KERJA PRAKTIK

Implementasi CI/CD Pipeline pada Jenkins untuk Deployment Microservice Asset-Management hingga Environment SIT

PT Adira Dinamika Multi Finance Tbk.
Millenium Centennial Center Lantai 53, 56-61
Jl Jenderal Sudirman Kav. 25 Jakarta 12920
Periode: 6 Januari 2025 – 6 April 2025

Oleh:

Delai Resgista Setyawan 5025221221

Pembimbing Departemen
Shintami Chusnul Hidayati, S.Kom., M.Sc., Ph.D

Pembimbing Lapangan
Andika Anggakusuma

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember Surabaya 2025

[Halaman ini sengaja dikosongkan]

KERJA PRAKTIK

Implementasi CI/CD Pipeline pada Jenkins untuk Deployment Microservice Asset-Management hingga Environment SIT

PT Adira Dinamika Multi Finance Tbk.
Millenium Centennial Center Lantai 53, 56-61
Jl Jenderal Sudirman Kav. 25 Jakarta 12920
Periode: 6 Januari 2025 – 6 April 2025

Oleh:

Delai Resgista Setyawan 5025221221

Pembimbing Departemen
Shintami Chusnul Hidayati, S.Kom., M.Sc., Ph.D

Pembimbing Lapangan
Andika Anggakusuma

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2025

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN
KERJA PRAKTIK

Implementasi CI/CD Pipeline pada Jenkins untuk Deployment Microservice Asset-Management hingga Environment SIT

Oleh:

Delai Resgista Setyawan 5025221221

Disetujui oleh Pembimbing Kerja Praktik:

1. Shintami Chusnul Hidayati, S.Kom., M.Sc., Ph.D
NIP. 1987202012004

(Pembimbing Departemen)

2. Andika Anggakusuma

(Pembimbing Lapangan)

[Halaman ini sengaja dikosongkan]

Implementasi CI/CD Pipeline pada Jenkins untuk Deployment Microservice Asset-Management hingga Environment SIT

Nama Mahasiswa	: Delai Resgista Setyawan
NRP	: 5025221221
Departemen	: Teknik Informatika FTEIC-ITS
Pembimbing Departemen	: Shintami Chusnul Hidayati, S.Kom., M.Sc., Ph.D
Pembimbing Lapangan	: Andika Anggakusuma

ABSTRAK

PT Adira Dinamika Multi Finance Tbk. merupakan perusahaan pembiayaan terkemuka di Indonesia yang menyediakan berbagai solusi keuangan bagi masyarakat. Selama pelaksanaan kerja praktik, fokus utama adalah pengembangan microservice bernama Asset-Management, yang digunakan sebagai solusi penyimpanan dan pengelolaan aset digital perusahaan seperti gambar, ikon, dan maskot. Microservice ini dirancang untuk mendukung kebutuhan tim internal, seperti programmer dan desainer UI/UX, dalam mengakses aset dengan lebih mudah dan terpusat.

Proses pengembangan mencakup perancangan pipeline CI/CD dengan Jenkins, integrasi Bitbucket webhook, pengaturan credentials, serta deployment otomatis hingga environment SIT. Pipeline CI/CD ini memudahkan proses build, test, dan deploy microservice, sehingga mendukung praktik pengembangan perangkat lunak yang lebih terstruktur dan efisien di lingkungan perusahaan.

Dengan adanya pipeline ini, proses deployment menjadi lebih cepat, aman, dan dapat dipantau dengan lebih baik. Keberadaan microservice Asset-Management juga membantu meningkatkan kolaborasi antar tim dalam mengakses dan menggunakan aset digital perusahaan.

Kata Kunci: CI/CD, Jenkins, Microservice, SIT, Bitbucket

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

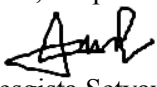
Puji syukur saya panjatkan kepada Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya, sehingga laporan akhir kerja praktik di PT Adira Dinamika Multi Finance Tbk. ini dapat diselesaikan dengan baik dan lancar.

Laporan ini disusun sebagai bentuk pertanggungjawaban dan dokumentasi atas kegiatan kerja praktik yang telah dilaksanakan selama satu semester di PT Adira Dinamika Multi Finance Tbk. Program kerja praktik ini bertujuan untuk memberikan pengalaman praktis dan memperluas pemahaman saya di bidang pengembangan perangkat lunak, khususnya dalam membangun *pipeline* CI/CD untuk mendukung proses *deployment microservice*.

Melalui laporan ini, penulis juga ingin menyampaikan rasa terima kasih kepada pihak-pihak yang telah membantu, mendukung, dan membimbing selama proses kerja praktik dan penyusunan laporan ini, baik secara langsung maupun tidak langsung. Ucapan terima kasih saya sampaikan kepada:

1. Kedua orang tua saya yang selalu memberikan dukungan dan doa yang tiada henti.
2. Ibu Shintami Chusnul Hidayati, S.Kom., M.Sc., Ph.D selaku dosen pembimbing kerja praktik.
3. Bapak Andika Anggakusuma selaku pembimbing lapangan di PT Adira Dinamika Multi Finance Tbk. yang telah memberikan arahan, bimbingan, dan wawasan yang sangat berharga selama kerja praktik berlangsung.
4. Teman-teman yang senantiasa memberikan dukungan, semangat, dan bantuan selama penulis menjalankan kerja praktik dan menyusun laporan ini.

Jakarta, 6 April 2025



Delai Resgista Setyawan

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

1.1. Latar Belakang

Perkembangan teknologi informasi yang pesat mendorong perusahaan untuk terus berinovasi dalam mengoptimalkan proses bisnis dan pengelolaan aset digital. Salah satu teknologi yang banyak diadopsi dalam mendukung proses pengembangan perangkat lunak adalah metode *Continuous Integration* dan *Continuous Deployment* (CI/CD). CI/CD memungkinkan tim pengembang untuk mengotomatisasi proses *build*, *test*, dan *deployment* secara berkesinambungan sehingga meminimalisir kesalahan manual, mempercepat waktu rilis, dan meningkatkan kualitas perangkat lunak.

PT Adira Dinamika Multi Finance Tbk., sebagai perusahaan pembiayaan terkemuka di Indonesia, juga memanfaatkan teknologi untuk mendukung efisiensi dan efektivitas dalam pengelolaan proses bisnis. Salah satu kebutuhan yang muncul adalah penyediaan Asset-Management, yaitu *microservice* yang berfungsi untuk menyimpan dan mengelola aset digital seperti gambar, ikon, dan maskot perusahaan. *Microservice* ini digunakan oleh berbagai pihak internal, seperti *programmer*, desainer UI/UX, dan tim kreatif, yang membutuhkan akses cepat dan terpusat terhadap aset digital perusahaan.

Dalam pengembangan Asset-Management, dibutuhkan *pipeline* CI/CD yang terintegrasi dengan sistem kontrol versi seperti Bitbucket dan *tools* otomasi seperti Jenkins. *Pipeline* ini memungkinkan

proses *build*, *test*, dan *deployment* dilakukan secara otomatis hingga ke *environment* SIT (*System Integration Testing*). Penggunaan *pipeline* CI/CD ini juga dilengkapi dengan pengaturan Bitbucket Webhook dan manajemen *credentials* untuk mendukung keamanan dan fleksibilitas integrasi antar sistem.

Melalui kerja praktik ini, diharapkan dapat memberikan kontribusi nyata dalam meningkatkan proses pengembangan perangkat lunak di lingkungan perusahaan, sekaligus memperluas pemahaman mengenai penerapan *pipeline* CI/CD dalam industri. Selain itu, pengalaman ini menjadi bekal penting bagi pengembangan kompetensi profesional dalam bidang teknologi informasi, khususnya pengembangan dan otomasi perangkat lunak.

1.2. Tujuan

Tujuan kerja praktik ini adalah:

1. Menyelesaikan kewajiban nilai kerja praktik sebesar 4 SKS.
2. Menyampaikan kontribusi yang telah diberikan kepada organisasi melalui proyek-proyek yang dilakukan.
3. Mengevaluasi keterampilan teknis dan non-teknis yang dikembangkan selama program magang.

1.3. Manfaat

Manfaat yang diperoleh dari kerja praktik ini adalah:

1. Memberikan kontribusi langsung dalam pengembangan *microservice* Asset-Management yang digunakan sebagai pusat penyimpanan dan pengelolaan aset digital perusahaan, seperti ikon, gambar, dan maskot, untuk mendukung kebutuhan tim internal seperti *programmer* dan

desainer UI/UX.

2. Membantu perusahaan dalam mengotomasi proses *build*, *test*, dan *deployment microservice* melalui penerapan *pipeline* CI/CD menggunakan Jenkins, sehingga mendukung praktik pengembangan perangkat lunak yang lebih efisien, terukur, dan terstruktur.
3. Memberikan perspektif baru dalam penerapan *tools* dan teknologi seperti Jenkins, Bitbucket Webhook, dan pengelolaan *credentials* untuk meningkatkan keamanan dan efisiensi integrasi antar sistem.
4. Memberikan pengalaman praktis bagi peserta kerja praktik dalam memahami proses pengembangan perangkat lunak di lingkungan industri, khususnya terkait pengembangan *pipeline* CI/CD, *deployment microservice*, dan kolaborasi lintas tim.

1.4.Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, rumusan masalah yang diangkat dalam kerja praktik ini adalah sebagai berikut:

1. Bagaimana merancang dan membangun *pipeline* CI/CD yang mendukung proses *build*, *test*, dan *deployment microservice* Asset-Management hingga ke *environment* SIT dengan menggunakan Jenkins?
2. Bagaimana mengintegrasikan Bitbucket webhook dan pengelolaan *credentials* untuk mendukung keamanan serta otomatisasi *pipeline* CI/CD yang dikembangkan?
3. Bagaimana memastikan *microservice* Asset-Management dapat diakses dan digunakan oleh berbagai tim internal perusahaan, seperti *programmer*, desainer UI/UX, dan tim terkait

lainnya, secara efektif dan efisien?

1.5. Lokasi dan Waktu Kerja Praktik

Kerja praktik ini dilaksanakan pada waktu dan tempat sebagai berikut:

Lokasi	: *Hybrid*
Waktu	: 6 Januari 2025 – 6 April 2025
Hari Kerja	: Senin – Jumat
Jam Kerja	: 08.30 – 17.00

1.6. Metodologi Kerja Praktik

1.6.1. Perumusan Masalah

Tahap ini diawali dengan diskusi bersama mentor dan tim terkait permasalahan pengelolaan proses CI/CD aplikasi *microservice* di lingkungan Kubernetes. Permasalahan yang diangkat meliputi kebutuhan automasi *build*, *deploy*, serta integrasi sistem keamanan untuk manajemen *secret* menggunakan HashiCorp Vault. Diskusi ini bertujuan untuk menentukan fokus implementasi, seperti konfigurasi Jenkins *pipeline*, *setup* Vault Injector, serta pengaturan otentikasi Kubernetes di Vault.

1.6.2. Studi Literatur

Melakukan studi literatur terhadap berbagai dokumentasi resmi dan referensi relevan yang berkaitan dengan CI/CD, Kubernetes, Jenkins, dan Vault. Literatur yang dipelajari meliputi *best practice pipeline* Jenkins, penggunaan Vault Injector untuk pengelolaan *secrets*, metode otentikasi Kubernetes di Vault, serta keamanan dalam pengelolaan konfigurasi *environment application deployment*.

1.6.3. Analisis dan Perancangan Sistem

Setelah memahami kebutuhan dan literatur yang ada, dilakukan analisis alur *pipeline* CI/CD untuk *microservice* aplikasi yang akan di-*deploy* ke dua

klaster Kubernetes (AliCloud dan GCP). Pada tahap ini juga dirancang struktur *role* dan *policy* di Vault untuk membatasi akses *secrets* berdasarkan kebutuhan aplikasi, serta menentukan metode integrasi antara Kubernetes dan Vault (melalui *service account*, *role*, dan JWT token).

1.6.4.Implementasi Sistem

Tahap ini mencakup pembuatan *pipeline* di Jenkins untuk proses *build image* Docker, push *image* ke *registry*, hingga *deployment* ke Kubernetes klaster. Selain itu dilakukan *setup* Vault Injector di klaster, konfigurasi *authentication method* Kubernetes di Vault, serta pembuatan *role* dan *policy* untuk masing-masing aplikasi. *Secrets* dari Consul juga diimpor ke Vault menggunakan *script* otomatis untuk memastikan pengelolaan *secrets* lebih terpusat.

1.6.5.Pengujian dan Evaluasi

Setelah sistem diimplementasikan, dilakukan pengujian dengan cara menjalankan *pipeline* CI/CD, melakukan *port-forwarding service* aplikasi di klaster, serta memastikan aplikasi berjalan menggunakan Vault Injector dengan memeriksa proses *service* (ps -ef). Evaluasi dilakukan bersama tim untuk memastikan integrasi Vault berfungsi dengan baik di lingkungan *non-production* sebelum digunakan lebih luas.

1.6.6.Kesimpulan dan Saran

Pada tahap ini disusun ringkasan hasil implementasi beserta rekomendasi pengembangan lebih lanjut, seperti penambahan pengujian performa menggunakan JMeter untuk aplikasi internal, serta perluasan integrasi Vault di lingkungan *Production*.

1.7.Sistematika Laporan

1.7.1.Bab I Pendahuluan

Bab ini menjelaskan komponen awal laporan kerja praktik yang meliputi latar belakang, tujuan, manfaat,

dan rumusan masalah yang menjadi dasar kegiatan. Selain itu, dicantumkan juga lokasi dan waktu pelaksanaan kerja praktik. Metodologi yang digunakan serta sistematika penulisan laporan turut dijelaskan secara ringkas untuk memberikan gambaran alur pembahasan dalam laporan ini.

1.7.2. Bab II Profil Perusahaan

Bab ini berisi gambaran umum mengenai PT Adira Dinamika Multi Finance Tbk. sebagai tempat pelaksanaan kerja praktik. Penjelasan mencakup profil perusahaan secara menyeluruh, termasuk sejarah singkat, visi dan misi, serta bidang usaha yang dijalankan. Selain itu, bab ini juga menyampaikan informasi mengenai lokasi kantor pusat.

1.7.3. Bab III Tinjauan Pustaka

Bab ini berisi dasar teori dan konsep yang mendukung pelaksanaan proyek kerja praktik, seperti teknologi Docker, Jenkins, Vault, serta metode *deployment* yang digunakan.

