Hasil Penelitian
Yang Tidak Dipublikasikan

Tinjauan Pustaka
Orkestrasi Layanan Containerized-Cloud

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

2022/2023

LEMBAR PENGESAHAN HASIL PENELITIAN YANG TIDAK DIPUBLIKASIKAN

1.	Judul Penelitian	: Tinjauan Pustaka Orkestrasi Layanan Containerized-Cloud
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 Mei 2023 – 28 Juni 2023

Jakarta, 25 Juli 2023

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 yang memiliki waktu rilis cepat, bersifat modular dan stabil merupakan hal wajib yang menjadi tuntutan organisasi atau perusahaan saat ini. Kecepatan pengembangan aplikasi modular tersebut membutuhkan sumber daya komputasi yang harus selalu tersedia, kapasitas sumber daya yang bersifat elastis dan orkestrasi varian sumber daya yang efisien. Variasi model sumber daya komputasi mulai dari Infrastructure-as-a-Service (IaaS) hingga Platform-as-a-Service (PaaS) merupakan kebutuhan *developer* maupun pihak *operational* aplikasi yang bersifat *mandatory* dan umum digunakan dalam rangka percepatan *time-to-deliver* aplikasi modern dari ranah *development* kepada *production*[1]. Terdapat beragam terobosan pada sisi penyediaan sumber daya komputasi terhadap kebutuhan pengembangan aplikasi secara *micro-services*–penambahan komponen-komponen layanan pada kode sumber aplikasi yang bersifat modular, adaptif dan berskala mikro, fungsional, membutuhkan dukungan layanan *multi-cloud* dengan pendekatan berbasis *container*. Beberapa platform orkestrasi *container* seperti Kubernetes dan pengelolaan Docker memiliki keterbatasan berikut keunggulan masing-masing. Kubernetes dirasakan masih memiliki keterbatasan kinerja secara keseluruhan apabila dibandingkan dengan Docker, sedangkan Docker pada mode Swarm terlihat memiliki keterbatasan pada pengelolaan platform yang bersifat heterogen[2]. Tantangan tim *developer* dan tim *operations* (DevOps) adalah bagaimana memanfaatkan kekayaan fitur sekaligus melakukan orkestrasi antara fleksibilitas Kubernetes dengan kinerja maksimal Docker pada mode Swarm untuk membentuk aplikasi modern berbasis *micro-services* dalam rangka peningkatan waktu pengembangan dan implementasi. Penulis mencoba untuk memberikan tinjauan literatur yang diharapkan dapat bermanfaat bagi para pengembang aplikasi modern yang masih membutuhkan mekanisme orkestrasi terintegrasi terhadap layanan berbasis *container* dengan memanfaatkan dukungan layanan *multi-cloud* agar antar komponen *micro-services* memiliki fungsionalitas maksimal dari masing-masing *cloud infrastructure* dan percepatan *deployment* aplikasi modern. Kubernetes, Docker pada mode Swarm dan Apache Mesos dapat menangani cluster container Docker secara efisien dan mampu melakukan pengelolaan beragam layanan, walaupun pendekatan teknis antara ketiga platform orkestrasi tersebut memiliki perbedaan. Beban overhead Kubernetes memberikan hasil yang cukup tinggi apabila dibandingkan dengan Docker pada mode Swarm yang memiliki beban overhead lebih rendah. Sedangkan pada Apache Mesos memberikan unjuk kerja yang tinggi pada mekanisme berbagi pakai sumber daya komputasi untuk beragam *platform*. Pada hasil keseluruhan, kita dapat melihat kinerja dan overhead ketiga platform orkestrasi container tersebut, yaitu Kubernetes memiliki kinerja lebih rendah dibandingkan dengan Docker pada mode Swarm atau Apache Mesos. Pada sisi lain, Kubernetes yang dijalankan pada implementasi desain kompleks menghasilkan sifat response yang lebih fleksibel dan memiliki kemampuan lebih baik apabila dibandingkan dengan Docker pada mode Swarm dan Apache Mesos, sehingga kedua *orchestration containerized platform* tersebut memiliki karakteristik unik masing-masing.

Daftar Isi

Bab 1

Pendahuluan

1.1 Latar Belakang

Trend penggunaan *cloud computing* pada saat ini telah menjadi paradigma komputasi korporat yang digunakan secara umum. Beberapa tujuan penggunaan *cloud computing* adalah penyampaian layanan yang stabil berikut efisiensi biaya, pencapaian skalabilitas, ketersediaan dan kemudahan akses [3]. Beberapa penyedia layanan *public cloud computing* seperti Amazon Web Services (AWS), Google, Alibaba mengadopsi teknik virtualisasi berikut *virtual machine* (VM) dan *container* untuk implementasi aplikasi secara otomatis pada infrastruktur sistem terdistribusi. *Containerization* merupakan teknologi virtualisasi pada pengembangan berikut operasional aplikasi bersifat lincah, ringan, konsisten, memiliki portabilitas sistem operasi terdistribusi dan isolasi sumber daya komputasi [3]. Penggunaan container yang bersifat modular dan lincah dalam hal penyediaan sumber daya komputasi bagi kebutuhan implementasi aplikasi mulai menggeser penggunaan VM pada beberapa tahun terakhir [4]. Layanan *container* tersebut tentunya memiliki banyak obyek pembentuk yang bersifat heterogen, modular hingga *multi-platform*. Heterogenitas dan kompleksitas infrastruktur dari banyaknya *platform* layanan *cloud computing* merupakan salah satu masalah bagi organisasi atau perusahaan yang memiliki aplikasi penunjang proses bisnis berbasis multi-cloud [5]. Tim pengembang aplikasi dan tim operasional selaku pihak penunjang proses bisnis pada dasarnya menginginkan aplikasi yang dikembangkan bersifat aman, selalu dapat diakses, handal dan dinamis. Selain hal tersebut, tim pengembang serta tim operasional perlu memaksimalkan kelebihan masing-masing platform serta layanan *container*. Orkestrasi atas *container*, saat ini merupakan hal yang bersifat *mandatory* bagi pihak *developer* maupun pihak *operational* aplikasi untuk memastikan semua obyek pembentuk berfungsi secara maksimal [1]. Dalam rangka pengelolaan aplikasi secara otomatis seperti penanganan penyediaan berikut implementasi aplikasi secara otomatis, konfigurasi dan penjadwalan sumber daya aplikasi, perawatan aplikasi, peningkatan skalabilitas aplikasi secara otomatis, penyeimbangan beban kerja komputasi berikut *traffic routing* terkait koneksitas aplikasi antar *container* yang aman hingga mekanisme monitoring ketersediaan sumber daya *container*, maka mekanisme orkestrasi *container* mutlak diperlukan organisasi atau perusahaan [3]. Beberapa terobosan pada sisi pengembangan aplikasi secara *micro-services*–penambahan komponen-komponen layanan pada kode sumber aplikasi yang bersifat modular, fungsional, adaptif dan berskala mikro, membutuhkan dukungan layanan *multi-cloud* dengan pendekatan berbasis *container* dalam rangka percepatan *time-to-deliver* aplikasi modern dari ranah *development* kepada *production*. Beberapa platform *container* seperti Docker dan Kubernetes dengan segala keterbatasan berikut keunggulan masing-masing, merupakan pembentuk layanan yang saat ini banyak digunakan oleh pengembang aplikasi, hanya saja masih digunakan terbatas pada *single-cloud infrastructure*. Docker pada mode Swarm masih memiliki keterbatasan pada fleksi-

