LAPORAN AKHIR PROYEK
PENGEMBANGAN PERANGKAT LUNAK SCALABLE

Disusun oleh :

1. **Hafid Sasayuda Ambardi (23/519841/PA/22308)**
2. **Joseph Greffen Komala (23/519909/PA/22318)**
3. **David Neilleen Irvinne (23/517639/PA/22199)**
4. **Nugroho Adi Susanto (23/ 520312 /PA/22367)**

PROGRAM STUDI ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS GADJAH MADA

2025

I. Pendahuluan

1. Overview

Aplikasi ini merupakan sebuah platform pembelajaran digital yang dirancang khusus untuk membantu pengguna meningkatkan keterampilan dalam bahasa Inggris. Untuk mencapai tujuan ini, platform dibangun di atas arsitektur microservices yang modular dan dapat diskalakan.

Fitur utama dari platform ini adalah pemanfaatan kecerdasan buatan (AI) yang mampu menganalisis audio pengguna. Ketika pengguna berlatih berbicara, AI akan memberikan umpan balik (feedback) secara langsung terkait pelafalan, tata bahasa, dan aspek lainnya, menciptakan pengalaman belajar yang interaktif dan efektif.

Untuk mendukung fungsi utama tersebut, platform ini terdiri dari empat layanan inti yang saling terintegrasi sebagai berikut:

- Layanan AI Feedback: Layanan ini menjadi fitur utama dari platform pembelajaran ini. Tanggung jawab utama layanan ini adalah untuk menyediakan penilaian otomatis dan umpan balik secara *real-time* terhadap kemampuan berbicara (*speaking*) pengguna. Setelah memberikan umpan balik, layanan ini akan mengirimkan log hasil penilaian ke *Progress Tracking Service* untuk dicatat sebagai bagian dari riwayat belajar pengguna.
- Layanan Pelacakan Progres (Progress Tracking Service): Layanan ini bertindak sebagai pusat data personal pengguna. Semua aktivitas belajar, beserta hasilnya seperti skor, durasi, dan umpan balik dari AI, akan dicatat secara sistematis. Berdasarkan data historis ini, layanan juga mampu memberikan rekomendasi pembelajaran yang dipersonalisasi untuk membantu pengguna fokus pada area yang perlu ditingkatkan.
- Layanan Autentikasi (Authentication Service): Untuk menjamin keamanan data dan personalisasi, platform ini dilengkapi dengan layanan autentikasi. Layanan ini mengelola identitas pengguna, memastikan bahwa hanya pengguna yang sah yang dapat mengakses data dan riwayat belajar mereka.
- API Gateway: Bertindak sebagai pintu gerbang utama dari keseluruhan arsitektur. Semua permintaan dari sisi klien akan diterima oleh API Gateway terlebih dahulu, yang kemudian akan meneruskannya ke layanan internal yang sesuai (AI, Progress Tracking, atau Autentikasi). Selain itu, API Gateway juga bertanggung jawab untuk mengelola aspek skalabilitas platform, memastikan sistem dapat menangani peningkatan beban pengguna secara efisien.

Dengan arsitektur berbasis microservices ini, platform dirancang untuk menjadi modular, aman, dan dapat diskalakan, guna memberikan pengalaman belajar bahasa Inggris yang personal dan andal bagi setiap pengguna.

