CS 816 - Software Production Engineering
Final Project Report

Text2Scene
Crafting Visual Scenes from Textual Descriptions

under guidance of
Prof. B. Thangaraju

Report by,

Team – 13
(T13_MT001_MT005)

Sharvari Mishra
MT2023001
Sharvari.Mishra@iiitb.ac.in

Bhumika Jindal
MT2023005
Bhumika.Jindal@iiitb.ac.in

Submitted On
24th May, 2024

Contents

List of Figures

1. Problem Statement

Text-to-Image generation is an integrated task that involves concepts of Natural language processing, Machine Learning, and Computer vision to generate an image that corresponds to a given textual description. Making this multilingual will facilitate effective communication, marketing, education, and cultural exchange across linguistic and cultural boundaries.

In the dynamic and competitive landscape of creative industries, professionals such as advertising experts, movie directors, and designers face a constant challenge in creating unique and attention-grabbing content. The need for content that transcends language barriers and resonates with diverse audiences is paramount. Developing a user-friendly UI interface for a Multilingual Text-to-Image Generator is essential to empower creative professionals to seamlessly transform their imaginative concepts, expressed in any language, into visually striking images. This innovation not only enhances creative expression but also serves as a universal bridge, ensuring the global reach of compelling content.

To ensure the efficient development, deployment, and maintenance of the Multilingual Text-to-Image Generator, integrating this project with DevOps practices is crucial. This integration addresses several key challenges:

- **Continuous Integration/Continuous Deployment (CI/CD):** Ensuring stability through automated tests on code changes and deploying new features and bug fixes without significant downtime.
- **Collaboration and Version Control:** Facilitating better collaboration among developers, allowing simultaneous work on different parts of the project, and tracking code changes while restoring previous versions as needed.
- **Infrastructure as Code (IaC):** Easily scaling the application to handle more users and larger workloads, and maintaining consistent environments across development, testing, and production.

Integrating DevOps practices will ensure a robust, scalable, and maintainable system, allowing creative professionals to focus on content creation without the burden of underlying technical challenges. DevOps provides the necessary tools and practices for efficient code development, testing, and deployment, ensuring the seamless operation of the Multilingual Text-to-Image Generator.

2. Deliverables

- GitHub Repository - https://github.com/bhumika-16/text2scene/tree/main
- Docker Hub Repository - https://hub.docker.com/repositories/bhumika16

3. What and Why DevOps?

DevOps is a combination of software development (dev) and operations (ops). It is defined as a software engineering methodology which aims to integrate the work of development teams and operations teams by facilitating a culture of collaboration and shared responsibility.

DevOps is a software development methodology that emphasizes collaboration between development and operations teams, aiming for streamlined and automated workflows. It seeks to improve communication, efficiency, and deployment speed, ultimately enhancing the software delivery process. DevOps integrates continuous integration, continuous delivery, and automation to achieve rapid and reliable software development and deployment cycles.

i. Introduction

DevOps combines development (Dev) and operations (Ops) to unite people, process, and technology in application planning, development, delivery, and operations. DevOps enables coordination and collaboration between formerly siloed roles like development, IT operations, quality engineering, and security.

Teams adopt DevOps culture, practices, and tools to increase confidence in the applications they build, respond better to customer needs, and achieve business goals faster. DevOps helps teams continually provide value to customers by producing better, more reliable products.

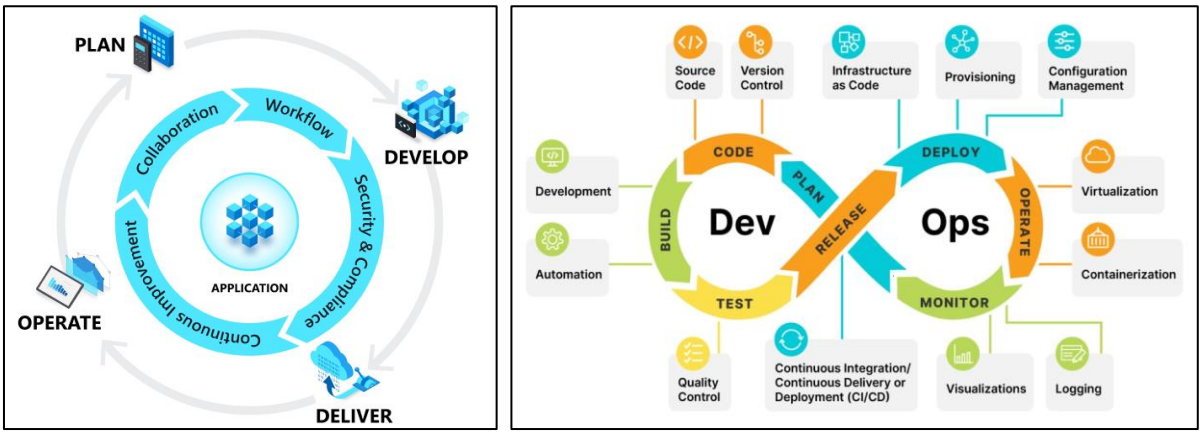


Figure 1: DevOps Life Cycle Phases

DevOps promotes collaboration between development and operations teams, including IT operations, quality engineering, and security. It emphasizes cross-team communication, team empowerment, and technology automation. DevOps enables teams to: Respond better to customer needs, increase confidence in their applications, achieve business goals faster, deliver applications and services at a faster pace, and compete more effectively in the market.

DevOps teams automate delivery processes to make them scalable, repeatable, controlled, and well-tested. They also define a release management process with clear manual approval stages, and set automated gates to move applications between stages until final release.

ii. Core Principles of DevOps

The DevOps methodology comprises four key principles that guide the effectiveness and efficiency of application development and deployment. These principles, listed below, centre on the best aspects of modern software development.

- **Automation of the software development lifecycle:** This includes automating testing, builds, releases, the provisioning of development environments, and other manual tasks that can slow down or introduce human error into the software delivery process.
- **Collaboration and communication:** A good DevOps team has automation, but a great DevOps team also has effective collaboration and communication.
- **Continuous improvement and minimization of waste:** From automating repetitive tasks to watching performance metrics for ways to reduce release times or mean-time-to-recovery, high performing DevOps teams are regularly looking for areas that could be improved.
- **Hyperfocus on user needs with short feedback loops:** Through automation, improved communication and collaboration, and continuous improvement, DevOps teams can take a moment and focus on what real users really want, and how to give it to them.

By adopting these principles, organizations can improve code quality, achieve a faster time to market, and engage in better application planning.

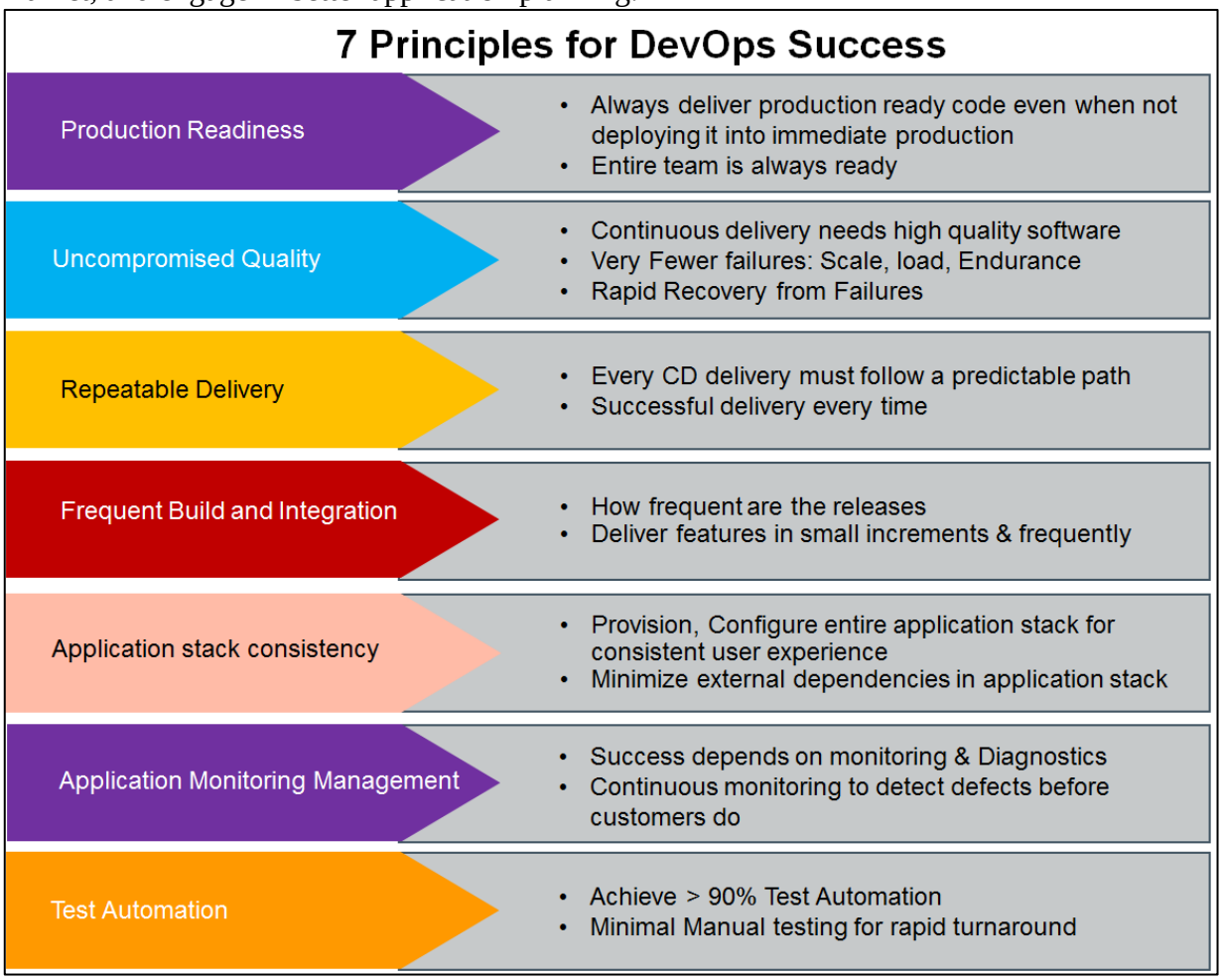


Figure 2: 7 Principles for DevOps Success

iii. Goal of DevOps

DevOps represents a change in mindset for IT culture. In building on top of Agile practices, DevOps focuses on incremental development and rapid delivery of software. Success relies on the ability to create a culture of accountability, improved collaboration, empathy, and joint responsibility for business outcomes. Adopting a DevOps strategy enables businesses to increase operational efficiencies, deliver better products faster, and reduce security and compliance risk. DevOps is a set of practices and principles that aim to streamline the software development process. The goals of DevOps include:

- **Continuous integration and delivery (CI/CD):** Most DevOps teams aim to achieve CI/CD by using tools to improve communication and throughput.
- **Agility:** DevOps uses continuous delivery to speed up product releases. Frequent releases of new features and faster identification and fixing of bugs at early stages ensure the quality of product releases.
- **Automation:** Automation enables organizations to automate repetitive, manual tasks and processes involved in software development and IT operations. The primary goal of automation in DevOps is to increase efficiency, reduce errors, and speed up the delivery of software.
- **Feedback:** DevOps automates repetitive tasks like reporting and testing to speed up the process and garner prompt feedback. This rapid feedback gives the development team a clear picture of changes, and then teams can deploy updated versions at lightning speed.
- **Software testing:** DevOps testing is highly automated, and takes into account the skills, goals, and feedback cycles of development, operations, and QA teams.
- **Better collaboration between teams:** Increased collaboration leads to a better understanding of each other's roles and responsibilities and helps streamline the process.
- **Containerization:** Containerization enables the rapid deployment of applications. Containers are also becoming more popular in production environments because they offer increased security and scalability.

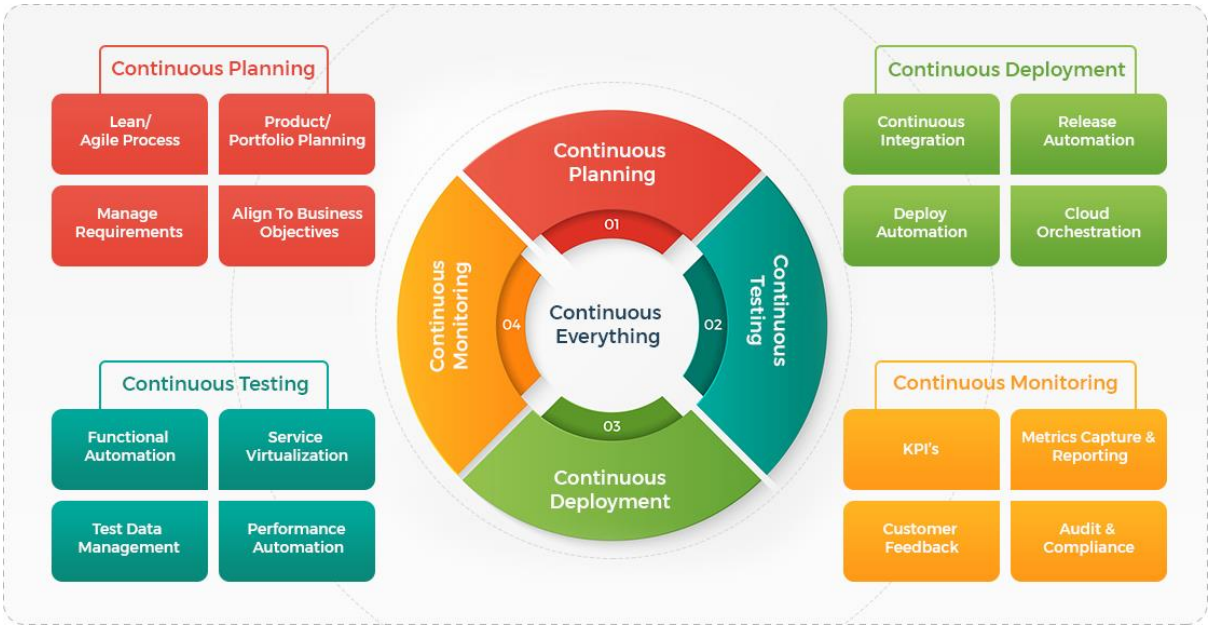
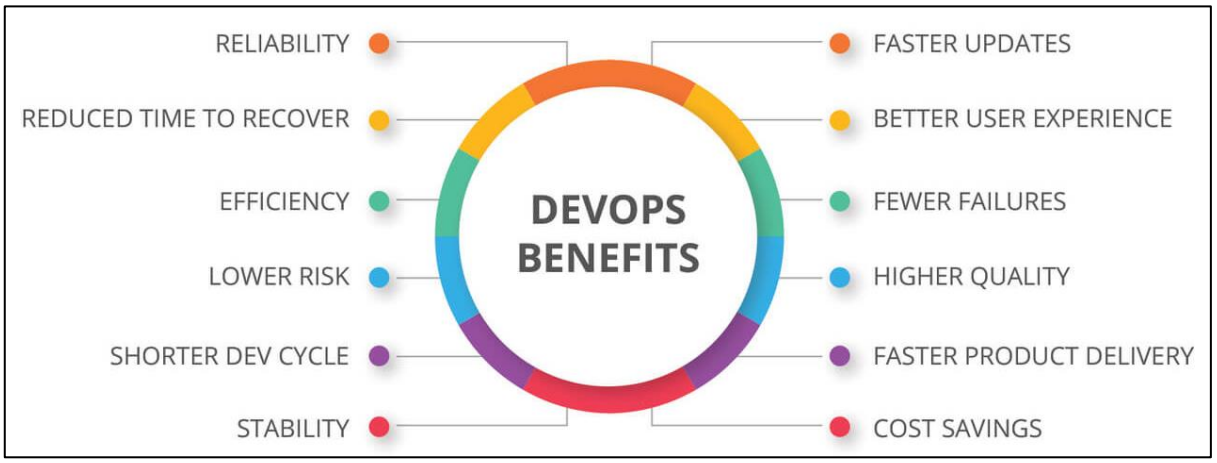


Figure 3: Goal of DevOps

iv. Benefits

Adopting DevOps breaks down barriers so that development and operations teams are no longer siloed and have a more efficient way to work across the entire development and application lifecycle. Without DevOps, organizations often experience handoff friction, which delays the delivery of software releases and negatively impacts business results.

- **Collaboration**: Adopting a DevOps model creates alignment between development and operations teams; handoff friction is reduced and everyone is all in on the same goals and objectives.
- **Fluid responsiveness**: More collaboration leads to real-time feedback and greater efficiency; changes and improvements can be implemented quicker and guesswork is removed.
- **Shorter cycle time:** Improved efficiency and frequent communication between teams shortens cycle time; new code can be released more rapidly while maintaining quality and security.



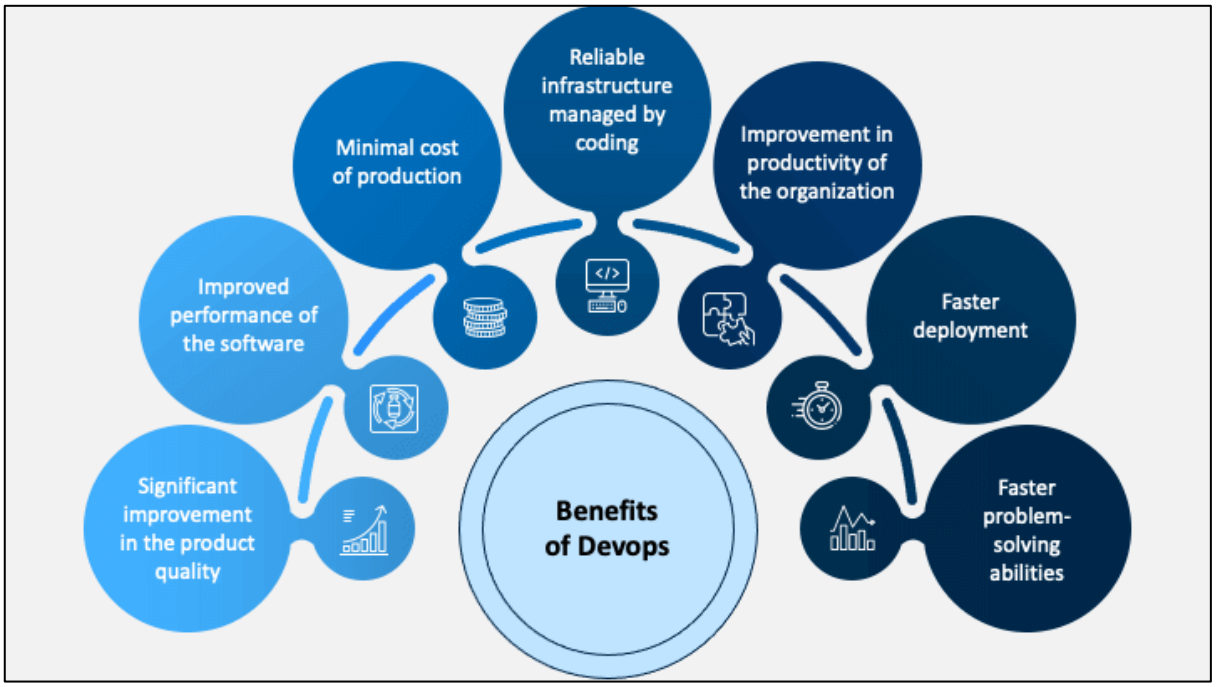


Figure 4: Benefits of DevOps

v. How to Implement DevOps Practices?

Implementing DevOps practices involves a combination of cultural, organizational, and technical changes. Here's a brief overview:

- **Cultural Shift:** Foster a collaborative culture where development and operations teams work together seamlessly, breaking down silos and emphasizing shared goals and responsibilities.
- **Automation:** Implement automation tools for continuous integration (CI) and continuous delivery (CD) to streamline and accelerate the software development lifecycle, reducing manual errors and enhancing efficiency.
- **Continuous Testing:** Integrate automated testing throughout the development process to ensure code quality and catch issues early, facilitating faster and more reliable releases.
- **Infrastructure as Code (IaC):** Treat infrastructure configuration as code, allowing for automated provisioning and management of infrastructure, ensuring consistency and scalability.
- **Monitoring and Feedback:** Implement robust monitoring tools to gather performance data and provide feedback loops, enabling quick detection and resolution of issues, and facilitating continuous improvement.
- **Collaboration Tools:** Utilize collaboration platforms and tools that enable effective communication and knowledge sharing among team members, fostering collaboration and transparency.
- **Security Integration:** Integrate security practices into the development process from the outset, ensuring that security is a fundamental consideration at every stage of the software delivery pipeline.
- **Containerization and Orchestration:** Adopt containerization technologies (e.g., Docker) and container orchestration tools (e.g., Kubernetes) to enhance portability, scalability, and deployment consistency.
- **Continuous Learning:** Encourage a culture of continuous learning and improvement by regularly reviewing processes, gathering feedback, and adapting practices to meet evolving needs and challenges.
- **Executive Support:** Obtain leadership buy-in and support to drive the organizational changes needed for successful DevOps implementation, aligning business goals with the principles of agility and efficiency.

vi. Overview of DevOps Practices

Continuous integration and continuous delivery (CI/CD):

Continuous Integration (CI) is the practice used by development teams to automate, merge, and test code. CI helps to catch bugs early in the development cycle, which makes them less expensive to fix. Automated tests execute as part of the CI process to ensure quality. CI systems produce artifacts and feed them to release processes to drive frequent deployments.

Continuous Delivery (CD) is a process by which code is built, tested, and deployed to one or more test and production environments. Deploying and testing in multiple environments increases quality. CD systems produce deployable artifacts, including infrastructure and apps. Automated release processes consume these artifacts to release new versions and fixes to existing systems. Systems that monitor and send alerts run continually to drive visibility into the entire CD process.

Version Control:

Version control is the practice of managing code in versions—tracking revisions and change history to make code easy to review and recover. This practice is usually implemented using version control systems such as Git, which allow multiple developers to collaborate in authoring code. These systems provide a clear process to merge code changes that happen in the same files, handle conflicts, and roll back changes to earlier states.

The use of version control is a fundamental DevOps practice, helping development teams work together, divide coding tasks between team members, and store all code for easy recovery if needed. Version control is also a necessary element in other practices such as continuous integration and infrastructure as code.

Agile software development:

Agile is a software development approach that emphasizes team collaboration, customer and user feedback, and high adaptability to change through short release cycles. Teams that practice Agile provide continual changes and improvements to customers, collect their feedback, then learn and adjust based on customer wants and needs. Agile is substantially different from other more traditional frameworks such as waterfall, which includes long release cycles defined by sequential phases. Kanban and Scrum are two popular frameworks associated with Agile.

Infrastructure as code:

Infrastructure as code defines system resources and topologies in a descriptive manner that allows teams to manage those resources as they would code. Those definitions can also be stored and versioned in version control systems, where they can be reviewed and reverted—again like code.

Practicing infrastructure as code helps teams deploy system resources in a reliable, repeatable, and controlled way. Infrastructure as code also helps automate deployment and reduces the risk of human error, especially for complex large environments. This repeatable, reliable solution for environment deployment lets teams maintain development and testing environments that are identical to production. Duplicating environments to different data centres and cloud platforms likewise becomes simpler and more efficient.

Configuration management:

Configuration management refers to managing the state of resources in a system including servers, virtual machines, and databases. Using configuration management tools, teams can roll out changes in a controlled, systematic way, reducing the risks of modifying system configuration. Teams use configuration management tools to track system state and help avoid configuration drift, which is how a system resource’s configuration deviates over time from the desired state defined for it. Along with infrastructure as code, it's easy to templatize and automate system definition and configuration, which help teams operate complex environments at scale.

Continuous monitoring:

Continuous monitoring means having full, real-time visibility into the performance and health of the entire application stack. This visibility ranges from the underlying infrastructure running the application to higher-level software components. Visibility is accomplished through the collection of telemetry and metadata and setting of alerts for predefined conditions that warrant attention from an operator. Telemetry comprises event data and logs collected from various parts of the system, which are stored where they can be analysed and queried.

High-performing DevOps teams ensure they set actionable, meaningful alerts and collect rich telemetry so they can draw insights from vast amounts of data. These insights help the team mitigate issues in real time and see how to improve the application in future development cycles.

Planning:

In the planning phase, DevOps teams ideate, define, and describe the features and capabilities of the applications and systems they plan to build. Teams track task progress at low and high levels of granularity, from single products to multiple product portfolios.

Development:

The development phase includes all aspects of developing software code. In this phase, DevOps teams do the following tasks:

- Select a development environment.
- Write, test, review, and integrate the code.
- Build the code into artifacts to deploy into various environments.
- Use version control, usually Git, to collaborate on code and work in parallel.

To innovate rapidly without sacrificing quality, stability, and productivity, DevOps teams:

- Use highly productive tools.
- Automate mundane and manual steps.
- Iterate in small increments through automated testing and continuous integration (CI).

Deliver:

Delivery is the process of consistently and reliably deploying applications into production environments, ideally via continuous delivery (CD). In the delivery phase, DevOps teams:

- Define a release management process with clear manual approval stages.
- Set automated gates to move applications between stages until final release to customers.
- Automate delivery processes to make them scalable, repeatable, controlled, and well-tested.

Delivery also includes deploying and configuring the delivery environment's foundational infrastructure. DevOps teams use technologies like infrastructure as code (IaC), containers, and microservices to deliver fully governed infrastructure environments. Safe deployment practices can identify issues before they affect the customer experience. These practices help DevOps teams deliver frequently with ease, confidence, and peace of mind.

Operations:

The operations phase involves maintaining, monitoring, and troubleshooting applications in production environments, including hybrid or public clouds like Azure. DevOps teams aim for system reliability, high availability, strong security, and zero downtime.

Automated delivery and safe deployment practices help teams identify and mitigate issues quickly when they occur. Maintaining vigilance requires rich telemetry, actionable alerting, and full visibility into applications and underlying systems.

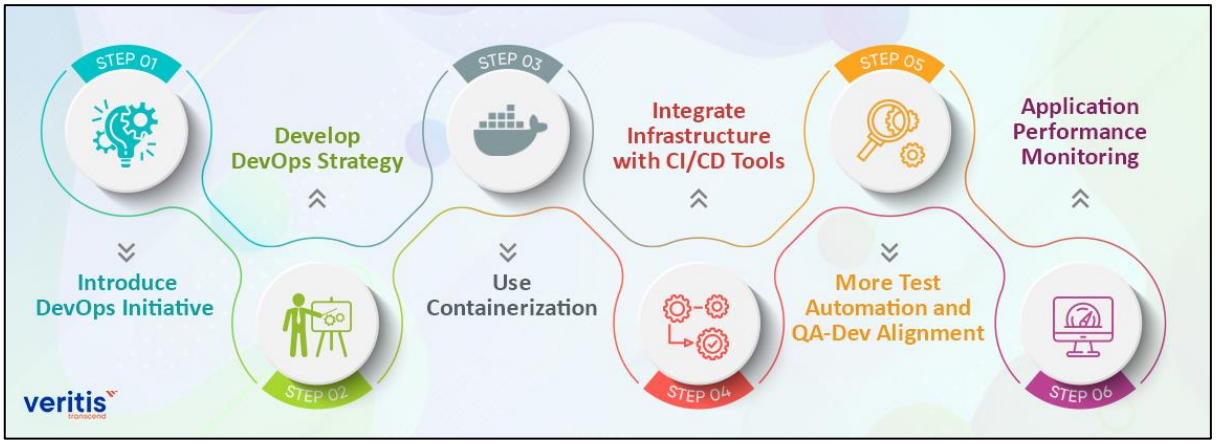


Figure 5: DevOps Implementation Practices

vii. Popular DevOps Tools

DevOps relies on a variety of tools to facilitate collaboration, automation, and efficiency across the software development and IT operations lifecycle. Here's an overview of some commonly used DevOps tools, categorized based on their functions:

1. Version Control:

Git: A distributed version control system that allows multiple developers to collaborate on projects, track changes, and manage code repositories efficiently.

2. Continuous Integration (CI) Tools:

Jenkins: An open-source automation server that supports building, testing, and deploying code continuously. Jenkins integrates with various plugins and supports a wide range of languages and tools.

Travis CI: A cloud-based CI service that automates the testing and deployment process for GitHub repositories. It's especially popular for open-source projects.

CircleCI: A CI/CD platform that automates software development processes, providing support for building, testing, and deploying applications.

3. Continuous Deployment/Delivery (CD) Tools:

Ansible: An open-source automation tool for configuration management, application deployment, and task automation. It uses simple, human-readable YAML scripts.

Docker: A platform for containerization, allowing developers to package applications and their dependencies into containers for consistent deployment across various environments.

Kubernetes: An open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications.

Spinnaker: An open-source, multi-cloud continuous delivery platform that supports deploying and managing applications across different cloud providers.

4. Infrastructure as Code (IaC) Tools:

Terraform: An open-source IaC tool that allows users to define and provision infrastructure using a declarative configuration language.

AWS CloudFormation: An Amazon Web Services (AWS) service for defining and deploying infrastructure as code on the AWS platform.

5. Configuration Management Tools:

Chef: An automation platform that manages infrastructure as code. It uses recipes and cookbooks to define configurations and automate infrastructure tasks.

Puppet: A configuration management tool that automates the provisioning and management of infrastructure. It uses a declarative language for defining system configurations.

Ansible (again): In addition to its role in CI, Ansible is also used for configuration management.

6. Continuous Monitoring Tools:

Prometheus: An open-source monitoring and alerting toolkit designed for reliability and scalability. It collects metrics from configured targets and stores them.

Grafana: A popular open-source analytics and monitoring platform that integrates with various data sources, including Prometheus, to visualize and analyse metrics.

7. Collaboration and Communication Tools:

Slack: A messaging platform that facilitates team communication, collaboration, and the integration of various tools and services.

Microsoft Teams: A collaboration platform that provides chat, video conferencing, and integration with other Microsoft 365 applications.

These are just a few examples, and the DevOps tool landscape is extensive, with new tools emerging regularly. The choice of tools depends on factors such as team preferences, project requirements, and existing technology stacks.

