Lab: Prefect - Flows

Flows are like functions. They can take inputs, perform work, and return an output. In fact, you can turn any function into a Prefect flow by adding the `@flow` decorator. When a function becomes a flow, its behavior changes, giving it the following advantages:

- All runs of the flow have persistent `state`. Transitions between states are recorded, allowing for flow execution to be observed and acted upon.
- Input arguments can be type validated as workflow parameters.
- Retries can be performed on failure.
- Timeouts can be enforced to prevent unintentional, long-running workflows.
- Metadata about `flow runs`, such as run time and final state, is automatically tracked.
- They can easily be elevated to a `deployment`, which exposes a remote API for interacting with it

Run flow

The simplest way to get started with Prefect is to annotate a Python function with the `@flow` decorator. The script below fetches statistics about the `main Prefect repository`. Note that httpx is an HTTP client library and a dependency of Prefect. Let's turn this function into a Prefect flow and run the script:

Create a python file `repo_info.py`:

```
import httpx
from prefect import flow


@flow
def get_repo_info():
    url = "https://api.github.com/repos/PrefectHQ/prefect"
    response = httpx.get(url)
    response.raise_for_status()
    repo = response.json()
    print("PrefectHQ/prefect repository statistics 🤓:")
    print(f"Stars 🌠 : {repo['stargazers_count']}")
    print(f"Forks 🍴 : {repo['forks_count']}")

if __name__ == "__main__":
    get_repo_info()
```

Running this file will result in some interesting output:

```
12:47:42.792 | INFO | prefect.engine - Created flow run 'ludicrous-warthog' for flow
'get-repo-info'
PrefectHQ/prefect repository statistics 🤓:
Stars 🌠 : 12146
Forks 🍴 : 1245
12:47:45.008 | INFO | Flow run 'ludicrous-warthog' - Finished in state Completed()
```

Flows can contain arbitrary Python

As we can see above, flow definitions can contain arbitrary Python logic.

Parameters

As with any Python function, you can pass arguments to a flow. The positional and keyword arguments defined on your flow function are called parameters. Prefect will automatically perform type conversion using any provided type hints. Let's make the repository a string parameter with a default value:

Update following code in python file `repo_info.py`:

```
import httpx
from prefect import flow


@flow
def get_repo_info(repo_name: str = "PrefectHQ/prefect"):
    url = f"https://api.github.com/repos/{repo_name}"
    response = httpx.get(url)
    response.raise_for_status()
    repo = response.json()
    print(f"{repo_name} repository statistics 🤓:")
    print(f"Stars 🌠 : {repo['stargazers_count']}")
    print(f"Forks 🍴 : {repo['forks_count']}")


if __name__ == "__main__":
    get_repo_info(repo_name="PrefectHQ/marvin")
```

We can call our flow with varying values for the repo_name parameter (including "bad" values):

`python repo_info.py`

Try passing repo_name="missing-org/missing-repo".

You should see

```
HTTPStatusError: Client error '404 Not Found' for url
'<https://api.github.com/repos/missing-org/missing-repo>'
```

Now navigate to your Prefect dashboard and compare the displays for these two runs.

Logging

Prefect enables you to log a variety of useful information about your flow and task runs, capturing information about your workflows for purposes such as monitoring, troubleshooting, and auditing. If we navigate to our dashboard and explore the runs we created above, we will notice that the repository statistics are not captured in the flow run logs.

Let's fix that by adding some logging to our flow:

Update following code in python file `repo_info.py`:

```
import httpx
from prefect import flow, get_run_logger


@flow
def get_repo_info(repo_name: str = "PrefectHQ/prefect"):
    url = f"https://api.github.com/repos/{repo_name}"
    response = httpx.get(url)
```

```
    response.raise_for_status()
    repo = response.json()
    logger = get_run_logger()
    logger.info("%s repository statistics 🤓:", repo_name)
    logger.info(f"Stars 🌠 : %d", repo["stargazers_count"])
    logger.info(f"Forks 🍴 : %d", repo["forks_count"])
```

Now the output looks more consistent and, more importantly, our statistics are stored in the Prefect backend and displayed in the UI for this flow run:

```
12:47:42.792 | INFO    | prefect.engine - Created flow run 'ludicrous-warthog' for
flow 'get-repo-info'
12:47:43.016 | INFO    | Flow run 'ludicrous-warthog' - PrefectHQ/prefect repository
statistics 🤓:
12:47:43.016 | INFO    | Flow run 'ludicrous-warthog' - Stars 🌠 : 12146
12:47:43.042 | INFO    | Flow run 'ludicrous-warthog' - Forks 🍴 : 1245
12:47:45.008 | INFO    | Flow run 'ludicrous-warthog' - Finished in state Completed()
log_prints=True
```

We could have achieved the exact same outcome by using Prefect's convenient log_prints keyword argument in the flow decorator:

```
@flow(log_prints=True)
def get_repo_info(repo_name: str = "PrefectHQ/prefect"):
    ...
```

Retries

So far our script works, but in the future unexpected errors may occur. For example the GitHub API may be temporarily unavailable or rate limited. Retries help make our flow more resilient. Let's add retry functionality to our example above:

Update following code in python file `repo_info.py`:

```
import httpx
from prefect import flow


@flow(retries=3, retry_delay_seconds=5, log_prints=True)
def get_repo_info(repo_name: str = "PrefectHQ/prefect"):
    url = f"https://api.github.com/repos/{repo_name}"
    response = httpx.get(url)
    response.raise_for_status()
    repo = response.json()
    print(f"{repo_name} repository statistics 🤓:")
    print(f"Stars 🌠 : {repo['stargazers_count']}")
    print(f"Forks 🍴 : {repo['forks_count']}")

if __name__ == "__main__":
    get_repo_info()
```

Next: Tasks

As you have seen, adding a flow decorator converts our Python function to a resilient and observable workflow. In the next section, you'll supercharge this flow by using tasks to break down the workflow's complexity and make it more performant and observable