IMPLEMENTASI KUBERNETES CLUSTER MENGGUNAKAN VAGRANT

M Ramadhan Muttakin[1)], Muhammad Arif Fadhly Ridha[2)]

[1]Teknik Informatika, Politeknik Caltex Riau, Jl. Umban Sari, Pekanbaru 28265

[2]Teknik Informatika, Politeknik Caltex Riau, Jl. Umban Sari, Pekanbaru 28265

E-mail : muttakin@alumni.pcr.ac.id, fadhly@pcr.ac.id

Abstrak

The development of technology that can package an application and its needs in a virtualization called a container, therefore we need a software that can manage containers, one of which is docker. However, the need is always increasing, the system is not able to accommodate many applications, therefore we need a software capable of managing clustering containers on a large scale. One such software is Kubernetes. Kubernetes can run in virtualization, virtualization that can run Kubernetes in it is vagrant and KVM. In this study, a comparison is made between the two types of virtualization, the comparison is to see the success rate of high-availability, scalability and performance testing. The results of the research that was carried out during the implementation of kubernetes cluster on vagrant virtualization and KVM virtualization, KVM virtualization was superior to vagrant virtualization, it could be seen from the difference in latency and performance test which was quite large between the two virtualizations.

Keywords: *Cluster, Virtualisasi, Kuberetes, Vagrant, KVM.*

Abstrak

Perkembangan teknologi yang dapat mengemas sebuah aplikasi dan kebutuhannya dalam sebuah virtualisasi yang disebut *container*, oleh sebab itu dibutuhkan sebuah software yang dapat melakukan management terhadap *container*, salah satunya adalah docker. Namun kebutuhan selalu meningkat sistem tidak mampu untuk menampung banyak aplikasi, oleh karna itu dibutuhkan sebuah *software* yang mampu melakukan management *clustering container* dalam skala besar. Salah satu *software* tersebut adalah kubernetes. Kubernetes dapat berjalan didalam virtualisasi, virtualisasi yang dapat menjalankan kubernetes didalamnya adalah vagrant dan KVM. Pada penelitian ini dilakukan perbandingan antara 2 jenis virtualisasi tersebut, perbandingan yang dilakukan yaitu melihat tingkat keberhasilan dari *high-avaliability*, *scalability* dan *performance testing*. Hasil dari penelitian yang telah dilakukan pada saat diterapkannya kubernetes *cluster* pada virtualisasi vagrant dan virtualisasi KVM, virtualisasi KVM lebih unggul dari virtualisasi vagrant dapat dilihat dari perbedaan *latency* dan *performance test* yang cukup besar antara kedua virtualisasi tersebut.

