MSc Research Project
MSc Cloud Computing

Manali Yadav
Student ID: 21225451

School of Computing
National College of Ireland

Supervisor: Rejwanul Haque

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Manali Amarnath Yadav		
Student ID:	21225451		
Programme:	MSc in Cloud Computing	**Year:**	2023-2024
Module:	Research Project		
Lecturer:	Rejwanul Haque		
Submission Due Date:	31/01/2024		
Project Title:	Custom Kubernetes Scheduler based on priority scheduling for Serverless Framework		
Word Count:	2829	**Page Count:**	10

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: Manali Yadav

Date: 30/01/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

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

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

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

Configuration Manual

Manali Yadav
Student ID: 21225451

The config manual contains prerequisites and steps required to configure or implement the research technical work, long with the steps and detailed procedures.

1 Prerequisites

Before embarking on the setup and deployment of the custom Kubernetes Scheduler answer, it's far essential to ensure that positive prerequisites are met. those stipulations are foundational to a success implementation and operation of the scheduler in a Kubernetes environment, especially within AWS. beneath is a detailed breakdown of these necessities:

1.1 AWS Account Requirements

- Active AWS Account: access to an Amazon net offerings (AWS) account is obligatory. if you do no longer have an account, you can sign on for one on the AWS website.
- Appropriate Permissions: make certain that your AWS account has the necessary permissions to create and manage EC2 instances, EKS clusters, and other applicable AWS resources. This normally entails having administrative get entry to or specific IAM roles configured.
- Billing Considerations: Be aware of the AWS pricing model, as putting in EC2 times and EKS clusters may incur charges.

1.2 Basic Kubernetes knowledge

- Kubernetes Concepts: Familiarity with fundamental Kubernetes standards which includes pods, nodes, clusters, deployments, and services is essential. knowledge how these components have interaction within a Kubernetes atmosphere will aid within the powerful use of the custom scheduler.
- Kubernetes architecture: simple information of Kubernetes structure, along with the manage aircraft and worker nodes, is beneficial.

1.3 Technical skills in Cloud Computing, Containerisation, and Serverless Architectures

- Cloud Computing basics: A solid know-how of cloud computing standards, which include scalability, elasticity, and cloud aid management.
- Containerisation: understanding of containerisation concepts and container orchestration. Familiarity with Docker or comparable container technologies is wonderful.
- Serverless Computing: understanding serverless architectures and their function in present day cloud environments. This includes information of ways serverless function’s and are controlled.

1.4 Required tools

- AWS CLI: [1]The AWS Command Line Interface (CLI) should be mounted and configured for get entry to to AWS offerings. This device allows you to interact with AWS services at once from the command line.
- kubectl: that is the command-line device for interacting with the Kubernetes cluster. It must be installed and configured to speak together with your Kubernetes cluster.
- Helm: Helm, a package supervisor for Kubernetes, is used for coping with Kubernetes packages. It simplifies the deployment and management of packages on Kubernetes.
- Python: Python is required for strolling simulation scripts and potentially for other automation obligations. make certain that a current version of Python is mounted and configured on your gadget.

2 Tools Installation

Here are concise step-with the aid of-step commands for putting in the necessary equipment: AWS CLI, kubectl, Helm, and Python. Following those steps will make sure you have the desired tools to control and set up your Kubernetes scheduler.

2.1 AWS CLI set up.

- Download: Visit the AWS CLI legit page and download the right version on your running device.
- Installation: Run the downloaded installer and comply with the on-screen commands.
- Configure: Open a terminal or command prompt and run aws configure to set up your AWS credentials (access Key identity, secret get admission to Key) and default location.

2.2 kubectl set up

- Download: observe the commands on the Kubernetes official documentation to download kubectl to the running machine.
- Installation: Execute the installation instructions as per the commands for your precise platform.
- Verify: Run kubectl version --customer to make certain it's hooked up efficaciously.

2.3 Helm set up

- Download: go to the Helm releases web page and download the modern-day launch in your working machine.
- Set up: Unpack the downloaded report and pass the helm binary to a listing to your machine's route.
- Verify: Run helm version to test the installation.

