

KARYA ILMIAH

INTRODUCTION TO OPENSHIFT APPLICATIONS

Pengantar Konsep, Arsitektur, dan Siklus Deployment Aplikasi pada Red Hat OpenShift Container Platform 4.10

Disusun oleh

Muh Zakhy Zulfandi, CISSP.

Makassar

2022

KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa karena karya ilmiah berjudul “Introduction to OpenShift Applications” ini dapat diselesaikan. Tulisan ini disusun sebagai pengantar untuk memahami bagaimana aplikasi modern dibangun, dijalankan, dipublikasikan, dan dikelola pada Red Hat OpenShift Container Platform, khususnya dalam konteks teknologi yang digunakan pada tahun 2022.

Pembahasan dalam karya ilmiah ini sengaja disusun dari sudut pandang aplikasi. Materi dimulai dari konsep container dan Kubernetes, kemudian bergerak menuju Project, proses build, ImageStream, Deployment, Pod, Service, Route, konfigurasi, storage, observability, keamanan, hingga CI/CD dan GitOps. Urutan tersebut dipilih agar pembaca tidak hanya mengenal istilah, tetapi juga memahami hubungan antarobjek sebagai satu siklus kerja yang utuh.

Penulis menyadari bahwa OpenShift merupakan platform yang luas dan terus berkembang. Karena itu, ruang lingkup tulisan dibatasi pada konsep serta alur kerja yang paling relevan untuk pengembang pemula dan menengah. Contoh perintah, diagram, tabel, dan studi kasus yang disertakan diharapkan dapat membantu pembaca membangun gambaran teknis sebelum mempelajari administrasi cluster dan arsitektur yang lebih kompleks.

Penulis menyampaikan terima kasih kepada berbagai pihak yang telah menyediakan dokumentasi dan sumber pembelajaran terbuka, khususnya Red Hat dan komunitas Kubernetes. Dokumentasi resmi tersebut menjadi rujukan penting untuk menjaga ketepatan istilah dan penjelasan teknis di dalam karya ilmiah ini.

Akhirnya, penulis berharap karya ilmiah ini dapat menjadi bahan belajar yang berguna bagi mahasiswa, pengembang, maupun pembaca yang ingin mengenal OpenShift secara lebih terstruktur. Kritik dan saran tetap diperlukan agar pembahasan serupa dapat disusun dengan lebih baik pada kesempatan berikutnya.

Makassar, 2022

Muh Zakhy Zulfandi

DAFTAR ISI

KATA PENGANTAR **2**
DAFTAR ISI **3**
ABSTRAK **4**
ABSTRACT **5**
BAB I PENDAHULUAN **6**
1.2 Rumusan Masalah dan Arah Penulisan 7
BAB II LANDASAN TEORI **9**
2.2 Image dan Container sebagai Unit Aplikasi 10
2.3 Kubernetes sebagai Fondasi Orkestrasi 11
2.4 Red Hat OpenShift Container Platform 4.10 12
2.5 Arsitektur Dasar Cluster OpenShift 13
BAB III METODE PENULISAN **15**
BAB IV PEMBAHASAN **16**
4.2 Project, Namespace, dan Hak Akses 17
4.3 Membuat Aplikasi dari Source Code 18
4.4 Image Registry dan ImageStream 19
4.5 Deployment, ReplicaSet, dan Strategi Rollout 19
4.6 Pod Lifecycle dan Health Probe 20
4.7 Service, Route, dan Jalur Trafik 21
4.8 Routing, DNS, dan TLS 22
4.9 ConfigMap, Secret, dan Konfigurasi Lingkungan 23
4.10 Persistent Storage dan PersistentVolumeClaim 24
4.11 Resource Request, Limit, dan Scaling 25
4.12 Observability: Metrics, Log, dan Event 26
4.13 CI/CD sebagai Kelanjutan Lifecycle Aplikasi 27
4.14 OpenShift Pipelines dan Tekton 28
4.15 GitOps dan Sumber Kondisi yang Diinginkan 29
4.16 Operator dan Otomasi Operasional 30
4.17 Keamanan: RBAC, ServiceAccount, dan SCC 31
4.18 DevSecOps, Image, dan Supply Chain 31
4.19 Troubleshooting Aplikasi secara Sistematis 32
BAB V STUDI KASUS DEPLOYMENT APLIKASI SEDERHANA **34**
5.1 Contoh Resource dan Jalur Verifikasi 35
BAB VI ANALISIS DAN PEMBAHASAN LANJUT **36**
6.1 OpenShift dan Kubernetes: Persamaan dan Perbedaan Praktis 37
BAB VII PENUTUP **38**
7.1 Implikasi dan Saran Lanjutan 39
DAFTAR PUSTAKA **40**

ABSTRAK

OpenShift merupakan platform aplikasi berbasis Kubernetes yang mempertemukan kebutuhan pengembang dan operasi dalam satu lingkungan kerja. Tulisan ini membahas OpenShift dari sudut pandang aplikasi: bagaimana kode sumber dibangun menjadi image, bagaimana image dijalankan dalam pod, bagaimana layanan dibuat stabil melalui Service dan Route, serta bagaimana siklus tersebut dipantau dan diotomatisasi. Fokus pembahasan ditempatkan pada OpenShift Container Platform 4.10 karena versi ini berada pada periode 2022 dan menggunakan Kubernetes 1.23 dengan CRI-O sebagai container runtime [1]. Pilihan tersebut membuat pembahasan tetap selaras dengan tahun penerbitan karya ilmiah ini, bukan memakai perilaku produk yang baru muncul pada versi sesudahnya.

Metode yang digunakan adalah studi literatur deskriptif-teknis terhadap dokumentasi resmi Red Hat, dokumentasi Kubernetes, dan bahan pendukung ekosistem cloud-native. Materi kemudian disusun mengikuti perjalanan sebuah aplikasi. Urutan ini dipilih agar istilah seperti Project, BuildConfig, ImageStream, Deployment, Pod, Service, Route, ConfigMap, Secret, PersistentVolumeClaim, Pipeline, dan Operator tidak dipelajari sebagai daftar objek yang terpisah, tetapi sebagai bagian dari satu sistem. Untuk memperjelas hubungan antarobjek, karya ini dilengkapi tabel, contoh perintah, diagram arsitektur, dan flowchart siklus aplikasi.

Hasil kajian menunjukkan bahwa kekuatan OpenShift bukan sekadar kemampuan menjalankan container. Nilai tambahnya tampak pada konsistensi proses build dan deployment, pemisahan konfigurasi dari image, pengelolaan akses berbasis project dan role, routing aplikasi, monitoring, serta jalur CI/CD dan GitOps yang terintegrasi [2][3]. Bagi pengembang, pemahaman terhadap alur aplikasi lebih penting daripada menghafal perintah. Ketika hubungan antara source code, image, workload, jaringan, storage, dan observability sudah dipahami, proses troubleshooting serta pengembangan aplikasi menjadi lebih sistematis.

Kata kunci: OpenShift, Kubernetes, container, deployment, pod, service, route, image, CI/CD, GitOps.

ABSTRACT

OpenShift is a Kubernetes-based application platform that integrates build, deployment, networking, configuration, observability, and delivery. This paper discusses OpenShift Container Platform 4.10 from an application developer perspective, following the path from source code to image, Pod, Service, Route, monitoring, and automated delivery. A descriptive technical literature review is used, based mainly on Red Hat OpenShift 4.10 and Kubernetes documentation. The study shows that understanding the relationship among these resources gives developers a clearer model for deployment and troubleshooting. Keywords: OpenShift, Kubernetes, container, deployment, service, route, CI/CD, GitOps.

BAB I
PENDAHULUAN

Perkembangan aplikasi web dan layanan digital mengubah cara perangkat lunak dipasang serta dirawat. Model lama yang mengandalkan instalasi manual pada setiap server sering menimbulkan perbedaan konfigurasi antara lingkungan pengembangan, pengujian, dan produksi. Aplikasi dapat berjalan baik di komputer pengembang, tetapi gagal ketika dipindahkan karena versi runtime, library, variabel lingkungan, atau konfigurasi sistem operasi tidak sama. Container muncul sebagai salah satu jawaban terhadap persoalan tersebut karena aplikasi dan dependensinya dapat dikemas menjadi artefak yang relatif konsisten.

Container sendiri belum menyelesaikan seluruh kebutuhan operasional. Ketika satu aplikasi berkembang menjadi banyak layanan, jumlah container dapat bertambah cepat. Tim perlu memutuskan container dijalankan di node mana, bagaimana memulihkan proses yang gagal, bagaimana menambah replika ketika trafik meningkat, serta bagaimana mengatur komunikasi antarservice. Kubernetes menyediakan model deklaratif untuk kebutuhan itu. Pengguna menyatakan kondisi yang diinginkan, sedangkan control plane berupaya mempertahankan kondisi tersebut melalui scheduler dan controller.

OpenShift Container Platform membangun platform aplikasi di atas fondasi Kubernetes. Pada rilis 4.10, Red Hat menempatkan OpenShift sebagai platform hybrid cloud untuk menjalankan aplikasi baru maupun aplikasi yang telah ada, dengan Kubernetes 1.23 dan CRI-O sebagai basis runtime [1]. Di atas fondasi tersebut, OpenShift menghadirkan pengalaman pengembang melalui Developer Console, CLI oc, build, image management, routing, registry, keamanan, Operator, dan integrasi CI/CD. Dengan pendekatan ini, pengembang tidak hanya berurusan dengan tempat menjalankan container, tetapi memperoleh jalur kerja dari source code sampai aplikasi dapat digunakan.

Masalah yang sering muncul bagi pengguna baru adalah banyaknya objek yang harus dipahami dalam waktu singkat. Istilah Pod, Deployment, Service, Route, ImageStream, BuildConfig, Secret, dan PersistentVolumeClaim terdengar terpisah, padahal semuanya membentuk satu alur. Karya ilmiah ini berangkat dari masalah tersebut: menjelaskan hubungan antarobjek melalui sudut pandang aplikasi, sehingga konsep OpenShift dapat dipahami sebagai proses yang runtut dan bukan kumpulan istilah teknis.

