머신러닝 파이프라인

쿠베플로우 파이프라인 Part 2

송호연

목차

쿠베플로우 파이프라인 Part 2

학습목표

쿠베플로우 파이프라인 Part 2

01. 재사용 가능한 컴포넌트의 원칙에 대해 이해한다.

어떻게 컴포넌트를 작성하면 재사용성이 높아지는 지 이해한다.

02. 쿠베플로우 실습을 통해 작동 방식을 이해한다.

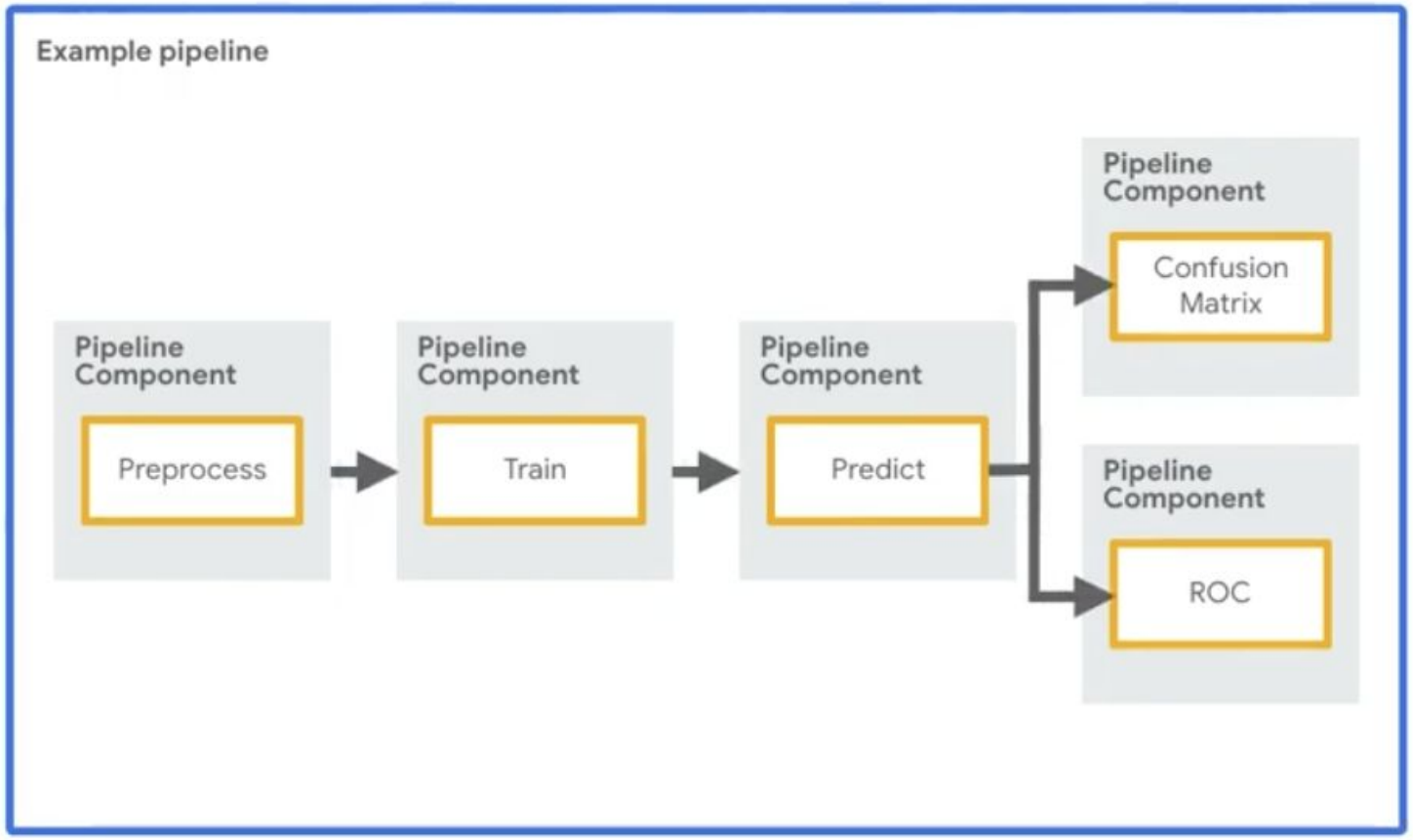
실습을 진행하면서 쿠베플로우 파이프라인의 기본 사용법에 대해 공부한다.

재사용 가능한 컴포넌트 만들기

01

재사용 가능한 컴포넌트 만들기

파이프라인 & 컴포넌트

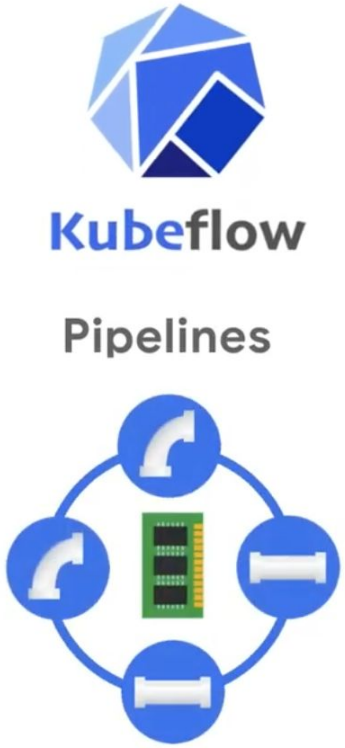


*출처 : Google Cloud Tech Youtue(https://www.youtube.com/watch?v=_AY8mmbR1o4)

재사용 가능한 컴포넌트 만들기

컴포넌트의 데이터 전달

컴포넌트의 개념은 함수의 개념과 매우 유사합니다. 모든 컴포넌트에는 입력 및 출력이있을 수 있습니다 (구성 요소 스펙에 선언되어야 함).

