Red Hat OpenShift AI

RHOAI Advanced Topics

Agenda

- Customizing Workbenches
- Pipelines
- LLM Serving
- Recommended Practices

Customizing Workbenches

Base Notebook Images

Reproducible and shareable environments for building, training and serving

Python 2.x, 3.x

jupyter

NumPy
Pandas
Matplotlib
Other Libraries

App Deps and Libs

Base Image

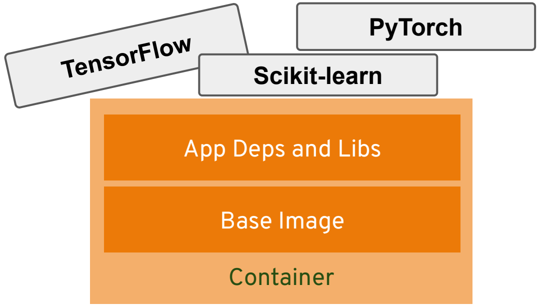
Container

Supported and maintained by

- Red Hat (e.g. Tensorflow, PyTorch, CUDA)
- partner (Anaconda, Intel)
- you (custom notebooks)

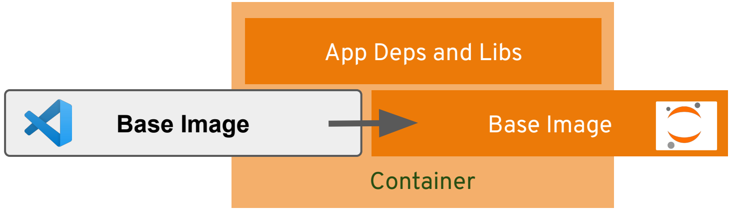
Customizing the workbench

Adding packages on top of a good image



Just remember that they are removed when restarting the workbench*

Creating your own custom image with all dependencies you need



You can now version and maintain it according to your preferences

* This is on purpose so that you can un-mess-up your environment easily if you get into dependency issues.

Notebook image

Image selection *

Select one

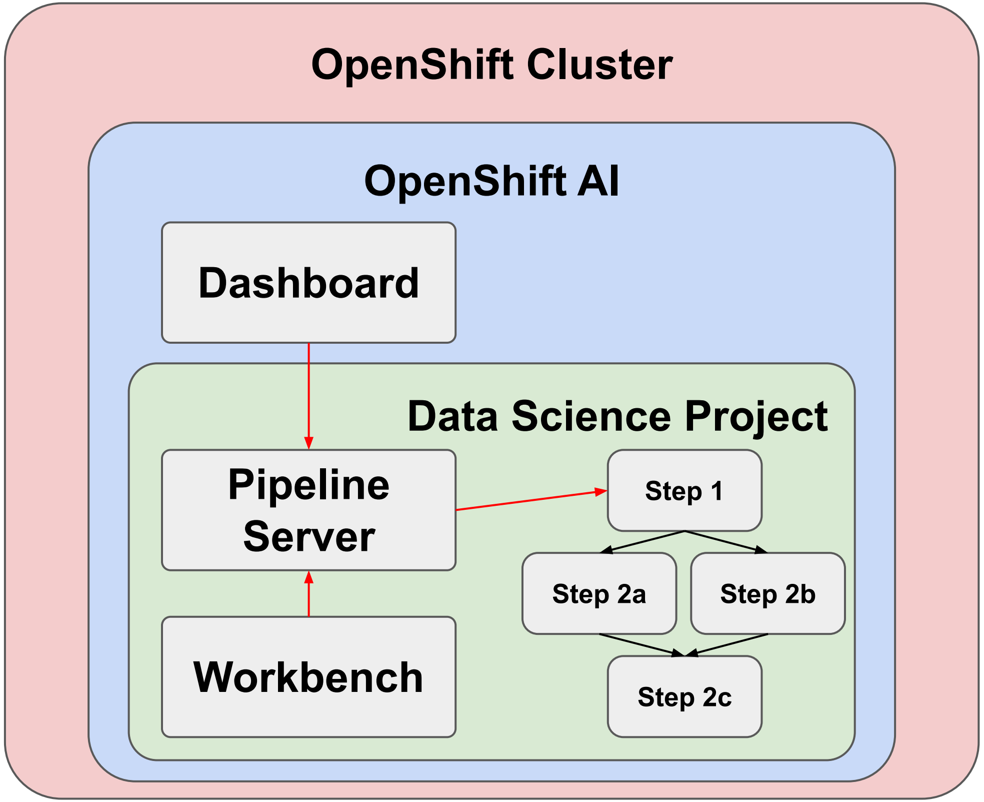
- Minimal Python
- Standard Data Science
- CUDA
- PyTorch
- TensorFlow
- TrustyAI
- HabanaAI
  Python v3.8, Habana v1.10
- code-server
  Python v3.9
- CUSTOM: VS Code

Pipelines

Data Science Pipeline Concepts

- **Pipeline** - is a workflow definition containing the steps and their input and output artifacts.
- **Run** - is a single execution of a pipeline whereas a recurring run is a scheduled, repeated execution of a pipeline.
- **Step** - is a self-contained pipeline component that represents an execution stage in the pipeline.

