KONTAINERISASI *SHIBBOLETH IDP* SEBAGAI AKSES MANAJEMEN *SINGLE SIGN ON* PADA ARSITEKTUR *MICROSERVICE* SISTEM ENTERPRISE DENGAN METODE *LOAD BALANCING*

Ikhwan Alfath Nurul Fathony
19917028

Tesis diajukan sebagai syarat untuk meraih gelar Magister Komputer
Konsentrasi Sistem Informasi Enterprise
Program Studi Informatika Program Magister
Fakultas Teknologi Industri
Universitas Islam Indonesia
2022

Lembar Pengesahan Pembimbing

Kontainerisasi *Shibboleth IdP* Sebagai Akses Manajemen *Single Sign On* Pada Arsitektur *Microservice* Sistem Enterprise dengan Metode *Load Balancing*

Ikhwan Alfath Nurul Fathony

19917028

Yogyakarta, 4 Desember 2022

Pembimbing

Mukhammad Andri Setiawan, S.T., M.Sc., Ph.D.

Lembar Pengesahan Penguji

Kontainerisasi *Shibboleth IdP* Sebagai Akses Manajemen *Single Sign On* Pada Arsitektur *Microservice* Sistem Enterprise dengan Metode *Load Balancing*

Ikhwan Alfath Nurul Fahony
19917028

Yogyakarta, 4 Desember 2022

Tim Penguji,

Mukhammad Andri Setiawan, S.T., M.Sc., Ph.D. _____
Ketua

Dr. R. Teduh Dirgahayu., S.T., M.Sc. _____
Anggota I

Ahmad Munasir Rafie Pratama, S.T., M.I.T.,Ph.D. _____
Anggota II

Mengetahui,
Ketua Program Studi Informatika Program Magister
Universitas Islam Indonesia

Irving Vitra Paputungan, S.T., M.Sc., Ph.D.

Abstrak

Kontainerisasi *Shibboleth IdP* Sebagai Akses Manajemen *Single Sign On* Pada Arsitektur *Microservice* Sistem Enterprise dengan Metode *Load Balancing*

Banyak organisasi menggunakan teknologi informasi sebagai alat untuk mengoptimalkan proses bisnisnya, tetapi masih belum terintegrasi dengan baik. Hal ini menjadi hambatan ketika suatu proses membutuhkan kolaborasi dengan layanan lain. Dengan demikian, dibutuhkan strategi baru untuk memanajemen sistem yang terintegrasi, agar dapat meningkatkan layanan tersebut. Salah satunya yaitu dengan menerapkan arsitektur *microservices* pada sistem enterprise. Layanan sistem dan teknologi informasi (SI/TI) di lingkungan Universitas Islam Indonesia, berada dibawah tanggung jawab pengelolaan Badan Sistem Informasi (BSI) telah merancang dan mengimplementasikan sistem enterprise ke dalam layanan satu pintu, yaitu UII Gateway. BSI membangun aplikasi web UII Gateway dengan mengadopsi arsitektur *microservices*. *Microservices* berarti membagi aplikasi menjadi layanan yang lebih kecil dan saling terhubung agar lebih responsif untuk mengikuti perkembangan zaman terhadap perubahan kebutuhan yang sangat cepat. Namun, ketika pertama kali diluncurkan aplikasi UII Gateway telah mengalami kendala pada layanan *login*. Oleh sebab itu, perlu solusi untuk meningkatkan ketersediaan pada layanan login, karena menjadi gerbang utama bagi pengguna untuk mengakses fungsi-fungsi yang ada pada UII Gateway. Pemusatan login aplikasi dengan menggunakan *Shibboleth Identity Provider (IdP)* sebagai manajemen *Single Sign On (SSO)* menjadi perhatian khusus karena masih terdapat resiko *Single Point of Failure (SPOF),* yang mengakibatkan terjadinya *downtime* pada aplikasi UII Gateway. Solusi dalam penelitian ini yaitu dengan membungkus *Shibboleth IdP* menjadi sebuah *container* pada arsitektur *microservice*. Dalam penelitian ini, membuktikan bahwa layanan *SSO* menggunakan *container Shibboleth IdP,* beserta proses automasi yang dijalankan pada *cluster Kubernetes* dan penyimpanan data *Longhorn,* dapat mencegah terjadinya *Single Point of Failure* dengan fungsi redundansinya. Sehingga kebutuhan autentikasi pada login aplikasi enterprise UII Gateway menggunakan *Shibboleth IdP* tetap aktif dan *reliable* walaupun terjadi kerusakan ataupun kegagalan pada salah satu mesin *server*.

Kata kunci

architecture microservices system, container, Shibboleth, *Single Sign On (SSO)*

Abstract

Shibboleth IdP Containerization as an SSO Management Access Method in Enterprise System Microservice Architecture with Load Balancing

Many businesses use information technology to improve their business processes, but it is still not well integrated. This becomes a bottleneck when a process requires collaboration with other services. Thus, a new strategy is needed to manage an integrated system, in order to improve these services. One of them is by implementing microservices architecture on enterprise systems. System and information technology (IS/IT) services within the Islamic University of Indonesia, are under the management responsibility of the *Badan Sistem Informasi* (BSI), which has designed and implemented an enterprise system into a one-stop service, namely the UII Gateway. BSI built the UII Gateway web application by adopting the microservices architecture. Microservices refers to the division of applications into smaller, interconnected services in order to be more responsive to rapidly changing needs. However, when the UII Gateway application was first launched, it encountered login service issues. As a result, a solution to increase the availability of the login service is required, as it is the primary gateway for users to access the functions available on the UII Gateway. Centralizing application logins using Shibboleth Identity Provider (IdP) as Single Sign On (SSO) management is especially concerning because there is still a Single Point of Failure (SPOF) risk, which results in UII Gateway application downtime. In this study, the solution is to wrap Shibboleth IdP in a container on a microservice architecture. This study demonstrates that the Shibboleth IdP container, in conjunction with the automation process that runs on the Kubernetes cluster and Longhorn data storage, can prevent Single Point of Failure through its redundancy function. So that the need for authentication on the UII Gateway enterprise application login using Shibboleth IdP remains active and reliable even if one of the server machines is damaged or fails.

Keywords

architecture microservices system, container, Shibboleth, *Single Sign On (SSO)*

Pernyataan Keaslian Tulisan

Dengan ini saya menyatakan bahwa tesis ini merupakan tulisan asli dari penulis, dan tidak berisi material yang telah diterbitkan sebelumnya atau tulisan dari penulis lain terkecuali referensi atas material tersebut telah disebutkan dalam tesis. Apabila ada kontribusi dari penulis lain dalam tesis ini, maka penulis lain tersebut secara eksplisit telah disebutkan dalam tesis ini.

Dengan ini saya juga menyatakan bahwa segala kontribusi dari pihak lain terhadap tesis ini, termasuk bantuan analisis statistik, desain survei, analisis data, prosedur teknis yang bersifat signifikan, dan segala bentuk aktivitas penelitian yang dipergunakan atau dilaporkan dalam tesis ini telah secara eksplisit disebutkan dalam tesis ini.

Segala bentuk hak cipta yang terdapat dalam material dokumen tesis ini berada dalam kepemilikan pemilik hak cipta masing-masing. Apabila dibutuhkan, penulis juga telah mendapatkan izin dari pemilik hak cipta untuk menggunakan ulang materialnya dalam tesis ini.

Yogyakarta, 4 Desember 2022

Ikhwan Alfath Nurul Fathony, S.Kom.

Daftar Publikasi

Sitasi publikasi 1

Kontributor	Jenis Kontribusi
Ikhwan Alfath Nurul Fathony, S.Kom.	Mendesain eksperimen (60%) Menulis *paper* (70%)
Mukhammad Andri Setiawan, S.T.,M.Sc.,Ph.D.	Mendesain eksperimen (40%) Menulis dan mengedit *paper* (30%)

Halaman Kontribusi

Dalam penelitian ini terdapat beberapa kontribusi dari pihak lain seperti pada aplikasi UII Gateway yang di dalamnya terdapat banyak modul aplikasi terkait, dan dikembangkan oleh Badan Sistem Informasi Universitas Islam Indonesia.

Halaman Persembahan

Bismillahirrahmanirrahim

Saya persembahkan tesis ini kepada bapak dan ibu saya yang selalu tiada henti mendoakan serta memberikan semangat dan dukungan yang tak henti – hentinya agar selalu dipermudahkan dalam segala urusan.

Untuk Program Studi Magister Informatika yang telah memberikan beasiswa alumni kepada saya pada program magister ini. Sahabat – sahabat seperjuangan yang saya banggakan, sudah 2 tahun ini selalu bersama dalam menjalani masa kuliah. Terimakasih sahabat Badan Sistem Informasi (BSI) UII yang telah memberikan dukungan dan tempat penelitian, sahabat seperjuangan himpunan mahasiswa magister informatika (HIMMAIF) UII. Selalu jaga sportifitas dan tetap semangat semoga sukses untuk kita semua! Aamiin

Kata Pengantar

Assalamualaikum Wr. Wb.

Syukur Alhamdulillah segala rahmat yang telah diberikan oleh Allah SWT, sebab tiada hidayah yang lebih besar daripada hidayah yang telah diberikan oleh-Nya. Shalawat serta salam semoga tetap tercurah kepada junjungan kita Nabi Muhammad SAW beserta keluarga dan para sahabat. Sehingga atas ridho-Nya tesis yang berjudul "Kontainerisasi *Shibboleth IdP* Sebagai Akses Manajemen *Single Sign On* Pada *Architecture Microservice* Sistem Enterprise dengan Metode *Load Balancing*" dapat diselesaikan dengan baik.

Tesis ini disusun sebagai syarat terakhir yang harus ditempuh untuk menyelesaikan pendidikan pada jenjang Magister, pada Program Study Magister Informatika Universitas Islam Indonesia. Peneliti menyadari bahwa tanpa bimbingan, dorongan dan bantuan dari berbagai pihak tesis ini tidak akan terwujud. Oleh karena itu dengan kerendahan hati peneliti mengucapkan terima kasih sebesar-besarnya kepada:

1. Bapak Fathul Wahid, S.T., M.Sc., Ph.D. selaku Rektor Universitas Islam Indonesia,
2. Bapak Prof. Dr. Ir. Hari Purnomo, M.T. selaku Dekan Fakultas Teknologi Industri Universitas Islam Indonesia,
3. Irving Vitra Paputungan, S.T., M.Sc., Ph.D. selaku Ketua Program Studi Informatika Program Magister Universitas Islam Indonesia
4. Dr. Mukhammad Andri Setiawan, S.T., M.Sc., Ph.D. selaku pembimbing tesis yang telah memberikan bimbingan dan pendampingan kepada penulis,
5. Bapak dan ibu dosen Jurusan Informatika yang telah memberikan ilmunya kepada penulis, semoga bapak dan ibu dosen selalu dalam rahmat dan lindungan Allah SWT, sehingga ilmu yang telah diajarkan dapat bermanfaat di kemudian hari,
6. Ucapan terima kasih sebesar-besarnya kepada kedua orang tua yang tercinta, dengan segala pengorbanannya yang luar biasa, serta doa - doa yang tidak pernah putus untuk penulis serta nasihat dan petunjuk dari mereka yang menjadikan motivasi terbesar bagi penulis untuk dapat menyelesaikan tesis ini.
7. Ucapan terima kasih sebesar-besarnya kepada sahabat Badan Sistem Informasi (BSI) UII, yang telah memberikan dukungan, semangat, dan juga tempat untuk saya dapat melakukan penelitian ini,
8. Teman- teman seperjuangan yang telah memberikan dorongan dan semangat kepada penulis dalam menyelesaikan tesis ini, terima kasih sahabat – sahabat seperjuangan himpunan magister informatika (HIMMAIF),

Tentunya sebagai manusia tidak pernah lepas dari kesalahan, sehingga dalam penyusunan tesis ini masih banyak terdapat kesahalan dan kekurangan. Oleh karena itu penulis memohon maaf atas segala kesalahan dan kekurangan yang ada.

Wassalamualaikum Wr. Wb.

Yogyakarta, 4 Desember 2022

Ikhwan Alfath Nurul Fathony, S.Kom.

Daftar Isi

Daftar Tabel

Daftar Gambar

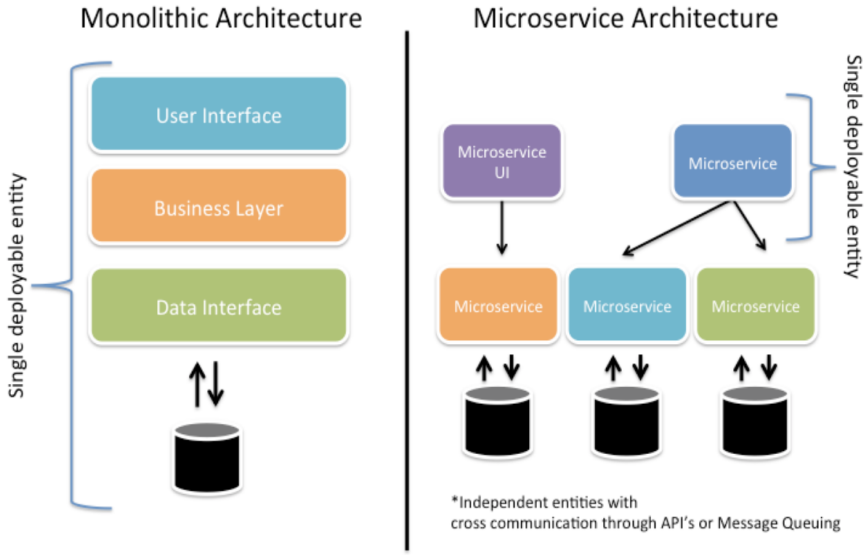
Glosarium

SSO	- Single Sign On
K8s	- Kubernetes
SPOF	- Single Point of Failure
API	- Application Programming Interface
CI/CD	- Continuous Integration Continuous Delivery
SP	- Service Provider
IdP	- Identity Provider
RKE	- Rancher Kubernetes Engine
SAML	- Security Assertion Markup Language
CPU	- Central Processing Unit
RAM	- Random Access Memory
HPA	- Horizontal Pod Autoscaler
HTTP	- Hypertext Transfer Protocol
AAA	- Authentication, Authorization, Accounting
DRC	- Data Recovery Center

BAB 1
Pendahuluan

1.1 Latar Belakang

Sistem informasi enterprise pada umumnya dibangun dengan pendekatan tradisional menggunakan pengembangan berbasis arsitektur monolitik, yaitu sebuah aplikasi terbungkus menjadi sebuah system yang besar, kemudian ketika terjadi perubahan pada salah satu bagian kode program dapat berpengaruh terhadap kode baris program lainnya. Saat ini pendekatan monolitik telah banyak bergeser menjadi pendekatan terdistribusi, yaitu aplikasi dibagi menjadi bagian kecil yang berfungsi secara spesifik *(microservices)* dan tidak bergantung pada komponen kode program lainnya (Blinowski et al., 2022). Perbedaan spesifik antara *monolith architecture* dan *microservices architecture* seperti pada Gambar 1.1.



