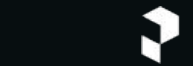
PACC

Prefect Associate
Certification Course

Part 2 agenda

- 104: Easily switch infrastructure and manage teams with work pool-based deployments
- 104 lab
- 105: Workflow design patterns: Compose workflows that meet your needs
- 105 lab
- 106: Interact with workflows, run tasks concurrently, create advanced automation triggers
- 106 lab
- Bonus: Misc topics



104 - Easily switch infrastructure and manage teams with work pool-based deployments

104 Agenda

- Create work pool-based deployments with *.deploy()*
- Run flows on Prefect's infrastructure with a Prefect Managed work pool
- Use a worker with a hybrid work pool for maximum control
  - Process
  - Docker
- Store your flow code
  - On GitHub
  - In a Docker image

Why use a work pool-based deployment?

Infrastructure is a pain, Prefect makes it better. 🙂

- Run workflows on a variety of dynamic infrastructure
- Provide a template for teams
- Scale infrastructure to 0 (serverless)
- Prioritize work

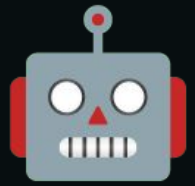
Create deployment with *.deploy()*

Very similar syntax to *.serve()*

Differences:

- Must specify work pool
- Must specify flow code storage source (or Docker image)
- Does **not** start a process watching for scheduled runs

Managed work pools



Prefect Managed work pools

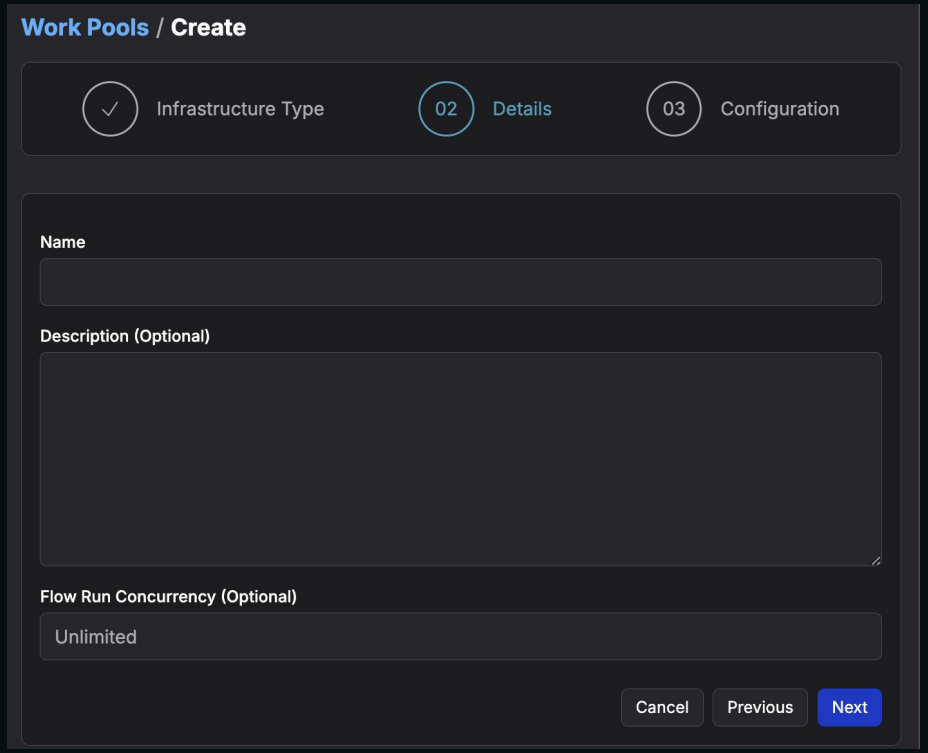
- Run workflows on Prefect’s infrastructure
- Cloud only
- Easy mode - no worker required
- Limitations
    - Compute hours
    - Concurrency
    - No custom Docker image

First work pool-based deployment

- Create deployment with *.deploy()*
- Specify flow code stored in a GitHub repository with *.from_source()*
- Use a **Prefect Managed** work pool

Create a Prefect Managed work pool

In the UI, **Work Pools -> + -> Prefect Managed**



Work pools

- Don’t modify the job template for now
- You can specify environment variables, etc.
- Work pools make it easier for data engineering platform teams to create guardrails for other teams

Deployment with Prefect Managed work pool

```python
from prefect import flow


if __name__ == "__main__":
    flow.from_source(
        source="https://github.com/prefecthq/pacc-2024-v6.git",
        entrypoint="102/weather2-tasks.py:pipeline",
    ).deploy(
        name="my-first-managed-deployment",
        work_pool_name="managed1",
    )
```


Run script to create the deployment

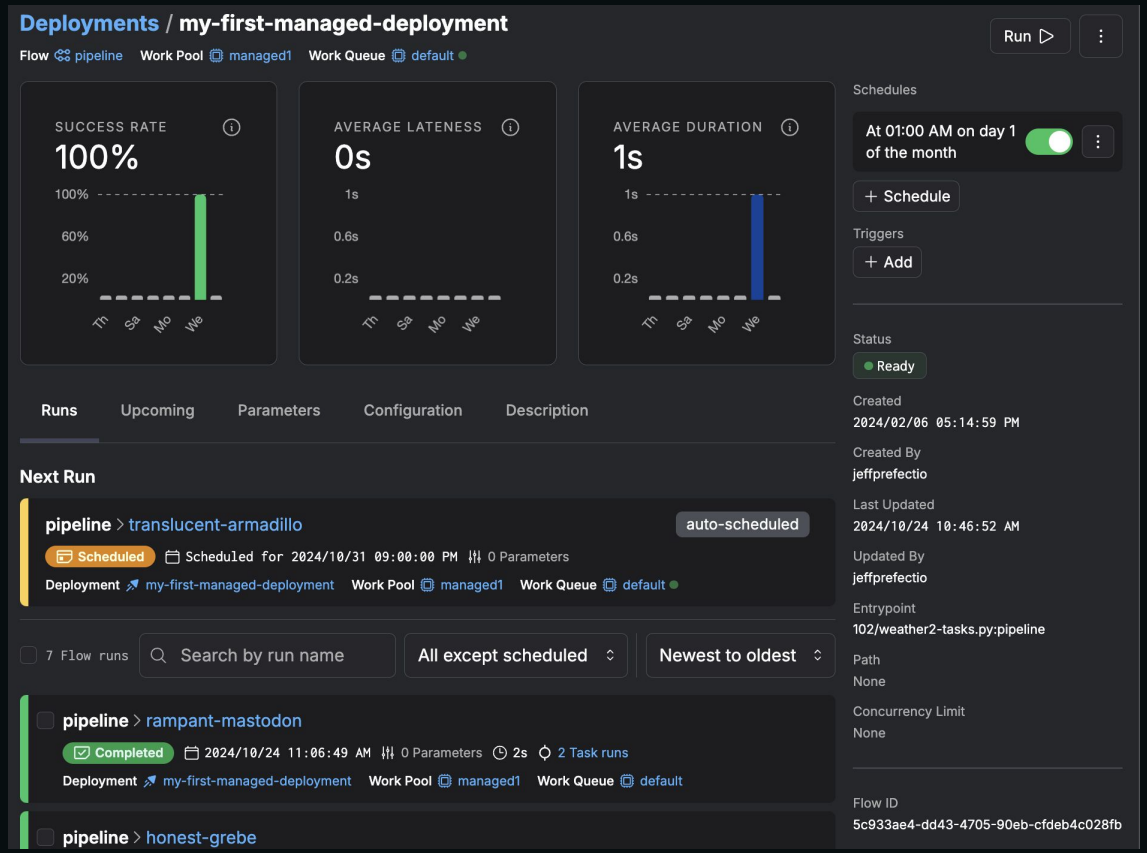
```
(base) jeffhale pacc-2024-v6/104 [main] $ python deploy-managed.py
Successfully created/updated all deployments!

                    Deployments
```

Name	Status	Details
pipeline/my-first-managed-deployment	applied	

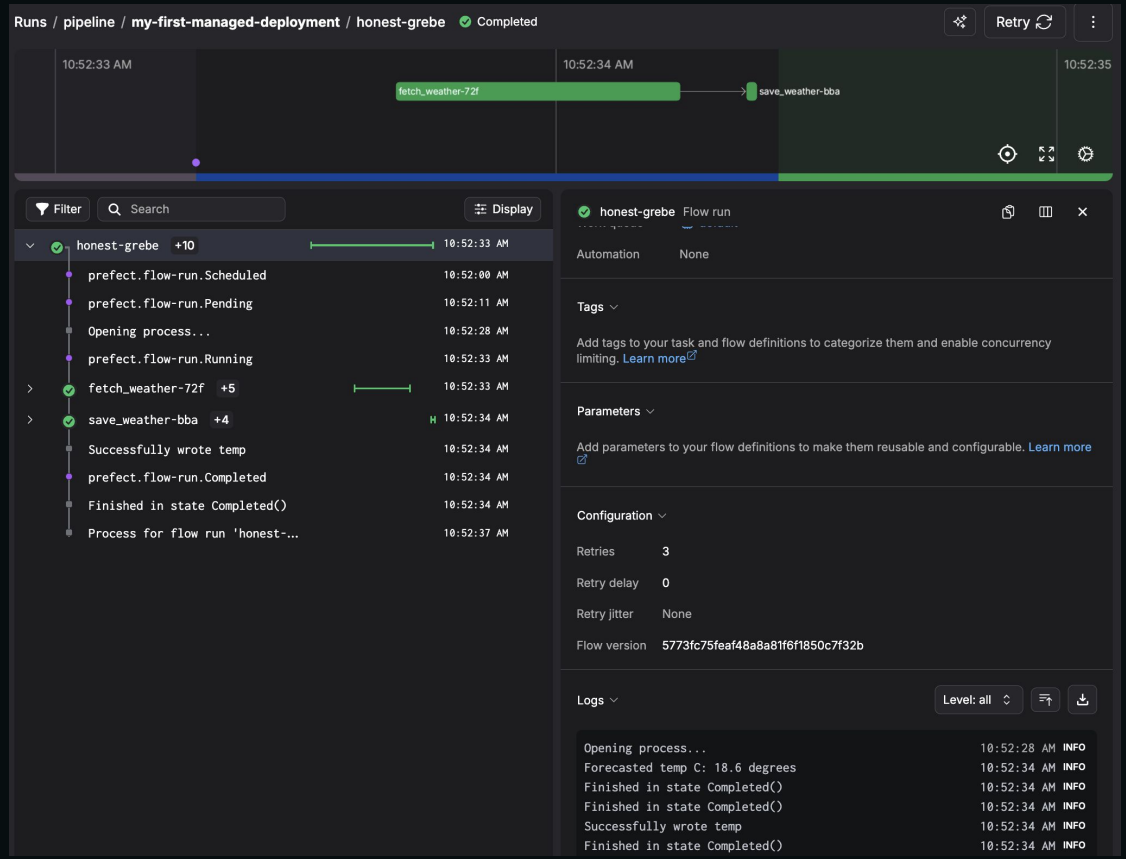
```
To schedule a run for this deployment, use the following command:

        $ prefect deployment run 'pipeline/my-first-managed-deployment'


You can also run your flow via the Prefect UI: https://app.prefect.cloud/account/9b649228-0419-40e1-9e0d-44954b5c0ab6/workspace/d137367a-5055-44ff-b91c-6f7366c9e4c4/deployments/deployment/d448be8f-2092-47f9-8d0b-ee06ce182480
```


See deployment details in the UI

1.



Run the deployment



At runtime, Prefect:

1. Pulls your flow code from GitHub
2. Runs your code in a Docker container on our infrastructure
3. Monitors and reports on state
4. Exits container and cleans up

Run the deployment

- Run state progression:
  Scheduled -> Pending -> Running -> Completed

- ⌚ Takes a moment to spin up Docker Container on our infrastructure

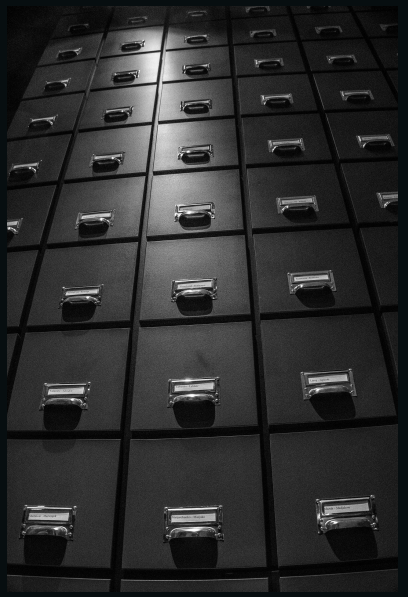
Let’s break this down

Work pools

Work pools

- Server side
- Provide default infrastructure configuration for deployments
- **👆Deployments that use this work pool inherit these settings**

Flow code storage



Flow code storage options

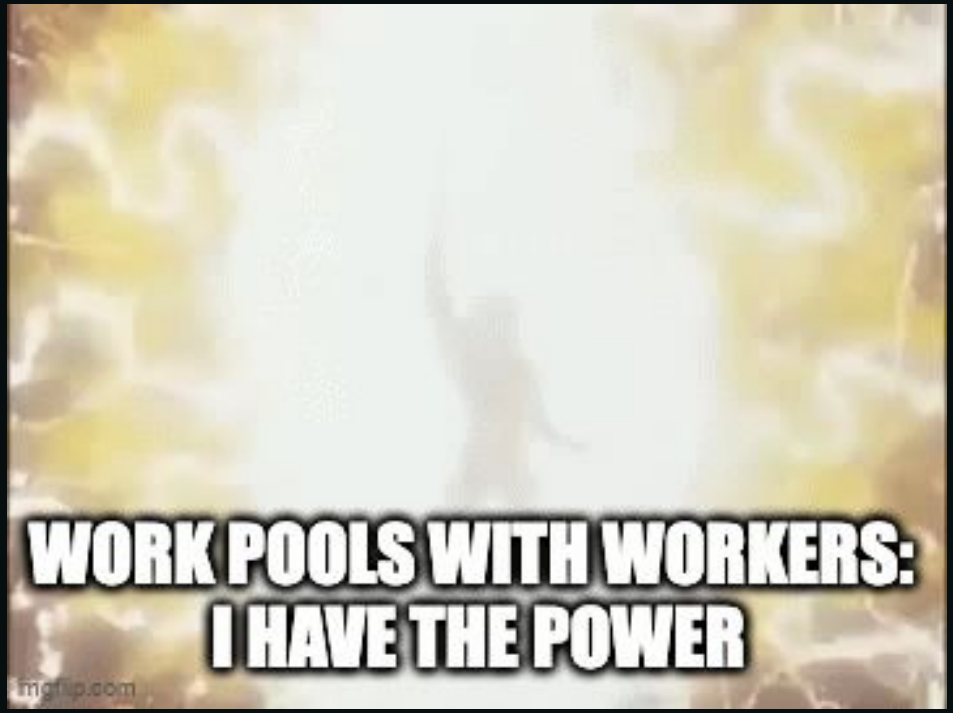
1. Local
2. Git-based remote repository (e.g. GitHub, GitLab)
3. Bake your code into a Docker image
4. Cloud provider storage

Flow code storage

- Specified public GitHub repo with *.from_source()* class method
- Call *flow.from_source()* or *flow_name.from_source()*
- Provide repo URL and *entrypoint path:flow function name*
- 🔐 Private repos are fine, just pass credentials

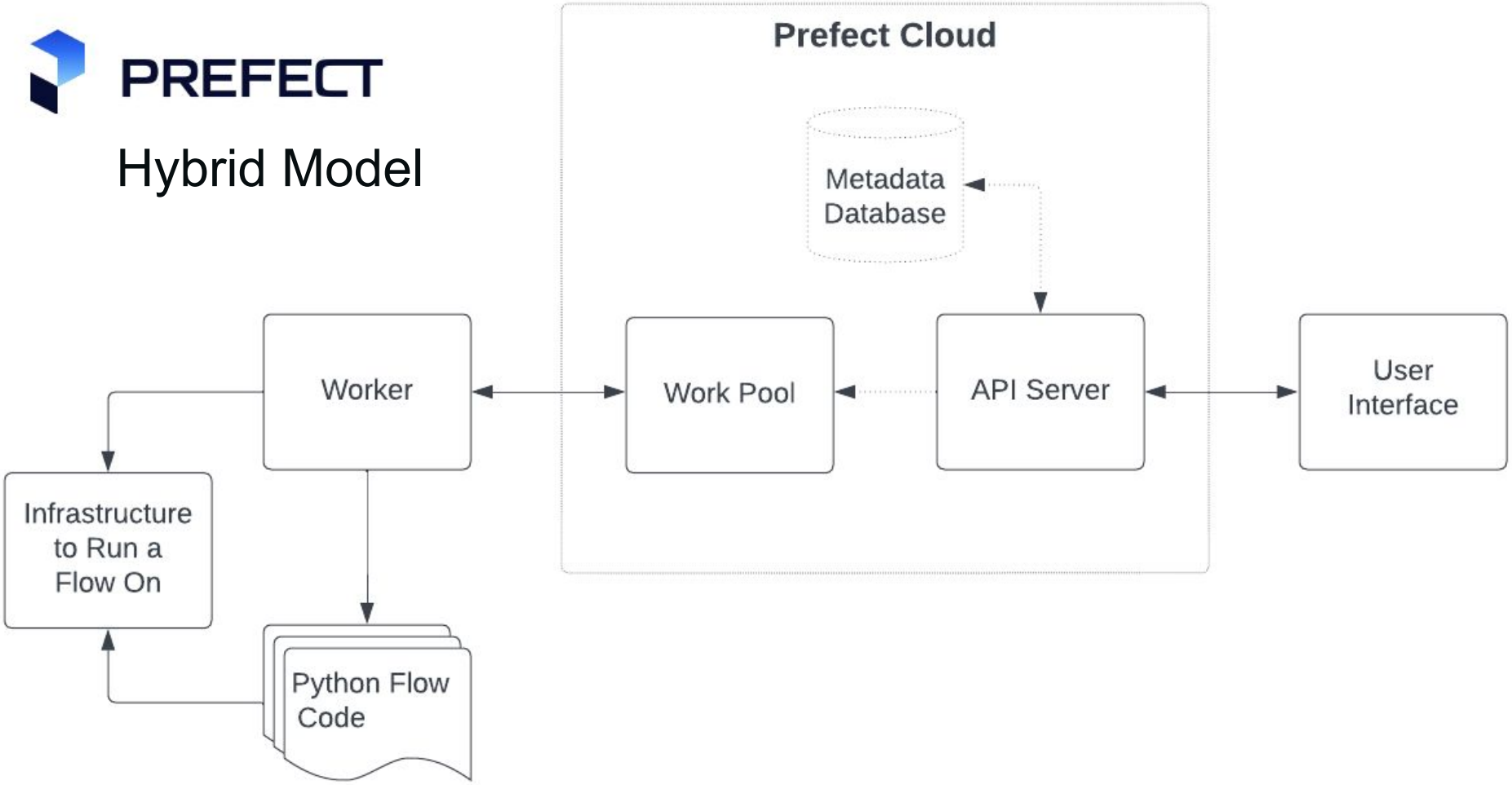
Hybrid model - hybrid work pools with workers

