Optimizing Cloud-Native App Deployment using Kubernetes Scheduler on AWS Cloud

MSc Research Project
Cloud Computing

Swapnil Deshpande
Student ID: x22159401

School of Computing
National College of Ireland

Supervisor: Dr Ahmed Makki

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Swapnil Deshpande
Student ID:	x22159401
Programme:	Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr Ahmed Makki
Submission Due Date:	14/12/2023
Project Title:	Optimizing Cloud-Native App Deployment using Kubernetes Scheduler on AWS Cloud
Word Count:	
Page Count:	23

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:	
Date:	13th December 2023

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):	

Optimizing Cloud-Native App Deployment using Kubernetes Scheduler on AWS Cloud

Swapnil Deshpande
x22159401

Abstract

The demand for effective orchestration technologies with smart scheduling algorithms containerization gain traction.This paper presents a novel approach to for scheduling of microservice application and also their management using a popular software containerization platform .The suggested approach, known as the Dependency-Based Scheduler, is centered on ensuring that the interconnected components of a program, known as microservices, function well together within same node. This is crucial because improper handling of these connections by current techniques could result in a less effective use of computer resources and high network bandwidth usage. The new method seeks to decrease the amount of data that is shared between these microservices and speed up overall performance and ultimately help organization to run their microservices efficiently. By doing this, it enables developers and companies to make better use of their resources, resulting in cost savings and improved system performance. This work helps improve Kubernetes' efficiency on cloud platforms, such as AWS.The demonstrated scheduler's quality is assessed by comparing its results with that of the default scheduler provided by kubernetes, which simply considers the network usage for communication between different microservices. In the dynamic world of cloud computing, this creative solution addresses the increasing complexity of microservices architectures and advances container orchestration platforms like Kubernetes. It also fits in with the changing landscape of cloud-native technologies.

Keywords: Microservice oriented Architecture, Kubernetes, Scheduling, Go Language, AWS EC2

1 Introduction

The transition from monolithic to microservices-based applications has become a standard of innovation in the dynamic field of software architecture. Large organizations have embraced this shift, which offers improved administration, cost-efficiency, and agility. Because of their loosely linked components, microservices provide simpler maintenance procedures and shorter development lifecycles, providing a scalable solution for changing system requirements. These benefits are further enhanced by the synergy between microservices and cloud deployment, which permits variable resource allocation in accordance with Quality of Service (QoS) standards.

However, optimizing resource usage and avoiding unnecessary costs in large-scale microservices deployments still poses challenges, necessitating innovative scheduling strategies.

This study explores a new scheduling strategy that takes microservice dependencies into account to effectively distribute containers among cluster nodes. The incorporation of microservices orchestration, as demonstrated by Kubernetes , is essential to guaranteeing smooth deployment, continuous monitoring, and elastic scaling. However, this orchestration improves system performance by addressing latency issues and optimizing resource use.This research Aiming to contribute innovative scheduling to magnify the microservice orcharstration in kubernetes. We also cover QoS classes, implicit QoS techniques, and advanced scheduling features like priority-based scheduling and preemption as we go deeper into the nuances of resource scheduling and relationship scenarios in Kubernetes. This detailed investigation lays the groundwork for a thorough comprehension of microservices orchestration and its essential function in the modern software environment.

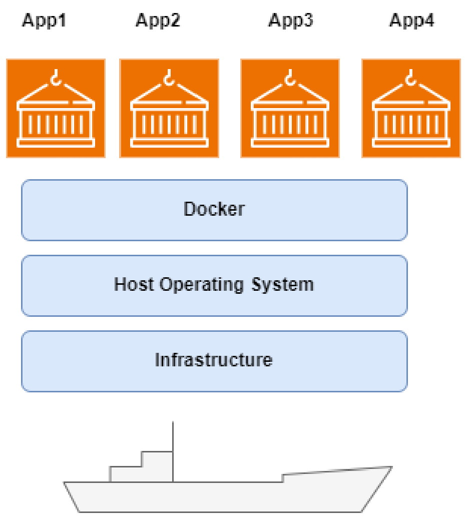


Figure 1: Evolution of Containerization

Containerization is a software development methodology that isolates applications and their dependencies into lightweight components known as containers.Containerization, which started with Docker in recent years, changed how we make softwares. It is similar to making image of apps and all their necessary library in boxes and make their containers. This made creating software faster and using computer resources better because these containers share the computer's main system. Docker made it easy for everyone to use containers with its simple tools. further, tools like Kubernetes helped developers to automate tasks like putting containers on servers and making sure they run well. Containers are popular because they make software portable, and in addition to that evolution of cloud computing make it scalable and fault tolerant.

The research makes a unique contribution in that it develops and implements a dependency-aware custom scheduler for microservices, a novel approach that optimizes deployment in a cloud environment by taking into account the complex dependencies among microservices. This novel approach reduces data transfer across microservices by choosing appropriate nodes based on dependencies, in contrast to conventional schedulers. The custom scheduler showed measurable benefits in response time, workload reduction, and improved Quality of Service (QoS) after extensive testing and assessments compared to the default Kubernetes scheduler. This work presents a novel development in microservices orchestration, providing a more effective and nimble way to deal with the problems caused by complex dependencies in modern cloud infrastructures.

1.1 Background

In the dynamic landscape of Kubernetes and microservices architecture, the need for dependency-aware scheduling arises from the complex interconnections between microservices within a given application. Due to the inherent interdependence between these microservices, traditional scheduling techniques frequently allocate resources inefficiently and run the risk of causing performance bottlenecks. for functioning of software, Microservices are depends on each others to share data. Scheduling microservices without taking these dependencies into consideration might lead to co-locating them on the different node, which can cause high usage of network usage over shared resources, higher network latency, and decreased efficiency (Pontarolli et al.; 2020). Understanding the importance of these dependencies becomes essential for efficiently scheduling microservices and maximizing a Kubernetes cluster's overall performance.

Companies are deploying softwares that consist of number of microservices made up of multiple interconnected microservices with Kubernetes, To handle this complex requirement, these applications require a scheduling method that accounts for in the complex relationships across microservices in addition to basic resource considerations. By reducing challenges related to incorrect management of microservices dependencies and allowing intelligent placement choices, dependency-aware scheduling arises as a tactical way to improve Kubernetes cluster performance (Vayghan et al.; 2021).

1.2 Motivation

The development and deployment of a dependency-aware custom scheduler in Kubernetes marks a significant breakthrough in cloud computing and container orchestration. Default Kubernetes schedulers tend to concentrate on load balancing and resource allocation, frequently ignoring the complex dependencies that exist between different microservices or applications in a distributed system. This error, particularly in complicated applications with interconnected components, can result in decreased performance, resource wastage, and increased latency.

The custom scheduler that considers dependencies plays a role in improving resource usage in Kubernetes clusters, which is the main objective. Through careful consideration of relationships across various microservices or application components, the scheduler is able to allocate workloads in a way that maximizes node efficiency throughout the cluster. In conjunction with optimizing the use of existing resources reduce network latency, enables you to fully utilize Kubernetes while carefully customizing it to the specific requirements according to your organization's software ecosystem. It therefore turns into a strategic investment in the scalability, robustness, and general efficiency of your applications.

1.3 Research question

Scheduling is an essential part of the orchestration process in Kubernetes and plays a major role in the lifecycle of containers. By distributing the containers across appropriate nodes in an efficient manner, it maximizes resource usage and keeps the workload evenly distributed.

- In what ways does the Dependency-Aware Scheduler, a pioneering microservices scheduling strategy that takes into account microservice dependencies, contribute to the optimization of scheduling and reduce network latency when integrated with Kubernetes in a cloud environment?

1.4 Objective

The primary objective of this study is to develop a custom Kubernetes scheduler that will optimize resource scheduling for pods by utilizing advanced relational scheduling of cluster nodes.

- Improve resource scheduling techniques and will touch the unresolved issue which is performance overhead by prioritize Quality of Service (QoS) principles, and implement effective resource quotas to strengthen Kubernetes basic scheduling features. Evaluate and implement to use advanced relational scheduling situations, with a focus on relationships between pods for scheduling constraints and by taking data transfer latency into consideration caching function which will help in reducing overally network usage over kubernetes cluster.

- The microservices that are present on the node and its CPU use rate are taken into account by the proposed scheduler as node selection criteria. The node that has the lowest CPU utilization rate is qualified for selection, and it is only confirmed if it offers the microservice needs.

