InComTech: Jurnal Telekomunikasi dan Komputer
vol.14, no.1, April 2024, 60-77
http://publikasi.mercubuana.ac.id/index.php/Incomtech
P-ISSN: 2085-4811 E-ISSN: 2579-6089

Analisis Performa *Container* Pada Kubernetes Service di Google Cloud Platform

Palda Puspita Dianti, Galura Muhammad Suranegara*, Hafiyyan Putra Pratama

Sistem Telekomunikasi, Universitas Pendidikan Indonesia,
Jl. Veteran No.8, Purwakarta, 41115, Indonesia
*Email Penulis Koresponden: galurams@upi.edu

Abstrak **:**
Container-based virtualization telah diterima di seluruh *Platform Cloud* dalam beberapa tahun terakhir, dan tren ini kemungkinan akan berlanjut di tahun-tahun mendatang. Akibatnya, sistem *container orchestration* menjadi semakin penting. Karena *stability*, *maturity*, dan fungsionalitasnya yang komprehensif, Kubernetes telah menjadi standar de facto. Semua penyedia *Cloud* utama pada saat ini menawarkan solusi Kubernetes terkelola *Cloud-Native* untuk membebaskan pengguna dari tekanan karena harus merancang dan memelihara infrastruktur Kubernetes yang rumit sambil tetap memanfaatkan fungsinya. Sebagai contoh pada tahun 2022, sebanyak 96% dari Sysdig's global *customer* yang menggunakan layanan *container* memilih Kubernetes sebagai *Container Orchestration*-nya. Dan dalam laporan awal CNCF pada tahun 2022 sebanyak 79% responden menggunakan layanan terkelola seperti Google Kubernetes Engine (GKE). Tujuan artikel ini adalah untuk menganalisis performa *container* yang berjalan secara *hosted* seperti pada Google Cloud Platform (GCP). Analisis dilakukan dengan cara pengujian secara *real-time* yang dilakukan didalam cluster Kubernetes, yang terdiri dari dua node dengan *instance* mesin 4vcpu yang dideploy dengan nginx:1.23. *Execution time* dilakukan sebanyak 50.000 eksekusi, menghasilkan waktu terbaik 0,0000106 sec per loop. Total pengunaan memori adalah 18,36 MiB, dengan network *received* sebesar 16,015 KiB/s dan *network transmitted* sebesar 16,057 KiB/s. Adapun *value price performance* yang didapatkan Google Cloud Platform (GCP) yaitu sebesar 952,4 MIPS/$.

Kata Kunci:
Container,
Kubernetes,
GCP,
GKE

Riwayat Artikel:
Diserahkan 18 Agustus, 2023
Direvisi 23 Januari, 2024
Diterima 31 Jan, 2024

DOI:
10.22441/incomtech.v14i1.22466

1. PENDAHULUAN

Pada beberapa tahun terakhir, banyak perusahaan di dunia yang memilih untuk bermigrasi dari sistem berbasis *monolithic* ke *microservice* [1]. Hal ini didukung dengan data yang diambil dari *website* jrebel.com pada tahun 2021. Disebutkan bahwa sebanyak 49% responden menggunakan arsitektur berbasis *microservice* pada aplikasi utamanya. Dari data tersebut, sebanyak 36% responden baru saja bermigrasi ke layanan arsitektur berbasis *microservice,* 30% responden sudah sepenuhnya menggunakan aplikasi berbasis *microservice*, 21% responden masih dalam tahap perbincangan dan 13% responden tidak berencana untuk berpindah ke layanan arsitektur berbasis *microservice* [2]. Fenomena ini bukan tanpa dasar, percepatan digitalisasi membuat semuanya mejadi jauh lebih cepat dan dituntut untuk memenuhi permintaan tersebut. Selain itu terdapat banyak keuntungan yang ditawarkan layanan arsitektur berbasis *microservice* seperti proses *deployment* yang dapat membuka model baru untuk meningkatkan *scale* dan ketahanan sistem dan dapat memadupadankan teknologi.

Sebagai *services*, *microservice* dapat dikerjakan secara paralel. Hal ini memungkinkan untuk mempekerjakan lebih banyak *developers* untuk mengatasi masalah tanpa mereka saling menghalangi. Selain itu hal ini membuat para *developers* lebih memahami *jobdesk* karena mereka dapat memusatkan perhatian hanya pada satu bagian saja. Proses *isolation* juga memungkinkan untuk memvariasikan teknologi seperti menggabungkan berbagai bahasa pemrograman, *styles* pemrograman, *deployment platforms*, atau *databases* untuk menemukan perpaduan yang tepat [3]. Dibalik keunggulannya, *microservice* juga memiliki kekurangan seperti tingkat kompleksitas *networking* yang lebih tinggi serta kemungkinan terjadinya *overhead* pada *database* dan *server* sehingga dibutuhkan pengetahuan dan pemahaman yang baik terhadap *containerization*, *container orchestrator*, dan *deployment*.

Pada tahun 2017, sebanyak 78,66% responden memilih Docker sebagai teknologi *container* yang paling banyak digunakan disusul rkt sebanyak 1,67% dan Mesos *Container*izer sebanyak 2,93% [4]. Teknologi *container* memberikan kemudahan dalam membuat *services* aplikasi dalam jumlah banyak dan cepat. Seperti pada penelitian yang dilakukan Arango pada 2017 yang mengevaluasi kinerja dari implementasi teknologi virtualisasi berbasis *container* Linux termasuk Linux Vserver, OpenVZ, dan Linux Container (LXC). Penelitian tersebut bertujuan untuk menentukan kesesuaian teknologi *high performance computing* (HPC) dan bereksperimen pada aplikasi HPC dengan beberapa tolak ukur untuk mengukur CPU, *Memory System*, *Disk*, dan Performa *Network*. Kesimpulannya adalah bahwa teknologi *container* tersebut memiliki beberapa kekurangan terkait keamanan dan *isolation* terutama pada *memory*, *disk*, dan *network* [5]. Penelitian serupa pun pernah dilakukan pada dua tahun sebelumnya oleh Preeth tepatnya pada 2015. Penelitian dilakukan dengan mengevaluasi kinerja Docker *container* dengan bantuan *software* Bonnie++. Hasilnya Docker berjalan dengan baik dan bisa dibandingkan dengan performa dari OS yang berjalan pada *bare-metal*. Pada penelitian ini juga keamanan menjadi salah satu *concern* pada sistem Docker *Container* [6]. Selain *issue* mengenai keamanan, Docker juga memiliki beberapa kekurangan lainnya seperti tidak cocok untuk aplikasi yang membutuhkan antarmuka pengguna grafis yang kuat (*rich* GUI), sulit untuk mengoperasikan

container dalam jumlah besar, Docker juga tidak bisa dan tidak kompatibel jika dipakai untuk lintas *platform* maksudnya ketika suatu aplikasi di desain Docker *container* untuk dijalankan di windows, maka *container* tersebut tidak bisa dijalankan di Linux Docker *container* oleh karena itu Docker cocok dipakai hanya jika *development* dan *testing Operating System* nya sama. Dan yang terakhir Docker tidak menyediakan opsi untuk pencadangan dan pemulihan data [7].

