PACC

Prefect Associate
Certification Course

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

Norms

Code of conduct

- We expect all participants to be kind and respectful
- Reach out to any of the instructors 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

Agenda for 1st half of the course

- Intro workflow orchestration and Prefect’s role
- 101: Prefect basics: Write workflows you can schedule and monitor
- 101 lab
- 102: Orchestration and observation: Understand workflow state and guard against failure
- 102 lab
- 103: Work with data and create automatic alerts
- 103 lab

Agenda for 2nd half of the course

Conquer infrastructure with workpool-based deployments and much more! 🎉

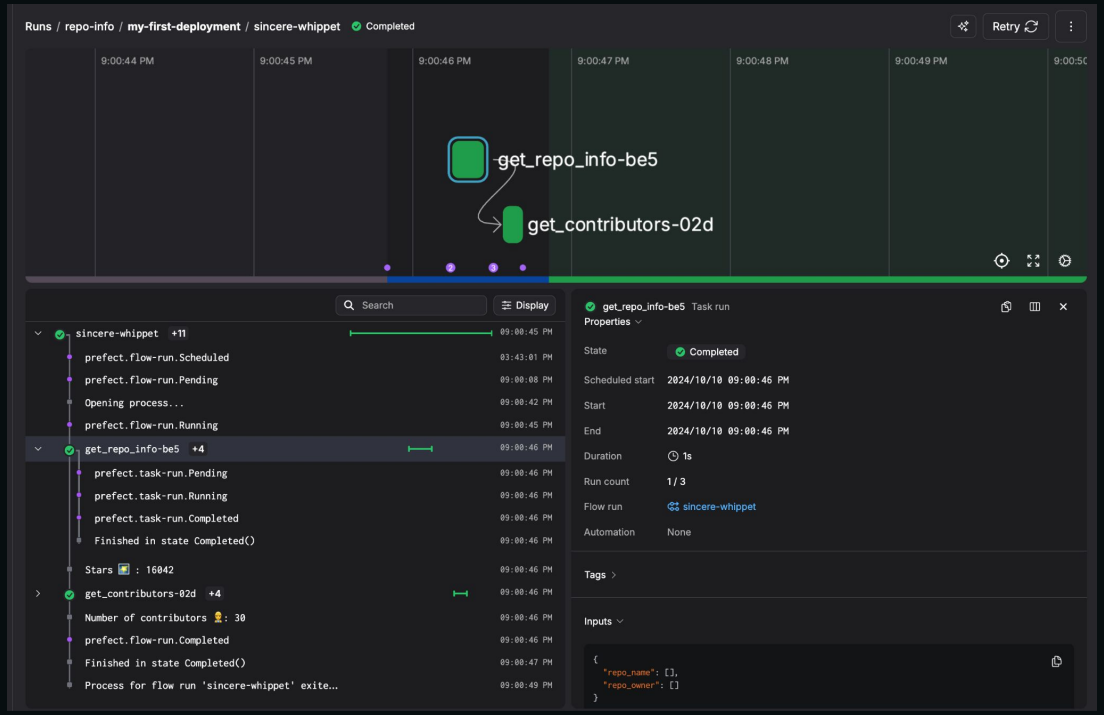
Why workflow management?

Answers the questions:

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

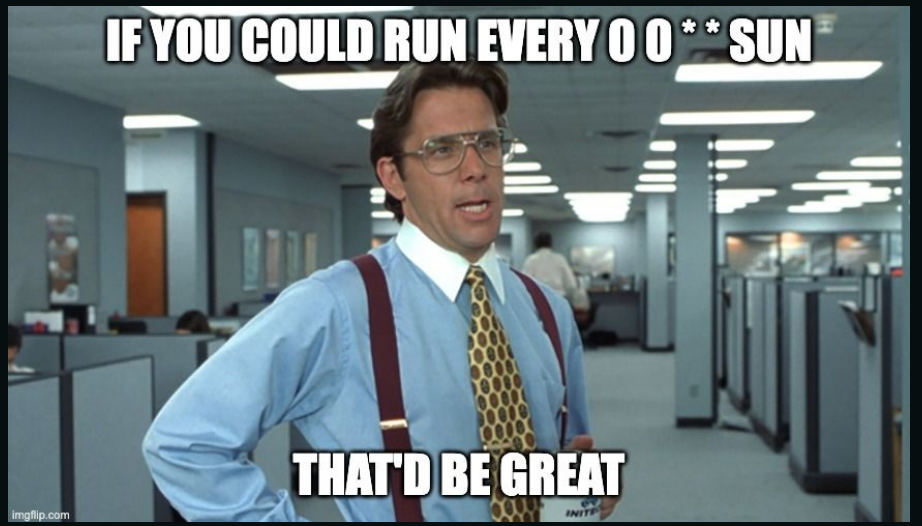
What?

Code that moves and transforms data



When?

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



Where?

- Locally
- In the cloud

How?

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

Who?

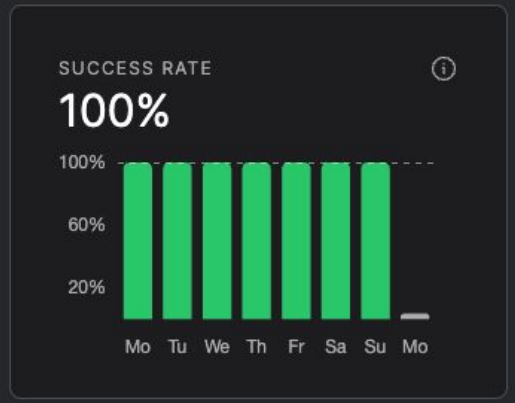
- Auth - SSO/SCIM
- RBAC
- Auditable

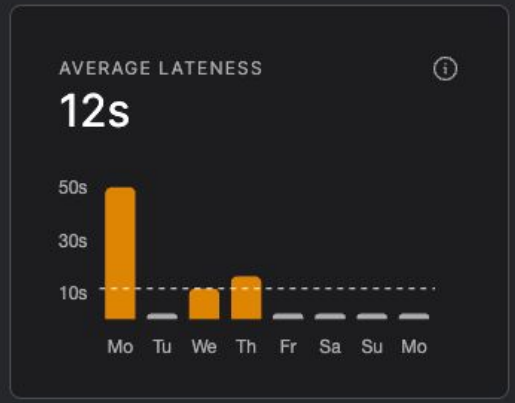
What is Prefect?

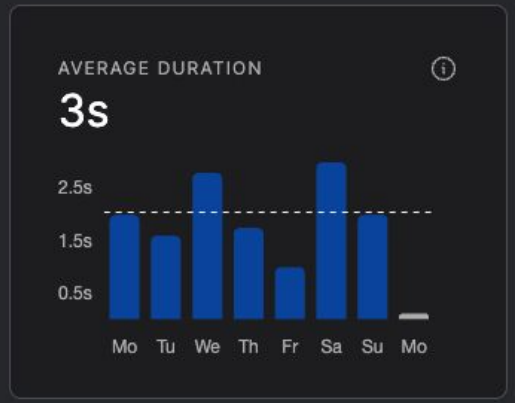
Prefect is an orchestration and observability platform that empowers developers to build resilient workflows you can trust. 💙

Deployments / datalake_listener

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







Runs | Upcoming | Parameters | Configuration | Description

104 Flow runs

Search by run name | All except scheduled | Newest to oldest

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

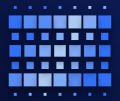
Completed | 2024/06/16 06:01:51 AM | 2 Parameters | 3s | 3 Task runs

Deployment datalake_listener | Work Pool Demo-ECS | Work Queue default

Prefect data workflow orchestration

Automation

Flexible Orchestration in Pure Python



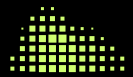
Resilience

Faster Recovery with Resilient Code

Scalability

Scalable compute resources and governance

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

- Manage agentic workflow troubleshooting
- Unify orchestration across multiple LLM tasks

Data platform engineers

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

Outcomes

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

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

101 Agenda

- Setup
- From Python function to Prefect flow
- Create a deployment with *.serve()*
- Run a deployment
- Schedule a deployment
- Parameters
- Helpful resources

prefect version

Prefect information in the CLI

prefect version

```
Version:              3.1.5
API version:          0.8.4
Python version:       3.12.6
Git commit:           3c06654e
Built:                Mon, Dec 2, 2024 6:57 PM
OS/Arch:              darwin/arm64
Profile:              cloud
Server type:          cloud
Pydantic version:     2.9.2
```


Install most recent version of Prefect 3 in

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
    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 🔑

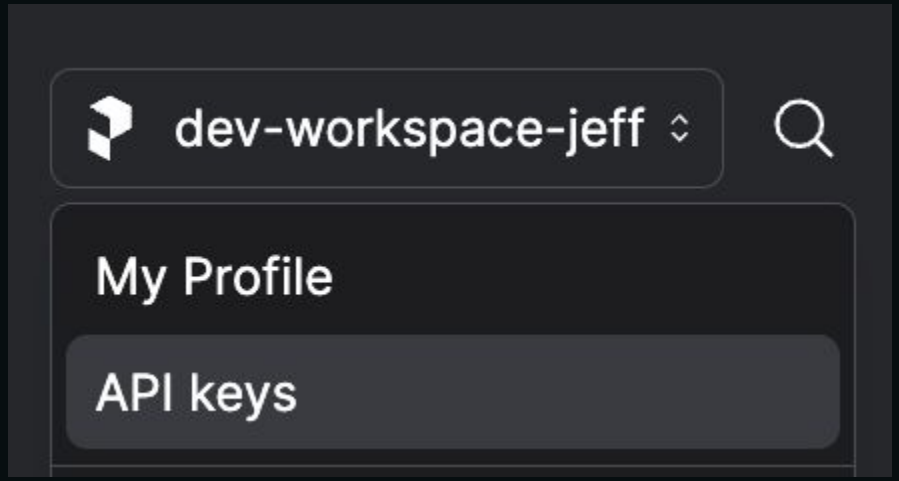
Prefect Cloud

OR, if UI doesn’t work:

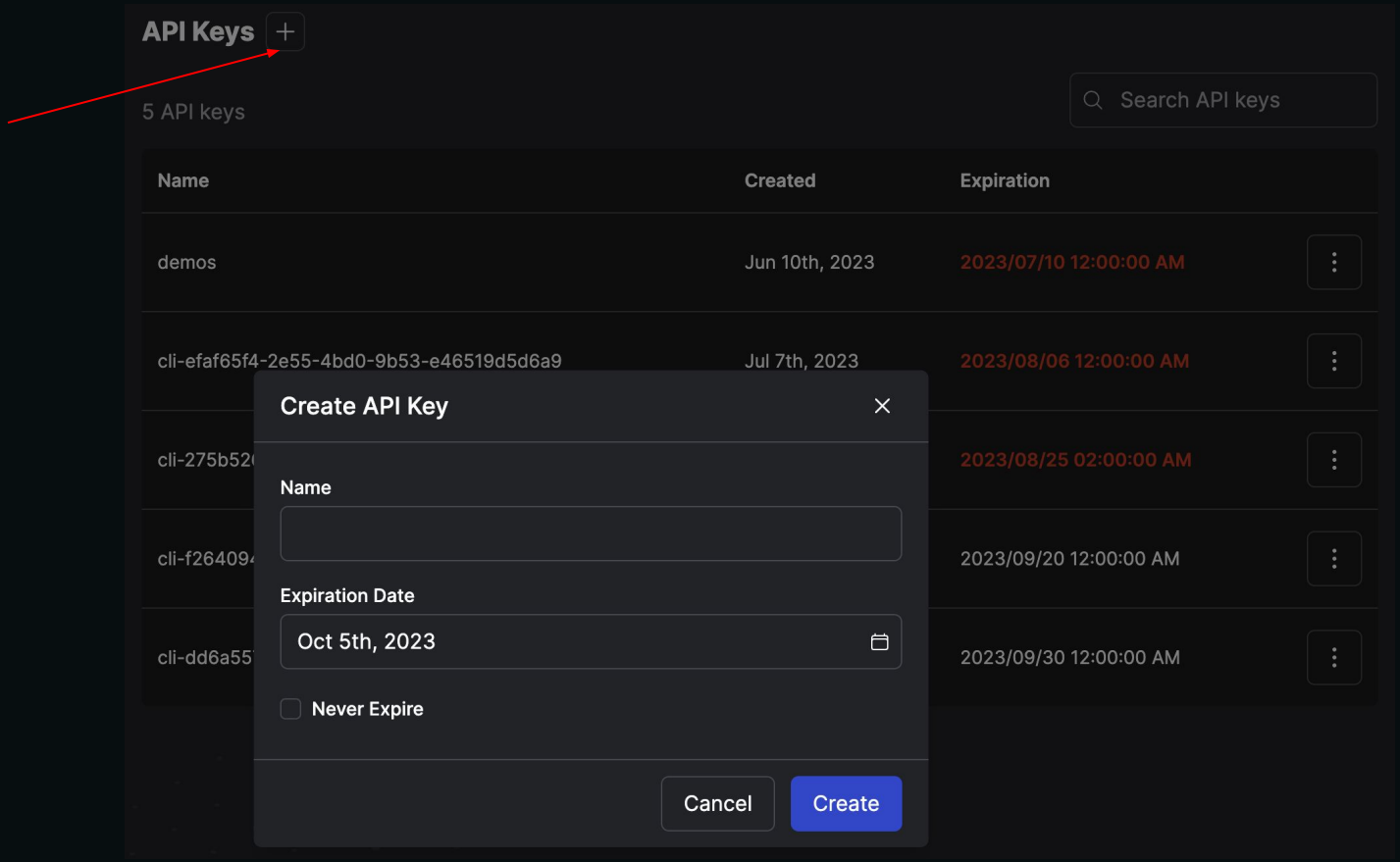
- Select *Paste an API key*
- Manually create an API key from Prefect Cloud in the UI

Prefect Cloud - API key

(Top left in UI)



Prefect Cloud - API key



Prefect Cloud - API key

Paste API key at terminal prompt

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


To just 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
```


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

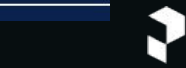
Prefect profiles

List: *prefect profile ls*

Prefect profiles

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

```
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. API URL
2. API key for Prefect Cloud (if using Cloud)
3. 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*

Flows: Add superpowers to your Python 

Project

We are consulting for a company that uses weather forecast data in financial products.

We'll fetch and use weather forecast data from Open-Meteo 🌦️ 🌡️

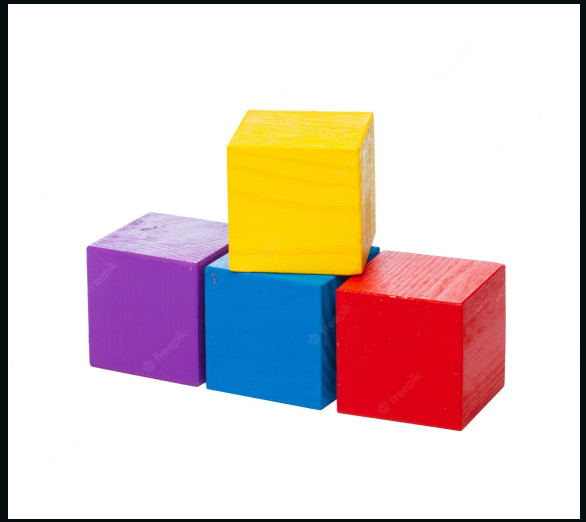
open-meteo.com

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



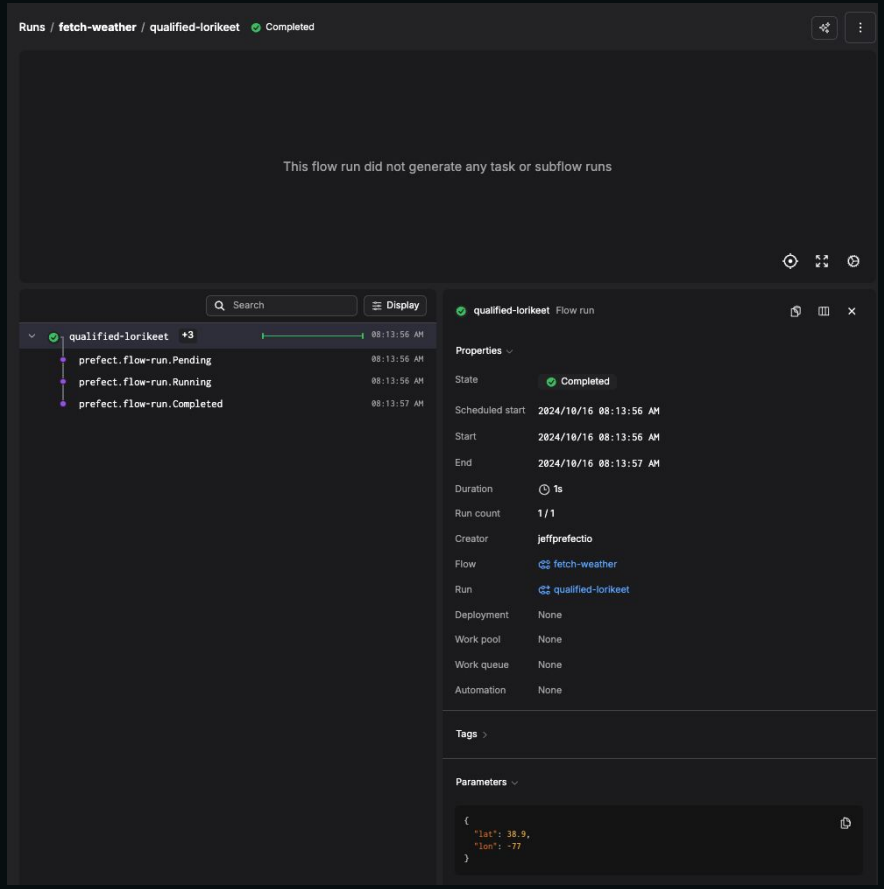
Make it a flow

```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*

```
08:13:56.632 | INFO    | prefect.engine - Created flow run 'qualified-lorikeet' for flow 'fetch-weather'
08:13:56.633 | INFO    | prefect.engine - View at https://app.prefect.cloud/account/9b649228-0419-40e1-9e0d-44
954b5c0ab6/workspace/d137367a-5055-44ff-b91c-6f7366c9e4c4/runs/flow-run/0219f38e-a1b5-437e-b271-749c22a1e929
Forecasted temp C: 10.4 degrees
08:13:57.381 | INFO    | Flow run 'qualified-lorikeet' - Finished in state Completed()
```

Check it out your flow run from the **Runs** page 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 Python 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 flow function's *.serve()* method

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


.serve() method

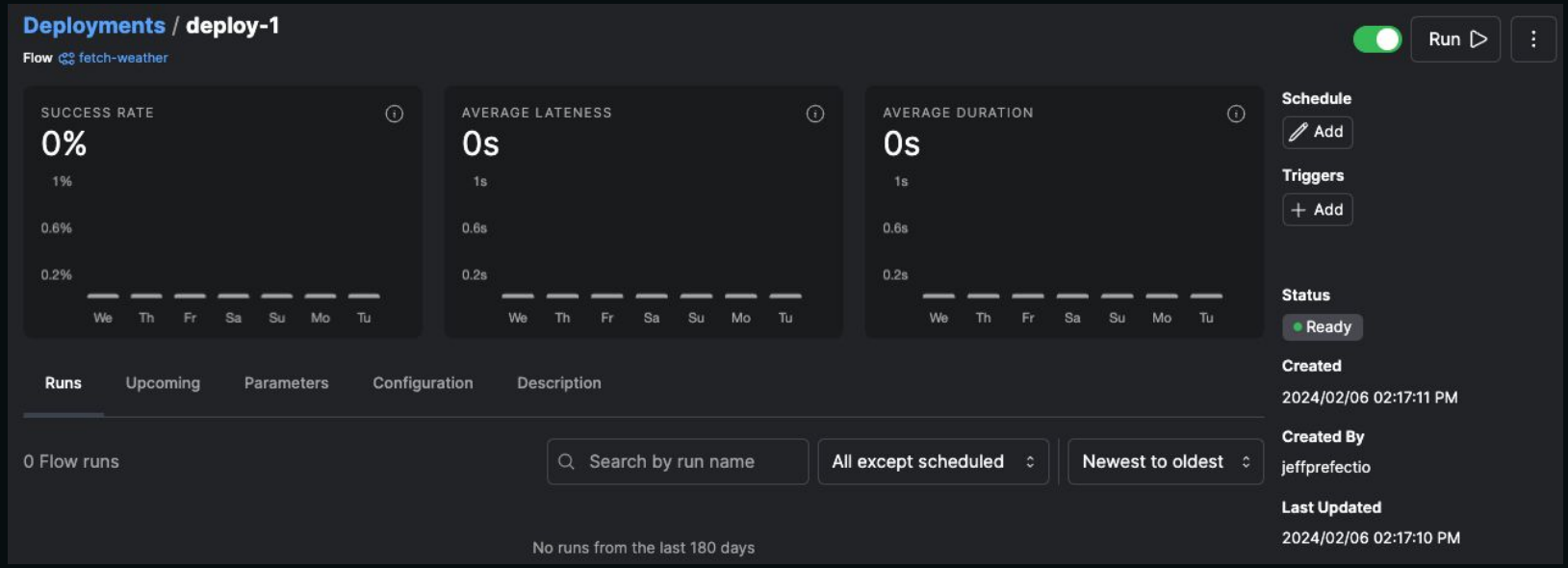
Run the script - creates a deployment and starts a process that polls for scheduled runs

```
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!

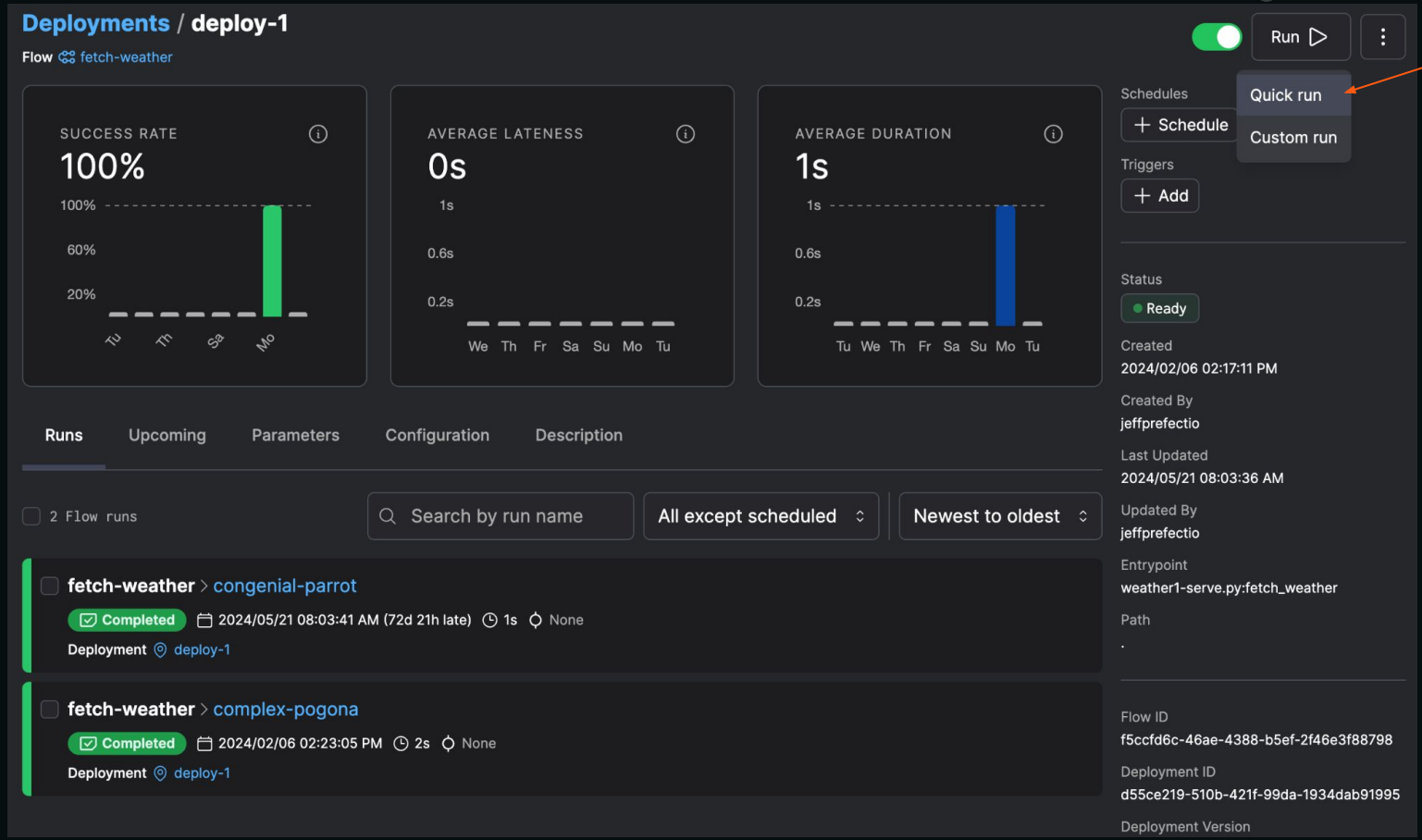
See the deployment in the UI

Deployment page

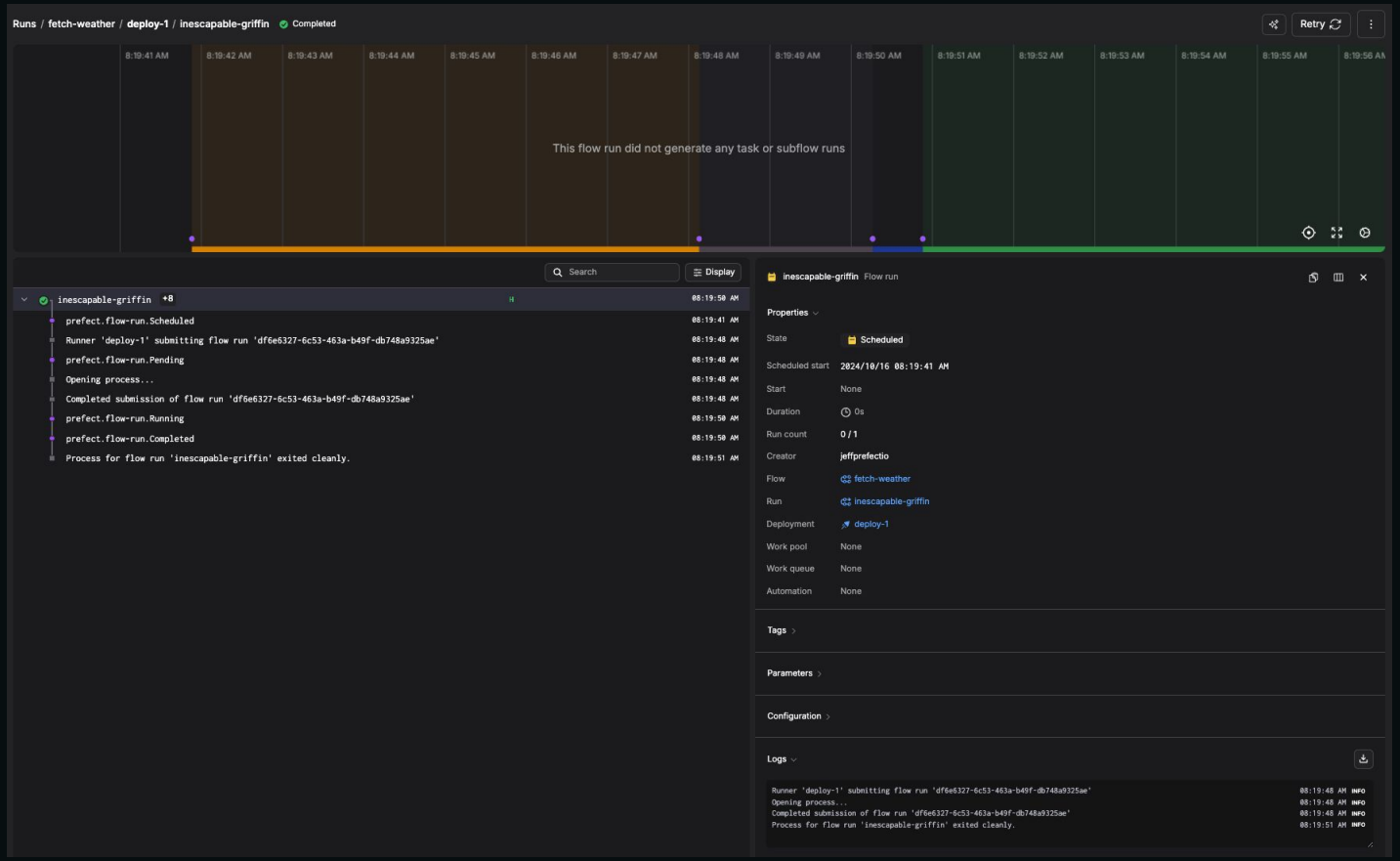


Run a deployment

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

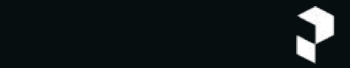


View the flow run logs in the UI (or CLI)



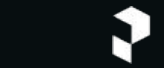
Run deployment manually from CLI

prefect deployment run my_entrypoint_flow:my_deployment



.serve()

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

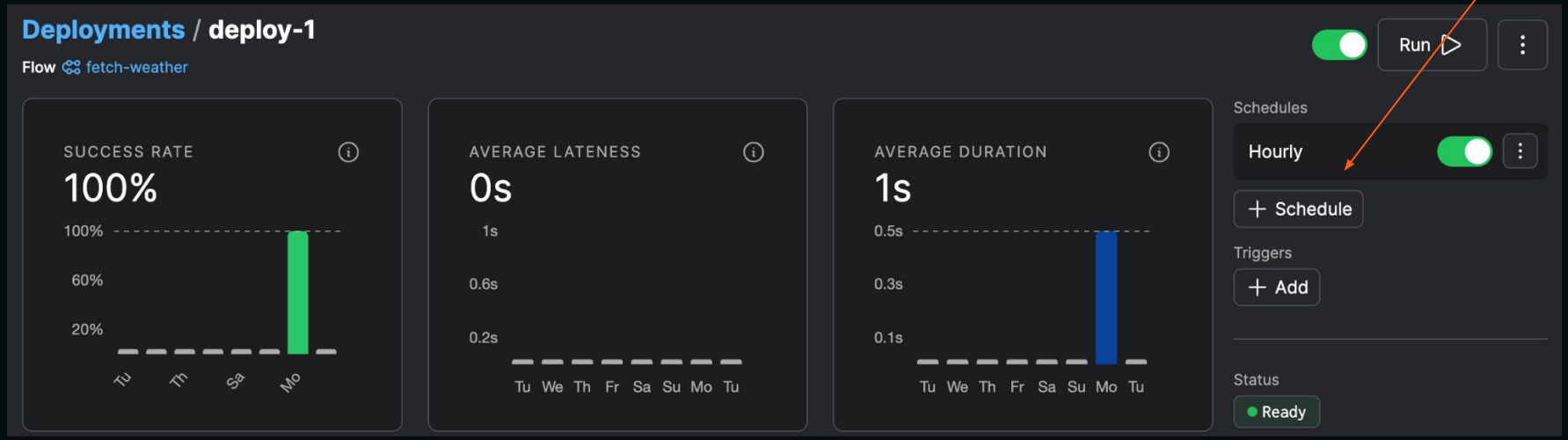


Scheduling

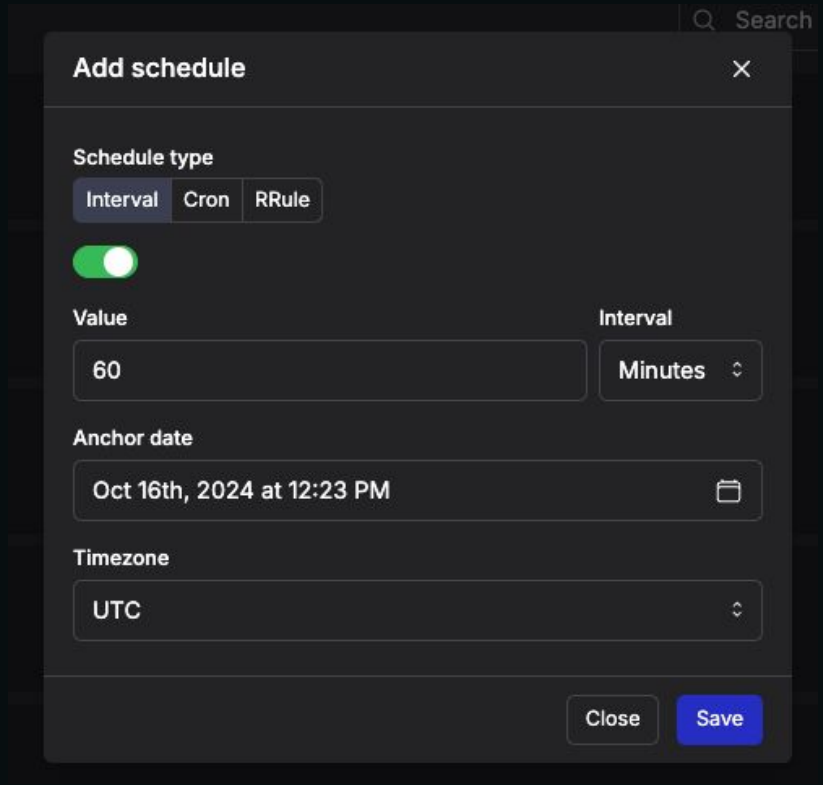
Create a deployment schedule

1. When create deployment
2. After deployment creation in UI or CLI

Create, pause, and delete schedules from UI



Create schedule from UI



Add schedule when create 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

RRule

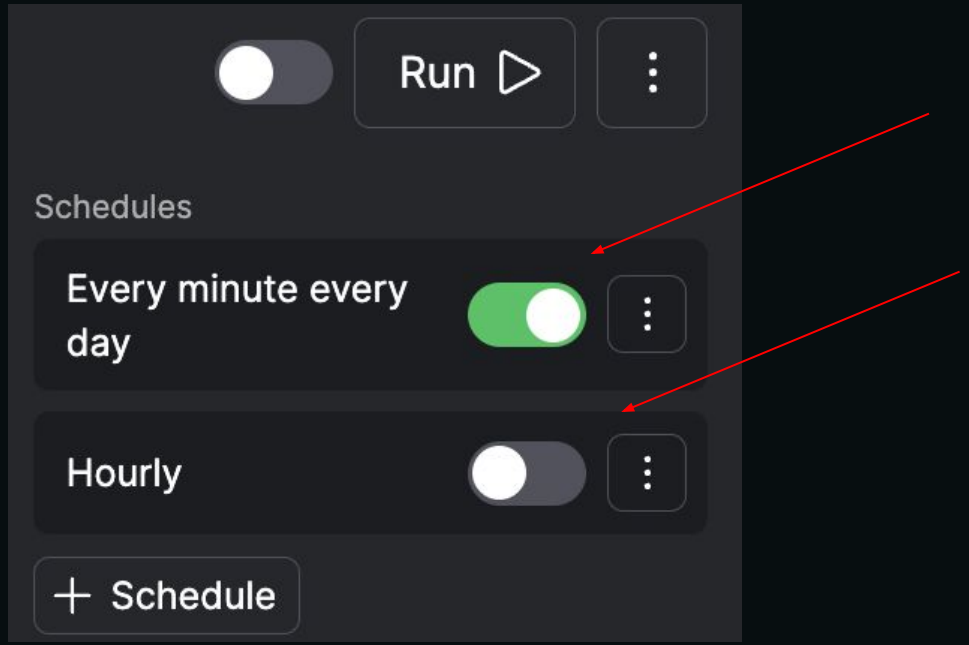
RRule cheat sheet: https://jkbrzt.github.io/rrule/

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 deployment schedules from UI



Note ⏸️

Shutting down your *serve* process 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

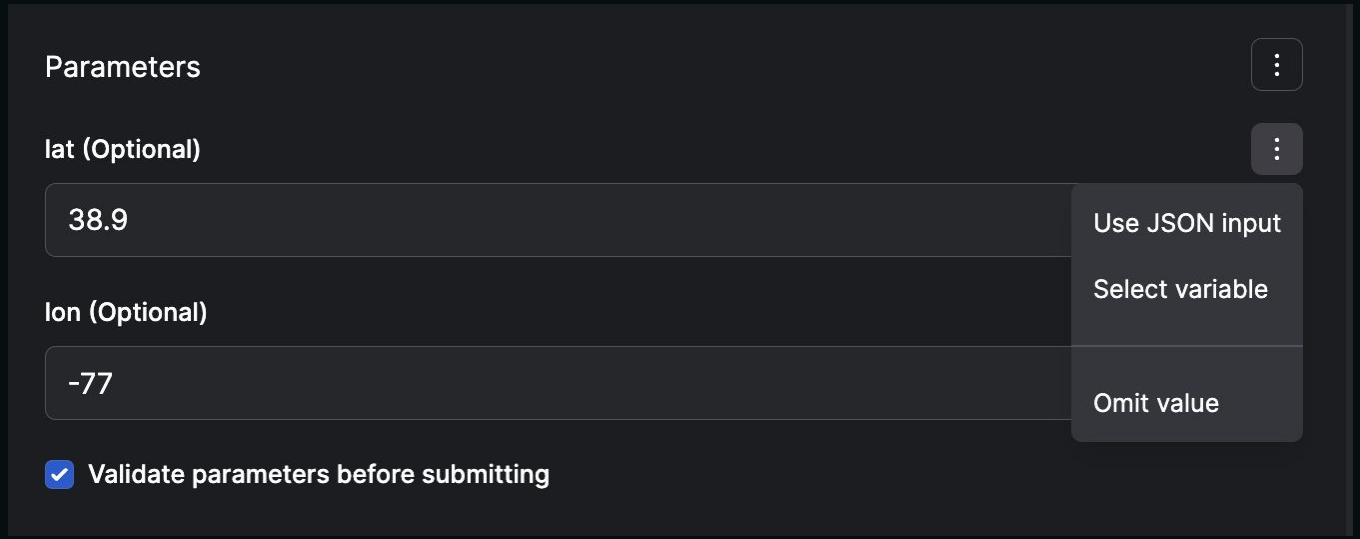
Parameters at deployment creation time

Can specify in *.serve()*

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


Parameters in the UI at runtime

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



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

Use the docs

docs.prefect.io

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*, profiles
- From Python function to Prefect flow
- Create a deployment with *flow.serve()*
- Run a deployment
- Create and pause schedules
- Resources: docs, 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) - test now
3. 💻 One person shares screen (if need to leave and reenter Zoom to enable screen sharing, do now)
4. 👩‍💻 Everyone codes
5. 🙋 Ask a question if you don’t follow something
6. 😌 Low-pressure, welcoming environment: lean in

Course GitHub Repository

Resource for labs:

github.com/PrefectHQ/pacc-2024-v6

101 Lab

Use Open-Meteo API to fetch your first weather forecast:

- 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

102 Agenda

- Tasks
- Logging - observe
- Runtime context - introspect runs
- States - understand your workflow state
- Retries - automatically retry on failure
- Blocks - save configuration with a handy form or code
- Integrations - pre-built, ready to use libraries
- More helpful resources

Tasks

Tasks

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

- Task retries
- Caching
- Async convenience

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"
```

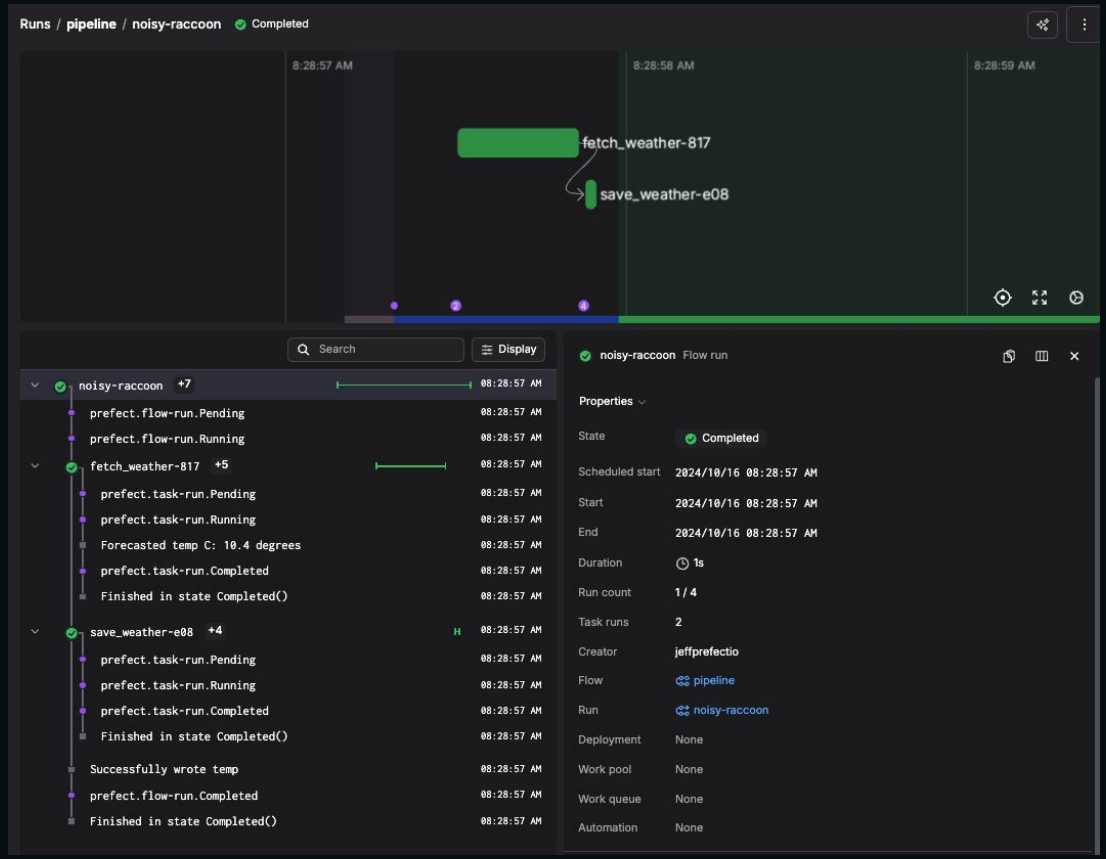
Pipeline flow function

```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

```
08:28:57.233 | INFO    | prefect.engine - Created flow run 'noisy-raccoon' for flow 'pipeline'
08:28:57.236 | INFO    | prefect.engine - View at https://app.prefect.cloud/account/9b649228-0419-40e1-9e0d-44
954b5c0ab6/workspace/d137367a-5055-44ff-b91c-6f7366c9e4c4/runs/flow-run/0edd8b17-3476-4372-8709-876945f2e4f0
08:28:57.855 | INFO    | Task run 'fetch_weather-817' - Forecasted temp C: 10.4 degrees
08:28:57.863 | INFO    | Task run 'fetch_weather-817' - Finished in state Completed()
08:28:57.892 | INFO    | Task run 'save_weather-e08' - Finished in state Completed()
08:28:57.894 | INFO    | Flow run 'noisy-raccoon' - Successfully wrote temp
08:28:57.998 | INFO    | Flow run 'noisy-raccoon' - Finished in state Completed()
```

Visualize dependencies in the UI

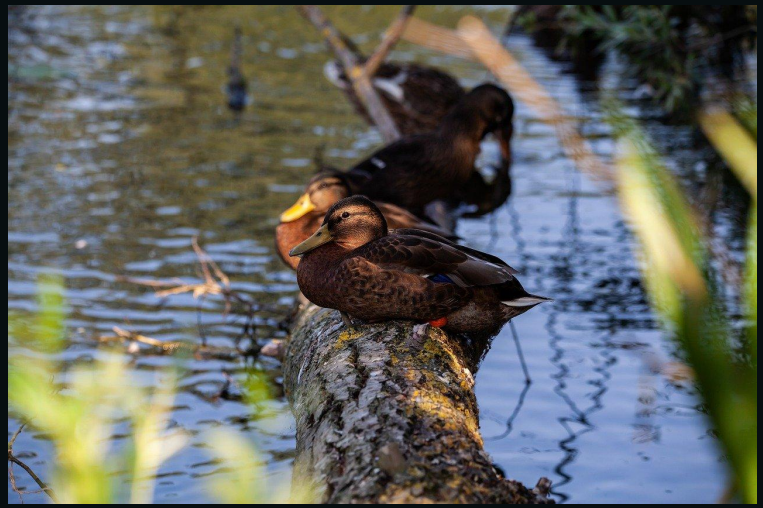


Tasks dos and don’ts

- ⛔ Don’t pass huge amounts of data between tasks
- ✅ Do keep tasks small
- 🙂 You can use Prefect tasks as a replacement for Celery tasks. Tasks can run outside flows and call other tasks.

Note: Prefect is super Pythonic - conditionals are 👍

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)

prefect config set PREFECT_LOGGING_LOG_PRINTS = True

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:

```
08:34:50.681 | INFO    | prefect.engine - Created flow run 'great-snake' for flow 'log-example-flow'
08:34:50.683 | INFO    | prefect.engine - View at https://app.prefect.cloud/account/9b649228-0419-40
d137367a-5055-44ff-b91c-6f7366c9e4c4/runs/flow-run/09f93c71-5aca-46c5-b08c-dffc71c570e1
08:34:50.897 | INFO    | Flow run 'great-snake' - INFO level log message.
08:34:51.079 | INFO    | Flow run 'great-snake' - Finished in state Completed()
```


Logging

Output with **DEBUG** logging level set:

```
08:32:46.176 | DEBUG   | prefect.profiles - Using profile 'local'
08:32:46.589 | INFO    | prefect.engine - Created flow run 'burrowing-caiman' for flow 'log-example-flow'
08:32:46.590 | INFO    | prefect.engine - View at https://app.prefect.cloud/account/9b649228-0419-40e1-9e0d-44954b5c0ab6/wo
d137367a-5055-44ff-b91c-6f7366c9e4c4/runs/flow-run/44aac3f2-27ec-4a6c-80ff-fa7330ee734c
08:32:46.766 | DEBUG   | prefect.task_runner.threadpool - Starting task runner
08:32:46.779 | DEBUG   | Flow run 'burrowing-caiman' - Executing flow 'log-example-flow' for flow run 'burrowing-caiman'...
08:32:46.788 | INFO    | Flow run 'burrowing-caiman' - INFO level log message.
08:32:46.789 | DEBUG   | Flow run 'burrowing-caiman' - You only see this message if the logging level is set to DEBUG. 🙂
08:32:46.799 | DEBUG   | prefect.client - Connecting to API at https://api.prefect.cloud/api/accounts/9b649228-0419-40e1-9e
b5c0ab6/workspaces/d137367a-5055-44ff-b91c-6f7366c9e4c4/
08:32:46.912 | DEBUG   | prefect.task_runner.threadpool - Stopping task runner
08:32:46.913 | INFO    | Flow run 'burrowing-caiman' - Finished in state Completed()
```


prefect.runtime

prefect.runtime

Module for runtime context access.

Useful for labeling, logs, etc.

Includes:

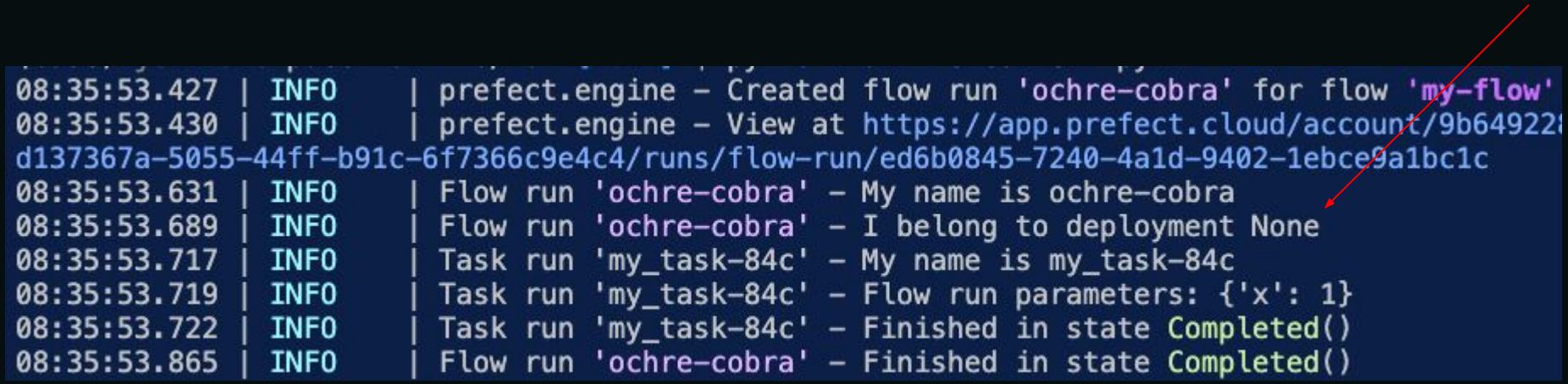
- ***deployment:*** info about current deployment
- ***flow_run***: info about current flow run
- ***task_run***: info about current task run

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.



States

Prefect flow run states

What’s the state of your workflows?

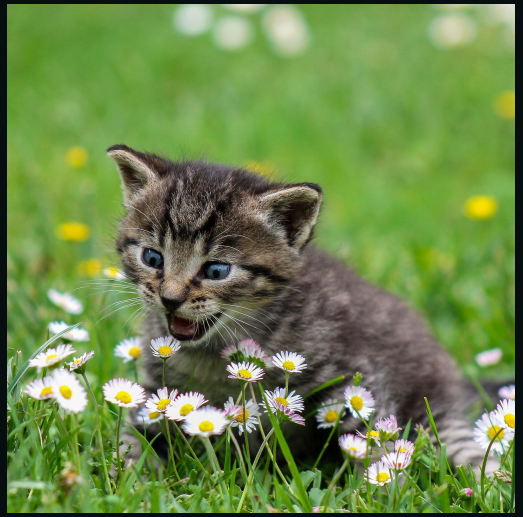
Prefect flow run states: non-terminal

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 (15 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 execute, 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 completing successfully.
`Paused`	`PAUSED`	No	The run code has stopped executing until it receives manual approval to proceed.
`Cancelling`	`CANCELLING`	No	The infrastructure on which the code was running is being cleaned up.

Prefect flow run states: terminal

`Cancelled`	`CANCELLED`	Yes	The run did not complete because a user determined that it should not.
`Completed`	`COMPLETED`	Yes	The run completed successfully.
`Cached`	`COMPLETED`	Yes	The run result was loaded from a previously cached value.
`RolledBack`	`COMPLETED`	Yes	The run completed successfully but the transaction rolled back and executed rollback hooks.
`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.

Retries



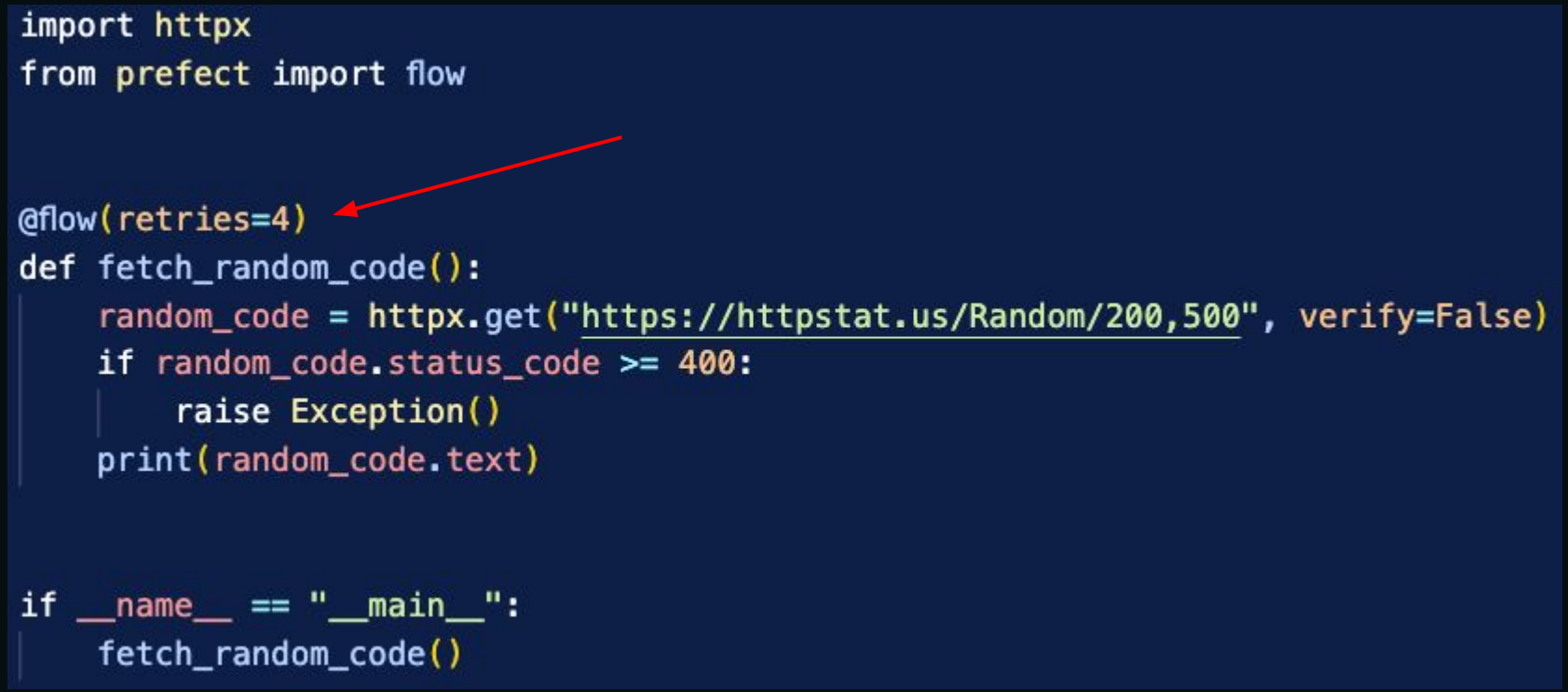
Retries - guard against failure

- Automatically retry a task or flow
- Specify in decorator

@task(retries=2)

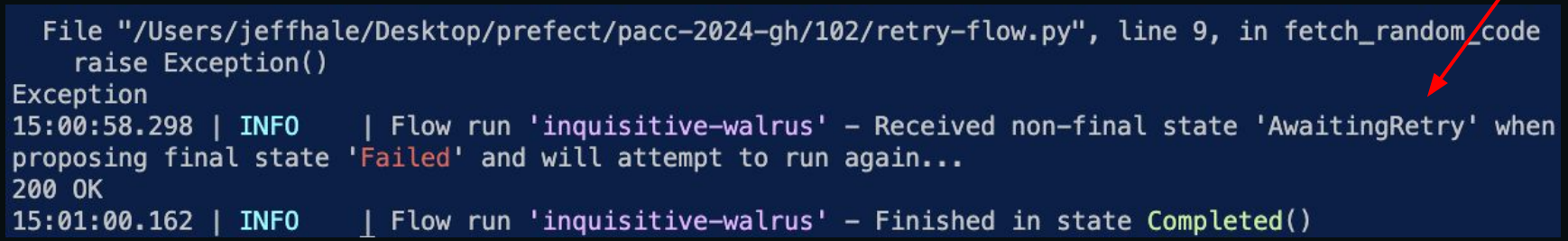
@flow(retries=3)

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* for tasks.

Blocks 

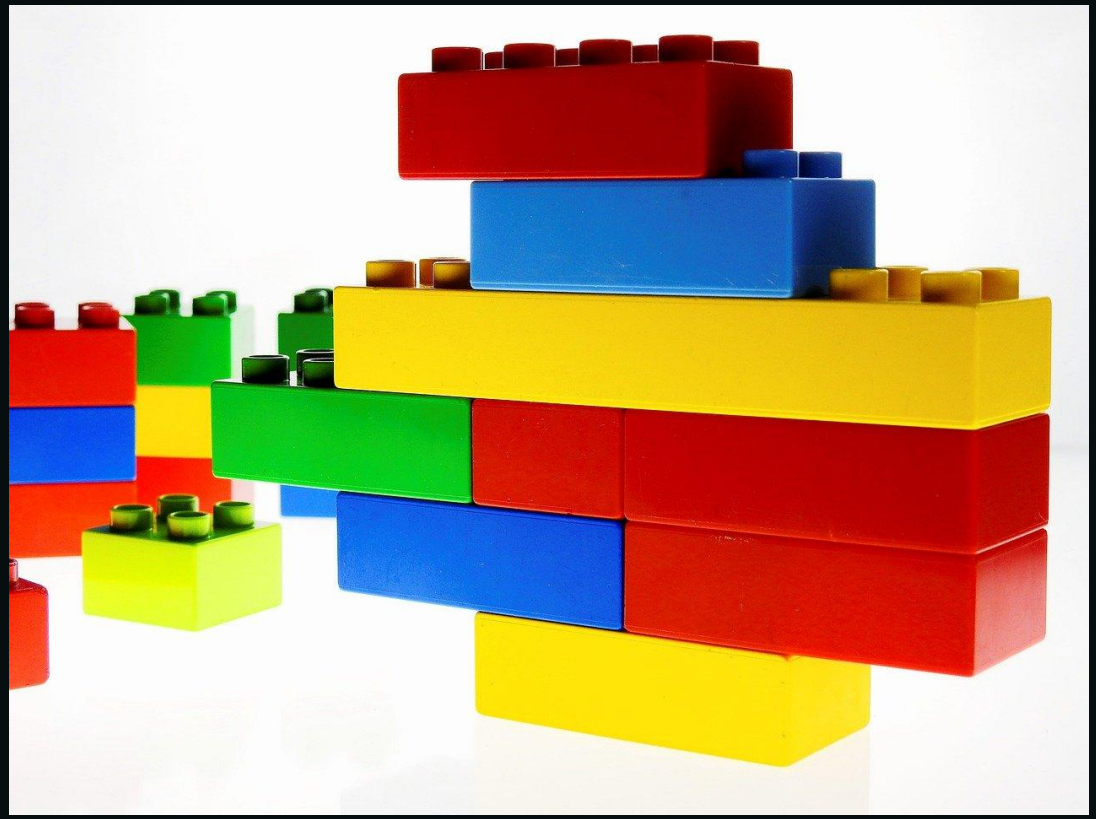
Blocks

Configuration

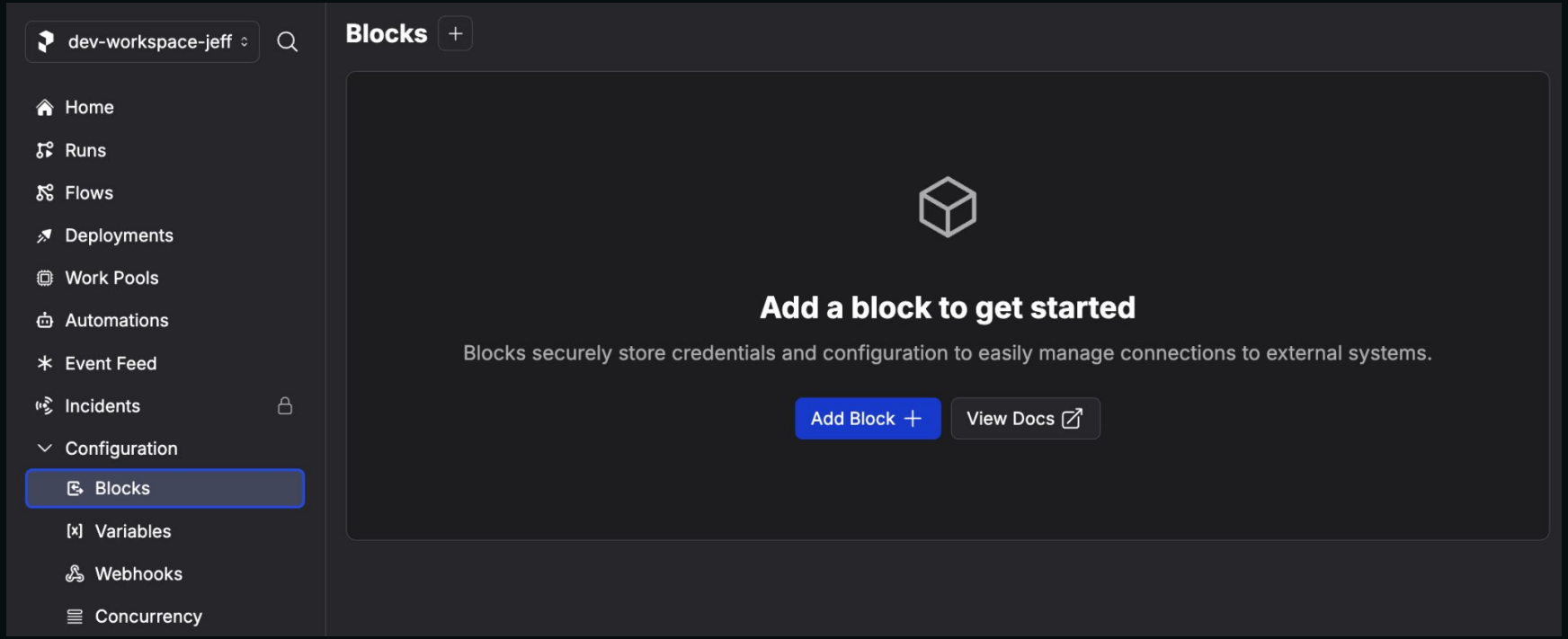
+

Code

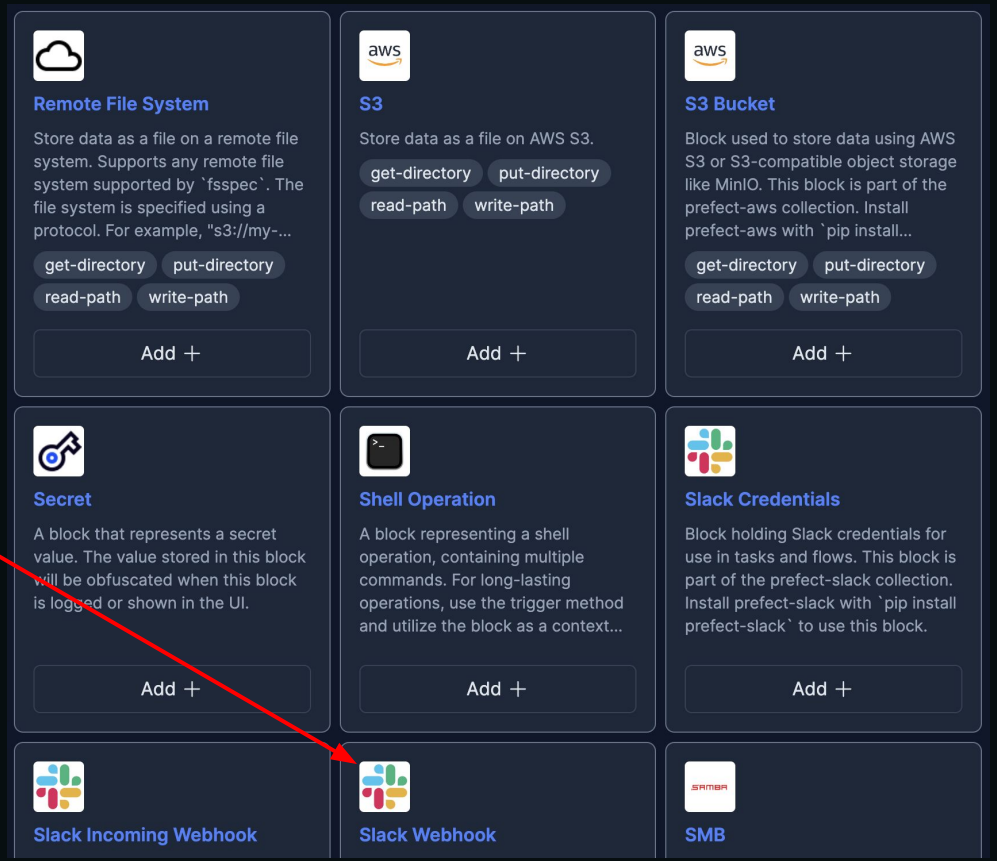
Useful for storing configuration for connecting to external systems



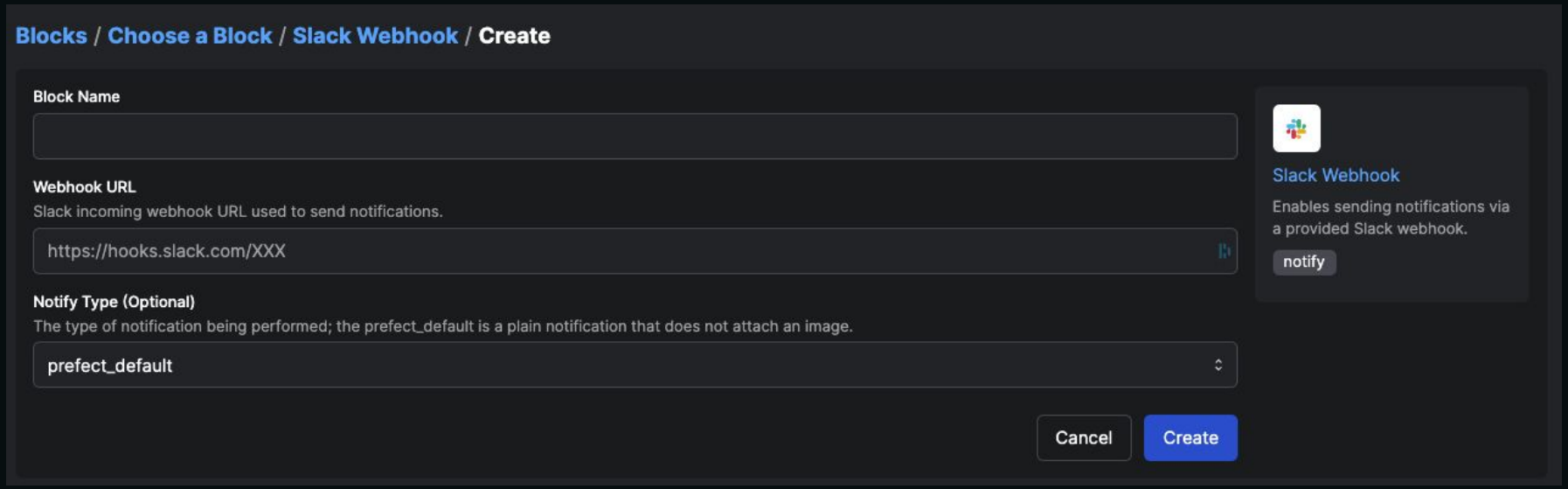
Create a block from the UI



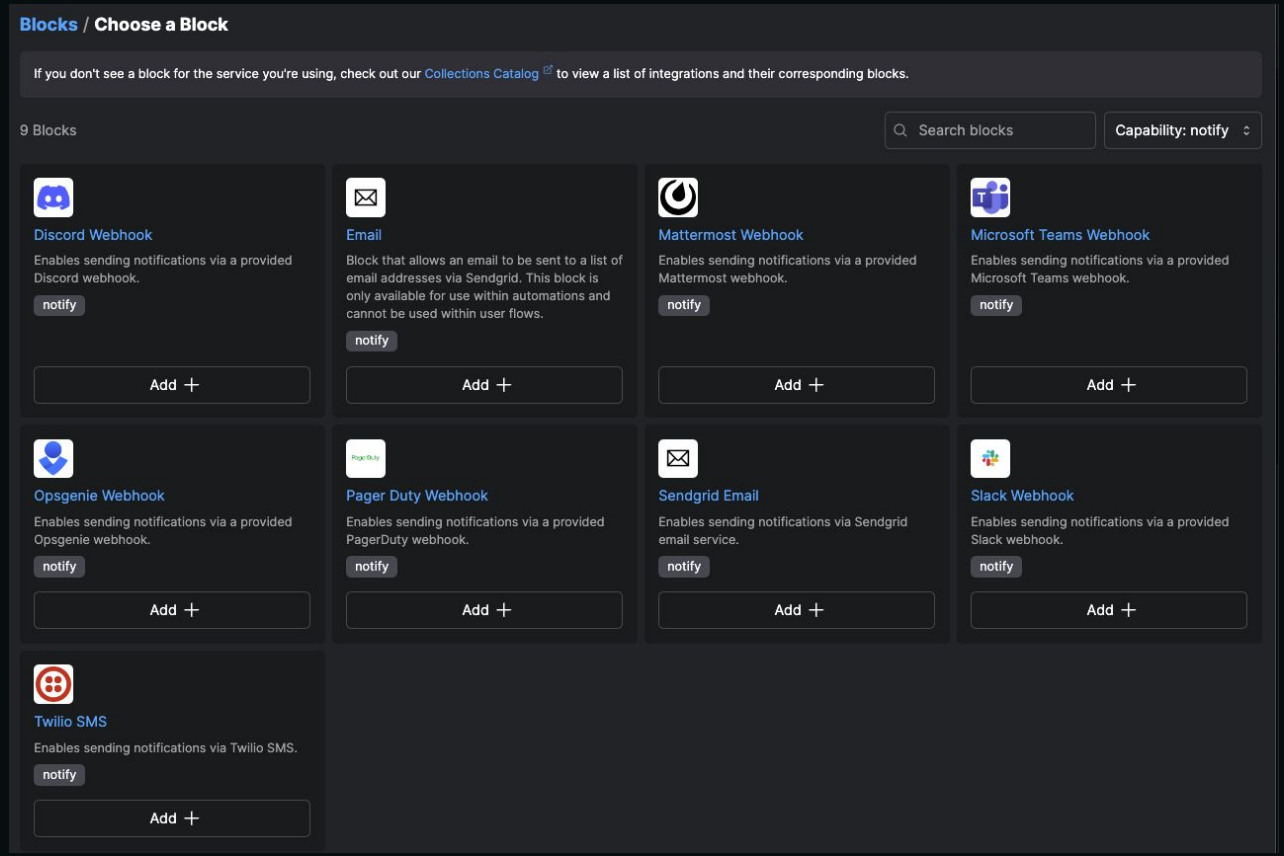
Create a block from the UI - choose a block type



Create a block from the UI



Block types in UI - filter by capability

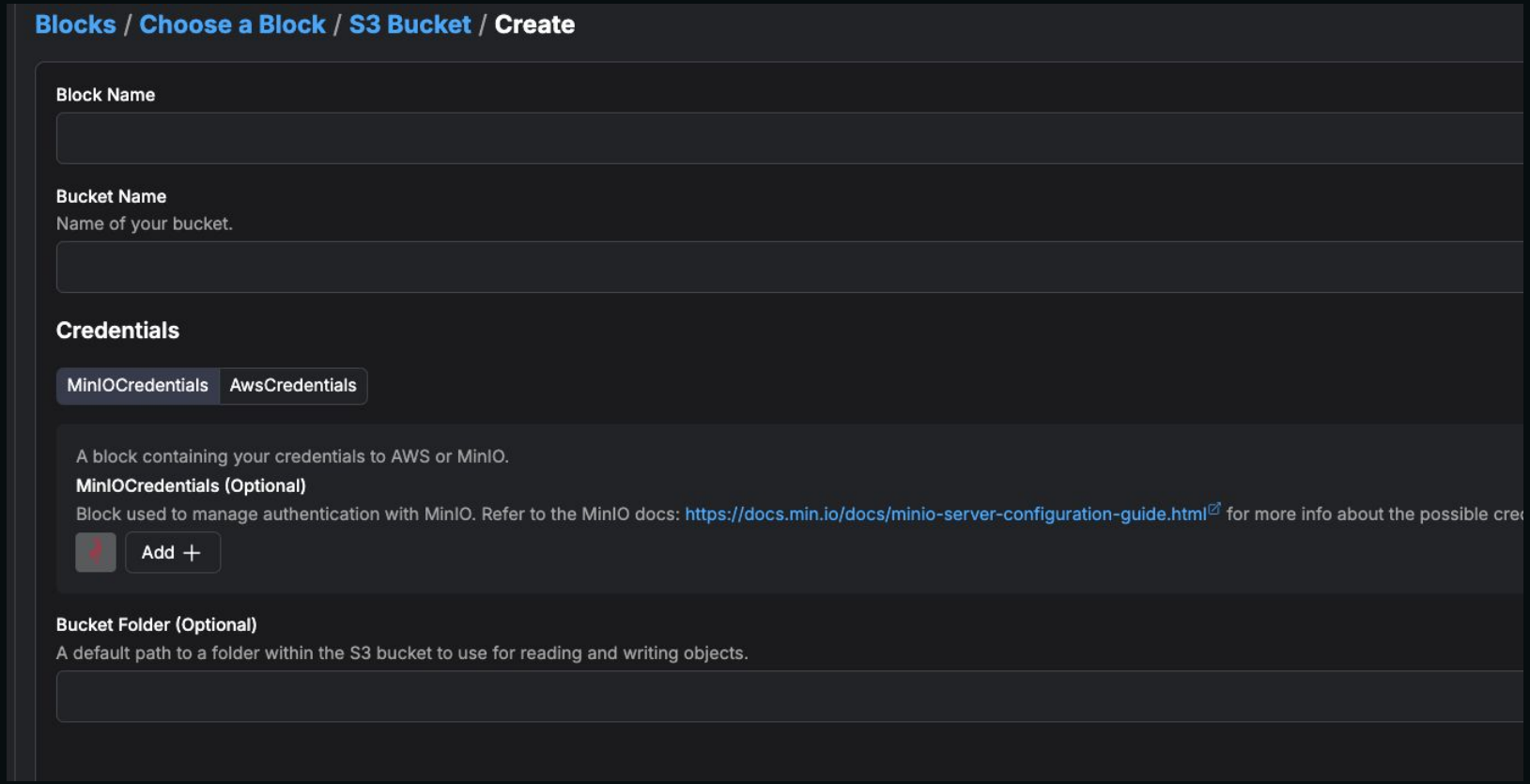


Under the hood, block types are Python classes

Block types are Python classes

```python
class S3Bucket(WritableFileSystem, WritableDeploymentStorage, ObjectStorageBlock):
    """
    Block used to store data using AWS S3 or S3-compatible object storage like MinIO

    Attributes:
        bucket_name: Name of your bucket.
        credentials: A block containing your credentials to AWS or MinIO.
        bucket_folder: A default path to a folder within the S3 bucket to use
            for reading and writing objects.
    """

    _logo_url = "https://cdn.sanity.io/images/3ugk85nk/production/d74b16fe84ce626345
    _block_type_name = "S3 Bucket"
    _documentation_url = (
        "https://prefecthq.github.io/prefect-aws/s3/#prefect_aws.s3.S3Bucket"  # noq
    )

    bucket_name: str = Field(default=..., description="Name of your bucket.")
```


Block types are Python classes (with nice forms)



Create a block in Python - an instance of the class

```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

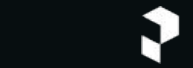
- Nestable
- Stored in server database
- Can create own block types

Integrations

Integrations

Prefect is Python-based and designed for flexibility

Use with most any Python library - no special integration package required



Integrations

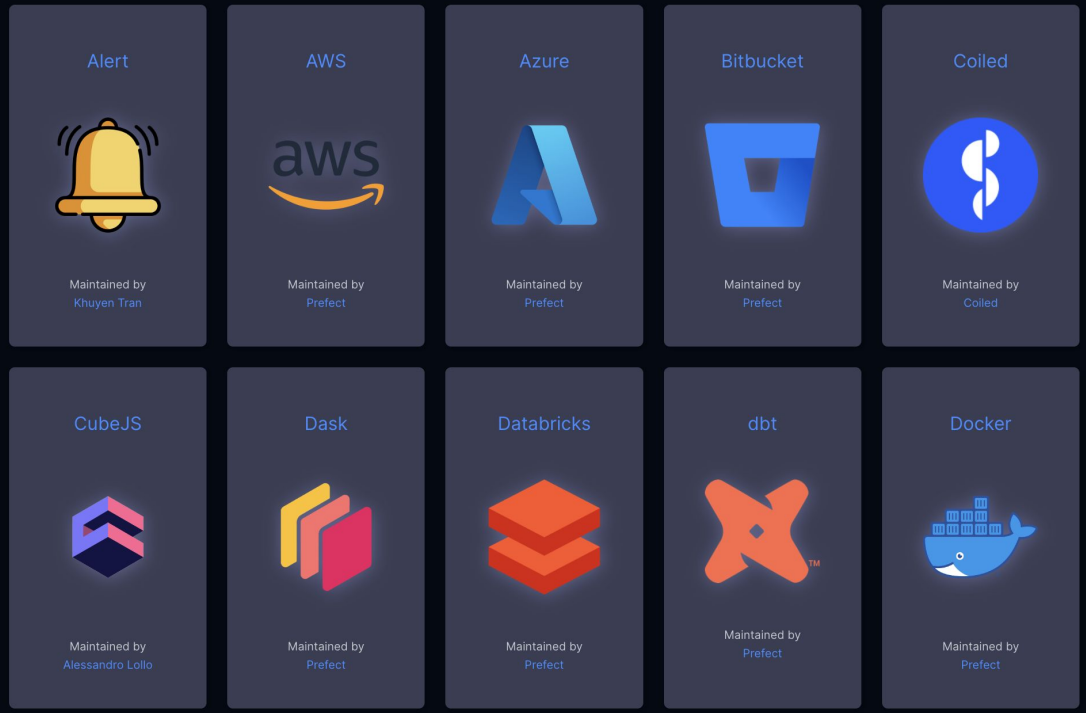
pip-installable Python packages that add convenience

- May contain new block types you'll register
- May contain pre-built tasks or flows

- Can modify or create your own
- Can contribute to the community

Integrations

docs.prefect.io/integrations



More helpful resources

Prefect Community Slack

Helpful Q & A from the community, Prefect staff, & Marvin

#ask-marvin channel for @Marvin LLM in Community Slack

@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 CLI

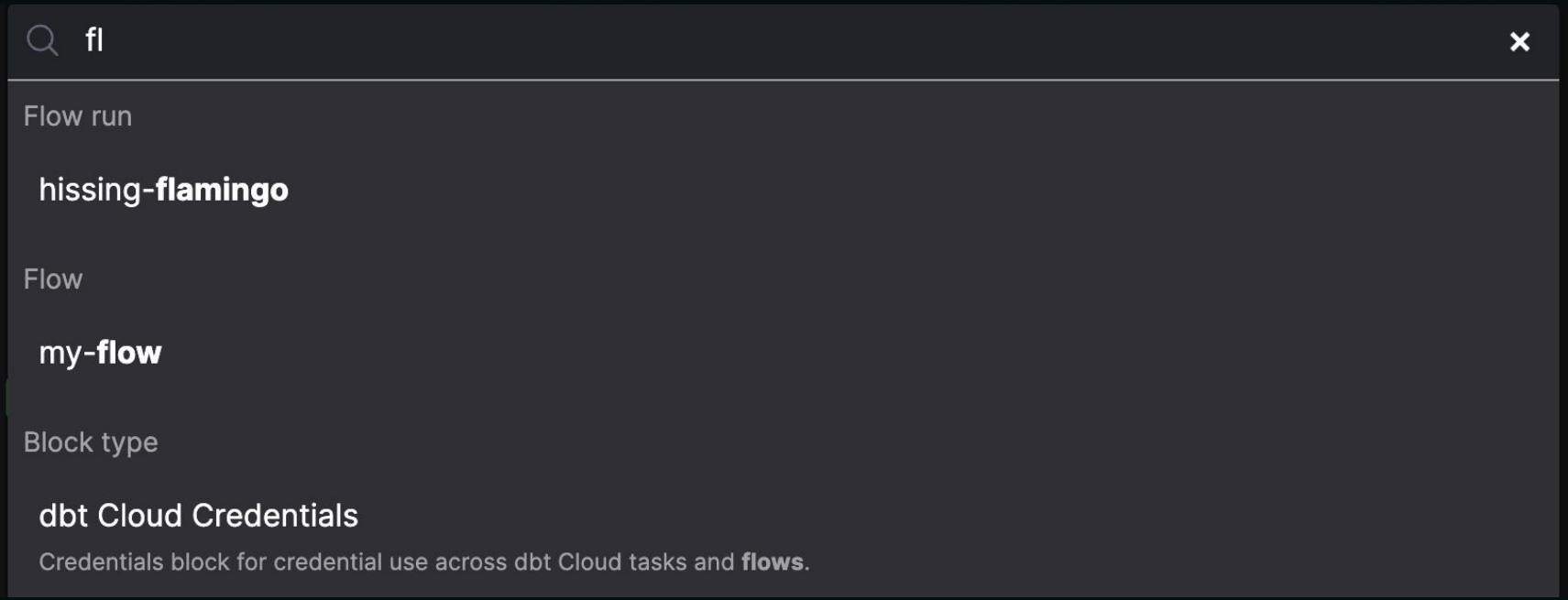
Start commands with *prefect*

End with *--help*

prefect --help

Search in the UI

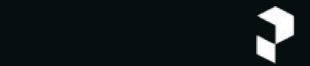
cmd + **k** or 🔍



102 Recap

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

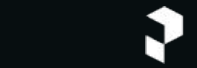
- Tasks
- Logging
- States
- Retries
- Blocks
- Integrations
- More resources: Community Slack, *help*, & search



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

103 Agenda

- Work with data to save time and money
  - Save results
  - Use caching
  - Create 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

```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()
```


Persisted results

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

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/latest/resources/big-data

Caching

Caching

What?

Why?

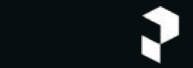
⚠️ task only

Requires persisting results (so must be serializable)

cache_policy - for computing cache keys

INPUTS - rerun only if inputs change

```python
from prefect import task
from prefect.cache_policies import INPUTS


@task(cache_policy=INPUTS, log_prints=True)
def my_cached_task(x: int):
    print(f"Result is: {x + 42}")


my_cached_task(8)  # Task runs
my_cached_task(8)  # Task doesn't run, uses the cached result
my_cached_task(33)  # Task runs
```

Caching with *INPUTS* policy

Task runs two times:

```
12:51:32.389 | INFO    | Task run 'my_cached_task' - Result is: 50
12:51:32.398 | INFO    | Task run 'my_cached_task' - Finished in state Completed()
12:51:32.416 | INFO    | Task run 'my_cached_task' - Finished in state Cached(type=COMPLETED)
12:51:32.441 | INFO    | Task run 'my_cached_task' - Result is: 75
12:51:32.443 | INFO    | Task run 'my_cached_task' - Finished in state Completed()
```

What happens if run the task again with same three inputs?

Caching

```
12:53:48.663 | INFO    | Task run 'my_cached_task' - Finished in state Cached(type=COMPLETED)
12:53:48.684 | INFO    | Task run 'my_cached_task' - Finished in state Cached(type=COMPLETED)
12:53:48.702 | INFO    | Task run 'my_cached_task' - Finished in state Cached(type=COMPLETED)
```

If you delete the cached result (the file stored in *~/.prefect/storage* by default) and rerun the flow, the task will run.

Cache policy options

DEFAULT: task inputs, code definition, and current flow run ID.

INPUTS: only task inputs.

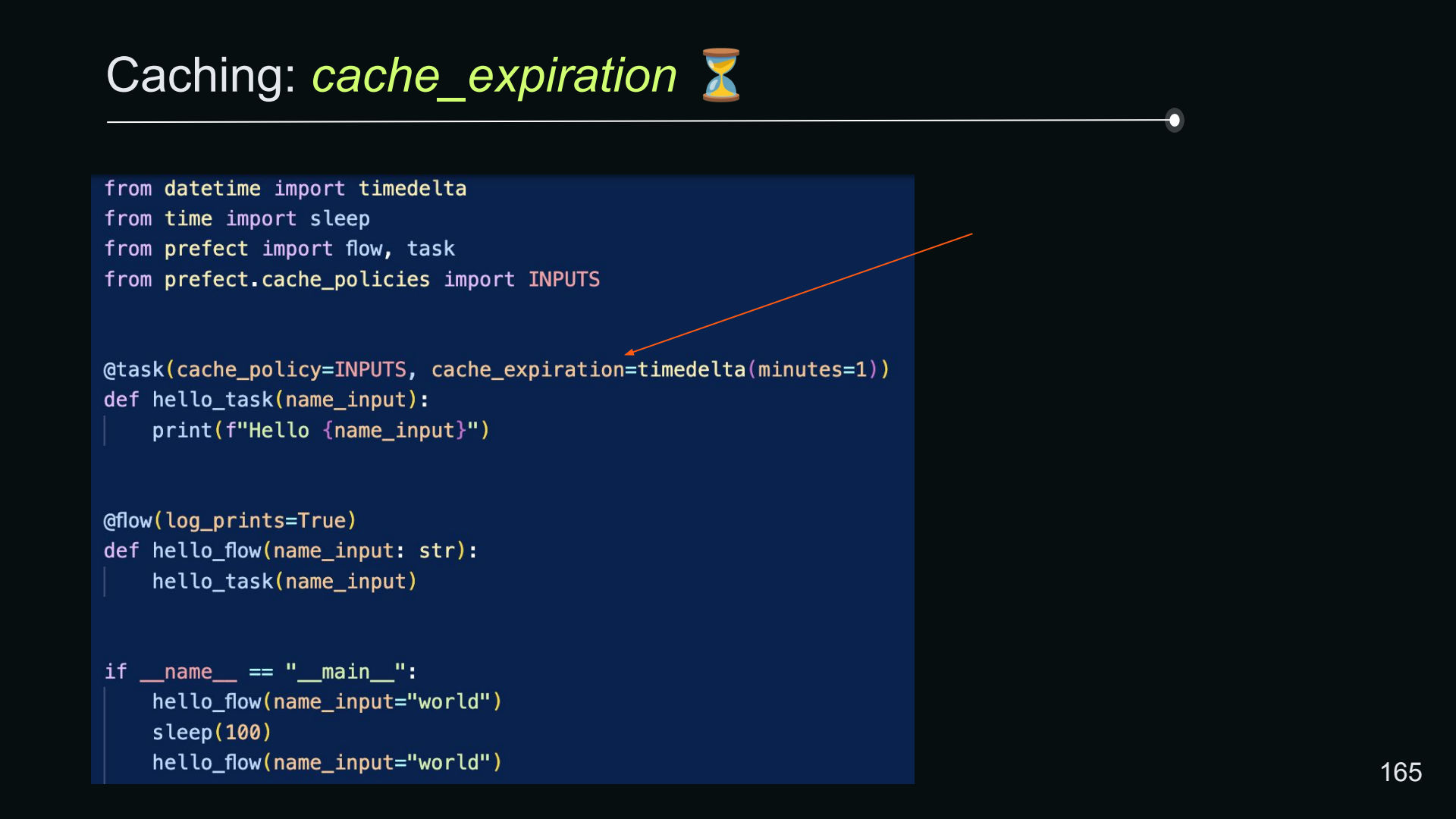
TASK_SOURCE: only task code definition.

FLOW_PARAMETERS: only parameter values provided to parent flow run.

NONE: always returns *None*. Avoids caching and result persistence altogether.

Caching: *cache_expiration* ⏳

```python
from datetime import timedelta
from time import sleep
from prefect import flow, task
from prefect.cache_policies import INPUTS


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


@flow(log_prints=True)
def hello_flow(name_input: str):
    hello_task(name_input)


if __name__ == "__main__":
    hello_flow(name_input="world")
    sleep(100)
    hello_flow(name_input="world")
```



Transaction rollbacks ↩️

Transactions

- Can roll back a side effect if code fails
- Like an undo button to return to previous state

docs.prefect.io/latest/develop/transactions

Transactions

In a flow, call tasks using a *with transaction():* context block

```
@flow
def pipeline(contents: str):
    with transaction():
        write_file(contents)
        quality_test()
```


Transactions

- Can specify action to take if an error is raised
- Decorate a function with *my_task.on_rollback()*

```
@write_file.on_rollback
def del_file(transaction):
    "Deletes file."
    os.unlink("side-effect.txt")
```

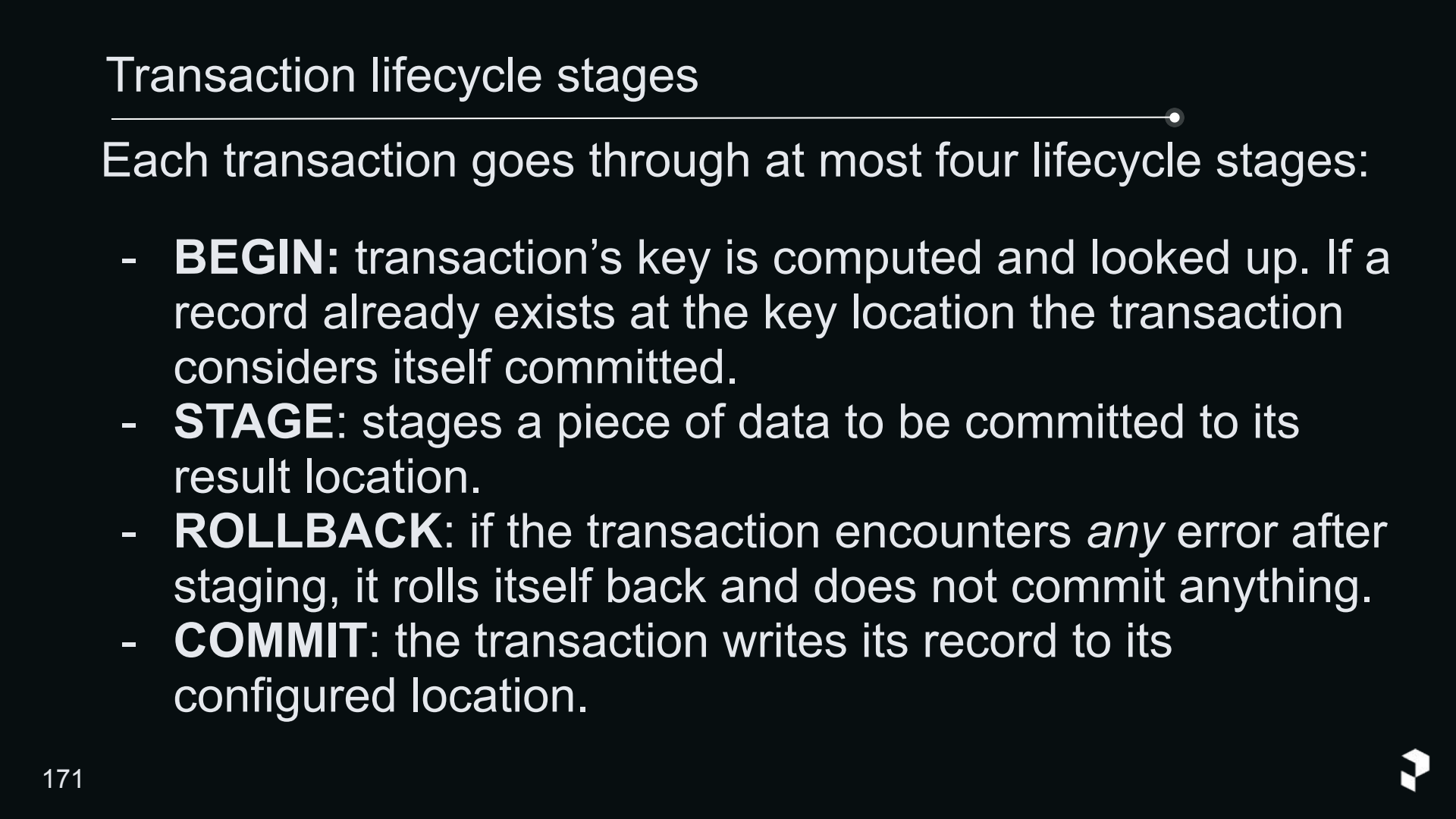

Transactions

```python
from prefect import task, flow
from prefect.transactions import transaction

@task
def write_file(contents: str):
    "Writes to a file."
    with open("side-effect.txt", "w") as f:
        f.write(contents)

@write_file.on_rollback
def del_file(transaction):
    "Deletes file."
    os.unlink("side-effect.txt")

@task
def quality_test():
    "Checks contents of file."
    with open("side-effect.txt", "r") as f:
        data = f.readlines()

    if len(data) < 2:
        raise ValueError("Not enough data!")

@flow
def pipeline(contents: str):
    with transaction():
        write_file(contents)
        quality_test()
```


Transaction lifecycle stages

Each transaction goes through at most four lifecycle stages:

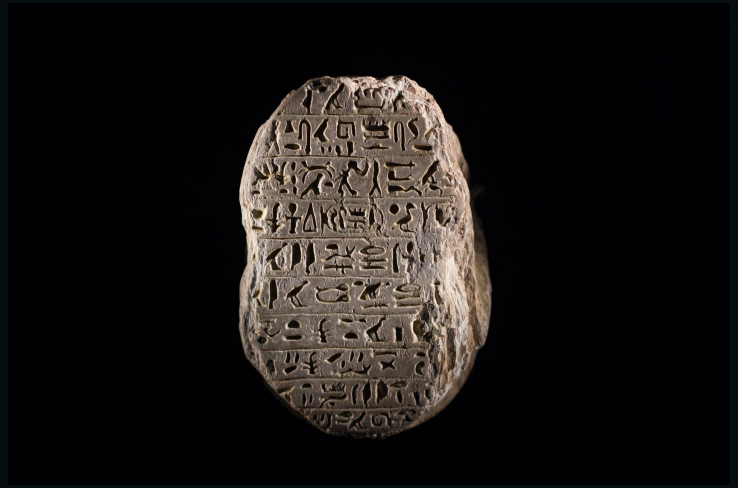
- **BEGIN:** transaction's key is computed and looked up. If a record already exists at the key location the transaction considers itself committed.
- **STAGE**: stages a piece of data to be committed to its result location.
- **ROLLBACK**: if the transaction encounters *any* error after staging, it rolls itself back and does not commit anything.
- **COMMIT**: the transaction writes its record to its configured location.



Artifacts

Artifacts

Communicate with team members through persisted outputs such as Markdown, tables, images, and links



Artifacts

- Meant for human consumption
- Examples:
  - Model scores
  - Data quality checks
  - Reports
- Stored in database & shown in UI
- Custom RBAC can limit user access (Enterprise)

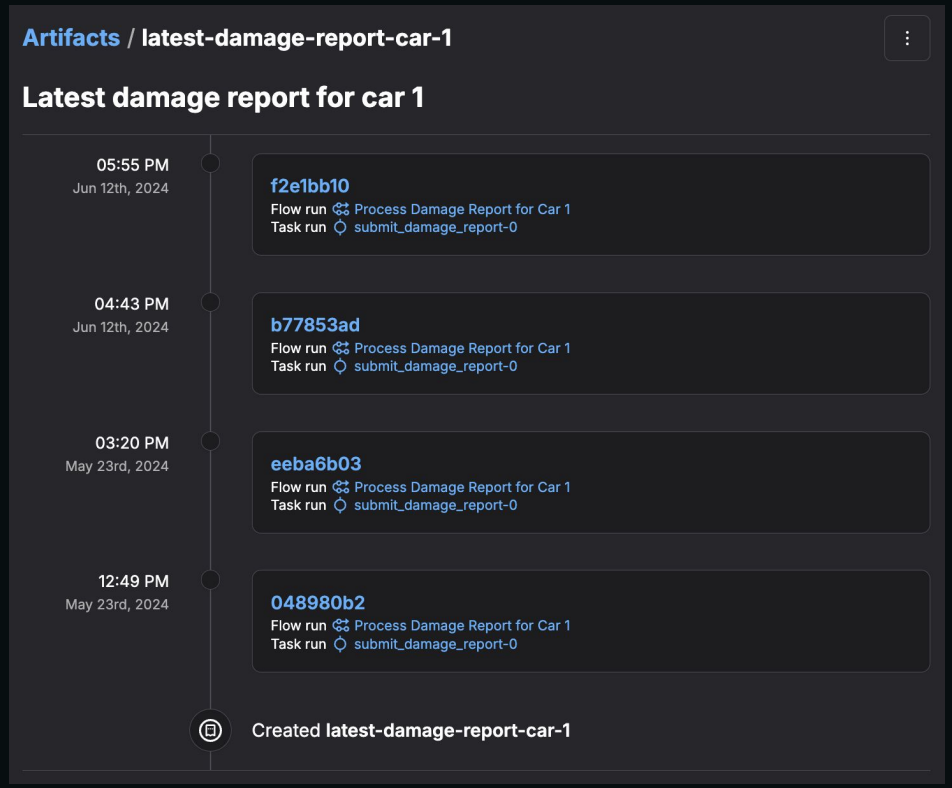
Artifacts - Markdown example of weather report

```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 of weather report

Access from UI: *Runs* timeline

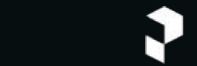
Artifacts are versioned



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

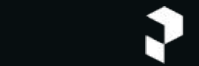
Prefect Cloud - Default Roles (Pro + Enterprise)

Account level

- Owner
- Admin
- Member

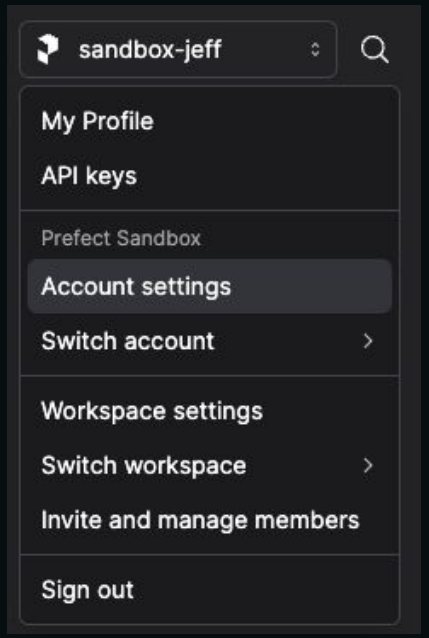
Workspace level

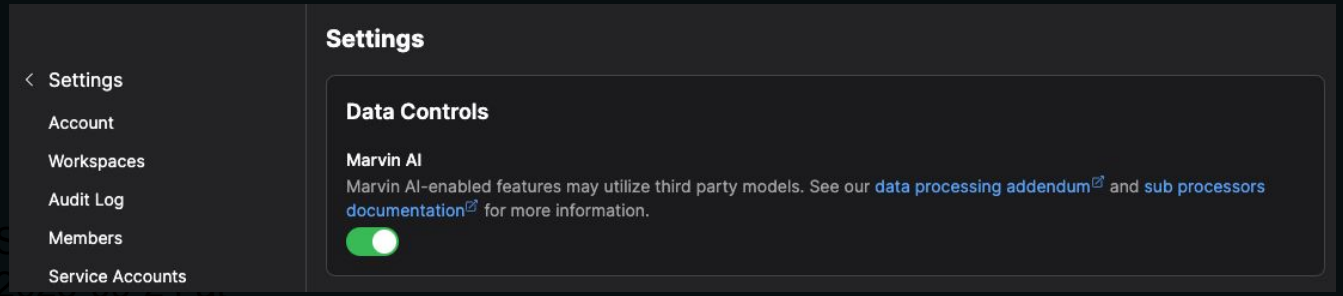
- Owner
- Developer
- Runner
- Viewer
- Worker



Error summaries by Marvin AI

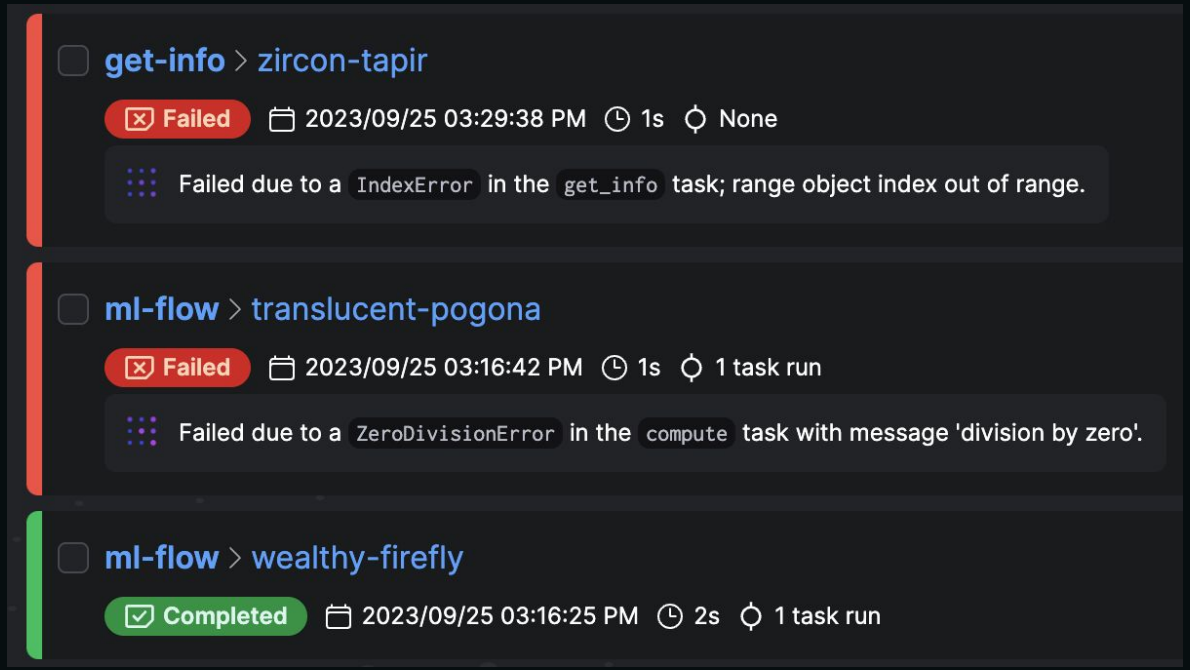
Error summaries by Marvin AI





12.11.58 PM

Error summaries by Marvin AI



Events

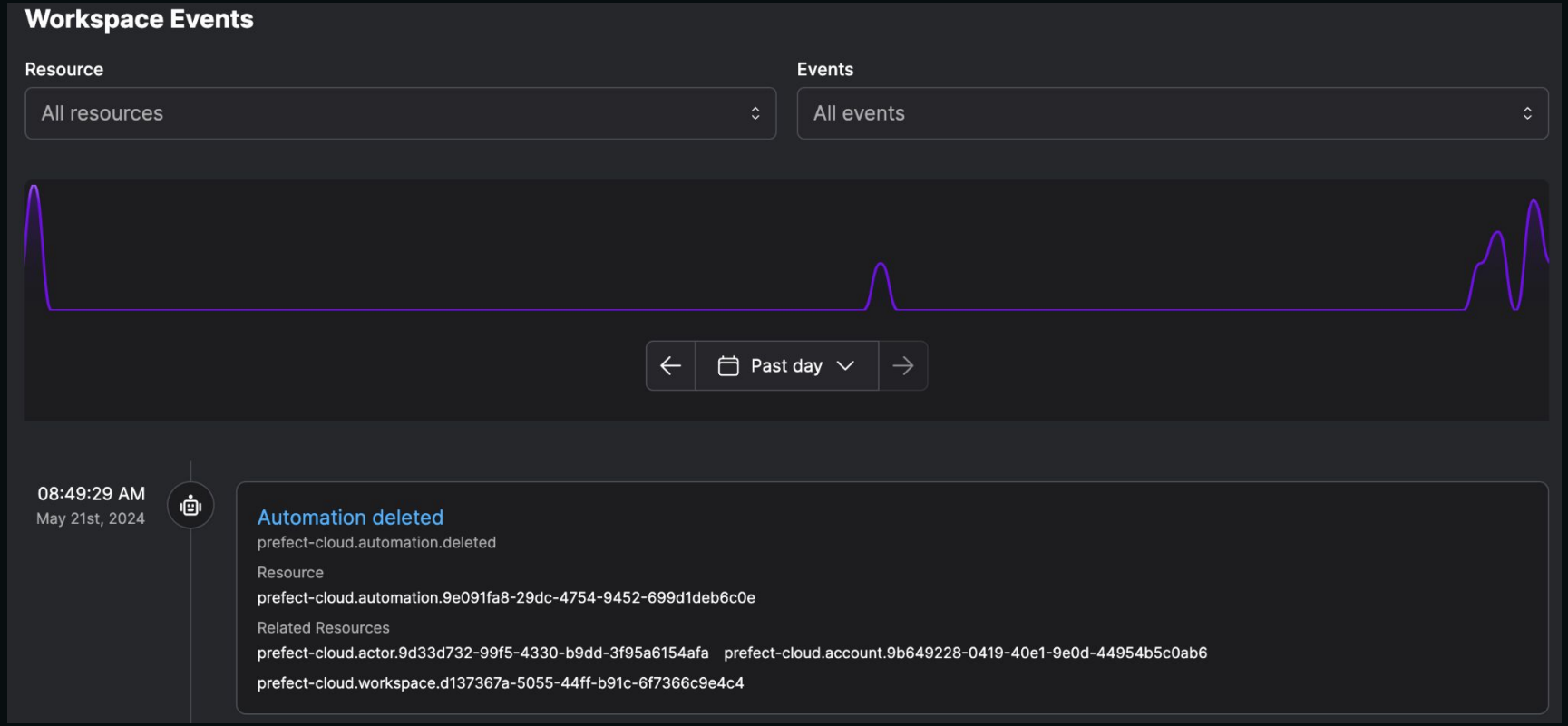
Events

- Record of what happened

Represent:

- API calls
- State transitions
- Changes in environment

Event feed


Workspace Events
Resource
All resources
Events
All events
Past day
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

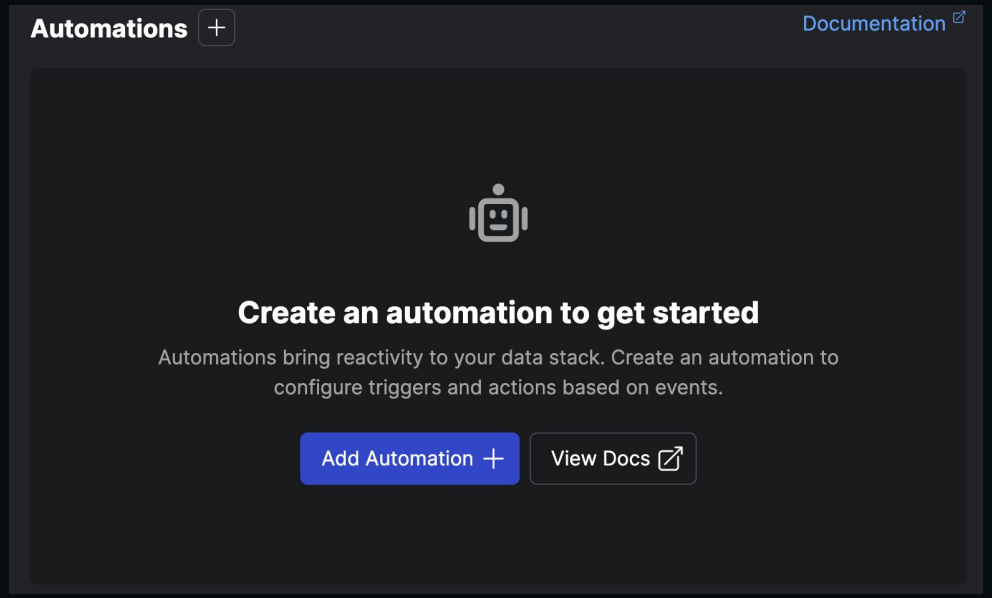
- If *Trigger* happens, do *Action*
- If *Trigger* doesn’t happen in a time period, do *Action*

Automation examples

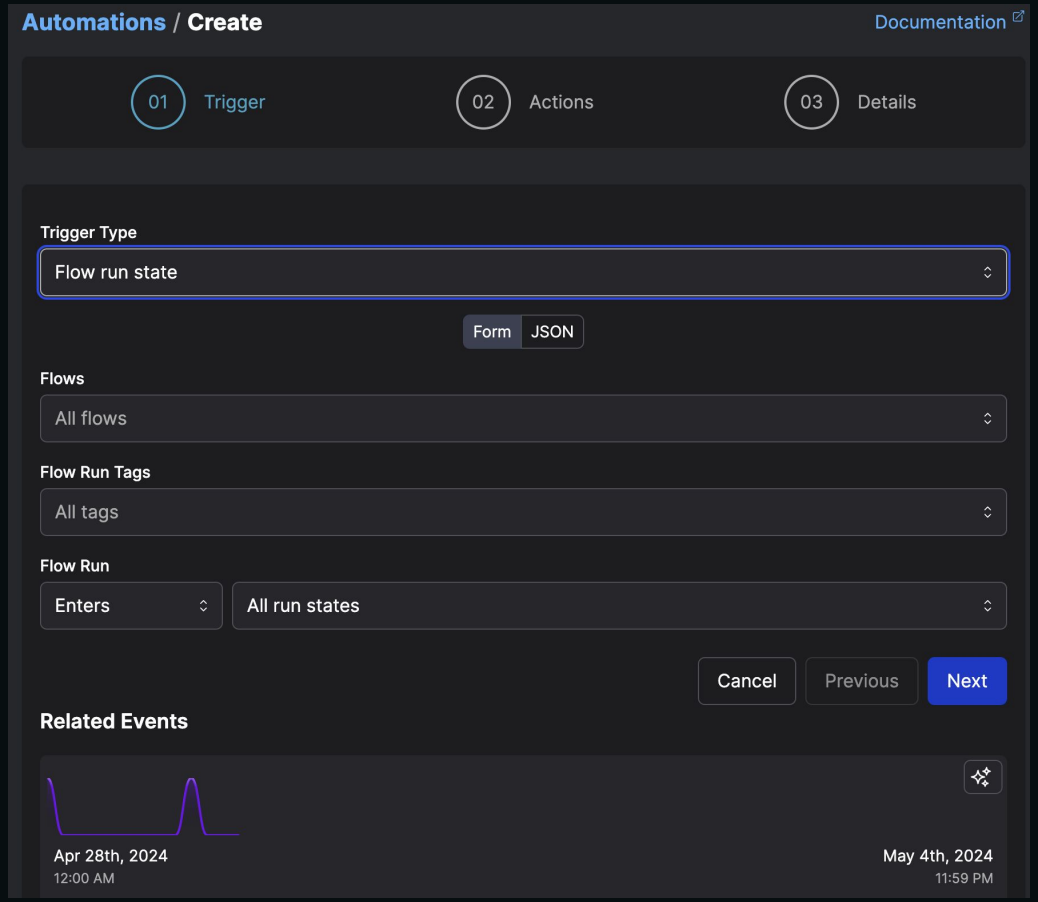
- If a flow run with tag **prod** fails, send an email 📧
- If a data quality check fails, run a deployment to fetch more data 📊
- If a work pool changes state to *Not Ready*, call a webhook 🕸️🪝

Create an automation

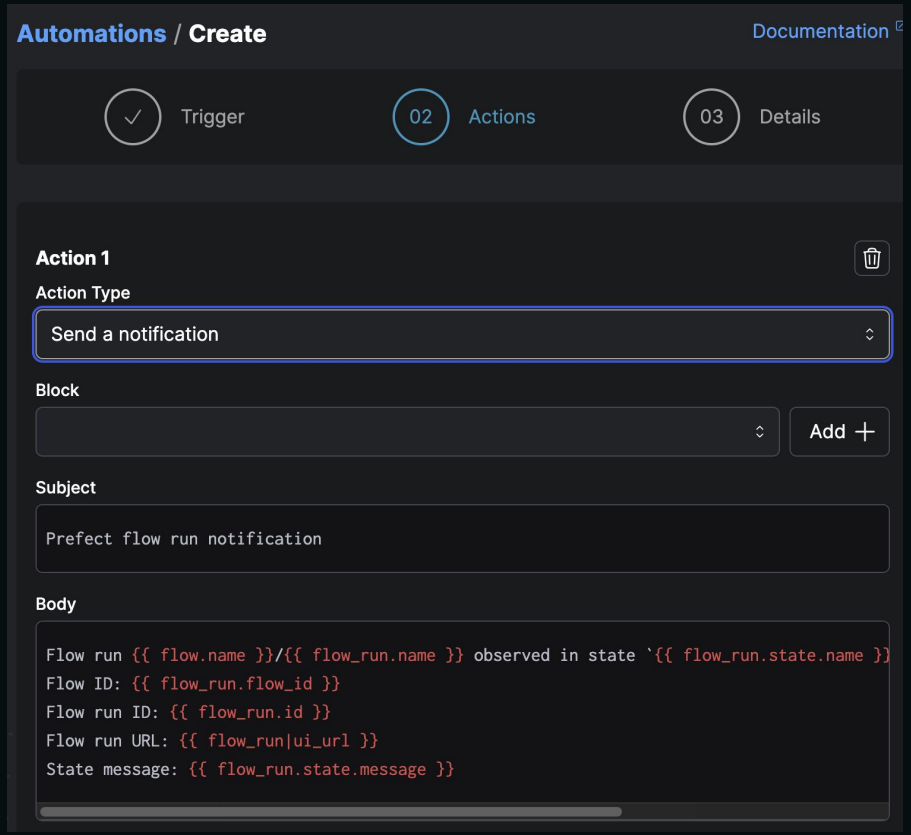
Trigger: flow run failure
Action: notification - email



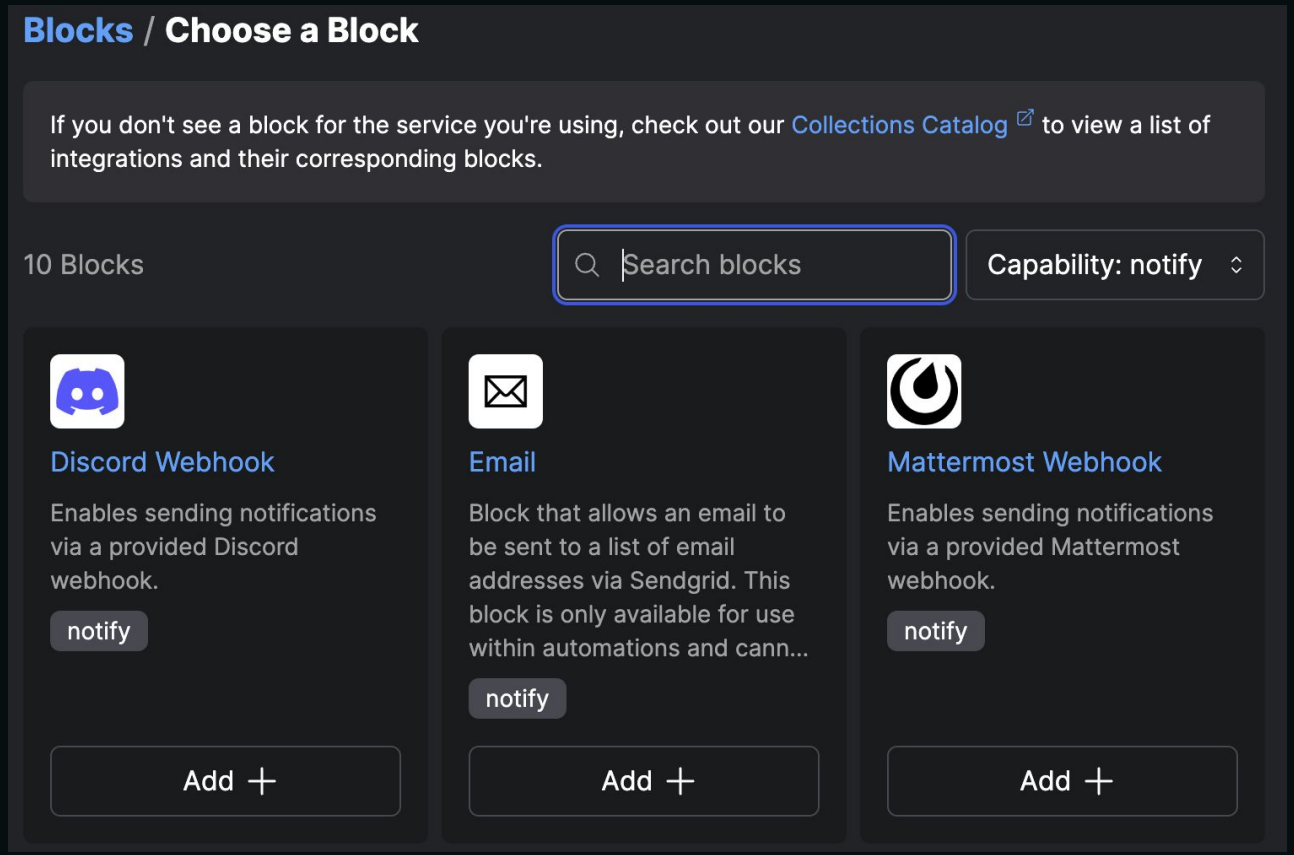
Automation trigger



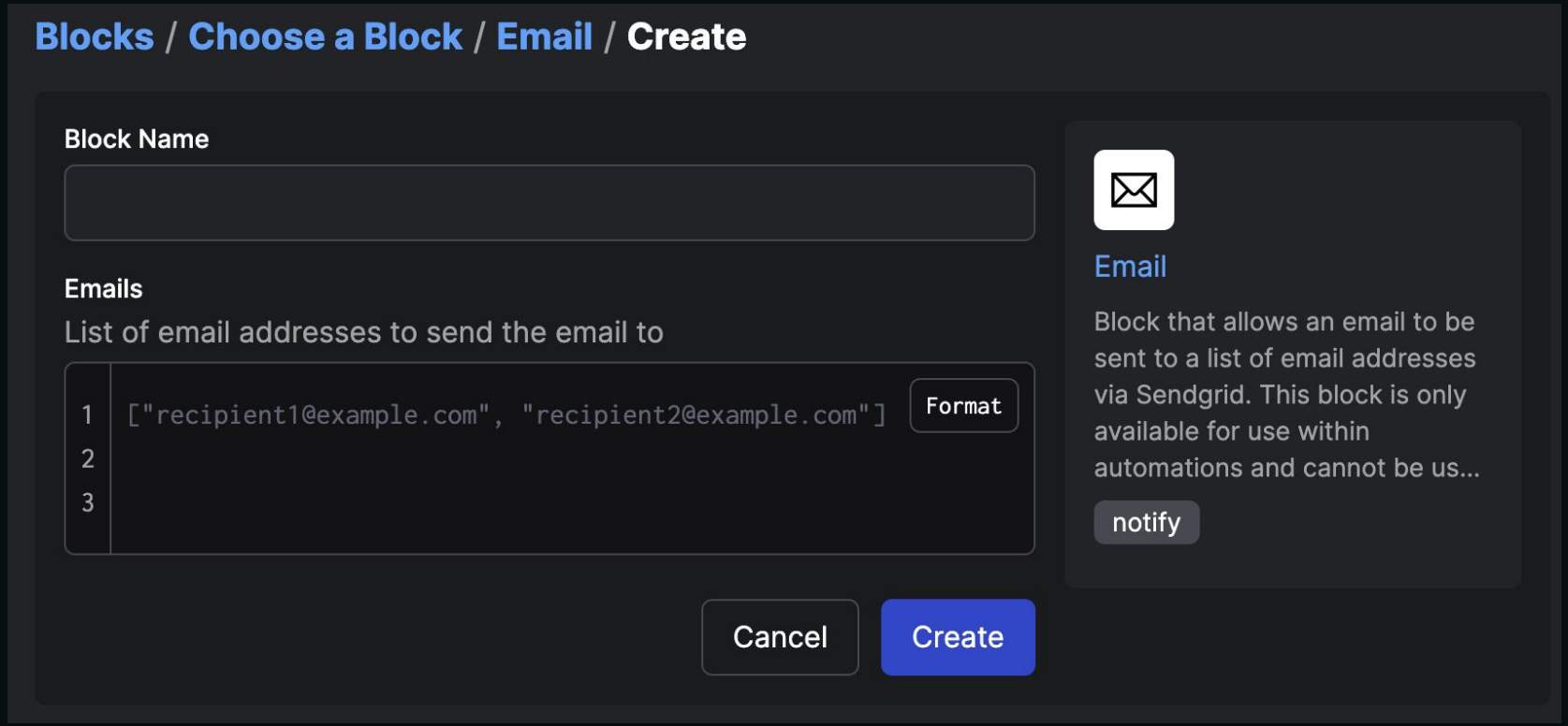
Automation action



Create a block with **notify** capability



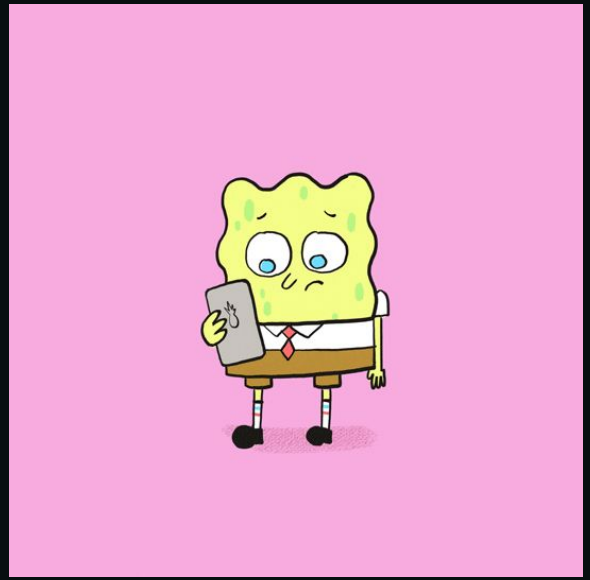
Create an **Email** block



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

- Working with data
- Prefect Cloud
- Events
- Automations

Lab 103

103 Lab

- In the UI, make an email notification automation that fires when flow runs complete
  - ❗ use **Email** block type
- See event feed in the UI
- Create a Markdown artifact that prints a weather forecast in a nicely formatted table
- Stretch 1: Create a flow that contains a task that uses caching
- Stretch 2: Change the cache policy, does the task run or not run as you expect?