Load Balancing On IoT Fog

Christopher Gonzales

SYSTEM REPORT

REVISION – 1
4 December 2023

# Concept of Operations
for
Load Balancing On IoT Fog

Team <21>

Approved by:

Project Leader	Date
Prof. Kalafatis	Date
T/A	Date

Change Record

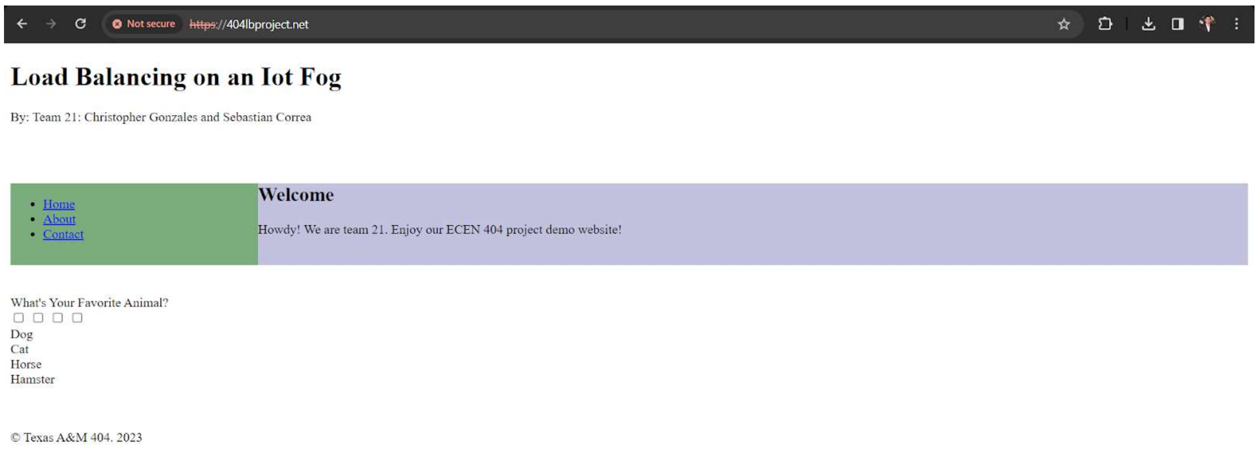
Rev.	Date	Originator	Approvals	Description
-	12/4/2023	Sebastian Correa		Draft Release
1	12/4/2023	Christopher Gonzales		Revision-1

Table of Contents

List of Figures

1 Overview

Load balancing systems are generally done on either local edge devices such as a set of computers or Raspberry Pi's or they are done on cloud services using virtual machines. We were tasked with creating a load balancing system between an edge device and cloud services and have loads traverse an IoT fog. This would grant users more accessibility and a more flexible approach to load balancing. This would also prevent machines from overloading and prevent applications from failing when the machines fail. For our system, we use a Raspberry Pi as our edge device and we will use Microsoft Azure's virtual machine service to establish our cloud nodes. The nodes communicate with one another using a service called Netmaker Networks and WireGuard VPNs. For our method of load balancing and containerization in our system, we use kubernetes, specifically a light-weighted version called K3s. To test our system, we deploy a website from our Raspberry Pi and have it be hosted on every worker node. The website will have a poll that allows users to answer and then transmits the data of the poll to the kubernetes cluster. End users would access the website using a URL that points to external IPs from our worker nodes.



Load Balancing on an Iot Fog

By: Team 21: Christopher Gonzales and Sebastian Correa

- Home
- About
- Contact

Welcome

Howdy! We are team 21. Enjoy our ECEN 404 project demo website!

