

# 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

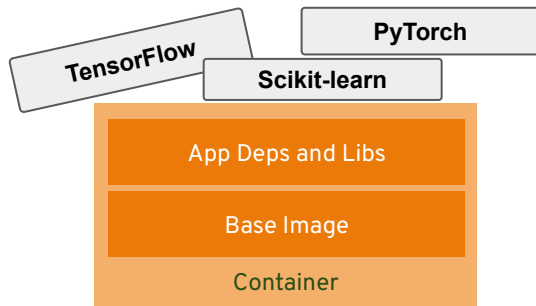


Supported and maintained by

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

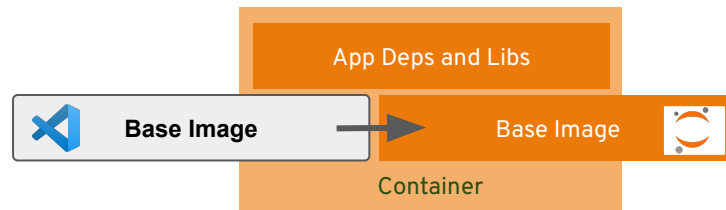
## Customizing the workbench

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

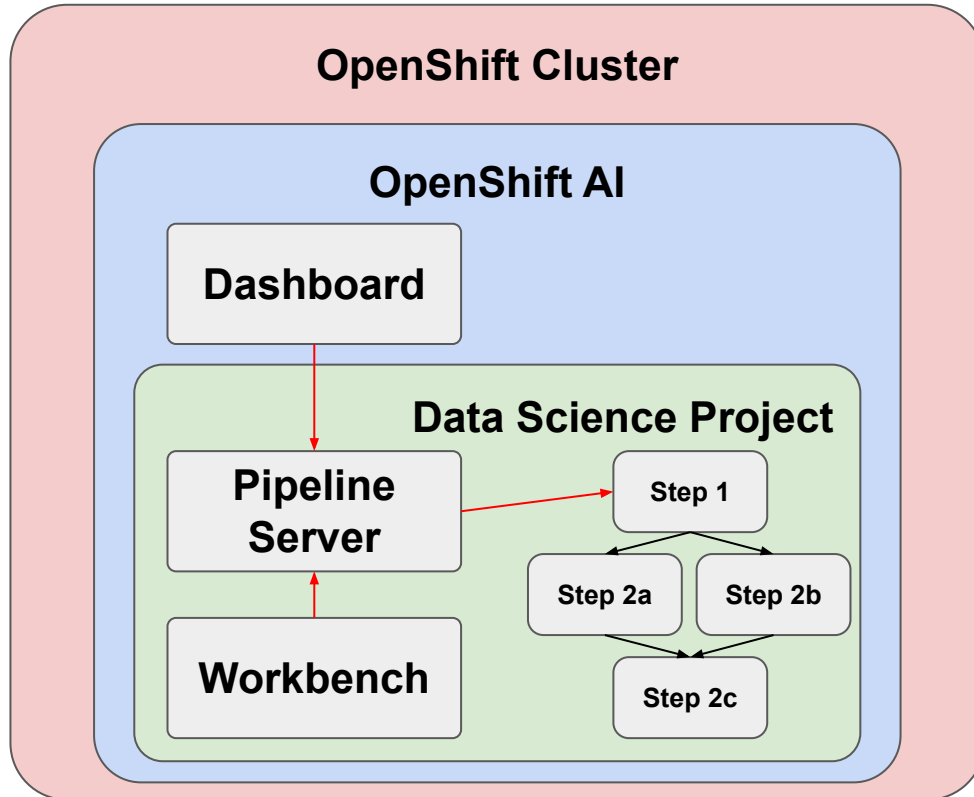
# Pipelines

# 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

```

graph LR
    A[process_...] --> B[model_gra...]
    A --> C[model_ran...]
    B --> D[compare_a...]
    C --> D
  
```

🔍

🔍

✖

🔗

Details

Run output

|               |                                      |
|---------------|--------------------------------------|
| Name          | Train a new model                    |
| Pipeline      | <a href="#">train_new_model1</a>     |
| Project       | <a href="#">Robert Serving Test</a>  |
| 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)

