EEL 6871: Cloud Computing Sys. Mgmt. Fall 2023 Instructor: J. Fortes

Project Assigned: 11/2/2023 **Task 1 Due: 11/17/2023 @ 11 pm**

Task 2 Due: 11/27/2023 @ 11 pm

Task 3 Due: 12/4/2023 @ 11 pm

Feedback-based Control and Kubernetes Cluster

Preamble:

The aim of this project is to monitor a Kubernetes cluster (hereon referred to as the cluster) in production and manage the same to maximize performance of the cluster. In this project, you will use the CloudLab resources to deploy a Kubernetes cluster running stress-ng tests and implement a feedback-based controller architecture to optimize the resource utilization of the cluster. Towards this you will model the cluster using the techniques described in chapter 2 of the textbook. You will use the model to design and implement controllers that will monitor CPU utilization and scale the pods and/or nodes to meet the SLA requirements. Figure 1 shows the overview of the cluster that you will be deploying in CloudLab.

This is a group project. Each group will contain 5 students randomly assigned by the instructor. There are 5 modules in this project (discussed below) each of which will be headed by a student of the group. Everyone must participate in designing and/or implementing all modules but the student heading the module will contribute the most towards finishing that module. This must be clearly shown in the report to receive full credit.

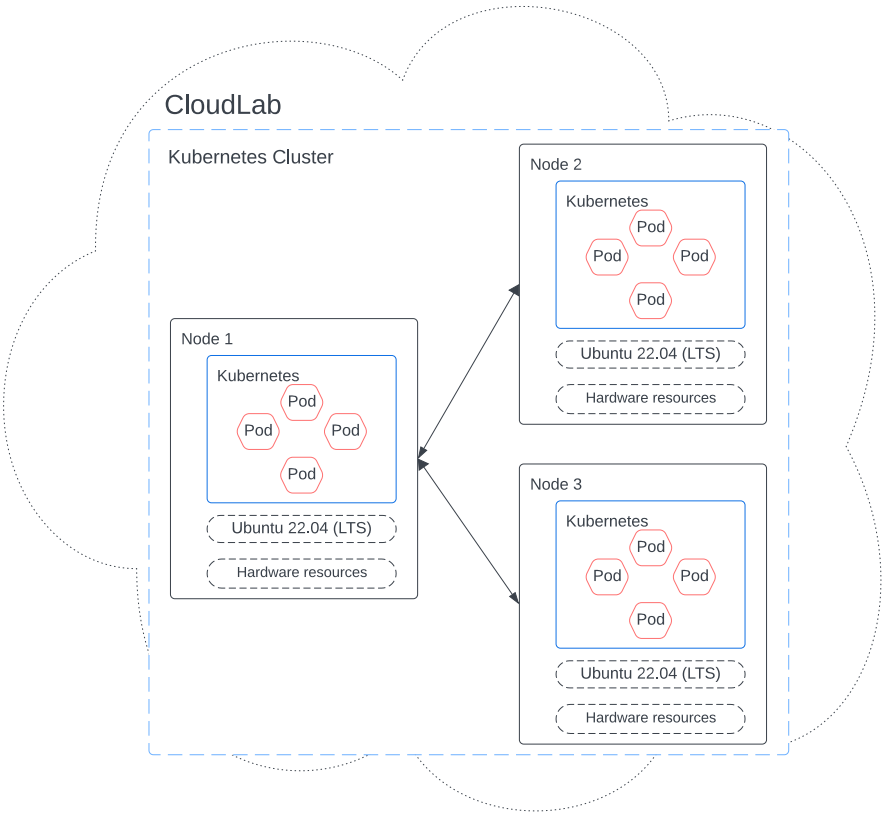

Figure 1: Overview of the Kubernetes cluster.