Hybrid work pools with workers



PREFECT

Hybrid Model



Hybrid model = separation

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

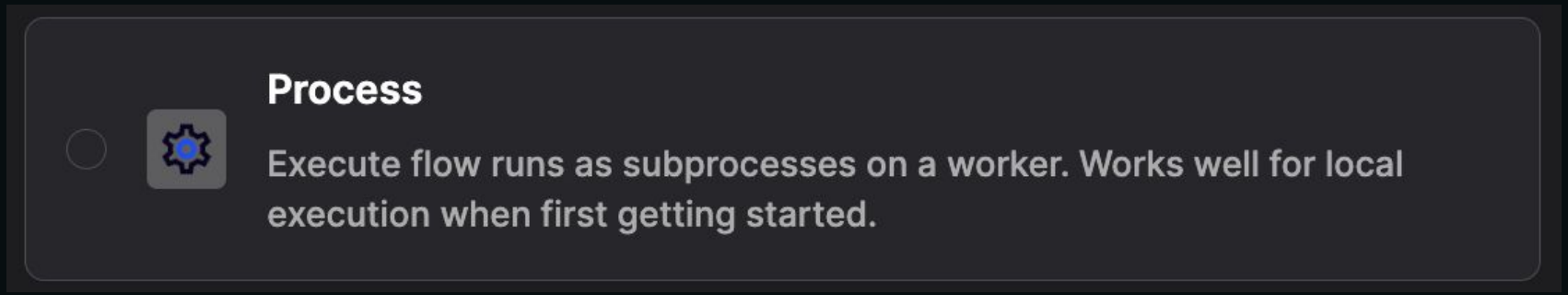
https://www.prefect.io/security

Example: Process work pool & worker

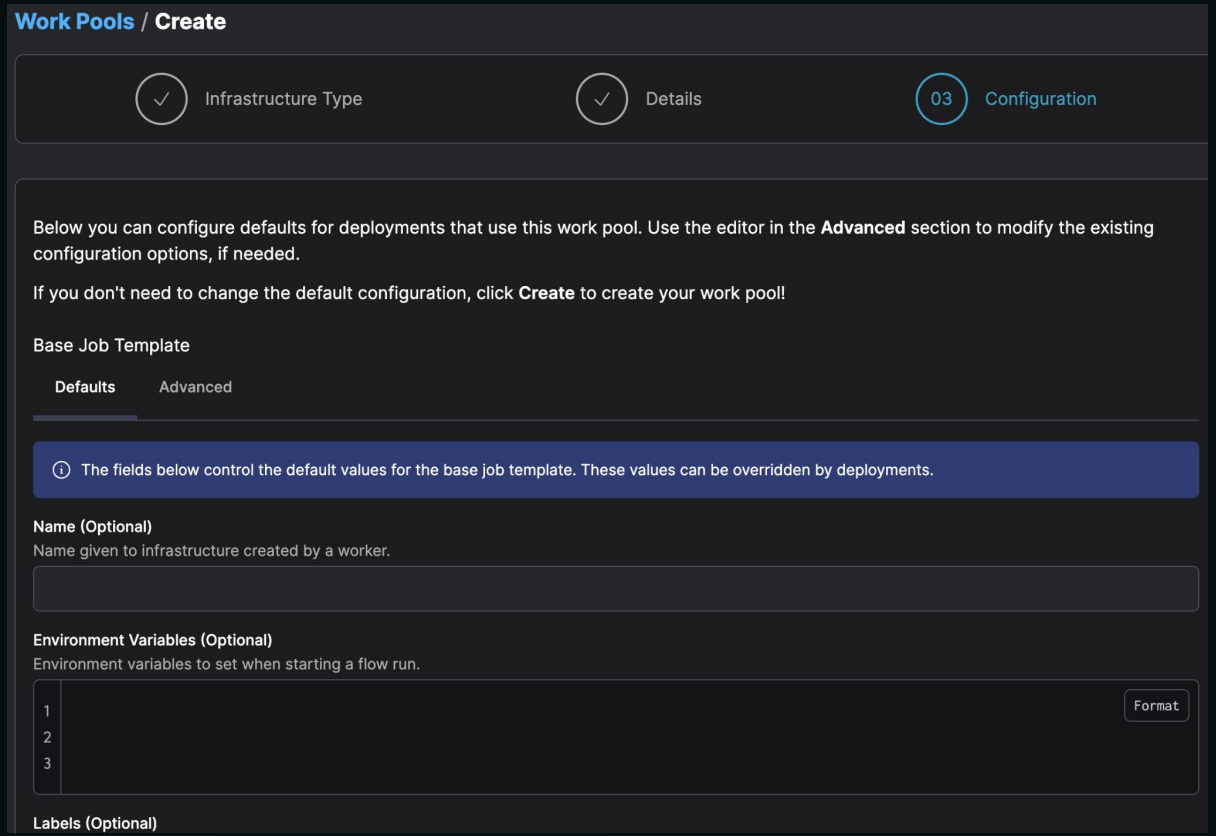
First hybrid work pool-based deployment

- Create with *.deploy()*
- Specify flow code stored in a GitHub repository with *.from_source()*
- Use a **Process** work pool
- Start a worker to pick up scheduled flow runs ⭐

Create a Process work pool



Create a Process work pool



Deployment creation code with *.deploy()*

```python
from prefect import flow


@flow(log_prints=True)
def my_flow(name: str = "World"):
    print(f"Hello {name}!")


if __name__ == "__main__":
    my_flow.from_source(
        source="https://github.com/PrefectHQ/pacc-2024-v6.git",  # code stored in GitHub
        entrypoint="104/local-process-deploy-remote-code.py:my_flow",
    ).deploy(
        name="pacc-local-process-deploy-remote-code",
        work_pool_name="pacc-process-pool",
    )
```


Run script to create the deployment

```
Successfully created/updated all deployments!

                                Deployments
```

Name	Status	Details
my-flow/pacc-local-process-deploy-remote-code	applied	

```
To schedule a run for this deployment, use the following command:

        $ prefect deployment run 'my-flow/pacc-local-process-deploy-remote-code'


You can also run your flow via the Prefect UI: https://app.prefect.cloud/account/9b649228-0419-40e1-9e0d-44954b5c0ab6/workspace/d137367a-5055-44ff-b91c-6f7366c9e4c4/deployments/deployment/1cff0894-6beb-46d3-9530-6f9165b8b836
```


Start a worker

- In a new terminal window
- Watches for scheduled flow runs in the work pool

prefect worker start --pool 'pacc-process-pool'

Run the deployment using the CLI

Just like running a deployment with *.serve*

prefect deployment run 'my-flow/pacc-local-process-deploy-remote-code'

See flow run logs in the UI or worker's terminal window

```
12:43:34.930 | INFO    | prefect.flow_runs.worker - Worker 'ProcessWorker c7a72edc-47
56-4238-8083-bd615c763c60' submitting flow run '6e1140ff-7155-4288-8f8c-7d5ba0676c33'
12:43:35.872 | INFO    | prefect.flow_runs.worker - Opening process...
12:43:36.019 | INFO    | prefect.flow_runs.worker - Completed submission of flow run
'6e1140ff-7155-4288-8f8c-7d5ba0676c33'
12:43:38.318 | INFO    | prefect.deployment - Cloned repository 'https://github.com/P
refectHQ/pacc-2024-v6.git' into 'pacc-2024-v6'
12:43:38.670 | INFO    | Flow run 'cryptic-ringtail' - Hello World!
12:43:38.813 | INFO    | Flow run 'cryptic-ringtail' - Finished in state Completed()
```


At runtime:

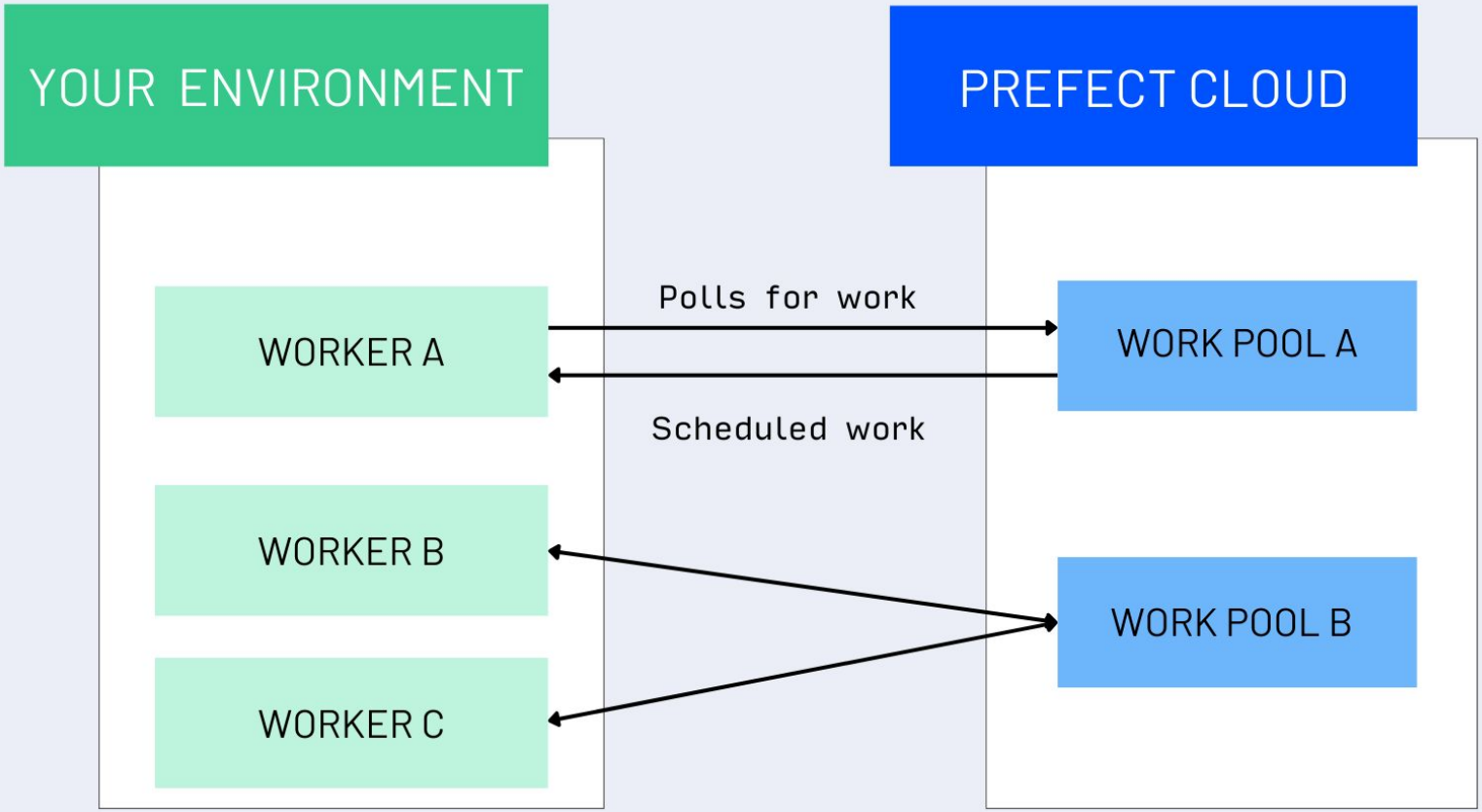
1. Worker kicks off scheduled flow run
2. Pulls flow code from GitHub
3. Runs code in a local subprocess
4. Prefect monitors state
5. Subprocess exits

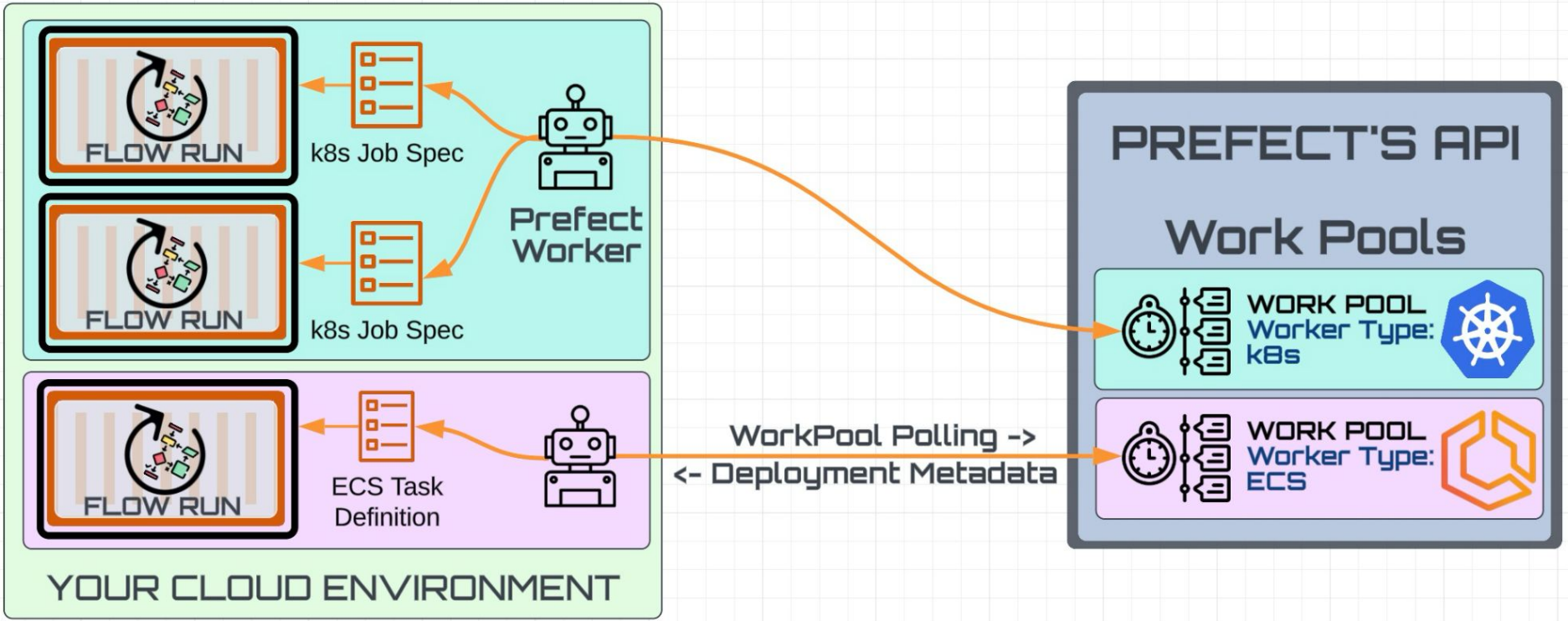
Workers

Workers

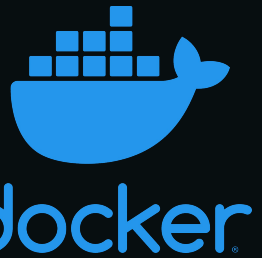
- Long-running process on the client
- Poll for scheduled flow runs from work pools
- Must match a work pool to pick up work
- If familiar with agents (old concept), workers are like smarter, typed agents

WORKERS & WORK POOLS




FLOW RUN
FLOW RUN
k8s Job Spec
k8s Job Spec
Prefect Worker
FLOW RUN
ECS Task Definition
YOUR CLOUD ENVIRONMENT
WorkPool Polling ->
<- Deployment Metadata
PREFECT'S API
Work Pools
WORK POOL
Worker Type:
k8s
WORK POOL
Worker Type:
ECS

Example: Docker work pool & worker



Why use Docker?

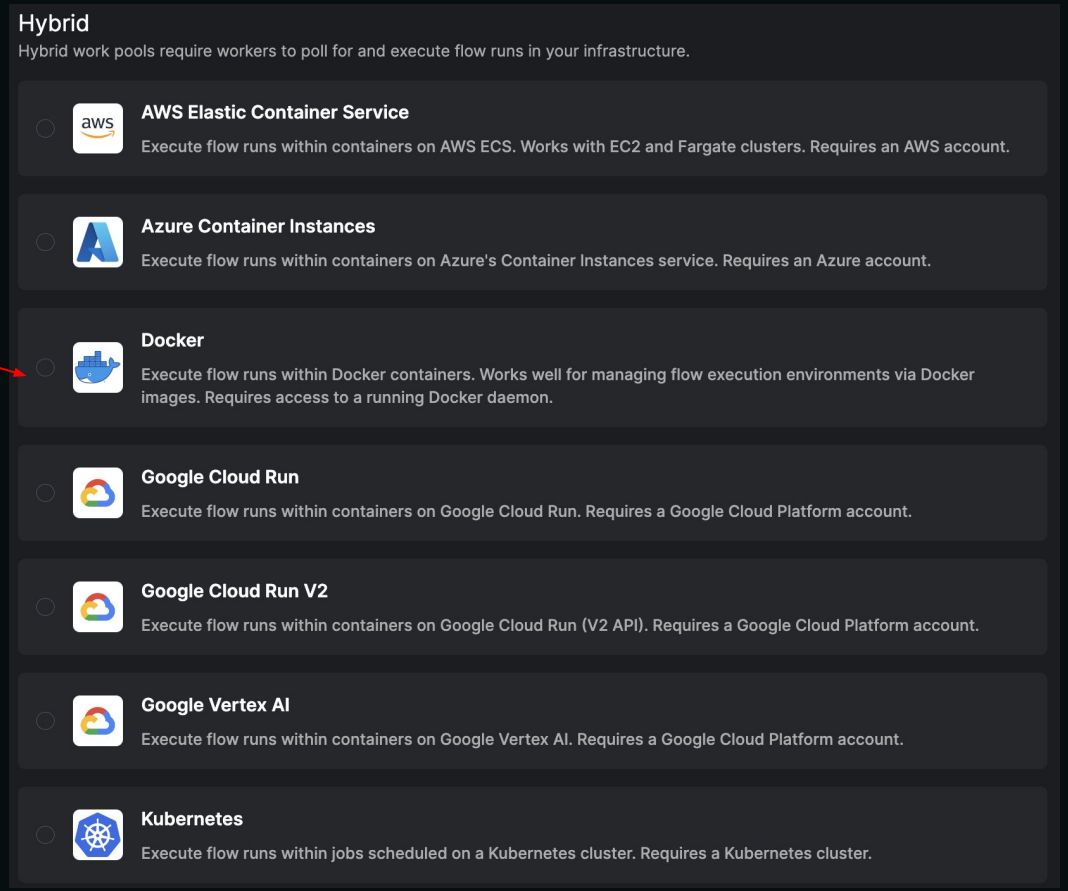
- Same operating environment everywhere
- Lighter weight than a VM
- Linux (generally)
- Portable
- Very popular
- All Prefect work pools other than Process use it

Docker work pool

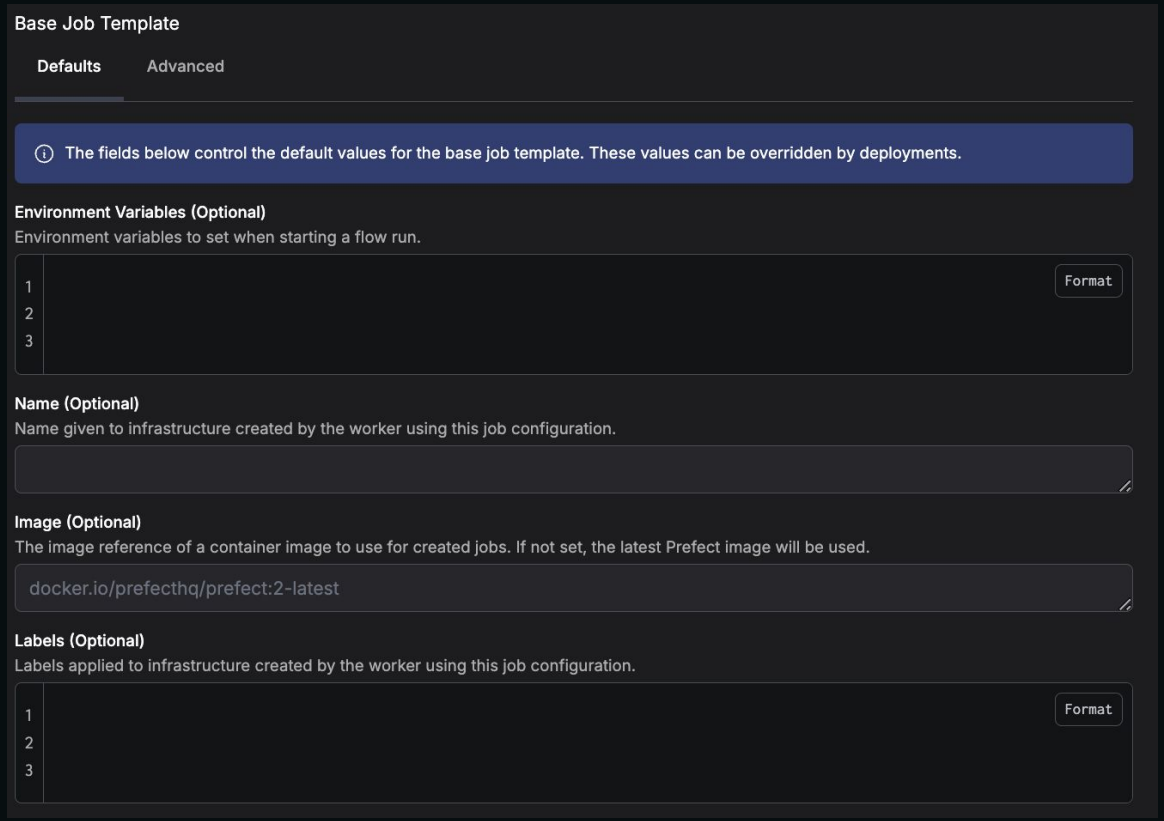
Run a flow in a Docker container

1. Start Docker on your machine
2. Create a Docker type work pool
3. Start a worker that polls the work pool
4. Create a deployment that specifies the work pool
5. Run the deployment
6. Auto spins up a Docker container & runs flow in it

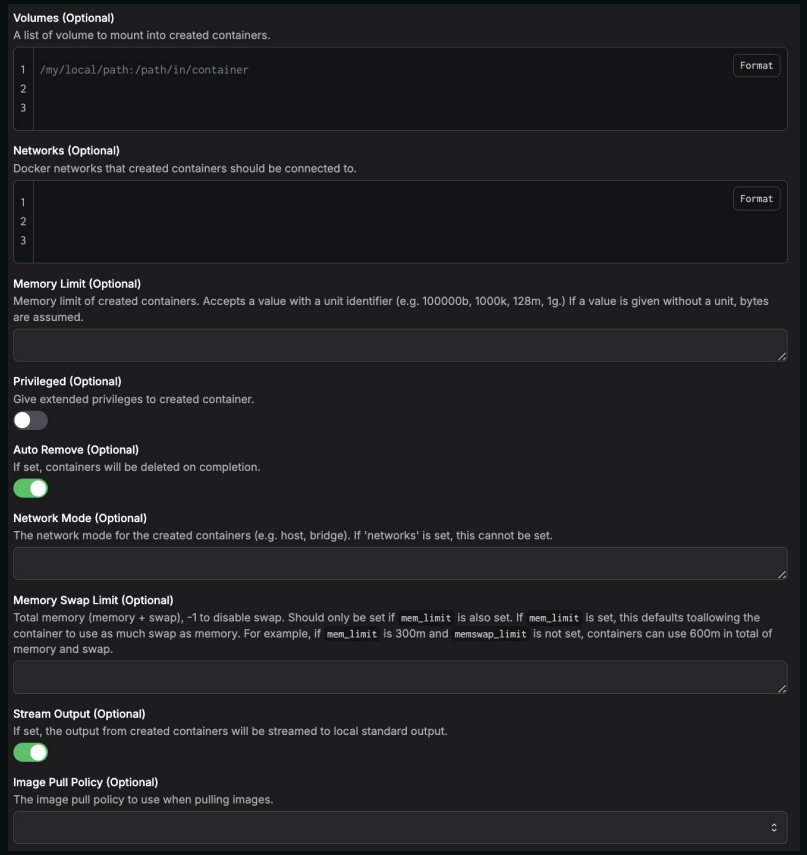
Create a Docker work pool



Docker work pool - base job template



Docker work pool - base job template



Package flow code into a Docker image with *.deploy()*

```python
from prefect import flow


@flow(log_prints=True)
def buy():
    print("Buying securities")


if __name__ == "__main__":
    buy.deploy(
        name="my-code-in-an-image-deployment",
        work_pool_name="my-docker-pool",
        image="discdiver/local-image:1.0",
        push=False,
    )
```

❗ *.from_source()* method not needed if baking flow code into image

.deploy() method

Creates a Docker image with your flow code baked in by default!

