PACC

Prefect Associate Certification Course

PREFECT

Slack

✅ **Join Prefect Community Slack**

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

Norms

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

Norms

Zoom

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

Slack

- Use threads
- Emoji responses 🙂

Introductions

Goals

Goals

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

Overview

What is Prefect?

Prefect is an **orchestration and observability platform** that **empowers developers** to **build and scale resilient code quickly**, turning scheduled jobs into **resilient, data applications**. 💙

Prefect helps you avoid roadblocks on the route to production

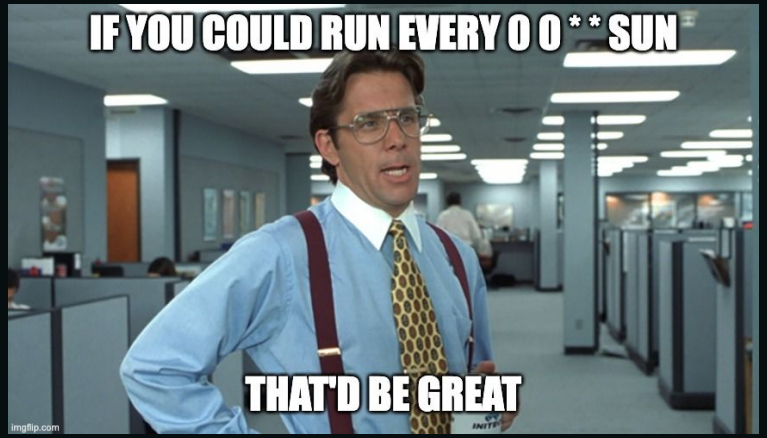
Why workflow management?

Answers the questions:

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

When?

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



Where?

- Locally
- Easily move to cloud providers

How?

- Docker, K8s, or a subprocess
- From the UI, CLI, or code
- Human-in-the-loop approval workflow option

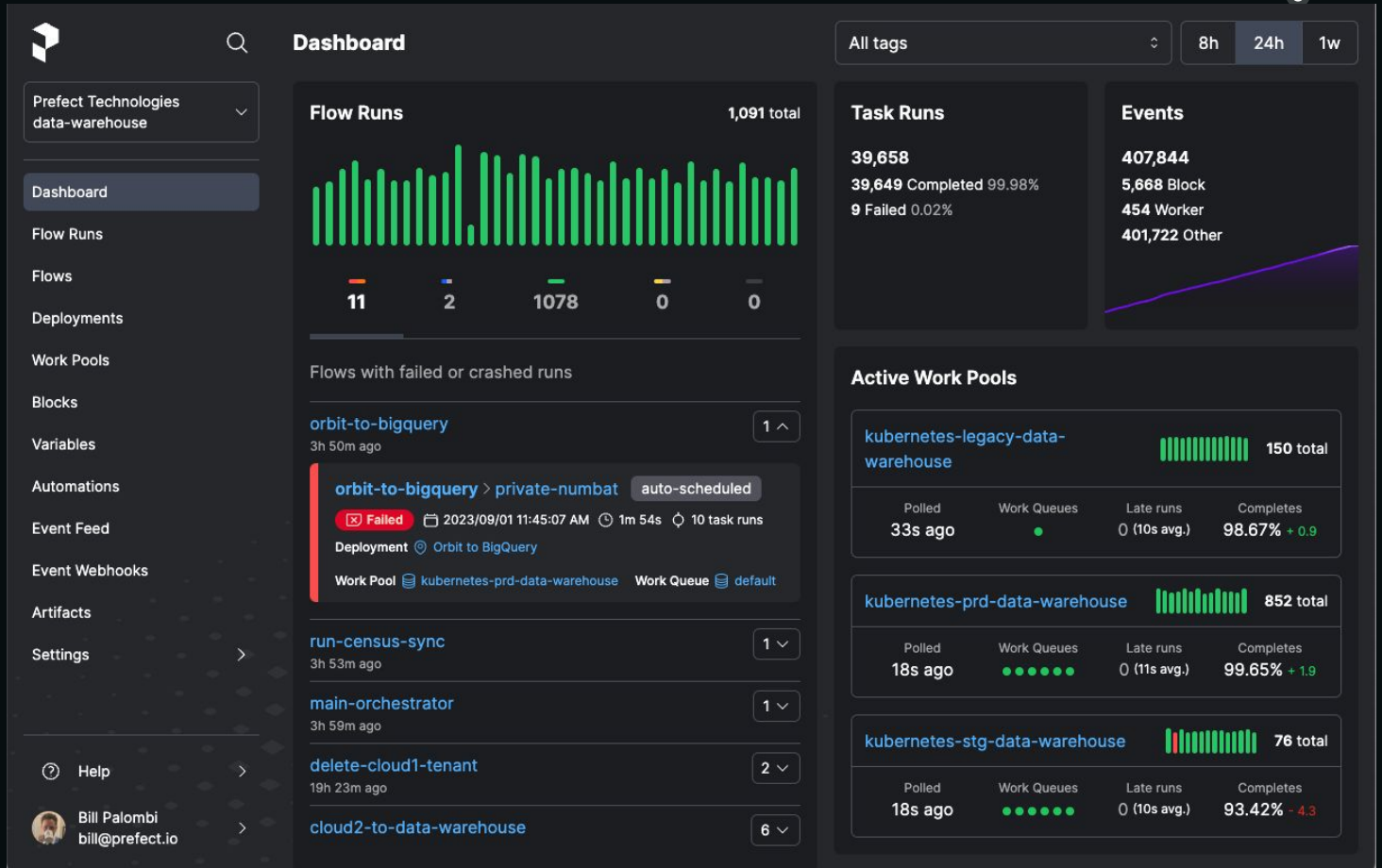
Who?

- Auth - SSO/SCIM
- RBAC
- Auditable
- Object level access controls

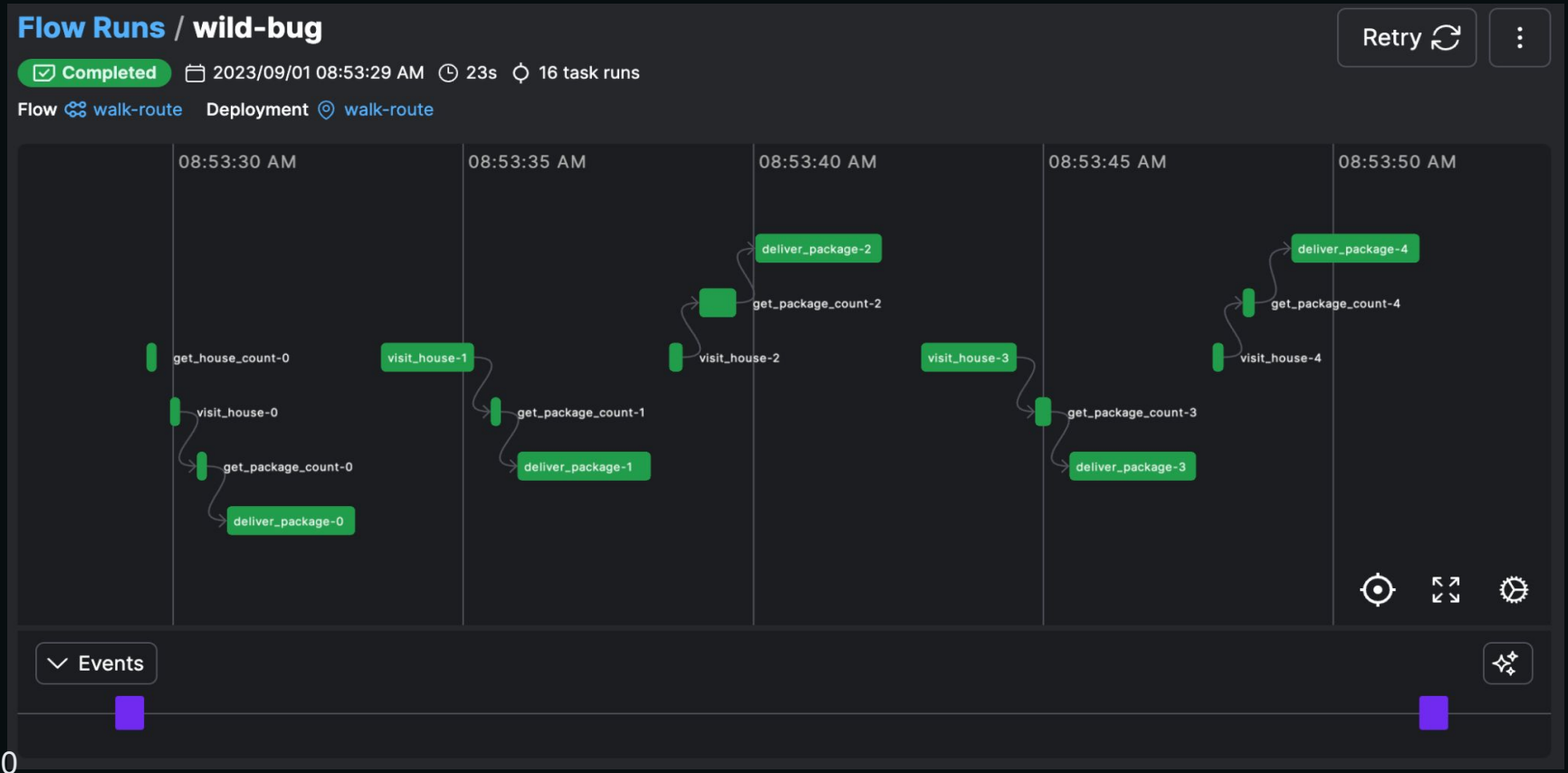
Why Prefect for workflow management?

- Pythonic
- Monitoring & observability
- With teams: standardized workflow management is a must - Prefect provides guardrails

Understand the state of your workflows



Orchestrate and observe



If you give an engineer a job…

Could you just fetch this data and save it? Oh, and …

1. set up logging?
2. do it every hour?
3. visualize the dependencies?
4. automatically retry if it fails?
5. create an artifact for human viewing?
6. add caching?
7. add collaborators to run and view - who don’t code?
8. send me a message when it succeeds?
9. run it in a Docker container-based environment?
10. pause for input?
11. automatically declare an incident when a % of workflows fail?
12. automatically run it in response to an event?

Business outcomes

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

101 - Prefect basics

101 Agenda

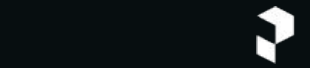
- Setup: *version, login, set*
- From Python function to Prefect flow
- Ceate a deployment with *.serve()*
- Run a deployment
- Deployment schedules
- Resources

prefect version

Prefect information in the CLI

prefect version

```
Version:              2.18.1
API version:          0.8.4
Python version:       3.12.2
Git commit:           8cff545a
Built:                Thu, Apr 25, 2024 3:40 PM
OS/Arch:              darwin/arm64
Profile:              sandbox-jeff
Server type:          cloud
```

Run *prefect version* now

If you see *version* lower than *2.18.1*

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
    c. Additional features such as automations, push work pools, managed work pools, metrics, incidents
    d. 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 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*

Prefect profiles

- Profiles live in ***~/.prefect/profiles.toml*** 📁
- Your profile stays active until you switch to another profile 🙂
- Save connection info to Prefect Cloud in a profile

Prefect Cloud

Authenticate your CLI via browser or API key:

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***

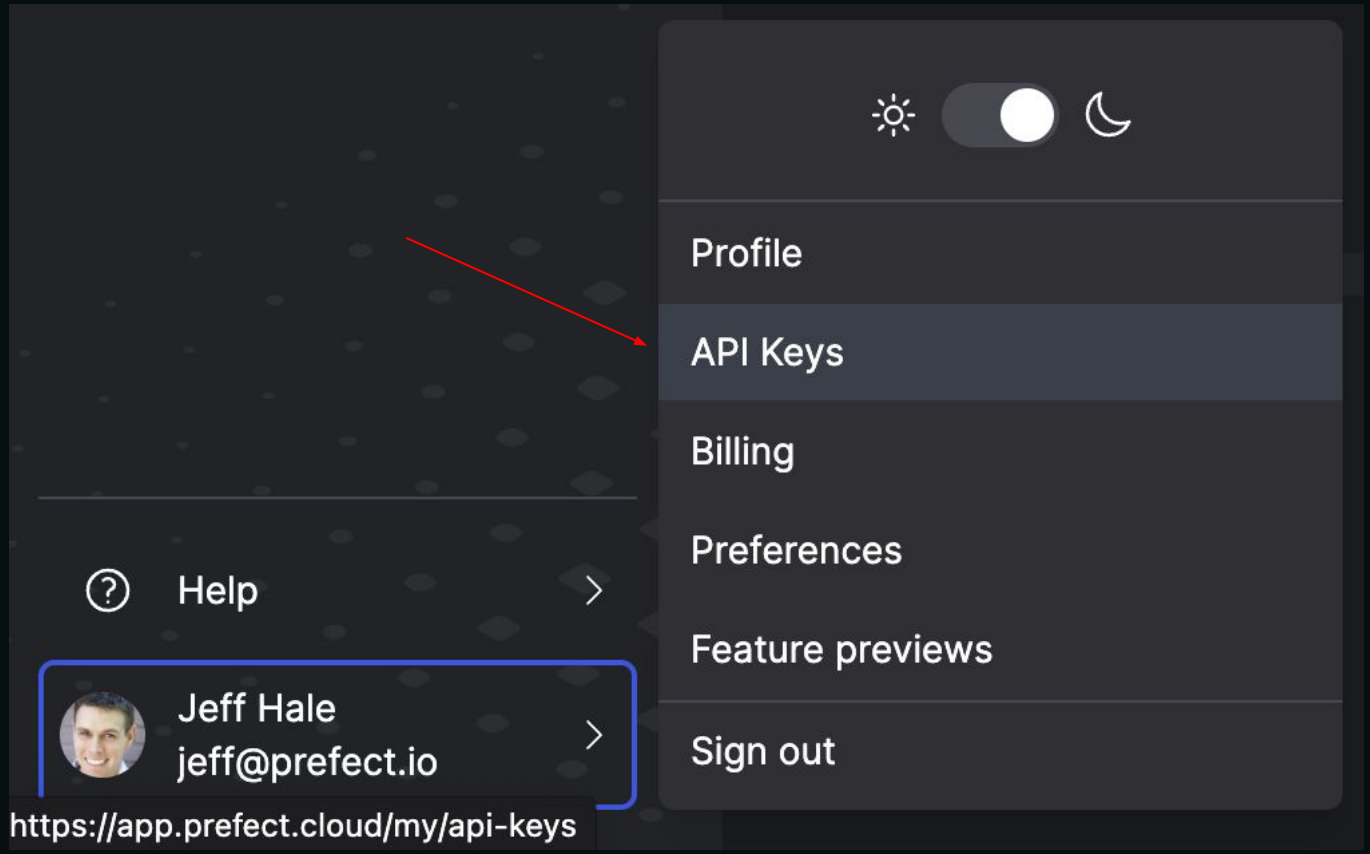
Creates and saves an API key for you 🔑

Prefect Cloud

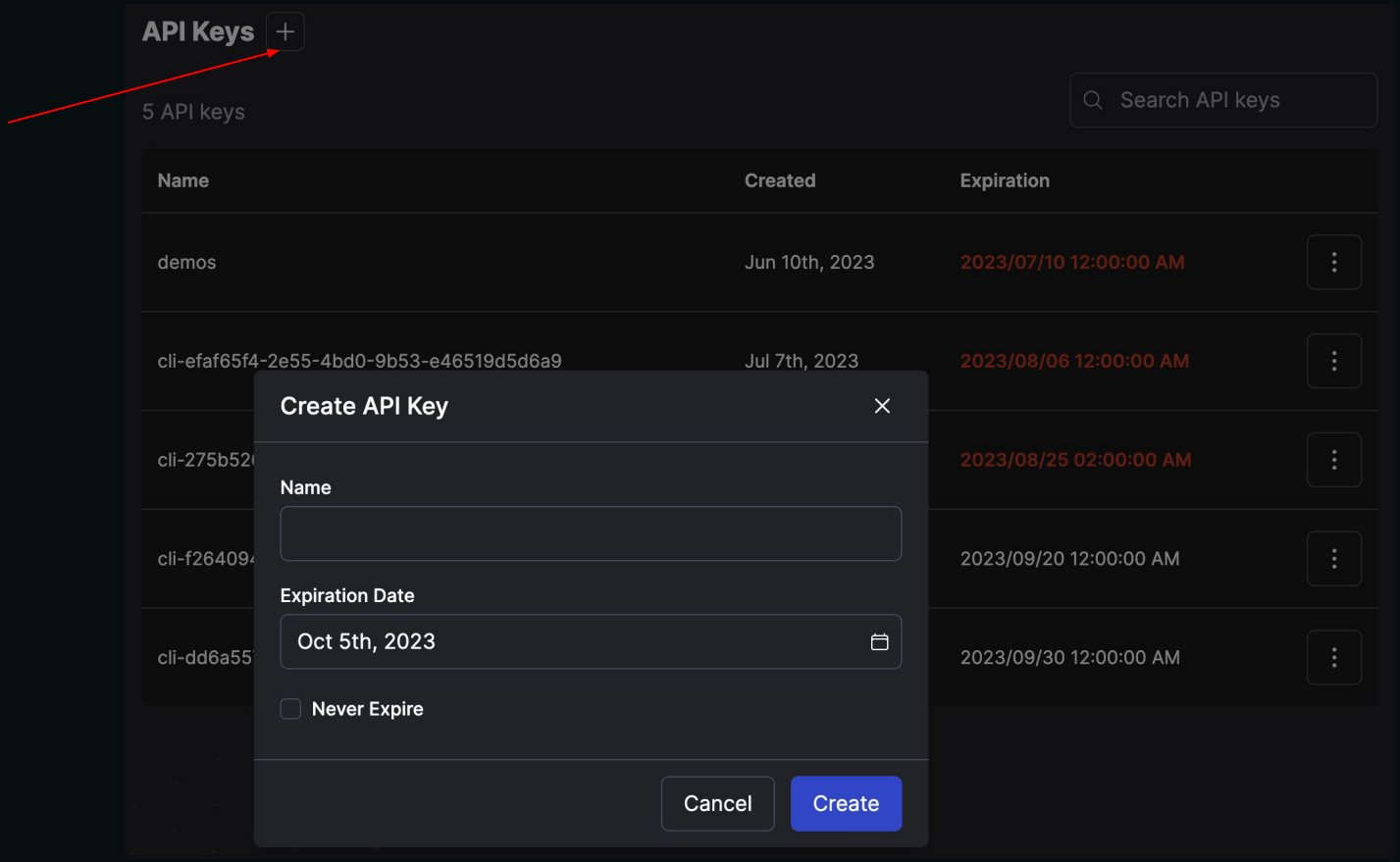
Or, if UI doesn't work: create and paste an API key

Manually create an API key from Prefect Cloud in the UI

Prefect Cloud - API key



Prefect Cloud - API key



Flows

Course project

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

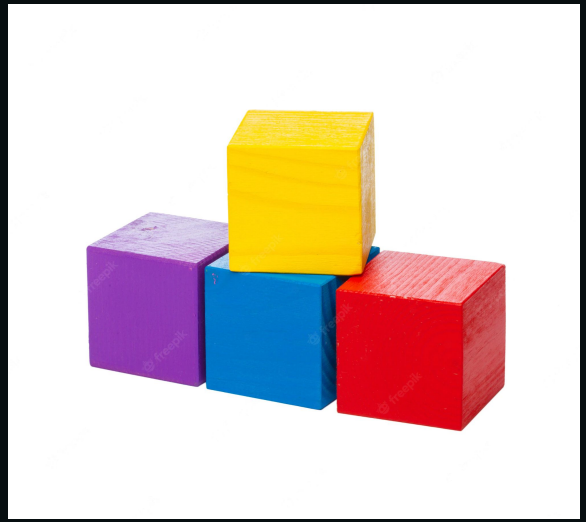
open-meteo.com

Starting point: 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



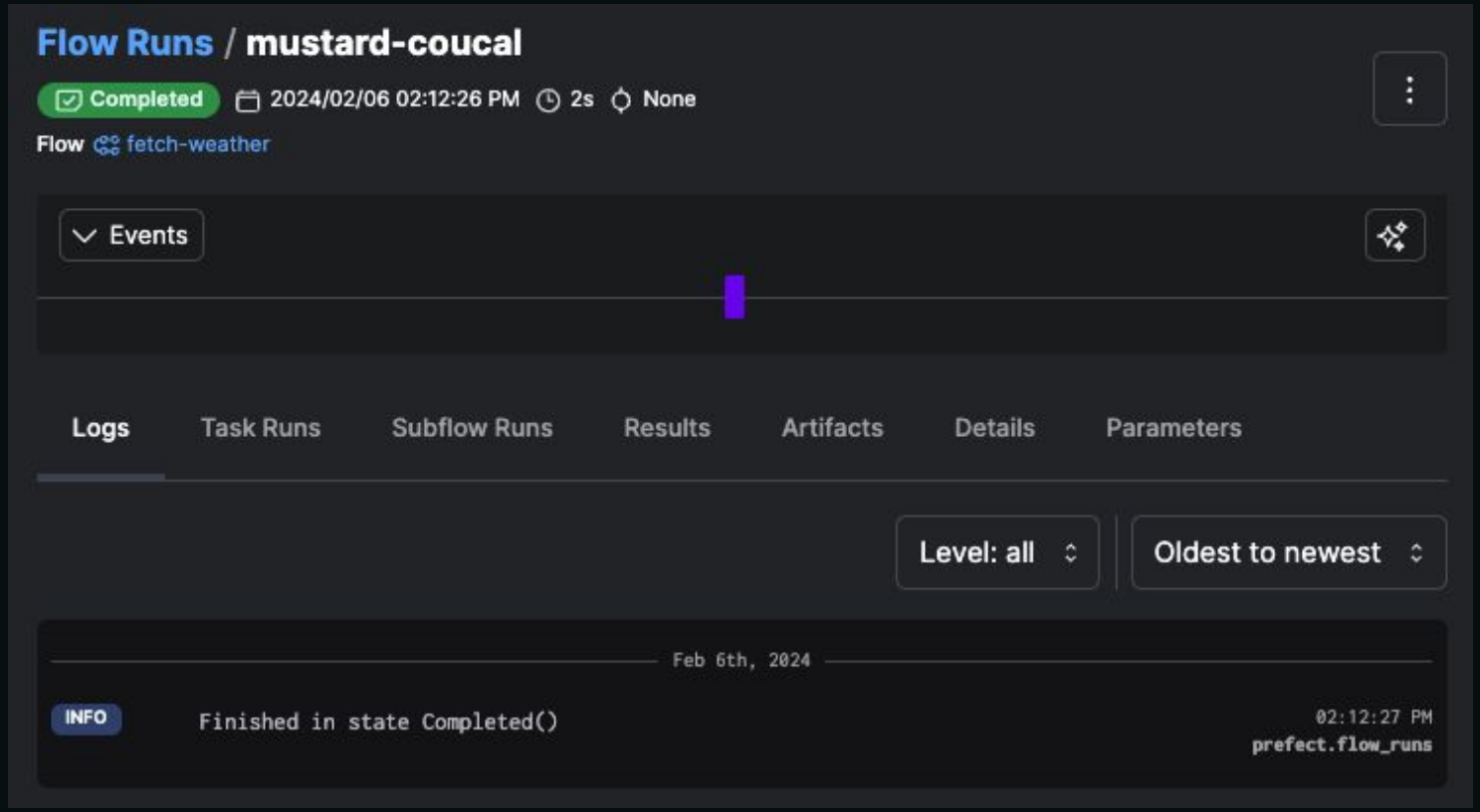
Make it a flow

```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*

```
14:12:25.969 | INFO    | prefect.engine - Created flow run 'mustard-coucal' for flow 'fetch-weather'
14:12:25.972 | INFO    | Flow run 'mustard-coucal' - View at https://app.prefect.cloud/account/9b649228-041
9-40e1-9e0d-44954b5c0ab6/workspace/d137367a-5055-44ff-b91c-6f7366c9e4c4/flow-runs/flow-run/60b02758-beeb-4a
17-bb67-02f0d259811c
Forecasted temp C: 5.1 degrees
14:12:27.962 | INFO    | Flow run 'mustard-coucal' - Finished in state Completed()
```

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



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! 🎉

Deployments

- Server-side representation of a flow
- Contains meta-data for remote orchestration
- Can be run on various infrastructure
- Can be kicked off
  - manually (from the UI or CLI)
  - on a schedule
  - automatically, in response to an event trigger

.serve() method

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