Gambar 1.1 *Monolithic Architecture vs Microservice Architecture.*

Layanan sistem dan teknologi informasi (SI/TI) di lingkungan Universitas Islam Indonesia (UII), berada di bawah tanggung jawab pengelolaan Badan Sistem Informasi (BSI UII), dan telah melakukan transformasi digital pada sistem informasi yang ada di UII. Sistem informasi UII sebelumnya menggunakan arsitektur monolitik pada laman web https://unisys.uii.ac.id yang sudah beroperasi lebih dari 20 tahun namun masih belum memiliki pendefinisian yang jelas mengenai arsitektur data, proses, dan jaringan dalam

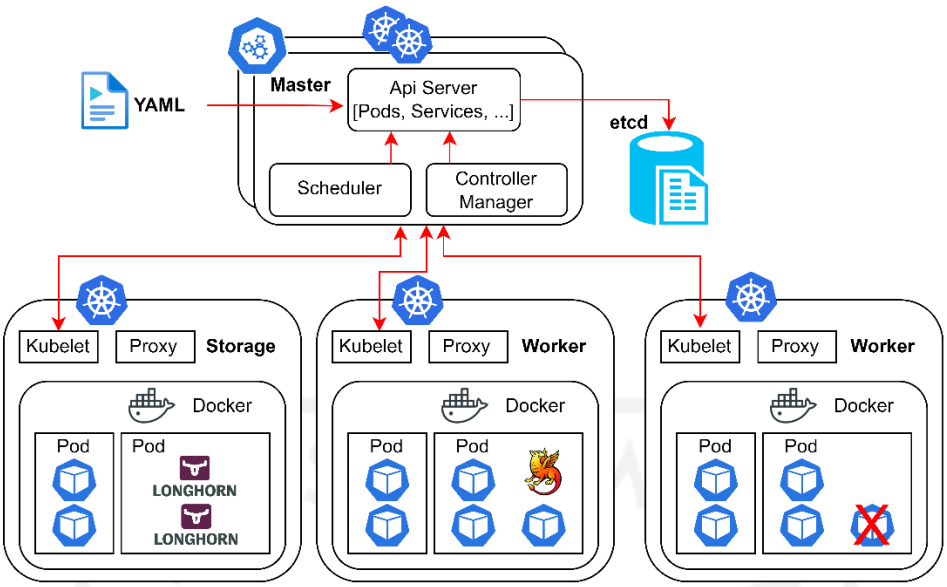
sistem informasi. Hal ini menyebabkan beberapa permasalahan diantaranya adalah standar data yang tidak jelas, sulitnya analisis pada pengembangan sistem yang berkelanjutan, sulit diintegrasikan dengan layanan lain, kurang tanggap dengan kebutuhan pengguna, serta tidak mampu menangani jumlah permintaan pengguna aplikasi yang semakin bertambah. Akibat dari permasalahan tersebut, Universitas Islam Indonesia tidak dapat melayani mahasiswa, orang tua mahasiswa, staff, dan dosen sebagai pengguna sistem dengan optimal. Sehingga mengakibatkan kebutuhan informasi yang tidak cepat, tidak terdapat redundansi data, sistem pelaporan yang tidak komprehensif serta terjadi pulau-pulau sistem antar bagian. Untuk merancang struktur tata kelola Teknologi Informasi (TI) yang efektif membutuhkan pemahaman, kekuatan bersaing dalam organisasi besar dan menciptakan harmoni antara tujuan bisnis, pola dasar tata kelola, dan tujuan kinerja bisnis. Struktur tata kelola TI yang efektif merupakan satu-satunya prediktor terpenting untuk mendapatkan nilai dari TI (Arbiansyah et al., 2010). Oleh karena itu, dengan tata kelola TI yang baik pada saat melakukan transformasi ini, akan sangat memungkinkan jika banyak sistem informasi yang terpisah akan dapat digabungkan menjadi layanan satu pintu yang *reliable*, dalam penelitian ini yaitu pada sistem enterprise UII Gateway. Sehingga bukan hal yang mustahil apabila sistem akademik dengan arsitektur monolitik yang menyebabkan sebuah sistem dengan yang lainnya terpisah, dapat mulai dimigrasikan ke dalam bentuk arsitektur *microservices*. Sehingga memungkinkan sistem yang awalnya terpisah menjadi beberapa layanan kecil, kemudian dapat dikumpulkan menjadi sebuah sistem satu pintu melalui laman web https://gateway.uii.ac.id.

Proses pengembangan *microservices* untuk komunikasi antar layanan dikembangkan melalui antarmuka *API* (*Application Programming Interface*) atau biasanya digunakan untuk komunikasi antara *client* dengan *server*. Penyederhanaan terhadap implementasi dan pengembangan aplikasi UII Gateway ini dengan menggunakan beberapa bahasa pemrograman seperti *php lumen, java, python, golang* sebagai *backend* untuk memproses dan megolah data, dan bahasa pemrograman *angular* sebagai *frontend* atau antarmuka pengguna. Dengan mengadopsi arsitektur sistem *microservices*, maka pada praktiknya hal ini telah menjawab beberapa tantangan dari arsitektur monolitik yang sudah tidak relevan dengan teknologi terbaru dan ketidakmampuannya beradaptasi terhadap kebutuhan sistem *(requirement changes)* terutama dalam pengelolaan kompleksitas kode program serta pemeliharaan sistem yang sulit. Pada penggunaan arsitektur sistem *microservices* juga dibutuhkan proses kontainerisasi layanan untuk mengoptimalkan penggunaan resource pada server yaitu dengan metode *load balancing*. Metode ini diperlukan untuk meminimalisir

adanya *bottleneck* pada layanan dan dapat membagi beban permintaan pengguna ke dalam server dapat terbagi secara merata. Dalam pengembangan sistem enterprise dapat didukung dengan penambahan proses automasi sehingga pengelolaan dan pengembangan sistem akan menjadi lebih adaptif. Salah satu teknologi yang menawarkan proses automasi pada saat pengembang telah selesai menulis program dan aplikasi siap di pasang ke server dengan menggunakan *GitLab Continuous Integration and Continuous Delivery* (*CI/CD*) *Pipeline*. *Continuous Integration* dapat dikatakan berhasil apabila setiap perubahan kode program terbaru pada aplikasi telah berhasil terbungkus, teruji, serta dapat langsung dioperasikan di server dalam bentuk *container* secara otomatis. Proses tersebut dapat menjadi solusi terbaik untuk meminimalisir adanya perubahan kode program pada server *production environment* secara langsung, dan dapat mempercepat proses pemasangan aplikasi ke dalam server karena berjalan secara otomatis, serta proses pengembangan menjadi lebih transparan pada sebuah *repository GitLab*.

Beberapa kerperluan seperti *cache* pada proses *continuous Integration* dapat membantu meminimalisir penggunaan *bandwidth internet*. Dengan menggunakan konsep *cache*, proses download pada kebutuhan sistem hanya perlu dilakukan sekali saja di awal pengembangan. Selama tidak terdapat pembaharuan dari *library* program, maka proses download ulang tidak perlu dilakukan. Hal ini menjadi salah satu kelebihan dari penggunaan *CI/CD* pada arsitektur *microservices*. Dengan mengusung konsep sistem informasi enterprise dengan arsitektur *microservice* dapat membentuk tim pengembangan menjadi lebih fokus dan spesifik pada sub layanan yang sedang dikembangkan. Sehingga permasalahan besar pada sebuah proses bisnis dapat dipecahkan menjadi beberapa solusi kecil yang disusun dalam suatu layanan, dimana setiap layanan memiliki tanggung jawab pada masing-masing tim pengembangan. Dengan pendekatan ini, maka sebuah sistem informasi akan terdiri dari beberapa layanan yang dapat dikelola dan didistribusikan secara independen, namun dapat dikelola menjadi sebuah aplikasi terpadu seperti pada aplikasi web UII Gateway.

Dalam mengaplikasikan arsitektur sistem *micoservice* menggunakan infrastruktur layanan terdistribusi ini akan menjadi solusi bagi banyak perusahaan yang memiliki sebuah proses bisnis yang rumit dan sistem tersebut diakses oleh banyak pengguna dalam satu waktu yang bersamaan. Salah satu upaya agar aplikasi *microservice* dapat berjalan dan saling berkomunikasi antar layanan yaitu dengan menjalankan aplikasi *microservice* diatas infrastruktur *cluster Kubernetes*. Untuk membangun sebuah *cluster Kubernetes* diperlukan beberapa kebutuhan seperti pada Gambar 1.2 dibawah ini.



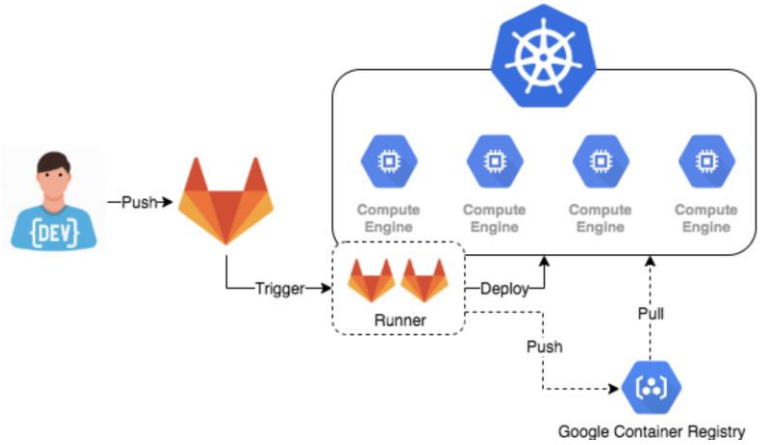
Gambar 1.2 Ekosistem *Cluster Kubernetes (K8s).*

Kubernetes berasal dari bahasa Yunani, yang berarti juru mudi atau pilot, dan merupakan asal kata gubernur dan *cybernetic*. *K8s* merupakan sebuah singkatan yang didapat dengan mengganti 8 huruf "ubernete" dengan "8" (Chakraborty, 2019). *Kubernetes* pada awalnya dikembangkan oleh *Google*, kemudian menjadi *opensource* sejak diluncurkan dan dikelola oleh komunitas kontributor yang besar (Prasetyo & Salimin, 2021). Seiring dengan adanya kebutuhan penggunaan komputasi yang besar dan fleksibel *Kubernetes* semakin banyak diminati oleh perusahaan IT, dengan ditambah dukungan dari berbagai perusahaan IT raksasa seperti *Microsoft, RedHat, IBM, Docker, AWS*, dan lain sebagainya. Kemudian pada tanggal 2 Maret 2018 Versi Beta Pertama *Kubernetes* 1.10 diumumkan. Hal ini menjadi menarik dari sisi infrastruktur teknologi informasi. Karena jika dilihat dari sejarahnya hingga sekarang, *Kubernetes* membentuk sebuah ekosistem yang membuat aplikasi web menjadi lebih fleksibel. Dimanapun lokasi server, tidak akan menjadi hambatan bagi perusahaan yang ingin mengaplikasikan sebuah sistem berbasis *micorservice* (Poniszewska-Marańda & Czechowska, 2021).

Berdasarkan Gambar 1.2 terdapat beberapa komponen pada *Kubernetes* yang saling melengkapi. Mulai dari server *Master* yang terdapat komponen *etcd* yaitu berfungsi sebagai penyimpanan *key value* konsisten yang digunakan sebagai penyimpanan data pada *cluster Kubernetes*. Komponen *control plane* yang bertugas mengamati *pod* ataupun *container* baru yang belum ditempatkan pada server *worker* manapun dan kemudian dapat secara otomatis dapat menempatkan *container* baru tersebut akan dijalankan. *Pod* sendiri merupakan representasi dari unit deployment sebuah *instance* aplikasi di dalam *Kubernetes*, dan dapat

terdiri dari satu kontainer atau sekumpulan kontainer yang saling berbagi sumber daya komputasi pada server.

Setelah *cluster Kubernetes* siap untuk digunakan, maka konsep pengembangan aplikasi *microservice* menjadi lebih sederhana seperti pada Gambar 1.3, yang menunjukkan bahwa *GitLab* sebagai repository penyimpanan terpadu pada pengembangan kode program aplikasi UII Gateway terintegrasi dengan *cluster K8s* dengan bantuan *gitlab runner* seperti pada Gambar 1.3.



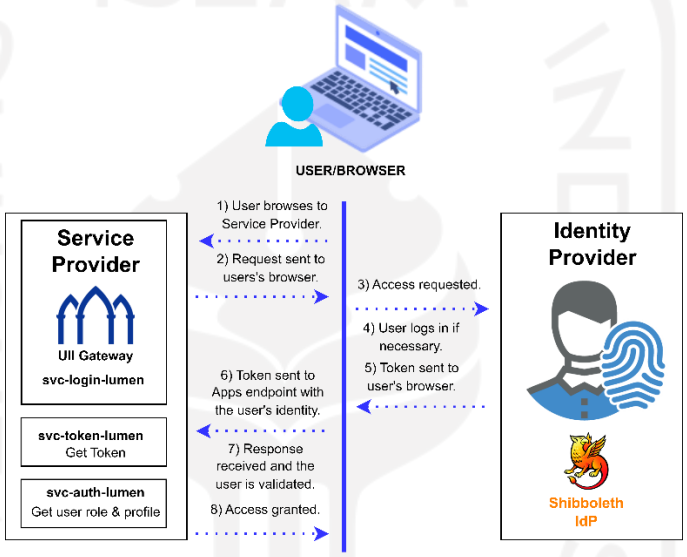
Gambar 1.3 Proses *Deploy* yang terintegrasi dengan *GitLab CI* dan *K8s*.

Berdasarkan pengalaman, pada saat layanan UII Gateway baru diluncurkan, terjadi beberapa kesalahan sehingga menyebabkan kegagalan proses login yang berakibat pada lumpuhnya proses bisnis layanan UII Gateway. Kesalahan tersebut antara lain yaitu belum adanya proses uji beban pengguna untuk menentukan besaran kebutuhan sumber daya server yang dibutuhkan pada layanan login. Serta belum menggunakan *persistent volume* terdistribusi menggunakan replikasi *Longhorn* dan belum digunakannya layanan *Shibboleth IdP* sebagai manajemen single sign on (SSO) untuk menggantikan proses autentikasi dan otorisasi layanan login aplikasi pada UII Gateway. Untuk menjawab tantangan tersebut, maka ditentukan *Shibboleth IdP* sebagai salah satu bagian dari arsitektur *microservice*, yang bertindak sebagai *Identity Provider* dan diintegrasikan dengan layanan login aplikasi UII Gateway menggunakan *Shibboleth Service Provider (SP)*.

Shiboleth IdP dalam penelitian ini dipilih menjadi sebuah solusi untuk menjawab tantangan dari adanya kendala pada layanan login sebagai gerbang utama pintu masuk aplikasi UII Gateway yang bertindak sebagai layanan autentikasi dan otorisasi. *Shibboleh IdP* dipilih karena fungsinya yang sangat lengkap, sesuai dengan kompleksitas dan kebutuhan dari layanan UII Gateway. Termasuk adanya dukungan untuk mengambil dan

menggabungkan beberapa sumber data seperti *database oracle*, *active directory*, dan juga *database percona mysql*.

Namun, penggunaan *Shibboleth IdP* dapat berakibat fatal apabila tidak diimbangi dengan proses kontainerisasi dan perhitungan sumber daya server yang dibutuhkan pada layanan. Karena *Shibboleth IdP* telah menjadi pusat manajemen *Single Sign On* pada proses login atau layanan *Authentication, Authorization, Accounting (AAA)*, salah satunya pada aplikasi UII Gateway. Untuk mengimplementasikannya, dibutuhkan beberapa komponen pendukung seperti *Shibboleth Service Provider (SP)* sebagai pengguna atau *consumer* dari layanan *Shibboleth IdP* seperti pada Gambar 1.4 dibawah ini:

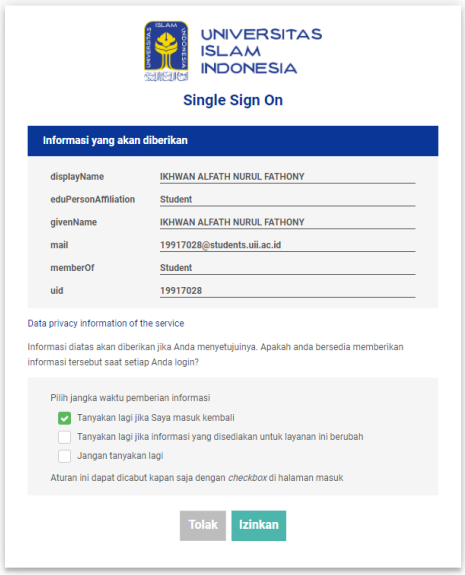


Gambar 1.4 *Flow* Login pada UII Gateway

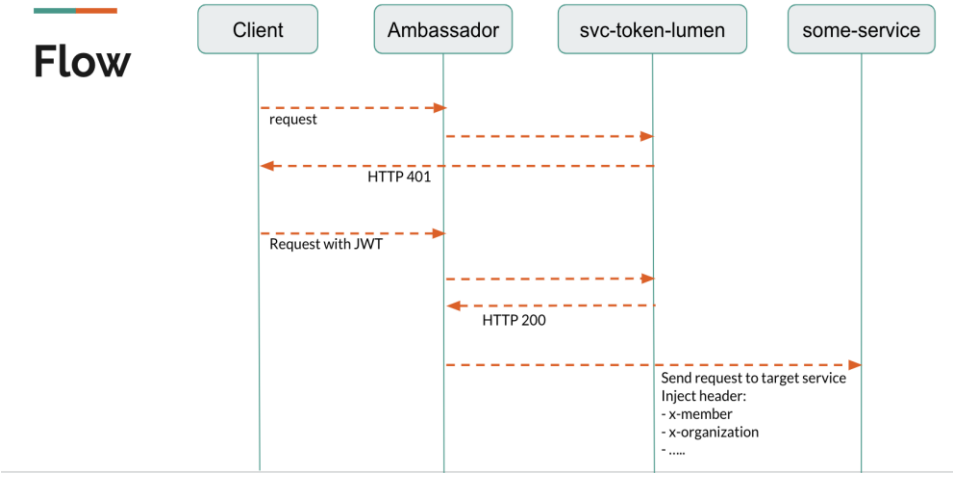
Berdasarkan gambar di atas terdapat alur login pengguna yaitu dengan menekan tombol login *SSO* pada halaman utama UII Gateway kemudian permintaan akan diteruskan ke *Shibboleth IdP* sebagai layanan autentikasi. Setelah itu kemudian pengguna layanan diminta untuk memasukkan username beserta password yang sesuai. Pada layanan login *Shibboleth SP* sebagai konsumer dari *Shibboleth IdP*, melakukan pengecekan *user* dan *password*. Apabila password yang dimasukkan benar, maka permintaan akan diteruskan pada sebuah form yang berisi *attribute* pengguna. Respon balik dari data atribut pengguna tersebut kemudian diteruskan menuju ke *service* token sesuai dengan hak akses yang melekat pada user tersebut.