Kubernetes Cluster:

To recap, Kubernetes is an orchestration platform which allows managing containerized applications. Docker Inc., the company that developed Docker containers, provides their own orchestration platform called Docker swarm. However, it is limited in its capabilities and Kubernetes is preferred in the industry to manage complex containerized applications and their inter-connections. There are multiple distributions of Kubernetes available either as standalone open-source or as a service. To learn about the various standalone offerings, refer to: https://kubernetes.io/docs/setup/. To learn more about Kubernetes and the various concepts/terminologies used in the documentation refer to: https://kubernetes.io/docs/concepts/.

So far in this course, you have used the KinD flavor of Kubernetes to create clusters. KinD obfuscated the management tasks that are required while setting up a cluster thereby making it simpler to learn Kubernetes. As part of this project, you will instead use Kubeadm and Kubectl to deploy a Kubernetes cluster in CloudLab. You must follow the best practices while deploying your cluster, i.e., your cluster must be of production quality.

Note: For this project, you can use KinD to finish the project. You will earn extra credits (20 points) if you use Kubeadm to deploy a production quality Kubernetes cluster for the project.

Modeling:

The controller architecture consists of several controllers, namely, one controller for each node (called local controller) and a controller for the entire cluster (called the global controller). Thus, there are two target systems in this project: (1) individual nodes and (2) the overall cluster. Figure 2 shows the feedback loop of the global controller for this project where the Kubernetes cluster is the target system. Figure 3 shows the feedback loop of the local controller for this project where the individual nodes of the cluster are the (identical) target systems. There will be one global controller that monitors the entire cluster while there will be an instance of local controller running inside every node in the cluster monitoring the node. More details about the controller are provided in the Controller section of this document.

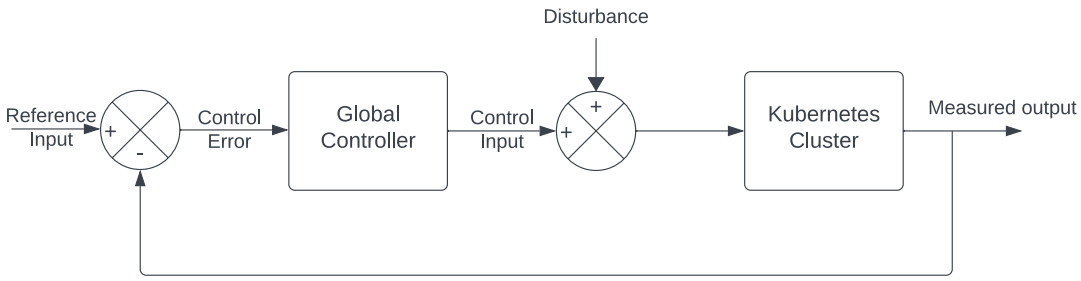

Figure 2: Schematic of the feedback-control loop for the global controller including the disturbance.

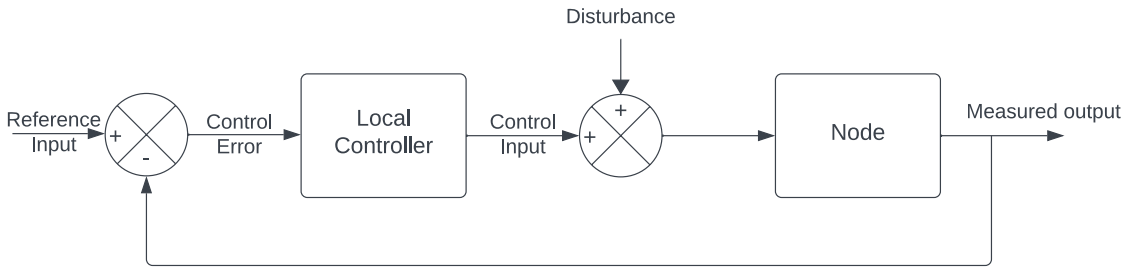

Figure 3: Schematic of the feedback-control loop for the local controller including the disturbance.

