Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 2, No. 12, Desember 2018, hlm. 6849-6854 http://j-ptiik.ub.ac.id

Implementasi *Failover* Dan *Autoscaling* Kontainer *Web Server* Nginx Pada Docker Menggunakan Kubernetes

Yosua Tito Sumbogo[1], Mahendra Data[2], Reza Andria Siregar[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]yosuatito@gmail.com, [2] mahendra.data@ub.ac.id, [3]reza.jalin@ub.ac.id

Abstrak

Web server adalah bagian penting yang digunakan untuk menyediakan layanan *website*. *Web server* digunakan untuk melayani *request website* selama 24 jam 7 hari, ketika *web server* mati maka layanan *website* tidak akan dapat digunakan. Kondisi *web server* yang tidak dapat bekerja atau *failure* dapat disebabkan oleh beberapa hal seperti malfungsi perangkat keras, *crash* pada *web server*, putusnya sumber daya listrik, dan kegagalan jaringan. NGINX merupakan salah satu *software* penyedia *web server*. NGINX dapat di-*install* pada komputer *server* secara virtual *menggunakan* metode kontainer. hal ini memudahkan proses *deployment* karena kontainer bersifat *lightweight*, dan porTabel. Agar *web server* dapat melayani *request* maka diperlukan mekanisme *failover* dan *autoscaling*. Kubernetes merupakan perangkat lunak yang memiliki mekanisme *failover* dan *autoscaling* kontainer berbasis Docker. Didalam penelitian ini jumlah *hardware* yang digunakan adalah tiga *server* yang membentuk sebuah *cluster* menggunakan Kubernetes, satu *server* akan menjadi *master* dan dua *server* sebagai *minion*. Dari penelitian didapat rata-rata waktu yang dibutuhkan untuk *failover* adalah 264,74s untuk *node failure*, dan 3s untuk *service failure*. rata-rata waktu *autoscaling* sistem membutuhkan 45s untuk *scale up* dan 120s untuk *scale down*. *Autoscaling* Kubernetes mampu mengurangi CPU *usage* sebesar 0,4%. Sistem yang dibangun mampu menjalankan *web server* dengan metode *clustering*.

Kata kunci: *Kubernetes, kontainer, web server, NGINX, failover, autoscaling.*

Abstract

Web server is an important part used to provide website services. Web server is used to serve website requests for 24 hours 7 days if the web server is dead then the website service will not be used. The condition of web server that can not work or failure can be caused by some things like hardware malfunction, crash on web server, power outage, and network failure. NGINX is one of the web server software provider. NGINX is able to install on virtual server computers using container methods. this facilitates the deployment process because the containers are lightweight and portable. In order for the web server to serve requests it requires failover mechanism and autoscaling. Kubernetes is a software that has a Docker-based failover and autoscaling container mechanism. In this research the amount of hardware used is three servers that make up a cluster using Kubernetes, where one server will become master and two servers as minion. From the research obtained the average time required for failover is 264,74s for node failure, and 3s for service failure. the average autoscaling time of the system requires 45s for scale up and 120s for scale down. Autoscaling Kubernetes is able to reduce CPU usage by 0,4%. The built system is capable of running web server with clustering method.

Keywords: *Kubernetes, containers, web server, NGINX, failover, autoscaling.*

1. PENDAHULUAN

Website atau situs merupakan tempat untuk menyebarkan informasi seperti profil perusahaan, hasil penelitian, e-commerce, tulisan, Gambar, video dan masih banyak lagi secara online. Secara sederhana website terdiri dari beberapa halaman yang saling terhubung untuk menyalurkan informasi (Yuhefizar, Mooduto, & Hidayat, 2008). Untuk menjalankan sebuah website diperlukan server yang berjalan selama 24 jam untuk melayani permintaan dari pengguna. Web server merupakan server yang berfungsi untuk melayani request dan response pengguna menggunakan protokol HTTP (Hyper Text Transfer Protocol)

sebagai protokol pengiriman data (Gourley & Totty, 2002). NGINX merupakan salah satu web server yang dapat digunakan untuk menjalankan sebuah situs. NGINX bersifat open source dan memiliki banyak fitur seperti load balancer, reverse proxy, IMAP/POP3 proxy server. NGINX dapat menjalankan situs dengan mekanisme pertukaran data menggunakan protokol HTTP dan memiliki banyak dukungan pengguna (NGINX, 2018).

