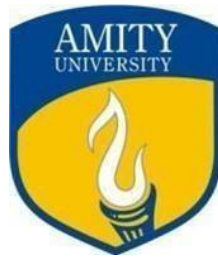


A Project Report
on
Analyzing Autoscaling Techniques Using Kubernetes

Submitted to:



Amity University Uttar Pradesh

In partial fulfillment of the requirements for the award of the degree of
Bachelor of Technology
Computer Science & Engineering

By

Bhavy Kumar Rajput, Kotik Sharma And Udit Rana

Under the guidance of
Dr. Nancy Gulati

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
AMITY SCHOOL OF ENGINEERING & TECHNOLOGY
AMITY UNIVERSITY UTTAR PRADESH**

DECLARATION

We, Bhavy Kumar Rajput, Kotik Sharma and Udit Rana, students of B.Tech (CSE) hereby declare that the project titled “**Analysing Different Autoscaling Techniques Using** Kubernetes” which is submitted by us to Department of Computer Science and Engineering, Amity School of Engineering and Technology, Amity University Uttar Pradesh, Noida, in partial fulfillment of requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering, has not been previously formed the basis for the award of any degree, diploma or other similar title or recognition. We hereby declare that we have gone through project guidelines including policy on health and safety, policy on plagiarism, etc.

Place: Noida



Date: 18/05/2023

Bhavy Kumar Rajput

Kotik Sharma

Udit Rana

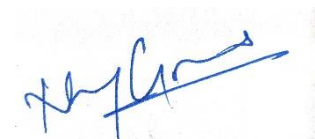
CERTIFICATE

On the basis of the declaration submitted by Mr. Bhavy Kumar Rajput, Mr. Kotik Sharma & Mr. Udit Rana ,students of B.Tech. Computer Science & Engineering, I hereby certify that the project titled “**To analyse different autoscaling techniques using Kubernetes**” which is submitted to the Department of Computer Science & Engineering, Amity School of Engineering & Technology, Amity University, Noida, Uttar Pradesh, in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Science & Engineering is an original contribution with existing knowledge and faithful record of work carried out by them under my guidance & supervision.

To the best of my knowledge this work has not been submitted in part or full for any degree or diploma to this university or elsewhere.

Place: Noida

Date: 18/05/2023



Dr. Nancy Gulati

(Assistant Professor-I)

Department of Computer Science and Engineering,
Amity School of Engineering and Technology, Amity
University Uttar Pradesh, Noida

CONSENT FORM

This is to certify that we, Bhavy Kumar Rajput, Kotik Sharma and Udit Rana, students of B.Tech. (CSE) of (2019-23) batch presently in VIII Semester at Amity School of Engineering and Technology, Department of Computer Science & Engineering, Amity University Uttar Pradesh, give my consent to include all my personal details (i.e. Name, Enrollment ID, etc.) for all accreditation purposes.

Place: Noida

Date:

Signature of the Students:

Three handwritten signatures in blue ink. The first signature is 'Bhavy' with a horizontal line underneath. The second signature is 'Kotik' with 'Sharma' written below it. The third signature is 'Udit' with a horizontal line underneath.

Enrollment Numbers:

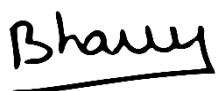
Bhavy Kumar Rajput - A2345919033

Kotik Sharma- A2345919025

Udit Rana- A2345919053

ACKNOWLEDGEMENT

The fulfillment that goes with that the productive completion of any errand would be divided without the indication of people whose ceaseless interest made it conceivable, whose steady heading and bolster crown all endeavors with victory. We would like to thank Prof Dr. Sanjeev Thakur, Head of Department-CSE, and Amity University for giving us the opportunity to embrace this extension. We would like to thank our faculty guide Dr. Nancy Gulati who is the greatest constraint behind my effective completion of the venture. She has been continuously there to clarify any inquiry of ours and moreover guided us inside the correct heading regarding the wander. Without her offer of help and inspiration, we would not have been able to add up to the amplify. As well we would like to thank my batch mates who guided us, made a difference in us and gave us contemplations and motivation at each step.



Bhavy Kumar Rajput
A2345919033



Kotik Sharma
A2345919025



Udit Rana
A2345919053

ABSTRACT

This research paper delves into the investigation of autoscaling techniques within the Kubernetes framework, focusing on their application in diverse resource and workload environments. The primary objective is to devise efficient and error-free strategies for implementing each autoscaling technique while determining the most suitable approach for specific architectural setups. The study encompasses an examination of Horizontal Pod Autoscaler (HPA), Vertical Pod Autoscaler (VPA), and Cluster Autoscaler. Different workload types, encompassing a range of images and metrics, are briefly explored and implemented in conjunction with these autoscaling techniques. The experimentation includes diverse scenarios involving varying numbers of nodes, CPU utilization levels, and memory usage patterns. Through an iterative refinement process across multiple workload scenarios, the research endeavors to minimize errors in the implementation. The outcomes of the research will provide specific insights and results for each autoscaling technique based on substantial workload implementations within these scenarios. Furthermore, a comprehensive comparative analysis will be conducted, evaluating all the autoscaling approaches, culminating in the formulation of an autoscaling strategy.

INDEX

SNO.	TOPIC	PAGE NO.
1.	Declaration	
2.	Certificate	
3.	Consent Form	
4.	Acknowledgement	
5.	Abstract	
6.	Introduction	
7.	Kubernetes	
8.	Horizontal Pod Autoscaling	
9.	Requirement Analysis	
10.	Lab Environment	
11.	Methodology	
12.	Implementation	
13.	Walkthrough	
14.	Result Analysis	
15.	Operator	
16.	Conclusion	
17.	Future Scope	
18.	Reference	

Introduction

What are Microservices?

Microservices, or Microservice architecture, refers to an architectural approach that organizes an application into a collection of services. These services are designed to be loosely interconnected, can be deployed independently, and are straightforward to oversee and control.

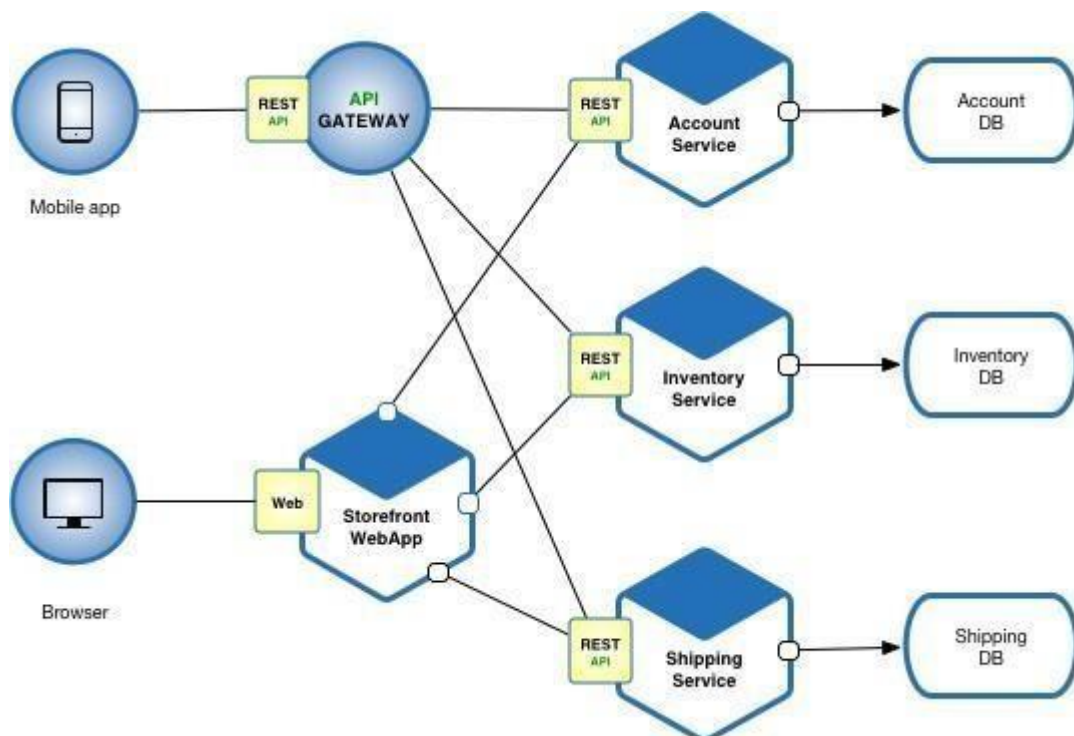


Fig. 1: Microservice Architecture

Although this architecture seems to be more complex in comparison to other architectures like Monolith due to its distributive nature, one of the greatest benefits it offers is Scalability which is difficult in Monolith Architecture.

In this architecture, all the microservices are written in modern programming language according to their types and functions. This provides a great amount of flexibility when matching microservice with specific hardware when required. There is a process which enables transformation of Monolith based applications to Microservice based applications. This process is called refactoring. Some applications which used refactoring with the help of Kubernetes are, AppDirect, box, Crowdfire, GolfNow, Wikimedia, Spotify, Squarespace, and Pinterest.

Kubernetes

Kubernetes is also known as K8s. It is an open-source system that is highly inspired by the Google Borg system. It is an open-source project written in the Golang language and automates deployment, scaling, and management of containerized applications. Kubernetes was started by Google and, with its v1.0 release in July 2015, Google donated it to the CNCF (Cloud Native Computing Foundation).

Some of the features provided by Kubernetes are:

1. **Automated bin packing** – It automates scheduling of containers depending upon their resource needs and the constraints, to further maximize utilization without giving up on availability.
2. **Self-healing** – It reschedules and replaces the containers from the failed nodes, kills, and then reboots unresponsive containers to the health checks, based on the existing rules. This feature is also used to prevent traffic from being routed to containers which are unresponsive.
3. **Horizontal Scaling** – In this feature the apps are scaled (automatically/manually) based on CPU metrics utilization or custom metrics utilization. The pods responsible for this are called HPA (Horizontal Pod Autoscaler).
4. **Load Balancing and Service Discovery** – Containers have their own Internet Protocol addresses in K8s, while it also allocates a DNS (Domain Name System) to a few containers to help them in load balancing requests across these containers.
5. **Automated rollouts and rollback** – It rolls-out and rolls-back app configuration changes and updates, while also constantly monitoring the app's health to avoid any downtime in it. This is a very important feature in organization as that require fast updates without much downtime.
6. **Secret and configuration management** – In K8s, ConfigMaps manage sensitive data and the configuration details for the app separately from their container image, to prevent a rebuild of their source image. Secrets consists of sensitive/confidential data passed to the app without exposing the sensitive content to the stack configuration.
7. **Storage orchestration** – SDS (Software-defined storage) solutions are automatically mounted to the containers from the local storage, the external cloud providers, the distributed storage, or the network storage systems.

8. **Batch execution** – K8s supports long-running jobs, batch execution, and automatically replaces failed containers.

In addition to these features, K8s is also extensible and portable. K8s can be deployed in many environments such as local or remote Virtual Machines, or in public/private/hybrid/multi-cloud setups. It supports and is supported by many third-party open-source tools which enhance K8s' capabilities and provide a feature-rich experience to its users.

Also, K8s' architecture is modular and pluggable. Not only that it orchestrates modular, decoupled microservices type applications, but also its architecture follows decoupled microservices patterns. Kubernetes' functionality can be extended by writing custom resources, operators, custom APIs, scheduling rules or plugins.

Some companies using Kubernetes are Nokia, Huawei, Pearson, Wikimedia, eBay, IBM and many more.

Kubernetes: Basic Units

1. Fundamental unit of computing in a Kubernetes is a POD. Which describes an application.
2. Pod can consist of one or more containers tightly coupled.
3. Within a pod container can talk using port numbers.
4. Controller Manager:
 - a. Take care of running the cluster
 - b. Several functions like logic, interacting with scheduler, etc
5. Scheduler: Schedules Pods on nodes based on the resources (CPU, memory, up/down).
6. API server: Mechanism of interaction with Kubernetes cluster.
7. Kubelet:
 - a. Run the pods containers.
 - b. Report status of nodes and pods to API Server
 - c. Run container probes
 - d. Retrieve container metrics
8. Kube-proxy facilitates networking services

9. Etcd: Saves
 - a. Configuration data
 - b. Information about state of the cluster

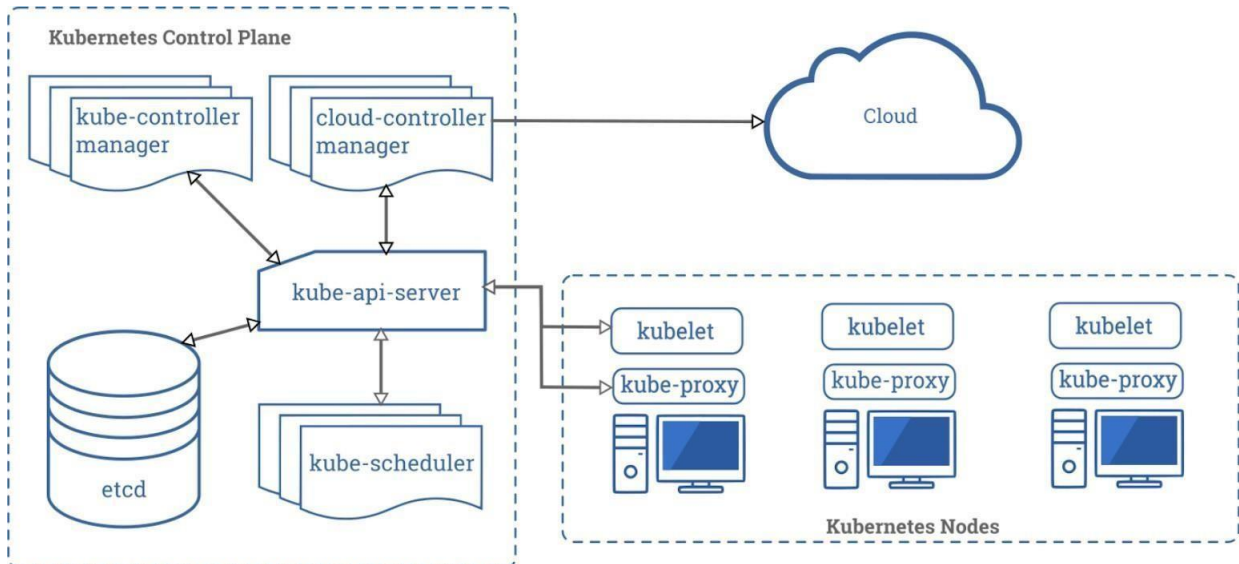


Fig. 2: Kubernetes Architecture

Kubernetes: Objects

Objects: Persistent entities to represent the desired state/ “SPEC” of the cluster. Kubernetes works to consistently be in that desired state by monitoring “STATE” of the components in the figure. This is known as feedback loop.



Fig. 3: Different Levels of Abstraction in Kubernetes

Kubernetes: Deployment

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

Fig. 4: Basic nginx Deployment

Horizontal Pod Autoscaler

Horizontal scaling is a native capability provided by Kubernetes. It involves adjusting the number of replicas for an application, either increasing or decreasing them as needed. The Horizontal Pod Autoscaler (HPA) is a controller within the Kubernetes controller manager that performs this function. It monitors a specified metric related to an application and dynamically adjusts the number of replicas to effectively meet the demand. The HPA can scale various resources, such as Deployment, Stateful Set, Replica Set, and Replication Controller

Horizontal Pod Autoscaler

Horizontal scaling is a native capability provided by Kubernetes. It involves adjusting the number of replicas for an application, either increasing or decreasing them as needed. The Horizontal Pod Autoscaler (HPA) is a controller within the Kubernetes controller manager that performs this function. It monitors a specified metric related to an application and dynamically adjusts the number of replicas to effectively meet the demand. The HPA can scale various resources, such as Deployment, StatefulSet, ReplicaSet, and Replication Controller.

