E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 03, Nomor 02, Desember 2018 : 84 - 93

ANALISIS CLUSTER CONTAINER PADA KUBERNETES DENGAN INFRASTRUKTUR GOOGLE CLOUD PLATFORM

M. Agung Nugroho[1)], Cuk Subiyantoro[2)]
[1, 2)]Teknik Informatika STMIK AKAKOM
Jl. Raya Janti No.143, Banguntapan, Bantul, Yogyakarta
e-mail: m.agung.n@akakom.ac.id [1)], cuks@akakom.ac.id [2]

ABSTRAK

Container merupakan teknologi virtualisasi terbaru, dengan container memudahkan system administrator dalam mengelola aplikasi pada server. Docker container dapat digunakan untuk membangun, mempersiapkan, dan menjalankan aplikasi. Dapat membuat aplikasi dari bahasa pemrograman yang berbeda pada lapisan apapun. Kubernetes digunakan untuk memanajemen container dalam jumlah besar untuk membangun layanan microservices. Orchestration tools ini menyediakan performansi fitur untuk menjaga container tetap berjalan dengan teknik replication control dan autoscale pada container. Penelitian ini melakukan uji komparasi performansi pada layanan yang menggunakan container sebelum dan sesudah menerapkan model kubernetes, dimana proses pengujian dilakukan pada tingkat availability services di layanan yang menggunakan google cloud platform. Ujicoba dilakukan dengan menggunakan locus.io dengan memberikan beban request sampai dengan 25600 pada rancangan arsitektur layanan kubernetes. Hasil pengujian menunjukkan bahwa tingkat availability mengalami peningkatan dengan penurunan hasil failure request.

Kata kunci : *docker, container, kubernetes, google cloud platform*

ABSTRACT

Container is new virtualization technology that uses OS-Level virtualization method for deploying and running application in distributed environment. Container have portability feature which is help system administrator easy to maintain application in server. This container also can run on different type of application stack. Kubernetes is an orchestration tools which help the container to maintain the number of containers for providing microservices infrastructure. Kubernetes also have replication controller and autoscaling features which can replace the failures container. This research compares services performaces based on availability services of container before and after using kubernetes in google cloud platform. Locus.io is an stress tools that used in the research to give simulation request in the services which uses kubernetes. The result shows that failures request in availability services decreased after the kubernetes implemented.

Keywords: *docker, container, kubernetes, performance testing, cloud computing*

I. PENDAHULUAN

Penerapan cluster dalam memberikan manfaat dalam menangani proses komputasi yang tinggi. Keterbatasan dari perangkat yang digunakan sekalipun dalam jumlah banyak tetap tidak dapat memenuhi kebutuhan performansi dari infrastruktur sebuah layanan. Raspberry Pi Cluster yang memiliki keterbatasan sumber daya, dengan 14 node dapat menangani dan menyelesaikan pekerjaan dengan performance yang kurang baik. Hal ini normal karena sumber daya yang terbatas akan berdampak waktu eksekusi. Arsitektur Core i5 pada Mac OS X yang digunakan sebagai pembanding memiliki sumber daya lebih dapat diandalkan dan menghasilkan kinerja yang lebih dari Raspberry Pi Cluster. Arsitektur Core i7 memiliki kinerja terbaik terutama ketika menjalankan matriks. Arsitektur high performance computing yang telah dibangun dapat memberikan pelajaran pada pengembangan model arsitektur HPC, dan menjadi baseline performance. Di masa depan, akan digunakan untuk menentukan model arsitektur HPC dan bisa dilakukan banyak variasi model test [1].

Mekanisme lain yang dapat digunakan untuk cluster adalah dengan melakukan cluster pada teknologi container, dimana container ini dapat dimanajemen oleh perangkat tools lain seperti kubernetes, shipyard, dan apache mesos. Teknologi ini memungkinkan untuk membuat container cluster didalam satu mesin server yang sama dan di cluster dengan mesin server yang lain. Mesin-mesin server dapat berada pada satu provider cloud atau lebih. Hal ini karena *Docker container* tidak menggunakan *hardware* untuk virtualisasi. Program berjalan dalam *docker container* berhubungan langsung dengan *linux kernel* pada *host operating system*. Karena tidak ada tambahan

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 03, Nomor 02, Desember 2018 : 84 - 93

lapisan antara program yang berjalan didalam *container* dengan sistem operasi pada komputer, sehingga tidak ada *resources* yang habis karena redudansi aplikasi atau simulasi *virtual hardware*. *Container* memungkinkan mengisolasi lingkungan program, sehingga program dapat berjalan tanpa gangguan dari permasalahan di sistem operasi. Dalam perkembangannya, kemudahan pengelolaan *container* menjadi basis untuk melakukan skalabilitas [2].