1.7.4. Bab IV Analisis dan Perancangan Infrastruktur Sistem

Bab ini membahas analisis kebutuhan sistem CI/CD yang digunakan untuk proses *build*, *deploy*, dan pengelolaan *secrets* aplikasi *microservice* di lingkungan Kubernetes. Selain itu dijelaskan pula perancangan solusi, meliputi desain *pipeline* Jenkins, struktur *role* dan *policy* Vault, serta integrasi Kubernetes dengan Vault melalui Vault Injector untuk pengamanan *secrets* aplikasi.

1.7.5. Bab V Implementasi Sistem

Bab ini menjelaskan tahap implementasi sistem berdasarkan hasil analisis dan perancangan sebelumnya. Tahapan yang dibahas mencakup pembuatan *pipeline* Jenkins, *setup* Vault Injector di klaster Kubernetes, konfigurasi *service account*, *role*, dan *policy* di Vault, pembuatan *authentication method* Kubernetes, serta proses migrasi *secrets* dari Consul.

1.7.6.Bab VI Pengujian dan Evaluasi

Bab ini memuat proses pengujian untuk memastikan *pipeline* Jenkins berjalan sesuai fungsinya serta aplikasi yang di-*deploy* dapat mengambil *secrets* dari Vault Injector. Evaluasi dilakukan dengan cara melakukan *port-forwarding*, pemeriksaan proses aplikasi untuk memastikan Vault Injector aktif, serta validasi konfigurasi integrasi Vault di klaster Kubernetes.

1.7.7.Bab VII Kesimpulan dan Saran

Bab ini berisi kesimpulan dari seluruh proses kerja praktik yang telah dilakukan, mulai dari analisis kebutuhan hingga implementasi sistem CI/CD dan Vault. Selain itu, disampaikan pula saran untuk pengembangan lebih lanjut seperti penambahan proses *automated testing* menggunakan JMeter atau alat sejenis guna meningkatkan keandalan sistem di masa depan.

BAB II
PROFIL PERUSAHAAN

2.1. Profil PT Adira Dinamika Multi Finance Tbk.

PT Adira Dinamika Multi Finance Tbk, dikenal sebagai Adira Finance, adalah perusahaan pembiayaan terkemuka di Indonesia yang didirikan pada 13 November 1990 dan mulai beroperasi pada tahun 1991. Perusahaan ini terdaftar di Bursa Efek Indonesia (BEI) sejak tahun 2004 dengan kode emiten ADMF. Sejak tahun yang sama, Bank Danamon menjadi pemegang saham mayoritas, dan saat ini memiliki kepemilikan sebesar 92,07%. Sebagai anak perusahaan Bank Danamon, Adira Finance juga merupakan bagian dari Mitsubishi UFJ Financial Group (MUFG), salah satu grup keuangan terbesar di dunia [1].

Adira Finance menyediakan berbagai solusi pembiayaan, termasuk pembiayaan kendaraan bermotor (mobil dan sepeda motor), baik baru maupun bekas, serta pembiayaan non-otomotif seperti elektronik, *gadget*, furnitur, dan pinjaman multiguna. Perusahaan juga menawarkan layanan pembiayaan berbasis syariah melalui Unit Usaha Syariah Adira Finance.

Dalam upaya mendukung transformasi digital, Adira Finance telah meluncurkan berbagai *platform* digital, termasuk Adiraku, aplikasi layanan konsumen yang memudahkan pelanggan dalam mengakses layanan secara *real-time*. Selain itu, perusahaan juga mengembangkan *marketplace* seperti momobil.id dan momotor.id untuk jual beli kendaraan secara *online*.

Hingga pertengahan tahun 2022, Adira Finance mengoperasikan lebih dari 460 jaringan usaha di seluruh Indonesia dan didukung oleh sekitar 17.000 karyawan, melayani lebih dari 1,8 juta konsumen dengan total piutang yang dikelola mencapai Rp 41,1 triliun.

Gambar 2.1 Logo PT Adira Dinamika Multi Finance Tbk.

2.2. Visi dan Misi

PT Adira Dinamika Multi Finance Tbk. memiliki visi dan misi yang dijabarkan sebagai berikut:

1. Visi
   Menciptakan nilai bersama untuk meningkatkan kesejahteraan.
2. Misi
   Menyediakan beragam solusi keuangan sesuai dengan kebutuhan setiap pelanggan melalui sinergi dan ekosistem.

2.3. Lokasi

Kerja praktik dilaksanakan di PT Adira Dinamika Multi Finance Tbk., yang berlokasi di Millenium Centennial Center Lantai 53, 56–61, Jalan Jenderal Sudirman Kavling 25, Jakarta 12920. Kegiatan kerja praktik ini berlangsung selama tiga bulan, dimulai pada tanggal 6 Januari 2025 dan berakhir pada 6 April 2025. Selama periode tersebut, penulis ditempatkan di tim DevOps dan terlibat langsung dalam proses pengembangan serta pengelolaan infrastruktur teknologi informasi di lingkungan perusahaan.

[Halaman ini sengaja dikosongkan]

BAB III
TINJAUAN PUSTAKA

3.1. Microservices Architecture

Microservices Architecture (MSA) merupakan gaya arsitektur *cloud-native* yang terinspirasi dari Service-Oriented Architecture (SOA). MSA menyusun aplikasi sebagai kumpulan layanan kecil dan terpisah yang dapat dikembangkan, diuji, dan di-*deploy* secara independen menggunakan beragam *platform* dan teknologi. Setiap layanan berjalan dalam proses tersendiri dan saling berkomunikasi melalui API berbasis RESTful atau RPC. Arsitektur ini populer di industri karena menawarkan keunggulan seperti ketersediaan tinggi, fleksibilitas, skalabilitas, *loose coupling*, serta kecepatan pengembangan. Selain itu, penerapan MSA juga erat kaitannya dengan adopsi praktik DevOps di berbagai organisasi untuk meningkatkan kelincahan, kinerja, dan skalabilitas sistem [2].

3.2. Jenkins

Jenkins merupakan sebuah *platform* otomasi yang digunakan untuk membangun, menguji, dan melakukan *deployment* perangkat lunak secara otomatis. Salah satu fitur utama Jenkins adalah kemampuannya dalam membangun *pipeline*, yaitu rangkaian proses CI/CD (*Continuous Integration/Continuous Deployment*) yang didefinisikan dalam bentuk kode. Pendekatan ini memungkinkan proses pengembangan perangkat lunak berjalan secara terstruktur dan terotomatisasi. Namun demikian, dalam skala organisasi besar, penggunaan *pipeline* Jenkins sering kali menimbulkan permasalahan terkait duplikasi kode dan kesulitan dalam pemeliharaan *pipeline* tersebut [3].

3.3. CI/CD

CI/CD merupakan gabungan dari dua konsep utama, yaitu *Continuous Integration* (CI) dan *Continuous Delivery* (CD). *Continuous Integration* mengacu pada praktik integrasi kode secara berkelanjutan ke dalam repositori utama oleh pengembang, sedangkan *Continuous Delivery* melibatkan proses otomatisasi *build* dan *deployment* hingga ke lingkungan produksi. Melalui sistem CI/CD, kode yang dikembangkan langsung masuk ke dalam siklus *build* dan *testing* begitu di-*submit* oleh pengembang, sehingga mempercepat proses pengembangan perangkat lunak [4].

3.4. Bitbucket dan Webhook

Bitbucket merupakan sebuah *platform* berbasis Git yang digunakan untuk pengelolaan kode sumber secara terpusat, baik untuk pengembangan perangkat lunak secara individual maupun kolaboratif dalam tim. Selain menyediakan layanan *version control*, Bitbucket juga mendukung integrasi dengan Bitbucket *Pipeline*s untuk otomatisasi proses *Continuous Integration* dan *Continuous Deployment* (CI/CD), sehingga proses *build*, *test*, dan *deployment* dapat berjalan secara terstruktur dan terotomatisasi [5].

Webhook adalah mekanisme notifikasi otomatis berbasis HTTP yang digunakan untuk menghubungkan Bitbucket dengan sistem eksternal, seperti Jenkins, Slack, atau layanan lainnya. Webhook bekerja dengan mengirimkan permintaan HTTP POST ke URL tertentu saat terjadi *event* tertentu pada *repository*, seperti *push commit* atau *pull request*, yang kemudian dapat digunakan untuk memicu proses CI/CD secara otomatis tanpa intervensi manual.

3.5. Vault

HashiCorp Vault adalah sebuah alat manajemen *secret* yang dirancang untuk mengamankan, menyimpan, dan mengontrol akses terhadap informasi sensitif seperti kredensial, token API, sertifikat, serta data rahasia lainnya dalam lingkungan infrastruktur *cloud*. Vault menyediakan mekanisme enkripsi data dan kontrol akses yang ketat, sehingga memungkinkan organisasi untuk menjaga kerahasiaan data penting serta memastikan pemenuhan terhadap regulasi privasi dan keamanan yang berlaku secara global. Selain berperan dalam perlindungan data, Vault juga mendukung kelangsungan operasional sistem dengan memberikan akses rahasia secara aman, bahkan dalam situasi insiden yang memerlukan penyesuaian sistem secara cepat. Dengan integrasi bersama alat *monitoring* lainnya, Vault berkontribusi dalam menjaga keamanan data sekaligus meningkatkan keandalan infrastruktur IT [6].

3.6. Software Deployment Environment

Software Deployment Environment adalah tahapan lingkungan yang digunakan dalam proses pengembangan dan penerapan perangkat lunak untuk memastikan kualitas dan kesiapan sistem sebelum digunakan secara resmi. Terdapat tiga environment utama, yaitu *System Integration Testing* (SIT), *User Acceptance Testing* (UAT), dan *Production*. SIT digunakan untuk melakukan pengujian integrasi guna memastikan semua komponen sistem dapat berjalan dan berinteraksi sesuai dengan desain yang telah ditentukan. UAT merupakan tahap di mana pengguna akhir melakukan pengujian untuk memastikan bahwa aplikasi atau sistem sudah memenuhi kebutuhan bisnis dan persyaratan fungsional yang diharapkan. Sementara itu, *Production* adalah lingkungan operasional sebenarnya di mana aplikasi dijalankan secara langsung untuk melayani kebutuhan pengguna

secara nyata. Ketiga environment ini berperan penting dalam menjaga keandalan, keamanan, dan performa aplikasi sebelum diluncurkan ke pengguna akhir [7].

3.7. Kubernetes

Kubernetes adalah *platform open-source* yang digunakan untuk mengotomatisasi proses *deployment*, *scaling*, dan pengelolaan aplikasi berbasis *container*. *Platform* ini memungkinkan pengembang untuk berfokus pada pengembangan dan penerapan aplikasi tanpa perlu memikirkan detail infrastruktur yang mendasarinya. Kubernetes menggunakan pendekatan deklaratif, di mana pengguna menentukan kondisi aplikasi yang diinginkan, dan sistem akan secara otomatis menjaga kondisi tersebut. Selain itu, Kubernetes dilengkapi dengan mekanisme *self-healing* untuk mendeteksi dan memulihkan kegagalan secara otomatis, sehingga memberikan solusi yang andal dan fleksibel dalam pengelolaan aplikasi *container* di lingkungan produksi [8].

3.8. Docker

Docker adalah *platform open-source* yang digunakan untuk mengembangkan, mengemas, dan menjalankan aplikasi di dalam *container*. Container merupakan unit terisolasi yang berisi semua komponen yang diperlukan untuk menjalankan sebuah aplikasi, seperti kode program, *runtime*, pustaka, dan dependensi lainnya, sehingga aplikasi dapat berjalan secara konsisten di berbagai lingkungan. Dengan menggunakan Docker, pengembang dapat memastikan bahwa aplikasi yang dibuat di satu sistem dapat dijalankan dengan cara yang sama di sistem lain tanpa masalah perbedaan konfigurasi lingkungan. Selain itu, Docker memudahkan proses distribusi dan *deployment* aplikasi karena setiap *container* bersifat portabel, ringan, dan cepat untuk dijalankan [4].

3.9. Snyk dan Sonarqube

Snyk adalah sebuah alat *open-source* yang digunakan untuk mendeteksi kerentanan keamanan pada dependensi proyek, khususnya pada *package manager* seperti node modules. Tidak seperti alat seperti Retire.js yang memindai pustaka JavaScript, Snyk berfokus pada dependensi yang terpasang melalui *package manager* dan dilengkapi dengan basis data kerentanan miliknya sendiri yang selalu diperbarui oleh komunitas *open-source*. Snyk juga menyediakan integrasi dengan *platform* seperti GitHub, sehingga setiap *pull request* yang dibuat di *repository* dapat secara otomatis dipindai untuk mendeteksi potensi celah keamanan, mendukung penerapan sistem manajemen keamanan *open-source* secara berkelanjutan di lingkungan modern [9].

SonarQube merupakan alat analisis kualitas kode yang banyak digunakan untuk memeriksa keamanan, keandalan, dan *maintainability* kode program. SonarQube mendukung lebih dari 27 bahasa pemrograman seperti Java, Python, dan Ruby, serta dapat diintegrasikan dalam *pipeline* CI/CD pada praktik DevOps. SonarQube mampu mendeteksi berbagai potensi masalah dalam kode, seperti *bugs*, kerentanan keamanan (*vulnerabilities*), dan *code smells*, sehingga dapat membantu pengembang menjaga kualitas dan keamanan *codebase* yang dikembangkan. Dengan kemampuan tersebut, SonarQube berperan penting dalam meningkatkan keandalan dan performa aplikasi secara keseluruhan [4].

BAB IV
ANALISIS DAN PERANCANGAN INFRASTRUKTUR SISTEM

4.1. Analisis Sistem

Pada bab ini akan dijelaskan mengenai tahapan dalam membangun infrastruktur aplikasi yang digunakan untuk mendukung pengelolaan aset digital internal perusahaan melalui pengembangan *microservice* Asset-Management. Aplikasi ini dirancang sebagai solusi untuk memfasilitasi kebutuhan tim internal seperti *programmer*, UI/UX *designer*, dan pengguna internal lainnya dalam mengakses aset gambar, seperti ikon, maskot, dan elemen visual lainnya, secara terpusat dan terstruktur. Infrastruktur sistem ini bertujuan untuk menyediakan layanan yang terintegrasi serta terstandarisasi, sehingga proses pengelolaan dan distribusi aset menjadi lebih efisien dan terkontrol.

Aplikasi yang dikembangkan dalam kerja praktik ini bertujuan untuk mempermudah pengelolaan aset digital perusahaan agar dapat diakses oleh berbagai pihak yang membutuhkan di lingkungan PT Adira Dinamika Multi Finance Tbk. Dengan adanya aplikasi ini, diharapkan tidak terjadi redundansi data maupun inkonsistensi penggunaan aset visual antar tim, sehingga proses desain maupun pengembangan dapat berjalan dengan lebih terarah dan seragam.