2.4 Python set up

- Download: go to the Python legit internet site and download the ultra-modern version to your running system.
- Install: Run the installer and comply with the setup commands. make certain you pick out the choice to feature Python to your course.
- Verify: Open a terminal or command set off and kind python --version to verify the set up.

[1] https://www.knowledgehut.com/blog/cloud-computing/aws-cli

2.5 Verification of successful set up

To affirm that all tools are established effectively, run the subsequent commands on your terminal or command prompt:

- aws --version
- kubectl version --client
- helm version
- python --version

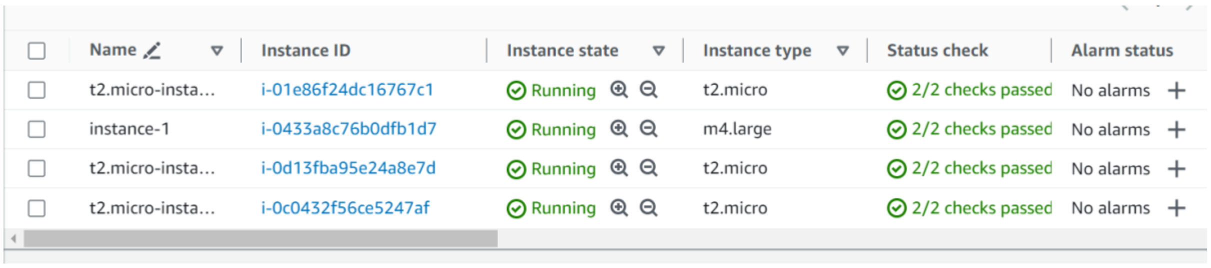
Those steps will equip you with the vital tools to continue with configuring and deploying your custom Kubernetes scheduler in AWS surroundings.

3 Setting up the Kubernetes Environment on AWS

3.1 Create EC2 Instances

- Log in to AWS Console: access your AWS account and navigate to the EC2 Dashboard.
- Launch Instances: click on 'launch instances' to begin the setup method.
- Select AMI: select an Amazon system photograph (AMI) appropriate for your Kubernetes cluster.
- Select instance type: pick out the instance sorts (e.g., m4.big, t2.micro) as consistent with your requirements.
- Configure example info: set up community and subnet settings and configure other example settings as needed.
- Add storage: connect additional storage if required.
- Configure Security group: set up protection businesses with appropriate policies for inbound and outbound visitors.
- Review and Launch: evaluation your configurations and click 'launch'. choose a key pair or create a new one for SSH get right of entry to to the instances.

After creating the instances, we can see the running instances in the instances dashboard as shown below.



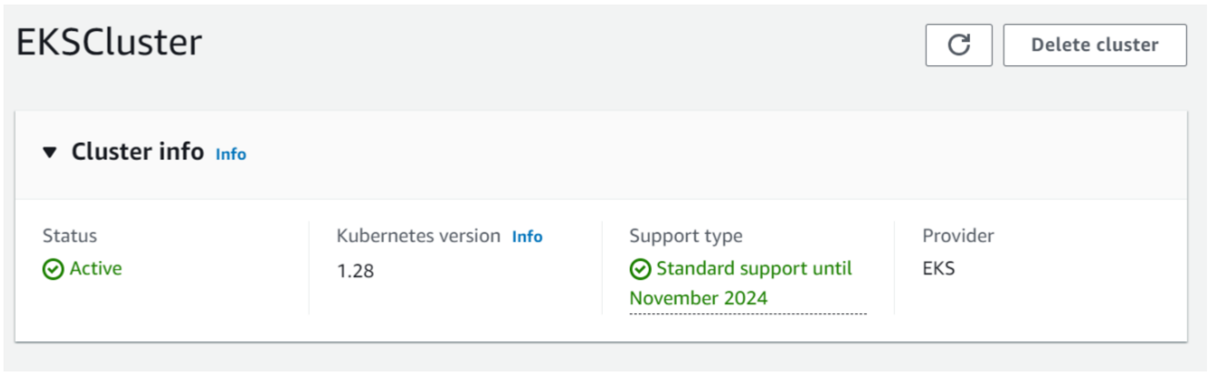
Name	Instance ID	Instance state	Instance type	Status check	Alarm status
t2.micro-insta...	i-01e86f24dc16767c1	Running	t2.micro	2/2 checks passed	No alarms
instance-1	i-0433a8c76b0dfb1d7	Running	m4.large	2/2 checks passed	No alarms
t2.micro-insta...	i-0d13fba95e24a8e7d	Running	t2.micro	2/2 checks passed	No alarms
t2.micro-insta...	i-0c0432f56ce5247af	Running	t2.micro	2/2 checks passed	No alarms

