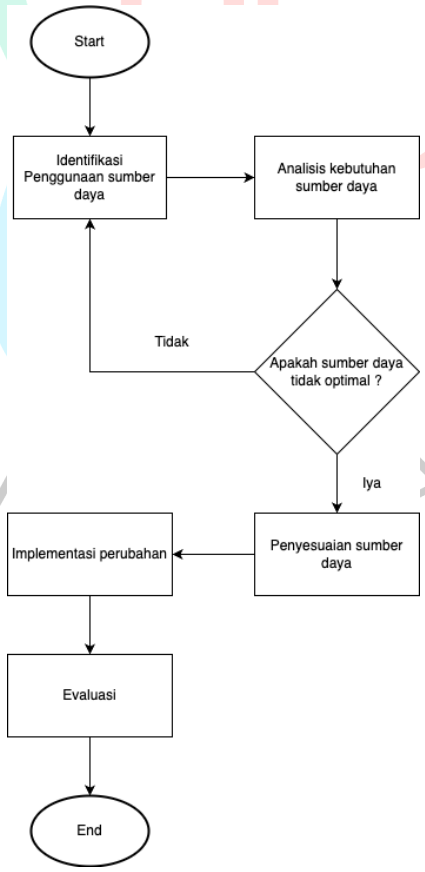
BAB IV

PERANCANGAN

BAB ini merinci rancangan sistem automasi *DevOps* yang dirancang untuk mengidentifikasi dan mengoptimalkan sumber daya cloud yang tidak digunakan atau tidak efisien. Ini akan dilakukan melalui analisis sistem terdahulu, spesifikasi kebutuhan sistem baru, dan perancangan sistem tersebut. Sistem baru yang akan dikembangkan bertujuan untuk mengotomatisasi deteksi dan optimasi sumber daya cloud yang tidak digunakan atau tidak efisien di PT. XYZ, dengan integrasi *DevOps* yang melibatkan GitLab.

4.1 Analisis Sistem Terdahulu

Sistem yang ada sebelumnya di PT. XYZ melibatkan pemantauan dan pengelolaan sumber daya cloud secara manual. Analisis ini mengungkap bahwa proses manual tersebut tidak hanya memakan waktu dan sumber daya manusia yang signifikan tetapi juga sering kali menghasilkan identifikasi yang lambat terhadap sumber daya yang tidak optimal, menyebabkan penggunaan biaya cloud yang lebih tinggi dari yang seharusnya. Kelemahan utama dari sistem ini adalah kurangnya visibilitas atas penggunaan sumber daya secara real-time dan keterbatasan dalam respons otomatis terhadap perubahan kebutuhan sumber daya.



Gambar 4.1 Flowchart Sistem Terdahulu

Pada gambar 4.1, dapat dilihat alur sistem yang ada sebelumnya di PT. XYZ. Di tahap pertama, dilakukan identifikasi penggunaan sumber daya, di mana data dikumpulkan untuk menilai penggunaan cloud. Kemudian dilakukan analisis kebutuhan sumber daya untuk mengevaluasi apakah kapasitas yang tersedia sudah sesuai dengan permintaan operasional. Jika hasil analisis menunjukkan bahwa sumber daya tidak optimal, maka langkah selanjutnya adalah melakukan penyesuaian untuk menyesuaikan penggunaan sumber daya dengan kebutuhan yang ada. Proses ini dilakukan secara manual dan memerlukan banyak waktu untuk memonitor dan menyesuaikan, yang mengarah pada pemborosan biaya dan ketidakmampuan untuk merespons secara cepat terhadap perubahan beban kerja.

4.2 Spesifikasi Kebutuhan Sistem Baru

Setelah melakukan analisis terhadap sistem yang ada, didapatkan hasil yang mencakup spesifikasi yang diperlukan untuk sistem baru, yaitu sistem automasi DevOps yang berfokus pada optimalisasi biaya cloud di *e-commerce*. Untuk membangun sistem ini, diperlukan spesifikasi yang meliputi perangkat keras, perangkat lunak, serta kebutuhan *input* dan *output* yang akan diproses oleh sistem. Selain itu, perancangan model untuk *machine learning* dan perancangan DevOps juga merupakan bagian integral dalam membangun sistem ini untuk memastikan optimasi yang lebih efisien dan otomatis.

4.2.1 Spesifikasi Perangkat Keras

Dalam pengembangan *system* aplikasi, perangkat keras yang digunakan untuk menjalankan sistem ini harus sesuai dengan kebutuhan pemrosesan data dan kapasitas untuk mendukung alur kerja sistem yang efisien. Berikut adalah detail spesifikasi perangkat keras yang diperlukan yang tercantum pada Tabel 4.1

Tabel 4.1 Spesifikasi Perangkat Keras

Perangkat	Konfigurasi
Prosesor	Amd64 2 vCPU
Memori	8 GB
Sistem Operasi	Linux x86-64
Tipe Sistem	64-bit

4.2.2 Spesifikasi Perangkat Lunak

Untuk mendukung pengembangan sistem ini, sejumlah perangkat lunak dan alat pendukung diperlukan. Berikut adalah spesifikasi perangkat lunak yang digunakan dalam pengembangan aplikasi pada Tabel 4.2

Tabel 4.2 Spesifikasi Perangkat Lunak

Perangkat	Deskripsi
Bahasa Pemograman	Python
Browser	Mozilla Firefox
Library	*Google APIs*, *pandas*, *python-dotenv*, *oauth2client*, *db-types* dan *Jinja2*
Tools	Visual Code Studio
CI/CD	Gitlab
Alat Monitoring	Datadog

4.2.3 Spesifikasi Kebutuhan Input

Data yang digunakan dalam penelitian ini diambil dari beberapa sumber yang menyediakan metrik terkait penggunaan sumber daya *cloud*, baik untuk VM di GCP maupun untuk analisis optimasi sumber daya. Sumber data yang digunakan adalah sebagai berikut:

1. Datadog
   Data utama mengenai penggunaan CPU, memori, serta metrik terkait permintaan baca/tulis pada VM yang berfungsi sebagai node Cassandra dikumpulkan melalui platform pemantauan Datadog. Pemantauan ini berlangsung selama periode 10 bulan, mulai dari Januari hingga Oktober 2024. Fokus utama pada metrik CPU dan memori memungkinkan penelitian ini untuk mendeteksi penggunaan sumber daya yang tidak efisien, sehingga dapat memberikan rekomendasi perubahan *machine type* atau alokasi sumber daya pada VM untuk mengoptimalkan biaya operasional dan performa sistem.
2. *BigQuery*
   BigQuery digunakan untuk mendeteksi *project* di GCP yang hanya mengaktifkan API monitoring tanpa ada layanan lain yang aktif. Selain itu, BigQuery membantu mengidentifikasi *snapshot* yang sudah lama tidak digunakan dan IP address yang tidak terpakai. Data dari BigQuery ini meliputi informasi terkait pemakaian penyimpanan, biaya operasional, dan penggunaan komponen cloud lainnya yang tidak efisien, sehingga memudahkan proses identifikasi sumber daya yang perlu dioptimalkan.

4.2.4 Spesifikasi Kebutuhan Output

Output yang dihasilkan oleh sistem ini berupa laporan yang diintegrasikan dengan *Google Sheets*. Laporan ini akan berisi informasi tentang sumber daya cloud yang diidentifikasi sebagai tidak terpakai atau tidak efisien, serta rekomendasi optimasi yang diperlukan untuk menyesuaikan alokasi sumber daya dan mengurangi biaya operasional. Laporan ini menjadi dasar bagi pengambilan keputusan dalam mengoptimalkan sumber daya di PT. XYZ.