Pipelines

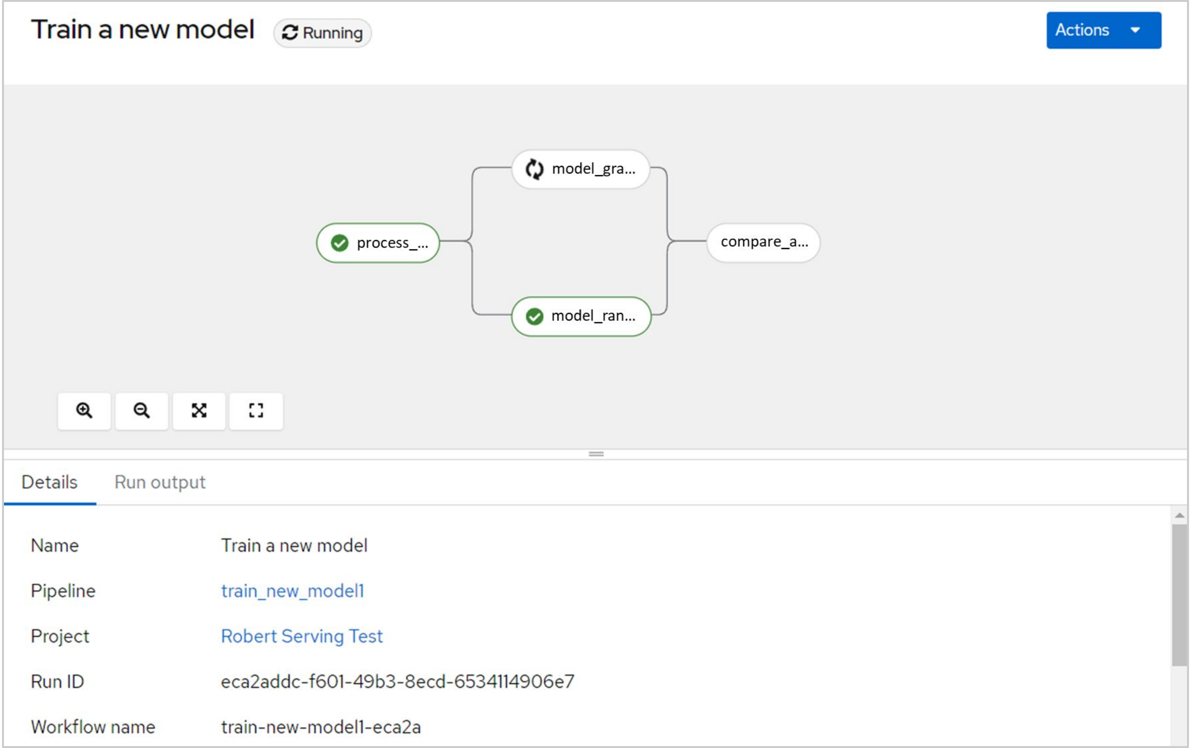


A **Pipeline** is a **sequence of steps** that will be executed in defined order.

Each **Step** will execute some **code** that will be run into a single **container** using a defined **runtime**.

A **Pipeline** can be **graphically created** and submitted through a **Workbench** or through the **Dashboard** as a **YAML** file.

A submitted **Pipeline** is **executed** by the **Pipeline Server**, which will create and delete the steps containers.

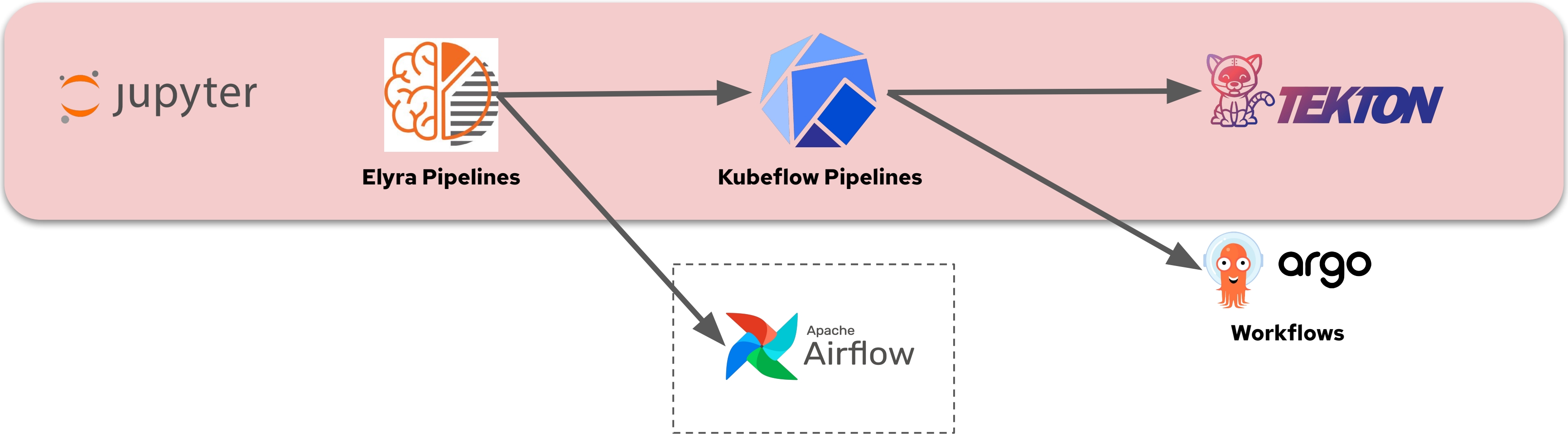

Train a new model
Running
Actions
model_gra...
process_...
compare_a...
model_ran...
Details
Run output
Name
Train a new model
Pipeline
train_new_model1
Project
Robert Serving Test
Run ID
eca2addc-f601-49b3-8ecd-6534114906e7
Workflow name
train-new-model1-eca2a

Data Science Pipelines Technologies

- **Elyra Pipelines** – A JupyterLab extension, which provides a visual editor for creating data science pipelines
- **Kubeflow Pipelines** – Specialized data science pipelines engine which can translate an Elyra visual pipeline execution (or Kubeflow SDK calls) into a Tekton pipeline running in OpenShift.
- **OpenShift Pipelines (based on Tekton, soon Argo Workflows)** – It executes each step in the pipeline as an individual container and ensures that each container gets allocated the resources (GPU, CPU, memory)
- **OpenShift GitOps with ArgoCD for automated model deployments** – GitOps can be leveraged to push models to other OpenShift instances (including OpenShift Edge devices)

