Hasil Penelitian
Yang Tidak Dipublikasikan

Tinjauan Pustaka
Kubernetes Container Management

Berkah I. Santoso, ST, MTI
Guson P. Kuntarto, ST, M.Sc
Irwan Prasetya Gunawan, Ph.D
Yusuf Lestanto, ST, M.Sc, MBA

Fakultas Teknik dan Ilmu Komputer
Universitas Bakrie
Jakarta

2025

LEMBAR PENGESAHAN HASIL PENELITIAN YANG TIDAK DIPUBLIKASIKAN

1.	Judul Penelitian	: Tinjauan Pustaka Kubernetes Container Management
2.	Peneliti Utama	
	a. Nama Lengkap	: Berkah I. Santoso, ST, MTI
	b. Jenis Kelamin	: Laki Laki
	c. Pangkat/Golongan/NIDN	: Asisten Ahli / III b / 0309068003
	d. Bidang Keahlian	: Cloud Computing, Networking
	e. Program Studi	: Informatika
3.	Tim Peneliti	: Guson P. Kuntarto, ST, M.Sc, Irwan Prasetya Gunawan, Ph.D, Yusuf Lestanto, ST, M.Sc, MBA
4.	Jangka waktu penelitian	: 1 November 2024 – 19 Desember 2024

Jakarta, 20 Desember 2024

Menyetujui,

Ketua Lembaga Penelitian dan Pengembangan	**Peneliti Utama**
(**Deffi Ayu Puspito Sari, Ph.D.**)	(**Berkah I. Santoso, ST, MTI**.)
NIDN: 0308078203	NIDN: 0309068003

Abstrak

Pengembangan aplikasi moderen yang bersifat modular dengan titik berat pada faktor-faktor seperti kecepatan waktu rilis aplikasi, kestabilan aplikasi, elastisitas pengaturan komponen berikut modul-modul aplikasi, kemampuan migrasi *host* aplikasi dan komunikasi antar aplikasi tanpa harus mengorbankan fungsi dan keamanan aplikasi merupakan beberapa syarat wajib yang menjadi tuntutan korporasi dan organisasi terkait pengembangan aplikasi inti untuk menunjang keberlangsungan bisnis perusahaan saat ini. Aplikasi moderen dengan skala kritikalitas tinggi selalu memerlukan elastisitas pengaturan sumber daya komputasi yang aman, memiliki ketersediaan sumber daya, memiliki efisiensi sumber daya komputasi yang tinggi dan pengelolaan kapasitas yang bersifat akuntabel, obyek komputasi yang selalu dapat dimonitor serta memiliki siklus keberlangsungan layanan yang terjamin [1]. Saat ini terdapat berbagai terobosan teknologi lebih lanjut dalam rangka mempersiapkan ketersediaan sumber daya komputasi pada kebutuhan pengembangan aplikasi dengan cara *micro-services*: penambahan berbagai atribut layanan pada kode sumber aplikasi utama dengan atribut berskala mikro, obyek yang bersifat modular, layanan yang bersifat elastis dan fungsional, serta obyek yang siap terhadap kebutuhan layanan antar *multi-cloud* dengan pendekatan pengelolaan berbasis *container*. Saat ini Kubernetes terlihat memiliki keterbatasan pada kompleksitas penyiapan *platform* komputasi dan pengaturan kapasitas obyek Kubernetes yaitu *pod* yang bersifat *rigid* agar tidak melebihi batas CPU dan memori yang tersedia [2], sehingga hal tersebut menjadi beberapa tantangan tim *developer* dan tim *operations* (DevOps). Tim DevOps membutuhkan pengaturan Kubernetes yang tepat untuk membentuk aplikasi modern berbasis *micro-services* dalam rangka pencapaian ketersediaan *object* yang maksimal untuk peningkatan waktu pengembangan dan implementasi. Penulis mencoba untuk memberikan tinjauan literatur yang diharapkan dapat bermanfaat bagi para pengembang aplikasi modern yang masih membutuhkan mekanisme pengelolaan terstruktur atas layanan aplikasi berbasis *container* dengan memanfaatkan layanan *multi-cloud* agar antar komponen Kubernetes memiliki fungsionalitas maksimal pada *cloud infrastructure* dan mempercepat implementasi aplikasi. Kompleksitas Kubernetes dapat dimaksimalkan dalam rangka penanganan *cluster container* dengan fasilitas lengkap dan Kubernetes mampu mengelola beragam layanan, walaupun perlu dilakukan pembatasan *pod* lebih lanjut agar tidak melebihi batas *Central Processing Unit* (CPU) dan memori fisik yang tersedia. Pada hasil keseluruhan, kita dapat melihat kinerja maksimal Kubernetes untuk menangani beban komputasi yang berat dengan menggunakan *container*. Lebih lanjut, Kubernetes dapat mencapai kinerja komputasi lebih tinggi dan sifat response yang lebih fleksibel untuk pemantauan kesehatan layanan yang bersifat adaptif dengan kemampuan pemuatan ulang layanan pada saat ditemukan adanya kegagalan pada pengelolaan *container*.