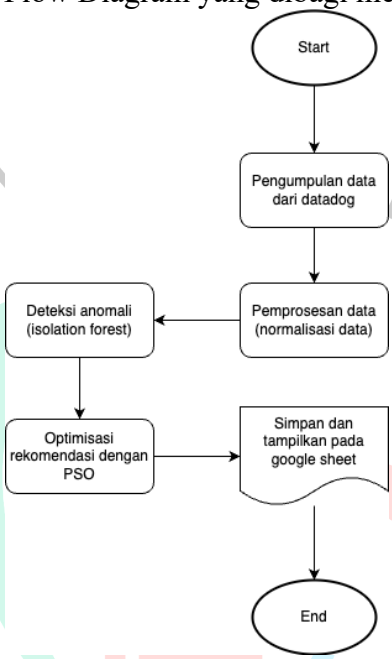
4.3 Perancangan Sistem

Untuk mengembangkan sistem automasi DevOps yang mampu mendeteksi dan mengoptimalkan penggunaan sumber daya cloud yang tidak efisien atau tidak terpakai, peneliti merancang sebuah sistem yang komprehensif. Rancangan ini menggambarkan cara kerja sistem secara keseluruhan sehingga proses pengembangan dapat dilakukan dengan lebih rinci dan efisien.

Sistem ini didesain untuk melakukan beberapa fungsi utama, seperti deteksi VM dengan jenis *machine type* yang tidak sesuai, deteksi snapshot lama yang tidak diperlukan, serta deteksi project GCP yang hanya mengaktifkan API monitoring tanpa layanan lain. Selain itu, sistem juga mendeteksi IP address dan disk yang tidak terpakai. Untuk memberikan rekomendasi optimasi, sistem akan memanfaatkan algoritma seperti PSO untuk mengoptimalkan *machine type* berdasarkan pola penggunaan CPU dan memori yang terdeteksi.

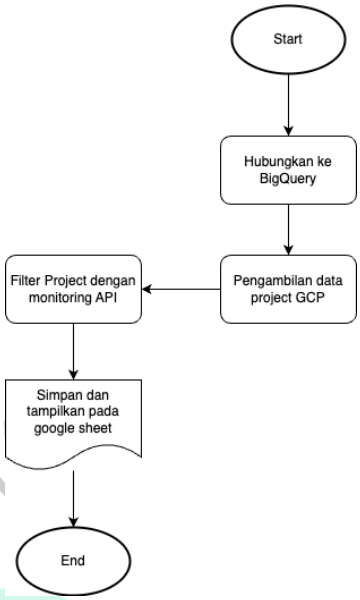
Rancangan sistem ini menggunakan *Flow Diagram* untuk memvisualisasikan setiap alur kerja dalam sistem. Diagram ini membantu peneliti dan pengembang untuk memetakan keseluruhan proses, mulai dari pengumpulan data melalui GCP Monitoring dan BigQuery, analisis, hingga penyajian hasil di *Google Sheets*. Dengan menggunakan *Flow Diagram*, alur kerja sistem dapat tergambar lebih sederhana dan jelas, tanpa memerlukan detail teknis mendalam seperti pada diagram *Use Case*, *Activity*, atau *Sequence*.

Berikut adalah proses Flow Diagram yang dibagi menjadi beberapa bagian



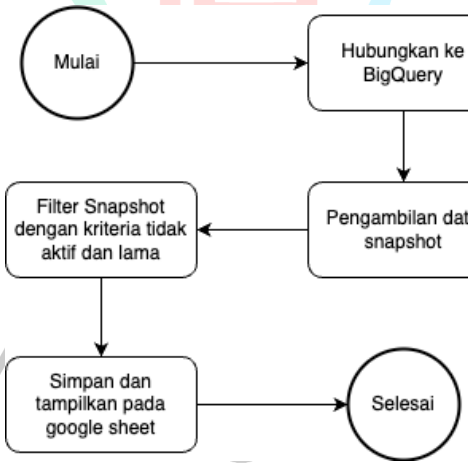
Gambar 4.2 Process Flow Diagram Rekomendasi VM

Gambar 4.2 adalah *Flow Diagram* untuk proses rekomendasi VM berdasarkan penggunaan CPU dan memori. Proses dimulai ketika pengguna menjalankan pipeline DevOps dengan memasukkan variabel yang diperlukan. Sistem kemudian mengumpulkan data metrik CPU dan memori dari Datadog, melakukan preprocessing data dengan normalisasi dan agregasi mingguan, dan melatih model deteksi anomali menggunakan *Isolation Forest*. Hasil deteksi tersebut dioptimalkan menggunakan *PSO* untuk menghasilkan rekomendasi tipe VM yang lebih efisien. Terakhir, hasil rekomendasi disimpan ke Google Sheets agar dapat diakses oleh tim DevOps.



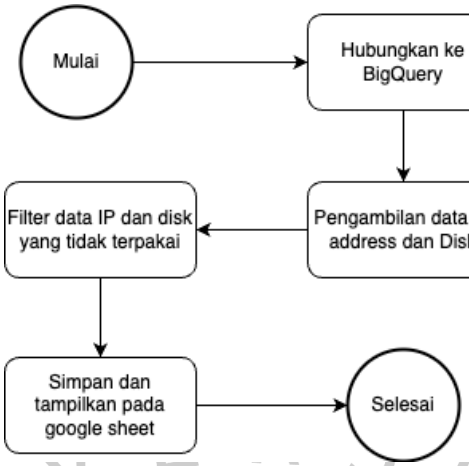
Gambar 4.3 Process Flow Diagram mendeteksi project GCP

Gambar 4.3 adalah *Flow Diagram* untuk mendeteksi project GCP yang hanya mengaktifkan API monitoring tanpa layanan lain. Proses dimulai ketika pengguna menjalankan pipeline DevOps. Sistem terhubung ke BigQuery untuk mengumpulkan data proyek, kemudian melakukan query untuk menemukan proyek-project yang hanya memiliki API monitoring aktif tanpa layanan lain. Hasil deteksi disimpan ke Google Sheets untuk memudahkan pemantauan dan tindak lanjut oleh tim.



Gambar 4.4 Process flowchart identifikasi snapshot

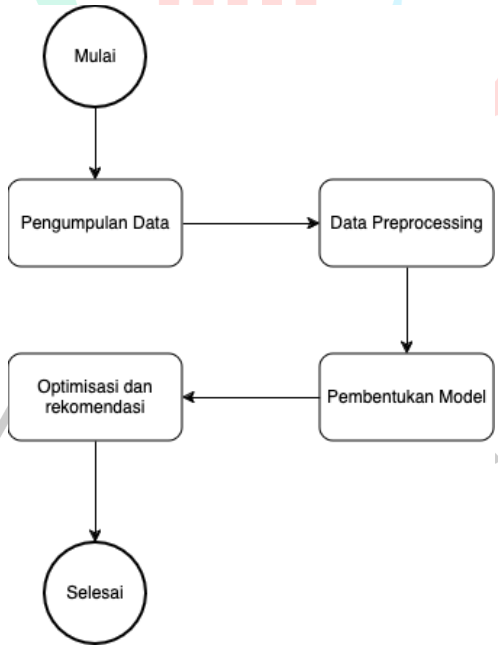
Gambar 4.4 adalah *Flow Diagram* untuk proses identifikasi snapshot lama yang tidak diperlukan untuk mengoptimalkan penggunaan penyimpanan. Proses ini dimulai dengan pengguna yang menjalankan pipeline di DevOps, kemudian sistem terhubung ke *BigQuery* untuk mengambil *data snapshot*. Selanjutnya, sistem memfilter *snapshot* yang sudah tidak aktif atau lama, dan menyimpan daftar *snapshot* ini ke *Google Sheets* untuk memudahkan penghapusan.