You will use the knowledge that you have acquired through the various assignments to model the target systems (the cluster that you deploy in CloudLab and the node(s) in the cluster). Specifically, your parameter of interest will be CPU utilization of the whole cluster and the individual nodes. Recall, that there are two components of a Kubernetes cluster that provide compute power, namely, pods and nodes. Pods are the compute entities in a Kubernetes cluster within the namespace of which the containers are executed. Pods run within a node which determines the resource availability to the pods. Thus, while scaling, either the number of pods can be changed or the number of nodes in the cluster can be changed. In a cloud environment, the Cloud Controller Manager (CCM) component of Kubernetes will use the cloud's API to procure nodes, when necessary, which can then be used to run pods. For this project, since CloudLab does not expose an API to get nodes, you will have a static number of nodes (3 nodes) in hot-standby. When the cluster is unable to meet the expected CPU utilization levels, you will add one of the hot-standby nodes to the cluster. Thus, the modeling part of this assignment requires you to identify two models, i.e. an individual node and a model for the cluster.

Monitoring:

Monitoring is a key aspect of the feedback control loop. You will monitor the cluster that you deployed specifically for the CPU utilization metric. You will monitor the CPU utilization at the pod level as well as the node level. Monitoring also includes watching the health of the cluster. Towards this, you will monitor the health of the pods and the nodes aka *health check*. Observing the activity through "top" might not always provide sufficient information for a health check. For instance, a process could be in a deadlock but monitoring via "top" will show that the process is alive and active or waiting for resources. Thus, you must use other probes, such as *liveness probes* to monitor the health of your process. Chapter 4 of Kubernetes Patterns – Reusable Elements for Designing Cloud-Native Applications by Bilgin Ibryam and Roland Hub provides detailed information on how to monitor the health of a Kubernetes cluster. This book can be accessed at: https://www.redhat.com/rhdc/managed-files/cm-oreilly-kubernetes-patterns-ebook-f19824-201910-en.pdf

To monitor the nodes and pods, you will deploy the metrics server (https://github.com/kubernetes-sigs/metrics-server) in your cluster. You will implement a script that will periodically use the APIs exposed by the metrics server to monitor the CPU utilization of the pods and nodes. You can also integrate into this script your code to monitor the cluster health or you could create a separate script to monitor the cluster health periodically.

Controller:

You will design and implement two controllers, a local controller, and a global controller, based on the models you obtained for the nodes and the cluster. The concept of a local controller and a global controller is common in computing systems where the scope of the local controller is contained within a subsystem and focuses on optimizing the subsystem whereas the scope of the global controller is the entire system and focuses on optimizing the system. Drawing inference from the examples in the textbook, consider Apache servers deployed in a Software Defined Environment (SDE). The Apache controller with Maxclients and KeepAlive as control inputs could be seen as a local controller optimizing the individual server's performance while the Software Defined Infrastructure (SDI) controller could be seen as a global controller that looks at the overall resource utilization to optimize the infrastructure and meet SLA/SLO requirements.

The local controller will monitor the node and make decisions on how to scale the pods to track the reference input. Referring to Figure 3, the reference input will correspond to the desired overall node CPU utilization while the control input will be the maximum number of Pods (MAX_PODS) that the node can run to track the reference input. The textbook introduces you to various feedback-based controllers. You must make sure that your choice of controller is a good fit for the target system. For this controller, the operating point will be 80% (CPU utilization) and the operating range will be 75% to 85%. Note: For testing purposes, the reference input will be changed at runtime and your local controller must track the changes to the reference input (step changes will be used)

The global controller will monitor the cluster and make decisions on how to scale the nodes to track the threshold. Referring to Figure 2, the reference input will be a step signal with an amplitude of 80% which is the desired overall CPU utilization of the cluster, while the control input will be the total number of nodes in the cluster. This controller will be a simple rule-based controller which will spawn a new node if the CPU utilization of existing nodes exceeds 80%. The global controller will also have an additional responsibility of allocating jobs to the available nodes. To achieve this, the global controller will have access to the current configuration of MAX_PODS in each node and the current total number of PODS running in each node. The global controller will use this information to determine the node to which the job can be allocated. More details about the jobs are provided in the Jobs section of this document.