What's Your Favorite Animal?
☐ ☐ ☐ ☐
Dog
Cat
Horse
Hamster

© Texas A&M 404. 2023

Figure 1: An example of our website

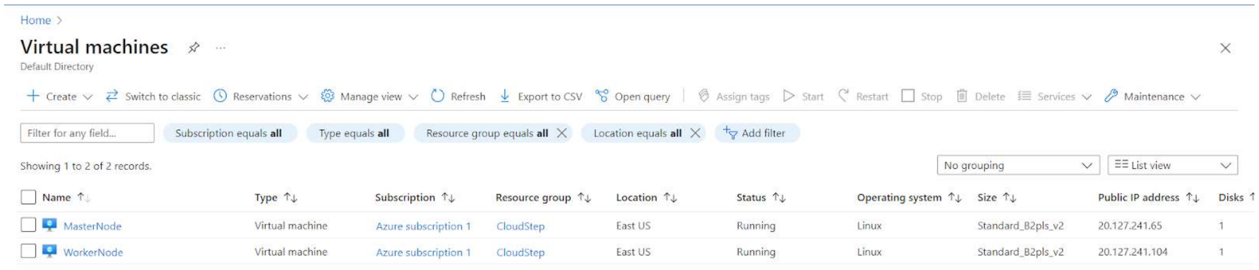
Loads and traffic will be balanced using a custom service and an ingress controller in our kubernetes cluster. Anybody accessing the website using the URL will produce traffic in our system. The purpose of the System Report is to review the choices we made to accomplish this task, the challenges we faced in the development process, and the integration between our Raspberry Pi and Cloud Service.

2 Design Plan & Execution

2.1 Design Plan

The original system involved using AWS as our cloud service with 3 worker nodes and a master node in the cloud. The nodes would be deployed in EC2 instances and load balancing would be done using an external load balancer in AWS. On another note, our original test for our system was to demonstrate using a video game server and have the server hosted on the Raspberry Pi.

Our test failed due to insufficient resources to host the video game server. Our subsystems also failed to connect to each other using a Site-to-Site or Point-to-Site connection. We also switched to Microsoft Azure because Azure provided $200 of free credit to be used for any resource they provided for 30 days and the department had stopped funding for cloud services.



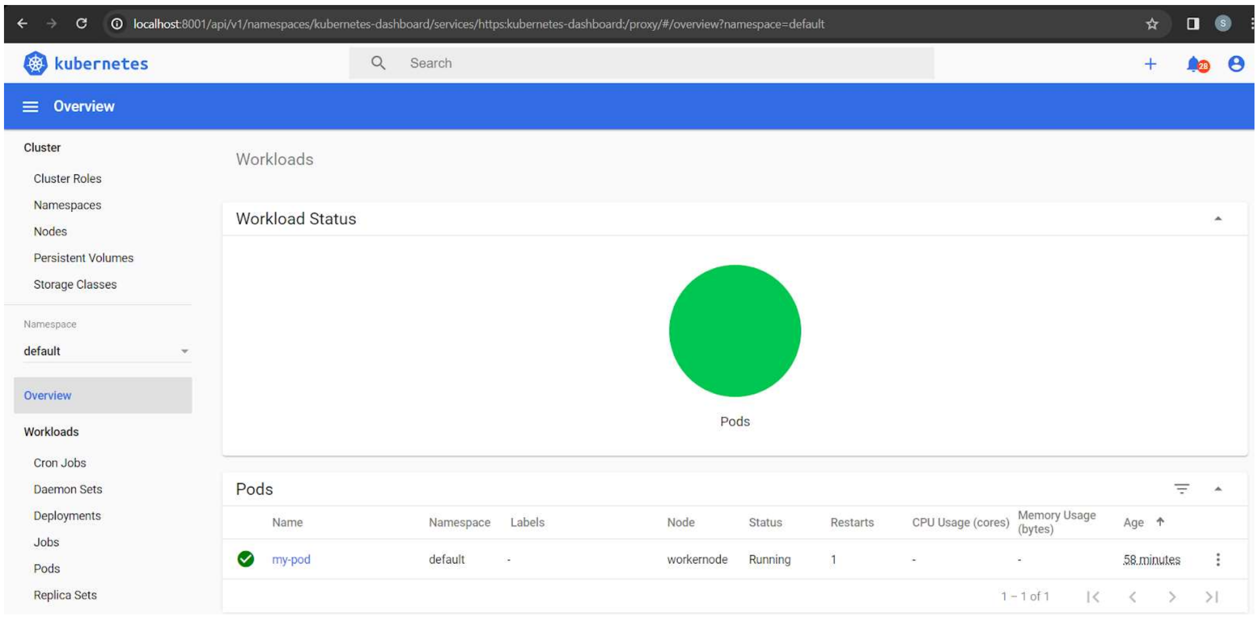
Showing 1 to 2 of 2 records.