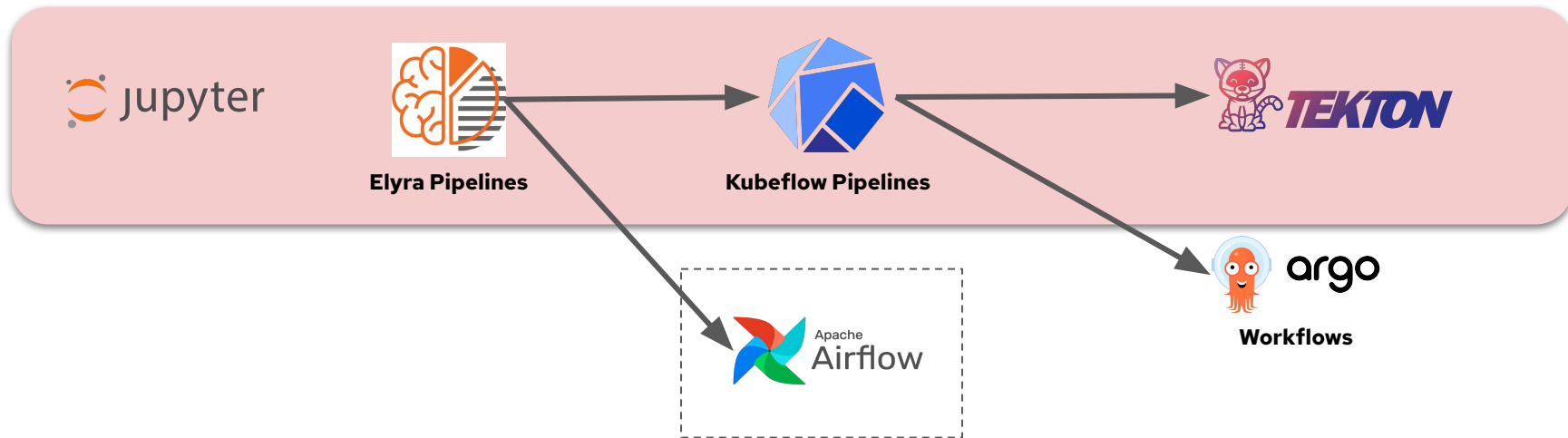
# Kubeflow Pipelines Ecosystem

IDE

Editor

Orchestrator

Runtime



# Experiment tracking

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

## Runs

Manage and view your runs.

Project User1 Workshop ▾

Scheduled Triggered

▼ Name ▾ Triggered run name Create run ⋮

| <input type="checkbox"/> Name ↑                        | Experiment ↑ | Pipeline ↑                 | Started ↑                    | Duration ↑ | Status ↑ | ⋮ |
|--|--------------|----------------------------|------------------------------|------------|----------|---|
| <input type="checkbox"/> <a href="#">test</a>          | Default      | <a href="#">coin-toss2</a> | <a href="#">23 hours ago</a> | 0:39       | ✓        | ⋮ |
| <input type="checkbox"/> <a href="#">coin-toss run</a> | Default      | <a href="#">coin-toss</a>  | <a href="#">24 hours ago</a> | 1:05       | ✓        | ⋮ |

# 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

|                          |                                      |
|--------------------------|--------------------------------------|
| <                        | pipeline-artifacts / artifacts       |
| <input type="checkbox"/> | ▲ Name                               |
| <input type="checkbox"/> | conditional-execution-pipeline-2e524 |
| <input type="checkbox"/> | conditional-execution-pipeline-50352 |

# Passing Data

A few ways to pass data between steps:

- Small data:
  - Through parameters
- Large data:
  - Through volumes
  - 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