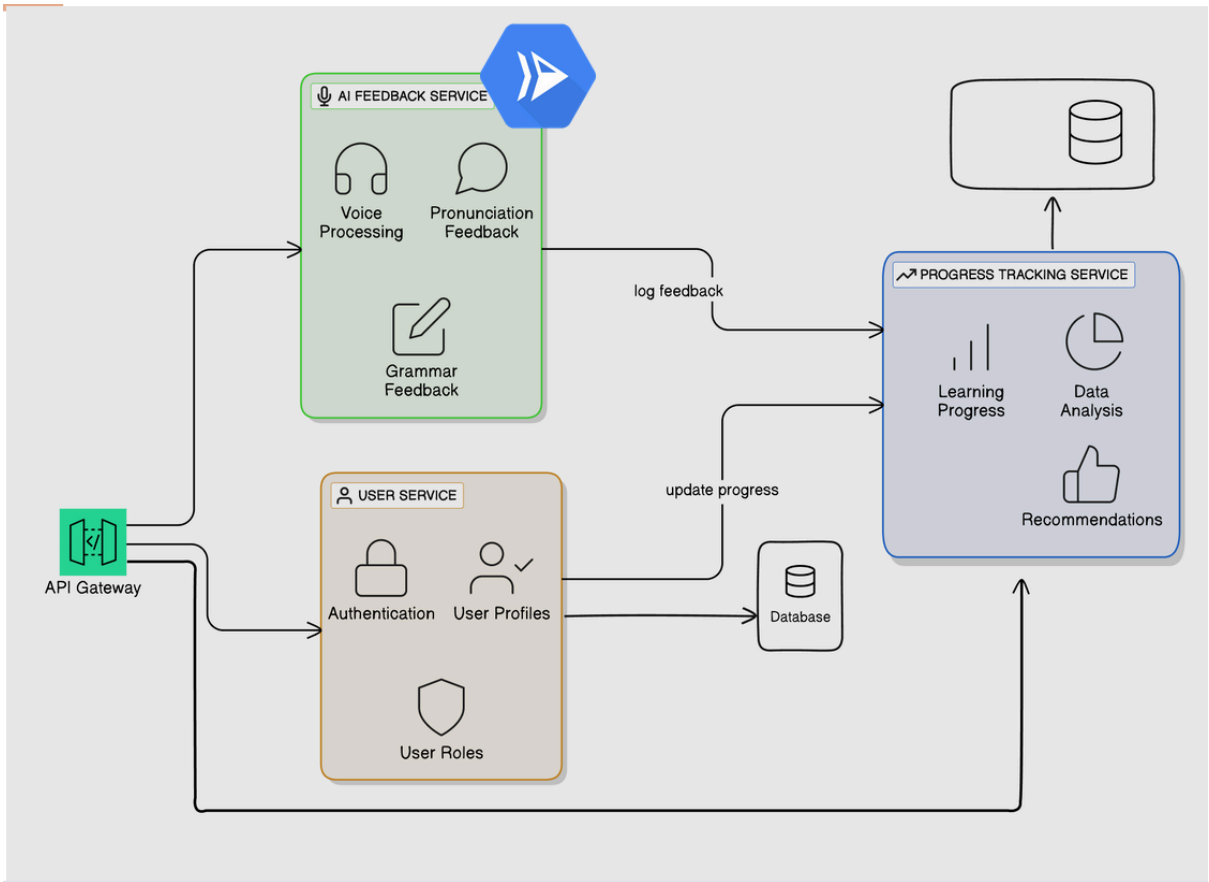
2. **Teknologi yang di gunakan**

Platform ini menggunakan teknologi modern yang disesuaikan dengan fungsi tiap layanan:

1. API Gateway: Dibangun dengan Express.js, bertugas mengatur lalu lintas permintaan dan menghubungkan klien ke layanan internal.
2. Progress Tracking Service: Menggunakan Express.js dan MongoDB untuk menyimpan riwayat belajar dan memberikan rekomendasi berdasarkan data pengguna.
3. Authentication Service: Memanfaatkan Firebase Authentication dan Firestore untuk autentikasi yang aman, cepat.
4. AI Feedback Service: Dikembangkan dengan Flask untuk memproses audio dan text serta memberi umpan balik berbasis AI secara real-time.

Seluruh layanan akan di-*dockerize* lalu dijalankan di atas Kubernetes, memungkinkan skalabilitas, deployment terotomasi, dan manajemen beban kerja yang efisien.

3. **Arsitektur Aplikasi**



Dalam arsitektur microservices ini, sistem telah dipisah menjadi beberapa bagian independen sesuai dengan domain fungsionalnya. Domain fungsi dari service dan fungsinya dibagi menjadi:

a. **User Service**
   Mengelola autentikasi pengguna, profil, dan peran pengguna.
b. **AI Feedback Service**
   Fokus pada pemrosesan data suara pengguna untuk memberikan umpan balik pengucapan dan grammar.
c. **Progress Tracking Service**
   Menyimpan dan menganalisis perkembangan belajar setiap pengguna.
d. **API Gateway**
   Perantara antara klien dan seluruh layanan di atas.

Semua *service* di atas menggunakan REST sebagai API language penghubung antara satu service dengan service lainnya. REST API menggunakan HTTP sebagai protokol transfer informasi. REST ini bersifat stateless, sehingga server tidak menyimpan session apapun pada database, pengelolaan session dilakukan menggunakan metode enkripsi dan hashing pada protokol JWT. Ketiadaan session database membuat service ini lebih mudah diskalakan.