- Specify the image name
- Specify *push=False* to not push image to registry
- Best practice: create a *requirements.txt* file with pinned package versions to install into image

Docker type worker

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

prefect worker start -p my-docker-pool

If you want to make sure you have the packages needed:

prefect worker start -p my-docker-pool --install-policy always

Dockerfile used to create your image (under the hood)

```
FROM prefecthq/prefect:3-latest
COPY requirements.txt /opt/prefect/104/requirements.txt
RUN python -m pip install -r requirements.txt
COPY . /opt/prefect/pacc-2024-v6/
WORKDIR /opt/prefect/pacc-2024-v6/
```


Docker

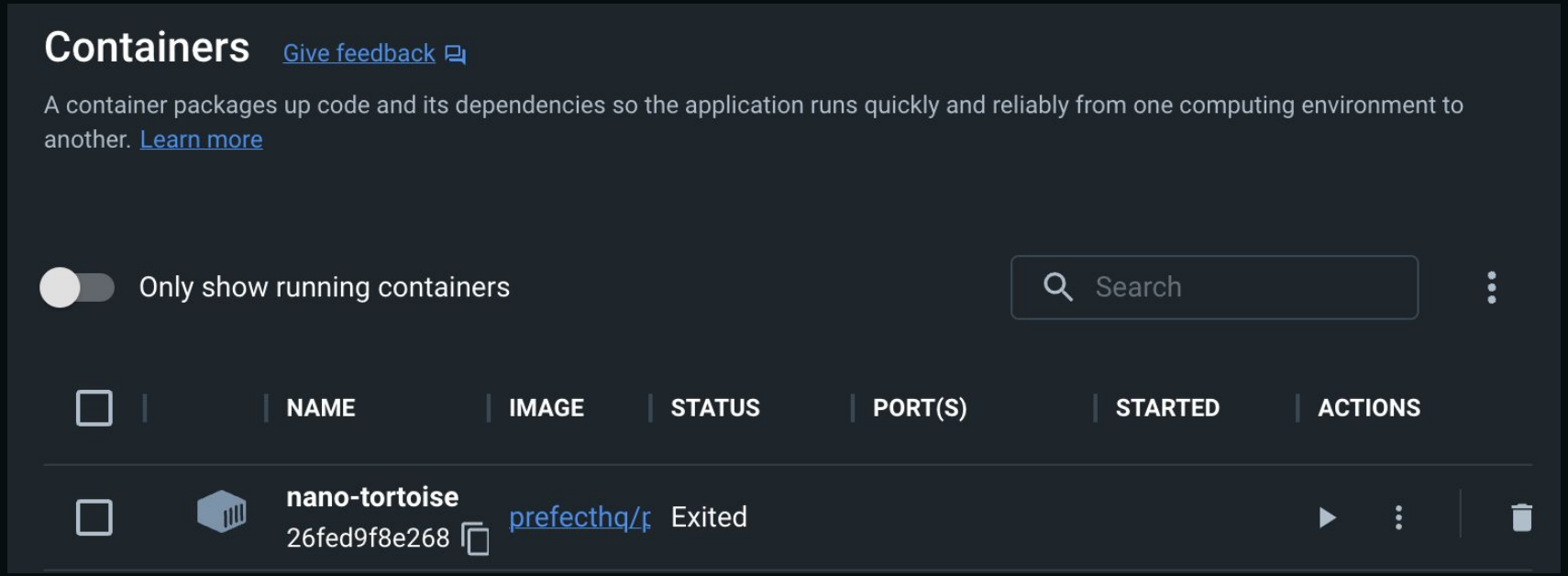
- Prefect provides base Docker images
- You can customize base image

Docker

- Run your deployment
- Worker pulls image and spins up Docker container
- Flow code runs in Docker container and exits 🚀

Docker

See container in Docker Desktop if running locally



Docker

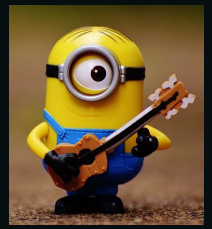
Reminders:

- Docker *installed* & **running**
- *prefect-docker* package installed
- Start a worker to poll for scheduled runs

Hybrid work pool types

1. Process (local subprocess)
2. Docker
3. Serverless options such as ECS, ACI, GCR, VertexAI
4. Kubernetes

* Worker required for all



Push work pools 📌

Push work pools

Serverless. No worker required.

- AWS ECS
- Google Cloud Run
- Azure Container Instances
- Modal

Push work pools

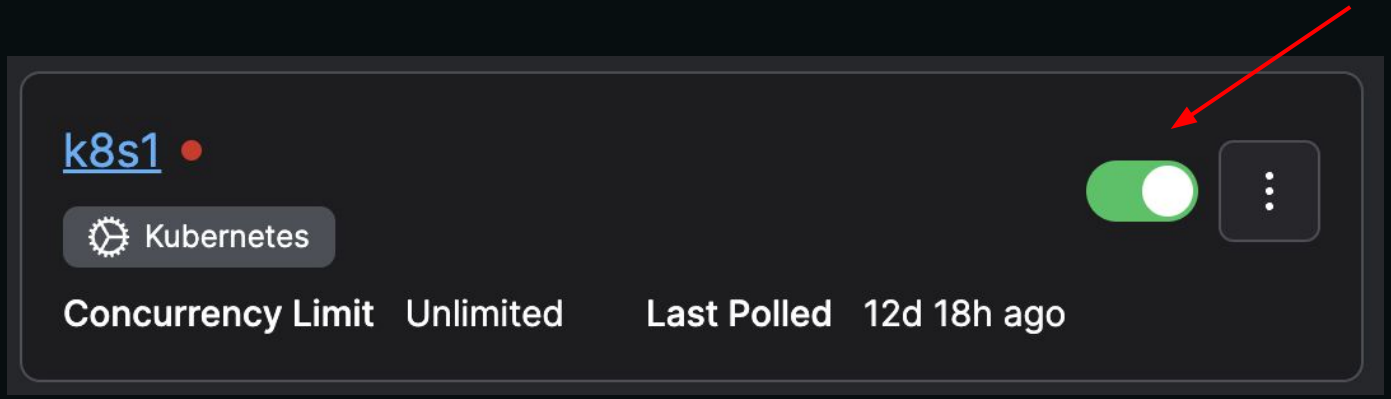
Prefect will create everything for you with *--provision-infra*

Prerequisites:

- Cloud provider account
- CLI tool installed
- Authenticated locally

prefect work-pool create --type modal:push --provision-infra my-modal-pool

Pause **scheduled runs** for work pools from UI (or CLI)



104 Recap

You've seen how to

- Create work-pool based deployments! 🎉
- Use the hybrid model with workers
- Bake flow code into Docker images
- Run flows on a variety of infrastructure
- Pause and resume work pools

Lab 104

Lab norms for breakout rooms

1. 🙂 Introduce yourselves
2. 🎥 Camera on (if possible)
3. 💻 One person shares screen
4. 🧑‍💻 Everyone codes
5. 🙋 Ask a question if you don’t follow something
6. 😌 Low-pressure, welcoming environment: lean in

104 Lab

- Let’s make one of our weather forecast workflows more powerful
- Create a deployment with *.deploy()* that uses a **Prefect Managed** or **Process** work pool. Reminder, **Managed** is Prefect Cloud only
- Create work pool from the UI
  - Create a deployment that references flow code stored in your own GitHub repository
    - Use your earlier fetch weather flow if you like
    - ❗ Push your code to your GitHub repo manually
  - If using a **Process** work pool start a worker to pick up scheduled flow runs
  - Run it! 🚀

104 Lab Extensions

Stretch 1: Pause and resume the work pool from the UI.

Stretch 2: Experiment with adjusting fields in a work pool base job template.

Stretch 3: If you have Docker installed:

Create a deployment where you bake your flow code into a Docker image with *.deploy()*.

- Don’t push the image (or log in + push to DockerHub).

Don’t forget to:

- Start Docker on your machine
- Create a Docker work pool

105 - Workflow design patterns: Compose workflows that meet your needs

Quick Recap

How Prefect works - Architecture Overview

Your Laptop

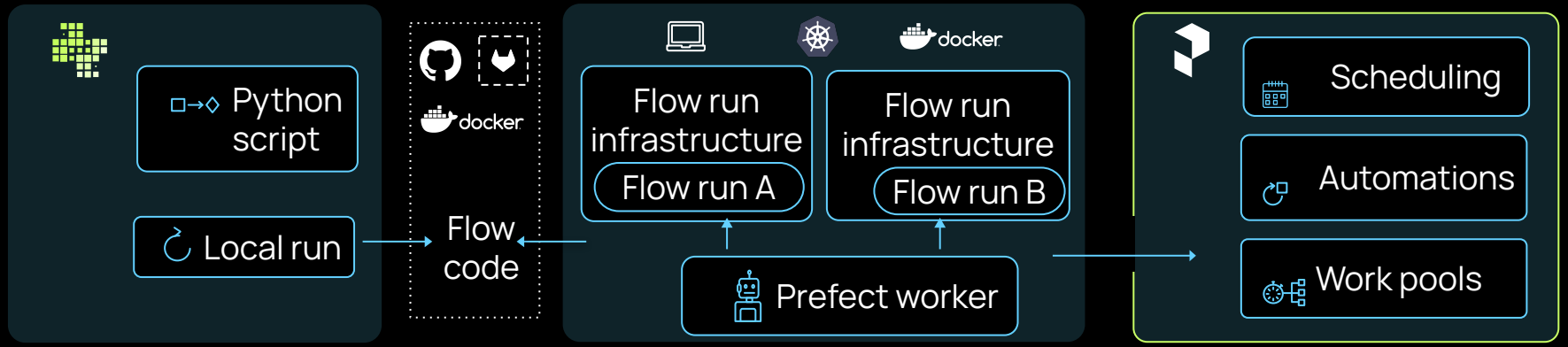
Write data workflow logic through Prefect's Python SDK

Your Execution Environment

Run data workflows on any infrastructure environment

Prefect Cloud

Understand & manage with the Prefect dashboard



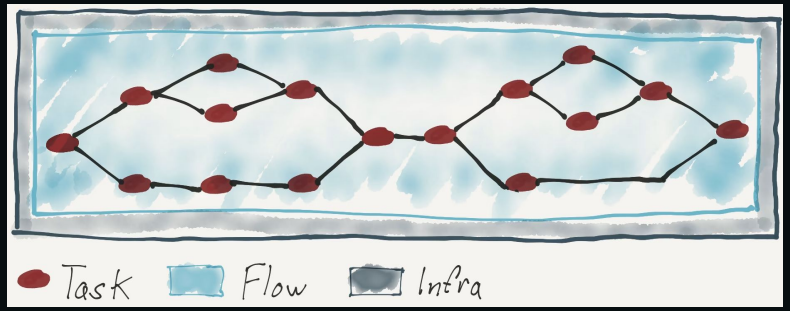
105 Agenda

Workflow pattern archetypes

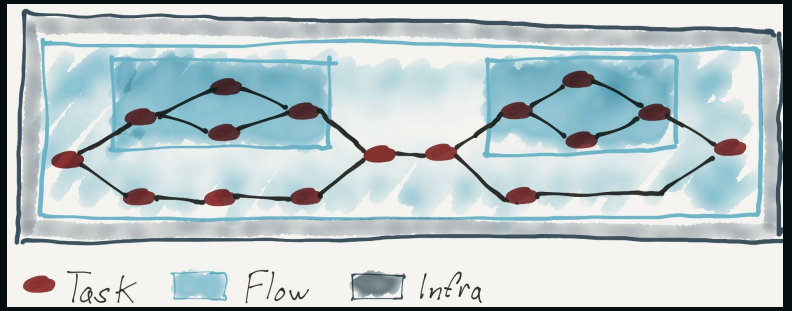
- Monoflow
- Subflows
- *run_deployment*
- Event-driven
  - Deployment triggers
  - Custom events
  - Webhooks
- Tasks alone

Workflow patterns

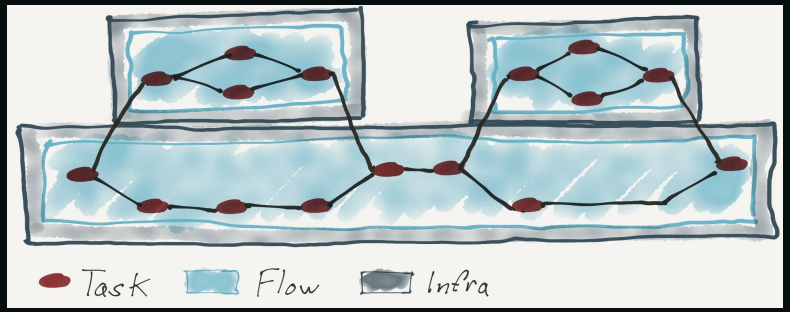
Workflow patterns - based on prefect.io/blog/workflow-design-patterns



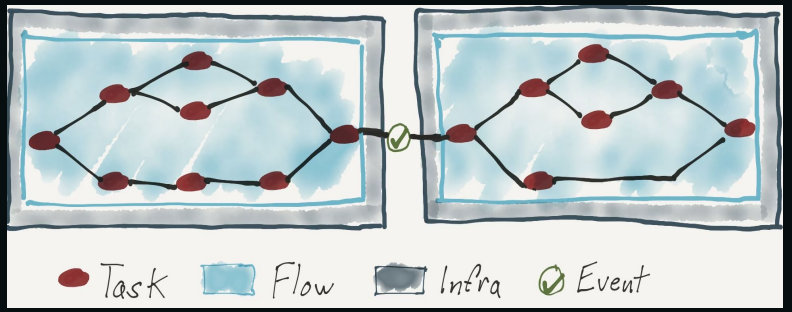
Flow of tasks



Flow of nested flows



Flow of deployments



Event-triggered flow

When to use which?

Pattern	Conceptual	Execution	Awaremeness
Flow of tasks	Coupled	Coupled	Coupled
Flow of nested flows	Separate	Coupled	Coupled
Flow of deployments	Separate	Separate	Coupled
Event-triggered flow	Separate	Separate	Separate

UI/
Organization

Infra/
Process

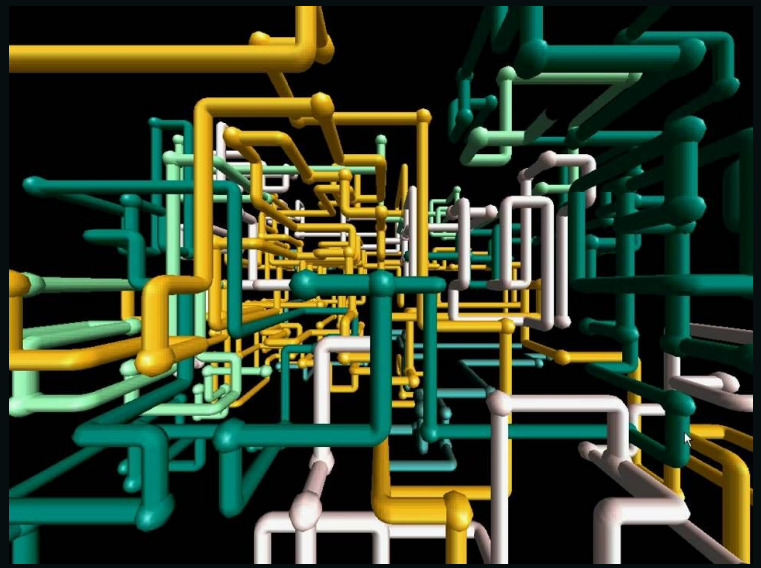
Context/
State

Nested flows

(AKA subflows)

Nested flow

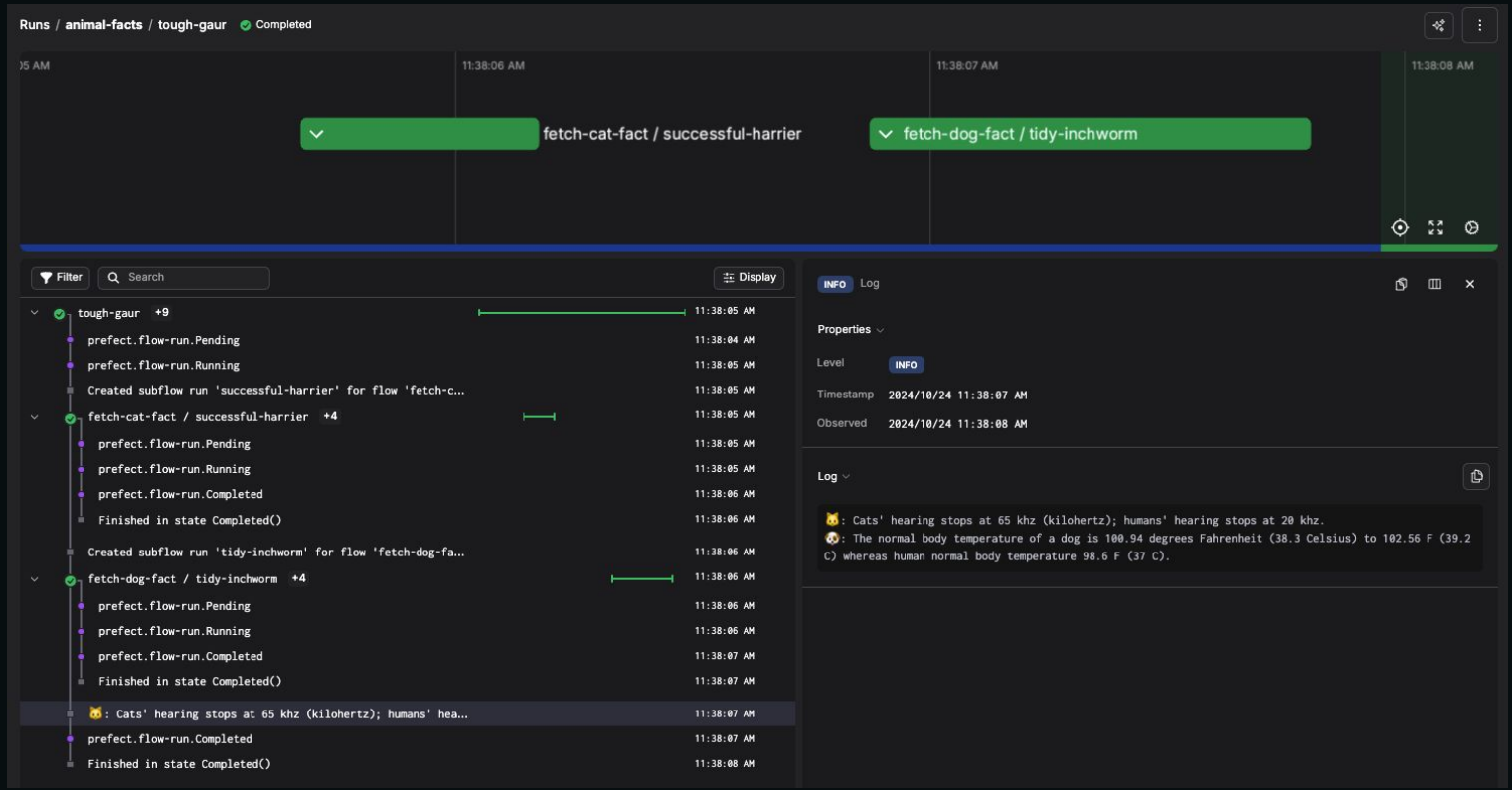
- A flow called from another flow
- Useful for grouping related tasks