Setelah proses authentikasi mendapatkan kode respon 200, alur proses akan dilanjutkan pada sebuah halaman persetujuan pengguna, untuk menyetujui atribut yang digunakan oleh *Shibboleth SP* untuk keperluan otorisasi pada layanan UII Gateway seperti

Gambar 1.5. Setelah pengguna mengizinkan penggunaan atribut pada halaman *Shibboleth IdP* pada laman web https://idp.uii.ac.id, maka permintaan pengguna akan dikembalikan ke halaman dashboard utama UII Gateway, dengan tampilan aplikasi yang sesuai dengan hak akses atau atribut yang melekat pada user pengguna.



Gambar 1.5 Form persetujuan penggunaan atribut user pada layanan login UII Gateway.



Gambar 1.6 *Flow Request Header* pada UII Gateway.

Awal mula layanan UII Gateway diluncurkan pada tahun 2019 untuk melakukan kegiatan berupa penginputan Rencana Studi atau Rencana Akademik Semester (key-in RAS) mahasiswa dengan jumlah akses ±1000 pengguna, namun layanan UII Gateway telah mendapatkan kendala yaitu terjadinya *downtime* pada layanan login, yang mengakibatkan mahasiswa tidak dapat melakukan login ke dalam Aplikasi UII Gateway, seperti Gambar 1.7

yang di keluhkan oleh pengguna UII Gateway dan memposting keluhannya pada aplikasi web twitter.

Gambar 1.7 Keluhan pengguna UII Gateway pada *twitter*.

Untuk mendapatkan solusi dari permasalahan tersebut, maka dilakukan penelitian dan uji coba lebih mendalam terhadap layanan *login,* yaitu dengan menjaga layanan *Shibboleth IdP* sebagai pusat dari proses autentikasi dan otorisasi agar tetap aktif walaupun terdapat gangguan pada salah satu perangkat server. Penggunaan *cluster Kubernetes* dan *persistent volume Longhorn* menjadi fokus pada penelitian ini. Perhitungan kebutuhan server *worker* yang disesuaikan dengan jumlah *pod* dan jumlah *resource server* yang dimiliki perlu dihitung sesuai dengan hasil uji beban dengan menggunakan alat yaitu *JMeter,* untuk mendapatkan nilai dari kebutuhan memori dan juga *cpu* pada setiap layanan, sehingga dari uji coba tersebut akan mendapatkan nilai jumlah replikasi dari layanan login untuk menjaga tingkat ketersediaan layanan agar tidak terjadi *downtime*. Kemudian untuk mengetahui tingkat ketahanan pada penyimpanan data yaitu dengan menggunakan alat bernama *fio,* sehingga dapat mengetahui performa dan resiko terjadinya *Single Point of Failure (SPOF)* dari infrastruktur penyimpanan pada *cluster Kubernetes*. Penulis juga melakukan *monitoring uptime* pada layanan *Shibboleth IdP*, untuk mengetahui tingkat keberhasilan dari adanya resiko *downtime* pada layanan login UII Gateway dengan menggunakan *Shibboleth IdP*, sebagai pusat autentikasi dan otorisasi pada sistem enterprise.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, permasalahan yang akan dijawab dan di bahas dalam tulisan ini yaitu, bagaimana kontainerisasi *Shibboleth IdP* sebagai pusat autentikasi dan otorisasi dari manajemen *Single Sign On* pada arsitektur *microservices* dapat menjadi

solusi dari permasalahan *downtime*, akibat dari adanya resiko *SPOF* pada layanan login UII Gateway. Dalam penelitian ini layanan autentikasi dan otorisasi pada sistem informasi enterprise akan di jalankan pada server *cluster Kubernetes* dan *persistent volume Longhorn* dengan metode *load balancing*.

1.3 Tujuan Penelitian

Tujuan penelitian yang dapat dilakukan pada tulisan ini yaitu dengan tiga pendekatan di bawah ini.

1. Untuk mengetahui solusi dari masalah *downtime* pada salah satu bagian *microservice* yaitu pada pusat layanan autentikasi dan otorisasi *Shibboleth IdP*. Dalam penelitian ini yaitu dengan mengkontainerisasikan *Shibboleth IdP* ke dalam server *cluster Kubernetes* dan *persistent volume Longhorn,* agar mampu menjalankan sistem *microservices* secara optimal karena dilengkapi dengan redundansi layanan, serta fleksibel dari sisi skalabilitas sistem enterprise UII Gateway dengan metode *load balancing*.
2. Untuk pengembangan aplikasi yang adaptif terhadap perubahan, dalam peneliatian ini juga mengimplementasikan automasi *GitLab CI* yang terintegrasi dengan *cluster Kubernetes*. Sehingga tidak membutuhkan *downtime* layanan, ketika terdapat perubahan terhadap *container Shibboleth IdP* sebagai pusat autentikasi dan otorisasi. Karena *Shibboleth IdP* secara otomatis memperbarui perubahan yang dilakukan oleh pengembang, serta otomatis dapat terpasang ke dalam *cluster Kubernetes* dan *persistent volume Longhorn.*
3. Memberikan rekomendasi strategi yang dapat diterapkan untuk meminimalisir adanya *downtime* pada sistem enterprise utamanya pada proses autentikasi dan otorisasi layanan. Dalam penelitian ini menggunakan *Shibboleth IdP* sebagai pusat manajemen *Single Sign On*, yang diintegrasikan dengan *Shibboleth SP* sebagai *consumer* dari layanan login UII Gateway.

1.4 Manfaat Penelitian

Hasil penelitian ini diharapkan dapat dijadikan sebagai bahan pertimbangan dan masukan bagi BSI dalam membangun layanan enterprise begitu juga dengan perusahaan ataupun organisasi yang sedang melakukan proses transformasi dari layanan monolitik menuju layanan *microservice* sebagai berikut.

1. Memperbaiki, serta mengevaluasi pemanfaatan teknologi yang digunakan untuk meningkatkan layanan sistem informasi enterprise dengan bentuk transparansi pada repositori *GitLab* di lingkungan Universitas Islam Indonesia.
2. Pengambilan keputusan mengenai langkah yang harus ditempuh dalam upaya menjaga *reliability* layanan sistem enterprise utamanya pada proses login sistem enterprise dalam penelitian ini *Shibboleth IdP* sebagai pusat autentikasi dan otorisasi pada manajemen *Single Sign On* layanan UII Gateway.
3. Mengoptimalkan penggunaan *cluster Kubernetes* dan *persistent volume Longhorn* dengan fungsi redundansinya, sehingga dapat mencegah terjadinya *downtime* akibat dari adanya resiko *Single Point of Failure* pada layanan yang di deploy ke dalam server *cluster Kubernetes*.

BAB 2

Tinjauan Pustaka

2.1 Kontainerisasi

Ketika suatu organisasi berupaya memindahkan aplikasinya ke layanan *Cloud Computing*, pilihan yang sering digunakan yaitu memanfaatkan proses virtualisasi. Dengan memanfaatkan mesin virtual (*Virtual Machine* atau *VM*), pengelola sistem dapat memindahkan infrastrukturnya dari Data Center *on-premises* ke layanan *Cloud Computing* melalui model *IaaS (Infrastructure as a Service)*, dengan mudah dan cepat.

Namun, pilihan virtualisasi bukan satu-satunya cara untuk menjalankan aplikasi pada teknologi *Cloud Computing*. Alternatif lain yang mulai mendominasi yaitu kontainerisasi, yang disebut sebagai salah satu tren *Cloud Computing* pada tahun 2021. Pada kontainer, aplikasi dikemas cukup dengan perangkat lunak pustaka *(library)* yang di sesuaikan dengan kebutuhan kode program.

Kontainerisasi sebelumnya sudah diramalkan akan menjadi pilihan bagi banyak perusahaan terkemuka di seluruh dunia. Sebuah survei yang diadakan tahun 2019 oleh lembaga periset pasar *Market Cube* menyebutkan bahwa 87% responden sudah menjalankan teknologi kontainer. Dari beberapa organisasi yang menggunakan teknologi kontainer, 90% menjalankannya di tahap produksi.

Gartner memprediksikan bahwa 75% perusahaan global akan menjalankan lebih dari dua aplikasi dalam kontainer pada tahun 2023 mendatang. Bandingkan dengan statistik pada tahun 2020 lalu, saat hanya 30% perusahaan global yang memanfaatkan kontainer (*Kontainerisasi, Salah Satu Tren Cloud Computing 2021*, n.d.).

2.2 Manajemen *Single Sign On*

Sebagai sebuah sistem layanan autentikasi, *Single Sign On* dapat menjadi peran untuk menanggulangi masalah lupa password serta mempermudah proses masuk ke suatu aplikasi (*How Does Single Sign-On (SSO) Work? | OneLogin*, n.d.). Dengan sistem ini, pengguna hanya perlu sekali login untuk masuk ke dalam beberapa aplikasi sekaligus. Caranya yaitu dengan mengidentifikasi pengguna secara ketat, setelah pengguna memasukkan kredensial, seperti nomor induk dan password. Sistem tersebut kemudian mengizinkan informasi kredensial digunakan dalam sekumpulan sistem terpercaya. Layanan ini dapat digunakan oleh organisasi, perorangan, atau bahkan perusahaan untuk memberikan

kemudahan dalam pengaturan *username* dan *password*. Terdapat salah satu alat yang dapat diimplemantasi pada layanan SSO yaitu *Shibboleth* (Sinnott et al., 2006).

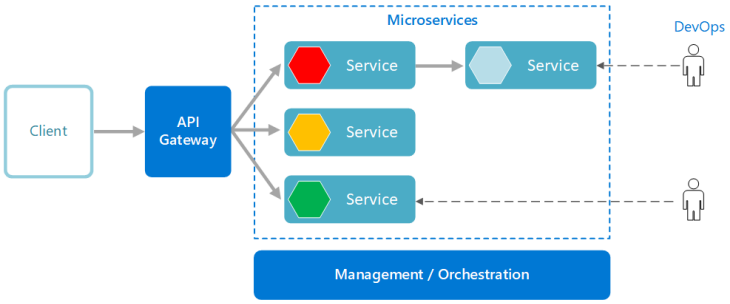
Shibboleth didasarkan pada protokol permintaan otentikasi berbasis *HTTP* sederhana dan kemudian diubah menjadi *SAML* 1.1 sebagai protokol *SP* pertama. Protokol tersebut pertama kali diimplementasikan di *Shibboleth IdP* 1.0, kemudian seiring waktu ditingkatkan pada *Shibboleth IdP* 1.3. *Liberti Alliance* memperkenalkan protokol Otentikasi dan permintaan yang sepenuhnya diperluas ke dalam kerangka Federasi Identitas *Liberty*. Berdasarkan peningkatan ini *Shibboleth* sekarang menjadi perangkat lunak yang digunakan oleh lebih dari 15 juta pengguna di seluruh dunia. Untuk mendukung basis pengguna yang luas, memungkinkan *Shibboleth* untuk terus berevolusi dan berkembang dalam lanskap manajemen identitas.

Shibboleth IdP menjadi pilihan teknologi berdasarkan fitur dan juga fungsinya yang sangat fleksibel terhadap kebutuhan data pengguna. *IdP* menyediakan *SAML* & *attribute Assertion* yang dapat di konfigurasi berdasarkan kelas objek yang telah di standarisasi seperti *eduPerson, inetOrgPerson*. Manajemen attribut dibagi dalam dua langkah, yang pertama yaitu *attribute resolver* yaitu dengan mengambil data *attribute* dari sumber eksternal seperti *LDAP*, database *(MySQL, Oracle, etc)*, dan *attribute* dapat dikonversi serta di petakkan dengan *URNs* yang sudah di standarisasi. Kemudian yang kedua yaitu dengan *attribute* yang disesuaikan dengan kebutuhan aplikasi atau *Shibboleth SP* sebagai *consumer* dari *SSO*, sehingga memungkinkan skenario filter yang kompleks sesuai dengan kebutuhan *Shibboleth SP* (KRISTAN W B, 2007).

2.3 Architecture Microservice

Arsitektur *microservices* merupakan jenis arsitektur aplikasi yang dikembangkan sebagai kumpulan layanan dan membentuk ekosistem enterprise, dengan menyediakan kerangka kerja untuk mengembangkan, menyebarkan, dan memelihara diagram pada *microservice* secara mandiri (*What Is Microservices Architecture? | Google Cloud*, n.d.).

Dalam arsitektur *microservices,* setiap layanan mikro merupakan layanan tunggal yang dibangun untuk mengakomodasi fitur aplikasi dan menangani tugas-tugas terpisah. Setiap layanan mikro berkomunikasi dengan layanan lain melalui antarmuka sederhana untuk menyelesaikan masalah pada proses bisnis (*Microservice Architecture Style - Azure Architecture Center | Microsoft Docs*, n.d.). Sebagai gambaran daripada layanan yang mengimplementasikan arsitektur *microservice,* sebagai berikut ini:



Gambar 2.1 Manajemen pada *microservice*.

Dengan pemisahan aplikasi berdasarkan fungsi-nya ini, pada akhirnya akan menemui keragaman teknologi dalam sebuah layanan digital. Dalam hal ini, terdapat layanan *Shibboleth IdP* yang memiliki *basic* pemrograman *java*. Pada bagian fungsi Otorisasi menggunakan bahasa pemrograman *PHP* serta untuk database dengan menggunakan *Percona mysql*. Dalam praktiknya terlihat bahwa setiap fungsi / permasalahan teknis dapat diselesaikan dengan cara dan teknologi yang berbeda-beda.

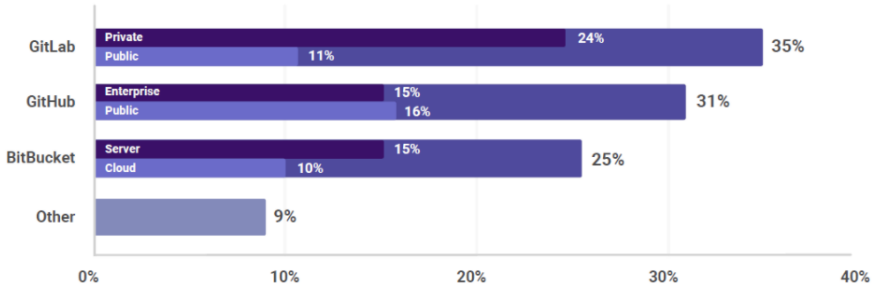
Dalam konsep *microservices*, pada penulisan tesis ini tidak hanya melalukan pemisahan di level aplikasi saja, tetapi juga dari sisi infrastruktur layanan server. Dalam implementasinya memungkinkan untuk ditemukan keragaman arsitektur infrastruktur, konfigurasi dan optimalisasi sistem yang berbeda, serta sering pula ditemukan jumlah spesifikasi *server* yang tidak sama antara layanan yang satu dengan yang lainnya.

