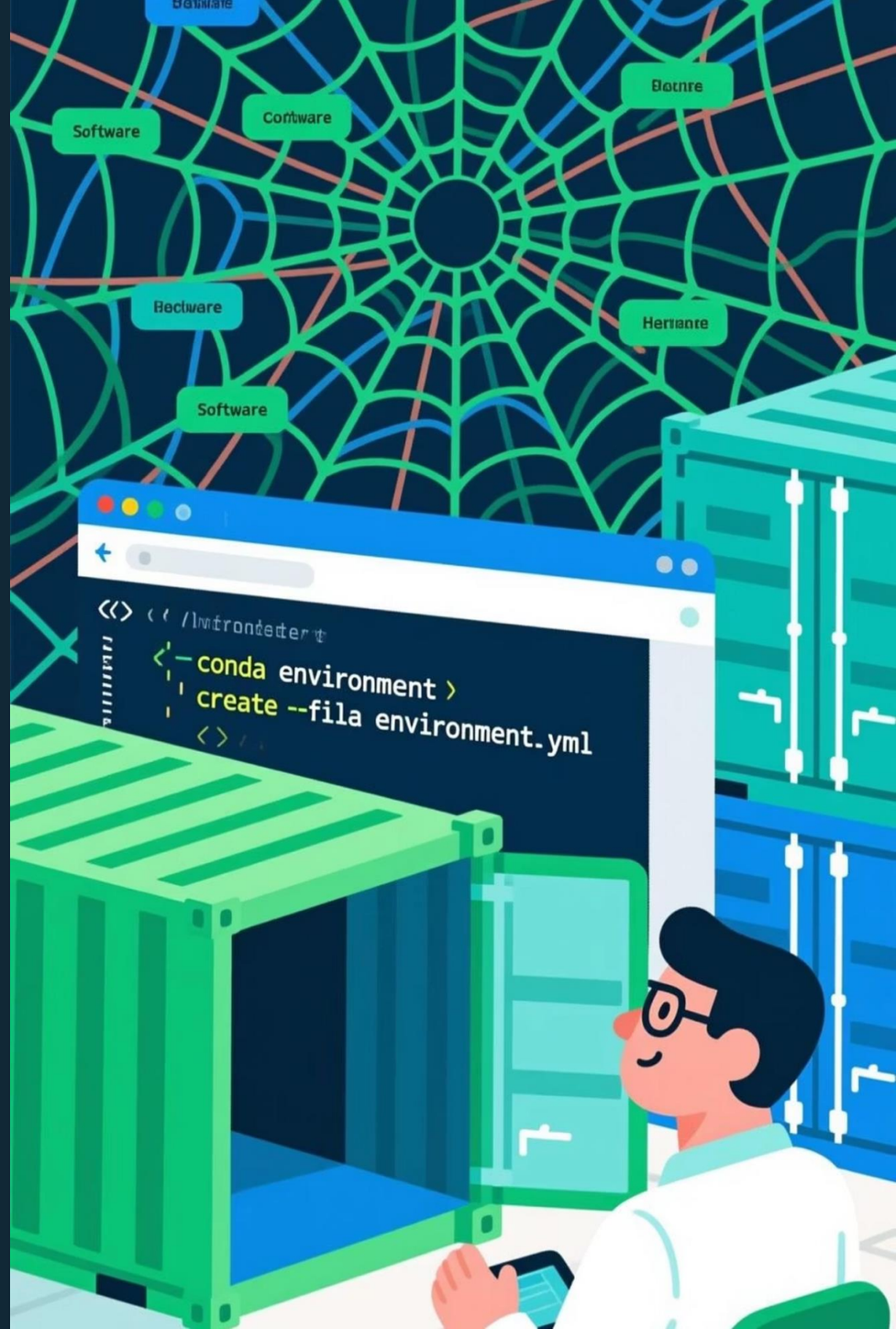
Reusable Conda Environments as JupyterLab Kernels

Tired of environment conflicts, 'works on my machine' issues, and endless dependency management? This presentation tackles the core pain points of AI development, offering a robust solution for consistent, portable, and scalable research and deployment workflows.

Presented by Vivek Kolasani & Uday Malgireddy | November 4, 2025



Agenda

01

Core Problem

Environment drift and dependency conflicts plaguing AI teams

02

Solution

Centralized, scalable, reusable environments

03

Core Technologies

Docker, Kubernetes, Conda, JupyterLab

04

Architecture

Pods and persistent volumes design

05

Persistence

Managing Conda environments in Kubernetes

06

Demo

Creating a Ray environment in action

07

Collaboration

Sharing environments across teams

08

Impact

Metrics and measurable benefits

Problem

The "It Works on My Machine" Nightmare

Environment inconsistencies create a cascade of problems that slow teams, break deployments, and waste valuable engineering time. The cost of these issues extends far beyond immediate troubleshooting—they erode team morale, delay product launches, and undermine stakeholder confidence.

