Architecture Document Design

**Mushroom Classification Project**

## Contents

[Abstract.](#_TOC_250020) 3

1. [Introduction](#_TOC_250019) 4
   1. [Why this Architecture Design Document?.](#_TOC_250018) 4
   2. Objective and [Scope.](#_TOC_250017) 4
2. [System Overview.](#_TOC_250015) 5
   1. [Architecture Drivers](#_TOC_250014)
      1. [Business Goals](#_TOC_250014)………………………………………………………………………6
      2. Non-Functional Requirements……………………………………………………..6

2.2 [Model Architecture](#_TOC_250013) 6

1. **Tools used ………….………….………….………….………….………….………….…….**8
2. Infrastructure Architecture 10
3. Deployment Architecture….…………………………………………………………………..11

Abstract

The Mushroom Classification Project aims to develop a robust machine learning web application capable of accurately predicting the edibility of mushrooms based on their attributes. Leveraging sophisticated MLOps frameworks such as DVC, MLflow, and Docker, the project integrates seamless data management, model tracking, and deployment processes. Through a user-friendly interface, users can input mushroom characteristics and receive immediate predictions regarding their edibility. The project embodies a comprehensive approach, encompassing data fetching, preprocessing, model training, evaluation, and deployment pipelines. By adopting industry best practices and leveraging cloud infrastructure, the Mushroom Classification Project ensures scalability, reliability, and transparency in its operations. Through continuous improvement and innovation, the project seeks to enhance user experience and contribute to the advancement of machine learning applications in real-world scenarios.

1. **Introduction**

###### Why this Architecture Design Document?

The purpose of this Architecture Design (ADD) Document is to add the necessary detail to the current project description to represent a suitable model for coding. This Architecture Design Document (ADD) serves as a blueprint for the development and implementation of the mushroom classification project. Its purpose is multi-faceted and crucial for the success of the endeavor

The ADD will give:

* + - Clarity: Ensure stakeholders understand project architecture, components, and interactions clearly.
    - Alignment: Ensure project team, stakeholders, and partners are in sync.
    - Risk Mitigation: Identify and mitigate potential risks early in project development.
    - Scalability: Design architecture to accommodate future growth and increased demand.
    - Flexibility: Allow for easy modifications, enhancements, and integrations in the architecture.
    - List and describe the non-functional attributes like:
* Accuracy: Ensure precise mushroom classification results.
* Scalability: Expand capacity to handle growth.
* Performance: Optimize speed for classification tasks.
* Reliability: Minimize downtime, ensure dependability.
* Security: Protect data, prevent unauthorized access.
* Usability: Prioritize intuitive interfaces, clear instructions.
* Maintainability: Simplify upkeep, facilitate updates efficiently.
* Portability: Enable deployment across diverse environments seamlessly.
* Interoperability: Support integration with external systems seamlessly.

##### Objective and Scope

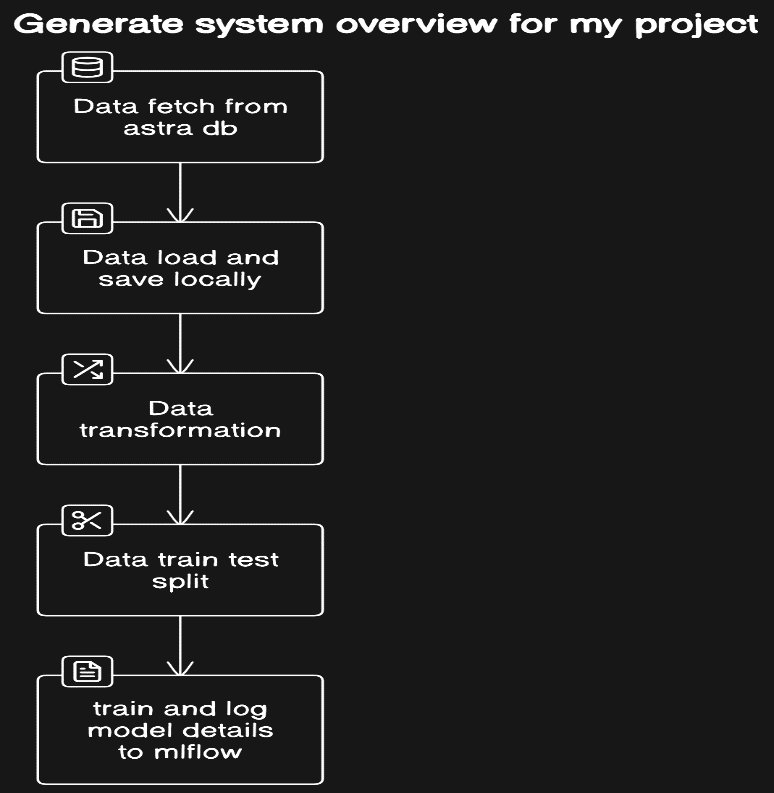
The objective of the Architecture Design Document (ADD) for the mushroom classification project is to provide a comprehensive blueprint outlining the design, components, and interactions of the system architecture. The document aims to ensure clarity, alignment, and adherence to best practices throughout the development, implementation, and evolution of the project. By defining the architecture's non-functional attributes, such as scalability, performance, security, and maintainability, the ADD aims to create a robust foundation for building a reliable and efficient mushroom classification system.

