Comparative Study of JupyterLab Deployment Models

EECS 268

Michael Pisman

UC Merced

Spring 2025

https://github.com/michael-pisman/jupyter-knative

Abstract

This project explores two ways to run JupyterLab notebooks at scale: a serverless approach using Knative Serving and a traditional Kubernetes deployment under JupyterHub. I built a lean JupyterLab container, deployed it on a MicroK8s cluster with Knative, and measured startup times. I also installed JupyterHub's default spawner on the same cluster. While I successfully gathered Knative results, technical issues with storage and custom spawners prevented full performance tests for the JupyterHub setup. This report describes the goals, methods, achievements, difficulties, current findings, and next steps toward a more efficient "serverless IPyKernel" model.

Introduction

Interactive notebooks are essential for data analysis and teaching, but keeping them running wastes resources when idle. Users expect fast startup times when they open a notebook. This study aims to compare a serverless model where notebook servers scale to zero when idle and restart on demand with a more common Kubernetes plus JupyterHub model, where each user has a dedicated pod. The objective is to see which approach offers better responsiveness and resource efficiency.

Background

Knative Serving is a Kubernetes extension that treats containers as serverless functions. It will automatically scale pods down to zero when there is no traffic and bring them back up immediately when a request arrives. It uses an internal proxy to queue incoming requests during cold starts, which helps reduce perceived delays.

In the traditional Kubernetes model, we create a Deployment and optionally use the Horizontal Pod Autoscaler (HPA) to scale pods based on CPU use. The HPA cannot scale to zero, so pods remain running until they are manually removed or cleaned up by a separate process. JupyterHub's default KubeSpawner creates one pod per user, giving good isolation but maintaining idle pods when the user is away.

Existing Jupyter solutions include Elyra pipelines, which turn notebooks into batch jobs, and Jupyter Enterprise Gateway, which runs kernels remotely but still launches a container for each kernel. These systems address different needs but do not combine low idle cost with fast interactive startup at scale.

Implementation Overview

I set up a three-node MicroK8s cluster and enabled DNS, storage, ingress, and Knative addons. To create the notebook environment, I started from a minimal Python image and installed only JupyterLab. A simple entrypoint script launched the server on port 8888 with a fixed token for authentication.

For the serverless experiment, I defined a Knative Service that could scale from zero to five pods, targeting one request per pod and allowing up to ten concurrent requests inside each container. I applied this to the cluster and verified that a cold start spun up a pod and served the notebook interface, then that further requests were served quickly by the same pod.

At the same time, I deployed JupyterHub using its Helm chart in a separate namespace. I configured a dummy authenticator so a single test user could log in, and I used ephemeral storage to avoid delays caused by persistent volume claims. I exposed the Hub through Traefik and confirmed that logging in and spawning a notebook pod worked correctly.

Throughout the process, I recorded Dockerfile and Kubernetes manifest details, automated deployments with Helm, and summarized the work in a slide deck.

Achievements and Challenges

I succeeded in building and publishing the container image, setting up Knative and Traefik for serverless notebooks, and measuring startup times for that setup. I also got JupyterHub running with its default spawner. However, I faced several hurdles. The Ceph-based storage volumes would bind but not mount correctly in the Hub pods, delaying further tests. Attempts to integrate a custom KnativeSpawner into JupyterHub ran into missing dependencies, RBAC conflicts, and Helm release ownership errors. I also discovered that using more than one Knative pod broke Jupyter's in-memory session state, leading me to explore sticky sessions and external state storage.

Current Results and Status

The Knative deployment achieved a cold-start latency of 3.19 seconds and a warm-start latency of 0.13 seconds for a single user. These measurements show that a small, optimized container combined with Knative's request-driven scaling can deliver interactive notebooks with minimal delay. The JupyterHub deployment is fully functional in terms of user login and pod management, but its performance metrics remain pending due to the storage and spawner integration issues. The custom KnativeSpawner code exists as a prototype but has not yet been tested in production.

Future Work

To complete the comparison, I will automate performance tests for the JupyterHub setup, measuring cold-start and warm-start times over multiple runs. I plan to implement session affinity in Traefik or develop a shared session store so that Knative services can run multiple pods without breaking user sessions. Beyond that, I will prototype a "serverless IPyKernel" system that dispatches individual notebook cells to pooled lightweight containers, keeping common libraries in memory and storing interpreter state externally. I will also create a cost model mapping CPU-seconds and pod-hours to cloud pricing, and compare those costs for each deployment model.

Conclusion

This project highlights the trade-offs between serverless and traditional hosting for JupyterLab. The serverless Knative model provides rapid startup and zero idle cost, but it introduces new complexity around session state and multi-user routing. The Kubernetes plus JupyterHub model is simpler to operate and integrates well with storage and authentication, but it yields slower startup times and requires manual cleanup of idle pods. Although time constraints prevented completion of the performance comparison, the work accomplished lays a clear foundation for finalizing the study and exploring a next-generation serverless kernel architecture.

References

Knative documentation - https://knative.dev

Kubernetes documentation - https://kubernetes.io

Jupyter Enterprise Gateway - https://jupyter-enterprise-gateway.readthedocs.io

Zero-to-JupyterHub - https://zero-to-jupyterhub.readthedocs.io