Gambar 4.5 Process flowdiagram deteksi IP address and disk

Gambar 4.5 adalah *Flow Diagram* untuk mendeteksi IP address dan disk yang tidak terpakai, sehingga dapat mengurangi biaya operasional cloud. Pengguna menjalankan pipeline melalui DevOps, kemudian sistem terhubung ke BigQuery untuk mengumpulkan data IP address dan disk yang tidak digunakan. Setelah data dikumpulkan dan dianalisis untuk aktivitas, sistem menyimpan daftar resource yang tidak terpakai ke Google Sheets sebagai rekomendasi penghapusan atau pengoptimalan.

4.3.1 Perancangan Model Machine Learning



Gambar 4.6 Flow Chart Pembuatan Model

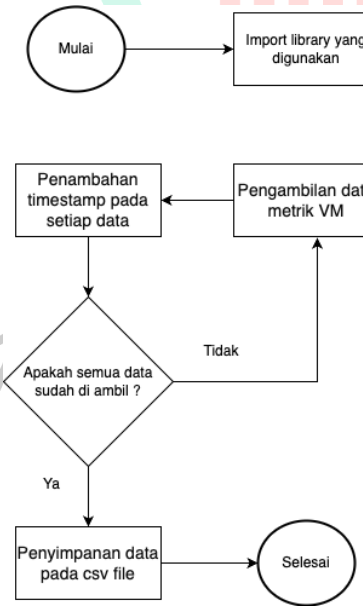
Gambar 4.6 menunjukkan flowchart perancangan model yang diusulkan untuk sistem optimasi sumber daya cloud menggunakan algoritma machine learning dan DevOps, sebagaimana yang telah dijelaskan pada Bab 2. Proses ini dimulai dengan *pengumpulan data*, dilanjutkan dengan tahapan *data preprocessing*, deteksi anomali, pembentukan model, evaluasi model, hingga optimasi dan pembuatan rekomendasi.

Data yang digunakan dalam penelitian ini dikumpulkan dari platform pemantauan berbasis cloud seperti *Datadog*, yang memuat metrik penggunaan CPU, memori, dan permintaan baca/tulis pada VM Cassandra. Data yang terkumpul kemudian diproses melalui tahapan *preprocessing* untuk memastikan konsistensi dan kesiapan data, termasuk normalisasi dan perhitungan *Average Predicted Usage (APU)* sebagai indikator kinerja.

Setelah proses *data cleansing* dan normalisasi selesai, data yang sudah diproses akan dianalisis menggunakan algoritma *Isolation Forest* untuk mendeteksi anomali dalam pola penggunaan sumber daya. Langkah ini bertujuan untuk mengidentifikasi pola penggunaan yang tidak efisien, seperti penggunaan sumber daya yang berlebihan atau kurang optimal.

Setelah anomali berhasil dideteksi, proses dilanjutkan ke tahap *pembentukan model* menggunakan algoritma *Boosted Decision Tree*. Algoritma ini dilatih untuk mengenali pola data guna merekomendasikan tipe VM yang optimal berdasarkan data penggunaan CPU dan memori. Model yang terbentuk kemudian dievaluasi dengan metrik *Mean Squared Error (MSE)*, yang membantu mengukur akurasi model terhadap data aktual dan mengidentifikasi potensi kesalahan dalam prediksi.

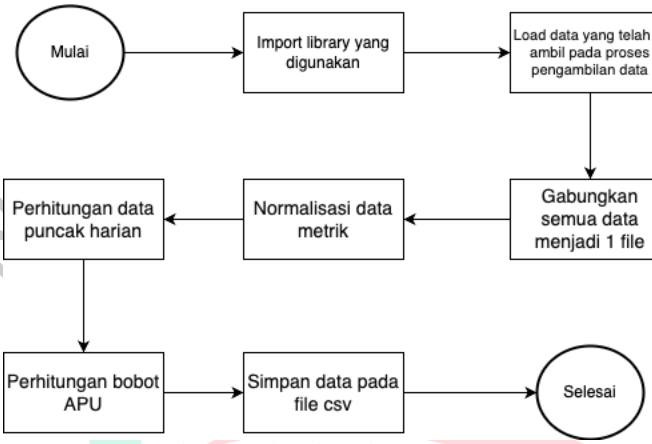
Tahap terakhir adalah optimasi model *menggunakan PSO (Particle Swarm Optimization)*, yang bertujuan untuk menyempurnakan rekomendasi berdasarkan data penggunaan sumber daya yang dinamis dan kondisi lingkungan yang berubah. Melalui proses optimasi ini, sistem dapat memberikan rekomendasi tipe VM yang paling efisien dan sesuai dengan kebutuhan operasional VM.



Gambar 4.7 Flowchart pengumpulan data

Gambar 4.7 Flowchart Pengumpulan Data menunjukkan tahapan pengumpulan data dalam penelitian ini. Proses dimulai dengan mengimpor *library* yang diperlukan, seperti *datetime*, *timedelta*, *re*, *logging*, *csv*, *os*, dan *datadog*, yang mendukung pengambilan, pemrosesan, dan penyimpanan data. Setelah itu, *API Datadog* diinisialisasi dengan API key yang telah dikonfigurasi, yang memungkinkan akses ke *Datadog* untuk memperoleh data metrik yang dibutuhkan.

Rentang waktu untuk pengambilan data kemudian ditentukan menggunakan *datetime* dan *timedelta*, memastikan data yang diambil berada dalam periode yang jelas. Setelah rentang waktu ditentukan, permintaan data dikirim ke *API Datadog* untuk mengambil metrik penggunaan CPU dan memori. Proses ini dilakukan berulang kali hingga seluruh data yang dibutuhkan berhasil diambil. Kemudian diberi timestamp untuk mempermudah analisis dan pelacakan. Pada tahap akhir, data yang telah bersih disimpan dalam file *CSV* di direktori yang telah ditentukan, sehingga siap digunakan untuk tahap analisis atau diproses oleh komponen lainnya dalam alur penelitian.



Gambar 4.8 Flowchart data preprocessing

Gambar 4.8 menunjukkan alur proses data preprocessing dalam penelitian ini. Tahapan ini dimulai dengan mengimpor library penting seperti *pandas*, *numpy*, dan *glob* yang berfungsi untuk mempermudah pengolahan dan manipulasi data. Selanjutnya, data file CSV yang diambil dari proses pengambilan data yang berisi metrik-metrik utama, yaitu *cpu_usage*, *memory_usage*, *cassandra_read_request*, dan *cassandra_write_request*, digabungkan ke dalam satu file harian untuk mempermudah langkah-langkah analisis selanjutnya.

