BAB II

TINJAUAN PUSTAKA DAN LANDASAN TEORI

2.1 TINJAUAN PUSTAKA

Penelitian Vayghan, Saied, Toeroe, dan Khendek (2018), penulis melakukan pembangunan aplikasi berbasis *microservices* pada *private cloud* menggunakan *cluster* Kubernetes, selanjutnya dilakukan evaluasi dan eksperimen pengujian berdasarkan parameter *reaction time, repair time, recovery time,* dan *outage time*. Pada pengujian tersebut didasarkan pada dua skenario pengujian yaitu skenario pengujian *pod failure* dan skenario pengujian *node failure.* Pengujian *pod failure* didasarkan pada kegagalan aplikasi yang terjadi pada pod. Selanjutnya pada skenario pengujian *node failure*, didasarkan pada kegagalan aplikasi yang terjadi pada *pod* di masing-masing node. *Dari* kedua pengujian tersebut dihasilkan adanya perbedaan data hasil pengujian yang mengindikasikan bahwa syarat *high availability server* tidak terpenuhi secara otomatis dengan menerapkan aplikasi berbasis *microservices* menggunakan Kubernetes [3]. Dari penelitian tersebut, penulis mendapatkan inspirasi menggunakan Kubernetes pada *computer cluster* Raspberry Pi 4 untuk dapat melakukan uji performa aplikasi berbasis *microservices.*

Penelitian Ferreira dan Sinnott (2019), penulis melakukan uji coba performa pada *container* menggunakan Docker yang berjalan pada Kubernetes *cluster*. Uji coba dilakukan pada *Cloud Platform* penyedia teknologi Kubernetes seperti Azure Kubernetes Service (AKS), Amazon Elastic Container Service for Kubernetes (EKS), dan Google Kubernetes Engine (GKE). Pengujian tersebut didasarkan pada *deployment* Kubernetes *cluster* secara manual pada *cloud platform* yang sama, dan pada *managed* Kubernetes *cluster* yang disediakan oleh *cloud platform.* Metode pengujian yang dipakai pada penelitian tersebut yaitu *Cloud Evaluation Experiment Methodology*, dengan parameter pengujian *computing performance*, *memory performance*, *disk performance,* dan *network performance.* Dari hasil pengujian tersebut, dapat dengan

mudah dilakukan pengujian melalui *container* menggunakan Docker yang berjalan pada cluster, maka dari itu pada penelitian ini penulis akan melakukan pengujian pada Kubernetes *cluster* menggunakan Docker sebagai *container* untuk dapat menjalankan aplikasi berbasis *microservices* [7].

Penelitian dengan judul "*Performance analysis of single board computer clusters*", di penelitian tersebut dilakukan pengembangan *computer cluster* menggunakan tiga model *single board computer* yaitu Raspberry Pi 3 Model B, Raspberry Pi 3 Model B+, dan Odroid C2. Dari pengembangan *computer cluster* tersebut dilakukan pengujian dengan didasarkan pada *scaling, power efficiency,* dan *value for mone*y. Berdasarkan hasil pengujian dari ketiga model *single board computer* tersebut, Raspberry Pi 3 Model B+ mengungguli penghematan penggunaan biaya pada proses pengujian secara terpisah. Kemudian pada pengujian secara *clustering, cluster* Raspberry Pi Model 3B+ masih mengungguli efisiensi penggunaan biaya, namun dengan efisiensi penggunaan listrik yang kurang dari *cluster* Odroid C2. Dari hasil pengujian tersebut dapat disimpulkan bahwa penggunaan *single board computer* Raspberry Pi pada *computer clustering* memiliki keunggulan dalam penghematan penggunaan biaya [6]. Pada penelitian ini, penulis akan melakukan pengembangan *computer clustering* dengan menggunakan Raspberry Pi 4 untuk melakukan pengujian performa pada aplikasi berbasis *microservices* dengan implementasi *container orchestration* Kubernetes.

