BAB III

METODE PENELITIAN

3.1 Jenis Penelitian

Sesuai dengan judul yang akan diajukan yaitu "Analisis Perbandingan Performa *Container* pada Kubernetes *Service* di *Platform* GCP, AWS, dan Azure" maka metode penelitian yang akan digunakan adalah *Research and Development* (RnD) dengan metodologi umum *System Development Life Cycle* (SDLC). Model SDLC yang dipakai pada penelitian ini adalah metode *waterfall*.

Menurut Nusa Putra pada tahun 2015, *Research and Development* (RnD) ialah suatu proses atau langkah-langkah untuk mengembangkan suatu produk baru atau menyempurnakan suatu produk yang telah ada sehingga dapat dipertanggungjawabkan. Sedangkan menurut Sugiyono pada tahun 2016, RnD ialah metode penelitian yang digunakan untuk menghasilkan produk tertentu dan menguji keefektifan produk tersebut. Sehingga dapat disimpulkan bahwa RnD ialah suatu metode penelitian yang menjelaskan suatu proses untuk menghasilkan maupun mengembangkan suatu produk melalui uji keefektifan agar dapat dipertanggungjawabkan.

SDLC merupakan metodologi yang pertama kali diperkenalkan oleh Meir Manny Lehman pada tahun 1969. Beliau membuat pendekatan dasar dengan memisahkan desain, evaluasi, dan dokumentasi dalam suatu penelitian mendesain suatu *software*. SDLC dapat dibagi menjadi dua tipe generik. Pertama, tipe model *waterfall*, hal ini dikarenakan pada tahun 1970 Winston Royce menguraikan SDLC dengan tahapan model berurutan yang diurutkan kebawah seperti aliran air terjun. Model *waterfall* akan menyajikan strategi ideal dalam membangun sebuah proyek dengan menguraikan beberapa prinsip praktik seperti desain, *coding*, dokumentasi dari setiap tahap dan perencanaan yang tepat. Maka model *waterfall* akan menggambarkan pengembangan urutan tahapan yang dapat diringkas dalam lima langkah yaitu analisis, desain, implementasi, *testing*, *deployment* dan *maintenance* (Isaias & Issa, 2015). Selain model *waterfall*, Model SDLC yang memiliki langkah

serupa adalah model *prototyping*. Namun model *waterfall* dan *prototyping* tentu memiliki perbedaan seperti yang tertera pada Tabel 3.1

Tabel 3. 1
Perbandingan Model *Waterfall* dan *Prototype*

Tahapan Development	***Waterfall***	***Prototype***
System Planning	Berdasarkan Kebutuhan	Berdasarkan kebutuhan
System Analysis	Kebutuhan data harus dianalisis diawal dengan lengkap dan menyeluruh	Kebutuhan data dapat ditambah ataupun dikurangi sesuai kebutuhan *user* ketika dilakukan *Testing*
	Perubahan data maupun fungsional akan merubah keseluruhan proses pada tahapan berikutnya	Perubahan dapat dilakukan selama sistem atau *software* masih dalam bentuk *prototype*
System Desain	*Testing* dilakukan ketika semua tahapan pada model sudah selesai	*Testing* dapat dilakukan ketika *prototype* telah dibangun sehingga hasil *Testing* dapat merubah rancangan sistem
	Tidak dapat memberikan gambaran jelas mengenai sistem yang dibangun karena sistem bisa dilihat jika tahapan sudah dilakukan	Memberikan *prototype* sebagai gambaran sistem yang akan dibangun sehingga *user* dapat melihat dan berinteraksi langsung dengan gambaran sistem
		User berperan aktif dalam pengembangan sistem

		Sistem yang dibangun akan sesuai dengan keinginan *user*
System Implementation	Menerapkan proses perancangan yang baik	Tidak menerapkan proses perancangan yang baik
	Evaluasi dilakukan ketika sistem telah dibangun	Evaluasi dilakukan ketika *prototype* telah dibangun
	Mengedepankan kebutuhan fungsional sistem	Mengedepankan aspek kenyamanan *user*
System Maintenance	Dilakukan sesuai kesepakatan	Dilakukan sesuai kesepakatan

