BAB I

PENDAHULUAN

1.1 Latar Belakang

Sebuah perangkat lunak sebelum digunakan oleh pengguna secara luas perlu dilakukan proses deployment pada perangkat lunak tersebut. Deployment sebuah perangkat lunak dapat didefinisikan sebagai akusisi dan eksekusi sebuah perangkat lunak. Proses ini biasanya dilakukan oleh seorang software deployer atau dalam bahasa yang sering digunakan belakangan ini seorang SRE (site reliability engineer) [1]. Maka dari itu dapat dikatan deployment adalah aktivitas post-production sebuah perangkat lunak untuk digunakan oleh konsumen. Proses deployment sebuah perangkat lunak terdiri dari beberapa proses yang saling berhubungan seperti proses release sebuah perangkat lunak, instalasi perangkat lunak kedalam environment exectuion, dan aktivasi sebuah software [2].

Untuk mendeploy sebuah sistem perangkat lunak juga ada beberapa yang harus diperhatikan antara lain sub-komponen yang dibutuhkan atau package external, resource (hardware). Untuk melakukan deploymen sub-komponen ini dibutuhkannya sebuah konfigurasi yang mendeskripsikan versi sub-komponen yang digunakan oleh perangkat lunak. Dengan bahasa modern sekarang konfigurasi sub-komponen yang digunakan oleh main aplikasi biasanya sudah automatis terbuat contohnya antara lain adalah; go modules [3] dalam bahasa Golang, requirement file [4] dalam bahasa Python, atau file RubySpec [5] pada bahasa Ruby.

Terdapat beberapa karakteristik yang mendasar pada deployment sebuah perangkat lunak yang ditulis oleh Alan Dearle yaitu [6]; ***Release*** berupa jembatan antara proses deployment dengan proses development. yang meliputi semua operasi yang diperlukan untuk mempersiapkan sebuah sistem untuk di-transfer ke konsumen. Aktivitas release ini juga menentukan resource yang dibutuhkan oleh sebuah sistem perangkat lunak untuk dapat beroperasi pada environment nya. Setelah itu dilakukan packaging pada sistem perangkat lunak. Package tersebut harus mengandung komponen yang dibutuhkan oleh sistem, deskripsi sistem, dependencies pada komponen eksternal, prosedur deployment, dan semua informasi yang relevan dari sistem tersebut pada environment yang akan dijalankan. ***Installation*** diperlukan untuk persiapan melakukan activation. ***Activation*** adalah proses ekseksui sebuah perangkat lunak pada waktu tertentu, biasa menggunakan grafik antar muka ataupun

proses daemon. Updating adalah proses untuk mengganti bagian dari perangkat lunak yang terinstal dengan versi yang lebih baru. Selanjutnya yaitu ***Undeployment*** yaitu proses menghapus software yang terinstall dalam sebuah mesin ini dapat disebut dengan *deinstallation*.

Menurut Mockus dkk [7] kualitas deployment sebuah perangkat lunak masuk kedalam faktor utama dalam persepsi konsumer dalam hal kualitas sebuah perangkat lunak. Jansen dan Brinkkemper [8] juga mengatakan bahwa kelancaran sebuah deployment perangkat lunak adalah hal esensial untuk meningkatkan produk perangkat lunak sebuah perusahaan/organisasi. Tetapi terdapat beberapa tantangan yang dihadapi pada saat melakukan aktivitas deploymet. Menurut Antonio Carzaniga [2] terdapat beberapa tantangan yang sering dihadapi pada saat melakukan deployment yaitu; **Mengganti** atau melakukan update sebuah sistem terhadap komponen yang sudah berjalan, **Dependencies** komponen antar satu sama lain, dan **Koordinasi** ketika melakukan update apakah akan mengganggu proses bisnis yang sedang berjalan atau tidak, dan juga **Mengatur** platform yang heterogen misalnya terhadap spesifik sistem operasi yang digunakan.

Seiring berjalannya waktu, sebuah aplikasi cenderung menjadi semakin kompleks [9, 10]. Dengan team pengembang dan aplikasi yang selalu bertumbuh dari sisi kompleksitas dan *maintainability* biasanya menyebabkan model pengembangan aplikasi menjadi susah untuk dikembangkan atau dapat dikatakan terdapat *bottleneck* sehingga aplikasi menjadi tidak efisien [11]. Dengan berkembangnya kompleksitas sebuah aplikasi diperlukan sebuah arsitektur yang bisa menyelesaikan hal itu, salah satu caranya yaitu menggunakan arsitektur *microservice* [9]. Dengan *microservice* aplikasi dibagi menjadi bagian-bagian kecil (unit) yang terpisah satu dengan lainnya [9]. Masing-masing bagian aplikasi ini dapat dijalankan dan dikembangkan secara independent (dari sisi developer) [12].

Saat ini aplikasi berbasis *microservice* menjadi pilihan sebagai arsitektur utama ketika membangun aplikasi yang scalable [13]. Aplikasi berbasis *microservice* ini dulunya dikembangkan dengan dengan menjalankan beberapa VM (virtual machine) yang saling berkomunikasi satu sama lain melalui REST/HTTP (Hypertext Transfer Protocol) ataupun RPC *(Remote Procedure Call)* [14]. Karena pada saat itu deployment *microservice* itu sendiri masih berbasis VM yang membutuhkan operasi manual dan juga biaya yang mahal, aplikasi berbasis *microservice* pun menjadi susah dan kompleks dari sisi biaya dan waktu yang dibutuhkan untuk dilakukan pengembangan dan juga *scaling* [14] .

