PACC

Prefect Associate Certification Course

PREFECT

Slack

✅ **Join Prefect Community Slack**

✅ **Join the *pacc-* channel for the course**

Norms

Norms

Zoom

- Camera on
- Mute unless asking a question
- Use hand raise to ask a question

Slack

- Use threads
- Emoji responses 🙂

Norms

Code of conduct

- We expect all participants to be kind and respectful
- Reach out to any of the instructors via Slack if you see or experience an issue

Introductions

Goals

Goals

1. Competence with Prefect so you can build workflows you can trust
2. Connect with each other
3. Have fun! 🎉

Overview

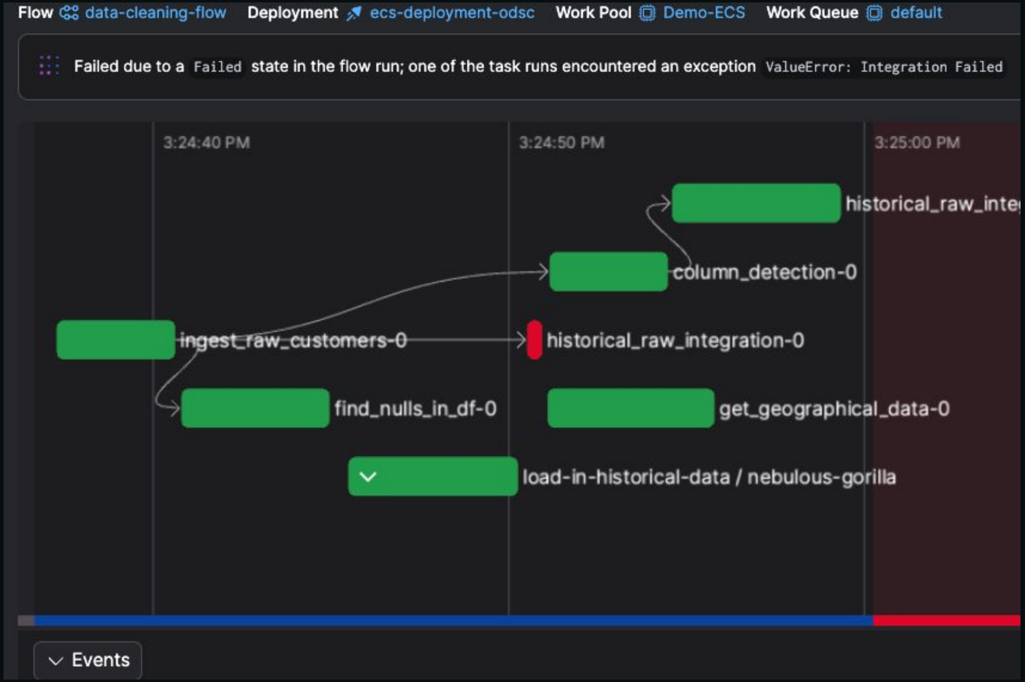
Why workflow management?

Answers the questions:

- What?
- When?
- Where?
- How?
- Who?

What?

Code that moves and transforms data

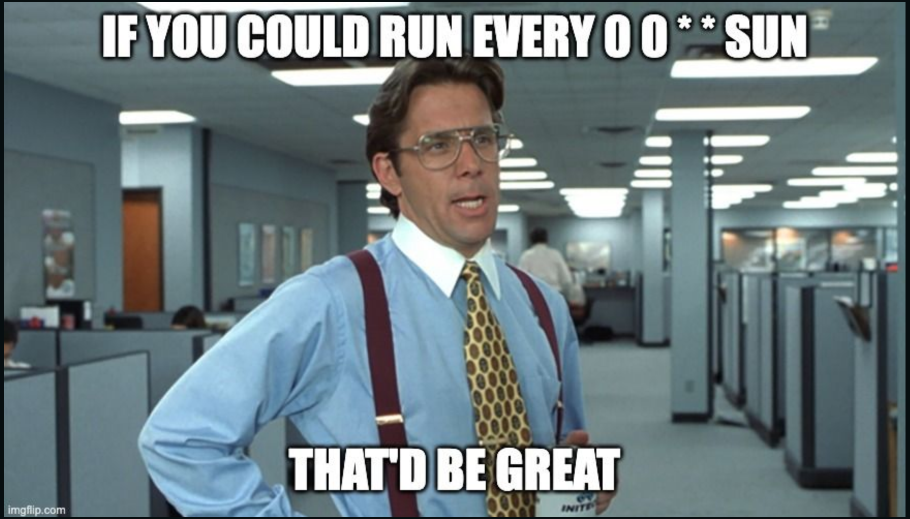


Flow data-cleaning-flow Deployment ecs-deployment-odsc Work Pool Demo-ECS Work Queue default

Failed due to a `Failed` state in the flow run; one of the task runs encountered an exception `ValueError: Integration Failed`

When?

- Ad hoc (manually)
- On a schedule
- In response to events



Where?

- Locally
- In the cloud

Azure

aws

Google Cloud

How?

- Docker, K8s, subprocess
- Trigger from the UI, CLI, code
- Pause for 🧍

Who?

- Auth - SSO/SCIM
- RBAC
- Auditable

What is Prefect?

Prefect is an orchestration and observability platform that empowers developers to build resilient workflows you can trust. 💙

Prefect data workflow orchestration

Automation

Flexible Orchestration in Pure Python

Resilience

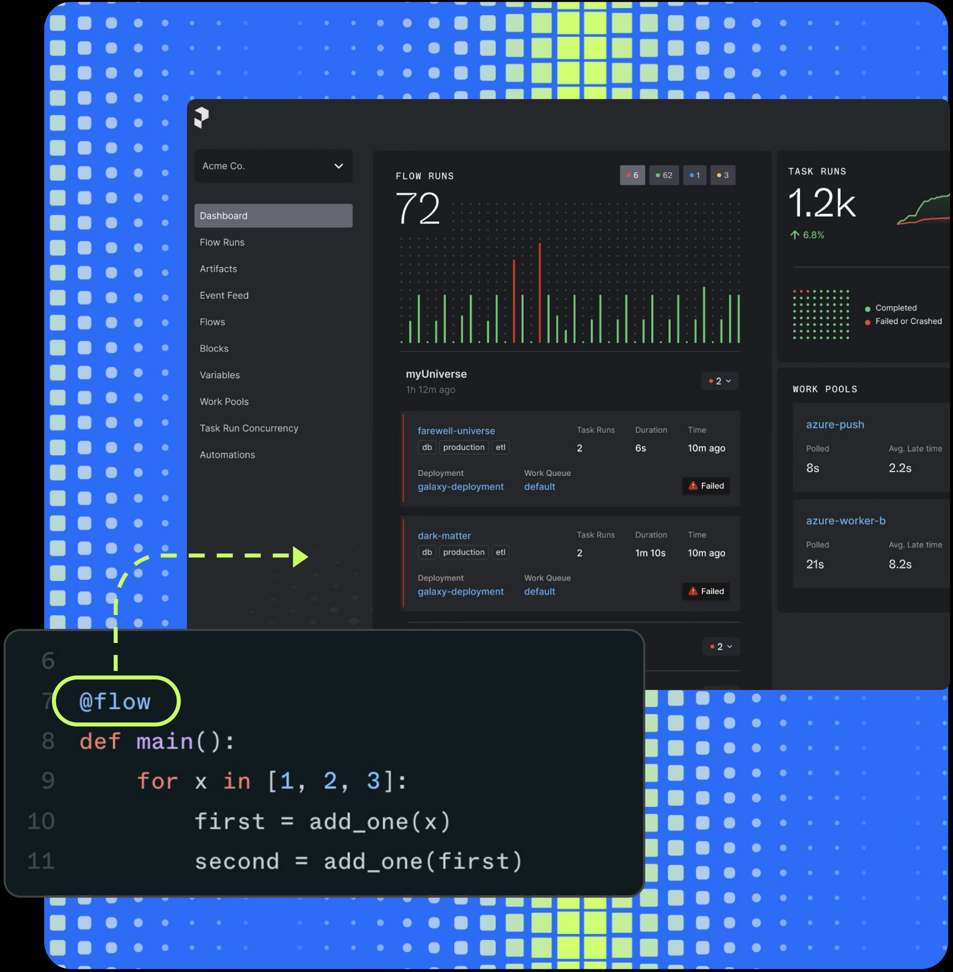
Faster Recovery with Resilient Code

Scalability

Scalable compute resources and governance

Flexible orchestration in Python

- Develop quickly in Python
- Model any workflow
- Run on any compute



myUniverse
1h 12m ago

farewell-universe — db, production, etl — Task Runs: 2 — Duration: 6s — Time: 10m ago
Deployment: galaxy-deployment — Work Queue: default — Failed

dark-matter — db, production, etl — Task Runs: 2 — Duration: 1m 10s — Time: 10m ago
Deployment: galaxy-deployment — Work Queue: default — Failed

azure-push — Polled: 8s — Avg. Late time: 2.2s

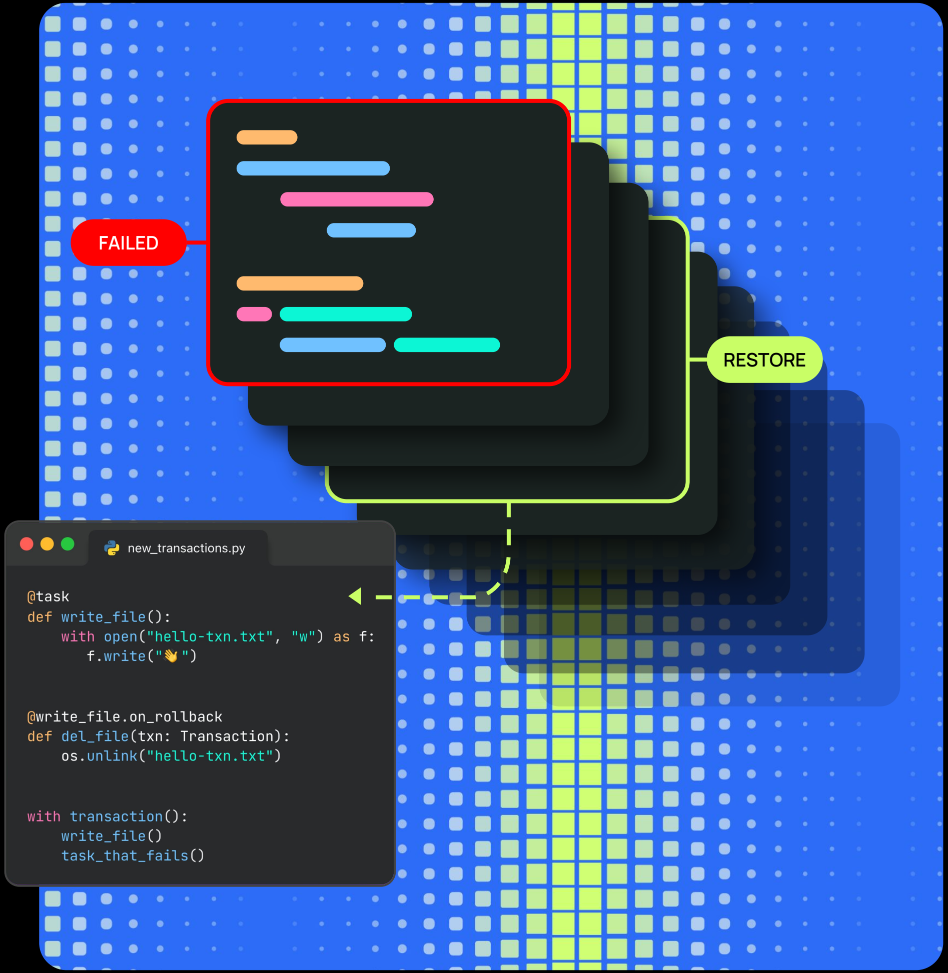
azure-worker-b — Polled: 21s — Avg. Late time: 8.2s

```python

@flow
def main():
    for x in [1, 2, 3]:
        first = add_one(x)
        second = add_one(first)
```

PREFECT

Recover faster with resilient code

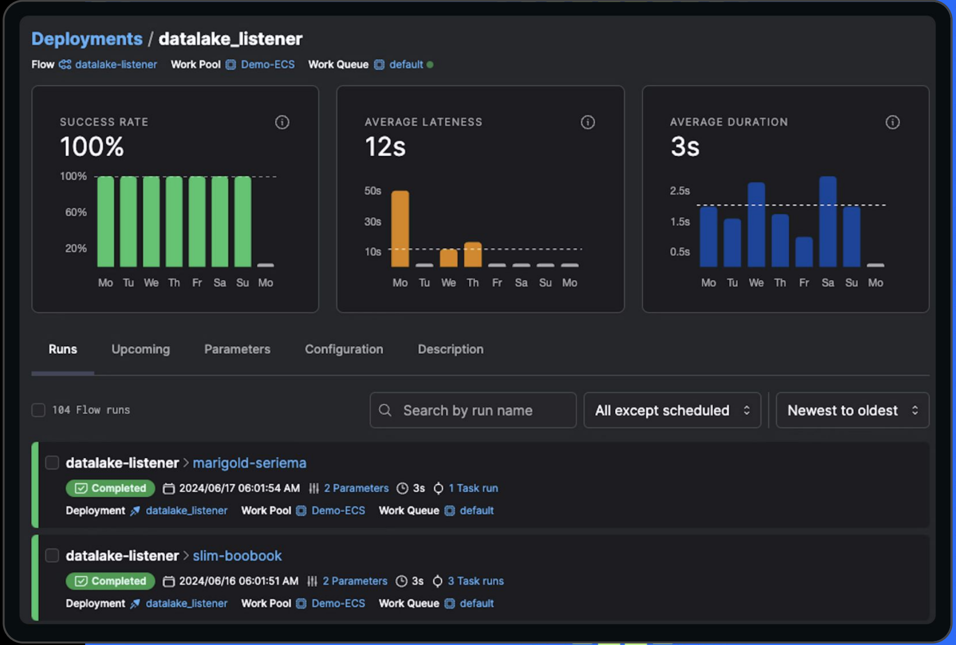
- Gain visibility
- Respond to failure automatically
- Build resilient pipelines



```python
@task
def write_file():
    with open("hello-txn.txt", "w") as f:
       f.write("👋")


@write_file.on_rollback
def del_file(txn: Transaction):
    os.unlink("hello-txn.txt")


with transaction():
    write_file()
    task_that_fails()
```

Scalable compute and governance

- Infrastructure templates
- Limit access appropriately
- Efficiently scale compute


Deployments / datalake_listener

Flow datalake-listener Work Pool Demo-ECS Work Queue default

104 Flow runs

datalake-listener > marigold-seriema
Completed 2024/06/17 06:01:54 AM 2 Parameters 3s 1 Task run
Deployment datalake_listener Work Pool Demo-ECS Work Queue default

datalake-listener > slim-boobook
Completed 2024/06/16 06:01:51 AM 2 Parameters 3s 3 Task runs
Deployment datalake_listener Work Pool Demo-ECS Work Queue default

PREFECT

Orchestration benefits across your business

Data engineers

- Reduce pipeline errors
- Increase productivity through automation
- At-a-glance understanding

Data science & ML engineers

- Iterate on ML models faster
- Reduce data processing time
- Move to production quickly

AI engineers

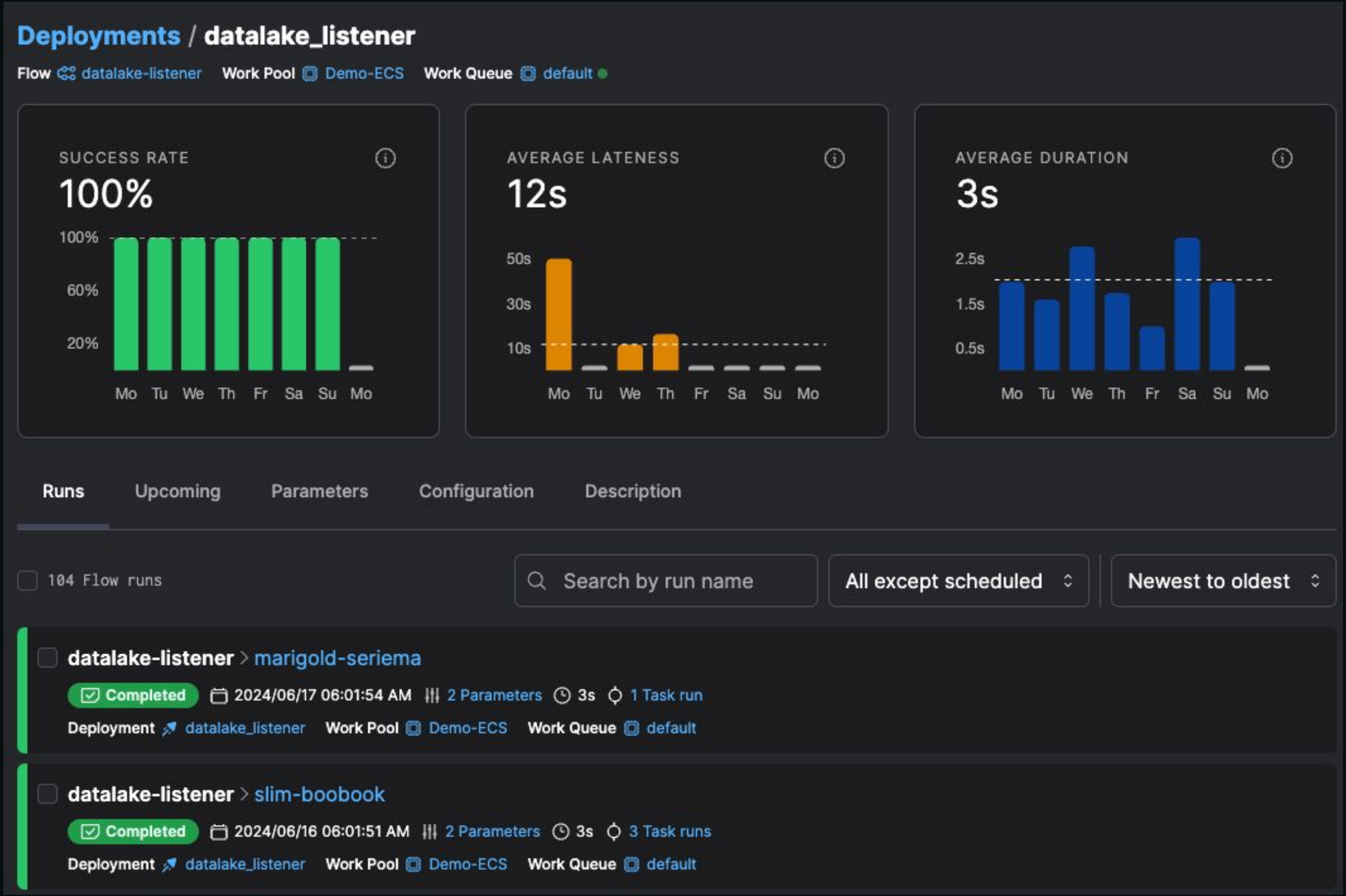
- Improve agentic workflow troubleshooting and auditing
- Unify orchestration across multiple LLM tasks and data sources

Data platform engineers

- Self-serve, turn-key infrastructure setup
- Faster onboarding
- Compute governance
- Leverage!

Outcome: workflows you can trust

- Save time ⏱️
- Save money 💰
- Increase productivity 🚀



Deployments / datalake_listener

Flow datalake-listener Work Pool Demo-ECS Work Queue default

104 Flow runs

datalake-listener > marigold-seriema
Completed 2024/06/17 06:01:54 AM 2 Parameters 3s 1 Task run
Deployment datalake_listener Work Pool Demo-ECS Work Queue default

datalake-listener > slim-boobook
Completed 2024/06/16 06:01:51 AM 2 Parameters 3s 3 Task runs
Deployment datalake_listener Work Pool Demo-ECS Work Queue default

101 Prefect basics: Create a workflow you can schedule and observe

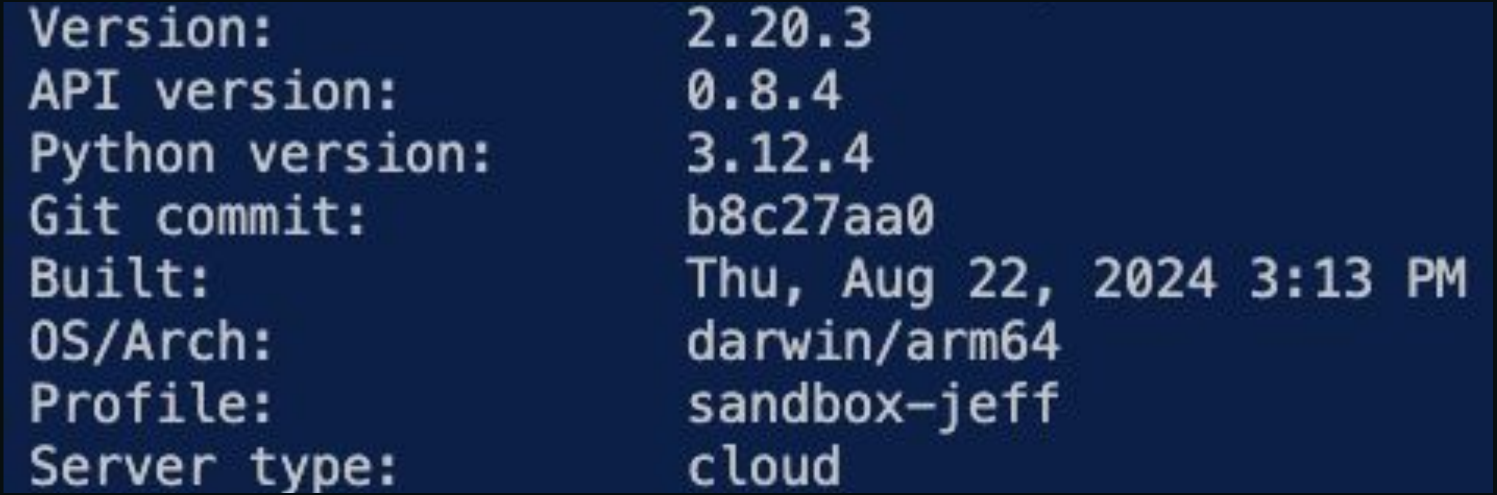
101 Agenda

- Setup: *version, login*
- From Python function to Prefect flow
- Create a deployment with *.serve()*
- Run a deployment
- Deployment schedules
- Parameters
- Resources

prefect version

Prefect information in the CLI

prefect version



```
Version:             2.20.3
API version:         0.8.4
Python version:      3.12.4
Git commit:          b8c27aa0
Built:               Thu, Aug 22, 2024 3:13 PM
OS/Arch:             darwin/arm64
Profile:             sandbox-jeff
Server type:         cloud
```

Run *prefect version* now

If you see *version* lower than *2.20.3*, in your virtual environment:

pip install -U prefect

(You can do this and any of the other items you'll see on upcoming slides during the first lab)

Prefect has two options for server interaction

1. Self-host a Prefect server
    a. You spin up a local server
    b. Backed by SQLite db (or PostgreSQL)
2. Use the Prefect Cloud platform
    a. Free tier
    b. Organization management capabilities on other tiers
    a. Additional features such as event webhooks, push work pools, managed work pools, incidents
    c. No database management required

To the Cloud

Prefect Cloud

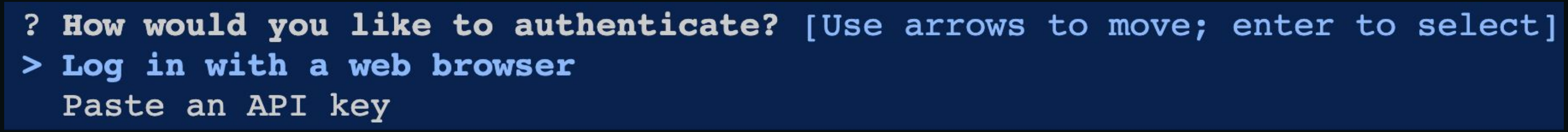
Go to app.prefect.cloud in browser

- Sign up or sign in
- Use a free personal account if you don’t want to use an organization account

Prefect Cloud

Authenticate your CLI

prefect cloud login



```
? How would you like to authenticate? [Use arrows to move; enter to select]
> Log in with a web browser
  Paste an API key
```

Select ***Log in with a web browser***

Creates and saves an API key for you 🔑

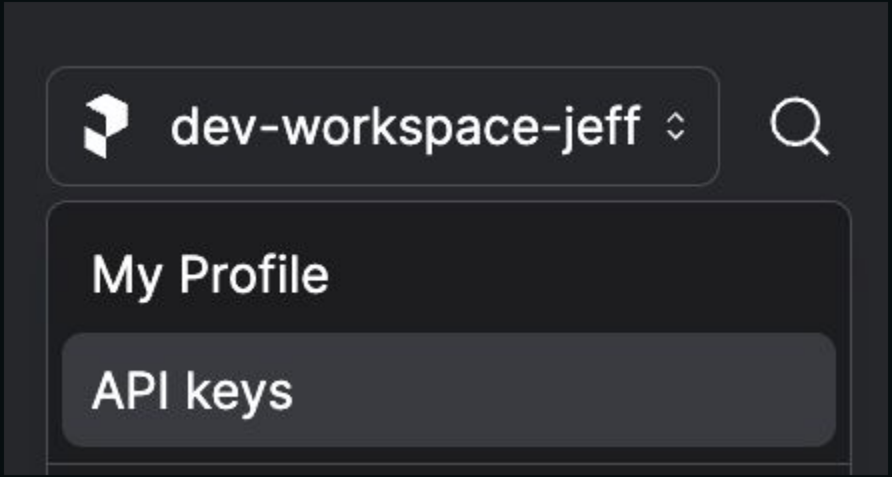
Prefect Cloud

OR, if UI doesn’t work:

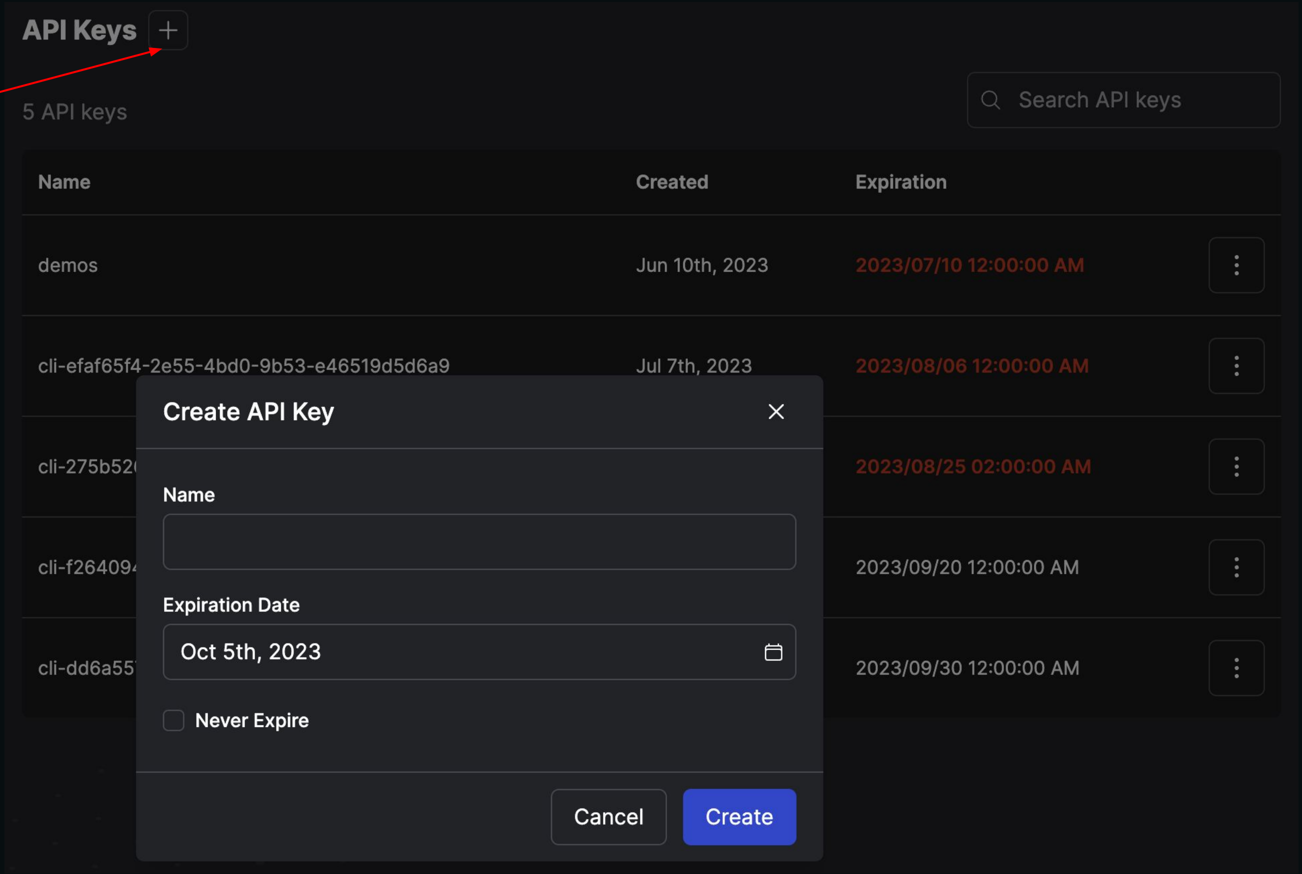
- Select *Paste an API key*
- Manually create an API key from Prefect Cloud in the UI

Prefect Cloud - API key

(Top left of UI)



Prefect Cloud - API key



API Keys +

5 API keys

Search API keys

Name	Created	Expiration
demos	Jun 10th, 2023	2023/07/10 12:00:00 AM
cli-efaf65f4-2e55-4bd0-9b53-e46519d5d6a9	Jul 7th, 2023	2023/08/06 12:00:00 AM
cli-275b52		2023/08/25 02:00:00 AM
cli-f26409		2023/09/20 12:00:00 AM
cli-dd6a55		2023/09/30 12:00:00 AM

