National College of Ireland

Configuration Manual

Cloud Computing

Divi Vishnu Praneeth
Student ID: x23293080

School of Computing
National College of Ireland

Supervisor: Shrayas Setlur Arun

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Vishnu Praneeth Divi
Student ID:	x23293080
Programme:	cloud computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Shreyas Setlur Arun
Submission Due Date:	11/8/25
Project Title:	Configuration
Word Count:	w3
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Divi Vishnu Praneeth
Date:	11th August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Divi Vishnu Praneeth
x23293080

1 Introduction

This manual provides a guide for setting up the experimental environment used in the thesis, "Maximizing Power Efficiency in k8s using Custom Scheduler". It details the necessary hardware and software configurations, Kubernetes cluster deployment, custom scheduler implementation, and monitoring tool setup.

2 Tools and Technologies Required

To replicate the experimental setup, the following tools and technologies are required:

Category	Component	Version	Purpose
Operating System	Ubuntu Server	20.04 LTS (Recommended)	Base OS for Kubernetes nodes
Container Runtime	Docker	20.10.12 or compatible	Containerization engine
Orchestration	Kubernetes	1.20.0 or compatible	Cluster management
Kubernetes Tools	`kubeadm`, `kubectl`, `kubelet`	Compatible with Kubernetes version	Cluster bootstrapping and management
Programming Language	GoLang	1.18 or higher	Custom scheduler development
Monitoring	Prometheus	Latest stable	Time-series database for metrics
Node Metrics Collector	Node Exporter	Latest stable	Host-level metrics collection
Power Metrics Collector	Kepler	Latest stable	Power consumption metrics collection
Utilization Monitor	LoadWatcher	Latest stable	Resource utilization monitoring
Metrics Visualization	Grafana	Latest stable	Metrics visualization
Workload Generation	`stress-ng`	Latest stable	CPU stress testing
Cloud Platform	AWS EC2 / IBM Cloud	Bare-metal instances recommended	Hosting the Kubernetes cluster

Table 1: Detailed Table of System Components, Versions, and Their Purposes

3 Kubernetes Cluster Setup

This section outlines the steps to setup a kubernetes cluster. The theis utilized a heterogeneous cluster on bare-metal IBM cloud machines but the steps are generalized applicable to other cloud providers or on-premise setups with Ubuntu Server 20.04 LTS.

3.1 Node Provisioning

Provision three Ubuntu 20.04 LTS instances: - Master Node: 4vpcu, 8 GiB RAM minimum (e.g., AWS EC2 t2.medium or equivalent) - Worker Node: 16 vpcu, 128 Gib RAM (IBM mx3-metal) and 16 vpcu, 256 Gib RAM (IBM vx3-metal) (e.g., AWS EC2 m5.metal and m5zn.metal)

3.2 Initial Server Configuration (All Nodes

1. **Switch to root user**: bash sudo -i

2. **Disable Swap**: kubernetes requires swap to be disabled. Edit /etc/fstab to comment the swap line and then disable swap. bash

   ```
   sudo swapoff -a
   sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
   ```

3. **Install Container Runtime (Docker)**:

   ```
   bash sudo apt update sudo apt install -y apttransport-https \
   ca-certificates curl software-properties-common \
   curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
   sudo gpg --dearmor -o \
   /usr/share/keyrings/docker-archive-keyring.gpg \
   echo "deb [arch=$‘(dpkg --printarchitecture) \
   signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] \
   https://download.docker.com/linux/ubuntu ‘$(lsb_release -cs) stable" \
   | sudo tee
   /etc/apt/sources.list.d/docker.list > /dev/null sudo apt update \
   sudo apt install -y
   ```

4. **Configure Docker Daemon**: create or modify

   ```
        /etc/docker/daemon.json to use systemd as the
   cgroup driver. json { "exec-opts": ["native.cgroupdriver=systemd"],\
   "log-driver":
   "json-file", "log-opts": { "max-size": "100m" }, "storage-driver": "overlay2" }\
   Reload and restart Docker: bash sudo systemctl daemon-reload \
   sudo systemctl restart
   ```

5. **Install kubeadm, kubelet and kubectl**:

   ```
   bash sudo apt update sudo apt install -y apttransport-https curl \
   curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg \
   sudo apt-key add - echo "deb https://apt.kubernetes.io/ \
   kubernetes-xenial main" \
   sudo tee /etc/apt/sources.list.d/kubernetes.list \
   sudo apt update sudo apt install -
   ```

3.3 Master Node Initialization

: On the **Master Node** only:

1. **Initialize Kubernetes with kubeadm**: choose a Pod network CIDR (e.g., 192.168.0.0/16 for calico).

```
bash sudo kubeadm init --pod-network-cidr=192.168.0.0/16
```

After successful initialization, kubeadm will provide commands to set up kubectl for the current user and a kubeadm join command for worker nodes. Copy these commands.

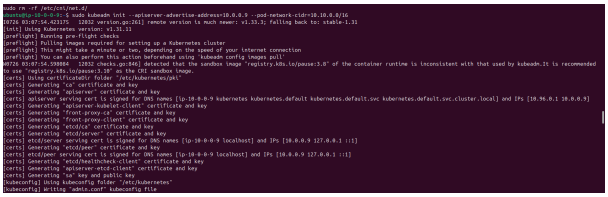


Figure 1: kubernetes cluster init

2. **Configure kubectl for current user**:

```
bash mkdir -p $HOME/.kube sudo cp -i
/etc/kubernetes/admin.conf $HOME/.kube/config sudo chown $‘(id -u):‘$(id -g)
```

3. **Install Pod Network (calico)**:

```
bash kubectl apply -f
kubectl create -f https://raw.githubusercontent.com/ \
projectcalico/calico/v3.28.0/manifests/tigera-operator.yaml
bash kubectl apply -f kubectl create -f https://raw.githubusercontent.com/ \
projectcalico/calico/v3.28.0/manifests/custom-resources.yaml
```

3.4 Worker Node Join

On each **Worker Node:**

1. **Join the cluster**: use the kubeadm join command obtained from the master node initialization. it will look something like :

```
        bash sudo kubeadm join <master-ip>:<master-port> --token
<token> --discovery-token-ca-cert-hash sha256:<hash>
```

3.5 Verify Cluster Status

On the **Master Node**:

1. **Check node status**: bash kubectl get nodes. All nodes will show ready status.

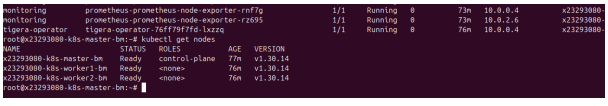


Figure 2: Nodes Ready Status

4 Custom Scheduler Implementation

This Section details the steps to deploy and configure the custom scheduler

4.1 GoLang Environment Setup (On Master Node)

install GoLang 1.18 or higher

```
sudo apt install -y golang-go
```

4.2 clone and Build the Custom Scheduler

1. **Clone the Scheduler repository**: clone the custom scheduler repository as it is published to my git which is public

   ```
   bash git clone https://github.com/Ursvishnnu/custom-scheduler
   ```

2. **Build the Kubernetes binary**: after cloning checkout to scheduler-plugins folder under build folder go to scheduler folder and run

   ```
   docker build scehdulerbin:v1 .
   ```

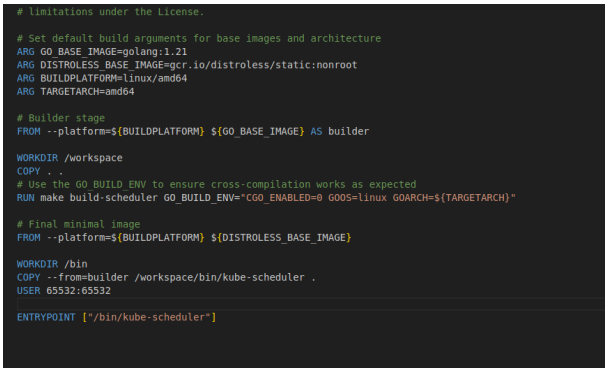


Figure 3: scheduler bin dockerfile

3. **Build the Custom Scheduler**: after building the kuberentes scheduler binary we need to attach our custom scheduler config to it / go to 'scheduler-plugins/pkg/trimaran /peaks/deployment' and run

