BAB 1
PENDAHULUAN

1.1 Latar Belakang

Container adalah perkembangan teknologi virtualisasi yang banyak digunakan untuk mengembangkan dan mengirimkan perangkat lunak untuk layanan *cloud computing* [1]. *Container* bersifat portabel, ringan, dan mudah dikelola siklus hidupnya dan telah menjadi alternatif dari Virtual Mesin (VM). Teknologi *container* memiliki peran utama dalam perubahan ini yang memungkinkan untuk berbagi sumber daya *host* secara efisien [2]. *Container* memungkinkan untuk berbagi sumber daya *hardware* secara langsung di server dengan efisien diantara penyewa *cloud*, sehingga container lebih ringan dan lebih cepat daripada virtual mesin [1]. *Host* dengan jumlah *container* yang banyak menyebabkan penurunan performansi yang signifikan, baik container yang dipasangkan pada lingkungan penyewa *cloud* maupun pada *local* [2]. Penurunan performansi *container* disebabkan karena *container runtime* bertanggung jawab menjalankan *container* pada sistem operasi *host*.

Kubernetes orkestrator *container* yang banyak digunakan mendukung setiap *container runtime* yang mengimplementasikan *Container Runtime Interface* (CRI) [2]. Dalam arsitektur *container*, *container runtime* bertanggung jawab untuk memuat *image container* dari repositori, memantau sumber daya sistem lokal, mengisolasi sumber daya sistem untuk penggunaan *container*, dan mengelola siklus hidup *container* [3]. Pemilihan *container runtime* menjadi sangat krusial untuk mengimbangi pertambahan jumlah *container* yang digunakan dan juga setiap runtime memiliki karakteristik berbeda, sehingga kinerja dan penggunaan sumber daya juga berbeda [1].

Pada penelitian ini berfokus menganalisis performansi dari tiga *container runtime* yaitu Containerd, CRI-O, dan Kata Containers pada orkestrasi Kubernetes. Penggunaan ketiga *container runtime* pada penelitian ini dipilih berdasarkan kajian Pustaka dan beberapa penelitian lain. Pemilihan *Containerd* mewakili *Low-level* container runtime karena *Containerd* tersedia *open-source*, memiliki fasilitas pengelolaan siklus hidup container melalui *Application Programming Interface*

(API) *request*, serta dapat meningkatkan portabilitas container. Pemilihan CRI-O mewakili *High-level container runtime* karena CRI-O merupakan implementasi *open-source* dari *container runtime interface* Kubernetes. Pemilihan Kata Containers mewakili *Low-level container runtime* karena *container runtime open-source* yang memberikan isolasi beban kerja yang lebih kuat menggunakan perangkat keras teknologi virtualisasi sebagai lapisan pertahanan kedua. Skenario penelitian ini menggunakan skalabilitas dengan jumlah *container* yang berbeda yaitu 10, 20, dan 40 *container* pada Containerd, CRI-O, dan Kata Containers. Parameter yang akan dianalisis yaitu performansi dari CPU, *memory*, *throughput*, dan *latency* dengan menggunakan *tools* ping, netperf, y-cruncher, dan STREAM.

1.2 Rumusan Masalah

Rumusan masalah dari penelitian ini adalah:

1) Bagaimana performansi *container runtime* yaitu *Containerd*, CRI-O, dan Kata *Containers* pada orkestrasi Kubernetes berdasarkan parameter hardware (CPU dan *memory*) dan parameter jaringan (*throughput* dan *latency*)?
2) Bagaimana pengaruh skalabilitas *container* menggunakan *container runtime* yaitu *Containerd*, CRI-O, dan Kata *Containers* pada orkestrasi Kubernetes berdasarkan parameter hardware (CPU dan *memory*) dan parameter jaringan (*throughput* dan *latency*)?

1.3 Batasan Masalah

Batasan masalah dari penelitian ini adalah:

1) Penelitian ini menguji *container runtime* yaitu *Containerd*, CRI-O, dan Kata *Containers*.
2) Pengujian dilakukan pada orkestrasi Kubernetes.
3) Server yang digunakan pada Kubernetes terdiri dari 1 master node dan 3 worker node.
4) Parameter yang dianalisis yaitu performansi *hardware* (CPU dan *memory*) dan performansi *network* (*throughput* dan *latency*).
5) Pengujian dilakukan menggunakan *Google Cloud Platform*.

6) Pengambilan data mengunakan aplikasi yaitu ping, netperf, y-cruncher, dan STREAM.

1.4 Tujuan

Tujuan dari penelitian ini adalah:

1) Menganalisis performansi *container runtime* yaitu Containerd, CRI-O, dan Kata Containers pada orkestrasi Kubernetes berdasarkan parameter hardware (CPU dan *memory*) dan parameter jaringan (*throughput* dan latency).
2) Menganalisis pengaruh skalabilitas *container* menggunakan *container runtime* yaitu *Containerd*, CRI-O, dan Kata *Containers* pada orkestrasi Kubernetes berdasarkan parameter hardware (CPU dan *memory*) dan parameter jaringan (*throughput* dan *latency*).

1.5 Manfaat

Penelitian ini diharapkan dapat memberikan gambaran mengenai kinerja dari *container runtime* menggunakan *Containerd*, CRI-O, dan Kata *Containers* orkestrasi pada *Kubernetes*. Dengan mengetahui pengaruh penggunaan pertambaha jumlah *container* pada *container runtime* yang berbeda-beda diharapkan dalam implementasinya dapat memberikan informasi performansi *container runtime*.

1.6 Sistematika Penulisan

Sistematika penulisan penelitian ini dibagi manjadi 3 bagian:

1. BAB 1 : PENDAHULUAN

   Bagian pendahuluan berisi mengenai latar belakang, rumusan masalah yang diangkat, manfaat dan tujuan penelitian.
2. BAB 2 : DASAR TEORI

   Pada bagian ini membahas tentang referensi penelitian sebelumnya, landasan teori mengenai konsep *container*, orkestrasi Kubernetes, *Container Runtime*, serta parameter *benchmark*.
3. BAB 3 : METODE PENELITIAN

Pada bagian ini membahas mengenai alat dan bahan yang digunakan dalam penelitian, alur penelitian meliputi: parameter simulasi, pemodelan sistem, parameter *benchmark*, serta pengambilan data.

4. BAB 4 : HASIL DAN PEMBAHASAN

   Pada bagian ini hasil penelitian dan analisis data berdasarkan hasil percobaan.

5. BAB 5 : KESIMPULAN DAN SARAN

   Pada bagian ini membahas mengenai kesimpulan penelitian dan saran pengembangan untuk kedepannya.