1.2 Rumusan Masalah dan Arah Penulisan

Berdasarkan latar belakang tersebut, pembahasan diarahkan pada beberapa pertanyaan utama. Pertama, bagaimana posisi OpenShift terhadap Kubernetes dan mengapa keduanya tidak tepat dipandang sebagai dua teknologi yang saling menggantikan? Kedua, resource apa saja

yang paling penting ketika sebuah aplikasi dibuat, dibangun, dijalankan, dan dipublikasikan? Ketiga, bagaimana mekanisme konfigurasi, storage, scaling, observability, serta keamanan berhubungan dengan lifecycle aplikasi? Keempat, bagaimana OpenShift memfasilitasi otomasi delivery melalui build, pipeline, dan GitOps?

Tujuan penulisan adalah membentuk pemahaman teknis yang cukup untuk membaca kondisi aplikasi pada OpenShift. Setelah mempelajari tulisan ini, pembaca diharapkan dapat menjelaskan perjalanan aplikasi dari source code ke image, memahami perbedaan fungsi Deployment dan Pod, menjelaskan mengapa Service diperlukan meskipun setiap Pod memiliki alamat IP, serta mengetahui kapan Route digunakan. Pembaca juga diharapkan mampu menempatkan ConfigMap, Secret, PersistentVolumeClaim, health probe, resource request/limit, monitoring, dan pipeline pada posisi yang benar dalam siklus aplikasi.

Manfaat praktis karya ilmiah ini ialah menyediakan bahan pengantar yang dapat digunakan sebelum masuk ke administrasi cluster yang lebih dalam. Fokusnya bukan instalasi control plane, desain jaringan fisik, atau pengelolaan infrastruktur cloud. Fokusnya adalah aplikasi yang berjalan di atas cluster. Batasan ini sengaja dipilih karena pengembang umumnya berinteraksi lebih dulu dengan project, console, CLI, build, workload, route, log, dan pipeline sebelum mempelajari konfigurasi cluster secara menyeluruh.

Sistematika penulisan dibagi ke dalam tujuh bab. Bab I menjelaskan latar belakang, masalah, tujuan, dan batasan. Bab II membahas landasan teori mengenai container, Kubernetes, dan OpenShift. Bab III menjelaskan metode kajian. Bab IV menjadi bagian utama yang mengurai objek dan proses aplikasi secara teknis. Bab V menyajikan studi kasus deployment sederhana. Bab VI memberi analisis mengenai manfaat, keterbatasan, dan relevansi OpenShift bagi tim pengembang. Bab VII berisi kesimpulan serta saran. Daftar pustaka ditempatkan pada halaman tersendiri agar sumber rujukan mudah diperiksa.

BAB II
LANDASAN TEORI

Sebelum memahami OpenShift, perlu dibedakan tiga lapisan yang sering tercampur dalam percakapan sehari-hari: virtual machine, container, dan orchestrator. Virtual machine membawa sistem operasi tamu yang lengkap di atas hypervisor. Container bekerja lebih dekat dengan kernel host dan menggunakan mekanisme isolasi proses, filesystem, jaringan, serta resource. Orchestrator kemudian mengatur banyak container di banyak node. Perbedaan ini bukan sekadar istilah, karena setiap lapisan menentukan bagaimana aplikasi dikemas, dipindahkan, diskalakan, dan dipulihkan.

Container menjadi menarik bagi pengembang karena artefak aplikasi dapat dibentuk sebagai image. Image bersifat tidak berubah pada saat dijalankan; perubahan kondisi aplikasi sebaiknya disimpan pada storage atau layanan eksternal, bukan dengan mengubah isi container secara manual. Pola tersebut mendukung reproduktibilitas. Jika image yang sama dipakai pada lingkungan uji dan produksi, risiko perbedaan dependensi dapat ditekan. Dalam praktik DevOps, image juga berfungsi sebagai unit promosi: image yang telah melewati pengujian dapat diberi tag dan dipindahkan ke tahap rilis berikutnya.

Namun, sebuah image belum melakukan apa pun sampai dijalankan sebagai container. Pada skala kecil, container dapat dijalankan langsung dengan runtime. Pada skala organisasi, cara ini segera menimbulkan persoalan penjadwalan, ketersediaan, restart, discovery, konfigurasi, keamanan, dan observability. Kubernetes hadir untuk menangani persoalan itu melalui objek deklaratif dan control loop. OpenShift memanfaatkan model tersebut, lalu menambahkan layanan dan pengalaman yang lebih terintegrasi untuk kebutuhan pengembangan aplikasi enterprise.

Pemahaman tiga lapisan tersebut membantu menghindari kesalahan umum. OpenShift bukan pengganti image container, karena OpenShift tetap memerlukan image untuk workload. OpenShift juga bukan sekadar 'Kubernetes dengan tampilan web'. Platform ini menggunakan API Kubernetes tetapi menambahkan resource, kebijakan, Operator, image management, route, console, serta proses lifecycle yang khas. Dengan kata lain, container adalah bentuk paket aplikasi, Kubernetes adalah mekanisme orkestrasi, sedangkan OpenShift menyusun keduanya menjadi platform aplikasi yang lebih lengkap.

2.2 Image dan Container sebagai Unit Aplikasi

Image container dapat dipahami sebagai paket berlapis yang berisi filesystem aplikasi, runtime, library, dan metadata yang diperlukan saat proses dijalankan. Satu image dapat digunakan untuk membuat banyak container. Karena image sebaiknya bersifat immutable,

perubahan kode atau dependensi dilakukan dengan membangun image baru, bukan memodifikasi container yang sedang berjalan. Pola ini menjadikan versi aplikasi lebih mudah ditelusuri dan mengurangi konfigurasi manual yang sulit direproduksi.

Dalam OpenShift, image tidak hanya dianggap sebagai file yang berada di registry. Platform menyediakan ImageStream untuk melacak referensi image dan tag di dalam project. ImageStream dapat membantu memisahkan cara aplikasi merujuk image dari lokasi registry yang sebenarnya. Ketika sebuah tag diperbarui, perubahan tersebut dapat digunakan sebagai pemicu proses lain, misalnya rollout aplikasi. Dokumentasi OpenShift 4.10 menempatkan image dan build sebagai bagian penting dari alur pembuatan aplikasi [3].

Container yang dijalankan dari image memiliki lifecycle yang lebih pendek daripada server tradisional. Ia dapat dibuat, dihentikan, dan diganti kapan saja. Karena itu, aplikasi cloud-native tidak boleh bergantung pada identitas satu container tertentu. Data penting tidak seharusnya hanya tersimpan pada filesystem sementara container. Alamat jaringan juga tidak boleh dianggap permanen. Prinsip ini menjelaskan mengapa Kubernetes memiliki Pod, Service, PersistentVolume, ConfigMap, dan Secret sebagai resource yang terpisah.

Bagi pengembang, konsekuensi praktisnya adalah disiplin dalam memisahkan tiga jenis isi: kode dan dependensi masuk ke image, konfigurasi lingkungan masuk ke ConfigMap atau Secret, sedangkan data yang harus bertahan dipindahkan ke storage persisten atau layanan basis data. Pemisahan tersebut membuat image dapat dipakai ulang pada beberapa lingkungan. Image yang sama dapat dijalankan di development dan production dengan konfigurasi berbeda tanpa perlu dibangun ulang hanya karena alamat database, mode logging, atau kredensial berubah.

2.3 Kubernetes sebagai Fondasi Orkestrasi

Kubernetes menggunakan pendekatan deklaratif. Pengguna tidak menulis instruksi rinci tentang node mana yang harus dipilih setiap saat; pengguna mendefinisikan kondisi yang diinginkan melalui objek API. Controller kemudian membandingkan kondisi aktual dengan kondisi yang diinginkan dan melakukan rekonsiliasi. Jika sebuah Deployment meminta tiga replika dan hanya dua Pod yang aktif, controller berupaya membuat Pod tambahan. Pola control loop inilah yang membedakan orkestrasi dari sekadar menjalankan proses.

Pod merupakan unit komputasi terkecil yang dapat dibuat dan dikelola oleh Kubernetes [21]. Satu Pod dapat berisi satu atau lebih container yang berbagi konteks jaringan dan volume tertentu. Dalam aplikasi umum, satu container utama per Pod adalah pola yang sering digunakan, sementara container tambahan biasanya dipakai untuk fungsi yang sangat erat hubungannya. Pod bersifat ephemeral: ia dapat diganti ketika node gagal, ketika rollout dilakukan, atau ketika scaling terjadi. Oleh sebab itu, Pod bukan endpoint yang ideal untuk digunakan secara langsung oleh klien.

Deployment mengelola kondisi yang diinginkan bagi sekumpulan Pod. Service menyediakan identitas jaringan yang stabil untuk mengakses kumpulan Pod yang sesuai dengan selector. Kubernetes memisahkan identitas layanan dari identitas Pod karena Pod dapat muncul dan hilang sepanjang lifecycle aplikasi [22]. Pemisahan ini memungkinkan rollout atau scaling dilakukan tanpa meminta konsumen layanan mengubah alamat tujuan setiap kali Pod berubah.

OpenShift mempertahankan konsep-konsep tersebut. Pengembang yang sudah memahami resource Kubernetes akan mengenali Pod, Deployment, Service, ConfigMap, Secret, PersistentVolumeClaim, Job, dan berbagai API lain. Perbedaannya, OpenShift menyediakan layer tambahan seperti Project, Route, ImageStream, BuildConfig, Developer Console, Operator Lifecycle Manager, serta integrasi pipeline. Karena fondasinya sama, pemahaman Kubernetes tetap menjadi modal utama; kemampuan khas OpenShift kemudian dipelajari sebagai perluasan terhadap model yang sudah ada.

2.4 Red Hat OpenShift Container Platform 4.10

OpenShift Container Platform 4.10 dirilis dalam konteks perkembangan Kubernetes 1.23. Release notes Red Hat menyatakan bahwa rilis ini menggunakan Kubernetes 1.23 dengan CRI-O runtime, dan versi GA publiknya dimulai dari 4.10.3 [1]. Keterangan ini penting karena perilaku serta fitur OpenShift berubah dari waktu ke waktu. Menggunakan versi 4.10 sebagai dasar membuat pembahasan konsisten dengan tahun 2022 dan menghindari pencampuran fitur dari generasi yang lebih baru.