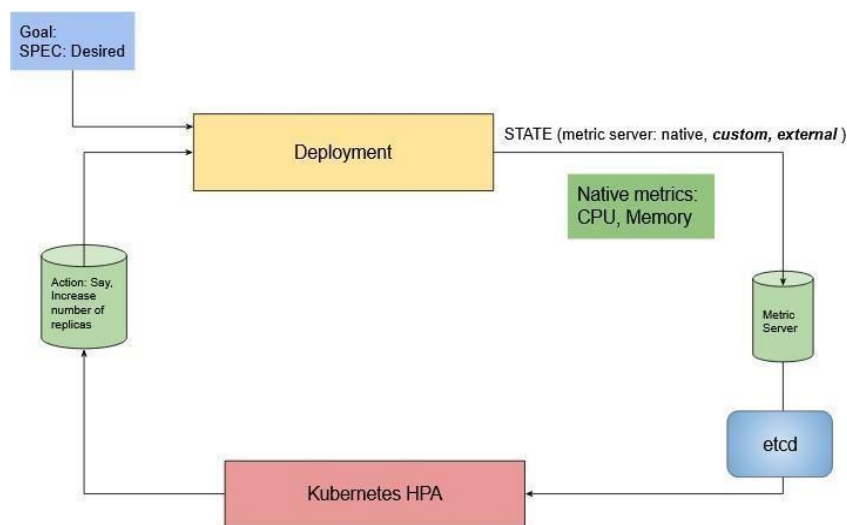


Fig. 5 (A): Default HPA

Cluster Autoscaling

The Cluster Autoscaler is a notable feature in Kubernetes that automates the scaling of the underlying cluster infrastructure to meet the demands of the workload. Unlike the Horizontal Pod Autoscaler, which focuses on scaling individual pods, the Cluster Autoscaler dynamically adjusts the number of nodes in the cluster.

When the workload experiences an increase and requires additional resources, the Cluster Autoscaler provisions new nodes to accommodate the growing demand. This results in more pods being scheduled and distributed across the expanded infrastructure. Conversely, if the workload decreases and the existing nodes are underutilized, the Cluster Autoscaler removes unnecessary nodes to optimize resource allocation and improve cost-efficiency.

It's important to note that the Cluster Autoscaler operates at the cluster level and is not limited to specific workloads or objects. It can scale any workload that can be scheduled on the cluster, including Deployments, Stateful Sets, and Daemon Sets. This adaptability allows the Cluster Autoscaler to cater to various workload requirements and effectively manage resources.

Acting as a controller within the Kubernetes control plane, the Cluster Autoscaler is configured to monitor specific metrics like CPU utilization or custom-defined metrics. These observed metrics guide the Cluster Autoscaler in continuously adjusting the desired number of nodes in the cluster. By leveraging the resources provided by the Kubernetes API, the Cluster Autoscaler ensures that the cluster's capacity aligns with the workload demands, resulting in optimal performance and scalability.

To summarize, the Cluster Autoscaler is an essential component in Kubernetes for efficient cluster resource management. It automates the scaling of the cluster infrastructure based on workload demands, ensuring optimal resource allocation and maximizing the utilization of computing resources.

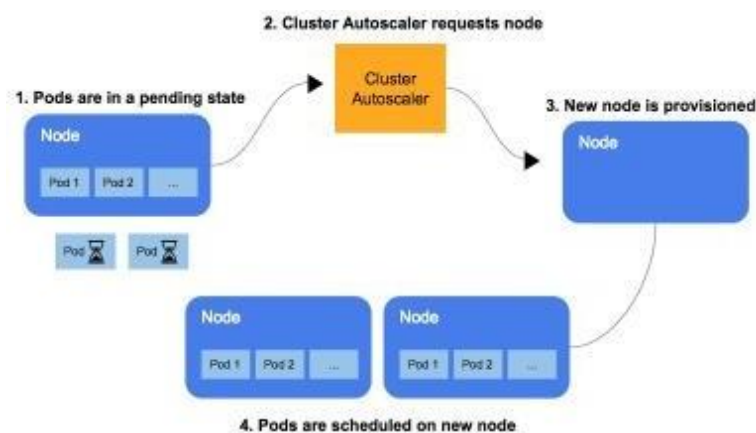


Fig. 5 (B): Default CA

Requirement Analysis

Software Requirement

Ubuntu 18.04 Server iso file to install it in a Virtual Machine Net stat Bridge installed in the local host machine

Creating two VMs (one master node, one worker node)

In both the VMs, install docker-ce (18.06.1~ce~3-0~ubuntu), kubernetes-cni

Also install kubeadm (1.20.5-00), kubelet (1.20.5-00), and kubectrl (1.20.5-00) such that their versions match, e.g- *sudo apt-get install -y kubelet=1.20.5-00 kubeadm=1.20.5-00 kubectrl=1.20.5-00*

Hardware Requirement

System with 8GB RAM (At least, to create 2 VMs, one worker and one master node of at least 2GB RAM each).

Lab Environment

1. **Ubuntu 18.04 VM (inside the student VM, NAT network)**
2. **Install docker-ce, kubeadm, kubelet, kubectrl on student VM and new VM inside student VM**
 - a. `sudo apt install lsb-core`
 - b. `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -`
 - c. `sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"`
 - d. `curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -`
 - e. `cat << EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list`
 - f. `deb https://apt.kubernetes.io/ kubernetes-xenial main`
 - g. EOF
 - h. `sudo apt update`

- i. `sudo apt-get install -y docker-ce=18.06.1~ce~3-0~ubuntu kubernetes-cni kubelet=1.20.5-00 kubeadm=1.20.5-00 kubectl=1.20.5-00`
- j. `sudo apt-mark hold kubelet kubeadm kubectl`
- k. `sudo systemctl daemon-reload`
- l. `sudo systemctl restart kubelet`

3. On the student VM Initialize cluster: Acts like Kubernetes Control Plane

- a. `#sudo swapoff -a` #if required
- b. `sudo kubeadm init --pod-network-cidr=10.244.0.0/16` # Note the last output: Tokens and certificates
- c. `mkdir -p $HOME/.kube`
- d. `sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config`
- e. `sudo chown $(id -u):$(id -g) $HOME/.kube/config`
- f. `kubectl apply -f`
<https://raw.githubusercontent.com/coreos/flannel/2140ac876ef134e0ed5af15c65e414cf26827915/Documentation/kube-flannel.yml>

4. On worker node

- a. `sudo kubeadm join <IP:PORT> --token <token> --discovery-token-ca-cert-hash sha256:<checksum>`

5. To reset the configuration on controller and worker node

- a. `sudo kubeadm reset`

Methodology

In this section, a stepwise procedure for the implementation of Kubernetes has been discussed. The following points briefly describe the methodology used.

Setting up Linux as the Base Operating System

In this project, we will be using Linux as the base operating system, to act as a node which is the basic entry point resource in Kubernetes for most administrative tasks and commands.

Configuring Kubernetes on a Virtual Machine

These nodes (one worker and one master) will be used on a Virtual Machine, the Linux version would be Ubuntu 18.04 Server. In this we'll be setting up Kubernetes using kubeadm, kubelet, and kubectl which are all Kubernetes tools.

Utilizing Different Types of Metrics for Autoscaling

The autoscalers will be used in different types of metrics, some default, some custom created, while some borrowed from web sources.

Configuring Resources, Versions, and Architectures for Autoscalers

The resources, versions and architectures will need to be configured for different autoscalers.

Handling Versions with Care for Data Integrity

The versions specifically will be needed to handle very delicately, since any mistake could result in corruption of some data or the whole VM, needing to be created anew.

Visualising and comparing the efficiency of Autoscalers

After the efficiency of different autoscalers has been recorded, graphical representation depicting their comparison would be shown along with some strategies that we will create after learning from this.

Implementation

Horizontal Pod Autoscaler

The Horizontal Pod Autoscaler in Kubernetes automates the process of scaling a workload, such as a Deployment or Stateful Set, to match the current demand. It achieves this by automatically adjusting the number of pods based on observed metrics.

Horizontal scaling involves deploying additional pods in response to increasing load, as opposed to vertical scaling, which would involve providing more resources to the existing pods in the workload. If the load decreases and the number of pods is still above the minimum configured value, the Horizontal Pod Autoscaler will instruct the workload to scale back down.

It's important to note that the Horizontal Pod Autoscaler cannot be applied to objects that cannot be scaled, such as a DaemonSet. The Horizontal Pod Autoscaler functions both as a controller and a Kubernetes API resource. The behavior of the controller is determined by the associated resources. Within the Kubernetes control plane, the horizontal pod autoscaling controller regularly adjusts the desired scale of its designated target based on observed metrics, such as average CPU utilization, average memory utilization, or other custom metrics that have been specified.

Walkthrough

1. Installing tools

1) Kubectl

```
#!/bin/bash
# Download kubectl from the dynamically generated URL
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

# Download kubectl directly from a specific version URL
curl -LO https://dl.k8s.io/release/v1.24.0/bin/linux/amd64/kubectl

# Download the SHA256 checksum file for kubectl
curl -LO "https://dl.k8s.io/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"

# Verify the integrity of the downloaded kubectl binary
echo "$(cat kubectl.sha256) kubectl" | sha256sum -check

# Install kubectl with appropriate permissions
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

# Check the client version of kubectl
/usr/local/bin/kubectl version --client
```

2) Kind

```
curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.12.0/kind-linux-amd64
chmod +x ./kind
mv ./kind /some-dir-in-your-PATH/kind
kind create cluster
kind create cluster --name kind-2
kind get clusters
kind
kind-2
kubectl cluster-info --context kind-kind
kubectl cluster-info --context kind-kind-2
kind delete cluster
```

3) Minikube

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube

minikube start
kubectl get po -A
minikube dashboard
kubectl create deployment hello-minikube -- image=k8s.gcr.io/echoserver:1.4

kubectl expose deployment hello-minikube --type=NodePort --port=8080

kubectl get services hello-minikube
minikube service hello-minikube
kubectl port-forward service/hello-minikube 7080:8080

kubectl create deployment balanced -- image=k8s.gcr.io/echoserver:1.4

kubectl expose deployment balanced --type=LoadBalancer -- port=8080

minikube tunnel

kubectl get services balanced
minikube pause
minikube unpause
minikube stop
minikube config set memory 16384
minikube addons list
minikube start -p aged --kubernetes-version=v1.16.1
minikube delete --all
```

2. Administer a cluster

1) Administration with kubeadm

i. Certificate management with kubeadm

Command: `kubeadm certs check-expiration`

Output:

CERTIFICATE	EXPIRES	RESIDUAL TIME	CERTIFICATE AUTHORITY	EXTERNALLY MANAGED
admin.conf	Dec 30, 2020 23:36 UTC	364d		no
apiserver	Dec 30, 2020 23:36 UTC	364d	ca	no
apiserver-etcd-client	Dec 30, 2020 23:36 UTC	364d	etcd-ca	no
apiserver-kubelet-client	Dec 30, 2020 23:36 UTC	364d	ca	no
controller-manager.conf	Dec 30, 2020 23:36 UTC	364d		no
etcd-healthcheck-client	Dec 30, 2020 23:36 UTC	364d	etcd-ca	no
etcd-peer	Dec 30, 2020 23:36 UTC	364d	etcd-ca	no
etcd-server	Dec 30, 2020 23:36 UTC	364d	etcd-ca	no
front-proxy-client	Dec 30, 2020 23:36 UTC	364d	front-proxy-ca	no
scheduler.conf	Dec 30, 2020 23:36 UTC	364d		no

CERTIFICATE AUTHORITY	EXPIRES	RESIDUAL TIME	EXTERNALLY MANAGED
ca	Dec 28, 2029 23:36 UTC	9y	no
etcd-ca	Dec 28, 2029 23:36 UTC	9y	no
front-proxy-ca	Dec 28, 2029 23:36 UTC	9y	no

This command is not applicable for external CA. By default, certificates are generated by kubeadm. These certificates expire after 1 year. These certificates can be found with the help of certificates Dir field, or the `–cert-dir` flag of kubeadm’s Cluster Configuration.

ii. Configuring a cgroup driver

According to the documentation of Container runtimes, it is advised to use the `systemd` driver instead of the `cgroupfs` driver for kubeadm based setups. This recommendation is made because kubeadm manages the kubelet as a `systemd` service.

When using `kubeadm init`, we have the option to provide a Kubelet Configuration structure. This structure allows us to specify the cgroup driver for the kubelet through the `cgroupDriver` field.

Here is a basic example of explicitly configuring the field:

```
# kubeadm-config.yaml
kind: ClusterConfiguration
apiVersion: kubeadm.k8s.io/v1beta3
kubernetesVersion: v1.21.0
---
kind: KubeletConfiguration
apiVersion: kubelet.config.k8s.io/v1beta1
cgroupDriver: systemd
```

Command: `kubeadm init --config kubeadm-config.yaml`

This command will open the kubelet-config ConfigMap in the kube-system namespace for editing. Within the ConfigMap, you can modify the existing `cgroupDriver` value to **cgroup Driver: systemd** if it already exists, or add the field if it is not present. Save the changes once you have made the necessary modifications.

iii. Reconfiguring a kubeadm cluster

Automated reconfiguration of components deployed on managed nodes is not supported by kubeadm. To automate this process, a custom operator can be utilized. However, for manually modifying the components' configuration, cluster objects and associated files on disk need to be edited.

kubeadm writes cluster-wide component configuration options in ConfigMaps and other objects, which require manual editing. The `kubectl edit` command is used to accomplish this task. When executed, it opens a text editor where the object can be modified and saved.

To specify the location of the kubeconfig file consumed by `kubectl` and the preferred text editor, the environment variables `KUBECONFIG` and `KUBE_EDITOR` can be utilized.

```
export KUBECONFIG=/etc/kubernetes/admin.conf
export KUBE_EDITOR=nano
kubectl edit <parameters>
```

When creating or upgrading a cluster, kubeadm stores its Cluster Configuration in a ConfigMap named "kubeadm-config" within the "kube-system" namespace. To modify a specific option within the Cluster Configuration, we can edit the corresponding Config Map. Command:

```
kubectrl edit cm -n kube-system kubeadm-config
```

Reflecting ClusterConfiguration changes on control plane nodes: To write new certificates we can use:

```
kubeadm init phase certs <component-name> --config <config-file>
```

To write new manifest files in /etc./ Kubernetes/ manifests:

```
kubeadm init phase control-plane <component-name> --config <config-file>
```

Applying kubelet configuration changes:

We can edit the ConfigMap with this command:

```
kubectrl edit cm -n kube-system kubelet-config
```

Command:

The configuration is located under the data.kubelet key.

To apply configuration changes to kube-proxy, you can modify the ConfigMap using the following command:

Command: `kubectrl edit cm -n kube-system kube-proxy`

After updating the kube-proxy ConfigMap, you need to restart all kube-proxy Pods.

To obtain the names of the Pods, you can use the following command:

```
kubectrl get po -n kube-system | grep kube-proxy
```

Delete a Pod with:

```
kubectrl delete po -n kube-system <pod-name>
```

Command:

To apply configuration changes to Core DNS, you can modify the Deployment and Service objects. These objects control the deployment and service discovery of Core DNS.

To update any of the Core DNS settings, you can edit the corresponding Deployment and Service objects using the appropriate `kubectl edit` command.

Commands:

```
kubectl edit deployment -n kube-system coredns  
  
kubectl edit service -n kube-system kube-dns
```

To delete core DNS pods:

Command:

```
kubectl get po -n kube-system | grep coredns
```

Delete a

Pod with:

```
kubectl delete po -n kube-system <pod-name>
```

Command:

Persisting the reconfiguration:

If we want to modify the contents of a Node object in Kubernetes, we can use the following command: `kubectl edit no <node-name>`. This allows you to directly edit the Node object and make the desired changes.

However, it's important to note that during a kubeadm upgrade process, the contents of the Node object might be overwritten, potentially undoing any modifications made manually.

To ensure that your modifications to the Node object persist after an upgrade, you can prepare a kubectl patch. This involves creating a patch file that contains the specific changes you want to make to the Node object. Once the upgrade is complete, you can apply the patch using the `kubectl patch` command to reapply your modifications to the Node object. This ensures that your desired configuration changes are maintained even after the upgrade process.

