BAB II

LANDASAN TEORI

2.1. Penelitian Terdahulu

Penelitian pertama yang dilakukan oleh Angga Friyanto dengan judul '*High Availability Aspects of SDN-IP Reactive Routing*' tentang *SDN* (*Software-Defined Network*) adalah paradigma baru dalam dunia jaringan, dimana *control plane* dipisahkan dari *data plane*. Dalam beberapa tahun terakhir, penyebaran jaringan *SDN* menghadapi kendala karena terisolasi dari jaringan IP yang ada saat ini dan harus memperhatikan aspek redundansi. Tujuan dari penelitian ini adalah untuk menganalisis aspek *High Availability* dalam *SDN-IP Reactive Routing*, yang dilaksanakan dengan mengintegrasikan beberapa *onos controllers* dalam satu sistem *cluster onos*. Pengujian dilakukan terhadap aspek *BGP speakers, ONOS controller,* dan link antar komponen [2]. Persamaan antara penelitian yang dilakukan oleh Angga Friyanto dan penelitian yang peneliti lakukan adalah bahwa keduanya fokus pada implementasi *High Availability* dan menghadapi isu terkait redundansi. Sedangkan perbedaannya terletak pada objek penelitian, Angga Friyanto melakukannya pada *SDN-IP Reactive Routing,* sementara peneliti berfokus pada *Autoscaling server* pada aplikasi *microservice ImageServer*.

Peneliti kedua yang dilakukan oleh Tioreza Febrian dengan judul 'Analisis Performansi Web *Services* Pada Arsitektur *Microservice* Pada Domain Kasus *Learning Management System* di SESKOAU' bertujuan untuk menganalisis kinerja layanan web dalam arsitektur *microservices*. Hal ini dilakukan untuk menentukan

jenis arsitektur apa yang paling efektif sehingga diharapkan dapat meningkatkan performansi dari sistem *LMS* di SESKOAU. Dalam penelitian ini, pendekatan *Domain Driven Design* digunakan sebagai metode dalam memecahkan setiap layanan yang dikembangkan berdasarkan konteksnya, dan *Kubernetes* digunakan sebagai *platform deployment* berbasis *container* dalam menjalankan sistem pada *server* pengujian [3]. Persamaan antara penelitian yang Anda lakukan dengan penelitian yang dilakukan oleh Tioreza Febrian adalah bahwa keduanya berfokus pada eksplorasi dan analisis arsitektur *microservices*, dan memiliki tujuan yang sama dalam meningkatkan kinerja sistem melalui penggunaan arsitektur *microservices*. Namun, perbedaannya terletak pada fokus spesifik tujuan penelitian. Tioreza Febrian secara khusus menargetkan untuk menganalisis performansi layanan web dalam konteks *Learning Management System*, sedangkan penelitian yang Anda lakukan lebih mengarah pada implementasi *autoscaling server* yang berkaitan erat dengan skalabilitas dan penanganan beban tinggi.

2.2. Implementasi

Berdasarkan Kamus Besar Bahasa Indonesia, implementasi merujuk pada tahap pelaksanaan dan penerapan, yang bertujuan untuk merealisasikan apa yang telah disepakati sebelumnya. Implementasi merupakan proses yang menjamin suatu kebijakan dapat diterapkan dan tujuan dari kebijakan tersebut dapat dicapai. Tujuan dari proses implementasi suatu sistem adalah untuk merampungkan desain sistem yang telah disetujui, melakukan pengujian dan dokumentasi program dan prosedur sistem yang diperlukan, memastikan bahwa personil yang terlibat mampu

mengoperasikan sistem baru dan memastikan bahwa perpindahan dari sistem lama ke sistem baru dapat dilakukan dengan baik dan benar [4].

2.3. *Autoscaling*

Autoscaling adalah kapabilitas sebuah sistem untuk menyesuaikan sumber daya yang dimilikinya, seperti mengurangi atau menambah jumlah proses berdasarkan kebutuhan sistem, tanpa menginterupsi proses yang sedang berlangsung. Metode *autoscaling* di mana sistem secara periodik memantau *cluster server*. Ketika sistem mendeteksi bahwa layanan pada suatu *server* tidak mampu memenuhi permintaan pengguna, sistem tersebut akan menambahkan layanan itu ke *server* lain yang tersedia [5].