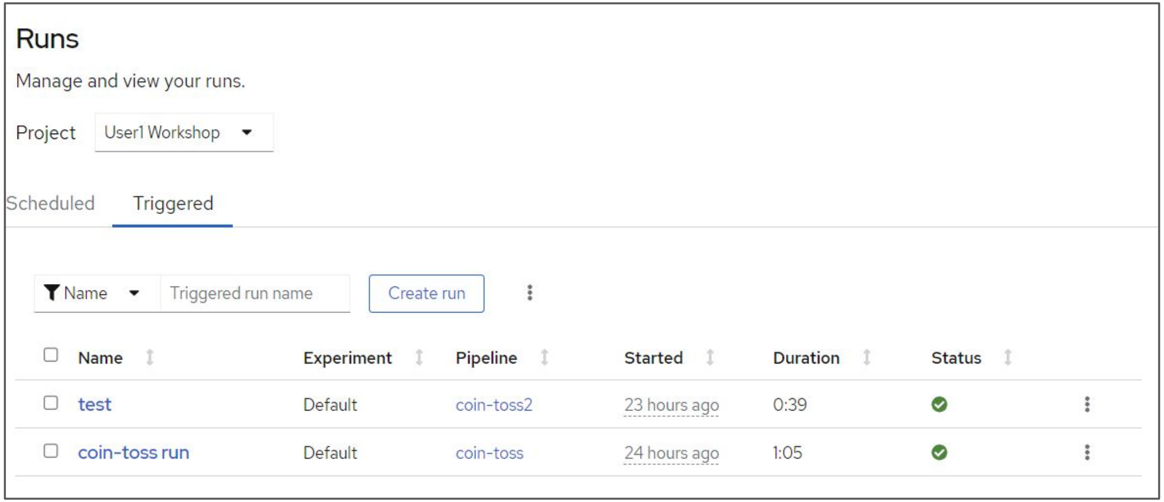
Kubeflow Pipelines Ecosystem

IDE | Editor | Orchestrator | Runtime



Experiment tracking

Pipeline runs can be used as experiments, and the run view can be used to track those experiments.



Artifacts

Artifacts are stored in S3, it stores things such as:

- Logs
- Passed data
- Code and dependent files needed to run the pipeline
- (Soon) results

Passing Data

A few ways to pass data between steps:

- Small data:
  - Through parameters
- Large data:
  - Through volumes
  - Tekton Workspaces
  - Object storage

Automation

We see a pattern of pushing pipelines into production instead of models.

Why?

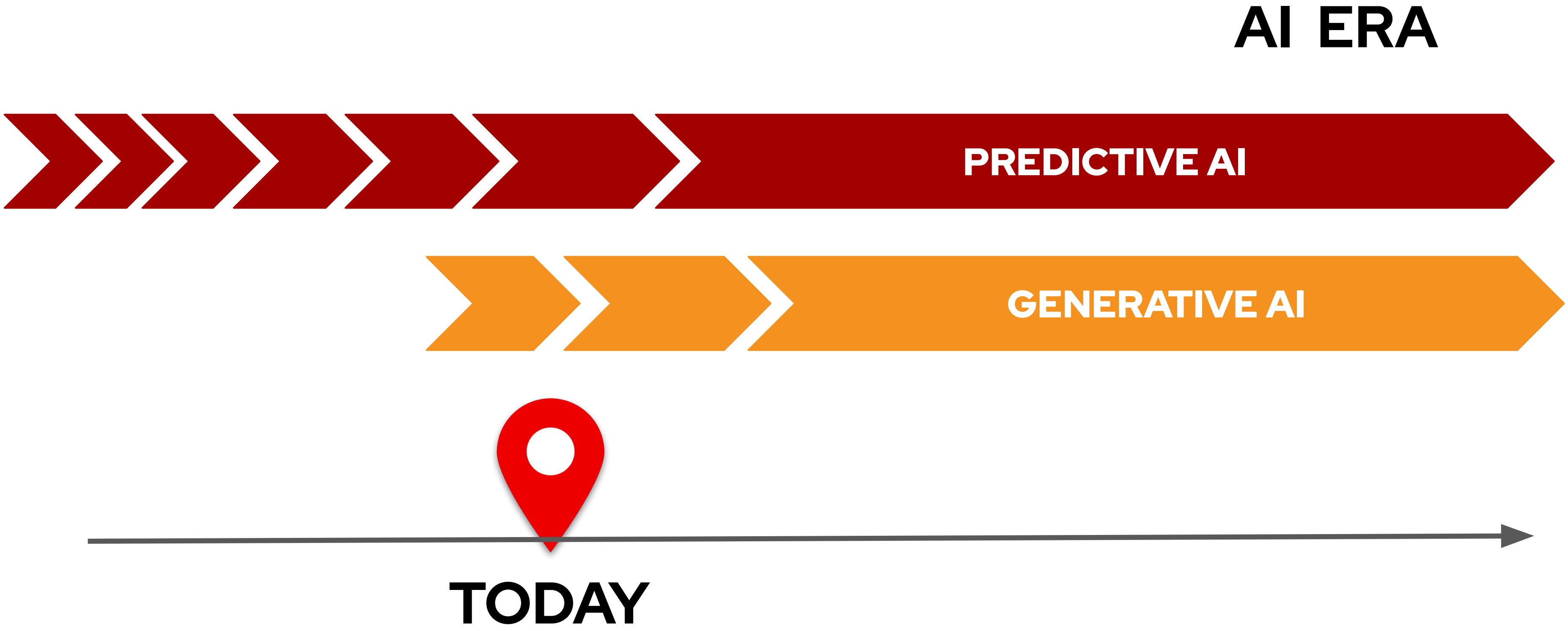
- Ability to automatically retrain a model on new data and get it served.

How?