Dari sisi pengembang, OpenShift menyediakan dua antarmuka utama. Web console menawarkan Developer perspective yang menampilkan aplikasi dalam bentuk topology, menyediakan katalog, alur penambahan aplikasi, akses ke log, status build, route, scaling, serta detail resource. CLI oc memberi akses yang lebih dekat ke API dan cocok untuk otomasi maupun troubleshooting. Keduanya tidak menciptakan dua model yang berbeda; console dan CLI bekerja terhadap resource yang sama [2][3].

Platform juga menyediakan jalur pembuatan aplikasi yang beragam. Source code dari repository Git dapat dibangun melalui mekanisme build, image yang sudah tersedia dapat langsung dideploy, dan resource dapat dibuat dari manifest YAML atau template. Perintah oc new-app menganalisis input dan dapat membuat BuildConfig, Deployment, serta Service sesuai jenis sumber yang diberikan [3]. Fleksibilitas ini memungkinkan OpenShift digunakan oleh tim dengan pola kerja yang berbeda tanpa mengubah konsep dasar di belakangnya.

Dalam organisasi, OpenShift bukan hanya alat deployment. Project dan role membantu pemisahan akses, registry dan image stream membantu pengelolaan artefak, Route memberi jalur publikasi layanan, monitoring menyediakan visibilitas kondisi, sedangkan Pipelines dan GitOps menyediakan otomasi delivery [15]. Operator digunakan untuk mengemas pengetahuan

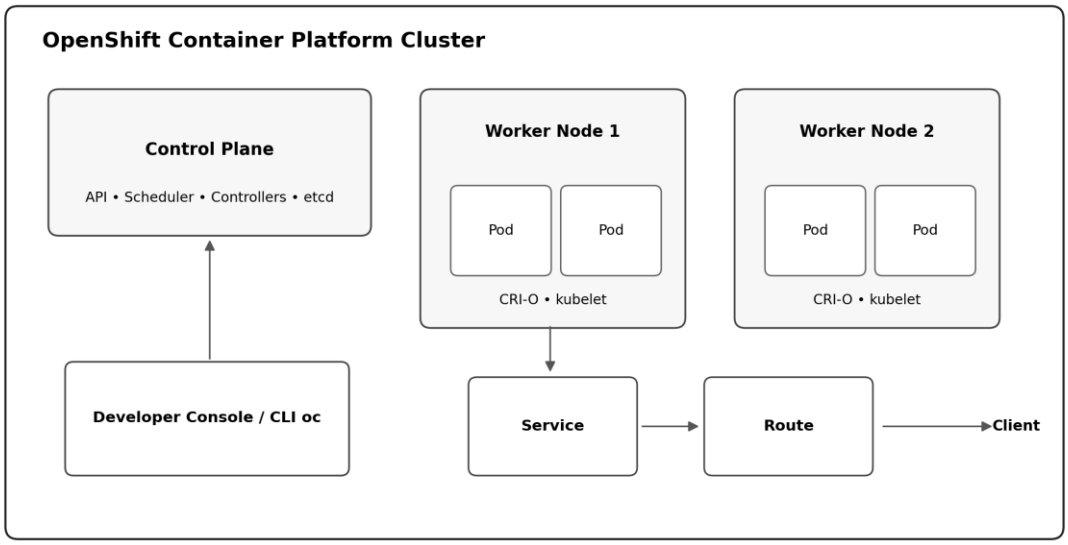
operasional layanan sehingga instalasi, konfigurasi, upgrade, dan rekonsiliasi dapat dilakukan secara lebih konsisten [18]. Kombinasi tersebut menjadikan OpenShift sebuah platform aplikasi, bukan hanya scheduler container.

2.5 Arsitektur Dasar Cluster OpenShift

Sebuah cluster OpenShift terdiri atas control plane dan compute atau worker nodes. Control plane menjalankan komponen yang menyimpan serta mengelola kondisi cluster, termasuk API server, scheduler, controller, dan etcd. Worker node menjalankan workload aplikasi melalui kubelet dan CRI-O. Pengembang biasanya tidak berinteraksi langsung dengan komponen control plane; interaksi dilakukan melalui API menggunakan web console, oc, atau sistem otomatis lain.

Ketika sebuah Deployment dibuat, scheduler menentukan node yang sesuai untuk Pod berdasarkan resource, constraint, dan kondisi cluster. Kubelet pada node menjalankan Pod melalui runtime. Service kemudian menyediakan endpoint internal yang stabil ke Pod, sementara Route pada OpenShift dapat digunakan untuk mengekspos Service ke luar cluster melalui router. Jalur ini memperlihatkan pemisahan tugas: control plane mengambil keputusan, node menjalankan workload, Service menjaga identitas layanan, dan Route mengatur akses eksternal.

Diagram berikut merangkum hubungan tersebut. Diagram sengaja disederhanakan untuk kebutuhan aplikasi; komponen jaringan, storage, Operator, dan layanan platform lain tidak seluruhnya ditampilkan. Tujuannya adalah menunjukkan posisi pengembang terhadap control plane dan jalur trafik dari Pod menuju pengguna.



Gambar 1. Gambaran sederhana posisi control plane, worker node, Pod, Service, dan Route.

BAB III
METODE PENULISAN

Karya ilmiah ini menggunakan metode studi literatur dengan pendekatan deskriptif-teknis. Sumber utama berasal dari dokumentasi Red Hat OpenShift Container Platform 4.10, terutama bagian release notes, building applications, networking, storage, monitoring, security, CI/CD, dan Operators. Dokumentasi Kubernetes digunakan untuk memperkuat konsep dasar Pod, Deployment, Service, konfigurasi, probe, dan autoscaling. Sumber dipilih berdasarkan kedekatannya dengan versi teknologi yang menjadi fokus tulisan, yaitu OpenShift 4.10 pada tahun 2022.

Tahap pertama adalah identifikasi objek yang ditemui pengembang ketika menjalankan aplikasi. Objek tersebut dikelompokkan menjadi enam area: ruang kerja dan akses, build dan image, workload, jaringan, konfigurasi dan data, serta observability dan delivery. Tahap kedua adalah menyusun objek berdasarkan urutan lifecycle aplikasi. Source code atau image ditempatkan sebagai input, dilanjutkan build, image management, deployment, Pod, Service, Route, monitoring, dan proses perubahan berikutnya.

Tahap ketiga adalah validasi istilah terhadap dokumentasi resmi. Contohnya, status OpenShift 4.10 sebagai platform berbasis Kubernetes 1.23 dan CRI-O diperiksa pada release notes [1]; alur pembuatan aplikasi menggunakan new-app diperiksa pada Building Applications [3]; solusi CI/CD yang meliputi Builds, Pipelines, dan GitOps diperiksa pada dokumentasi CI/CD [15]; sedangkan karakteristik Operator diperiksa pada dokumentasi Operators [18]. Dengan cara ini, penjelasan tidak bergantung pada kebiasaan satu lingkungan saja.

Batasan kajian ditetapkan agar pembahasan tetap fokus. Tulisan tidak membahas instalasi cluster secara rinci, konfigurasi jaringan fisik, desain multi-cluster, OpenShift Virtualization, atau strategi disaster recovery enterprise. Contoh perintah dan YAML bersifat ilustratif; implementasi produksi harus menyesuaikan kebijakan organisasi, versi patch, security policy, kapasitas cluster, serta integrasi yang digunakan. Analisis ditujukan untuk membentuk pemahaman aplikasi, bukan menggantikan runbook administrasi produksi.

BAB IV
PEMBAHASAN

Pada OpenShift, sebuah aplikasi tidak identik dengan satu file atau satu Pod. Aplikasi adalah himpunan resource yang saling berhubungan dan berada dalam suatu project. Satu komponen web, misalnya, dapat terdiri atas BuildConfig, ImageStream, Deployment, beberapa Pod, Service, Route, ConfigMap, Secret, dan PersistentVolumeClaim. Ketika aplikasi tumbuh menjadi beberapa komponen, setiap komponen dapat memiliki resource sendiri, tetapi tetap dibaca sebagai satu topology pada Developer Console.

Cara pandang berbasis resource memiliki keuntungan penting. Setiap bagian dapat berubah tanpa harus membongkar keseluruhan sistem. Pod dapat diganti pada saat rollout, sementara Service mempertahankan endpoint internal. Route dapat tetap menggunakan hostname yang sama walaupun Service mengarahkan trafik ke Pod versi baru. ConfigMap dapat diubah tanpa membangun ulang image. PersistentVolumeClaim dapat mempertahankan data ketika Pod diganti. Pemisahan ini menciptakan loose coupling antara aspek-aspek yang memang memiliki lifecycle berbeda.

OpenShift juga menyimpan state yang diinginkan sebagai objek API. Karena resource bersifat deklaratif, kondisi sistem dapat diperiksa melalui YAML atau CLI. Hal ini bermanfaat pada saat troubleshooting. Pengembang dapat memeriksa apakah masalah berada pada build, image, deployment, Pod, Service, atau Route, bukan hanya menyimpulkan bahwa 'aplikasi tidak bisa dibuka'. Pendekatan diagnosis berbasis resource membuat pencarian masalah lebih terarah.

Untuk memahami lifecycle secara utuh, bagian berikut membahas tiap lapisan mulai dari Project sampai CI/CD. Urutannya mengikuti pengalaman pengembang ketika membawa aplikasi ke cluster: memilih ruang kerja, memasukkan source atau image, menghasilkan workload, memberikan akses jaringan, menambahkan konfigurasi dan data, memantau kondisi, lalu mengotomatisasi perubahan berikutnya.

4.2 Project, Namespace, dan Hak Akses

Project pada OpenShift merupakan ruang kerja logis yang berhubungan erat dengan namespace Kubernetes. Melalui project, resource aplikasi dikelompokkan agar nama, quota, role, dan kebijakan dapat dikelola pada batas yang jelas. Tim dapat memilih satu project per aplikasi, per lingkungan, atau mengikuti pola lain yang sesuai kebutuhan organisasi. Pemisahan antara development, staging, dan production, misalnya, membantu mencegah perubahan eksperimen langsung menyentuh workload produksi.

