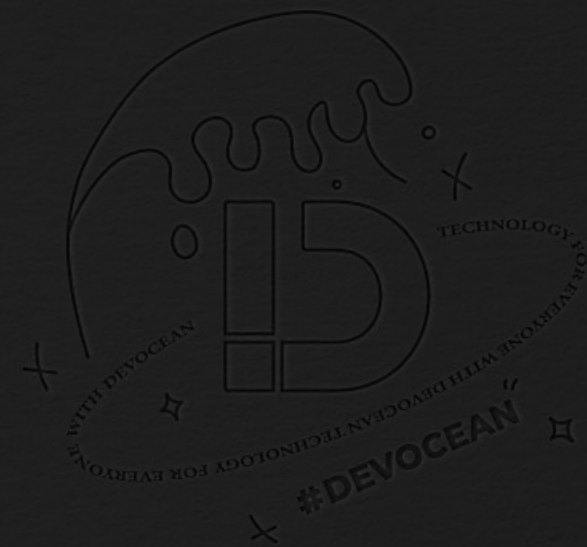


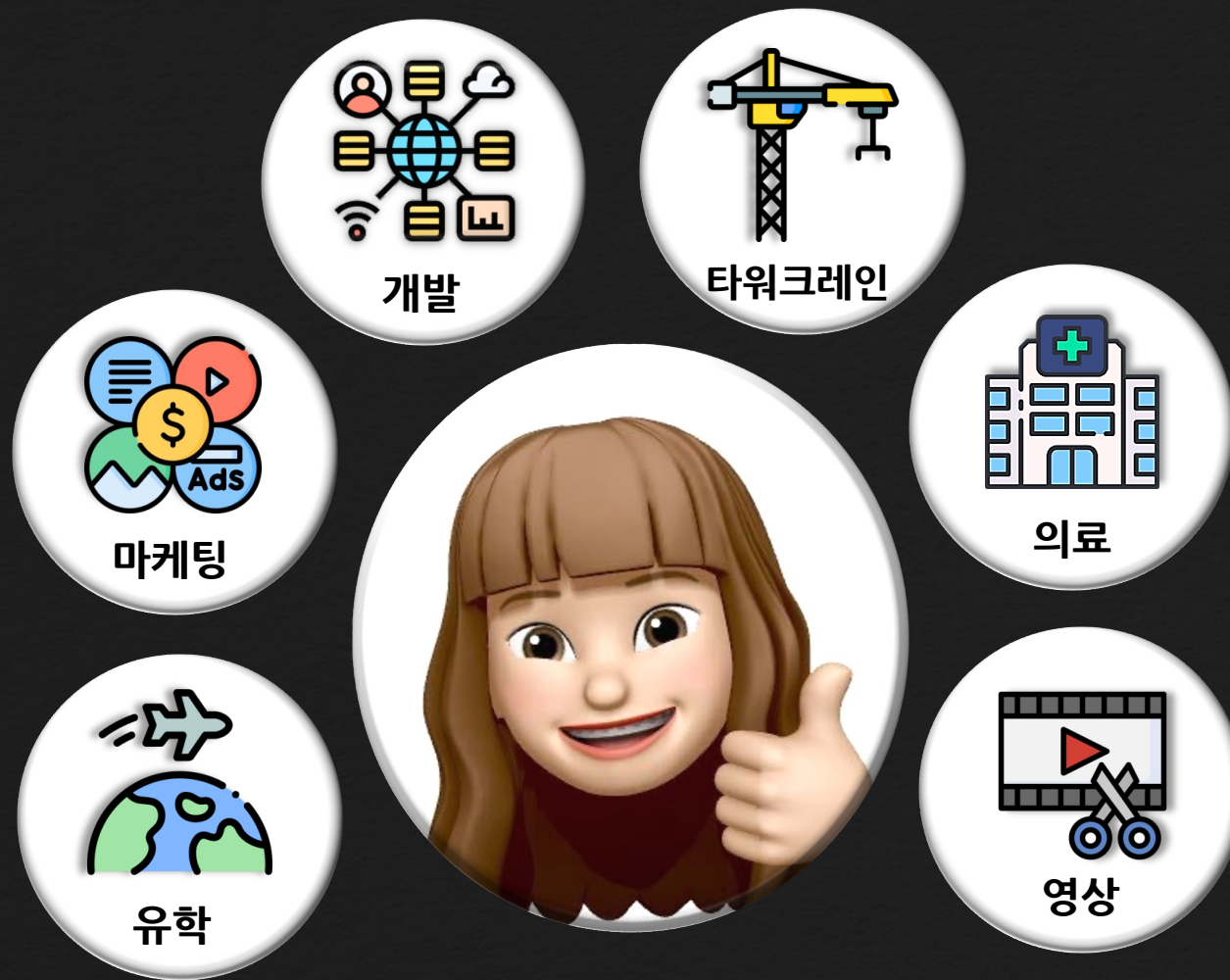


Kubeflow 알아보기

김예은



경험에 경험을 더하는 개발자, 김예은



목차

01 Kubeflow란?

02 Kubeflow Components

- Dashboard
- Notebooks
- Pipeline
- Training Operator
- Katib
- KServe

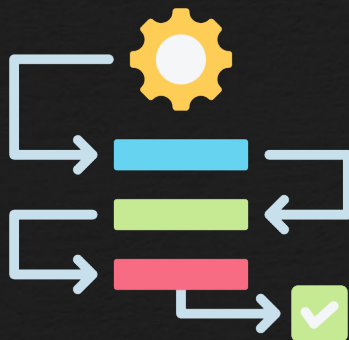
03 MLOps



01 Kubeflow란?



Kubernetes



ML Workflow



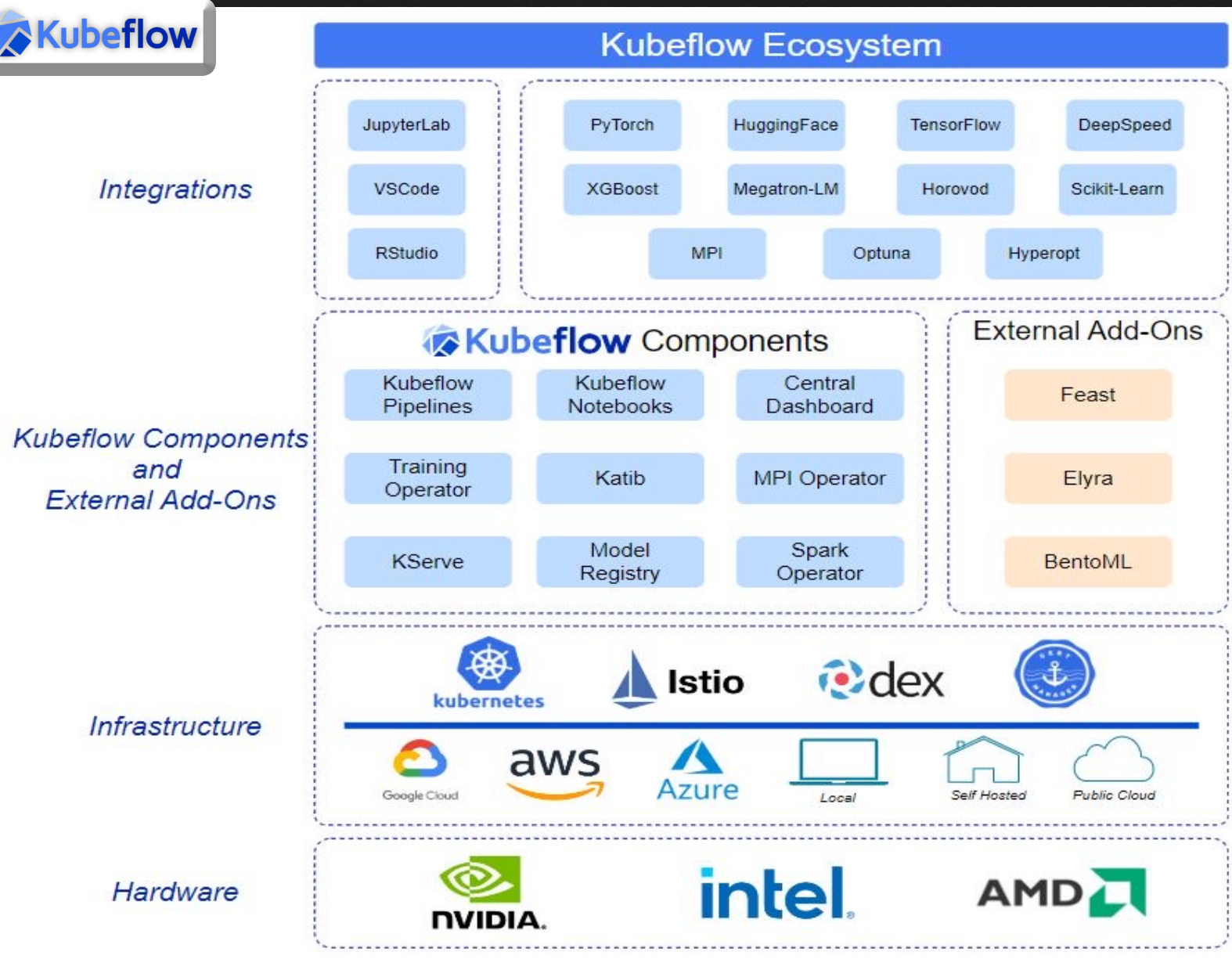
- Kubernetes 환경에서 ML 라이프사이클의 각 단계를 다룰 수 있는 오픈소스 프로젝트
- 개발자, 데이터 사이언티스트, ML 엔지니어 등 모두가 모델 구축부터 테스트, 배포까지 ML 라이프사이클의 모든 부분을 처리할 수 있도록 서비스를 제공

01 Kubeflow란?



Why?