Nested flows

```python
import httpx
from prefect import flow


@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}")


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

Timeline view



run_deployment

to create a flow of deployments

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.

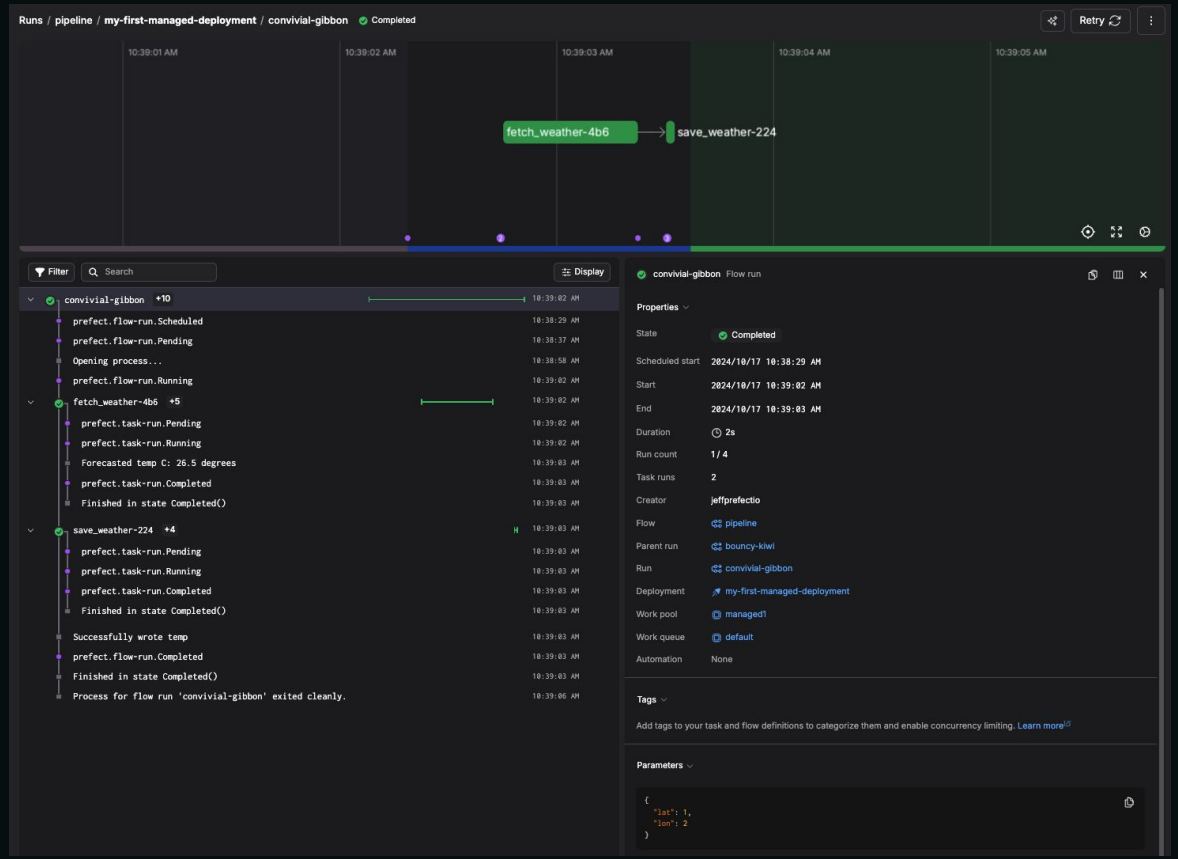
Parameters:

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

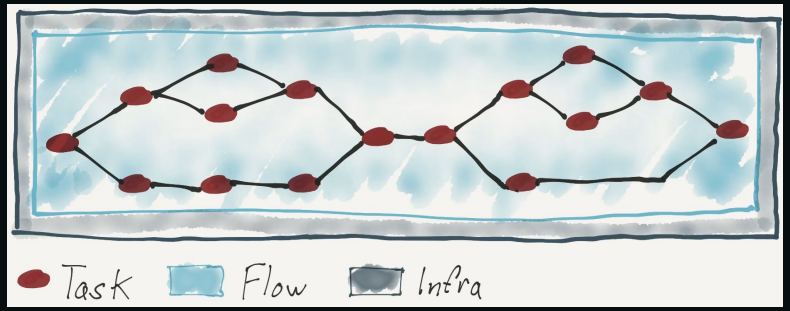
```python
from prefect import flow
from prefect.deployments import run_deployment


@flow
def run_deployment_from_flow():
    print("Running deployment from a flow")
    run_deployment(
        name="pipeline/my-first-managed-deployment", parameters={"lat": 1, "lon": 2}
    )
    return


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


run_deployment

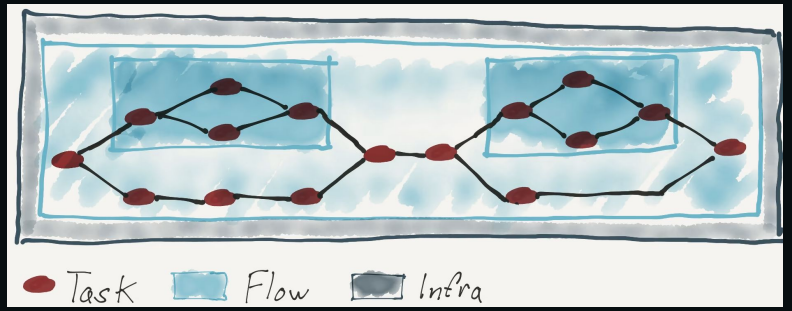


Event-triggered workflows

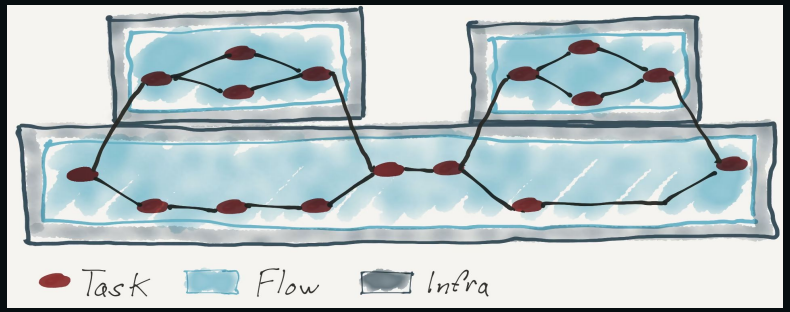
Workflow patterns - Event-triggered



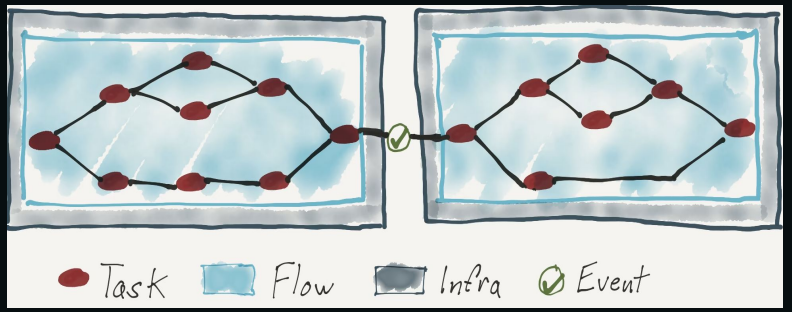
Flow of tasks



Flow of nested flows



Flow of deployments



Event-triggered flow

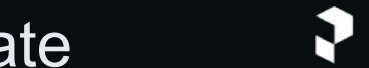
When to use which?

Pattern	Conceptual	Execution	Awaremeness
Flow of tasks	Coupled	Coupled	Coupled
Flow of nested flows	Separate	Coupled	Coupled
Flow of deployments	Separate	Separate	Coupled
Event-triggered flow	Separate	Separate	Separate

UI/ Organization

Infra/ Process

Context/ State



Event-driven flow runs with deployment triggers

Events (refresh)

Lightweight JSON bits

Describe what happened, who did it, etc.

Internal (Prefect-created events), examples:

- Work pool ready
- Flow run failed

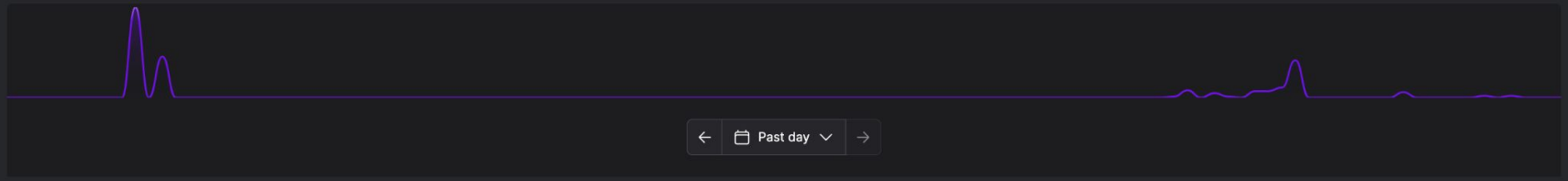
External examples

- S3 object created
- Github PR opened

Workspace Events

Resource: All resources

Events: All events



01:48:02 PM
May 22nd, 2024

Deployment updated
prefect.deployment.updated

Resource
Deployment process-s3-file

Related Resources
prefect-cloud.actor.b86c9e38-edb3-4617-aa4a-78b5ebb3e369 prefect-cloud.account.9a67b081-4f14-4035-b000-1f715f46231b prefect-cloud.workspace.f4689e77-b7c0-41cc-9ac9-139d6b69e030

01:47:58 PM
May 22nd, 2024

Deployment not ready
prefect.deployment.not-ready

Resource
Deployment process-s3-file

Related Resources
Flow process-file

01:24:51 PM
May 22nd, 2024

Deployment ready
prefect.deployment.ready

Resource
Deployment process-s3-file

Related Resources
Flow process-file

Deployment triggers

Allow deployments to run in response to the presence (or absence) of events

- Specify trigger condition in a *DeploymentEventTrigger* object and pass to *.deploy()*
- Linked automation created when deployment created

Deployment triggers - the flow to be triggered

```python
from prefect import flow
from prefect.events import DeploymentEventTrigger


@flow(log_prints=True)
def downstream_flow(ticker: str = "AAPL") -> str:
    print(f"got {ticker}")
```


Deployment triggers - the trigger

Create a *DeploymentEventTrigger* object

```
downstream_deployment_trigger = DeploymentEventTrigger(
    name="Upstream Flow - Pipeline",
    enabled=True,
    match_related={"prefect.resource.id": "prefect.flow.*"},
    expect={"prefect.flow-run.Completed"},
)
```

See the event specification docs:

docs.prefect.io/latest/automate/events/events

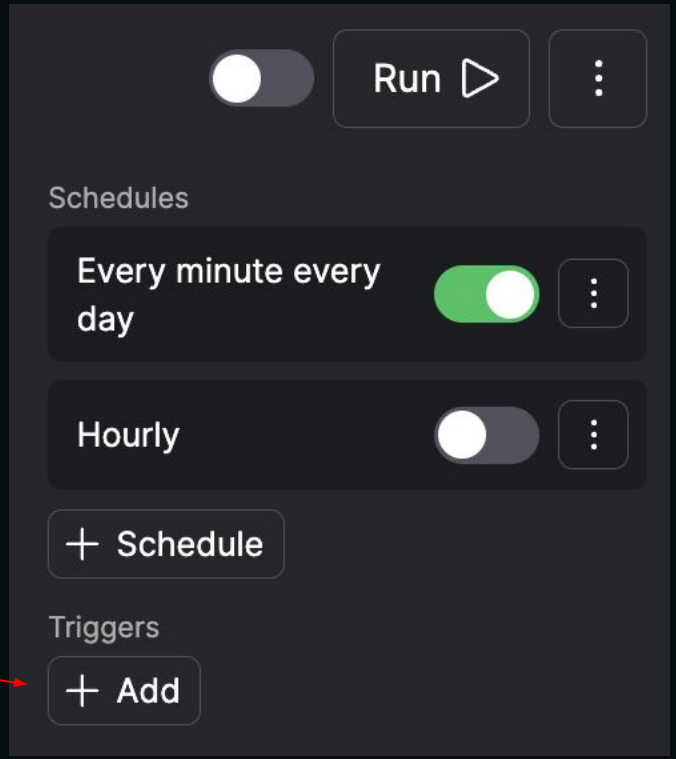
Deployment triggers - create

Pass the trigger object to *.deploy* and run the script

```python
if __name__ == "__main__":
    downstream_flow.from_source(
        source="https://github.com/prefecthq/pacc-2024-v6.git",
        entrypoint="105/dep-trigger.py:downstream_flow",
    ).deploy(
        name="ticker-deploy",
        work_pool_name="managed1",
        triggers=[downstream_deployment_trigger],
    )
```


Another way to begin automation creation in the UI

- Start from a deployment page
- Click the **+ Add** button under **Triggers**
- Pre-populates the automation action with the deployment run

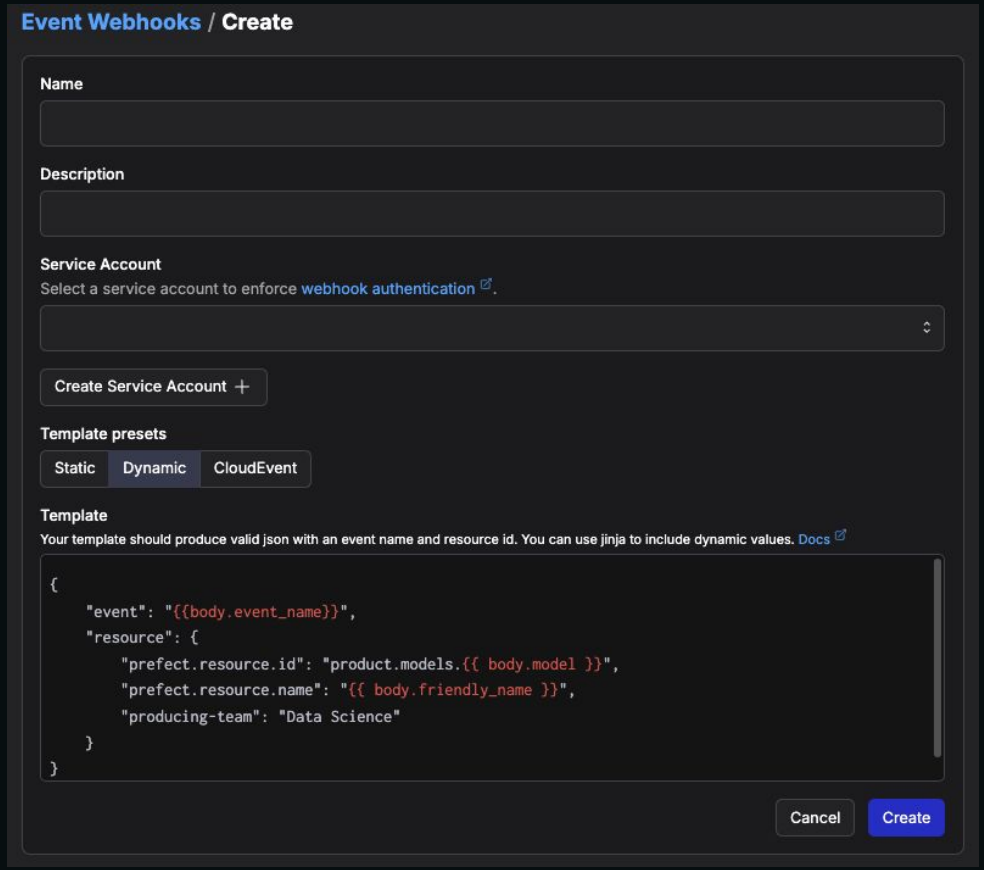


Event webhooks 🕸️🪝

Event webhooks

- Exposes URL endpoint
- Interface for integrating external apps
- When webhook URL pinged, creates Prefect event
  - Can be used as automation trigger
- Great when **not** in Python land
- Prefect Cloud only

Event webhooks



Event webhooks

- Use Jinja2 for dynamic templating
- Template must be valid JSON
- Create from UI or CLI
- Prefect Cloud Pro tier has authentication option through Service Accounts

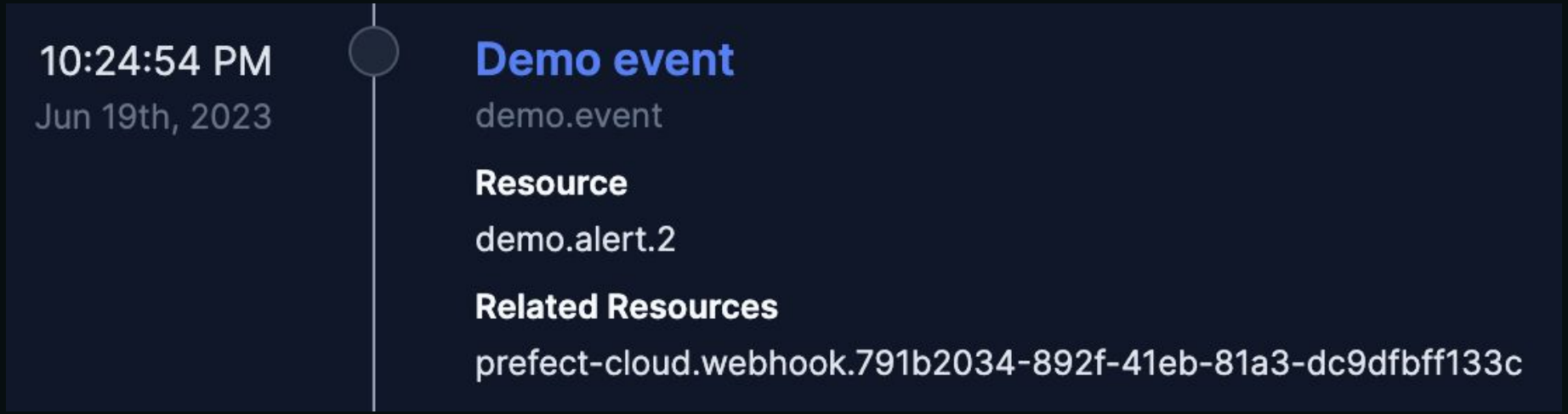
Event webhooks

Hit the endpoint provided by Prefect:

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

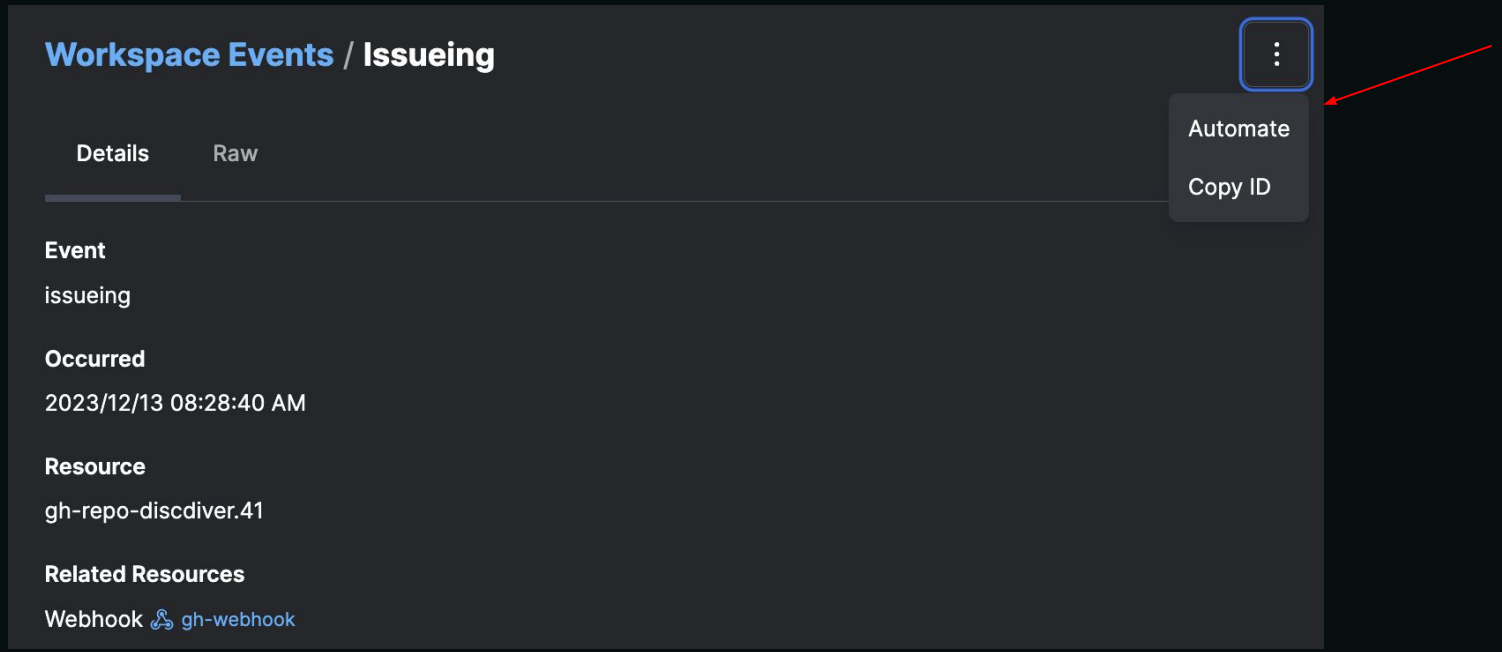
Event webhooks

See new event on **Event Feed** tab in the UI



Event webhooks

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



Event webhooks - example

When an object lands in an S3 bucket, can use EventBridge or Lambda to hit a Prefect webhook you've created.