4. Aspek Scalable

Kami memilih arsitektur microservice karena memiliki keunggulan signifikan dalam hal skalabilitas, terutama ketika diimplementasikan dengan Kubernetes sebagai orchestration platform. Keunggulan utama dari pendekatan ini adalah kemampuan untuk melakukan horizontal scaling secara independen pada setiap service berdasarkan traffic dan beban kerja masing-masing. Ketika satu service mengalami lonjakan traffic yang tinggi, hanya service tersebut yang akan ditambahkan instance tersebut tanpa mempengaruhi service lain, sehingga resource menghindari *over-provisioning* pada service yang tidak membutuhkan penambahan kapasitas. Kubernetes ini diimplementasikan pada semua servis, termasuk layanan authentication, progress tracking, AI feedback, serta API gateway.

Selain itu, beberapa teknologi yang digunakan juga mendukung fitur skalabilitas. Servis *Authentication* menggunakan Firebase Authentication dan Firestore yang bisa mendukung registrasi user tidak terbatas, *data partitioning* secara otomatis, *synchronous replication*, dan bisa di-deploy pada berbagai region. Semua hal ini memastikan servis *authentication* memiliki aspek *high-availability* dan *fault tolerant.*

Progress Tracking Service memanfaatkan MongoDB yang memiliki sharding capabilities untuk *horizontal partitioning* data di berbagai server dan *replica sets* untuk *redundancy*. Dalam arsitektur AI Feedback, semua service diimplementasikan menggunakan Kubernetes untuk memberikan konsistensi dalam management dan scaling capabilities.

II. Implementasi

1. Layanan API Gateway

Layanan API Gateway berperan sebagai pintu gerbang utama yang menerima seluruh permintaan dari sisi klien dan mendistribusikannya ke layanan internal sesuai tujuan.

Layanan ini menjadi pengatur lalu lintas komunikasi antar komponen dan menyederhanakan interaksi antara frontend dengan berbagai layanan microservices.

API Gateway ini dibangun menggunakan Express.js, framework berbasis Node.js, yang memberikan fleksibilitas tinggi dalam penanganan routing dan middleware.

Beberapa fitur utama dari layanan ini meliputi:

a. Routing Dinamis dan Terstruktur
   Semua permintaan API akan diteruskan ke layanan seperti AI Feedback, Progress Tracking, atau Authentication berdasarkan path dan metode HTTP.
b. Middleware Validasi dan Logging
   API Gateway mengintegrasikan middleware untuk logging permintaan, autentikasi JWT (untuk melindungi endpoint tertentu), serta penanganan error secara global.
c. Containerization dan Skalabilitas
   Layanan ini dikontainerisasi menggunakan Docker dan diorkestrasi oleh Kubernetes. Untuk mendukung tingginya traffic, API Gateway bisa diskalakan secara horizontal, yaitu dengan menjalankan beberapa instance (pod) secara paralel. Tetapi, meskipun terjadi penambahan jumlah pod saat scaling, alamat endpoint yang diakses klien tetap sama. Hal ini karena layanan diekspos ke luar melalui NodePort, dan Kubernetes akan secara otomatis mendistribusikan permintaan ke salah satu instance yang tersedia di balik satu alamat IP service dan port publik yang tetap. Dengan demikian, pengguna tidak perlu mengetahui atau mengakses setiap pod secara langsung, cukup melalui satu endpoint stabil yang dipetakan oleh service Kubernetes.

Dengan pendekatan ini, API Gateway menjadi simpul utama yang menghubungkan seluruh microservices dengan efisien, menjaga keamanan, serta mendukung pengelolaan skala besar secara optimal.

2. Layanan User Authentication

Service user authentication adalah *microservice* yang dibuat dengan Node.js, Express.js, dan TypeScript, memanfaatkan PostgreSQL untuk penyimpanan data melalui supabase. Layanan ini mendukung autentikasi berbasis JWT. JWT pada dasarnya adalah protokol autentikasi berbasiskan enkripsi. JWT menyimpan data session pada token message yang akan diberikan user saat user ingin mengakses service. Hal ini membuat sebagian besar kinerja komputasi berada pada user dan service lain. User hanya perlu mengirim token yang sudah diberikan service user dan service lain hanya perlu mendekripsi token jwt berdasarkan secret key untuk melakukan otorisasi. Hal ini membuatnya scalable dan bisa menangani karena database hanya di hit saat ada user yang login atau refresh token.