Daftar Isi

Bab 1

Pendahuluan

1.1 Latar Belakang

Tren penggunaan dan pengelolaan *container* pada *microservices* telah menjadi keniscyaan dalam rangka penyediaan layanan komputasi untuk percepatan pengembangan dan implementasi aplikasi korporat. Beberapa tujuan pemanfaatan *container* adalah meningkatkan modularitas layanan, mempermudah pengelolaan layanan pada modul pembentuk aplikasi dengan tidak harus mengganggu layanan aplikasi lain, meningkatkan elastisitas penempatan dan perpindahan layanan aplikasi pada *virtual machine* ataupun *physical host* serta memudahkan mekanisme alokasi beban komputasi untuk modul-modul layanan aplikasi [3]. Platform Kubernetes sebagai sarana pengelolaan *container* tersedia pada beberapa penyedia layanan *public cloud* seperti Amazon Web Services (AWS)®, Alibaba®, Google Cloud®mengadopsi pemanfaatan *virtual machine* (VM) dan *container*. Pemanfaatan tersebut bertujuan agar mendapatkan beberapa fasilitas otomasi pada implementasi layanan. Modularitas dan elastisitas pada penerapan *container* terkait dengan penyediaan sumber daya komputasi untuk mempercepat implementasi layanan, penulis rasakan mulai menggeser penggunaan VM pada beberapa tahun terakhir [4]. Apabila jumlah sumber daya komputasi berupa *container* sudah sangat banyak untuk dikelola, maka konfigurasi *container* menjadi lebih rumit dan menyita waktu *cloud administrator*. Dengan semakin besarnya kebutuhan aplikasi yang dikembangkan tentunya memerlukan dukungan sumber daya komputasi yang sangat besar [5]. Peranan kedua tim, yaitu tim pengembang layanan aplikasi moderen dan tim operasional menjadi sangat besar, hingga kedua tim tersebut menginginkan layanan yang dikelola bersifat aman, dapat diakses kapan saja, handal dan dinamis mengikuti besaran beban komputasi. Pengelolaan *container*, saat ini merupakan keharusan, baik pihak *cloud administrator* maupun pihak *developer* untuk memastikan bahwa semua obyek pendukung berfungsi secara maksimal [6]. Terdapat penerapan lebih lanjut terkait pengembangan aplikasi dengan memanfaatkan *micro-services* yaitu membutuhkan dukungan layanan *multi-cloud* dengan pendekatan berbasis *container* sebagai percepatan *time-to-deliver* aplikasi modern dari ranah *development* kepada *production* serta penambahan komponen-komponen layanan pada kode sumber aplikasi yang bersifat fungsional, adaptif, modular dan berskala mikro. Pada *platform* pengelolaan *container* seperti Kubernetes masih memiliki keterbatasan kinerja secara keseluruhan dibalik kompleksitas penyiapan *platform* layanan komputasi [7]. Tim DevOps diharapkan dapat mencapai peningkatan waktu pengembangan dan *deployment* layanan moderen berbasis *microservices* dengan cara memanfaatkan *orchestration framework* berbentuk *platform terintegrasi* pada Kubernetes.

1.2 Rumusan Masalah

Laporan ini mencoba membahas penggunaan *framework* pengelolaan *container* Kubernetes dengan menggunakan pendekatan literatur secara deskriptif terkait bagaimana pemanfaatan *framework* pengelolaan *container* Kubernetes untuk memenuhi kebutuhan penyediaan layanan komputasi dan implementasi aplikasi modern.

1.3 Batasan Masalah

Ruang lingkup pada pembahasan ini meliputi beberapa hal sebagai berikut, diantaranya adalah :

1. *Framework* pengelolaan *container* yang diteliti adalah Kubernetes *Container Management*.
2. Fokus pembahasan terkait aspek fitur dan unjuk kerja pengelolaan *container* Kubernetes.

1.4 Tujuan Penelitian

Penggunaan pendekatan tinjauan fitur unggulan dan unjuk kerja pada laporan ini ditujukan untuk identifikasi *framework* pengelolaan *container* Kubernetes untuk mendukung pengembangan dan *deployment* aplikasi moderen. Identifikasi tersebut bertujuan agar terdapat arah optimalisasi yang jelas terkait penggunaan *framework* pengelolaan *container* Kubernetes, dengan cara menyederhanakan mekanisme penyiapan layanan komputasi yang memiliki sifat kompleksitas tinggi.

1.5 Sistematika Penulisan

Adapun sistematika penulisan pada laporan penelitian ini meliputi :

1. Bab 1 membahas latar belakang permasalahan dari kompleksitas Kubernetes *Container Management* pada penyiapan layanan komputasi. Selanjutnya merupakan perumusan masalah yang dibingkai berikut batasan masalah pembahasan agar dapat memenuhi tujuan penelitian yang diharapkan.
2. Bab 2 membahas konsep yang mendasari penelitian terkait arsitektur pembentuk *framework* pengelolaan *container* Kubernetes berikut keuntungan dan *drawbacks* pada *framework* pengelolaan *container* tersebut.
3. Bab 3 membahas tinjauan *framework* Kubernetes *Container Management*.
4. Bab 4 memberikan kesimpulan dari hasil penelitian terkait tinjauan deskriptif terkait fitur dan unjuk kerja pada Kubernetes *Container Management*.