bilitas untuk platform yang bersifat heterogen, sedangkan Kubernetes juga masih memiliki keterbatasan kinerja secara keseluruhan apabila dibandingkan dengan Docker [2]. Apabila tim *developer* dan *operations* (DevOps) dapat memanfaatkan serta melakukan orkestrasi antara fleksibilitas Kubernetes dengan kinerja maksimal Docker pada mode Swarm, kedekatan erat terhadap kernel pada Apache Mesos untuk membentuk aplikasi modern berbasis *micro-services* dengan dukungan *multi-cloud infrastructure* diharapkan dapat mencapai peningkatan waktu pengembangan dan implementasi. Para pengembang aplikasi modern masih membutuhkan *orchestration framework* berbentuk *platform terintegrasi* yang mampu melakukan pengelolaan layanan berbasis *container* dengan memanfaatkan dukungan layanan *multi-cloud* agar antar komponen *micro-services* memiliki fungsionalitas maksimal dari masing-masing *cloud infrastructure* dan percepatan *deployment* aplikasi modern.

1.2 Rumusan Masalah

Laporan ini mencoba membahas penggunaan orkestrasi *container* pada layanan *cloud computing* dengan menggunakan pendekatan tinjauan sistematis literatur terkait bagaimana pembentukan *framework* orkestrasi *container* tepat guna agar fungsionalitas berikut fitur-fitur pada infrastruktur *containerized cloud* menjadi maksimal dalam rangka percepatan penyediaan sumber daya aplikasi dan implementasi aplikasi modern.

1.3 Batasan Masalah

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

1. Platform *containerized cloud* yang diteliti adalah Kubernetes, Docker Swarm, Apache Mesos.
2. Fokus pembahasan terkait aspek orkestrasi keempat *containerized cloud platform*.

1.4 Tujuan Penelitian

Penggunaan pendekatan tinjauan sistematis literatur pada laporan ini ditujukan untuk identifikasi terkait bagaimana pembentukan *framework* sebagai *orchestration platform* terintegrasi untuk mengelola *containerized cloud* yaitu Kubernetes, Docker Swarm dan Apache Mesos dalam rangka pemenuhan fungsionalitas maksimal antar komponen *micro-services* pada infrastruktur komputasi terdistribusi guna percepatan penyediaan sumber daya terkait dengan implementasi aplikasi modern.

1.5 Sistematika Penulisan

Adapun sistematika penulisan pada laporan penelitian ini meliputi :

1. Bab 1 membahas latar belakang permasalahan dari munculnya beberapa terobosan *containerized cloud platform* pembentuk *micro-services* seperti Kubernetes, Docker Swarm dan Apache Mesos dengan pendekatan orkestrasi pada infrastruktur terdistribusi. 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 *container orchestration* seperti *micro-services, containerized cloud platform, orchestration* berikut keuntungan

dan *drawbacks* dari *container orchestration* serta metode tinjauan literatur yang digunakan pada penelitian.

3. Bab 3 memberikan paparan kesimpulan dari hasil penelitian terkait tinjauan literatur pada identifikasi pembentukan framework *container* terintegrasi agar dapat melakukan orkestrasi *containerized cloud*.