- Create the kube-scheduler plugin especially for microservices deployment, for developer who do not want to concentrate more on scripting, they just need to import this scheduler and make necessary high level changes and use for their kubernetes deployment

- To build a cloud based solution which is more flexible and adaptable cluster environment, integrate priority-based preemption into the custom scheduler. To ensure optimal usage and efficiency in the AWS EC2-based Kubernetes cluster, this involves integrating various pod types, creating priority classes, and allowing the scheduler dynamically alter resource allocation depending on priorities.

1.5 Paper Structure

The paper is put in order as follows: in the 1 segment, discuss in brief about evolution of containerization, motivation and objective of research and in the Section 2,there is discussion of different research done in Kubernetes custom schedule. Section 3 describes the Methadology of proposed custom scheduler and their components and functioning of custom scheduler. In section 4 brief about design specification and tools which used develop the custom scheduler.Section 5 describes about implmentation steps.Section 6 showcases the evaluation of custom scheduler while comparing it with default scheduler and their output images and their comparative qualitative analysis and section 7 contains conclusion and future work which can done to help kubernetes community,

2 Literature Review

Upon reviewing several research publications, it became clear that the default scheduler was not best for dynamic workloads and should not be the primary option due to its design limitations, which means that some criteria don't apply to it. Despite some limitations in existing scheduler,small organizations still prefer to work on default scheduler for their workload managment along with taking advantage of cloud computing for non-critical application as explain in (Merkouche et al.; 2022) and very few research which actually addressd microservice dependency issues in kubernetes scheduling and which is one of the peak problem coming into the lights.

According to the survey of (Kratzke and Quint; 2017), over 83% of participants are using hybrid or multi-cloud environments, indicating a significant shift in the adoption of these models. The idea of a worldwide, all-encompassing mesh is not actually realized in practice, as corporations use different environments depending on their needs. One of the main players is Kubernetes, which is used by 14% of users for everything, over 20% for bare metal and virtual machines, and over 29% for a combination of VMs, bare metal, and Kubernetes. According to 38% of respondents, security and resource fragmentation is still the most important factor to take into account when using Kubernetes. Additionally, the survey shows that over 50% of respondents have rising faith in expert-built operators.Looking ahead, Kubernetes continues to define the future of container orchestration, and while high-level goals like bettering maintenance, monitoring, and automation remain, the growth of use cases is foreseen, especially in AI/ML and data platforms.

2.1 Kubernetes Survey

Before Kubernetes, organizations managed containerized apps and coordinated their deployment mostly through manual techniques and different technologies. It was difficult to handle many aspects of container orchestration, including scalability, load balancing, service discovery, and failover maunually. While some businesses packaged and distributed programs using containerization platforms like Docker, orchestrating these containers was frequently a laborious and tedious process.

Kubernetes is an open-source platform designed for managing services and workloads. Its provides features flexibility and portability make it highly adaptable for various environments. The main role of Kubernetes revolves around efficiently handling service and workload management. It excels in providing declarative configuration, allowing users to specify the desired state of their system, and robust automation support for seamless operations. The expansive ecosystem surrounding Kubernetes is marked by a rich array of supporting tools and services, contributing to its rapid growth and widespread adoption, as highlighted by Burns and Beda in 2019.

The evolution of containerization in process of DevOps has revolutionized the deployment process, offering a plenty of advantages to organizations. Initially, manual deployment methods and disparate tools posed challenges in managing containerized applications.A paradigm change occurred with the emergence of containerization, as demonstrated by technologies such as Docker. Software dependencies and configurations are isolated by containers, which encourages portability and facilitates easy transitions between various environments. By enabling developers to operate in lightweight, isolated

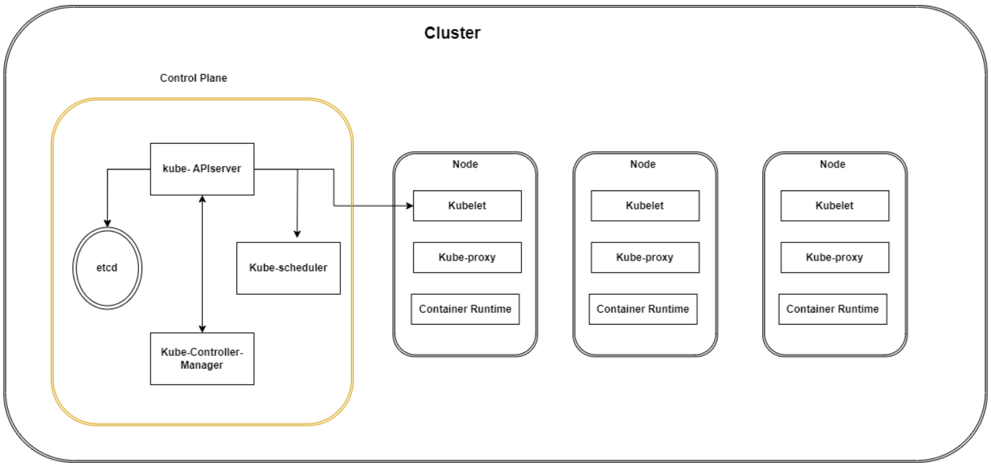


Figure 2: Kubernetes Architecture

situations, this development speeds up deployment and testing, lowers conflict, and increases productivity. Containerization also gives developers fine-grained control over the application environment, resulting in deployment settings that are reliable and predictable as stated by (Narasimhulu et al.; 2023). Efficient utilization of resources leads to cost-effectiveness, and scalability becomes easier.

2.2 Scheduling in Kubernetes

Kubernetes is an open-source container orchestration system which can be use for automating software deployment, scaling the application based on request,and management of applications containers efficiently.A pod is the smallest execution unit in Kubernetes which holds actual application code and their functionality. A pod encapsulates one or more applications.Pods are transient in nature. In case of microservices each microservice in termed as a pod, as it is collection of pods which connected with each other to run as full tier application.In kubernetes the pod specification and metadata are written in YAML file. The deployment file which we wrote basically contains the name of image and needed amount of resource (CPU, Memory) that is required by each container of microservice. On the other side the file also contains metadata like labels that helps to distinguish the pod. Executing the the script though kube-API will create the pod and send it to multiple nodes as per describe in the pod and deployment specification file.

Kubernetes follows master-slave architecture.The master components manage the overall state of the cluster. This contains accepting client requests, scheduling containers and running control loops to instruct to perform operations on the actual cluster state towards the desired cluster state.and worker nodes are responsible for hosting different containers in terms of pods. Each node can host multiple pods.

Master node is also known as control plane which contains the components like kube-APIserver,etcd,kube-controller-manager and kube-scheduler. On the Master node, the Kube Scheduler operates as a process daemon. It keeps a watch on each worker node's condition and resource usage. When a user requests the creation of a POD, the scheduler uses a filtering and scoring technique to identify a worker node that best matches the resource specification. The API server is then instructed by the scheduler to bind the POD to the intended node.

This section will describe how the scheduler operates. To decide which available node is best suited to host a pod, the scheduler will first filter the nodes and then assign a score to each of them.

Kube scheduler filtering technique is used to opt out worker nodes that contains the fundamental components suitable for a POD placement.Examples of Filtering predicates include:

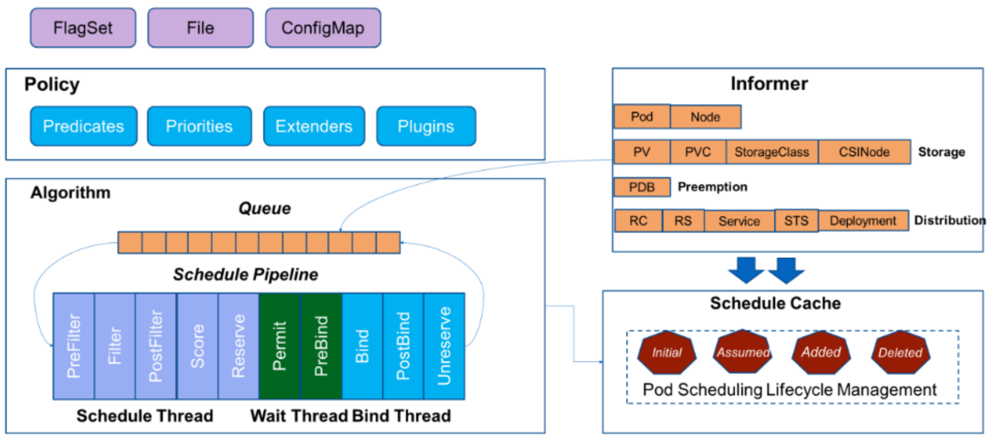


Figure 3: Scheduling Framework

For filtering of nodes it uses General predicates (e.g., PodFitsResourcesPodFitsNodeSelector Storage related predicates (e.g., NoDiskConflictMaxCSIVolumeCount Compute node related predicates (e.g., PodToleratesNodeTaint) Runtime related predicatese.g., CheckNodeCondition , CheckNodeMemoryPressure,

2.2.1 Working of Default Scheduler

- Kube scheduler scoring phase is use to rank the nodes that left out after filtering phase and will be eligible for pod placement.Based on the scoring priorities, the scheduler gives each of the surviving nodes a score. Ultimately, the node with the greatest ranking receives the Pod assignment from kube-scheduler.
- Scoring Priorities include SelectorSpreadPriority, InterPodAffinityPriority, LeastRequestedPriority, NodeAffinityPriority, BalancedResourceAllocate, etc. For each priority passed, the corresponding node will be given a score between 1 to 10, based on the weight assigned by the user in the deployment script, Worker node scores are calculated as the total of all priority scores.
- On other hand default scheduler determines all the nodes that it knows they exist and make sure they are healthy. The scheduler keeps an eye on predicate tests to filter out nodes that are not suitable. The rest of the nodes form a group of possible nodes.
- Priority tests are performed by the scheduler on the potential nodes. The candidates are ranked highest to lowest based on their scores. The node with the highest score

is selected at this moment. However, it is possible for many nodes to have the same score at times.