Pada tahap berikutnya, dilakukan normalisasi untuk menyesuaikan skala data. Normalisasi *CPU* dan *memori* menggunakan formula yang lebih spesifik. *CPU Usage* dinormalisasi dengan rumus

$$norm\_cpu\_usage = 100 - cpu\_usage$$

yang memberikan informasi kapasitas *CPU* yang terpakai dalam persentase

Memory Usage dinormalisasi dengan rumus

$$(1 - memory\_usage) \times 100$$

yang menampilkan kapasitas *memory* terpakai. Sementara itu, untuk permintaan read dan write dari Cassandra, dilakukan transformasi menggunakan log-normalisasi untuk mengatasi variasi nilai yang tinggi pada data. Langkah ini bertujuan agar skala penggunaan sumber daya lebih seragam untuk analisis lebih lanjut. Selanjutnya tahap untuk menghitung nilai *peak usage* atau puncak penggunaan harian untuk setiap metrik, dengan mengambil nilai tertinggi pada data harian. Selanjutnya menghitung nilai *APU* yang dihasilkan dari metrik rata-rata harian. PT. XYZ telah menetapkan bobot khusus untuk setiap metrik dalam penilaian *APU*

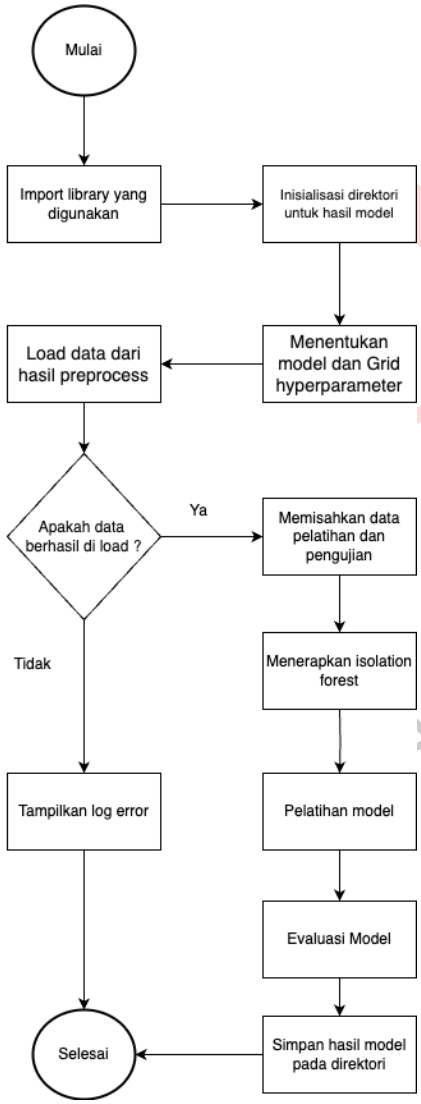
Tabel 4 3 Tabel bobot APU

Metriks	**Bobot**
CPU Usage	48%
Memory Usage	48%
Wrire Request	2 %
Read Request	2 %

Rumus perhitungan APU adalah:

$$\text{APU} = 0.48 \times \text{norm\_average\_cpu\_usage} + 0.48 \times \text{norm\_average\_memory\_usage} + 0.02 \times \text{norm\_average\_write\_request} + 0.02 \times \text{norm\_average\_read\_request}$$

Nilai *APU* ini kemudian disimpan untuk digunakan dalam proses analisis dan optimasi lebih lanjut.



Gambar 4.9 Flowchart Pembentukan Model

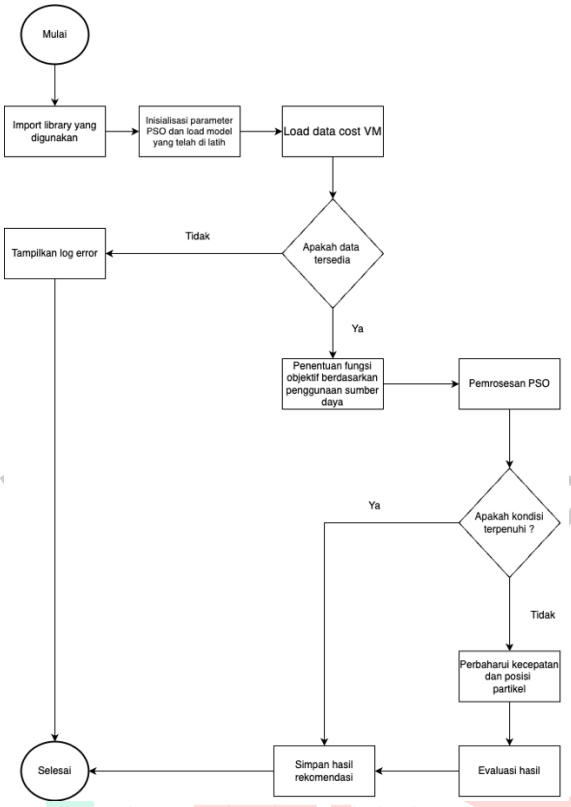
Gambar 4.9 merupakan flowchart yang menggambarkan alur pembentukan model dalam penelitian ini. Proses dimulai dengan mengimpor library yang diperlukan, seperti *pandas*, *json*, *logging*, dan *sklearn*. Library ini mendukung berbagai fungsi, seperti manipulasi data, penyimpanan hasil dalam format *JSON*, logging untuk pencatatan proses, dan implementasi model *Machine Learning*. *Library* tambahan seperti *tqdm* juga digunakan untuk menampilkan progress bar selama proses iterasi *hyperparameter*.

Setelah tahap *import library* selesai, proses dilanjutkan dengan inisialisasi direktori yang akan digunakan untuk menyimpan model, *hyperparameter*, serta hasil performa model. Langkah inisialisasi direktori ini bertujuan untuk menyiapkan ruang penyimpanan terstruktur, sehingga setiap artefak yang dihasilkan dari pembentukan model dapat dikelola dan diakses dengan mudah selama proses pengembangan maupun analisis lebih lanjut.

Langkah berikutnya adalah mendefinisikan model utama, yaitu *Boosted Decision Tree*, dan menentukan grid hyperparameter yang akan dievaluasi. Model ini dipilih untuk memprediksi *APU*, yang kemudian akan menjadi dasar dalam menentukan rekomendasi optimal untuk tipe VM. Pengaturan *grid hyperparameter* meliputi parameter seperti jumlah *n_estimators*, *learning_rate*, *max_leaf_nodes*, dan *min_samples_leaf*. Setiap kombinasi dari hyperparameter ini akan dievaluasi selama pelatihan untuk menemukan konfigurasi terbaik yang menghasilkan nilai *MSE* terendah.

Selanjutnya, data yang telah diproses pada tahap data preprocessing akan dimuat untuk digunakan dalam proses pelatihan model. Jika pemuatan data berhasil, data tersebut dibagi menjadi dua bagian: set pelatihan dan set pengujian. Set pelatihan digunakan untuk melatih model, sedangkan set pengujian digunakan untuk mengukur performa model pada data yang belum pernah dilihat sebelumnya. Pada tahap ini, *Isolation Forest* diterapkan untuk mendeteksi dan menghilangkan *outliers*, guna memastikan bahwa data yang digunakan adalah representatif dan bebas dari anomali yang dapat mempengaruhi hasil pembentukan model.

Setelah persiapan data selesai, proses pelatihan model dilakukan menggunakan *Boosted Decision Tree* pada set pelatihan, dengan mencoba setiap kombinasi hyperparameter yang tersedia dalam grid. Untuk setiap kombinasi, dilakukan evaluasi menggunakan *cross-validation* untuk menghitung nilai *MSE*. Proses *cross-validation* ini dilakukan berulang kali hingga seluruh kombinasi hyperparameter dievaluasi, dan model dengan *MSE* terendah dipilih sebagai model terbaik.