Bab 2

Mengenal Arsitektur Container Orchestration

Containerized cloud pada dasarnya merupakan penerapan mekanisme paket-paket *logic* yang menyatukan perangkat lunak dengan aspek-aspek ketergantungan pendukung perangkat lunak tersebut secara bersama dalam rangkat membentuk abstraksi aplikasi. *Container* berbeda halnya dengan *virtual machine* (VM)–penyediaan sumber daya komputasi pada level ketersediaan abstraksi perangkat keras dengan teknik virtualisasi dimana masing-masing VM harus mengelola sistem operasinya sendiri, *container* menggunakan teknik untuk virtualisasi sumber daya pada level abstraksi sistem operasi dimana semua *container* berbagi pakai sistem operasi yang sama dengan beban overhead komputasi yang rendah. Berdasarkan perbedaan level abstraksi tersebut, maka *container* menjadi lebih lincah dan lebih adaptif dalam kaitannya dengan portabilitas aplikasi yang tinggi, efisiensi sumber daya komputasi dan lingkungan komputasi yang konsisten. *Container* memiliki teknologi untuk mendefinisikan obyek-obyek terstandarisasi bagi implementasi aplikasi pada lingkungan sistem komputasi yang terisolasi [4]. Berdasarkan perkembangan tren *container*, maka pengelola departemen teknologi informasi (TI) pada organisasi atau perusahaan memerlukan teknik orkestrasi *container* untuk memudahkan pengelolaan aplikasi yang berjalan diatas sumber daya *container*. Berbeda dengan pengelolaan orkestrasi VM–membutuhkan kendali dalam skala besar, orkestrasi *container* hanya membutuhkan kendali dalam skala kecil dan proses pengelolaan siklus *container* dilakukan secara otomatis dalam hal alokasi sumber daya komputasi, penyediaan sumber daya komputasi untuk implementasi aplikasi, peningkatan skalabilitas sumber daya komputasi secara otomatis, monitoring kesehatan sumber daya komputasi *container*, perpindahan container antar sistem operasi, mekanisme penyeimbangan beban komputasi, konfigurasi keamanan dan konfigurasi jaringan antar *container*. Sifat-sifat otomatis yang adaptif pada *container* akan mempermudah pengelolaan *container* dalam jumlah besar secara simultan bagi penyedia layanan *cloud computing*. Sistem orkestrasi *container* yang bersifat halus dan kuat merupakan faktor kunci dalam pengelolaan utilisasi sumber daya komputasi secara menyeluruh, efisiensi energi dan kinerja aplikasi. Bertambahnya beban kerja *cloud computing* secara eksponensial dalam hal permintaan sumber daya komputasi, peningkatan konsumsi bandwidth untuk aplikasi dan pemenuhan persyaratan *quality of services* maka akan menyebabkan pergeseran lingkungan *cloud computing* dari tradisional ke lingkungan *fog computing* serta infrastruktur TI yang mendekat pada pengguna. Pergeseran lingkungan komputasi tersebut membutuhkan sistem orkestrasi *container* pada infrastruktur bersifat *hybrid* yang kemampuannya harus ditingkatkan untuk menjawab tingginya heterogenitas sumber daya komputasi, distribusi aplikasi, keberagaman beban kerja komputasi dan pemenuhan persyaratan keamanan. Beberapa lapisan tugas pada orkestrasi container terdiri dari: beban kerja aplikasi, cluster komputasi, infrastruktur dan optimalisasi *engine* sumber daya

komputasi. Beban kerja komputasi didefinisikan terkait dengan permintaan pengguna aplikasi terhadap orkestrator, cluster komputasi yang merupakan kumpulan obyek-obyek komputasi fisik maupun virtual terkoneksi pada jaringan, infrastruktur berupa private cloud, public cloud, fog computing berikut edge device dan optimalisasi *engine* sumber daya komputasi untuk mempermudah pembentukan klasifikasi karakter beban kerja komputasi dan analisis kinerja komputasi [4].

Arsitektur *container orchestration* secara umum terdiri dari (1) arsitektur aplikasi yang menjelaskan perilaku dan struktur internal komponen-komponen container aplikasi secara bersama-sama sebagai satu kesatuan; (2) infrastruktur yang menjelaskan lingkungan komputasi atau platform tempat operasional bermacam-macam aplikasi; (3) tujuan orkestrasi container yang menjelaskan berbagai macam peningkatan yang akan dicapai; (4) pemodelan perilaku dan analisis prediksi yang menjelaskan pengenalan bentuk dan simulasi perilaku aplikasi serta sistem terkait tren pada masa yang akan datang berdasarkan data-data yang dikumpulkan sebelumnya; (5) penyediaan sumber daya komputasi yang menjelaskan tentang kebijakan komputasi secara heuristik dalam hal pengelolaan sumber daya komputasi [3]. Pada arsitektur aplikasi yang mendefinisikan komposisi aplikasi pada *container* berikut mekanisme implementasi, eksekusi dan perawatan, terdiri dari beberapa hal berikut ini :

1. Monolithic ;
2. Microservice ;
3. Serverless.

Pada infrastruktur yang mendefinisikan lingkungan komputasi atau *platform* tempat operasional bermacam-macam aplikasi, terdiri dari beberapa hal berikut ini :

1. Single Cloud ;
2. Multi Cloud ;
3. Hybrid Cloud.

Pada tujuan orkestrasi *container* yang memberikan definisi berbagai macam peningkatan yang akan dicapai oleh lingkungan komputasi, terdiri dari beberapa hal sebagai berikut :

1. Efisiensi sumber daya komputasi ;
2. Efisiensi energi untuk komputasi ;
3. Efisiensi biaya ;
4. Jaminan pencapaian Service Level Agreement (SLA).

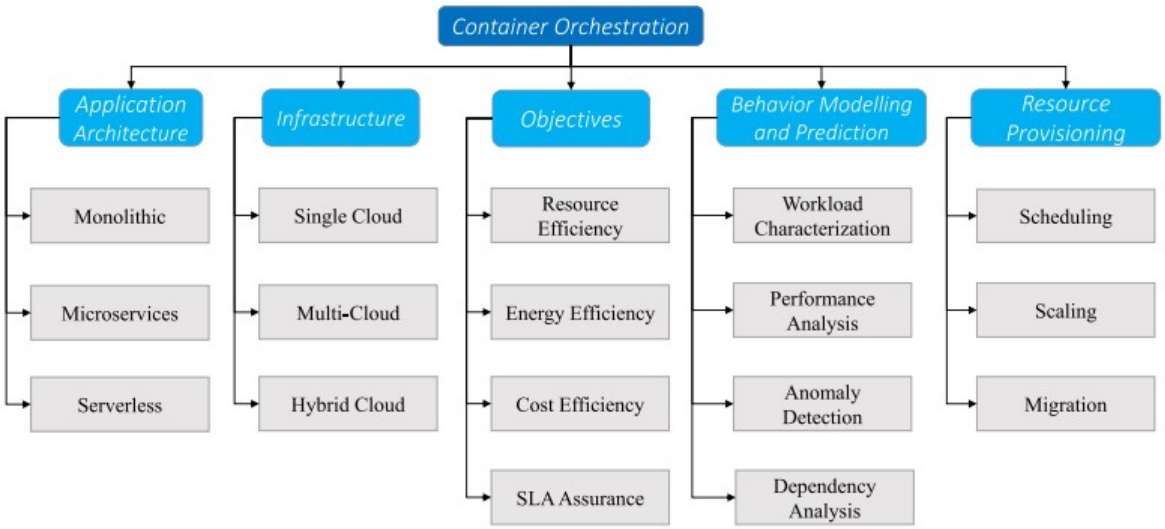
Pada pemodelan perilaku dan analisis prediksi yang menjelaskan pengenalan bentuk dan simulasi perilaku aplikasi serta sistem terkait tren pada masa yang akan datang berdasarkan data-data yang dikumpulkan sebelumnya, terdiri dari beberapa hal, diantaranya adalah :

1. Analisis dan pembentukan karakter beban kerja komputasi ;
2. Analisis kinerja komputasi ;
3. Pendeteksian keanehan atau anomali pada sistem ;
4. Analisis terhadap ketergantungan sistem.