Penelitian Ginting, Ikhwan, dan Ammar (2021), pada penelitian tersebut dilakukan pengembangan *high availability server* menggunakan Google Kubernetes Engine pada *platform* Google Cloud dan terdapat sebanyak tiga node yang berjalan pada *cluster*. Dilakukan konfigurasi aplikasi web menggunakan apache *web server* dan database MySQL. Kemudian dilakukan uji coba performa pada *web server* dengan pengukuran nilai *availability* dan parameter pengujian QOS yaitu pada pengukuran nilai *CPU usage, throughput, delay,* dan *packet loss.* Skenario pengujian yang dilakukan yaitu menggunakan jumlah koneksi bertahap dari 200 jumlah koneksi sampai dengan 5000 jumlah koneksi. Dari data hasil pengujian menunjukan bahwa

sistem dapat berjalan optimal sampai dengan pengujian 5000 jumlah koneksi [8]. Dari penelitian sebelumnya tersebut, penulis akan melakukan penelitian yang sama dengan menggunakan Kubernetes sebagai *container orchestration* namun pada *computer cluster* yang dibangun menggunakan Raspberry Pi 4 sebagai efisiensi penggunaan biaya infrastruktur.

Penelitian Hadiwandra dan Candra (2021), penelitian tersebut melakukan pengujian *web server* pada *computer cluster* Raspberry Pi menggunakan Docker swarm sebagai *container orchestration* yaitu pada pengembangan *high availability server*. Dari hasil pengujian tersebut didapatkan bahwa Raspberry Pi dapat digunakan untuk membuat *computer clustering* yang mendukung *high availability server* dengan menggunakan virtualisasi Docker yang dapat berjalan dengan baik. Berdasarkan hasil pengujian, Raspberry Pi dapat menangani *request* dengan tingkat kesalahan 0% dan kecepatan transfer data (*Throughput*) mencapai 161,812,298 *requests*/*sec* [9]. Kemudian pada penelitian ini penulis akan melakukan pengembangan *computer clustering* menggunakan Raspberry Pi dan virtualisasi menggunakan Docker *container* yang sama seperti pada penelitian sebelumnya, namun pengujian performa dilakukan pada aplikasi berbasis arsitektur *microservices* dan menggunakan Kubernetes sebagai *container orchestration*, berbeda dengan penelitian sebelumnya yang menggunakan Docker swarm.

Tabel 2. 1 Perbandingan Penelitian

No	Judul Penelitian	Masalah	Metode	Hasil
1.	*Deploying Microservice Based Applications with Kubernetes: Experiments and Lessons Learned* [3]	Kelemahan arsitektur monolitik pada proses pembangunan aplikasi.	Metode yang digunakan adalah metode eksperimen, di mana dilakukan dua skenario pada proses eksperimen pengujian.	Dari hasil pengujian terdapat perbedaan data yang mengindikasikan bahwa syarat *high availability server* tidak terpenuhi secara langsung dengan menerapkan aplikasi berbasis *microservices* menggunakan Kubernetes.
2.	*A Performance Evaluation of Containers Running on Managed Kubernetes Services* [7]	Performa *running* container pada beberapa *cloud platform* yang belum banyak diketahui.	Metode yang digunakan pada proses pengujian yaitu *Cloud Evaluation Experiment Methodology*.	Pengujian yang dilakukan pada *running container* Kubernetes yang berjalan di beberapa *cloud platform* menghasilkan bahwa Google Kubernetes Engine memiliki kinerja jaringan yang terbaik, kemudian kontainerisasi aplikasi pada *platform* AWS merupakan pilihan yang terbaik. Disamping itu Docker digunakan sebagai *platform* kontainerisasi pada Kubernetes *cluster*.

