Load Balancing on an IoT Fog

Christopher Gonzales
Sebastian Correa

FUNCTIONAL SYSTEM REQUIREMENTS

REVISION – 1
22 February 2023

Functional System Requirements
FOR
Load Balancing on an IoT Fog

Prepared by:

Author Date

Approved by:

Project Leader Date

John Lusher, P.E. Date

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	2/22/2023	Load Balancing on an IoT Fog		Revision 1
2	12/4/2023	Sebastian Correa		Revision 2
3	12/4/2023	Christopher Gonzales		Revision 3

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1. Purpose and Scope

Load balancing is a technique used to distribute incoming requests across multiple edge nodes to improve system performance and reliability. In the context of an IoT fog system, load balancing helps to ensure that all edge nodes are utilized equally, reducing the risk of overload or bottleneck situations. This specification defines the technical requirements for the development items and support subsystems delivered to the client for a load-balancing algorithm on an IoT fog. Figure 1 shows a representative integration of the project in the proposed CONOPS document. The verification requirements for the project are contained in a separate execution and validation Plan.

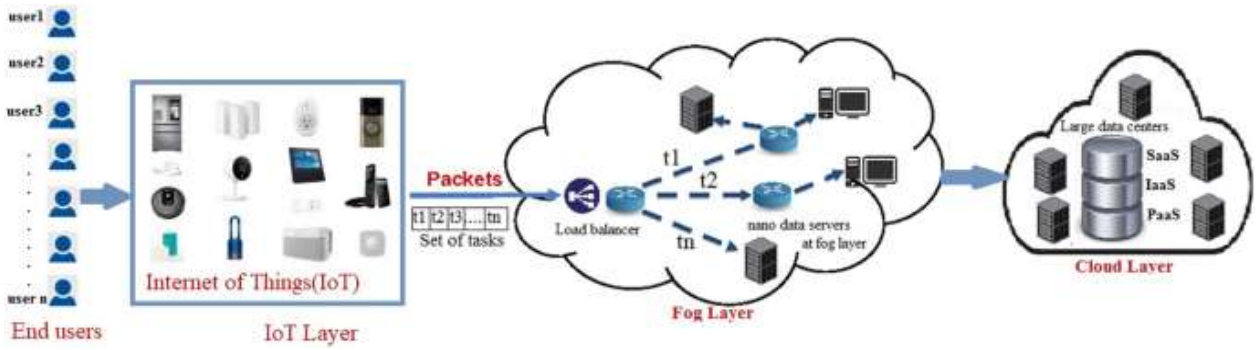

Figure 1: Load-balancer Conceptual Image

The following definitions differentiate between requirements and other statements.

Shall:	This is the only verb used for the binding requirements.
Should/May:	These verbs are used for stating non-mandatory goals.
Will:	This verb is used for stating facts or declaration of purpose.

1.2. Responsibility and Change Authority

The team leader, NAME, will be responsible for verifying all requirements of the project are met. These requirements can only be changed with the approval of the team leader and Professor Stavros Kalafatis.

Subsystem	**Responsibility**
Team Leader	Christopher Gonzales
RPI Edge Node	Christopher Gonzales

Cloud Server Node	Sebastian Correa

2. Applicable and Reference Documents

2.1. Applicable Documents

The following documents, of the exact issue and revision shown, form a part of this specification to the extent specified herein:

Document Number	Revision/Release Date	Document Title	Link
	21/06/2019 First release	RPI4 Datasheet	https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf
	2021 Dec 9	An Overview of Fog Computing and Edge Computing Security and Privacy Issues	https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8708798/#:~:text=However%2C%20fog%20computing%20also%20has,data%20management%2C%20and%20security%20threats.

Table 1: Applicable Document Info

2.2. Reference Documents

The following documents are reference documents utilized in the development of this specification. These documents do not form a part of this specification and are not controlled by their reference herein.