Titik akhir API mencakup registrasi pengguna, login, request token baru menggunakan refresh token, dan fungsi pengaturan ulang password. Alur autentikasi mengikuti *best practices*, memvalidasi kredensial, menerbitkan JWT, dan memverifikasi token untuk rute yang dilindungi. Layanan ini di kontainerisasi dengan Docker, memastikan portabilitas, dan

mencakup dokumentasi API yang komprehensif. Arsitektur ini memastikan solusi autentikasi yang aman, dapat diskalakan, dan ramah bagi *developer*.

3. Layanan AI

AI Feedback Service ini memberikan umpan balik kepada pengguna mengenai teks Bahasa Inggris yang dikirimkan. Layanan ini mampu mendeteksi hal seperti *filler words*, penggunaan tanda baca yang salah, dan kapitalisasi teks yang salah. Layanan ini dibangun menggunakan framework Flask dengan Python sebagai bahasa pemrograman utama, dilengkapi dengan Flask-CORS untuk menangani cross-origin requests. Sistem logging terintegrasi memastikan monitoring dan debugging yang efektif. Docker digunakan untuk teknologi kontainerisasi, memungkinkan deployment yang konsisten di berbagai macam environments.

Layanan ini dirancang sebagai stateless, di mana semua pemrosesan dilakukan secara real-time tanpa menyimpan status sesi atau data pengguna secara internal, sehingga setiap instance layanan dapat beroperasi secara independen.

Implementasi layanan ini dibagi menjadi tiga fitur utama:

1. Layanan dilengkapi dengan endpoint health check pada /_ah/health yang memungkinkan Kubernetes dan load balancer untuk memantau status kesehatan layanan secara real-time, memastikan high availability dan automatic failover capabilities.
2. Fitur feedback yang menganalisis teks input dari pengguna dan memberikan koreksi grammar serta saran perbaikan. Melalui endpoint API POST /speech2text, layanan menerima teks dalam format JSON dan melakukan serangkaian pemrosesan untuk memperbaiki bahasa yang salah. Outputnya berupa sentence_pairs (array pasangan kalimat asli dan hasil koreksi dengan distance score) dan stats (statistik komprehensif termasuk average distance, jumlah kalimat yang dikoreksi, dan total kalimat yang diproses).

4. Layanan Progress Tracking

Progress Tracking Service bertanggung jawab sebagai pusat pencatatan, pemrosesan, dan penyajian data terkait kemajuan belajar setiap pengguna. Layanan ini dirancang sebagai *stateless*, di mana semua data pengguna disimpan secara eksternal di dalam database MongoDB, sehingga setiap *instance* layanan tidak menyimpan status sesi apa pun.

Skalabilitas layanan ini dicapai melalui *containerization* menggunakan Docker. *Image* yang dihasilkan kemudian siap untuk diorkestrasi oleh Kubernetes melalui Dockerhub, yang dalam arsitektur ini dikelola melalui API Gateway, untuk menciptakan sistem yang dapat diskalakan secara horizontal.

Implementasi layanan ini dibagi menjadi tiga fitur utama:

1. Mencatat Aktivitas Belajar Pengguna
   Fitur ini bertugas mengumpulkan setiap aktivitas belajar pengguna dari layanan lain. Melalui *endpoint* API /api/progress/activityLog, data aktivitas, seperti hasil umpan balik dari *AI Service,* akan disimpan ke database MongoDB.
2. Memberikan Daftar Aktivitas Pengguna
   Fitur ini menyediakan riwayat belajar bagi pengguna. Sebuah *endpoint* API GET /api/progress/activityLog/:userId/api/progress/activityLog disediakan untuk memungkinkan *frontend* atau layanan lain meminta data riwayat aktivitas untuk pengguna tertentu. *Endpoint* ini melakukan *query* langsung ke MongoDB berdasarkan userId untuk mengambil data, dan mengembalikannya sebagai respons tanpa menyimpan status apapun secara internal.
3. Membuat Rekomendasi Belajar Pengguna
   Berdasarkan histori data yang sudah terkumpul di MongoDB, *endpoint* API api/progress/recommendation/:userId menjalankan proses analisis untuk menghasilkan rekomendasi materi belajar yang dipersonalisasi.

Layanan ini dibangun menggunakan framework backend Express.js, dengan MongoDB sebagai database, dan Docker untuk teknologi kontainerisasi.

5. Dockerhub + Kubernetes

Setiap layanan dalam arsitektur ini dikemas (containerized) menjadi sebuah Docker image, yang menjadikannya unit yang portabel dan konsisten di berbagai lingkungan. Container-container ini kemudian siap untuk diorkestrasi oleh Kubernetes, di mana dalam sistem ini, API Gateway bertindak sebagai titik masuk sekaligus pengelola orkestrasi.