2.4 Automasi Gitlab CI

Dengan metode pengembangan perangkat lunak berkelanjutan, tim pengembangan terus membangun, menguji, dan mengimplementasikan perubahan kode secara berulang. Proses berulang ini dapat meminimalisir bagi pengembang aplikasi untuk menulis kode baru berdasarkan bug atau versi sebelumnya yang gagal. Dengan metode ini, organisasi dapat berusaha untuk mengurangi intervensi manusia atau bahkan tidak ada intervensi sama sekali, mulai dari pengembangan kode baru hingga penerapannya.

GitLab merupakan *Platform DevOps* yang disajikan sebagai aplikasi tunggal. Ini membuat *GitLab* unik dan menciptakan alur kerja perangkat lunak yang disederhanakan, untuk mencapai tujuan dari sebuah organisasi dalam mengembangkan serta menyatukan aplikasi. *GitLab* juga merupakan proyek *open source* yang dikelola oleh *GitLab Inc* dengan lebih dari 3.000 kontributor. Sebuah organisasi dapat menginstal dan mengelola sendiri Edisi Komunitas *GitLab* yang sepenuhnya *open source* di bawah lisensi *MIT*.

Berdasarkan statistik pengguna *GitLab* telah mencapai rata-rata pengguna tertinggi yaitu sebesar 35%, pada tahun 2019 yang menjadikan *GitLab* semakin popular di kalangan organisasi IT (*GitLab vs GitHub: Which Is Right for You - Spectral*, n.d.).

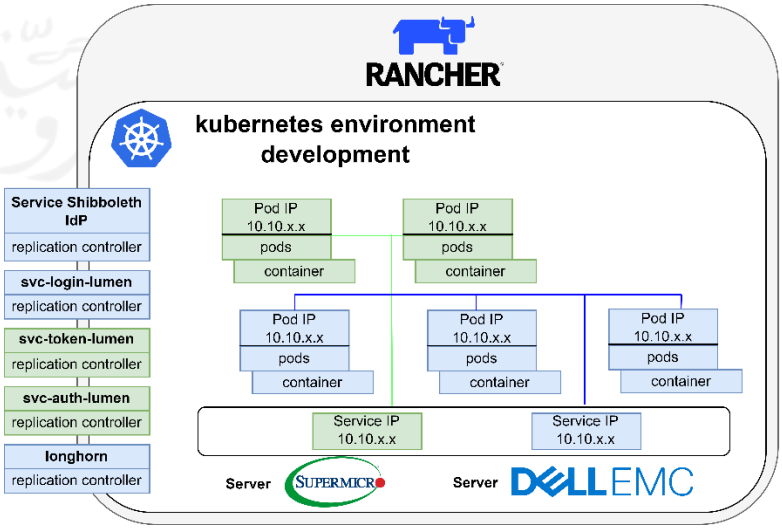


Gambar 2.2 Statistik pengguna pada *repository* program.

2.5 Rancher Kubernetes Engine

Rancher pada mulanya dikenal pada tahun 2014 dengan menyediakan layanan yang dapat digunakan oleh organisasi ataupun enterprise yang sedang membangun aplikasi menggunakan kontainer. Dengan hadirnya *Rancher* yang berjalan dalam komunitas *Docker* dapat menjawab pertanyaan tentang pengelolaan, pengamanan, serta mengatur kontainer yang berjalan di atas *cluster Kubernetes* (Schairer & Architect, n.d.).

Dengan kompleksitas *platform Kubernetes* akan berjalan mulus apabila diimbangi dengan pengelolaan arsitektur *infrastruktur* yang memadai. Salah satu cara agar *Kubernetes* lebih mudah dikelola yaitu di orkestrasi dengan menggunakan *software Rancher Kubernetes Engine (RKE)*, maka *cluster Kubernetes* akan seperti pada Gambar 2.3 dibawah.



Gambar 2.3 *Rancher Kubernetes Engine* Environment.

Rancher Kubernetes Engine (RKE) merupakan sebuah *software* yang dapat mengorkestrasi, mendistribusikan dan memanage *cluster Kubernetes* bersertifikat *The Cloud Native Computing Foundation (CNCF*) yang beroperasi sepenuhnya di atas *Docker container*. Dengan menggunakan *RKE*, pemasangan dan pengoperasian *Kubernetes* disederhanakan secara otomatis dan sepenuhnya independen dari sistem operasi serta *platform* yang berjalan di atas *Docker container* (Kusnetzky, 2019).

Pada *Kubernetes* terdapat fitur yang dapat mengatur *workload* secara otomatis sesuai dengan kebutuhan salah satunya yaitu dengan *Horizontal Pod Autocaler (HPA)*. *Horizontal scaling* yang berarti bahwa respons terhadap peningkatan beban akan diiringi dengan proses replikasi pada *pod*, sesuai dengan konfigurasi yang telah di tetapkan berdasarkan perhitungan seperti penggunaan *CPU* rata-rata, penggunaan memori rata-rata, atau metrik kusus lainnya. Dari perspektif dasar *HPA* beroperasi pada rasio antara nilai metrik yang diinginkan dan nilai metrik saat ini, bersarkan rumus (Özer, 2020).

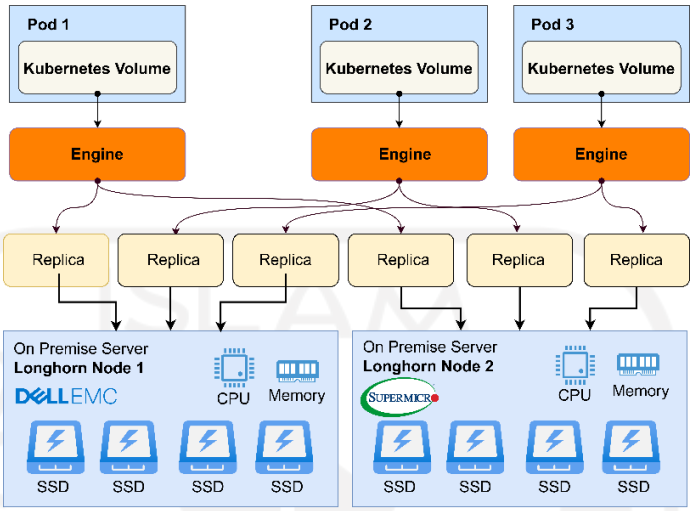
$$desiredReplicas = ceil[currentReplicas * (currentMetricValue / desiredMetricValue)]$$

Dalam hal ini desiredReplicas yang berarti hitungan replica yang akan dikirim ke *controller* setelah perhitungan, ceil() merupakan fungsi yang membulatkan bilangan pecahan ke atas misalkan ceil(12.18) menjadi 13, *currentReplicas* merupakan jumlah pod saat ini yang sedang berjalan, *currentMetricValue* yaitu nilai metrik saat ini yang digunakan sebagai acuan replikasi, dapat berupa *CPU, RAM*, atau metrik kusus dapat berupa *event* per detik, dll; sedangkan *desiredMetricValue* yaitu metrik yang telah di atur menjadi batas pada *HPA* dengan nilai yang tidak boleh terlalu rendah ataupun terlalu tinggi.

2.6 Storage Longhorn

Longhorn merupakan proyek pengembangan aplikasi *open source* resmi dari *Cloud Native Computing Foundation (CNCF)*. Longhorn merupakan salah satu platform penyimpanan terdistribusi *cloud-native* untuk *Kubernetes* yang andal. Longhorn sebagai *persistent block storage* didalam kluster *Kubernetes* dengan fitur replikasi membuatnya terlihat sederhana, cepat, dan andal. Blok *storage* ini berkomunikasi melalui protokol *iSCSI* dengan perangkat keras *server storage*. Sangat penting untuk memperhatikan penggunaan perangkat keras *server storage* dan konfigurasi *server* jaringan, karena keduanya dapat

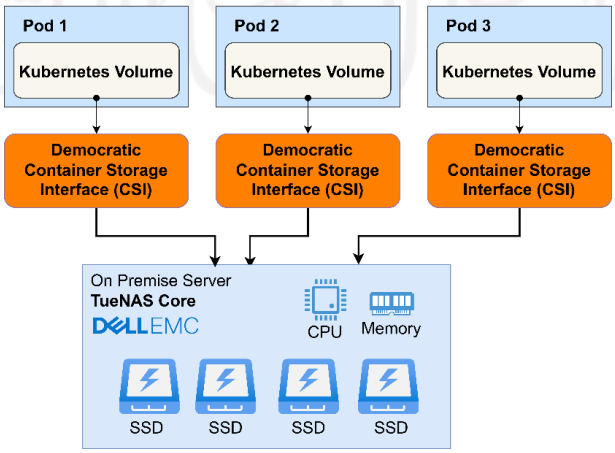
berdampak signifikan pada pengguna saat menggunakan *Longhorn*. Arsitektur *Longhorn* direpresentasikan pada Gambar 2.4 dibawah ini.



Gambar 2.4 Arsitektur *Block Storage Longhorn*.

2.7 Storage TrueNAS Democratic CSI

Network-Attached Storage (NAS) merupakan salah satu jenis penyimpanan data server yang terhubung ke jaringan dan menyediakan akses data ke berbagai klien. Sistem *NAS* merupakan peralatan jaringan yang berisi satu atau lebih *hard drive* dan dapat dilakukan *RAID*, apabila terdapat *RAID Controller* pada perangkat server yang mampu menjaga redundansi dalam satu server fisik. Namun, itu tidak dapat mereplikasi pada perangkat keras server yang terpisah. Pada penyimpanan yang terhubung ke jaringan menyediakan akses data yang lebih cepat, administrasi yang lebih mudah, dan konfigurasi yang sederhana melalui *TrueNAS* dan harus menggunakan *Container Storage Interface (CSI)* Demokratik untuk berkomunikasi dengan *Kubernetes*.



Gambar 2.5 Arsitektur *Storage TrueNAS Democratic CSI*.

2.8 Load balancer

Load balancer merupakan salah satu metode untuk membagi beban *resource* pada server dengan perangkat lunak yang bertindak sebagai *reverse proxy* atau mendistribusikan lalu lintas jaringan pada aplikasi ke sejumlah server. *Load balancer* digunakan untuk meningkatkan kapasitas (*concurrent user*) dan *reliability* pada aplikasi. Dengan proses *load balancing* dapat meningkatkan kinerja aplikasi secara keseluruhan dengan membagi beban pada server, pemeliharaan sesi aplikasi dan jaringan.

Load balancing umumnya dikelompokkan ke dalam dua kategori: layer 4 dan layer 7. *Load balancer* layer 4 bekerja berdasarkan data yang ditemukan dalam protokol lapisan jaringan dan transportasi *(IP, TCP, FTP, UDP)*. *Load balancer* layer 7 mendistribusikan *request* berdasarkan data yang ditemukan dalam protokol layer aplikasi seperti *HTTP*. Dalam penelitian ini akan dibahas *load balancer* layer 7 dengan menggunakan software *NGINX*.

Pada proses pembagian beban *request* yang diterima oleh *load balancer* kemudian didistribusikan ke server tertentu berdasarkan algoritma yang telah di atur pada konfigurasi file NGINX. Beberapa algoritma standart industri yaitu *Round robin*, *Weighted round robin, Least connections, Least response time*. *Load balancer* layer 7 selanjutnya dapat mendistribusikan *request* berdasarkan data spesifik aplikasi seperti *header HTTP*, *cookie*, atau data dalam pesan aplikasi itu sendiri seperti nilai parameter tertentu. *Load balancer* memastikan *reliability* dan *availability* dengan memantau respon dari aplikasi dan hanya mengirimkan permintaan ke server apabila aplikasi dapat merespon secara tepat waktu.

BAB 3
Metodologi

Pada Bab III ini akan dijabarkan mengenai metodologi penelitian yang diusulkan. Penelitian ini akan dibagi dalam enam tahapan, diantaranya studi literatur, perancangan penelitian, implementasi penelitian, uji coba dan analisis penelitian serta penulisan laporan berupa paper.

3.1 Perumusan Masalah

Masalah yang dihadapi dalam penelitian ini yaitu pengembangan layanan login sistem enterprise UII Gateway menggunakan *Shibboleth IdP* sebagai pusat manajemen *Single Sign On (SSO)*. Untuk menjawab tantangan kebutuhan pengguna yang terus bertambah, kustomisasi kontainerisasi pada *Shibboleth IdP* dapat menjadi salah satu solusi yang dapat diambil untuk menunjang kebutuhan login pada layanan enterprise. Strategi pemilihan infrastruktur dengan mengadopsi teknologi *Rancher Kubernetes Engine (RKE),* dapat menjadi pilihan untuk menjawab tantangan dari beban pengguna yang semakin meningkat. Untuk mengoptimalkan penggunakan resource server dengan *RKE*, dalam penelitian ini melakukan perhitungan kebutuhan resource layanan login dengan konsep *Horizontal Pod Autoscaling (HPA)* serta menggunakan metode *load balancing* dengan algoritma *round robin,* agar layanan dapat menerima permintaan pengguna secara merata. Namun setelah mengimplementasikan *Kubernetes* saja tidak cukup, karena masih terdapat risiko terjadinya *Single Point of Failure* pada penggunaan *storage*. Pada penelitian ini mencoba menawarkan solusi replikasi storage dengan menggunakan *Longhorn* seperti pada Gambar 2.4.

3.2 Studi Literatur

Tahapan ini merupakan tahapan untuk mencari referensi pembelajaran yang dapat dipertanggungjawabkan yaitu referensi yang bersumber dari jurnal, artikel, website ataupun buku. Referensi ini dapat berguna untuk menambah wawasan dari penelitian terdahulu untuk menunjang penelitian yang dapat bermanfaat di kemudian hari. Topik referensi yang terkait pada penelitian ini diantaranya mengenai implementasi kontainer *Shibboleth IdP* dan terintegrasi dengan layanan login UII Gateway menggunakan automasi *GitLab CI*, perhitungan *HPA* pada penerapan layanan arsitektur *microservices* dengan *Rancher*

Kubernetes Engine dan *persistent volume Longhorn* berdasarkan pengujian performa menggunakan *JMeter* dengan metode *load balancing*.

3.3 Desain Sistem

Arsitektur sistem komputasi cluster yang diusulkan untuk penelitian ini secara umum sesuai dengan Gambar 2.3 yaitu dengan memecah layanan menjadi bentuk yang lebih kecil *(microservice)*, termasuk dalam mengimplementasikan *Shibboleth IdP* sebagai manajemen *Single Sign On* yang terintegrasi dengan layanan login menggunakan metode *load balancing* pada sistem enterprise *microservice architecture*.

3.4 Langkah-langkah Penelitian

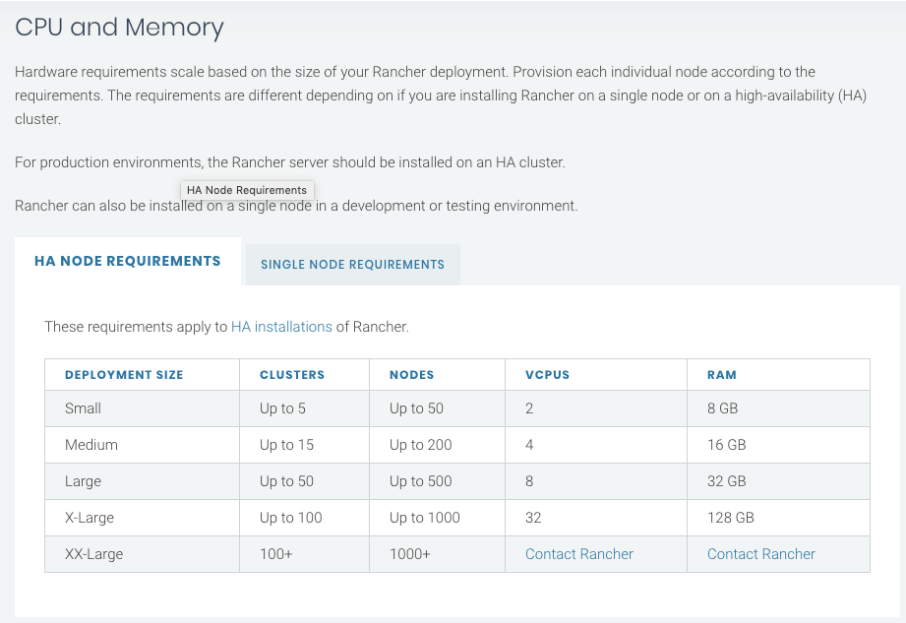
Tahapan implementasi merupakan proses eksekusi rancangan penelitian. Pada tahap implementasi ini, perlu diketahui spesifikasi perangkat keras dan perangkat lunak yang akan digunakan. Perangkat keras yang digunakan merupakan server *Nutanix*, *supermicro*, dan juga *DELL* untuk keperluan virtualisasi OS dengan menggunakan beberapa *Virtual Private Server* yang nantinya digunakan sebagai *cluster Kubernetes* pada *Rancher Kubernetes Engine*. Selain itu untuk kebutuhan *persistent storage* pada *cluster Kubernetes* yaitu dengan mesin *supermicro* menggunakan *storage* berbasis *nvme*, kemudian dalam penelitian ini juga membutuhkan beberapa perangkat lunak pendukung dengan spesifikasi sebagai berikut:

- Sistem Operasi Ubuntu 18.04
- Docker versi 20.10.9
- Kubernetes versi 1.21.5
- Rancher Kubernetes Engine versi 2.6.1
- GitLab CE 11.10.4
- Gitlab Runner versi 0.29.0
- Shibboleth IdP versi 4.0.1
- NGINX 1.20.1
- JMeter

Adapun tahap-tahap implementasi dari penelitian dijelaskan sebagai berikut.