01 Kubeflow란?



02 Kubeflow Components



Pipeline
Kubeflow Pipeline



Notebooks
Kubeflow Notebooks



Dashboard
Kubeflow Central Dashboard



AutoML
Katib

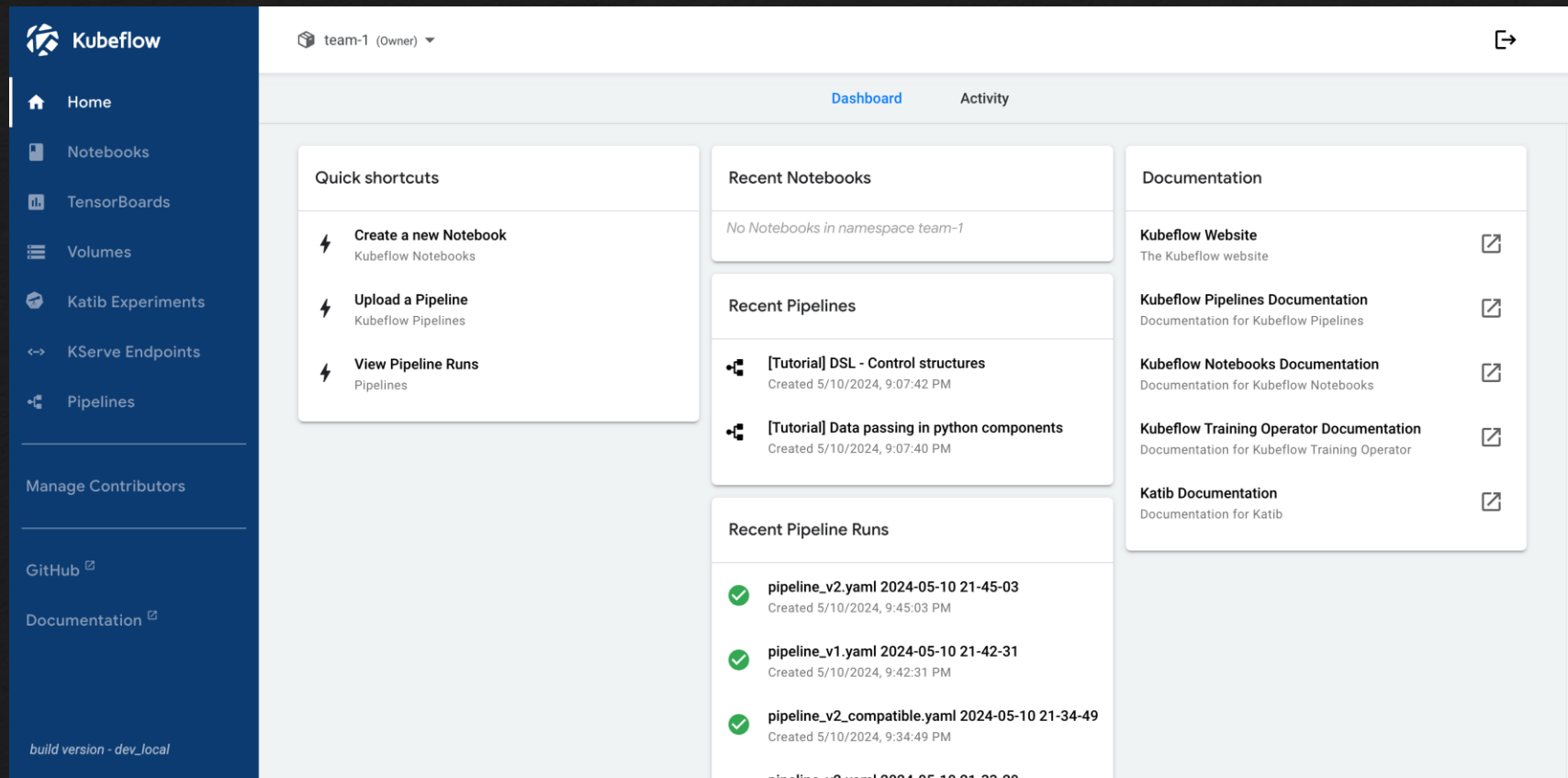


Model Training
Kubeflow Training Operator



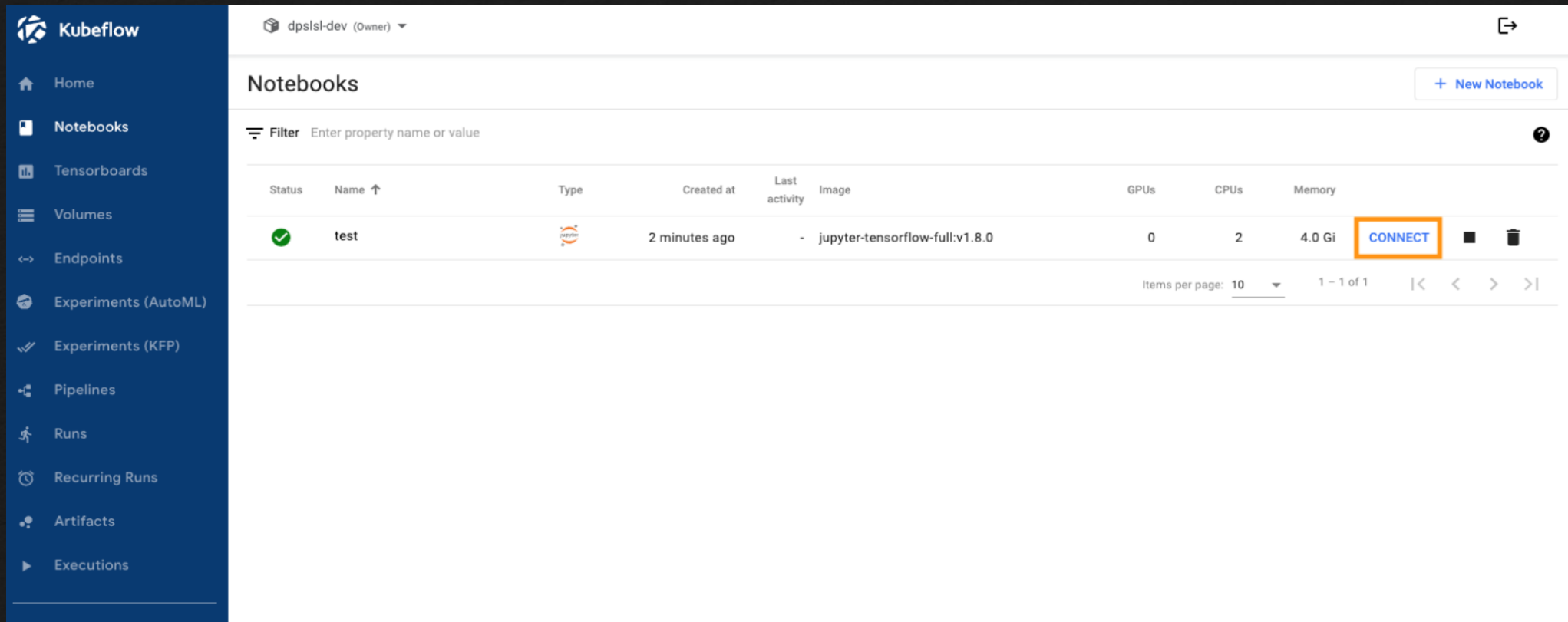
Model Serving
KServe

02 Kubeflow Components – Central Dashboard



- 클러스터에서 실행되는 구성 요소의 실행 및 내용들을 확인할 수 있도록 웹 인터페이스를 제공
- 프로필을 이용하여, 사용자를 등록 및 삭제하여 관리하고 사용자에게 읽기, 쓰기 등의 권한을 부여

02 Kubeflow Components - Kubeflow Notebooks



The screenshot displays the Kubeflow web interface. On the left is a dark blue sidebar with navigation links: Home, Notebooks, Tensorboards, Volumes, Endpoints, Experiments (AutoML), Experiments (KFP), Pipelines, Runs, Recurring Runs, Artifacts, and Executions. The main panel is titled 'Notebooks' and shows a table with one entry:

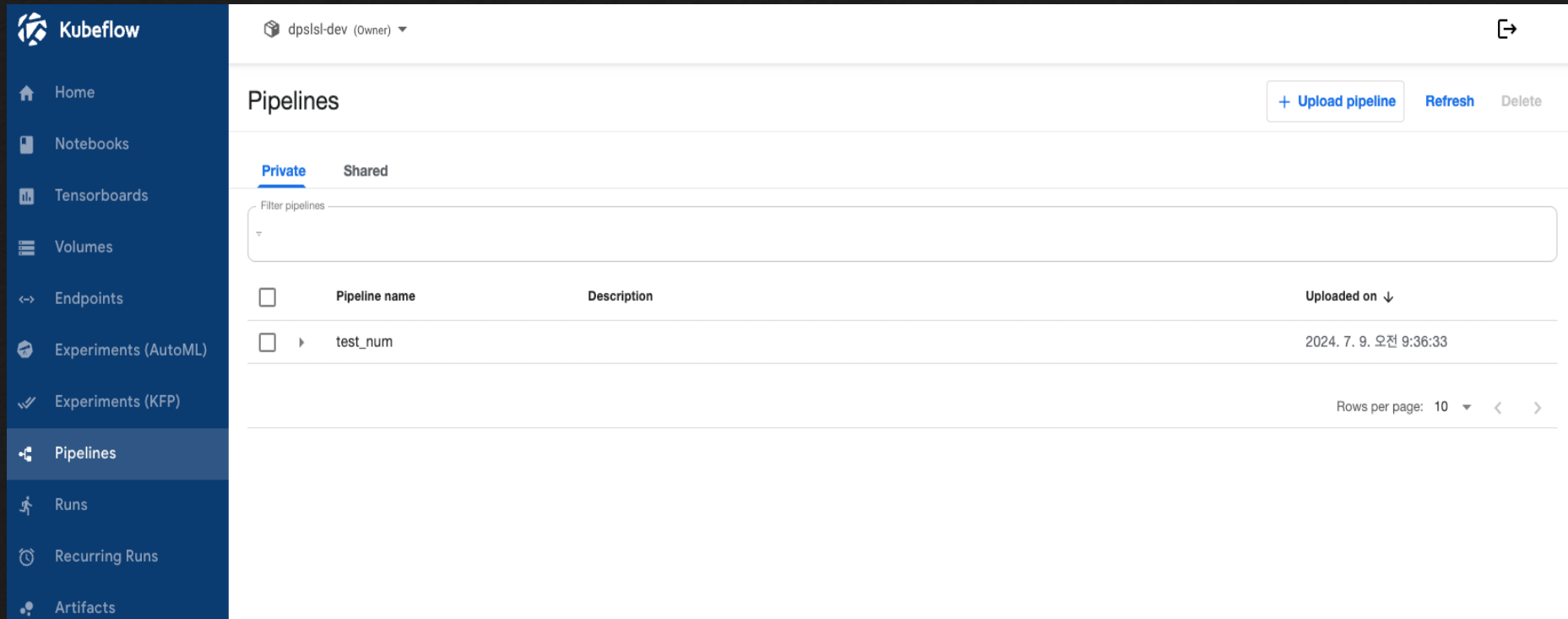
Status	Name ↑	Type	Created at	Last activity	Image	GPUs	CPUs	Memory	
✓	test		2 minutes ago	-	jupyter-tensorflow-full:v1.8.0	0	2	4.0 Gi	CONNECT  

At the bottom right of the table, there is a pagination control showing 'Items per page: 10' and '1 - 1 of 1'.