Create API Key

Name

Expiration Date

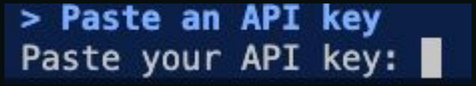
Oct 5th, 2023

☐ Never Expire

Cancel Create

Prefect Cloud

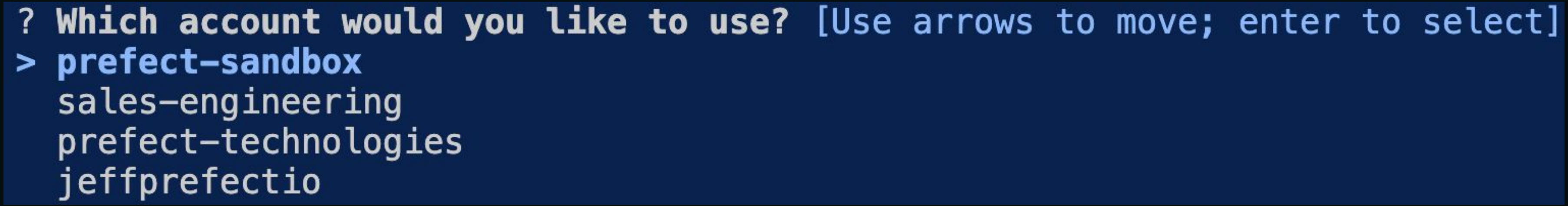
Paste API key at terminal prompt



```
> Paste an API key
Paste your API key:
```

Switch Between Workspaces from the CLI

prefect cloud workspace set



```
? Which account would you like to use? [Use arrows to move; enter to select]
> prefect-sandbox
  sales-engineering
  prefect-technologies
  jeffprefectio
```

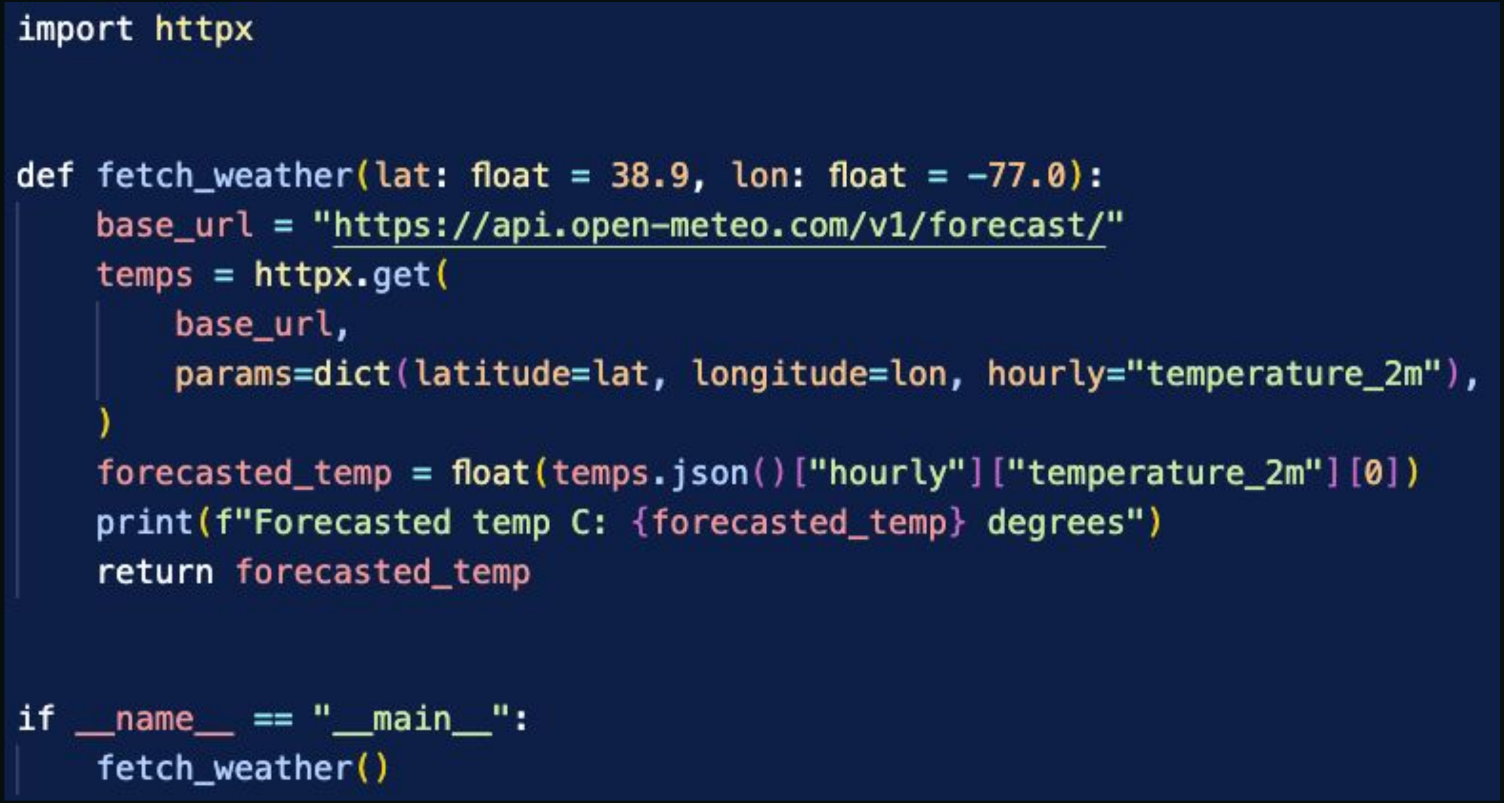
Flows: Add superpowers to your Python 🦸

Project

Fetch and use weather forecast data from Open-Meteo 🌦️ 🌡️

open-meteo.com

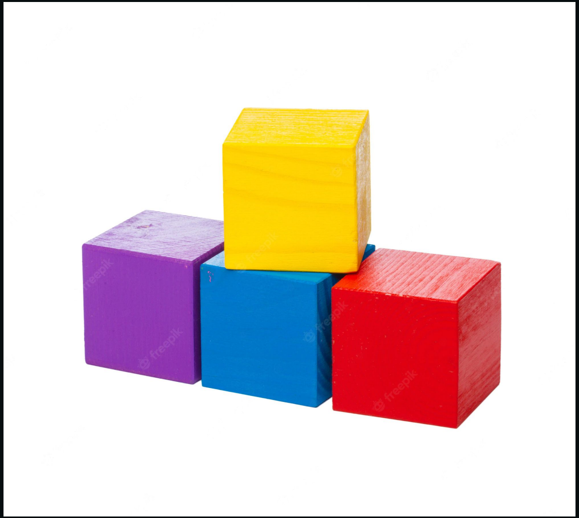
Start: basic Python function



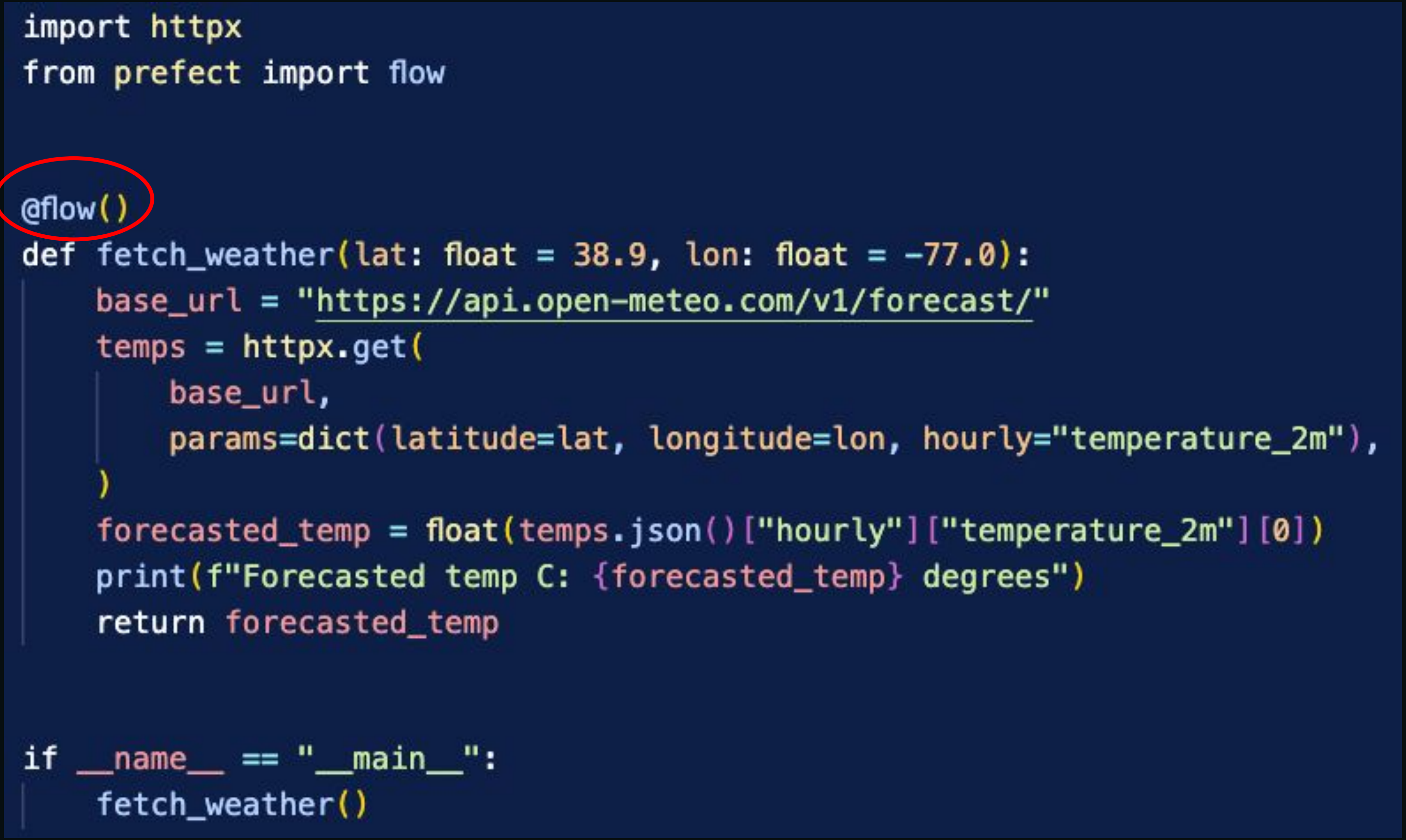
```python
import httpx


def fetch_weather(lat: float = 38.9, lon: float = -77.0):
    base_url = "https://api.open-meteo.com/v1/forecast/"
    temps = httpx.get(
        base_url,
        params=dict(latitude=lat, longitude=lon, hourly="temperature_2m"),
    )
    forecasted_temp = float(temps.json()["hourly"]["temperature_2m"][0])
    print(f"Forecasted temp C: {forecasted_temp} degrees")
    return forecasted_temp


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

Flows

- Add a Prefect *@flow* decorator
- Most basic Prefect object
- All you need to start

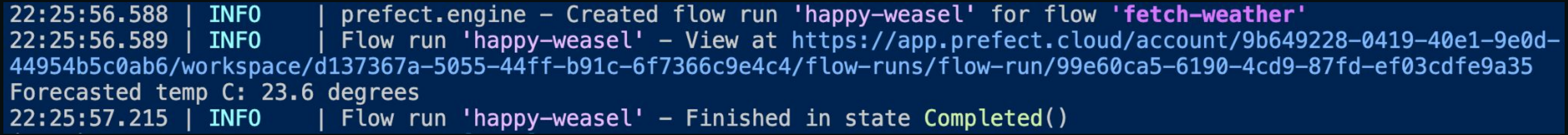


Make it a flow



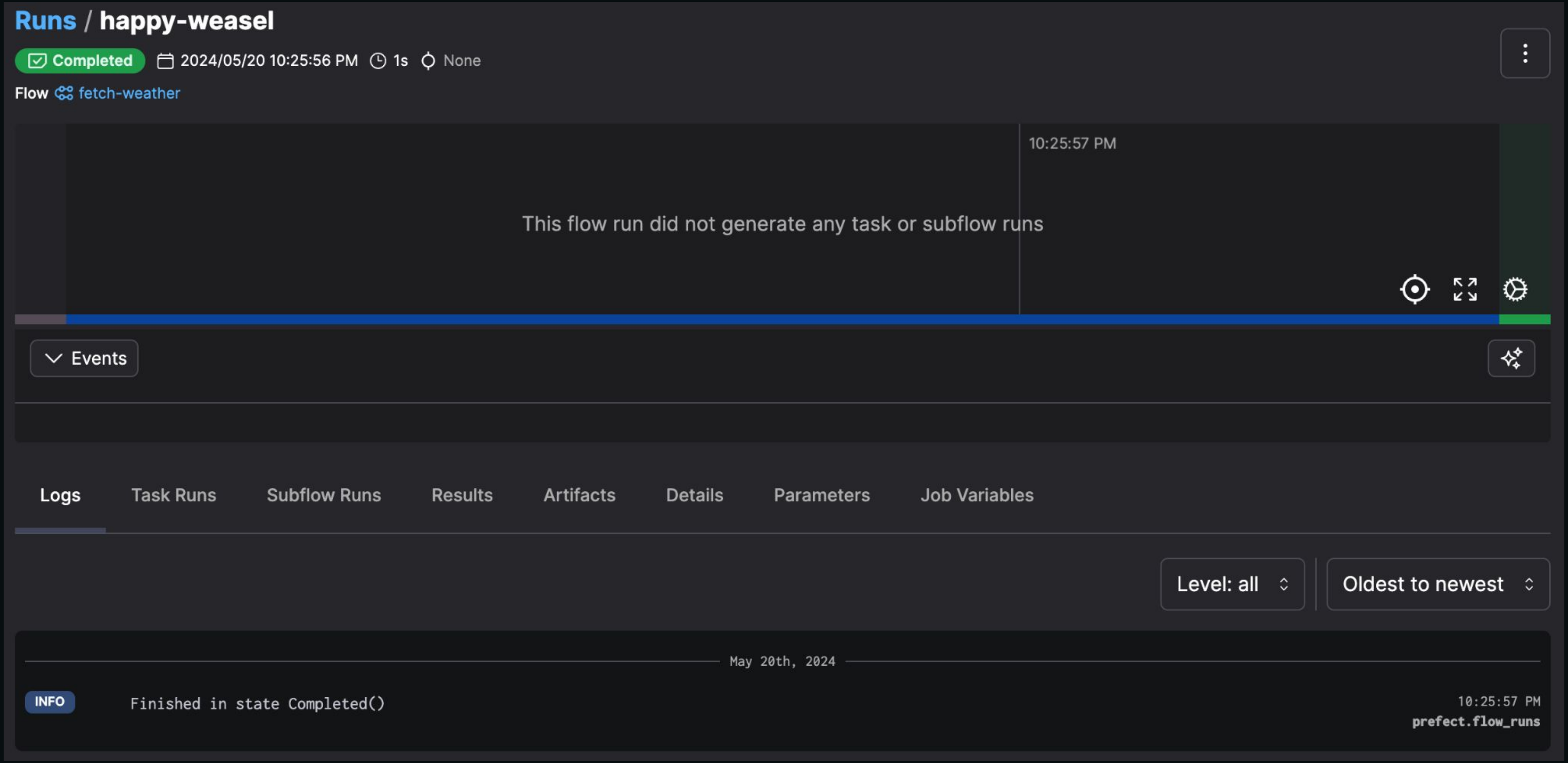
```python
import httpx
from prefect import flow


@flow()
def fetch_weather(lat: float = 38.9, lon: float = -77.0):
    base_url = "https://api.open-meteo.com/v1/forecast/"
    temps = httpx.get(
        base_url,
        params=dict(latitude=lat, longitude=lon, hourly="temperature_2m"),
    )
    forecasted_temp = float(temps.json()["hourly"]["temperature_2m"][0])
    print(f"Forecasted temp C: {forecasted_temp} degrees")
    return forecasted_temp


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

Run the code: *python my_file.py*



```
22:25:56.588 | INFO    | prefect.engine - Created flow run 'happy-weasel' for flow 'fetch-weather'
22:25:56.589 | INFO    | Flow run 'happy-weasel' - View at https://app.prefect.cloud/account/9b649228-0419-40e1-9e0d-44954b5c0ab6/workspace/d137367a-5055-44ff-b91c-6f7366c9e4c4/flow-runs/flow-run/99e60ca5-6190-4cd9-87fd-ef03cdfe9a35
Forecasted temp C: 23.6 degrees
22:25:57.215 | INFO    | Flow run 'happy-weasel' - Finished in state Completed()
```

Check it out your flow run from the **Runs** page in the UI


Runs / happy-weasel

Completed 2024/05/20 10:25:56 PM 1s None

Flow fetch-weather

10:25:57 PM

This flow run did not generate any task or subflow runs

Events

Logs | Task Runs | Subflow Runs | Results | Artifacts | Details | Parameters | Job Variables

Level: all   Oldest to newest

```
May 20th, 2024
INFO    Finished in state Completed()                10:25:57 PM
                                                  prefect.flow_runs
```

Flows give you

- Auto logging
- State tracking info sent to API
- Input arguments type checked/coerced
- Timeouts can be enforced
- Lots of other benefits you’ll see soon 🚀

Deployments

Deployments

Turn your workflow into an interactive application! 🎉

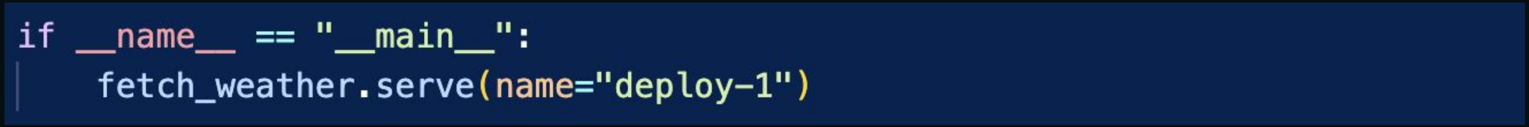
- Switch infrastructure easily
- You and teammates can run:
    - manually (from the UI or CLI)
    - on a schedule
    - in response to an automation trigger

Deployments

- Server-side representation of a flow
- Contains metadata for remote orchestration
- Your flow's passport to orchestration land!

.serve() method

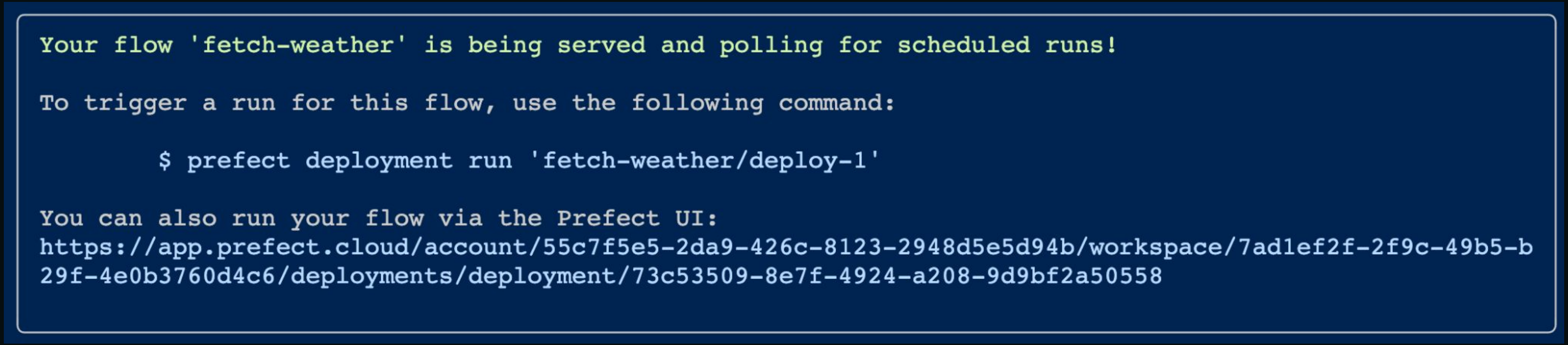
Create a deployment by calling the flow function’s *.serve()* method.



```python
if __name__ == "__main__":
    fetch_weather.serve(name="deploy-1")
```

.serve() method

Run the script - creates a deployment and starts a server

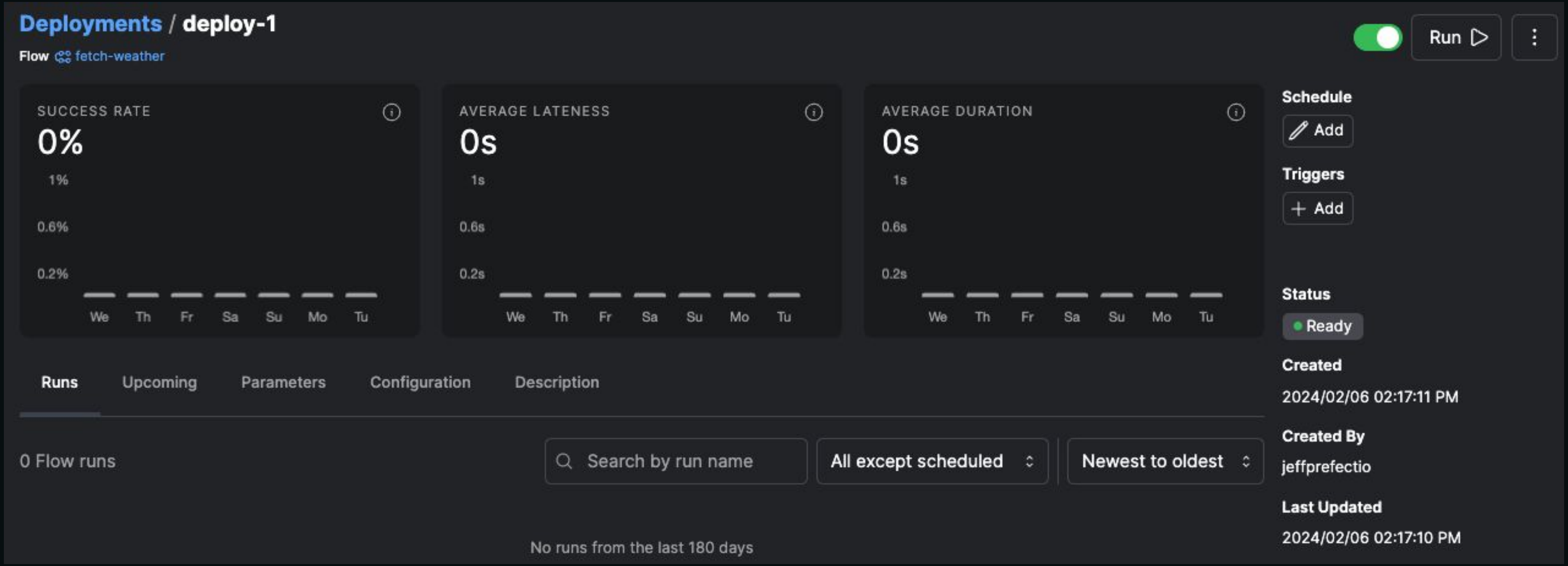


```
Your flow 'fetch-weather' is being served and polling for scheduled runs!

To trigger a run for this flow, use the following command:

        $ prefect deployment run 'fetch-weather/deploy-1'

You can also run your flow via the Prefect UI:
https://app.prefect.cloud/account/55c7f5e5-2da9-426c-8123-2948d5e5d94b/workspace/7ad1ef2f-2f9c-49b5-b
29f-4e0b3760d4c6/deployments/deployment/73c53509-8e7f-4924-a208-9d9bf2a50558
```

You just made a deployment!

Check out the deployment in the UI

Deployment page



Deployments / deploy-1

Flow fetch-weather

Status: Ready

Created: 2024/02/06 02:17:11 PM

Created By: jeffprefectio

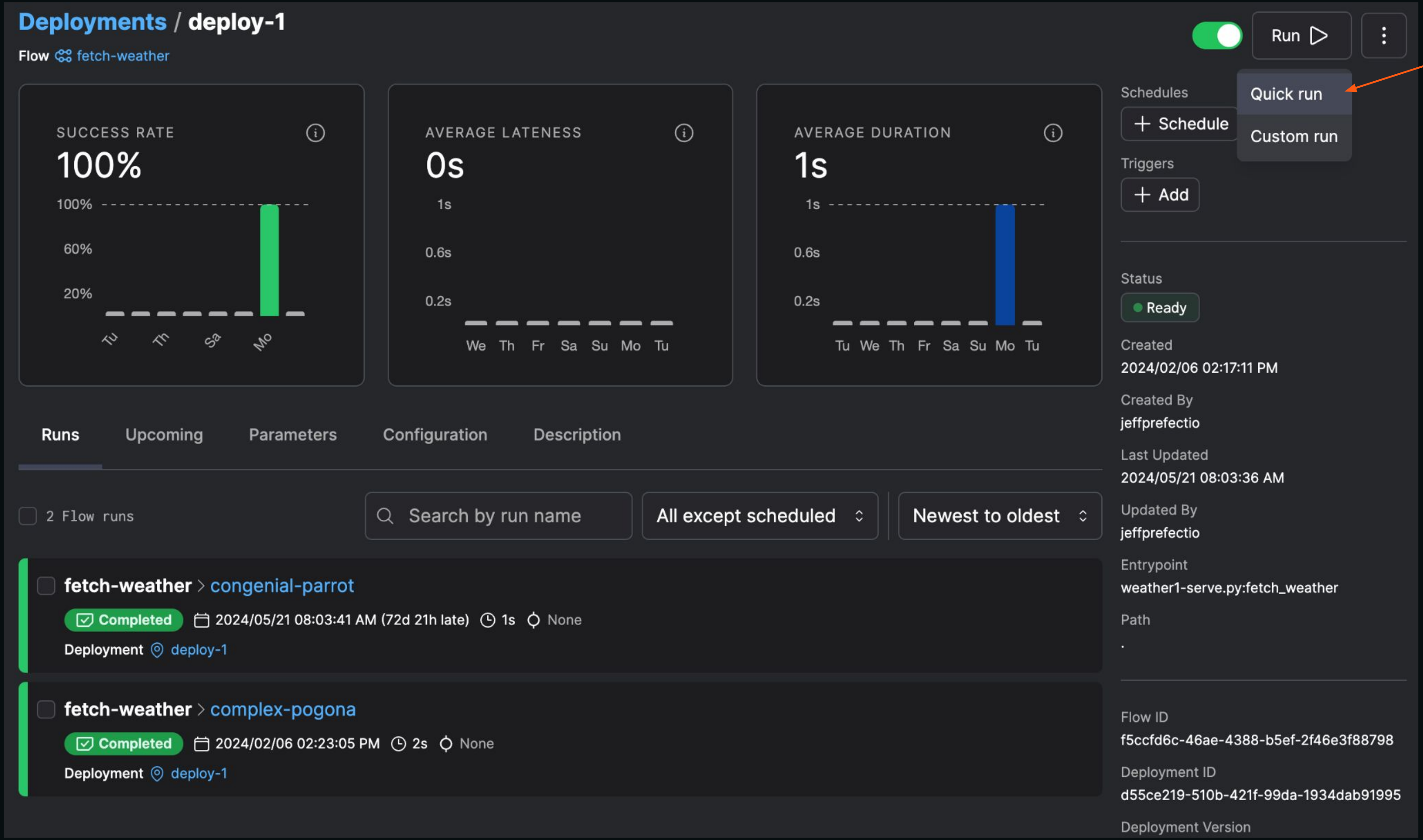
Last Updated: 2024/02/06 02:17:10 PM

0 Flow runs

No runs from the last 180 days

Run a deployment

Run manually from UI: **Run -> Quick run**



Deployments / **deploy-1**

Flow fetch-weather

2 Flow runs

fetch-weather > congenial-parrot
Completed 2024/05/21 08:03:41 AM (72d 21h late) 1s None
Deployment deploy-1

fetch-weather > complex-pogona
Completed 2024/02/06 02:23:05 PM 2s None
Deployment deploy-1

Status
Ready

Created
2024/02/06 02:17:11 PM

Created By
jeffprefectio

Last Updated
2024/05/21 08:03:36 AM

Updated By
jeffprefectio

Entrypoint
weather1-serve.py:fetch_weather

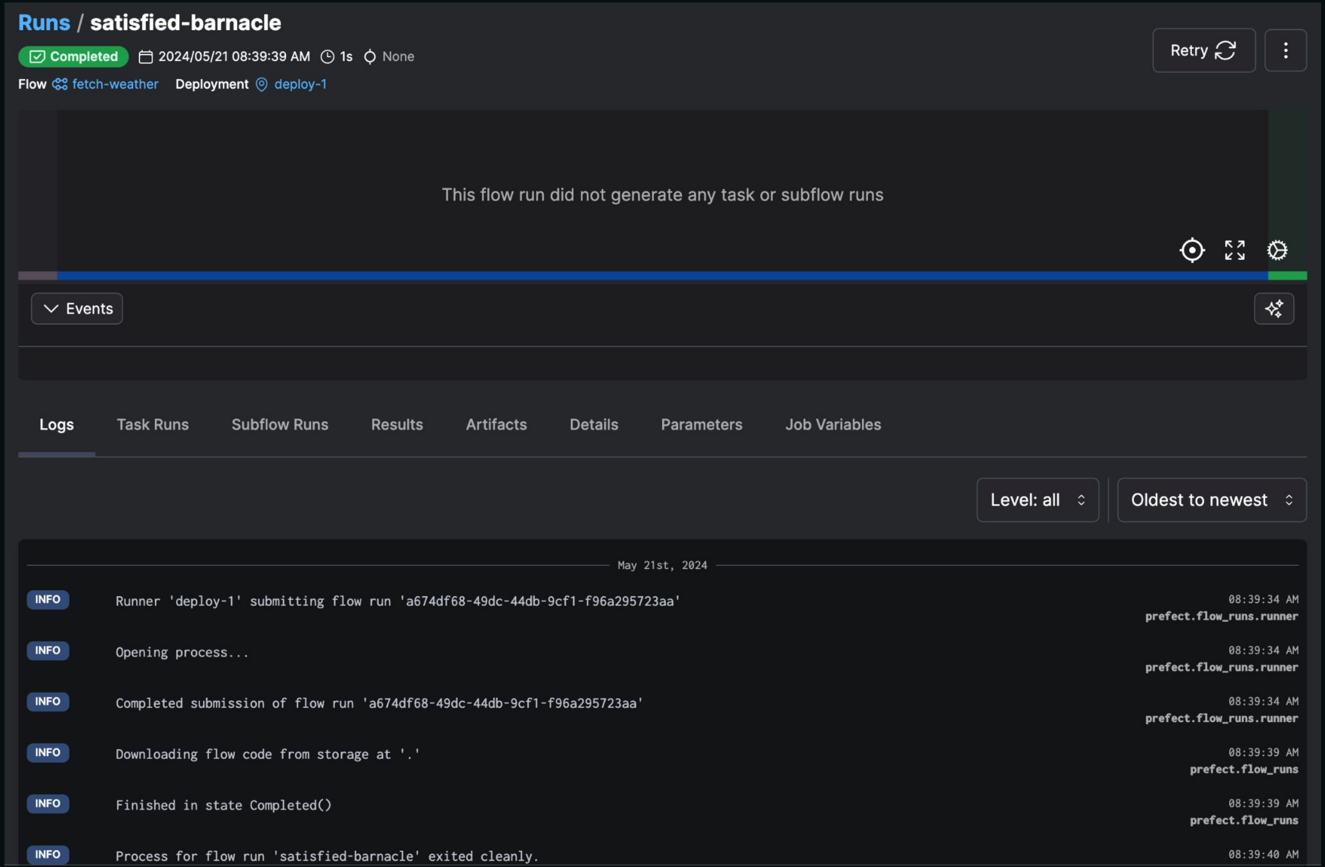
Path
.