Dalam proses pengembangan *microservice* ini, digunakan pendekatan arsitektur *microservice* untuk menjaga modularitas dan skalabilitas sistem. *Source code* aplikasi dikelola menggunakan Bitbucket sebagai *repository* utama, yang juga diintegrasikan dengan Webhook untuk menghubungkan perubahan kode secara otomatis ke Jenkins. Jenkins digunakan untuk membangun *pipeline* CI/CD yang mengotomatiskan

proses *build*, *testing*, hingga *deployment microservice* ke *environment System Integration Testing* (SIT). *Pipeline* ini dibuat tidak hanya untuk proses *deploy* ke SIT, tetapi juga sampai ke tahap produksi meskipun penggunaannya saat ini dibatasi hanya sampai SIT karena aplikasi bersifat internal.

Untuk menjaga keamanan dan kualitas aplikasi, *pipeline* Jenkins dilengkapi dengan proses integrasi beberapa *tools* tambahan. Salah satunya adalah Snyk, yang digunakan untuk melakukan pemindaian terhadap dependencies yang digunakan dalam project guna mendeteksi adanya kerentanan keamanan (*vulnerabilities*). Selain itu, diterapkan pula SonarQube untuk melakukan analisis kualitas kode secara statis (*static code analysis*), seperti mendeteksi *code smells*, *bug* potensial, dan kerentanan keamanan dalam *Source code*, sehingga dapat meningkatkan *maintainability* dan *reliability* aplikasi yang dikembangkan.

Dalam pengelolaan kredensial, digunakan sistem Vault *Secret management* sebagai solusi untuk menyimpan dan mengatur *secrets* secara terpusat, aman, dan terenkripsi. Penggunaan Vault ini memungkinkan *pipeline* mengambil kredensial yang diperlukan tanpa harus menuliskan data sensitif secara langsung di dalam konfigurasi *pipeline*, sehingga risiko kebocoran data dapat diminimalisir.

Dengan penerapan sistem ini, proses pengembangan *microservice* menjadi lebih terstruktur, aman, dan terstandarisasi, serta mendukung kebutuhan internal perusahaan dalam pengelolaan aset digital secara efektif. Seluruh rangkaian *pipeline* yang dibangun telah mencakup aspek kualitas kode, keamanan aplikasi, serta manajemen rahasia (*secret management*) untuk memastikan layanan *microservice* dapat berjalan secara optimal dan dapat dipelihara dalam jangka panjang.

4.2. Perancangan Infrastruktur Sistem

Desain sistem ini berfokus pada pengembangan *microservice* Asset-Management yang efisien, aman, dan dapat diakses oleh pengguna internal seperti *programmer*, UI/UX *designer*, maupun tim internal lain yang memerlukan aset gambar seperti ikon atau maskot perusahaan. Infrastruktur sistem dirancang untuk memastikan ketersediaan layanan yang stabil, proses *deployment* yang terstandarisasi, serta integrasi *pipeline* yang mendukung kualitas dan keamanan kode secara menyeluruh.

Infrastruktur sistem dibangun menggunakan pendekatan *Continuous Integration/Continuous Deployment* (CI/CD) melalui Jenkins. *Pipeline* Jenkins dirancang untuk mengotomatisasi proses *build*, *testing*, dan *deployment microservice* dari *environment* pengembangan hingga ke *System Integration Testing* (SIT). Meskipun *pipeline* juga mendukung *deployment* ke *production*, untuk saat ini batasan penggunaan hanya sampai SIT karena aplikasi diperuntukkan bagi kebutuhan internal.

Source code microservice dikelola melalui Bitbucket *repository*, yang telah diintegrasikan dengan webhook untuk memicu *pipeline* Jenkins secara otomatis setiap kali ada perubahan kode (*commit* atau *pull request*). Pada tahap awal *pipeline*, dilakukan proses *build* dan *dependency check* menggunakan Snyk untuk mendeteksi potensi *vulnerability* pada pustaka atau *package* yang digunakan. Setelah itu, *pipeline* menjalankan *static code analysis* menggunakan SonarQube untuk mengevaluasi kualitas kode, mendeteksi *code smells*, *bugs*, serta potensi celah keamanan.

Dalam proses *deployment*, *pipeline* mengambil kredensial penting secara aman dari Vault Secret management, yang memungkinkan pengambilan *secret*

tanpa harus menyimpannya secara *hardcoded* dalam *pipeline script* atau kode sumber. Hal ini meningkatkan keamanan pengelolaan data sensitif seperti API *key* atau *credential database*.

Hasil *build microservice* kemudian di-*deploy* ke *environment* SIT menggunakan *container* yang berjalan di Kubernetes klaster internal perusahaan. Proses *deployment* ini terstandarisasi sehingga meminimalisir konfigurasi manual dan mempercepat waktu rilis aplikasi ke *environment* pengujian.

Secara keseluruhan, infrastruktur sistem ini dirancang untuk:

1. Menyediakan layanan pengelolaan aset digital internal secara terpusat dan terstandarisasi.
2. Memastikan proses pengembangan *microservice* berjalan dengan prinsip DevSecOps melalui penerapan Snyk, SonarQube, dan Vault.
3. Meminimalisir *error* manusia melalui automasi CI/CD *pipeline* di Jenkins.
4. Menjamin keamanan data sensitif dengan pengelolaan *secret* terpusat di Vault.
5. Mendukung *scalability* dan *maintainability microservice* untuk kebutuhan internal perusahaan ke depannya.

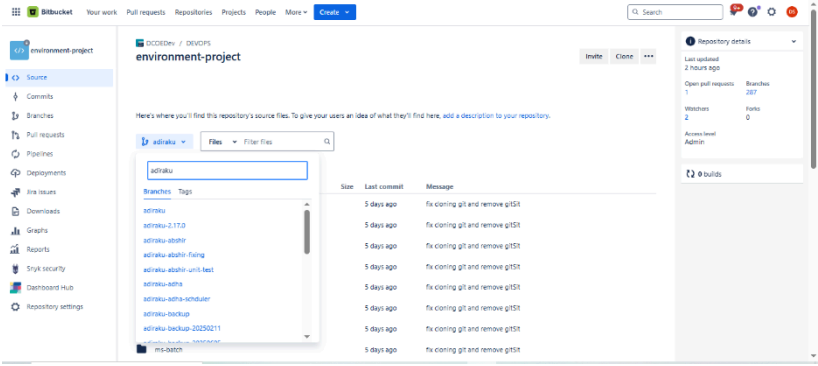
Dengan desain ini, pengembangan dan pengelolaan aplikasi Asset-Management menjadi lebih terkontrol, efisien, dan memenuhi standar keamanan serta kualitas kode yang baik.

[Halaman ini sengaja dikosongkan]

BAB V
IMPLEMENTASI SISTEM

Implementasi sistem dalam proyek kerja praktik ini mencakup pengembangan *microservice* Asset-Management beserta *pipeline* CI/CD yang terintegrasi hingga *environment* SIT di PT Adira Dinamika Multi Finance Tbk. Tahapan implementasi dimulai dari pembuatan *repository* di Bitbucket, pembuatan Dockerfile untuk kontainerisasi aplikasi, hingga konfigurasi *webhook* agar *pipeline* dapat berjalan otomatis melalui Jenkins. *Pipeline* yang dibangun mencakup proses *build*, pengujian, analisis kualitas kode menggunakan SonarQube, pemeriksaan keamanan dependensi dengan Snyk, serta *deployment* ke Kubernetes klaster pada *environment* SIT. Untuk menjaga kerahasiaan data penting seperti *credentials*, *pipeline* ini juga memanfaatkan Vault sebagai *secret management* system. Setelah proses *deployment* selesai, dilakukan pengujian fungsional untuk memastikan layanan dapat berjalan sesuai kebutuhan pengguna internal seperti tim pengembang dan desainer UI/UX.

5.1. Implementasi Pembuatan Repository Bitbucket



Gambar 5.1 Repository Adiraku.

Pembuatan *repository* untuk konfigurasi *deployment* dimulai dari membuat *branch* baru agar jika nanti terjadi kesalahan tidak menggangu *branch* master. Masuk ke tahapan pembuatan konfigurasi sebagai berikut:

5.1.1. Dockerfile

```
# Base Image
FROM node:20.18.3-alpine

# Create home directory & Copy source code
RUN mkdir -p /home/node/app/node_modules
WORKDIR /home/node/app
RUN wget https://releases.hashicorp.com/envconsul/0.12.1/envconsul_0.12.1_linux_386.zip
RUN unzip envconsul 0.12.1 linux 386.zip
COPY ./sourcecode/ /home/node/app/

# Compile code
RUN npm run build

# Config Timezone Asia/Jakarta
RUN apk add tzdata && cp /usr/share/zoneinfo/Asia/Jakarta /etc/localtime && echo "Asia/Jakarta" >  /etc/timezone

# User non root
RUN chown -R node:node /home/node/app
USER node

# Run Apps
CMD ["sh", "run.sh"]
```

Gambar 5.2 Dockerfile.

Pembuatan Dockerfile melalui koordinasi dengan tim pengembang terlebih dahulu. Sesuai dengan koordinasi, memakai node version 20.18.3 dan menggunakan *base image* alpine. Selanjutnya membuat *home directory* dan menginstal envconsul yang diperlukan untuk integrasi dengan Vault. Menyalin *sourcode* kedalam *home directory*-nya. Setelah itu menuliskan perintah untuk mem-*build* aplikasi dan konfigurasi *timezone* asia/jakarta. Untuk keamanan, dockerfile dibuat dengan *user* node bukan sebagai *user* root, agar tidak mudah seseorang mengubahnya dari *pod* di Kubenernetes. Tahap terakhir menuliskan perintah untuk menjalankan run.sh yang digunakan untuk menjalankan aplikasinya.

5.1.2. Run.sh

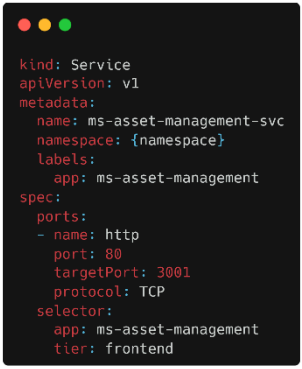
```
#!/bin/bash

if [ "$CHECK_ENV" = "preprod" ]; then
    /home/node/app/envconsul -consul-addr=http://consul-ui.infra -prefix
preprod/{appsname} -sanitize npm run start
elif [ "$CHECK_ENV" = "sit" ]; then
    ./envconsul -config=/vault/secrets/envconsul.hcl npm run start
else
    /home/node/app/envconsul -consul-addr=http://consul-ui.infra -prefix {appsname} -
sanitize npm run start
fi
```

Gambar 5.3 Run.sh.

Script di atas merupakan *script* Bash yang digunakan untuk menentukan perintah eksekusi aplikasi Node.js berdasarkan nilai dari variabel lingkungan CHECK_ENV. Jika nilai CHECK_ENV adalah "preprod", maka *script* akan menjalankan aplikasi menggunakan envconsul dengan konfigurasi *prefix preprod* dan alamat Consul tertentu untuk mengambil *secret* yang diperlukan sebelum menjalankan perintah npm run start. Jika nilai CHECK_ENV adalah "sit", maka aplikasi dijalankan menggunakan envconsul dengan *file* konfigurasi yang berada di *path* /vault/secrets/envconsul.hcl. Untuk kondisi selain "preprod" dan "sit", *script* akan menjalankan envconsul dengan *prefix default* tanpa *environment* spesifik dan kemudian menjalankan aplikasi menggunakan perintah yang sama. Dengan demikian, *script* ini berfungsi untuk mengatur pengambilan konfigurasi *environment* secara dinamis sesuai dengan lingkungan aplikasi dijalankan, baik itu di lingkungan *preprod*, SIT, maupun *default*.

5.1.3. Service.yaml



Gambar 5.4 Service.yaml.

Service.yaml bertujuan untuk *pod* dapat diakses oleh *pod* lain dan dapat diakses melalui *port-forwarding*. Service.yaml tersebut mendefinisikan sebuah Kubernetes Service, yang berfungsi sebagai abstraksi untuk menghubungkan *pod-pod* dalam klaster tanpa harus mengetahui IP masing-masing *pod* secara langsung. Service ini diberi nama ms-asset-management-svc dan berada di *namespace* tertentu (dapat berupa *default* atau *namespace* lain), sehingga bisa diakses oleh *pod* lain melalui DNS internal ms-asset-management-svc.{namespace}.svc.cluster.local. Pada bagian ports, Service ini membuka port 80 di level klaster, namun sebenarnya akan meneruskan *traffic* ke port 3001 di dalam *pod*, di mana aplikasi (seperti Node.js) berjalan. Selector pada Service ini memastikan bahwa hanya *pod* dengan label app: ms-asset-management dan tier: frontend yang menjadi target dari Service ini, sehingga setiap *request*

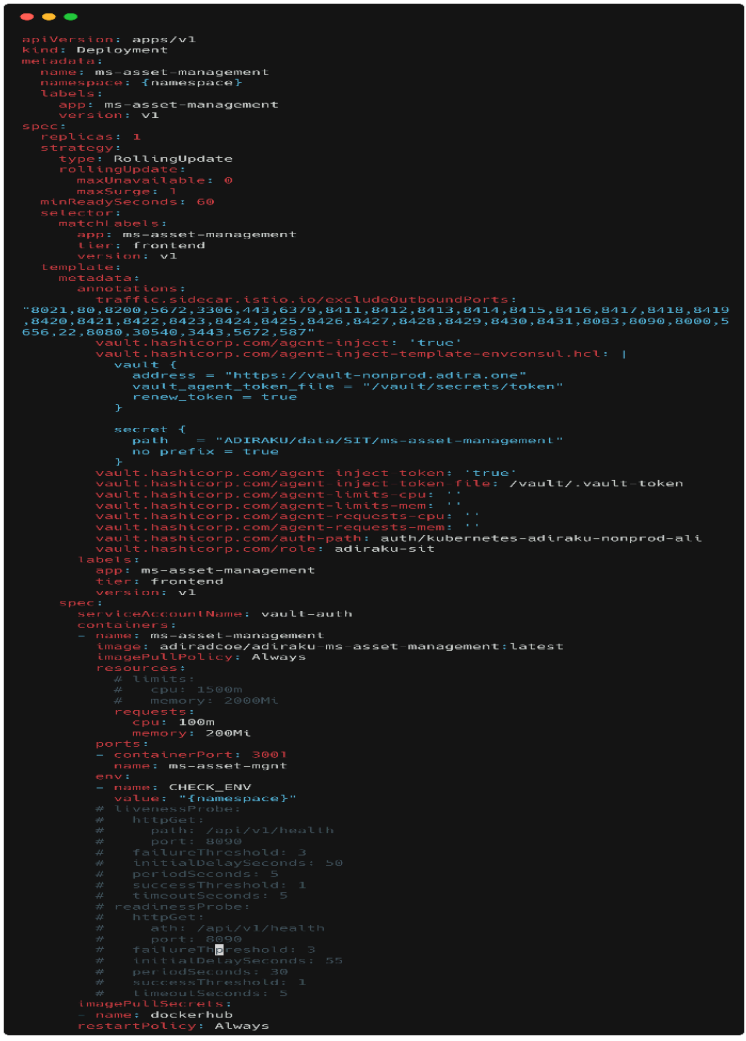
ke Service akan otomatis diteruskan ke *pod-pod* yang sesuai. Selain itu, label pada metadata hanya berfungsi sebagai penanda tambahan untuk keperluan seleksi atau *monitoring*, tidak mempengaruhi *routing traffic*.

