CONF42

# Exploring container-native simplicity in AI/ML workflows

- Shivay Lamba (@howdevelop)

## A Unique Challenge Exists

Transforming AI models from experimental notebooks into robust, production-ready deployable solutions.

80% of machine learning models never makes it to production because of the complex model deployment procedures.

# What is the biggest challenge with Machine learning packaging today?

There is no standard for packaging and versioning the various artifacts needed to reproduce an AI/ML project:

- Models in Jupyter notebooks or MLOps tools
- Datasets in data lakes, databases, or file systems
- Code in Git repositories
- Metadata (such as hyperparameters, features, and weights) scattered across different storage systems

# Why can't we use the same pipeline for ML products (MLOps) and conventional software projects (DevOps)?

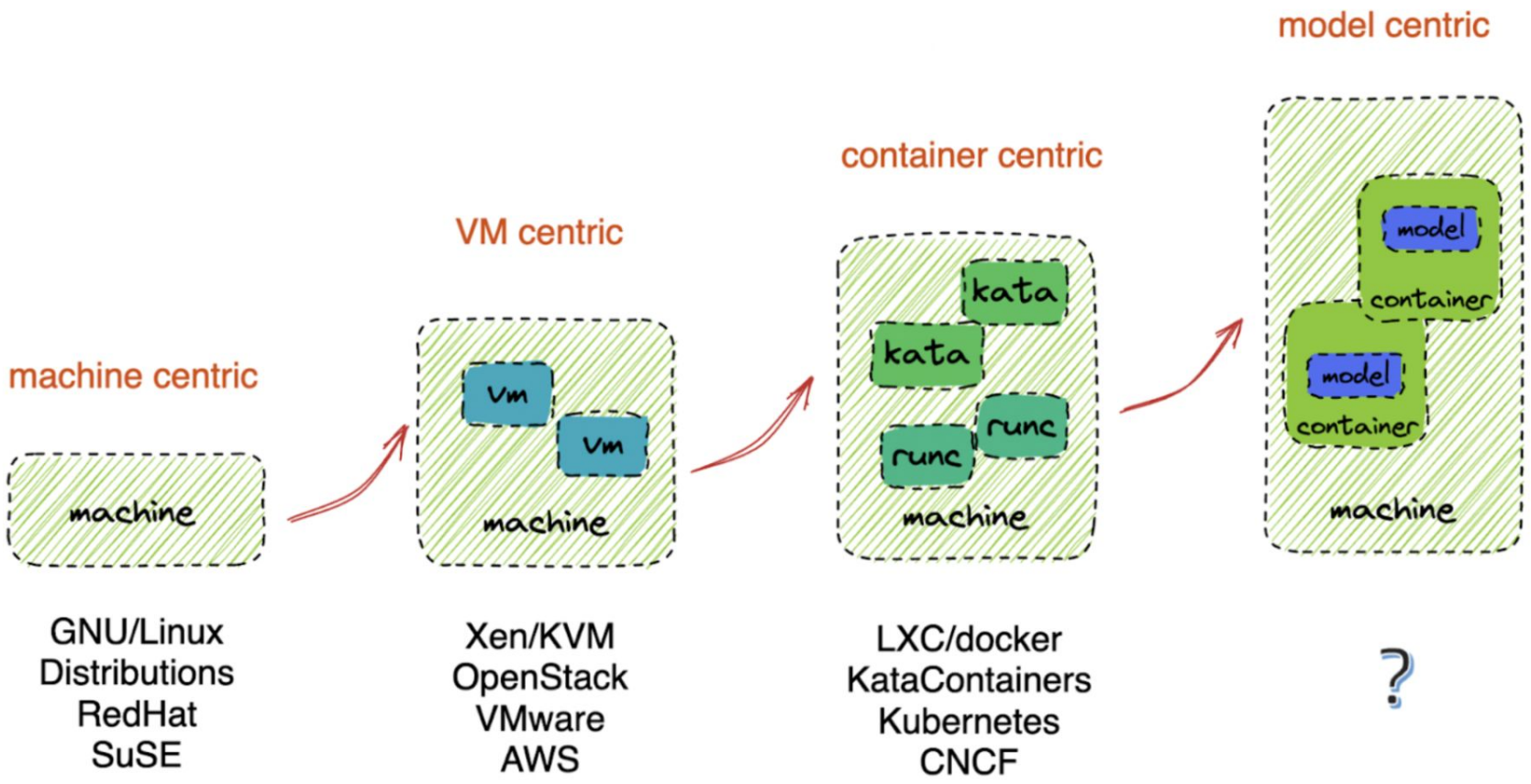
- There are three main reasons that necessitate separate pipelines for deploying ML products and conventional software engineering projects:
    - Nature of projects
    - Individual expertise
    - Size and complexity
- The above challenges have led organizations to adopt a separate MLOps pipeline. However, implementing a separate MLOps pipeline has some unique challenges of its own:
    - Data management and standardization issues
    - Security and compliance issues
    - Complex model lifecycle management