Dikarenakan beberapa kekurangan dan fitur yang tidak bisa diberikan oleh Docker, Kubernetes hadir sebagai salah satu solusi dalam sistem *containerisasi*. Kubernetes dapat menjalankan Docker *Container* dan *Workloads* sehingga dapat mengatasi beberapa kompleksitas operasi saat berpindah ke penskalaan beberapa *container* yang diterapkan pada beberapa *server*. Pada awalnya Kubernetes dikembangkan oleh Google sebagai proyek *open source* di tahun 2014. Seiring dengan penelitian yang terus berkembang, *Cloud Service Provider* (CSPs) mulai membangun Kubernetes pada layanan terkelolanya sendiri misalnya AWS dengan Amazon Elastic Kubernetes Service (EKS), Microsoft Azure dengan Microsoft Azure Kubernetes Service (AKS) dan GCP dengan Google Kubernetes Engine (GKE). Selain dapat berjalan pada CSPs atau *Hosted*, Kubernetes juga dapat berjalan secara *Self-Hosted* atau dibangun pada *local* komputer *user*. Seperti penelitian yang dilakukan Muddinagiri pada tahun 2019 yang melakukan *deploying* Docker *container* dengan bantuan *software* Minikube [8].

Laporan awal CNCF pada tahun 2022 mengemukakan bahwa 96% responden menggunakan atau mengevaluasi Kubernetes. Dan 79% responden menggunakan layanan terkelola seperti Google Kubernetes Engine (GKE) [9]. Dalam memilih suatu *platform* yang dapat membuat, menguji, dan menerapkan suatu aplikasi dengan *container* dibutuhkan suatu pengujian performa. Seperti penelitian yang dilakukan oleh Pereira dan Sinnott yang membahas pengujian performa pada *container* yang berjalan pada Kubernetes *Service* dengan *host cloud provider* AWS, Azure, dan GCP. Penelitian tersebut berfokus pada evaluasi karakteristik dari *container* yang berjalan di Kubernetes *services* yang berbeda dengan tujuan untuk menganalisis beberapa level dari *overhead* yang diperkenalkan *cloud provider* ketika Docker *Container* berjalan [10]. Penelitian serupa juga pernah dilakukan oleh Nugroho, dkk dimana mereka menguji *availability* dan *request* dari *container* yang berjalan pada Kubernes *Service* GCP [11]. Selain itu pemantauan dan monitoring juga diperlukan agar dapat meningkatkan performa [12].

Oleh karena itu penelitian akan dilakukan pada Kubernetes service pada layanan terkelola yaitu Google Cloud Platform (GCP). Pengujian akan dilakukan secara *realtime* dengan bantuan beberapa *tools* seperti Prometheus dan Python. Prometheus merupakan sebuah sistem dan sistem *monitoring service*. Prometheus akan mengumpulkan *metrics* dari tar*get* yang dikonfigurasi pada interval tertentu, mengevaluasi *rule expressions*, menampilkan hasil dan bisa men-*trigger alert* ketika suatu kondisi tertentu diamati [13]. Sedangkan Python akan bertugas sebagai *benchmarking tool* untuk mendapatkan jumlah *execution time.* Python timeit akan menjadi *personal argument* yang dipakai dalam penelitian ini [14]. Secara umum, kontribusi dalam penelitian ini adalah mendapatkan *realtime performance* dari *container* yang berjalan pada Google Kubernetes Engine (GKE) yang bertujuan

untuk mendapatkan parameter baru dalam menganalisa suatu *platform* yang dapat menjalankan *container* diatas Kubernetes *service*.

Paper ini akan terdiri dari 4 *section*. *Section* 1 mendeskripsikan latar belakang dan penelitian sebelumnya yang relevan dengan topik penelitian paper ini. Diskusi mengenai metode dan rencana penelitian akan dibahas pada *section* 2. *Section* 3 akan membahas hasil penelitian yang didapatkan dari pengujian yang telah dilakukan. Dan kesimpulan akan berada pada *section* 4, yang berarti akan membahas pernyataan singkat terkait dengan hasil yang diperoleh dari pengujian pada section sebelumnya.

2. METODE

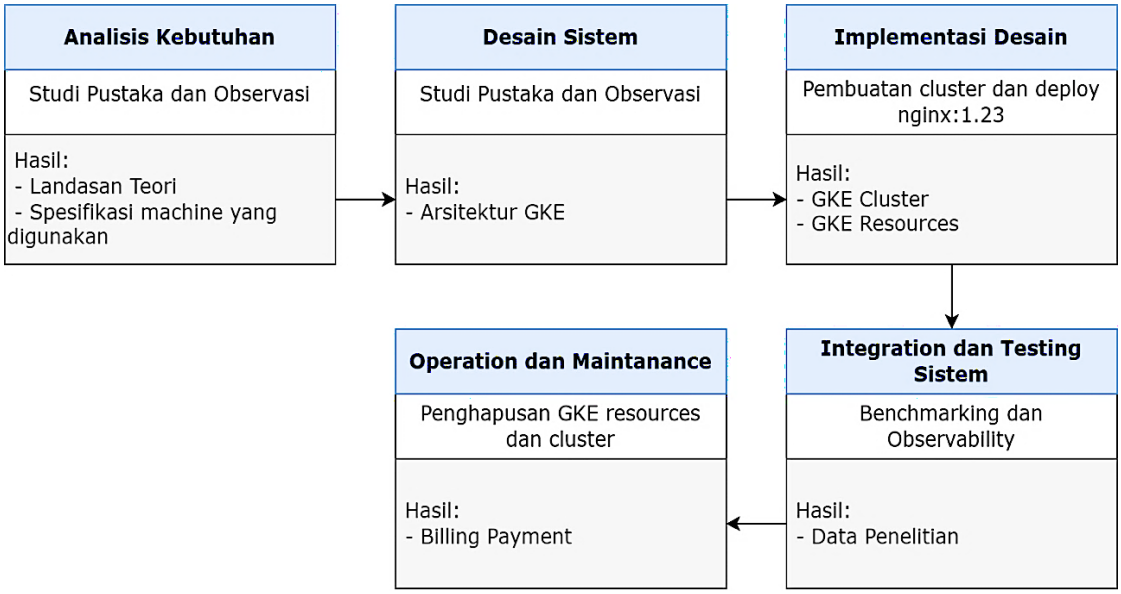
Metode penelitian yang akan digunakan adalah RnD dengan metodologi umum *System Development Life Cycle* (SDLC). Model SDLC yang dipakai pada penelitian ini adalah metode *waterfall*. SDLC merupakan metodologi yang pertama kali diperkenalkan oleh Meir Manny Lehman pada tahun 1969. Beliau membuat pendekatan dasar dengan memisahkan desain, evaluasi, dan dokumentasi dalam suatu penelitian mendesain suatu *software*. SDLC dapat dibagi menjadi dua tipe generik. Pertama, tipe model *waterfall*, hal ini dikarenakan pada tahun 1970 Winston Royce menguraikan SDLC dengan tahapan model berurutan yang diurutkan kebawah seperti aliran air terjun. Model *waterfall* akan menyajikan strategi ideal dalam membangun sebuah proyek dengan menguraikan beberapa prinsip praktik seperti desain, *coding*, dokumentasi dari setiap tahap dan perencanaan yang tepat. Maka model *waterfall* akan menggambarkan pengembangan urutan tahapan yang dapat diringkas dalam lima langkah yaitu analisis, desain, implementasi, *testing*, *deployment* dan *maintenance* [15]. Selain model *waterfall*, Model SDLC yang memiliki langkah serupa adalah model *prototyping*. Namun model *waterfall* dan *prototyping* tentu memiliki perbedaan seperti yang tertera pada tabel 1

Tipe kedua SDLC terdiri dari model tipe inkremental. Model inkremental bertentangan dengan prinsip *waterfall* dalam mengembangkan sistem dalam proses *singlepass*, dengan dokumentasi yang ketat dan tahap pengujian yang ekstensif, untuk menghasilkan produk akhir yang dapat digunakan sepenuhnya pada akhirnya. Model inkremental malah mengusulkan pengembangan sistem dalam proses *build* atau inkremen yang berurutan. Dengan setiap *build*, sistem dirancang dan dikembangkan, dan versi kerja atau *prototype* diimplementasikan. *User* kemudian dapat mengujinya secara aktif, dalam konteks kerja, dan memberikan umpan balik yang berharga. Umpan balik ini kemudian akan digunakan sebagai titik awal untuk membangun berikutnya. Dengan setiap *build* yang berurutan, sistem menjadi lebih lengkap, lebih fungsional, dan lebih dekat dengan apa yang diinginkan user [16].