The cost of the resources is out of scope of this project. However, when designing a controller, you must keep the cost of unused resources in mind. Your controller must make decisions in such a way that the goal (CPU utilization) is always met while minimizing the cost of operating the cluster. In other words, a simple controller that only adds pods/nodes to the cluster is not the desired solution. Your controller must be able to decide when to scale up and when to scale down the system to minimize the overall cost.

Middleware:

Separation of concerns is important for ease of management as well as ease of maintenance. You will implement middleware that will act as an intermediary between the controller and the cluster (i.e., it will serve as the actuator that executes the necessary system adaptations – corresponding to the E-part of the MAPE loop). The middleware will implement methods to scale up or scale down the pods/nodes. It will also implement any other administrative tasks that you will be performing on the cluster as part of this project. The middleware will expose the methods that can be used by the controller to effect its control decisions. The middleware must also expose two methods, one to kill a pod, and another to kill a node.

Jobs:

Job scheduling in computing systems is a critical aspect of resource management and optimization. Job scheduling is a very complex task. There are a wide range of scheduling algorithms and strategies that are tailored to specific goals and environment. Job scheduling is out of scope of this project, instead you will be implementing a simple FIFO-like job scheduling in this project.

In this project, you will use the stress-ng tool to generate load on the cluster (Pods). You will create a text file (Job queue) where each line of the file will denote a stress test (job). You will create a mix of stress tests (refer to HW3 for the various stress tests available) which will form the various workloads for your cluster. It is important that your stress tests not only test the CPU but rather consists of a good mix, a good example is "stress-ng --io 4 --vm 5 --vm-bytes 2G --timeout 5m". You will create various such stress tests, listed one per line of the text file, with varying stressors and varying timeouts to simulate different job completion times.

The global controller will read from the job queue file periodically (every 15 seconds) and allocate the head job to an available node (determined using the logic mentioned in the Controller section). Figure 4 shows an overview of the simplified job scheduling process.

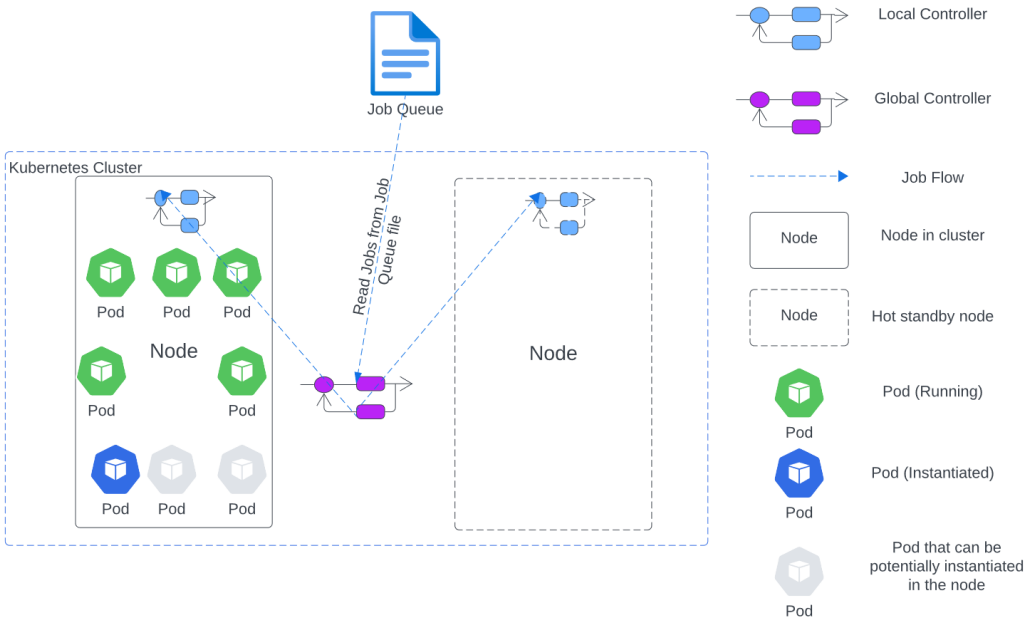

Figure 4: Schematic showing job flow, the controllers, and the various states of pods