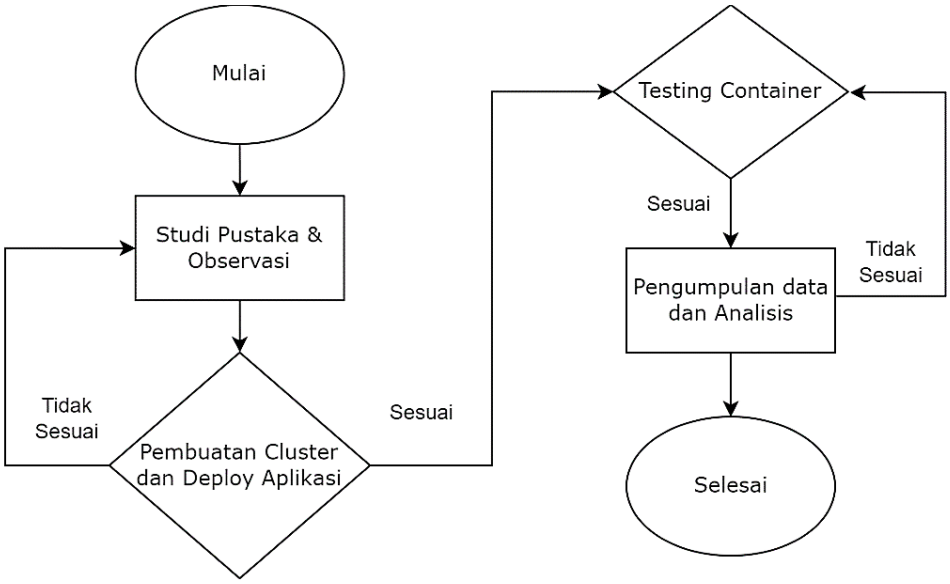
Tipe kedua SDLC terdiri dari model tipe inkremental. Model inkremental bertentangan dengan prinsip *waterfall* dalam mengembangkan sistem dalam proses *singlepass*, dengan dokumentasi yang ketat dan tahap pengujian yang ekstensif, untuk menghasilkan produk akhir yang dapat digunakan sepenuhnya pada akhirnya. Model inkremental malah mengusulkan pengembangan sistem dalam proses *build* atau inkremen yang berurutan. Dengan setiap *build*, sistem dirancang dan dikembangkan, dan versi kerja atau *prototype* diimplementasikan. *User* kemudian dapat mengujinya secara aktif, dalam konteks kerja, dan memberikan umpan balik yang berharga. Umpan balik ini kemudian akan digunakan sebagai titik awal untuk membangun berikutnya. Dengan setiap *build* yang berurutan, sistem menjadi lebih lengkap, lebih fungsional, dan lebih dekat dengan apa yang diinginkan *user* (Massey & Satao, 2012).

3.2 Desain Penelitian

Penelitian ini akan berfokus pada evaluasi performa dari *container* yang berisikan *image* nginx:1.23 yang berjalan pada beberapa *cloud service* berbasis Kubernetes seperti Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes

*Service*s (EKS), dan Azure Kubernetes *Service* (AKS). Untuk mendapatkan hasil yang diinginkan penulis akan menerapkan metode SLDC dengan model *waterfall*.

Alur Penelitian ini akan dimulai dengan Studi Pustaka dan Observasi dilanjutkan dengan pembuatan *cluster* dan *deploy* aplikasi. Setelah proses *deployment* selesai *container* akan di-*testing* dan penulis akan mengumpulkan data dari hasil testing tersebut lalu menganalisis data untuk dibandingkan performanya sebagaimana *flowchart* dari desain alur penelitian pada Gambar 3.1



Gambar 3. 1
Desain Alur Penelitian

3.3 Teknik Pengumpulan Data

3.3.1 Studi Pustaka

Pada tahap ini pengumpulan data dilakukan dengan meriset sumber-sumber bacaan seperti buku, dokumen, maupun jurnal publikasi dari berbagai sumber. Sumber bacaan pengetahuan dapat dijadikan landasan teori untuk menganalisis performa *container* pada Kubernetes *service* di *Platform* GCP, AWS, dan Azure.

3.3.2 Observasi

Observasi atau pengamatan langsung *container* yang sedang berjalan akan dilakukan guna mendapatkan data yang diinginkan.