Masalah yang sering dialami ketika banyak pengungjung mengakses sebuah situs adalah server yang berperan sebagai web server tidak mampu menangani permintaan sehingga layanan tidak dapat diproses. Beberapa penyebab *server failure* adalah putusnya sumber daya listrik, malfungsi perangkat keras, sistem operasi yang crash dan kegagalan jaringan (Oracle, 2018). Salah satu cara untuk mengatasi failure adalah melakukan mekanisme failover yaitu sebuah proses pergantian layanan dalam hal ini web server dari server yang mengalami failure menuju server lain yang dalam keadaan standby, mekanisme failover berjalan secara otomatis (Jayaswal, 2005).

Permasalahan lain yang dapat terjadi adalah terkait kemampuan server dalam menangani permintaan dari pengguna seperti menerima request dan mengirim response HTTP. Proses penambahan layanan atau server tanpa mengganggu kinerja server disebut scalability atau skalabilitas (Bondi, 2000). Terdapat pula sebuah metode autoscaling dimana sistem akan memonitoring sebuah server cluster secara berkala dimana ketika sistem mendeteksi layanan pada sebuah server tidak dapat memenuhi permintaan pengguna maka sistem akan menambahkan layanan tersebut ke server lain yang tersedia (Armbrust, Fox, Griffith, Joseph, & Katz, 2009).

Untuk dapat mengatasi masalah failure dan skalabilitas dapat menggunakan sebuah metode server cluster yaitu membangun sistem yang terdiri lebih dari beberapa server yang bekerja sama untuk menjalankan sebuah layanan seperti web server (Oracle, 2018). Kubernetes merupakan tools yang dapat membuat sebuah sistem *server cluster* dimana beberapa server bekerja sama untuk menyediakan sebuah layanan, beberapa fitur seperti *failover* dan *autoscaling* dimiliki oleh Kubernetes sehingga mampu menangani permasalahan *failure* dan skalabilitas pada *web server*. Penelitian ini dilaksanakan untuk mengetahui bagaimana cara merancang dan mengimplmentasikan sebuah *server cluster* untuk menjalankan *web server* dengan mekanisme *autoscaling* dan *failover*.

2. KAJIAN PUSTAKA

2.1 Failover

Failover merupakan sebuah mekanisme pergantian sebuah layanan seperti *software* atau *hardware* dari sebuah *node* yang sedang mengalami *failure* ke *node* lain yang tersedia didalam sistem. Mekanisme ini dapat berjalan secara otomatis dan menggantikan layanan yang mengalami *failure* seperti sebelumnya (Jayaswal, 2005).

2.2 Autoscaling

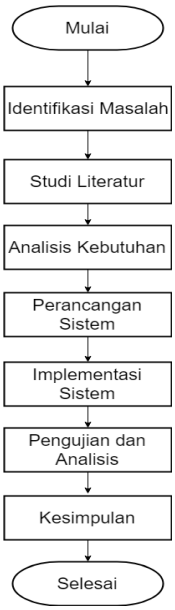
Scaling merupakan kemampuan sistem untuk menyesuaikan sumber daya yang dimiliki seperti menurunkan atau menambah jumlah proses sesuai kebutuhan sistem tanpa mengganggu proses yang sedang berjalan (Bhowmik, 2017).

2.3 Kubernetes

Kubernetes merupakan *platform open source* yang berfungsi sebagai kontainer *orchestration* yaitu *platform* yang akan bertugas melakukan penjadwalan, *scaling*, *recovery* dan monitoring pada kontainer (Medel, Rana, Bañares, & Arronategui, 2016).

