BAB II

TINJAUAN PUSTAKA DAN DASAR TEORI

2.1 Tinjauan Pustaka

Adapun penelitian yang dilakukan oleh M. Agung Nugroho, Cuk Subiyantoro (2018), tentang Analisis Cluster Container pada Kubernetes dengan Infrastruktur Google Cloud Platform. Pada penelitian tersebut membahas tentang penerapan *clustering* pada Kubernetes yang di install pada layanan Cloud milik Google, dan dalam proses pengujian yang dilakukan adalah menggunakan tools online bernama *locus.io* (Nugroho, 2018).

Selain itu penelitian yang dilakukan oleh Anwar Fahlepi (2020), tentang Microservice Deployment Menggunakan Google Kubernetes Engine Dengan Otomatisasi Infrastructure as a Code Dan CI/CD Pipeline menggunakan Jenkins. Dimana pada penelitian tersebut membahas tentang proses pengembangan sistem dari modeling pengembangan secara *deployment* yang terotomatisasi sampai dengan proses *Maintenance* (PAHLEPI, 2020).

Kemudian pada penelitian yang dilakukan oleh Ali Akbar Khatami (2020), tentang Implementasi *High Availability Storage Server*. Pada penelitian tersebut membahas tentang bagaimana implementasi *big data* pada *storage server* tetap dapat tersedia dengan kemungkinan keadaan *down* yang kecil menggunakan Kubernetes sebagai *container orchestration* (KHATAMI, 2020).

Lalu pada penelitian yang dilakukan oleh Muhammad Naufalammar (2021), tentang Analisis Performansi High Availability Web Server Pada Cluster GKE (Google Kubernetes Engine) Menggunakan Infrastruktur Google Cloud Platform. Penelitian tersebut membahas tentang penggunaan Kubernetes Engine Google pada Google Cloud Platform untuk penelitian peforma web server dengan memanfaatkan teknologi Kubernetes (Muhammad, 2021).

Tabel 2.1 Rangkuman Keterkaitan Dengan Penelitian Sebelumnya

No	Penulis	Judul	Tahun	Metode
1	M. Agung Nugroho, Cuk Subiyantoro	Analisis Cluster Container pada Kubernetes dengan Infrastruktur Google Cloud Platform	2018	*Load Testing* pada *cluster container* Kubernetes dengan infrastruktur GCP (Google Cloud Platform)
2	Anwar Fahlepi	Microservice Deployment Menggunakan Google Kubernetes Engine Dengan Otomatisasi Infrastructure as a Code Dan CI/CD Pipeline menggunakan Jenkins	2020	Implementasi CI/CD menggunakan Google Kubernetes Engine
3	Ali Akbar Khatami	Implementasi *High Availability Storage Server*	2020	*High Availability Storage Server* dengan Kubernetes
4	ROSYADI, IMRON and Utama, Shoffin Nahwa and Putra, Oddy Virgantara	Implementation Autoscaling Container Web Server using Kubernetes Promox Based on Server University of Darussalam Gontor	2019	Scaling Web Server pada Container dengan Kubernetes pada Proxmox
5	Saleh Dwiyatno, Edi Rakhmat, dan Oki Gustiawan	*Implementasi Virtualisasi Server Berbasis Docker Container*	2019	Virtualisasi dengan Docker Container

6	Muhammad Naufalammar	Analisis Performansi High Availability Web Server Pada Cluster GKE (Google Kubernetes Engine) Menggunakan Infrastruktur Google Cloud Platform	2021	Testing Avaibility Web Server Pada Cluster Google Kubernetes Engine

