PACC

Prefect Associate Certification Course

PREFECT

Norms reminder

Zoom

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

Slack

- Use threads
- Emoji responses 🙂

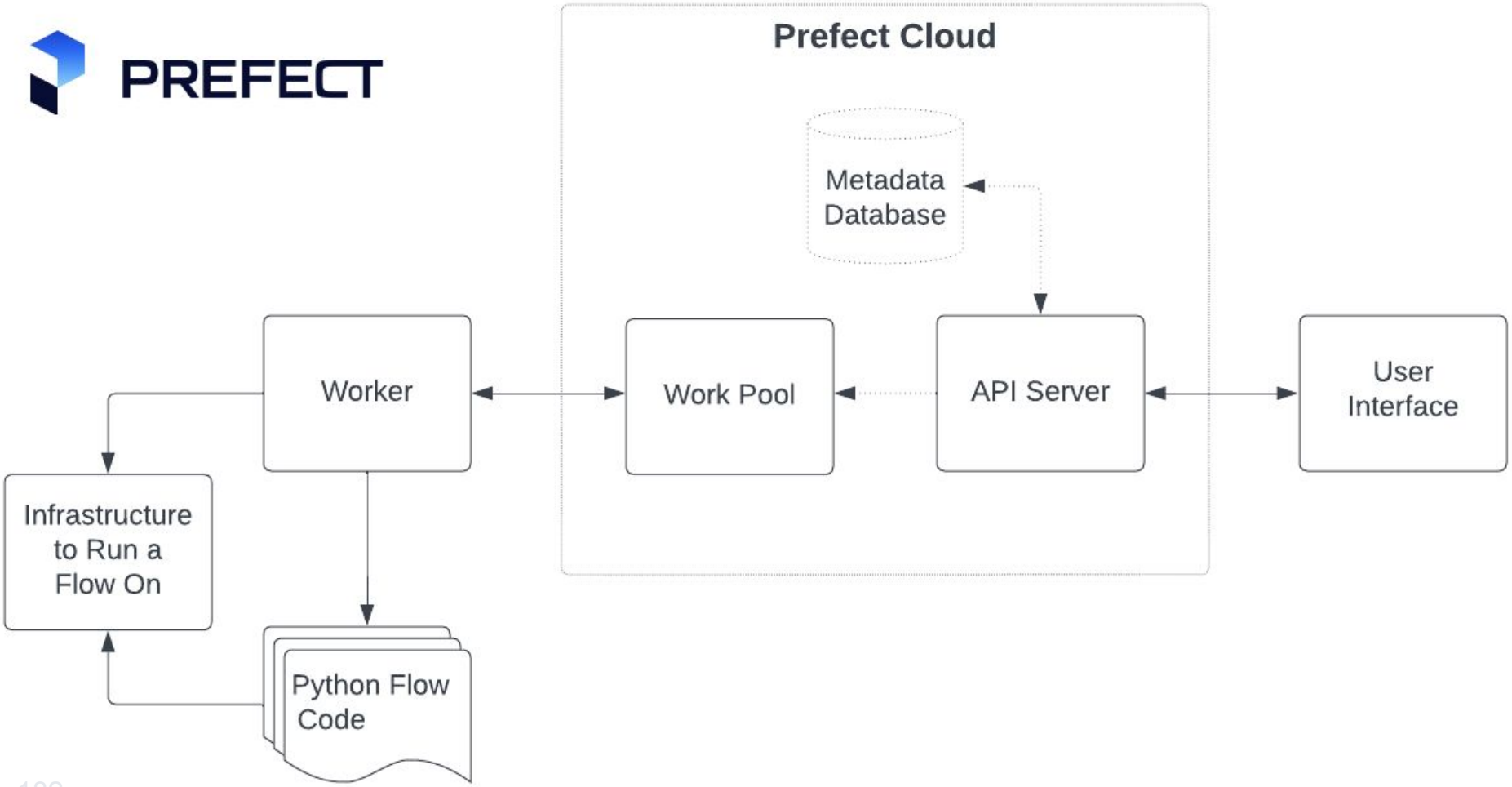
MODULE

104 - Worker-based Deployments

104 Agenda

- Hybrid model
- Worker-based deployments
- Create a deployment with interactive prompts
- Work pools
- Workers
- Run a deployment

Hybrid Model


Prefect Cloud
Metadata Database
Worker
Work Pool
API Server
User Interface
Infrastructure to Run a Flow On
Python Flow Code

Hybrid model

- 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

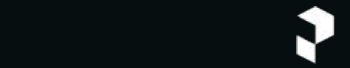
Guided deployment creation with workers

Deployments: ETL code

```python
@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

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

Send deployment to server

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

prefect deploy

Choose the flow you want to put into a deployment

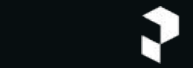
```
? Select a flow to deploy [Use arrows to move; enter to select; n to select none]
```

	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.

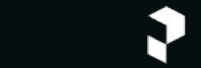
```
? Deployment name (default): first_deploy
? Would you like to schedule when this flow runs? [y/n] (y): n
```

Create a work pool

```
? 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 pools

- 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

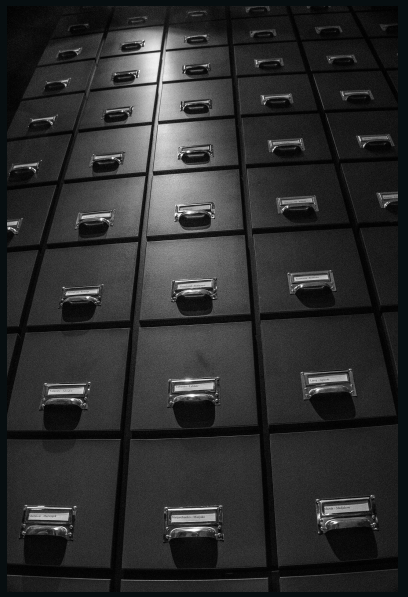
Give your work pool a name.

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	
>	**my-pool**	process	
	prod-pool	kubernetes	
	staging-pool	kubernetes	
	zoompool	process	

Flow code storage

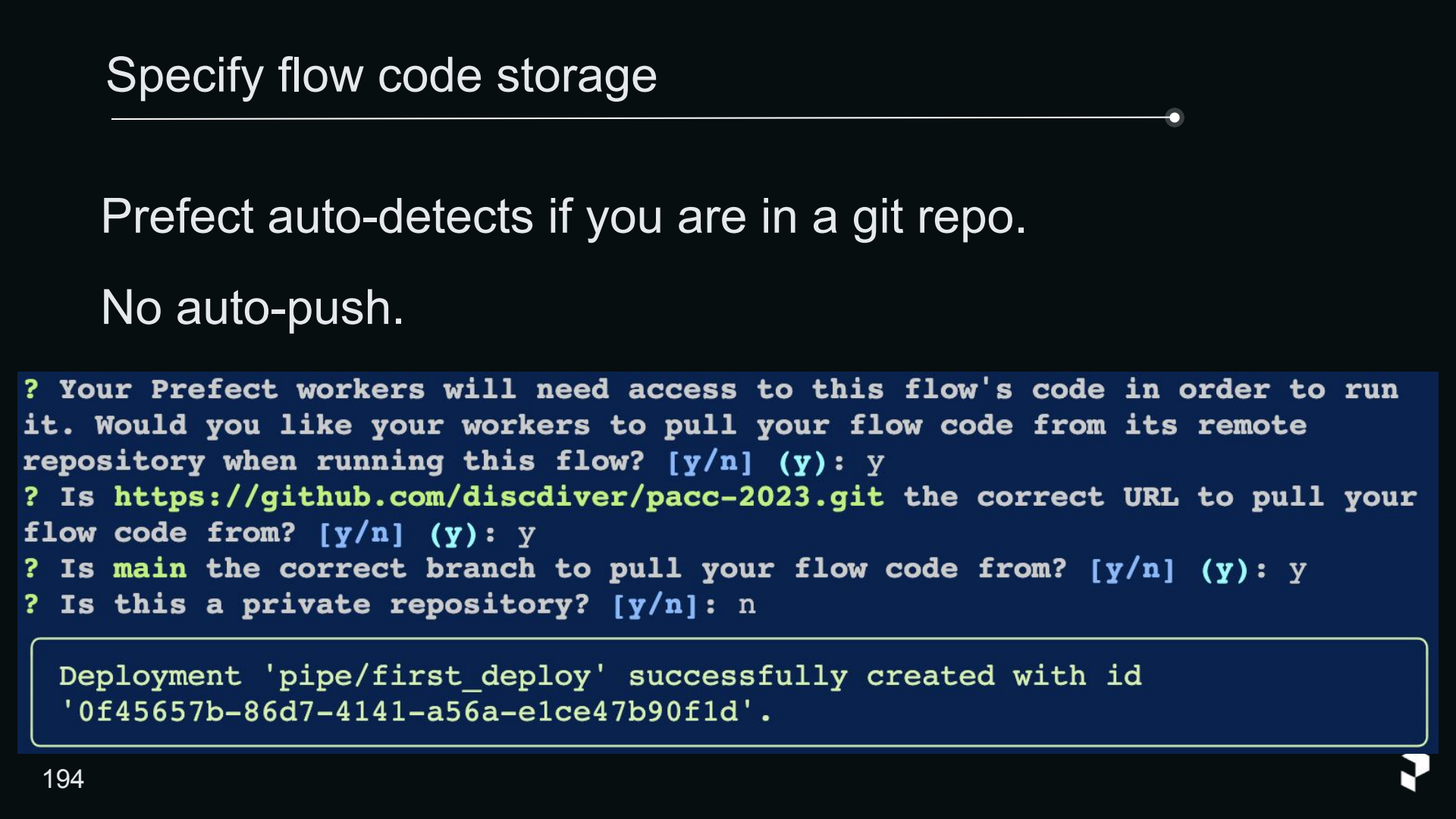


Specify flow code storage

Prefect auto-detects if you are in a git repo.

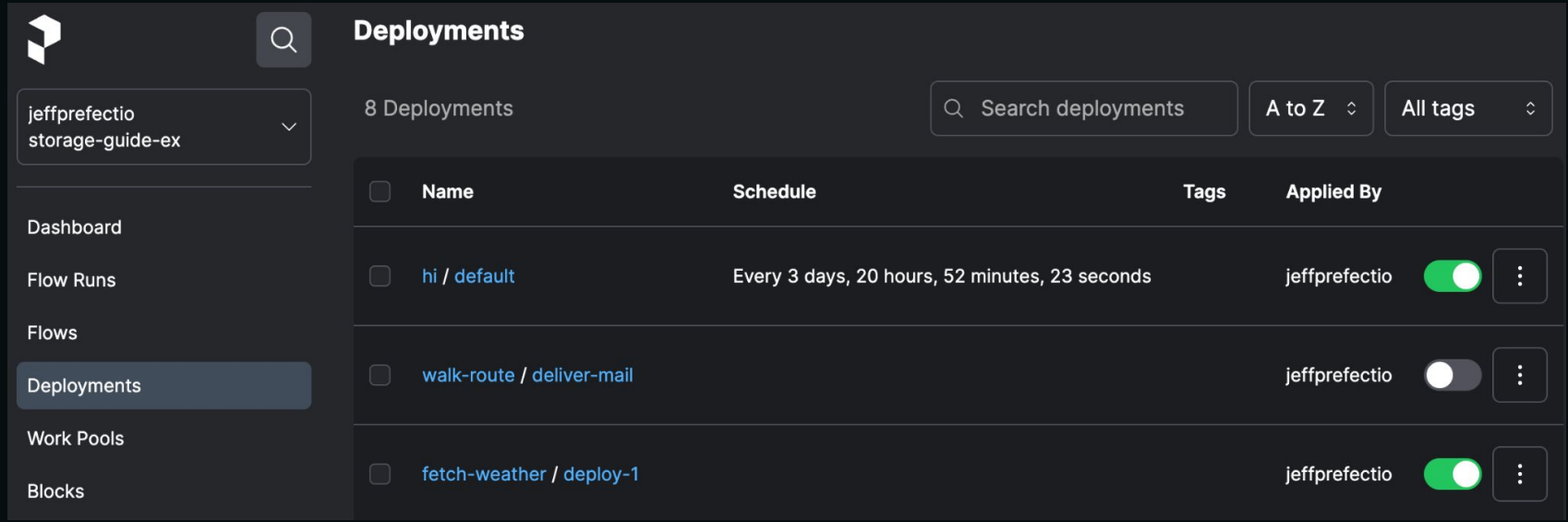
No auto-push.

```
? 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/discdiver/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.



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.
```

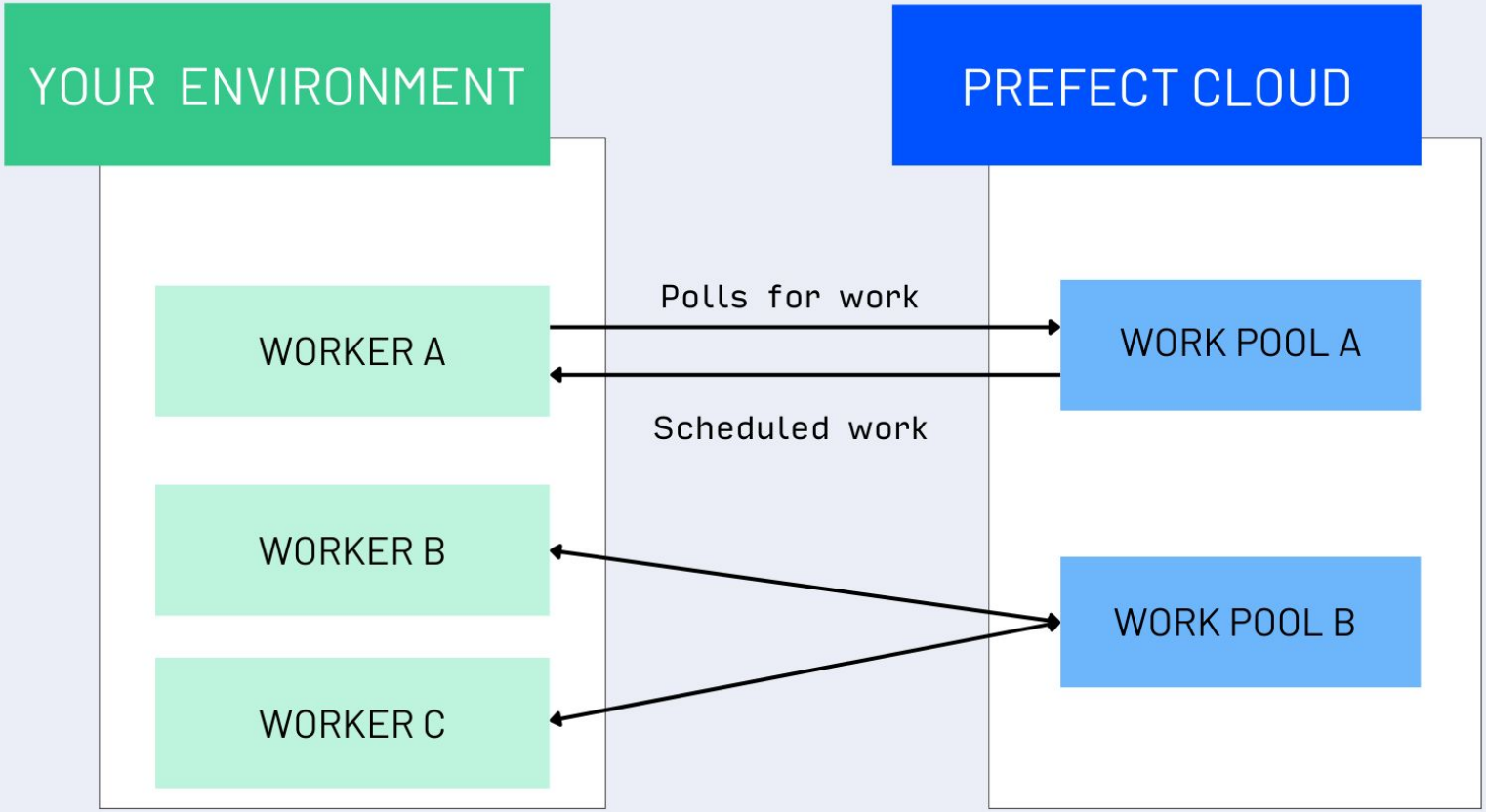
Workers

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:

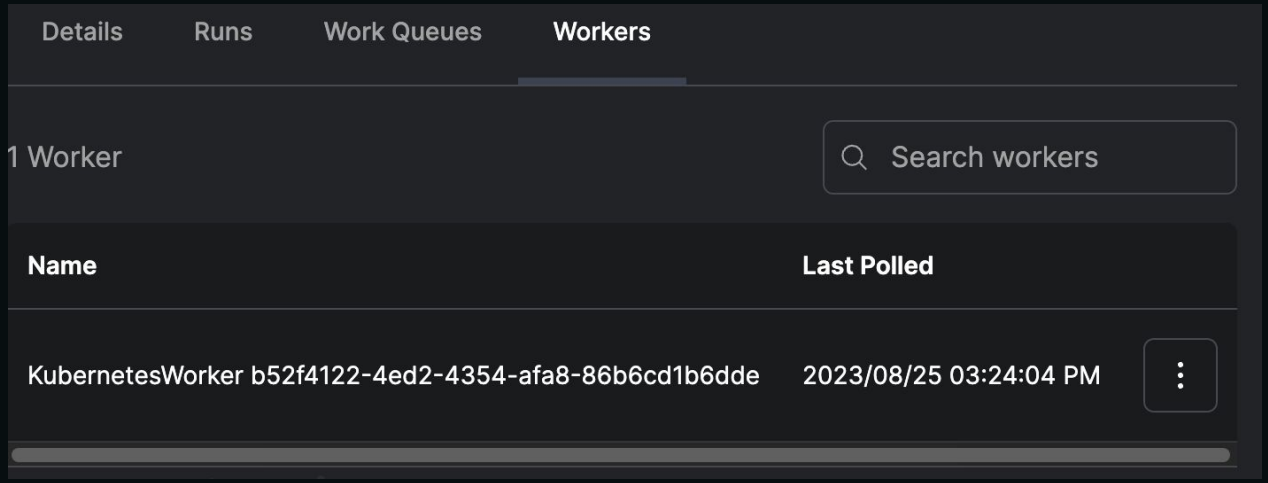
prefect worker start -p my_pool

WORKERS & WORK POOLS



Workers

Can see when a work pool was last polled by a worker in the UI

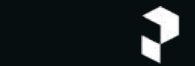


Recap of our setup

- 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*

Schedule a run - what happened?

- Running worker finds scheduled work in *my_pool* work pool.
- Worker and work pool are typed. *Local subprocess* 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.



Parameters

Parameters - argument values for flow function

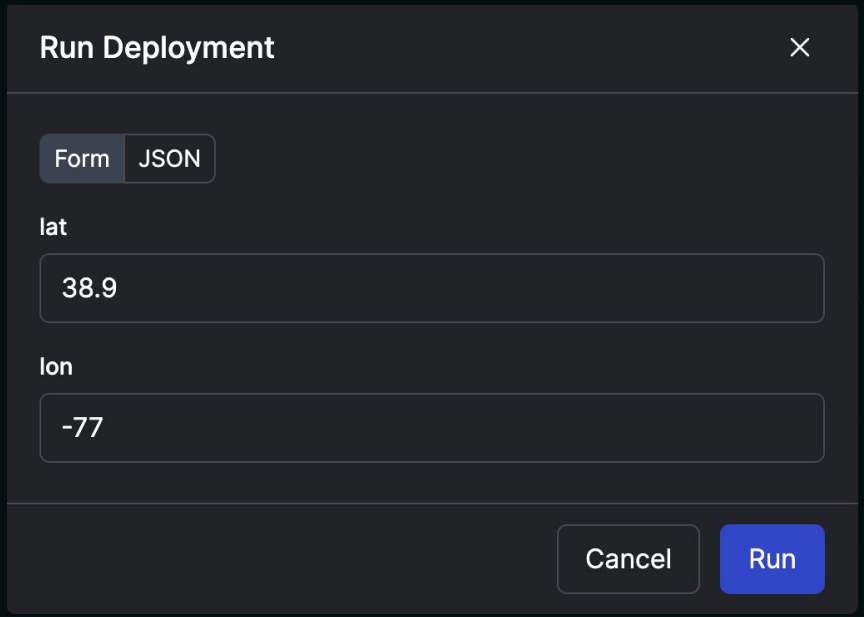
If your flow function has params and no defaults, you must feed it (give it values).

Parameters options

1. Make default arguments in flow function definition
2. Can override in deployment config
3. Can override both of the above at runtime

Parameters in the UI at runtime

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



Parameters from the CLI at runtime

prefect deployment run parametrized/dev --param user=Marvin --param answer=42

OR

prefect deployment run parametrized/dev --params '{"user": "Marvin", "answer": 42}'

104 Recap

You've learned

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

Lab 104

Reminder: breakout room norms

1. 🙂 Introduce yourselves
2. 🎥 Camera on (if possible)
3. 💻 One person shares screen
4. 👩‍💻 Everyone codes
5. 🙋 Each person talks
6. 😌 Low-pressure, welcoming environment: lean in

Breakout rooms with lots of participation = more fun + more learning! 😎

104 Lab

Use a flow from the 103 lab

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

If you give an engineer a job…

Could you

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

Lab 104: a solution

One person from each group, share your code in Slack 

Discuss

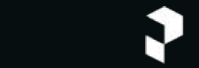
Questions?

MODULE

105 - Advanced orchestration

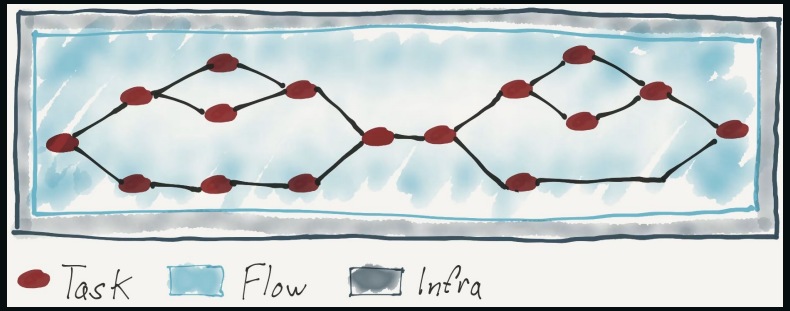
105 Agenda

- 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

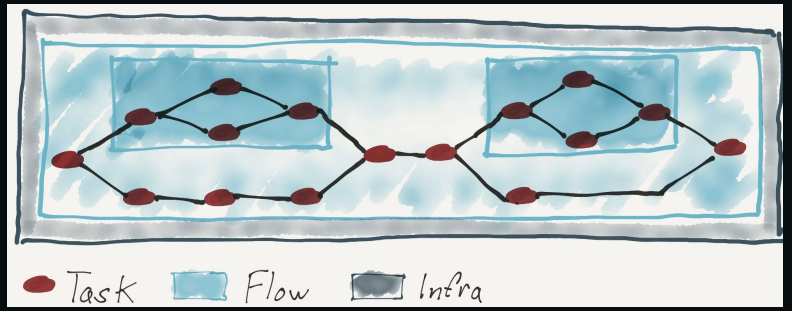


Workflow Patterns

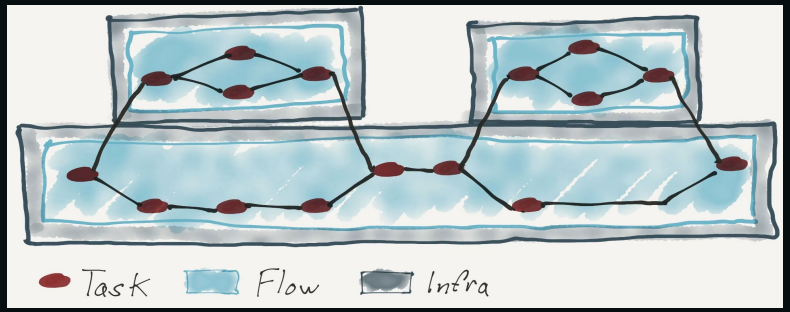
Workflow patterns - prefect.io/blog/workflow-design-patterns



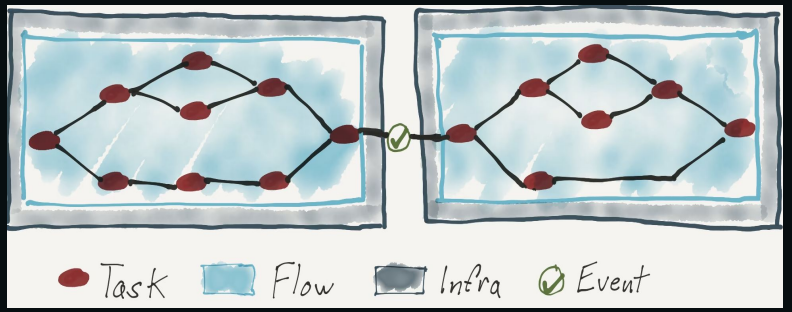
Monoflow



Flow of subflows



Flow of deployments

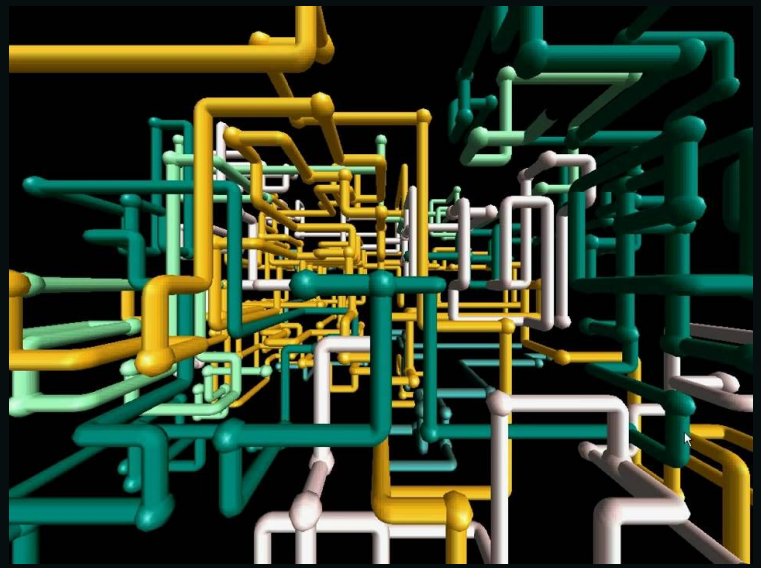


Event triggered flow

Subflows

Subflows

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



Subflows 🐈 + 🐩

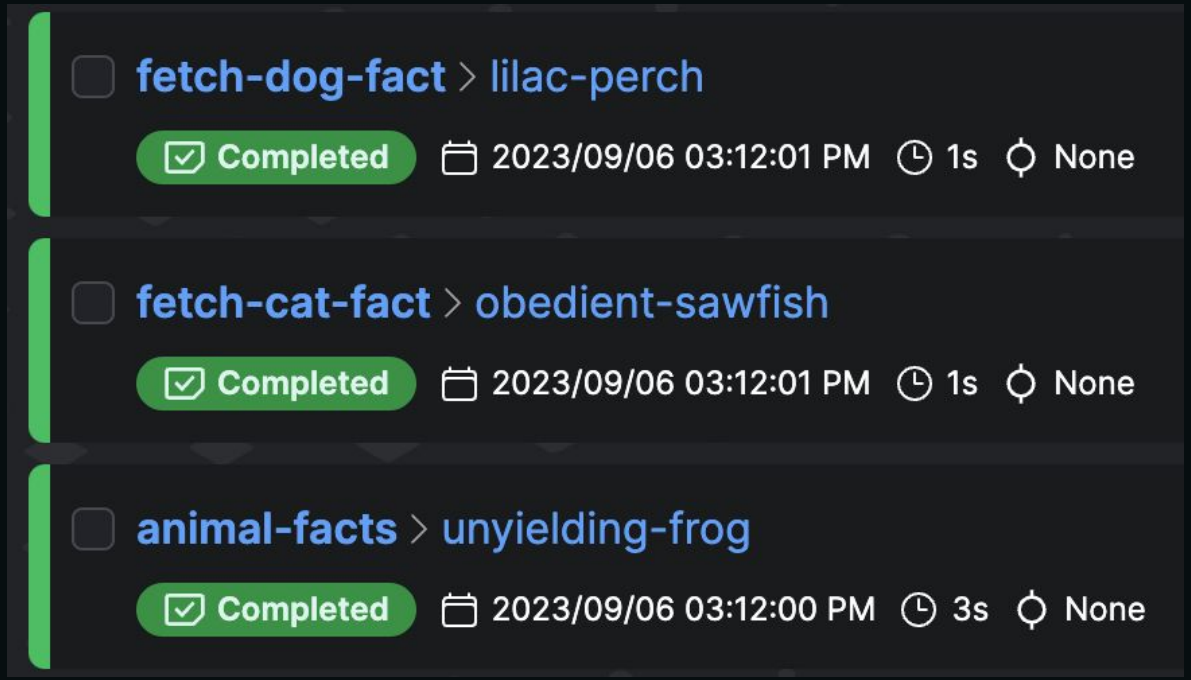
```python
@flow
def fetch_cat_fact():
    return httpx.get("https://catfact.ninja/fact?max_length=140").json()["fact"]

@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"🐱: {cat_fact} \n🐶: {dog_fact}")
```

