

# 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?

---



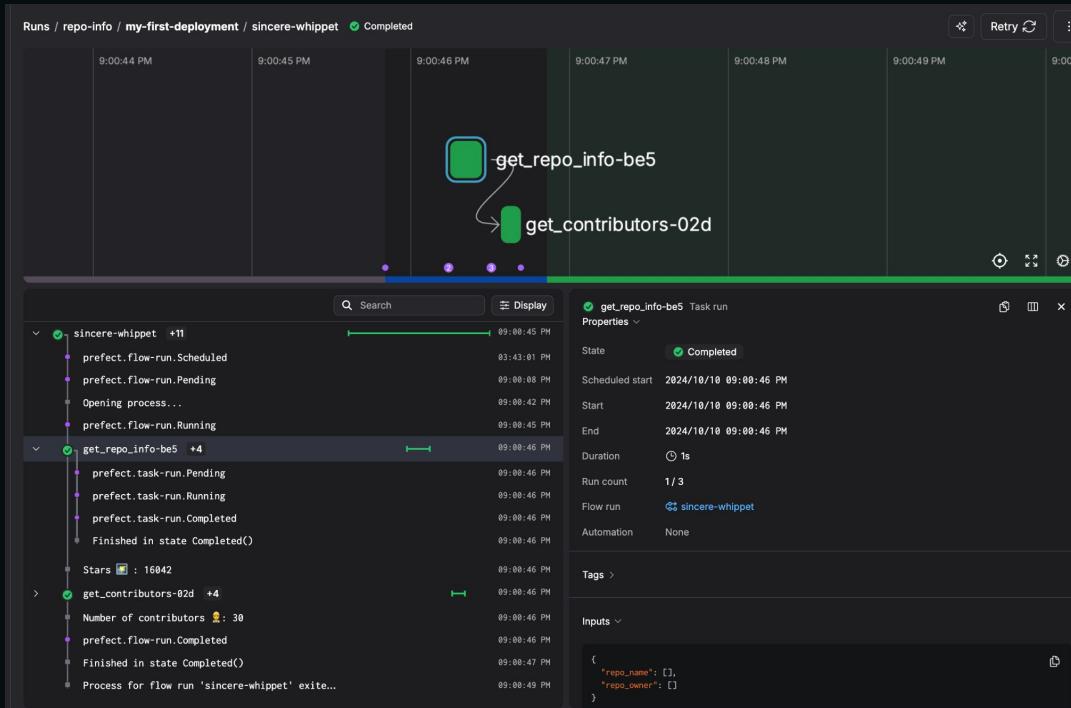
Answers the questions:

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



# What?

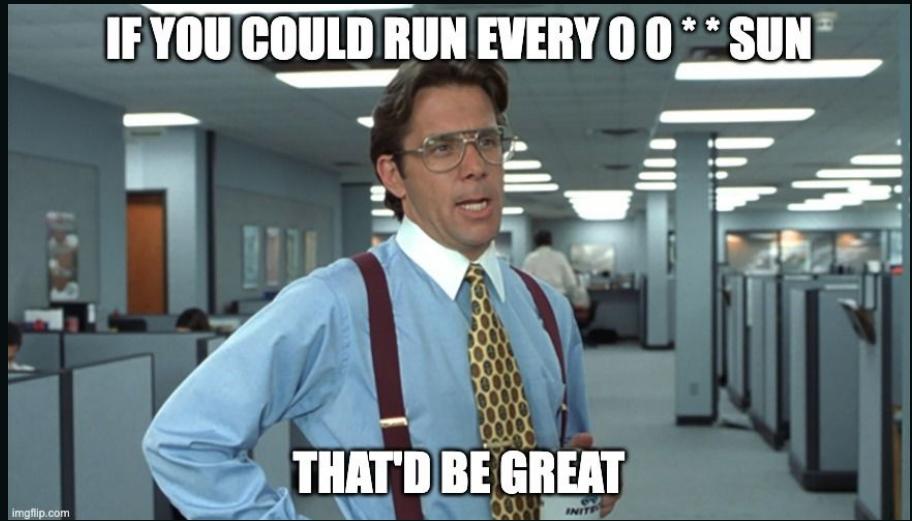
Code that moves and transforms data



# When?

---

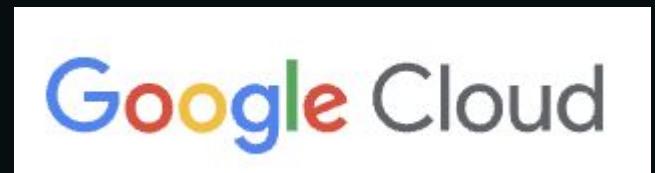
- Ad hoc (manually)
- On a schedule
- In response to events



# 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?

---

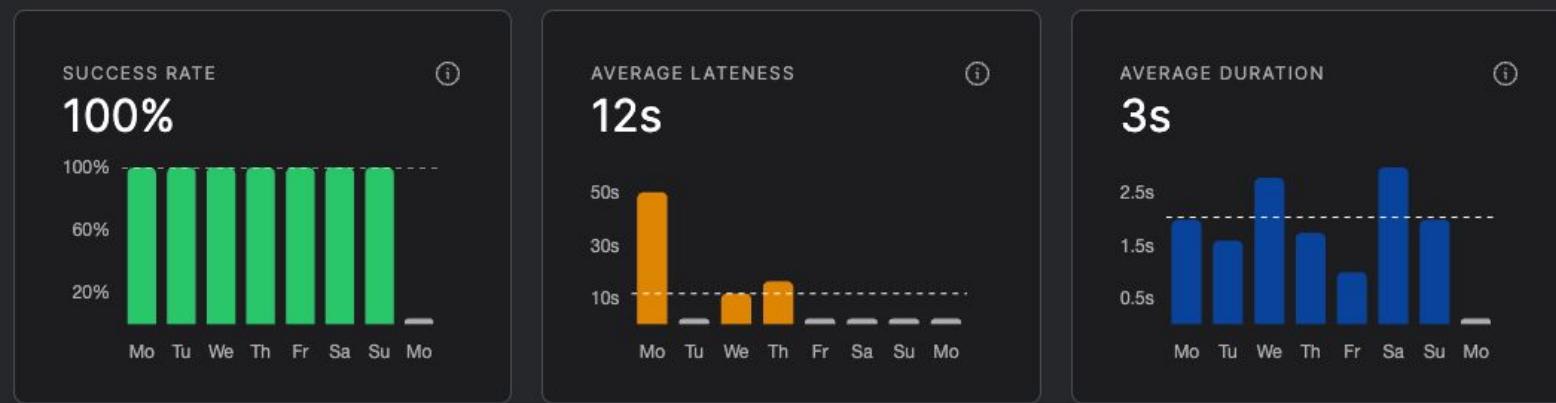


Prefect is an orchestration and observability platform that empowers developers to build resilient workflows you can trust. ❤️



## 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-seriemea

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

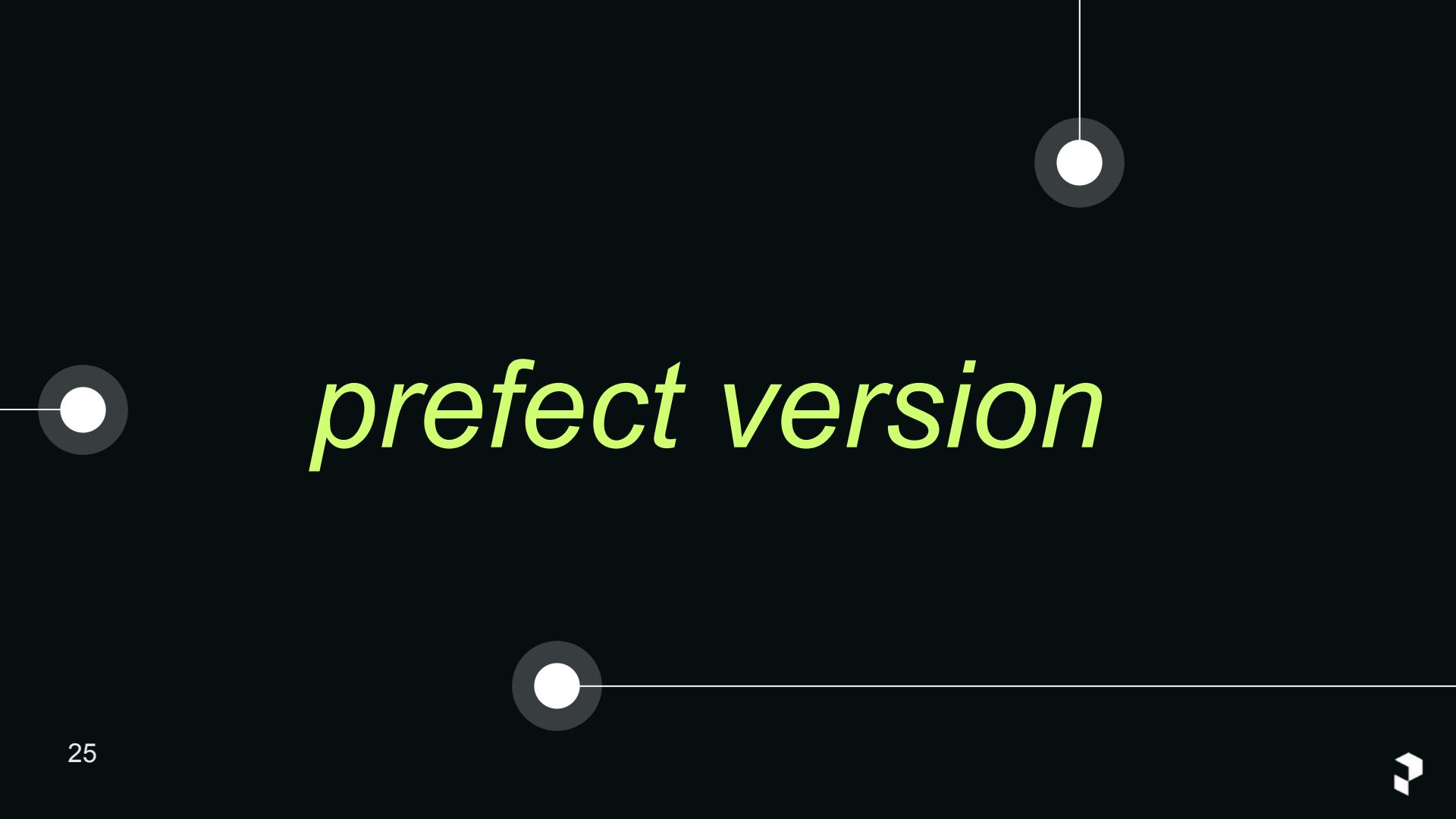


# 101 Agenda

---

- 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](https://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 





OR, if UI doesn't work:

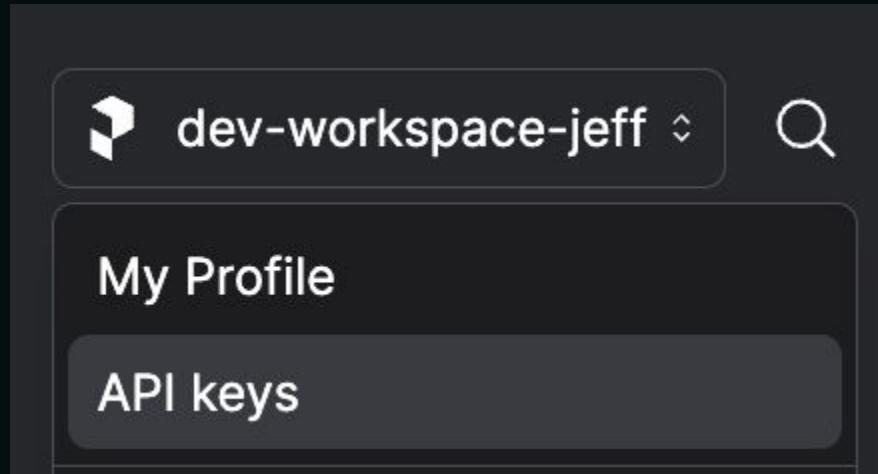
- Select *Paste an API key*
- Manually create an API key from Prefect Cloud in the UI



# Prefect Cloud - API key

---

(Top left in UI)



# Prefect Cloud - API key

The screenshot shows the Prefect Cloud interface for managing API keys. At the top left, there's a button labeled "API Keys" with a plus sign, which has a red arrow pointing to it from the bottom left. To the right of this button is a search bar with the placeholder "Search API keys". Below the header, it says "5 API keys". The main area displays a table with columns: "Name", "Created", and "Expiration". There are five rows of data, each with a three-dot menu icon on the far right. A modal window titled "Create API Key" is overlaid on the page. It contains fields for "Name" (with a placeholder "Name") and "Expiration Date" (set to "Oct 5th, 2023"). Below these fields is a checkbox labeled "Never Expire". At the bottom of the modal are two buttons: "Cancel" and "Create".

Name	Created	Expiration
demos	Jun 10th, 2023	2023/07/10 12:00:00 AM
cli-efaf65f4-2e55-4bd0-9b53-e46519d5d6a9	Jul 7th, 2023	2023/08/06 12:00:00 AM
cli-275b520		2023/08/25 02:00:00 AM
cli-f264094		2023/09/20 12:00:00 AM
cli-dd6a555		2023/09/30 12:00:00 AM

34

# 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*

## Available Profiles:

```
* default
  local
jeffmshale
  gh2
prefect-more
```



# Prefect profiles

---

Profiles live in `~/.prefect/profiles.toml` 

```
active = "sandbox-jeff"

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

[profiles.qawork]
PREFECT_API_KEY = "pnu_GSLLSpUFz83ZgecfLsEeBy9TDdAwqu3xAQX"
PREFECT_API_URL = "https://api.stg.prefect.dev/api/accounts/8e8e0fcc-53a5-46f4-80b1-d8fdf4fae7"
PREFECT_LOGGING_LEVEL = "DEBUG"
PREFECT_DEFAULT_DOCKER_BUILD_NAMESPACE = "us-central1-docker.pkg.dev/prefect-sbx-community-eng"

[profiles.storage-demo]
PREFECT_API_KEY = "pnu_F16ZsLxoen5gj1QHGkDCatS7LiUB042xgfQ"
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](https://open-meteo.com)



# Start: basic Python function

```
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

```
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-44954b5c0ab6/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

The screenshot shows the Prefect UI Runs page for a flow named "fetch-weather". A specific run titled "qualified-lorikeet" is selected, indicated by a green checkmark icon. The status of this run is "Completed". The UI displays a tree view of the run's tasks:

- qualified-lorikeet** +3 (Completed at 08:13:56 AM)
- prefect.flow-run.Pending** (Completed at 08:13:56 AM)
- prefect.flow-run.Running** (Completed at 08:13:56 AM)
- prefect.flow-run.Completed** (Completed at 08:13:57 AM)

A message at the top of the page states: "This flow run did not generate any task or subflow runs". To the right of the tree view, there is a detailed "Properties" panel for the selected run:

Property	Value
State	Completed
Scheduled start	2024/10/16 08:13:56 AM
Start	2024/10/16 08:13:56 AM
End	2024/10/16 08:13:57 AM
Duration	1s
Run count	1 / 1
Creator	jeffprefectio
Flow	fetch-weather
Run	qualified-lorikeet
Deployment	None
Work pool	None
Work queue	None
Automation	None

Below the properties, there are sections for "Tags" and "Parameters". The "Parameters" section contains the following JSON object:

```
{  
  "lat": 38.9,  
  "lon": -77  
}
```

# 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!

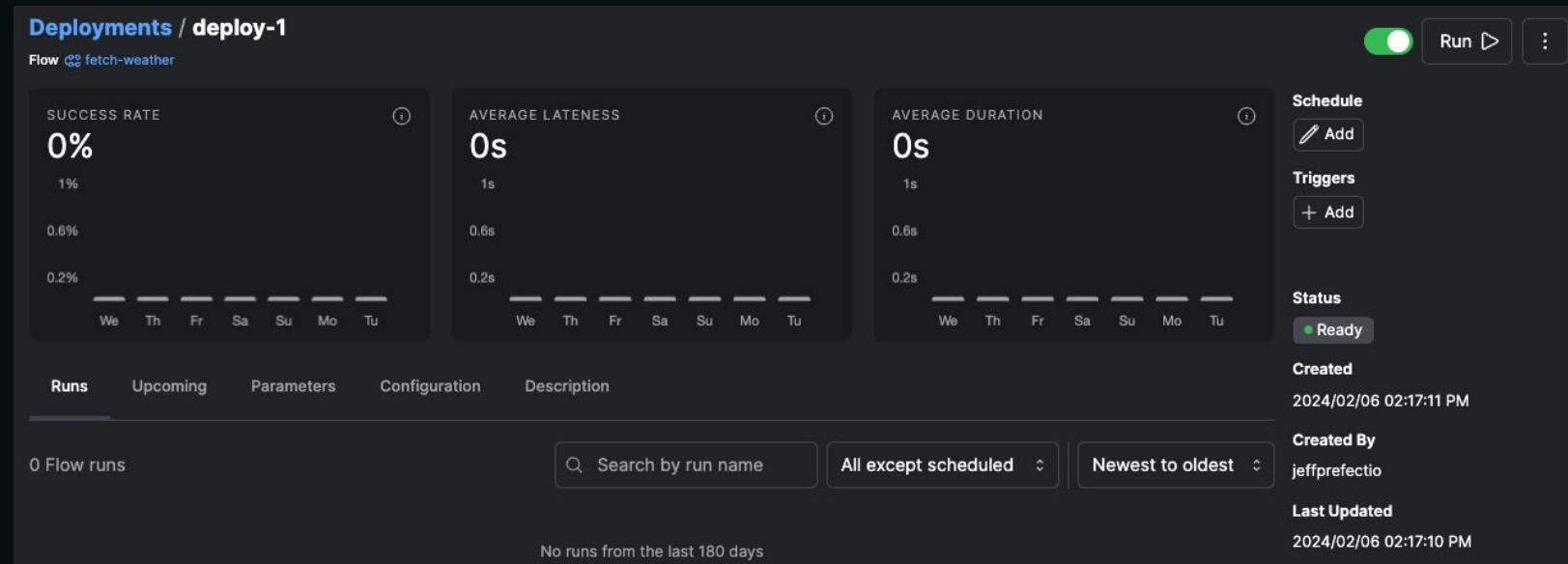
---

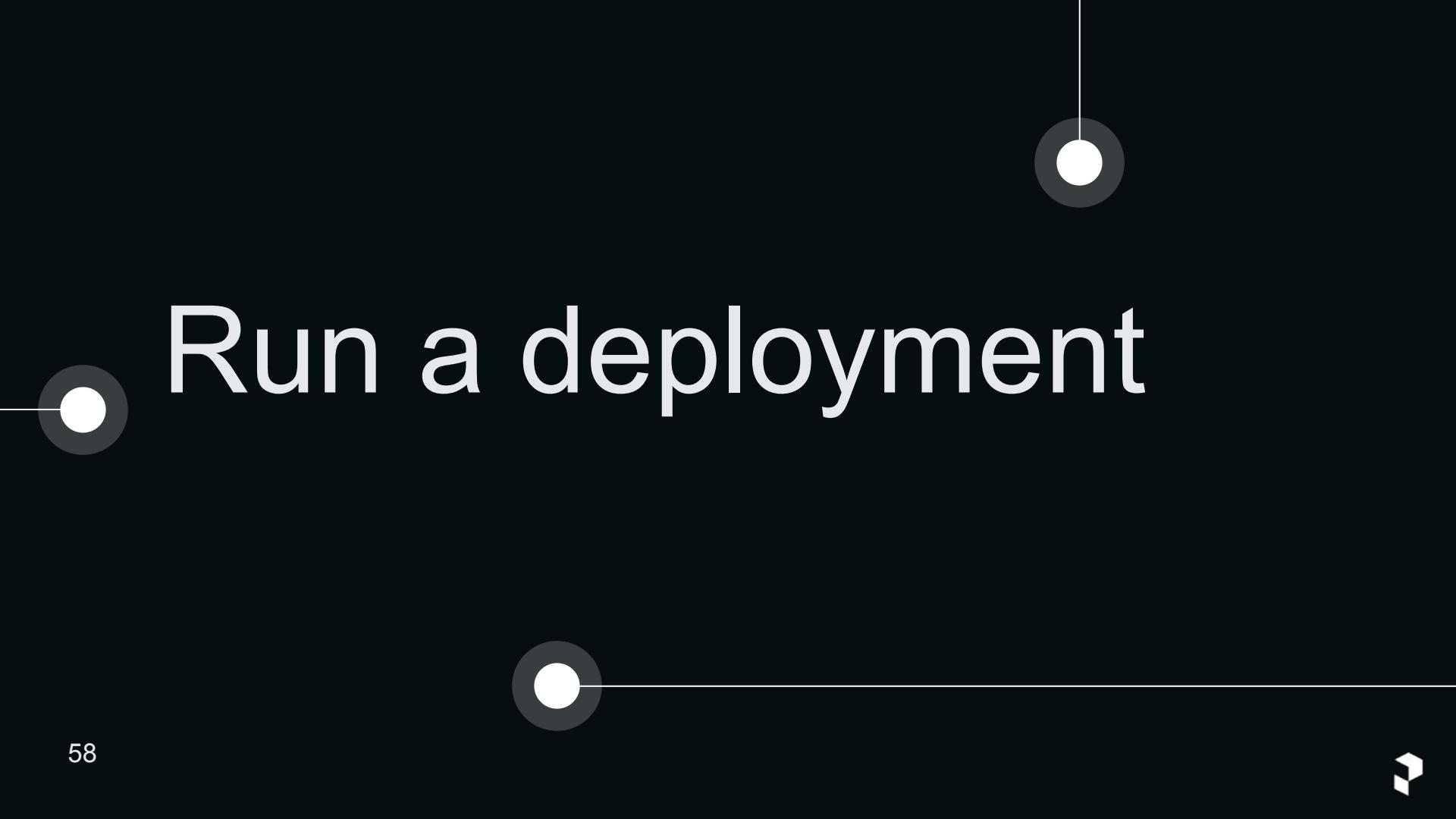


# See the deployment in the UI



## Deployment page





# Run a deployment

# Run manually from UI: Run -> Quick run

Deployments / deploy-1

Flow `fetch-weather`

SUCCESS RATE  
100%

AVERAGE LATENESS  
0s

AVERAGE DURATION  
1s

Runs Upcoming Parameters Configuration Description

2 Flow runs

Search by run name

All except scheduled

Newest to oldest

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

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

Run Run ⋮

Schedules + Schedule

Triggers + Add

Status Ready

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

Created By jeffprefectio

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

Updated By jeffprefectio

Endpoint weather1-serve.py:fetch\_weather

Path .

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

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

Deployment Version

Quick run

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

The screenshot shows the Prefect UI interface for viewing flow run logs. At the top, it displays the path: Runs / fetch-weather / deploy-1 / inescapable-griffin. A green circular icon indicates the status is Completed. Below this is a timeline from 8:19:41 AM to 8:19:56 AM. A message states: "This flow run did not generate any task or subflow runs".

The main area contains a tree view of the flow run events:

- inescapable-griffin +8
  - prefect.flow-run.Scheduled
  - Runner 'deploy-1' submitting flow run 'df6e6327-6c53-463a-b49f-db748a9325ae'
  - prefect.flow-run.Pending
  - Opening process...
  - Completed submission of flow run 'df6e6327-6c53-463a-b49f-db748a9325ae'
  - prefect.flow-run.Running
  - prefect.flow-run.Completed
  - Process for flow run 'inescapable-griffin' exited cleanly.

To the right of the tree view is a detailed Properties panel for the flow run:

Property	Value
State	Scheduled
Scheduled start	2024/10/16 08:19:41 AM
Start	None
Duration	0s
Run count	0 / 1
Creator	jeffprefectio
Flow	fetch-weather
Run	inescapable-griffin
Deployment	deploy-1
Work pool	None
Work queue	None
Automation	None

Below the Properties panel are sections for Tags, Parameters, Configuration, and Logs.

The Logs section shows the following output:

```
Runner 'deploy-1' submitting flow run 'df6e6327-6c53-463a-b49f-db748a9325ae'  
Opening process...  
Completed submission of flow run 'df6e6327-6c53-463a-b49f-db748a9325ae'  
Process for flow run 'inescapable-griffin' exited cleanly.
```

At the bottom right of the UI is a small Databricks logo.

## Run deployment manually from CLI

---

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



`.serve()`

---



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



# Scheduling



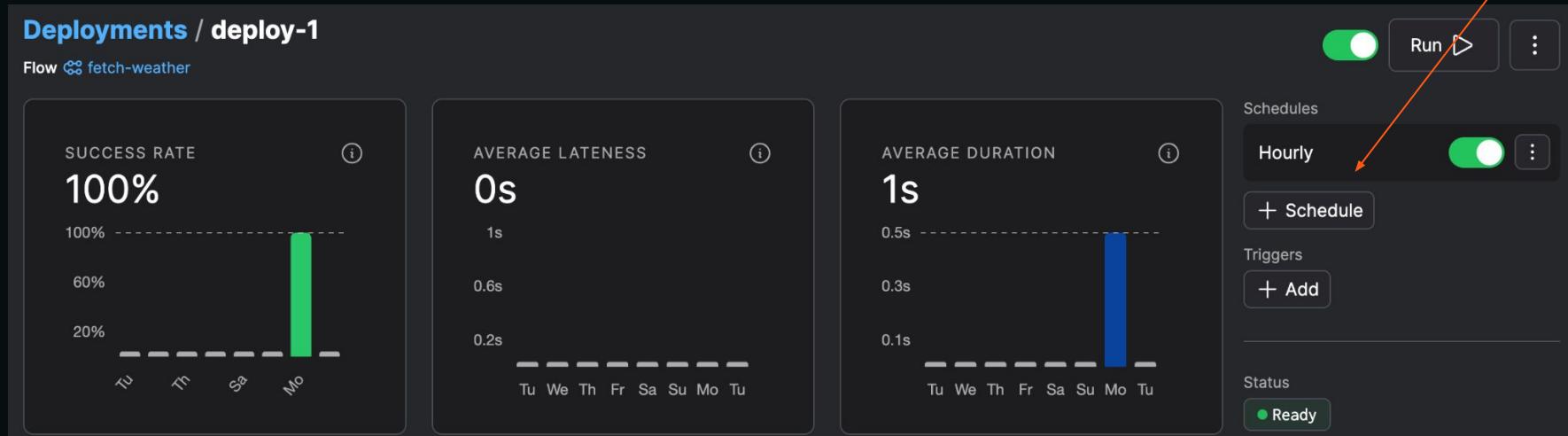
# Create a deployment schedule

---

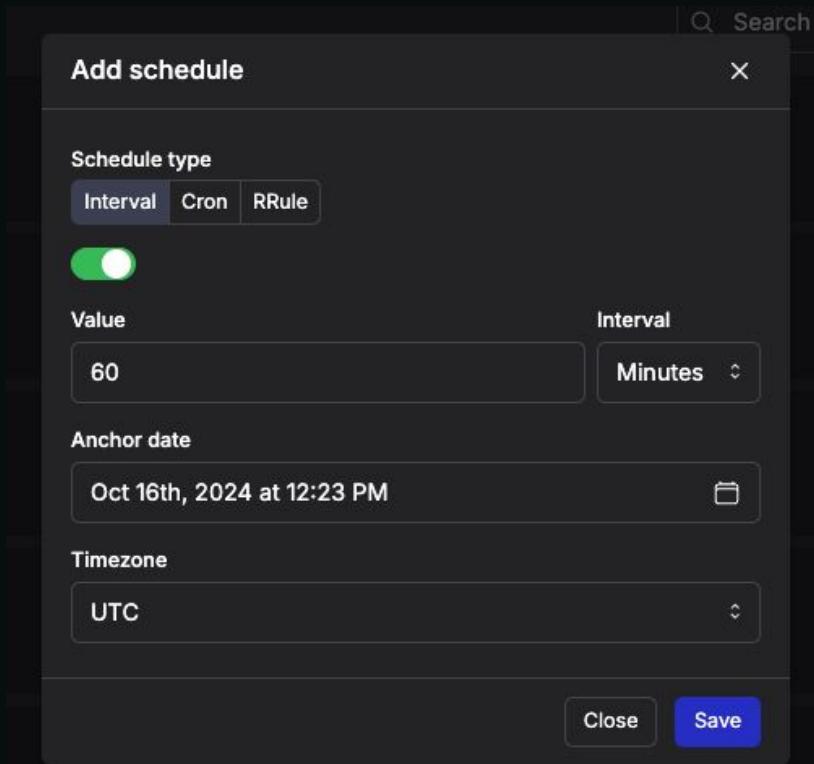
1. When create deployment
2. After deployment creation in UI or CLI



# Create, pause, and delete schedules from UI



# Create schedule from UI



# Add schedule when create deployment with `.serve()`

```
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*

```
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"
```



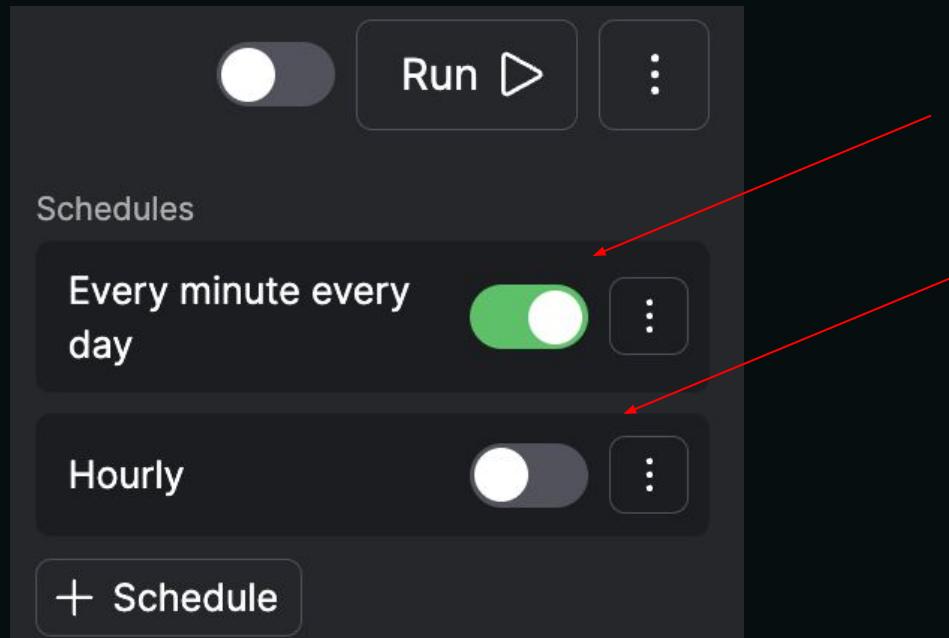
# Pausing and resuming deployment schedules

---



# 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()`

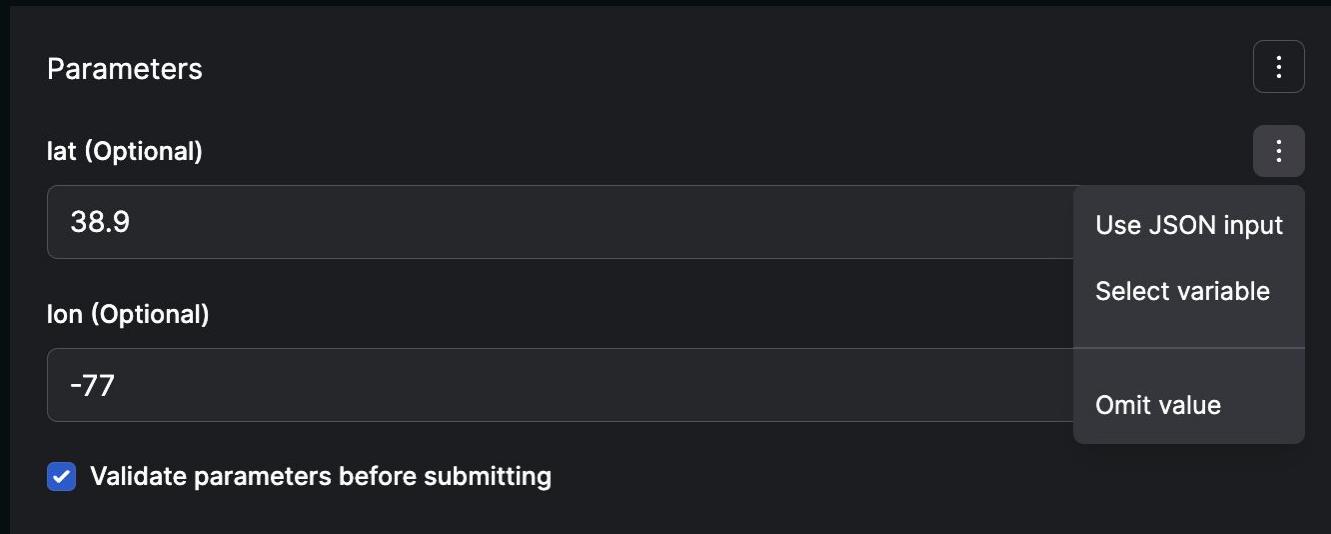
```
if __name__ == "__main__":
    fetch_weather.serve(name="deploy-params", parameters={"lat": 11, "lon": 12})
```



# 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](https://docs.prefect.io)



# Prefect codebase

---

[github.com/PrefectHQ/prefect](https://github.com/PrefectHQ/prefect)

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

 **prefect** Public

 Edit Pins ▾

 Unwatch 162 ▾

 Fork 1.5k ▾

 Starred 15.5k ▾



# 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](https://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](https://open-meteo.com/en/docs)

Example: wind speed for the last hour:

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



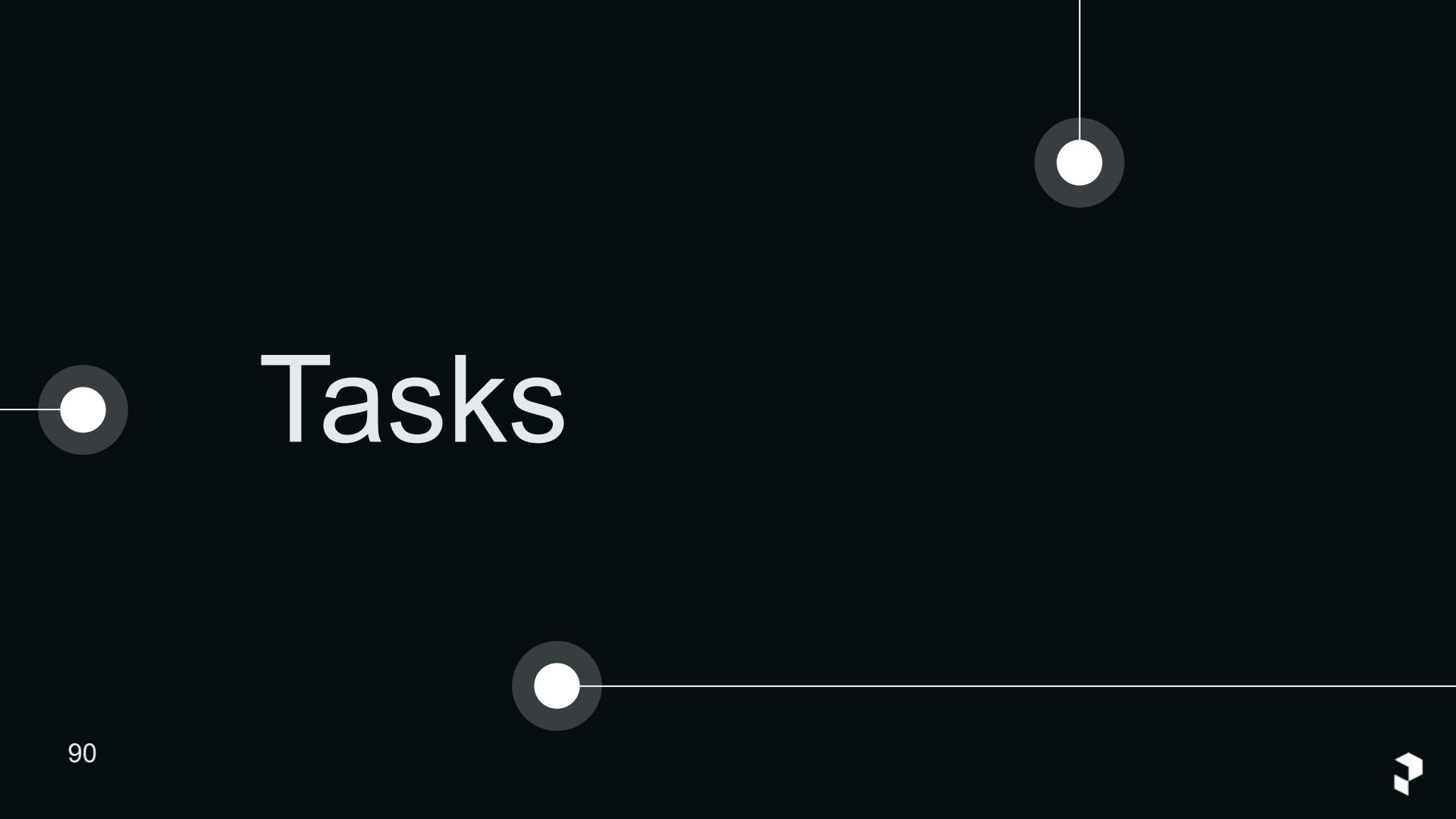
# 102 - Orchestration and observation: Understand workflow state and guard against failure

# 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

---

```
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

---

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

# Pipeline (assembly) function

---

```
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

---

```
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

---

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

# Pipeline flow function

---

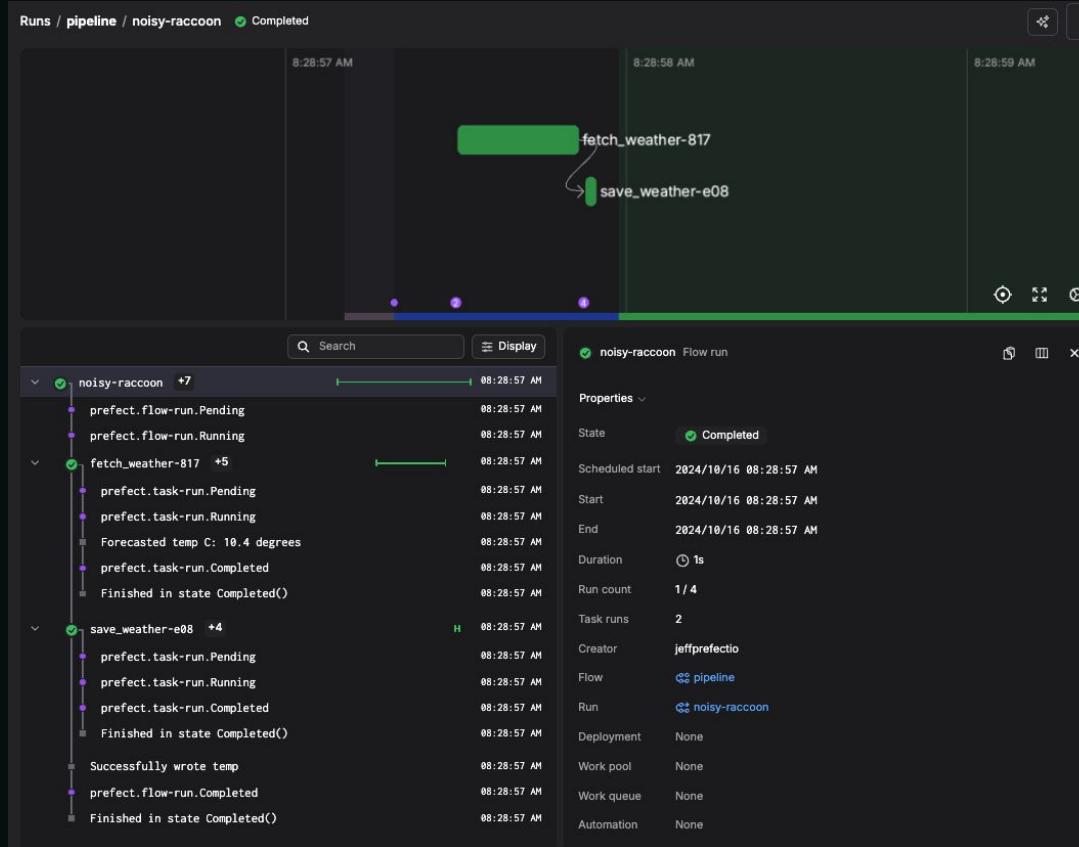
```
@flow
def pipeline(lat: float = 38.9, lon: float = -77.0):
    temp = fetch_weather(lat, lon)
    result = save_weather(temp)
    return result
```

# 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-44954b5c0ab6/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

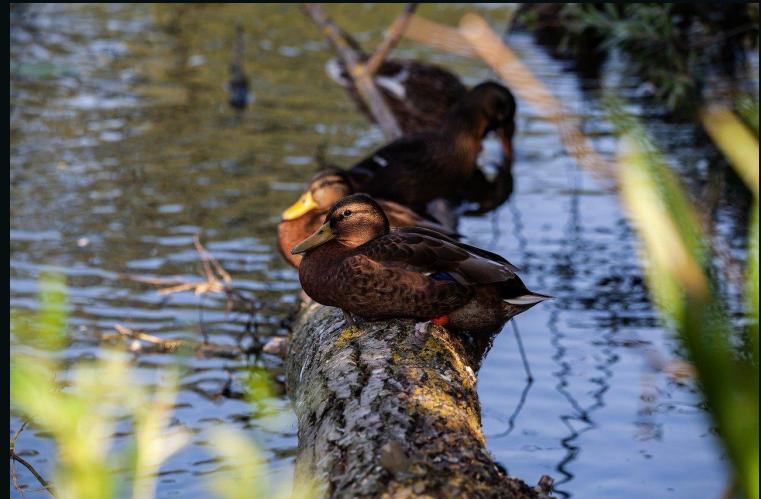
---

-  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*

```
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/9b649228-0419-40d137367a-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()
```



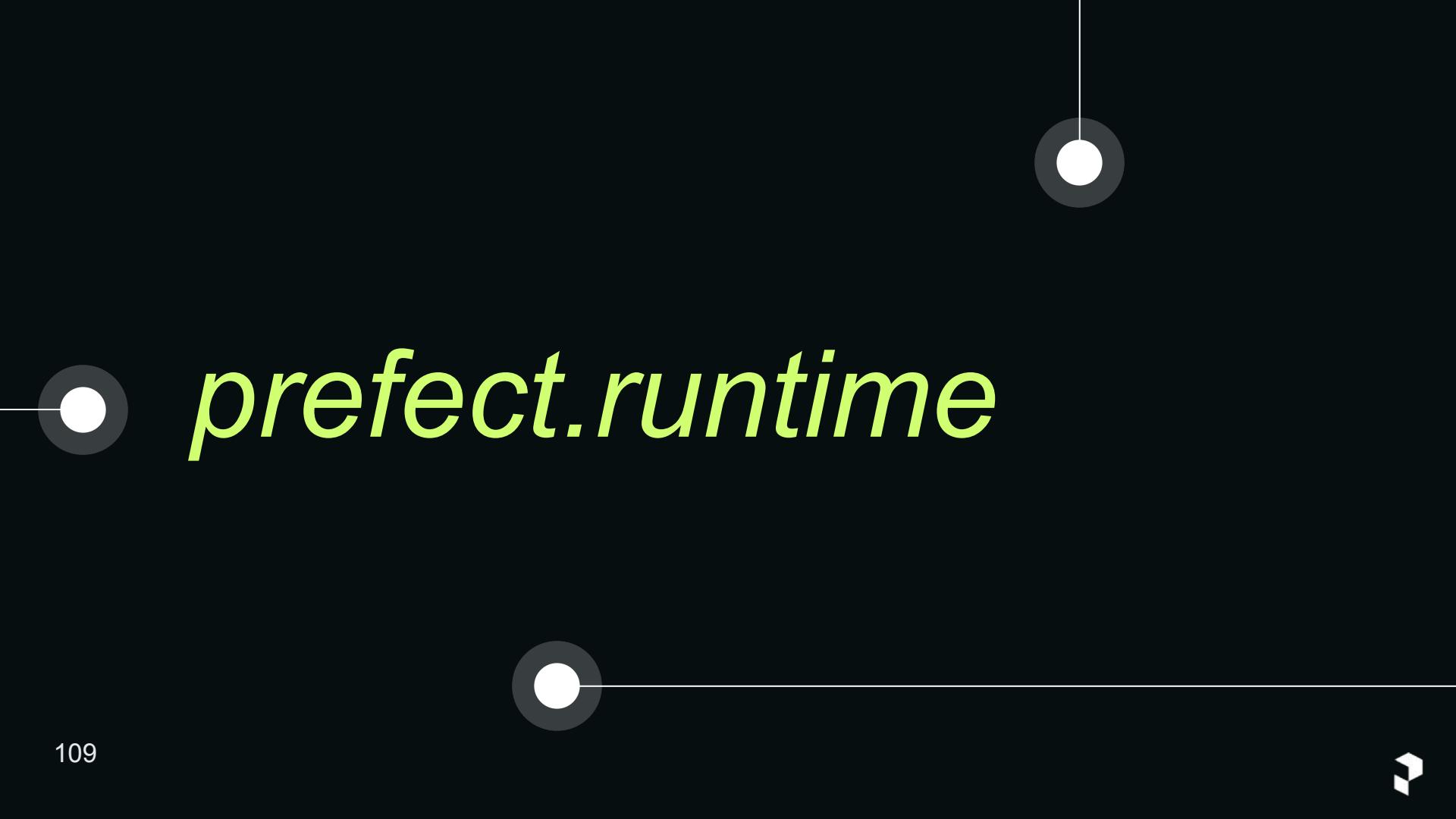
# Logging

---

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*

---

```
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)
```

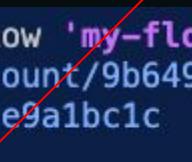


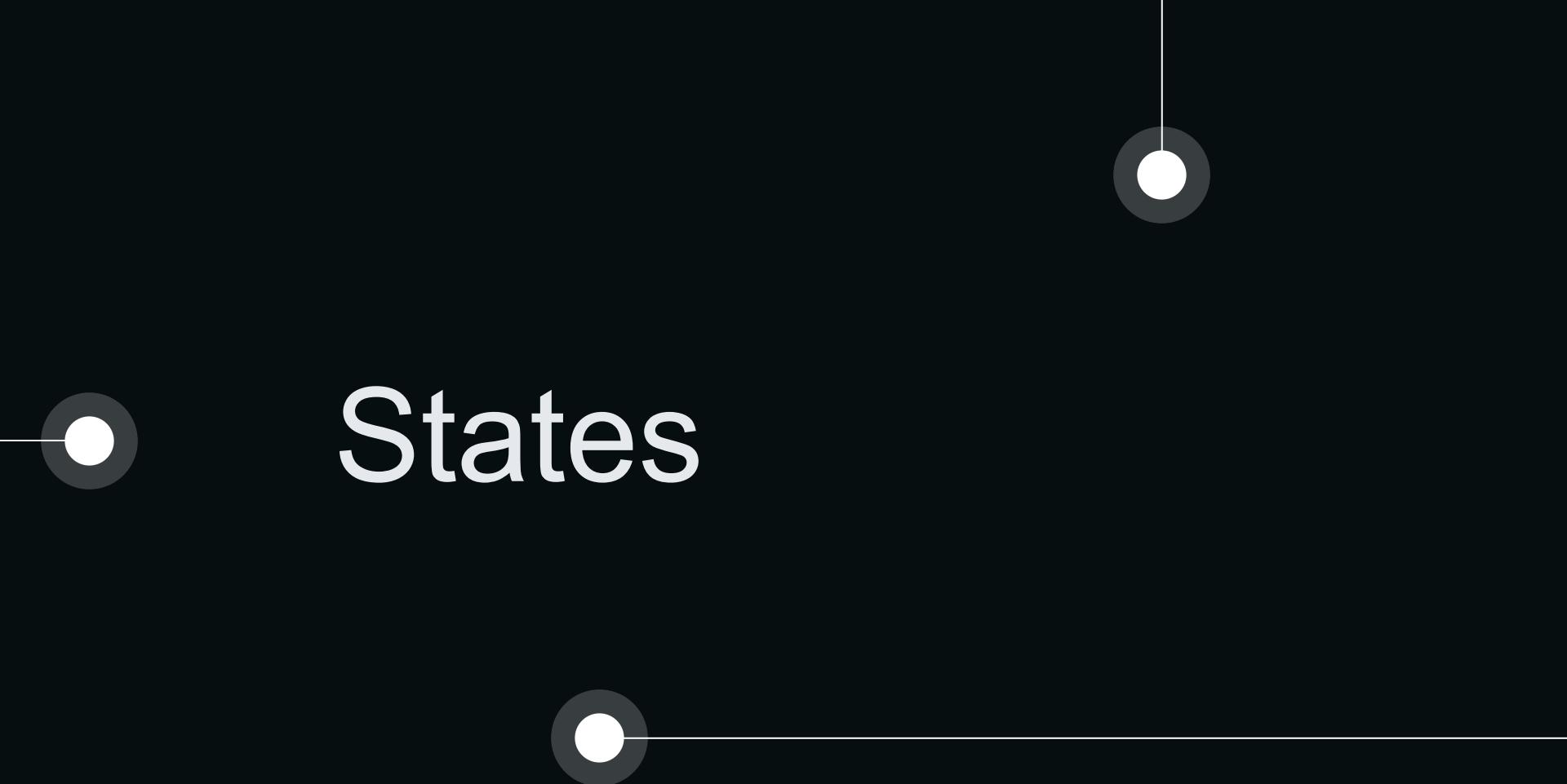
## *prefect.runtime*

---

Useful for labeling, logs, etc.

```
08:35:53.427 | INFO    | prefect.engine - Created flow run 'ochre-cobra' for flow 'my-flow'
08:35:53.430 | INFO    | prefect.engine - View at https://app.prefect.cloud/account/9b64922d137367a-5055-44ff-b91c-6f7366c9e4c4/runs/flow-run/ed6b0845-7240-4a1d-9402-1ebce9a1bc1c
08:35:53.631 | INFO    | Flow run 'ochre-cobra' - My name is ochre-cobra
08:35:53.689 | INFO    | Flow run 'ochre-cobra' - I belong to deployment None
08:35:53.717 | INFO    | Task run 'my_task-84c' - My name is my_task-84c
08:35:53.719 | INFO    | Task run 'my_task-84c' - Flow run parameters: {'x': 1}
08:35:53.722 | INFO    | Task run 'my_task-84c' - Finished in state Completed()
08:35:53.865 | INFO    | Flow run 'ochre-cobra' - Finished in state Completed()
```





# 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

---

<code>Cancelled</code>	<code>CANCELLED</code>	Yes	The run did not complete because a user determined that it should not.
<code>Completed</code>	<code>COMPLETED</code>	Yes	The run completed successfully.
<code>Cached</code>	<code>COMPLETED</code>	Yes	The run result was loaded from a previously cached value.
<code>RolledBack</code>	<code>COMPLETED</code>	Yes	The run completed successfully but the transaction rolled back and executed rollback hooks.
<code>Failed</code>	<code>FAILED</code>	Yes	The run did not complete because of a code issue and had no remaining retry attempts.
<code>Crashed</code>	<code>CRASHED</code>	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

---

```
import httpx
from prefect import flow

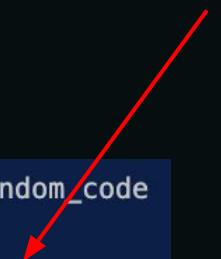
@flow(retries=4) 
def fetch_random_code():
    random_code = httpx.get("https://httpstat.us/Random/200,500", verify=False)
    if random_code.status_code >= 400:
        raise Exception()
    print(random_code.text)

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

# Automatic retry

---

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



# Automatic retry with delay

---



## Automatic retry with delay

---

Specify in task or flow decorator

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



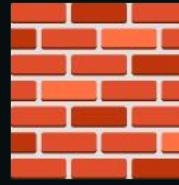
# Task retries with delay

---

```
@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

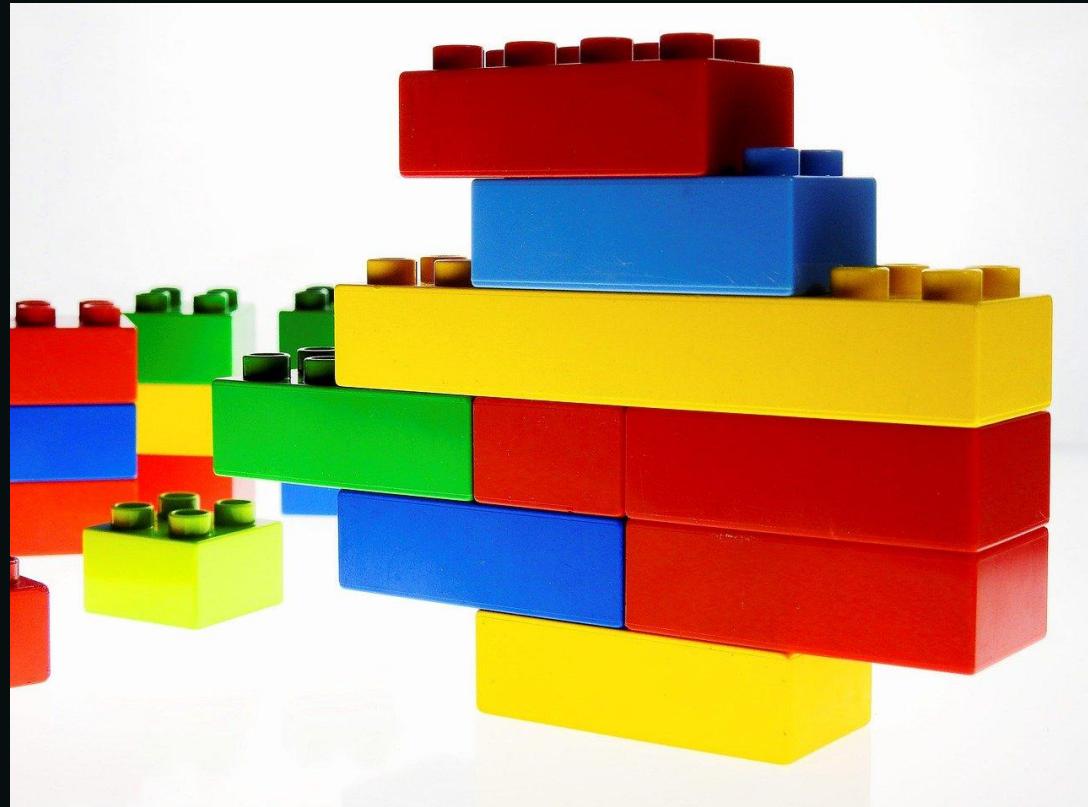
---

Configuration

+

Code

Useful for storing configuration for connecting to external systems



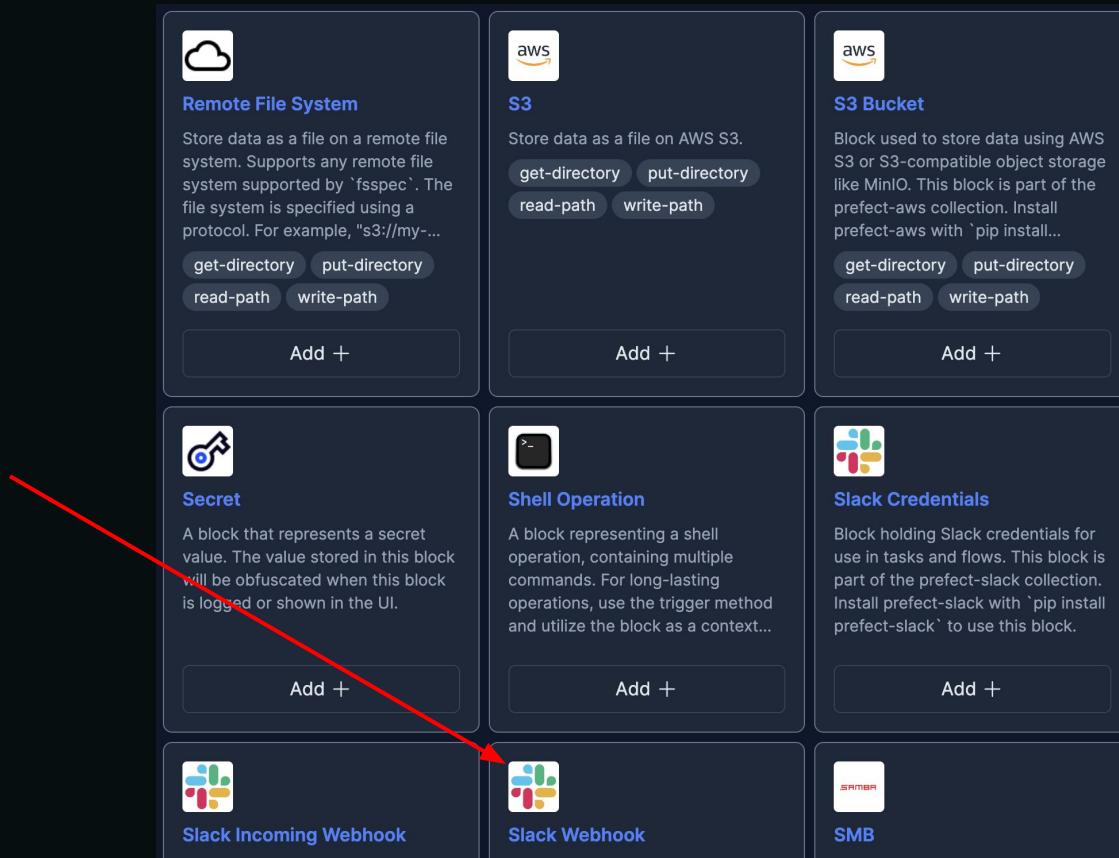
# Create a block from the UI

---

The screenshot shows a dark-themed user interface for a workspace named 'dev-workspace-jeff'. On the left, a sidebar lists various workspace sections: Home, Runs, Flows, Deployments, Work Pools, Automations, Event Feed, Incidents, Configuration (with 'Blocks' selected), Variables, Webhooks, and Concurrency. The 'Blocks' section is highlighted with a blue border. The main content area is titled 'Blocks' and features a large, light-gray cube icon. Below the icon, the text 'Add a block to get started' is displayed in bold. A descriptive subtitle explains: 'Blocks securely store credentials and configuration to easily manage connections to external systems.' At the bottom right of the main area are two buttons: 'Add Block +' and 'View Docs' with a link icon.



# Create a block from the UI - choose a block type



# Create a block from the UI

---

Blocks / Choose a Block / Slack Webhook / Create

**Block Name**

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

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

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

`notify`

**Cancel** **Create**



# Block types in UI - filter by capability

**Blocks / Choose a Block**

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

9 Blocks

Search blocks

Capability: notify

 Discord Webhook Enables sending notifications via a provided Discord webhook. <a href="#">notify</a>  <a href="#">Add +</a>	 Email Block that allows an email to be sent to a list of email addresses via Sendgrid. This block is only available for use within automations and cannot be used within user flows. <a href="#">notify</a>  <a href="#">Add +</a>	 Mattermost Webhook Enables sending notifications via a provided Mattermost webhook. <a href="#">notify</a>  <a href="#">Add +</a>	 Microsoft Teams Webhook Enables sending notifications via a provided Microsoft Teams webhook. <a href="#">notify</a>  <a href="#">Add +</a>
 Opsgenie Webhook Enables sending notifications via a provided Opsgenie webhook. <a href="#">notify</a>  <a href="#">Add +</a>	 PagerDuty Webhook Enables sending notifications via a provided PagerDuty webhook. <a href="#">notify</a>  <a href="#">Add +</a>	 Sendgrid Email Enables sending notifications via Sendgrid email service. <a href="#">notify</a>  <a href="#">Add +</a>	 Slack Webhook Enables sending notifications via a provided Slack webhook. <a href="#">notify</a>  <a href="#">Add +</a>
 Twilio SMS Enables sending notifications via Twilio SMS. <a href="#">notify</a>  <a href="#">Add +</a>			



# Under the hood, block types are Python classes

---



# Block types are Python classes

---

```
✓  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"  # noqa
    )

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



# Block types are Python classes (with nice forms)

**Blocks / Choose a Block / S3 Bucket / Create**

**Block Name**

**Bucket Name**  
Name of your bucket.

**Credentials**

**MinIOCredentials** **AwsCredentials**

A block containing your credentials to AWS or MinIO.

**MinIOCredentials (Optional)**  
Block used to manage authentication with MinIO. Refer to the MinIO docs: <https://docs.min.io/docs/minio-server-configuration-guide.html> for more info about the possible credentials.

 **Add +**

**Bucket Folder (Optional)**  
A default path to a folder within the S3 bucket to use for reading and writing objects.



## Create a block in Python - an instance of the class

---

```
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

---

```
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

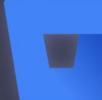


# Integrations

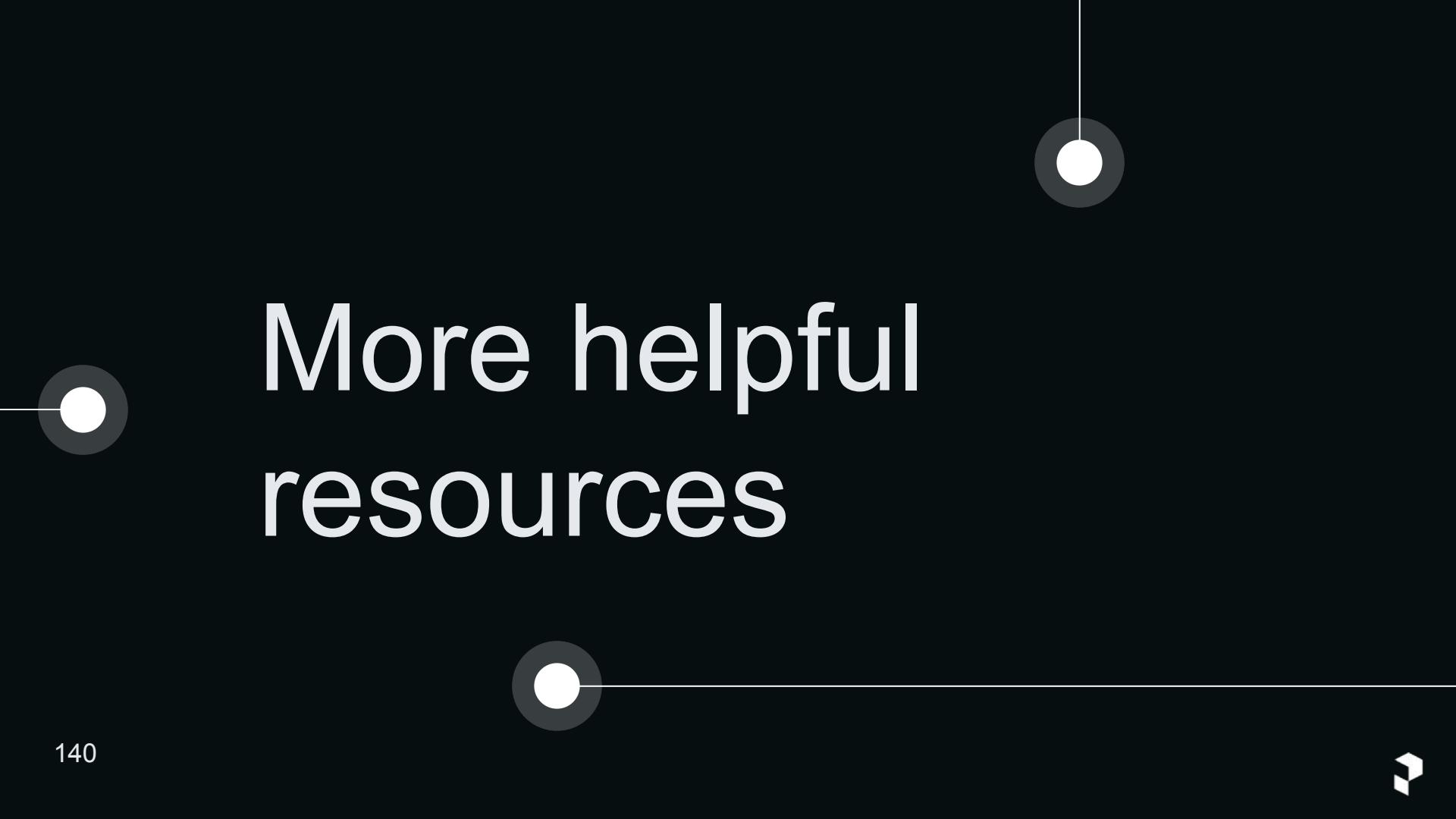
---



[docs.prefect.io/integrations](https://docs.prefect.io/integrations)

<p>Alert</p>  <p>Maintained by Khuyen Tran</p>	<p>AWS</p>  <p>Maintained by Prefect</p>	<p>Azure</p>  <p>Maintained by Prefect</p>	<p>Bitbucket</p>  <p>Maintained by Prefect</p>	<p>Coiled</p>  <p>Maintained by Coiled</p>
<p>CubeJS</p>  <p>Maintained by Alessandro Lollo</p>	<p>Dask</p>  <p>Maintained by Prefect</p>	<p>Databricks</p>  <p>Maintained by Prefect</p>	<p>dbt</p>  <p>Maintained by Prefect</p>	<p>Docker</p>  <p>Maintained by Prefect</p>





# 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*

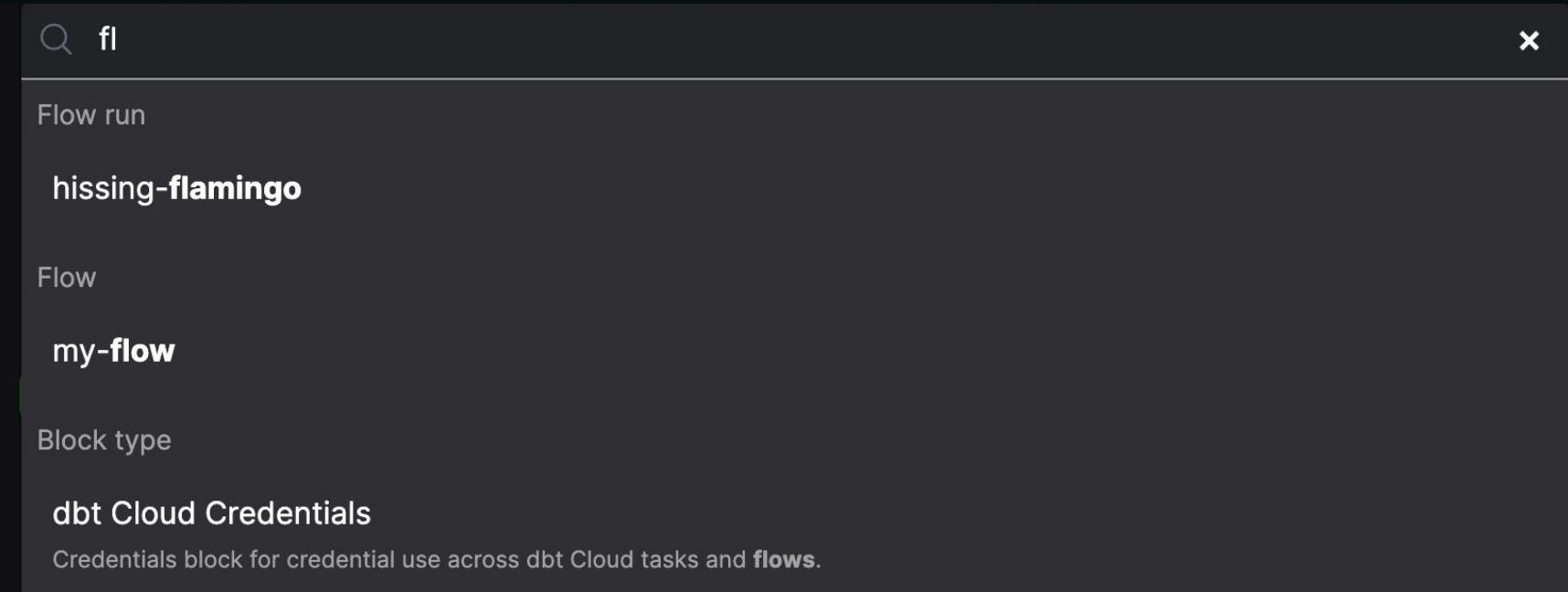
Commands	
<code>artifact</code>	Inspect and delete artifacts.
<code>automation</code>	Manage automations.
<code>block</code>	Manage blocks.
<code>cloud</code>	Authenticate and interact with Prefect Cloud
<code>concurrency-limit</code>	Manage task-level concurrency limits.
<code>config</code>	View and set Prefect profiles.
<code>dashboard</code>	Commands for interacting with the Prefect UI.
<code>deploy</code>	Create a deployment to deploy a flow from this project.
<code>deployment</code>	Manage deployments.
<code>dev</code>	Internal Prefect development.
<code>events</code>	Stream events.
<code>flow</code>	View and serve flows.
<code>flow-run</code>	Interact with flow runs.
<code>global-concurrency-limit</code>	Manage global concurrency limits.
<code>init</code>	Initialize a new deployment configuration recipe.
<code>profile</code>	Select and manage Prefect profiles.
<code>server</code>	Start a Prefect server instance and interact with the database
<code>shell</code>	Serve and watch shell commands as Prefect flows.
<code>task</code>	Work with task scheduling.
<code>task-run</code>	View and inspect task runs.
<code>variable</code>	Manage variables.
<code>version</code>	Get the current Prefect version and integration information.
<code>work-pool</code>	Manage work pools.
<code>work-queue</code>	Manage work queues.
<code>worker</code>	Start and interact with workers.

# Search in the UI

---



cmd + k or 



A screenshot of a search interface. At the top left is a search bar with the placeholder "Search" and a magnifying glass icon. To the right of the search bar are two small icons: a gear and a close button (an "x"). Below the search bar, the text "fl" is typed into the search field. A list of search results is displayed below the search bar:

- Flow run
- hissing-flamingo**
- Flow
- my-flow**
- Block type
- dbt Cloud Credentials**

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



## 102 Recap

---



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

- Tasks
- 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

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

1 is the result



## Passing results

---

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

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

# Results

---

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



# Persist results with *persist\_result=True*

---

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

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

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

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



# Persisted results

---

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

```
✓ .PREFECT
  ✓ storage
    {} c65d28dcc374424ba7212a39dd19418b
      storage > {} c65d28dcc374424ba7212a39dd19418b > ...
        1   {"serializer": {"type": "pickle", "picklelib": "cloudpickle", "picklelib_version": "2.2.1"},}
        2   "data": "gAWVxwIAAAAAACMEXBhbRhcy5jb3JlLmZyYW1llIwJRGF0YUZyYW1llJOUKYGUfZQojARfbWdy\nlIwec"
        3   "prefect_version": "2.10.3"}
        4
        5
        6
      memo_store.toml
      prefect.db
      profiles.toml
```



# Results - remote data storage

---

Store results in cloud provider storage - use a block

```
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](https://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

```
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*



```
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](https://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

---

```
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

---

```
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*

Key  
weather-report

Description  
Very scientific weather report

## Weather Report

### Recent weather

Time	Temperature
Temp Forecast	25.9



# Artifacts are versioned

---

Artifacts / latest-damage-report-car-1

Latest damage report for car 1

05:55 PM  
Jun 12th, 2024

**f2e1bb10**  
Flow run `Process Damage Report for Car 1`  
Task run `submit_damage_report-0`

04:43 PM  
Jun 12th, 2024

**b77853ad**  
Flow run `Process Damage Report for Car 1`  
Task run `submit_damage_report-0`

03:20 PM  
May 23rd, 2024

**eeba6b03**  
Flow run `Process Damage Report for Car 1`  
Task run `submit_damage_report-0`

12:49 PM  
May 23rd, 2024

**048980b2**  
Flow run `Process Damage Report for Car 1`  
Task run `submit_damage_report-0`

 Created **latest-damage-report-car-1**



# 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

---

The screenshot shows the Prefect Cloud user interface. On the left, a sidebar menu is open under the heading "Account settings". The menu items include "My Profile", "API keys", "Prefect Sandbox", "Account settings" (which is highlighted), "Switch account", "Workspace settings", "Switch workspace", "Invite and manage members", and "Sign out". Above the sidebar, the user's name "sandbox-jeff" is displayed next to a profile icon and a search bar.

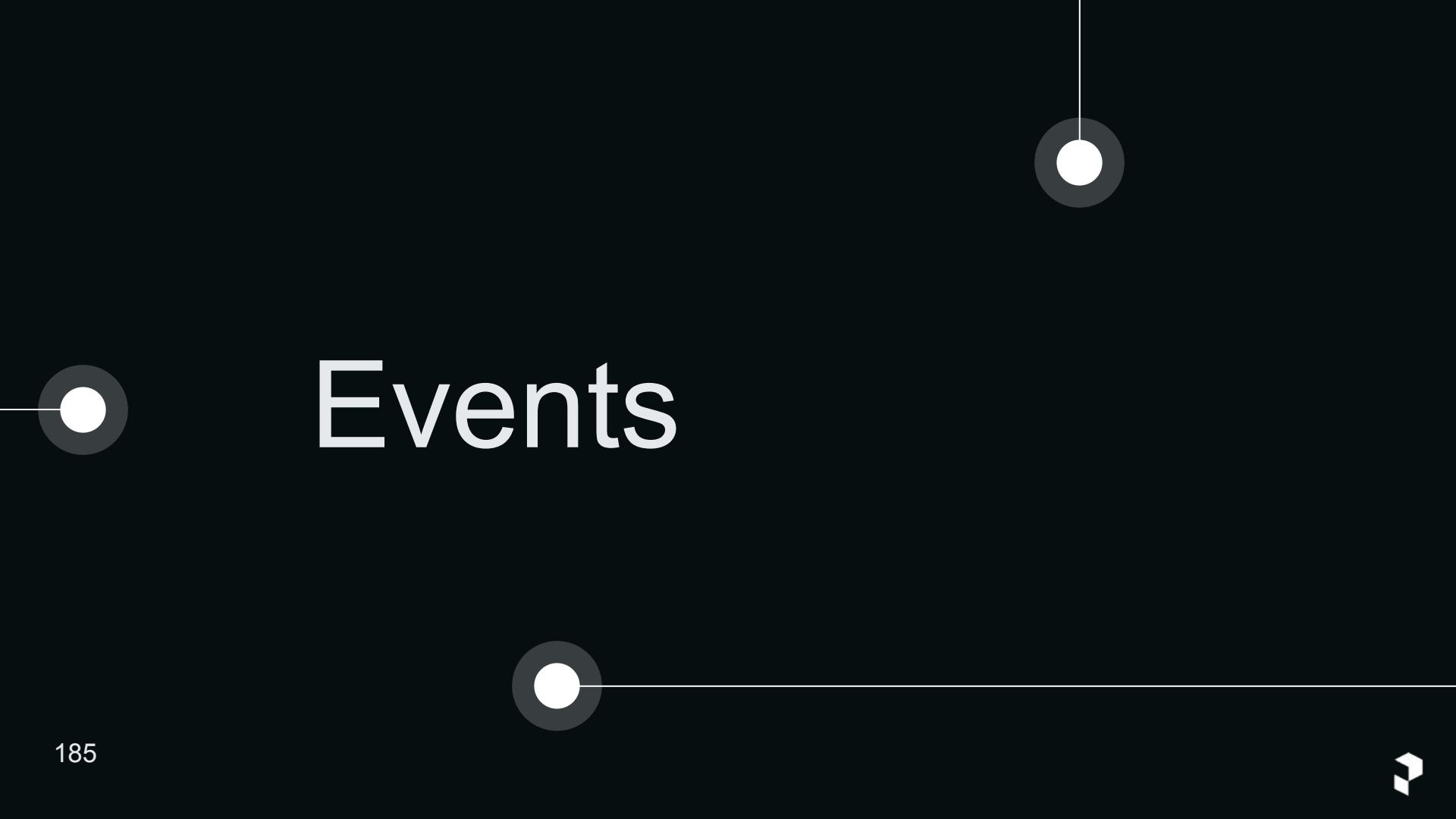
The main content area is titled "Settings". Under "Data Controls", there is a section for "Marvin AI". It contains the text: "Marvin AI-enabled features may utilize third party models. See our [data processing addendum](#) and [sub processors documentation](#) for more information." Below this text is a green toggle switch that is turned on.



# Error summaries by Marvin AI

- [get-info > zircon-tapir](#)  
☒ Failed ⌚ 2023/09/25 03:29:38 PM ⌚ 1s ∅ None  
⠇ Failed due to a `IndexError` in the `get_info` task; range object index out of range.
- [ml-flow > translucent-pogona](#)  
☒ Failed ⌚ 2023/09/25 03:16:42 PM ⌚ 1s ∅ 1 task run  
⠇ Failed due to a `ZeroDivisionError` in the `compute` task with message 'division by zero'.
- [ml-flow > wealthy-firefly](#)  
☑ Completed ⌚ 2023/09/25 03:16:25 PM ⌚ 2s ∅ 1 task run





# Events

# Events

---



- Record of what happened

Represent:

- API calls
- State transitions
- Changes in environment



# Event feed

Workspace Events

Resource: All resources

Events: All events

Timeline: 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

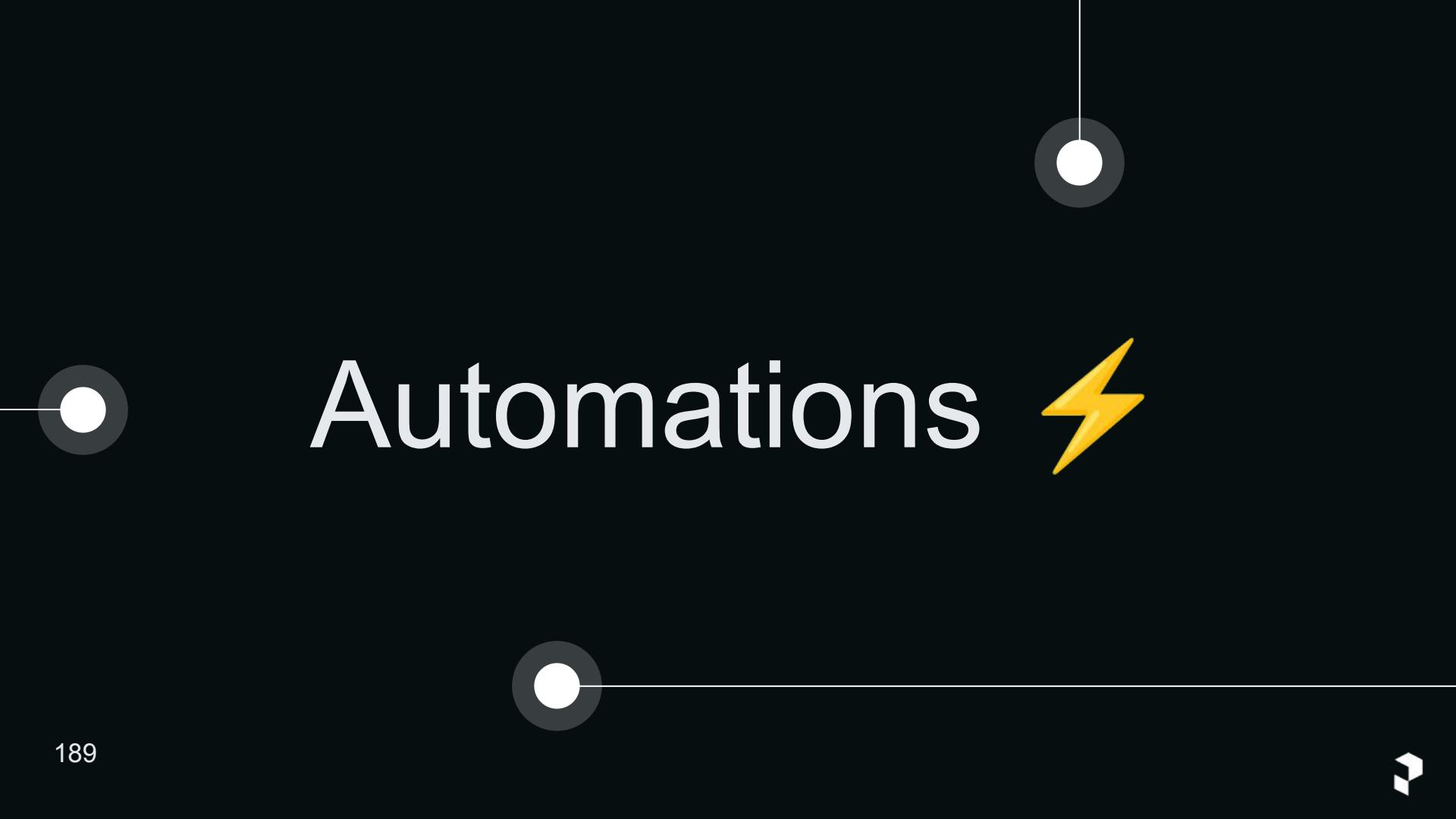
---



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 

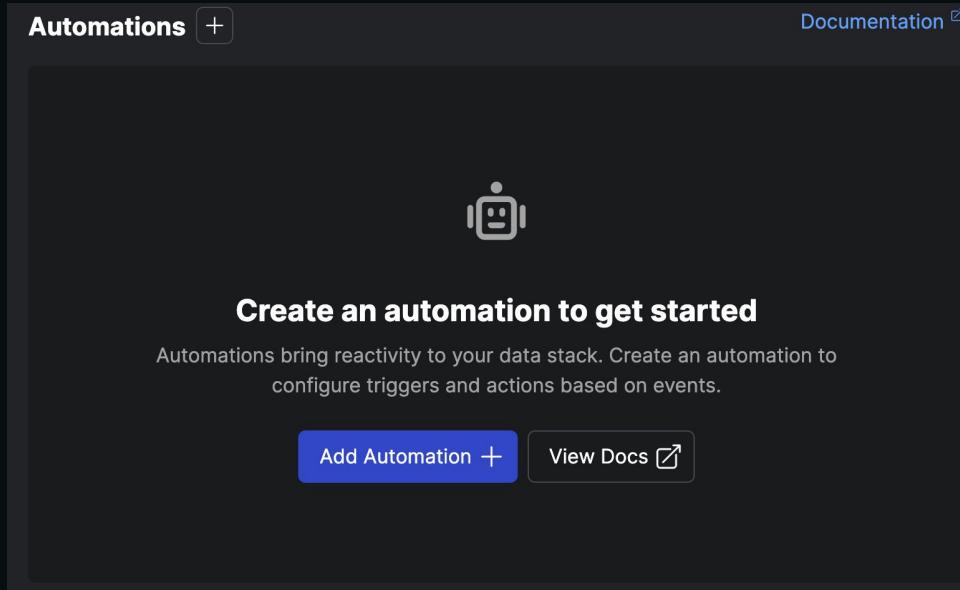


# Create an automation

---

**Trigger:** flow run failure

**Action:** notification - email



# Automation trigger

Automations / Create Documentation ↗

01 Trigger    02 Actions    03 Details

**Trigger Type**  
Flow run state

Form JSON

**Flows**  
All flows

**Flow Run Tags**  
All tags

**Flow Run**  
Enters All run states

Cancel Previous Next

**Related Events**

Apr 28th, 2024  
12:00 AM

May 4th, 2024  
11:59 PM



# Automation action

The screenshot shows the "Automations / Create" interface. At the top, there are three tabs: "Trigger" (with a checkmark icon), "Actions" (selected, indicated by a blue border and the number 02), and "Details" (with the number 03). Below the tabs, the "Action 1" section is visible. It includes an "Action Type" field containing "Send a notification", which is highlighted with a blue border. The "Block" section below it has an "Add +" button. The "Subject" section contains the text "Prefect flow run notification". In the "Body" section, there is a code block with the following content:

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



# Create a block with **notify** capability

**Blocks / Choose a Block**

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

10 Blocks  Capability: notify



**Discord Webhook**  
Enables sending notifications via a provided Discord webhook.  
**notify**



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



**Mattermost Webhook**  
Enables sending notifications via a provided Mattermost webhook.  
**notify**

**Add +** **Add +** **Add +**



# Create an Email block

---

Blocks / Choose a Block / Email / Create

Block Name

Emails

List of email addresses to send the email to

```
1 ["recipient1@example.com", "recipient2@example.com"]  
2  
3
```

Format



Email

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

notify

Cancel

Create



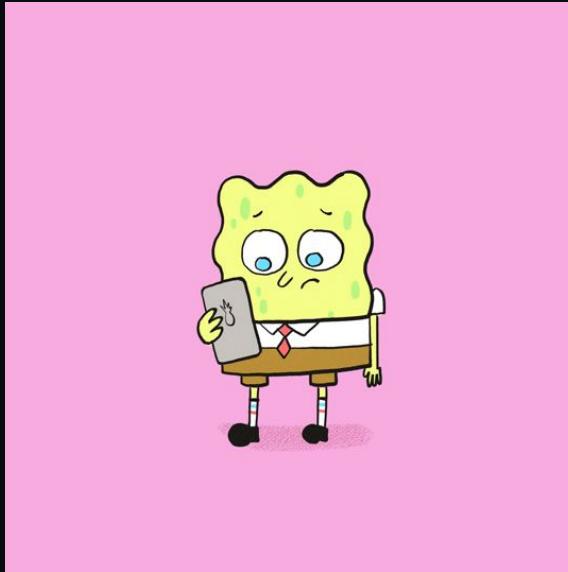
# Create an **Email** block

---



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?

