IMPLEMENTASI DEVOPS LIFECYCLE BERBASIS CLOUD-NATIVE PADA ORACLE CLOUD INFRASTRUCTURE

Oleh:
Muhammad Oskhar Mubarok (1122091000042)
Muhammad Fatihul Choir (11220910000121)
Program Studi: Teknik Informatika
Fakultas: Sains dan Teknologi
Institusi: Universitas Islam Negeri Syarif Hidayatullah Jakarta

Abstrak

Eksperimen ini mendemonstrasikan penerapan menyeluruh dari siklus kerja DevOps menggunakan platform Oracle Cloud Infrastructure (OCI). Proyek ini difokuskan pada integrasi berbagai teknologi *cloud-native* dengan pendekatan *Infrastructure as Code* (IaC) untuk membangun sistem yang otomatis, efisien, dan tangguh. Eksperimen dilakukan dalam lima tahap utama, yaitu perencanaan dan desain infrastruktur, implementasi otomatisasi dengan Terraform, pembuatan *pipeline* CI/CD menggunakan GitHub Actions dan Jenkins, *monitoring* sistem secara *real-time* dengan Prometheus dan Grafana, serta pengujian skenario pemulihan bencana (*disaster recovery*).

Hasil eksperimen menunjukkan adanya peningkatan signifikan dalam efisiensi operasional, kecepatan *deployment*, serta kemampuan pemantauan dan pemulihan sistem. Peningkatan ini tidak hanya berdampak pada waktu *delivery* aplikasi yang lebih cepat, tetapi juga pada pengurangan *human error* dan peningkatan konsistensi lingkungan. Eksperimen ini memberikan gambaran nyata tentang bagaimana prinsip-prinsip DevOps dapat diimplementasikan secara praktis dan terstruktur menggunakan layanan OCI dan alat-alat *open-source* yang populer, membuktikan bahwa kombinasi ini mampu mendukung siklus hidup pengembangan perangkat lunak modern secara efektif dan efisien.

PENDAHULUAN

Dalam era transformasi digital yang semakin cepat, kebutuhan organisasi terhadap sistem teknologi yang andal, fleksibel, dan hemat biaya menjadi sangat penting. Aplikasi dan layanan digital dituntut untuk dapat berkembang secara cepat tanpa mengorbankan stabilitas dan keamanannya. Fenomena ini mendorong adopsi pendekatan DevOps secara luas, sebuah metodologi yang mampu menjembatani proses pengembangan perangkat lunak (*development*) dan operasional (*operations*) secara terintegrasi dan kolaboratif.

DevOps memungkinkan tim untuk merancang, membangun, menguji, dan merilis

aplikasi dengan lebih cepat melalui otomatisasi menyeluruh dan kolaborasi berkelanjutan. Filosofi ini berfokus pada penghapusan *silo* antara tim *dev* dan *ops*, mendorong budaya berbagi tanggung jawab, dan menerapkan praktik seperti *Continuous Integration* (CI), *Continuous Delivery* (CD), dan *Continuous Monitoring*. Tujuannya adalah untuk mempercepat siklus *feedback*, mengurangi *time-to-market*, dan meningkatkan kualitas perangkat lunak secara keseluruhan.

Untuk mendukung proses DevOps yang efisien, penggunaan *cloud-native technology* menjadi pilihan utama karena skalabilitas, ketahanan, dan kemudahan pengelolaannya. *Cloud-native* berarti membangun dan menjalankan aplikasi yang memanfaatkan sepenuhnya model komputasi *cloud*. Salah satu platform yang menyediakan fasilitas lengkap untuk implementasi DevOps adalah Oracle Cloud Infrastructure (OCI). OCI menawarkan berbagai layanan infrastruktur sebagai layanan (IaaS) dan *platform as a service* (PaaS) yang dirancang untuk performa tinggi, keamanan, dan skalabilitas.