- Data Science Pipelines as CI
- ArgoCD/Tekton as CD

LLM Serving

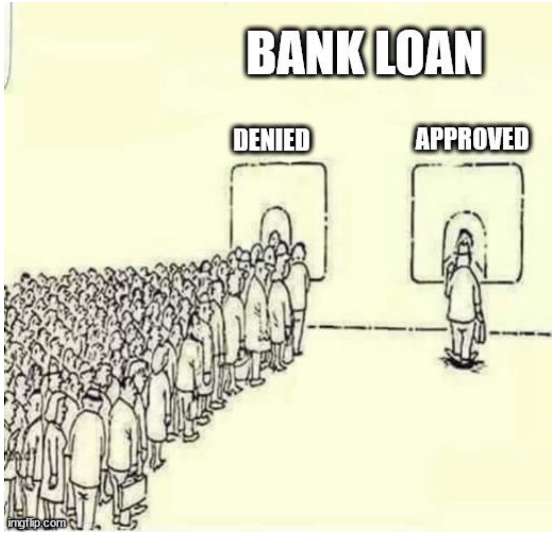
Generative vs Predictive models



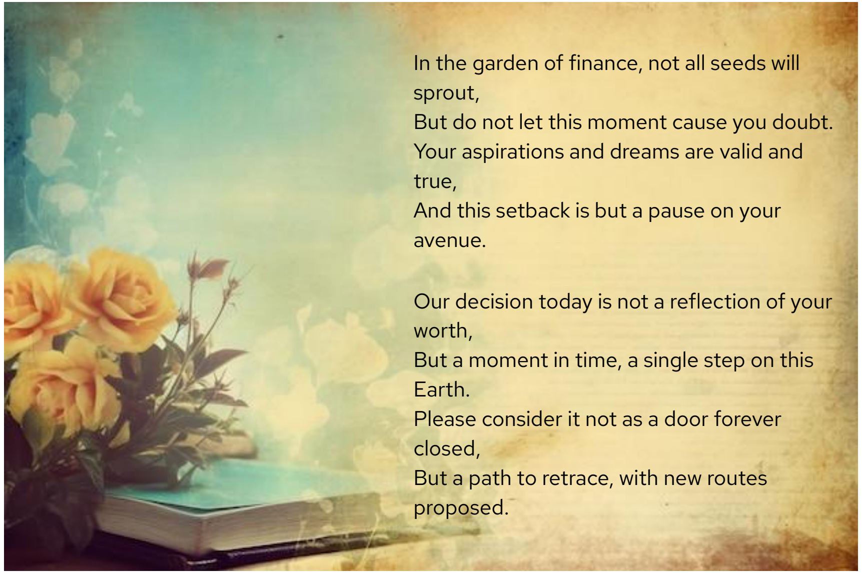
Generative vs Predictive models

Bank is not only predictive

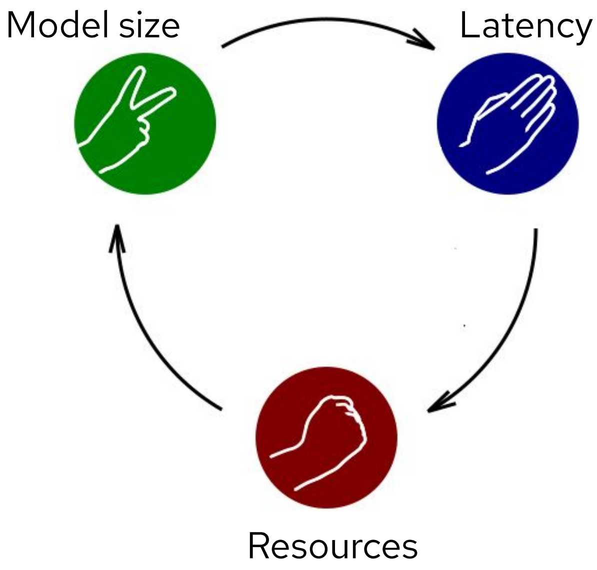
Predictive



Generative

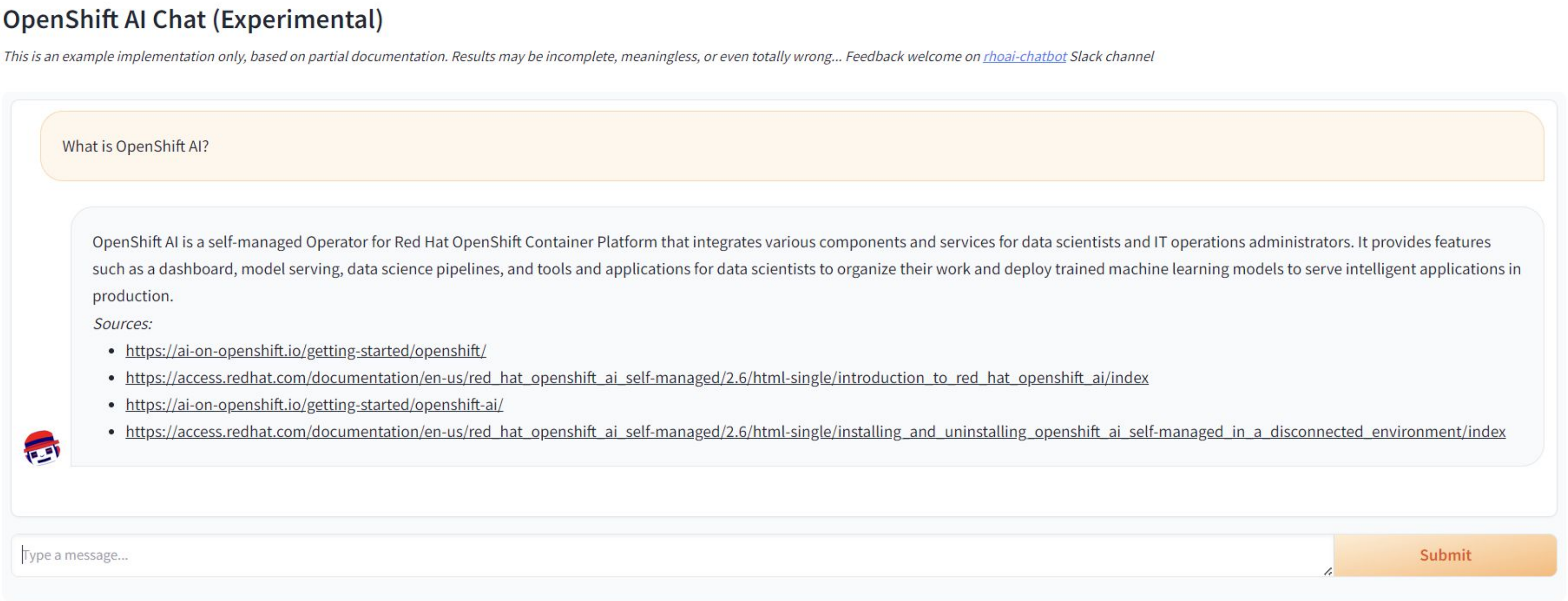


What to think about when serving an LLM



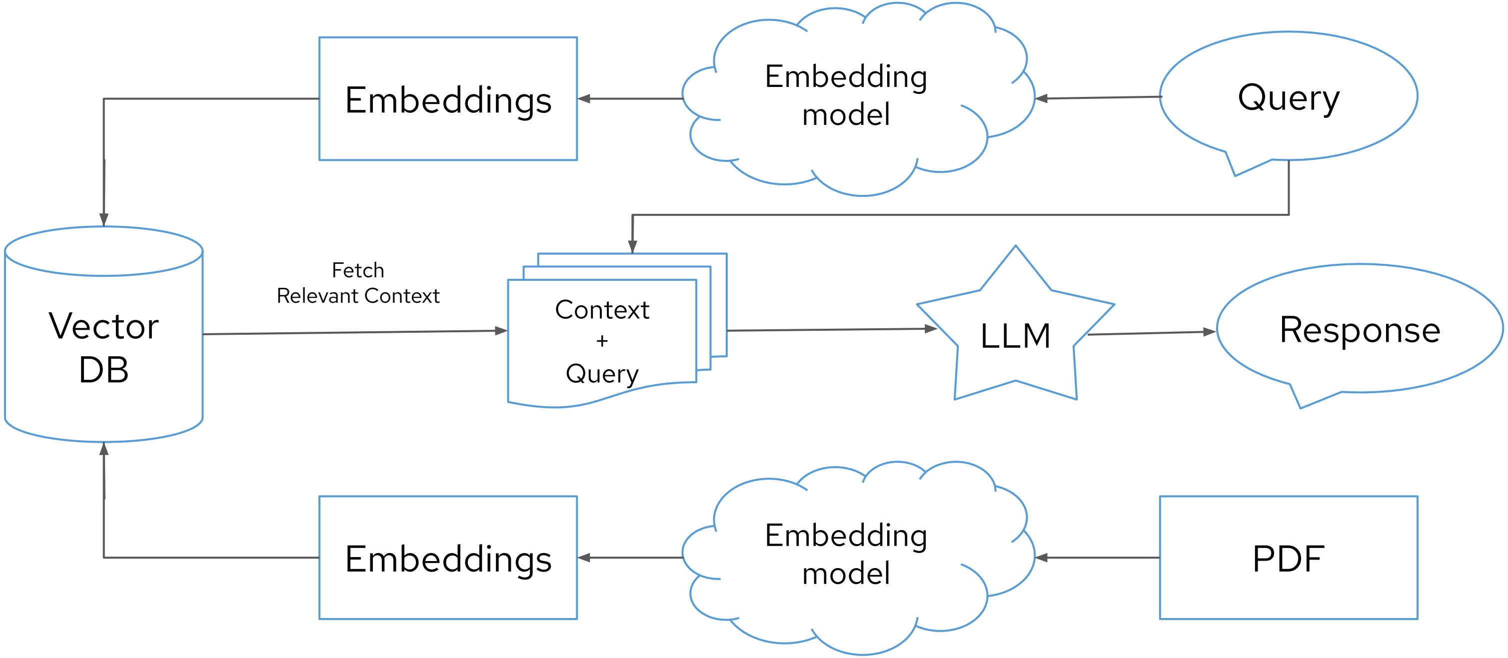
Red Hat

RAG Example

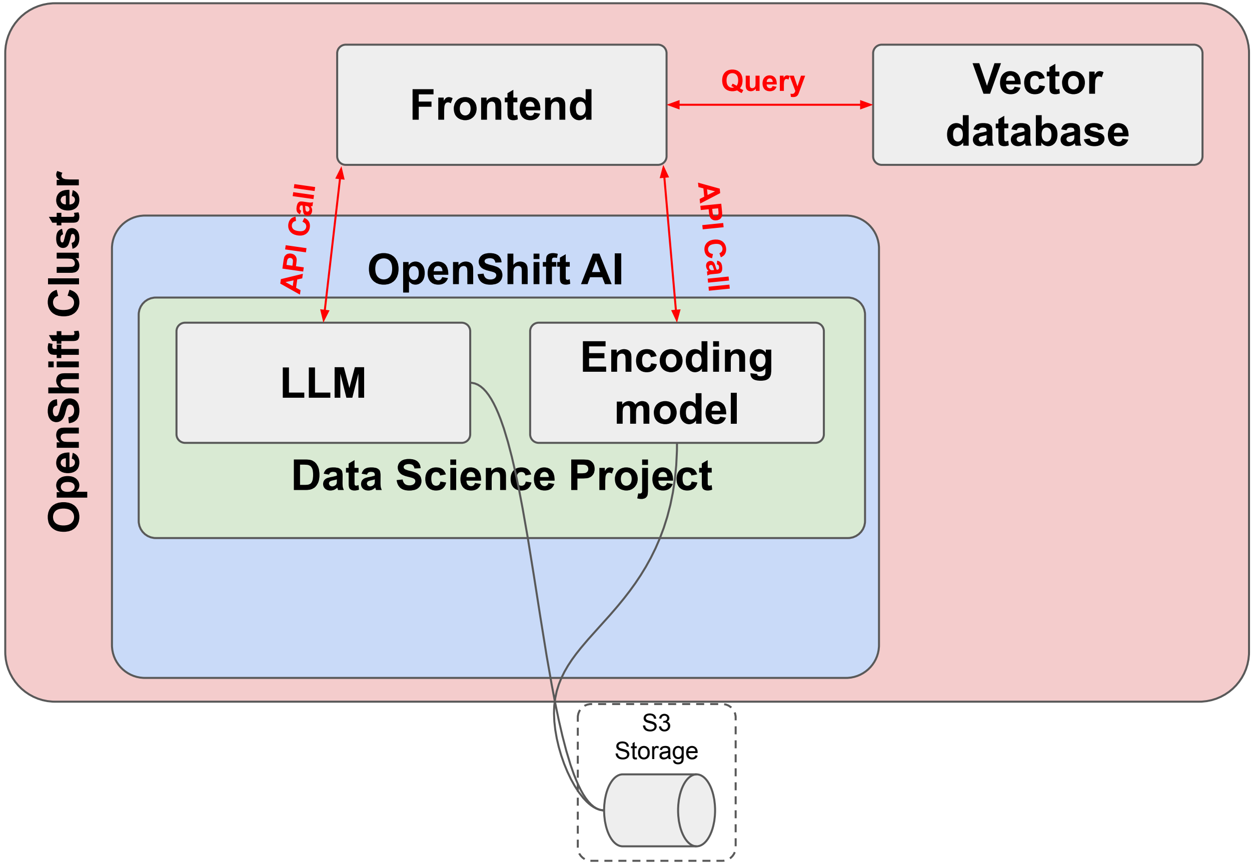


https://kbchat-llm-hosting.apps.rhods-internal.61tk.p1.openshiftapps.com/

RAG Example


Embeddings
Embedding model
Query
Vector DB
Fetch Relevant Context
Context + Query
LLM
Response
Embeddings
Embedding model
PDF

RAG Example


OpenShift Cluster
Frontend
Query
Vector database
API Call
API Call
OpenShift AI
LLM
Encoding model
Data Science Project
S3 Storage

Recommended Practices

