

**ANALISIS DAN IMPLEMENTASI *VERTICAL AUTOSCALING*
WEB APLIKASI MENGGUNAKAN KUBERNETES**

Analysis and Implementation of Vertical Autoscaling Web Applications using Kubernetes

PROPOSAL PROYEK AKHIR

Diajukan sebagai syarat untuk mengambil Mata Kuliah Proyek Akhir

oleh :

AULIA RIZQI PUTRA

6705180006



**D3 TEKNOLOGI TELEKOMUNIKASI
FAKULTAS ILMU TERAPAN
UNIVERSITAS TELKOM
2020**

LEMBAR PENGESAHAN

Proposal Proyek Akhir dengan judul :

ANALISIS DAN IMPLEMENTASI *VERTICAL AUTOSCALING* WEB APLIKASI MENGUNAKAN KUBERNETES

Analysis and Implementation of Vertical Autoscaling Web Applications using Kubernetes

oleh :

AULIA RIZQI PUTRA
6705180006

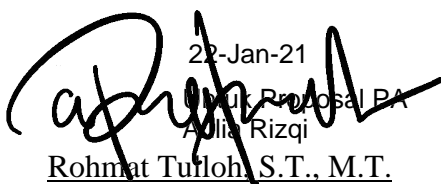
Telah diperiksa dan disetujui untuk diajukan sebagai syarat mengambil
Mata Kuliah Proyek Akhir
pada Program Studi D3 Teknologi telekomunikasi Universitas Telkom

Bandung, 19 Januari 2020


Menyetujui,

Pembimbing I

Pembimbing II

22-Jan-21
Untuk Proposal PA
Aulia Rizqi

Rohmat Tuflohi, S.T., M.T.

NIP. 06830002


Proposal PT
Aulia Rizqi
2021-01-19 19:
03:41

Muhammad Iqbal, S.T., M.T.

NIP. 10840012

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 manajemen 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.

Pada penelitian ini, telah diimplementasikan layanan *web e-Commerce* berbasis *platform* Kubernetes dengan fitur *vertical autoscaling* agar system secara otomatis dapat menyesuaikan kebutuhan dengan permintaan user. Untuk mendukung layanan *web e-Commerce* dibutuhkan beberapa aplikasi yang saling terintegrasi yaitu *reaction commerce* untuk *front-end system*, mongoDB untuk *back-end system*, dan Minikube untuk membuat sebuah Kubernetes secara local. Untuk meninjau performa sistem, dilakukan pengujian berbeda dengan menggunakan satu master node, dan dua worker node.

Parameter pengukuran yang ditinjau dilihat dari performansi *server* berdasarkan *scalling up*, dan *scalling down*, serta performansi aplikasi berdasarkan *response time*, *throughput*, *delay*, dan *request*. Hasil penelitian dapat diharapkan pada performansi *server* berdasarkan *scalling up* dibawah 50 *second* dan *scalling down* dibawah 130 *second*, serta performansi aplikasi berdasarkan *response time* dibawah 40 *second*, rata – rata *throughput* 3 MB/s, *delay* sekitar 200 ms – 500 ms, dan dengan *request* 500 user.

kata kunci : Kubernetes, Docker, Minikube, *autoscalling*

DAFTAR ISI

LEMBAR PENGESAHAN	i
ABSTRAK	ii
DAFTAR ISI	iii
DAFTAR GAMBAR.....	v
DAFTAR TABEL	vi
BAB I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Tujuan dan Manfaat	2
1.3 Rumusan Masalah.....	2
1.4 Batasan Masalah	2
1.5 Metodologi.....	3
BAB II DASAR TEORI.....	4
2.1 <i>Cloud Computing</i>	4
2.1.1 <i>Software as a Service</i>	4
2.1.2 <i>Platform as a Service</i>	4
2.1.3 <i>Infrastructure as a Service</i>	4
2.2 <i>Virtualization dan Containerization</i>	5
2.3 Docker.....	6
2.3.1 Arsitektur Docker	6
2.4 <i>Container Orchestration</i>	8
2.4.1 Kubernetes	8
2.4.2 Arsitektur Kubernetes.....	9
2.4.2.1 Master	9

2.4.2.2 Worker	10
2.5 Minikube	11
2.6 <i>Vertical Pod Autoscaling</i>	11
2.7 Reaction Commerce	11
2.8 MongoDB	12
2.9 <i>Load Balancing</i>	12
BAB III MODEL SISTEM	14
3.1 Blok Diagram Sistem	14
3.2 Tahapan Perancangan	16
3.3 Perancangan	16
BAB IV BENTUK KELUARAN YANG DIHARAPKAN	17
4.1 Keluaran yang Diharapkan	17
4.2 Jadwal Pelaksanaan	17
DAFTAR PUSTAKA	18

DAFTAR GAMBAR

Gambar 2. 1 Perbedaan Arsitektur <i>Virtualization</i> dan <i>Containerization</i> [3].....	6
Gambar 2. 2 Arsitektur Docker [8].....	8
Gambar 2. 3 Arsitektur Kubernetes [14]	9
Gambar 2. 4 <i>Load balancing</i> pada <i>cloud</i>	12
Gambar 3. 1 Model Sistem.....	14
Gambar 3. 2 Diagram Alir Perancangan	16

DAFTAR TABEL

Tabel 4. 1 Jadwal Pelaksanaan	17
-------------------------------------	----

BAB I

PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi semakin cepat, termasuk datangnya teknologi baru yaitu *container*. *Container Orchestration* adalah salah satu teknologi *Container*. Dengan *Container Orchestration* proses pembuatan maupun penggunaan *system* tersebut akan semakin mudah tetapi seiring dengan permintaan pengguna yang terlalu banyak sehingga layanan tersebut tidak berjalan maksimal. Oleh karena itu *Container Orchestration* harus memiliki skalabilitas dan performansi yang bagus. Skalabilitas di perlukan untuk *system* dapat menyesuaikan kebutuhan dengan permintaan user . Dan performansi di perlukan untuk menjaga kualitas layanan yang diberikan. Pada sistem web hosting modern, di dalam setiap servernya, mengelola banyak aplikasi web.