5.1.4. Deployment.yaml

Deployment.yaml mendefinisikan sebuah Kuberne tes Deployment yang bertugas mengatur proses *deployment* aplikasi bernama ms-asset-management pada *namespace* tertentu. Deployment ini mengatur agar selalu ada satu replika *pod* yang berjalan, dengan strategi *update* menggunakan metode RollingUpdate, sehingga saat terjadi *update*, maksimal hanya ada satu *pod* tambahan (maxSurge: 1) dan tidak ada *pod* yang *unavailable* (maxUnavailable: 0). Setiap *pod* baru harus dalam keadaan siap minimal 60 detik (minReadySeconds: 60) sebelum dianggap aktif. Selector Deployment ini memastikan hanya *pod* dengan label app: ms-asset-management, tier: frontend, dan version: v1 yang dikelola oleh Deployment ini.

Bagian *template* menentukan spesifikasi *pod*, di mana *pod* dilengkapi dengan berbagai anotasi untuk integrasi dengan HashiCorp Vault guna melakukan injeksi *secrets* secara otomatis, seperti konfigurasi envconsul.hcl yang mengatur *path secret* di Vault serta pengaturan token. *Pod* juga menggunakan *service account* bernama vault-auth untuk proses autentikasi Vault di dalam klaster. Kontainer utama yang dijalankan bernama ms-asset-management, menggunakan *image* dari *repository* Docker adiradcoe/adiraku-ms-asset-management:latest dengan

imagePullPolicy: Always, memastikan *image* terbaru selalu di-*pull* saat *pod* dibuat.



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ms-asset-management
  namespace: {namespace}
  labels:
    app: ms-asset-management
    version: v1
spec:
  replicas: 1
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 0
      maxSurge: 1
  minReadySeconds: 60
  selector:
    matchLabels:
      app: ms-asset-management
      tier: frontend
      version: v1
  template:
    metadata:
      annotations:
        traffic.sidecar.istio.io/excludeOutboundPorts:
"8021,80,8200,5672,3306,443,6379,8411,8412,8413,8414,8415,8416,8417,8418,8419
,8420,8421,8422,8423,8424,8425,8426,8427,8428,8429,8430,8431,8083,8090,8000,5
656,22,8080,30540,3443,5672,587"
        vault.hashicorp.com/agent-inject: 'true'
        vault.hashicorp.com/agent-inject-template-envconsul.hcl: |
          vault {
            address = "https://vault-nonprod.adira.one"
            vault_agent_token_file = "/vault/secrets/token"
            renew_token = true
          }

          secret {
            path   = "ADIRAKU/data/SIT/ms-asset-management"
            no_prefix = true
          }
        vault.hashicorp.com/agent-inject-token: 'true'
        vault.hashicorp.com/agent-inject-token-file: /vault/.vault-token
        vault.hashicorp.com/agent-limits-cpu: ''
        vault.hashicorp.com/agent-limits-mem: ''
        vault.hashicorp.com/agent-requests-cpu: ''
        vault.hashicorp.com/agent-requests-mem: ''
        vault.hashicorp.com/auth-path: auth/kubernetes-adiraku-nonprod-ali
        vault.hashicorp.com/role: adiraku-sit
      labels:
        app: ms-asset-management
        tier: frontend
        version: v1
    spec:
      serviceAccountName: vaull-auth
      containers:
      - name: ms-asset-management
        image: adiradcoe/adiraku-ms-asset-management:latest
        imagePullPolicy: Always
        resources:
          # limits:
          #   cpu: 1500m
          #   memory: 2000Mi
          requests:
            cpu: 100m
            memory: 200Mi
        ports:
        - containerPort: 3001
          name: ms-asset-mgnt
        env:
        - name: CHECK_ENV
          value: "{namespace}"
        # livenessProbe:
        #   httpGet:
        #     path: /api/v1/health
        #     port: 8090
        #   failureThreshold: 3
        #   initialDelaySeconds: 50
        #   periodSeconds: 5
        #   successThreshold: 1
        #   timeoutSeconds: 5
        # readinessProbe:
        #   httpGet:
        #     path: /api/v1/health
        #     port: 8090
        #   failureThreshold: 3
        #   initialDelaySeconds: 55
        #   periodSeconds: 30
        #   successThreshold: 1
        #   timeoutSeconds: 5
      imagePullSecrets:
      - name: dockerhub
      restartPolicy: Always
```

Gambar 5.5 Deployment.yaml.

Resource request di-*set* ringan yaitu CPU 100m dan memory 200Mi, tanpa limit eksplisit yang diatur. Kontainer ini mengekspos port 3001 dengan nama ms-asset-mgnt agar bisa diakses oleh Service atau *pod* lain. Variabel *environment* CHECK_ENV di-*set* sesuai *namespace* yang digunakan. Konfigurasi untuk livenessProbe dan readinessProbe disiapkan namun dikomentari, sehingga saat ini Health Check *pod* belum diaktifkan. Deployment juga menggunakan *secret* dockerhub untuk otentikasi *image registry*, serta *restart policy pod* di-*set* selalu Always, agar *pod* selalu di-*restart* jika terjadi kegagalan.

Secara keseluruhan, YAML ini mengatur *lifecycle* aplikasi *microservice* secara terintegrasi dengan Vault untuk manajemen *secret*, serta memanfaatkan fitur rolling update Kubernetes untuk *update* tanpa *downtime*.

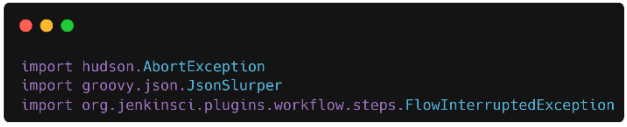
5.1.5. Sonar-project

Gambar 5.6 Sonar-project.properties.

File sonar-project.properties di atas digunakan untuk konfigurasi analisis kode statis menggunakan SonarQube pada project bernama adiraku-ms-asset-ma nagement. Properti sonar.projectKey dan sonar.project Name masing-masing berfungsi untuk menentukan identitas unik *project* di dalam SonarQube *server*, sehingga hasil analisis dapat dipisahkan dari *project*

lain. Baris konfigurasi sonar.javascript.lcov.reportPaths yang dikomentari (#) seharusnya digunakan untuk menentukan *path file* laporan Coverage hasil *testing* JavaScript dalam format LCOV (misalnya ./coverage/lcov.info), yang memungkinkan SonarQube untuk mengukur seberapa besar bagian kode yang sudah diuji melalui Unit Test. Karena baris ini masih dikomentari, maka untuk sementara laporan Coverage tidak dibaca oleh SonarQube, sehingga metrik terkait Coverage tidak akan muncul di *dashboard* SonarQube proyek ini.

5.1.6. Jenkinsfile



Gambar 5.7 Bagian Import Jenkinsfile.

Pada bagian awal Jenkinsfile ini, terdapat beberapa impor *library* yang digunakan untuk mendukung proses *pipeline*. hudson.AbortException diimpor untuk memungkinkan *pipeline* menghentikan proses *build* secara paksa apabila terjadi kondisi *error* tertentu. Selanjutnya, groovy.json.JsonSlurper digunakan untuk memproses atau melakukan *parsing* terhadap data berformat JSON, misalnya ketika *pipeline* perlu membaca konfigurasi dari *file* JSON atau hasil respons API. Terakhir, org.jenkinsci.plugins.workflow.steps.FlowInterruptedException disiapkan untuk menangani situasi di mana *pipeline* terhenti secara tidak normal, seperti akibat *manual abort* atau *timeout*, sehingga *exception* ini dapat ditangani dengan baik di dalam *script pipeline*.

```
def devopsTeamExpression = {
    def devopsTeam = ['andikanggakusuma', 'febriansyah',
'v_adha_setyawan', 'v_delai_setyawan',
    'v_irfan_nurrohman', 'v_ravilo_divanil', 'v_jovan_steven',
'abshirhammam', 'v_even_thamrin', 'admindevops']
    def causes = currentBuild.rawBuild.getCauses()
    def startedByUserId = null

    for (cause in causes) {
        if (cause instanceof hudson.model.Cause$UserIdCause) {
            startedByUserId = cause.getUserId()
            break
        }
    }
    return startedByUserId != null &&
devopsTeam.contains(startedByUserId)
}

def loadSecretsFromVault() {
    unstash 'vault-secrets'
    def props = readProperties file: 'secrets.env'
    env.KUBECONFIG_ADIRAKU_NONPROD =
props.KUBECONFIG_ADIRAKU_NONPROD.replace('\\n', '\n').trim()
    env.KUBECONFIG_ADIRAKU_PROD =
props.KUBECONFIG_ADIRAKU_PROD.replace('\\n', '\n').trim()
    env.TELEGRAM_BOT_TOKEN = props.TELEGRAM_BOT_TOKEN
    env.TELEGRAM_CHAT_ID = props.TELEGRAM_CHAT_ID
    env.MYSQL_HOSTNAME = props.MYSQL_HOSTNAME
    env.MYSQL_USERNAME = props.MYSQL_USERNAME
    env.MYSQL_PASSWORD = props.MYSQL_PASSWORD
    env.MYSQL_DB_PORT = props.MYSQL_DB_PORT
    env.VAULT_TOKEN_NONPROD = props.VAULT_TOKEN_NONPROD
    env.BITBUCKET_DCOE = props.BITBUCKET_DCOE
    env.CTX_CLUSTER1 = props.CTX_CLUSTER1
    env.CTX_CLUSTER2 = props.CTX_CLUSTER2
    env.CTX_CLUSTER1_PROD = props.CTX_CLUSTER1_PROD
    env.CTX_CLUSTER2_PROD = props.CTX_CLUSTER2_PROD
    env.DOCKER_USERNAME = props.DOCKER_USERNAME
    env.DOCKER_PASSWORD = props.DOCKER_PASSWORD
    env.BITBUCKET_USERNAME = props.BITBUCKET_USERNAME
    env.BITBUCKET_PASSWORD = props.BITBUCKET_PASSWORD
    env.SNYK_TOKEN = props.SNYK_TOKEN
    env.SONARQUBE_TOKEN = props.SONARQUBE_TOKEN
    env.SONARQUBE_HOST_URL = props.SONARQUBE_HOST_URL
    env.DEFECTDOJO_TOKEN = props.DEFECTDOJO_TOKEN
}
```

Gambar 5.8 Bagian Function Jenkinsfile.

Bagian kode ini terdiri dari dua fungsi penting dalam Jenkins *pipeline*. Fungsi pertama, devopsTeamExpression, digunakan untuk memeriksa apakah *user* yang memicu *build pipeline* adalah anggota tim DevOps tertentu, dengan cara mencocokkan userId dari penyebab *build* (Cause$UserIdCause) dengan daftar *user* yang telah ditentukan dalam variabel devopsTeam.

Fungsi ini mengembalikan nilai boolean yang menandakan apakah *user* tersebut adalah bagian dari tim DevOps. Sementara itu, fungsi kedua, loadSecretsFromVault, bertugas untuk memuat dan menyimpan *secrets* yang diambil dari *file* secrets.env ke dalam *environment variabels pipeline* menggunakan fungsi readProperties, setelah terlebih dahulu mengambil *stash* vault-secrets. *Secrets* yang dimuat meliputi berbagai konfigurasi penting seperti *credential* Kubernetes (KUBECONFIG_ADIRAKU_NONPROD dan KUBECONFIG_ADIRAKU_PROD), token untuk Telegram, informasi *database* MySQL, *credential* Bitbucket, Docker *registry*, hingga token untuk *tools* keamanan seperti Snyk, SonarQube, dan DefectDojo, yang semuanya akan digunakan di tahap-tahap *pipeline* selanjutnya.

Bagian awal Jenkinsfile mendefinisikan blok utama *pipeline* dengan *agent* yang berjalan pada node bertanda *built-in*, yang artinya *build* akan dijalankan di agent Jenkins *default*. Pada bagian *environment*, dideklarasikan sejumlah variabel *environment* yang akan digunakan sepanjang *pipeline*, seperti appsName untuk menentukan nama aplikasi (ms-asset-management), serta *registry* yang berisi lokasi Docker Image *repository* berdasarkan nama aplikasi tersebut. Variabel registryCredential menyimpan kredensial untuk mengakses Docker Hub, sedangkan gitBranch berisi nama *branch* Git yang digunakan yaitu main. Ada juga variabel getUser yang masih kosong, serta beberapa variabel terkait koneksi *database* seperti DB_HOST, DB_USER, DB_PASSWORD, dan DB_PORT yang diambil dari Jenkins Credentials Store untuk menjaga kerahasiaan data sensitif, sementara nama *database* DB_NAME ditetapkan secara langsung sebagai asset_management. Selain itu terdapat konfigurasi alamat Vault *production*

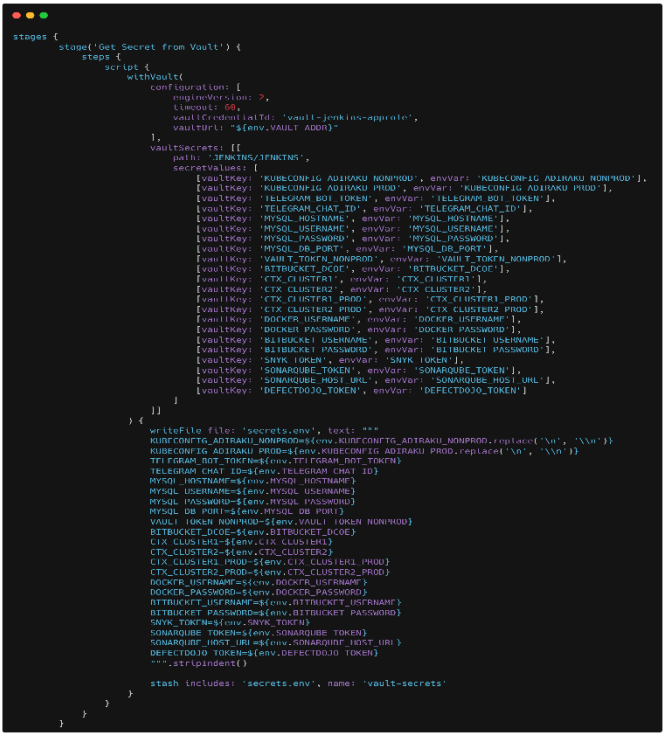
(VAULT_ADDR), nama produk (productName), ID *scanning* tool (idScan), dan nama *repository* (repoName). Pada blok *tools*, *pipeline* diatur untuk menggunakan Node.js versi node20183 yang sebelumnya telah disiapkan di Jenkins, memastikan *pipeline* menggunakan *environment* Node.js yang sesuai untuk *build* atau *testing*.