- The nodes are added to the final list if their scores are same. For the purpose of evenly distributing the load among the machines, the Kubernetes Scheduler uses a round-robin method to choose the winning node.

2.3 Custom based scheduling

When Kubernetes first started out, it mostly depended on a default scheduler that made choices based on the availability of resources, which occasionally resulted in less-than-ideal deployment choices. The Kubernetes community realized the value of custom schedulers when the demand for more specialized and effective scheduling techniques increased.

In order to improve resource usage in cluster situations, the (Ning; 2023) suggests a modified Kubernetes scheduling technique. The motivation is the increasing acceptance of container-based methods, especially when implemented in large-scale applications with Kubernetes serving as the container orchestrator. The default Kubernetes scheduling algorithm is found to have flaws by the authors, especially when it comes to establishing scoring guidelines for node selection and assigning suitable resource request values for pods. In order to tackle these problems, the suggested approach uses a custom scoring rule that takes into account the Euclidean distance between the target and optimal resource values, and it dynamically modifies resource requests based on past pod resource utilization data. In order to maximize efficiency, the authors stress the necessity for improvements and draw attention to the widespread issue of inefficient resource usage in Kubernetes clusters. In general, the suggested algorithm is positioned as an enhancement to the default.Kubernetes scheduler, emphasizing better load balancing and resource utilization among cluster nodes.

Low-latency applications thrive in the quickly evolving Fog and Edge computing landscape, but scheduling container-based microservices becomes a significant challenge due to the dynamic and often unstable nature of Edge infrastructure. While container orchestration technologies such as Kubernetes have made it easier for traditional Cloud data centers to deploy applications, typical.scheduling methodologies are not up to the specific communication requirements of microservices, especially when latency is an issue. This literature review highlights the inadequacies of existing container orchestration systems and emphasizes the need for better scheduling methods. The (Marchese and Tomarchio; 2022)suggested method adds a custom scoring plugin to the Kubernetes scheduler and uses the TOSCA language to record traffic history, network latency metrics, and microservices communication requirements.The objective is to optimize Pod placement taking into account the application topology, communication patterns, and real-time network conditions in order to minimize end-to-end latencies for increased Quality of Service (QoS). The evaluation highlights how crucial communication-aware scheduling solutions are for enhancing the deployment of microservices in Cloud-to-Edge continuums, particularly for novel applications like as process management, augmented reality, data processing, and smart automobiles.

2.4 Need of Custom Scheduler

This paper discusses the role that schedulers play in Kubernetes, a popular container orchestration platform for cloud application management. The integrated Kubernetes scheduler has been proven to be poor, given its speed, and can occasionally lead to bad service placements or even deployment failures. The authors introduce Boreas, a novel Kubernetes scheduler designed to simultaneously evaluate bursts of deployment requests and optimal service container placements with deployment limits. The study emphasizes the importance of efficient scheduling in cloud environments and introduces the idea of using Zephyrus2, a configuration optimizer, to achieve provably optimal service deployment. Through a number of simulated experiments, including a more complex affinity test and a simpler deployment scenario, the (Lebesbye et al.; 2021) demonstrate that Boreas outperforms the default scheduler in terms of successful installations and resource efficiency.The study emphasizes the need for deployment approaches that are more accurate and resource-efficient in certain scenarios, contributing to the development of research on Kubernetes schedulers.

The aim of this paper is to address resource fluctuation and uneven job distribution by presenting a novel hybrid shared-state scheduling mechanism for Kubernetes (Ungureanu et al.; 2019). Each of the centralized, distributed, and hybrid scheduling models that are currently in use is discussed along with its shortcomings. The proposed architecture combines the advantages of centralized and distributed approaches by using master-state agents for cluster state synchronization and decentralized scheduling agents for local optimizations. This approach was motivated by issues that were observed in an operational Kubernetes cluster, issues that are challenging for conventional centralized schedulers to handle. High-availability issues, collocation interference, and priority preemption are some of these issues. The study highlights the necessity of a synchronized cluster state and the scheduling correction function in order to optimize resource consumption.

2.5 Summary

Upon performing a thorough review of the literature, it is clear that numerous researchers have concentrated on scheduling technique optimization. But because microservices are highly interdependent, there is still a significant problem with their deployment. Application performance as a whole can be negatively impacted by improper scheduling, which can also result in higher latency, poor resource consumption, network bottlenecks, inconsistent scaling decisions, and data transfer overhead.

In case of mircoservices deployment it is necessary to take care of Using affinity and anti-affinity rules that are laid out in the pod design, pods can establish mutual exclusion or affinity. These rules specify how pods should be scheduled according to node properties or other pod labels. They are specified in the affinity field. Pod affinity, for example, can be used to make sure that pods on the same node share common features, such a label. Conversely, node affinity enables constraints or preferences based on node attributes by defining the circumstances under which a pod should be scheduled on specified nodes. Within a Kubernetes cluster, these affinity rules offer a reliable method for fine-tuning pod placement, meeting deployment limitations, and optimizing resource allocation.

Author	Workload Target	Main Idea	Limitations
(Ning; 2023)	Heterogeneous	Optimizing Resource Utilization	Euclidean Distance Bias Concerns for metrics optimization
(Lai et al.; 2023)	IOT	Enhance network bandwidth	Challenges in Edge Heterogeneity
(Bac and Kim; 2023)	Cloud	Intent-based Scheduling for Kubernetes Workloads	Manual implementation of complicated configurations, hindering enterprise service management.
(Marchese and Tomarchio; 2022)	Edge	Enhancing Kubernetes for Edge Microservices	lack of consideration for communication aspects, network conditions, and historical information related to resource utilization and application performance.
(Lebesbye et al.; 2021)	Cloud	Boreas: Optimizing Kubernetes Placement Constraints	slower than the default scheduler.
(Ungureanu et al.; 2019)	Hybrid	addresses issues of uneven workload distribution, resource fluctuations, and problems with existing centralized and distributed scheduling frameworks.	Conceptual Hybrid Scheduler, lacks implementation
(Shen et al.; 2023)	Edge-Cloud	Decentralized Edge Scheduling for Kubernetes for overall system performance enhancement	Focus on scheduling optimization, less on overall system design

3 Methodology

3.1 Components of Kubernetes and Proposed Design

- **API server** This component is responsible for the communication with Kubelet on the worker nodes. also provide functionality of auth-entication and check for the authorization of the requestor.

- **Etcd** This is the key-value store for the system's critical state. The Kubernetes cluster's status snapshots are taken using etcd.

- **Controller manager** Responsible for using the apiserver to monitor the cluster's shared state and for making modifications to try to bring the current state closer to the intended state. The namespace controller, endpoints controller, replication controller, and service accounts controller are a few of the controllers that come with Kubernetes.

- **Kube-Scheduler** The Kube Scheduler is in the role of choosing the worker node for a POD and setting up the POD on the intended node with the resources that it requires.

- **Namespaces** When multiple teams or group of developers share a Kubernetes cluster, namespaces can be useful for organizing the cluster into virtual subclusters. Within a cluster, any number of namespaces can be maintained, each logically isolated from the others yet able to interact with one another.

- **Monitoring Tools** The Prometheus application will be made available for external access by the Prometheus Kubernetes Service. The Grafana application will be made available by the Prometheus Kubernetes Service, enabling external users to view the Grafana dashboards.