Document Number	Revision/Release Date	Document Title	Link
	10 October 2022	Energy-Efficient and Secure Load Balancing Technique for SDN-Enabled Fog Computing	https://www.mdpi.com/2071-1050/14/19/12951/pdf
	February 2021	A systematic study of load balancing approaches in	https://www.researchgate.net/publication/349041190_A_systema

		the fog computing environment	tic_study_of_load_balancing_approaches_in_the_fog_computing_environment

Table 2: Reference Document Info

2.3. Order of Precedence

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings, or other documents that are invoked as "applicable" in this specification are incorporated as cited. All documents that are referred to within an applicable report are considered to be for guidance and information only, except ICDs that have their relevant documents considered to be incorporated as cited.

3. Requirements

This section defines the minimum requirements that the development item(s) must meet. The requirements and constraints that apply to the performance, design, interoperability, reliability, etc. of the system are covered.

3.1. System Definition

The load balancer on an IoT fog is a system that will effectively distribute workloads among worker nodes in a kubernetes cluster. It will allow the application on the system to run more efficiently and securely. The system should never stop running as the distribution of traffic will account for maintenance and node failure. The system will have four main subsystems: the cloud server node, data monitoring subsystem, IoT fog network integration, and Raspberry Pi edge node.

3.2. Characteristics

3.2.1. Functional / Performance Requirements.

3.2.1.1. Distribution Requirement

The load balancer shall be able to take incoming traffic and distribute it to multiple pods among the worker nodes in the kubernetes cluster. It will determine how many pods will go in every worker node based on the total number of pods in the deployment resource used by a custom service

3.2.1.2. Traffic and Load Detection.

The load balancer shall detect the type of traffic it's taking in and know which pods among the worker nodes are the most efficient for taking in applications and data. It shall also handle SSL/TLS encryption without affecting the performance of the system. The pods in the worker nodes themselves will tell the user who has access to the kubernetes cluster using the custom messages we created for the logs in the pods.

> *Rationale: To be specific, the load balancer should be able to take HTTP/HTTPS traffic and route loads to pods based on the specific requirements of that application. SSL/TLS encryption and decryption of traffic can cause delays if not treated correctly, but our system will make sure the process will be quick and efficient.*

3.2.1.3. Monitoring and Management

Kubernetes Dashboard will be used to see a general overview of everything in our system. Netmaker Dashboard will be used to see how all the subsystems are integrated. If a user would like to see the logs of services, our load balancer, pods, or other resources, they would need to use Kubectl commands in any node in the kubernetes cluster.

3.2.1.4. Integration Connection

To communicate with the cloud, the Raspberry Pi will connect to a Netmaker Network using a WireGuard VPN to create a Point-to-Point connection. The kubernetes dashboard will also need the Pi or a local computer connected to the Netmaker Network in order to view the dashboard

3.2.1.4.1 Netmaker Dashboard

In this dashboard, a user can add hosts and clients to the network and configure gateways to make everything connect in the system.

3.2.1.4.2 Egress Gateway

The egress gateway makes a device a host using a remote access gateway. The host will be the main source of every connection in the netmaker network and communicate with client gateways in the network to establish connections.

3.2.1.4.3 Client Gateway

The client gateway is used to connect clients to the network so they can communicate with the host of the network. Clients can connect to the gateway using a WireGuard VPN.

3.2.1.4.4 WireGuard VPN

The WireGuard VPN is used to connect to client gateways in the Netmaker Network. It requires no port forwarding and any advice can use the VPN to make a Point-to-Point connection.

3.2.1.4. Response Time

With a high amount of traffic, the load balancer shall maintain the distribution process without affecting the response time. We should have a response time that's less than 500ms.

3.2.2. Physical Characteristics

3.2.2.1. Power Supply

The Raspberry Pi 4 should have a stable and consistent power supply to avoid any interruptions to the load balancer. A 5V and 5A power supply will be perfect for the operation.

3.2.2.2. Cooling

To withstand the heat of the load balancing process, high traffic, and a long period of runtime, we will add a passive heat sink to the Raspberry Pi 4.

3.2.2.3. Network Connection

Wi-Fi could be a potential problem for the process because it can be interrupted. Therefore, we will use an ethernet cable to connect the system and prevent disconnections from the internet.

