**Jurnal Teknik Informatika (JUTIF)**
Vol. 4, No. 6, Desember 2023, pp. 1313-1325
DOI: https://doi.org/10.52436/1.jutif.2023.4.6.1062
p-ISSN: 2723-3863
e-ISSN: 2723-3871

# *AUTOMATIC GIT REPOSITORY DEPLOYER IN UBUNTU USING PYTHON, JENKINS AND CLOUD FIRESTORE AT PT XYZ*

**Felix Vicky Lugas Dewangga*[1], Pratyaksa Ocsa Nugraha Saian[2]**

[1,2]Informatics Engineering Study Program, Universitas Kristen Satya Wacana, Indonesia
Email: [1]672018159@student.uksw.edu, [2]pratyaksa.ocsa@uksw.edu

(Article received: May 23, 2023; Revision: June 20, 2023; published: Desember 23, 2023)

***Abstract***

*PT XYZ is a retail company using Cloud Computing to support its business operations. The company requires the back-end application deployment process to be done automatically on each server computer. The problem occurs insecure in the process of the back-end application deployed manually causing human error vulnerabilities and inefficient when the version of application change occurs to operate on each server computer takes time as long as the number of server computers that need to be configured. The research aims to design a deployer system that help company to deploy back-end applications on each Ubuntu server computer automatically efficiently and securely. The research method uses Research and Development to create system. The research produces a design and build of an automated deployer system using Jenkins to create data configuration that aim the version of back-end application as target deploy and stored in the Cloud Firestore database. The stored data causing deployer system change version back-end application as stored data using GIT command and provide application to operate as service on each server computer. Operation result of application recorded as data deployment that stored in the Cloud Firestore then Jenkins detect it to store data as deployment log. Based on the results, the deployer system considered is able to operate back-end application automatically according the target version on each server computer as eficient and secure deployment process. This conclusion is supported by the results of the questionnaire, which obtained a 85% percentage and was classified as "Strongly Agree" with the created system.*

**Keywords**: *Cloud Firestore, Deployment, GIT Repository, Jenkins, Python, Research and Development.*

# *DEPLOYER* OTOMATIS GIT REPOSITORY PADA UBUNTU MENGGUNAKAN PYTHON, JENKINS, DAN CLOUD FIRESTORE DI PT XYZ

**Abstrak**

PT XYZ adalah perusahaan dibidang ritel dengan teknologi *Cloud Computing* untuk menunjang operasional bisnisnya. Perusahaan tersebut ingin melakukan proses *deployment* secara otomatis aplikasi *back-end* pada setiap komputer *server*. Permasalahan yang terjadi pada proses *deployment* aplikasi *back-end* yang dilakukan secara manual menyebabkan kerentanan *human error* sehingga tidak aman dan tidak efisien setiap terjadi perubahan versi dari aplikasi *back-end* untuk beroperasi pada setiap komputer *server* membutuhkan waktu selama banyaknya komputer *server* yang perlu dilakukan konfigurasi. Penelitian bertujuan untuk merancang sistem *deployer* untuk memenuhi kebutuhan perusahaan dalam melakukan deployment aplikasi *back-end* pada setiap komputer *server* Ubuntu secara otomatis dengan efisien dan aman. Metode penelitian menggunakan *Research and Development* untuk membuat sistem. Penelitian ini menghasilkan rancang bangun sistem *deployer* otomatis dengan Jenkins untuk membuat data konfigurasi berupa versi aplikasi *back-end* yang menjadi tujuan *deployment* yang disimpan pada *database* Cloud Firestore. Data yang disimpan menyebabkan sistem *deployer* pada setiap komputer *server* melakukan *deployment* secara otomatis dengan merubah versi aplikasi menggunakan perintah GIT kemudian menjalankan aplikasi agar tersedia dan beroperasi sebagai *service*. Hasil *deployment* aplikasi pada setiap komputer disimpan pada Cloud Firestore dan terdeteksi pada Jenkins yang dapat dicatat sebagai log. Berdasarkan hasil penelitian sistem *deployer* dinilai dapat melakukan proses *deployment* pada aplikasi *back-end* yang beroperasi di setiap komputer *server* secara otomatis dengan efisien dan aman. Kesimpulan tersebut didukung dengan hasil kuesioner yang memperoleh persentase sebesar 85% dan digolongkan “Sangat Setuju”.

**Kata kunci**: *Cloud Firestore, Deployment, GIT Repository, Jenkins, Python, Research and Development*.

**Jurnal Teknik Informatika (JUTIF)**
Vol. 4, No. 6, Desember 2023, pp. 1313-1325
DOI: https://doi.org/10.52436/1.jutif.2023.4.6.1062
p-ISSN: 2723-3863
e-ISSN: 2723-3871

## 1. PENDAHULUAN

*Software deployment* adalah suatu proses untuk merilis perangkat lunak yang dibuat (fungsionalitas baru, perubahan, dan perbaikan bug) kepada pengguna [1]. Kode sumber dari perangkat lunak yang akan rilis disimpan pada GIT Repository untuk merekam setiap perubahan kode sumber berupa hasil pengembangan perangkat lunak tersebut. GIT Repository adalah penyimpanan informasi suatu versi dalam koleksi yang terdiri atas banyak *file* dan *folder* sebagai *version control system* [2]. Pelacakan suatu versi kode sumber pada GIT Repository dapat dilakukan dengan cuplikan khusus bernama *commits* yang berisi perbedaan antara setiap file orisinal dengan setiap file baru [3]. Proses *deployment* pada perangkat lunak tersebut meliputi proses yang kompleks dengan menggunakan versi dari suatu kode sumber program agar dapat beroperasi dengan berbagai fitur yang ada untuk digunakan pengguna.

PT XYZ adalah perusahaan di bidang ritel dengan menggunakan teknologi *Cloud Computing* untuk menunjang operasional bisnisnya. Teknologi tersebut menggunakan Load Balancer sebagai teknik dalam jaringan komputer dengan sistem kerja membagi permintaan yang masuk untuk terbagi ke sejumlah komputer server [4]. Aplikasi *back-end* beroperasi pada banyak komputer server dengan versi yang sama agar fitur tetap tersedia dengan beban kerja yang terbagi. Beberapa aplikasi *back-end* yang dibuat dengan bahasa pemrograman Python dijalankan sebagai *services* pada komputer *server* dengan sistem operasi Ubuntu. Ubuntu adalah sistem operasi distribusi linux populer yang umumnya digunakan sebagai server [5]. Aplikasi tersebut berjalan pada banyak komputer *server*. Perusahaan ini menggunakan *version control system* GIT untuk melakukan perubahan kode sumber aplikasi itu untuk beroperasi pada setiap komputer *server* yang masih dilakukan secara manual oleh DevOps (*developer operations*) Engineer. Proses *deployment* konvensional pada PT XYZ adalah proses secara manual yang dilakukan dengan masuk ke setiap komputer *server* melalui akses *secure shell* kemudian mengubah kode sumber aplikasi *back-end* dengan perintah GIT dan melakukan *deployment* aplikasi sebagai *services* pada komputer tersebut. Proses *deployment* konvensional menyusahkan DevOps Engineer karena harus melakukan konfigurasi pada setiap komputer *server* ketika terdapat kode sumber aplikasi *back-end* dengan fitur baru yang telah siap untuk produksi agar aplikasi tersedia dan beroperasi pada komputer tersebut. Devops Engineer bertanggung jawab untuk membuat naskah, membuat kode program, memahami arsitektur *Cloud*, komunikasi dan berkolaborasi, pengiriman dan integrasi secara berkala, dan pengetahuan pada peralatan maupun teknologi DevOps yang bervariasi [6]. Proses konfigurasi tersebut membutuhkan waktu yang lama dengan mengikuti banyaknya komputer server yang perlu dikonfigurasi sehingga tidak efisien. Kesulitan lain yang dihadapi ketika kesalahan konfigurasi terjadi akibat dari *human error* menyebabkan DevOps Engineer memeriksa hasil *deployment* aplikasi *back-end* dengan mengakses setiap komputer *server* sehingga menjadi tidak efisien dan aman.

Oleh karena itu diperlukan sistem *deployer* yang dapat melakukan perubahan kode sumber pada suatu aplikasi *back-end* dengan perintah GIT, menjalankan aplikasi tersebut pada setiap komputer *server*, dan menampilkan hasil *deployment* secara otomatis [7]. Sistem *deployer* otomatis dibangun menggunakan bahasa pemrograman Python dengan mengikuti arsitektur sistem yang dapat terintegrasi pada Jenkins dan Cloud Firestore. Jenkins adalah peralatan *continuous deployment* [8]. Cloud Firestore adalah *database* NoSQL yang dapat melakukan sinkronisasi data antara *server* dan *client* terlepas dari latensi jaringan [9]. Sinkronisasi konfigurasi *deployment* dilakukan ketika DevOps Engineer memilih target *deploy* pada Jenkins berupa tujuan versi kode sumber. Versi kode sumber tersebut berupa *branch* atau *tags* dari GIT Repository aplikasi *back-end* di komputer *server* yang terdapat pada *remote repository*. Target *deploy* yang dipilih kemudian disimpan sebagai data konfigurasi *deployment* pada suatu dokumen di *database* Cloud Firestore. Data yang disimpan tersebut menyebabkan setiap komputer *server* melalui sistem *deployer* melakukan proses *deployment* aplikasi *back-end* secara otomatis dengan merubah kode sumber aplikasi *back-end* sesuai target *deploy* yang berasal dari data tersebut, menjalankan aplikasi tersebut, dan menyimpan hasil *deployment* aplikasi tersebut dari komputer *server* ke database Cloud Firestore. Jenkins melalui sistem *deployer* mengambil hasil *deployment* tersebut dan mencatatnya sebagai log. Proses kinerja dari sistem *deployer* ini berdasarkan *puppet master* berupa Jenkins sebagai peralatan yang memberikan suatu perintah pada setiap komputer server yang menjalankankan aplikasi *back-end* sebagai *puppet agent* [10]. Jenkins memberikan perintah dalam bentuk *pipeline* yang dimuat pada *pipeline script* untuk melakukan *deployment* pada aplikasi *back-end* pada setiap komputer *server* [11].