Teknologi VM sendiri sudah perlahan ditinggalkan ketika merancang *microservice*. Dengan adanya teknologi *containerization* aplikasi dapat dibungkus agar dapat dijalankan dengan mudah dan efisien [14]. Menjalankan aplikasi dengan abstraksi yang diberikan oleh sebuah *container* juga membawa fleksibilitas dalam pengelolaannya. Salah satu manfaatnya adalah scaling aplikasi yang jauh lebih mudah, yaitu hanya dengan melakukan penyesuaian jumlah *container* yang dijalankan [15].

Containerization [16] ini sangat berdampak pada aspek infrastruktur dan *runtime* sebuah aplikasi. Tapi dengan adanya *container* diperlukan juga sebuah sistem yang melakukan orkestrasi secara otomatis pada *container* tersebut. Dengan adanya kubernetes kita dapat memanfaatkan fitur fitur yang diberikan oleh kubernetes antara lain fitur automatic scaling [17, 18]. Saat ini untuk melakukan deployment sebuah *container* (*service)*) dilakukan secara manual dengan merubah file deployment. Dengan demikian, diperlukan cara otomatis untuk melakukan deployment *service* yang baru/diubah.

Dengan banyaknya persaingan dalam dunia perangkat lunak yang terjadi dalam waktu ini sebuah perusahaan atau developer memerlukan waktu yang cepat untuk melakukan deployment sebuah perangkat lunak. Terdapat beberapa macam workflow sebuah deploymen perangkat lunak saat ini, tetapi yang industri saat ini lakukan adalah metode DevOps [19]. Metode DevOps sendiri merupakan metode yang digunakan untuk mengembangkan perangkat lunak yang menjembatani antara dua team yang terisolasi dalam struktur organisasi, contohnya adalah team pengembang (Dev) dan team operasi (Ops) [20]. Konsep DevOps [19] sendiri memungkinkan team devloper dan operasi untuk membangun sebuah perangkat lunak yang dapat dijalankan secara otomatis dan secara berkala dengan menggunakan alat bantu DevOps. Tujuan utama dari itu adalah untuk meningkatkan kecepatan, reliabilitas, dan perangkat lunak yang lebih baik. Dalam DevOps sendiri terdapat beberapa sub-metode yang digunakan yaitu, *Continuous Integration* (CI), *Continuous DElivery* (CDE), dan *Continuous Deployment* (CD) [21].

Walaupun DevOps sendiri memiliki beberapa keunggulan, tetapi implementasi CI,CDE, dan CD bukanlah hal yang mudah. Pemilihan tools dan adaptasinya, adaptasi karyawan, dan miskonfigurasi yang biasa terjadi pada saat migrasi menggunakan metode DevOps.

Terdapat beberapa masalah yang sering terjadi pada saat menggunakan metode DevOps yaitu; Arsitektur [20], tools [22], metode baru [23, 24], Keamanan dari flow CI/CD [25], dan mekanisme rollback [26]. Pada penelitian yang dilakukan oleh Ramadoni dkk. [27] dilakukan analisis menggunakan metode GitOps yang dapat menyelesaikan permasalahan mekanisme rollback, dan keamanan flow CI/CD

Pada penelitian yang dilakukan oleh Ramadoni dkk. [27] digunakan metode pull-based untuk menyelesaikan permasalahan diatas. Pada penelitian tersebut tidak dijelaskan perbedaan kenapa harus menggunakan metode pull-based atau push-based dan juga tidak dijelaskan perbandingan tools yang digunakan sebagai operator GitOps yaitu ArgoCD pada sebuah cluster Kubernetes. Maka tujuan dari penelitian ini adalah untuk melakukan analisis pada tools yang digunakan sebagai operator GitOps dan melakukan perbandingan metode pull-based dan push-based pada metode DevOps.

1.2 Rumusan Masalah

Berdasarkan latar belakang masalah diatas, adapun rumusan masalah pada penelitian ini antara lain:

a. Bagaimana cara sebuah service yang source nya diubah dilakukan deployment secara otomatis pada sistem kubernetes dengan metode GitOps ?

b. Bagaimana cara provisioning infrastruktur yang dibutuhkan oleh service secara otomatis pada sistem microservice yang berjalan pada kubernetes?

c. Bagaimana perbedaan mendasar dari tools yang digunakan sebagai operator GitOps pada sistem kubernetes?

d. Bagaimana hasil analisis tipe deployment Pull-based dan Push-based pada continous deployment?

1.3 Tujuan Penelitian

Berdasarkan rumusan masalah yang telah diformulasikan, maka terdapat beberapa tujuan pada studi ini:

a. Melakukan implementasi continous deployment pada sebuah sistem microservice yang berjalan pada kubernetes.

b. Melakukan analisis terhadap metode deployment secara automatis menggunakan Pull-based dan Push-based pada continuous deployment.

c. Melanjutkan saran dari penelitian sebelumnya [27] dengan merancang workflow continuous integration yang diintegrasikan dengan ArgoCD.

1.4 Batasan Masalah

Dalam penelitian ini, peneliti akan membatasi masalah yang akan diteliti antara lain:

a. Menggunakan kubernetes sebagai alat orkestrasi *container*.

b. Implementasi CI/CD dilakukan menggunakan GitLabCI.

c. Container runtime yang digunakan adalah Docker.

d. Aplikasi microservice yang digunakan berupa aplikasi berbasis web.