubSub data and is beyond the scope of this project.

Tasks to Complete

1. Copy the 3 working services and Ingress resource from the task4 folder to the task5 folder.

```
cp -r task4/Ingress task4/profile-service task4/group-chat-service task4/login-service tas
k5
```

2. In this step, you will create a **HorizontalPodAutoscaler (HPA)** for every deployment - the profile, chat and login services. You should monitor the CPU utilization of the deployments to determine an appropriate **targetCPUUtilizationPercentage** level to ensure scaling will occur under a high load.

   WeChat uses queue-based request routing and supports different priorities for different kinds of requests, so they may use additional metrics for scaling. CPU utilization is a reasonable metric to use to trigger scaling in this project. You **only** need to make changes to the YAML files under the **helm** folder of each service.

   Use the HPA template below as the starting point. Please pay attention to the apiVersion, a GKE cluster supports multiple HorizontalPodAutoscaler API object versions such as stable version `autoscaling/v1` and beta version `autoscaling/v2beta2` . In this project, you must the stable version `autoscaling/v1` .

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

You need to add an HPA to each **deployment.yaml** file except for the Redis deployment in redis.yaml, i.e., you will add 3 HorizontalPodAutoscaler sections in total (1 for profile, 1 for chat, and 1 for login). **In order to get the current CPU Usage in the HPA, you must add the following in the containers object in each of the deployment.yaml files.**

```
resources:
 requests:
   cpu: 200m
```

3. Use **Helm** to update each service in the GKE cluster, **DO NOT** use `kubectl`.

```
$ helm upgrade profile profile-service/src/main/helm/profile/

$ helm upgrade login login-service/src/main/helm/login/

$ helm upgrade chat group-chat-service/src/main/helm/chat/
```

4. Validate that the HPA contains two percentage numbers - **Target CPU utilization** and **Current CPU utilization** by running `kubectl describe hpa`.
5. Once you have created the HPAs for each service on the GKE cluster, you will create an AKS cluster to implement a multi-cloud deployment. First, create a new AKS cluster with ACR integration (https://docs.microsoft.com/en-us/azure/aks/cluster-container-registry-integration). Azure Container Registry (ACR) name should be globally unique. Creating an AKS cluster may take up to 30 minutes to complete, so be patient.

```
$ az login

# Check and set the Azure subscription.
$ az account list --output table --refresh
$ az account set --subscription <name or id>

# Initialize these variables

# Container registry resource group
$ export RESOURCE_GROUP=...

# Azure Container registry, the name must be in all lowercase
$ export ACR_NAME=...

# AKS Cluster name
$ export CLUSTER_NAME=...

$ az group create -n ${RESOURCE_GROUP} -l eastus

$ az acr create -n ${ACR_NAME} -g ${RESOURCE_GROUP} --sku basic

$ az aks create -n ${CLUSTER_NAME} -g ${RESOURCE_GROUP} --attach-acr ${ACR_NAME}  --genera
te-ssh-keys
```

6. Before pushing and pulling container images, you must log in to the container registry with the `az acr login` command.

```
az acr login --name ${ACR_NAME}
# Expected output:
# Login Succeeded
```

7. Run the following commands to set up the access to the AKS cluster. Note that you should always remember to run this `az aks get-credentials` command once before you can use `kubectl` to manage an AKS cluster.

```
az aks get-credentials --resource-group=${RESOURCE_GROUP} --name=${CLUSTER_NAME}
# Expected output:
# Merged "${CLUSTER_NAME}" as current context in .../.kube/config
```

8. Now you have set up the connection to multiple Kubernetes clusters, you need to switch between Kubernetes clusters by using the following commands:

```
$ kubectl config get-contexts  # display list of contexts (i.e., clusters)

$ kubectl config use-context GCP_OR_AZURE_CONTEXT  # set the default context (i.e, set the default cluster you will work on)
```

9. After you have **changed the context to Azure Kubernetes clusters**, create an ingress controller by running the following code:

```
$ helm install my-nginx stable/nginx-ingress
```

10. Create an Ingress resource file **ingress.yaml** under **/home/clouduser/Project_Containers/task5/Ingress_Azure**. The path and service should have the following mapping:

path	serviceName	servicePort
/login	spring-login-service	80
/profile	spring-profile-service	80

11. Create an Ingress resource by running the following code

```
$ kubectl create -f ingress.yaml
```

12. Create a new folder under `profile-service/src/main/` called **helm-multi-cloud**. Copy the profile chart from `profile-service/src/main/helm` folder to `profile-service/src/main/helm-multi-cloud`. You may use the helm charts and YAML files developed previously as the starting point, but do not forget to **update the image registry**. You will need to use the Azure's registry address, which will be of the form `azurecr.io`.

13. Add the HorizontalPodAutoscaler for the profile service on the AKS cluster (https://docs.microsoft.com/en-us/azure/aks/tutorial-kubernetes-scale#autoscale-pods). Please pay attention to the apiVersion, the apiVersion supported by the AKS cluster is `autoscaling/v1`.

```
# Sample for defining Azure HPA:  [Hint: you may need to set specific CPU requests ]

apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: azure-vote-back-hpa
spec:
  maxReplicas: 10 # define max replica count
  minReplicas: 3  # define min replica count
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: azure-vote-back
  targetCPUUtilizationPercentage: 50 # target CPU utilization

---

apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: azure-vote-front-hpa
spec:
  maxReplicas: 10 # define max replica count
  minReplicas: 3  # define min replica count
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: azure-vote-front
  targetCPUUtilizationPercentage: 50 # target CPU utilization
```

14. Install a separate MySQL service for the profile service on the AKS cluster.
15. After you have installed MySQL for the profile service on Azure, revisit the `configMap.yaml` of the profile service on Azure and decide if you need to update any environment variables.
16. Install the profile service on Azure and validate that it returns the expected results. Since we are using Azure at this step, be sure to **push the profile docker image to ACR**. Otherwise, the pods may not be able to pull the image from the container registry.
17. Update the chat service on GCP so that it can communicate with the services running in Azure.
    1. Modify the `configMap.yaml` of the chat service on GCP and add the external IP of the load balancer of the Ingress service on Azure.
    2. **Switch the context to the GKE cluster**, and reinstall the chat service on the GCP.
18. Finally, deploy the login service to Azure.
    1. **Switch the context to the AKS cluster**.
    2. Create a new folder under `login-service/src/main/` called **helm-multi-cloud**.
    3. Copy the login chart from `login-service/src/main/helm` folder to `login-service/src/main/helm-multi-cloud`. Make sure to update the the image in `deployment.yaml` with the correct name.
    4. Add the HPA with the apiVersion as `autoscaling/v1`.
    5. Install a separate MySQL service for the login service on Azure.
    6. After you have installed MySQL for the login service on Azure, revisit the `configMap.yaml` of the login service on Azure and decide if you need to update any environment variables.
    7. Install the login service to Azure using Helm.

What to Submit

When submitting, please make sure the following files are in your project folder:

- the Dockerfiles under

  - ~/Project_Containers/task5/profile-service/src/main/docker
  - ~/Project_Containers/task5/group-chat-service/src/main/docker
  - ~/Project_Containers/task5/login-service/src/main/docker
- the Helm Chart files under
  - ~/Project_Containers/task5/profile-service/src/main/helm/profile/templates
  - ~/Project_Containers/task5/profile-service/src/main/helm-multi-cloud/profile/templates
  - ~/Project_Containers/task5/group-chat-service/src/main/helm/chat/templates
  - ~/Project_Containers/task5/login-service/src/main/helm/login/templates
  - ~/Project_Containers/task5/login-service/src/main/helm-multi-cloud/login/templates
- the citation file references under `~/Project_Containers/task5/` to include all the links that you referred to for completing this task.

How to Submit

1. You will submit from your student VM on GCP.
2. Export your submission username and submission password:

```
$ export SUBMISSION_USERNAME="your_submission_username"
$ export SUBMISSION_PASSWORD="your_submission_pwd"
```

3. Before running the submitter, validate that the required helm releases and kubernetes objects exist with the required names.