No	Judul Penelitian	Masalah	Metode	Hasil
3.	*Performance analysis of single board computer clusters* [6]	Efisiensi penggunaan listrik dan penggunaan biaya pada model *single board computer* untuk *computer cluster* yang belum banyak diketahui.	Metode yang digunakan pada pengujian performa *cluster* yaitu *high performance linkpack benchmarking.*	Dari hasil pengujian tersebut dapat disimpulkan bahwa penggunaan *single board computer* Raspberry Pi pada *computer clustering* memiliki keunggulan dalam penghematan penggunaan biaya.
4.	Analisis Performa High Availability Web *Server* Pada Cluster GKE (Google Kubernetes Engine) Menggunakan Infrastruktur Google Cloud Platform [8]	Dibutuhkannya ketersediaan *resource* untuk web *server* dikarenakan pengguna internet yang semakin banyak.	Metode yang digunakan yaitu metode eksperimental dan prinsip yang digunakan pada metode tersebut adalah *pre-experimental* dengan *one shot case study*.	*High availability server* menggunakan *container orchestration* Kubernetes dan teknologi *Google Kubernetes Engine* pada *platform* Google Cloud. Dari pengujian performa, sistem dapat berjalan secara optimal dengan pengujian banyaknya koneksi dimulai dari pengujian 200 koneksi hingga 5000 koneksi.

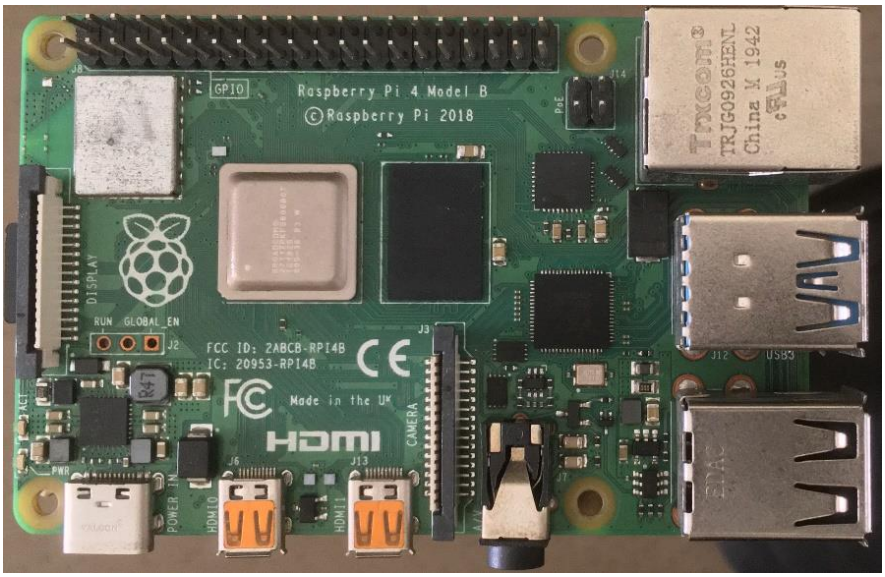
No	Judul Penelitian	Masalah	Metode	Hasil
5.	*High Availability Server Using Raspberry Pi 4 Cluster and Docker Swarm* [9]	Dibutuhkannya infrastruktur berperforma tinggi demi menunjang berjalannya aplikasi secara lancar.	Pengujian performa *computer cluster* menggunakan Raspberry Pi 4 dengan berdasarkan parameter QOS.	Raspberry Pi dapat digunakan untuk membuat *computer clustering* yang mendukung *high availability server* dengan menggunakan *virtualisasi* Docker yang dapat berjalan dengan baik. Berdasarkan hasil pengujian, Raspberry Pi dapat menangani request dengan tingkat kesalahan 0% dan kecepatan transfer data (Throughput) mencapai 161,812,298 *requests/sec*.

2.2 LANDASAN TEORI

2.2.1 Raspberry Pi

Raspberry Pi merupakan salah satu *single board computer* paling populer di dunia yang dikembangkan oleh Raspberry Pi Foundation. Tujuan utama dari proyek pengembangan Raspberry Pi ini adalah untuk memberikan Pendidikan dasar ilmu komputer di sekolah. Papan pertama hadir dan rilis pada bulan februari tahun 2012. Raspberry Pi memiliki *processor* yang lebih *powerfull* terutama pada model Raspberry Pi terbaru yaitu Raspberry Pi 4 Model B dengan spesifikasi *processor* 1.5GHz 64-bit quad-core ARM Cortex-A72. Disamping itu Raspberry Pi memiliki spesifikasi RAM yang terdiri dari beberapa pilihan seperti 4GB dan 8GB [10].