Metode penelitian yang digunakan untuk melakukan penyelesaian masalah menggunakan metode *Research and Development* oleh Sugiyono [12]. Metode penelitian tersebut didukung dengan pengujian sistem *Black Box Testing* untuk menguji suatu sistem yang telah dibangun [13]. Proses pengujian sistem tersebut menjadi mudah dengan menggunakan *activity diagram* untuk menganalisa perilaku sistem [14]. Hasil dari sistem *deployer* yang dibangun menggunakan penilaian metode skala likert. Skala likert adalah skala yang digunakan untuk mengukur persepsi, sikap atau pendapat seseorang atau kelompok mengenai sebuah peristiwa [15].

Berdasarkan latar belakang yang telah disebutkan, penelitan ini bertujuan untuk merancang sistem *deployer* agar dapat memenuhi kebutuhan PT XYZ untuk melakukan *deployment* aplikasi *back-end* pada setiap komputer *server* Ubuntu secara otomatis dengan efisien dan aman. Penelitian ini menghasilkan sebuah rancangan sistem *deployer* otomatis aplikasi *back-end* pada setiap komputer *server* Ubuntu menggunakan Python, Jenkins, dan Cloud Firestore sehingga aplikasi *back-end* dapat tersedia dan beroperasi pada setiap komputer tersebut dengan fitur sesuai dengan versi kode sumber secara efisien dan aman.

## 2. METODE PENELITIAN

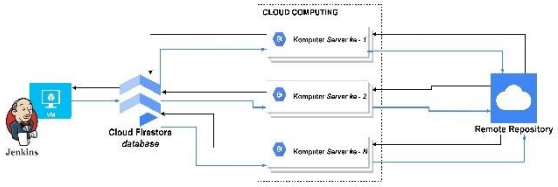
*Research and Development* (R&D) adalah model pengembangan penelitian oleh Sugiyono [12]. Tujuan disesuaikan dengan kondisi penelitian dari pembuatan *deployer* otomatis pada sistem operasi Ubuntu. Model sebagai alur penelitian ini dapat dilihat pada Gambar 1.


Gambar 1. Model *Research and Development*

Gambar 1 menjelaskan semua proses yang dilakukan dalam penelitian. Tahap pengumpulan awal dilakukan untuk mendapatkan permasalahan PT XYZ yaitu ingin melakukan proses *deployment* dari aplikasi *back-end* yang berjalan pada sistem operasi Ubuntu secara otomatis dengan efisien dan aman. Tahap perencanaan dilakukan dengan menggunakan informasi awal yang menghasilkan perancangan sistem *deployer* otomatis GIT repository pada sistem operasi Ubuntu dibuat dengan bahasa pemrograman Python yang dapat terhubung dengan Cloud Firestore dan Jenkins. Tahap perencanaan dilanjutkan Pengembangan produk awal *deployer otomatis* dengan memanfaatkan GIT Repository, Cloud Firestore, dan Jenkins. Uji coba awal *deployer* otomatis pada melibatkan 6 karyawan PT XYZ yang biasa melakukan proses *deployment* secara manual. Uji coba awal tersebut bertujuan untuk mengetahui permasalahan dan kebutuhan bagi PT XYZ yang terjadi pada sistem yang dikembangkan telah sesuai. Kemudian revisi produk awal dilakukan dan kembali melakukan uji coba yang melibatkan 20 karyawan *Back-End Developer* yang melakukan proses *deployment* secara manual pada PT XYZ. Dari uji coba diperoleh data berupa hasil diskusi dan kuesioner. Permasalahan yang diterima pada hasil tersebut, maka akan dilakukan revisi produk. Tahap akhir adalah implementasi sistem dengan produk aplikasi *deployer* otomatis yang dapat menunjang proses *deployment* / penyebaran aplikasi *back-end* yang tersedia dan beroperasi pada komputer *server* dengan aman dan efisien pada PT XYZ.

### 2.1. Perancangan Sistem *deployer* otomatis

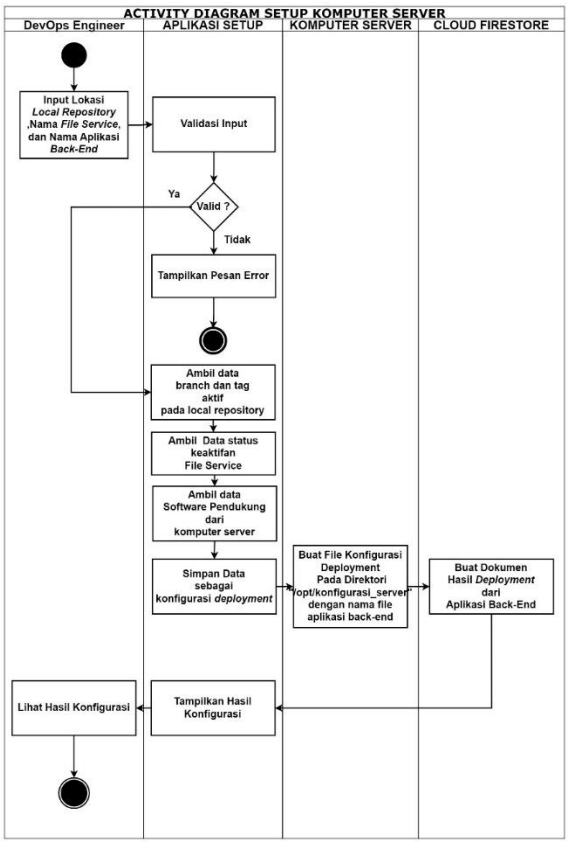
Awal suatu perancangan sistem dilakukan dengan pembuatan arsitektur sistem yang berfungsi sebagai kerangka kerja yang mendeskripsikan bentuk dan struktur komponen yang saling berinteraksi satu sama lain. Arsitektur sistem menggunakan simbol dari suatu produk. Desain dari arsitektur sistem pembuatan *deployer* otomatis dapat dilihat pada Gambar 2.




Gambar 2. Arsitektur Sistem *Deployer* Otomatis

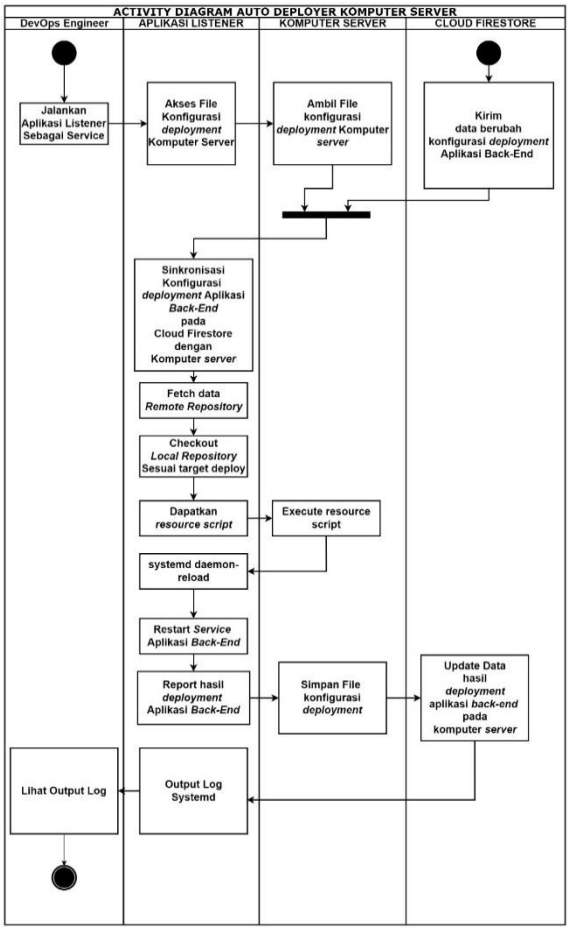
Pada Gambar 2 menggambarkan *Virtual Machine* melalui aplikasi Jenkins berinteraksi dengan *database* Cloud Firestore untuk melakukan perubahan data di *database* tersebut berupa konfigurasi *deployment*. Data yang berubah tersebut dilakukan sinkronisasi dengan pemantauan dan penyesuaian data konfigurasi *deployment* yang terdapat di setiap komputer *server* mengikuti konfigurasi yang terdapat pada *database* Cloud Firestore. Data konfigurasi *deployment* pada komputer *server* yang tidak sinkron dengan data konfigurasi *deployment* pada Cloud Firestore menyebabkan komputer tersebut melakukan *deployment* secara otomatis dengan mengambil kode sumber dari *remote repository* mengikuti konfigurasi *deployment* dari Cloud Firestore.