Pengembang biasanya memulai pekerjaan dengan memilih project yang tepat. Melalui CLI, project baru dapat dibuat dengan oc new-project dan konteks aktif diperiksa dengan oc project.

Seluruh resource yang dibuat setelahnya masuk ke namespace tersebut kecuali perintah menyebut namespace lain. Kesalahan memilih project merupakan salah satu penyebab umum ketika resource seolah-olah hilang; sebenarnya resource berada pada namespace yang berbeda.

Project juga menjadi batas praktis untuk Role-Based Access Control. Seseorang dapat diberi hak melihat resource tanpa hak mengubah, atau diberi kemampuan mengelola resource di project tertentu tanpa menjadi administrator cluster. Pola least privilege penting karena cluster sering digunakan oleh banyak tim. ServiceAccount memberi identitas kepada workload ketika Pod perlu mengakses API atau layanan lain. Dengan memisahkan identitas manusia dan identitas aplikasi, kebijakan akses dapat disusun lebih terukur.

Contoh pola kerja sederhana adalah membuat project demo, menambahkan pengguna sebagai anggota, lalu menjalankan aplikasi di dalamnya. Perintah di bawah hanya menggambarkan alur dasar; kebijakan role di lingkungan produksi harus mengikuti standar organisasi.

```
oc new-project demo-app
oc project demo-app
oc get all
oc get rolebinding
```

4.3 Membuat Aplikasi dari Source Code

Salah satu jalur yang paling mudah dipahami adalah membuat aplikasi dari repository Git. OpenShift dapat menganalisis source, memilih builder yang sesuai, membuat BuildConfig, menghasilkan image, lalu membuat workload dan Service. Perintah oc new-app adalah contoh antarmuka yang menggabungkan beberapa langkah tersebut. Menurut dokumentasi 4.10, objek yang dibuat bergantung pada artefak input, misalnya source repository, image, atau template [3].

Ketika source code menjadi input, proses build perlu mengetahui cara mengubah kode menjadi aplikasi yang dapat dijalankan. Source-to-Image (S2I) menggunakan builder image yang sudah membawa runtime dan skrip build. Source code dimasukkan ke builder, dependensi diproses, kemudian dihasilkan image aplikasi. Pendekatan ini mengurangi kebutuhan menulis Dockerfile untuk kasus yang didukung builder dan membantu menjaga pola build lebih seragam antarproyek.

OpenShift tidak mewajibkan S2I. Tim dapat memakai Docker build strategy, custom build, binary build, atau pipeline lain sesuai kebutuhan. Prinsip yang tetap sama adalah hasil build berupa image yang dapat dipromosikan serta dijalankan berulang kali. Dengan memisahkan source dan image, proses release tidak perlu membangun ulang kode di setiap lingkungan. Image yang telah diuji dapat dipindahkan atau dirujuk oleh environment berikutnya.

Dalam praktik, developer perlu memastikan repository dapat diakses, branch atau context directory benar, dan builder cocok dengan versi runtime aplikasi. Build log harus diperiksa ketika proses gagal. Masalah yang sering terlihat ialah dependensi tidak tersedia, permission

salah, proxy tidak dikonfigurasi, atau builder tidak menemukan struktur source yang diharapkan. Menyelesaikan masalah pada tahap build lebih baik daripada membiarkan image yang tidak tervalidasi bergerak ke deployment.

4.4 Image Registry dan ImageStream

Registry menyimpan image container, sedangkan ImageStream menyediakan abstraksi dan pelacakan tag di dalam OpenShift. Perbedaan ini penting. Registry adalah tempat layer image disimpan; ImageStream adalah resource yang dapat merujuk image dari registry internal maupun eksternal dan mencatat perubahan tag. Dengan ImageStream, aplikasi dapat menggunakan nama logis yang lebih stabil tanpa selalu menuliskan alamat registry lengkap.

Tag seperti latest, stable, atau versi numerik sebaiknya memiliki makna yang jelas. Dalam proses rilis yang disiplin, tag yang mutable digunakan dengan hati-hati karena dapat menunjuk image berbeda dari waktu ke waktu. Untuk audit yang lebih kuat, digest image memberi identitas yang tidak berubah terhadap konten tertentu. Tim dapat menggunakan tag untuk kemudahan manusia dan digest untuk verifikasi teknis saat diperlukan.

Image change trigger dapat membuat deployment merespons perubahan ImageStream. Pola ini berguna ketika build menghasilkan image baru dan aplikasi harus melakukan rollout. Namun, otomatisasi semacam itu perlu dibedakan antara lingkungan. Development mungkin menerima setiap build, sedangkan production biasanya memerlukan persetujuan atau promosi versi tertentu. OpenShift menyediakan mekanisme teknis, tetapi desain governance tetap menjadi tanggung jawab tim.

Registry juga berkaitan erat dengan keamanan. Image sebaiknya berasal dari sumber yang dipercaya, diperbarui ketika base image memiliki perbaikan keamanan, dan dibangun dengan user non-root bila memungkinkan. Kredensial registry eksternal dapat disediakan melalui pull secret sesuai scope yang diperlukan. Praktik utama yang perlu dijaga ialah tidak menaruh credential registry langsung di source code atau Dockerfile.

4.5 Deployment, ReplicaSet, dan Strategi Rollout

Setelah image siap, aplikasi memerlukan resource workload. Deployment menyimpan template Pod, jumlah replika, selector, dan strategi pembaruan. Ketika spec berubah, Deployment membuat ReplicaSet baru dan secara bertahap mengganti Pod versi lama. Mekanisme ini memungkinkan perubahan dilakukan tanpa menghentikan seluruh aplikasi secara bersamaan, selama aplikasi dan resource pendukung dirancang untuk rollout bertahap.

Pada OpenShift 4.10, pengguna masih dapat menemui Deployment maupun DeploymentConfig. Keduanya sama-sama berhubungan dengan rollout, tetapi Deployment merupakan objek Kubernetes yang semakin umum dipakai. Ketika mempelajari aplikasi baru, pengembang perlu melihat kind resource terlebih dahulu agar tidak memakai perintah atau

asumsi yang salah. Dokumentasi versi yang digunakan harus menjadi rujukan karena detail fitur dan rekomendasi dapat berubah antarversi.

Strategi rolling update biasanya membuat Pod baru sambil mengurangi Pod lama. Parameter seperti maxSurge dan maxUnavailable mengatur seberapa agresif perubahan dilakukan. Aplikasi yang memiliki readiness probe membantu rollout karena trafik baru hanya diberikan kepada Pod yang sudah siap. Tanpa readiness probe, proses dapat dianggap tersedia hanya karena container berjalan, padahal aplikasi mungkin belum selesai memuat konfigurasi atau koneksi database.

Rollback dan riwayat rollout penting untuk mengurangi dampak kegagalan rilis. Tim sebaiknya tidak mengandalkan ingatan mengenai image sebelumnya; versioning dan manifest perlu dikelola dengan jelas. Deployment yang baik juga memisahkan image dari konfigurasi. Dengan cara itu, rollback image tidak otomatis menghapus konfigurasi lingkungan yang memang harus dipertahankan.

4.6 Pod Lifecycle dan Health Probe

Pod adalah tempat container benar-benar berjalan. Kubernetes memperlakukan Pod sebagai resource yang dapat diganti, bukan mesin yang harus dipertahankan selamanya. Jika node gagal, Pod dapat dibuat ulang pada node lain. Jika deployment berubah, Pod lama diganti oleh Pod baru. Karena itu, pengembang sebaiknya menghindari desain yang menyimpan state kritis hanya pada filesystem lokal Pod atau bergantung pada nama Pod tertentu.

Status Pod perlu dibaca bersama event dan kondisi container. Fase Running belum selalu berarti aplikasi siap melayani trafik. Container dapat hidup tetapi proses internal masih memuat cache, melakukan migration, atau menunggu dependency. Readiness probe menyatakan apakah Pod siap menerima trafik. Liveness probe membantu mendeteksi proses yang macet dan perlu direstart. Startup probe berguna bagi aplikasi yang membutuhkan waktu inisialisasi panjang sehingga liveness tidak mematikan proses terlalu awal [24].

Probe dapat menggunakan HTTP request, TCP socket, atau command. Endpoint health sebaiknya ringan dan tidak menimbulkan beban besar. Readiness juga perlu dirancang dengan hati-hati; jika ia bergantung pada layanan eksternal yang fluktuatif, seluruh replika dapat keluar dari endpoint secara bersamaan. Health check adalah kontrak antara aplikasi dan platform, bukan sekadar syarat konfigurasi.

Pod juga memiliki restart policy dan dapat menghasilkan CrashLoopBackOff ketika container terus gagal. Diagnosis dimulai dari log container, exit code, event, environment, mount, dan resource limit. Dalam banyak kasus, status Pod memberi petunjuk awal yang cukup untuk membedakan kesalahan aplikasi dari kesalahan jaringan atau storage.

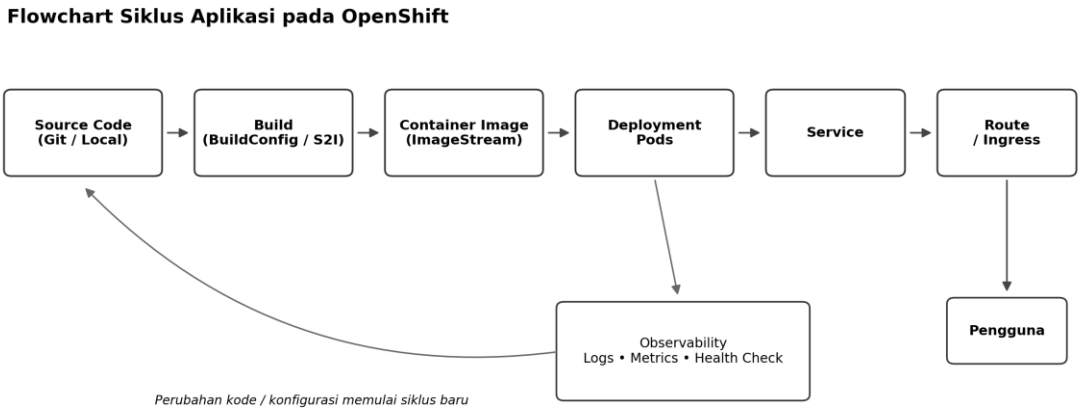
4.7 Service, Route, dan Jalur Trafik

Pod memperoleh alamat IP, tetapi alamat itu tidak dimaksudkan sebagai identitas stabil aplikasi. Deployment dapat mengganti Pod kapan saja sehingga daftar IP berubah. Service menyelesaikan masalah ini dengan menyediakan endpoint logis yang mengarah ke sekumpulan Pod berdasarkan selector. Konsumen internal cukup menggunakan nama Service dan port; mereka tidak perlu mengetahui Pod mana yang sedang aktif [22].