1. Pembuatan *Rancher Kubernetes Engine (RKE)*

   Pada pembuatan *cluster RKE*, perlu menyesuaikan kebutuhan resource server yang akan digunakan. Sebagai pedoman pada pembuatan cluster RKE tertera pada gambar 3.1 dibawah ini.



CPU and Memory

Hardware requirements scale based on the size of your Rancher deployment. Provision each individual node according to the requirements. The requirements are different depending on if you are installing Rancher on a single node or on a high-availability (HA) cluster.

For production environments, the Rancher server should be installed on an HA cluster.

HA Node Requirements

Rancher can also be installed on a single node in a development or testing environment.

HA NODE REQUIREMENTS | SINGLE NODE REQUIREMENTS

These requirements apply to HA installations of Rancher.

DEPLOYMENT SIZE	CLUSTERS	NODES	VCPUS	RAM
Small	Up to 5	Up to 50	2	8 GB
Medium	Up to 15	Up to 200	4	16 GB
Large	Up to 50	Up to 500	8	32 GB
X-Large	Up to 100	Up to 1000	32	128 GB
XX-Large	100+	1000+	Contact Rancher	Contact Rancher

Gambar 3.1 *Hardware requirements* pada *Rancher Kubernetes Engine*.

Dalam penelitian ini menggunakan pedoman *deployment size Medium*, yaitu dengan meggunakan 4 *Core CPU* dan 16 *GB RAM* dapat mengorkestrasi *cluster Kubernetes* hingga 15 *cluster*.

2. Pembuatan Klaster *Kubernetes*

   Berbeda dengan jumlah *resource* yang digunakan oleh *cluster RKE*. Dalam pembuatan klaster *Kubernetes* dilengkapi dengan *persistent storage Longhorn* ini, perlu di sesuaikan dengan kebutuhan pada setiap cluster. Dalam penelitian ini akan dibuatkan *cluster staging* untuk melakukan *testing* pada layanan *AAA* untuk menjawab kegagalan proses login UII Gateway sebelum dan sesudah terintegrasi dengan *Shibboleth IdP*.

3. Pembuatan image *container Shibboleth IdP*

   Dalam pembuatan image *container Shibboleth*, terdapat beberapa konfigurasi yang harus disesuaikan dengan kebutuhan layanan. Setelah *Shibboleth* disesuaikan kemudian perlu proses integrasi antara *Shibboleth IdP* dengan *Shibboleth SP* pada layanan login, yaitu dengan mendaftarkan *Shibboleth SP* menjadi bagian dari *Shibboleth IdP* agar dapat mengkonsumsi data pengguna dari *Shibboleth IdP* yang terintegrasi dengan sumber data seperti dijelaskan pada bab 2.2.

4. Konfigurasi *Load balancing*

   Metode *load balancing* disini menggunakan *nginx* dengan algoritma *round robin* yaitu dengan mengarahkan akses pada *backend node* yang memiliki beban kecil

berapapun jumlah koneksinya. Keunggulan *round roubin* ini dapat membagi beban akan merata ke masing-masing *virtual machine* atau *node worker* pada *cluster Kubernetes*. Namun ada kelemahan ketika diimplementasikan pada *Shibboleth IdP* yaitu adanya potensi *client* terputus sessionnya karena pertama akses login session mendapatkan *request* ke *node* pertama, namun saat akses layanan ke halaman lain boleh jadi mendapatkan request ke node kedua atau ketiga dimana node tersebut tidak tersimpan file sessionnya. Hal tersebut dapat di atasi dengan mengimplementasikan *IP Hash* atau *remote_addr* pada metode *load balancing* nginx dengan rumus *hashing* tertentu, sehingga mendapatkan penentuan *persistent client* yang akan mengarah pada node tertentu. Hal tersebut akan lebih menjamin pengguna untuk tidak kehilangan session pada aplikasi khususnya *Shibboleth IdP* sebagai manajemen *Single Sign On*.

5. Testing Layanan

   Setelah layanan *Shibboleth IdP* sebagai manajemen *Single Sign On* terintegrasi dengan layanan *AAA* pada sistem enterprise UII Gateway. Agar layanan dapat berjalan sesuai dengan kebutuhan pengguna, berbagai aspek pengujian seperti penggunaan *resource server* dengan jumlah *request* pada layanan *service* login yang sudah terintegrasi dengan *Shibboleth IdP*, perlu di lakukan untuk memastikan layanan *SSO* tetap *reliable*. Hasil uji coba ini akan berguna bagi perusahaan ataupun organisasi lain yang menggunakan SSO pada sistem pendukung proses bisnisnya. Ketika seluruh layanan telah terintegrasi menggunakan SSO dalam hal ini dengan *Shibboleth IdP*, menjadi pertimbangan utama untuk menjaga stabilitas sistem agar tidak terjadi *single point of failure* pada layanan SSO. Seperti pada penelitian ini, *Shibboleth IdP* digunakan oleh layanan enterprise UII Gateway, dan juga layanan lain yang berada di bawah organisasi Universitas Islam Indonesia. Sehingga pemilihan teknologi dan implementasi perhitungan *resource server* perlu di hitung dengan beberapa skenario sesuai dengan kompleksitas, sehingga tidak terjadi *downtime* pada layanan *Shibboleth IdP* pada halaman web https://idp.uii.ac.id dengan dibuktikan pada *monitoring uptime* layanan.

BAB 4
Hasil dan Pembahasan

Tahap pengujian dilakukan untuk menguji kebenaran dari hipotesa serta menjawab pertanyaan penelitian yang diuraikan pada BAB 2 dan sebagai bahan pertimbangan untuk menjawab tantangan adanya keluhan pada *twitter* oleh beberapa pengguna UII Gateway sebelum aplikasi login menggunakan SSO seperti pada gambar 1.7 maka peneliti mencoba untuk menambah alur service login dengan menggunakan *SSO Shibboleth IdP* seperti pada Gambar 4.1 dibawah ini.

Gambar 4.1 *Registrasi SSO UII Gateway* menggunakan *Shibboleth IdP*

4.1 Tahapan Penelitian

Penelitian ini dilakukan pada infrastruktur uji coba dengan konfigurasi *resource* server yang mirip dengan klaster *Kubernetes production*. Secara umum, peneliti melakukan perhitungan terhadap kebutuhan *resource* pada layanan yang dijalankan pada klaster *Kubernetes*. Dalam hal ini layanan login sebagai *SP* pada layanan *AAA* yang terintegrasi dengan *Shibboleth IdP* menjadi pedoman testing melalui beberapa tahapan sebagai berikut:

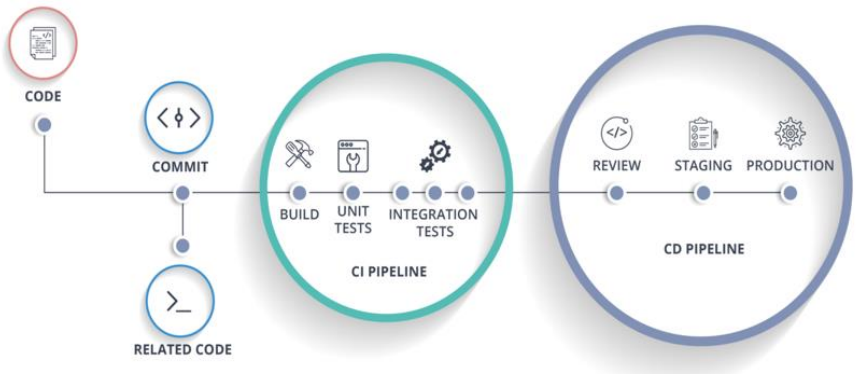
1. Tahap pertama merupakan tahap perancangan dan implementasi sistem diawali dengan melakukan kontainerisasi *Shibboleth IdP* sebagai pusat autentikasi pada *Single Sign On*, yang digunakan oleh aplikasi UII Gateway dan terintegrasi dengan layanan sistem enterprise *AAA* pada klaster *Kubernetes* dan *persistent volume Longhorn*, menggunakan metode *load balancer Nginx*.
2. Tahap kedua yaitu melakukan testing dengan *fio* pada persistent storage yang digunakan di klaster *Kubernetes*, untuk meminimalisir adanya resiko *SPOF*.
3. Tahap ketiga yaitu melakukan *load testing* pada layanan autentikasi dalam hal ini yaitu layanan login yang terhubung dengan *Shibboleth SP*, dan apabila pengguna berhasil memasukkan user dan password pada alur autentikasi akan mendapatkan *response code* 200, maka alur proses akan di lanjutkan pada layanan token untuk membuat token yang melekat pada user pengguna.

4.2 Implementasi Sistem

4.2.1 Kontainerisasi Shibboleth IdP

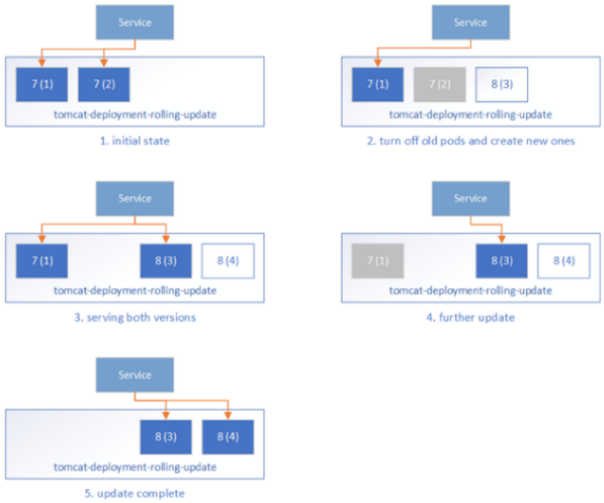
Langkah pertama dalam penelitian ini yaitu dengan melakukan kustomisasi pada layanan *Shibboleth IdP*, agar dapat di *deploy* ke dalam infrastruktur klaster *Kubernetes* dengan mengimplementasikan *Docker* berbasis *Ubuntu* server versi *18.04 LTS (Focal Fossa)* sebagai *base image*, bersama dengan *load balancer Nginx* sebagai *reverse proxy.* Sehingga klien dapat mengakses domain server secara publik melalui *node Port* dari *Shibboeth IdP* yang ada di dalam *cluster Kubernetes*. Salah satu hal penting dalam proses *proxy* ini adalah *sticky session* pada *Nginx*, yaitu untuk menempatkan sesi pengguna pada *node worker Kubernetes* secara permanen, sehingga sesi otentikasi dan *request* dari pengguna dapat diteruskan ke *node worker* dengan alamat *IP* yang sama. Beberapa pertimbangan diperlukan untuk memigrasikan layanan dari virtualisasi ke kontainerisasi, termasuk penggunaan resource yang lebih optimal karena dapat digunakan secara fleksibel dan bersamaan dengan aplikasi lain dalam klaster *Kubernetes*.

Untuk mempercepat pengembangan integrasi *Service Provider (SP)* dengan layanan *svc-login-lumen* pada sistem enterprise serta meningkatkan keamanan, *high availability*, dan untuk meminimalkan *downtime* layanan kontainerisasi pada sistem *Shibboleth IdP* sebagai pusat manajemen *Single Sign On (SSO)*, maka beberapa proses sistem otomatis juga dibangun berdasarkan kolaborasi *GitLab* menggunakan konsep *Continuous Integration/ Continuous Deployment (CI/CD)*. Integrasi *GitLab* dengan *Kubernetes Cluster* dapat membuat proses pengembangan transparan, dan integrasi layanan SP dapat langsung terhubung dengan proses pengujian pada *staging environment*. Ketika integrasi telah berhasil melalui pengujian, maka dapat diteruskan ke *production environment* berdasarkan *trigger tagging* pada *CI/CD*, seperti yang disajikan pada Gambar 4.2.



Gambar 4.2 *CI/CD Pipeline* melalui *GitLab CI* Universitas Islam Indonesia.

Untuk menjaga tingkat ketersediaan layanan, penulis menggunakan strategi *Rolling Update Strategy* untuk menerapkan penerapan *zero-downtime* di dalam *cluster Kubernetes*, yaitu secara bertahap menggantikan *pod* lama dengan yang baru tanpa adanya *downtime* pada layanan. Misalnya, jika *Tomcat* yang berjalan pada *container Shibboleth IdP* yaitu versi 7 kemudian akan dilakukan pembaruan versi dengan *Tomcat* versi 8, maka proses tersebut tidak menimbulkan *downtime* pada layanan *Shibboleth IdP*. Untuk lebih detail mengenai proses penggantian versi layanan seperti pada Gambar 4.3 dibawah ini.



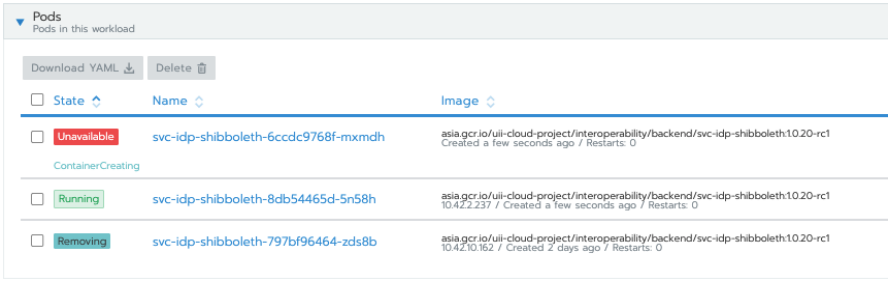
Gambar 4.3 Konsep *Zero Downtime* pada *Deployment Kubernetes*

- Awalnya, semua *pod* menjalankan *Tomcat 8* dan layanan *frontend* mengarahkan lalu lintas ke *pod* ini.
- Selama pembaruan sedang berlangsung, *Kubernetes* menghapus beberapa *pod Tomcat 8* dan membuat *pod Tomcat 9* baru yang sesuai. Hal tersebut untuk

memastikan banyak *pod* yang di izinkan untuk dapat di hapus atau *maxUnavailable* pada *Pod*, setidaknya pada *pod* dengan jumlah replika – *maxUnavailable,* harus melayani lalu lintas pengguna, dalam kasus penelitian ini yaitu 2-1=1. Kemudian paling banyak atau *maxSurge* banyaknya *pod* yang dapat dibuat selama proses pembaruan, yaitu 2*50%=1.

- Satu *pod Tomcat 8* secara otomatis di hapus, dan *satu pod Tomcat 9* dibuat. *Kubernetes* tidak akan mengarahkan lalu lintas ke salah satu dari mereka karena terdapat pemeriksaan kesiapan *pods* yang belum berhasil untuk dibuat.
- Saat *pod Tomcat 9* baru sudah siap, *Kubernetes* akan mulai merutekan lalu lintas ke *pod* tersebut. Ini berarti selama proses pembaruan, pengguna dapat melihat layanan lama dan layanan baru.
- Pembaruan berlanjut dengan menghapus *pod Tomcat 8* dan membuat *pod Tomcat 9*, lalu mengarahkan lalu lintas ke *pod* yang sudah siap digunakan.
- Akhirnya, semua *pod* ada di *Tomcat 9* sesuai dengan jumlah reprikasi yang telah di tentukan.