**AI ERA**



**PREDICTIVE AI**

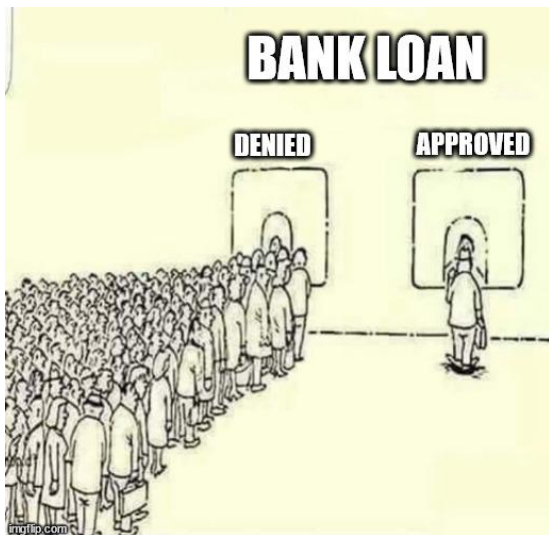
**GENERATIVE AI**

**TODAY**

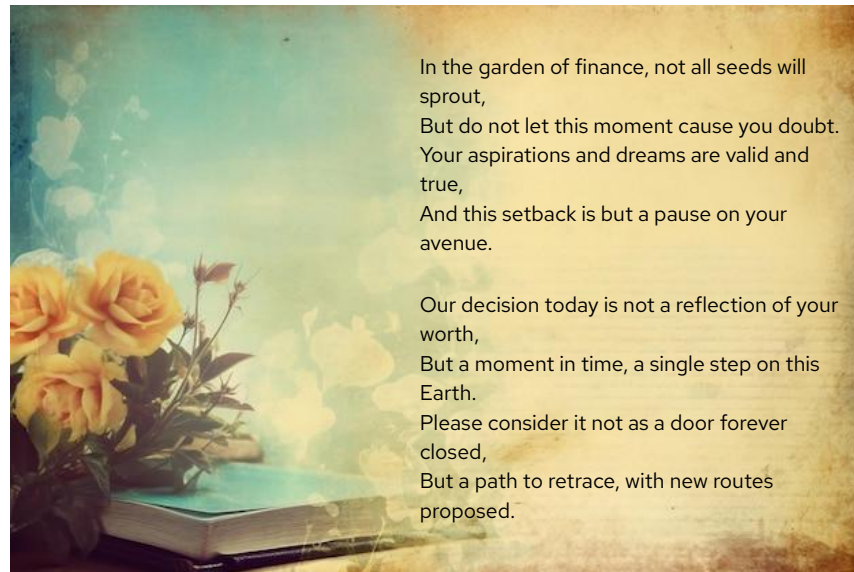
# Generative vs Predictive models

Bank is not only predictive

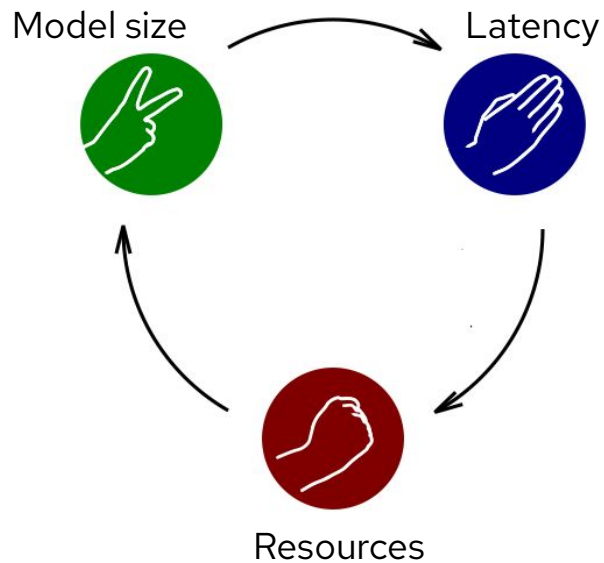
## Predictive



## Generative



## What to think about when serving an LLM



# RAG Example

## OpenShift AI Chat (Experimental)

*This is an example implementation only, based on partial documentation. Results may be incomplete, meaningless, or even totally wrong... Feedback welcome on [rhoai-chatbot](#) Slack channel*

What is OpenShift AI?