Pada penyediaan sumber daya komputasi yang menjelaskan tentang kebijakan komputasi secara heuristik dalam hal pengelolaan sumber daya komputasi terdiri dari beberapa hal sebagai berikut :

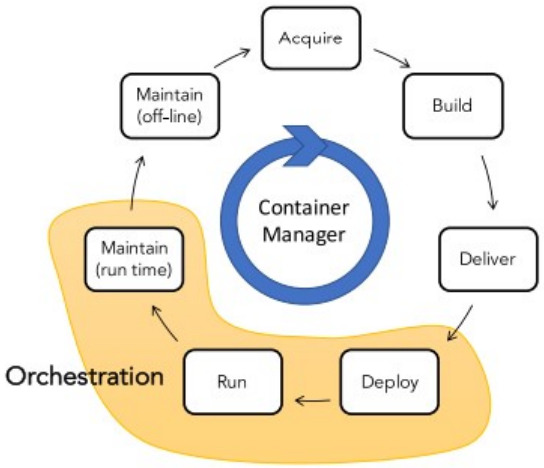
1. Penjadwalan sistem orkestrasi *container* ;
2. Pengaturan skalabilitas sumber daya komputasi *container* ;
3. Pengaturan perpindahan obyek-obyek *container*.

Skema penggambaran lengkap terkait dengan arsitektur orkestrasi *container* dapat kita lihat pada gambar berikut ini : Arsitektur orkestrasi *container* sangat berkaitan dengan siklus *container* dalam hal



Gambar 2.1: Arsitektur *containerized orchestration* [3]

pengelolaan koneksitas antar *container*, penyiapan sumber daya *container* dan implementasi *container*. Container manager menyediakan Application Programming Interface (API) untuk mendukung sekurang-kurangnya fase Acquire ke fase Run pada saat *Orchestrator* memperbolehkan fase implementasi, eksekusi dan perawatan fase-fase *runtime*. Penggambaran lengkap mengenai siklus *container* dapat kita lihat pada gambar berikut ini :



Gambar 2.2: Siklus *container* [4]

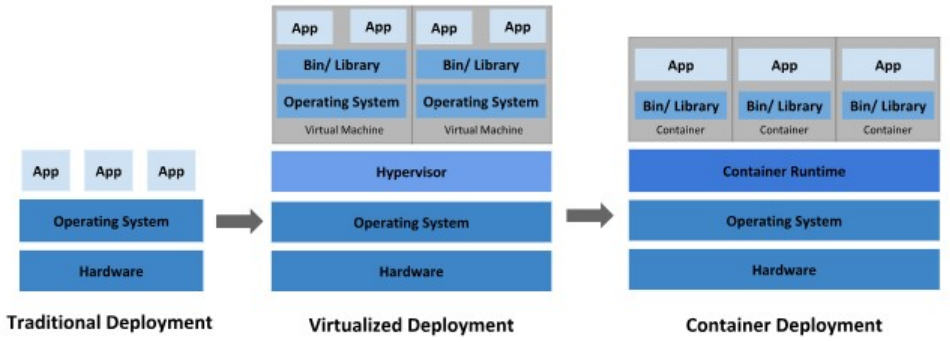
2.1 Kubernetes

Kubernetes (https://www.kubernetes.io) merupakan perangkat bantu orkestrasi container berbasis *open source software* yang pertama kali didesain dan dikembangkan oleh *software engineer* Google. Selanjutnya Google melakukan donasi proyek Kubernetes pada yayasan Cloud Native Computing pada tahun 2015. Orkestrasi pada Kubernetes memungkinkan *container administrator* membangun layanan aplikasi yang tersebar pada *container* dengan platform yang berbeda-beda, melakukan penjadwalan *container* untuk dapat terkoneksi antar *cluster*, melakukan peningkatan skalabilitas *container* dan melakukan pengelolaan kesehatan komputasi *container* setiap saat. Salah satu kemampuan Kubernetes adalah dapat mengurangi proses-proses manual yang terjadi pada saat implementasi aplikasi berbasis *container* dan peningkatan kapasitas sumber daya komputasi aplikasi berbasis *container*. Kemampuan Kubernetes dalam membangun *cluster* untuk kelompok *host* yang terdiri dari mesin fisik dan mesin virtual, menjalankan *container* Linux pada suatu *platform* dapat mempermudah dan membantu meningkatkan efisiensi pekerjaan pengelolaan *cluster*. *Cluster* pada orkestrasi *container* Kubernetes dapat tersebar antara *public cloud, private cloud* atau *hybrid cloud*, sehingga menjadikan Kubernetes sebagai *platform* yang cukup ideal bagi penyedia jasa hosting aplikasi berbasis *cloud-native* atau *container* untuk percepatan peningkatan kapasitas sumber daya komputasi. Kubernetes juga memiliki fitur pemindahan aplikasi tanpa harus melakukan desain ulang dalam rangka mempermudah perpindahan beban kerja komputasi dan mekanisme penyeimbangan beban kerja komputasi [6]. Beberapa komponen utama dari Kubernetes adalah sebagai berikut :

- **Cluster**, untuk pengelolaan obyek atau node komputasi ;
- **Control Plane**, sekumpulan proses yang mengendalikan banyak node pada Kubernetes sebagai awal penugasan perintah komputasi ;
- **Kubelet**, layanan yang berjalan pada banyak *node* dan memiliki tugas untuk membaca keterangan lengkap *container* serta digunakan untuk memastikan *container* yang telah didefinisikan sebelumnya dapat menjalankan layanannya ;
- **Pod**, kumpulan *container* yang diimplementasikan kepada *node* tunggal, hal ini terkait bahwa semua *container* pada Pod berbagi pakai alamat Internet Protocol (IP), *Inter Process Communication* (IPC), nama *host* dan sumber daya lainnya.