3. METODOLOGI

Gambar 1 menunjukan metodologi



Gambar 1. Metodologi penelitian

didalam penelian yang akan dilakukan. Dimulai dari proses identifikasi masalah, yaitu bagaimana cara merancang dan mengimplementasikan *web server* berbasis kontainer dengan menggunakan Kubernetes. Lalu studi literatur mengenai konsep virtualisasi, Kubernetes, *autoscaling*, dan *failover*. Setiap kebutuhan sistem seperti fungsional dan non fungsional akan dianalisis untuk keperluan perancangan dan implementasi. sistem dibuat dalam lingkungan uji virtual untuk memudahkan penelitian. Setelah sistem diuji dan dianalisis akan ditarik kesimpulan dan saran untuk penelitian selanjutnya.

4. PERANCANGAN DAN IMPLEMENTASI

4.1 Kebutuhan Fungsional Sistem

Kebutuhan fungsional daris sistem yang dibangun adalah

1. sistem harus dapat menjalankan *web server* secara clustering menggunakan Kubernetes
2. Sistem mampu menjalankan mekanisme *failover* dan *autoscaling* pada layanan *web server*.

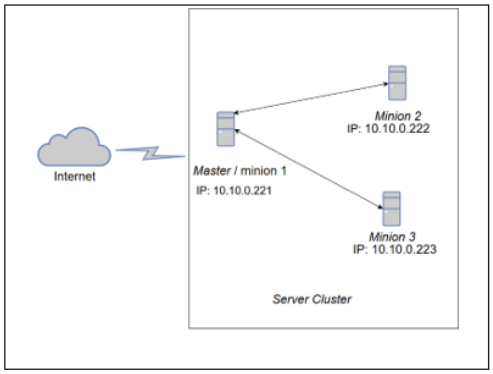
4.2 Kebutuhan Non Fungsional

Kebutuhan non fungsional diperlukan untuk menguji perilaku dari sistem, berikut kebutuhan fungsional yang dibutuhkan:

1. Sistem mampu menjalankan *web server* NGINX
2. Sistem mampu menyediakan layanan *web server* bersifat *high availability*
3. Sistem mampu mengurangi CPU *usage*.

4.3 Perancangan Sistem

Sistem yang dibangun akan menggunakan lingkungan virtual dimana terdapat 3 *node*. 1 *node* akan beperan sebagai *master* dan *minion*, serta 2 *node* akan berperan sebagai *minion* didalam *cluster*, *web server* yang digunakan adalah NGINX dimana *web server* tersebut telah tersedia secara *images* pada Docker. Gambar 2 menunjukan topologi jaringan yang akan dibangun.



Gambar 2. Topologi jaringan

5. PENGUJIAN DAN ANALISIS HASIL

5.1 Pengujian Fungsional

5.1.1 Pengujian Layanan *Web Server*

Pengujian ini untuk mengetahui apakah sistem mampu menyediakan layanan *web server*. NGINX akan dideploy kedalam sistem dan akan meng-*hosting hompage default* dari NGINX, berikut hasil dari pengujian layanan

Gambar 3. Layanan *web server* NGINX

Gambar 3 menunjukan bahwa *web server* telah berhasil dijalankan pada sistem tanpa kendala.

5.1.2 Pengujian *Failover*

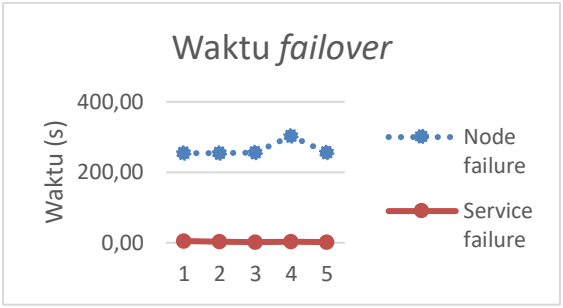
Pengujian ini bertujuan untuk mendapatkan berapa rata-rata waktu yang dibutuhkan pada saat mekanisme *failover* berjalan. Pengujian dilakukan dengan dua skenario yaitu *node failure* dan *service failure.*