Untuk mencapai skalabilitas otomatis, fitur Horizontal Pod Autoscaler (HPA) dari Kubernetes dimanfaatkan. HPA dikonfigurasi untuk secara terus-menerus memonitor penggunaan sumber daya setiap layanan. Jika rata-rata penggunaan sumber daya (memori dan CPU) mencapai ambang batas 70%, HPA akan secara otomatis melakukan *scale-up* dengan menambah jumlah instance (pod) untuk menangani peningkatan beban.

Karena semua layanan dirancang sebagai stateless, dengan semua data disimpan secara eksternal di database, proses scaling ini dapat berjalan dengan aman. Kubernetes dapat menambah atau mengurangi jumlah replika container secara dinamis tanpa resiko kehilangan data atau inkonsistensi status, memastikan platform tetap responsif dan efisien.

III. Analisis Skalabilitas

1. Testing Skalabilitas

a. *Script* Pengujian

Untuk menganalisis dan menguji skalabilitas platform, kami mengimplementasikan sebuah skrip pengujian beban (load testing) menggunakan framework k6. Skrip ini dirancang untuk mensimulasikan akses dari sejumlah pengguna secara bersamaan ke beberapa layanan utama melalui API Gateway.

b. Konfigurasi Beban Pengujian

Skenario pengujian ini dijalankan dengan konfigurasi sebagai berikut:

1. Virtual Users (VUs)**:** 20 pengguna virtual disimulasikan untuk mengakses sistem secara serentak.
2. Durasi: Pengujian berlangsung selama 30 detik.
3. Pacing: Setiap pengguna virtual akan jeda selama **2 detik** setelah menyelesaikan satu siklus permintaan, yang berarti setiap pengguna akan mengirimkan satu set permintaan baru kira-kira setiap 2 detik.

c. Skenario Alur Pengguna

Setiap pengguna virtual akan menjalankan serangkaian aksi yang mencerminkan alur penggunaan aplikasi secara umum. Skenario ini melibatkan pemanggilan empat endpoint yang berbeda secara berurutan:

1. Login Pengguna (POST /auth/api/auth/login): Mensimulasikan proses autentikasi dengan mengirimkan email dan password untuk mendapatkan akses.
2. Analisis Teks oleh AI (POST /ai/speech2text): Mensimulasikan fitur inti aplikasi, di mana pengguna mengirimkan teks untuk dianalisis oleh layanan AI. Permintaan ini menggunakan token autentikasi statis untuk otorisasi.
3. Rekomendasi Progres (GET/progress/api/progress/recommendation/123): Mensimulasikan permintaan pengguna untuk mendapatkan rekomendasi belajar dari Progress Tracking Service. Permintaan ini juga memerlukan otorisasi.
4. Pengecekan Kesehatan Sistem (GET /health): Sebuah permintaan sederhana untuk memeriksa status kesehatan umum dari API Gateway.