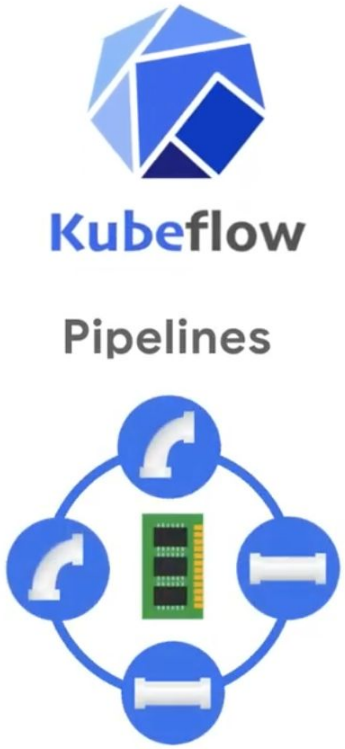


*출처 : Google Cloud Tech Youtue(https://www.youtube.com/watch?v=_AY8mmbR1o4)

재사용 가능한 컴포넌트 만들기

컴포넌트의 데이터 전달

구성 요소를 만들 때 구성 요소가 업스트림 및 다운 스트림 구성 요소와 통신하는 방법을 고려해야합니다. 즉, 입력 데이터를 소비하고 출력 데이터를 생성하는 방법입니다.



*출처 : Google Cloud Tech Youtue(https://www.youtube.com/watch?v=_AY8mmbR1o4)

재사용 가능한 컴포넌트 만들기

컨테이너화 된 프로그램과의 데이터 전달

재사용성 원칙.

프로그램은 **출력 데이터의 경로를 컴포넌트의 파라이터**로 받아 들여야합니다. 즉, 경로를 하드 코딩해서는 안됩니다.

*출처 : Google Cloud Tech Youtue(https://www.youtube.com/watch?v=_AY8mmbR1o4)

재사용 가능한 컴포넌트 만들기

쿠베플로우 파이프라인 코드 예시

```
In [14]: import kfp.dsl as dsl

         def my_pipeline_step(step_name, param1, param2, ...):
             return dsl.ContainerOp(
                 name = step_name,
                 image = '<path to my container image>',
                 arguments = [
                     '--param1', param1,
                     '--param2', param2,
                     ...
                 ],
                 file_outputs = {
                     'output1': '/output1.txt',
                     'output2': '/output2.json',
                     ...
                 }
             )
```

*출처 : Google Cloud Tech Youtue(https://www.youtube.com/watch?v=_AY8mmbR1o4)

재사용 가능한 컴포넌트 만들기

쿠베플로우 파이프라인 코드 예시

```
In [14]: import kfp.dsl as dsl

def my_pipeline_step(step_name, param1, param2, ...):
    return dsl.ContainerOp(
        name = step_name,
        image = '<path to my container image>',
        arguments = [
            '--param1', param1,
            '--param2', param2,
            ...
        ],
        file_outputs = {
            'output1': '/output1.txt',
            'output2': '/output2.json',
            ...
        }
    )
```

input

*출처 : Google Cloud Tech Youtue(https://www.youtube.com/watch?v=_AY8mmbR1o4)

재사용 가능한 컴포넌트 만들기

쿠베플로우 파이프라인 코드 예시

```
In [14]: import kfp.dsl as dsl

def my_pipeline_step(step_name, param1, param2, ...):
    return dsl.ContainerOp(
        name = step_name,
        image = '<path to my container image>',
        arguments = [
            '--param1', param1,
            '--param2', param2,
            ...
        ],
        file_outputs = {
            'output1': '/output1.txt',
            'output2': '/output2.json',
            ...
        }
    )
```

output

*출처 : Google Cloud Tech Youtue(https://www.youtube.com/watch?v=_AY8mmbR1o4)

재사용 가능한 컴포넌트 만들기

컨테이너화 된 프로그램과의 데이터 전달

고급: 외부 시스템에 데이터를 추가해야하는 경우

특정 경우에는 컴포넌트의 목적이 외부 시스템에 데이터를 만들 때가 있습니다. (예를 들어 BigQuery 테이블 생성)

이런 경우에는 쿠베플로우의 관리 밖의 일이 되기에 항상 컴포넌트가 같은 결과를 만들어낼 것이라는 점은 보장할 수 없습니다.

*출처 : Google Cloud Tech Youtue(https://www.youtube.com/watch?v=_AY8mmbR1o4)