Tabel 1. Pengujian *node failure*

Pengujian	Waktu *failover (s)*
1	255
2	254.4
3	255.6
4	303
5	255.6
Rata-rata	264.74

Tabel 2. Pengujian *service failure*

Pengujian	Waktu *failover (s)*
1	5
2	3
3	2
4	3
5	2
Rata-rata	3



Gambar 4. Grafik perbandingan waktu *failover*

Tabel 1 menunjukan bahwa rata-rata waktu yang dibutuhkan untuk melakukan failover adalah 264.74s untuk *node failure*, sementara Tabel 2 menunjukan rata-rata waktu untuk *service failure adalah* 3s, terdapat perbedaan sebesar 261.74s pada terhadap kedua proses. Gambar 4 menunjukan perbandingan rata-rata waktu yang dibutuhkan untuk proses *failover* dimana terdapat selisih waktu yang cukup besar pada proses *node failure* dikarenakan *node* yang mengalami *failure* tidak dapat mengirim informasi kepada *master* sehingga waktu yang dibutuhkan oleh *node master* lebih lama.

5.1.3 Pengujian *Autoscaling*

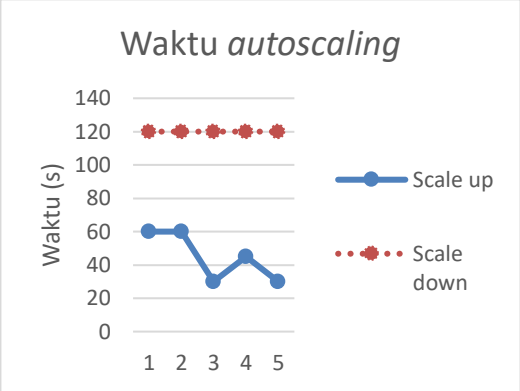
Pengujian ini bertujuan untuk mendapatkan berapa hasil rata-rata waktu yang dibutuhkan pada proses *autoscaling*. Proses *autoscaling* terbagi menjadi *scale up* dan *scale down* dimana *scale up* berarti menambah layanan, dan *scale down* mengurangi layanan *web server*. *web server* akan diberikan *request* secara terus menererus hingga CPU *usage* mencapai parameter yang telah ditentukan.

Tabel 3.Waktu *scale up*

Percobaan	Waktu (s)
1	60
2	60
3	30
4	45
5	30
Rata-rata	45

Tabel 4. Waktu *scale down*

Percobaan	Waktu (s)
1	120
2	120
3	120
4	120
5	120
Rata-rata	120



Gambar 6. Grafik perbandingan waktu autoscaling

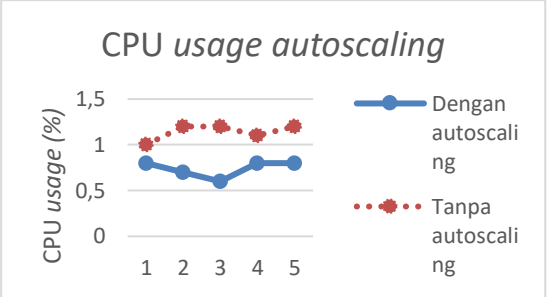
Pada Tabel 3 menunjukan bahwa rata-rata waktu yang dibutuhkan untuk *scale up* adalah 45s, dan pada Tabel 4 waktu rata-rata yang dibutuhkan untuk *scale down* adalah 120s. Gambar 5 menunjukan perbandingan waktu *autoscaling* yang terjadi. Terdapat selisih rata-rata waktu 63s pada proses *scaling*. Proses *scale down* memakan waktu yang lebih lama karena saat *traffic request* menurun karena sistem akan memberi delay sebelum melakukan *scale down* sehingga ketika *traffic request* kembali meningkat maka sistem tidak perlu melakukan *scaling* kembali.