Container sangat ideal untuk mengemas layanan microservices karena mengisolali package aplikasi, library, dan sistem operasi, selain itu container berukuran kecil dan tidak mengalami overhead seperti yang ditemukan pada virtual machine ketika melakukan proses deployment pada microservices. Pengalokasian virtual machine untuk microservice akan membutuhkan biaya yang tinggi, sehingga membuat container ideal untuk deployment pada layanan cloud. Kubernetes adalah container orchestration yang berfungsi untuk memastikan seluruh container dalam berjalan dengan baik dalam memproses workload yang berjalan pada mesin fisik atau mesin virtual. Container akan dikemas secara efisien dengan deployment environment dan konfigurasi cluster. Kubernetes akan memastikan seluruh container dapat berjalan dan melakukan penggantian ketika container mati, tidak merespon, atau dalam kondisi bermasalah [3].

II. DASAR TEORI

A. Cloud computing

1) Definisi

Cloud computing memiliki beberapa defenisi berbeda dan berasal dari beberapa ahli di bidang nya. Dari beberapa definisi tersebut, dapat disimpulkan bahwa cloud computing merupakan layanan komputasi teknologi informasi yang mencakup layanan hardware, software dan aplikasi yang dapat diperoleh melalui internet. Layanan tersebut harus dapat disesuaikan dengan kebutuhan pengguna dan biaya penggunaan layanan dikenakan sesuai jumlah sumber daya yang telah digunakan pengguna menggunakan basis per bulan atau per menit.

Cloud Computing adalah paradigma komputasi yang digunakan untuk efisiensi biaya dimana informasi dan daya komputer dapat diakses dari browser web oleh pelanggan. Cloud Computing adalah pengembangan berbasis internet dan penggunaan komputer teknologi. Cloud Computing adalah paradigma komputasi dengan keunggulan sumber daya yang mudah dikembangkan secara real-time seperti file, data, program, hardware, dan third party services dapat diakses dari browser Web melalui Internet untuk pengguna. pelanggan hanya membayar sumber daya komputer yang digunakan sesuai dengan perjanjian yang tercantum pada Services Level Agreement (SLA), dan tidak memiliki pengetahuan tentang bagaimana penyedia layanan teknologi infrastruktur untuk mendukung kebutuhan pengguna. Beban layanan di Cloud Computing berubah secara dinamis sesuai dengan kebutuhan pengguna [4].

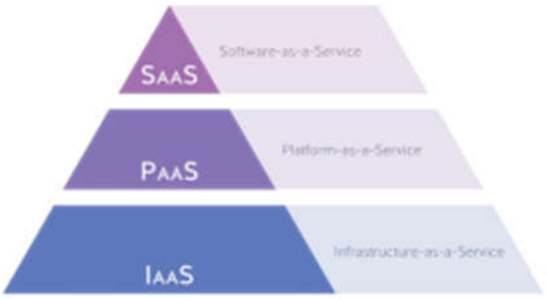
Cloud Computing menyediakan proses komputasi dari local, perangkat individual pengguna sampai penggunaan terdistribusi, virtual, dan pengembangan sumber daya, sehingga memungkinkan pengguna untuk memanfaatkan perhitungan, penyimpanan, dan sumber daya aplikasi lainnya, yang membentuk Cloud secara on-demand [5].

2) Layanan Cloud Computing

Cloud computing secara umum memiliki tiga lapisan, seperti pada gambar [6] :

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 03, Nomor 02, Desember 2018 : 84 - 93



Gambar 1. Tiga lapisan cloud computing

3) Software-as-a-Service (SaaS)

SaaS merupakan abstraksi tingkat tertinggi pada Cloud dan aplikasi disampaikan melalui World Wide Web (www) sebagai sebuah layanan. SaaS menawarkan aplikasi dalam cakupan yang luas, seperti aplikasi untuk menunjang produktivitas (e.g. tipe-office) sampai aplikasi yang menunjang enterprise atau perusahaan seperti e-mail hosting, supply chain management atau enterprise resource planning.