Bab 2

Gambaran Arsitektur Kubernetes *Container Cluster Management*

Containers merupakan terobosan pengembangan layanan aplikasi dalam rangka penyediaan paket layanan terintegrasi yang modular bagi kebutuhan aplikasi moderen. Pada aplikasi bisnis berskala korporat yang berjalan diatas *container*, tim *Development* bersama dengan tim *Operations* (DevOps) memiliki peran penting dan tanggung jawab dalam rangka memberikan jaminan tingkat layanan aplikasi tanpa *downtime* terkait pengelolaan *container* tersebut. Pada kejadian suatu *container* yang mengalami *downtime*, maka *container* yang lain harus mengambil alih peran *container* yang mengalami *downtime* tersebut dengan berperilaku *active start* sesuai dengan sifat konsep layanan *high availability, scalability, failover* [8].

Beberapa fitur pada Kubernetes diantaranya meliputi hal sebagai berikut, yaitu:

1. *Service discovery and load balancing*
   Kubernetes dapat melakukan pengelolaan suatu *container* berdasarkan nama domain atau alamat *Internet Protocol* (IP) yang telah dilakukan konfigurasi pada *Domain Name Service* (DNS) server. Pada saat *traffic* jaringan menjadi tinggi pada suatu *container*, maka Kubernetes akan melakukan pengaturan keseimbangan beban komputasi atau *load balancing* dan pengaturan distribusi *traffic* jaringan dalam rangka menjaga kestabilan implementasi layanan aplikasi.

2. *Storage orchestration*
   Kubernetes dapat melakukan pengelolaan *storage* secara otomatis untuk dapat menggunakan sistem secara lokal, sistem pada penyedia *public cloud* atau sistem *storage* pada jaringan.

3. *Automated rollouts and rollbacks*
   You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.

4. *Automatic bin packing*
   Kubernetes dapat melakukan pengaturan dengan konsep *cluster of nodes* untuk dapat menjalankan tugas-tugas komputasi yang diletakkan pada *container*. Tim DevOps dapat melakukan pembatasan kapasitas CPU dan memori yang dibutuhkan untuk *container* masing-masing. Kubernetes dapat mengatur ketepatan konfigurasi yang sesuai dengan ketersediaan sumber daya komputasi.

5. *Self-healing*
   Kubernetes dapat melakukan *restart containers* yang mengalami kegagalan, menggantikan peran

suatu *container* dengan *container* lainnya, mematikan layanan *container* yang tidak memberikan respon pada mekanisme pemeriksaan ketepatan alokasi CPU dan memori, mengatur kesiapan *container* sebelum memberikan layanan komputasinya kepada pengguna.

6. *Secret and configuration management*
   Kubernetes dapat digunakan untuk menyimpan dan mengelola informasi yang bersifat sensitif, seperti *password*, token OAuth untuk otentikasi, dan kunci untuk Secure Shell (SSH). Tim DevOps dapat melakukan implementasi profil keamanan untuk aplikasi pada *container* tanpa harus membangun ulang *container image* yang bertujuan agar profil keamanan dan atribut kerahasiaan data tidak terekspos pada konfigurasi *stack*.

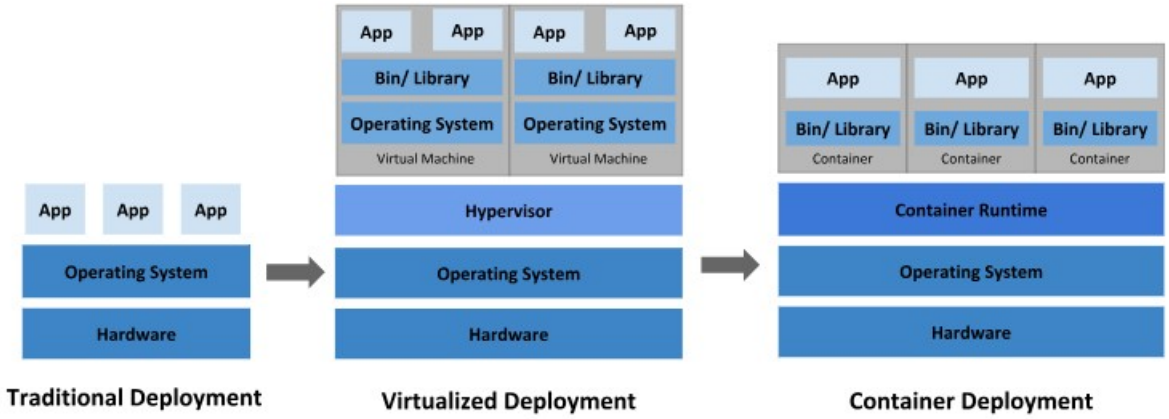
7. *Batch execution*
   Kubernetes dapat melakukan pengelolaan untuk menangani beban komputasi seperti pemrosesan *batch* dan integrasi secara berkelanjutan serta penggantian *container* yang mengalami kegagalan layanan.

8. *Horizontal scaling*
   Peningkatan skalabilitas atau penurunan skalabilitas layanan dapat dilakukan pada Kubernetes menggunakan perintah sederhana pada *Command Line Interface* (CLI), antarmuka pengguna dan berdasarkan tersentuhnya batas atas dari penggunaan *Central Processing Unit* (CPU) secara otomatis.

9. *IPv4/IPv6 dual-stack*
   Alokasi alamat IPv4 and IPv6 untuk digunakan pada Pods dan layanan.

10. *Designed for extensibility*
    Penambahan fasilitas pada *cluster* Kubernetes yang dapat dilakukan tanpa harus melakukan pengubahan kode sumber aplikasi utama.