Flow ID
f5ccfd6c-46ae-4388-b5ef-2f46e3f88798

Deployment ID
d55ce219-510b-421f-99da-1934dab91995

Deployment Version

View the flow run logs in the UI (or CLI)



```
May 21st, 2024
INFO  Runner 'deploy-1' submitting flow run 'a674df68-49dc-44db-9cf1-f96a295723aa'   08:39:34 AM
                                                                       prefect.flow_runs.runner
INFO  Opening process...                                                             08:39:34 AM
                                                                       prefect.flow_runs.runner
INFO  Completed submission of flow run 'a674df68-49dc-44db-9cf1-f96a295723aa'        08:39:34 AM
                                                                       prefect.flow_runs.runner
INFO  Downloading flow code from storage at '.'                                      08:39:39 AM
                                                                              prefect.flow_runs
INFO  Finished in state Completed()                                                  08:39:39 AM
                                                                              prefect.flow_runs
INFO  Process for flow run 'satisfied-barnacle' exited cleanly.                      08:39:40 AM
```

Run deployment manually from CLI

```
prefect deployment run my_entrypoint_flow:my_deployment
```

.serve()

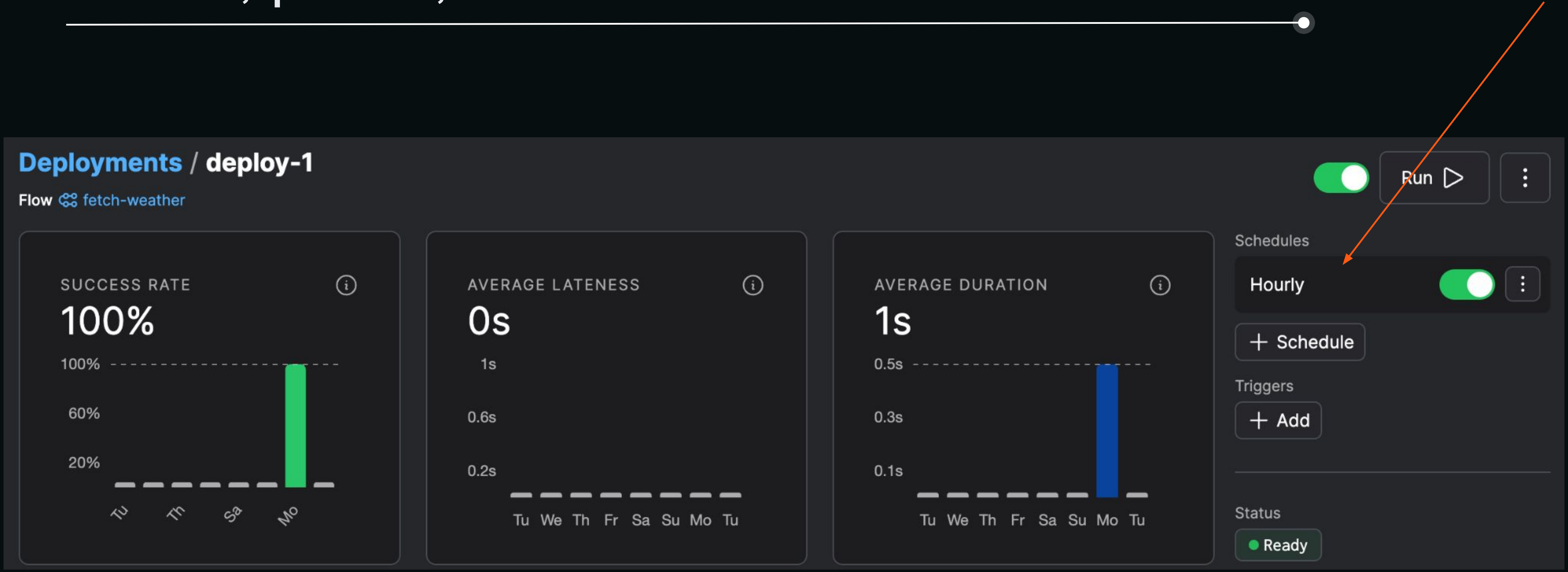
Shut down the server with ***control* + *c***

Scheduling

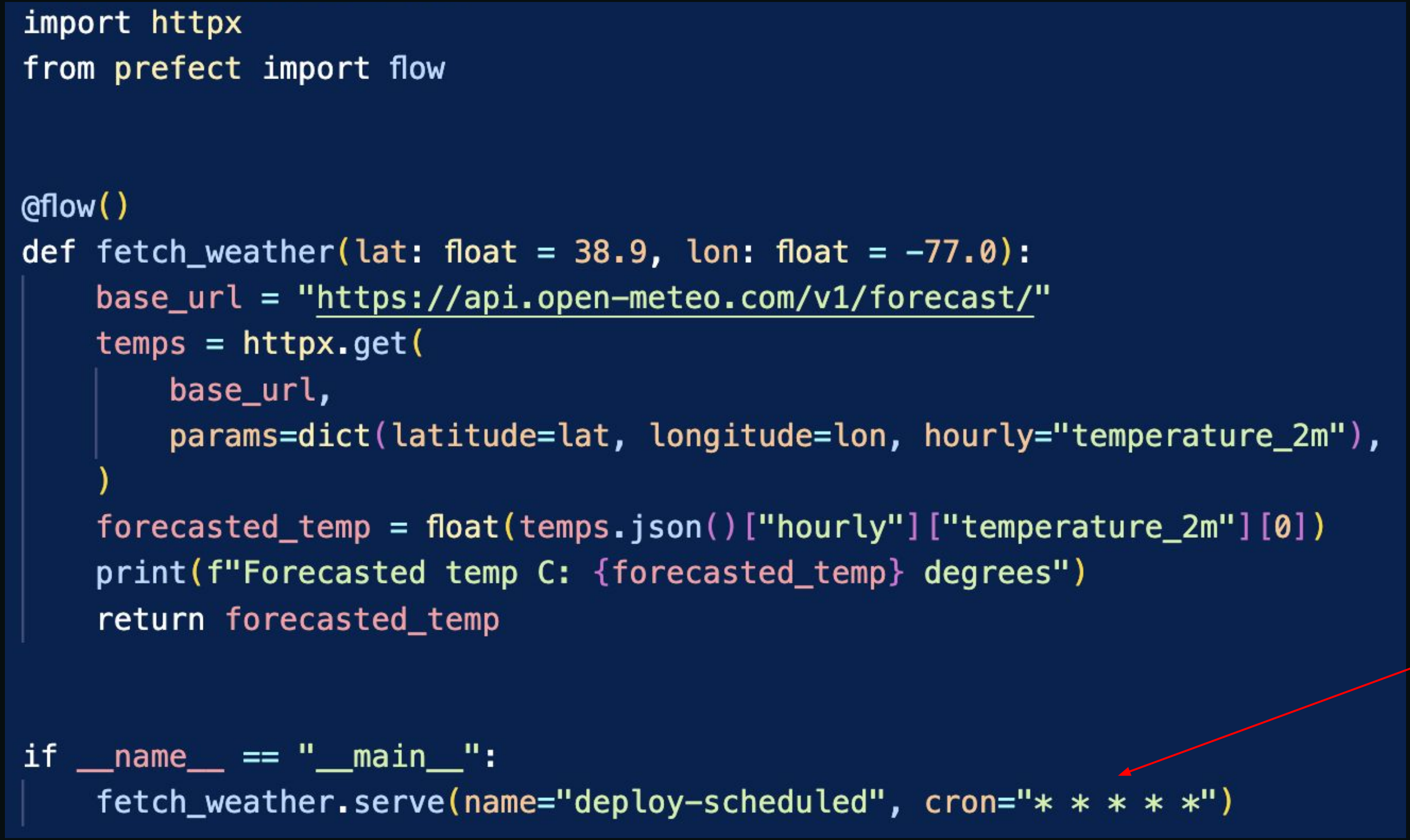
Create a deployment schedule

1. When creating a deployment
2. After deployment creation in the UI or CLI

Create, pause, and delete schedules from the UI



Add a schedule when creating a deployment with *.serve()*

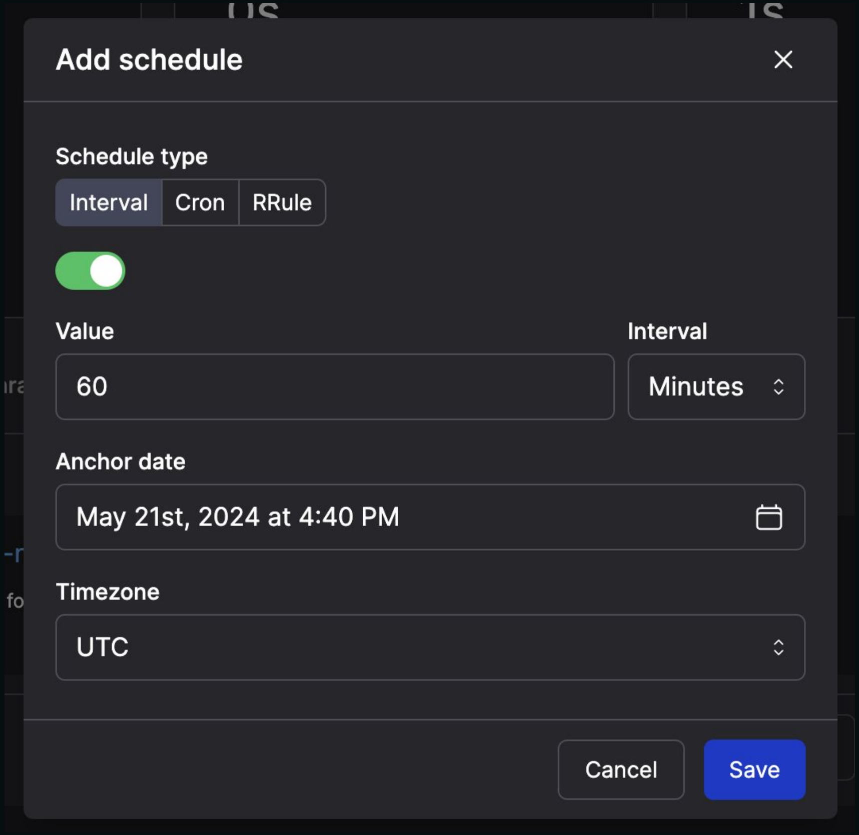


```python
import httpx
from prefect import flow


@flow()
def fetch_weather(lat: float = 38.9, lon: float = -77.0):
    base_url = "https://api.open-meteo.com/v1/forecast/"
    temps = httpx.get(
        base_url,
        params=dict(latitude=lat, longitude=lon, hourly="temperature_2m"),
    )
    forecasted_temp = float(temps.json()["hourly"]["temperature_2m"][0])
    print(f"Forecasted temp C: {forecasted_temp} degrees")
    return forecasted_temp


if __name__ == "__main__":
    fetch_weather.serve(name="deploy-scheduled", cron="* * * * *")
```

Schedule types

- Interval
- Cron
- RRule

Choose **Interval** or **Cron** if in the UI



Add schedule

Schedule type

Interval | Cron | RRule

Value: 60

Interval: Minutes

Anchor date: May 21st, 2024 at 4:40 PM

Timezone: UTC

RRule

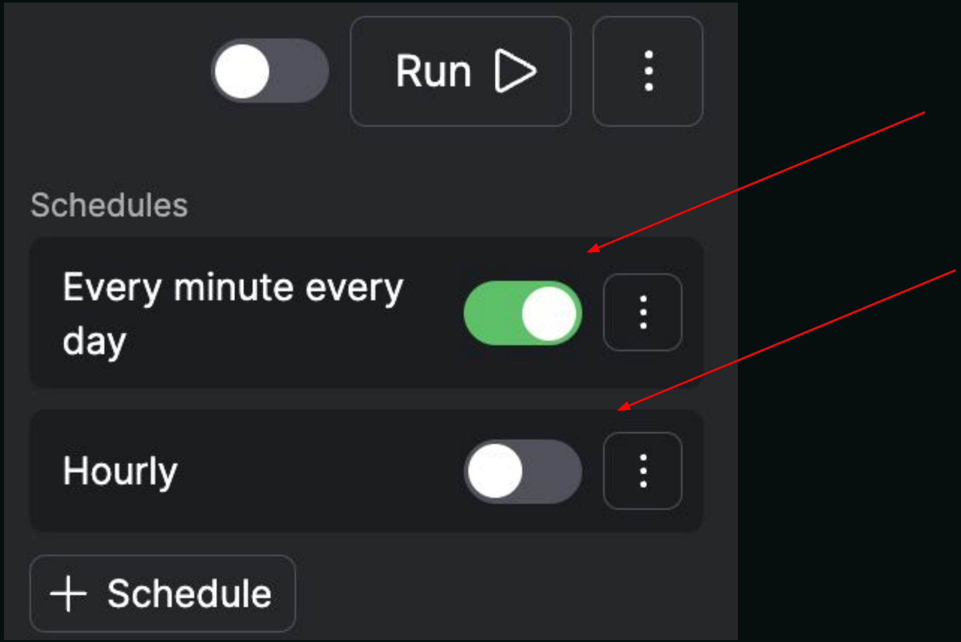
RRule cheat sheet: https://jkbrzt.github.io/rrule/

Or ask Marvin (another Prefect package) *pip install -U marvin*

```python
from marvin import ai_fn

@ai_fn
def rrule(text: str) -> str:
    """
    Generate valid RRULE strings from a natural language description of an event
    """
    yield pendulum.now.isoformat()

rrule('every hour from 9-6 on thursdays')
# "RRULE:FREQ=WEEKLY;BYDAY=TH;BYHOUR=9,10,11,12,13,14,15,16;BYMINUTE=0;BYSECOND=0"
```

Pausing and resuming deployment schedules

Pause/resume deployment schedules from UI



Note ⏸️

Shutting down your server with *.serve()* pauses a deployment’s schedules

Parameters

Parameters - argument values for entrypoint flow function

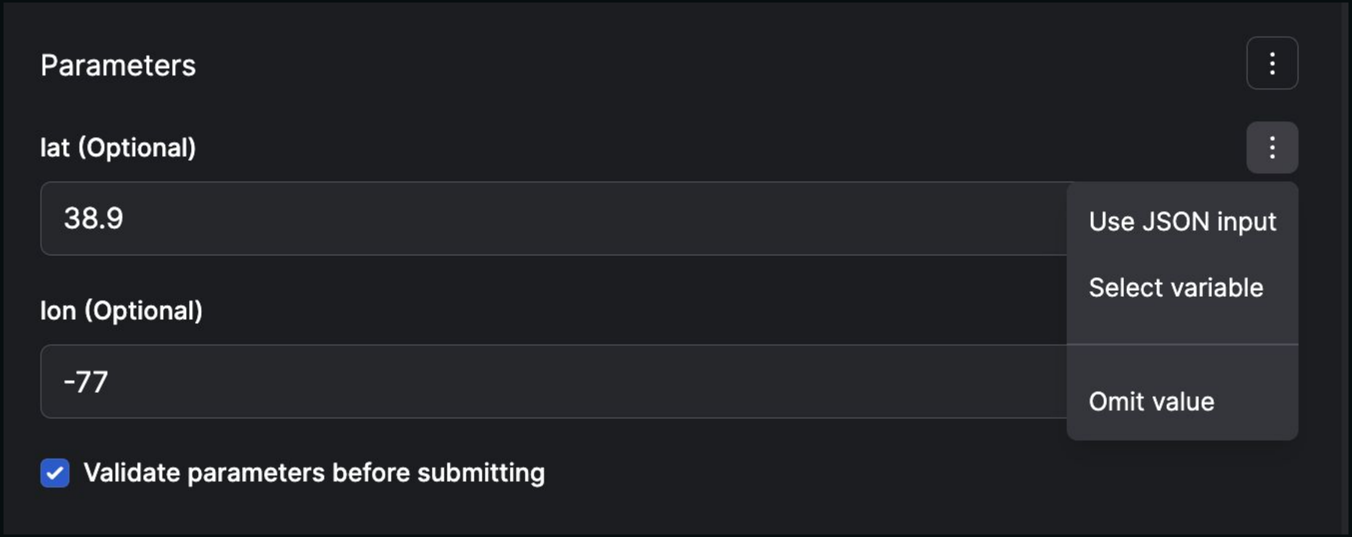
If your flow function has params and no defaults, you must feed it (give it values).

Parameter options

1. Make default arguments in flow function definition
2. Can override at deployment creation
3. Can override both of the above at runtime

Parameters in the UI at runtime

Collaborators can run with custom values in a **Custom run** in the UI


Parameters

lat (Optional)

38.9

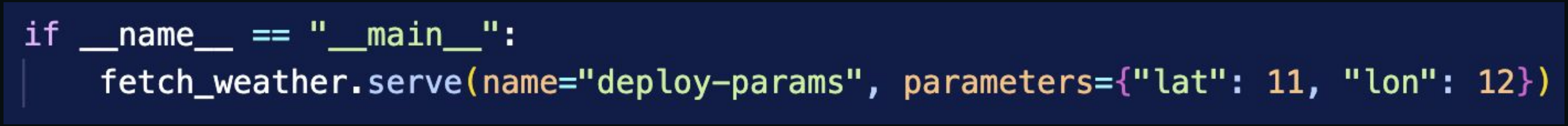
lon (Optional)

-77

✓ Validate parameters before submitting

Parameters at deployment creation time

Can specify in *.serve()*



```python
if __name__ == "__main__":
    fetch_weather.serve(name="deploy-params", parameters={"lat": 11, "lon": 12})
```

Parameters from the CLI at runtime

prefect deployment run parametrized/dev --param user=Marvin --param answer=42

OR

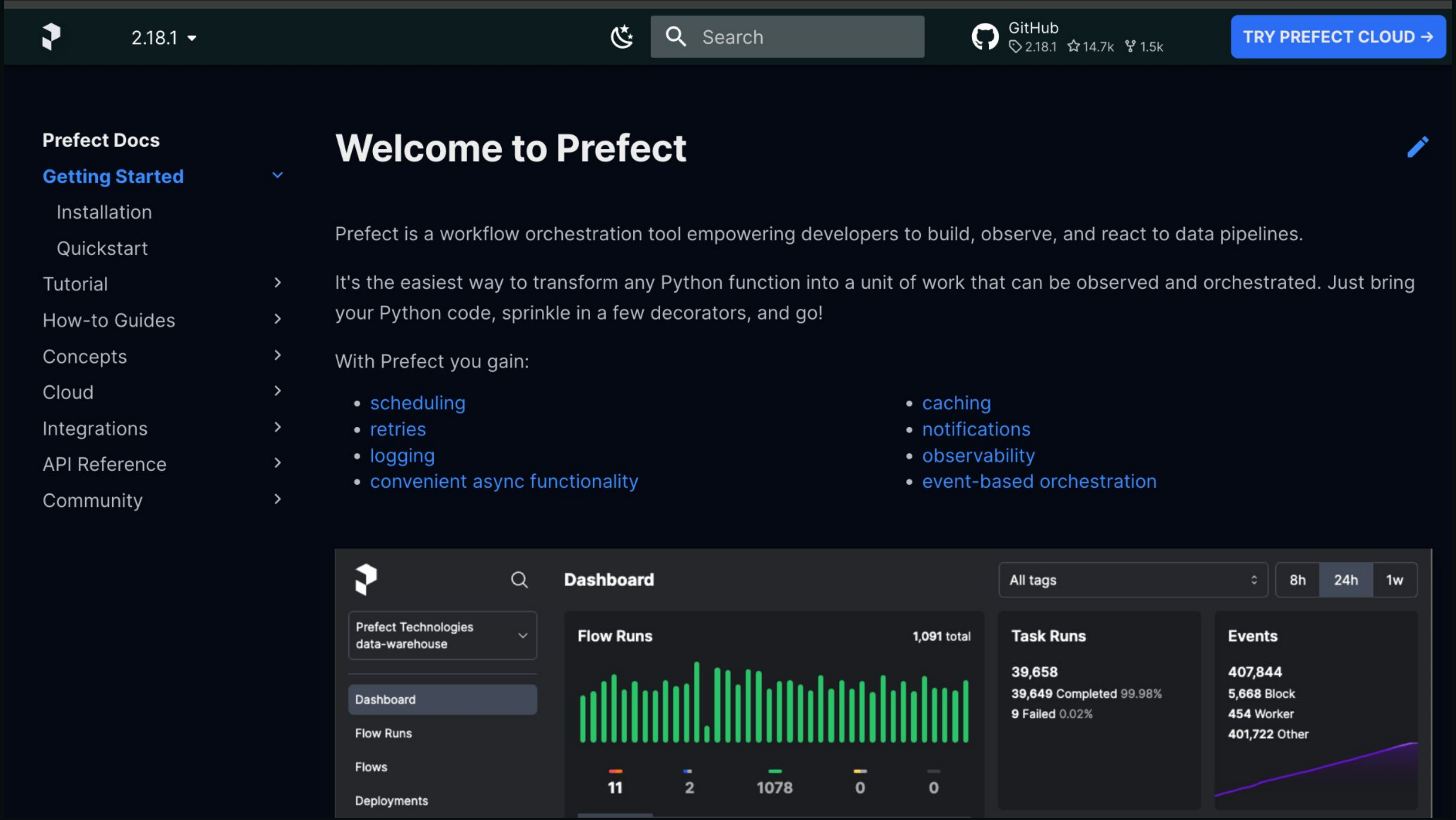
prefect deployment run parametrized/dev --params '{"user": "Marvin", "answer": 42}'

Resources

Docs - *docs.prefect.io*

Use the docs

Docs - *docs.prefect.io*



2.18.1

Search

GitHub 2.18.1 14.7k 1.5k

TRY PREFECT CLOUD →

Prefect Docs
- **Getting Started**
  - Installation
  - Quickstart
- Tutorial
- How-to Guides
- Concepts
- Cloud
- Integrations
- API Reference
- Community

Welcome to Prefect

Prefect is a workflow orchestration tool empowering developers to build, observe, and react to data pipelines.

It's the easiest way to transform any Python function into a unit of work that can be observed and orchestrated. Just bring your Python code, sprinkle in a few decorators, and go!

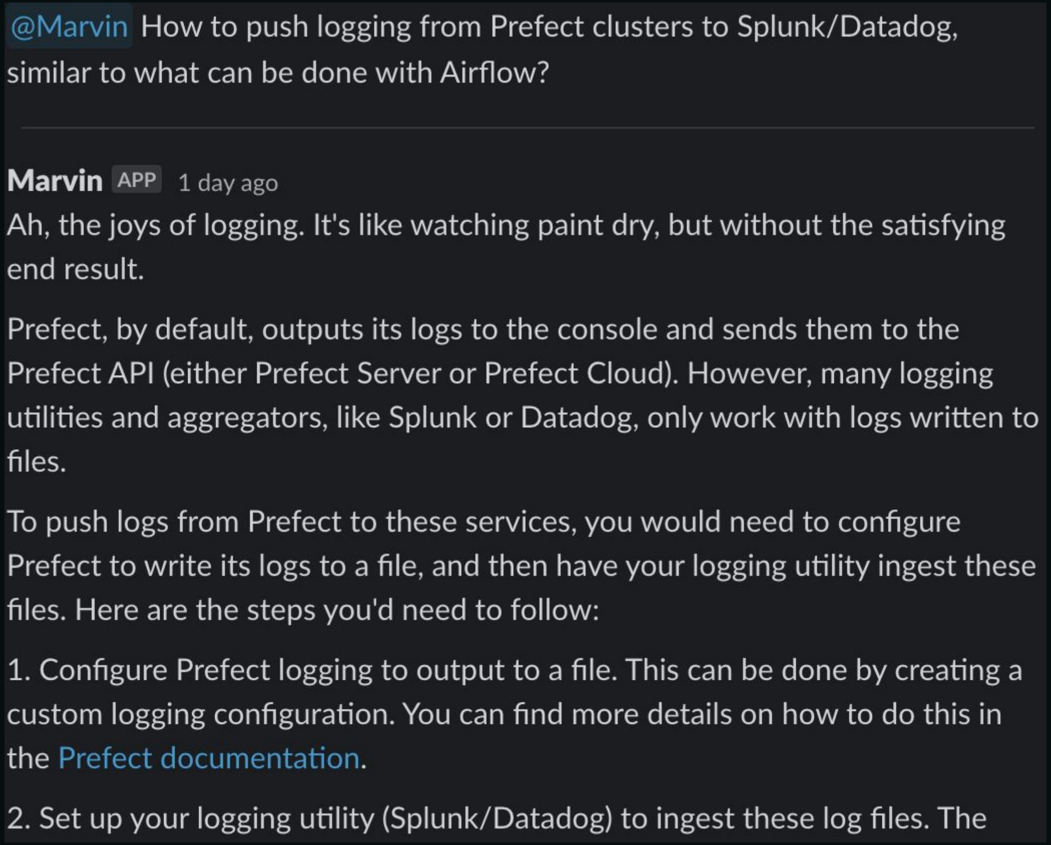
With Prefect you gain:

- scheduling
- retries
- logging
- convenient async functionality
- caching
- notifications
- observability
- event-based orchestration

Prefect Community Slack

Helpful Q & A from the community and Prefect staff

#ask-marvin: Community Slack channel



@Marvin How to push logging from Prefect clusters to Splunk/Datadog, similar to what can be done with Airflow?

Marvin APP 1 day ago

Ah, the joys of logging. It's like watching paint dry, but without the satisfying end result.

Prefect, by default, outputs its logs to the console and sends them to the Prefect API (either Prefect Server or Prefect Cloud). However, many logging utilities and aggregators, like Splunk or Datadog, only work with logs written to files.

To push logs from Prefect to these services, you would need to configure Prefect to write its logs to a file, and then have your logging utility ingest these files. Here are the steps you'd need to follow:

1. Configure Prefect logging to output to a file. This can be done by creating a custom logging configuration. You can find more details on how to do this in the Prefect documentation.

2. Set up your logging utility (Splunk/Datadog) to ingest these log files. The

Prefect codebase

github.com/PrefectHQ/prefect

- Dig into the code
- Create an issue
- Make a PR
- Give it a ⭐

101 Recap

You've seen how to get started with Prefect!