```
kubectl patch node <node-name> --patch-file <patch-file>
```

iv. Upgrading kubeadm clusters

Determine which version to upgrade to:

```
sudo apt update  
  
apt-cache madison kubeadm
```


Upgrading control plane nodes:

```
sudo apt-mark unhold kubeadm && \  
sudo apt-get update && \  
sudo apt-get install -y kubeadm=1.24.x-00 && \  
sudo apt-mark hold kubeadm
```

Verify that the download works and has the expected version:

kubeadm version

Verify the upgrade plan:

kubeadm upgrade plan

Example:

Command:

```
sudo kubeadm upgrade apply v1.24.x
```

Output:

```
[upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.24.x". Enjoy!  
[upgrade/kubelet] Now that your control plane is upgraded, please proceed with upgrading your kubelets if you haven't already done so.
```

Drain the node:

```
kubectyl drain <node-to-drain> --ignore-daemonsets
```

Upgrade kubelet and kubectyl:

```
apt-mark unhold kubelet kubectyl && \  
apt-get update && \  
apt-get install -y kubelet=1.24.x-00 kubectyl=1.24.x-00 && \  
apt-mark hold kubelet kubectyl
```

Restart the kubelet:

```
sudo systemctl daemon-reload
```



```
sudo systemctl restart kubelet
```

Uncordon the node:

```
kubectl uncordon <node-to-drain>
```

v. Adding Windows nodes

Configuring Flannel:

Prepare Kubernetes control plane for Flannel:

```
sudo sysctl net.bridge.bridge-nf-call-iptables=1
```

Download & configure Flannel for Linux:

wget

<https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml>

```
net-conf.json: |
  {
    "Network": "10.244.0.0/16",
    "Backend": {
      "Type": "vxlan",
      "VNI": 4096,
      "Port": 4789
    }
  }
```

Apply the Flannel manifest and validate:

```
kubectl apply -f kube-flannel.yml
kubectl get pods -n kube-system
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
...					
kube-system	kube-flannel-ds-54954	1/1	Running	0	1m

2) Certificates

Easysrsa can manually generate certificates for our cluster. Download, unpack, and initialize the patched version of easysrsa3:

```
curl -LO https://storage.googleapis.com/kubernetes-release/easy-rsa/easy-rsa3
tar xzf easy-rsa.tar.gz
cd easy-rsa-master/easysrsa3
./easysrsa init-pki
```

Generate a new certificate authority (CA). --batch sets automatic mode; --req-cn specifies the Common Name (CN) for the CA's new root certificate:

```
./easysrsa --batch "--req-cn=${MASTER_IP}@`date +%s`" build-ca nopass
```

Create a server certificate and key. Use the --subject-alt-name argument to define the allowed IPs and DNS names that can be used to access the API server. The MASTER_CLUSTER_IP typically corresponds to the initial IP address in the service CIDR range specified as the --service-cluster-ip-range argument for both the API server and the controller manager component. The --days argument determines the validity period of the certificate in days. In the provided example, it assumes the default DNS domain name is cluster.local.

```
./easysrsa --subject-alt-name="IP:${MASTER_IP}," \
"IP:${MASTER_CLUSTER_IP}," \
"DNS:kubernetes," \
"DNS:kubernetes.default," \
"DNS:kubernetes.default.svc," \
"DNS:kubernetes.default.svc.cluster," \
"DNS:kubernetes.default.svc.cluster.local" \
--days=10000 \
build-server-full server nopass
```

Fill pki/ca.crt, pki/issued/server.crt, and pki/private/server.key to our directory and add parameters into API server.

```
--client-ca-file=/yourdirectory/ca.crt
--tls-cert-file=/yourdirectory/server.crt
--tls-private-key-file=/yourdirectory/server.key
```

3) Manage Memory, CPU, and API Resources

i. Configure Default Memory Requests and Limits for a Namespace

Create a namespace so that the resources created further are isolated from the rest of our cluster:

```
kubectl create namespace default-mem-example
```

Create a LimitRange and a Pod:

```
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-limit-range
spec:
  limits:
  - default:
      memory: 512Mi
    defaultRequest:
      memory: 256Mi
    type: Container
---
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
  - name: example-container
    image: nginx
```

Create Limit Range:

```
Create Limit Range:  
kubectl apply -f https://k8s.io/examples/admin/resource/memory-defaults.yaml --namespace=default-mem-example
```

Create new Pod :

```
kubectl get pod default-mem-demo --output=yaml -- namespace=default-mem-example
```

Output:

```
containers:  
- image: nginx  
  imagePullPolicy: Always  
  name: default-mem-demo-ctr  
  resources:  
    limits:  
      memory: 512Mi  
    requests:  
      memory: 256Mi
```

Delete the pod:

```
kubectl delete pod default-mem-demo --namespace=default-mem-example
```

- ii. To configure default CPU requests and limits for a namespace in Kubernetes

Create a LimitRange and a Pod:

```
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-limit-range
spec:
  limits:
    - default:
        cpu: "1"
      defaultRequest:
        cpu: "0.5"
      type: Container
---
apiVersion: v1
kind: Pod
metadata:
  name: cpu-limit-pod
spec:
  containers:
    - name: cpu-limit-container
      image: nginx
```

To create a `LimitRange` in the `default-cpu-example` namespace, you can use the following command:

.Manifest:

```
apiVersion: v1
kind: Pod
metadata:
  name: default-cpu-demo
spec:
  containers:
    - name: default-cpu-demo-ctr
      image: nginx
```

Output:

```
containers:
- image: nginx
  imagePullPolicy: Always
  name: default-cpu-demo-ctr
  resources:
    limits:
      cpu: "1"
    requests:
      cpu: 500m
```

iii. Configures Minimum and Maximum Memory Constraints for a Namespace

Create LimitRange and Pod:

```
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-min-max-demo-lr
spec:
  limits:
  - max:
      memory: 1Gi
    min:
      memory: 500Mi
    type: Container
---
apiVersion: v1
kind: Pod
metadata:
  name: mem-min-max-demo-pod
spec:
  containers:
  - name: mem-min-max-demo-container
    image: nginx
```

Output:

```
limits:
- default:
    memory: 1Gi
  defaultRequest:
    memory: 1Gi
  max:
    memory: 1Gi
  min:
    memory: 500Mi
  type: Container
```

iv. Configure Minimum and Maximum CPU Constraints for a Namespace

Create a LimitRange and a Pod:

```
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-min-max-demo-lr
spec:
  limits:
  - max:
      cpu: "800m"
    min:
      cpu: "200m"
    type: Container
```

LimitRange:

```
kubectl apply -f
https://k8s.io/examples/admin/resource/cpu-
constraints.yaml --namespace=constraints-cpu-example
```

Detailed information:

```
kubectl get limitrange cpu-min-max-demo-lr --output=yaml -
--namespace=constraints-cpu-example
```

Output:

```
limits:
- default:
  cpu: 800m
  defaultRequest:
    cpu: 800m
  max:
    cpu: 800m
  min:
    cpu: 200m
  type: Container
```

v. Configure Memory and CPU Quotas for a Namespace

Create a Resource Quota:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-demo
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```


Create a Pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: quota-mem-cpu-demo
spec:
  containers:
  - name: quota-mem-cpu-demo-ctr
    image: nginx
    resources:
      limits:
        memory: "800Mi"
        cpu: "800m"
      requests:
        memory: "600Mi"
        cpu: "400m"
```

Create the pod:

```
kubectl apply -f https://k8s.io/examples/admin/resource/quota-mem-cpu- pod.yaml --namespace=quota-mem-
-cpu-example
```

Verifying that pod is running healthy or not .

```
kubectl get pod quota-mem-cpu-demo --namespace=quota-mem- cpu-example
```

Viewing detailed info. About resource Quota:

```
kubectl get resourcequota mem-cpu-demo --namespace=quota- mem-cpu-example --output=yaml|
```

Output:

```
status:
  hard:
    limits.cpu: "2"
    limits.memory: 2Gi
    requests.cpu: "1"
    requests.memory: 1Gi
  used:
    limits.cpu: 800m
    limits.memory: 800Mi
    requests.cpu: 400m
    requests.memory: 600Mi
```

vi. Configure a Pod Quota for a Namespace

Resource Quota:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: pod-demo
spec:
  hard:
    pods: "2"
```

Output:

```
spec:
  hard:
    pods: "2"
status:
  hard:
    pods: "2"
  used:
    pods: "0"
```

Manifest of Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pod-quota-demo
spec:
  selector:
    matchLabels:
      purpose: quota-demo
  replicas: 3
  template:
    metadata:
      labels:
        purpose: quota-demo
    spec:
      containers:
        - name: pod-quota-demo
          image: nginx
```

Detailed information:

```
kubectl get deployment pod-quota-demo --namespace=quota-  
pod-example --output=yaml
```

Output:

```
spec:  
  ...  
  replicas: 3  
  ...  
status:  
  availableReplicas: 2  
  ...  
lastUpdateTime: 2021-04-02T20:57:05Z  
message: 'unable to create pods: pods "pod-quota-demo-1650323038-" is forbidden:  
  exceeded quota: pod-demo, requested: pods=1, used: pods=2, limited: pods=2'
```

4) Access Clusters using Kubernetes API

i. Accessing the Kubernetes API

A. Accessing for the first time with kubectl

To interact with a Kubernetes cluster, it is essential to have information about the cluster's location and valid credentials. You can verify the cluster's location and credentials known to `kubectl` by using the following command:

```
kubectl config view
```

B. Directly accessing the REST API

a) Using kubectl proxy

Commands:

```
kubectl proxy --port=8080 &
```

```
curl http://localhost:8080/api/
```

Output:

```
{
  "versions": [
    "v1"
  ],
  "serverAddressByClientCIDRs": [
    {
      "clientCIDR": "0.0.0.0/0",
      "serverAddress": "10.0.1.149:443"
    }
  ]
}
```

b) **Without kubectl proxy**

Using grep/cut approach:

```
import subprocess
import json

def get_cluster_server(cluster_name):
    output = subprocess.check_output(['kubectl', 'config', 'view', '-o', 'json'])
    config = json.loads(output)
    clusters = config['clusters']
    for cluster in clusters:
        if cluster['name'] == cluster_name:
            return cluster['cluster']['server']
    return None

def get_token():
    subprocess.run(['kubectl', 'apply', '-f', '-'], input="""
apiVersion: v1
kind: Secret
metadata:
  name: default-token
  annotations:
    kubernetes.io/service-account.name: default
type: kubernetes.io/service-account-token
""", encoding='utf-8')

    while True:
        output = subprocess.check_output(['kubectl', 'describe', 'secret', 'default-token'])
        if 'token:' in output:
            return output.split('token:')[1].strip()

def main():
    cluster_name = "some_server_name"
    api_server = get_cluster_server(cluster_name)
    token = get_token()
    if api_server and token:
        subprocess.run(['curl', '-X', 'GET', f'{api_server}/api', '--header', f'Authorization: Bearer {token}', '--insecure'])
```

```

token = get_token()
if api_server and token:
    subprocess.run(['curl', '-X', 'GET', f'{api_server}/api', '--header', f'Authorization: Bearer {token}', '--insecure'])

if __name__ == "__main__":
    main()

```

Output:

```

{
  "kind": "APIVersions",
  "versions": [
    "v1"
  ],
  "serverAddressByClientCIDRs": [
    {
      "clientCIDR": "0.0.0.0/0",
      "serverAddress": "10.0.1.149:443"
    }
  ]
}

```

C. Access to the API

a) Go client

```

from kubernetes import client, config

def main():
    config.load_kube_config()
    api = client.CoreV1Api()
    namespaces = api.list_namespace().items
    num_pods = 0

    for ns in namespaces:
        pods = api.list_namespaced_pod(namespace=ns.metadata.name).items
        num_pods += len(pods)

    print(f"There are {num_pods} pods in the cluster")

if __name__ == "__main__":
    main()

```

b) Python client

```
from kubernetes import client, config

# Load the kubeconfig file
config.load_kube_config()

# Create an instance of the CoreV1Api
v1 = client.CoreV1Api()

# List pods with their IPs
print("Listing pods with their IPs:")
ret = v1.list_pod_for_all_namespaces(watch=False)
for pod in ret.items:
    print(f"{pod.status.pod_ip}\t{pod.metadata.namespace}\t{pod.metadata.name}")
```

c) Java client

Installation:

```
git clone --recursive https://github.com/kubernetes-
client/java
```

```
cd java mvn install
```

The java client can use the same kube config file as kube CLI and authenticate to API server :

```
import io.kubernetes.client.openapi.ApiClient;
import io.kubernetes.client.openapi.ApiException;
import io.kubernetes.client.openapi.Configuration;
import io.kubernetes.client.openapi.apis.CoreV1Api;
import io.kubernetes.client.openapi.models.V1Pod;
import io.kubernetes.client.openapi.models.V1PodList;
import io.kubernetes.client.util.Config;

public class KubeConfigFileClientExample {
    public static void main(String[] args) throws ApiException {
        ApiClient client = Config.defaultClient();
        Configuration.setDefaultApiClient(client);

        CoreV1Api api = new CoreV1Api();
        V1PodList list = api.listPodForAllNamespaces(null, null, null, null, null, null, null, null,
            null);

        System.out.println("Listing all pods:");
        for (V1Pod item : list.getItems()) {
            System.out.println(item.getMetadata().getName());
        }
    }
}
```

5) Advertise Extended Resources for a Node

To add a new extended resource to a Node in Kubernetes, you can use an HTTP PATCH request to the Kubernetes API server. Suppose we want to register four dongles on a Node as an example. Here is an example of a PATCH request that notifies the Kubernetes API server about these four dongle resources for the Node.