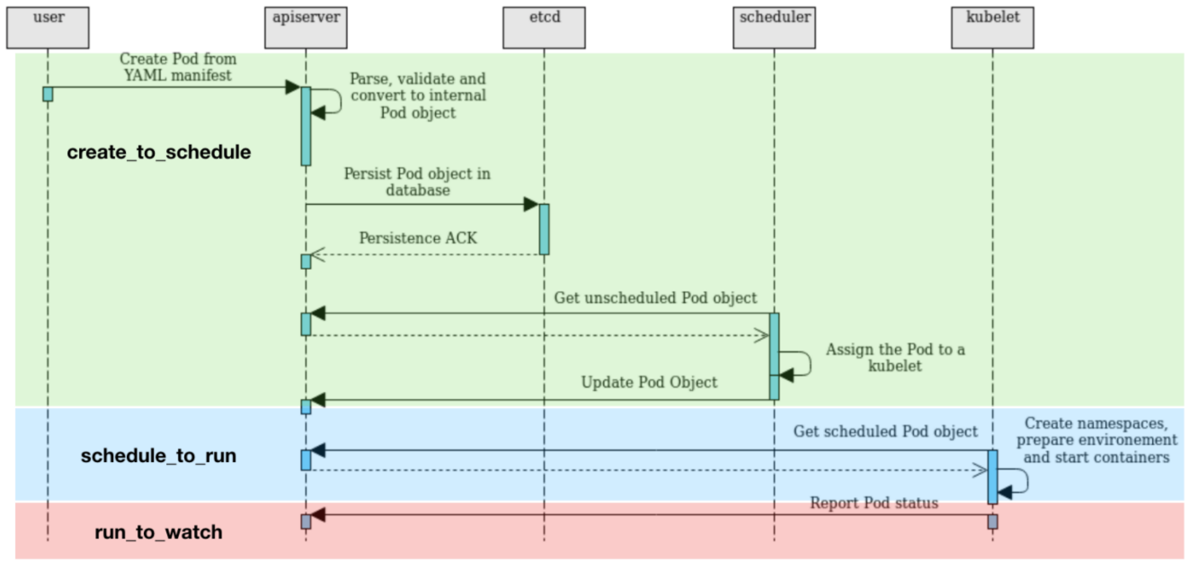

user
apiserver
etcd
scheduler
kubelet
Create Pod from YAML manifest
Parse, validate and convert to internal Pod object
create_to_schedule
Persist Pod object in database
Persistence ACK
Get unscheduled Pod object
Assign the Pod to a kubelet
Update Pod Object
schedule_to_run
Get scheduled Pod object
Create namespaces, prepare environement and start containers
Report Pod status
run_to_watch

Figure 4: Pod Scheduling Phases

3.2 Custom scheduler's objective

The custom scheduler that proposing designed aims to performance enhancement for microservices. The custom scheduler is more focused on the amount of network and cpu that the nodes are using during their communication within the kubernetes cluster and also check for dependency, even if the default scheduler also accounts for CPU and network requests upto certain extent but it fails in case of having multiple microservices placed within the kubernetes cluster.Yet, because of their strongly interdependency, a significant problem with microservices deployment still exists. Ineffective resource management, network bottlenecks, uneven scaling choices, higher latency, and data transfer overhead are just a few of the adverse impacts of improper scheduling that can lower the performance of an application as a whole. Comparing the behavior of the default and custom schedulers will clearly shows difference in final results.

When we consider the deployment of microservices on kuberentes cluster , it depends on factors like available nodes, dependencies, resource constraints, and memory availability. For the purpose of determining which nodes provides the most resources and efficiency will be pick by scheduler, Node exporter will help in this situation to collect the real time metrics and send it to the master node and finally it takes best decision of picking right node for the deployment of microservice. In case of proposed custom scheduler, the Bash script initially perform a number of steps to optimize the scheduling of a service within a Kubernetes environment. It initially identify_available_nodes and get a list of available node names using the Kubernetes command '"kubectl get nodes". then, it assesses dependencies on a specified location-constrained node through "check_dependencies_running_on_node," crucial for determining whether the dependent microservice assigned to the designated node. if they are not dependent then proposed scheduler go for the "selecting the node with most resources available function, intelligently opting for the node with optimal resources, enhancing deployment efficiency. The script efficiently navigates through these steps, ensuring an optimized deployment strategy within the Kubernetes cluster.

3.3 Functioning of the proposed custom scheduler

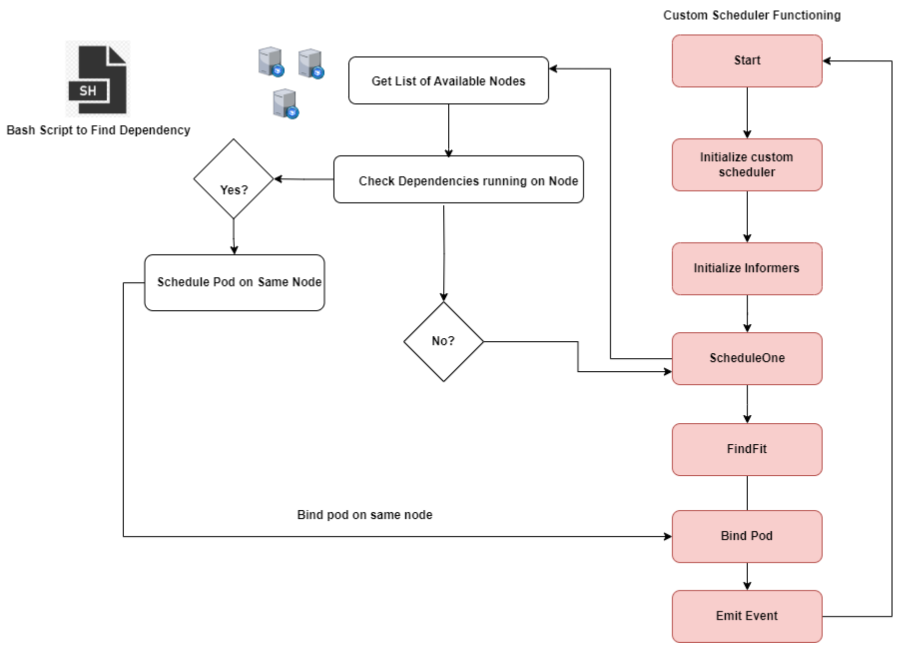

Custom Scheduler Functioning
SH
Bash Script to Find Dependency
Get List of Available Nodes
Check Dependencies running on Node
Yes?
Schedule Pod on Same Node
No?
Bind pod on same node
Start
Initialize custom scheduler
Initialize Informers
ScheduleOne
FindFit
Bind Pod
Emit Event

Figure 5: Architecture Diagram

The custom scheduler written in Go langauge start by setting up the scheduler component. For accessing the scheduler components, kubernetes provided list of kubernetes API to directly communicate with the components and get the result instead of going by imperative way.After that scheduler will initialize the informers which helps in monitoring the unscheduled nodes and pods, then it will actually start scheduling process by taking one pod in queue, go to the bash script and check for dependency if not then find fit node that is identifying a suitable node for pod deployment and bind the pod to that node,on other hand if any pod contains depdency with any other pod on any node and if that node is having enough memory it will directly schedule the pod on the same node.

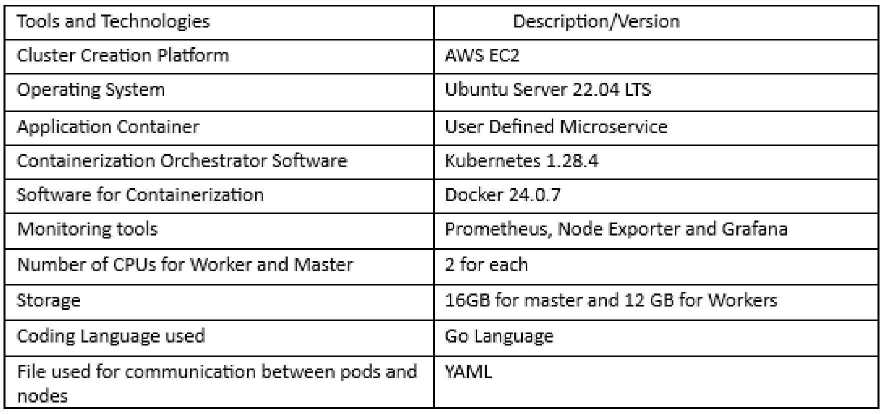
Bash script written aims to replicate key functionalities of a Kubernetes scheduler for service deployment.Using the **identify_available_nodes()** function, the script demonstrates the process of listing down available nodes within the Kubernetes cluster in the first phase. Then, using the **check_dependencies_running_on_node** function, it determines whether dependencies are available on a given node and verifies that the necessary microservice—represented by the "hello-payment" which s functioning on that node. In order to determine whether the another microservice may be scheduled on the same node without going against dependency limitations, this step is essential. If dependencies are not satisfied, the script moves back to the custom scheduler function. All things considered, custom scheduler do better job than default scheduler.