*Activity diagram* adalah spesifikasi semi-formal bersifat intuitif dan fleksibel yang digunakan untuk mendeskripsikan perilaku sistem beserta dengan logika dari operasi yang kompleks [14]. Deskripsi perilaku tersebut menggambarkan proses yang dinotasikan dengan aktivitas yang memiliki tanda panah dengan menunjuk alur yang jelas antara proses satu dengan yang lainnya. *Activity diagram* dari sistem *deployer* otomatis dibagi menjadi tiga aktivitas yang saling berkaitan satu dengan yang lainnya. Aktivitas tersebut adalah aktivitas *setup* komputer server, aktivitas auto *deployer*, dan aktivitas *input* konfigurasi *deployment* pada Cloud Firestore. *Activity diagram* setup komputer server terdapat pada Gambar 3.




Gambar 3. Activity Diagram Setup Komputer Server

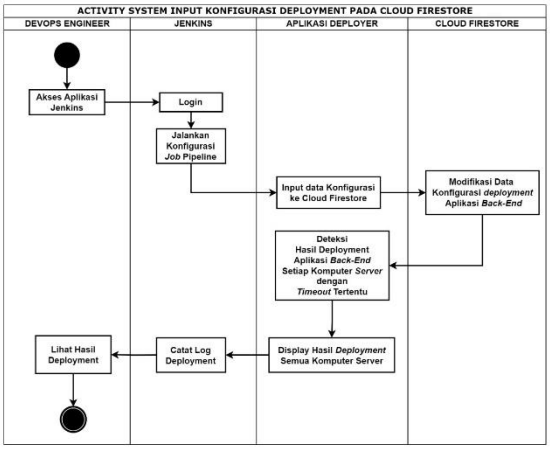
Pada Gambar 3 adalah *activity diagram* untuk membuat konfigurasi *deployment* yang disimpan pada komputer server dan membuat laporan hasil *deployment* aplikasi Back-End pada komputer server di *database* Cloud Firestore. DevOps Engineer memasukan data lokasi *local repository* aplikasi *back-end*, nama file *service*, dan nama aplikasi *back-end* dari komputer server Ubuntu sebagai data *input*. Data tersebut akan dilakukan proses validasi untuk memastikan lokasi *local repository* dan file *service* dari aplikasi back-end tersedia. Data *input* yang valid kemudian digunakan oleh aplikasi setup untuk mengambil data *branch* dan *tags* aktif dari lokasi *local repository*, data status keaktifan dari file *service* tersebut, dan data software pendukung yang tersedia pada komputer *server* tersebut. Semua data yang diperoleh akan disimpan pada komputer server dengan file bernama aplikasi *back-end* sebagai konfigurasi *deployment*. Aplikasi *setup* kemudian menyimpan semua data sebagai hasil *deployment* dari komputer *server* dengan aplikasi *back-end* pada *database* Cloud Firestore.




Gambar 4. Activity Diagram *Auto Deployer* Komputer Server

Pada Gambar 4 adalah *activity diagram* untuk *deploy* otomatis suatu aplikasi *back-end* yang telah terdapat konfigurasi *deployment* yang pada suatu penyimpanan di komputer *server*. Konfigurasi tersebut di dapat dari proses *setup*. DevOps Engineer menjalankan aplikasi listener sebagai *services* pada komputer *server*, aplikasi *listener* secara otomatis akan tetap berjalan sebagai *background services* walau komputer *server* mengalami *restart* tanpa proses manual yang perlu dilakukan oleh DevOps Engineer. Aplikasi *listener* memastikan konfigurasi *deployment* dalam bentuk file yang berada pada komputer *server* dapat diakses. Konfigurasi tersebut kemudian diambil dan dilakukan proses sinkronisasi. Proses tersebut akan mengambil data konfigurasi *deployment* dengan identifikasi berupa nama aplikasi *back-end* yang sama antara komputer *server* berupa nama file dengan *database* Cloud Firestore berupa nama dokumen. Data konfigurasi *deployment* dengan nama yang sama akan dilakukan proses sinkronisasi dengan data dari file komputer *server* yang menyesuaikan dengan data dari dokumen *database* Cloud Firestore. Aplikasi listener kemudian melakukan proses *fetch* untuk mengambil kode sumber dari *remote repository* yang kemudian disimpan pada *local repository* aplikasi *back-end* di komputer *server* tersebut. Data yang telah diambil kemudian dilakukan proses *checkout* untuk

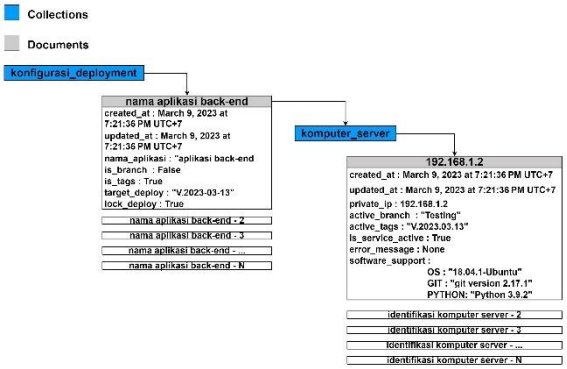
mengganti kode sumber program sesuai dengan versi yang dipilih sebagai *target deploy*. Kode sumber yang telah diubah kemudian mengambil *resource script* berupa *file* yang berasal dari kode sumber tersebut. *Resource script* adalah kode program berupa *file* yang digunakan untuk instalasi perangkat lunak dan menyediakan kode sumber tambahan yang bertujuan untuk melengkapi kebutuhan aplikasi *back-end* agar dapat berjalan pada komputer *server*. *File* tersebut adalah berkas berisi suatu perintah yang dapat dilakukan intepretasi pada sistem operasi Ubuntu [5]. Kemudian dilanjutkan proses *systemd daemon-reload* untuk memuat ulang *file service* agar aplikasi berjalan dalam sistem manajemen pada Ubuntu bernama Systemd. *Restart service* digunakan untuk menjalankan ulang aplikasi *back-end* agar aplikasi berjalan dengan kode sumber yang telah berubah. Hasil laporan konfigurasi *deployment* kemudian disimpan pada komputer *server* dan *database* Cloud Firestore.




Gambar 5. Activity Diagram *Input* Konfigurasi *Deployment* pada Cloud Firestore

Pada Gambar 5 adalah *activity diagram* untuk membuat konfigurasi *deployment* yang disimpan pada *database* Cloud Firestore. Tujuan dari aktivitas ini untuk membatasi hak akses pengguna, membuat konfigurasi *deployment* pada suatu dokumen pada Cloud Firestore, menampilkan hasil *deployment*, dan mencatat seluruh proses *deployment* dari aplikasi *back-end* dalam bentuk *log* pada Jenkins yang menjadi target *deployment*. Tahap awal adalah DevOps mengakses aplikasi Jenkins berupa *web application*. Proses *login* pada aplikasi dapat dilakukan dengan memberikan *credentials* berupa *username* dan *password*. Tahap berikutnya DevOps Engineer membuat konfigurasi *deployment* aplikasi *back-end* dengan menggunakan Pipeline yang telah dikonfigurasi pada Jenkins. Jenkins berintegrasi aplikasi *deployer* dengan mengirimkan konfigurasi *deployment* yang dibuat oleh DevOps Engineer ke *database* Cloud Firestore. Aplikasi *deployer* mendeteksi semua hasil *deployment* aplikasi *back-end* pada semua komputer *server*. Hasil *deployment* tersebut kemudian ditampilkan oleh aplikasi *deployer* dan Jenkins mencatat hasil tersebut sebagai *log deployment*.

Model dari *database* Cloud Firestore digunakan untuk menyimpan data konfigurasi *deployment* serta sinkronisasi data. Data tersebut dibuat melalui Jenkins menggunakan aplikasi *deployer* untuk disimpan. Sinkronisasi data dilakukan dengan menggunakan aplikasi *listener* yang berjalan pada komputer *server* untuk mendeteksi perubahan data yang dipantau pada koleksi "*konfigurasi_deployment*". Berbagai dokumen dengan nama aplikasi *back-end* yang unik antara satu dengan yang lainnya akan tampil pada koleksi tersebut. Model tersebut dapat dilihat pada Gambar 6.




Gambar 6. Model Data Konfigurasi *Deployment* pada Cloud Firestore

Pada Gambar 6 terdapat *subkoleksi* dari suatu dokumen aplikasi *back-end* bernama "komputer_server". *Subkoleksi* pada model tersebut digunakan sebagai tempat untuk menyimpan hasil *deployment* dalam bentuk dokumen yang berkorelasi dengan dokumen aplikasi *back-end* tersebut. Dokumen yang disimpan pada *subkoleksi* tersebut memiliki nama yang unik antara dokumen satu dengan dokumen yang lainnya dengan nama identitas dari komputer *server*. Identitas dari komputer server tersebut berupa *private internet protocol* yang berbeda pada setiap komputer *server*.

