

Kubeflow Workshop

葉信和 / Hsin-Ho Yeh
Software Engineer / CEO @ 信誠金融科技
hsinho.yeh@footprint-ai.com



2021/04/26

Download Slides

<https://reurl.cc/Lm4Ab9>



Workshop materials

git clone

The screenshot shows a GitHub repository page for 'FootprintAI / kubeflow-workshop'. The repository is public and has 1 branch and 0 tags. The main branch is 'main'. The repository has 49 commits from 'hsinhyeh'. The commits are listed below:

Commit	Message	Date
hsinhyeh feat: add stock price ipynb file	2b912e4 5 days ago	49 commits
hack	fix: add workaround tensorboard image and update its slides	4 months ago
install	fix kf1.4 script	5 months ago
misc	feat(examples) add materials including installation script and pipeline...	13 months ago
pipelines	chore(kfserv): add notes and give names to testdata	6 months ago
slides	feat: add slide for e2e example	4 months ago
tutorials	feat: add stock price ipynb file	5 days ago
LICENSE	Initial commit	13 months ago
README.md	Update README.md	2 months ago

The repository has 10 forks and 5 stars. There are no releases published. A 'Create a new release' button is available.

About me

- 2020 - Present at 信誠金融科技
 - Shrimping: A data-sharing platform
 - <https://get-shrimping.footprint-ai.com>
 - Tintin: a machine learning platform for everyone
 - <https://get-tintin.footprint-ai.com>
- 2016 - 2020 at Igloolnsure (16M+ in series A+ 2020)
 - Provide digital insurance for e-economic world
 - Funded in KUL, Headquartered in Singapore
 - First employee/ Engineering Lead / Regional Head/ Chief Engineer
- 2013 - 2016 at Studio Engineering @ hTC
 - Principal Engineer on Cloud Infrastructure Team
- 2009 - 2012 at IIS @ Academia Sinica
 - Computer vision, pattern recognition, and data mining
- CS@CCU, CS@NCKU alumni



Agenda

- Pre-requirement
- Why Kubeflow?
- What is Kubeflow?
- Kubeflow Architecture
- Hands-on time
- QA

Pre-requirement

- Be comfortable with UNIX command line
 - Navigating directories with `cd` or `tree`
 - Editing files, like `vim`, `nano`
 - Bash scripting, like env or looping
- Be an export with `Google`
 - <https://letmegoogletest.com/?q=you+can+google+it>
- It is totally OK if you don't know what is Container and Kubernetes

孩子，您多久沒唸中文了？

荀子《儒效篇》

「不聞不若聞之，聞之不若見之，見之不若知之，知之不若行之；學至于行之而止矣。」

What is deployment automation in Machine learning?



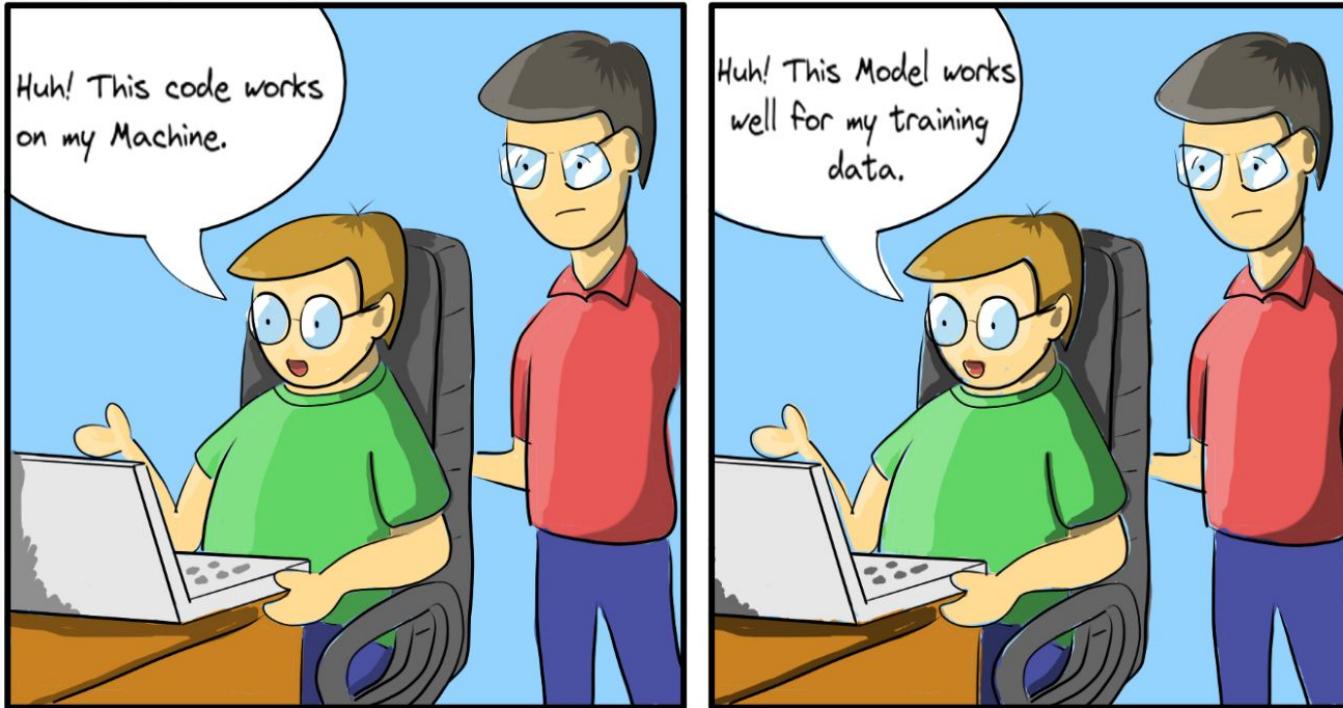
DevOps + ML
= MLOps

MLOps is the process of taking an experimental Machine Learning model into a production system by including continuous development practice of DevOps in the software field.

Ref: <https://en.wikipedia.org/wiki/MLOps>

Source: <https://www.kubeflow.org/>

A common scenario that we both experienced.

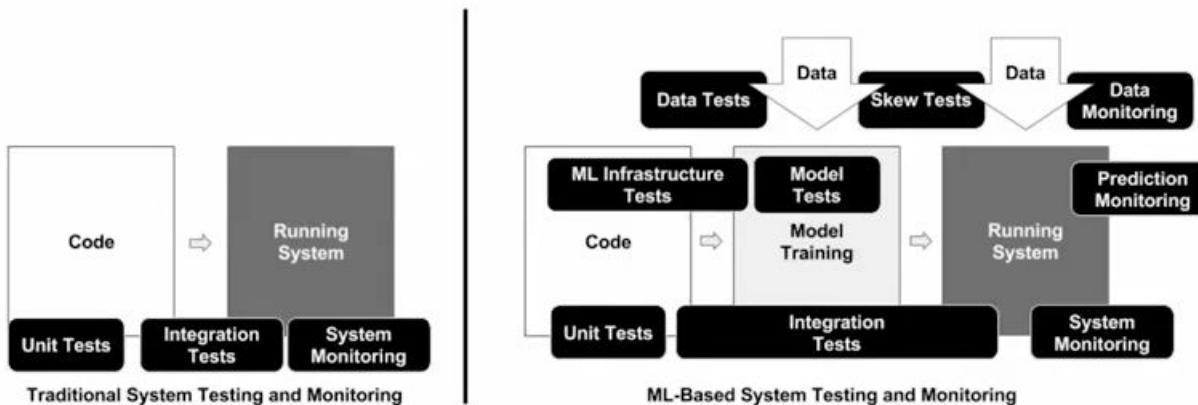


Building & deploying real-world ML application is *hard* and *costly* because of *lack of tooling* that covers end-to-end ML development & deployment

- CloudNext'19

How Involving Machine Learning model could change the current software design?

Traditional vs. ML infused systems



ML introduces two new assets into the software development lifecycle – **data** and **models**.

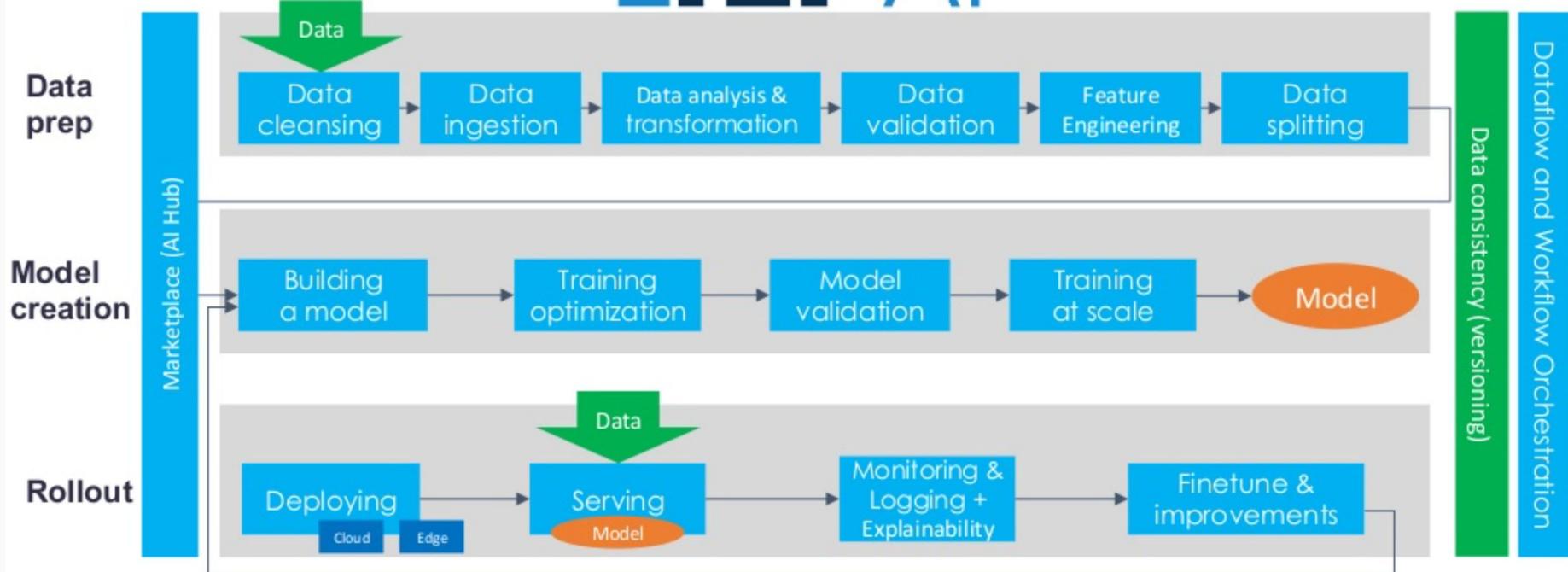
Why we should care about drifting?

- Data drifting
 - A skew grows between training data and serving data.
 - The discrepancies between training data and serving data can usually be classified as schema skews or distribution skews
- Concept drifting
 - The interpretation of the relationship between the input predictors and the target feature evolves



Ref: <https://cloud.google.com/architecture/ml-modeling-monitoring-analyzing-training-server-skew-in-ai-platform-prediction-with-tfdv>

Real-world Machine Learning Application - End-to-End ML LifeCycle



Source: <https://www.slideshare.net/AnimeshSingh/advanced-model-inferencing-leveraging-kubeflow-serving-knative-and-istio-196096385>

Why machine learning on Kubernetes?

- Composability
 - Each stage are independent systems and are able to compose together
- Portability
 - Dev/Staging/Prod
 - Laptop/Edge/Cloud environment
- Scalability
 - Hyperparameter tuning, production workloads

Oh, you want to use ML on K8s?

Before that, can you become an expert in:

- Containers
- Packaging
- Kubernetes service endpoints
- Persistent volumes
- Scaling
- Immutable deployments
- GPUs, Drivers & the GPL
- Cloud APIs
- DevOps
- ...