OpenShift AI is a self-managed Operator for Red Hat OpenShift Container Platform that integrates various components and services for data scientists and IT operations administrators. It provides features such as a dashboard, model serving, data science pipelines, and tools and applications for data scientists to organize their work and deploy trained machine learning models to serve intelligent applications in production.

Sources:

- <https://ai-on-openshift.io/getting-started/openshift/>
- [https://access.redhat.com/documentation/en-us/red\\_hat\\_openshift\\_ai\\_self-managed/2.6/html-single/introduction\\_to\\_red\\_hat\\_openshift\\_ai/index](https://access.redhat.com/documentation/en-us/red_hat_openshift_ai_self-managed/2.6/html-single/introduction_to_red_hat_openshift_ai/index)
- <https://ai-on-openshift.io/getting-started/openshift-ai/>
- [https://access.redhat.com/documentation/en-us/red\\_hat\\_openshift\\_ai\\_self-managed/2.6/html-single/installing\\_and\\_uninstalling\\_openshift\\_ai\\_self-managed\\_in\\_a\\_disconnected\\_environment/index](https://access.redhat.com/documentation/en-us/red_hat_openshift_ai_self-managed/2.6/html-single/installing_and_uninstalling_openshift_ai_self-managed_in_a_disconnected_environment/index)

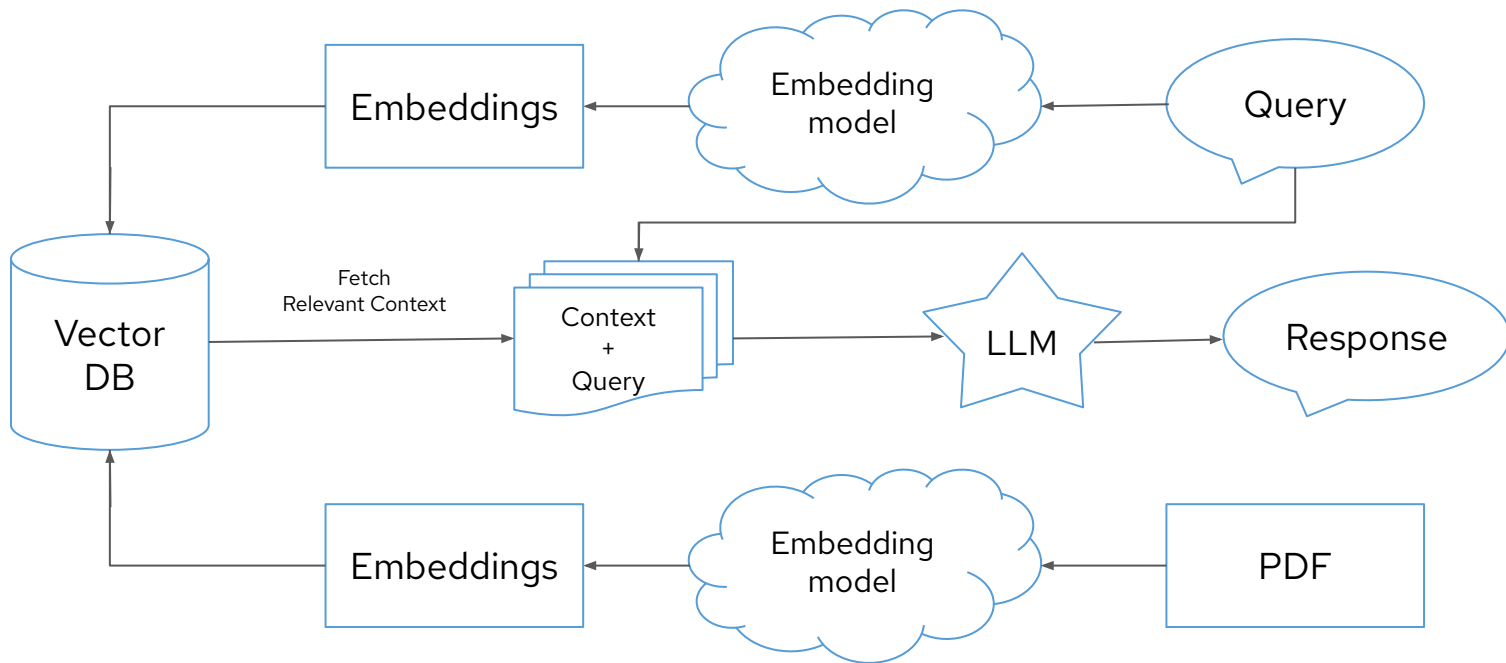


Type a message...