## 2.2. GIT Repository

GIT Repository adalah penyimpanan informasi suatu versi dalam koleksi yang terdiri atas banyak file dan folder sebagai *version control system* [2]. Kode sumber aplikasi disimpan pada suatu *repository* sebagai VCS (*Version Control System)*. *Version Control System* dapat melakukan manajemen terhadap kode sumber program sesuai dengan versi yang ditandai. Setiap perubahan kode sumber tersebut disimpan sebagai *commit*. Setiap *commit* terjadi perubahan file dan folder yang terjadi pada aplikasi tersebut berupa penambahan, penghapusan, dan perubahan kode. Perintah GIT digunakan untuk melakukan *update* kode sumber aplikasi pada suatu *repository*. Penelitian ini memanfaatkan GIT Repository untuk mendapatkan kode sumber sesuai dengan versi aplikasi yang diinginkan yang telah

dikembangkan oleh *developer* untuk dilakukan proses *deployment*. Versi kode sumber tersebut dipilih sesuai dengan tanda yang tersedia berupa *tags* atau *branch*. Proses *deployment* oleh sistem *deployer* otomatis dapat dilakukan sesuai dengan *tags* atau *branch* dari kode sumber tersebut sebagai *target deploy*. GIT Repository yang terdapat pada komputer *server* disebut sebagai *local repository* dalam sistem *deployer* otomatis.

### 2.3. Jenkins

Jenkins merupakan peralatan DevOps yang berguna untuk melakukan *automation* server dengan mendukung CI/CD (*Continuous Integration* dan *Continuous Deployment*) bersifat *open source* [11]. DevOps Engineer menggunakan peralatan ini untuk melakukan otomatisasi proses *deployment* pada aplikasi *back-end* yang terdapat pada setiap komputer server secara otomatis. Proses *deployment* aplikasi *back-end* dapat dilakukan dengan menggunakan fitur *pipeline script* yang digunakan untuk membuat perintah eksekusi. Setiap perintah dapat disatukan dalam setiap *stage* yang digunakan untuk memberikan nama proses tersebut. Tujuan pemberian nama proses tersebut digunakan untuk menandai dan mengatur perilaku jika perintah yang dijalankan terjadi eror akan dapat diketahui. Fitur lain pada Jenkins dapat mendukung pencatatan log hasil eksekusi perintah yang dilakukan. Jenkins terdapat *User Interface* yang dapat diakses berbasis *web applications*. Jenkins berguna pada penelitian ini yang digunakan untuk memberikan perintah untuk melakukan eksekusi perintah tersebut dan mendapatkan pelaporan hasil eksekusi. Hasil eksekusi perintah tersebut dapat ditampilkan dan dicatat sebagai *log*.

### 2.4. Cloud Firestore

Cloud Firestore adalah *database* NoSQL yang dapat melakukan sinkronisasi data antara server dan client terlepas dari latensi jaringan [9]. Cloud Firestore berguna sebagai *database* yang dapat melakukan sinkronisasi data secara *realtime*. Sinkronisasi data yang terjadi berguna untuk mengirimkan perubahan data sebagai notifikasi terhadap bahasa pemrograman yang didukung untuk terhubung dengan *database* Cloud Firestore. Bahasa pemrograman Python dapat terhubung dengan Cloud Firestore untuk melakukan sinkronisasi perubahan data.

### 2.5. Ubuntu

Ubuntu adalah distribusi linux yang populer yang umumnya digunakan untuk server [5]. Ubuntu mendukung *service* manajemen melalui perangkat lunak Systemd mengoperasikan aplikasi sebagai *service*. Systemd adalah sistem penjadwalan yang digunakan pada distribusi Linux dengan fungsi untuk melakukan logging, penjadwalan, pemantauan service, dan insisialisasi sistem dengan mempersiapkan operasional perangkat ketika sistem operasi termuat [5]. Aplikasi *Back-End* dapat beroperasi pada Ubuntu sebagai *background services* dengan *output* yang dapat dicatat sebagai *logs* untuk melakukan penyelesaian masalah yang terjadi pada *service* tersebut. Sistem *deployer* otomatis menggunakan sistem operasi Ubuntu untuk menjalankan *service* yang berguna sebagai proses *deployment* secara otomatis.

## 3. HASIL DAN PEMBAHASAN

Penelitian ini menghasilkan sistem *deployer* yang berguna untuk melakukan proses *deployment* aplikasi *Back-End* untuk tersedia dan beroperasi pada setiap komputer *server* Ubuntu sesuai dengan versi kode sumber yang dipilih sebagai *target deploy* secara otomatis. Kode sumber aplikasi *Back-End* tersedia pada setiap *komputer* server melalui *version control system* GIT pada GIT Repository. Perintah GIT digunakan untuk mengubah versi kode sumber aplikasi tersebut. Versi yang berubah akan beroperasi pada komputer server sebagai *service* melalui proses *deployment*. Perubahan versi kode sumber dan proses *deployment* pada aplikasi *Back-End* dilakukan secara otomatis melalui aplikasi Python yang dibangun menggunakan Cloud Firestore dan Jenkins.

Sistem *deployer* dibangun sebagai bentuk implementasi sesuai dengan perancangan sistem dan hasil uji coba sistem yang dilakukan dengan menggunakan metode *Black Box Testing* serta uji kelayakan sistem. *Black Box Testing* adalah pengujian fungsional perangkat lunak tanpa mengetahui struktur internal program [13]. Hasil dari perancangan sistem berupa implementasi program beserta dengan hasil pengujian sistem dan uji kelayakan sistem.

### 3.1. Implementasi

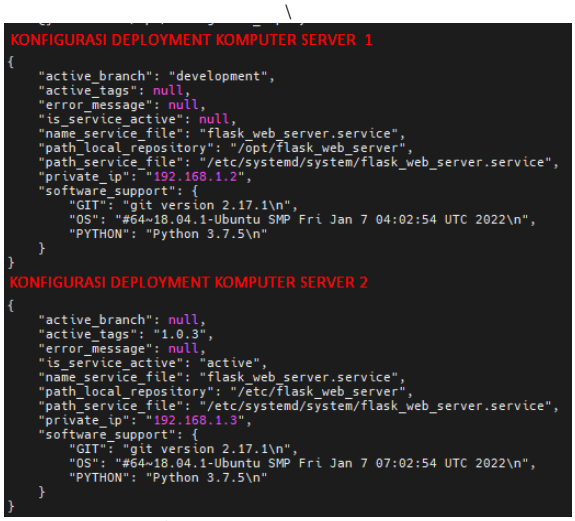
Implementasi sistem *deployer* otomatis ini menggunakan satu *virtual machine* yang menjalankan aplikasi Jenkins, dua komputer *server* dengan sistem operasi Ubuntu yang akan menjalankan aplikasi *Back-End*, dan GIT Repository dari aplikasi *Back-End* tersebut. Kedua komputer server tersebut memiliki identitas *private internet protocol version 4* masing-masing dengan identifikasi “192.168.1.2” dan “192.168.1.3”. Penerapan rancangan ini tergabung dengan tahap pengembangan produk awal untuk melakukan pembuatan kode sumber yang telah direvisi sebagai kode sumber yang dapat digunakan beserta dengan hasil berupa tampilan dari produk sebagai tahap implementasi. Tahap ini menggunakan desain dari perancangan sistem yang telah dibuat. Kode program aplikasi Python dibuat secara eksplisit yang saling berkaitan satu dengan yang lainnya pada sistem *deployer* untuk melakukan *deployment* secara otomatis. Aplikasi Python adalah aplikasi yang dibuat

dengan menggunakan bahasa pemrograman Python. Terdapat tiga aplikasi yang digunakan bernama aplikasi *setup*, aplikasi *listener*, dan aplikasi *deployer* agar sistem *deployer* otomatis dapat bekerja.

Kode program 1. Aplikasi Setup

| | **Kode Program** |
|---|---|
| 1 | import FIRESTORE |
| 2 | import FILE_MANAGEMENT |
| 3 | import DataKonfigurasiDeployment |
| 4 | data_konfigurasi = DataKonfigurasiDeployment() |
| 5 | data_konfigurasi.getInputUser() |
| 6 | FILE_MANAGEMENT.createFile(file_location = data_konfigurasi.lokasi_file, data = data_konfigurasi.komputer_server) |
| 7 | FIRESTORE.modifyData(document_location = data_konfigurasi.lokasi_dokumen_hasil_deployment, data = data_konfigurasi.hasil_deployment_cloud_firestore) |

Kode program aplikasi *setup* tersebut bertujuan untuk membuat konfigurasi *deployment* aplikasi *Back-End* yang disimpan pada komputer *server* dan melaporkan hasil *deployment* aplikasi *Back-End* pada *database* Cloud Firestore. Konfigurasi *deployment* yang disimpan pada komputer *server* menandakan aplikasi *Back-End* tersebut dapat dilakukan *deployment* secara otomatis oleh sistem *deployer*. Konfigurasi tersebut dibuat dengan memasukan lokasi *local repository*, nama aplikasi *Back-End*, dan nama *file service* dari aplikasi *Back-End* pada komputer *server* oleh DevOps Engineer. Aplikasi *setup* kemudian membuat hasil *deployment* aplikasi Back-End yang disimpan pada *database* Cloud Firestore.