- *prefect version, login*
- From Python function to Prefect flow
- Create a deployment with *flow.serve()*
- Run a deployment
- Create and pause schedules
- Resources: docs, Slack, Prefect GitHub repo

Recap key terms

Flow = a workflow

Flow run = an individual run of a flow

Deployment = a flow + orchestration capabilities

- Can schedule
- Can run remotely
- Other team members can access

Lab 101

Lab norms for breakout rooms

1. 🙂 Introduce yourselves
2. 🎥 Camera on (if possible)
3. 💻 One person shares screen (if need to leave Zoom to enable screen sharing, do that now)
4. 🧑‍💻 Everyone codes
5. 🙋 Ask a question if you don’t follow something
6. 😌 Low-pressure, welcoming environment: lean in

101 Lab - ❗ **see course GitHub repo for example code**

Use Open-Meteo API:

- Authenticate your CLI to Prefect Cloud
- Fine to use a personal account or an organization test workspace
- Take a function that fetches data and make it a flow
- Use *.serve()* method to deploy your flow
- Run your flow from the UI
- Create a schedule for your deployment
- Shut down your server and restart it
- Stretch 1: Run a deployment from the CLI, override the params

API docs: open-meteo.com/en/docs

Example: wind speed for the last hour:

weather.json()["hourly"]["windspeed_10m"][0]

102 - Orchestration and observation: Understand workflow state and guard against failure

PREFECT

102 Agenda

- Tasks
- Logging - observe
- Runtime context - introspect runs
- Retries - automatically retry on failure
- States - understand your workflow state
- Blocks - save configuration with a handy form
- More resources

Tasks

Tasks

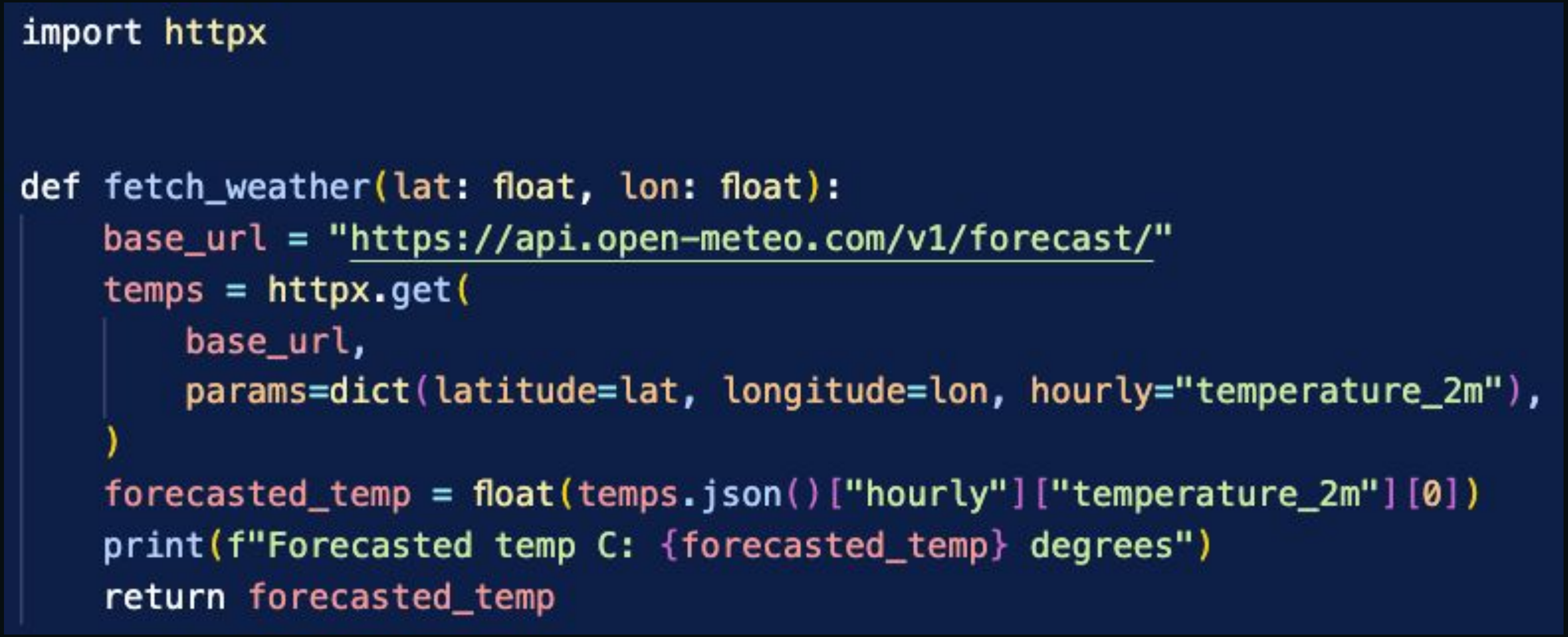
Add the *@task* decorator to a function to enable

- Task retries
- Caching
- Easy async

Starting Point: example pipeline functions

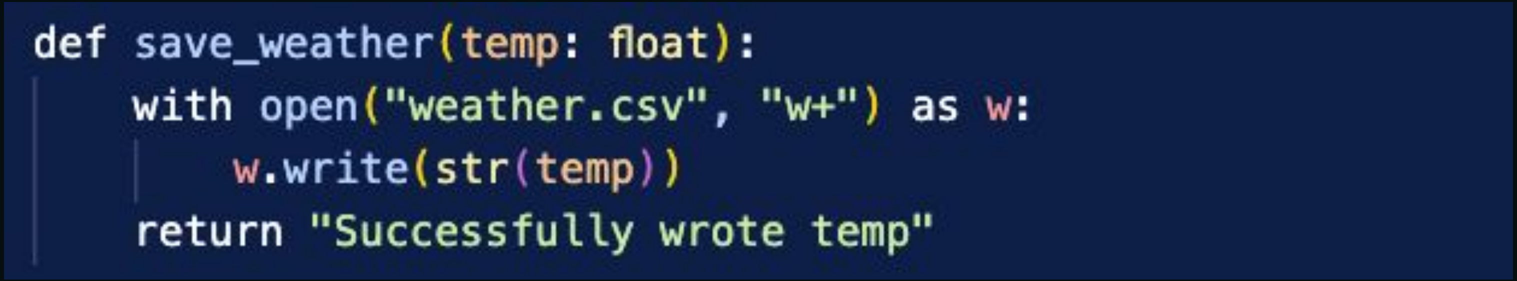
1. Fetch weather data and return it ✅
2. Save data to csv and return success message 🙂
3. Pipeline to call 1 and 2 📞

Fetch data function



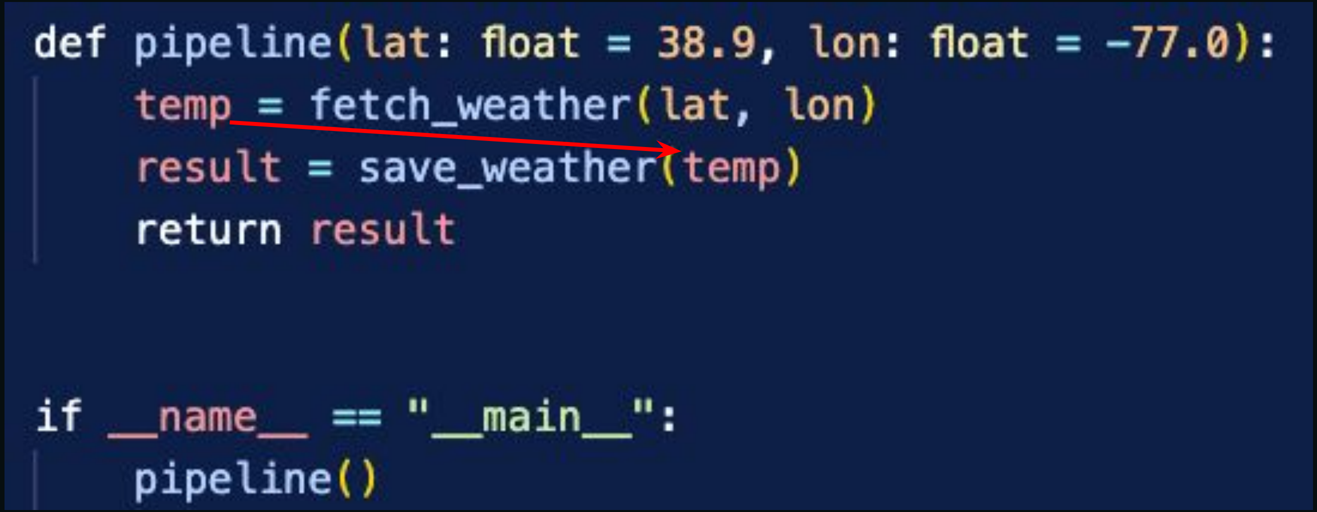
```python
import httpx


def fetch_weather(lat: float, lon: float):
    base_url = "https://api.open-meteo.com/v1/forecast/"
    temps = httpx.get(
        base_url,
        params=dict(latitude=lat, longitude=lon, hourly="temperature_2m"),
    )
    forecasted_temp = float(temps.json()["hourly"]["temperature_2m"][0])
    print(f"Forecasted temp C: {forecasted_temp} degrees")
    return forecasted_temp
```

Save data function



```python
def save_weather(temp: float):
    with open("weather.csv", "w+") as w:
        w.write(str(temp))
    return "Successfully wrote temp"
```

Pipeline (assembly) function

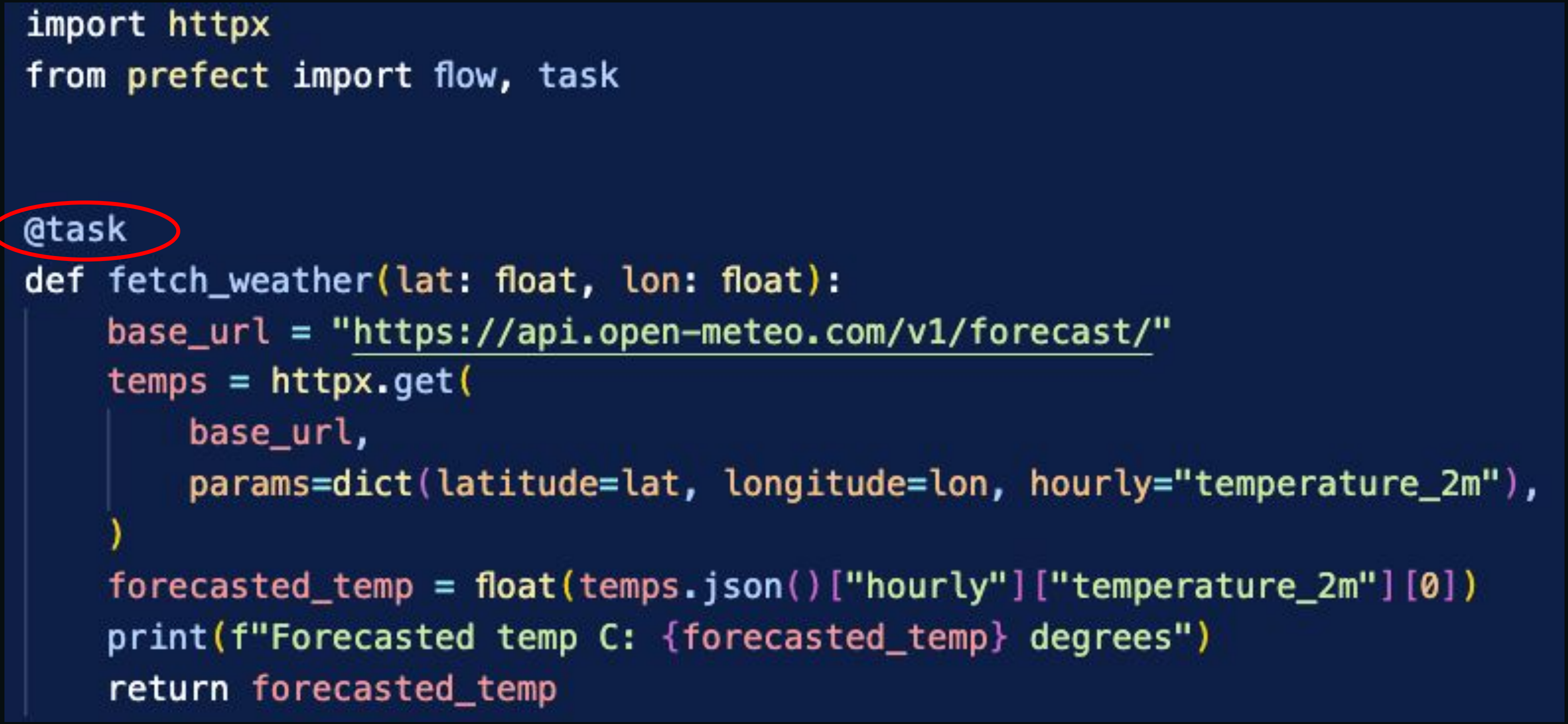


```python
def pipeline(lat: float = 38.9, lon: float = -77.0):
    temp = fetch_weather(lat, lon)
    result = save_weather(temp)
    return result


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

Tasks

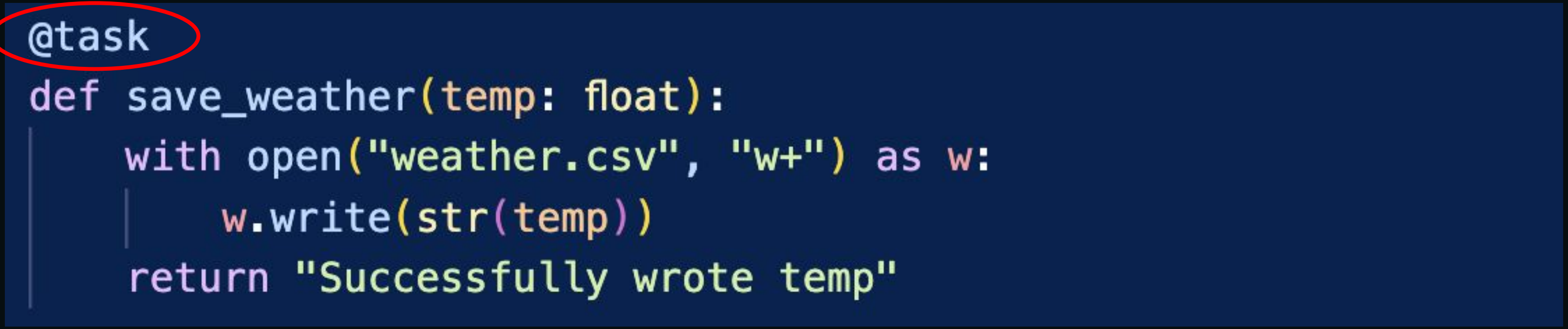
Turn the first two functions into *tasks* with the *@task* decorator

Turn into a task



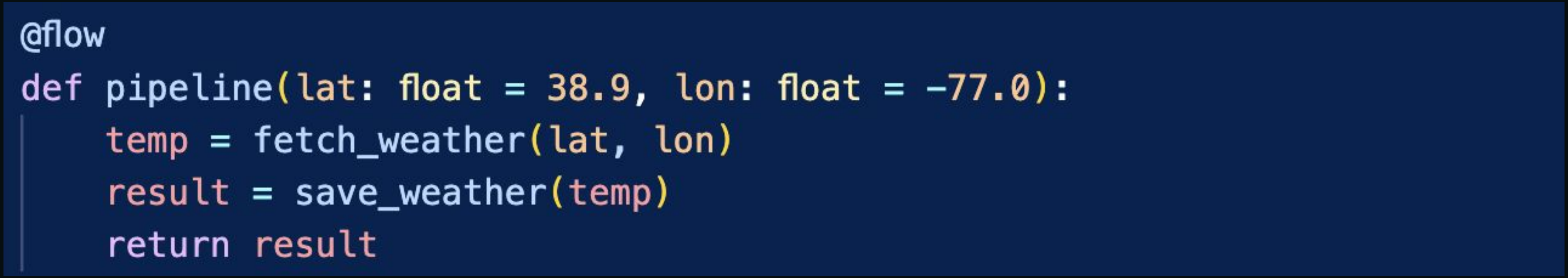
```python
import httpx
from prefect import flow, task


@task
def fetch_weather(lat: float, lon: float):
    base_url = "https://api.open-meteo.com/v1/forecast/"
    temps = httpx.get(
        base_url,
        params=dict(latitude=lat, longitude=lon, hourly="temperature_2m"),
    )
    forecasted_temp = float(temps.json()["hourly"]["temperature_2m"][0])
    print(f"Forecasted temp C: {forecasted_temp} degrees")
    return forecasted_temp
```

Turn into a task



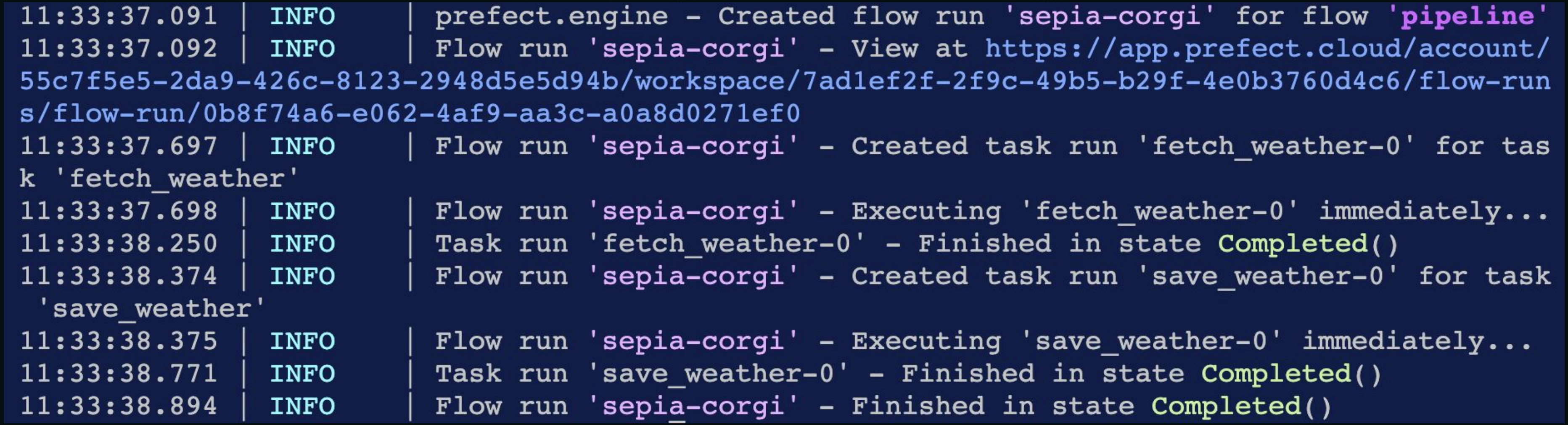
```python
@task
def save_weather(temp: float):
    with open("weather.csv", "w+") as w:
        w.write(str(temp))
    return "Successfully wrote temp"
```

Pipeline flow function



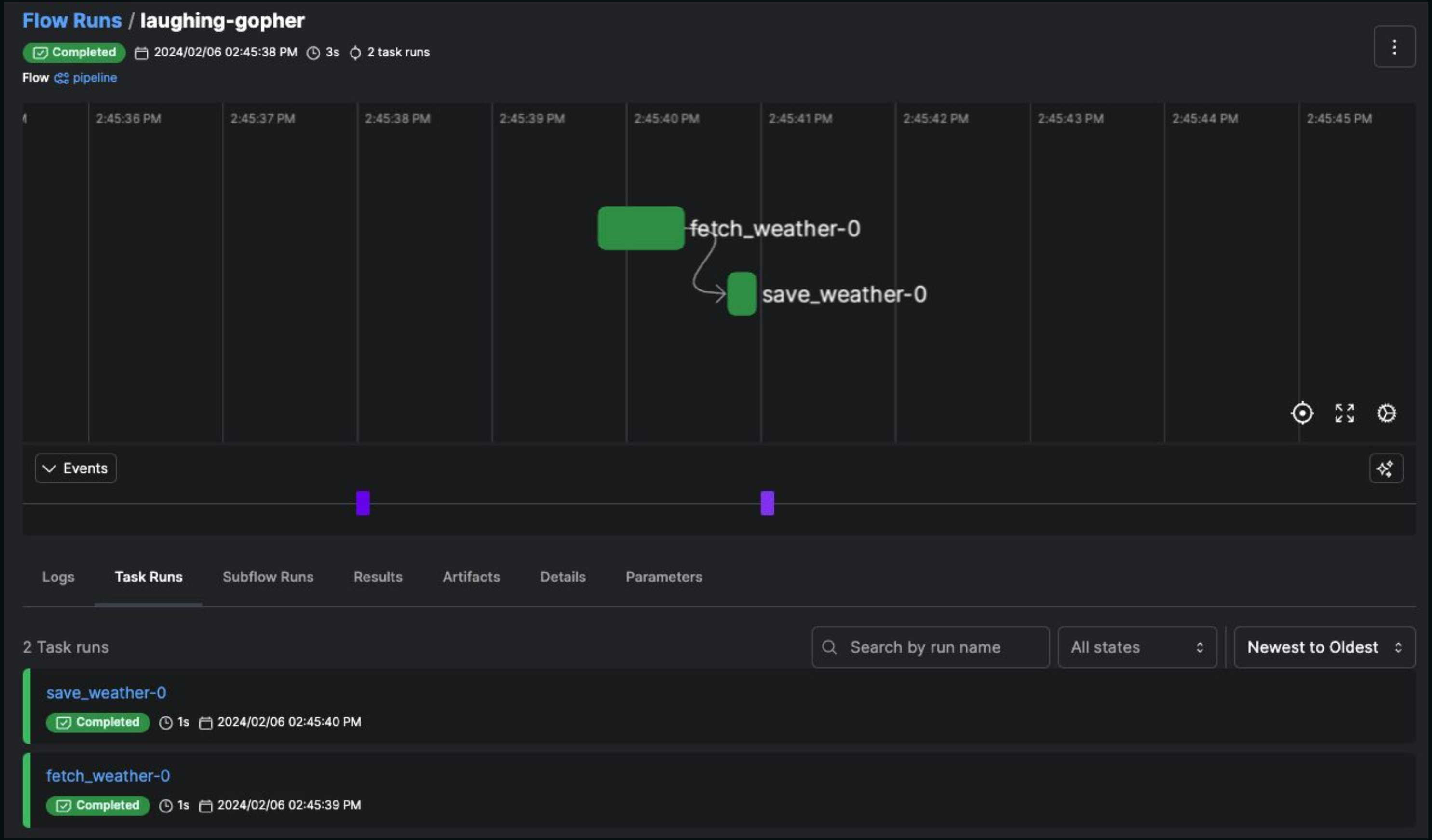
```python
@flow
def pipeline(lat: float = 38.9, lon: float = -77.0):
    temp = fetch_weather(lat, lon)
    result = save_weather(temp)
    return result
```

Logs from flow run



```
11:33:37.091 | INFO    | prefect.engine - Created flow run 'sepia-corgi' for flow 'pipeline'
11:33:37.092 | INFO    | Flow run 'sepia-corgi' - View at https://app.prefect.cloud/account/
55c7f5e5-2da9-426c-8123-2948d5e5d94b/workspace/7ad1ef2f-2f9c-49b5-b29f-4e0b3760d4c6/flow-run
s/flow-run/0b8f74a6-e062-4af9-aa3c-a0a8d0271ef0
11:33:37.697 | INFO    | Flow run 'sepia-corgi' - Created task run 'fetch_weather-0' for tas
k 'fetch_weather'
11:33:37.698 | INFO    | Flow run 'sepia-corgi' - Executing 'fetch_weather-0' immediately...
11:33:38.250 | INFO    | Task run 'fetch_weather-0' - Finished in state Completed()
11:33:38.374 | INFO    | Flow run 'sepia-corgi' - Created task run 'save_weather-0' for task
 'save_weather'
11:33:38.375 | INFO    | Flow run 'sepia-corgi' - Executing 'save_weather-0' immediately...
11:33:38.771 | INFO    | Task run 'save_weather-0' - Finished in state Completed()
11:33:38.894 | INFO    | Flow run 'sepia-corgi' - Finished in state Completed()
```

Visualize dependencies in the UI



Flow Runs / laughing-gopher

Completed 2024/02/06 02:45:38 PM 3s 2 task runs

Flow pipeline

2 Task runs

save_weather-0
Completed 1s 2024/02/06 02:45:40 PM

fetch_weather-0
Completed 1s 2024/02/06 02:45:39 PM

Tasks dos and don’ts

- ⛔ Don’t pass huge amounts of data between tasks
- ✅ Do keep tasks small
- 🙂 You can now use Prefect tasks as a replacement for Celery tasks

Note: Prefect is super Pythonic - conditionals are 👍

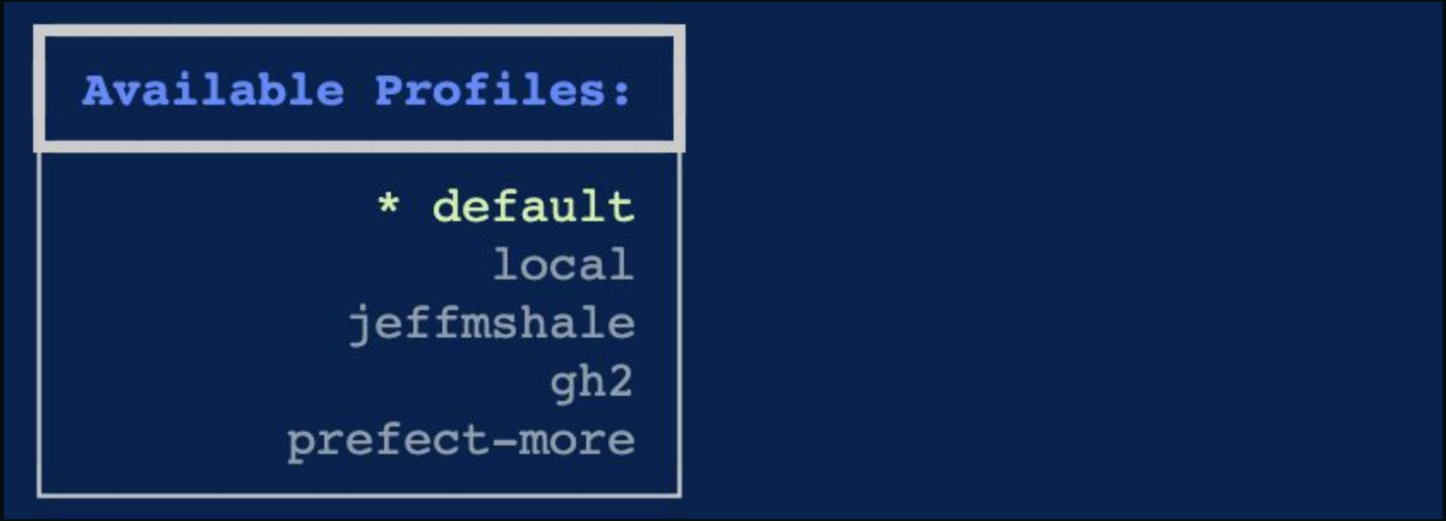
Prefect profiles

Prefect profiles

- Persistent settings for interacting with Prefect
- One profile active at all times
- Common to switch between:
  - Cloud and a self-hosted Prefect server
  - Cloud workspaces
  - Saved settings such as logging level

Prefect profiles

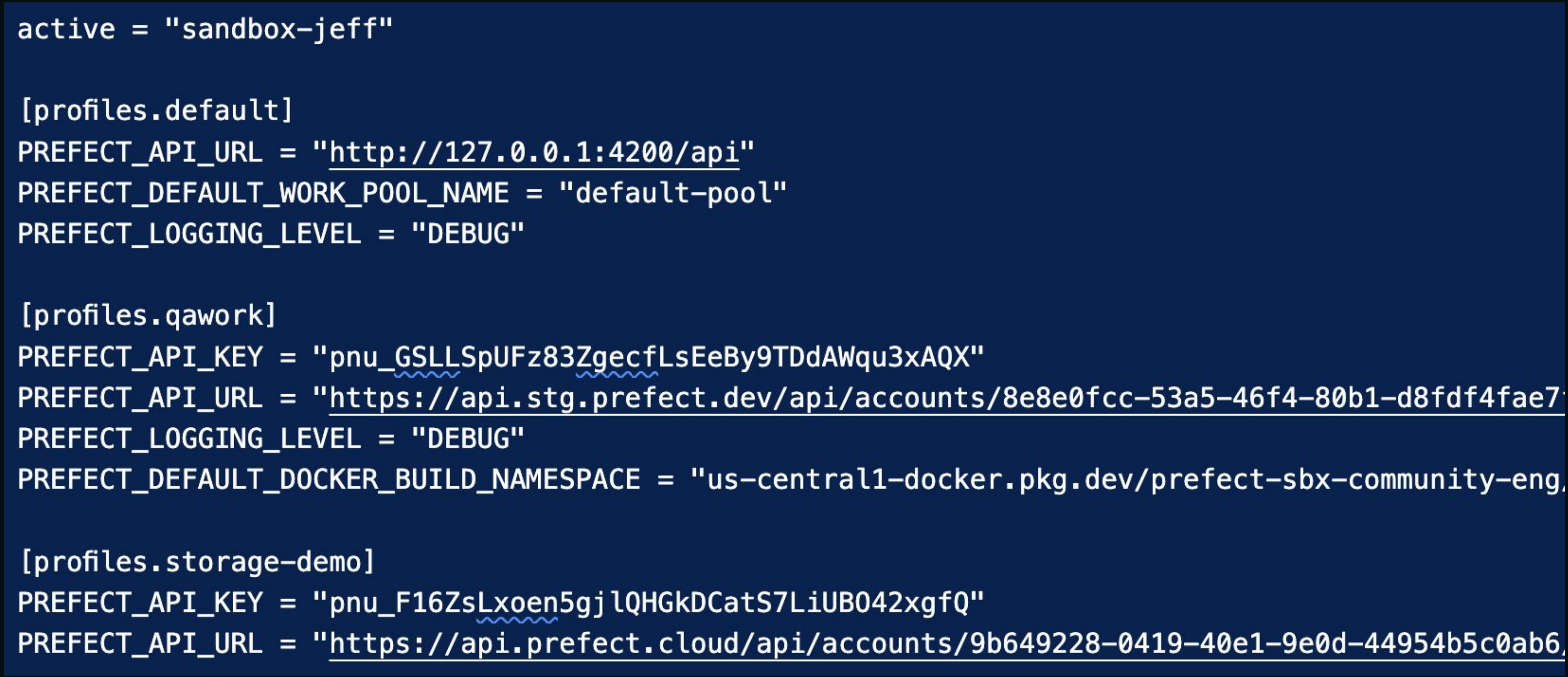
List: *prefect profile ls*



```
┌──────────────────────┐
│ Available Profiles:  │
├──────────────────────┤
│          * default   │
│              local   │
│        jeffmshale    │
│                gh2   │
│       prefect-more   │
└──────────────────────┘
```

Prefect profiles

Profiles live in *~/.prefect/profiles.toml* 📁



```
active = "sandbox-jeff"

[profiles.default]
PREFECT_API_URL = "http://127.0.0.1:4200/api"
PREFECT_DEFAULT_WORK_POOL_NAME = "default-pool"
PREFECT_LOGGING_LEVEL = "DEBUG"

[profiles.qawork]
PREFECT_API_KEY = "pnu_GSLLSpUFz83ZgecfLsEeBy9TDdAWqu3xAQX"
PREFECT_API_URL = "https://api.stg.prefect.dev/api/accounts/8e8e0fcc-53a5-46f4-80b1-d8fdf4fae7
PREFECT_LOGGING_LEVEL = "DEBUG"
PREFECT_DEFAULT_DOCKER_BUILD_NAMESPACE = "us-central1-docker.pkg.dev/prefect-sbx-community-eng