2.4. *Microservice*

Arsitektur *microservice* adalah metode pengembangan aplikasi yang dilakukan dalam bentuk layanan web yang lebih kecil dan saling berinteraksi satu dengan lainnya. *Microservice* menyediakan alternatif arsitektur yang lebih *scalable* dan fleksibel. Dengan kata lain, arsitektur *microservice* membagi aplikasi menjadi beberapa layanan berdasarkan fungsi yang spesifik. Setiap layanan ini dirancang untuk beroperasi secara independent dan dapat menggunakan teknologi yang berbeda sesuai dengan kebutuhannya [6].

2.5. *Distributed*

Sistem terdistribusi adalah sistem yang melakukan proses informasi di sejumlah komputer, bukan hanya terbatas pada satu mesin saja. Saat ini, masih ada banyak sistem berskala besar yang menggunakan sistem tersentralisasi, di mana semua berjalan dalam satu *mainframe* dengan terminal-terminal yang terhubung ke

komputer tersebut. Sistem semacam itu seringkali memiliki kekurangan, karena terminal-terminal hanya memiliki sedikit kapabilitas untuk memproses data, dan Sebagian besar tergantung pada komputer pusat [7].

2.6. *Docker*

Docker adalah platform terbuka yang dirancang untuk para pengembang dan administrator sistem untuk membangun, mengirim, dan menjalankan aplikasi terdistribusi. Secara praktis, *docker* dapat diartikan sebagai metode untuk menempatkan layanan dalam lingkungan yang terisolasi, yang disebut *container*, yang memungkinkan layanan tersebut dapat dikemas menjadi satu unit bersama dengan semua *library* dan perangkat lunak lainnya yang diperlukan [8].

2.6.1. *Docker images*

Docker images adalah template yang hanya bisa dibaca (*read-only*) yang digunakan untuk menjalankan *containers*. Sebuah *image* bisa mencakup sistem operasi serta beberapa aplikasi dengan *image*s lainnya, dimana *image* teratas disebut sebagai *parent image* dan *image* paling dasar disebut sebagai *base image*. Sebagai contoh, sebuah *image* mungkin bersisi sistem operasi *ubuntu* dengan *nginx* dan aplikasi web yang telah terinstal. *Image* ini kemudian digunakan untuk menjalankan *container*

2.6.2. *Docker container*

Docker container adalah direktori yang berisi segala yang diperlukan agar suatu aplikasi dapat berjalan. Setiap *container* berjalan dari *docker image* yang telah ditentukan, dan juga berjalan dalam lingkungan dan *platform* aplikasi yang

terisolasi dan aman, sehingga tidak akan berbenturan dengan aplikasi lain dalam *host* yang sama.

2.6.3. *Docker registry*

Docker registry adalah *repository*, bisa publik atau pribadi, yang menyimpan ribuan *Docker images*. *Docker hub* adalah contoh dari *Docker registry* publik. Pengguna dapat menggunakan perintah *'push'* melalui *Docker client* untuk menyimpan dan membagikan *Docker images* ke *Docker registry*. Kemudian, pengguna lain dapat menggunakan perintah *'pull'* untuk mengunduh dan menjalankan *Docker images* tersebut secara langsung.

2.7. *Kubernetes*

Kubernetes adalah sistem *open source* yang otomatis mengelola penyebaran penskalaan, dan manajemen *container*. Awalnya dibuat oleh *google*, kini *Kubernetes* dikelola oleh *Cloud Native Computing Foundation*. *Kubernetes* berperan sebagai pengelola bagi *container-container* dan menyediakan *platform* yang mendukung fungsi-fungsi tersebut [9]. Berikut ini adalah beberapa objek dasar yang dimiliki oleh *kubernetes* :

1. *Pod* adalah unit terkecil yang dapat dikelola dan di *schedule* dalam *Kubernetes*. *Pod* bisa berisi satu atau lebih *container* yang berbagi *storage*, jaringan IP, dan bagaimana menjalankan *container*.
2. *Service* adalah abstraksi yang mendefinisikan set logis dari *Pod* dan kebijakan akses mereka. Ini memungkinkan komunikasi antara *Pod*, dan juga dapat mengekspos aplikasi ke luar *cluster*.

3. *Namespace* digunakan untuk memisahkan sumber daya fisik antara lingkungan, seperti lingkungan pengembangan, pengujian, dan produksi dalam *cluster* yang sama.
4. *Deployment* mengotomatisasi pembaruan untuk *Pod* dan *ReplicaSet*.