```
pipeline {
    agent { label 'built-in' }

    environment {
        appsName = "ms-asset-management"
        registry = "adiradcoe/adiraku-$appsName"
        registryCredential = 'dockerhub'
        gitBranch = "main"
        getUser = ''
        DB_HOST = credentials('db_host')
        DB_USER = credentials('db_user')
        DB_PASSWORD = credentials('db_password')
        DB_NAME = 'asset_management'
        DB_PORT = credentials('db_port')
        VAULT_ADDR = 'https://vault-prod.adira.one'
        productName = "adiraku"
        idScan = "98"
        repoName = "asset-management"
    }

    tools {nodejs "node20183"}
```

Gambar 5.9 Bagian Awal Jenkinsfile.



Gambar 5.10 Stage Get Secret From Vault.

Pada stage ‘Get Secret from Vault’ ini, *pipeline* Jenkins digunakan untuk mengambil *secret* dari HashiCorp Vault menggunakan *plugin* withVault. Konfigurasinya mengatur Vault versi engine 2, dengan kredensial vault-jenkins-approle dan URL Vault yang diambil dari *environment* variabel VAULT_ADDR. *Secrets* diambil dari *path* JENKINS/JENKINS, lalu setiap *key* di Vault dipetakan ke *environment* variabel yang sesuai di Jenkins, seperti misalnya KUBECONFIG_ADIRAKU_NONPROD, TELEGRAM_BOT_TOKEN, MYSQL_USERNAME, dan

lainnya. Setelah *secrets* dimuat ke *environment* variabel, *secrets* tersebut dituliskan ke *file* secrets.env dalam format *key-value* agar bisa digunakan di stage *pipeline* berikutnya, lalu *file* tersebut di-*stash* dengan nama vault-secrets sehingga bisa di-*unstash* kembali pada stage lain di *pipeline*. Tahapan ini penting untuk menjaga kerahasiaan data sensitif seperti token, *credential database*, dan konfigurasi klaster Kubernetes selama proses *build* berlangsung.

```
stage ('Bitbucket Check'){
            steps {
                script {
                    loadSecretsFromVault()
                    def response = sh(
                        script: """
                            set +x
                            curl -s -X POST -u "${BITBUCKET_DCOE}" https://bitbucket.org/site/oauth2/access_token -
d grant_type=client_credentials
                            set -x
                        """,
                        returnStdout: true
                    ).trim()
                    def json = readJSON(text: response)
                    def accessToken = json.access_token
                    sh(script: """
                        set +x
                        mkdir -p output
                        curl -s --request GET \\
                            --url 'https://api.bitbucket.org/2.0/repositories/dcoedev/${repoName}/pullrequests' \\
                            --header 'Authorization: Bearer ${accessToken}' \\
                            --header 'Accept: application/json' \\
                            > output/file.json
                        set -x
                    """)
                    def fileContent = readFile(file: 'output/file.json')
                    def jsonBranch = readJSON(text: fileContent)
                    def gitSourcePr = ""
                    def gitPrId = ""
                    def filteredPR = jsonBranch.values.find { pr ->
                        pr.state == "OPEN" && pr.destination.branch.name == "main"
                    }
                    if (filteredPR) {
                        gitSourcePr = filteredPR?.source.branch.name ?: ""
                        gitPrId = filteredPR?.id ?: ""
                    }
                    echo "Git Source PR: ${gitSourcePr}"
                    echo "Git PR ID: ${gitPrId}"
                    env.accessToken = accessToken
                    env.gitPrId = gitPrId
                    env.gitSourcePr = gitSourcePr
                }
            }
        }
```

Gambar 5.11 Stage Bitbucket Check.

Pada stage 'Bitbucket Check' ini, *pipeline* Jenkins digunakan untuk melakukan pengecekan *pull request* (PR) yang terbuka di *repository* Bitbucket tertentu.

Pertama-tama, *secrets* yang diperlukan seperti kredensial Bitbucket di-*load* melalui fungsi loadSecretsFromVault(). Kemudian Jenkins menjalankan perintah curl untuk meminta *access* token OAuth2 dari Bitbucket menggunakan *credential* yang sudah di-*set*, dan hasil responnya dibaca dalam format JSON untuk mendapatkan *access* token. Dengan token ini, *pipeline* mengirimkan *request* GET ke Bitbucket API untuk mengambil daftar PR pada *repository* yang ditentukan, khususnya untuk *branch* main. Data hasil respons disimpan ke *file* output/file.json, lalu dibaca kembali untuk mencari PR yang statusnya "OPEN" dan ditujukan ke *branch* main. Jika ada, *pipeline* mengambil informasi *branch* source PR dan ID PR untuk disimpan sebagai *environment* variabel (env.gitSourcePr dan env.gitPrId) agar bisa digunakan di stage berikutnya. Proses ini memungkinkan *pipeline* untuk mengenali PR aktif yang relevan secara otomatis selama proses *build* berlangsung.

Pada stage 'Cloning Git' ini, *pipeline* Jenkins bertugas untuk menentukan *branch* Git mana yang akan digunakan untuk proses *build* berdasarkan beberapa kondisi. Pertama, *secrets* yang dibutuhkan di-*load* melalui fungsi loadSecretsFromVault(). Kemudian *pipeline* mengecek apakah variabel gitRelease dan gitSourcePr kosong; jika keduanya kosong, maka *pipeline* menggunakan *branch default* (gitBranch) dari *environment* untuk kebutuhan *non-production*. Jika gitRelease berisi dan *job* dijalankan oleh *user* dari tim DevOps tertentu (divalidasi lewat fungsi devopsTeamExpression()), maka branch gitRelease akan digunakan untuk kebutuhan *deployment production*. Jika tidak memenuhi dua kondisi tersebut, *pipeline* akan menggunakan *branch* dari *pull request* (gitSourcePr). Setelah *branch* ditentukan, proses *clone repository* Bitbucket dilakukan ke dalam *folder*

sourcecode, menggunakan *script custom* git-askpass.sh untuk menangani autentikasi username dan password Bitbucket secara aman. Setelah proses *clone* selesai, *file* penting seperti Dockerfile, run.sh, dan sonar-project.properties dari *folder* aplikasi disalin ke *folder sourcecode*, termasuk *file* konfigurasi .npmrc untuk kebutuhan *build* Node.js.

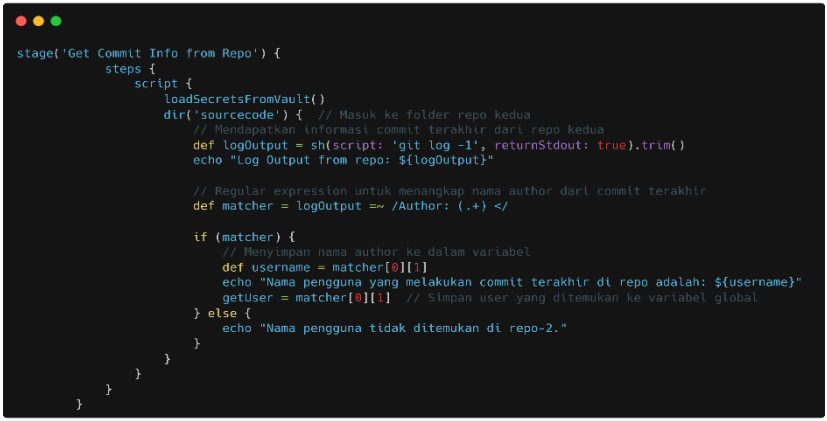
```
stage('Cloning Git') {
            steps {
                script {
                    loadSecretsFromVault()
                    def branchUsed
                    // Membuat if else condition untuk memilih branch yang akan dipilih sesuai kondisi
                    /*
                        Gunakan gitBranch yang telah di set pada Environment diatas, ketika ingin melakukan deploy
pada
                        tahap non-production secara manual atau dengan merged pr
                        Kondisi ini akan berjalan apabila gitRelease tidak ada isi, dan tidak ada PR pada
repository
                    */
                    if (!gitRelease && !env.gitSourcePr){
                        branchUsed = gitBranch
                    }
                    /*
                        Gunakan gitRelease untuk melakukan deployment tahap Production.
                        Kondisi ini akan berjalan ketika gitRelease telah mempunyai isi dan user tertentu yang
menjalankan job pada Jenkins
                    */
                    else if(gitRelease != "" && devopsTeamExpression()){
                        branchUsed = gitRelease
                    }
                    else {
                        branchUsed = gitSourcePr
                    }

                    // Membuat dan memasukkan hasil clone ke dalam folder sourcecode
                    dir('sourcecode'){
                        writeFile file: 'git-askpass.sh', text: """#!/bin/bash
                            case "\$1" in
                                Username*) echo "${BITBUCKET_USERNAME}" ;;
                                Password*) echo "${BITBUCKET_PASSWORD}" ;;
                            esac
                        """
                        sh 'chmod +x git-askpass.sh'

                        // Use credential helper instead of URL credentials
                        withEnv(["GIT_ASKPASS=${WORKSPACE}/git-askpass.sh"]) {
                            checkout([
                                $class: 'GitSCM',
                                branches: [[name: branchUsed]],
                                userRemoteConfigs: [[
                                    url: "https://bitbucket.org/dcoedev/${repoName}.git"
                                ]]
                            ])
                        }
                    }

                    // Melakukan copy dan paste untuk memasukkan file ke dalam folder sourcecode
                    sh "cp -rf $appsName/Dockerfile $appsName/run.sh $appsName/sonar-project.properties
sourcecode/"
                    sh 'cp -rf /var/lib/jenkins/.npmrc ./sourcecode/'
                }
            }
        }
```

Gambar 5.12 Stage Cloning Git.



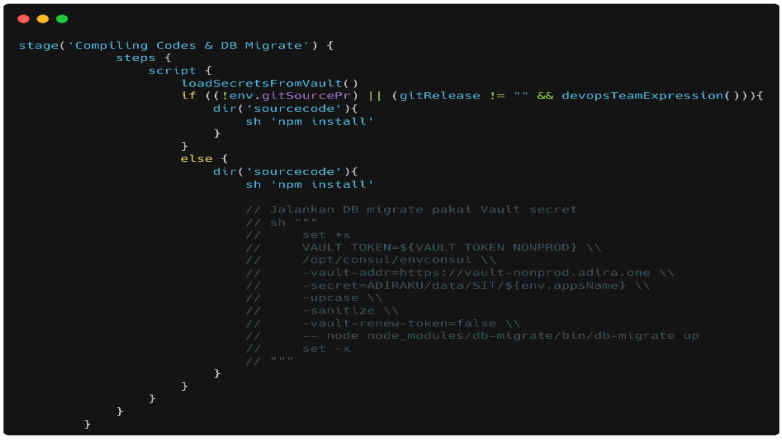
```
stage('Get Commit Info from Repo') {
            steps {
                script {
                    loadSecretsFromVault()
                    dir('sourcecode') {  // Masuk ke folder repo kedua
                        // Mendapatkan informasi commit terakhir dari repo kedua
                        def logOutput = sh(script: 'git log -1', returnStdout: true).trim()
                        echo "Log Output from repo: ${logOutput}"

                        // Regular expression untuk menangkap nama author dari commit terakhir
                        def matcher = logOutput =~ /Author: (.+) </

                        if (matcher) {
                            // Menyimpan nama author ke dalam variabel
                            def username = matcher[0][1]
                            echo "Nama pengguna yang melakukan commit terakhir di repo adalah: ${username}"
                            getUser = matcher[0][1]  // Simpan user yang ditemukan ke variabel global
                        } else {
                            echo "Nama pengguna tidak ditemukan di repo-2."
                        }
                    }
                }
            }
        }
```

Gambar 5.13 Stage Get Commit Info From Repo.

Pada stage 'Get *Commit* Info from Repo', *pipeline* bertugas untuk mengambil informasi *commit* terakhir dari *repository* yang telah dikloning di *folder sourcecode*. Setelah *secrets* dimuat menggunakan loadSecretsFromVault(), *pipeline* masuk ke direktori *sourcecode* dan menjalankan perintah git log -1 untuk mengambil detail *commit* terakhir, lalu hasil *output* tersebut disimpan dalam variabel logOutput. Dengan menggunakan regular expression, *pipeline* mengekstrak nama *author* dari *commit* tersebut, dan jika ditemukan, nama pengguna yang melakukan *commit* terakhir akan dicetak melalui perintah echo dan disimpan ke dalam variabel global getUser. Jika nama pengguna tidak dapat ditemukan, *pipeline* akan menampilkan pesan bahwa nama pengguna tidak ditemukan pada *repository* tersebut. Tahapan ini berguna untuk melacak siapa yang terakhir kali melakukan perubahan pada kode sumber.



Gambar 5.14 Stage Compiling Codes & DB Migrate.

Pada stage ‘Compiling Codes & DB Migrate’, *pipeline* melakukan proses instalasi dependensi proyek Node.js dengan menjalankan perintah npm install di dalam *folder sourcecode*, setelah *secrets* dimuat melalui loadSecretsFromVault(). Tahapan ini juga memeriksa kondisi apakah *pipeline* dijalankan bukan dari *Pull Request* (!env.gitSourcePr) atau untuk proses *release production* (gitRelease terisi dan dijalankan oleh anggota tim DevOps yang valid melalui fungsi devopsTeamExpression()). Jika salah satu kondisi ini terpenuhi, maka hanya proses instalasi dependensi yang dijalankan. Namun, untuk proses dari *Pull request* (PR) atau pengembangan, selain npm install, disiapkan pula (meskipun dikomentari) *script* untuk menjalankan proses database migration menggunakan *tool* envconsul dan *secret* dari Vault. Hal ini mengindikasikan bahwa migrasi *database* disiapkan untuk dieksekusi otomatis di lingkungan *non-production* dengan pengaturan *environment* dari Vault, meskipun belum diaktifkan dalam *script* tersebut.