Pembaca dapat melihat penggambaran perbandingan arsitektur komputasi berbasis *container* dengan arsitektur komputasi berbasis virtualisasi dan arsitektur komputasi tradisional, seperti terlihat pada gambar berikut :



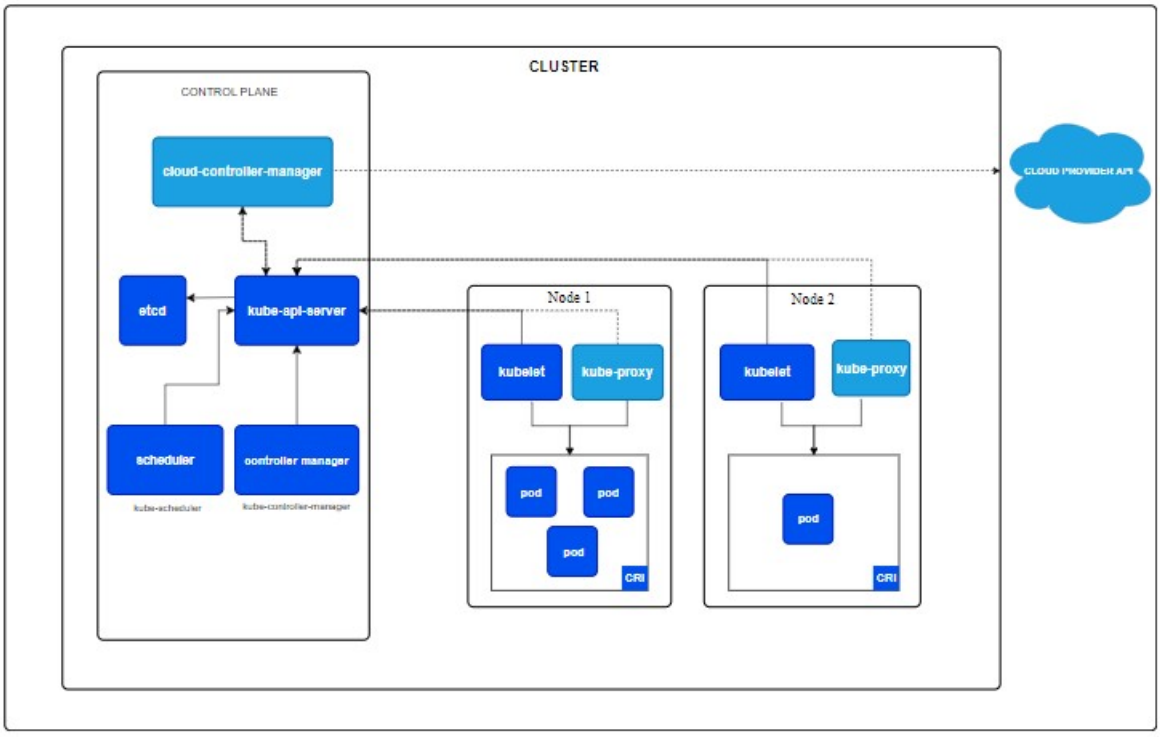
Gambar 2.1: Perbandingan Arsitektur Komputasi Tradisional, Virtualisasi dan Container [9]

Pada implementasi komputasi tradisional terlihat bahwa aplikasi dilakukan instalasi diatas sistem operasi dengan *hardware server* sebagai pondasinya, sehingga utilisasi komputasi tradisional tersebut merupakan yang paling rendah dalam operasional pemenuhan layanan. Beranjak pada implementasi

komputasi virtualisasi terlihat bahwa aplikasi dilakukan instalasi diatas pustaka yang terletak diatas sistem operasi pada mesin virtual (*virtual machine*–VM). Sementara kumpulan VM tersebut dikelola menggunakan Hypervisor selaku manajer VM yang diinstalasi diatas sistem operasi pada *hardware*. Utilisasi komputasi virtualisasi menempati posisi menengah dimana beban komputasi dijalankan pada VM dengan pengelolaan Hypervisor sepanjang cukup tersedia sumber daya komputasi fisik pada *hardware server*. Selanjutnya pada implementasi komputasi berbasis *container*, terlihat bahwa aplikasi dilakukan instalasi berikut konfigurasi diatas pustaka yang terletak diatas *container*. Pengelolaan kumpulan *container* tersebut dilakukan oleh *container runtime* yang diinstalasi diatas sistem operasi pada *hardware*. Utilisasi komputasi berbasis *container* menempati posisi tertinggi dimana beban komputasi dijalankan pada *container* dengan pengelolaan oleh *container runtime* dengan memaksimalkan ketersediaan sumber daya komputasi fisik pada *hardware server*.

Cluster pada Kubernetes terdiri dari *control plane* berikut sekumpulan aplikasi yang bertindak seperti mesin komputasi disebut sebagai *nodes* dengan menjalankan berbagai aplikasi pada *container*. Setiap *cluster* membutuhkan sekurang-kurangnya satu *worker node* untuk dapat menjalankan *Pods*. Suatu *worker node* menjadi tempat bagi *Pod* yang menjadi komponen untuk menangani beban komputasi aplikasi. Suatu *control plane* mengelola *worker node* dan *Pod* pada suatu *cluster*. Pada area *production*, *control plane* biasanya berjalan untuk mengelola antar *hardware server* dan suatu *cluster* seringkali terdiri dari bermacam-macam *node* dalam rangka menyediakan sistem komputasi berbasis *container* dengan mengutamakan aspek *fault-tolerance* dan *high availability*.