Pada Raspberry Pi 4 terdapat *gigabit internet support* yang memudahkan sistem mendapatkan jaringan dengan kecepatan yang tinggi. Raspberry Pi memungkinkan untuk dipasang sistem operasi dengan dukungan teknologi arm yaitu Raspbian OS, Windows 10 IOT, Arch Linux, dan Ubuntu. Papan Raspberry Pi 4 Model B dapat dilihat pada Gambar 2. 1 berikut [10].



Gambar 2. 1 papan Raspberry Pi 4 Model B

2.2.2 Computer Cluster

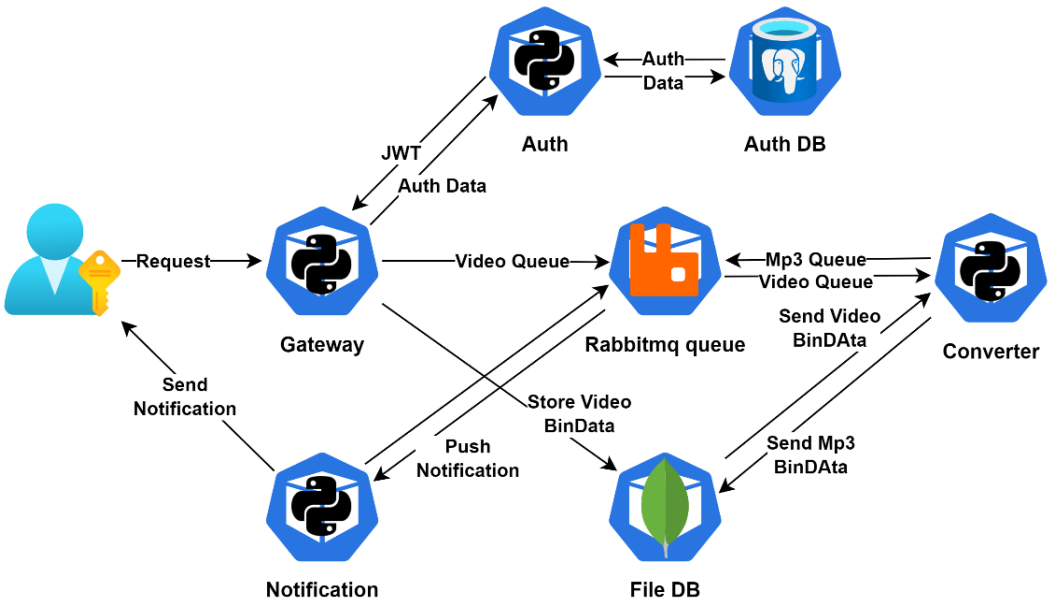
Computer cluster merupakan proses pendistribusian sumber daya komputasi seperti CPU dan *memory* dengan pendekatan modern yang berbentuk arsitektur melalui kumpulan *node* berbentuk perangkat keras yang serupa dan berkomunikasi satu sama lain melalui jaringan khusus. Sistem *computer cluster* mempunyai kelebihan dengan mampu meningkatkan sumber daya tambahan yaitu dengan cara menambahkan *node* pada *cluster*. Umumnya setiap *computer cluster* mencakup lapisan hirarki sejumlah enam lapisan yaitu aplikasi, *middleware,* pendistribusian model program, *internetworking,* komputasi, sistem operasi, dan *compilers* [11].

Computer cluster membutuhkan biaya yang cukup besar diawal, demikian juga sebuah *cluster* yang dapat tumbuh seiring dengan meningkatnya kebutuhan sumber daya. Konfigurasi *server* yang mengandalkan banyak perangkat keras dapat diimplementasikan dengan biaya yang lebih murah, yaitu dengan penggunaan Raspberry Pi yang merupakan sumber daya paling ideal untuk dapat merancang sebuah pengembangan *computer cluster* [11].