Tabel 1 Perbandingan Model *Waterfall* dan *Prototype*

Tahapan Development	***Waterfall***	***Prototype***
System Planning	Berdasarkan Kebutuhan	Berdasarkan kebutuhan
System Analysis	Kebutuhan data harus dianalisis diawal dengan lengkap dan menyeluruh	Kebutuhan data dapat ditambah ataupun dikurangi sesuai kebutuhan *user* ketika dilakukan *Testing*
	Perubahan data maupun fungsional akan merubah keseluruhan proses pada tahapan berikutnya	Perubahan dapat dilakukan selama sistem atau *software* masih dalam bentuk *prototype*
System Desain	*Testing* dilakukan ketika semua tahapan pada model sudah selesai	*Testing* dapat dilakukan ketika *prototype* telah dibangun sehingga hasil *Testing* dapat merubah rancangan sistem
	Tidak dapat memberikan gambaran jelas mengenai sistem yang dibangun karena sistem bisa dilihat jika tahapan sudah dilakukan	Memberikan *prototype* sebagai gambaran sistem yang akan dibangun sehingga *user* dapat melihat dan berinteraksi langsung dengan gambaran sistem
		User berperan aktif dalam pengembangan sistem
		Sistem yang dibangun akan sesuai dengan keinginan *user*
System Implementation	Menerapkan proses perancangan yang baik	Tidak menerapkan proses perancangan yang baik
	Evaluasi dilakukan ketika sistem telah dibangun	Evaluasi dilakukan ketika *prototype* telah dibangun
	Mengedepankan kebutuhan fungsional sistem	Mengedepankan aspek kenyamanan *user*
System Maintenance	Dilakukan sesuai kesepakatan	Dilakukan sesuai kesepakatan

Penelitian ini akan berfokus pada evaluasi performa dari *container* yang berisikan *image* nginx:1.23 yang berjalan pada *cloud service* berbasis Kubernetes yaitu Google Kubernetes Engine (GKE). Adapun beberapa tahapan yang harus dilalui dalam penelitian ini. Metode SDLC dengan model waterfall memiliki tahapan-tahapan yang dapat dilihat dalam gambar 1



Gambar 1 Metedologi Penelitian dengan menggunakan SDLC model waterfall

2.1 Tahap Analisis Kebutuhan

Untuk melakukan penelitian ini ada beberapa hal yang harus disiapkan seperti Laptop/PC, akun GCP dan Internet. Penulis menggunakan laptop Lenovo dengan spesifikasi yang dapat dilihat pada tabel 2

Tabel 2 Spesifikasi Laptop Yang Digunakan

OS	Processor	CPU	*Memory*
Windows 10 Home Single Language 64-bit	AMD 3020e with Radeon Graphics	2 CPUs 1,2GHz	8192MB RAM

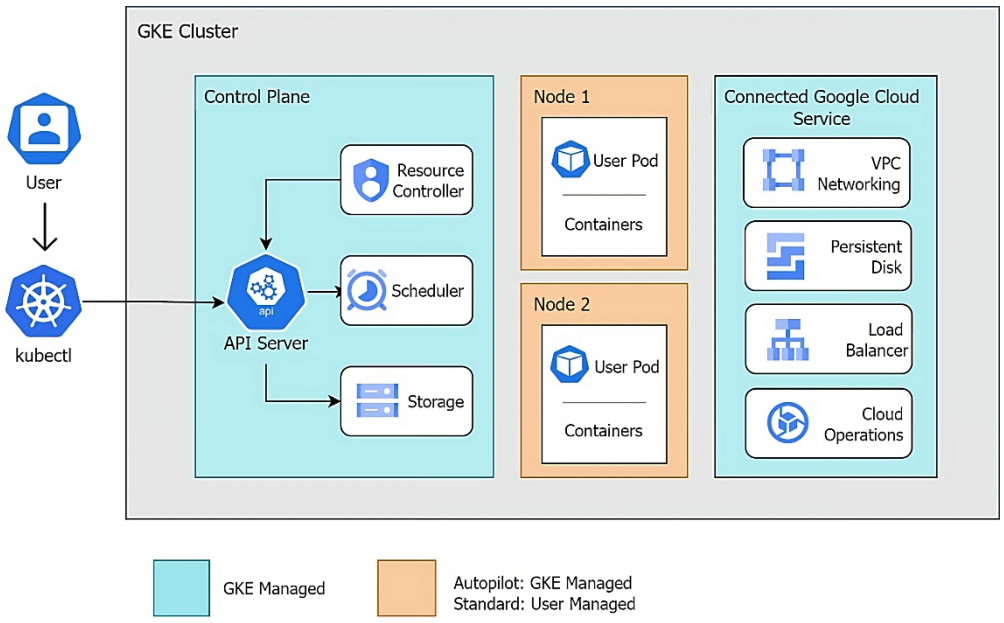
Sedangkan untuk penelitian ini, GKE akan memiliki spesifikasi mesin yang terdiri dari 4 buah CPU dan RAM sebesar 16 GiB dan dapat dilihat pada tabel 3

Tabel 3 Spesifikasi Instance Machine Yang Digunakan

Cloud	*Instance* type	*Instance* size	OS	CPU	vCPUs	RAM	Harga Per jam	*Network*	K8s ver	Location
GKE	General Purpose	e2-standard-4	*Container-Optimized OS*	Intel Xeon CPU @ 2.20GHz	4	16GB	$0.134	*Network Standard Service Tier*	1.27.2-gke.1200	Us-central1

2.2 Tahap Desain Sistem

Pada langkah ini akan dilakukan observasi guna mengetahui desain dari arsitektur GKE. Arsitektur GKE dapat dilihat pada Gambar 2.