```
PATCH /api/v1/nodes/<your-node-name>/status HTTP/1.1 Accept: application/json
Content-Type: application/json-patch+json Host: k8s-master:8080
[
{
  "op": "add",
  "path": "/status/capacity/example.com~1dongle", "value": "4"
}
]
```

Start a proxy, so that we can easily send requests to the Kubernetes API server:

```
kubectl proxy
```

We can send the HTTP PATCH request. Replace <node-name> with the name of our Node:

```
curl --header "Content-Type: application/json-patch+json" \
  --request PATCH \
  --data ' [{"op": "add", "path": "/status/capacity/example.com~1dongle", "value": "4"}]' \
  http://localhost:8001/api/v1/nodes/<your-node-name>/status
```

The output shows that the Node has a capacity of 4 dongles:

```
"capacity": {
  "cpu": "2",
  "memory": "2049008Ki",
  "example.com/dongle": "4",
```

Describe the Node:

```
kubectl describe node <node-name>
```



```
Capacity:
  cpu: 2
  memory: 2049008Ki
  example.com/dongle: 4
```

a. Autoscale the DNS Service in a Cluster

i. Determine whether DNS horizontal autoscaling is already enabled

Output:

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
...				
dns-autoscaler	1/1	1	1	...
...				

ii. Get the name of your DNS Deployment

Output:

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
...				
coredns	2/2	2	2	...
...				

Our scale target is

Deployment/<your-deployment-name>

iii. Enable DNS horizontal autoscaling

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: kube-dns-autoscaler
  namespace: kube-system
---

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kube-dns-autoscaler-role
rules:
- apiGroups: [""]
  resources: ["nodes"]
  verbs: ["list", "watch"]
- apiGroups: [""]
  resources: ["replicationcontrollers/scale"]
  verbs: ["get", "update"]
- apiGroups: ["apps"]
  resources: ["deployments/scale", "replicasets/scale"]
  verbs: ["get", "update"]
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["get", "create"]
---

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kube-dns-autoscaler-binding
subjects:
- kind: ServiceAccount
  name: kube-dns-autoscaler
  namespace: kube-system
```

```

roleRef:
  kind: ClusterRole
  name: kube-dns-autoscaler-role
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kube-dns-autoscaler
  namespace: kube-system
  labels:
    app: kube-dns-autoscaler
spec:
  selector:
    matchLabels:
      app: kube-dns-autoscaler
  template:
    metadata:
      labels:
        app: kube-dns-autoscaler
    spec:
      serviceAccountName: kube-dns-autoscaler
      containers:
        - name: autoscaler
          image: k8s.gcr.io/cpa/cluster-proportional-autoscaler:1.8.4
          resources:
            requests:
              cpu: "20m"
              memory: "10Mi"
          command:
            - /cluster-proportional-autoscaler
            - --namespace=kube-system
            - --configmap=kube-dns-autoscaler
            - --target=<SCALE_TARGET>
            - --default-params={"linear":{"coresPerReplica":256,"nodesPerReplica":16
              ,"preventSinglePointFailure":true,"includeUnschedulableNodes":true}}
            - --logtostderr=true
            - --v=2

```

Output:

```
deployment.apps/dns-autoscaler created
```

iv. Tune DNS autoscaling parameters

NAME	DATA	AGE
...		
dns-autoscaler	1	...
...		

Modify the data in the ConfigMap:

```
kubectl edit configmap dns-autoscaler --namespace=kube-system
```

v. Disable DNS horizontal autoscaling

1. Scale down the dns-autoscaler deployment to 0 replicas

Output:

```
deployment.apps/dns-autoscaler scaled
```

Verify that the replica count is zero:

Output:

NAME	DESIRED	CURRENT	READY	AGE
...				
dns-autoscaler-6b59789fc8	0	0	0	...
...				

2. Delete the dns-autoscaler deployment

Output:

```
deployment.apps "dns-autoscaler" deleted
```

b. Change the default Storage Class

List the StorageClasses in the cluster

Output:

NAME	PROVISIONER	AGE
standard (default)	kubernetes.io/gce-pd	1d
gold	kubernetes.io/gce-pd	1d

Mark the default StorageClass as non-default:

```
kubectl patch storageclass standard -p '{"metadata":  
{"annotations":{"storageclass.kubernetes.io/is-default-  
class":"false"}}}'
```

Mark a StorageClass as default

Output:

NAME	PROVISIONER	AGE
standard	kubernetes.io/gce-pd	1d
gold (default)	kubernetes.io/gce-pd	1d

6) Extend Kubernetes

a. Use Custom Resources

i. Extend the Kubernetes API with Custom Resource Definitions

1. Create a Custom Resource Definition

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  versions:
    - name: v1
      served: true
      storage: true
  scope: Namespaced
  names:
    plural: crontabs
    singular: crontab
    kind: CronTab
    shortNames:
      - ct
  subresources:
    status: {}
```

Create CRD:

2. Create custom objects

```

apiVersion: v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  versions:
    - name: v1
      served: true
      storage: true
  scope: Namespaced
  names:
    plural: crontabs
    singular: crontab
    kind: CronTab
    shortNames:

```

- ctCreate Object:

```
kubectl apply -f crontab-crd.yaml
```

Orchestrate Object from CronTab:

```
kubectl get crontab
```

To get raw YAML:

```
kubectl get ct -o yaml
```

3) Delete a Custom Resource

Definition

1. `kubectl delete crd`

```
resourcedefinition.yaml kubectl get crontabs
```

b. Configure Multiple Schedulers

Specify schedulers for pods:

Pod spec without any scheduler name:

```
apiVersion: v1 kind: Pod metadata:
name: no-annotation labels:
name: multischeduler-example spec:
containers:
- name: pod-with-no-annotation-container image: k8s.gcr.io/pause:2.0
|
```

Pod spec with default-scheduler:

```
apiVersion: v1 kind: Pod metadata:
name: annotation-default-scheduler labels:
name: multischeduler-example spec:
schedulerName: default-scheduler containers:
- name: pod-with-default-annotation-container image: k8s.gcr.io/pause:2.0
|
```

```
Pod spec with my-scheduler: apiVersion: v1
kind: Pod metadata:

name: annotation-second-scheduler labels:
name: multischeduler-example spec:
schedulerName: my-scheduler containers:
- name: pod-with-second-annotation-container image: k8s.gcr.io/pause:2.0
|
```

4) Manage Cluster Daemons

a. Perform a Rolling Update on a DaemonSet

i. Creating a DaemonSet with RollingUpdate update strategy

```
#!/bin/bash

# Download the YAML file for the DaemonSet
wget -O fluentd-daemonset.yaml https://k8s.io/examples/controllers/fluentd-daemonset.yaml

# Create a DaemonSet with RollingUpdate update strategy
kubectl create -f fluentd-daemonset.yaml

# Alternatively, you can use the "apply" command to update an existing DaemonSet
kubectl apply -f fluentd-daemonset.yaml
|
```

ii. Checking DaemonSet RollingUpdate update strategy


```
#!/bin/bash

# Get the update strategy type for the DaemonSet
UPDATE_STRATEGY=$(kubectl get ds/fluentd-elasticsearch -o go-template='{{.spec.updateStrategy
.type}}' -n kube-system)

# Print the update strategy type
echo "$UPDATE_STRATEGY"

|
```

Examine a Pod IP:

```
#!/bin/bash

# Get the Pod IP addresses and hostnames
kubectl get pods --output=wide | awk '{print $6, $1}' | while read ip pod; do
    echo "Pod: $pod"
    echo "IP: $ip"
    echo "---"
done

# Create a Pod with host aliases
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: hostaliases-pod
spec:
  restartPolicy: Never
  hostAliases:
  - ip: "127.0.0.1"
    hostnames:
    - "foo.local"
    - "bar.local"
  - ip: "10.1.2.3"
    hostnames:
    - "foo.remote"
    - "bar.remote"
  containers:
  - name: cat-hosts
    image: busybox:1.28
    command: ["cat"]
    args: ["/etc/hosts"]
EOF
|
```

You can start a Pod with that configuration by running:

```
kubectl apply -f https://k8s.io/examples/service/networking/hostaliases-pod
```

b. Validate IPv4/IPv6 dual-stack

```
C:\Users\Hp>kubectl get nodes k8s-linuxpool1-34450317-0 -o go-template --template='{{range .spec.podCIDRs}}{{printf "%s\n" .}}{{end}}'
```

Validate Pod addressing:

```
#!/bin/bash

# Get the pod IP address
POD_IP=$(kubectl get pods pod01 -o go-template --template='{{range .status.podIPs}}{{.ip}}{{"\n"}}{{end}}')

# Print the pod IP address
echo "$POD_IP"

# Check for environment variables containing pod IP address
kubectl exec -it pod01 -- printenv | grep MY_POD_IPS

# View the /etc/hosts file inside the pod
kubectl exec -it pod01 -- cat /etc/hosts

|
```

Validate Services:

service/networking/dual-stack-default-svc.yaml:

```
#!/bin/bash

# Create the YAML for the Service
cat <<EOF > dual-stack-default-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
  labels:
    app: MyApp
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
EOF

# Display the contents of the YAML file
cat dual-stack-default-svc.yaml

|
```

Use kubectl to view the YAML for the Service:

```
#!/bin/bash
```

```
# Use kubectl to get the YAML for the Service
```

```
kubectl get svc my-service -o yaml > dual-stack-ipfamilies-ipv6.yaml
```

```
# Display the contents of the YAML file
```

```
cat dual-stack-ipfamilies-ipv6.yaml
```

Use kubectl to view the YAML for the Service:

```
#!/bin/bash
```

```
# Use kubectl to get the YAML for the Service
```

```
kubectl get svc my-service -o yaml > dual-stack-preferred-svc.yaml
```

```
# Display the contents of the YAML file
```

```
cat dual-stack-preferred-svc.yaml
```

```
|
```

Describe:

```
kubectl describe svc -l app=MyApp
```

Create a dual-stack load balanced Service:

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
labels:
  app: MyApp
spec:
  type: LoadBalancer
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
  ipFamilyPolicy: DualStack
  ipFamilies:
    - IPv4
    - IPv6
|
```

Check the Service:

```
kubectl get svc -l app=MyApp
```

5) Extend kubectl with plugins

a. Writing kubectl plugins

Example plugin:

```
#!/bin/bash

# Process optional arguments
case $1 in
    "version")
        echo "1.0.0"
        ;;
    "config")
        echo "$KUBECONFIG"
        ;;
    *)
        echo "I am a plugin named kubectl-foo"
        ;;
esac
|
```

Using a plugin:

```
#!/bin/bash

# Set executable permissions for the plugin
sudo chmod +x ./kubectl-foo

# Move the plugin to the /usr/local/bin directory with the name "kubectl-foo"
sudo mv ./kubectl-foo /usr/local/bin/kubectl-foo

# Execute the plugin command "kubectl foo version"
kubectl foo version

# Set the KUBECONFIG environment variable to ~/.kube/config and execute "kubectl foo config"
KUBECONFIG=~/.kube/config kubectl foo config

# Set the KUBECONFIG environment variable to /etc/kube/config and execute "kubectl foo config"
KUBECONFIG=/etc/kube/config kubectl foo config
|
```

Flags and argument handling:

```
#!/bin/bash

# Define the plugin functionality
echo "My first command-line argument was $1"

# Save the script as a plugin
cat << EOF > kubectl-foo-bar-baz
#!/bin/bash
echo "My first command-line argument was \"$1\""
EOF

# Set executable permissions
chmod +x ./kubectl-foo-bar-baz

# Move the plugin to a directory in the $PATH
sudo mv ./kubectl-foo-bar-baz /usr/local/bin

# Verify that kubectl recognizes the plugin
kubectl plugin list
|
```

6) Manage HugePages

If a pod needs to consume multiple huge page sizes within a single pod specification, it is required to utilize the "medium: HugePages-<hugepagesize>" notation for all volume mounts associated with it.

```
apiVersion: v1
kind: Pod
metadata:
  name: huge-pages-example
spec:
  containers:
    - name: example-container
      image: fedora:latest
      command: ["sleep", "infinity"]
      volumeMounts:
        - name: hugepage-2Mi
          mountPath: /hugepages-2Mi
        - name: hugepage-1Gi
          mountPath: /hugepages-1Gi
      resources:
        limits:
          hugepages-2Mi: 100Mi
          hugepages-1Gi: 2Gi
          memory: 100Mi
        requests:
          memory: 100Mi
  volumes:
    - name: hugepage-2Mi
      emptyDir:
        medium: HugePages-2Mi
    - name: hugepage-1Gi
      emptyDir:
        medium: HugePages-1Gi
        # pod may use medium: HugePages only if it requests huge pages of one size
        :

apiVersion: v1
kind: Pod
metadata:
```

```

  name: huge-pages-example

spec:
  containers:
    - name: example-container
      image: fedora:latest
      command: ["sleep", "infinity"]
      volumeMounts:
        - name: hugepage-volume
          mountPath: /hugepages
  resources:
    limits:
      hugepages-2Mi: 100Mi
      memory: 100Mi
    requests:
      memory: 100Mi
  volumes:
    - name: hugepage-volume
      emptyDir:
        medium: HugePages

```


7) Schedule GPUs

Using device plugins:

```
apiVersion: v1 kind: Pod metadata:  
name: cuda-vector-add spec:  
restartPolicy: OnFailure containers:  
- name: cuda-vector-add  
image: "k8s.gcr.io/cuda-vector-add:v0.1" resources:  
limits: nvidia.com/gpu: 1  
|
```

```
raman@ramango-lenovo: ~  
raman@ramango-lenovo:~$ sudo virsh net-start nat_bridge  
[sudo] password for raman:  
Network nat_bridge started  
raman@ramango-lenovo:~$
```

Fig. 6: Starting nat_bridge for virsh

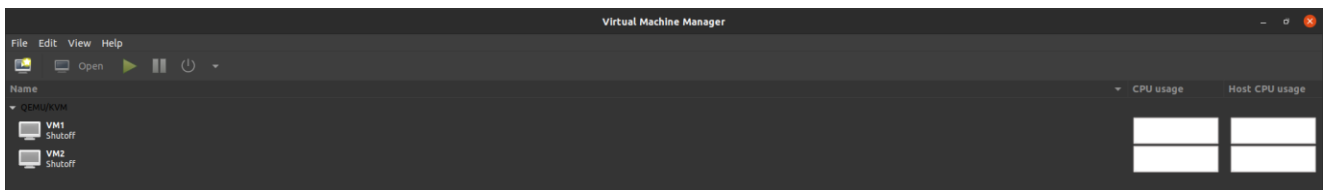


Fig. 7: virsh manager screen

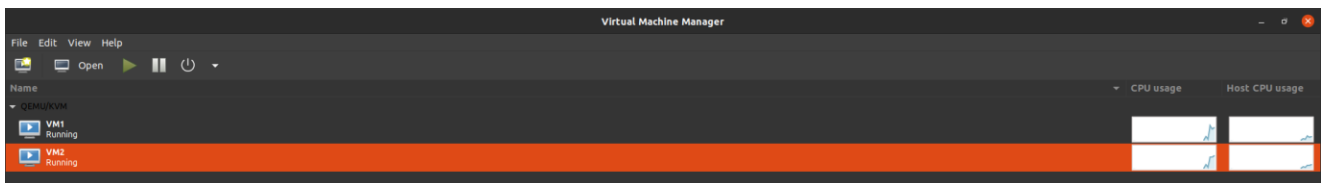


Fig. 8: Turning on VMs

```
raman@master: ~  
raman@master:~$ sudo virsh net-start nat_bridge  
[sudo] password for raman:  
Network nat_bridge started  
raman@ramango-lenovo:~$ ssh 192.168.123.43  
ssh: connect to host 192.168.123.43 port 22: No route to host  
raman@ramango-lenovo:~$ ssh 192.168.123.43  
raman@192.168.123.43's password:  
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.15.0-167-generic x86_64)  
  
 * Documentation:  https://help.ubuntu.com  
 * Management:    https://landscape.canonical.com  
 * Support:       https://ubuntu.com/advantage  
  
System information disabled due to load higher than 2.0  
  
 * Canonical Livepatch is available for installation.  
   - Reduce system reboots and improve kernel security. Activate at:  
     https://ubuntu.com/livepatch  
  
83 packages can be updated.  
2 updates are security updates.  
  
New release '20.04.3 LTS' available.  
Run 'do-release-upgrade' to upgrade to it.  
  
Last login: Mon Feb 14 02:52:59 2022  
raman@master:~$
```