2.2.3 Microservices architecture

Arsitektur *microservices* hadir pada permasalahan penyebaran aplikasi dengan arsitektur *monolithic* yang menyulitkan proses pembaharuan aplikasi dan kurang teraturnya sebuah arsitektur penyebaran aplikasi. Pada arsitektur *monolithic*, pembaharuan sebuah aplikasi pada menyebabkan penerapan ulang sebuah sistem dan proses penyebaran ulang seluruh sistem pada infrastruktur aplikasi. Untuk dapat menangani permasalahan keterbatasan pada arsitektur *monolithic*, arsitektur *microservices* hadir dengan memisahkan aplikasi yang cukup rumit menjadi sebuah komponen-komponen ringan yang digabungkan. Setiap komponen secara independen melakukan tugas secara ringan dan dapat diperbaharui tanpa melibatkan komponen lain [12].

Arsitektur *microservices* sangat mendukung peningkatan skala masing-masing komponen secara independent. Maka dari itu, arsitektur *microservices* dapat meningkatkan elastisitas dan skalabilitas pada pengembangan sebuah aplikasi. Virtualisasi berbasis *container* digunakan pada arsitektur *microservices* karena dapat berjalan secara ringan dan dengan tujuan untuk dapat mempercepat penerapan arsitektur *microservices* [12]. Berikut merupakan Gambar 2. 2 contoh arsitektur *microservices* pada pengembangan aplikasi konversi video ke MP3.



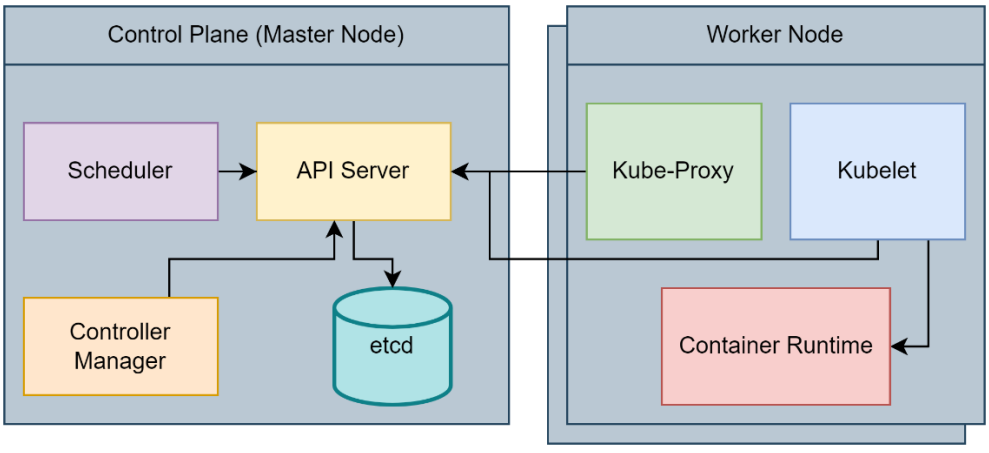
Gambar 2. 2 Contoh arsitektur *microservices*

2.2.4 Kubernetes

Kubernetes merupakan salah satu *open-source container orchestration* paling populer yang diciptakan oleh google. *Platform* ini menyediakan pengotomatisasian penyebaran aplikasi, fleksibilitas, skalabilitas, dan pengoperasian kontainerisasi aplikasi melalui *cluster*. Kubernetes sangat banyak digunakan untuk manajemen *container orchestration* pada kontainerisasi aplikasi berbasis *cluster*, seperti manajemen Docker *container*, manajemen otomatisasi penyebaran aplikasi dan

penskalaan sebuah *container*. Kubernetes juga menyediakan otomatisasi memulai ulang kembali *container* yang gagal dan melakukan penjadwalan ulang *container* ketika *host* mati. Ini sangat meningkatkan *availability* sebuah aplikasi dan memudahkan *developer* dalam melakukan penskalaan, otomatisasi penyebaran aplikasi, dan manajemen aplikasi berbentuk *container* [13].

