**Overview**

The goal of this project is to recreate an elastic cloud infrastructure like Kubernetes. We used Flask to implement a REST API. Along with the REST API, we implemented an elastic cloud manager and a load balancer. Those 3 components runs on a virtual machine together. On another VM, we created 3 pods. These pods each contain multiple nodes. In this project, a node is Docker container. Therefore, a pod is a network of docker containers having similar resource limit. Clients will send request which will be handled by REST API. The load balancer will redirect the request to the least busy node. In this project, we defined only two resource limits: memory and number of CPUs. We have 3 pods: heavy, medium and light. Each pod has different resource limit with ‘heavy’ having the highest limits. See Fig.1.

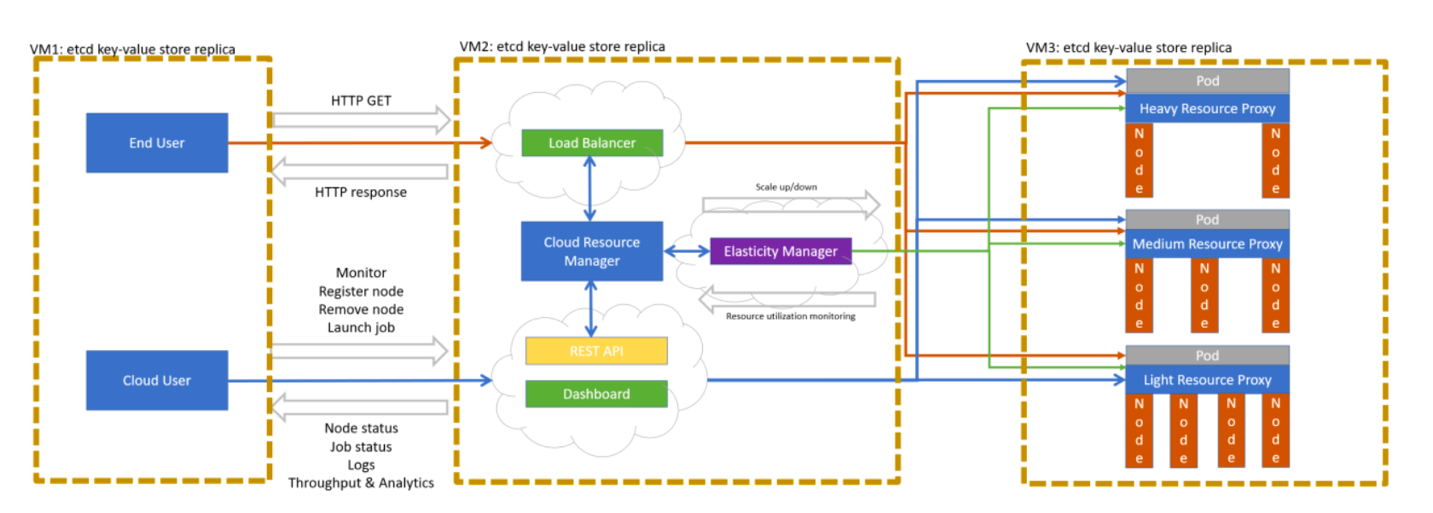


Fig.1: Cloud architecture

**Virtual machine 3**

In each pod, we create a docker network to allow the nodes(containers) to communicate between each other and each pod has its own docker image which tells a container that it needs to run a python script with the required python library. Each pod has its own python script. Each script requires different amount of resources when running. For example, the python file in the heavy pod is very computation heavy. Thus, greater resource limit needs to be set in this pod. Those python scripts represent jobs that are performed on the cloud. A client can send a GET request and this will launch a job. This will run the python script and a response will be sent back.   
  
Initially each pod is initialized with a fix number of nodes. When a client sends a job request, the job will be executed in 1 node (a docker container). If another client sends a request, the request will be executed in the current node unless the resource limit has been reached in that node. In that case, the job will need to be run in a different docker container.

**Virtual Machine 2**

On this VM, we ran the cloud resource manager. This resource manager consists of a REST API and a load balancer.

The REST API handles the GET request of the client and sends response back. It also handles the request from the cloud owner, who can start up the cloud and set up the cloud resource parameters like the initial number of nodes in each pod.

We used an open-source load balancer from HA proxy. The load balancer distributes the request among the nodes using Round-Robin scheduling algorithm.

We also implemented an elastic cloud manager that scales up or down the number of nodes in a pod based on the current user demand and the amount of resources being used by all the containers in the pod.