[profiles.storage-demo]
PREFECT_API_KEY = "pnu_F16ZsLxoen5gjlQHGkDCatS7LiUBO42xgfQ"
PREFECT_API_URL = "https://api.prefect.cloud/api/accounts/9b649228-0419-40e1-9e0d-44954b5c0ab6
```

Prefect profiles

Profile stays active until you switch to another profile

Contains:

1. Connection URL & API key for Prefect Cloud
2. Optional configuration

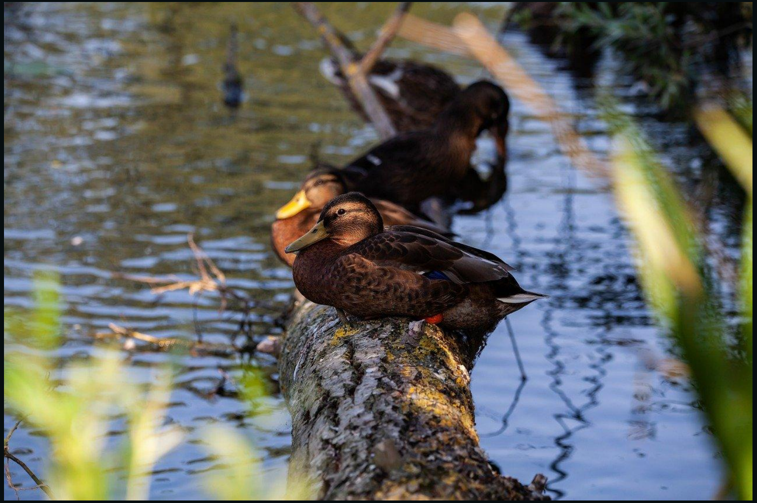
Prefect profiles

Create: *prefect profile create my_cloud_profile*

Inspect: *prefect profile inspect my_cloud_profile*

Switch: *prefect profile use my_cloud_profile*

Logging



Log *print* statements with *log_prints*

@flow(log_prints=True)

Want to log print statements by default?

Set environment variable

export PREFECT_LOGGING_LOG_PRINTS = True

(or set in your Prefect Profile)

Change logging level

Prefect default logging level: **INFO**

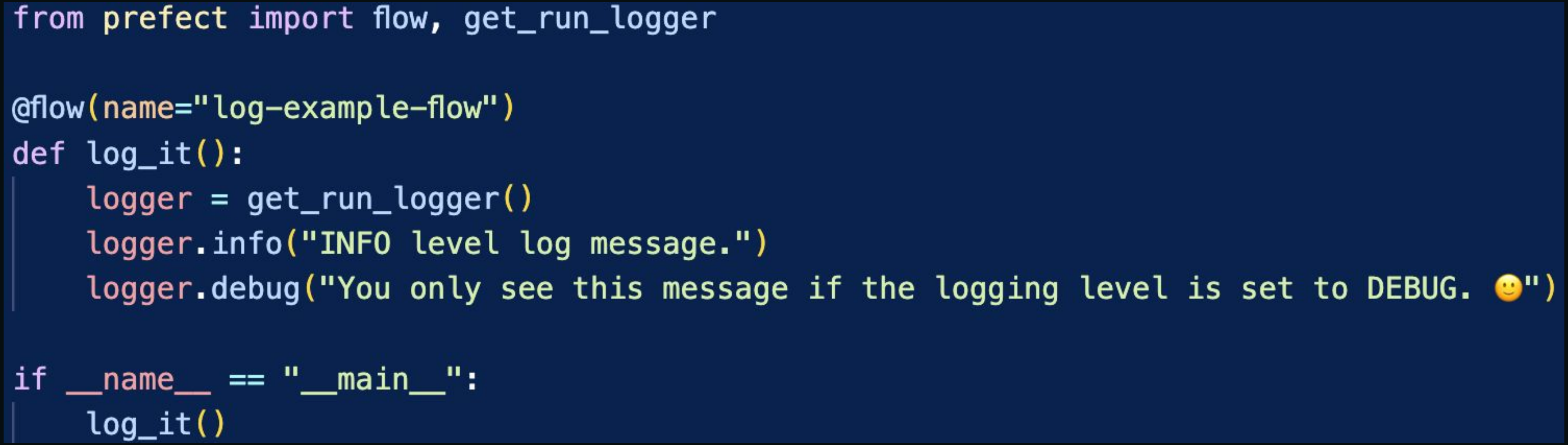
Change to **DEBUG**

Set environment variable:

```
export PREFECT_LOGGING_LEVEL="DEBUG"
```

Logging

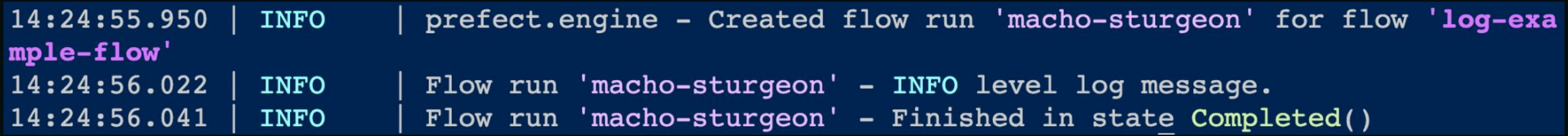
Create custom logs with *get_run_logger*



```python
from prefect import flow, get_run_logger

@flow(name="log-example-flow")
def log_it():
    logger = get_run_logger()
    logger.info("INFO level log message.")
    logger.debug("You only see this message if the logging level is set to DEBUG. 🙂")

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

Logging

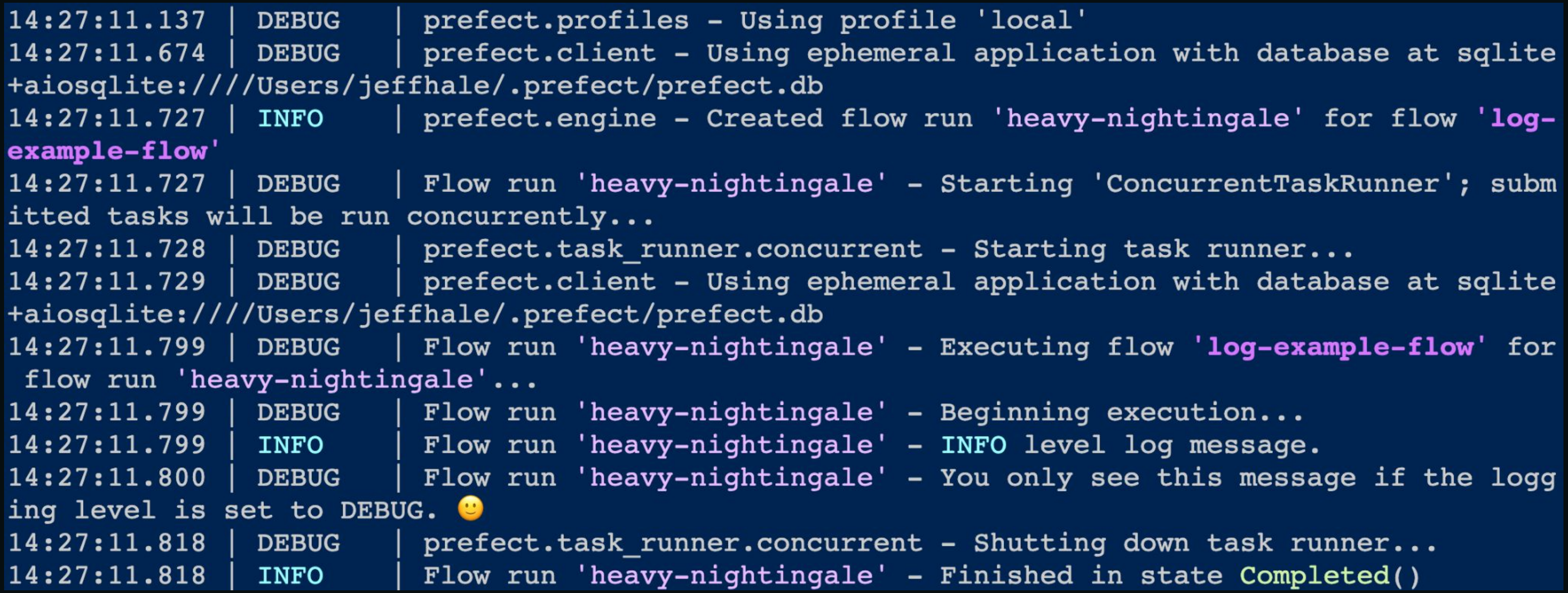
Output with **INFO** logging level set:



```
14:24:55.950 | INFO     | prefect.engine - Created flow run 'macho-sturgeon' for flow 'log-exa
mple-flow'
14:24:56.022 | INFO     | Flow run 'macho-sturgeon' - INFO level log message.
14:24:56.041 | INFO     | Flow run 'macho-sturgeon' - Finished in state Completed()
```

Logging

Output with **DEBUG** logging level set:



```
14:27:11.137 | DEBUG    | prefect.profiles - Using profile 'local'
14:27:11.674 | DEBUG    | prefect.client - Using ephemeral application with database at sqlite
+aiosqlite:////Users/jeffhale/.prefect/prefect.db
14:27:11.727 | INFO     | prefect.engine - Created flow run 'heavy-nightingale' for flow 'log-
example-flow'
14:27:11.727 | DEBUG    | Flow run 'heavy-nightingale' - Starting 'ConcurrentTaskRunner'; subm
itted tasks will be run concurrently...
14:27:11.728 | DEBUG    | prefect.task_runner.concurrent - Starting task runner...
14:27:11.729 | DEBUG    | prefect.client - Using ephemeral application with database at sqlite
+aiosqlite:////Users/jeffhale/.prefect/prefect.db
14:27:11.799 | DEBUG    | Flow run 'heavy-nightingale' - Executing flow 'log-example-flow' for
 flow run 'heavy-nightingale'...
14:27:11.799 | DEBUG    | Flow run 'heavy-nightingale' - Beginning execution...
14:27:11.799 | INFO     | Flow run 'heavy-nightingale' - INFO level log message.
14:27:11.800 | DEBUG    | Flow run 'heavy-nightingale' - You only see this message if the logg
ing level is set to DEBUG. 🙂
14:27:11.818 | DEBUG    | prefect.task_runner.concurrent - Shutting down task runner...
14:27:11.818 | INFO     | Flow run 'heavy-nightingale' - Finished in state Completed()
```

prefect.runtime

prefect.runtime

Module for runtime context access.

Useful for labeling, logs, etc.

Includes:

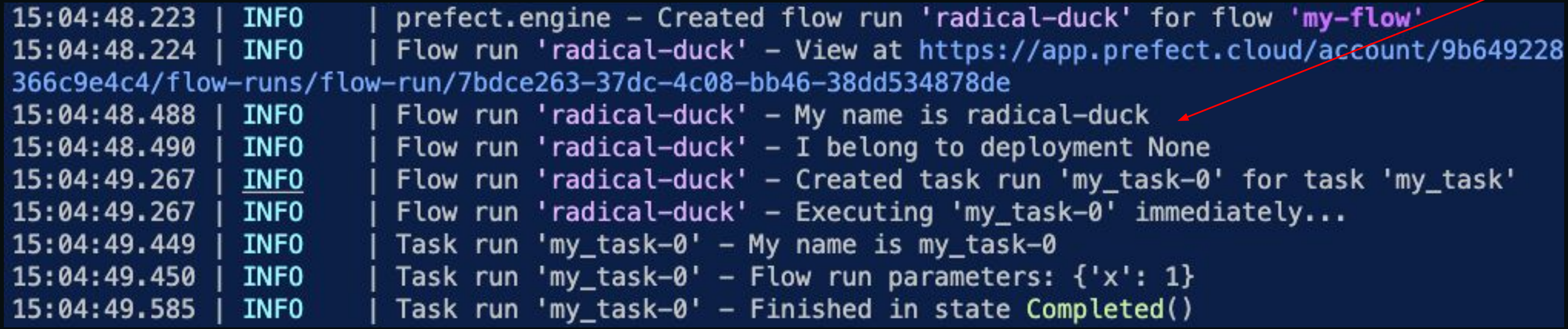
- ***deployment:*** info about current deployment
- ***flow_run***: info about current flow run
- ***task_run***: info about current task run

prefect.runtime

```python
from prefect import flow, task
from prefect import runtime


@flow(log_prints=True)
def my_flow(x):
    print("My name is", runtime.flow_run.name)
    print("I belong to deployment", runtime.deployment.name)
    my_task(2)


@task
def my_task(y):
    print("My name is", runtime.task_run.name)
    print("Flow run parameters:", runtime.flow_run.parameters)
```

prefect.runtime

Useful for labeling, logs, etc.



```
15:04:48.223 | INFO    | prefect.engine - Created flow run 'radical-duck' for flow 'my-flow'
15:04:48.224 | INFO    | Flow run 'radical-duck' - View at https://app.prefect.cloud/account/9b649228
366c9e4c4/flow-runs/flow-run/7bdce263-37dc-4c08-bb46-38dd534878de
15:04:48.488 | INFO    | Flow run 'radical-duck' - My name is radical-duck
15:04:48.490 | INFO    | Flow run 'radical-duck' - I belong to deployment None
15:04:49.267 | INFO    | Flow run 'radical-duck' - Created task run 'my_task-0' for task 'my_task'
15:04:49.267 | INFO    | Flow run 'radical-duck' - Executing 'my_task-0' immediately...
15:04:49.449 | INFO    | Task run 'my_task-0' - My name is my_task-0
15:04:49.450 | INFO    | Task run 'my_task-0' - Flow run parameters: {'x': 1}
15:04:49.585 | INFO    | Task run 'my_task-0' - Finished in state Completed()
```

Retries

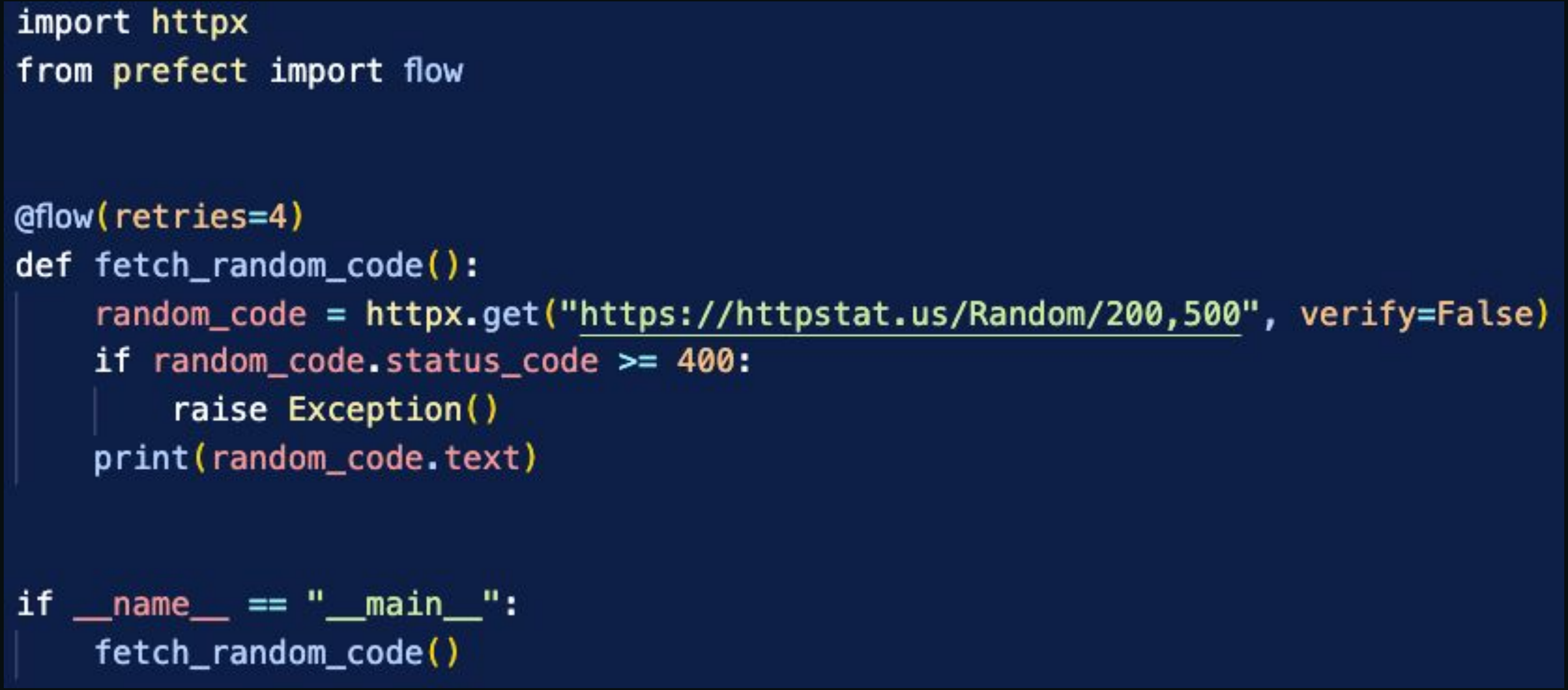
Retries - guard against failure

Specify in task or a flow decorator

@task(retries=2)

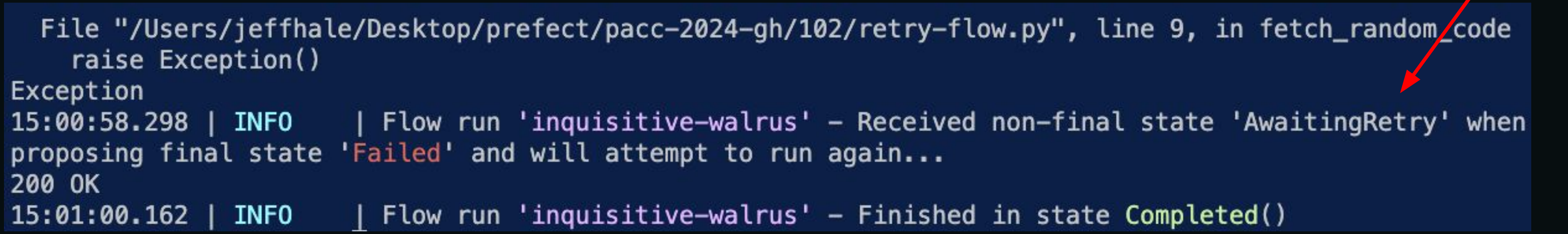
@flow(retries=3)

Flow retries



```python
import httpx
from prefect import flow


@flow(retries=4)
def fetch_random_code():
    random_code = httpx.get("https://httpstat.us/Random/200,500", verify=False)
    if random_code.status_code >= 400:
        raise Exception()
    print(random_code.text)


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

Automatic retry



```
  File "/Users/jeffhale/Desktop/prefect/pacc-2024-gh/102/retry-flow.py", line 9, in fetch_random_code
    raise Exception()
Exception
15:00:58.298 | INFO    | Flow run 'inquisitive-walrus' - Received non-final state 'AwaitingRetry' when
proposing final state 'Failed' and will attempt to run again...
200 OK
15:01:00.162 | INFO    | Flow run 'inquisitive-walrus' - Finished in state Completed()
```

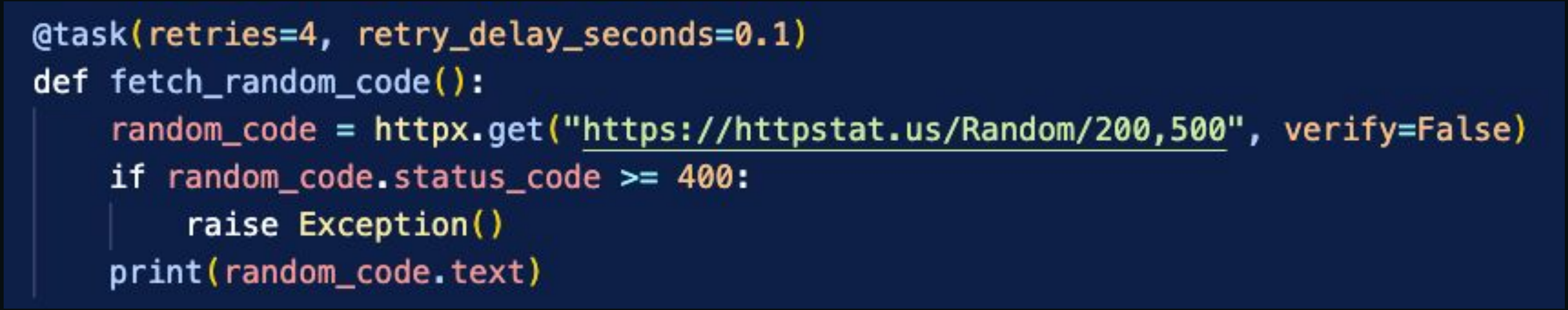
Automatic retry with delay

Automatic retry with delay

Specify in task or flow decorator