Important notes (Read carefully before you start the assignment):

1. For security purposes, treat the CloudLab machines as your personal computer and set strong passwords where needed. Do not share your private ssh key with anyone. CloudLab admins constantly monitor the network traffic, and you risk losing access to CloudLab resources for the rest of the semester if the CloudLab admins detect any anomalous traffic from your nodes.

2. CloudLab has a user forum for discussions and questions. Treat the forum like Stack Overflow. Search for answers before posting questions and follow the community guidelines. CloudLab admins are very helpful and prompt in their response provided you follow the guidelines while posting questions.
3. Follow the guidelines that were discussed in the CloudLab lecture while deploying Docker containers and Kubernetes cluster. Do not use commonly used ports (https://www.cloudflare.com/learning/network-layer/what-is-a-computer-port/) to deploy the Docker application.

Important Note: As a group, you must procure 3 nodes from CloudLab (3 nodes per group). To help you with retaining the resources until the end of the semester, nodes have been reserved in CloudLab starting 11/4/2023 at 1 PM. The lead student from each group must procure 3 nodes for the group. These are the nodes that you will use to complete the project tasks.

Tasks:

There are 3 tasks as part of this project. The tasks have different deadlines (refer to submission guidelines).

1. **Setting up the Kubernetes Cluster and modeling a node:**
   This task has a group component and an individual component.
   a. Install the necessary tools (refer to HW2) to deploy a Kubernetes cluster on CloudLab. You will only use Kubeadm and Kubectl to create and manage the cluster. Your cluster must be of production quality, i.e., you must follow the best practices for deploying a cluster including but not limited to security – creating a separate non-root user for the middleware, exposing only the necessary ports, maintaining, and managing the various configuration files used to deploy the cluster, etc. The cluster will be created by the group and there will be one cluster deployed for the entire group using the 3 nodes procured from CloudLab.
   b. This part of task 1 must be done individually. You will use the cluster created in the previous step towards completing this part of task 1. The submission of this task will be considered as HW 5 for this semester. You will use stress-ng to generate load in the Pods. You may or may not implement a job queue at this point. Students in the same group should agree on when each student has exclusive access to the cluster node(s) so that their modeling efforts do not interfere with those of other students.
   After deploying the cluster as group, each student in the group must model the node for its CPU utilization versus the number of pods using the knowledge from lectures, homeworks and chapter 2 of the textbook. Remember, without a good model of the cluster, you will not be able to successfully complete this project.

   Submission Task 1 (**11/17/2023@11 pm**) **Individual**:
   - You will submit a PDF file named <your_name>_<your_group_#>_hw5.pdf through Canvas.
   - Your PDF must:
     - Describe the steps taken to deploy the Kubernetes cluster (500 words or less)
     - Describe in detail the process you followed to model the node performance (must include details of your experiments, the input signal, justifications for any assumptions etc.).
     - Include the plots (especially CPU utilization vs. # of Pods) to support your model along with verification and validation metrics.

2. **Designing and Implementing the Local Controller:**
   This task is group based and requires one PDF submission per group.
   a. You will deploy the metrics server (https://github.com/kubernetes-sigs/metrics-server) in the cluster. You will implement a script that will periodically monitor the CPU utilization of the cluster, the nodes, and the pods using the API endpoint exposed by the metrics server. You must choose a good sampling rate to obtain as much useful information as possible from the metrics server about the cluster.
   b. As a group you will collectively discuss the various models obtained by the group members and choose the model that you think is the best.
   c. You will determine the controller that is best suited to manage the MAX_PODS in the node. You will use the chosen model to design the local controller and implement the local controller.
   d. You will use stress-ng to generate the load. The stressors used in this task must be different from the stressors used in the previous task. You may or may not implement a job queue at this point.