```
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-b29f-4e0b3760d4c6/deployments/deployment/73c53509-8e7f-4924-a208-9d9bf2a50558
```

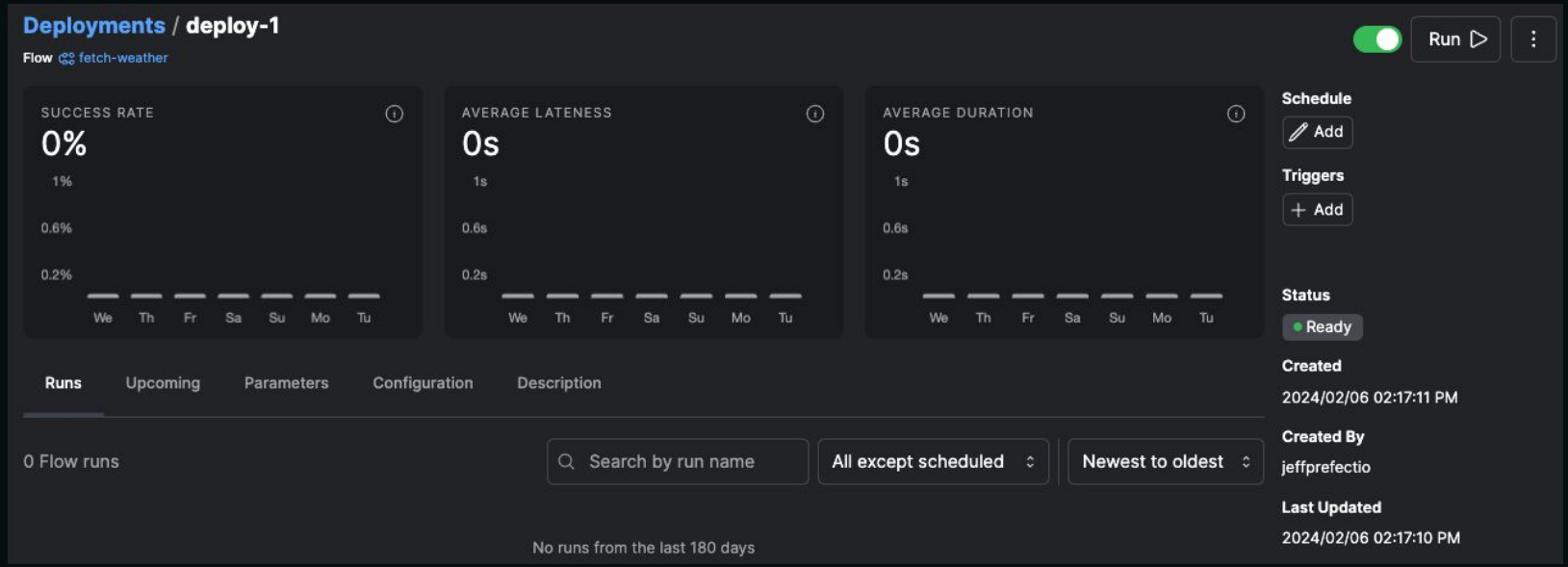

You just made a deployment!

Deployment

- Wraps your **flow:** turns it into a **workflow application**
- Contains all the needed metadata to run your flow in production
- Your flow's passport to orchestration land!

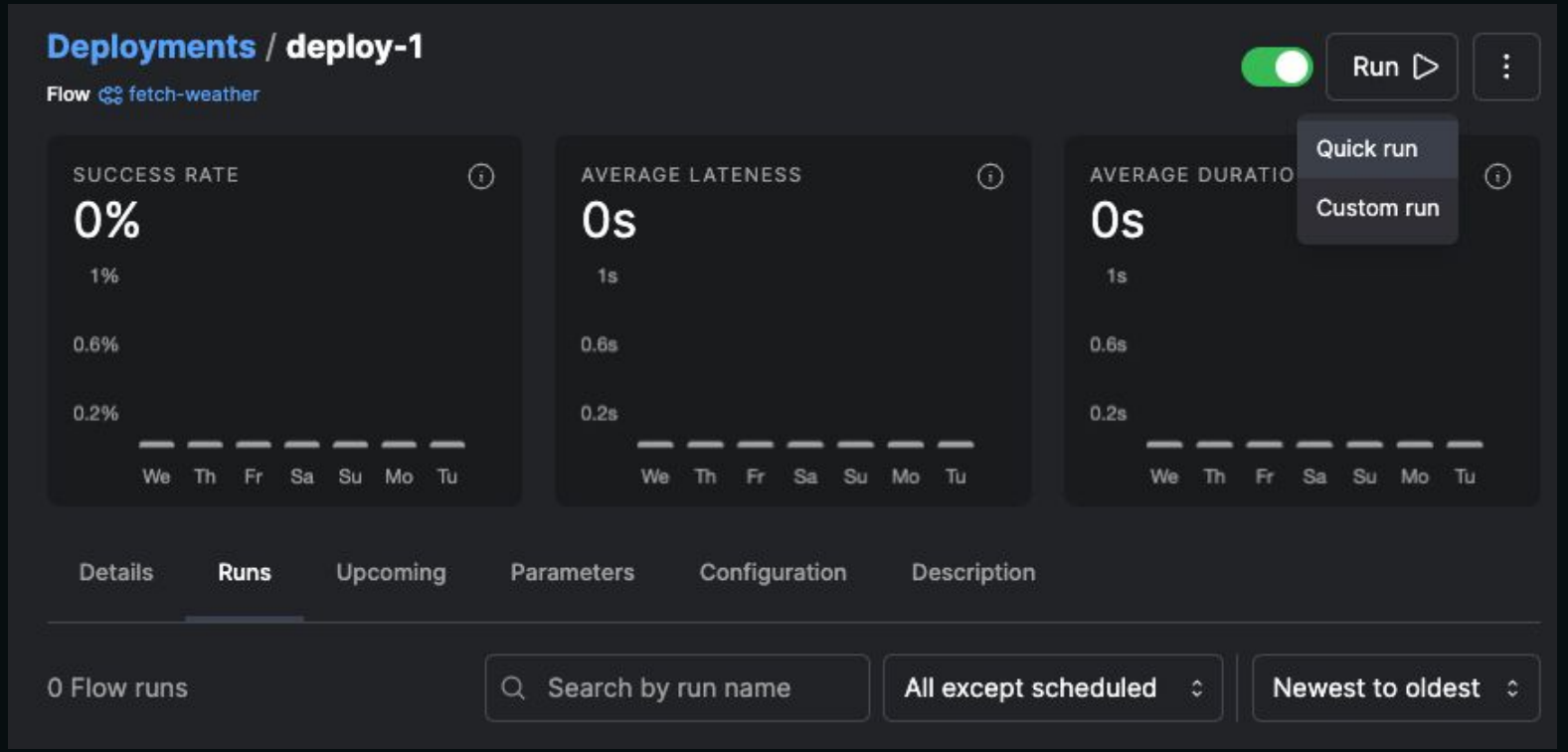
Check out the deployment in the UI

Deployment page

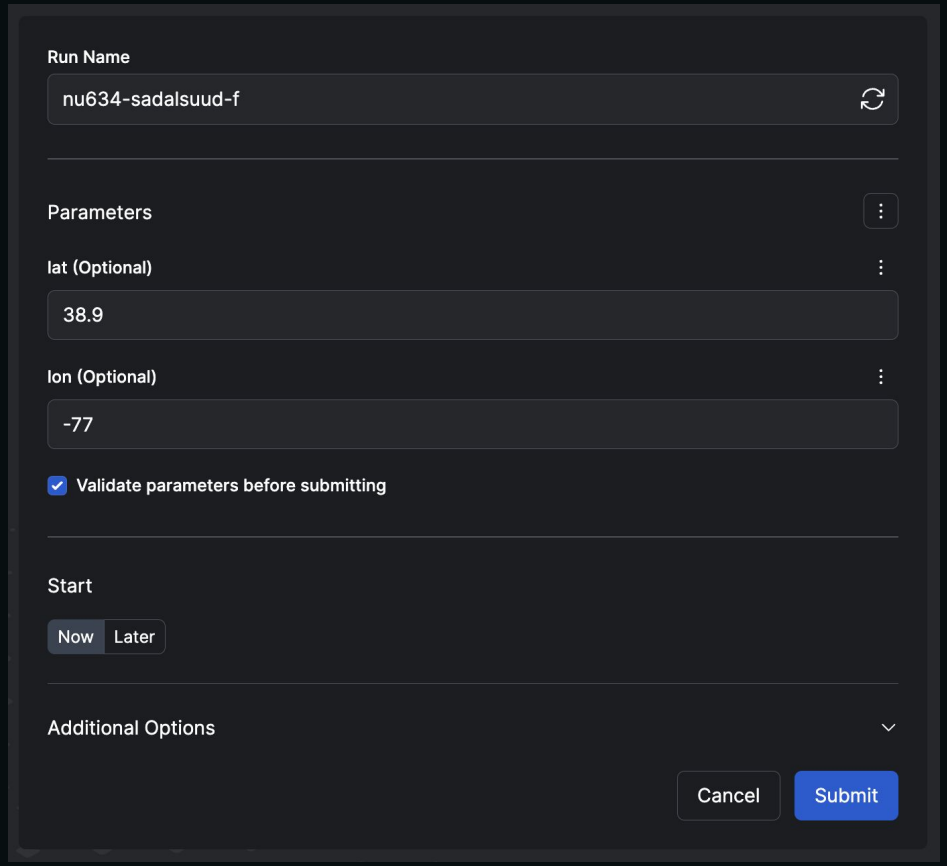


Run a deployment

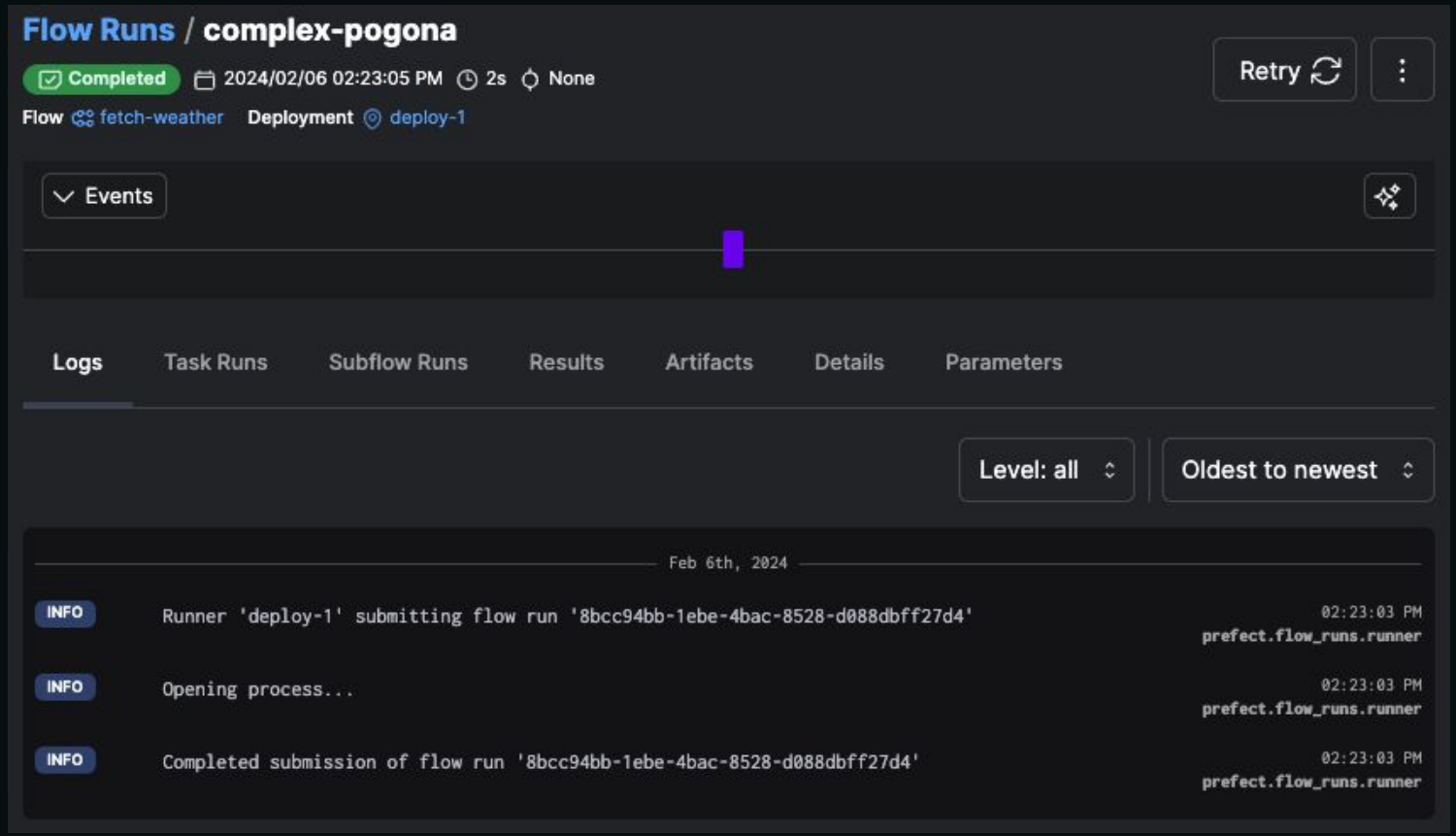
Run manually from UI: **Run** -> **Quick run**



Adjust the entrypoint flow params with a **Custom run**



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



Run deployment manually from CLI

prefect deployment run my_entrypoint_flow:my_deployment

.serve()

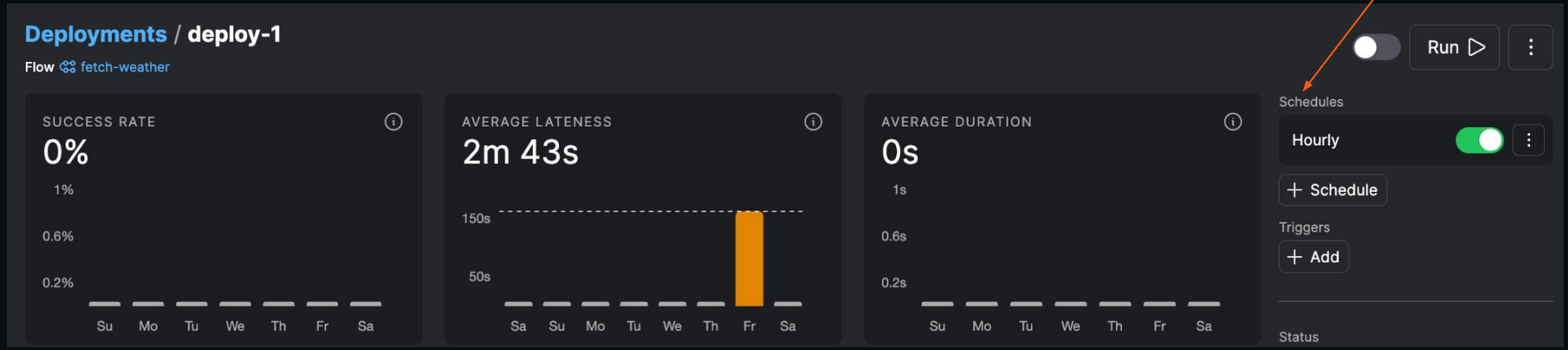
Shut down the server with ***control*** **+** ***c***

Scheduling

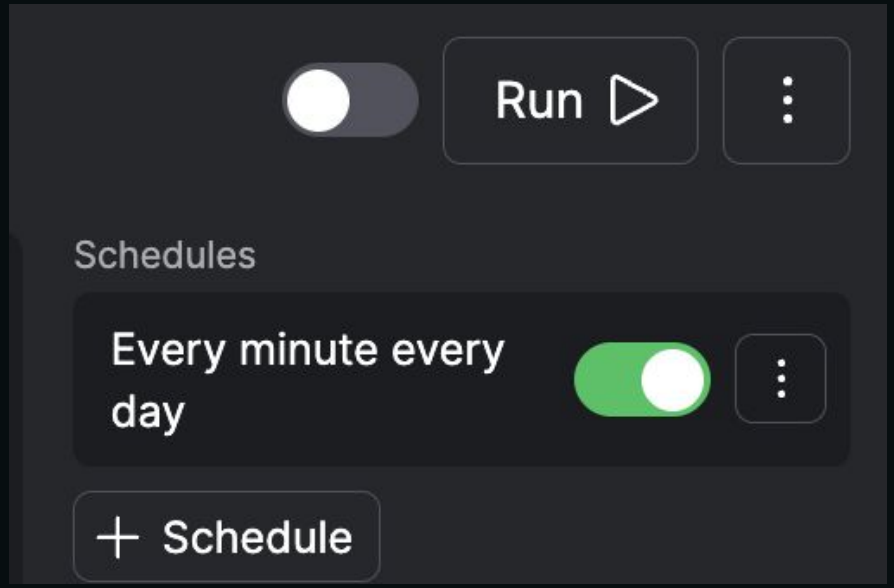
Create a deployment schedule

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

Create, pause, and delete schedules from the UI



Click **+ Schedule on** the **Deployment** page in the UI



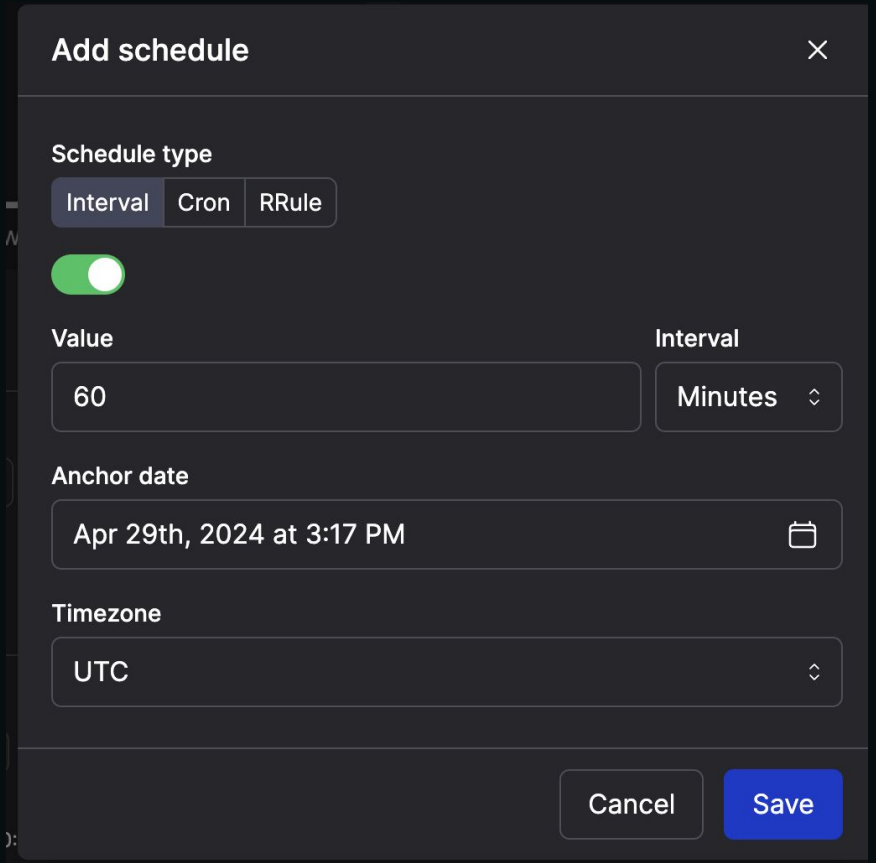
Add a schedule when creating a deployment with *.serve()*

```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



RRule

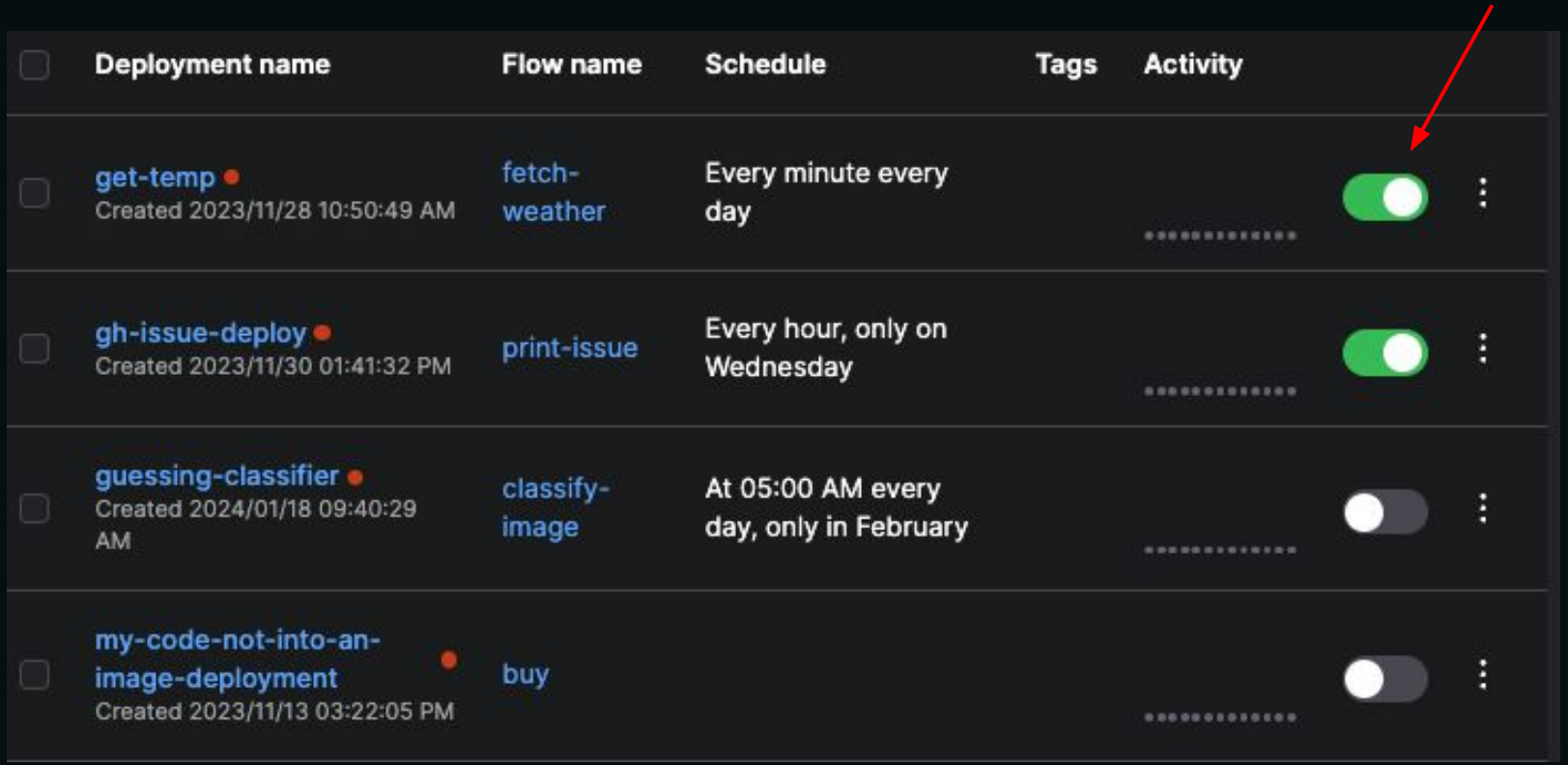
RRule cheat sheet: https://jkbrzt.github.io/rrule/