Teknologi *virtual machine* dimanfaatkan untuk menyelesaikan masalah heterogenitas (perbedaan versi *library* atau *tool* dari beberapa aplikasi web). Peningkatan jumlah aplikasi web yang harus dihosting harus diikuti dengan peningkatan kualitas ataupun kuantitas sumber daya, terlebih saat hadirnya kebutuhan *high availability* dari layanan web tersebut. Teknik kontainerisasi (virtualisasi berbasis *container*) hadir sebagai solusi dan menjadi trend saat ini. Kubernetes adalah *platform open source* yang digunakan untuk manajemen *container* dan sebagai *container orchestration* yaitu *platform* yang akan bertugas melakukan penjadwalan, *scaling*, *recovery* dan monitoring pada *container*.

Pada penelitian ini, akan dikembangkan layanan web e-commerce pada *platform* berbasis Kubernetes dengan menggunakan *vertical autoscaling*, selanjutnya akan dilakukan simulasi untuk melihat gambaran performa sistem yang telah dikembangkan berdasarkan parameter *Load Testing* untuk skalabilitas, waktu *scaling up* dan *scaling down* untuk performansi, *throughput*, *response time*, *request*, dan *delay*. Hasil simulasi menunjukkan sistem yang dibangun mampu memenuhi standar dengan performa baik ketika menerima sejumlah *http load*.

1.2 Tujuan dan Manfaat

Adapun tujuan dari Proyek akhir ini, sebagai berikut:

1. Dapat mengimplementasikan *platform* berbasis Kubernetes.
2. Dapat mengimplementasikan layanan web e-commerce pada *platform* berbasis Kubernetes.
3. Dapat mengimplementasikan Kubernetes, Docker, dan Minikube secara local dalam satu laptop atau pc.
4. Mengukur performansi layanan web e-commerce dengan parameter *scaling up, scaling down, response time, throughput, delay, dan request*.

Adapun manfaat dari Proyek akhir ini, sebagai berikut:

1. Mempermudah hardware menyesuaikan kapasitas untuk mempertahankan kinerja yang stabil karena fitur *vertical pod autoscaling*
2. Mampu mendukung jumlah pemuatan atau kompleksitas proses dalam website / aplikasi yang tidak menentu.

1.3 Rumusan Masalah

Adapun rumusan masalah dari Proyek akhir ini, sebagai berikut:

1. Bagaimana mengukur performansi layanan web e-commerce dengan parameter *scaling up, scaling down, response time, throughput, delay, dan request*.
2. Bagaimana mengimplementasikan layanan web e-commerce pada *platform* berbasis Kubernetes?
3. Bagaimana sistem kerja dari *platform* berbasis Kubernetes?
4. Perangkat apa saja yang akan dibutuhkan untuk mengimplementasikan layanan web e-commerce pada *platform* berbasis Kubernetes?

1.4 Batasan Masalah

Adapun yang menjadi Batasan masalah pada Proyek akhir ini antara lain:

1. Hanya mengimplementasikan *platform* berbasis Kubernetes.
2. Fokus pengukuran hanya pada performansi *server* berdasarkan *scaling up*, dan *scaling down*, serta performansi aplikasi berdasarkan *response time, throughput, delay, dan request*.

3. Implementasi hanya sebatas pada pembuktian bahwa layanan web e-commerce dapat berjalan pada *platform* yang dibangun.
4. Layanan yang digunakan melalui *http*.

1.5 Metodologi

Metodologi yang digunakan dalam penyusunan Proyek akhir ini sebagai berikut :

1. Melakukan pencarian dan pengumpulan informasi yang berhubungan dengan tugas akhir ini melalui sumber jurnal, *internet*, dan buku referensi.
2. Melakukan persiapan terhadap perancangan yang akan dilakukan.
3. Melakukan implementasi layanan web e-commerce pada *platform* berbasis Kubernetes berdasarkan referensi yang didapatkan dari berbagai studi literatur.
4. Melakukan pengujian terhadap layanan web e-commerce pada *platform* berbasis Kubernetes yang telah dibuat serta mengumpulkan data yang kemudian digunakan untuk analisa parameter *scalling up*, *scalling down*, *response time*, *throughput*, *delay*, dan *request*.

BAB II

DASAR TEORI

2.1 *Cloud Computing*

Cloud Computing adalah sebuah model komputasi yang membuat *resource Information and Technology (IT)* seperti *server, network, middleware, storage*, dan aplikasi dapat diakses secara *public, private*, maupun *hybrid* kapan pun melalui medium internet. berdasarkan model penyampaian (*delivery model*) *cloud computing* terbagi atas 3 jenis, diantaranya adalah.

2.1.1 *Software as a Service*

Software As A Service (SaaS) adalah sebuah layanan yang disediakan oleh *cloud* provider berupa aplikasi yang dapat diakses melalui web browser tanpa melalui proses download dan install aplikasi yang ingin digunakan oleh pengguna sehingga pengguna dapat langsung menggunakan aplikasi tersebut ataupun berupa aplikasi yang terhubung pada layanan yang berjalan di *cloud*. Contoh dari layanan SaaS adalah google aps (google docs dan google spreadsheets), *salesforce customer relationships management (CRM) system*, Microsoft dynamic, dll.

2.1.2 *Platform as a Service*

Platform As A Service (PaaS) adalah sebuah layanan yang menyediakan lingkungan yang terdiri dari berbagai macam komponen yang dibutuhkan untuk mengembangkan aplikasi melalui medium internet. Layanan ini biasa digunakan oleh *developer* yang membutuhkan jenis *Application Programming Interfaces (API)* yang spesifik untuk mengembangkan, menguji coba, dan mengatur aplikasinya melalui *cloud platform*. Contoh dari layanan PaaS adalah Google App Engine yang menyediakan *Python and Java runtime environments* dan API sehingga aplikasi dapat berinteraksi dengan *Google's runtime environment*.