Gambar 2 Arsitektur GKE

2.3 Tahap Implementasi Desain

Setelah mengetahui arsitektur desain GKE, selanjutnya adalah tahap implementasi untuk mewujudkan desain tersebut. Implementasi dimulai dengan pembuatan *cluster*, setelah *cluster* siap maka tahap selanjutnya adalah *deploy images*. *Images* yang dipakai adalah nginx:1.23 yang berasal dari repositori docker yaitu docker.io. Proses deploy berhasil apabila *container* ketika di cek dalam status *running*

2.4 Tahap Integration dan Testing Sistem

Pada tahap ini akan dilakukan *benchmarking* dan *observability*. *Benchmarking* akan dilakukan dengan python timeit yang berfungsi sebagai metode untuk mengukur *execution time* yang diambil oleh *code snippet*. Pengujian ini akan dilakukan dengan jumlah *execution code* sebanyak 50.000 kali dengan *code* seperti pada gambar 3 berikut

```
paldavidi37@cloudshell:~ (sigma-bay-390707)$ python -m timeit -n 50000 -u sec -v "'-'.join([str(i) for i in range(100)])"
raw times: 0.542 sec, 0.537 sec, 0.548 sec, 0.531 sec, 0.53 sec

50000 loops, best of 5: 1.06e-05 sec per loop
```

Gambar 3 Python *timeit* untuk mengukur *execution time*

Sedangkan *observability* akan dilakukan dengan bantuan Prometheus. Prometheus akan mengumpulkan *metrics* dari *service* yang berjalan pada *container* di GKE *cluster*

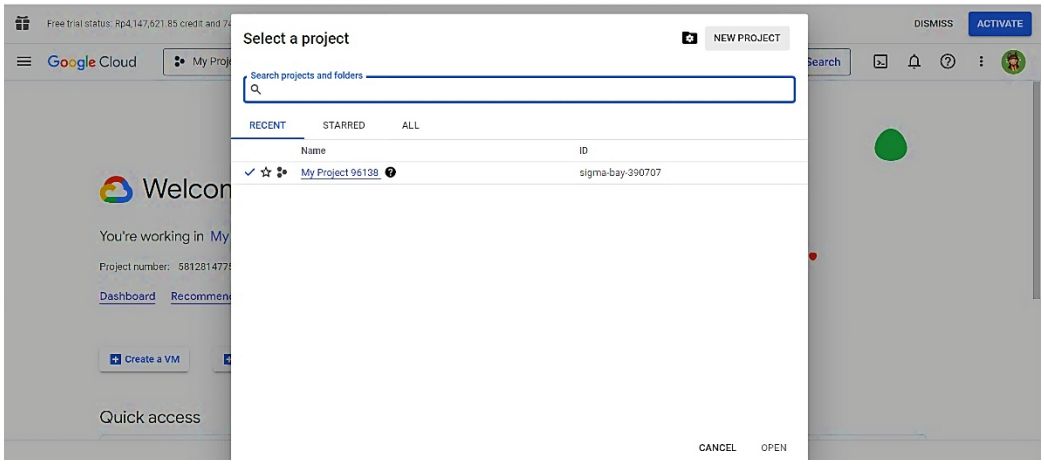
2.5 Tahap Operation dan Maintanance

Setelah mendapatkan data yang dibutuhkan melalui proses *testing*, GKE *cluster* akan dihapus untuk menghindari beban pembiayaan

3. HASIL DAN PEMBAHASAN

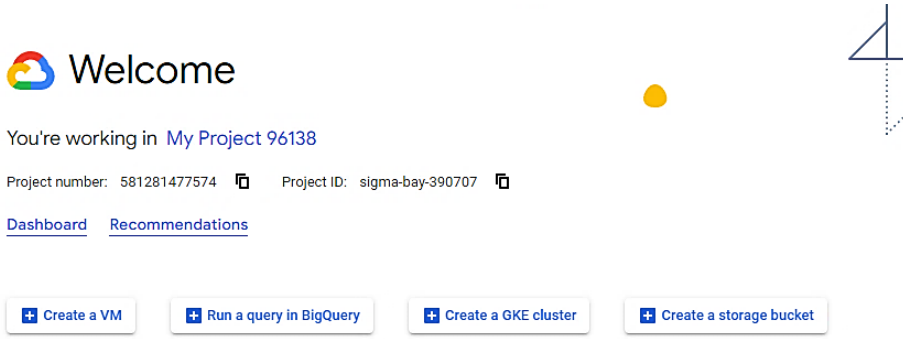
3.1 Deploy Aplikasi

1. Menyiapkan *Environment*
   a. *Sign In* dan masuk dengan kredensial akun Google
   b. Pilih Proyek yang sudah ada atau jika belum klik buat proyek baru



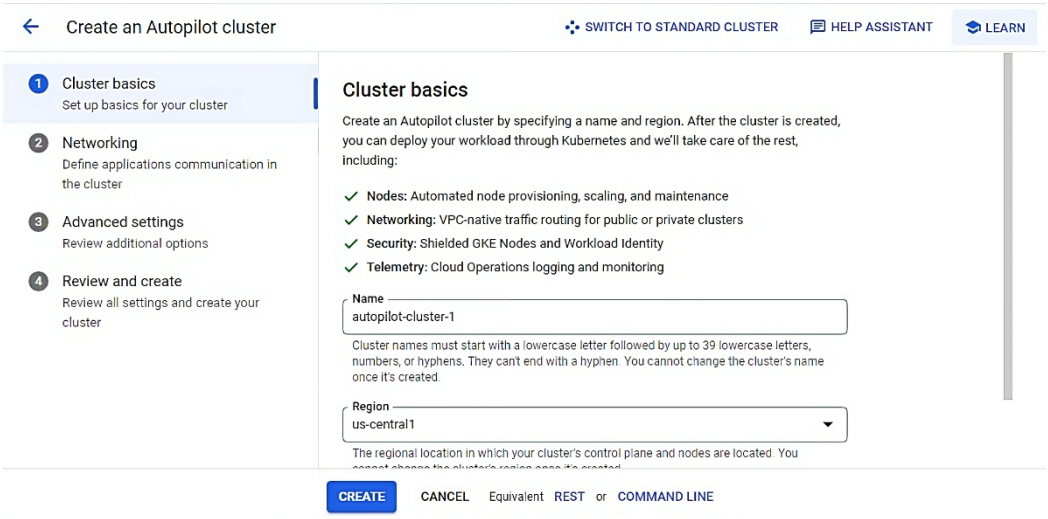
Gambar 4 *Select* Proyek GKE

2. Membuat GKE *Cluster*
   a. Buka laman proyek yang sudah dibuat lalu klik *Create a gke cluster*



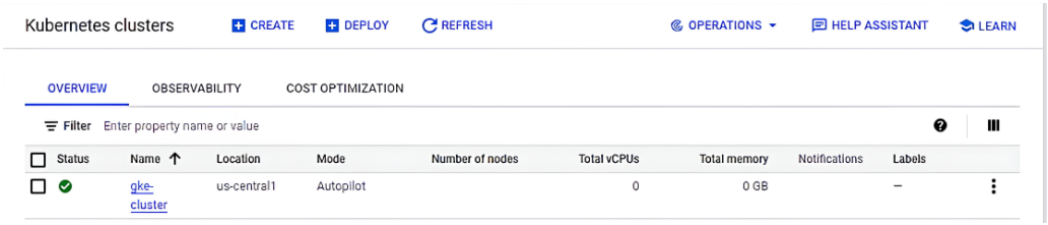
Gambar 5 Laman Proyek GKE

b. Buat *cluster* Kubernetes seperti gambar 6 berikut



Gambar 6 *Create Cluster* GKE

c. Tunggu proses selesai maka *cluster* akan terlihat pada *dashboard* seperti gambar 7 berikut



Gambar 7 *Cluster* GKE *Deployed*

d. Verifikasi *connection* ke *cluster* yang telah dibuat untuk mendapatkan daftar dari nodes *cluster*

```
paldavidi37@cloudshell:~ (sigma-bay-390707)$ kubectl get nodes
NAME                                                STATUS   ROLES    AGE   VERSION
gk3-gke-cluster-default-pool-8ea11080-pz6r   Ready    <none>   58m   v1.27.2-gke.1200
gk3-gke-cluster-default-pool-ee2d3b12-3g93   Ready    <none>   61m   v1.27.2-gke.1200
```