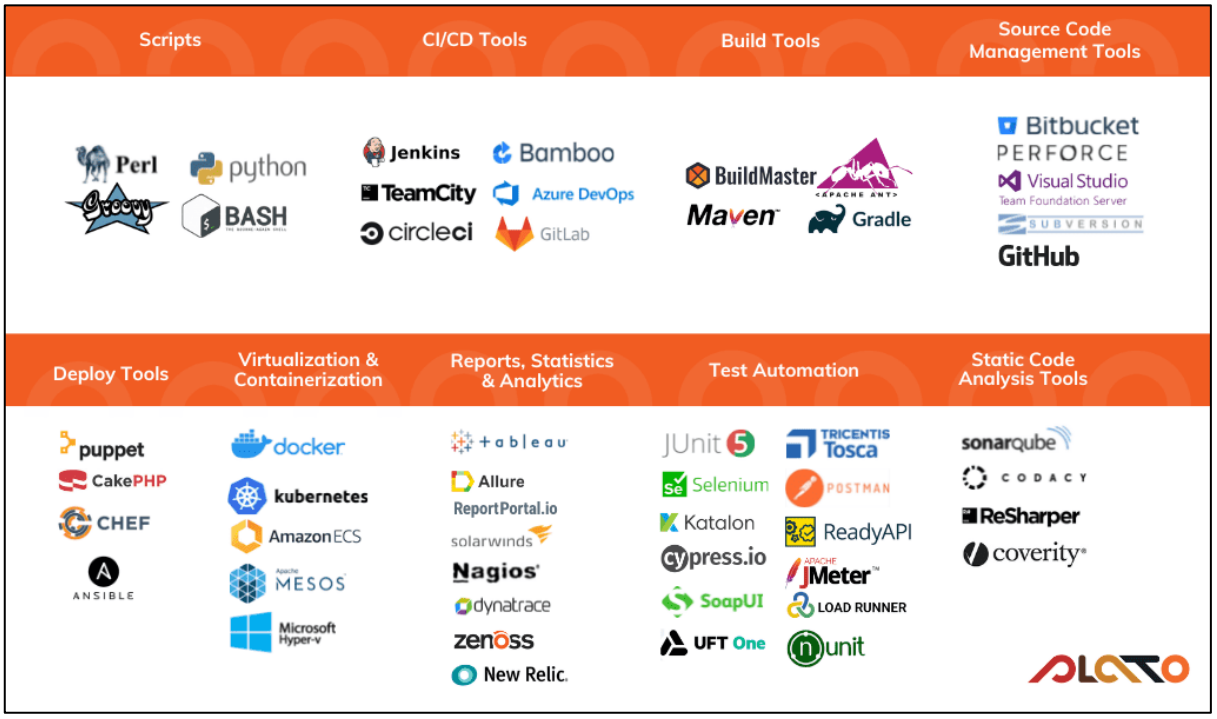


Figure 6: DevOps Tools Ecosystem

viii. Importance of DevOps in our Project

Integrating DevOps into the Multilingual Text-to-Image Generator project is crucial for several reasons:

- **Seamless Workflow:** Enhances application stability and reliability through continuous integration and continuous deployment (CI/CD).
- **Automated Testing:** Catches issues early, ensuring code changes do not introduce new bugs.
- **Rapid Deployment:** Allows for quick deployment of new features and bug fixes, minimizing downtime and improving user experience.
- **Enhanced Collaboration:** Facilitates better collaboration among developers, enabling efficient work on different project parts simultaneously.
- **Version Control:** Tracks all code changes, making it easy to revert to previous versions if necessary.
- **Infrastructure as Code (IaC):** Ensures easy scalability to handle increased workloads and maintains consistent environments across development, testing, and production.
- **Consistency:** Reduces risk of discrepancies and errors from different environment configurations.
- **Focus on Content Creation:** Allows creative professionals to focus on content creation without technical challenges.

Overall, integrating DevOps practices ensures a robust, scalable, and maintainable system.

4. Docker

Docker is an open-source platform designed to automate the deployment, scaling, and management of applications. It does this through the use of containerization, which allows developers to package applications and their dependencies into lightweight, portable containers. These containers can run consistently across various computing environments, making Docker an essential tool for modern software development and deployment.

Why and When to Use Docker?

Docker is particularly useful in scenarios where consistent and reliable deployment environments are crucial. This includes:

- **Development**: Docker enables developers to create standardized development environments, ensuring that the application runs the same way on every developer's machine.
- **Testing**: Automated tests can be run in isolated environments, ensuring that tests are not affected by external factors.
- **Continuous Integration/Continuous Deployment (CI/CD)**: Docker containers can be integrated into CI/CD pipelines, making the build, test, and deployment processes more efficient and consistent.
- **Microservices Architecture**: Docker is ideal for deploying microservices, as each service can run in its own container with its own dependencies, without conflicting with other services.
- **Cross-Platform Deployment**: Docker containers can run on any system that supports Docker, making it easier to deploy applications across different platforms.

Benefits of Using Docker

- **Consistency Across Environments**: Docker ensures that software behaves the same way, regardless of where it is run. This eliminates the "works on my machine" problem, as the same Docker image can be used in development, testing, and production.
- **Isolation and Security**: Each Docker container is isolated from others, which means that applications run in their own environment. This isolation improves security, as vulnerabilities in one application do not affect others.
- **Portability**: Docker containers can run on any system with Docker installed, making it easy to move applications across different environments, such as from a developer's laptop to a cloud server.
- **Efficiency and Speed**: Docker containers are lightweight and use fewer resources than traditional virtual machines. This efficiency leads to faster start-up times and better resource utilization.
- **Simplified Configuration**: Docker allows for easy configuration of environments. Developers can specify all dependencies and configurations in a Dockerfile, which can be version-controlled and shared.
- **Scalability**: Docker integrates well with orchestration tools like Kubernetes, making it easier to manage and scale applications. Containers can be scaled up or down quickly based on demand.
- **Continuous Integration/Continuous Deployment (CI/CD) Integration**: Docker fits seamlessly into CI/CD workflows, allowing for automated testing and deployment. This integration improves the overall efficiency and reliability of the development pipeline.
- **Cost Savings**: By using containers, organizations can run multiple applications on the same server, leading to better resource utilization and cost savings.

Details About Docker Components

- **Docker Engine**: The core component of Docker, it is responsible for building and running Docker containers. It consists of:
- **Docker Daemon**: Runs in the background and manages Docker containers.
- **Docker CLI**: The command-line interface used to interact with the Docker Daemon.
- **Docker Images**: Read-only templates used to create containers. Images can be built from a Dockerfile, which contains instructions for how to set up the environment inside the container.
- **Docker Containers**: Lightweight, portable, and self-sufficient units that can run applications. They are created from Docker images.
- **Docker Hub**: A public registry where Docker images can be stored, shared, and accessed. Developers can push their own images to Docker Hub and pull images created by others.
- **Docker Compose**: A tool for defining and running multi-container Docker applications. It uses a YAML file to configure application services, making it easy to manage multi-container setups.
- **Docker Swarm**: A native clustering and orchestration tool for Docker. It allows for the deployment and management of a cluster of Docker nodes as a single virtual system.

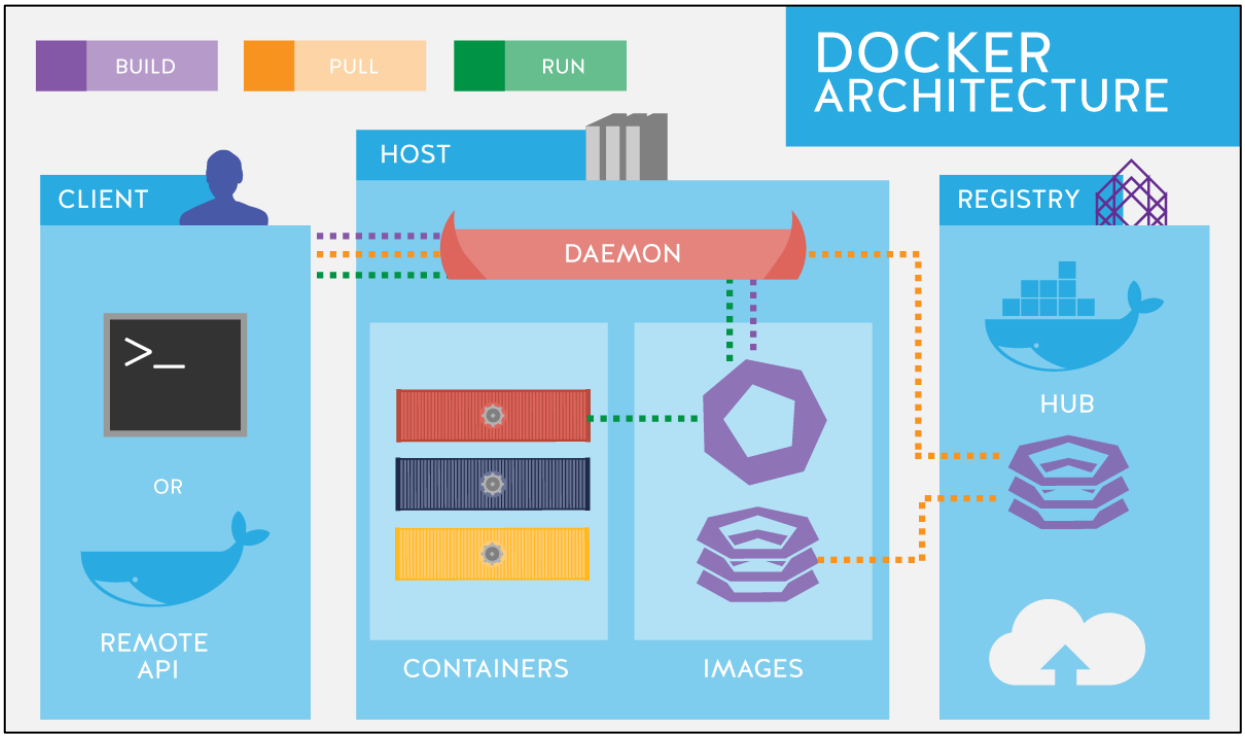


Figure 7: Docker Architecture

Docker revolutionizes the way applications are developed, tested, and deployed by providing a consistent environment across different stages of the development lifecycle. Its benefits, including portability, efficiency, and scalability, make it an indispensable tool for modern software development. Whether you are working on a small project or a large-scale microservices architecture, Docker can help you streamline your processes and improve productivity.

5. Kubernetes

Kubernetes, often abbreviated as K8s, is an open-source platform designed for automating the deployment, scaling, and operation of containerized applications. It was originally developed by Google and is now maintained by the Cloud Native Computing Foundation (CNCF). Kubernetes orchestrates containers, providing a framework to run distributed systems resiliently.

Why and When to Use Kubernetes?

Kubernetes is ideal for managing applications that require deployment, scaling, and operations of multiple containers. Specific scenarios include:

- **Microservices Architecture**: Kubernetes excels in managing applications built as a collection of small, loosely coupled services, each running in its own container.
- **Continuous Deployment and Continuous Integration (CI/CD)**: Kubernetes integrates well with CI/CD pipelines, automating deployment processes and ensuring that updates and rollbacks are handled smoothly.
- **Scaling Applications**: Kubernetes automatically scales applications up or down based on demand, ensuring optimal use of resources.
- **Multi-Cloud and Hybrid Deployments**: Kubernetes can be deployed across various cloud providers or on-premises, providing flexibility in managing hybrid and multi-cloud environments.
- **High Availability**: Kubernetes ensures application availability and self-healing by automatically restarting failed containers, redistributing load, and rescheduling when nodes die.

Benefits of Using Kubernetes

- **Automated Operations**: Kubernetes automates various operations like deployment, scaling, and maintenance of containerized applications, reducing manual intervention.
- **Scalability**: Kubernetes can automatically scale applications up and down based on real-time demand, ensuring efficient resource utilization.
- **Portability**: Kubernetes provides a consistent environment across different infrastructure environments, whether on-premises, in the cloud, or hybrid, enhancing application portability.
- **Resource Efficiency**: By managing resources effectively, Kubernetes ensures that applications are using the optimal amount of computing power, memory, and storage.
- **Self-Healing**: Kubernetes can detect and replace failed containers, ensuring that the desired state of the application is always maintained.
- **Service Discovery and Load Balancing**: Kubernetes provides built-in mechanisms for service discovery and load balancing, simplifying the architecture of distributed applications.
- **Secret and Configuration Management**: Kubernetes manages secrets and application configuration separately from code, enhancing security and flexibility.
- **Rolling Updates and Rollbacks**: Kubernetes allows for zero-downtime deployments through rolling updates and easy rollbacks, ensuring continuous delivery without impacting the user experience.

Details About Kubernetes Components

- **Kubernetes Master**: The master node is responsible for managing the Kubernetes cluster. Key components include:
  - **API Server**: Exposes the Kubernetes API, serving as the main entry point for all administrative tasks.
  - **Scheduler**: Assigns workloads to nodes based on resource availability and other constraints.
  - **Controller Manager**: Runs various controllers that ensure the desired state of the cluster.
  - **etcd**: A consistent and highly-available key-value store used for configuration data, state, and metadata of the cluster.
- **Nodes**: These are worker machines (virtual or physical) where containers are deployed. Key components include:
  - **Kubelet:** An agent that ensures containers are running in a Pod.
  - **Kube-proxy:** Maintains network rules and handles communication within and outside of the cluster.
  - **Container Runtime:** The software responsible for running containers, such as Docker, containerd, or CRI-O.
- **Pods**: The smallest deployable units in Kubernetes, a Pod is a group of one or more containers with shared storage/network and a specification for how to run the containers.
- **Services**: An abstraction that defines a logical set of Pods and a policy for accessing them, providing load balancing and service discovery.
- **Volumes**: Abstractions for data storage, allowing data to persist across container restarts.
- **Namespaces**: Virtual clusters within a Kubernetes cluster, used to organize and manage resources.
- **ConfigMaps and Secrets**: Tools for decoupling configuration artifacts and sensitive information from container images to keep the images portable.
- **Ingress**: Manages external access to services within a cluster, typically HTTP.

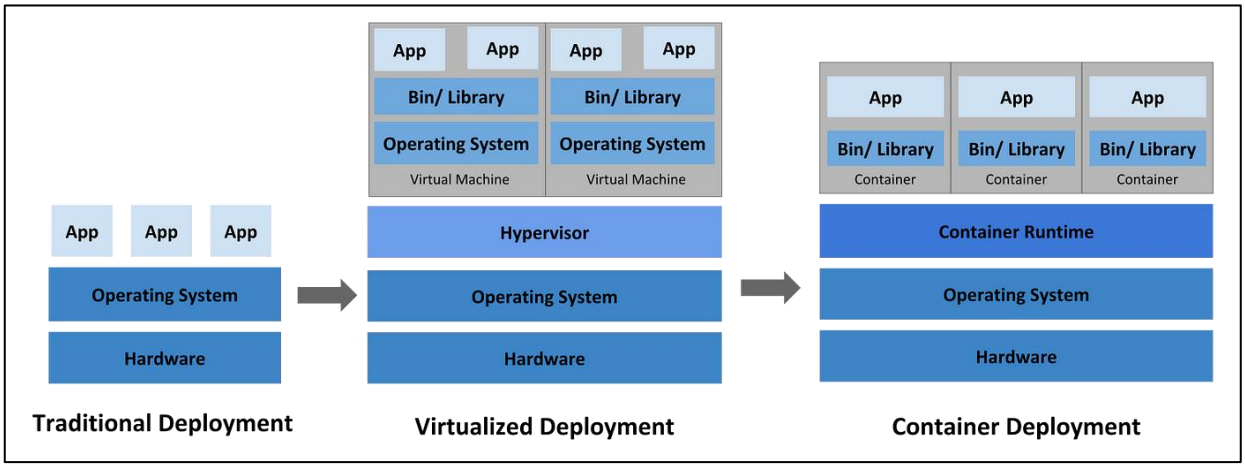


Figure 8: History of Deployments

Kubernetes provides a powerful, scalable, and reliable framework for managing containerized applications. It offers significant benefits, including automated operations, scalability, portability, and high availability. Whether you are managing a microservices architecture, setting up CI/CD pipelines, or operating in a multi-cloud environment, Kubernetes can streamline your workflows and enhance the robustness of your deployments.

6. ELK Stack

The ELK Stack is a collection of three open-source tools: Elasticsearch, Logstash, and Kibana, used for searching, analyzing, and visualizing log data in real-time. Often used for logging and log analysis, the ELK Stack enables organizations to gain insights from their data quickly and efficiently.

- **Elasticsearch**: A distributed search and analytics engine built on Apache Lucene. It provides powerful full-text search capabilities and is designed for scalability and high performance.
- **Logstash**: A server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and sends it to a designated "stash" like Elasticsearch.
- **Kibana**: A data visualization tool that works seamlessly with Elasticsearch. It provides a web interface for searching, visualizing, and analyzing data stored in Elasticsearch.

Why and When to Use the ELK Stack?

The ELK Stack is ideal for scenarios involving large volumes of log data, real-time analytics, and the need for comprehensive search capabilities. Specific use cases include:

- **Log and Event Data Analysis**: Aggregating and analyzing logs from various sources like servers, applications, and devices to monitor systems and troubleshoot issues.
- **Security Information and Event Management (SIEM)**: Collecting, analyzing, and visualizing security-related data to detect and respond to security threats.
- **Performance Monitoring**: Tracking and analyzing performance metrics to identify bottlenecks and optimize system performance.
- **Operational Intelligence**: Gaining insights into business operations by analyzing logs and events from various systems.
- **Compliance and Auditing**: Storing and analyzing log data to meet regulatory requirements and perform audits.

Benefits of Using the ELK Stack

- **Centralized Log Management**: Collect and manage logs from various sources in a single, centralized location, simplifying log analysis and management.
- **Scalability**: Elasticsearch's distributed architecture allows it to handle large volumes of data and scale horizontally by adding more nodes.
- **Real-Time Data Processing**: Logstash and Elasticsearch enable real-time data ingestion, transformation, and indexing, providing up-to-date insights.
- **Powerful Search Capabilities**: Elasticsearch offers advanced search features, including full-text search, structured search, and analytics.
- **Flexible Data Ingestion**: Logstash supports numerous input, filter, and output plugins, allowing it to integrate with a wide range of data sources and formats.
- **Rich Data Visualization**: Kibana provides interactive charts, graphs, and dashboards, enabling users to visualize and explore data effortlessly.
- **Open-Source and Extensible**: The ELK Stack is open-source, providing flexibility and extensibility to customize and extend its functionality according to specific needs.
- **Community and Ecosystem**: A vibrant community and a rich ecosystem of plugins and extensions support the ELK Stack, enhancing its capabilities and integration options.

Details About ELK Stack Components

- **Elasticsearch**:
  - **Core Functions**: Indexing, searching, and analyzing large volumes of data quickly and in near real-time.
  - **Features**: Distributed, RESTful search engine; supports multi-tenancy, full-text search, structured search, and analytics.
- **Logstash**:
  - **Core Functions:** Data collection, processing, and forwarding.
  - **Features:** Supports numerous input sources (files, databases, message queues, etc.), filters (grok, mutate, date, etc.), and output destinations (Elasticsearch, databases, etc.).
- **Kibana**:
  - **Core Functions:** Data visualization and exploration.
  - **Features:** Provides interactive visualizations (histograms, line graphs, pie charts, etc.), customizable dashboards, and integration with Elasticsearch queries.

Common Use Cases

- **System Monitoring**: Collecting and analyzing system logs to monitor the health and performance of infrastructure.
- **Application Performance Monitoring (APM)**: Tracking application metrics and logs to identify performance issues and optimize application behavior.
- **Security Analytics**: Analyzing security logs to detect and respond to threats and vulnerabilities.
- **Business Analytics**: Gaining insights from business data to drive decision-making and strategy.

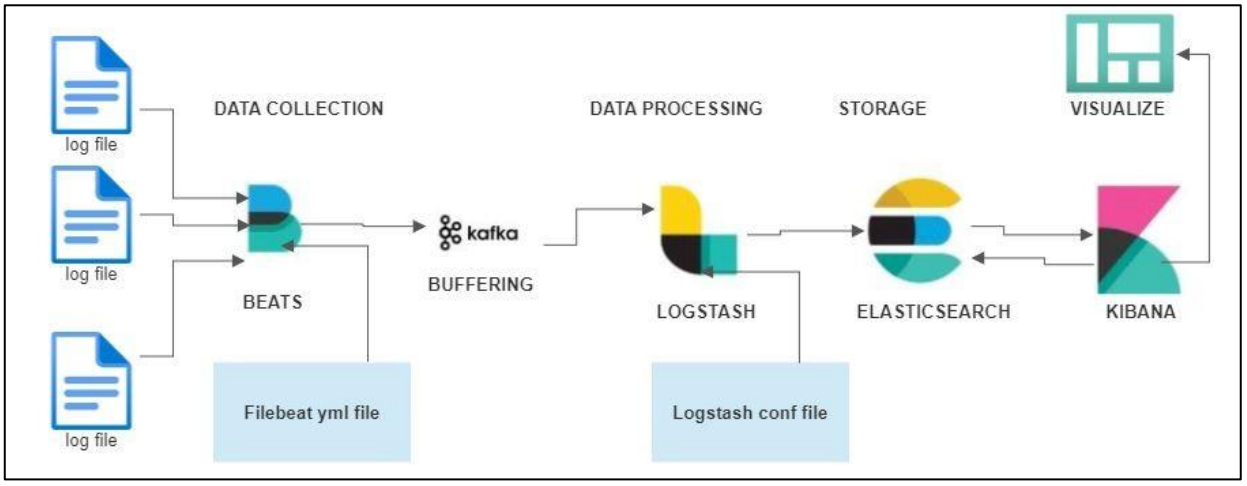


Figure 9: ELK stack - Log Analysis and Monitoring

The ELK Stack is a powerful and flexible toolset for handling log data and deriving valuable insights from it. By centralizing log management, enabling real-time analytics, and providing rich visualizations, the ELK Stack helps organizations improve system monitoring, enhance security, and drive operational intelligence. Its scalability, extensibility, and robust search capabilities make it a preferred choice for managing and analyzing large volumes of log data across various industries and use cases.

7. DevOps Toolchain for the Project

a. Version Control:

- **Tool: GitHub**
- **Description:** GitHub is a web-based platform for version control and collaboration, providing a centralized repository for source code. It enables collaboration, code review, and centralized storage of the project's source code.
- **Role in the Project:** GitHub serves as the version control system, enabling collaboration, code review, and centralized storage of the project's source code.

b. Continuous Integration/Continuous Deployment (CI/CD) Pipeline:

- **Tool: Jenkins**
- **Description:** Jenkins is an open-source automation server that supports building, testing, and deploying code continuously. It provides a framework for continuous integration and continuous delivery (CI/CD) by automating tasks such as code builds, testing, and deployment.
- **Role in the Project:** Jenkins orchestrates the CI/CD pipeline, ensuring that code changes are integrated, tested, and deployed automatically.

c. Build:

Backend Build:

- **Tool: Flask CLI or custom scripts**
- **Description:** Flask applications typically don't require a complex build process like Java projects. You may use Flask CLI commands or custom scripts to handle tasks like setting up the environment, installing dependencies (using pip), and running any pre-deployment tasks like database migrations or static file generation.
- **Role in the Project:** The backend build process ensures that the Flask application is properly configured, dependencies are installed, and any necessary pre-deployment tasks are executed.

Frontend Build:

- **Tool: Node.js/npm (Node Package Manager)**
- **Description:** React applications usually require more involved build processes to transpile JSX, bundle assets, optimize code, and prepare for deployment. Node.js/npm will be used to manage dependencies and run scripts defined in the package.json file, typically including commands like npm install, npm run build, and possibly others for tasks like testing (npm test) or code linting (npm run lint).
- **Role in the Project:** The frontend build process ensures that the React application is compiled, bundled, and optimized for deployment, generating static files that can be served by the backend or a separate server.

d. Testing:

Backend Testing:

- **Tool: Pytest**
- **Description:** Pytest is a widely used testing framework for Python that allows you to write simple and scalable test cases. With Pytest, you can define test functions and use various assertions to verify the behavior of your Flask backend. Tests can cover endpoints, business logic, database operations, and more.
- **Role in the Project:** Pytest is employed for automated testing of the Flask backend, enabling developers to write and execute unit tests, integration tests, and possibly end-to-end tests to ensure the correctness and reliability of the backend code.

Frontend Testing:

- **Tool: Jest**
- **Description:** Jest is a testing framework developed by Facebook for JavaScript code, particularly suited for React applications. It provides utilities for mocking, assertion, and running tests in parallel. Jest is often used alongside React Testing Library or Enzyme for testing React components, allowing you to write unit tests, integration tests, and snapshot tests to ensure the correctness of your React frontend.
- **Role in the Project:** Jest is employed for automated testing of the React frontend, enabling developers to write and execute unit tests and integration tests for React components, ensuring they render correctly, respond to user interactions as expected, and maintain expected behavior across updates.

By using Pytest for backend testing and Jest for frontend testing, you ensure comprehensive test coverage for both parts of your application, allowing you to maintain code quality and catch regressions effectively during development and deployment.

e. Containerization:

- **Tool: Docker**
- **Description:** Docker is a containerization platform that allows applications and their dependencies to be packaged into lightweight, portable containers. Containers provide consistency and portability across different environments.
- **Role in the Project:** Docker is used for containerization, enabling the packaging of the application and its dependencies into a container, ensuring consistent deployment.

f. Container Registry:

- **Tool: DockerHub**
- **Description:** Container registries store Docker images, enabling efficient distribution and deployment of containerized applications.
- **Role in the Project:** Docker Hub (or alternatives) securely stores Docker images, with Docker Hub credentials used for authentication during image push operations.

g. Kubernetes Deployment:

- **Environment: Kubernetes Cluster**
- **Description:** Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It provides features such as automatic scaling, rolling updates, service discovery, and load balancing, making it ideal for deploying and managing microservices-based applications.
- **Role in the Project:** Kubernetes is used to deploy both the frontend and backend components of the application in a production-like environment. It ensures high availability, scalability, and reliability by managing containerized instances of the application, handling traffic routing, and providing self-healing capabilities in case of failures.

h. Container Orchestration:

- **Environment: Kubernetes Cluster**
- **Description:** Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It provides features such as automatic scaling, rolling updates, service discovery, and load balancing, making it ideal for deploying and managing microservices-based applications. Kubernetes abstracts away the underlying infrastructure complexities and provides a unified API for managing containerized workloads across on-premises, cloud, and hybrid environments.
- **Role in the Project:** Kubernetes plays a critical role in orchestrating the deployment and management of both frontend and backend components of the application. It ensures high availability, scalability, and reliability by automatically scheduling containers, managing resource allocation, and handling network communication between different parts of the application. Kubernetes simplifies the deployment process, improves resource utilization, and enables seamless scaling to meet varying workload demands, making it an essential component of the project's infrastructure.

i. Monitoring & Logging:

- **Tool: ELK Stack (Elasticsearch, Logstash, Kibana)**
- **Description:** The ELK Stack is a popular open-source solution for log management and real-time analytics. It consists of three main components:
  - **Elasticsearch:** A distributed, RESTful search and analytics engine designed for horizontal scalability, real-time search, and analysis of structured and unstructured data.
  - **Logstash:** A data processing pipeline that ingests, transforms, and enriches log data from various sources before sending it to Elasticsearch for indexing and storage.
  - **Kibana:** A web-based visualization and exploration tool that provides a user-friendly interface for querying, analyzing, and visualizing data stored in Elasticsearch. It offers features such as dashboards, charts, and graphs for monitoring and troubleshooting.
- **Role in the Project:** The ELK Stack is used for monitoring and logging in the project. Elasticsearch serves as the centralized repository for storing and indexing logs generated by both frontend and backend components. Logstash processes and enriches log data before sending it to Elasticsearch, ensuring consistency and accuracy. Kibana provides a graphical interface for developers and operations teams to search, analyze, and visualize log data in real-time, facilitating troubleshooting, performance monitoring, and trend analysis. Additionally, Fluent Bit may be used as a lightweight log shipper to collect logs from various sources and forward them to Logstash for processing, enhancing the scalability and efficiency of the logging infrastructure.

j. Load Balancing:

- **Tool: Kubernetes' Horizontal Pod Autoscaler (HPA)**
- **Description:** Load balancing distributes incoming network traffic to maintain high availability, reliability, and scalability. It evenly distributes workload, prevents server overload, and boosts performance. Our project utilizes Kubernetes' Horizontal Pod Autoscaler (HPA) for dynamic scaling based on CPU utilization, ensuring efficient resource allocation.
- **Role:** Crucial for efficient traffic distribution between backend and frontend components. HPA scales pod replicas, while load balancing routes traffic across these instances. Kubernetes' service load balancer ensures even distribution, maintaining application availability, reducing latency, and enhancing user experience. Additional features like session persistence and health checks ensure optimal performance and reliability.

By integrating these tools and practices into the DevOps toolchain, the project automates the software development lifecycle, ensuring reliability, scalability, and efficiency from code commit to production deployment.