4) Platform-as-a-Service (PaaS)

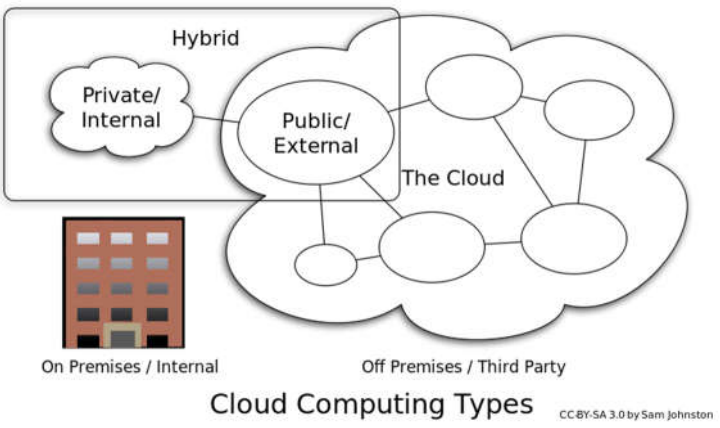
PaaS merupakan tingkatan selanjutnya dari abstraksi pada cloud, yang mana tidak hanya melakukan abstraksi teknis tetapi juga layanan aplikasi infrastruktur esensial seperti: komputasi, konektivitas, kontrol akses, dan lain sebagainya. Pada model komputasi konvensional, sekelompok jaringan, database, dan ahli sistem manajemen dibutuhkan untuk memastikan bahwa segalanya berjalan sesuai dengan yang diinginkan. Namun dengan cloud computing, hal-hal tersebut tidak lagi diperluka karena telah disediakan oleh penyedia layanan cloud computing.

5) Infrastructure-as-a-Service (IaaS)

IaaS merupakan lapisan yang paling bawah dari cloud computing. Iaas menyediakan abstrak dari sumber daya infrastruktur TI seperti penyimpanan dan memori sebagai layanan. Penyedia layanan cloud mengelola infrastruktur fisik, provisi infrastruktur dari sistem operasi.

6) Model Penyebaran Cloud computing

Menurut NIST, ada 4 (empat) model penyebaran cloud computing, yaitu:



Gambar 2. Model penyebaran cloud computing [7]

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 03, Nomor 02, Desember 2018 : 84 - 93

7) Private cloud

Private cloud merupakan layanan komputasi yang disediakan untuk memenuhi kebutuhan internal dari perusahaan. Layanan ini dapat dilakukan baik oleh penyedia layanan cloud computing atau dilakukan sendiri oleh perusahaan.

8) Community cloud

Community cloud adalah layanan cloud computing yang dibangun khusus untuk komunitas tertentu, yang penggunanya berasal dari organisasi yang mempunyai kebutuhan umum dan kebutuhan khusus. Community Cloud ini bisa dimiliki, dipelihara, dan dioperasikan oleh satu atau lebih organisasi dari komunitas tersebut, pihak ketiga, ataupun kombinasi dari keduanya.

9) Public cloud

Public cloud merupakan model penyebaran yang paling sering dianggap sebagai cloud, yang mana didalamnya terdapat banyak pengguna yang mungkin sama sekali tidak memiliki kesamaan apapun.

10) Hybrid cloud

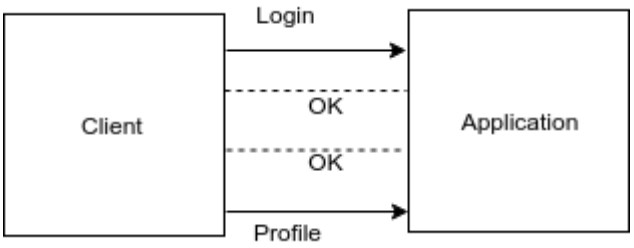
Adalah gabungan dari layanan Public cloud dan Private cloud yang diimplementasikan oleh suatu perusahaan. Dalam Hybrid cloud ini, perusahaan dapat memilih proses bisnis mana yang ingin dipindahkan ke Public cloud dan proses bisnis mana yang harus tetap berjalan di Private cloud.

