Automatic Deployment Aplikasi berbasis *Microservice* Pada Platform Kubernetes Dengan Metode *Pull-Up*

Proposal Tugas Akhir

Diajukan Untuk Memenuhi
Persyaratan Guna Meraih Gelar Sarjana
Informatika Universitas Muhammadiyah Malang

Muhammad Zein Ihza Fahrozi
201810370311072

Bidang Minat
(Rekayasa Perangkat Lunak)

PROGRAM STUDI TEKNIK INFORMATIKA FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH MALANG
2021

CHAPTER 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.

CHAPTER II

METODE PENELITIAN

2.1 Permasalahan pada Software Deployment

Dimasa saat ini dimana sebuah permintaan pasar pada sektor teknologi berubah cepat dan digunakan oleh banyak orang. Diperlukan juga cara mendeliver teknologi tersebut secara cepat, aman, dan reliable. Dengan adanya permintaan yang cukup banyak dan berubah ubah setiap saat. Banyak perusahaan yang menerapkan metode agile development pada pengembangan produk nya. Pada sistem agile biasanya suatu masalah dipecah menjadi sebuah stories dan dilakukan estimasi effort oleh developer untuk menyelesaikannya. Banyak juga manager yang mengukurnya ketika sebuah stories selesai itu terdapat pikiran jika team nya meningkat dalam hal kecepatan (velocity) development. Dengan menggunakan metrik velocity tersebut untuk mengukur sebuah produktifitas sebuah team menjadikan itu bagian hal yang tidak absolute. Apakah setiap team yang "menyelesaikan" banyak stories dapat dibilang produktif dari segi feedback yang didapatkan ketika produk/kode nya berjalan pada produksi ?.

Menurut Forsgen pada bukunya "Accelerate: The Science of DevOps" [28] secara tradisional reliability diukur ketika waktu sistem tersebut gagal. Tetapi pada software product atau services yang modern yang selalu berganti ganti dan kompleks, kegagalan tidak dapat dihindari lagi. Pada bukunya juga dia melakukan survey yang diambil pada taun 2014-2016 yang saya simpulkan secara garis besar bahwa perusahaan yang menerapkan observability terhadap sistem nya mendapatkan failure rate yang sangat rendah dibanding dengan yang lainnya.

2.2 Continous Delivery Meningkatkan Software Delivery Performance

Menurut Forsgen [28] terdapat dampak penerapan continuous delivery pada perusahaan yaitu:

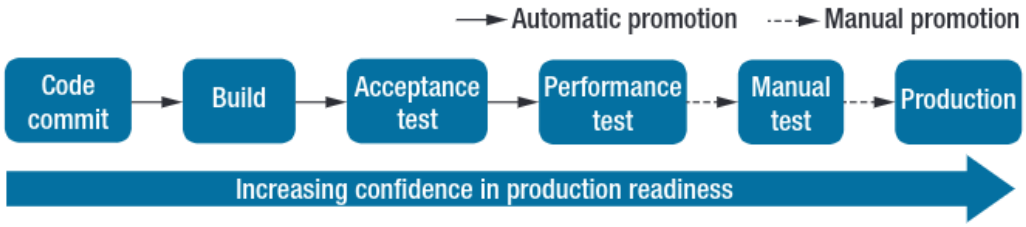
a. Sebuah team dapat melakukan deployment ke tangan user secara langsung dengan cepat.

b. Mendapatkan feedback cepat dalam kualitas software.

c. Meningkatkan produktifikasi developer.

2.2.1 Metode Continuous Delivery (CDE)

Continuous Delivery (CDE) adalah disiplin dalam rekayasa perangkat lunak dimana sebuah team dapat memproduksi perangkat lunak yang valuable secara incremental dalam siklus yang pendek dan menjamin sebuah perangkat lunak dapat di-release pada waktu kapan saja [29].



Gambar 2.1 Contoh pipeline dalam Continous Delivery

2.2.2 Metode Continuous Deployment

Perbedaan mendasar dari Continuous Delivery (CDE) dan Continuous Deployment (CD) adalah dimana implementasi semua fase dilakukan secara automatis tanpa memerlukan intervensi manusia, deployment ke tahap environment produksi juga bisa dilakukan secara automatis, tetapi biasanya memerlukan intervensi pada satu tahap [27].