For network optimization custom script has caching function which takes into consideration when two microservices share same amount of data, in this case microservices which is having dependencies share a common redis cache, so whenever microservice want to take some data from other microservice it will firstly check in the cache, if it is there data trasnfer rate and ultimately bandwidth usage will be less. This approach will drastically reduce network usage in kubernetes.

4 Design Specification

The following resources were utilized to create and test the custom scheduler, decided to use AWS cloud services in order to set up a Kubernetes cluster formation so that it may avail the benefits of cloud services Elastic Compute.For the purpose of this research experiment,

cluster of Kubernetes created by utilizing the Kubeadm-cluster to create multinode cluster. For forming cluster Kubernetes and Docker required. Cluster contains master node, which is having 2 CPUs and the 16GB storage, instance type is t3.large, as master node require more storage and computation power than worker nodes, on other hand worker node is hosted on t2.medium,Bash Scripting with YAML was chosen to develop the scheduler because it increase the flexibility, and adaptability with low runtime overhead, Also, Bash scripting offers a simple and user-friendly method for integrating custom scheduling logic. It is by far the most common language used for putting microservices into action.Use of a node exporter done, which has to be installed in the slave nodes as well master node to export metrics to the prometheus, After that, For monitoring purpose prometheus installed on master node along with the grafana for getting effective visulization of node health. For installation of prometheus and grafana helm used. With Helm, you can create, package, configure, and launch Kubernetes applications automatically by grouping all of your configuration files into a single, reusable package.Event monitoring and alerting can be done with Prometheus, it is a free tool for analysis. Also Users can view their data using charts and graphs that are combined into a single dashboard using Grafana, an open source interactive data-visualization platform created by Grafana Labs.



Tools and Technologies	Description/Version
Cluster Creation Platform	AWS EC2
Operating System	Ubuntu Server 22.04 LTS
Application Container	User Defined Microservice
Containerization Orchestrator Software	Kubernetes 1.28.4
Software for Containerization	Docker 24.0.7
Monitoring tools	Prometheus, Node Exporter and Grafana
Number of CPUs for Worker and Master	2 for each
Storage	16GB for master and 12 GB for Workers
Coding Language used	Go Language
File used for communication between pods and nodes	YAML

4.1 Proposed Kubernetes Scheduler-Architecture

When it comes to choosing where to best arrange pods on available nodes in the cluster, Kubernetes' scheduler is essential. User-defined logic for node selection is incorporated into the custom scheduler, which expands on this functionality. This is demonstrated in the corresponding Bash script. To intelligently determine where to deploy a pod, the scheduler takes into account resource limits, checks for specific dependencies, and evaluates factors like node availability. With this customization, users can adjust the scheduling process to meet their own needs, making sure that resources are used effectively placed and that dependencies are followed strictly. All things considered, the customized scheduler improves Kubernetes by offering a customizable and adaptable method of pod placement, which helps with workload orchestration and cluster optimization and making the system reliable and fault tolerant even for heavy workload.

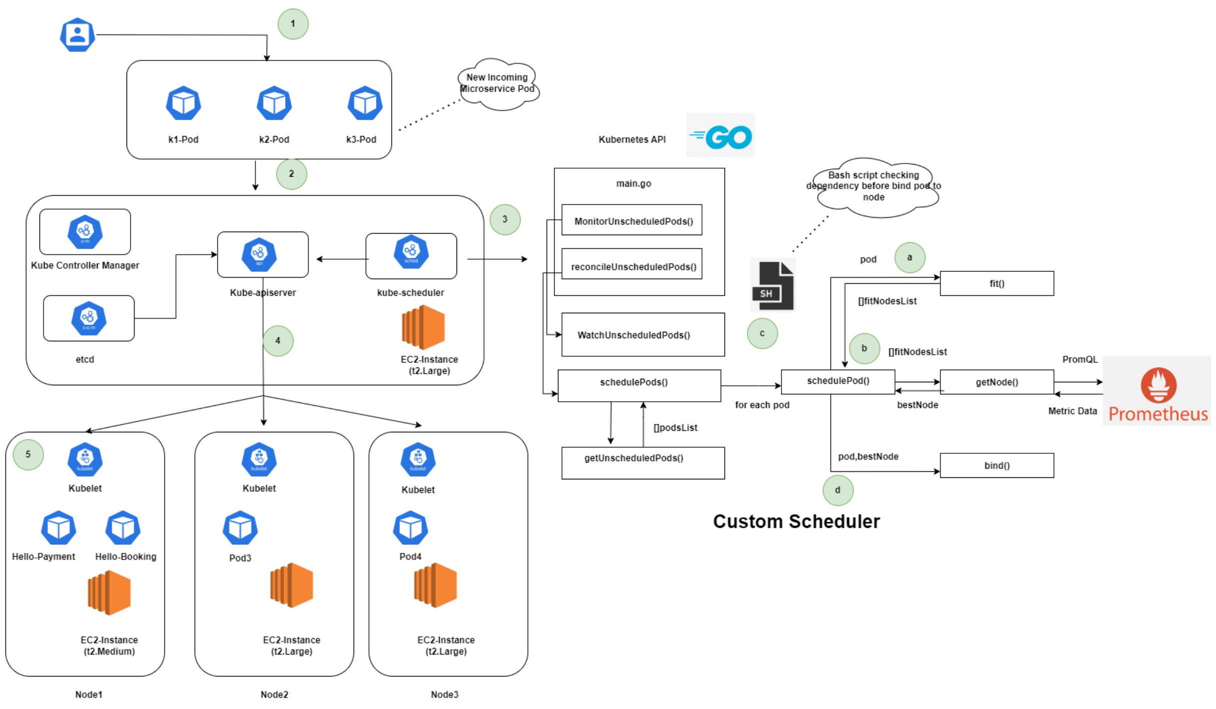


Figure 6: Custom Scheduler Flow Diagram

Figure 6 displays the custom scheduler's elaborate architecture diagram, and thereby explaining the steps which are involved in this process.

- **Step1** First stage consist of formation of the Kubernetes cluster on AWS cloud .I utilized an EC2 instance to achieve this objective. The entire network consists of two slave nodes and one worker node. In addition, as scripting language for pod definition and deployment we need different Yaml files.

- **Step2** Docker, Kubernetes, and Kubeadm were downloaded with the use of a shell script that included all the instructions required to download the technologies correctly. Both the worker nodes and each of the master nodes require to execute set of instruction in this step.

- **Step3** To get BestNode using custom scheduler, there is requirement of installing observability tool like Prometheus client , which expose important information like node availability and resource use, bandwidth usage and data transfer and then integrating Prometheus with Grafana, and Node Exporter will help in gathering and analyzing data for our custom scheduler effectively. With help of Node Exporter on each Kubernetes node system-level metrics can be obtained.

- **step4** During the execution of the custom scheduler which was written in go langauge, Filtering and binding methods implemented in the scheduling process using and using Bash scripting language will allow us to obtain the the best node to fit depndent microservices together in same node.Assisting developer place the pod in particular node rather than placing randomly to reduce data transfer latency between pods and hence,will be able to get the best node to achieve desired objective.

5 Implementation

Within the context of Kubernetes orchestration, the data transfer rate is crucial in determining how a dependency-aware scheduler performs. This scheduler is specifically meant to improve the coordination of connected microservices; it relies on placing pods in the best possible positions to facilitate effective communication. The impact of data transfer rate also extends to the critical factor of network latency. Microservices are highly dependent on smooth data transmission, hence the scheduler's job is to reduce latency through placing related microservices on the same nodes in a planned way. In along with speeding up data transfer across the network, this purposeful co-location guarantees low-latency communication between these interdependent components, creating an ecosystem that promotes low-latency communication. The scheduling script is inspired by Hungarian Algorithm, which is a probabilistic optimization technique that can be used for microservices scheduling in a dependency-based system. By determining the best way to distribute microservices among nodes while minimizing total cost or optimizing efficiency, it solves the assignment problem. The algorithm iteratively decreases rows and columns, determines assignments, and augments paths until an optimal solution is obtained, representing the scheduling problem as a cost matrix. The Hungarian Algorithm can be used in a dependency-based scheduler to distribute microservices to nodes intelligently while taking into account their dependencies and reducing data transmission costs. This optimization helps to achieve a balanced and effective resource allocation, which improves system performance and lowers latency.