B. Docker

Docker merupakan open platform yang dapat digunakan agar developer dan system administrator dalam bekerjasama untuk melakukan proses build, package dan deploy aplikasi, dengan tidak bergantung pada arsitektur mesin tujuan, bahasa pemrograman, dan sistem operasi. Melakukan proses deploy pada layanan cloud adalah kemudahan yang diberikan oleh docker, karena aplikasi dapat dipaketkan didalam container sehingga dapat di deploy dimana saja. Setiap aplikasi akan terisolasi dan terjaga keamanannya didalam container, dengan menjalan banyak container secara bersamaan pada host yang sama tanpa mengganggu hypervisor.

III. PERANCANGAN

Untuk memberikan gambaran tentang model penelitian eksperiment ini, peneliti akan memberikan gambar proses workload pada layanan yang akan dibuat dan diuji dalam penelitian ini.

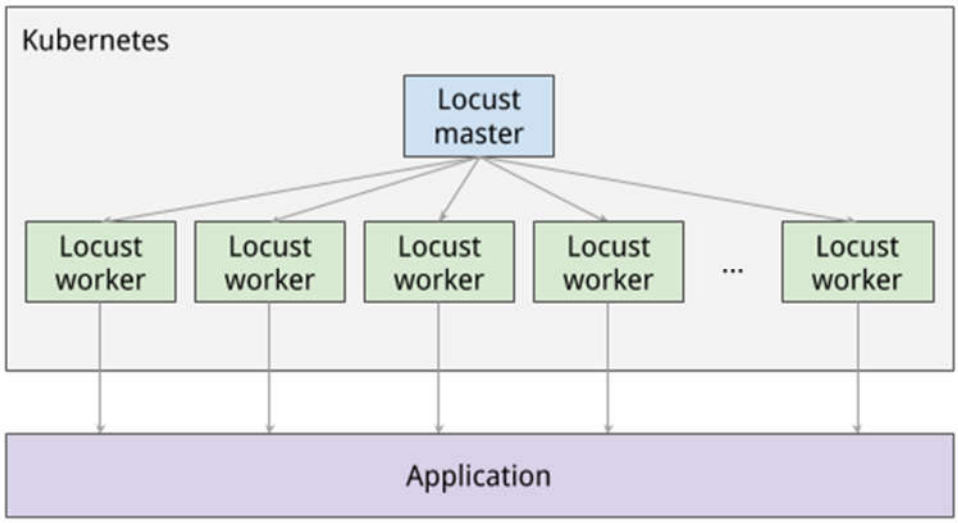


Gambar 3. workload proses client dan application

Dalam proses diatas workload merupakan proses komunikasi antara klien dengan aplikasi / layanan. Dalam hal ini, klien akan melakukan akses ke halaman utama / dan halaman /profile. Aplikasi / layanan yang dituju oleh klien akan menjadi endpoint bagi locust master dan worker yang akan digunakan untuk menguji performa layanan tersebut.

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 03, Nomor 02, Desember 2018 : 84 - 93



Gambar 4. Distribusi sistem untuk load testing aplikasi[1]

Untuk implementasi distribusi sistem tersebut, penelitian ini menggunakan cluster docker yang dimanajemen oleh kubernetes dengan menggunakan platform google cloud, layanan ini dikenal dengan Google Kubernetes Engine. Untuk cluster locust worker menggunakan 4 node dengan tipe n1-highcpu-4 (High-CPU machine type dengan 4 virtual CPUs dan memory 3.60 GB).

1) Persiapan locust dan Google Cloud Platform

Locust yang digunakan untuk performasi testing terdiri dari 3 fungsi yaitu master, worker dan services definition. Pada tahap awal diperlukan konfigurasi untuk 3 fungsi yaitu locust-master-controller.yaml, locust-master-service.yaml, dan locust-worker-controller.yaml. Seluruh konfigurasi akan disimpan dalam direktori kubernetes-config.