Subflows 🐈 + 🐩

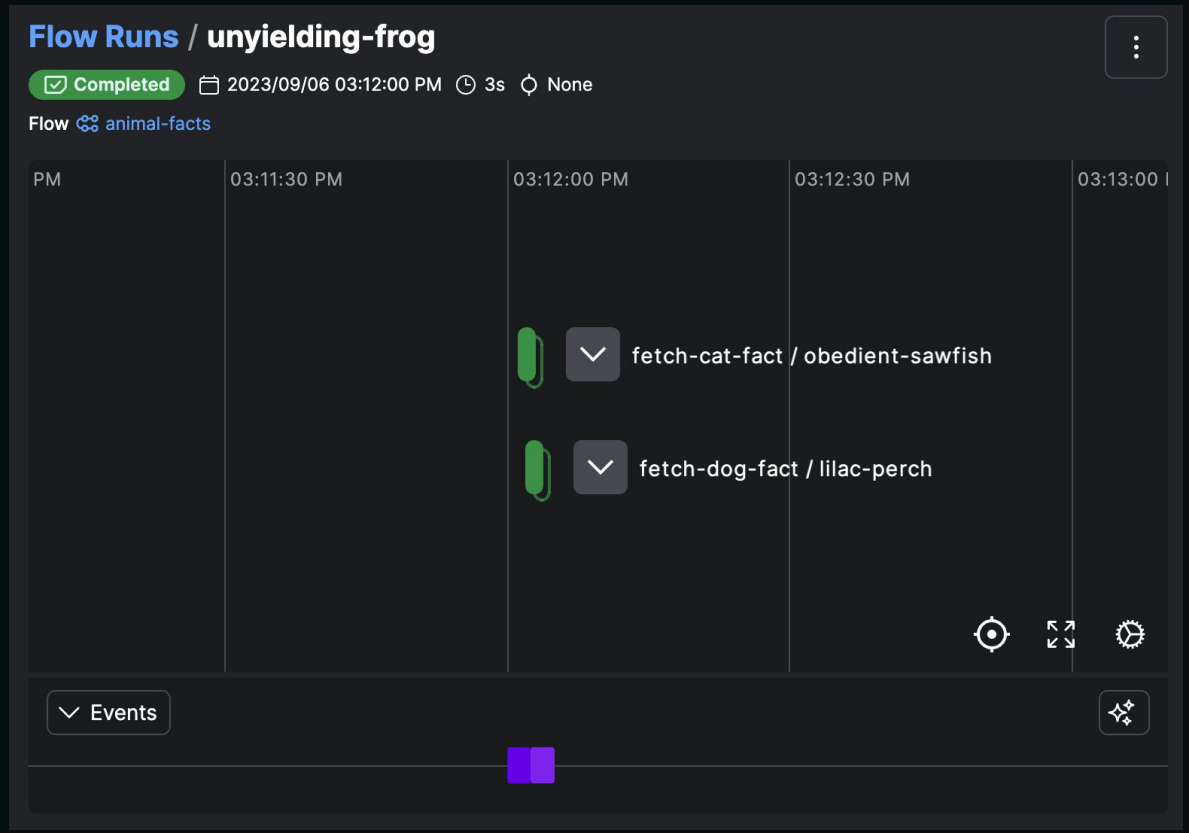
```
15:11:45.564 | INFO    | Flow run 'chirpy-agouti' - Finished in state Completed('All states completed.
')
(base) jeffhale prefect/pacc-2023 [main] $ python 105/subflow.py
15:12:00.416 | INFO    | prefect.engine - Created flow run 'unyielding-frog' for flow 'animal-facts'
15:12:00.420 | INFO    | Flow run 'unyielding-frog' - View at https://app.prefect.cloud/account/55c7f5
e5-2da9-426c-8123-2948d5e5d94b/workspace/7ad1ef2f-2f9c-49b5-b29f-4e0b3760d4c6/flow-runs/flow-run/90b24
4f2-260c-4c05-a323-79d41e036c4e
15:12:01.151 | INFO    | Flow run 'unyielding-frog' - Created subflow run 'obedient-sawfish' for flow
'fetch-cat-fact'
15:12:01.153 | INFO    | Flow run 'obedient-sawfish' - View at https://app.prefect.cloud/account/55c7f
5e5-2da9-426c-8123-2948d5e5d94b/workspace/7ad1ef2f-2f9c-49b5-b29f-4e0b3760d4c6/flow-runs/flow-run/96b5
f7ca-02c9-4e83-b2ba-3ab553e70248
15:12:01.556 | INFO    | Flow run 'obedient-sawfish' - Finished in state Completed()
15:12:01.848 | INFO    | Flow run 'unyielding-frog' - Created subflow run 'lilac-perch' for flow 'fetc
h-dog-fact'
15:12:01.850 | INFO    | Flow run 'lilac-perch' - View at https://app.prefect.cloud/account/55c7f5e5-2
da9-426c-8123-2948d5e5d94b/workspace/7ad1ef2f-2f9c-49b5-b29f-4e0b3760d4c6/flow-runs/flow-run/4431e414-
e0ce-4f01-b4fa-cc1c23d0a3e0
15:12:02.661 | INFO    | Flow run 'lilac-perch' - Finished in state Completed()
15:12:02.663 | INFO    | Flow run 'unyielding-frog' - 🐱: There are approximately 60,000 hairs per squ
are inch on the back of a cat and about 120,000 per square inch on its underside.
🐶: Puppies are blind, deaf and toothless when born.
15:12:02.750 | INFO    | Flow run 'unyielding-frog' - Finished in state Completed('All states complete
d.')
```

Run over to the UI

Subflows in UI



Timeline view



Logs in CLI

Sep 6th, 2023

INFO	Created subflow run 'obedient-sawfish' for flow 'fetch-cat-fact'	03:12:01 PM **prefect.flow_runs**
INFO	Created subflow run 'lilac-perch' for flow 'fetch-dog-fact'	03:12:01 PM **prefect.flow_runs**
INFO	🐱: There are approximately 60,000 hairs per square inch on the back of a cat and about 120,000 per square inch on its underside. 🐶: Puppies are blind, deaf and toothless when born.	03:12:02 PM **prefect.flow_runs**
INFO	Finished in state Completed('All states completed.')	03:12:02 PM **prefect.flow_runs**

run_deployment

`run_deployment` `async` ¶

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

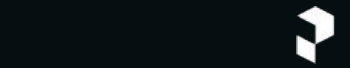
This function will return when the created flow run enters any terminal state or the timeout is reached. If the timeout is reached and the flow run has not reached a terminal state, it will still be returned. When using a timeout, we suggest checking the state of the flow run if completion is important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

Parameters:

Name	Type	Description	Default
`name`	`Union[str, UUID]`	The deployment id or deployment name in the form: `<slugified-flow-name>/<slugified-deployment-name>`	*required*
`parameters`	`Optional[dict]`	Parameter overrides for this flow run. Merged with the deployment defaults.	`None`

run_deployment

```python
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

- React to events from other systems
- Each exposes a URL endpoint
- Transforms external events into *Prefect events* for use in automations

Webhooks

```
prefect cloud webhook create my-webhook \
    --description some details' \
    --template '{
        "event": "demo.event",
        "resource": {"prefect.resource.id": "demo.alert.1"}
    }'
```


Webhooks

- 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

Webhooks

prefect cloud webhook ls

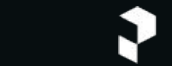
to see existing webhooks and get the url endpoint

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

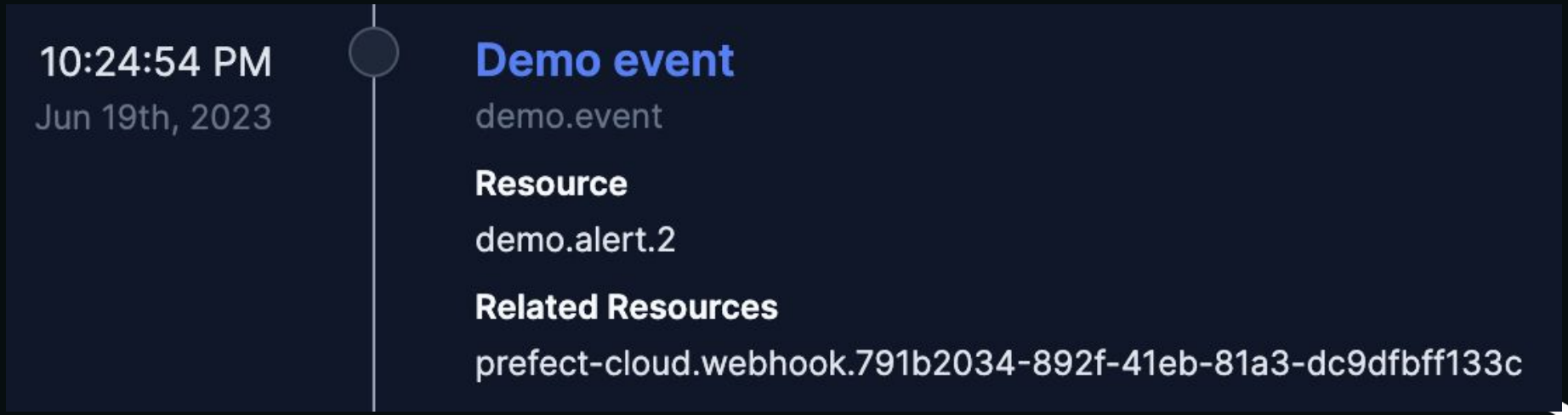
Hit the endpoint:

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



Webhooks

See the event in the Prefect Cloud workspace event feed



Webhooks

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

Webhooks

Potential use case:

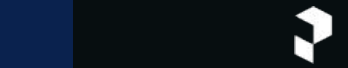
When a file arrives in cloud storage, trigger a lambda function to hit a Prefect webhook.

The webhook then triggers a deployment run.

prefect.yaml

prefect.yaml

```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/discdiver/pacc-2023.git
    branch: main
```

prefect.yaml

Configuration for creating deployments

- ***pull*** step (repository & branch): from git repo

prefect.yaml

- ***deployments***:

  Config for one or more deployments

Required keys:

- *name*
- *entrypoint*
- *work_pool -> name*

```yaml
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

```yaml
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/discdiver/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/discdiver/pacc-london-2023.git
          branch: staging
```


Re-deploy 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
>	first_deploy	104/flows.py:pipe	

Deploy multiple deployments at once

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

prefect deploy --all

Scheduling

Scheduling

Scheduling

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

Schedule types

- Interval
- Cron
- RRule

Add a schedule

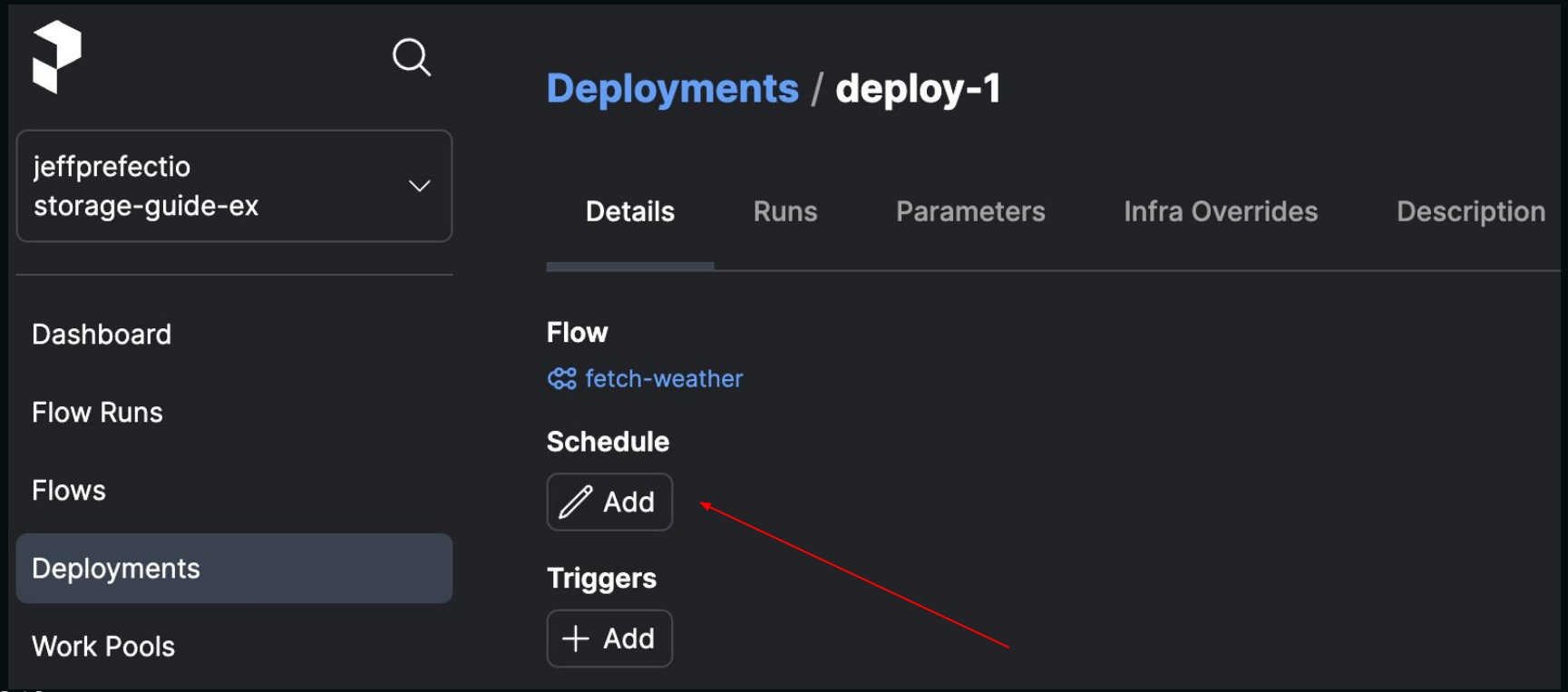
prefect deploy command - interactive

```
? 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]
```