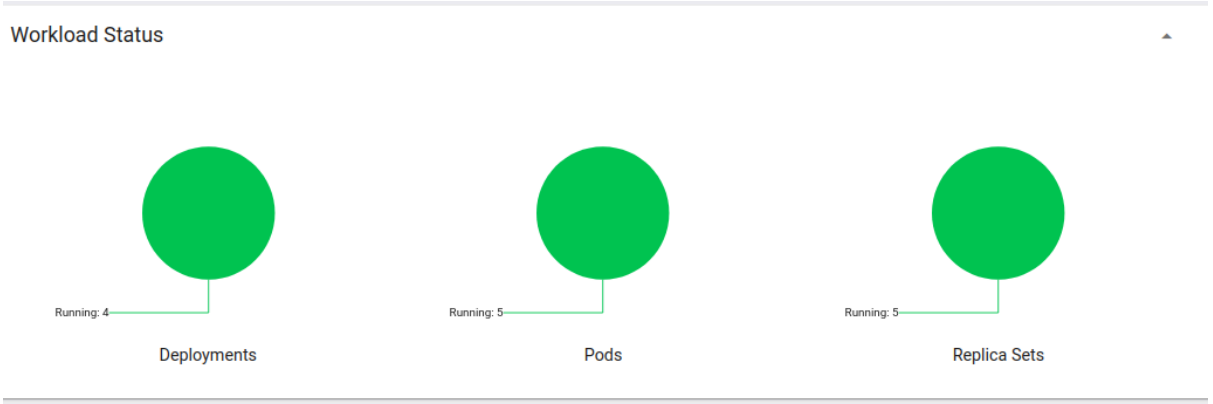
d. Tujuan Pengujian

Tujuan utama dari skrip ini adalah untuk menghasilkan beban yang cukup pada sistem guna memicu mekanisme *scaling* otomatis dari Kubernetes, yaitu Horizontal Pod Autoscaler (HPA). Dengan memonitor metrik seperti penggunaan CPU dan memori selama pengujian 30 detik ini, kami dapat menganalisis pada titik beban mana sistem melakukan scale-up (menambah jumlah *instance*/pod) dan kapan sistem melakukan scale-down (mengurangi jumlah *instance*/pod) setelah beban kembali normal.

e. Hasil Pengujian

i. Sebelum Dilakukan Pengujian

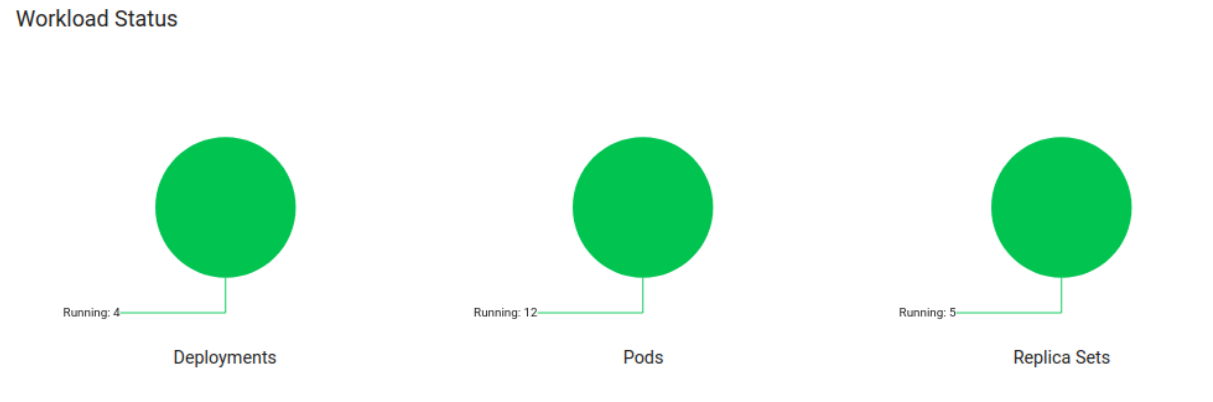
```
afen@afen:~/Scalable/api_gateway$ kubectl get hpa
NAME                        REFERENCE                              TARGETS                        MINPODS   MAXPODS   REPLICAS   AGE
ai-service-hpa              Deployment/ai-service                  cpu: 10%/70%, memory: 26%/75%  1         10        1          6h32m
api-gateway-hpa             Deployment/api-gateway                 cpu: 10%/70%, memory: 36%/70%  1         5         1          7h51m
auth-service-hpa            Deployment/auth-service                cpu: 10%/70%, memory: 50%/70%  1         10        2          7h33m
progress-tracking-service-hpa   Deployment/progress-tracking-service   cpu: 20%/70%, memory: 26%/75%  1         10        1          6h32m
afen@afen:~/Scalable/api_gateway$ S
```



Sebelum dilakukan pengujian, sistem Kubernetes berada dalam kondisi stabil dengan total 5 pod aktif yang tersebar pada 4 deployment. Masing-masing layanan (api-gateway, auth-service, dan progress-tracking-service,ai-service) memiliki konfigurasi Horizontal Pod Autoscaler (HPA) dengan jumlah pod awal sebanyak 1, kecuali auth-service yang memiliki 2 pod. Kondisi ini menunjukkan bahwa setiap layanan berjalan pada jumlah pod minimum sesuai konfigurasi sebelum dilakukan uji beban.