```
@task(retries=2, retry_delay_seconds=0.1)
```

Task retries with delay



```python
@task(retries=4, retry_delay_seconds=0.1)
def fetch_random_code():
    random_code = httpx.get("https://httpstat.us/Random/200,500", verify=False)
    if random_code.status_code >= 400:
        raise Exception()
    print(random_code.text)
```

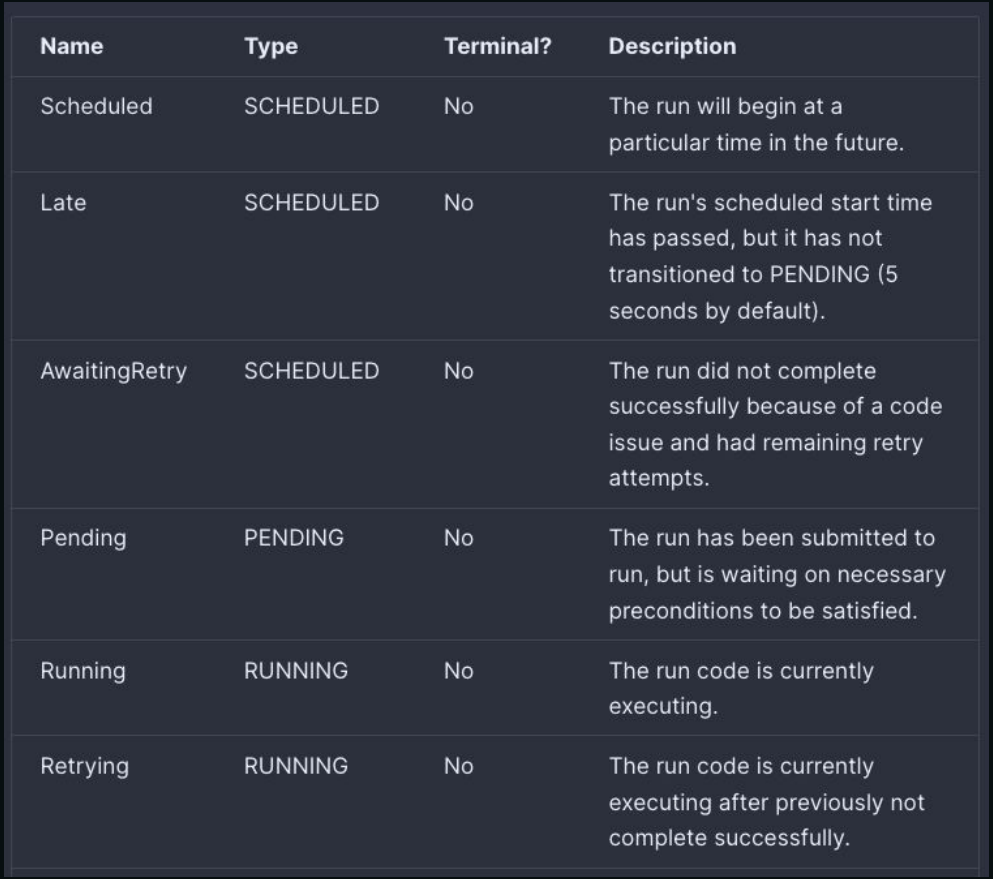
👆You can pass a list of values or an *exponential_backoff* to *retry_delay_seconds* for tasks.

States

Prefect flow run states

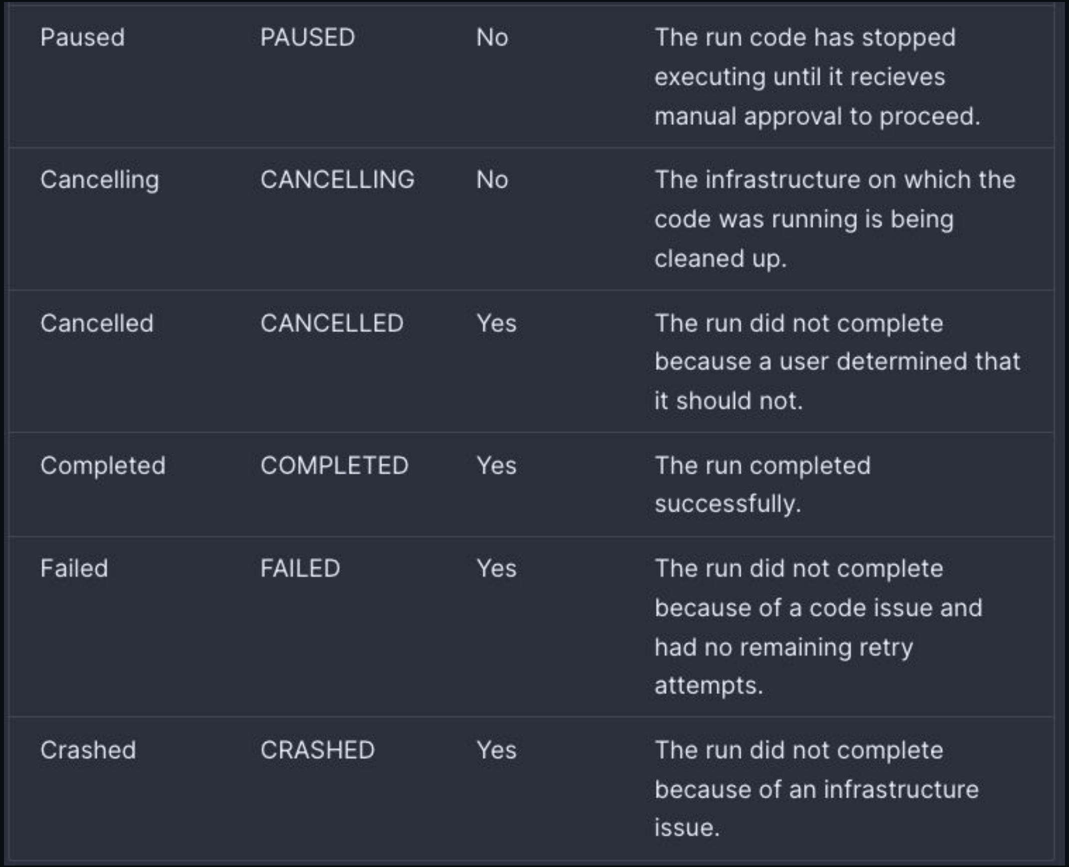
What’s the state of your workflows?

Prefect flow run states



Name	Type	Terminal?	Description
Scheduled	SCHEDULED	No	The run will begin at a particular time in the future.
Late	SCHEDULED	No	The run's scheduled start time has passed, but it has not transitioned to PENDING (5 seconds by default).
AwaitingRetry	SCHEDULED	No	The run did not complete successfully because of a code issue and had remaining retry attempts.
Pending	PENDING	No	The run has been submitted to run, but is waiting on necessary preconditions to be satisfied.
Running	RUNNING	No	The run code is currently executing.
Retrying	RUNNING	No	The run code is currently executing after previously not complete successfully.

Prefect flow run states



Paused	PAUSED	No	The run code has stopped executing until it recieves manual approval to proceed.
Cancelling	CANCELLING	No	The infrastructure on which the code was running is being cleaned up.
Cancelled	CANCELLED	Yes	The run did not complete because a user determined that it should not.
Completed	COMPLETED	Yes	The run completed successfully.
Failed	FAILED	Yes	The run did not complete because of a code issue and had no remaining retry attempts.
Crashed	CRASHED	Yes	The run did not complete because of an infrastructure issue.

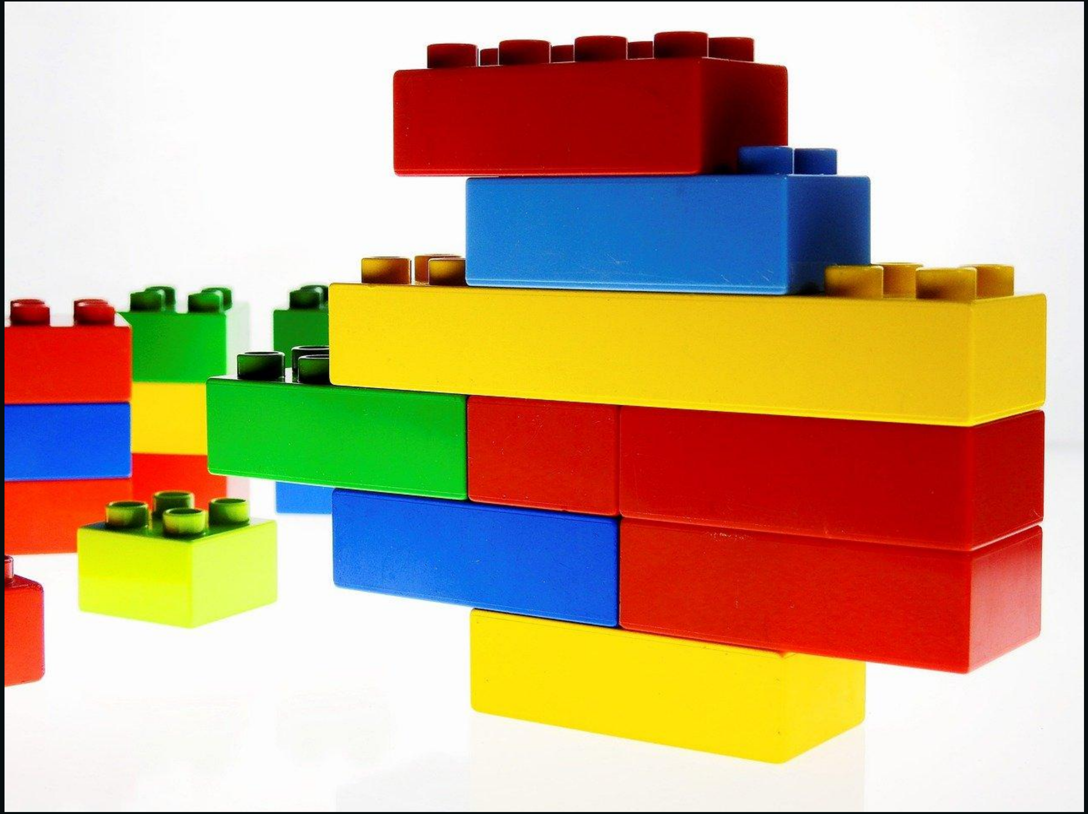
Blocks 🧱

Blocks

Configuration

+

Code

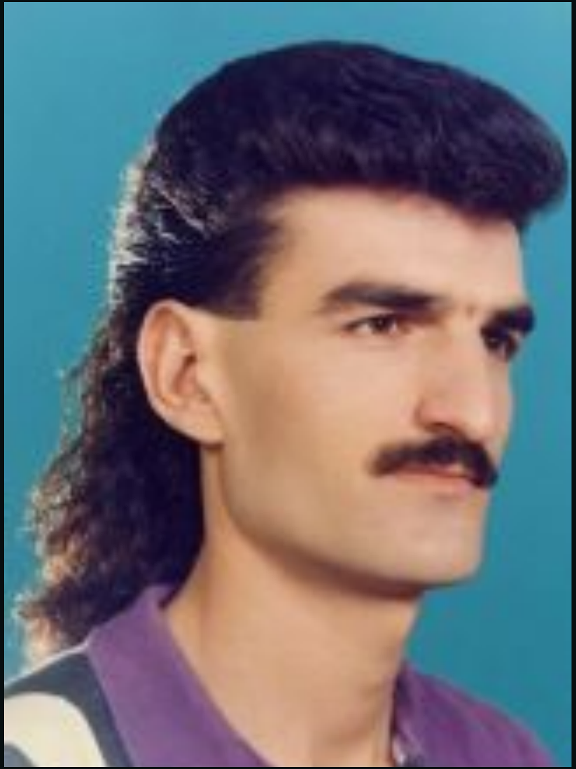


Blocks

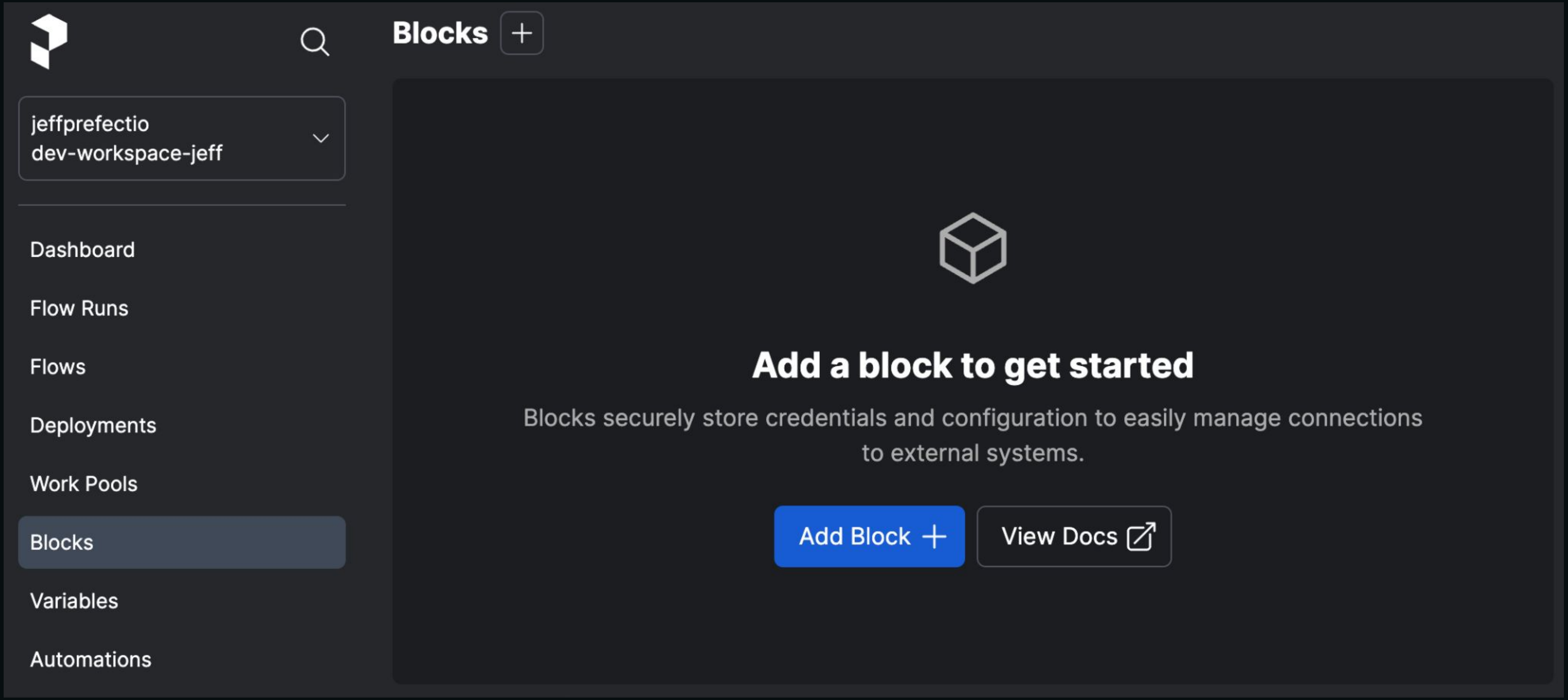
The Block mullet:

Structured form in front,

flexible code in back



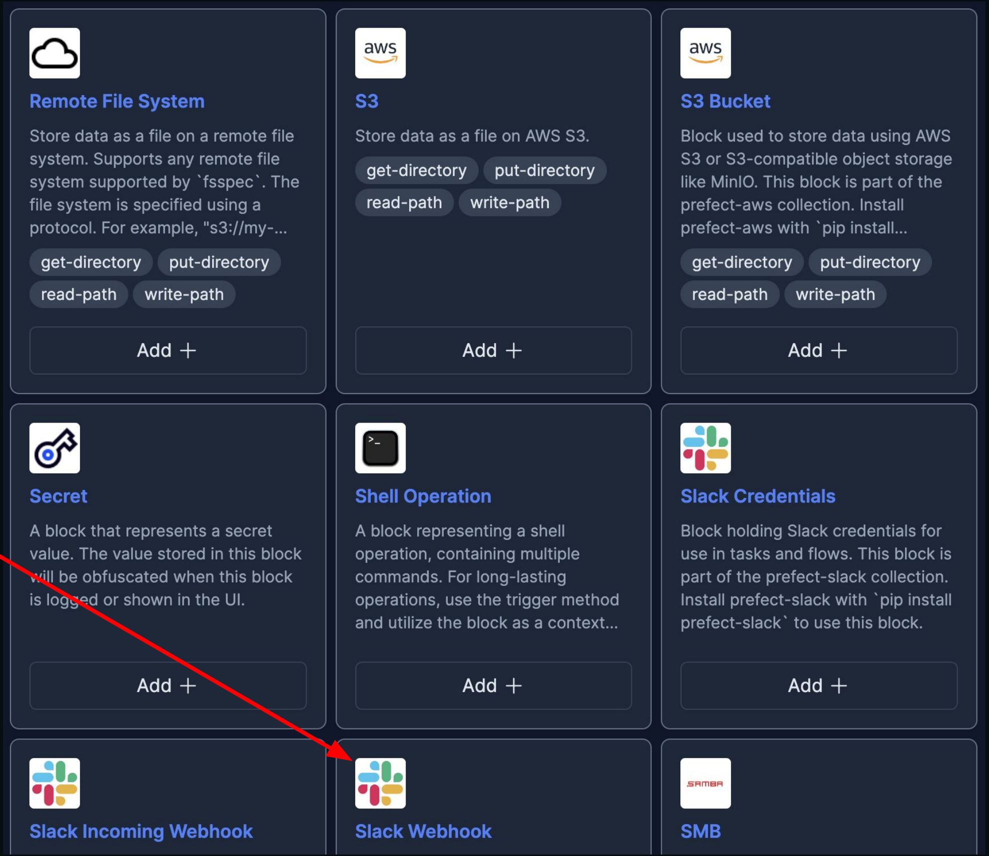
Create a Block from the UI



Add a block to get started

Blocks securely store credentials and configuration to easily manage connections to external systems.

Create a block from the UI - choose a block type



Remote File System

Store data as a file on a remote file system. Supports any remote file system supported by `fsspec`. The file system is specified using a protocol. For example, "s3://my-...

get-directory put-directory read-path write-path

Add +

S3

Store data as a file on AWS S3.

get-directory put-directory read-path write-path

Add +

S3 Bucket

Block used to store data using AWS S3 or S3-compatible object storage like MinIO. This block is part of the prefect-aws collection. Install prefect-aws with `pip install...

get-directory put-directory read-path write-path

Add +

Secret

A block that represents a secret value. The value stored in this block will be obfuscated when this block is logged or shown in the UI.

Add +

Shell Operation

A block representing a shell operation, containing multiple commands. For long-lasting operations, use the trigger method and utilize the block as a context...

Add +

Slack Credentials

Block holding Slack credentials for use in tasks and flows. This block is part of the prefect-slack collection. Install prefect-slack with `pip install prefect-slack` to use this block.

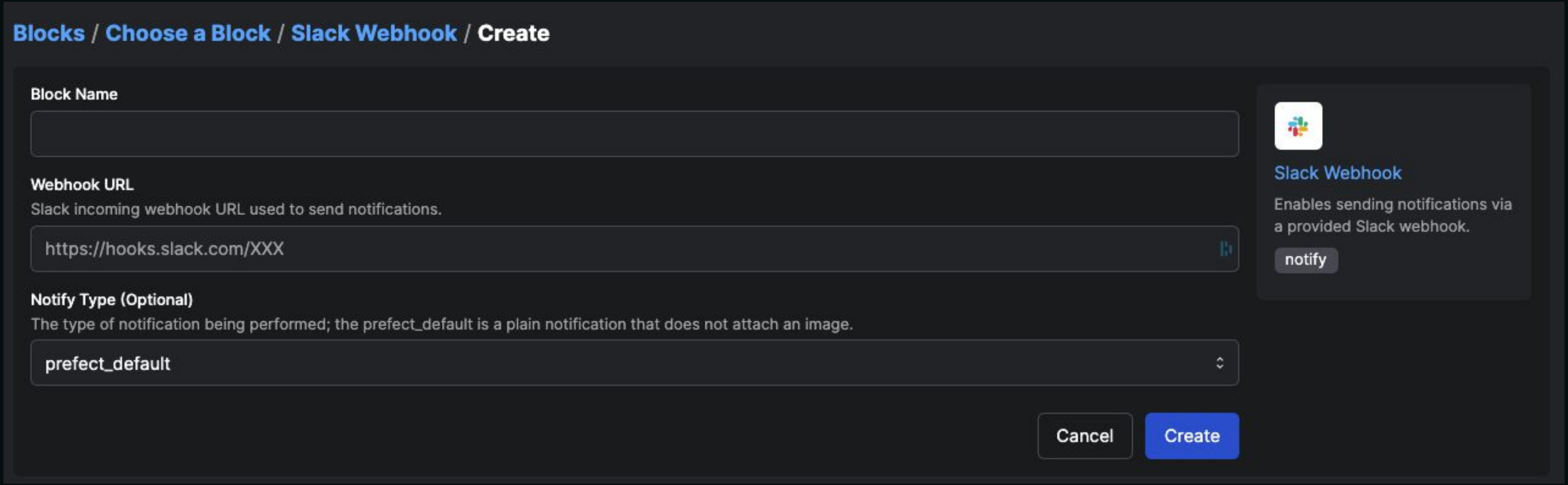
Add +

Slack Incoming Webhook

Slack Webhook

SMB

Create a block from the UI



Blocks / Choose a Block / Slack Webhook / Create

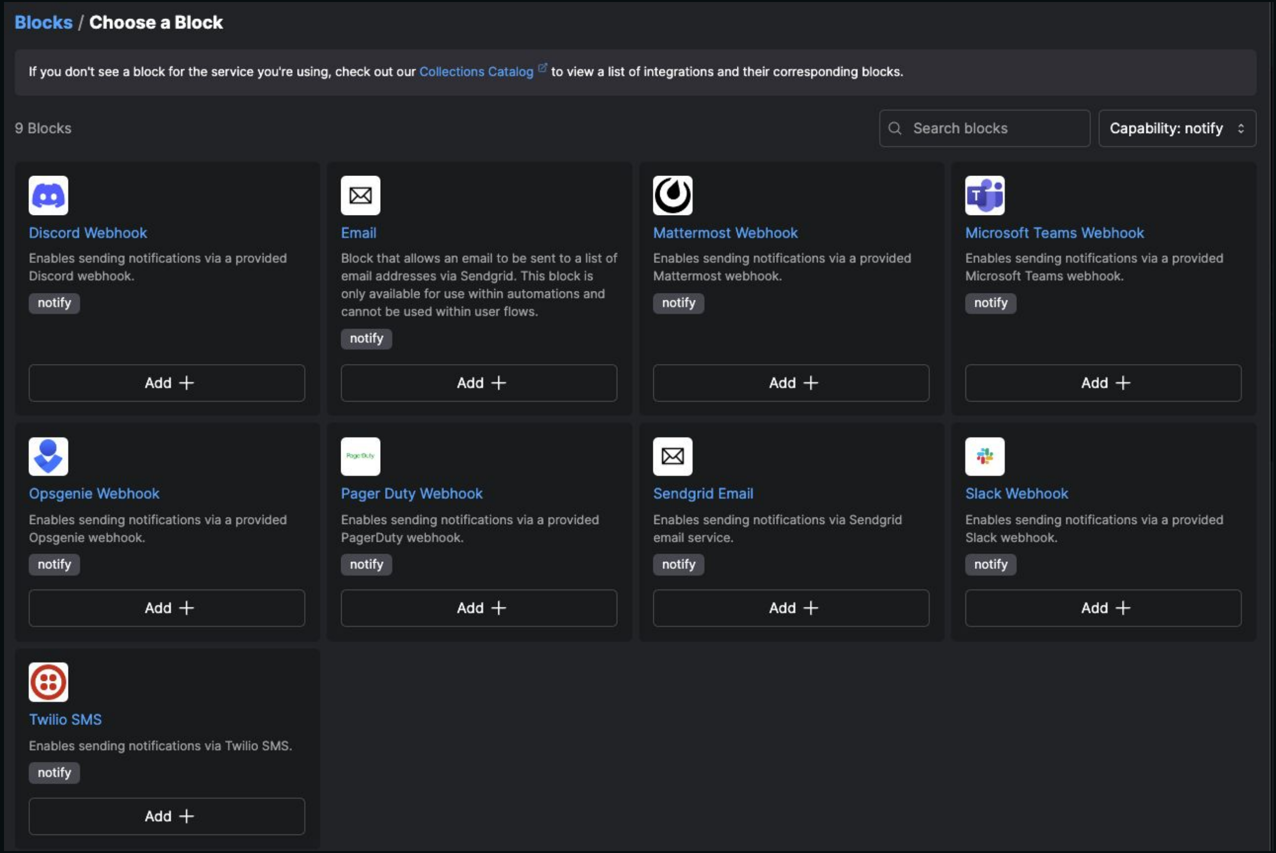
Block Name

Webhook URL
Slack incoming webhook URL used to send notifications.
https://hooks.slack.com/XXX

Notify Type (Optional)
The type of notification being performed; the prefect_default is a plain notification that does not attach an image.
prefect_default

Slack Webhook
Enables sending notifications via a provided Slack webhook.
notify

Block types in UI - filter by capability


Blocks / Choose a Block

If you don't see a block for the service you're using, check out our Collections Catalog to view a list of integrations and their corresponding blocks.

9 Blocks

Search blocks

Capability: notify

Discord Webhook
Enables sending notifications via a provided Discord webhook.
notify
Add +

Email
Block that allows an email to be sent to a list of email addresses via Sendgrid. This block is only available for use within automations and cannot be used within user flows.
notify
Add +

Mattermost Webhook
Enables sending notifications via a provided Mattermost webhook.
notify
Add +

Microsoft Teams Webhook
Enables sending notifications via a provided Microsoft Teams webhook.
notify
Add +

Opsgenie Webhook
Enables sending notifications via a provided Opsgenie webhook.
notify
Add +

Pager Duty Webhook
Enables sending notifications via a provided PagerDuty webhook.
notify
Add +

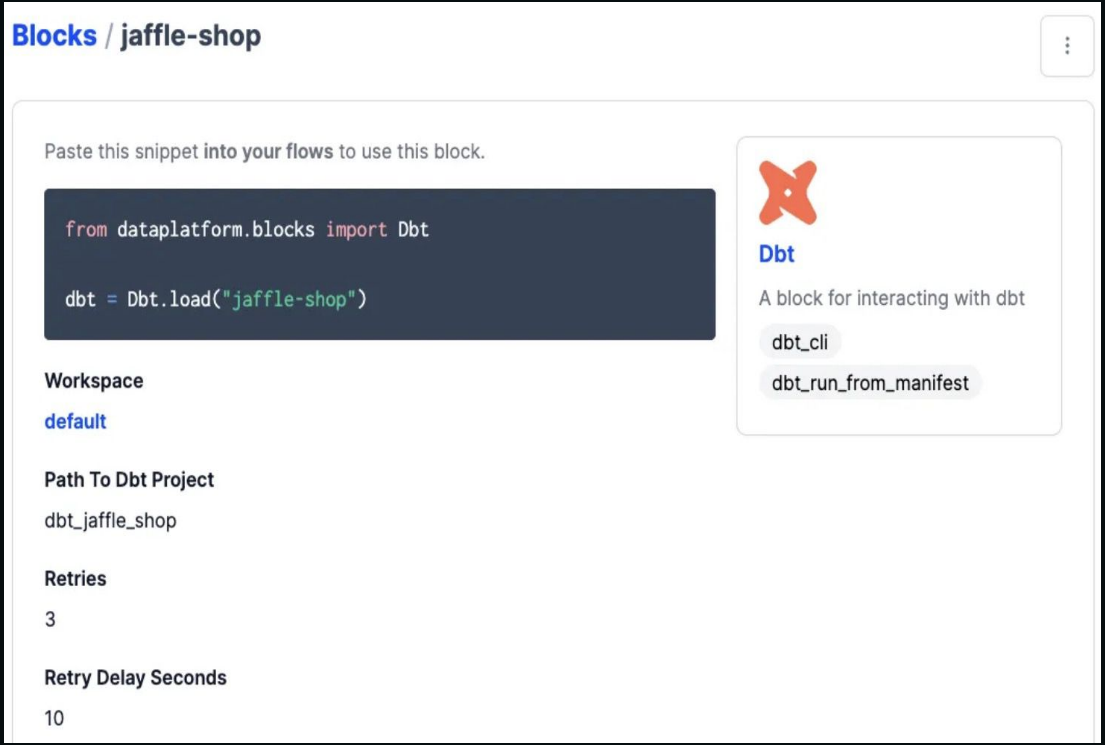
Sendgrid Email
Enables sending notifications via Sendgrid email service.
notify
Add +

Slack Webhook
Enables sending notifications via a provided Slack webhook.
notify
Add +

Twilio SMS
Enables sending notifications via Twilio SMS.
notify
Add +

Under the hood, block types are Python classes

Blocks are instances of those Python classes (TODO add python class)


Blocks / jaffle-shop

Paste this snippet **into your flows** to use this block.

```
from dataplatform.blocks import Dbt

dbt = Dbt.load("jaffle-shop")
```

Workspace

default

Path To Dbt Project

dbt_jaffle_shop

Retries

3

Retry Delay Seconds

10

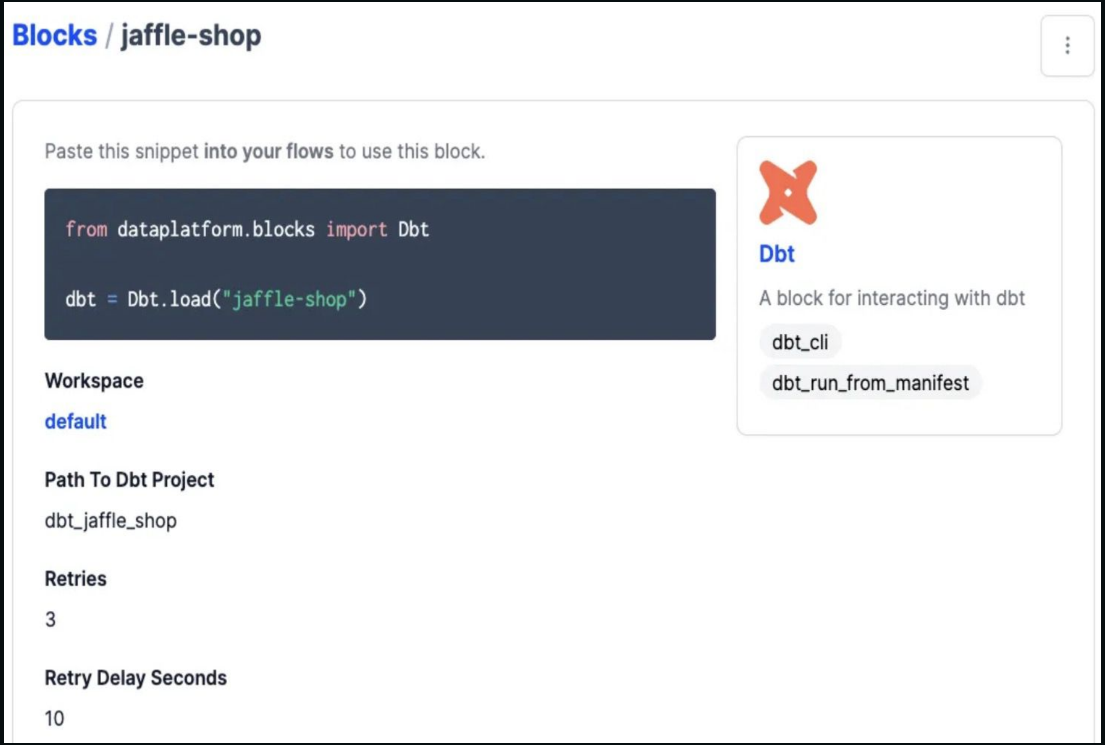
Dbt

A block for interacting with dbt

dbt_cli

dbt_run_from_manifest

Blocks are instances of those Python classes



Blocks / jaffle-shop

Paste this snippet **into your flows** to use this block.

```
from dataplatform.blocks import Dbt

dbt = Dbt.load("jaffle-shop")
```

Workspace

default

Path To Dbt Project

dbt_jaffle_shop

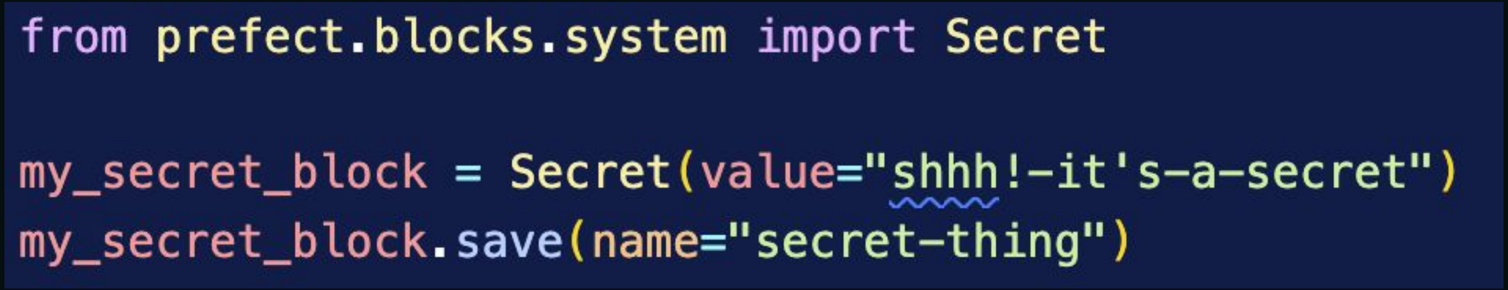
Retries

3

Retry Delay Seconds

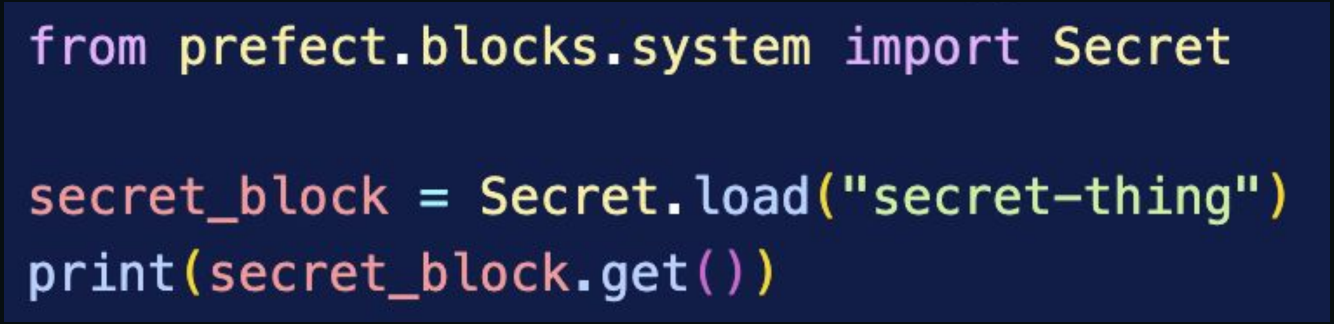
10

Create a block in Python



```python
from prefect.blocks.system import Secret

my_secret_block = Secret(value="shhh!-it's-a-secret")
my_secret_block.save(name="secret-thing")
```

Retrieve and use a block in Python



```python
from prefect.blocks.system import Secret

secret_block = Secret.load("secret-thing")
print(secret_block.get())
```

Blocks

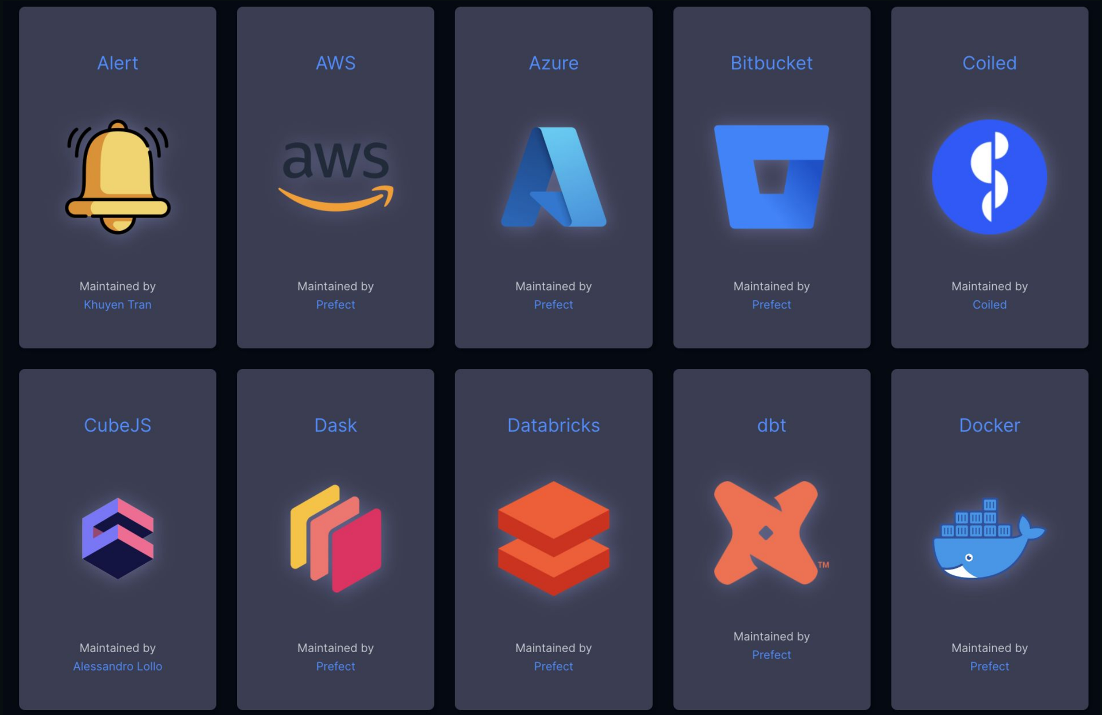
Reusable, modular, configuration + code

- Nestable
- Stored in database
- Can create own types

Integrations

Integrations

docs.prefect.io/integrations/catalog/



Alert
Maintained by
Khuyen Tran

AWS
Maintained by
Prefect

Azure
Maintained by
Prefect

Bitbucket
Maintained by
Prefect

Coiled
Maintained by
Coiled

CubeJS
Maintained by
Alessandro Lollo

Dask
Maintained by
Prefect

Databricks
Maintained by
Prefect

dbt
Maintained by
Prefect

Docker
Maintained by
Prefect

Integrations

Python packages that add convenience

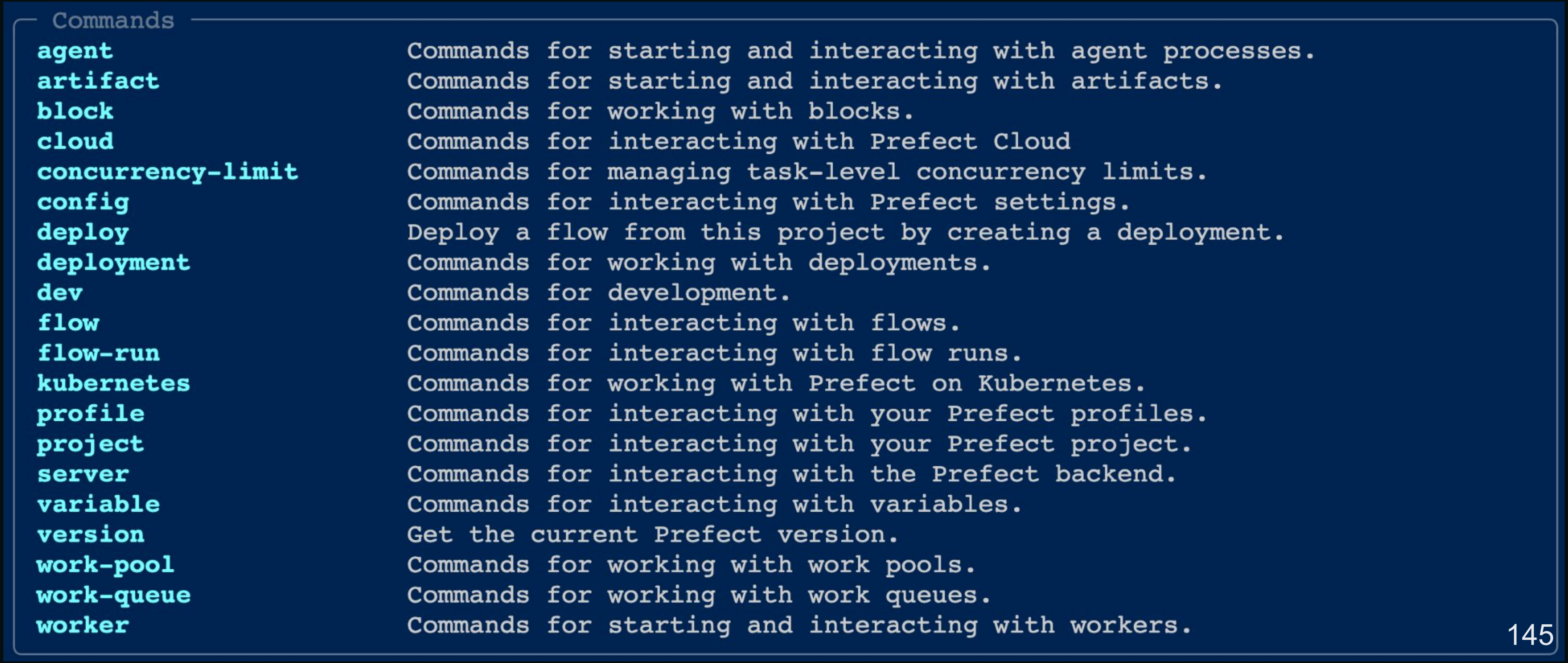
- Template to create your own
- Can contribute to the community
- May contain new block types you’ll register

More helpful resources

Prefect CLI

Start commands with *prefect* *--help* is always available

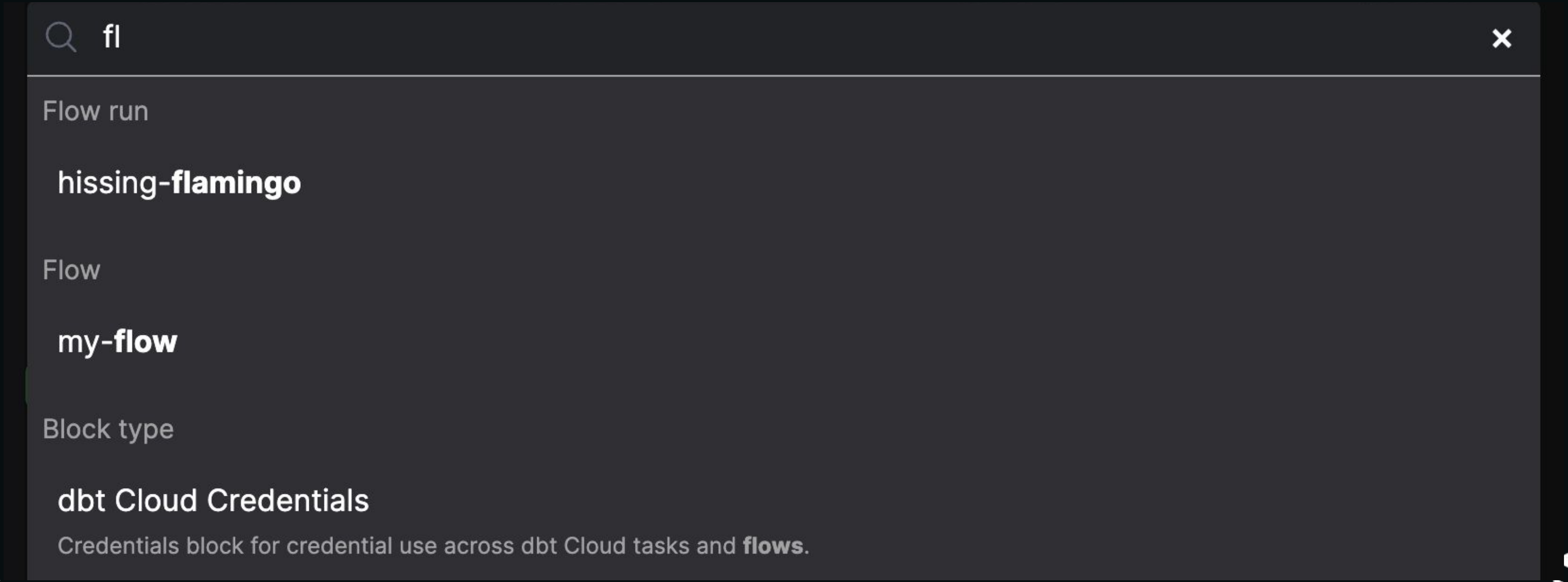
prefect --help