Name	Type	Subscription	Resource group	Location	Status	Operating system	Size	Public IP address	Disks
MasterNode	Virtual machine	Azure subscription 1	CloudStep	East US	Running	Linux	Standard_B2pls_v2	20.127.241.65	1
WorkerNode	Virtual machine	Azure subscription 1	CloudStep	East US	Running	Linux	Standard_B2pls_v2	20.127.241.104	1

Figure 2: Virtual Machines deployed in Azure

We also reduced the number of worker nodes to one in the cloud to better demonstrate our system load balancing. We also changed our test to a website that would be deployed on the Pi and hosted on every worker node in the kubernetes cluster using a custom kubernetes service. The load balancing in our system would also change and now be done using an ingress controller in the kubernetes cluster.

Another issue we had was monitoring. The solution before was the monitoring system AWS had but the monitoring system would not show metrics for the Pi. Our new solution was to use Kubernetes Dashboard, provided by kubernetes to monitor the cluster and its nodes. While the dashboard can be used on the Pi, we decided it would be best to have a user use the dashboard on their local computer to prevent an excessive amount of memory and overloading on the Pi.



Name	Namespace	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Age
✓ my-pod	default	-	workernode	Running	1	-	-	58 minutes

1 – 1 of 1

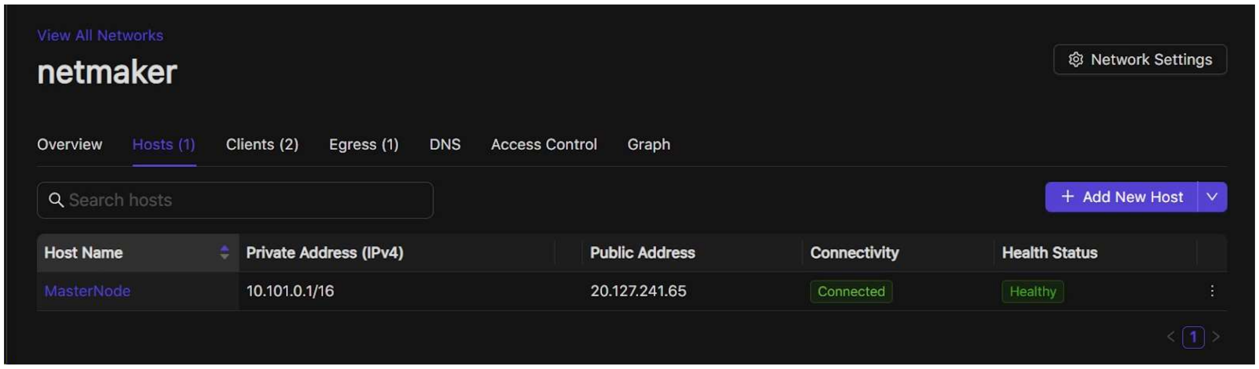
Figure 3: An example of our Kubernetes Dashboard

2.2 Execution Plan

In the early stages of our development process, we were able to make the transition from AWS to Microsoft Azure and create the website needed for our test. However, the main issue we faced was integrating our subsystems. The problem was that the Raspberry Pi could not access Azure's private virtual network. This was because we had to set up the Raspberry Pi under a Wi-Fi network and without access to the router's settings, we could not port forward.

Port forwarding was important because the Pi needed certain ports to be open to establish communication with Azure using VPNs. We needed to set the Pi under a Wi-Fi network because we could not find an open ethernet port that is not under a Wi-Fi network. There were solutions Azure provided but the solutions were not affordable.

After three weeks of communicating with each other and discussing with our sponsor, we tried a solution that consisted of using Netmaker Networks and a WireGuard VPN. Netmaker would let the master node in the cloud service become a gateway device and accept client gateways. The Pi subsystem would connect to its client gateway using a WireGuard VPN, which would allow the Pi to be added to the cluster and establish communication.