Some Recommended practices

- These are general recommended practices and are not all specific to OpenShift AI
- There can be exceptions and situations where other factors can overrule these recommendations

Version Control (code)

- Ensure that you use appropriate tools to version control and collaborate on code
- git-based version control is the standard
- All text-based code (*.ipynb, *.py, requirements.txt, *.pipeline, *.yaml) should be tracked with version control

Version Control (data)

- Ensure that data used for training is tracked in such a way that model training is a reproducible exercise.
  - code
  - container image
  - data
  - parameters
- Multiple approaches can be used to achieve this
  - DVC
  - S3-storage with versioning scheme
  - etc...

"Off-Cluster" storage

- If all data/code/artifacts is stored off-cluster, it makes it easier to leverage the same code/data into multiple workbenches/projects/clusters.
  - git repo
  - s3 storage (or other)

Pinning versions

- All else being equal, ensure that you are very explicit in the exact versions of the packages you are using
- Good:
  - `lxml==2.3.4`
- Less stable over time:
  - `lxml`
  - `lxml>=2.2.0`
  - `lxml>=2.2.0,<2.3.0`
- See https://pip.pypa.io/en/stable/topics/repeatable-installs/
- pip freeze helps with pinning sub-dependencies

Use Custom Workbench/Runtime Images

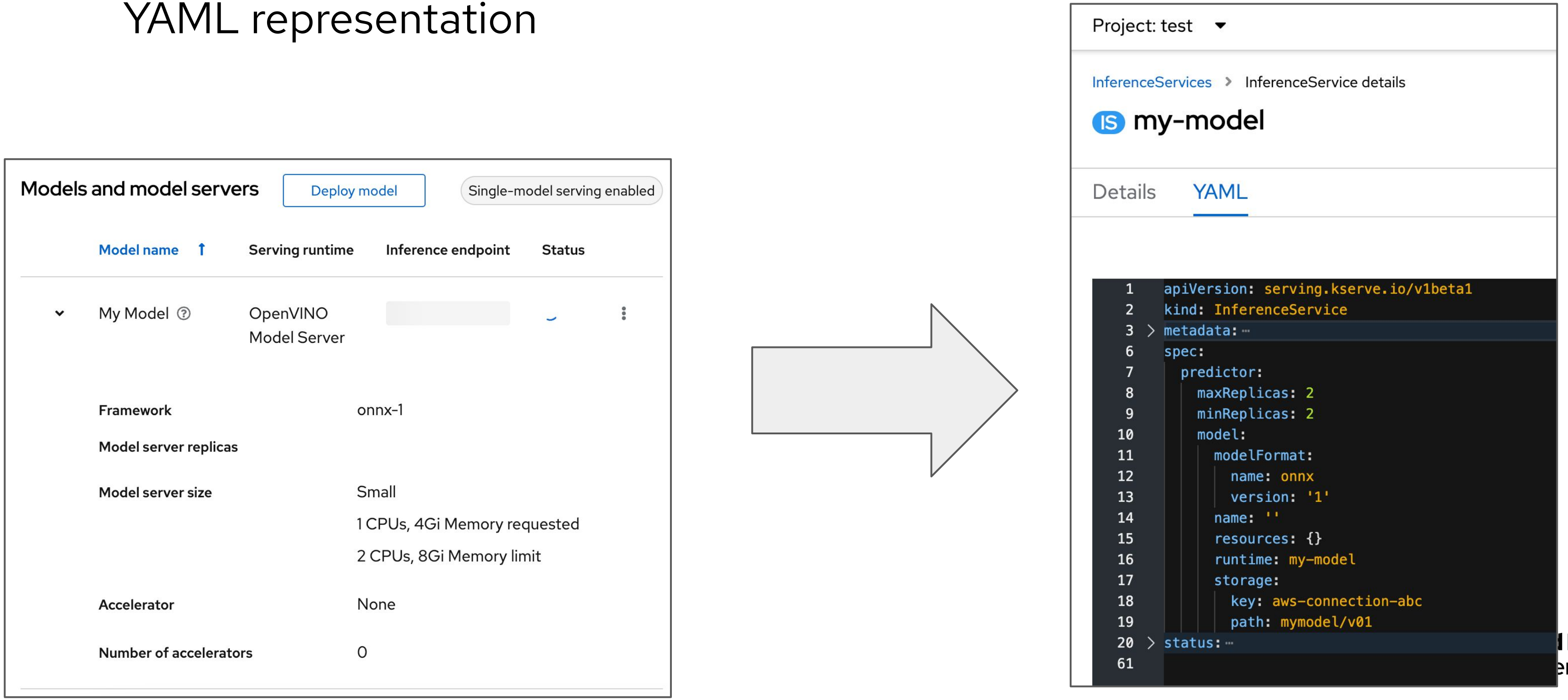
- The default images provided as part of RHOAI
  - are starting points
  - are supported
  - will change over time
- The pace of change in these images may be faster or slower than what you need
- Their content may not be 100% of what you need
- As your projects mature, you may require more stability in these environments
- Custom versions of image give you complete and fine-grained control