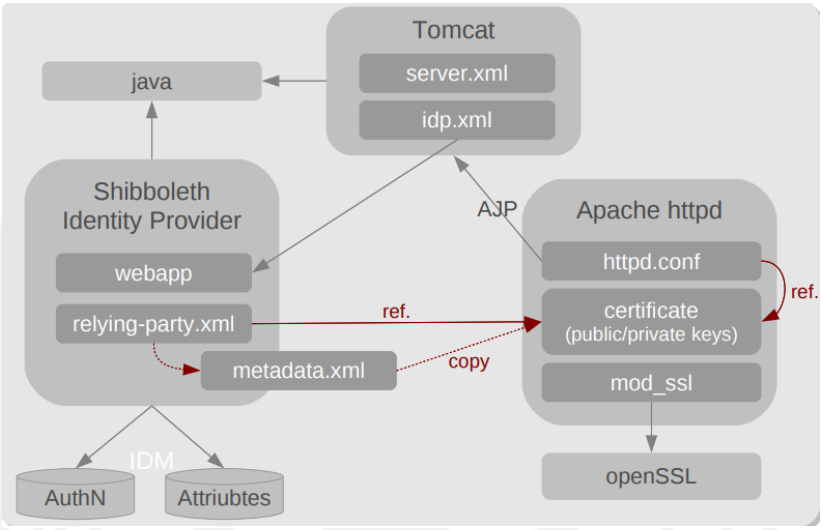
Untuk lebih jelas terkait implementasi zero *downtime* dapat di lihat pada Gambar 4.4 dibawah ini



Gambar 4.4 Replikasi *Pods IdP Shibboleth*.

Setelah memahami konsep tersebut, perlu beberapa tahapan dalam membuat kontainerisasi *Shibboleth IdP*. Dalam penelitian ini, Image Docker dikonfigurasi menggunakan *Dockerfile*, kemudian terdapat beberapa perintah untuk melakukan instalasi *Supervisor* guna menjalankan *Tomcat 9*. Untuk membuat *Shibboleth IdP* dapat diakses dari internal *container* diperlukan *proxy* di dalam *image container* dengan layanan *web Apache2* seperti pada gambar 4.5. Setelah *internal pods* dapat berkomunikasi antara *tomcat 9* dengan *apache2*, dan kemudian *Shibboleth IdP* dapat di akses dari luar klaster *Kubernetes* dengan *nodePort yang mana lalu lintas Shibboleth IdP* akan di teruskan menggunakan *proxy load balancing Nginx*. Sehingga

lalu lintas layanan *Shibboleth IdP* dapat di translasikan dari domain https://idp.uii.ac.id menuju ke *ip worker* disertai *nodePort* di *Kubernetes*. Proses ini akan menambah keamanan pada layanan IdP, karena Shibboleth IdP tersebut terisolasi dan berada di dalam jaringan internal klaster *Kubernetes*.



Gambar 4.5 Alur konfigurasi web service di dalam *container IdP*.

- File httpd.conf untuk melakukan konfigurasi dengan *apache* port 443

```
<IfModule mod_ssl.c>
    <VirtualHost _default_:443>
        ServerName idp.uii.ac.id:443
            RedirectMatch ^/$ https://idp.uii.ac.id/idp/shibboleth
            DocumentRoot /var/www/html
            SSLCertificateFile    /root/cert/ssl.crt
            SSLCertificateKeyFile /root/cert/ssl.key
         </VirtualHost>
</IfModule>
```

Dengan menggunakan layanan *web Apache2* perlu melakukan konfigurasi pada file *apache.conf* untuk menjalankan *Shibboleth IdP* dalam kontainer dengan memodifikasi *HTTP Apache2* yang mengarah ke *Shibboleth IdP* yaitu dengan menentukan lokasi *root* dokumen dan sertifikat *SSL*. *Request* pada *Shibboleth IdP* melalui *protocol https* dapat diakses dengan modul *AJP* yaitu dengan meneruskan lalu lintas *IdP* ke *Tomcat*. Tugas dapat dilakukan dengan menyesuaikan file *proxy_ajp.conf*, yang terletak pada file konfigurasi */etc/apache2/sites-available/idp.conf* pada *image Ubuntu* server versi 18.04.

Penulis menambahkan file konfigurasi *ProxyPass* pada file konfigurasi *idp.conf* seperti di bawah ini.

- File *idp.conf* untuk melakukan konfigurasi *proxy IdP* dengan *apache2*

```
<<IfModule mod_proxy.c>
    ProxyPreserveHost On
    RequestHeader set X-Forwarded-Proto "https"
      <Proxy ajp://localhost:8009>
          Require all granted
      </Proxy>
     ProxyPass /idp ajp://localhost:8009/idp retry=5
     ProxyPassReverse /idp ajp://localhost:8009/idp retry=5
</IfModule>
```

- *Dockerfile*

```
# Initialize
FROM ubuntu:18.04

# Basic Packages Apache Tomcat - Servlet and JSP Engine
ENV TZ=Asia/Jakarta
ENV JAVA_HOME ${openjdk_home}
ENV CATALINA_HOME /usr/share/${tomcat}
ENV CATALINA_BASE /var/lib/${tomcat}
ENV PATH $CATALINA_HOME/bin:$PATH
COPY default-ssl.conf /etc/apache2/sites-available/
COPY start.sh /root/
COPY shib /tmp/
COPY shibboleth-identity-provider-4.0.1.tar.gz /tmp/

RUN apt-get update \
 && apt-get -y install ntp curl wget sudo openssl ${openjdk} \
 && supervisor expect apache2 install ${tomcat} \
 && chown -R tomcat:tomcat /var/log/${tomcat}/ \
 && chmod -R 644 /var/log/${tomcat}/
 && chmod +x /tmp/shib \
  && chmod +x /root/start.sh \
 && apt install libmysql-java libcommons-dbcp-java \
 && libcommons-pool-java --no-install-recommends ca-certificates-java -
y \
 && tar -xzf /tmp/shibboleth-identity-provider-4.0.1.tar.gz \
 && /tmp/shib \
 && a2enmod proxy_http ssl headers alias include negotiation \
 && proxy_ajp rewrite remoteip a2ensite default-ssl.conf idp.conf \
EXPOSE 443
ENTRYPOINT ["sh", "/root/start.sh"]
```

- *Pseudocode* dari konfigurasi *file.sh*

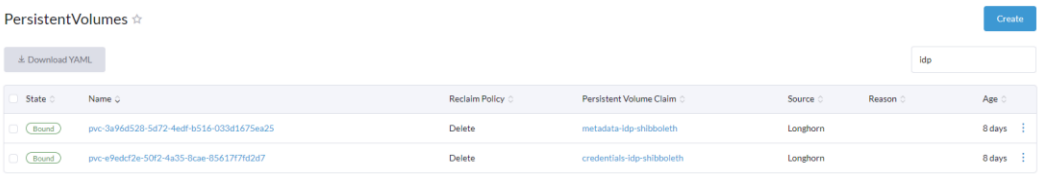
```
Initialize
   #!/bin/bash
   DOMAIN_IDP = idp.uii.ac.id

begin
        sed -i "s/IDP_ENV/$DOMAIN_IDP/g" /etc/apache2/sites-available/default-ssl.conf
    chown tomcat:tomcat /opt/shibboleth-idp/credentials/ -R
    chown tomcat:root /opt/shibboleth-idp/metadata/ -R
    /etc/init.d/apache2 start
    /usr/bin/supervisord -c /etc/supervisor/supervisord.conf
    tail -F /opt/shibboleth-idp/logs/idp-process.log
End
```

Untuk membuat sebuah kontainer, penulis menyiapkan sebuah *Dockerfile* yang berisi instruksi dari kebutuhan aplikasi *Shibboleth IdP*, serta konfigurasi file *Shibboleth IdP* berupa *peudocode* dari *file.sh*. Setelah file kustomisasi dan konfigutasi file selesai dibuat, maka untuk membangun sebuah *image* dengan *CI/CD*, yaitu dengan melakukan proses automasi *build* kemudian hasil *build* tersebut secara otomatis terunggah ke dalam sebuah penyimpanan pada layanan *google container registry*. Sehingga saat proses *deployment* aplikasi *Shibboleth IdP* maka akan mengambil *file image* dari *repository google container registry* yang beralamat di https://asia.gcr.io/Project/Shibboleth-IdP:tag.

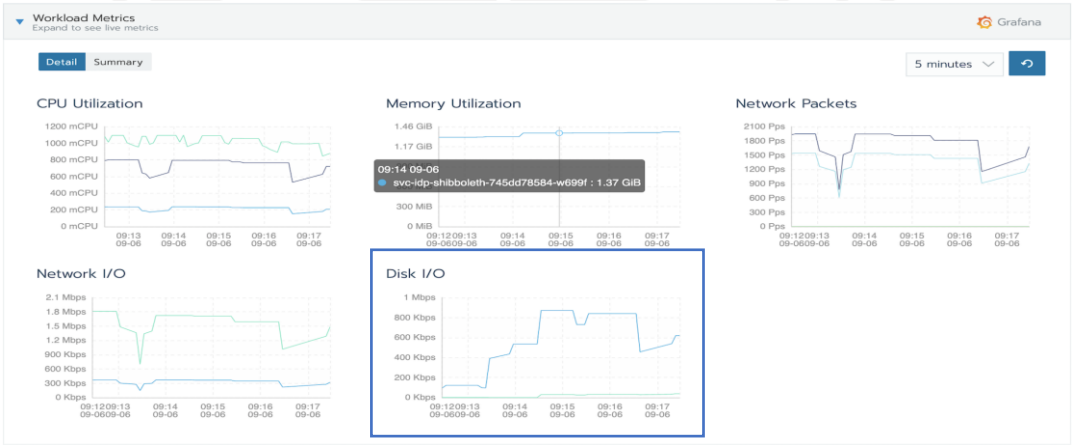
4.2.2 *Persistent storage* pada server klaster *Kubernetes*

Pada layanan *persistent storage* untuk mendukung infrastruktur layanan klaster *Kubernetes* pada penelitian ini melakukan uji coba performa dengan menggunakan alat bernama *fio*. Dalam hal ini, perangkat keras server yang digunakan dapat mempengaruhi performa dan nilai yang dihasilkan pada layanan *Shibboleth IdP,* sehingga hasil pengujian ini mungkin dapat berbeda dengan studi kasus lainnya. Menurut metrik pada gambar 4.7, penggunaan *bandwidth disk I/O* layanan *Shibboleth IdP* kurang dari *1Mbps*. Sebelumnya infrastruktur *storage* di BSI dengan menggunakan layanan *persistent storage Democratic CSI TrueNAS*, namun terdapat kendala *down* pada layanan *storage* karena arsitekturnya yang tidak dapat di replikasi berdasarkan hardware server yang berbeda seperti yang sudah di jelaskan pada sub bab 2.7. Penggunaan *persistent storage* yang tidak direplikasi seperti

Democratic CSI TrueNAS dapat memicu timbulnya resiko *single point of failure* seperti yang pernah terjadi di Badan Sistem Informasi UII. Akibatnya, diperlukan solusi lain untuk mengimplementasikan *persistent storage* di dalam infrastruktur *Kubernetes* yang dapat melakukan proses replikasi atau redundansi yaitu dengan *persistent storage Longhorn* berbasis penyimpanan blok yang terdistribusi seperti yang telah di jelaskan pada sub bab 2.6. Penggunaan *persistent storage* pada *Shibboleth IdP* tergambar seperti Gambar 4.6 dibawah ini.



Gambar 4.6 *Persistent Volume Claims Shibboleth IdP.*



Gambar 4.7 Penggunaan *resource* layanan *Shibboleth IdP* pada klaster *Kubernetes*

Tabel 4.1, 4.2, dan 4.3 menunjukkan hasil uji coba yang dilakukan menggunakan alat bernama *fio*, bahwa performa dari penggunaan *persistent storage* antara *Democratic CSI TrueNAS* dan *Longhorn* dengan perangkat keras yang sama; rata-rata *random read* dan *random write* dari *Democratic CSI TrueNAS* lebih cepat dibandingkan *Longhorn*, berdasarkan implementasinya pada Gambar 4.7, *Shibboleth IdP* hanya membutuhkan *bandwidth* kurang dari 1Mbps. Terdapat variabel *iodepth* pada pengujian, yaitu untuk mengetahui seberapa besar nilai yang dapat dijalankan pada unit I/O *hardware*. Kemudian juga terdapat variabel *bs* yang berfungsi untuk menentukan ukuran blok yang akan dihasilkan oleh pengujian *I/O*. Nilai defaultnya adalah 4k; jika tidak ditentukan, pengujian akan menggunakan nilai default. variabel *size* dapat didefinisikan sebagai ukuran file pada *fio*, yang digunakan sebagai variable

pembandingan pada skema testing performa dari penggunaan *persistent storage* di dalam infrastruktur server klaster *Kubernetes*.

Tabel 4.1 *Storage Performance Evaluation random read and write*

Environment	**-iodepth=128 --bs=4k -size=1G -runtime=60**	
	randread	**Randwrite**
TrueNAS Core Storage with CSI Democratic	IOPS = 30.1k BW = 118MiB/s	IOPS=881 BW=3525KiB/s
Longhorn Storage	IOPS = 9145 BW = 35.7MiB/s	IOPS = 731 BW = 2927KiB/s

Tabel 4.2 *Storage Performance Evaluation random read and write*

Environment	**-iodepth=128 --bs=1M -size=1G -runtime=60**	
	randread	**randwrite**
TrueNAS Core Storage with CSI Democratic	IOPS = 428 BW = 429MiB/s	IOPS = 103 BW = 140MiB/s
Longhorn Storage	IOPS = 259 BW = 260MiB/s	IOPS = 76 BW = 76.0MiB/s

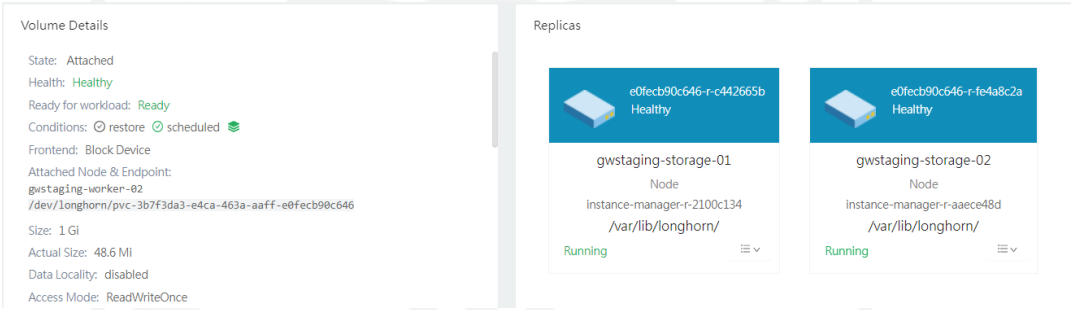
Tabel 4.3 *Storage Performance Evaluation random read and write*

Environment	**-iodepth=1 -bs=4K -size=1G -runtime=60**	
	Randread	**randwrite**
TrueNAS Core Storage with CSI Democratic	IOPS=3340 BW=13.0MiB/s	IOPS=416 BW=1666KiB/s
Longhorn Storage	IOPS=545 BW=2182KiB/s	IOPS=84 BW=336KiB/s

Berdasarkan Tabel 4.1, hasil pengujian menunjukkan bahwa skor IOPS pada *random read* dengan menggunakan *Democratic CSI TrueNAS* dapat mencapai 30.1k, dan skor *random write* hingga 881, sedangkan pada penggunaan Longhorn, skor IOPS pada *random read* dapat mencapai 9145 dan skor *random write* 731. Pada Tabel 4.2 skor *random read* pada bandwidth *Democratic CSI TrueNAS* dapat mencapai 429MiB/s, dan *random write* mencapai 140MiB/s, sedangkan *random read* pada *Longhorn* dapat mencapai bandwidth 260MiB/s, dan bandwidth *random write* sebesar 76,0MiB/s. Pada ke 3 tabel hasil pengujian *persistent storage* di atas,

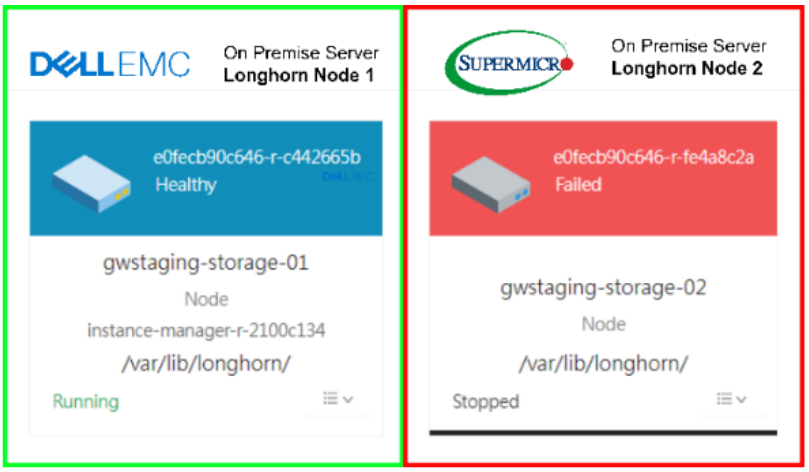
setiap skor IOPS dan bandwidth dapat dilihat bahwa skor yang diperoleh *Democratic CSI TrueNAS* lebih besar dari pada *Longhorn*.