OCI menyediakan layanan esensial seperti Oracle Kubernetes Engine (OKE) untuk orkestrasi kontainer, *compute instance* untuk menjalankan *workload* komputasi, *storage* yang beragam (blok, objek, file), dan *monitoring tools* terintegrasi. Layanan-layanan ini dapat dikombinasikan secara sinergis dengan alat *open-source* yang telah menjadi standar industri seperti Terraform untuk *Infrastructure as Code* (IaC), Jenkins untuk otomatisasi *pipeline* CI/CD, serta Prometheus dan Grafana untuk *monitoring* dan *observability*. Dengan pendekatan *Infrastructure as Code* (IaC), infrastruktur tidak lagi dibangun secara manual melalui konsol, melainkan dapat ditulis dalam bentuk skrip yang terversioning dan dikelola layaknya kode program. Hal ini menjamin konsistensi, kemudahan replikasi, dan kemampuan untuk melakukan *rollback* jika terjadi masalah.

Eksperimen ini bertujuan untuk menguji secara langsung bagaimana siklus DevOps dapat diimplementasikan secara menyeluruh dengan memanfaatkan OCI sebagai platform utama. Selain itu, penelitian ini juga akan menilai efektivitas dan keandalannya dalam mendukung proses pengembangan aplikasi modern, memberikan panduan praktis bagi organisasi yang ingin mengadopsi praktik DevOps di lingkungan *cloud*.

METODE PENELITIAN

Eksperimen ini dilakukan dengan pendekatan eksperimental, yaitu dengan merancang dan mengimplementasikan siklus DevOps secara penuh menggunakan Oracle Cloud Infrastructure (OCI) dan berbagai teknologi pendukung berbasis *open-source*.

Pendekatan ini memungkinkan peneliti untuk mengamati secara langsung dampak dari setiap fase implementasi dan memvalidasi efektivitas solusi yang dibangun. Terdapat lima fase utama dalam implementasi, yang masing-masing dirancang untuk merepresentasikan tahapan nyata dalam proses pengembangan perangkat lunak modern. Berikut adalah penjelasan masing-masing fase:

3.1 Fase Perencanaan dan Desain: Merancang Arsitektur Cloud-Native

Fase pertama dalam penelitian ini dimulai dengan merancang arsitektur infrastruktur berbasis *cloud-native* yang akan menjadi pondasi utama dari sistem yang dibangun. Pendekatan *cloud-native* dipilih karena kemampuannya dalam menyediakan sistem yang elastis, mudah diskalakan, dan siap mendukung otomatisasi serta integrasi lanjutan. Tujuan utama dari fase ini adalah memastikan bahwa rancangan infrastruktur dapat mendukung semua proses DevOps secara menyeluruh, mulai dari *deployment* otomatis, *monitoring real-time*, hingga pemulihan jika terjadi gangguan.

Dalam tahap ini, perancang sistem memanfaatkan layanan dari Oracle Cloud Infrastructure (OCI) sebagai platform utama. Beberapa komponen penting yang dirancang antara lain:

- **Virtual Cloud Network (VCN):** Berfungsi sebagai jaringan virtual utama tempat semua *resource* OCI saling terhubung secara aman dan terisolasi dari jaringan lain. VCN menyediakan kontrol penuh atas topologi jaringan, termasuk *IP addressing*, *routing*, dan *security lists*.
- **Subnet:** Digunakan untuk mengelompokkan *resource* berdasarkan fungsi atau kebutuhan akses, misalnya *subnet* publik untuk *gateway* dan *load balancer*, serta *subnet* privat untuk *node* aplikasi (OKE *worker nodes*) dan *database* yang memerlukan isolasi lebih ketat.
- **Internet Gateway:** Digunakan agar *resource* dalam *subnet* publik dapat terhubung ke internet untuk keperluan *update*, akses *image registry* (*Docker Hub*, OCI Container Registry), atau *monitoring* eksternal.
- **Oracle Kubernetes Engine (OKE):** Adalah layanan *managed Kubernetes* dari OCI yang akan menjadi platform utama untuk menjalankan aplikasi kontainer. OKE menyederhanakan manajemen *cluster* Kubernetes, termasuk *provisioning*, *scaling*, dan *upgrades* dari *control plane*.

Selanjutnya, perancangan juga mencakup:

- **Menentukan topologi jaringan:** Seperti bagaimana distribusi *subnet* antar zona ketersediaan (*availability domain*) atau *fault domain* untuk memastikan layanan tetap aktif jika terjadi gangguan di salah satu zona, sehingga meningkatkan ketahanan (*resilience*) sistem.