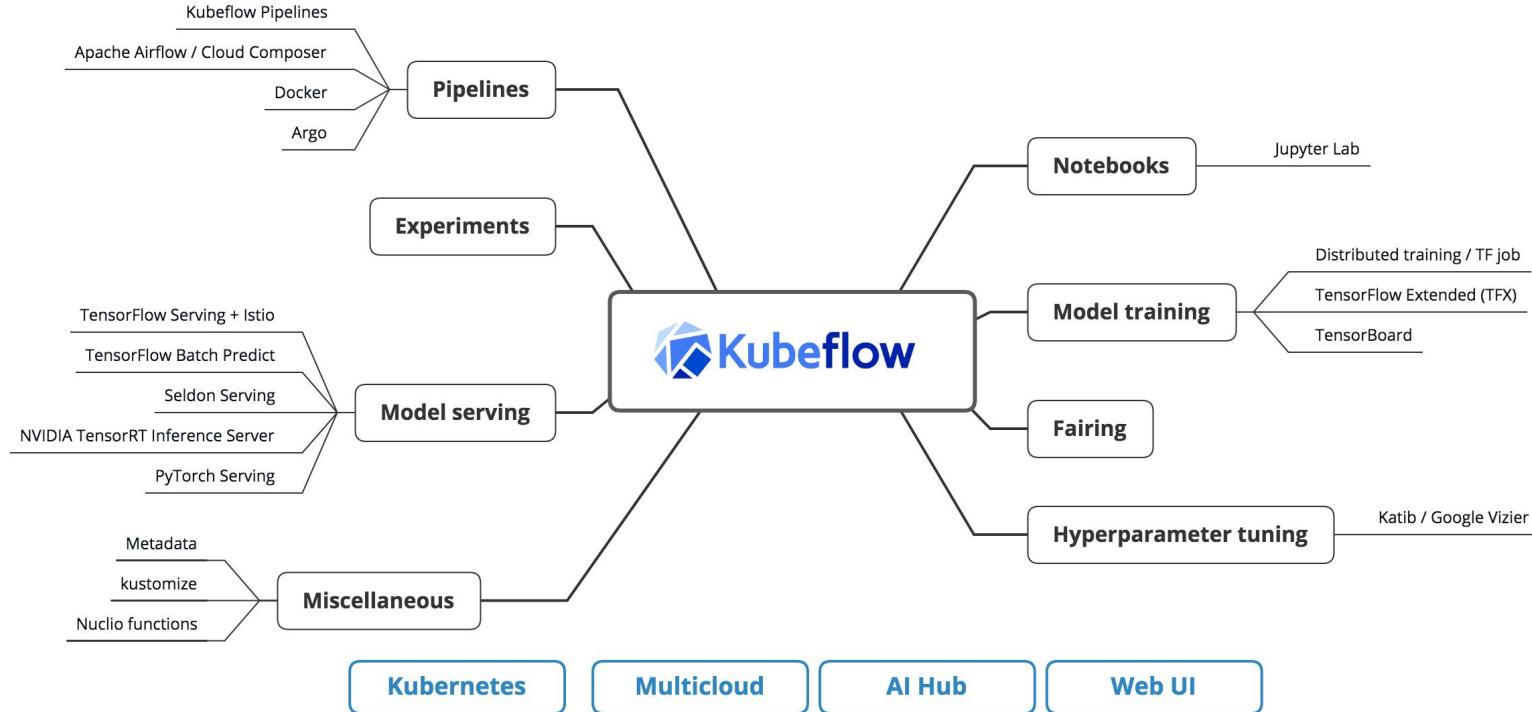




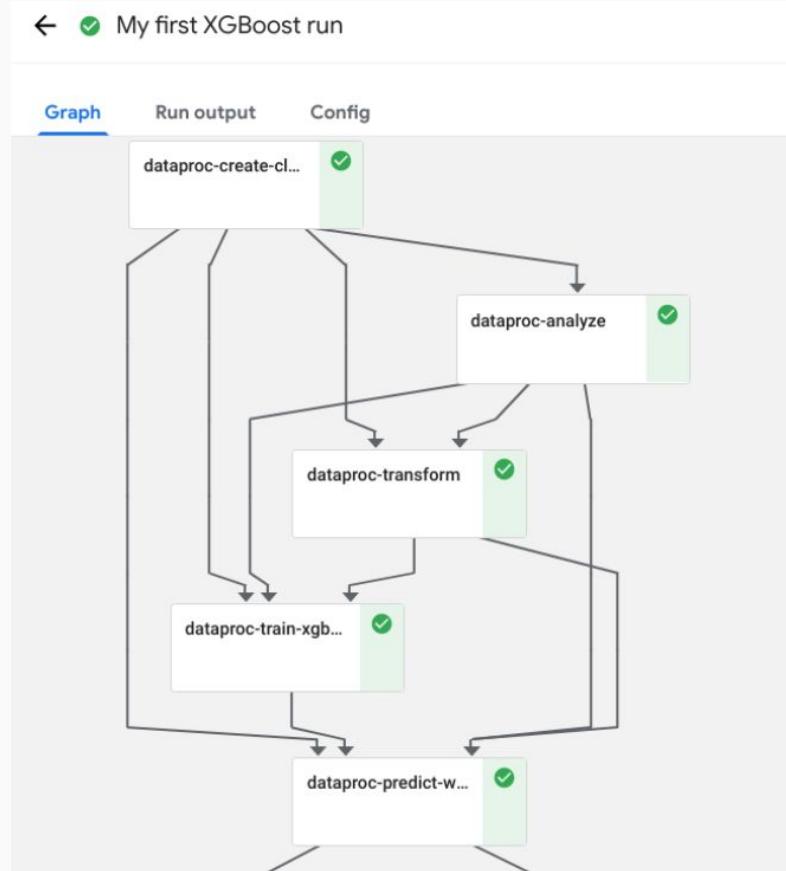
Kubernetes + ML
= Kubeflow

The Kubeflow project
is dedicated to making
deployments of
machine learning (ML)
workflows on
Kubernetes simple,
portable and scalable.

Architectures



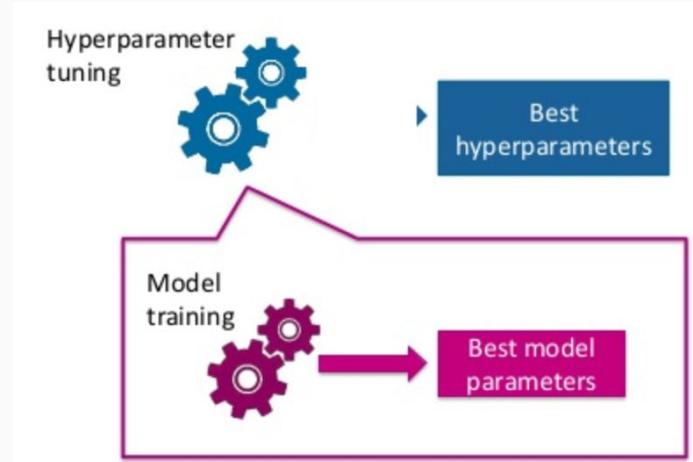
Kubeflow Pipelines



Source:
<https://www.kubeflow.org/docs/pipelines/overview/pipelines-overview/>

Hyperparameter tuning

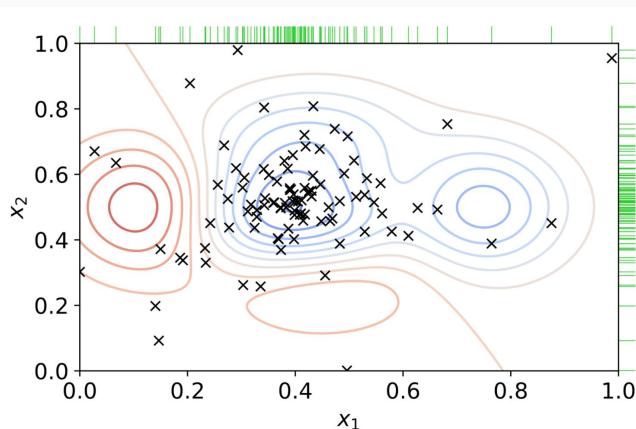
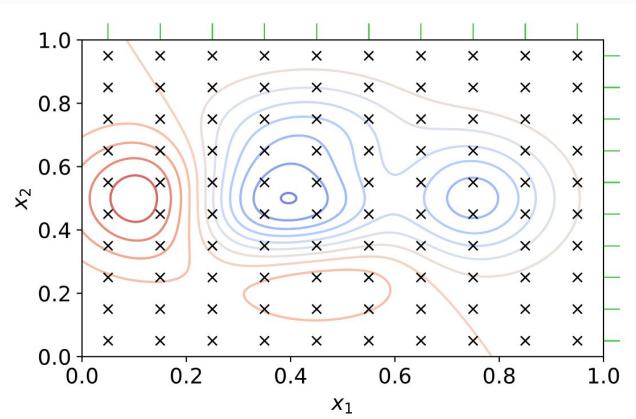
- Model Parameter vs Hyperparameter
 - Model parameters that will learn on its own during training process by the ML model, ex: weights and biases for a classifier.
 - Hyperparameter that directly control the behavior of training algorithm.



Source: <https://towardsdatascience.com/understanding-hyperparameters-and-its-optimisation-techniques-f0debba07568>

Hyperparameter tuning

- Algorithms
 - Grid Search (top)
 - Random Search
 - Bayesian Optimization
 - Gradient-based optimization (bottom)
 - ... and more



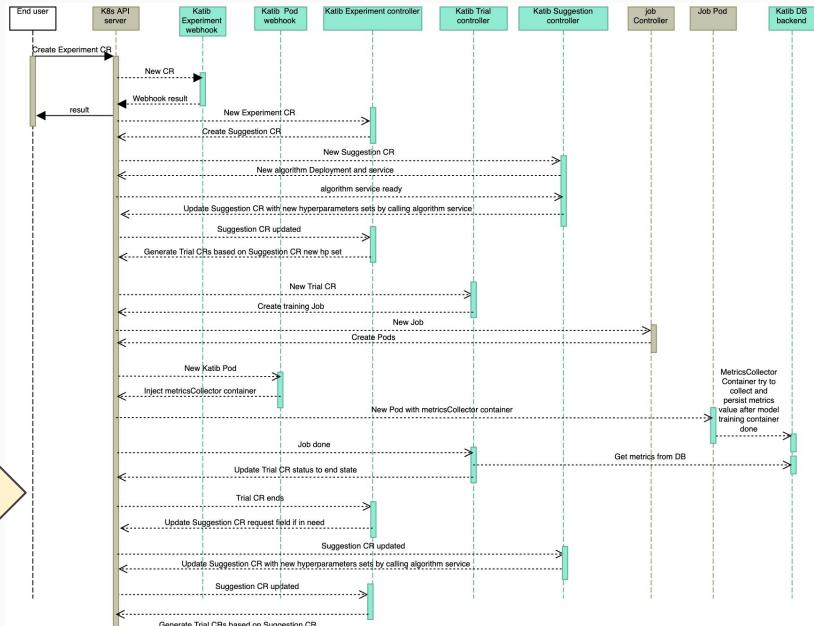
Source: https://en.wikipedia.org/wiki/Hyperparameter_optimization

Kubeflow Katib

- All works that katib has been doing can be described the following pseudocode and then, in reality, turn into the flow diagram below.