```
stage('Unit Test') {
            when {
                expression { !gitRelease }
            }
            steps {
                dir('sourcecode') {
                    script {
                    loadSecretsFromVault()
                        try {
                            // Jalankan envconsul pakai token login
                            sh """
                                set +x
                                VAULT_TOKEN=${VAULT_TOKEN_NONPROD} \\
                                /opt/consul/envconsul \\
                                -vault-addr=https://vault-nonprod.adira.one \\
                                -secret=ADIRAKU/data/SIT/${env.appsName} \\
                                -upcase \\
                                -sanitize \\
                                -vault-renew-token=false \\
                                -no-prefix \\
                                -- npm run test:coverage
                                set -x
                            """

                            env.stageResultunit = true
                        } catch (Exception e) {
                            unstable("${STAGE_NAME} failed!")
                            currentBuild.result = 'SUCCESS'
                            env.stageResultunit = false
                        }
                    }
                }
            }
        }
```

Gambar 5.15 Stage Unit Test.

Pada stage ‘Unit Test’ ini, *pipeline* dikonfigurasi untuk hanya dijalankan ketika variabel gitRelease kosong, artinya proses ini hanya berlaku untuk tahap pengembangan atau *non-production*. Di dalam *folder sourcecode*, *pipeline* memuat *secrets* melalui fungsi loadSecretsFromVault(), lalu menjalankan Unit Test menggunakan perintah npm run test:coverage yang dibungkus dengan *tool* envconsul. *Tool* ini digunakan untuk menyuntikkan *environment* variabel dari Vault *non-production* ke dalam proses *testing* tanpa harus menuliskannya langsung di *file* konfigurasi, memastikan keamanan *secrets*. Jika proses Unit Test berhasil, maka *environment* variabel stageResultunit di-*set* ke *true*. Namun jika terjadi *error* selama *testing*, *pipeline* menangkap *exception* tersebut, menandai stage sebagai *unstable*, namun tetap melanjutkan *build*

dengan status SUCCESS, serta mengatur stageResultunit ke false, sehingga kegagalan Unit Test tidak langsung menggagalkan *pipeline* secara keseluruhan.

Pada stage '*Scanning* with Snyk', *pipeline* melakukan proses *scanning* keamanan menggunakan Snyk untuk memeriksa dependensi proyek terhadap kerentanan. *Secrets* terlebih dahulu dimuat melalui fungsi loadSecretsFromVault(). Di dalam *folder sourcecode*, dilakukan otentikasi ke Snyk menggunakan token (snyk auth), lalu *scanning* dijalankan dengan perintah snyk test yang disetel untuk mendeteksi kerentanan dengan tingkat minimal medium. Hasil *scanning* juga diekspor ke *file* JSON (adiraku-${appsName}.json) untuk kebutuhan dokumentasi atau integrasi selanjutnya. Selanjutnya, hasil *scanning* dikirim ke *platform* DefectDojo melalui API *reimport-scan*, yang memungkinkan hasil Snyk diimpor dan ditampilkan dalam DefectDojo untuk kebutuhan manajemen kerentanan aplikasi, termasuk fitur grouping dan auto-close temuan lama. Jika proses *scanning* berhasil, variabel *environment* stageResultsnyk diatur menjadi true; jika gagal (misalnya karena *error* Snyk atau *upload*), *exception* akan ditangkap, stage ditandai sebagai *unstable*, *pipeline* tetap dianggap berhasil (SUCCESS), dan stageResultsnyk diset ke false, agar tidak menggagalkan keseluruhan *pipeline*.

```
// Stage untuk melakukan scan dengan Snyk
        stage('Scanning with Snyk') {
            steps {
                script {
                    loadSecretsFromVault()
                    try {
                        dir('sourcecode'){
                            sh """
                                set +x
                                snyk auth ${SNYK_TOKEN} || true
                                set -x
                                snyk test --severity-threshold=medium --
project-name=${appsName} --remote-repo-url=adiraku || true
                                snyk test --severity-threshold=medium --
project-name=${appsName} --remote-repo-url=adiraku --json-file-
output=adiraku-${appsName}.json
                                snyk monitor --project-name=${appsName} --
remote-repo-url=adiraku
                            """

                            sh """
                                curl -X 'POST' \\

 'http://defectdojo.adira.one/api/v2/reimport-scan/' \\
                                    -H 'accept: application/json' \\
                                    -H 'Content-Type: multipart/form-data'
\\
                                    -H 'Authorization: Token
${DEFECTDOJO_TOKEN}' \\
                                    -F 'minimum_severity=Low' \\
                                    -F 'active=true' \\
                                    -F 'verified=true' \\
                                    -F 'scan_type=Snyk Scan' \\
                                    -F
'file=@\"${productName}-${appsName}.json\"' \\
                                    -F 'product_type_name=${productName}'
\\
                                    -F 'product_name=${productName}' \\
                                    -F 'engagement_name=snyk-${appsName}'
\\
                                    -F 'auto_create_context=true' \\
                                    -F 'close_old_findings=true' \\
                                    -F
'close old findings product scope=false' \\
                                    -F
'create_finding_groups_for_all_findings=true' \\
                                    -F 'group_by=file_path' \\
                                    -F 'push_to_jira=true' \\
                                    -F 'deduplication_on_engagement=true'
                            """
                            env.stageResultsnyk = true
                        }
                    }
                    catch (Exception e) {
                        // Untuk menjaga pipeline tetap berjalan walaupun
terdapat temuan bug/failed oleh snyk
                        unstable("${STAGE_NAME} failed!")
                        currentBuild.result = 'SUCCESS'

                        // Set variable untuk false ketika scan snyk gagal
                        env.stageResultsnyk = false
                    }
                }
            }
        }
```

Gambar 5.16 Stage Scanning Snyk.

```
stage('Scanning with Sonarqube') {
            when {
                expression{
                    !gitRelease
                }
            }
            steps {
                script {
                    loadSecretsFromVault()
                    try {
                        dir('sourcecode'){
                            sh """
                                set +x
                                sonar-scanner \
                                -Dsonar.login=${SONARQUBE_TOKEN}\
                                -Dsonar.host.url=${SONARQUBE_HOST_URL}
                                set -x
                            """

                            sh """
                                curl -X 'POST' \\

 'http://defectdojo.adira.one/api/v2/reimport-scan/' \\
                                    -H 'accept: application/json' \\
                                    -H 'Content-Type: multipart/form-data'
\\
                                    -H 'Authorization: Token
${DEFECTDOJO_TOKEN}' \\
                                    -F 'minimum_severity=Low' \\
                                    -F 'active=true' \\
                                    -F 'verified=true' \\
                                    -F 'scan_type=SonarQube API Import' \\
                                    -F 'product_type_name=${productName}'
\\
                                    -F 'product_name=${productName}' \\
                                    -F
'engagement_name=sonarqube-${appsName}' \\
                                    -F 'auto_create_context=true' \\
                                    -F 'close_old_findings=true' \\
                                    -F
'close_old_findings_product_scope=false' \\
                                    -F
'create_finding_groups_for_all_findings=true' \\
                                    -F 'group_by=component_name' \\
                                    -F 'push_to_jira=true' \\
                                    -F 'deduplication_on_engagement=true'
\\
                                    -F 'api_scan_configuration=${idScan}'
                            """
                            env.stageResultsonar = true
                        }
                    }
                    catch (Exception e) {
                        unstable("${STAGE_NAME} failed!")
                        currentBuild.result = 'SUCCESS'

                        env.stageResultsonar = false
                    }
                }
            }
        }
```

Gambar 5.17 Stage Scanning Sonarqube.

Pada stage ‘*Scanning* with Sonarqube’, *pipeline* menjalankan proses analisis kode menggunakan SonarQube untuk mengevaluasi kualitas dan kerentanan kode sumber. Proses ini hanya dijalankan jika *pipeline* tidak dalam mode *production* (!gitRelease). *Secrets* diambil lebih dulu dari Vault menggunakan fungsi loadSecretsFromVault(). Di dalam direktori *sourcecode*, perintah sonar-scanner dijalankan dengan konfigurasi token autentikasi (SONARQUBE_TOKEN) dan URL server SonarQube (SONARQUBE_HOST_URL) untuk mengunggah hasil analisis. Setelah analisis selesai, hasilnya diimpor ke DefectDojo menggunakan API *reimport-scan* dengan tipe *scan* SonarQube API Import, sehingga hasil pemindaian dari SonarQube dapat dikelola di DefectDojo, termasuk grouping findings, auto-closing temuan lama, dan integrasi dengan Jira jika diaktifkan. Jika proses berhasil, variabel env.stageResultsonar diset ke true, jika terjadi *error*, *exception* ditangkap, stage ditandai *unstable*, *pipeline* tetap dianggap berhasil (SUCCESS), dan env.stageResultsonar di-*set* ke false untuk menandai kegagalan proses ini tanpa menghentikan *pipeline* secara keseluruhan.

Pada stage ‘Change PR Status’, *pipeline* secara otomatis menentukan apakah *pull request* (PR) di Bitbucket harus di-*approve* atau di-*decline* berdasarkan hasil dari tiga proses sebelumnya yaitu Snyk Scan, SonarQube Scan, dan Unit Test. Stage ini hanya dijalankan jika *pipeline* berasal dari sebuah PR (env.gitSourcePr != ""). *Secrets* diambil terlebih dahulu dari Vault menggunakan fungsi loadSecretsFromVault(). Jika ketiga proses tersebut berhasil (env.stageResultsnyk, env.stageResultsonar, dan env.stageResultunit bernilai 'true'), maka *pipeline* mengirimkan permintaan *approve* ke API Bitbucket untuk menyetujui PR secara otomatis. Sebaliknya, jika

ada proses yang gagal, *pipeline* akan mengirimkan permintaan *decline* untuk menolak PR dan menandai *pipeline* sebagai gagal menggunakan perintah error("PR Failed"). Dengan cara ini, *pipeline* dapat mengotomatiskan proses validasi PR berdasarkan kualitas kode dan hasil pengujian sebelum kode digabungkan ke branch utama.

```
stage('Change PR Status'){
            when {
                expression {env.gitSourcePr != ""}
            }
            steps {
                script {
                    loadSecretsFromVault()
                    if((env.stageResultsnyk == 'true' &&
env.stageResultsonar == 'true' && env.stageResultunit =='true') &&
(env.gitSourcePr != "")){
                        echo "PR Passed, approved !!"
                        def approvepr = "curl --request POST \
                        --url
'https://api.bitbucket.org/2.0/repositories/dcoedev/${repoName}/pullreques
ts/$env.gitPrId/approve' \
                        --header 'Authorization: Bearer
${env.accessToken}' \
                        --header 'Accept: application/json'"
                        sh """
                            set +x
                            ${approvepr}
                            set -x
                        """
                    }
                    else{
                        echo "PR Failed, decline !!"
                        def declinepr = "curl --request POST \
                        --url
'https://api.bitbucket.org/2.0/repositories/dcoedev/${repoName}/pullreques
ts/$env.gitPrId/decline' \
                        --header 'Authorization: Bearer
${env.accessToken}' \
                        --header 'Accept: application/json'"
                        sh """
                            set +x
                            ${declinepr}
                            set -x
                        """
                        error("PR Failed")
                    }
                }
            }
        }
```

Gambar 5.18 Stage Change PR Status.

```
stage('Building Docker Image') {
            when {
                expression {
                    !env.gitSourcePr && !gitRelease
                }
            }
            steps {
                script {
                    loadSecretsFromVault()
                    dir('sourcecode'){
                        sh "rm -rf Jenkinsfile Dockerfile $appsName-
deployment.yaml $appsName-service.yaml"
                        sh "sed -i 's%{appsname}%$appsName%g' run.sh"
                        sh "rm -rf .husky"
                    }
                    sh "cp -rf $appsName/* ."
                    sh """
                        set +x
                        echo "\$DOCKER_PASSWORD" | docker login -u
"\$DOCKER_USERNAME" --password-stdin
                        set -x
                        docker build -t ${registry}:${BUILD_NUMBER} .
                    """
                }
            }
        }
```

Gambar 5.19 Stage Building Docker Image.

Pada stage 'Building Docker Image', *pipeline* melakukan proses pembuatan *image* Docker dari *Source code* aplikasi. Stage ini hanya dijalankan jika *pipeline* bukan berasal dari *pull request* (!env.gitSourcePr) dan bukan untuk rilis *production* (!gitRelease). *Secrets* diambil terlebih dahulu dari Vault menggunakan fungsi loadSecretsFromVault(). Selanjutnya, *pipeline* masuk ke dalam *folder sourcecode* untuk membersihkan *file* yang tidak diperlukan dalam *image* Docker seperti Jenkinsfile, Dockerfile lama, dan *file deployment* YAML. *Script* run.sh diubah dengan mengganti placeholder {appsname} menjadi nama aplikasi sebenarnya ($appsName). *Folder* .husky yang berisi konfigurasi Git hook juga dihapus. Setelah persiapan selesai, *file* aplikasi disalin dari folder aplikasi ke direktori kerja saat ini. *Pipeline* kemudian melakukan

proses login ke Docker *registry* secara aman menggunakan kredensial dari *environment* variabel (DOCKER_USERNAME dan DOCKER_PASSWOR-D), dan terakhir membangun *image* Docker dengan tag versi berdasarkan nomor *build* (${*BUILD*_NUMBER}). Tahapan ini memastikan *image* Docker yang dihasilkan sudah bersih dan siap untuk tahap *deployment* berikutnya.