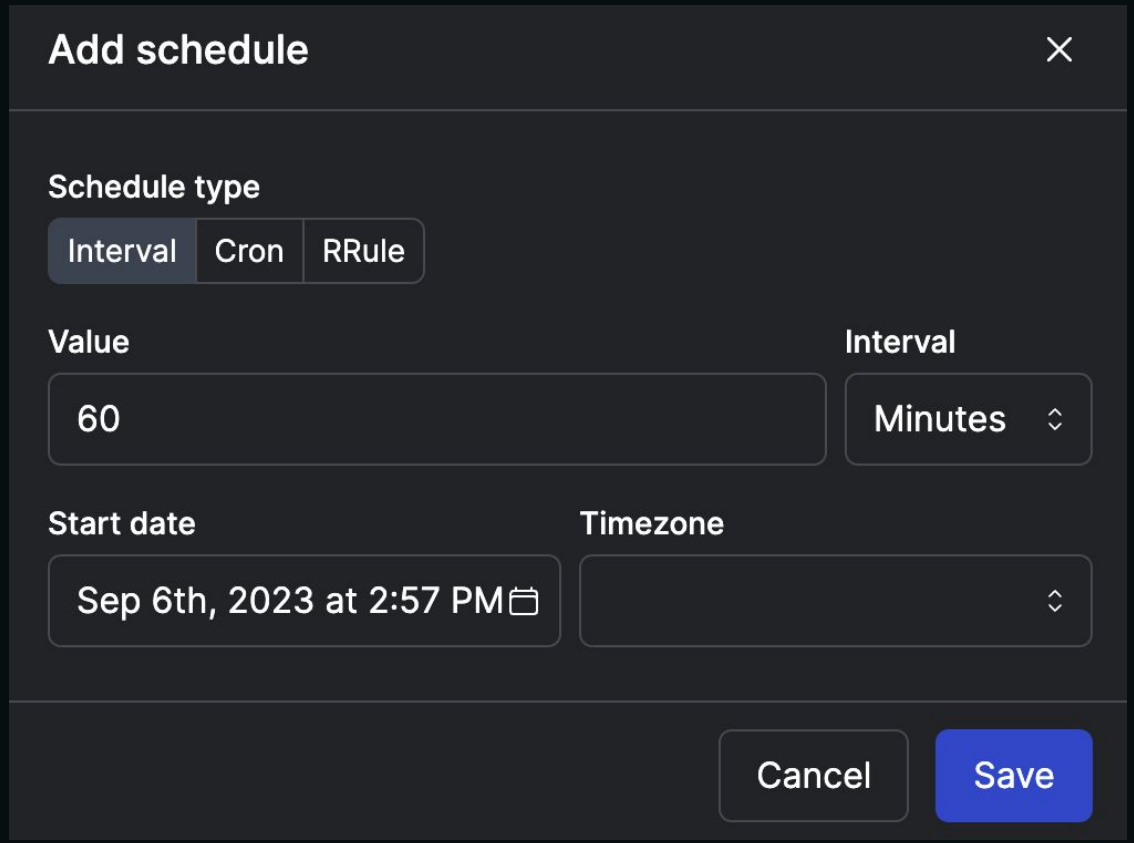
	Schedule Type	Description
>	Interval	Allows you to set flow runs to be executed at fixed time intervals.
	Cron	Allows you to define recurring flow runs based on a specified pattern using cron syntax.
	RRule	Allows you to define recurring flow runs using RFC 2445 recurrence rules.

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


In the UI


jeffprefectio
storage-guide-ex
Dashboard
Flow Runs
Flows
Deployments
Work Pools
Deployments / deploy-1
Details
Runs
Parameters
Infra Overrides
Description
Flow
fetch-weather
Schedule
Add
Triggers
Add

In the UI


Add schedule
Schedule type
Interval
Cron
RRule
Value
60
Interval
Minutes
Start date
Sep 6th, 2023 at 2:57 PM
Timezone
Cancel
Save

RRule

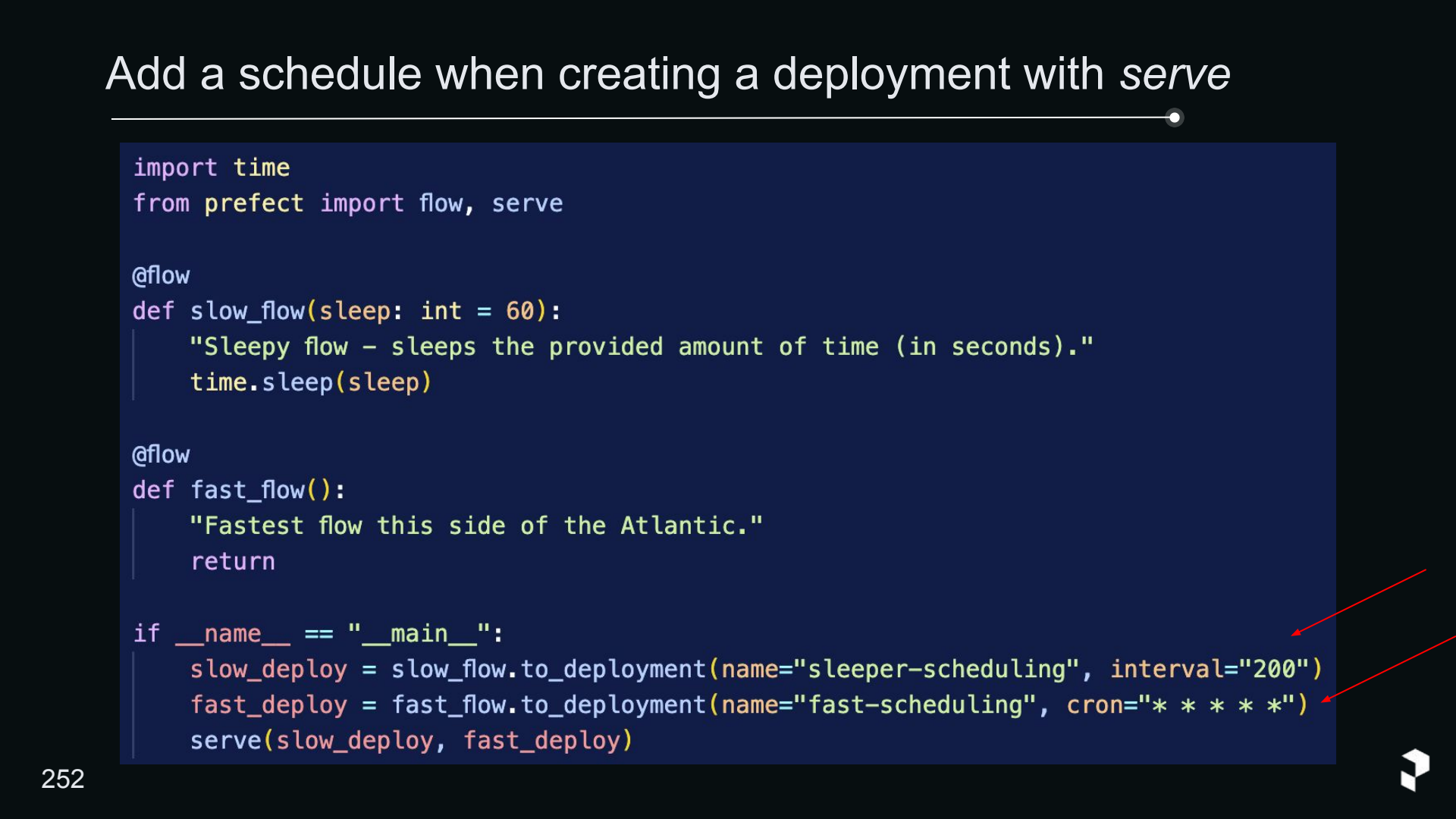
RRule cheat sheet: jakubroztocil.github.io/rrule/

Or ask Marvin (another Prefect package) *pip install*

```python
from marvin import ai_fn

@ai_fn
def rrule(text: str) -> str:
    """
    Generate valid RRULE strings from a natural language description of an event
    """
    yield pendulum.now.isoformat()

rrule('every hour from 9-6 on thursdays')
# "RRULE:FREQ=WEEKLY;BYDAY=TH;BYHOUR=9,10,11,12,13,14,15,16;BYMINUTE=0;BYSECOND=0"
```