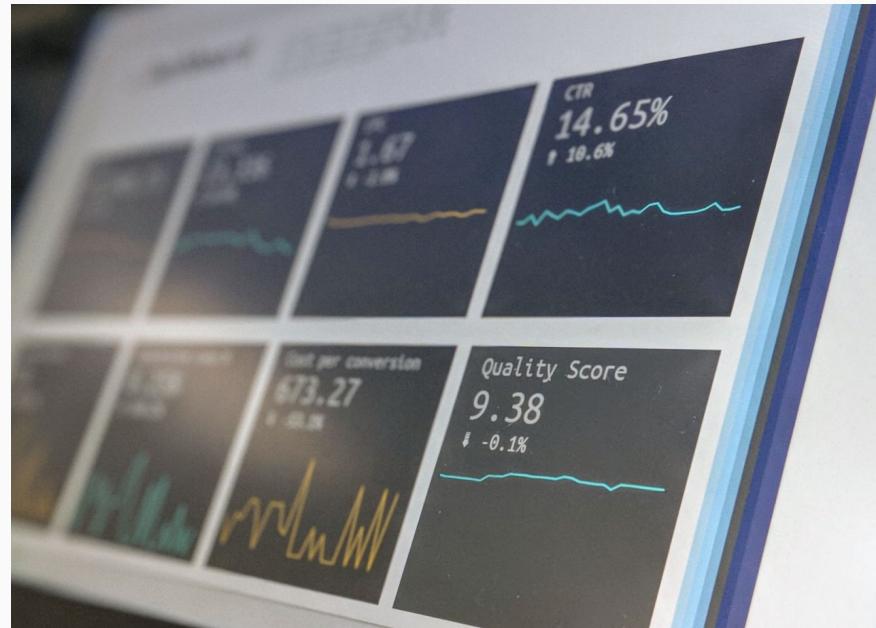
PROGRAM KATIB:

```
exp = CreateExperiment();
WHILE exp.objective is NOT reached:
    DO sugst = exp.CreateSuggestion();
        metrics = exp.CreateTrials(sugst.Assignments);
        exp.Report(metrics);
END.
```



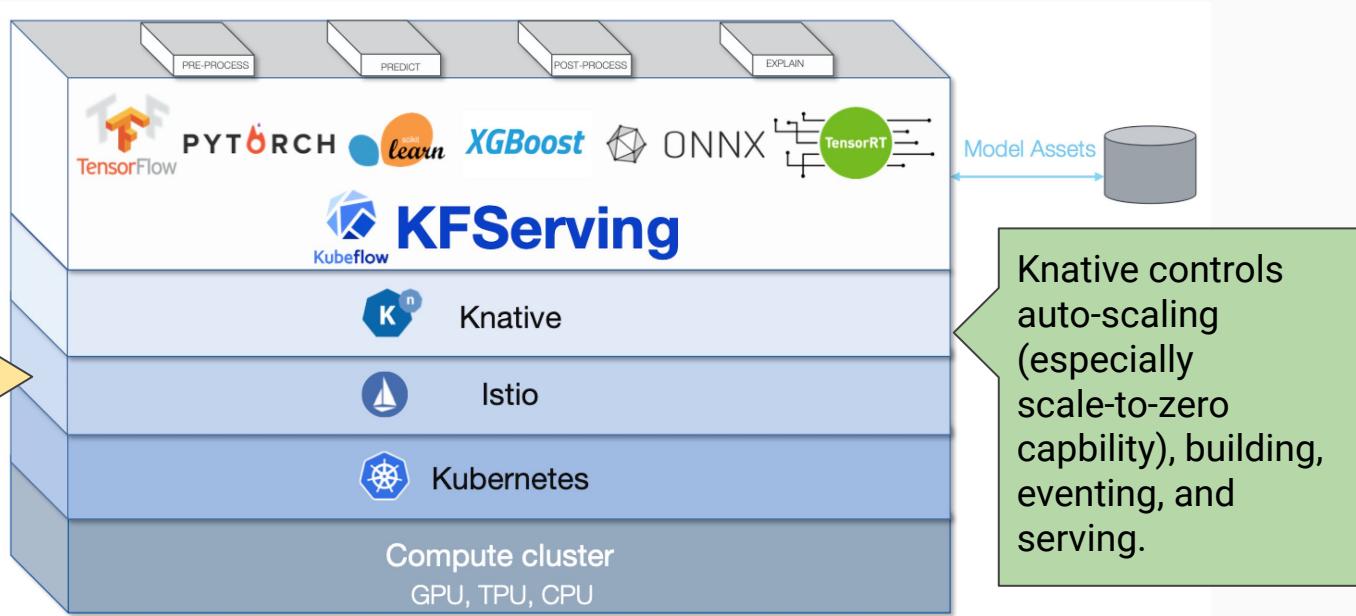
Model Serving

- How hard it could be to serve ML models in production scale?
 - Scale vs Cost
 - Seamless Rollout
 - Canary Rollouts
 - Service/Model monitoring



Source: https://unsplash.com/@srd844?utm_source=medium&utm_medium=referral

KServe



Source: <https://towardsdatascience.com/understanding-hyperparameters-and-its-optimisation-techniques-f0debba07568>

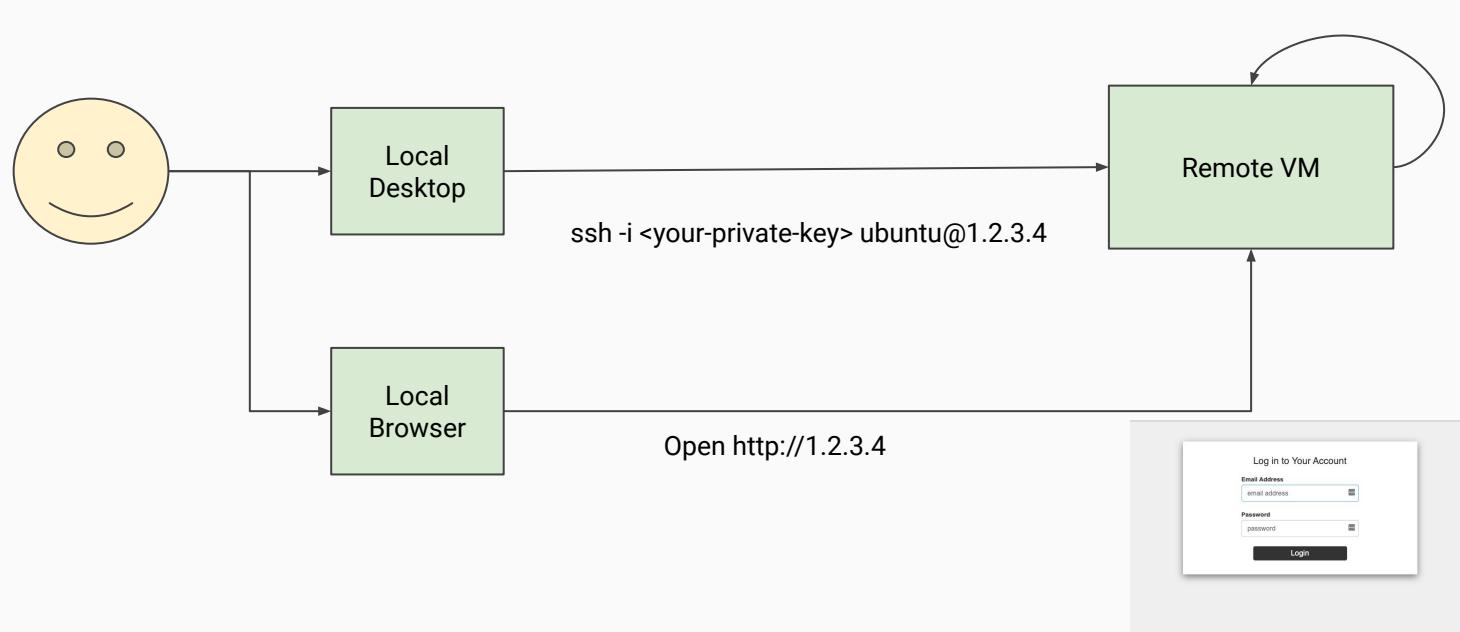
Environment Setup

Manual Installation

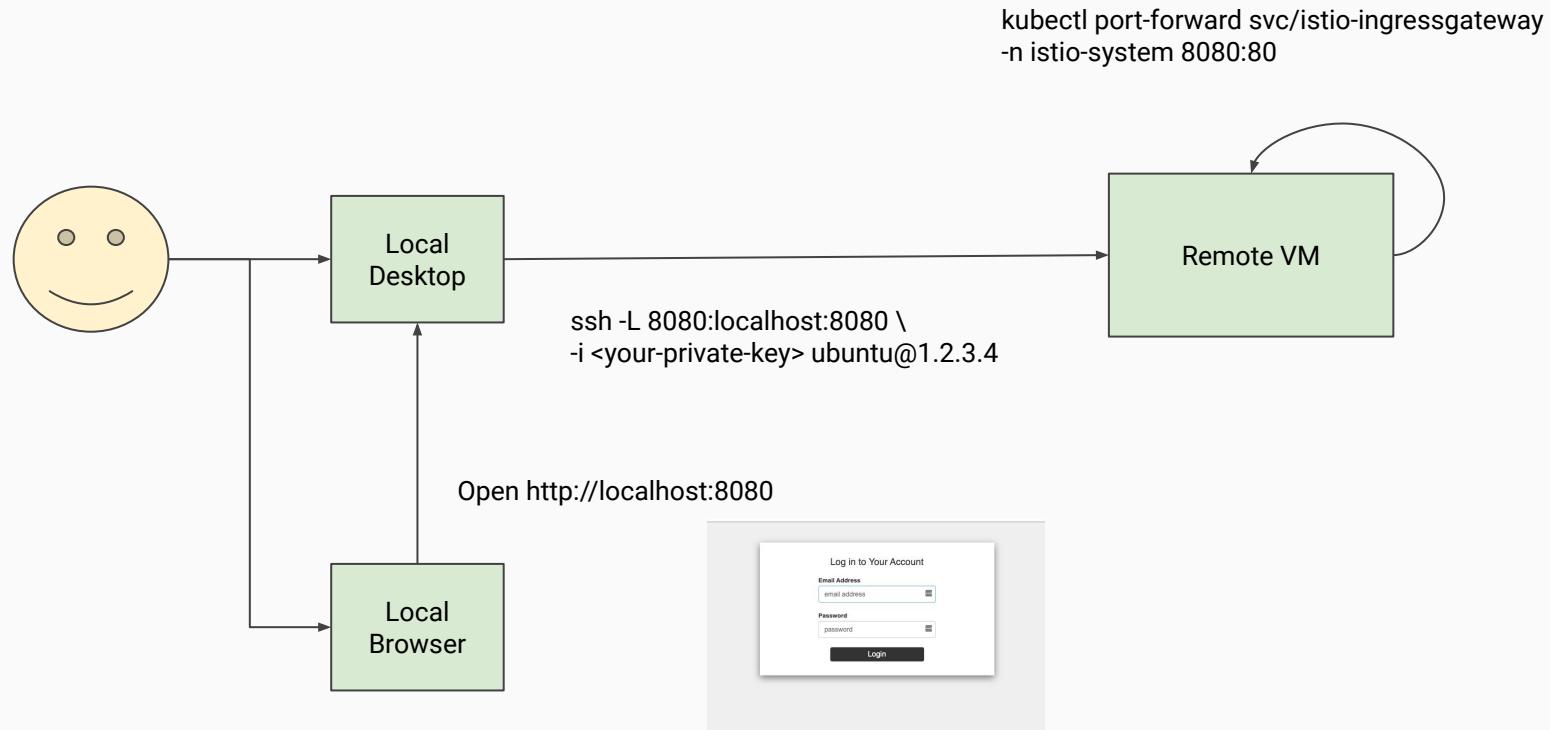
- Manual Installation
- <https://github.com/FootprintAI/kubeflow-workshop/tree/main/install>

- Windows Installation
- <https://github.com/FootprintAI/kubeflow-workshop/tree/main/install/windows>