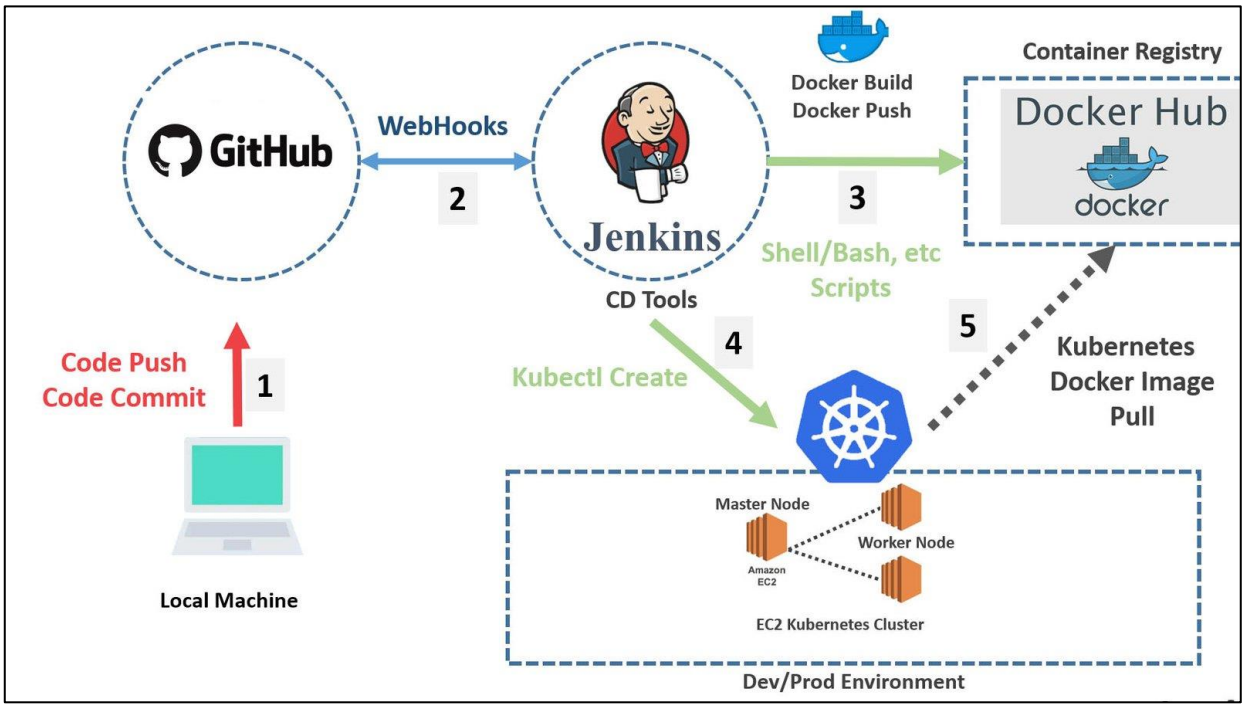


Figure 10: DevOps Toolchain for the Project

k. Automation Plugins or Tools:

i. Webhooks

- **Description:** Webhooks are user-defined HTTP callbacks triggered by specific events or actions on a web application. When an event occurs, a webhook sends real-time data via an HTTP POST request to a specified URL endpoint. This allows for the seamless integration of different services and the automation of workflows.
- **Usage in DevOps:** Webhooks can be configured with Jenkins to trigger automated actions in response to specific events, enhancing the automation and efficiency of the continuous integration and delivery (CI/CD) pipeline.

ii. Ngrok

- **Description**: Ngrok is a popular tool designed to create secure tunnels from a public endpoint to a locally running web server. It establishes a secure connection, allowing developers to expose a local web server to the public internet through a secure tunnel. Ngrok supports HTTP, HTTPS, and TCP protocols and provides a user-friendly interface for managing tunnels and inspecting traffic.
- **Usage in DevOps:** Ngrok simplifies the development and testing of web applications locally by eliminating the need to deploy applications to public servers. It is widely used for testing, debugging, integrating with third-party services, and securely exposing internal services to the public internet.

iii. Nginx

- **Description**: Nginx is a high-performance web server and reverse proxy server renowned for its stability, rich feature set, simple configuration, and low resource consumption. It can serve static content, handle reverse proxying for HTTP and HTTPS, load balancing, and provide caching.
- **Usage in DevOps**: In a DevOps environment, Nginx is often used as a reverse proxy to manage and direct traffic, enhance security, balance load across multiple servers, and improve the performance and reliability of web applications. It integrates well with CI/CD pipelines to ensure that deployments are efficiently and securely managed.

These automation tools—Webhooks, Ngrok and Nginx —play crucial roles in enhancing the automation, integration, and testing processes within the DevOps lifecycle. They contribute to the efficiency of development and deployment workflows by enabling real-time data exchange, secure tunneling, efficient traffic management, and robust database management.

l. MySQL Database

- **Description**: MySQL is an open-source relational database management system (RDBMS) based on Structured Query Language (SQL). It is widely used for managing and organizing data, supporting transactions, and ensuring data integrity and security.
- **Usage in DevOps**: MySQL is frequently used in DevOps for managing application databases. It supports automated database setup, migration, and seeding as part of CI/CD pipelines. MySQL's robust transaction and replication capabilities ensure high availability and data consistency in development and production environments.

8. Process Flow

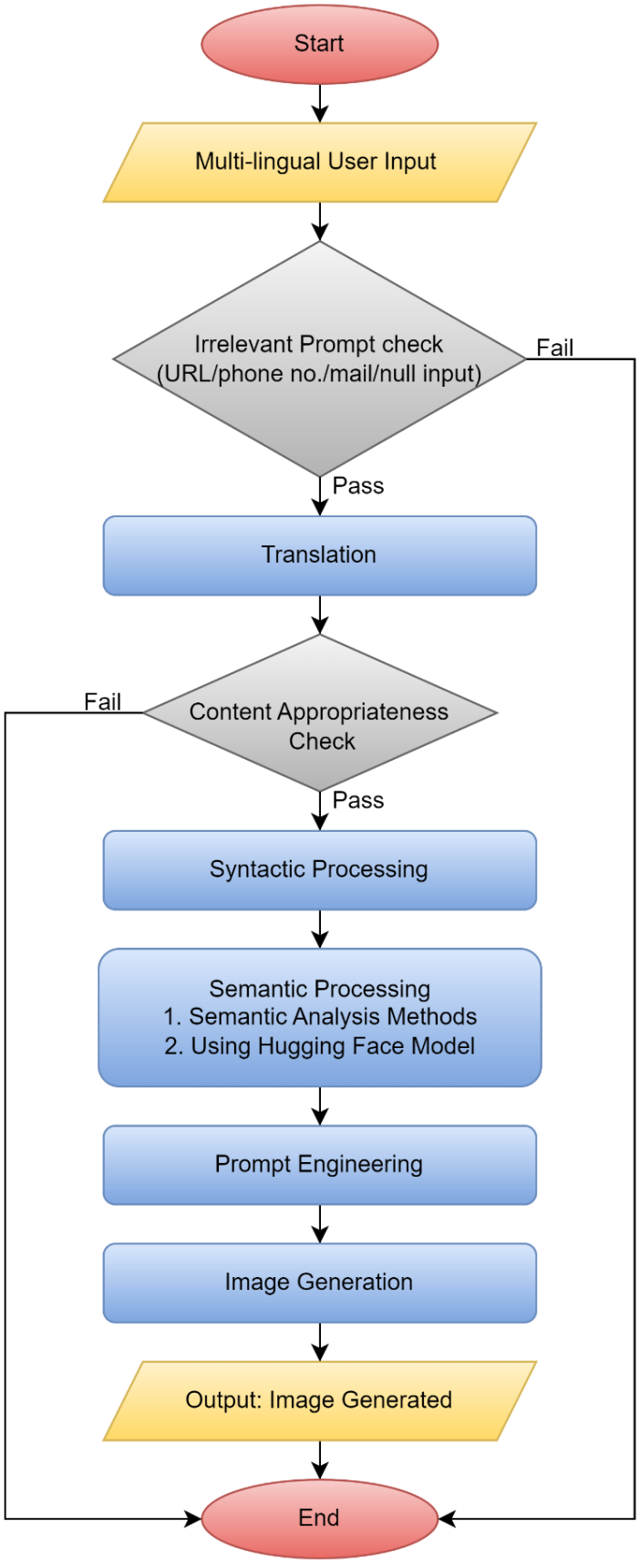


Figure 11: Process Flow of the Application

9. Application Overview and Features

i. Irrelevant Input Filtration

Filtering out **URLs, phone numbers, and email addresses** is important for a multilingual text-to-image generator to ensure that the generated images focus solely on the linguistic content of the input text, regardless of the language used. URLs, often appearing as hyperlinks or web addresses, can introduce noise and distractions into the input text, hindering the effectiveness of syntactic analysis algorithms. By identifying and removing these elements, the input text can focus on meaningful linguistic content rather than irrelevant web addresses. Similarly, phone numbers and email addresses are typically considered metadata rather than linguistic content, and their presence can interfere with syntactic analysis. Removing these auxiliary pieces of information ensures that the input text contains only relevant linguistic content, allowing syntactic processing algorithms to operate more effectively and produce more accurate results. Therefore, incorporating mechanisms to filter out URLs, phone numbers, and email addresses is essential for optimizing syntactic processing workflows.

ii. Multi-Lingual Text Translation

Machine translation ensures accurate conversion of text from various source languages into English. It tackles challenges arising from linguistic and cultural nuances, aiming to convey the intended meaning faithfully. Prominent tools in this domain include Google Translate, Meta AI Translator, and DeepL, which leverage advanced algorithms to facilitate seamless communication across different languages. These tools play a crucial role in breaking down language barriers and fostering global connectivity by providing efficient and reliable translation services.

Preprocessing Steps for Multi-lingual Translation:

- Language Identification: Identify source language(s).
- Tokenization and Normalization: Break text into units and normalize.
- Sub-word Segmentation: Handle rare words.
- Multilingual Morphological Analysis: Analyze morphology.
- Multilingual Part-of-Speech Tagging: Assign grammatical categories.
- Multilingual Named Entity Recognition: Identify named entities.
- Multilingual Word Sense Disambiguation: Determine word meanings.
- Multilingual Representation: Encode text for translation.
- The Multilingual Translation Model: Employ a translation model.
- Post-processing and Language-specific Adjustments: Refine the output.

Utilizing pretrained translator tools in natural language processing offers efficiency, leveraging existing knowledge and resources, saving time and computational effort. These models capture broad language patterns and semantics, enabling effective generalization across diverse domains and linguistic nuances. Pretrained models excel in transferring knowledge across languages, even with limited training data, ensuring superior translation quality. Among the options, Google Translate stands out for its consistent high performance, availability across numerous language pairs, and continuous improvements through user feedback and research advancements. Its NLP-based approach effectively captures complex linguistic patterns, making it a reliable choice for various translation tasks.

Google Translation API utilizes Neural Machine Translation (NMT) trained on large datasets and tokenizes text while embedding it for language processing. It leverages language modeling to grasp linguistic patterns and considers contextual understanding for accurate translations. Through continuous fine-tuning and optimization, it addresses polysemy and ambiguity, adapting and improving over time for enhanced translation quality.

iii. Offensive Language Detection

Offensive language detection involves identifying and flagging text with offensive, abusive, or inappropriate content using linguistic analysis, context, and sentiment evaluation. This process is vital for moderating online content, ensuring user safety, and promoting a positive and inclusive digital environment.

Importance in our Project

In the context of a text-to-image generation project, ensuring the appropriateness of user input remains crucial for several reasons:

- **Ethical Content Creation:** Prevents the generation of inappropriate or offensive images, maintaining ethical standards and contributing positively to the community.
- **User Protection:** Shields users, especially vulnerable ones, from harmful or offensive imagery, ensuring a safe and positive experience.
- **Brand Reputation:** Protects the project's or platform's reputation by avoiding the association with negative or offensive content, maintaining trust and credibility.
- **Compliance with Guidelines:** Adheres to content moderation guidelines and legal regulations, reducing legal risks and ensuring regulatory compliance.
- **Enhanced User Experience:** Filters out offensive content, creating a welcoming and enjoyable environment that encourages user engagement.
- **Prevention of Misuse:** Prevents the creation of harmful imagery that could be used maliciously, promoting responsible platform use.

Text Moderation Model from Koala

Several models are available for offensive language detection, including fine-tuned BERT-based models and OpenAI's GPT-3, which can recognize offensive content with appropriate training. Jigsaw's Perspective API scores comments for toxicity, while KoalaAI offers text moderation services using machine learning and natural language processing techniques.

The **KoalaAI/Text-Moderation** model, based on Deberta-v3 architecture, classifies text into categories like "sexual content," "hate speech," "violence," and "harassment." Known for its nuanced multi-class classification, it excels in text moderation. Accessible via Hugging Face and licensed under OpenRAIL-M, it supports commercial use while ensuring ethical compliance. Trained on the mmathys/openai-moderation-api-evaluation dataset and other datasets, it aims for high precision and recall, achieving an accuracy of 74.9% and a weighted F1 score of 0.703. It is environmentally friendly, with minimal CO2 emissions during training.

Category	Label	Definition
sexual	S	Content meant to arouse sexual excitement, such as the description of sexual activity, or that promotes sexual services (excluding sex education and wellness).
hate	H	Content that expresses, incites, or promotes hate based on race, gender, ethnicity, religion, nationality, sexual orientation, disability status, or caste.
violence	V	Content that promotes or glorifies violence or celebrates the suffering or humiliation of others.
harassment	HR	Content that may be used to torment or annoy individuals in real life, or make harassment more likely to occur.
self-harm	SH	Content that promotes, encourages, or depicts acts of self-harm, such as suicide, cutting, and eating disorders.
sexual/minors	S3	Sexual content that includes an individual who is under 18 years old.
hate/threatening	H2	Hateful content that also includes violence or serious harm towards the targeted group.
violence/graphic	V2	Violent content that depicts death, violence, or serious physical injury in extreme graphic detail.
OK	OK	Not offensive

Workflow for Content Filtration Using the Model

- **Calculate Offensive Filtration Score:** Assess the likelihood of text being offensive using the model's output.
- **Determine Offensive Categorization:** Compare the offensive score with other category scores. If the offensive score is lower, classify the prompt as offensive; otherwise, as non-offensive.
- **Generate Image:** Generate the image if the prompt is non-offensive; otherwise, prompt the user to try again.
- **Feedback to User:** Inform the user about the result, presenting the image if generated, or asking for a different prompt if deemed offensive.

iv. Semantic Analysis

Semantic analysis in NLP involves understanding text by examining context, word relationships, and structure. It encompasses tasks like sentiment analysis, named entity recognition, and word sense disambiguation. Emotional tone assessment, a subset of semantic analysis, focuses on determining the emotional sentiment conveyed in text, crucial for tasks like text-to-image generation where capturing emotional context is vital for creating relevant visuals.

Importance in our Project

Semantic analysis is crucial in text-to-image generation for several reasons:

- **Semantic Understanding:** Enables AI to comprehend text meaning and context, ensuring accurate image generation.
- **Content Relevance:** Ensures generated images are coherent with the textual content.
- **Detail Incorporation:** Identifies and incorporates important details from the text into the images.
- **User Intent Alignment:** Aligns generated images with user intentions and expectations.
- **Reduced Ambiguity:** Clarifies text meaning, reducing misinterpretation and improving accuracy.
- **Personalization and Customization:** Tailors images based on text semantics and user preferences.

These factors enhance the effectiveness, relevance, and user satisfaction of text-to-image generation systems.

Chosen Model Overview

Several models are available for emotion classification and sentiment analysis: BERT, known for strong performance but high resource demands; RoBERTa, an optimized version of BERT with improved accuracy; ALBERT, a lightweight BERT variant with reduced complexity; XLNet, which captures bidirectional context through permutation-based training; and GPT models like GPT-2 and GPT-3, which excel at text generation and can be fine-tuned for sentiment analysis.

The distilbert-base-uncased-go-emotions-student model is a variant of DistilBERT fine-tuned for emotion classification in text. It efficiently analyzes emotional content, offering scores for a range of emotions. Its speed, especially on CPU, makes it notable. By categorizing emotions as positive or negative, it helps understand the text's sentiment. Trained on the GoEmotions dataset, it predicts emotion probabilities for input text, facilitating insights into various emotion categories like admiration, curiosity, surprise, sadness, fear, and anger.

The joeddav/distilbert-base-uncased-go-emotions-student model is an ideal choice for emotion classification due to its specialized training on the GoEmotions dataset, efficient DistilBERT architecture, widespread community adoption, and comprehensive coverage of emotion categories. Its tailored design ensures accurate detection of a wide range of emotions, making it a reliable tool for analyzing textual sentiment.

Workflow for Semantic Analysis Using the Model

- **Text Input and Model Prediction:** Input text into the distilbert-base-uncased-go-emotions-student model to predict probabilities for various emotions.
- **Assessment of Probabilities:** Evaluate each emotion's probability against predefined positive and negative lists, categorizing them accordingly.
- **Aggregation of Probabilities:** Aggregate probabilities separately for positive and negative emotions using the sentiment_score function.
- **Comparative Analysis:** Compare sums of positive and negative emotion probabilities to determine overall sentiment. A higher sum of positive emotions indicates a positive sentiment, while a higher sum of negative emotions suggests a negative tone.
- **Nuanced Understanding:** This approach provides a nuanced understanding of the emotional context, facilitating deeper comprehension and interpretation of the author's sentiment for sophisticated analysis and response strategies.

Examples:

- **Positive Sentiment:** "The joyous celebration filled the room with laughter and happiness." Emotions like joy, celebration, laughter, and happiness are classified as positive, resulting in a predominantly positive sentiment.
- **Negative Sentiment:** "The tragic news left everyone in shock and despair." Emotions such as tragedy, shock, and despair are classified as negative, leading to a predominantly negative sentiment.

Visualization: A graph visually represents the distribution of positive and negative emotions, offering clear insight into the text's emotional tone. Incorporating sentiment analysis into text-to-image generation ensures alignment with intended emotional context, enhancing visual representations' meaning and engagement.

v. Prompt Engineering

Prompt engineering involves crafting inputs to AI models to optimize output relevance and effectiveness. Precision in prompts is crucial, ensuring outputs are accurate, relevant, and varied. Techniques like keyword optimization, negative prompting, iterative refinement, and contextual framing enhance prompt quality.

In our project, a Filter Panel aids prompt engineering by allowing users to specify parameters like image quantity, quality, art style, theme, location, weather, and photography technique. This panel facilitates a dialogue between users and AI, fostering collaborative content creation. To improve prompt engineering, future considerations include integrating user feedback loops, AI learning, granular control, and guided tutorials. The Filter Panel exemplifies prompt engineering's potential to revolutionize AI interactions, enabling creative collaboration between human intuition and machine intelligence.

vi. Image Generation Model

Latent Diffusion Models (LDMs)

Latent Diffusion Models (LDMs) streamline the training and sampling process of Diffusion Models (DMs) for high-resolution image synthesis by introducing a more computationally efficient latent space obtained through an autoencoder. Key contributions include enabling efficient exploration of diffusion models for image-to-image and text-to-image tasks, achieving competitive performance with reduced computational costs, eliminating the need for delicate weighting between reconstruction and generative abilities, allowing convolutional application for densely conditioned tasks, and designing a general-purpose conditioning mechanism based on cross-attention. By separating compressive and generative learning phases, LDMs significantly reduce computational demands while maintaining synthesis quality. Experimental results demonstrate their effectiveness across various datasets and tasks, offering a promising approach to improving the efficiency and accessibility of diffusion models for image synthesis.

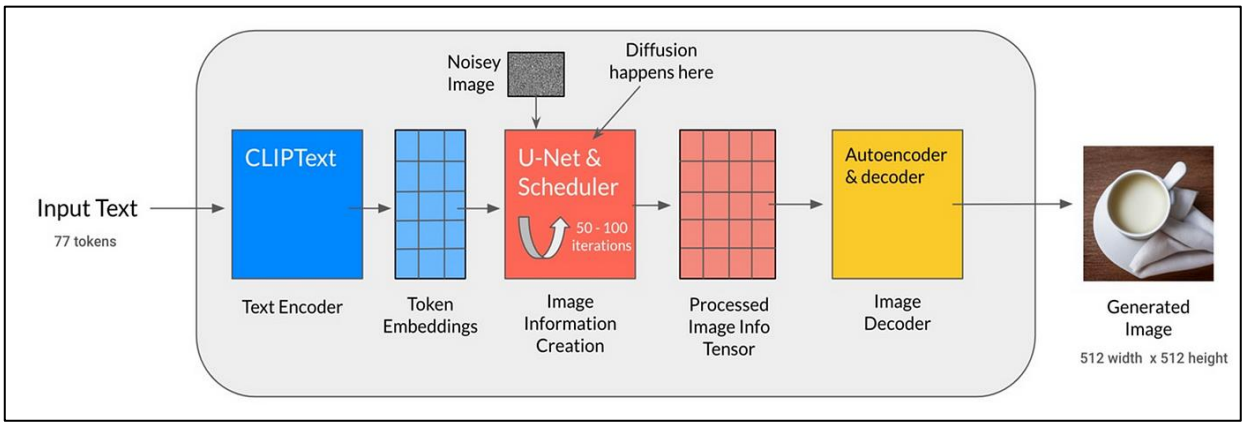


Figure 12: Stable Diffusion Architecture

Stable Diffusion 3

Stability AI's Stable Diffusion 3 is the latest advancement in text-to-image generative AI, offering improved performance in multi-subject prompts, image quality, and spelling abilities. Available via API today, it features a suite of models ranging from 800 million to 8 billion parameters, catering to diverse creative needs. Utilizing a diffusion transformer architecture, Stable Diffusion 3 combines diffusion models for detail with transformers for layout, resulting in better organization of complex scenes. Key enhancements include faster image generation, more effective sampling techniques, and improved text rendering capabilities.

While Stable Diffusion 3 outperforms its predecessors and competitors in various performance assessments, it still has limitations such as occasional inconsistencies in text generation and image patches. Accessible in an early preview stage, researchers can provide feedback on its performance and safety before public release. Despite its potential, questions remain regarding copyright issues, technical details, and the incorporation of techniques like recaptioning for prompt engineering. Overall, Stable Diffusion 3 represents another stride forward in text-to-image generative AI, promising new use cases and advancements upon its public release.

10. Development Steps

i. Setup Development Environment

Follow the steps to implement the pipeline as per our DevOps tool chain:

1. Set up Git and GitHub:

- Create a Git repository to host the project code and configure GitHub for code storage.
- Establish a branch strategy, defining workflows for branching, merging, and pull requests to ensure efficient collaboration.
- Implement a release process to streamline code deployment, ensuring smooth transitions between development, testing, and production environments.

2. Install and Configure Jenkins:

- Install Jenkins on a server and connect it to the Git repository.
- Create Jenkins jobs for building, testing, and deploying code.
- Utilize Jenkins plugins for automation and streamline the CI/CD pipeline.

3. Set up Docker:

- Establish a Docker registry for storing images.
- Create Docker images for the application, enabling deployment in various environments.
- Utilize Docker containers for running the application in development, testing, and production environments.

4. Test and Deploy:

- Leverage the DevOps toolchain to conduct comprehensive testing, including unit tests, integration tests, and end-to-end tests, ensuring the application meets quality standards.
- Implement continuous monitoring using ELK stack (Elasticsearch, Logstash, Kibana) to track application performance, identify issues, and troubleshoot effectively.
- Deploy the application seamlessly across development, testing, and production environments using Kubernetes, ensuring consistency and reliability in deployment processes.

5. Continuously Improve:

- Evaluate and enhance the DevOps toolchain, processes, and workflows continuously.
- Use data and feedback to identify improvement areas and optimize DevOps practices,

ii. Source Control Management Setup

Source Control Management (SCM) is pivotal for maintaining software stability while allowing concurrent development. GitHub, a central repository, facilitates collaboration and code versioning. Its branching and pull request features enable parallel work on new features or repairs, crucial for effective development. GitHub's role extends to establishing a release process, ensuring systematic code deployment. It acts as a collaborative hub, harmonizing diverse efforts while safeguarding baseline integrity. GitHub, in essence, stands as a crucial ally in the perpetual cycle of systematic and collaborative software development.

GitHub Remote Repository:

Create a new repository after logging into your GitHub account.

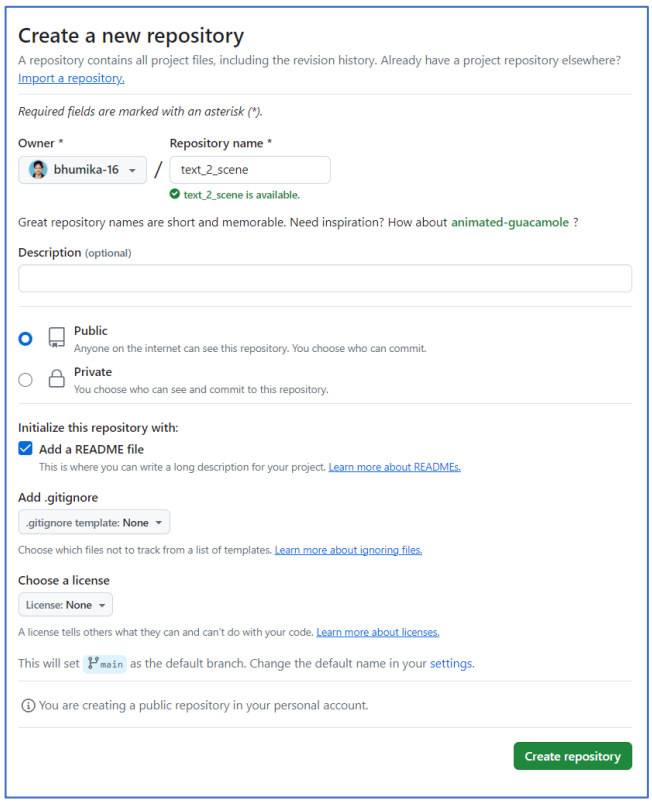


Figure 13: Creating a GitHub Remote Repository

Git - Local Repository:

Create a local repository for source code management. Since Git is a distributed system, version history is managed at both local and remote repositories. Run these commands in the working directory of your application.

Follow the steps below to create a local repository and link it to the remote repository:

```
echo "# text-2-scene" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/your_username/text2scene.git
git push -u origin main
```

Develop the project code according to requirements and commit the changes to the local repository using git add . and git commit -m "some message". Finally, push the changes to the remote repository using git push -u origin <branch_name>.

When prompted for a password, use your GitHub Personal Access Token (PAT) as the password, and your GitHub username as the username. The commit history figures illustrate the development stages from the initial setup to the final setup, including commit IDs. At any stage, developers can checkout to a previous commit for reference or rollback.

iii. Docker and Docker-Hub

Docker is a robust software platform designed for swift application building, testing, and deployment. It employs containers, standardized units encompassing all essentials for software execution – libraries, tools, code, and runtime. Docker enables rapid, consistent deployment across diverse environments, ensuring the seamless operation of your code.

Creating a Docker Hub Account:

- Sign Up: Visit Docker Hub and sign up for an account. Provide necessary details to create your account.
- Repository Creation:
- Login: Log in to Docker Hub using your newly created account.
- Navigate to Repositories: Find the "Repositories" tab and click on it.
- Create Repository: Select "Create Repository" and fill in details. Choose visibility (public/private) and create the repository.

Note: Docker Hub repositories serve as storage for your Docker images, allowing you to push and pull images as needed.

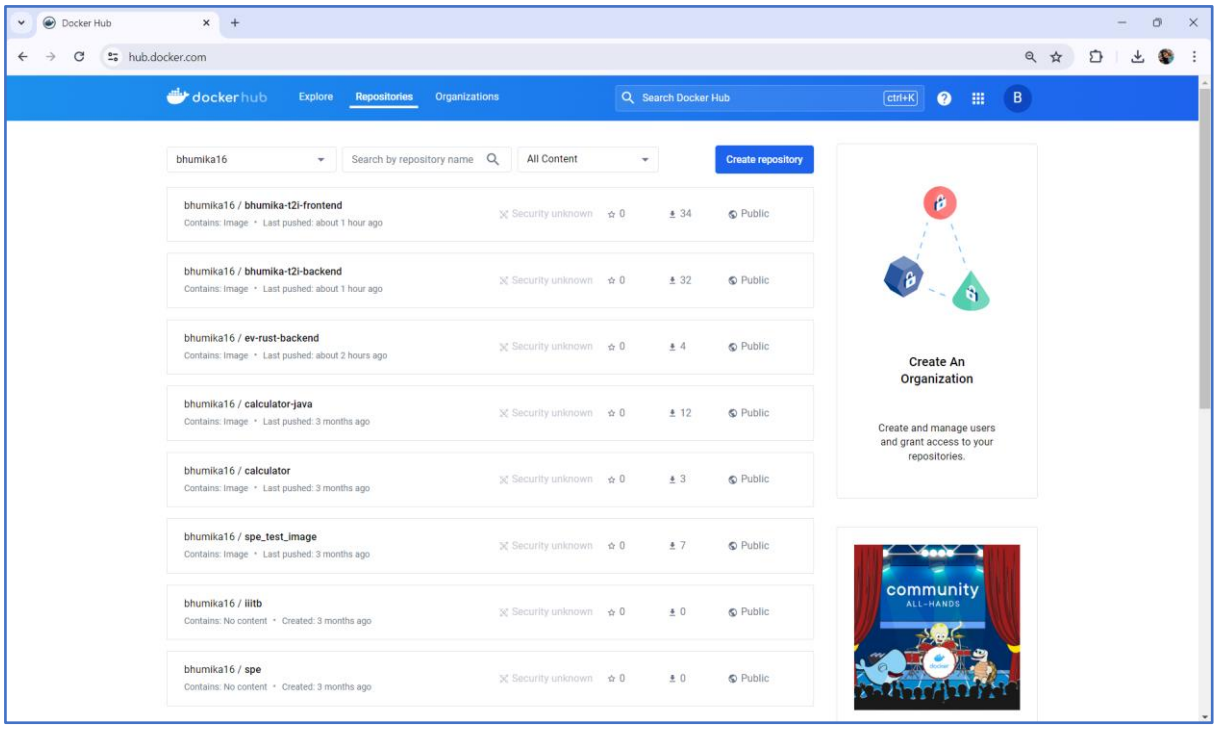


Figure 14: Docker-Hub Repositories

iv. Jenkins

Jenkins is an open-source automation server that facilitates continuous integration and delivery (CI/CD) processes. It automates building, testing, and deploying code changes, ensuring rapid and reliable software development. With a vast plugin ecosystem, Jenkins supports integration with various tools, making it a cornerstone in modern DevOps practices.

Installing Jenkins on Ubuntu:

- **Update Package List:**

```
sudo apt update
```

- **Install Java:** Jenkins requires Java. Install OpenJDK 21:

```
sudo apt install openjdk-21-jdk
```

- **Add Jenkins Repository Key:**

```
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
```

- **Add Jenkins Repository:**

```
sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
```

- **Update Package List:**

```
sudo apt update
```

- **Install Jenkins:**

```
sudo apt install jenkins
```

- **Start Jenkins Service:**

```
sudo service jenkins start
```

- **Check Jenkins Service Status:**

```
sudo service jenkins status
```

If active, you should see "active (running)."

- **Access Jenkins in Browser:**
  Open your browser and go to http://localhost:8080. You'll see the Jenkins login page.

- **Unlock Jenkins:** Find the initial admin password:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

  Copy and paste the password on the Jenkins login page.

- **Follow Jenkins Setup Wizard:**
  The wizard guides you through the installation. Set up your Jenkins instance as prompted.