5.1 Kubernetes Cluster Formation

This section will discuss the EC2 instance creation with formation of the Kubernetes cluster on AWS. we have created one master and two slave nodes. Since the master nodes have the custom scheduler, Control Panel, and APIserver, it requires more computing workthat can be addressed by go with T2.Large size instance and having Ubuntu 22.04 LTS as the operating system of choice. Two T2.Medium slave nodes, each with two CPUs, have been created. For cluster formation commands needs to be executed on each instance.containerd used which is a container runtime that handles every component of the container lifecycle, such as execution, storage, and image transfer. It facilitates the creation, execution, and termination of containers by serving as a bridge between the underlying operating system and the container orchestration platform, such as Kubernetes. for Kubernetes ecosystem kubelet, kubeadm, and kubectl are important components of kubernetes to run orchestration of containers sommothly. The command-line tool for communicating with Kubernetes clusters is kubectl, while kubelet is in responsible of ensuring that containers are operating within a pod. Kubeadm also helps with Kubernetes cluster configuration. Container Network Interface also known as kubernetes-cni, gives containers in a Kubernetes cluster networking capabilities. Together, these elements support Kubernetes cluster deployment, administration, and interaction. Also Docker is a platform that facilitates the development, transportation, and execution of applications within containers. Docker is a well-liked option for building and packaging containers, even though Kubernetes may operate with a variety of container runtimes

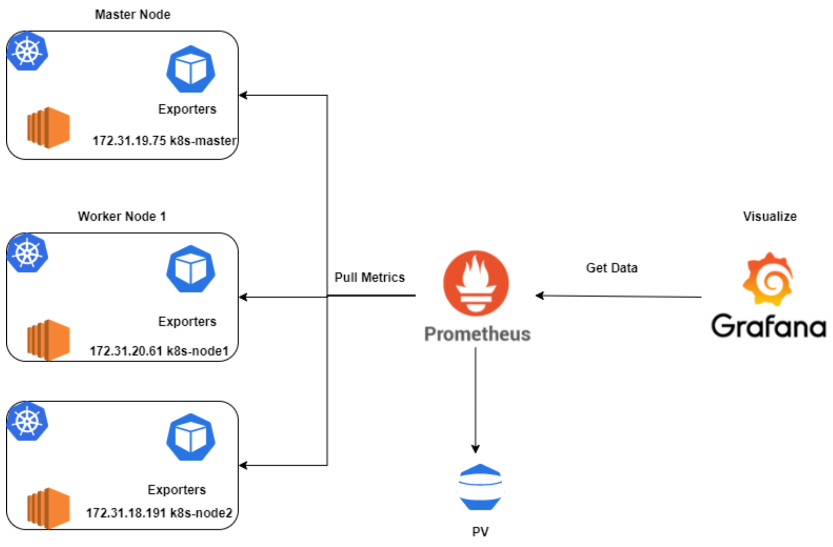


Figure 7: Architecture For observability tools

5.2 Proposed scheduling algorithm working

Using relevant Kubernetes API calls included within the scheduler script, our Go-based custom scheduler communicates with the Kubernetes control plane during the start-up phase. The scheduler then sets up informers to watch nodes and pods in the Kubernetes cluster closely to check available nodes and unscheduled pods. The ScheduleOne stage begins when a new pod needs to be scheduled. Here, the scheduler carefully chooses which node is most suited for deployment. In parallel, it runs a customized bash script that is meant to perform dependencies across microservices. Using methods such as "identify available nodes()" and "check dependencies running on node()," the script generates a complete list of nodes that are available and examines any dependencies that are present.After that, the scheduler carefully assesses each of these nodes to determine which one is the best fit for the pod based on variables like resource availability and script outcome. The scheduler broadcasts its decision by binding the pod to the selected node and emitting an event upon identification. Most importantly, a caching mechanism is integrated to enhance network efficiency through facilitating data exchange among microservices, hence considerably minimizing bandwidth usage within the Kubernetes framework. The complete approach guarantees effective deployment while reducing network overhead for improved system performance.

5.3 Monitoring tool-Prometheus tool

Monitoring in the kubernetes is one of essential part that every developer need to take into consideration.Prometheus with Node Exporter captures detailed node-level metrics, which improves monitoring capabilities. Node Exporter provides various system-level metrics from a host computer available. By integrating Node Exporter with Prometheus, detailed information regarding the resource usage, network activity, and general system health of each node can be gathered. It would be possible to improve the custom scheduler script to incorporate the settings required for each node's by collecting real-time important metric.

Prometheus obtains insights into the underlying infrastructure when it is set up to pull metrics from Node Exporter endpoints. This allows for an in-depth knowledge of node-specific performance characteristics. Operators can then have an in-depth picture of the cluster and individual nodes by adding visualizations of these additional node-level metrics to their Grafana dashboards. In addition to assisting in identifying cluster-wide patterns, this integrated monitoring system enables detailed analysis at the node level, enabling operators to take accurate choices about workload distribution, resource optimization, and cluster management in general. overall, the integration of Prometheus, Grafana, and Node Exporter results in an effective observability stack that improves the monitoring capabilities of the custom scheduler for both high-level and low-level insights into the Kubernetes cluster.

Here are few components of prometheus and grafana that can be use to obtain metrics details:

5.3.1 Components of Monitoring Tools

- **Node Exporter:** A metrics exporter for machine level metrics such as CPU, memory, disk, and network usage on each node.
- **PromQL:** The query language used to retrieve and analyze data from Prometheus.
- **Dashboards:** Customizable visualizations that display metrics in user understandable graphs and allow users to monitor and analyze data.
- **Data Sources:** Connections to databases or monitoring systems, with Prometheus being a common data source for Kubernetes monitoring.

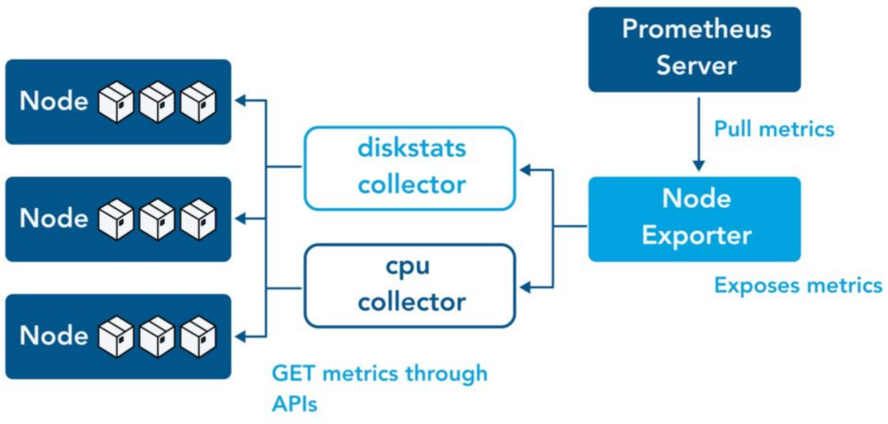


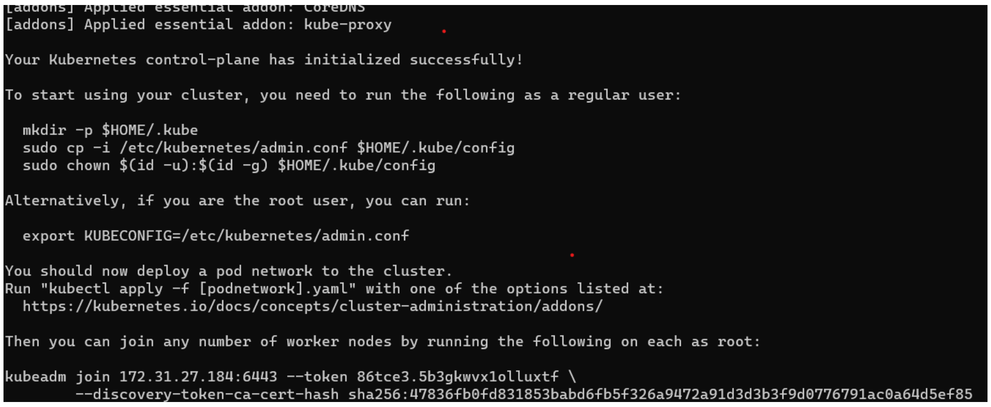
Figure 8: Monitoring Tools used for Custom Scheduler

5.4 Distinctive Features Compared to Default Kubernetes Schedulers

- **Dependency-Aware Node Selection:** Intelligent node selection is made possible by the custom scheduler's efficient knowledge and utilization of microservices dependencies. The custom Kubernetes scheduler optimizes resource allocation based on complex relationships, in contrast to default schedulers, which do not have this level of dependence awareness.
- **Dynamic Resource Allocation and Optimization:** In a dynamic microservices context, the custom scheduler ensures optimal consumption and efficiency by dynamically adjusting resources based on evolving dependencies and workload, in contrast to default Kubernetes schedulers that frequently rely on static resource allocation.
- **Optimized Data Transfer and Bandwidth Management:** By incorporating advanced data transmission optimization strategies like differential updates and compression, the custom scheduler surpasses default schedulers. This reduces the amount of bandwidth used, which is important in situations where there are complex microservice dependencies.
- **Cloud Integration and Elasticity:** Utilizing the elasticity of resources in services like AWS EC2, the custom scheduler integrates with cloud architecture with ease. This guarantees effective scalability and flexibility to changing workloads, something that default Kubernetes schedulers might not be as good at naturally.