Manual Installation: Access Remote VM via Public IP



Manual Installation: Access Remote VM via SSH Local Forwarding



multikf: One-click Installation

```
// install dockerd
wget https://raw.githubusercontent.com/FootprintAI/multikf/main/docker/linux.sh
chmod +x linux.sh
sudo ./linux.sh

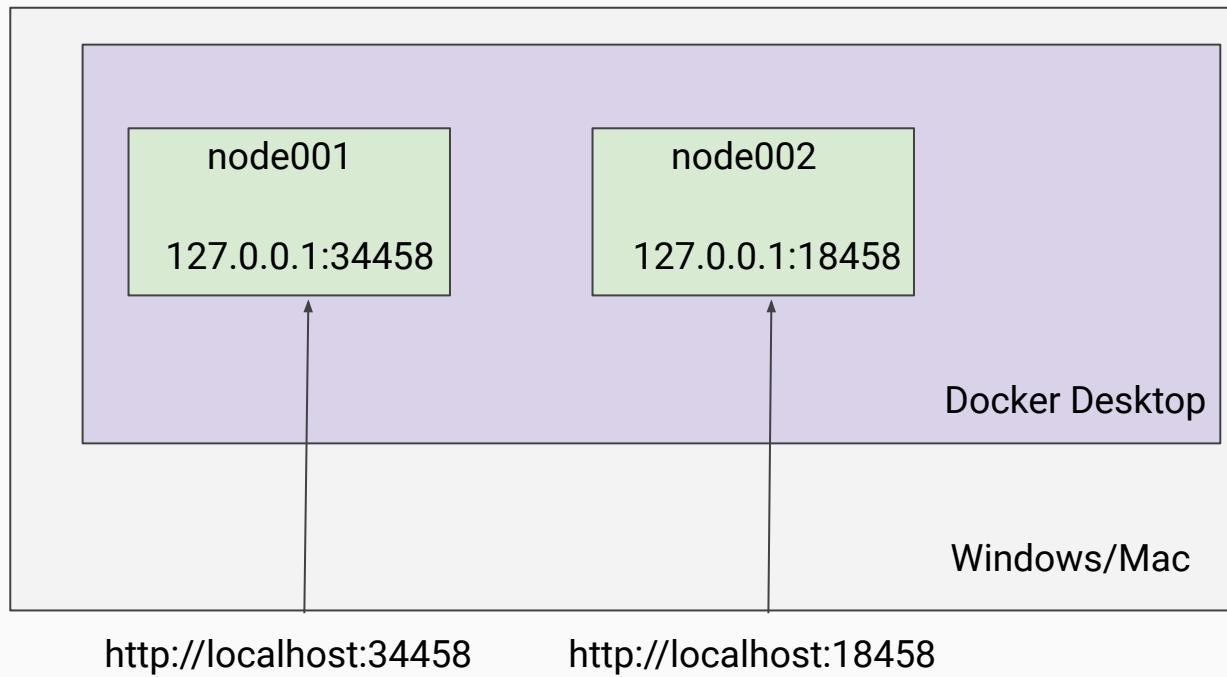
// install multikf
wget https://github.com/FootprintAI/multikf/raw/main/build/multikf.linux
chmod +x multikf.linux

// add an instances with port 80/443 exported
./multikf.linux add node002 --export_ports 80:80,443:443

// connect kubeflow
./multikf.linux connect kubefloe node002
```

multikf: One-click Installation

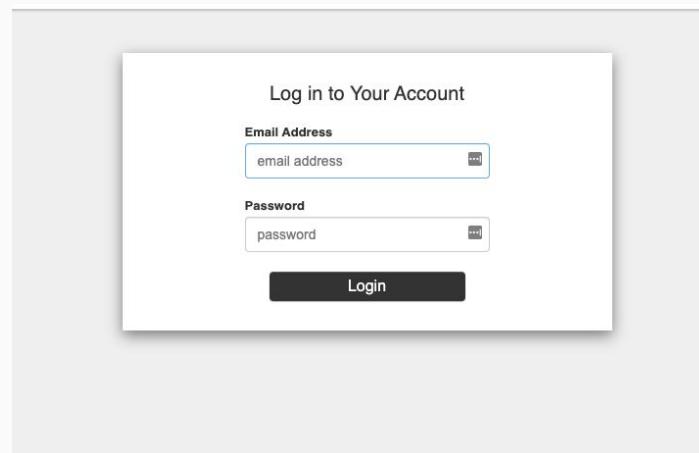
- Multikf: <https://github.com/footprintai/multikf>



Hands-on Time

Wait! 所以我說那個帳號密碼呢?

Account: user@example.com
Password: 12341234



Step1: Use Notebook as Online IDE (1/3)

The screenshot shows the Kubeflow web interface. On the left, a sidebar menu is visible with the following items:

- Home
- Notebooks** (highlighted with a red box)
- Tensorboards
- Volumes
- Models
- Experiments (AutoML)
- Experiments (KFP)
- Pipelines
- Runs
- Recurring Runs
- Artifacts
- Executions

At the bottom of the sidebar, there is a link to "Privacy • Usage Reporting" and a note about the build version: "build version dev_local".

The main content area is titled "Notebooks" and displays a table with columns: Status, Name, Type, Age, Image, GPUs, CPUs, Memory, and Volumes. There is no data in the table.

A large red arrow points upwards from the "+ NEW NOTEBOOK" button towards the top right of the screen.

Step1: Use Notebook as Online IDE (2/3)

The screenshot shows the Kubeflow interface for creating a new Notebook Server. On the left is a sidebar with various options: Home, Notebooks (selected), Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP), Pipelines, Runs, Recurring Runs, Artifacts, Executions, Manage Contributors, and Privacy, Help, Reporting.

The main area is titled "Specify the name of the Notebook Server and the Namespace it will belong to." It contains two input fields: "Name" (containing "demo") and "Namespace" (containing "kubeflow-user-example-com").

Below these are sections for "Image" and "Advanced Options". The "Image" section includes a "Custom Image" checkbox, a dropdown menu set to "jupyterlab" (with options 1 and 2), and another dropdown menu showing "j1r0q0g6/notebooks/notebook-servers/jupyter-tensorflow-full:v1.4". A large red arrow points from the text "Specify a name and resources like CPU and Memory" to the "jupyterlab" dropdown.

The "Advanced Options" section has a "CPU / RAM" subsection where users can specify the total amount of CPU and RAM reserved. The "Requested CPU" field is set to "0.5" and the "Requested memory (in GiB)" field is set to "1". Red boxes highlight both of these input fields.

A large red arrow points from the text "Specify a name and resources like CPU and Memory" to the "jupyterlab" dropdown.

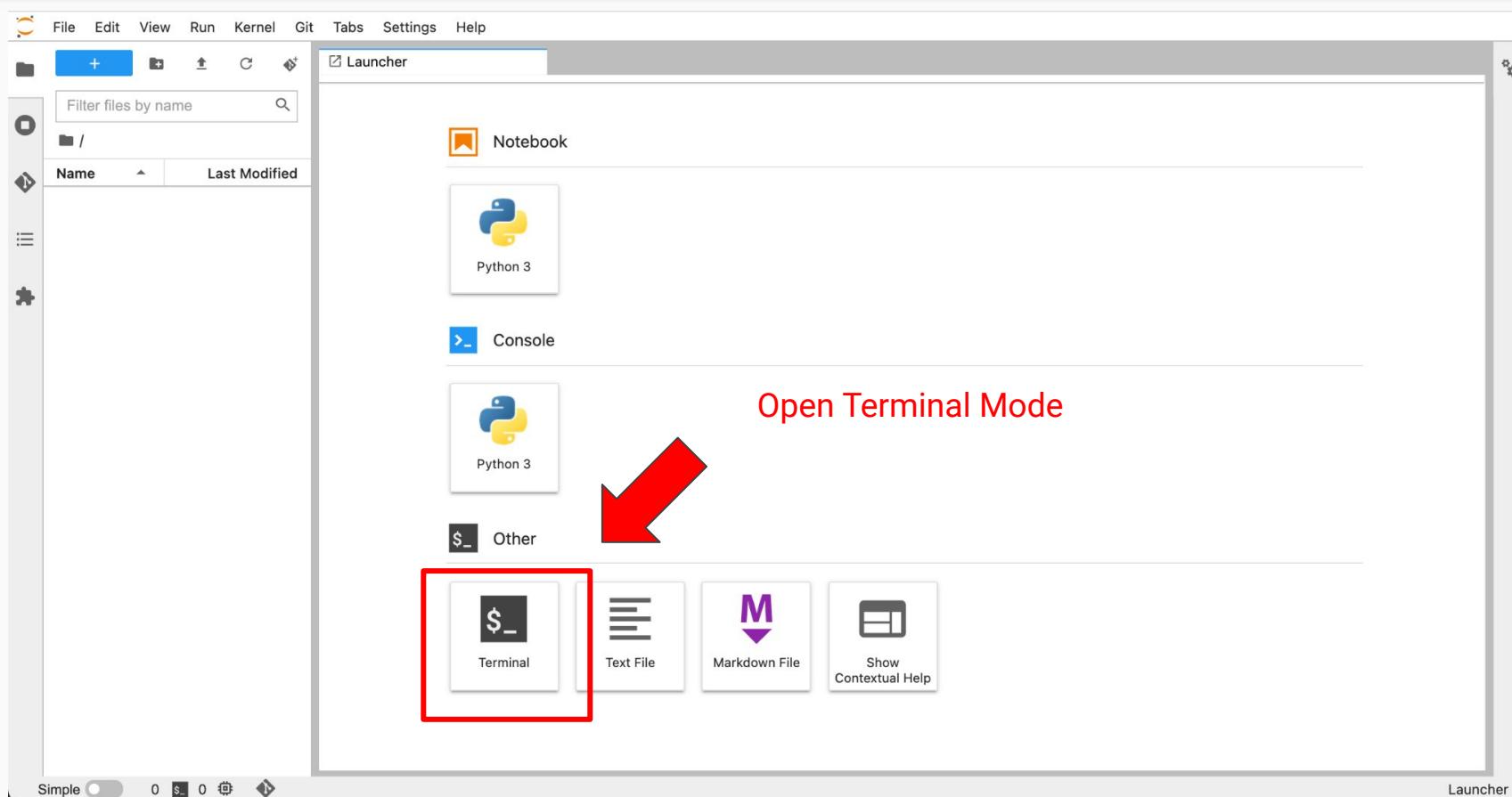
Specify a name and resources like CPU and Memory

Step1: Use Notebook as Online IDE (3/3)

The screenshot shows the Kubeflow interface with the 'Notebooks' tab selected in the sidebar. The main area displays a table of notebooks with columns for Status, Name, Type, Age, Image, GPUs, CPUs, Memory, and Volumes. Two notebooks are listed: 'demo1' and 'demo2'. Both are in a 'Running' state (indicated by green checkmarks). The 'demo1' row includes a 'CONNECT' button and a trash bin icon. The 'demo2' row also includes a 'CONNECT' button and a trash bin icon. A large red arrow points upwards from the bottom of the screen towards the 'CONNECT' button for 'demo2'. A red box highlights the 'CONNECT' button for 'demo2'.

Status	Name	Type	Age	Image	GPUs	CPUs	Memory	Volumes
✓	demo1	jupyter	20 hours ago	jupyter-scipy:v1.4	0	0.5	1Gi	
✓	demo2	jupyter	2 hours ago	jupyter-tensorflow-full:v1.4	0	0.5	1Gi	

Step2: Use terminal to download the materials (1/3)



Step2: Use terminal to download the materials (2/3)

The screenshot shows a Jupyter Notebook interface. On the left is a file browser sidebar with a tree view of files and a search bar. The main area has a terminal window titled "Terminal 1" and a code block.

Terminal 1

```
groups: cannot find name for group ID 1232
(base) jovyan@dem01-0:~$ git clone https://github.com/footprintai/kubeflow-workshop
Cloning into 'kubeflow-workshop'...
remote: Enumerating objects: 164, done.
remote: Counting objects: 100% (164/164), done.
remote: Compressing objects: 100% (98/98), done.
remote: Total 164 (delta 89), reused 133 (delta 62), pack-reused 0
Receiving objects: 100% (164/164), 1.71 MiB | 8.29 MiB/s, done.
Resolving deltas: 100% (89/89), done.
(base) jovyan@dem01-0:~$
```

git clone https://github.com/footprintai/kubeflow-workshop

A large red arrow points upwards from the text "git clone https://github.com/footprintai/kubeflow-workshop" towards the terminal output. The terminal output is highlighted with a red rectangle.