2.1.3 *Infrastructure as a Service*

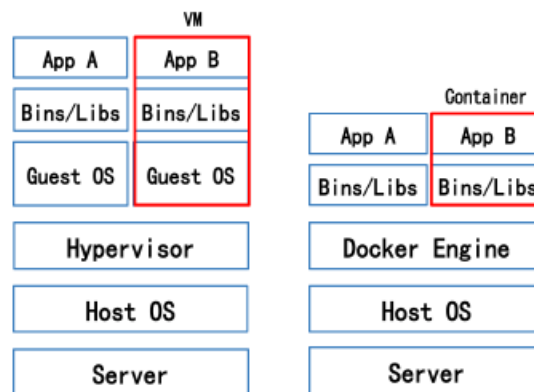
Infrastructure As A Service (IaaS) adalah sebuah layanan yang menyediakan beragam sumber daya virtual (*computation, storage, and communication*) yang dapat dibeli sesuai dengan kebutuhan pengguna. Pengguna

mendapatkan akses penuh untuk menjalankan operasi pada server tersebut seperti menjalankan dan menghentikan *server*, melakukan pengaturan *server* seperti instalasi paket *software*, menambah *virtual disk*, serta konfigurasi perizinan akses dan pengaturan *firewall*. Contoh dari penyedia layanan IaaS adalah Amazon EC2 dan Windows Azure.

2.2 *Virtualization dan Containerization*

Teknologi virtualisasi merupakan hal yang fundamental dalam dunia *cloud computing*. Dimulai dari penyedia layanan *cloud* IaaS yang menyediakan *virtual machine*, hingga ke PaaS dan SaaS yang membangun dan menjalankan layanannya lewat *virtual machine*. Virtualisasi dapat didefinisikan sebagai abstraksi dari empat sumber daya komputasi (*storage, processing power, memory, and network or I/O*) [2] yang memungkinkan satu perangkat keras untuk dialokasikan menjadi beberapa perangkat virtual yang dapat beroperasi secara independen agar dapat menjalankan berbagai macam jenis aplikasi yang dibutuhkan di waktu yang bersamaan untuk menyediakan sebuah layanan. Namun pada sistem berskala besar yang terus berkembang sistem yang dibangun dengan fondasi arsitektur *microservice* teknologi virtualisasi kurang efisien dalam penggunaan *resource* karena pada arsitektur *microservice* sebuah sistem terdiri dari banyak pecahan komponen aplikasi dibandingkan dengan sistem yang berbentuk satu kesatuan aplikasi yang dibangun dengan arsitektur monolitik, sehingga jika kita ingin melakukan proses isolasi pada setiap aplikasi yang dibangun pada *virtual machine* akan menghabiskan *resource* dalam jumlah yang besar, dengan rata-rata minimal *virtual machine* memakan kapasitas per gigabyte hanya untuk menjalankan sebuah aplikasi walaupun ukuran aplikasi tersebut relatif jauh lebih kecil. Hal ini dikarenakan *virtual machine* dibangun atas keseluruhan sistem yang dimulai dari *operating system (OS)*, *Virtualization Layer* berupa *hypervisor* yang menjembatani komunikasi dan kinerja diantara *host operating system* dan *virtual machine / guest operating system*, dan program-program lainnya yang perlu diinstalasi dan dijalankan untuk mendukung kinerja *virtual machine* [3].

Teknologi *container* dikembangkan untuk menyederhanakan segala persoalan yang telah dijabarkan. *Container* adalah *lightweight operating system* yang dapat bekerja secara langsung didalam *host operating system*. *Container* menjalankan segala proses instruksi secara langsung ke *core CPU*. *Container* mampu memberikan penghematan penggunaan resource tanpa *overhead* dari *virtualization* serta menjamin kinerja aplikasi yang terisolasi [4]. Besaran *container* juga jauh lebih kecil jika dibandingkan dengan *virtualization*, biasanya ukuran *container* hanya berskala megabyte karena *container* hanya berisi paket aplikasi yang dibutuhkan tanpa *operating system*. Dan tanpa adanya *hypervisor* performansi aplikasi yang dijalankan jauh lebih baik, terutama jika aplikasi membutuhkan proses interaksi dengan *I/O devices* [3].



Gambar 2. 1 Perbedaan Arsitektur *Virtualization* dan *Containerization* [3]

2.3 Docker

Docker adalah sebuah aplikasi open source yang berfungsi untuk mengembangkan, mendistribusi, serta menjalankan *software application*. Desain arsitektur Docker memudahkan untuk mendistribusi serta mengembangkan aplikasi secara lebih cepat karena Docker memiliki sifat *lightweight containerization* yang dilengkapi dengan beragam komponen serta fitur sehingga mampu memudahkan para *developer* untuk mengembangkan dan memantau kinerja aplikasi yang diciptakan [5].

2.3.1 Arsitektur Docker

Arsitektur Docker dibangun dengan fondasi *Linux container* (LXC) dengan fitur-fitur seperti *cgroups* dan *namespaces* untuk mengatur sumber daya komputasi dan proses isolasi yang kuat. Dengan hal ini Docker mampu memproses kinerja

sistem serupa dengan *kernel based virtual machine* (KVM) dengan *resource* yang lebih sedikit [6].

Linux container (LXC) adalah *OS level virtualization* yang mampu menjalankan beberapa proses *linux system* sekaligus secara terisolasi dalam sebuah *linux control host* dan menjadi penjemputan dengan kinerja fitur-fitur sistem linux kernel. *Linux container* (LXC) dibangun dengan basis sistem *chroot* yang termuat atas *binaries*, *libraries*, dan file konfigurasi yang disebut *chroot jail* sehingga mampu membuat lingkungan pengembangan yang terisolasi yang berjalan diatas kernel untuk aplikasi [7].

