Flyte *Cheat Sheet*

A data, machine learning and infrastructure orchestrator

Homepage | Documentation | Github | Slack | Twitter

Python SDK Installation

```
pip install flytekit
```

Task

```
from flytekit import task

@task
def hello_world() -> str:
    return "Hello World!"
```

Workflow

```
from flytekit import workflow

@workflow
def wf() -> str:
   return hello_world()
```

Bash Script

```
from flytekit import kwtypes, workflow
from flytekit.extras.tasks.shell import ShellTask
from flytekit.types.file import FlyteFile

t1 = ShellTask(
 name="task_1",
    debug=True,
    script="""
    set -ex
    echo "Hey there! Let's run some bash scripts!"
    echo "Showcasing Flyte's Shell Task." >> {inputs.x}
    """,
    inputs=kwtypes(x=FlyteFile),
)

@workflow
def wf(
    file_name: FlyteFile = "https://raw.githubusercontent.com/mwaskom/
seaborn-data/master/iris.csv",
):
    t1(x=file_name)
```

Scheduling a Workflow

```
from datetime import datetime

from flytekit import CronSchedule, LaunchPlan, task, workflow

@task
def format_date(run_date: datetime) -> str:
 return run_date.strftime("%Y-%m-%d %H:%M")

@workflow
def date_formatter_wf(kickoff_time: datetime) -> str:
    return format_date(run_date=kickoff_time)

cron_lp = LaunchPlan.get_or_create(
    name="my_cron_scheduled_lp",
    workflow=date_formatter_wf,
    schedule=CronSchedule(
        schedule="*/1 * * * *",  # every minute
        kickoff_time_input_arg="kickoff_time",
    ),
)

Activate a schedule via
$ flytectl update launchplan -p flytesnacks -d development
"my_cron_scheduled_lp" --version <version> --activate
```

Caching

```
import numpy as np
from flytekit import task
from sklearn import datasets

@task(cache=True, cache_version="1.0")
def get_data() -> tuple[np.ndarray, np.ndarray]:
    iris = datasets.load_iris()
    X, y = (iris.data[:, :2], iris.target)
    return (X, y)
```

Customizing Resources

```
import numpy as np
import pandas as pd
from flytekit import Resources, task, workflow

@task
def generate_df(n: int, mean: float, sigma: float) -> pd.DataFrame:
 return pd.DataFrame({"numbers": np.random.normal(mean, sigma, size=n)})

@task(limits=Resources(cpu="1", mem="500Mi"))
def compute_stats(df: pd.DataFrame) -> tuple[float, float]:
    return float(df["numbers"].mean()), float(df["numbers"].std())

@workflow
def wf(n: int = 200, mean: float = 0.0, sigma: float = 1.0) ->
tuple[float, float]:
    return compute_stats(df=generate_df(n=n, mean=mean, sigma=sigma))
```

Dynamic Workflow

```
from flytekit import dynamic, task, workflow

Point = dict[str, float]

@task
def black_box_function(point: Point) -> float:
    x, y = point["x"], point["y"]
    return -(x**2) - (y - 1) ** 2 + 1

@dynamic
def concurrent_trials(points: list[Point]) -> list[float]:
    return [black_box_function(point=point) for point in points]

@workflow
def wf(
    points: list[Point] = [{"x": 1.2, "y": 0.9}, {"x": 2.3, "y": 4.5}]
) -> list[float]:
    return concurrent_trials(points=points)
```

FlyteRemote

```
from basic_workflow import my_wf
from flytekit.configuration import Config, PlatformConfig
from flytekit.remote import FlyteRemote

remote = FlyteRemote(
    config=Config(
        platform=PlatformConfig(
            endpoint="...", client_id="...",
client_credentials_secret="..."
        )
    ),
    default_project="flytesnacks",
    default_domain="development",
)

registered_workflow = remote.register_script(my_wf)

execution = remote.execute(registered_workflow, inputs={"a": 19, "b":
"Hello"})
print(f"Execution successfully started: {execution.id.name}")
```

Imperative Workflow

```
from flytekit import task, Workflow

@task
def t1(a: str) -> str:
 return a + " world"

wf = Workflow(name="core.my_workflow")
wf.add_workflow_input("in1", str)
node_t1 = wf.add_entity(t1, a=wf.inputs["in1"])
wf.add_workflow_output("output_from_t1", node_t1.outputs["o0"])

if __name__ == "__main__":
    print(wf(in1="hello"))
```