2.3 Deployment Pada Kubernetes

Pada padasarnya terdapat 2 metode yang digunakan untuk melakukan deployment pada sebuah aplikasi pada kubernetes yaitu pull method dan push method [27]. Perbedaan mendasar dari pull method dan push method terdapat pada agent yang melakukan deployment [30]. Pada push method sebuah agent melakukan deployment sebuah service terhadap suatu platform (eg, kubernetes).

2.3.1 Literatur Pull-based Deployment

Berdasarkan penelitian terbaru, pendekatan pull-based deployment telah terbukti memberikan beberapa keunggulan signifikan dalam konteks GitOps [31]. Beberapa keunggulan utama dari pull-based deployment antara lain:

1. **Keamanan yang Lebih Baik**: Tidak memerlukan kredensial akses eksternal

ke cluster Kubernetes, sehingga mengurangi risiko kebocoran kredensial.

2. **State yang Konsisten**: Operator secara berkala memeriksa dan menyinkronkan state yang diinginkan dengan state aktual di cluster.

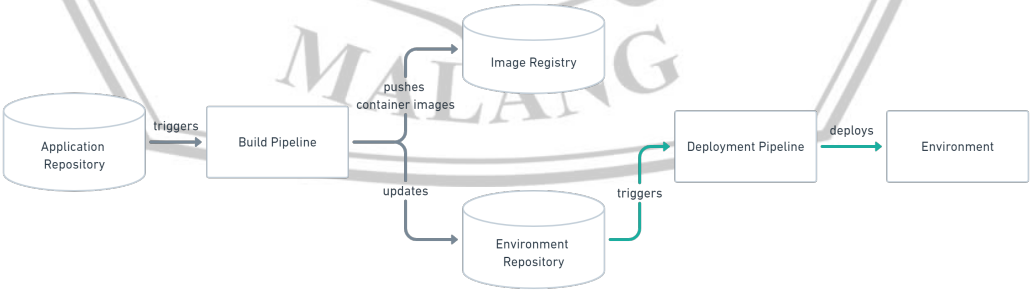
3. **Recovery Otomatis**: Dapat secara otomatis mengembalikan konfigurasi ke state yang diinginkan jika terjadi perubahan yang tidak diinginkan.

4. **Multi-cluster Management**: Memudahkan pengelolaan beberapa cluster Kubernetes sekaligus dari satu sumber kebenaran.

Menurut [32], implementasi pull-based deployment dengan Argo CD telah menunjukkan peningkatan keandalan deployment sebesar 40% dibandingkan dengan pendekatan push-based tradisional.

2.3.2 Push-based Deployment

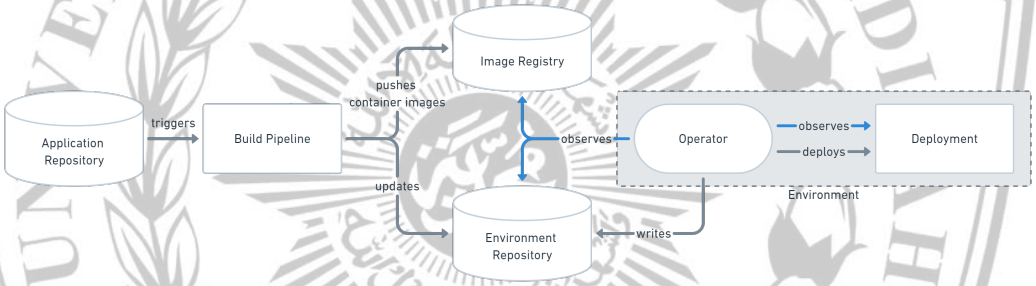
Push-based deployment [30] merupakan strategy yang popular yang diimplementasikan oleh tools seperti Jenkins, CircleCI, atau TravisCI. Sourcecode dari sebuah aplikasi terdapat pada repository yang sama dengan konfigurasi YAML Kubernetes yang diperlukan untuk melakukan deployment aplikasi tersebut. Kapan pun code dari aplikasi tersebut diupdat, pipeline akan berjalan, dimana akan membuat container image yang diperlukan. Perlu diperhatikan juga bahwa biasanya credential environment untuk melakukan deployment pada metode Push-based. Jadi pada pipeline tersebut kita dapat melihat configurasi rahasia yang mungkin saja terlihat oleh orang lain.