2.2 Dasar Teori

2.2.1. Scalability

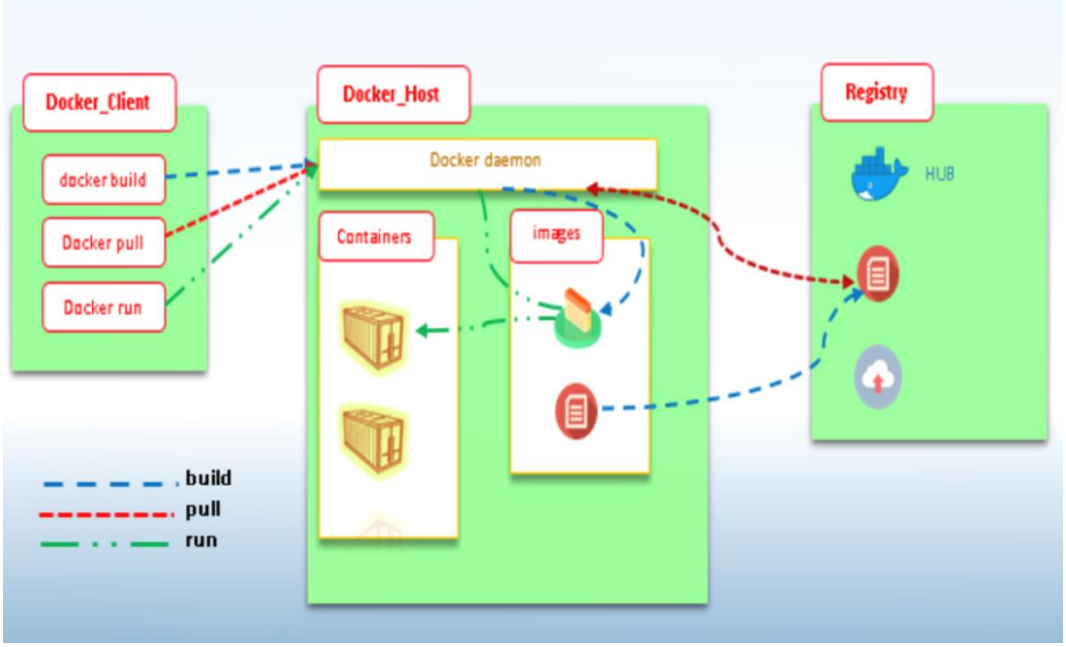
Scalability adalah kemampuan suatu sistem, jaringan, atau proses untuk dapat menangani penambahan beban yang diberikan tanpa harus berdampak pada kinerja. *Scalability* dapat mengacu pada dua prespektif, yaitu *latency* dan *throughput*, dimana dari kedua prespektif tersebut jika terjadi penambahan pengguna secara signifikan, *latency* adalah jumlah detik atau milidetik setiap respon yang umunya relatif konstan, sedangkan *throughput* adalah jumlah pengguna yang di tangani dalam satuan waktu. Skalabilitas (Lehrig et al., 2015a) merupakan kemampuan sistem berbasis cloud untuk meningkatkan kapasitas pengiriman layanan perangkat lunak dengan memperluas kuantitas layanan perangkat lunak yang disediakan, saat diperlukan peningkatan permintaan layanan pada periode waktu tertentu.

2.2.2. Container

Container (Madhumathi, 2018a) adalah suatu alternative teknologi *open-source stand-alone* yang ringan dan berdiri sendiri. Cara kerja dari Container adalah memvirtualisasi komputer sistem agar dapat berbagi sumber daya *Kernel* untuk menjalankan kode, *runtime, system tools, system library, environtment*, dan

konfigurasi sistem . Container akan mengemas *library*, konfigurasi, *dependency*, dan aplikasi dalam sebuah image, menjadikan Container dapat dengan mudah dipindah-pindah antar sistem.

Salah satu container yang umum digunakan adalah docker. Docker memiliki 4 komponen yaitu docker *daemon*, docker *image*, docker *registry*, dan docker *container*. Arsitektur docker (Shah & Dubaria, 2019a) dapat dilihat pada Gambar 2.1.



Gambar 2.1 Arsitektur Docker

Komponen-komponen tersebut memiliki cara kerja yang berbeda dan memiliki fungsi masing-masing yang dapat dijelaskan sebagai berikut (Rad et al., 2017a) :