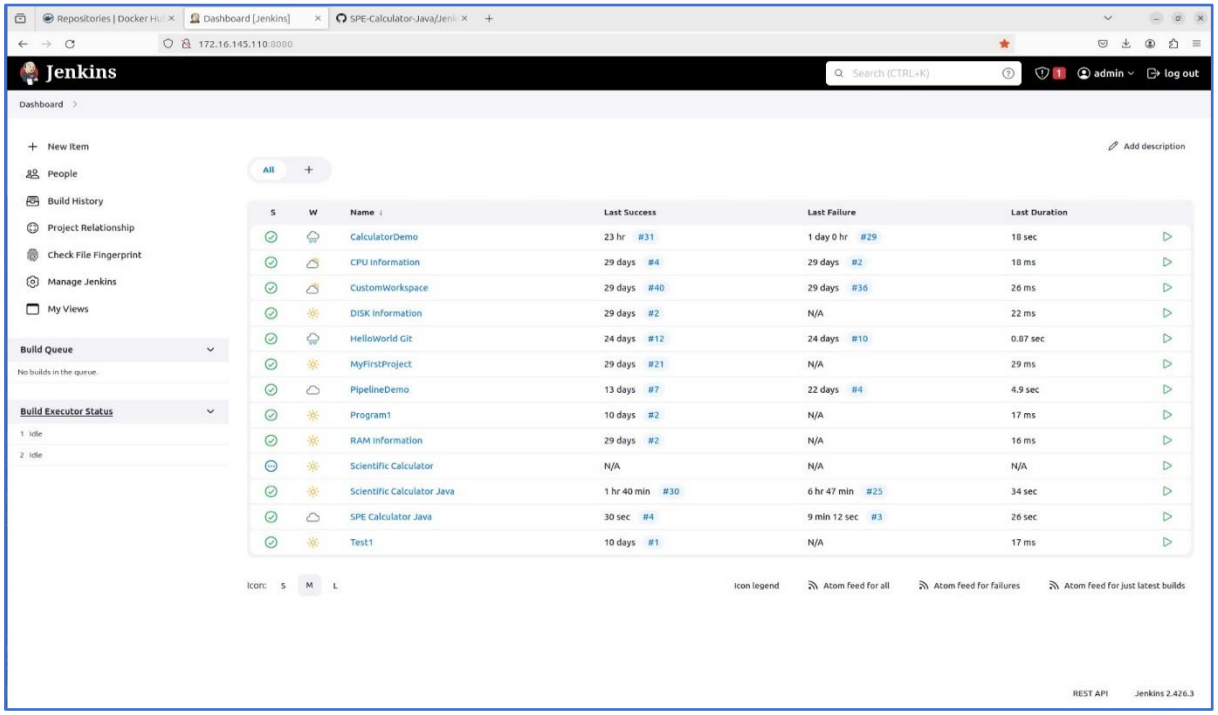


Figure 15: Jenkins Dashboard

Install Required Jenkins Plugins:

Maven, Git, Ansible, Docker:

- Open Jenkins, go to "Manage Jenkins" -> "Manage Plugins" -> "Available."
- Search and install Maven, Git, Ansible, and Docker plugins.

Global Tool Configuration:

Manage Jenkins:

- Navigate to "Manage Jenkins" -> "Global Tool Configuration."

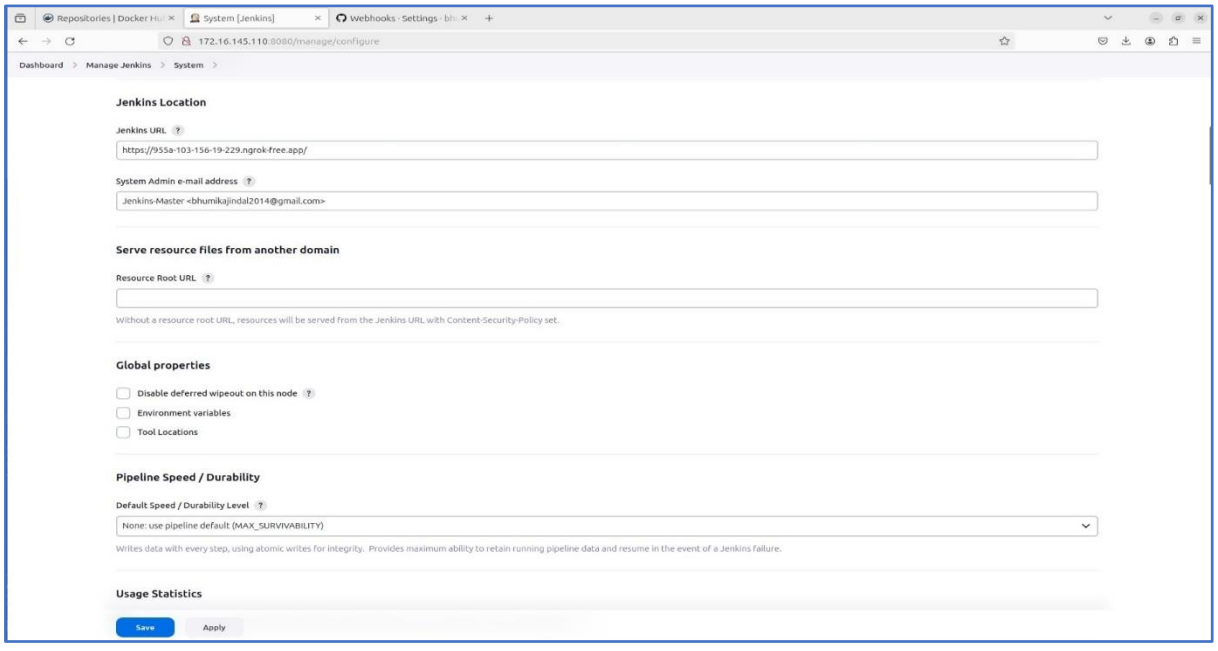


Figure 16: Jenkins Global Tool Configuration

Add GitHub and Dockerhub Credentials:

1. **Manage Jenkins >> Configure System:**

- Go to "Manage Jenkins" -> "Configure System."

2. **Add GitHub Server:**

- In the "GitHub" section, add the GitHub server using credentials set up earlier.

Ensure credentials for GitHub and Dockerhub are correctly configured to enable Jenkins access. These clear steps ensure a smooth installation of Jenkins on Ubuntu, including the addition of essential plugins and global tool configurations. The setup allows seamless integration with GitHub and Dockerhub for efficient DevOps practices.

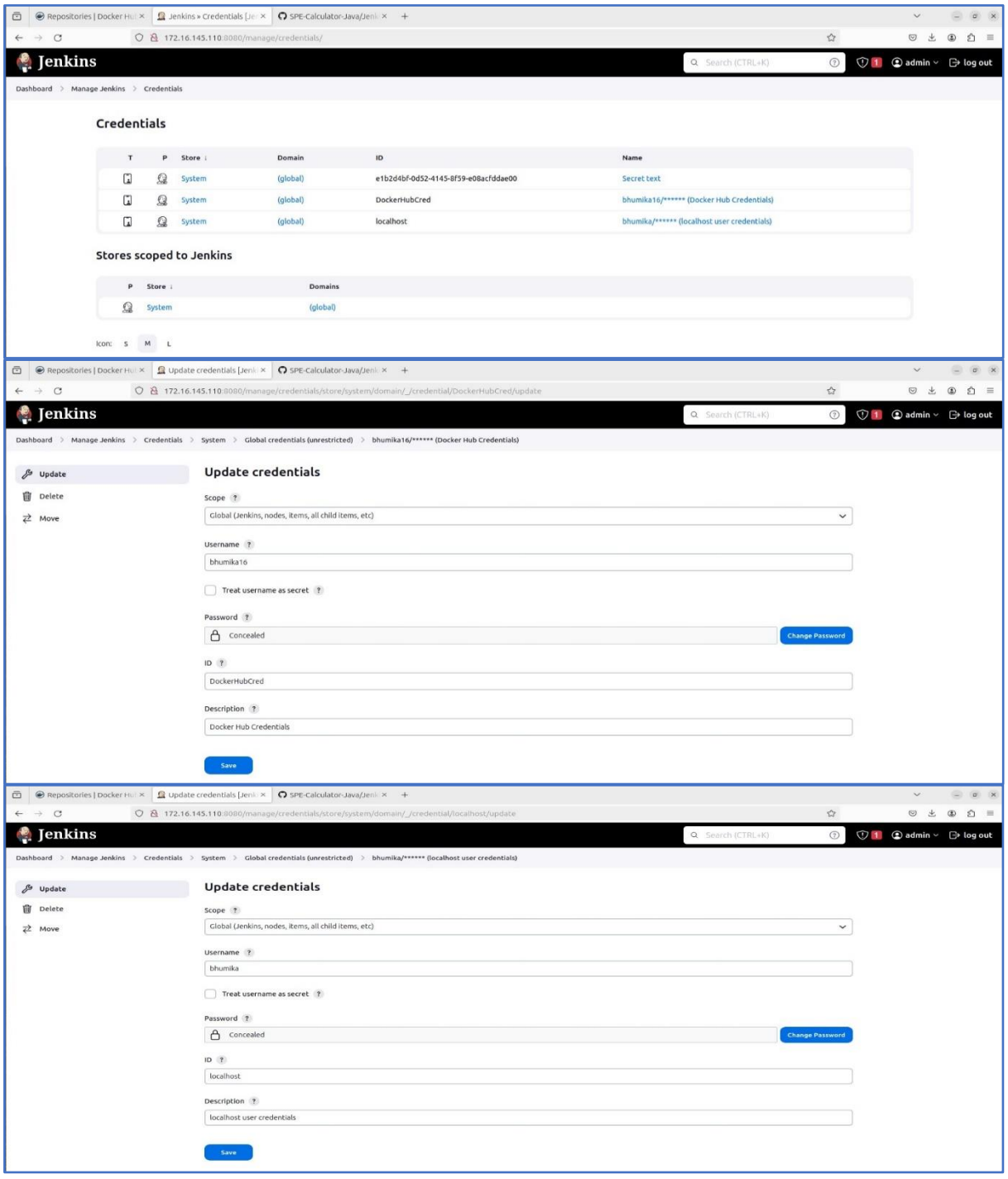


Figure 17: Add DockerHub and Localhost Credentials

v. CI/CD Pipeline

A continuous integration and continuous deployment (CI/CD) pipeline are a series of steps that must be performed in order to deliver a new version of software. CI/CD pipelines are a practice focused on improving software delivery throughout the software development life cycle via automation. Here is the pipeline script for the application that orchestrates the setup of a comprehensive DevOps toolchain from configuring Git and Jenkins to setting up Docker and Kubernetes for automation along with the ELK Stack for Logging and Monitoring.

Jenkins_final

text2scene-main > Jenkins_final

```
pipeline {
    agent any
    environment {
        KUBECONFIG = "fix/kubeconfig"
        DOCKER_IMAGE_TAG = "${currentBuild.number}"
        BACKEND_IMAGE = "bhumika-t2i-backend:${DOCKER_IMAGE_TAG}"
        FRONTEND_IMAGE = "bhumika-t2i-frontend:${DOCKER_IMAGE_TAG}"
    }
    stages {
        stage("Code Fetch") {
            steps {
                git url: "https://github.com/bhumika-16/text2scene.git", branch: "main"
            }
        }
        stage("Build Backend Docker Image") {
            steps {
                bat "docker build -t bhumika-t2i-backend:${DOCKER_IMAGE_TAG} src/backend"
            }
        }
        stage("Testing Backend application") {
            steps {
                bat "docker run --rm ${BACKEND_IMAGE} pytest ."
            }
        }
        stage("Build Frontend Docker Image") {
            steps {
                bat "docker build -t bhumika-t2i-frontend:${DOCKER_IMAGE_TAG} src/frontend"
            }
        }
       stage("Testing Frontend application") {
            steps {
               bat "docker run --rm ${FRONTEND_IMAGE} npm run test:ci"
            }
        }

        stage("Push Backend Docker Images") {
            steps {
                withCredentials([usernamePassword(credentialsId:"DockerHubCred",passwordVariable:"dockerpassword",
                usernameVariable:"dockerusername")]){
                    bat "docker login -u ${env.dockerusername} -p ${env.dockerpassword}"
                    bat "docker tag bhumika-t2i-backend:${DOCKER_IMAGE_TAG} ${env.dockerusername}/bhumika-t2i-backend"
                    bat "docker push ${env.dockerusername}/bhumika-t2i-backend"
                }
            }
        }
        stage("Push Frontend Docker Images") {
            steps {
                withCredentials([usernamePassword(credentialsId:"DockerHubCred",passwordVariable:"dockerpassword",
                usernameVariable:"dockerusername")]){
                    bat "docker login -u ${env.dockerusername} -p ${env.dockerpassword}"
                    bat "docker tag bhumika-t2i-frontend:${DOCKER_IMAGE_TAG} ${env.dockerusername}/bhumika-t2i-frontend"
                    bat "docker push ${env.dockerusername}/bhumika-t2i-frontend"
                }
            }
        }
        stage("Deploy ELK") {
            steps {
                bat "kubectl apply -f manifests/deploy/fluentbit-deploy.yaml"
                bat "kubectl apply -f manifests/deploy/logtash-deploy.yaml"
                bat "kubectl apply -f manifests/deploy/elasticsearch-deploy.yaml"
                bat "kubectl apply -f manifests/deploy/kibana-deploy.yaml"
                bat "kubectl apply -f manifests/svc/elasticsearch-svc.yaml"
                bat "kubectl apply -f manifests/svc/logtash-svc.yaml"
                bat "kubectl apply -f manifests/svc/kibana-svc.yaml"
            }
        }

```

```
        stage("Deploy Kubernetes") {
            steps {
                withCredentials([usernamePassword(credentialsId:"DockerHubCred",passwordVariable:"dockerpassword",
                usernameVariable:"dockerusername")]){
                    bat "kubectl apply -f manifests/deploy/mysql-deploy.yaml"
                    bat "kubectl apply -f manifests/deploy/nginx-deploy.yaml"
                    bat "kubectl apply -f manifests/deploy/backend-deploy.yaml"
                    bat "kubectl apply -f manifests/deploy/frontend-deploy.yaml"
                }
            }
        }
        stage("Set Ingress rules for Frontend and Backend") {
            steps {
                bat "kubectl apply -f manifests/ingress/ingress-backend.yaml"
                bat "kubectl apply -f manifests/ingress/ingress-frontend.yaml"
            }
        }
        stage("Deploy Kubernetes services") {
            steps {
                bat "kubectl apply -f manifests/svc/backend-service.yaml"
                bat "kubectl apply -f manifests/svc/frontend-service.yaml"
                bat "kubectl apply -f manifests/svc/nginx-svc.yaml"
            }
        }
         stage("Setting up autoscaling") {
            steps {
                script {
                    def hpaBackendExists = bat (
                        script: 'kubectl get hpa bhumika-backend-deployment',
                        returnStatus: true
                    ) == 0
                    if (!hpaBackendExists) {
                        bat "kubectl autoscale deployment bhumika-backend-deployment --cpu-percent=25 --min=1 --max=2"
                    } else {
                        echo "HPA bhumika-backend-deployment already exists"
                    }
                    def hpaFrontendExists = bat (
                        script: 'kubectl get hpa bhumika-frontend-deployment',
                        returnStatus: true
                    ) == 0
                    if (!hpaFrontendExists) {
                        bat "kubectl autoscale deployment bhumika-frontend-deployment --cpu-percent=25 --min=1 --max=2"
                    } else {
                        echo "HPA bhumika-frontend-deployment already exists"
                    }
                }
            }
        }
    }

    post {
        success {
            script {
                bat "kubectl delete pod -l app=bhumika-backend --field-selector=status.phase==Running"
                bat "kubectl delete pod -l app=bhumika-frontend --field-selector=status.phase==Running"
                bat "kubectl delete pod -l app=bhumika-backend --field-selector=status.phase!=Running"
                bat "kubectl delete pod -l app=bhumika-frontend --field-selector=status.phase!=Running"
                bat "docker image prune -f"
                bat "docker rmi ${BACKEND_IMAGE} ${FRONTEND_IMAGE}"
            }
        }
        failure {
            bat 'echo "Build failed"'
            bat "docker rmi ${BACKEND_IMAGE} ${FRONTEND_IMAGE}"
        }
    }
}
```

Figure 18: Jenkins Pipeline Script

a. Pipeline and Environment Setup:

```
pipeline {
  agent any
  environment {
    KUBECONFIG = "fix/kubeconfig"
    DOCKER_IMAGE_TAG = "${currentBuild.number}"
    BACKEND_IMAGE = "bhumika-t2i-backend:${DOCKER_IMAGE_TAG}"
    FRONTEND_IMAGE = "bhumika-t2i-frontend:${DOCKER_IMAGE_TAG}"
  }
```

- pipeline {agent any}: Specifies that this pipeline can run on any available agent.
- Environment Setup: Defines environment variables for configuration and versioning:
  - KUBECONFIG: Path to the Kubernetes configuration file.
  - DOCKER_IMAGE_TAG: Tag for Docker images.
  - BACKEND_IMAGE: Docker image name for the backend service.
  - FRONTEND_IMAGE: Docker image name for the frontend service.

b. Code Fetch

```
stage("Code Fetch") {
        steps {
            git url: "https://github.com/bhumika-16/text2scene.git", branch: "main"
        }
    }
```

In this stage, we retrieve code from a designated Git repository, specifying its URL and branch for cloning. Git enables version control, vital for tracking changes and collaborating efficiently within development teams.

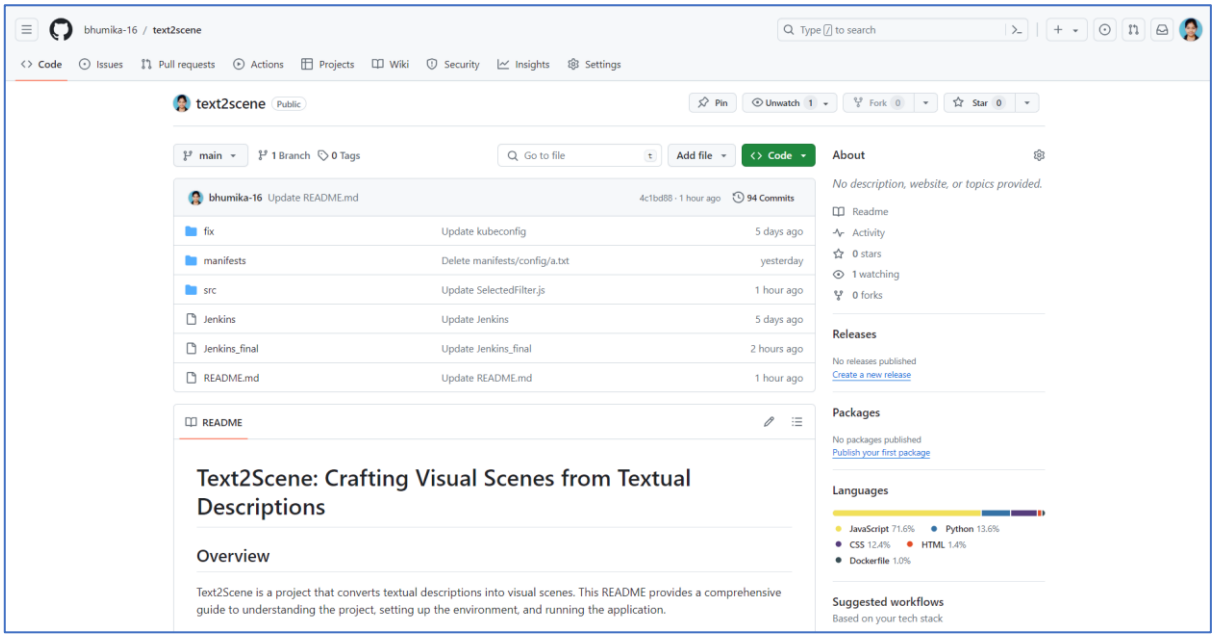
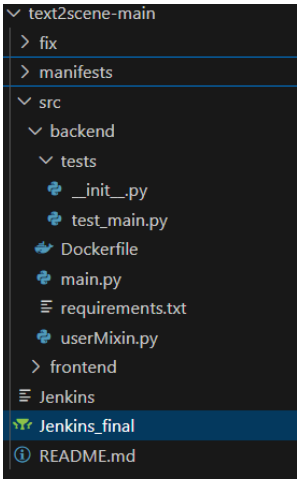


Figure 19: Snapshot of the GitHub Repository

c. Build Backend Docker Image

```
stage("Build Backend Docker Image") {
        steps {
           bat "docker build -t bhumika-t2i-backend:${DOCKER_IMAGE_TAG} src/backend"
        }
}
```

This stage builds the Docker image for the backend service.



- **bat:** Executes a Windows batch command.
- **docker build:** Builds a Docker image using the Dockerfile located in the "src/backend" directory.
- **-t:** Tags the built image with the specified tag.

Note: For Ubuntu, utilize sh instead of bat for shell commands, and prepend sudo before each Docker and Kubernetes command if elevated privileges are required.

Backend Files:

• Requirements.txt

The **requirements.txt** file contains a list of Python packages required for the project. These packages include libraries for web development (Flask), natural language processing (Stanza, spaCy), machine learning (Hugging Face Transformers, PyTorch), data manipulation (NumPy, Pandas), and web APIs (Requests). Additionally, it includes utilities for testing (Pytest), database interaction (PyMySQL), and security (Cryptography). Properly managing these dependencies ensures the project's functionality, efficiency, and security.

• Main.py

This Python script is a Flask application that serves various endpoints for text processing, sentiment analysis, user authentication, and image generation. Here's a detailed breakdown of its functionality:

1. **Importing Libraries:** The script imports necessary Python libraries such as base64, hashlib, json, datetime, logging, matplotlib, pandas, pymysql, requests, Flask, Flask-CORS, Flask-JWT-Extended, googletrans, and pyngrok. These libraries facilitate functionalities like data manipulation, HTTP requests, JWT authentication, logging, and more.
2. **Database Table Creation:** The create_users_table() function establishes a connection to a MySQL database and creates a table named users if it doesn't exist already. This table stores user credentials (email and hashed password).
3. **Text Translation:** The get_translation() function utilizes the Google Translate API (googletrans) to translate text to a specified destination language.
4. **Text Filtration:** The filtration_query() function sends a text payload to the Hugging Face Text Moderation API to assess its appropriateness.
5. **Sentiment Analysis:** The semantic_query() function sends text data to the Hugging Face GoEmotions API to analyze its sentiment and emotional content.
6. **User Authentication:** Endpoints /register and /login handle user registration and login processes respectively. These endpoints interact with the MySQL database to store and validate user credentials.

7. **JWT Token Management:** JWT (JSON Web Token) authentication is implemented using Flask JWT Extended (Flask-JWT-Extended). The application generates access tokens upon successful login, which are required to access protected routes.
8. **Image Generation:** The /text2image route generates images based on textual prompts using the Stability AI Image Generation API. It sends a POST request to the API endpoint with the prompt text and optional parameters such as output format, style preset, scale, and steps. Upon receiving a successful response (status code 200), it saves the generated image locally and encodes it to base64 for sending it as a JSON response.
9. **Route Handlers:**
   Various Flask routes are defined to handle different types of requests:
   - **/logout:** Logs out the user by clearing JWT cookies.
   - **/filtration-scores:** Calculates filtration scores for text using the filtration_query() function.
   - **/sentiment-scores:** Computes sentiment scores for text using the semantic_query() function.
   - **/translation:** Provides translation for text using the get_translation() function.
   - **/sentiment-scores-sum:** Computes sentiment scores summary using the sentiment_score() function.
   - **/text2image:** Generates images based on textual prompts using the Stability AI Image Generation API.
10. **Error Handling:** The application handles various errors gracefully and logs relevant information for debugging purposes.
11. **Running the Flask Application:** Finally, the Flask application is run on host 0.0.0.0 and port 5001 with debugging enabled.

• Dockerfile

Jenkins_final ● | main.py | Dockerfile ×

text2scene-main > src > backend > Dockerfile

```
# Use the official Python image as a base image
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /app

# Install build dependencies
RUN apt-get update && apt-get install -y \
  gcc \
  g++ \
  make \
  cmake \
  libboost-all-dev \
  libmarisa-dev

# Copy the requirements file into the container at /app
COPY requirements.txt /app/

# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt

RUN huggingface-cli login --token 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

# Copy the rest of the application code into the container at /app
COPY . /app/

# Expose port 5000 to the outside world
EXPOSE 5001

# Command to run the application
CMD ["python", "main.py"]

```

Figure 20: Dockerfile for Backend

The provided Dockerfile is used to build a Docker image for deploying a Python application. Here's the documentation for each section of the Dockerfile:

- **Base Image Selection:** The Dockerfile starts with selecting the official Python 3.9 slim image (python:3.9-slim) as the base image. This image provides a lightweight Python environment for running Python applications.
- **Setting Working Directory:** The WORKDIR instruction sets the working directory inside the container to /app. This directory will be used for storing application code and related files.
- **Installing Build Dependencies:** The Dockerfile installs build dependencies required for building certain Python packages. These dependencies include gcc, g++, make, cmake, libboost-all-dev, and libmarisa-dev.
- **Copying Requirements File:** The COPY instruction copies the requirements.txt file from the host machine into the /app directory in the container. This file lists all Python dependencies required by the application.
- **Installing Python Dependencies:** Python dependencies listed in the requirements.txt file is installed using pip install. The --no-cache-dir flag is used to avoid caching of the installed packages, ensuring that the dependencies are installed freshly.
- **Hugging Face CLI Login:** The Dockerfile logs into the Hugging Face model hub using the huggingface-cli login command. This step requires providing an authentication token to access private models and resources.
- **Copying Application Code:** The COPY instruction copies the rest of the application code from the host machine into the /app directory in the container. This includes the main application script (main.py) and any other files required for running the application.
- **Exposing Port:** The EXPOSE instruction exposes port 5001 on the container to allow external access to the application running inside the container. Port 5001 is typically used for serving web applications over HTTP.
- **Command to Run Application:** The CMD instruction specifies the command that should be executed when the container starts. In this case, it runs the Python application using the python main.py command, which starts the Flask web server defined in main.py.

This Dockerfile automates the process of setting up the Python environment, installing dependencies, and configuring the application, making it easier to deploy the application consistently across different environments.

d. Testing Backend application

```
stage("Testing Backend application") {
        steps {
           bat "docker run --rm ${BACKEND_IMAGE} pytest ."
        }
    }
```

This stage aims to execute automated tests for the backend application within a Docker container using the Pytest framework. By running tests automatically, it ensures the reliability and correctness of the backend functionality.

- The **docker run** command is used to launch a Docker container from the backend application Docker image.
- The **--rm** flag ensures that the container is removed after execution, maintaining cleanliness.
- **${BACKEND_IMAGE}** is the Docker image tag for the backend application, dynamically generated within the pipeline.
- **pytest .** specifies the command to execute within the container, where Pytest is instructed to run all test cases found in the current directory (.).

Pytest automatically discovers and executes test files by recursively searching the specified directory (.) and its subdirectories for files that match the naming pattern for test files. By default, Pytest identifies test files based on their names, recognizing files prefixed with test_ or suffixed with _test. Once test files are discovered, Pytest collects and executes the test functions defined within them. While not directly related to Pytest, the __init__.py file is essential for Python packages and modules, enabling proper package organization and import functionality. Including an __init__.py file in the directory where tests are located may be necessary for organizing tests within a package structure or enabling package-level imports if needed.

- ## Test_main.py

1. **Test get_translation Functionality:**

- Validates the accuracy of text translation across multiple languages using the get_translation function.
- Parameterized tests cover various input texts and destination languages, ensuring correct translations.

2. **Mocked API Testing for Text Filtration and Semantic Analysis:**

- Utilizes unittest.mock.patch to simulate external API responses for text filtration and semantic analysis.
- Mocked responses enable thorough testing of filtration_query and semantic_query functions without actual API calls.

3. **Test sentiment_score Function for Sentiment Analysis:**

- Evaluates the sentiment scoring mechanism based on semantic analysis results.
- Parametrized tests assess the accuracy of sentiment percentages and appropriateness scores for diverse sentiment scenarios.

These test cases ensure the correctness of the translation, text filtration, semantic analysis, and sentiment analysis functionalities implemented in the main.py module. Additionally, the use of mocking allows testing the integration with external APIs without actually making real API calls during the test execution.

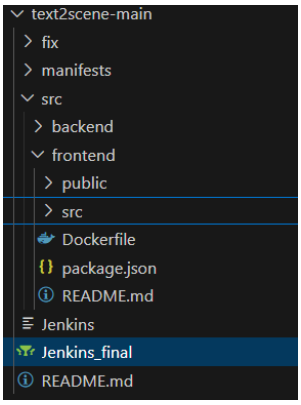
e. Build Frontend Docker Image

```
stage("Build Frontend Docker Image") {
        steps {
           bat "docker build -t bhumika-t2i-frontend:${DOCKER_IMAGE_TAG} src/frontend"
        }
    }
```

The purpose of this stage is to create a Docker image for the frontend application from the source code. This image will be tagged with a version identifier that corresponds to the current build number in Jenkins.

- **bat** indicates that the command should be run in a Windows shell (batch file).
- **docker build:** Initiates the build process for a Docker image.
- **-t bhumika-t2i-frontend:${DOCKER_IMAGE_TAG}:** Tags the created image with the name bhumika-t2i-frontend and the current build number from Jenkins (${DOCKER_IMAGE_TAG}).

- **src/frontend:** Specifies the directory containing the Dockerfile and the source code for the frontend application.



Frontend Files:

• Package.json

The **package.json** file for the image-generation project defines its metadata, dependencies, and scripts. It includes libraries for React, Redux, MUI, and testing tools like Jest. Key scripts facilitate starting the development server, building the project, and running tests. It also configures ESLint and specifies browser support for production and development environments.

• Src Directory

The src folder for the Text2Scene frontend application is structured to organize the codebase efficiently and facilitate maintainability. Below is a detailed breakdown of its contents:

1. **tests**
- Contains unit and integration tests for various components and functionalities of the application.
- Ensures code reliability and functionality through rigorous testing.

2. **redux**
- **action.js:** Defines Redux action creators that dispatch actions to the store.
- **actionTypes.js:** Contains constants representing different action types, preventing typos and ensuring consistency.
- **reducer.js:** Implements Redux reducers to update the application state based on dispatched actions.
- **store.js:** Configures and exports the Redux store, integrating middleware if necessary.

3. **sidebarlayout**
- **index.js:** The main file for the sidebar layout component, managing the sidebar's structure and behavior.