Offloading & Downloading a File & Folder

```
import os
from typing import NamedTuple

import flytekit
from flytekit import task, workflow
from flytekit.types.directory import FlyteDirectory
from flytekit.types.file import JPEGImageFile
from PIL import Image

ImageOutput = NamedTuple("ImageOutput", image=JPEGImageFile,
image_dir=FlyteDirectory)

@task
def rgb_to_bw(
 image: JPEGImageFile,
) -> ImageOutput:
    image_file = Image.open(image.download())
    image_file = image_file.convert("1")  # convert image to black and
white
    working_dir = flytekit.current_context().working_directory
    local_dir = os.path.join(working_dir, "jpeg_images")
    os.makedirs(local_dir, exist_ok=True)
    out_path = os.path.join(local_dir, "bw_" +
os.path.basename(image.path))
    image_file.save(out_path)
    return ImageOutput(image=out_path, image_dir=local_dir)

@workflow
def wf() -> ImageOutput:
    return rgb_to_bw(
        image="https://media.sketchfab.com/models/
e13940161fb64746a4f6753f76abe886/thumbnails/
b7e1ba951ffb46a4ad584ba8ae400d17/e9de09ac9c7941f1924bd384e74a5e2e.jpeg"
    )
```

Branching

```
import random

from flytekit import conditional, task, workflow

@task
def coin_toss(seed: int) -> bool:
    r = random.Random(seed)
    if r.random() < 0.5:
        return True
    return False

@task
def failed() -> int:
    return -1

@task
def success() -> int:
    return 0

@workflow
def basic_boolean_wf(seed: int = 5) -> int:
    result = coin_toss(seed=seed)
    return (
conditional("test").if_(result.is_true()).then(success()).else_().then(f
ailed())
    )
```

Reference Task

```
from flytekit import reference_task
from flytekit.types.file import FlyteFile

@reference_task(
    project="flytesnacks",
    domain="development",
    name="core.flyte_basics.files.normalize_columns",
    version="v1",
)
def normalize_columns(
    csv_url: FlyteFile,
    column_names: list[str],
    columns_to_normalize: list[str],
    output_location: str,
) -> FlyteFile:
    ...
```

Flyte Deck

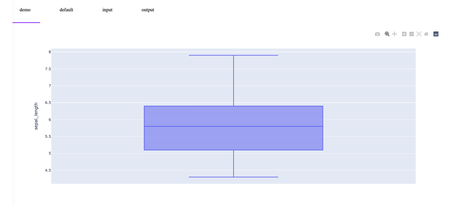
```
import plotly.express as px
from flytekit import Deck, task, workflow
from flytekitplugins.deck import BoxRenderer

iris_df = px.data.iris()

@task(container_image="ghcr.io/flyteorg/flytecookbook:core-latest")
def t1():
    Deck("demo", BoxRenderer("sepal_length").to_html(iris_df))

@workflow
def wf():
    t1()
```

Output:



Map Task

```
from collections import Counter, defaultdict

from flytekit import map_task, task, workflow

@task
def mappable_task(string_item: str) -> dict[str, int]:
 return dict(Counter(string_item.split()))

@task
def coalesce(string_list: list[dict[str, int]]) -> dict[str, int]:
    temp_dict = defaultdict(int)
    for each_item in string_list:
        for k, v in each_item.items():
            temp_dict[k] += v
    return dict(temp_dict)

@workflow
def wf(
    string_list: list[str] = ["Deer Bear", "Car Bear", "Car Deer"]
) -> dict[str, int]:
    mapped_out = map_task(mappable_task)
(string_item=string_list).with_overrides(
        retries=1
    )
    return coalesce(string_list=mapped_out)
```

Multiple Images in a Single Workflow

```
import numpy as np
from flytekit import task, workflow
from sklearn import datasets

@task(container_image="ghcr.io/flyteorg/flytecookbook:core-latest")
def get_data() -> np.ndarray:
    iris = datasets.load_iris()
    X, y = (iris.data[:, :2], iris.target)
    return X

@task(container_image="ghcr.io/flyteorg/flytecookbook:core-latest")
def normalize(X: np.ndarray) -> np.ndarray:
    return (X - X.mean(axis=0)) / X.std(axis=0)

@workflow
def wf() -> np.ndarray:
    X = get_data()
    X = normalize(X=X)
    return X
```

Structured Dataset

```
from typing import Annotated

import pandas as pd
from flytekit import kwtypes, task, workflow
from flytekit.types.structured.structured_dataset import
StructuredDataset

subset_cols = kwtypes(Age=int)

@task
def get_df(a: int) -> pd.DataFrame:
 return pd.DataFrame(
        {"Name": ["Tom", "Joseph"], "Age": [a, 22], "Height": [160,
178]}
    )

@task
def get_subset_df(
    df: Annotated[StructuredDataset, subset_cols]
) -> Annotated[StructuredDataset, subset_cols]:
    df = df.open(pd.DataFrame).all()
    df = pd.concat([df, pd.DataFrame([[30]], columns=["Age"])])
    return StructuredDataset(dataframe=df)

@workflow
def wf() -> Annotated[StructuredDataset, subset_cols]:
    return get_subset_df(df=get_df(a=5))
```

Chaining Flyte Entities

```
from flytekit import task, workflow

@task
def t1():
 print("Trigger me first")

@task
def t2():
    print("Trigger me second")

@workflow
def wf():
    t1() >> t2()
```

ⓘ Tested the examples against flytekit 1.1.1 & python 3.9

Learn more about Flyte at flyte.org