재사용 가능한 컴포넌트 만들기

컨테이너화 된 프로그램과의 데이터 전달

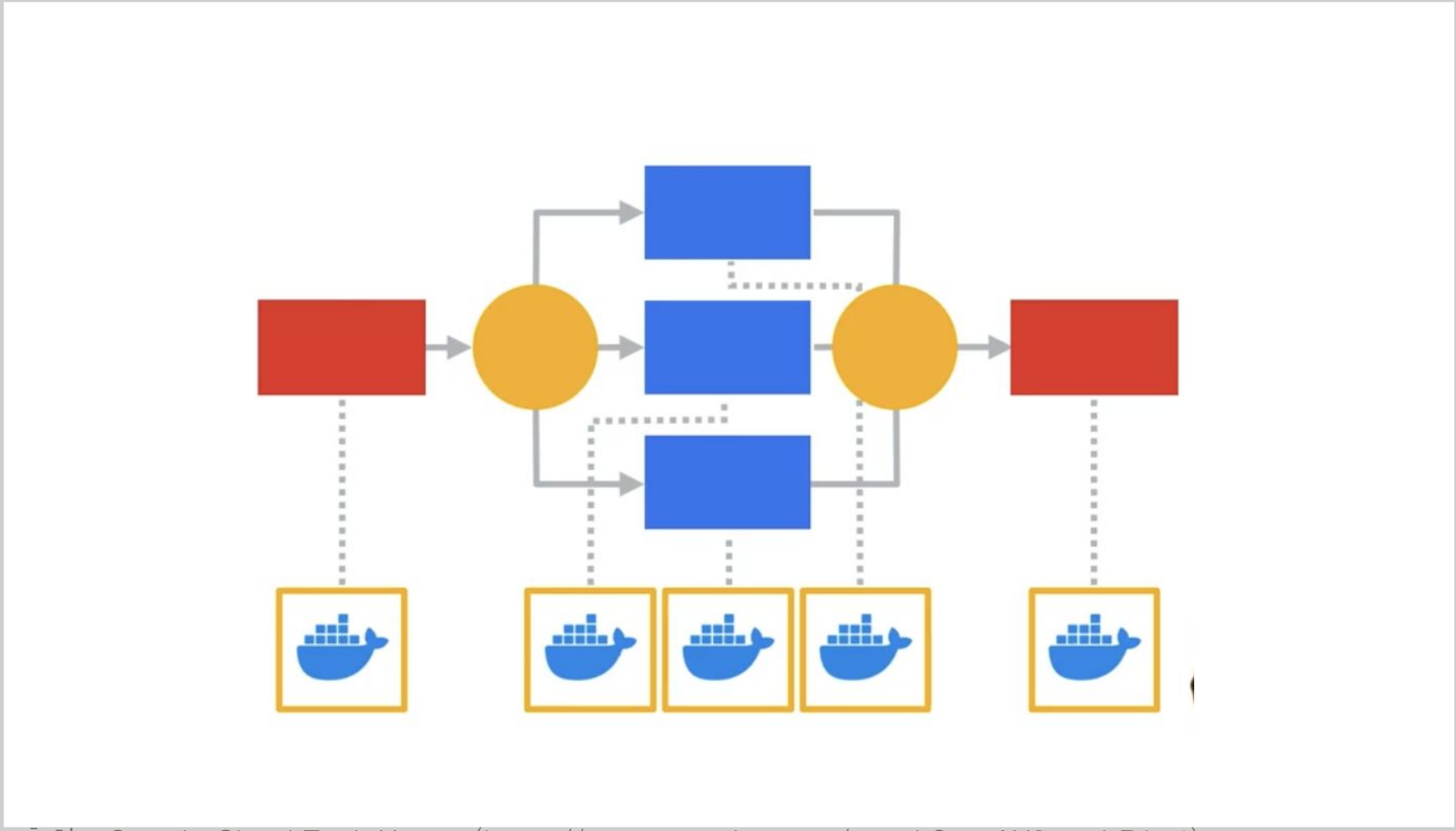
두 가지 방식으로 command-line 프로그램이 데이터를 소비할 수 있습니다.

- 작은 데이터: program.py --param1 100
- 큰 데이터: program.py --data1 /inputs/data1.txt

*출처 : Google Cloud Tech Youtue(https://www.youtube.com/watch?v=_AY8mmbR1o4)

재사용 가능한 컴포넌트 만들기

재사용 가능한 컴포넌트



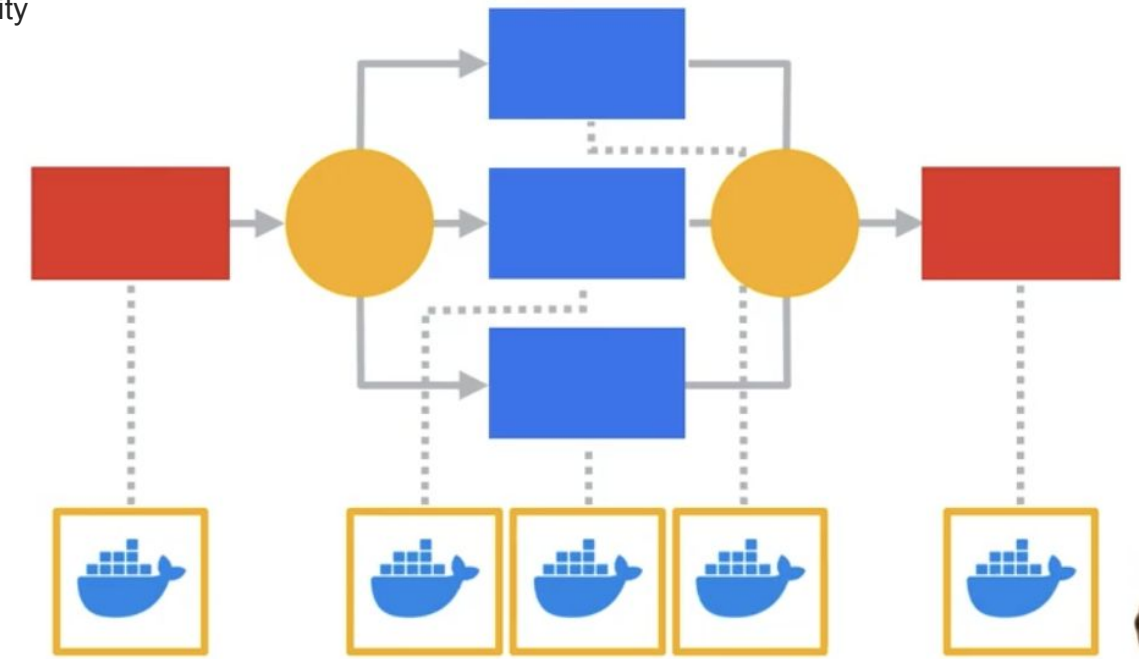
*출처 : Google Cloud Tech Youtue(https://www.youtube.com/watch?v=_AY8mmbR1o4)