- Kubernetes 클러스터 내에서 웹 기반 개발 환경을 실행할 수 있는 방법을 제공
- JupyterLab, RStudio 및 Visual Studio Code를 기본적으로 지원(유형 및 패키지는 Docker 이미지로 선택 가능)

02 Kubeflow Components – Kubeflow Pipeline



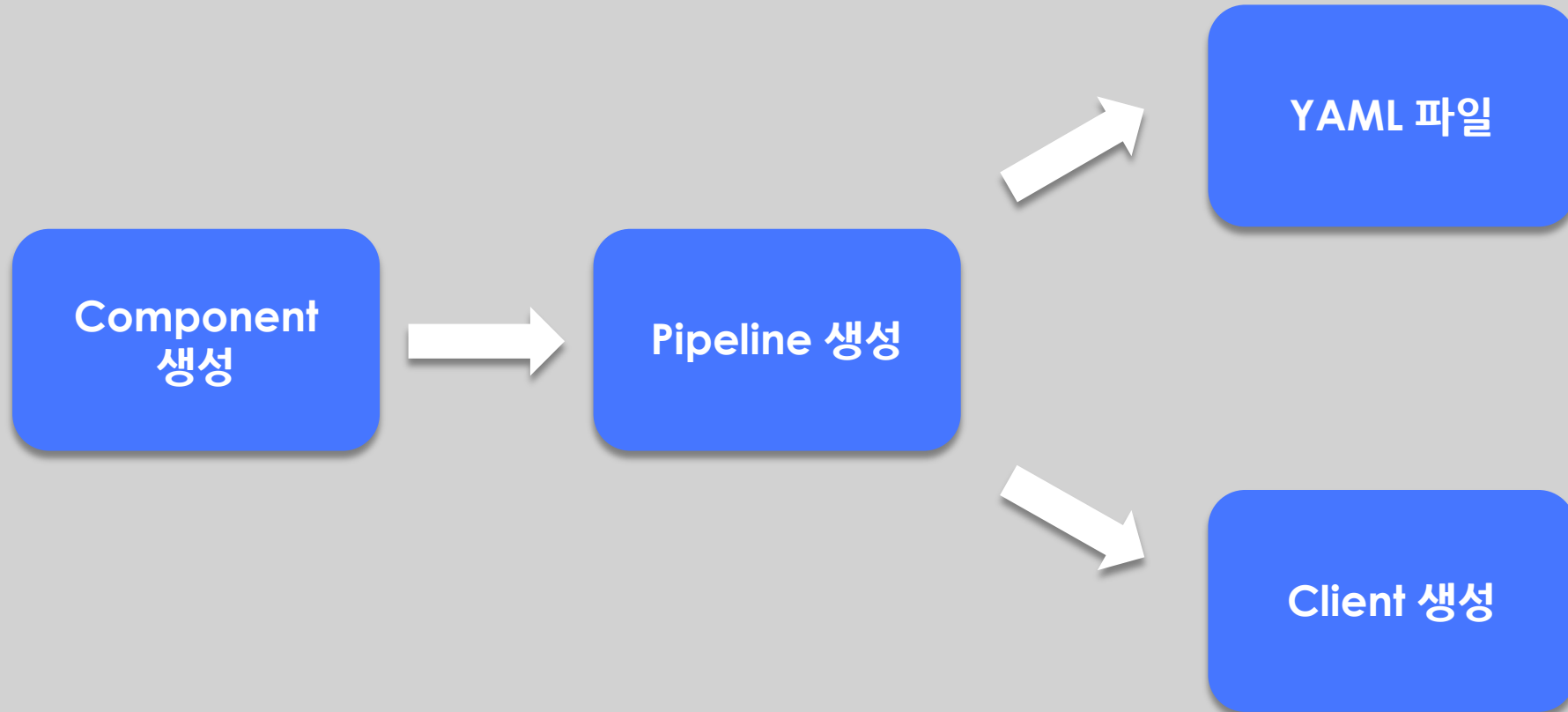
The screenshot displays the Kubeflow Pipelines web interface. On the left is a dark blue sidebar with navigation links: Home, Notebooks, Tensorboards, Volumes, Endpoints, Experiments (AutoML), Experiments (KFP), Pipelines (selected), Runs, Recurring Runs, and Artifacts. The main content area is titled 'Pipelines' and shows the user 'dpslsl-dev (Owner)'. It includes buttons for '+ Upload pipeline', 'Refresh', and 'Delete'. Below these are tabs for 'Private' and 'Shared'. A search bar labeled 'Filter pipelines' is present. A table lists the pipelines with columns for selection, Pipeline name, Description, and Uploaded on. One pipeline, 'test_num', is listed with an upload time of '2024. 7. 9. 오전 9:36:33'. At the bottom right, it shows 'Rows per page: 10' with navigation arrows.

	Pipeline name	Description	Uploaded on ↓
☐	test_num		2024. 7. 9. 오전 9:36:33

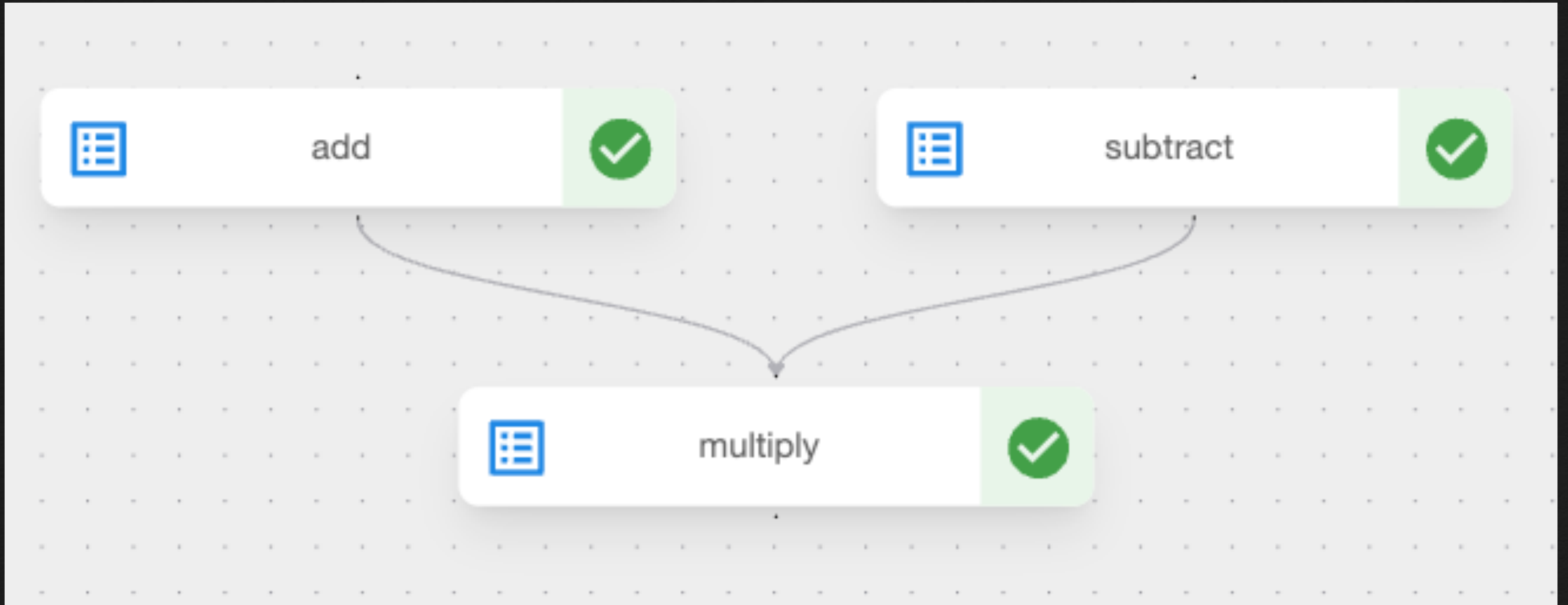


- Kubernetes를 사용하여 이식과 확장이 가능한 ML 워크플로우를 구축, 배포하기 위한 플랫폼
- KFP Python SDK를 사용하거나, YAML 파일을 이용하여 누구나 쉽게 Pipeline 구축 가능
- 사용자가 커스텀하거나, 기존의 ML 구성 요소들을 활용해서 다양하게 사용 가능
- 파이프라인 정의, 실행, 실험 등을 쉽게 관리하고 시각화로 확인 가능

02 Kubeflow Components – Kubeflow Pipeline



02 Kubeflow Components – Kubeflow Pipeline



02 Kubeflow Components – Kubeflow Pipeline

The screenshot displays the JupyterLab environment with a file browser on the left and a code editor on the right. The file browser shows a directory structure with files like 'lost+found' and 'num_pipeline.ipynb'. The code editor shows the contents of 'num_pipeline.ipynb', which defines a Kubeflow Pipeline using the DSL. The pipeline consists of three steps: 'add', 'subtract', and 'multiply'. The 'add' and 'subtract' steps are connected to the 'multiply' step, indicating a sequential flow. The pipeline is compiled and saved as 'num_pipeline.yaml'.

```
[2]: from kfp import dsl
import kfp

@dsl.component()
def add(num1: int, num2: int) -> int:
    result = num1 + num2
    return result

@dsl.component()
def subtract(num1: int, num2: int) -> int:
    result = num1 - num2
    return result

@dsl.component()
def multiply(num1: int, num2: int) -> int:
    result = num1 * num2
    return result

@dsl.pipeline(name="num pipeline")
def pipeline(num1: int, num2: int):
    step1 = add(num1=num1, num2=num2)
    step2 = subtract(num1=num1, num2=num2)
    step3 = multiply(num1=step1.output, num2=step2.output)

pipeline_func = pipeline
pipeline_filename = 'num_pipeline.yaml'
kfp.compiler.Compiler().compile(pipeline_func, pipeline_filename)
```

The DAG visualization on the right shows the pipeline's execution flow. It features three nodes: 'add', 'subtract', and 'multiply'. The 'add' and 'subtract' nodes are connected to the 'multiply' node, indicating that the output of 'add' is used as the first input to 'multiply', and the output of 'subtract' is used as the second input to 'multiply'. Each node has a green checkmark, indicating successful execution.