```
╭─ Commands ─────────────────────────────────────────────────────────────────────╮
│ agent                  Commands for starting and interacting with agent processes. │
│ artifact               Commands for starting and interacting with artifacts.       │
│ block                  Commands for working with blocks.                           │
│ cloud                  Commands for interacting with Prefect Cloud                 │
│ concurrency-limit      Commands for managing task-level concurrency limits.        │
│ config                 Commands for interacting with Prefect settings.             │
│ deploy                 Deploy a flow from this project by creating a deployment.   │
│ deployment             Commands for working with deployments.                      │
│ dev                    Commands for development.                                   │
│ flow                   Commands for interacting with flows.                        │
│ flow-run               Commands for interacting with flow runs.                    │
│ kubernetes             Commands for working with Prefect on Kubernetes.            │
│ profile                Commands for interacting with your Prefect profiles.        │
│ project                Commands for interacting with your Prefect project.         │
│ server                 Commands for interacting with the Prefect backend.          │
│ variable               Commands for interacting with variables.                    │
│ version                Get the current Prefect version.                            │
│ work-pool              Commands for working with work pools.                       │
│ work-queue             Commands for working with work queues.                      │
│ worker                 Commands for starting and interacting with workers.         │
╰────────────────────────────────────────────────────────────────────────────────╯
```

Search in the UI

cmd + **k** or 🔍


fl

Flow run

hissing-**flamingo**

Flow

my-**flow**

Block type

dbt Cloud Credentials

Credentials block for credential use across dbt Cloud tasks and **flows**.

102 Recap

You've seen how to understand the state of your workflows and guard against failure.

- Tasks
- Profiles
- Logging
- Retries
- States
- Blocks
- Integrations
- More resources: *help* & search

Lab 102

Lab 102

- Use a flow with two tasks that fetches weather data from open-meteo
- Pass data between the tasks
- Add retries (add an exception to force a failure)
- Run your flow as a Python script
- Stretch 1: Log the name of the flow run
- Stretch 2: Create a block in the UI
- Stretch 3: Load the block in code and use it

103 - Work with data and create automatic alerts

PREFECT

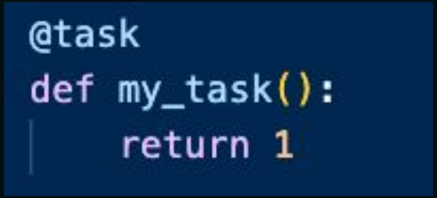
103 Agenda

- Work with data to save time and money
    - Save results
    - Use caching
    - Create Markdown artifacts to communicate insights
- Learn about user management in Prefect Cloud
- Set up automatic notifications for workflow states

Results

Results

The data returned by a flow or a task

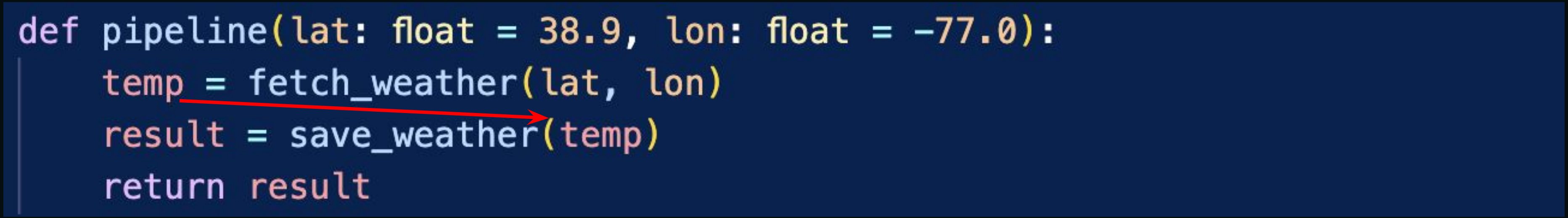


```
@task
def my_task():
    return 1
```

1 is the result

Passing results

Pass results from one task to another so Prefect can discover dependency relationships at runtime

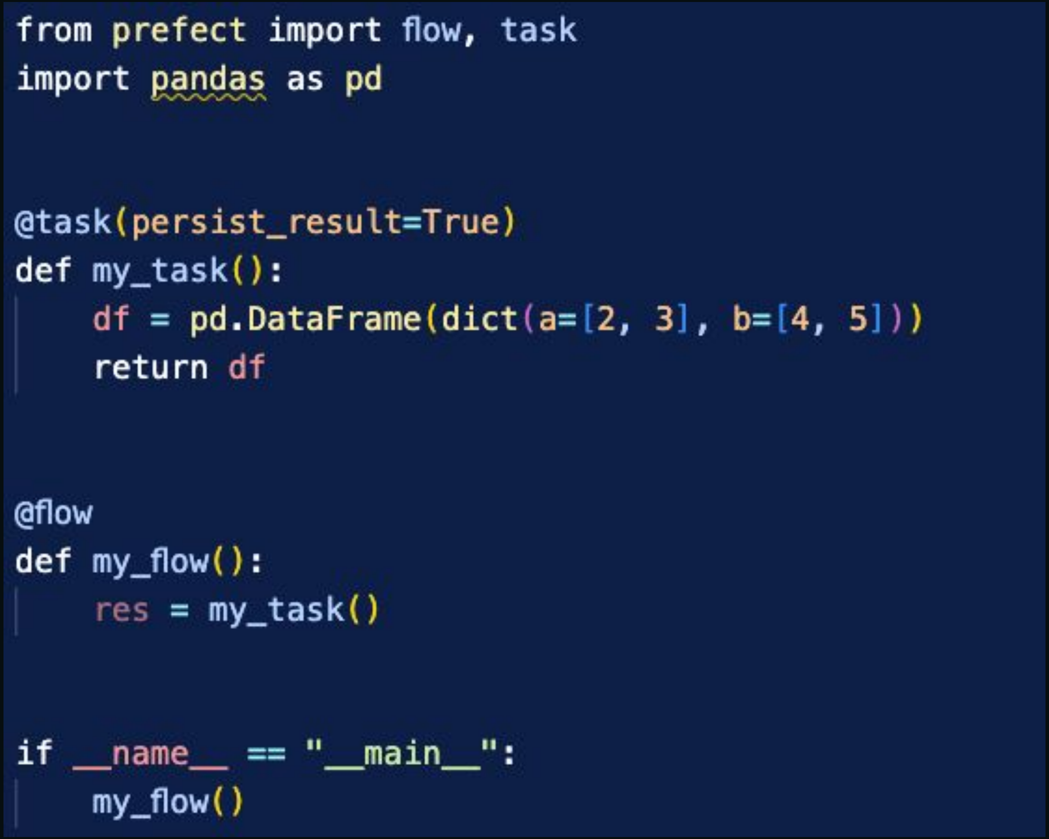


```
def pipeline(lat: float = 38.9, lon: float = -77.0):
    temp = fetch_weather(lat, lon)
    result = save_weather(temp)
    return result
```

Results

👆By default, Prefect returns a result that is ***not*** persisted to disk. It is only stored in memory.

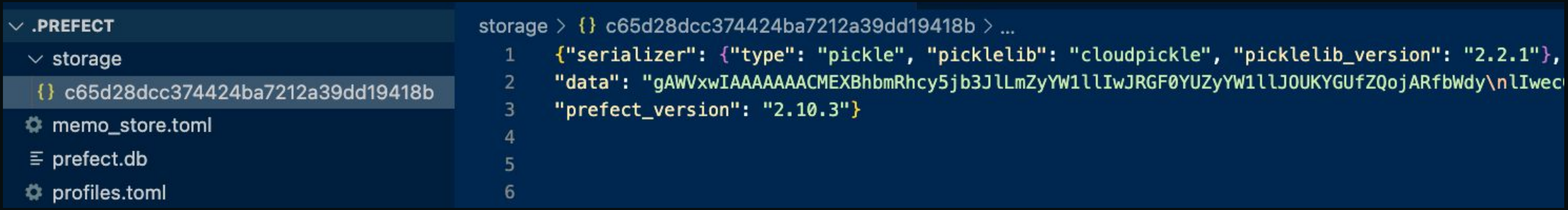
Persist results with *persist_result=True*



```python
from prefect import flow, task
import pandas as pd


@task(persist_result=True)
def my_task():
    df = pd.DataFrame(dict(a=[2, 3], b=[4, 5]))
    return df


@flow
def my_flow():
    res = my_task()


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

Persisted results

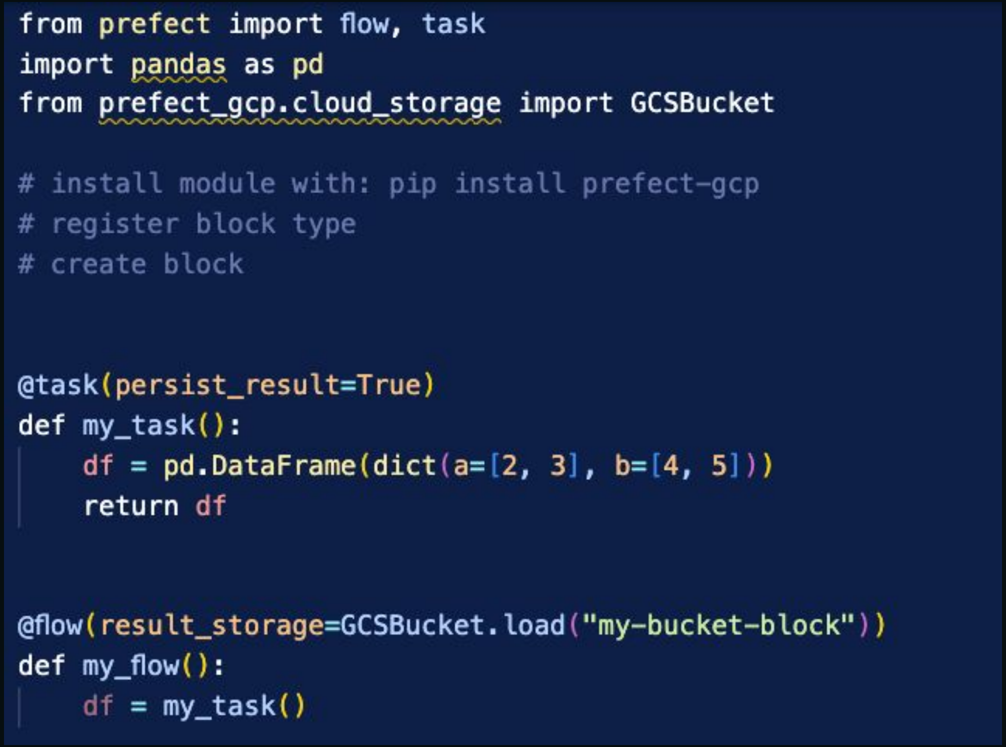
- Stored in **.PREFECT/*storage*** folder by default
- Pickled by default 🥒
- You can use other serializer or compress


.PREFECT
storage
{} c65d28dcc374424ba7212a39dd19418b
memo_store.toml
prefect.db
profiles.toml
storage > {} c65d28dcc374424ba7212a39dd19418b > ...

```
{"serializer": {"type": "pickle", "picklelib": "cloudpickle", "picklelib_version": "2.2.1"},
"data": "gAWVxwIAAAAAAACMEXBhbmRhcy5jb3JlLmZyYW1llIwJRGF0YUZyYW1llJOUKYGUfZQojARfbWdy\nlIwec
"prefect_version": "2.10.3"}
```

Results - remote data storage

Store results in cloud provider storage - use a block



```python
from prefect import flow, task
import pandas as pd
from prefect_gcp.cloud_storage import GCSBucket

# install module with: pip install prefect-gcp
# register block type
# create block


@task(persist_result=True)
def my_task():
    df = pd.DataFrame(dict(a=[2, 3], b=[4, 5]))
    return df


@flow(result_storage=GCSBucket.load("my-bucket-block"))
def my_flow():
    df = my_task()
```

Working with big data

Read and write data to cloud provider without passing the data around.

See discussion of options:

docs.prefect.io/guides/big-data/

Caching

Caching

What?

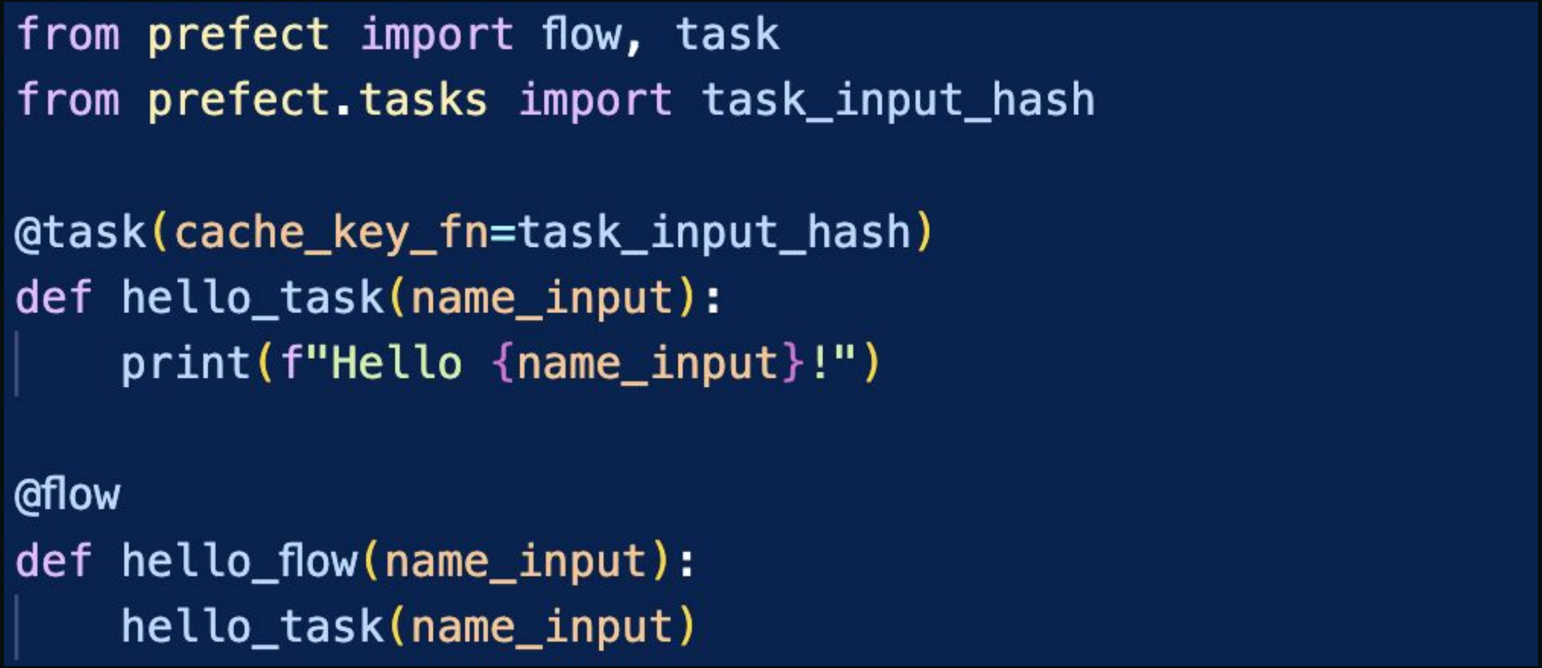
Why?

⚠️ task only

Requires persisting results (so must be serializable)

Caching: *cache_key_fn*

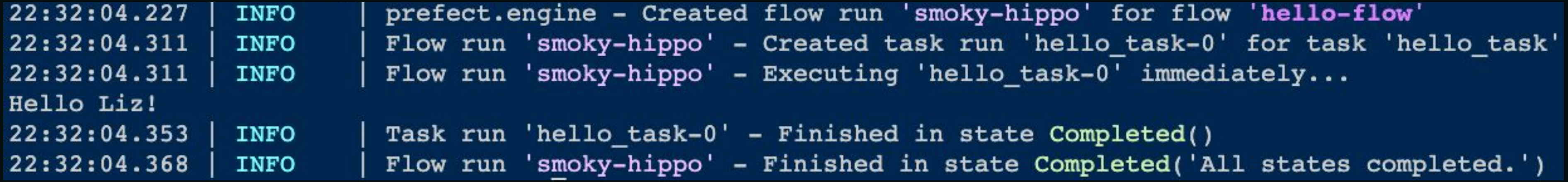
@task(cache_key_fn=task_input_hash)



```python
from prefect import flow, task
from prefect.tasks import task_input_hash

@task(cache_key_fn=task_input_hash)
def hello_task(name_input):
    print(f"Hello {name_input}!")

@flow
def hello_flow(name_input):
    hello_task(name_input)
```

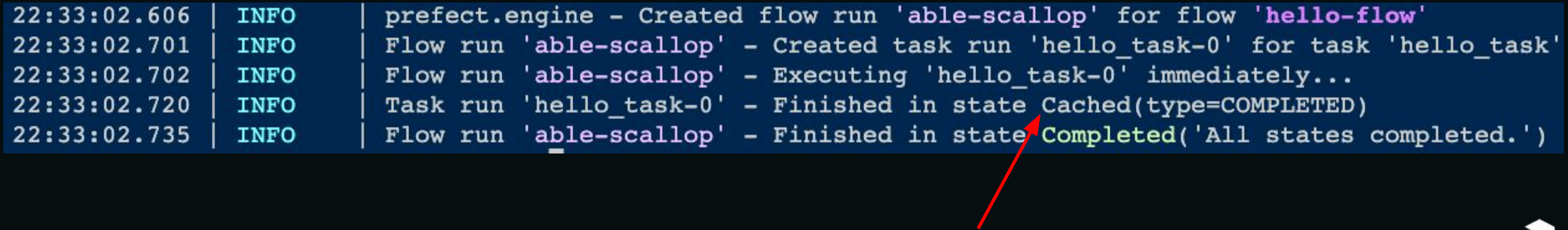
Caching

First run



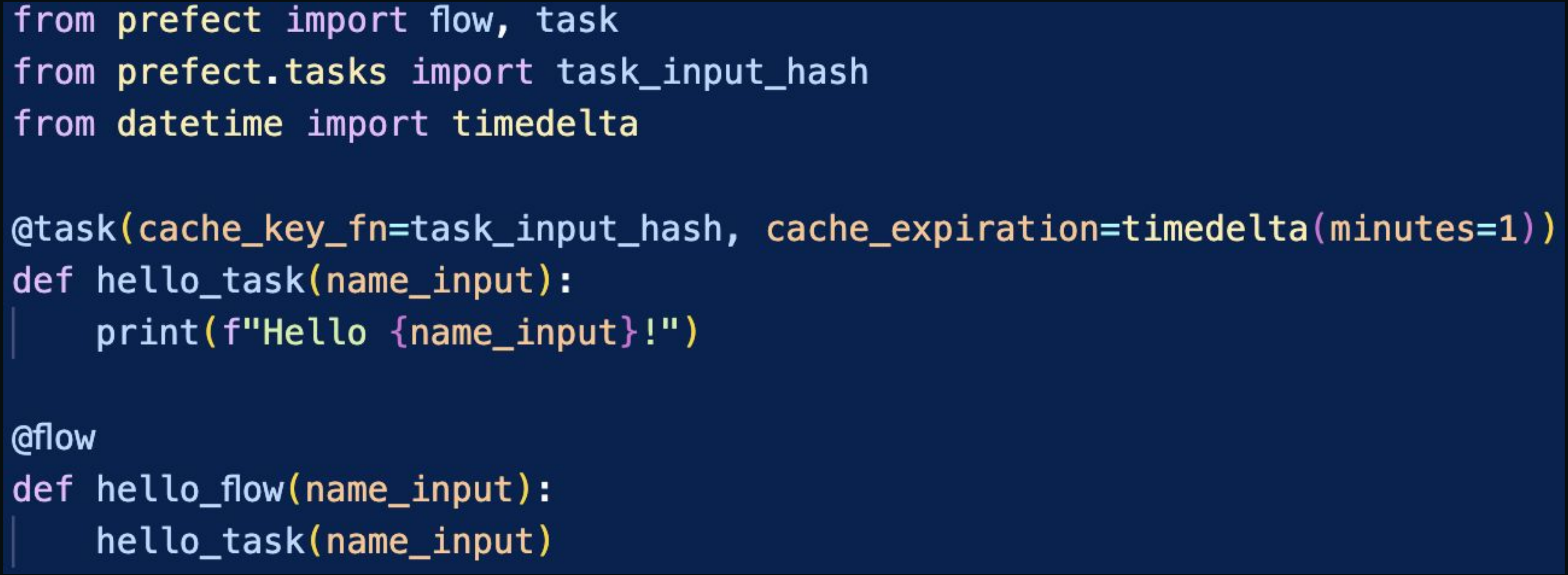
```
22:32:04.227 | INFO    | prefect.engine - Created flow run 'smoky-hippo' for flow 'hello-flow'
22:32:04.311 | INFO    | Flow run 'smoky-hippo' - Created task run 'hello_task-0' for task 'hello_task'
22:32:04.311 | INFO    | Flow run 'smoky-hippo' - Executing 'hello_task-0' immediately...
Hello Liz!
22:32:04.353 | INFO    | Task run 'hello_task-0' - Finished in state Completed()
22:32:04.368 | INFO    | Flow run 'smoky-hippo' - Finished in state Completed('All states completed.')
```

Second run



```
22:33:02.606 | INFO    | prefect.engine - Created flow run 'able-scallop' for flow 'hello-flow'
22:33:02.701 | INFO    | Flow run 'able-scallop' - Created task run 'hello_task-0' for task 'hello_task'
22:33:02.702 | INFO    | Flow run 'able-scallop' - Executing 'hello_task-0' immediately...
22:33:02.720 | INFO    | Task run 'hello_task-0' - Finished in state Cached(type=COMPLETED)
22:33:02.735 | INFO    | Flow run 'able-scallop' - Finished in state Completed('All states completed.')
```

Caching: *cache_expiration* ⏳

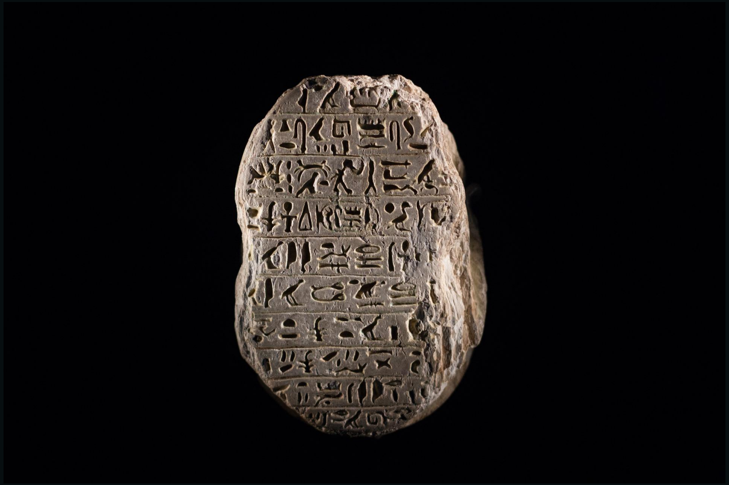


```python
from prefect import flow, task
from prefect.tasks import task_input_hash
from datetime import timedelta

@task(cache_key_fn=task_input_hash, cache_expiration=timedelta(minutes=1))
def hello_task(name_input):
    print(f"Hello {name_input}!")

@flow
def hello_flow(name_input):
    hello_task(name_input)
```

Artifacts

Artifacts

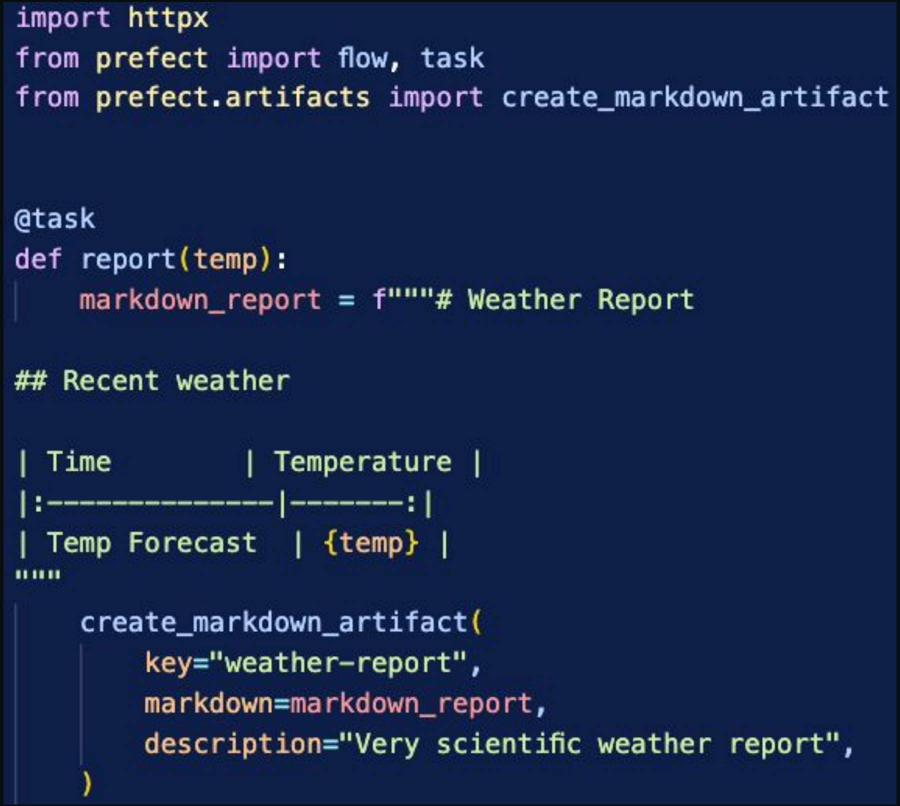
Persisted outputs such as Markdown, tables, or links.