Administrator *container* dapat menggunakan Kubernetes *pattern* untuk melakukan pengelolaan konfigurasi, siklus dan skalabilitas layanan serta aplikasi berbasis *container*. Red Hat® selaku korporasi penyedia layanan berbasis open source software berskala *enterprise* telah mengembangkan Red Hat® OpenShift® sebagai Kubernetes *container orchestrator* untuk skala *enterprise* [6]. Pembaca dapat melihat perbandingan implementasi aplikasi tradisional, implementasi pada sumber daya virtualisasi dan implementasi pada sumber daya container, seperti terlihat pada gambar berikut ini : Adapun fitur-fitur Kubernetes adalah sebagai berikut :

- *Service discovery* dan *load balancing* ;
- *Storage orchestration* ;
- *Automated rollout* dan *rollback* ;
- *Automatic bin packing* ;
- *Self-healing* ;



Gambar 2.3: Perbandingan implementasi aplikasi tradisional, virtualisasi dan *container* [7]

- *Secret and configuration management.*

Sedangkan beberapa constraint yang terdapat pada Kubernetes adalah sebagai berikut [7]:

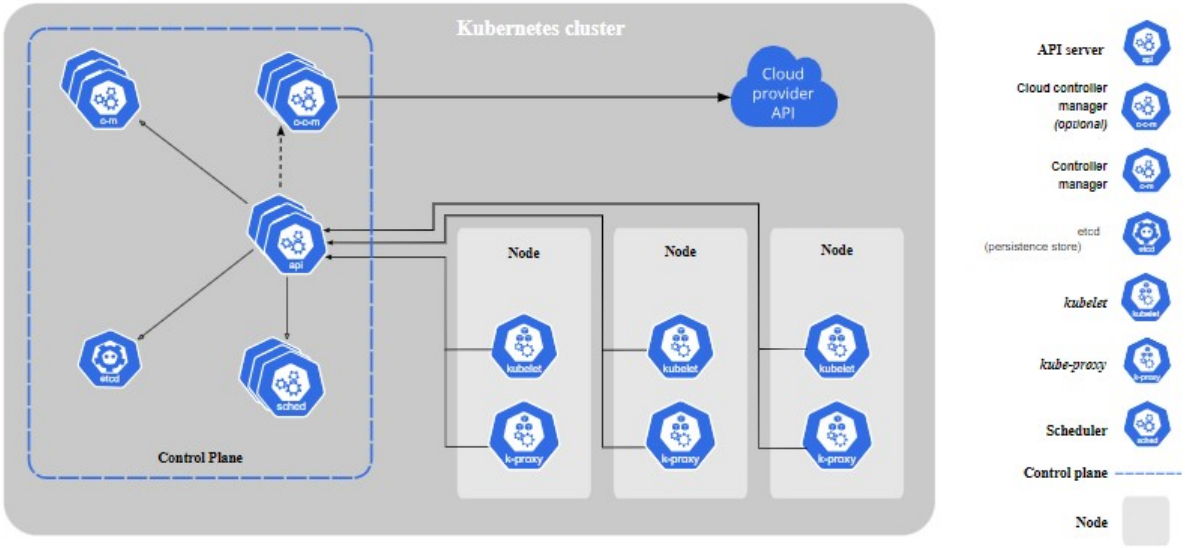
- Kubernetes tidak membatasi tipe-tipe aplikasi yang didukung;
- Kubernetes tidak melakukan implementasi kode sumber aplikasi dan tidak membangun aplikasi pembaca;
- Kubernetes tidak menyediakan layanan-layanan pada level aplikasi;
- Kubernetes tidak melakukan dikte untuk mekanisme pencatatan, monitoring atau solusi berbasis peringatan;
- Kubernetes tidak mengharuskan menggunakan satu bahasa konfigurasi tertentu;
- Kubernetes tidak menyediakan atau tidak melakukan adopsi konfigurasi mesin secara komprehensif;
- Kubernetes tidak menggunakan kendali secara terpusat dalam menjalankan proses-prosesnya.

Berikut ini merupakan komponen dari Kubernetes cluster dalam rangka memberikan gambaran bagi pembaca :

2.2 Docker Swarm

Docker (`https://www.docker.com`) mode Swarm merupakan fitur *advance* pada Docker untuk mengelola *cluster* Docker *daemons* yang dikembangkan kedalam Docker Engine. Beberapa fitur pada Docker mode Swarm adalah sebagai berikut [8] :

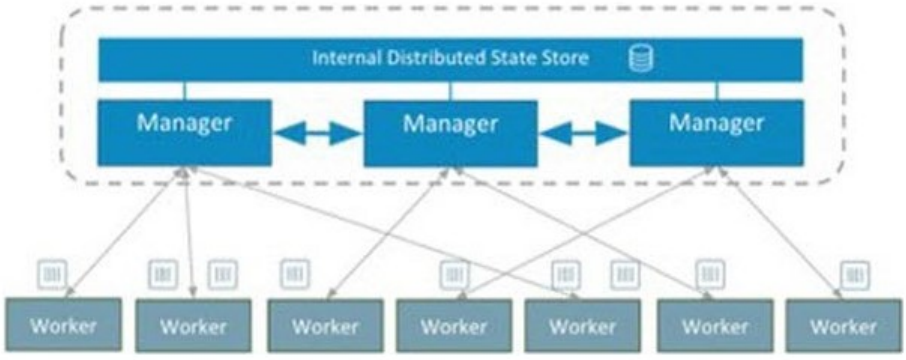
- Pengelolaan cluster yang terintegrasi dengan Engine Docker;
- Desain yang tidak tersentralisasi;
- Model layanan yang bersifat deklaratif;
- Pengubahan skalabilitas layanan;
- Rekonsiliasi status layanan yang diinginkan;
- Koneksitas untuk *multi-host*;
- Mekanisme untuk penemuan layanan;



Gambar 2.4: Komponen cluster Kubernetes [7]