Pembaca dapat melihat penggambaran arsitektur cluster Kubernetes seperti terlihat pada gambar berikut :



Gambar 2.2: Arsitektur Cluster Kubernetes [9]

2.1 Komponen Kubernetes *Control Plane*

Fungsi dari Kubernetes *control plane* adalah sebagai komponen yang melakukan pengambilan keputusan secara global terkait dengan pengelolaan *cluster*, seperti mekanisme penjadwalan, pendeteksian dan pemberian respon atas kondisi *cluster*. Komponen *control plane* dapat dijalankan pada setiap mesin pada *cluster* dan kode *setup* dapat digunakan untuk menghidupkan semua layanan *control plane* pada mesin yang sama secara otomatis, tanpa harus menghidupkan *container* yang digunakan secara bersama-sama.

2.1.1 kube-apiserver

Application Programming Interface (API) *server* merupakan komponen dari Kubernetes *control plane* yang memberikan akses pengelolaan terhadap Kubernetes API. API *server* merupakan antar muka Kubernetes *control plane* pada sisi terdepan. Implementasi utama Kubernetes API *server* adalah komponen kube-apiserver yang dikembangkan untuk keperluan skalabilitas secara horizontal dengan cara implementasi beberapa obyek yang dapat dilakukan penyeimbangan traffic jaringan antar obyek tersebut.

2.1.2 etcd

Fungsi etcd adalah sebagai tempat penyimpanan nilai kunci Kubernetes yang selalu tersedia dan bersifat konsisten untuk semua data *cluster*. Apabila cluster Kubernetes menggunakan etcd sebagai media penyimpanan data *cluster*, maka *administrator* Kubernetes perlu menjalankan rencana *back up* data.

2.1.3 kube-scheduler

kube-scheduler merupakan komponen *control plane* yang melakukan monitoring Pods terbaru tanpa adanya *node* yang dipasangkan dalam rangka pemilihan node untuk dijalankan.

Beberapa faktor yang mempengaruhi terhadap penentuan keputusan penjadwalan pada *container* terdiri dari:

- Persyaratan sumber daya individual dan kolektif.
- Batasan-batasan *hardware/software/policy*.
- Kesamaan karakteristik spesifikasi dan perbedaan karakteristik spesifikasi.
- Penempatan data.
- Interferensi beban kerja inter *container*.

2.1.4 kube-controller-manager

Komponen *control plane* yang menjalankan proses *controller*. Masing-masing *controller* berada pada proses terpisah dengan pengurangan kompleksitas melalui langkah kompilasi menjadi program biner tunggal yang berjalan pada proses tunggal.

Beberapa tipe *controller* diantaranya adalah:

1. Node *controller*: tipe *controller* yang bertanggung jawab untuk fungsi pemberitahuan dan pemberian respon ketika *node* mengalami henti layanan.
2. Job *controller*: tipe *controller* yang melakukan monitoring terhadap obyek-obyek *job* dalam rangka representasi tugas-tugas bersifat *one-off*. Selanjutnya *job controller* membangun *Pods* untuk penyelesaian eksekusi tugas-tugas tersebut.

3. EndpointSlice *controller*: tipe *controller* yang mengumpulkan obyek-obyek EndpointSlice dalam rangka penyediaan koneksi antara layanan dengan *Pods*.

4. ServiceAccount controller: tipe *controller* yang membuat layanan dasar akun untuk *namespace* baru.

2.1.5 cloud-controller-manager

Cloud-controller-manager merupakan komponen *control plane* Kubernetes yang menanamkan logika kendali khusus untuk kebutuhan *cloud*. *Cloud controller manager* dapat menghubungkan *cluster* kedalam *Application Programming Interface* (API) dari penyedia layanan *cloud* dan memisahkan beberapa komponen yang berinteraksi dengan *platform cloud* dari komponen yang hanya berinteraksi dengan *cluster*. *Cloud-controller-manager* hanya menjalankan *controller* yang khusus menyesuaikan pada penyedia layanan *cloud*. Apabila tim DevOps menjalankan Kubernetes pada *data center* organisasi sendiri atau pada komputer pribadi, maka tidak memerlukan lagi *cloud controller manager*. *Cloud-controller-manager* menggabungkan beberapa kendali loop secara mandiri kedalam program biner yang dijalankan sebagai proes tunggal. Tim DevOps dapat memperbesar kapasitas secara horizontal untuk meningkatkan kinerja agar dapat mengatasi kegagalan pada *node*. Ketergantungan *controller* berikut ini terhadap penyedia layanan *cloud* diantaranya adalah:

- *Node controller*; berfungsi untuk memeriksa apakah suatu *node* telah dihapus dari cloud setelah berhenti memberikan respon.

- *Route controller*; berfungsi untuk membangun lintasan pada infrastruktur *cloud*.

- *Service controller*; berfungsi untuk membuat, memperbaharui dan menghapus komponen pembagi beban komputasi pada penyedia layanan *cloud*.

2.2 Komponen Kubernetes *Node*

Komponen *node* dapat berjalan pada setiap *node*, melaksanakan fungsi pengelolaan *pods* yang bekerja dan menyediakan lingkungan kerja Kubernetes.

2.2.1 kubelet

Kubelet merupakan obyek *agent* yang dapat berjalan pada setiap *node* pada *cluster* dalam rangka memastikan berjalannya *container* pada suatu *Pod*.