Untuk memudahkan proses eksperimen ini, peneliti harus dapat mengakses layanan google cloud platform melalui console dengan menggunakan cloud SDK. Proses instalasi dari cloud SDK adalah sebagai berikut :

```
# Create an environment variable for the correct distribution
export CLOUD_SDK_REPO="cloud-sdk-$(lsb_release -c -s)"
# Add the Cloud SDK distribution URI as a package source
echo "deb https://packages.cloud.google.com/apt $CLOUD_SDK_REPO main" | sudo
tee -a /etc/apt/sources.list.d/google-cloud-sdk.list
# Import the Google Cloud Platform public key
curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
# Update the package list and install the Cloud SDK
```

Proses inisialisasi dari cloud SDK menggunakan perintah **gcloud init** dan untuk proses login menggunakan **gcloud auth**.

Untuk memastikan bahwa seluruh project berarti dalam project ID yang sama di cloud

```
$ export PROJECTID=riset-akakom
```

[1] https://cloud.google.com

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 03, Nomor 02, Desember 2018 : 84 - 93

Karena dalam riset ini menggunakan locust dalam bentuk container, sehingga image container ini perlu untuk di upload ke dalam google cloud storage

```
$ docker tag locust-tasks gcr.io/$PROJECTID/locust-tasks
```

```
$ sudo gcloud docker -- push gcr.io/$PROJECTID/locust-tasks
```

Setelah docker image di upload pada registry google cloud, perlu dilakukan update untuk letak lokasi image container. Perubahan ini dapat langsung diganti pada file YAML dari konfigurasi locust master dan worker.

2) Deploy kubernetes

Untuk membuat cluster container di dalam google cloud platform dapat menggunakan fasilitas Google Container Engine dengan menggunakan perintah gcloud. Pada penelitian ini menggunakan tipe mesin node n1-standard-1. Namun karena dalam penelitian ini akan menggunakan mesin dengan CPU yang besar untuk melihat performa dari clusters, sehingga digunakan tipe mesin n1-highcpu-4. Spesifikasi dapat ditingkatkan dengan mudah, jika terjadi kegagalan layanan dalam memberikan service.

Karena dalam penelitian ini menggunakan 3 mesin worker dan 1 controller, maka dibutuhkan 4 node dalam clusters. Nama clusters yang digunakan adalah locust-cluster

```
$ gcloud container clusters create locust-cluster –machine-type=n1-highcpu-4 –num-nodes=8
```

```
$ gcloud config set container/cluster locust-cluster
```

```
$ gcloud container clusters get-credentials locust-cluster
```

3) Deploy locust

Perintah kubectl digunakan untuk deploy locust master, services, dan worker. Dilanjutkan dengan konfigurasi firewall agar mesin cluster dapat diakses melalui public.

```
$ kubectl create -f  locust-master-controller.yaml
```

```
$ kubectl create -f  locust-master-service.yaml
```

```
$ kubectl create -f  locust-worker-controller.yaml
```

```
$ gcloud compute firewall-rules create riset-akakom-locust –allow=tcp:8089 –target-tags gke-locust-cluster-[…]-node
```

Untuk melihat services berjalan, external ip dan beberapa informasi lain

```
$ kubectl get services
```

```
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)
AGE
```

```
kubernetes      ClusterIP      10.43.240.1     <none>          443/TCP
23m
```

```
locust-master   LoadBalancer   10.43.248.214   35.193.94.236
8089:30332/TCP,5557:32684/TCP,5558:30419/TCP,5559:32639/TCP,5560:30676/TCP,5561:310
60/TCP,5562:30393/TCP,5563:31704/TCP   3m
```

4) locust access

Setelah seluruh proses selesai, locust dapat diakses melalui eksternal IP dari layanan google container engine. Pada menu ini, dapat dilakukan pengaturan akses user secara simultan pada sebuah layanan. Model terdistribusi dari locust dapat memungkinkan untuk mengakses seluruh layanan.

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 03, Nomor 02, Desember 2018 : 84 - 93

Type	Name	# requests	# fails	Median	Average	Min	Max	Content Size	# reqs/sec
GET	/	4065	0	5	6	3	80	22	75.8
POST	/login	0	0	0	0	0	0	0	0
GET	/profile	1975	0	5	6	3	122	8	36.2
	Total	6040	0	5	6	0	122	17	112

Gambar 5. Akses ke locust dan hasil simulasi akses users.

IV. PEMBAHASAN

Hasil dari ujicoba menghasilkan 2 data yang berupa request dan distribution. Pertama, pada request, data yang diperoleh adalah data mengenai request pada protocol http dengan method GET. Method GET pada halaman / dan /profile dan method POST pada halaman /login. Dalam tabel request diperoleh data mengenai jumlah request, failures request, response time, content size dan request per detik.