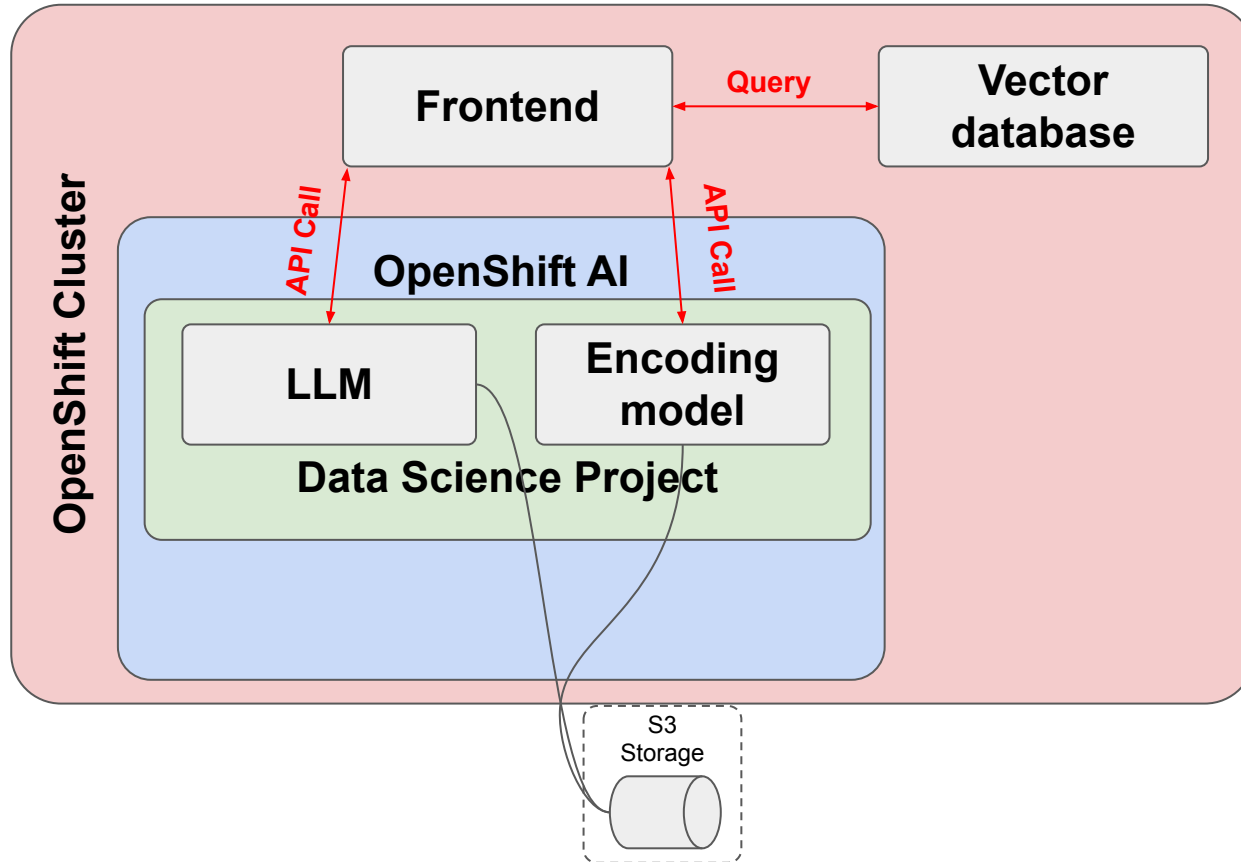
Submit

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

# RAG Example



# RAG Example



# 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

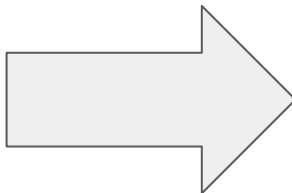
- During the prototyping phase, velocity is key:
  - try out different things
  - explore, try, etc...
  - "inner" loop
  - GUI-driven
- When things get closer to production, stability will become key:
  - reproducibility
  - validation tests (confirm assumptions hold true)
  - stability
  - code-driven