Kubelet mengambil sekumpulan *PodSpecsy* ang disediakan melalui berbagai macam mekanisme dalam rangka memastikan *container* pada *PodSpecs* berjalan dengan baik. Selain itu, kubelet tidak mengelola *container* yang tidak dibuat pada lingkungan Kubernetes.

2.2.2 kube-proxy

kube-proxy merupakan suatu layanan *proxy* jaringan pada setiap *node* yang tergabung pada suatu *cluster* sebagai bagian dari layanan Kubernetes. kube-proxy mengelola aturan jaringan pada suatu *node*, sehingga memperbolehkan *traffic* jaringan pada *Pod* dapat keluar atau masuk melalui *cluster*. kube-proxy dapat memanfaatkan lapisan *filtering* paket pada sistem operasi apabila terdapat *traffic* jaringan yang apabila tidak tersedia maka kube-proxy akan meneruskan *traffic* jaringan kepada obyek kube-proxy itu sendiri.

2.2.3 container runtime

container runtime merupakan fungsi komponen dasar dalam rangka meningkatkan efektifitas pengelolaan Kubernetes. *container runtime* memiliki tanggung jawab dalam pengelolaan eksekusi dan siklus container dalam Kubernetes. Kubernetes melakukan pengelolaan *container runtime* dalam bentuk *containerd*, CRI-O, dan implementasi Kubernetes *Container Runtime Interface* (CRI).

2.3 Kubernetes *Addons*

Addons menggunakan sumber daya Kubernetes seperti *DaemonSet, Deployment*, dan lainnya dalam rangka mendapatkan fasilitas pengelolaan *cluster*. Pada penyediaan fasiltas ditingkat *cluster* mmembutuhkan sumber daya *namespaced* sebagai *addons* dalam *namespace kube-system*.

2.3.1 DNS

Keseluruhan *cluster* Kubernetes memerlukan *cluster* Domain Name System (DNS) yang merupakan suatu layanan tambahan dari fungsi DNS pada jaringan untuk mengelola salinan konfigurasi penamaan untuk layanan Kubernetes. *Container* yang dijalankan oleh Kubernetes akan memasukkan pencarian salinan DNS pada server DNS.

2.3.2 *Web User Interface* sebagai *Dashboard*

Dashboard digunakan untuk antar muka pengguna berbasis web dalam rangka mengelola dan melakukan fungsi *troubleshoot* terhadap aplikasi yang berjalan pada *cluster*.

2.3.3 container resource monitoring

Container Resource Monitoring berfungsi untuk menyimpan salinan umum terkait ukuran berdasarkan satuan waktu terkait container pada *database* terpusat serta penyedia antar muka berbasis web untuk melihat data tersebut.

2.3.4 cluster-level logging

cluster-level logging menyediakan mekanisme yang bertanggung jawab dalam pencatatan operasional sistem *container* pada pencatatan terpusat dengan penambahan antar muka pengguna untuk fasilitas pencarian catatan.

2.3.5 network plugins

Network plugins merupakan komponen *software* untuk keperluan implementasi *container network interface* (CNI). *Network plugins* bertanggung jawab untuk mengalokasikan alamat *Internet Protocol* (IP) pada *pod* dalam rangka membangun komunikasi dengan komponen lainnya pada suatu *cluster*.

2.4 Variasi Arsitektur *Cluster* Kubernetes

Kubernetes mengelola komponen-komponen inti secara konsisten, sedangkan cara implementasi dan pengelolaan dapat bervariasi. Tim DevOps perlu memiliki pengetahuan terkait variasi pengelolaan Kubernetes dalam rangka perumusan desain dan tata kelola *cluster* Kubernetes agar dapat memenuhi persyaratan operasional.

2.4.1 Implementasi *Control Plane*

Beberapa komponen *control plane* dapat diimplemntasikan dalam beberapa cara, diantaranya adalah sebagai berikut:

- Implementasi secara tradisional; dijalankan langsung pada server fisik atau VM sebagai layanan *systemd*.
- *Static Pods*; langsung dikelola oleh *kubelet* pada suatu *node* yang dijalankan melalui *kubeadm*.
- *Self-hosted*; langsung menjalankan *Pod* pada cluster Kubernetes itu sendiri yang dikelola oleh fungsi-fungsi dasar pada Kubernetes seperti implementasi dan penetepan konfigurasi dasar.
- *Managed Kubernetes services*; disediakan langsung oleh penyedia layanan *cloud computing*.

2.4.2 Pertimbangan Penempatan Beban Kerja Komputasi

Penempatan beban kerja komputasi yang meliputi beberapa komponen dapat bervariasi berdasarkan ukuran *cluster*, persyaratan kinerja dan aturan pada operasional *cluster*. Pada *cluster* yang berukuran kecil atau pada *cluster* untuk keperluan pengembangan, komponen *control plane* dan beban kerja komputasi pengguna *container* dapat dijalankan pada *node* yang sama. Berbeda halnya dengan implementasi *cluster* berskala besar yang menggunakan *node-node* tersendiri untuk mengelola komponen *control plane* dalam rangka pemisahan fungsi dari beban kerja komputasi pengguna. Beberapa organisasi atau perusahaan seringkali menjalankan *add-on* yang memiliki fungsi penting dan perangkat bantu monitoring pada *node-node* suatu *control plane*.