TABEL I
DATA REQUEST LOCUST TESTING LOAD / PERFORMA

Method Name	GET /	POST /login	GET /profile	None Total
Request	425	0	0	425
Failures	0	0	187	187
Median response time	1500	0	0	1500
Average response time	1441	0	0	1441
Min reponse time	255	0	0	0
Max response time	1538	0	0	1538
Average content size	38	0	0	38
Request / s	3.32	0	0	3.32

Data kedua adalah tabel distribution yang memberikan informasi mengenai kemampuan sebuah layanan dalam menerima request dalam persentase. Akses ke layanan juga tetap menggunakan metode HTTP GET pada halaman / dan /profile. Method POST pada /login.

TABEL II
DISTRIBUSI REQUEST LOCUST TESTING LOAD / PERFORMA

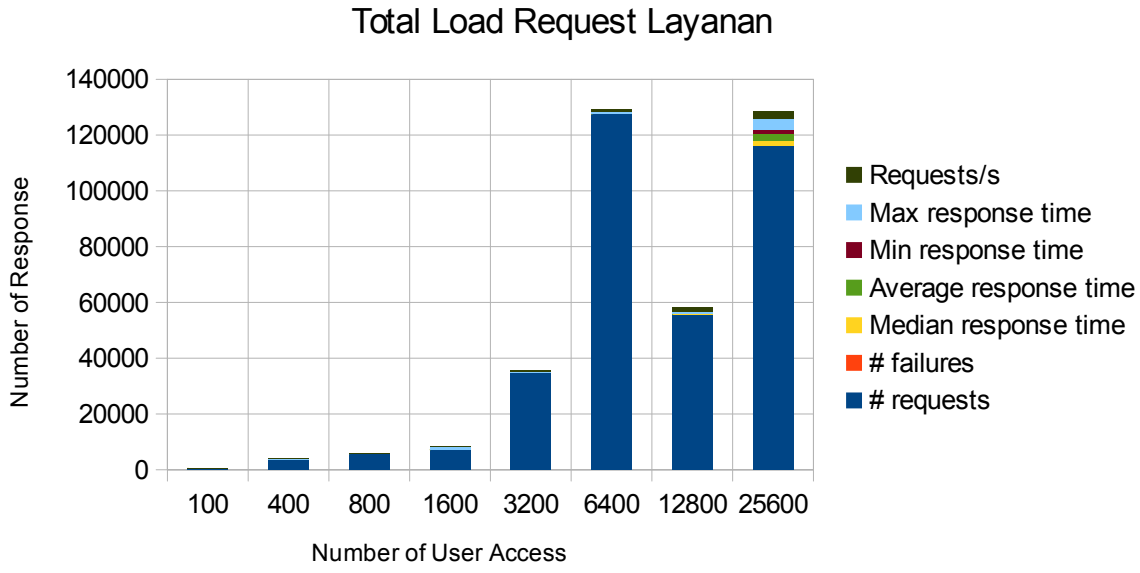
Name	GET /	POST /login	/profile	None Total

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 03, Nomor 02, Desember 2018 : 84 - 93

Request	2620	6	0	2626
50%	260	1500	N/A	260
66%	1500	1500	N/A	1500
75%	1500	1500	N/A	1500
80%	1500	1500	N/A	1500
90%	1500	1500	N/A	1500
95%	1500	1500	N/A	1500
98%	1500	1500	N/A	1500
99%	1500	1500	N/A	1500
100%	1547	1502	N/A	1547

Secara keseluruhan request yang diujicoba dalam bentuk simulasi ke layanan terdiri dari 8 model jumlah user yang berbeda. Dimulai dari jumlah user akses sebanyak 100 user, 400 user, 800 user, 1600 user, 3200 user, 6400 user, 12800 user, dan 25600 user. Jumlah user tersebut kemudian diujicoba dengan request akses 10 detik per user, untuk mensimulasikan beban akses pada layanan yang sebenarnya. Pada Gambar 5.3. terlihat bahwa simulasi dengan akses user 25600 memiliki request paling tinggi dibandingkan dengan jumlah akses user yang lain, jumlah tersebut disusul oleh jumlah akses user sebanyak 6400 user dan kemudian 12800 user. Nilai response per detik terbesar juga dapat dilihat pada jumlah akses user sebanyak 25600. Walaupun demikian, hampir tidak ada jumlah failures dalam grafik dibawah, hal ini membuktikan bahwa sekalipun jumlah akses user dan request mencapai 120000/s , layanan tetap dapat menyediakan layanan dengan availability yang tinggi.