Kata Kunci: *Cluster, Virtualisasi, Kubernetes, Vagrant, KVM.*

ISSN: 2339 – 2053

Pekanbaru, 25 Agustus 2021

PENDAHULUAN

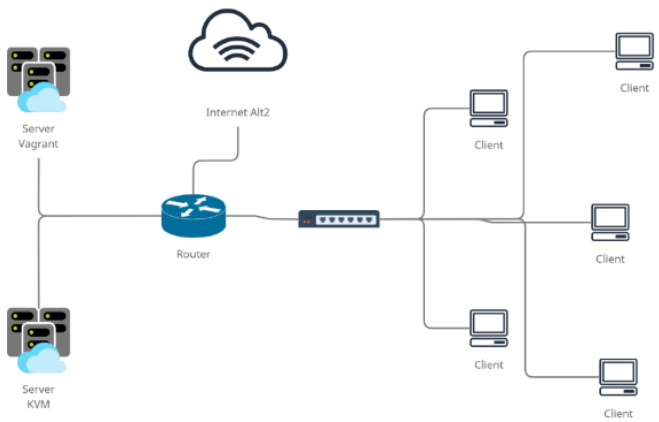
Pesatnya perkembangan teknologi yang mampu memudahkan seluruh bidang pekerjaan, teknologi terus berkembang karena kebutuhan dan sumber daya yang semakin meningkat. Permasalahan yang dihadapi adalah beban *traffic* yang besar sering kali membuat server *down* dan *overload*. Dengan adanya *clustering computing* dapat membuka peluang untuk dapat hadir dimanapun, memberikan kenyamanan, akses jaringan sesuai permintaan (*on-deman*) ke lokasi sumber daya komputasi terkonfigurasi (misalnya, jaringan, server, penyimpanan, aplikasi dan layanan), yang dapat dengan cepat dijalankan dan diluncurkan, dengan upaya pengelolaan minimal atau dengan menggunakan penyedia jasa layanan (Mell & Grance, 2011). *Clustering computing* dapat dikombinasikan dengan *container, container* adalah virtualisasi pada level sistem operasi dimana tiap proses atau aplikasi yang dijalankan tiap container memiliki kernel yang sama. dimana virtual machine membutuhkan kernel sistem operasi yang berbeda-beda tiap aplikasi yang dijalankan (Dwiyatno, Rakhmat, & Gustiawan, 2020). Oleh karna itu dibutuhkan sebuah aplikasi yang mampu melakukan *management* terhadap *container*, salah satu aplikasi docker. Docker adalah sebuah project open-source yang menyediakan platform terbuka untuk developer maupun sysadmin untuk dapat membangun, mengemas, dan menjalankan aplikasi dimanapun di dalam sebuah *container* (Dwiyatno, Rakhmat, & Gustiawan, 2020). Namun karena produktivitas semakin meningkat Maka dibutuhkan sebuah *software* yang dapat melakukan *clustering management container* agar aplikasi selalu tersedia dan dapat menerima banyak *traffic*. Salah satu *platform* yang dapat melakukan *clustering management container* adalah Kubernates. Kubernetes adalah sistem open source untuk mengotomatisasi penyebaran, penskalaan, dan pengelolaan *container*. Kubernetes sendiri berfungsi sebagai pengelola *container-container* serta menyediakan Platform untuk mendukung fungsinya. Design Kubernetes terdiri dari *Pods*, *Labels and Selectors*, *Controllers*, *Services* (Kubernetes, 2018). Kubernetes dapat berjalan didalam *virtual machine*, salah satu *virtual machine* yang dapat menjalankan kubernetes didalamnya adalah vagrant. Vagrant adalah sebuah *opensource* untuk membangun dan mengelola lingkungan virtualisasi dalam satu alur kerja yang berfokus pada otomatisasi (Khalifi, 2017). Vagrant adalah alat untuk otomatisasi dengan bahasa khusus yaitu DSL. DSL adalah sebuah *script* Yang

ISSN: 2339 – 2053

digunakan untuk melakukan otomatisasi pembuatan *virtual machine* dan lingkungan *virtual machine*. Pengguna dapat membuat sekumpulan intruksi dengan menggunakan vagrant DSL untuk melakukan konfigurasi pada *virtual machine* tersebut (Vagrant, 2020). Kemudian kubernetes *cluster* juga dapat berjalan pada kernel based virtual machine (KVM), KVM adalah teknologi virtualisasi yang dikembangkan oleh linux dengan perangkat keras type x86 (64-bit). KVM diimplementasikan sebagai modul kernel *loadable* yang mengubah kernel Linux menjadi *bare metal hypervisor* (IDreg, 2013), Salah satu keunggulan utama KVM adalah terintegrasi nya modul KVM dengan kernel Linux sehingga KVM dapat langsung dipergunakan pada native kernel tanpa harus melakukan patch atau melakukan instalasi kernel terpisah (Kartikasari, 2012). Dari penelitian ini didapatkan hasil bahwa kubernetes *cluster* dapat diterapkan dengan baik pada virtualisasi KVM dan vagrant. Kemudian dilakukan pengujian *high-avaliability*, *scalability* dan *performance testing* sebanyak 15 kali, data pengujian dianalisis untuk melihat perbedaan antara 2 jenis teknologi virtualisasi tersebut saat diterapkan kubernetes *cluster* didalamnya.

METODE PENELITIAN

Adapun dalam melakukan implementasi pada penelitian ini, menggunakan 2 buah server yaitu server dengan menggunakan virtualisasi vagrant dan virtualisasi KVM. Ada pun tompologi yang digunakan seperti Gambar 1.



Gambar 1. Tompologi Jaringan

ISSN: 2339 – 2053

Pada Gambar 1 server akan menggunakan sistem operasi CentOS 7, kemudian akan dilakukan *instalasi* kebutuhan *package* dari kernel-based virtual machine (KVM) dan virtualisasi vagrant pada server yang berbeda. Setiap server akan memiliki 3 buah *nodes* KVM dan vagrant.