- **Menentukan jumlah *node* dalam *cluster* Kubernetes:** Sesuai beban kerja yang diperkirakan, dengan mempertimbangkan kebutuhan *CPU*, *memory*, dan *storage* untuk aplikasi yang akan di-*deploy*. Strategi *auto-scaling* juga dipertimbangkan untuk menangani fluktuasi *workload*.
- **Mengatur aspek keamanan:** Seperti penggunaan *Network Security Groups* (NSG) dan *firewall rules* untuk membatasi lalu lintas hanya pada *port* dan protokol yang dibutuhkan, menerapkan *least privilege principle*. Integrasi dengan Oracle Identity and Access Management (IAM) juga dirancang untuk mengelola otentikasi dan otorisasi akses ke *resource* OCI.

Sebagai bagian dari dokumentasi dan komunikasi teknis, tim menyusun diagram arsitektur logis yang menggambarkan bagaimana hubungan antar komponen bekerja, termasuk aliran data dari pengguna ke aplikasi, komunikasi antar *service* (*microservices*), hingga interaksi dengan layanan *monitoring* dan *logging*. Diagram ini berfungsi sebagai cetak biru yang memandu implementasi dan memfasilitasi pemahaman bersama di antara tim.

Fase ini merupakan fondasi penting karena arsitektur yang baik akan menentukan sejauh mana sistem dapat berkembang, mudah dirawat, dan tangguh terhadap perubahan dan gangguan. Sebuah desain yang matang akan mengurangi risiko masalah di kemudian hari dan memastikan skalabilitas yang optimal.

3.2 Fase Infrastructure as Code (IaC) Implementation: Automasi Provisioning Infrastruktur

Setelah desain arsitektur selesai dan disepakati, fase berikutnya adalah membangun infrastruktur secara otomatis menggunakan pendekatan *Infrastructure as Code* (IaC). Dengan IaC, seluruh komponen infrastruktur, mulai dari jaringan, *compute instance*, hingga *cluster* Kubernetes, dituliskan dalam bentuk kode (konfigurasi) yang bisa dijalankan, disimpan, diubah, dan dibagikan, layaknya pengembangan perangkat lunak. Ini menghilangkan kebutuhan untuk *provisioning* manual yang rentan terhadap kesalahan.

Dalam penelitian ini, digunakan dua teknologi utama:

- **Terraform:** Alat otomasi *open-source* yang dikembangkan oleh HashiCorp, digunakan untuk mendeskripsikan dan men-*deploy resource cloud* seperti VCN, *subnet*, *security lists*, dan *cluster* Kubernetes dalam file konfigurasi berformat .tf. Terraform mendukung berbagai penyedia *cloud* (*cloud provider*), termasuk OCI, melalui *provider plugin* spesifik. Keunggulan Terraform adalah kemampuannya untuk mengelola *state* infrastruktur, memastikan *idempotency* (menjalankan skrip berulang kali akan menghasilkan *state* yang sama), dan mendukung *declarative*

syntax yang mudah dibaca.

- **GitHub:** Digunakan sebagai sistem *version control* terpusat, tempat menyimpan semua konfigurasi Terraform. GitHub tidak hanya memungkinkan kolaborasi tim secara sinkron melalui fitur *pull request* dan *branching*, tetapi juga menyediakan riwayat perubahan yang lengkap, sehingga memudahkan *auditing* dan *rollback* ke versi sebelumnya jika diperlukan.

Langkah-langkah yang dilakukan pada fase ini meliputi:

1. **Menulis skrip Terraform:** Membuat file .tf yang mencakup seluruh *resource* OCI yang sudah dirancang sebelumnya, seperti VCN, *subnets*, *Internet Gateway*, *Route Tables*, *Security Lists* atau NSG, dan konfigurasi OKE *cluster* (termasuk *node pools* dan *worker nodes*). Penggunaan modul Terraform dapat meningkatkan *reusability* dan modularitas kode.
2. **Menginisialisasi *project*:** Dengan perintah terraform init. Perintah ini mengunduh *provider* OCI yang diperlukan dan menginisialisasi *backend* untuk menyimpan *state* Terraform.
3. **Melakukan perencanaan *deployment*:** Melalui terraform plan. Perintah ini menganalisis konfigurasi Terraform dan membandingkannya dengan *state* infrastruktur saat ini, kemudian menampilkan daftar perubahan yang akan dilakukan (penambahan, modifikasi, atau penghapusan *resource*). Ini adalah langkah krusial untuk memverifikasi perubahan sebelum diterapkan.
4. **Menjalankan *deployment*:** Dengan terraform apply. Perintah ini mengeksekusi perubahan yang telah direncanakan, sehingga *resource* dibangun secara otomatis di OCI sesuai dengan konfigurasi yang didefinisikan.
5. **Menyimpan file terraform.tfstate:** File ini berisi status terakhir dari infrastruktur yang dikelola oleh Terraform. File ini sangat penting karena Terraform menggunakannya sebagai referensi pada perubahan berikutnya untuk menentukan *delta* yang perlu diterapkan. Dalam lingkungan tim, *state file* ini harus disimpan di lokasi yang aman dan terpusat (misalnya, OCI Object Storage atau Terraform Cloud) untuk menghindari konflik dan memastikan konsistensi.