Gambar 7. Konfigurasi *Deployment* Aplikasi *Back-End* yang disimpan pada Komputer Server 1 dan 2 melalui Aplikasi *Setup*

Pada Gambar 7 adalah konfigurasi *deployment* aplikasi *Back-End* yang disimpan dalam bentuk JSON (JavaScript Object Notation) pada masing – masing komputer *server* melalui aplikasi *setup*. Konfigurasi tersebut sama seperti yang disimpan pada *database* Cloud Firestore sebagai hasil *deployment* namun terdapat perbedaan pada isi data nama file *service*, lokasi file *service*, dan lokasi *local repository* yang hanya disertakan pada komputer *server*. Perbedaan data tersebut bertujuan membatasi informasi yang ditampilkan pada *database* Cloud Firestore. Pada Gambar 7 menunjukan perbedaan versi aplikasi pada setiap komputer *server*. Komputer *server* 1 sedang menggunakan versi aplikasi berupa *branch* ”development” dan komputer *server* 2 menggunakan versi aplikasi berupa *tags* ”1.0.3”.


Gambar 8. Hasil *Deployment* Aplikasi *Back-End* pada Komputer Server 1 dan 2 yang disimpan pada *database* Cloud Firestore melalui Aplikasi *Setup*

Pada Gambar 8 adalah hasil *deployment* Aplikasi *Back-End* pada komputer server 1 dan 2 yang disimpan pada *Cloud Firestore* dengan lokasi dokumen yang berbeda. Lokasi dokumen hasil konfigurasi pada komputer server 1 dan 2 secara urut adalah “/konfigurasi_deployment/flask_web_server/komputer_server/192.168.1.2” dan “/konfigurasi_deployment/flask_web_server/komputer_server/192.168.1.3” sesuai dengan model data konfigurasi deployment pada Cloud Firestore di perancangan sistem. “flask_web_server” adalah nama aplikasi *Back-End* berupa dokumen nama aplikasi pada koleksi “konfigurasi_deployment”. Pada dokumen tersebut memiliki subkoleksi bernama “komputer_server”. Pada subkoleksi tersebut

terdapat hasil *deployment* pada aplikasi *Back-End* dengan nama dokumen "192.168.1.2" dan "192.168.1.3" berupa identifikasi *private IPv4* masing-masing dari komputer server 1 dan 2.

Aplikasi *Back-End* yang telah diatur melalui aplikasi *setup* menghasilkan konfigurasi *deployment* yang terdapat pada komputer *server* tidak perlu diatur ulang lagi melalui proses manual oleh DevOps Engineer. Proses otomatis akan menggantikan proses manual tersebut melalui aplikasi *listener* yang berjalan sebagai *service* pada komputer *server*. *Service* tersebut akan berjalan secara otomatis meskipun komputer *server* mengalami proses *reboot* (menutup dan membuka kembali sistem operasi).

Systemd adalah sistem penjadwalan yang digunakan pada distribusi Linux dengan fungsi untuk melakukan *logging*, penjadwalan, pemantauan *service*, dan insisialisasi sistem dengan mempersiapkan operasional perangkat ketika sistem operasi termuat [5]. Perangkat lunak tersebut digunakan untuk menjalankan aplikasi *Back-End* sebagai *service*. Aplikasi tersebut perlu dilakukan konfigurasi berupa *file service* agar dapat berjalan pada komputer *server* dengan sistem penjadwalan tersebut. Konfigurasi *file service* pada sistem manajemen Systemd terdapat pada kode program 2.

Kode Program 2. Konfigurasi Aplikasi *listener* Sebagai *Services* Manajemen Sistem Systemd

| | **Kode Program** |
|---|---|
| 1 | [Unit] |
| 2 | Description = Aplikasi listener sebagai auto deployer |
| 3 | After=multi-user.target |
| 4 | [Service] |
| 5 | User=root |
| 6 | Environment="PYTHONUNBUFFERED=1" |
| 7 | WorkingDirectory=/opt/ADS |
| 10 | ExecStart=/usr/local/bin/python3 aplikasi_listener.py |
| 11 | Type=simple |
| 12 | Restart=always |
| 13 | [Install] |
| 14 | WantedBy=multi-user.target |

Aplikasi *listener* berjalan sebagai *service* untuk melakukan proses *deployment* secara otomatis setiap aplikasi *Back-End* pada komputer *server* yang telah terdapat konfigurasi *deployment* melalui aplikasi *setup*. Konfigurasi *deployment* yang terdapat pada komputer *server* akan diatur melalui aplikasi ini sebagai informasi hasil *deployment* yang berisi data versi, status aktif *services*, dan pesan eror dari aplikasi *Back-End* serta informasi perangkat lunak yang tersedia pada komputer *server*. Proses pengaturan konfigurasi *deployment* setiap komputer *server* secara otomatis mengikuti data konfigurasi *deployment* aplikasi *Back-End* berupa dokumen dari *database* Cloud Firestore. Perubahan dokumen tersebut dapat dideteksi melalui aplikasi *listener* dengan menggunakan API (*Application Programming Interfaces*) Python dari Cloud Firestore. Dokumen yang berubah akan dilakukan proses sinkronisasi konfigurasi *deployment* pada komputer *server* dengan konfigurasi *deployment* pada dokumen pada koleksi "konfigurasi_deployment" di *database* Cloud Firestore. Kode sumber untuk melakukan proses sinkronisasi konfigurasi tersebut terdapat pada kode program 3.

Kode Program 3. Sinkronisasi Data Konfigurasi Deployment dari Aplikasi *Listener*

| | **Kode Program** |
|---|---|
| 1 | for change in changes: |
| 2 | if change.type.name == "MODIFIED" or change.type.name == "ADDED": |
| 3 | nama_dokumen = change.document.id |
| 4 | konfigurasi_cloud_firestore = change.document.to_dict() |
| 5 | data_konfigurasi = DataKonfigurasiDeployment() |
| 6 | data_konfigurasi.cloud_firestore(data=konfigurasi_cloud_firestore) |
| 7 | if not data_konfigurasi.checkKonfigurasiKomputerServer(nama_file=nama_dokumen): |
| 8 | return continue |
| 9 | GIT_SCM.changeVersion(data=data_konfigurasi) |
| 10 | DEPLOYMENT.executeResourceScript(data=data_konfigurasi) |
| 11 | DEPLOYMENT.restartService(data=data_konfigurasi) |
| 12 | FILE_MANAGEMENT.modifyFile(file_location=data_konfigurasi.lokasi_file, data=data_konfigurasi.komputer_server |
| 13 | FIRESTORE.modifyData(document_location=data_konfigurasi.lokasi_dokumen,data=data_konfigurasi.cloud_firestore) |

Kode program 3 adalah kode program dari *activity diagram auto deployer* pada rancangan sistem. Aplikasi *listener* beroperasi sebagai *background service* pada perangkat lunak Systemd pada setiap komputer *server*. Aplikasi tersebut bekerja secara berkelanjutan untuk mendeteksi perubahan data konfigurasi *deployment* aplikasi *Back-End* pada Cloud Firestore. Perubahan data tersebut kemudian dilakukan proses validasi untuk mendeteksi ketersediaan konfigurasi *deployment* pada komputer server yang dibuat melalui aplikasi *setup* sebagai proses sinkronisasi. Proses sinkronisasi terjadi jika nama dokumen konfigurasi *deployment* pada Cloud Firestore sesuai dengan nama *file* konfigurasi *deployment* yang terdapat pada komputer *server* tersebut.

Kode Program 4. Aplikasi *deployer*

| | **Kode Program** |
|---|---|
| 1 | data_konfigurasi = DataKonfigurasiDeployment() |
| 2 | data_konfigurasi.setTargetDeploy() |
| 3 | FIRESTORE.modifyData(document_location = data_konfigurasi.lokasi_dokumen, data = data_konfigurasi.cloud_firestore) |
| 4 | FIRESTORE.snapshotHasilDeployment(sub_koleksi_komputer_server = data_konfigurasi.sub_koleksi_cloud_firestore) |

Kode program 4 Aplikasi *deployer* digunakan untuk membuat konfigurasi *deployment* yang disimpan pada *database* Cloud Firestore. Konfigurasi tersebut berisikan data berupa *target deploy* sebagai versi aplikasi *Back-End* yang dipilih untuk dilakukan *deployment* pada setiap komputer *server* yang telah menjalan *service auto deployer* melalui aplikasi

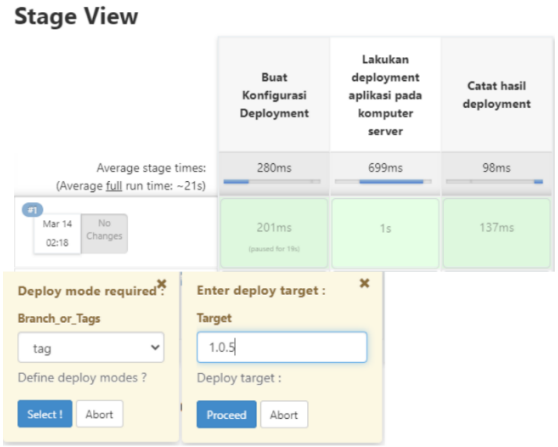
*listener* dan telah terdapat konfigurasi *deployment* aplikasi *Back-End* yang disimpan pada komputer tersebut. Target deploy tersebut berupa *branch* atau *tags* yang terdapat pada suatu GIT Repository pada setiap komputer server. Aplikasi *deployer* ini beroperasi melalui interaksi *input* user pada perangkat lunak Jenkins.