Or ask Marvin (another Prefect package) *pip install 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

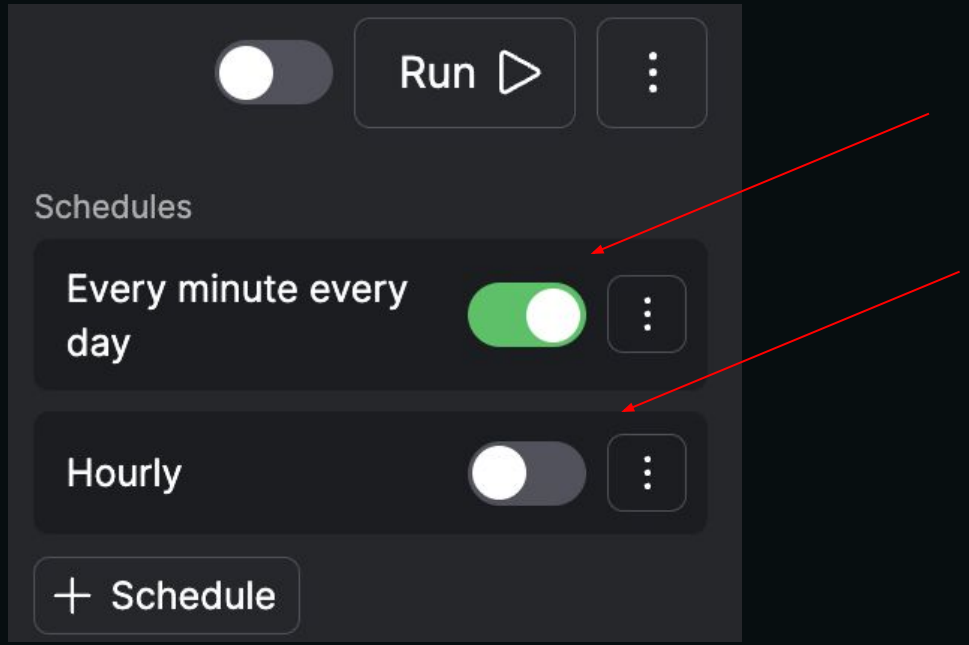
Pause/resume a deployment's schedules from UI



Note ⏸️

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

Pause/resume individual schedules from UI



Parameters

Parameters - argument values for entrypoint flow function

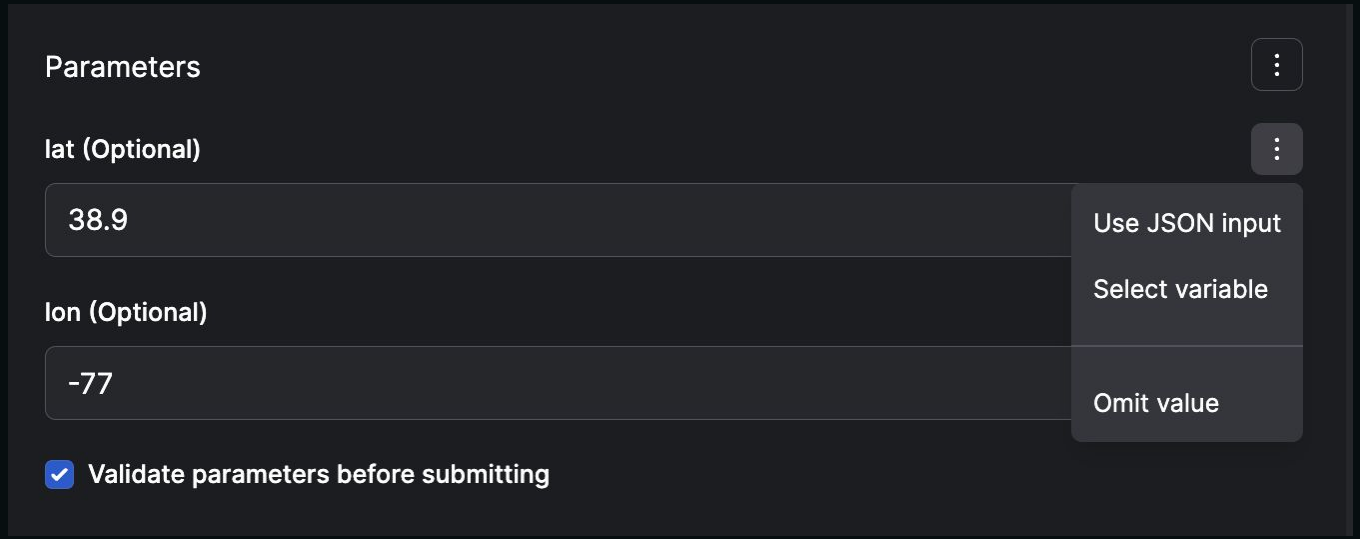
If your flow function has params and no defaults, you must feed it (give it values).

Parameters 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 at deployment creation time

Can specify in *.serve()*

```
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}'

Terms recap

Flow = a workflow

Flow run = an individual run of a flow

Deployment = a workflow application

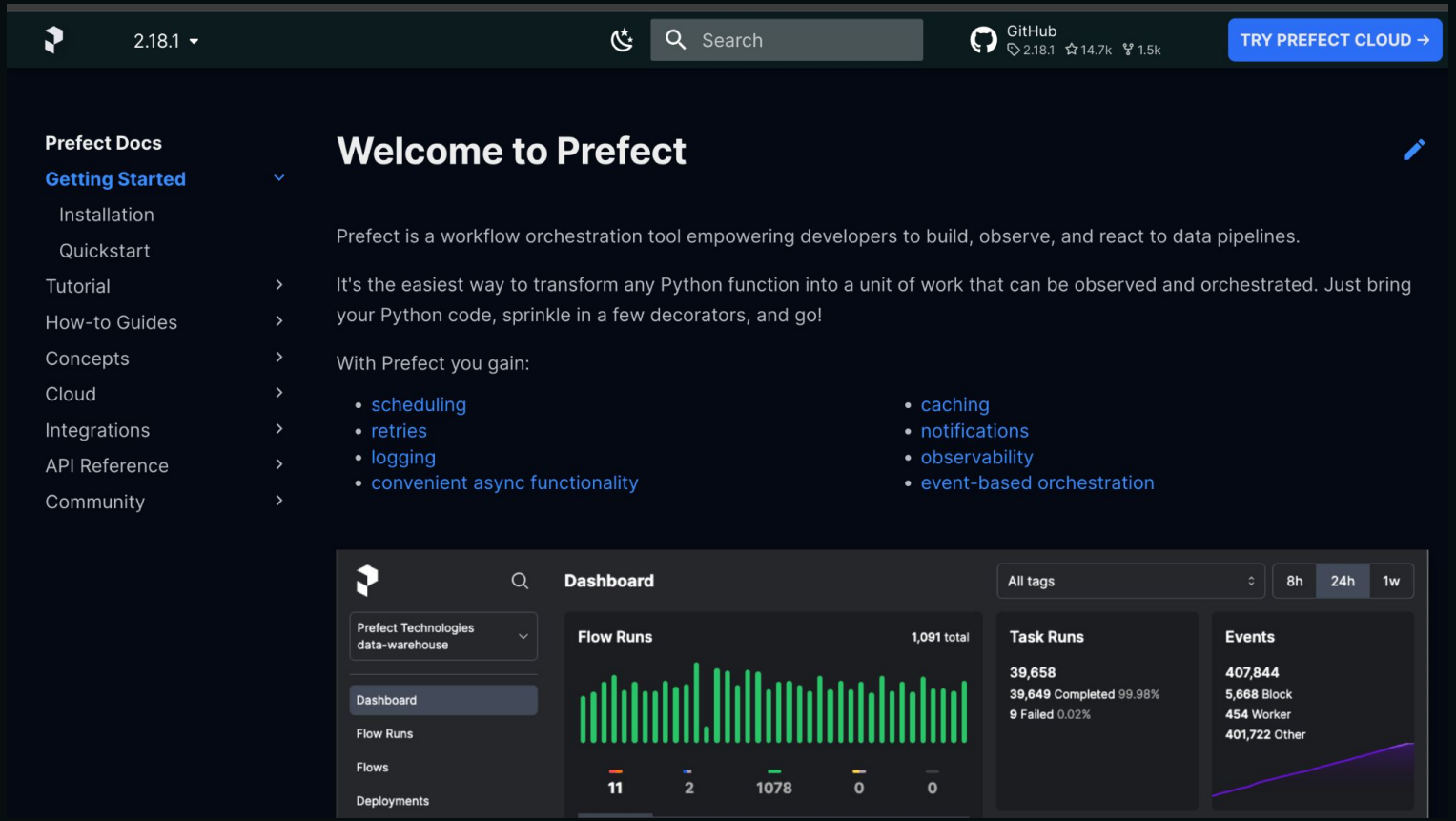
- Can schedule repeated flow runs
- Can run remotely
- Other team members can access

Resources

Docs - *docs.prefect.io*

Use the docs

Docs - *docs.prefect.io*



Prefect Community Slack

Lots of 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*
- Prefect Profiles
- From Python function to Prefect flow
- Create a deployment with *flow.serve()*
- Run a deployment from the UI
- Create and pause schedules
- Resources: docs, Slack, Prefect GitHub repo

Lab 101

Lab norms for breakout rooms

1. 🙂 Introduce yourselves
2. 🎥 Camera on (if possible)
3. 💻 One person shares screen (if you need to leave and come back to Zoom to enable screen sharing, do that now)
4. 🧑‍💻 Everyone codes
5. 🙋 Each person talks
6. 📝 Share code in Slack thread - learn from other groups
7. 😌 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 a 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
- 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]
```


If you give an engineer a job…

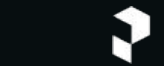
Could you just fetch this data and save it? Oh, and …

1. set up logging?
2. do it every hour?
3. visualize the dependencies?
4. automatically retry if it fails?
5. create an artifact for human viewing?
6. add caching?
7. add collaborators to run and view - who don't code?
8. send me a message when it succeeds?
9. run it in a Docker container-based environment?
10. pause for input?
11. automatically declare an incident when a % of workflows fail?
12. automatically run it a workflow response to an event?

102 - Intro to orchestration

102 Agenda

- Tasks
- Logging
- Retries
- Results
- Artifacts
- Caching



Tasks

Tasks

Add the *@task* decorator to a function

- Enable task retries
- Enable caching
- Enable 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"
```

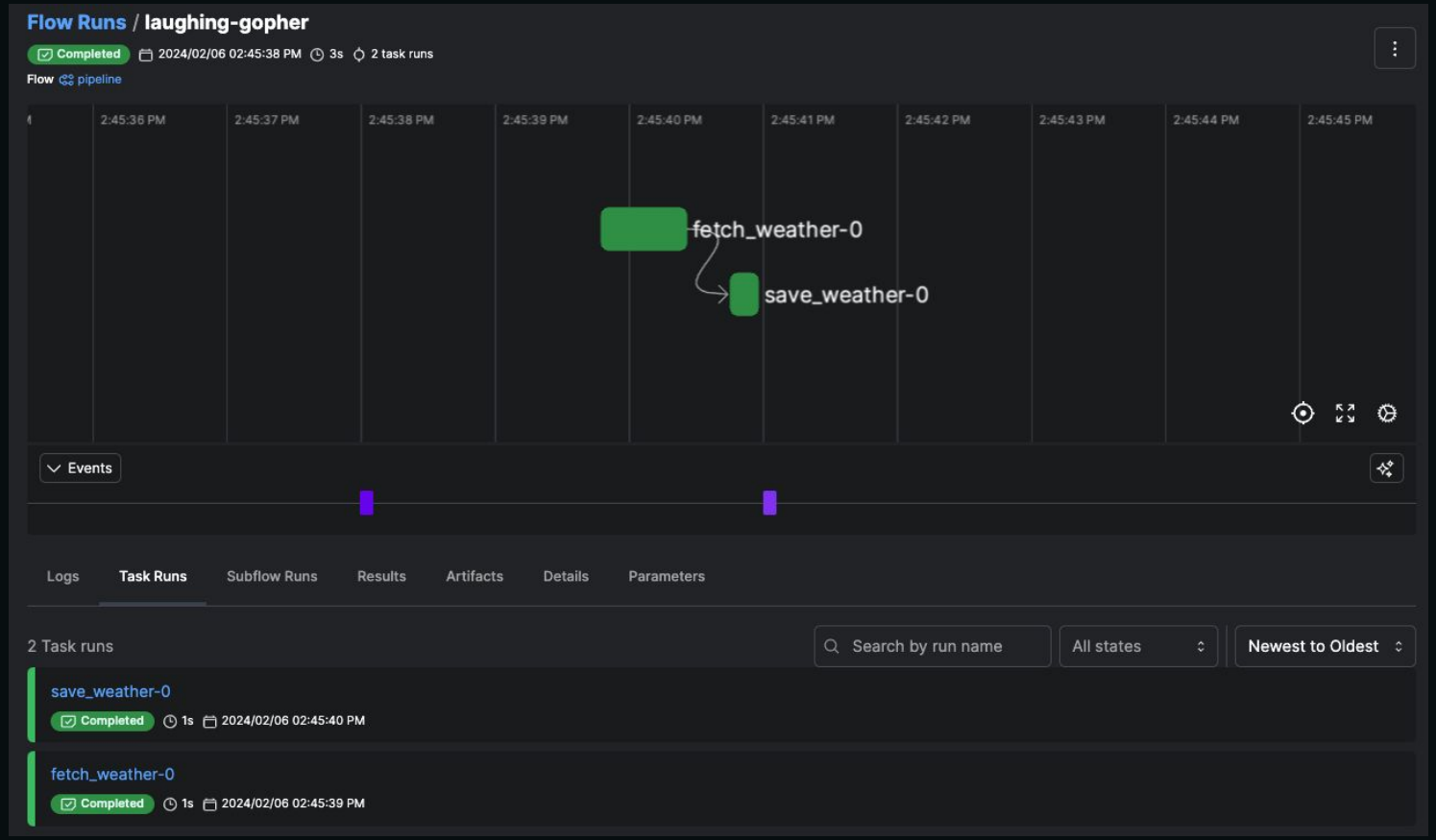
Pipeline flow function

```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

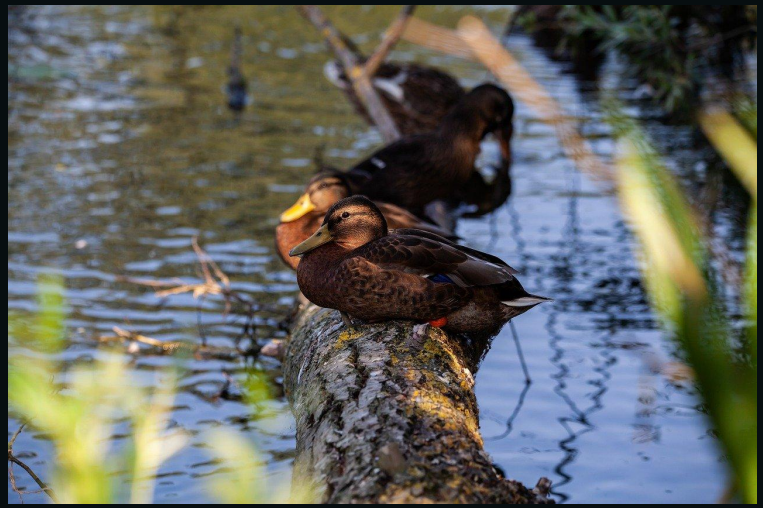


Tasks dos and don’ts

- ⛔ Don’t pass huge amounts of info between tasks
- ✅ Do keep tasks small

Note: Prefect is super Pythonic - conditionals are 👍

Logging



Log *print* statements with *log_prints*

@flow(log_prints=True)

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:

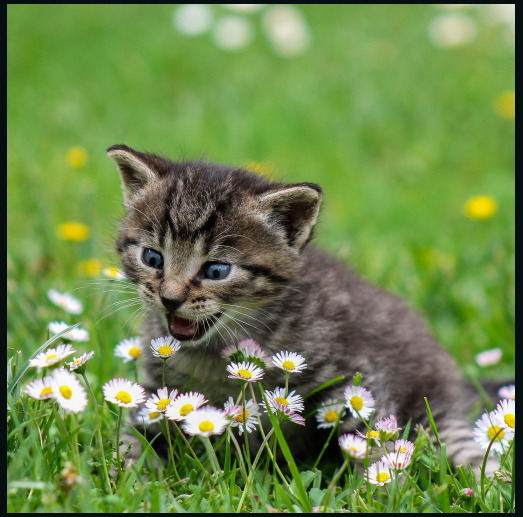
```
14:24:55.950 | INFO    | prefect.engine - Created flow run 'macho-sturgeon' for flow 'log-exa
mple-flow'
14:24:56.022 | INFO    | Flow run 'macho-sturgeon' - INFO level log message.
14:24:56.041 | INFO    | Flow run 'macho-sturgeon' - Finished in state Completed()
```


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()
```


Retries



Retries

Specify in task or a flow decorator

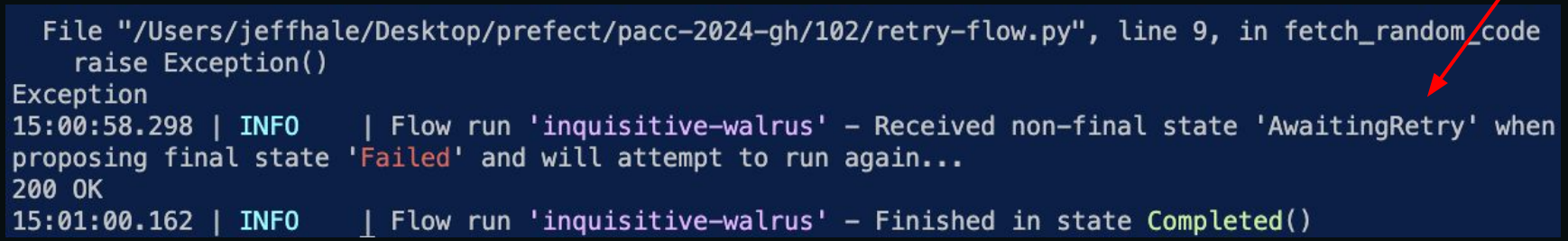
@task(retries=2)

@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



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*.

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.

Results

Results

The data returned by a flow or a task

```python
@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

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

Results

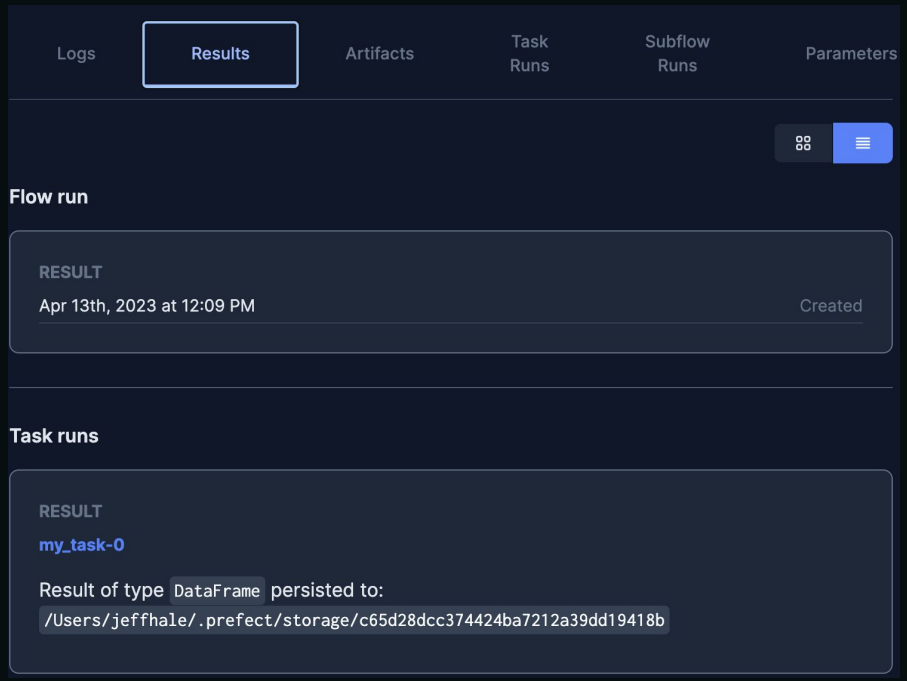
👆By default, Prefect returns a result that is ***not*** persisted to disk. It is only stored in memory.

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()
```


Results

Info **about** a result is viewable in the UI - **the result is not viewable**



Persisted results

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

Results - remote storage

Use a block (future topic) to store results in cloud provider storage

```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()
```


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)
```

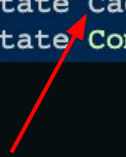

Caching

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.')
```



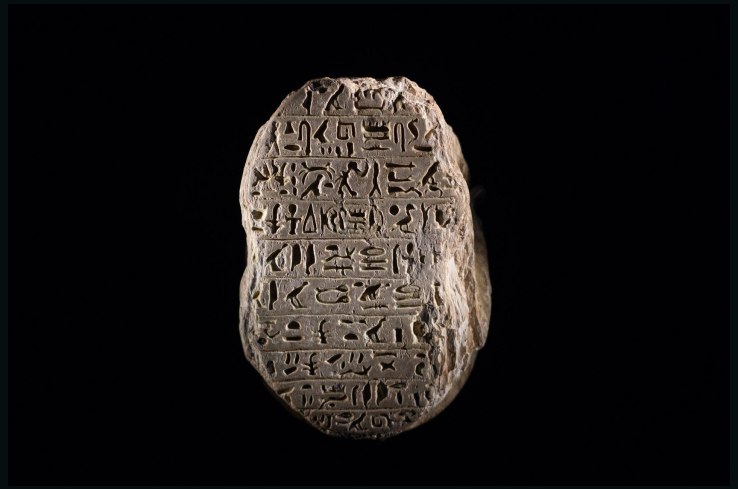
Caching: *cache_expiration* ⏳