6 Evaluation

The experiment was carried out utilizing on AWS EC2 services, and Ubuntu Server 22.04 to build the Kubernetes cluster. In order to assess the performance of the custom scheduler against the Kubernetes default scheduler, we install two pods. These are the names of these pods: Payment and booking apps on the custom scheduler and the default scheduler with two different namespaces

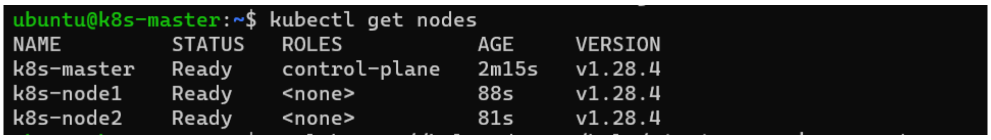


```
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

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

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 172.31.27.184:6443 --token 86tce3.5b3gkwvx1olluxtf \
        --discovery-token-ca-cert-hash sha256:47836fb0fd831853babd6fb5f326a9472a91d3d3b3f9d0776791ac0a64d5ef85
```

Figure 9: Cluster Formation

A Kubernetes cluster's control plane can be initialized using the kubeadm init command. The machine that is supposed to serve as the master node is usually the one running it. The scheduler, controller manager, and API server belong to the main Kubernetes components that are configured with default values by using this command. To provide secure communication within the cluster, it also creates the certificates and authentication tokens needed. Users can start Kubernetes operations more easily by using the kubeadm init command, which offers a standardized and efficient method of bootstrapping a cluster.

In Kubernetes, the kubeadm token create –print-join-command command generates a bootstrap token and outputs the matching kubeadm join command right away. This command makes it easier to add worker nodes to a Kubernetes cluster. It is run on the master node after cluster initialization with kubeadm init. The created token and the API server address are included in the printed join command, which simplifies the node joining process for administrators which then pass to slave node for forming the cluster.



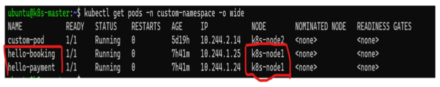
```
ubuntu@k8s-master:~$ kubectl get nodes
NAME         STATUS   ROLES           AGE     VERSION
k8s-master   Ready    control-plane   2m15s   v1.28.4
k8s-node1    Ready    <none>          88s     v1.28.4
k8s-node2    Ready    <none>          81s     v1.28.4
```

Figure 10: List Of Nodes

Kubectl get nodes is the command we need to use to see if the cluster has formed.

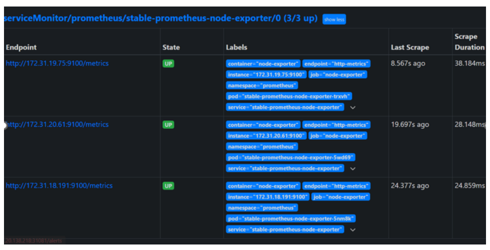
6.1 Configuring Prometheus and Grafana to Work with Node Exporter on a Kubernetes Cluster

Helm charts are used to deploy Prometheus, a preferred Kubernetes monitoring and alerting solution, using the commands provided. To access pre-configured charts, first install the Prometheus Community Helm repository using the helm repo add command. The local repository cache is refreshed by the next helm repo update command. To examine available charts, particularly those from the Prometheus Community repository, use the helm search repo command. For the Prometheus deployment, a specific namespace called ”prometheus” is created using the kubectl create namespace command.Prometheus is deployed into the ”prometheus” namespace by the helm install command, which makes use of the ”kube-prometheus-stack” chart from the Prometheus Community repository. This Helm chart provides a standardized and easy method for configuring Prometheus in a Kubernetes environment by encapsulating the full monitoring stack.



```
ubuntu@k8s-master:~$ kubectl get pods -n custom-namespace -o wide
NAME            READY   STATUS    RESTARTS   AGE     IP            NODE        NOMINATED NODE   READINESS GATES
custom-pod      1/1     Running   0          5d19h   10.244.2.14   k8s-node2   <none>           <none>
hello-booking   1/1     Running   0          7h41m   10.244.1.25   k8s-node1   <none>           <none>
hello-payment   1/1     Running   0          7h41m   10.244.1.24   k8s-node1   <none>           <none>
```

(a) Scheduled Pods on Same Node



serviceMonitor/prometheus/stable-prometheus-node-exporter/0 (3/3 up)

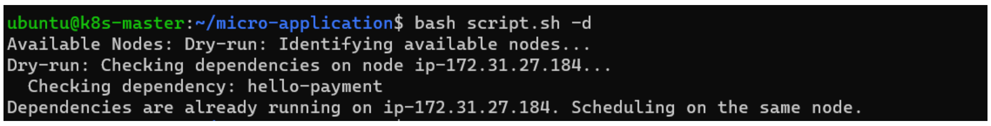
Endpoint	State	Labels	Last Scrape	Scrape Duration
http://172.31.19.75:9100/metrics	UP	container="node-exporter" endpoint="http-metrics" instance="172.31.19.75:9100" job="node-exporter" namespace="prometheus" pod="stable-prometheus-node-exporter-trxvh" service="stable-prometheus-node-exporter"	8.567s ago	38.184ms
http://172.31.20.61:9100/metrics	UP	container="node-exporter" endpoint="http-metrics" instance="172.31.20.61:9100" job="node-exporter" namespace="prometheus" pod="stable-prometheus-node-exporter-5wd69" service="stable-prometheus-node-exporter"	19.697s ago	28.148ms
http://172.31.18.191:9100/metrics	UP	container="node-exporter" endpoint="http-metrics" instance="172.31.18.191:9100" job="node-exporter" namespace="prometheus" pod="stable-prometheus-node-exporter-5nm8k" service="stable-prometheus-node-exporter"	24.377s ago	24.859ms

(b) Prometheus Dashboard

6.2 Conducting tests with the Custom Scheduler

There are two parts in evaluation which basically focuses on microservices deployment and network optimization, First objective is to place the dependent microservices in the same node and second objective is while placing the pods on the node, network consumption and data transfer rate should be low as compared to default scheduler.

- **Step 1:** Deploy the microservices call Hello booking on one node and while deployment of dependent microservices called Hello Payment, it should go on the same node using our custom scheduler as shown in figure



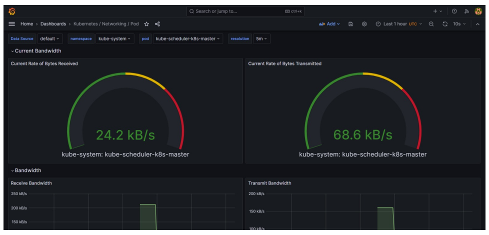
```
ubuntu@k8s-master:~/micro-application$ bash script.sh -d
Available Nodes: Dry-run: Identifying available nodes...
Dry-run: Checking dependencies on node ip-172.31.27.184...
  Checking dependency: hello-payment
Dependencies are already running on ip-172.31.27.184. Scheduling on the same node.
```

Figure 11: Scheduler Script

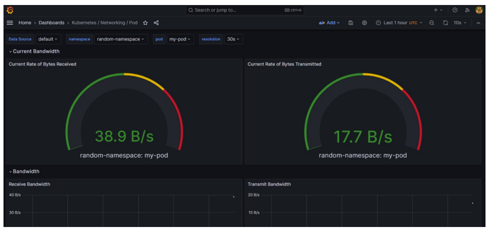
- **Step 2:** For demonstration of optimization of network utilization will deploy the microservice using default scheduler as well as using custom scheduler and assess the performance difference

6.3 Results and Discussion

To assess the efficiency of the custom scheduler alongside to the default scheduler's performance, two nodes along with master node deployed and formed a cluster. The Booking microservice deployed on one random node and second dependent microservice deployed and checked whether it will go to same node and their network utilization is less as compared to default scheduler network utilization. When determining the bestNode,