02 Kubeflow Components – Kubeflow Pipeline

File Edit View Run Kernel Git Tabs Settings Help

+

+

+

+

+

Filter files by name

Name

Last Modified

lost+found 3 minutes ago

num_pipeline.ipynb a minute ago

num_pipeline.ipynb

Components

```
[2]: from kfp import dsl
import kfp

@dsl.component()
def add(num1: int, num2: int) -> int:
    result = num1 + num2
    return result

@dsl.component()
def subtract(num1: int, num2: int) -> int:
    result = num1 - num2
    return result

@dsl.component()
def multiply(num1: int, num2: int) -> int:
    result = num1 * num2
    return result

@dsl.pipeline(name="num pipeline")
def pipeline(num1: int, num2: int):
    step1 = add(num1=num1, num2=num2)
    step2 = subtract(num1=num1, num2=num2)
    step3 = multiply(num1=step1.output, num2=step2.output)

pipeline_func = pipeline
pipeline_filename = 'num_pipeline.yaml'
kfp.compiler.Compiler().compile(pipeline_func, pipeline_filename)
```

add

subtract

multiply

Python 3 (ipykernel)

[]:

02 Kubeflow Components – Kubeflow Pipeline

The screenshot shows a JupyterLab environment with a file browser on the left and a code editor on the right. The file browser shows a directory structure with a file named `num_pipeline.ipynb` selected. The code editor displays the following Python code:

```
[2]: from kfp import dsl
import kfp

@dsl.component()

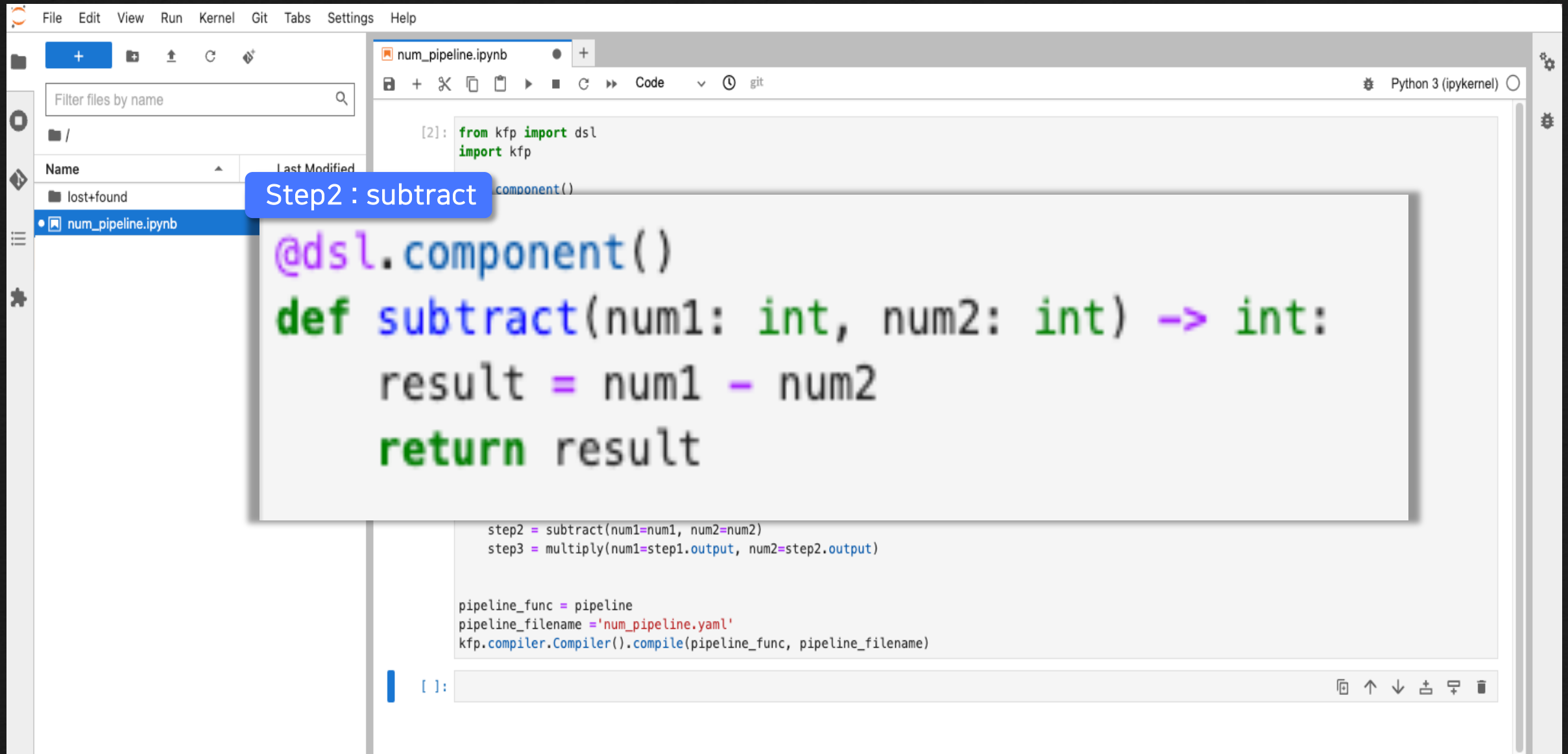
@dsl.component()
def add(num1: int, num2: int) -> int:
    result = num1 + num2
    return result

step2 = subtract(num1=num1, num2=num2)
step3 = multiply(num1=step1.output, num2=step2.output)

pipeline_func = pipeline
pipeline_filename = 'num_pipeline.yaml'
kfp.compiler.Compiler().compile(pipeline_func, pipeline_filename)
```

A blue callout box with the text "Step1 : add" is positioned over the first `@dsl.component()` decorator in the code.

02 Kubeflow Components – Kubeflow Pipeline



The screenshot shows a JupyterLab environment with a file browser on the left and a code editor on the right. The file browser shows a directory structure with a file named `num_pipeline.ipynb` selected. The code editor displays the content of `num_pipeline.ipynb`, which includes a Python function `subtract` decorated with `@dsl.component()`. A blue callout box highlights the text "Step2 : subtract".

```
[2]: from kfp import dsl
import kfp

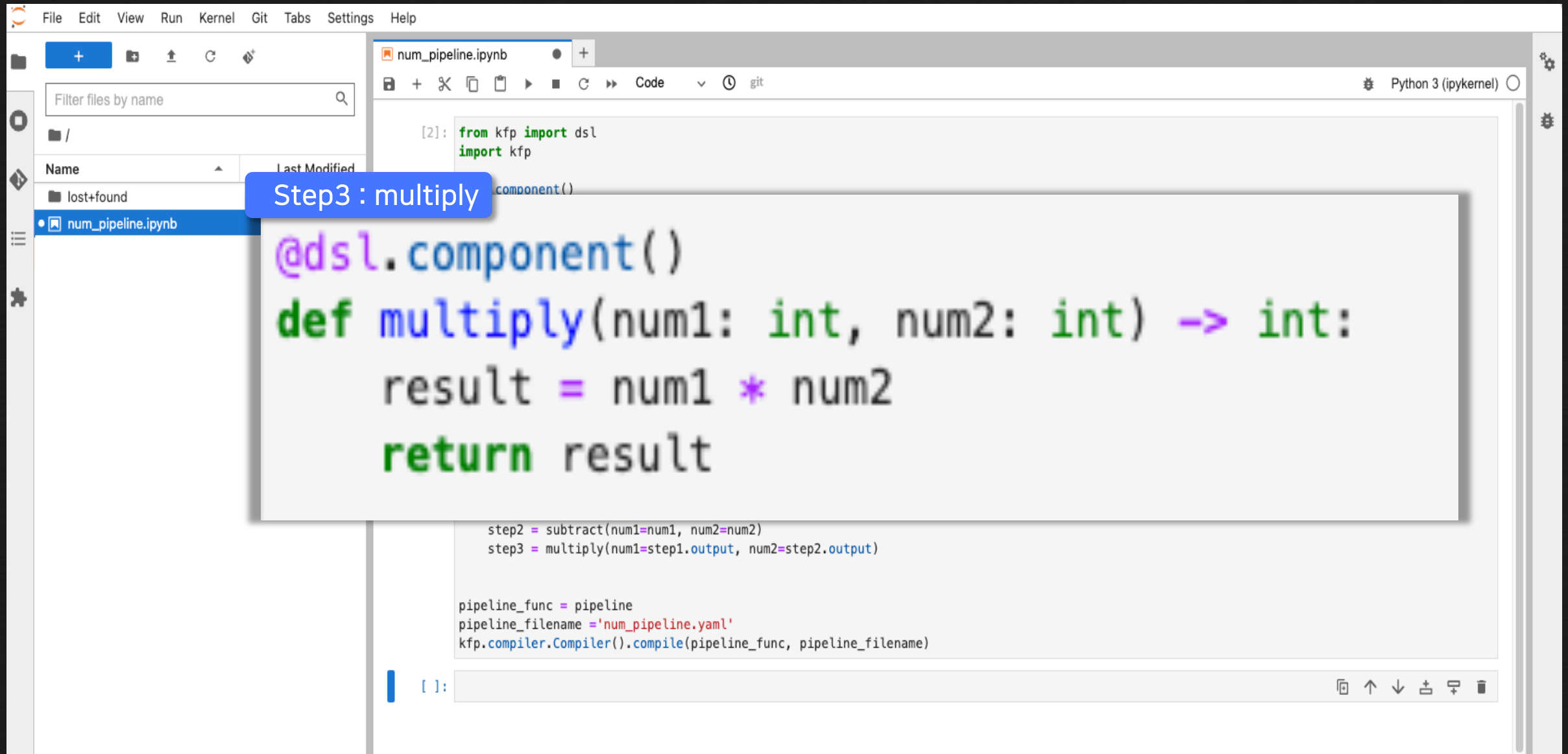
@dsl.component()
def subtract(num1: int, num2: int) -> int:
    result = num1 - num2
    return result

step2 = subtract(num1=num1, num2=num2)
step3 = multiply(num1=step1.output, num2=step2.output)

pipeline_func = pipeline
pipeline_filename = 'num_pipeline.yaml'
kfp.compiler.Compiler().compile(pipeline_func, pipeline_filename)
```

[]:

02 Kubeflow Components – Kubeflow Pipeline



The screenshot shows a JupyterLab environment with a file browser on the left and a code editor on the right. The file browser shows a directory structure with a file named `num_pipeline.ipynb` selected. The code editor displays the following Python code:

```
[2]: from kfp import dsl
import kfp

component()

@dsl.component()
def multiply(num1: int, num2: int) -> int:
    result = num1 * num2
    return result

step2 = subtract(num1=num1, num2=num2)
step3 = multiply(num1=step1.output, num2=step2.output)

pipeline_func = pipeline
pipeline_filename = 'num_pipeline.yaml'
kfp.compiler.Compiler().compile(pipeline_func, pipeline_filename)
```

A blue callout box with the text "Step3 : multiply" is overlaid on the code editor, highlighting the `multiply` function definition.