5.1.4 Pengujian CPU *Usage*

Pengujian CPU *usage* bertujuan untuk mendapatkan data apakah sistem *autoscaling* pada Kubernetes mampu mengurangi CPU *usage* ketika sistem mengalami lonjakan *traffic request*. *Web server* akan dikirim *request* secara berulang hingga *autoscaling* terjadi.

Tabel 5. hasil pengujian *CPU usage*

Percobaan	CPU *usage* (%)	
	Dengan *autoscaling*	Tanpa *autoscaling*
1	0.8	1
2	0.7	1.2
3	0.6	1.2
4	0.8	1.1
5	0.8	1.2
Rata-rata	0.74	1.14



Gambar 5. Grafik perbandingan *CPU usage*

Tabel 5 menunjukan data pengujian yang didapat setelah *web server* diberi *request* secara berulang. Gambar 5 menunjukan grafik perbandingan antara *web server* yang menggunakan *autoscaling* dan tidak. Terdapat perbedaan 0.4% CPU *usage* dimana *web server* yang menggunakan *autoscaling* memiliki CPU *usage* lebih rendah. Hal ini dapat terjadi karena saat proses *scale up* layanan Kubernetes akan membagi beban kerja terhadap *server* yang memiliki layanan *web server* sehingga CPU *usage* menjadi lebih rendah.

6. PENUTUP

6.1 Kesimpulan

1. Sistem yang dibangun mampu mengimplementasikan *web server* berbasis *kontainer* menggunakan Kubernetes
2. Sistem mampu mengimplementasikan mekanisme *failover* dan *autoscaling* kontainer *web server* menggunakan Kubernetes
3. *Failover* memiliki dua hasil rata-rata waktu yang berbeda diamana proses *failover* pada *system failure* memiliki rata-rata waktu 261.74s lebih cepat daripada *node failure.* Proses *autoscaling* memiliki hasil rata-rata waktu yang berbeda dimana *proses scale up* memiliki waktu 63s lebih cepat daripada *scale down*, serta *autoscaling* membuat CPU *usage* menjadi 0.4% lebih hemat dibanding

web server yang tidak menggunakan *autoscaling*.

6.2 Saran

Pada penelitian berikutnya diharapkan Sistem yang dibangun dapat diimplementasikan pada *server* fisik sebenarnya dengan menambah fitur *persistent storage* agar data yang dihasilkan oleh aplikasi tidak hilang ketika aplikasi dihapus.

7. DAFTAR PUSTAKA

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., & Katz, R. (2009). *Above the Clouds: A Berkeley View of Cloud.* California: UC Berkeley Reliable Adaptive Distributed Systems Laboratory.

Bhowmik, S. (2017). *Cloud Computing.* India: Cambridge University Press.

Bondi, A. B. (2000). Characteristics of scalability and their impact on performance. *2nd international workshop on Software and performance* , 195-203.

Gourley, D., & Totty, B. (2002). *HTTP: The Definitive Guide* (1st ed.). Sebastopol: O'reilly Media.

Jayaswal, K. (2005). *Administering Data Centers: Servers, Storage, And Voice Over Ip.* new delhi: Wiley India Pvt. Limited.

Kubernetes. (2017, agustus 8). *What is Kubernetes*. Diambil kembali dari kubernetes.io: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Medel, V., Rana, O., Bañares, J. Á., & Arronategui, U. (2016). Modelling Performance & Resource Management in. *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing*, 257.

NGINX. (2018). *Welcome to NGINX Wiki!* Retrieved maret 11, 2018, from https://www.nginx.com/resources/wiki/

oracle. (2018). *Avoiding and Recovering From Server Failure*. Retrieved maret 3, 2018, from https://docs.oracle.com/cd/E13222_01/wls/docs90/server_start/failures.html

Oracle. (2018). *Understanding WebLogic Server Clustering* . Retrieved maret 11, 2018, from https://docs.oracle.com/cd/E11035_01/wls100/cluster/overview.html

Yuhefizar, Mooduto, H., & Hidayat, R. (2008). *Cara Mudah Membangun Website Interaktif Menggunakan Content Management System Joomla* (2nd ed.). Jakarta: Elex Media Komputindo.