Gambar 2.2 Contoh pipeline dalam push-based deployment

2.3.3 Pull-based Deployment

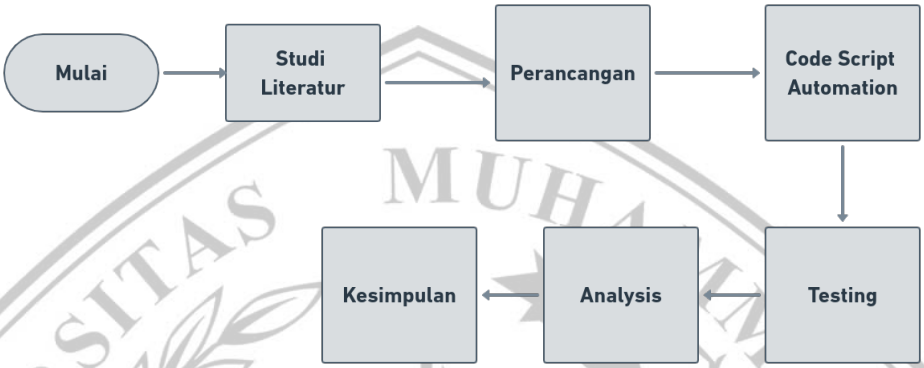
Pada website insert gitops Strategi Pull-based deployment menggunakan konsep yang sama dengan push-based tetapi berbeda pada bagaimana cara kerja pada pipeline deployment. Secara traditional pipeline CI/CD akan di-trigger oleh event eksternal, contohnya ketika ada update kode baru yang pada repository. Dengan metode pull-based deployment, dikenalkan sebuah operator. Operator tersebut secara kontinu dengan interval yang dapat diatur sendiri melakukan komparasi terhadap state yang ada di repository dengan state yang ter-deploy pada infrastructure. Ketika terdapat perbedaan, operator melakukan update pada infrastructure untuk mencocokan state dengan apa yang ada di repository. Operator harus berada pada environment atau kluster yang sama dengan applikasi yang akan dideploy. Dengan ini pull-based method tidak perlu mengetahui environment eksternal karena semua sudah ada pada kluster yang sama.



Gambar 2.3 Contoh pipeline dalam pull-based deployment

2.4 Alur Penelitian dan Implementasi

Penelitian ini akan mengikuti alur metodologi yang terstruktur untuk memastikan pencapaian tujuan penelitian. Berikut adalah bagan alur penelitian yang akan diimplementasikan:



Gambar 2.4 Alur Penelitian dan Implementasi

Adapun penjelasan rinci dari setiap tahapan adalah sebagai berikut:

1. **Studi Literatur**: Mengkaji teori dan penelitian terdahulu terkait GitOps, Argo CD, dan Kubernetes.
2. **Perancangan Sistem**: Merancang arsitektur dan alur kerja sistem automasi deployment.
3. **Implementasi Infrastruktur**: Menyiapkan lingkungan Kubernetes dan komponen pendukungnya.
4. **Implementasi Argo CD**: Mengintegrasikan Argo CD ke dalam cluster Kubernetes.
5. **Pengujian dan Evaluasi**: Melakukan pengujian fungsional dan non-fungsional.
6. **Analisis Hasil**: Menganalisis hasil pengujian dan mengevaluasi pencapaian tujuan penelitian.

2.5 Skenario Pengujian

Untuk memastikan bahwa implementasi Argo CD berfungsi sesuai dengan yang diharapkan, akan dilakukan pengujian dengan skenario sebagai berikut:

2.5.1 Test Case 1: Deployment Aplikasi

- **Tujuan**: Memverifikasi bahwa perubahan kode aplikasi yang di-push ke repository dapat terdeploy otomatis ke cluster Kubernetes.
- **Langkah-langkah**:
  1. Lakukan perubahan pada kode aplikasi
  2. Push perubahan ke repository Git
  3. Amati proses sinkronisasi Argo CD
  4. Verifikasi aplikasi berjalan dengan versi terbaru
- **Hasil yang Diharapkan**: Aplikasi terdeploy otomatis dengan versi terbaru tanpa intervensi manual.