Helm objects on the GKE cluster	**Names**
Helm Releases (`helm list`)	mysql-chat mysql-profile mysql-login my-nginx chat profile login

Helm objects on the AKS cluster	**Names**
Helm Releases (`helm list`)	my-nginx mysql-profile mysql-login profile login

Kubernetes API objects on the GKE cluster	**Names**
Deployments (`kubectl get deployment`)	my-nginx-nginx-ingress-controller my-nginx-nginx-ingress-default-backend mysql-chat mysql-login mysql-profile redis-deployment spring-chat-deployment spring-login-deployment spring-profile-deployment

Services (`kubectl get services`)	kubernetes (the default service) my-nginx-nginx-ingress-controller my-nginx-nginx-ingress-default-backend mysql-chat mysql-login mysql-profile redis-service spring-chat-service spring-login-service spring-profile-service
HPAs (`kubectl get hpa`)	spring-login-autoscaling spring-profile-autoscaling spring-chat-autoscaling

Kubernetes API objects on the AKS cluster	**Names**
Deployments (`kubectl get deployment`)	my-nginx-nginx-ingress-controller my-nginx-nginx-ingress-default-backend spring-profile-deployment mysql-profile spring-login-deployment mysql-login
Services (`kubectl get services`)	kubernetes (the default service) my-nginx-nginx-ingress-controller my-nginx-nginx-ingress-default-backend mysql-login mysql-profile spring-login-service spring-profile-service
HPAs (`kubectl get hpa`)	spring-login-autoscaling spring-profile-autoscaling

4. Using kubectl, cURL and the browser to verify that your service deployments are able to reply to requests via the load balancer and all the services can communicate with one another.
5. In the `task5` directory, execute the submitter `./submitter` .

Hints

1. `kubectl get deployment` will tell you the number of desired, current and available replicas for a given deployment. `kubectl get HorizontalPodAutoscaler` will return a summary of the autoscaling policy you have defined in your YAML template. `kubectl get hpa` will additionally show the CPU utilization for the deployment each autoscaler is monitoring.
2. Note that you should confirm you are in the expected context when running any helm or kubectl commands. `kubectl config get-contexts` lists the contexts available and `kubectl config use-context CONTEXT_NAME` will switch between available contexts.
3. For the Azure HPA objects, you will need to specify the CPU usage for the **deployment.yaml**, otherwise, you might not be able to get the CPU metrics. **Remember to use API version autoscaling/v1 for GCP and Azure. Please refer to the official documentation (https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/)!**
4. If you are failing the HPA for Azure, revisit step 2 and ensure you have added the resources object under the containers object in each of the deployment.yaml files.

Troubleshooting

1. If you get '-' as your score, please make sure the services are all running and connected, double check the endpoint of both profile services are reflected in the deployment.yaml for chat service.
2. Deleting an Azure cluster sometimes cannot delete the contexts in kubectl. If running `az aks get-credentials` returns the error message `A different object named XXX already exists in YYY` you should unset the previous context values by running:

```
$ kubectl config unset users.[USER]
$ kubectl config unset contexts.[CONTEXT]
$ kubectl config unset clusters.[CLUSTER]
```

3. If you encountered "az: error: unrecognized arguments: --attach-acr myContainerRegistry", double-check you are using the latest version of Azure CLI.
4. If you encounter the **ImagePullBackOff** error in Azure, please refer to the official Azure CLI documentation (https://docs.microsoft.com/en-us/azure/container-registry/container-registry-authentication) and docker authentication (https://docs.microsoft.com/en-us/azure/container-registry/container-registry-get-started-azure-cli) for more information. It is very likely that you did not follow AKS to ACR integration (https://docs.microsoft.com/en-us/azure/aks/cluster-container-registry-integration).
5. If you encounter the **CrashLoopBackOff** error in Azure, double-check you enabled the MySQL backend.

Domain Name with Azure Front Door

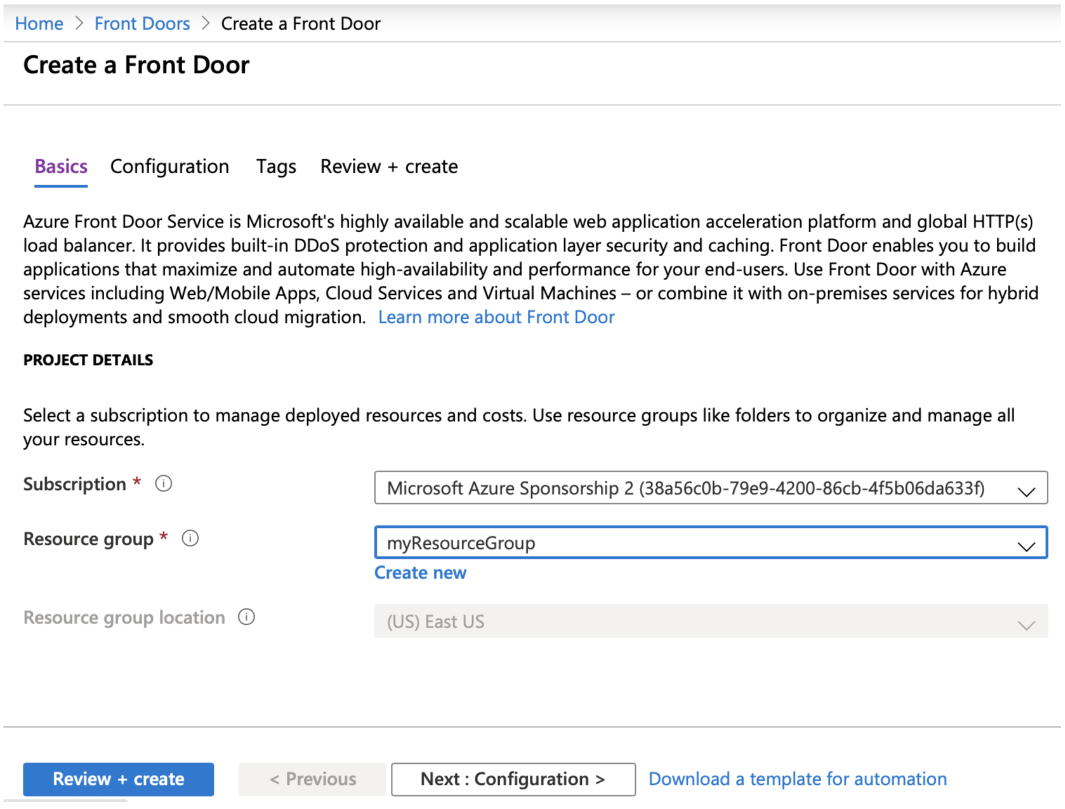
Domain Name with Azure Front Door

In the real world, it is not practical to visit websites with IP addresses since IPs are hard for human beings to remember. Imagine that you have to remember all your friend's phone numbers to make phone calls! Domain Name System (DNS) is like a phone book for the internet. If you know a person's name but don't know their telephone number, you can simply look it up in a phone book. What's more, your friend may have multiple phone numbers, and in most cases, you don't really care which one you dialed as long as you can reach your friend. In our previous tasks, we have deployed the profile service, the login service and the chat service on Azure and GCP, and we are left with two IPs. Similar to the phone book example, we can define routing rules to map a single domain name to these two IPs. In this task, you will use Azure Front Door Service to achieve a path-based routing to the web application deployed on Azure and GCP.

As per official Microsoft docs, "Azure Front Door Service enables you to define, manage, and monitor the global routing for your web traffic by optimizing for best performance and instant global failover for high availability. With Front Door, you can transform your global (multi-region) consumer and enterprise applications into robust, high-performance personalized modern applications, APIs, and content that reach a global audience with Azure".

Tasks to Complete

1. Go to **Azure Front Door Service** on Azure console (https://portal.azure.com/#blade/HubsExtension/BrowseResourceBlade/resourceType/Microsoft.Network%2Ffrontdoors). Click **"Create"** to create a new Azure Front Door Service configuration.
2. Under the **"Basic"** tab, choose the subscription and the resource group you created in Task 5. Click **"Next: Configuration"** to go to the next section.