Fig. 9: Creating a ssh session for the VM

```

apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: cpu-autoscale
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageValue: 500m

```

Fig. 10: Default php-apache metric for HPA

```

ece792@t11vm1:~$ kubectl get hpa
NAME                REFERENCE                TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
cpu-autoscale       Deployment/php-apache     1m/500m   1         10        1         115m
ece792@t11vm1:~$ kubectl delete hpa cpu-autoscale
horizontalpodautoscaler.autoscaling "cpu-autoscale" deleted
ece792@t11vm1:~$ kubectl get hpa
No resources found in default namespace.
ece792@t11vm1:~$ kubectl create -f DeployCPU.yaml
horizontalpodautoscaler.autoscaling/cpu-autoscale created
ece792@t11vm1:~$ kubectl get hpa
NAME                REFERENCE                TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
cpu-autoscale       Deployment/php-apache     <unknown>/50m   1         10        0         7s
ece792@t11vm1:~$

```

Fig. 11: Using kubectl to get report of default metric

```

ece792@t11vm1:~$ kubectl create -f DeployCPU.yaml
Error from server (AlreadyExists): error when creating "DeployCPU.yaml": horizontalpodautoscalers.autoscaling "cpu-autoscale" already exists
ece792@t11vm1:~$ kubectl get hpa
NAME                REFERENCE                TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
cpu-autoscale       Deployment/php-apache     1m/50m   1         10        1         20s
ece792@t11vm1:~$ kubectl describe hpa
Name:                cpu-autoscale
Namespace:           default
Labels:               <none>
Annotations:          <none>
CreationTimestamp:    Sat, 22 May 2021 14:55:52 +0000
Reference:            Deployment/php-apache
Metrics:              ( current / target )
  resource cpu on pods: 1m / 50m
Min replicas:         1
Max replicas:         10
Deployment pods:      1 current / 1 desired
Conditions:
  Type           Status  Reason                        Message
  ----           -
  AbleToScale    True    ReadyForNewScale              recommended size matches current size
  ScalingActive  True    ValidMetricFound              the HPA was able to successfully calculate a replica count from cpu resource
  ScalingLimited False   DesiredWithinRange            the desired count is within the acceptable range
Events:          <none>
ece792@t11vm1:~$

```

Fig. 12: Reloading php-apache metric


```

apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: memory-autoscale
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: memory
      target:
        type: AverageValue
        averageValue: 500Mi

```

```

apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: memory-autoscale
spec:
  scaleTargetRef:
    apiVersion: apps.openshift.io/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 50

```

Fig. 16: Custom Metric derived from php-apache using default API and openshift API

```

ece792@t11vm1:~$ kubectl get hpa
NAME                REFERENCE                TARGETS      MINPODS  MAXPODS  REPLICAS  AGE
memory-autoscale    Deployment/php-apache    <unknown>/5Mi  1         10        0          7s
ece792@t11vm1:~$ kubectl describe hpa
Name:                memory-autoscale
Namespace:           default
Labels:              <none>
Annotations:         <none>
CreationTimestamp:    Sat, 22 May 2021 15:19:12 +0000
Reference:            Deployment/php-apache
Metrics:              ( current / target )
  resource memory on pods: 14000128 / 5Mi
Min replicas:         1
Max replicas:         10
Deployment pods:      1 current / 3 desired
Conditions:
  Type            Status  Reason                        Message
  ----            -
  AbleToScale     True    SucceededRescale              the HPA controller was able to update the target scale to 3
  ScalingActive   True    ValidMetricFound              the HPA was able to successfully calculate a replica count from memory resource
  ScalingLimited  False   DesiredWithinRange            the desired count is within the acceptable range
Events:
  Type    Reason              Age    From                      Message
  ----    -
  Normal  SuccessfulRescale   5s     horizontal-pod-autoscaler  New size: 3; reason: memory resource above target
ece792@t11vm1:~$

```

Fig. 17: Result of custom metric of php-apache in HPA, unable to Limit Scaling


```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
  - type: Pods
    pods:
      metric:
        name: I/O operations per second
        target:
          type: AverageValue
          averageValue: 1k
  - type: Object
    object:
      metric:
        name: requests-per-second
      describedObject:
        apiVersion: networking.k8s.io/v1beta1
        kind: Ingress
        name: main-route
      target:
        type: Value
        value: 10k
```

Fig. 18: Final Custom Metric

[illegible]

Fig. 19: Readings for Custom Metric in HPA

```

ece792@t11vm1:~$ ls
components.yaml DeployCPU.yaml DeployCustomCPU.yaml DeployCustom.yaml DeployMemory.yaml dockerfile dockerfiles index1.php index.php
ece792@t11vm1:~$ vim DeployCustom.yaml
ece792@t11vm1:~$ kubectl get hpa
NAME                REFERENCE                TARGETS      MINPODS   MAXPODS   REPLICAS   AGE
memory-autoscale    Deployment/php-apache     12941312/5Mi 1          10        10         5m25s
ece792@t11vm1:~$ kubectl describe hpa
Name:                memory-autoscale
Namespace:           default
Labels:              <none>
Annotations:         <none>
CreationTimestamp:   Sat, 22 May 2021 15:19:12 +0000
Reference:           Deployment/php-apache
Metrics:             ( current / target )
  resource memory on pods: 12941312 / 5Mi
Min replicas:        1
Max replicas:        10
Deployment pods:     10 current / 10 desired
Conditions:
  Type           Status  Reason                        Message
  ----           -
  AbleToScale    True    ReadyForNewScale             recommended size matches current size
  ScalingActive  True    ValidMetricFound             the HPA was able to successfully calculate a replica count from memory resource
  ScalingLimited True    TooManyReplicas              the desired replica count is more than the maximum replica count
Events:
  Type           Reason             Age           From                    Message
  ----           -
  Normal         SuccessfulRescale  5m21s        horizontal-pod-autoscaler  New size: 3; reason: memory resource above target
  Normal         SuccessfulRescale  4m51s        horizontal-pod-autoscaler  New size: 6; reason: memory resource above target
  Normal         SuccessfulRescale  4m5s         horizontal-pod-autoscaler  New size: 10; reason: memory resource above target
ece792@t11vm1:~$

```

Fig. 20: Readings after decreasing load

```

memcached-operator deployment.yaml
Project: memcached-operator
2 # Resource
3 # kube clients
4 # metadata, spec, status
5
6 apiVersion: apps/v1 v1
7 kind: Deployment
8 metadata:
9   annotations:
10     deployment.kubernetes.io/revision: "3"
11     kubectl.kubernetes.io/last-applied-configuration: |
12       {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations":{},"labels":{"control-plane":"controller-manager"}}}
13   creationTimestamp: "2021-07-15T00:05:30Z"
14   generation: 3
15   labels:
16     control-plane: controller-manager
17   name: memcached-operator-controller-manager
18   namespace: memcached-operator-system
19   resourceVersion: "28818"
20   selfLink: /apis/apps/v1/namespaces/memcached-operator-system/deployments/memcached-operator-controller-manager

```

Fig. 21: Memcached Operator deployment YAML file

```

apiVersion: apps/v1 v1
kind: Deployment
metadata: <9 keys>
spec: <6 keys>
status:
  availableReplicas: 1
  conditions:
  - lastTransitionTime: "2021-07-15T00:05:30Z"
    lastUpdateTime: "2021-07-15T00:19:45Z"
    message: ReplicaSet "memcached-operator-controller-manager-6d4bf87bd6" has successfully progressed.
    reason: NewReplicaSetAvailable
    status: "True"
    type: Progressing
  - lastTransitionTime: "2021-07-15T00:22:55Z"
    lastUpdateTime: "2021-07-15T00:22:55Z"
    message: Deployment has minimum availability.
    reason: MinimumReplicasAvailable
    status: "True"

```

Fig. 22: Memcached Operator deployment status


```

ece792@t11vm1:~$ kubectl describe hpa
Name:          cpu-autoscale
Namespace:     default
Labels:        <none>
Annotations:   <none>
CreationTimestamp: Sat, 22 May 2021 15:50:31 +0000
Reference:     Deployment/php-apache
Metrics:       ( current / target )
  resource cpu on pods:   63m / 50m
  resource memory on pods: 13047398400m / 15Mi
Min replicas:      1
Max replicas:     10
Deployment pods:   10 current / 10 desired
Conditions:
  Type           Status  Reason                        Message
  ----           -
  AbleToScale    True    ReadyForNewScale             recommended size matches current size
  ScalingActive  True    ValidMetricFound             the HPA was able to successfully calculate a replica count from cpu resource
  ScalingLimited True    TooManyReplicas              the desired replica count is more than the maximum replica count
Events:
ece792@t11vm1:~$

```

Fig. 26: Readings of php-apache metric in HPA using Cluster Autoscaler

```

apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Object
    object:
      metric:
        name: requests-per-second
      describedObject:
        apiVersion: networking.k8s.io/v1beta1
        kind: Ingress
        name: main-route
      target:
        type: Value
        value: 10k

```

Fig. 27: Custom metric in HPA using Cluster Autoscaler

```

ece792@t11vm1:~$ kubectl get hpa
NAME                REFERENCE                TARGETS      MINPODS   MAXPODS   REPLICAS   AGE
memory-autoscale    Deployment/php-apache     12941312/5Mi 1          10         10          6m52s
ece792@t11vm1:~$ kubectl get hpa
NAME                REFERENCE                TARGETS      MINPODS   MAXPODS   REPLICAS   AGE
memory-autoscale    Deployment/php-apache     12941312/5Mi 1          10         10          8m38s
ece792@t11vm1:~$ kubectl describe hpa
Name:                memory-autoscale
Namespace:           default
Labels:              <none>
Annotations:         <none>
CreationTimestamp:   Sat, 22 May 2021 15:19:12 +0000
Reference:           Deployment/php-apache
Metrics:             ( current / target )
  resource memory on pods: 12941312 / 5Mi
Min replicas:        1
Max replicas:        10
Deployment pods:     10 current / 10 desired
Conditions:
  Type            Status  Reason
  ----            -
  AbleToScale     True    ReadyForNewScale
  ScalingActive   True    ValidMetricFound
  ScalingLimited  True    TooManyReplicas
Message:          recommended size matches current size
                  the HPA was able to successfully calculate a replica count from memory resource
                  the desired replica count is more than the maximum replica count
Events:
  Type    Reason
  ----    -
  Normal  SuccessfulRescale 8m25s horizontal-pod-autoscaler New size: 3; reason: memory resource above target
  Normal  SuccessfulRescale 7m55s horizontal-pod-autoscaler New size: 6; reason: memory resource above target
  Normal  SuccessfulRescale 7m9s  horizontal-pod-autoscaler New size: 10; reason: memory resource above target

```

Fig. 28: Readings of custom metric in HPA using Cluster Autoscaler

```

ece792@t11vm1:~$ kubectl get hpa
No resources found in default namespace.
ece792@t11vm1:~$ kubectl create -f DeployCustom.yaml
horizontalpodautoscaler.autoscaling/php-apache created
ece792@t11vm1:~$ kubectl get hpa
NAME                REFERENCE                TARGETS      MINPODS   MAXPODS   REPLICAS   AGE
php-apache          Deployment/php-apache     <unknown>/1k 1          10         0           4s
ece792@t11vm1:~$ kubectl describe hpa
Name:                php-apache
Namespace:           default
Labels:              <none>
Annotations:         <none>
CreationTimestamp:   Sat, 22 May 2021 15:34:36 +0000
Reference:           Deployment/php-apache
Metrics:             ( current / target )
  "requests-per-second" on Ingress/main-route (target value): <unknown> / 1k
Min replicas:        1
Max replicas:        10
Deployment pods:     10 current / 0 desired
Conditions:

```

Fig. 29(a): Final Readings of custom metric in HPA using Cluster Autoscaler

```

Conditions:
  Type            Status  Reason
  ----            -
  AbleToScale     True    SucceededGetScale
  ScalingActive   False   FailedGetObjectMetric
Message:          the HPA controller was able to get the target's current scale
                  the HPA was unable to compute the replica count: unable to get metric requests-per-second: Ingress on default main-route/unable to fe
                  tch metrics from custom metrics API: no custom metrics API (custom.metrics.k8s.io) registered
Events:
  Type    Reason
  ----    -
  Warning  FailedGetObjectMetric 0s horizontal-pod-autoscaler unable to get metric requests-per-second: Ingress on default main-route/unable to fetch metrics from custom
                  metrics API: no custom metrics API (custom.metrics.k8s.io) registered
  Warning  FailedComputeMetricsReplicas 0s horizontal-pod-autoscaler invalid metrics (1 invalid out of 1), first error is: failed to get object metric value: unable to get metr
                  ics requests-per-second: Ingress on default main-route/unable to fetch metrics from custom metrics API: no custom metrics API (custom.metrics.k8s.io) registered
ece792@t11vm1:~$

```

Fig. 29(b): Error in Previous Readings of custom metric in HPA using Cluster Autoscaler
(No custom metric API registered)

Result and Analysis

HPA

In this study, we conducted a comprehensive exploration and comparison of diverse autoscaling strategies and metrics, aiming to evaluate their effects on CPU and memory utilization.

Initially, we implemented horizontal autoscaling utilizing both default and custom metrics. The results of this analysis are presented in Figures 30 and 31, which demonstrate CPU utilizations at different intervals. Additionally, Figures 32 and 34 illustrate the dynamic scrapping of pods at various intervals, while Figures 35 and 36 exhibit the corresponding memory usage during the horizontal autoscaling process.

Furthermore, we extended our investigation to include cluster autoscaling. In this context, Figures 37 and 38 provide insights into the CPU utilizations at different intervals. Additionally, Figures 39 and 40 showcase the autoscaling of nodes, depicting the adaptability of the cluster to varying workload demands. Lastly, Figures 41 and 42 highlight the memory usage patterns observed during the operation of the cluster autoscaler.

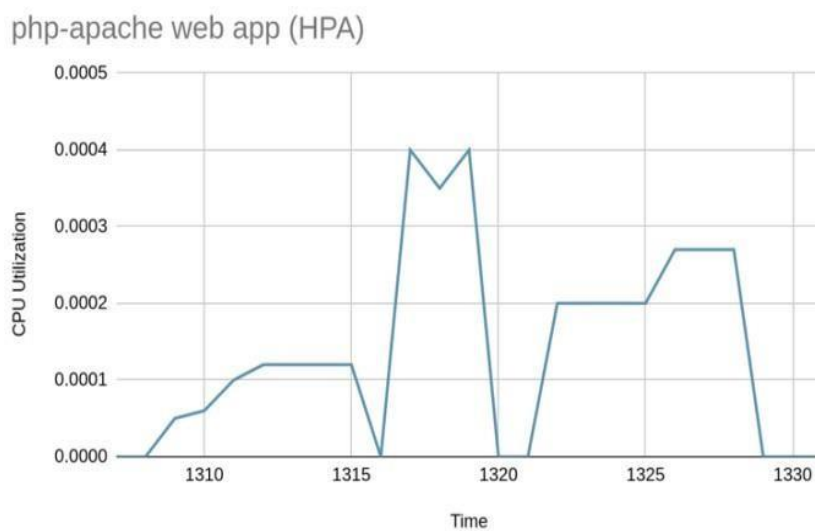


Fig: 30 The average CPU utilization (Default Metrics).