# Challenges with Traditional Separation of Pipelines and shortcomings

- Distinct workflows for:
  - Data Scientists (Jupyter Notebooks, model training)
  - DevOps Teams (deployment, infrastructure)
  - MLOps Engineers (model management)

- Increased Costs and Inefficiencies
  - Duplicate efforts
  - Manual handoffs
  - Lack of standardized versioning

- Technical Debt
  - Difficulty in reproducing experiments
  - Inconsistent deployment processes


machine centric
machine
GNU/Linux
Distributions
RedHat
SuSE
VM centric
vm
vm
machine
Xen/KVM
OpenStack
VMware
AWS
container centric
kata
kata
runc
runc
machine
LXC/docker
KataContainers
Kubernetes
CNCF
model centric
model
container
model
container
machine
?

# How KitOps helps solve this problem?

KitOps is an open-source, standards-based packaging and versioning system designed for AI/ML projects.

KitOps takes advantage of existing software standards so the tools and processes your DevOps / SRE teams use with their containerized applications, can be used with AI/ML projects.

KitOps allows AI teams to package AI/ML models, datasets, code, and metadata into what's called a ModelKit.

# Diving Deeper Into KitOps

### ModelKit

At the heart of KitOps is the ModelKit, an OCI-compliant packaging format that enables the seamless sharing of all necessary artifacts involved in the AI/ML model lifecycle. This includes datasets, code, configurations, and the models themselves.

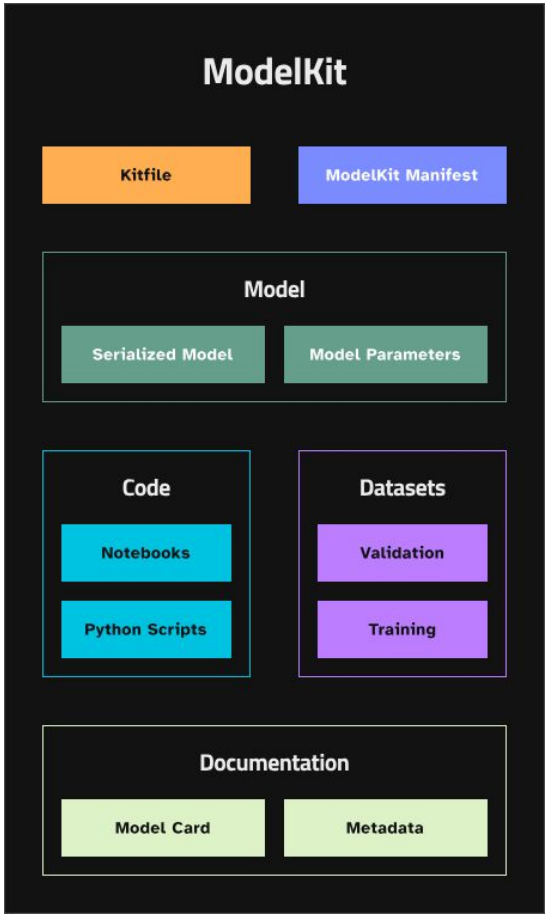
# What are ModelKits?

### Comprehensive Packaging Solution

- Versioned bundles containing:
  - Machine learning models
  - Datasets
  - Configuration
  - Dependencies

### Standardized Packaging

- Uses OCI (Open Container Initiative) standard
- Enables easy movement across environments



# Diving Deeper Into KitOps

### Kitfile

Complementing the ModelKit is the Kitfile, a YAML-based configuration file that simplifies the sharing of model, dataset, and code configurations. The Kitfile is designed with both ease of use and security in mind, ensuring that configurations can be efficiently packaged and shared without compromising on safety or governance.

# Kitfile

The Kitfile manifest for AI/ML is a YAML file designed to encapsulate all the necessary information about the package, including code, datasets, model, and their metadata.

The manifest is structured into several key sections: manifestVersion, package, code, datasets, docs, and model

```yaml
manifestVersion: 1.0
package:
  name: AIProjectName
  version: 1.2.3
  description: >-
    A brief description of the AI/ML project.
  authors: [Author Name, Contributor Name]
code:
  - path: src/
    description: Source code for the AI models.
    license: Apache-2.0
datasets:
  - name: DatasetName
    path: data/dataset.csv
    description: Description of the dataset.
    license: CC-BY-4.0
model:
    name: ModelName
    path: models/model.h5
    framework: TensorFlow
    version: 1.0
    description: Model description.
    license: Apache-2.0
```

# Diving Deeper Into KitOps

**Kit CLI**

Bringing everything together is the Kit Command Line Interface (CLI). The Kit CLI is a powerful tool that enables users to create, manage, run, and deploy ModelKits using Kitfiles. Whether you are packaging a new model for development or deploying an existing model into production, the Kit CLI provides the necessary commands and functionalities to streamline your workflow.

