



**PREFECT ASSOCIATE  
CERTIFICATION**



# Norms

## Zoom

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

## Slack

- Use threads
- Emoji responses 😊



# Introductions

# Introductions

---



PREFECT ASSOCIATE  
CERTIFICATION

- Name
- Pronouns
- Favorite Musical Act



# Overview

# If you give an engineer a job...

---

- Could you just move this data and clean it a bit?
- Could you set up logging?
- Could you do it every night?
- Could you make it retry if it fails?
- Could you send me a message when it succeeds?
- Could you visualize the dependencies?
- Could you add caching?
- Could you add collaborators to run ad hoc - who don't code?

# What is Prefect?

---



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

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

Prefect  
helps  
teams:

Develop  
Faster

Reduce  
Failures

Increase  
Visibility

How:

- Intuitive Python-based library
- Caching
- Integrations
- World-class support

- Clear & maintainable code
- Test convenience
- Easy async
- Automatic retries

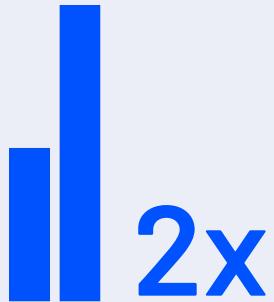
- UI
- Notifications
- Event feed
- Collaboration



## With extraordinary impacts.



of users reported much faster development cycles



average development productivity boost



reduction in pipeline errors

SOURCE: Prefect community survey, November 2021.

# Prefect makes it easy to orchestrate and observe Python code



PREFECT ASSOCIATE  
CERTIFICATION

Prefect Technologies, Inc.  
**prod-releases**

- Flow Runs
- Flows
- Work Pools
- Blocks
- Variables
- Automations
- Task Run Concurrency
- Event Feed
- Artifacts

Workspace Sharing  
Workspace Settings

## Flow Runs / sweet-mink

Completed 2023/05/04 11:55:51 AM 31s 7 task runs

Flow release-package

Logs Task Runs Subflow Runs Results Artifacts Details Parameters

Level: all Oldest to newest

May 4th, 2023

- INFO Created task run 'Get directory-0' for task 'Get directory'
- INFO Submitted task run 'Get directory-0' for execution.
- INFO Created task run 'Clone repo-0' for task 'Clone repo'
- INFO Submitted task run 'Clone repo-0' for execution.
- INFO Created task run 'Install dependencies-0' for task 'Install dependencies'
- INFO Submitted task run 'Install dependencies-0' for execution.
- INFO Created task run 'Build / cyber-impala' for task 'Build / cyber-impala'
- INFO Submitted task run 'Build / cyber-impala' for execution.
- INFO Created task run 'Release-0' for task 'Release'
- INFO Submitted task run 'Release-0' for execution.
- INFO Created task run 'Publish latest version-0' for task 'Publish latest version'
- INFO Submitted task run 'Publish latest version-0' for execution.
- INFO Created task run 'Cleanup-0' for task 'Cleanup'
- INFO Submitted task run 'Cleanup-0' for execution.
- INFO Created task run 'Send notifications-0' for task 'Send notifications'
- INFO Submitted task run 'Send notifications-0' for execution.



# Goals



# Goals

---

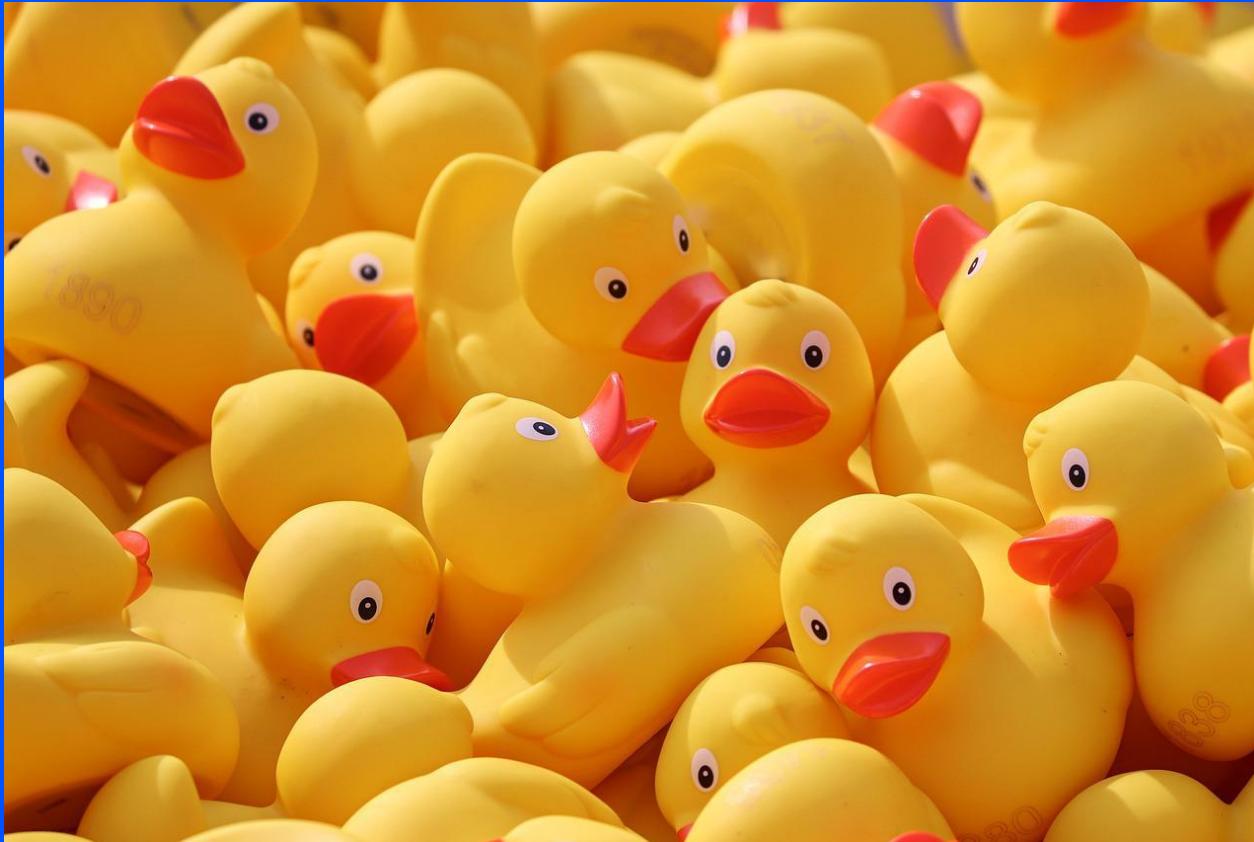
1. Competence with Prefect 2
2. Connect with each other
3. Have fun! 🎉

# Welcome to PACC!

---



PREFECT ASSOCIATE  
CERTIFICATION



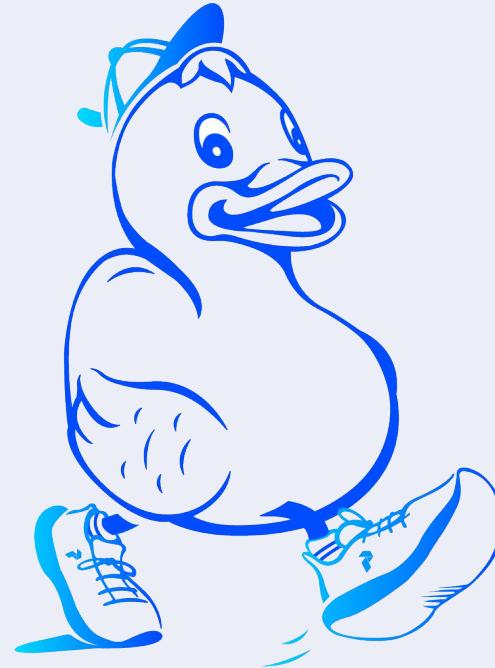
# Meet Minerva

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

- Weather forecasts ☀️RAINING
- Cat & dog facts 🐱🐶
- Stock data 📈



PREFECT ASSOCIATE  
CERTIFICATION



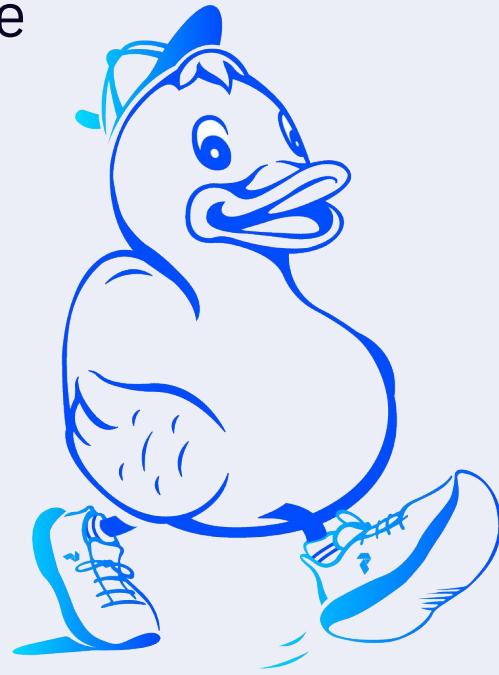


PREFECT ASSOCIATE  
CERTIFICATION

# Your charge

Help Minerva build data pipelines that are

- Reliable
- Resilient
- Observable
- Scheduled
- Easy to understand



MODULE

# 101 - Prefect basics

# 101 Agenda

---

- From Python function to Prefect flow
- Tasks
- Docs

# Initial project

---

Help Minerva get some weather data. 

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

# Fetch data: Basic Python function



```
import httpx

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

if __name__ == "__main__":
    fetch_weather(38.9, -77.0)
```

# Flows

---



PREFECT ASSOCIATE  
CERTIFICATION

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



# Make it a flow



PREFECT ASSOCIATE  
CERTIFICATION

```
from prefect import flow

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

# Flow results

---



PREFECT ASSOCIATE  
CERTIFICATION

```
17:32:35.177 | INFO      | prefect.engine - Created flow run 'observant-chihuahua' for flow 'fetch-weather'  
Most recent temp C: 15.2 degrees  
17:32:36.738 | INFO      | Flow run 'observant-chihuahua' - Finished in state Completed()
```



# Flows



# Tasks

# Pipeline

---



PREFECT ASSOCIATE  
CERTIFICATION

1. Fetch weather data and return it
2. Save data to csv and return success message
3. Pipeline to call the other two

# Fetch data function



PREFECT ASSOCIATE  
CERTIFICATION

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

# Save data function

---



PREFECT ASSOCIATE  
CERTIFICATION

```
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, lon: float):
    temp = fetch_weather(lat, lon)
    result = save_weather(temp)
    return result
```

# Tasks

---



PREFECT ASSOCIATE  
CERTIFICATION

Turn the first two functions into *tasks*



# Turn into tasks



```
from prefect import flow, task

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

# Turn into tasks



PREFECT ASSOCIATE  
CERTIFICATION

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

# Pipeline function flow



Pass the result of one task to another inside a flow

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

# Logs from run

---



PREFECT ASSOCIATE  
CERTIFICATION

```
17:34:15.961 | INFO    | prefect.engine - Created flow run 'rapid-smilodon' for flow 'pipeline'
17:34:16.735 | INFO    | Flow run 'rapid-smilodon' - Created task run 'fetch_weather-0' for task 'fetch_weather'
17:34:16.736 | INFO    | Flow run 'rapid-smilodon' - Executing 'fetch_weather-0' immediately...
17:34:17.820 | INFO    | Task run 'fetch_weather-0' - Finished in state Completed()
17:34:17.941 | INFO    | Flow run 'rapid-smilodon' - Created task run 'save_weather-0' for task 'save_weather'
17:34:17.942 | INFO    | Flow run 'rapid-smilodon' - Executing 'save_weather-0' immediately...
17:34:18.389 | INFO    | Task run 'save_weather-0' - Finished in state Completed()
17:34:18.513 | INFO    | Flow run 'rapid-smilodon' - Finished in state Completed()
```

# Tasks dos and don'ts

---

-  Do call tasks within a flow
-  Don't call tasks from other tasks
-  Do keep tasks small



## Welcome to Prefect

Prefect enables you to build and observe resilient data workflows so that you can understand, react to, and recover from unexpected changes. It's the easiest way to transform any Python function into a unit of work that can be observed and orchestrated. Just bring your Python code, sprinkle in a few decorators, and go!

With Prefect you gain:

- scheduling
- retries
- logging
- caching
- async
- notifications
- observability

Trying to implement these features from scratch is a huge pain that takes time, headaches, and money. That's why Prefect offers all this functionality and more!

The screenshot shows the Prefect Cloud interface. On the left, there is a sidebar with navigation links: Prefect Docs, Getting Started (selected), Installation, Cloud Quickstart, Tutorial, Concepts, Cloud, Host, Guides, Integrations, API References, and Community. The main area displays a "Flow Runs / sweet-mink" section. It shows a timeline from 11:55:50 AM to 11:56:20 AM with a completed flow run. The flow run details include: Completed, 2023/05/04 11:55:51 AM, 31s, 7 task runs. Below this, a task graph is visualized with nodes like "Get directory-0", "Install dependencies-0", "Release-0", "Publish latest version-0", "Cleanup-0", and "Send notifications-0". Arrows indicate the flow of tasks and dependencies between them.



## Use the docs



# Prefect information



PREFECT ASSOCIATE  
CERTIFICATION

*prefect version*

<b>Version:</b>	2.10.18
<b>API version:</b>	0.8.4
<b>Python version:</b>	3.10.8
<b>Git commit:</b>	cf177852
<b>Built:</b>	Thu, Jun 29, 2023 2:34 PM
<b>OS/Arch:</b>	darwin/arm64
<b>Profile:</b>	local
<b>Server type:</b>	ephemeral
<b>Server:</b>	
<b>Database:</b>	sqlite
<b>SQLite version:</b>	3.40.0

Run `prefect version` now

---



If you see **Server type: cloud** at the bottom run:

- `prefect profile create local` to create a new profile
- `prefect profile use local` to switch to the new profile

Now you are set to use a local Prefect server instance backed by a SQLite database.

We will discuss profiles later.

Run *prefect version* now

---



PREFECT ASSOCIATE  
CERTIFICATION

If you see *Version* lower than 2.10.18:

*pip install -U prefect*

# Resources

---

- Prefect Community Slack:

<https://www.prefect.io/slack/>

- Prefect Discourse:

<https://discourse.prefect.io/>

# Prefect codebase



PREFECT ASSOCIATE  
CERTIFICATION

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

Give it a

A screenshot of a GitHub repository page for the 'prefect' repository. The page has a dark theme. At the top, there is a navigation bar with the following items from left to right: a blue icon, the repository name 'prefect' in white, a 'Public' button, 'Edit Pins' with a dropdown arrow, 'Unwatch' with a count of '154' and a dropdown arrow, 'Fork' with a count of '1.3k' and a dropdown arrow, 'Starred' with a count of '12.2k' and a dropdown arrow, a 'main' dropdown, 'Go to file', 'Add file' with a dropdown arrow, a green 'Code' button with a dropdown arrow, and a gear icon.



You've seen how to get started with Prefect!

- From Python function to Prefect flow
- Create modular tasks that can be called **from inside** a flow
- Prefect is super Pythonic - conditionals are
- Docs are your friends
- GitHub Codebase, Slack, Discourse are helpful



PREFECT

# Lab 101

# Lab norms for breakout rooms

---

1.  Introduce yourselves
2.  Camera on (if possible)
3.  One person shares screen (if you need to leave and come back to Zoom to enable screen sharing, do that now)
4.  Everyone codes
5.  Each person talks
6.  Share code in Slack thread - learn from other groups
7.  Low-pressure, welcoming environment: lean in

## Use Open-Meteo API

- Write flow code that fetches other weather metrics
- Make at least 3 tasks in the flow
- Example: windspeed for the last hour:

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

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

MODULE

# 102 - Intro to orchestration

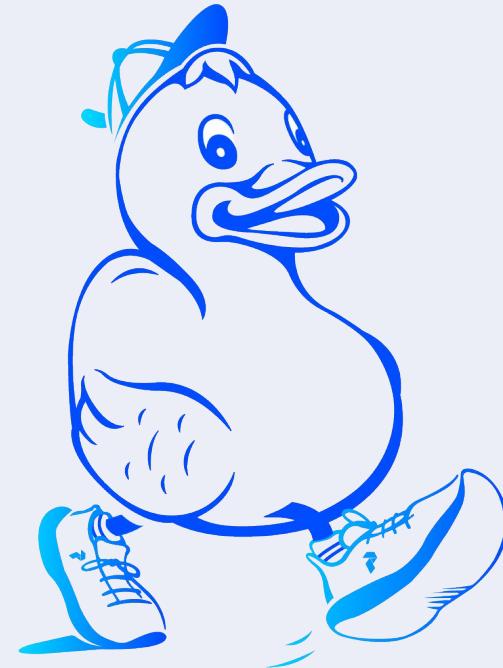
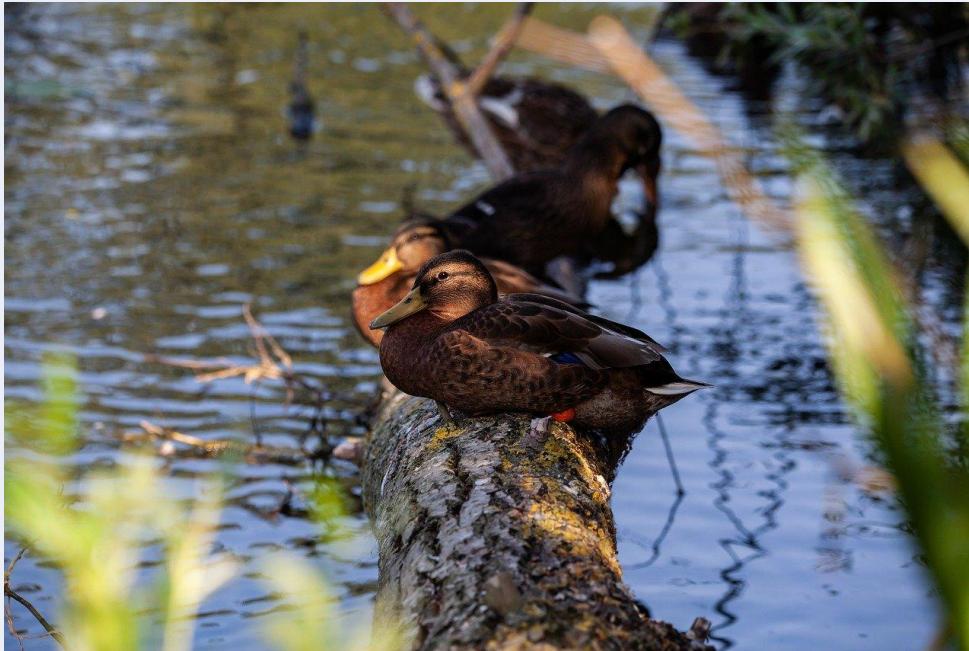
# 102 Agenda

---

- Logging
- Retries for tasks and flows
- API server
- UI
- Results
- Caching
- Learn about cats 

# Logging

Ducks on logs:



# *log\_prints*

---



PREFECT ASSOCIATE  
CERTIFICATION

Log the print statements

```
@flow(log_prints=True)
```

# 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



PREFECT ASSOCIATE  
CERTIFICATION

## Output with DEBUG logging level:

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

# Logging

Output with INFO logging level:

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

# Cat fact data

Minerva's a smart duck. She made an app to share cat facts.

Unfortunately, her app uses a cat-fact API that can be a bit buggy.



# Retries

---



PREFECT ASSOCIATE  
CERTIFICATION

Specify in task or a flow decorator

```
@task(retries=2)
```

```
@flow(retries=3)
```

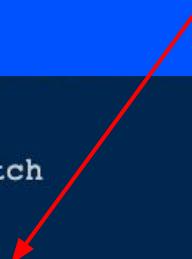
# Flow retries for an unreliable API



```
@flow(retries=4)
def fetch():
    cat_fact = httpx.get("https://f3-vyx5c2hfpq-ue.a.run.app/")
    if cat_fact.status_code >= 400:
        raise Exception()
    print(cat_fact.text)
```

# Fail then automatic retry

```
line 135, in capture_worker_thread_and_result
    result = __fn(*args, **kwargs)
  File "/Users/jeffhale/Desktop/prefect/demos/pacc/retry-flow.py", line 9, in fetch
      raise Exception()
Exception
22:23:21.040 | INFO    | Flow run 'mysterious-agouti' - Received non-final state 'AwaitingRetry' when proposing final state 'Failed' and will attempt to run again...
Cats have the largest eyes of any mammal.
22:23:22.534 | INFO    | Flow run 'mysterious-agouti' - Finished in state Completed()
```



# Retry delays

---



PREFECT ASSOCIATE  
CERTIFICATION



# Retry delay

---



PREFECT ASSOCIATE  
CERTIFICATION

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_cat_fact():
    cat_fact = httpx.get("https://f3-vyx5c2hfpq-ue.a.run.app/")
    if cat_fact.status_code >= 400:
        raise Exception()
    print(cat_fact.text)
```

# Fail then automatic retry with delay

```
Exception
22:30:35.032 | INFO    | Task run 'fetch_cat_fact-0' - Received non-final state 'AwaitingRetry' when proposing final state 'Failed' and will attempt to run again...
Blue-eyed, pure white cats are frequently deaf.
22:30:36.856 | INFO    | Task run 'fetch_cat_fact-0' - Finished in state Completed()
22:30:36.874 | INFO    | Flow run 'ivory-finches' - Finished in state Completed('All states completed')
```



PREFECT

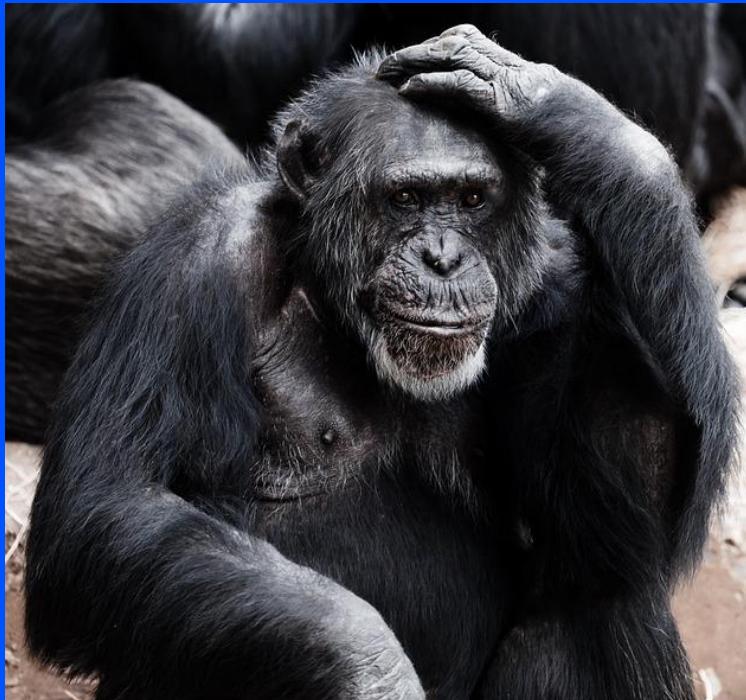
# States

# Prefect Flow Run States

---



What's the state of your workflows?



# Prefect Flow Run States



PREFECT ASSOCIATE  
CERTIFICATION

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

# Prefect Flow Run States



PREFECT ASSOCIATE  
CERTIFICATION

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.
Cancelled	CANCELLED	Yes	The run did not complete because a user determined that it should not.
Completed	COMPLETED	Yes	The run completed successfully.
Failed	FAILED	Yes	The run did not complete because of a code issue and had no remaining retry attempts.
Crashed	CRASHED	Yes	The run did not complete because of an infrastructure issue.



# Prefect server



# The Prefect CLI

---

Start commands with `prefect`

`--help` is always available



# *prefect --help*



PREFECT ASSOCIATE  
CERTIFICATION

## Commands

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

# Start a Prefect server locally

---



Open another terminal window

Run:

```
prefect server start
```

# Start a Prefect server locally



Configure Prefect to communicate with the server with:

```
prefect config set PREFECT_API_URL=http://127.0.0.1:4200/api
```

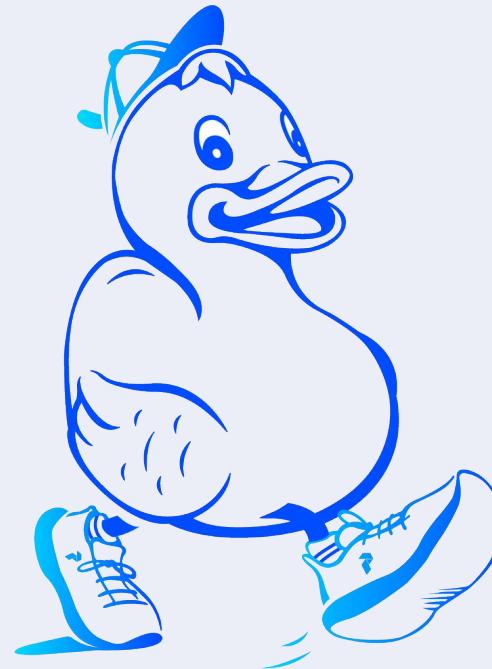
View the API reference documentation at <http://127.0.0.1:4200/docs>

Check out the dashboard at <http://127.0.0.1:4200>

Let's get Visual



**Head over to the UI!**



# Prefect UI



PREFECT ASSOCIATE  
CERTIFICATION

Flow Runs

Flows

Work Pools

Blocks

(x) Variables

Notifications

Task Run Concurrency

Artifacts

## Flow Runs

Date Range: 05/02/2023 > 05/10/2023

States: All run states

Flows: All flows

Deployments: All deployments

Work Pools: All pools

Tags: All tags

Timeline: 1s, 0.75s, 0.50s, 0.25s, 0s

Tue 02 Wed 03 Thu 04 Fri 05 Sat 06 May 07 Mon 08 Tue 09 Wed 10

1 Flow run

Search by run name

Newest to oldest

fetch-weather > righteous-aardwolf

Completed 2023/05/09 01:41:05 PM 1s None

# Prefect UI

---



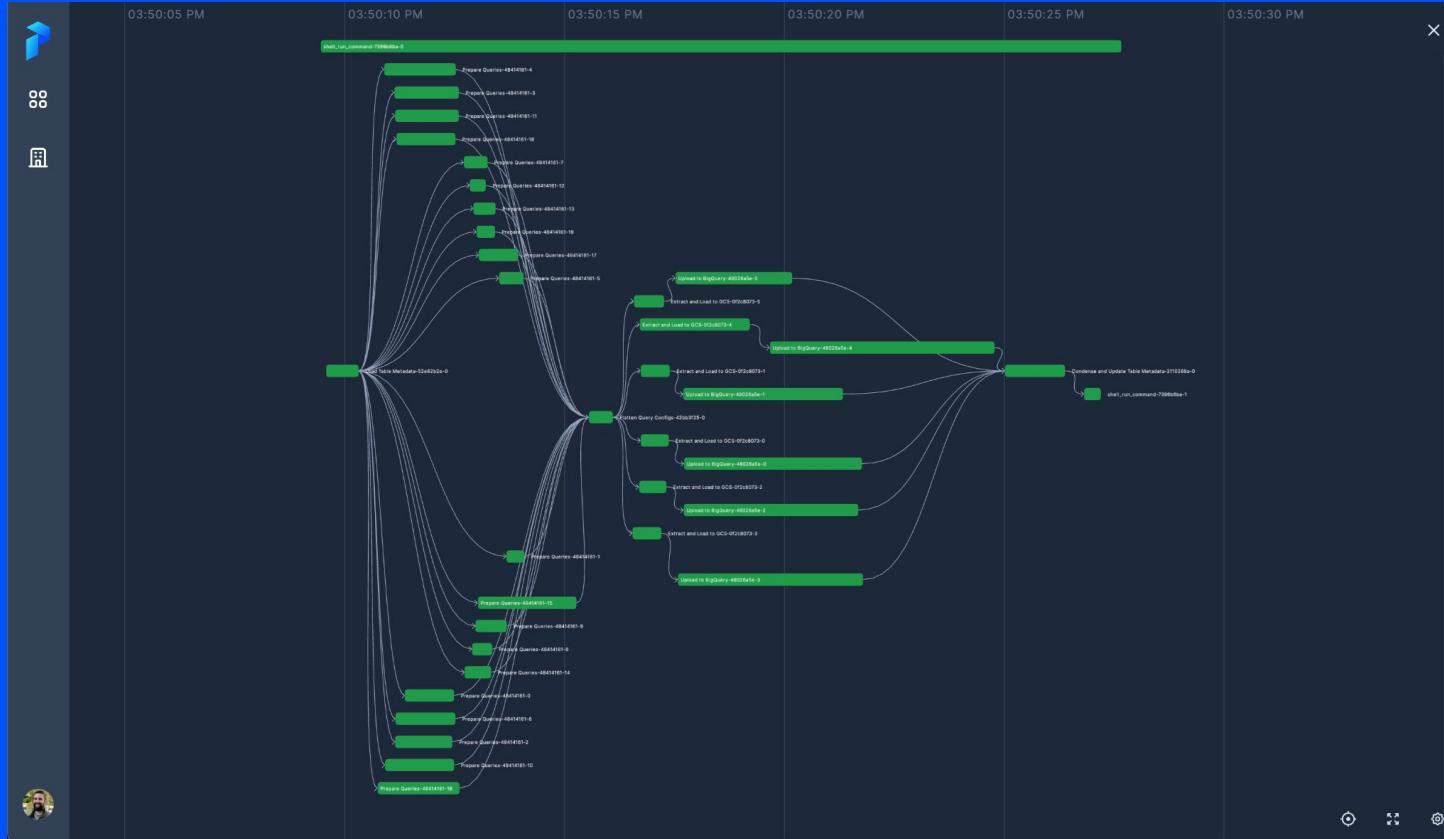
PREFECT ASSOCIATE  
CERTIFICATION

We will explore aspects of the UI throughout the course!

# Prefect UI - A much more involved flow run



**PREFECT ASSOCIATE  
CERTIFICATION**





PREFECT

# Results

# Results

---



PREFECT ASSOCIATE  
CERTIFICATION

The data returned by a flow or a task

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

1 is the result

# Results

---



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

# Persist results with `persist_result=True`

---



PREFECT ASSOCIATE  
CERTIFICATION

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

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

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

my_flow()
```

# Results



PREFECT ASSOCIATE  
CERTIFICATION

Info about a result is  
viewable in the UI

The screenshot shows the Prefect UI interface for viewing results. At the top, there is a navigation bar with tabs: Logs, Results (which is selected and highlighted with a border), Artifacts, Task Runs, Subflow Runs, and Parameters. Below the navigation bar, the main content area is titled "Flow run". Under this title, there is a section labeled "RESULT" with the timestamp "Apr 13th, 2023 at 12:09 PM" and a status indicator "Created". Further down, there is a section titled "Task runs" which lists a single task named "my\_task-0". Below this task listing, it states "Result of type DataFrame persisted to:" followed by the path "/Users/jeffhale/.prefect/storage/c65d28dcc374424ba7212a39dd19418b".

# Results



PREFECT ASSOCIATE  
CERTIFICATION

## The result in the storage folder

```
✓ .PREFECT
  ✓ storage
    {} c65d28dcc374424ba7212a39dd19418b
      storage > {} c65d28dcc374424ba7212a39dd19418b > ...
        1   {"serializer": {"type": "pickle", "picklelib": "cloudpickle", "picklelib_version": "2.2.1"},}
        2   "data": "gAWVxwIAAAAAACMEXBhbmRhcy5jb3JlLmZyYW1llIwJRGF0YUZyYW1llJ0UKYGuFZQojARfbWdy\\nIwec"
        3   "prefect_version": "2.10.3"
        4
        5
        6
  ✓ memo_store.toml
  ≡ prefect.db
  ✓ profiles.toml
```



PREFECT

# Caching

# Caching

---

What?

Why?

 task only

Requires persisting results

# Caching: *cache\_key\_fn*

---



PREFECT ASSOCIATE  
CERTIFICATION

```
@task(cache_key_fn=task_input_hash)
```

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

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

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

# Caching



PREFECT ASSOCIATE  
CERTIFICATION

## First run

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

## Second run

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

# Caching



PREFECT ASSOCIATE  
CERTIFICATION

What will happen if you run with a different argument?

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

# Caching: *cache\_expiration*



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

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

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

# Caching

---



PREFECT ASSOCIATE  
CERTIFICATION

You can cache to S3 or other cloud storage provider.

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

Blocks are a future topic. 😊



You've seen more of the power of Prefect tasks and flows.

- Logging
- States
- Retries for tasks and flows
- API server
- UI
- Results
- Caching



PREFECT

# Lab 102

# Lab 102

---

- Use a flow that grabs weather data from open-meteo
- Add retries
- Start a Prefect server instance
- Run your flow
- Inspect in the UI
- Stretch: add caching

MODULE

# 103 - Cloud & Blocks

# 103 Agenda

---

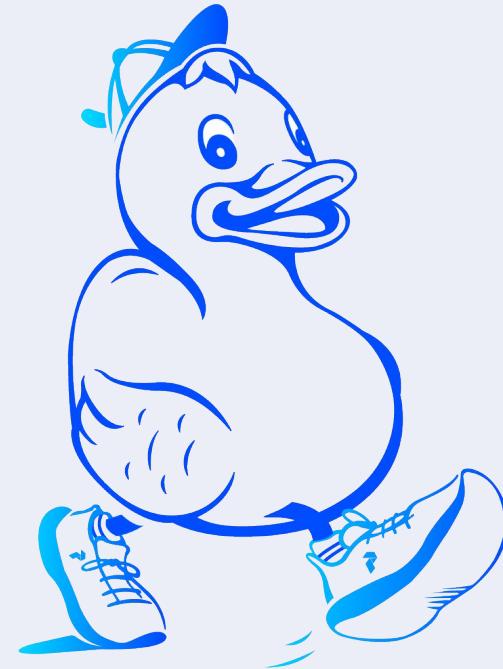
- Cloud
- Profiles
- Blocks 
- Automations
- Events



PREFECT ASSOCIATE  
CERTIFICATION

# To the cloud

Like other ducks, Minerva is into clouds.





Go to [app.prefect.cloud](https://app.prefect.cloud) in browser

- Sign up or sign in



## Authenticate CLI through browser or API key

Run: *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
```



Or paste an API key

Manually create an API key from Prefect Cloud in the UI

# Prefect Cloud - API Key



PREFECT ASSOCIATE  
CERTIFICATION

The screenshot shows the Prefect Cloud interface. At the top, there's a navigation bar with back, forward, and refresh icons, followed by the URL `app.prefect.cloud`. Below the URL is a sidebar with three icons: a blue cube, two overlapping circles, and a grid. The main area is titled "Workspaces" with a "+" button. A workspace card for "jeffgmai" (highlighted in blue) is displayed, showing the name "data-time", a user icon, and the number "1". To the right of the workspace card is a timeline visualization with green, red, and grey bars. In the bottom right corner of the workspace card, there's a three-dot menu icon. A red arrow points from the bottom left towards the user profile menu. The profile menu itself is a dark card with a sun/moon toggle switch, the name "Jeff Hale", and a gear icon. Below the name are links to "Prefect Docs", "Prefect Cloud REST API Docs", "Community Forum", "Community Slack", and "Sign Out". A small circular profile picture of Jeff Hale is at the bottom left of the menu.

# Prefect Cloud - API Key



PREFECT ASSOCIATE  
CERTIFICATION

The screenshot shows the Prefect Cloud user interface. On the left, there's a sidebar with a profile picture for 'Jeff Hale' and several menu items: 'Profile' (with a person icon), 'API Keys' (which is highlighted with a blue background and has a key icon), 'Billing' (with a credit card icon), and 'Preferences' (with a gear icon). A red arrow points from the 'API Keys' button in the sidebar to the 'API Keys' section header on the main page. The main page has a dark background. At the top, it says 'API Keys' with a '+' button. Below that, it shows '4 API keys'. There's a search bar with the placeholder 'Search API keys'. A table then lists four API keys:

Name	Created	Expiration
cli-e5c61a7f-0232-413d-8353-320e4ffc9311	Apr 13th, 2023	2023/05/13 12:00:00 AM
cli-d207f44e-ec61-4943-9ad6-f9e1bd727875	Apr 19th, 2023	2023/05/19 12:00:00 AM
demos	Aug 28th, 2022	Never
jkey	Jan 7th, 2023	Never

# Prefect Cloud - API Key



PREFECT ASSOCIATE  
CERTIFICATION

### Create API Key

X

Name

Expiration Date

09/30/2023

Never Expire

Cancel Create



PREFECT

# Profiles

# Prefect profiles - list



PREFECT ASSOCIATE  
CERTIFICATION

```
prefect profile ls
```

## Available Profiles:

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



Persist settings that you can switch between

Common for switching between

- Cloud and server
- Different Cloud workspaces
- Different settings like logging level

# Profiles

---



PREFECT ASSOCIATE  
CERTIFICATION

Create: `prefect profile create my_cloud_profile`

Inspect: `prefect profile inspect my_cloud_profile`

Select: `prefect profile use my_cloud_profile`

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



PREFECT

# Prefect Cloud



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

# Prefect Cloud Workspaces

---



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





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

# Prefect Cloud - Organization

---



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

# Prefect Cloud - Enterprise

---



PREFECT ASSOCIATE  
CERTIFICATION

- Custom Roles
- SSO SCIM
- Custom most everything 😊

# Prefect Cloud



PREFECT ASSOCIATE  
CERTIFICATION

## Personal

Start with managed orchestration.

**Free  
Forever**

[CREATE YOUR WORKSPACE](#)

3 free Users

1 free Workspace

7 day history

## Organization

Prefect for scaling out with your team. Get a free 14 day trial.

Starts at

**\$450/mo**

[START YOUR FREE TRIAL](#)

Read-Only Users

Service Accounts + Enforced SSO

30 day history

## Enterprise

Security, compliance and custom configurations.

**Get an estimate**

[TALK TO SALES](#)

Enterprise support

SCIM + Custom RBAC

Custom configurations

## Dedicated Cloud

Dedicated Instance of Prefect Cloud

**Get in touch**

[TALK TO A PLATFORM ENGINEER](#)

No Resource Limits

Performance x Security

Implementation and Extended SLA

## Organization level

- Admin
- Member

## Workspace level

- Viewer
- Runner
- Developer
- Owner



PREFECT

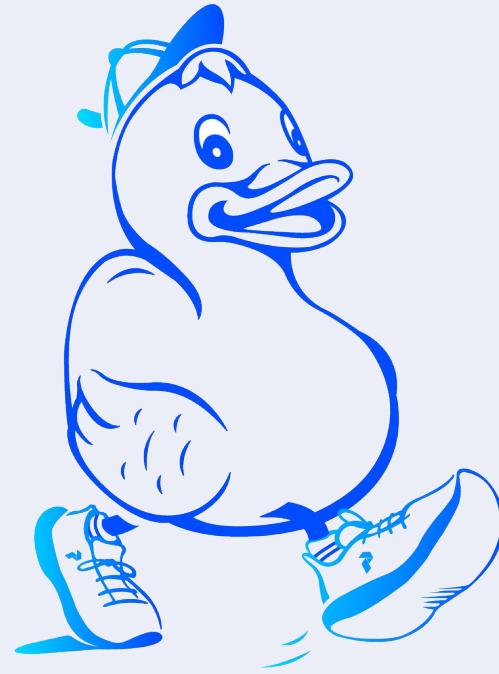
# Blocks



PREFECT ASSOCIATE  
CERTIFICATION

# Blocks

Blocks are a cool Prefect feature



# Blocks

---

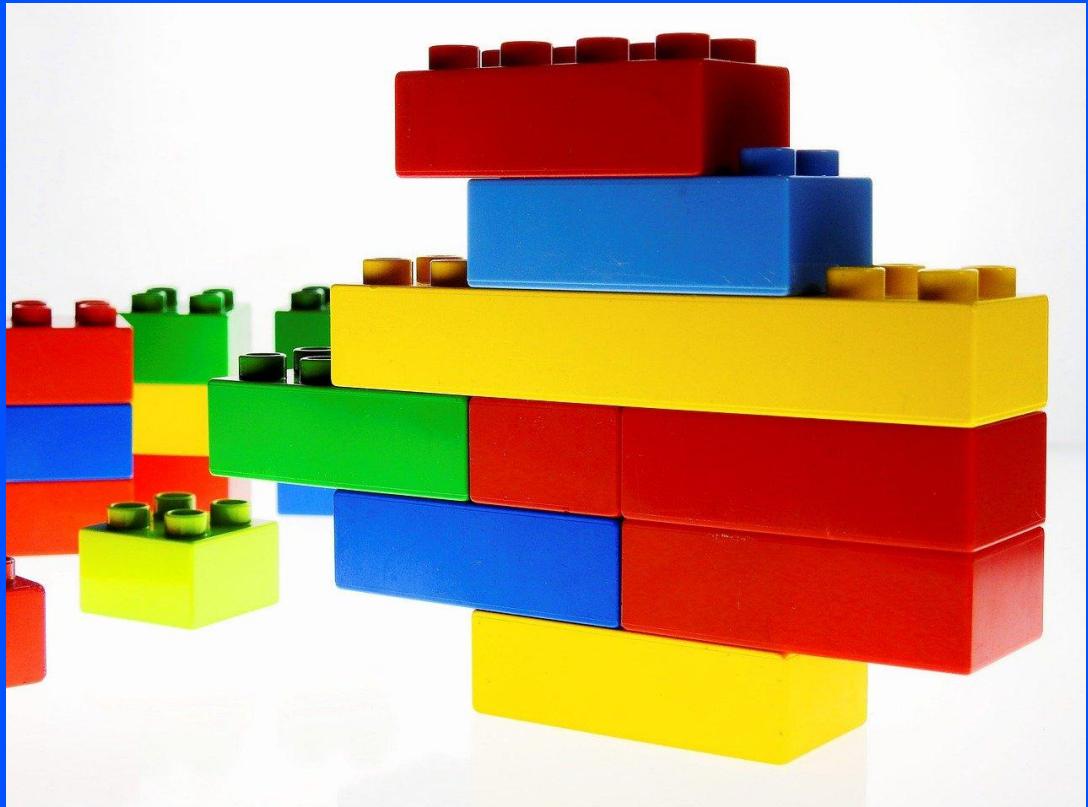


PREFECT ASSOCIATE  
CERTIFICATION

Configuration

+

Code



# Blocks in UI



PREFECT ASSOCIATE  
CERTIFICATION

Sales Engineering  
tay-sandbox

- Flow Runs
- Flows
- Deployments
- Work Pools
- Blocks**
- (x) Variables
- Automations
- Task Run Concurrency
- Event Feed
- Artifacts

## Blocks +

Jun 20th, 2023 12:00 AM Jun 20th, 2023 11:59 PM

4 Blocks Search blocks Capability: all Type: all

Name	Capabilities
dbt Cloud Credentials / <a href="#">my-dbt-creds</a>	
GitHub / <a href="#">my-github-block</a>	get-directory
S3 Bucket / <a href="#">result-storage</a>	get-directory put-directory read-path write-path
Slack Webhook / <a href="#">my-slack-block</a>	notify

# Create a Block from the UI



PREFECT ASSOCIATE  
CERTIFICATION

 <b>Remote File System</b> Store data as a file on a remote file system. Supports any remote file system supported by 'fsspec'. The file system is specified using a protocol. For example, "s3://my-...." <code>get-directory</code> <code>put-directory</code> <code>read-path</code> <code>write-path</code>  <b>Add +</b>	 <b>S3</b> Store data as a file on AWS S3. <code>get-directory</code> <code>put-directory</code> <code>read-path</code> <code>write-path</code>  <b>Add +</b>	 <b>S3 Bucket</b> Block used to store data using AWS S3 or S3-compatible object storage like MinIO. This block is part of the prefect-aws collection. Install prefect-aws with 'pip install...' <code>get-directory</code> <code>put-directory</code> <code>read-path</code> <code>write-path</code>  <b>Add +</b>
 <b>Secret</b> A block that represents a secret value. The value stored in this block will be obfuscated when this block is logged or shown in the UI.  <b>Add +</b>	 <b>Shell Operation</b> A block representing a shell operation, containing multiple commands. For long-lasting operations, use the trigger method and utilize the block as a context...  <b>Add +</b>	 <b>Slack Credentials</b> Block holding Slack credentials for use in tasks and flows. This block is part of the prefect-slack collection. Install prefect-slack with 'pip install prefect-slack' to use this block.  <b>Add +</b>
 <b>Slack Incoming Webhook</b>	 <b>Slack Webhook</b>	 <b>SMB</b>

# Creating a block from the UI.



PREFECT ASSOCIATE  
CERTIFICATION

## Blocks / Choose a Block / Slack Webhook / Create

### Block Name

my-slack-block

### 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

# Under the hood, block types are Python Classes



PREFECT ASSOCIATE  
CERTIFICATION

```
from prefect.blocks.core import Block

class Dbt(Block):
    """
    A block for interacting with dbt
    """

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

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

# Under the hood, block types are Python Classes



PREFECT ASSOCIATE  
CERTIFICATION

Blocks / jaffle-shop

Paste this snippet into your flows to use this block.

```
from dataplatform.blocks import Dbt

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

Workspace  
[default](#)

Path To Dbt Project  
dbt\_jaffle\_shop

Retries  
3

Retry Delay Seconds  
10



**Dbt**  
A block for interacting with dbt  
[dbt\\_cli](#)  
[dbt\\_run\\_from\\_manifest](#)



Reusable, modular, configuration + code

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



PREFECT

# Cloud Only Features

# Automations + Events API



PREFECT ASSOCIATE  
CERTIFICATION

jeffgmail  
**data-time**

Flow Runs

Flows

Work Pools

Blocks

(x) Variables

Automations

Task Run Concurrency

Event Feed

Artifacts

## Flow Runs

### Date Range

05/02/2023 > 05/10/2023

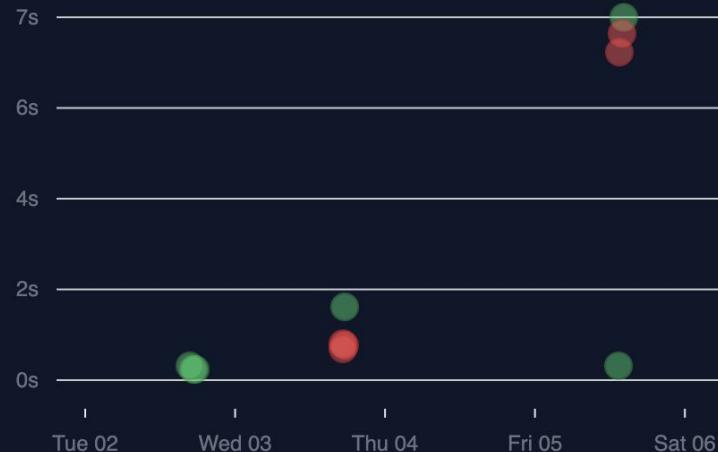


### Flows

All flows

### Deployments

All deployments





PREFECT

Automations ⚡

Cloud only

Flexible framework

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

# Automation notifications

---

For example,

- Send an email if a flow with tag prod fails
- Start a back up work-queue if flows are late for longer than 10 minutes.
- Kick off a failure handling deployment if data quality check fails.

# Automation notifications



PREFECT ASSOCIATE  
CERTIFICATION

The screenshot shows the Prefect Cloud interface with a dark theme. On the left, a sidebar menu lists various resources: Flow Runs, Flows, Work Pools, Blocks, Variables, and Automations. The 'Automations' item is highlighted with a blue bar at the bottom of the sidebar. The main content area is titled 'Automations' and features a card for an automation named 'email-jeff-cat'. The card description is 'Email Jeff when cat fact fails.' and the action is 'Send a notification'. A toggle switch is shown as 'On' (blue), and a three-dot menu icon is available for more options.

jeffgmail  
data-time

Automations +

Documentation

email-jeff-cat

Email Jeff when cat fact fails.

Action Send a notification

⋮

Flow Runs

Flows

Work Pools

Blocks

(x) Variables

Automations

# Automation notifications



PREFECT ASSOCIATE  
CERTIFICATION

Automations / Create Documentation 

01 Trigger      02 Actions      03 Details

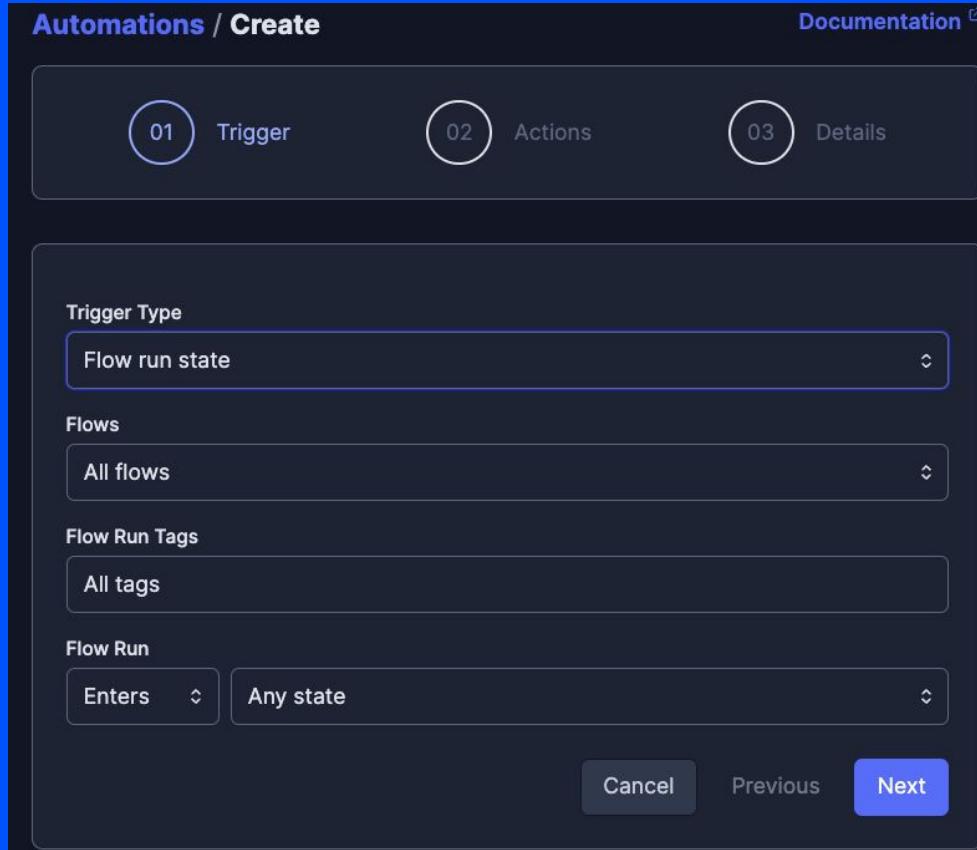
**Trigger Type**  
Flow run state 

**Flows**  
All flows 

**Flow Run Tags**  
All tags 

**Flow Run**  
Enters  Any state 

**Buttons**  
Cancel Previous Next



# Automation notifications



PREFECT ASSOCIATE  
CERTIFICATION

Automations / Create Documentation

Trigger 02 Actions 03 Details

Action 1

Action Type

Send a notification

Block

Add +

Subject

Prefect flow run notification

Body

```
Flow run {{ flow.name }}/{{ flow_run.name }} entered state `{{ flow_run.state.name }}` at {{ flow_run.state.timestamp }}.  
Flow ID: {{ flow_run.flow_id }}  
Flow run ID: {{ flow_run.id }}  
Flow run URL: {{ flow_run|ui_url }}
```

# Automation notifications



PREFECT ASSOCIATE  
CERTIFICATION

**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.

8 Blocks

Search blocks

Capability: 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...

notify

Add +

**Mattermost Webhook**

Enables sending notifications via a provided Mattermost webhook.

notify

Add +

**Microsoft Teams Webhook**

Enables sending notifications via a provided Microsoft Teams webhook.

notify

Add +

**Opsgenie Webhook**

Enables sending notifications via a provided Opsgenie webhook.

notify

Add +

# Automation notifications

**Blocks / Choose a Block / Email / Create**

**Block Name**

**Emails**  
List of email addresses to send the email to

**Create**



**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...  
**notify**



PREFECT

# Events

# Events

---



PREFECT ASSOCIATE  
CERTIFICATION

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

Workspace Events Beta

Block events are in Beta. To gather events from blocks, including storage and infrastructure events, enable client side events with `prefect config set PREFECT_EXPERIMENTAL_ENABLE_EVENTS_CLIENT=True`.

## Related Resource

All resources

## Events

All events

Mar 19th, 06:10 PM - Mar 22nd, 10:48 PM

Mar 17th, 2023  
12:00 AM

1H 1D 1W

Mar 21st, 2023  
06:32 PMMar 23rd, 2023  
11:59 PM

Block Document collection-registry-github-token

5426 events

Block Document marvin-monday-slack-posts-k8s-job

1683 events

Block Document pd-on-call-slack-update-k8s-job

819 events

Block Document closed-won-k8s-job

756 events

Block Document collection-registry-update-job

460 events

Block Document collection-registry-result-storage

93 events

See More

10:30:52 PM • Flow run scheduled

Mar 22nd, 2023 prefect.flow-run.Scheduled

## Resource

Flow run steady-ocelot

## Related Resources

Flow Monday Flow Deployment Marvin Monday Slack Posts Work Queue internal-tools-cluster 1 other resource

10:30:28 PM • Block kubernetes job get client called

Mar 22nd, 2023 prefect.block.kubernetes-job.get\_client.called

## Resource

Block Document marvin-monday-slack-posts-k8s-job

## Related Resources

1 resource

10:30:26 PM • Block kubernetes job get batch client called

Mar 22nd, 2023 prefect.block.kubernetes-job.get\_batch\_client.called

## Resource

Block Document marvin-monday-slack-posts-k8s-job

## Related Resources

1 resource

10:30:24 PM • Flow run completed

# Events

---



PREFECT ASSOCIATE  
CERTIFICATION

Can represent

- API calls
- state transitions
- changes in environment

# Event feeds

---



PREFECT ASSOCIATE  
CERTIFICATION

Power several Cloud features

- flow run logs
- audit logs
- automations

# Custom Events can be Emitted



```
from prefect.events import emit_event
```

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



## You've learned

- how to use Prefect Cloud
- about blocks
- how to create Prefect Cloud automations
- where to see events



PREFECT

# Lab 103

- Use a previously made flow
- Use Prefect Cloud
- Make an email notification automation for flow run completion
- Run the flow a few times from the CLI
- See the event feed in the UI
- Stretch: create a Slack notification automation

MODULE

# 104 - Deployments

# 104 Agenda

---



PREFECT ASSOCIATE  
CERTIFICATION

- Hybrid model
- Deployments
- Create a deployment with interactive prompts
- Work pools
- Workers
- Run a deployment

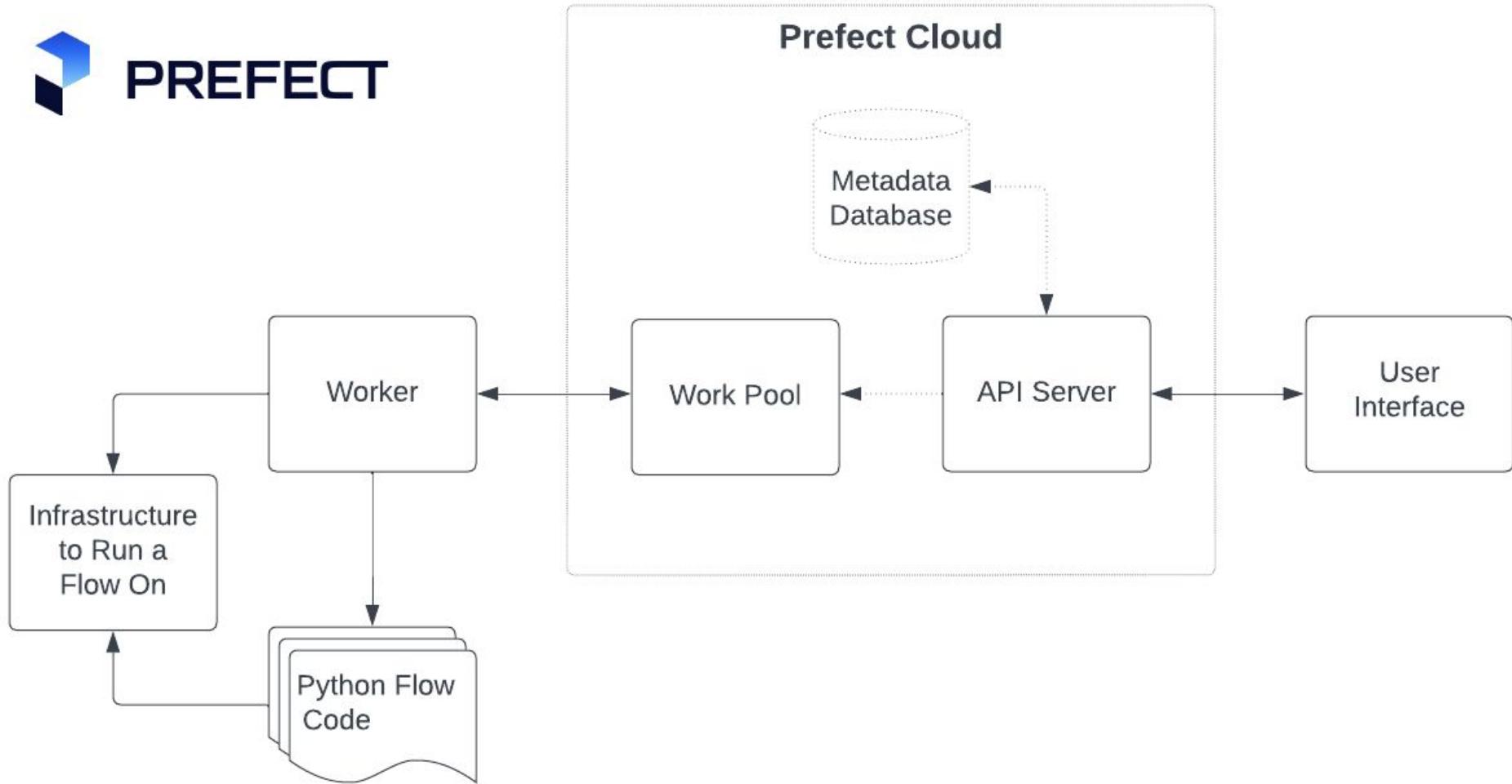


PREFECT

# Hybrid Model



PREFECT



# Hybrid model

---



PREFECT ASSOCIATE  
CERTIFICATION

- Your flow code runs on your infrastructure
- Your flow code is stored on your storage (GH, AWS, Docker image, etc)
- Prefect Cloud stores metadata and logs
- Data encrypted at rest
- Prefect Technologies, Inc. is SOC2 Type II compliant

<https://www.prefect.io/security>

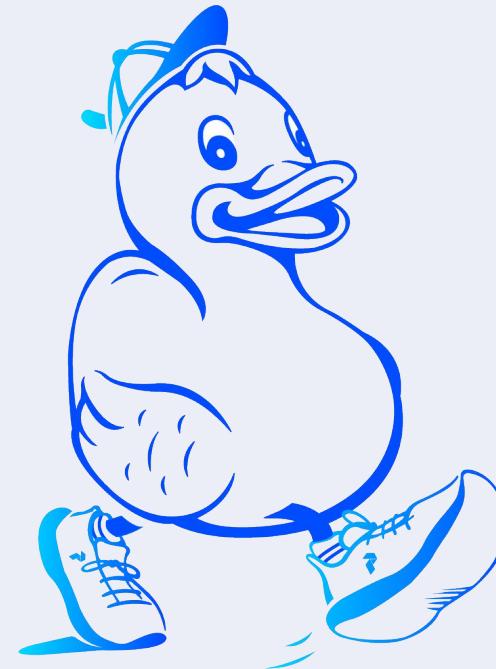


PREFECT ASSOCIATE  
CERTIFICATION

# Deployments

Minerva wants to step up her Prefect orchestration game.

*Deployments* bring features needed for production including scheduling.



# Deployment

---



PREFECT ASSOCIATE  
CERTIFICATION

Contains info about all the things your flow needs to run in production.

# Deployments: ETL code



PREFECT ASSOCIATE  
CERTIFICATION

```
@task
def fetch_cat_fact():
    return httpx.get("https://catfact.ninja/fact?max_length=140").json()["fact"]

@task
def formatting(fact: str):
    return fact.title()

@task
def write_fact(fact: str):
    with open("fact.txt", "w+") as f:
        f.write(fact)
    return "Success!"
```

# Deployments: ETL code



PREFECT ASSOCIATE  
CERTIFICATION

```
@flow
def pipe():
    fact = fetch_cat_fact()
    formatted_fact = formatting(fact)
    msg = write_fact(formatted_fact)
    print(msg)
```



Guided deployment  
creation

A decorative graphic element consisting of three blue circles of increasing size from left to right, connected by thin white lines. The circles are partially cut off by the edges of the slide.

# Send deployment to server

From the **root of your repo** run:

```
prefect deploy
```

Choose the flow you want to put into a deployment

```
? Select a flow to deploy [Use arrows to move; enter  
to select; n to select none]
```

	Flow Name	Location
>	pipe	104/flows.py
	hello_flow	102/caching1.py
	log_it	102/logflow.py

# Send deployment to server

Enter a deployment name and then *n* for no schedule.

```
? Deployment name (default): first_deploy
? Would you like to schedule when this flow runs? [y/n] (y): n
```

# Create a work pool



PREFECT ASSOCIATE  
CERTIFICATION

```
? Looks like you don't have any work pools this flow can be deployed to. Would you like to
create one? [y/n] (y): y
? What infrastructure type would you like to use for your new work pool? [Use arrows to
move; enter to select]
```

	Type	Description
>	process	Execute flow runs as subprocesses on a worker. Works well for local execution when first getting started.
	ecs	Execute flow runs within containers on AWS ECS. Works with existing ECS clusters and serverless execution via AWS Fargate. Requires an AWS account.



# Work Pools

# Work pool

---



PREFECT ASSOCIATE  
CERTIFICATION

- Server side
- Where scheduled flow runs go
- Typed by infrastructure (e.g. Docker, Kubernetes)
- Created via UI or CLI (Interactive prompt or command)

# Work pools



PREFECT ASSOCIATE  
CERTIFICATION

Give your work pool a name.

Or, if you have existing work pools, choose one

? Which work pool would you like to deploy this flow to? [Use arrows to move; enter to select]

	Work Pool Name	Infrastructure Type	Description
>	docker-work	docker	
	local-work	process	
	<b>my-pool</b>	process	
	prod-pool	kubernetes	
	staging-pool	kubernetes	
	zoompool	process	



PREFECT

# Flow code storage



# Specify flow code storage

Prefect auto-detects if you are in a git repo

```
? Your Prefect workers will need access to this flow's code in order to run it. Would you like your workers to pull your flow code from its remote repository when running this flow? [y/n] (y): y
? Is https://github.com/discover/pacc-2023.git the correct URL to pull your flow code from? [y/n] (y): y
? Is main the correct branch to pull your flow code from? [y/n] (y): y
? Is this a private repository? [y/n]: n
```

```
Deployment 'pipe/first_deploy' successfully created with id  
'0f45657b-86d7-4141-a56a-e1ce47b90f1d'.
```

# Deployments in the UI



The deployment lives on the server. See it in the UI.

A screenshot of the Prefect UI interface. On the left, there is a sidebar with the user's name "jeffgmail" and a section titled "data-time". Below this are several menu items: "Flow Runs" (with a flow icon), "Flows" (with a flow icon), "Deployments" (with a circular icon containing a play button), "Work Pools" (with a stack of cylinders icon), and "Blocks" (with a cube icon). The main content area has a dark header with the title "Deployments" and the subtitle "14 Deployments". Below this is a table with three rows. Each row contains a checkbox and a deployment name: "Name" (checkbox is empty), "buy/buy\_deployment" (checkbox is empty), and "check-links/check-links" (checkbox is empty).

# Save deployment configuration to `prefect.yaml`



```
? Would you like to save configuration for this deployment for  
faster deployments in the future? [y/n]: y
```

```
Deployment configuration saved to prefect.yaml! You can now deploy  
using this deployment configuration with:
```

```
$ prefect deploy -n first_deploy
```

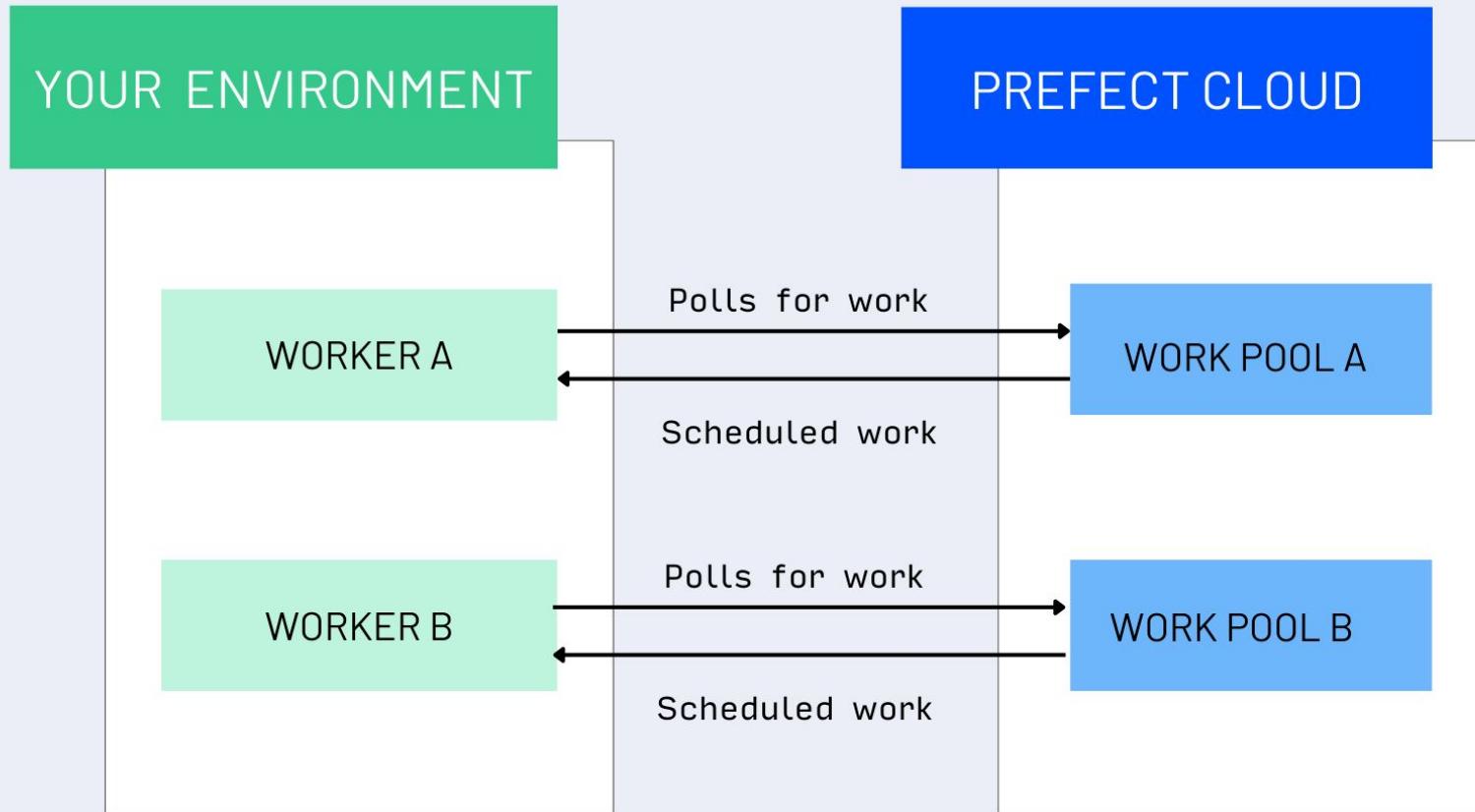
```
You can also make changes to this deployment configuration by  
making changes to the prefect.yaml file.
```



PREFECT

# Workers





- Long-running process on the client
- Poll for scheduled flow runs from work pools
- Must match a work pool to pick up work
- In a separate terminal run:

```
prefect worker start -p my_pool
```

# Recap

---



PREFECT ASSOCIATE  
CERTIFICATION

- Deployment & work pool created on Prefect Cloud
- Worker runs on local machine
- Worker polls Prefect Cloud, looking for scheduled work in the *my\_pool* work pool
- Deployment configuration saved to *prefect.yaml*

# Run a deployment from the UI



PREFECT ASSOCIATE  
CERTIFICATION

Choose *Quick run* from the three dot menu

The screenshot shows the Prefect UI interface. On the left is a sidebar with the following items:

- jeffprefectio
- prod-workspace-jeff
- Flow Runs
- Flows
- Deployments
- Work Pools
- Blocks
- (x) Variables
- Automations
- Task Run Concurrency
- Event Feed
- Artifacts

The main area is titled "Deployments" and shows "2 Deployments". It includes a search bar, a sorting dropdown set to "A to Z", and a "All tags" button. The table lists two deployments:

	Name	Schedule	Tags	Applied By
<input type="checkbox"/>	pipe/first-deploy			jeffprefectio <input checked="" type="checkbox"/> 
<input type="checkbox"/>	main/my-deploy			jeffpre

A context menu is open over the first deployment row, specifically over the three-dot menu icon. The menu options are:

- Quick run
- Custom run
- Copy ID
- Edit
- Delete

# View the flow run logs in the UI



PREFECT ASSOCIATE  
CERTIFICATION

Feb 14th, 2023

```
[INFO] Downloading flow code from storage at '/Users/jeffhale/Desktop/prefect/demos/pacc/pacc-nyc-demos-2023-02-13' 04:31:29 PM
[INFO] Created task run 'fetch_cat_fact-0' for task 'fetch_cat_fact' 04:31:31 PM
[INFO] Executing 'fetch_cat_fact-0' immediately... 04:31:31 PM
[INFO] Finished in state Completed() 04:31:32 PM
[INFO] fetch_cat_fact-0

[INFO] Created task run 'formatting-0' for task 'formatting' 04:31:32 PM
[INFO] Executing 'formatting-0' immediately... 04:31:32 PM
[INFO] Finished in state Completed() 04:31:33 PM
[INFO] formatting-0

[INFO] Created task run 'write_fact-0' for task 'write_fact' 04:31:33 PM
[INFO] Executing 'write_fact-0' immediately... 04:31:33 PM
[INFO] Finished in state Completed() 04:31:34 PM
[INFO] write_fact-0
```

# Schedule a run - what happened?

---

- Running worker finds scheduled work in *my\_pool* work pool.
- Worker and work pool are typed. *process* in this case.
- Worker creates a local subprocess to kick off flow run.
- Flow code cloned from GitHub into temporary directory.
- Flow code runs.
- Metadata and logs sent to Prefect Cloud.
- Temporary directory deleted.



PREFECT

# Parameters

# Parameters - argument values for flow function

---

## Options

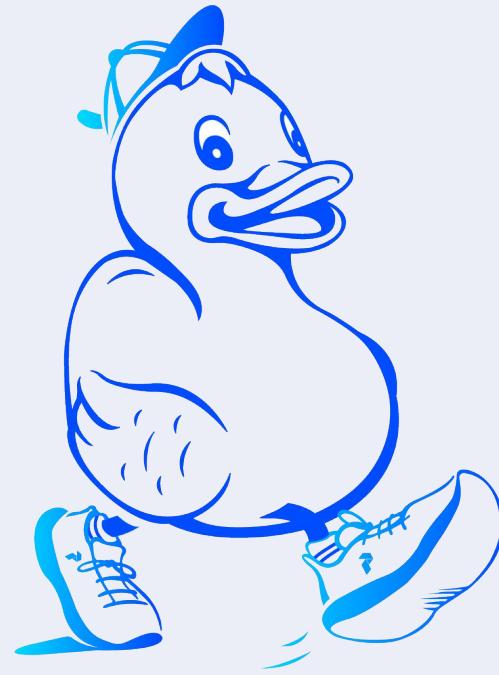
- Make default arguments in flow function definition
- Can override in deployment config
- Can override both of the above at runtime when you run a deployment



PREFECT ASSOCIATE  
CERTIFICATION

## Parameters need values

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



# Parameters in the UI



PREFECT ASSOCIATE  
CERTIFICATION

## Deployments / taxi\_s3\_data

Runs      **Parameters**      Infra Overrides      Description

2 parameters

Search parameters

Key	Override	Default	Type
train_path		./data/green_tripdata_2021-01.parquet	string
val_path		./data/green_tripdata_2021-02.parquet	string

**Flow**  
main-flow-s3

**Work Pool**  
zoompool

**Work Queue**  
default

**Schedule**  
[Add](#)

**Created**  
2023/05/12 07:34:35 PM

**Created By**  
jeffgmail

# Parameters in the UI



PREFECT ASSOCIATE  
CERTIFICATION

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

**Parameters**

Default   Custom   JSON

**train\_path (Optional)**

```
./data/green_tripdata_2021-01.parquet
```

**val\_path (Optional)**

```
./data/green_tripdata_2021-02.parquet
```



## You've learned

- how to create *deployments* via interactive CLI with *prefect deploy*
- how to start a *worker* that polls a *work pool* looking for scheduled *flow runs*
- how to run a *flow* from the UI
- how to work with parameters



PREFECT

# Lab 104

## Use a flow from the 103 lab

- Make a deployment via interactive CLI commands
- Start a worker
- Run a deployment from the UI
- See the flow run logs in the UI
- Stretch: Create a deployment that has default parameters & override the default parameters at runtime in the UI

MODULE

# 105 - Workflow patterns, prefect.yaml, and schedules



- Workflow patterns with
  - subflows
  - *run\_deployment*
  - event webhooks with automations
- Explore *prefect.yaml* file for deployments
- Create schedules
- Pause and resume schedules and work pools



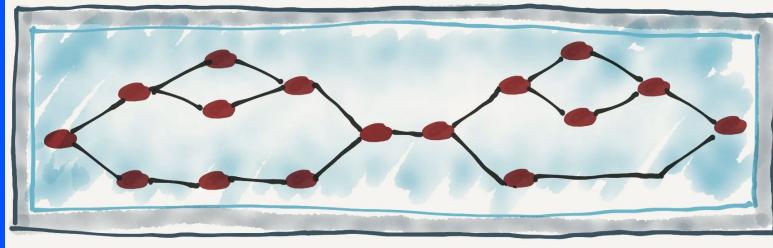
PREFECT

# Workflow Patterns

# Workflow patterns

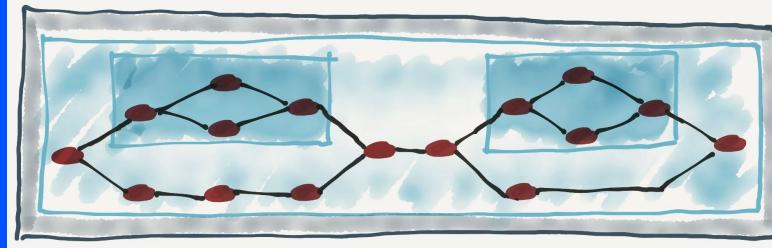


PREFECT ASSOCIATE  
CERTIFICATION



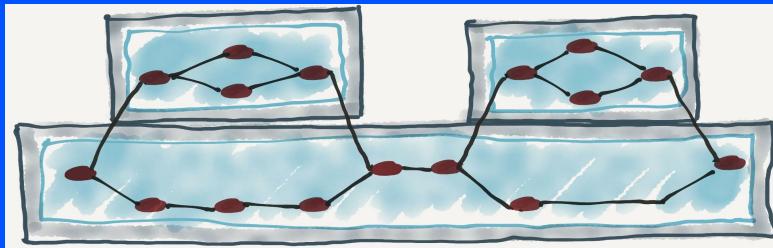
● Task    ■ Flow    □ Intra

Monoflow



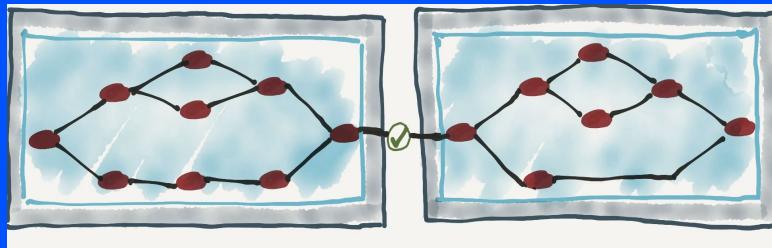
● Task    ■ Flow    □ Intra

Flow of subflows



● Task    ■ Flow    □ Intra

Flow of deployments



● Task    ■ Flow    □ Intra    ○ Event

Event triggered flow



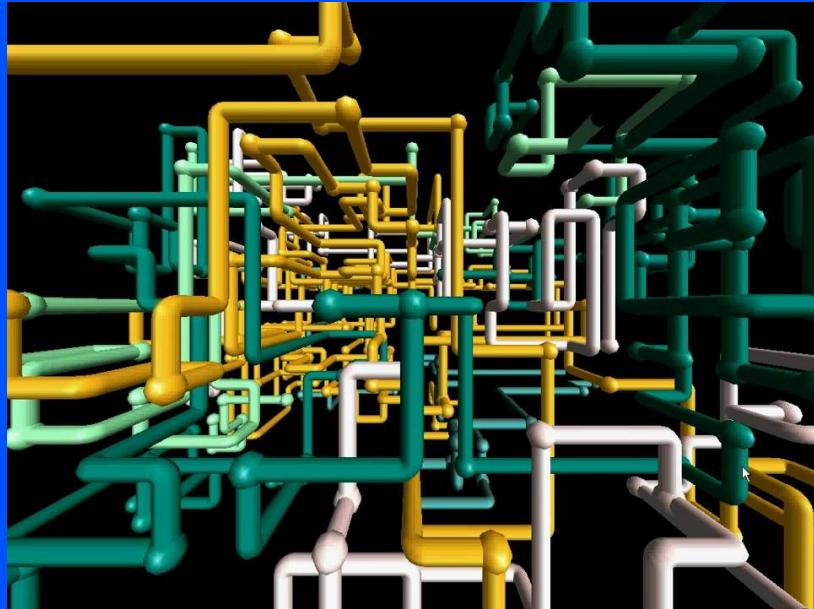
PREFECT

# Subflows

# Subflows

---

- A flow that calls another flow
- Useful for grouping related tasks



# Subflows



```
@flow
def fetch_cat_fact():
    return httpx.get("https://catfact.ninja/fact?max_length=140").json()["fact"]

@flow
def fetch_dog_fact():
    return httpx.get(
        "https://dogapi.dog/api/v2/facts",
        headers={"accept": "application/json"},
    ).json()["data"][0]["attributes"]["body"]

@flow(log_prints=True)
def animal_facts():
    cat_fact = fetch_cat_fact()
    dog_fact = fetch_dog_fact()
    print(f"\n🐱: {cat_fact} \n🐶: {dog_fact}")
```

# Subflows



PREFECT ASSOCIATE  
CERTIFICATION

```
14:16:09.654 | INFO    | prefect.engine - Created flow run 'whimsical-tody' for flow 'animal-facts'
14:16:09.760 | INFO    | Flow run 'whimsical-tody' - Created subflow run 'silky-rattlesnake' for flow 'fetch-cat-fact'
14:16:09.988 | INFO    | Flow run 'silky-rattlesnake' - Finished in state Completed()
14:16:10.029 | INFO    | Flow run 'whimsical-tody' - Created subflow run 'stoic-wolverine' for flow 'fetch-dog-fact'
14:16:10.612 | INFO    | Flow run 'stoic-wolverine' - Finished in state Completed()
14:16:10.613 | INFO    | Flow run 'whimsical-tody' - 🐱: A cat lover is called an Ailurophilia (Greek: cat+lover).
🐱: Wow, check out those choppers! Puppies have 28 teeth and normal adult dogs have 42.
14:16:10.633 | INFO    | Flow run 'whimsical-tody' - Finished in state Completed('All states completed.')
```

# Run over to the UI

---



# Subflows in UI



PREFECT ASSOCIATE  
CERTIFICATION

The screenshot shows the Prefect UI interface. On the left, a sidebar menu lists several options: Flow Runs, Flows (which is selected and highlighted in blue), Work Pools, Blocks, Variables, Notifications, Task Run Concurrency, and Artifacts. The main area is titled "Flows" and displays three completed subflows:

- animal-facts**: Last run: `whimsical-tody` (Completed)
- fetch-cat-fact**: Last run: `silky-rattlesnake` (Completed)
- fetch-dog-fact**: Last run: `stoic-wolverine` (Completed)

Each subflow card also indicates "No deployments" with a question mark icon.

# Timeline view



PREFECT ASSOCIATE  
CERTIFICATION

Flow Runs

Flows

Work Pools

Blocks

Variables

Notifications

Task Run Concurrency

Artifacts

## Flow Runs / whimsical-tody

Completed 2023/05/09 02:16:09 PM 1s None

Flow `animal-facts`

	02:15:30 PM	02:16:00 PM	02:16:30 PM	02:17:00 PM
			 fetch-cat-fact / silky-rattlesnake	
			 fetch-dog-fact / stoic-wolverine	

Logs Task Runs Subflow Runs Results Artifacts Details Parameters

Level: all Oldest to newest

May 9th, 2023

INFO 02:16:09 PM prefect.flow\_runs Created subflow run 'silky-rattlesnake' for flow 'fetch-cat-fact'

INFO 02:16:10 PM prefect.flow\_runs Created subflow run 'stoic-wolverine' for flow 'fetch-dog-fact'

INFO 02:16:10 PM prefect.flow\_runs 🐱: A cat lover is called an Ailurophilia (Greek: cat-lover).

INFO 02:16:10 PM prefect.flow\_runs 🐕: Wow, check out those choppers! Puppies have 28 teeth and normal adult dogs have 42.

INFO 02:16:10 PM prefect.flow\_runs Finished in state Completed('All states completed.')

# Logs in CLI



PREFECT ASSOCIATE  
CERTIFICATION

Feb 6th, 2023

```
INFO    Created subflow run 'fractal-hamster' for flow 'fetch-cat-fact'          10:05:15 PM
INFO    Created subflow run 'xanthic-nautilus' for flow 'fetch-dog-fact'          10:05:15 PM
INFO    🐱: Cats have the largest eyes of any mammal.                            10:05:16 PM
INFO    🐶: A dog's nose is the equivalent of a human's fingerprints, as no two dogs have
       the same nose print.
INFO    Finished in state Completed('All states completed.')                      10:05:16 PM
```

# run\_deployment



PREFECT ASSOCIATE  
CERTIFICATION

## run\_deployment [async]

Create a flow run for a deployment and return it after completion or a timeout.

This function will return when the created flow run enters any terminal state or the timeout is reached. If the timeout is reached and the flow run has not reached a terminal state, it will still be returned. When using a timeout, we suggest checking the state of the flow run if completion is important moving forward.

### Parameters:

Name	Type	Description	Default
<code>name</code>	<code>str</code>	The deployment name in the form: '/'	<code>required</code>
<code>parameters</code>	<code>Optional[dict]</code>	Parameter overrides for this flow run. Merged with the deployment defaults.	<code>None</code>
<code>scheduled_time</code>	<code>Optional[datetime]</code>	The time to schedule the flow run for, defaults to scheduling the flow run to start now.	<code>None</code>
<code>flow_run_name</code>	<code>Optional[str]</code>	A name for the created flow run	<code>None</code>
<code>timeout</code>	<code>Optional[float]</code>	The amount of time to wait for the flow run to complete before returning. Setting <code>timeout</code> to 0 will return the flow run immediately. Setting <code>timeout</code> to <code>None</code> will allow this function to poll indefinitely. Defaults to <code>None</code>	<code>None</code>
<code>poll_interval</code>	<code>Optional[float]</code>	The number of seconds between polls	<code>5</code>

# run\_deployment



PREFECT ASSOCIATE  
CERTIFICATION

```
from prefect.deployments import run_deployment

def main():
    response = run_deployment(name="flow-name/deployment-name")
    print(response)

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



Event-based runs with  
webhooks

# Webhooks

---



PREFECT ASSOCIATE  
CERTIFICATION

- React to events from other systems
- Each exposes a URL endpoint
- Transforms external events into *Prefect events* for use in automations
- CLI creation of webhooks new in 2.10.14

# Webhooks



PREFECT ASSOCIATE  
CERTIFICATION

```
prefect cloud webhook create my-webhook \
```

```
--description some details' \
```

```
--template '{
```

```
    "event": "demo.event",
```

```
    "resource": {"prefect.resource.id":
```

```
        "demo.alert.1"}
```

```
}'
```

# Webhooks

---



PREFECT ASSOCIATE  
CERTIFICATION

- Can use Jinja2 for dynamic templating
- Template should be valid JSON
- UI webhook creation availability
- See more in the docs:

[docs.prefect.io/latest/cloud/webhooks](https://docs.prefect.io/latest/cloud/webhooks)

# Webhooks



PREFECT ASSOCIATE  
CERTIFICATION

```
prefect cloud webhook ls
```

to see existing webhooks and get the url endpoint

webhook id	url slug	name	enabled?	template
af590029-bdf2-4da4-9204-4c6076a8c75d	gVus6dtdkrfY7Xq-r2iK5g	first-webhook	True	{ "event": "demo.event", "resource": {"prefect.resource.id": "demo.alert.1"} }

# Webhooks

---



PREFECT ASSOCIATE  
CERTIFICATION

Hit the endpoint:

```
curl https://api.prefect.cloud/hooks/your_slug_here
```

# Webhooks



PREFECT ASSOCIATE  
CERTIFICATION

See the event in the Prefect Cloud workspace event feed

10:24:54 PM  
Jun 19th, 2023



## Demo event

demo.event

## Resource

demo.alert.2

## Related Resources

prefect-cloud.webhook.791b2034-892f-41eb-81a3-dc9dfbf133c

# Webhooks

---



PREFECT ASSOCIATE  
CERTIFICATION

⚡ Use this event as a trigger in an automation! ⚡

# Webhooks

---



PREFECT ASSOCIATE  
CERTIFICATION

Potential use case:

When a file arrives in cloud storage, trigger a lambda function to hit a Prefect webhook.

The webhook then triggers a deployment run.



PREFECT



A decorative graphic at the bottom of the slide features three white circles with blue outlines, each containing a smaller white circle. They are arranged in a triangular pattern: one on the left, one on the right, and one centered below them, all connected by thin white lines.

prefect.yaml

# *prefect.yaml*

```
# Generic metadata about this project
name: pacc-2023
prefect-version: 2.10.18

# build section allows you to manage and build docker images
build: null

# push section allows you to manage if and how this project is
push: null

# pull section allows you to provide instructions for cloning
pull:
- prefect.deployments.steps.git_clone:
    repository: https://github.com/dscdver/pacc-2023.git
    branch: main
```



## Configuration for creating deployments

- ***name***: from directory where called (*pacc-2023*)
- ***prefect-version***: from active environment (*2.10.18*)
- ***pull*** step (repository & branch): from git repo



- ***deployments:***

Config for one or more deployments

Required keys:

- *name*
- *entrypoint*
- *work\_pool -> name*

```
deployments:  
  
  - name: deployment1  
    entrypoint: 202/flows.py:pipe  
    work_pool:  
      name: local-work  
  
  - name: deployment2  
    entrypoint: 202/flows2.py:pipe2  
    work_pool:  
      name: local-work
```



# Can override steps above on per-deployment basis

```
deployments:
  - name: prod-deployment
    entrypoint: 202/flows.py:pipe
    work_pool:
      name: prod-pool
    schedule:
      interval: 600
    pull:
      - prefect.deployments.steps.git_clone:
          repository: https://github.com/dsdiscover/pacc-london-2023.git
          branch: prod
          access_token: "{{prefect.blocks.secret.gh-secret}}"

  - name: staging-deployment
    entrypoint: 202/flows.py:pipe
    work_pool:
      name: staging-pool
    pull:
      - prefect.deployments.steps.git_clone:
          repository: https://github.com/dsdiscover/pacc-london-2023.git
          branch: staging
```

# Redeploy a deployment

Requires a `prefect.yaml` file

```
prefect deploy
```

? Would you like to use an existing deployment configuration? [Use arrows to move; enter to select; n to select none]

	Name	Entrypoint	Description
>	<code>first_deploy</code>	<code>104/flows.py:pipe</code>	

# Deploy multiple deployments at once

---



Deploy all deployments in a *prefect.yaml* file:

```
prefect deploy --all
```



PREFECT

# Scheduling

# Scheduling

Like other ducks, Minerva likes things to happen on time.



# Scheduling

---



PREFECT ASSOCIATE  
CERTIFICATION

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

# Schedule types

---

- Interval
- Cron
- RRule



# Add a schedule



PREFECT ASSOCIATE  
CERTIFICATION

## *prefect deploy command - interactive*

```
? Would you like to schedule when this flow runs? [y/n] (y): y
? What type of schedule would you like to use? [Use arrows to move;
enter to select]
```

	Schedule Type	Description
>	Interval	Allows you to set flow runs to be executed at fixed time intervals.
	Cron	Allows you to define recurring flow runs based on a specified pattern using cron syntax.
	RRule	Allows you to define recurring flow runs using RFC 2445 recurrence rules.

```
? Seconds between scheduled runs (3600): 100
```

# In the UI



PREFECT ASSOCIATE  
CERTIFICATION

**Deployments / cat-fact-fetch**

Runs Parameters Infra Overrides Description

0 Flow runs All run states Newest to oldest

No runs from the last 180 days

Flow  
pipe

Work Pool  
default-agent-pool

Work Queue  
default

Infrastructure  
anonymous-250496b3-c7db-419a-9f67-f5066980eba4

Schedule

Add



# In the UI



PREFECT ASSOCIATE  
CERTIFICATION

Add schedule

Schedule type

Interval    Cron    RRule

Value                          Interval

60                              Minutes ▾

Start date                      Timezone

04/05/2023                     ▾

Cancel                         Save

RRule cheat sheet: [jakubbroztocil.github.io/rrule/](https://jakubbroztocil.github.io/rrule/)

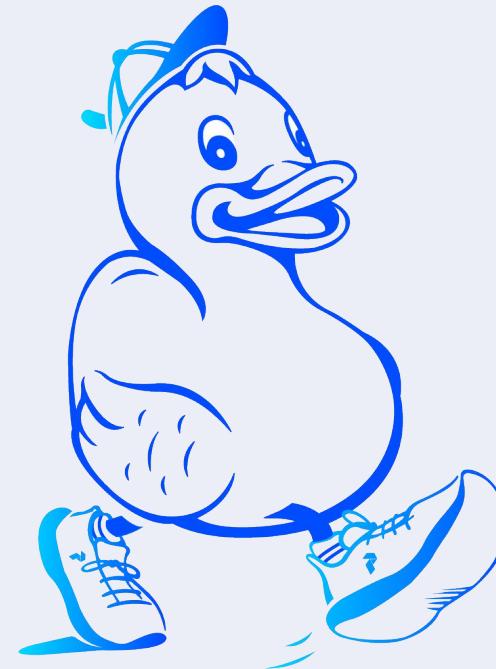
Or ask Marvin (another Prefect package) `pip install marvin`

```
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 restarting schedules



# Pause/resume schedule from UI



PREFECT ASSOCIATE  
CERTIFICATION

A screenshot of the Prefect UI interface. At the top left is a blue button labeled "Deployments". To its right is a dark grey card containing the text "fetch-weather/Python deploy file with params" next to a small square icon. To the right of this card is the name "jeffgmail". Further to the right is a blue toggle switch with a white circle, which is currently in the "on" position. A red arrow points from the text "Pause/resume schedule from UI" at the top to this toggle switch. At the far right end of the card is a vertical ellipsis (...).

# Pause/resume schedule from CLI

---



Pause

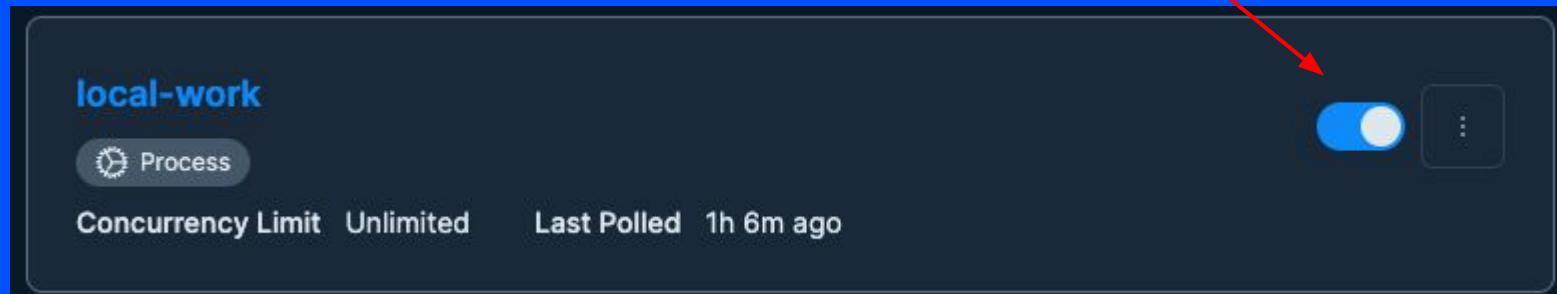
```
prefect deployment pause-schedule  
pipe_flow/"RRule Scheduled Deployment"
```

Resume

```
prefect deployment resume-schedule  
pipe_flow/"RRule Scheduled Deployment"
```

# Pause work pools

You can also pause work pools



# 105 Recap

---



PREFECT ASSOCIATE  
CERTIFICATION

You've seen how to:

- use several workflow patterns
- modify *prefect.yaml*
- create schedules from the UI & CLI
- pause and resume schedules 



PREFECT

# Lab 105

# 105 Lab

---



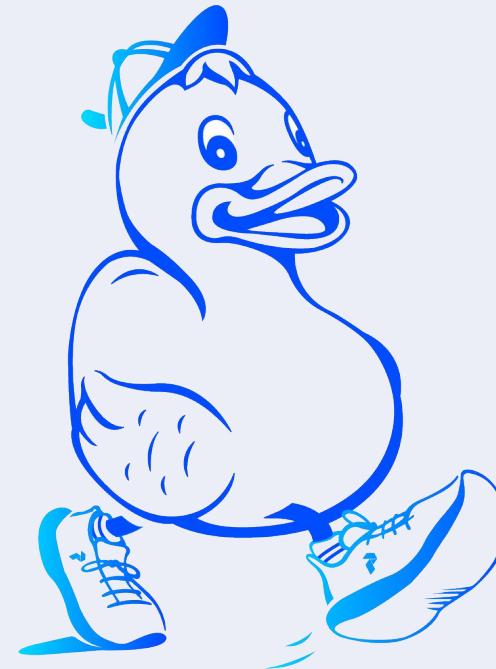
- Create a deployment that uses one of the new workflow patterns you've seen in this module.
- See the *prefect.yaml*. Use GitHub or other git-repo-based code storage.
  - Don't forget to push your code!
- Add a schedule.
- Pause the schedule.

MODULE

# 201 - Integrations & Docker

# Integrations

Ducks integrate well.



# 201 Agenda

---

- Integrations
- Docker
- Work pools with Docker
- Building Docker-based deployments

# Integrations



PREFECT ASSOCIATE  
CERTIFICATION

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

## Prefect Collections Catalog

Below you can find a list of all available Prefect Collections.

[prefect-airbyte](#)



Maintained by Prefect

[prefect-aws](#)



Maintained by Prefect

[prefect-azure](#)



Maintained by Prefect

[prefect-cubejs](#)



Maintained by Alessandro Lollo

[prefect-dask](#)



Maintained by Prefect

[prefect-databricks](#)



Maintained by Prefect

[prefect-dbt](#)



Maintained by Prefect

[prefect-email](#)



Maintained by Prefect

# Integrations

---



PREFECT ASSOCIATE  
CERTIFICATION

Python packages that add convenience

- 40+ integrations
- Template to create your own
- Can contribute to the community

# Integrations

---



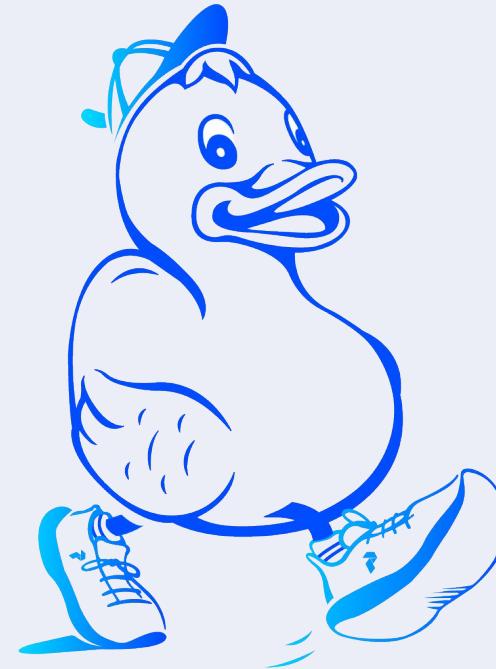
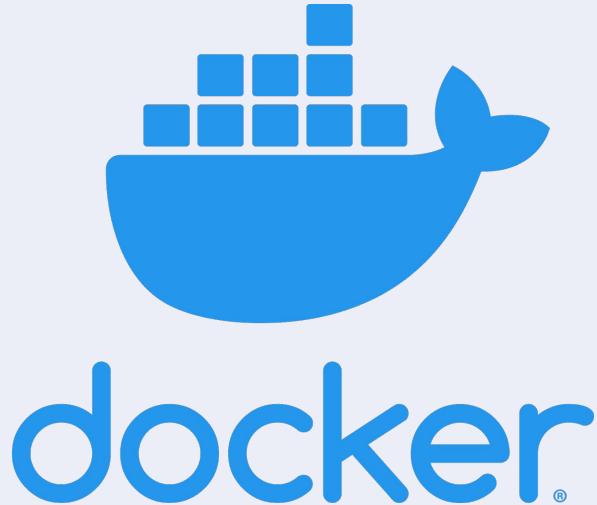
PREFECT ASSOCIATE  
CERTIFICATION

Prefect is Python-based and designed for flexibility

Use with most any Python library - no special  
integration required 

# Docker

Little known fact: ducks are all about containerization with Docker.



# Why use Docker?

---



PREFECT ASSOCIATE  
CERTIFICATION

- Same operating environment everywhere
- Lighter weight than a VM
- Linux (generally)
- Portable
- Very popular

# Docker

---



PREFECT ASSOCIATE  
CERTIFICATION

- Prefect provides base Docker images
- Can customize base image
- Read about choosing images at  
[docs.prefect.io/latest/concepts/infrastructure/#standard-python](https://docs.prefect.io/latest/concepts/infrastructure/#standard-python)

## Prerequisites

- Docker installed & **running**
- `prefect-docker` package installed



# Work pools - Docker

# Work pools

---



PREFECT ASSOCIATE  
CERTIFICATION

Worker kicks off flow runs in the infrastructure type specified by work pool

# Work pools



PREFECT ASSOCIATE  
CERTIFICATION

Work Pools / Create

01 Basic Information 02 Infrastructure Type 03 Configuration

Select the infrastructure you want to use to execute your flow runs

- Prefect Agent**  
Execute flow runs on heterogenous infrastructure using infrastructure blocks.
- AWS Elastic Container Service** Beta  
Execute flow runs within containers on AWS ECS. Works with existing ECS clusters and serverless execution via AWS Fargate. Requires an AWS account.
- Azure Container Instances** Beta  
Execute flow runs within containers on Azure's Container Instances service. Requires an Azure account.
- Docker** Beta  
Execute flow runs within Docker containers. Works well if you manage flow execution environments via Docker images. Requires access to a running Docker daemon.

# Work pools

Can specify a particular image your team created

The screenshot shows the 'Base Job Template' configuration page in Prefect, which is currently in Beta. The interface is dark-themed with light-colored text and highlights. It includes tabs for 'Defaults' and 'Advanced'. A note at the top states: 'The fields below control the default values for the base job template. These values can be overridden by deployments.' Below this, there are several optional configuration sections:

- Command (Optional)**: A text input field for the command to use when starting a flow run.
- Environment Variables (Optional)**: A text input field for environment variables to set when starting a flow run, containing entries 1, 2, and 3.
- Labels (Optional)**: A text input field for labels applied to infrastructure created by the worker, containing entries 1, 2, and 3.
- Name (Optional)**: A text input field for the name given to infrastructure created by the worker, which is currently empty.
- Image (Optional)**: A text input field for the image reference of a container image to use for created jobs, with the value 'docker.io/prefecthq/prefect:2-latest' entered.
- Image Pull Policy (Optional)**: A text input field for the image pull policy, which is currently empty.

# Work pools



PREFECT ASSOCIATE  
CERTIFICATION

Base Job Template (Beta)

Defaults Advanced

This is the JSON representation of the base job template. A work pool's job template controls infrastructure configuration for all flow runs in the work pool, and specifies the configuration that can be overridden by deployments.

ⓘ For more information on the structure of a work pool's base job template, check out [the docs](#).

```
{  
    "job_configuration": {  
        "command": "{{ command }}",  
        "env": "{{ env }}",  
        "labels": "{{ labels }}",  
        "name": "{{ name }}",  
        "image": "{{ image }}",  
        "image_pull_policy": "{{ image_pull_policy }}",  
        "networks": "{{ networks }}",  
        "network_mode": "{{ network_mode }}",  
        "auto_remove": "{{ auto_remove }}",  
        "volumes": "{{ volumes }}",  
        "stream_output": "{{ stream_output }}",  
        "mem_limit": "{{ mem_limit }}",  
        "memswap_limit": "{{ memswap_limit }}",  
        "privileged": "{{ privileged }}"  
    },  
    "variables": {  
        "description": "Configuration class used by the Docker worker.\n\nAn instance of this class is passed to the Docker worker's `run`  
    }  
}
```



# Prefect Docker Deployments

# *prefect deploy*



PREFECT ASSOCIATE  
CERTIFICATION

If choose *docker* typed work pool you will be asked docker-related questions

	Work Pool Name	Infrastructure Type	Description
>	<b>docker-pool</b> my-pool	docker process	

```
? Would you like to build a custom Docker image for this deployment? [y/n]
(n) : 
```

# Method 1: *prefect deploy*

---

Use the defaults for the work pool

OR

Build a custom Docker image with flow code

- Push image to a Docker registry
  - Use existing Dockerfile
  - Auto-includes packages in *requirements.txt*

Follow the prompts. 😊



# Resulting prefect.yaml

```
- name: dock-interact
  version:
  tags: []
  description:
  entrypoint: 104/flows.py:pipe
  parameters: {}
  work_pool:
    name: docker-pool
    work_queue_name:
    job_variables:
      image: '{{ build-image.image }}'
  schedule:
  build:
    - prefect_docker.deployments.steps.build_docker_image:
        requires: prefect-docker>=0.3.1
        id: build-image
        dockerfile: auto
        image_name: discdiver/dock-interact
        tag: 0.0.1
```

## Development environment

Where you write your flows

- 1 Create a project in a new directory

```
$ mkdir my-project  
$ cd my-project  
$ prefect init
```

my-project

```
prefect.yaml  
prefect-version: 2.10  
name: my-project  
  
build:  
  - step  
push:  
  - step  
pull:  
  - step
```

- 2 Add an @flow decorator to your code's entrypoint function, give it a name, and save it to the new directory

```
flow.py  
from prefect import flow  
  
@flow(name="Hello")  
def say_hello():  
    print("Hello, World!")  
  
if __name__ == "__main__":  
    say_hello()
```

Creates

Adds

- 5 Deploy your flow

```
$ prefect deploy my-flow.py:say_hello
```

Creates

- 6 Schedule a run of the deployed flow from the CLI...

```
$ prefect deployment run Hello/my-deployment
```

## Orchestration environment

Where Prefect Cloud or server runs

- 3 Log in to Prefect Cloud

```
$ prefect cloud login
```

- OR Start an open source Prefect server

```
$ prefect server start
```



OR

...from the UI



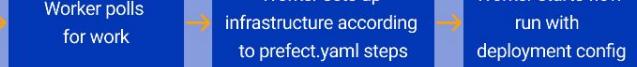
## Execution environment

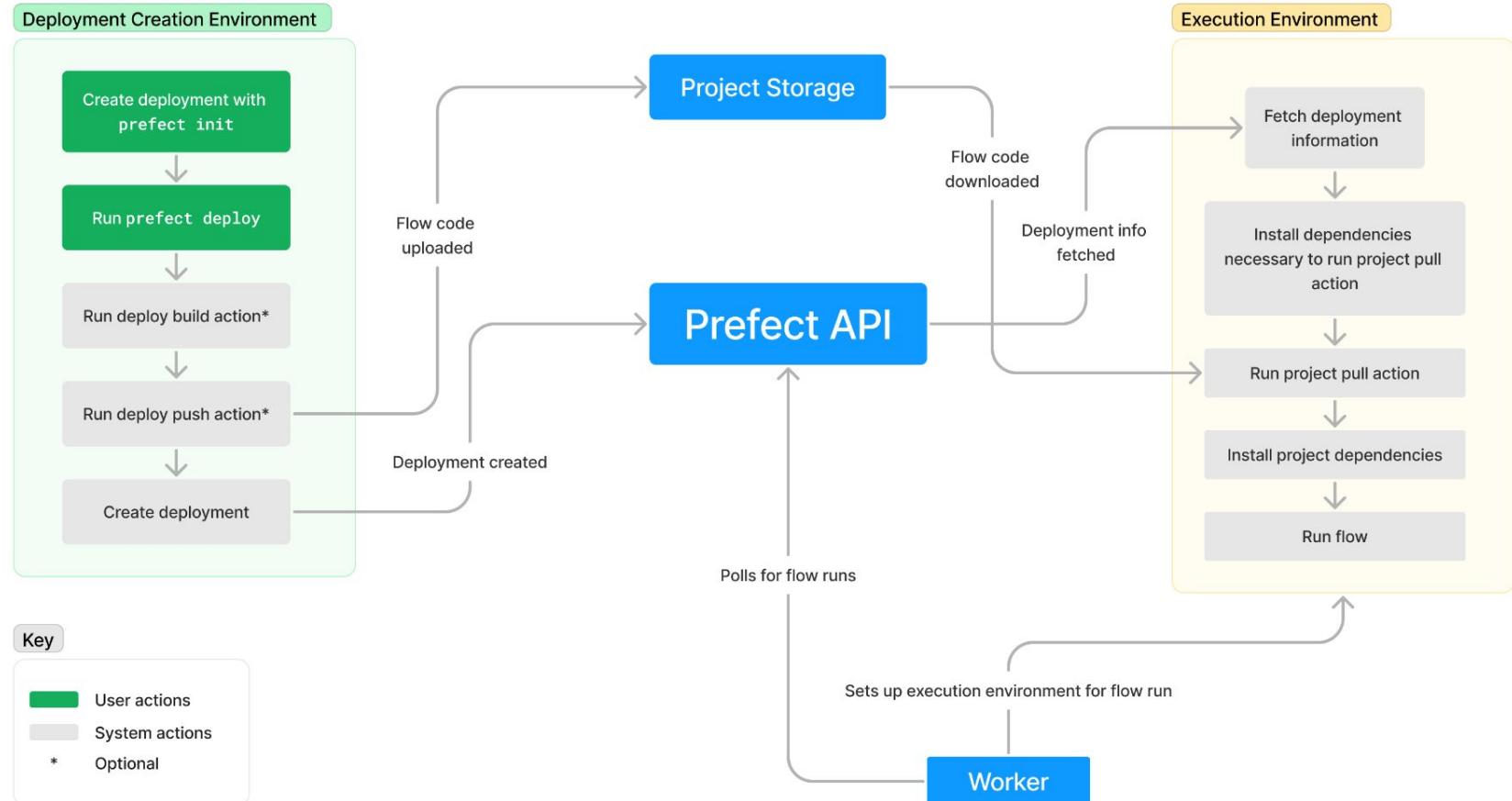
Where your flows run

- 4 Start a worker for a corresponding work pool

```
$ prefect worker start -p my-pool -t process
```

4





# Docker - worker

---



Start a Docker type worker to auto-create and connect to the work pool named *docker-work*

```
prefect worker start -t docker -p docker-work
```

# Dockerfile



PREFECT ASSOCIATE  
CERTIFICATION

```
FROM prefecthq/prefect
COPY requirements.txt /opt/prefect/201/requirements.txt
RUN python -m pip install -r requirements.txt
COPY . /opt/prefect/pacc-2023/
WORKDIR /opt/prefect/pacc-2023/
```

# Flow run

---



Docker configuration in the work pool can be overridden in deployments

(Can use *deployments* section of `prefect.yaml`)

# Docker

---



PREFECT ASSOCIATE  
CERTIFICATION

- Run deployment
- Worker spins up Docker container
- Flow runs in Docker container 

# Infrastructure - Docker Container



## Check out Docker Desktop

The screenshot shows the Docker Desktop interface. At the top, it says "Containers" and has a "Give feedback" link. Below that, a descriptive text states: "A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another." with a "Learn more" link. There are filters for "Only show running containers" (unchecked) and a search bar. The main table lists a single container:

	NAME	IMAGE	STATUS	PORT(S)	STARTED	ACTIONS
<input type="checkbox"/>	nano-tortoise 26fed9f8e268	prefecthq/r	Exited			

# 201 Recap

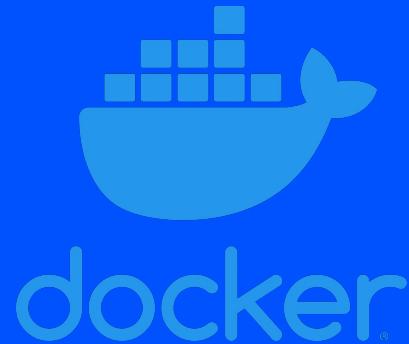
---



PREFECT ASSOCIATE  
CERTIFICATION

You've seen how to use Docker with deployments, workers & work pools!

Docker containers are the basis for many production infrastructure setups: e.g. K8s, ECS.





PREFECT

# Lab 201

As time allows/on your own:

- Make a Docker work pool
- Start a Docker type worker that polls the pool
- Run the deployment
- Stretch 1: push the image to Docker Hub
- Stretch 2: use Kubernetes



PREFECT

# Wrap

# Brief feedback survey

---

Please let us know what went well  
and what could be improved.





# Congratulations!!!



**PREFECT ASSOCIATE**  
CERTIFICATION



PREFECT

# Bonus Content



PREFECT

# Variables

# Prefect variables

---



- String values evaluated at runtime
- Store and reuse non-sensitive, small data
- Create via UI or CLI

# Prefect variables



PREFECT ASSOCIATE  
CERTIFICATION

Only string values

### New variable

**Name**

**Value**

**Tags**

**Create**

# Prefect variables



PREFECT ASSOCIATE  
CERTIFICATION

Flow Runs

Flows

Deployments

Work Pools

Blocks

**Variables**

Notifications

Task Run Concurrency

## Variables +

3 Variables

Search variables

A to Z

Filter by tags

<input type="checkbox"/>	Name	Value	Updated	Tags
<input type="checkbox"/>	age	twenty-two	2023/04/13 03:36:53 PM	<span>:</span>
<input type="checkbox"/>	height	72	2023/04/13 04:00:32 PM	<span>:</span>
<input type="checkbox"/>	url	abc123.com	2023/04/13 04:01:15 PM	<span>:</span>



PREFECT

Write data to GCS

<https://prefecthq.github.io/prefect-gcp>

GCP services via blocks

BigQuery, AIPlatform, GCS & more



⚠️ *GcsBucket* block ≠ *GCS* block

Install with dependencies you want (e.g.):

```
pip install "prefect-gcp[cloud_storage]"
```

Register blocks with server:

```
prefect block register -m prefect_gcp
```

# New block types then show in UI



PREFECT ASSOCIATE  
CERTIFICATION



Google Cloud

### GCP Credentials

Block used to manage authentication with GCP. GCP authentication is handled via the `google.oauth2` module or through the CLI. Specify either one of...

Add +



Google Cloud

### GcpSecret

Manages a secret in Google Cloud Platform's Secret Manager.

Add +



Google Cloud

### GCS

Store data as a file on Google Cloud Storage.

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

Add +



Google Cloud

### GCS Bucket

Block used to store data using GCP Cloud Storage Buckets.

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

Add +



### GitHub

Interact with files stored on public GitHub repositories.

get-directory

Add +



### GitLab Credentials

Store a GitLab personal access token to interact with private GitLab repositories.

Add +

# Create a GCS Bucket in GCP



PREFECT ASSOCIATE  
CERTIFICATION

The screenshot shows the 'Create a bucket' wizard in the Google Cloud Storage interface. The left sidebar lists 'Buckets', 'Monitoring (NEW)', and 'Settings'. The main panel has a title 'Create a bucket' and a sub-section 'Name your bucket' with a text input field containing 'pacc-gcs'. It includes a tip about not including sensitive information and a 'CONTINUE' button. Below this are sections for 'Choose where to store your data' (Location: us, Multi-region), 'Choose a storage class for your data' (Default storage class: Standard), 'Choose how to control access to objects' (Public access prevention: On, Access control: Uniform), and 'Choose how to protect object data' (Protection tools: None, Data encryption: Google-managed key). At the bottom are 'CREATE' and 'CANCEL' buttons.



## Create a GCS Bucket block



## Let's write data to a GCS bucket



## Add GCP Credentials Block

**Block Name**

**Bucket**  
Name of the bucket.

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

**Gcp Credentials**  
The credentials to authenticate with GCP.

**GcpCredentials (Optional)**  
Block used to manage authentication with GCP. GCP authentication is handled via the `google.oauth2` module or through the CLI. Specify either one of service `account\_file` or `service\_account\_info`; if both are not specified, the client will try to detect the service account info stored in the env from the command, `gcloud auth application-default login`. Refer to the [Authentication docs] (<https://cloud.google.com/docs/authentication/production>) for more info about the possible credential configurations.

None

Add +

**GCS Bucket**  
Block used to store data using GCP Cloud Storage Buckets.

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



## Paste service account credential file contents

**Blocks / Choose a Block / GCP Credentials / Create**

**Block Name**

**Project (Optional)**  
The GCP project to use for the client.

**Service Account File (Optional)**  
Path to the service account JSON keyfile.

**Service Account Info (Optional)**  
The contents of the keyfile as a dict.  
 Format

**GCP Credentials**  
Block used to manage authentication with GCP. GCP authentication is handled via the `google.oauth2` module or through the CLI. Specify either...

**Cancel** **Create**





Create flow that loads GCSBucket block and uploads file

```
from pathlib import Path
from prefect import flow, task
from prefect_gcp.cloud_storage import GcsBucket

@flow()
def upload_to_gcs(color: str, year: int, month: int) -> None:
    """The main flow function to upload taxi data"""
    path = Path(f"data/{color}/{year}/{color}_tripdata_{year}-{month:02}.parquet")
    gcs_block = GcsBucket.load("pacc-gcs-bucket")
    gcs_block.upload_from_path(from_path=path, to_path=path)
```



Create deployment for the flow (or just test by running the flow function script)



## Logs look good

Feb 8th, 2023

INFO

Downloading flow code from storage at '/Users/jeffhale/Desktop/prefect/demos/pacc/pacc-demos-feb-6-2023' 10:19:13 PM

INFO

Getting bucket 'pacc-gcs'. 10:19:14 PM

INFO

Uploading from PosixPath('data/green/2020/green\_tripdata\_2020-01.parquet') to the bucket 'pacc-gcs' path 'data/green/2020/green\_tripdata\_2020-01.parquet'. 10:19:14 PM



## See results in GCS

### pacc-gcs

Location	Storage class	Public access	Protection
us (multiple regions in United States)	Standard	Not public	None

OBJECTS    CONFIGURATION    PERMISSIONS    PROTECTION    LIFECYCLE    OBSERVABILITY

Buckets > pacc-gcs > data > green > 2020

UPLOAD FILES    UPLOAD FOLDER    CREATE FOLDER    TRANSFER DATA ▾    MANAGE HOLDS    DOWNLOAD    DELETE

Filter by name prefix only ▾

Filter Filter objects and folders

<input type="checkbox"/>	Name	Size	Type	Created	Storage class
<input type="checkbox"/>	<a href="#">green_tripdata_2020-01.parquet</a>	6.9 MB	application/octet-stream	Feb 8, 2023, 10:12:07 PM	Standard



# Parallel execution

# Parallelism

---

- Two or more operations happening at the same time on one or more machines
- Helpful when operations limited by CPU
- Many machine learning algorithms parallelizable

# Task Runners for parallelism

---

- *DaskTaskRunner*
- *RayTaskRunner*

Both require *prefect-dask* or *prefect-ray* packages

# DaskTaskRunner



PREFECT ASSOCIATE  
CERTIFICATION

```
● ● ●

from prefect import flow, task
from prefect_dask.task_runners import DaskTaskRunner

@task
def say_hello(name):
    print(f"hello {name}")

@task
def say_goodbye(name):
    print(f"goodbye {name}")

@flow(task_runner=DaskTaskRunner())
def greetings(names):
    for name in names:
        say_hello.submit(name)
        say_goodbye.submit(name)

if __name__ == "__main__":
    greetings(["arthur", "trillian", "ford", "marvin"])
```

# DaskTaskRunner for parallelism

---



- Can see the Dask UI if have *bokeh* package installed
- UI will be linked in the terminal at run time

# Advanced topics

---



- Async task runners for concurrency and parallelization
- Testing
- CI/CD with GitHub Actions
- PrefectClient for REST API



PREFECT

# Async

# Concurrency

---



PREFECT ASSOCIATE  
CERTIFICATION

- Helpful when waiting for external systems to respond
- Allows other work to be done while waiting
- Prefect's *ConcurrentTaskRunner* replaces need for using Python's *async*, *await*, etc.

# Concurrency



PREFECT ASSOCIATE  
CERTIFICATION

```
from prefect import flow, task
from prefect.task_runners import ConcurrentTaskRunner

@task
def stop_at_floor(floor):
    print(f"elevator moving to floor {floor}")
    print(f"elevator stops on floor {floor}")

@flow(task_runner=ConcurrentTaskRunner())
def elevator():
    for floor in range(3, 0, -1):
        stop_at_floor.submit(floor)

elevator()
```

# Concurrency

---



PREFECT ASSOCIATE  
CERTIFICATION

```
elevator moving to floor 3
elevator stops on floor 3
elevator moving to floor 1
elevator stops on floor 1
elevator moving to floor 2
elevator stops on floor 2
```

# Task Runners

---



PREFECT ASSOCIATE  
CERTIFICATION

- Specify in flow decorator
- ConcurrentTaskRunner is ready by default
- Use `.submit()` when call a task to return a *Prefect Future* instead of direct result



PREFECT

# Testing

# Testing



PREFECT ASSOCIATE  
CERTIFICATION

- Context manager for unit tests provided
- Run flows against temporary local SQLite database

```
from prefect import flow
from prefect.testing.utilities import prefect_test_harness

@flow
def my_favorite_flow():
    return 42

def test_my_favorite_flow():
    """basic test running the flow against a temporary testing database"""
    with prefect_test_harness():
        assert my_favorite_flow() == 42
```



- Use in a Pytest fixture

```
from prefect import flow
import pytest
from prefect.testing.utilities import prefect_test_harness

@pytest.fixture(autouse=True, scope="session")
def prefect_test_fixture():
    with prefect_test_harness():
        yield
```



# CI/CD with GitHub Actions

# GitHub Actions with deployments

---



- CI/CD - when you push code or make a PR automatically take an action
- Pre-built Github Action to deploy a Prefect deployment
- [github.com/marketplace/actions/deploy-a-prefect-flow](https://github.com/marketplace/actions/deploy-a-prefect-flow)

# GitHub Action



PREFECT ASSOCIATE  
CERTIFICATION

```
name: Deploy a Prefect flow
on:
  push:
    branches:
      - main
jobs:
  deploy_flow:
    runs-on: ubuntu-latest
    steps:
      - uses: checkout@v3

      - uses: actions/setup-python@v4
        with:
          python-version: '3.10'

      - name: Run Prefect Deploy
        uses: PrefectHQ/actions-prefect-deploy@v1
        with:
          prefect-api-key: ${{ secrets.PREFECT_API_KEY }}
          prefect-workspace: ${{ secrets.PREFECT_WORKSPACE }}
          requirements-file-path: ./examples/simple/requirements.txt
          entrypoint: ./examples/simple/flow.py:call_api
          additional-args: --cron '30 19 * * 0'
```



# Prefect REST API

# PrefectClient to interact with the REST API



PREFECT ASSOCIATE  
CERTIFICATION

```
import asyncio
from prefect.client import get_client

async def get_flows():
    client = get_client()
    r = await client.read_flows(limit=5)
    return r

r = asyncio.run(get_flows())

for flow in r:
    print(flow.name, flow.id)

if __name__ == "__main__":
    asyncio.run(get_flows())
```

Or use Requests/curl/etc.

---



PREFECT ASSOCIATE  
CERTIFICATION

Cloud and server REST API interactive docs:

[docs.prefect.io/latest/api-ref/rest-api](https://docs.prefect.io/latest/api-ref/rest-api)

# Work Queues



PREFECT ASSOCIATE  
CERTIFICATION

Runs

Work Queues

Workers

1 Work Queue



Search

Name

Concurrency Limit

Priority

Status

default

1

Healthy



# Work Queues: Features

---



- Concurrency limits
- Health
- Priority

# 203 Recap

---



PREFECT ASSOCIATE  
CERTIFICATION

You've learned about:

- Concurrency with ConcurrentTaskRunner
- CI/CD with GitHub Actions
- Prefect Runtime
- REST API

MODULE

# 202 - Moving Data with S3

# 202 Agenda

---

- Upload data to a cloud provider



# Upload data to AWS S3

# Steps

---



PREFECT ASSOCIATE  
CERTIFICATION

1. Install prefect-aws
2. Register new blocks
3. Create S3 bucket
4. Create S3Bucket block from UI or CLI
5. Use in a flow

# Install prefect-aws

---



PREFECT ASSOCIATE  
CERTIFICATION

```
pip install -U prefect-aws
```

# Register new blocks

---



PREFECT ASSOCIATE  
CERTIFICATION

```
prefect blocks register -m prefect_aws
```

```
Successfully registered 5 blocks

Registered Blocks
AWS Credentials
AWS Secret
ECS Task
MinIO Credentials
S3 Bucket
```

# See block types & blocks from CLI

---



```
prefect block type ls
```

```
prefect block ls
```



## ⚠️ **S3Bucket block from prefect-aws != S3 block that ships with Prefect**

- Both block types upload and download data
- S3Bucket block has many methods
- We are showing how to use S3Bucket block

# Create S3 Bucket



PREFECT ASSOCIATE  
CERTIFICATION

Amazon S3 > Buckets > Create bucket

## Create bucket Info

Buckets are containers for data stored in S3. [Learn more](#)

### General configuration

#### Bucket name

myawsbucket

Bucket name must be globally unique and must not contain spaces or uppercase letters. [See rules for bucket naming](#)

#### AWS Region

US East (N. Virginia) us-east-1



#### Copy settings from existing bucket - *optional*

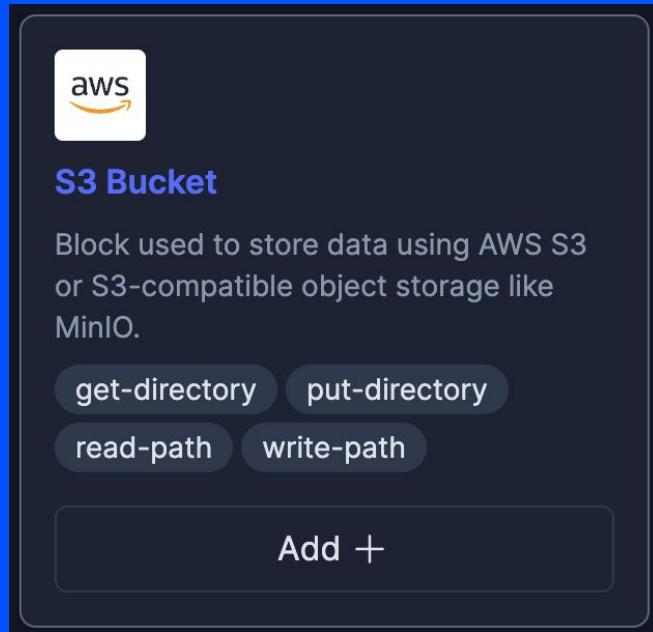
Only the bucket settings in the following configuration are copied.

[Choose bucket](#)

# Create S3Bucket block from UI



PREFECT ASSOCIATE  
CERTIFICATION





# Create S3Bucket block from UI

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

**Block Name**

**Bucket Name**  
Name of your bucket.

**Credentials**  
[AwsCredentials](#) [MinIOCredentials](#)

A block containing your credentials to AWS or MinIO.

**AwsCredentials (Optional)**  
Block used to manage authentication with AWS. AWS authentication is handled via the `boto3` module. Refer to the [boto3 docs](#) for more info about the possible credential configurations.

[Add +](#)

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

[Cancel](#) [Create](#)

**S3 Bucket**  
Block used to store data using AWS S3 or S3-compatible object storage like MinIO.  
[get-directory](#) [put-directory](#) [read-path](#) [write-path](#)

# AWS Credentials block from UI



PREFECT ASSOCIATE  
CERTIFICATION

Use the nested AWS Credentials block as needed

**Blocks / Choose a Block / AWS Credentials / Create**

Block Name

Region Name (Optional)  
The AWS Region where you want to create new connections.

Profile Name (Optional)  
The profile to use when creating your session.

AWS Access Key ID (Optional)  
A specific AWS access key ID.



**AWS Credentials**  
Block used to manage authentication with AWS. AWS authentication is handled via the `boto3` module. Refer to the [boto3 docs]...

# AWS Credentials block from UI



Leave most fields blank.

Probably use AWS Access Key *ID* & AWS Access Key *Secret*.

**AWS Access Key Secret (Optional)**

A specific AWS secret access key.

Cancel

Create



# Or create blocks with Python code

```
from time import sleep
from prefect_aws import S3Bucket, AwsCredentials

def create_aws_creds_block():
    # environment variables can be helpful for creating credentials blocks
    # do not store credential values in public locations (e.g. GitHub public repo)
    my_aws_creds_obj = AwsCredentials(
        aws_access_key_id="123abc",
        aws_secret_access_key="ab123",
    )
    my_aws_creds_obj.save(name="my-aws-creds-block", overwrite=True)

def create_s3_bucket_block():
    aws_creds = AwsCredentials.load("my-aws-creds-block")
    my_s3_bucket_obj = S3Bucket(
        bucket_name="my-first-bucket-abc", credentials=aws_creds
    )
    my_s3_bucket_obj.save(name="s3-bucket-block", overwrite=True)

if __name__ == "__main__":
    create_aws_creds_block()
    sleep(5) # ensure server has time to create credentials block before loading
    create_s3_bucket_block()
```

# View block in the UI



PREFECT ASSOCIATE  
CERTIFICATION

Blocks / my-aws-creds-block

May 3rd, 2023 12:00 AM May 3rd, 2023 11:59 PM

Block document my-aws-creds-block 2 events

Paste this snippet into your flows to use this block. Need help? [View Docs](#)

```
from prefect_aws import AwsCredentials
aws_credentials_block = AwsCredentials.load("my-aws-creds-block")
```

**Region Name**  
None

**Profile Name**  
None

**AWS Access Key ID**  
123abc

**AWS Session Token**  
None

**AWS Client Parameters**  
{ "config": null, "verify": true, "use\_ssl": true, "api\_version": "", "endpoint\_url": "", "verify\_cert\_path": "" }

**AWS Access Key Secret**  
\*\*\*\*\*

**AWS Credentials**  
Block used to manage authentication with AWS. AWS authentication is handled via the 'boto3' module. Refer to the [boto3 docs]...

# Flow code loads S3 block and uploads data file



PREFECT ASSOCIATE  
CERTIFICATION

```
from pathlib import Path
from prefect import flow
from prefect_aws.s3 import S3Bucket

@flow()
def upload_to_s3(color: str, year: int, month: int) -> None:
    """The main flow function to upload taxi data"""
    path = Path(f"data/{color}/{year}/{color}_tripdata_{year}-{month:02}.parquet")
    s3_block = S3Bucket.load("s3-bucket-block")
    s3_block.upload_from_path(from_path=path, to_path=path)

if __name__ == "__main__":
    upload_to_s3(color="green", year=2020, month=1)
```

# Use your flow code!

---



PREFECT ASSOCIATE  
CERTIFICATION

- Can test with `python my_script.py`
- Then create a deployment and run it! 

# See file in S3 bucket



PREFECT ASSOCIATE  
CERTIFICATION

Amazon S3 > Buckets > prefect-aws-demos > data/ > green/ > 2020/

2020/

Copy S3 URI

Objects

Properties

## Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Copy S3 URI   Copy URL   Download   Open   Delete   Actions ▾   Create folder   Upload

Find objects by prefix

< 1 >

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	green_tripdata_2020-01.parquet	parquet	May 3, 2023, 17:31:04 (UTC-04:00)	6.9 MB	Standard

# 202 Recap

---



PREFECT ASSOCIATE  
CERTIFICATION

You've seen how to upload data to a cloud provider



PREFECT

# Lab 202

# 202 Lab

---



- Create blocks and flow code that uploads and downloads data from AWS (or do the equivalent with GCP or Azure).



PREFECT

# Prefect Runtime

*prefect.runtime* module: home for runtime context access  
submodule for major concepts:

***deployment***: Access information about the deployment  
for the current run

***flow\_run***: Access information about the current flow run

***task\_run***: Access information about the current task run

# Prefect Runtime



PREFECT ASSOCIATE  
CERTIFICATION

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

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

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

my_flow(1)
```



PREFECT

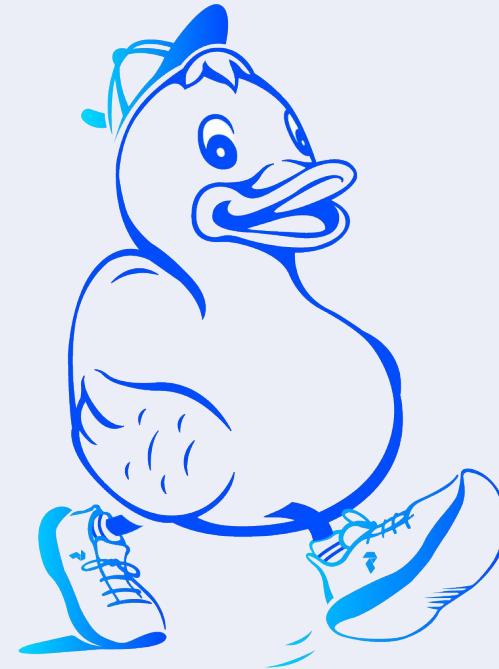
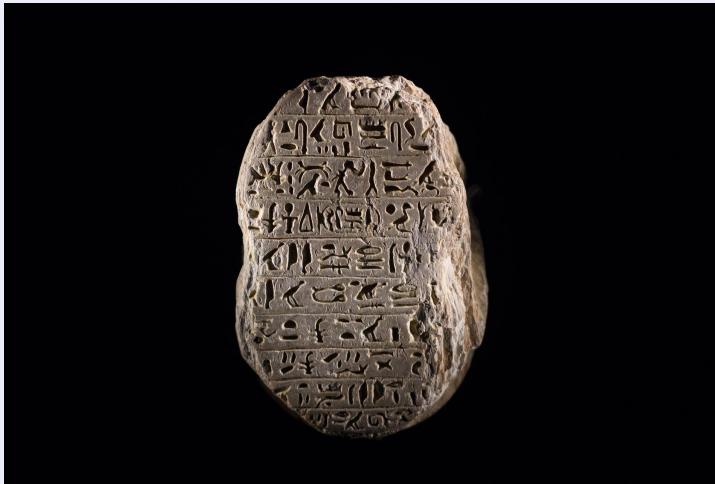
# Artifacts



PREFECT  
ASSOCIATE  
CERTIFICATION

# Artifacts

Persisted outputs such as Markdown, tables, or links.



# Artifacts

---



PREFECT ASSOCIATE  
CERTIFICATION

- Meant for human consumption
  - Data quality checks
  - Documentation
- Prettier output
- Not the main data transform result of a flow

# Artifacts - Markdown example



PREFECT ASSOCIATE  
CERTIFICATION

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

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

## Recent weather

| Time | Revenue |
| :----- | -----: |
| Now | {temp} |
| In 1 hour | {temp + 2} |
"""

    create_markdown_artifact(
        key="weather-report",
        markdown=markdown_report,
        description="Very scientific weather report",
    )


```

# Artifacts - Markdown example

Access from *Flow Runs* or  
*Artifacts* page



Very scientific weather report

Artifact Details Raw

## Weather Report

### Recent weather

Time	Revenue
Now	26.0
In 1 hour	28.0

# Deployment creation with `prefect init`



? You haven't specified a deployment configuration recipe. Would you like to see a list of available recipes? [Use arrows to move; enter to select]

	Name	Description
>	s3	Store code within an S3 bucket
	docker	Store code within a custom docker image alongside its runtime environment
	docker-s3	Store code within S3 and build a custom docker image for runtime
	docker-azure	Store code within an Azure Blob Storage container and build a custom docker image for runtime
	azure	Store code within an Azure Blob Storage container
	docker-gcs	Store code within GCS and build a custom docker image for runtime
	docker-git	Store code within a git repository and build a custom docker image for runtime
	local	Store code on a local filesystem
	git	Store code within git repository
	gcs	Store code within a GCS bucket

No, I'll use the default deployment configuration.

# Prefect Docker deployment recipe



```
prefect init --recipe docker
```

Prompts for image name and tag if not provided

## *Required inputs for 'docker' recipe*

Field Name	Description
<code>image_name</code>	The image name, including repository, to give the built Docker image
<code>tag</code>	The tag to give the built Docker image