   Simulation for task 2

   a. Start the local controller and any other components that are required for monitoring in a node.
   b. Jobs from the job queue will be serviced by the node (periodically read once every 15 seconds).
   c. The local controller will determine the MAX_PODS depending on the CPU utilization of the node.
   d. If MAX_PODS number of Pods are running in the node, no new job will be started.
   e. The simulation ends when all the jobs in the job queue are completed.
   f. Create a scenario where you demonstrate the controller increasing the value of MAX_PODS and create a scenario where you demonstrate the controller decreasing the value of MAX_PODS.
   g. You will collect CPU utilization of the node and the MAX_PODS values throughout the run.

   Submission Task 2 (**11/27/2023@11 pm**):
   - This will be a group submission (i.e., one submission/group).
   - The group leader will submit a PDF file named <your_group_#>_task2.pdf.
   - Your PDF must:
     - List the names of the group members.
     - List the 5 models obtained by the group members and must list the chosen model along with a justification of why.
     - Provide the reasoning for choosing a particular controller and describe in detail the various assumptions and criteria chosen towards designing the controller.
     - Show CPU utilization vs time, MAX_PODS vs time plots for the simulation for task 2.

3. **Designing and Implementing the Global Controller:**
   a. You will determine the best controller that is best suited to manage the total number of nodes in the cluster. Remember it is a fixed threshold at 80% (CPU utilization) i.e., the overall CPU utilization of the cluster should not exceed 80%.
   b. You will design and implement the global controller to manage the nodes.

c. The global controller must also be able to read from the job queue file to read the jobs periodically (every 15 seconds). Towards this, the global controller must also have the ability to find the MAX_PODS for each node and the total number of Pods running in each node. This will allow the global controller to assign the job to the node with current number of pods < MAX_PODS.
d. Your global controller must also have the capability to remove a node from the cluster i.e., switch a node from active state to hot-standby state.

Simulation for task 3:

Note: You can re-use the job queue you designed for task 2 in task 3.

a. Start the local controller, global controller, and any other components that are required for monitoring in a node.
b. Jobs from the job queue will be serviced by the cluster (periodically read once every 15 seconds).
c. The local controller will determine the MAX_PODS depending on the CPU utilization of the node.
d. The global controller will determine the total number of nodes in the cluster depending on the overall CPU utilization of the cluster.
e. The simulation ends when all the jobs in the job queue are completed.
f. You must also simulate the following scenario. For this scenario only node-0 is part of the cluster. Create a scenario through jobs where node-0 is running at capacity so that the next job can only be scheduled on a new node (node-1). Let two more jobs be scheduled in node-1. At this point, manually remove node-0 from the cluster (you can achieve this in various ways – the main idea is to simulate a dead node to the global controller). The global controller must detect the failure and only assign new jobs to node-2 and not to node-0.
g. You must also simulate a scenario where there are two nodes in the cluster node-0 and node-1 both running at capacity. You must let all the jobs in node-1 complete. After 1 minute, the global controller must scale down the number of nodes in the cluster where only node-0 is active and either waiting to run tasks or running tasks.

Extra credit (You can attempt both the tasks for a total of 80 points):

1. **Logging (30 points)**:
   Similar to monitoring, logging is a very important aspect of application deployment. Logs provide information on the various states that the application/infrastructure had been through and provide critical information for the devops engineers to debug issues. Log messages are usually classified into different levels based on the severity of the issue. The most common classifications are INFO, DEBUG, WARN, and ERROR. INFO level messages contain information about the application, possibly the different methods that are being invoked along with a timestamp. DEBUG messages potentially contain object level information possibly object dumps that the developers can use to understand the different values. WARN and ERROR messages predominantly contain any runtime errors that the application had encountered. These categories can be viewed as levels with ERROR messages being at the highest level and the INFO messages being at the lowest level (as shown in Figure 5).

