**Building a Modern ESG Data Pipeline: Unveiling Sustainability Metrics with AI and Cloud Technologies**

This playbook outlines the deployment and management processes for the ESG Metrics Web Application.

**Target Audience:** This playbook is intended for developers and operations personnel responsible for deploying and maintaining the application.

**Assumptions:**

* You have a Google Cloud Platform (GCP) project set up.
* You have an OpenAI API key.
* This is a prototype application with no user authentication. (**Security Note:**Implement user authentication for production environments)

**Tools:**

* Git version control system
* Google Cloud SDK (gcloud)
* Google Cloudrun for deployment
* Google bigtable for document storage

**Stages:**

1. **Development:**
   * Developers write and commit code to a Git repository.
   * Unit tests are run automatically on code commits.
   * Integration tests are run manually or automatically as needed.
2. **Testing:**
   * Automated integration tests are run against the application in a staging environment.
   * Manual testing is performed to verify functionality and user experience.
3. **Deployment:**
   * Infrastructure for the application is provisioned on GCP using Terraform. This includes:
     + Google Cloud Storage buckets
     + Google Cloud Bigtable instance
   * Docker images are built for the Flask application, Streamlit application, and OpenAI API integration script.
   * The application is deployed using cloud run.
   * Environment variables (e.g., GCP project ID, OpenAI API key) are securely stored and injected into the containers.
4. **Monitoring and Logging:**
   * Application logs are collected and monitored for errors and warnings.
   * Application health checks are implemented to ensure all components are functioning properly.
   * Infrastructure metrics (e.g., CPU, memory usage) are monitored for performance optimization.
5. **Updates and Rollbacks:**
   * New code versions are deployed following the same steps as the initial deployment.
   * A rollback strategy is defined to revert to a previous version in case of issues.

**Detailed Steps:**

**1. Development:**

* Developers clone the Git repository and set up their development environment.
* Developers write and test code according to established coding standards and best practices.
* Unit tests are run automatically using a testing framework like pytest.

**2. Testing:**

* Integration tests are defined to verify interactions between different components of the application.
* These tests can be automated using tools like Selenium or pytest-docker.
* Manual testing is performed to validate user experience and overall functionality.

**3. Deployment:**

* **Infrastructure Provisioning:**
  + Define Terraform configurations to create GCP resources (Cloud Storage buckets, Bigtable instance).
  + Use Terraform Cloud or a similar tool to manage infrastructure as code.
* **Google Cloudrun:**
  + Create Dockerfiles for each application component (Flask app, Streamlit app, OpenAI API script).
  + Define dependencies and build instructions for each Dockerfile.
* **Deployment with Cloudrun:**
  + Create a Docker Compose configuration file defining services and their dependencies.
  + Use environment variables to store sensitive information (e.g., GCP project ID, OpenAI API key).
  + Deploy the application to a container orchestration platform (e.g., Kubernetes) using Docker Compose.

**4. Monitoring and Logging:**

* Configure logging frameworks (e.g., Python logging) in each application component.
* Use a centralized logging service (e.g., Stackdriver Logging) to collect and analyze application logs.
* Implement application health checks (e.g., health endpoints) to monitor service availability.
* Set up monitoring for GCP resources using Cloud Monitoring.

**5. Updates and Rollbacks:**

* Follow the same deployment process for new code versions.
* Implement a rollback strategy using tools like Terraform workspaces or container versioning.
* This allows reverting to a previous deployment in case of issues with the new version.

**Security Considerations:**

* (**Important Note:** This is a prototype and lacks user authentication)
* Implement user authentication and authorization mechanisms for production environments.
* Securely store and manage sensitive information (e.g., API keys) using environment variables or secrets management tools.
* Regularly update software dependencies to address security vulnerabilities.

**Version Control:**

* Use Git for version control and maintain a clear history of code changes.
* Implement branching strategies for development, testing, and deployment.

**Continuous Integration/Continuous Delivery (CI/CD):**

* Consider implementing a CI/CD pipeline to automate testing and deployment processes.
* This can be achieved using tools like Jenkins or Harness.