Arsitektur Kubernetes memiliki beberapa komponen seperti *API Server* yang mana Kubernetes menyediakan fungsionalitasnya, kemudian terdapat komponen *controller manager* yang merupakan komponen utama untuk memantau dan mengubah status *cluster* melalui *API server*. *Scheduler* merupakan salah satu komponen Kubernetes pada bidang kontrol yang bertanggung jawab atas penjadwalan *pod* di beberapa *node*. Selain itu terdapat komponen basis data yaitu etcd yang merupakan *key-value based database* yang memelihara semua konfigurasi informasi mengenai konfigurasi pada Kubernetes *cluster*. *User* dapat berinteraksi dengan *API server* pada *node master* menggunakan perintah kubectl [14]. Dapat dilihat arsitektur Kubernetes pada Gambar 2. 3 berikut.



Gambar 2. 3 Arsitektur Kubernetes

2.2.5 Container

Teknologi *container* menjadi sebuah teknologi yang sangat populer dan banyak digunakan pada banyak aplikasi berbasis *cloud*. *Container* telah berhasil digunakan

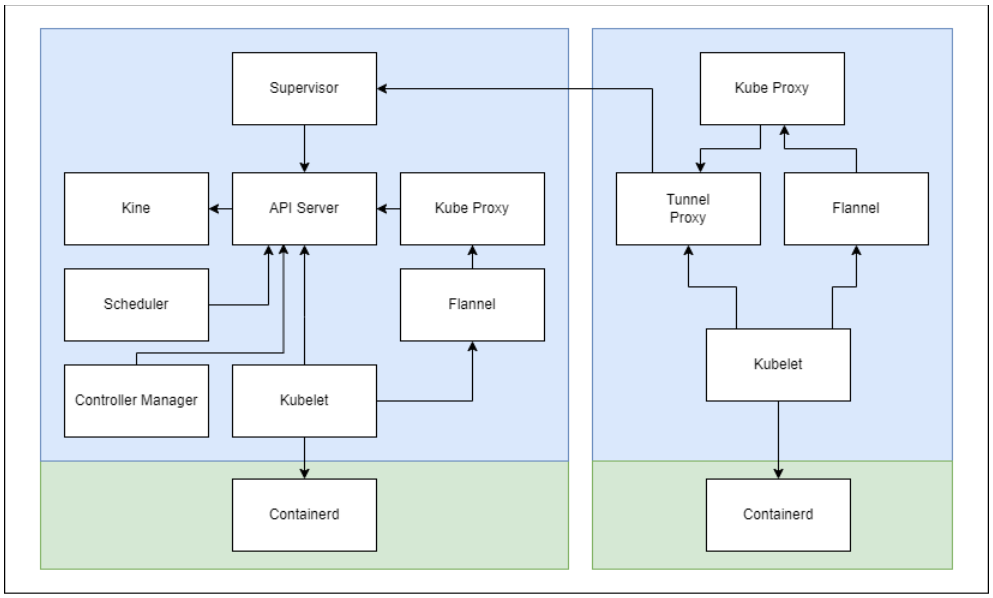
pada berbagai aplikasi, *container* dapat berjalan pada sistem operasi yang sama dan memungkinkan *server* dapat lebih efisien dan memungkinkan proses penyebaran sebuah aplikasi dapat berjalan lebih cepat. *Container* dapat bekerja lebih baik untuk dapat meningkatkan kehandalan sebuah perangkat lunak pada saat terjadi peralihan dari satu IT *environment* ke *environment* lain yang mana *container* dapat mengurangi kompleksitas *platform* aplikasi, *dependency*, dan strukturnya. Singkatnya, semua melibatkan pembangunan *container* apa pun pada sebuah *running environment*, aplikasi, *library,* dan file konfigurasi yang diperlukan untuk dapat menjalankannya [15].

2.2.6 K3s

K3s merupakan distribusi Kubernetes yang sangat ringan yang ditawarkan oleh perusahaan teknologi terkemuka yaitu Rancher. Selain itu distribusi Kubernetes ini difokuskan pada pengembangan aplikasi dengan spesifikasi yang ringan dan tidak kompleks. K3s juga sangat kompatibel dibandingkan dengan distribusi Kubernetes K8s, yaitu memiliki semua komponen dasar secara bawaan, dan menargetkan sistem yang cepat, sederhana, dan efisien untuk menyediakan *high availability cluster* dan toleran terhadap *cluster* dengan satu *node*. Penerapan tersebut berlangsung melalui satu *binary* yang tunggal dan kecil termasuk dependensi [16].