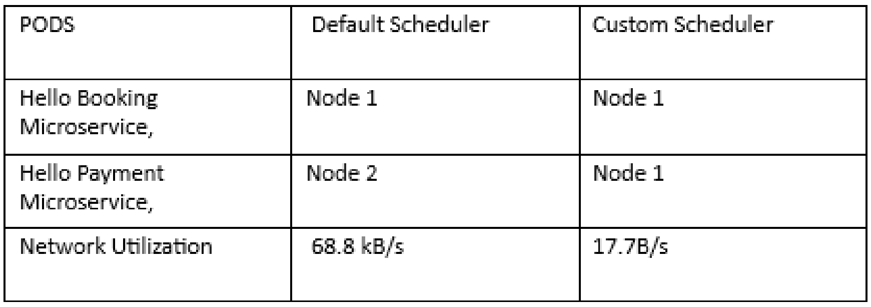

Current Bandwidth
Current Rate of Bytes Received
Current Rate of Bytes Transmitted
24.2 kB/s
68.6 kB/s
kube-system: kube-scheduler-k8s-master
kube-system: kube-scheduler-k8s-master
Bandwidth
Receive Bandwidth
Transmit Bandwidth

(a) Default Scheduler



(b) Custom Scheduler

the proposed custom scheduler will make use of the node exporter tool in conjunction with the Prometheus and grafana tool. The combination of node exporter, Prometheus, and Grafana enabled the evaluation process, which included dependency-aware decision-making, intelligent placement of microservices, and a thorough examination of network usage. The main objective was to show that the proposed custom scheduler, offered with these monitoring tools, could satisfy dependency requirements and surpass the performance of the default scheduler by strategically placing microservices for enhanced effectiveness and decreased network overhead in the Kubernetes cluster.



PODS	Default Scheduler	Custom Scheduler
Hello Booking Microservice,	Node 1	Node 1
Hello Payment Microservice,	Node 2	Node 1
Network Utilization	68.8 kB/s	17.7B/s

Figure 12: Performance Evaluation

There are possible scalability concerns as well as challenges associated with implementing a dependency-based scheduler. The growth of microservices and their dependencies may add to the inherent complexity of the scheduler's logic, increasing computing overhead and affecting the effectiveness of scheduling decisions. It is difficult to maintain and accurately handle dependence information, especially in dynamic microservices systems where changes are frequently error-prone and resource-intensive. When microservices are co-located on a single node, a phenomenon known as node fragmentation occurs. This can lead to resource underutilization on individual nodes, which may compromise system performance. The scheduler may be less flexible in a variety of scenarios due to its emphasis on inter-service dependencies at the expense of external resource dependencies. As the dependency graph grows, computationally demanding graph processing activities and more computational overhead in determining optimal assignments give rise to scalability difficulties. Real-time modifications to dependencies and connection overhead could make it harder for the scheduler to decide quickly.

For Kubernetes deployments, some developers might still favor utilizing EC2 directly over EKS because of advantages like more control and customization possibilities. More precise control over the underlying infrastructure is possible by directly controlling EC2 instances, which can be tailored to meet particular needs or preferences for instance kinds, networking setups, and storage options.Enterprises can discover that directly managing EC2 instances is a more economical option, since it allows them to optimize expenses according to the nature of their workload.

7 Conclusion and Future Work

The objective of the project is to create a unique dependency-based scheduler for microservices that takes into account the dependencies and minimze the data transfer among them achieved succesfully.Through the use of test scenarios and evaluations against the Kubernetes default scheduler, custom scheduler were developed with the goal of improving performance in a microservices architecture. The experimentation findings seemed promising because our scheduler was able to select from a number of suitable nodes, which enabled it to optimize the microservices deployment in the cloud, decrease workload, and enhance QoS while lowering latency.

Overall, the custom scheduler, developed in Go language for Kubernetes, showcases a robust approach to microservices orchestration scheduling. It configures the scheduler components and uses informers to monitor unscheduled nodes and pods by utilizing Kubernetes APIs. Intelligent dependency checks and node selection are part of the scheduling process, which improves deployment efficiency. Key Kubernetes scheduler features are replicated using bash script, which emphasizes node availability and dependency considerations. Notably, by taking into account the nuances in service deployment, the custom scheduler performs better than default schedulers. The custom script also incorporates a caching function for network optimization, using a shared cache technique for microservices with dependencies. This creative solution significantly decreases Kubernetes' overall bandwidth consumption and data transfer rates, demonstrating the custom scheduler's power in maximizing network efficiency and deployment.

This Proposed scheduler will able to efficiently handle the microservice dependencies at small scale and it check dependency only while deployment,suppose developer have changed dependencies among them after deployment, that is not address in given scenario which can be achieved as part of future work it might be with the strategic integration of machine learning (ML) and artificial intelligence (AI) approaches to improve the accuracy and adaptability of dependence forecasts in schedulers. More precise forecasts are made possible by the scheduler's ability to learn from and adjust to changing patterns in microservice dependencies by utilizing advanced algorithms. By incorporating real-time evaluations, historical data, and dynamic shifts in the microservices ecosystem, AI and ML models can further improve the decision-making process. This method not only improves the scheduler's accuracy in predicting dependencies, but it also offers a more adaptable and flexible framework that can handle complex situations in the Kubernetes environment.

References

Bac, T. P. and Kim, Y. (2023). Intent-based scheduling approach for kubernetes-oriented cloud systems, pp. 884–885.

Burns, B., Grant, B., Oppenheimer, D., Brewer, E. and Wilkes, J. (2016). Borg, omega,

and kubernetes, *Commun. ACM* **59**(5): 50–57.
URL: *https://doi.org/10.1145/2890784*

Kratzke, N. and Quint, P.-C. (2017). Understanding cloud-native applications after 10 years of cloud computing - a systematic mapping study, *Journal of Systems and Software* **126**: 1–16.
URL: *https://www.sciencedirect.com/science/article/pii/S0164121217300018*

Lai, W.-K., Wang, Y.-C. and Wei, S.-C. (2023). Delay-aware container scheduling in kubernetes, *IEEE Internet of Things Journal* .

Lebesbye, T., Mauro, J., Turin, G. and Yu, I. C. (2021). Boreas–a service scheduler for optimal kubernetes deployment, *Service-Oriented Computing: 19th International Conference, ICSOC 2021, Virtual Event, November 22–25, 2021, Proceedings 19*, Springer, pp. 221–237.

Marchese, A. and Tomarchio, O. (2022). Communication aware scheduling of microservices-based applications on kubernetes clusters., *CLOSER*, pp. 190–198.

Merkouche, S., Haroun, T., Bouanaka, C. and Smaali, M. (2022). Tera-scheduler for a dependency-based orchestration of microservices, *2022 International Conference on Advanced Aspects of Software Engineering (ICAASE)*, pp. 1–8.

Narasimhulu, M., Mounika, D. V., Varshini, P., K, A. and Rao, T. R. K. (2023). Investigating the impact of containerization on the deployment process in devops, *2023 2nd International Conference on Edge Computing and Applications (ICECAA)*, pp. 679–685.

Ning, A. (2023). A customized kubernetes scheduling algorithm to improve resource utilization of nodes, *2023 3rd Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS)*, pp. 588–591.

Pontarolli, R., Bigheti, J., Fernandes, M., Domingues, F., Risso, S. and Godoy, E. (2020). Microservice orchestration for process control in industry 4.0, pp. 245–249.

Shen, S., Han, Y., Wang, X., Wang, S. and Leung, V. C. M. (2023). Collaborative learning-based scheduling for kubernetes-oriented edge-cloud network, *IEEE/ACM Transactions on Networking* p. 1–15.
URL: *http://dx.doi.org/10.1109/TNET.2023.3267168*

Ungureanu, O.-M., Vlădeanu, C. and Kooij, R. (2019). Kubernetes cluster optimization using hybrid shared-state scheduling framework, *Proceedings of the 3rd International Conference on Future Networks and Distributed Systems*, pp. 1–12.

Vayghan, L. A., Saied, M. A., Toeroe, M. and Khendek, F. (2021). A kubernetes controller for managing the availability of elastic microservice based stateful applications, *Journal of Systems and Software* **175**: 110924.
URL: *https://www.sciencedirect.com/science/article/pii/S0164121221000212*

Zheng, G., Fu, Y. and Wu, T. (2021). Research on docker cluster scheduling based on self-define kubernetes scheduler, *Journal of Physics: Conference Series* **1848**: 012008.