Adapun metode pengujian yang dilakukan pada penelitian ini yaitu :

1. Pengujian *High-Avaliability*

Pengujian ini dilakukan sebanyak 15 kali dengan meng-*offline*-kan salah satu *node* pada server pengujian ini bertujuan untuk mengetahui keberhasilan *failover*, *failback* dan *loadbalancer*. Adapun pengujian yang dilakukan sebagai berikut :

- Melihat tingkat keberhasilan *Failover* dan *Load Balancing.*
- Melihat tingkat keberhasilan *Failback.*

2. Pengujian *Scalability*

Pengujian *scalability* adalah kemampuan melakukan penambahan saat beban layanan meningkat dan melakukan pengurangan layanan pada saat penurunan beban. Untuk mengetahui tingkat keberhasilan dari *scalability* sebagai berikut :

- Dapat menambah pod pada saat beban layanan meningkat, dan mematikan pod pada saat tidak digunakan.
- Dapat menambah node saat layanan meningkat dan mengurangi node pada saat node tidak digunakan.

3. Pengujian *Performance testing*

Pengujian *performance* dilakukan dengan menggunakan aplikasi *strees tools webserver* yaitu JMeter. pembebanan dilakukan untuk mengetahui besar penggunaan CPU dan *memory* pada kedua server tersebut. Untuk monitoring tersebut menggunakan sebuah *software* SNMP untuk menganalisis perbedaan penggunaaan CPU dan *Memory* pada kedua server tersebut. Adapun pengujian yang dilakukan seperti berikut :

- Pengujian menggunakan aplikasi JMeter untuk dilakukan pembebanan dan *request* pada layanan webserver.
- Pengujian perbandingan persentase penggunaan CPU dan Memory pada server KVM dan server vagrant menggunakan aplikasi SNMP.

ISSN: 2339 – 2053

HASIL DAN PEMBAHASAN

1. Hasil dan Analisis pengujian *High-Avaliability*.

Setelah melakukan pengujian *failover, failback* dan *loadbalancer* yang dilakukan sebanyak 15 kali dengan skenario melakukan *offline* pada salah satu *nodes* masing-masing server. Berikut tabel hasil dari pengujian tingkat keberhasilan dari *failback* dan *failover* :

Tabel 1. Tingkat Keberhasilan *failback, failover*

Pengujian Ke -	Vagrant		KVM	
	Failback	Failover	Failback	Failover
1	✓	✓	✓	✓
2	✓	✓	✓	✓
3	✓	✓	✓	✓
4	✓	✓	✓	✓
5	✓	✓	✓	✓
6	✓	✓	✓	✓
8	✓	✓	✓	✓
9	✓	✓	✓	✓
10	✓	✓	✓	✓
11	✓	✓	✓	✓
12	✓	✓	✓	✓
13	✓	✓	✓	✓
14	✓	✓	✓	✓
15	✓	✓	✓	✓

Pada tabel 1. tingkat keberhasilan *failback* dan *failover* pada 2 server berhasil dilakukan sebanyak 15 kali. *Loadbalancer* yang diterapkan pada kubernetes *cluster* yaitu distribusi *resource/pods* kepada setiap *nodes.*

2. Hasil dan Analisis pengujian *Scalability*

1) Melakukan penambahan pods & Pengurangan pods

ISSN: 2339 – 2053

Teknik yang dilakukan untuk melakukan penambahan dan pengurangan *pods* yaitu dengan menggunakan *horizontal pods autoscaller* (HPA) yang merupakan salah satu teknik otomatisasi dengan menyesuaikan beban dari *traffic*. Sehingga ketiga beban *traffic* meningkat, *pods* akan otomatis melakukan pemanbahan. Dan pada saat *traffic* menurut *pods* akan menyesuaikan layanan sesuai dengan user yang mengakses.

```
NAME            REFERENCE                         TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
wordpress-hpa   Deployment/wordpress-deploy       192%/50%   1         10        4          52m
```

Gambar 2. *Horizontal Pods Autoscaller*

2) Melakukan penambahan dan pengurangan nodes

Pengurangan *nodes* dilakukan sebanyak 1 kali dengan skenario melakukan *offline* pada salah satu *nodes*. Kemudian untuk penambahan *nodes, nodes* yang sebelumnya *offline* di-*online*-kan kembali.

Melakukan *offline nodes* :

```
[root@madanvagrant vagrant]# vagrant halt node2
==> node2: Attempting graceful shutdown of VM...
```

Gambar 3. *Offline node 2*

Pada Gambar 3 adalah printah untuk melakukan *offline* terhadap *nodes*.

Status *nodes* :

```
[root@madanvagrant vagrant]# vagrant status
Current machine states:

master                    running (virtualbox)
node1                     running (virtualbox)
node2                     poweroff (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.
```

Gambar 4. Status *node2* setelah *offline*

Pada Gambar 4 adalah perintah untuk melihat status *nodes* pada lingkungan virtualisasi. Dimana *node 2* telah dilakukan *offline*.