Home > Front Doors > Create a Front Door

Create a Front Door

Basics | Configuration | Tags | Review + create

Azure Front Door Service is Microsoft's highly available and scalable web application acceleration platform and global HTTP(s) load balancer. It provides built-in DDoS protection and application layer security and caching. Front Door enables you to build applications that maximize and automate high-availability and performance for your end-users. Use Front Door with Azure services including Web/Mobile Apps, Cloud Services and Virtual Machines – or combine it with on-premises services for hybrid deployments and smooth cloud migration. Learn more about Front Door

PROJECT DETAILS

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	Microsoft Azure Sponsorship 2 (38a56c0b-79e9-4200-86cb-4f5b06da633f)
Resource group *	myResourceGroup
	Create new
Resource group location	(US) East US

Review + create | < Previous | Next : Configuration > | Download a template for automation

Figure: Create Azure Front Door Service.

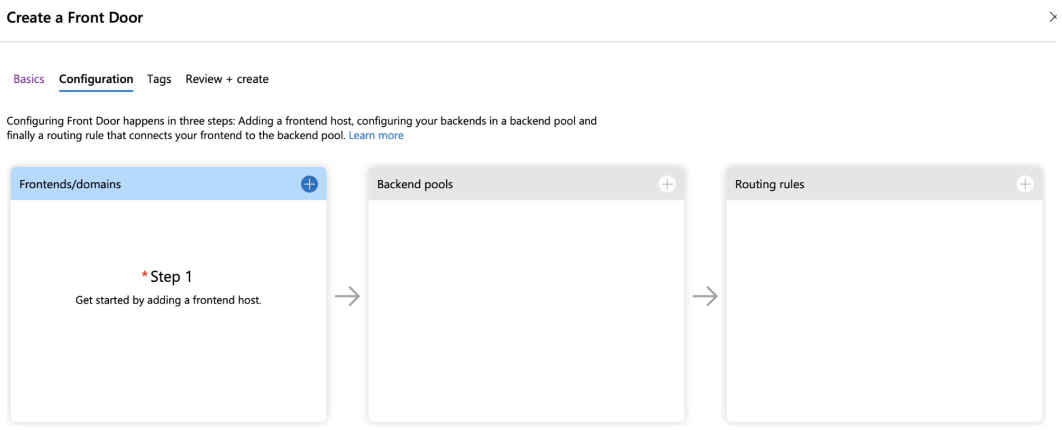
3. The configuration for Azure Front Door Service consists of 3 steps:

   a) Frontends/domains

   b) Backend pools

   c) Routing rules

   We will first assign a **domain name** to the services. Click the **"+"** sign of the **"Frontends/domains"**.

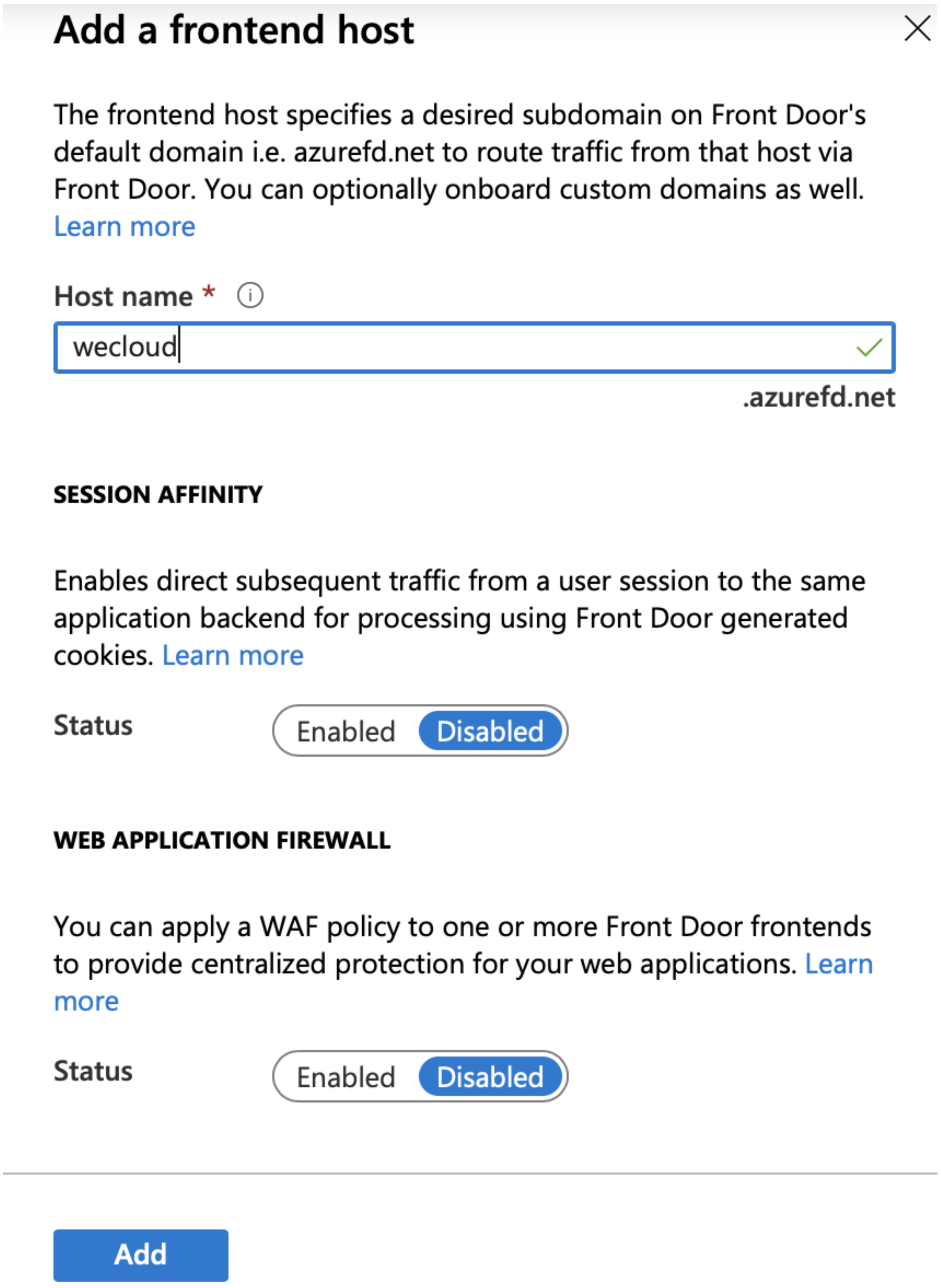

Create a Front Door

Basics | Configuration | Tags | Review + create

Configuring Front Door happens in three steps: Adding a frontend host, configuring your backends in a backend pool and finally a routing rule that connects your frontend to the backend pool. Learn more

Figure: Create Frontends/domains.

4. For the **"Frontend hosts"** configuration, you need to assign a name in the "Host name" field. **The name should be unique globally.** Leave the other configurations as default. Then, click **"Add"** to continue.


Add a frontend host

The frontend host specifies a desired subdomain on Front Door's default domain i.e. azurefd.net to route traffic from that host via Front Door. You can optionally onboard custom domains as well. Learn more

Host name * : wecloud .azurefd.net

SESSION AFFINITY

Enables direct subsequent traffic from a user session to the same application backend for processing using Front Door generated cookies. Learn more

Status: Enabled [✓ Disabled]

WEB APPLICATION FIREWALL

You can apply a WAF policy to one or more Front Door frontends to provide centralized protection for your web applications. Learn more