Gambar 4.10 Flowchart Optimization PSO

Gambar 4.10 menunjukkan flowchart yang mendeskripsikan alur optimisasi menggunakan *Particle Swarm Optimization* (*PSO*) dalam penelitian ini. Proses dimulai dengan mengimpor library yang diperlukan, seperti *pso*, *pandas*, *joblib*, dan *logging*, yang mendukung pelaksanaan *PSO* serta pemuatan model yang telah dilatih.

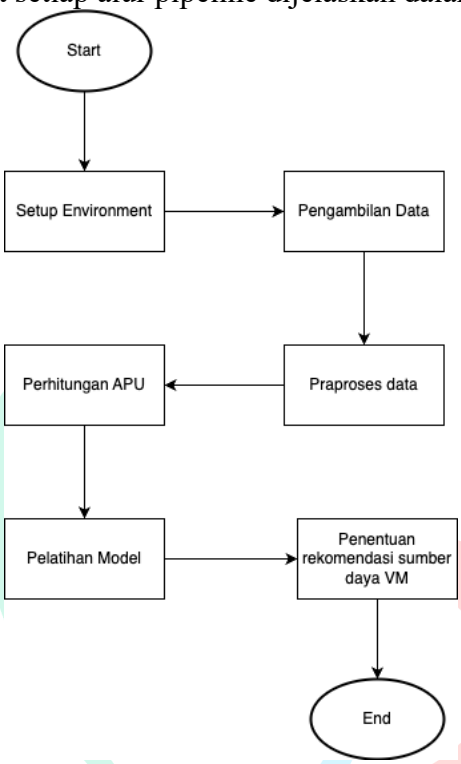
Setelah library di import, parameter *PSO* diinisialisasi. Parameter ini mencakup jumlah partikel, kecepatan awal, dan batas iterasi, yang sangat penting untuk mengatur perilaku partikel selama proses optimisasi. Tahap selanjutnya adalah mendefinisikan *fungsi objektif*, yang dibangun berdasarkan output dari model yang telah dilatih. Fungsi objektif ini dirancang untuk menentukan konfigurasi *VM* yang optimal dengan mempertimbangkan efisiensi penggunaan sumber daya.

Setiap partikel dalam swarm kemudian diperbarui dalam hal kecepatan dan posisi berdasarkan solusi optimal terbaik yang ditemukan, baik pada tingkat individu (*personal best*) maupun pada tingkat swarm secara keseluruhan (*global best*). Pada setiap iterasi, *fitness* dari setiap partikel dihitung menggunakan fungsi objektif. Jika solusi yang ditemukan lebih optimal dari solusi sebelumnya, maka *solusi terbaik personal* dan *global* akan diperbarui.

Setelah evaluasi, sistem melakukan pencarian tipe *machine* yang paling mendekati solusi optimal yang ditemukan. Proses ini bertujuan untuk memastikan bahwa rekomendasi yang diberikan bisa diimplementasikan sesuai dengan ketersediaan tipe *machine* yang ada. Jika solusi terbaik ditemukan, proses akan diakhiri. Namun, jika kriteria terminasi, seperti jumlah iterasi atau ambang perbedaan solusi, belum tercapai, proses akan berlanjut hingga kriteria tersebut terpenuhi.

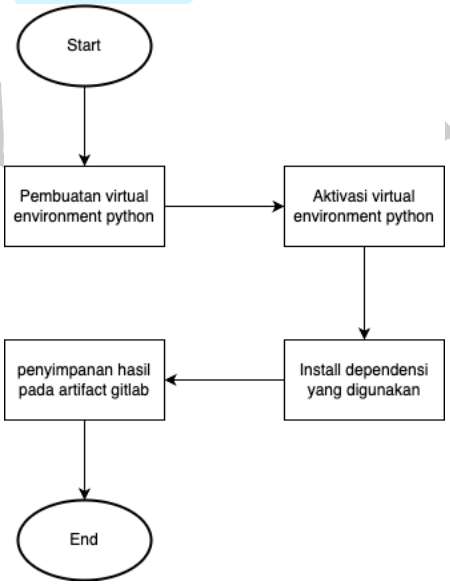
4.3.2 Perancangan *DevOps*

Perancangan DevOps dalam penelitian ini dibagi menjadi dua bagian utama, yaitu pipeline untuk *Machine Learning* dan pipeline untuk analisis sumber daya. Pipeline ini dirancang dengan struktur yang mencakup semua langkah utama mulai dari persiapan lingkungan, pengumpulan data, hingga pengoptimalan sumber daya. Penjelasan detail terkait setiap alur pipeline dijelaskan dalam gambar di bawah ini.



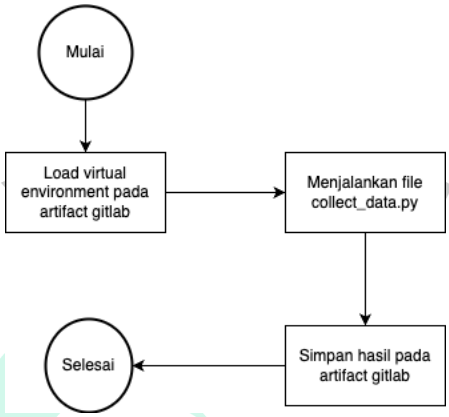
Gambar 4.11 Diagram alir Machine learning

Gambar 4.11 menunjukkan alur pipeline untuk *Machine Learning* dalam penelitian ini. Pipeline ini mencakup beberapa tahap utama untuk mengotomatisasi proses analisis dan pengoptimalan sumber daya. Berikut adalah penjelasan per stage dalam pipeline ini:



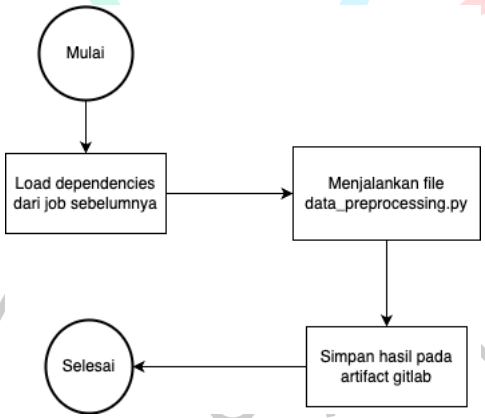
Gambar 4.12 diagram alir stage setup

Gambar 4.12 menjelaskan pada stage setup, terdapat job *setup environment* yang dirancang untuk menyiapkan *environment Python* yang akan digunakan oleh stage lainnya dalam pipeline *Machine learning*. Job ini terdiri dari beberapa langkah, yaitu pembuatan *virtual environment*, aktivasi *virtual environment*, dan instalasi dependensi yang terdapat pada file *requirements.txt*. Hasil dari job ini disimpan sebagai *artifact* dalam direktori *venv*, yang kemudian digunakan oleh job lain pada pipeline.



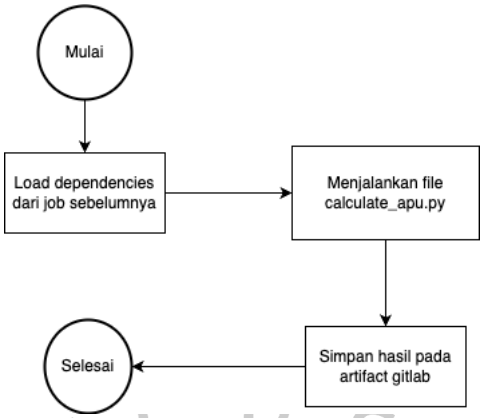
Gambar 4.13 diagram alir stage collect data

Gambar 4.13 menjelaskan pada stage *collect_data*, terdapat job *collect data* yang dirancang untuk mengumpulkan data terkait penggunaan sumber daya dari platform monitoring, seperti CPU, memori, disk, dan koneksi. Job ini menjalankan script *collect_data.py* untuk menghasilkan data mentah yang disimpan dalam direktori output (*$OUTPUT_DIR*). Data yang dihasilkan pada job ini akan digunakan sebagai input pada stage berikutnya dalam pipeline *Machine learning*.