ISSN: 2339 – 2053

Pekanbaru, 25 Agustus 2021

Melakukan *online nodes* :

```
[root@madanvagrant vagrant]# vagrant up node2
Bringing machine 'node2' up with 'virtualbox' provider...
```

Gambar 5. *Online node 2*

Pada Gambar 5 adalah perintah untuk melakukkan *online* terhadap *nodes*.

Status *nodes* :

```
[root@madanvagrant vagrant]# vagrant status
Current machine states:

master                    running (virtualbox)
node1                     running (virtualbox)
node2                     running (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.
```

Gambar 6. Status *node 2* dilakukan *online*

Pada Gambar 6 adalah perintah untuk melihat status pada lingkungan virtualisasi, dimana *node 2* yang sebelumnya *offline* di-*online*-kan kembali.

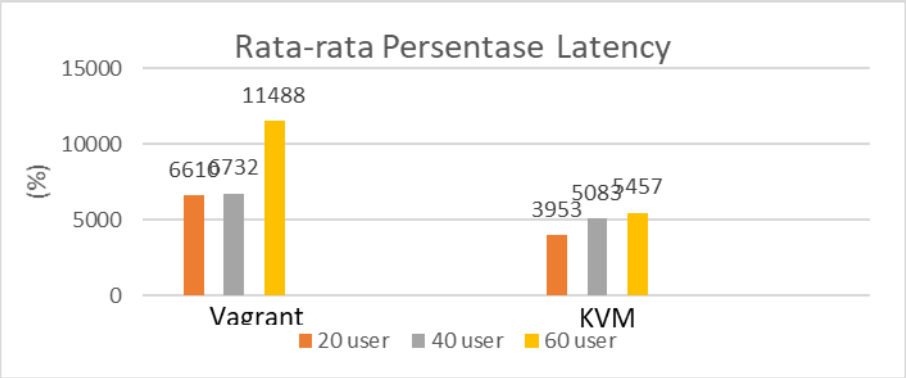
3. *Performance Testing*

1) Pengujian *stress request* Menggunakan aplikasi Jmeter

Pengujian *performance testing* dilakukan sebanyak 15 kali pada kedua server. Skenario yang digunakan pada satu kali pengujian terdiri dari 20, 40 dan 60 user. Lama waktu yang dilakukan pada setiap user yaitu 5 menit.

Adapun rata-rata *latency* pada server vagrant dan server KVM dapat dilihat pada Gambar 3 :

ISSN: 2339 – 2053



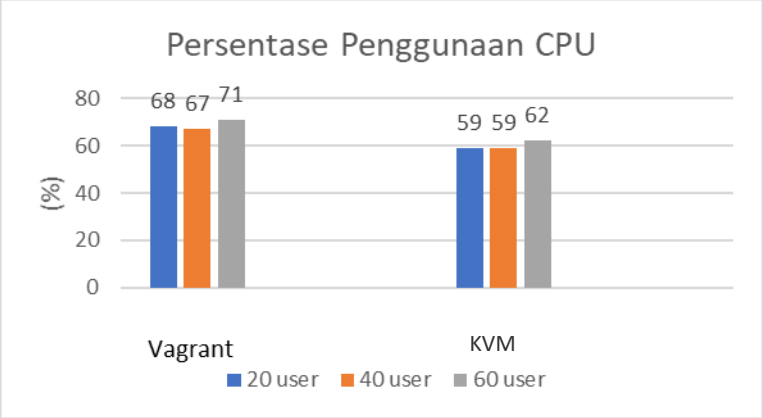
Gambar 3. Rata-Rata Latency

Pada Gambar 3. adalah data *view result table* yang didapat dari *software* Jmeter pada sub-menu *latency*, ada pun perbedaan penggunaan CPU setelah dilakukan *stress request* sebanyak 20 user yaitu 2,66 detik, 40 user yaitu sebesar 1,67 dan 60 user yaitu sebesar 6,03 detik.

2) Pengujian *performance* CPU dan *Memory*

Data pengujian *performance* CPU dan Memory diambil pada saat dilakukan *stress request* pada kedua server, sehingga pada saat dilakukan *stress request* dapat dilihat persentase penggunaan CPU dan Memory dengan skenario yang digunakan pada pengujian *stress request*.