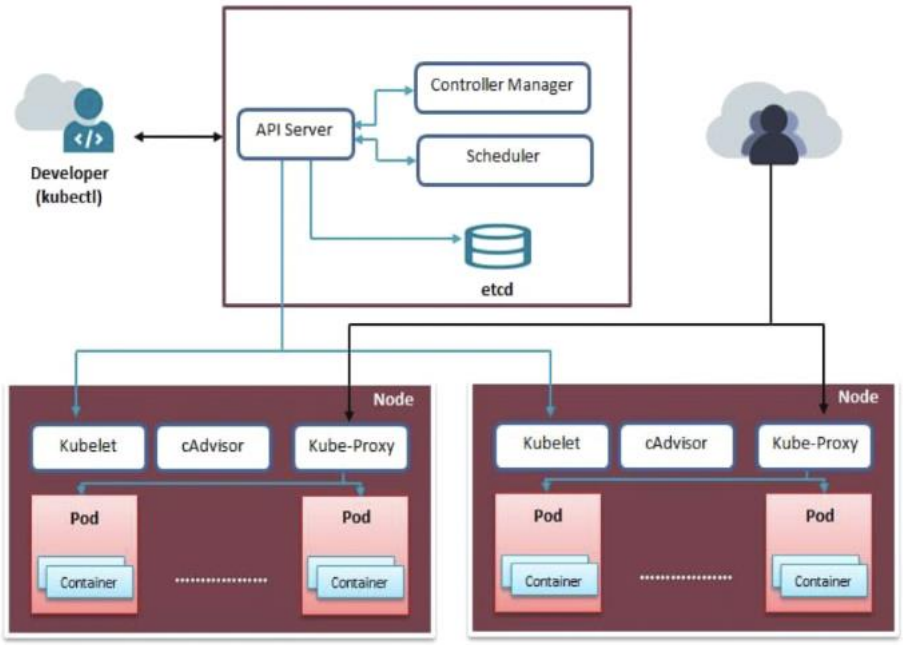
a. Docker *Daemon*: Docker dapat digambarkan sebagai *server* dan *client* berbasis aplikasi. Docker *daemon* menerima request dari docker *client* yang merupakan *command line interface* (CLI) dan memberikan *output* sesuai *request*. Selanjutnya REST API dan binary CLI didistribusikan oleh sistem docker.
b. Docker *image* merupakan sebuah image yang digunakan sebagai basis untuk menjalankan docker *container*.
c. Docker *registry* merupakan sebuah *repository* yang menyimpan *image* dari docker yang dapat didistribusikan secara *private* atau *public*.

Docker *container* merupakan bagian dari sistem docker yang dibangun dengan basis docker *image*

2.2.3. Kubernetes

Kubernetes (Jakóbczyk, 2020a) adalah platform *open-source* yang digunakan untuk melakukan manajemen *workloads* pada aplikasi yang telah di kontainerisasi. Selain manajemen *workloads* Kubernetes juga menangani proses otomatisasi *deploy*, dan *scaling* pada perangkat lunak yang dikembangkan. Dikenalkan pertama kali oleh Google, akan tetapi sekarang telah dilanjutkan oleh Cloud Native Computing Fondation (CNCF).

Kubernetes mengadopsi tipe arsitektur master-slave dengan bagian utama yang terdiri dari controller/master dan slave/workers sebagai nodes. Sistem kerja atau arsitektur kubernetes dengan 1 master dan 2 nodes dapat dilihat pada Gambar 2.2.



Gambar 2.2 Arsitektur Kubernetes (Hausenblas, 2018)

Kubernetes master memiliki tanggung jawab untuk melakukan manajemen pada kubernetes kluster dimana terdapat beberapa komponen didalamnya, yaitu (Shah & Dubaria, 2019a):

a. API *server* yang merupakan *gateway* untuk semua *command* REST dalam mengatur *input* dari kubernetes. *Developer* umumnya menggunakan perintah *kubectl* untuk berkomunikasi.
b. *Controller Manager* berperan sebagai pengatur proses dalam kubernetes, misalkan *pods*, *end point*, dsb.
c. *Scheduler* berfungsi untuk memastikan implementasi dari konfigurasi pada *service* dan *pod* yang terjadwal ke dalam *worker node* Kubernetes
d. Etcd merupakan *key-value* yang dibuat CoreOS sebagai *backend* database yang menyimpan status informasi dan data konfigurasi pada kluster.

Pada *workers node* juga memiliki beberapa komponen yang terdiri dari (Shah & Dubaria, 2019a) :

a. Kubelet yang memiliki peran sebagai *gateway* antar *node* dan *master*. Komponen ini bekerja berdasarkan *PodSpec* yang merupakan obyek berformat yaml yang berperan dalam mendeskripsikan sebuah *pod*. Selanjutnya komponen ini memastikan container yang dideskripsikan pada *PodSpec* berjalan dengan baik.
b. *Kube-proxy* adalah *network proxy* yang menghubungkan kubernetes API pada setiap *node*.

Pods terdiri atas host yang dimanajemen oleh kubernetes. Host memiliki sekelompok container yang menjalankan tugas secara spesifik sejenis atau aplikasi yang dikenal dengan nama pod. Pod dapat direplikasi menjadi replica pod yang berfungsi dalam proses redudansi dan skalabilitas

2.2.4. Kubeadm

Kubeadm (kubernetes.io, 2022) adalah fitur yang digunakan untuk menjalankan fasilitas pembuatan *cluster* pada Kubernetes. Kubeadm adalah alat resmi yang dibuat oleh Cloud Native Computing Foundation (CNCF) layaknya Kubernetes, menjadikan *tools* ini sebagai standar perusahaan untuk memberikan teknologi *cluster* diantaranya *single-node, multi-node, HA, self-hosted*.