3.2 Configure EKS Cluster

- [2]Access EKS Service: inside the AWS Console, visit the EKS carrier.
- Create Cluster: click on 'Create Cluster'. name your cluster and choose the Kubernetes model.
- Cluster Configuration: pick out the VPC and subnets to your cluster. Configure different settings-like IAM roles and logging as wanted (Berman, 2023).
- Create: evaluate your settings and create the cluster.

[2] https://logz.io/blog/amazon-eks-cluster/

Here we created an EKS Cluster named EKSCluster in the AWS which acts as the Master Node as shown below.



EKSCluster

Delete cluster

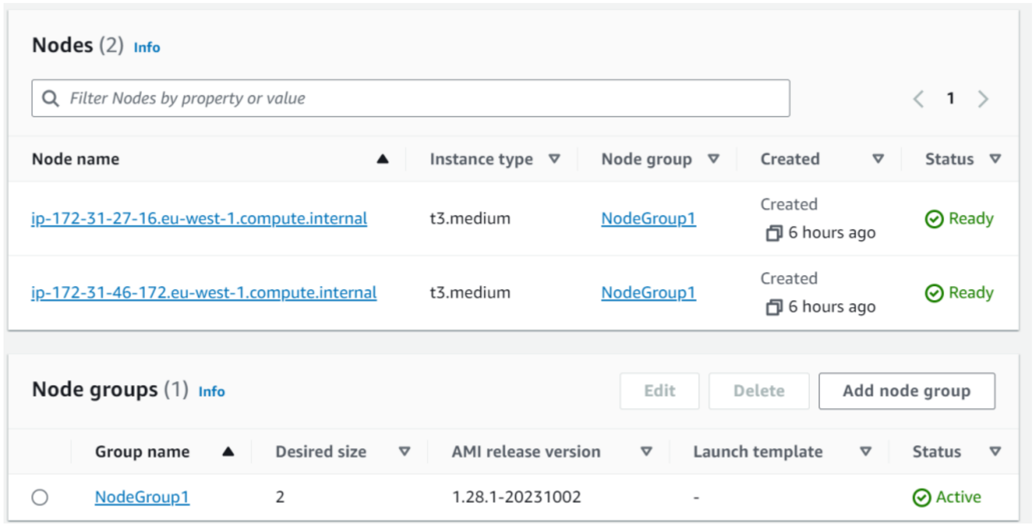
▼ Cluster info Info

Status	Kubernetes version Info	Support type	Provider
✓ Active	1.28	✓ Standard support until November 2024	EKS

3.3 Create Node Groups

- Access Cluster: within the EKS Dashboard, pick out your cluster.
- Create Node Group: click on 'add Node organisation'. call your node institution and set node IAM position.
- Configure Nodes: pick the instance kind on your nodes and set the favored scaling parameters.
- Networking: pick the subnets to your nodes.
- Launch: overview and create the node group.

Here we created one Node Group called NodeGroup1 which acts as a Worker Node Inside the EKS Cluster as shown below.



Nodes (2) Info

Filter Nodes by property or value

Node name	Instance type	Node group	Created	Status
ip-172-31-27-16.eu-west-1.compute.internal	t3.medium	NodeGroup1	Created 6 hours ago	✓ Ready
ip-172-31-46-172.eu-west-1.compute.internal	t3.medium	NodeGroup1	Created 6 hours ago	✓ Ready

Node groups (1) Info

Edit Delete Add node group

	Group name	Desired size	AMI release version	Launch template	Status
○	NodeGroup1	2	1.28.1-20231002	-	✓ Active

4 Deploying Apache OpenWhisk

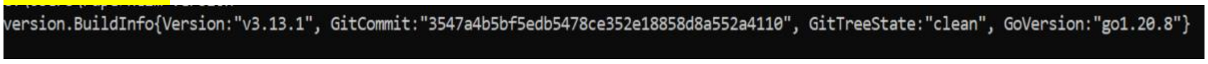
4.1 Install Helm

- [3]Download Helm: go to the Helm releases web page and download the proper version for your device.
- Install: observe the set-up instructions-specific to your operating machine.
- Initialise Helm: Run helm init to installation Helm for your gadget.