   ```
   docker build customscheduler:v1 .
   after building  push to you remote image repository so that kubernetes will\
   fetch this image while deploying custom scheduler
   ```

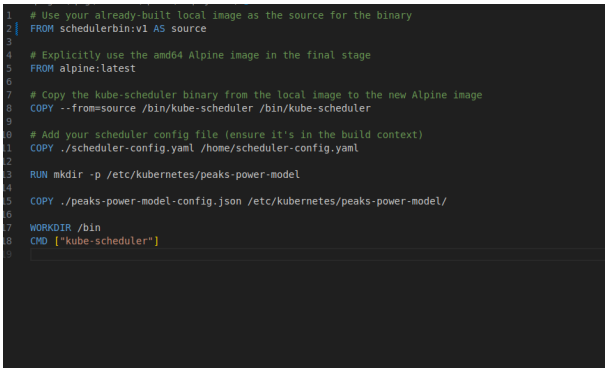


Figure 4: Custom scheduler dockerfile

4.3 Deploy the Custom Scheduler to kubernetes

1. **Create Kubernetes Deployment for the Custom scheduler**: create an YAML file like 5 to deploy your scheduler as pod in the

   `kube-system namespace`

   This Deployment will create your scheduler run along with Default kubernetes scheduler

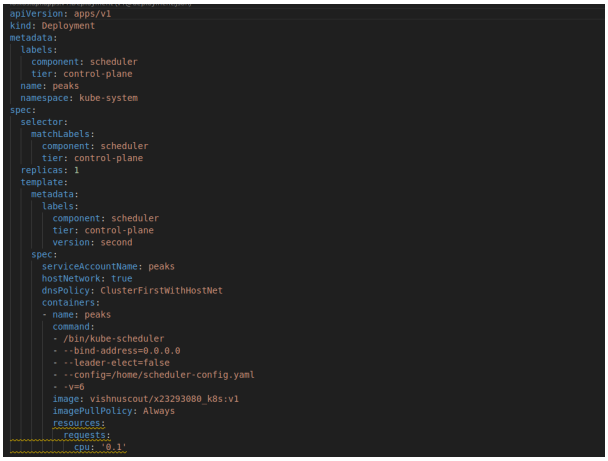
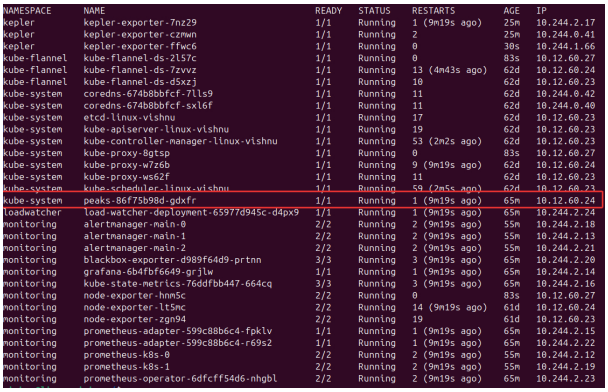


Figure 5: Custom scheduler deployment

2. **Apply the deployment**: Apply the above deployment file with the command

   `kubectl apply -f <deployment_fileName>.yaml`

3. **Verify the Scheduler status**: After applying you can check the custom scheduler will deployed into kube-system namespace



NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP
kepler	kepler-exporter-7nz29	1/1	Running	1 (9m19s ago)	25m	10.244.2.17
kepler	kepler-exporter-czrwn	1/1	Running	2	25m	10.244.0.41
kepler	kepler-exporter-ffwc6	1/1	Running	0	30s	10.244.1.66
kube-flannel	kube-flannel-ds-2l57c	1/1	Running	0	83s	10.12.60.27
kube-flannel	kube-flannel-ds-7zvvz	1/1	Running	13 (4m43s ago)	62d	10.12.60.24
kube-flannel	kube-flannel-ds-d5xzj	1/1	Running	10	62d	10.12.60.23
kube-system	coredns-674b8bbfcf-7lls9	1/1	Running	11	62d	10.244.0.42
kube-system	coredns-674b8bbfcf-sxl6f	1/1	Running	11	62d	10.244.0.40
kube-system	etcd-linux-vishnu	1/1	Running	17	62d	10.12.60.23
kube-system	kube-apiserver-linux-vishnu	1/1	Running	19	62d	10.12.60.23
kube-system	kube-controller-manager-linux-vishnu	1/1	Running	53 (2m2s ago)	62d	10.12.60.23
kube-system	kube-proxy-8gtsp	1/1	Running	0	83s	10.12.60.27
kube-system	kube-proxy-w7z6b	1/1	Running	9 (9m19s ago)	62d	10.12.60.24
kube-system	kube-proxy-ws62f	1/1	Running	11	62d	10.12.60.23
kube-system	kube-scheduler-linux-vishnu	1/1	Running	59 (2m5s ago)	62d	10.12.60.23
kube-system	peaks-86f75b98d-gdxfr	1/1	Running	1 (9m19s ago)	65m	10.12.60.24
loadwatcher	load-watcher-deployment-65977d945c-d4px9	1/1	Running	1 (9m19s ago)	65m	10.244.2.24
monitoring	alertmanager-main-0	2/2	Running	2 (9m19s ago)	55m	10.244.2.18
monitoring	alertmanager-main-1	2/2	Running	2 (9m19s ago)	55m	10.244.2.13
monitoring	alertmanager-main-2	2/2	Running	2 (9m19s ago)	55m	10.244.2.21
monitoring	blackbox-exporter-d989f64d9-prtnn	3/3	Running	3 (9m19s ago)	65m	10.244.2.20
monitoring	grafana-6b4fbf6649-grjlw	1/1	Running	1 (9m19s ago)	65m	10.244.2.14
monitoring	kube-state-metrics-76ddfbb447-664cq	3/3	Running	3 (9m19s ago)	65m	10.244.2.16
monitoring	node-exporter-hnm5c	2/2	Running	0	83s	10.12.60.27
monitoring	node-exporter-lt5mc	2/2	Running	14 (9m19s ago)	61d	10.12.60.24
monitoring	node-exporter-zgn94	2/2	Running	19	61d	10.12.60.23
monitoring	prometheus-adapter-599c88b6c4-fpklv	1/1	Running	1 (9m19s ago)	65m	10.244.2.15
monitoring	prometheus-adapter-599c88b6c4-r69s2	1/1	Running	1 (9m19s ago)	65m	10.244.2.22
monitoring	prometheus-k8s-0	2/2	Running	2 (9m19s ago)	55m	10.244.2.12
monitoring	prometheus-k8s-1	2/2	Running	2 (9m19s ago)	55m	10.244.2.19
monitoring	prometheus-operator-6dfcff54d6-nhgbl	2/2	Running	2 (9m19s ago)	65m	10.244.2.23

Figure 6: Custom scheduler deployed in kube-system

5 Monitoring Tools Configuration

This section details the setup of Prometheus, Node Exporter, kepler, Loadwatcher and grafana for monitoring.

5.1 Prometheus ,Grafana and Node Exporter Setup

Prometheus will be installed on Master Node and Node Exporter on all nodes (Master and Worker Nodes) to collect host-level metrics

1. **install helm**:

   ```
   curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
   ```

2. **Download and install Prometheus & Grafana via helm**:

   ```
   helm install prometheus prometheus-community/kube-prometheus-stack \
   --namespace monitoring \
   --create-namespace \
   --wait \
   --set prometheus.prometheusSpec.maximumStartupDurationSeconds=120
   ```

3. **Reload Systemd and start Prometheus (Master Node)**:

   ```
   bash sudo systemctl daemon-reload
   sudo systemctl start prometheus
   sudo systemctl enable prometheus sudo systemctl status prometheus ,
   Prometheus UI should be accessible at http://<deployed_node_ip>:9090 .
   Grafana UI should be accessible at
   http://<deployed_node_ip>:3000
   ```

4. **verify the pods**: After successfull installation and u can verify the pods of prometheus and grafan deployed on monitoring namespace

```
kubectl get pods -n monitoring
```