02 Kubeflow Components – Kubeflow Pipeline

The screenshot displays a JupyterLab environment with a file browser on the left and a code editor in the center. The file browser shows a directory structure with files like 'lost+found' and 'num_pipeline.ipynb'. The code editor shows the following Python code:

```
[2]: from kfp import dsl
import kfp

@dsl.component()
def add(num1: int, num2: int) -> int:
    result = num1 + num2
    return result

@dsl.component()
def subtract(num1: int, num2: int) -> int:
    result = num1 - num2
    return result

@dsl.component()
def multiply(num1: int, num2: int) -> int:
    result = num1 * num2
    return result

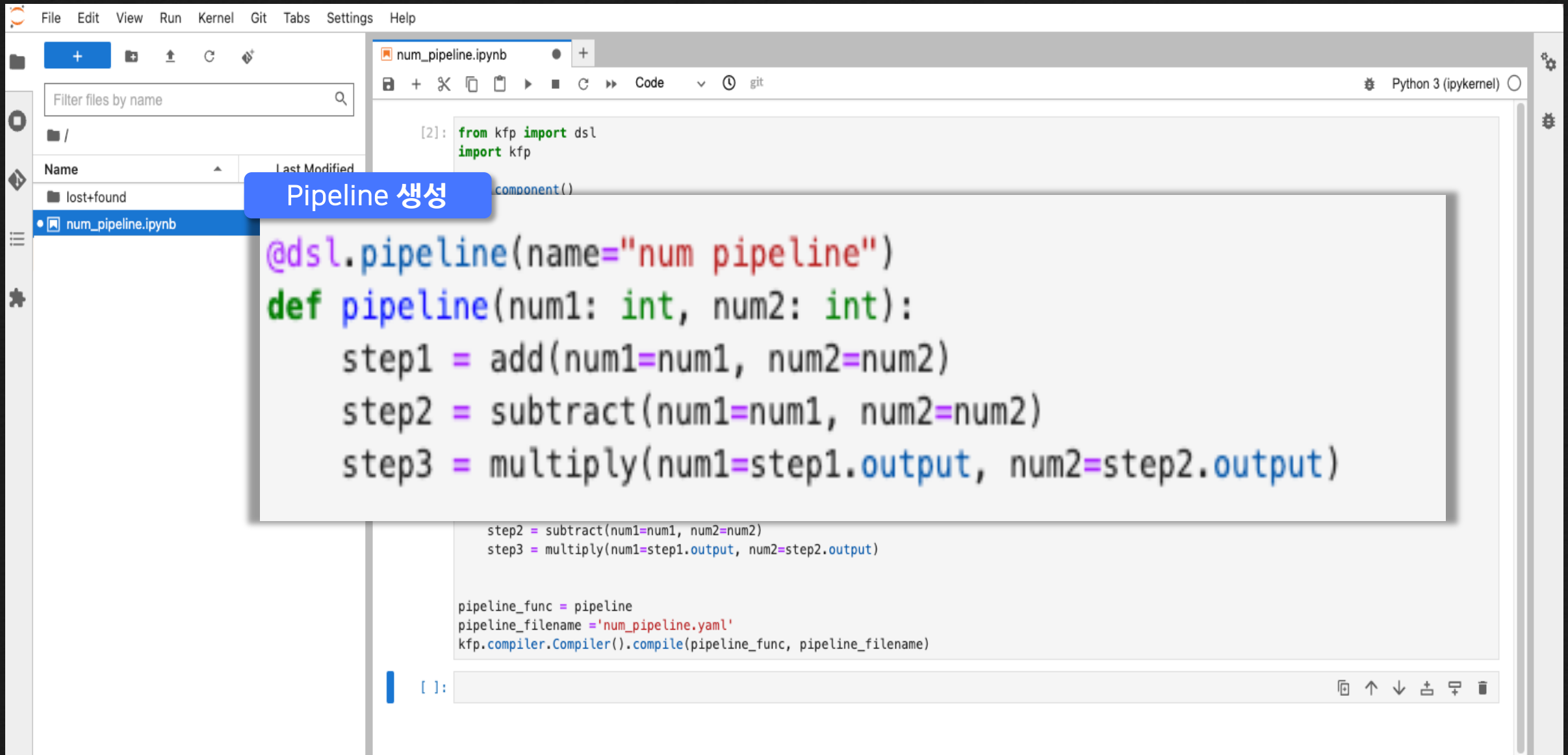
@dsl.pipeline(name="num pipeline")
def pipeline(num1: int, num2: int):
    step1 = add(num1=num1, num2=num2)
    step2 = subtract(num1=num1, num2=num2)
    step3 = multiply(num1=step1.output, num2=step2.output)

pipeline_func = pipeline
pipeline_filename = 'num_pipeline.yaml'
kfp.compiler.Compiler().compile(pipeline_func, pipeline_filename)
```

On the right side of the code editor, a visual representation of the pipeline is shown. It consists of three nodes: 'add', 'subtract', and 'multiply'. The 'add' and 'subtract' nodes are connected to the 'multiply' node, indicating that the output of 'add' is used as the first input to 'multiply', and the output of 'subtract' is used as the second input to 'multiply'. Each node has a green checkmark next to it, indicating that it has been successfully executed.

Pipeline

02 Kubeflow Components – Kubeflow Pipeline



The screenshot shows a JupyterLab environment with a file browser on the left and a code editor on the right. The file browser shows a directory structure with a file named `num_pipeline.ipynb`. The code editor displays the following Python code:

```
[2]: from kfp import dsl
import kfp

component()

@dsl.pipeline(name="num pipeline")
def pipeline(num1: int, num2: int):
    step1 = add(num1=num1, num2=num2)
    step2 = subtract(num1=num1, num2=num2)
    step3 = multiply(num1=step1.output, num2=step2.output)

    step2 = subtract(num1=num1, num2=num2)
    step3 = multiply(num1=step1.output, num2=step2.output)

    pipeline_func = pipeline
    pipeline_filename = 'num_pipeline.yaml'
    kfp.compiler.Compiler().compile(pipeline_func, pipeline_filename)
```

A blue callout box with the text "Pipeline 생성" (Pipeline Creation) is overlaid on the code editor, highlighting the pipeline definition section.

02 Kubeflow Components – Kubeflow Pipeline

The screenshot displays a JupyterLab environment with a file browser on the left and a code editor on the right. The file browser shows a directory with files 'lost+found' and 'num_pipeline.ipynb'. The code editor is open to 'num_pipeline.ipynb' and shows the following Python code:

```
[2]: from kfp import dsl
import kfp

@dsl.component()
def add(num1: int, num2: int) -> int:
    result = num1 + num2
    return result

@dsl.component()
def subtract(num1: int, num2: int) -> int:
    result = num1 - num2
    return result

@dsl.component()
def multiply(num1: int, num2: int) -> int:
    result = num1 * num2
    return result

@dsl.pipeline(name="num pipeline")
def pipeline(num1: int, num2: int):
    step1 = add(num1=num1, num2=num2)
    step2 = subtract(num1=num1, num2=num2)
    step3 = multiply(num1=step1.output, num2=step2.output)

pipeline_func = pipeline
pipeline_filename = 'num_pipeline.yaml'
kfp.compiler.Compiler().compile(pipeline_func, pipeline_filename)
```

Below the code editor, the text "YAML 생성" (YAML Generation) is visible. The right side of the interface shows a DAG visualization of the pipeline. It consists of three nodes: 'add', 'subtract', and 'multiply'. The 'add' and 'subtract' nodes are connected to the 'multiply' node, indicating that the output of the 'add' step is used as the first input for the 'multiply' step, and the output of the 'subtract' step is used as the second input for the 'multiply' step. All three nodes have a green checkmark, indicating they are successful.

02 Kubeflow Components – Kubeflow Pipeline

The screenshot displays the JupyterLab environment for developing a Kubeflow Pipeline. On the left, a file browser shows the current directory with files like 'lost+found' and 'num_pipeline.ipynb'. The main area is divided into a code editor and a visual pipeline view.

Code Editor: The code defines a pipeline component 'add' and a pipeline function 'pipeline'.

```
[2]: from kfp import dsl
import kfp

@dsl.component()
def add(num1: int, num2: int) -> int:
    result = num1 + num2
    return result

...

@dsl.pipeline(name="num pipeline")
def pipeline(num1: int, num2: int):
    step1 = add(num1=num1, num2=num2)
    step2 = subtract(num1=num1, num2=num2)
    step3 = multiply(num1=step1.output, num2=step2.output)

pipeline_func = pipeline
pipeline_filename = 'num_pipeline.yaml'
kfp.compiler.Compiler().compile(pipeline_func, pipeline_filename)
```

Visual Pipeline View: The right side shows a visual representation of the pipeline with two components, 'add' and 'subtract', each marked with a green checkmark. Arrows indicate the flow of data between these components.