Prototyping vs Production

- During the prototyping phase, velocity is key:
  - try out different things
  - explore, try, etc...
  - "inner" loop
  - GUI-driven
- When things get closer to production, stability will become key:
  - reproducibility
  - validation tests (confirm assumptions hold true)
  - stability
  - code-driven

GitOps Principles

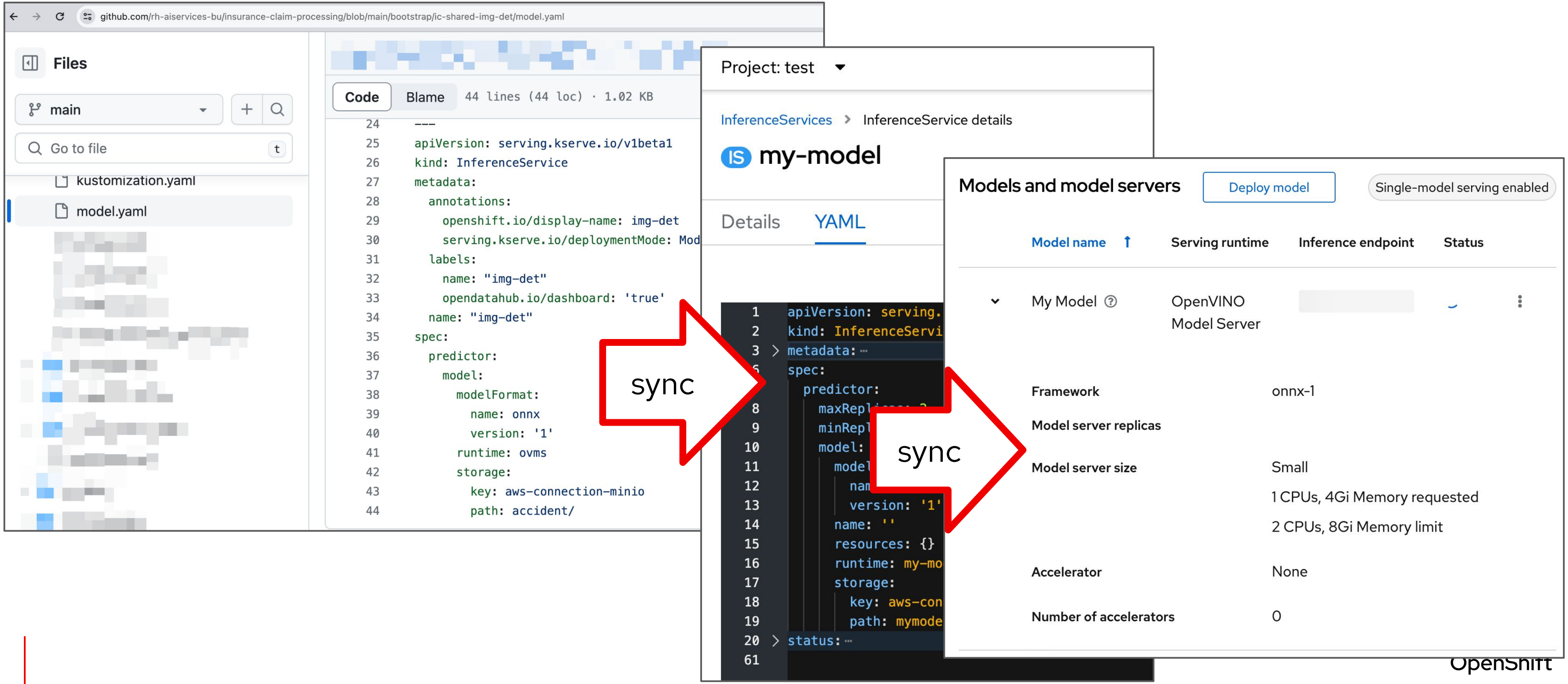
- Every artifacts created by the RHOAI user interface has a YAML representation