재사용 가능한 컴포넌트 만들기

아키텍쳐의 원칙

실행 환경과 코드 런타임을 분리

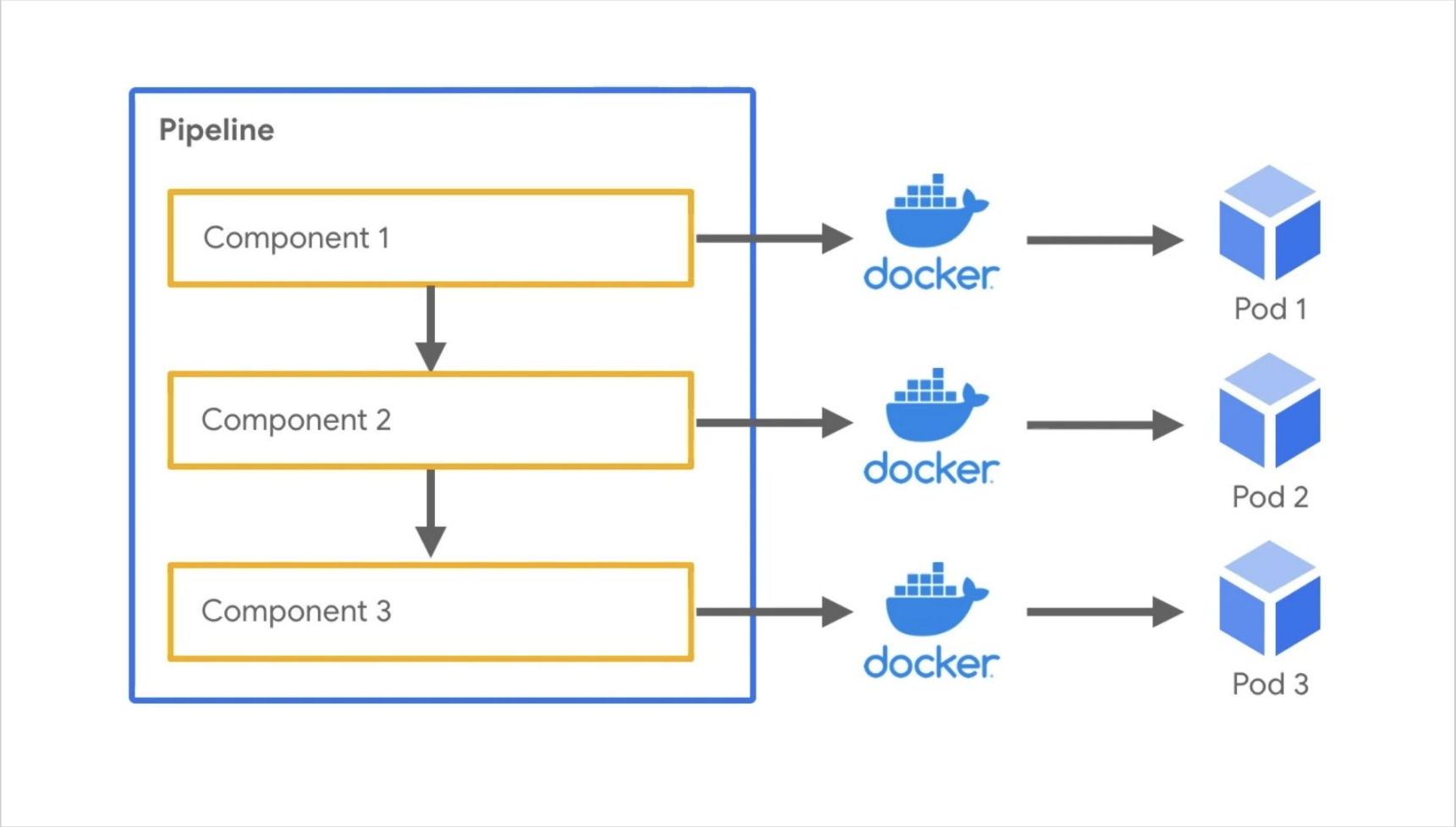
- 이식성 Portability
- 반복가능성 Repeatability
- 캡슐화 Encapsulation



*출처 : Google Cloud Tech Youtue(https://www.youtube.com/watch?v=_AY8mmbR1o4)