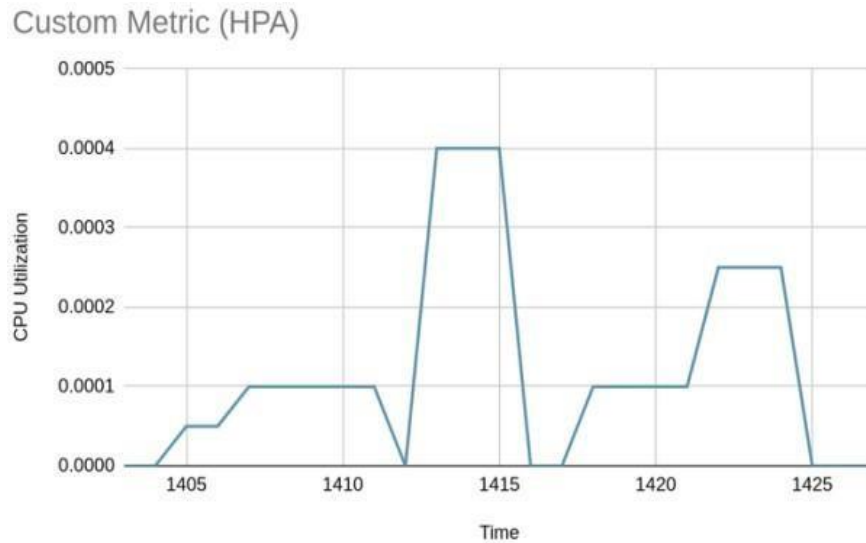


Fig 31: The average CPU utilization (Custom Metrics).

Comparing the php apache metric with the custom metric designed by the team with respect to the CPU utilization on the same workload over an equal period, we conclude that CPU utilization is more uniform as well as has reduced in many time periods on the custom metric.

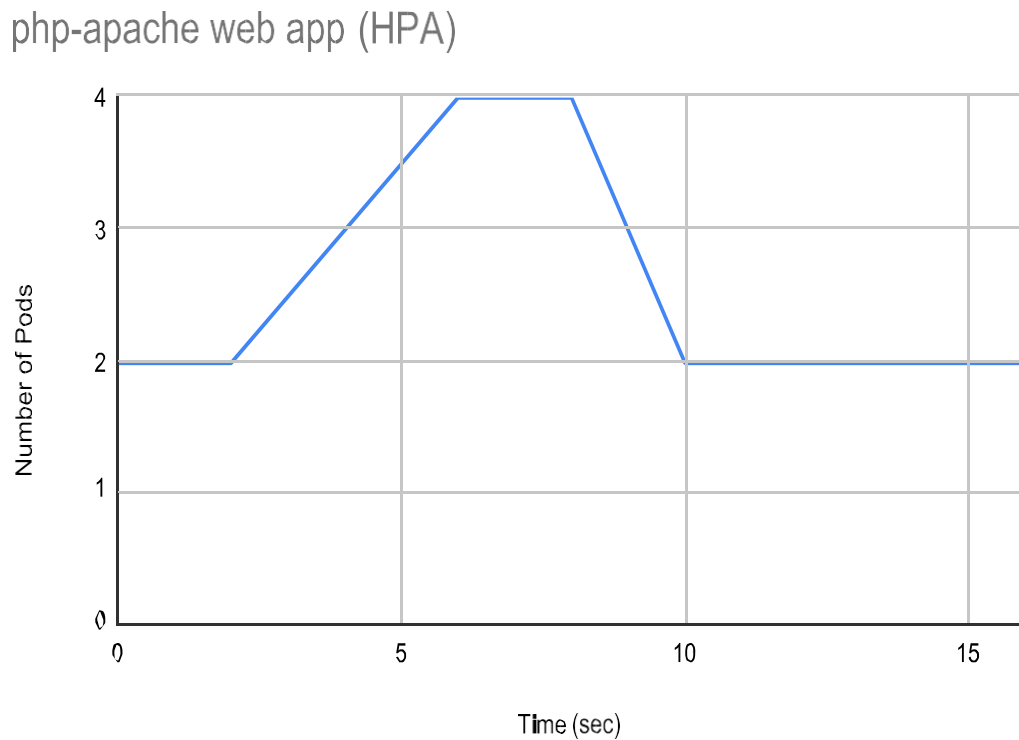
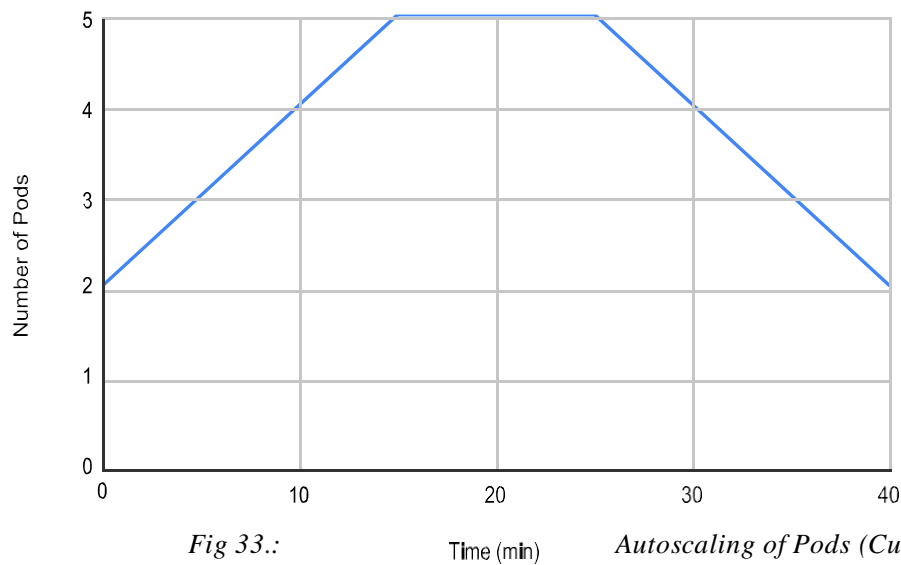


Fig 32: Autoscaling of Pods (Default Metrics).

Custom Metric (HPA)



The Default and Custom Metric Horizontal Autoscaler graphs show the number of pods adjusting in response to changes in CPU or memory usage. The Custom Metric graph allows users to define their own metrics for scaling behavior. Both graphs show the number of pods increasing as usage increases, but the Custom Metric graph reaches a higher maximum number of pods due to the specific workload being tracked.[5]

Horizontal Pod Autoscaler - Default Metric:

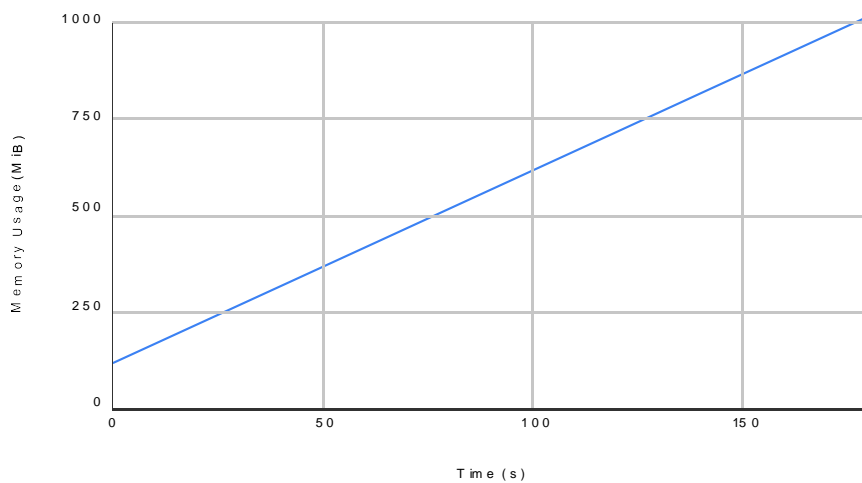


Fig 34: Memory usage in (Default Metrics).

Horizontal Pod Autoscaler - Custom Metric:

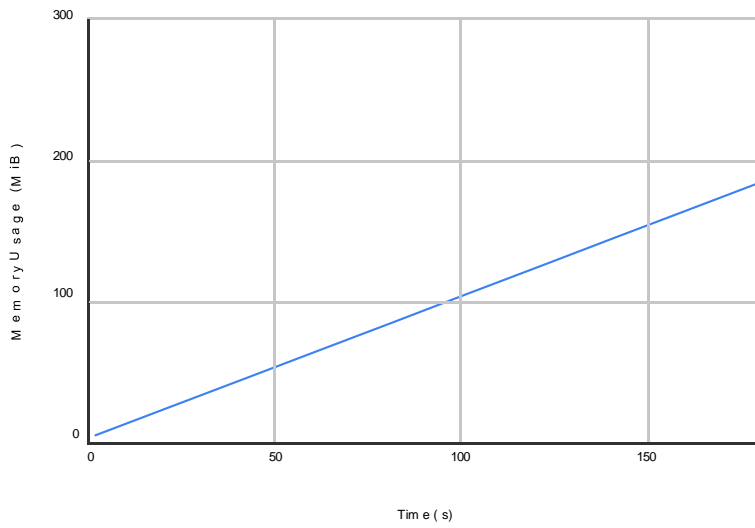


Fig 35: Memory usage in (Custom Metrics).

For the Horizontal Pod Autoscaler, the default metric graph shows a linear increase in memory usage over time due to CPU usage, while the custom metric graph shows a slower and more stable increase in memory usage over time due to a more specific metric that is relevant to the application. The custom metric allows for more accurate scaling decisions that consider the specific requirements of the application. [5]

Cluster Autoscaler

We benchmarked both the metrics on the same workload for an equivalent period. On comparison of both graphs, we conclude that the custom metric has a much more uniform and more efficient CPU utilization.

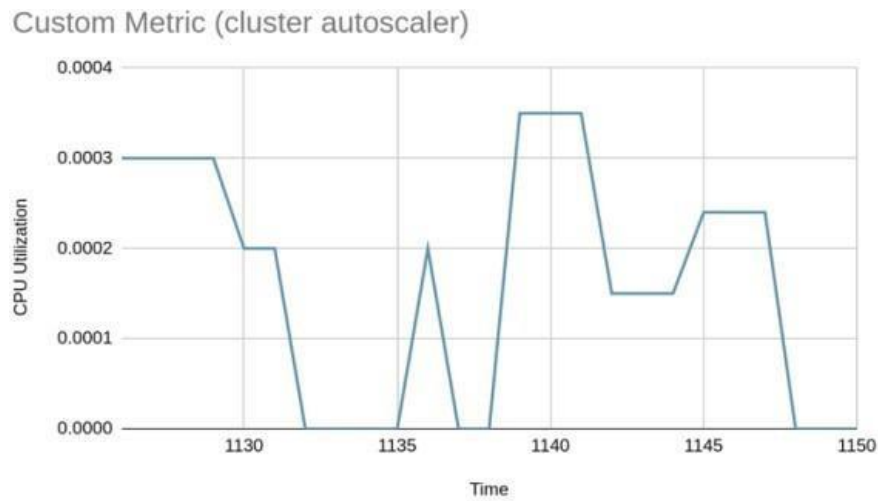


Fig 36: The average CPU utilization (Default Metrics).

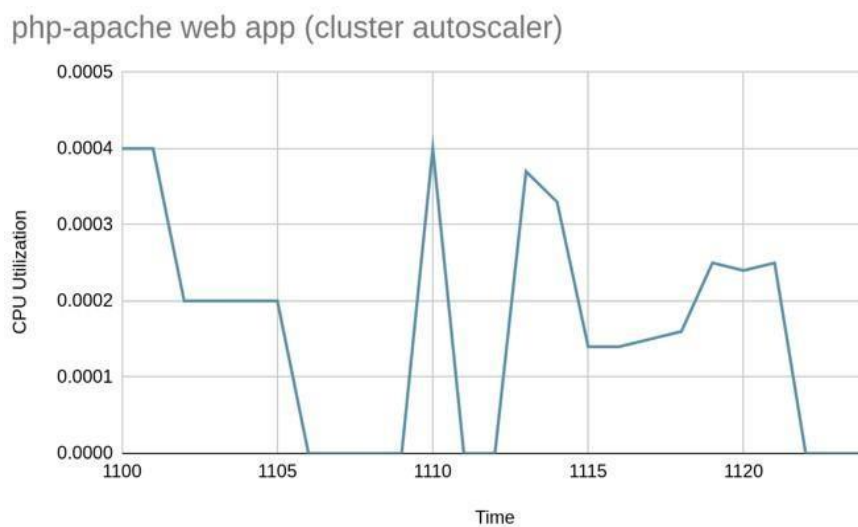


Fig 37: The average CPU utilization (Custom Metrics).

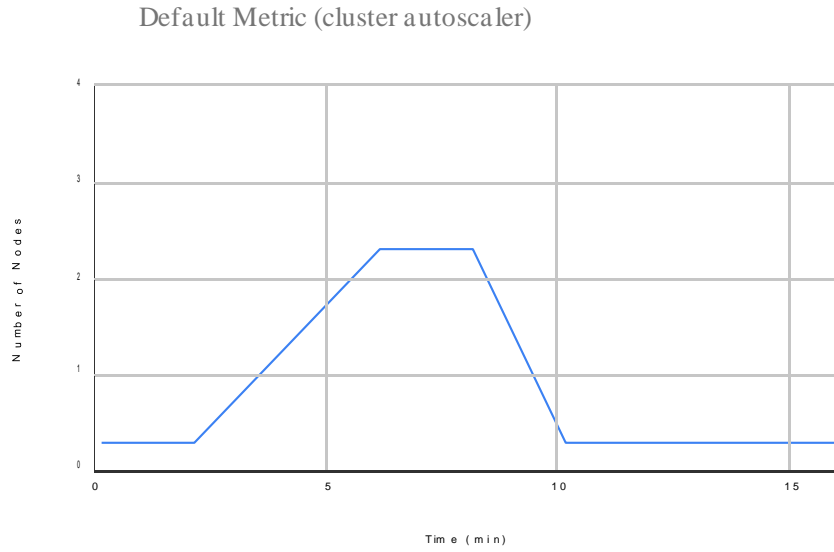


Fig 38: Autoscaling of Nodes (Default Metrics).

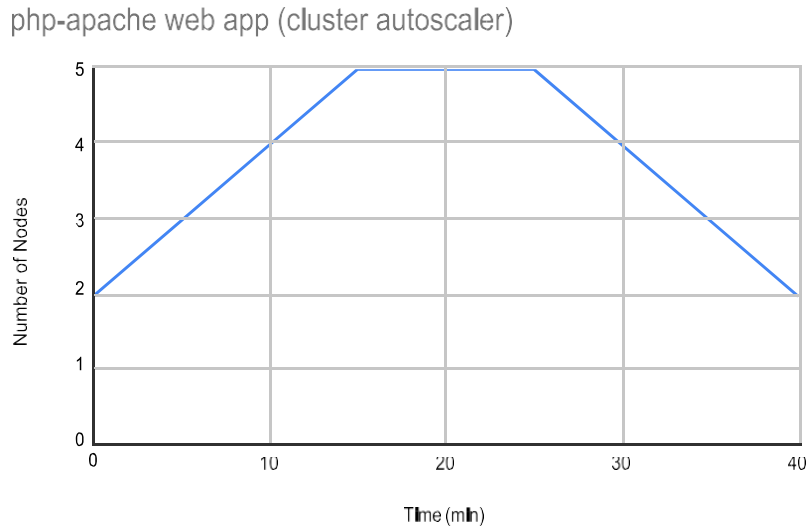


Fig 39: Autoscaling of Nodes (Custom Metrics).