2.5.2 Test Case 2: Rollback Otomatis

- **Tujuan**: Memverifikasi bahwa Argo CD dapat mendeteksi dan melakukan rollback ketika terjadi kegagalan deployment.
- **Langkah-langkah**:
  1. Deploy konfigurasi yang tidak valid
  2. Amati proses deteksi kegagalan oleh Argo CD
  3. Verifikasi sistem kembali ke state terakhir yang stabil
- **Hasil yang Diharapkan**: Sistem otomatis melakukan rollback ke versi stabil sebelumnya.

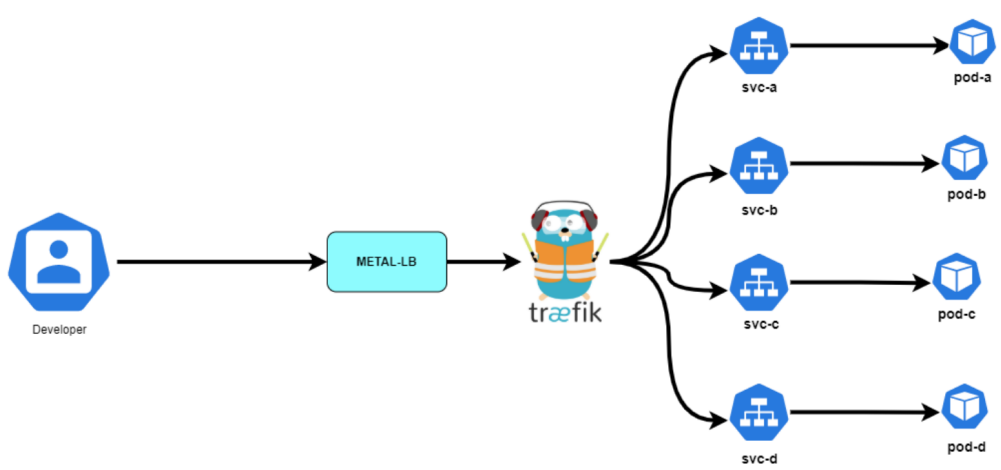
2.5.3 Test Case 3: Multi-Environment Deployment

- **Tujuan**: Memverifikasi kemampuan Argo CD dalam mengelola deployment di berbagai environment (development, staging, production).
- **Langkah-langkah**:
  1. Konfigurasi Argo CD untuk multiple environments
  2. Lakukan deployment ke environment development
  3. Promosikan ke staging setelah pengujian
  4. Lakukan final deployment ke production

- **Hasil yang Diharapkan**: Deployment berhasil dilakukan di semua environment dengan konfigurasi yang sesuai.

2.6 Arsitektur Cluster Kubernetes yang akan digunakan

Dibawah ini merupakan rancangan arsitektur high-level pada implementasi Kubernetes nantinya. Disini menggunakan Metal-LB sebagai provider load balancer dan Traefik sebagai ingress controller.



Gambar 2.5 Arsitektur Kubernetes yang akan digunakan

CHAPTER III

KESIMPULAN

Deployment software berbasis microservice saat ini banyak menggunakan container dan dibutuhkan kubernetes untuk melakukan orkestrasi container-container tersebut. Deployment sistem terdistribusi khususnya pada platform kubernetes cukuplah kompleks. Cepatnya perubahan dan meningkatnya kompleksitas sebuah software yang dikembangkan dibutuhkan suatu cara untuk melakukan deployment secara automatis agar seorang developer dapat mendapatkan feedback secara cepat tentang apa yang dikerjakannya. Oleh karena itu diperlukan penerapan Continous Deployment yang mengikuti prinsip GitOps akan didapatkan produktifitas tinggi dari sisi developer dan management.

DAFTAR PUSTAKA

[1] M. R. Lyu, "Software reliability engineering: A roadmap," 2007, pp. 153–170.

[2] A. Carzaniga, A. Fuggetta, R. S. Hall, D. Heimbigner, A. V. D. Hoek, and A. L. Wolf, "A characterization framework for software deployment technologies," 1998.

[3] "The go blog on go modules," Jul 2021. [Online]. Available: https://go.dev/blog/using-go-modules

[4] "How to install python packages with pip and requirements.txt." [Online]. Available: https://note.nkmk.me/en/python-pip-install-requirements/

[5] O. S. Software, "Ruby/spec: The ruby spec suite aka ruby/spec." [Online]. Available: https://github.com/ruby/spec

[6] A. Dearle, "Software deployment, past, present and future," 2007.