재사용 가능한 컴포넌트 만들기

각 컴포넌트는 Docker와 매핑되고, Pod에서 실행됨



*출처 : 출처 작성

실습 1

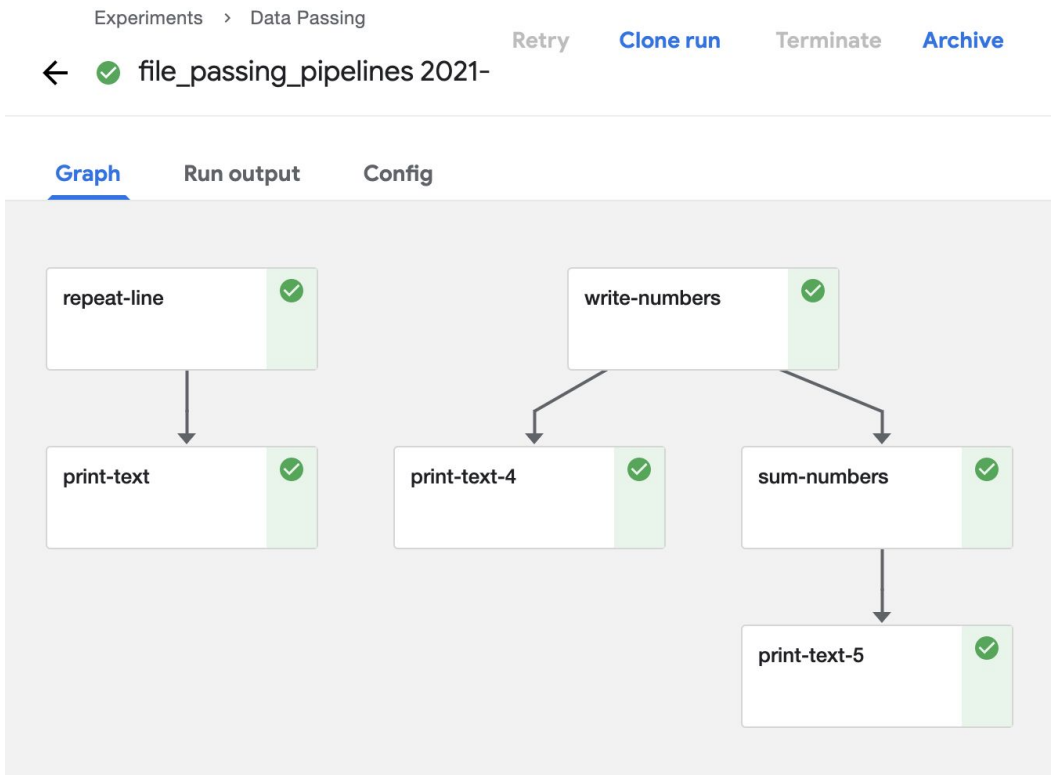
Data Passing

Output a Directory

실습 1

Data Passing

https://github.com/chris-chris/kubeflow-tutorial/tree/master/lesson6_data_passing



실습 1

Data Passing

```
import kfp
from kfp.components import func_to_container_op, InputPath, OutputPath

EXPERIMENT_NAME = 'Data Passing'
KUBEFLOW_HOST = "http://127.0.0.1:8080/pipeline"


@func_to_container_op
def repeat_line(line: str, output_text_path: OutputPath(str), count: int = 10):
    '''Repeat the line specified number of times'''
    with open(output_text_path, 'w') as writer:
        for i in range(count):
            writer.write(line + '\n')


@func_to_container_op
def print_text(text_path: InputPath()): # The "text" input is untyped so that any data can be printed
    '''Print text'''
    with open(text_path, 'r') as reader:
        for line in reader:
            print(line, end = '')
```

실습 1

Data Passing

```
def print_repeating_lines_pipeline():
    repeat_lines_task = repeat_line(line='Hello', count=5000)
    print_text(repeat_lines_task.output) # Don't forget .output !


@func_to_container_op
def split_text_lines(source_path: InputPath(str), odd_lines_path: OutputPath(str), even_lines_path: OutputPath(str)):
    with open(source_path, 'r') as reader:
        with open(odd_lines_path, 'w') as odd_writer:
            with open(even_lines_path, 'w') as even_writer:
                while True:
                    line = reader.readline()
                    if line == "":
                        break
                    odd_writer.write(line)
                    line = reader.readline()
                    if line == "":
                        break
                    even_writer.write(line)
```

실습 1

Data Passing

```
def text_splitting_pipeline():
    text = '\n'.join(['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten'])
    split_text_task = split_text_lines(text)
    print_text(split_text_task.outputs['odd_lines'])
    print_text(split_text_task.outputs['even_lines'])


@func_to_container_op
def write_numbers(numbers_path: OutputPath(str), start: int = 0, count: int = 10):
    with open(numbers_path, 'w') as writer:
        for i in range(start, count):
            writer.write(str(i) + '\n')
```

실습 1

Data Passing

```
# Reading and summing many numbers
@func_to_container_op
def sum_numbers(numbers_path: InputPath(str)) -> int:
    sum = 0
    with open(numbers_path, 'r') as reader:
        for line in reader:
            sum = sum + int(line)
    return sum


# Pipeline to sum 100000 numbers
def sum_pipeline(count: int = 100000):
    numbers_task = write_numbers(count=count)
    print_text(numbers_task.output)

    sum_task = sum_numbers(numbers_task.outputs['numbers'])
    print_text(sum_task.output)
```