Wasted Time

Teams spend countless hours debugging environment issues instead of focusing on model development and innovation, significantly slowing project progress and time-to-market. Engineers lose productive days wrestling with dependency conflicts rather than building features.

Collaboration Breakdown

Environment drift causes inconsistent setups across team members, making onboarding new data scientists slow and frustrating while hindering effective collaboration and knowledge sharing. Code that works for one person mysteriously fails for another.

Deployment Failures

Inconsistent dependencies between development and production environments lead to failed deployments, risking project reliability, stakeholder trust, and production stability. Critical models fail in production despite passing all development tests.



Solution

Centralized, Reusable, and Scalable Environments

Our solution eliminates environment inconsistencies through a unified architecture that separates compute from storage, enabling true reproducibility across the entire ML lifecycle. By centralizing environment management and leveraging Kubernetes orchestration, teams achieve consistent workflows from development through production.

1

Centralized Environment Management

Define Conda environments once and store them centrally to eliminate configuration drift and ensure all team members use the exact same setup, reducing overhead and maintenance burden. No more hunting down missing dependencies or resolving version conflicts.

2

Portability and Reusability

Share environments seamlessly across machines and projects, enabling data scientists to focus on development without rebuilding dependencies or managing complex installation procedures. Environments travel with your work, not against it.

3

Automated Orchestration

Leverage Kubernetes to automatically manage, scale, and deploy containerized JupyterLab instances with consistent environments, ensuring high availability and resource efficiency. Infrastructure scales to meet demand without manual intervention.

4

Complete Reproducibility

Version control and persistent storage guarantee that environments remain stable and identical from development to production, enabling true reproducibility across the ML lifecycle. What works in development works in production, every time.

The Building Blocks

Core Technologies

Our solution leverages industry-standard technologies that work together to deliver a robust, scalable platform for reproducible AI development. Each component plays a critical role in ensuring environment consistency, portability, and ease of use.

Docker

Containerizes our JupyterLab application to ensure it runs identically across all environments and platforms, eliminating "works on my machine" issues and simplifying deployment pipelines. Containers package everything needed to run the application.

Kubernetes

Orchestrates and scales containers automatically, managing compute resources and persistent storage efficiently for reliable AI workflows with self-healing capabilities. Kubernetes handles infrastructure complexity so teams can focus on development.

Conda

Creates and manages isolated Python environments with complex dependency resolution, enabling reproducible and portable AI projects across diverse package ecosystems. Conda handles the notorious challenge of Python dependency management.

JupyterLab

Interactive development environment favored by data scientists for exploratory analysis and seamless integration with kernels and environments, supporting collaborative notebook workflows. JupyterLab provides the familiar interface data scientists love.

Architecture

Architecture at a Glance

The architecture separates stateless compute from stateful storage, enabling environment persistence, team collaboration, and seamless pod lifecycle management. This separation ensures that user work and environments survive infrastructure changes while maintaining flexibility and scalability.



Data Scientist Access

Users connect via web browsers to the JupyterLab interface running inside Kubernetes, enabling seamless remote development from anywhere with an internet connection.



Stateless JupyterLab Pod

The JupyterLab application runs inside a Kubernetes pod designed to be stateless, allowing easy scaling and redeployment without losing user data or environment configurations.



Persistent Volume Storage

A persistent volume is mounted to the pod, providing durable network-attached storage that survives pod restarts, migrations, and infrastructure changes.