Gambar 8 GKE Nodes

3. *Deploy* Aplikasi
   a. *Connect to cluster*

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to sigma-bay-390707.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
paldavidi37@cloudshell:~ (sigma-bay-390707)$ gcloud container clusters get-credentials gke-cluster \
    --location us-central1
Fetching cluster endpoint and auth data.
kubeconfig entry generated for gke-cluster.
```

Gambar 9 *Connect Cluster* GKE

b. *Deploy* aplikasi dengan *command* seperti berikut lalu *expose deployment* ke internet agar *user* bisa mengaksesnya

```
paldavidi37@cloudshell:~ (sigma-bay-390707)$ kubectl create deployment gke-server \
    --image=nginx:1.23
Warning: autopilot-default-resources-mutator:Autopilot updated Deployment default/gke-server:
autopilot-defaults)
deployment.apps/gke-server created
paldavidi37@cloudshell:~ (sigma-bay-390707)$ kubectl expose deployment gke-server \
    --type LoadBalancer \
    --port 80 \
    --target-port 80
service/gke-server exposed
```

Gambar 10 *Deploy* Aplikasi di GKE

c. Lihat semua *resources* yang ada di *namespace* tempat ter*deploy*nya aplikasi

```
paldavidi37@cloudshell:~ (sigma-bay-390707)$ kubectl get all -n default
NAME                              READY   STATUS    RESTARTS   AGE
pod/gke-server-567d5d6858-4drss   1/1     Running   0          7m11s

NAME                 TYPE           CLUSTER-IP    EXTERNAL-IP      PORT(S)        AGE
service/gke-server   LoadBalancer   10.78.2.54    34.136.169.106   80:30713/TCP   6m41s
service/kubernetes   ClusterIP      10.78.0.1     <none>           443/TCP        19m

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/gke-server     1/1     1            1           7m12s

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/gke-server-567d5d6858     1         1         1       7m12s
```

Gambar 11 Informasi *Cluster* GKE

d. Lihat detail dari salah satu pod yang terdaftar di *output* pada *namespace*

```
paldavidi37@cloudshell:~ (sigma-bay-390707)$ kubectl describe pod gke-server-567d5d6858-4drss
Name:             gke-server-567d5d6858-4drss
Namespace:        default
Priority:         0
Service Account:  default
Node:             gk3-gke-cluster-pool-1-07f77f8c-jf9p/10.128.0.26
Start Time:       Mon, 17 Jul 2023 21:28:20 +0000
Labels:           app=gke-server
                  pod-template-hash=567d5d6858
Annotations:      <none>
Status:           Running
SeccompProfile:   RuntimeDefault
IP:               10.77.128.136
IPs:
  IP:           10.77.128.136
Controlled By:  ReplicaSet/gke-server-567d5d6858
Containers:
  nginx:
    Container ID:   containerd://2b0dc22ba59dabe0a93adef7340c9e32ad7ad3125399516db9a1305674f792c5
    Image:          nginx:1.23
    Image ID:       docker.io/library/nginx@sha256:f5747a42e3adcb3168049d63278d7251d91185bb5111d2563d58729a5c9179b0
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Mon, 17 Jul 2023 21:29:26 +0000
    Ready:          True
    Restart Count:  0
```

Gambar 12 Detail pod GKE 1

```
    Limits:
      cpu:                500m
      ephemeral-storage:  1Gi
      memory:             2Gi
    Requests:
      cpu:                500m
      ephemeral-storage:  1Gi
      memory:             2Gi
    Environment:          <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-sxzgs (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  kube-api-access-sxzgs:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   Guaranteed
Node-Selectors:              <none>
Tolerations:                 kubernetes.io/arch=amd64:NoSchedule
                             node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
```

Gambar 13 Detail pod GKE 2