실습 1

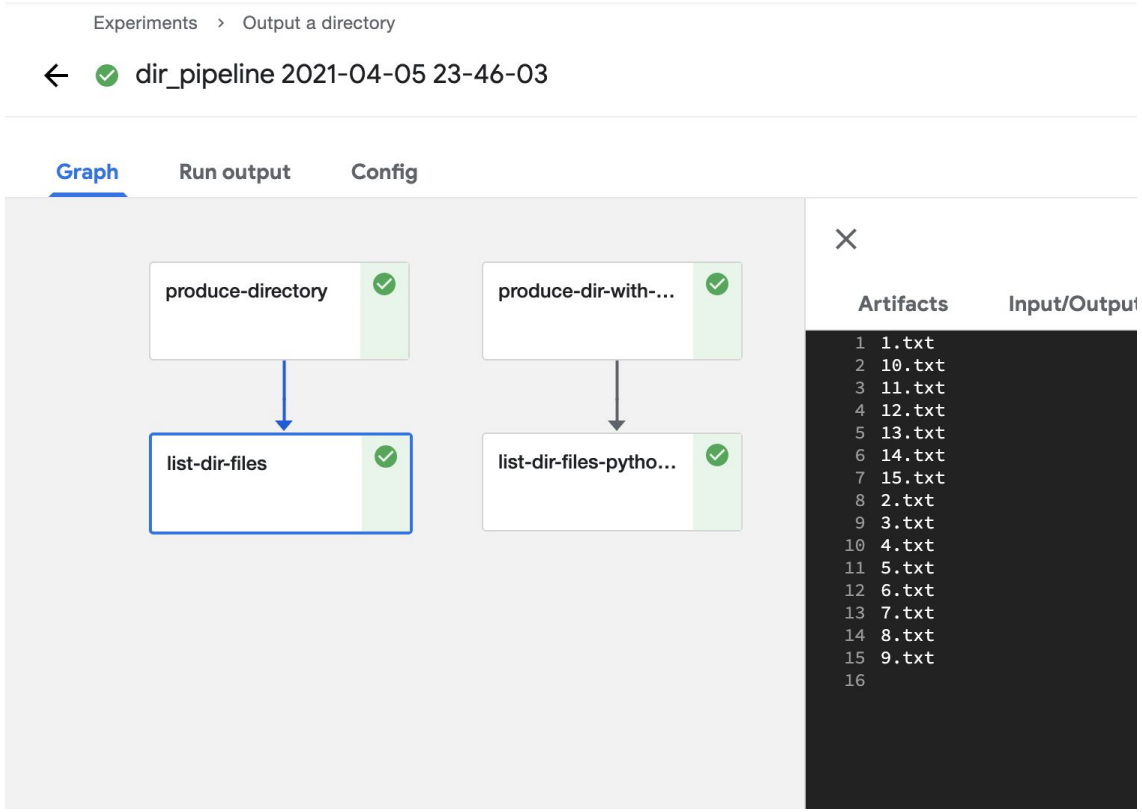
Data Passing

```
def file_passing_pipelines():
    print_repeating_lines_pipeline()
    text_splitting_pipeline()
    sum_pipeline()


if __name__ == '__main__':
    # Compiling the pipeline
    kfp.compiler.Compiler().compile(file_passing_pipelines, __file__ + '.zip')
    kfp.Client(host=KUBEFLOW_HOST).create_run_from_pipeline_func(
        file_passing_pipelines,
        arguments={},
        experiment_name=EXPERIMENT_NAME)
```

실습 1

Output a Directory

https://github.com/chris-chris/kubeflow-tutorial/tree/master/lesson7_output_a_directory



실습 1

Output a Directory

```
import kfp
from kfp.components import create_component_from_func, load_component_from_text, InputPath, OutputPath


EXPERIMENT_NAME = 'Output a directory'        # Name of the experiment in the UI
KUBEFLOW_HOST = "http://127.0.0.1:8080/pipeline"


# Outputting directories from Python-based components:

@create_component_from_func
def produce_dir_with_files_python_op(output_dir_path: OutputPath(), num_files: int = 10):
    import os
    os.makedirs(output_dir_path, exist_ok=True)
    for i in range(num_files):
        file_path = os.path.join(output_dir_path, str(i) + '.txt')
        with open(file_path, 'w') as f:
            f.write(str(i))
```

실습 1

Output a Directory

```
@create_component_from_func
def list_dir_files_python_op(input_dir_path: InputPath()):
    import os
    dir_items = os.listdir(input_dir_path)
    for dir_item in dir_items:
        print(dir_item)
```

실습 1

Output a Directory

```
produce_dir_with_files_general_op = load_component_from_text('''
name: Produce directory
inputs:
- {name: num_files, type: Integer}
outputs:
- {name: output_dir}
implementation:
  container:
    image: alpine
    command:
    - sh
    - -ecx
    - |
      num_files="$0"
      output_path="$1"
      mkdir -p "$output_path"
      for i in $(seq "$num_files"); do
        echo "$i" > "$output_path/${i}.txt"
      done
    - {inputValue: num_files}
    - {outputPath: output_dir}
''')
```

실습 1

Output a Directory

```
list_dir_files_general_op = load_component_from_text('''
name: List dir files
inputs:
- {name: input_dir}
implementation:
  container:
    image: alpine
    command:
    - ls
    - {inputPath: input_dir}
''')
```

실습 1

Output a Directory

```
# Test pipeline

def dir_pipeline():
    produce_dir_python_task = produce_dir_with_files_python_op(num_files=15)
    list_dir_files_python_op(input_dir=produce_dir_python_task.output)

    produce_dir_general_task = produce_dir_with_files_general_op(num_files=15)
    list_dir_files_general_op(input_dir=produce_dir_general_task.output)


if __name__ == '__main__':
    kfp.Client(host=KUBEFLOW_HOST).create_run_from_pipeline_func(
        dir_pipeline,
        arguments={},
        experiment_name=EXPERIMENT_NAME)
```