Kode program 5. Pipeline Script Jenkins Membuat Konfigurasi *Deployment* Aplikasi *Deployer*

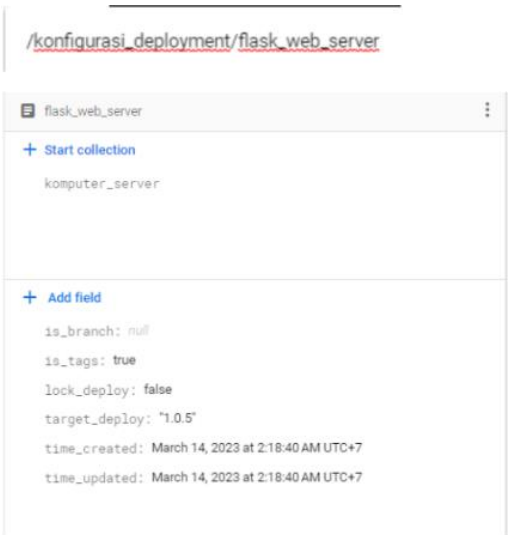
**Kode Program**

```
1  def deploy_target = ""
2  def deploy_mode = "" //"branch" or "tag"
3  def nama_dokumen_cloud_firestore = "flask_web_server"
4  def auto_deployer_location = "/opt/ADS"
5  def hasil_deployment = ""
6  pipeline{
7      agent {label "built-in"}
8      stages{
9          stage('Buat Konfigurasi Deployment'){
               steps{
10                 script{deploy_mode = input
11 message: 'Deploy mode required :', ok: 'Select !', parameters: [choice(name: 'Branch_or_Tags', choices: 'branch\ntag', description: 'Define deploy modes ?')]
12                 def user_input = input(message : 'Enter deploy target : ', parameters: [string(defaultValue: null, description: 'Deploy target : ', name: 'Target')])
13                 deploy_target = user_input?:"
14             }
15         }
16     }
17     stage('Lakukan deployment aplikasi pada komputer server'){
18         steps{
19             script{
20                 hasil_deployment = sh(script: """cd """+auto_deployer_location+""" && python3 aplikasi_deployer.py -n """+nama_dokumen_cloud_firestore+""" -t """+deploy_target+""" -m """+deploy_mode, returnStdout: true).trim()
21             }
22         }
23     }
24     stage('Catat hasil deployment'){
25         steps{
26             script{
27                 echo "${hasil_deployment}"
28             }
29         }
30     }
31 }
32 }
```

Pada Gambar 9 adalah tampilan dari konfigurasi Jenkins untuk membuat konfigurasi *deployment* dari aplikasi *back-end* bernama ”flask_web_server” yang di *hardcode* yang terlihat pada Kode Program 5 untuk disimpan pada dokumen di *database* Cloud Firestore. Dokumen tersebut disimpan dengan identifikasi nama dari aplikasi *Back-End* pada *database* Cloud Firestore dengan lokasi dokumen “/konfigurasi_deployment/flask_web_server”. Versi aplikasi *Back-End* yang dipilih berupa *tag* “1.0.5” sebagai *Target Deploy*. Proses *deployment* pada Jenkins menampilkan tiga *stage* bernama “Buat Konfigurasi Deployment”, “Lakukan deployment aplikasi pada komputer *server*”, dan “Catat hasil deployment”. *Target deploy* disimpan pada dokumen Cloud Firestore dengan identifikasi nama aplikasi *Back-End*. Dokumen yang disimpan menyebabkan perubahan data sehingga setiap komputer *server* melalui aplikasi *listener* yang beroperasi sebagai *service* melakukan sinkronisasi data konfigurasi *deployment* untuk mengubah versi kode sumber dan melakukan *deployment* aplikasi *back-end* tersebut. Proses mencatat hasil deployment pada Jenkins digunakan untuk menampilkan dan menyimpan sebagai *logs* pada komputer *server* yang telah berhasil melakukan *deployment* aplikasi *Back-End* melalui sistem *deployer* otomatis.




Gambar 9. Tampilan Konfigurasi *Deployment* Jenkins Pipeline Aplikasi *Deployer*




Gambar 10. Tampilan Konfigurasi *Deployment* Jenkins Pipeline Aplikasi *Deployer*

Pada Gambar 10 adalah konfigurasi yang disimpan pada Cloud Firestore. Konfigurasi tersebut berisi data target *deploy* berupa *tags* yang menjadi tujuan *deployment* berupa versi aplikasi *back-end*. Aplikasi *back-end* tersebut mengarah pada versi ”1.0.5”. Konfigurasi yang disimpan menyebabkan perubahan data pada Cloud Firestore. Perubahan data

tersebut menyebabkan aplikasi *listener* yang berjalan pada setiap komputer *server* melakukan *deployment* secara otomatis. Contoh proses *deployment* aplikasi *back-end* berupa log terdapat pada Gambar 11.

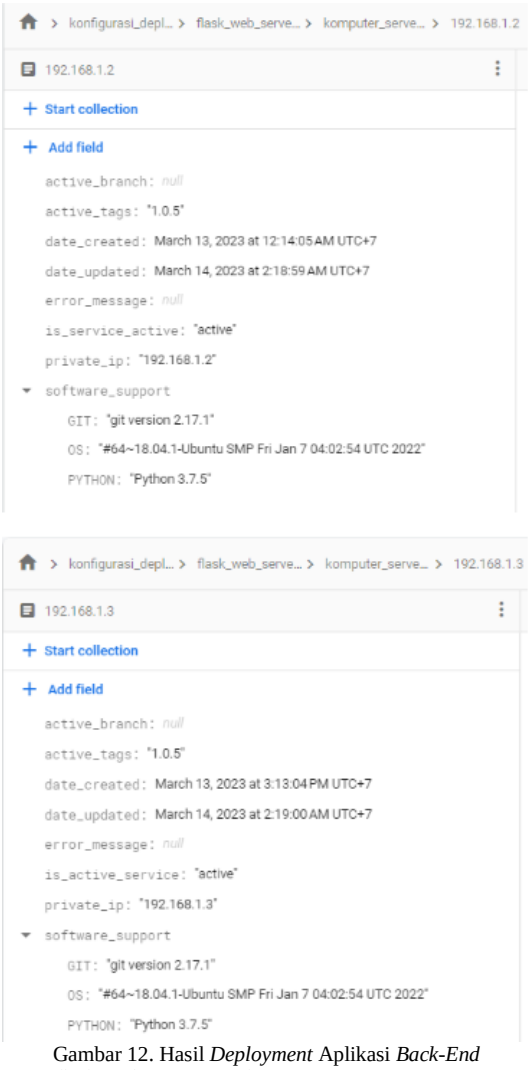
```
Mar 14 02:18:42 vm-jenkins python3: +------------------------+
Mar 14 02:18:42 vm-jenkins python3: ACTIVITY       : MODIFIED
Mar 14 02:18:42 vm-jenkins python3: DOCUMENT_ID    : flask_web_server
Mar 14 02:18:42 vm-jenkins python3: DIR_LOCAL_REPO : /opt/flask_web_server
Mar 14 02:18:42 vm-jenkins python3: +------------------------+
Mar 14 02:18:42 vm-jenkins python3:   [DATA   ] Reporting to firestore with "konfigurasi_deployment/flask_web_server/komputer_server/192.168.1.2"
Mar 14 02:18:42 vm-jenkins python3:   [INFO   ] Grabbing aplication active branch or tags = {'active_branch': "development", 'active_tags': None}
Mar 14 02:18:42 vm-jenkins python3:   [INFO   ] Data deployment location "konfigurasi_deployment/flask_web_server"
Mar 14 02:18:42 vm-jenkins python3:   [PREPARE] Target deployment : {"is_branch" : None, "is_tags" : True, "target_deploy": "1.0.5"}
Mar 14 02:18:42 vm-jenkins python3: ==========================
Mar 14 02:18:42 vm-jenkins python3:   [INFO   ] Found deployment file on '/opt/konfigurasi_deployment/flask_web_server'
Mar 14 02:18:42 vm-jenkins python3:   [INFO   ] Found repository location '/opt/flask_web_server'
Mar 14 02:18:51 vm-jenkins python3:   [SUCCESS] TARGET DEPLOY "1.0.5" as Tags Found on git ls-remote /opt/flask_web_server
Mar 14 02:18:51 vm-jenkins python3:   [INFO   ] active branch or tags local repository = {'branch': 'development', 'tags': None}
Mar 14 02:18:51 vm-jenkins python3: +---GIT CHECKOUT TAGS "1.0.5"----+
Mar 14 02:18:51 vm-jenkins python3:   [DOING  ] checkout to tags "1.0.5" on local repository
Mar 14 02:18:51 vm-jenkins python3:   [SUCCESS] path local repository "/opt/flask_web_server" found and deploy to tags "1.0.5"
Mar 14 02:18:51 vm-jenkins python3:   [DOING  ] git fetch --all --prune --tags
Mar 14 02:18:54 vm-jenkins python3:   [SUCCESS] git fetch --all --prune --tags
Mar 14 02:18:54 vm-jenkins python3:   [DOING  ] git clean -fdx
Mar 14 02:18:54 vm-jenkins python3:   [SUCCESS] git clean -fdx
Mar 14 02:18:54 vm-jenkins python3:   [DOING  ] git reset --hard
Mar 14 02:18:54 vm-jenkins python3:   [SUCCESS] git reset --hard
Mar 14 02:18:54 vm-jenkins python3:   [DOING  ] git checkout tags/1.0.5
Mar 14 02:18:54 vm-jenkins python3:   [SUCCESS] git checkout tags/1.0.5
Mar 14 02:18:54 vm-jenkins python3:   [DOING  ] resource script installer "installer.sh" on local repository
Mar 14 02:18:56 vm-jenkins python3:   [DOING  ] installing resource script
Mar 14 02:18:59 vm-jenkins python3:   [SUCCESS] install result resource script , error_message = None
Mar 14 02:18:59 vm-jenkins python3:   [DOING  ] daemon reload to service "flask_web_server.service"
Mar 14 02:18:59 vm-jenkins python3:   [DOING  ] restart service for "flask_web_server.service"
Mar 14 02:18:59 vm-jenkins python3:   [INFO   ] service "flask_web_server.service" status "active"
Mar 14 02:18:59 vm-jenkins python3:   [DOING  ] reporting to cloud firestore with "konfigurasi_deployment/flask_web_server/komputer_server/192.168.1.2"
Mar 14 02:18:59 vm-jenkins python3:   [INFO   ] Get local repository active branch or tags as deployment information = {'branch': None, 'tags': '1.0.5'}
Mar 14 02:18:59 vm-jenkins python3:   [PREPARE] Data : {'error_message': None, 'software_support': {'GIT': 'git version 2.17.1', 'PYTHON': 'Python 3.7.5', 'OS'
: '#64~18.04.1-Ubuntu SMP Fri Jan 7 04:02:54 UTC 2022'}, 'date_created': DatetimeWithNanoseconds(2023, 3, 13, 02, 14, 5, 43000, tzinfo=<UTC>), 'active_tags': '
1.0.5', 'private_ip': '192.168.1.2', 'active_branch': None, 'is_service_active':  "active", 'date_updated': DatetimeWithNanoseconds(2023, 3, 14, 02, 18, 59, 69
9000, tzinfo=<UTC>)
Mar 14 02:18:59 vm-jenkins python3:   [SUCCESS] Data reported to Cloud Firestore
Mar 14 02:18:59 vm-jenkins python3: ==========================
```