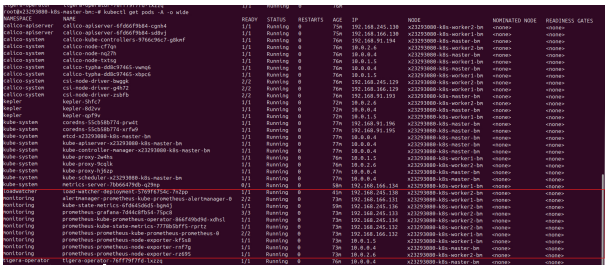


Figure 7: Prometheus and grafan pods running on monitoring namespace

5.2 Kepler and LoadWatcher Setup

kepler and loadwatcher are important for collecting power and resource utilization metrics within the kubernetes cluster. These will be deployed as DeamonSets

1. **Deploy Kepler (Master Node)**: Follow the command install kepler via helm:

```
helm install kepler kepler/kepler \
--namespace kepler \
--create-namespace \
--set serviceMonitor.enabled=true \
--set serviceMonitor.labels.release=prometheus \
```

2. **Deploy LoadWatcher (Master Node)**: Follow the command to install loadwatcher via git clone

```
git clone https://github.com/paypal/load-watcher && \
cd manifests  \
&& kubectl apply -f load-watcher-deployment.yaml
```

3. **verify the Pods**

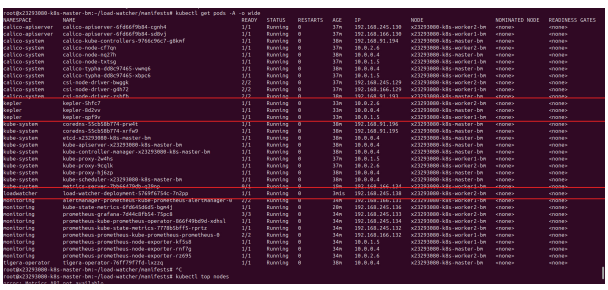


Figure 8: kepler and loadwtacher running across all nodes

4. **Import Grafana Dashboards**: import custom written built dashboards which will be available in the repo where it consists of Node Exporter, kepler metrics

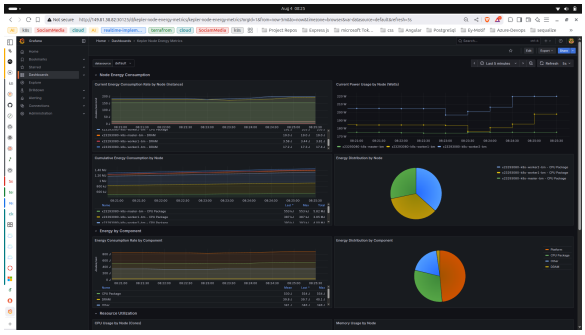


Figure 9: container power utilization

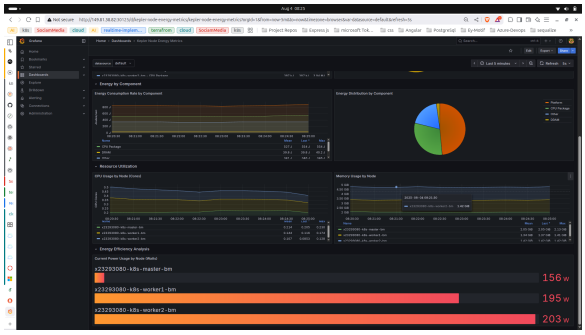


Figure 10: grafana rendering power metrics

6 Workload Generation and Testing

The section describes how can we generate workloads and conduct tests to evaluate the custom scheduler

6.1 stress-ng Workload Generation

stress-ng is used to create specific CPU loads , can deploy stress-ng as a kubernetes Deployment

Deployment for stress-ng (using default scheduler):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: default-stress-deployment
  labels:
    app: cpu-stress
    scheduler: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cpu-stress
      scheduler: default
  template:
    metadata:
      labels:
        app: cpu-stress
        scheduler: default
    spec:
      containers:
      - name: cpu-stress
        image: polinux/stress
        resources:
```