```
Events:
  Type     Reason                  Age    From                                   Message
  ----     ------                  ----   ----                                   -------
  Warning  FailedScheduling        4m30s  gke.io/optimize-utilization-scheduler  0/2 nodes are available: 2 Insufficient cpu, 2 Insufficient memory. preemption: 0/2 nodes
 are available: 2 No preemption victims found for incoming pod..
  Normal   TriggeredScaleUp        4m23s  cluster-autoscaler                     pod triggered scale-up: [{https://www.googleapis.com/compute/v1/projects/sigma-bay-390707
/zones/us-central1-a/instanceGroups/gk3-gke-cluster-pool-1-07f77f8c-grp 0->1 (max: 1000)}]
  Normal   Scheduled               2m59s  gke.io/optimize-utilization-scheduler  Successfully assigned default/gke-server-567d5d6858-4drss to gk3-gke-cluster-pool-1-07f77
f8c-jf9p
  Warning  FailedCreatePodSandBox  2m24s  kubelet                                Failed to create pod sandbox: rpc error: code = Unknown desc = failed to setup network fo
r sandbox "031c7f09410e6f15e8d84d9c62f2344705b17ca20be40dd78ffc7a9d8bf96e25": plugin type="cilium-cni" failed (add): unable to connect to Cilium daemon: failed to create
cilium agent client after 30.000000 seconds timeout: Get "http:///var/run/cilium/cilium.sock/v1/config": dial unix /var/run/cilium/cilium.sock: connect: no such file or d
irectory
Is the agent running?
  Normal  Pulling  2m5s  kubelet  Pulling image "nginx:1.23"
  Normal  Pulled   115s  kubelet  Successfully pulled image "nginx:1.23" in 9.753943443s (9.754189253s including waiting)
  Normal  Created  115s  kubelet  Created container nginx
  Normal  Started  113s  kubelet  Started container nginx
```

Gambar 14 Detail pod GKE 3

e. Lihat *service* yang telah di-*deploy*

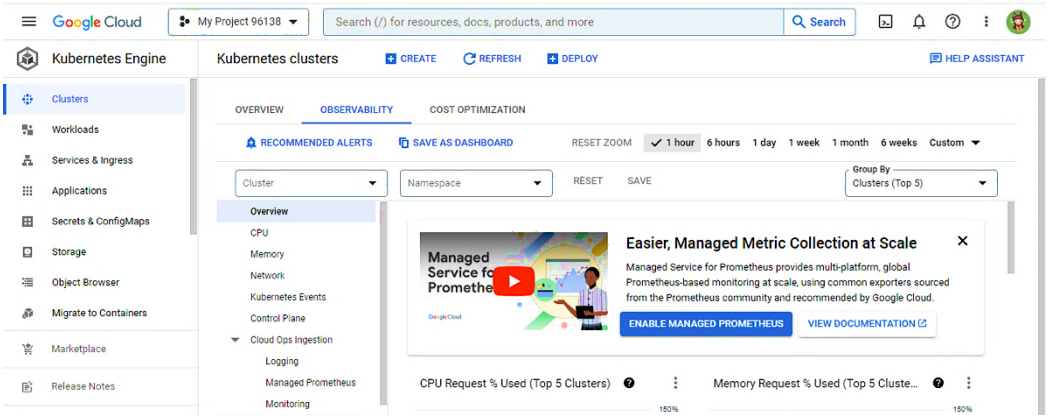
```
paldavidi37@cloudshell:~ (sigma-bay-390707)$ kubectl get service gke-server
NAME         TYPE           CLUSTER-IP    EXTERNAL-IP      PORT(S)        AGE
gke-server   LoadBalancer   10.78.2.54    34.136.169.106   80:30713/TCP   5m29s
```

Gambar 15 *Service* didalam *namespace* GKE

f. Buka External-IP untuk memvalidasi

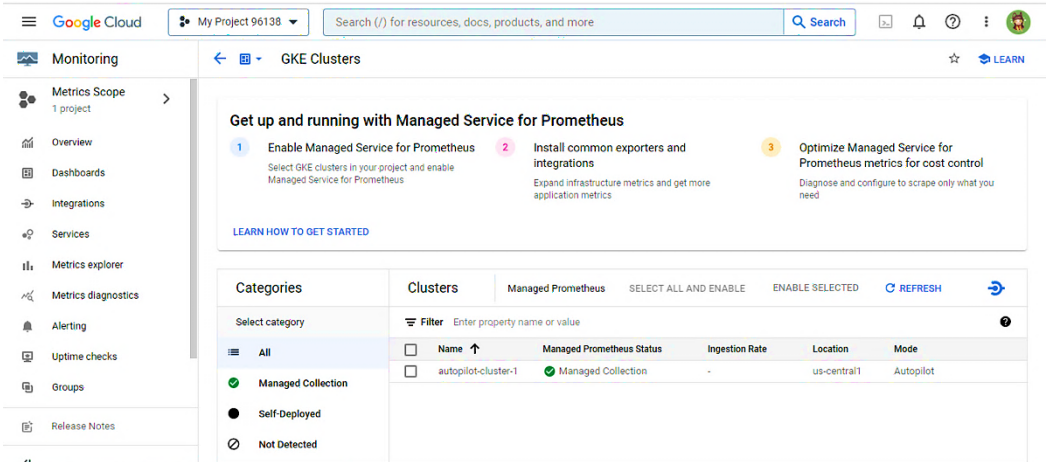
Gambar 16 Membuka External-IP GKE

4. *Connect* Prometheus dan Ambil data
   a. Pada menu *clusters*, klik *Observability*



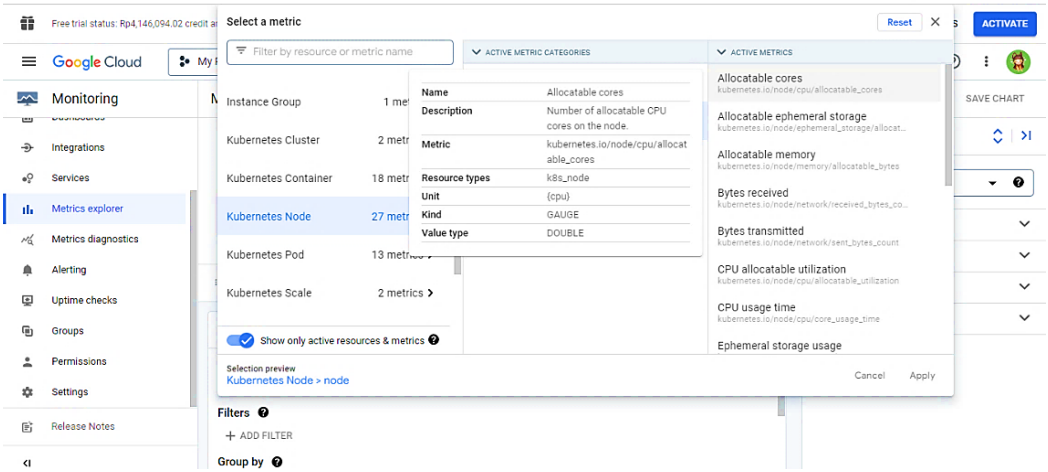
Gambar 17 Menu *Observability*

   b. Klik *enable managed* Prometheus



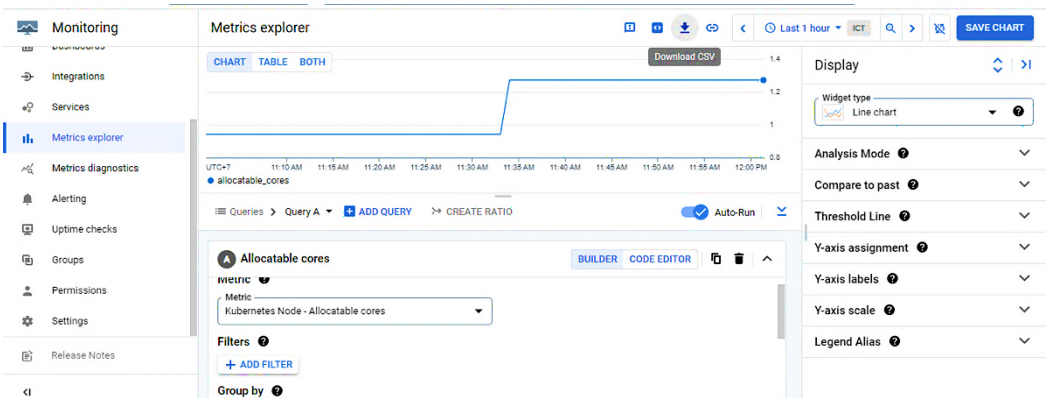
Gambar 18 *Enable* Prometheus

   c. Buka menu *Monitoring*, klik *metrics explorer*
   d. Pilih *metrics* yang mau dilihat

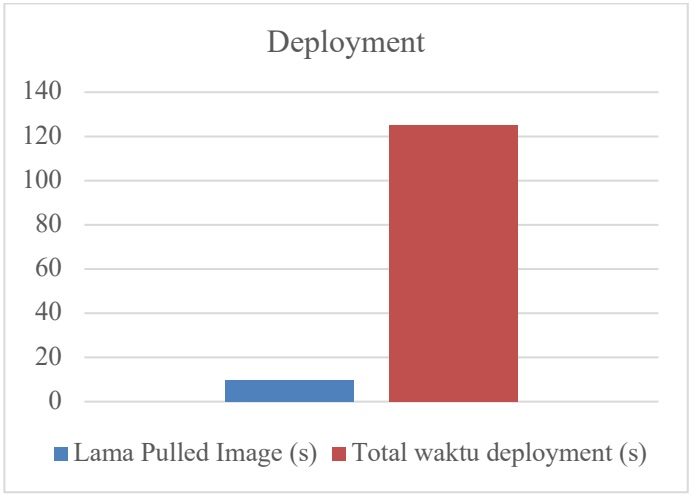


Gambar 19 Pilih *Metrics* yang ingin dilihat GKE

e. Untuk mengambil data, *download* csv pada simbol panah kebawah



Gambar 20 Ambil Data GKE



Gambar 21 Grafik lama waktu deployment

Dari proses *deploy* aplikasi yang dapat dilihat pada Gambar 14, terdapat suatu *event* yang menunjukan lama waktu *pulled image* hingga berjalannya *container*. Lama waktu *deployment* dirangkum pada tabel 4

Tabel 4 Waktu Deployment

Lama Pulled Image (s)	**Total waktu *deployment* (s)**
9,77	125

Dari segi kecepatan *deployment* aplikasi, GCP membutuhkan total waktu sebanyak 125s dengan lama *pulled images* sebesar 9,77s.

3.2 CPU

Tabel 5 Informasi CPU

GCP	
CPU family	23
Model	49
Thread(s) per core	2
Core(s) per socket	2
socket	1
CPU MHz	2249,998
BogoMIPS	4399,99

CPU MHz berasal dari teknologi variabel *clock* dimana CPU melakukan beban rendah untuk menghemat daya. Namun angka MHz yang tertera saja tidak lantas memberi tahu tentang seberapa baik CPU mengeksekusi *code*. Hal ini dikarenakan hampir semua CPU saat ini adalah *superscalar*. Maksudnya jika memungkinkan mereka akan mengeksekusi lebih dari satu instruksi per *cycle*. Oleh karena itu dibutuhkan suatu *benchmarking execution time. Execution time* dilakukan dengan Python dan menghasilkan luaran seperti tabel 6 berikut

Tabel 6 Execution Time GKE

***Execution* Time GKE**		
Num of *execution*	**Raw times**	**Best of 5**
50000	0,542 sec 0,537 sec 0,548 sec 0,531 sec 0,53 sec	1,06E-05 sec per loop

GCP menggunakan CPU dengan frequensi 2.20 GHz atau 2200 MHz dan berjalan pada 2249,998 MHz artinya beban berjalan dalam keadaan tinggi dan dalam pengujian *execution time* ini menghasilkan waktu terbaik sebesar 0,0000106 *second per loop* yang bisa dikategorikan sangat cepat.

3.3 *Memory*

Tabel 7 Memory Metrics

Prometheus Metrics	**Hasil (avg)**

Kubernetes *container – memory limit*	196,829MiB
Kubernetes *container – memory request*	86,661MiB
Kubernetes *container – memory usage*	18,362MiB

Berdasarkan *observability* dan pengambilan data melalui Prometheus didapatkanlah *metrics* dan besaran *memory* yang terdiri dari *memory limit* sebesar 196,829 MiB, *memory request* sebesar 86,661 MiB, dan *memory usage* sebesar 18,362 MiB.