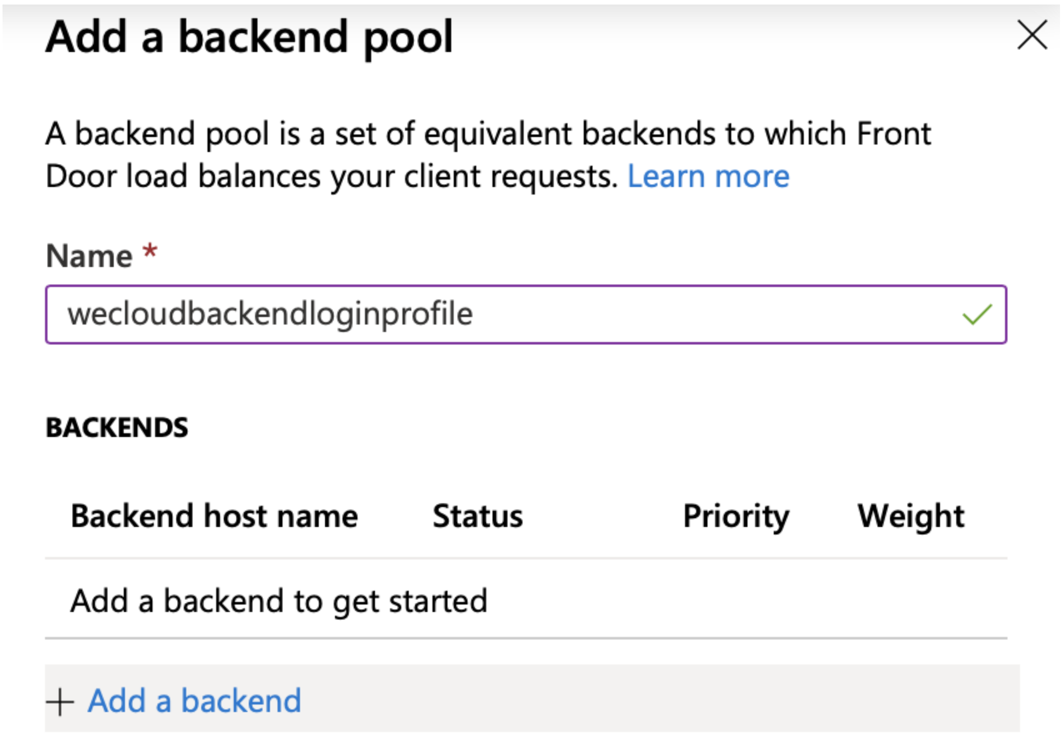
Status: Enabled [✓ Disabled]

Figure: Add a frontend host. Note that you need to use a globally unique name.

5. Next, you need to create the backend pool. A backend pool is a set of equivalent backends which the Front Door uses to balance the client request load to the respective service. Revisiting the architecture of the WeCloud Chat application in Task 5, we will need to create two backend pools:

    - For the login service and the profile service, which exist on both GCP and Azure, we need a backend pool consisting of two IPs: one is the external IP of the Ingress on GCP, other one is the external IP of the Ingress on Azure.
    - For the chat service, which exists only on GCP, the backend pool will consist of only a single IP of GCP

   To create a backend pool with both GCP and Azure IPs, give a name to your backend pool. Then, click **"Add a backend"**.


Add a backend pool

A backend pool is a set of equivalent backends to which Front Door load balances your client requests. Learn more

Name *

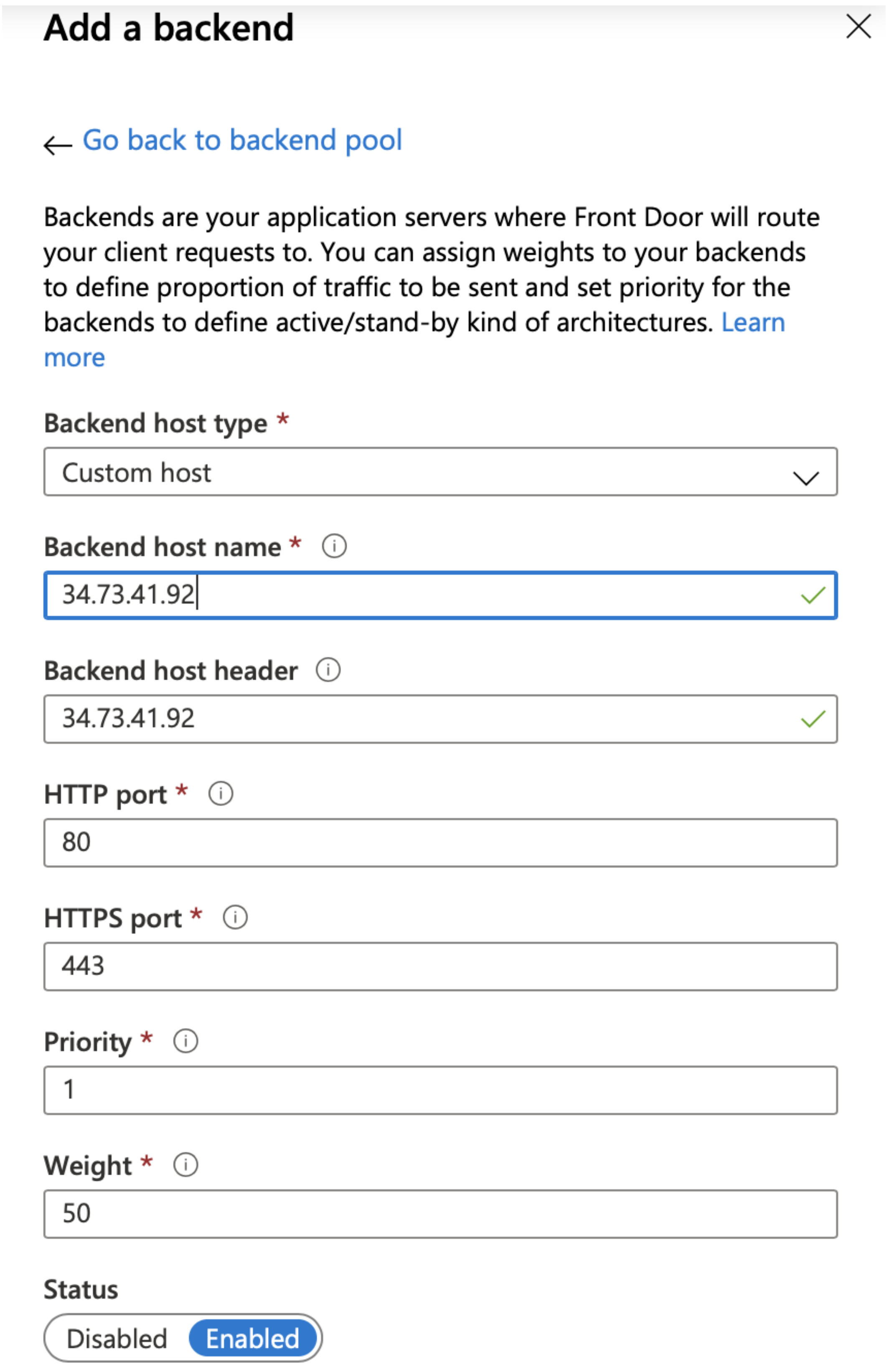
wecloudbackendloginprofile

BACKENDS

Backend host name	Status	Priority	Weight
Add a backend to get started			

Figure: Add backend pool for login service and profile service.

6. Click on **"Add a backend"** to add a new backend. Choose Custom host for Backend host type. In Backend host name and for Backend host header, enter the external IP of the Ingress of GKE. Leave the rest of the configurations to their default.



Add a backend

← Go back to backend pool

Backends are your application servers where Front Door will route your client requests to. You can assign weights to your backends to define proportion of traffic to be sent and set priority for the backends to define active/stand-by kind of architectures. Learn more