```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 db

Artifacts - Markdown example

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

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

## Recent weather

| Time        | Revenue |
|:------------|-------:|
| Now | {temp} |
| In 1 hour       | {temp + 2} |
"""
    create_markdown_artifact(
        key="weather-report",
        markdown=markdown_report,
        description="Very scientific weather report",
    )
```


Artifacts - Markdown Example

Access from *Artifacts* page

(or *Flow Runs* if part of a flow run)

More helpful resources

Prefect CLI

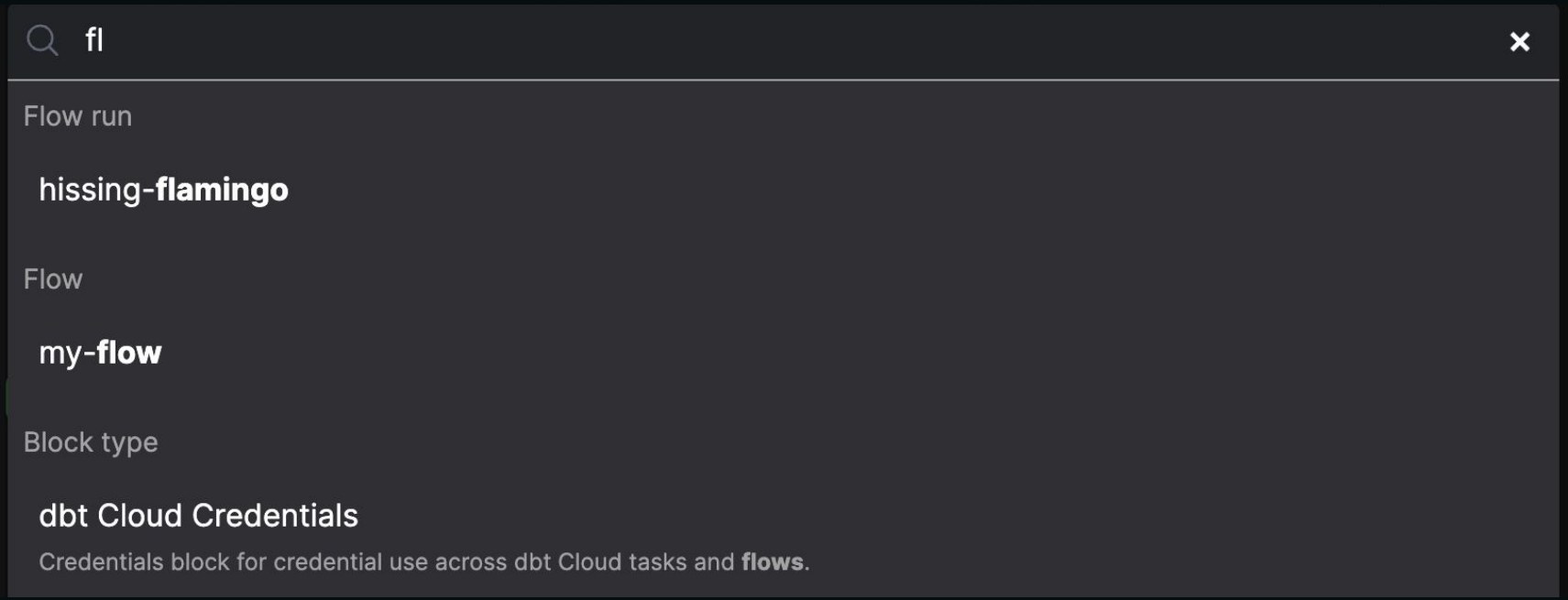
Start commands with *prefect --help* is always available

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 🔍



102 Recap

You've seen more of the power of Prefect.

- Tasks
- Logging
- States
- Retries
- Caching
- Results
- Artifacts
- More resources: *help* & search

Lab 102

Lab 102

- Use a flow that grabs weather data from open-meteo
- Add at least three tasks
- Add retries
- Run your flow
- Inspect in the UI
- Stretch: create an artifact
- Stretch: add caching

If you give an engineer a job…

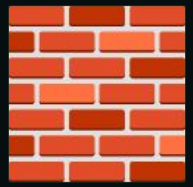
Could you just fetch this data and save it? Oh, and …

1. set up logging?
2. do it every hour?
3. visualize the dependencies?
4. automatically retry if it fails?
5. create an artifact for human viewing?
6. add caching?
7. add collaborators to run and view - who don't code?
8. send me a message when it succeeds?
9. run it in a Docker container-based environment?
10. pause for input?
11. automatically declare an incident when a % of workflows fail?
12. automatically run it in response to an event?

103 - Blocks & Cloud features

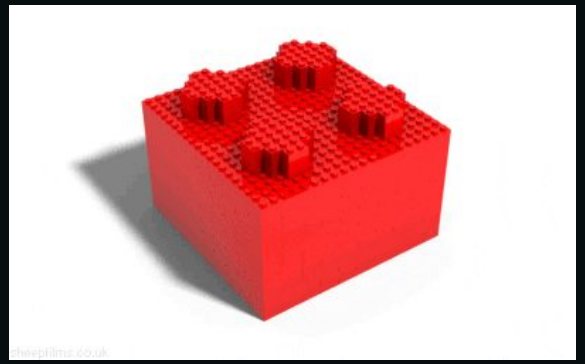
103 Agenda

- Blocks 
- Cloud features
- Automations
- Events

Blocks 

Blocks

Blocks are a cool Prefect feature



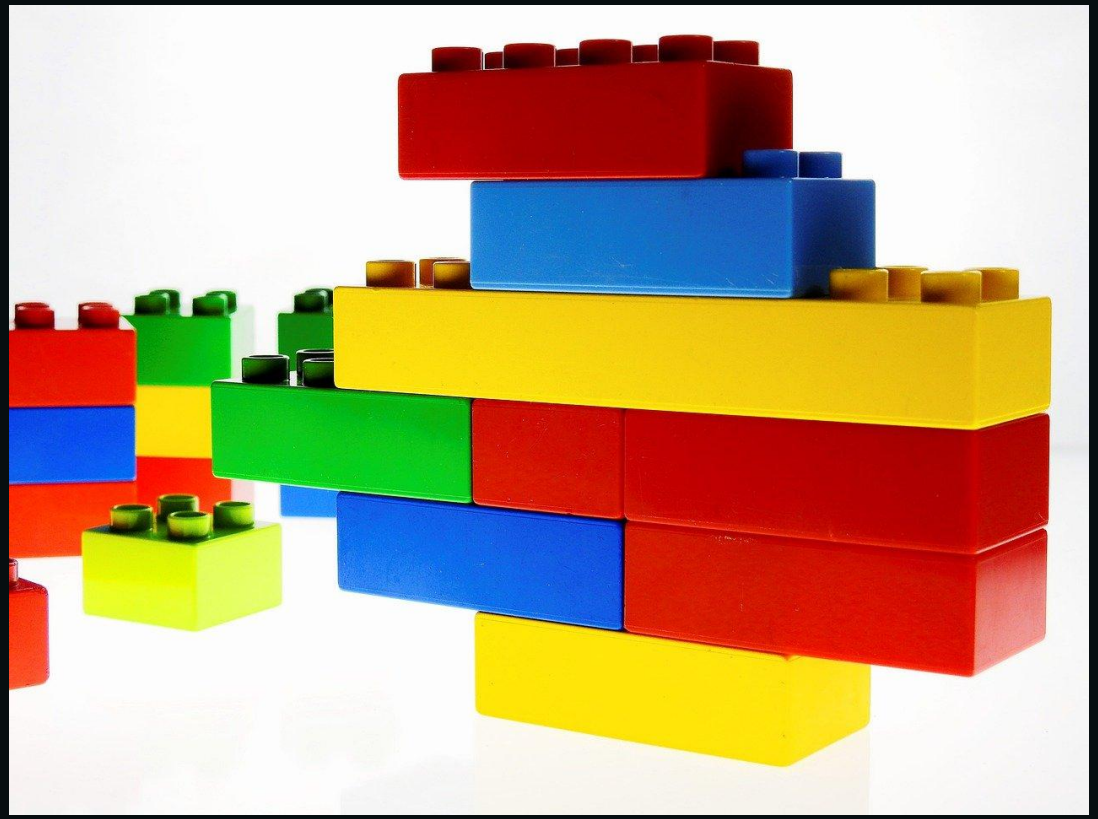
Available on Cloud and self-hosted

Blocks

Configuration

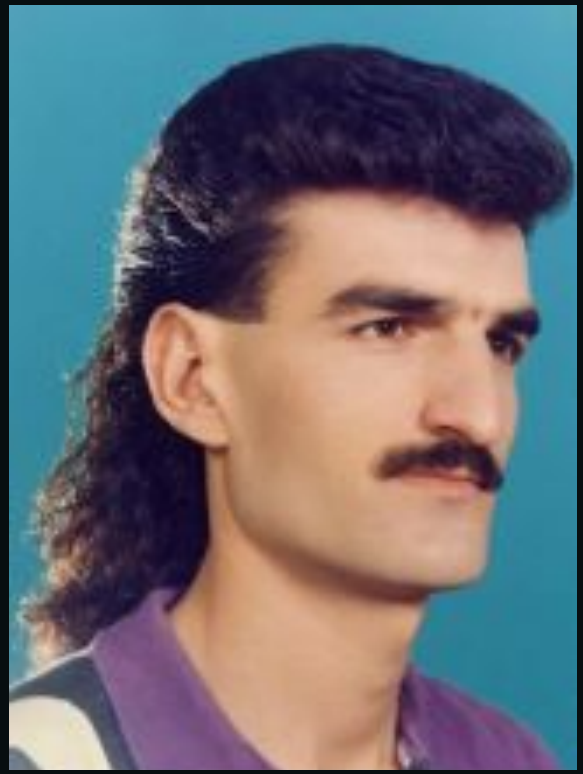
+

Code



Blocks

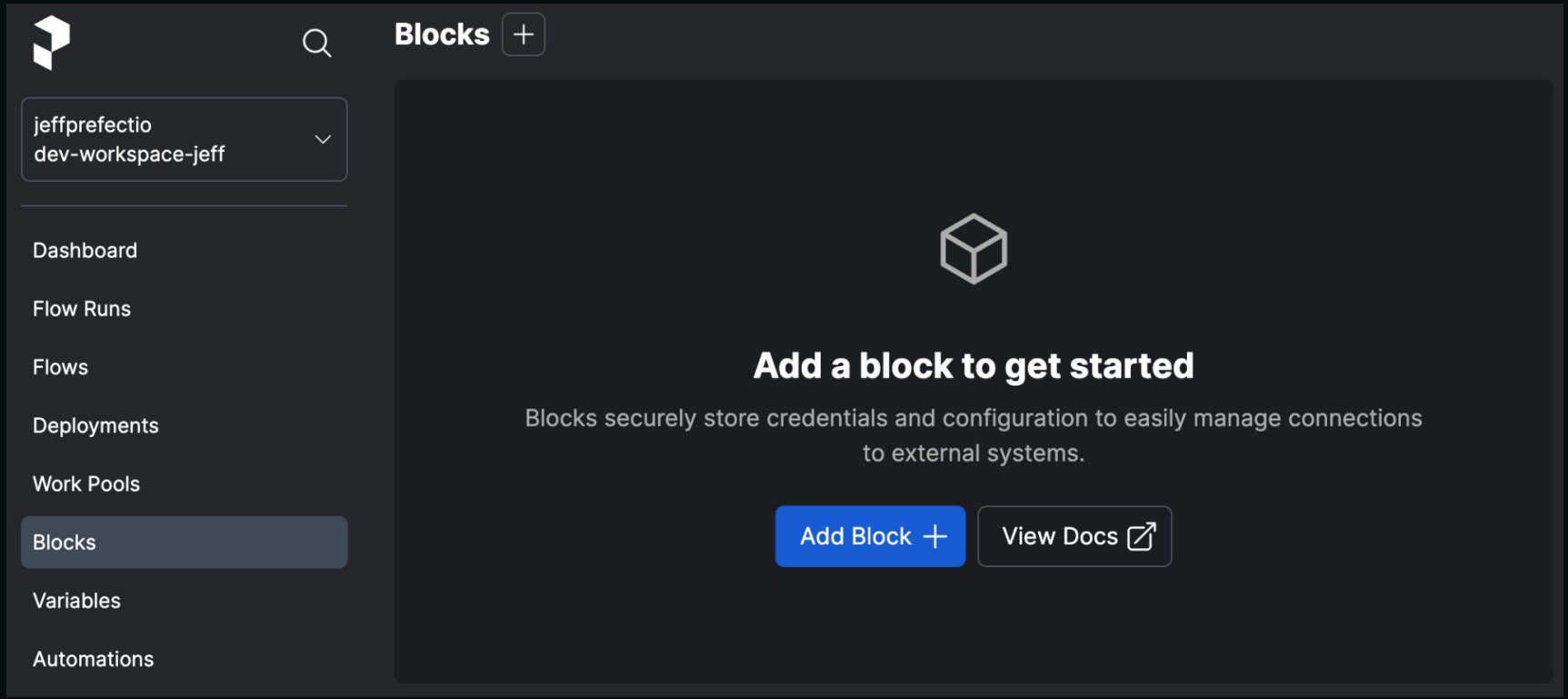
The Block mullet:



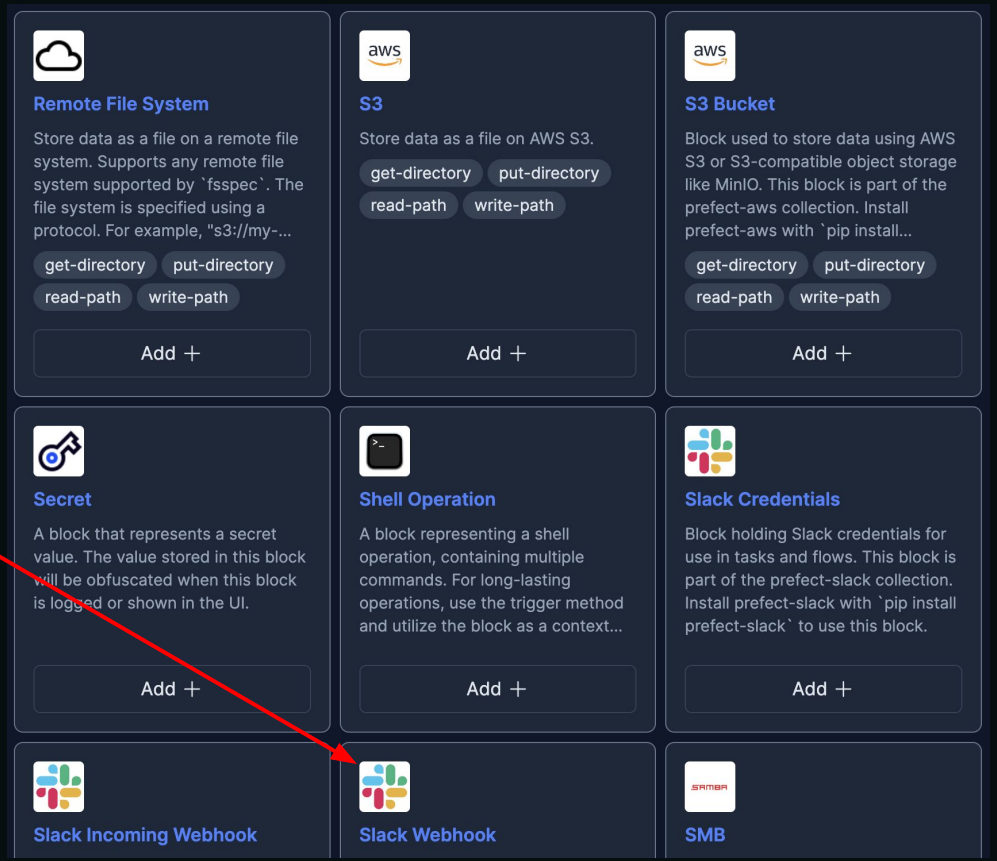
Structured form in front,

flexible code in back

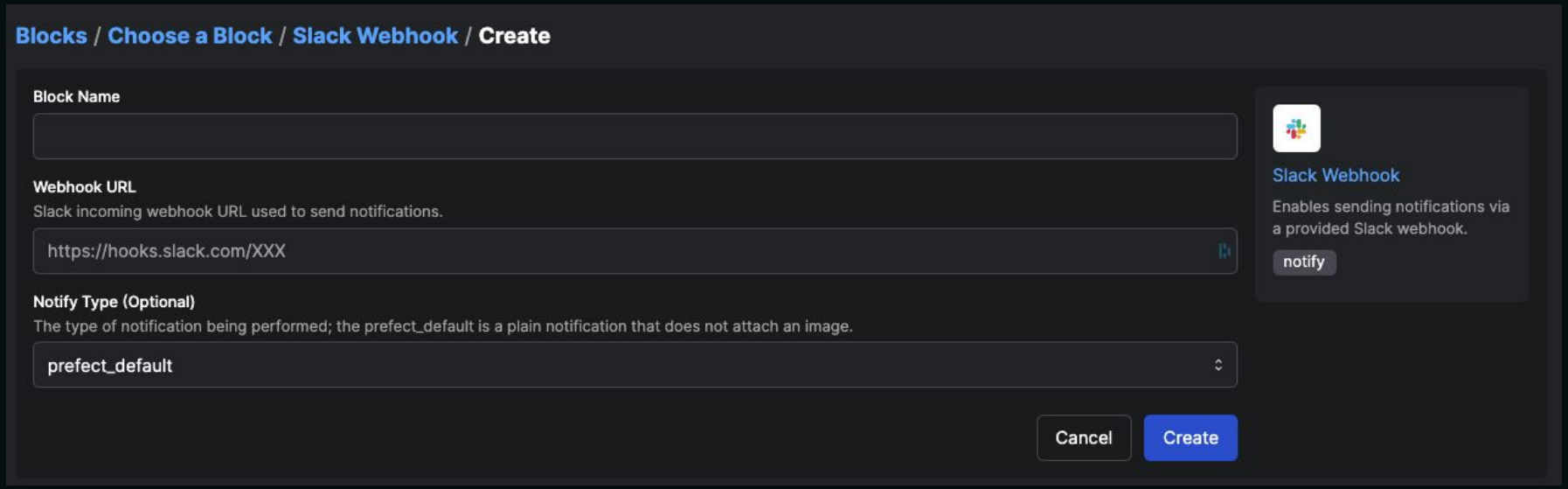
Create a Block from the UI



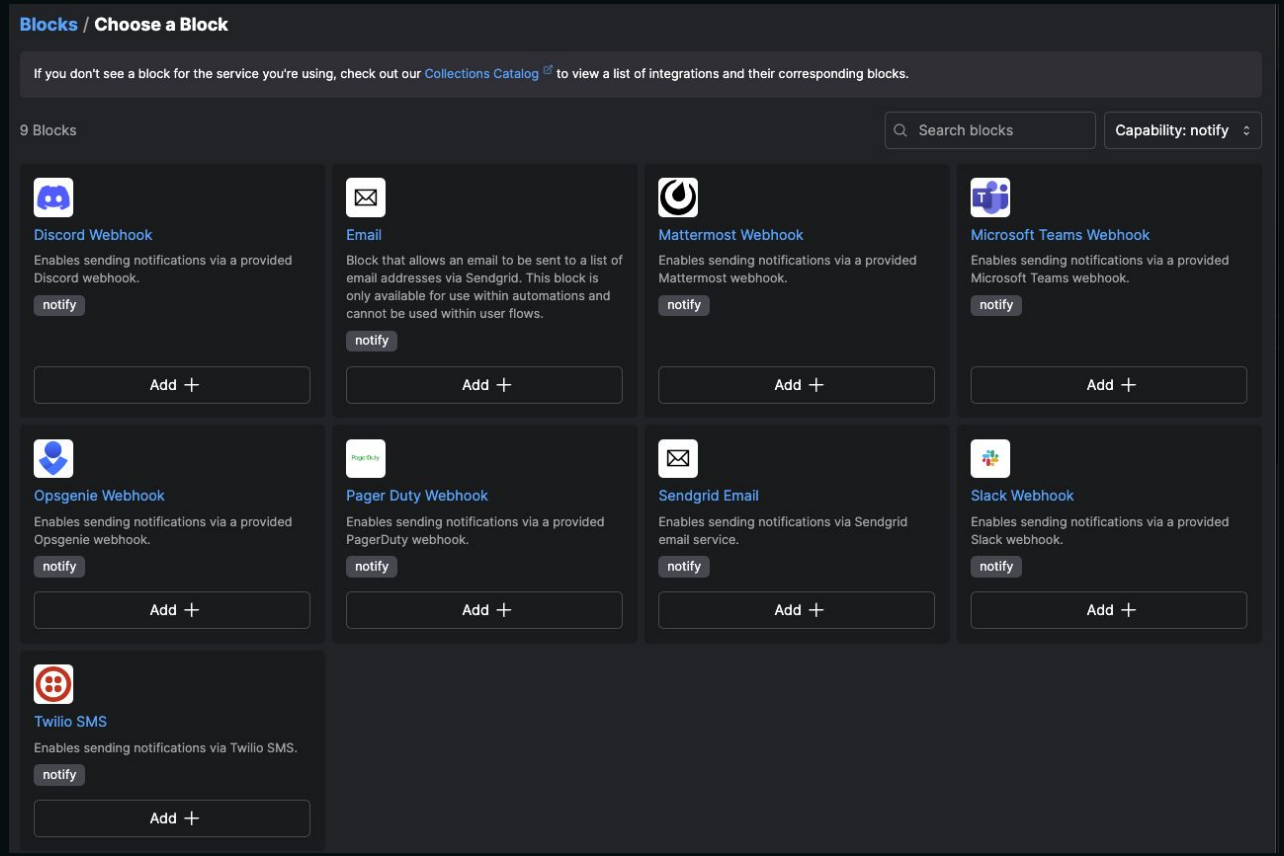
Create a block from the UI - choose a block type



Create a block from the UI

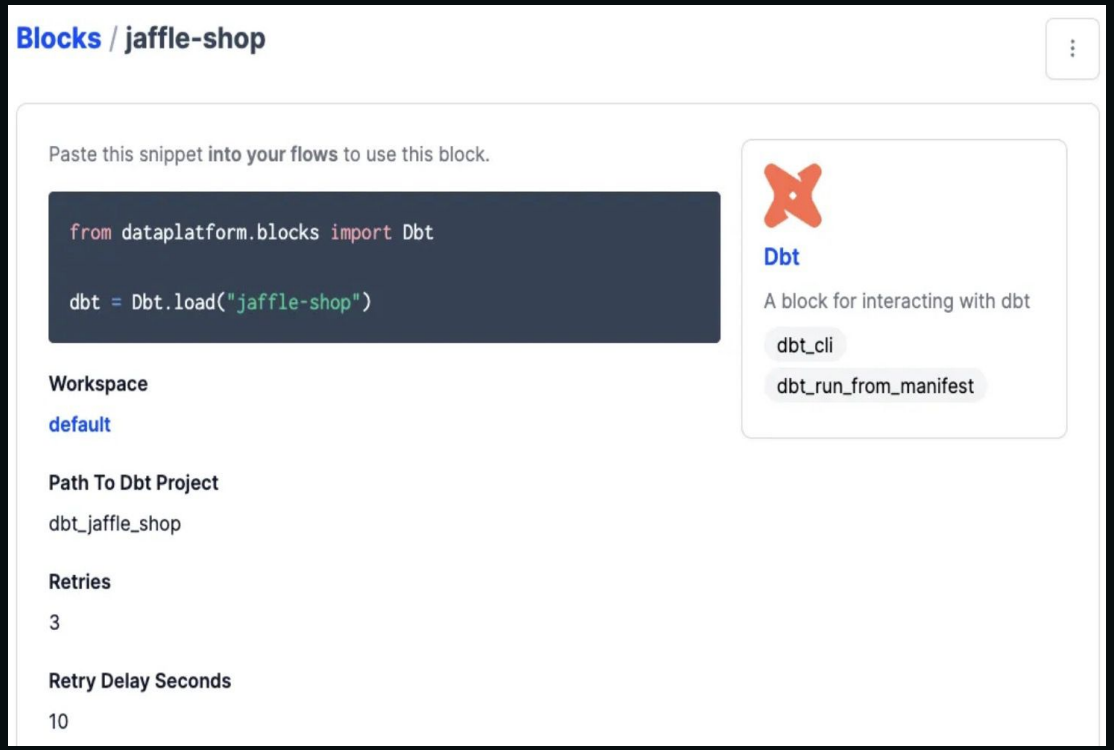


Block types in UI - filter by capability



Under the hood, block types are Python classes

Blocks are instances of those Python classes



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

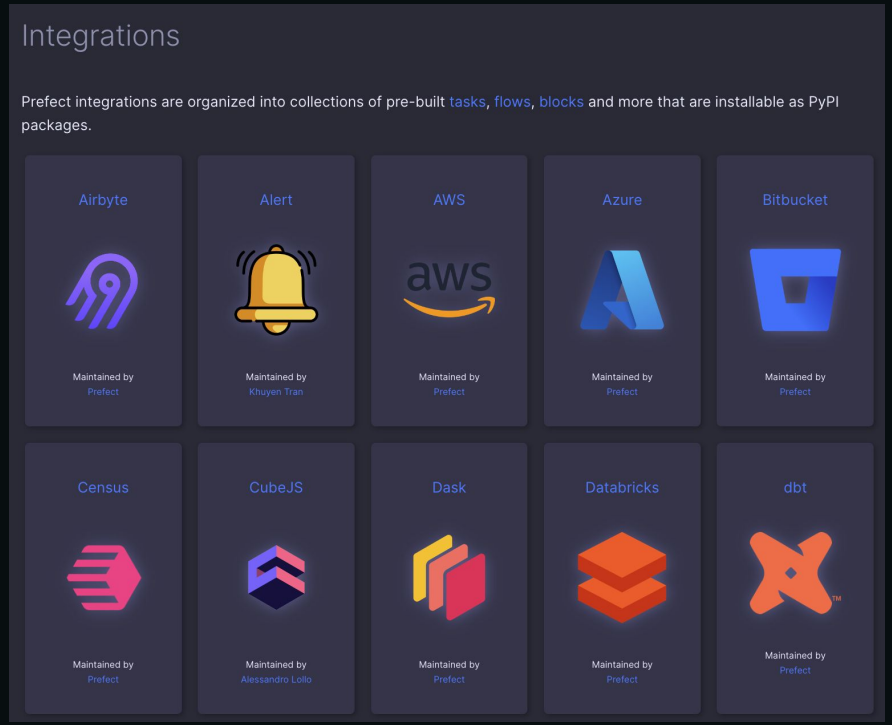
Reusable, modular, configuration + code

- Better than hard coding
- Nestable
- Stored in db
- Can create own types

Integrations

Integrations

docs.prefect.io/integrations/catalog/



Integrations

Python packages that add convenience