Ensure Helm is properly configured and ready to use in your cluster by checking its status:

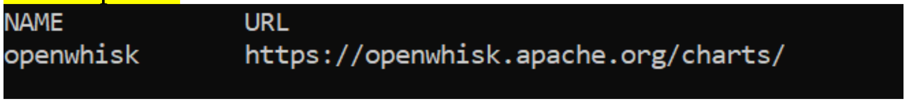
[3] https://phoenixnap.com/kb/install-helm/

helm version



```
version.BuildInfo{Version:"v3.13.1", GitCommit:"3547a4b5bf5edb5478ce352e18858d8a552a4110", GitTreeState:"clean", GoVersion:"go1.20.8"}
```

To list the available Helm repositories, run:
helm repo list



```
NAME            URL
openwhisk       https://openwhisk.apache.org/charts/
```

Deploy Applications Using Helm Charts
Steps:
1. Add Helm repositories for charts you want to use.

Use -> helm repo adds command

2. Install applications using Helm charts. For example, to install a sample application.

helm install myapp stable/sample-chart

Monitor and manage your deployed applications with Helm. You can upgrade, rollback, and delete releases, among other operations.

4.2 Upload OpenWhisk to Helm

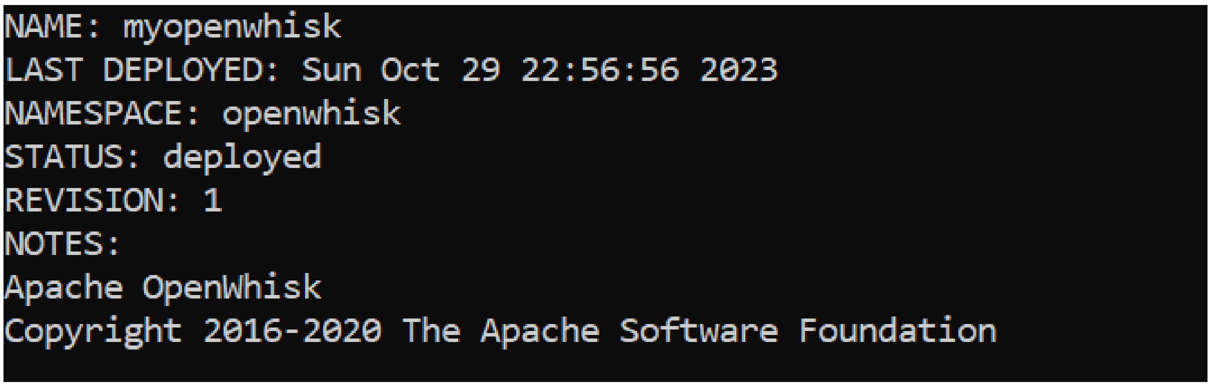
- [4]Add Repository: Run helm repo add openwhisk to add the OpenWhisk Helm chart repository to your Helm configuration.

4.3 Configure OpenWhisk

- Create Configuration File: Create a myvalues.yaml document to specify custom configurations for OpenWhisk.
- Edit Configuration: customise the settings in myvalues.yaml in line with your deployment needs.

4.4 Install OpenWhisk

- Installation Chart: Use helm install my-openwhisk openwhisk/openwhisk -f myvalues.yaml to deploy OpenWhisk to your Kubernetes cluster.