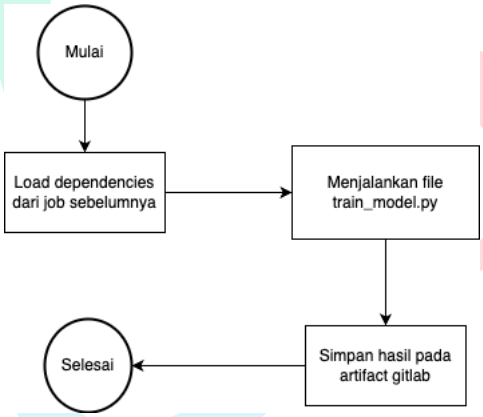
Gambar 4.14 flowchart stage preprocess data

Gambar 4.14 menjelaskan pada stage *preprocess_data*, terdapat job *preprocess data* yang dirancang untuk menjalankan proses preprocessing data. Job ini memiliki *dependencies* dari job sebelumnya, yaitu *setup-job* dan *collect_data*. Pada job ini, *virtual environment* diaktifkan, dan script *data_preprocessing.py* dijalankan dengan parameter input dan output. Hasil dari job ini disimpan sebagai *artifact* dalam direktori *$OUTPUT_DIR*, yang akan digunakan oleh stage berikutnya.



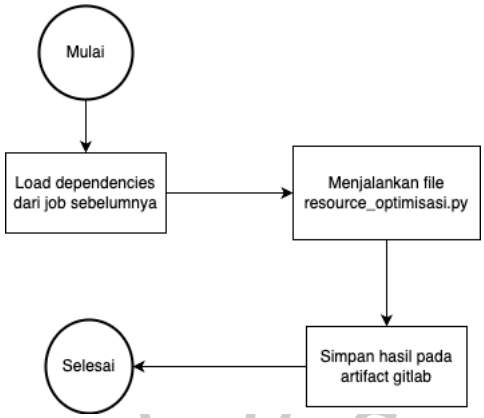
Gambar 4.15 flowchart stage calculate apu

Gambar 4.15 menjelaskan pada stage *calculate_apu*, terdapat job *calculate APU* yang dirancang untuk menghitung metrik APU dari data yang telah diproses pada stage sebelumnya. Job ini memiliki *dependencies* dari *setup-job* dan *preprocess_data*. Pada job ini, *virtual environment* diaktifkan, dan script *calculate_apu.py* dijalankan. Hasil dari perhitungan disimpan sebagai *artifact* dalam direktori *$VISUALIZATION_DIR*, yang kemudian digunakan pada stage berikutnya.



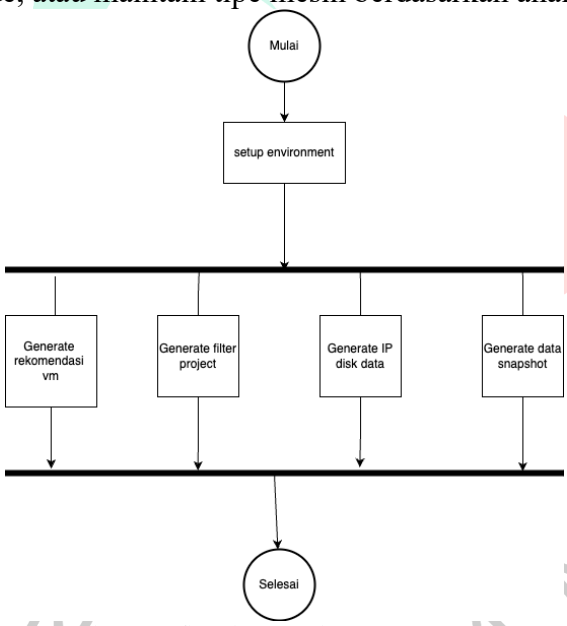
Gambar 4.16 flowchart stage train model

Gambar 4.16 menjelaskan pada stage *train_model*, terdapat job *train model* yang dirancang untuk melatih model *Machine Learning* berdasarkan data APU yang telah dihitung pada stage sebelumnya. Job ini memiliki *dependencies* dari *setup-job* dan *calculate_apu*. *Virtual environment* diaktifkan, dan script *train_model.py* dijalankan dengan data input dan parameter yang telah ditentukan. Hasil dari pelatihan model disimpan sebagai *artifact* dalam direktori *$OUTPUT_DIR*, yang akan digunakan oleh stage selanjutnya untuk optimasi sumber daya.



Gambar 4.17 flowchart stage optimize resources

Gambar 4.17 menjelaskan pada stage *optimize_resources*, terdapat job *resource optimization* yang dirancang untuk menjalankan optimasi sumber daya berdasarkan model yang telah dilatih. Job ini memiliki *dependencies* dari *setup-job* dan *train_model*. Setelah *virtual environment* diaktifkan, script *resources_optimize.py* akan dijalankan. Hasil dari optimasi memberikan rekomendasi tindakan seperti upgrade, downgrade, atau maintain tipe mesin berdasarkan analisis sumber daya.

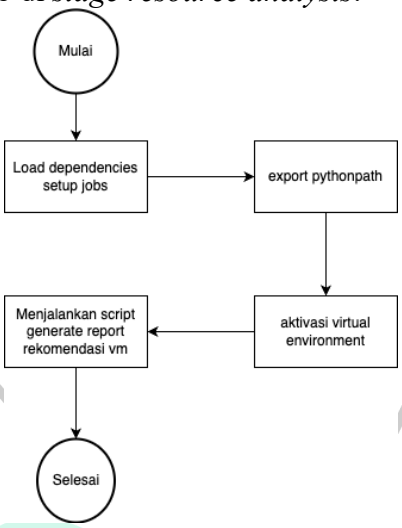


Gambar 4.18 flowchart pipeline resource analysis

Gambar 4.18 menjelaskan perancangan *pipeline* untuk *resources analysis*, yang terdiri dari beberapa tahapan utama untuk menganalisis dan mengoptimalkan penggunaan sumber daya cloud. *Pipeline* ini dimulai dari *stage setup*, yang bertugas mempersiapkan *environment* yang akan digunakan oleh seluruh job pada *stage* berikutnya. Setelah tahap *setup* selesai, *pipeline* berlanjut ke *stage resource analysis*, yang terdiri dari beberapa job untuk menjalankan analisis secara paralel. Berikut adalah penjelasan per stage dalam pipeline ini:

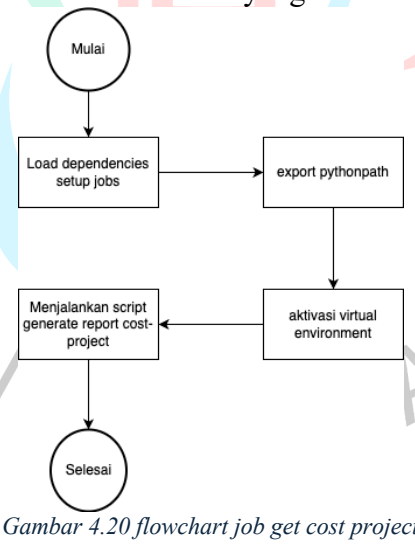
Pada *stage setup*, sama seperti yang telah dijelaskan pada Gambar 4.12, terdapat job *setup environment*, yang dirancang untuk menyiapkan *environment* Python yang akan digunakan oleh semua job di *stage* berikutnya. Job ini terdiri dari beberapa langkah utama, yaitu pembuatan *virtual environment*, aktivasi *virtual environment*, dan instalasi dependensi yang terdapat pada file *requirements.txt*.