OpenShift menambahkan Route sebagai cara mengekspos Service ke luar cluster. Route menghubungkan hostname dengan Service melalui router. Pola yang umum adalah pengguna mengakses hostname, router menerima trafik, Route menentukan Service tujuan, kemudian Service meneruskan permintaan ke Pod yang sesuai. Pemisahan ini membuat hostname tetap stabil saat jumlah Pod berubah atau rollout dilakukan.

Untuk aplikasi HTTP/HTTPS, Route dapat dikonfigurasi dengan TLS termination sesuai kebutuhan, misalnya edge, passthrough, atau re-encrypt. Pemilihan mode menentukan di mana koneksi TLS diakhiri dan apakah trafik antara router dan backend dienkripsi ulang. Sertifikat, hostname, dan kebijakan keamanan harus disesuaikan dengan lingkungan. Pada sistem produksi, keputusan routing tidak seharusnya dibuat hanya demi cepat dapat diakses; ia berkaitan dengan identitas domain, trust, dan keamanan data.

Flowchart berikut merangkum perjalanan aplikasi dari source code hingga pengguna. Panah kembali dari observability ke source menegaskan bahwa deployment bukan akhir proses. Log, metrics, health check, dan umpan balik pengguna akan memicu perubahan source atau konfigurasi, kemudian siklus build dan rollout dimulai kembali.



Gambar 2. Flowchart siklus aplikasi dari source code hingga dapat diakses pengguna.

4.8 Routing, DNS, dan TLS

Route membuat aplikasi lebih mudah dipublikasikan, tetapi keberhasilan akses tetap melibatkan DNS, router, Service, dan Pod. Ketika hostname tidak dapat dibuka, pemeriksaan

perlu dilakukan berurutan. Pertama, pastikan Route memiliki host yang benar. Kedua, periksa apakah router menerima host tersebut. Ketiga, lihat Service dan endpoint. Keempat, pastikan Pod berada pada kondisi Ready dan aplikasi mendengarkan port yang sesuai. Urutan ini mencegah diagnosis melompat langsung ke aplikasi padahal masalah berada pada DNS atau selector Service.

Port juga sering menjadi sumber kesalahan. Service memiliki port dan targetPort, sementara container memiliki containerPort sebagai dokumentasi dan referensi. targetPort harus mengarah ke port tempat proses aplikasi mendengarkan. Jika aplikasi berjalan pada 8080 tetapi Service meneruskan ke 8081, Route tetap terlihat valid namun backend tidak merespons. Pemeriksaan endpoint dan log router dapat membantu membedakan timeout dari error aplikasi.

TLS memberi lapisan tambahan. Edge termination mengakhiri TLS di router dan meneruskan trafik ke backend sesuai konfigurasi. Re-encrypt mengakhiri koneksi klien pada router lalu membuat koneksi TLS baru ke backend. Passthrough mempertahankan TLS sampai Pod sehingga router tidak mendekripsi isi. Tidak ada satu mode yang selalu paling benar; pilihan bergantung pada kebutuhan aplikasi dan kebijakan keamanan.

Pada lingkungan multi-tenant, administrator juga dapat menerapkan network policy untuk membatasi komunikasi antarPod atau namespace. Service dan Route membuat jalur komunikasi menjadi mudah, sementara network policy menegaskan jalur mana yang boleh digunakan. Pengembang perlu menyadari bahwa aplikasi yang berjalan baik di cluster tanpa policy dapat gagal berkomunikasi ketika policy ketat diterapkan. Oleh karena itu, kebutuhan trafik sebaiknya didokumentasikan sejak desain.

4.9 ConfigMap, Secret, dan Konfigurasi Lingkungan

Aplikasi tidak seharusnya menanam seluruh konfigurasi ke image. URL layanan, mode log, nama queue, feature flag, dan pengaturan lain dapat berbeda antara development dan production. ConfigMap digunakan untuk data konfigurasi non-rahasia, sedangkan Secret digunakan untuk data sensitif seperti password, token, atau certificate material. Keduanya dapat dipasang sebagai environment variable atau file ke dalam Pod.

Pemisahan konfigurasi dari image memberi keuntungan operasional. Tim dapat menjalankan image yang sama di beberapa environment tanpa rebuild. Hal ini mengurangi variasi artefak dan mempermudah promosi. Namun, perubahan ConfigMap atau Secret tidak selalu membuat aplikasi otomatis memuat ulang nilai baru. Jika nilai dibaca sebagai environment variable pada startup, Pod perlu direstart. Jika dipasang sebagai volume, file dapat berubah tetapi aplikasi tetap harus mendukung reload bila ingin menggunakan nilai baru tanpa restart.

Secret bukan berarti data otomatis aman dalam semua kondisi. Base64 bukan enkripsi. Keamanan Secret bergantung pada kontrol akses, konfigurasi etcd, kebijakan cluster, dan cara aplikasi menggunakan data. Prinsip least privilege tetap berlaku. Hanya ServiceAccount atau pengguna yang membutuhkan secret tertentu yang seharusnya memiliki hak untuk membacanya. Credential juga tidak perlu dicetak ke log.

Nama konfigurasi sebaiknya konsisten dan mudah ditelusuri. Menggunakan satu Secret besar berisi seluruh password berbagai aplikasi membuat blast radius terlalu lebar. Lebih baik membagi secret berdasarkan kebutuhan workload. Demikian pula, ConfigMap yang terlalu besar dan memuat konfigurasi tidak berkaitan akan sulit dirawat. Struktur yang rapi memudahkan audit serta deployment.

4.10 Persistent Storage dan PersistentVolumeClaim

Filesystem container pada Pod cocok untuk data sementara tetapi tidak untuk data yang harus bertahan setelah Pod diganti. OpenShift, melalui Kubernetes, menggunakan PersistentVolume dan PersistentVolumeClaim untuk menghubungkan kebutuhan aplikasi dengan storage yang disediakan cluster. PVC menyatakan kapasitas, access mode, dan storage class yang dibutuhkan. Administrator atau dynamic provisioner kemudian menyediakan volume yang sesuai [10].

Bagi pengembang, pertanyaan pertama bukan 'bagaimana membuat PVC', melainkan 'data apa yang benar-benar harus persisten'. Cache yang dapat dibangun ulang mungkin tidak perlu volume persisten. Sebaliknya, file pengguna, data basis data, atau artefak yang harus bertahan memerlukan solusi storage yang tepat. Banyak aplikasi modern memilih layanan database eksternal atau managed service, tetapi prinsip ketahanannya sama: state tidak boleh bergantung pada umur satu Pod.

Access mode perlu dipahami. ReadWriteOnce, ReadOnlyMany, dan ReadWriteMany memiliki konsekuensi berbeda serta bergantung pada storage backend. Aplikasi yang ingin beberapa replika menulis volume yang sama membutuhkan backend yang mendukung pola tersebut. Jika tidak, arsitektur aplikasi perlu diubah, misalnya dengan menggunakan object storage atau memisahkan state ke service khusus.

Storage juga memengaruhi rollout. Pod baru mungkin gagal Ready karena volume tidak dapat dimount, PVC belum Bound, atau permission filesystem tidak sesuai. Event pada Pod dan PVC biasanya memberi petunjuk. Troubleshooting storage sebaiknya memeriksa status PVC, storage class, event, mount path, security context, dan perilaku aplikasi ketika direktori tidak tersedia. Dengan demikian, masalah tidak disalahartikan sebagai bug aplikasi semata.

4.11 Resource Request, Limit, dan Scaling

Cluster memiliki kapasitas terbatas sehingga setiap workload perlu menyatakan kebutuhan resource secara rasional. Request digunakan scheduler untuk mempertimbangkan penempatan Pod, sedangkan limit membatasi penggunaan maksimum sesuai jenis resource. Menetapkan request terlalu besar membuang kapasitas dan membuat Pod sulit dijadwalkan. Menetapkan terlalu kecil dapat membuat node kelebihan beban atau aplikasi mengalami throttling. Nilai sebaiknya berasal dari pengukuran, bukan tebakan tetap untuk semua aplikasi.

Scaling horizontal menambah jumlah replika Pod. Untuk layanan stateless, pendekatan ini sering lebih mudah daripada memperbesar satu instance. Namun, replika tambahan hanya berguna jika dependency, database, queue, dan storage juga mampu menampung peningkatan trafik. Horizontal Pod Autoscaler dapat menyesuaikan replika berdasarkan metrics tertentu [25]. Autoscaling bukan pengganti capacity planning; ia hanya mengotomatisasi respons terhadap sinyal yang sudah dipilih.

Scaling manual tetap berguna untuk pengujian dan situasi tertentu. Developer Console dapat menampilkan jumlah Pod dan menyediakan kontrol scaling, sedangkan CLI dapat mengubah replicas. Ketika jumlah replika dinaikkan, Service tetap menjadi endpoint yang sama dan mendistribusikan trafik ke Pod yang Ready. Hal ini memperlihatkan manfaat pemisahan antara workload dan jaringan.

Aplikasi juga perlu memahami graceful shutdown. Saat rollout atau scale down, Pod menerima sinyal terminasi. Jika proses langsung berhenti tanpa menyelesaikan request aktif, pengguna dapat menerima error. Readiness, termination grace period, dan handler shutdown yang benar membantu mengurangi gangguan. Scalability akhirnya bukan hanya soal menambah replika, tetapi memastikan lifecycle aplikasi selaras dengan lifecycle Pod.