Example:

github.com/PrefectHQ/prefect-demos/blob/main/flows/aws/datalake/README.md

Custom events defined in Python

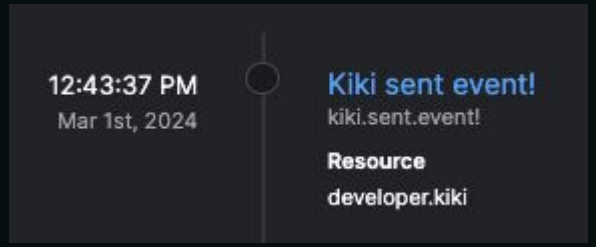
Create custom event to be emitted when code runs

emit_event requires two args:

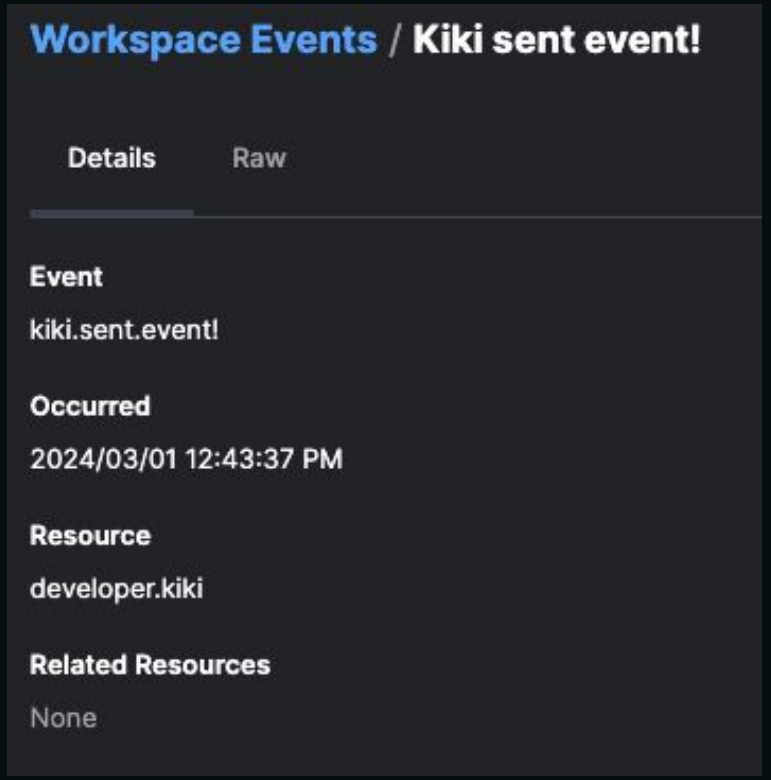
event and resource={":prefect.resource.id: val"}

```python
from prefect.events import emit_event


def emit_name_event(name: str = "kiki"):
    """Emit a basic Prefect event with a dynamically populated name"""
    print(f"Hi {name}!")
    emit_event(
        event=f"{name}.sent.event!",
        resource={"prefect.resource.id": f"developer.{name}"},
        payload={"name": name},
    )


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

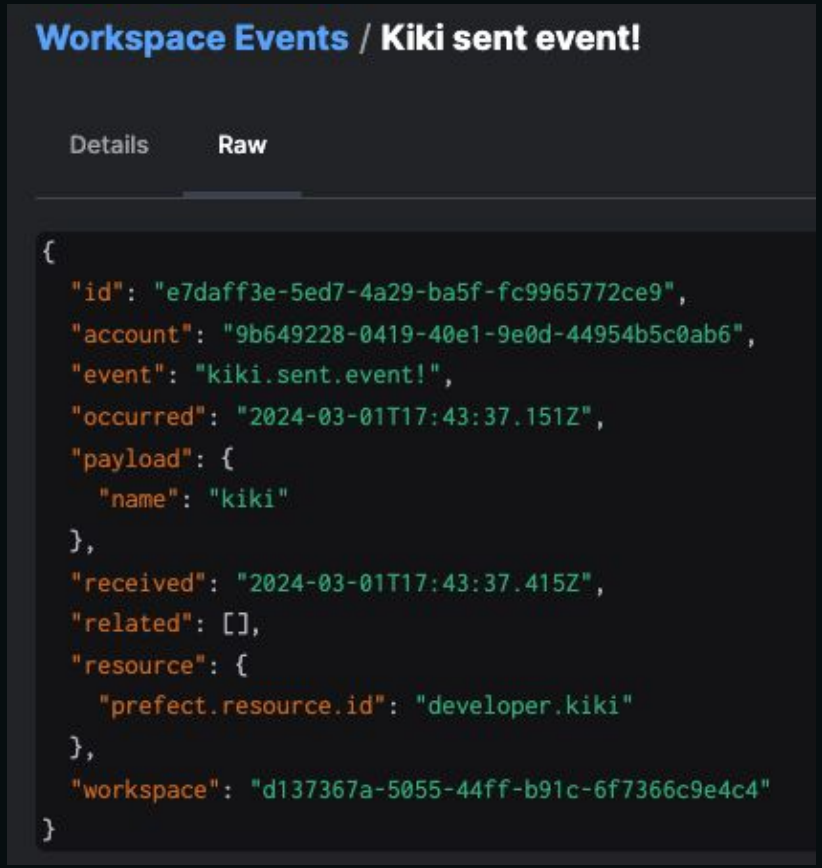

Run code and head to the **Event Feed** page



Click link to see event page



See event details on the **Raw** tab



Data from event can be used in an automation action

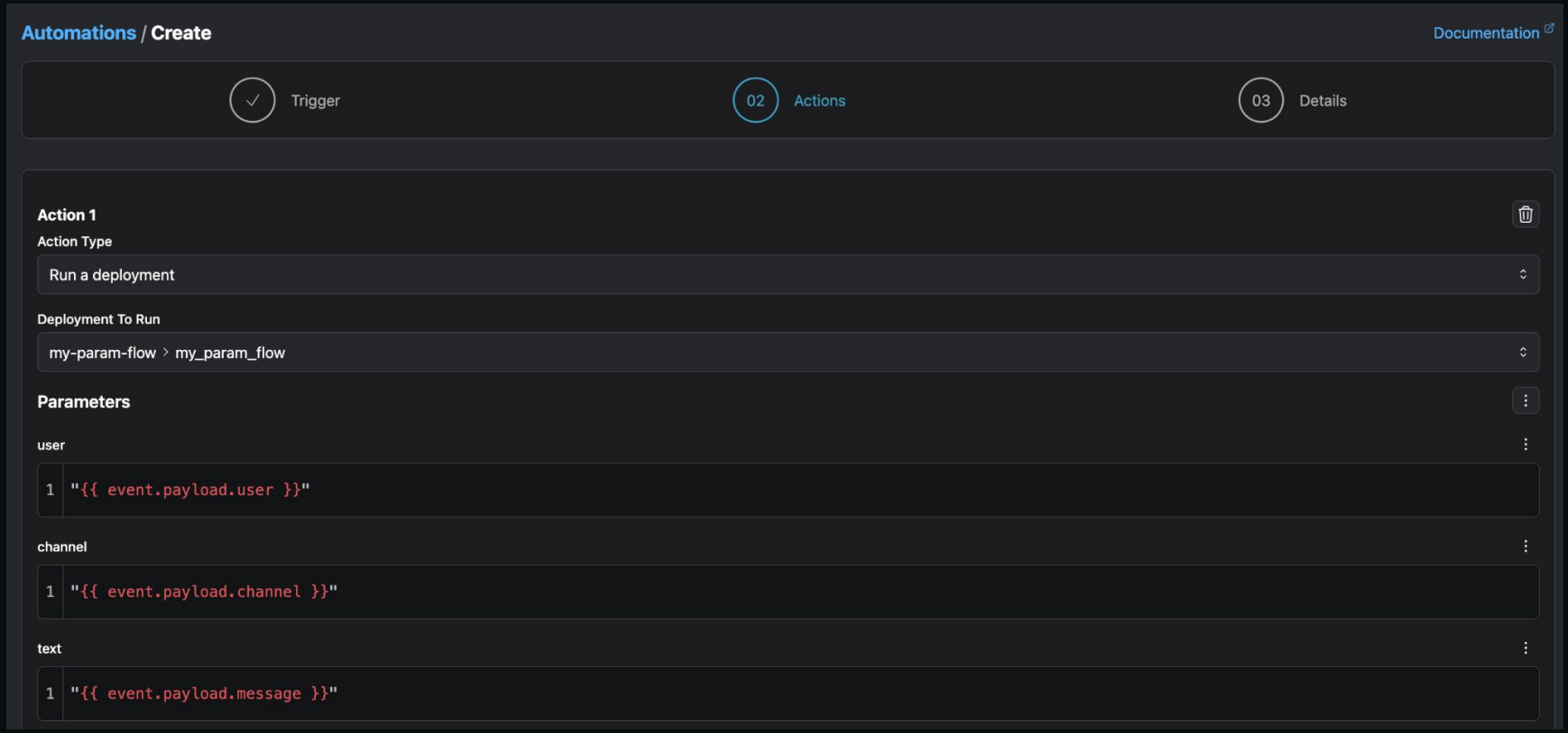
For example: Populate a flow param via a *Run Deployment* action

Use *emit_event*'s *payload* parameter

Example: custom event with detailed payload

```python
from prefect.events import emit_event

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


Use payload data in event-driven flow runs



Advice

- Use a flow with tasks for data engineering use cases unless you need another solution
- Use *run_deployment* if need to run a flow on different infrastructure
- Use event-driven workflows if want to run in response to an event

Tasks ++ 💪

Tasks on their own

- Lightweight
- Celery replacement
- Can run in background
- Can be nested and run outside of flows
- Can be parallelized
- Run client side; info sent to the API in batches
- Fast, but less real-time info available in UI

Note: you need a flow to create a deployment

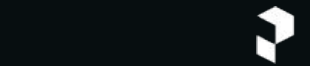
docs.prefect.io/latest/develop/deferred-tasks

105 Recap

You've seen how to use several workflow patterns with

- Nested flows
- *run_deployment*
- Event-based automations
  - Deployment event triggers defined in Python
  - Webhooks
  - Custom events defined in Python

- Tasks alone



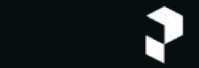
105 Lab

- Create a deployment with nested flows and run it.
- Create a flow that uses *run_deployment* to run another deployment.
- Stretch 1: Create a custom event in Python that triggers a notification action in an automation.
- Stretch 2: Create a webhook. Create an automation that pauses work when the webhook fires.

106 - Get more out of your workflows: Interact with workflows, run tasks concurrently, and create advanced automation triggers

106 Agenda

- Interact with workflows: pause workflows for human input
- Store values in variables on the server
- Prioritize and limit work
- Use more advanced triggers in automations
- Run tasks concurrently
- Test workflows



Interactive workflows

Interactive workflows

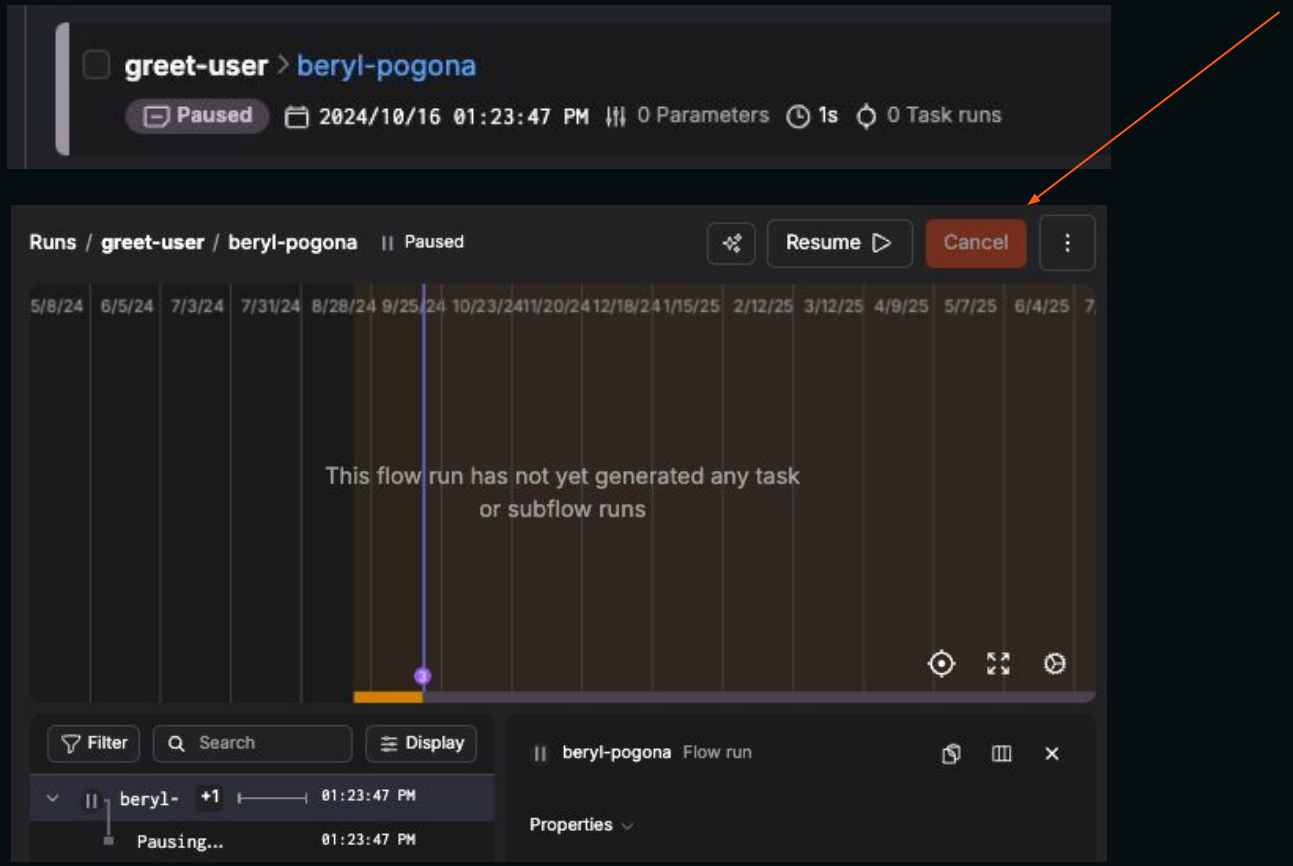
Pause a flow run to wait for input from a user via a web form (human-in-the-loop) 🙂

pause_flow_run function

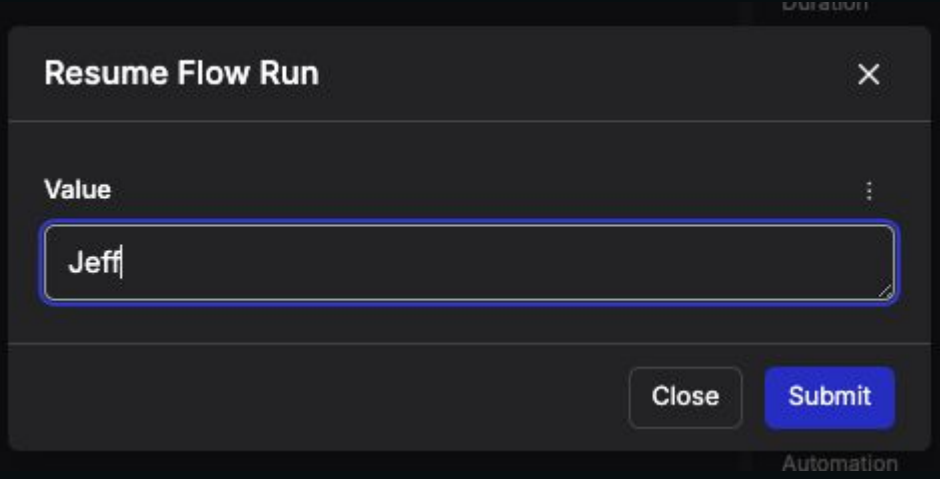
Human-in-the-loop: basic

```python
from prefect import flow, pause_flow_run


@flow(log_prints=True)
def greet_user():
    name = pause_flow_run(str)
    print(f"Hello, {name}!")


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

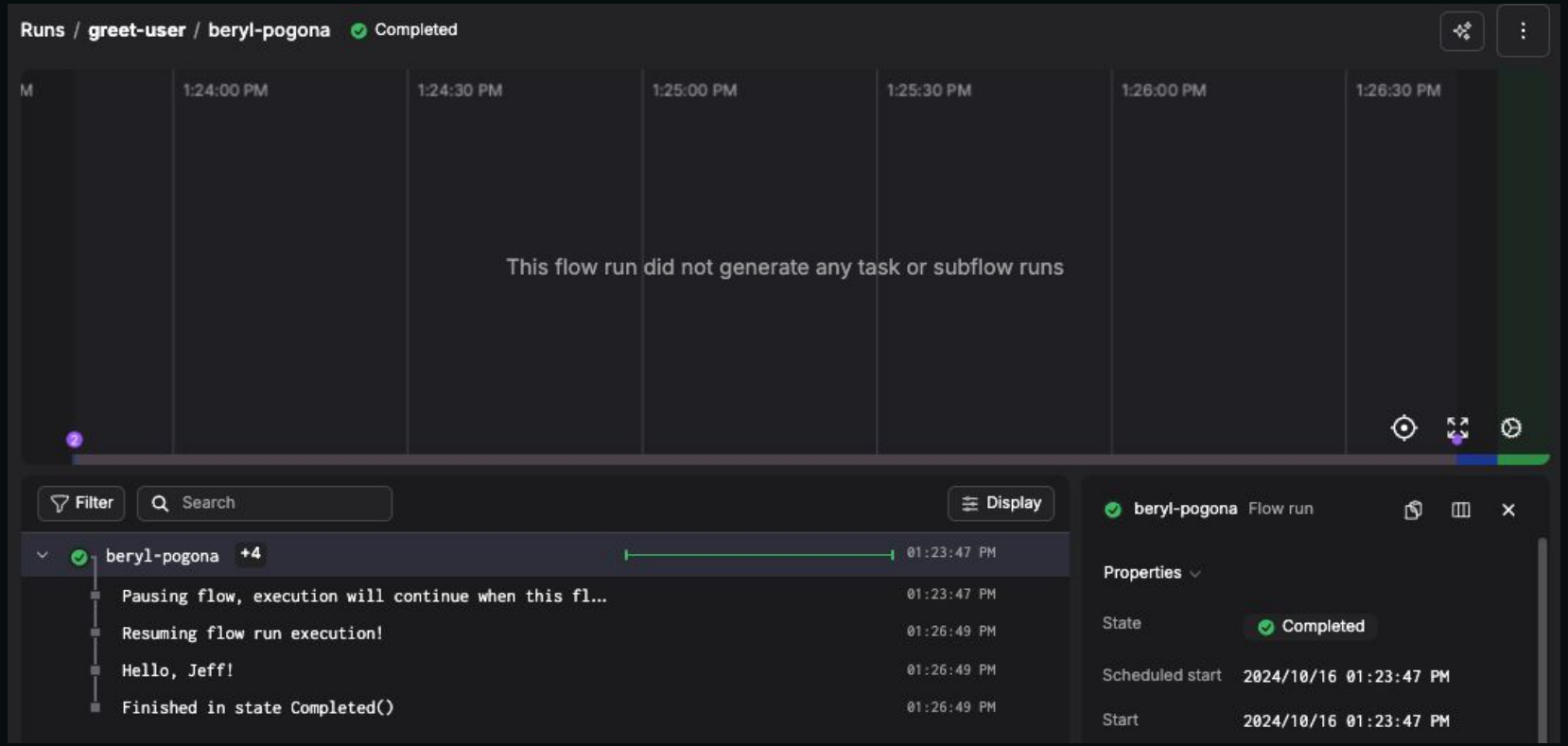

Human-in-the-loop: basic



Human-in-the-loop: basic



Human-in-the-loop: basic

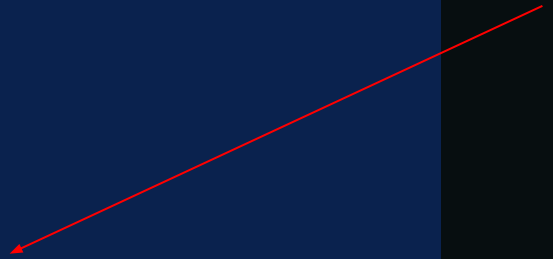


Human-in-the-loop: options

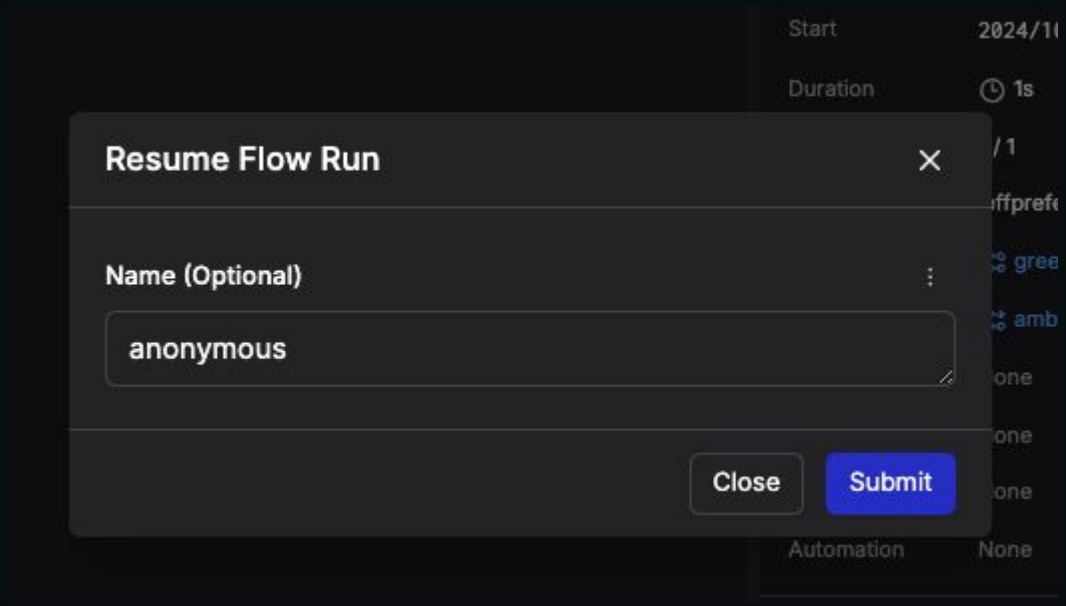
- Validate using *RunInput* class (subclass of Pydantic's *BaseModel*)
- Specify default value
- Create dropdown

Human-in-the-loop: default value

```python
import asyncio
from prefect import flow, pause_flow_run
from prefect.input import RunInput


class UserNameInput(RunInput):
    name: str


@flow(log_prints=True)
async def greet_user():
    user_input = await pause_flow_run(
        wait_for_input=UserNameInput.with_initial_data(name="anonymous")
    )

    if user_input.name == "anonymous":
        print("Hello, stranger!")
    else:
        print(f"Hello, {user_input.name}!")