Hasil dari job ini disimpan sebagai *artifact* di direktori *venv*, yang kemudian digunakan pada semua job di *stage resource analysis*.



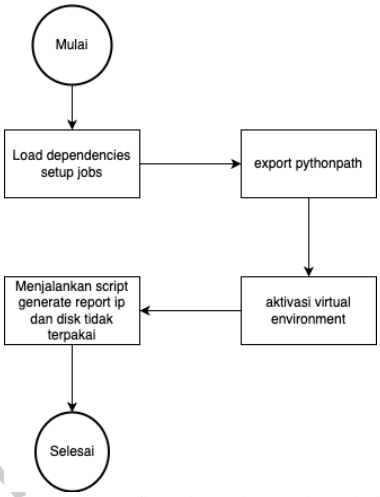
Gambar 4.19 flowchart job rekomendasi vm

Gambar 4.19 menjelaskan pada *stage resource analysis*, terdapat job *recommendation-vm*, yang bertugas menjalankan program *Machine learning* untuk memberikan rekomendasi tipe mesin virtual berdasarkan data penggunaan sumber daya. Job ini memiliki *dependencies* dari *setup-job* dan dijalankan hanya jika *commit branch* adalah *main*, sesuai dengan validasi yang didefinisikan dalam *pipeline CI/CD*. Pada job ini, *script resources_optimize.py* dijalankan menggunakan *environment* Python yang telah disiapkan sebelumnya, dan hasil rekomendasi dihasilkan berdasarkan model yang telah dilatih.



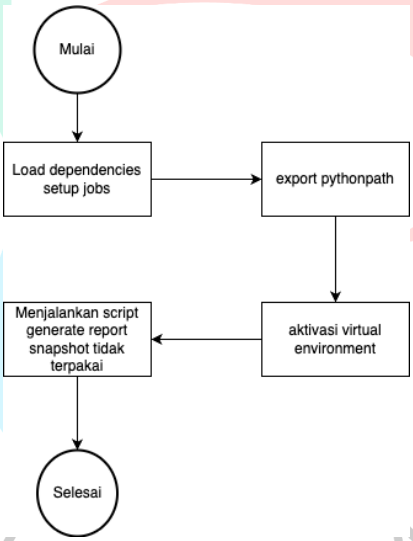
Gambar 4.20 flowchart job get cost project

Gambar 4.20 menjelaskan job *cost project job* yang terdapat pada *stage resource analysis*, yang bertugas untuk mengevaluasi efisiensi biaya project berdasarkan data penggunaan sumber daya terkini. Job ini juga memiliki *dependencies* dari *setup-job* dan hanya dijalankan jika *commit branch* adalah *main*. Pada job ini, *script main.py* dengan perintah *cost-project* dijalankan untuk menghasilkan laporan biaya.



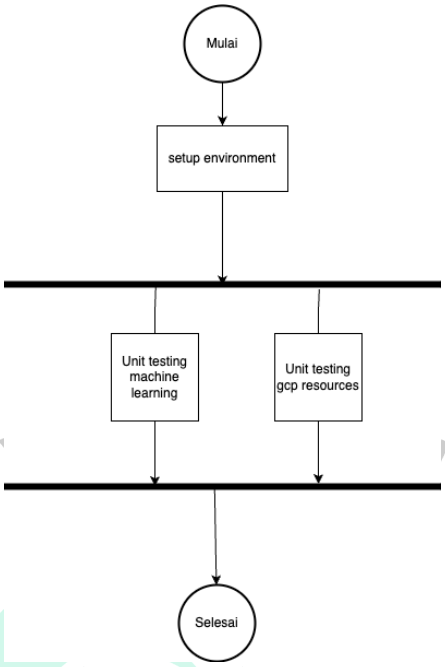
Gambar 4.21 flowchart deattach ip disk

Gambar 4.21 menjelaskan *job deattach IP disk job* yang terdapat pada *stage resource analysis*, yang bertugas mendeteksi disk dan *IP address* yang tidak terpakai di lingkungan cloud. *Dependencies* dari job ini adalah *setup-job*, dan validasi *commit branch* memastikan job hanya dijalankan pada branch *main*. Analisis ini dilakukan menggunakan *script main.py* dengan perintah *deattach-ip-disk*.



Gambar 4.22 flowchart job get old snapshot

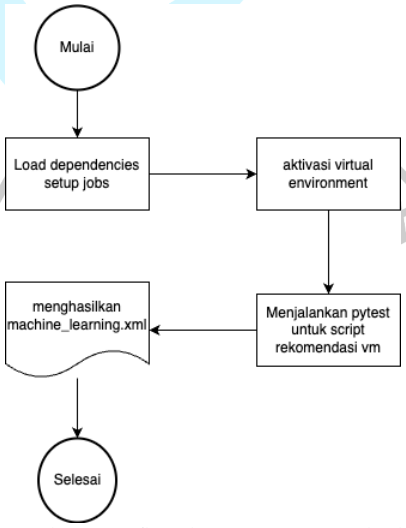
Gambar 4.22 menjelaskan *job old snapshots job*, yang bertugas untuk menganalisis snapshot lama guna mengidentifikasi snapshot yang dapat dihapus. *Job* ini memiliki *dependencies* dari *setup-job* dan hanya dijalankan jika *commit branch* adalah *main*, sesuai validasi yang didefinisikan. Pada *job* ini, *script main.py* dijalankan dengan perintah *old-snapshots*, yang menggunakan parameter seperti *JOBS*, *YEARS*, *MONTHS*, *DAYS*, dan *DISK_SIZE* untuk menentukan kriteria analisis snapshot.



Gambar 4.23 CI pipeline merge request

Gambar 4.23 menjelaskan perancangan *pipeline* untuk tahapan *unit testing*, yang dijalankan secara otomatis saat terjadi *merge request*. *Merge request* adalah permintaan yang diajukan oleh pengembang untuk menggabungkan perubahan kode pada sebuah *branch* ke dalam *main branch*, setelah melalui proses review dan pengujian untuk memastikan bahwa kode yang diubah tidak menimbulkan error atau konflik. *Pipeline* untuk *unit testing* ini terdiri dari dua *stages*, yaitu *setup* dan *unit test*. Berikut adalah penjelasan per *stage* dalam *pipeline* ini:

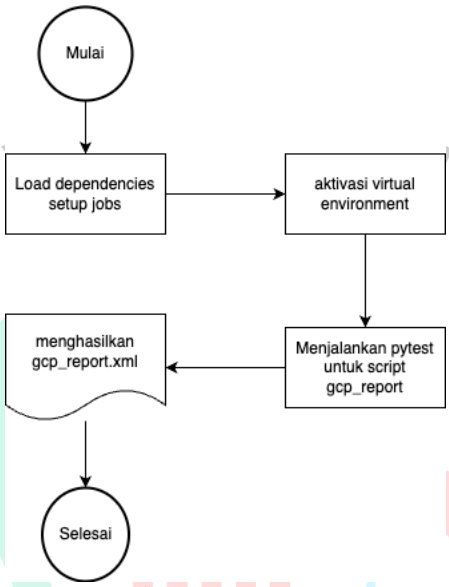
Pada stage setup, sama seperti yang telah dijelaskan pada bagian gambar 4.12 Hasil dari stage ini disimpan sebagai artifact di direktori venv, yang kemudian digunakan pada semua job di stage unit testing.