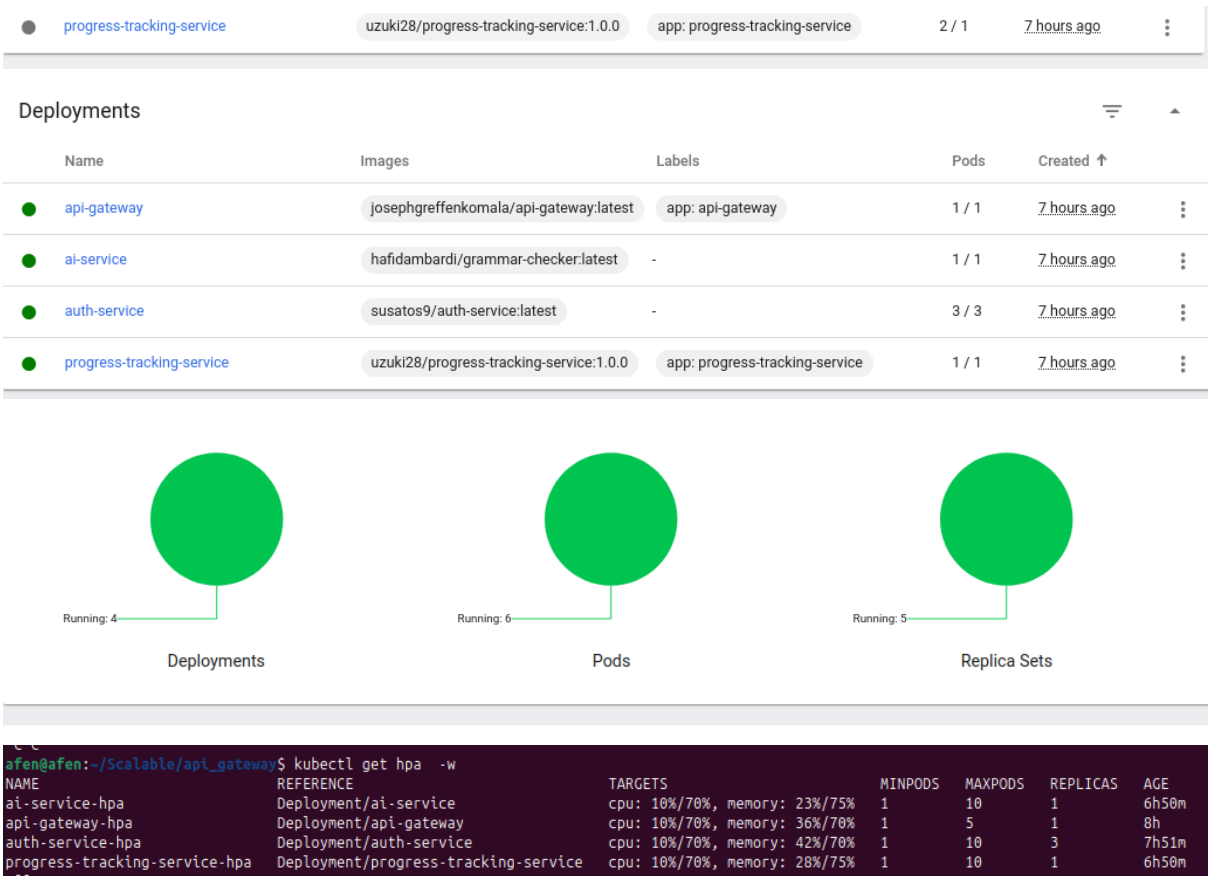
ii. Pengujian (*Scale Up*)

```
afen@afen:~/Scalable/api_gateway$ kubectl get hpa  -w
NAME                        REFERENCE                              TARGETS                        MINPODS   MAXPODS   REPLICAS   AGE
ai-service-hpa              Deployment/ai-service                  cpu: 20%/70%, memory: 21%/75%  1         10        2          6h38m
api-gateway-hpa             Deployment/api-gateway                 cpu: 30%/70%, memory: 36%/70%  1         5         3          7h57m
auth-service-hpa            Deployment/auth-service                cpu: 30%/70%, memory: 47%/70%  1         10        5          7h39m
progress-tracking-service-hpa   Deployment/progress-tracking-service   cpu: 40%/70%, memory: 31%/75%  1         10        2          6h38m
```



Saat pengujian dilakukan, terjadi peningkatan jumlah pod secara otomatis akibat aktifnya mekanisme autoscaling (HPA) pada setiap layanan. Total pod meningkat menjadi 12, API-Gateway memiliki 3 pod, AI Service memiliki 2 pod, Auth Service memiliki 5 pod dan Progress Tracking Service memiliki 2 pod. Hal ini menunjukkan bahwa beban kerja layanan meningkat, sehingga HPA menambahkan jumlah pod untuk menjaga performa sesuai target pemakaian CPU dan memori yang telah ditentukan

iii. Setelah Pengujian (*Scale Down)*




```
afen@afen:~/Scalable/api_gateway$ kubectl get hpa  -w
NAME                        REFERENCE                          TARGETS                        MINPODS   MAXPODS   REPLICAS   AGE
ai-service-hpa              Deployment/ai-service              cpu: 10%/70%, memory: 23%/75%   1         10        1          6h50m
api-gateway-hpa             Deployment/api-gateway             cpu: 10%/70%, memory: 36%/70%   1         5         1          8h
auth-service-hpa            Deployment/auth-service            cpu: 10%/70%, memory: 42%/70%   1         10        3          7h51m
progress-tracking-service-hpa   Deployment/progress-tracking-service   cpu: 10%/70%, memory: 28%/75%   1         10        1          6h50m
```