Backend host type *
Custom host

Backend host name *
34.73.41.92

Backend host header
34.73.41.92

HTTP port *
80

HTTPS port *
443

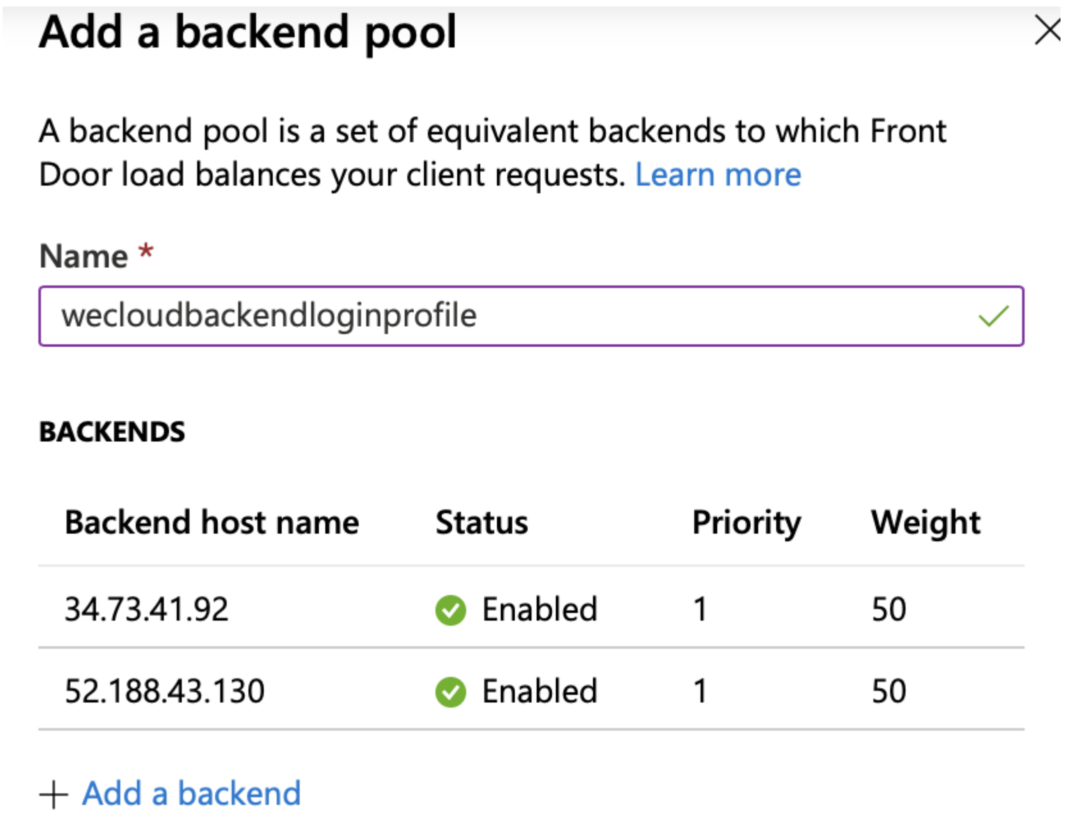
Priority *
1

Weight *
50

Status
Disabled [✓ Enabled]

Figure: Add GCP backend to the backend pool.

7. You need to add another backend, with Backend host type as Custom host and Backend host header as the external IP of the Ingress on AKS. The following is the screenshot after you have added two IP addresses for the backend pool.



Add a backend pool

A backend pool is a set of equivalent backends to which Front Door load balances your client requests. Learn more

Name *

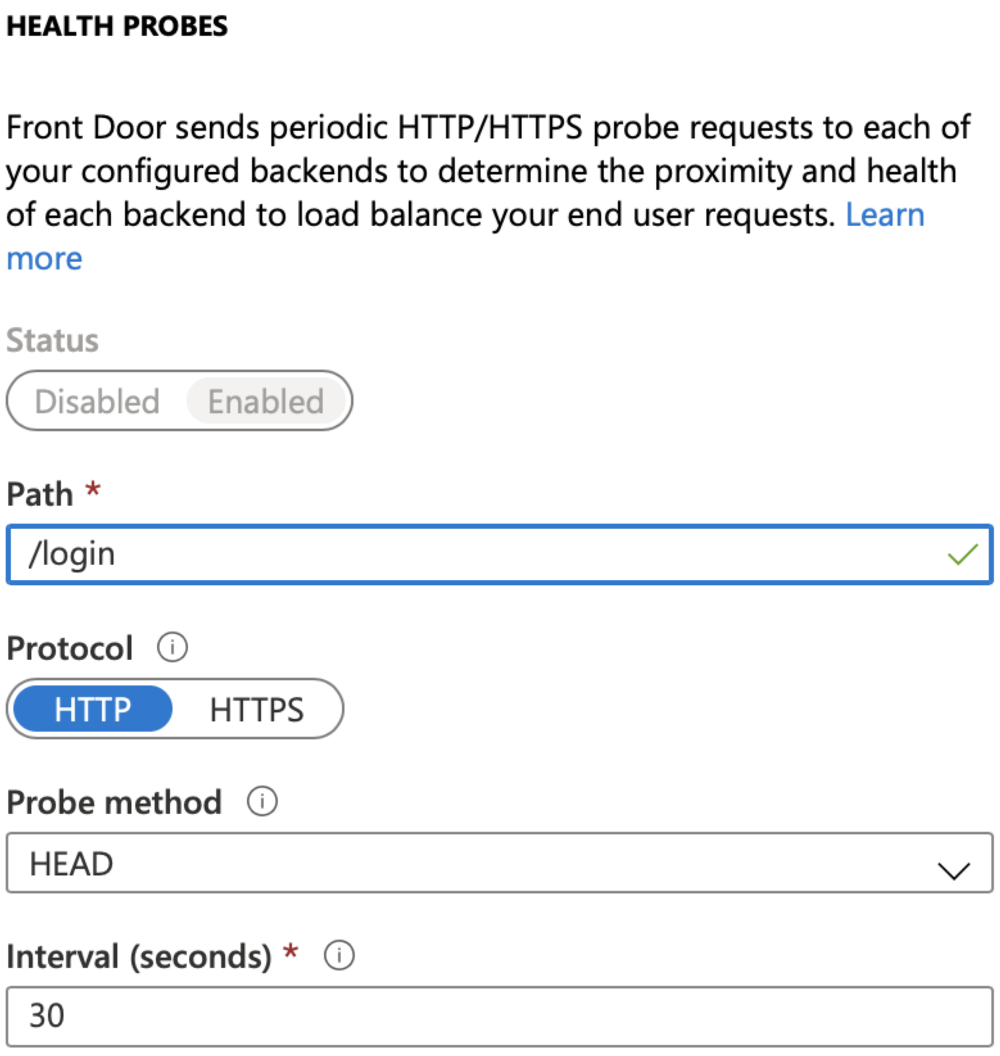
wecloudbackendloginprofile

BACKENDS

Backend host name	Status	Priority	Weight
34.73.41.92	✓ Enabled	1	50
52.188.43.130	✓ Enabled	1	50

Figure: Overview of the backend pool for login service and profile service.

8. For **"Health Probes"**, since there is no application under the root ("/") path of the AKS, we need to set the health check path to be **"/login"**. Choose **"HTTP"** as the Protocol.


HEALTH PROBES

Front Door sends periodic HTTP/HTTPS probe requests to each of your configured backends to determine the proximity and health of each backend to load balance your end user requests. Learn more

Status: Disabled [✓ Enabled]