Keunggulan pendekatan IaC ini adalah efisiensi waktu yang signifikan dalam *provisioning* infrastruktur, konsistensi antar *environment* (misalnya: *development*, *staging*, *production*), serta kemudahan *rollback* dan pemeliharaan jangka panjang. IaC juga memfasilitasi *disaster recovery* karena infrastruktur dapat dibangun ulang dengan cepat dari kode jika terjadi kegagalan sistem.

3.3 Fase CI/CD Pipeline: Implementasi Continuous Integration & Deployment

Tahapan ini merupakan inti dari praktik DevOps, di mana integrasi kode dan proses *deployment* dilakukan secara otomatis dan berkesinambungan. CI/CD (*Continuous*

Integration / Continuous Deployment) adalah serangkaian praktik yang memungkinkan tim *developer* untuk mengembangkan aplikasi dengan lebih cepat, karena setiap perubahan kode dapat langsung diuji dan dideploy tanpa perlu dilakukan secara manual. Ini mengurangi risiko integrasi dan mempercepat *feedback loop*.

Beberapa alat utama yang digunakan dalam fase ini adalah:

- **GitHub Actions:** Digunakan sebagai orkestrator utama untuk memicu *pipeline* secara otomatis ketika ada perubahan pada *repository* kode (misalnya saat *developer* melakukan *push* ke *branch* tertentu atau membuat *pull request*). GitHub Actions menyediakan *workflow engine* yang fleksibel dan terintegrasi langsung dengan GitHub.
- **Jenkins:** Berperan sebagai *server* otomasi yang mengatur seluruh proses *pipeline* CI/CD yang lebih kompleks, termasuk *build*, *test*, dan *deployment*. Jenkins dapat di-*host* di OCI *compute instance* atau di dalam *cluster* OKE itu sendiri. Jenkins menyediakan fleksibilitas yang tinggi dalam mendefinisikan *pipeline* dan integrasi dengan berbagai alat.
- **KubeCTL:** *Command-line tool* resmi Kubernetes yang digunakan untuk berinteraksi dengan *cluster* OKE, mengelola *resource* Kubernetes (seperti *Deployments*, *Services*, *Pods*), dan melakukan *deployment* aplikasi.

Pipeline CI/CD yang dibangun terdiri dari empat tahapan utama:

1. **Build:** Pada tahap ini, *source code* aplikasi yang telah di-*commit* oleh *developer* di-*compile* (jika menggunakan bahasa yang dikompilasi) dan dikemas menjadi *container image* menggunakan Docker. Dockerfile yang mendefinisikan cara *build image* akan digunakan. Hasilnya adalah *Docker image* yang siap untuk dijalankan.
2. **Test:** *Image* yang telah dibangun diuji menggunakan *unit test* atau *integration test* (jika tersedia). Tahap ini memastikan bahwa perubahan kode tidak merusak fungsionalitas yang ada dan memenuhi standar kualitas yang ditetapkan. Jika ada kegagalan tes, *pipeline* akan berhenti dan memberikan *feedback* segera kepada *developer*.
3. **Push:** Jika lolos pengujian, *image* kontainer yang telah divalidasi dikirim ke Oracle Container Registry (OCR) untuk disimpan dan digunakan pada *deployment* berikutnya. OCR adalah *private Docker registry* yang terkelola di OCI, memastikan keamanan dan ketersediaan *image*.
4. **Deploy:** Jenkins menjalankan perintah kubectl apply -f deployment.yaml (atau file konfigurasi Kubernetes lainnya) untuk mendeploy aplikasi ke dalam *cluster* OKE. Ini akan membuat atau memperbarui *resource* Kubernetes yang diperlukan, seperti *Deployment* untuk menjalankan *pod* aplikasi dan *Service* untuk mengekspos aplikasi.