2.4.3 Perangkat Bantu Pengelolaan *Cluster*

Beberapa perangkat bantuk seperti **kubeadm, kops**, dan **Kubespray** memiliki pendekatan teknis yang berbeda dalam rangka pengelolaan dan implementasi *cluster*. Sifat adaptif dari arsitektur Kubernetes dapat dimanfaatkan untuk keperluan desain dan konfigurasi secara spesifik yang sesuai dengan kebutuhan masing-masing organisasi dengan mempertimbangkan kompleksitas operasional, kinerja komputasi dan beban kerja pengelolaan obyek-obyek komputasi.

2.5 Keuntungan Penggunaan Kubernetes Container Management

Penyebaran dan implementasi Kubernetes yang dikelola tim DevOps pada organisasi atau perusahaan disebabkan karena tim DevOps dipermudah dalam memperoleh pengetahuan dan keahlian dalam pelaksanaan tata kelola *platform* orkestrasi berbasis *container* dan efisiensi penggunaan sumber daya komputasi yang tinggi untuk dapat menjalankan banyak layanan aplikasi berskala korporat. Berikut ini merupakan beberapa keuntungan penggunaan Kubernetes dalam pengelolaan arsitektur komputasi berbasis *microservice*, diantaranya adalah sebagai berikut citebenefitKubernetes2022:

1. Penghematan biaya operasional untuk mengelola orkestrasi sumber daya komputasi berbasis *container*;
2. Peningkatan efisiensi sumber daya komputasi untu keperluan pengembangan dan operasional pada arsitektur berbasis *container*;
3. Beban kerja komputasi dapat didistribusikan pada lingkungan komputasi yang bersifat *multicloud* seperti IBM®Cloud, Amazon Web Services (AWS®), Google Cloud Platform®dan Microsoft Azure®;

4. Fleksibilitas untuk melakukan migrasi Kubernetes antar penyedia layanan cloud untuk menghindari terjadinya *vendor lock-in*;
5. Kemudahan otomatisasi implementasi dan skalabilitas sumber daya komputasi pada Kubernetes;
6. Kestabilan dan ketersediaan layanan aplikasi yang tinggi pada platform orkestrasi berbasis container Kubernetes;
7. Kemudahan untuk dapat melihat kode sumber layanan sebagai sifat penggunan Kubernetes yang berbasis *open source software*;

2.6 Kelemahan pada pengelolaan Kubernetes Container eksisting

Kubernetes selain memiliki sejumlah keuntungan, juga memiliki beberapa kekurangan dan batasan untuk dapat dilakukan mitigasi dan penyempurnaan serta pengembangan selanjutnya. Berikut ini merupakan beberapa kekurangan dan batasan pada pengelolaan Kubernetes container eksisting adalah sebagai berikut [10] :

1. Kompleksitas Kubernetes dan usaha lebih lanjut untuk mendapatkan pengetahuan tentang tata kelola Kubernetes;
2. Skalabilitas Kubernetes dan kinerja aplikasi pada *container* yang masih perlu ditingkatkan;
3. Persyaratan sumber daya komputasi yang terdiri dari prosesor, memori dan staff DepOps pengelola sumber daya;
4. Keamanan Kubernetes dan kepatuhan kepada regulasi yang perlu disempurnakan, terutama apabila diimplementasikan pada organisasi atau perusahaan yang memilki kepatuhan peraturan ketat;
5. Risiko terkait *vendor lock-in* dari implementasi Kubernetes pada salah satu penyedia layanan *public cloud* dimana terdapat fitur spesifik pada penyedia layanan *public cloud* eksisting yang tidak dimiliki oleh penyedia layanan *public cloud* lainnya.

Bab 3

Tinjauan Kubernetes Container Cluster Management

Pada tinjauan Kubernetes *Container Cluster Management*, penulis melakukan instalasi pada sumber daya komputasi Kali Linux server untuk mendapatkan gambaran pada review yang dilakukan. Penulis menggunakan *kubectl* sebagai perangkat bantu berbasis *command line interface* untuk melakukan instalasi dan mengelola aplikasi pada Kubernetes. Perintah untuk melakukan instalasi kubectl adalah sebagai berikut:

Listing 3.1: Perintah untuk instalasi kubectl pada Kali Linux

```
1 $ sudo apt-get install -y kubectl
```

Selanjutnya penulis melakukan instalasi minikube sebagai aplikasi bantu untuk dapat menjalankan *cluster* Kubernetes pada *node* server. Perintah untuk melakukan instalasi minikube adalah sebagai berikut:

Listing 3.2: Perintah untuk instalasi minikube pada Kali Linux

```
1 $ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 &&
      sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Selanjutnya penulis melakukan aktivasi layanan minikube dan verifikasi Kubernetes dengan perintah sebagai berikut:

Listing 3.3: Perintah untuk start minikube service pada Kali Linux