4.12 Observability: Metrics, Log, dan Event

OpenShift 4.10 menyediakan monitoring stack terintegrasi untuk komponen platform dan dapat dikonfigurasi agar memonitor project pengguna [11]. Bagi pengembang, observability setidaknya memiliki tiga sumber utama: metrics, logs, dan events. Ketiganya menjawab pertanyaan berbeda. Metrics memperlihatkan tren dan kondisi numerik, logs menjelaskan kejadian dari sudut aplikasi atau komponen, sedangkan events memberikan konteks perubahan resource pada cluster.

Metrics berguna untuk membaca CPU, memory, request rate, latency, atau indikator aplikasi lain. Nilai sesaat harus dibaca bersama tren. CPU tinggi selama beberapa detik mungkin normal, sedangkan memory yang terus naik dapat menunjukkan leak. Dashboard membantu visualisasi, tetapi alert harus dirancang agar memiliki tindakan yang jelas. Alert yang terlalu sering dan tidak actionable akan diabaikan oleh tim.

Log aplikasi sebaiknya diarahkan ke stdout dan stderr sehingga collector platform dapat mengambilnya. Log yang baik mencantumkan konteks cukup untuk diagnosis, tetapi tidak membocorkan password, token, atau data sensitif. Format terstruktur seperti JSON dapat membantu pencarian pada sistem logging terpusat. Dokumentasi logging OpenShift membedakan log aplikasi, infrastruktur, dan audit [12].

Events sering dilupakan padahal sangat berguna. ImagePullBackOff, FailedScheduling, mount error, atau probe failure biasanya meninggalkan event yang memberi petunjuk sebelum log aplikasi tersedia. Alur diagnosis yang baik dimulai dari status resource, kemudian events, log, metrics, dan dependency. Observability harus dirancang bersama aplikasi sejak awal, bukan ditambahkan setelah insiden terjadi.

4.13 CI/CD sebagai Kelanjutan Lifecycle Aplikasi

Deployment manual dapat diterima untuk eksperimen, tetapi sulit dijaga konsistensinya pada tim besar. Setiap langkah manual membuka peluang perbedaan urutan, kesalahan parameter, atau perubahan yang tidak tercatat. Continuous Integration dan Continuous Delivery mengubah langkah-langkah tersebut menjadi proses yang dapat diulang. Dokumentasi OpenShift 4.10 mengelompokkan solusi CI/CD menjadi OpenShift Builds, OpenShift Pipelines, dan OpenShift GitOps [15].

Continuous Integration berfokus pada integrasi perubahan source secara sering melalui build dan test otomatis. Tujuannya bukan sekadar menghasilkan image baru, tetapi memberi umpan balik cepat apakah perubahan masih memenuhi standar. Continuous Delivery membawa artefak yang telah diuji menuju environment berikutnya melalui proses yang konsisten. Deployment ke production dapat otomatis atau memerlukan persetujuan, tergantung tingkat risiko dan kebijakan organisasi.

OpenShift Builds menangani proses build deklaratif. Pipelines menyusun beberapa task seperti clone, test, build, scan, push, dan deploy. GitOps menggunakan repository Git sebagai sumber kondisi yang diinginkan. Ketiganya dapat dipakai bersama. Contohnya, pipeline membangun image dan memperbarui tag pada repository konfigurasi; GitOps controller kemudian menyelaraskan cluster dengan konfigurasi baru.

Desain CI/CD sebaiknya sederhana dan dapat dibaca. Pipeline yang sangat panjang serta penuh logika kondisional sulit dirawat. Tugas yang berulang dapat dipisah menjadi task yang dapat digunakan ulang. Secret untuk registry atau repository disediakan melalui mekanisme yang aman. Yang paling penting, pipeline harus menghasilkan jejak yang dapat diaudit: commit apa yang dibangun, image apa yang dihasilkan, test apa yang berjalan, dan konfigurasi mana yang akhirnya dideploy.

4.14 OpenShift Pipelines dan Tekton

OpenShift Pipelines dibangun di atas Tekton, sebuah framework pipeline cloud-native yang menggunakan custom resource Kubernetes untuk mendefinisikan Task, Pipeline, dan PipelineRun. Pendekatan ini membuat pipeline menjadi objek deklaratif yang dapat dibuat, diperiksa, dan dikelola melalui API cluster. Langkah pipeline tidak bergantung pada satu server CI tradisional yang menyimpan konfigurasi tersembunyi.

Task mendefinisikan unit pekerjaan, misalnya menjalankan test atau membangun image. Pipeline menghubungkan beberapa task dan mengatur dependency. PipelineRun adalah instans eksekusi dengan parameter tertentu. Workspace menyediakan mekanisme berbagi file antartask, sedangkan ServiceAccount dan Secret digunakan untuk kredensial. Dengan komponen ini, pipeline dapat disusun modular dan digunakan ulang.

Sebuah alur sederhana dapat terdiri dari git-clone, unit-test, build-image, image-scan, dan deploy. Jika test gagal, tahap berikutnya tidak perlu dijalankan. Jika image berhasil dibangun, digest dapat dicatat sehingga artefak yang dipromosikan diketahui secara pasti. Pada organisasi yang membutuhkan approval, pipeline dapat berhenti sebelum deployment production dan dilanjutkan setelah persetujuan.

Pengembang perlu menghindari asumsi bahwa pipeline otomatis selalu lebih aman. Credential yang terlalu luas, task dari sumber tidak terpercaya, atau pipeline yang langsung mengubah production tanpa kontrol dapat meningkatkan risiko. Pipeline perlu mengikuti least privilege dan prinsip supply-chain security. Otomasi seharusnya mengurangi variasi sekaligus memperkuat audit, bukan mempercepat kesalahan.

4.15 GitOps dan Sumber Kondisi yang Diinginkan

GitOps menerapkan gagasan bahwa kondisi yang diinginkan untuk aplikasi atau infrastruktur disimpan dalam repository Git. Controller di cluster membandingkan kondisi aktual dengan manifest di repository lalu melakukan sinkronisasi. OpenShift GitOps menggunakan pendekatan berbasis Argo CD untuk kebutuhan tersebut. Pada OpenShift 4.10, GitOps menjadi salah satu solusi CI/CD yang didokumentasikan bersama Builds dan Pipelines [15].

Keuntungan utama GitOps adalah auditability. Perubahan konfigurasi dapat dilihat sebagai commit, direview melalui pull request, dan dikembalikan ke versi sebelumnya. Administrator tidak harus menerima perubahan melalui perintah manual tanpa catatan. Git menjadi sumber sejarah yang mudah dibaca oleh manusia. Namun, repository harus dikelola dengan disiplin karena kesalahan manifest yang disetujui juga dapat disebarkan secara konsisten ke cluster.

GitOps cocok untuk konfigurasi deklaratif seperti Deployment, Service, Route, ConfigMap, dan policy. Secret membutuhkan perhatian lebih karena tidak semestinya disimpan dalam

bentuk plaintext. Organisasi dapat menggunakan encrypted secret, external secret management, atau mekanisme lain sesuai kebijakan. Prinsipnya, repository harus tetap dapat dibagikan kepada pihak yang berwenang tanpa membocorkan credential.

Hubungan Pipelines dan GitOps perlu dibedakan. Pipeline biasanya mengerjakan proses imperatif seperti build dan test. GitOps controller menjaga kondisi deklaratif cluster. Pipeline dapat memperbarui repository konfigurasi, tetapi perubahan ke cluster idealnya dilakukan melalui controller agar jalur deployment tetap konsisten. Pemisahan ini membuat build artefak dan deployment konfigurasi memiliki rekam jejak yang jelas.

4.16 Operator dan Otomasi Operasional

Operator adalah pola untuk mengemas pengetahuan operasional ke dalam software yang bekerja melalui API Kubernetes. Dokumentasi OpenShift menjelaskan Operator sebagai metode packaging, deploying, dan managing aplikasi Kubernetes, termasuk kemampuan melakukan health check, update, dan rekonsiliasi kondisi [18]. Operator cocok untuk layanan yang memiliki lifecycle lebih kompleks daripada Deployment biasa, misalnya database, messaging, atau platform observability.

Operator biasanya memperkenalkan Custom Resource Definition. Pengguna membuat custom resource dengan konfigurasi tingkat domain, lalu controller Operator menerjemahkannya menjadi resource yang diperlukan. Alih-alih pengguna membuat puluhan objek secara manual, Operator menjalankan prosedur yang telah dikodekan oleh pengembang platform atau vendor. Pendekatan ini dapat meningkatkan konsistensi, terutama untuk instalasi dan upgrade yang memerlukan langkah khusus.

Operator Lifecycle Manager membantu distribusi dan pengelolaan Operator di OpenShift. Administrator dapat memilih Operator yang tersedia dan mengatur scope instalasi sesuai kebutuhan. Namun, pemasangan Operator tetap perlu dinilai dari sisi permission, sumber image, update channel, dan dampak resource. Operator memiliki controller yang berjalan terus menerus sehingga hak aksesnya harus dipahami.

Bagi pengembang aplikasi, Operator tidak selalu perlu dibuat sendiri. Banyak kebutuhan dapat diselesaikan dengan Deployment, Service, ConfigMap, dan pipeline. Operator menjadi tepat ketika terdapat pengetahuan operasional berulang yang perlu diotomatisasi, misalnya backup, failover, schema upgrade, atau scaling berbasis kondisi domain. Menggunakan Operator hanya karena teknologi terlihat canggih justru dapat menambah kompleksitas yang tidak diperlukan.

4.17 Keamanan: RBAC, ServiceAccount, dan SCC

Keamanan OpenShift hadir pada beberapa lapisan. RBAC mengatur siapa dapat melakukan tindakan terhadap resource. ServiceAccount memberi identitas kepada workload. Security

Context Constraints atau SCC membatasi karakteristik Pod yang diperbolehkan, seperti user ID, privilege, volume, dan capability. Dokumentasi Security and Compliance OpenShift 4.10 menempatkan keamanan container, sertifikat, dan enkripsi sebagai bagian dari pertahanan platform [13].