Artifacts

- Meant for human consumption
- Examples:
    - Model scores
    - Data quality checks
    - Reports
- Gets stored in the database and shown in the UI

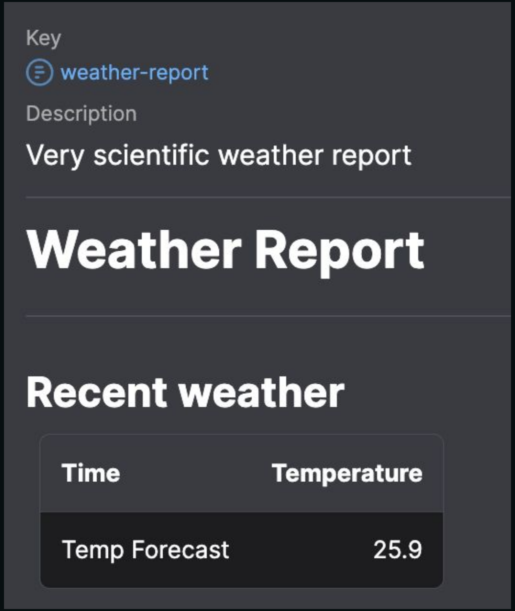
Artifacts - Markdown example of weather report



```python
import httpx
from prefect import flow, task
from prefect.artifacts import create_markdown_artifact


@task
def report(temp):
    markdown_report = f"""# Weather Report

## Recent weather

| Time          | Temperature |
|:--------------|-------:|
| Temp Forecast  | {temp} |
"""
    create_markdown_artifact(
        key="weather-report",
        markdown=markdown_report,
        description="Very scientific weather report",
    )
```

Artifacts - Markdown example

Access from UI: *Runs* timeline or *Runs->Artifacts* tab



Key

weather-report

Description

Very scientific weather report

Weather Report

Recent weather

Time	Temperature
Temp Forecast	25.9

Prefect Cloud

Prefect Cloud

- Prefect takes care of the server
- User account management (some at higher tiers)
    - Workspaces
    - Service accounts
    - RBAC
    - SSO
    - Audit logs
- Additional features

Prefect Cloud workspaces

- Paid plans can have multiple workspaces
- Each workspace is self-contained

Prefect Cloud - Default Roles (Pro + Enterprise)

Account level

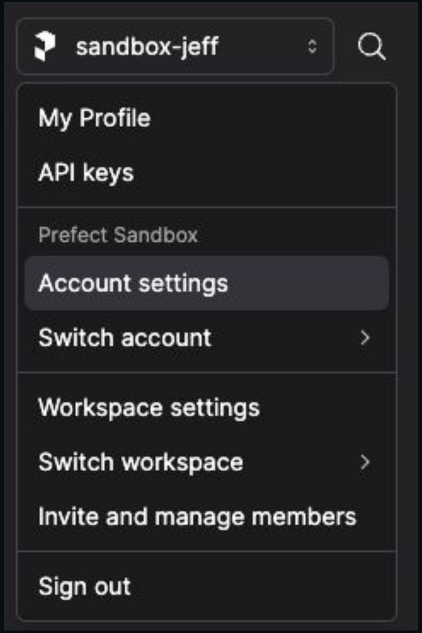
- Owner
- Admin
- Member

Workspace level

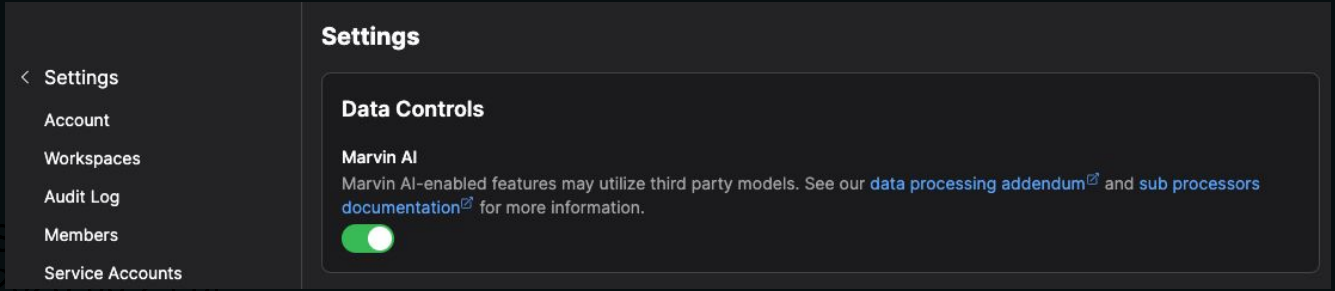
- Owner
- Developer
- Runner
- Viewer
- Worker

Error summaries by Marvin AI

Error summaries by Marvin AI



12.11.58 PM



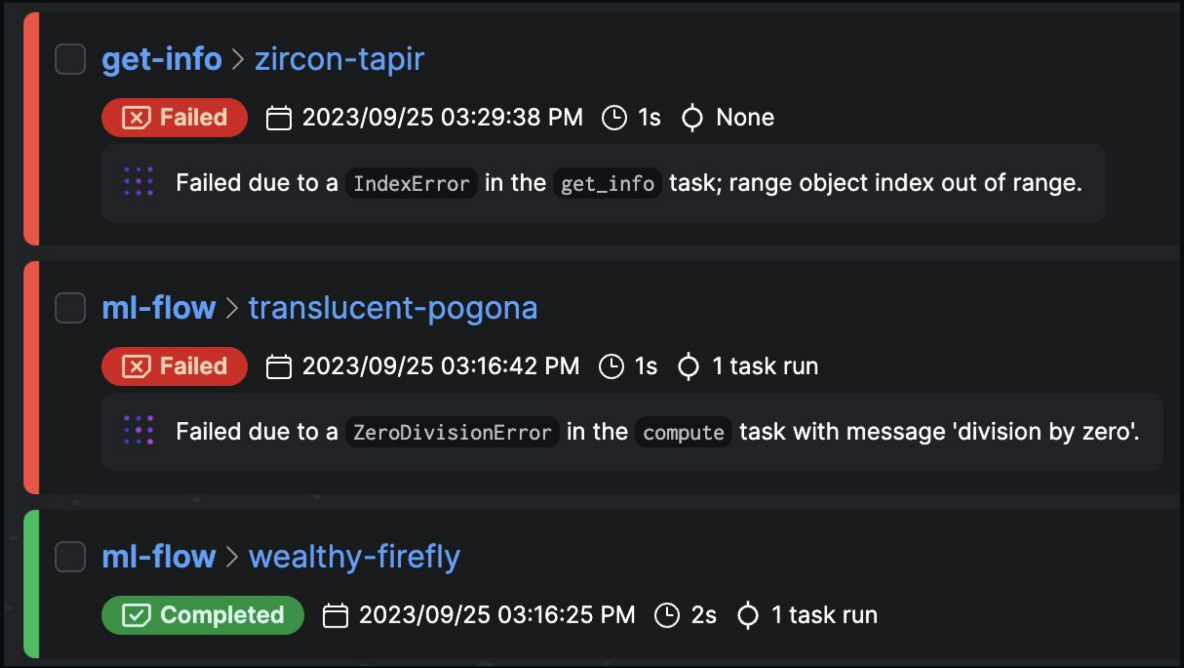
Settings

Data Controls

Marvin AI

Marvin AI-enabled features may utilize third party models. See our data processing addendum and sub processors documentation for more information.

Error summaries by Marvin AI



get-info > zircon-tapir

Failed 2023/09/25 03:29:38 PM 1s None

Failed due to a `IndexError` in the `get_info` task; range object index out of range.

ml-flow > translucent-pogona

Failed 2023/09/25 03:16:42 PM 1s 1 task run

Failed due to a `ZeroDivisionError` in the `compute` task with message 'division by zero'.

ml-flow > wealthy-firefly

Completed 2023/09/25 03:16:25 PM 2s 1 task run

Events

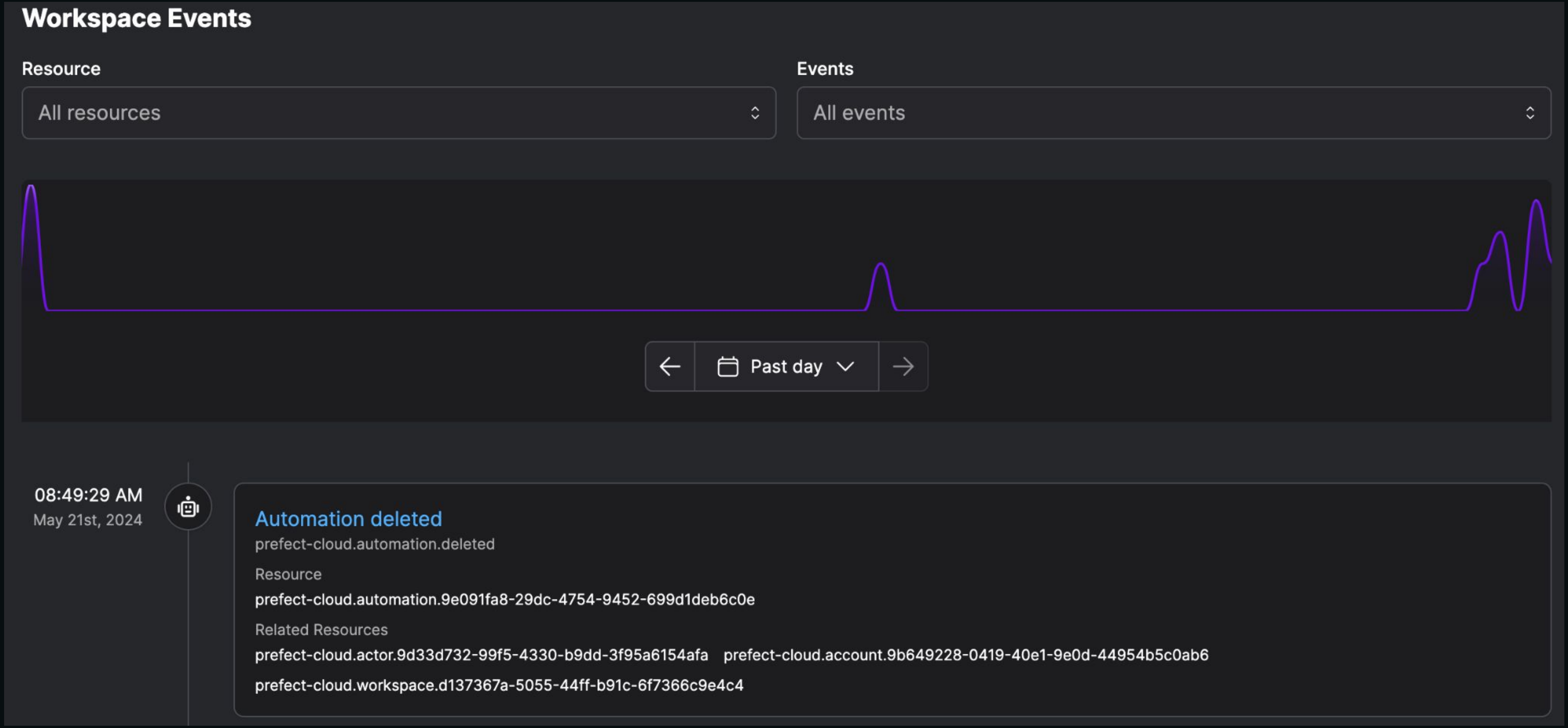
Events

- A record of what has happened

Represent:

- API calls
- State transitions
- Changes in environment

Event feed



Workspace Events

08:49:29 AM
May 21st, 2024

Automation deleted
prefect-cloud.automation.deleted

Resource
prefect-cloud.automation.9e091fa8-29dc-4754-9452-699d1deb6c0e

Related Resources
prefect-cloud.actor.9d33d732-99f5-4330-b9dd-3f95a6154afa prefect-cloud.account.9b649228-0419-40e1-9e0d-44954b5c0ab6
prefect-cloud.workspace.d137367a-5055-44ff-b91c-6f7366c9e4c4

Events

Power several Cloud features:

- Flow run logs
- Audit logs
- Automations (triggers)

Automations ⚡

Automations

Flexible framework

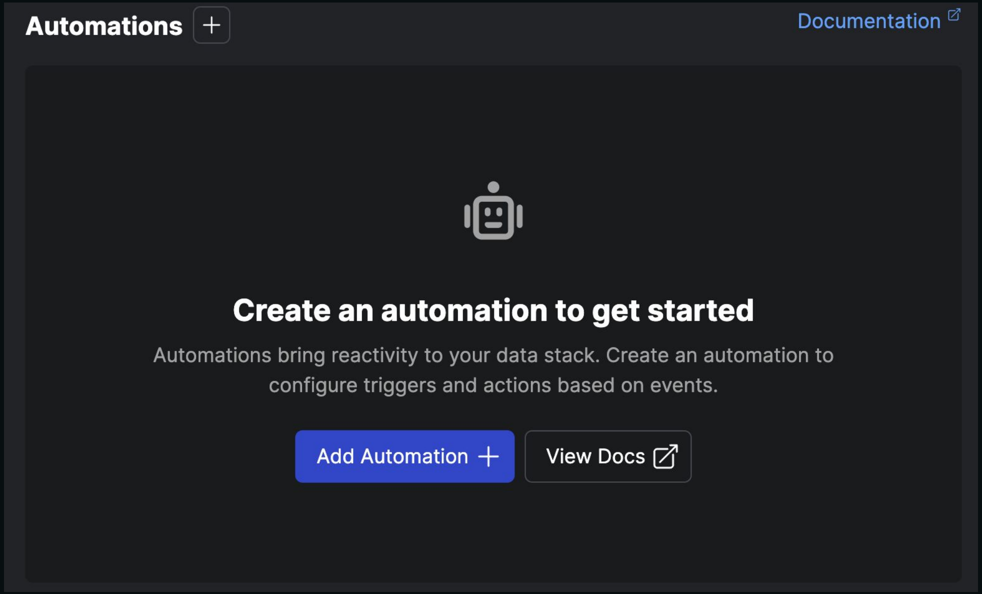
- If *Trigger* happens, do *Action*
- If *Trigger* doesn't happen in a time period, do *Action*

Automation examples

- If a flow run with tag **prod** fails, send an email 📧
- If a data quality check fails, run a deployment to fetch more data 📊
- If a work pool changes state to *Not Ready*, create an incident 🚨

Create an automation

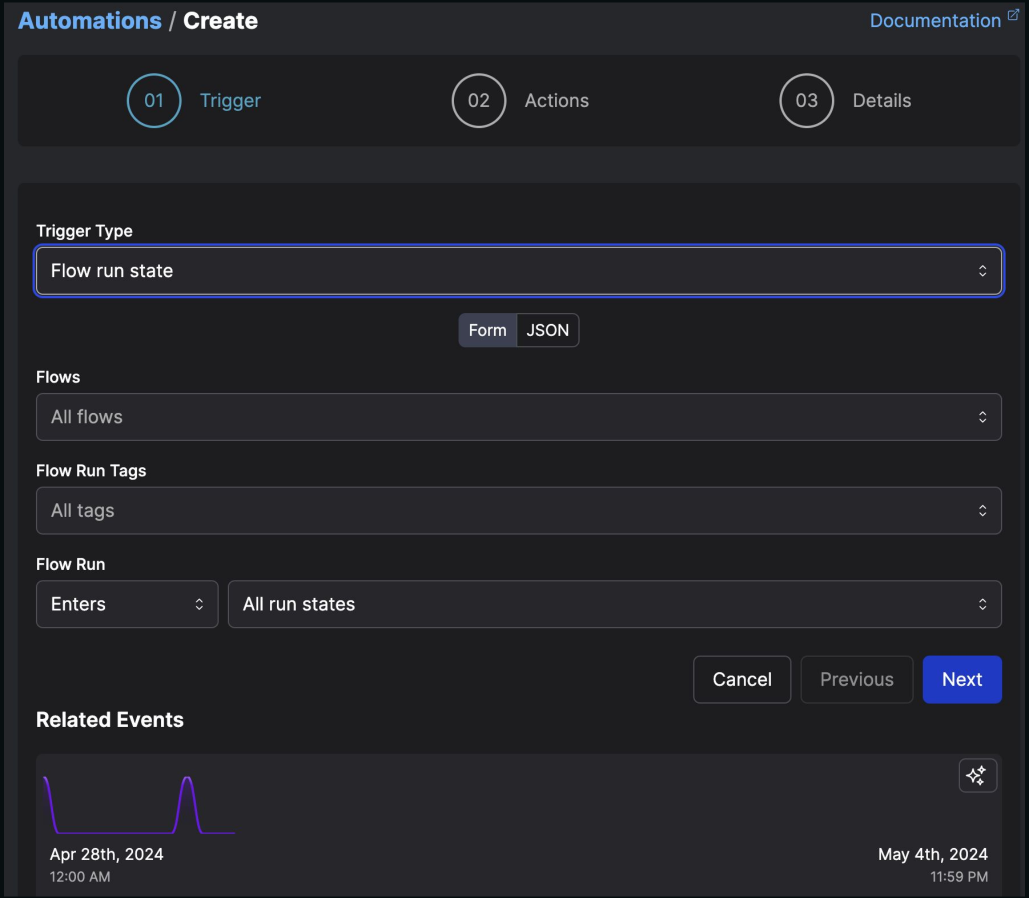
Trigger: flow run failure
Action: notification - email


Automations
Documentation

Create an automation to get started

Automations bring reactivity to your data stack. Create an automation to configure triggers and actions based on events.

Automation trigger



Automations / Create

Documentation

01 Trigger

02 Actions

03 Details

Trigger Type

Flow run state

Form | JSON

Flows

All flows

Flow Run Tags

All tags

Flow Run

Enters

All run states

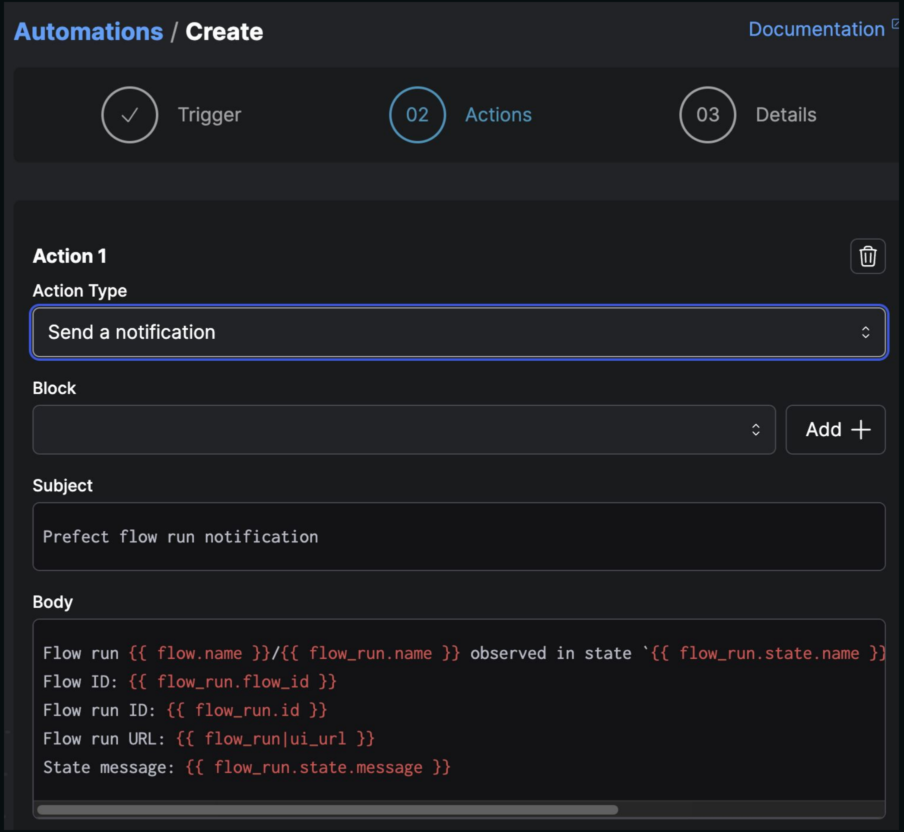
Cancel | Previous | Next

Related Events

Apr 28th, 2024
12:00 AM

May 4th, 2024
11:59 PM

Automation action



Automations / Create

Documentation

Action 1

Action Type: Send a notification

Block:

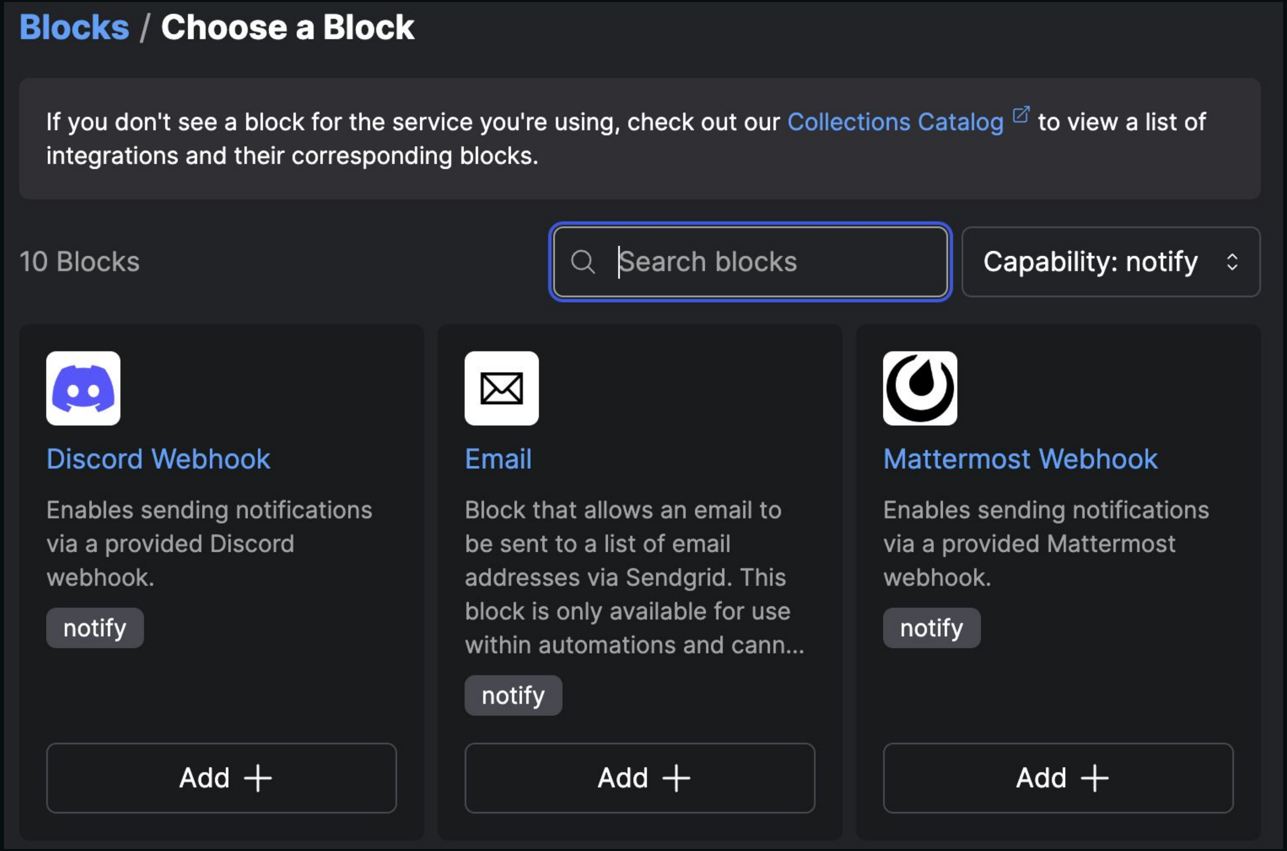
Subject

```
Prefect flow run notification
```

Body

```
Flow run {{ flow.name }}/{{ flow_run.name }} observed in state `{{ flow_run.state.name }}
Flow ID: {{ flow_run.flow_id }}
Flow run ID: {{ flow_run.id }}
Flow run URL: {{ flow_run|ui_url }}
State message: {{ flow_run.state.message }}
```

Create a block with **notify** capability



Blocks / **Choose a Block**

If you don't see a block for the service you're using, check out our Collections Catalog to view a list of integrations and their corresponding blocks.

10 Blocks

Search blocks

Capability: notify

Discord Webhook

Enables sending notifications via a provided Discord webhook.

notify

Add +

Email

Block that allows an email to be sent to a list of email addresses via Sendgrid. This block is only available for use within automations and cann...

notify

Add +

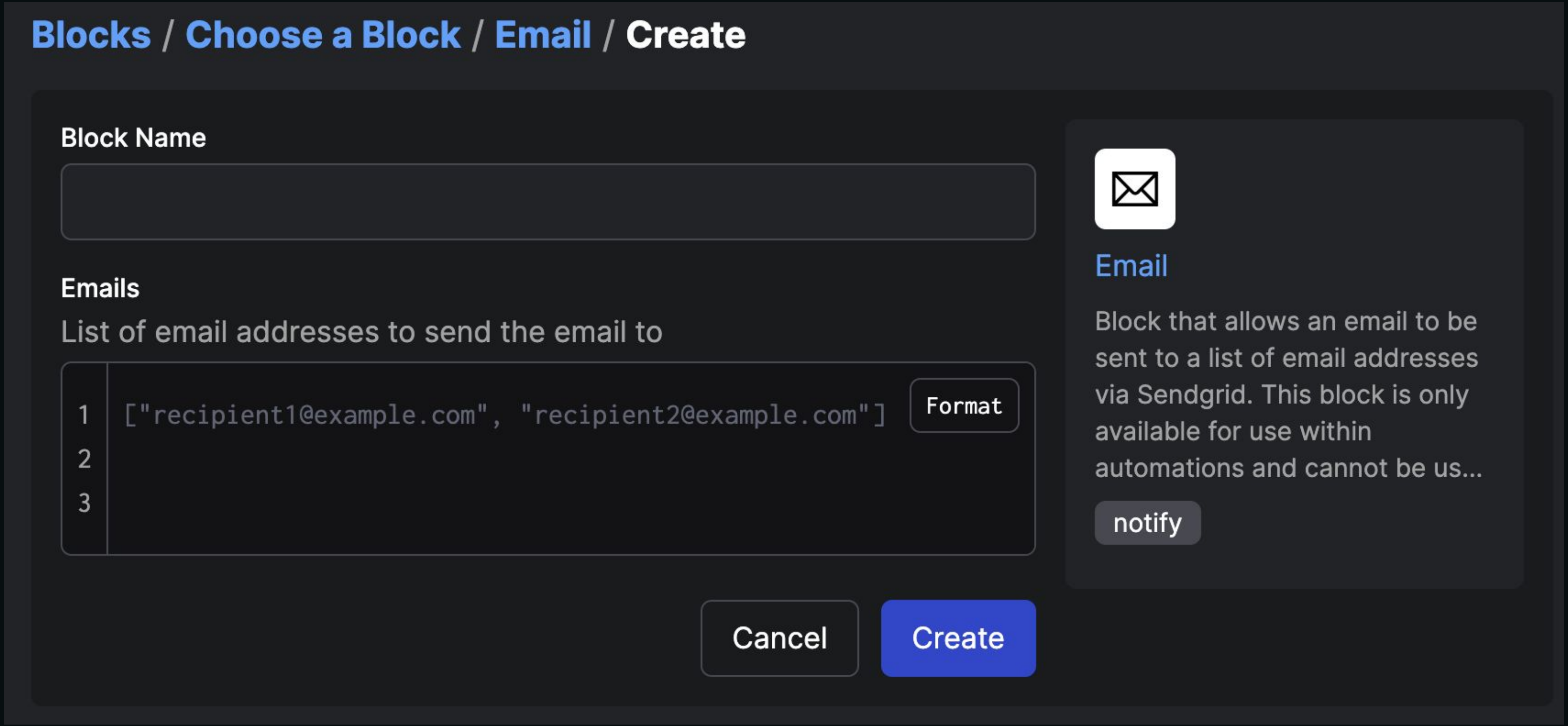
Mattermost Webhook

Enables sending notifications via a provided Mattermost webhook.

notify

Add +

Create an **Email** block



Blocks / Choose a Block / Email / Create

Block Name

Emails

List of email addresses to send the email to

```
["recipient1@example.com", "recipient2@example.com"]
```

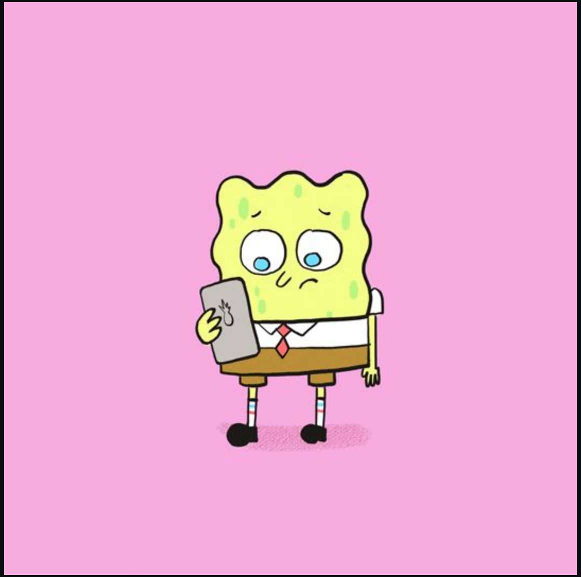
Email

Block that allows an email to be sent to a list of email addresses via Sendgrid. This block is only available for use within automations and cannot be us...

Create an **Email** block

Name and save your automation.

Now you’ll receive an email when a flow run changes state!



103 Recap

You’ve learned about

- Working with data
- Prefect Cloud
- Error summaries by Marvin AI
- Events
- Automations

Lab 103

103 Lab

- In the UI, make an email notification automation for a flow run completion
  - ❗ use an **Email** block type
- Check out the event feed in the UI
- Stretch 1: Create a flow that contains a task that takes advantage of caching using *task_input_hash*
- Stretch 2: Create a Markdown artifact that prints a weather forecast in a nicely formatted table