K3s mengganti ETCD sebagai *data store* pada Kubernetes dengan *data store* lainnya yaitu sqlite3, juga terdapat beberapa komponen penyimpanan pada K3s yang dihapus dari distribusi Kubernetes K8s untuk menjaga ukurannya tetap kecil. *Master node* pada K3s disebut *server* dan *worker node* disebut *agent* yang mana merangkum semua komponen dalam satu proses tunggal. K3s dapat diinstal melalui beberapa konfigurasi seperti menggunakan Ansible. Untuk dapat mencapai *high availability* pada K3s dapat dengan mudah menambahkan *worker node* pada *cluster* dengan beberapa perintah [16].

K3s dapat berjalan secara ringan pada Raspberry Pi dengan spesifikasi *hardware* yang cukup terbatas, oleh sebab itu digunakan K3s pada pengembangan *computer cluster* Raspberry Pi 4 pada penelitian ini untuk dapat menciptakan *cluster* Kubernetes dapat berjalan secara lancar pada Raspberry Pi tanpa adanya kendala untuk dapat menjalankan aplikasi berbasis arsitektur *microservices.* Berikut merupakan Gambar 2. 4 dari arsitektur K3s.



Gambar 2. 4 Arsitektur Kubernetes (K3s)

2.2.7 Django

Django merupakan kerangka kerja pengembangan aplikasi yang dibangun menggunakan Bahasa pemrograman Python. Kerangka kerja Django menawarkan teknik pengembangan aplikasi berbasis web yang cepat dan sederhana. Django juga memberikan tingkat keamanan yang baik pada pengembangan *website* dengan mengimplementasi *cross-site scripting (XSS)* dan *cross-site request forgery (CSRF)*. Keunggulan pengembangan aplikasi dengan menggunakan kerangka kerja Django yaitu pada penggunaan bahasa pemrograman Python, Python mempunyai *library* yang sangat banyak yang tersedia pada internet dan membantu *developer* mengembangkan

aplikasinya secara luar biasa. Selain itu Bahasa pemrograman Python mudah dipahami dan dapat membuat banyak fungsi dengan penggunaan kode yang minimal [17].

2.2.8 PostgreSQL

PostgreSQL merupakan *database* relasional atau yang sering disebut *Relational DataBase Management System* (RDBMS) paling populer yang dikembangkan oleh Berkeley *Computer Science Department*. *Database* PostgreSQL memiliki banyak keunggulan dan fitur seperti transaksi, *views, sub selects, checks*, dan *foreign key support* [18]. PostgreSQL memiliki kelebihan pada proses memuat data yang lebih tinggi, disamping itu arsitektur pada PostgreSQL memiliki tingkat stabilitas yang cukup tinggi dibandingkan *database* relasional lainnya. Database PostgreSQL mampu mengatasi permasalahan pada kompleksitas yang cukup besar [19].

2.2.9 Ansible

Ansible merupakan alat pengelolaan *server* yang bekerja untuk mengkonfigurasi *server* dengan berdasarkan internet dan *ssh keys* yang digunakan untuk *remote login*. Kemudian digunakan implementasi *built in python binaries* yang nantinya akan digunakan untuk dapat mengkonfigurasi *server*. Arsitektur pada Ansible dirancang untuk mengontrol dan mengelola *server*. Pada Ansible terdapat *inventory* yang merupakan konfigurasi *file* terkait *server* yang nantinya akan dikelola oleh Ansible. Pada *inventory file* terdapat IP *address server*, konfigurasi *variable,* dan konfigurasi *ssh keys* yang nantinya dapat diakses oleh Ansible [20].