Gambar 6. Total Load Request Performa Kubernetes

Sementara itu, pada response time distribution locust yang ditampilkan dalam rentang persentase menunjukkan persentase dari request diselesaikan dalam waktu tertentu (ms). Dengan data yang terdapat pada grafik di tabel 5.3 dapat disimpulkan bahwa range user akses sebanyak 100 – 12800 user 50% request dapat diselesaikan dalam waktu 5 ms, dan untuk menyelesaikan sebanyak 100% requests dibutuhkan waktu yang cukup beragam dimulai dari 57 ms sampai 1 s. Sementara

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 03, Nomor 02, Desember 2018 : 84 - 93

itu, untuk menyelesaikan request sebanyak 149639 yang disimulasikan dengan user akses sebanyak 25600 user diperlukan waktu antara 2 s – 4 s untuk menyelesaikan request mulai dari 50% - 100%.

TABEL III
TOTAL DISTRIBUSI LOAD PERFORMA KUBERNETES

Jumlah User access	**100**	**400**	**800**	**1600**	**3200**	**6400**	**12800**	**25600**
Jumlah Request	658	5860	6784	10636	40382	141811	145611	149638
50%	5	5	5	5	5	5	5	2100
66%	5	5	5	5	5	6	6	2300
75%	5	5	5	5	6	6	7	2400
80%	5	5	5	6	6	7	9	2500
90%	6	6	6	6	7	8	10	2700
95%	7	7	7	8	9	11	15	3000
98%	19	25	25	33	26	37	50	3300
99%	31	38	40	47	42	54	60	3400
100%	57	68	84	824	232	816	1008	3844

V. KESIMPULAN

Hasil penelitian menunjukkan bahwa terjadi peningkatan availability layanan dengan rancangan kubernetes tidak ditemukan persentase failure request, dari test simulasi pengiriman request dengan jumlah beban akses 100 user, 400 user, 800 user, 1600 user, 3200 user, 6400 user, 12800 user, dan 25600 user. Dan rata-rata diselesaikan 100% selama range waktu 1 – 4 detik. Dari data-data tersebut, dapat disimpulkan bahwa cluster kubernetes dengan optimalisasi dapat mengurangi persentase request failure dan meningkatkan availability layanan.

Penyedia layanan dapat melakukan analisis performa berdasarkan resiko request yang dapat mereka tangani, sebagai contoh, jika penyedia layanan ingin dapat memberikan layanan dengan jumlah maksimal user 25600 dengan waktu dibawah 5 s, maka target tersebut telah tercapai. Namun jika penyedia layanan perlu memberikan layanan ketercapaian adalah 1 s, sekalipun jumlah akses user maksimal, maka diperlukan beberapa optimalisasi dari sisi layanan.

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 03, Nomor 02, Desember 2018 : 84 - 93

DAFTAR PUSTAKA

[1] A. Ashari and M. Riasetiawan, "High Performance Computing on Cluster and Multicore Architecture," *TELKOMNIKA Telecommun. Comput. Electron. Control*, vol. 13, no. 4, pp. 1408–1413, 2015.
[2] J. Nickoloff, *Docker in Action*. Manning Publications Co., 2016.
[3] G. Sayfan, "Mastering Kubernetes: automating container deployment and management," 2017.
[4] K. Xiong and H. Perros, "Service performance and analysis in cloud computing," in *Services-I, 2009 World Conference on*, 2009, pp. 693–700.
[5] J. Osis, *Model-driven domain analysis and software development: Architectures and functions: Architectures and functions*. IGI Global, 2010.
[6] S. Dhar, "From outsourcing to Cloud computing: evolution of IT services," *Manag. Res. Rev.*, vol. 35, no. 8, pp. 664–675, 2012.
[7] B. Furht and A. Escalante, *Handbook of cloud computing*, vol. 3. Springer, 2010.