```
          requests:
            cpu: 100m
            memory: 100Mi
          limits:
            cpu: 500m
            memory: 200Mi
        command: ["stress"]
        args: ["--cpu", "16", "--timeout", "3600s"]
      schedulerName: default-scheduler # Explicitly use default scheduler
```

Example Job for stress-ng (using custom scheduler):

```
apiVersion: batch/v1
kind: Deployment
metadata:
  name: custom-stress-job
spec:
  template:
    metadata:
      labels:
        app: cpu-stress-job
        scheduler: custom
    spec:
      containers:
      - name: cpu-stress
        image: polinux/stress
        resources:
          requests:
            cpu: 400m
            memory: 100Mi
          limits:
            cpu: 800m
            memory: 200Mi
        command: ["stress"]
        args: ["--cpu", "16", "--timeout", "600s"]
      restartPolicy: Never
      schedulerName: custom-scheduler # Explicitly use  custom scheduler name
```

6.2 Horizontal Pod Autoscaler (HPA) Configuration

Configure HPA to simulate dynamic scaling scenarios based on CPU utilization
Example HPA for Deployment

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: stress-hpa
```

```
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: default-stress-deployment # Or your custom-scheduler deployment
  minReplicas: 1
  maxReplicas: 30
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

6.3 Running Experiments

1. **Deployment Comparison**: Deploy identical workloads using both the default-scheduler and your custom-scheduler. Monitor energy consumption and resource utilization via Grafana and shell script is provided in the git repo where it can executed to generate graphs

2. **Scaling Comparison**: Apply HPA to deployments and observe how each scheduler handles dynamic scaling events in terms of energy efficiency and resource placement. For these as well a schell script is provided in the git repo where it can be executed to generate graphs

3. **Data Collection**: Ensuring Prometheus is scraping metrics at regular intervals (e.g., 10 seconds) and use Grafana to visualize and export data for analysis.

7 Troubleshooting and common issues

Below are some common issues and checks to ensure smooth operation of your Kubernetes experimental setup:

1. **Kubernetes Version Incompatibility:** Ensure all Kubernetes components (`kubeadm`, `kubelet`, `kubectl`) and your custom scheduler are compatible with the chosen Kubernetes version.

2. **Firewall Rules:** Verify that necessary ports are open between nodes and for external access to Prometheus and Grafana.

3. **Prometheus Scrape Issues:** Check Prometheus target status in its UI at `http://<master-node-ip>:9090/targets` to ensure all exporters (Node Exporter, Kepler, LoadWatcher) are being scraped correctly.

4. **Custom Scheduler Not Scheduling:** Check custom scheduler logs using:

   ```
   kubectl logs <custom-scheduler-pod-name> -n kube-system
   ```

Ensure the `schedulerName` in your pod/deployment YAML matches your custom scheduler's name.

5. **Resource Limits:** Ensure your nodes have sufficient resources (CPU, memory) to run the Kubernetes components, monitoring tools, and workloads.

References

1. Kubernetes Authors, "*Kubernetes Documentation.*" . Available: `https://kubernetes.io/docs/`. Accessed: Aug. 8, 2025.

2. Prometheus Authors, "*Prometheus Documentation.*" Available: `https://prometheus.io/docs/`. Accessed: Aug. 8, 2025.

3. Grafana Labs, "*Grafana Documentation.*" . Available: `https://grafana.com/docs/`. Accessed: Aug. 8, 2025.

4. Kepler Project, "*Kepler GitHub Repository.*". Available: `https://github.com/sustainable-computing-io/kepler`. Accessed: Aug. 8, 2025.

5. LoadWatcher Maintainers, "*LoadWatcher GitHub Repository.*" . Available: `https://github.com/sustainable-computing-io/loadwatcher`. Accessed: Aug. 8, 2025.

6. PEAKS Developers, "*custom scheduler implemented GitHub Repository.*". Available: `https://github.com/Ursvishnnu/custom-scheduler`. Accessed: Aug. 8, 2025.

7. C. I. King, "*stress-ng GitHub Repository.*" [. Available: `https://github.com/ColinIanKing/stress-ng`. Accessed: Aug. 8, 2025.