Prinsip least privilege harus diterapkan pada pengguna maupun aplikasi. Pengembang tidak perlu meminta cluster-admin hanya agar dapat melihat log. Demikian pula, ServiceAccount aplikasi tidak seharusnya memiliki hak mengubah seluruh namespace bila tugasnya hanya membaca ConfigMap tertentu. Permission yang terlalu luas memperbesar dampak jika credential bocor atau aplikasi dieksploitasi.

Container sebaiknya dapat berjalan sebagai non-root dan tidak mengandalkan user ID tetap kecuali benar-benar diperlukan. OpenShift secara historis menerapkan pembatasan yang membuat banyak image yang hanya dapat berjalan sebagai root perlu diperbaiki. Aplikasi yang mengikuti praktik container aman akan lebih mudah dipindahkan antarcluster dan tidak memerlukan pengecualian kebijakan berlebihan.

Keamanan juga berkaitan dengan network policy, image provenance, secret, TLS, dan update dependency. Tidak ada satu resource yang dapat menyelesaikan semua masalah. Pengembang perlu memandang security sebagai bagian dari lifecycle: source diperiksa, image dibangun dari base yang dipercaya, credential tidak dimasukkan ke image, permission dibatasi, trafik dienkripsi bila diperlukan, dan log tidak mencetak data rahasia.

4.18 DevSecOps, Image, dan Supply Chain

DevSecOps menempatkan pemeriksaan keamanan di sepanjang pipeline, bukan hanya pada tahap akhir sebelum rilis. Pada aplikasi container, supply chain mencakup source code, dependency, builder image, base image, pipeline task, registry, dan manifest deployment. Jika salah satu komponen berasal dari sumber yang tidak dapat dipercaya, image akhir dapat membawa risiko walaupun kode aplikasi sendiri terlihat benar.

Base image perlu dikelola seperti dependency. Tim harus mengetahui sumbernya, versi, dan cara menerima update keamanan. Membiarkan tag mutable tanpa proses rebuild membuat aplikasi mungkin tetap memakai layer lama yang memiliki kerentanan. Sebaliknya, melakukan rebuild otomatis tanpa pengujian juga dapat membawa perubahan yang tidak kompatibel. Pipeline perlu menyeimbangkan update, test, dan kontrol rilis.

Image scanning dapat digunakan untuk menemukan paket rentan, tetapi hasil scan perlu diprioritaskan berdasarkan exploitability dan konteks aplikasi. Tidak semua temuan memiliki risiko yang sama. Policy dapat mencegah image tertentu masuk ke production, sementara exception harus memiliki alasan serta batas waktu. Catatan mengenai digest image membantu memastikan artefak yang diuji adalah artefak yang benar-benar dideploy.

Secret management merupakan bagian supply chain yang sering menjadi titik lemah. Token CI, registry credential, dan kunci deploy tidak seharusnya disimpan di repository plaintext. Pipeline menggunakan ServiceAccount dan Secret dengan scope minimal. Audit terhadap siapa dapat membaca secret sama pentingnya dengan cara secret dibuat. Keamanan yang baik tidak bergantung pada satu scanner, tetapi pada rangkaian kontrol yang saling memperkuat.

4.19 Troubleshooting Aplikasi secara Sistematis

Troubleshooting pada OpenShift lebih efektif bila mengikuti alur resource. Ketika pengguna melaporkan aplikasi tidak bisa diakses, pemeriksaan dapat dimulai dari Route, Service, endpoint, Pod, lalu ke aplikasi. Jika Pod tidak ada, telusuri Deployment dan ReplicaSet. Jika Pod ada tetapi gagal start, baca describe, events, dan logs. Jika image tidak dapat di-pull, periksa nama image, registry, tag, dan pull secret. Urutan ini menghindari perubahan acak yang dapat memperburuk keadaan.

Untuk masalah build, periksa status Build, build log, source, builder, serta akses network. Untuk masalah scheduling, lihat events yang mungkin menunjukkan resource tidak cukup, node selector tidak cocok, atau PVC belum tersedia. Untuk masalah readiness, periksa endpoint probe dari dalam Pod dan dependency yang digunakan. Untuk masalah trafik, pastikan selector Service cocok dengan label Pod dan targetPort benar.

CLI menyediakan pola pemeriksaan yang sederhana namun kuat. oc get memberi gambaran state, oc describe memberi event dan detail, oc logs membaca output proses, dan oc get <resource> -o yaml memperlihatkan spec serta status lengkap. Memakai watch dapat membantu melihat perubahan saat rollout. Pengembang sebaiknya menyimpan urutan diagnosis sebagai runbook tim sehingga pengetahuan tidak hanya berada pada satu orang.

Hal penting lain adalah menghindari perbaikan yang menghapus bukti. Menghapus Pod berulang kali tanpa membaca log sebelumnya dapat menghilangkan konteks. Mengubah banyak resource sekaligus membuat penyebab sulit diketahui. Praktik yang baik adalah mengumpulkan status, event, log, dan perubahan terakhir; membuat satu hipotesis; menerapkan perubahan minimal; lalu mengukur hasilnya. OpenShift menyediakan banyak informasi, tetapi kualitas diagnosis tetap bergantung pada disiplin proses.

```
oc get pods -o wide
oc describe pod <nama-pod>
oc logs <nama-pod> --previous
oc get svc,route,endpoints
oc get events --sort-by=.lastTimestamp
```

BAB V
STUDI KASUS DEPLOYMENT APLIKASI SEDERHANA

Studi kasus ini menggunakan aplikasi web sederhana yang source code-nya berada di repository Git. Tujuannya bukan menyediakan tutorial produksi yang lengkap, melainkan menunjukkan hubungan antarresource dalam satu alur yang mudah diikuti. Diasumsikan pengguna telah memiliki akses ke cluster OpenShift 4.10, dapat menggunakan oc, dan memiliki hak membuat resource pada project sendiri.

Langkah pertama adalah membuat project, misalnya demo-web. Setelah konteks project aktif, aplikasi dibuat dari source code. OpenShift dapat membuat BuildConfig, ImageStream, Deployment, dan Service sesuai hasil deteksi source. Pengembang memantau build sampai image berhasil dihasilkan. Jika build gagal, proses berhenti pada tahap ini dan log build menjadi sumber diagnosis.

Setelah image siap, Deployment membuat Pod. Pengembang memeriksa oc get pods dan menunggu kondisi Ready. Jika Pod restart, oc logs dan oc describe pod digunakan untuk mencari penyebab. Setelah Pod sehat, Service diperiksa untuk memastikan selector menunjuk label Pod yang benar. Route kemudian dibuat untuk mengekspos Service. Host Route menjadi alamat yang digunakan pengguna untuk mengakses aplikasi.

Langkah terakhir adalah melakukan perubahan source dan mengamati siklus berikutnya. Commit baru memicu build sesuai konfigurasi, image baru muncul, dan deployment melakukan rollout. Service serta Route tetap menjadi titik akses yang sama. Observability digunakan untuk memastikan versi baru sehat. Dari studi kasus ini terlihat bahwa perubahan kode tidak mengharuskan pengembang membuat ulang seluruh jalur jaringan; resource memiliki tanggung jawab masing-masing dan dapat bereaksi terhadap perubahan di lapisan sebelumnya.

```
oc new-project demo-web
oc new-app <repository-git> --name=demo-web
oc get builds,pods
oc expose service demo-web
oc get route demo-web
```

5.1 Contoh Resource dan Jalur Verifikasi

Pada praktik nyata, manifest deklaratif lebih mudah direview dibanding rangkaian perintah manual. Deployment menyebut image dan template Pod, Service memilih Pod berdasarkan label, sedangkan Route menunjuk Service. Kunci konsistensi terdapat pada nama, label, port, dan namespace. Kesalahan kecil pada salah satu elemen tersebut cukup membuat jalur trafik terputus.

Setelah resource diterapkan, verifikasi sebaiknya dilakukan berlapis. Pertama, periksa Deployment dan jumlah replika yang tersedia. Kedua, lihat Pod dan pastikan readiness bernilai benar. Ketiga, periksa Service serta endpoint untuk memastikan alamat Pod terdaftar. Keempat,

lihat Route dan lakukan request ke host. Jika request gagal, lokasi masalah dapat dipersempit berdasarkan lapisan yang terakhir berhasil.

Pemeriksaan konfigurasi juga penting. Environment variable dan secret harus sesuai environment. PVC harus Bound sebelum aplikasi yang membutuhkannya dianggap siap. Resource request/limit perlu cukup untuk startup. Health probe harus menunjuk endpoint yang benar. Pipeline atau GitOps sebaiknya tidak melanjutkan promosi jika kondisi aplikasi tidak memenuhi kriteria.

Studi kasus sederhana ini menunjukkan alasan OpenShift mengandalkan banyak resource kecil. Setiap resource mengabstraksikan satu bagian lifecycle dan memungkinkan automation bekerja pada bagian tersebut. Arsitektur memang terlihat lebih banyak dibanding satu server monolitik, tetapi pembagian ini memberi kemampuan scaling, rollout, recovery, dan observability yang sulit dicapai secara konsisten dengan konfigurasi manual.

Resource	Fungsi utama	Verifikasi awal
Deployment	Menjaga jumlah dan template Pod	oc get deploy
Pod	Menjalankan container aplikasi	oc get pods
Service	Endpoint internal stabil	oc get svc,endpoints
Route	Akses eksternal melalui hostname	oc get route
ConfigMap/Secret	Konfigurasi dan data sensitif	oc get cm,secret

Tabel 1. Resource yang diperiksa pada jalur verifikasi aplikasi.

BAB VI
ANALISIS DAN PEMBAHASAN LANJUT

Dari sudut pengembang, kelebihan utama OpenShift adalah konsistensi jalur aplikasi. Platform menyediakan pola yang sama untuk project, build, image, deployment, service, routing, monitoring, dan delivery. Tim tidak perlu merangkai seluruh komponen dari nol untuk setiap proyek. Developer Console memberi pengalaman visual, sementara API dan CLI tetap mendukung automation. Kombinasi ini dapat mempercepat onboarding asalkan standar organisasi dibuat cukup jelas.