**The scope of the Architecture Design Document encompasses the following:**

* **System Architecture: Define system design components and interactions.**
* **Non-functional Requirements: Identify key performance attributes and constraints.**
* **Technology Stack: Specify essential frameworks and tools.**
* **Data Architecture: Outline data preprocessing and storage.**
* **Model Architecture: Detail machine learning model deployment.**
* **Infrastructure Architecture: Define scalable hardware and software.**
* **Security Architecture: Implement robust data protection measures.**
* **Monitoring and Logging: Track system performance and errors.**
* **Deployment Architecture: Specify deployment process and strategies.**

### System Overview

The mushroom classification system is designed to accurately categorize mushrooms into edible and poisonous categories, prioritizing user safety. It consists of several main components:



* **Data Fetch from Astra DB:** Responsible for retrieving mushroom data from the Astra DB, ensuring a reliable and consistent source of information.
* **Data Load and Save Locally:** Handles the loading and saving of mushroom data locally, facilitating efficient data processing and model training.
* **Data Transformation:** Performs data transformation tasks such as cleaning, normalization, and feature engineering to prepare the mushroom data for model training.
* **Data Train-Test Split:** Divides the transformed data into training and testing sets, enabling the evaluation of model performance on unseen data.
* **Train and Log Model Details to MLFlow:** Executes the model training process and logs important details, such as hyperparameters, metrics, and artifacts, to MLFlow for tracking and reproducibility.
* **Assign a Dev/Production/Staging Model:** Assigns the trained model to a designated environment (e.g., development, production, staging) based on its performance and readiness, ensuring seamless deployment and testing.

By orchestrating these components effectively, the mushroom classification system aims to deliver accurate and reliable predictions, thereby enhancing user safety and confidence in mushroom identification.

#### Architecture Drivers

* + 1. ***Business Goals:***

The business goals of the mushroom classification project are centered on ensuring user safety, building trust and confidence in the classification system, promoting public health awareness, and supporting regulatory compliance. By providing an accurate and reliable solution for distinguishing between edible and poisonous mushrooms, the project aims to empower individuals to make informed decisions, reduce the incidence of mushroom-related poisonings, and uphold user privacy and trust through adherence to regulatory standards such as GDPR. Through these efforts, the project seeks to enhance overall public health outcomes and foster a culture of responsible mushroom consumption.

***2.1.2 Non-Functional Requirements:***

**High Accuracy:** The system must achieve a high level of accuracy in classifying mushrooms to minimize the risk of misidentification and ensure user safety.

**Scalability:** The architecture should be designed to handle varying loads of data and user requests, ensuring scalability to accommodate potential increases in usage without compromising performance.

**Security:** Security measures must be implemented to protect user data and system integrity, including encryption of sensitive information and robust access control mechanisms. Compliance with relevant regulations such as GDPR is essential to safeguard user privacy and trust.