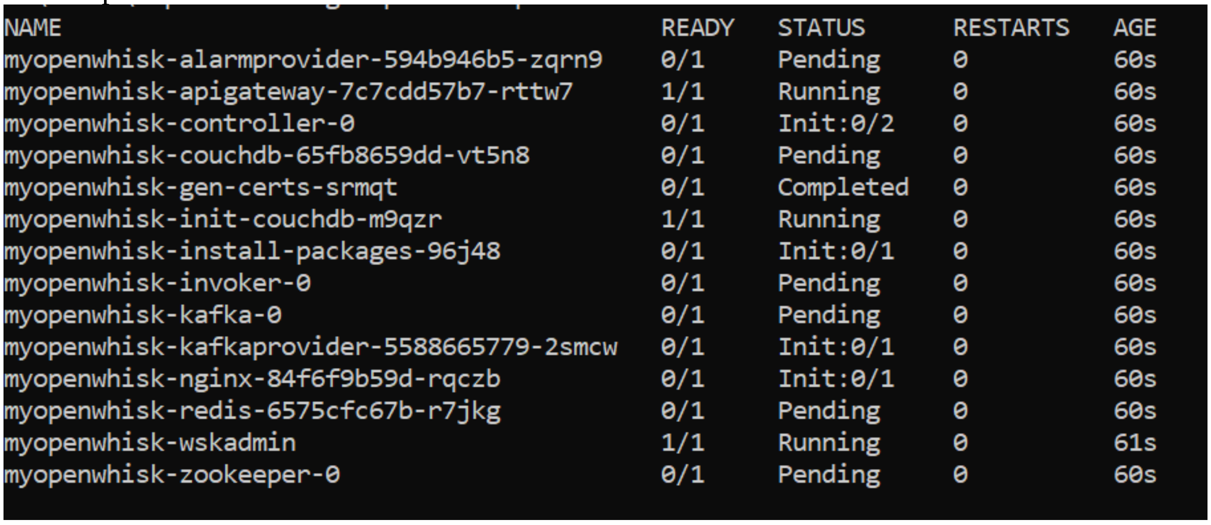
```
NAME: myopenwhisk
LAST DEPLOYED: Sun Oct 29 22:56:56 2023
NAMESPACE: openwhisk
STATUS: deployed
REVISION: 1
NOTES:
Apache OpenWhisk
Copyright 2016-2020 The Apache Software Foundation
```

[4] https://jamesthom.as/2018/01/starting-openwhisk-in-sixty-seconds/

After we can see the pods running using the command
kubectl get pods -n openwhisk

- Verify set up: check the status of the deployment the usage of kubectl get pods -n openwhisk to ensure all components are walking correctly (Thomas, 2018).

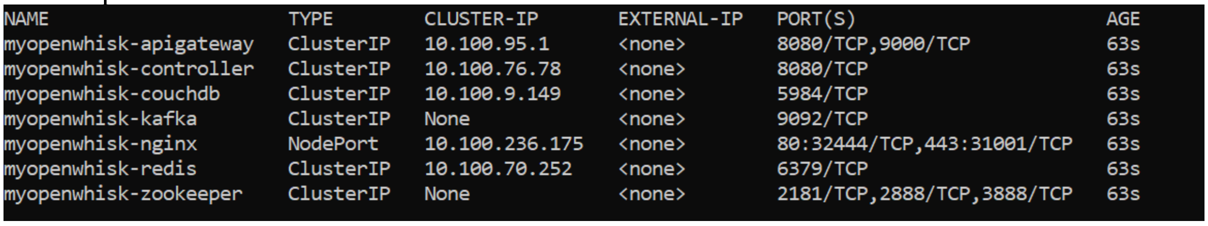
The output looks like below.



```
NAME                                        READY   STATUS      RESTARTS   AGE
myopenwhisk-alarmprovider-594b946b5-zqrn9   0/1     Pending     0          60s
myopenwhisk-apigateway-7c7cdd57b7-rttw7     1/1     Running     0          60s
myopenwhisk-controller-0                    0/1     Init:0/2    0          60s
myopenwhisk-couchdb-65fb8659dd-vt5n8        0/1     Pending     0          60s
myopenwhisk-gen-certs-srmqt                 0/1     Completed   0          60s
myopenwhisk-init-couchdb-m9qzr              1/1     Running     0          60s
myopenwhisk-install-packages-96j48          0/1     Init:0/1    0          60s
myopenwhisk-invoker-0                       0/1     Pending     0          60s
myopenwhisk-kafka-0                         0/1     Pending     0          60s
myopenwhisk-kafkaprovider-5588665779-2smcw  0/1     Init:0/1    0          60s
myopenwhisk-nginx-84f6f9b59d-rqczb          0/1     Init:0/1    0          60s
myopenwhisk-redis-6575cfc67b-r7jkg          0/1     Pending     0          60s
myopenwhisk-wskadmin                        1/1     Running     0          61s
myopenwhisk-zookeeper-0                     0/1     Pending     0          60s
```