## GitOps Principles

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

Models and model servers Deploy model Single-model serving enabled

| Model name             | Serving runtime              | Inference endpoint | Status |
|------------------------|------------------------------|--------------------|--------|
| My Model               | OpenVINO Model Server        |                    |        |
| Framework              | onnx-1                       |                    |        |
| Model server replicas  |                              |                    |        |
| Model server size      | Small                        |                    |        |
|                        | 1 CPUs, 4Gi Memory requested |                    |        |
|                        | 2 CPUs, 8Gi Memory limit     |                    |        |
| Accelerator            | None                         |                    |        |
| Number of accelerators | 0                            |                    |        |



Project: test

InferenceServices > InferenceService details

**my-model**

Details YAML

```
1  apiVersion: serving.kserve.io/v1beta1
2  kind: InferenceService
3  > metadata: ...
6  spec:
7    predictor:
8      maxReplicas: 2
9      minReplicas: 2
10     model:
11       modelFormat:
12         name: onnx
13         version: '1'
14       name: ''
15       resources: {}
16       runtime: my-model
17       storage:
18         key: aws-connection-abc
19         path: mymodel/v01
20 > status: ...
61
```



## 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

The diagram illustrates the GitOps workflow for deploying machine learning models to OpenShift. It consists of three main components:

- GitHub Repository:** A repository containing the source code and configuration files. The file `model.yaml` is highlighted.
- Kubernetes Manifest (model.yaml):** A YAML file defining the InferenceService. The manifest includes the following configuration:
 

```

24 ---
25 apiVersion: serving.kserve.io/v1beta1
26 kind: InferenceService
27 metadata:
28   annotations:
29     openshift.io/display-name: img-det
30     serving.kserve.io/deploymentMode: ModelServer
31   labels:
32     name: "img-det"
33     opendatahub.io/dashboard: "true"
34   name: "img-det"
35 spec:
36   predictor:
37     model:
38       modelFormat:
39         name: onnx
40         version: '1'
41       runtime: ovms
42       storage:
43         key: aws-connection-minio
44         path: accident/
      
```
- OpenShift Console:** The OpenShift console displays the deployment details for the `my-model` InferenceService. The console shows the following configuration:
 

| Model name | Serving runtime       | Inference endpoint | Status  |
|------------|-----------------------|--------------------|---------|
| My Model   | OpenVINO Model Server |                    | Running |

 The console also displays the following configuration details:
 

| Property               | Value                        |
|------------------------|------------------------------|
| Framework              | onnx-1                       |
| Model server replicas  | 1                            |
| Model server size      | Small                        |
|                        | 1 CPUs, 4Gi Memory requested |
|                        | 2 CPUs, 8Gi Memory limit     |
| Accelerator            | None                         |
| Number of accelerators | 0                            |

Red arrows labeled **sync** indicate the flow of updates from the GitHub repository to the Kubernetes manifest and then to the OpenShift console.

## Model server replicas

Number of model server replicas to deploy <sup>?</sup>

-

2

+

# GitOps Principles

## Update runtime.yaml to 2 replicas #1

Edit

<> Code

Open

erwangranger wants to merge 1 commit into `main` from `change-replicas`

Conversation 0

Commits 1

Checks 0

Files changed 1

+2 -1

Changes from all commits File filter Conversations Jump to

0 / 1 files viewed

Review changes

bootstrap/ic-shared-img-det/runtime.yaml

Viewed

@@ -141,6 +141,7 @@ spec:

```
141         medium: Memory
142         sizeLimit: 2Gi
143         name: shm
144 -       replicas: 1
```

```
145     tolerations: []
146     grpcDataEndpoint: 'port:8001'
```

```
141         medium: Memory
142         sizeLimit: 2Gi
143         name: shm
```

```
144 +   ## I would rather have 2 replicas instead of 1
145 +   replicas: 2
```

```
146     tolerations: []
147     grpcDataEndpoint: 'port:8001'
```