We benchmarked both the metrics on the same workload for an equivalent period. On comparison of both graphs, we conclude that the custom metric has a much more uniform and more efficient CPU utilization.

The default metric cluster autoscaler graph shows how the number of nodes in a cluster change over time based on CPU or memory usage, while the custom metric cluster autoscaler graph tracks a specific workload that requires more nodes as usage increases. The cluster autoscaler adds nodes as usage increases and removes them as usage decreases to keep resource utilization within a specified range [6].

Cluster Autoscaler - Default Metric:

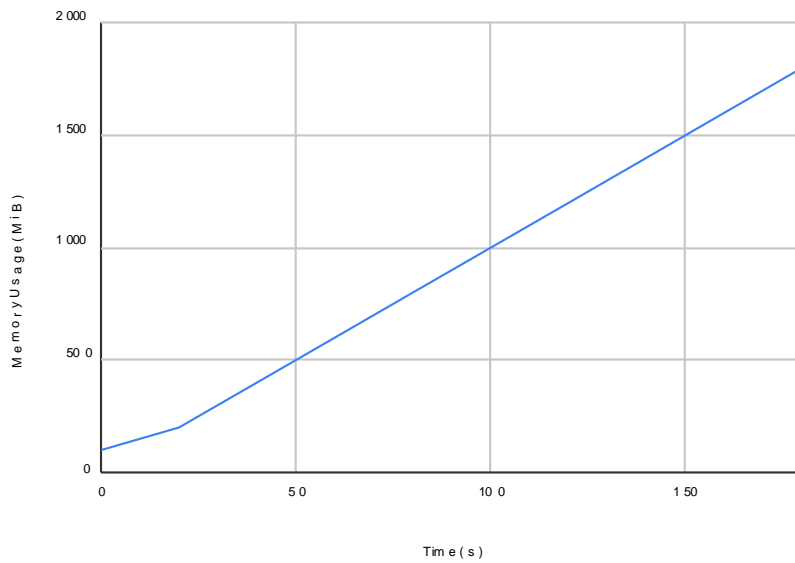


Fig 40: Memory usage in (Default Metrics).

Cluster Autoscaler - Custom Metric:

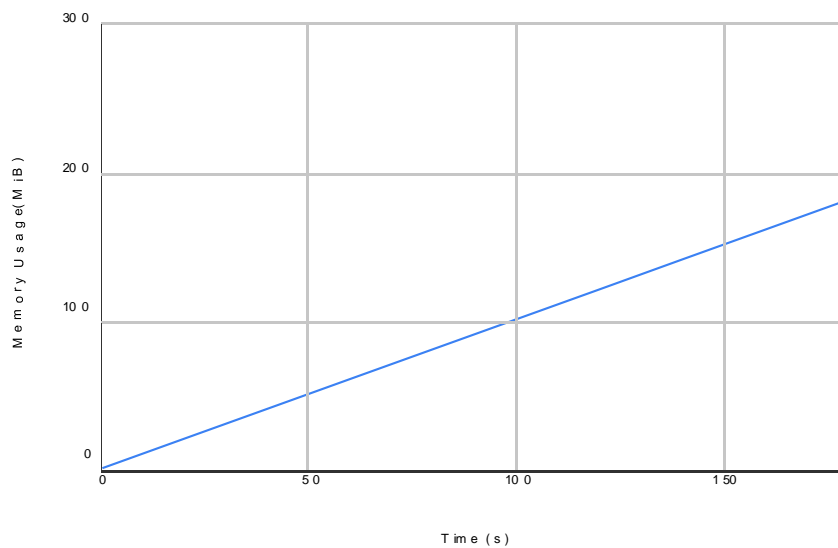


Fig 41: Memory usage in (Custom Metrics).

The default metric graph for Cluster Autoscaler shows steep increase in memory usage as workload increases, resulting in inefficient resource utilization. In contrast, the custom metric graph shows gradual increase in memory usage as the cluster is scaled up to match the specific requirements of the workload, resulting in more efficient resource utilization.

Operator

An Operator is used for packaging, deploying, and orchestrating a Kubernetes application, where the application is our workload. The workload is both deployed on K8s and orchestrated using the API and tools present in it, specifically the kubectl command line tool.

Operator Pattern Key Points

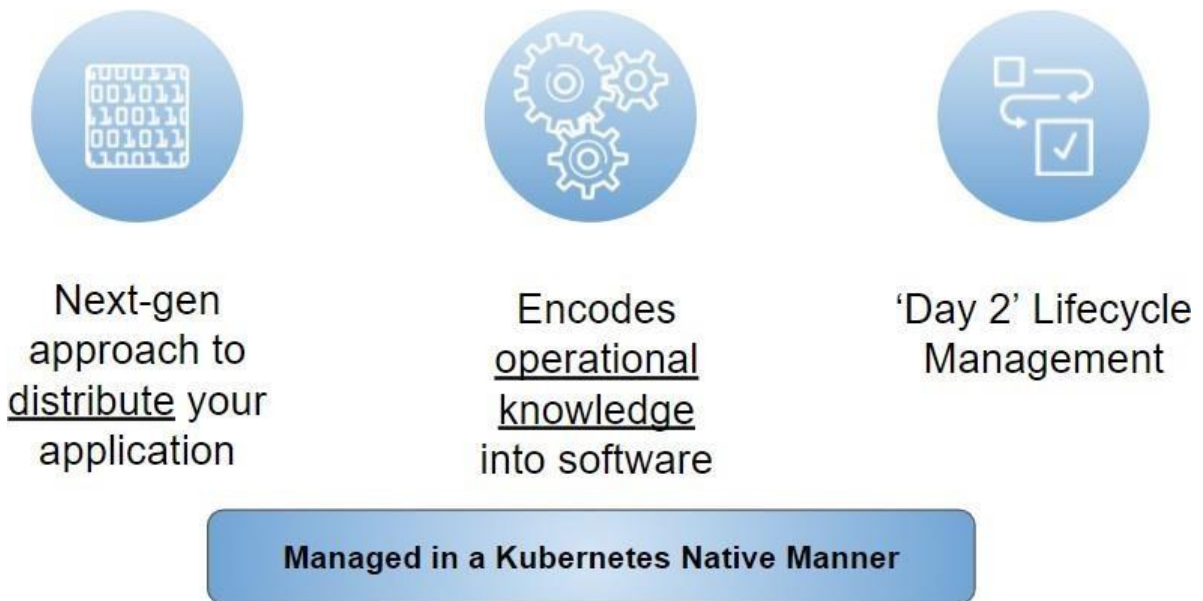


Fig. 42: The basic operator pattern

Operator Architecture

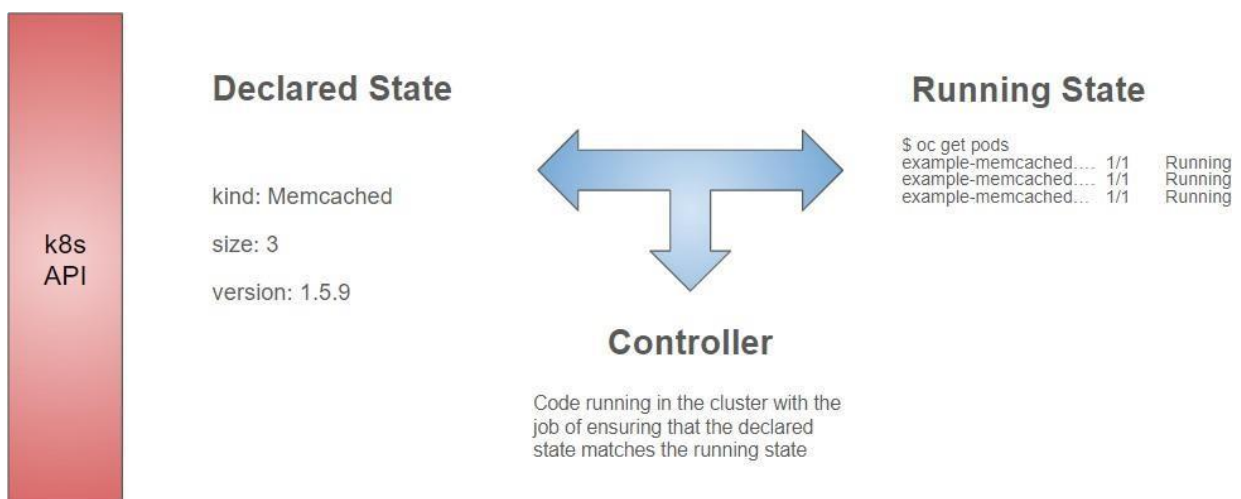


Fig. 43: Operator management in Kubernetes API; Declared desired state and wait for the controller to make it happen

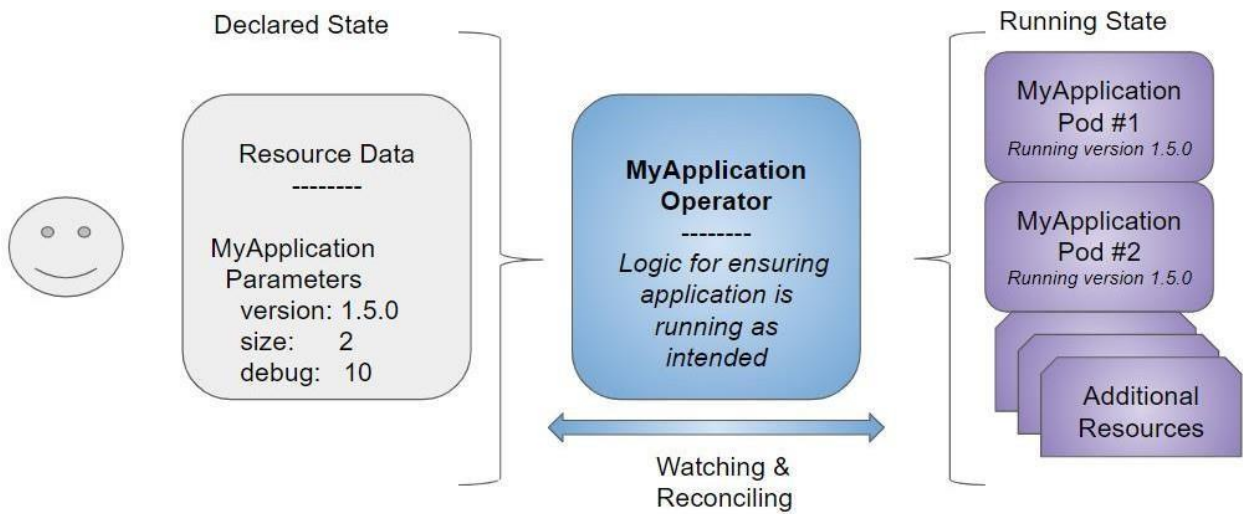


Fig. 44: Separate controller watching declared state and reconciling with running state

End User perspective with Custom Resources

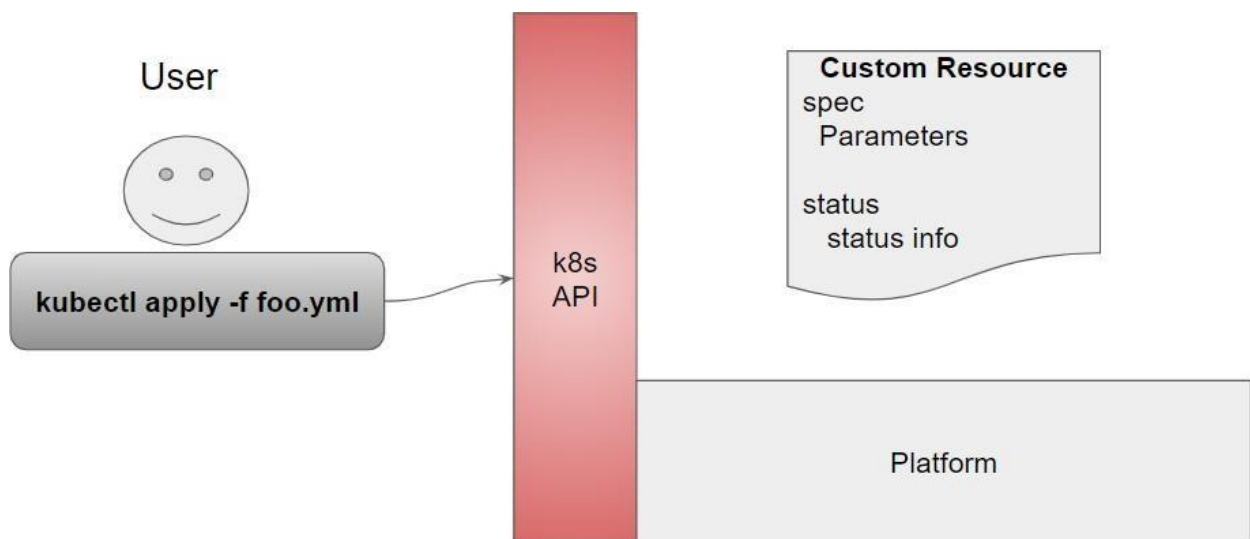


Fig. 45: End user declares intent and waits for system to make it happen

Memcached Operator

```
apiVersion: cache.example.com/v1alpha1
kind: Memcached
metadata:
  name: example-memcached
spec:
  size: 3
  debug: 10
```

Fig. 46: Our Controller will register to k8s API for info related to GVK

Final Operator Architecture

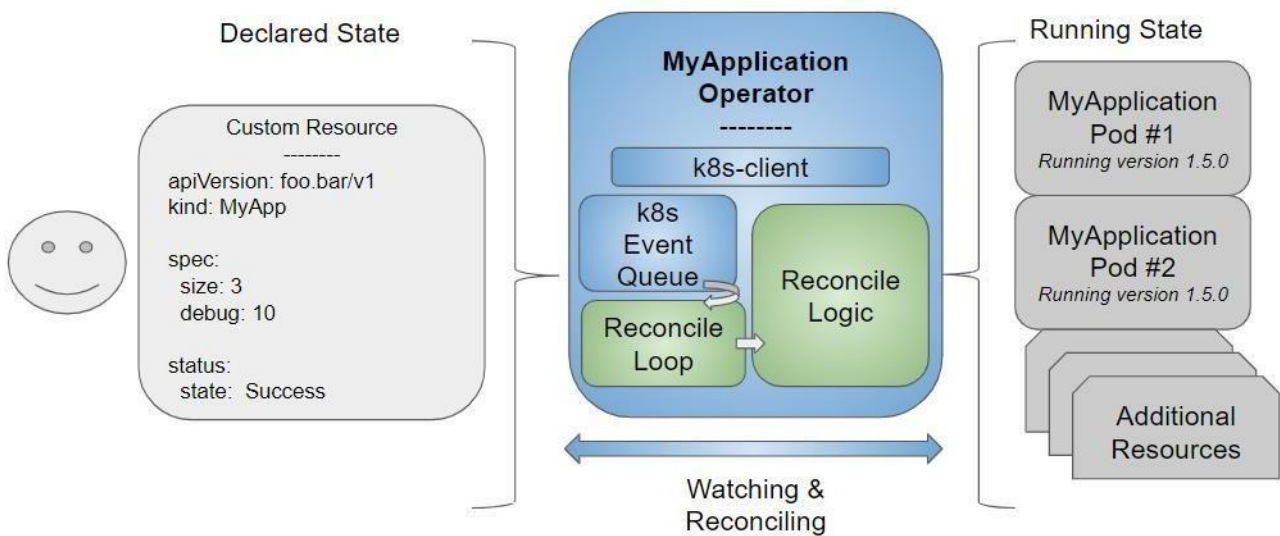


Fig. 47: Reconcile loop receives events on watched resources and executes reconcile logic