Setelah pengujian selesai, jumlah pod menurun kembali menjadi 6 seiring berkurangnya beban kerja pada masing-masing layanan. Hal ini menunjukkan bahwa mekanisme autoscaling (HPA) berjalan dengan baik dalam menyesuaikan jumlah pod secara dinamis. Beberapa layanan masih mempertahankan lebih dari satu pod karena sistem tidak langsung mengembalikan jumlah pod ke kondisi awal, melainkan menunggu kestabilan trafik untuk menghindari fluktuasi yang tidak perlu. Selain itu, penggunaan CPU atau memori yang masih mendekati batas target juga menjadi alasan HPA belum langsung menurunkan jumlah pod ke nilai minimum.

2. Pembagian Resource Layanan

a. Permintaan Sumber Daya Minimal

Setiap layanan memiliki jumlah sumber daya yang didapatkan oleh setiap *container* dari layanan.

- CPU **"10m"**: Setiap instance layanan dijamin mendapatkan 10 millicores (atau 1% dari satu core CPU) setiap saat. Kubernetes tidak akan menempatkan pod ini di node (server fisik) yang tidak bisa memberikan jaminan CPU sekecil ini.
- Memory **"128Mi"**: Setiap *instance* dijamin mendapatkan 128 Megabytes memori.

b. Batas Maksimal Sumber Daya

Setiap layanan memiliki jumlah sumber daya maksimal yang didapatkan

- CPU **"40m":** Penggunaan CPU oleh satu *instance* dibatasi agar tidak melebihi 40 millicores (4% dari satu core CPU).
- Memory **"256Mi"**: Penggunaan memori oleh satu *instance* tidak boleh melebihi 256 Megabytes.

c. Bagaimana Ini Menunjukkan Sistem yang Scalable?

Konfigurasi requests dan limits ini adalah fondasi penting untuk skalabilitas otomatis, terutama saat menggunakan Horizontal Pod Autoscaler (HPA).

1. Memberi HPA Target yang Jelas: HPA bekerja dengan memonitor penggunaan sumber daya saat ini dan membandingkannya dengan nilai yang ditentukan di requests. Misalnya, bisa mengatur HPA untuk melakukan scale-up (menambah instance) jika rata-rata penggunaan CPU melebihi 70% dari requests CPU (yaitu, 70% dari 40m).
2. Menentukan Momen untuk Scale-Up: Ketika beban pada layanan meningkat, penggunaan CPU untuk satu instance akan naik dari 10m mendekati batas 40m. HPA akan mendeteksi peningkatan ini. Jika rata-rata penggunaan CPU dari semua instance sudah mencapai target (misalnya, 70% dari 40m), HPA akan secara otomatis menambah instance baru untuk membagi beban tersebut.
3. Menentukan Momen untuk Scale-Down: Sebaliknya, setelah pengujian selesai dan beban kembali normal, penggunaan CPU akan turun drastis. HPA akan melihat bahwa sumber daya yang digunakan jauh lebih rendah dari yang dialokasikan. Setelah periode waktu tertentu, HPA akan mengurangi jumlah instance kembali ke jumlah minimal untuk menghemat sumber daya.

IV. Pembagian Tugas

Sesuai dengan jumlah layanan pada aplikasi ini, kami membagi setiap layanan ke satu anggota kelompok dengan bagian-bagian berikut:

1. API Gateway
   Dikerjakan oleh Joseph Greffen Komala pada *repository* Github: https://github.com/josephgreffenkomala/api_gateway
2. AI Service:
   Dikerjakan oleh Hafid Sasayuda Ambardi pada repository Github: https://github.com/HafidAmbardi/AIfeedackservice
3. User Authentication
   Dikerjakan oleh Nugroho Adi Susanto pada *repository* Github: https://github.com/csfighterrr/scalable-auth
4. Progress Tracking
   Dikerjakan oleh David Neilleen Irvinne pada *repository* Github: https://github.com/david-irvinne/progress_tracking_service.