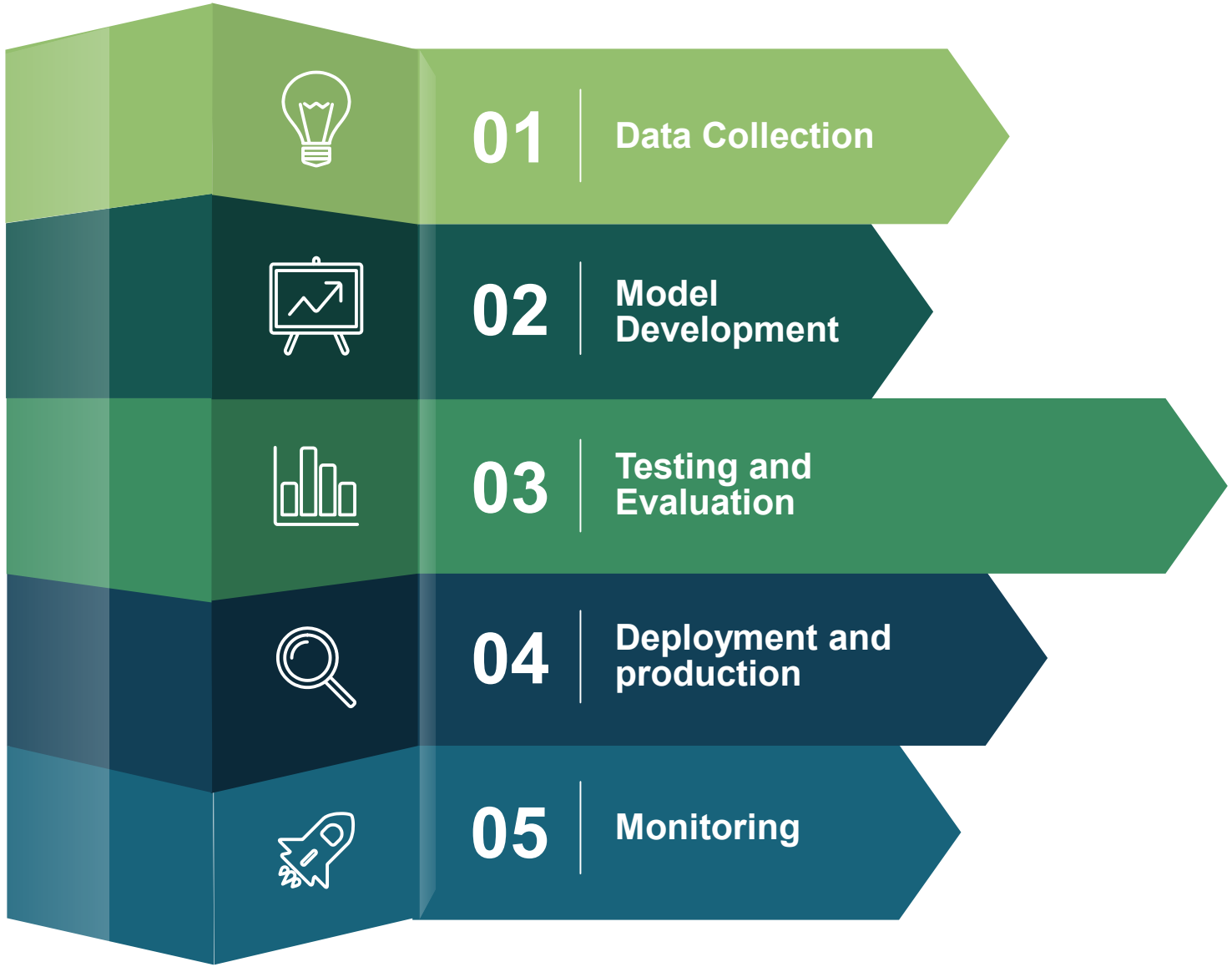
DEPLOYMENT PLAN

Model deployment is a term used to refer to the phase of machine learning wherein the model is used in real-time, collecting and processing incoming data, to meet the business requirement. Developing a deployment strategy for loan default prediction models involves careful planning and consideration of various factors, including model performance monitoring and updating protocols.



