Cloud Platform Engineer Assessment

You need to plan and architect the migration of an existing analytics platform to a modern cloud-based solution. Your solution must preserve the current system's strengths while resolving critical security and operational inefficiencies.

ℹ️ Sample repo: https://github.com/lcardno10/dashboard-sample

Deliverables

1. System Architecture & Migration Plan - Recommended time: 1hr
    - Architecture diagram
    - Migration strategy with phases and timeline
    - Any additional relevant information you consider valuable
2. Implement an MVP - Recommended time: 1hr
    - Using the sample repo, produce an MVP for one aspect of your solution. At the end of the 2 hours, you will have a 30 minute paired session to go through your solutions.

ℹ️ Note: It is really important that you attempt both of the deliverables so that we can assess
both sets of skills.

Current System

The current system is a monolithic application written in Python using the framework PlotlyDash. It’s hosted on a single container that runs using AWS ECS Fargate.

Development Process

- Data scientists are domain specific, they develop dashboards on specific topics, each
  of these has its own repo.
- They run the dashboards locally on their laptops against data in CSV files on their
  machines.

Dashboard Deployment Process

- A team member, in charge of deployment, manually copies data scientist code into a
  subdirectory of the main monolithic repository.
- This monolithic repository is one large plotly dash application which has subpages for
  each of the different repos.

Data Deployment Process

- Data scientists email CSV files to a designated deployment person
- CSV files are manually copied to an S3 bucket
- All data files loaded into application memory at startup, these persist in memory
  throughout the application runtime

Auth

- Authentication: Python-based password system with hashed passwords stored in database

- Authorisation: Custom internal logic, within the monolith, controls user access to
  specific sub-paths

Design Requirements

- Preserve developer experience, unified user interface and granular access control.
- Implement enterprise-grade security and authentication.
- Automate deployment with CI/CD best practices.
- Turn it into a scalable and maintainable architecture.
- Needs to support 20 Data Scientists, who run at least 5 dashboards each, with the
  ability to scale further.
- The expected user base of the dashboard is of the order of 100 daily government
  users.
- Any cloud hosting needs to be done in the UK

Resources Available

To help with constraining your design of the platform, you can have the following available to
you:

- The cloud platform of your choice.
- Github, including github actions.
- Any other programs/pieces of software that you think are necessary for your design.

Assessment Criteria

Your solution will be evaluated on:

- Technical architecture and design decisions
- Security implementation and considerations
- Migration strategy feasibility
- Code quality and implementation approach
- Documentation and presentation clarity

Guidance

- Please work like you would normally, use any AI assistants that you find useful
- Do not worry about completing this in its entirety. We care about seeing your thought
  process, not about getting any 'right' answer.

Solution

Architecture

This repository demonstrates a modern, cloud-native architecture for a Plotly Dash analytics platform. It replaces the existing monolithic system with a scalable, secure, and maintainable solution, deployed in GCP while avoiding vendor lock-in.

Migration plan

The current risks and proposed solution:

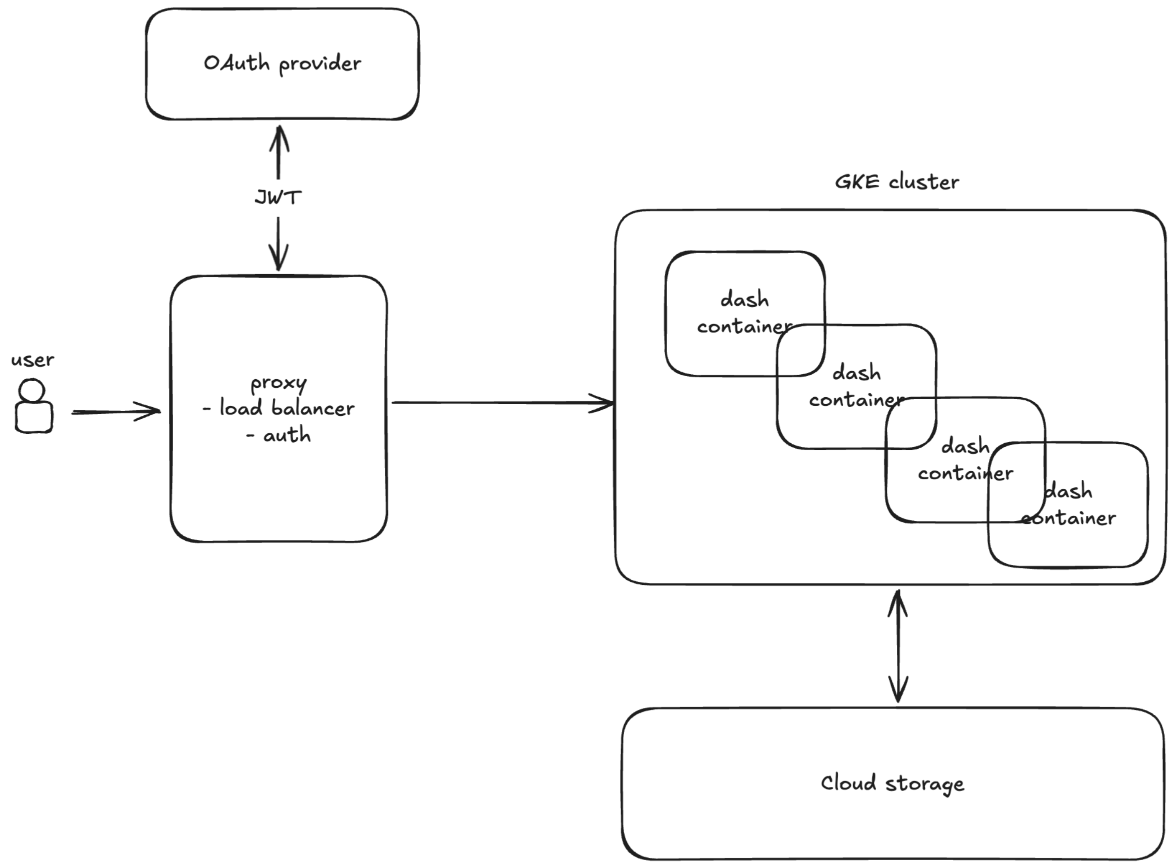
- Deployment process
    - Manual work increases risk of mistakes

    - Separate deployment process is slower. Even if there’s version control and CICD by the deployment team, they don’t have context and it’s an extra process and a bottleneck
    - security risk sending data on email
    - **SOLUTION**: CI/CD using GitHub Actions
        - developers can take ownership of process
        - code and data not passing more hands so more secure
- Data handling
    - Operational risk
    - No version control
    - No validation
    - No auditability
    - Not scalable
    - Slow and error-prone
    - **SOLUTION**: Store datasets in GCS or other cloud storage
        - Provide signed upload endpoint or automated ingestion pipeline
        - Considerations:
            - Since data is not large scale I won’t challenge handling of CSVs but if size of data grows we might need to think about a different dev process and move away from CSV towards a datalake (Bigquery). That way we can test and monitor data better and there are python packages to pull data or samples into memory for development and it’s more secure than storing data locally.
            - Since I’m more familiar with GCP I will use that as an example but I’m sure we can easily replace the services on AWS with minor tweaks. Will design it so that there’s no vendor lock in
- Monolith
    - If monolith crashes, all dashboards go down

    - All dashboards redeploy together
    - Cannot scale dashboards independently
    - Codebase becomes harder to maintain as dashboards grow
    - SOLUTION: independent containerised dashboard services
        - Each dashboard runs as its own Docker container so can be deployed separately
        - Scales independently using Kubernetes
- Auth:
    - Risky to have all user-pass on a DB
    - Difficult to control granular permissions for dashboards
    - **Solution**: External authentication OICD layer such as Okta
        - Role-based access and SSO
        - JWT can be expired in case of breach