Step2: Use terminal to download the materials (3/3)

The screenshot shows the Jupyter Notebook interface. On the left is a file browser pane with a red border around the list of files. The list includes:

- img
- 0.helloworld.ipynb
- 1.conditional-flow.ipynb
- 2.persistvolume.ipynb
- 3.calc_metrics.ipynb
- 4.mnist.ipynb
- 5.auto-mnist-with-katib.ipynb
- 6.kfserving.ipynb
- 7.kfserving-canary-rollout.ipynb
- kfp.ipynb
- testdata.jpg
- testdata2.jpg
- testdata3.jpg

On the right is a terminal window titled "Terminal 1" showing the output of a git clone command:

```
groups: cannot find name for group ID 1337
(base) jovyan@demo1-0:~$ git clone https://github.com/footprintai/kubeflow-workshop
Cloning into 'kubeflow-workshop'...
remote: Enumerating objects: 164, done.
remote: Counting objects: 100% (164/164), done.
remote: Compressing objects: 100% (98/98), done.
remote: Total 164 (delta 89), reused 133 (delta 62), pack-reused 0
Receiving objects: 100% (164/164), 1.71 MiB | 8.29 MiB/s, done.
Resolving deltas: 100% (89/89), done.
(base) jovyan@demo1-0:~$
```

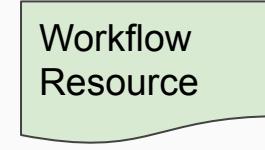
Kubeflow Terms

詞彙說明

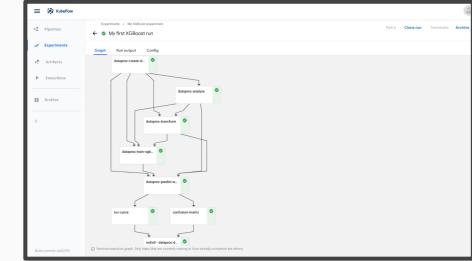


```
||| with open('requirements.txt', 'w') as f:  
|||     f.write("pyyaml<4.4.0")  
|||     pip install -r requirements.txt --upgrade --user  
  
||| import sys  
|||  
||| def echo(text):  
|||     return sys.stdout.write(  
|||         text)  
|||  
||| def echo_task(text):  
|||     echo(text)  
|||     echo("task_id=" + str(4.4.2), text)  
|||     arguments = text.split(" ", 1)  
|||     arguments[0] = "echo", arguments[1]  
|||  
|||     del arguments[1]  
|||  
|||     # run execution order pipeline  
|||     # This pipeline demonstrates execution order management.  
|||     def execution_order_pipeline(text="message 1", text2="message 2"):  
|||         step1_task = echo_task(text)  
|||         step2_task = echo_task(text2)  
|||         step2_task.after(step1_task)  
  
||| # generate workflow artifacts in .zip format  
||| MyWorkflow=CompilePipeline.compile_execution_order_pipeline('yellowworld.zip')
```

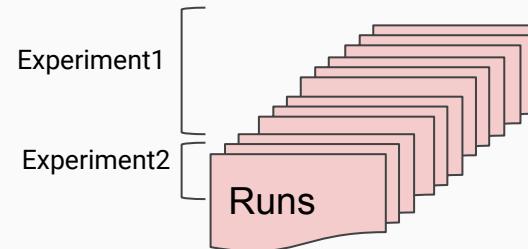
Compiled



Create a Pipeline



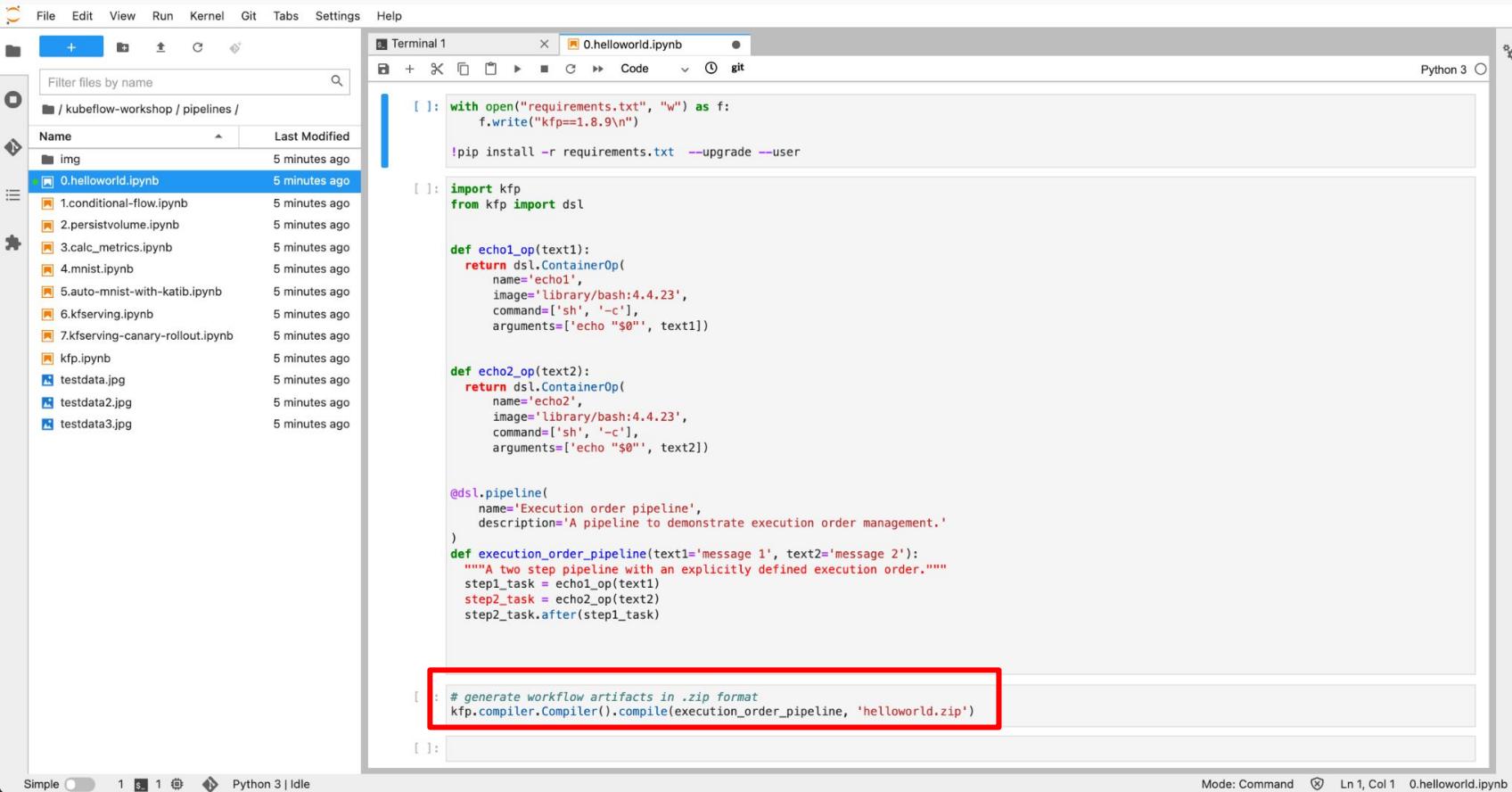
Pipeline Code



Create Run

Hello World Example

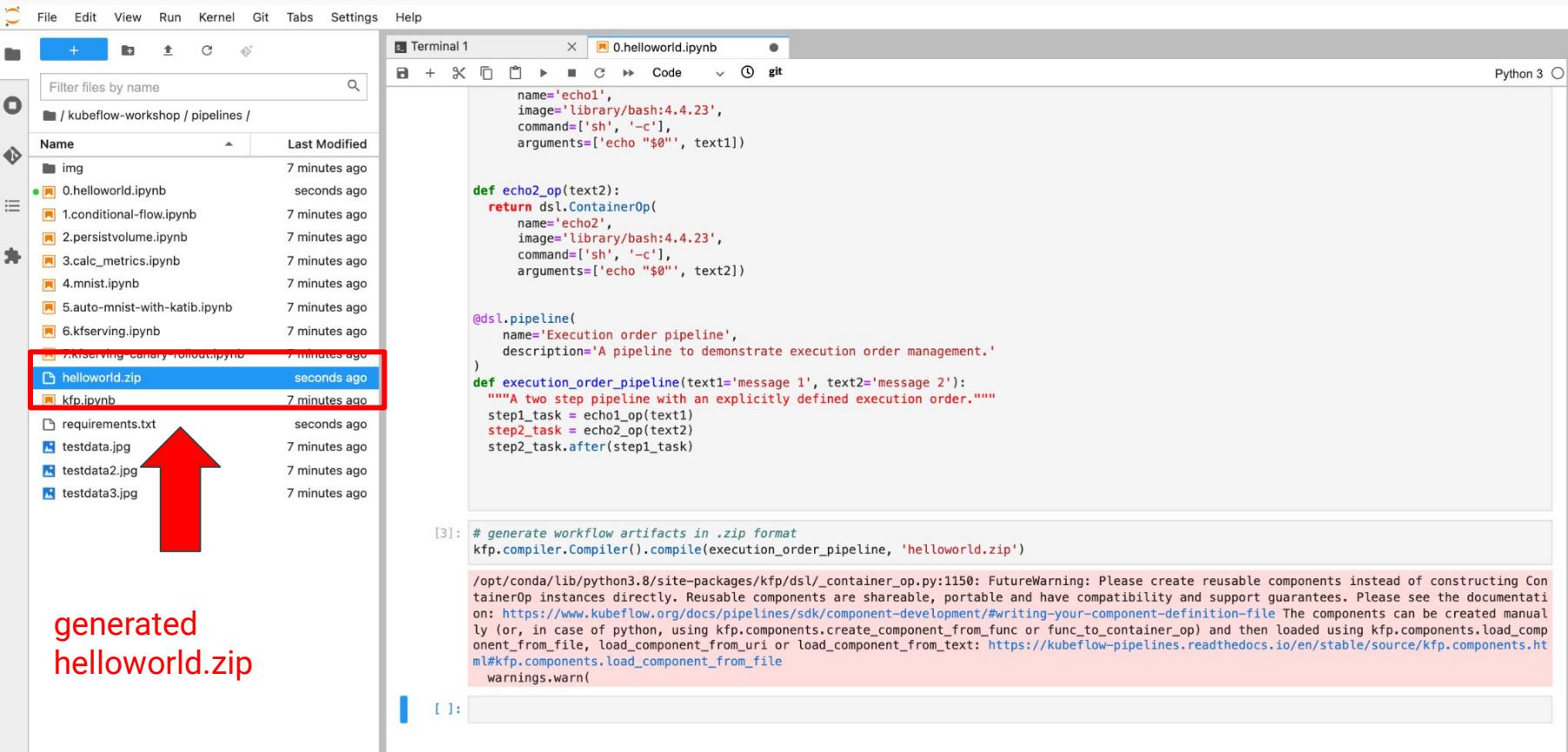
Step4: Compile helloworld.ipynb (1/2)



The screenshot shows a Jupyter Notebook interface with a sidebar on the left containing file navigation and search tools. The main area has two tabs: 'Terminal 1' and '0.helloworld.ipynb'. The terminal tab shows command-line output, while the notebook tab displays Python code. A red box highlights the last line of code in the notebook, which is the command to compile the pipeline.

```
[ ]: with open("requirements.txt", "w") as f:  
    f.write("kfp==1.8.9\n")  
  
!pip install -r requirements.txt --upgrade --user  
  
[ ]: import kfp  
from kfp import dsl  
  
def echo1_op(text1):  
    return dsl.ContainerOp(  
        name='echo1',  
        image='library/bash:4.4.23',  
        command=['sh', '-c'],  
        arguments=['echo "$0"', text1])  
  
def echo2_op(text2):  
    return dsl.ContainerOp(  
        name='echo2',  
        image='library/bash:4.4.23',  
        command=['sh', '-c'],  
        arguments=['echo "$0"', text2])  
  
@dsl.pipeline(  
    name='Execution order pipeline',  
    description='A pipeline to demonstrate execution order management.')  
def execution_order_pipeline(text1='message 1', text2='message 2'):  
    """A two step pipeline with an explicitly defined execution order."""  
    step1_task = echo1_op(text1)  
    step2_task = echo2_op(text2)  
    step2_task.after(step1_task)  
  
[ ]: # generate workflow artifacts in .zip format  
kfp.compiler.Compiler().compile(execution_order_pipeline, 'helloworld.zip')
```

Step4: Compile helloworld.ipynb (2/2)



The screenshot shows a Jupyter Notebook interface with a sidebar and a main workspace.