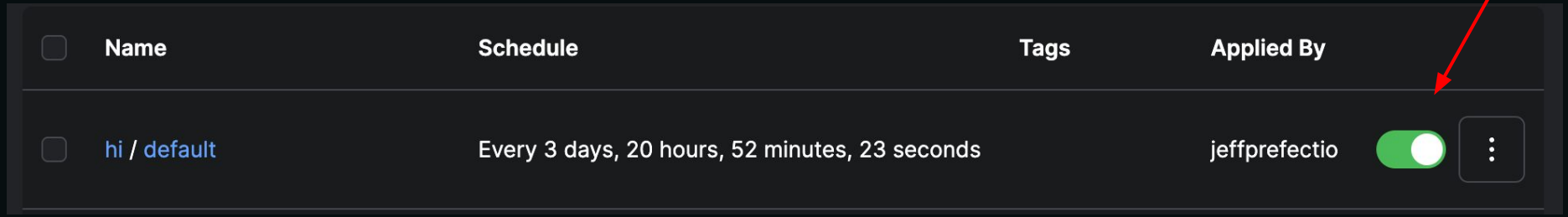

Add a schedule when creating a deployment with *serve*

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

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

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

if __name__ == "__main__":
    slow_deploy = slow_flow.to_deployment(name="sleeper-scheduling", interval="200")
    fast_deploy = fast_flow.to_deployment(name="fast-scheduling", cron="* * * * *")
    serve(slow_deploy, fast_deploy)
```



Pausing and restarting schedules

Pause/resume schedule from UI



Pause/resume schedule from CLI

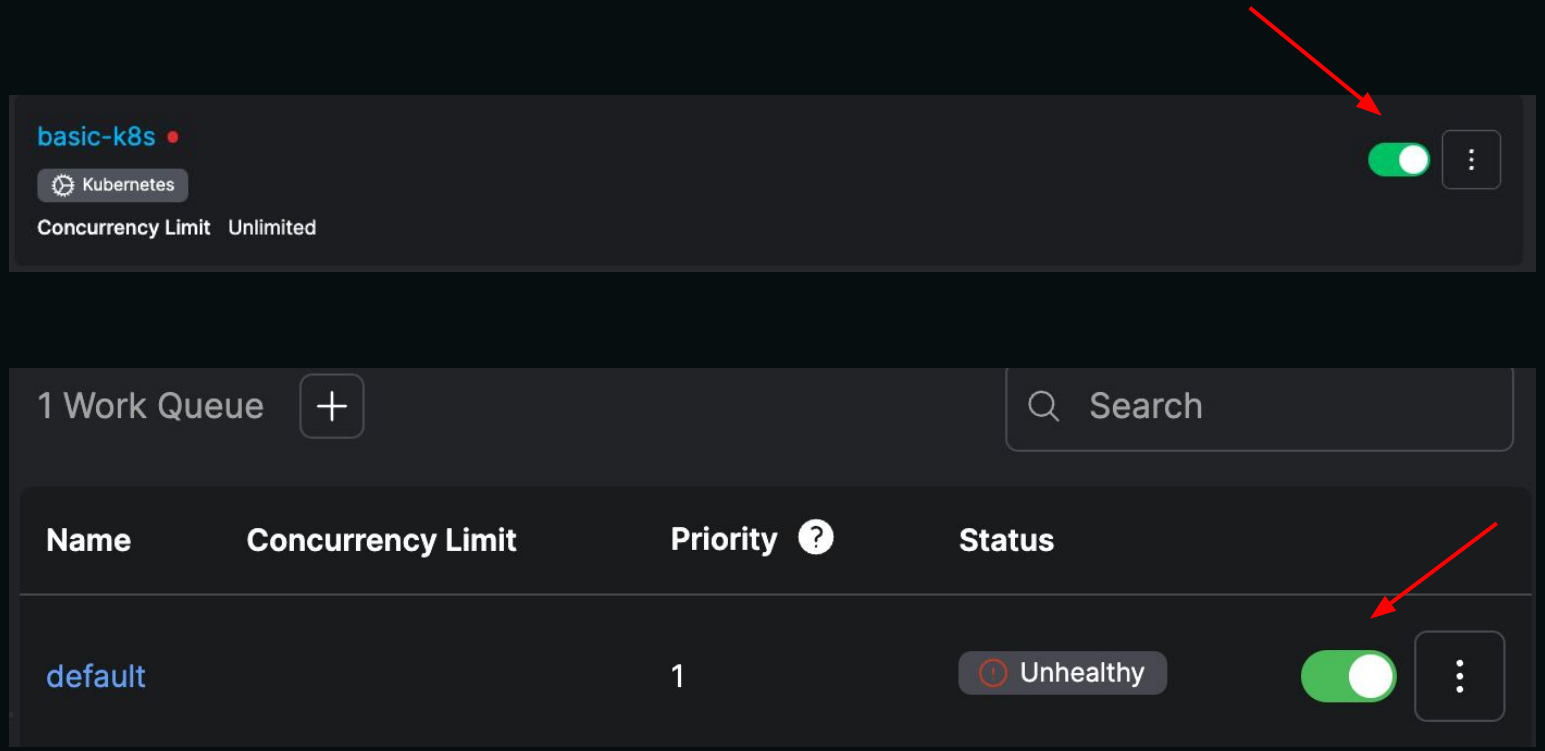
Pause ⏸️

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

Resume ▶️

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

Pause work pools or work queues



What’s a work queue for anyway?

Prioritizing work

⚠️ default work queue is created automatically

105 Recap

You've seen how to:

- Use several workflow patterns
- Modify *prefect.yaml*
- Create schedules from the UI & CLI
- Pause and resume schedules ⏯️

Lab 105

105 Lab

- Create a deployment that uses a subflow.
- Use GitHub or other git-repo-based code storage.
  - Don’t forget to push your code!
- Check out the *prefect.yaml* file.
- Add a cron schedule.
- Pause the schedule.
- Stretch: Create a second deployment that uses *run_deployment*
- Super stretch: Create a webhook and automation that runs a deployment in response to an event firing

If you give an engineer a job…

Could you

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

Lab 105: a solution

One person from each group, share your code in Slack 

Discuss

Questions?

MODULE

201 - Container-based work pools

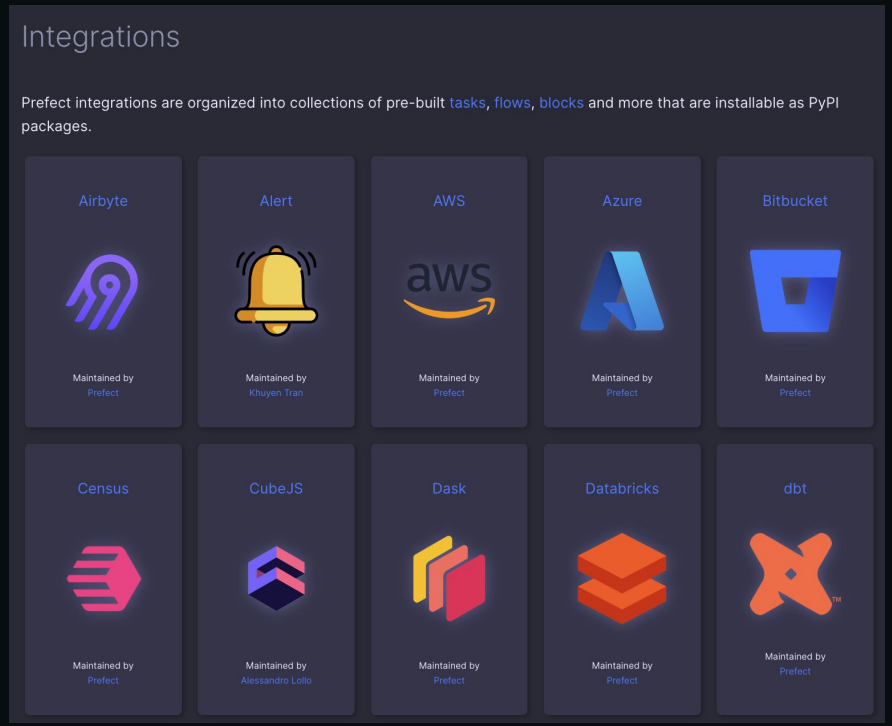
201 Agenda

- Integrations
- Docker
- Container-based work pools with Docker
- Building Docker-based deployments

Integrations

Integrations

docs.prefect.io/integrations/catalog/



Integrations

Python packages that add convenience

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

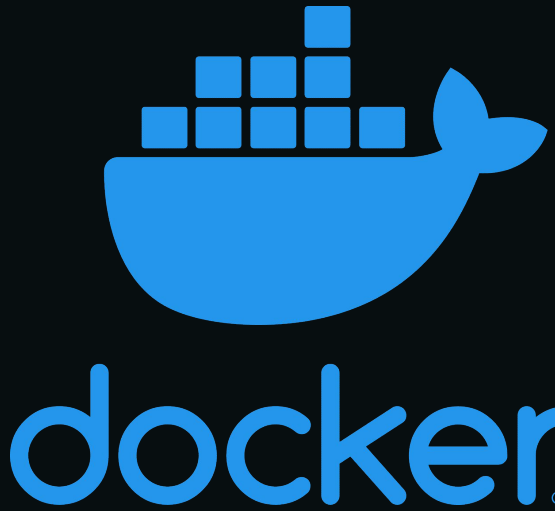
Integrations

Prefect is Python-based and designed for flexibility

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

Container-based deployments

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

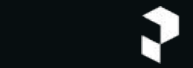


Why use Docker?

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

Docker

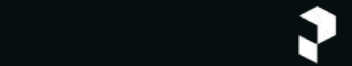
- Prefect provides base Docker images
- Can customize base image
- Read about choosing images at docs.prefect.io/latest/concepts/infrastructure/#standard-python



Docker

Prerequisites

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

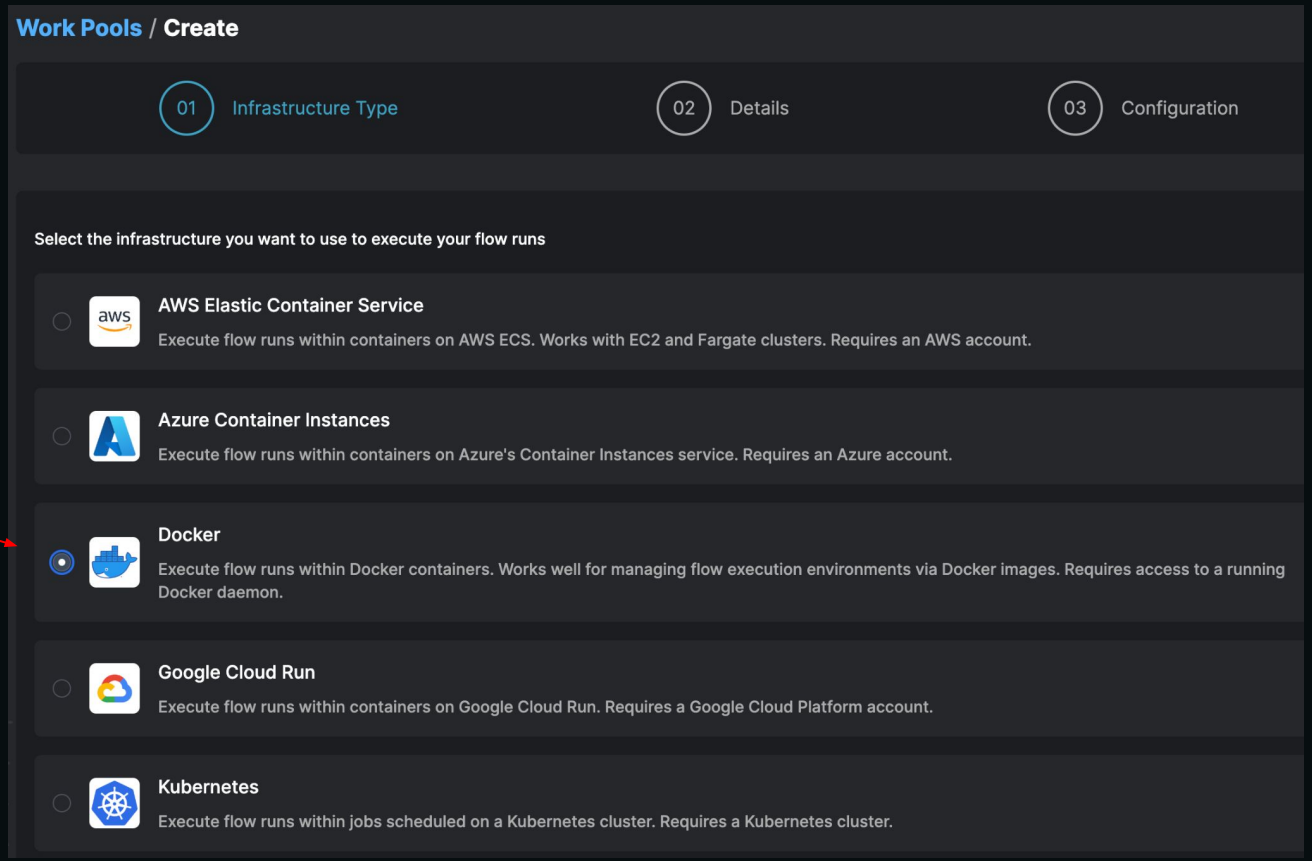


Work pools - Docker

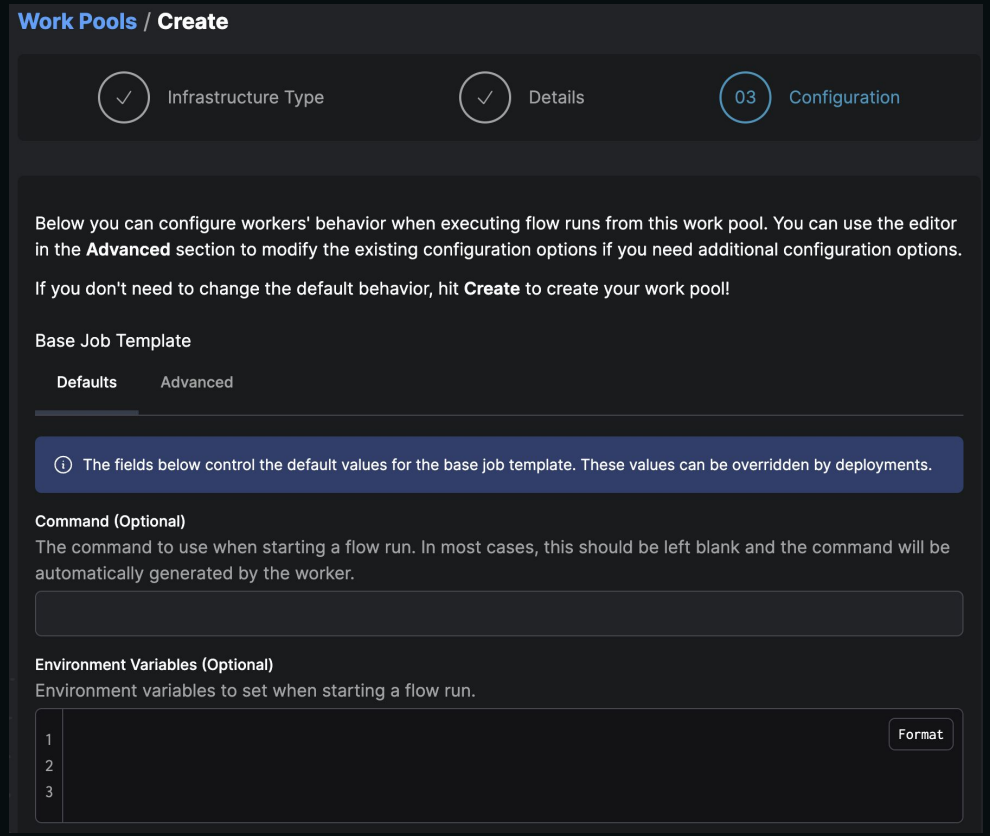
Work pools

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

Work pools

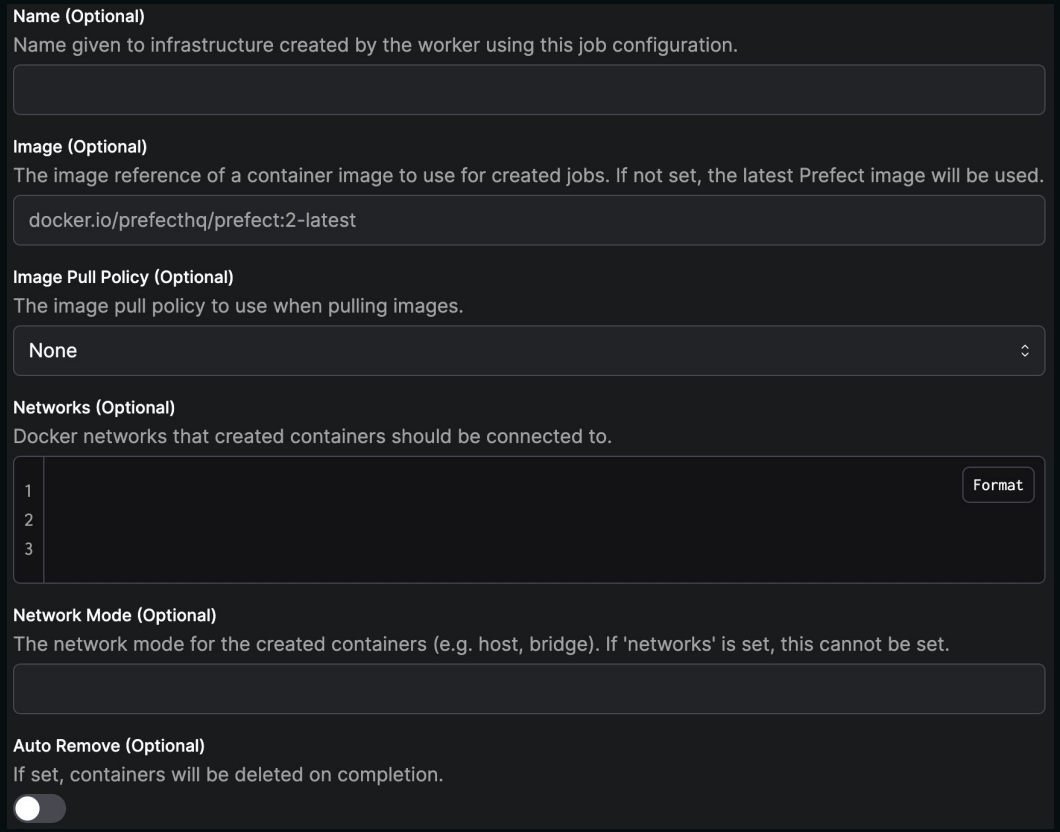

Work Pools / Create
01 Infrastructure Type
02 Details
03 Configuration
Select the infrastructure you want to use to execute your flow runs
AWS Elastic Container Service
Execute flow runs within containers on AWS ECS. Works with EC2 and Fargate clusters. Requires an AWS account.
Azure Container Instances
Execute flow runs within containers on Azure's Container Instances service. Requires an Azure account.
Docker
Execute flow runs within Docker containers. Works well for managing flow execution environments via Docker images. Requires access to a running Docker daemon.
Google Cloud Run
Execute flow runs within containers on Google Cloud Run. Requires a Google Cloud Platform account.
Kubernetes
Execute flow runs within jobs scheduled on a Kubernetes cluster. Requires a Kubernetes cluster.

Container-based work pools



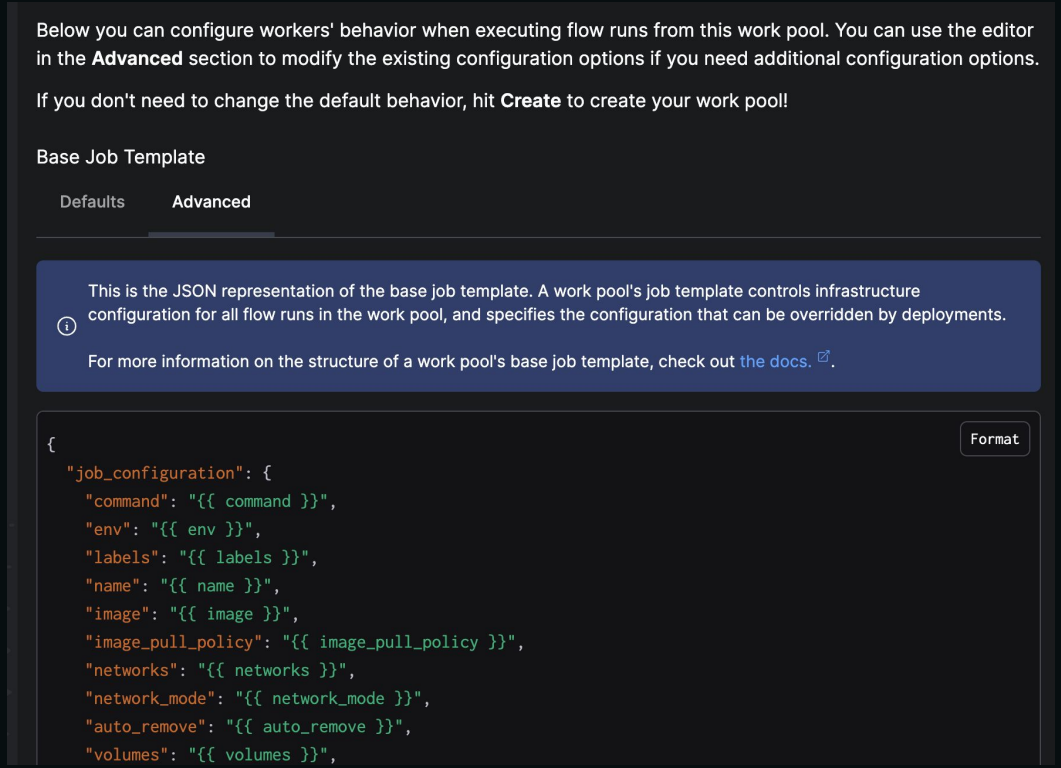
Container-based work pools

Can specify a particular image your team created



Container-based work pools

Or click on **Advanced** tab to customize anything and add or limit fields



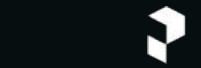
Prefect Docker Deployments

prefect deploy

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

	Work Pool Name	Infrastructure Type	Description
>	docker-pool	docker	
	my-pool	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

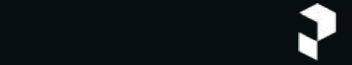
Build a custom Docker image with flow code

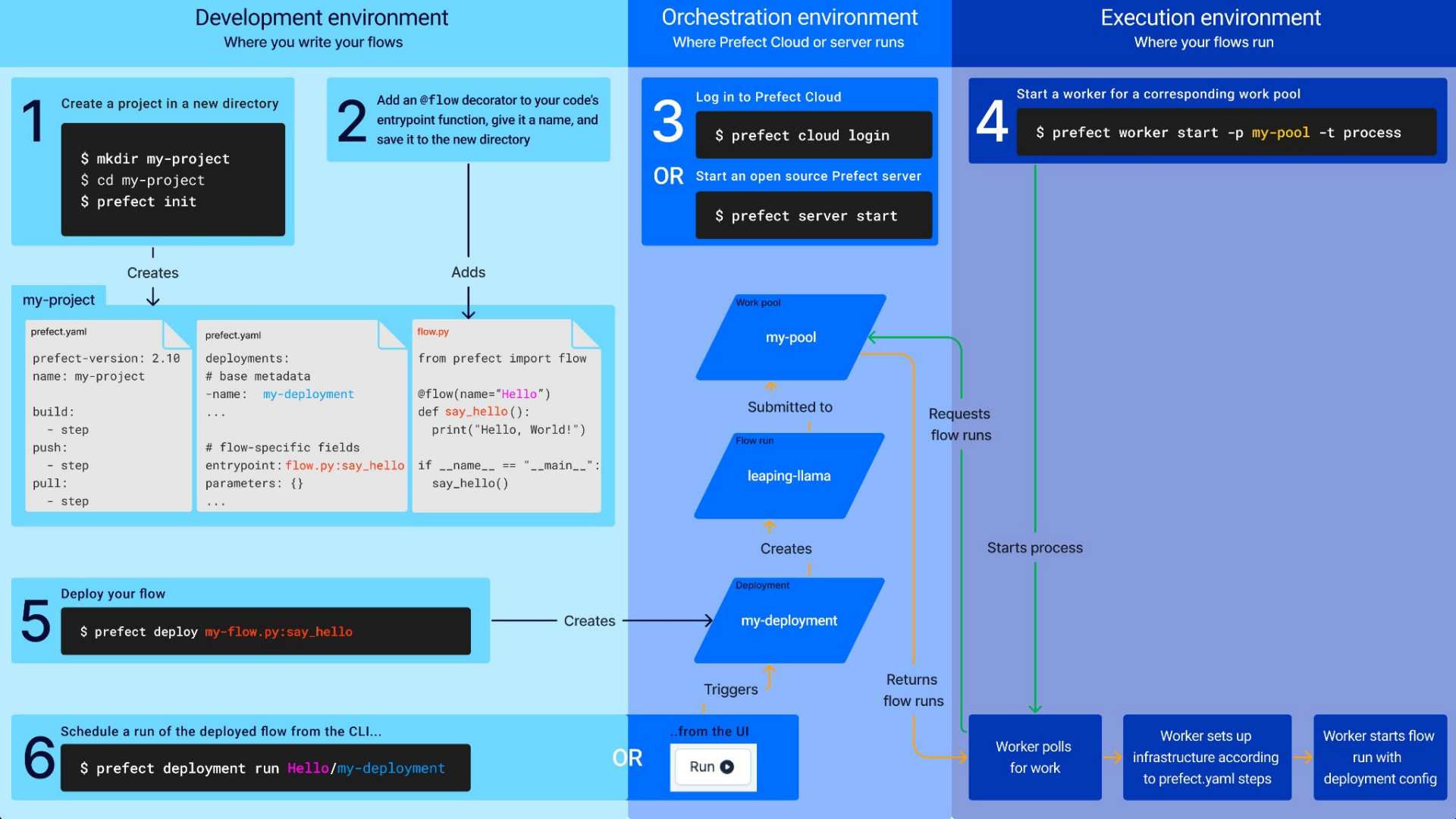
- Push image to a Docker registry
  - Use existing Dockerfile
  - Auto-includes packages in *requirements.txt*

Follow the prompts. 🙂

Resulting *prefect.yaml*