```
1 $ minikube start
2 $ kubectl get nodes
```

Pembaca dapat melihat hasil dari perintah pada listing 3.3 seperti terlihat pada gambar berikut : Beberapa listing tersebut diatas merupakan contoh-contoh implementasi Kubernetes pada sistem operasi Kali Linux server yang dapat dilakukan menggunakan *command line interface*. Apabila pembaca ingin melakukan implementasi Kubernetes pada sistem operasi lain perlu memperhatikan persyaratan agar instalasi dan konfigurasi dapat dilakukan dengan baik. Beberapa persyaratan tersebut diantaranya meliputi:

- Ketersediaan versi atau porting Kubernetes untuk sistem operasi target;
- Repositori aktif paket Kubernetes yang akan digunakan untuk keperluan instalasi dan aktivasi Kubernetes;

Gambar 3.1: Hasil perintah pada listing 3.3

- Koneksi internet untuk melakukan *download* paket *installattion files* Kubernetes dari repositori;
- Ketersediaan prosesor, *hardisk space* dan memori pada server target;
- Dokumentasi yang cukup lengkap untuk referensi pengelolaan Kubernetes pada sistem operasi target;
- Ketersediaan dan kelengkapan aplikasi pendukung pada sistem operasi target untuk dapat menjalankan Kubernetes.

Implementasi Kubernetes *Cluster Container Management* dapat dilakukan oleh tim DevOps suatu organisasi atau perusahaan dengan memperhatikan keunggulan, kelemahan, batasan dan sumber daya pengetahuan dalam rangka pelaksanaan tata kelola Kubernetes berbasis *container* sebagai pilihan dalam penyediaan sumber daya komputasi bagi keberlangsungan layanan aplikasi.

Bab 4

Kesimpulan

Kubernetes *Container Cluster Management* dapat mengelola *cluster container* secara tepat dalam rangka pengelolaan beragam layanan aplikasi yang mengedepankan efisiensi operasional dan pelaksanaan tata kelola Kubernetes. Beban *overhead* komputasi pada Kubernetes *Container Cluster Management* memiliki beban *overhead* lebih rendah, dengan kinerja komputasi yang cukup tinggi. Mekanisme pengelolaan *container* pada Kubernetes *Container Cluster Management* masih sangat terbuka untuk pengembangan lebih lanjut, terutama untuk penyederhanaan tugas-tugas komputasi yang lebih kompleks. Hal tersebut ditujukan untuk meningkatkan adaptasi pengelolaan komponen *container* pada *add-on* serta perangkat bantu Kubernetes *Container Cluster Management* untuk dapat dikustomisasi oleh penyedia layanan *public cloud*.

Beberapa hal yang masih terbuka untuk dikembangkan lebih lanjut dari penelitian ini adalah karakteristik respons *container* dari Kubernetes *Container Cluster Management* atas kemungkinan terjadinya puncak beban kerja Kubernetes secara tiba-tiba, baik karena terdapat peningkatan permintaan layanan atau karena terdapat serangan siber terhadap infrastruktur Kubernetes berbasis *container*. Aspek lainnya yang masih dapat dikembangkan dari penelitian ini adalah peningkatan kecepatan *backup and recovery* layanan Kubernetes secara adaptif baik secara *inter cloud* maupun antar *public cloud*, terlebih pada saat *system maintenance* atau terdapat *techincal faulty* pada pengelolaan Kubernetes *container cluster*.

Bibliography

[1] Z. Li, H. Wei, Z. Lyu, and C. Lian, "Kubernetes-container-cluster-based architecture for an energy management system," *IEEE Access*, vol. 9, pp. 84 596–84 604, 2021. 1

[2] A. Verma, A. Satpathy, S. K. Das, and S. K. Addya, "Lease: Leveraging energy-awareness in serverless edge for latency-sensitive iot services," in *2024 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, 2024, pp. 302–307. 1

[3] S. Galantino, F. Risso, A. Cazzaniga, F. Garrone, R. Terruggia, and R. Lazzari, "An edge-based architecture for phasor measurements in smart grids," in *2022 AEIT International Annual Conference (AEIT)*, 2022, pp. 1–6. 3

[4] R. Gao, X. Xie, and Q. Guo, "K-tahp: A kubernetes load balancing strategy base on topsis+ahp," *IEEE Access*, vol. 11, pp. 102 132–102 139, 2023. 3

[5] T. Goethals, F. De Turck, and B. Volckaert, "Extending kubernetes clusters to low-resource edge devices using virtual kubelets," *IEEE Transactions on Cloud Computing*, vol. 10, no. 4, pp. 2623–2636, 2022. 3

[6] B. Ramasamy, Y. Na, W. Kim, K. Chea, and J. Kim, "Hacm: High availability control method in container-based microservice applications over multiple clusters," *IEEE Access*, vol. 11, pp. 3461–3471, 2023. 3

[7] X. Zhang, L. Li, Y. Wang, E. Chen, and L. Shou, "Zeus: Improving resource efficiency via workload colocation for massive kubernetes clusters," *IEEE Access*, vol. 9, pp. 105 192–105 204, 2021. 3

[8] L. Poggiani, C. Puliafito, A. Virdis, and E. Mingozzi, "Live migration of multi-container kubernetes pods in multi-cluster serverless edge systems," in *Proceedings of the 1st Workshop on Serverless at the Edge*, ser. SEATED '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 9–16. [Online]. Available: https://doi.org/10.1145/3660319.3660330 5

[9] K. Author, "Overview of kubernetes documentation," accessed February 5th, 2025. [Online]. Available: https://kubernetes.io/docs/concepts/overview/ 6, 7

[10] A. Skotnicky, "The pros and cons of using kubernetes for microservices architecture," accessed February 5th, 2025. [Online]. Available: https://taikun.cloud/the-pros-and-cons-of-using-kubernetes-for-microservices-architecture/ 12