3.4 *Network*

Tabel 8 Network Metrics

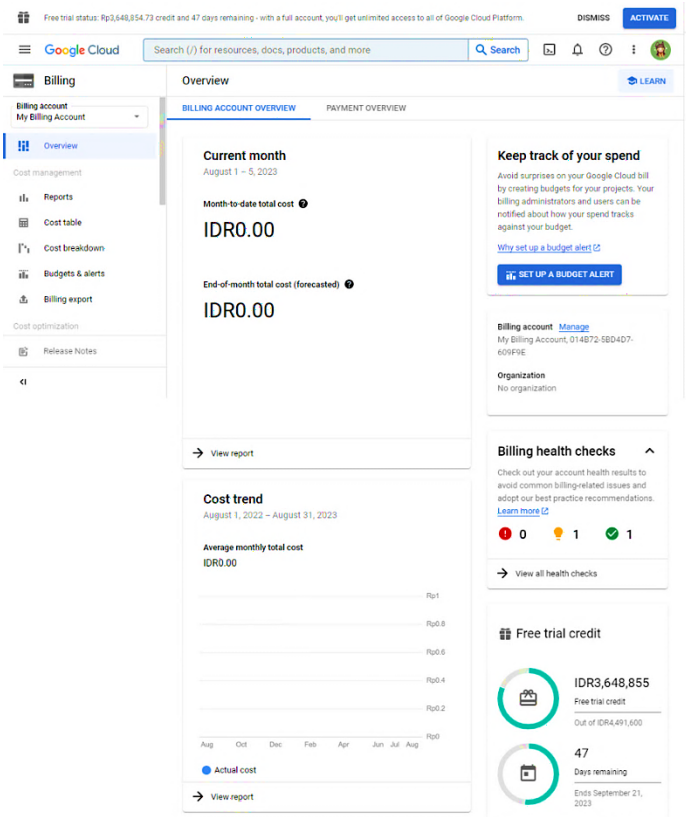
Prometheus Metrics	Hasil (avg)
Kubernetes nodes – *bytes received*	16,015KiB/s
Kubernetes nodes – *bytes transmitted*	16,57KiB/s
Kubernetes pod – *bytes received*	6,034KiB/s
Kubernetes pod – *bytes transmitted*	5,891KiB/s

Berdasarkan *observability* dan pengambilan data melalui Prometheus didapatkanlah *metrics* dan besaran *network* yang terdiri dari *node bytes received* sebesar 16,015 KiB/s, *node bytes transmitted* sebesar 16,057 KiB/s, *pod bytes received* sebesar 6,034KiB/s, dan *pod bytes transmitted* sebesar 5,891 KiB/s.

Node bytes received mengacu pada jumlah total *bytes* yang diterima oleh sebuah node. *Node bytes transmitted* mengacu pada jumlah total *bytes* yang dikirim oleh sebuah node. *Pod bytes received* mengacu pada jumlah total *bytes* yang diterima oleh sebuah pod. Dan *Pod bytes transmitted* mengacu pada jumlah total *bytes* yang dikirim oleh sebuah pod. *Bytes* yang dimaksud dapat mencakup *ingress* maupun *egress traffic*. Hal ini mewakili keseluruhan data yang ditransfer terlepas dari sumber atau tujuannya.

3.5 *Analisis Cost*

GCP menawarkan *credit* sebesar $300 untuk pelanggan baru selama 90 hari. Maka selama batas waktu tersebut user bebas menggunakan semua fitur GCP yang disediakan dengan maksimal *credit* $300 dan selama *credit* masih tersedia *user* tidak perlu membayar *payment* atau bisa dikatakan gratis.



Gambar 22 Billing payment

Gambar 22 merupakan *billing overview*. Terlihat bahwa *credit* yang didapatkan sebesar Rp 4.491.000 dan tersisa Rp 3.648.855 dengan waktu pemakaian selama 43 hari dan tersisa 47 hari waktu *free trial*.

Tabel 9 Price Performance

Performance/Price (MIPS/$)	**CPU Benchmark (MIPS)**	**Harga ($)**
952,4	4399,99	4,62

Price performance berasal dari *performance* yang diambil berdasarkan BogoMIPS dibagi harga yang berasal dari pemakaian GCP selama 3 hari yang merupakan waktu selama implementasi hingga *testing container*. Hasilnya GCP memiliki *value price performance* sebesar 952,4 MIPS/$.

3.6 *Perbandingan implementasi Kubernetes di GCP, AWS, dan Azure*

Bab ini mengevaluasi dan membandingkan implementasi Kubernetes di tiga penyedia layanan cloud utama yaitu Google Cloud Platform (GCP), Amazon Web Services (AWS), dan Microsoft Azure. Perbadingan akan berfokus pada layanan Kubernetes masing-masing platform, yakni Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes Service (EKS), dan Azure Kubernetes Service (AKS).

Perbandingan dilakukan berdasarkan kriteria utama seperti layanan terkelola, kemudahan penggunaan, harga, keamanan, customisasi, dan integrasi dengan layanan lain. Penyajian informasi ini memberikan landasan untuk pengambilan

keputusan yang informasional dan kontekstual mengenai GKE, EKS, dan AKS dalam implementasi Kubernetes.

Tabel 10 Perbandingan Implementasi Kubernetes

Kriteria	Google Kubernetes Engine (GKE)	Amazon Elastic Kubernetes Service (EKS)	Azure Kubernetes Service (AKS)
Layanan Terkelola	Penawaran Kubernetes asli dari Google	Dibangun di atas platform cloud AWS	Dibangun di atas platform cloud Azure
Kemudahan Penggunaan	Dianggap paling mudah dengan antarmuka yang sederhana	Memerlukan konfigurasi manual lebih banyak	Mudah digunakan dengan fitur seperti autoscaling
Harga	Harga bervariasi; lebih tinggi untuk beberapa fitur	Harga bervariasi; biaya lebih tinggi untuk beberapa fitur tertentu	Umumnya bersaing; menawarkan opsi *cluster* gratis
Keamanan	Fitur keamanan yang kuat, integrasi dengan IAM, otorisasi biner	Fitur keamanan yang kuat, integrasi dengan sistem manajemen akses identitas (IAM), enkripsi sertifikat	Fitur keamanan yang kuat, integrasi dengan IAM, enkripsi sertifikat
Customisasi	Sangat dapat disesuaikan, memungkinkan penyesuaian konfigurasi *cluster*	Beberapa kemungkinan penyesuaian, kompleksitas tambahan	Beberapa kemungkinan penyesuaian, lebih mudah diimplementasikan
Integrasi dengan Layanan	Integrasi mulus dalam ekosistem Google Cloud	Terintegrasi dengan layanan AWS, mungkin memerlukan konfigurasi tambahan	Terintegrasi dengan layanan Azure, mungkin memerlukan penyiapan tambahan

4. KESIMPULAN