* 1. Model Architecture

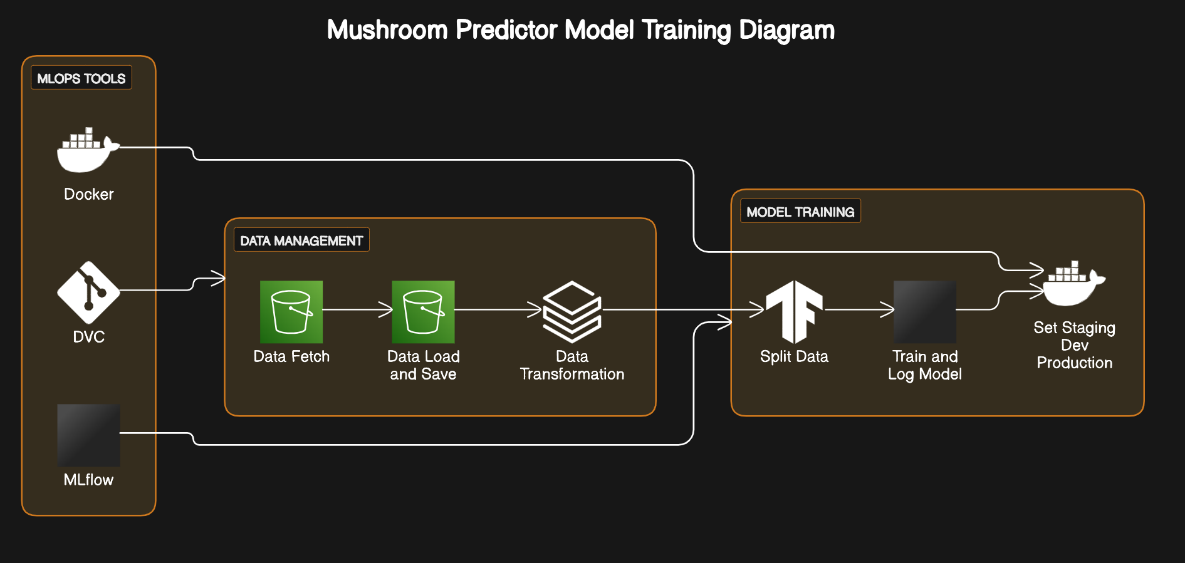
In our machine learning project, the Adaboost (Adaptive Boosting) algorithm is utilized as the primary model for mushroom classification. Adaboost is an ensemble learning technique that combines multiple weak learners, typically decision trees, to create a strong classifier. Let's delve into the details of the Adaboost model architecture and the parameters used:

* **Overview of Adaboost**:

Adaboost works by sequentially training a series of weak learners on different subsets of the training data. Each weak learner focuses on the instances that were misclassified by the previous ones, effectively reducing bias and improving overall classification accuracy.

The final prediction is made by combining the predictions of all weak learners, weighted by their individual performance during training.

* **Parameters Used**:
* ***n\_estimators***: This parameter specifies the number of weak learners (decision trees) to be used in the ensemble. In our project, we set n\_estimators to 100, indicating that 100 decision trees will be sequentially trained and combined to form the final classifier.
* ***learning\_rate***: The learning rate parameter controls the contribution of each weak learner to the final ensemble. A lower learning rate typically results in a more conservative updating of weights, while a higher learning rate allows for faster convergence but may lead to overfitting. We chose a learning rate of 0.1, which strikes a balance between convergence speed and model generalization.
* ***algorithm***: The algorithm parameter specifies the boosting algorithm to be used by Adaboost. The options available include 'SAMME' (Stagewise Additive Modeling using a Multiclass Exponential loss function) and 'SAMME.R' (Real SAMME). In our project, we selected 'SAMME', which is suitable for discrete class labels.

Model Training Pipeline:-

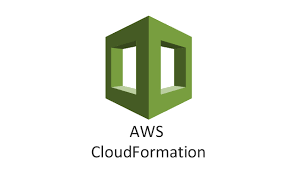
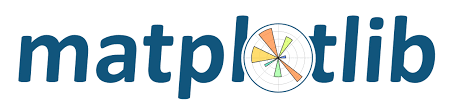
* **Data Loading**: Retrieve mushroom dataset from local file, ensuring completeness and integrity, readying it for preprocessing.
* **Data Preprocessing**: Clean, transform, and standardize mushroom data, handling missing values and outliers for improved model performance.
* **Data Splitting**: Partition preprocessed data into training and testing subsets to assess model generalization and prevent overfitting.
* **Model Training**: Employ training data to build machine learning models, optimizing parameters to capture underlying patterns in mushroom characteristics.
* **Model Evaluation**: Assess trained models' performance using testing data, employing metrics like accuracy and precision to gauge effectiveness and identify potential improvements.
* **Model Logging**: Record trained models, along with relevant metadata and performance metrics, facilitating model tracking, reproducibility, and future iterations.