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



Human-in-the-loop: default value



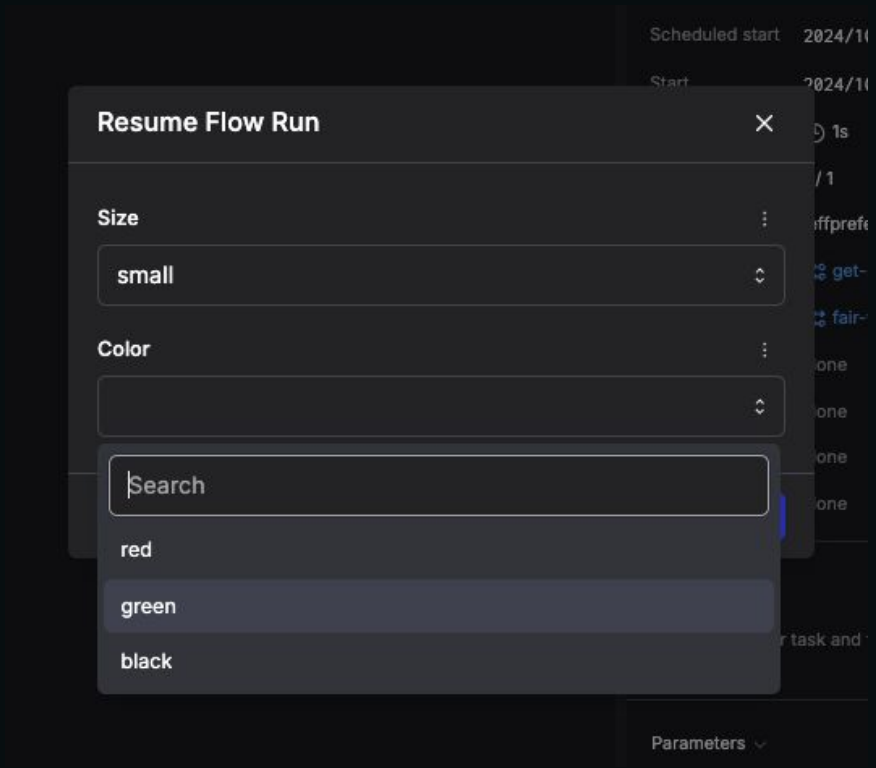
Human-in-the-loop: custom validation

```python
from typing import Literal
import pydantic
from prefect import flow, pause_flow_run
from prefect.input import RunInput


class ShirtOrder(RunInput):
    size: Literal["small", "medium", "large", "xlarge"]
    color: Literal["red", "green", "black"]

    @pydantic.validator("color")
    def validate_age(cls, value, values, **kwargs):
        if value == "green" and values["size"] == "small":
            raise ValueError("Green is only in-stock for medium, large, and XL sizes.")

        return value
```

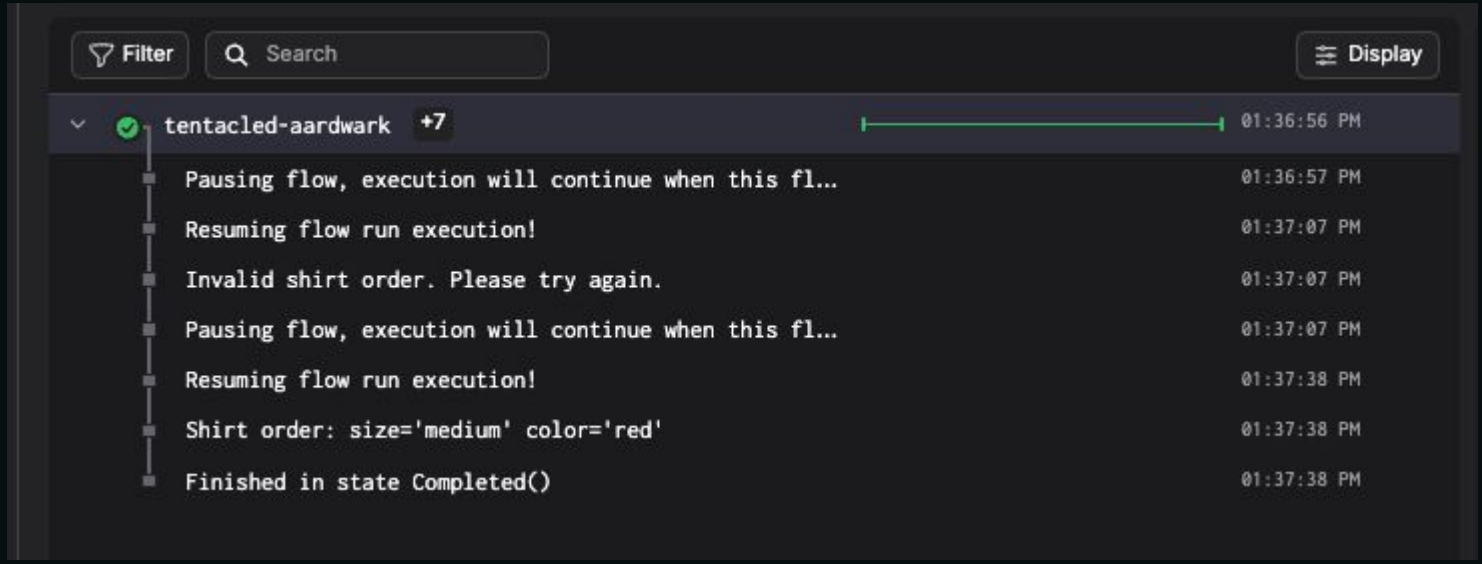

Human-in-the-loop: custom validation

```python
@flow(log_prints=True)
def get_shirt_order():
    shirt_order = None

    while shirt_order is None:
        try:
            shirt_order = pause_flow_run(wait_for_input=ShirtOrder)
            print(f"Shirt order: {shirt_order}")
        except pydantic.ValidationError:
            print("Invalid shirt order. Please try again.")
```


Human-in-the-loop: custom validation



Human-in-the-loop: custom validation



Human-in-the-loop

Potential uses:

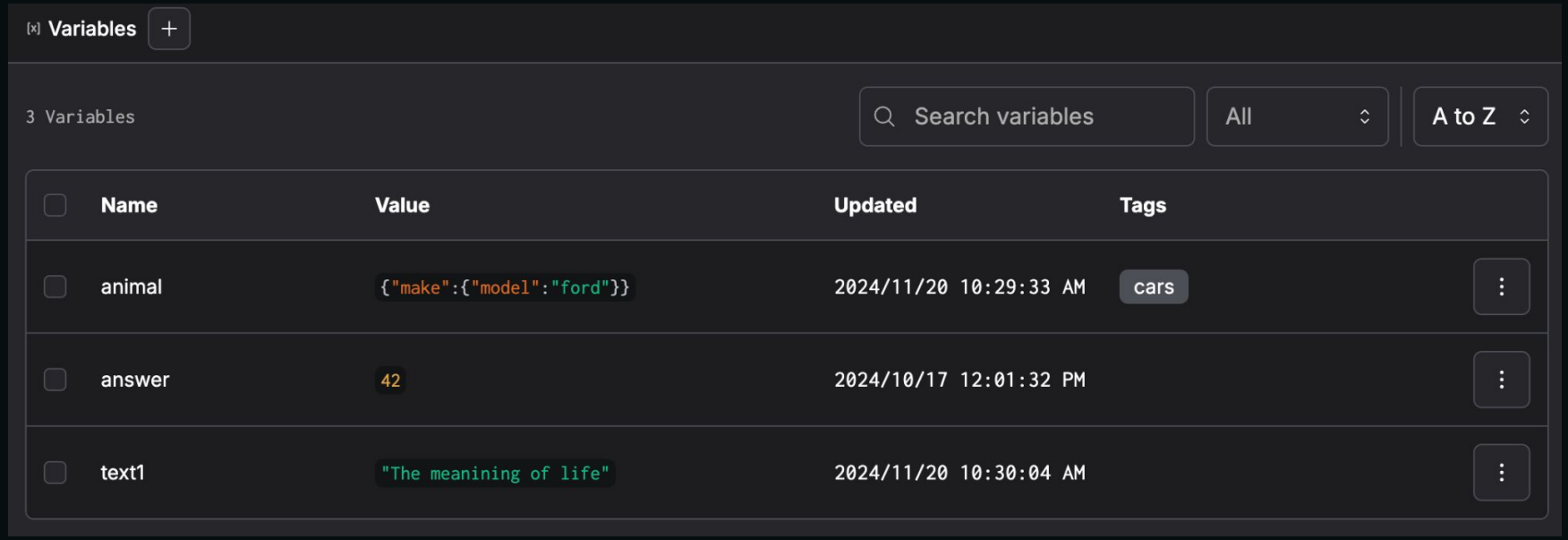
- Finance approval prior to running an expensive workflow
- AI classifier labeling - request human intervention when low confidence in label

Variables

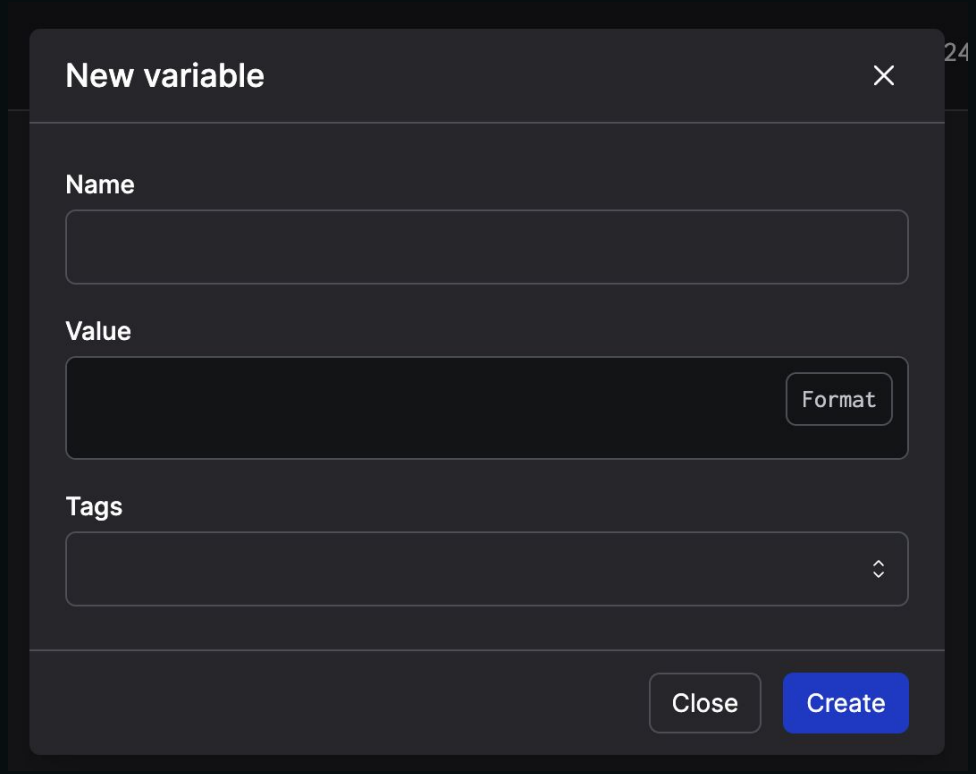
Prefect variables

- Store and reuse non-sensitive, small data
- Key-value pairs stored in the database
- Create via UI, Python code, or CLI
- Can be any serializable JSON
- Replacement for basic block types

Prefect variables



Prefect variables



Prefect variables

```python
from prefect.variables import Variable

Variable.set(name="answer", value=42)
```

```python
from prefect.variables import Variable

var = Variable.get("answer")
print(var)
```


Concurrency

Concurrency

- Helpful when waiting for external systems to respond
- Allows other work to be done while waiting
- Use *ThreadPoolTaskRunner* to execute code concurrently

Task Runners

- Specify in *flow* decorator
- Creates a Prefect future object

⚠ Prefect future objects must be resolved explicitly before returning from a flow. Dependencies between futures will be automatically resolved whenever their dependents are resolved. This means that only *terminal* futures need to be resolved, either by:

- returning the terminal futures from your flow
- calling `.wait()` or `.result()` on each terminal future
- using one of the top level `wait` or `as_completed` utilities to resolve terminal futures

Not doing so may leave your tasks in an unfinished state.

Concurrency

```python
from prefect import flow, task
from prefect.futures import wait
from prefect.task_runners import ThreadPoolTaskRunner
import time


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


@flow(task_runner=ThreadPoolTaskRunner(max_workers=3))
def elevator():
    floors = []

    for floor in range(3, 0, -1):
        floors.append(stop_at_floor.submit(floor))

    wait(floors)
```


Concurrency

```
elevator moving to floor 3
elevator moving to floor 1
elevator moving to floor 2
elevator stops on floor 1
15:24:23.662 | INFO    | Task run 'stop_at_floor-599' - Finished in state Completed()
elevator stops on floor 2
15:24:24.661 | INFO    | Task run 'stop_at_floor-06f' - Finished in state Completed()
elevator stops on floor 3
15:24:25.662 | INFO    | Task run 'stop_at_floor-8ac' - Finished in state Completed()
```


Parallelism

Parallelism

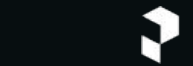
- Two or more operations happening at the same time on one or more machines
- Helpful when operations limited by CPU
- Many machine learning algorithms parallelizable

Task Runners for parallelism

- DaskTaskRunner
- RayTaskRunner

Required integration packages:

- *prefect-dask*
- *prefect-ray*

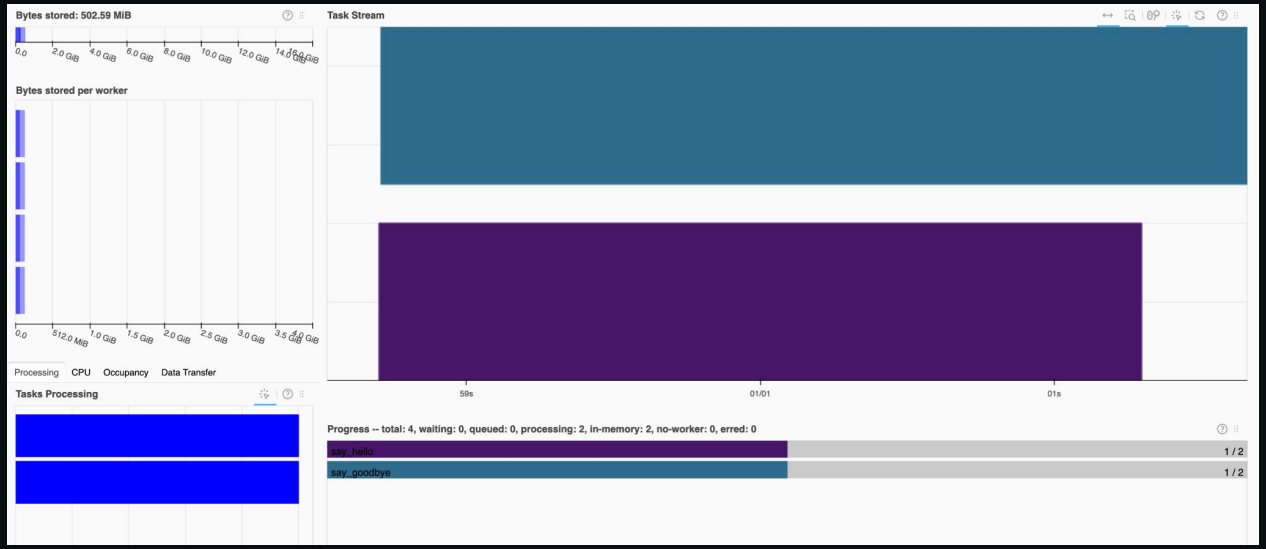


DaskTaskRunner for parallelism

```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

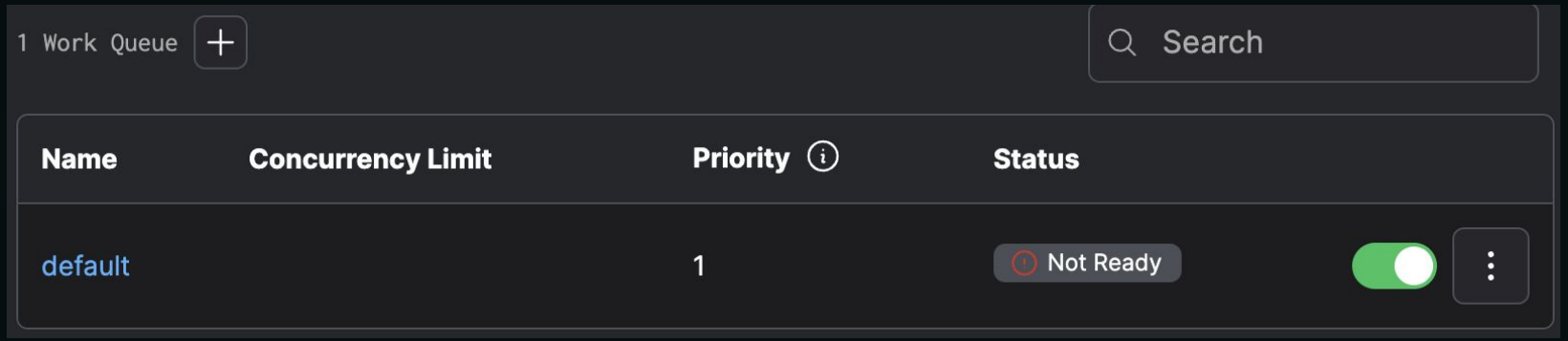
- Can see the Dask UI if have bokeh package installed: *pip install -U bokeh*
- UI will be linked in the terminal at run time



Prioritize and limit work

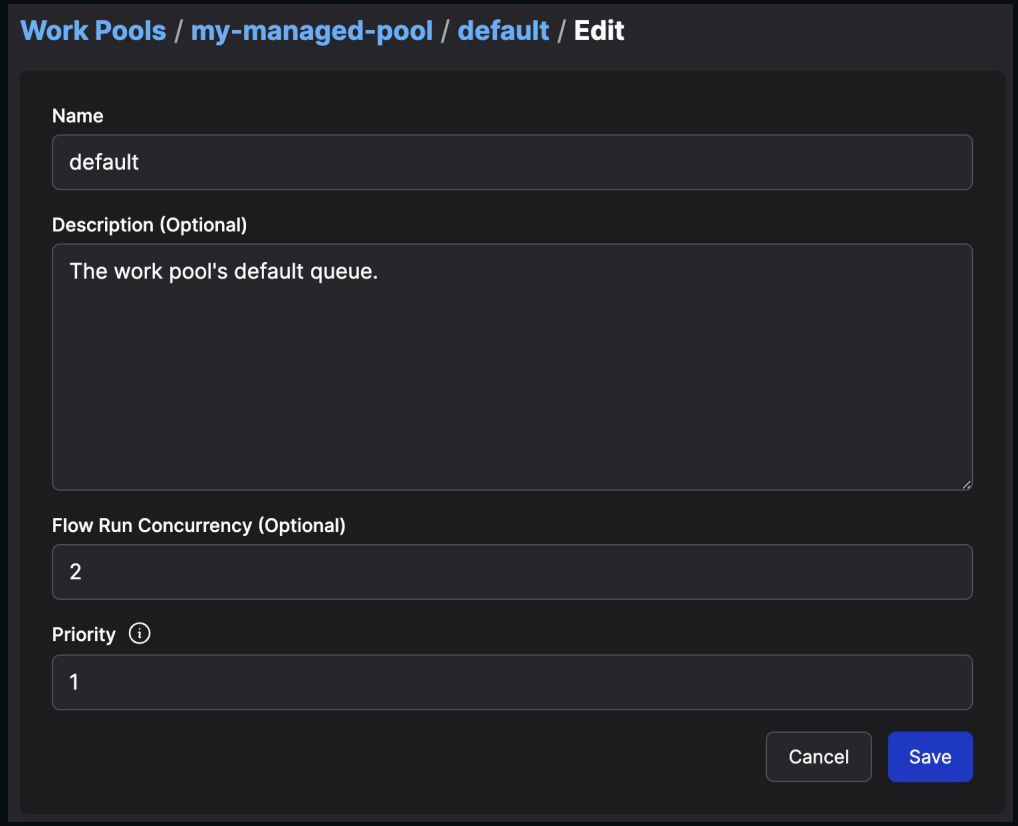
What's a work queue for?

- Prioritize work
- Limit concurrent runs
- A work pool can have many work queues



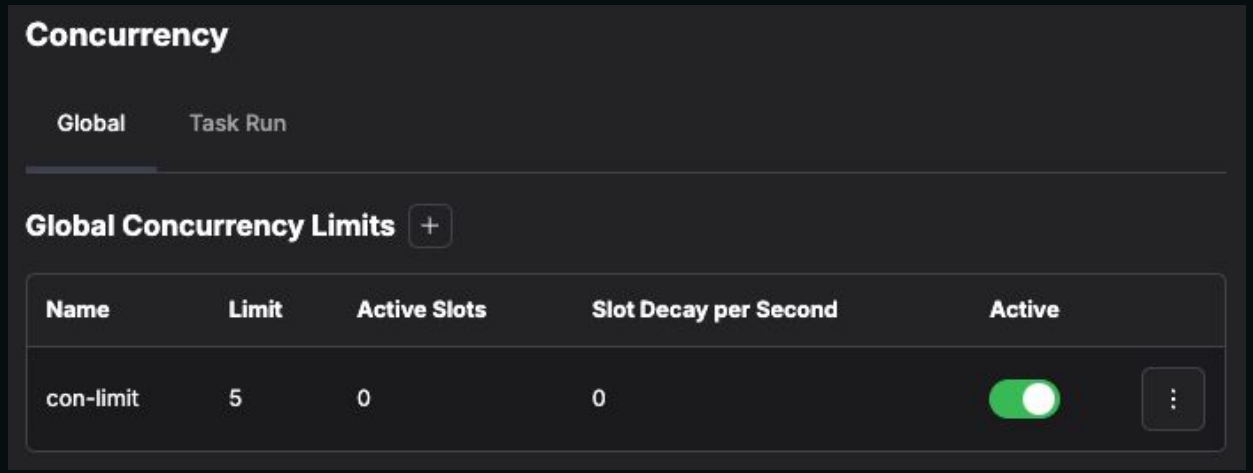
⚠️ *default* work queue created automatically

What’s a work queue for?



Other places to limit concurrency

- Globally (workspace level)
- Deployment
- Task run
- Work pool



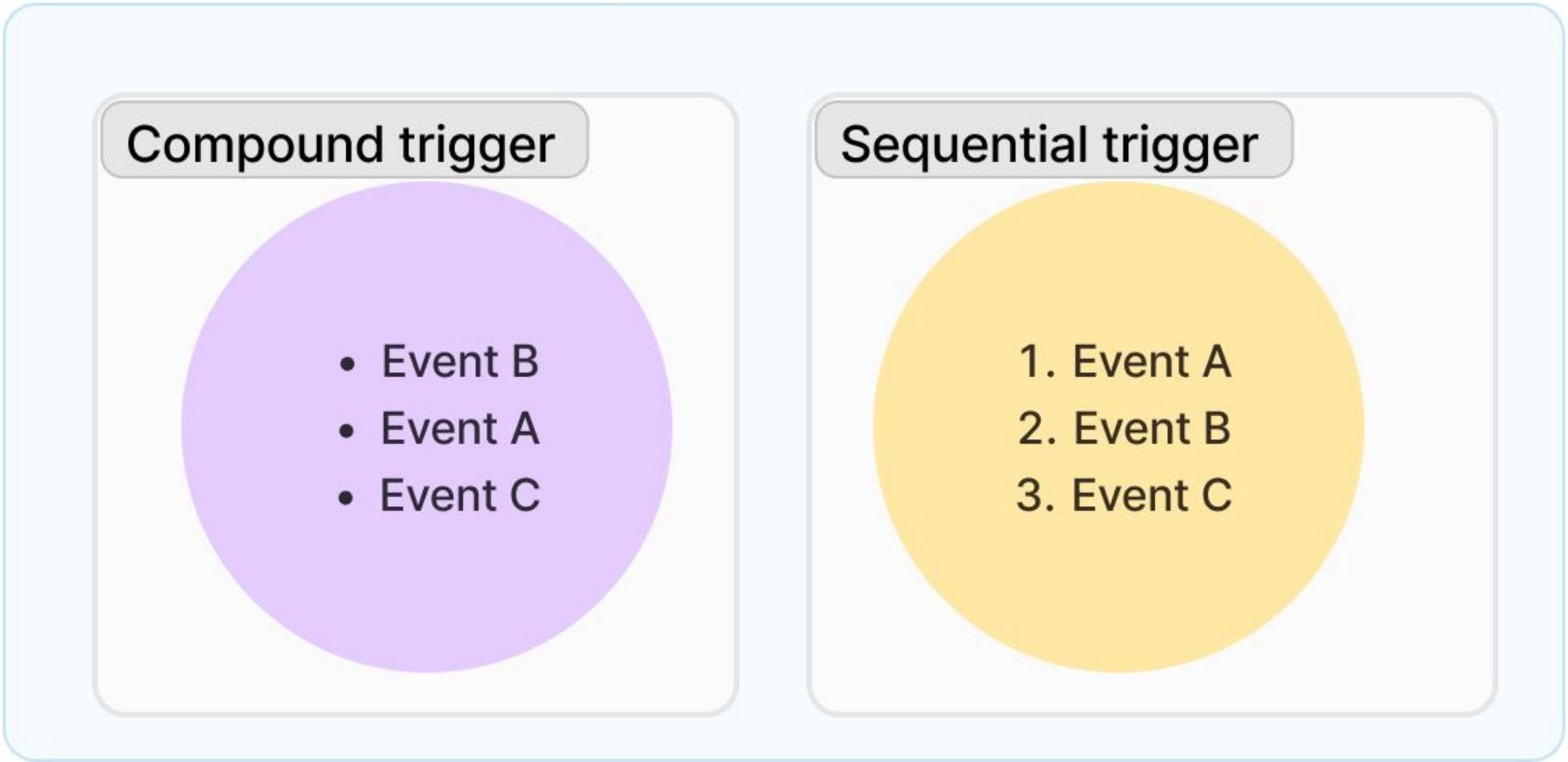
Advanced triggers

Advanced triggers

- Composite triggers
- Metric triggers

Composite triggers

Composite trigger types



Composite triggers

An automation trigger made of more than one event

- **Compound:** any order
- **Sequential:** must occur in prescribed order

Optional: set a time period for events to occur

AND or OR

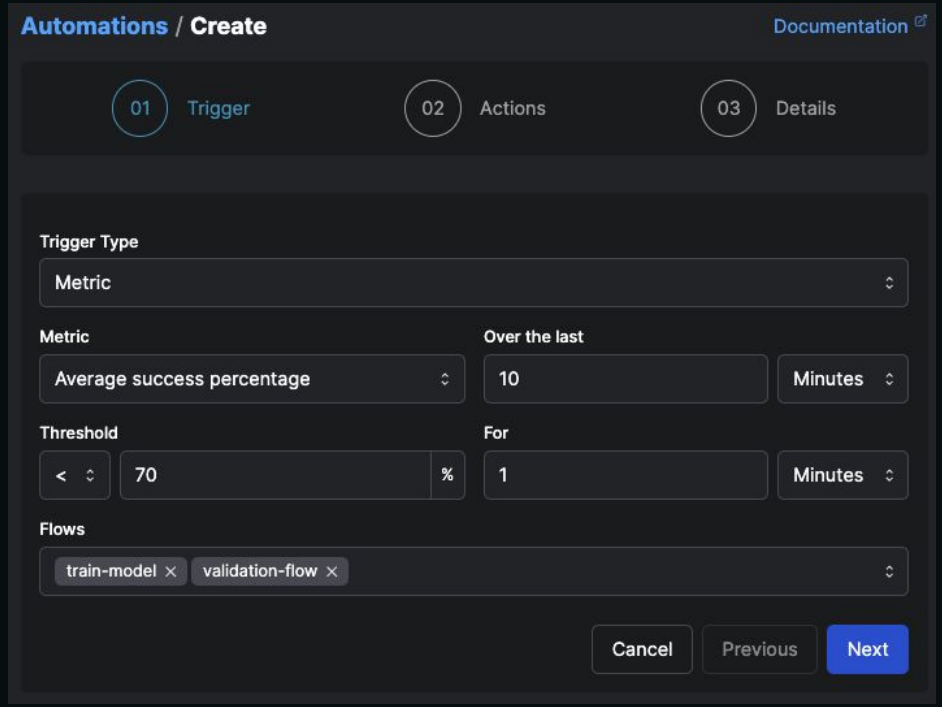
Composite triggers - example JSON

```
{
  "type": "compound",
  "require": "all",
  "within": 3600,
  "triggers": [
    {
      "type": "event",
      "posture": "Reactive",
      "expect": ["prefect.block.remote-file-system.write_path.called"],
      "match_related": {
        "prefect.resource.name": "daily-customer-export",
        "prefect.resource.role": "flow"
      }
    },
    {
      "type": "event",
      "posture": "Reactive",
      "expect": ["prefect.block.remote-file-system.write_path.called"],
      "match_related": {
        "prefect.resource.name": "daily-revenue-export",
        "prefect.resource.role": "flow"
      }
    },
```


Metric triggers

Metric triggers

Create an automation that uses a metric as a trigger



Metric triggers

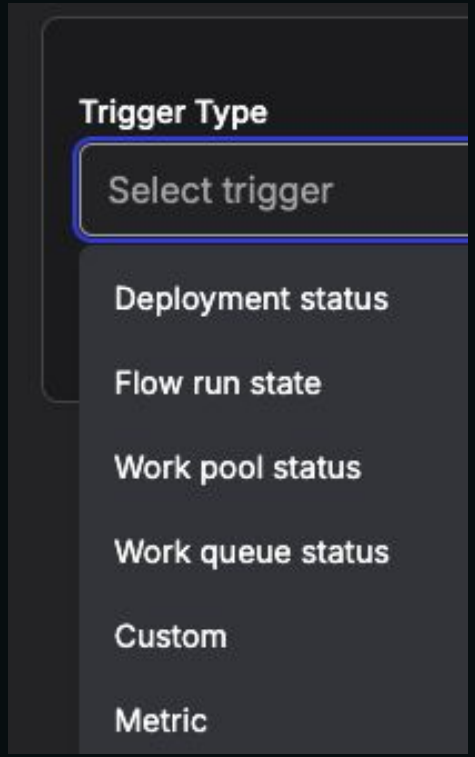
When a pattern is detected, then take an action.

Examples:

- Send a notification
- Toggle on a work pool
- Run a deployment
- Resume a flow run

Other trigger types

- Can use status of many Prefect objects as trigger



Testing

Testing

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

```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

```
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
```


106 Recap

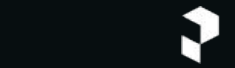
You've seen how to:

- Pause a flow run for human input
- Store values on the server with variables
- Prioritize and limit work
- Create automations with compound and metric triggers
- Run tasks concurrently and in parallel
- Test workflows

Lab 106

106 Lab

- Create an interactive workflow that pauses a flow run for input from a user.
- Create a variable in the UI or with code. Use the variable in your code.
- Stretch 1: Run tasks concurrently with *ThreadPoolTaskRunner*
- Stretch 2: Use a compound trigger in an automation.
- Stretch 3: Use a metric trigger in an automation.



Wrap

Brief feedback survey

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

Congratulations!

Prefect Associate
Certification Course

Bonus

Bonus content

- Productionization & CI/CD
  - Define deployments in prefect.yaml
  - GitHub actions - pre-built deployment actions
  - Helm Chart
  - Terraform
  - Rest API - bypass the Python SDK
- Prefect shell to wrap bash scripts in a flow for observability
- State change hooks to take action in response to state changes
- ControlFlow for managing agentic AI workflows with Prefect

Deployment creation with prefect.yaml

prefect.yaml

- Define a deployment in YAML
- Alternative to Python-based deployment creation methods

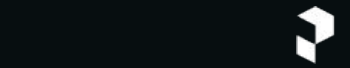
We'll use an interactive CLI experience to generate our YAML spec

Deployment: 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!"
```


Deployment: ETL code

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

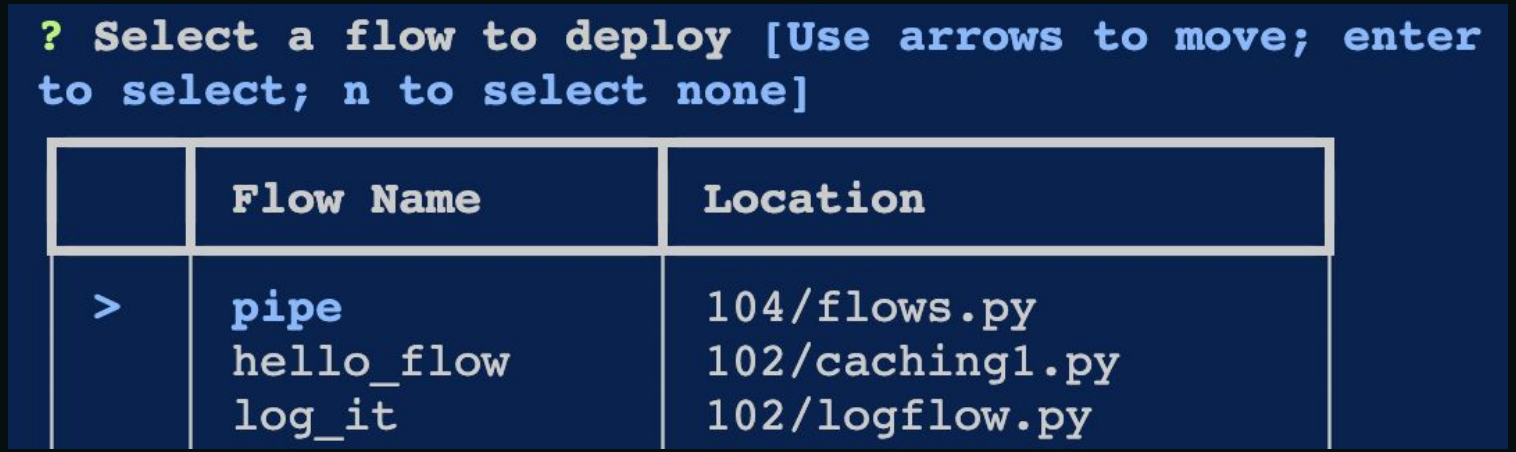



Create a deployment

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

prefect deploy

Choose the flow you want to put into a deployment



Deployment creation

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

```
? Deployment name (default): buy_deploy
? Would you like to configure schedules for this deployment? [y/n] (y): n
```


Work pools

Choose a work pool

```
? 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	

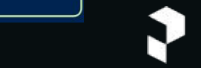
Specify flow code storage

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

```
? Would you like to build a custom Docker image for this deployment?
[y/n] (n): n
? 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 a
remote storage location when running this flow? [y/n] (y): y
? Please select a remote code storage option. [Use arrows to move;
enter to select]
```

	Storage Type	Description
>	Git Repo	Use a Git repository (recommended).
	S3	Use an AWS S3 bucket.
	GCS	Use a Google Cloud Storage bucket.
	Azure Blob Storage	Use an Azure Blob Storage bucket.

```
? Is https://github.com/PrefectHQ/pacc-2024-v6.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 'buy/buy deploy' successfully created with id '3e6d1fe9-8150-4428-9f2e-32f462f39f51'.
```

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 buy_deploy

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

prefect.yaml

```yaml
# Generic metadata about this project
name: pacc-2024-v6
prefect-version: 3.0.3

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

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

# pull section allows you to provide instructions for cloning this project in remote locations
pull:
- prefect.deployments.steps.git_clone:
    repository: https://github.com/PrefectHQ/pacc-2024-v6.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
```


Use *deployments:* section to override steps 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

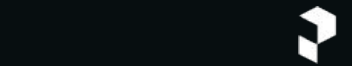
```
? Would you like to use an existing deployment configuration? [Use arrows to move; enter to select; n to select none]
```

	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



prefect deploy

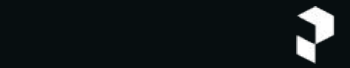
If choose *docker* typed work pool you will be asked docker-related questions

	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):

Resulting *prefect.yaml*

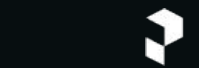
```yaml
- name: dock-interact
  version:
  tags: []
  description:
  entrypoint: 104/flows.py:pipe
  parameters: {}
  work_pool:
    name: docker-pool
    work_queue_name:
    job_variables:
      image: '{{ build-image.image }}'
  schedule:
  build:
  - prefect_docker.deployments.steps.build_docker_image: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

.prefectignore

Like *.gitignore* but for Prefect deployments

Created with the interactive deploy experience



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 create a Prefect deployment
- github.com/marketplace/actions/deploy-a-prefect-flow

GitHub Action

See the guide:

docs.prefect.io/latest/deploy/infrastructure-concepts/deploy-ci-cd

```yaml
name: Deploy Prefect flow

on:
  push:
    branches:
      - main

jobs:
  deploy:
    name: Deploy
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Log in to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Setup Python
        uses: actions/setup-python@v5
        with:
          python-version: "3.11"

      - name: Prefect Auth
        uses: PrefectHQ/actions-prefect-auth@v1
        with:
          prefect-api-key: ${{ secrets.PREFECT_API_KEY }}
          prefect-workspace: ${{ secrets.PREFECT_WORKSPACE }}