3.4 Teknik Analisis Data

Sebagaimana model penelitian yang akan diterapkan pada penelitian ini maka teknik analisis data yang digunakan ialah sebagai berikut,

3.4.1 Analisis Kebutuhan

Untuk melakukan penelitian ini ada beberapa hal yang harus disiapkan seperti Laptop atau PC, Akun GCP, AWS, serta Azure, dan Internet. Penulis menggunakan laptop Lenovo dengan spesifikasi yang terangkum pada Tabel 3.2 berikut

Tabel 3. 2
Spesifikasi Laptop yang digunakan

OS	Processor	CPU	*Memory*
Windows 10 Home Single Language 64-bit	AMD 3020e with Radeon Graphics	2 CPUs 1,2GHz	8192MB RAM

Sedangkan untuk penelitian ini, penulis akan membandingkan ketiga CSPs dengan spesifikasi mesin yang sama yaitu terdiri dari 4 buah CPU dan RAM sebesar 16 GB yang terangkum pada Tabel 3.3 berikut

Tabel 3. 3
Spesifikasi *Instance Machine* yang digunakan

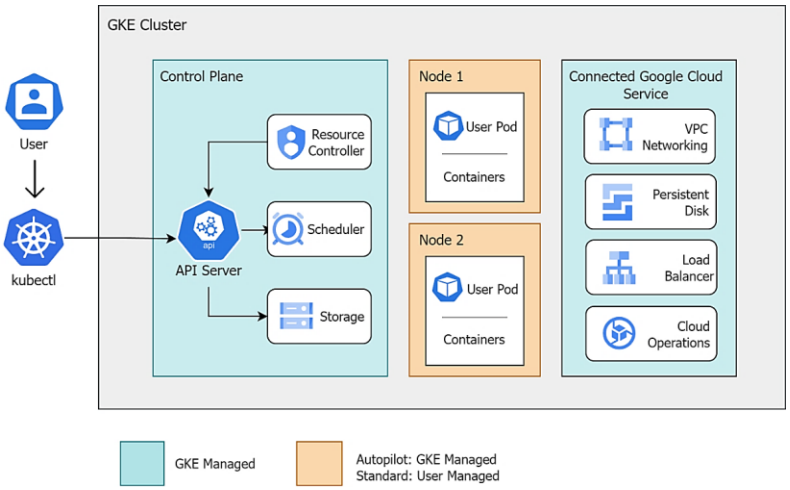
Cloud	*Instance* type	*Instance* size	OS	CPU	vCPUs	RAM	Harga Per jam	*Network*	K8s ver	Location
GKE	*General Purpose*	e2-standard-4	*Container-Optimized OS*	Intel Xeon CPU @ 2,20GHz	4	16GB	$0,134	*Network Standard Service Tier*	1.27.2-gke.1200	Us-central1
EKS	*General Purpose*	m5.xlarge	Amazon Linux 2	Intel Xeon CPU @ 2,50GHz	4	16GB	$0,192	*Low to moderate*	1.27	US-East-1
AKS	*General Purpose*	D4 v3	Linux	Intel Xeon CPU E5-2673 v4 @ 2,30GHz	4	16GB	$0,192	*Standard*	1.27.1	East US

3.4.2 Desain Sistem

Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes Service (EKS), dan Azure Kubernetes Service (AKS) merupakan layanan terkelola yang mendukung orkestrasi Kubernetes. Meskipun sama-sama dibangun diatas *Cloud Provider*, GKE, EKS, dan AKS memiliki arsitekturnya sendiri yang disesuaikan dengan infrastruktur masing-masing penyedia *cloud*. Berikut merupakan deskripsi dari masing-masing arsitektur layanan.

3.4.2.1 Arsitektur GKE

GKE dibangun diatas layanan terkola Google Cloud oleh karena itu GKE memiliki arsitektur yang memanfaatkan infrastruktur dan layanan Google seperti pada Gambar 3.2 berikut



Gambar 3. 2
Arsitektur GKE

1. ***Control Plane***

GKE mengabstaksi detail dari *Master Node* dan Google akan mengelola komponen dari *control plane* seperti API *server*, etcd, *scheduler*, dan *controller manager*. Hal ini dilakukan untuk memastikan *high availability* dan *upgrading* yang mulus.

2. ***Node Pools***

Cluster GKE terdiri dari kumpulan *node* yang merupakan grup VM *instance*. *Node* menjalankan komponen Kubernetes yang diperlukan dan GKE memastikan

bahwa *node* tersebut dalam keadaan sehat (*healthy*), *scaled*, dan didistribusikan ke seluruh zona yang tersedia.

3. *Managed Instance Groups*

GKE menggunakan *Managed Instance Groups* (MIGs) untuk menangani manajemen *node*, *health check*, penskalaan, dan pembaharuan. Hal ini memberikan keandalan (*reliability*) dan kemampuan untuk melakukan *auto-scaling.*

4. Google *Cloud Load Balancing*

GKE terintegrasi dengan *Load Balancing* Google Cloud untuk mendistribusikan lalu lintas antar *nodes* sehingga meningkatkan *availability* dan performa aplikasi.

