BAB 1 PENDAHULUAN

1.1. Latar Belakang

Penggunaan teknologi informasi dalam pertukaran informasi semakin marak digunakan. Hal tersebut tidak lepas dari penyediaan layanan aplikasi yang semakin menyebar di berbagai lini kehidupan masyarakat. Perusahaan penyedia layanan aplikasi pun dituntut untuk menyediakan aplikasi yang mampu memberikan layanan secara optimal kepada pengguna. Teknologi kontainerisasi pun muncul sebagai salah satu solusi untuk mengelola aplikasi dengan lebih cepat dan mudah [1]. Kontainer atau *container* merupakan teknologi yang menyediakan lingkungan terisolasi bagi aplikasi dengan menyediakan independensi serta fleksibilitas dalam pengembangan aplikasi. Sifat independensi pada *container* berhasil mengatasi permasalahan inkonsistensi ketika sebuah aplikasi dikembangkan pada lingkungan yang berbeda. Penggunaan kontainerisasi saat pengembangan aplikasi dapat memudahkan pengembang aplikasi ketika hendak melakukan pemindahan aplikasi ke lingkungan produksi [2]. Menurut data dari laporan International Business Machines Corporation atau IBM pada Maret 2020, sebanyak 45% hingga 59% pengembang aplikasi merekomendasikan sebuah aplikasi untuk dikontainerisasikan dan 37% perusahaan besar di dunia telah menggunakan *container* pada aplikasi mereka [3]. Dari survei yang diadakan oleh organisasi Cloud Native Computing Foundation atau CNCF yang menaungi pengembangan teknologi *cloud native* memaparkan bahwa penggunaan *container* di lingkungan produksi telah meningkat 92% dibanding tahun 2015 yaitu 84% dan meningkat cukup jauh sebanyak 300% sejak survei CNCF pada tahun 2016 [4].

Dalam kerangka kerjanya, sebuah *container* memerlukan komponen inti yang kerap disebut *container runtime*. *Container runtime* merupakan komponen yang bertugas untuk membuat, menjalankan, hingga mengatur

system calls dari *container* ke *host kernel*. *Container* dapat bekerja dengan cepat saat menjalankan sebuah aplikasi dikarenakan *container* tidak memiliki dependensi tambahan dari sistem operasi melainkan hanya terdiri dari dependensi untuk kebutuhan sistem aplikasi yang dimuatnya saja. Oleh karena itu, pemrosesan sistem di dalam *container* menggunakan konsep berbagi kernel (*sharing kernel* atau *host shared kernel*) dengan kernel milik *host* yang menaungi *container* tersebut. *Container runtime* berperan sebagai jembatan antara sistem internal *container* dengan *host kernel* tersebut sehingga peran *container runtime* cukup penting bagi *container* [5].

Penggunaan *container* yang semakin banyak digunakan oleh perusahaan juga mengundang kewaspadaan terkait masalah keamanan *container* itu sendiri. Hal ini pun berkaitan erat dengan isu keamanan pada *container runtime* sebagai komponen penting dari sebuah *container*. Salah satu jenis serangan yang umum menyerang sistem aplikasi yaitu serangan *Denial of Service* (DOS). Berdasarkan laporan Microsoft Cloud pada tahun 2022, tercatat rerata 1435 serangan DOS secara terdistribusi per hari dengan lebih dari 63% serangannya terjadi menggunakan metode serangan *TCP Flood* dan 22% menggunakan *UDP Flood* [6]. *Denial of Service* atau DOS merupakan jenis serangan yang menyerang sistem dengan mengirim paket data secara terus menerus ke satu sistem target hingga sistem target tidak dapat memenuhi permintaan pengguna. Umumnya, serangan DOS membanjiri target dengan paket data TCP SYN secara terus menerus [7]. Dengan *container runtime* sebagai komponen utama *container*, menjadikannya tumpuan utama ketika terkena serangan DOS.