```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
Orchestration environment
Where Prefect Cloud or server runs
Execution environment
Where your flows run
1
Create a project in a new directory
$ mkdir my-project
$ cd my-project
$ prefect init
2
Add an @flow decorator to your code's entrypoint function, give it a name, and save it to the new directory
3
Log in to Prefect Cloud
$ prefect cloud login
OR
Start an open source Prefect server
$ prefect server start
4
Start a worker for a corresponding work pool
$ prefect worker start -p my-pool -t process
Creates
Adds
my-project
prefect.yaml
prefect-version: 2.10
name: my-project
build:
- step
push:
- step
pull:
- step
prefect.yaml
deployments:
base metadata
-name: my-deployment
...
flow-specific fields
entrypoint: flow.py:say_hello
parameters: {}
...
flow.py
from prefect import flow
@flow(name="Hello")
def say_hello():
print("Hello, World!")
if __name__ == "__main__":
say_hello()
Work pool
my-pool
Submitted to
Flow run
leaping-llama
Creates
Deployment
my-deployment
Requests flow runs
Starts process
5
Deploy your flow
$ prefect deploy my-flow.py:say_hello
Creates
Triggers
Returns flow runs
6
Schedule a run of the deployed flow from the CLI...
$ prefect deployment run Hello/my-deployment
OR
..from the UI
Run
Worker polls for work
Worker sets up infrastructure according to prefect.yaml steps
Worker starts flow run with deployment config

Docker - worker

Start a Docker type worker to connect to a work pool named *docker-work*

prefect worker start -p docker-work

Dockerfile

```
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

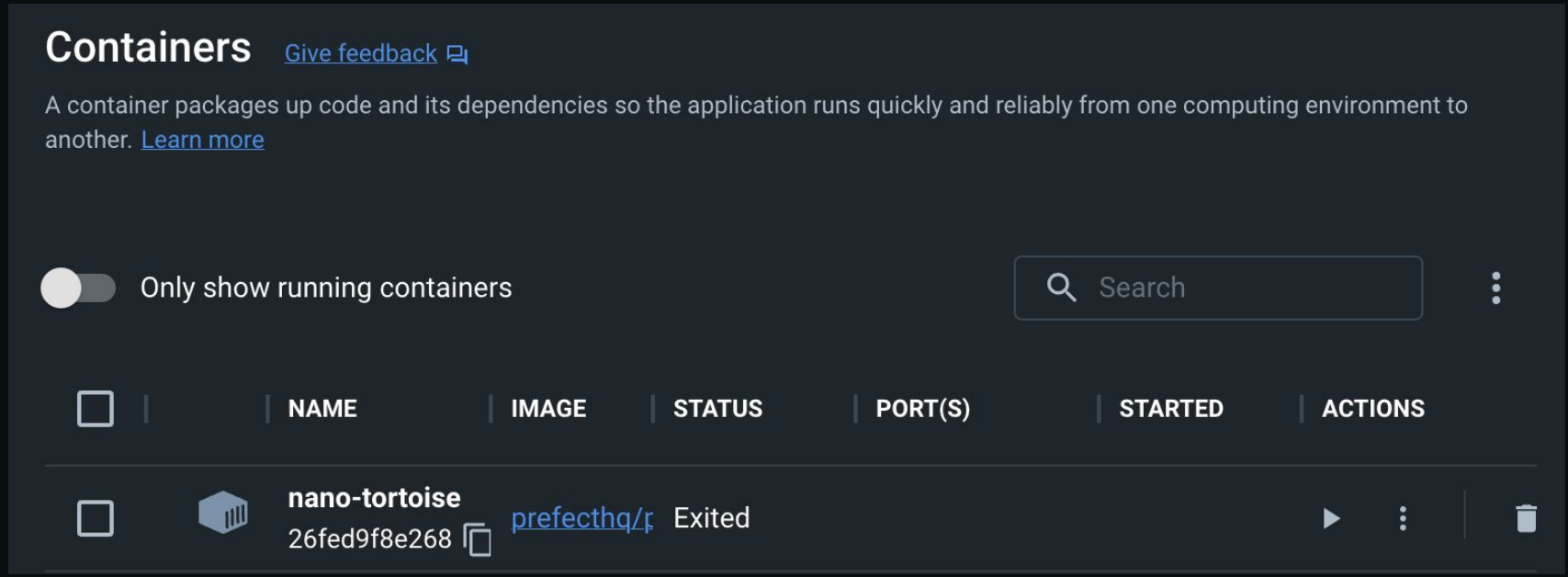
(Can use *deployments* section of *prefect.yaml*)

Docker

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

Infrastructure - Docker Container

Check out in Docker Desktop if running Docker locally



Infrastructure choices

More on choosing between *serve* and worker-based deployments

Q: Do you have scalable compute needs?

If no, ***serve* is good**.

Q: If yes, where does your compute happen?

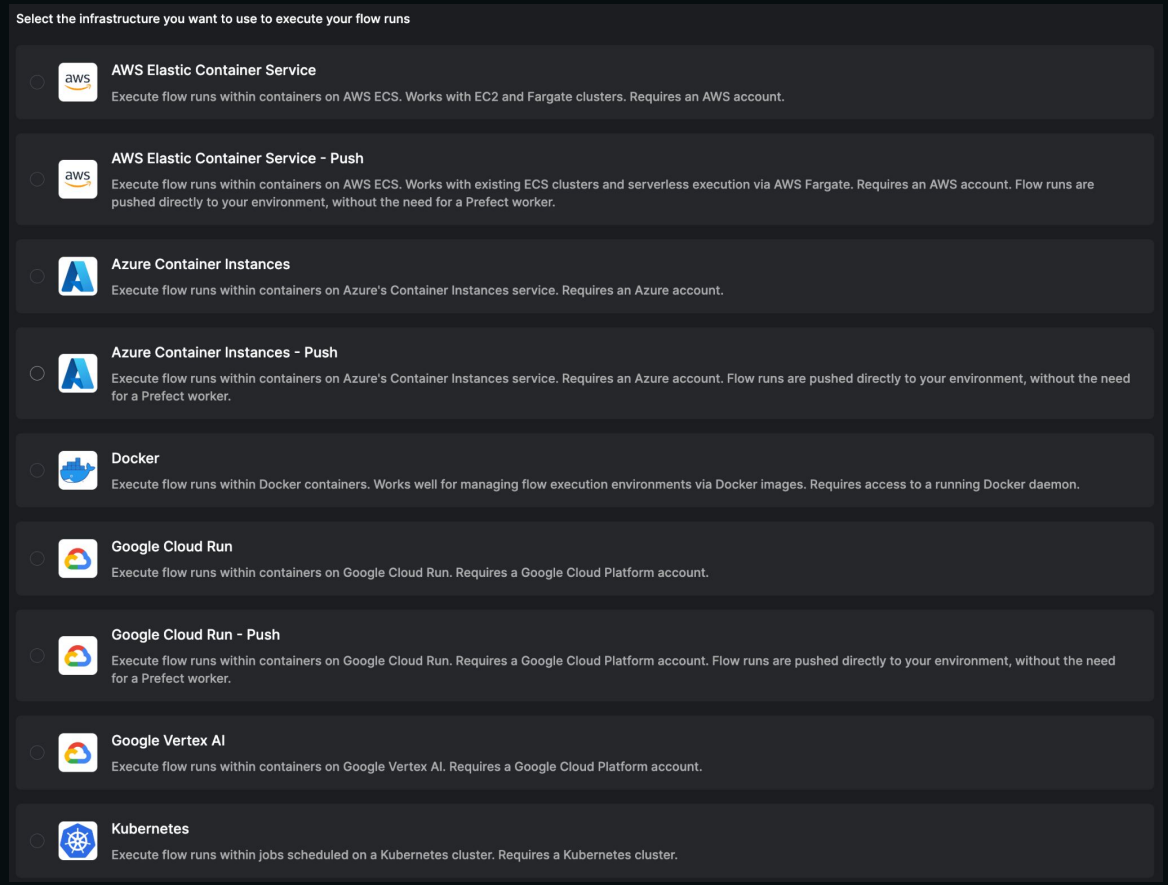
If exclusively in a platform like Snowflake or Databricks, then probably don't need Prefect's worker capabilities with infrastructure. ***serve* is good**.

If compute happens elsewhere, **likely want worker-based deployments** for max infrastructure control. (e.g. spin up an ECS container only when need to retrain ML model)

Alternative ways to run flows in containers

1. Kubernetes
2. Serverless containers such as ECS
3. Push work pools (no worker required) - uses serverless options such as ECS
4. Use *serve* inside a container - no worker or work pool required

Work pool options



201 Recap

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

Docker containers are the basis for many production infrastructure setups.

Lab 201

201 Lab

As time allows/on your own:

- Make a Docker work pool
- Create a deployment that will run in a Docker container
- Start a Docker type worker that polls the pool
- Run the deployment
- Stretch 1: push the image to Docker Hub
- Stretch 2: use Kubernetes
- Stretch 3: use a push work pool
- Stretch 4: deploy a flow in a Docker container with *serve*

If you give an engineer a job…

Could you

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

Wrap

Brief feedback survey

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

Congratulations!!!

PREFECT ASSOCIATE

CERTIFICATION

PACC

Prefect Associate Certification Course

PREFECT

MODULE

Bonus Content

Bonus content

- Prefect variables
- Task runners & async code
- Testing
- CI/CD with GitHub Actions
- Prefect REST API
- Prefect Runtime
- Upload data to AWS S3

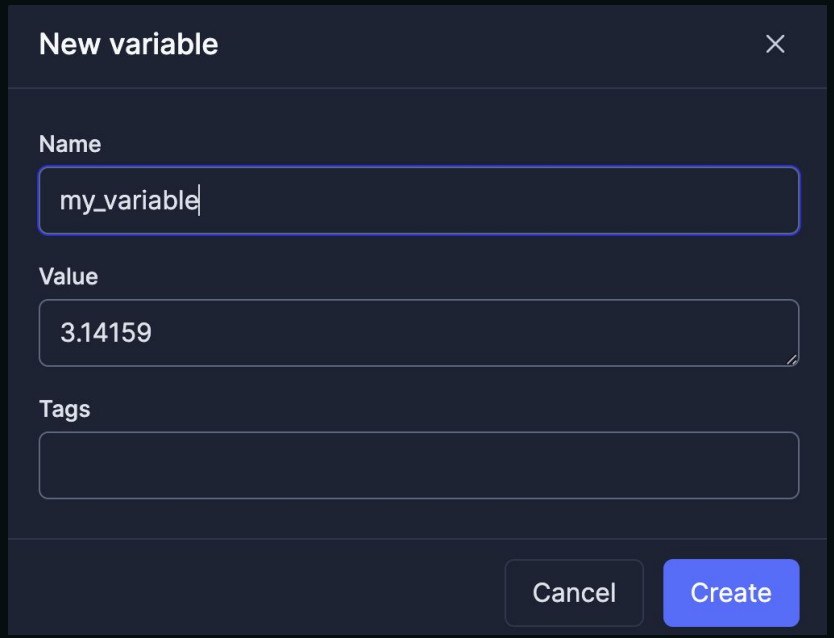
Variables

Prefect variables

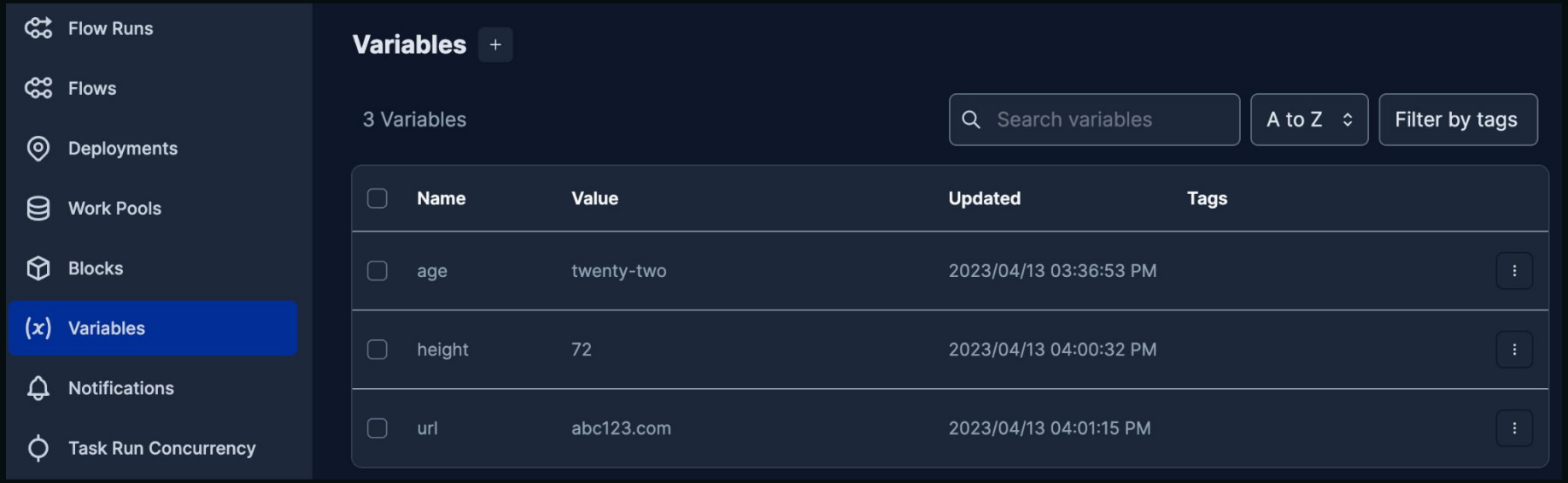
- String values evaluated at runtime
- Store and reuse non-sensitive, small data
- Create via UI or CLI

Prefect variables

Only string values



Prefect variables



Task runners for concurrency

Concurrency

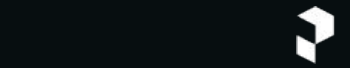
- Helpful when waiting for external systems to respond
- Allows other work to be done while waiting
- Prefect's *ConcurrentTaskRunner* replaces need for using Python's *async, await,* etc.

Concurrency

```python
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()
```

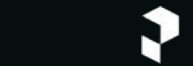

Concurrency

```
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

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



Task runners for parallelization

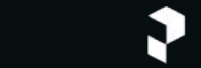
Parallelism

- 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 for parallelism

```python
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

Testing

Testing

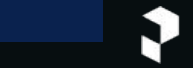
- Context manager for unit tests provided
- Run flows against temporary local SQLite db

```python
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
```


Testing

- Use in a Pytest fixture

```python
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

GitHub Action

```yaml
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

```python
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.

Cloud and server REST API interactive docs:

docs.prefect.io/latest/api-ref/rest-api

Prefect Runtime

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

```python
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)
```

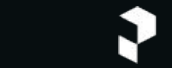

Upload data to AWS S3

Steps

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

pip install -U prefect-aws



Register new blocks

prefect blocks register -m prefect_aws

See block types & blocks from CLI

prefect block type ls

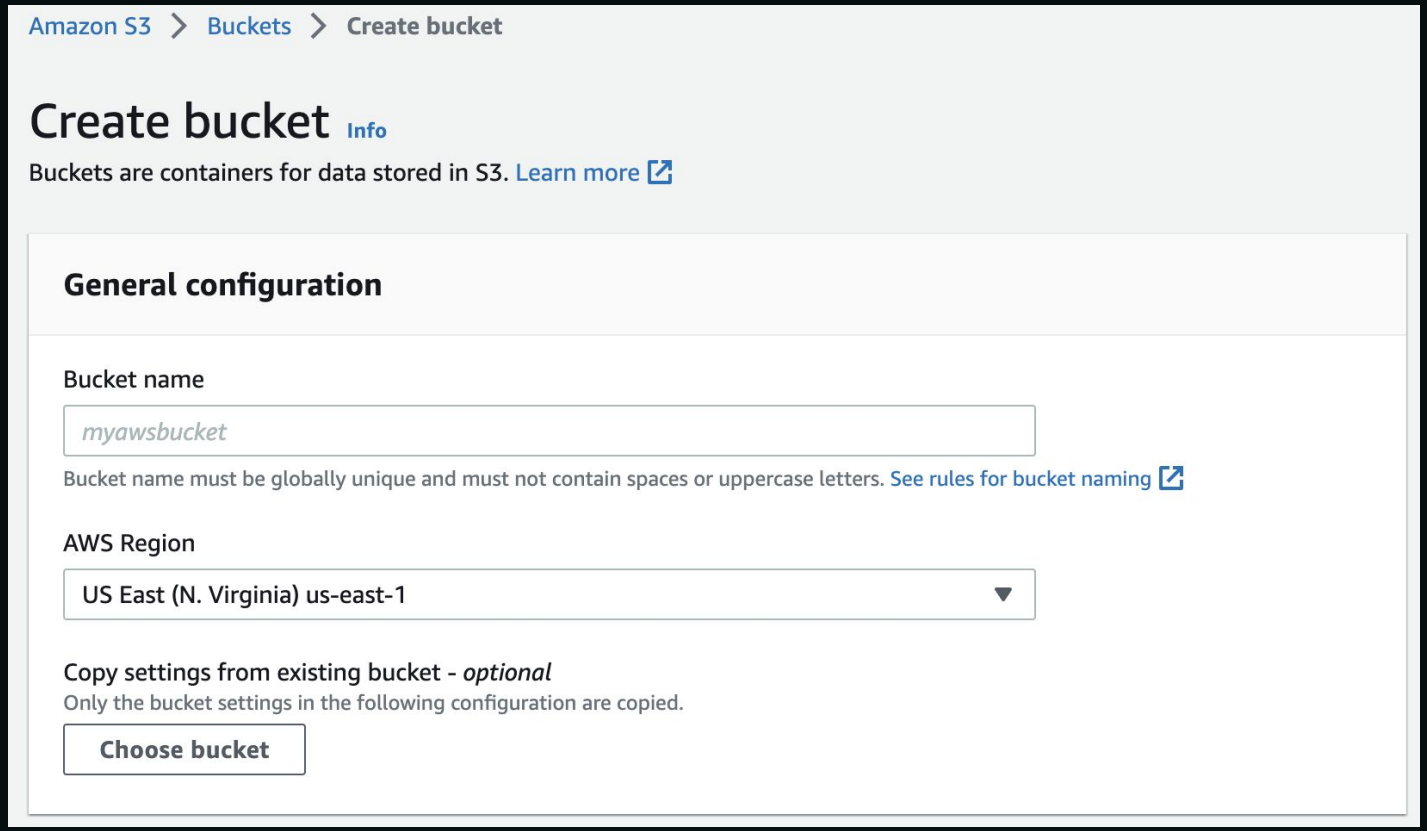
prefect block ls

Make an *S3Bucket* block

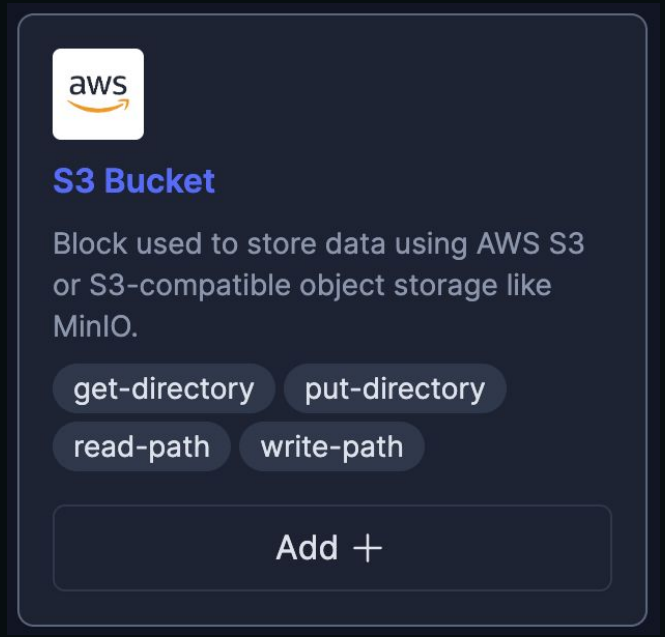
⚠️ ***S3Bucket* block from prefect-aws != *S3* block that ships with Prefect**

- Both block types upload and download data
- *S3Bucket* block has many methods
- We are showing how to use *S3*Bucket block

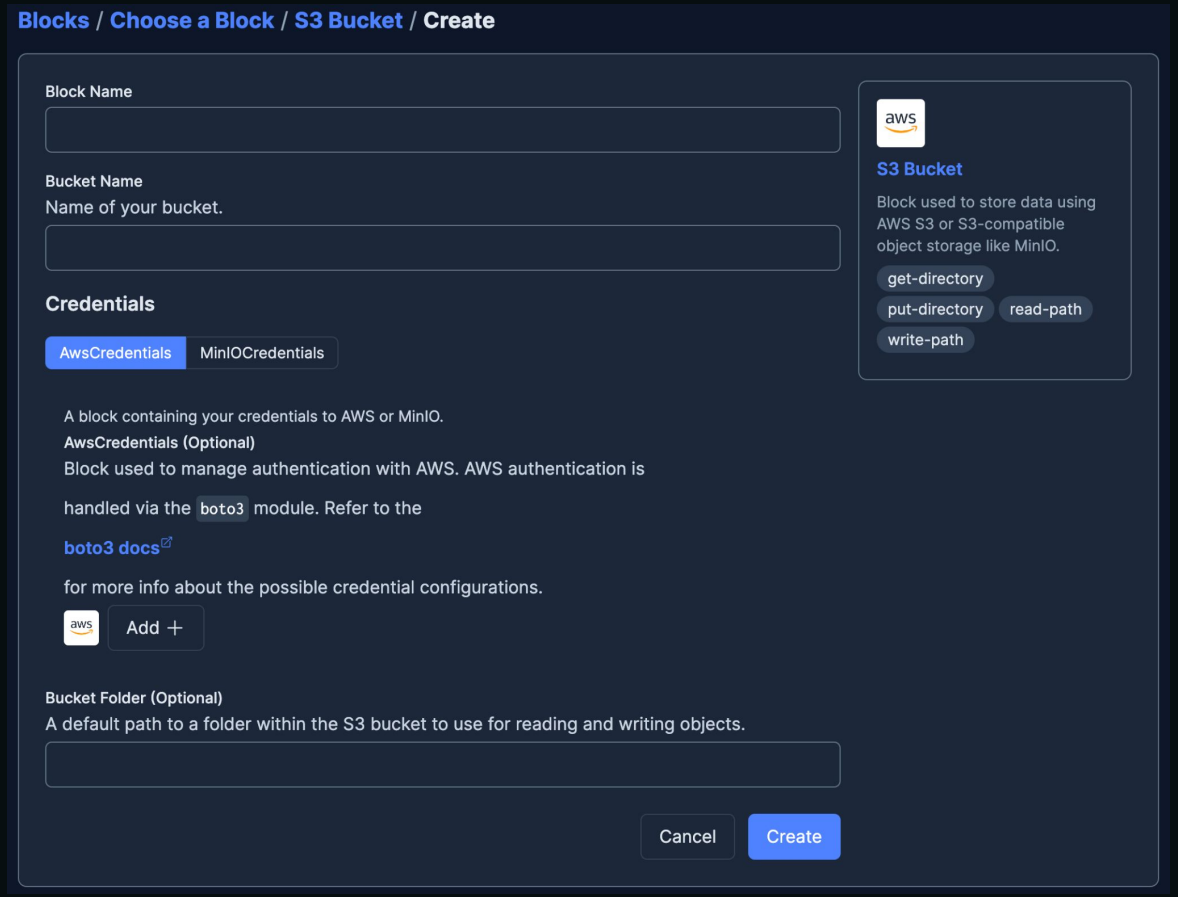
Create S3 Bucket



Create S3Bucket block from UI

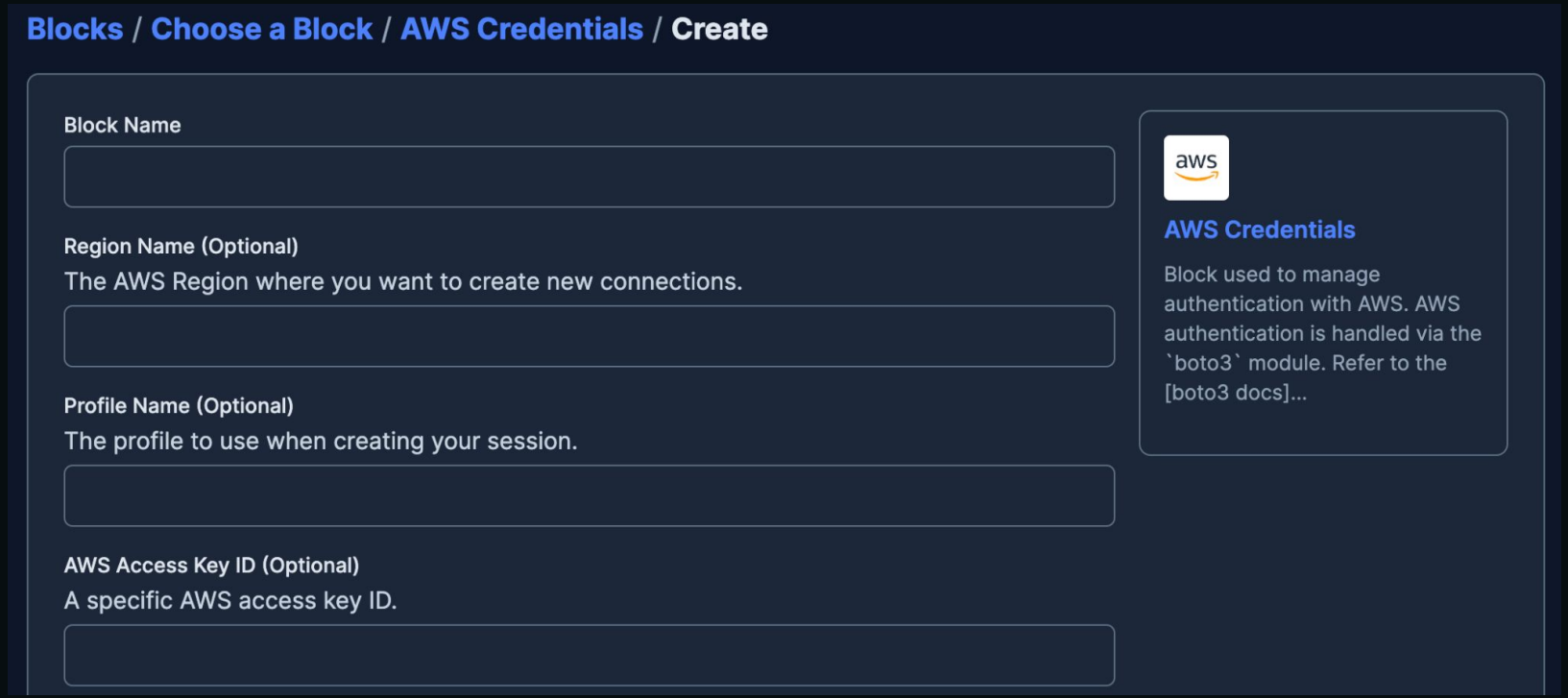


Create S3Bucket block from UI



AWS Credentials block from UI

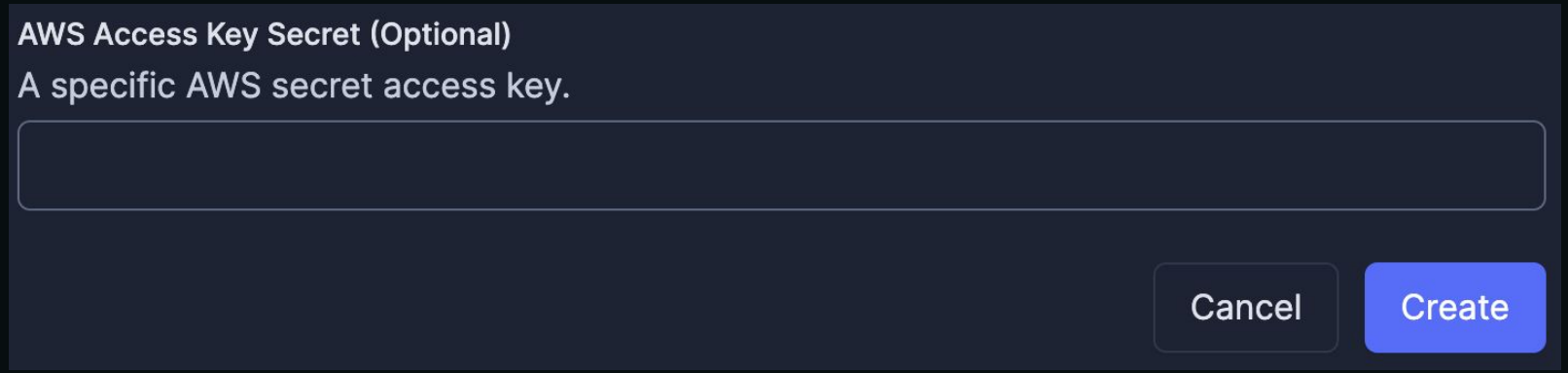
Use the nested AWS Credentials block as needed



AWS Credentials block from UI

Leave most fields blank.

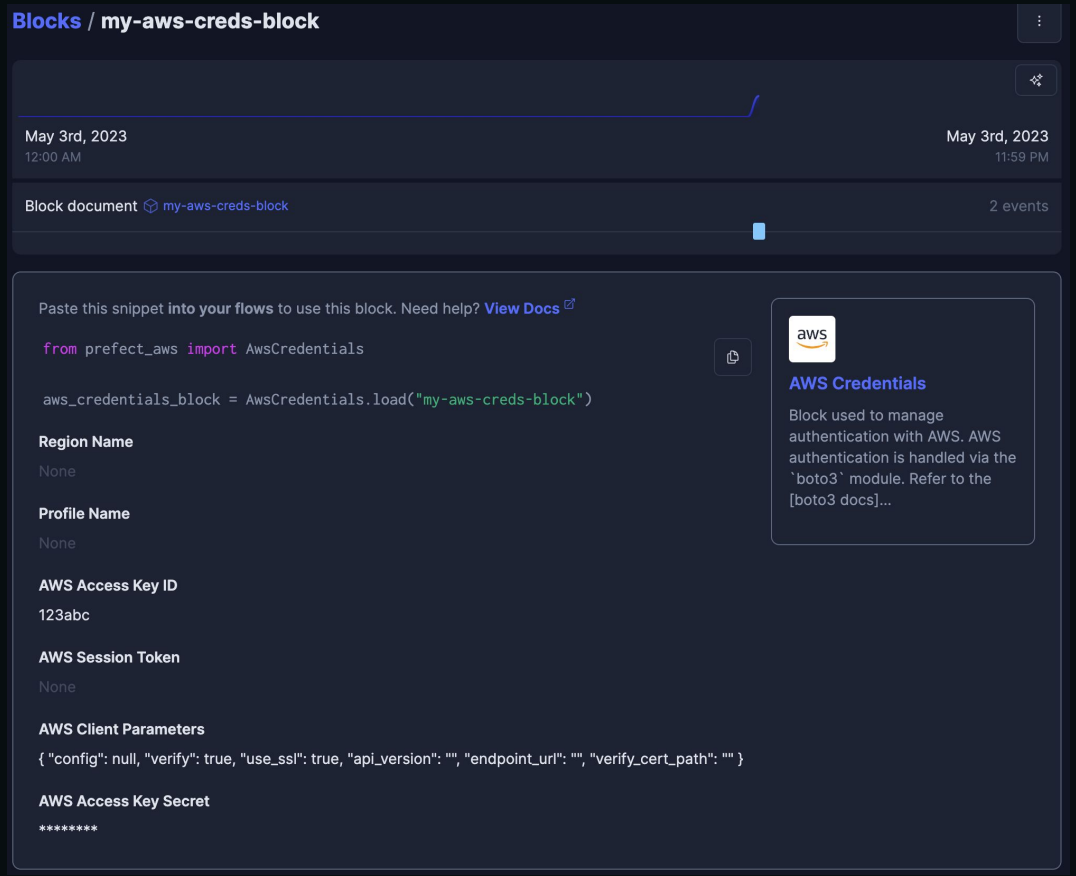
Probably use *AWS Access Key ID* & *AWS Access Key Secret*.



Or create blocks with Python code

```python
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



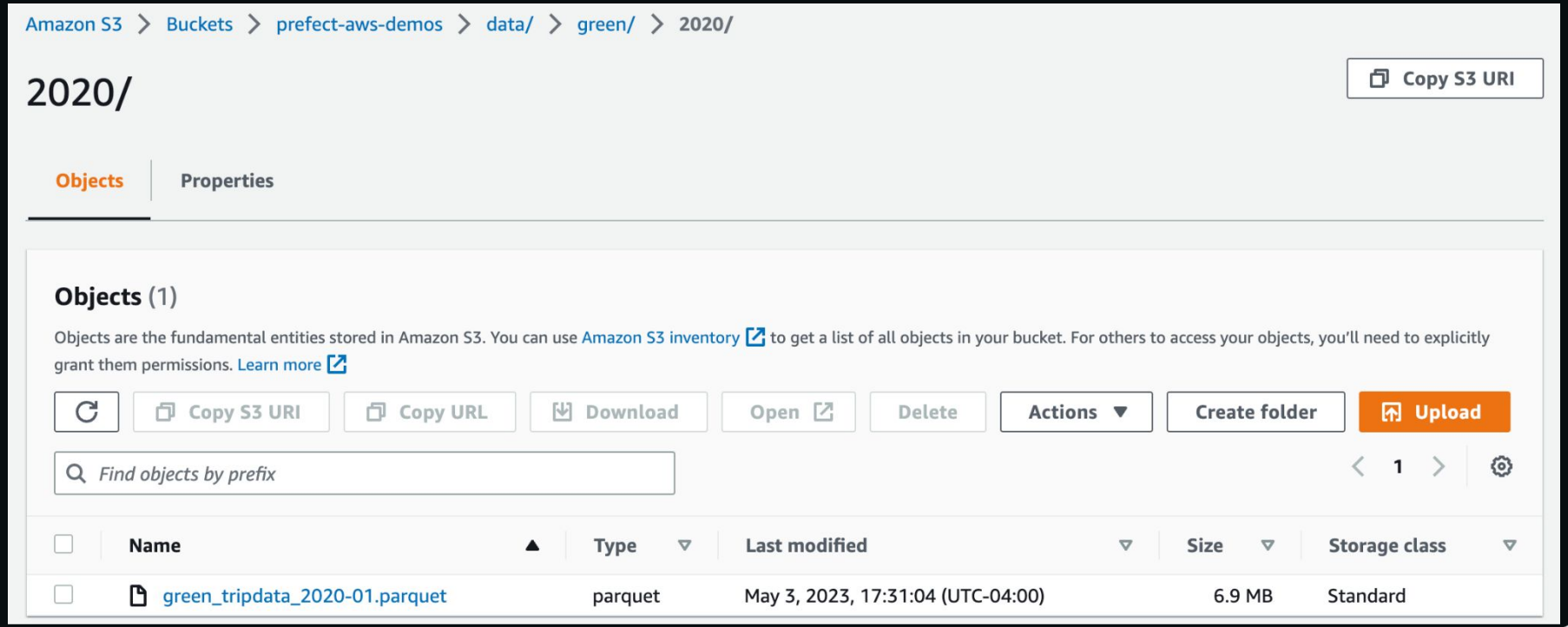
Flow code loads S3 block and uploads data file

```python
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)
```

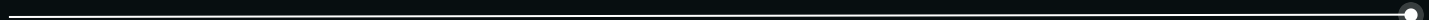
Use your flow code!

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

See file in S3 bucket



Additional deployments diagram



Prefect server instance

Prefect server instance

Alternative to Prefect Cloud: host your own Prefect server instance

- Backed by SQLite db by default
- Or use PostgreSQL in production
- Similar UI
- No events, push work pools, email server, authentication, user management, error summaries, etc.

Prefect server instance

- Switch to a new profile
- Use an ephemeral API (default) or set the API endpoint (required if in a Docker container)

Prefect server instance

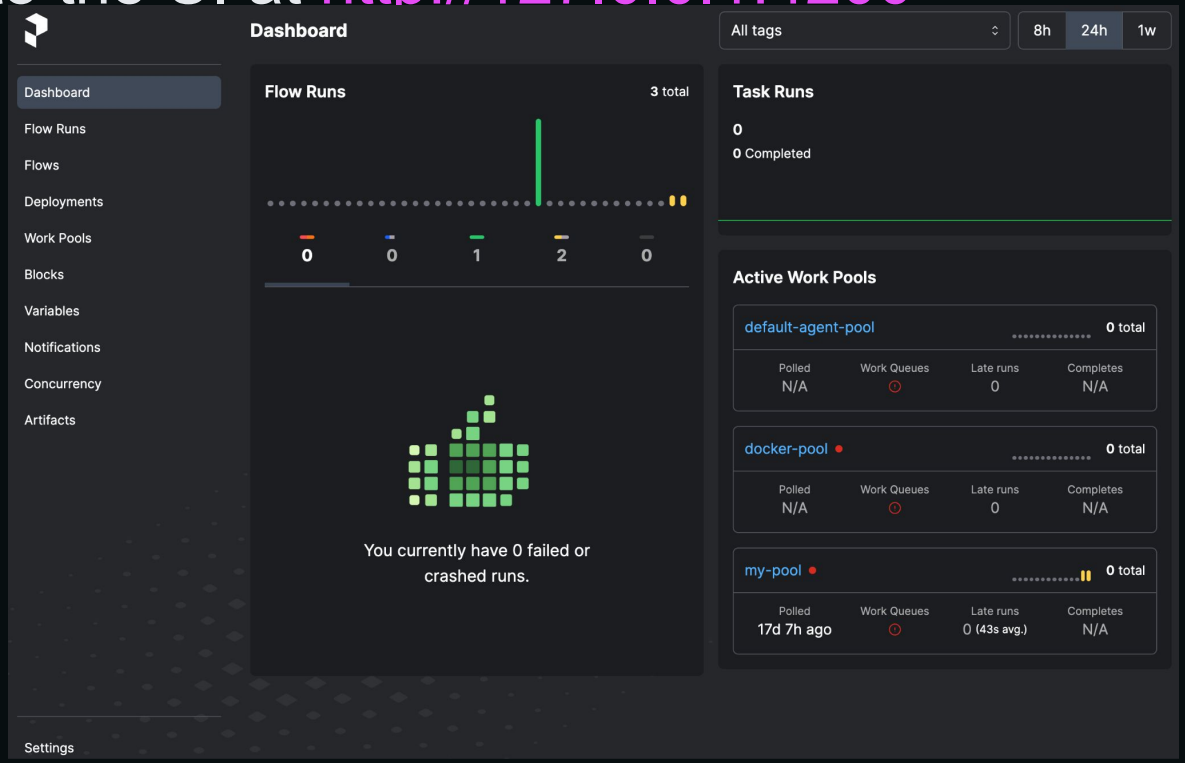
Start a server in another terminal with:

prefect server start

```
 ___ ___ ___ ___ ___ ___ _____
| _ \ _ \ __| __| __/ __|_   _|
|  _/   / _|| _|| _| (__  | |
|_| |_|_\___|_| |___\___| |_|

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
```


Prefect server instance

Head to the UI at http://127.0.0.1:4200



Prefect server instance

Required when running Prefect inside a container:

PREFECT_API_URL="http://127.0.0.1:4200/api"

See Prefect Helm Chart if running on Kubernetes
github.com/PrefectHQ/prefect-helm

Bonus content

PACC