      - name: Run Prefect Deploy
        uses: PrefectHQ/actions-prefect-deploy@v3
        with:
          deployment-names: my-deployment
          requirements-file-paths: requirements.txt
```


Helm Chart

Prefect Helm Chart for K8s

Provides a variety of functionality

Creating workers is a popular use case

See more in the docs:
github.com/PrefectHQ/prefect-helm/tree/main/charts/prefect-worker

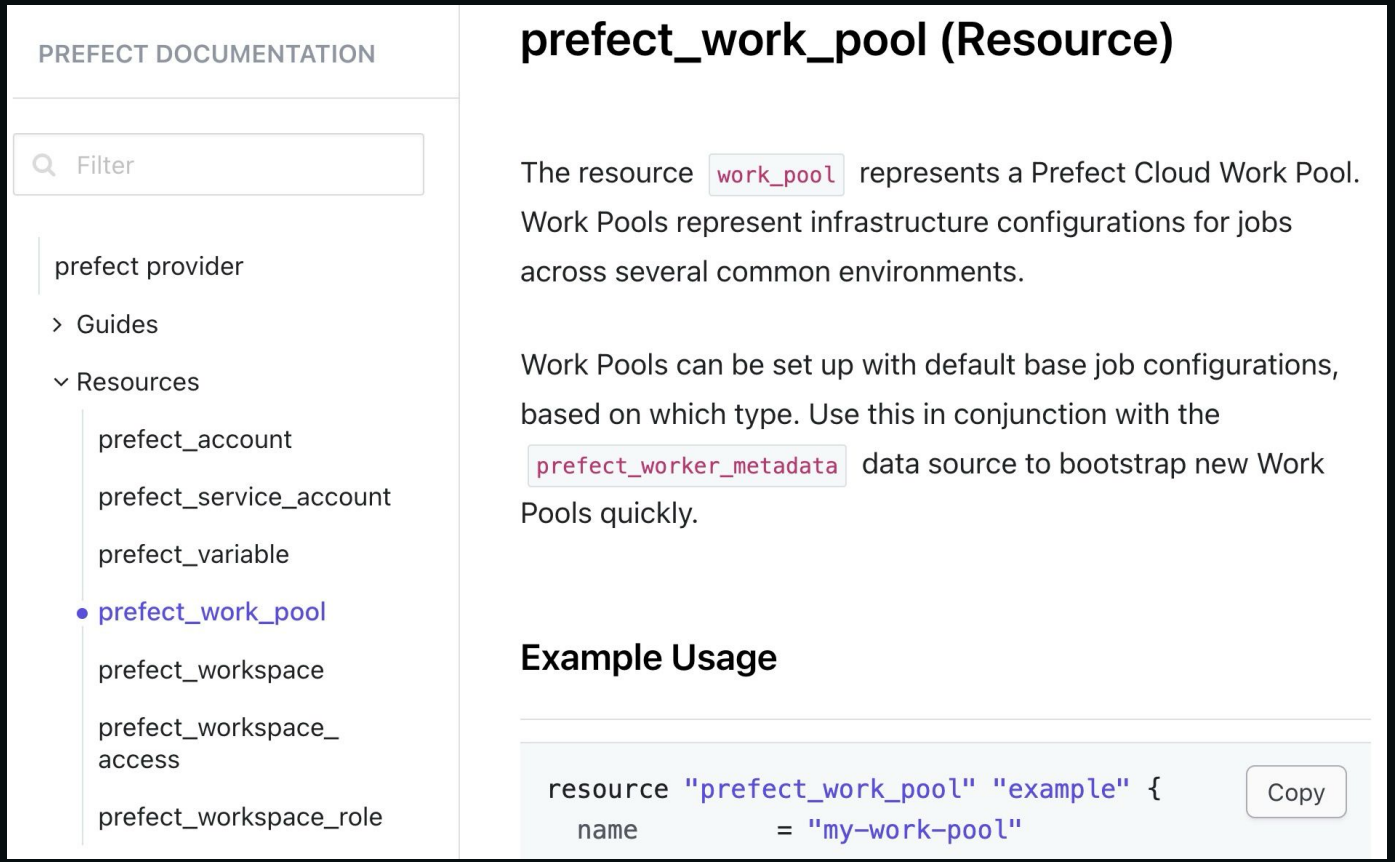
Terraform provider

Prefect Cloud Terraform Provider

See the docs:

registry.terraform.io/providers/PrefectHQ/prefect/latest/docs

Prefect Cloud Terraform Provider



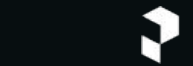
Prefect REST API

Interact with the REST API

Cloud and server REST API interactive docs:

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

curl or use an HTTP client (*httpx*, *requests*)



Interact with the REST API - *requests* example

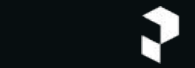
```python
import requests


PREFECT_API_URL="https://api.prefect.cloud/api/accounts/abc-my-cloud-account-id-
PREFECT_API_KEY="123abc_my_api_key_goes_here"
data = {
    "sort": "CREATED_DESC",
    "limit": 5,
    "artifacts": {
        "key": {
            "exists_": True
        }
    }
}

headers = {"Authorization": f"Bearer {PREFECT_API_KEY}"}
endpoint = f"{PREFECT_API_URL}/artifacts/filter"

response = requests.post(endpoint, headers=headers, json=data)
assert response.status_code == 200
for artifact in response.json():
    print(artifact)
```

PrefectClient - convenient for interacting with REST API

Return list of flows:

```python
import asyncio
from prefect.client.orchestration 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())
```

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



Common methods

- *create_flow_run_from_deployment*
- *read_flow_run / read_flow_runs*
- *update_deployment*
- *delete_flow_run*

Full list:
github.com/PrefectHQ/prefect/blob/main/src/prefect/client/orchestration.py

Turn shell commands into Prefect flows

Turn shell commands into a Prefect flow

Useful if you already have shell script workflows

Helps a team adopt Prefect incrementally

Turn a shell command into a Prefect flow you can **observe**

No Python required!

prefect shell watch *"curl http://wttr.in/Chicago?format=3"*

Turn a shell command into a Prefect flow you can **observe**

Look familiar?

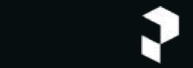
```
09:38:01.587 | INFO    | prefect.engine - Created flow run 'outrageous-octopus' for flow 'Shell Command'
09:38:01.588 | INFO    | Flow run 'outrageous-octopus' - View at https://app.prefect.cloud/account/55c7f
5e5-2da9-426c-8123-2948d5e5d94b/workspace/0c527f43-4688-4a1c-bc65-4856ea3a52a5/flow-runs/flow-run/978ca8
7e-6670-4a56-ad3b-2065d1ac8994
09:38:02.431 | INFO    | Flow run 'outrageous-octopus' - Chicago: ⛅  +47°F
09:38:03.018 | INFO    | Flow run 'outrageous-octopus' - Finished in state Completed()
```


OR turn a shell command into a Prefect flow you can **orchestrate**

Create a long running *serve* process and deployment

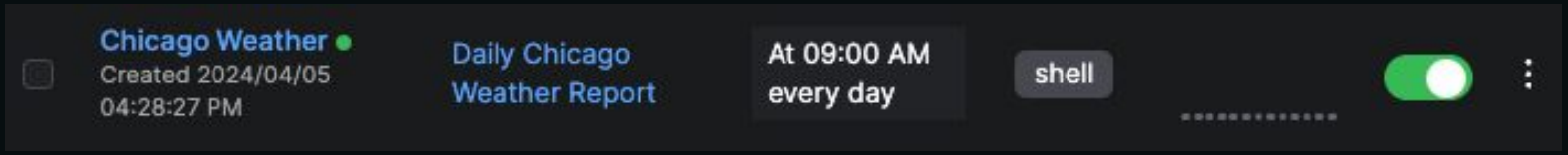
prefect shell serve *"curl http://wttr.in/Chicago?format=3"*

Still no Python required!



Turn a shell command into a Prefect flow you can **orchestrate**

*prefect shell serve "curl http://wttr.in/Chicago?format=3" --flow-name "Daily Chicago Weather Report" --cron-schedule "0 9 * * *" --deployment-name "Chicago Weather"*



Creates *serve* process and deployment with schedule

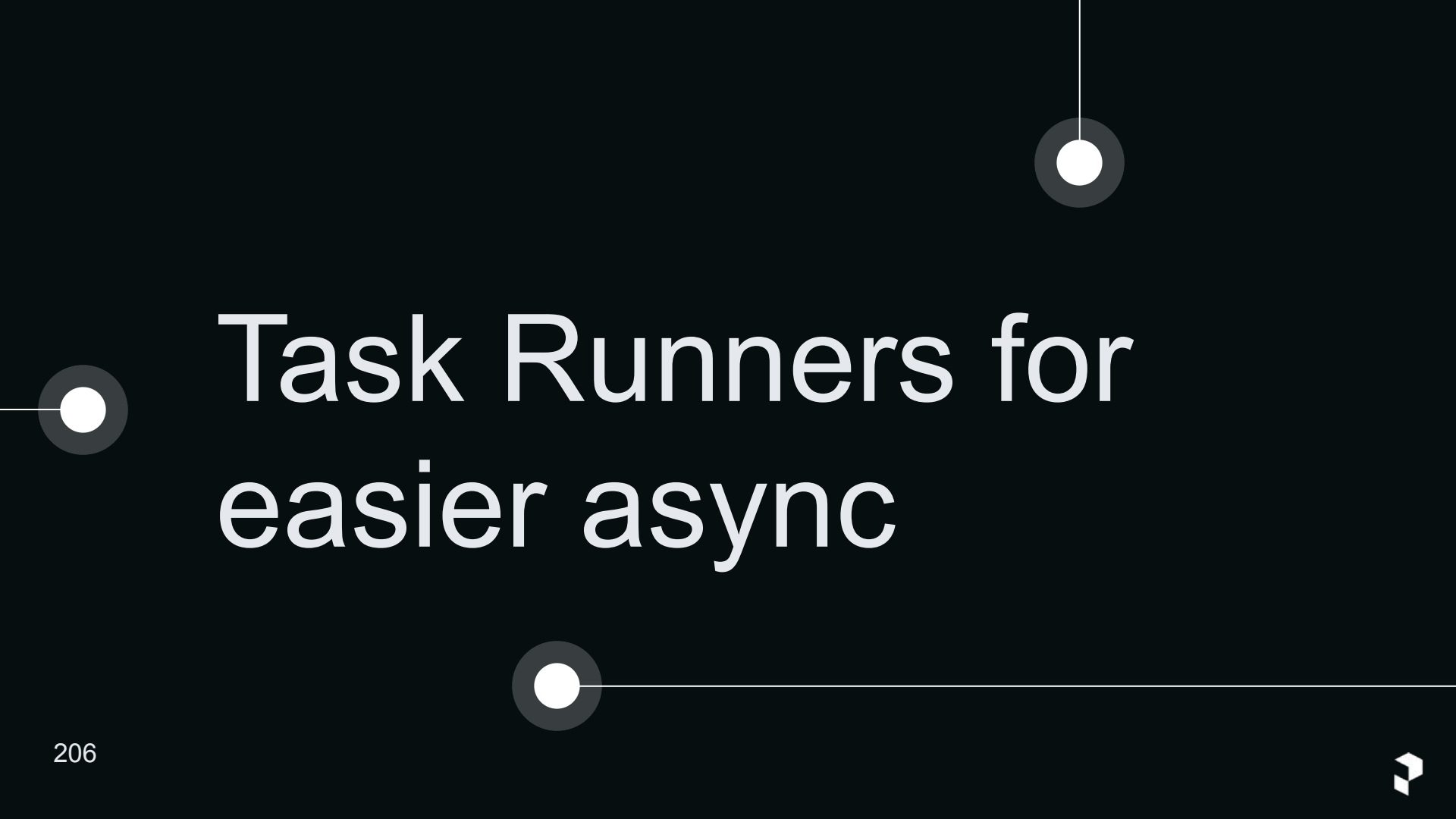
prefect shell serve

prefect shell watch for quick flow runs

prefect shell serve for deployment creation with a *serve* process

Docs: docs.prefect.io/latest/resources/cli-shell

Task Runners for easier async



Concurrency & Parallelism

- **Sequential:** run to completion before next run starts
- **Concurrency:** non-blocking, single-thread, interleaving
- **Parallelism:** multiple operations at the same time

Your Prefect code runs **sequentially** by default

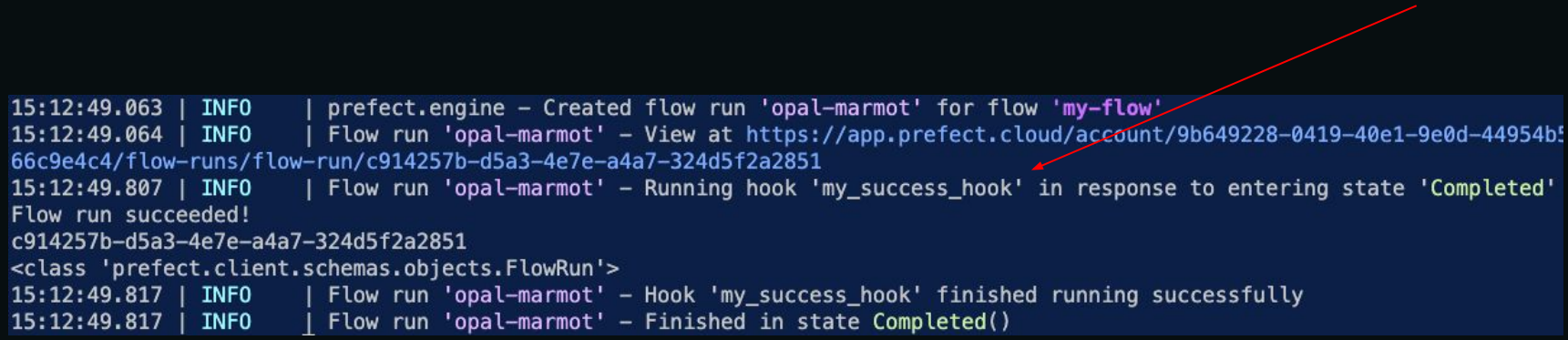
State change hooks

State change hooks

Execute code in response to flow run or task run state changes

```python
from prefect import flow


def my_success_hook(flow, flow_run, state):
    print(f"Flow run {flow_run.id} succeeded!")


@flow(on_completion=[my_success_hook])
def my_flow():
    return 42


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


State change hooks



State change hooks

Type	Flow	Task	Description
`on_completion`	✓	✓	Executes when a flow or task run enters a `Completed` state.
`on_failure`	✓	✓	Executes when a flow or task run enters a `Failed` state.
`on_cancellation`	✓	-	Executes when a flow run enters a `Cancelling` state.
`on_crashed`	✓	-	Executes when a flow run enters a `Crashed` state.

State change hooks

- Can create a custom event in Python code to fire in response to a state-change hook
- Alternative to an automation
- Generally, automations are preferred

ControlFlow

ControlFlow

- Python framework for building agentic AI workflows
- OSS
- Uses Prefect under the hood - Observation and orchestration benefits! 🎉

controlflow.ai

ControlFlow

- Default model: OpenAI GPT-4o
- Other models optional, including self-trained

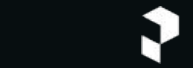
controlflow.ai/guides/configure-llms

- Can use LangChain tools
- Requires Prefect 3

ControlFlow example: summarize and analyze sentiment

controflow version

```
   ControlFlow version: 0.11.3
       Prefect version: 3.1.3
LangChain Core version: 0.3.9
        Python version: 3.12.6
              Platform: macOS-15.0.1-arm64-arm-64bit
```

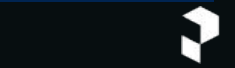



ControlFlow example: summarize and analyze sentiment

```python
import controlflow as cf
from langchain_community.tools import DuckDuckGoSearchRun


summarizer = cf.Agent(
    name="Headline Summarizer",
    description="An AI agent that fetches and summarizes current events",
    tools=[DuckDuckGoSearchRun()],
)

extractor = cf.Agent(
    name="Entity Extractor",
    description="An AI agent that does named entity recognition",
)
```


ControlFlow example: summarize and analyze sentiment

```python
@cf.flow
def get_headlines():

    summarizer_task = cf.Task(
        "Retrieve and summarize today's two top business headlines",
        agents=[summarizer],
        result_type=list[str],
    )

    extractor_task = cf.Task(
        "Extract any fortune 500 companies mentioned in the headlines and whether the sentiment is positive, neutral, or negative",
        agents=[extractor],
        depends_on=[summarizer_task],
    )

    summarizer_task_output = summarizer_task.run()
    extractor_task_output = extractor_task.run()

    return summarizer_task_output, extractor_task_output


if __name__ == "__main__":
    headlines, entity_sentiment = get_headlines()
    print(headlines, entity_sentiment)
```

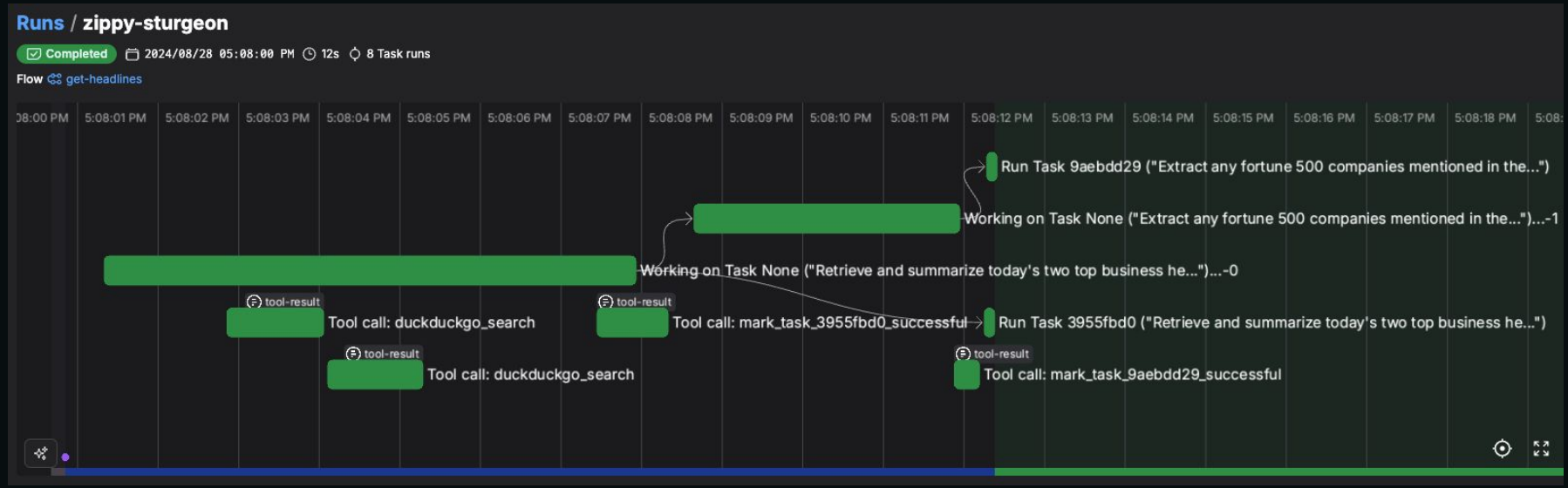
ControlFlow example: summarize and analyze sentiment

Extraction and Sentiment Analysis

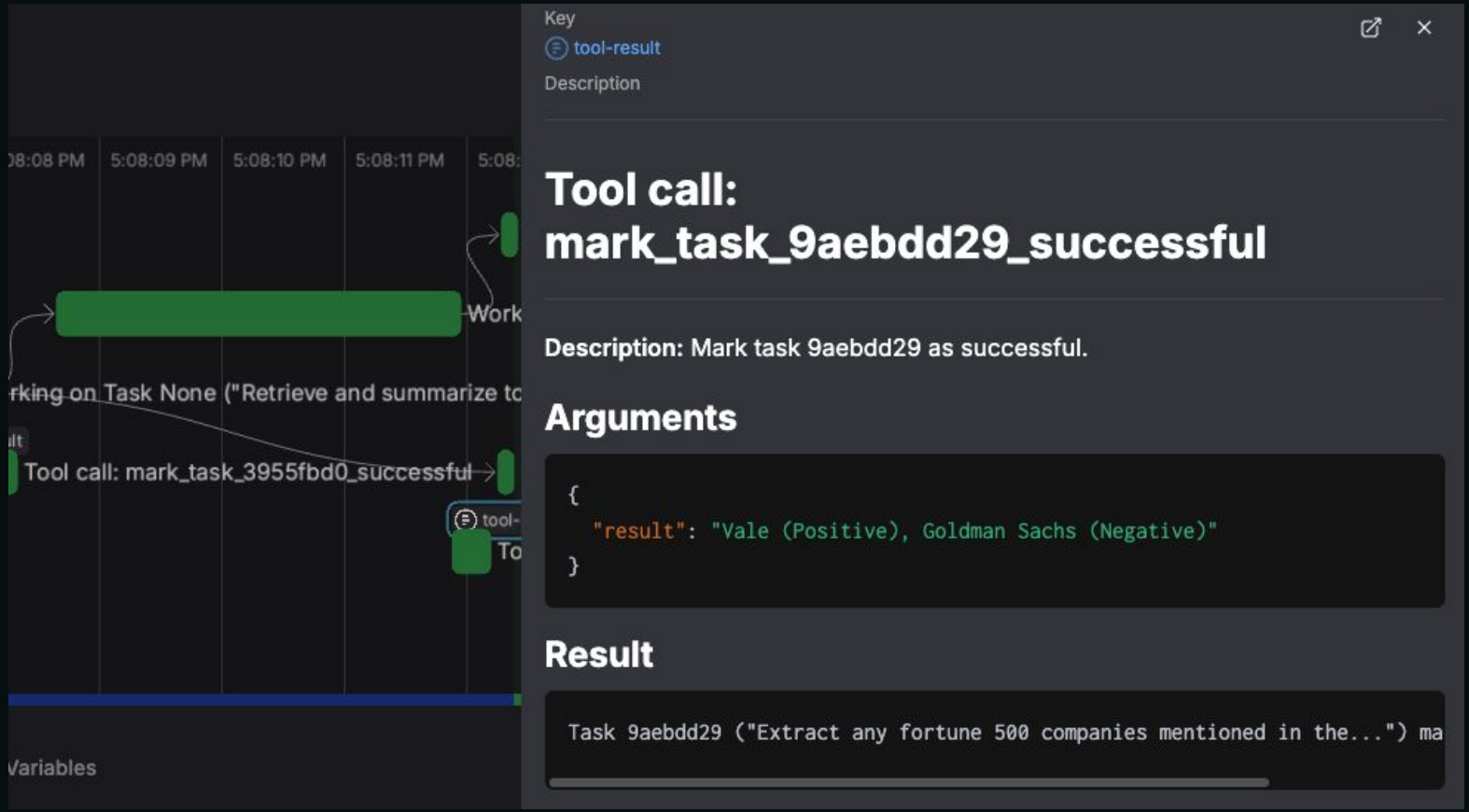
1 **Vale:**
   - **Company:** Vale
   - **Sentiment:** Positive (due to stock price increase and positive market reception)

2 **Goldman Sachs:**
   - **Company:** Goldman Sachs
   - **Sentiment:** Negative (due to license revocation and failed sale)

ControlFlow example: summarize and analyze sentiment



ControlFlow example: summarize and analyze sentiment



PACC

Prefect Associate
Certification Course