Host Name	Private Address (IPv4)	Public Address	Connectivity	Health Status
MasterNode	10.101.0.1/16	20.127.241.65	Connected	Healthy

Figure 4: Netmaker Network Dashboard

This not only connects the Pi to Azure, but it also means the Pi can be connected to any Wifi Network and still access the Cloud as long as it can run the WireGuard VPN. This connection also allows us to connect a local computer to the cluster so we can use the kubernetes dashboard.

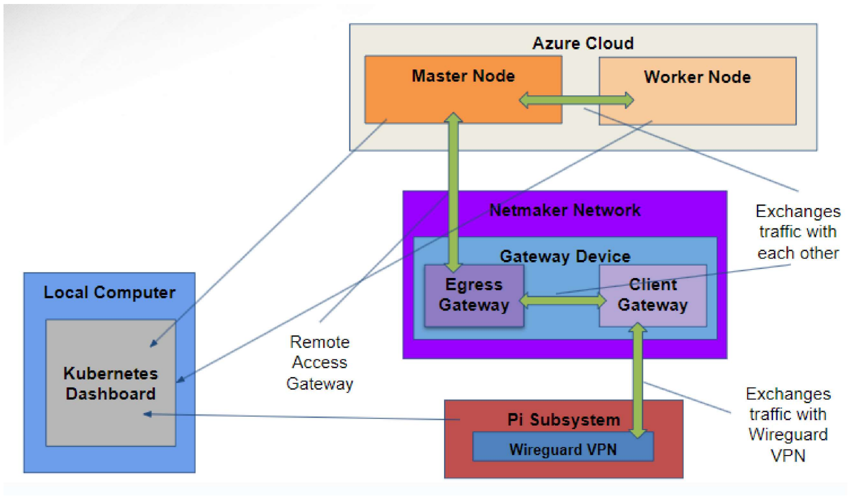


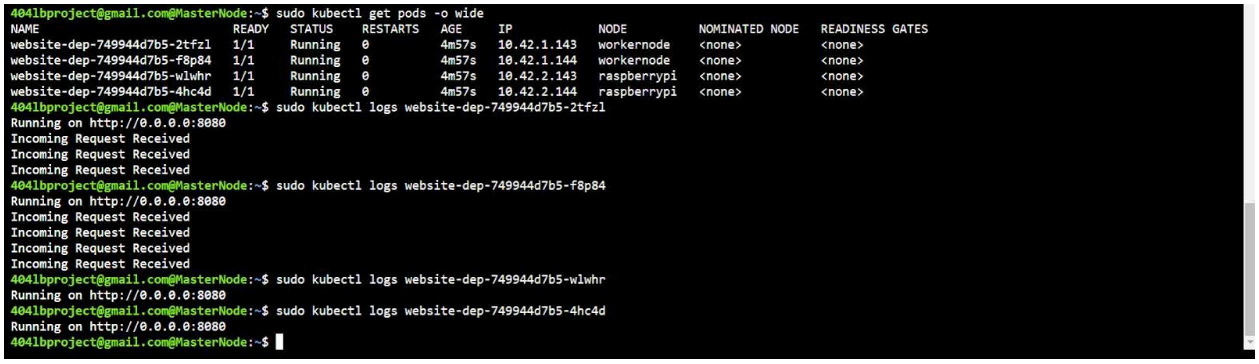
Figure 5: Netmaker Connection Diagram

After finishing integration another major issue we faced was that the worker nodes could not use kubectl commands, the main commands kubernetes use to monitor and edit resources in the cluster. This was resolved by copying the k3s configuration file to each worker node and creating an environment variable to point to the path to the file. Once this was configured, we were able to begin testing with services provided by Kubernetes such as Nginx and validate our system. A majority of our validations needed our subsystems integrated so were able to validate most of our system. Although we did have trouble validating the test. We considered using Prometheus and Grafana services but it produced another port issue. However we were able to determine the system was load balancing using the logs in pods in our worker nodes.

2.3 Validation Plan

The first validation we could do was to ensure that the Raspberry Pi could connect to the cloud. This was done using ping and telnet between the worker nodes and the master node IP. As long as the worker nodes and cloud could connect they would be in ready status for deploying any application. This was only possible after we used a point-to-point connection between the cloud and the Pi.