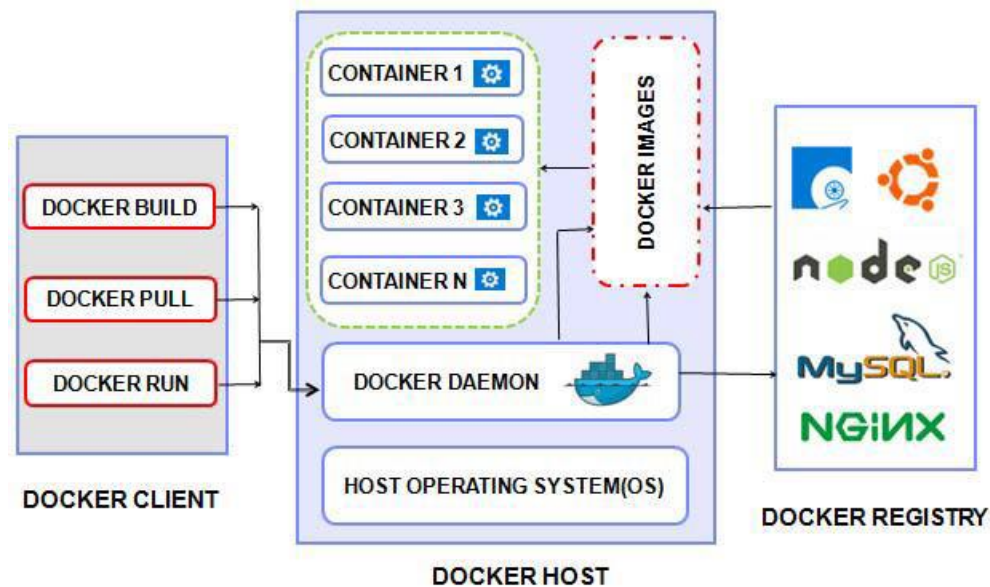
Cgroups adalah kumpulan *resource* yang dapat dibuat pada *Linux kernel level*. *Cgroups* bertugas untuk membatasi penggunaan *resource* sehingga setiap *virtual machine/container* hanya menyerap *resource* sesuai kebutuhan, hal ini menjadikan teknologi *containerization* memiliki lingkungan pengembangan yang efisien.

Docker mempunyai dua komponen utama, yaitu *open source containerization platform* yang disebut Docker dan Docker hub yang merupakan *Software-as-a-Service* (SaaS) *platform* untuk berbagi dan mengatur Docker container, Docker menggunakan model arsitektur client-server, dimana Docker *client* dapat terhubung ke Docker daemon yang membuat, menjalankan, serta mendistribusikan Docker container.

Docker *client* dan Docker daemon dapat berjalan diatas sistem yang sama ataupun Docker *client* dapat melakukan komunikasi melewati sockets atau RESTful API ke Docker daemon secara *remote* [6].

Docker daemon berjalan di *host machine* dan pengguna terhubung dengan Docker daemon melalui Docker *client*. Fungsi dari Docker *client* adalah menerima serangkaian perintah dari pengguna lalu menghubungkan perintah tersebut agar dapat diproses oleh Docker daemon.

Docker *images* berbentuk *read-only templates* dan Docker registries menampung Docker *images* tersebut. Docker *container* terbuat dari Docker *image* yang berisi segala paket yang dibutuhkan untuk menjalankan suatu aplikasi secara terisolasi [8].



Gambar 2. 2 Arsitektur Docker [8]

2.4 Container Orchestration

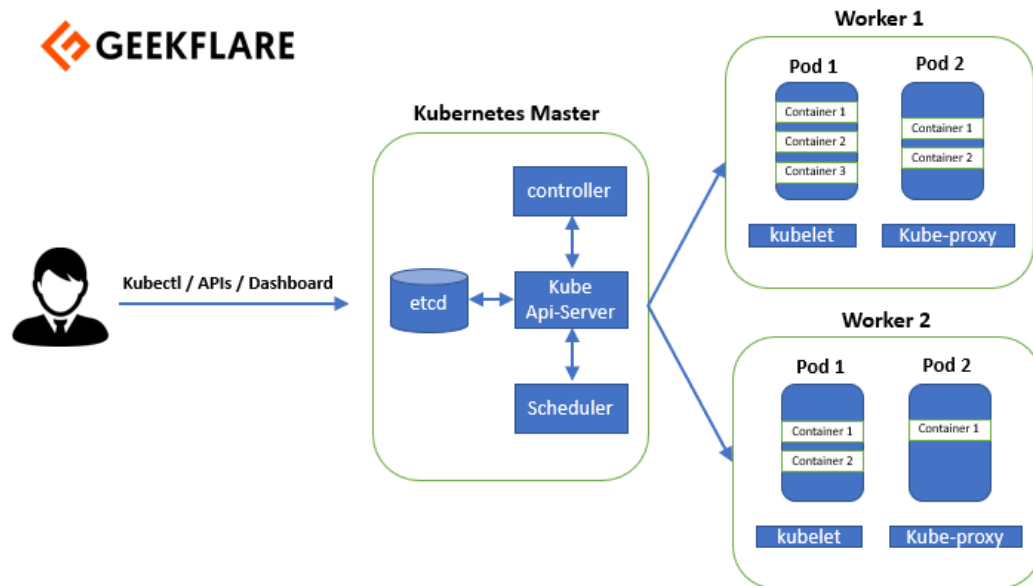
yang dapat menyederhanakan dalam membangun, menyebarkan, dan memelihara *container* pada seluruh *server*. Dengan mengotomatiskan distribusi aplikasi pada *cluster*, memastikan tingkat pemanfaatan yang lebih tinggi, dan sebagai *platform* tunggal dalam penyebaran aplikasi pada *server* dan *cloud* [19]. Salah satu contoh *container orcheatration* adalah Kubernetes.

