Project 5 - JupyterLab Plugin for Kubeflow

Summary

This project aims to develop a unified JupyterLab plugin for Kubeflow that modernizes and consolidates existing tools like Elyra, Kale, and Jupyter Scheduler. The current fragmentation of tools creates complexity for data scientists working in Kubernetes environments, requiring them to switch between different interfaces. By creating a cohesive plugin that seamlessly integrates with Kubeflow Pipelines and other Kubeflow components, this project will significantly improve the workflow efficiency for data scientists and ML engineers, allowing them to develop, test, and deploy machine learning workflows directly from their familiar JupyterLab environment.

Personal Information

- **Full Name**: Amrit Kumar
- **Email Address**: amritkmr4272@gmail.com
- **GitHub Profile**: https://github.com/Amrit27k
- **LinkedIn**: http://www.linkedin.com/in/amritkmr72
- **University/College**: Newcastle University
- **Degree Program**: MSc Cloud Computing
- **Year of Study**: Final Year (2023-2024)
- **Country of Residence**: United Kingdom
- **Timezone**: GMT/BST (UTC+0/+1)

Qualifications & Motivation:

Drawing from my recent master's dissertation on "Adaptive Multi-Model Serving using KServe" at Newcastle University, I have gained deep expertise in Kubernetes-based MLOps infrastructure and model serving systems. My experience includes:

- Working extensively with Kubeflow, Kubernetes, and containerized ML applications
- Implementing and evaluating KServe for multi-model inference deployments
- Designing and testing autoscaling mechanisms for ML models in production environments
- Proficiency in Python backend development for ML systems
- Experience with JavaScript/TypeScript for building web-based interfaces, including work with React for building interactive UIs
- Practical knowledge of model monitoring and observability using Prometheus and Grafana
- Familiarity with JupyterLab architecture and extension development through my university research project on custom visualization extensions

My research involved creating an adaptive multi-model serving framework that optimized resource usage, improved latency, and enhanced throughput through dynamic scaling and request batching. This background provides me with unique insights into the challenges of ML model deployment and management that would be valuable for designing an effective JupyterLab plugin for Kubeflow.

I commit to dedicating 35-40 hours per week to this project throughout the GSoC period. My motivation stems from my firsthand experience with the fragmentation of tools in the Kubeflow

ecosystem. During my dissertation, I witnessed how model deployment workflows could be streamlined through integration, and I'm excited to apply these insights to create a unified development experience for data scientists and ML engineers, bridging the gap between interactive notebook development and production-grade ML infrastructure.

Goals

1. Develop a unified JupyterLab plugin architecture that can replace and consolidate the functionality of Elyra, Kale, and Jupyter Scheduler
2. Implement seamless integration with Kubeflow Pipelines, enabling users to:
    - Convert notebook cells to pipeline steps
    - Define and visualize pipeline workflows within JupyterLab
    - Submit pipeline runs to Kubeflow
    - Monitor pipeline execution status
3. Create a user-friendly UI for pipeline development and management using modern React components
4. Implement initial integration with Kubeflow Notebooks for easier notebook provisioning
5. Provide comprehensive documentation and usage examples to encourage adoption
6. Establish a robust testing framework for ensuring plugin reliability

Non-Goals

- This project will not implement full integration with all Kubeflow components beyond Pipelines and basic Notebooks functionality in the initial phase
- The plugin will not attempt to replicate all advanced features from Elyra, Kale, and Jupyter Scheduler, but will focus on the core functionality most valuable to users
- This project will not include integration with external CI/CD systems beyond what's provided by Kubeflow Pipelines
- We will not build custom visualizations for pipeline monitoring beyond basic status reporting (full monitoring UI will rely on Kubeflow's existing UI)
- The plugin will not include features for managing cluster resources or Kubernetes configurations

Technical Details

The architecture of the JupyterLab plugin will follow a modular design with these key components:

1. **Backend Components (Python)**:
    - API client for Kubeflow Pipelines using the KFP SDK
    - Notebook parsing module to extract cell metadata and code for pipeline conversion
    - Configuration management for storing Kubeflow cluster information
    - Pipeline template generation based on notebook structure and annotations

2. **Frontend Components (TypeScript/React)**:
    - Pipeline editor panel for visually creating and editing pipelines
    - Execution status viewer for monitoring pipeline runs
    - Settings interface for configuring Kubeflow connections
    - Extension commands and launcher items for quick access to functionality
3. **Integration Architecture**:
    - JupyterLab extension points will be used to add UI components (panels, menus, commands)
    - Communication between frontend and backend will use JupyterLab's server extension API
    - Kubeflow Pipelines integration will utilize the KFP Python SDK
    - Notebook-to-pipeline conversion will support parameterization and dependencies