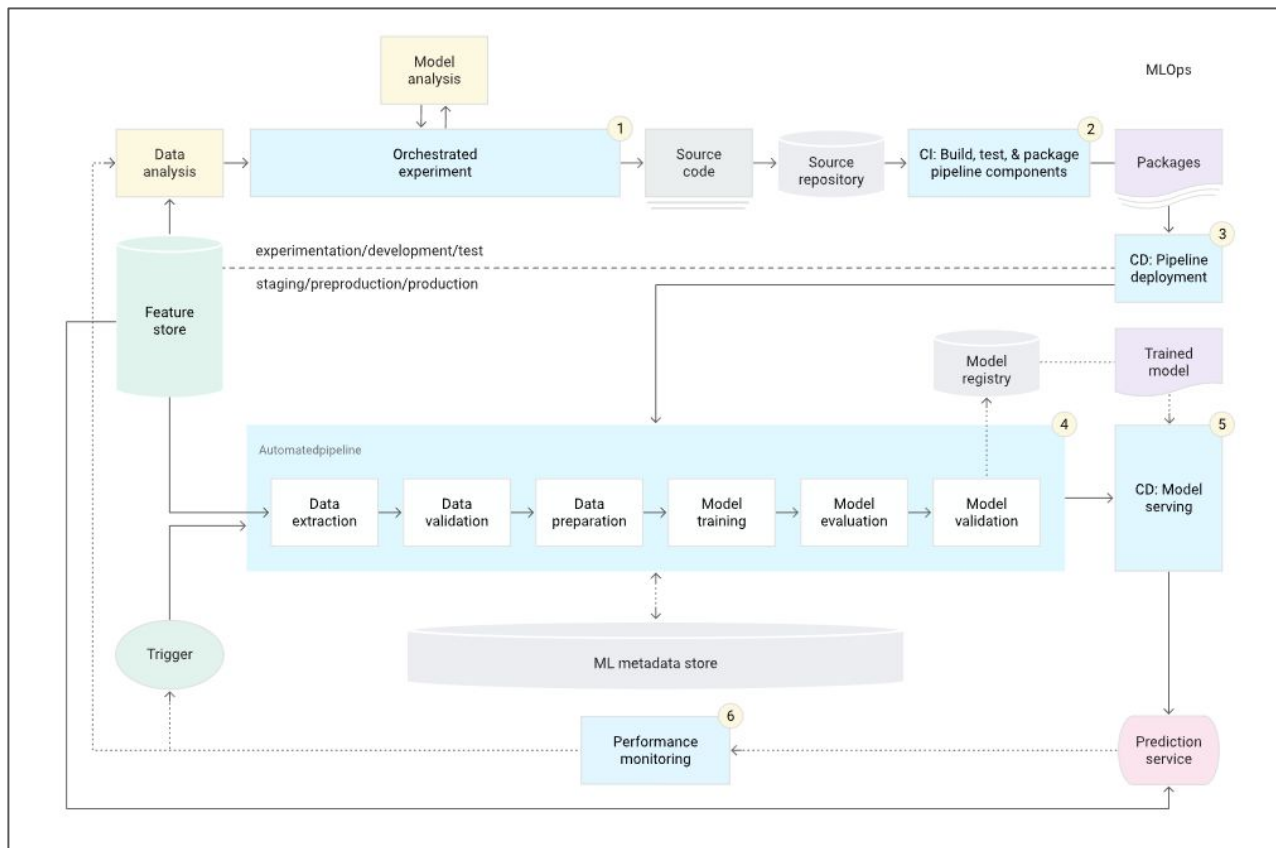
## GitOps Principles

- 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



## 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](https://linkedin.com/company/red-hat)

 [youtube.com/user/RedHatVideos](https://youtube.com/user/RedHatVideos)

 [facebook.com/redhatinc](https://facebook.com/redhatinc)

 [twitter.com/RedHat](https://twitter.com/RedHat)