4. **CSS and JS Components**
- **App.js and App.css:** The root component and its associated styles, setting up the primary structure and routing.
- **Carousel.js and Carousel.css:** Manages image carousels, allowing users to browse through generated images.
- **Content.js and content.css:** Handles the display and styling of main content areas within the application.
- **ContentAnalysis.js:** Analyzes and displays content-related data.
- **FilterPanel.js and FilterPanel.css:** Provides filtering options for image generation based on user input.
- **FiltrationScoresGraph.js:** Visualizes filtration scores through graphs.
- **gallery.js:** Manages the display of generated images in a gallery format.
- **Header.js and Header.css:** Implements the header component, including navigation and branding elements.
- **Homepage.js and Homepage.css:** The main landing page component of the application.

- **ImageGenerator.js:** Core functionality for generating images from textual descriptions.
- **LoginRegisterForm.js:** Components for user authentication, including login and registration forms.
- **Prompt.js and Prompt.css:** Manages user prompts for text input used in image generation.
- **ReportWebVital.js:** Monitors and reports web vital metrics for performance tracking.
- **SemanticAnalysisGraph.js and SemanticAnalysisGraph.css:** Displays semantic analysis results through graphs.
- **SentimentData.js:** Manages sentiment data derived from textual input.
- **Translate.js:** Handles multilingual text translation functionalities.
- **setupTests.js:** Configures the testing environment, ensuring proper test setup and execution.

This structure ensures a clear separation of concerns, making the codebase modular, scalable, and easy to navigate for developers working on the Text2Scene project.

• Dockerfile

text2scene-main > src > frontend > Dockerfile

```
# Use official Node.js image as base
FROM node:21-alpine

# Set the working directory in the container
WORKDIR /app

# Copy package.json and package-lock.json to the working directory
COPY package.json ./

# Install dependencies
RUN npm install --legacy-peer-deps

# Copy the rest of the application code to the working directory
COPY . .

# Set environment variable for CI
ENV CI=true

# Build the React application (not necessary if using 'npm start')
# RUN npm run build

# Expose port 3000
EXPOSE 3000

# Command to start serving the React application
CMD ["npm", "start"]

```

Figure 21: Dockerfile for Frontend

This Dockerfile sets up a Docker container for the Text2Scene frontend application, ensuring a consistent environment for development, testing, and deployment. Below is a step-by-step explanation of each instruction in the Dockerfile:

- **Base Image** - Uses the official Node.js image with version 21 and the Alpine Linux distribution for a lightweight base.
- **Set Working Directory** - Sets the working directory inside the container to /app. All subsequent commands will be run from this directory.
- **Copy package.json and package-lock.json** - Copies the package.json file to the working directory. This file contains metadata and dependencies for the project.
- **Install Dependencies** - Installs the project dependencies specified in package.json. The --legacy-peer-deps flag is used to resolve any peer dependency conflicts.
- **Copy Application Code** - Copies the rest of the application code from the host to the working directory in the container.

- **Set Environment Variable -** Sets the environment variable CI to true, which can be used to indicate that the build is running in a Continuous Integration environment.
- **Expose Port -** Exposes port 3000 on the container, which is the default port for serving the React application.
- **Start the Application -** Specifies the command to start the React application when the container runs. npm start launches the development server.

f. Testing Frontend application

```
stage("Testing Frontend application") {
        steps {
          bat "docker run --rm ${FRONTEND_IMAGE} npm run test:ci"
        }
    }
```

This stage runs automated tests for the frontend application using a Docker container.

- **docker run --rm**: This command runs a Docker container and automatically removes the container once the command has finished executing, ensuring no leftover containers.
- **${FRONTEND_IMAGE}**: This variable represents the Docker image for the frontend application, tagged with the current build number. It ensures the tests are run against the latest version of the code.
- **npm run test:ci**: This command runs the test suite in continuous integration mode.
- **test:ci:** This script is defined in the package.json file of the frontend application. It typically includes configurations to run tests in a CI environment, such as running tests without watch mode and ensuring they can be run in a headless browser if necessary.

Ensure Docker is installed and running on the Jenkins agent. The frontend Docker image should be built and tagged with the current build number in a previous stage.

The frontend project uses **Jest** for running tests. Jest is configured to automatically recognize and run test files based on certain naming conventions and directory structures. Jest looks for files with any of the following naming patterns:

- Files with **.test.js** or **.test.jsx** suffix.
- Files with **.spec.js** or **.spec.jsx** suffix.

In this project, the test files follow the .test.js naming convention.

- **FilterPanel.test.js** verifies that the FilterPanel component renders the text "Filter" correctly. It uses React Testing Library to render the component and check for the presence of the text.
- **Home.test.js** ensures that the Home component renders correctly by checking for the presence of the text "Why use us?". It also uses React Testing Library for rendering and assertion.

In the package.json, there are scripts defined to run the tests:

- **"test":** This script runs the tests using Jest. The --transformIgnorePatterns option is used to include specific modules during the transformation process.
- **"test:ci":** This script runs the tests in continuous integration mode. It sets the CI=true environment variable using cross-env, ensuring the tests run in a CI environment without interactive prompts.

Jest automatically discovers and runs tests based on the naming conventions and directory structure. The test files FilterPanel.test.js and Home.test.js located in the src/__tests__ directory follows these conventions. The tests are executed as part of the CI/CD pipeline using a Docker container, ensuring consistent and isolated test environments.

g. Push Backend Docker Images

```
stage("Push Backend Docker Images") {
        steps {

withCredentials([usernamePassword(credentialsId:"DockerHubCred",passwordVariable:"dockerp
assword",usernameVariable:"dockerusername")]){
                bat "docker login -u ${env.dockerusername} -p ${env.dockerpassword}"
                                        bat "docker tag bhumika-t2i-backend:${DOCKER_IMAGE_TAG}
${env.dockerusername}/bhumika-t2i-backend"
                bat "docker push ${env.dockerusername}/bhumika-t2i-backend"
            }
        }
    }
    stage("Push Frontend Docker Images") {
        steps {

withCredentials([usernamePassword(credentialsId:"DockerHubCred",passwordVariable:"dockerp
assword",usernameVariable:"dockerusername")]){
                bat "docker login -u ${env.dockerusername} -p ${env.dockerpassword}"
                                    bat "docker tag bhumika-t2i-frontend:${DOCKER_IMAGE_TAG}
${env.dockerusername}/bhumika-t2i-frontend"
                bat "docker push ${env.dockerusername}/bhumika-t2i-frontend"
            }
        }
    }
```

To push Docker images for both the backend and frontend components, follow these steps:

- **Authenticate to Docker Hub:** The withCredentials block retrieves Docker Hub credentials stored in Jenkins. dockerusername and dockerpassword environment variables are set with these credentials.
- **Login to Docker Hub:** The bat "docker login -u ${env.dockerusername} -p ${env.dockerpassword}" command logs into Docker Hub using the retrieved credentials.
- **Tag the Docker Image:**bat "docker tag bhumika-t2i-frontend:${DOCKER_IMAGE_TAG} ${env.dockerusername}/bhumika-t2i-frontend" tags the Docker image with the appropriate tag.
- **Push the Docker Image:** bat "docker push ${env.dockerusername}/bhumika-t2i-frontend" pushes the tagged Docker image to the Docker registry.

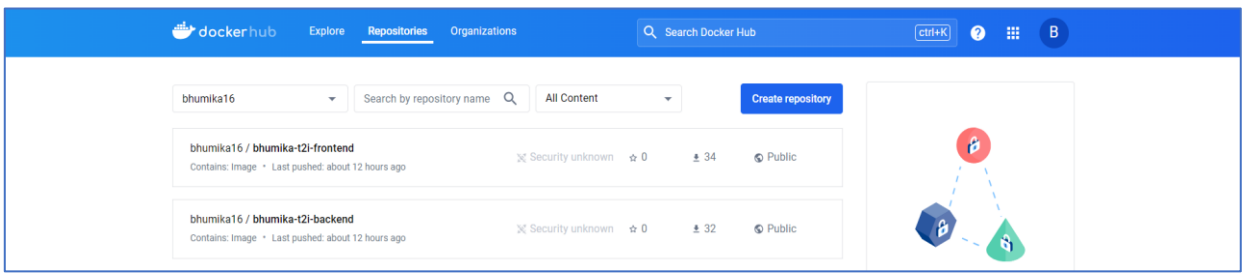


Figure 22: Snapshot of Docker Images Created

h. Deploy ELK Stack

```
stage("Deploy ELK") {
        steps {
           bat "kubectl apply -f manifests/deploy/fluentbit-deploy.yaml"
           bat "kubectl apply -f manifests/deploy/logtash-deploy.yaml"
           bat "kubectl apply -f manifests/deploy/elasticsearch-deploy.yaml"
           bat "kubectl apply -f manifests/deploy/kibana-deploy.yaml"
           bat "kubectl apply -f manifests/svc/elasticsearch-svc.yaml"
           bat "kubectl apply -f manifests/svc/logtash-svc.yaml"
           bat "kubectl apply -f manifests/svc/kibana-svc.yaml"
        }
    }
```

This stage orchestrates the deployment of the ELK (Elasticsearch, Logstash, Kibana) stack, a comprehensive solution for log management and analysis. It sequentially applies Kubernetes manifest files to provision Fluent Bit, Logstash, Elasticsearch, and Kibana components, along with associated services. These components collectively facilitate log collection, processing, storage, and visualization, empowering users to efficiently monitor and analyze system logs within the Kubernetes cluster.

Deployment of Components:

- **fluentbit-deploy.yaml:** Deploys Fluent Bit, a lightweight log collector, to gather logs from various sources.
- **logstash-deploy.yaml:** Deploys Logstash, which processes and transforms log data before sending it to Elasticsearch.
- **elasticsearch-deploy.yaml:** Deploys Elasticsearch, a distributed search and analytics engine, to store and index log data.
- **kibana-deploy.yaml:** Deploys Kibana, a data visualization and exploration tool, to interactively analyze log data.

Service Deployment:

- **elasticsearch-svc.yaml, logtash-svc.yaml, kibana-svc.yaml:** Deploys Kubernetes services for Elasticsearch, Logstash, and Kibana respectively, enabling communication with these components within the cluster.

Each YAML file is applied using **kubectl apply -f** command, which instructs Kubernetes to create or update resources described in the manifest files. This stage sets up the ELK stack for log management and analysis, providing a centralized platform for monitoring application logs, troubleshooting issues, and gaining insights into system behavior.

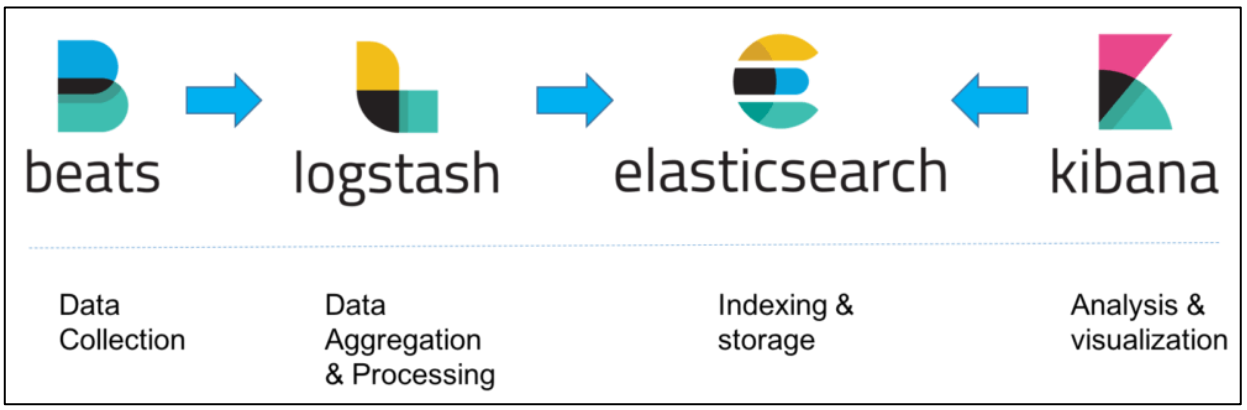


Figure 23: ELK Stack

- **fluentbit-deploy.yaml**

This YAML document defines a ConfigMap named "fluent-bit-config" containing configuration data for Fluent Bit, specifying log input paths, filters, and output destinations. Additionally, it declares a DaemonSet resource named "fluent-bit" responsible for deploying Fluent Bit pods across all nodes in the cluster. The DaemonSet's template defines a Fluent Bit container configured with resource limits, volume mounts for log directories, and a reference to the ConfigMap for configuration. This setup ensures the deployment of Fluent Bit agents tailored to collect and forward container logs to the Logstash service for further processing within the Kubernetes cluster.

- **logtash-deploy.yaml**

This YAML snippet defines a ConfigMap named "logstash-config" containing Logstash configuration data, specifying input from TCP port 5044 with JSON codec, and output to Elasticsearch. The associated Deployment resource "logstash" ensures the deployment of a Logstash container with one replica. The container uses the Logstash image from Elastic's Docker registry, mounts the ConfigMap as a volume for Logstash configuration, and exposes ports 5044 for input and 9600 for monitoring. This setup enables Logstash to ingest JSON-formatted logs from Fluent Bit and forward them to Elasticsearch for indexing and further analysis.

- **elasticsearch-deploy.yaml**

This Kubernetes YAML defines a Deployment resource named "elasticsearch" responsible for deploying a single instance of Elasticsearch. It labels the deployment as "app: elasticsearch" and specifies the Elasticsearch container using the official Docker image from Elastic's registry. The container exposes port 9200 for communication. Additionally, it sets the environment variable "discovery.type" to "single-node," configuring Elasticsearch to operate in a single-node mode suitable for development or small-scale testing environments.

- **kibana-deploy.yaml**

This Kubernetes Deployment YAML defines a single-instance deployment of Kibana labeled as "app: kibana". It specifies the Kibana container using the official Docker image from Elastic's registry. The container exposes port 5601 for accessing the Kibana web interface. Additionally, it sets the environment variable "ELASTICSEARCH_URL" to "http://elasticsearch:9200", configuring Kibana to connect to Elasticsearch at the specified URL.

- **elasticsearch-svc.yaml, logtash-svc.yaml, kibana-svc.yaml**

These Kubernetes Service YAML files define services for Elasticsearch, Kibana, and Logstash. Each service is configured with a selector to route traffic to pods labeled accordingly. They expose ports for communication within the Kubernetes cluster, with Elasticsearch using port 9200, Kibana using port 5601, and Logstash using port 5044. Additionally, they are of type LoadBalancer, enabling external access to these services from outside the Kubernetes cluster.

i. Kubernetes Deployment

```
stage("Deploy Kubernetes") {
        steps {

withCredentials([usernamePassword(credentialsId:"DockerHubCred",passwordVariable:"dockerp
assword",usernameVariable:"dockerusername")]){
                bat "kubectl apply -f manifests/deploy/mysql-deploy.yaml"
                bat "kubectl apply -f manifests/deploy/nginx-deploy.yaml"
                bat "kubectl apply -f manifests/deploy/backend-deploy.yaml"
                bat "kubectl apply -f manifests/deploy/frontend-deploy.yaml"
            }
        }
    }
```

This Jenkins pipeline stage automates the deployment of Kubernetes resources. It utilizes Kubernetes manifests to deploy MySQL, NGINX, backend, and frontend services. The withCredentials block securely injects Docker Hub credentials to authenticate image pulling during deployment. Each kubectl apply command applies the respective deployment YAML files, initiating the deployment process for each service defined in the manifests.

- **mysql-deploy.yaml**

This YAML configuration encompasses both the service and deployment specifications for MySQL within a Kubernetes environment. The service definition, named **mysql-t2i**, ensures accessibility to the MySQL database by exposing port 3306, the default MySQL port. This service is associated with pods labeled as **app:mysql,** facilitating traffic routing to the corresponding MySQL deployment.

Meanwhile, the deployment definition, identified as mysql, orchestrates the MySQL database instance. It specifies a single replica, ensuring a singular instance of MySQL within the cluster. The deployment's pod template includes a container named mysql, utilizing the latest MySQL image available. This container is configured with an environment variable, **MYSQL_ROOT_PASSWORD**, set to "root," enabling password authentication for the MySQL root user. Additionally, port 3306 is exposed within the container to allow communication with the MySQL database. This deployment ensures the availability and functionality of the MySQL service within the Kubernetes ecosystem.

- **nginx-deploy.yaml**

This YAML configuration defines a Kubernetes Deployment named **nginx-deployment**, specifying the deployment of two replicas of the NGINX web server. The deployment utilizes a selector to match pods labeled as **app: nginx**, ensuring proper pod assignment. Within the pod template, a container named nginx is defined, utilizing the latest NGINX Docker image. Port 80 is exposed within the container, allowing HTTP traffic to be handled by the NGINX server. This configuration ensures high availability and scalability of the NGINX service within the Kubernetes cluster.

- **backend-deploy.yaml**

The provided YAML snippet describes a Kubernetes Deployment named **bhumika-backend-deployment**, designed to manage one replica of the backend application labeled as **app: bhumika-backend**. The deployment specifies a pod template with a container named **bhumika-backend**, utilizing the latest image **bhumika16/bhumika-t2i-backend:latest**. Port 5001 is exposed within the container to handle incoming traffic. This deployment ensures the availability and operation of the backend application within the Kubernetes cluster.

- **frontend-deploy.yaml**

The provided YAML snippet outlines a Kubernetes Deployment named **bhumika-frontend-deployment**, responsible for managing a single replica of the frontend application labeled as **app: bhumika-frontend**. The deployment utilizes a pod template with a container named **bhumika-frontend**, using the latest image **bhumika16/bhumika-t2i-frontend:latest**. Port 3000 is exposed within the container to handle incoming traffic. This deployment ensures the availability and operation of the frontend application within the Kubernetes cluster.

j. Ingress in Kubernetes

```
stage("Set Ingress rules for Frontend and Backend") {
        steps {
           bat "kubectl apply -f manifests/ingress/ingress-backend.yaml"
           bat "kubectl apply -f manifests/ingress/ingress-frontend.yaml"
        }
    }
```

This pipeline stage applies Kubernetes Ingress rules for both the frontend and backend services. The provided YAML manifests (**ingress-backend.yaml** and **ingress-frontend.yaml**) define the routing configuration for incoming traffic to the corresponding services. Once applied, these rules enable external access to the frontend and backend applications deployed within the Kubernetes cluster.

In Kubernetes, Ingress is an API object that manages external access to services running in a cluster. It provides HTTP and HTTPS routing to services based on defined rules, allowing external traffic to reach the appropriate backend services. Ingress acts as a layer of abstraction between the internet and the services within the cluster, enabling features such as load balancing, SSL termination, and URL-based routing. Importing Ingress configurations into Kubernetes is necessary to define how external traffic should be routed to specific services within the cluster. By applying Ingress rules via YAML manifests, administrators can control how incoming requests are directed to backend services, manage domain-based routing, and configure other aspects of traffic management and security.

- **ingress-backend.yaml**

Creating an Ingress in Kubernetes involves defining routing rules to direct incoming traffic to specific services within the cluster. In this configuration, an Ingress resource named "**bhumika-backend-ingress**" is configured to route traffic with the host "**localhost**" to the service named "**bhumika-backend-svc**" on port 80. The annotation "**kubernetes.io/ingress.class: nginx**" specifies that this Ingress resource should be handled by the NGINX Ingress controller.

- **ingress-frontend.yaml**

This Kubernetes Ingress resource named "**bhumika-frontend-ingress**" specifies routing rules for directing incoming traffic to the frontend service. Traffic with the host "**demo.text2scene.ai**" is directed to the "**bhumika-frontend-svc**" service on port 3000. The annotation "**ingressClassName: nginx**" indicates that NGINX is the controller managing this Ingress. Additionally, the status section shows the load balancer's hostname as "**localhost**".

k. Deploy Kubernetes services

```
stage("Deploy Kubernetes services") {
        steps {
           bat "kubectl apply -f manifests/svc/backend-service.yaml"
           bat "kubectl apply -f manifests/svc/frontend-service.yaml"
           bat "kubectl apply -f manifests/svc/nginx-svc.yaml"
        }
    }
```

This YAML configuration defines Kubernetes services for the backend, frontend, and NGINX components. Each service exposes the corresponding pods to allow internal communication within the cluster. The backend service listens on port 5001, the frontend service on port 3000, and the NGINX service on port 80.

- **backend-service.yaml**

This Kubernetes service configuration defines a LoadBalancer type service named "**bhumika-backend-svc**" with labels indicating it as a backend service. It exposes port 80 externally and directs traffic to port 5001 of pods labeled as "**bhumika-backend**". This service is designed to facilitate external access to the backend application deployed within the Kubernetes cluster.

- **frontend-service.yaml**

This Kubernetes service configuration defines a LoadBalancer type service named "**bhumika-frontend-svc**". It selects pods labeled as "**bhumika-frontend**" and exposes port 3000 externally. Requests to this port are directed to port 3000 of the selected pods, facilitating external access to the frontend application deployed within the Kubernetes cluster.

- **nginx-svc.yaml**

This Kubernetes service configuration defines a NodePort type service named "**nginx-service**". It selects pods labeled as "**nginx**" and exposes port 80. Additionally, it specifies a nodePort of 30080, allowing external access to port 80 on the nodes within the cluster.

l. Setting up Horizontal Pod Autoscaling (HPA)

In this stage, autoscaling for both the backend and frontend deployments is set up. It first checks if the Horizontal Pod Autoscaler (HPA) for each deployment already exists. If not, it creates an HPA with CPU utilization target set to 25%, a minimum of 1 pod, and a maximum of 2 pods. If the HPA already exists, it prints a message indicating so.

This ensures that the deployments are automatically scaled based on CPU usage, optimizing resource utilization and performance. Setting up Horizontal Pod Autoscaling (HPA) is crucial for ensuring optimal resource utilization and maintaining application performance under varying workloads. By automatically adjusting the number of running pods based on CPU utilization, HPA helps to efficiently scale the application up or down, ensuring responsiveness and cost-effectiveness.

```
stage("Setting up autoscaling") {
        steps {
          script {
            def hpaBackendExists = bat (
              script: 'kubectl get hpa bhumika-backend-deployment',
              returnStatus: true
            ) == 0
            if (!hpaBackendExists) {
              bat "kubectl autoscale deployment bhumika-backend-deployment --cpu-percent=25 --
min=1 --max=2"
            } else {
              echo "HPA bhumika-backend-deployment already exists"
            }
            def hpaFrontendExists = bat (
              script: 'kubectl get hpa bhumika-frontend-deployment',
              returnStatus: true
            ) == 0
            if (!hpaFrontendExists) {
               bat "kubectl autoscale deployment bhumika-frontend-deployment --cpu-percent=25 -
-min=1 --max=2"
            } else {
              echo "HPA bhumika-frontend-deployment already exists"
            }
          }
        }
    }
```

m. Post-Build Cleanup and Recovery

```
post {
      success {
        script {
          bat "kubectl delete pod -l app=bhumika-backend --field-selector=status.phase==Running"
          bat "kubectl delete pod -l app=bhumika-frontend --field-selector=status.phase==Running"
          bat "kubectl delete pod -l app=bhumika-backend --field-selector=status.phase!=Running"
          bat "kubectl delete pod -l app=bhumika-frontend --field-selector=status.phase!=Running"
          bat "docker image prune -f"
          bat "docker rmi ${BACKEND_IMAGE} ${FRONTEND_IMAGE}"
        }
      }
      failure {
        bat 'echo "Build failed"'
        bat "docker rmi ${BACKEND_IMAGE} ${FRONTEND_IMAGE}"
      }
    }
```

This stage is a crucial part of the continuous integration/continuous deployment (CI/CD) pipeline. Its primary purpose is to ensure the cleanliness and stability of the deployment environment after the build process, as well as to handle potential failures gracefully. Here's a detailed explanation of each action within this stage:

- **Pod Deletion Based on Status:**

➢ The stage starts by deleting pods associated with the backend and frontend applications. It does so by executing **kubectl delete pod** commands with appropriate label selectors and field selectors.

- Specifically, it first deletes pods labeled with **app=bhumika-backend** and having a status of "**Running**". This ensures that any actively running pods are gracefully terminated, allowing for fresh instances to be created.
- Next, it deletes pods labeled with **app=bhumika-frontend** and having a status of "**Running**", following the same rationale as the previous step.
- Afterward, it deletes any remaining pods labeled with **app=bhumika-backend** or **app=bhumika-frontend** but not in the "**Running**" state. This cleanup step removes any lingering or stale pods that might have failed to terminate properly.

- ### Docker Image Cleanup:

Following the pod deletion steps, the stage executes a Docker command (**docker image prune -f**) to remove any unused images from the local Docker environment. This helps reclaim disk space and keeps the Docker image registry tidy.

- ### Handling Build Failure:

In case of a build failure, the stage executes specific actions within the failure block. It first echoes a message indicating the failure (**echo "Build failed"**). Then, it proceeds to remove the Docker images associated with the backend and frontend applications (**docker rmi ${BACKEND_IMAGE} ${FRONTEND_IMAGE}**). This cleanup ensures that any partially built or failed images are removed to prevent clutter and potential issues in subsequent builds.

Overall, the "Post-Build Cleanup and Recovery" stage plays a critical role in maintaining the stability, reliability, and efficiency of the CI/CD pipeline by cleaning up resources, handling failures, and preparing the environment for subsequent build and deployment processes.

vi. Building & Viewing the Pipeline in Jenkins

Steps to build the Jenkins Pipeline along with the screenshots for reference:

1. **Commit Code:** To commit code to GitHub and create a README file, follow these steps:
   - **Committing Code:** After making changes to your code, use the following commands:

```
git add .
git commit -m "Your commit message"
git push origin <branch_name>
```

   - **Creating README File:** Use a text editor to create a file named README.md. Add relevant project information, usage guidelines, or documentation. Commit the README file using the same Git commands mentioned above.

2. **Creating a Jenkins pipeline:** It involves defining a continuous integration and continuous deployment (CI/CD) workflow as code. Follow these steps:
   - **Open Jenkins:** Access your Jenkins instance through the web interface.
   - **Create a New Pipeline:** Navigate to Jenkins and click on "New Item." Choose "Pipeline" as the project type.

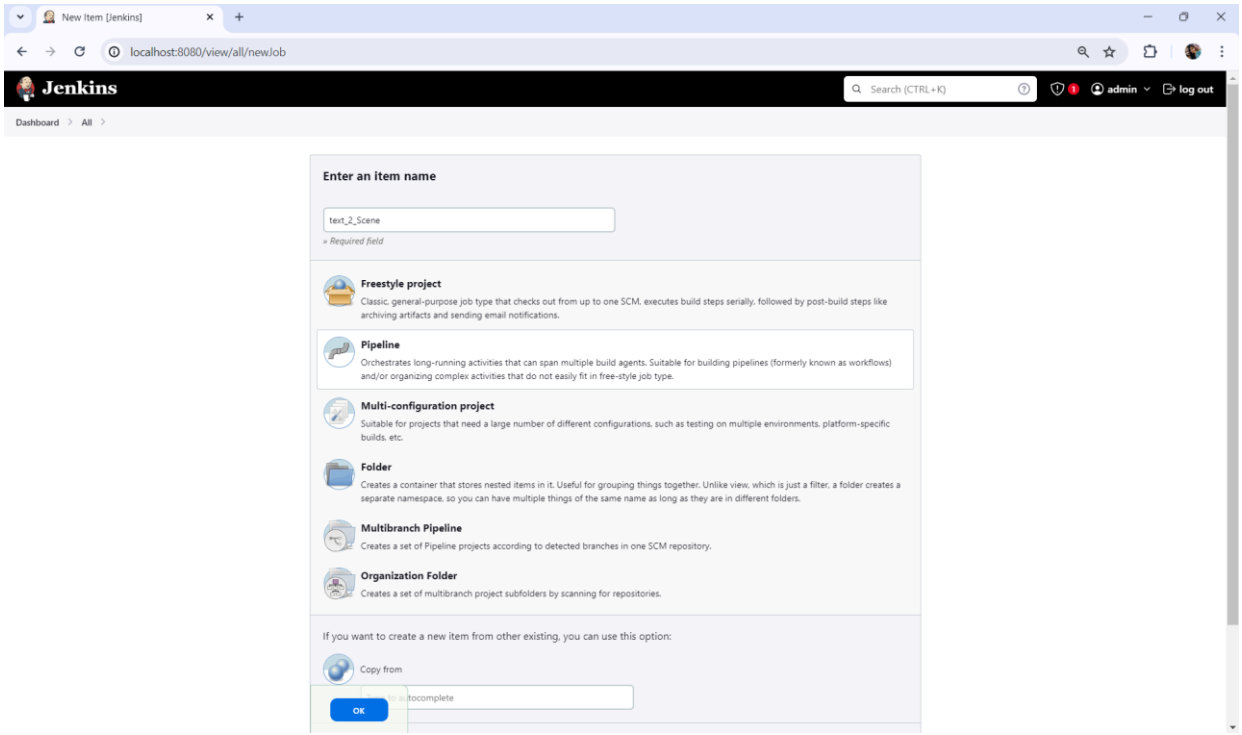


Figure 24: Creating a new Pipeline

- **Configure Pipeline:** In the pipeline configuration, specify details such as the pipeline name, choose the pipeline script from SCM (like Git), and define the repository URL and branch.