Dewasa ini, terdapat banyak jenis *container runtime* yang dapat digunakan dengan berbagai keunggulannya. Salah dua *container runtime* yang populer dengan keunggulan arsitektur keamanannya yaitu Kata Containers dan gVisor. Kata Containers memiliki arsitektur yang memanfaatkan *hypervisor* seperti QEMU/KVM guna menyediakan lingkungan terisolasi dengan performa mumpuni [5]. Sementara gVisor merupakan

lingkungan *sandbox* yang memanfaatkan area terisolasi bernama *user space* untuk membatasi akses langsung dari *host* ke *container* [8].

Pada penelitian ini penulis menggunakan *container runtime* Kata Containers dan gVisor untuk menjalankan sebuah *container* dan menganalisa performanya berdasarkan kinerja *Central Processing Unit* atau CPU, memori, *web response time*, dan *throughput* ketika terkena serangan DOS. Selain kedua *container runtime* tersebut, penulis juga menggunakan *container runtime default* yang digunakan oleh Docker yaitu runC sebagai pembanding antara *container runtime* dengan tambahan arsitektur keamanan seperti Kata Containers dan gVisor dengan *container runtime* yang umum digunakan. Docker sendiri merupakan perangkat lunak yang populer digunakan untuk membuat sebuah aplikasi dalam bentuk *container* dengan memanfaatkan *container runtime* runC [9], [10]. Semakin meningkatnya penggunaan *container* dalam industri teknologi menjadikan penelitian ini penting untuk dilakukan guna menakar seberapa besar pengaruh serangan DOS terhadap performa *container* yang menggunakan *container runtime* runC, gVisor, maupun Kata Container. Hasil perbandingan tersebut diharapkan dapat memberikan pandangan pada dunia industri teknologi yang ingin menggunakan *container runtime* dengan tepat guna mengurangi dampak serangan DOS. Berdasar dengan pemaparan tersebut, penulis mengangkat topik penelitian berjudul "Analisis Performansi *Container Runtime* RunC, gVisor, dan Kata Containers Terhadap Serangan *Denial of Service* (DOS)" sebagai tugas akhir.

1.2. Rumusan Masalah

Berdasarkan latar belakang yang telah dipaparkan sebelumnya, terdapat rumusan masalah yang berkaitan dengan penelitian ini sebagai berikut:

- Bagaimana perbedaan performa antar *container* yang menggunakan *container runtime* runC, gVisor, dan Kata Container ketika terkena serangan DOS TCP dan UDP Flood?
- Bagaimana pengaruh serangan DOS TCP dan UDP Flood terhadap performa layanan web pada *container runtime* runC, gVisor, dan Kata Container?
- Apakah penggunaan *container runtime* runC, gVisor, dan Kata Container mempengaruhi performa *host server* ketika terkena serangan DOS TCP dan UDP Flood?

1.3. Tujuan dan Manfaat

Penelitian ini dilakukan dengan tujuan untuk menganalisis perbedaan performa *container* dengan *runtime* runC, gVisor, dan Kata Containers sebelum serta ketika diserang dengan *Denial of Service*. Adapun, manfaat yang dapat diperoleh dari penelitian ini sebagai referensi bagi dunia industri teknologi dalam menentukan *container runtime* yang tepat sesuai dengan kebutuhan.

1.4. Batasan Masalah

Penelitian tugas akhir ini memiliki batasan masalah yang digunakan dalam membatasi cakupan penelitian, sebagai berikut:

- Aplikasi kontainer dalam pengujian merupakan *web server* Nginx.
- Pengujian serangan DOS dilakukan dengan tipe serangan *TCP Flood* dan *UDP Flood*.

1.5. Metode Penelitian

Pekerjaan dalam penelitian ini dilakukan menggunakan metode serangan DOS selama proses pengujian. Penelitian dilakukan dengan menganalisa setiap metriks data secara kuantitatif dalam proses eksperimental berdasarkan skenario pengujian yang telah ditentukan.