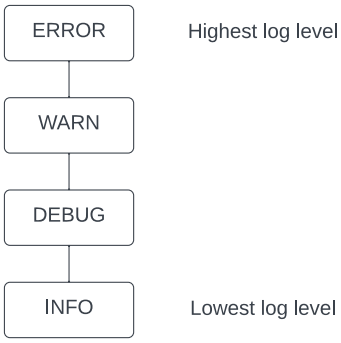

Figure 5: Log levels

When an application is deployed, the log message level is usually configured at deployment time by the devops engineer. The log files encompass log messages corresponding to the configured logging level, as well as log messages from levels higher than the one set for application logging. For example, if an application is configured to capture logs at the DEBUG level, the log file includes DEBUG, WARN, and ERROR messages.

For this extra credit part, you will introduce logs (INFO, DEBUG, and ERROR categories) in the controller, and in the middleware. You can use the standard python library *logging* to achieve this. You will also collect logs from the cluster using Kubectl. You will use (https://fluentbit.io/) to collect and aggregate the logs from the various components. You will use the various tools provided by Fluentbit to filter important logs such as scaling-up decisions and scaling-down decisions made by the controller. You will also show the corresponding logs from the middleware and the cluster to demonstrate that the controller decision was effected.

2. **REST-API service and load-balancing (50 points):**
In this project, you implemented a shell-script that periodically invokes the application to generate load on the pods. In practice, services expose REST-APIs for users to interact programmatically with the application. You must implement a web API that allows users to start the load and potentially read the result back (application dependent). In this project, your global controller was reading the job queue and assigning tasks to the nodes. In this extra-credit part, you must expose 2 APIs, one that will allow you to submit a job to the local controller and another one that will allow you to stop the job. To implement REST-API service, you must create a web-server using python (https://webpy.org and https://webpy.org/0.3/tutorial).

Load balancing is the process of distributing network traffic across pods. You will deploy an Ingress service (https://kubernetes.io/docs/concepts/services-networking/ingress/) to your cluster and utilize the load-balancing feature provided by Kubernetes Ingress. You will implement a client-script that will interact with the API server running in the cluster to start and stop the application. Figure 6 shows a schematic of the setup with REST-API.