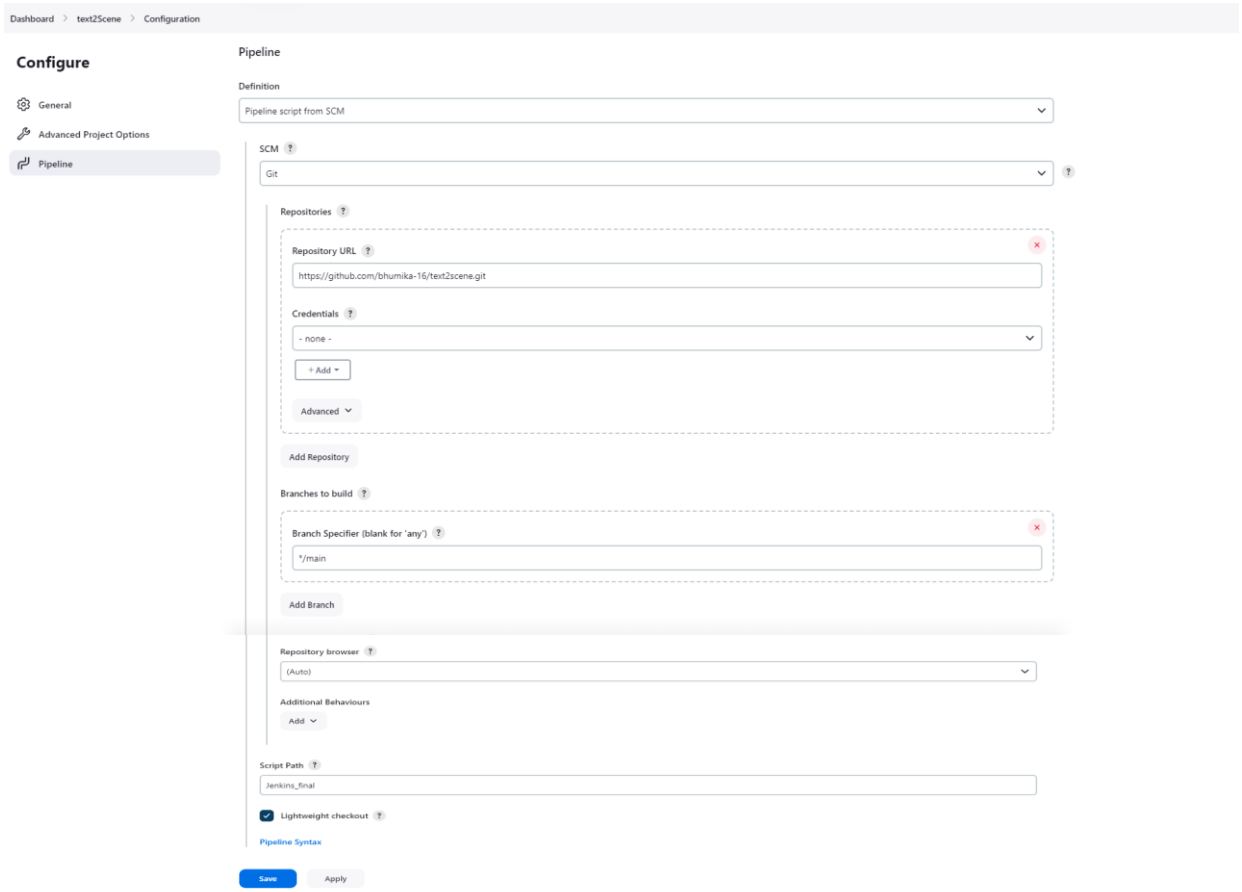


Figure 25: Configure Pipeline script from SCM

- **Write Pipeline Script:** In the pipeline script, use the Declarative Pipeline or Scripted Pipeline syntax to define stages, steps, and actions. This script outlines the entire CI/CD process.

3. **Run and view the Jenkins pipeline:**
   - **Save and Run:** Save the pipeline configuration, and Jenkins will automatically detect changes in the repository. Trigger the pipeline manually or set up webhooks for automatic builds upon code changes.
   - **Monitor and Analyze:** Jenkins will execute the defined stages, running build, test, deploy, and other tasks as per the pipeline script. Monitor the pipeline's progress through the Jenkins dashboard and review logs for any issues.

Creating a Jenkins pipeline allows for the automation of software development processes, ensuring consistent and efficient delivery of applications.

Figure 26: Jenkins Pipeline Successful

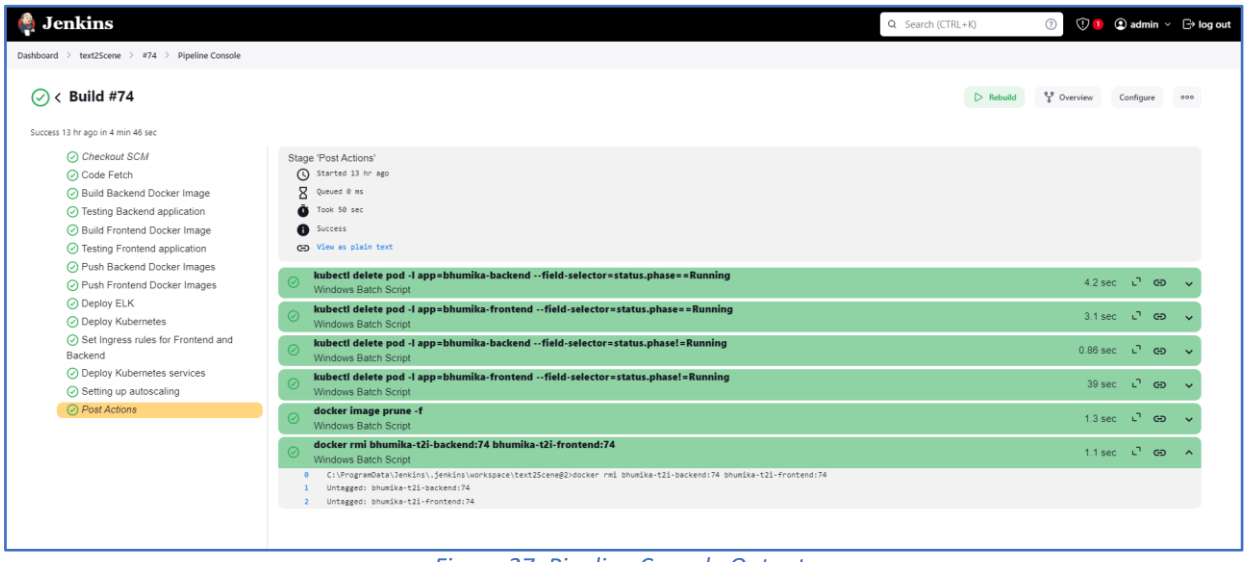


Figure 27: Pipeline Console Output

11. Running the Application

After the Jenkins pipeline has successfully completed, you can run the application locally by following these steps:

- Start by opening your web browser.
- In the address bar, type localhost:3000 and hit Enter.
- This action will open the login/register page of the application.

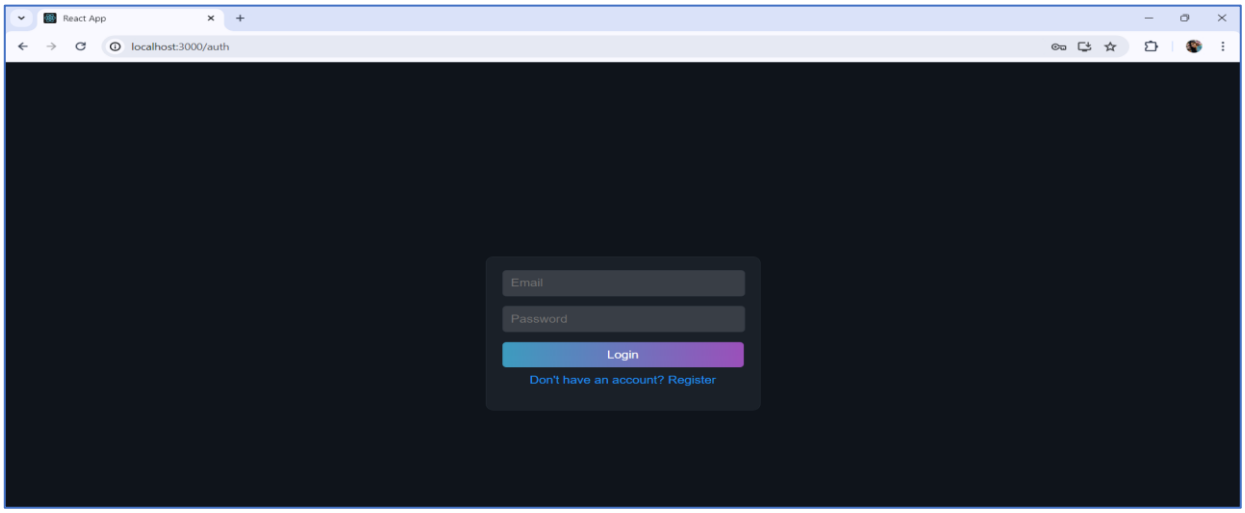


Figure 28: Register/Login Page

- Proceed by entering your email address and password into the respective fields.
- Once you've entered your credentials, click on the login button.
- If the provided credentials are correct, you will be logged in and directed to the main page or dashboard of the application.

Following these steps will enable you to access the login/register page of the application locally, where you can log in using your credentials.

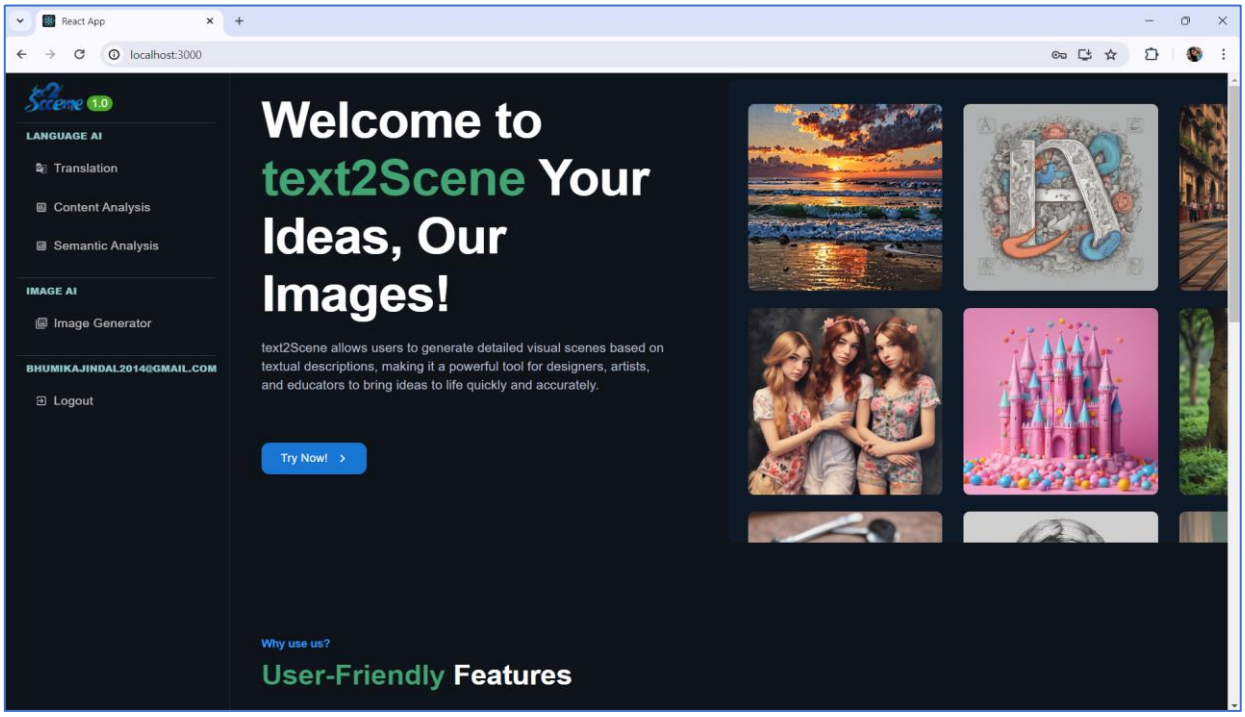


Figure 29: Homepage of the Application

The homepage of text2Scene welcomes users with a sleek and inviting design, featuring the AI-powered logo and a user-friendly interface. Users are introduced to the tool's capabilities, highlighting its ability to generate detailed visual scenes from textual descriptions. Prominent galleries showcase past creations, enticing users to explore the platform's creative possibilities. Clear and concise instructions guide users through the process, from entering scene descriptions to customizing outputs and generating visuals with just a few clicks.

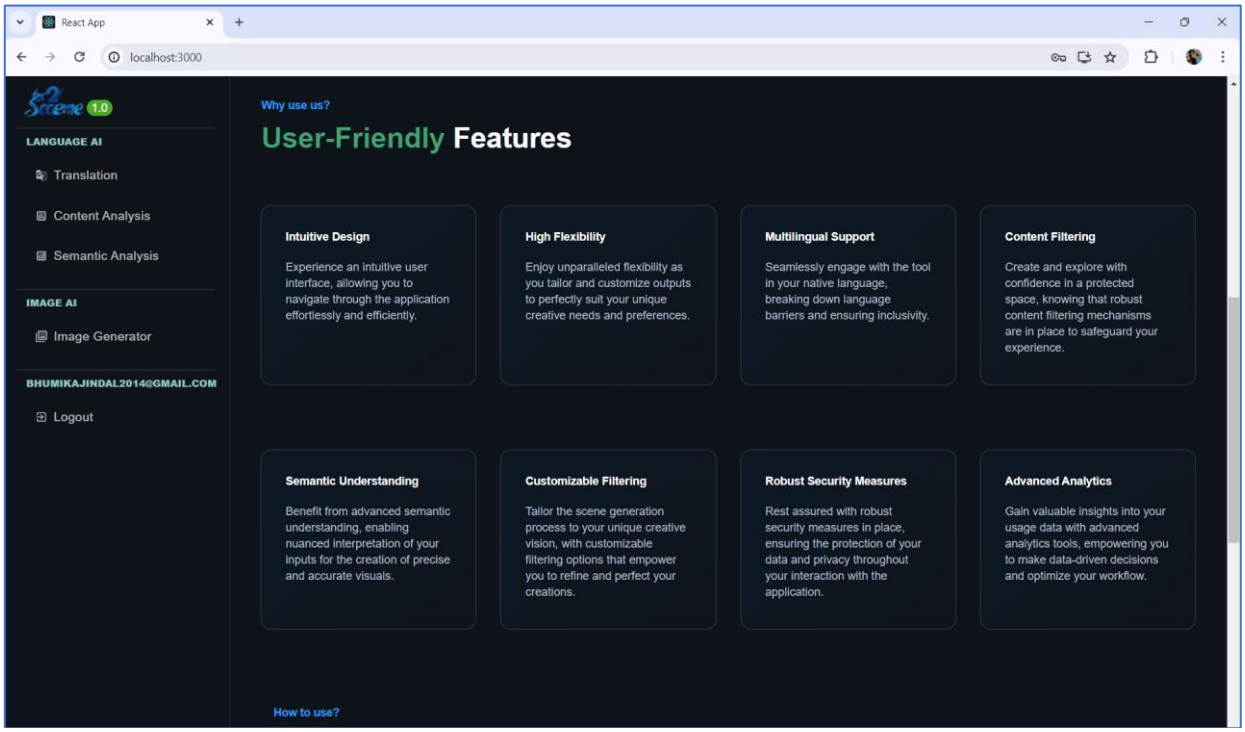


Figure 30: Features of the Application

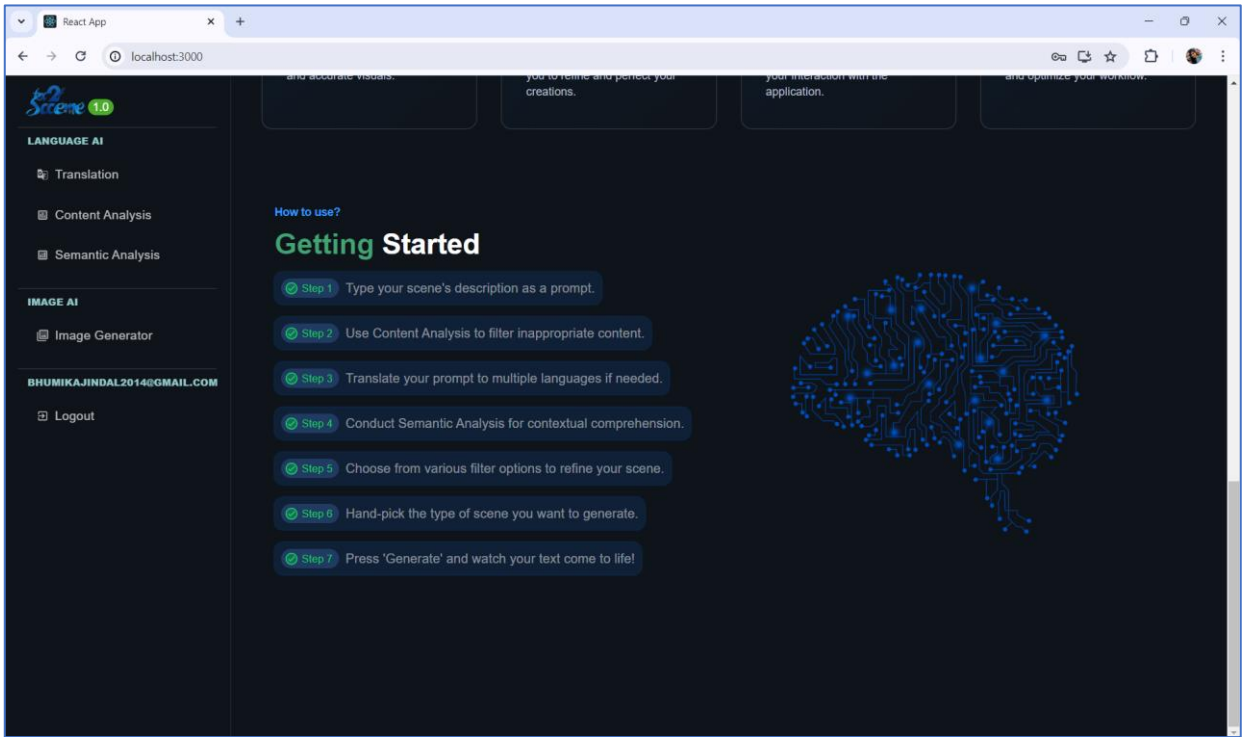


Figure 31: Usage Steps in the Homepage guides the user

The translation page of the multi-lingual text-to-image generator showcases the seamless integration of language processing capabilities. Users can input prompts in their preferred language, with translations instantly available for cross-lingual interaction. Through intuitive design and robust translation algorithms, the platform ensures accessibility and inclusivity for users worldwide.

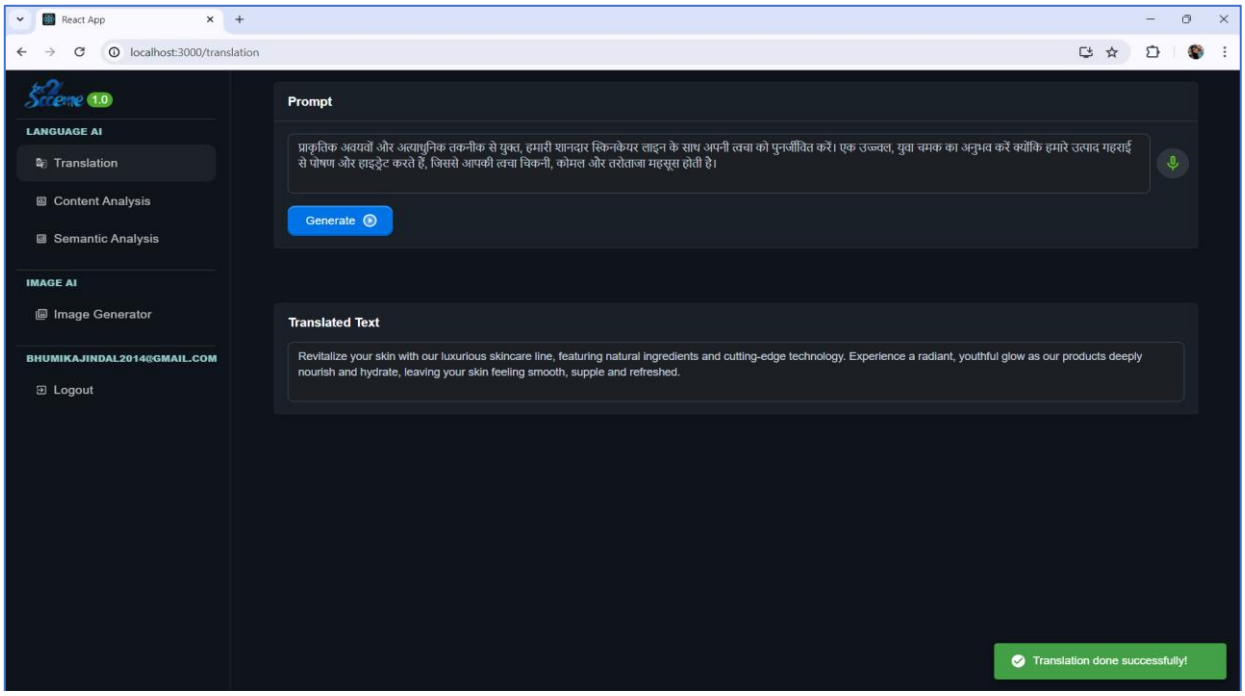


Figure 32: Translation of a Sample Prompt in Hindi

The content analysis page successfully retrieves filtration scores, providing users with valuable insights into the suitability of their prompts. By displaying scores associated with various filtration parameters, such as "H" (Hate Speech), "SH" (Sexual Harassment), and "V" (Violence), users can ensure their content aligns with desired standards. With scores ranging from 0 to 1, the platform offers transparency and control over content quality, empowering users to create responsibly and ethically.

Figure 33: Content Analysis of the Sample Prompt

The emotional tone assessment provides valuable insights into the emotional context conveyed by the prompt, facilitating the generation of visuals that accurately reflect the intended mood or sentiment. Users can leverage this analysis to ensure that the generated images align with the desired emotional tone, enhancing the overall effectiveness and impact of their visual storytelling.

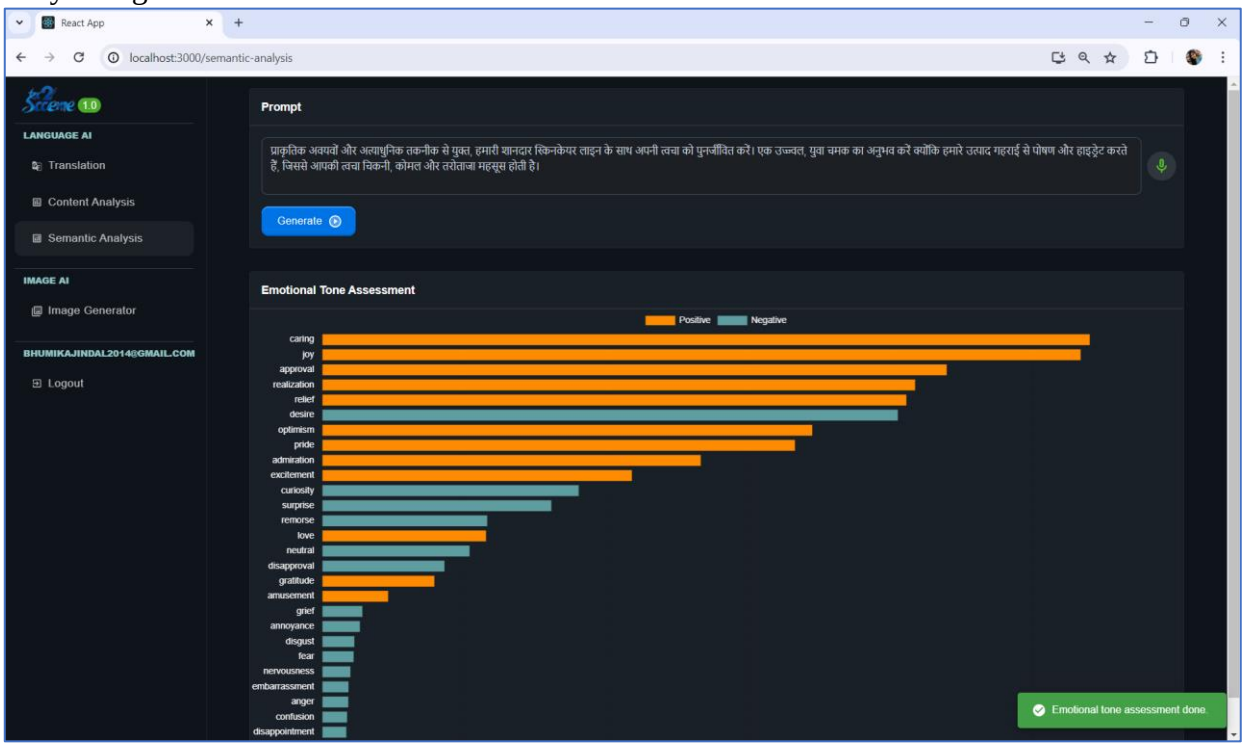


Figure 34: Semantic Analysis of the given prompt

Image Generator page facilitates the generation of images based on textual prompts provided by users. Users can input a prompt and configure various parameters like image format, art style, theme, location, and weather conditions. The system then utilizes AI technology to translate the text into visual scenes, considering semantic analysis, emotional tone assessment, and content filtration to ensure accuracy and relevance. This page transforms text descriptions into vivid images, facilitating diverse applications such as education, marketing, storytelling, and creative ideation. It empowers users to visually represent concepts, enhance comprehension, and streamline content creation across different domains.

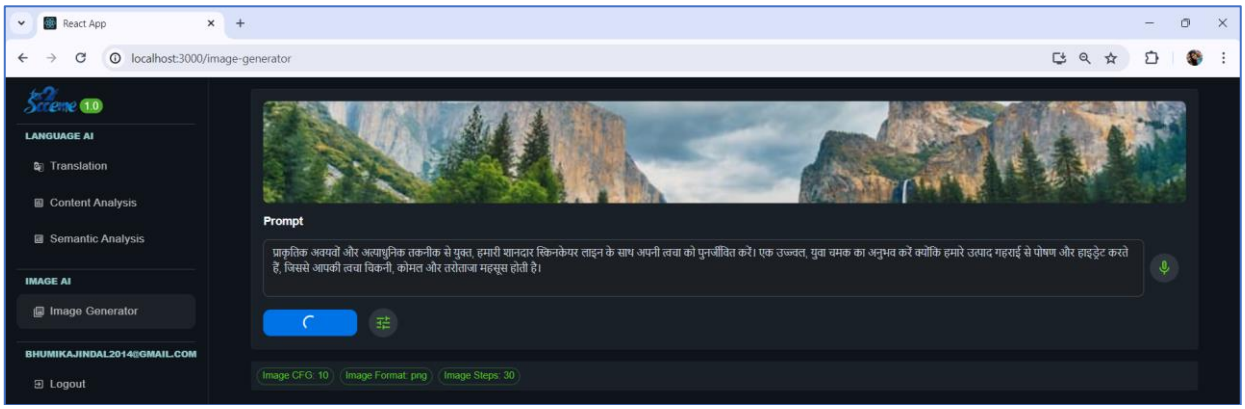


Figure 35: Image Generator Page

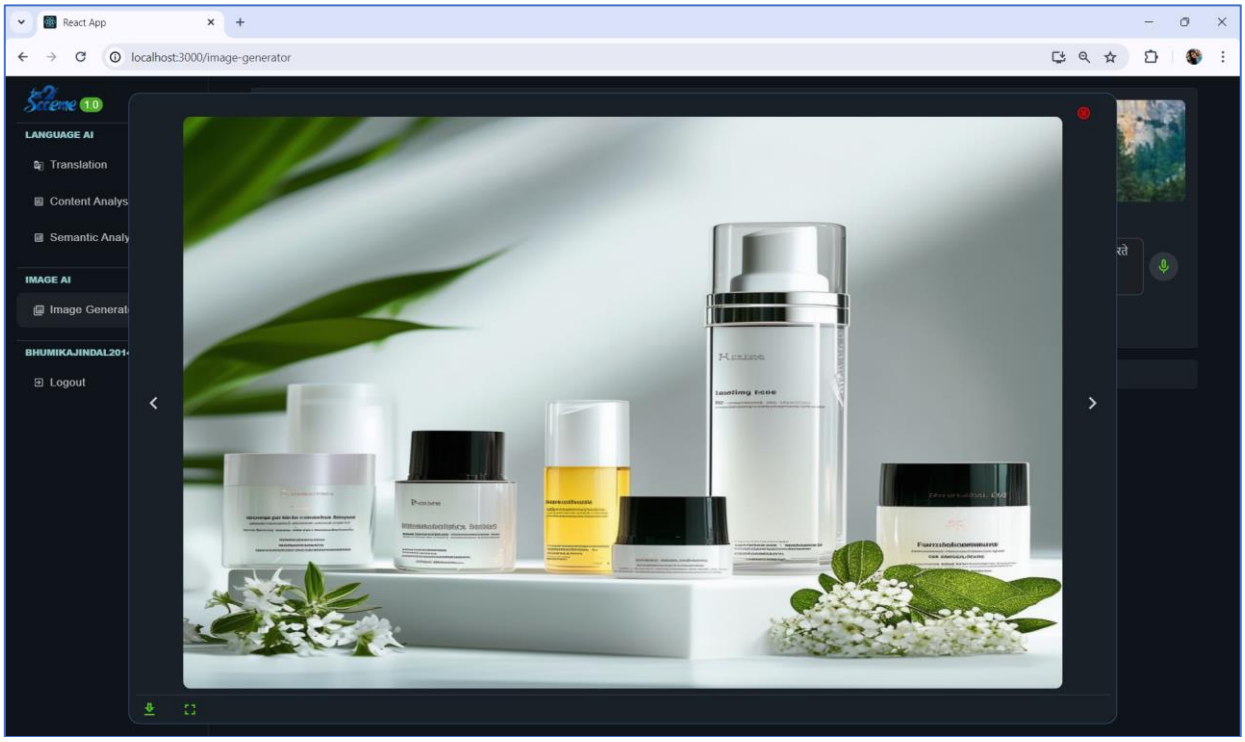


Figure 36: Image Generated for the given Skincare Advertisement

Users can engineer prompts by providing detailed descriptions, including specific elements like characters, settings, and actions. The filter panel offers granular control, allowing users to refine their prompts based on parameters such as art style, theme, location, and weather conditions. This ensures that users can clearly specify their desired scene characteristics, resulting in more accurate and tailored image generation according to their creative vision.