2.4.1 Kubernetes

Kubernetes merupakan *platform open source* yang digunakan untuk melakukan manajemen *workload* aplikasi yang dikontainerisasi, serta menyediakan konfigurasi dan otomatisasi secara deklaratif. Kubernetes berada di dalam ekosistem yang besar dan berkembang cepat. *Service* dan perkakas Kubernetes tersedia secara meluas [13]. Dengan menggunakan Kubernetes, *container* lebih mudah untuk ditingkatkan jumlahnya, dihancurkan, kemudian dibuat kembali. Terdapat banyak hal yang melatarbelakangi mengapa banyak menggunakan *container* dan *container API* seperti Kubernetes, diantaranya, adalah kecepatan, *scaling*, meringkaskan infrastruktur, efisiensi [12].

2.4.2 Arsitektur Kubernetes

Ada dua jenis *node* dalam Kubernetes yang memiliki komponen yang berbeda-beda, seperti pada Gambar II-3. Berikut penjelasan komponen dalam Kubernetes [14].



Gambar 2. 3 Arsitektur Kubernetes [14]

2.4.2.1 Master

Master berperan dalam mengelola kondisi dalam *cluster*. Di dalamnya terdapat komponen-komponen yang memanajemen operasi dalam *cluster*. Sebagian besar komponen dari Master disebut dengan *control plane*. *Control plane* merupakan agen-agen dengan peran yang sangat penting dalam manajemen *cluster*.

Komponen Master menyediakan *control plane* bagi *cluster*. Komponen ini berperan dalam proses pengambilan secara global pada *cluster*, serta berperan dalam proses deteksi dan pemberian respon terhadap *event* yang berlangsung di dalam klaster. Komponen pada Master terdiri atas:

a. Kube API Server

Komponen Master yang berperan dalam memberikan interupsi atas permintaan layanan dengan arsitektur RESTful dari pengguna untuk memvalidasi dan memproses permintaan yang telah diterima. Selama proses berlangsung, komponen ini membaca status terbaru dalam *cluster* yang

didapatkan dari etcd, kemudian mengeksekusikannya dan menyimpan kembali status terakhir ke dalam etcd.

b. Etcd

Komponen Master yang berperan dalam menyimpan data (*data store*) konfigurasi atau status terbaru dalam *cluster*, namun tidak dengan *workload* dari *client*. Hanya komponen Kube API Server yang dapat berkomunikasi dengan etcd.

c. Kube Scheduler

Komponen Master yang berperan dalam melakukan penetapan (*assign*) pada objek baru, contohnya Pod dengan *node* tertentu. Komponen ini mendapatkan informasi dalam etcd melalui komponen Kube API Server. Informasi tersebut berupa informasi mengenai sumber daya yang dibutuhkan dalam *Worker*. Sehingga semua keputusan dalam proses *scheduling* diambil berdasarkan status dalam *cluster* dan permintaan ataupun persyaratan dari objek baru yang akan ditetapkan.

d. Controller Manager

Komponen Master yang berperan dalam mengontrol dan memberikan aturan mengenai status dalam *cluster*. Komponen ini selalu memonitor dan membandingkan antara status *cluster* yang diinginkan dengan status *cluster* yang sekarang.

2.4.2.2 Worker

Worker berperan dalam menyediakan lingkungan kerja untuk layanan dari *client*. Dengan layanan-layanan kecil yang berinteraksi kemudian dibungkus dalam *container*. Kumpulan *container* dibungkus ke dalam Pod. Pod bekerja dalam Worker dengan dikontrol oleh *control plane* dalam Master. Pod membutuhkan sumber daya komputasi, *storage*, *network* sendiri agar bisa saling berkomunikasi dengan yang lain. Komponen pada Worker terdiri atas:

a. Kubelet

Komponen Worker yang berperan sebagai agen dalam menjalankan *node* dan berkomunikasi dengan komponen *control plane* dalam Master. Komponen ini menerima informasi dari komponen Kube API Server untuk

dikomunikasikan ke dalam Pod. Komponen ini juga berkomunikasi dengan Container Runtime untuk dapat menjalankan *container* yang ada dalam Pod.

b. Kube Proxy

Komponen Worker yang berperan sebagai agen jaringan dalam menjalankan *node*. Bertanggung jawab dalam melakukan pembaharuan secara dinamis dan aturan jaringan dalam *node*. Sehingga dapat memastikan bahwa *container* dapat dijalankan di dalam Pod.

c. Container Runtime

Komponen Worker yang berperan dalam menjalankan dan mengendalikan beberapa *container* yang ada dalam Pod. Kubernetes mendukung beberapa Container Runtime, diantaranya adalah: Docker, Containerd, Cri-o, Rktlet dan semua implementasi Kubernetes *Container Runtime Interface* (CRI).

2.5 Minikube

Minikube adalah alat yang memudahkan untuk menjalankan Kubernetes pada komputer lokal. Minikube menjalankan satu Node klaster Kubernetes di dalam *Virtual Machine* (VM) pada laptop kamu untuk pengguna yang ingin mencoba Kubernetes atau mengembangkannya [1].

2.6 Vertical Pod Autoscaling

Vertikal Pod autoscaling membebaskan developer untuk mengatur batas resource CPU, memori, dan permintaan untuk container di pod mereka atau pengguna. Autoscaler dapat merekomendasikan nilai untuk CPU dan permintaan dan batas memori, atau dapat secara otomatis memperbarui nilai sesuai dengan banyak user yang akses [20] .