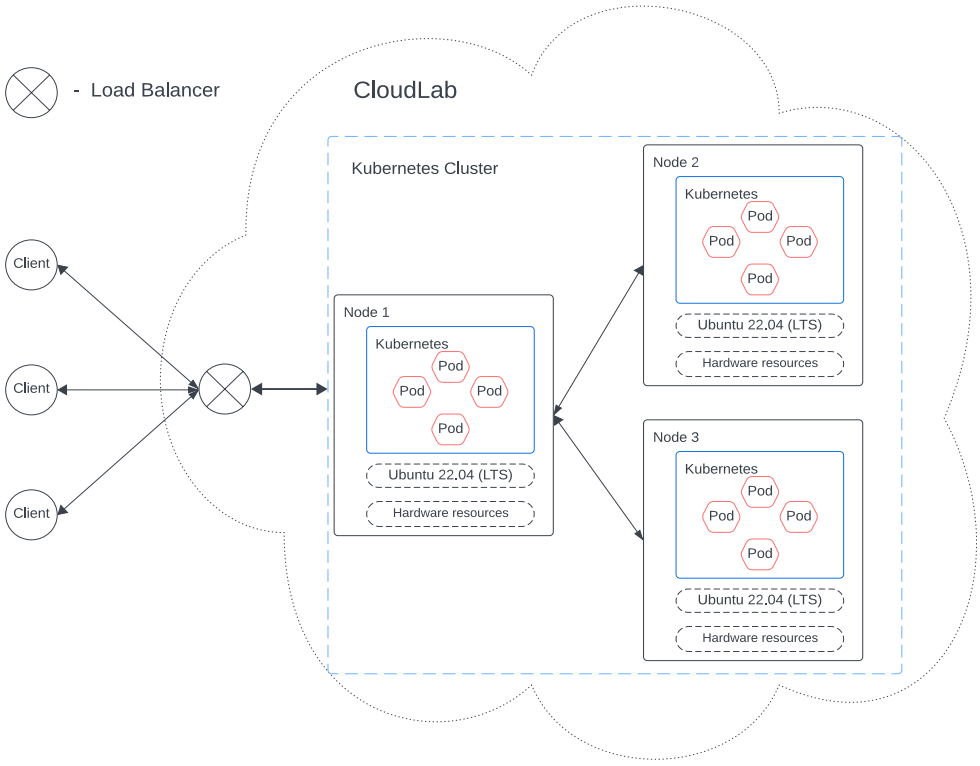

Figure 6: Schematic showing client interacting with the load balancer which forwards the requests to the cluster.

To earn full credits:

1. You must deploy an Ingress service and demonstrate that load-balancing is active
2. You must repeat the steps described in the simulation for task 3 section of this document except that instead of a global controller scheduling the jobs, you will use REST-API to schedule the jobs. You can implement a script that reads from the job queue but makes REST-API calls to the local-controller to schedule the jobs.
3. You must run jobs until node-1 is active and at least 2 pods are running in node-1. You will kill node-0 and start more jobs in node-1 until node-2 is active. When node-2 has at least 2 pods running, you must stop all the pods running in node-1 (using REST-API) to demonstrate that the global controller can scale-down.

Report (as part of task 3 submission):

You will submit a project report (5 – 7 pages, 1.5-line spaced, Arial 11point font) as part of the project submission. You can re-use content from your task 2 PDF. You must provide a brief introduction of the system, explain the system modeling techniques and the model that you obtained of the node, explain your controller design choices, the results of your experiments (MAX_PODS vs. time, total # of pods vs. time, total # of nodes in the cluster vs. time etc.), and a brief conclusion. Don't forget to include references. You must also include each team member's contribution and division of work.

If you attempted extra-credit, you must clearly state the extra-credits that you attempted and include detailed information on how you implemented the extra-credit system and show the results that are required as part of the extra-credit task. You get an additional page per extra-credit subtask that you attempt.

Submission (12/4/2023@11 pm): **No deadline extension requests will be considered**

1. Project report named <group#>_project.pdf
2. A TAR file named <group#>_project.tar that includes the following files:
    a. All the source code files that you have implemented as part of the project (task 2 and task 3)
    b. A single text file containing the various job queues that you used to simulate task 2 and task 3. You must include the job queues that simulated parts *d* and *f* of simulation section in task 2. You must include the job queues that simulated parts *f* and *g* of simulation section in task 3. Each job queue must be clearly marked with the task it achieves.
    c. A readme file that clearly explains the source code and what it achieves. You must also include the commands that are required to test your setup. You must provide step-by-step instructions to execute your source code. You do not have to provide instructions to install Docker and Kubernetes.
    Note: The TA will only copy-paste the command. It is your responsibility to ensure that there are no typos in the command that you provide in the readme for the TA to execute.
    d. A file named "makefile" which will install all the necessary libraries and packages to successfully run your source code from all the tasks. Refer to https://opensource.com/article/18/8/what-how-makefile for more information on makefile. The TA will only type "make" to install all the necessary libraries and packages.

    Note: As a graduate student, you are expected to write programs that are well documented (through comments) and are self-explanatory.

Submission policy:

- Submit a TAR file and a PDF file named as specified above. Do not submit makefile, and the source code as individual files. Incorrect submission formats will lead to a grade reduction.
- All submissions are expected by the deadline specified in the homework assignment. Grade is automatically reduced by **25% for every late day**.