실습 2

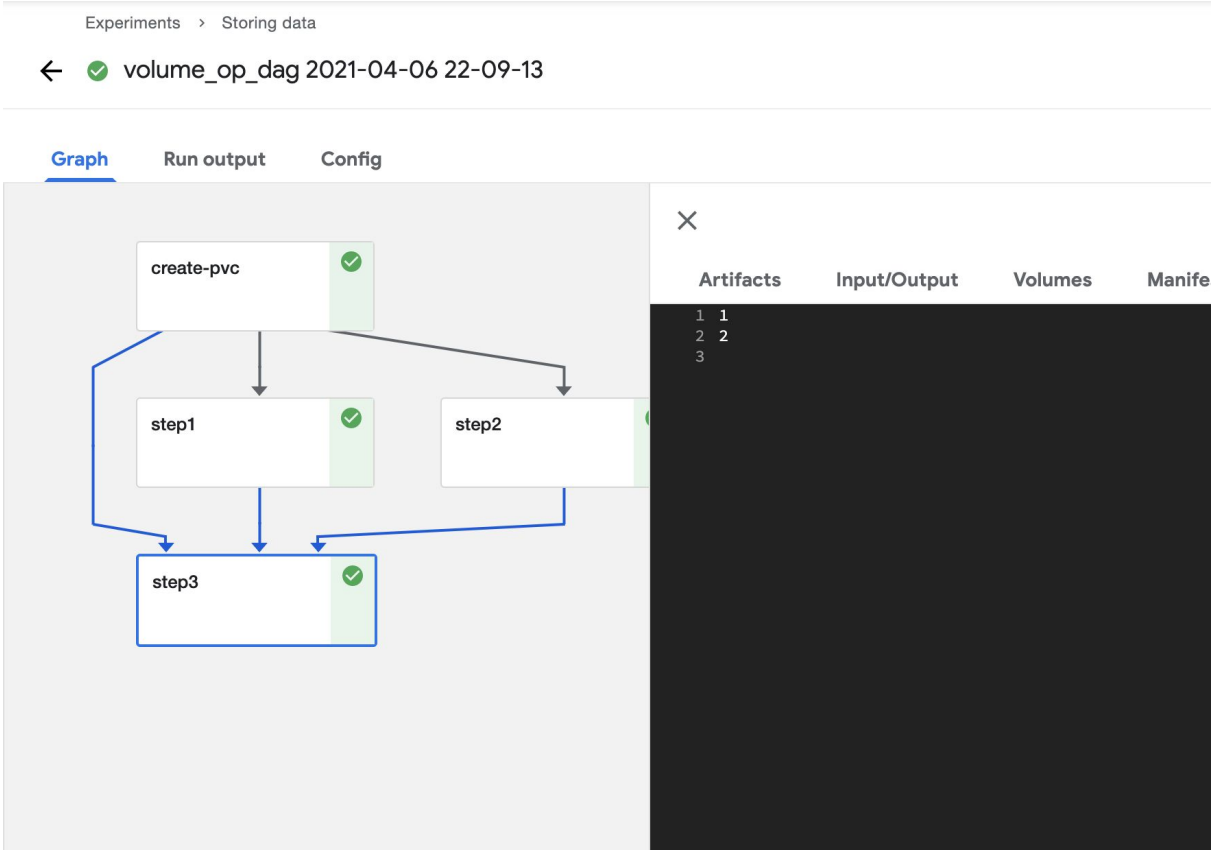
Storing Data

S3 Download

실습 2

Storing Data

https://github.com/chris-chris/kubeflow-tutorial/tree/master/lesson8_storing_data



실습 2

Storing Data

```
import kfp
import kfp.dsl as dsl

EXPERIMENT_NAME = 'Storing data'        # Name of the experiment in the UI
KUBEFLOW_HOST = "http://127.0.0.1:8080/pipeline"


@dsl.pipeline(
    name="Volume Op DAG",
    description="The second example of the design doc."
)
def volume_op_dag():
    vop = dsl.VolumeOp(
        name="create_pvc", resource_name="my-pvc", size="10Gi", modes=dsl.VOLUME_MODE_RWM
    )

    step1 = dsl.ContainerOp(
        name="step1", image="library/bash:4.4.23", command=["sh", "-c"],
        arguments=["echo 1 | tee /mnt/file1"], pvolumes={"/mnt": vop.volume}
    )
```