Kelebihan kedua adalah pemisahan tanggung jawab. Pengembang dapat fokus pada image, konfigurasi, health, dan resource aplikasi; administrator menangani kapasitas cluster, node, storage class, ingress infrastructure, serta policy global. Batas tersebut tidak mutlak, tetapi membuat kerja tim lebih terstruktur. Platform yang sama juga dapat digunakan pada data center maupun cloud sehingga konsep aplikasi relatif konsisten meskipun infrastruktur di bawahnya berbeda.

Tantangannya adalah kurva belajar. Pengguna yang baru berpindah dari server tradisional harus memahami container, YAML, resource Kubernetes, network service, security, dan observability sekaligus. OpenShift menambahkan resource khas seperti Route, ImageStream, BuildConfig, SCC, dan Operator. Jika organisasi tidak menyediakan pola aplikasi, naming convention, template, dan runbook, jumlah fitur justru dapat membingungkan.

Tantangan lain adalah biaya operasional dan kompleksitas platform. OpenShift menyediakan banyak kemampuan, tetapi kemampuan tersebut perlu dikelola: cluster harus diperbarui, monitoring diperhatikan, permission diaudit, storage direncanakan, dan pipeline dipelihara. Karena itu, OpenShift paling bernilai ketika digunakan sebagai platform bersama dengan governance yang matang, bukan sekadar cara lebih rumit untuk menjalankan satu container sederhana.

6.1 OpenShift dan Kubernetes: Persamaan dan Perbedaan Praktis

OpenShift dan Kubernetes sering dibandingkan seolah-olah harus memilih salah satu pada tingkat konsep. Perbandingan yang lebih tepat adalah bahwa Kubernetes menjadi fondasi API dan orkestrasi, sedangkan OpenShift menyusun produk platform di atas fondasi tersebut. Pod, Deployment, Service, ConfigMap, Secret, dan PVC tetap resource Kubernetes. OpenShift menambahkan integrasi seperti Route, ImageStream, BuildConfig, Developer Console, SCC, Operator lifecycle, dan paket CI/CD.

Bagi pengembang, hal ini berarti pengetahuan Kubernetes tetap dapat dibawa ke OpenShift. Manifest Deployment yang standar umumnya tetap relevan. Perintah oc bahkan memiliki

banyak pola yang mirip dengan kubectl. Perbedaan baru terlihat ketika aplikasi memanfaatkan fasilitas platform, misalnya membuat Route untuk publikasi, menggunakan ImageStream untuk image trigger, atau mengikuti constraint keamanan OpenShift.

Keamanan menjadi salah satu area yang terasa berbeda. Image yang dibuat dengan asumsi harus berjalan sebagai root mungkin berhasil pada cluster Kubernetes yang permisif, tetapi menemui pembatasan pada OpenShift. Perbedaan ini bukan sekadar hambatan; ia mendorong image lebih portabel dan aman. Pengembang perlu menyesuaikan permission filesystem, user, port, dan capability agar image tidak membutuhkan privilege yang tidak perlu.

Kesimpulan praktisnya bukan bahwa satu lebih baik untuk semua kondisi, melainkan bahwa OpenShift memberi paket platform dan opini operasional yang lebih terintegrasi. Organisasi yang sudah memiliki tim platform kuat dapat merangkai Kubernetes dengan tool lain. Organisasi yang membutuhkan platform enterprise terpadu dapat memilih OpenShift untuk mengurangi variasi komponen. Bagi aplikasi, konsep dasar cloud-native tetap sama: deklaratif, immutable image, stateless bila memungkinkan, observability, dan automation.

BAB VII
PENUTUP

OpenShift Container Platform dapat dipahami sebagai platform aplikasi berbasis Kubernetes yang menyatukan proses build, deployment, networking, configuration, storage, observability, security, dan automation. Pada versi 4.10 yang digunakan sebagai rujukan tahun 2022, platform berjalan di atas Kubernetes 1.23 dengan CRI-O runtime [1]. Fondasi Kubernetes tetap terlihat pada Pod, Deployment, Service, ConfigMap, Secret, dan PersistentVolumeClaim, sementara OpenShift menambahkan pengalaman dan resource yang membuat lifecycle aplikasi lebih terpadu.

Alur paling penting untuk diingat adalah perjalanan dari source code atau image menuju aplikasi yang dapat diakses. Source dapat dibangun melalui BuildConfig atau pipeline, hasilnya dilacak sebagai image, Deployment membuat Pod, Service memberi endpoint internal yang stabil, dan Route membuka akses eksternal. ConfigMap serta Secret memisahkan konfigurasi dari image, PVC mempertahankan data, probe membantu platform menilai kesehatan, sedangkan metrics, logs, dan events memberi visibilitas terhadap kondisi berjalan.

CI/CD, GitOps, dan Operator memperluas lifecycle tersebut. Pipeline membuat build dan test dapat diulang, GitOps menjaga kondisi cluster agar sesuai dengan repository deklaratif, dan Operator mengemas pengetahuan operasional untuk layanan yang kompleks. Semua mekanisme tersebut tetap memerlukan kontrol keamanan, permission yang tepat, serta proses rilis yang dapat diaudit. Otomasi tidak menghapus tanggung jawab desain; ia hanya membuat desain yang baik dapat diterapkan secara konsisten.

Sebagai saran, pembelajaran berikutnya sebaiknya dilakukan melalui praktik pada aplikasi kecil yang memiliki source, build, route, ConfigMap, Secret, PVC, readiness probe, dan pipeline. Setelah alur tersebut dikuasai, topik lanjutan seperti NetworkPolicy, autoscaling berbasis custom metrics, service mesh, advanced GitOps, Operator SDK, dan strategi multi-cluster akan lebih mudah dipahami. Dasar yang kuat pada lifecycle aplikasi membuat perubahan teknologi dapat diikuti tanpa kehilangan kerangka berpikir.

7.1 Implikasi dan Saran Lanjutan

Bagi pengembang yang baru memulai, praktik paling efektif adalah membangun satu aplikasi kecil secara berulang sampai hubungan antarresource dapat dibaca tanpa bergantung pada hafalan perintah. Setiap perubahan sebaiknya diamati dari source, build, image, rollout, readiness, Service, Route, hingga log dan metrics. Latihan seperti ini membentuk kebiasaan diagnosis berbasis bukti dan mengurangi kecenderungan melakukan perubahan acak ketika terjadi gangguan.

DAFTAR PUSTAKA

[1] Red Hat. 2022. OpenShift Container Platform 4.10 Release Notes. Red Hat Documentation.

[2] Red Hat. 2022. Getting Started with OpenShift Container Platform 4.10. Red Hat Documentation.

[3] Red Hat. 2022. Building Applications: Creating and Managing Applications on OpenShift Container Platform 4.10. Red Hat Documentation.

[4] Red Hat. 2022. CLI Tools: OpenShift CLI (oc), OpenShift Container Platform 4.10. Red Hat Documentation.

[5] Red Hat. 2022. Images: Managing Images and Image Streams, OpenShift Container Platform 4.10. Red Hat Documentation.

[6] Red Hat. 2022. Builds: Configuring and Managing Builds, OpenShift Container Platform 4.10. Red Hat Documentation.

[7] Red Hat. 2022. Workloads: Deployments and DeploymentConfig, OpenShift Container Platform 4.10. Red Hat Documentation.

[8] Red Hat. 2022. Networking: Configuring Cluster Networking and Application Access, OpenShift Container Platform 4.10. Red Hat Documentation.

[9] Red Hat. 2022. Configuring Routes and Ingress, OpenShift Container Platform 4.10. Red Hat Documentation.

[10] Red Hat. 2022. Storage: Configuring and Managing Persistent Storage in OpenShift Container Platform 4.10. Red Hat Documentation.

[11] Red Hat. 2022. Monitoring: Configuring and Using the Monitoring Stack in OpenShift Container Platform 4.10. Red Hat Documentation.

[12] Red Hat. 2022. Logging: Collecting and Forwarding Application, Infrastructure, and Audit Logs, OpenShift Container Platform 4.10. Red Hat Documentation.

[13] Red Hat. 2022. Security and Compliance: Learning About and Managing Security for OpenShift Container Platform 4.10. Red Hat Documentation.

[14] Red Hat. 2022. Authentication and Authorization, OpenShift Container Platform 4.10. Red Hat Documentation.

[15] Red Hat. 2022. CI/CD: Builds, Pipelines, and GitOps for OpenShift Container Platform 4.10. Red Hat Documentation.

[16] Red Hat. 2022. OpenShift Pipelines Documentation. Red Hat OpenShift Container Platform 4.10.

[17] Red Hat. 2022. OpenShift GitOps Documentation. Red Hat OpenShift Container Platform 4.10.

[18] Red Hat. 2022. Operators: Working with Operators in OpenShift Container Platform 4.10. Red Hat Documentation.

[19] Red Hat. 2022. Web Console: Developer Perspective and Topology View, OpenShift Container Platform 4.10. Red Hat Documentation.

[20] Red Hat. 2022. Registry: Configuring Registries and Integrated Image Registry, OpenShift Container Platform 4.10. Red Hat Documentation.

[21] Kubernetes Authors. 2022. Pods. Kubernetes Documentation.

[22] Kubernetes Authors. 2022. Services, Load Balancing, and Networking. Kubernetes Documentation.

[23] Kubernetes Authors. 2022. Deployments and ReplicaSets. Kubernetes Documentation.

[24] Kubernetes Authors. 2022. Configure Liveness, Readiness and Startup Probes. Kubernetes Documentation.

[25] Kubernetes Authors. 2022. Horizontal Pod Autoscaling. Kubernetes Documentation.

[26] Kubernetes Authors. 2022. ConfigMaps and Secrets. Kubernetes Documentation.

[27] Kubernetes Authors. 2022. Persistent Volumes. Kubernetes Documentation.

[28] OpenShift Source-to-Image Project. 2022. Source-to-Image (S2I) Documentation. OpenShift Project.

[29] Tekton Authors. 2022. Tekton Pipelines Documentation: Tasks, Pipelines, PipelineRuns, Workspaces, and Results.

[30] Argo Project. 2022. Argo CD Documentation: Declarative GitOps Continuous Delivery for Kubernetes.