Fase ini menghilangkan hambatan manual dalam proses *delivery* perangkat lunak, secara signifikan mengurangi *human error*, dan menjamin bahwa setiap versi aplikasi yang dideploy sudah melalui proses validasi yang ketat. Ini memungkinkan *developer* untuk fokus pada penulisan kode, sementara otomatisasi menangani proses *delivery* yang kompleks.

3.4 Fase Monitoring dan Observability: Setup Observability Stack

Setelah aplikasi berhasil berjalan di lingkungan Kubernetes, sistem harus dapat dipantau dengan baik untuk menjaga kinerja, mendeteksi masalah sejak dini, dan memahami perilaku sistem secara keseluruhan. Fase ini bertujuan untuk membangun *observability stack*, yaitu sekumpulan alat dan mekanisme yang digunakan untuk melihat "isi" sistem melalui metrik, *log*, dan data performa lainnya. *Observability* lebih dari sekadar *monitoring*; ini adalah kemampuan untuk memahami *state* internal sistem hanya dengan melihat data eksternal yang dihasilkannya.

Dua *tools* utama yang digunakan adalah:

- **Prometheus:** Digunakan sebagai sistem *monitoring* dan *alerting* berbasis *time-series database*. Prometheus mengumpulkan metrik dari berbagai komponen sistem, seperti penggunaan CPU, *memory*, jumlah *request*, *latency*, dan status *pod* Kubernetes. Prometheus menggunakan model *pull* untuk mengumpulkan metrik dari *endpoint* yang terekspos oleh aplikasi atau infrastruktur (melalui *exporters*).
- **Grafana:** Digunakan untuk memvisualisasikan metrik yang dikumpulkan Prometheus ke dalam *dashboard* yang interaktif, mudah dibaca, dan dapat disesuaikan. Grafana mendukung berbagai *data source*, termasuk Prometheus, memungkinkan pembuatan visualisasi yang kaya untuk mendapatkan *insight* operasional.

Langkah-langkah implementasi meliputi:

1. **Deploy Prometheus dan Grafana:** Kedua komponen ini di-*deploy* ke dalam *cluster* Kubernetes sebagai aplikasi kontainer. Biasanya menggunakan *Helm charts* untuk penyebaran yang mudah dan terkonfigurasi dengan baik.
2. **Konfigurasi Prometheus:** Mengatur Prometheus untuk mengenali *pod* dan *service* secara otomatis (*service discovery*) di dalam *cluster* Kubernetes. Ini memungkinkan Prometheus untuk secara dinamis menemukan *endpoint* metrik baru saat aplikasi di-*scale* atau di-*deploy*.
3. **Membuat *dashboard* Grafana:** Mendesain dan membuat *dashboard* Grafana yang menampilkan metrik-metrik penting dari *cluster* Kubernetes (misalnya, kesehatan *node*, penggunaan *resource cluster*), aplikasi (misalnya, *request rate*,

error rate, *latency*), dan komponen infrastruktur lainnya. *Dashboard* ini dirancang untuk memberikan gambaran umum yang cepat tentang kesehatan sistem.

4. **(Opsional) Mengatur notifikasi dan *alert*:** Mengkonfigurasi Prometheus Alertmanager untuk mengirim notifikasi (misalnya, ke Slack, email, PagerDuty) jika terjadi anomali atau kondisi yang mengkhawatirkan, seperti lonjakan CPU mendadak, *memory leak*, *service* yang tidak merespons, atau *pod* yang gagal. Ini memungkinkan tim *ops* untuk bereaksi proaktif terhadap masalah sebelum berdampak pada pengguna.

Fase ini sangat penting karena *observability* adalah kunci dari sistem yang stabil dan mudah ditangani ketika terjadi kegagalan. Dengan *monitoring* yang efektif, tim dapat mengidentifikasi *bottleneck* kinerja, mendiagnosis masalah dengan cepat, dan membuat keputusan berdasarkan data untuk mengoptimalkan sistem. Ini juga mendukung budaya *continuous improvement* dalam DevOps.

