Power BI Semantic Model Deployment - Design Document

1. Introduction

- **1.1 Project Overview** This project automates the deployment of a Power BI semantic model by reading TMDL code, serializing it into a mod, and publishing it to Power BI Premium using a .NET application. The application is containerized to run on Azure Kubernetes Service (AKS) and integrated into a CI/CD pipeline using GitLab, enabling efficient and consistent deployments. The deployment process is managed by UBS Deploy, an internal orchestration tool that collects the parameters required for deployment and securely passes them to AKS, ensuring a streamlined, controlled, and audited deployment process.

- **1.2 Objectives**

    - Automate the deployment of Power BI models using TMDL code.
    - Integrate with GitLab CI/CD pipelines to ensure continuous delivery.
    - Utilize UBS Deploy for centralized management of deployment parameters.
    - Containerize the application for consistent execution across environments.
    - Manage configurations securely and allow scalability through AKS.

- **1.3 Scope**

    - **Included**: Deployment automation, TMDL serialization, secure environment configurations, containerization on AKS, integration with UBS Deploy.
    - **Excluded**: Manual deployments, non-TMDL-based deployments, features outside of the semantic model deployment.

2. Architecture

- **2.1 System Overview** The system architecture is designed to support end-to-end automation of Power BI semantic model deployments. Key interactions are as follows:

    - Developers push TMDL code to the GitLab repository.
    - The GitLab CI/CD pipeline is triggered, which initiates the build process and packages the application into a Docker container.
    - UBS Deploy collects deployment parameters from authorized users and passes them as environment variables to the AKS deployment via Helm.
    - The containerized .NET application reads the TMDL code, processes it, and uses the Power BI REST API to deploy the semantic model to Power BI Premium.

    Architecture Diagram

The following diagram illustrates the overall architecture and workflow of the deployment process:

- **2.2 Components**
  - **.NET Application**:
    - Processes the TMDL code and interfaces with the Power BI REST API for model deployment.
    - Ensures secure handling of sensitive information via environment variables.
  - **GitLab CI/CD Pipeline**:
    - Automates the build, test, and deploy steps to streamline development and deployment processes.
  - **Docker Container**:
    - Provides a consistent and isolated environment for running the .NET application.
  - **Azure Kubernetes Service (AKS)**:
    - Hosts the Docker container, ensuring scalability, availability, and simplified management.
  - **UBS Deploy**:
    - An orchestration tool that centralizes deployment by collecting parameters and passing them securely to AKS using Helm. It also manages authentication, authorization, and audit trails.

3. Workflow

- **3.1 Development Workflow**

  1. **Developers write and update TMDL code**: Developers implement changes to the TMDL code and commit these changes to the GitLab repository.
  2. **GitLab CI/CD Pipeline**: Once code is pushed, GitLab triggers a pipeline that initiates the build, packages the .NET application into a Docker container, and prepares it for deployment.
  3. **UBS Deploy Parameter Collection**: UBS Deploy collects necessary deployment parameters (such as environment variables) from authorized users and passes them to AKS, ensuring compliance with security and operational protocols.
  4. **Deployment to AKS**: UBS Deploy executes Helm commands to deploy the containerized application on AKS, providing consistent runtime environments.

- **3.2 Deployment Workflow**
  - **Step 1**: Developer pushes TMDL code to GitLab.

- **Step 2**: GitLab CI/CD builds the Docker container and stores it in a container registry.
- **Step 3**: UBS Deploy collects deployment parameters (e.g., `PBI_SEMANTIC_MODEL_NAME`, `PBI_CONNECTION_STRING`, `SECRET_STORE_FILE`) and injects them as environment variables into the Helm deployment.
- **Step 4**: The application on AKS reads configurations and TMDL code, processes them, and deploys the semantic model to Power BI Premium.
- **Step 5**: Logs are generated for deployment activities, which can be monitored for troubleshooting or audit purposes.

4. Technical Implementation

- **4.1 Environment Configuration**
  - Environment variables are managed by UBS Deploy and passed to AKS via Helm:
    - **Key Variables**: `PBI_SEMANTIC_MODEL_NAME`, `PBI_CONNECTION_STRING`, `PBI_WORKSPACE`, `DATABRICKS_INSTANCE`, `DATABRICKS_SQL_PATH`, `DATABRICKS_DATABASE_NAME`, `CLIENT_ID`, `TENANT_ID`, `SECRET_STORE_FILE`, `TMDL_MODEL_PATH`.
  - **Secure Handling**: Sensitive information is securely stored, with UBS Deploy ensuring controlled access and compliance with security protocols.
- **4.2 Docker Container Setup** The Dockerfile uses a multi-stage build to optimize image size and runtime efficiency:
  - **Build Stage**: Restores dependencies, compiles the .NET application, and runs tests.
  - **Final Stage**: Copies the compiled application to a clean runtime image, minimizing the attack surface.
  - **Security**: Includes certificates and secure configurations for establishing trusted connections.
- **4.3 AKS Deployment**
  - **Helm Chart Configuration**: UBS Deploy utilizes Helm charts to manage AKS deployments, ensuring configurations are consistent across environments.
  - **Scaling**: The deployment can be scaled horizontally by adjusting AKS configuration, managed by UBS Deploy.
  - **Configuration Management**: Environment-specific configurations are handled via Helm values files, ensuring that deployments are consistent, repeatable, and secure.
- **4.4 Application Code Structure**
  - **ConfigurationService.cs**: Reads and applies configurations from environment variables.
  - **PowerBiRestApiClient.cs**: Manages interactions with the Power BI REST API, handling authentication and model deployment.
  - **SemanticModel.cs**: Processes TMDL code, serializes it into the required format, and prepares it for deployment.

5. UBS Deploy Integration

- **5.1 Role of UBS Deploy** UBS Deploy is an internal orchestration tool that manages deployments across various environments. It ensures controlled, secure, and compliant deployments by:
    - Collecting required parameters from developers and operations teams.
    - Authenticating and authorizing deployment requests.
    - Injecting parameters securely into deployments via Helm.
    - Tracking deployments and maintaining audit logs for compliance.

- **5.2 UBS Deploy Workflow**
    1. **Release Planning**: Developers plan release schedules and manage code check-ins through GitLab, which triggers the CI/CD pipeline.
    2. **Artifact Management**: Code artifacts are built and published to a repository (e.g., Nexus). UBS Deploy verifies the integrity and immutability of these artifacts before proceeding.
    3. **Parameter Collection**: Deployment parameters are gathered via the UBS Deploy user interface, where authorized users input or approve required settings.
    4. **Controlled Deployment Execution**: UBS Deploy passes the parameters to AKS through Helm, ensuring all configurations meet the necessary security and operational standards.
    5. **Logging and Audit**: Deployment details, including parameter values (secured), deployment status, and timestamps, are logged for monitoring and compliance.

6. Error Handling and Logging

- **6.1 Error Detection**
    - The application includes robust error handling to capture and log issues during the deployment process.
    - UBS Deploy further ensures that errors or discrepancies are flagged during parameter collection or deployment execution.

- **6.2 Logging**
    - **Application Logs**: Logs generated by the .NET application for each deployment step.
    - **Deployment Logs**: UBS Deploy maintains comprehensive logs that capture all actions, errors, and outcomes during the deployment process. These logs support troubleshooting and audit requirements.