[7] A. Mockus and P. Zhang, "Predictors of customer perceived software quality," 2005.

[8] S. Jansen and S. Brinkkemper, "Definition and validation of the key process of release, delivery and deployment for product software vendors: turning the ugly duckling into a swan," 2006. [Online]. Available: http://www.swebok.org

[9] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of Grid Computing*, vol. 12, pp. 559–592, 11 2014.

[10] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 1st ed. O'Reilly Media, 2015.

[11] Y. Yu, H. Silveira, and M. Sundaram, "A microservice based reference architecture model in the context of enterprise architecture." IEEE, 10 2016, pp. 1856–1860.

[12] Z. Xiao, I. Wijegunaratne, and X. Qiang, "Reflections on soa and microservices." Institute of Electrical and Electronics Engineers Inc., 3 2017, pp. 60–67.

[13] J. Q. Wu and T. Wang, "Research and application of soa and cloud computing model." Institute of Electrical and Electronics Engineers Inc., 12 2014, pp. 294–299.

[14] H. Khazaei, C. Barna, N. Beigi-Mohammadi, and M. Litoiu, "Efficiency analysis of provisioning microservices," 2016.

[15] V. Singh and S. K. Peddoju, "Container-based microservice architecture for cloud applications." IEEE, 5 2017, pp. 847–852.

[16] Davidbritch, "Containerized microservices - xamarin." [Online]. Available: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/containerized-microservices#:~:text=with%20client%20apps.-,Containerization,to%20a%20host%20operating%20system.

[17] L. A. Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "Deploying microservice based applications with kubernetes: Experiments and lessons learned," vol. 2018-July. IEEE Computer Society, 9 2018, pp. 970–973.

[18] "What is kubernetes?" Jul 2021. [Online]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[19] L. Bass, "The software architect and devops," *IEEE Software*, vol. 35, pp. 8–10, 1 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8239924/

[20] R. Bolscher and M. Daneva, "Designing software architecture to support continuous delivery and devops: A systematic literature review." SciTePress, 2019, pp. 27–39.

[21] G. Kim, K. Behr, and G. Spafford, *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win*, 1st ed. IT Revolution Press, 2013.

[22] A. Proulx, F. Raymond, B. Roy, and F. Petrillo, "Problems and solutions of continuous deployment: A systematic review," 12 2018. [Online]. Available: http://arxiv.org/abs/1812.08939

[23] M. K. A. Abbass, R. I. E. Osman, A. M. H. Mohammed, and M. W. A. Alshaikh, "Adopting continuous integeration and continuous delivery for small teams." IEEE, 9 2019, pp. 1–4.

[24] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A survey of devops concepts and challenges," 11 2019.

[25] M. Shahin, M. A. Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," pp. 3909–3943, 2017.

[26] J. Fritzsch, J. Bogner, S. Wagner, and A. Zimmermann, "Microservices migration in industry: Intentions, strategies, and challenges." Institute of Electrical and Electronics Engineers Inc., 9 2019, pp. 481–490.

[27] Ramadoni, E. Utami, and H. A. Fatta, "Analysis on the use of declarative and pull-based deployment models on gitops using argo cd." Institute of Electrical and Electronics Engineers (IEEE), 10 2021, pp. 186–191.

[28] N. Forsgren, J. Humble, and G. Kim, *Accelerate: The Science of Lean Software and DevOps Building and Scaling High Performing Technology Organizations*, 1st ed. IT Revolution Press, 2018.

[29] L. Chen, "Continuous delivery: Huge benefits, but challenges too," *IEEE Software*, vol. 32, pp. 50–54, 3 2015.

[30] F. Beetz, A. Kammer, and S. Harrer, "Gitops." [Online]. Available: https://www.gitops.tech/

[31] M. Korhonen, "Gitops tool argo cd in service management," 2021. [Online]. Available: https://www.theseus.fi/bitstream/10024/505942/2/Thesis_Korhonen_Matti.pdf

[32] A. Sharma, R. Gupta, and P. Singh, "A comparative study of gitops tools: Argo cd vs. flux in multi-cluster kubernetes environments," *International Journal of Cloud Computing*, 2022.