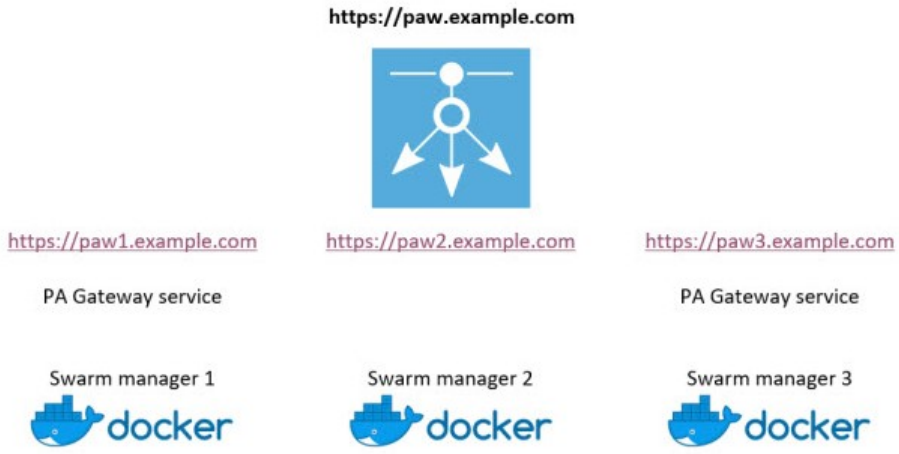
- Penyeimbangan beban kerja komputasi;
- Pengamanan *default* pada layanan;
- Mekanisme update secara berangsur-angsur.

Pengelolaan cluster dan orkestrasi Docker ditanamkan pada engine Docker menggunakan paket *swarmkit* yang merupakan implementasi lapisan orkestrasi Docker digunakan secara langsung oleh Docker itu sendiri. Suatu swarm terdiri dari banyak host Docker yang berjalan pada mode Swarm, bertindak sebagai manager terkait keanggotaan Docker dan delegasi Docker serta obyek pekerja Docker yang menjalankan layanan swarm. Pada saat seorang administrator Docker membuat suatu layanan, maka administrator tersebut perlu melakukan definisi kondisi optimal terkait dengan jumlah replika, interkoneksi jaringan dan sumber daya penyimpanan yang tersedia hingga melakukan porting layanan agar dapat diakses oleh pengguna melalui jaringan internet. Pembaca dapat melihat arsitektur Docker Swarm yang bersifat modular antara *worker* dengan manager, seperti terlihat pada gambar berikut ini : Pembaca juga dapat



Gambar 2.5: Arsitektur Docker Swarm [9]

melihat konfigurasi Swarm dengan bantuan planning analytics workspace distributed, seperti terlihat pada gambar berikut ini : Pada konfigurasi tersebut terdapat tiga manager cluster Swarm yang diletakkan dibelakang *load balancer*. Load balancer memiliki fungsi sebagai pengarah koneksi yang datang



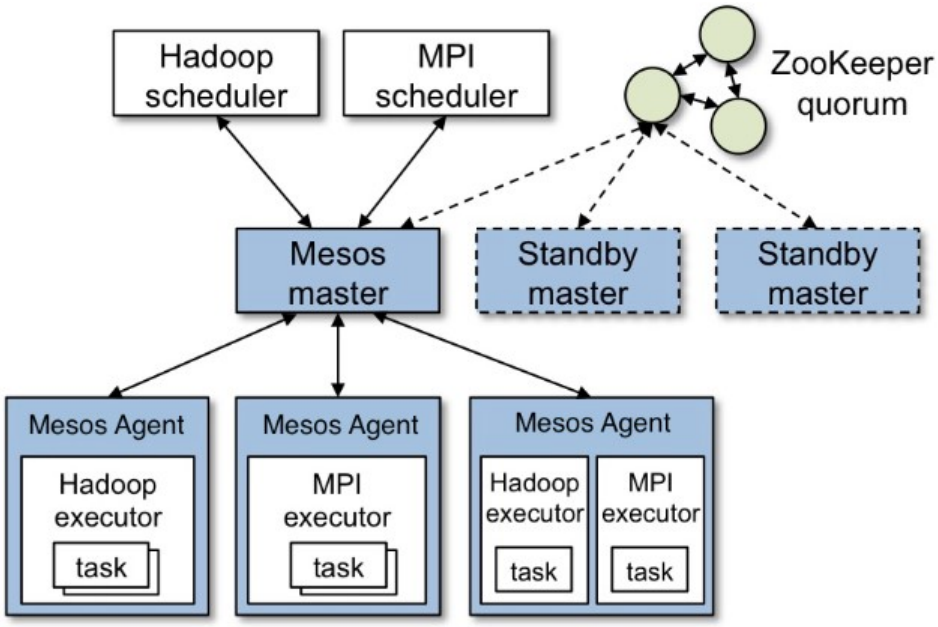
Gambar 2.6: Konfigurasi Docker Swarm [9]

pada link `https://paw.example.com` ke setiap node pada cluster. Walaupun obyek pada layanan PA Gateway tidak berjalan pada paw2.example.com, routing Docker Swarm akan meneruskan traffic kepada setiap node yang menjalankan layanan.

2.3 Apache Mesos

Apache Mesos (`https://https://mesos.apache.org/documentation/latest/architecture/`) merupakan suatu platform untuk berbagi pakai beragam *framework* seperti Hadoop atau MPI, komputasi cluster pada suatu *data center* secara halus. Apache Mesos akan berbagi pakai sumber daya untuk membolehkan *framework* agar dapat mencapai lokalitas data dengan cara melakukan pembacaan data yang tersimpan pada setiap mesin [10]. Apache Mesos menggunakan mekanisme penjadwalan dua tingkat terdistribusi yang disebut penawaran sumber daya. Apache Mesos akan memutuskan berapa banyak sumber daya untuk menawarkan setiap kerangka kerja, sementara kerangka kerja akan memutuskan sumber daya mana yang akan diterima dan mekanisme perhitungan sumber daya komputasi untuk dijalankan. Adapun arsitektur Apache Mesos dapat kita lihat pada gambar berikut ini : Apache Mesos terdiri dari *master daemon* yang melakukan pengelolaan *agent daemon* yang berjalan pada setiap *node cluster* berikut *framework* yang menjalankan tugas komputasi untuk setiap *agent*. MasterApache Mesos memungkinkan terjadinya berbagi pakai sumber daya komputasi (CPU, RAM) secara lebih halus diantara *framework* menggunakan mekanisme penawaran sumber daya. Master Apache Mesos menentukan seberapa banyak sumber daya komputasi yang ditawarkan kepada masing-masing framework terkait kebijakan organisasi, seperti berbagi pakai sumber daya komputasi secara adil atau penentuan prioritas secara ketat. *Framework* yang berjalan pada Mesos terdiri dari dua komponen yaitu: mekanisme penjadwalan untuk melakukan pendaftaran kepada *master agent* untuk mendapatkan sumber daya yang ditawarkan, dan proses pelaksana yang diluncurkan pada *node agen* untuk menjalankan tugas dari *framework*. Pada saat *master agent* menentukan berapa banyak sumber daya komputasi yang ditawarkan pada setiap *framework*, mekanisme penjadwalan *framework* memilih sumber daya mana yang ditawarkan untuk digunakan. Ketika *framework* menerima sumber daya yang ditawarkan, *framework* tersebut meneruskan deskripsi tugas yang ingin dijalankan pada Apache Mesos, sehingga pada akhirnya, Apache Mesos akan meluncurkan tugas kepada agen yang sesuai.
Beberapa keuntungan penggunaan Apache Mesos adalah sebagai berikut [10] :