Docker merupakan teknologi *container* paling populer di tahun 2017 namun dikarenakan memiliki beberapa kekurangan dan fitur yang tidak bisa diberikan, Kubernetes diciptakan sebagai salah satu solusi dalam sistem *containerisasi*. Kubernetes dapat menjalankan Docker *Container* dan *Workloads* sehingga dapat mengatasi beberapa kompleksitas operasi saat berpindah ke penskalaan beberapa *container* yang diterapkan pada beberapa *server*. Kubernetes juga sangat *flexible*, bisa berjalan secara *self-hosted* contohnya dapat dibangun di *local computer user* dan juga bisa berjalan secara *hosted* atau dengan kata lain dapat berjalan pada layanan terkelola seperti Cloud Service Provider (CSPs). Penelitian untuk menganalisis performa pun dilakukan pada salah satu CSPs bernama Google Cloud Platform (GCP) pada Kubernetes servicenya yang bernama Google Kubernetes Engine (GKE). Hasilnya waktu yang diperlukan untuk *pulled image* nginx:1.23 adalah selama 9,77 detik dengan total waktu *deployment* selama 125 detik. Performa pada CPU dilakukan dengan melakukan *execution time* dengan 50.000 kali jumlah eksekusi menghasilkan cpu berjalan dalam keadaan beban tinggi dan dalam pengujian *execution time* ini menghasilkan waktu terbaik sebesar 0,0000106 *second per loop*. Sedangkan untuk kategori *memory* dan *network* di *obervability* melalui Prometheus hingga didapatkanlah *metrics* dan besaran *memory* yang terdiri dari *memory limit* sebesar 196,829 MiB, *memory request* sebesar 86,661 MiB,

memory usage sebesar 18,362 MiB, *node bytes received* sebesar 16,015 KiB/s, *node bytes transmitted* sebesar 16,057 KiB/s, *pod bytes received* sebesar 6,034KiB/s, dan *pod bytes transmitted* sebesar 5,891 KiB/s. Adapun *value price performance* yang didapatkan GCP yaitu sebesar 952,4 MIPS/$.

REFERENSI

[1] R. Capuano dan H. Muccini, "A Systematic Literature Review on Migration to Microservices: a Quality Attributes perspective," *IEEE*, Mei 2022, doi: 10.1109/ICSA-C54293.2022.00030.

[2] C. Johnson, "2021 Microservices Developer Report," 2021. [Daring]. Tersedia pada: https://www.jrebel.com/blog/2021-microservices-developer-report

[3] S. Newman, *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. O'Reilly Media, Inc., 2019.

[4] L. S. Vailshery, "Most frequently used container technologies worldwide as of March 2017," April 2017. [Daring]. Tersedia pada: https://www.statista.com/statistics/588771/worldwide-container-technology-use/

[5] C. Arango, R. Dernat, dan J. Sanabria, "Performance Evaluation of Container-based Virtualization for High Performance Computing Environments." arXiv, 28 September 2017. Diakses: 8 Januari 2023. [Daring]. Tersedia pada: http://arxiv.org/abs/1709.10140

[6] Preeth E N, Fr. J. P. Mulerickal, B. Paul, dan Y. Sastri, "Evaluation of Docker containers based on hardware utilization," dalam *2015 International Conference on Control Communication & Computing India (ICCC)*, Trivandrum, Kerala, India: IEEE, Nov 2015, hlm. 697–700. doi: 10.1109/ICCC.2015.7432984.

[7] D. Jamil, "Docker-An Overview/Pros and Cons," *linkedIn*, 23 April 2022. https://www.linkedin.com/pulse/docker-an-overviewpros-cons-danish-jamil/ (diakses 15 Agustus 2023).

[8] R. Muddinagiri, S. Ambavane, dan S. Bayas, "Self-Hosted Kubernetes: Deploying Docker Containers Locally With Minikube," dalam *2019 International Conference on Innovative Trends and Advances in Engineering and Technology (ICITAET)*, SHEGAON, India: IEEE, Des 2019, hlm. 239–243. doi: 10.1109/ICITAET47105.2019.9170208.

[9] B. Doerrfeld, "2022 in Review: Kubernetes' Big Year," 4 Januari 2023. [Daring]. Tersedia pada: https://cloudnativenow.com/features/2022-in-review-kubernetes-big-year/#:~:text=Going%20Mainstream,like%20EKS%2C%20AKS%20or%20GKE.

[10] A. Pereira Ferreira dan R. Sinnott, "A Performance Evaluation of Containers Running on Managed Kubernetes Services," dalam *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Sydney, Australia: IEEE, Des 2019, hlm. 199–208. doi: 10.1109/CloudCom.2019.00038.

[11] M. A. Nugroho, "Analisis Cluster Container Pada Kubernetes Dengan Infrastruktur Google Cloud Platform," *JIPI J. Ilm. Penelit. Dan Pembelajaran Inform.*, vol. 3, no. 2, Des 2018, doi: 10.29100/jipi.v3i2.651.

[12] M. N. Birje dan C. Bulla, "Commercial and Open Source Cloud Monitoring Tools: A Review," dalam *Advances in Decision Sciences, Image Processing, Security and Computer Vision*, S. C. Satapathy, K. S. Raju, K. Shyamala, D. R. Krishna, dan M. N. Favorskaya, Ed., dalam Learning and Analytics in Intelligent Systems, vol. 3. Cham: Springer International Publishing, 2020, hlm. 480–490. doi: 10.1007/978-3-030-24322-7_59.

[13] F. Reinartz, "prometheus," *GitHub*, 2023. https://github.com/prometheus/prometheus (diakses 24 Juli 2023).

[14] P. Rysak, "Comparative analysis of C and Python on the basis of the execution time of applications implementing selected algorithms," *J. Comput. Sci. Inst.*, 2023.

[15] P. Isaias dan T. Issa, *High Level Models and Methodologies for Information Systems*. New York, NY: Springer New York, 2015. doi: 10.1007/978-1-4614-9254-2.

[16] V. Massey dan K. Satao, "Comparing various SDLC models and the new proposed model on the basis of available methodology," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 2(4), hlm. 170–177, 2012.