2.8. *WebP*

WebP adalah format gambar yang dibuat oleh *Google*, yang menawarkan kompresi *lossy* dan *lossless* yang unggul. Ukuran gambar *WebP* dalam format *lossless* 26% lebih kecil dibandingkan dengan *PNG*. Di sisi lain, gambar *WebP* dalam format *lossy* 25%-34% lebih kecil dibandingkan dengan gambar *JPEG* dengan kualitas gambar yang setara [10]. Berikut adalah beberapa fitur utama dari *WebP* berdasarkan situs resmi *Google Developer*:

1. Kompresi *Lossless* dan *Lossy*: *WebP* mendukung kompresi baik *lossless* (tanpa kehilangan data) maupun *lossy* (dengan kehilangan data), memungkinkan fleksibilitas dalam penyeimbangan antara ukuran file dan kualitas gambar.
2. Transparansi: Dikenal juga sebagai alpha channel, *WebP* mendukung transparansi, yang sangat penting untuk gambar web.
3. Metadata: *WebP* dapat menyertakan metadata dalam gambar, seperti informasi *EXIF*, *geotagging*, dan lainnya
4. Kemampuan Warna: *WebP* mendukung 24-bit *RGB* warna dengan 8-bit *alpha channel*, yang memungkinkan warna yang kaya dan transparansi.

2.9. *Golang*

Golang, juga dikenal sebagai Go, adalah bahasa pemrograman yang dibuat dan dikembangkan oleh tim insinyur Google pada tahun 2009. Pada awalnya,

bahasa ini hanya digunakan untuk keperluan internal Google. Namun, kemudian bahasa ini dirilis ke publik sebagai proyek *open source*, yang berarti siapa saja dapat berkontribusi untuk pengembangannya. *Golang* memiliki berbagai penggunaan, termasuk penggunaan dalam pembuatan *backend stack*, pengembangan aplikasi *e-commerce*, dan pengembangan *cloud native* [11].

2.10. *Python*

Python adalah bahasa pemrograman yang banyak digunakan oleh berbagai perusahaan besar dan pengembang untuk membuat berbagai jenis aplikasi, termasuk yang berbasis *desktop*, web, dan *mobile*. *Python* diciptakan oleh Guido van Rossum di Belanda pada tahun 1990, dengan nama yang diambil dari acara televisi favoritnya, "Monty Python's Flying Circus." Van Rossum awalnya mengembangkan *Python* sebagai hobi, tetapi seiring berjalannya waktu, *Python* telah menjadi bahasa pemrograman yang sangat populer dalam industri dan pendidikan, berkat sintaksnya yang sederhana dan intuitif, serta pustaka yang luas [12].

2.11. *Postgresql*

Postgresql atau juga dikenal sebagai *Postgres*, adalah Sistem Manajemen *Database Relasional Objek* (ORDBMS) yang bersifat *open source*. *PostgreSQL* sangat menonjol dalam hal ekstensibilitas, standar kepatuhan, dan inovasi. Ini bersaing dengan vendor basis data relasional besar seperti *Oracle, MySQL, SQL Server*, dan lainnya. Digunakan di berbagai sektor, termasuk pemerintah dan publik serta sektor swasta. *PostgreSQL* adalah DBMS lintas *platform*, yang berarti dapat berjalan pada berbagai sistem operasi [13].

PostgreSQL adalah Sistem Manajemen *Database* yang bersifat *open source* dan menawarkan dukungan luas untuk bahasa *SQL*. Selain itu, *PostgreSQL* juga menyediakan sejumlah fitur modern, seperti:

1. *Complex queries*
2. *Foreign key*
3. *Triggers*
4. *Views*
5. *Transaction integrity*
6. *Multi version Concurrency Control*

2.12. *Redis*

Redis adalah perangkat lunak *open source* yang digunakan untuk penyimpanan data dalam struktur memori dan dapat berfungsi sebagai *database*, sistem *caching*, atau pesan *broker*. Disebut juga sebagai "*leather man of database*", *Redis* dikenal dengan desainnya yang sederhana namun fleksibel, membuatnya menjadi pilihan yang efektif untuk mengatasi berbagai permintaan pemrosesan data. *Redis* dirancang untuk membangun basis pengetahuan yang mendalam mengenai konteks dan teori teknologi. Beroperasi pada *RAM*, *Redis* menawarkan kecepatan yang lebih tinggi dalam pengelolaan data [14].