Adapun rata-rata penggunaan CPU saat dilakukan *stress request* pada server vagrant dan server KVM dapat dilihat pada Gambar 4 :



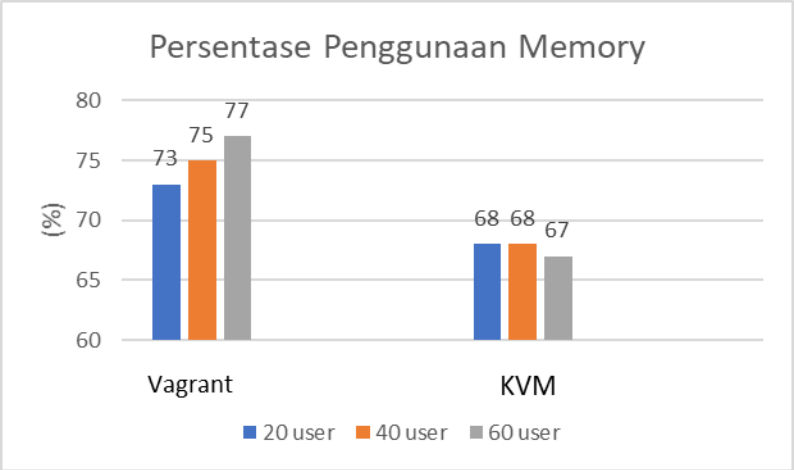
Gambar 4. Persentase Penggunaan CPU

ISSN: 2339 – 2053

Pekanbaru, 25 Agustus 2021

Pada Gambar 4. Adalah hasil rata-rata persentase penggunaan CPU saat dilakukan *stress request* pada kedua server. Adapun persentase penggunaan CPU saat dilakukan *stress request* pada server vagrant yaitu 20 user sebesar 68%, 40 user sebesar 67% dan 60 user yaitu 71%. Sementara pada server KVM yaitu 20 user sebesar 59%, 40 user sebesasr 60% dan 60 user sebesar 62%.

Adapun rata-rata *latency* pada server vagrant dan server KVM dapat dilihat pada Gambar 3 :



Gambar 5. Persentase Penggunaan Memory

Pada Gambar 5. Adalah hasil rata-rata persentase penggunaan memory sebelum dilakukan *stress request* pada kedua server. Adapun persentase penggunaan memory pada saat dilakukan *stress request* pada server vagrant yaitu 20 user sebesar 73%, 40 user sebesar 75% dan 60 user yaitu sebesar 77%. Sementara pada server KVM yaitu 20 user sebesar 68%, 40 user sebesasr 68% dan 60 user sebesar 67%.

KESIMPULAN

Dari hasil Implementasi kubernetes *cluster* menggunakan vagrant dan KVM, maka dapat disimpulkan sebagai berikut:

1) Kubernetes *cluster* dapat dibangun didalam virtual mesin vagrant dan KVM.
2) *Horizontal pods autoscaller* (HPA) dapat melakukan replikasi *pods* secara otomatis pada teknologi kubernetes.

ISSN: 2339 – 2053

3) *Failback* dan *failover* dapat diimplementasikan setelah dilakukan pengujian sebanyak 15 kali pada virtualisai vagrant dan KVM. *Loadbalancing* melakukan distribusi *resource* berupa *pods* kepada *nodes*.
4) *Scalability* dengan menggunakan HPA dapat mempermudah pembesaran pada saat *traffic* meningkat dan menyesuaikan layanan sesuai user yang mengakses. HPA sangat berperan untuk menyesuaikan *resource* dengan sumber daya yang dimiliki.
5) Setelah melakukan pengujian *performance testing* implementasi kubernetes *cluster* lebih baik diterapkan pada KVM, sebab persentase penggunaan CPU dan Memory yang lebih kecil dari pada saat diimplementasikannya kubernetes *cluster* pada virtualisasi vagrant.

DAFTAR PUSTAKA

Dwiyatno, S., Rakhmat, E., & Gustiawan, O. (2020). Implementasi Virtualisasi Server Berbasis Docker Container. *Jurnal PROSISKO*.

IDreg. (2013). *Pengertian KVM, XEN, OPENVZ.* Retrieved from https://www.idreg.net/pengertian-kvm-xen-openvz/

Kartikasari, D. (2012). Analisa Perbandingan Metode KVM Dengan OpenVZ Pada Mesin VPS (Virtual Private Server) Di PT. Lintas Data Prima Yogyakarta. *Sekolah Tinggi Manajemen Informatika Dan Komputer, Yogyakarta* .

Khalifi, N. e. (2017). *Ceph Cluster Creation And Management On Vagrant Virtual Machine.* Barcelona: UPC.

Kubernetes. (2018). *Kubernetes*. Retrieved from Kubernetes.io: https://kubernetes.io/id/docs/concepts/

Mell, P., & Grance, T. (2011). *The NIST Definition of Cloud Computing.* Gaithersburg: Elsevier.

ISSN: 2339 – 2053