Kedua teknologi penyimpanan tersebut memiliki kelebihan dan kekurangannya masing-masing, seperti *Democratic CSI TrueNAS* dapat menangani lebih banyak *IOPS* daripada *Longhorn* dan lebih cepat daripada *Longhorn*. Pada saat yang sama, *Longhorn* berfokus pada tingkat replikasi dan redundansi datanya untuk menjamin kehandalan layanannya terjaga ketika terjadi kegagalan pada salah satu server *storage*. Dalam penelitian ini, *Longhorn* dapat menjawab tantangan akan adanya kegagalan perangkat keras yang pernah terjadi di BSI UII. Setelah beberapa percobaan dengan *Persistent Storage Longhorn* menunjukkan hasil yang lebih *reliable* seperti pada Gambar 4.8 dan 4.9. Hasil *performance* testing ini bisa saja berbeda jika dibandingkan dengan penelitian lain. Perbedaan tersebut dapat disebabkan oleh penggunaan perangkat keras yang berbeda atau jaringan yang telah diimplementasikan di *data center*.

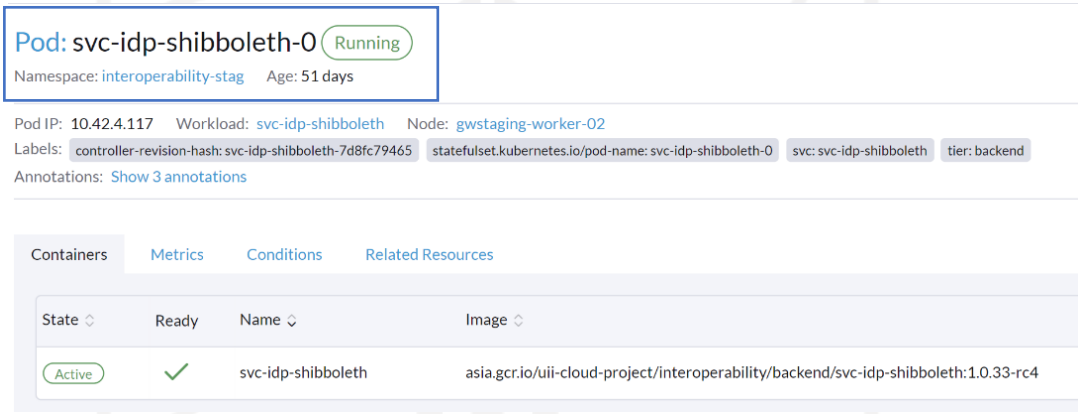


Gambar 4.8 Penggunaan replikasi pada *persistent storage Longhorn.*

Gambar 4.9 menunjukkan hasil uji kegagalan pada salah satu mesin server *storage* dengan status data tereplikasi pada 2 mesin hardware server secara terpisah seperti pada gambar 4.8. Dengan menggunakan *persistent storage Longhorn* yang memanfaatkan fungsi replikasi penyimpanan, mendapatkan hasil layanan *Shibboleth IdP* tetap berfungsi normal, seperti yang ditunjukkan pada Gambar 4.10.



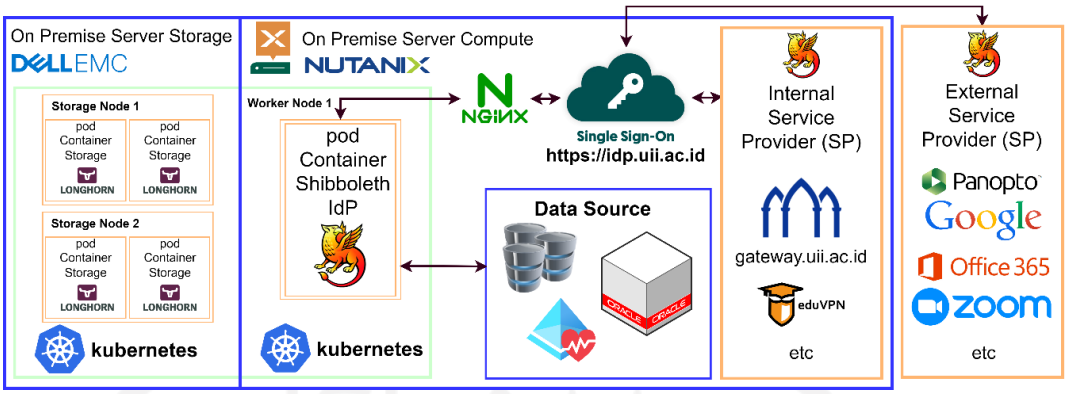
Gambar 4.9 Pengujian *hardware failure* pada *persistent storage Longhorn*.



Gambar 4.10 *Shibboleth IdP* dengan *state running*.

4.2.3 *Shibboleth IdP* pada layanan AAA sistem enterprise dengan metode *load balancer*

Setelah infrastruktur *cluster Kubernetes* dan *persistent volume Longhorn* yang menjadi wadah dari aplikasi *Shibboleth IdP* dipastikan *reliable* dengan memanfaatkan fungsi redundansinya, maka dapat dipastikan resiko adanya *single point of failure* dapat dihindari. Sehingga dapat dilakukan uji beban pada aplikasi yang ada di dalam *Kubernetes*, dalam hal ini *Shibboleth IdP* sebagai objek penelitian, karena aplikasi tersebut menjadi pusat manajemen *single sign on* yaitu menjadi pusat autentikasi serta otorisasi bagi seluruh aplikasi yang terhubung ke dalam *Single Sign On* pada beberapa layanan seperti pada Gambar 4.11.



Gambar 4.11 Alur komunikasi layanan *IdP Shibboleth*.

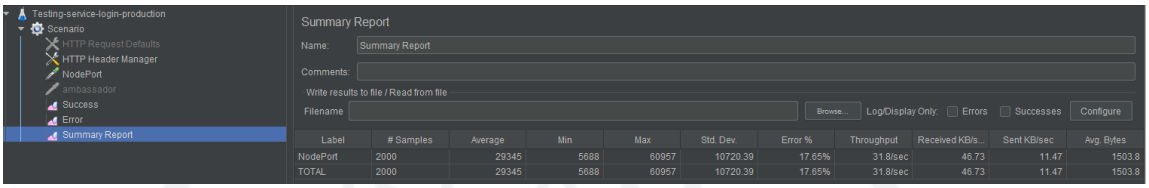
Untuk memastikan layanan login dapat digunakan pada UII Gateway yaitu dengan *Shibboleth SP* yang berada di dalam kontainer svc-login-lumen terdaftar pada layanan *SSO Shibboleth IdP* seperti pada Gambar 1.4. Dengan layanan login UII Gateway terintegrasi dengan *Shibboleth IdP*, maka beberapa layanan autentikasi dan otorisasi pada UII Gateway dapat menjadi bahan uji coba *load testing*, untuk memastikan bahwa kontainerisasi *Shibboleth IdP* sebagai pusat manajemen *single sign on* dengan metode *load balancing*, mampu menangani permintaan pengguna dengan optimal. Sehingga dapat diketahui seberapa besar nilai beban resource yang dibutuhkan dan dapat terbagi secara merata sesuai dengan kebutuhan dari layanan agar tidak terjadi *downtime* pada layanan.

4.3 Load testing pada layanan login

Pertama kali aplikasi UII Gateway diluncurkan pada tahun 2019 lalu, telah mengalami kendala pada layanan login. Hal tersebut tentu mengakibatkan gagalnya proses bisnis yang ada pada layanan sistem enterprise ini. Oleh karena itu, perlu adanya uji coba untuk mendapatkan solusi dari masalah tersebut.

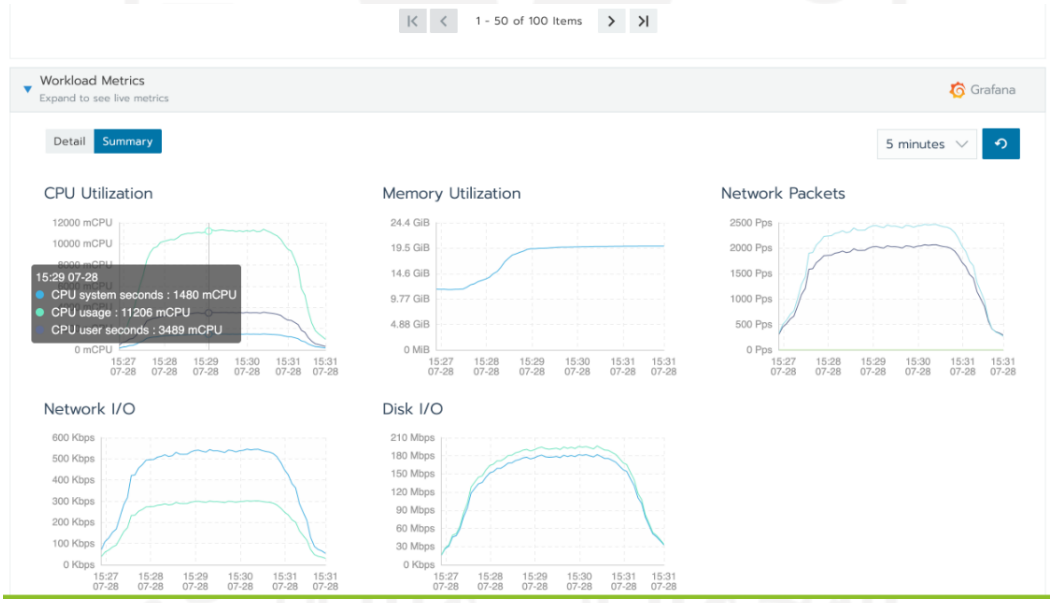
Dalam penelitian ini yaitu dengan membandingkan penggunaan *resource* dari sebelum menggunakan *Shibboleth SP* yang terintegrasi dengan *Shibboleth IdP* dan setelah mengintegrasikan layanan login pada *Shibboleth IdP*. Dalam penelitian ini terbagi menjadi 2 bagian, yang pertama yaitu melakukan melakukan *load testing* pada svc-login-lumen yang belum terintegrasi dengan layanan *Shibboleth IdP* dan kemudian yang ke 2 yaitu melakukan *load testing* pada svc-login-lumen yang di dalamnya terdapat *Shibboleth SP* yang terintegrasi dengan *Shibboleth IdP*. Kemudian dari *load testing*

tersebut dapat diketahui tingkat *error* svc-login-lumen yang terhubung pada *Shibboleth IdP* dan juga kebutuhan resource yang diperlukan pada *Shibboleth IdP* setelah di integrasikan ke *environment production*. Gambar 1.12 merupakan hasil *load testing* pada svc-login-lumen yang belum terintegrasi dengan SSO dengan replikasi pod container *maximum* 100.



Gambar 4.12 *Load testing* menggunakan *JMeter* pada *svc-login-lumen* max 100.

Gambar 4.13 merupakan penggunaan resource pada layanan *svc-login-lumen* dengan replikasi pod sebanyak 100 kali.

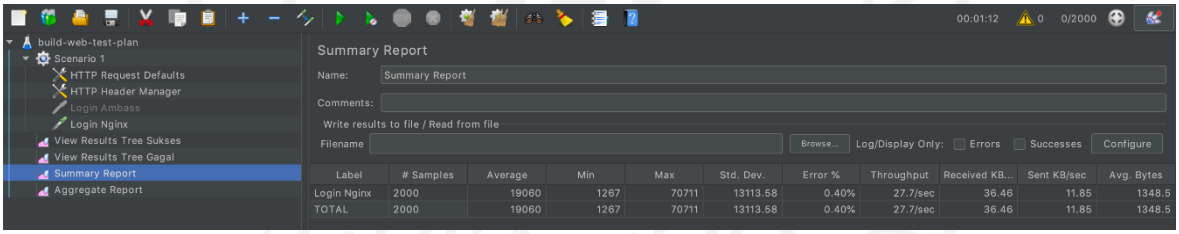


Gambar 4.13 Penggunaan *resource svc-login-lumen*.

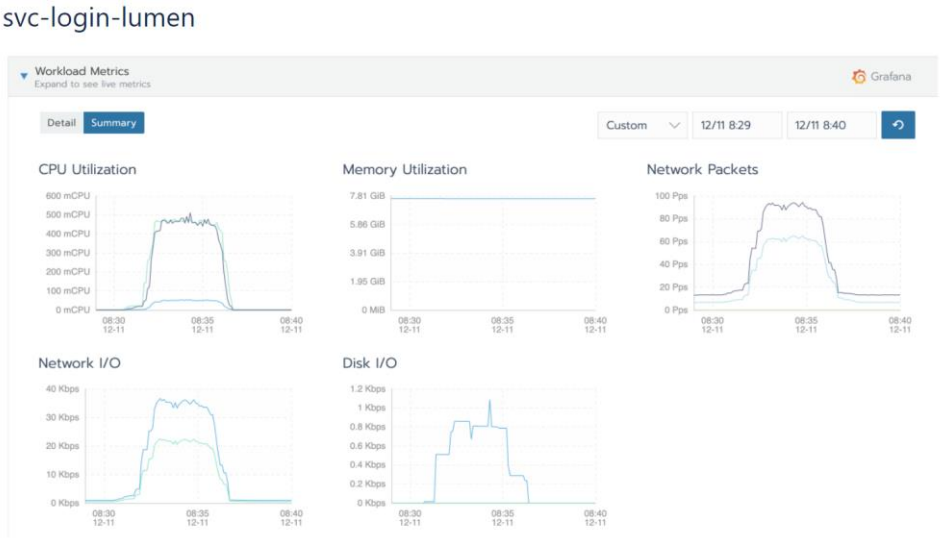
```
resources:
  requests:
    cpu: 100m
    memory: 128Mi
  limits:
    memory: 128Mi
---
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
```

```
  name: svc-login-lumen
  labels:
    svc: svc-login-lumen
    tier: backend
  namespace: interoperability-stag
spec:
  scaleTargetRef:
    apiVersion: apps/v1beta2
    kind: Deployment
    name: svc-token-lumen
  minReplicas: 50
  maxReplicas: 100
  targetCPUUtilizationPercentage: 50
```

Berdasarkan *resource* yang dimiliki pada klaster *Kubernetes* untuk mengetahui kebutuhan *resource* layanan baik menggunakan *trigger* berdasarkan *CPU* ataupun berdasarkan penggunaan *RAM*, maka dapat dengan rumus ini "*desiredReplicas = ceil[currentReplicas * (currentMetricValue / desiredMetricValue)]",* sehingga penggunaan *resource* dengan skema maksimum replikasi 100 seperti pada gambar 4.13, dan pada gambar 4.12 merupakan uji beban dengan menggunakan *JMeter* dan mendapatkan hasil bahwa *pod* telah replikasi menjadi sejumlah 100 *pod* dengan penggunaan *CPU* total hingga lebih dari 11206mCPU, dan penggunaan memori hingga mencapai 19.5 GiB. Hal ini tentu menjadi sebuah masalah pada layanan, dengan tingkat *error* mencapai 17.65%. Berdasarkan hasil uji tersebut perlu dilakukan perhitungan ulang pada penggunaan resource setiap *pod*, seperti penggunaan limit atau batas pada *mCPU* menyebabkan terjadinya *throttle* pada *CPU*. Sehingga proses yang sedang berlangsung pada *pod* menjadi lambat bahkan terhenti di akibatkan adanya kemacetan dan kepadatan pada *CPU* yang kecil.