YAML 파일 생성 (YAML File Generation): A blue callout box highlights the code used to generate the pipeline's YAML file:

```
pipeline_func = pipeline
pipeline_filename = 'num_pipeline.yaml'
kfp.compiler.Compiler().compile(pipeline_func, pipeline_filename)
```

02 Kubeflow Components – Kubeflow Pipeline

The screenshot displays a JupyterLab environment with a file browser on the left and a code editor on the right. The file browser shows a directory structure with files like `lost+found`, `num_pipeline.ipynb`, and `Y: num_pipeline.yaml`. The code editor shows a Python script defining a Kubeflow Pipeline using the DSL. The script defines three components: `add`, `subtract`, and `multiply`, and then defines a pipeline that uses these components. The pipeline is named `num pipeline` and is defined as a function `pipeline` that takes `num1` and `num2` as inputs. The pipeline consists of three steps: `step1` (add), `step2` (subtract), and `step3` (multiply). The output of `step1` is passed to `step3`, and the output of `step2` is also passed to `step3`. The pipeline is then compiled and saved to a file named `num_pipeline.yaml`.

```
[2]: from kfp import dsl
import kfp

@dsl.component()
def add(num1: int, num2: int) -> int:
    result = num1 + num2
    return result

@dsl.component()
def subtract(num1: int, num2: int) -> int:
    result = num1 - num2
    return result

@dsl.component()
def multiply(num1: int, num2: int) -> int:
    result = num1 * num2
    return result

@dsl.pipeline(name="num pipeline")
def pipeline(num1: int, num2: int):
    step1 = add(num1=num1, num2=num2)
    step2 = subtract(num1=num1, num2=num2)
    step3 = multiply(num1=step1.output, num2=step2.output)

pipeline_func = pipeline
pipeline_filename = 'num_pipeline.yaml'
kfp.compiler.Compiler().compile(pipeline_func, pipeline_filename)
```

The DAG visualization on the right shows the execution flow of the pipeline. It consists of three nodes: `add`, `subtract`, and `multiply`. The `add` and `subtract` nodes are connected to the `multiply` node, indicating that the output of the `add` step is used as the first input to the `multiply` step, and the output of the `subtract` step is used as the second input to the `multiply` step. All nodes are marked with a green checkmark, indicating successful execution.

02 Kubeflow Components – Kubeflow Pipeline

 Kubeflow

Home

Notebooks

Tensorboards

Volumes

Endpoints

Experiments (AutoML)

Experiments (KFP)

Pipelines

Runs

Recurring Runs

Artifacts

Executions

Manage Contributors

GitHub 

Documentation 

 dpslsl-dev (Owner) 

Pipeline Versions

←

New Pipeline

Upload pipeline or pipeline version.

☒ Create a new pipeline

☐ Create a new pipeline version under an existing pipeline

Select if the new pipeline will be private or shared.

☒ Private

☐ Shared

Upload pipeline with the specified package.

Pipeline Name*

test_num

Pipeline Description

Choose a pipeline package file from your computer, and give the pipeline a unique name.
You can also drag and drop the file here.

For expected file format, refer to [Compile Pipeline Documentation](#).

☒ Upload a file

File*

num pipeline.yaml

Choose file

☐ Import by url

Package Url

Code Source

Create

Cancel



02 Kubeflow Components – Kubeflow Pipeline

 Kubeflow

Home

Notebooks

Tensorboards

Volumes

Endpoints

Experiments (AutoML)

Experiments (KFP)

Pipelines

Runs

Recurring Runs

Artifacts

Executions

Manage Contributors

dpslsl-dev (Owner) 



Pipelines

+ Upload pipeline

Refresh

Delete

PrivateShared

Filter pipelines

☐

Pipeline name

Description

Uploaded on ↓

☐

▶ test_num

2024. 7. 9. 오전 9:36:33

Rows per page: 10   

02 Kubeflow Components – Kubeflow Pipeline

The screenshot displays the Kubeflow Pipeline console interface. On the left is a dark blue sidebar with navigation links: Home, Notebooks, Tensorboards, Volumes, Endpoints, Experiments (AutoML), Experiments (KFP), Pipelines, Runs, Recurring Runs, Artifacts, Executions, Manage Contributors, GitHub, and Documentation. The main content area is titled 'Start a new run' and shows fields for Pipeline (client num pipeline), Pipeline Version (2.0), Run name, and Run of 2.0 (c0). A modal dialog titled 'Choose a pipeline version' is open in the foreground, displaying a table of available pipeline versions.

Version name	Description	Uploaded on ↓
☐ 2.0	the pipeline uploaded by python sdk	2024. 7. 9. 오후 6:09:50
☐ 1.0	the pipeline uploaded by python sdk	2024. 7. 9. 오후 1:46:28

Below the table, there is a 'Rows per page: 10' dropdown and navigation arrows. At the bottom right of the modal are 'Cancel' and 'Use this pipeline version' buttons.

02 Kubeflow Components – Kubeflow Pipeline

Run details

Pipeline*

test_num

Choose

Pipeline Version*

1.0

Choose

Run name*

Run of test_num (228cb)

Description

This run will be associated with the following experiment

Experiment*

test_num

Choose

This run will use the following Kubernetes service account. ?

Service Account

02 Kubeflow Components – Kubeflow Pipeline

Run Type

☒ One-off ☐ Recurring

Pipeline Root

Pipeline Root represents an artifact repository, refer to [Pipeline Root Documentation](#).

☐ Custom Pipeline Root

Run parameters

Specify parameters required by the pipeline

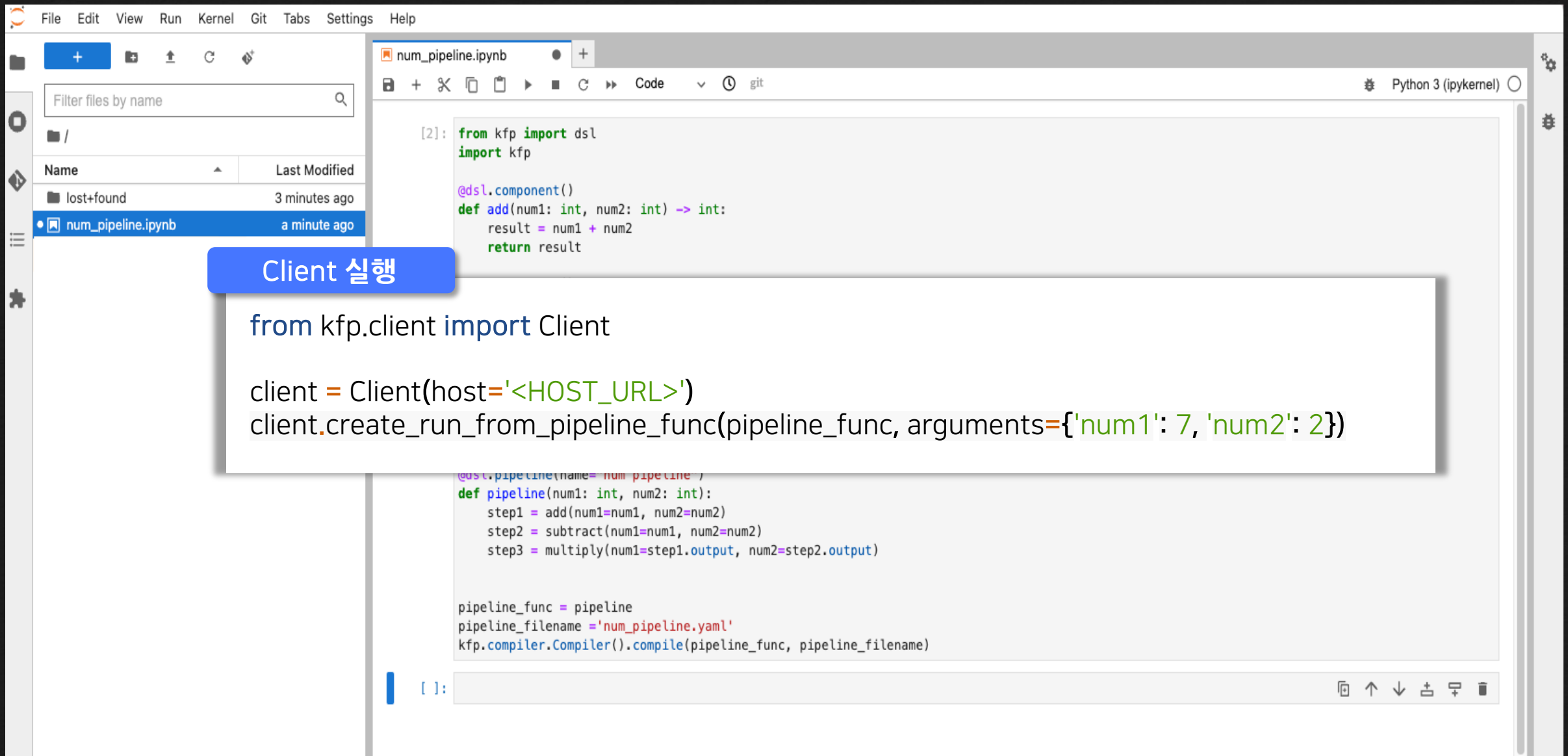
num1 - integer

7

num2 - integer

2

02 Kubeflow Components – Kubeflow Pipeline



Client 실행

```
from kfp.client import Client

client = Client(host='<HOST_URL>')
client.create_run_from_pipeline_func(pipeline_func, arguments={'num1': 7, 'num2': 2})
```

```
[2]: from kfp import dsl
import kfp

@dsl.component()
def add(num1: int, num2: int) -> int:
    result = num1 + num2
    return result

@dsl.pipeline(name='num_pipeline')
def pipeline(num1: int, num2: int):
    step1 = add(num1=num1, num2=num2)
    step2 = subtract(num1=num1, num2=num2)
    step3 = multiply(num1=step1.output, num2=step2.output)

pipeline_func = pipeline
pipeline_filename = 'num_pipeline.yaml'
kfp.compiler.Compiler().compile(pipeline_func, pipeline_filename)
```

02 Kubeflow Components – Kubeflow Pipeline

 Kubeflow

Home

Notebooks

Tensorboards

Volumes

Endpoints

Experiments (AutoML)

Experiments (KFP)

Pipelines

Runs

Recurring Runs

Artifacts

Executions

Manage Contributors

GitHub

dpslsl-dev (Owner)

Experiments > test_num

← ✓ Run of test_num (1d0c8)

Graph

Detail

Pipeline Spec

Layers | root

add

subtract

multiply

multiply

Input/Output

Task Details

Logs

This step corresponds to execution "multiply".