실습 2

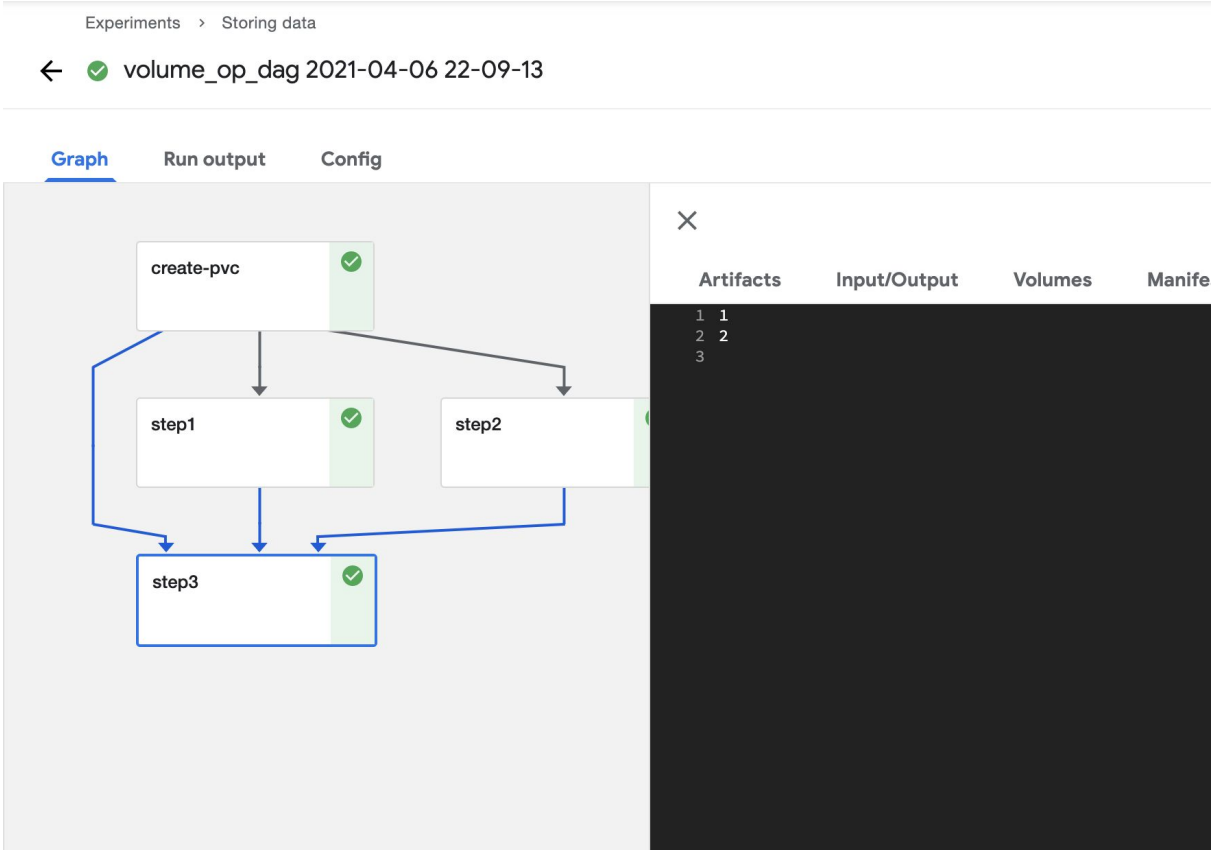
Storing Data

```
    step2 = dsl.ContainerOp(
        name="step2", image="library/bash:4.4.23",
        command=["sh", "-c"], arguments=["echo 2 | tee /mnt2/file2"],
        pvolumes={"/mnt2": vop.volume}
    )

    step3 = dsl.ContainerOp(
        name="step3", image="library/bash:4.4.23",
        command=["sh", "-c"], arguments=["cat /mnt/file1 /mnt/file2"],
        pvolumes={"/mnt": vop.volume.after(step1, step2)}
    )


if __name__ == "__main__":
    import kfp.compiler as compiler
    compiler.Compiler().compile(volume_op_dag, __file__ + ".tar.gz")
    kfp.Client(host=KUBEFLOW_HOST).create_run_from_pipeline_func(
        volume_op_dag,
        arguments={},
        experiment_name=EXPERIMENT_NAME)
```

실습 2

S3 Download

https://github.com/chris-chris/kubeflow-tutorial/tree/master/lesson9_download_s3



실습 2

S3 ls

```
import kfp
from kfp import dsl
from kfp.aws import use_aws_secret


EXPERIMENT_NAME = 'AWS S3 ls'        # Name of the experiment in the UI
KUBEFLOW_HOST = "http://127.0.0.1:8080/pipeline"


def s3_ls():
    return kfp.dsl.ContainerOp(
        name="s3_ls",
        image="amazon/aws-cli:latest",
        command=["aws", "s3", "ls"],
    )
```

실습 2

S3 ls

```
@dsl.pipeline(name="s3_ls_pipeline", description="s3 ls pipeline.")
def s3_ls_pipeline():
    echo_task = s3_ls().apply(use_aws_secret('aws-secret', 'AWS_ACCESS_KEY_ID', 'AWS_SECRET_ACCESS_KEY'))


if __name__ == "__main__":
    kfp.compiler.Compiler().compile(s3_ls_pipeline, __file__ + ".zip")
    kfp.Client(host=KUBEFLOW_HOST).create_run_from_pipeline_func(
        s3_ls_pipeline,
        arguments={},
        experiment_name=EXPERIMENT_NAME,
    )
```

실습 2

S3 sync

```
import kfp
from kfp import dsl
from kfp.aws import use_aws_secret


EXPERIMENT_NAME = "AWS S3 sync"
KUBEFLOW_HOST = "http://127.0.0.1:8080/pipeline"
S3_PATH = "s3://chris-private/s3sync"


def s3_sync():
    return kfp.dsl.ContainerOp(
        name="s3_sync",
        image="amazon/aws-cli:latest",
        command=["aws", "s3", "sync", S3_PATH, "/tmp"],
        file_outputs={
            "data": "/tmp"
        }
    )
```

실습 2

S3 sync

```
@dsl.pipeline(name="s3_sync_pipeline", description="s3 sync")
def s3_sync_pipeline():
    _ = s3_sync().apply(use_aws_secret('aws-secret', 'AWS_ACCESS_KEY_ID', 'AWS_SECRET_ACCESS_KEY'))


if __name__ == "__main__":
    kfp.Client(host=KUBEFLOW_HOST).create_run_from_pipeline_func(
        s3_sync_pipeline,
        arguments={},
        experiment_name=EXPERIMENT_NAME
    )
```

실습 2

kubectl apply -f secret.yml

```
apiVersion: v1
kind: Secret
metadata:
  name: aws-secret
  namespace: kubeflow
type: Opaque
data:
  AWS_ACCESS_KEY_ID: BASE64_AWS_ACCESS_KEY_ID
  AWS_SECRET_ACCESS_KEY: BASE64_AWS_SECRET_ACCESS_KEY

  # echo -n "AWS_ACCESS_KEY_ID" | base64
```

짚어보기

쿠베플로우 파이프라인 Part 2

01. 재사용 가능한 컴포넌트의 원칙에 대해 이해한다.

어떻게 컴포넌트를 작성하면 재사용성이 높아지는 지 이해한다.

02. 쿠베플로우 실습을 통해 작동 방식을 이해한다.

실습을 진행하면서 쿠베플로우 파이프라인의 기본 사용법에 대해 공부한다.

머신러닝 파이프라인

쿠베플로우 파이프라인 Part 2

송호연

감사합니다.

THANKS FOR WATCHING