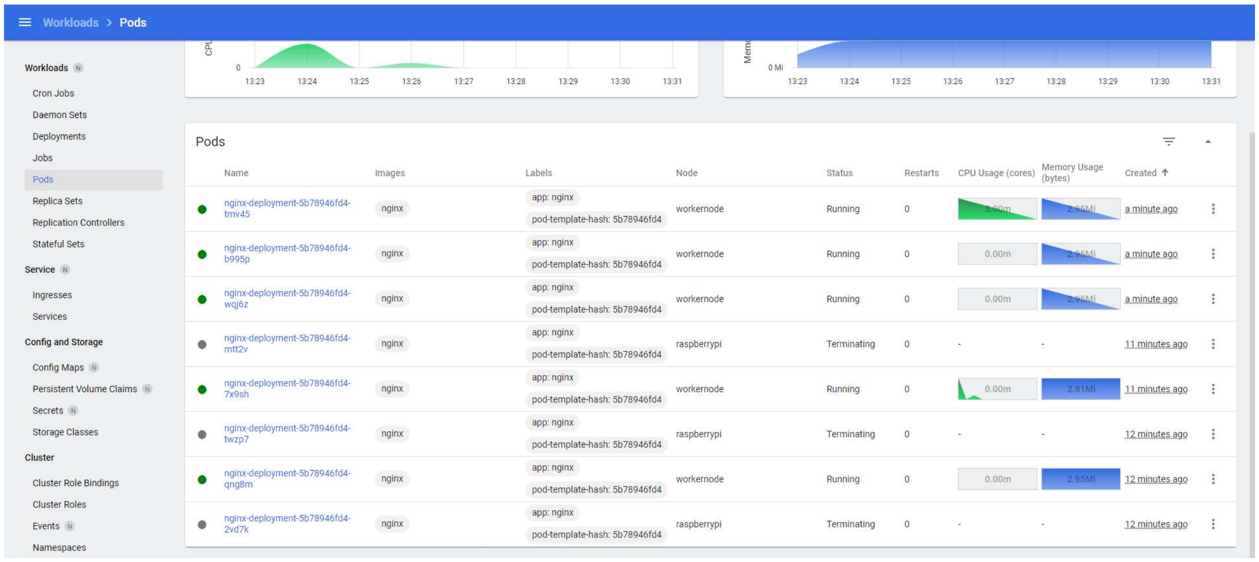
The main validation done was to ensure the load was being balanced between the worker nodes as well as handling any errors in the system. In our system, the pods created would be distributed equally between the Raspberry Pi edge node and the VM instance worker node. The website deployment included console logs that showed the movement of the loads. Once an end-user accesses the website, a line is written to the console explaining that a connection request was received. If we were able to port-forward on our home networks then we would be able to visually see the movements of the loads using the dashboard. Therefore, we had to adapt to see the load being balanced. Each of the pods in the cluster corresponded to one copy of the website on either worker node. So, if one person entered the URL to access the website it would be sent to one pod and that pod's console would show that the request was received. Therefore, the system was able to balance the load and we could view where each request was being sent.



```
404lbproject@gmail.com@MasterNode:~$ sudo kubectl get pods -o wide
NAME                         READY   STATUS    RESTARTS   AGE     IP            NODE          NOMINATED NODE   READINESS GATES
website-dep-749944d7b5-2tfzl   1/1     Running   0          4m57s   10.42.1.143   workernode    <none>           <none>
website-dep-749944d7b5-f8p84   1/1     Running   0          4m57s   10.42.1.144   workernode    <none>           <none>
website-dep-749944d7b5-wlwhr   1/1     Running   0          4m57s   10.42.2.143   raspberrypi   <none>           <none>
website-dep-749944d7b5-4hc4d   1/1     Running   0          4m57s   10.42.2.144   raspberrypi   <none>           <none>
404lbproject@gmail.com@MasterNode:~$ sudo kubectl logs website-dep-749944d7b5-2tfzl
Running on http://0.0.0.0:8080
Incoming Request Received
Incoming Request Received
Incoming Request Received
404lbproject@gmail.com@MasterNode:~$ sudo kubectl logs website-dep-749944d7b5-f8p84
Running on http://0.0.0.0:8080
Incoming Request Received
Incoming Request Received
Incoming Request Received
Incoming Request Received
404lbproject@gmail.com@MasterNode:~$ sudo kubectl logs website-dep-749944d7b5-wlwhr
Running on http://0.0.0.0:8080
404lbproject@gmail.com@MasterNode:~$ sudo kubectl logs website-dep-749944d7b5-4hc4d
Running on http://0.0.0.0:8080
404lbproject@gmail.com@MasterNode:~$
```