3.5 Fase Pengujian Skenario Pemulihan Bencana (Disaster Recovery)

Fase terakhir dari eksperimen ini adalah pengujian skenario pemulihan bencana (*disaster recovery*). Tujuan dari fase ini adalah untuk memvalidasi ketahanan (*resilience*) sistem yang telah dibangun dan memastikan bahwa aplikasi dapat dipulihkan dengan cepat dan minimal *downtime* jika terjadi kegagalan besar, seperti kegagalan zona ketersediaan (*availability domain*) atau *region* OCI. Pengujian ini sangat penting untuk membangun kepercayaan pada arsitektur dan proses DevOps yang telah diimplementasikan.

Langkah-langkah yang dilakukan dalam fase ini meliputi:

1. **Identifikasi Titik Kegagalan Potensial:** Menganalisis arsitektur yang telah dirancang untuk mengidentifikasi komponen-komponen kritis dan titik-titik kegagalan tunggal (*single points of failure*). Ini bisa termasuk kegagalan *node* Kubernetes, *database*, *network connectivity*, atau bahkan seluruh *availability domain*.
2. **Definisi RTO (Recovery Time Objective) dan RPO (Recovery Point Objective):** Menentukan target waktu pemulihan (*RTO*) dan target titik pemulihan (*RPO*). RTO adalah waktu maksimum yang diizinkan agar sistem kembali beroperasi setelah bencana, sedangkan RPO adalah jumlah data maksimum yang dapat hilang selama bencana. Target ini akan memandu strategi pemulihan.
3. **Implementasi Strategi *Backup* dan Replikasi:**
    - **Data Aplikasi:** Memastikan data persisten aplikasi (misalnya, *database* atau *object storage*) di-*backup* secara teratur dan/atau direplikasi ke lokasi geografis yang berbeda (misalnya, *cross-region replication* di OCI Object

Storage).
    - **Konfigurasi Infrastruktur (IaC):** Karena infrastruktur didefinisikan sebagai kode (Terraform), proses pemulihan infrastruktur dapat dilakukan dengan cepat dengan menjalankan kembali skrip Terraform di *region* atau *availability domain* yang berbeda. Ini adalah salah satu keuntungan terbesar dari IaC dalam konteks *disaster recovery*.
    - ***Container Images*:** Memastikan *container images* aplikasi disimpan di Oracle Container Registry yang *highly available* atau direplikasi ke *registry* di *region* lain.
4. **Pengujian Skenario Pemulihan:** Mensimulasikan skenario kegagalan dan menjalankan prosedur pemulihan yang telah direncanakan. Contoh skenario pengujian meliputi:
    - **Kegagalan *Node*:** Menghapus atau menghentikan *worker node* di OKE dan mengamati apakah OKE secara otomatis meluncurkan *node* baru dan memulihkan *pod* aplikasi.
    - **Kegagalan *Availability Domain*:** Mensimulasikan kegagalan seluruh *availability domain* (misalnya, dengan menghentikan semua *resource* di AD tersebut) dan mencoba meluncurkan kembali infrastruktur dan aplikasi di AD yang berbeda atau *region* cadangan.
    - **Kehilangan Data:** Mensimulasikan kehilangan data di *database* dan menguji proses pemulihan dari *backup*.
5. **Dokumentasi dan *Refinement*:** Mendokumentasikan setiap langkah proses pemulihan, termasuk *runbook* dan prosedur operasional standar. Hasil pengujian dianalisis untuk mengidentifikasi *bottleneck* atau area yang perlu ditingkatkan dalam strategi *disaster recovery*. Proses ini bersifat iteratif, di mana pengujian berulang dilakukan untuk terus meningkatkan *resilience* sistem.

Fase ini memastikan bahwa seluruh investasi dalam otomatisasi dan *observability* tidak sia-sia jika terjadi insiden besar. Dengan *disaster recovery* yang teruji, organisasi dapat memiliki kepercayaan diri yang lebih tinggi terhadap ketersediaan dan ketahanan aplikasi mereka di OCI.