- Template to create your own
- Can contribute to the community
- Often add new block types you will register

Prefect Cloud

Prefect Cloud

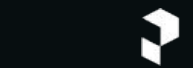
- Server is hosted by Prefect
- Workspaces
- Service Accounts
- RBAC
- SSO
- Automations
- Events

Prefect Cloud Workspaces

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

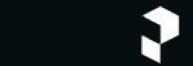
Prefect Cloud - Free Tier

- 2 free users
- 1 workspace
- 1 work pool
- 7 day flow run history



Prefect Cloud - Pro Tier

- Service accounts
- RBAC
- 30-day flow run history
- 72-hour audit log
- Higher rate limits
- More work pools
- More automations



Prefect Cloud - Custom Tier

- SSO & SCIM
- Custom roles
- Object access control lists
- Custom most everything 🙂

Prefect Cloud

Prefect Cloud - Default Roles (Pro + Custom)

Account level

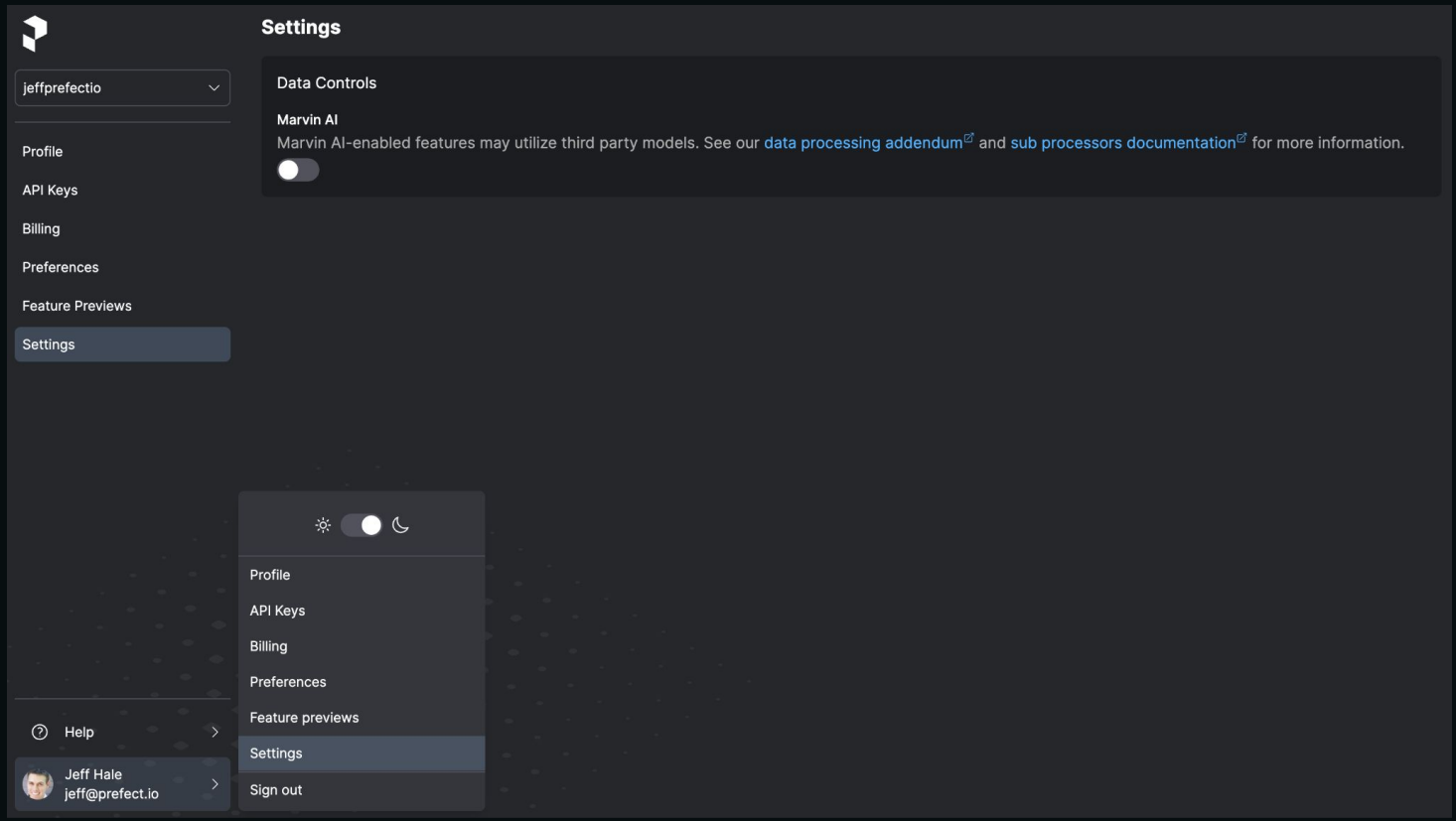
- Owner
- Admin
- Member

Workspace level

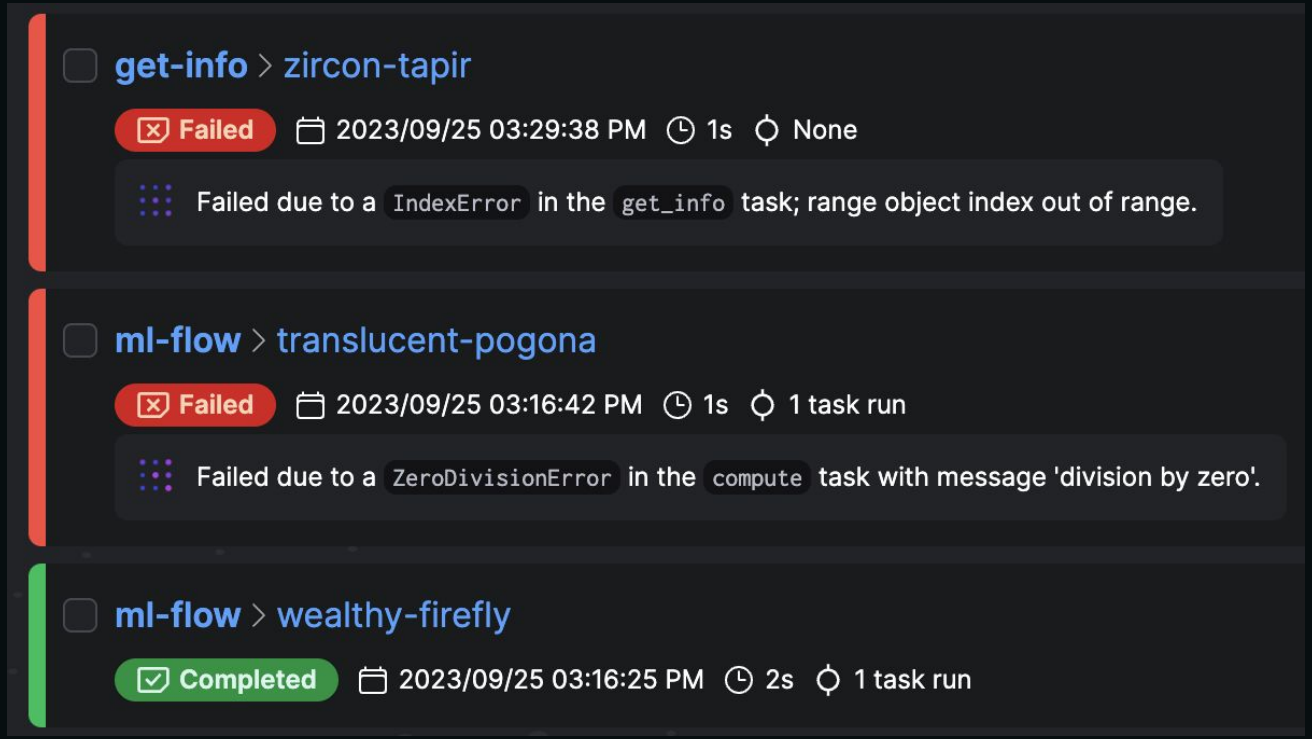
- Owner
- Developer
- Runner
- Viewer
- Worker

Error summaries by Marvin AI

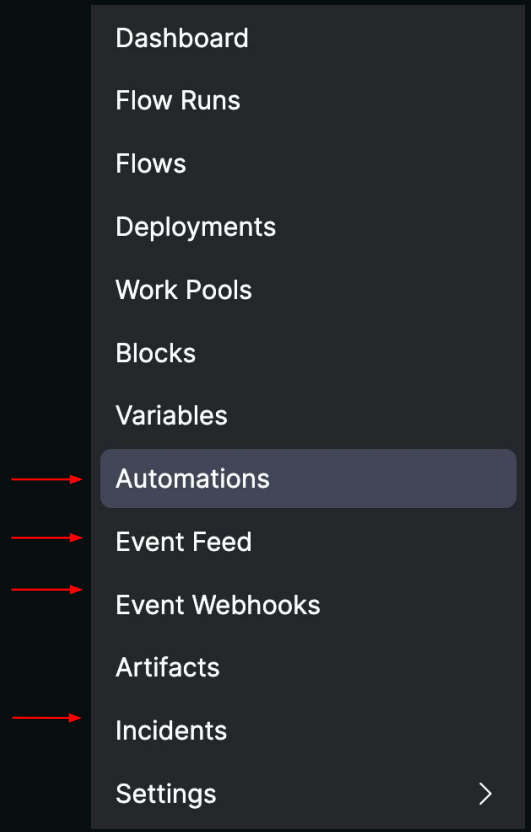
Error summaries by Marvin AI



Error summaries by Marvin AI



Cloud features: automations, events API, incidents



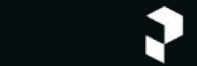
Events

Events

- A record of what has happened
- A notification of a change

Represent:

- API calls
- State transitions
- Changes in environment



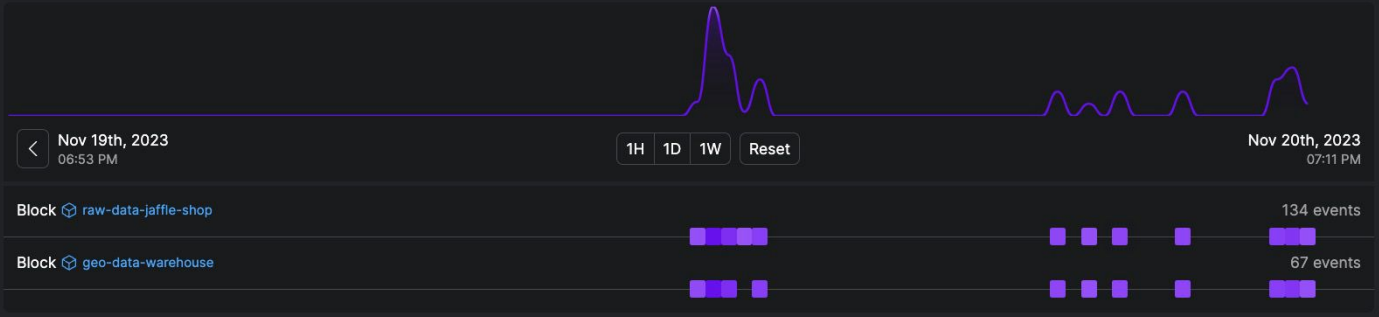
Workspace Events

Resource

All resources

Events

prefect.block.s3-bucket.* × prefect.block.snowflake-connection.* ×



06:05:16 PM
Nov 20th, 2023

Block snowflake connection read sql called

prefect.block.snowflake-connection.read_sql.called

Resource

Block geo-data-warehouse

Related Resources

prefect.block-type.snowflake-connection Flow run liberal-mule Task run get_geographical_data-0 Flow data-cleaning-flow Deployment k8s-deployment

Work queue default Work pool my-k8s-pool

06:05:15 PM
Nov 20th, 2023

Block snowflake connection load called

prefect.block.snowflake-connection.load.called

Resource

Block geo-data-warehouse

Related Resources

prefect.block-type.snowflake-connection Flow run liberal-mule Task run get_geographical_data-0 Flow data-cleaning-flow Deployment k8s-deployment

Work queue default Work pool my-k8s-pool

06:05:09 PM
Nov 20th, 2023

Block s3 bucket read path called

prefect.block.s3-bucket.read_path.called

Resource

Block raw-data-jaffle-shop

Related Resources

prefect.block-type.s3-bucket Flow run wild-inchworm Task run read_csv_to_df-1 Flow load-in-historical-data

Event Feed


Workspace Events
Beta
Block events are in Beta. To gather events from blocks, including storage and infrastructure events, enable client side events with prefect config set PREFECT_EXPERIMENTAL_ENABLE_EVENTS_CLIENT='True'
Related Resource
All resources
Events
All events
Mar 19th, 06:10 PM - Mar 22nd, 10:48 PM
Mar 17th, 2023
12:00 AM
1H
1D
1W
Mar 21st, 2023
06:32 PM