Ansible digunakan untuk menginisiasi *computer cluster* menggunakan Kubernetes dengan penerapan Kubernetes K3s yang sangat ringan dan kompatibel untuk dapat dibangun pada *computer cluster* Raspberry Pi 4. Penggunaan Ansible dikarenakan memudahkan untuk melakukan konfigurasi Kubernetes pada ketiga *node* pada *computer cluster* secara sekaligus dalam satu waktu tanpa harus mengkonfigurasi secara berulang-ulang pada tiap *node*.

2.2.10 Quality of Service (QOS)

Quality of Service merupakan sebuah teknik pada pengelolaan *delay*, *packet loss*, dan *bandwidth* untuk arus dalam jaringan. Tujuannya adalah untuk mempengaruhi setidaknya satu parameter diantara keempat dasar parameter pada QOS supaya pengguna menjadi lebih produktif melalui informasi yang didapatkan dari hasil pengujian performa aplikasi berbasis jaringan. *Quality of Service* mengacu pada kemampuan sebuah jaringan untuk dapat menyediakan layanan yang lebih baik di lalu lintas jaringan melalui teknologi yang beragam. Berikut merupakan parameter *Quality of Service* yang dapat digunakan [21].

1. *Throughput*

   *Throughput m*erupakan suatu total dari perhitungan jumlah kedatangan paket sukses pada *destination* selama interval waktu tertentu yang dibagi durasi interval waktu tertentu.

2. *Delay*

   Delay merupakan waktu tunda paket yang disebabkan oleh proses transmisi pada suatu titik ke titik yang lain yang menjadi tujuan.

3. *Packet loss*

   Packet loss merupakan kegagalan transmisi paket IP *address* pada saat mencapai tujuannya.

2.2.11 *High Availability*

High Availability merupakan sebuah karakteristik dari sistem untuk memulihkan dan melindungi sistem dari gangguan kecil dalam jangka waktu singkat dengan pendekatan cara yang otomatis. Terdapat beberapa karakteristik *high availability* atau ketersediaan tinggi dalam penerapannya, yang pertama *categorization outage* yang merupakan prediksi yang memberitahu bahwa sistem sedang mengalami masalah dan dapat menerapkan solusinya. Selanjutnya yaitu *system categorization* yang memberitahu tentang sebuah persyaratan waktu penonaktifan maksimum, dan

yang terakhir yaitu *automatic protection* atau *recovery* yang merupakan teknologi dan solusi yang berpengaruh pada kebutuhan *high availability* [22].

2.2.12 Apache JMeter

Apache JMeter merupakan *open-source software* yang dirancang untuk mengukur kinerja suatu sistem. *Software* ini dapat digunakan pada pengujian kinerja pada sumber dinamis maupun statis yang menstimulasikan beban pada *server* atau jaringan untuk menguji kekuatan dan menganalisa kinerja secara keseluruhan berdasarkan jenis beban yang berbeda [23].

2.2.13 CPU Usage

CPU merupakan komponen utama pada sistem yang dapat memproses data dan kode yang dapat menghasilkan sebuah aplikasi. Angka pada proses CPU yang sedang bekerja dinamakan CPU usage atau penggunaan jumlah CPU pada proses sebuah sistem [24].

2.2.14 Prometheus

Prometheus merupakan alat monitoring sistem yang dapat mengumpulkan data dalam bentuk metrik. Data monitoring pada Prometheus dapat diekspor ke Prometheus data *source* pada Grafana menggunakan Prometheus exporter yang diinstal pada sistem yang akan dilakukan monitoring sistem. Prometheus memiliki banyak pengguna aktif pada internet dan memiliki banyak komunitas pengembang [25].

2.2.15 Grafana

Grafana merupakan open-*source tool* yang digunakan untuk visualisasi data. Grafana dapat menyatukan data dari berbagai sumber untuk dimuat dalam visualisasi dengan berbagai tipe visualisasi data. Disamping itu Grafana memiliki komunitas

kontributor yang aktif dalam mengembangkan versi terbaru. Grafana sangat berguna untuk visualisasi data dari berbagai sumber dengan antar muka pengguna yang mudah dan dapat digunakan untuk mengekspor data visualisasi dalam format CSV untuk nantinya dilakukan analisis data [26].