File Explorer: On the left, it lists files and folders. A red arrow points from the "generated helloworld.zip" text below to the "helloworld.zip" file in the list, which is highlighted with a blue border. Other files listed include requirements.txt, testdata.jpg, testdata2.jpg, and testdata3.jpg.

Terminal 1: In the main workspace, the notebook cell 0.helloworld.ipynb is running. The code defines two components (echo1 and echo2) and a pipeline (execution_order_pipeline) that runs them sequentially. The final command in the cell is `kfp.compiler.Compiler().compile(execution_order_pipeline, 'helloworld.zip')`.

Output: Below the code cell, the terminal shows a warning message from the ContainerOp module about creating reusable components. It also shows the command being run: `/opt/conda/lib/python3.8/site-packages/kfp/dsl/_container_op.py:1150: FutureWarning: Please create reusable components instead of constructing ContainerOp instances directly. Reusable components are shareable, portable and have compatibility and support guarantees. Please see the documentation on: https://www.kubeflow.org/docs/pipelines/sdk/component-development/#writing-your-component-definition-file The components can be created manually (or, in case of python, using kfp.components.create_component_from_func or func_to_container_op) and then loaded using kfp.components.load_component_from_file, load_component_from_uri or load_component_from_text: https://kubeflow-pipelines.readthedocs.io/en/stable/source/kfp.components.html#kfp.components.load_component_from_file warnings.warn(`

[3]: # generate workflow artifacts in .zip format
kfp.compiler.Compiler().compile(execution_order_pipeline, 'helloworld.zip')

/opt/conda/lib/python3.8/site-packages/kfp/dsl/_container_op.py:1150: FutureWarning: Please create reusable components instead of constructing ContainerOp instances directly. Reusable components are shareable, portable and have compatibility and support guarantees. Please see the documentation on: https://www.kubeflow.org/docs/pipelines/sdk/component-development/#writing-your-component-definition-file The components can be created manually (or, in case of python, using kfp.components.create_component_from_func or func_to_container_op) and then loaded using kfp.components.load_component_from_file, load_component_from_uri or load_component_from_text: https://kubeflow-pipelines.readthedocs.io/en/stable/source/kfp.components.html#kfp.components.load_component_from_file
warnings.warn(

[]:

generated
helloworld.zip

Step4: Create a Pipeline (1/7)

The screenshot shows the Kubeflow Pipelines interface. On the left, a sidebar menu is visible with various options: Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP), Pipelines (highlighted with a red box), Runs, Recurring Runs, Artifacts, and Executions. Below this is a 'Manage Contributors' section. The main area is titled 'Pipelines' and contains a table of existing pipelines. The table has columns for Pipeline name, Description, and Last modified. A red box highlights the '+ Upload pipeline' button in the top right corner of the header. A large red arrow points upwards towards this button. The table data is as follows:

Pipeline name	Description	Last modified
[Tutorial] V2 lightweight Python com...	source code Shows different component input and output options for KFP v2 components.	11/30/2021, 1:02:25 PM
[Tutorial] DSL - Control structures	source code Shows how to use conditional execution and exit handlers. This pipeline will randomly fail to demonstr...	11/30/2021, 1:02:24 PM
[Tutorial] Data passing in python co...	source code Shows how to pass data between python components.	11/30/2021, 1:02:23 PM
[Demo] TFX - Taxi tip prediction mod...	source code GCP Permission requirements. Example pipeline that does classification with model analysis based on...	11/30/2021, 1:02:22 PM
[Demo] XGBoost - Iterative model tra...	source code This sample demonstrates iterative training using a train-eval-check recursive loop. The main pipeline ...	11/30/2021, 1:02:21 PM

Rows per page: 10 < >

Step4: Create a Pipeline (2/7)

The screenshot shows the Kubeflow interface for creating a new pipeline. The left sidebar lists various options like Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP), Pipelines, Runs, Recurring Runs, Artifacts, and Executions. The main area is titled 'Upload Pipeline or Pipeline Version' with a sub-section 'Pipeline Versions'. It has two radio button options: 'Create a new pipeline' (selected) and 'Create a new pipeline version under an existing pipeline'. A large red box highlights the 'Pipeline Name' field, which contains '0.helloworld'. A large red arrow points from the text '1.Pipeline Name' to this field. Below it is a 'Pipeline Description' field containing '0.helloworld'. The next section asks to choose a pipeline package file, with a red box highlighting the 'File*' input field which contains 'helloworld (19).zip' and a 'Choose file' button. A large red arrow points from the text '2.specify zip file location' to this section. At the bottom, there are 'Create' and 'Cancel' buttons, with a red box highlighting the 'Create' button. A large red arrow points from the text '3.Create' to this button.

1.Pipeline Name

2.specify zip file location

3.Create

Step4: Create a Pipeline (3/7)

The screenshot shows the Kubeflow Pipelines interface. On the left, a sidebar menu includes Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP), Pipelines, Runs, Recurring Runs, Artifacts, and Executions. Below this is a section for Manage Contributors.

The main area displays a pipeline named "0.helloworld (0.helloworld)". It shows a graph with two nodes: "echo1" at the top and "echo2" below it. An arrow points from "echo1" to "echo2". Above the graph, there are tabs for "Graph" (selected) and "YAML". To the right of the graph are buttons for "+ Create run", "+ Upload version", and "+ Create experiment". A large red box highlights the "+ Create experiment" button, and a red arrow points upwards towards it from the bottom right. A red callout box contains the text "Create an experiment".

Pipelines

← 0.helloworld (0.helloworld)

+ Create run + Upload version **+ Create experiment**

Graph **YAML**

Simplify Graph

```
graph TD; echo1[echo1] --> echo2[echo2]
```

Summary

ID: 6f25028f-01e3-4acd-9389-7ec2031fb04b
Version: 0.helloworld

Version source

Step4: Create a Pipeline (4/7)

The screenshot shows the Kubeflow interface for creating a new experiment. On the left is a dark sidebar with navigation links: Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP) (which is expanded to show Pipelines, Runs, Recurring Runs, Artifacts, and Executions), Manage Contributors, and GitHub integration.

The main area is titled "Experiments" and "← New experiment". It has a sub-section titled "Experiment details" with a descriptive text: "Think of an Experiment as a space that contains the history of all pipelines and their associated runs".

A form is displayed for entering experiment details:

- Name:** A text input field containing "0.helloworld.exp" is highlighted with a red border.
- Description (optional):** An empty text input field.
- Buttons:** A row of buttons with "Next" highlighted with a red border and a large red arrow pointing to it from the right. Other buttons include "Cancel".

A red annotation text on the right side of the form reads: "Enter experiment name and press Next button."

Step4: Create a Pipeline (5/7)

The screenshot shows the Kubeflow UI for creating a pipeline run. The left sidebar contains navigation links: Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP) (selected), Pipelines, Runs, Recurring Runs, Artifacts, Executions, Manage Contributors, GitHub, Documentation, Privacy + Usage Reporting, and build version dev_local.

The main area is titled "Run details". It includes fields for "Pipeline", "Pipeline Version", "Run name", "Description (optional)", "Experiment", "Service Account (Optional)", "Run Type" (set to "One-off"), and "Run parameters". The "Run parameters" section contains fields for "text1" and "text2". At the bottom are "Start" and "skip this step" buttons.

Two red arrows point from the right towards the "Pipeline" and "Experiment" fields, which are highlighted with red boxes. Red text annotations provide instructions:

1. Run a pipeline and specify its version
2. Add its experiment name

Step4: Create a Pipeline (6/7)

The screenshot shows the Kubeflow interface for managing experiments. On the left, a sidebar navigation bar includes links for Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP), Pipelines, Runs (which is selected and highlighted with a red box), Recurring Runs, Artifacts, and Executions. Below these are Manage Contributors, GitHub integration, and Documentation.

The main content area displays the 'Experiments' view for '0.helloworld.exp'. At the top, there are sections for 'Recurring run configs' (0 active) and 'Experiment description' (with a 'Manage' link). Below this is a 'Runs' section with tabs for 'Active' (selected) and 'Archived'. A search bar labeled 'Filter runs' is present. Two buttons are available: '+ Create run' and '+ Create recurring run'. To the right of the runs table are links for 'Compare runs', 'Clone run', and 'Archive'.

The 'Runs' table has columns for Run name, Status, Duration, Pipeline Version, Recurring Run, and Start time. The first row, 'Run of 0.helloworld (1e261)', is highlighted with a red box and a large red arrow pointing upwards from the bottom of the page towards it. The table also includes a 'Rows per page' dropdown set to 10, and navigation arrows.

Run name	Status	Duration	Pipeline Version	Recurring Run	Start time
Run of 0.helloworld (1e261)	?	-	0.helloworld	-	12/2/2021, 4:33:10 PM

Run List

Step4: Create a Pipeline (7/7)

The screenshot shows the Kubeflow interface for creating a pipeline. On the left, the sidebar includes options like Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP), Pipelines, Runs, Recurring Runs, Artifacts, and Executions. The main area displays a pipeline graph titled "Run of 0.helloworld (1e261)". The graph consists of two nodes: "echo1" and "echo2", connected by a downward arrow. A red box highlights this graph. To the right, a detailed view of the execution is shown for "execution-order-pipeline-fxxfn-4223123588". The "Input/Output" tab is selected, showing:

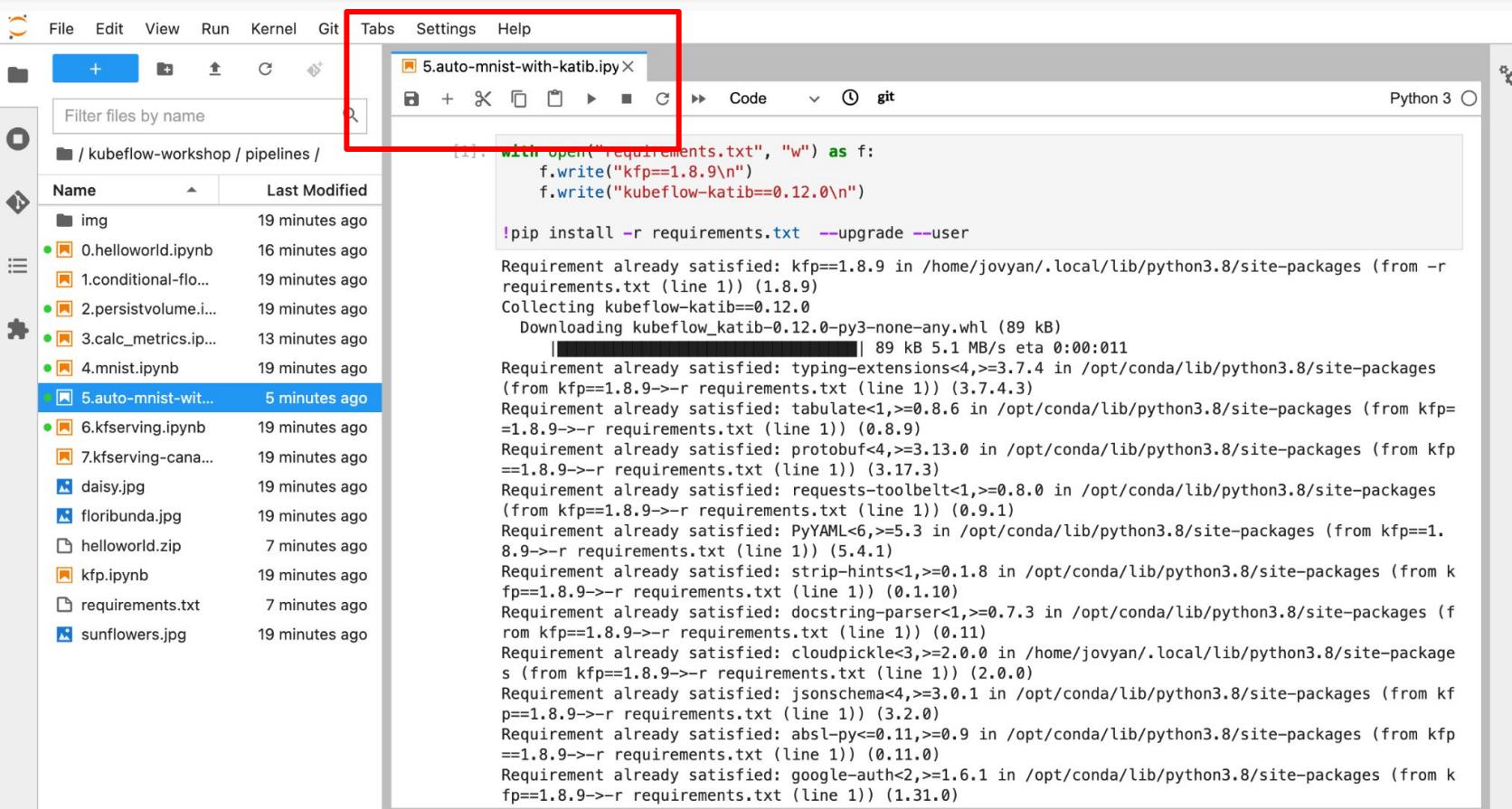
- Input parameters:** text1 (message 1)
- Input artifacts:** None
- Output parameters:** None
- Output artifacts:** main-logs (minio://mipipeline/artifacts/execution-order-pipeline-qvlt5/2021/11/30/execution-order-pipeline-qvlt5-137025724/main.log, message 1)

A large red arrow points upwards from the "Output artifacts" section towards the text "運行結果輸出" (Execution Result Output) in red at the bottom right.

↑
運行結果輸出

Hyperparameter Example

Step5: Hyperparameter tuning with katib (1/4)



File Edit View Run Kernel Git Tabs Settings Help

5.auto-mnist-with-katib.ipynb

```
[1]: with open("requirements.txt", "w") as f:  
    f.write("kfp==1.8.9\n")  
    f.write("kubeflow-katib==0.12.0\n")  
  
!pip install -r requirements.txt --upgrade --user  
  
Requirement already satisfied: kfp==1.8.9 in /home/jovyan/.local/lib/python3.8/site-packages (from -r requirements.txt (line 1)) (1.8.9)  
Collecting kubeflow-katib==0.12.0  
    Downloading kubeflow_katib-0.12.0-py3-none-any.whl (89 kB)  
     ██████████ | 89 kB 5.1 MB/s eta 0:00:011  
Requirement already satisfied: typing-extensions<4,>=3.7.4 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (3.7.4.3)  
Requirement already satisfied: tabulate<1,>=0.8.6 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (0.8.9)  
Requirement already satisfied: protobuf<4,>=3.13.0 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (3.17.3)  
Requirement already satisfied: requests-toolbelt<1,>=0.8.0 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (0.9.1)  
Requirement already satisfied: PyYAML<6,>=5.3 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (5.4.1)  
Requirement already satisfied: strip-hints<1,>=0.1.8 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (0.1.10)  
Requirement already satisfied: docstring-parser<1,>=0.7.3 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (0.11)  
Requirement already satisfied: cloudpickle<3,>=2.0.0 in /home/jovyan/.local/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (2.0.0)  
Requirement already satisfied: jsonschema<4,>=3.0.1 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (3.2.0)  
Requirement already satisfied: absl-py<=0.11,>=0.9 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (0.11.0)  
Requirement already satisfied: google-auth<2,>=1.6.1 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (1.31.0)
```

Step5: Hyperparameter tuning with katib (2/4)

The screenshot shows the Kubeflow interface. On the left, a sidebar navigation bar includes Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP), Pipelines, Runs, Recurring Runs, and Artifacts. The 'Runs' item is currently selected.

The main content area displays the 'Experiments' page for the experiment 'hello1'. It shows an 'Experiment description' section with a 'Recurring run configs' count of 0 active, and a 'Manage' button. Below this is the 'Runs' section, which has tabs for 'Active' (selected) and 'Archived'. It includes buttons for '+ Create run' and '+ Create recurring run', and links for Compare runs, Clone run, and Archive. The 'Active' runs table lists three entries:

Run name	Status	Duration	Pipeline Version	Recurri...	Start time	quotient	remainder
Run of hello-world_version_1	?	-	hello-world_version_1	-	5/17/2022, 9:42:...		
Run of hello-world_version_1	✓	0:06:19	hello-world_version_1	-	5/17/2022, 9:30:...		
Run of hello-world_version_1	✓	0:01:24	hello-world_version_1	-	5/17/2022, 9:23:...	0.000	6.000

A red box highlights the first row of the table.

Step5: Hyperparameter tuning with katib (3/4)

The screenshot shows the Kubeflow interface. On the left, a sidebar menu is visible with the following items:

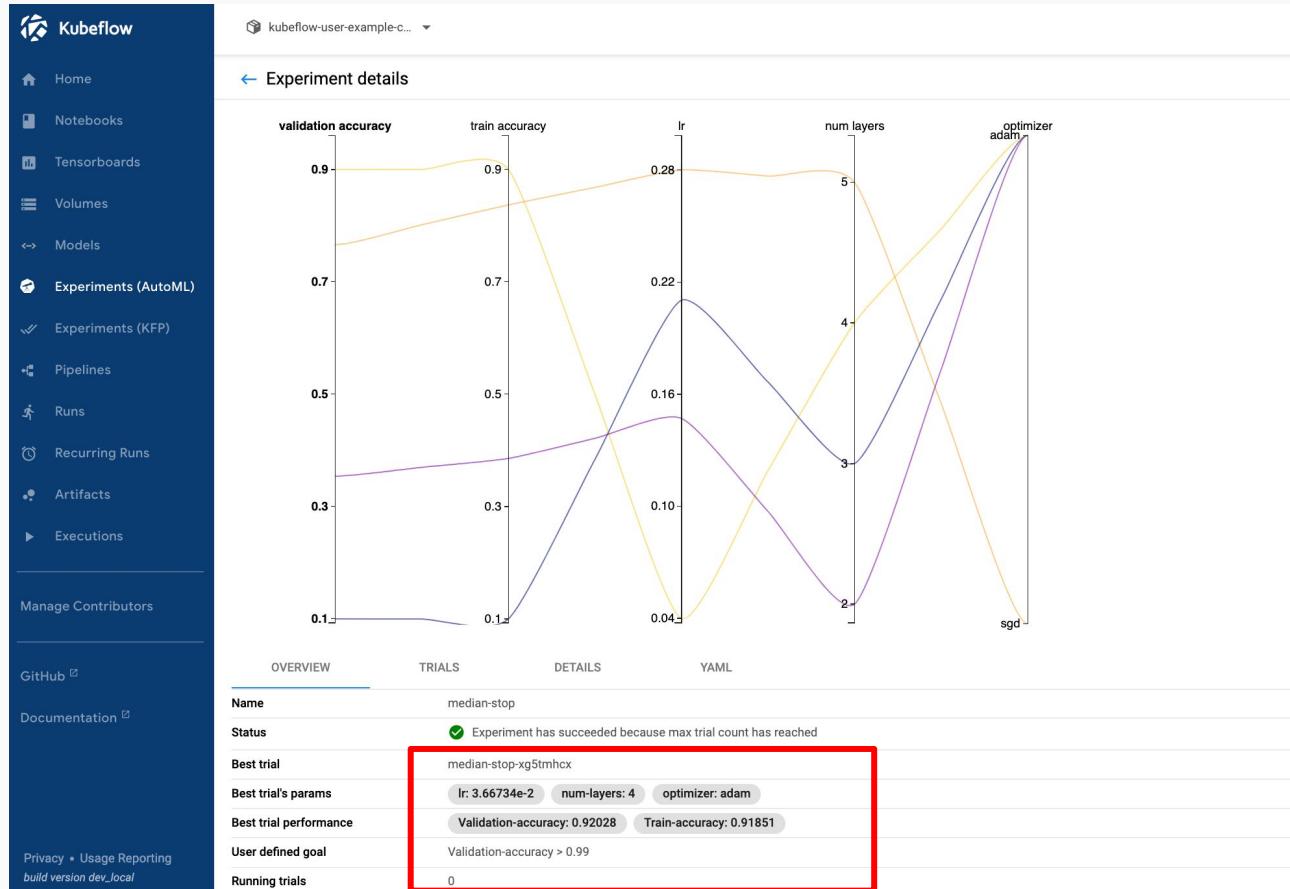
- Home
- Notebooks
- Tensorboards
- Volumes
- Models
 - Experiments (AutoML) (highlighted with a red box)
 - Experiments (KFP)
- Pipelines
- Runs
- Recurring Runs
- Artifacts

The main content area is titled "Experiments". It displays a table with the following columns: Status, Name, Age, Successful trials, Running trials, Failed trials, and Optimal trial. A single experiment entry is shown:

Status	Name	Age	Successful trials	Running trials	Failed trials	Optimal trial
✓	median-stop	8 minutes ago	4	0	0	Validation accuracy: 0.92028

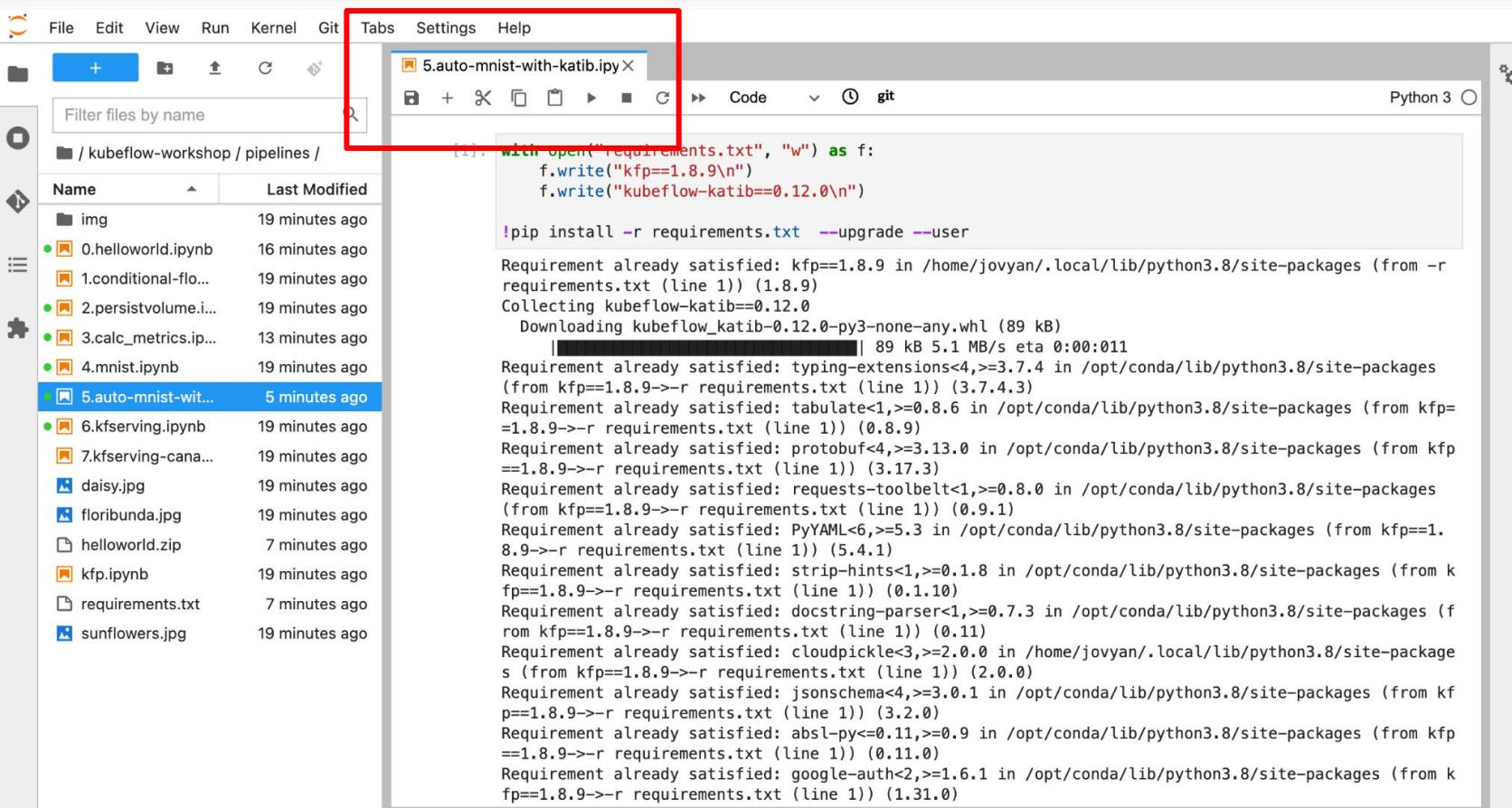
A blue "+ NEW EXPERIMENT" button is located in the top right corner of the experiments table.