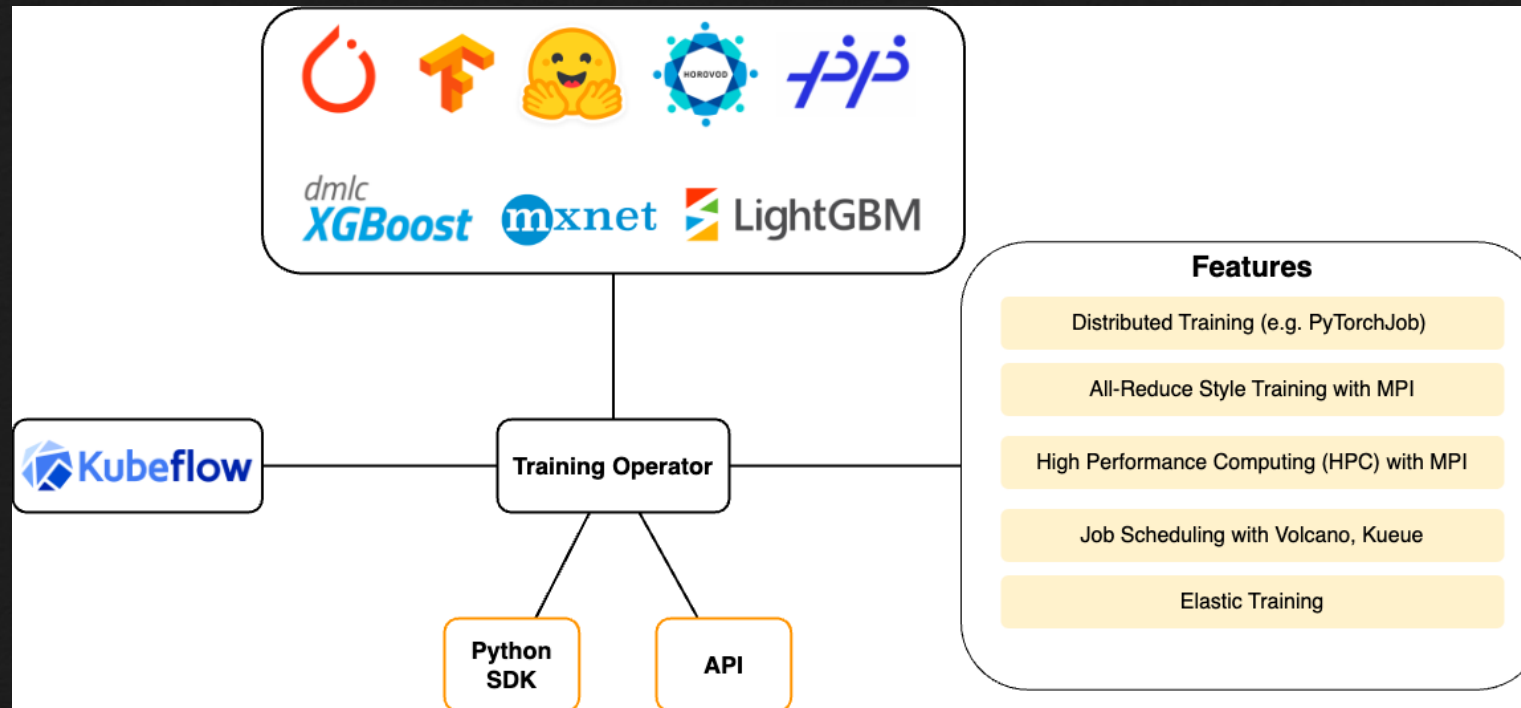
Input Parameters

num1	9
num2	5

Output Parameters

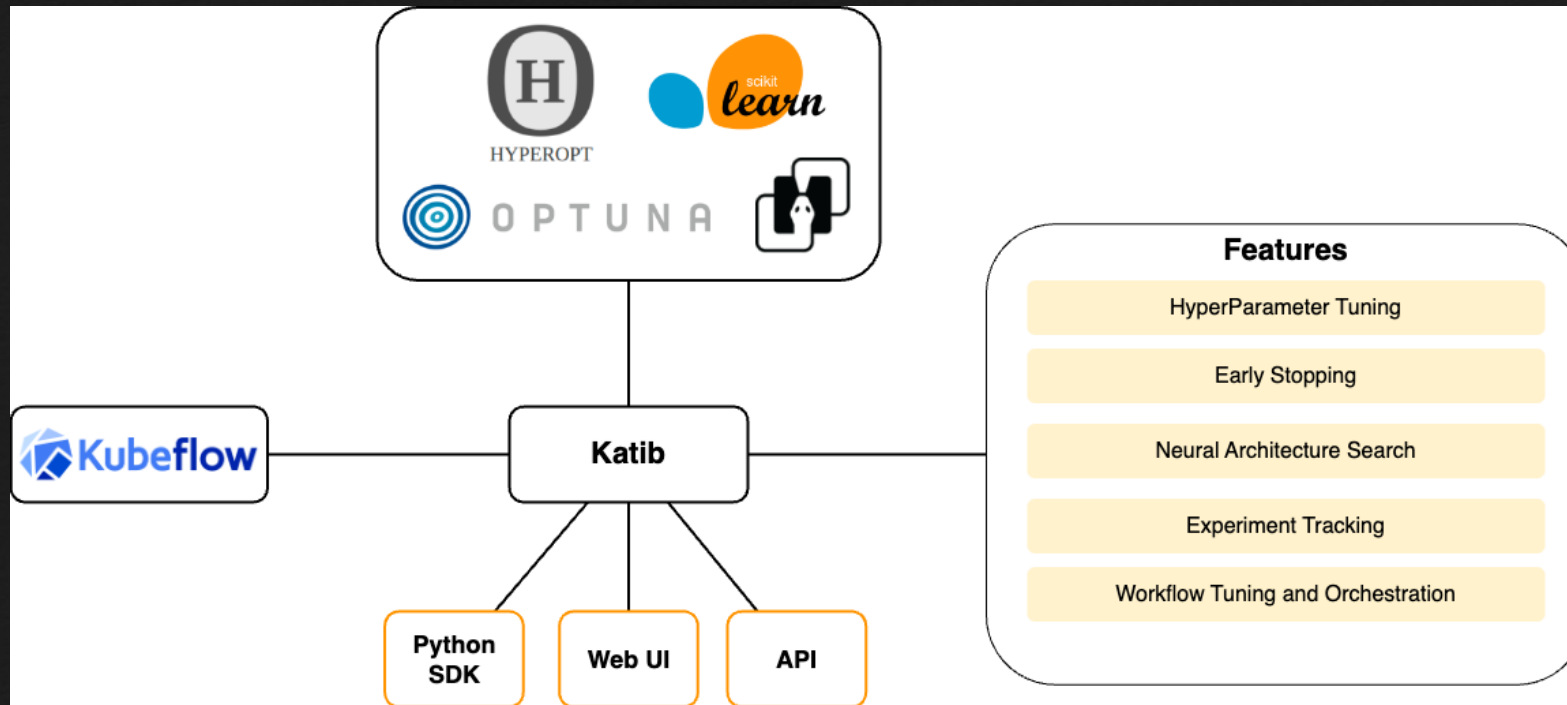
Output	45
--------	----

02 Kubeflow Components – Kubeflow Training Operator



- PyTorch, TensorFlow, XGBoost 등 다양한 ML 프레임워크를 사용하여 만든 ML 모델의 미세 조정 및 분산 학습을 위한 Kubernetes 기반 프로젝트
- HuggingFace, DeepSpeed와 같은 다른 ML 라이브러리를 Training Operator와 통합하여 Kubernetes에서 ML 학습 가능
- API와 Training Operator Python SDK를 이용하여 구성 가능

02 Kubeflow Components – Katib

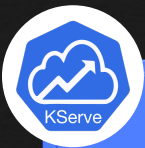
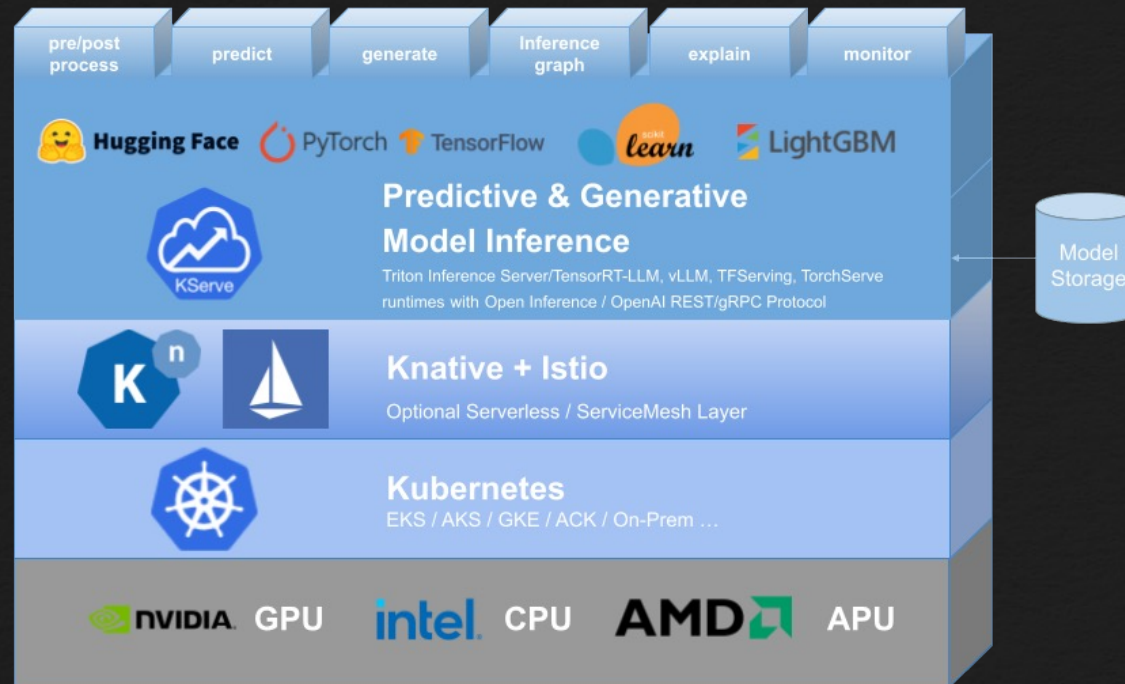