2.7 Reaction Commerce

Reaction Commerce adalah pionir platform e-commerce yang mampu bekerja secara real-time, artinya data dari server selalu diperbarui secara seketika kapanpun sistem menerima interaksi maupun perintah dari pengguna tanpa harus melakukan *refresh page* tersebut [9]. Tentu hal ini memberikan pelayanan dan pengalaman berbelanja online yang lebih baik bagi pengguna karena informasi tentang produk yang ditawarkan dan ketersediaannya selalu terbaharui. Platform reaction Commerce dilengkapi dengan fitur yang sangat lengkap seperti

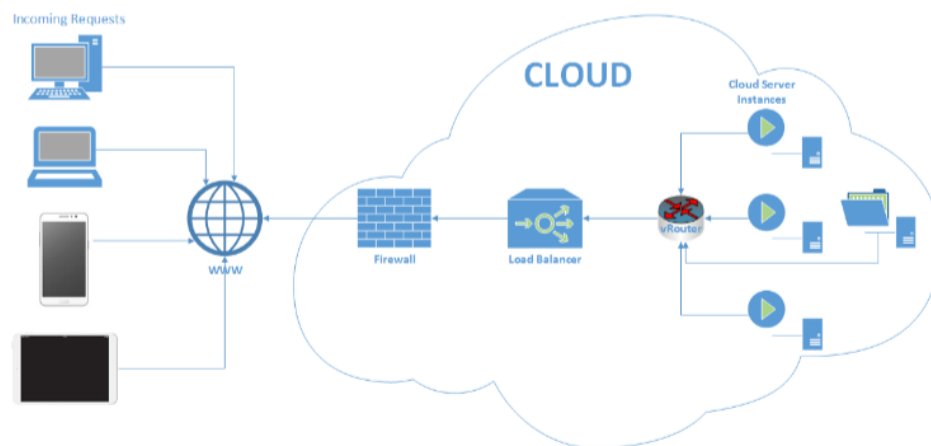
manajemen inventori, manajemen pemesanan, *profile* pengguna, notifikasi, *advanced search* dan lain-lain.

2.8 MongoDB

MongoDB adalah *cross-platform document-oriented database* program dan termasuk dalam klasifikasi NoSQL database program. Mekanisme NoSQL dikembangkan untuk menjawab kebutuhan pemrosesan data pada sistem dalam jumlah volume yang besar. Saat ini layanan internet seperti sosial media ataupun multimedia memiliki jumlah pengguna yang mampu mencapai angka hingga jutaan pengguna dan terus bertambah. Dengan konektivitas sepanjang hari tanpa waktu henti, mekanisme terdahulu yang disebut relational database tidak lagi efisien untuk menjawab kebutuhan database untuk sistem multimedia [10]. Sehingga mongoDB menjadi salah satu solusi atas permasalahan pemrosesan database yang berskala masif di era saat ini.

2.9 Load Balancing

Load balancing adalah teknik yang digunakan untuk mendistribusikan trafik yang masuk ke server yang tersedia sehingga permintaan dapat ditangani dengan cepat. Sistem jaringan komputasi yang kompleks melakukan perutean jutaan paket data setiap detiknya. Trafik data yang begitu besar harus didistribusikan secara efisien di antara server yang tersedia sehingga server dapat menanganinya dengan lebih baik dan cepat [11] .



Gambar 2. 4 Load balancing pada cloud

High availability merupakan salah satu faktor pendorong load balancing.

Keuntungan utama load balancing sebagai berikut [11] :

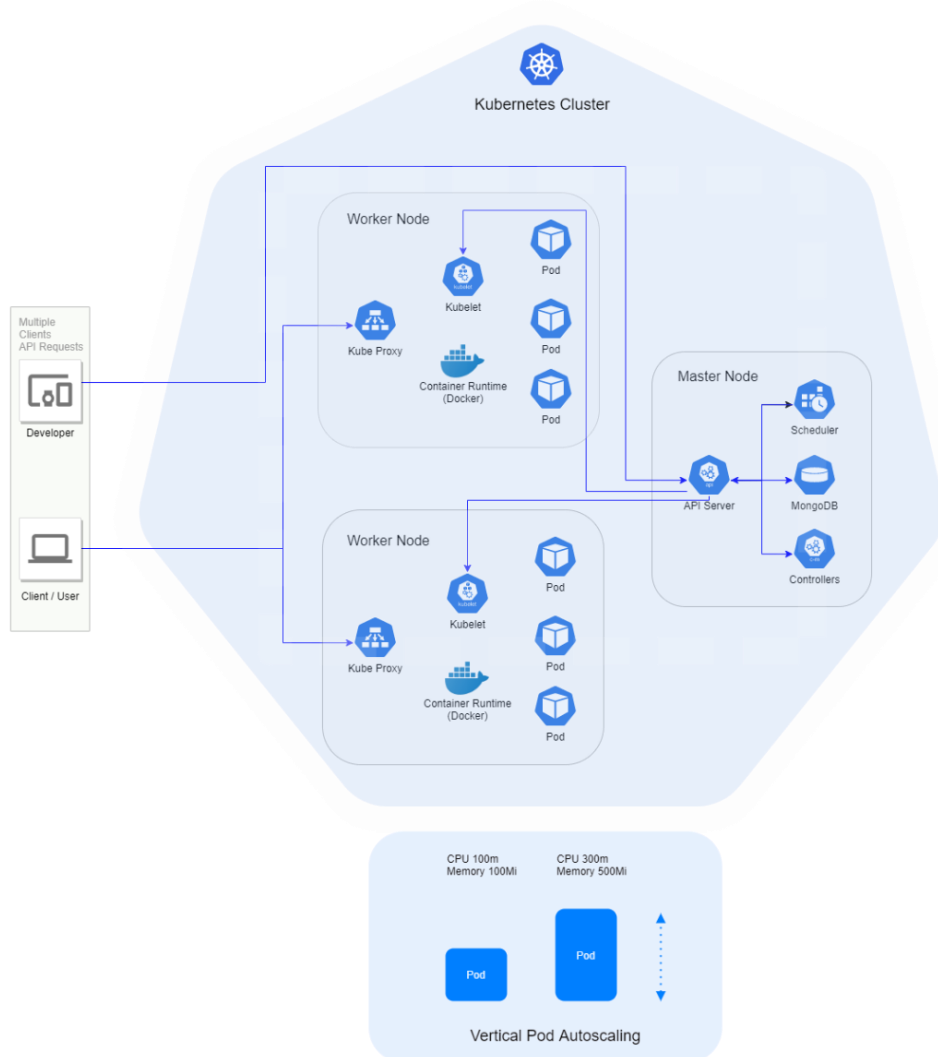
1. Membantu dalam mengendalikan dan melacak trafik.
2. Meningkatkan pemanfaatan sumber daya.
3. Menyeimbangkan beban jaringan.
4. Meningkatkan ketersediaan sumber daya.
5. Mengurangi *overprovision* pada infrastruktur.