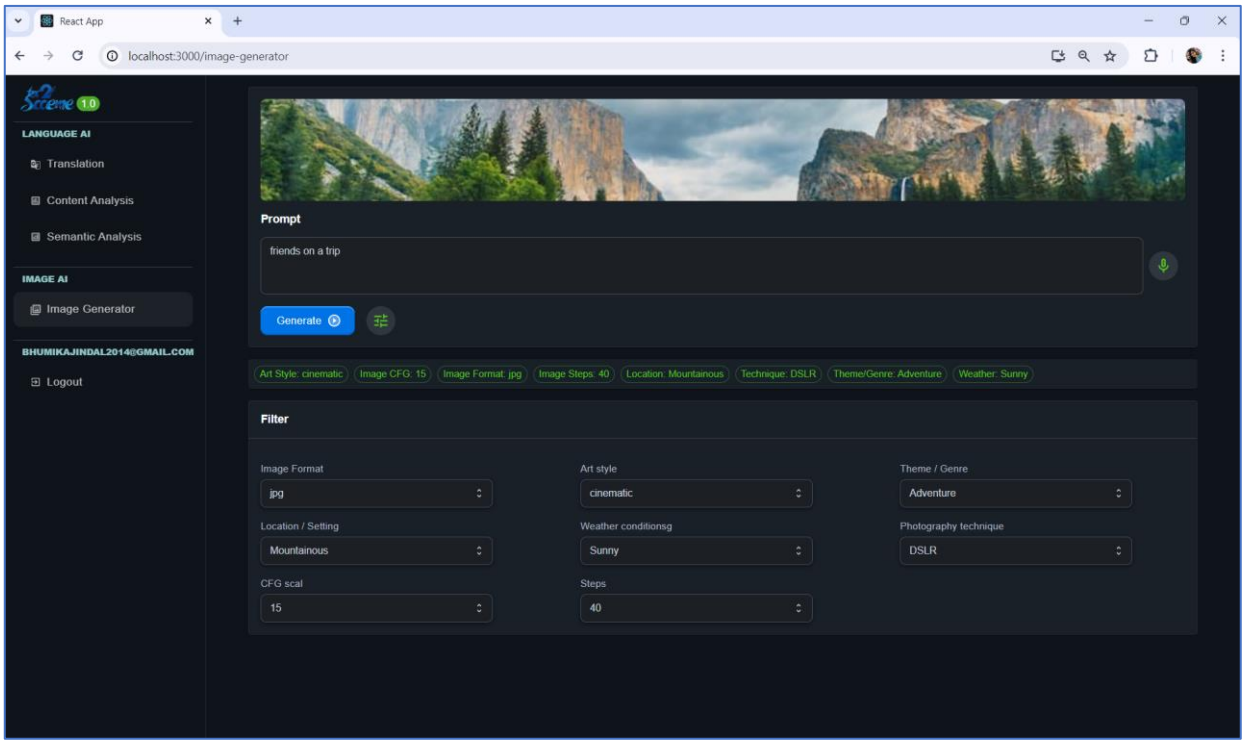


Figure 37: Prompts engineered using Filter Panel

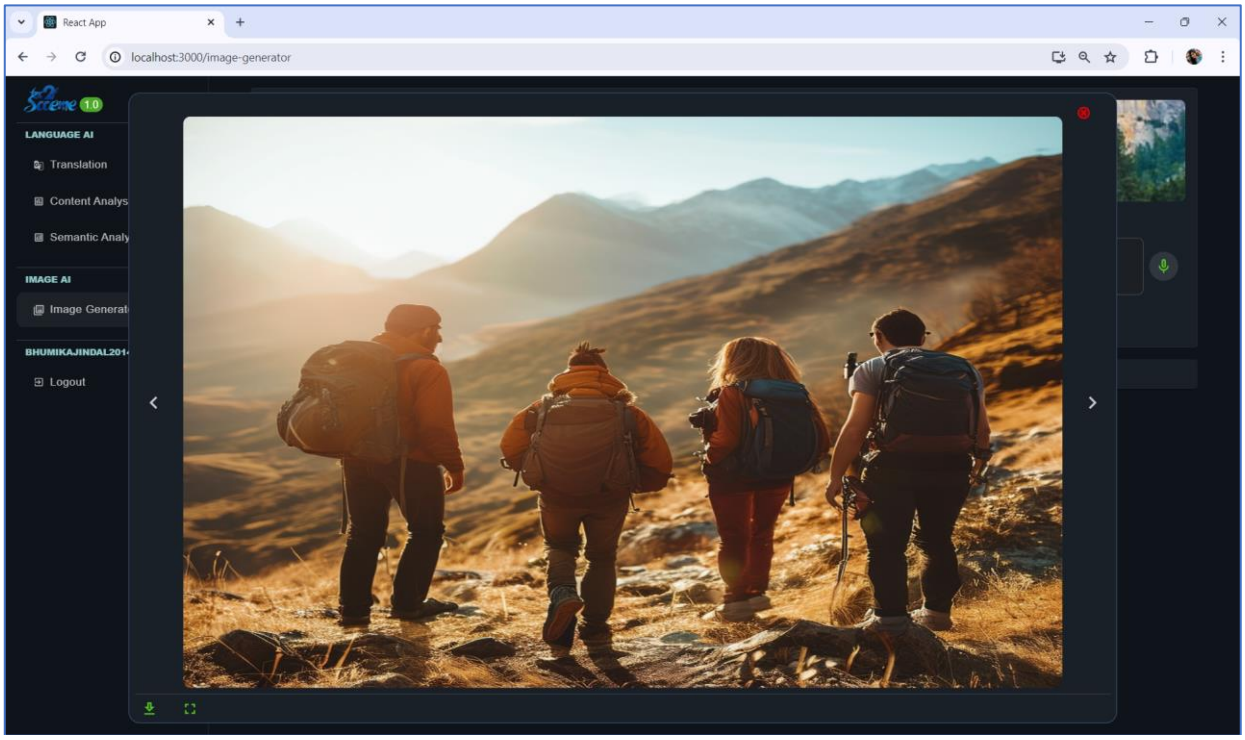


Figure 38: Image Generated for the Given Prompt

We're displaying filtration scores for a negative prompt, highlighting potential inappropriate content or language. This helps users ensure that their text aligns with appropriate communication standards and fosters respectful discourse.

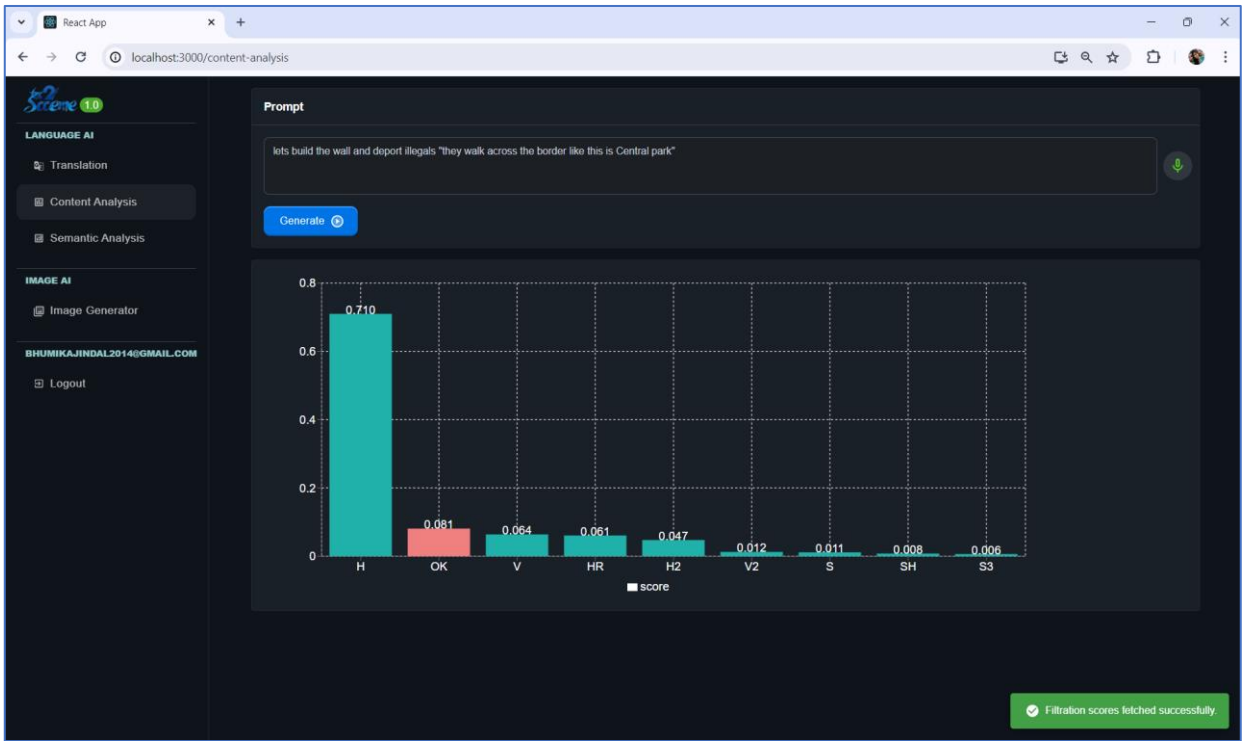


Figure 39: Inappropriate Prompt Given

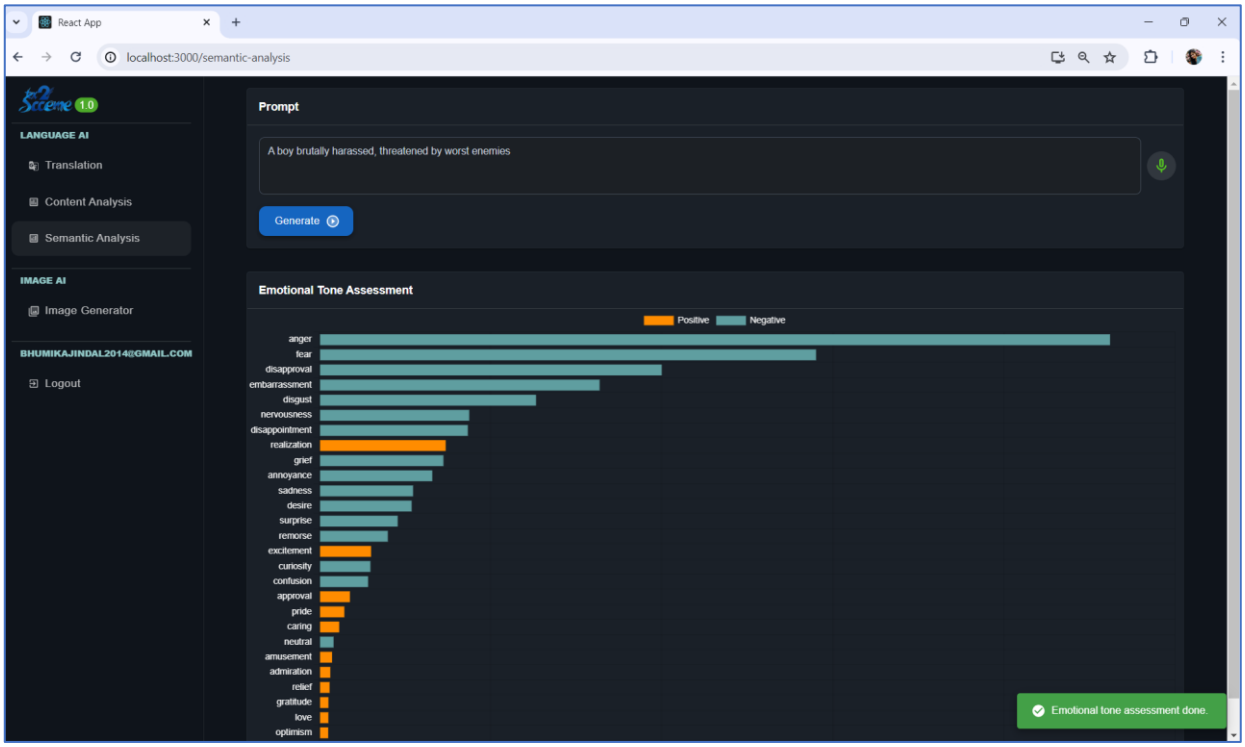


Figure 40: Semantic Analysis of Inappropriate Prompt

Docker containers have been successfully created, each serving a specific purpose within the application infrastructure. These containers are meticulously configured to encapsulate and deploy various components of the system, ensuring efficient resource utilization and streamlined management. With Docker's containerization technology, the application environment is modular, scalable, and easily reproducible across different deployment environments.

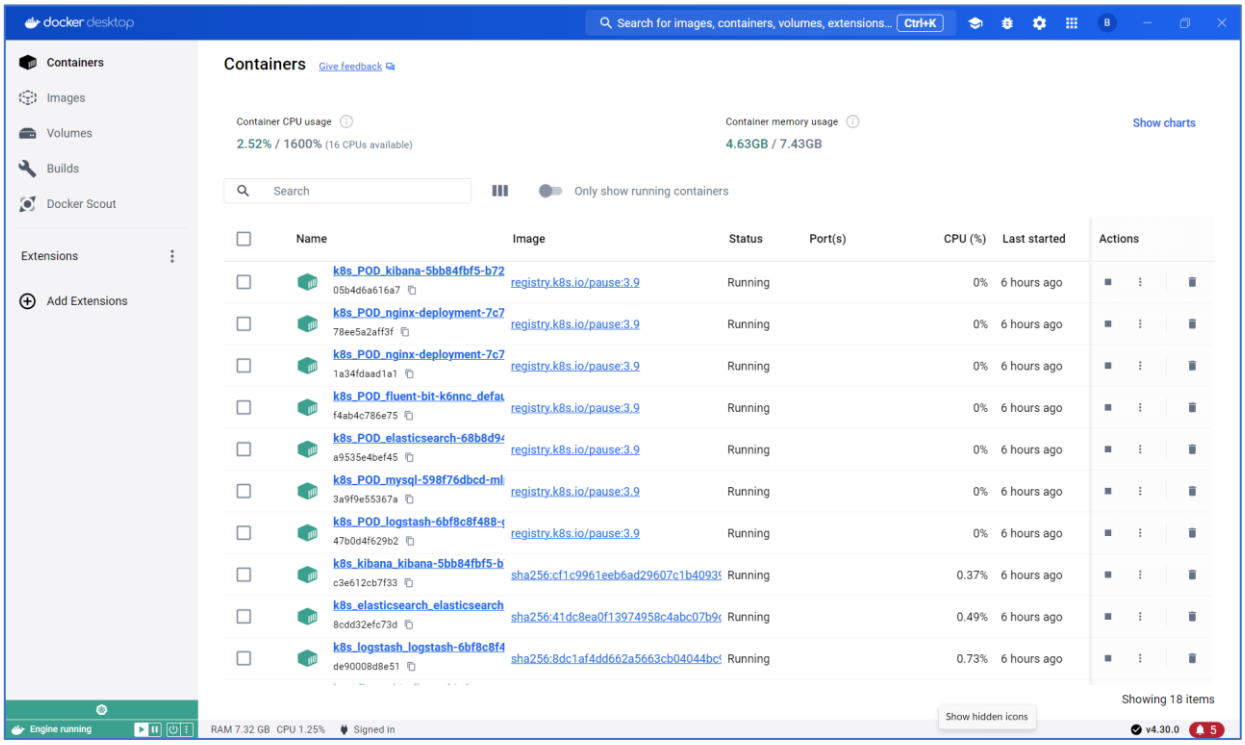


Figure 41: Docker Containers Created

Docker images serve as the building blocks for Docker containers, containing all the necessary dependencies and configurations to run specific applications or services. By listing these images, users can assess their size, version, and creation date, facilitating efficient management and deployment of containerized applications.

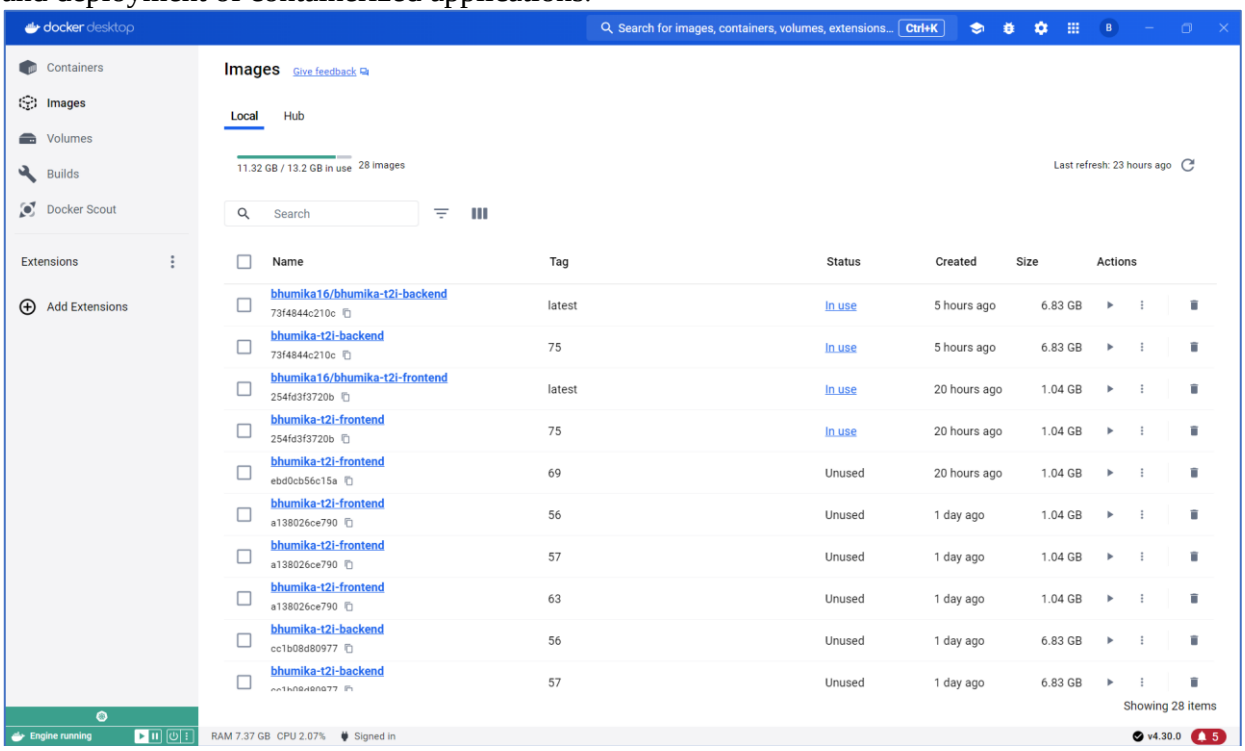


Figure 42: Docker Images Created

To view logs in Elasticsearch, you typically use tools like Kibana, Logstash, or Elasticsearch APIs directly. Here's how you can use Kibana:

- **Access Kibana:** Open your web browser and navigate to the Kibana URL. Usually, it's something like http://localhost:5601.
- **Navigate to Logs:** In Kibana, navigate to the Logs app, which is specifically designed for log analysis.
- **Index Pattern:** If you haven't set up an index pattern yet, you'll need to define one. This pattern specifies which Elasticsearch indices to search. You can create an index pattern by clicking on "Define your index pattern" and following the prompts.
- **Search Logs:** Once you have an index pattern defined, you can search for logs using the search bar. You can search by keywords, time range, and other criteria.
- **View Logs:** Kibana will display the logs matching your search criteria in a structured format. You can view log messages, filter them, aggregate data, and perform various other analysis tasks.

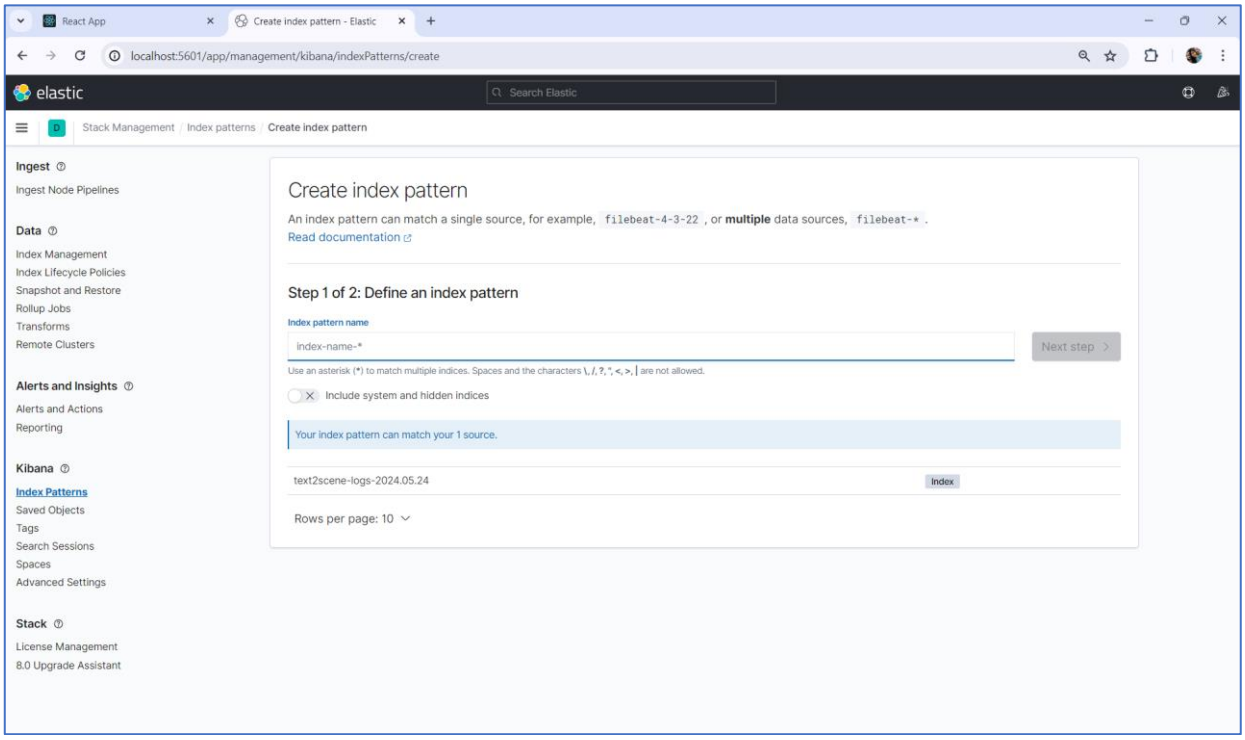


Figure 43: Setting the index pattern in Kibanna

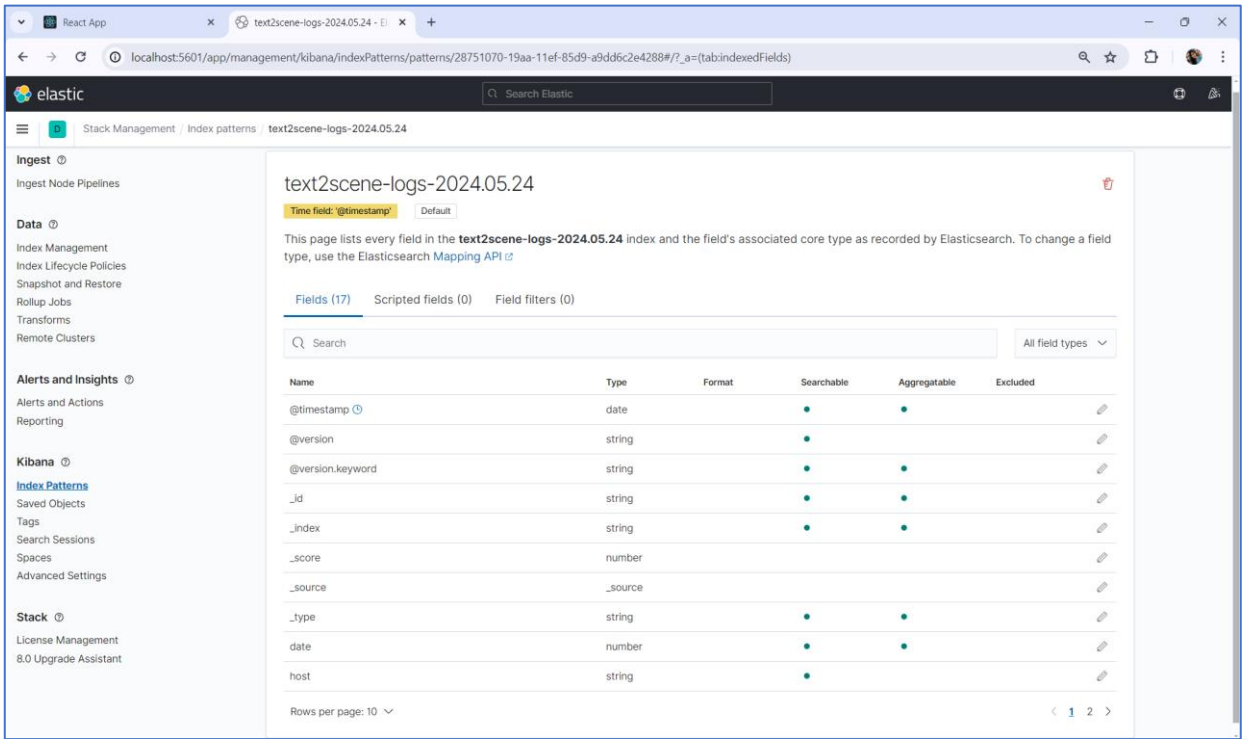


Figure 44: Logs collected

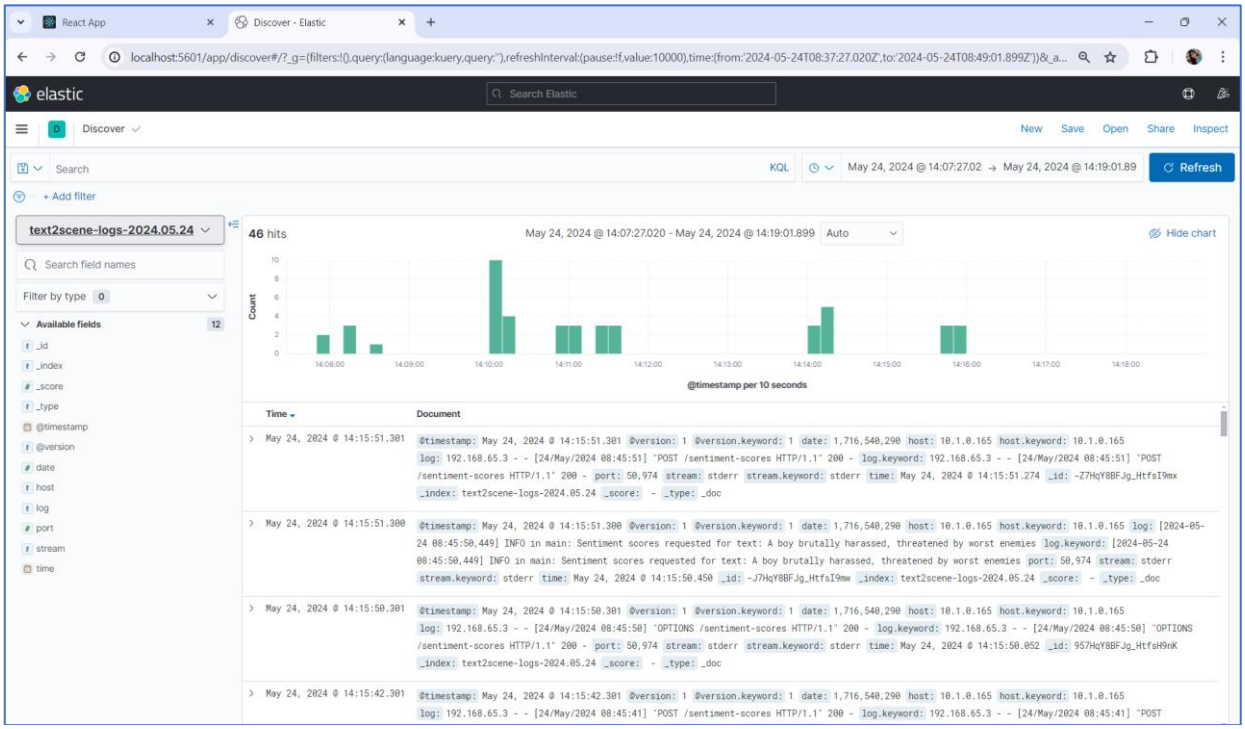


Figure 45: View Logs in Kibanna

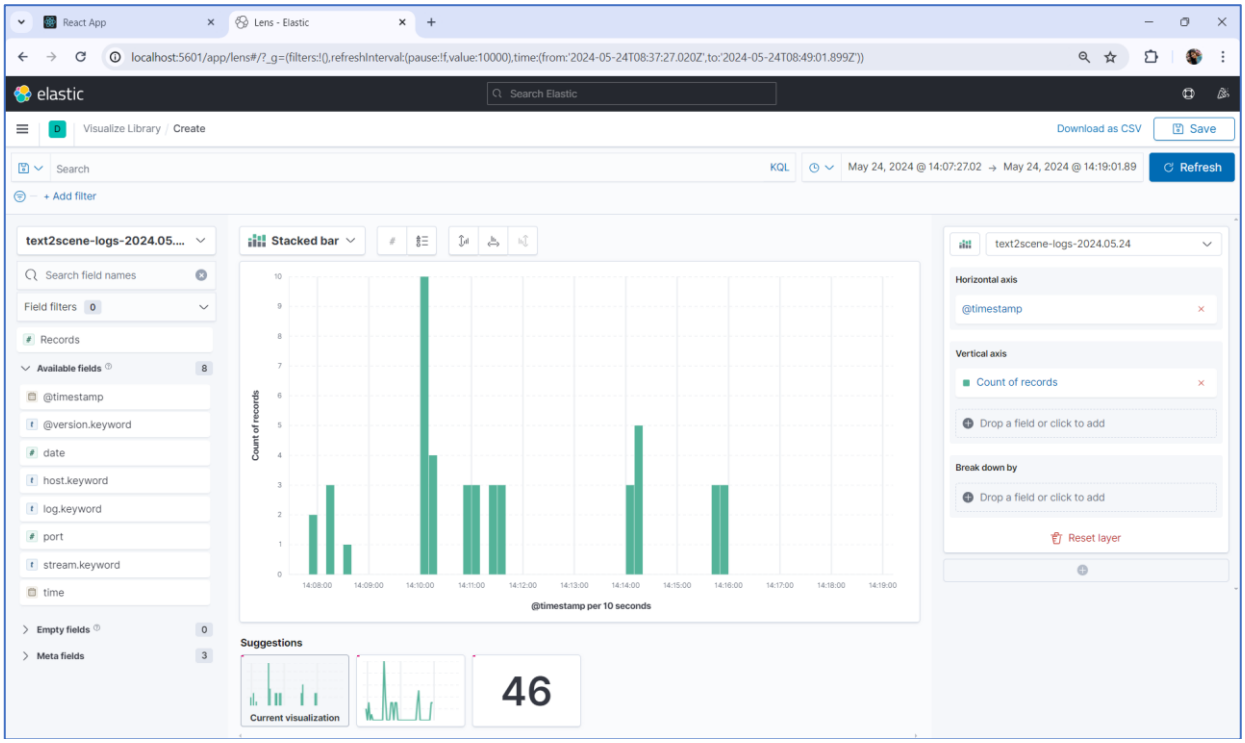


Figure 46: Visualization of Logs using Graphs

Viewing logs in Elasticsearch is streamlined through Kibana, providing an intuitive interface for log analysis and visualization. By defining index patterns and utilizing search capabilities, users can efficiently filter and analyze logs, gaining valuable insights into system and application performance. For more advanced users, direct API queries offer powerful options for customized log retrieval and analysis.