5. Google *Cloud Networking*

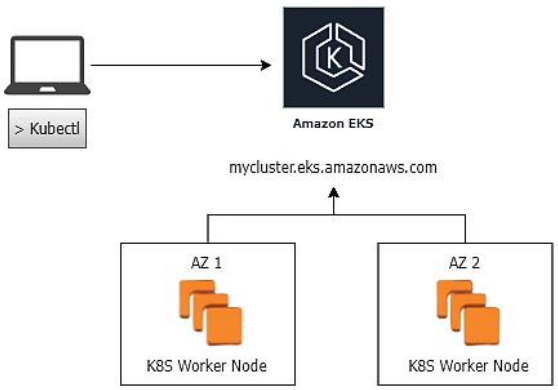
GKE terintegrasi dengan *Virtual Private Cloud* (VPC) Google Cloud untuk *networking*, memastikan keamanan dan komunikasi *private* dalam *cluster.*

6. Google *Cloud Identity and Access Management (IAM)*

GKE menggunakan IAM untuk *access control* dan *authorization*. GKE akan memastikan bahwa hanya *user* dan *services* yang terotorisasi yang bisa berinteraksi dengan *cluster.*

3.4.2.2 Arsitektur EKS

EKS merupakan layanan Kubernetes yang dikelola AWS. EKS dirancang untuk menyediakan *platform* Kubernetes yang *scalable* dan *highly available.* EKS terdiri dari beberapa komponen untuk menunjang berjalannya cluster dan aplikasi. Gambar 3.3 berikut merupakan arsitektur EKS yang terdiri dari dua *nodes* yang terhubung dengan Kubernetes *cluster* yang dikelola AWS.



Gambar 3. 3
Arsitektur EKS

1. ***Control Plane***

EKS mengisolasi manajemen *control plane* Kubernetes termasuk API *Server,* etcd, *scheduler* dan *control manager*. AWS memastikan *availability*, *security,* dan *upgrade* dari *control plane.*

2. ***Node Groups***

Node Group yang berada dalam *cluster* EKS diatur oleh Amazon EC2. Setiap *node group* dapat memiliki jenis dan ukuran *instance* yang berbeda dan AWS *auto-scaling* dapat mengelola penskalaan group ini.

3. ***Managed Node Groups***

AWS mengelola *Node Group* terkelola yang memungkinkan *user* menentukan kebijakan penskalaan, *rolling updates*, dan *versioning nodes.*

4. **AWS *Load Balancer***

EKS terintegrasi dengan layanan AWS *Load Balancing* seperti *Application Load Balancer* dan *Network Load Balancer* untuk mendistribusikan lalu lintas.

5. **Amazon VPC *Networking***

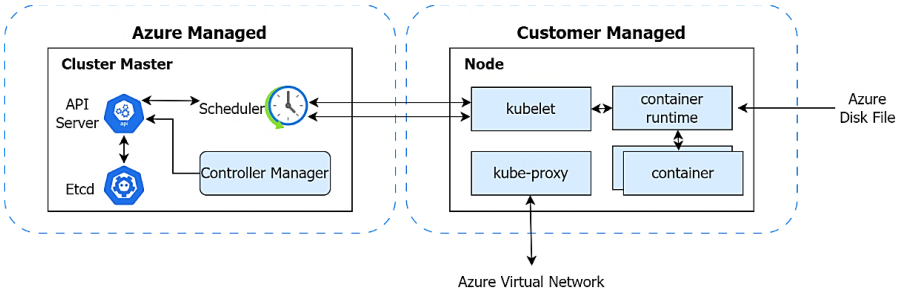
Cluster EKS di-*deploy* didalam Amazon *Virtual Private Cloud* (VPC) yang memungkinkan *user* menyesuaikan konfigurasi jaringan dan pengaturan keamanan.

6. **AWS *Identity and Access Management (IAM)***

IAM digunakan untuk mengatur *access control* ke EKS *cluster* sekaligus memastikan autentikasi dan otorisasi yang aman.

3.4.2.3 Arsitektur AKS

AKS merupakan layanan Kubernetes terkelola milik Microsoft Azure. AKS berfokus pada penyederhanaan *deployment* dan *operation* Kubernetes. AKS terdiri dari beberapa komponen untuk keberlangsungan hidup cluster dan aplikasi yang dapat dilihat pada Gambar 3.4 berikut



Gambar 3. 4
Arsitektur AKS

1. ***Control Plane***

Administasi *control plane* diabstaksikan oleh AKS mencakup API *server*, etcd, *scheduler* dan *controller manager*. Microsoft bertanggung jawab atas *availability*, *security*, dan *upgrades* dari *control plane.*

2. ***Node Pools***

Cluster AKS memanfaatkan *node pools* yang terdiri dari *Azure Virtual Machines* (VM). *User* dapat membuat banyak *node pools* dengan berbagai ukuran VM dan opsi *auto-scaling.*

3. ***Managed Node Pools***

AKS menangani *scaling* dan *health* dari *node pools* menggunkaan teknologi *auto-scaling* dan Azure *Virtual Machine Scale Sets* (VMSS).