BAB III

MODEL SISTEM

3.1 Blok Diagram Sistem

Pada bab ini akan dijelaskan mengenai analisis dan implelementasi *vertical autoscaling* web aplikasi menggunakan Kubernetes. Alat yang digunakan adalah satu buah laptop atau pc yang sudah terinstall Kubernetes sebagai *Container orchestration*, Docker sebagai *deploying image*, dan Minikube untuk membuat dan menjalankan sebuah cluster Kubernetes secara local yang terdiri dari tiga node.



Gambar 3. 1 Model Sistem

Pada gambar tersebut merupakan sebuah sistem yang merupakan layanan *web E-Commerce* yang dibangun menggunakan Kubernetes. Seluruh proses sistem yang bekerja terdapat pada Worker Node yang dikontrol oleh master node agar memudahkan penjadwalan serta pengalokasian sumber daya yang dibangun menggunakan teknologi Kubernetes serta *containerization* untuk membantu proses isolasi sistem aplikasi yang kompleks dan memungkinkan untuk scaling system sesuai perancangan dan kebutuhan hingga kapasitas tertentu sesuai dengan spesifikasi server. Sistem dapat dibangun dan dikembangkan melalui Docker yang merupakan sebuah *container management tools* sebagai medium penghubung dan pengatur dari beberapa Kubernetes *container* yang bekerja saling terkoneksi melalui medium web browser. Sistem juga dibangun menggunakan Minikube sebagai tools yang akan memudahkan untuk menjalankan Kubernetes pada komputer local dan Minikube juga digunakan untuk membuat multi node agar web aplikasi tersebut agar memudahkan untuk mempertahankan pod yang sedang berjalan secara terus menerus.

Desain sistem web *E-Commerce* yang dibangun terdiri dari variasi Reaction Commerce sebagai *front-end system* dan mongoDB sebagai *back-end system* yang bekerja saling terintegrasi untuk menciptakan layanan dengan performa terbaik. Sistem Reaction Commerce akan di scale yang dapat ditampung di host yang tersedia untuk meningkatkan performa server dan aplikasi. Kedua sistem tersebut terhubung berkat tools dan konfigurasi dari Kubernetes, terdapat konfigurasi automasi *load balancer* untuk membagi beban kerja server sehingga mampu menyajikan layanan *high performance*.

Sistem juga dibangun menggunakan vertical pod autoscaling sehingga saat user melakukan request secara bersamaan dengan waktu yang sama maka secara otomatis pods akan menyesuaikan menambah dan mengurangi sumber daya komputasi dari satu replica sehingga spesifikasi dari suatu pod juga akan berubah sesuai kebutuhan request user. Selanjutnya parameter yang akan diukur yaitu parameter *Load Testing* untuk skalabilitas, waktu *scaling up*, waktu *scaling down*, *throughput*, *response time*, *request*, dan *delay*.

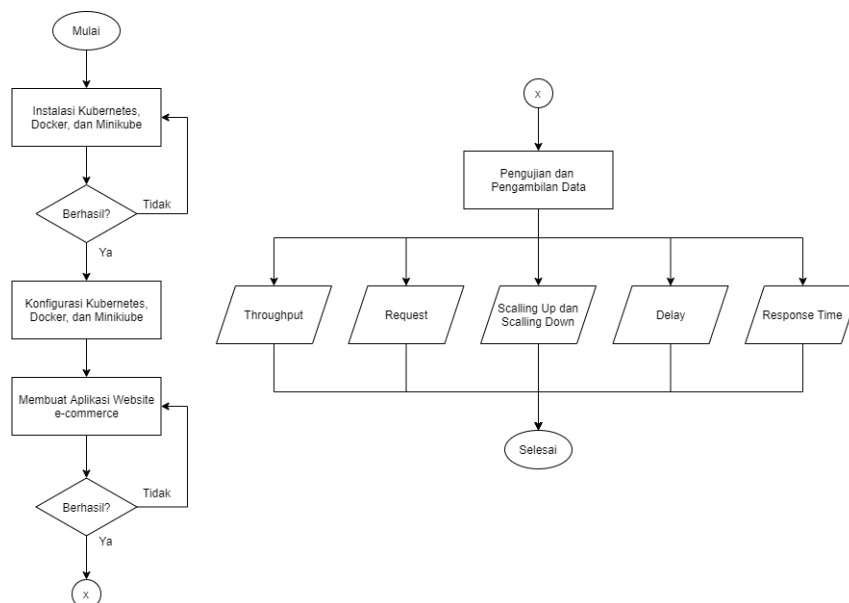
3.2 Tahapan Perancangan

Proses perancangan vertical autoscaling web aplikasi menggunakan Kubernetes ini dilakukan dengan metode eksperimental. Proses tahapan perancangan adalah sebagai berikut.

1. Langkah awal dalam merancang vertical autoscaling web aplikasi menggunakan Kubernetes adalah dengan membuat sebuah cluster Kubernetes yang berisikan tiga buah node.
2. Mengumpulkan informasi terkait perancangan vertical autoscaling dan melakukan instalasi maupun konfigurasi yang akan digunakan.
3. Melakukan pembuatan website e-commerce dengan menggunakan reaction commerce sebagai front-end system dan mongoDB sebagai back-end system.
4. Merealisasikan model konfigurasi dan perancangan.

3.3 Perancangan

Pada proyek akhir ini akan dirancang vertical autoscaling web aplikasi menggunakan Kubernetes. Proses perancangan vertical autoscaling web aplikasi menggunakan Kubernetes dapat dilihat pada gambar 3.2 dibawah ini.



Gambar 3. 2 Diagram Alir Perancangan

BAB IV

BENTUK KELUARAN YANG DIHARAPKAN

4.1 Keluaran yang Diharapkan

Hasil keluaran yang diharapkan dalam perancangan *vertical autoscaling* web aplikasi menggunakan Kubernetes adalah sebagai berikut.