```
stage('Scanning with Docker Scout') {
            when {
                expression {
                    !env.gitSourcePr && !gitRelease
                }
            }
            steps {
                script {
                    loadSecretsFromVault()
                    try {
                        dir('sourcecode') {
                            def exitCode = sh(
                                script: "/usr/local/bin/docker-scout cves
${registry}:$env.BUILD_NUMBER --exit-code " +
                                        "--only-severity
critical,high,medium,low,negligible,unknown " +
                                        "--only-vuln-packages " +
                                        "--only-fixed ",
                                returnStatus: true
                            )

                            if (exitCode != 0) {
                                env.stageResultscout = 'true'
                                unstable("Vulnerability scan detected
issues with exit code: ${exitCode}")
                            } else {
                                env.stageResultscout = 'false'
                            }

                            // Export vulnerabilities to a SARIF JSON file
                            sh(
                                script: "/usr/local/bin/docker-scout cves
${registry}:$env.BUILD_NUMBER " +
                                        "--format sarif " +
                                        "--output alpine.sarif.json",
                                returnStatus: false
                            )
                        }
                    } catch (Exception e) {
                        unstable("${STAGE_NAME} failed due to an
exception!")
                        currentBuild.result = 'UNSTABLE'
                    }
                        sh 'cp -rf ~/k8s/defectdojo/* .'
                        sh 'cp sourcecode/alpine.sarif.json .'
                        sh "sed -i 's/{productName}/adiraku/g'
importscout.sh"
                        sh "sed -i 's/scout-{appsName}/scout-$appsName/g'
importscout.sh"
                        sh 'sh importscout.sh'
                }
            }
        }
```

Gambar 5.20 Stage Scanning Docker Scout.

Pada stage ‘Scanning with Docker Scout’, *pipeline* menjalankan proses pemeriksaan keamanan terhadap *image* Docker yang telah dibangun menggunakan alat Docker Scout. Stage ini hanya dijalankan jika *pipeline* bukan dari *pull request* (!env.gitSourcePr) dan bukan untuk rilis *production* (!gitRelease). Pertama, *secrets* diambil dari Vault, kemudian *pipeline* masuk ke *folder sourcecode* untuk menjalankan perintah docker-scout cves guna memindai kerentanan (CVEs) pada *image* berdasarkan tag versi *build*. Hanya paket rentan yang memiliki perbaikan (*fixed*) dan memiliki tingkat keparahan dari *negligible* hingga *critical* yang dipindai. Jika hasil pemindaian menghasilkan kode keluar tidak nol (menandakan ada kerentanan), maka variabel stageResultscout di-*set* ke 'true' dan *pipeline* ditandai sebagai *unstable*. Selanjutnya, hasil pemindaian juga diekspor ke format SARIF (alpine.sarif.json) untuk dianalisis lebih lanjut. Setelah itu, *pipeline* menyalin *file* konfigurasi dari direktori ~/k8s/defectdojo/, menyesuaikan placeholder seperti {productName} dan scout-{appsName} dalam *script* importscout.sh, lalu mengeksekusi *script* tersebut untuk mengimpor hasil *scan* ke sistem DefectDojo sebagai pusat pengelolaan kerentanan. Tahapan ini memastikan *image* Docker yang dihasilkan bebas dari kerentanan yang diketahui, atau setidaknya didokumentasikan dengan baik.

Pada stage ‘Deploying Image to Registry’, *pipeline* melakukan proses *push image* Docker yang telah dibangun ke Docker *Registry*. Tahapan ini hanya dijalankan apabila *pipeline* bukan hasil dari *pull request* (!env.gitSourcePr) dan bukan untuk kebutuhan rilis *production* (!gitRelease). *Secrets* terlebih dahulu dimuat dari Vault untuk keperluan otentikasi jika diperlukan. Setelah itu, perintah docker push dijalankan untuk mengunggah *image* dengan tag sesuai variabel ${registry}:${*BUILD*_NUMBER} ke Docker Registry

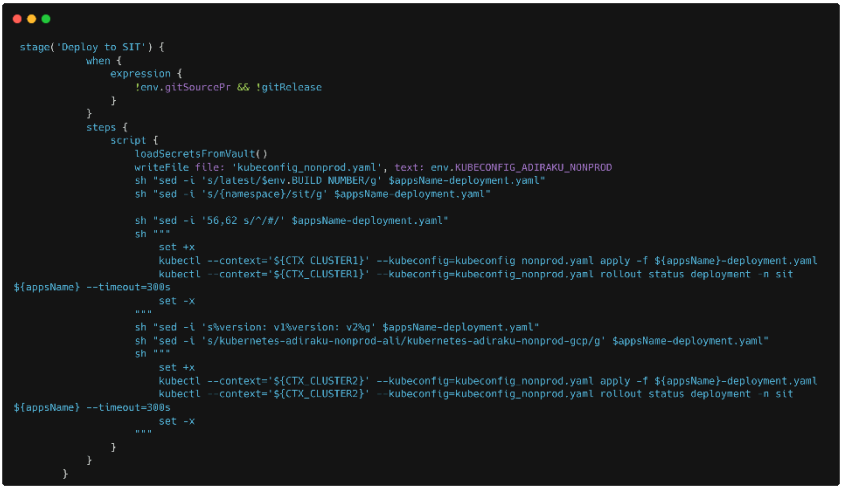
yang telah ditentukan. Tujuan dari tahap ini adalah untuk memastikan bahwa *image* hasil *build* tersedia di *registry*, sehingga dapat digunakan pada tahap *deployment* selanjutnya di lingkungan *non-production*.

```
stage('Deploying Image to Registry') {
            when {
                expression {
                    !env.gitSourcePr && !gitRelease
                }
            }
            steps {
                script {
                    loadSecretsFromVault()
                    sh """
                        docker push ${registry}:${BUILD_NUMBER}
                    """
                }
            }
        }
```

Gambar 5.21 Stage Deploying Image To Registry.

Pada stage 'Deploy to SIT', *pipeline* melakukan proses *deployment* aplikasi ke lingkungan SIT (*System Integration Testing*) pada dua klaster Kubernetes non-produksi. Tahap ini hanya dijalankan jika *pipeline* bukan hasil dari *pull request* (!env.gitSourcePr) dan bukan untuk kebutuhan rilis *production* (!gitRelease). Pertama, *secrets* diambil dari Vault untuk kebutuhan otentikasi, lalu *file* konfigurasi Kubernetes (kubeconfig_nonprod.yaml) ditulis menggunakan variabel *environment*. *Deployment file* ($appsName-*deployment*.yaml) dimodifikasi agar sesuai dengan *build* saat ini, misalnya dengan mengganti *tag image* Docker dari *latest* menjadi nomor *build* yang sedang berjalan ($env.*BUILD*_NUMBER) dan *namespace* diubah ke sit. Baris tertentu dalam YAML juga dikomentari untuk kebutuhan spesifik

klaster pertama. Setelah itu, *deployment* dilakukan ke klaster Alicloud (CTX_CLUSTER1), dilanjutkan dengan pengecekan status rollout untuk memastikan *deployment* berhasil. Kemudian *file* YAML diubah lagi untuk kebutuhan klaster GCP (CTX_CLUSTER2), seperti mengubah versi dan *context* klaster, lalu di-*deploy* ke klaster kedua dengan proses rollout check serupa. Tahap ini memastikan bahwa aplikasi telah berhasil di-*deploy* dan berjalan baik di kedua klaster SIT.



```
stage('Deploy to SIT') {
            when {
                expression {
                    !env.gitSourcePr && !gitRelease
                }
            }
            steps {
                script {
                    loadSecretsFromVault()
                    writeFile file: 'kubeconfig_nonprod.yaml', text: env.KUBECONFIG_ADIRAKU_NONPROD
                    sh "sed -i 's/latest/$env.BUILD_NUMBER/g' $appsName-deployment.yaml"
                    sh "sed -i 's/{namespace}/sit/g' $appsName-deployment.yaml"

                    sh "sed -i '56,62 s/^/#/' $appsName-deployment.yaml"
                    sh """
                        set +x
                        kubectl --context='${CTX_CLUSTER1}' --kubeconfig=kubeconfig_nonprod.yaml apply -f ${appsName}-deployment.yaml
                        kubectl --context='${CTX_CLUSTER1}' --kubeconfig=kubeconfig_nonprod.yaml rollout status deployment -n sit
${appsName} --timeout=300s
                        set -x
                    """
                    sh "sed -i 's%version: v1%version: v2%g' $appsName-deployment.yaml"
                    sh "sed -i 's/kubernetes-adiraku-nonprod-all/kubernetes-adiraku-nonprod-gcp/g' $appsName-deployment.yaml"
                    sh """
                        set +x
                        kubectl --context='${CTX_CLUSTER2}' --kubeconfig=kubeconfig_nonprod.yaml apply -f ${appsName}-deployment.yaml
                        kubectl --context='${CTX_CLUSTER2}' --kubeconfig=kubeconfig_nonprod.yaml rollout status deployment -n sit
${appsName} --timeout=300s
                        set -x
                    """
                }
            }
        }
```

Gambar 5.22 Stage Deploy To SIT.

5.2. Implementasi Sistem untuk Integrasi Vault

5.2.1. Vault Injector

Gambar 5.23 Perintah Install Vault Injector.

Pada tahap Integrasi Vault, perintah Helm digunakan untuk memasang komponen Vault Injector di Kubernetes klaster. Vault Injector ini bertugas menyuntikkan *secrets* Vault ke dalam *pod* aplikasi secara otomatis melalui mekanisme *mutating admission webhook*. Perintah helm install ini memasang *chart* vault-injector dari repositori resmi HashiCorp ke dalam *namespace* bernama vault, sekaligus membuat *namespace* tersebut jika belum ada (--create-namespace). Opsi konfigurasi yang diaktifkan adalah injector.enabled=true, yang berarti hanya komponen *injector* yang dijalankan tanpa mengaktifkan Vault server (server.enabled=false) maupun CSI driver (csi.enabled=false) karena asumsi Vault *server* sudah dikelola secara eksternal. Alamat *server* Vault eksternal ditentukan lewat parameter global.externalVaultAddr yang mengarah ke URL https://vault-nonprod.adira.one, yaitu *endpoint* Vault non-produksi yang digunakan untuk pengambilan *secrets*. Tahap ini memastikan aplikasi di klaster Kubernetes dapat melakukan *auto-injection secrets* dari Vault tanpa harus menyimpan *secrets* secara *hardcoded* di *manifest* atau *environment* variabel aplikasi.

5.2.2. Service account, RBAC dan SECRET

```
kubectl create sa vault-auth -n kube-system
```

Gambar 5.24 Perintah Create Service Account.

Perintah ini membuat ServiceAccount bernama vault-auth di *namespace* sit, yang akan digunakan oleh Vault Injector untuk mengidentifikasi *pod* dan mengizinkan akses ke *secrets* Vault melalui mekanisme Kubernetes Auth Method.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: vault-auth-clusterrolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:auth-delegator
subjects:
- kind: ServiceAccount
  name: vault-auth
  namespace: kube-system
```

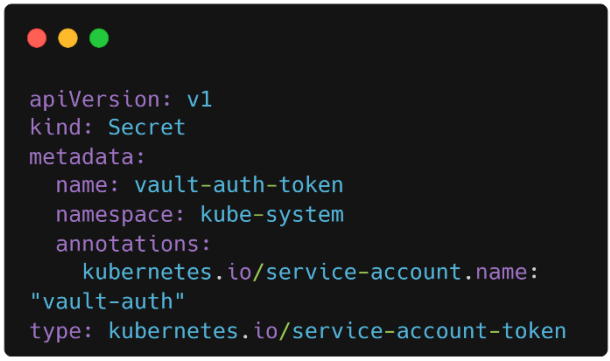
Gambar 5.25 Konfigurasi RBAC.

YAML di atas adalah konfigurasi ClusterRole Binding yang bernama vault-auth-clusterrolebinding, berfungsi untuk memberikan *role* system:auth-delegator kepada ServiceAccount bernama vault-auth di *namespace* kube-system. *Role* ini mengizinkan ServiceAccount tersebut untuk melakukan delegasi autentikasi, yang dibutuhkan agar Vault Injector dapat

memverifikasi identitas *pod* yang menjalankan aplikasi melalui Kubernetes Auth Method.

Setelah membuat *file* YAML ini, jalankan perintah berikut untuk menerapkan konfigurasi RBAC tersebut ke klaster:

Gambar 5.26 Perintah Apply RBAC.



Gambar 5.27 Konfigurasi Secret.

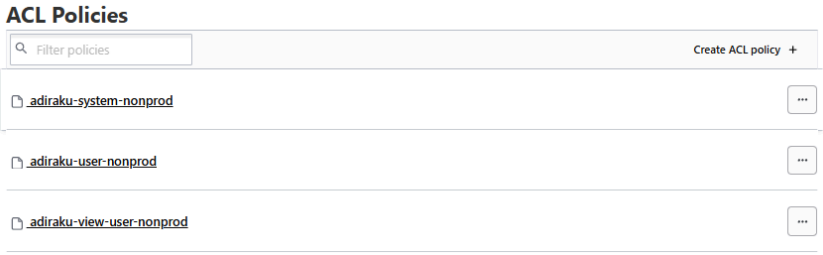
YAML di atas digunakan untuk membuat *Secret* bertipe kubernetes.io/service-account-token dengan nama vault-auth-token di *namespace* kube-system. *Secret* ini secara otomatis akan berisi token dari ServiceAccount bernama vault-auth karena adanya anotasi kubernetes.io/service-account.name. Token ini nantinya dipakai oleh Vault untuk proses autentikasi *pod* yang menggunakan Kubernetes Auth Method.

Setelah membuat *file* YAML ini, jalankan perintah berikut untuk menerapkan konfigurasi *Secret* ke klaster:

```
kubectl apply -f SECRET-SA.yaml
```

Gambar 5.28 Perintah Apply Secret.

5.2.3. Setup di Vault UI



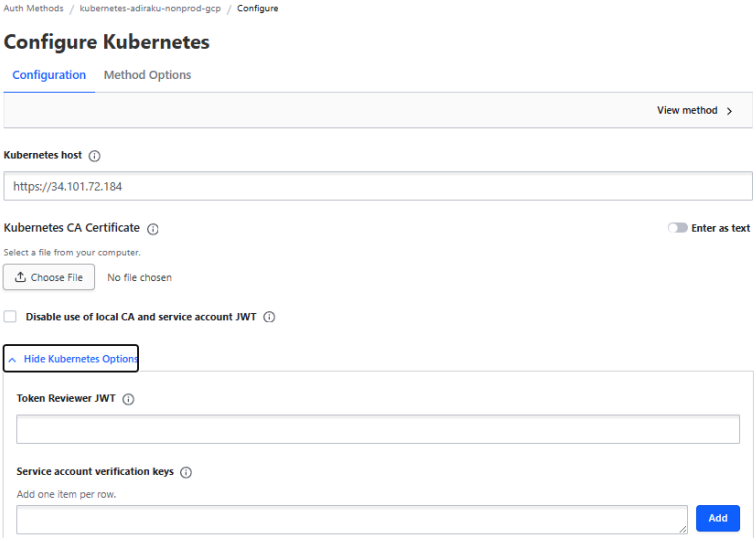
Gambar 5.29 Policies di Vault UI.

Pada konfigurasi Vault ini, telah dibuat beberapa *policy* yaitu adiraku-system-nonprod, adiraku-user-nonprod, dan adiraku-view-user-nonprod yang masing-masing mengatur hak akses ke *path* ADIRAKU di Vault. Policy adiraku-system-nonprod dan adiraku-view-user-nonprod hanya diberikan kemampuan *read* dan *list*, sedangkan adiraku-user-nonprod memiliki hak akses penuh (*create, read, update, delete, list*) serta akses khusus untuk mengubah *password user* melalui *path* tertentu. Setelah *policy* selesai diatur melalui Vault UI, lalu *user* ditambahkan menggunakan perintah:

```
vault write auth/userpass/users/<username> password=<password> policies=<policy1>,<policy2>
```

Gambar 5.30 Perintah Add User Vault.

dengan penentuan *policy* sesuai hasil koordinasi dengan tim DevOps agar akses yang diberikan ke masing-masing *user* sesuai kebutuhan fungsionalitas aplikasi atau proses yang dijalankan.