Step5: Hyperparameter tuning with kubeflow (4/4)



Kserve Example

Step6: Serving model with Kserve (1/4)



The screenshot shows a Jupyter Notebook interface with a red box highlighting the tab bar and the code editor area. The code editor contains a Python script named `5.auto-mnist-with-katib.ipynb`. The script begins with opening a file for writing and then executing a pip command to install requirements from a requirements.txt file.

```
[1]: with open("requirements.txt", "w") as f:  
    f.write("kfp==1.8.9\n")  
    f.write("kubeflow-katib==0.12.0\n")  
  
!pip install -r requirements.txt --upgrade --user  
  
Requirement already satisfied: kfp==1.8.9 in /home/jovyan/.local/lib/python3.8/site-packages (from -r requirements.txt (line 1)) (1.8.9)  
Collecting kubeflow-katib==0.12.0  
  Downloading kubeflow_katib-0.12.0-py3-none-any.whl (89 kB)  
[██████████] 89 kB 5.1 MB/s eta 0:00:011  
Requirement already satisfied: typing-extensions<4,>=3.7.4 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (3.7.4.3)  
Requirement already satisfied: tabulate<1,>=0.8.6 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (0.8.9)  
Requirement already satisfied: protobuf<4,>=3.13.0 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (3.17.3)  
Requirement already satisfied: requests-toolbelt<1,>=0.8.0 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (0.9.1)  
Requirement already satisfied: PyYAML<6,>=5.3 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (5.4.1)  
Requirement already satisfied: strip-hints<1,>=0.1.8 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (0.1.10)  
Requirement already satisfied: docstring-parser<1,>=0.7.3 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (0.11)  
Requirement already satisfied: cloudpickle<3,>=2.0.0 in /home/jovyan/.local/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (2.0.0)  
Requirement already satisfied: jsonschema<4,>=3.0.1 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (3.2.0)  
Requirement already satisfied: absl-py<=0.11,>=0.9 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (0.11.0)  
Requirement already satisfied: google-auth<2,>=1.6.1 in /opt/conda/lib/python3.8/site-packages (from kfp==1.8.9->-r requirements.txt (line 1)) (1.31.0)
```

tep6: Serving model with Kserve (2/4)

The screenshot shows the Kubeflow interface for managing experiments and runs. On the left, a sidebar lists various options: Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP), Pipelines, Runs, Recurring Runs, and Artifacts. The 'Runs' option is selected.

The main area displays the 'Experiments' section for 'hello1'. It includes a 'Recurring run configs' section with a 'Manage' link, an 'Experiment description' field, and a 'Create run' button. Below this is the 'Runs' section, which has tabs for 'Active' (selected) and 'Archived'. It features a 'Create recurring run' button and links for 'Compare runs', 'Clone run', and 'Archive'. The 'Active' runs table has columns: Run name, Status, Duration, Pipeline Version, Recurri..., Start time, quotient, and remainder. The first row is highlighted with a red border.

Run name	Status	Duration	Pipeline Version	Recurri...	Start time	quotient	remainder
Run of hello-world_version_1	?	-	hello-world_version_1	-	5/17/2022, 9:42:...		
Run of hello-world_version_1	✓	0:06:19	hello-world_version_1	-	5/17/2022, 9:30:...		
Run of hello-world_version_1	✓	0:01:24	hello-world_version_1	-	5/17/2022, 9:23:...	0.000	6.000

Step5: Hyperparameter tuning with katib (3/4)

The screenshot shows the Kubeflow interface. On the left, a dark sidebar lists various components: Home, Notebooks, Tensorboards, Volumes, Models (which is selected and highlighted with a red box), Experiments (AutoML), Experiments (KFP), Pipelines, Runs, Recurring Runs, and Artifacts. The main content area is titled "Model Servers" and displays a table of model servers. The table has columns: Status, Name, Age, Predictor, Runtime, Protocol, Storage URI, and Actions (represented by icons for edit and delete). One row is highlighted with a red box: "tensorflow-sample" (Status: green checkmark), "2 minutes ago" (Age), "Tensorflow" (Predictor), "1.14.0" (Runtime), "gs://kfserving-samples/models/tensorflow/flo..." (Storage URI), and edit/delete icons.

Status	Name	Age	Predictor	Runtime	Protocol	Storage URI	Actions
✓	tensorflow-sample	2 minutes ago	Tensorflow	1.14.0		gs://kfserving-samples/models/tensorflow/flo...	

Step5: Hyperparameter tuning with katib (4/4)

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, View, Run, Kernel, Git, Tabs, Settings, Help.
- Left Sidebar:** Shows a file tree with the current directory being `/kubeflow-workshop / pipelines /`. The list includes files like `img`, `0.helloworld.ipynb`, `1.conditional-flo...`, etc., up to `tf-flower.zip`.
- Current Notebook:** `6.kfserving.ipynb` is open.
- Code Editor:** The code cell [4] contains Python code for sending a REST API request to an Istio-ingressgateway. A red box highlights this code block.

```
[4]: ## the following example use python's request to send restapi requests
import base64
import json
import requests

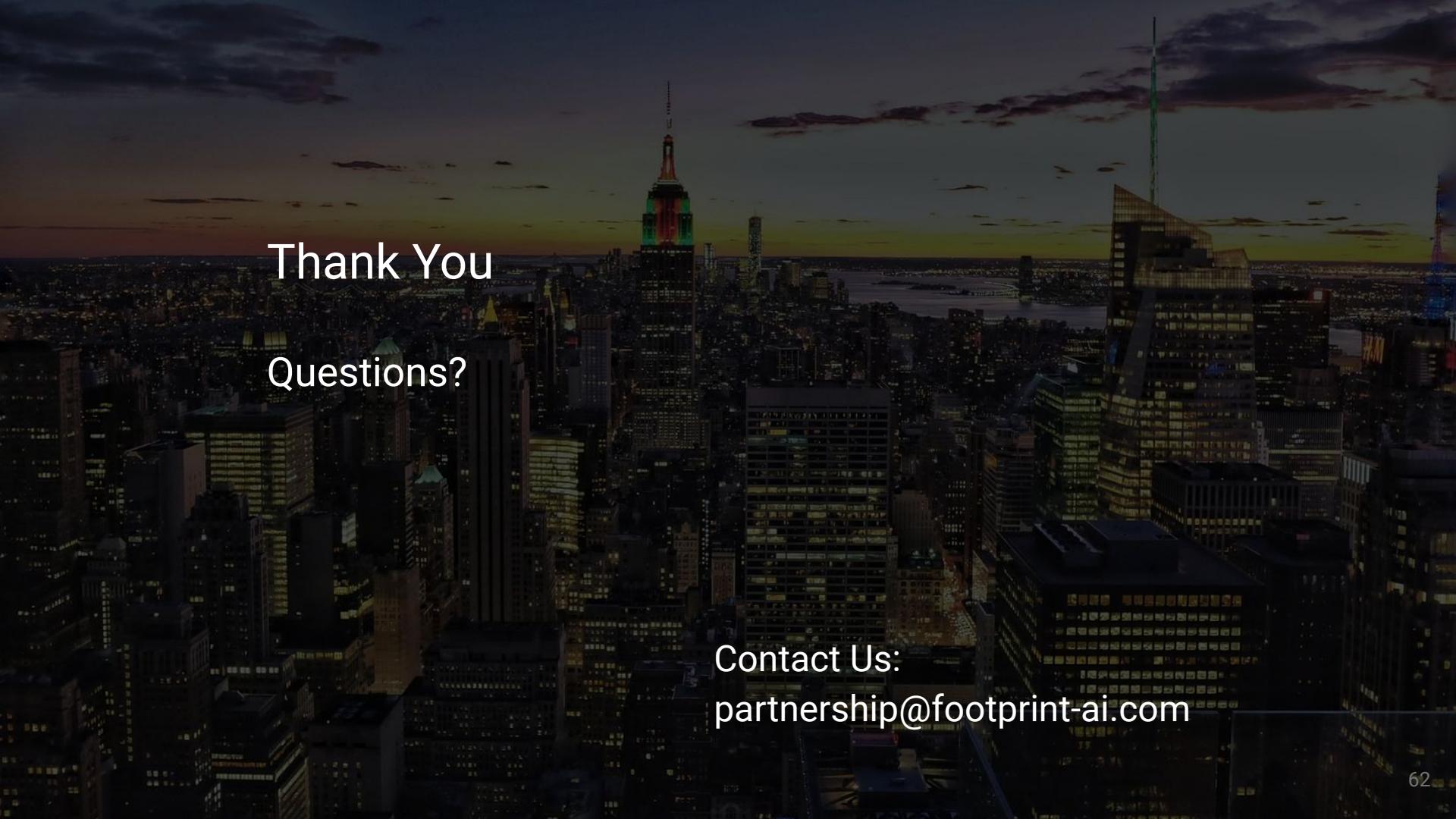
with open('floribunda.jpg', 'rb') as f:
    image_content = f.read()
    image_64_encode = base64.encodebytes(image_content).decode('utf-8')
headers = {"Cookie": "authservice_session=MTY1Mjc5MzE5M3x0d3dBTkZkUk4xZER0MGRZVmpZMldsTkpWVFUzU2s5V1dF
    "Host": "tensorflow-sample.kubeflow-user-example-com.example.com"
payload = {"instances": [{"image_bytes": {"b64": image_64_encode}, "key": "1"}]}
resp = requests.post('http://istio-ingressgateway.istio-system/tensorflow-sample:predict', he
print(resp.text)

{
    "predictions": [
        {
            "scores": [1.30671893e-07, 3.01086693e-08, 0.814807534, 9.6436537e-08, 0.185192183, 3.4390
2293e-08],
            "prediction": 2,
            "key": "1"
        }
    ]
}
```
- Bottom Cell:** Cell [1] contains a comment: `## the following example use curl to send restapi requests`.
- Top Bar:** Shows tabs for Code, Lighthouse, and a git status indicator.
- Right Side:** Shows the browser tab for `6.kfserving.ipynb` and the Network tab of the developer tools showing a cookie table with one entry: `authservice_session`.

Q&A

One minute takeaway

- Implemented remote development, training, and deployment.
- Pipeline-based development model
- Distributed model training and optimization
- Model deployment



Thank You

Questions?

Contact Us:
partnership@footprint-ai.com

Additional materials

- Documentations
 - <https://www.kubeflow.org/>
- Kubeflow materials
 - <https://www.kubeflow.org/>