We can see the services running using the command
kubectl get services -n openwhisk

The output looks like below.



```
NAME                     TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                        AGE
myopenwhisk-apigateway   ClusterIP   10.100.95.1     <none>        8080/TCP,9000/TCP              63s
myopenwhisk-controller   ClusterIP   10.100.76.78    <none>        8080/TCP                       63s
myopenwhisk-couchdb      ClusterIP   10.100.9.149    <none>        5984/TCP                       63s
myopenwhisk-kafka        ClusterIP   None            <none>        9092/TCP                       63s
myopenwhisk-nginx        NodePort    10.100.236.175  <none>        80:32444/TCP,443:31001/TCP     63s
myopenwhisk-redis        ClusterIP   10.100.70.252   <none>        6379/TCP                       63s
myopenwhisk-zookeeper    ClusterIP   None            <none>        2181/TCP,2888/TCP,3888/TCP     63s
```

5 Implementing the Custom Scheduler

5.1 Design the Scheduler

The design section of the custom scheduler is pivotal in tailoring Kubernetes scheduling to unique desires. This entails:

1. Developing the Scheduler Algorithm:
   - Recognition on developing a pod scoring set of rules that evaluates each pod-based totally on elements-like CPU and reminiscence requirements, priority, and different custom criteria.
   - Make certain the set of rules aligns together with your Kubernetes environment's operational goals, together with optimising aid usage or prioritising certain workloads.
2. Programming the Scheduler:
   - Write the scheduler code in a language likeminded with the Kubernetes API, generally pass or Python.
   - Utilise Kubernetes patron libraries to interact with the cluster and control sources.
3. Testing the algorithm:
   - Before deployment, very well look at the scheduler in managed surroundings.

- Simulate diverse scenarios to ensure the scheduler behaves as predicted below unique conditions.

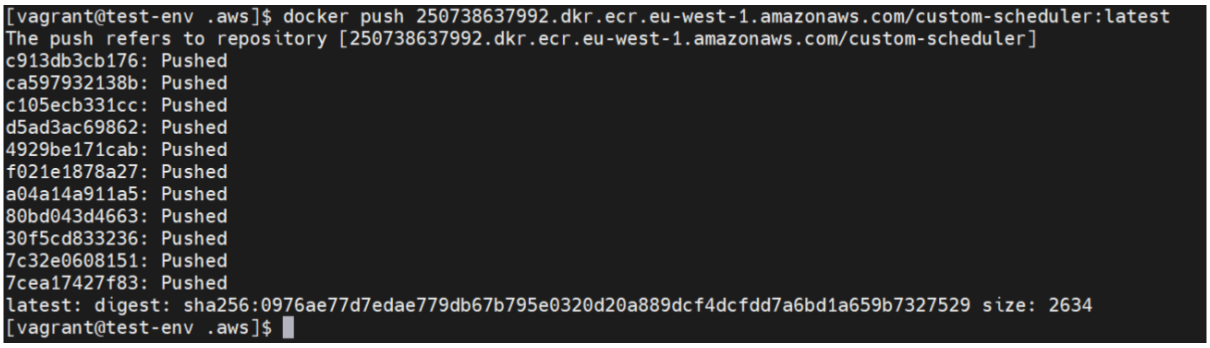
4. Documentation:
   - Report the scheduler's layout, such as the logic at the back of the scoring set of rules and commands for enhancing or extending its capability.

Scheduler.py

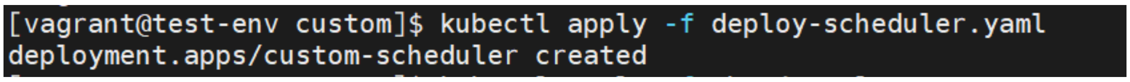
5.2 Deploy the Scheduler

Deploying the custom scheduler involves integrating it into your Kubernetes cluster:

1. Prepare the Deployment Configuration:
   - Create a Kubernetes deployment configuration file for the scheduler. This file needs to define the scheduler's deployment parameters, including aid requests and bounds, environment variables, and other vital settings.