1

Model Development

ML Model Development

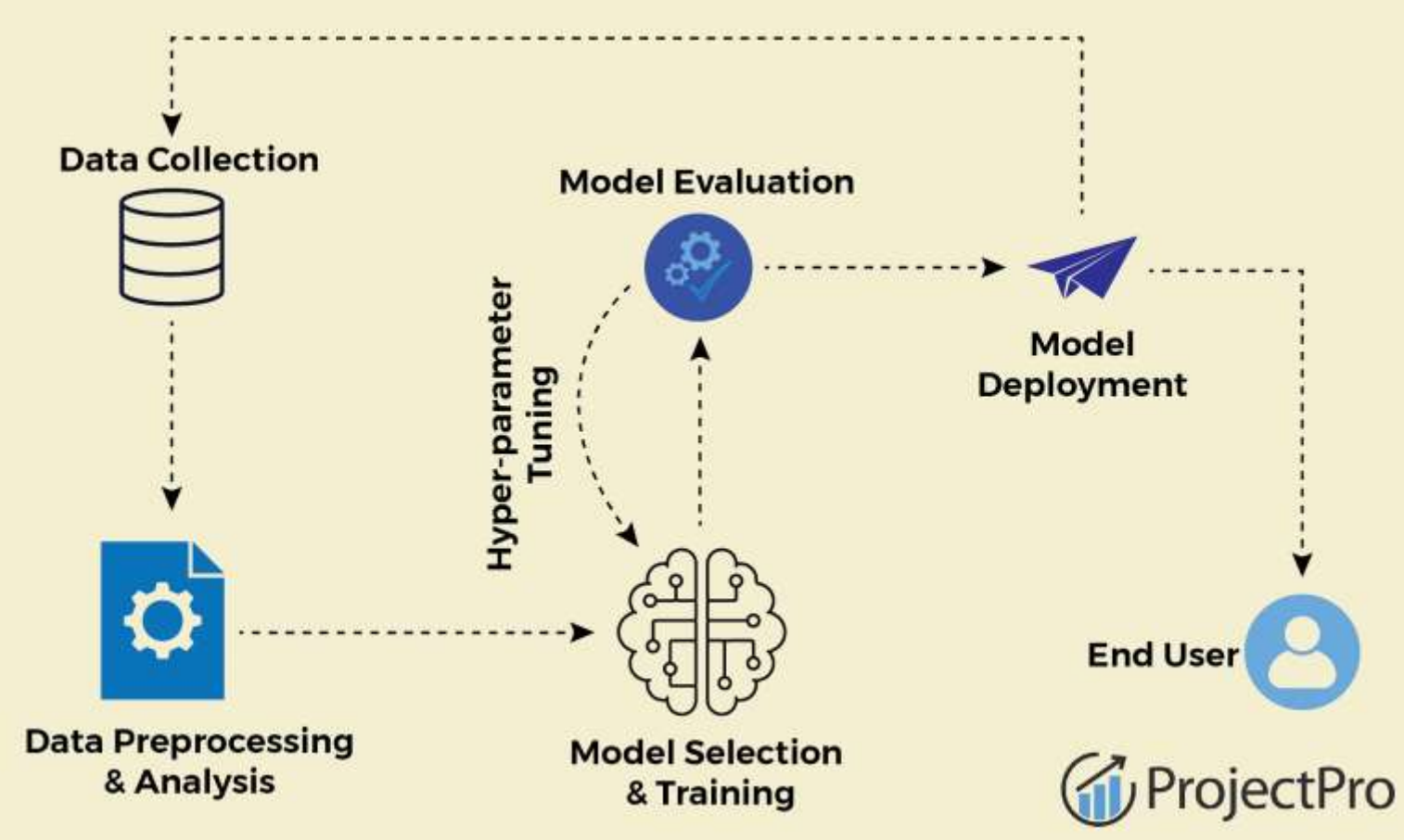
Technology: Scikit-learn for Naïve Bayes, Logistic Regression, Random Forest, and XGBoost framework for building machine learning models or Spark's MLlib library

•**Setup:** Choose appropriate algorithms (e.g., logistic regression, random forest, gradient boosting) based on the dataset and problem. Use cross-validation for model selection and hyperparameter tuning.

•**Development Environemnt:** Offline - Jupyter Notebook, Google Colab, Jenkins

2. Input

Personal Data



An earpiece with complete and clear identification information will often have higher reliability

2

Model Validation

Data Testing and Validation

Correlation Check

- This step involves training and validating the ML model using appropriate datasets.

- After that, you have to optimize and fine-tune the model for performance and accuracy.

- Finally, you save the trained model in a format compatible with the deployment environment.

3

Deployment and production

Deployment Environment

Select a deployment environment suitable for your specific requirements, such as cloud platforms (e.g., AWS, Azure, Google Cloud) or on-premises infrastructure.