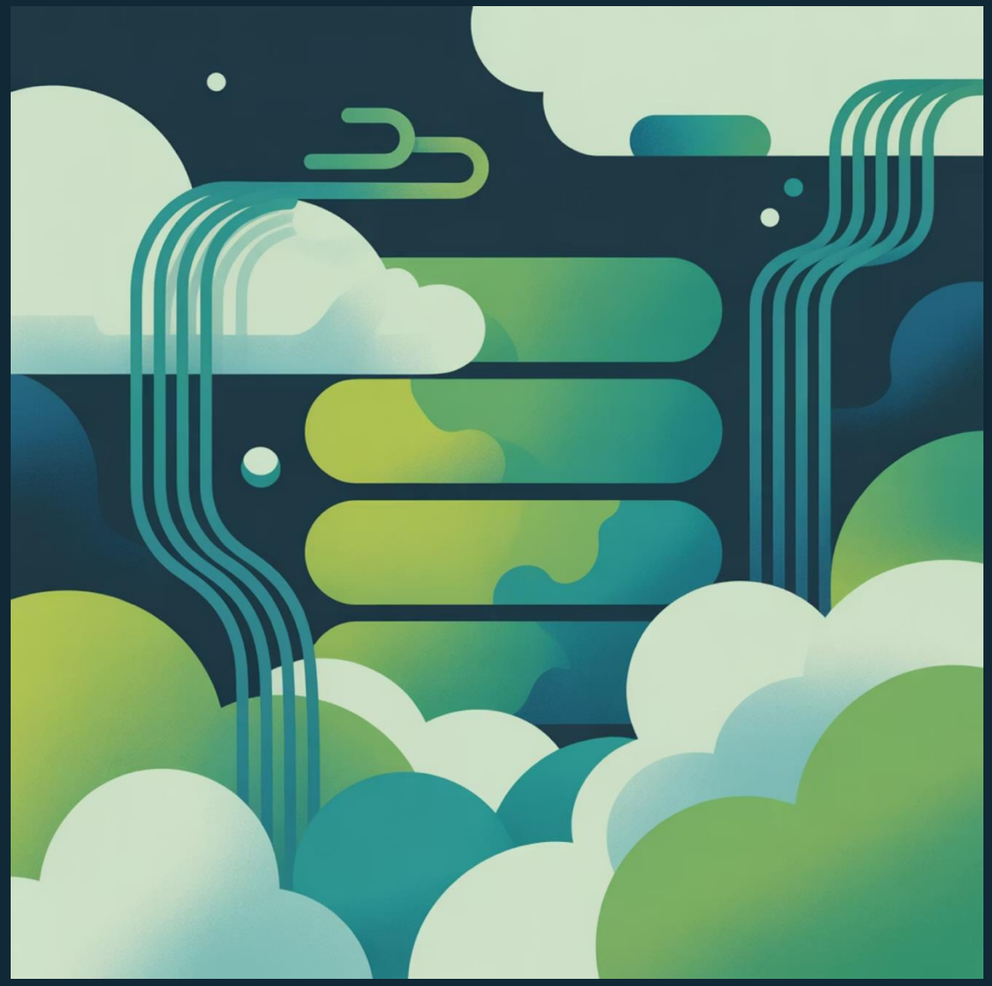
Shared Data and Environments

The persistent volume stores user notebooks, datasets, and shared Conda environments, enabling collaboration and consistent reproducibility across the entire team.

Persistence

The Magic: Persistent Conda Environments

How Persistent Storage Enables Environment Reusability



- **Kubernetes PersistentVolumeClaim** attaches network-backed storage to JupyterLab pods, making environment data persistent beyond pod lifecycle
- The `CONDA_ENVS_PATH` environment variable directs Conda to create and manage environments on the mounted persistent volume
- Pod configuration includes `volumeMounts` and `volumes` sections to bind persistent storage to the container's filesystem
- This setup ensures Conda environments and dependencies remain intact and accessible across pod restarts and scaling events

Customize the YAML snippet to reflect your specific PersistentVolumeClaim names and mount paths for your Kubernetes setup.

The key to environment reusability lies in decoupling environment storage from compute resources. By storing Conda environments on persistent volumes, they become shared assets



Demo

Creating a "Ray" Environment

This practical demonstration shows how simple it is to create a new shared environment that becomes immediately available to all team members. The entire process takes just minutes, yet provides lasting benefits for team productivity and collaboration.

> This command creates a new Conda environment named 'ray-env' directly on the persistent volume, ensuring it is shared and accessible to all team members:

```
conda create --prefix /home/local/data/conda-envs/ray-env
python=3.9 ray -y
```

After creation, register this environment as a JupyterLab kernel to make it available in the notebook interface for everyone on the team. The kernel will appear in the launcher and kernel selection menu, ready for immediate use across all notebooks. Team members can start using this environment within seconds of its creation, without any additional configuration or installation steps on their end.

Sharing Environments Across the Team

Collaboration

Environment sharing transforms team collaboration by eliminating redundant work and ensuring consistency. When one person creates an environment, everyone benefits immediately—no more duplicate installations, version mismatches, or configuration drift across team members.

Create Once

A single team member creates the Conda environment with all required dependencies on the shared persistent volume, eliminating duplication of effort and ensuring consistency across the organization.

Register Kernel

The created environment is registered as a JupyterLab kernel, making it instantly accessible from the notebook interface for all users without additional configuration or manual steps.

Access Everywhere

All team members with access to the persistent storage can select and use this shared kernel in their notebooks, ensuring consistency and eliminating environment drift across the team.

Update Seamlessly

Environment updates are propagated automatically to all users, ensuring everyone stays in sync with the latest dependencies, security patches, and improvements without manual intervention or downtime.

Impact

Real-World Impact

The benefits of this architecture extend far beyond technical elegance—they translate directly into measurable business value. Teams report dramatic improvements in productivity, consistency, and deployment success rates after implementing this solution.

85%

Faster Onboarding

New data scientists become productive in days, not weeks, with pre-configured environments ready to use immediately upon arrival

60%

Reduced Debugging Time

Teams spend significantly less time troubleshooting dependency issues and environment conflicts, redirecting effort to actual development

100%

Reproducibility

Complete environment consistency from development to production ensures reliable deployments and eliminates deployment surprises

0

Environment Drift

Centralized management eliminates configuration inconsistencies across team members and projects entirely

Track these metrics in your own deployment to quantify the impact and demonstrate ROI to stakeholders. Consider measuring time-to-first-commit for new hires, deployment success rates, and hours saved on environment troubleshooting.