3.2.2.4. Cloud Infrastructure

Microsoft Azure has a list of services we can use in order to produce a functioning server node in our IoT Fog. Azure also has a flexible management system for us to configure our virtual machines.

> *Rationale: Our sponsor suggested we use Amazon EC2 to send EC2 instances to the load balancer and Amazon RDS to operate databases in the cloud. Due to the department not funding for cloud services for our time in ECEN 404, we decided to switch to Microsoft Azure and use virtual machines to create a master node and worker node for our load balancing system.*

3.2.3. Software Requirements

3.2.3.1. Operating System

The Raspberry Pi System needs an OS to incorporate the load balancing system, monitor traffic, and other functional requirements. We shall use Raspbian if Azure can operate with this OS or Ubuntu since Azure virtual machines will most likely run on Linux.

3.2.3.2. Containerization (K3s)

We shall use K3s to isolate applications and run them in their own pod. This is so the load balancer can take the pods and distribute them between the Raspberry Pi 4 and the cloud's

worker node. The K3s architecture will have two main components: The Kubernetes master node and the Kubernetes worker nodes.

Rationale: There are alternatives for containerization such as only using Docker but our sponsor would like us to use K3s to establish containerization for our system.

3.2.3.2.1. The Kubernetes Control Plane

This part of the Kubernetes cluster is the set of components that manage and control the cluster's behavior. This includes the API server, etcd, scheduler, and controller manager. This component is a portable option for the orchestration of pods but the control plane is only on the master node of the kubernetes cluster.

3.2.3.2.1.1. API Server

The API Server acts as the front-end to the control plane and is a representational state transfer interface as a way to communicate and exchange data between clients and the Kubernetes cluster. The worker node's also communicate with the API server to establish themselves as worker nodes.

3.2.3.2.1.2. ETCD

This is a distributed key-value store that stores the configuration data in the cluster and takes place in the master node.

3.2.3.2.1.3. Scheduler and Controller Manager

The scheduler and controller manager are in charge of maintaining the state of the Kubernetes cluster and takes place in the master node.

3.2.3.2.2. Kubernetes Worker Nodes

The Kubernetes worker nodes connect to the control plane of the cluster. The nodes are in charge of running the pods hosting the applications and data and contain the resources for completing the items. With their connection to the control plane, they receive instructions from the plane on what to do with the loads along with sending feedback to the control plane about the status of pods and the node's resources.

3.2.3.2.2.1 K3s agent

The K3s agent is part of the worker nodes that communicates with the control plane to manage the deployment and scaling of pods. It manages and executes the workload on its worker node and can also provide local storage for the pods and workloads. This replaces components such as Kube-proxy and Kubelet.