Path *: /login

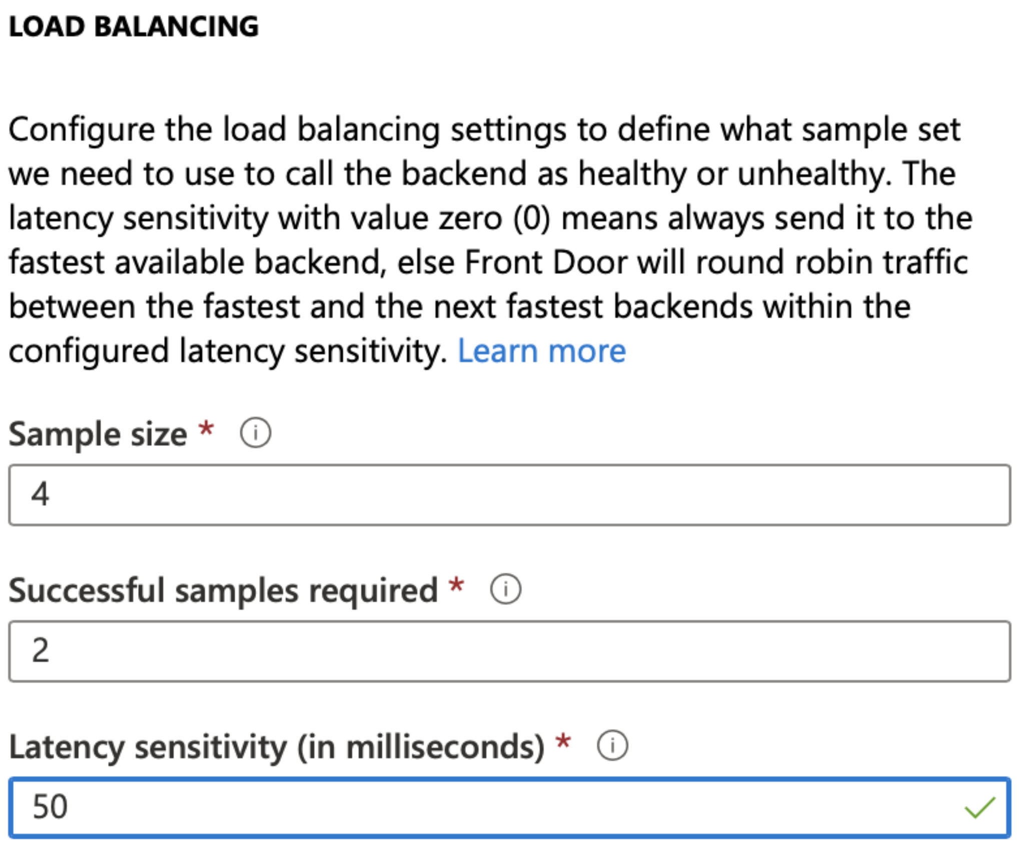
Protocol: [✓ HTTP] HTTPS

Probe method: HEAD

Interval (seconds) *: 30

Figure: Configure health probes.

9. For **"Load Balancing"** rules, note that the default value for latency sensitivity is 0, which means always send the request to the fastest available backend. Front Door will only round robin traffic between backends whose latencies are within the configured latency sensitivity. You need to give a reasonable value (e.g. 50) so that Front Door will round robin traffic between the GKE and AKS.


LOAD BALANCING

Configure the load balancing settings to define what sample set we need to use to call the backend as healthy or unhealthy. The latency sensitivity with value zero (0) means always send it to the fastest available backend, else Front Door will round robin traffic between the fastest and the next fastest backends within the configured latency sensitivity. Learn more

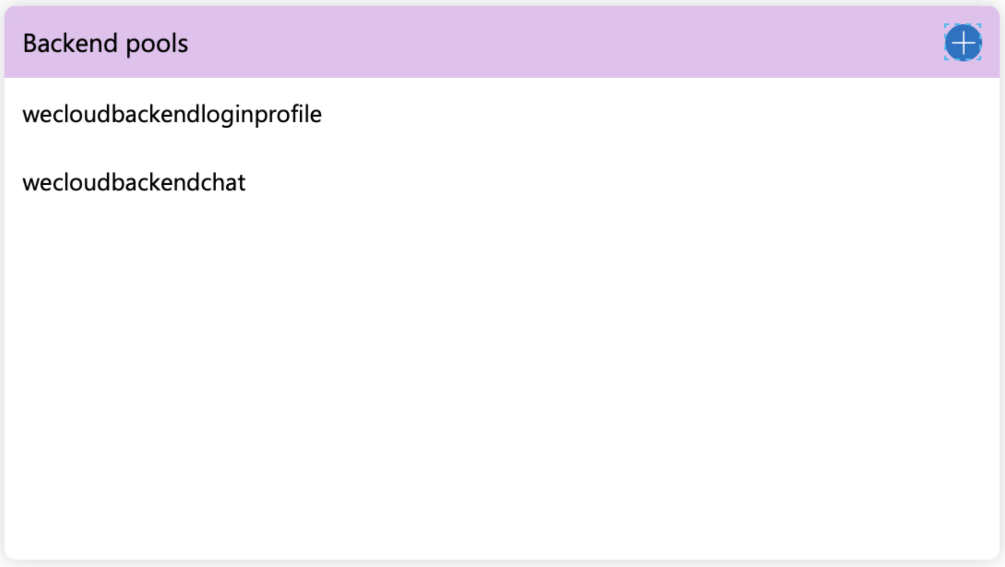
Sample size *: 4

Successful samples required *: 2

Latency sensitivity (in milliseconds) *: 50

Figure: Configure load balancing.

10. Click **"Add"** to register the backend pool you just configured. After adding the backend pool with both GCP and Azure, you need to add another backend pool with GCP only for the chat service routing. Once you have created both the backend pools, you will see the following under the Backend pools section.


Backend pools

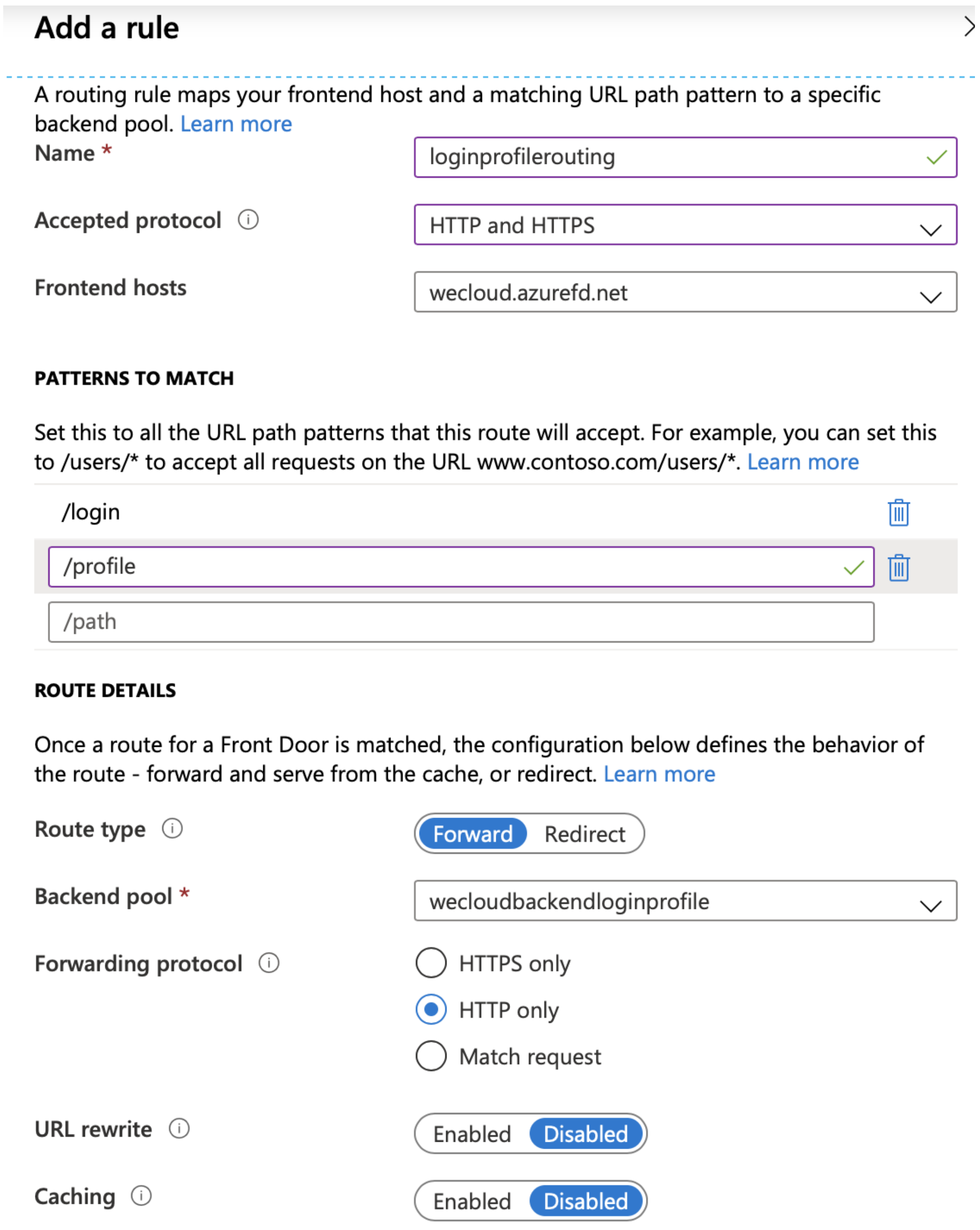
wecloudbackendloginprofile

wecloudbackendchat

Figure: Overview of two backend pools.

11. In the **"Routing Rule"** part, we need two routing rules. One for the login service and profile service, which route to the first backend pool; and another one for the chat service, which routes to the second backend pool.