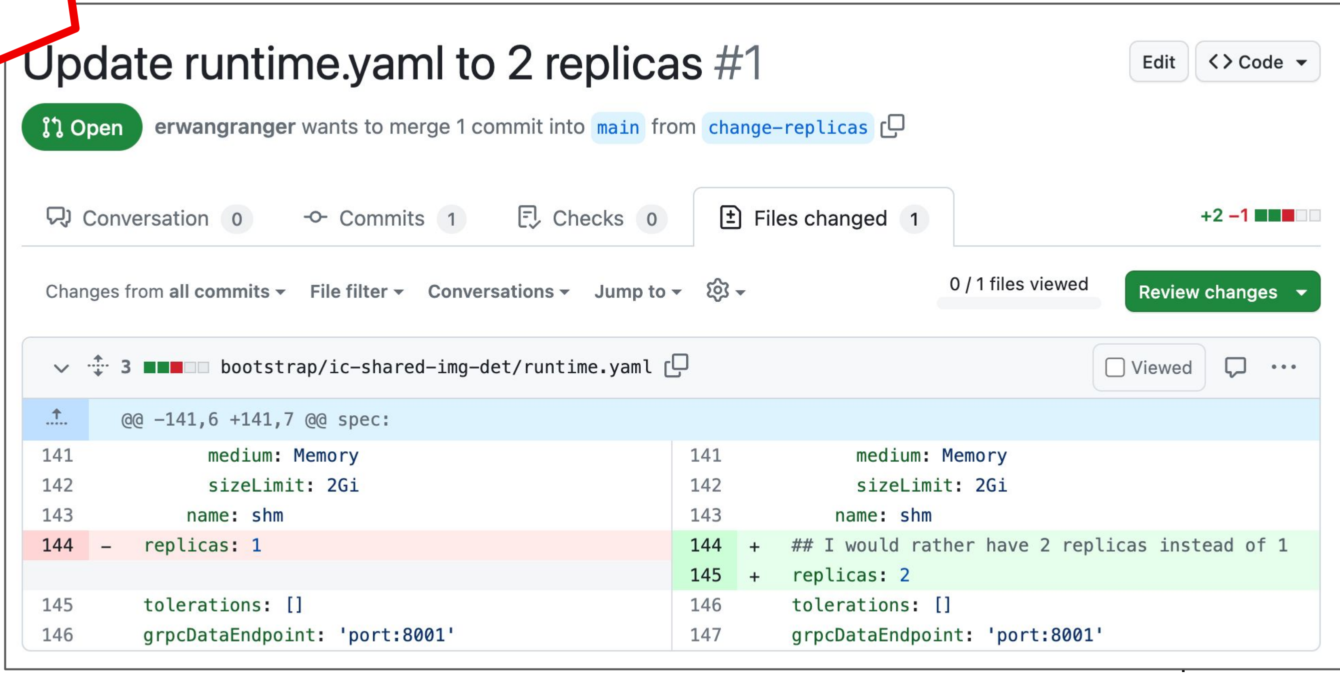
GitOps Principles

- This can be very useful:
- Large scale:
  - Script the creation of a large number of artifacts
  - (instead of having to manually create them in the GUI)
- Maintain State:
  - GitOps Principles

GitOps Principles



GitOps Principles



GitOps Principles

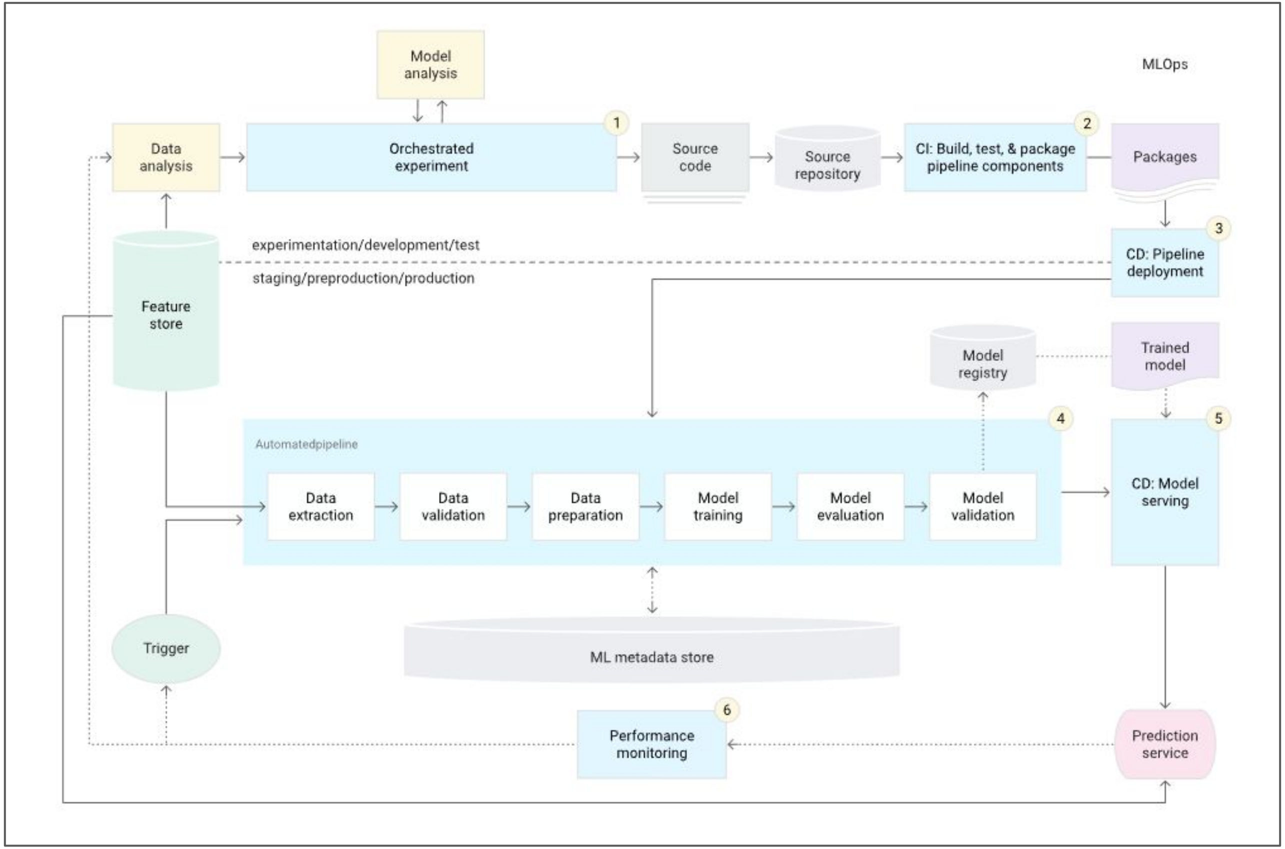
- Actual examples are beyond the scope of this training
- Main Principle:
  - instead of relying on evolving user interactions (imperative)
  - rely on a declarative description of state
  - to ensure environment matches with declared state

- Be aware of these possibilities, and leverage them if/when appropriate

MLOps Mindset

- Assume that every action you take will have to be repeated
- And that it will have to yield the same result
- Codify your assumptions, then create checks that ensure assumptions remain unchanged
    - "because the data is normally distributed, I will…"
- Move from an "artisan" mindset to a "factory" mindset
- Invest in automation
- Build it in such a way that it becomes a useful tool, and not an overhead

MLOps Flow



Red Hat OpenShift

Your turn

- General discussion:
    - What other recommended practices do you follow internally?
    - How do you share/encourage/enforce these practices?

Roadmap

Link here

End of section

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat