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

Introductions

Norms

Zoom

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

Slack

- Use threads
- Emoji responses 🙂

Overview

Introductions

- Name
- Pronouns
- Where you’re coming from
- Popcorn to next person

Goals

Goals

1. Competence with Prefect 2
2. Connect with each other
3. Have fun! 🎉

If you give an engineer a job…

Could you

1. just fetch this data and save it?
2. set up logging?
3. visualize the dependencies?
4. automatically retry if it fails?
5. create an artifact for human consumption?
6. add caching?
7. add collaborators to run and view - who don’t code?
8. send me a message when it succeeds?
9. fetch the code from a remote git repository?
10. do it every hour?
11. automatically run it in response to a webhook?
12. run it in a Docker container-based environment?

What is Prefect?

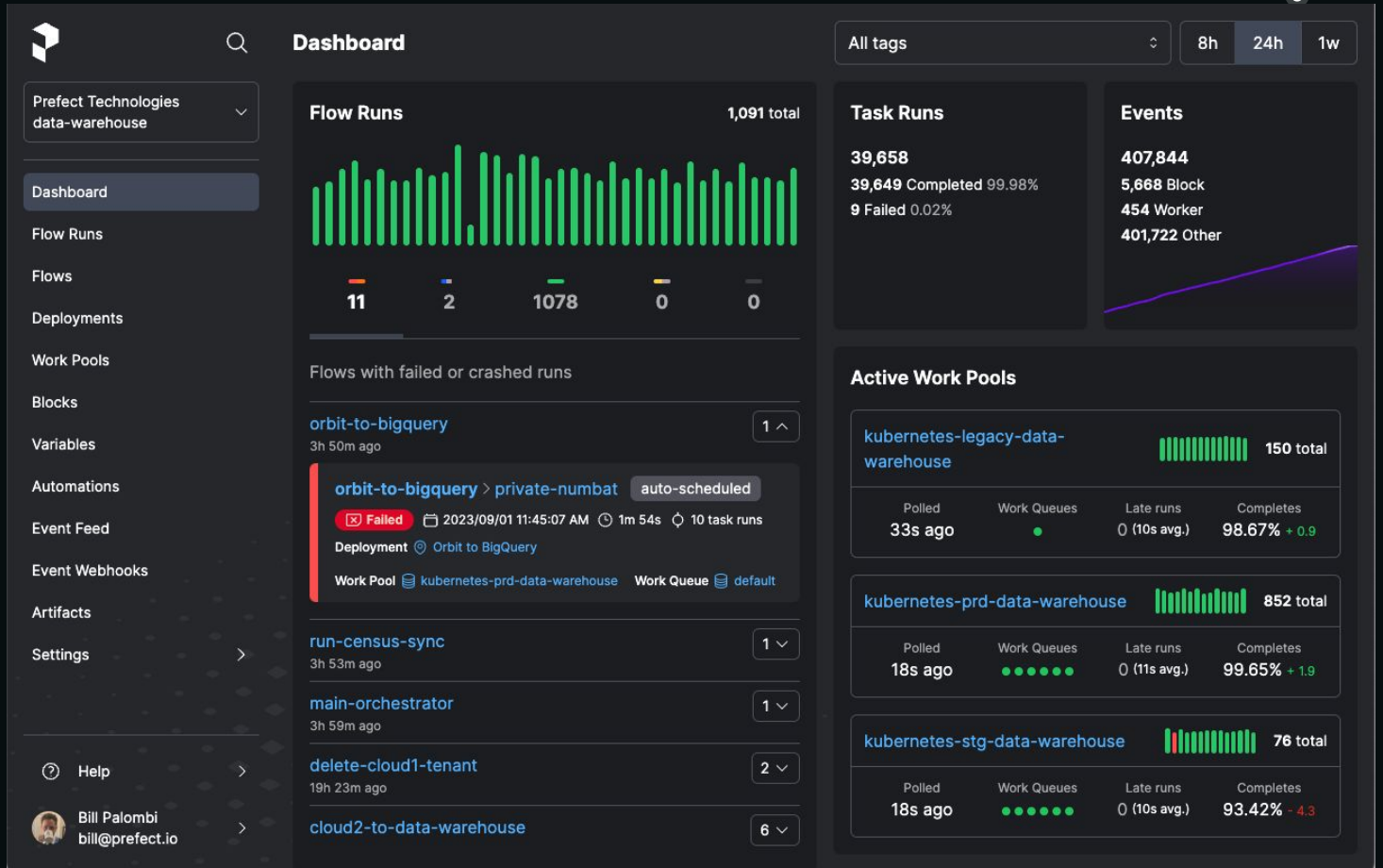
Prefect is a workflow orchestration tool empowering developers to build, observe, and react to data pipelines

Prefect helps data teams be **more efficient and effective**

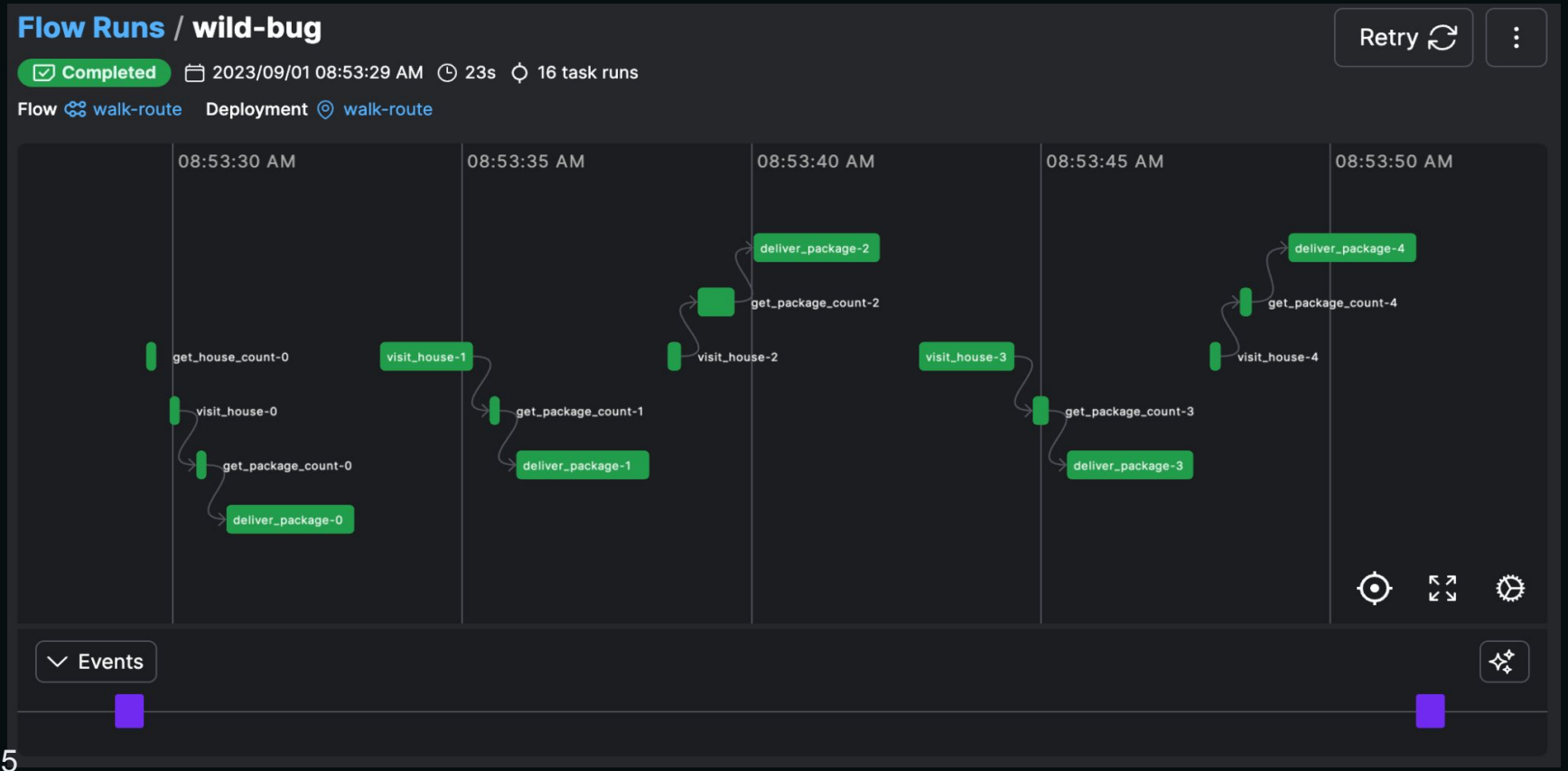
Prefect helps teams:

Develop Faster	Reduce Failures	Increase Visibility
How:		
• Intuitive Python-based library • Caching • Integrations • World-class support	• Clear & maintainable code • Test convenience • Easy async • Automatic retries	• UI • Notifications • Event feed • Collaboration

Understand the state of your workflows



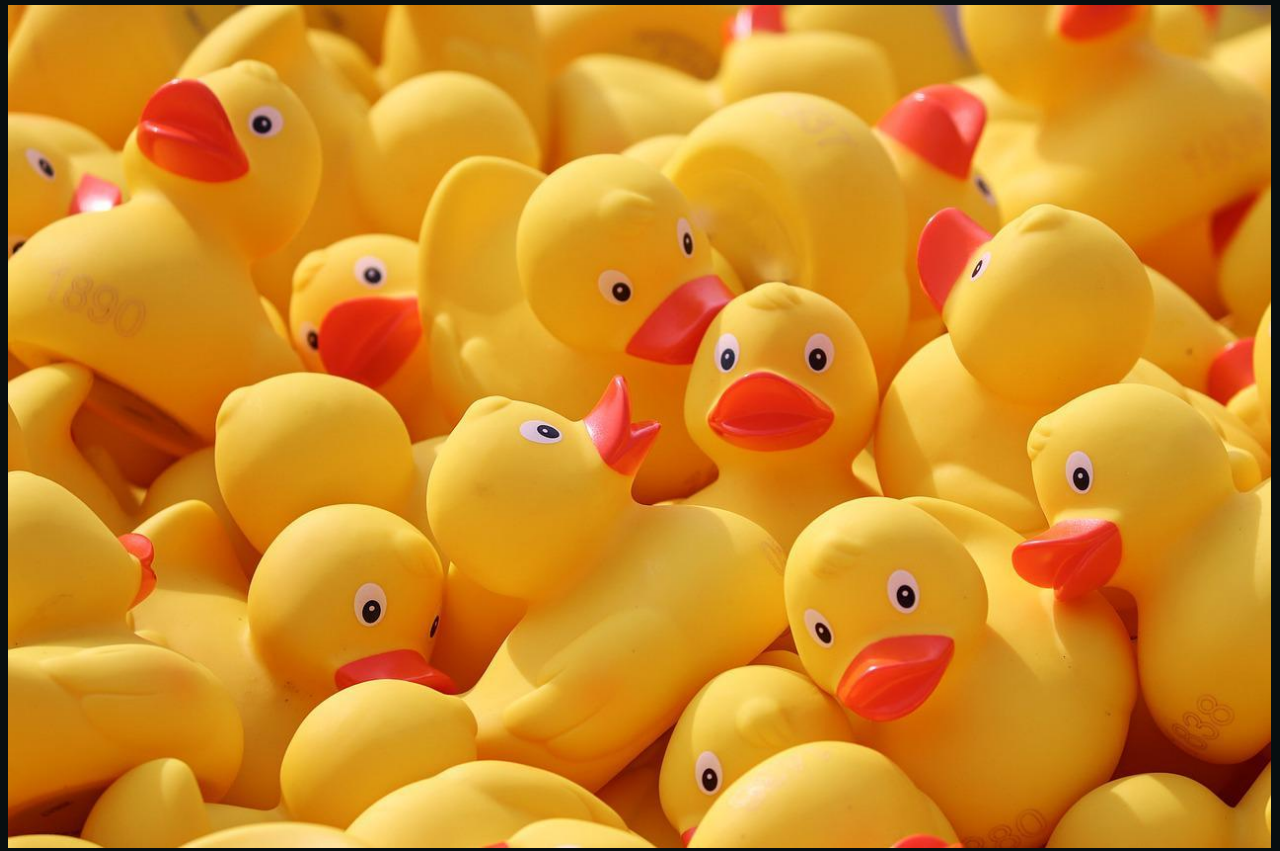
Orchestrate and observe Python code



Workflow orchestration is the coordination of tasks across infrastructure, systems, triggers, and teams.

Sarah Krasnik Bedell

Welcome to PACC!



Meet Minerva

Minerva does data things for the Quackme Co. consulting firm.

- Weather forecasts 🌦️
- Cat facts 🐈

Your charge: help Minerva orchestrate workflows with Prefect!

MODULE

101 - Prefect basics

101 Agenda

- *prefect version*
- Prefect profiles
- From Python function to Prefect flow
- Deployments with *.serve()*
- Tasks
- Resources

prefect version

Prefect information from the CLI

prefect version

```
Version:              2.13.2
API version:          0.8.4
Python version:       3.10.12
Git commit:           dcd8a58f
Built:                Thu, Sep 21, 2023 2:33 PM
OS/Arch:              darwin/arm64
Profile:              local
Server type:          ephemeral
Server:
  Database:           sqlite
  SQLite version:     3.43.0
```


Run *prefect version* now

If you see *version* lower than *2.13.5*

pip install -U prefect

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-forever tier
    b. Organization management capabilities on other tiers
    c. Additional features
    d. No db management

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
- One profile active at all times
- Common to switch between:
  - Cloud and a self-hosted Prefect server
  - Different Cloud workspaces
  - Different 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 Profiles

If you don’t already have a profile with Prefect Cloud you want to use for this course, create a new profile

Create: *prefect profile create my_cloud_profile*

Prefect Profiles

Inspect: *prefect profile inspect my_cloud_profile*

Select: *prefect profile use my_cloud_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
```

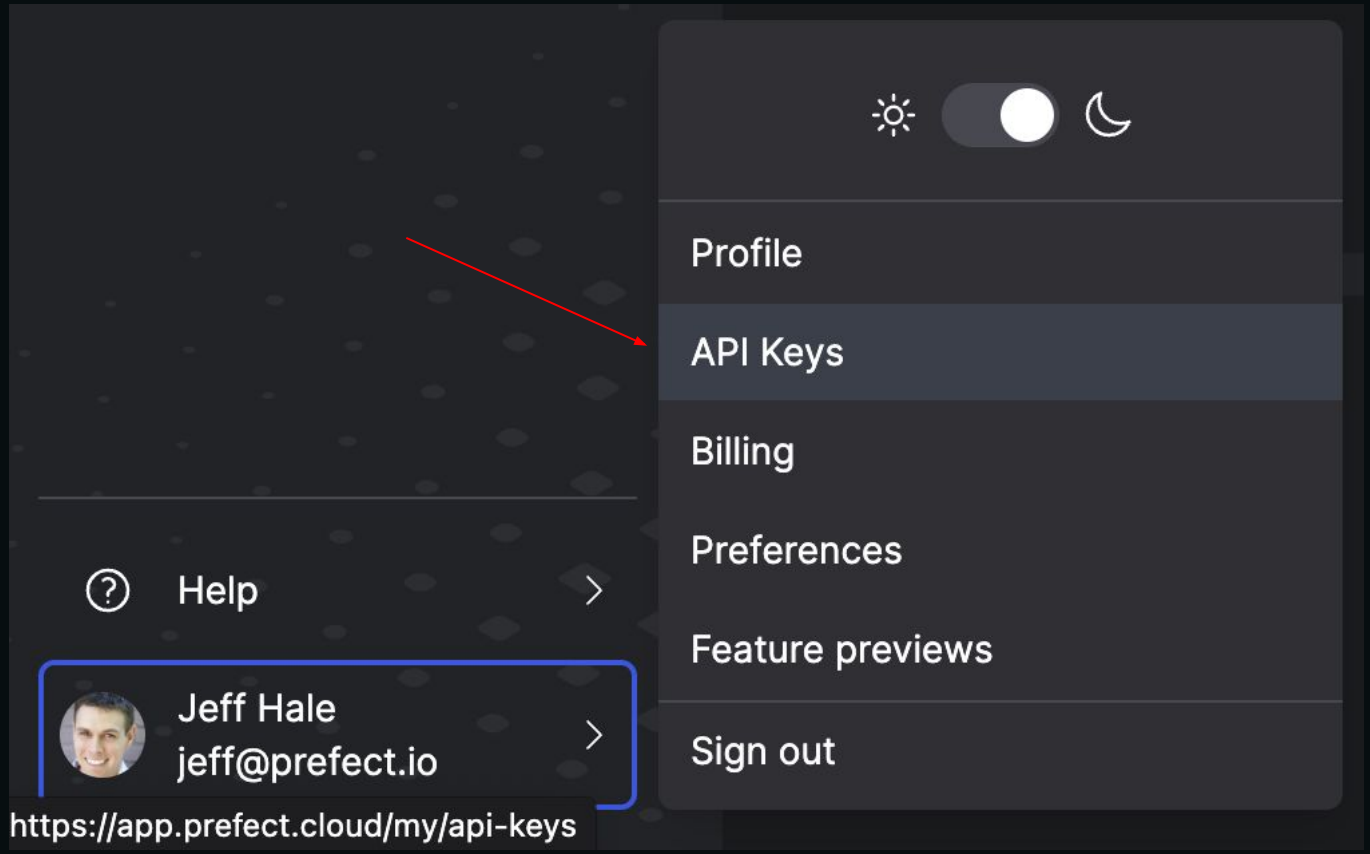
Select ***Log in with a web browser***

Prefect Cloud

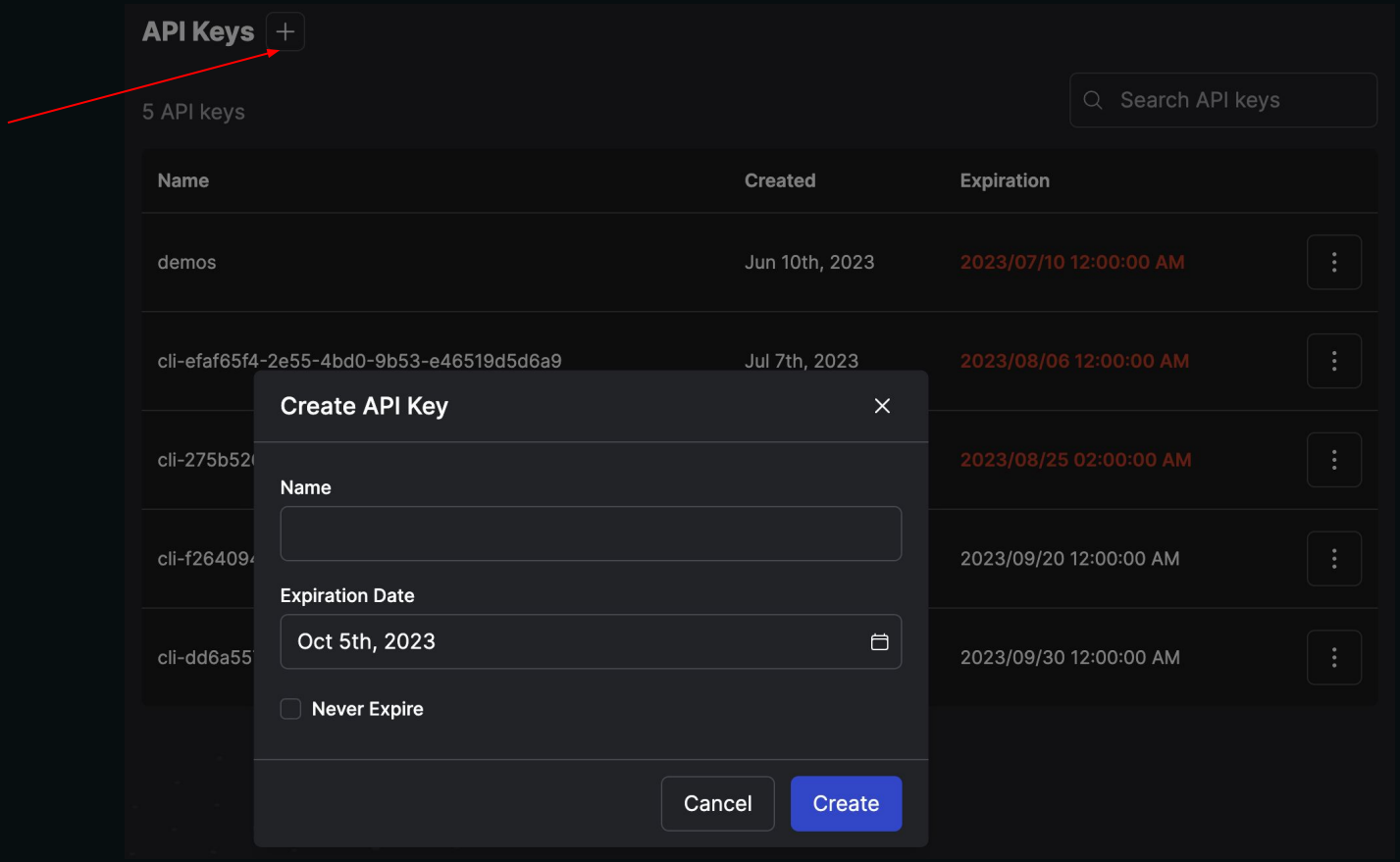
Or paste an API key

Manually create an API key from Prefect Cloud in the UI

Prefect Cloud - API Key



Prefect Cloud - API Key



From Python function to Prefect flow

Initial project

Help Minerva get some weather data. 🌦️

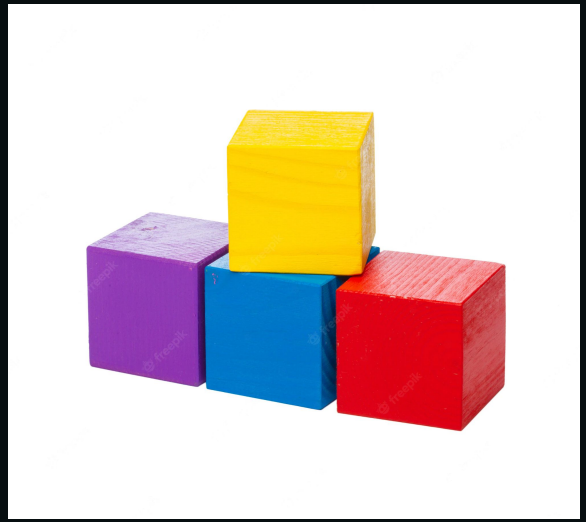
open-meteo.com/en

Fetch data: 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/"
    weather = httpx.get(
        base_url,
        params=dict(latitude=lat, longitude=lon, hourly="temperature_2m"),
    )
    most_recent_temp = float(weather.json()["hourly"]["temperature_2m"][0])
    print(f"Most recent temp C: {most_recent_temp} degrees")
    return most_recent_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

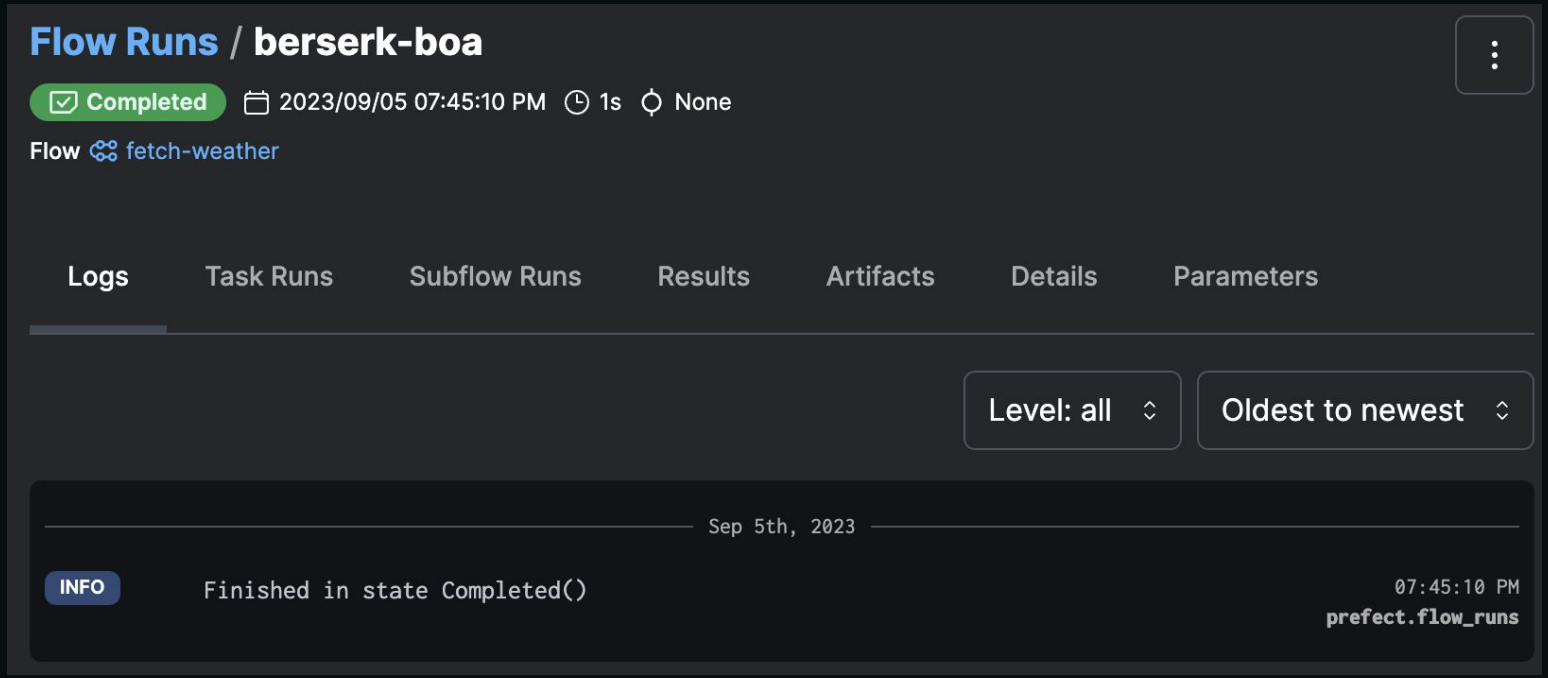
```python
import httpx
from prefect import flow

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

Run the file *python my_file.py*

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

```
19:45:09.631 | INFO    | prefect.engine - Created flow run 'berserk-boa' for flow 'fetch-weather'
19:45:09.633 | INFO    | Flow run 'berserk-boa' - View at https://app.prefect.cloud/account/55c7f5e5-2da9
-426c-8123-2948d5e5d94b/workspace/7ad1ef2f-2f9c-49b5-b29f-4e0b3760d4c6/flow-runs/flow-run/4fd2c641-4f13-4
1f7-b45c-722b246a3084
Most recent temp C: 31.9 degrees
```

Check it out 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

Deployment

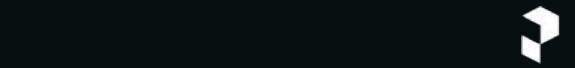
- Server-side representation of a flow
- Contains meta-data for remote orchestration
- Can be run on various infrastructure
- Can be kicked off
  - manually (from the UI or CLI)
  - on a schedule
  - automatically, in response to an event trigger

.serve() method

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

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

⚠️ This won't work in older versions of Prefect



.serve() method

Run the script.

Creates a deployment and starts a server.

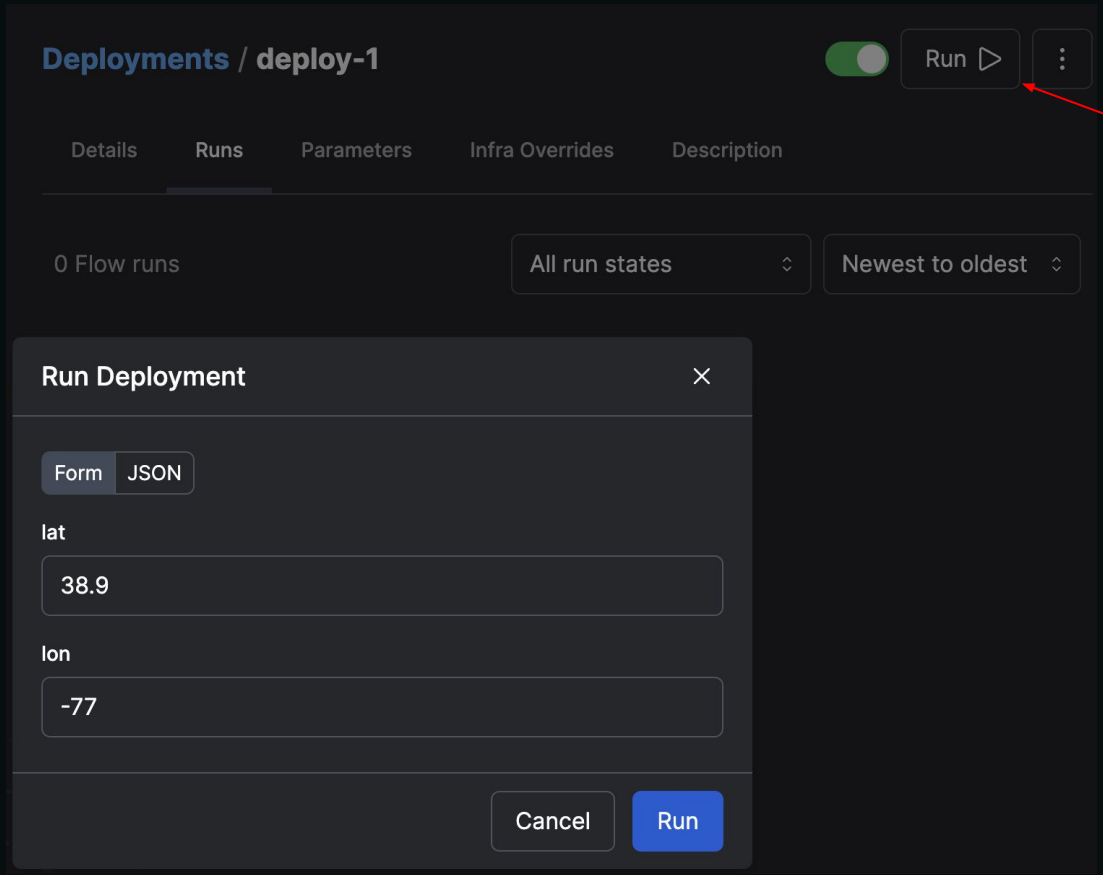
```
Your flow 'fetch-weather' is being served and polling for scheduled runs!

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
```


Check out in the UI

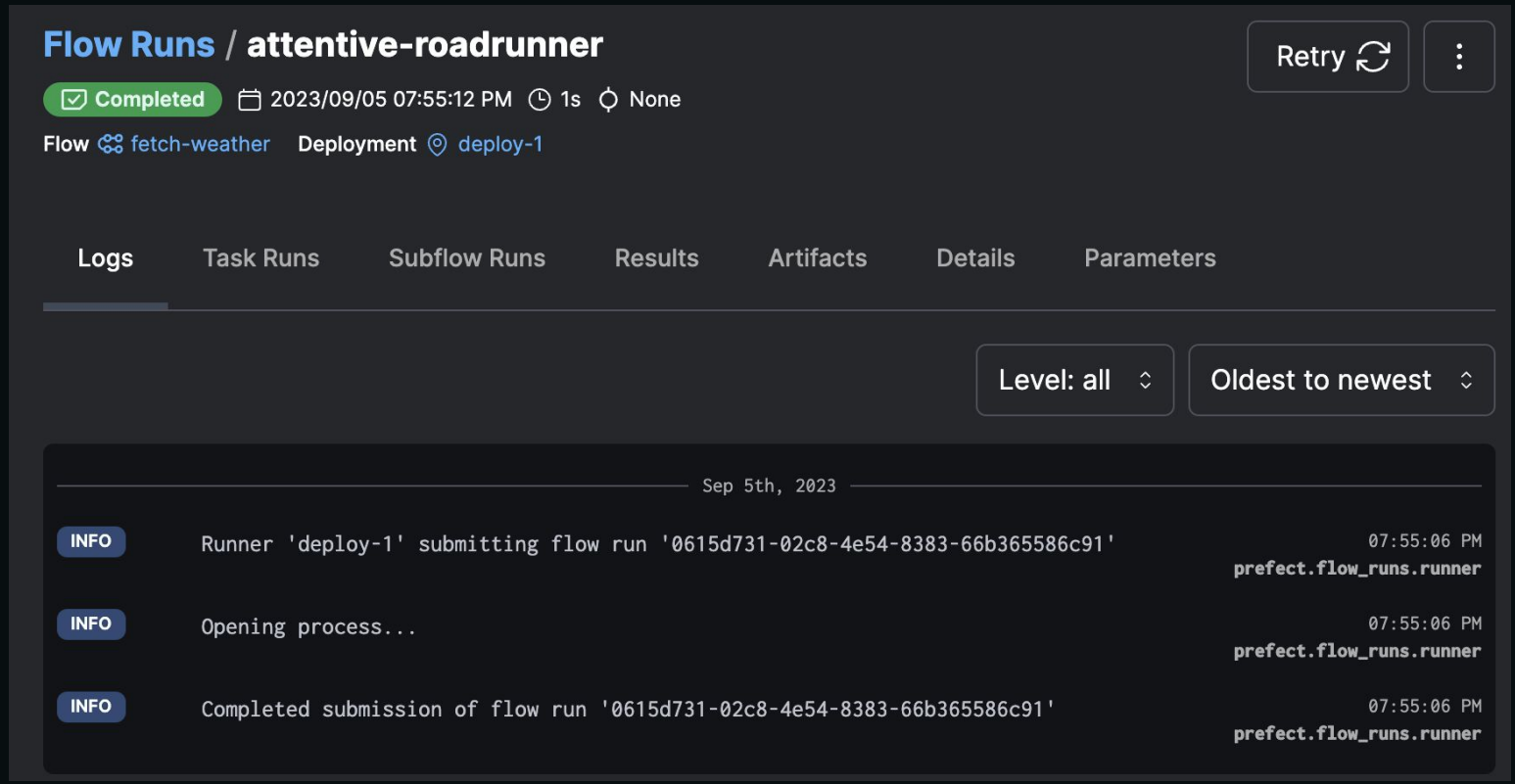
Deployments page

You just made a deployment!

Schedule a run from the UI

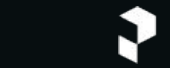


View the flow run logs



.serve()

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



Tasks

Tasks

- enable caching
- enable easy async
- are called inside flows

Example: pipeline

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
def fetch_weather(lat: float, lon: float):
    base_url = "https://api.open-meteo.com/v1/forecast/"
    weather = httpx.get(
        base_url,
        params=dict(latitude=lat, longitude=lon, hourly="temperature_2m"),
    )
    most_recent_temp = float(weather.json()["hourly"]["temperature_2m"][0])
    return most_recent_temp
```

Save data function

```
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
```

Tasks

Turn the first two functions into *tasks*

Turn into tasks

```python
from prefect import flow, task

@task
def fetch_weather(lat: float, lon: float):
    base_url = "https://api.open-meteo.com/v1/forecast/"
    weather = httpx.get(
        base_url,
        params=dict(latitude=lat, longitude=lon, hourly="temperature_2m"),
    )
    most_recent_temp = float(weather.json()["hourly"]["temperature_2m"][0])
    return most_recent_temp
```

Turn into tasks

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

Pipeline function flow

```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 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()
```

Tasks dos and don’ts

- ✅ Do call tasks within a flow
- ⛔ Don’t call tasks from other tasks
- ✅ Do keep tasks small

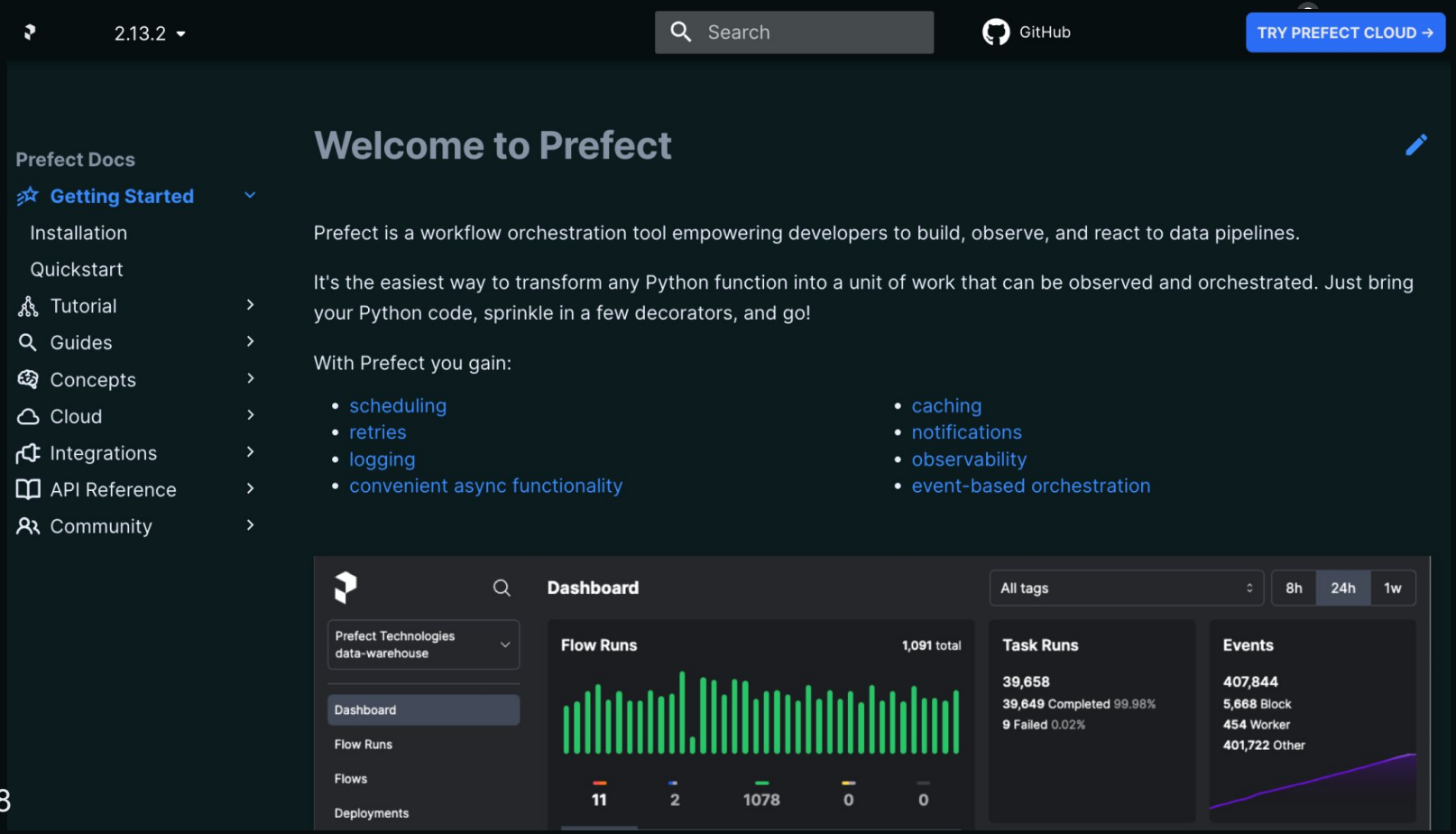
Note: Prefect is super Pythonic - conditionals are 👍

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:

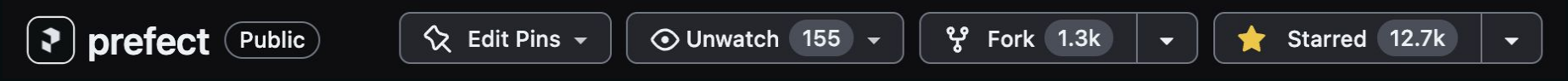
1. Configure Prefect logging to output to a file. This can be done by creating a custom logging configuration. You can find more details on how to do this in the Prefect documentation.

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
- *flow.serve()*
- Create modular tasks that can be called **from inside** a flow
- 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

Use Open-Meteo API

- Authenticate your CLI to Prefect Cloud
- Fine to use a personal account or a workspace
- Write flow code that fetches other weather metrics
- Make at least 3 tasks
- Use *.serve()* method to deploy your flow
- Run your flow from the UI and CLI
- Example: windspeed for the last hour:

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

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

If you give an engineer a job…

Could you

1. just fetch this data and save it?
2. set up logging?
3. visualize the dependencies?
4. automatically retry if it fails?
5. create an artifact for human consumption?
6. add caching?
7. add collaborators to run and view - who don’t code?
8. send me a message when it succeeds?
9. fetch the code from a remote git repository?
10. do it every hour?
11. automatically run it in response to a webhook?
12. run it in a Docker container-based environment?

Lab 101: a solution

One person from each group, share your code in Slack 

Discuss

Questions?

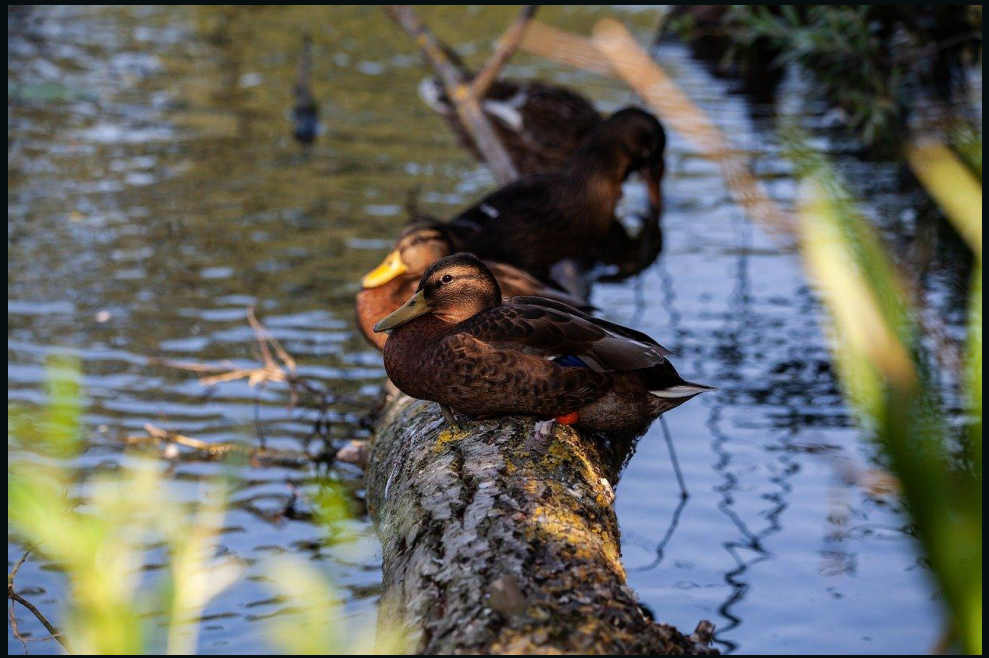
MODULE

102 - Intro to orchestration

102 Agenda

- Logging
- Retries for tasks and flows
- Results
- Artifacts
- Caching

Logging



log_prints

Log the print statements

@flow(log_prints=True)

log_prints

Want to log print statements by default?

Set environment variable:

export PREFECT_LOGGING_LOG_PRINTS = True

Logging

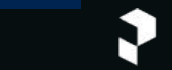
Create custom logs with *get_run_logger*

```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 DEBUG logging level:

```
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()
```

Logging

Output with INFO logging level:

```
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()
```


Retries

Minerva’s a smart duck.

She knows that sometimes APIs can be a bit buggy.

Retries

Specify in task or a flow decorator

@task(retries=2)

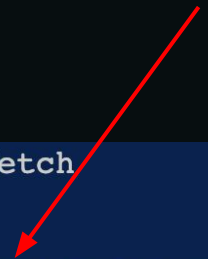
@flow(retries=3)

Flow retries for an unreliable API

```python
@flow(retries=4)
def fetch():
    cat_fact = httpx.get("https://httpstat.us/Random/200,500")
    if cat_fact.status_code >= 400:
        raise Exception()
    print(cat_fact.text)
```

Fail then automatic retry

```
  File "/Users/jeffhale/Desktop/prefect/pacc-2023/102/retry-flow.py", line 9, in fetch
    raise Exception()
Exception
10:16:47.412 | INFO    | Flow run 'flying-curassow' - Received non-final state 'AwaitingRetry'
when proposing final state 'Failed' and will attempt to run again...
200 OK
10:16:47.675 | INFO    | Flow run 'flying-curassow' - Finished in state Completed()
```



Retry delays

Retry 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_cat_fact():
    cat_fact = httpx.get("https://httpstat.us/Random/200,500")
    if cat_fact.status_code >= 400:
        raise Exception()
    print(cat_fact.text)
```

Fail then automatic retry with delay

```
  File "/Users/jeffhale/Desktop/prefect/pacc-2023/102/retries-delay.py", line 9, in fetch
_cat_fact
    raise Exception()
Exception
10:26:10.177 | INFO    | Task run 'fetch_cat_fact-0' - Received non-final state 'Awaiting
Retry' when proposing final state 'Failed' and will attempt to run again...
200 OK
10:26:10.495 | INFO    | Task run 'fetch_cat_fact-0' - Finished in state Completed()
10:26:10.518 | INFO    | Flow run 'careful-swine' - Finished in state Completed('All stat
es completed.')
```

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

Results

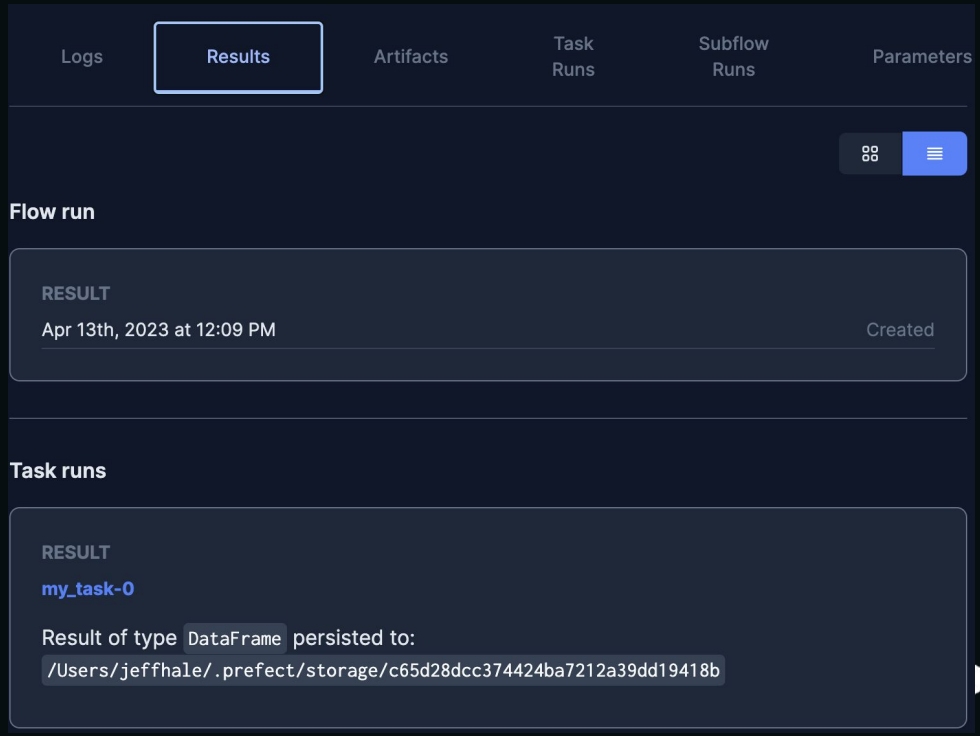
By default, Prefect returns a result that is ***not*** persisted to disk.

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()

my_flow()
```


Results

Info about a result is viewable in the UI



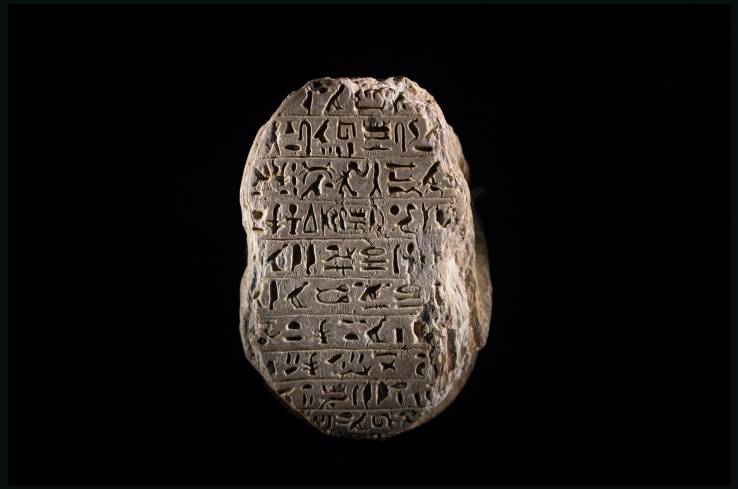
Results

The result in the *storage* folder by default

Artifacts

Artifacts

Persisted outputs such as Markdown, tables, or links.



Artifacts

- Meant for human consumption
  - Data quality checks
  - Documentation
- Prettier output
- Results
- Must have a key

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 *Flow Runs* or *Artifa*

Caching

Caching

What?

Why?

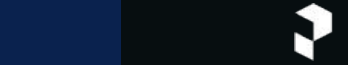
⚠️ task only

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)
```

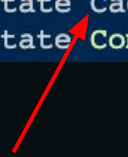
Caching

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

What will happen if you run with a different argument?

```
22:36:07.750 | INFO    | prefect.engine - Created flow run 'daffodil-millipede' for flow 'hello-flow'
22:36:07.836 | INFO    | Flow run 'daffodil-millipede' - Created task run 'hello_task-0' for task 'hello_task'
22:36:07.837 | INFO    | Flow run 'daffodil-millipede' - Executing 'hello_task-0' immediately...
Hello Marvin!
22:36:07.877 | INFO    | Task run 'hello_task-0' - Finished in state Completed()
22:36:07.892 | INFO    | Flow run 'daffodil-millipede' - 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)
```

Caching

You can cache to S3 or other cloud storage provider.

Just persist your result to a Prefect Block for your storage provider.

Blocks are a future topic. 🙂

More helpful resources

Prefect CLI

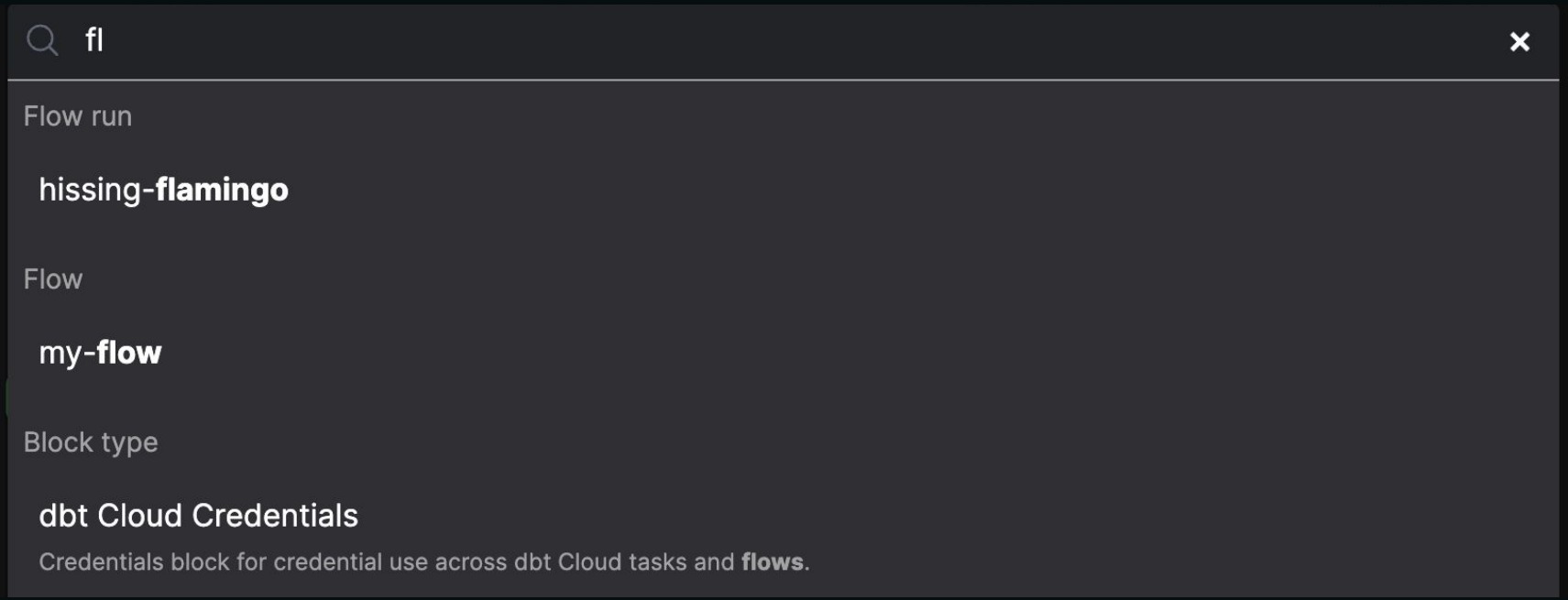
Start commands with *prefect --help* is always available

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.

- Logging
- States
- Retries for tasks and flows
- Results
- Artifacts
- Caching
- More resources: help & search

Lab 102

Lab 102

- Use your flow that grabs weather data from open-meteo
- 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

1. just fetch this data and save it?
2. set up logging?
3. visualize the dependencies?
4. automatically retry if it fails?
5. create an artifact for human consumption?
6. add caching?
7. add collaborators to run and view - who don’t code?
8. send me a message when it succeeds?
9. fetch the code from a remote git repository?
10. do it every hour?
11. automatically run it in response to a webhook?
12. run it in a Docker container-based environment?

Lab 102: a solution

One person from each group, share your code in Slack 

Discuss

Questions?

MODULE

103 - Cloud features & Blocks

103 Agenda

- Cloud features
- Blocks 
- Automations
- Events
- Advanced deployments with *serve*

Prefect Cloud

Prefect Cloud

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

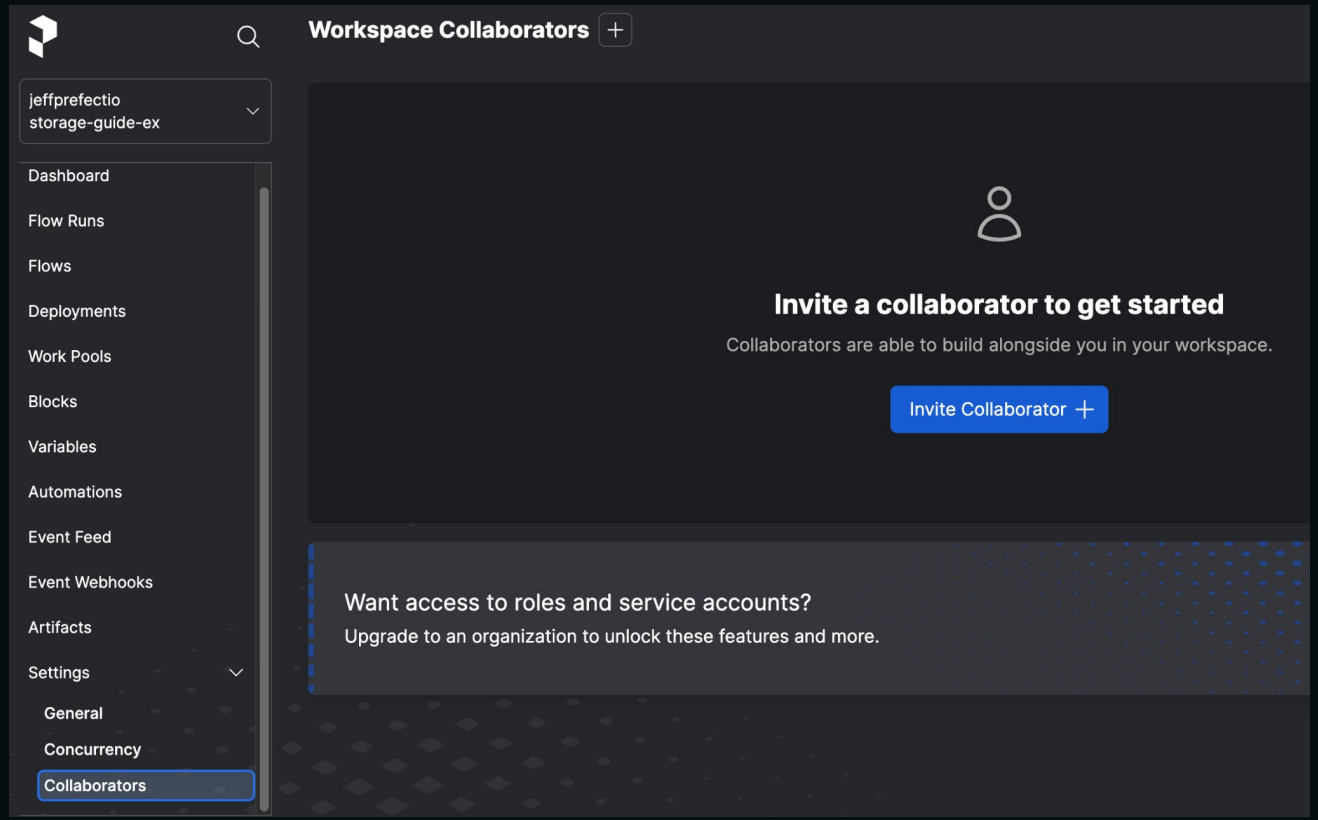
Prefect Cloud Workspaces

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

Prefect Cloud - Personal

- 3 free users (can buy more seats) - self service
- 1 free workspace
- 7 day flow run history

Prefect Cloud - Add Collaborators (personal account)



Prefect Cloud - Organization

- Service accounts
- SSO
- RBAC
- 30 day flow run history
- 72 hour audit log

Prefect Cloud - Enterprise

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

Prefect Cloud

PERSONAL

Free Forever

Great for getting started, solo data practitioners, and POCs

CREATE YOUR WORKSPACE ›

- All core features
- Basic auth & collaboration
- Good performance

ORGANIZATION POWERFUL

$450/Month

For teams with production workflows or access management requirements

START YOUR FREE TRIAL ›

- Service accounts
- Advanced collaboration
- Basic RBAC
- Better performance & longer retention

ENTERPRISE

Contact Us

Great for large teams, those with the highest volume, or for companies with specific security requirements

GET IN TOUCH ›

- SCIM & RBAC
- Support
- Discounts and custom terms
- Best performance & longest retention

Prefect Cloud - Default Roles (Org + Enterprise)

Organization level

- Admin
- Member

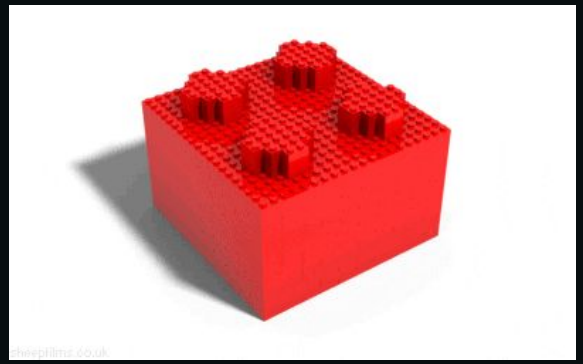
Workspace level

- Viewer
- Runner
- Developer
- Owner
- Worker

Blocks

Blocks

Blocks are a cool Prefect feature



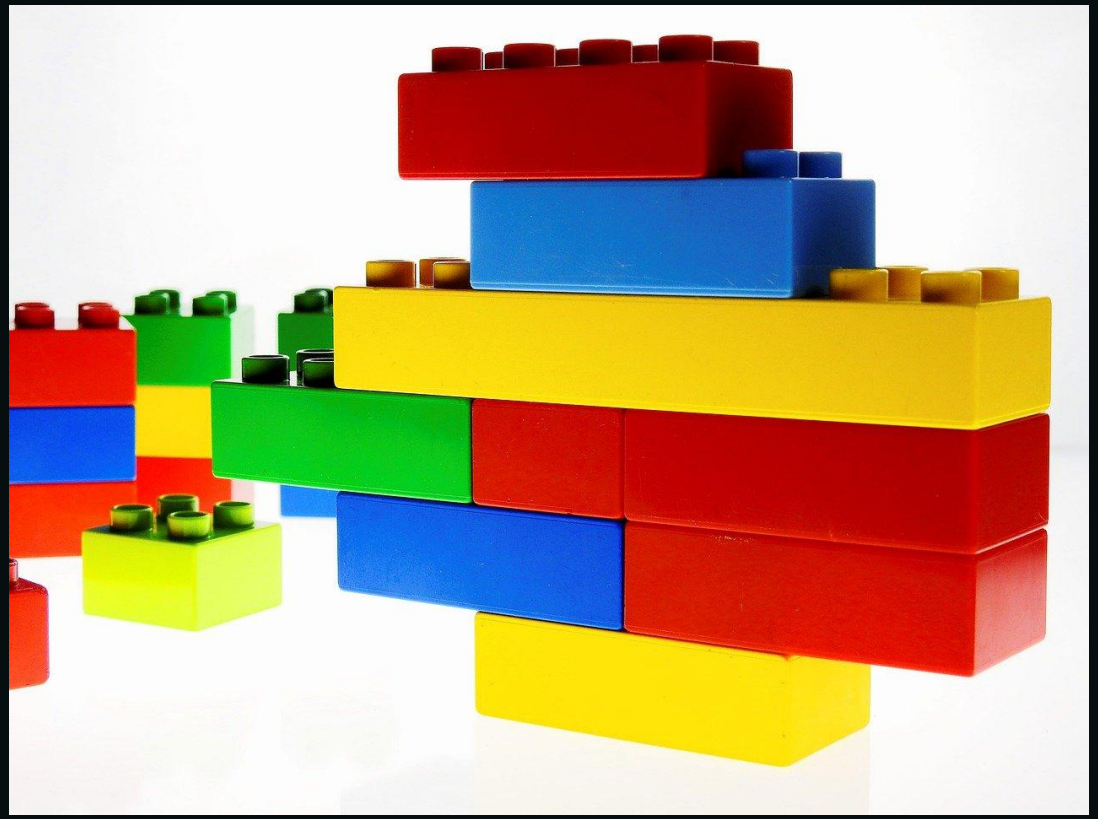
Available on Cloud and self-hosted

Blocks

Configuration

+

Code

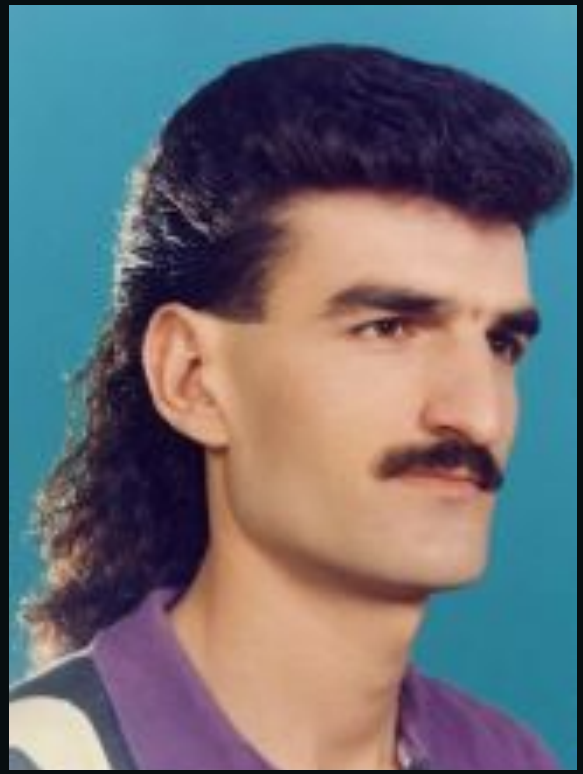


Blocks

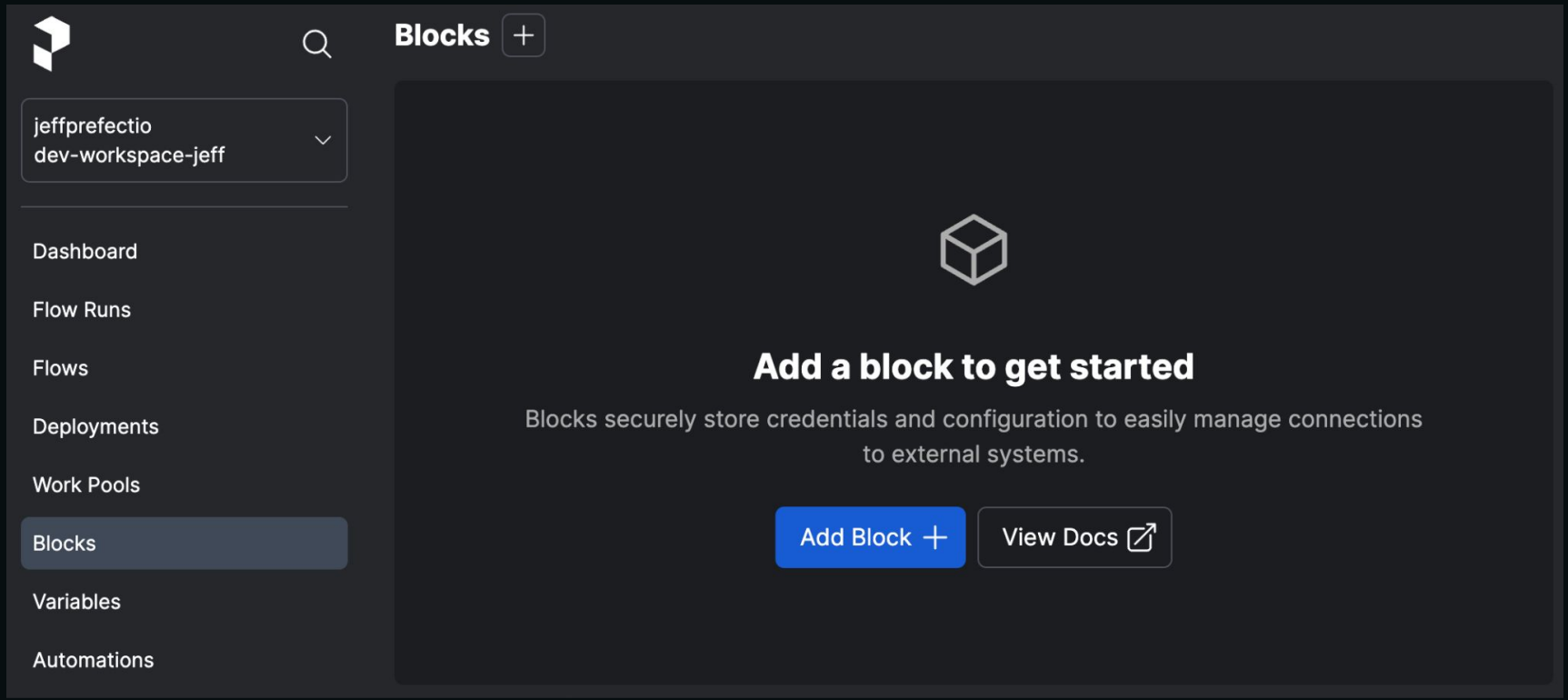
The Block mullet:

Structured form in front,

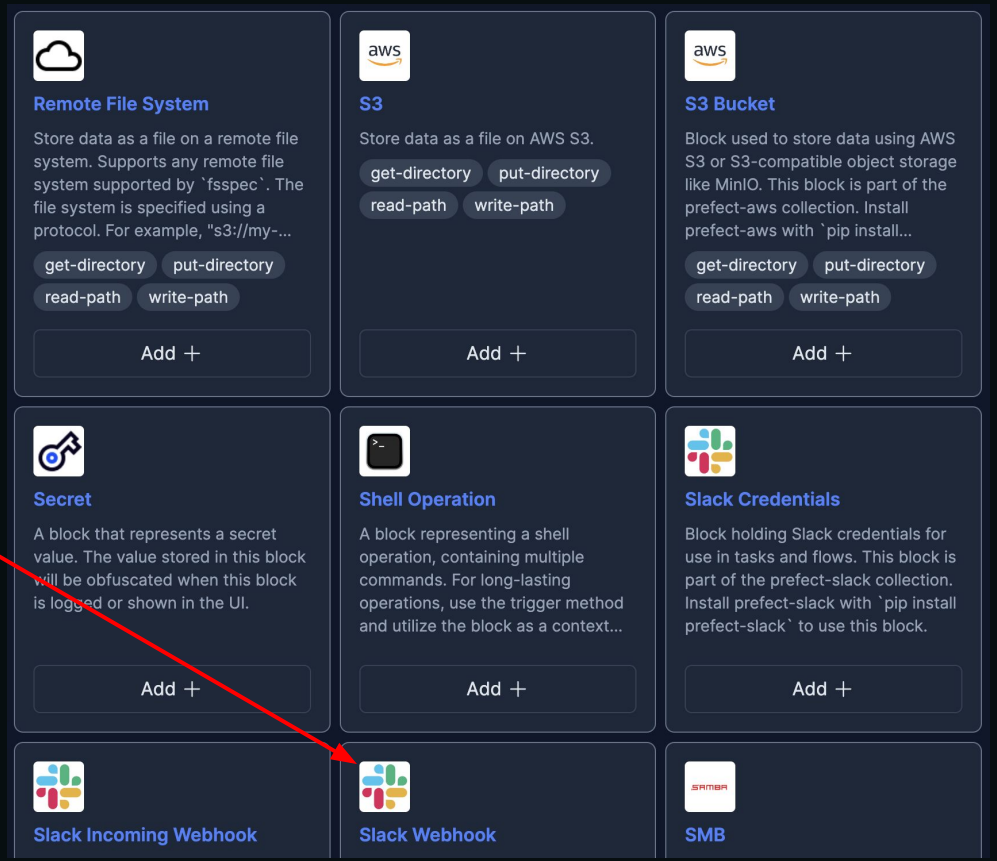
flexible code in back



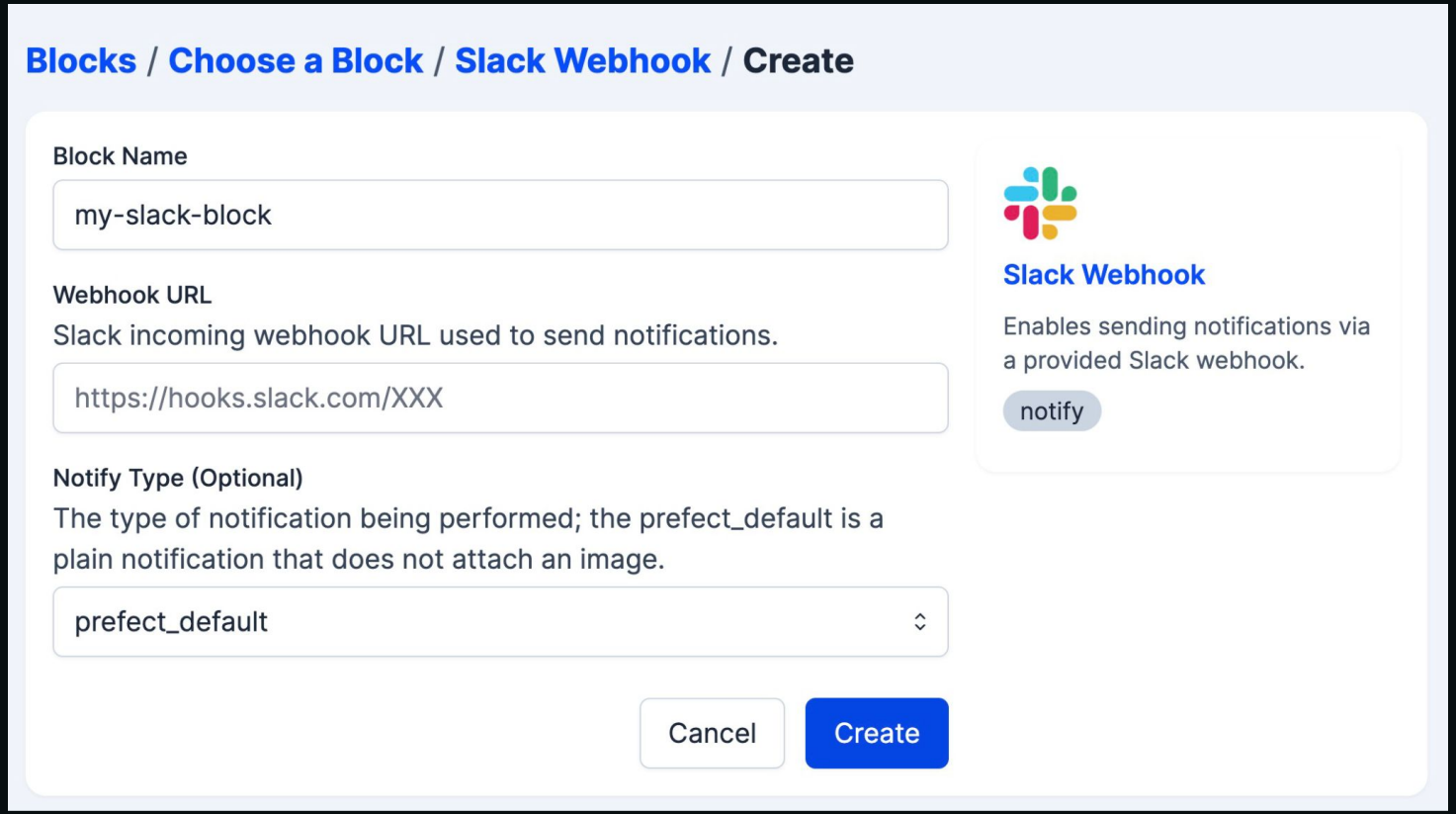
Create a Block from the UI



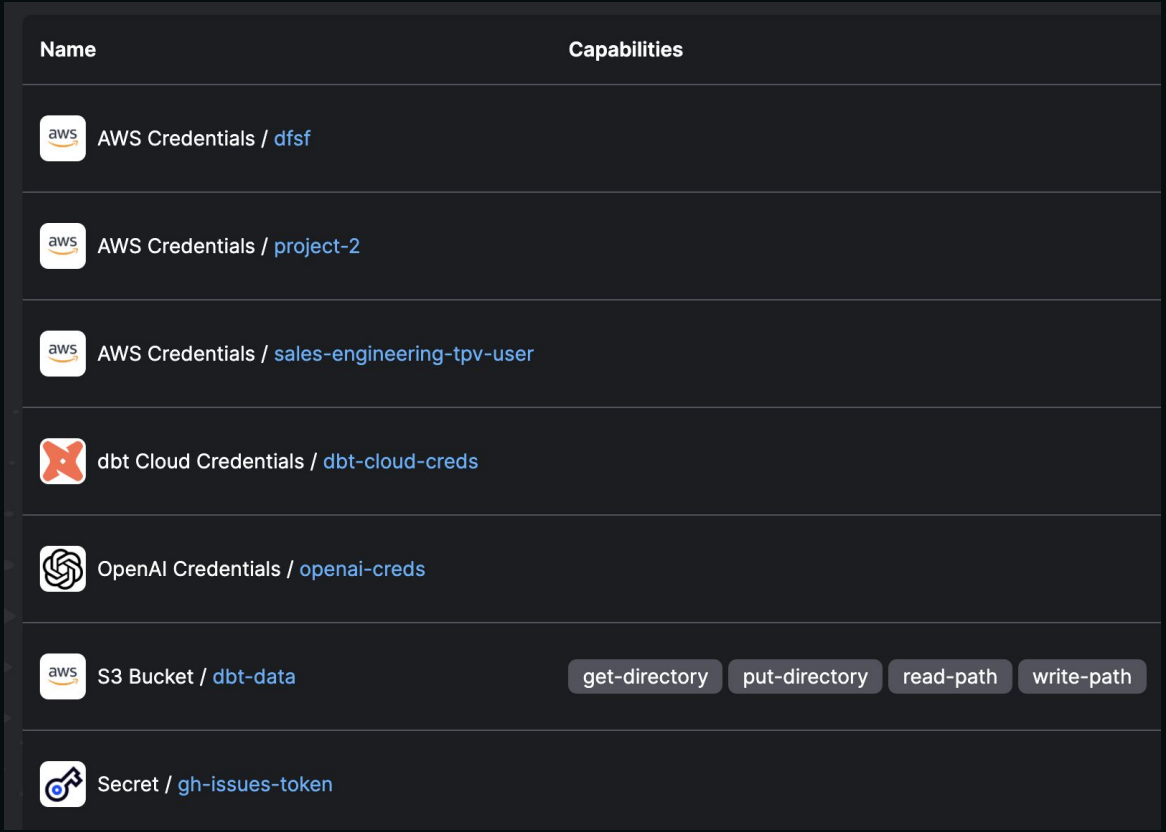
Create a block from the UI - choose a block type



Create a block from the UI



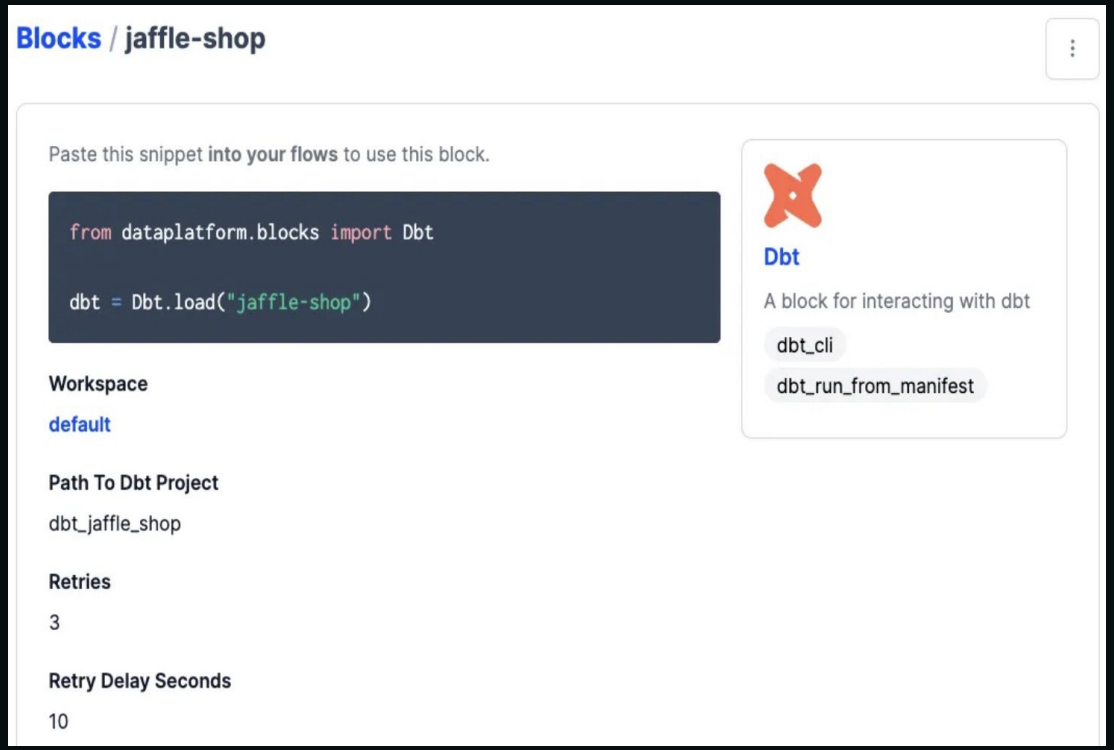
Blocks in UI


Name
Capabilities
AWS Credentials / dfsf
AWS Credentials / project-2
AWS Credentials / sales-engineering-tpv-user
dbt Cloud Credentials / dbt-cloud-creds
OpenAI Credentials / openai-creds
S3 Bucket / dbt-data
get-directory
put-directory
read-path
write-path
Secret / gh-issues-token

Under the hood, block types are Python classes

```python
from prefect.blocks.core import Block

class Dbt(Block):

    """
    A block for interacting with dbt
    """

    _block_type_name = "Dbt"
    _logo_url = "https://images.ctfassets.net/gm98wzqotmnx/5zE9lxf
    _block_schema_capabilities = ["dbt_cli", "dbt_run_from_manifes

    def dbt_cli(self, dbt_command: str) -> None:
        state = trigger_dbt_cli_command.with_options(
            name=f"{self.default_dbt_cli_emoji}{dbt_command}",
            retries=self.retries,
            retry_delay_seconds=self.retry_delay_seconds,
        )
```


Blocks are instances of those Python classes



Create a block with 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())
```

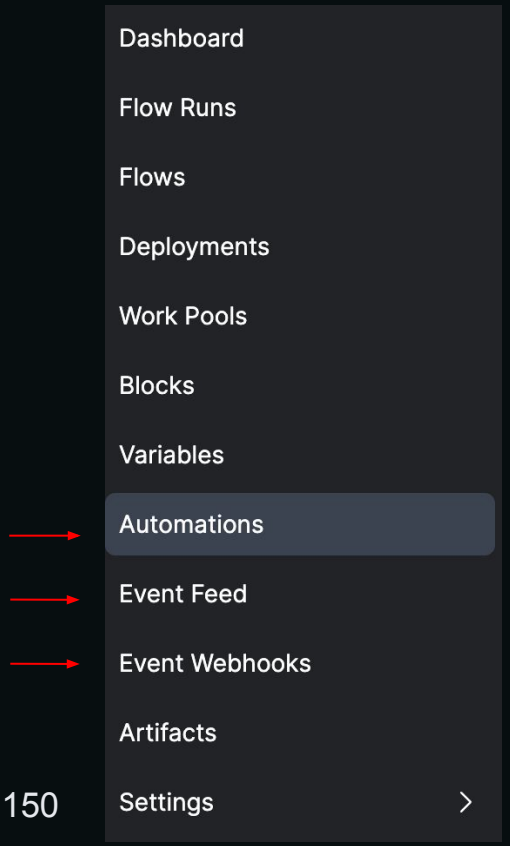

Blocks

Reusable, modular, configuration + code

- Better than hard coding
- Better than environment variables
- Nestable
- Can create own types

Cloud Only Features

Automations + Events API



Automations ⚡

Automations

Cloud only

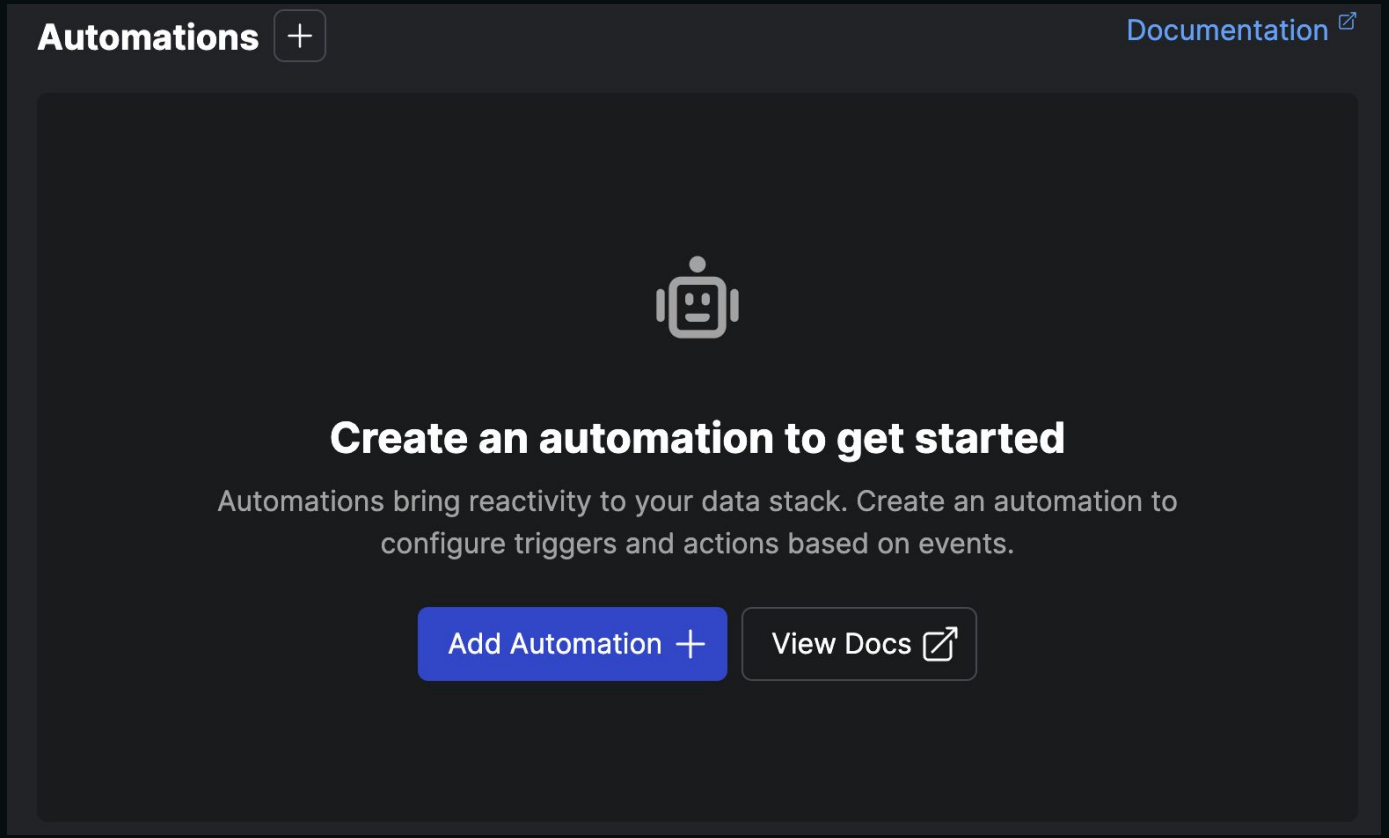
Flexible framework

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

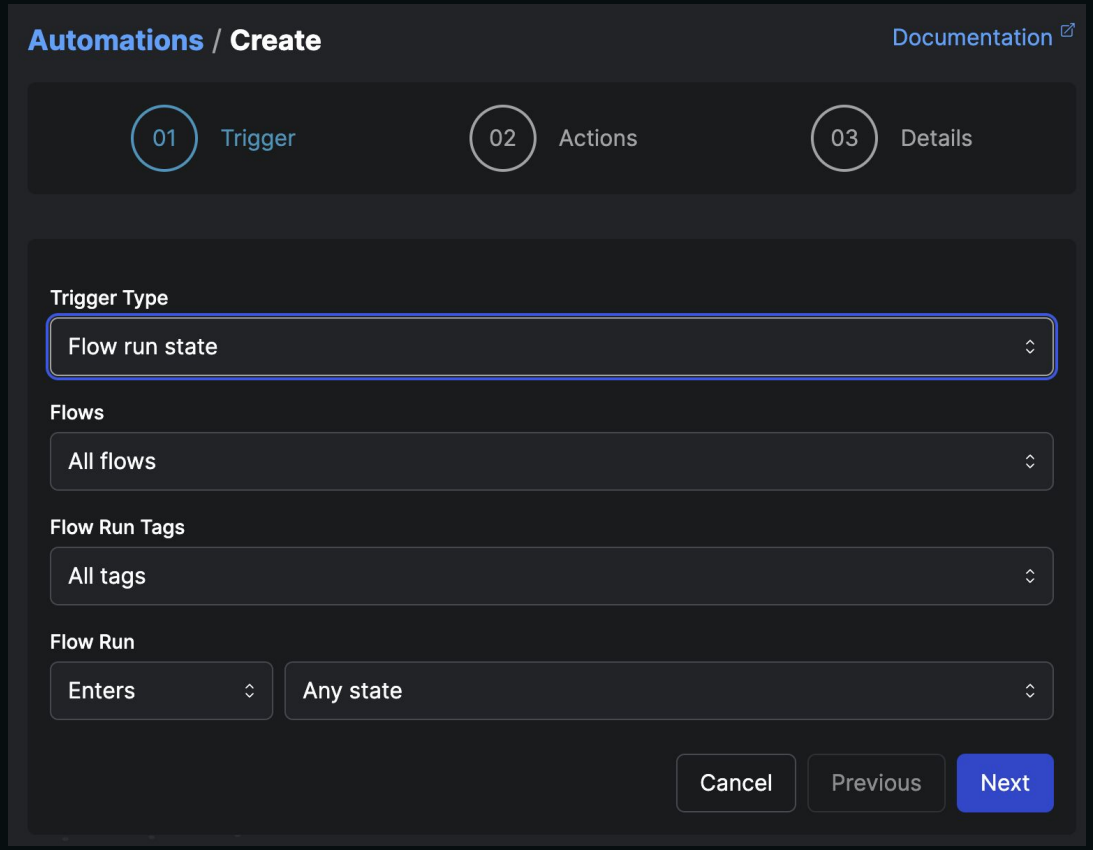
Automation examples

- If a flow with tag **prod** fails, send an email
- If a data quality check fails, run a deployment to fetch more data

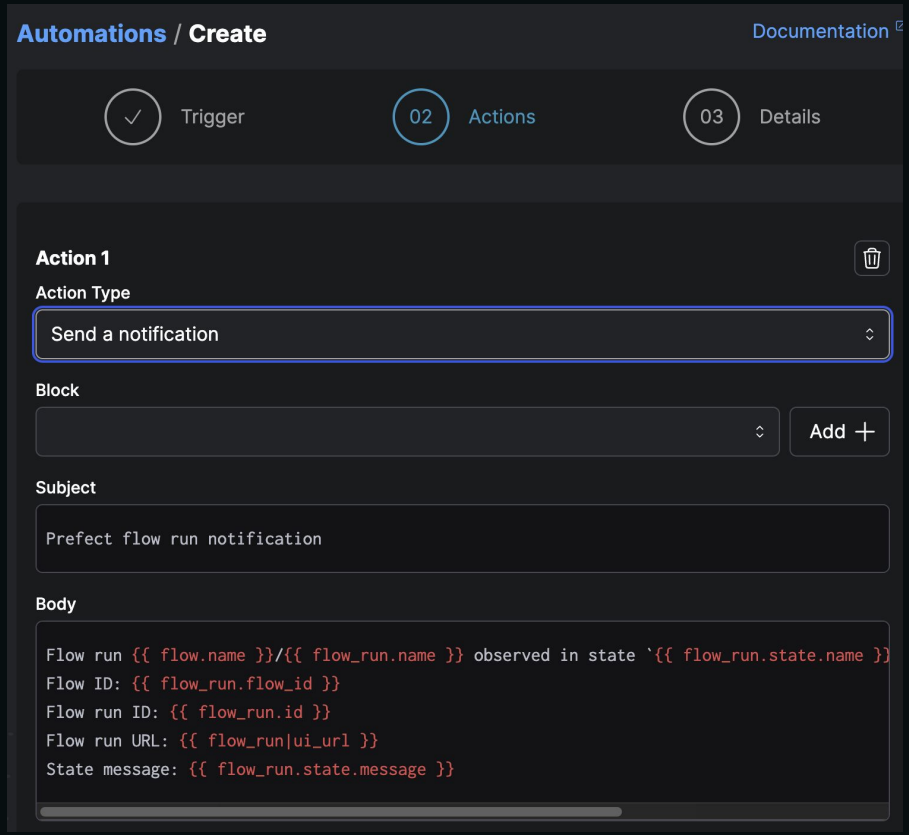
Create an automation notification



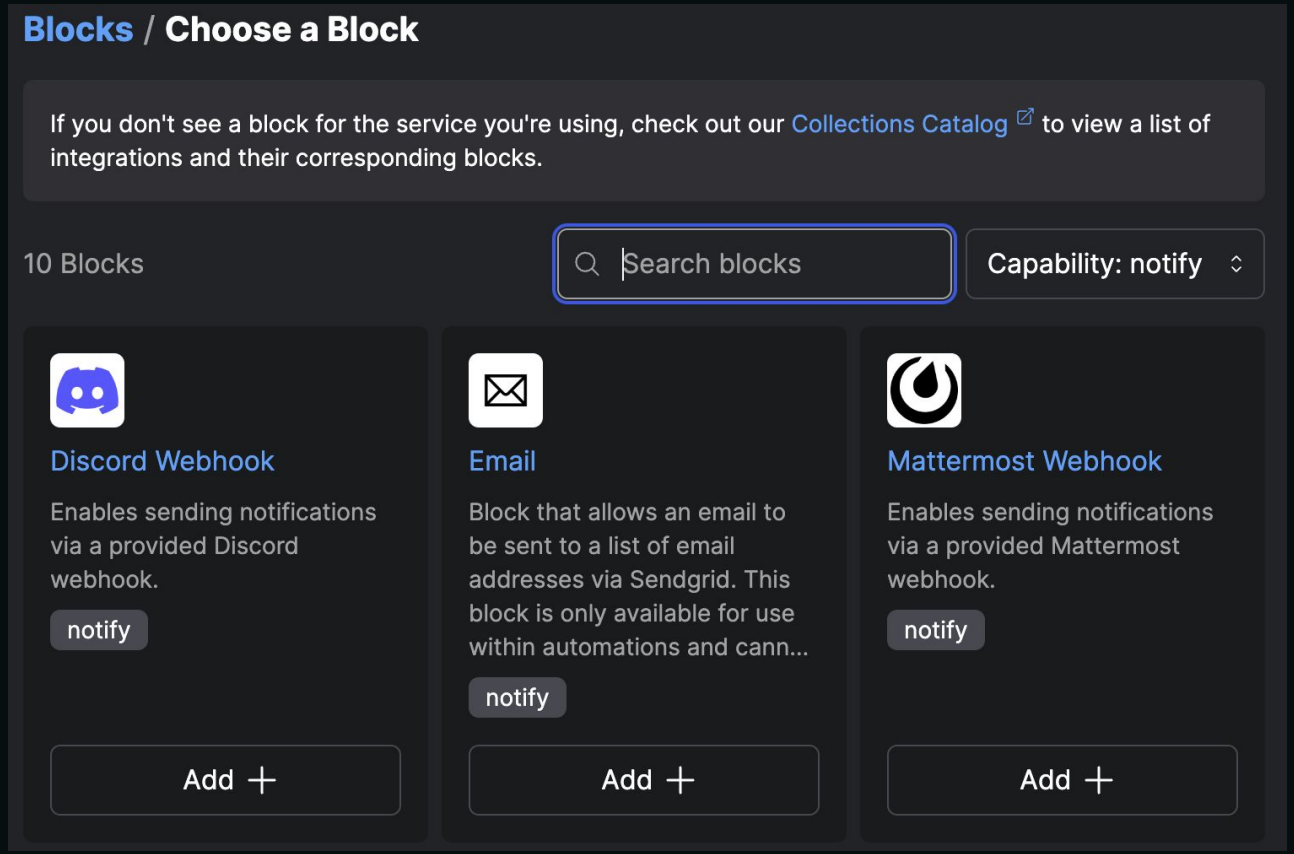
Automation notifications



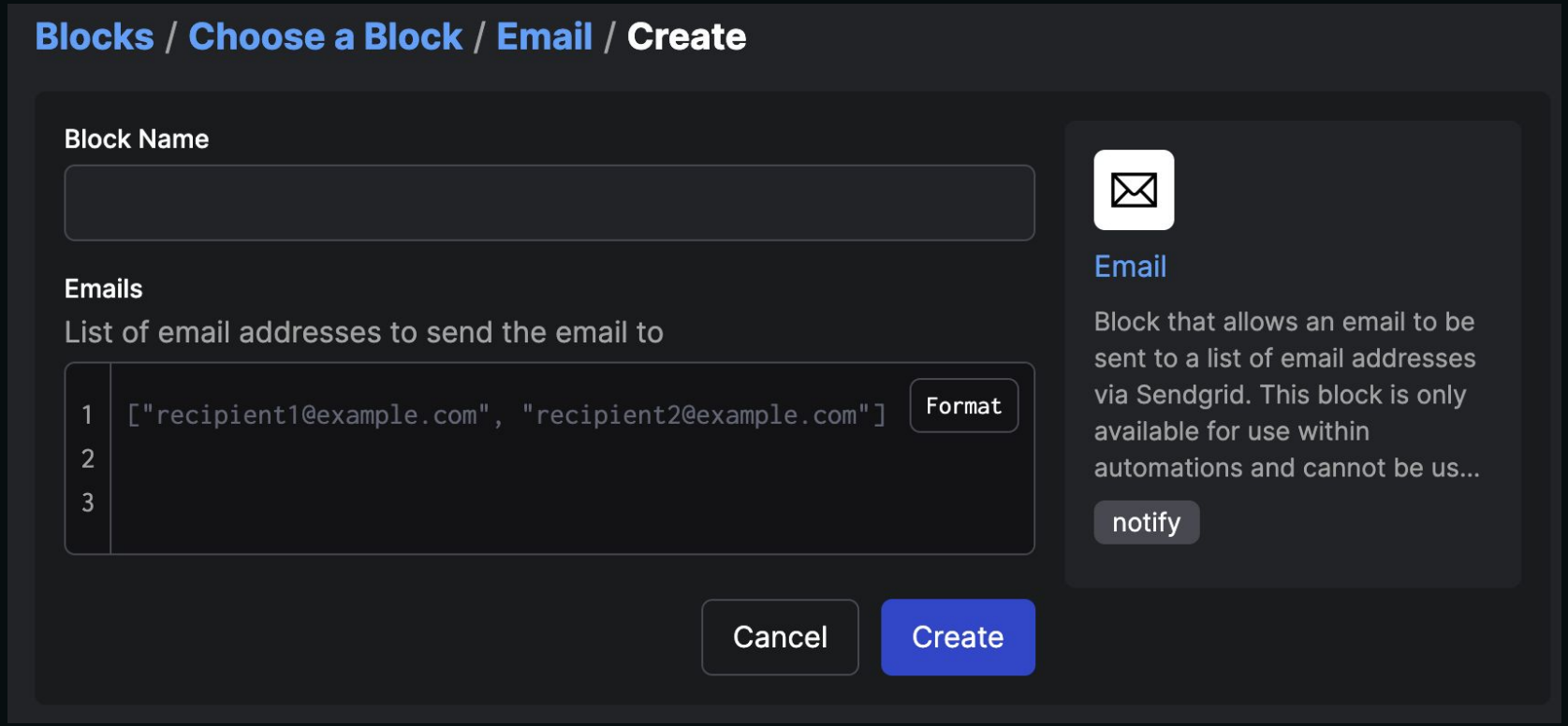
Automation notifications



Automation notifications - create a notification block



Automation notifications - create a notification block



Events

Events

- A notification of a change
- A feed of activity recording what's happening

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

Include

- API calls
- state transitions
- changes in environment

Events

Power several Cloud features

- flow run logs
- audit logs
- automations

Create a custom event

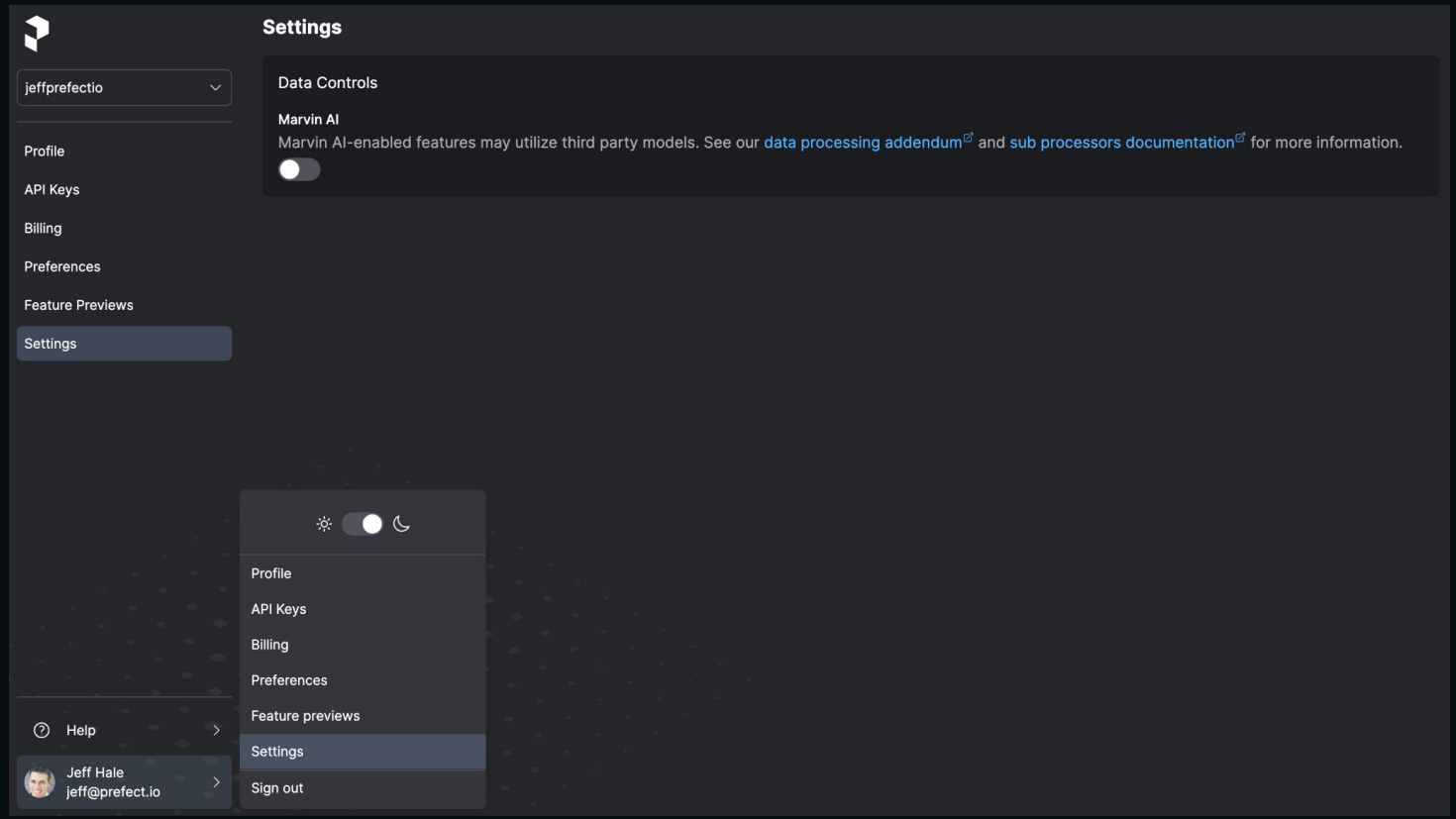
```python
from prefect.events import emit_event

emit_event(
    event=f"bot.{bot.name.lower()}.responded",
    resource={"prefect.resource.id": f"bot.{bot.name.lower()}"},
    payload={
        "user": event.user,
        "channel": event.channel,
        "thread_ts": thread,
        "text": text,
        "response": response.content,
        "prompt_tokens": prompt_tokens,
        "response_tokens": response_tokens,
        "total_tokens": prompt_tokens + response_tokens,
    },
)
```


Webhooks

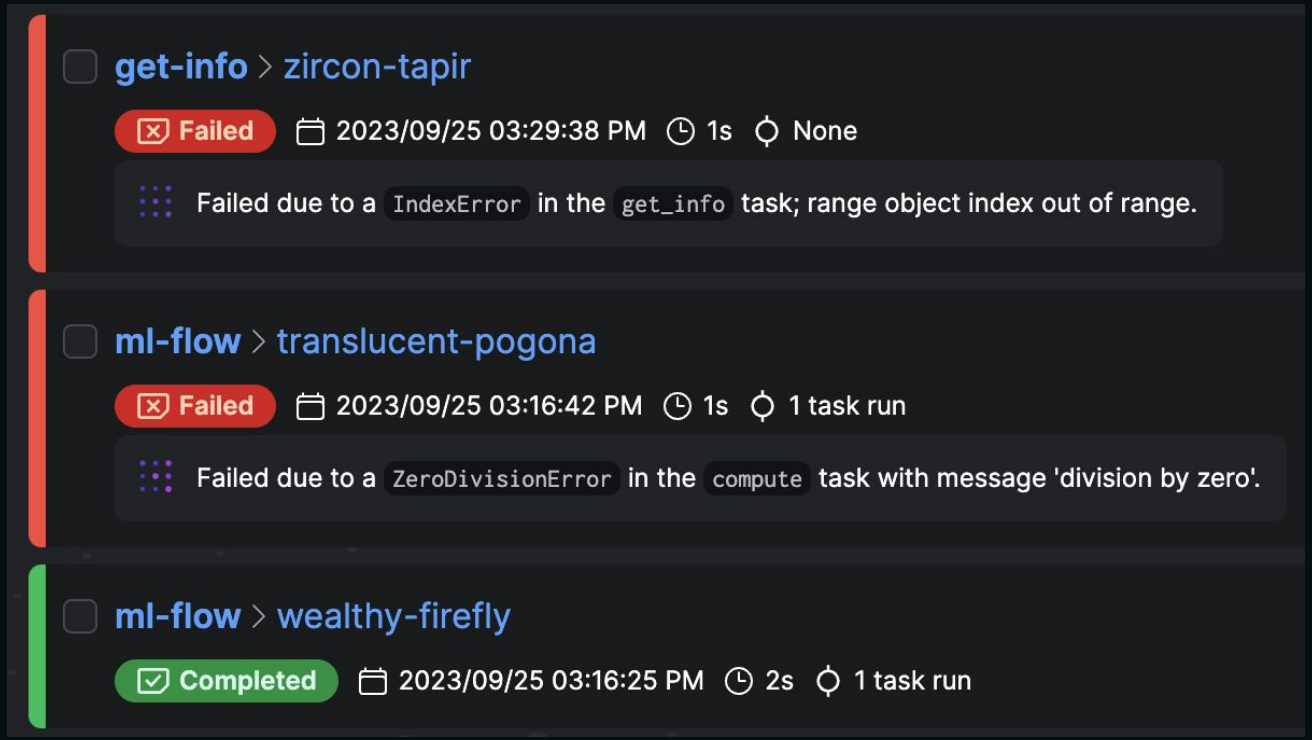
- Create events that can kick off automations
- Common in event-driven workflows
- Will explore in a future module 🙂

Error summaries by Marvin AI

Error summaries by Marvin AI



Error summaries by Marvin AI



Deploy multiple flows with *serve*

Deploy multiple flows

```python
import time
from prefect import flow, serve

@flow
def slow_flow(sleep: int = 60):
    "Sleepy flow - sleeps the provided amount of time (in seconds)."
    time.sleep(sleep)

@flow
def fast_flow():
    "Fastest flow this side of the Atlantic."
    return

if __name__ == "__main__":
    slow_deploy = slow_flow.to_deployment(name="sleeper-scheduling")
    fast_deploy = fast_flow.to_deployment(name="fast-scheduling")
    serve(slow_deploy, fast_deploy)
```

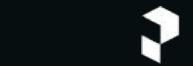

Deploy multiple flows

- import *serve*
- use *to_deployment()* method
- use *serve* function and pass it the deployment objects

103 Recap

You've learned about

- Prefect Cloud features
- Blocks
- Automations
- Events
- Error summaries by Marvin AI
- Multiple deployments with *serve*



Lab 103

103 Lab

- Make an email notification automation for any flow run completion
- Run a flow a few times from the CLI
- See the event feed in the UI
- Stretch: create a custom event - run it and see the event in the UI

If you give an engineer a job…

Could you

1. just fetch this data and save it?
2. set up logging?
3. visualize the dependencies?
4. automatically retry if it fails?
5. create an artifact for human consumption?
6. add caching?
7. add collaborators to run and view - who don’t code?
8. send me a message when it succeeds?
9. fetch the code from a remote git repository?
10. do it every hour?
11. automatically run it in response to a webhook?
12. run it in a Docker container-based environment?

Lab 103: a solution

One person from each group, share your code in Slack 

Discuss

Questions?