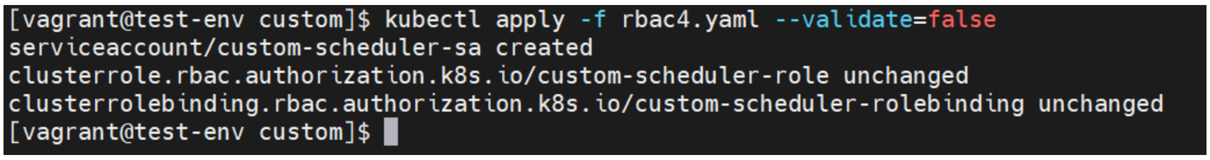


```
[vagrant@test-env .aws]$ docker push 250738637992.dkr.ecr.eu-west-1.amazonaws.com/custom-scheduler:latest
The push refers to repository [250738637992.dkr.ecr.eu-west-1.amazonaws.com/custom-scheduler]
c913db3cb176: Pushed
ca597932138b: Pushed
c105ecb331cc: Pushed
d5ad3ac69862: Pushed
4929be171cab: Pushed
f021e1878a27: Pushed
a04a14a911a5: Pushed
80bd043d4663: Pushed
30f5cd833236: Pushed
7c32e0608151: Pushed
7cea17427f83: Pushed
latest: digest: sha256:0976ae77d7edae779db67b795e0320d20a889dcf4dcfdd7a6bd1a659b7327529 size: 2634
[vagrant@test-env .aws]$
```



```
[vagrant@test-env custom]$ kubectl apply -f deploy-scheduler.yaml
deployment.apps/custom-scheduler created
```

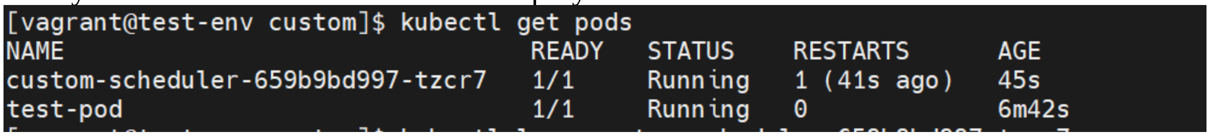
2. Set up Scheduler Permissions:
   - Make sure the scheduler has the important permissions to engage with the Kubernetes API. This typically involves setting up suitable RBAC (function-based access manipulate) policies.



```
[vagrant@test-env custom]$ kubectl apply -f rbac4.yaml --validate=false
serviceaccount/custom-scheduler-sa created
clusterrole.rbac.authorization.k8s.io/custom-scheduler-role unchanged
clusterrolebinding.rbac.authorization.k8s.io/custom-scheduler-rolebinding unchanged
[vagrant@test-env custom]$
```

.

3. Deploy using kubectl:
   - Use the kubectl apply -f [scheduler-config-file].yaml command to install the scheduler for your Kubernetes cluster.
   - Verify the deployment via che



```
[vagrant@test-env custom]$ kubectl get pods
NAME                               READY   STATUS    RESTARTS       AGE
custom-scheduler-659b9bd997-tzcr7  1/1     Running   1 (41s ago)    45s
test-pod                           1/1     Running   0              6m42s
```

cking the scheduler pod's popularity with kubectl get pods.

4. Integrate with the Kubernetes Cluster:

- Configure your Kubernetes cluster to apply the custom scheduler. this might contain updating the pod specification to specify the custom scheduler or editing the cluster's scheduler configuration.

5. Monitor and Validate:
   - As soon as deployed, reveal the scheduler's performance to make certain it's functioning as intended.
   - Validate its effectiveness by watching how it allocates assets and schedules pods in evaluation to the default scheduler.

6 Simulation and Testing

6.1 Develop Python Scripts

The improvement of Python scripts is a essential step in simulating and evaluating the performance of the custom Kubernetes scheduler. those scripts will serve two number one functions:

1. Workload Simulation:
   - Write scripts to generate synthetic workloads that mimic real-world eventualities. these workloads should vary in aid necessities (CPU, memory) and priorities to thoroughly test the scheduler's skills.
   - Make certain the scripts can create extraordinary types of pods, with various sizes and configurations, to simulate diverse software wishes.
2. Performance Data Collection:
   - Develop mechanisms inside the scripts to accumulate applicable performance metrics. This consists of pod start-up times, aid utilisation, node allocation efficiency, and greater.
   - Put into effect logging or data aggregation capabilities to seize and keep performance data for evaluation.