Mar 23rd, 2023
11:59 PM
Block Document collection-registry-github-token
5426 events
Block Document marvin-monday-slack-posts-k8s-job
1683 events
Block Document pd-oncall-slack-update-k8s-job
819 events
Block Document closed-won-k8s-job
756 events
Block Document collection-registry-update-job
460 events
Block Document collection-registry-result-storage
93 events
See More
10:30:52 PM
Mar 22nd, 2023
Flow run scheduled
prefect.flow-run.Scheduled
Resource
Flow run steady-ocelot
Related Resources
Flow Monday Flow
Deployment Marvin Monday Slack Posts
Work Queue internal-tools-cluster
1 other resource
10:30:28 PM
Mar 22nd, 2023
Block kubernetes job get client called
prefect.block.kubernetes-job.get_client.called
Resource
Block Document marvin-monday-slack-posts-k8s-job
Related Resources
1 resource
10:30:26 PM
Mar 22nd, 2023
Block kubernetes job get batch client called
prefect.block.kubernetes-job.get_batch_client.called
Resource
Block Document marvin-monday-slack-posts-k8s-job
Related Resources
1 resource
10:30:24 PM
Flow run completed

Events

Power several Cloud features:

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

Automations ⚡

Automations

Cloud only

Flexible framework

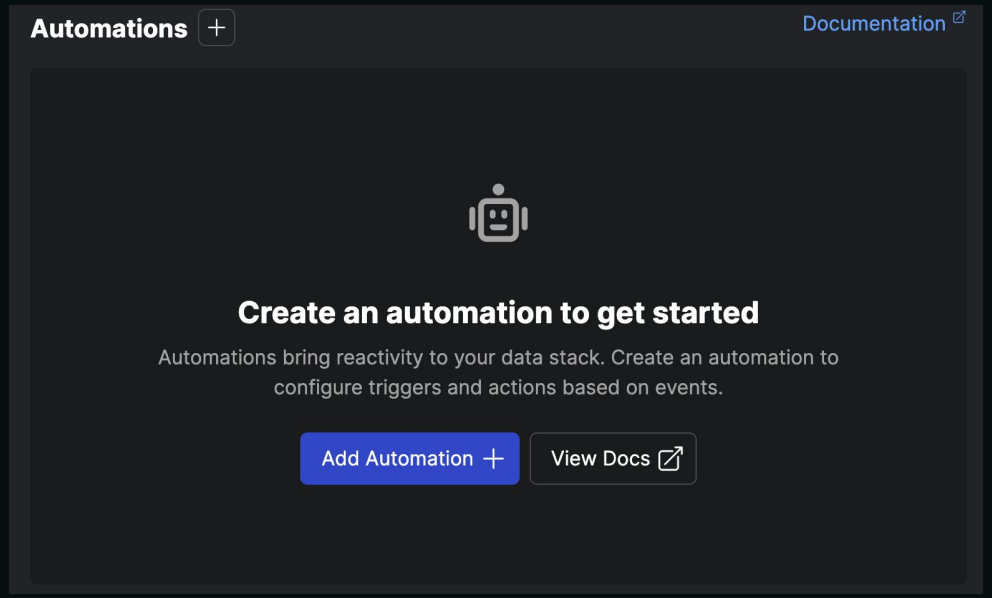
- If *Trigger* happens, do *Action*
- If *Trigger* doesn't happen in a time period, do *Action*

Automation examples

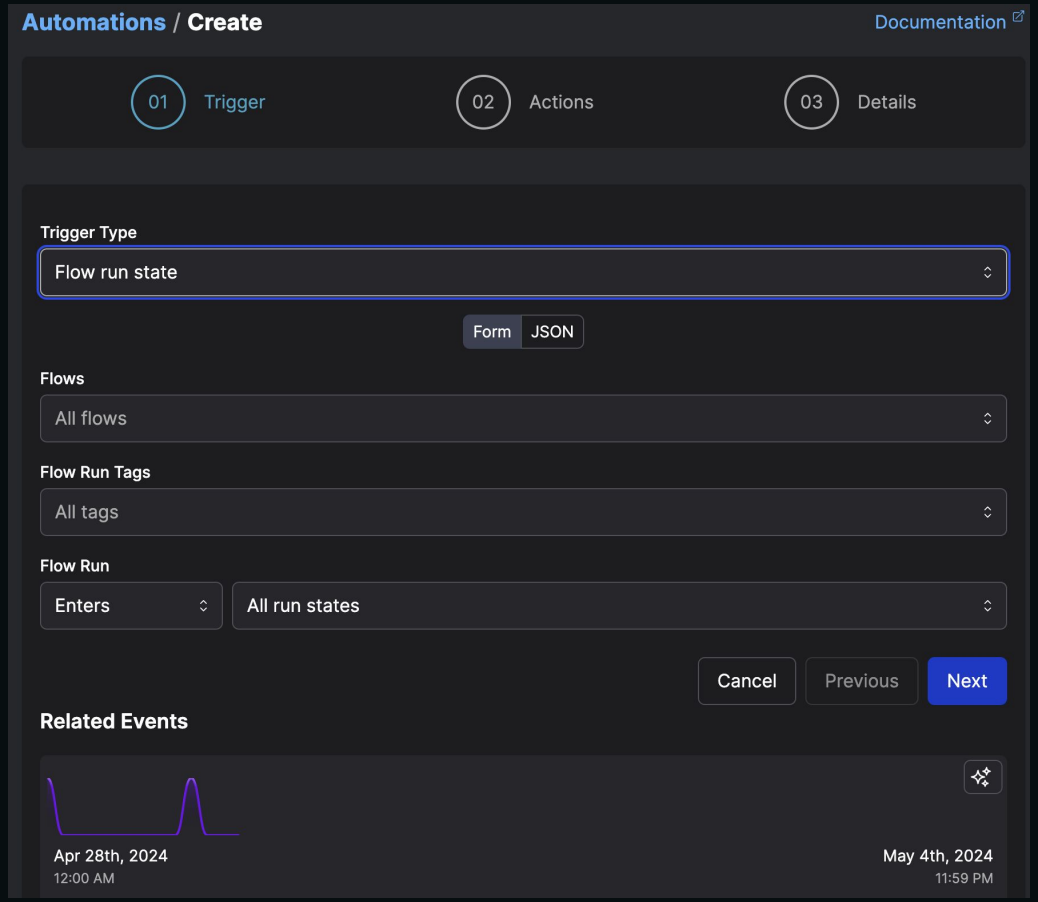
- If a flow run with tag **prod** fails, send an email 📧
- If a data quality check fails, run a deployment to fetch more data 📊
- If a work pool changes state to *Not Ready*, create an incident 🚨

Create an automation

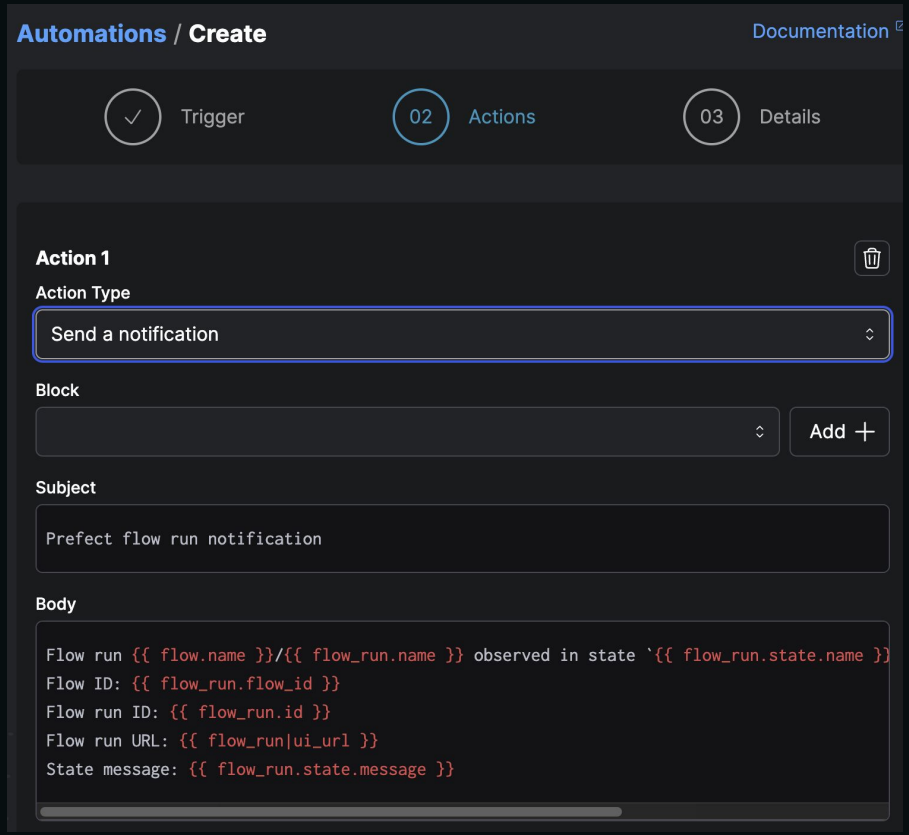
Trigger: flow run failure
Action: notification - email



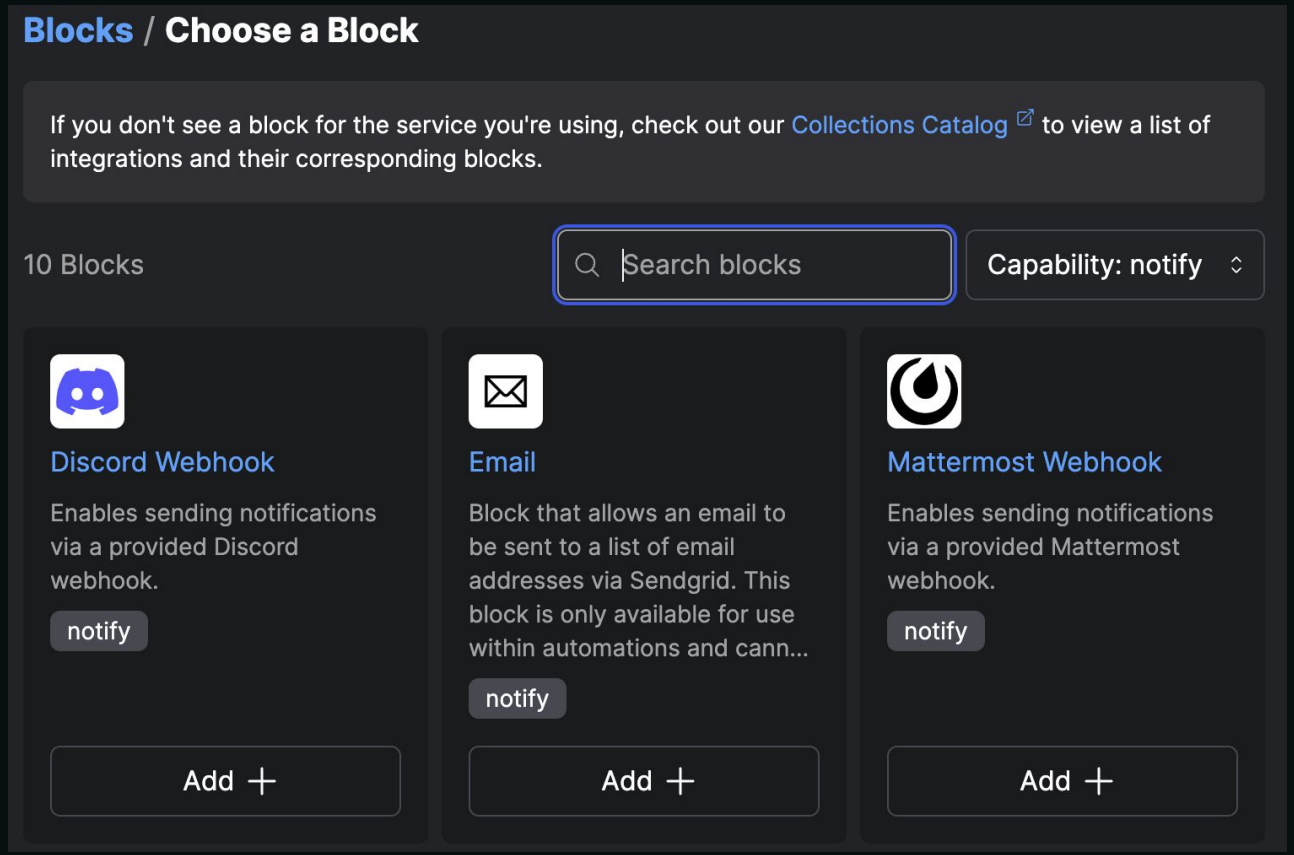
Automation trigger



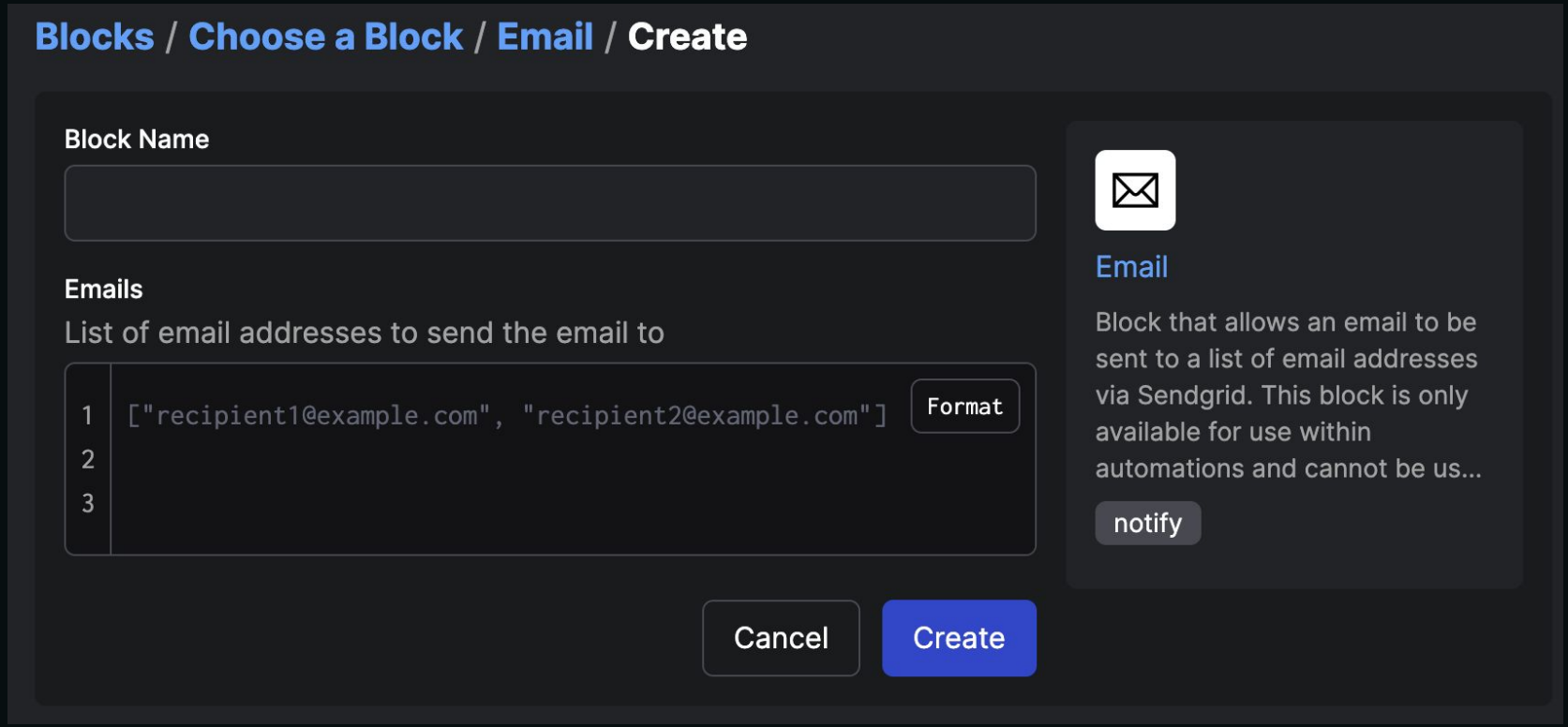
Automation action



Create a block with **notify** capability



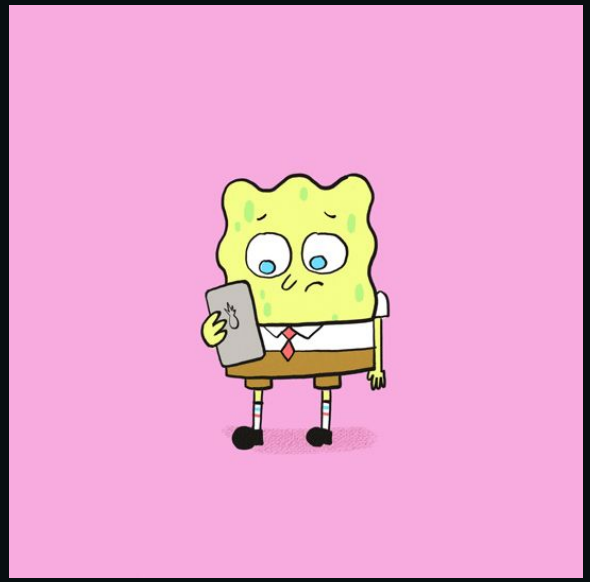
Create an **Email** block



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

- Blocks
- Integrations
- Prefect Cloud features
- Error summaries by Marvin AI
- Events
- Automations

Lab 103

103 Lab

- Make an email notification automation for a flow run completion
  - ❗ use an **Email** block type
- Run a flow a few times from the CLI
- See the event feed in the UI
- Stretch: create an automation that filters by a flow run tag - set the tag in your deployment

If you give an engineer a job…

Could you just fetch this data and save it? Oh, and …

1. set up logging?
2. do it every hour?
3. visualize the dependencies?
4. automatically retry if it fails?
5. create an artifact for human viewing?
6. add caching?
7. add collaborators to run and view - who don't code?
8. send me a message when it succeeds?
9. run it in a Docker container-based environment?
10. pause for input?
11. automatically declare an incident when a % of workflows fail?
12. automatically run it in response to an event?