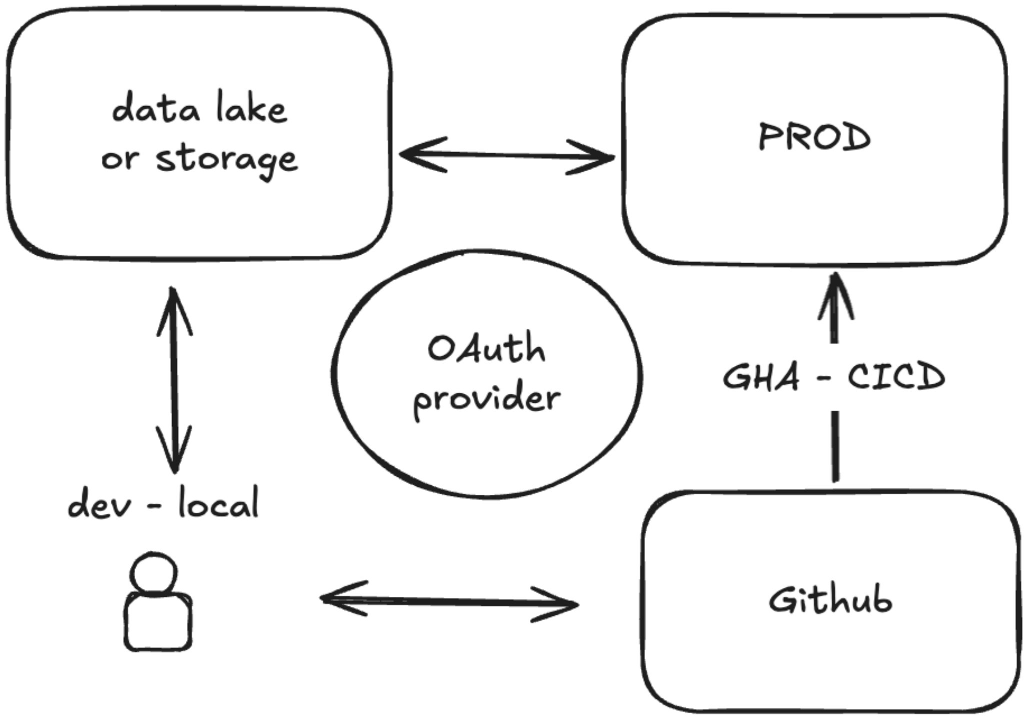
Design overview

Dashboard users and architecture



- Dashboard user hits the endpoint
- Oatuh server authenticates user for certain dashboards and hands JWT
- Proxy forwards request to GKE cluster with token
- User accesses only dashboards that are available to them
- GKE cluster talks to storage or data lake to pull data and serve dashboards

Dev process



- Developer pulls data from lake or storage
- dev works on data and dashboard locally
- pushes code to github
- once PR merges CICD is triggered to push the dashboard to prod
- OAuth so that dev only accesses the necessary data and service accounts to access lake to pull data for production

Migration Plan

Phase	Tasks	Timeline
Phase 1 – Foundation	Provision GKE, Cloud SQL, GCS; setup OIDC; Terraform infra	2–3 weeks

Phase	Tasks	Timeline
Phase 2 – Pilot Dashboard Extraction	Containerise dashboards; deploy independently; validate routing	2–3 weeks
Phase 3 – Full Dashboard Migration	Migrate remaining dashboards; decommission monolith	4–6 weeks
Phase 4 – Authentication Replacement	Integrate OIDC into monolith; replace custom passwords	2 weeks
Phase 5 – Data Pipeline Modernisation	Implement structured ingestion; Parquet conversion; lazy loading	3 weeks

Due to time and resource constraints I will not be able to set up OAuth client so I will just do follow the same password flow with Google secret manager

ℹ️ Note that I used GPT to create a boilerplate for the terraform