Gambar 4.14 *Load testing* menggunakan *JMeter* pada svc-login-lumen dengan replikasi max 10.



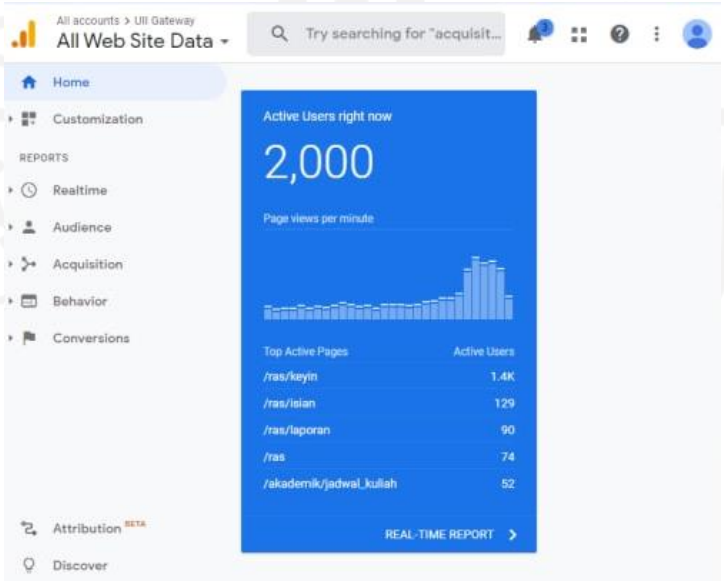
Gambar 4.15 Penggunaan *resource svc-login-lumen*.

Dengan skema konfigurasi *resource* dan kustomisasi variabel pada *PHP worker_connections* 20480, *pm_max_children* 200, dan *max_execution_time* 150 dan memperbesar *resource* pada masing-masing *pod* sebesar 10 kali lipat dari uji coba sebelumnya, sehingga masing masing menjadi *cpu* sebesar 1 *core* dan *ram* 1 Gb, maka dapat menekan tingkat *error* menjadi 0.40% dengan penggunaan memori dari svc-login-lumen lebih kecil yaitu 7.81GiB dan membutuhkan *CPU* rata-rata sebesar 500mCPU.

```
resources:
  requests:
    cpu: 1
    memory: 1Gi
  limits:
    memory: 1Gi
env:
  - name: WORKER_CONNECTIONS
    value: "20480"
  - name: PM_MAX_CHILDREN
    value: "200"
  - name: MAX_EXECUTION_TIME
    value: "150"
---
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  labels:
```

```
    svc: svc-login-lumen
    tier: backend
  name: svc-login-lumen
  namespace: interoperability-stag
spec:
  minReplicas: 3
  maxReplicas: 10
  metrics:
    - resource:
        name: cpu
        target:
          averageUtilization: 50
          type: Utilization
      type: Resource
  scaleTargetRef:
    apiVersion: apps/v1beta2
    kind: Deployment
    name: svc-login-lumen
```

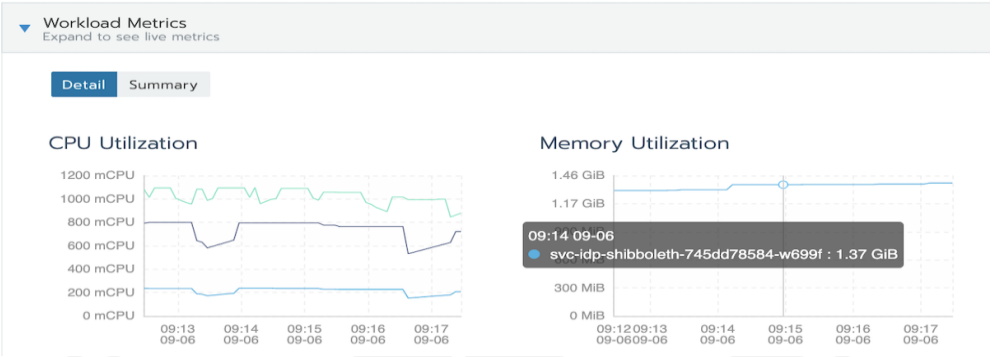
Setelah uji coba menggunakan *JMeter* mendapatkan nilai variable penggunaan *resource* yang sesuai, maka proses integrasi layanan svc-login-lumen dengan layanan *Shibboleth IdP* dapat dilakukan, dan berikut ini merupakan tampilan *monitoring* yang sudah digunakan sebagai *production environment*, terpantau saat pengguna mencapai 2000 seperti pada Gambar 4.16 dan penggunaan resource Shibboleth IdP seperti pada Gambar 4.17.



Gambar 4.16 *Monitoring google analytic* UII Gateway.

Konfigurasi penggunaan resource pada *Shibboleth IdP* seperti di bawah ini.

```
resources:
  requests:
    cpu: 500m
    memory: 2048Mi
  limits:
    memory: 2048Mi
```



Gambar 4.17 Penggunaan *resource Shibboleth IdP*.

4.4 Hasil Uji Coba dan Analisis

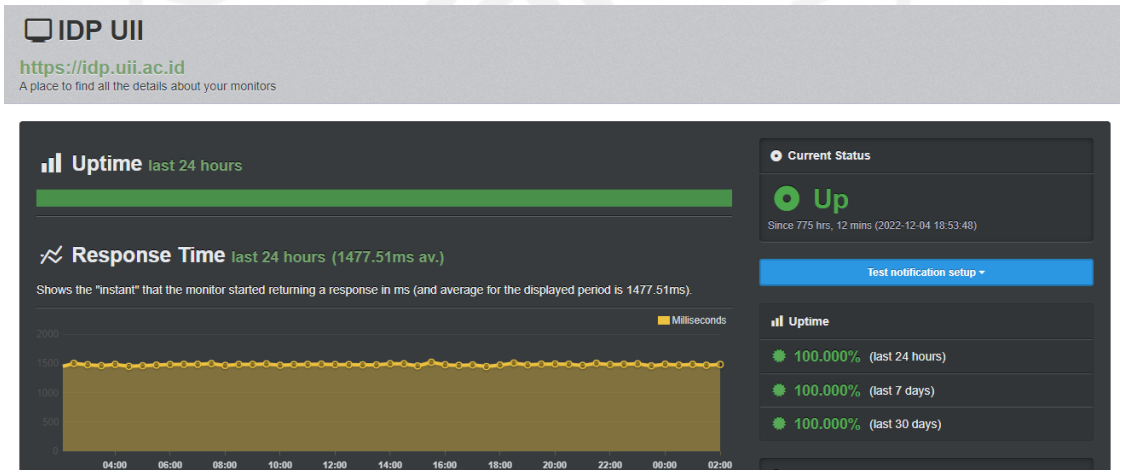
Tabel 4.4 *Load balancing nginx* pada layanan *Shibboleth IdP*

Server		**State**	**Response Time**	**Response**	
				2xx	**5xx**
Worker-01		UP	0ms	282144	0
Worker-02		UP	0ms	278691	0
Worker-03		UP	24ms	330677	1
Worker-04		UP	0ms	258052	0
Worker-05		UP	0ms	290366	0
Worker-06		UP	1ms	213927	1
Worker-07		UP	23ms	360018	0
Worker-08		UP	0ms	365679	0

Berdasarkan *monitoring* dari metode *load blancer nginx* seperti pada Tabel 4.4, menunjukkan bahwa tidak terdapat nilai *error* 5xx yang signifikan, dengan replikasi saat ini hanya berjalan 1 *pod* pada layanan *Shibboleth IdP* dan terpantau *request cpu* 500m dengan *memory limit* 2048. Kemudian untuk layanan *svc-login-lumen* yang terdapat pada tahap pertama dengan replikasi *pod* maksimum 100 dengan tahap uji beban yang ke dua dengan replikasi *pod* maksimum 10, terdapat perbedaan yang signifikan utamanya pada penggunaan *resource server* dan tingkat *error* layanan. Pada tahap uji coba pertama, konfigurasi *php* masih default konfigurasi, namun pada tahap uji coba ke dua telah dilakukan kustomisasi pada php *worker* proses menjadi 200. Hal

tersebut cukup berpengaruh pada proses yang berjalan di setiap *pod svc-login-lumen*, dan juga penggunaan *resource* yang di tambahkan dari konfigurasi pada uji coba pertama dengan request cpu 100m dan memory 128Mi dengan jumlah replikasi *pod* mencapai 100. Sedangkan pada uji coba kedua dengan request *cpu* 1000m dan *memory* 1024Mi setelah dilakukan uji beban menggunakan *JMeter* maka mendapatkan nilai penggunaan *resource* server yang jauh lebih kecil dan tingkat *error* yang lebih kecil pula seperti Gambar pada 4.14 dan 4.15.

Dengan hasil perhitungan pada ujicoba kedua tersebut, *svc-login-lumen* yang diintegrasikan dengan *Shibboleth IdP*, pada layanan login sistem enterprise UII Gateway tidak mengalami kendala yang fatal, stable, dan terhindar dari *single point of failure*. Sehingga proses bisnis yang terdapat pada layanan UII Gateway dapat berjalan normal sesuai dengan fungsinya masing-masing. Oleh karena itu penulis telah melakukan monitoring *uptime* dari layanan *Shibboleth IdP* pada laman web https://idp.uii.ac.id, untuk memastikan tidak terjadi *downtime* pada layanan login *UII Gateway*, dan didapatkan nilai *uptime* 100% pada kurun waktu 30 hari terakhir, dari penggunaan layanan login *Shibboleth IdP*, seperti pada gambar 4.18 di bawah ini. Sehingga nilai *uptime* tersebut dapat menjadi tolak ukur keberhasilan dari kontainerisasi *Shibboleth IdP* dengan menggunakan *cluster Kubernetes* dan *persistent volume Longhorn*, serta dapat menjadi bahan evaluasi untuk terus menjaga layanan agar tetap aktif, sehingga proses bisnis serta fungsi yang ada di dalam UII Gateway dapat digunakan oleh semua kalangan di Universitas Islam Indonesia.



Gambar 4.18 *Monitoring Uptime Shibboleth IdP* pada halaman web https://idp.uii.ac.id

BAB 5
Kesimpulan dan Saran

Bab ini merupakan kesimpulan akhir yang didapatkan dari penelitian yang telah dilakukan dan juga dipaparkan saran-saran yang bersifat membangun untuk penelitian selanjutnya di masa yang akan datang.

4.5 Kesimpulan

Berdasarkan pengujian dan analisis yang telah dilakukan menghasilkan beberapa kesimpulan penelitian sebagai berikut:

1. Implementasi GitLab CI pada pengembangan di lingkungan Badan Sistem Informasi pada layanan UII Gateway telah banyak merubah budaya manual yang dilakukan oleh developer menjadi proses automasi, yang tentunya hal tersebut dapat meminimalisir adanya kesalahan deploy manual karena semua proses *build* hingga proses *deployment* telah dilakukan secara otomatis oleh *runner* yang terintegrasi antara *GitLab* dengan *cluster Kubernetes*.
2. Dengan menggunakan *persistent storage Longhorn*, terbukti dapat meminimalisir terjadinya kegagalan sistem pada *Shibboleth IdP* serta layanan yang ada di dalam *cluster Kubernetes*, karena *persistent storage* yang terdistribusi dengan fungsi replikasi ke mesin server yang berbeda dapat meminimalisir dari adanya risiko *single point of failure (SPOF),* dan penggunaan metode *load balancing* yang berfungsi sebagai pembagi *traffic* pada *cluster Kubernetes*.
3. Setelah dilakukan *load testing* pada layanan *svc-login-lumen* dapat di ambil kesimpulan bahwasannya jumlah request *cpu* dan juga *ram* harus di sesuaikan dengan jumlah replikasi yang akan di tentukan. Apabila *request resource* terlalu kecil seperti pada uji coba pertama, maka justru penggunaan *resource pod* akan melebihi batas hingga puluhan *core*. Oleh karenanya, setelah dilakukan perhitungan ulang, maka di dapatkan nilai *request cpu* dan juga *ram* sesuai dengan kebutuhan yang ada di lingkungan UII seperti pada uji coba ke dua. Saat ini *svc-login-lumen* ini telah diubah arsitekturnya yaitu dengan menggabungkan layanan login dengan *Shibboleth SP* dan terintegrasi pada layanan *Shibboleth IdP*. Sehingga layanan login di sistem enterprise UII Gateway telah menggunakan login *SSO* secara keseluruhan.

4.6 Saran

Setelah berhasil mengimplementasikan kontainerisasi *Shibboleth IdP* pada sistem enterprise menggunakan *Kubernetes* dengan metode *load balancing*, maka perlu ada penelitian lanjutan dengan menyiapkan beberapa server di lokasi yang berbeda dengan *data center* utama, yaitu dengan mengimplementasikan *Data Recovery Center (DRC).* Sehingga dapat membuat *data center redundan* yang terpisah dan berbeda antara *data center primer* dan *data center backup* yang dapat digunakan untuk meminimalisir terjadinya *downtime* jika terjadi bencana di satu lokasi data center dari adanya resiko terputusnya listrik dan akses internet. Kemudian untuk memastikan *uptime* pada layanan dapat terpenuhi, perlu perhitungan *Service Level Agreement (SLA)* yang dapat menjadi acuan pada layanan untuk menentukan batas *downtime* yang diizinkan selama kurun waktu 1 tahun sebagai bentuk pelayanan yang optimal antara penyedia layanan dengan pengguna layanan.

Daftar Pustaka

Arbiansyah, G., Kristianto, D., & Lampung, B. (2010). *Pemetaan Model Tata Kelola Teknologi Informasi Yang*. *2010*(Snati), 133–137.

Blinowski, G., Ojdowska, A., & Przybylek, A. (2022). Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. *IEEE Access*, *10*, 20357–20374. https://doi.org/10.1109/ACCESS.2022.3152803

Chakraborty, T. (2019). Docker and Google Kubernetics. *International Journal of Research Studies in Computer Science and Engineering*, *6*(4), 24–35. https://doi.org/10.20431/2349-4859.0504004

GitLab vs GitHub: Which is right for you - Spectral. (n.d.). Retrieved August 3, 2022, from https://spectralops.io/blog/github-vs-gitlab/

How Does Single Sign-On (SSO) Work? | OneLogin. (n.d.). Retrieved August 3, 2022, from https://www.onelogin.com/learn/how-single-sign-on-works

Kontainerisasi, Salah Satu Tren Cloud Computing 2021. (n.d.). Retrieved August 3, 2022, from http://blog.lintasarta.net/article/cloud-services//kontainerisasi-salah-satu-tren-cloud-computing-2021

KRISTAN W B, I. I. I. (2007). Shibboleth Identity Provider - Login. *The Condor*. http://www.bioone.org/doi/pdf/10.1650/8348.1%5Cnpapers2://publication/uuid/5FED2274-B391-493E-886E-3127F3CDC90C

Microservice architecture style - Azure Architecture Center | Microsoft Docs. (n.d.). Retrieved August 3, 2022, from https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

Poniszewska-Marańda, A., & Czechowska, E. (2021). Kubernetes Cluster for Automating Software Production Environment. *Sensors 2021, Vol. 21, Page 1910*, *21*(5), 1910. https://doi.org/10.3390/S21051910

Prasetyo, S. E., & Salimin, Y. (2021). Analisis Perbandingan Performa Web Server Docker Swarm dengan Kubernetes Cluster. *CoMBInES - Conference on Management, Business, Innovation, Education and Social Sciences*, *1*(1), 825–833. https://journal.uib.ac.id/index.php/combines/article/view/4512

Schairer, U., & Architect, S. A. P. S. (n.d.). *SAP Data Intelligence 3 . 1 on Rancher Kubernetes Engine 2*. 1–56.

Sinnott, R. O., Jiang, J., Watt, J., & Ajayi, O. (2006). Shibboleth-based access to and usage

of Grid resources. *Proceedings - IEEE/ACM International Workshop on Grid Computing*, 136–143. https://doi.org/10.1109/ICGRID.2006.311008

What Is Microservices Architecture? | Google Cloud. (n.d.). Retrieved August 3, 2022, from https://cloud.google.com/learn/what-is-microservices-architecture