*** Server hosting requires a high initial investment but offers more control, making it a preferred choice for handling sensitive data.
=> Apache Spark, Hadoop, Kubernetes

*** On the other hand, cloud hosting can be more cost-effective initially and provides scalability and flexibility.

Deployment Environment

*** On-premises server hosting Apache Spark, Hadoop, Kubernetes

1. Implemented a cluster using a managed distribution, such as Hortonworks or Cloudera => Add the Spark service using Ambari (Hwx) or Cloudera Manager
2. Set up cluster managed with the native Hadoop execs: Download and untar Spark, configure the config files - including YARN integration
   + Install V.M.(virtual machine)
   + Install Cent-OS on that V.M
   + Setup a multi node cluster: Hadoop Master – Hadoop Slavers
   + Installing Java(on master's machine)
   + Installing-Configuring Hadoop on Master's machine (core-site.xml, hdfs-site.xml, yarn-site.xml)
   + Install Spark on master and node

https://www.confessionsofadataguy.com/create-your-very-own-apache-spark-hadoop-cluster-then-do-something-with-it/

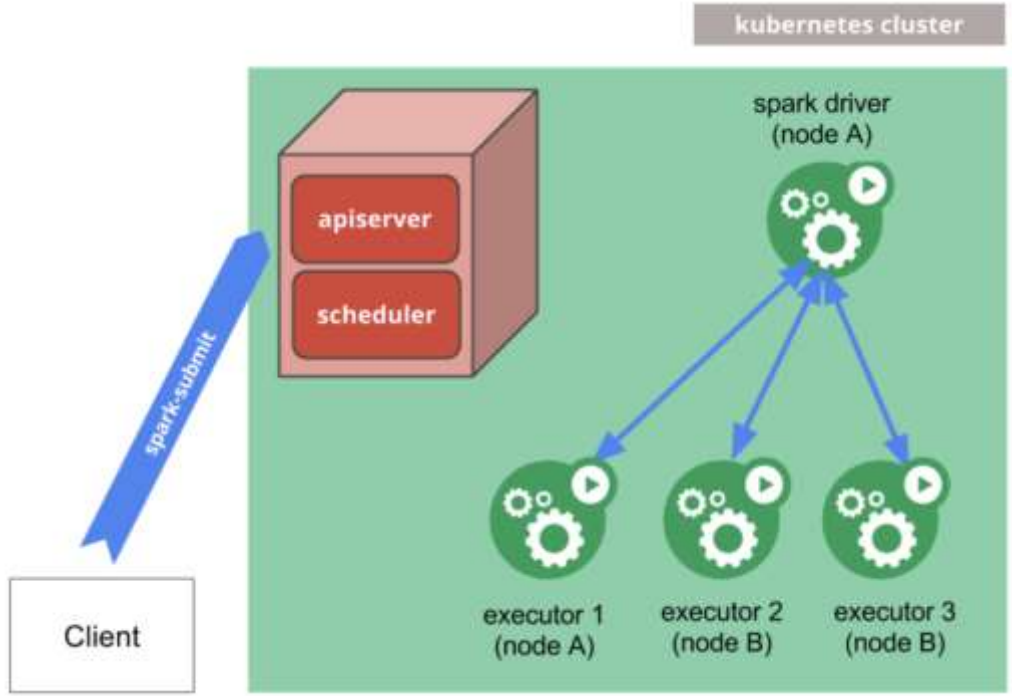
Deployment Environment

*** On-premises server hosting Apache Spark, Hadoop, Kubernetes

3. Set up Kubernetes cluster at version >= 1.24 with access configured to it using kubectl.

- Kubernetes DNS configured in your cluster.
- Creating a Deployment for a Spark Master.
- Create a service configuration as "spark-master-service.yml."
- Create a spark worker "spark-worker.yml" and deploy it with kubectl.

- spark-submit can be directly used to submit a Spark application to a Kubernetes cluster.



Benefits of Kubernetes

What makes Kubernetes such a popular choice for managing containerized applications? Building on more than a decade of running production workloads at Google, Kubernetes offers numerous potential benefits:

1. Scalability: Data ops teams can easily scale up to support a near-unlimited number of workloads.

2. Open Source: Kubernetes was originally developed by Google but is now maintained by Cloud Native Computing Foundation.

3. Flexibility: Whether you're in the cloud, multiple clouds, hybrid, or on-prem, Kubernetes provides flexibility to adapt to your infrastructure.

4. KS(Elastic Kubernetes Service), Azure Kubernetes Service (AKS) and Google Kubernetes Engine (GKE) are the popular k8s service platform provided by AWS, Azure and Google respectively which can be used for cloud deployment.

3. Deployment and production

Benefits of Spark on Kubernetes

Many companies are finding that Kubernetes offers better dependency management, resource management, and includes a rich ecosystem of integrations.

1. Dependency Management: Spark has traditionally struggled with dependency management. Running in containers on Kubernetes, however, allows them to package all of their dependencies in containers and easily take them from their laptop to production.

2. Resource Management: Containers also allow for better isolation of workloads, increasing the ability to bin pack workloads. Using YARN will require running multiple clusters can increase costs. Spark on Kubernetes, on the other hand, allows for different versions of Spark and Python to run in the same cluster and allows seamless resource sharing. As one job ends, another can be scheduled. It even allows cluster autoscaling to further optimize costs.