- AutoML을 위한 Kubernetes 기반 프로젝트
- Hyperparameter tuning, Early stopping 및 Neural Architecture Search(NAS)를 지원
- Bayesian optimization, Tree of Parzen Estimators, Random 등의 다양한 AutoML 알고리즘을 지원
- 최적화할 parameter, algorithm, early stopping point, metrix, maxTrialCount, maxFailedTrialCount 등을 통하여 조정 가능

02 Kubeflow Components – Katib



02 Kubeflow Components – KServe



- Scikit-Learn, XGBoost, Tensorflow, PyTorch와 같은 ML 프레임워크와 사용자 정의 모델을 사용하여 CPU, GPU에서 모델 추론이 가능한 서버리스 배포를 제공
- 자동 확장, 네트워킹 상태 검사, Scale to Zero 및 Canary Rollouts 등의 기능을 ML 배포시에 제공
- Inference Service 스토리지 제공 : Google GCS, AWS S3, Azure Blob Storage, 로컬 컨테이너 파일시스템, PVC 형식

02 Kubeflow Components – KServe



POST 요청



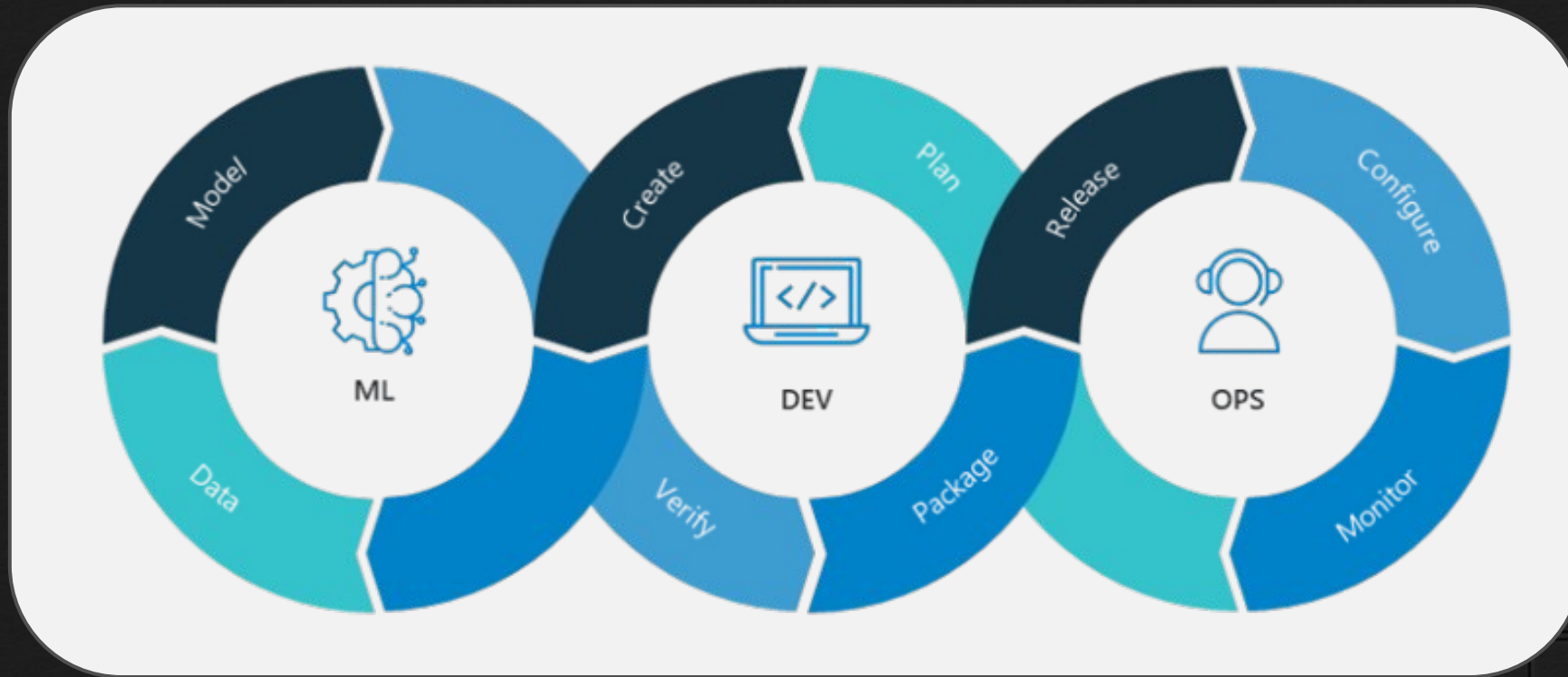
예측값 반환



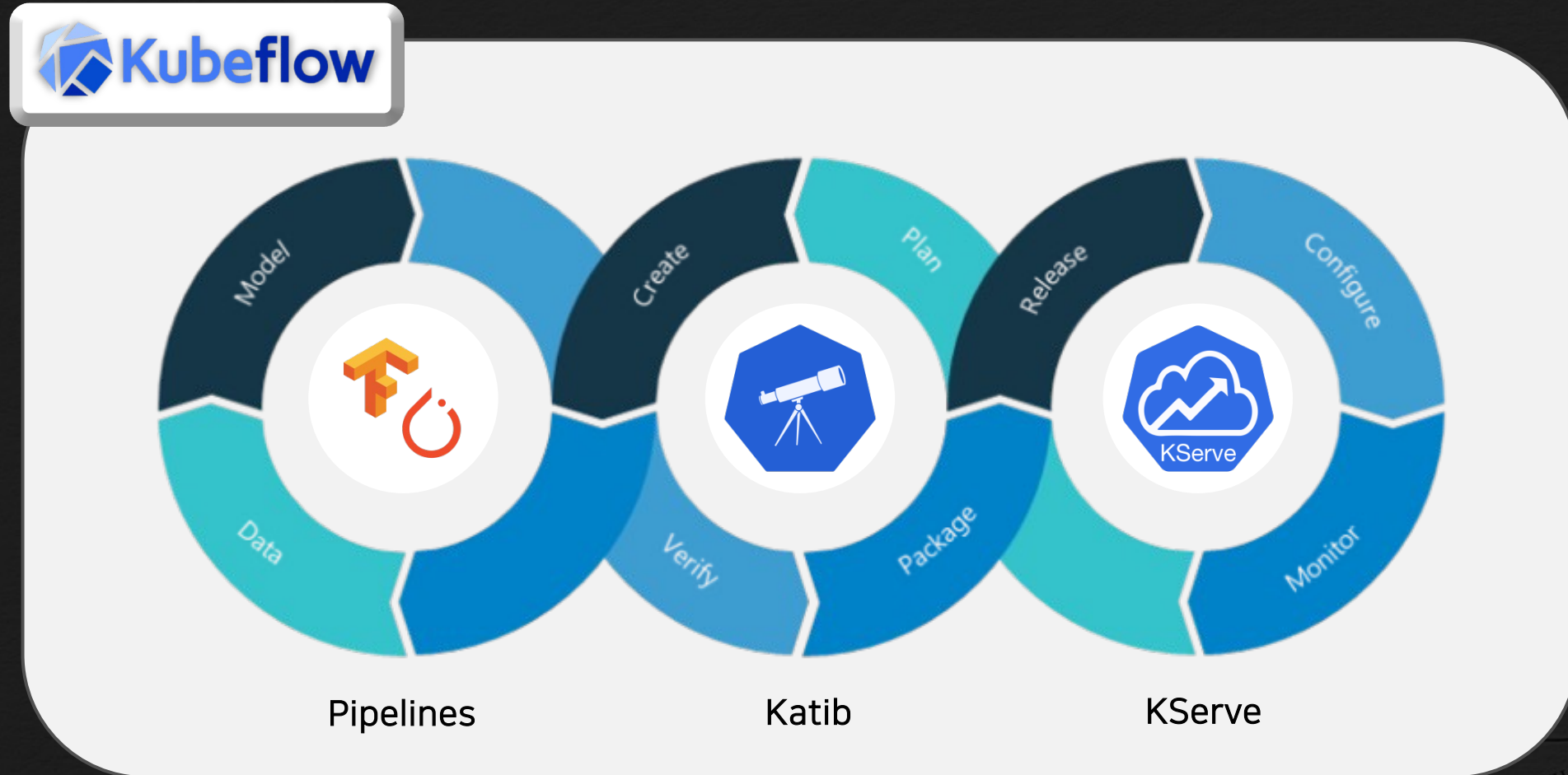
요청	반환값
GET /v1/models	모델 목록
GET /v1/models/<model_name>	모델 상태
POST /v1/models/<model_name>:predict	예측
POST /v1/models/<model_name>:explain	예측 설명

03 MLOps

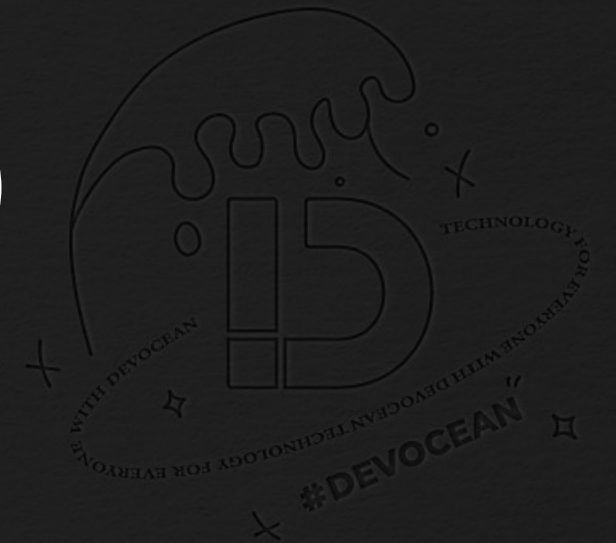
Machine Learning(ML) + Operations(Ops) = MLOps



03 MLOps



03 MLOps





감사합니다.