Gambar 11. Proses *Deployment* Otomatis Komputer Server 1

Gambar 11 menunjukan proses *deployment* dari aplikasi *listener* yang berjalan sebagai *service* pada komputer *server* 1 berupa *log* untuk melakukan *deployment* aplikasi *back-end* bernama ”flask_web_server”. Proses *deployment* secara otomatis dilakukan oleh aplikasi tersebut dengan mendeteksi perubahan konfigurasi berupa dokumen pada *database* Cloud Firestore. Nama aplikasi dari dokumen dengan *file* konfigurasi yang terdapat pada setiap komputer *server* akan dilakukan proses sinkronisasi. Proses tersebut adalah proses *deployment* yang mengubah dan menjalankan kode sumber yang sesuai dengan target *deploy* yang berasal dari perubahan konfigurasi dari *database* tersebut. Hasil dari peroses tersebut akan dilaporkan ke *database* dengan lokasi dokumen pada ”konfigurasi_deployment/flask_web_server/komput er_server/192.168.1.2”. Hasil tersebut terdapat pada Gambar 12.

Proses *deployment* pada komputer *server* yang lain juga menjalankan aplikasi *back-end* dengan versi yang sama. Hasil dari proses tersebut tertera pada Gambar 12. Aplikasi *listener* yang berjalan sebagai *service* pada setiap komputer *server* membuat data hasil *deployment* aplikasi *back-end* ke *database* Cloud Firestore. Data yang dibuat tersebut menyebabkan aplikasi *deployer* yang berjalan di Jenkins pada Gambar 13 mendeteksi hasil *deployment* komputer *server* yang ditampilkan dan dicatat sebagai log *deployment*.

Proses otomatis yang dilakukan menjadi lebih efisien hanya dengan melakukan sekali konfigurasi *deployment* aplikasi *back-end* pada setiap komputer *server*. Proses manual dengan masuk ke Jenkins untuk memilih Jenkins Pipeline sesuai aplikasi *back-end* yang menjadi tujuan *deployment* dan memilih target *deploy* untuk menjalankan aplikasi tersebut dengan versi kode sumber yang diinginkan dan berjalan untuk tersedia dengan beroperasi pada setiap komputer *server*.




Gambar 12. Hasil *Deployment* Aplikasi *Back-End* “flask_web_server” pada *database* Cloud Firestore


Gambar 13. Tampilan Hasil *Deployment* Aplikasi *Back-End* "flask_web_server" pada Jenkins yang Dicatat Sebagai Log *Deployment*

### 3.2. Pengujian Sistem

Pengujian sistem *deployer* GIT *Repository* pada Ubuntu menggunakan Python, Jenkins, dan Cloud Firestore menggunakan metode *Black Box Testing*. Pengujian sistem dilakukan untuk mengetahui sistem telah sesuai dengan rancangan yang direncanakan. Berikut adalah hasil dari pengujian sistem.

Tabel 1. Pengujian sistem *deployer* otomatis

| No | Skenario Pengujian | Hasil yang Diharapkan | Hasil Pengujian | Status Pengujian |
|---|---|---|---|---|
| 1 | Aplikasi *setup* dapat membuat konfigurasi *deployment* suatu aplikasi *back-end* pada komputer *server* | Konfigurasi berhasil jika aplikasi *back-end* terdapat *local repository* dan file *service* | Sesuai dengan yang diharapkan | *Valid* |
| 2 | Aplikasi *listener* melakukan *deployment* aplikasi *back-end* sesuai target *deploy* dari suatu dokumen pada *database* Cloud Firestore | Lakukan proses *deployment* jika target tersedia saja | Sesuai dengan yang diharapkan | *Valid* |
| 3 | Aplikasi *listener* menyimpan hasil *deployment* aplikasi *back-end* pada *database* Cloud Firestore | Hasil *deployment* aplikasi *back-end* disimpan dalam bentuk dokumen pada *database* Cloud Firestore | Sesuai dengan yang diharapkan | *Valid* |
| 4 | Aplikasi *Deployer* membuat target *deploy* untuk melakukan *deployment* aplikasi *back-end* pada komputer *server* yang telah di konfigurasi melalui aplikasi *setup* | Target *deploy* yang dibuat disimpan pada *database* Cloud Firestore dengan memilih salah satu data mode *deploy* berupa *tags* atau *branch* | Sesuai dengan yang diharapkan | *Valid* |
| 5 | Aplikasi *Deployer* Menampilkan hasil *deployment* pada setiap komputer *server* sesuai *timeout* pada Jenkins | Menampilkan hasil *deployment* aplikasi dari semua komputer *server* yang menjalankan aplikasi tersebut | Hanya menampilkan komputer *server* yang telah selesai melakukan *deployment* saja | *Invalid* |
| 6 | Jenkins mencatat hasil *deployment* aplikasi *back-end* dari aplikasi *deployer* sebagai log | Jenkins dapat mencatat *deployement* aplikasi *back-end* pada komputer *server* melalui aplikasi *deployer* | Sesuai dengan yang diharapkan | *Valid* |

Pengujian sistem menggunakan metode *Black Box Testing* yang telah dilakukan perlu menguji kelayakan sistem tersebut. Uji kelayakan dilakukan dengan melibatkan 20 *back-end developer* di PT XYZ. Kuesioner dibuat dengan tujuan penelitian yang berfokus untuk melakukan *deployment* otomatis GIT Repository pada sistem operasi Ubuntu menggunakan Python, Jenkins, dan Cloud Firestore.

Tabel 2. Hasil Kuesioner Skala Likert

| No | Pertanyaan | Hasil Pengujian | | | | |
|---|---|---|---|---|---|---|
| | | STS | TS | C | S | SS |
| 1 | Apakah sistem *deployer* yang dibuat dapat menyelesaikan masalah PT XYZ dalam melakukan *deployment* aplikasi *back-end* pada komputer *server* Ubuntu secara otomatis? | - | - | 1 | 10 | 9 |
| 2 | Apakah hasil *deployment* dapat dilihat melalui sistem *deployer* dari aplikasi *back-end* yang berjalan pada setiap komputer *server*? | - | - | 3 | 10 | 7 |
| 3 | Apakah sistem *deployer* dapat membantu pengguna | - | - | 3 | 11 | 6 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | untuk melihat error yang terjadi pada proses *deployment* suatu aplikasi *back-end*? | | | | | |
| 4 | Apakah sistem *deployer* aman digunakan untuk melakukan *deployment* secara otomatis? | - | - | 2 | 12 | 6 |

Tabel 3. Perhitungan Skala Likert

| No | Hasil Pengujian | | | | | Total Skor | Nilai Indeks ((Total Skor / Skor Maksimum) x 100)% |
|---|---|---|---|---|---|---|---|
| | STS (1 Skor) | TS (2 Skor) | C (3 Skor) | S (4 Skor) | SS (5 Skor) | | |
| 1 | - | - | 3 | 40 | 45 | 88 | 88% |
| 2 | - | - | 9 | 40 | 35 | 84 | 84% |
| 3 | - | - | 9 | 44 | 30 | 83 | 83% |
| 4 | - | - | 6 | 48 | 30 | 84 | 84% |

Tabel 4. Index Skala Likert

| Index Range | Hasil |
|---|---|
| 0% - 19,99% | Sangat Tidak Setuju |
| 20% - 39,99% | Tidak Setuju |
| 40% - 59,99% | Kurang Setuju |
| 60% - 79,99% | Setuju |
| 80% - 100% | Sangat Setuju |

Tabel 3 menyatakan bahwa dari keempat pertanyaan memiliki nilai index sebesar 85% dan dapat digolongkan "Sangat Setuju" mengikuti interval index skala likert pada Tabel 4. Kesimpulan persentasi penilaian yang didapat pada tabel tersebut menyatakan bahwa rancangan sistem deployer otomatis GIT Repository pada Ubuntu menggunakan Python, Jenkins, dan Cloud Firestore mampu melakukan *deployment* aplikasi back-end secara otomatis dengan efisien dan aman.