1. **Tools used :-**

Python programming language and frameworks such as NumPy, Pandas, Scikit-learn,are used to build the whole model.









|  |  |
| --- | --- |
| **Front End** | HTML/CSS/JS |
| **Backend** | Python Flask |
| **Database** | Data Stax Astra (Cassandra) |
| **Deployment** | AWS Docker-compose |
| **MLOps Tools** | Docker DVC Mlflow |
| **Source Control Management** | Git/Github |

* Python: Primary programming language for developing the project due to its extensive libraries for machine learning and web development.
* scikit-learn: Machine learning library for building and training classification models.
* Flask: Web development frameworks for creating the user interface and backend services.
* Datastax Astra: Database management systems for storing mushroom attribute data.
* DVC (Data Version Control): Tool for managing data versioning and facilitating reproducibility in machine learning experiments.
* MLflow: MLOps platform for tracking experiments, managing models, and deploying machine learning models.
* Docker: Containerization platform for packaging the application and its dependencies into portable containers.
* AWS or Google Cloud Platform: Cloud computing services for scalable infrastructure, storage, and deployment.
* Git: Version control system for managing codebase and collaborating with team members.
* Jupyter Notebook: Interactive environment for data exploration, experimentation, and model prototyping.
* Pandas: Data manipulation library for preprocessing and analyzing the mushroom attribute data.
* NumPy: Numerical computing library for handling numerical operations and array processing.
* Matplotlib or Seaborn: Visualization libraries for creating visualizations and plots to understand the data and model performance.
* AWS CloudFormation or Terraform: Infrastructure as code tools for automating the provisioning and management of cloud resources.
* GitHub: Collaboration platforms for hosting code repositories, managing issues, and facilitating code reviews.

**4 Infrastructure Architecture :-**

Our infrastructure architecture is designed to support the machine learning system for mushroom classification, ensuring scalability and efficient utilization of resources. Here's an overview of the hardware and software infrastructure required:

1. **Cloud Computing Resources**: We leverage cloud computing services, such as Amazon Web Services (AWS) or Google Cloud Platform (GCP), to host our machine learning infrastructure. These platforms provide on-demand access to scalable compute resources, allowing us to dynamically allocate computing power based on workload demands.
2. **Virtual Machines (VMs) or Containerized Environments**: We deploy our machine learning models and associated workflows within virtual machines or containerized environments, such as Docker containers orchestrated with Kubernetes. These containers encapsulate all dependencies and configurations required for model training and inference, ensuring portability and consistency across environments.
3. **Scalability Considerations**: Our infrastructure architecture is designed to handle varying loads of data and user requests, ensuring scalability to accommodate fluctuations in demand. We employ auto-scaling mechanisms provided by cloud platforms to automatically adjust compute resources based on workload metrics, such as CPU utilization or incoming requests.
4. **Storage Solutions**: We utilize cloud-based storage solutions, such as Amazon S3 or Google Cloud Storage, to store datasets, model artifacts, and other relevant files. These storage services offer scalability, durability, and low-latency access to data, facilitating efficient data processing and model training.
5. **Networking Infrastructure**: We configure networking infrastructure to facilitate communication between different components of the system, including data ingestion, model training, and inference. We ensure secure communication protocols, such as HTTPS, to protect data in transit and enforce access control policies to restrict unauthorized access.
6. **Monitoring and Logging**: We implement monitoring and logging mechanisms to track the performance and health of our infrastructure components. We use tools like AWS CloudWatch or Google Cloud Monitoring to monitor resource utilization, detect anomalies, and troubleshoot issues in real-time.
7. **Security Measures**: We enforce robust security measures to protect our infrastructure and data assets. This includes implementing encryption mechanisms to secure data at rest and in transit, implementing access control policies to restrict unauthorized access, and regularly updating and patching software components to address security vulnerabilities.

**5. Deployment Architecture :-**