For the first routing rule, add two paths **"/login"** and **"/profile"** to **"Patterns to Match"**. Choose the backend pool for the login service and the profile service. For **"Forwarding protocol"**, choose **"HTTP only"**. This routing rule will route http://<wecloudsubdomain>.azurefd.net/login and http://<wecloudsubdomain>.azurefd.net/profile to the backend pool with Azure and GCP.



Add a rule

A routing rule maps your frontend host and a matching URL path pattern to a specific backend pool. Learn more

Name *	loginprofilerouting ✓
Accepted protocol	HTTP and HTTPS
Frontend hosts	wecloud.azurefd.net

PATTERNS TO MATCH

Set this to all the URL path patterns that this route will accept. For example, you can set this to /users/* to accept all requests on the URL www.contoso.com/users/*. Learn more

/login

/profile ✓

/path

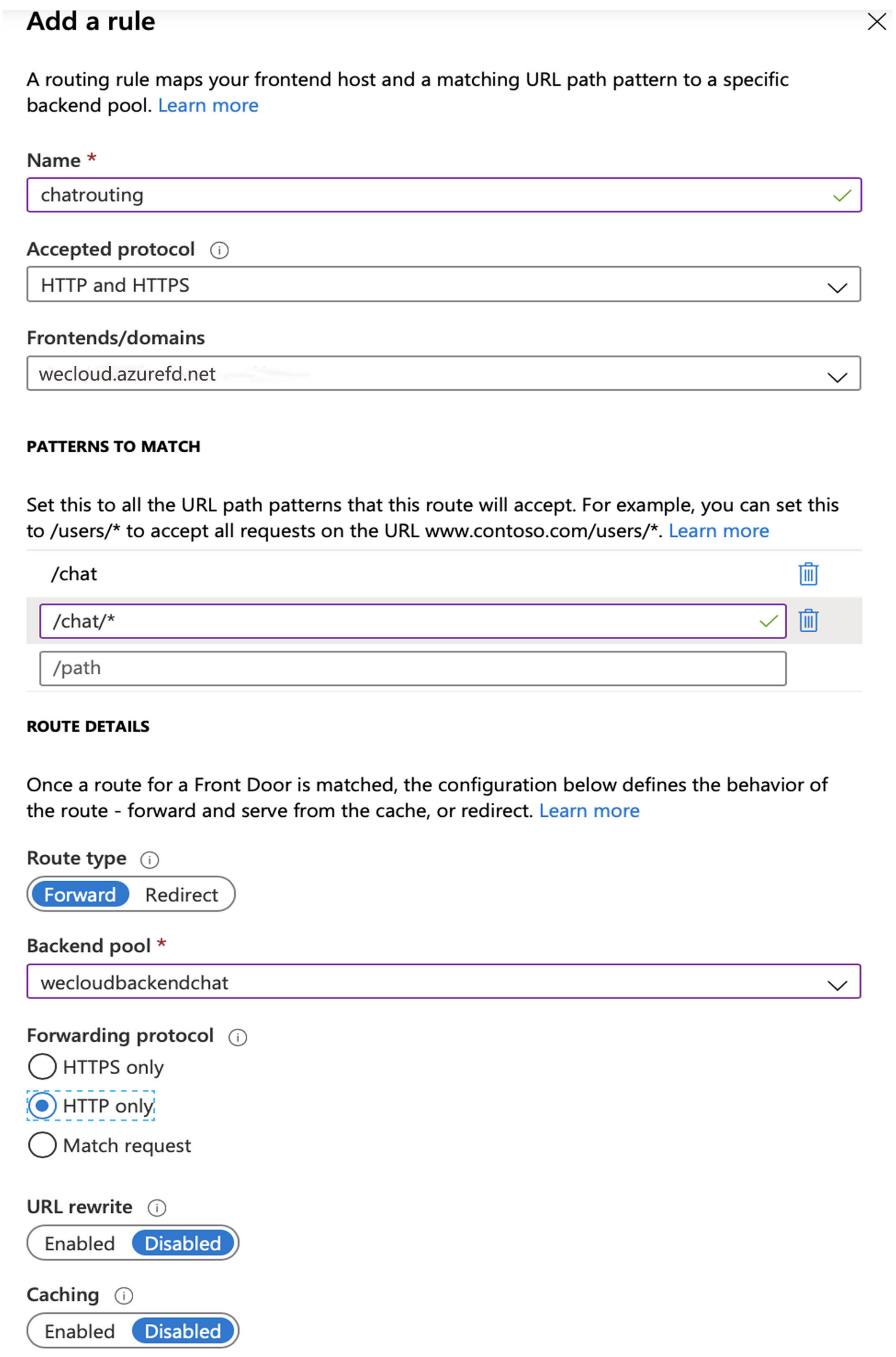
ROUTE DETAILS

Once a route for a Front Door is matched, the configuration below defines the behavior of the route - forward and serve from the cache, or redirect. Learn more

Route type	[✓ Forward] Redirect
Backend pool *	wecloudbackendloginprofile
Forwarding protocol	HTTPS only [✓ HTTP only] Match request
URL rewrite	Enabled [✓ Disabled]
Caching	Enabled [✓ Disabled]

Figure: Add the routing rule for login service and profile service.

12. Add another routing rule for the chat service. This time **"Patterns to Match"** has **"/chat"** and **"/chat/*"**, and it routes to the second backend pool (i.e. only GCP).


Add a rule

A routing rule maps your frontend host and a matching URL path pattern to a specific backend pool. Learn more

Name *
chatrouting

Accepted protocol
HTTP and HTTPS

Frontends/domains
wecloud.azurefd.net

PATTERNS TO MATCH

Set this to all the URL path patterns that this route will accept. For example, you can set this to /users/* to accept all requests on the URL www.contoso.com/users/*. Learn more

/chat
/chat/*
/path

ROUTE DETAILS

Once a route for a Front Door is matched, the configuration below defines the behavior of the route - forward and serve from the cache, or redirect. Learn more

Route type
[✓ Forward] Redirect

Backend pool *
wecloudbackendchat

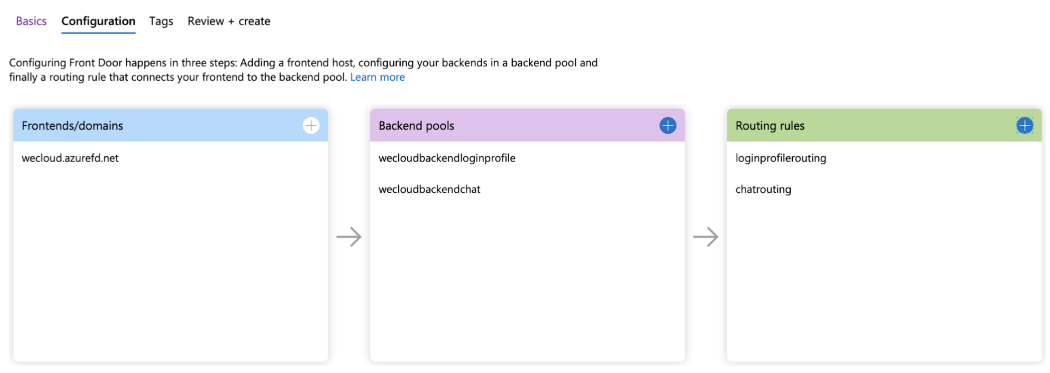
Forwarding protocol
HTTPS only
[✓ HTTP only]
Match request

URL rewrite
Enabled [✓ Disabled]

Caching
Enabled [✓ Disabled]

Figure: Add the routing rule for chat service.

13. Now you have designed the whole architecture for the Front Door, double check you have the same architecture as the following figure.



Configuring Front Door happens in three steps: Adding a frontend host, configuring your backends in a backend pool and finally a routing rule that connects your frontend to the backend pool. Learn more

Frontends/domains	Backend pools	Routing rules
wecloud.azurefd.net	wecloudbackendloginprofile	loginprofilerouting
	wecloudbackendchat	chatrouting

Figure: Overview of Front Door configurations.