4. **Azure *Node Balancers***

AKS terhubung ke Azure *Load Balancers* untuk mendistribusikan *traffic* yang masuk.

5. **Azure *Virtual Network***

Cluster AKS di-*deploy* didalam sebuah Azure *Virtual Network* yang menyediakan *network isolation* dan fitur *networking* yang bisa disesuaikan.

6. **Azure *Active Directory (AAD)***

AKS menggunakan Azure AD untuk manajemen identitas dan akses yang memastikan autentifikasi dan otorisasi yang aman.

3.4.3 Implementasi Desain

Setelah mengetahui desain sistem atau arsitektur dari masing-masing CSPs, Penulis akan mengimplementasikan dengan membuat *cluster* terlebih dahulu dilanjutkan dengan *deploy* aplikasi.

3.4.4 *Integration* dan *Testing* Sistem

3.4.4.1 *Benchmarking*

Benchmark merupakan sebuah tolak ukur atau poin referensi yang digunakan untuk mengukur sesuatu.

1. **Sysbench**

SysBench merupakan salah satu *Tools* yang digunakan untuk *Benchmarking*. SysBench paling sering digunakan untuk mencari tolak ukur *database* tetapi bisa

juga digunakan untuk membuat *arbitrarily complex workloads* yang tidak melibatkan *server database* (Kopytov, 2023). Tabel 3.4 merupakan parameter pengujian yang dapat SysBench lakukan.

Tabel 3. 4

SysBench *Test Profile*

Test ***Profile***	**Deskripsi**
oltp_*.lua	Sebuah koleksi dari OLTP seperti tolak ukur database
fileio	Sebuah tolak ukur *filesystem-level*
cpu	Sebuah tolak ukur cpu sederhana
memory	Sebuah tolak ukur akses memori
threads	Sebuah tolak ukur *threads* berbasis *scheduler*
mutex	Sebuah tolak ukur POSIX mutex

2. Kube-Bench

Kube-Bench merupakan sebuah alat yang memeriksa apakah Kubernetes di-*deploy* dengan aman dengan menjalankan pemeriksaan yang didokumentasikan dalam CIS Kubernetes *benchmark* (Rice, 2023).

3.4.4.2 *Observability*

Observability merupakan ukuran seberapa baik keadaan internal suatu sistem yang dapat disimpulkan dari pengetahuan mengenai *output* eksternalnya.

1. Prometheus

Prometheus merupakan sebuah sistem dan sistem *monitoring service*. Prometheus akan mengumpulkan *metrics* dari target yang dikonfigurasi pada

interval tertentu, mengevaluasi *rule expressions*, menampilkan hasil dan bisa men-*trigger alert* ketika suatu kondisi tertentu diamati (Reinartz, 2023).

2. Grafana

Grafana merupakan sebuah *platform* visualisasi data yang dikembangkan oleh perusahaan Bernama Grafana Labs. Grafana memungkinkan *user* untuk melihat data via *chart* maupun grafik yang dapat disatukan kedalam satu *dashboard* (RedHat, 2022).

3.4.5 *Operation* dan *Maintenance*

Dikarenakan *scenario* pertama yaitu melakukan *benchmarking* mengalami kendala dan kegagalan pada saat proses *deployment*, maka penulis memutuskan untuk mengambil data *performance* dari ketiga CSPs secara *real time* dengan bantuan Prometheus dan akan divisualisasikan dengan bantuan Grafana. Setelah mendapatkan data yang dibutuhkan dan dikarenakan adanya beban pembiayaan yang terus bertambah seiring terus berjalannya mesin yang yang dibutuhkan pada *cluster*, maka *cluster* yang telah digunakan akan dihapus.

3.5 Waktu dan Tempat Penelitian

3.5.1 Waktu Penelitian

Penelitian akan dilakukan pada Januari – Agustus 2023.

3.5.2 Tempat Penelitian

Penelitian ini dapat dilakukan dimana saja dengan syarat terhubung dengan internet untuk mengakses layanan GCP, AWS, dan Azure.