2.2.5. Load dan testing performa

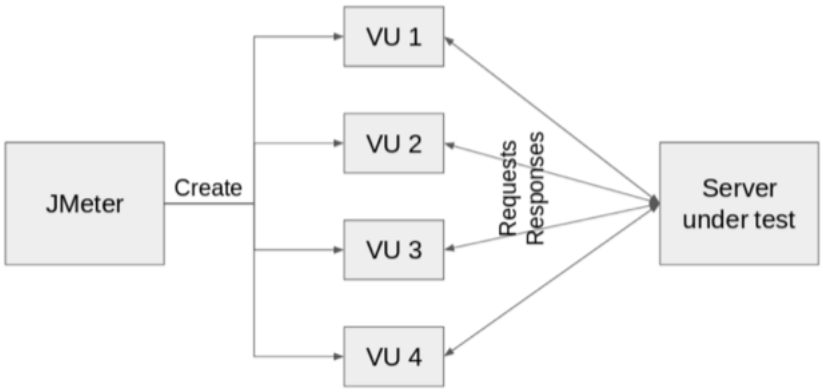
Load testing adalah teknik performance testing yang mana respon sistem diukur dalam berbagai load condition. Pengujian ini membantu menentuka

bagaimana software berperilaku ketika beberapa user mengakses software secara bersamaan. Load testing diperlukan untuk membuat simulasi akses aplikasi web / website secara simultan. Cara ini lebih baik dibandingkan dengan harus mengundang sekian belas, atau puluh orang sekaligus untuk mengakses sebuah website. *Performance testing* adalah jenis pengujian untuk memastikan perangkat lunak akan bekerja dengan baik dibawah beban kerja yang diharapkan. Tujuan utamanya bukan untuk mencari bug, tapi untuk mengeliminasi *performance bottleneck*. Fokus dari *Performance Testing*, yaitu *speed* untuk menentukan apakah aplikasi merespon dengan cepat, *scalability* untuk menentukan apakah jumlah maksimum user load dapat ditangani, dan *stability* untuk menentukan apakah aplikasi stabil dengan berbagai beban (Permatasari, 2020a).

2.2.6. Apache Jmeter

Apache JMeter adalah perangkat lunak open source, 100% aplikasi Java murni dirancang untuk memuat tes perilaku fungsional dan mengukur kinerja. Ini pada awalnya dirancang untuk pengujian load/stress testing Web Application, FTP Application dan Database server test. Apache JMeter dapat digunakan untuk menguji kinerja baik pada sumber daya statis dan dinamis. Perangkat lunak ini dapat digunakan untuk mensimulasikan beban berat pada server, sekelompok server, jaringan atau objek untuk menguji analisa kinerja secara keseluruhan dengan model beban yang berbeda (Simon, 2021a) .

Apache Jmeter berjalan secara *multithreaded* dan dapat disimulasikan dengan menggunakan sejumlah *virtual users* (VU), proses ini dijelaskan pada Gambar 2.3.



Gambar 2.3 Multithreading Pada Jmeter (Rodrigues et al., 2019)

Apache Jmeter dapat mendukung beberapa model load test yang mendukung fleksibilitas dan dapat digunakan dalam berbagai model skenario seperti beberapa model berikut (Rodrigues et al., 2019a):

a. Pengecekan respon time dari sebuah aplikasi web berdasarkan user yang melakukan akses
b. Melakukan beberapa test terkait behavior dari satu atau dua aplikasi atau lebih dengan memanfaatkan environment yang berbeda
c. Melakukan test terhadap limit akses dari aplikasi, misalkan mengetahui batas maksimum user yang dapat melakukan akses ke sebuah aplikasi
d. Melakukan test secara fungsi pada aplikasi secara ter-otomatisasi dengan memanfaatkan test *non-regression*, validasi fungsi dengan model *continuous integration*.
e. *Reproduce* load pada level *production* untuk melakukan tes bug pada lingkungan QA
f. Otomasi *load testing* dengan proses pada lingkungan *continuous integration*

g. Melakukan test untuk database, LDAP atau *active directory*, *mail server* dan network services lain

Dalam proses *reporting*, apache jmeter dapat melakukan analisa dengan hasil proses berupa OOTB HTML, hasil real time dengan integrasi influxDB dan grafana, hasil akhir dapat berupa csv atau xml, dan dapat menjadi solusi dengan integrasi pada proses *continuous integration* seperti *jenkins*.