- a) Scalling Up : < 50 s
- b) Scalling Down : < 130 s
- c) Throughput : ± 3 MB/s
- d) Request : 500 user
- e) Delay : 200 ms – 500 ms
- f) Response Time : < 40 s

4.2 Jadwal Pelaksanaan

Adapun jadwal pengerjaan Proyek akhir bisa dilihat pada tabel sebagai berikut :

Tabel 4. 1 Jadwal Pelaksanaan

Judul Kegiatan	Waktu							
	Nov	Des	Jan	Feb	Mar	Apr	Mei	Jun
Studi Literatur								
Perancangan dan Simulasi								
Pabrikasi								
Pengukuran								
Pengujian								
Analisa								
Pembuatan Laporan								

DAFTAR PUSTAKA

- [1] J. B. R. B. Andrzej Goscinski, "CLOUD COMPUTING," *Principles and Paradigms*.
- [2] L. L. C. P. Q. D. L. W. W. Z. Qi Zhang, "A Comparative Study of Containers and Virtual Machines in Big Data Environment," 2018.
- [3] A. R. R. D. K. Rajdeep Dua, "Virtualization vs Containerization to Support PaaS," 2014.
- [4] "About Docker," Docker documentation, [Online]. Available: <https://docs.docker.com/engine/misc/>.
- [5] "Why systemd Linux containers make sense," [Online]. Available: <https://searchservervirtualization.techtarget.com/tip/Why-systemd-Linux-containers-make-sense>.
- [6] "Jails," [Online]. Available: <https://www.freebsd.org/doc/en/books/handbook/jails.html>.
- [7] "Understand the architecture," [Online]. Available: <https://docs.docker.com/get-started/overview/>.
- [8] "Reaction Commerce: does the world really need another e-commerce platform?," [Online]. Available: <https://weknowinc.com/blog/reaction-commerce-does-world-really-need-another-e-commerce-platform>.
- [9] G. P. R. G. A. O. Cornelia Györödi, "A Comparative Study: MongoDB vs. MySQL," 2015.
- [10] S. I. J. G. Mazedur Rahman, "International Symposium on Service Oriented System Engineering," *Load Balancer as a Service in Cloud Computing*, 2014.
- [11] R. A. F. B. R. F. I. N. J. A. Arief Indriarto Haris, "Pengembangan Container Orchestration Berbasis Kubernetes Di Lembaga Penerbangan dan Antariksa Nasional

(LAPAN)," 2020.

- [12] "Apa itu Kubernetes?," Kubernetes, 2020. [Online]. Available:
<https://kubernetes.io/id/docs/concepts/overview/what-is-kubernetes/>.
- [13] "Komponen-Komponen Kubernetes," Kubernetes, 2020. [Online]. Available:
<https://kubernetes.io/id/docs/concepts/overview/components/>.
- [14] "Instalasi Kubernetes dengan Minikube," Kubernetes, [Online]. Available:
<https://kubernetes.io/id/docs/setup/learning-environment/minikube/>.
- [15] M. Fihri, "Implementasi & Analisis Performasi Layanan Web Pada Platform Berbasis Docker," 2019.
- [16] G. Rattihalli, "Exploring Potential for Non-Disruptive Vertical Auto Scaling and Resource Estimation in Kubernetes," 2019.
- [17] M. W. I. Santosa, "Implementasi Load Balancing Server Basis Data Pada Virtualisasi Berbasis Kontainer," 2018.
- [18] Y. T. Sumbogo, "Implementasi Failover Dan Autoscaling Kontainer Web Server Nginx Pada Docker Menggunakan Kubernetes," 2018.
- [19] V. K. Wyn Van Devanter, "Overview of Container Management," 2017.
- [20] "Vertical Pod Autoscaling," Google Cloud, [Online]. Available:
<https://cloud.google.com/kubernetes-engine/docs/concepts/verticalpodautoscaler>.



UNIVERSITAS TELKOM
FAKULTAS ILMU TERAPAN
KARTU KONSULTASI
SEMINAR PROPOSAL PROYEK TINGKAT

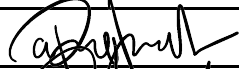
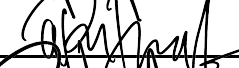
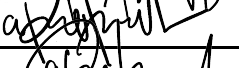
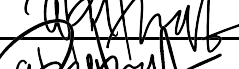
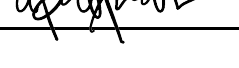
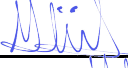
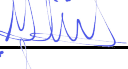
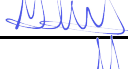
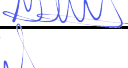

NAMA / PRODI : Aulia Rizqi Putra / D3 Teknologi Telekomunikasi NIM : 6705180006

JUDUL PROYEK TINGKAT :

Analisis dan Implementasi Vertical Autoscaling Web Aplikasi menggunakan Kubernetes

CALON PEMBIMBING : I. Rohmat Tulloh, S.T., M.T.

II. Muhammad Iqbal, S.T., M.T.

NO	TANGGAL	CATATAN HASIL KONSULTASI	TANDA TANGAN CALON PEMBIMBING I
1	22/01/2021	BAB 1 (SELESAI)	
2	22/01/2021	BAB 2 (SELESAI)	
3	22/01/2021	BAB 3 (SELESAI)	
4	22/01/2021	BAB 4 (SELESAI)	
5	22/01/2021	FINALISASI PROPOSAL	
6			
7			
8			
9			
10			
NO	TANGGAL	CATATAN HASIL KONSULTASI	TANDA TANGAN CALON PEMBIMBING II
1	19/01/2021	BAB 1 (SELESAI)	
2	19/01/2021	BAB 2 (SELESAI)	
3	19/01/2021	BAB 3 (SELESAI)	
4	19/01/2021	BAB 4 (SELESAI)	
5	19/01/2021	FINALISASI PROPOSAL	
6			
7			
8			
9			
10			