## 4. DISKUSI

Penelitian dari sistem *deployer* otomatis dibuat guna untuk memenuhi kebutuhan PT XYZ untuk melakukan *deployment* secara otomatis yang efisien dan aman untuk. Pada penelitian yang dibahas pada implementasi dan hasil merupakan hasil dari pengembangan dari penelitian sebelumnya yang telah memberikan ide berupa gagasan sistem yang dibangun dan dirancang, namun masih terdapat kekurangan untuk memenuhi kebutuhan PT XYZ yang telah diberikan pada pembahasan dan hasil.

### 4.1. Automation Provisioning Dev-Ops Website Server Menggunakan Ansible dan Vagrant

Ansible dan Vagrant merupakan perangkat lunak kembangan dari DevOps untuk melakukan proses otomatisasi dengan melakukan instalasi, *deployment*, dan update *server* [7]. Aplikasi *web server* dilakukan *deployment* secara otomatis pada setiap komputer *server* menggunakan Vagrant yang disusun dengan teknologi *load balancer* menggunakan Ansible. *Load balancing* adalah teknik dalam jaringan komputer dengan sistem kerja membagi permintaan yang masuk untuk dibagi ke sejumlah komputer *server* [4]. *Load balancer* bekerja ketika komputer *server* mencapai batas ketersediaan sumber daya maka menggunakan komputer selanjutnya agar aplikasi *web server* dapat dengan sumber daya yang cukup untuk menjalankan fitur tersebut. Vagrant dan Ansible digunakan untuk melakukan instalasi aplikasi *web server* pada suatu *Load Balancer* namun hasil *deployment* aplikasi tersebut tidak dapat terpantau setiap versi dari aplikasi yang berjalan dan tidak dapat mengetahui *error* yang terjadi. Melalui penelitan sistem *deployer* otomatis menggunakan Jenkins dapat membatasi hak akses pengguna yang ingin melakukan *deployment* aplikasi *back-end* dan dapat menampilkan hasil *deployment* aplikasi *back-end* pada setiap komputer *server* untuk dicatat sebagai log.

### 4.2. Continuous and Integrated Software Development using DevOps

Perancangan sistem *deployer* berasal dari ide dari penelitan ini berdasarkan arsitektur yang dibuat dengan satu komputer sebagai *puppet master* yang memberikan perintah ke setiap komputer bawahannya sebagai *puppet agent* [10]. Implementasi sistem *deployer* menggunakan *Virtual Machine* yang terdapat aplikasi Jenkins untuk membuat dan memberikan perintah mengoperasikan aplikasi *back-end* pada setiap komputer *server* yang telah menjalankan sistem tersebut. Pada penelitian sistem *deployer* menggunakan *Virtual Machine* menjalankan Jenkins sebagai *puppet master* untuk memberikan perintah pada setiap komputer melalui sistem *deployer* sebagai *puppet agent*.

## 5. KESIMPULAN

Sistem *deployer* pada sistem operasi Ubuntu dapat membantu proses *deployment* aplikasi *back-end* pada setiap komputer *server* di PT XYZ secara otomatis, efisien dan aman. Kesimpulan tersebut di dukung dengan uji kelayakan sistem melalui kuesioner yang dikalkulasi skala likert dengan memperoleh persentase sebesar 85% dan digolongkan "Sangat Setuju". Proses *deployment* dengan sistem *deployer* menggunakan Jenkins dilakukan oleh DevOps Engineer dapat membuat target *deploy* berupa versi aplikasi yang menjadi tujuan berupa data konfigurasi *deployment* yang disimpan pada Cloud Firestore. Data yang disimpan menyebabkan semua komputer *server* melakukan sinkronisasi data melalui sistem *deployer* untuk merubah versi aplikasi *back-end* sesuai dengan data tersebut pada setiap komputer server dengan perintah GIT dan menjalankan aplikasi secara otomatis. Hasil *deployment* aplikasi *back-end* dilaporkan sebagai data konfigurasi *deployment* pada komputer server yang disimpan pada Cloud Firestore. Data tersebut

kemudian ditampilkan pada Jenkins dan dicatat sebagai log.

Kelemahan pada sistem *deployer* otomatis terdapat pada fitur yang hanya menampilkan hasil *deployment* aplikasi *back-end* yang telah mencapai tahap proses akhir sesuai dengan *timeout* yang telah ditentukan. Proses *deployment* aplikasi *back-end* yang belum mencapai tahap akhir sesuai dengan *timeout* yang telah ditentukan user tidak dapat ditampilkan pada Sistem *deployer* melalui aplikasi *deployer*.

## DAFTAR PUSTAKA


[1] T. Xia, S. Bhardwaj, P. Dmitriev, and A. Fabijan, "Safe Velocity: A Practical Guide to Software Deployment at Scale using Controlled Rollout," *Proc. - 2019 IEEE/ACM 41st Int. Conf. Softw. Eng. Softw. Eng. Pract. ICSE-SEIP 2019*, pp. 11–20, 2019, doi: 10.1109/ICSE-SEIP.2019.00010.

[2] B. Johnson, *Essential Visual Studio 2019*. Berkeley: Apress, 2020. doi: 10.1007/978-1-4842-5719-7.

[3] M. Macak, D. Kruzelova, S. Chren, and B. Buhnova, "Using process mining for Git log analysis of projects in a software development course," *Educ. Inf. Technol.*, vol. 26, no. 5, pp. 5939–5969, 2021, doi: 10.1007/s10639-021-10564-6.

[4] S. D. Riskiono and D. Darwis, "Peran Load Balancing Dalam Meningkatkan Kinerja Web Server Di Lingkungan Cloud," *Krea-TIF*, vol. 8, no. 2, p. 1, 2020, doi: 10.32832/kreatif.v8i2.3503.

[5] P. Kirkbride, *Basic Linux Terminal Tips and Tricks*. New York: Apress, 2020. doi: 10.1007/978-1-4842-6035-7.

[6] A. S. Shitole and I. Priyadarshini, "Understanding Software Development with DevOps," *J. Adv. Softw. Eng. Test.*, vol. 4, no. 2, pp. 1–3, 2021, doi: 10.5281/zenodo.5118765.

[7] N. Evianti, A. Mulyana Winhandar, and A. Kurniawan, "AUTOMATION PROVISIONING DEV-OPS WEBSITE SERVER MENGGUNAKAN ANSIBLE DAN VAGRANT," *J. Nas. Inform.*, vol. 2, no. 2, pp. 72–91, 2021, doi: 10.55122/junif.v2i2.334.

[8] Y. F. Aladina, A. Bhawiyuga, R. A. Siregar, and P. H. Trisnawan, "Penerapan Mekanisme Continuous Deployment dalam Pengembangan dan Pembaruan Perangkat Lunak Sistem Benam Berbasis Internet of Things," *J. Teknol. Inf. dan Ilmu Komput.*, vol. 9, no. 3, p. 647, 2022, doi: 10.25126/jtiik.2022935750.

[9] E. Simanjuntak and N. Surantha, "Multiple time series database on microservice architecture for IoT-based sleep monitoring system," *J. Big Data*, vol. 9, no. 1, 2022, doi: 10.1186/s40537-022-00658-4.

[10] A. Aayush, G. Subhash, and C. Tanupriya, "Continuous and Integrated Software Development using DevOps," in *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, IEEE, 2018, pp. 290–293. doi: 10.1109/icacce.2018.8458052.

[11] Á. Révész and N. Pataki, "Visualisation of Jenkins Pipelines," *Acta Cybern.*, vol. 25, no. 2, pp. 877–895, 2021, doi: 10.14232/actacyb.284211.

[12] Sugiyono, *Metode Penelitian Kuantitatif, Kualitatif dan R&D*. Bandung: Alfabeta, 2010.

[13] D. I. Permatasari *et al.*, "Pengujian Aplikasi menggunakan metode Load Testing dengan Apache JMeter pada Sistem Informasi Pertanian," *J. Sist. dan Teknol. Inf.*, vol. 8, no. 1, p. 135, 2020, doi: 10.26418/justin.v8i1.34452.

[14] S. Al-Fedaghi, "Validation: Conceptual versus Activity Diagram Approaches," *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 6, pp. 287–297, 2021, doi: 10.14569/IJACSA.2021.0120632.

[15] V. H. Pranatawijaya, W. Widiatry, R. Priskila, and P. B. A. A. Putra, "Penerapan Skala Likert dan Skala Dikotomi Pada Kuesioner Online," *J. Sains dan Inform.*, vol. 5, no. 2, pp. 128–137, 2019, doi: 10.34128/jsi.v5i2.185.