- Apache Mesos dapat menjalankan berbagai macam obyek pada *framework* yang sama, dengan



Gambar 2.7: Arsitektur Apache Mesos [10]

menggunakan mekanisme pemisahan pekerjaan komputasi produksi dan eksperimental agar dapat menjalankan berbagai macam versi dari suatu *framework* secara bersamaan;

- Apache Mesos dapat dibangun untuk tujuan khusus framework dengan target area permasalahan secara spesifik agar mendapatkan kinerja komputasi yang lebih baik apabila dibandingkan dengan abstraksi untuk tujuan umum.

2.4 Keuntungan Penggunaan Container Orchestration

Container orchestration adalah proses mengotomatiskan penerapan, penskalaan, dan pengelolaan aplikasi dalam kontainer. Ini adalah cara untuk mengelola banyak wadah yang berjalan di satu host atau di beberapa host.

Ada beberapa manfaat menggunakan orkestrasi wadah, termasuk:

- Skalabilitas: *Container orchestration* dapat membantu Anda menaikkan atau menurunkan skala aplikasi sesuai kebutuhan. Ini karena container ringan dan mudah digunakan, sehingga Anda dapat menambahkan atau menghapusnya sesuai kebutuhan tanpa harus mengkhawatirkan infrastruktur yang mendasarinya.
- Ketersediaan: *Container orchestration* dapat membantu Anda meningkatkan ketersediaan aplikasi Anda. Hal ini karena dapat secara otomatis memulai ulang wadah yang gagal, sehingga aplikasi Anda tetap aktif dan berjalan meskipun ada masalah dengan masing-masing wadah.
- Efektivitas biaya: *Container orchestration* dapat membantu Anda menghemat uang untuk biaya infrastruktur TI Anda. Ini karena ini dapat membantu Anda mengoptimalkan penggunaan sumber daya Anda, sehingga Anda tidak perlu menyediakan infrastruktur secara berlebihan.

- Keamanan: *Container orchestration* dapat membantu Anda meningkatkan keamanan aplikasi Anda. Hal ini karena dapat membantu Anda menerapkan kebijakan keamanan di seluruh container, sehingga Anda dapat yakin bahwa aplikasi Anda aman.

Beberapa alat orkestrasi wadah paling populer termasuk Kubernetes, Docker Swarm, dan Mesos. Alat-alat ini menawarkan berbagai fitur, sehingga Anda dapat memilih salah satu yang paling sesuai dengan kebutuhan Anda.

Berikut adalah beberapa manfaat khusus menggunakan *Container orchestration*:

- Peningkatan keandalan: *Container orchestration* dapat membantu meningkatkan keandalan aplikasi dengan mengotomatiskan penerapan dan pengelolaan kontainer. Hal ini dapat membantu mengurangi jumlah kesalahan yang terjadi selama penerapan dan memastikan bahwa aplikasi selalu aktif dan berjalan.
- Peningkatan efisiensi: *Container orchestration* dapat membantu meningkatkan efisiensi aplikasi dengan mengoptimalkan penggunaan sumber daya. Ini dapat membantu mengurangi biaya menjalankan aplikasi dan meningkatkan kinerjanya.
- Keamanan yang ditingkatkan: *Container orchestration* dapat membantu meningkatkan keamanan aplikasi dengan menerapkan kebijakan keamanan di seluruh kontainer. Ini dapat membantu melindungi aplikasi dari akses tidak sah dan serangan jahat.
- Secara keseluruhan, *Container orchestration* dapat menawarkan sejumlah manfaat bagi organisasi yang ingin menerapkan dan mengelola aplikasi dalam container. Dengan mengotomatiskan penerapan dan pengelolaan kontainer, orkestrasi kontainer dapat membantu meningkatkan keandalan, efisiensi, dan keamanan aplikasi.

2.5 *Drawbacks* dari Container Orchestration

Beberapa kelemahan atau keterbatasan menggunakan orkestrasi wadah:

- Kompleksitas: *Container orchestration* bisa rumit untuk disiapkan dan dikelola. Ini karena membutuhkan pemahaman yang mendalam tentang teknologi wadah dan alat orkestrasi.
- Biaya: Orkestrasi wadah bisa mahal untuk diterapkan dan dipelihara. Ini karena memerlukan penggunaan perangkat keras dan perangkat lunak khusus.
- Penguncian vendor: Alat *Container orchestration* sering kali merupakan hak milik, yang dapat menyebabkan penguncian vendor. Ini berarti Anda mungkin terbatas untuk menggunakan alat dari satu vendor, yang dapat menyulitkan untuk beralih ke alat lain jika diperlukan.
- Satu titik kegagalan: Alat orkestrasi wadah dapat menjadi satu titik kegagalan. Artinya, jika alat orkestrasi gagal, alat ini dapat menghapus semua kontainer yang dikelolanya.
- Keamanan: Alat *Container orchestration* dapat menimbulkan risiko keamanan. Ini karena mereka memberi Anda kemampuan untuk mengelola kontainer dalam jumlah besar, yang dapat menyulitkan untuk melacak dan mengamankan semuanya.