6.2 Run Simulations

Executing the simulations includes strolling the evolved Python scripts for your Kubernetes surroundings:

1. Enviornment Setup:
   - Put together your Kubernetes cluster for simulation. make certain all-important assets and configurations are in vicinity.
   - Deploy the custom scheduler within the cluster if no longer already strolling.
2. Execute Scripts:
   - Run the Python scripts to initiate the workload simulation. screen the cluster to ensure the workloads are being deployed as anticipated.
   - Use Kubernetes tools like kubectl or the dashboard to look at the scheduler's conduct in real-time.
3. Monitor Simulations:
   - Continuously display the simulations for any unexpected conduct or errors.
   - Alter the scripts or cluster configuration as needed to make sure accurate and effective simulation.

6.3 Analyse Results

The very last step is to investigate the accrued information to assess the overall performance of the custom scheduler:

1. Data Compilation:
   - Accumulate all the performance information gathered for the duration of the simulations.
   - Arrange the data in a layout appropriate for evaluation, which includes spreadsheets or charts.
2. Performance comparison:
   - Examine the performance metrics of the custom scheduler towards the ones of the default Kubernetes scheduler. consciousness on key regions-like aid allocation performance, response to high-demand eventualities, and normal device balance.
   - Use statistical methods or visualisation tools to highlight variations and developments.
3. Insights and Conclusions:
   - Draw insights from the records evaluation. identify regions wherein the custom scheduler excels or wishes development.
   - Document your findings and conclusions, imparting a clear understanding of the custom scheduler's overall performance in diverse scenarios.

7 Monitoring and Maintenance

7.1 Monitor Scheduler Performance

Ordinary monitoring is crucial to make certain the custom scheduler operates efficaciously and effectively:

- Set up Monitoring tools: utilise Kubernetes monitoring equipment-like Prometheus and Grafana to music the scheduler's overall performance metrics, which includes pod scheduling times, aid utilisation, and node health.
- Regular Check: schedule recurring tests to study the scheduler's overall performance. pay attention to any anomalies or performance dips.
- Resource Utilisation Analysis: preserve an eye fixed on how well the scheduler is making use of cluster sources. look for patterns or trends that imply either efficient or inefficient useful resource utilisation.
- Alerts and Notifications: Configure indicators for crucial occasions or thresholds to proactively control capacity problems.

7.2 Update and Maintain

Non-stop improvement is prime to preserving an effective scheduler:

- Update Regularly: hold the scheduler up to date with the today's Kubernetes releases and protection patches.
- Performance-based Improvements: Use the records collected from tracking and checking out to refine and enhance the scheduler. implement enhancements to optimise its overall performance.
- Documentation Updates: maintain the documentation of the scheduler, along with setup and configuration publications, up to date with any modifications or improvements made.

8 Troubleshooting and FAQs

Common troubles and answers

Address common demanding situations users might stumble upon:

- Scheduler not Allocating Pods: provide solutions-like checking scheduler logs, verifying configuration files, and ensuring accurate Kubernetes API integration.
- Resource Allocation issues: provide steerage on adjusting useful resource allocation parameters or troubleshooting ability conflicts.

Frequently asked Questions

Assemble a listing of FAQs to assist users in understanding and using the custom scheduler efficaciously:

1. How to customise the Scheduler's Scoring algorithm?
2. What to Do If the Scheduler Fails to start?
3. Best Practices for Scheduler maintenance and Updates.

9 conclusion

Summary of the Configuration manual

This manual has furnished a comprehensive guide for putting in place, deploying, and managing a custom Kubernetes scheduler in an AWS environment. It blanketed the entirety from initial device installation to advanced scheduler configuration, trying out, and upkeep.

Final remarks and additional resources

- Encouragement for remarks: encourage users to provide feedback on the scheduler's overall performance and the manual's usefulness.
- Continuous gaining knowledge of: Remind users of the significance of staying up to date with Kubernetes and cloud computing advancements.
- Additional resources: offer hyperlinks to further reading materials, community boards, and assist channels for added assist and mastering.