# Let’s look into how does a KitOps Pipeline look like

1. kit unpack - to help pull Modelkits and run them locally (Produce the components from a modelkit on the local filesystem)

2. kit pull - Retrieve modelkits from a remote registry to your local environment

---

1. Create the kitfile for your local AI project

2. kit pack - to create Modelkit from the kitfile

3. kit push - helps to copy the newly built ModelKit from the local repository to the remote repository (registry)

# Demo

Let’s look into how a simple KitOps workflow look like using KitCLI

# Automating the Machine Learning Lifecycle with KitOps and CI/CD

You can add KitOps as part of your CI/CD pipeline to automate the deployment of AI models either triggered manually or automatically when changes takes place in the model or the artifacts.

KitOps simplifies the packaging of models and their dependencies while managing version control, among other features.

**Automated Model Deployment**

- Trigger deployments manually or automatically
- Streamline model lifecycle management

# Implementation steps for the Dagger CI/CD pipeline

1. Install KitOps & Dagger.io

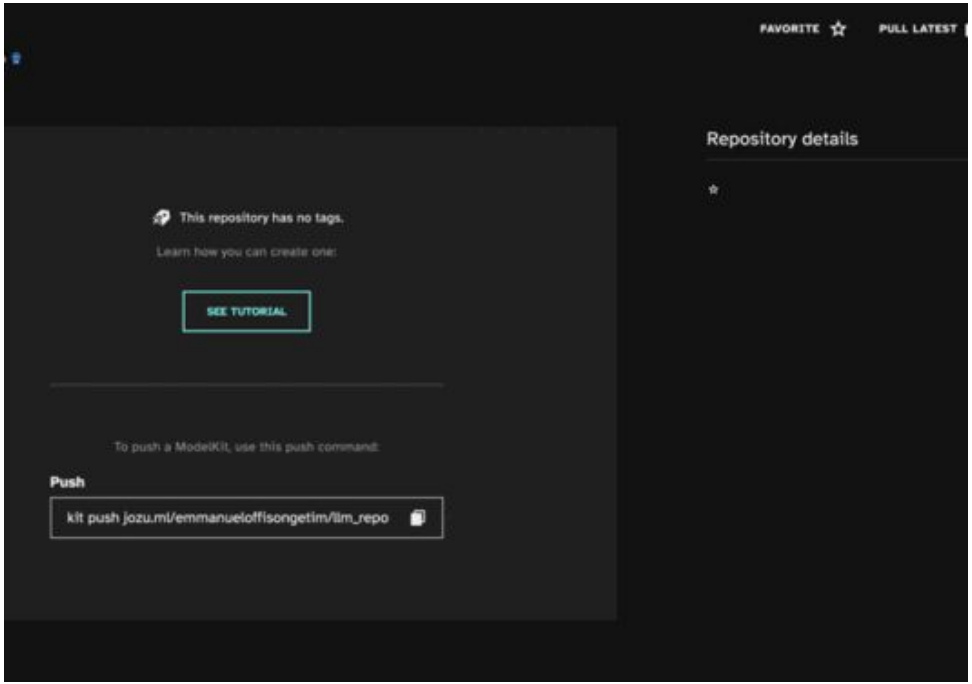
2. Create ModelKit with Kitfile

3. Initialize Dagger module, Daggerize your Kitfile

4. Define pipeline functions

5. Integrate with CI/CD (e.g., GitHub Actions)

6. Automate deployment to registry



# MLOps workflow with ModelKits

- Training and experimentation
- Package, validate, and deployment
- Inference

# Deploying ModelKits

You can create a container or Kubernetes deployment using a ModelKit

- Init Container - The init container unpacks the model reference from a ModelKit to a specific path and then exits. This makes it useful as a Kubernetes init container. This container also supports verifying signatures for containers automatically from key-based or keyless signers.

- KitCLI Container - The containerized Kit CLI can be used to tailor the running of a ModelKit because you can run any Kit CLI command. This gives you flexibility, but more manual work (the world is your oyster, but it may be hard to shuck).

# Resources

Checkout KitOps - https://kitops.ml/

KitOps Documentation - https://kitops.ml/docs/overview.html

Checkout the KitOps Dev.to Blogs - https://dev.to/kitops

# AI Model Specification

On CNCF Slack: #model-spec-discussion

https://docs.google.com/document/d/1YemmDF_0rOy3TuUNeRAONszBWtWf6MKjZx_BmVJTMYw/edit?usp=sharing

The primary target for the model spec is to make AI models first-class citizens in the infrastructure world. It describes AI models explicitly (rather than as opaque binaries) so that infrastructure components can be aware of the model details and optimize them wherever appropriate.

With a well-defined model specification, we can bring container development and deployment experience to AI models. Also, it is possible to build a cloud-native AI ecosystem based on it.

# Thanks for attending!

Join the KitOps Discord

https://discord.gg/Tapeh8agYy

You can connect with me here:

X/Twitter: @HowDevelop