Gambar 5.31 Konfigurasi Kubernetes di Vault UI.

Setelah pembuatan ServiceAccount dan *Secret* di Kubernetes, dilakukan perintah kubectl get secret untuk mengambil JWT Token dan CA Certificate dari *secret* vault-auth-token yang ada di *namespace* adiraku-*production*, kemudian disimpan ke dalam *file* sa_token_adiraku-sit.txt untuk token dan ca_cert_adiraku-sit.pem untuk sertifikat CA. *File* ini digunakan untuk konfigurasi Authentication Method Kubernetes di Vault, yang dibuat masing-masing untuk klaster Ali dan GCP. Pada konfigurasi tersebut, dimasukkan informasi berupa alamat *server* (IP klaster yang diambil dari kubeconfig), CA Certificate, serta JWT Token hasil dari *command* sebelumnya.

5.2.4. Import Secret dari Consul ke Vault

```powershell
# === Config ===
$CONSUL_ADDR = "http://localhost:61577"
$VAULT_ADDR = "http://localhost:53748"
$VAULT_TOKEN = "<vault-token>"
$PROJECT = "ADIRAKU"

# Daftar prefix yang ingin diproses
$CONSUL_PREFIXES = @(
    "prefix"
)

$VAULT_PREFIX = "PRODUCTION"

# === Start ===
foreach ($CONSUL_PREFIX in $CONSUL_PREFIXES) {
    Write-Host "Fetching keys from Consul with prefix: $CONSUL_PREFIX..."
    $response = Invoke-RestMethod -Uri "$CONSUL_ADDR/v1/kv/$CONSUL_PREFIX/?recurse=true" -
Method Get -ErrorAction SilentlyContinue

    if (-not $response) {
        Write-Host "No keys found for prefix: $CONSUL_PREFIX"
        continue
    }

    $payloadData = @{}

    foreach ($item in $response) {
        $key = $item.Key
        Write-Host "Processing key: $key"

        # Strip prefix
        $strippedKey = $key.Substring($CONSUL_PREFIX.Length + 1)

        # Get value
        $valueResponse = Invoke-RestMethod -Uri "$CONSUL_ADDR/v1/kv/$key" -Method Get
        $base64Value = $valueResponse[0].Value
        $decodedBytes = [System.Convert]::FromBase64String($base64Value)
        $decodedValue = [System.Text.Encoding]::UTF8.GetString($decodedBytes)

        # Optional: Parse as string literal for JSON
        $payloadData[$strippedKey] = $decodedValue
    }

    $jsonPayload = @{ data = $payloadData } | ConvertTo-Json -Depth 5

    # Capitalize prefix
    $splitPrefix = $CONSUL_PREFIX.Split("/")[0]
    $prefixParts = $splitPrefix.Split("-")
    $capitalizedPrefix = ($prefixParts[0].ToUpper()) + "/" + $CONSUL_PREFIX.Split("/")[1]

    # Vault path
    $vaultPath = "$PROJECT/data/$VAULT_PREFIX/$capitalizedPrefix"
    Write-Host "Writing to Vault path: $vaultPath"
    Write-Host "Payload: $jsonPayload"

    $vaultResponse = Invoke-RestMethod -Uri "$VAULT_ADDR/v1/$vaultPath" `
        -Method Post `
        -Headers @{ "X-Vault-Token" = $VAULT_TOKEN } `
        -Body $jsonPayload `
        -ContentType "application/json"

    Write-Host "Vault response: $vaultResponse"
    Write-Host "Migration for prefix $CONSUL_PREFIX completed!
}
```

Gambar 5.32 Script untuk Import Value Secret.

Setelah proses konfigurasi *authentication method* dan pembuatan *role* di Vault selesai, dilakukan proses migrasi *secret* dari Consul ke Vault menggunakan *script* berbasis PowerShell. *Script* ini membaca seluruh *key-value* yang tersimpan di Consul berdasarkan *prefix* tertentu yang telah didefinisikan. Setiap *key* yang ditemukan diproses untuk menghapus *prefix path* agar diperoleh nama *key* yang bersih (murni) untuk disimpan

di Vault. Nilai dari setiap *key* diambil dalam format base64, kemudian didekodekan menjadi *string* UTF-8 sehingga dapat digunakan sebagai *value* dalam format JSON. Semua *key* dan *value* yang telah diproses kemudian dikemas ke dalam struktur JSON *payload* sesuai dengan format penyimpanan Vault. *Path* tujuan penulisan di Vault ditentukan berdasarkan variabel $PROJECT, $VAULT_*PREFIX*, dan *prefix* yang sudah dikapitalisasi sebagian. *Payload* tersebut selanjutnya dikirim ke Vault melalui API HTTP menggunakan metode POST dengan menyertakan Vault token yang valid untuk otorisasi. Proses ini diulang untuk semua *prefix* yang terdaftar sehingga seluruh *secrets* dari Consul berhasil dimigrasikan ke Vault dengan struktur dan format yang sesuai kebutuhan aplikasi serta standar penyimpanan yang lebih aman dan terpusat.

5.3. Implementasi Sistem untuk Testing Aplikasi

Setelah proses *deployment* ke lingkungan SIT selesai, dilakukan verifikasi sederhana untuk memastikan aplikasi dapat berjalan dengan baik. Verifikasi ini meliputi *port-forwarding service* dari Kubernetes klaster ke lokal untuk memastikan *service* dapat diakses sesuai *endpoint* yang diharapkan. Selain itu, dilakukan pengecekan proses aplikasi di dalam *pod* menggunakan perintah ps -ef untuk memastikan aplikasi berjalan menggunakan Vault Injector sesuai konfigurasi *secret management* yang telah diimplementasikan. Pengujian fungsional lebih lanjut menggunakan *tools* seperti JMeter tidak dilakukan karena *microservice* ini bersifat internal dan tidak membutuhkan *load test* pada tahap ini. Validasi tambahan dilakukan oleh tim terkait di luar cakupan tugas ini.

BAB VI
PENGUJIAN DAN EVALUASI

Bab ini menjelaskan proses pengujian dan evaluasi terhadap implementasi *pipeline* CI/CD yang dibangun untuk aplikasi Adiraku selama pelaksanaan kerja praktik. Pengujian dilakukan untuk memastikan setiap tahapan *pipeline*. Mulai dari *build image*, *scanning vulnerability*, *push image* ke *container registry*, hingga *deployment* ke Kubernetes klaster berjalan sesuai rancangan dan memenuhi kebutuhan sistem yang telah ditetapkan.

6.1. Tujuan Pengujian

Pengujian dilakukan untuk memastikan bahwa:

1. Proses *build* Docker Image berhasil dilakukan tanpa *error*.
2. Proses *vulnerability scanning* menggunakan Docker Scout berjalan dan dapat mendeteksi potensi celah keamanan.
3. Image berhasil di-*push* ke *registry*.
4. *Deployment* ke klaster Kubernetes (baik di AliCloud maupun GCP) berjalan tanpa *error* dan *service* dapat diakses.
5. Vault Injector berhasil mengelola *secrets* secara otomatis di dalam *pod* aplikasi.

6.2. Kriteria Pengujian

Penilaian pencapaian tujuan pengujian dilakukan dengan kriteria berikut:

1. *Build* Docker Image berhasil dan tidak menghasilkan *error*.
2. *Scanning vulnerability* selesai dan laporan CVE dapat dihasilkan.
3. Image ter-*push* ke *registry container* sesuai *tag*.

4. Deployment berhasil, *pod* dalam kondisi Running, dan *service* dapat diakses melalui *port-forwarding*.
5. Secret dari Vault berhasil di-*inject* ke *pod*.
6. Validasi *environment* dan *service readiness* melalui *port-forwarding* dan pengecekan proses aplikasi (ps -ef).

6.3. Skenario Pengujian

Pengujian dilakukan dengan simulasi penggunaan sistem sesuai kasus nyata:

1. Build Image
   Menjalankan *pipeline* untuk *build* Docker Image dan memastikan tidak ada *error* pada tahap *build*.
2. Vulnerability Scanning
   Menjalankan Docker Scout untuk melakukan *scanning* terhadap *image* yang telah dibangun dan mengecek hasil *scan* dalam format SARIF.
3. Push Image
   Memastikan *image* berhasil di-*push* ke *container registry* internal.
4. Deployment ke Klaster SIT
   Deploy image ke *environment* SIT di dua klaster (AliCloud dan GCP), verifikasi *deployment* berhasil melalui *command*:

```
kubectl get pods -n sit
kubectl rollout status deployment <app-name> -n sit
```

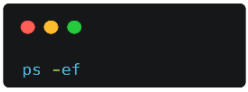
Gambar 6.1 Perintah Cek Status Pod.

5. Validasi Service
   Melakukan *port-forwarding service* untuk memastikan *service* dapat diakses:

Gambar 6.2 Perintah Port-forward Service.

Memastikan aplikasi berjalan dan dapat diakses via *browser*/postman.

6. Validasi Vault Injector
   Melakukan pengecekan proses di dalam *pod* untuk memastikan Vault Agent Injector berjalan:



Gambar 6.3 Perintah Cek Proses Pod.

6.4. Evaluasi Pengujian

Hasil pengujian dirangkum dalam tabel berikut:

Tabel 6.1 Hasil Evaluasi Pengujian.

Kriteria Pengujian	Hasil
Build Image	Berhasil
Vulnerability Scanning	Berhasil
Push ke Container Registry	Berhasil
Service Accessible (Port Forward)	Berhasil
Vault Secret Injection	Berhasil
Deployment ke SIT (Ali dan GCP)	Berhasil

[Halaman ini sengaja dikosongkan]

BAB VII
KESIMPULAN DAN SARAN

7.1. Kesimpulan

Berdasarkan pelaksanaan kerja praktik dan implementasi *pipeline* CI/CD pada aplikasi Adiraku, dapat disimpulkan hal-hal berikut:

1. *Pipeline* berhasil dibangun untuk proses *build*, *vulnerability scanning*, *push image* ke *registry*, serta *deployment* ke klaster Kubernetes di *environment* SIT (AliCloud dan GCP).
2. Proses pengelolaan *secret* berhasil diintegrasikan dengan HashiCorp Vault menggunakan Vault Agent Injector, sehingga pengelolaan *credential* menjadi lebih aman dan terpusat.
3. Pengujian fungsional dilakukan melalui *port-forwarding service* serta validasi *environment* pada *pod*, yang menunjukkan aplikasi berjalan normal sesuai harapan.
4. Konfigurasi Vault Authentication Method dan *Role* dilakukan di dua klaster (AliCloud dan GCP) untuk menjamin konsistensi *deployment* multi-klaster.

7.2. Saran

Untuk pengembangan lebih lanjut, terdapat beberapa saran yang dapat dipertimbangkan:

1. Melakukan pengujian performa menggunakan JMeter atau *tools* serupa untuk mengukur respons aplikasi setelah *deployment*, terutama untuk mengetahui waktu respons dan beban maksimum aplikasi.

2. Memperluas implementasi Vault Injector ke lingkungan *Production* dengan mempertimbangkan *audit log* Vault untuk memantau penggunaan *secrets*.
3. Menambahkan notifikasi otomatis (melalui Slack atau email) di *pipeline* CI/CD untuk memberitahu tim terkait hasil *deployment*, baik berhasil maupun gagal.
4. Melakukan dokumentasi teknis lebih lengkap mengenai *setup* Vault dan Kubernetes Authentication Method untuk memudahkan pengembangan di masa depan atau *onboarding* tim baru.

DAFTAR PUSTAKA

[1] Adira Finance, "Informasi Umum," Adira Finance. [Daring]. Tersedia: https://www.adira.co.id/informasi_umum. [Diakses : 16 Mei 2025].

[2] M. Waseem, P. Liang, dan M. Shahin, "A Systematic Mapping Study on *Microservice*s Architecture in DevOps," *Journal of Systems and Software*, vol. 170, Des 2020, doi: 10.1016/j.jss.2020.110798.

[3] P. Singh Yadav dan P. S. Yadav, "Improving DevOps Efficiency with Jenkins Shared Libraries and Templates," *European Journal of Advances in Engineering and Technology*, vol. 2021, no. 11, hlm. 116–120, doi: 10.5281/zenodo.13353843.

[4] R. A. Parama, H. Studiawan, R. J. Akbar, "Implementasi Continuous Integration dan Continuous Delivery Pada Aplikasi myITS Single Sign On," *Jurnal Teknik ITS*, vol. 11, No. 3, 2022, ISSN: 2337-3539.

[5] Atlassian, "Get started with Bitbucket," Bitbucket Guides. [Daring]. Tersedia: https://bitbucket.org/product/guides/getting-started/overview. [Diakses: 16 Mei 2025].

[6] Taiwo Joseph Akinbolaji, Godwin Nzeako, David Akokodaripon, dan Akorede Victor Aderoju, "Proactive *monitoring* and security in *cloud* infrastructure: leveraging *tools* like Prometheus, Grafana, and HashiCorp Vault for Robust DevOps Practices," *World Journal of Advanced Engineering Technology and Sciences*, vol. 13, no. 2, hlm. 074–089, Nov 2024, doi: 10.30574/wjaets.2024.13.2.0543.

[7] Telkom University, "Perbedaan UAT dan SIT," IT Telkom University. [Daring]. Tersedia: https://it.telkomuniversity.ac.id/perbedaan-uat-dan-sit/. [Diakses: 17 Mei 2025].

[8] K. Senjab, S. Abbas, N. Ahmed, dan A. ur R. Khan, "A survey of Kubernetes scheduling algorithms," 1 Desember 2023, *Springer Science and Business Media Deutschland GmbH*. doi: 10.1186/s13677-023-00471-1.

[9] A. Anupam, P. Gonchigar, dan S. Sharma, “Analysis of Open Source Node.js *Vulnerability* Scanners,” *International Research Journal of Engineering and Technology*, 2020, [Daring]. Tersedia pada: www.irjet.net.

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Nama	: Delai Resgista Setyawan
Tempat, Tanggal Lahir	: Bekasi, 27 September 2004
Jenis Kelamin	: Laki-laki
Telepon	: +6285773239752
Email	: delair27rs@gmail.com

AKADEMIS

Kuliah	: Departemen Teknik Informatika – FTEIC , ITS
Angkatan	: 2022
Semester	: 6 (Enam)