Gambar 4.24 flowchart unit testing ml job

Gambar 4.24 menjelaskan job unit testing *Machine Learning*, yang bertugas menjalankan pengujian fungsi-fungsi pada modul *Machine Learning*, seperti

perhitungan APU dan pelatihan model. Job ini memiliki dependencies dari *setup-job* dan hanya dijalankan jika pipeline dijalankan dalam konteks *merge request*, sesuai validasi yang didefinisikan. Pada job ini, pengujian dilakukan dengan menggunakan *pytest* untuk memastikan bahwa fungsi dalam modul *recommendation_vm* bekerja dengan baik. Hasil dari pengujian ini disimpan dalam laporan dengan format file *machine_learning.xml*, yang dapat digunakan untuk memeriksa status pengujian dan menganalisis hasilnya.



Gambar 4.26 flowchart unit testing gcp report

Gambar 4.26 menjelaskan job *unit testing GCP resources*, yang bertugas menjalankan pengujian fungsi-fungsi terkait integrasi GCP, seperti pengumpulan laporan dan analisis resource. Job ini juga memiliki dependencies dari *setup-job* dan hanya dijalankan jika pipeline dijalankan dalam konteks *merge request*, sesuai validasi yang telah ditentukan. Pada job ini, pengujian dilakukan dengan menggunakan *pytest* untuk memastikan bahwa fungsi dalam modul *gcp_reports* bekerja dengan baik. Hasil dari pengujian ini disimpan dalam laporan dengan format file *gcp_reports.xml*, yang dapat digunakan untuk memeriksa status pengujian dan menganalisis hasilnya.

4.3.3 Perancangan Pengujian

Pengujian sistem dilakukan dengan menggunakan metode *A/B Testing, white box*, dan *confusion matrix*. Tujuan dari perancangan pengujian ini adalah untuk memberikan gambaran dan memastikan sistem bekerja sesuai fungsinya tanpa ada kesalahan, baik dari sisi antarmuka maupun proses internal. Berikut adalah rincian rancangan pengujian sistem yang akan dilakukan:

4.3.3.1 Perancangan Pengujian A/B Testing

Tabel 4.4 Rancangan pengujian A/B Testing

Skenario Pengujian	Parameter yang dievaluasi	Hasil yang diharapkan
Evaluasi biaya sebelum dan sesudah optimisasi	*Cost Before, Cost After PSO, Cost Saving (USD), Cost Saving (%)*	PSO memberikan penghematan biaya yang signifikan dalam penggunaan sumber daya cloud dibandingkan kondisi awal.
Evaluasi rekomendasi optimasi berdasarkan PSO	*Total recommendation, valid recommendation, percentage recommendation optimization*	PSO memberikan rekomendasi yang tepat dan efisien dalam penyesuaian konfigurasi VM sesuai dengan kebutuhan sumber daya.

Pada Tabel 4.4, rancangan pengujian *A/B Testing* bertujuan untuk mengevaluasi efektivitas metode PSO dalam mengoptimalkan biaya penggunaan sumber daya cloud. Pengujian dilakukan dengan membandingkan biaya awal (*Cost Before*) dan biaya setelah optimisasi (*Cost After* PSO) untuk setiap bulan, dengan hasil penghematan dihitung dalam satuan dolar (*Cost Saving* (USD)) dan persentase (*Cost Saving* (%)). Hasil pengujian ini diharapkan dapat menunjukkan bahwa optimisasi berbasis PSO mampu mengurangi biaya penggunaan sumber daya cloud secara signifikan. Selain itu, pengujian ini juga akan mengevaluasi rekomendasi optimasi yang dihasilkan oleh PSO, dengan parameter yang dievaluasi berupa Total *recommendation, valid recommendation*, dan *percentage recommendation optimization*. Hasil yang diharapkan adalah PSO memberikan rekomendasi yang tepat dan efisien dalam penyesuaian konfigurasi VM sesuai dengan kebutuhan sumber daya.

4.3.3.2 Perancangan Pengujian *Whitebox*

Tabel 4.5 Rancangan Pengujian White Box

Skenario Pengujian	Hasil yang Diharapkan
Unit testing untuk fungsi pengumpulan data dari Datadog API	Fungsi mengambil data metrik dari Datadog API dan mengembalikannya dalam format yang sesuai untuk analisis, tanpa ada error koneksi atau format data yang salah.
Unit testing untuk fungsi preprocessing data	Fungsi melakukan normalisasi dan agregasi data harian menjadi data mingguan dengan benar, serta memeriksa data yang hilang atau tidak valid.
Unit testing untuk model *Machine learning* dalam tahap pelatihan	Model berhasil dilatih dengan data yang telah dipreproses, tanpa ada error selama proses training, dan menghasilkan model yang sesuai untuk deteksi anomali dan rekomendasi VM.
Unit testing untuk algoritma PSO dalam optimasi rekomendasi	Algoritma PSO berjalan dengan benar, mencari solusi optimal berdasarkan input model *Machine learning*, dan mengeluarkan hasil optimasi tipe VM tanpa kesalahan komputasi.

Pada Tabel *4.5*, rancangan pengujian *white box* bertujuan untuk menguji bagian dalam sistem dengan menggunakan pendekatan unit testing. Pengujian ini fokus

pada pengujian logika setiap fungsi utama dan integrasi antar-modul yang terkait dalam pipeline DevOps dan *Machine learning*. Proses pengujian dimulai dari tahap pengumpulan data, *preprocessing*, pelatihan model, hingga tahap optimasi menggunakan algoritma *PSO* dan penyimpanan hasil. Unit testing memastikan bahwa setiap fungsi bekerja sesuai dengan spesifikasinya, baik dalam hal logika komputasi maupun format data, sehingga setiap komponen berkontribusi pada keseluruhan fungsi sistem dengan optimal.

4.3.3.3 Perancangan Pengujian Confusion Matrix

Tabel 4.6 Perancangan Confusion Matrix

Pengukuran Performance	Perhitungan	Hasil yang diharapkan
Akurasi	$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$	> 0 -1
Presisi	$precision = \frac{TP}{TP + FP}$	> 0 -1
Recall	$recall = \frac{TP}{TP + FN}$	> 0 -1
F1 Score	$F1\ score = 2\ x \frac{precision\ x\ recall}{precision + recall}$	> 0 -1

Pada tabel 4.6, dirancang skenario pengujian untuk *confusion matrix* yang meliputi perhitungan akurasi, presisi, *recall*, dan *F1 score*, yang menjadi indikator utama performa model. Pengukuran ini memberikan pandangan menyeluruh terkait efektivitas model dalam membuat rekomendasi yang sesuai dengan kondisi aktual. Pengujian ini bertujuan untuk mengevaluasi performa model dalam memberikan rekomendasi optimal untuk tipe mesin virtual berdasarkan data penggunaan sumber daya di lingkungan cloud. Semakin tinggi nilai akurasi mendekati 1, semakin baik performa model dalam mendeteksi kebutuhan *upgrade* atau *downgrade* sumber daya pada GCP.