Figure 6: Console Showing Load Balancing

Another way we validated our system was by showing fault tolerance. We verified that the pod where the request is being directed is outputting that the request was received. We then shut down that worker node where the user is accessing the website. For example, in Figure 3 we see that the Raspberry Pi has been shut down and there is an error. Therefore, those pods currently being hosted on the pi are terminated and recreated on the VM worker node. In our system, we deployed 5 replicas of the website. Three of these pods were on the Pi. When the Pi got shut down these pods are created and run on the VM. This ensures that connections are still able to occur and the system can continue to run.



Pods

	Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created ↑
●	nginx-deployment-5b78946fd4-tmv45	nginx	app: nginx pod-template-hash: 5b78946fd4	workernode	Running	0	5.00m	2.95Mi	a minute ago
●	nginx-deployment-5b78946fd4-b995p	nginx	app: nginx pod-template-hash: 5b78946fd4	workernode	Running	0	0.00m	2.95Mi	a minute ago
●	nginx-deployment-5b78946fd4-wqj6z	nginx	app: nginx pod-template-hash: 5b78946fd4	workernode	Running	0	0.00m	2.95Mi	a minute ago
●	nginx-deployment-5b78946fd4-mtt2v	nginx	app: nginx pod-template-hash: 5b78946fd4	raspberrypi	Terminating	0	-	-	11 minutes ago
●	nginx-deployment-5b78946fd4-7x9sh	nginx	app: nginx pod-template-hash: 5b78946fd4	workernode	Running	0	0.00m	2.91Mi	11 minutes ago
●	nginx-deployment-5b78946fd4-twzp7	nginx	app: nginx pod-template-hash: 5b78946fd4	raspberrypi	Terminating	0	-	-	12 minutes ago
●	nginx-deployment-5b78946fd4-qng8m	nginx	app: nginx pod-template-hash: 5b78946fd4	workernode	Running	0	0.00m	2.95Mi	12 minutes ago
●	nginx-deployment-5b78946fd4-2vd7k	nginx	app: nginx pod-template-hash: 5b78946fd4	raspberrypi	Terminating	0	-	-	12 minutes ago

Figure 7: Dashboard Demonstrating Fault Tolerance.

As seen above in Figure 4 any pods that were running on the Raspberry Pi node are being terminated. new pods sooner than 11 minutes ago are created on the VM worker node. and then any access from the website is directed to any pods on that worker node. This process took approximately 5 minutes for the new pods to be created but it ensured that access to the website was possible at all times.

3 Conclusion

The overall system can be used to distribute a heavy workload between pods. It ensures that no worker node gets overloaded and can run at all times. We also can monitor the entire system and view the statistics of the cluster. We are also able to deploy an application once and it will be automatically distributed to the entire cluster. Using more worker nodes will only make the system more stable. In addition, using higher-quality hardware that can handle more heavyweight applications will allow for load-balancing apps that need it more than a simple website. Included is also a link to

the GitHub containing all of the code used to get the system running:
https://github.com/gonzalesch/404

3.1 Future Work

Moving forward, to ensure this system can be deployed properly port forwarding should be possible. Enabling port-forwarding allows for easier connection to the cluster if the hardware is stationary. Point-to-point connections are mainly useful if the system is changing networks. If the hardware is in one place it is simpler to use a point-to-site or site-to-site connection which is only possible with port-forwarding. Additionally, port-forwarding would allow for a simpler setup of the monitoring dashboard system. These port issues meant we were unable to view the loads moving between the nodes visually. Finally, by utilizing a more heavyweight application with better hardware the system could be more useful and load balance complex deployment applications.