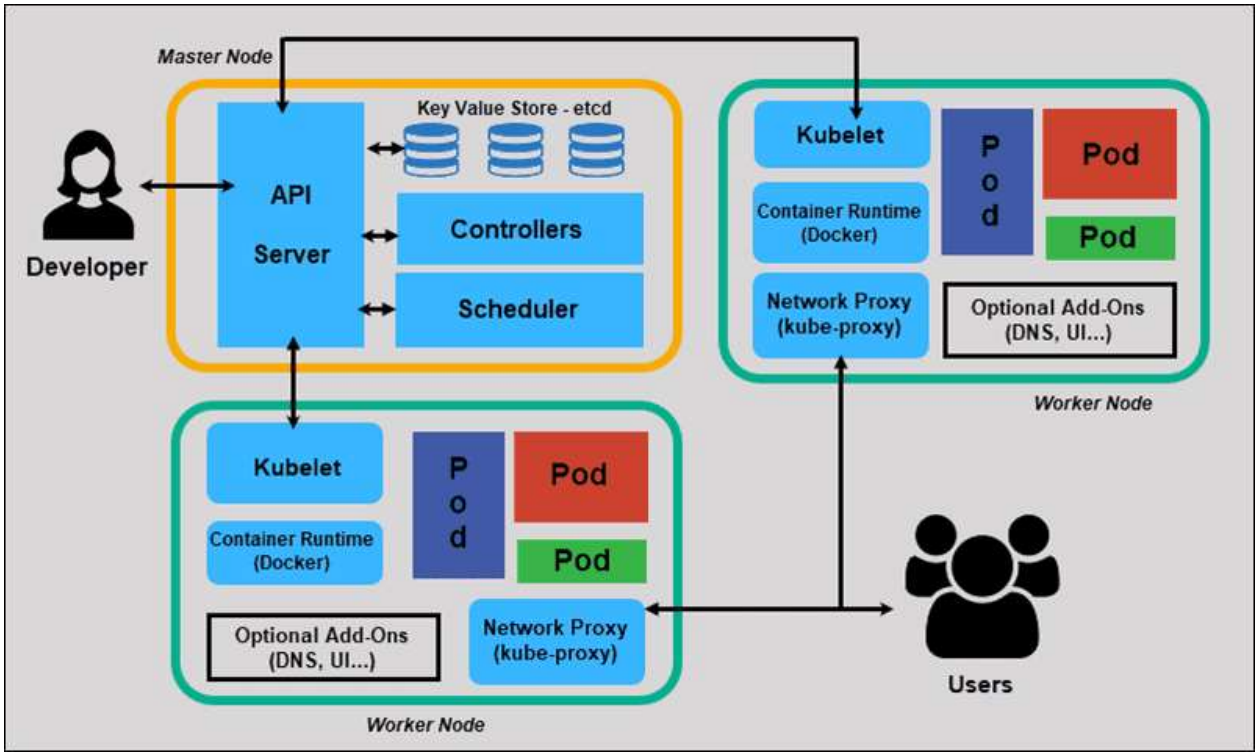

Figure 2: Kubernetes Architecture Block Diagram

3.2.3.2.3. Nginx Ingress Controller

An Nginx Ingress Controller will be of type “LoadBalancer” and use weighted round robin as its load balancing technique to create the load balancing system. It will communicate with our custom service to determine which pods it can send the loads too. The external IPs the ingress controller provides will be pointed toward a URL called 404lbproject.com

3.2.3.2.4. Custom service

A custom service with type “NodePort” will be created to expose a port on every worker node to deploy our website on. The ingress controller will communicate with the service to determine where loads will go.

Figure 3: Ingress controller communication with custom service

3.2.3.3. Python Packages

For developing our algorithm, we will need a set of Python packages to accomplish the task. These packages consist of

- Flask - To create the framework for handling the incoming traffic.
- Requests - To make HTTP requests to the cloud on AWS.
- Boto3 - Allows us to interact with Azure Virtual Machines and manage the load balancer.
- Kubernetes Client - To gather information about the running pods and modify the Kubernetes cluster.
- NumPy - To help with computation for distributing pods in the load balancer.

3.2.3.4. Docker

Used to containerize the website. Uses dockerfiles to build the docker image that will be used to host the website. docker image is pushed to my dockerhub account and used to be deployed to the entire cluster

4. Support Requirements

4.1. Laptop or PC

A laptop or PC is needed to use the kubernetes dashboard, work on the cloud, or connect the Raspberry Pi 4 to install the OS system on the device. Microsoft Azure is only accessible through a computer so this is necessary. The computer can also be used to work on the Raspberry Pi 4 in case you do not have a keyboard or a monitor to work on the Raspberry Pi 4. Fortunately, every member of our team has a computer.

4.2. Ethernet Cable

We need an ethernet cable to connect the Raspberry Pi 4 to the internet and establish an initial connection before it can connect to the internet automatically. The length does not matter, as long as we connect the pi and we can still use it. We contain a few ethernet cables so there's no need to purchase one.

4.3. Monitor

To view the process of our load balancer and make adjustments based on the logs and information the system is giving us, we need a monitor to see the progress. We possess a curved monitor for this sole purpose, we just need to configure the Raspberry Pi's OS for it to connect with the display.

Appendix A: Acronyms and Abbreviations

mA	Milliamp
mW	Milliwatt
MHz	Megahertz (1,000,000 Hz)
TBD	To Be Determined
RPI	Raspberry Pi (4)
V	Volts
A	Amps
AWS	Amazon Web Services
In	Inch
mm	Millimeter
OS	Operating System

Appendix B: Definition of Terms