12. Using Ngrok with Webhook in Jenkins

In Jenkins pipelines, a variety of additional contributions and enhancements can be introduced to improve automation, flexibility, and overall efficiency in the software delivery process. One key aspect is the incorporation of custom pipeline stages and steps tailored to specific project requirements. By leveraging shared libraries or creating reusable functions, development teams can streamline complex workflows and maintain consistency across multiple pipelines. Integrating advanced version control strategies, such as feature branching and pull request validation, enhances collaboration and ensures code quality.

Additionally, implementing automated testing at various stages of the pipeline helps catch issues early in the development cycle. Continuous improvement can be achieved through the integration of code analysis tools, security scans, and performance testing. Moreover, the utilization of deployment strategies, like canary releases or blue-green deployments, contributes to smoother and more controlled software rollouts. Overall, Jenkins pipelines provide a versatile framework, allowing teams to continuously refine and enhance their delivery pipelines for optimized software development practices.

1. **Install Ngrok:**
   - Download and install Ngrok on the machine where your CI/CD system (e.g., Jenkins) is running.

2. **Expose Local Service:**
   - Use Ngrok to expose the local service where your CI/CD system is running.

```
ngrok http 8080
```

Note the Ngrok-generated public URL.

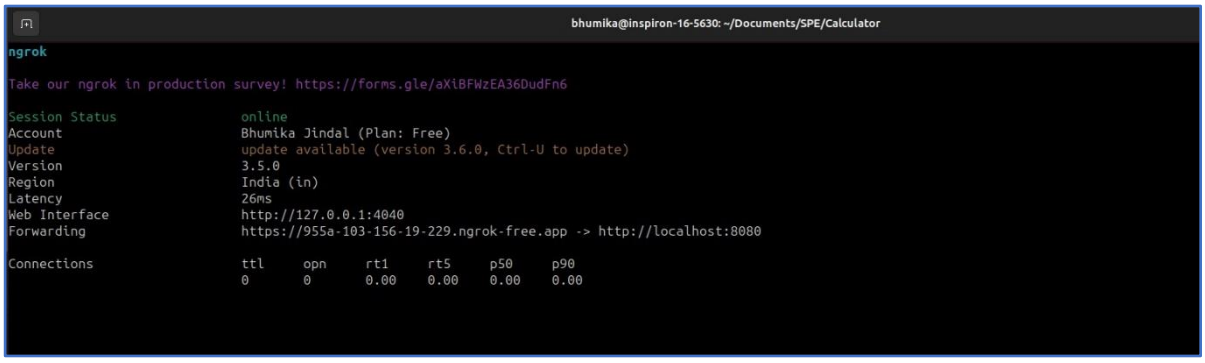


Figure 47: Ngrok Public URL

3. **Configure Webhook in CI/CD System:**
   - In your CI/CD system, navigate to the webhook or notification settings.
   - Add a webhook with the Ngrok public URL.

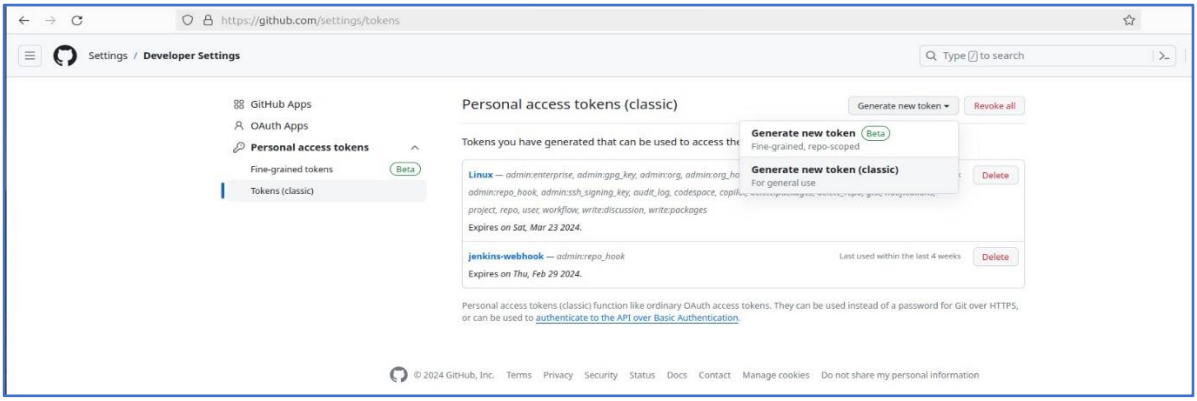


Figure 48: Generate Personal Access Token

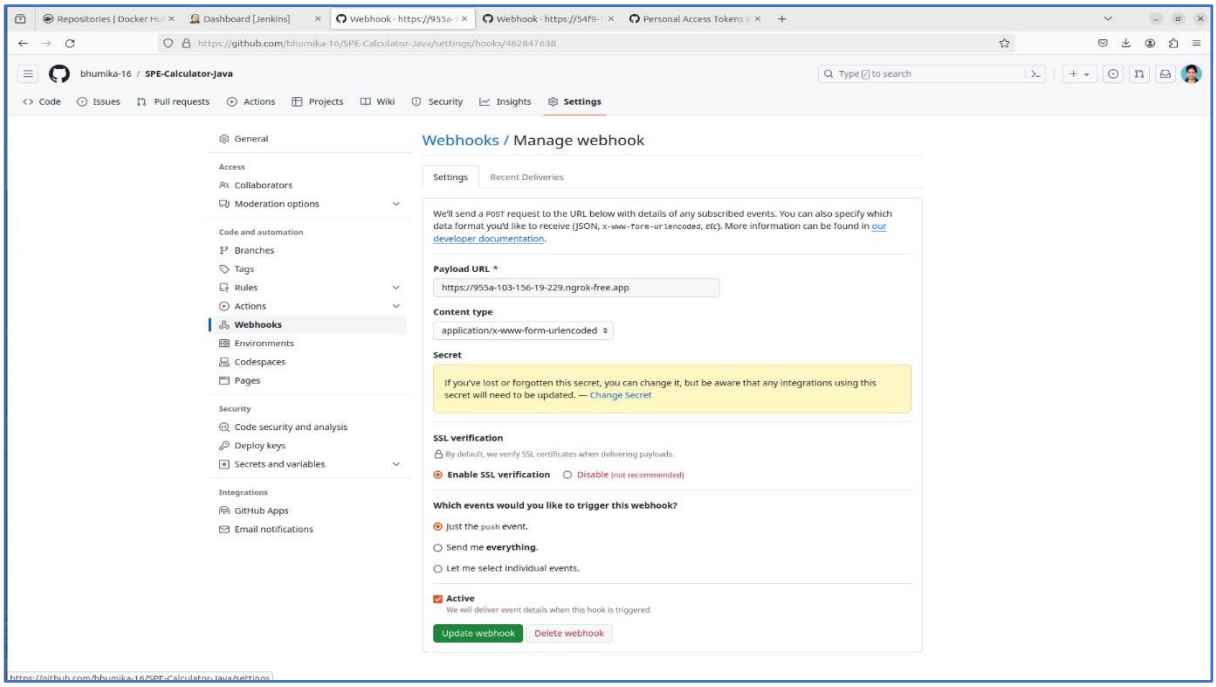


Figure 49: Webhook Created

4. **Test Webhook:**
   - Trigger a test build or event to verify that the webhook is successfully reaching your CI/CD system.

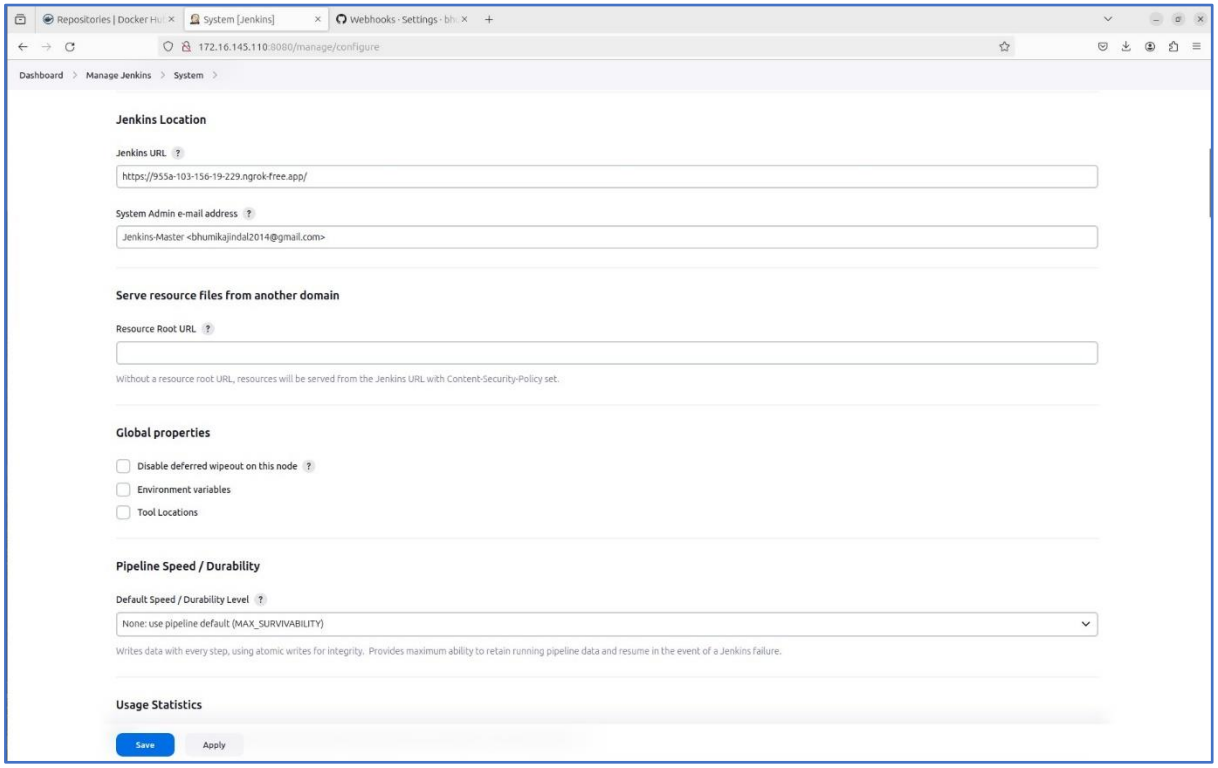


Figure 50: Add ngrok IP Address as Jenkins URL

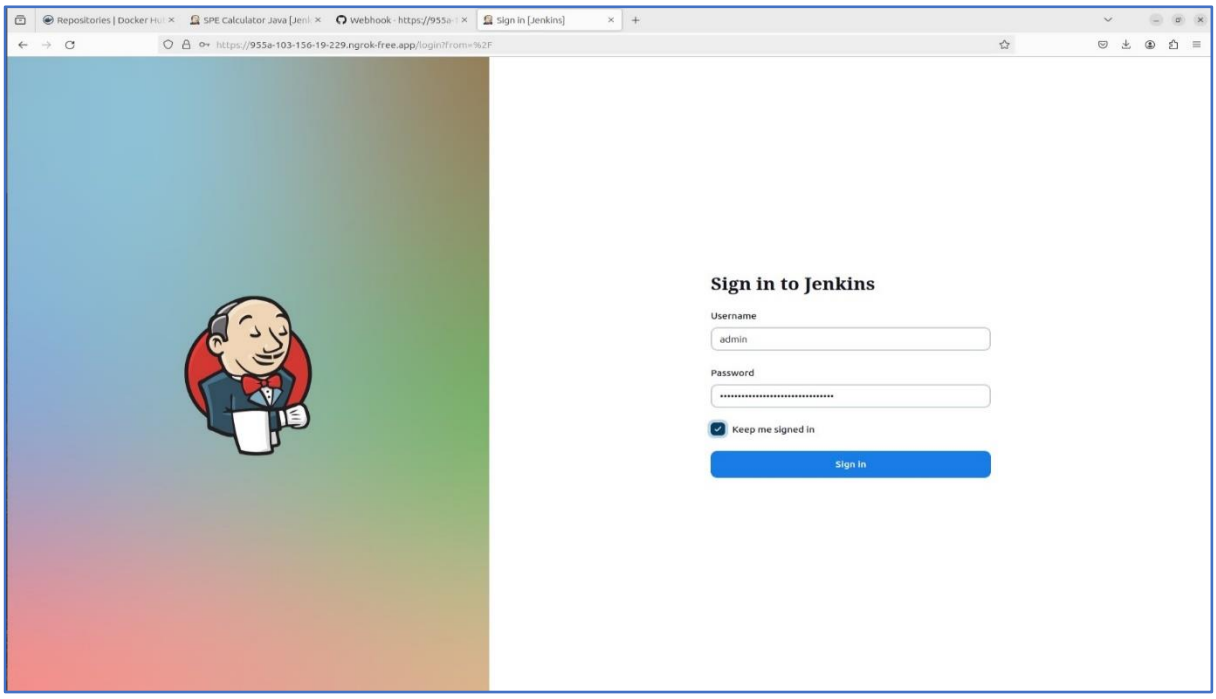


Figure 51: Login into Jenkins Using ngrok IP Address

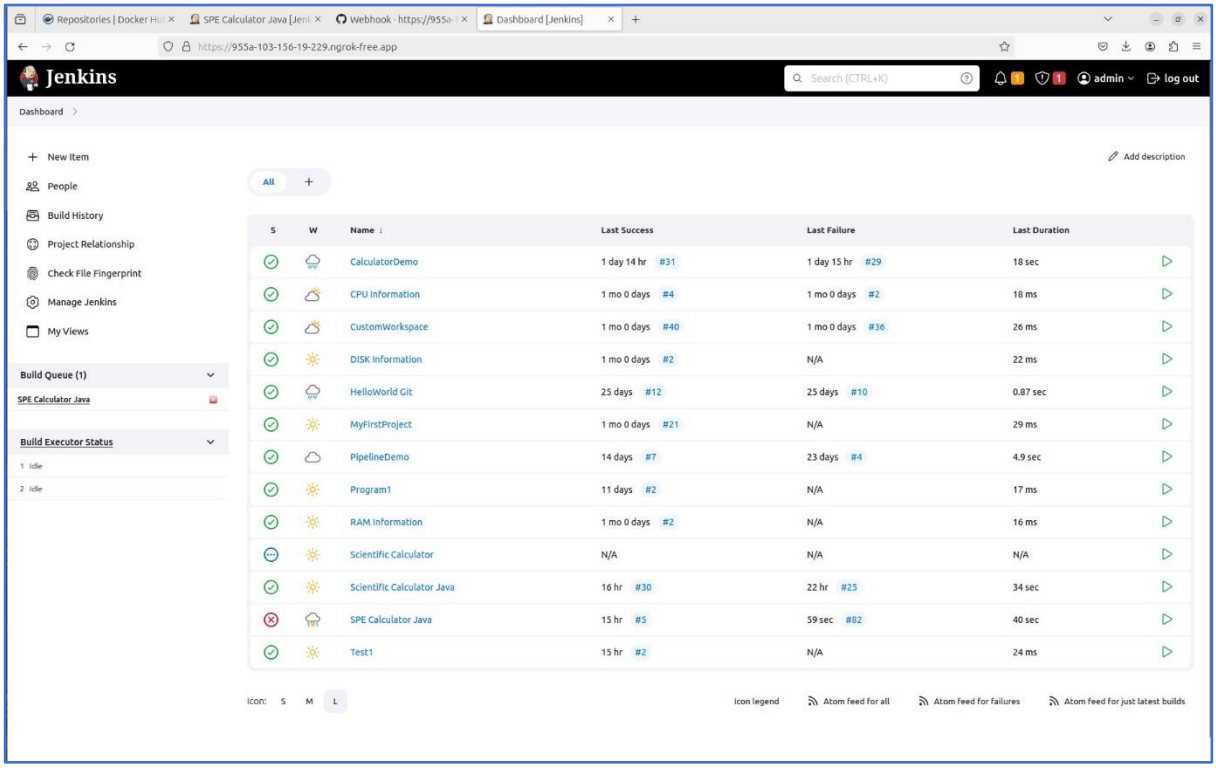


Figure 52: Jenkins Dashboard through ngrok URL

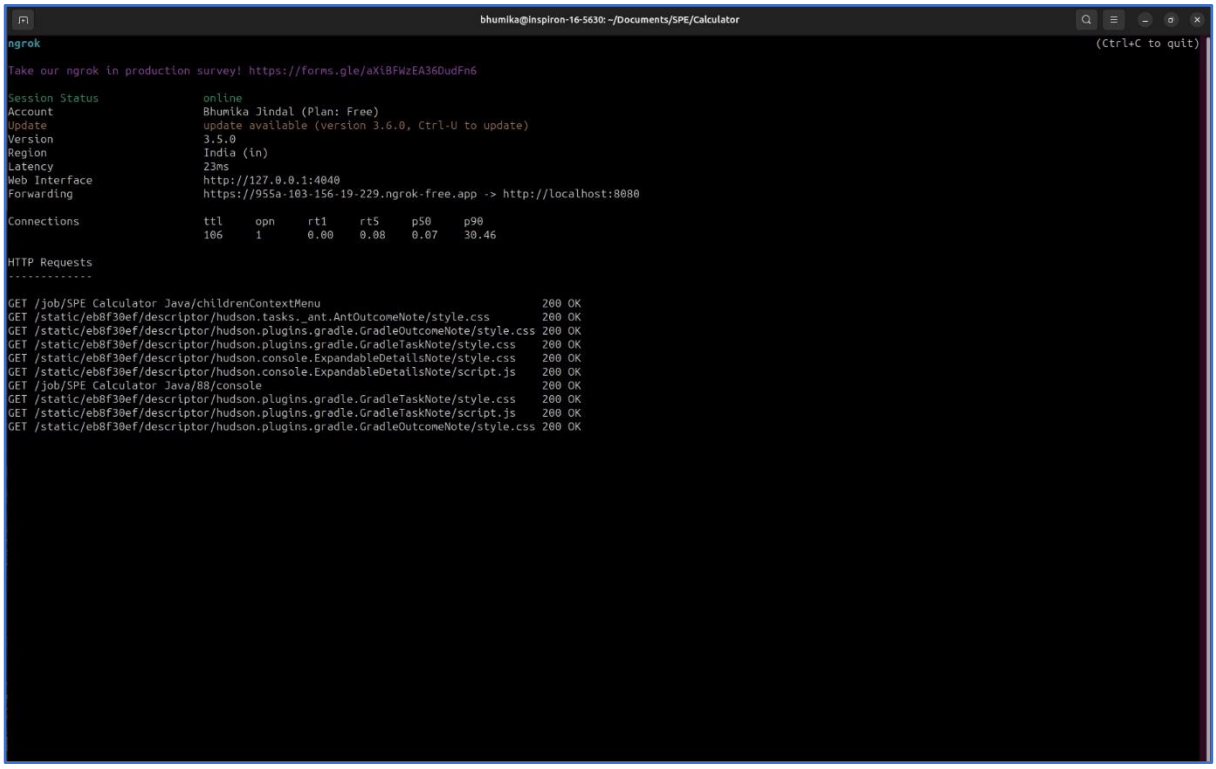


Figure 53: Terminal of ngrok

13. Application Testing

- **Trip Image with Preferences:** Generated image met criteria like "sunny," "adventurous," and "DSLR" quality while avoiding "front-facing" views.
- **Assam Garden Scene:** Image captured a girl plucking tea in Assam with "real-life HD" and "sunny" elements. Model successfully incorporated prompt enhancements, delivering realistic imagery with desired qualities.
- **Room Study Scene with Art Styles:** Evaluated model performance with real-life and anime prompts, showcasing versatility. Real-life prompt resulted in high-definition realism, while anime prompt likely produced stylized, animated artwork.
- **Rose Image Quality Evaluation:** Inference steps affected image quality, with 4 steps providing optimal balance. Model demonstrated varying performance based on inference settings, with 4 steps yielding satisfactory results.
- **Diverse Prompt Examples:** Prompt categories included places (Kolkata, Kedarnath, Kerala), food items (candies, oranges, Italian platter), animals (flamingo, tiger, cat), and people (family, friends, girl).

These prompts illustrate the model's ability to generate diverse imagery across different categories.

Figure 54: Samples Images Generated during Testing

14. Features of Our Application

- **Multilingual Support and Semantic Analysis:** "Text2Scene" supports multilingual prompts and employs semantic analysis to ensure relevance between text inputs and generated images, catering to a diverse user base and enhancing interaction across languages.
- **Accessibility Features and Content Management:** Users can specify both positive and negative prompts, ensuring precise image output alignment and facilitating customization. Additionally, audio input capability and content filtration enhance accessibility and maintain a safe and professional environment for all users.
- **Advanced AI-Powered Image Generation:** Utilizes state-of-the-art language models and image generation algorithms to produce high-quality, diverse images.
- **Backend Flask API:** Implements a robust Flask backend API to handle text inputs, image generation requests, and communication with frontend components.
- **Frontend React Interface:** Offers an intuitive and responsive React frontend interface for users to input text prompts and view generated images.
- **Scalable Kubernetes Deployment:** Orchestrates containerized workloads with Kubernetes to ensure scalability, resilience, and efficient resource utilization.
- **Containerization with Docker:** Utilizes Docker for containerization, simplifying application packaging, deployment, and environment consistency across diverse platforms.
- **Monitoring and Logging with ELK Stack:** Implements robust monitoring and logging capabilities using the ELK stack (Elasticsearch, Logstash, Kibana, Fluent Bit) to track system behavior and diagnose issues effectively.
- **Integration of Translation Tools:** Integrates translation tools such as Google Translate, Meta AI, and OpenNMT for optimal performance and multilingual support.
- **Content Filtration and Lexical Processing:** Incorporates advanced lexical processing techniques for accurate content filtration, ensuring the quality and appropriateness of generated images.
- **Continuous Integration with Jenkins:** Establishes seamless CI/CD pipelines with Jenkins for automated testing, building, and deploying changes across different environments while ensuring stability and reliability.
- **Interactive Features and User Experience:** The translation feature enables cross-lingual interaction and cultural resonance, while a carousel display and full-screen viewing options enhance browsing and inspection of images. A download option empowers users to save and utilize generated images, further enriching the user experience and encouraging creativity. Additionally, a loading button manages user expectations during image generation, ensuring a smooth and responsive interface.

These features collectively enhance the application's functionality, performance, and user experience, making it a powerful tool for text-to-image generation across languages.

Stable Diffusion's versatility makes it indispensable across industries. From character creation in visual effects to e-commerce marketing, it streamlines processes by generating detailed images based on specific prompts. Fashion enthusiasts can virtually try on clothes, while gaming studios can create assets swiftly. Additionally, it aids in web design, logo creation, and even book cover and movie poster design, showcasing its adaptability and efficiency across diverse use cases.

15. Challenges Faced

- **Dependency & Compatibility Challenges:** Integrating APIs encountered compatibility issues with existing libraries, necessitating extensive research for suitable replacements.
- **Complex Integration:** Coordinating the integration of diverse technologies like language models, image generation algorithms, and container orchestration platforms demanded meticulous planning and thorough testing.
- **Scalability and Resource Management:** Balancing the system's scalability with efficient resource allocation and management in cloud environments required careful optimization and implementation of auto-scaling mechanisms.
- **Algorithm Optimization:** Fine-tuning language models and image generation algorithms for real-time processing necessitated iterative improvements and careful consideration of model architecture and hyperparameters.
- **Monitoring, Debugging, and Security:** Setting up comprehensive monitoring with the ELK stack and ensuring robust security measures were essential for detecting and mitigating potential threats in real-time.
- **Continuous Integration and Deployment (CI/CD):** Implementing seamless CI/CD pipelines with Jenkins required rigorous testing and validation procedures to ensure smooth deployments across different environments.
- **Enhancing Graph Visualization:** Overcoming challenges in graph visualization involved thorough research and experimentation with JavaScript libraries due to compatibility issues
- **User Experience:** Designing an intuitive interface and optimizing performance across various devices and browsers involved iterative feedback and usability testing to enhance overall user satisfaction.

16. Future Scope

- **Interactive User Features:** Enhance engagement by implementing feedback mechanisms and customization options, ensuring a tailored user experience.
- **Cross-Modal Applications:** Expand application capabilities to include audio-to-image or text-to-video conversion, accommodating a wider range of user needs and preferences.
- **Real-Time Processing:** Improve user satisfaction by optimizing the platform for real-time interactions, providing immediate feedback on generated content.
- **Research Collaboration:** Foster collaboration with research institutions to stay at the forefront of AI advancements, ensuring the integration of cutting-edge techniques into the platform.
- **Cloud-Based Services:** Leverage scalable cloud solutions to support the platform's growth and ensure consistent performance and reliability across diverse user environments.
- **Enhanced Automation:** Streamline development processes through advanced automation techniques, reducing manual intervention and accelerating software delivery.
- **Microservices Architecture:** Transition to a microservices-based architecture to enhance scalability, flexibility, and fault isolation, enabling more efficient development and deployment.
- **Security Integration:** Strengthen the platform's security posture by integrating comprehensive security tools and processes throughout the CI/CD pipeline, mitigating risks and ensuring the integrity of the software.

17. Conclusion

Our meticulously designed toolchain for multi-lingual text-to-image generation serves as a cornerstone for development and operations teams. With its user-friendly interface, the calculator empowers teams to estimate costs for integrating cutting-edge AI and image generation tools, aligning with project requirements and budget constraints. This toolchain seamlessly collaborates advanced AI technologies and image generation tools with robust backend and frontend frameworks. By integrating state-of-the-art language models, image generation algorithms, Flask for backend development, and React for frontend, it streamlines content creation, reducing manual intervention and enhancing the quality and diversity of generated images.

The project highlights the pivotal role of advanced toolchains in automating content creation workflows, transcending traditional methods. Through AI-driven approaches and leveraging tools like language models, image generation algorithms, and cloud infrastructure, developers optimize workflows, minimize manual efforts, and focus on delivering high-quality, diverse content. In our multi-lingual text-to-image generation project, Kubernetes orchestrates containerized workloads for scalability and resilience, while Docker simplifies application packaging and deployment across diverse environments. The ELK stack, consisting of Elasticsearch, Logstash, Kibana, and Fluent Bit, provides robust monitoring and logging capabilities essential for maintaining application performance and diagnosing issues effectively. Together, these technologies form a cohesive and powerful toolchain, enabling seamless development, deployment, and monitoring of the text-to-image generation application.

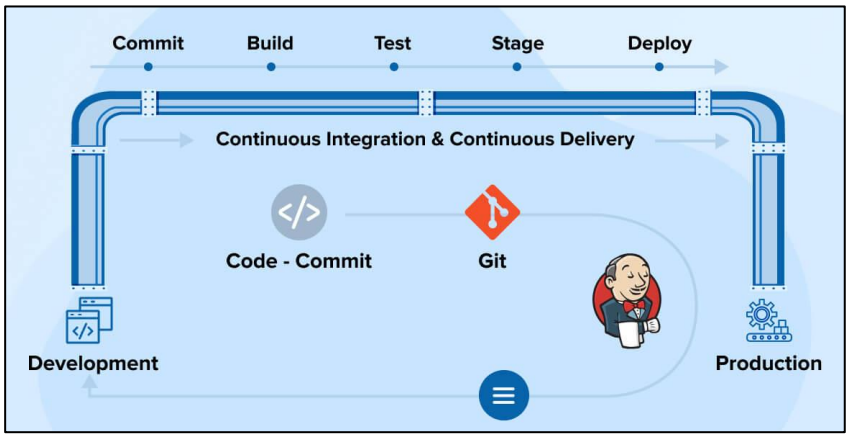


Figure 555: CI/CD Implementation using Jenkins

The seamless integration of language models, image generation algorithms, Flask, and React streamlines content generation. Cloud infrastructure automation tools optimize resource provisioning, while AI-powered image generation ensures content consistency and diversity. Cost estimation unveils multifaceted expenses, emphasizing the collective impact of advanced AI, image generation, backend/frontend frameworks, and cloud infrastructure on efficiency and informed decision-making in content creation.

18. References

In the course of developing our multi-lingual text-to-image generator project, we relied on a range of resources to harness the power of cutting-edge technologies and frameworks. Here is a compilation of the references and documentation sources that were instrumental in shaping our project:

- **React - A JavaScript library for building user interfaces.** React.js Documentation. Available at: https://reactjs.org/docs/getting-started.html
- **Spring Boot Reference Guide.** Spring Boot Documentation. Available at: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
- **MySQL Documentation.** MySQL. Available at: https://dev.mysql.com/doc/
- **Jenkins Documentation.** Jenkins. Available at: https://www.jenkins.io/doc/
- **Docker Documentation.** Docker. Available at: https://docs.docker.com/
- **Docker Compose Documentation.** Docker Compose. Available at: https://docs.docker.com/compose/
- **Kubernetes Documentation.** Kubernetes. Available at: https://kubernetes.io/docs/home/
- **Flask Documentation.** Flask. Available at: https://flask.palletsprojects.com/en/2.0.x/
- **Elastic Stack (ELK) Documentation.** Elastic. Available at: https://www.elastic.co/guide/en/elastic-stack/current/index.html

These valuable resources played a pivotal role in guiding our development efforts, ensuring that our multi-lingual text-to-image generator was built on a foundation of industry best practices and the latest advancements in the field. We express our gratitude to the creators and maintainers of these references, whose contributions have been instrumental in the successful realization of our project.

Additional References for AI and Text Moderation

- **Text Moderation Model.** KoalaAI. Available at: https://huggingface.co/KoalaAI/Text-Moderation
- **Emotion Classification Model.** Joeddav. Available at: https://huggingface.co/joeddav/distilbert-base-uncased-go-emotions-student
- **Stable Diffusion with AWS.** Available at: https://aws.amazon.com/what-is/stable-diffusion/
- **Stable Diffusion Documentation.** StabilityAI. Available at: https://huggingface.co/docs/hub/en/stanza
- **Stable Diffusion Model.** StabilityAI. Available at: https://stability.ai/
- **Research Papers:**
  - **Paper on Text Moderation:** Available at: https://arxiv.org/pdf/2307.01952
  - **Paper on Emotion Classification:** Available at: https://arxiv.org/pdf/2112.10752

These additional references provided crucial insights and technical support for implementing advanced features such as text moderation, emotion classification, and text-to-image generation in our project.