14. Click “Review + create”. Wait patiently until the deployment is completed.

Validate Your Work

After the Front Door is successfully deployed, you can visit the application with your domain name. Verify your deployment with http://<wecloudsubdomain>.azurefd.net:80/login, http://<wecloudsubdomain>.azurefd.net:80/chat?username=majd, and http://<wecloudsubdomain>.azurefd.net:80/profile?username=majd. If the above links don't work, please revisit the configuration of Azure Front Door. Alteratively, also try http://<wecloudsubdomain>.azurefd.net/login (without the port 80) to see if the configuration is correct.

What to Submit

- the citation file under `~/Project_Containers/task6/` to include all the links that you referred to for completing this task.

How to Submit

1. You will submit from your student VM on GCP.
2. Export your submission username and submission password:

```
$ export SUBMISSION_USERNAME="your_submission_username"
$ export SUBMISSION_PASSWORD="your_submission_pwd"
```

3. In the `task6` directory, execute the submitter `./submitter`.

Orchestration Visualization and Services Monitoring

Orchestration Visualization and Services Monitoring (Ungraded)

When your Microservices grow to a certain scale, they will become difficult to understand, manage and troubleshoot.

Weave-scope (https://github.com/weaveworks/scope) is a tool for **Troubleshooting & Monitoring for Docker & Kubernetes**.

If you follow the official instructions, you may find that it has too many steps. Helm will help make our life easier again.

1. Switch to the GCP Kubernetes context.

2. Install Weave Scope in the cluster:

```
$ helm install wecloudchat-weave-scope stable/weave-scope
```

3. Run:

```
$ kubectl -n default port-forward $(kubectl -n default get endpoints wecloudchat-weave-sco
pe-weave-scope -o jsonpath='{.subsets[0].addresses[0].targetRef.name}') 8080:4040
```

4. Set up an SSH tunnel so that you can visit port 8080 of the student VM without opening the port to the world, here is an example for the Mac user:

```
$ ssh -i $HOME/.ssh/google_compute_engine -L 8000:localhost:8080 clouduser@<your-student-v
m-ip>
```

5. Visit `localhost:8000` and you will find a visualization of the deployed applications.

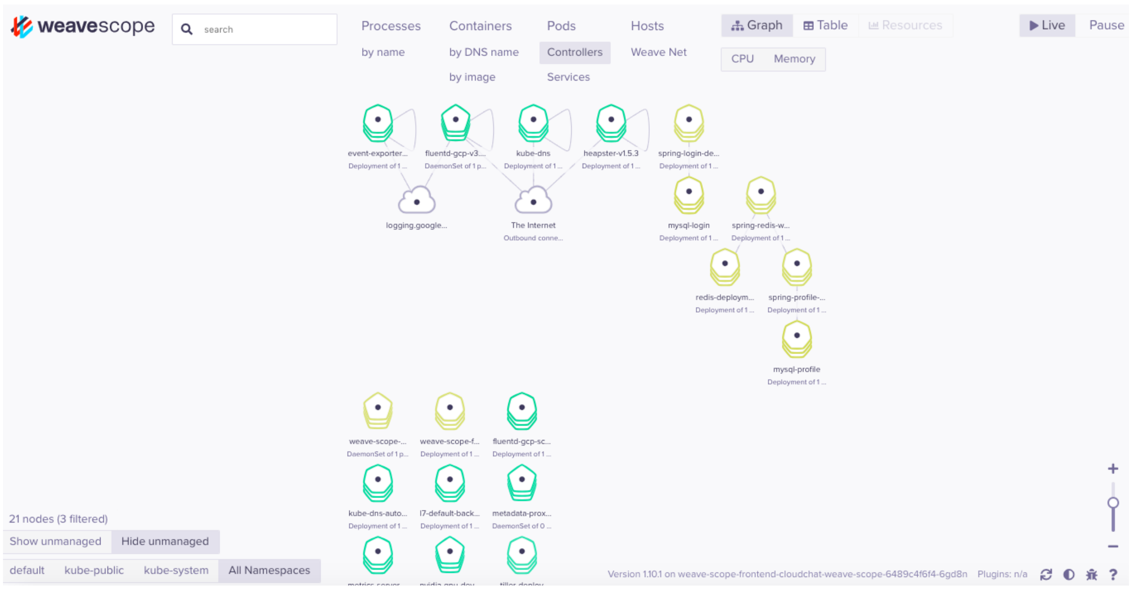


Figure: Weave Scope UI

Delete Cloud Resources

After finishing all the tasks, you need to delete all the resources on Azure and GCP.

- For Azure, delete all the resource groups related to this project in the Azure console or Azure CLI command:

```
$ az group delete --name $RESOURCE_GROUP
```

- For GCP, delete the GCP project in the console or gcloud CLI command:

```
$ gcloud projects delete $PROJECT_NAME
```

Troubleshooting Section

1.**Problem:** Seeing white label page when a service is invoked using curl or http

Symptom: Service.yaml and deployment.yaml files deployed using kubernetes with same label for multiple services under helm folder

Cause: Multiple services reference the same label causing whitelabel error page

Solution: Use different labels for the services in yaml files under helm folder

2.**Symptom:** [WARNING] unable to get access token for Google Container Registry, configuration for building image will not contain RegistryAuth for GCR

Cause: Not authenticated to GCR on student-vm

Solution: On the student VM authenticate to GCR by running the following commands:

gcloud auth application-default login gcloud config set project gcp-docker-kubernetes-xxxxxx gcloud config set compute/region us-east1 gcloud config set compute/zone us-east1-b gcloud auth login gcloud auth configure-docker

3.**Symptom:** The output of kubectl get deployment gives CrashLoopBackOff error

Cause: This means that pod starting, crashing, starting again, and then crashing again.

Solution: Use kubectl logs POD_NAME to log the output of specific pods to find error messages.

4.**Symptom:** The output of kubectl get deployment gives ImagePullBackOff error

Cause: The pod is not able to pull the docker image specified in deployment.yaml from ACR/GCR

Solution: Make sure you are authenticated to GCR and ACR Azure. It is very likely that you did not follow AKS to ACR integration or the docker image:tag is incorrect or not pushed to GCR/ACR.

5.**Symptom:** Error: Could not find or load main class when student trying to run java class

Cause: This is a Spring Boot specific behavior. The main class has been replaced by Spring Boot Launcher instead. On unzipping the jar and seeing the manifest file META-INF/MANIFEST.MF we can see the main class

Solution: Use java -jar profile-embedded-0.1.0.jar to create jar file. Reference (https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-executable-jar-format.html)

6.**Symptom:** h2validation: FAILED Hint: Please check if you have disabled the embedded database in the pom.xml and application.properties files

Cause: Either embedded database variables are not updated to mySQL in application.properties file or docker image was not rebuilt with new image tag

Solution: Make sure you give the image a new tag after making the required changes (e.g. bump the version from v001 to v002). Then do docker push, and remember to update the deployment.yaml file.

Project Reflection Task (Mandatory, graded)

Project Reflection Task (Mandatory, graded)

Upon completing this project you will make one (1) post in the [Forum] P2. Containers: Docker and Kubernetes (https://projects.sailplatform.org/cloud-forum/topic/publish/1030/) to reflect on your experience before the project deadline.

Consider the following topics when creating your post, however, you should never share any code snippets in your reflection:

- Describe your approach to solving each task in this project. Explain alternative approaches that you decided not to take and why.
- Describe any interesting problems that you had overcome while completing this project.
- If you were going to do the project over again, how would you do it differently, and why?

After completing this task, confirm that your ***Reflection Score*** has been automatically updated on the scoreboard before the project deadline.

Project Survey

Project Survey

Please leave us feedback for this project here. This will help us strengthen this project for future offerings.

Project Discussion

Project Discussion Task (Mandatory, graded)

After this project has completed (after the project deadline), all reflection posts by all students will become visible for review.

Your task is to reply and provide feedback to **3** posts in the [Forum] P2. Containers: Docker and Kubernetes (https://projects.sailplatform.org/cloud-forum/category/1030/), within **7** days after the project deadline.

After completing this task, confirm that your ***Discussion Score*** has been automatically updated on the scoreboard within 7 days after the project deadline.