Technical Challenges and Solutions:

1. **Challenge**: Maintaining compatibility with multiple JupyterLab and Kubeflow versions.
   Solution: Design a version-aware plugin architecture that can adapt to API changes through versioned interfaces, informed by my experience with microservices architecture. Use JupyterLab's extension system compatibility layers and implement thorough version testing.
2. **Challenge**: Handling diverse pipeline component types from notebooks.
   Solution: Develop a flexible annotation system that allows users to specify component requirements and dependencies.
3. **Challenge**: Ensuring a smooth user experience between local development and remote execution.
   Solution: Implement robust error handling and status updates to keep users informed during pipeline submission and execution.
4. **Challenge**: Managing authentication with Kubeflow securely.
   Solution: Utilize JupyterLab's credentials storage system and implement secure token management.

The implementation will follow these technical approaches:

- Use TypeScript for all frontend code to ensure type safety
- Implement React components for UI elements to maintain consistency with JupyterLab's design
- Utilize the Python KFP SDK for Kubeflow Pipelines integration
- Follow a test-driven development approach with Jest for frontend and pytest for backend
- Use GitHub Actions for CI/CD to ensure code quality

Test Plan

The testing strategy will include multiple layers to ensure quality and reliability:

1. **Unit Tests**:
    - Backend Python tests using pytest (target: 80%+ code coverage)
    - Frontend TypeScript tests using Jest and React Testing Library
    - Isolated testing of notebook parsing, pipeline generation, and API communication
2. **Integration Tests**:
    - End-to-end tests for notebook-to-pipeline conversion
    - Tests for Kubeflow Pipelines API integration using mock servers
    - JupyterLab extension activation and functionality tests
3. **User Acceptance Testing**:
    - Manual testing scenarios documented as test plans
    - Dogfooding by using the extension for real ML workflows
    - Soliciting feedback from Kubeflow community members
4. **Continuous Integration**:
    - GitHub Actions workflows for automated testing on pull requests
    - Linting and code style checks (flake8, eslint)
    - Package build tests to ensure installability
5. **Compatibility Testing**:
    - Testing across multiple JupyterLab versions (2.x, 3.x)
    - Testing with different Kubeflow versions
    - Cross-browser testing for the UI components

Estimation of Deliverables

- **Milestone 1: Core Architecture and Basic Integration (Weeks 1-3)**
    - Set up development environment and project structure
    - Develop the basic plugin architecture with JupyterLab integration
    - Implement configuration UI for Kubeflow connections
    - Create initial notebook cell annotation system
- **Milestone 2: Kubeflow Pipelines Integration (Weeks 4-6)**
    - Implement notebook-to-pipeline conversion logic
    - Develop Pipeline editing UI components
    - Create pipeline submission and basic status tracking

    - Initial documentation of core features
- **Milestone 3: Advanced Features and Refinement (Weeks 7-9)**
    - Implement more advanced pipeline features (parameters, artifacts)
    - Add basic integration with Kubeflow Notebooks
    - Enhance pipeline visualization and editing capabilities
    - Improve error handling and user feedback
- **Milestone 4: Testing, Documentation, and Polishing (Weeks 10-12)**
    - Comprehensive testing across different environments
    - Complete user and developer documentation
    - Performance optimization and bug fixing
    - Prepare final release and presentation

Timeline alignment with GSoC 2025:

- May 8 - June 1: Community Bonding Period (Get familiar with codebase, finalize design details)
- June 2: Coding begins (Milestone 1)
- July 14: Midterm evaluation (Expected completion through Milestone 2)
- August 25 - September 1: Final submission (Completion of all milestones)

Additional Notes (Optional)

My experience with MLOps automation, Edge AI optimization, and distributed training pipelines will be particularly valuable for this project. Having achieved 40% faster inference times through distributed training and 65% reduction in model training time in my recent work, I'm well-positioned to build a JupyterLab plugin that not only improves workflow but also enhances performance.

Additionally, my experience working in Agile environments and collaborating with cross-functional teams ensures that I can effectively engage with the Kubeflow community and incorporate feedback throughout the development process. I'm excited to contribute to the Kubeflow ecosystem by creating a tool that bridges the gap between development and deployment, making ML workflows more accessible and efficient for data scientists and ML engineers.