- Secara keseluruhan, *Container orchestration* bisa menjadi solusi yang rumit dan mahal. Namun, ini dapat menawarkan sejumlah manfaat bagi organisasi yang ingin menerapkan dan mengelola aplikasi dalam container. Penting untuk mempertimbangkan keuntungan dan kerugian *Container orchestration* sebelum memutuskan apakah akan menggunakannya atau tidak.

Berikut beberapa pertimbangan tambahan saat menggunakan *Container orchestration*:

- Ukuran dan kerumitan aplikasi Anda: Jika Anda memiliki aplikasi kecil dan sederhana, Anda mungkin tidak perlu menggunakan *Container orchestration.* Namun, jika Anda memiliki aplikasi yang besar dan kompleks, orkestrasi wadah dapat membantu Anda mengelolanya dengan lebih efektif.

- Keahlian teknis Anda: Jika Anda memiliki banyak pengalaman dengan teknologi container, Anda mungkin dapat menyiapkan dan mengelola *Container orchestration* tanpa terlalu banyak kesulitan. Namun, jika Anda tidak terbiasa dengan teknologi kontainer, Anda mungkin perlu menyewa konsultan atau menggunakan layanan terkelola.

- Anggaran Anda: Orkestrasi kontainer bisa jadi mahal untuk diterapkan dan dipelihara. Jika Anda memiliki anggaran yang ketat, Anda mungkin ingin mempertimbangkan solusi yang berbeda.

- Pada akhirnya, keputusan apakah akan menggunakan orkestrasi wadah atau tidak bergantung pada kebutuhan dan persyaratan khusus Anda. Jika Anda mencari cara untuk menerapkan dan mengelola aplikasi dalam container secara lebih efektif, *Container orchestration* dapat menjadi pilihan yang baik. Namun, penting untuk mempertimbangkan keuntungan dan kerugian dari orkestrasi wadah sebelum membuat keputusan.

Bab 3

Kesimpulan

Kubernetes, Docker pada mode Swarm dan Apache Mesos dapat menangani cluster container Docker secara efisien dan mampu melakukan pengelolaan beragam layanan, walaupun pendekatan teknis antara ketiga platform orkestrasi tersebut memiliki perbedaan.

Beban overhead Kubernetes memberikan hasil yang cukup tinggi apabila dibandingkan dengan Docker pada mode Swarm yang memiliki beban overhead lebih rendah. Sedangkan pada Apache Mesos memberikan unjuk kerja yang tinggi pada mekanisme berbagi pakai sumber daya komputasi untuk beragam *platform*.

Pada hasil keseluruhan, kita dapat melihat kinerja dan overhead ketiga platform orkestrasi container tersebut, yaitu Kubernetes memiliki kinerja lebih rendah dibandingkan dengan Docker pada mode Swarm atau Apache Mesos.

Pada sisi lain, Kubernetes yang dijalankan pada implementasi desain kompleks menghasilkan sifat response yang lebih fleksibel dan memiliki kemampuan lebih baik apabila dibandingkan dengan Docker pada mode Swarm dan Apache Mesos, sehingga kedua *orchestration containerized platform* tersebut memiliki karakteristik unik masing-masing.

Beberapa hal yang masih dapat dikembangkan sebagai lanjutan penelitian ini meliputi hal terkait response berikut perilaku dari ketiga *orchestrator container platform* (Kubernetes, Docker pada mode Swarm dan Apache Moses) terhadap adanya overhead serta beban komputasi pada saat dilakukan *vulnerability assessment* dan aplikasi yang bersifat rumit. Aspek lainnya sebagai keberlanjutan penelitian ini adalah jaminan keamanan berikut kecepatan pulih pada ketiga *orchestrator container platform* setelah terjadi gangguan teknis.

Bibliography

[1] N. Naik, "Building a virtual system of systems using docker swarm in multiple clouds," in *2016 IEEE International Symposium on Systems Engineering (ISSE)*, 2016, pp. 1–3. 1, 3

[2] Y. Pan, I. Chen, F. Brasileiro, G. Jayaputera, and R. Sinnott, "A performance comparison of cloud-based container orchestration tools," in *2019 IEEE International Conference on Big Knowledge (ICBK)*, 2019, pp. 191–198. 1, 4

[3] Z. Zhong, M. Xu, M. A. Rodriguez, C. Xu, and R. Buyya, "Machine learning-based orchestration of containers: A taxonomy and future directions," *ACM Comput. Surv.*, vol. 54, no. 10s, sep 2022. [Online]. Available: https://doi.org/10.1145/3510415 3, 7, 8

[4] E. Casalicchio and S. Iannucci, "The state-of-the-art in container technologies: Application, orchestration and security. concurrency and computation: Practice and experience," *Special Issue: Special Issue on Computational Intelligence Techniques for Industrial and Medical Applications (CITIMA2018). Special Issue on Cloud Computing, IoT, and Big Data: Technologies and Applications (CloudTech17)*, vol. 32, no. 17, sep 2020. [Online]. Available: https://doi.org/10.1002/cpe.5668 3, 6, 7, 8

[5] N. Ferry, H. Song, A. Rossini, F. Chauvel, and A. Solberg, "Cloudmf: Applying mde to tame the complexity of managing multi-cloud applications," in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, 2014, pp. 269–277. 3

[6] C. Coleman, "What is container orchestration?" accessed July 25th, 2023. [Online]. Available: https://www.redhat.com/en/topics/containers/what-is-container-orchestration 9

[7] Unknown, "Overview of kubernetes," accessed July 25th, 2023. [Online]. Available: https://kubernetes.io/docs/concepts/overview/ 10, 11

[8] ——, "Swarm mode overview," accessed July 25th, 2023. [Online]. Available: https://docs.docker.com/engine/swarm/ 10

[9] ——, "Docker swarm architecture," accessed July 25th, 2023. [Online]. Available: https://www.ibm.com/docs/ru/planning-analytics/2.0.0?topic=swarm-docker-architecture 11, 12

[10] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," accessed July 25th, 2023. [Online]. Available: https://www.usenix.org/conference/nsdi11/mesos-platform-fine-grained-resource-sharing-data-center 12, 13