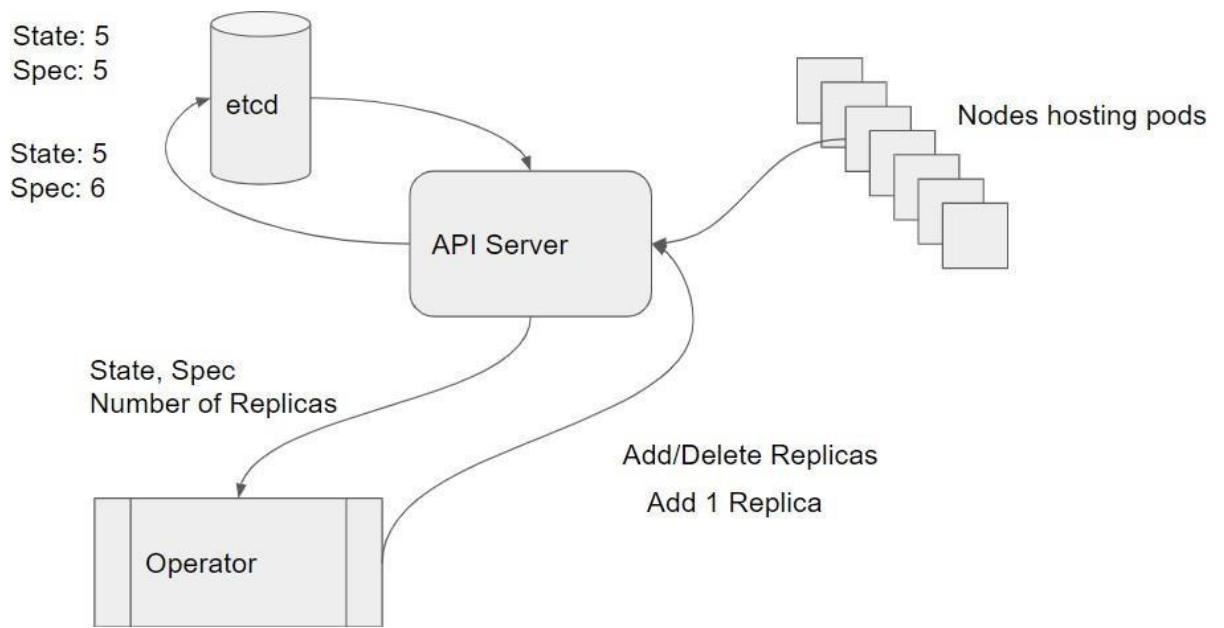


Fig. 48: Operator Feedback

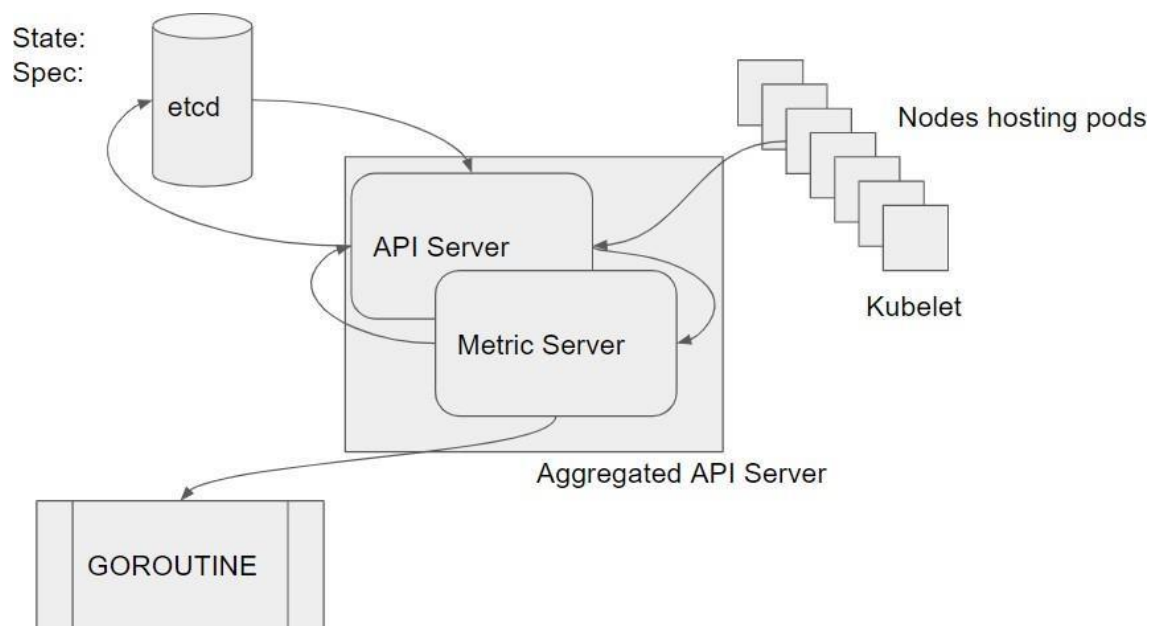


Fig. 49: Feedback/Masurement of the Operator Loop

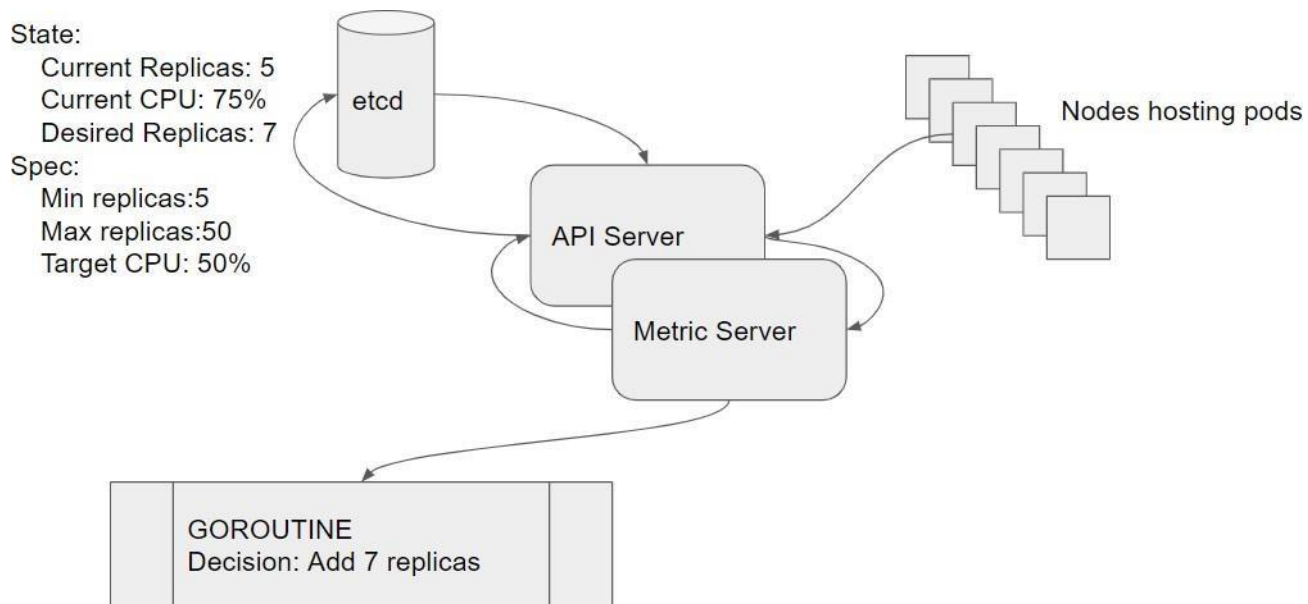


Fig. 50: Closing the loop

Conclusion

All the preceding observations are a result of taking a median of two different development environments, one by Linux VM Generic Method other by Multipass software Resource Allocation environment. From both the above comparisons, a conclusion can be drawn that the custom metric is more reliable than the standard php-apache web app metric in either case whether it be Horizontal Pod Autoscaler, or Cluster Autoscaler. Although, the default metric loads up faster and spikes up performance in between intervals, the custom metric provides a stable runtime in overall running of the workload.

Additionally, the number of pods and nodes, as well as memory utilization, were also discussed in the preceding analysis. The graphs show that as workload increases, the number of pods and nodes also increase to meet the demands of the workload. Memory utilization also increases as workload increases, indicating that the system is utilizing more resources to handle the workload. The use of custom metrics helps ensure that these resources are used more efficiently and accurately to meet the specific needs of the workload. So, from this we can conclude, that for small applications like web apps, and small databases, default metric is better, but if the application proves to have a larger workload on the resources, it is better to deploy custom based metrics, based on the requirements.

In case of techniques, Cluster Autoscaler with Horizontal Pod Autoscaler, although requires high resources, can handle much better workloads, then there is Cluster Autoscaler alone that is also better proven than Horizontal Pod Autoscaler. Horizontal Pod Autoscaler, although is a great technique, but in its primal form, not very efficient.

Future Scope

Autoscaling is crucial in cloud computing to match resource allocation with application demand. Comparing the Linux VM Generic Method and Multipass software Resource Allocation, custom metrics prove more reliable than the standard php-apache web app metric for larger workloads. While the default metric loads faster and enhances performance intermittently, the custom metric provides overall stability.

For small applications like web apps and small databases, the default metric is recommended. However, larger workloads benefit from deploying custom-based metrics tailored to specific requirements.

Future work can focus on enhancing the efficiency of Horizontal Pod Autoscaler and exploring alternative metrics and techniques for diverse workloads. Developing a framework that automates metric and technique selection based on workload requirements would streamline the autoscaling process.

Additionally, further research should investigate the impact of autoscaling on overall performance, cost, and scalability.

Reference

- [1] Elmroth, Guillaume Pierre. An Experimental Evaluation of the Kubernetes Cluster Autoscaler in the Cloud. CloudCom 2020 - 12th IEEE International Conference on Cloud Computing Technology and Science, Dec 2020, Bangkok, Thailand. pp.1-9. hal- 02958916
- [2] M. Wang, D. Zhang and B. Wu, "A Cluster Autoscaler Based on Multiple Node Types in Kubernetes," 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 2020, pp. 575-579, doi: 10.1109/ITNEC48623.2020.9084706.
- [3] Nguyen, Thanh-Tung, Yu-Jin Yeom, Taehong Kim, Dae-Heon Park, and Sehan Kim. 2020. "Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration" Sensors 20,no.16:4621. <https://doi.org/10.3390/s20164621>
- [4] Salman Taherizadeh, Marko Grobelnik, Key influencing factors of the Kubernetes auto- scaler for computing-intensive microservice- native cloud-based applications, Advances in Engineering Software, Volume 140, 2020, 102734,ISSN 0965-9978, <https://doi.org/10.1145/3366615.3368357>
- [5] Stef Verreydt, Emad Heydari Beni, Eddy Truyen, Bert Lagaisse, and Wouter Joosen. 2019. Leveraging Kubernetes for adaptive and costefficient resource management. In Proceedings of the 5th International Workshop on Container Technologies and Container Clouds (WOC '19). Association for Computing Machinery, New York, NY, USA, 37–42.
- [6] D. Balla, C. Simon and M. Maliosz, "Adaptive scaling of Kubernetes pods," NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, 2020, pp. 1-5, doi: 10.1109/NOMS47738.2020.9110428.
- [7] Khazaei, Hamzeh & Ravichandiran, Rajsimman & Park, Byungchul & Bannazadeh, Hadi & Tizghadam, Ali & Leon- Garcia, A.. (2017). Elascare: Autoscaling and Monitoring as a Service.
- [8] L.Versluis, M. Neacsu, and A. Iosup, "A trace-based performance study of autoscaling workloads of work-flows in datacenters," in Proc. IEEE/ACM CCGRID,2018
- [9] <https://arxiv.org/abs/2010.05049>
- [10] Ilyushkin, A. Ali-Eldin, N. Herbst, A. Bauer, A. V.Papadopoulos, D. Epema, and A. Iosup, "An experimental performance evaluation of autoscalers for complex workflows," ACM Transactions on Modeling and Performance iEvaluation of Computing Systems, vol. 3,no. 2, 2018
- [11] A. Ali-Eldin, O. Seleznev, S. Sj ostedt-de Luna, J.Tordsson, and E. Elmroth, "Measuring cloud workload burstiness," in Proc. IEEE UCC, 2014
- [12] Pahl C., Brogi A., Soldani J., Jamshidi P. Cloud Container Technologies: A State-of-the- Art Review. IEEE Trans. Cloud Comput. 2017; 7:677–692.
- [13] K. Berezovskyi, P. Laskov, J. Balazs, and A. Papadopoulos. "Autoscaling for Microservice Architectures using Machine Learning-based Time Series Forecasting." 2020 IEEE International Conference on Big Data (Big Data), 2020.
- [14] S. M. Salam, C. Caiazza, S. L. S. Rausch, and A. A. Menasc . "Real-Time Vertical and Horizontal Autoscaling in Container-Based Cloud Systems." Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), 2018.
- [15] G. D'Ambrosio, D. Didona, N. Nostro, P. Romano, and W. Zwaenepoel. "An Experimental Evaluation of

Autoscaling Policies for Complex Workflows." IEEE Transactions on Cloud Computing, vol. 6, no. 4, 2018.

[16] L. Wang, F. Liu, Z. Zong, and S. Chen. "Hierarchical Load Balancing and Autoscaling for Microservices in Kubernetes." IEEE Transactions on Services Computing, vol. 12, no. 4, 2019.

[17] A. Boominathan, B. Jain, S. Muralidharan, and H. Pucha. "Autoscaling for Cloud Microservices with Reinforcement Learning." Proceedings of the 19th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), 2019.

[18] Suresh Mohan, Huichen Zhang, Gaurav Somani, and Saurabh Bagchi. "PERIKLES: Feedback-Driven Autoscaling for Containerized Microservices." Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20), 2020.

● 9% Overall Similarity

Top sources found in the following databases:

- 8% Internet database
- 3% Publications database
- Crossref database
- Crossref Posted Content database
- 5% Submitted Works database

TOP SOURCES

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	leoh0.github.io	Internet	3%
2	kubernetes.io	Internet	2%
3	github.com	Internet	2%
4	casesup.com	Internet	1%
5	University of Portsmouth on 2022-07-22	Submitted works	<1%
6	cast.ai	Internet	<1%
7	Liverpool John Moores University on 2023-03-06	Submitted works	<1%
8	beanexpert.co.in	Internet	<1%

9	upcommons.upc.edu	<1%
	Internet	
10	International University - VNUHCM on 2020-07-19	<1%
	Submitted works	
11	Noroff University College on 2019-04-14	<1%
	Submitted works	
12	dzone.com	<1%
	Internet	