3. Ecosystem: There are many other powerful open source add-ons for management and monitoring, like Prometheus for time-series data and Fluentd for log aggregation

Containerize the model

Package the ML model and its dependencies into a container (e.g., Docker container). Containerization ensures that the model, its dependencies, and the runtime environment are encapsulated together, facilitating portability and reproducibility.

1. Serialize and save the trained model using libraries like joblib or pickle. This allows us to load the model later without retraining.

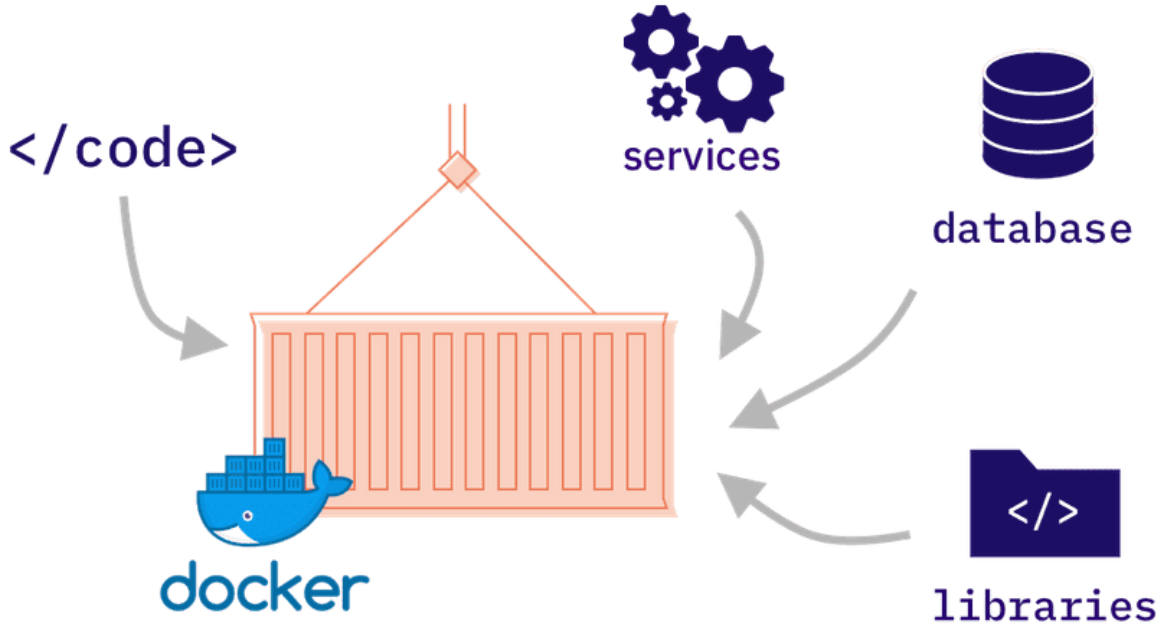
2. Created a virtual project environment in vs code using pipenv and created a Python file which uses *Flask* framework and model to create an API for the user to interact with the model and pass features of a candidate for prediction. The required libraries like scikit-learn, numpy, flask etc were installed in a virtual environment using pipenev.

3. Use a Dockerfile with a python base image was created for preparing the Docker image and running a container from it. Run the Python file on Kubernetes

4. Set up a cron job for scheduling jobs.

Deploy the containerized

Before deploying the containerized ML model, you have to set up the necessary infrastructure in the chosen environment. Once it's deployed, ensure its access to the required resources (e.g., compute power, storage) and then configure the networking and security settings appropriately..



4

Model Monitoring

3. Model Monitoring

Correlation Check

1. Monitor and scale
Install and set up Prometheus and Grafana for monitoring jobs and create real-time dashboards and charts.

2. Continuous integration and deployment (CI/CD)
Technology: GitLab CI/CD, Jenkins, or GitHub Actions for automating the deployment pipeline.
Use Git to track changes to the codebase, collaborate with team members, and manage different versions of the application.

3. Post-deployment maintenance
Confluence, or Google Docs for documenting the system architecture, setup instructions, and user guides.
Setup: Create comprehensive documentation covering the technology stack, setup procedures, and usage guidelines for team members and stakeholders.

References

- https://www.almabetter.com/bytes/tutorials/mlops/set-up-distributed-ml-environment-with-apache-spark
- https://www.acceldata.io/blog/why-move-from-spark-on-yarn-to-kubernetes
- https://www.techtarget.com/searchitoperations/tutorial/How-to-run-ML-workloads-with-Apache-Spark-on-Kubernetes
- https://kubernetes.io/docs/setup/
- https://spark.apache.org/docs/latest/running-on-kubernetes.html
- https://neptune.ai/blog/apache-spark-tutorial
- https://www.analyticsvidhya.com/blog/2022/01/deploying-ml-models-using-kubernetes/
- https://www.projectpro.io/article/machine-learning-model-deployment/872