KESIMPULAN

Eksperimen ini berhasil mendemonstrasikan implementasi menyeluruh dari siklus hidup DevOps berbasis *cloud-native* menggunakan Oracle Cloud Infrastructure (OCI) sebagai platform utama, didukung oleh alat-alat *open-source* terkemuka seperti Terraform, GitHub Actions, Jenkins, Prometheus, dan Grafana. Setiap fase dari siklus DevOps—mulai dari perencanaan dan desain arsitektur, otomatisasi infrastruktur dengan IaC, pembangunan *pipeline* CI/CD, *monitoring* dan *observability*, hingga

pengujian *disaster recovery*—telah berhasil diintegrasikan dan divalidasi.

Hasil eksperimen menunjukkan peningkatan yang signifikan dalam beberapa aspek kunci:

1. **Efisiensi Operasional:** Penerapan *Infrastructure as Code* (IaC) dengan Terraform secara drastis mengurangi waktu dan upaya manual yang diperlukan untuk *provisioning* dan manajemen infrastruktur. Ini tidak hanya mempercepat proses, tetapi juga meminimalkan *human error* dan memastikan konsistensi lingkungan di seluruh siklus pengembangan.
2. **Kecepatan *Deployment* dan *Time-to-Market*:** *Pipeline* CI/CD yang dibangun menggunakan GitHub Actions dan Jenkins memungkinkan integrasi kode yang berkelanjutan dan *deployment* aplikasi yang otomatis ke Oracle Kubernetes Engine (OKE). Hal ini mempercepat siklus *feedback* kepada *developer* dan memungkinkan organisasi untuk merilis fitur baru atau *bug fixes* dengan frekuensi yang lebih tinggi dan lebih cepat ke pasar.
3. **Peningkatan *Observability* dan Ketahanan Sistem:** Implementasi *monitoring stack* dengan Prometheus dan Grafana menyediakan visibilitas *real-time* terhadap kinerja dan kesehatan aplikasi serta infrastruktur. Kemampuan untuk mendeteksi anomali dan mengkonfigurasi *alert* secara proaktif sangat krusial dalam menjaga stabilitas sistem. Selain itu, pengujian skenario *disaster recovery* memvalidasi kemampuan sistem untuk pulih dari kegagalan besar, menjamin ketersediaan dan kontinuitas bisnis.
4. **Kolaborasi yang Lebih Baik:** Pendekatan DevOps dan penggunaan alat *version control* seperti GitHub mendorong kolaborasi yang lebih erat antara tim *development* dan *operations*, memecah *silo* tradisional dan menciptakan budaya berbagi tanggung jawab.

Secara keseluruhan, eksperimen ini membuktikan bahwa kombinasi OCI dengan alat-alat DevOps *open-source* adalah solusi yang kuat dan efektif untuk membangun sistem yang modern, skalabel, tangguh, dan efisien. Ini memberikan gambaran praktis tentang bagaimana organisasi dapat mengadopsi praktik DevOps untuk mempercepat inovasi, meningkatkan kualitas perangkat lunak, dan mencapai keunggulan kompetitif dalam lanskap digital yang terus berkembang.

Saran dan Pekerjaan Masa Depan:

Untuk pengembangan lebih lanjut, penelitian ini dapat diperluas dengan:

- **Integrasi Keamanan (DevSecOps):** Menambahkan praktik keamanan secara eksplisit ke dalam setiap tahap *pipeline* CI/CD, seperti pemindaian kerentanan kode (*SAST/DAST*), *image scanning*, dan manajemen rahasia (*secrets*

management).

- ***Cost Optimization*:** Menganalisis dan mengimplementasikan strategi optimasi biaya di OCI, seperti penggunaan *compute instance* dengan harga spot atau *auto-scaling* yang lebih cerdas.
- ***Advanced Logging and Tracing*:** Mengintegrasikan solusi *logging* terpusat (misalnya, Fluentd dengan Loki atau ELK Stack) dan *distributed tracing* (misalnya, Jaeger) untuk *troubleshooting* yang lebih mendalam pada aplikasi *microservices*.
- **AIOps:** Mengeksplorasi penerapan *Artificial Intelligence* (AI) dan *Machine Learning* (ML) untuk otomatisasi *monitoring*, deteksi anomali prediktif, dan respons insiden.
- **Multi-Cloud/Hybrid Cloud Strategy:** Menguji bagaimana arsitektur ini dapat diperluas atau diadaptasi untuk skenario *multi-cloud* atau *hybrid cloud*, memanfaatkan keunggulan dari berbagai penyedia *cloud*.